A New Look at the System, Algorithm and Theory Foundations of Distributed Machine Learning

[1]Eric P. Xing and [2]Qirong Ho

[1]Carnegie Mellon University
[2]Institute for Infocomm Research, A*STAR

Acknowledgements:
Wei Dai, Jin Kyu Kim, Abhimanu Kumar, Seunghak Lee, Jinliang Wei, Pengtao Xie, Xun Zheng
Yaoliang Yu, James Cipar, Henggang Cui,
and, Phil Gibbons, Greg Ganger, Garth Gibson

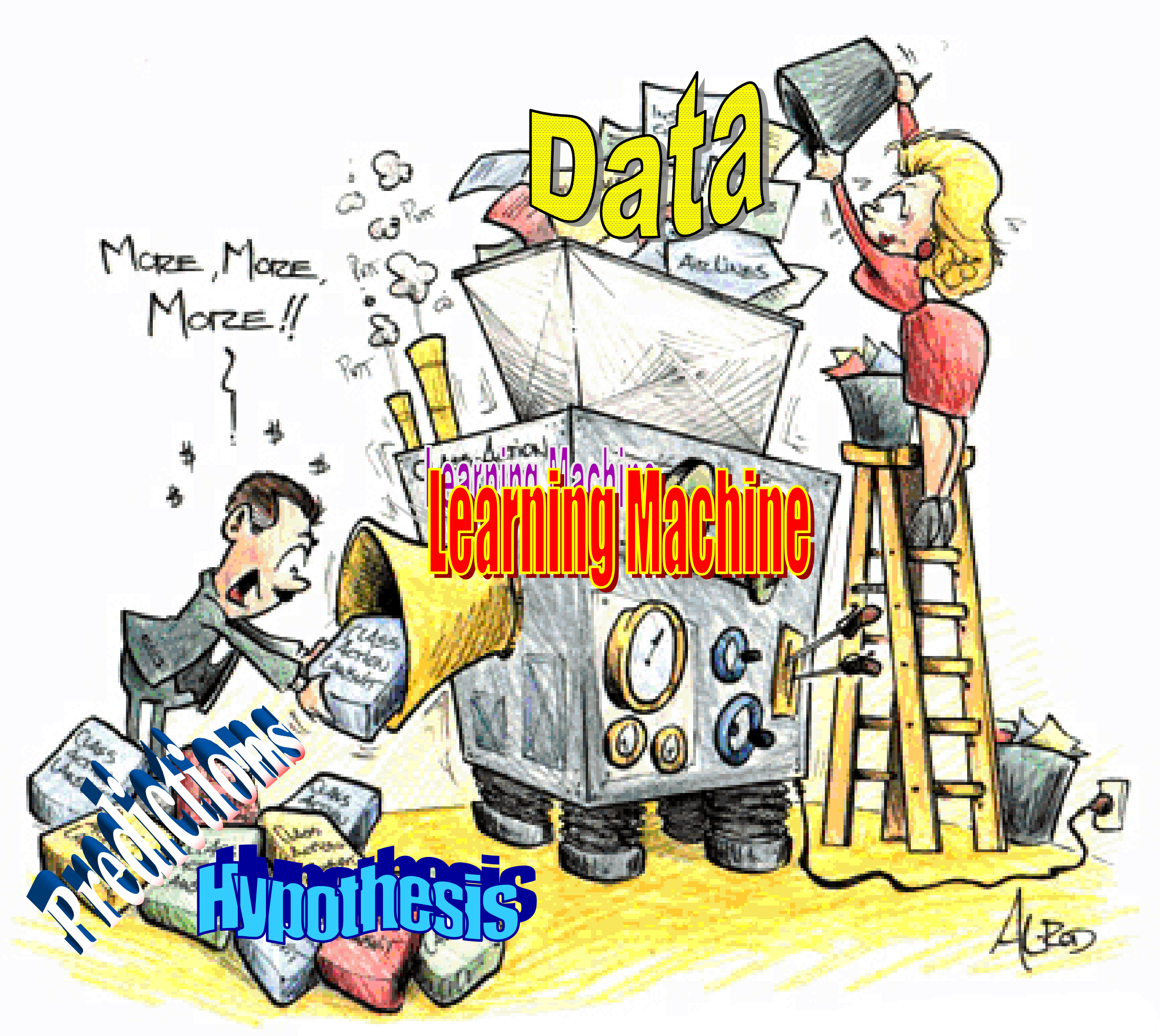
Trees Falling in the Forest

"If a tree falls in a forest and no one is around to hear it, does it make a sound?" --- George Berkeley

Data ≠ Knowledge

- Nobody knows what's in data unless it has been processed and analyzed
  - Need a scalable way to automatically search, digest, index, and understand contents

Machine Learning



Massive Data

facebook

1B+ USERS
30+ PETABYTES

WIKIPEDIA
The Free Encyclopedia

32 million pages

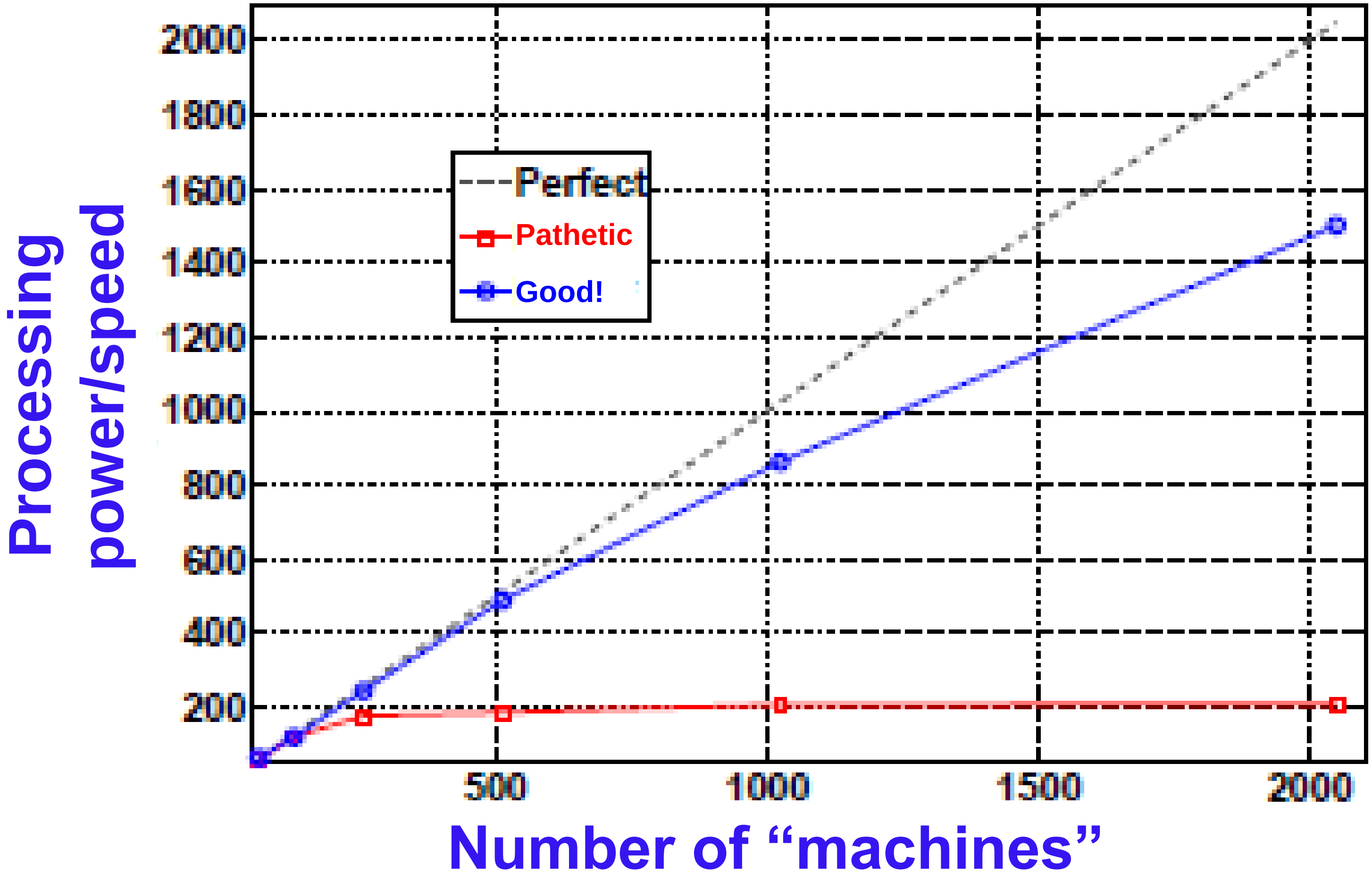
YouTube

100+ hours video uploaded every minute

twitter

645 million users
500 million tweets / day

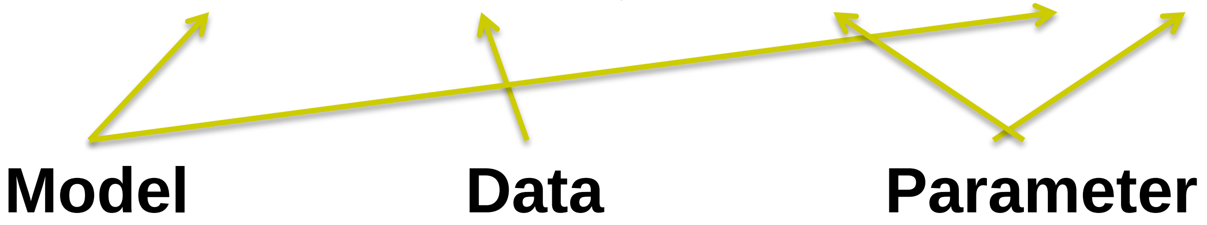
The Scalability Challenge



© Eric Xing @ CMU, 2015

An ML Program

$$\arg\max_{\vec{\theta}} \equiv \mathcal{L}(\{\mathbf{x}_i, \mathbf{y}i\}_{i=1}^{N} \;;\; \vec{\theta}) + \Omega(\vec{\theta})$$

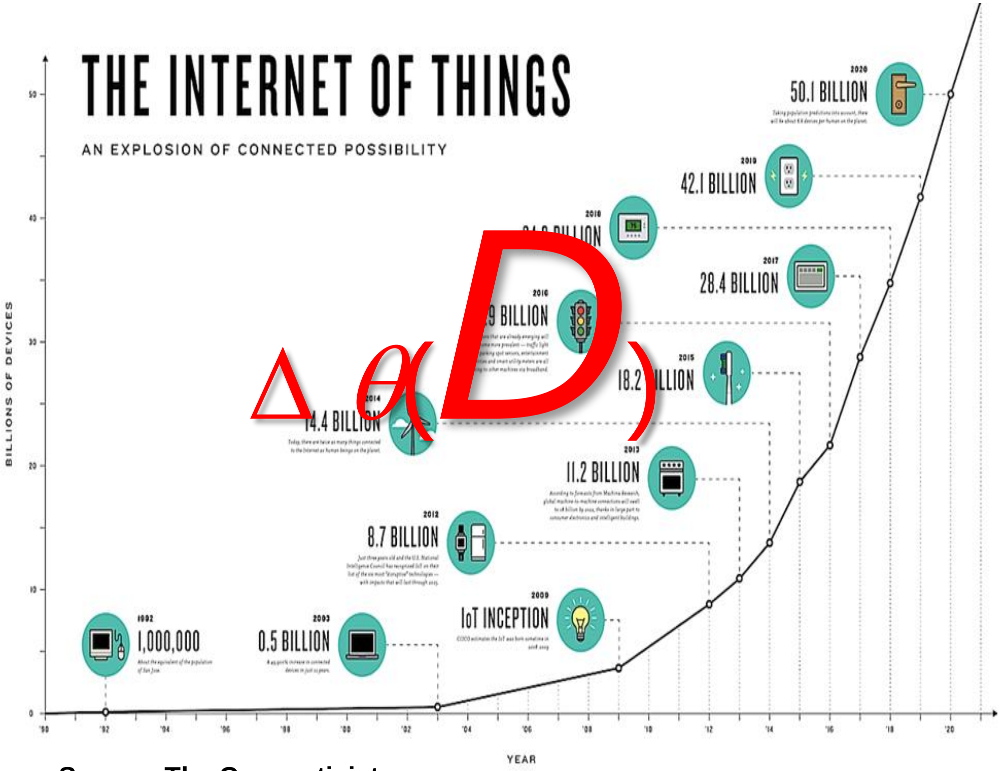


Solved by an iterative convergent algorithm

```
for (t = 1 to T) {
    doThings()
        $\vec{\theta}^{t+1} = g(\vec{\theta}^{t}, \underline{\Delta_f \vec{\theta}(\mathcal{D})})$
    doOtherThings()
}
```

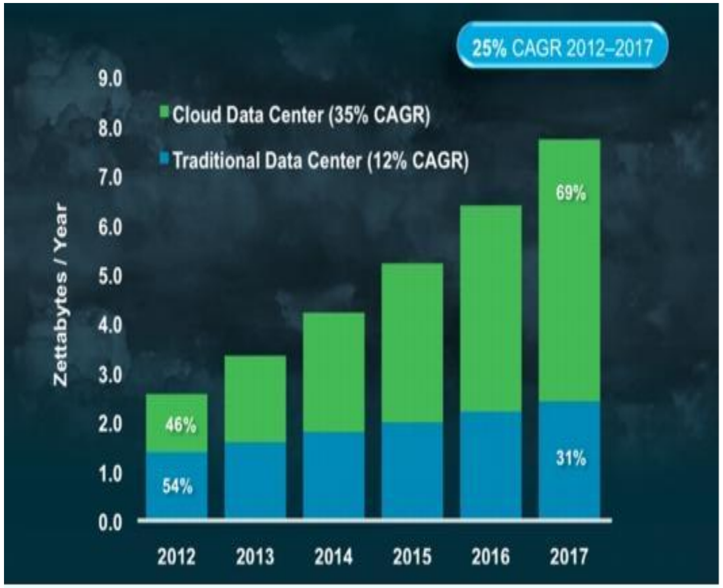
This computation needs to be scaled up !

Challenge 1 –
Massive Data Scale



$\Delta_{\theta}(D)$

Source: The Connectivist

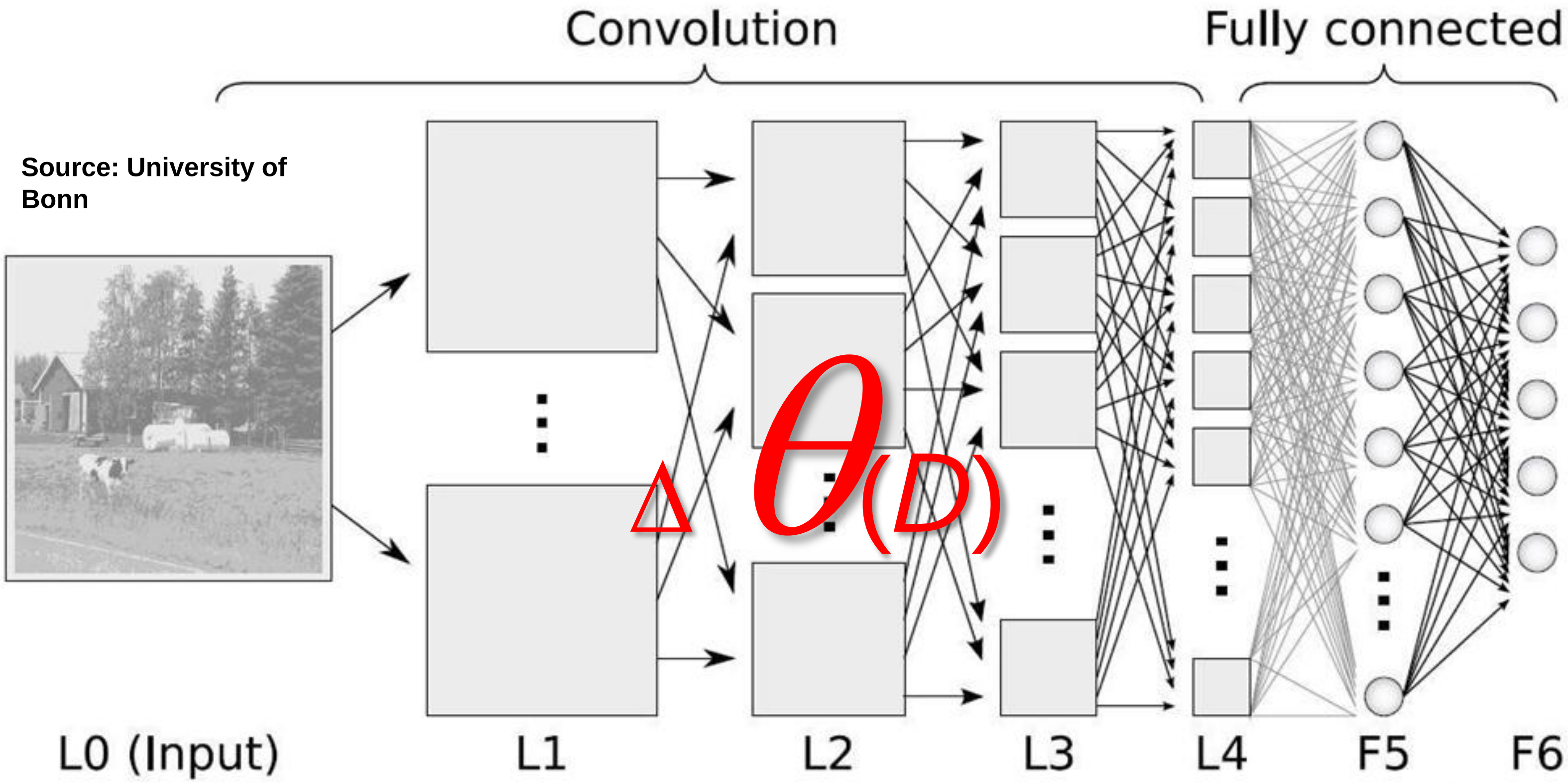


Source: Cisco Global Cloud Index

Familiar problem: data from 50B devices, data centers won't fit into memory of single machine

Challenge 2 – Gigantic Model Size

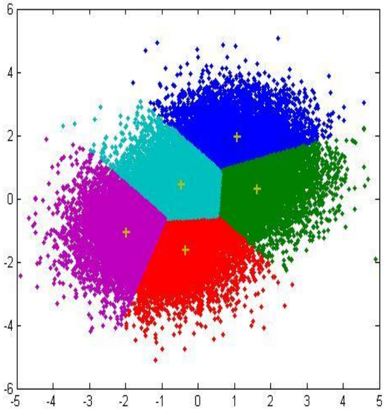
Source: University of Bonn



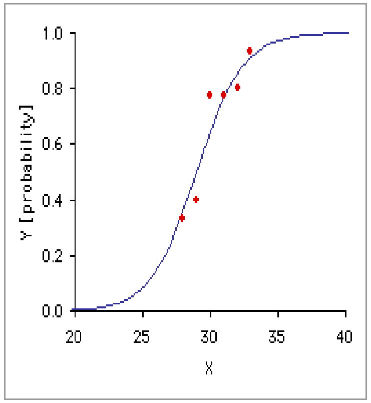
Maybe Big Data needs Big Models to extract understanding?
But models with >1 trillion params also won't fit!

Challenge 3 – Inadequate support for newer methods

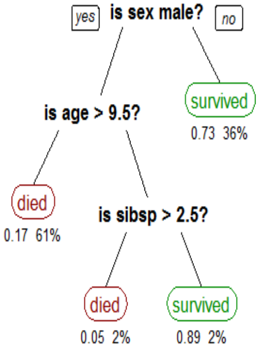
Classic algorithms used for decades



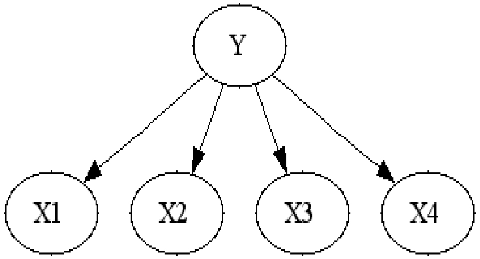
K-means



Logistic regression

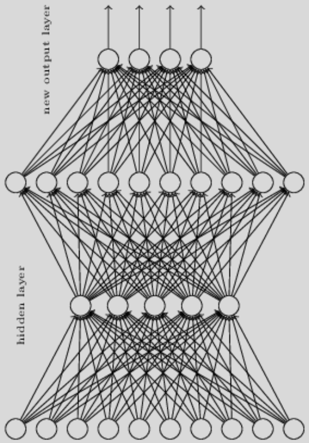


Decision trees

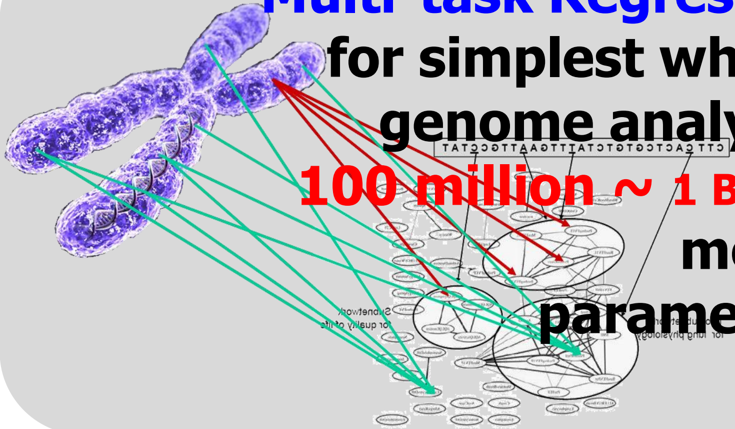


Naive Bayes

Growing Need for Big and Contemporary ML Programs



Google Brain
Deep Learning
for images:
1~10 Billion
model parameters



Multi-task Regression
for simplest whole-genome analysis:
100 million ~ 1 Billion
model parameters

The New York Times

"All the News That's Fit to Print"

VOL. CLXII . . . No. 56,024 NEW YORK, TUESDAY, JANUARY 22, 2013

OBAMA OFFERS LIBERAL VISION: 'WE MUS

Speech Gives Climate Goals Center Stage

Inaugural of Civil — Safet

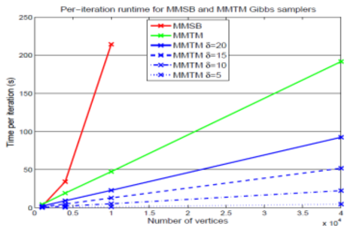
Topic Models
for news article analysis:
Up to 1 Trillion
model parameters

Collaborative filtering
for Video recommendation:
1~10 Billion
model parameters

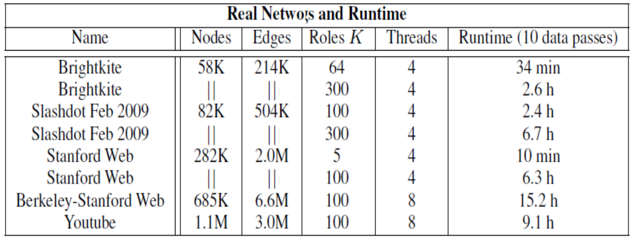
NETFLIX

The Need for Distributed ML

Say we want to analyze 10K roles in a 100M-node network, using a mixed membership model?



- We had developed
  - a highly cost-effective model (MMTM [Ho et al., 2012]),
  - two generations of highly efficient algorithms (δ-subsampling Gibbs [Ho et al., 2012], SVI [Yin et al., 2013])
  - and highly specialized implementations



Real Netwoŗs and Runtime

Name	Nodes	Edges	Roles K	Threads	Runtime (10 data passes)
Brightkite	58K	214K	64	4	34 min
Brightkite	||	||	300	4	2.6 h
Slashdot Feb 2009	82K	504K	100	4	2.4 h
Slashdot Feb 2009	||	||	300	4	6.7 h
Stanford Web	282K	2.0M	5	4	10 min
Stanford Web	||	||	100	4	6.3 h
Berkeley-Stanford Web	685K	6.6M	100	8	15.2 h
Youtube	1.1M	3.0M	100	8	9.1 h

→ State-of-the-art results: 1M node networks with 100 roles in a few hours, on just one machine, 2-3 order's of magnitudes speed-up

- But when we tried to do 10K roles in a 100M-node network:
  - Memory: 100M * 10K = 1 trillion latent states = 4TB of RAM
  - Computation: 10K+ hrs on one machine, i.e. yrs!
  - Attempt with Hadoop failed while in FB (see later) !!!

Many Open Questions:

- When is *Big Data* useful?
- Are *Big Models* useful?

-- Both positive and negative answers exist …

- Inference algorithms, or inference systems?
- Theoretical guarantees, or empirical performance?

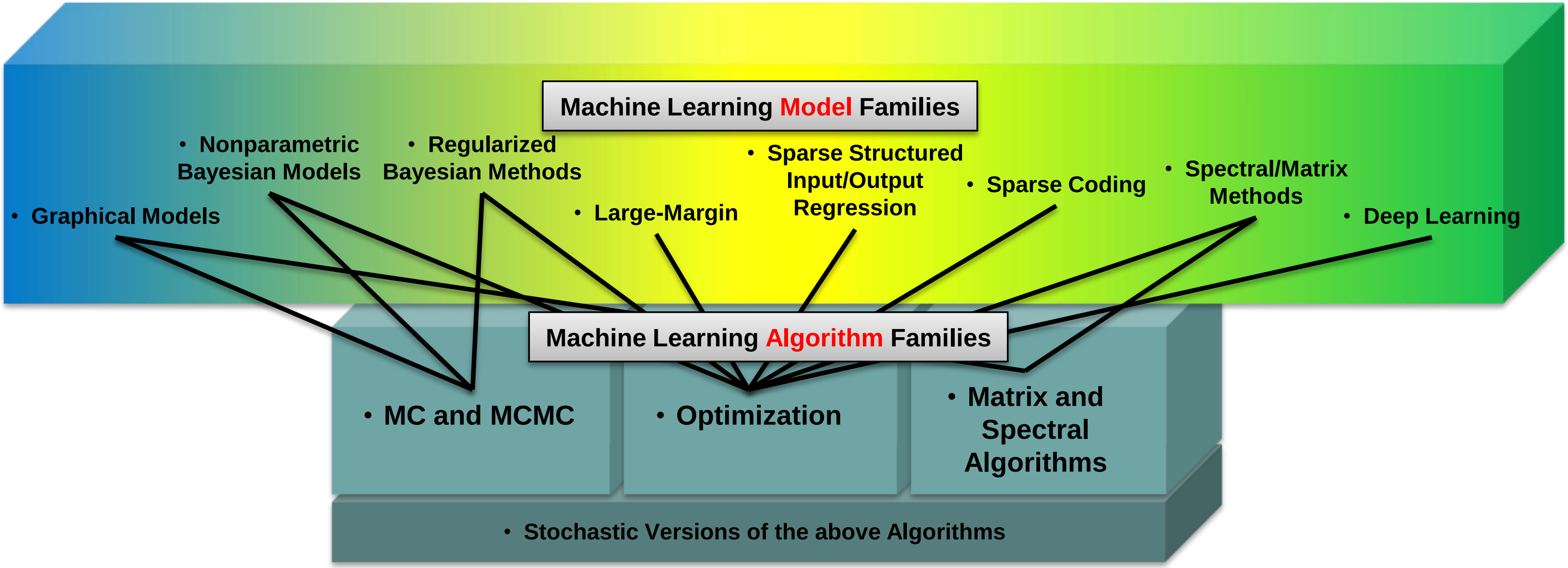
Current Solutions to Scalable ML

- Implementations of specific ML algorithms
  - YahooLDA, Vowpal Wabbit, Caffe, Torch, …
  - Provide a finely-tuned implementation of one (or a few) ML algorithms
- Platforms for general-purpose ML
  - Hadoop, Spark, GraphLab, Petuum, …
  - Allow others to write new ML programs
- Why this tutorial?
  - At first glance, ML problems seem radically different
  - We introduce a formal picture of ML to "bring order to the zoo"
  - We expose ML mathematical properties to be explored and later exploited
  - We note that many ML problems can be solved by a few "workhorse" algorithms
  - We explain how to design systems around these insights – thus achieving scalability, with both speed and solution quality guarantees
  - We provide theoretical guarantees for the system designs, and lay out roadmap for further analysis

Overview of Modern ML

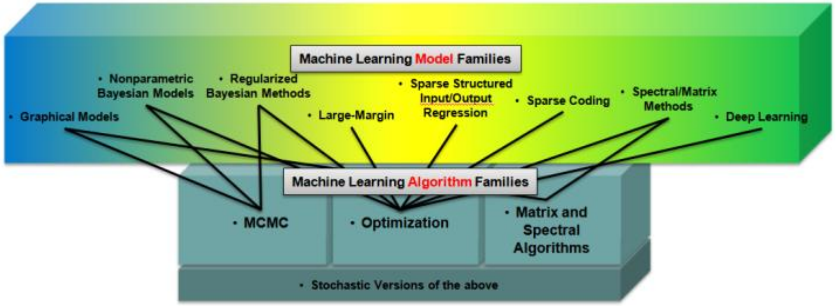
A “Classification” of ML Models and Tools

- An ML program consists of:
  - A mathematical “ML model” (from one of **many** families)…
  - … which is solved by an “ML algorithm” (from one of a **few** types)

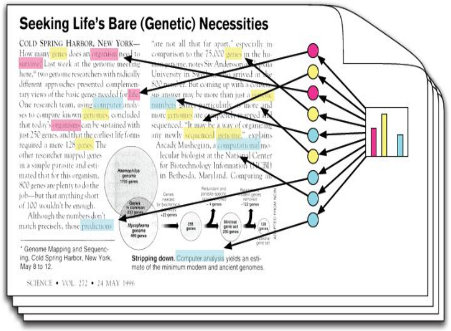

Machine Learning Model Families
• Nonparametric Bayesian Models
• Regularized Bayesian Methods
• Graphical Models
• Large-Margin
• Sparse Structured Input/Output Regression
• Sparse Coding
• Spectral/Matrix Methods
• Deep Learning
Machine Learning Algorithm Families
• MC and MCMC
• Optimization
• Matrix and Spectral Algorithms
• Stochastic Versions of the above Algorithms

A “Classification” of ML Models and Tools

- We can view ML programs as either
  - Probabilistic programs
  - Optimization programs


Machine Learning Model Families
• Nonparametric Bayesian Models
• Regularized Bayesian Methods
• Graphical Models
• Large-Margin
• Sparse Structured Input/Output Regression
• Sparse Coding
• Spectral/Matrix Methods
• Deep Learning
Machine Learning Algorithm Families
• MCMC
• Optimization
• Matrix and Spectral Algorithms
• Stochastic Versions of the above

Probabilistic Programs



Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough. Although the numbers don't match precisely, those predictions “are not all that far apart,” especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. “It may be a way of organizing any newly sequenced genome,” explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

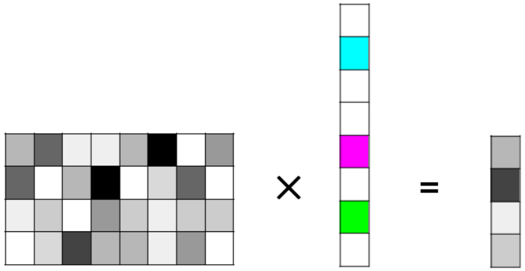
Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

SCIENCE • VOL. 272 • 24 MAY 1996

Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
256 genes
Redundant and parasite-specific genes removed −4 genes
Minimal gene set 250 genes
Replaced modern genes removed −122 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

$$\sum_{i=1}^{N}\sum_{j=1}^{N_i}\ln \mathbb{P}_{Categorical}(x_{ij} \mid z_{ij}, B) + \sum_{i=1}^{N}\sum_{j=1}^{N_i}\ln \mathbb{P}_{Categorical}(z_{ij} \mid \delta_i)$$

Optimization Programs



$$\sum_{i=1}^{N}\|y_i - X_i\beta\|_2^2 + \lambda\sum_{j=1}^{D}|\beta_j|$$

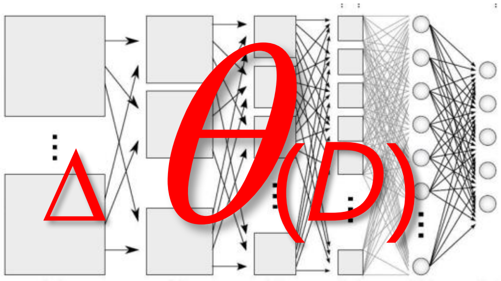
Key building blocks of an ML program

- ML program: $f(\theta,D) = L(\theta,D) + r(\theta)$
- Objective or Loss function: $L(\theta,D)$
  - θ = model, D = data
  - Common examples:
    - Least squares difference between predicted value and data
    - Log-likelihood of data
- Regularization / Prior / Structural Knowledge: $r(\theta)$
  - Common examples:
    - L2 regularization on θ to prevent overfitting
    - L1 regularization on θ to obtain sparse solution
    - (log of) Gaussian or Laplace priors over θ
    - (log of) Dirichlet prior over θ for smoothing
- Algorithm to solve for model given the data (cont' next slide)

Iterative-convergent view of ML

$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

New Model = Old Model + Update(Data)



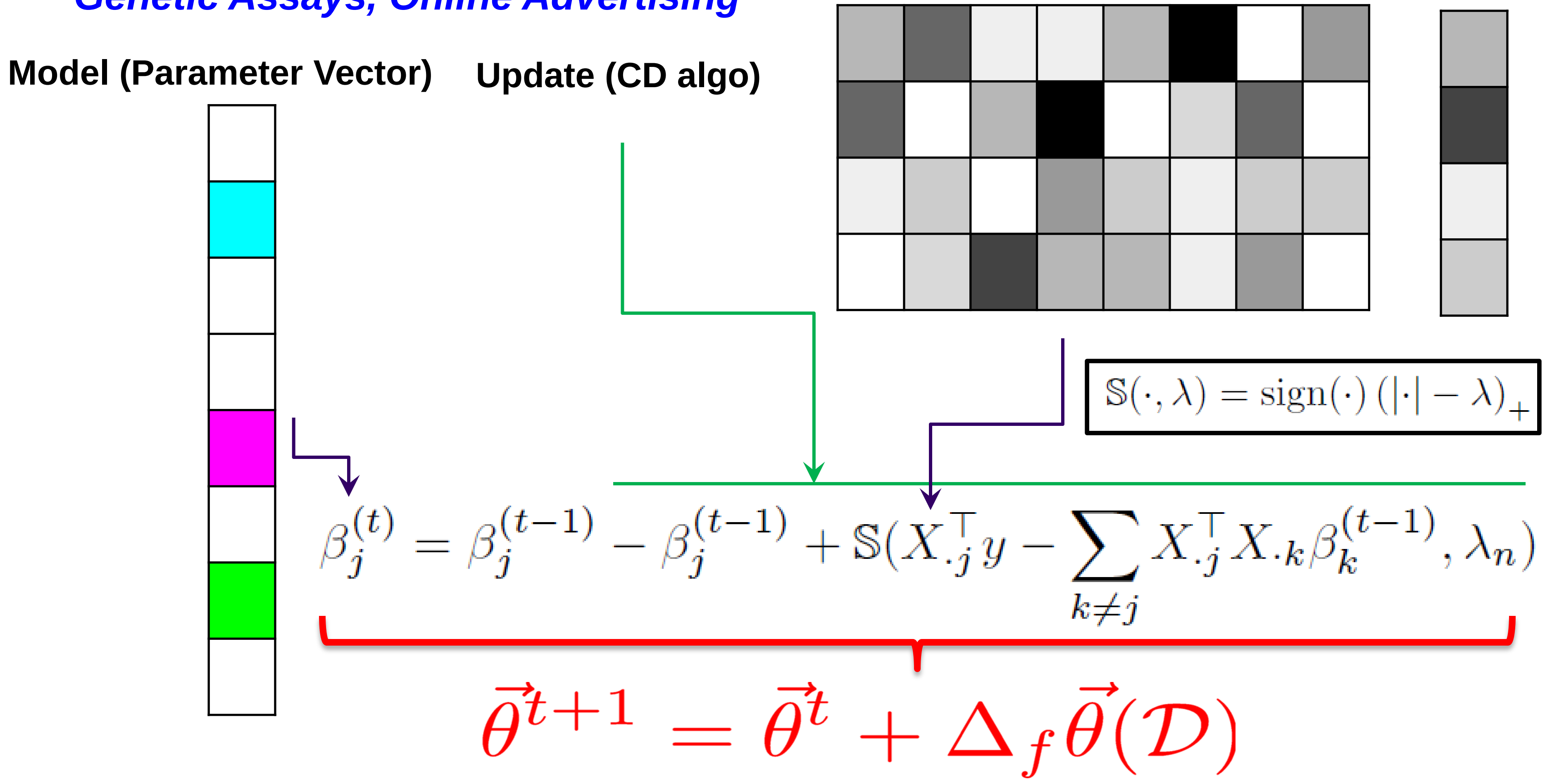
- ML models solved via iterative-convergent ML algorithms
  - Iterative-convergent algorithms repeat until θ is stationary. Examples:
    - Probabilistic programs: MC, MCMC, Variational Inference
    - Optimization programs: Stochastic Gradient Descent, ADMM, Proximal Methods, Coordinate Descent

Optimization Example: Lasso Regression

- Data, Model
  - D = {feature matrix X, response vector y}
  - θ = {parameter vector β)
- Objective L(θ,D)
  - Least-squares difference between y and Xβ: $\sum_{i=1}^{N} \|y_i - X_i\beta\|_2^2$
- Regularization r(θ)
  - L1 penalty on β to encourage sparsity: $\lambda \sum_{j=1}^{D} |\beta_j|$
  - λ is a tuning parameter
- Algorithms
  - Coordinate Descent
  - Stochastic Proximal Gradient Descent

Optimization Example: Lasso Regression

Applications:
Genetic Assays, Online Advertising



Model (Parameter Vector)

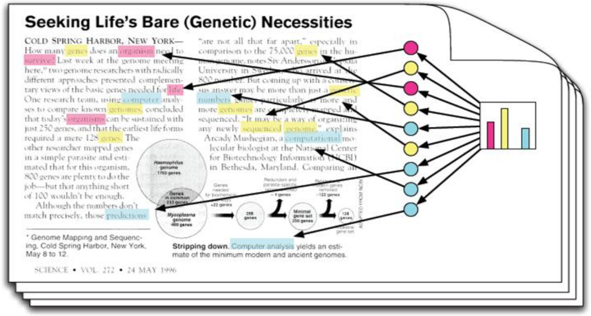
Update (CD algo)

Data (Feature + Response Matrices)

$$\boxed{\mathbb{S}(\cdot,\lambda) = \mathrm{sign}(\cdot)\left(|\cdot| - \lambda\right)_+}$$

$$\underbrace{\beta_j^{(t)} = \beta_j^{(t-1)} \overline{- \beta_j^{(t-1)} + \mathbb{S}(X_{\cdot j}^\top y - \sum_{k\neq j} X_{\cdot j}^\top X_{\cdot k}\beta_k^{(t-1)}, \lambda_n)}}_{\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})}$$

Probabilistic Example: Topic Models



Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK— How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
256 genes
Redundant and parasite-specific genes removed −4 genes
Minimal gene set 250 genes
Redundant modern genes removed −122 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

- Objective L(θ,D)
  - Log-likelihood of D = {document words x_{ij}} given unknown θ = {document word topic indicators z_{ij}, doc-topic distributions δ_i, topic-word distributions B_k}:

$$\sum_{i=1}^{N}\sum_{j=1}^{N_i} \ln \mathbb{P}_{Categorical}(x_{ij} \mid z_{ij}, B) + \sum_{i=1}^{N}\sum_{j=1}^{N_i} \ln \mathbb{P}_{Categorical}(z_{ij} \mid \delta_i)$$

- Prior r(θ)
  - Dirichlet prior on θ = {doc-topic, word-topic distributions}

$$\sum_{i=1}^{N} \ln \mathbb{P}_{Dirichlet}(\delta_i \mid \alpha) + \sum_{i=k}^{K} \ln \mathbb{P}_{Dirichlet}(B_k \mid \beta)$$

  - α, β are "hyperparameters" that control the Dirichet prior's strength

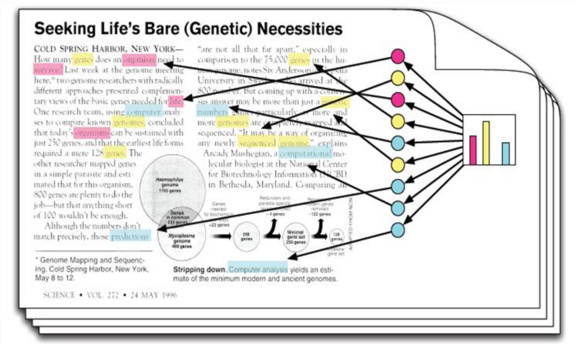
- Algorithm
  - Collapsed Gibbs Sampling

© Eric Xing @ CMU, 2015

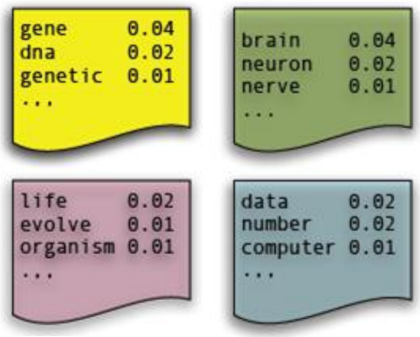
Probabilistic Example: Topic Models

Applications: Natural Language Processing, Information Retrieval

Data (Docs) = x_{ij}


Seeking Life's Bare (Genetic) Necessities
Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.
SCIENCE • VOL. 272 • 24 MAY 1996

Model (Topics) = B_k



gene	0.04
dna	0.02
genetic	0.01
...	

brain	0.04
neuron	0.02
nerve	0.01
...	

life	0.02
evolve	0.01
organism	0.01
...	

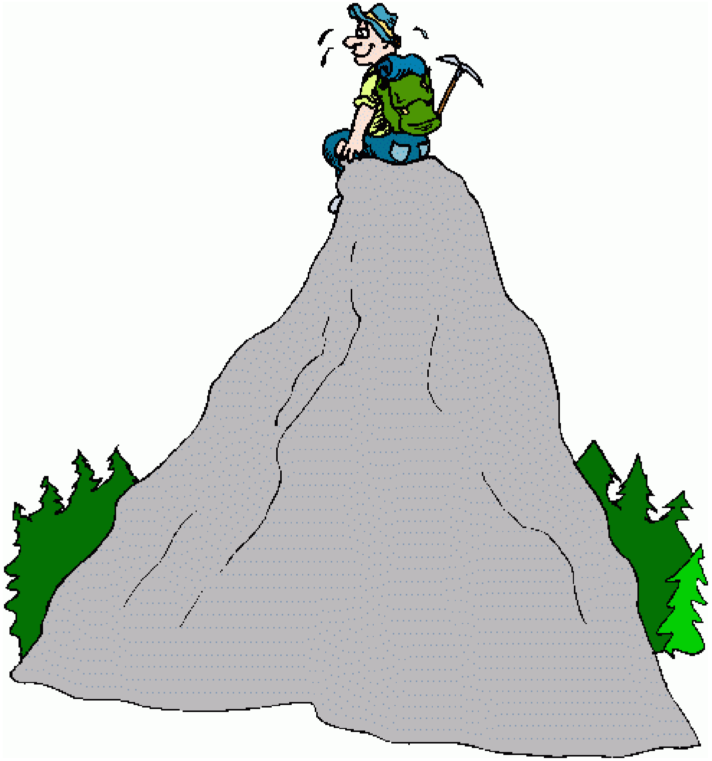
data	0.02
number	0.02
computer	0.01
...	

Update (Collapsed Gibbs sampling)

```
For each doc $i$, each token $j$:
    Set $k_{old} = z_{ij}$
    Gibbs sample new value of $z_{ij}$, according to $\mathbb{P}(z_{ij} \mid x_{ij}, \delta_i, B)$
    Set $k_{new} = z_{ij}$
    Perform updates to $B, \delta$:
```

$$\left.\begin{aligned} B_{k_{old},w_{ij}} &= B_{k_{old},w_{ij}} - 1 \\ B_{k_{new},w_{ij}} &= B_{k_{new},w_{ij}} + 1 \\ \delta_{i,k_{old}} &= \delta_{i,k_{old}} - 1 \\ \delta_{i,k_{new}} &= \delta_{i,k_{new}} + 1 \end{aligned}\right\} \quad \vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

ML Computation vs. Classical Computing Programs

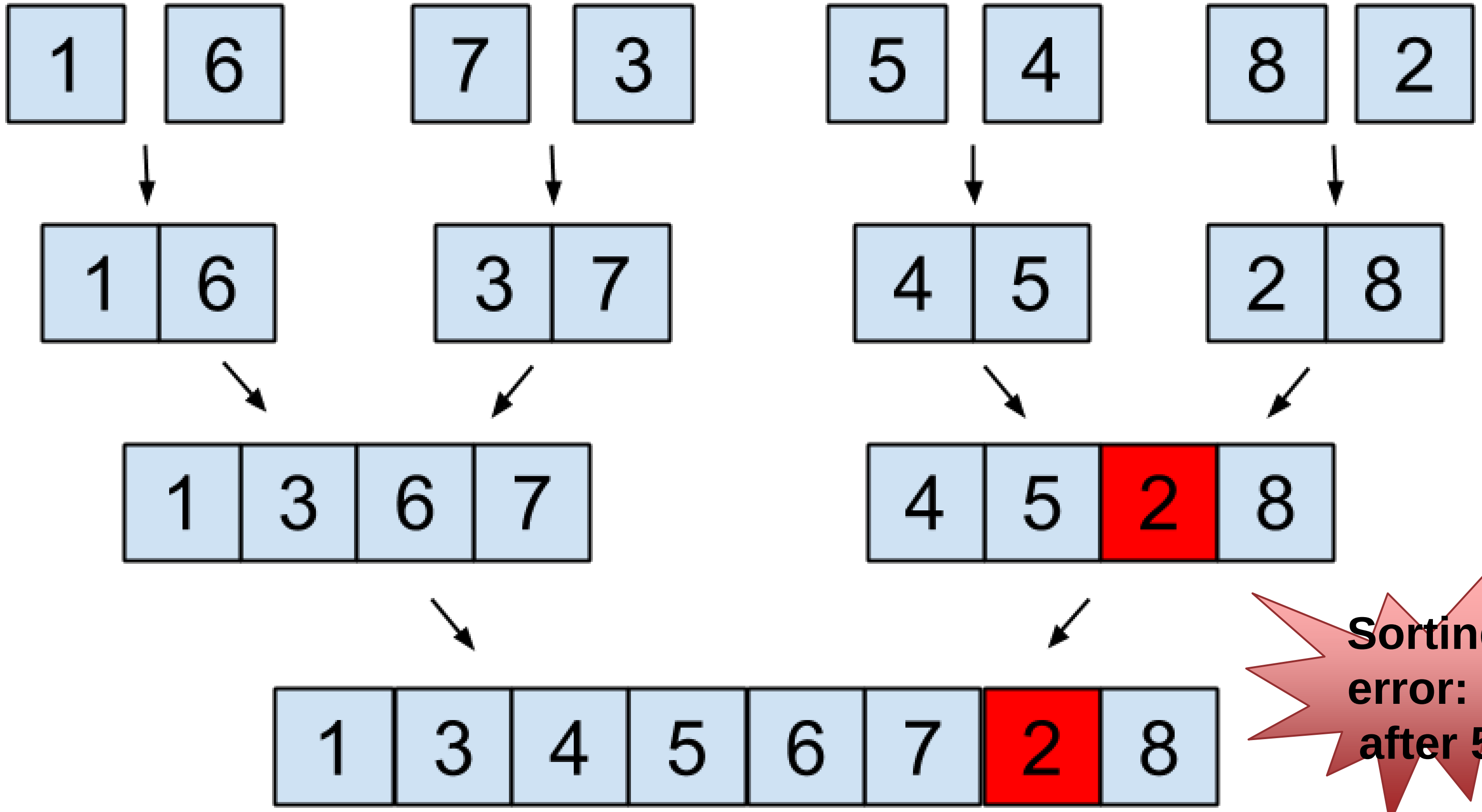


ML Program: optimization-centric and iterative convergent

Traditional Program: operation-centric and deterministic

Traditional Data Processing needs operational correctness …

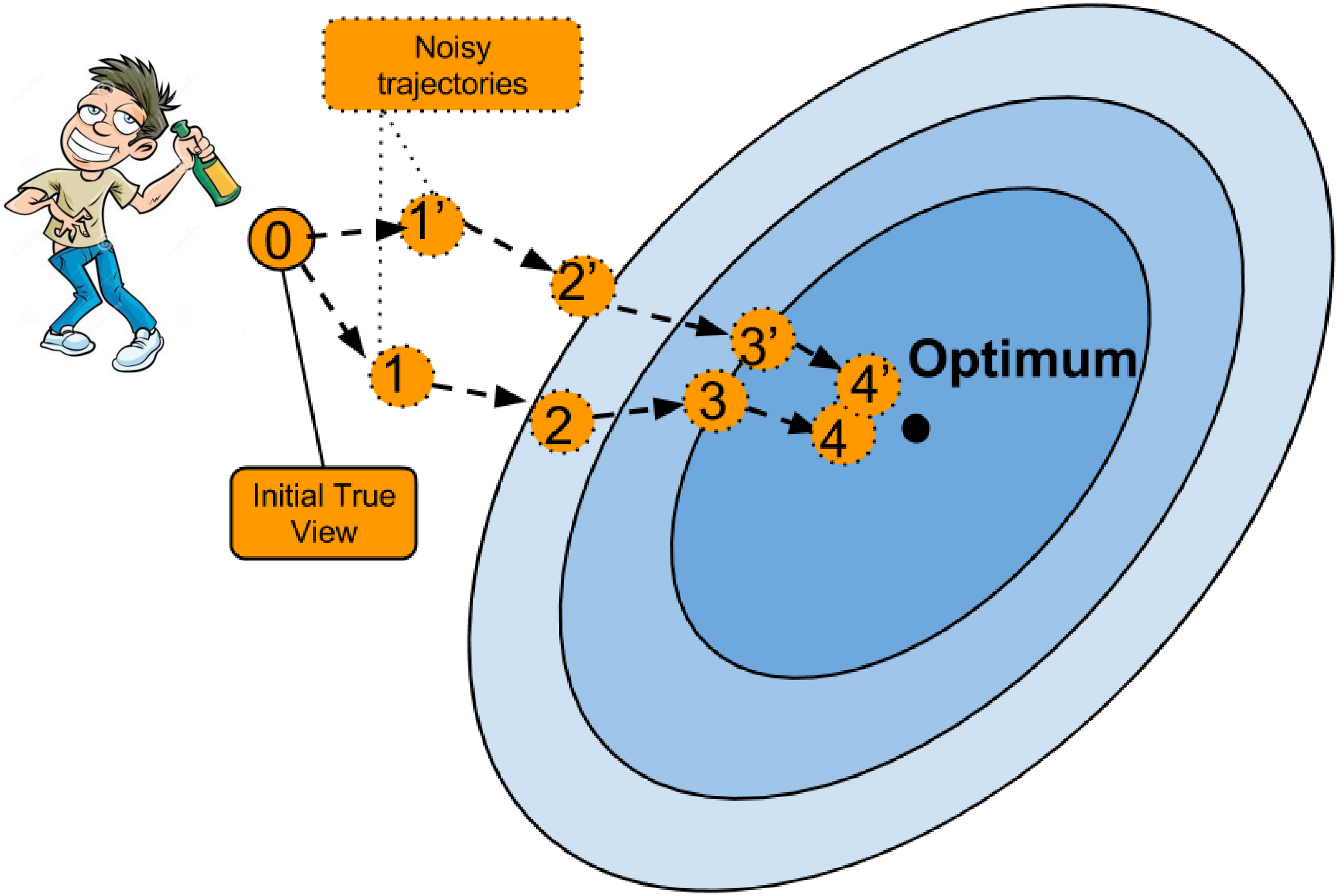
Example: Merge sort



Sorting error: 2 after 5

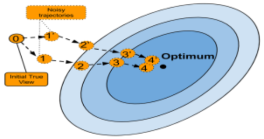
Error persists and is not corrected

… but ML Algorithms can Self-heal

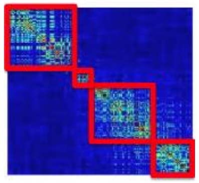


More Intrinsic Properties of ML Programs

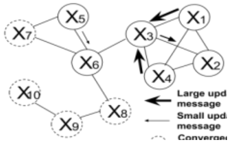
- ML is **optimization-centric**, and admits an **iterative convergent** algorithmic solution rather than a one-step closed form solution
  - **Error tolerance**: often robust against limited errors in intermediate calculations



  - **Dynamic structural dependency**: changing correlations between model parameters critical to efficient parallelization



  - **Non-uniform convergence**: parameters can converge in very different number of steps



- Whereas traditional programs are **transaction-centric**, thus only guaranteed by **atomic correctness** at every step

© Eric Xing @ CMU, 2015

Why come up with an ML classification?

- An ML classification helps to solve ML algorithm challenges systematically
  - No need to invent new algorithms for each new ML model or variant
  - Instead, re-use a smaller number of “workhorse” algorithms (engines) to solve entire classes of models
    - For each new ML model, determine which ML class it falls under
    - Then apply the most appropriate workhorse algorithm for that class

- Next tutorial section: Distributed ML Algorithms
  - We present a number of “workhorse” algorithms:
    - Basic form
    - Which units can be parallelized
    - What risks are incurred by parallelization (e.g. error or non-convergence)
    - Examples of scalable realizations (software)

Distributed ML Algorithms

An ML Program

$$\arg\max_{\vec{\theta}} \equiv \mathcal{L}(\{\mathbf{x}_i, \mathbf{y}i\}_{i=1}^{N} \; ; \; \vec{\theta}) + \Omega(\vec{\theta})$$

Model **Data** **Parameter**

Solved by an iterative convergent algorithm

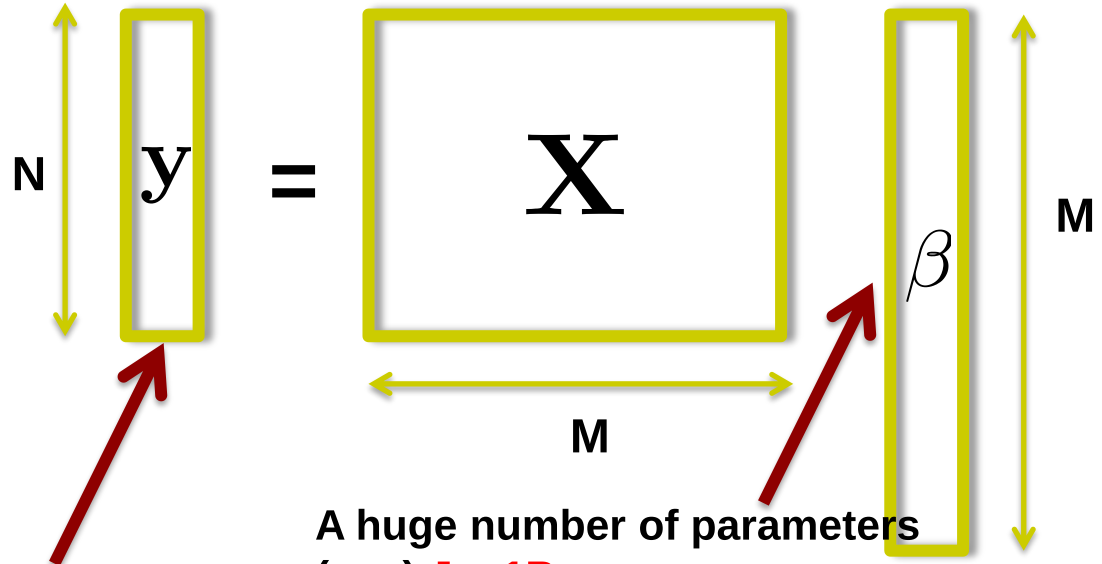
```
for (t = 1 to T) {
   doThings()
      $\vec{\theta}^{t+1} = g(\vec{\theta}^{t}, \; \underline{\Delta_f \vec{\theta}(\mathcal{D})})$
   doOtherThings()
}
```

This computation needs to be parallelized!

Challenge

- **Optimization programs:**

$$\Delta \leftarrow \sum_{i=1}^{N} \left[\frac{d}{d\theta_1}, \ldots, \frac{d}{d\theta_M} \right] f(\mathbf{x}_i, \mathbf{y}_i; \vec{\theta})$$



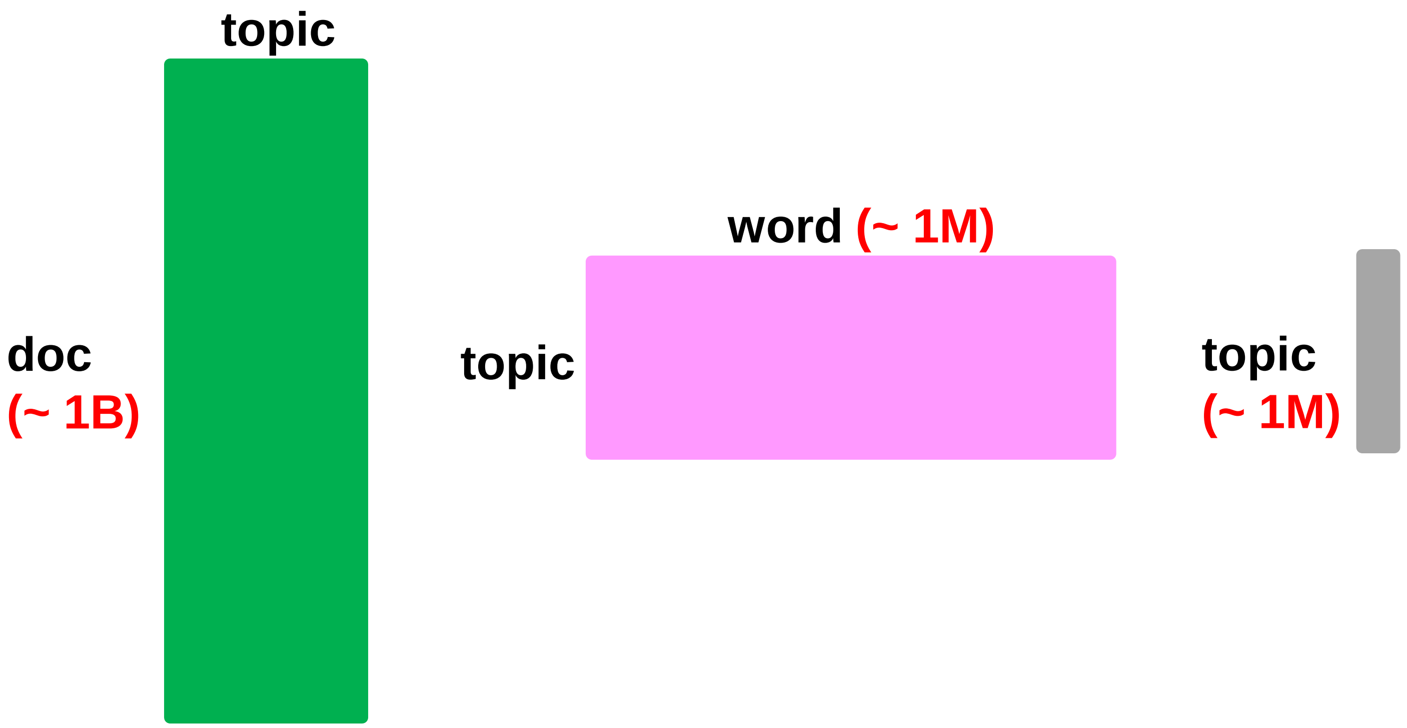
A huge volume of data (e.g.) N = 1B

A huge number of parameters (e.g.) J = 1B

Challenge

- **Probabilistic programs**

$$z_{ij} \sim p(z_{ij}=k|x_{ij},\delta_i,B) \propto (\boxed{\delta}_{ik}+\alpha_k)\cdot \frac{\beta_{x_{ij}}+\boxed{B}_{k,x_{ij}}}{V\beta+\boxed{\sum_{v=1}^{V} B_{k,v}}}$$

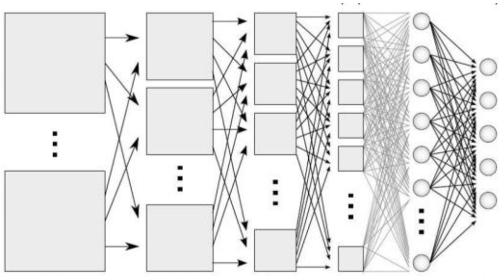


Parallelization Strategies

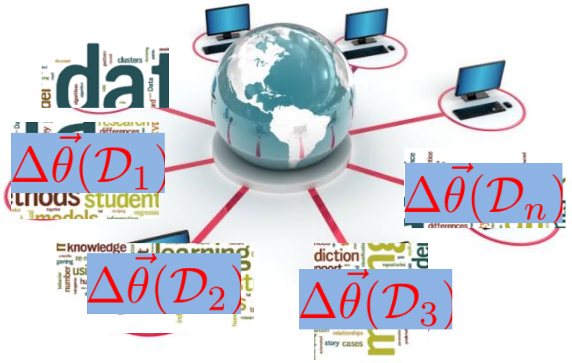
$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

New Model = Old Model + Update(Data)

$\Delta\theta(D)$



Data Parallel



$\Delta\vec{\theta}(\mathcal{D}_1)$

$\Delta\vec{\theta}(\mathcal{D}_n)$

$\Delta\vec{\theta}(\mathcal{D}_2)$

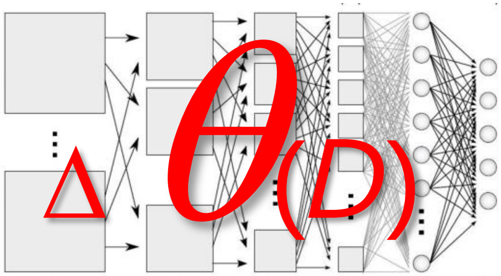
$\Delta\vec{\theta}(\mathcal{D}_3)$

$$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$$

Parallelization Strategies

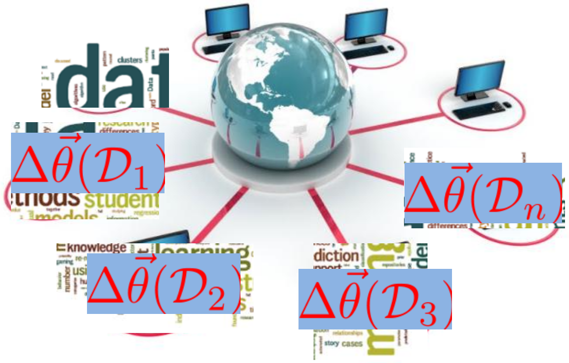
$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

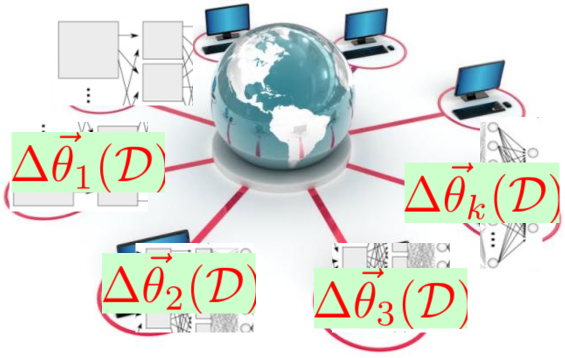
New Model = Old Model + Update(Data)



Data Parallel

Model Parallel


$\Delta\vec{\theta}(\mathcal{D}_1)$
$\Delta\vec{\theta}(\mathcal{D}_n)$
$\Delta\vec{\theta}(\mathcal{D}_2)$
$\Delta\vec{\theta}(\mathcal{D}_3)$


$\Delta\vec{\theta}_1(\mathcal{D})$
$\Delta\vec{\theta}_k(\mathcal{D})$
$\Delta\vec{\theta}_2(\mathcal{D})$
$\Delta\vec{\theta}_3(\mathcal{D})$

$$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$$

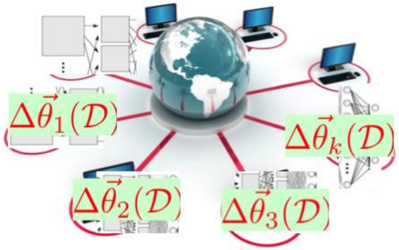
$$\vec{\theta} \equiv [\vec{\theta}_1^{\top}, \vec{\theta}_2^{\top}, \ldots, \vec{\theta}_k^{\top}\}^{\top}$$

Outline: Optimization & MCMC Algorithms

- Optimization Algorithms
  - Stochastic gradient descent
  - Coordinate descent
  - Proximal gradient methods
    - ISTA, FASTA, Smoothing proximal gradient
  - ADMM

$\Delta\vec{\theta}(\mathcal{D}_1)$
$\Delta\vec{\theta}(\mathcal{D}_n)$
$\Delta\vec{\theta}(\mathcal{D}_2)$
$\Delta\vec{\theta}(\mathcal{D}_3)$

- Markov Chain Monte Carlo Algorithms
  - Auxiliary Variable methods
  - Embarrassingly Parallel MCMC
  - Parallel Gibbs Sampling
    - Data parallel
    - Model parallel



$\Delta\vec{\theta}_1(\mathcal{D})$
$\Delta\vec{\theta}_k(\mathcal{D})$
$\Delta\vec{\theta}_2(\mathcal{D})$
$\Delta\vec{\theta}_3(\mathcal{D})$

Example Optimization Program: Sparse Linear Regression

$$\min_{\boldsymbol{\beta}} \underbrace{\frac{1}{2}\left\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\right\|_2^2}_{\textbf{Data fitting}} + \underbrace{\lambda\Omega(\boldsymbol{\beta})}_{\textbf{Regularization}}$$

Data fitting part:

- find **β** that fits into the data
- Squared loss, logistic loss, hinge loss, etc

Regularization part:

- induces sparsity in **β**.
- incorporates structured information into the model

Example Optimization Program: Sparse Linear Regression

$$\min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\Omega(\boldsymbol{\beta})$$

Examples of regularization $\Omega(\boldsymbol{\beta})$:

$$\left\{\ \Omega_{lasso}(\boldsymbol{\beta}) = \sum_{j=1}^{J} |\beta_j| \right.$$

Sparsity

$$\left\{\begin{array}{l} \Omega_{group}(\boldsymbol{\beta}) = \sum_{\mathbf{g}\in G} \|\boldsymbol{\beta}_{\mathbf{g}}\|_2 \\ \Omega_{tree}(\boldsymbol{\beta}) \\ \Omega_{overlap}(\boldsymbol{\beta}) \end{array}\right.$$

where $\|\boldsymbol{\beta}_{\mathbf{g}}\|_2 = \sum_{j\in\mathbf{g}} \sqrt{(\beta_j)^2}$

Structured sparsity
(sparsity + structured information)

Algorithm I: Stochastic Gradient Descent

- Consider an optimization problem:

$$\min_x \mathbb{E}\{f(x,d)\}$$

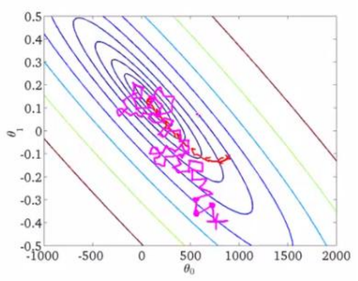
- Classical gradient descent: $x^{(t+1)} \leftarrow x^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_x f(x^{(t)}, d_i)$

- Stochastic gradient descent:
  - Pick a random sample d_i
  - Update parameters based on noisy approximation of the true gradient

$$x^{(t+1)} \leftarrow x^{(t)} - \gamma \nabla_x f(x^{(t)}, d_i)$$

Stochastic Gradient Descent

- **SGD converges almost surely to a global optimal for convex problems**



- **Traditional SGD compute gradients based on a single sample**

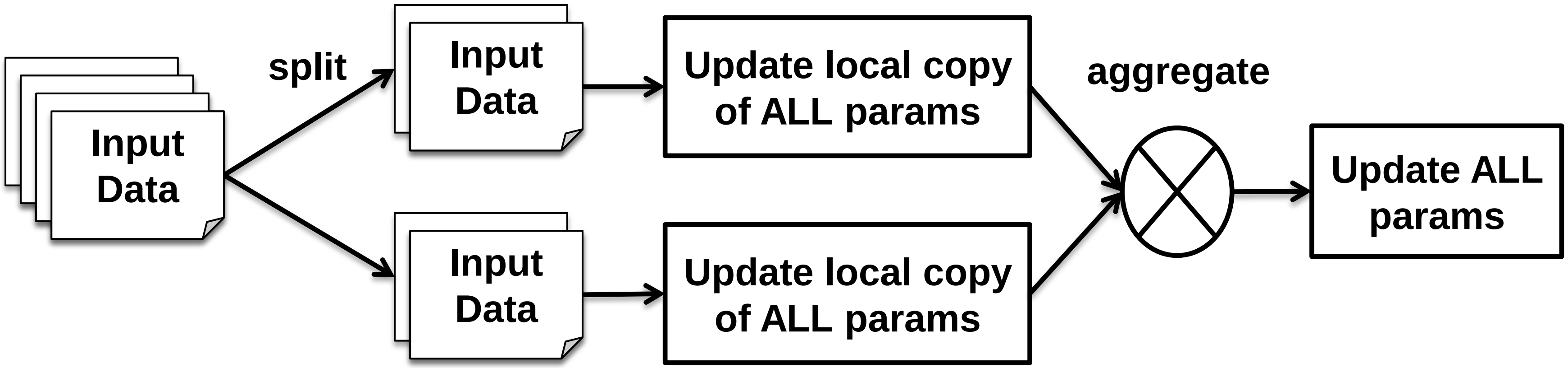
- **Mini-batch version computes gradients based on multiple samples**
  - **Reduce variance in gradients due to multiple samples**
  - **Multiple samples => represent as multiple vectors => use vector computation => speedup in computing gradients**

Parallel Stochastic Gradient Descent

$\Delta\vec{\theta}(\mathcal{D}_1)$
$\Delta\vec{\theta}(\mathcal{D}_n)$
$\Delta\vec{\theta}(\mathcal{D}_2)$
$\Delta\vec{\theta}(\mathcal{D}_3)$

- Parallel SGD: Partition data to different workers; all workers update full parameter vector

- Parallel SGD [Zinkevich et al., 2010]


Input Data
split
Input Data
Input Data
Update local copy of ALL params
Update local copy of ALL params
aggregate
Update ALL params

- PSGD runs SGD on local copy of params in each machine

Hogwild!: Lock-free approach to PSGD [Recht et al., 2011]

- Goal is to minimize a function in the form of

$$f(x) = \sum_{e \in E} f_e(x_e)$$

  - e denotes a small subset of parameter indices
  - x_e denotes parameter values indexed by x_e

- Key observation:
  - Cost functions of many ML problems can be represented by f(x)
  - In *SOME* ML problems, f(x) is sparse. In other words, |E| and n are large but f_e is applied only a small number of parameters in x

Hogwild!: Lock-free approach to PSGD [Recht et al., 2011]

- Example:
  - Sparse SVM

$$\min_{x} \sum_{\alpha \in E} \max(1 - y_\alpha x^T z_\alpha, 0) + \lambda \|x\|_2^2$$

    - z is input vector, and y is a label; (z,y) is an elements of E
    - Assume that z_α are sparse
  - Matrix Completion

$$\min_{W,H} \sum_{(u,v) \in E} (A_{uv} - W_u H_v^T)^2 + \lambda_1 \|W\|_F^2 + \lambda_2 \|H\|_F^2$$

    - Input A matrix is sparse
  - Graph cuts

$$\min_{x} \sum_{(u,v) \in E} w_{uv} \|x_u - x_v\|_1 \text{ subject to } x_v \in S_D, v = 1, \ldots, n$$

    - W is a sparse similarity matrix, encoding a graph

Hogwild! Algorithm [Recht et al., 2011]

- Hogwild! algorithm: iterate in parallel for each core
  - Sample e uniformly at random from E
  - Read current parameter x_e; evaluate gradient of function f_e
  - Sample uniformly at random a coordinate v from subset e
  - Perform SGD on coordinate v with small constant step size
- Advantages
  - Atomically update single coordinate, no mem-locking
  - Takes advantage of sparsity in ML problems
  - Near-linear speedup on various ML problems, on single machine
- Excellent on single machine, less ideal for distributed
  - Atomic update on multi-machine challenging to implement; inefficient and slow
  - Delay among machines requires explicit control… why? (see next slide)

The cost of uncontrolled delay – slower convergence [Dai et al. 2015]

- Theorem: Given lipschitz objective ft and step size ηt,

$$P\left[\frac{R\left[X\right]}{T} - \frac{1}{\sqrt{T}}\left(\sigma L^2 + \frac{F^2}{\sigma} + 2\sigma L^2 \epsilon_m\right) \geq \tau\right] \leq \exp\left\{\frac{-T\tau^2}{2\bar{\sigma}_T \boxed{\epsilon_v} + \frac{2}{3}\sigma L^2 (2s+1)P\tau}\right\}$$

  - where $R[X] \quad := \quad \sum_{t=1}^{T} f_t(\tilde{x}_t) \; - \; f(x^*)$
  - Where L is a lipschitz constant, and ε_m and ε_v are the mean and variance of the delay

- Intuition: distance between current estimate and optimal value decreases exponentially with more iterations
  - But high variance in the delay ε_v incurs exponential penalty!
- Distributed systems exhibit much higher delay variance, compared to single machine

The cost of uncontrolled delay – unstable convergence [Dai et al. 2015]

- Theorem: the variance in the parameter estimate is

$$\begin{aligned} \mathrm{Var}_{t+1} &= \mathrm{Var}_{t} - 2\eta_t cov(\boldsymbol{x}_t, \mathbb{E}^{\Delta_t}[\boldsymbol{g}_t]) + \mathcal{O}(\eta_t \xi_t) \\ &+ \mathcal{O}(\eta_t^2 \rho_t^2) + \boxed{\mathcal{O}^*_{\epsilon_t}} \end{aligned}$$

  - Where $cov(\boldsymbol{v}_1, \boldsymbol{v}_2) := \mathbb{E}[\boldsymbol{v}_1^T \boldsymbol{v}_2] - \mathbb{E}[\boldsymbol{v}_1^T]\mathbb{E}[\boldsymbol{v}_2]$
  - and $\mathcal{O}^*_{\epsilon_t}$ represents 5th order or higher terms, as a function of the delay ε_t

- Intuition: variance of the parameter estimate decreases near the optimum
  - But delay ε_t increases parameter variance => instability during convergence
- Distributed systems have much higher average delay, compared to single machine

Parallel SGD with Key-Value Stores

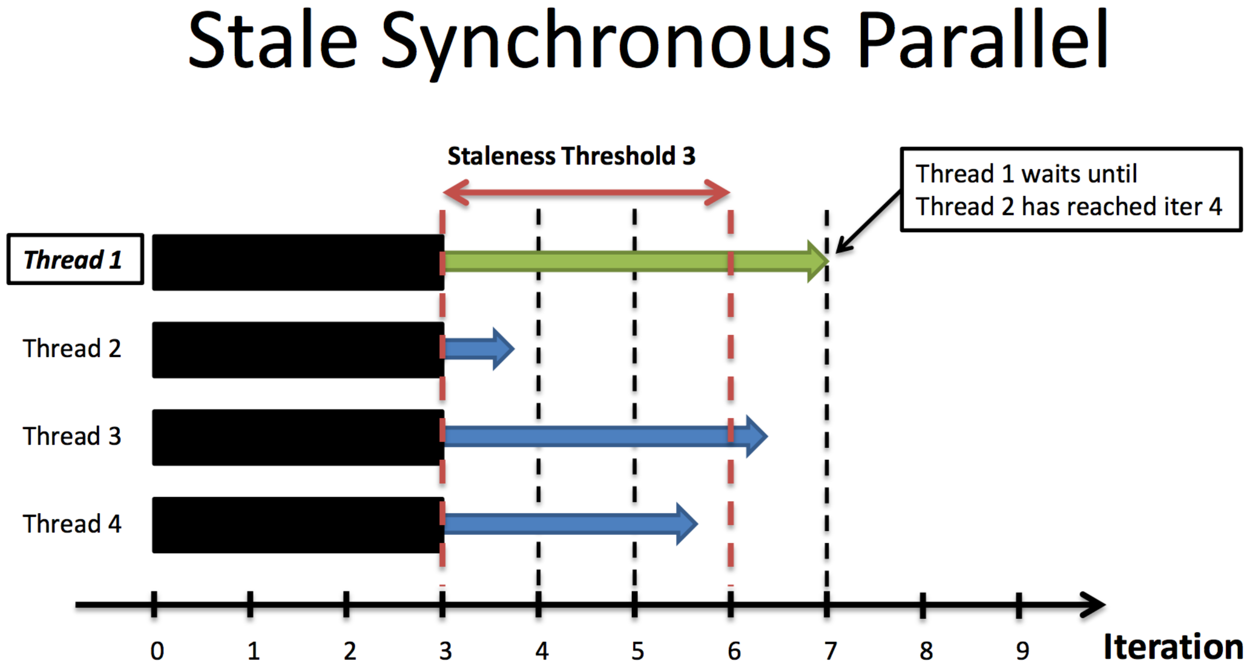
- We can parallelize SGD via
  - Distributed key-value store to share parameters
  - Synchronization scheme to synchronize parameters

- Shared key-value store provides easy interface to read/write shared parameters

- Synchronization scheme determines how parameters are shared among multiple workers
  - Bulk synchronous parallel (e.g., Hadoop)
  - Asynchronous parallel **[Ahmed et al., 2012, Li et al., 2014]**
  - Stale synchronous parallel **[Ho et al., 2013, Dai et al., 2015]**

Parallel SGD with Bounded Async KV-store

- Stale synchronous parallel (SSP) is a synchronization model with bounded staleness – “bounded async”
- Fastest and the slowest workers are ≤s clocks apart


Stale Synchronous Parallel
Staleness Threshold 3
Thread 1 waits until Thread 2 has reached iter 4
Thread 1
Thread 2
Thread 3
Thread 4
0
1
2
3
4
5
6
7
8
9
Iteration

Example KV-Store Program: Lasso

- Lasso example: want to optimize

$$\sum_{i=1}^{N} \|y_i - X_i\beta\|_2^2 + \lambda \sum_{j=1}^{D} |\beta_j|$$

- Put β in KV-store to share among all workers
- Step 1: SGD: each worker draws subset of samples X_i
  - Compute gradient for each term $\|y_i - X_i\beta\|^2$ with respect to β; update β with gradient

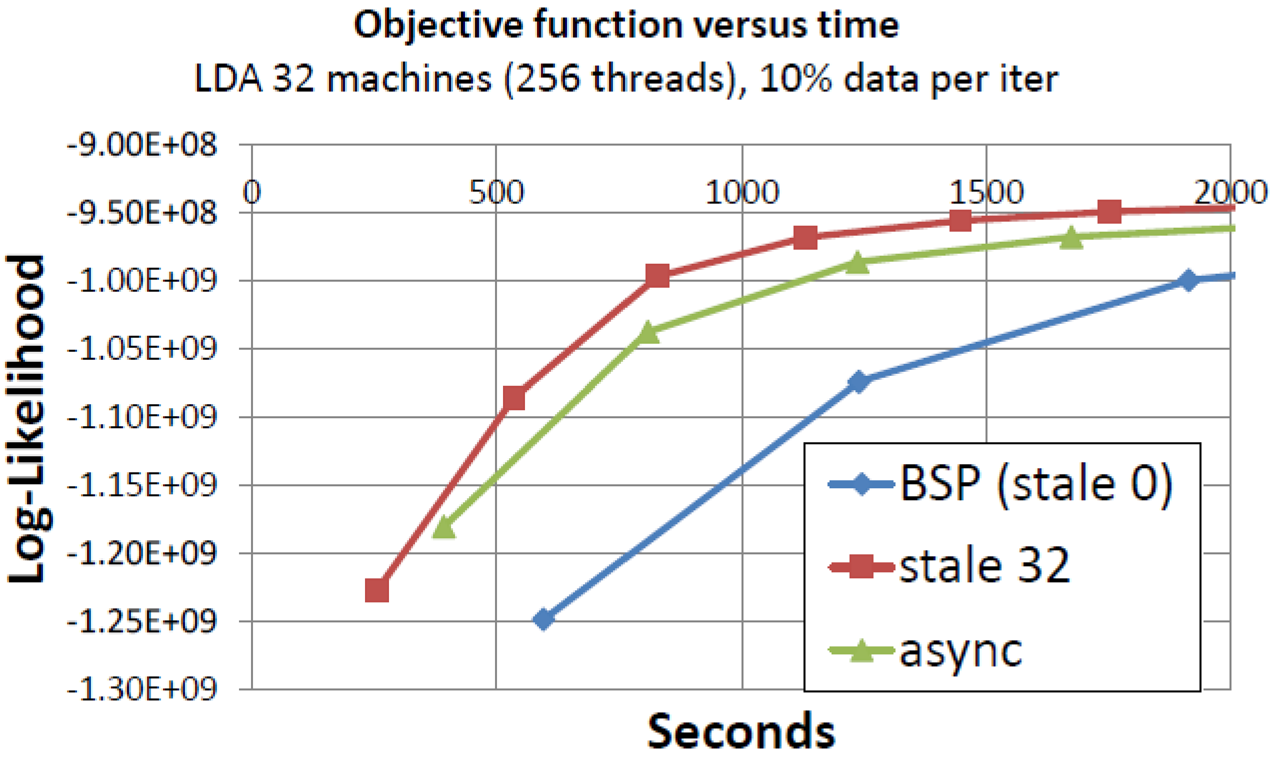
$$\beta^{(t)} = \beta^{(t-1)} + 2(y_i - X_i\beta^{(t-1)})X_i^\top$$

- Step 2: Proximal operator: perform soft thresholding on β

$$\beta_j = \mathrm{sign}(\beta_j)\left(|\beta_j| - \lambda\right)_+$$

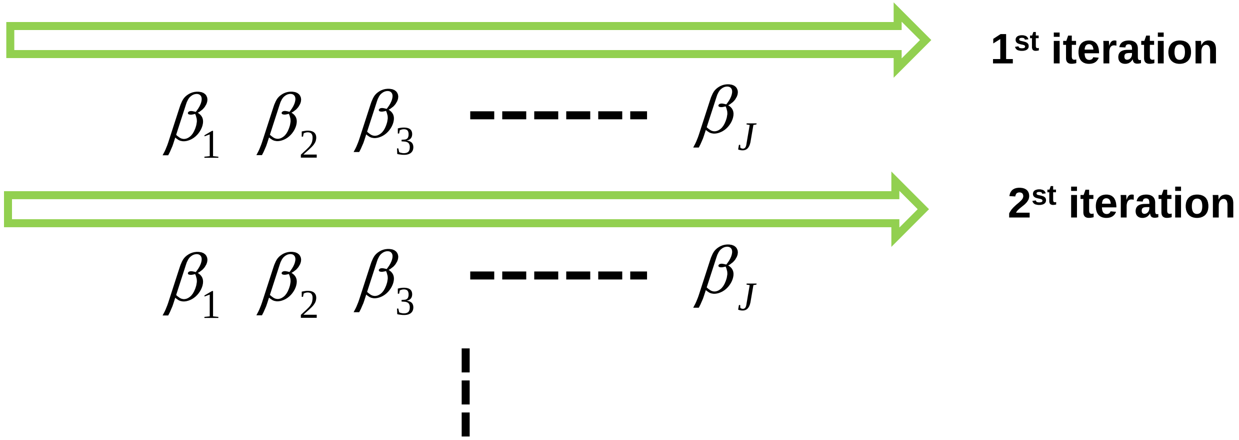
  - Can be done at workers, or at the key-value store itself
- Bounded Asynchronous synchronization allows fast read/write to β, even over slow or unreliable networks

Bounded Async KV-store: Faster and better convergence



Algorithm II: Coordinate Descent

Update each regression coefficient in a cyclic manner



1st iteration

$\beta_1 \;\; \beta_2 \;\; \beta_3$ ------ β_J

2st iteration

$\beta_1 \;\; \beta_2 \;\; \beta_3$ ------ β_J

- **Pros and cons**
  - **Unlike SGD, CD does not involve learning rate**
  - **If CD can be used for a model, it is often comparable to the state-of-the-art (e.g. lasso, group lasso)**
  - **However, as sample size increases, time for each iteration also increases**

Example: Coordinate Descent for Lasso

$$\hat{\boldsymbol{\beta}} = \min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\sum_j\left|\beta_j\right|$$

- Set a subgradient to zero:

$$-\mathbf{x}_j^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})+\lambda t_j = 0$$

Standardization

- Assuming that $\mathbf{x}_j^T\mathbf{x}_j = 1$, we can derive update rule:

$$\beta_j = S\left\{\mathbf{x}_j^T(\mathbf{y}-\sum_{l\neq j} x_l\beta_l),\lambda\right\}$$

Soft thresholding

$$S(x,\lambda) = sign(x)(|x|-\lambda)_+$$

Example: Block Coordinate Descent for Group Lasso

$$\hat{\boldsymbol{\beta}} = \min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_j |\beta_j|$$

- Set it to zero:

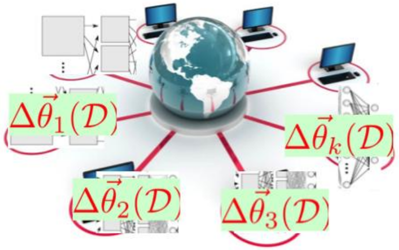
$$-\mathbf{x}_j^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda u_j = 0, \forall j \in \mathbf{g}$$

- In a similar fashion, we can derive update rule for group ***g***

Iterate over each group of coefficients

Parallel Coordinate Descent

[Bradley et al. 2011]

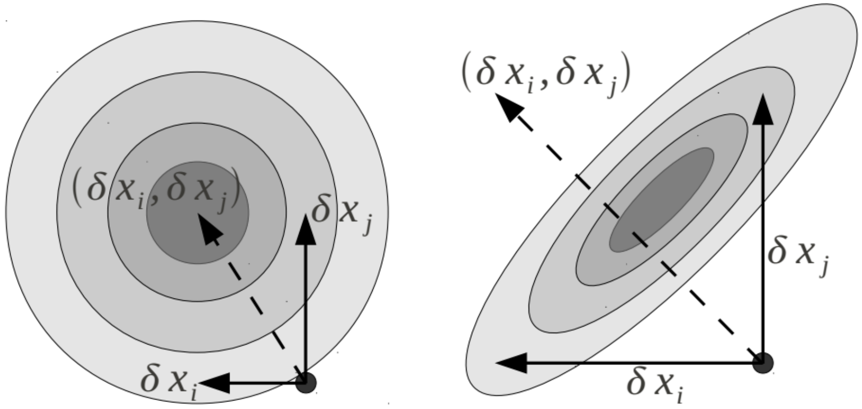

$\Delta\vec{\theta}_1(\mathcal{D})$
$\Delta\vec{\theta}_k(\mathcal{D})$
$\Delta\vec{\theta}_2(\mathcal{D})$
$\Delta\vec{\theta}_3(\mathcal{D})$

- Shotgun, a parallel coordinate descent algorithm
  - Choose parameters to update at random
  - Update the selected parameters in parallel
  - Iterate until convergence

- When features are nearly independent, Shotgun scales almost linearly
  - Shotgun scales linearly up to $P \leq \dfrac{d}{2\rho}$ workers, where ρ is spectral radius of A^TA
  - For uncorrelated features, ρ=1; for exactly correlated features ρ=d
  - No parallelism if features are exactly correlated!

Intuitions for Parallel Coordinate Descent

- Concurrent updates of parameters are useful when features are uncorrelated



Uncorrelated features **Correlated features**

Source:
[Bradley et al., 2011]

- Updating parameters for correlated features may slow down convergence, or diverge parallel CD in the worst case
  - To avoid updates of parameters for correlated features, block-greedy CD has been proposed

Block-greedy Coordinate Descent

[Scherrer et al., 2012]

- Block-greedy coordinate descent generalizes various parallel CD strategies
  - e.g. Greedy-CD, Shotgun, Randomized-CD
- Alg: partition *p* params into B blocks; iterate:
  - Randomly select P blocks
  - Greedily select one coordinate per P blocks
  - Update each selected coordinate
- Sublinear convergence O(1/k) for separable regularizer *r* :

$$\min_x \sum_i f_i(x) + r(x_i)$$

  - Big-O constant depends on the maximal correlation among the B blocks
- Hence greedily cluster features (blocks) to reduce correlation

Parallel Coordinate Descent with Dynamic Scheduler

[Lee et al., 2014]

- STRADS (STRucture-Aware Dynamic Scheduler) allows scheduling of concurrent CD updates
  - STRADS is a general scheduler for ML problems
  - Applicable to CD, and other ML algorithms such as Gibbs sampling

- STRADS improves CD performance via
  - Dependency checking
    - Update parameters which are nearly independent => small parallelization error
  - Priority-based updates
    - More frequently update those parameters which decrease objective function faster

Example Scheduler Program: Lasso

- Schedule step:
  - **Prioritization:** choose next variables β_j to update, with probability proportional to their historical rate of change

$$P(\text{select } \beta_j) \sim (|\beta_j^{(t-1)} - \beta_j^{(t-2)}|)^2 + \epsilon$$

  - **Dependency checking:** do not update β_j, β_k in parallel if feature dimensions j and k are correlated

$$|\boldsymbol{x}_{\cdot j}^{\top} \boldsymbol{x}_{\cdot k}| < \rho \text{ for all } j \neq k$$

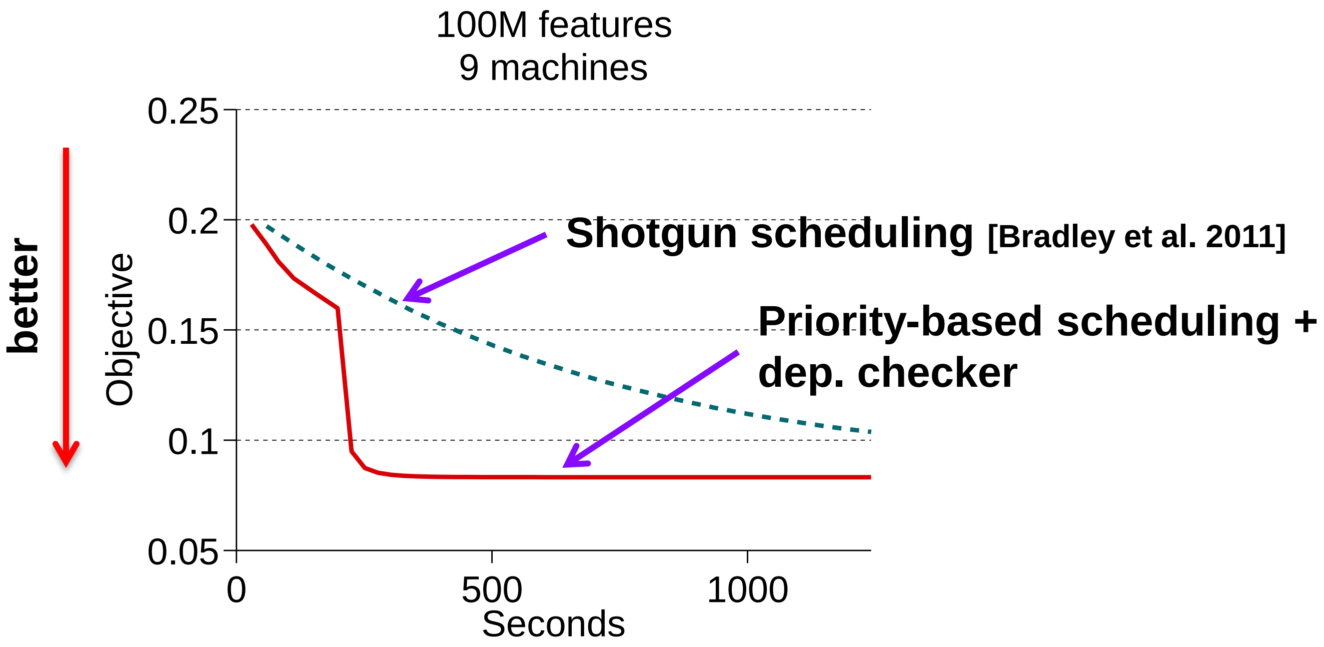
- Update step:
  - For all β_j chosen in Schedule step, in parallel, perform coordinate descent update

$$\beta_j^{(t)} = \beta_j^{(t-1)} - \beta_j^{(t-1)} + \mathbb{S}(X_{\cdot j}^{\top} y - \sum_{k \neq j} X_{\cdot j}^{\top} X_{\cdot k} \beta_k^{(t-1)}, \lambda_n)$$

  - Repeat from Schedule step

Comparison: priority vs. random-scheduling

- Priority-based scheduling converges faster than Shotgun (random) scheduling



Advanced Optimization Techniques

- What if simple methods like SPG, CD are not adequate?

- Advanced techniques at hand
  - Complex regularizer: PG
  - Complex loss: SPG
  - Overlapping loss/regularizer: ADMM

- How to parallelize them? Must understand **math** behind algorithms
  - Which terms should be computed at server
  - Which terms can be distributed to clients
  - …

When Constraints Are Complex:

-- Algorithm III: Proximal Gradient (a.k.a. ISTA)

$$\min_{\mathbf{w}} f(\mathbf{w}) + g(\mathbf{w})$$

- f: loss term, smooth (continuously differentiable)
- g: regularizer, non-differentiable (e.g. 1-norm)

Projected gradient

- g represents some constraint

$$g(\mathbf{w}) = \iota_C(\mathbf{w}) = \begin{cases} 0, & \mathbf{w} \in C \\ \infty, & \text{otherwise} \end{cases}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w})$$

$$\mathbf{w} \leftarrow \arg\min_{\mathbf{z}} \frac{1}{2\eta} \|\mathbf{w} - \mathbf{z}\|^2 + \iota_C(\mathbf{z})$$

$$= \arg\min_{\mathbf{z} \in C} \frac{1}{2} \|\mathbf{w} - \mathbf{z}\|^2$$

Proximal gradient

- g represents some simple function
  - e.g., 1-norm, constraint C, etc.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w}) \qquad \text{gradient}$$

$$\mathbf{w} \leftarrow \underbrace{\arg\min_{\mathbf{z}} \frac{1}{2\eta} \|\mathbf{w} - \mathbf{z}\|^2 + g(\mathbf{z})}_{\text{proximal map}}$$

Algorithm III: Proximal Gradient (a.k.a. ISTA)

- PG hinges on the proximal map **[Moreau, 1965]**:

$$\mathsf{P}_g^\eta(\mathbf{w}) = \arg\min_{\mathbf{z}} \frac{1}{2\eta}\|\mathbf{w}-\mathbf{z}\|^2 + g(\mathbf{z})$$

- Treated as black-box in PG
- Need proximal map efficiently computable, better closed-form
  - True when g is separable and "simple", e.g. 1-norm (separable in each coordinate), non-overlapping group norm, etc.
- Can be demanding if g = g_1+g_2, but vars in g_1, g_2 overlap
- **[Yu, 2013]** gave sufficient conditions for when g = g_1+g_2 can be easily handled: $$\mathsf{P}_{g_1+g_2}^\eta(\mathbf{w}) = \mathsf{P}_{g_1}^\eta\left(\mathsf{P}_{g_2}^\eta(\mathbf{w})\right)$$
  - Useful when $\mathsf{P}_{g_1}^\eta$ and $\mathsf{P}_{g_2}^\eta$ available in closed-forms
  - E.g. fused lasso (Friedman et al.'07): $\mathsf{P}^\eta_{\|\cdot\|_1+\|\cdot\|_{\text{tv}}}(\mathbf{w}) = \mathsf{P}^\eta_{\|\cdot\|_1}\left(\mathsf{P}^\eta_{\|\cdot\|_{\text{tv}}}(\mathbf{w})\right)$

Accelerated PG (a.k.a. FISTA)

[Beck & Teboulle, 2009; Nesterov, 2013; Tseng, 2008]

- PG convergence rate $O(1/(\eta t))$
- Can be boosted to $O(1/(\eta t^2))$
  - Same Lipschitz gradient assumption on f; similar per-step complexity!
  - Lots of follow-up work to the papers cited above

Proximal Gradient

$$\mathbf{v}^t \leftarrow \mathbf{w}^t - \eta \nabla f(\mathbf{w}^t)$$
$$\mathbf{u}^t \leftarrow \mathsf{P}_g^\eta(\mathbf{v}^t)$$
$$\mathbf{w}^{t+1} \leftarrow \mathbf{u}^t + \underbrace{0}_{no} \cdot \underbrace{(\mathbf{u}^t - \mathbf{u}^{t-1})}_{momentum}$$

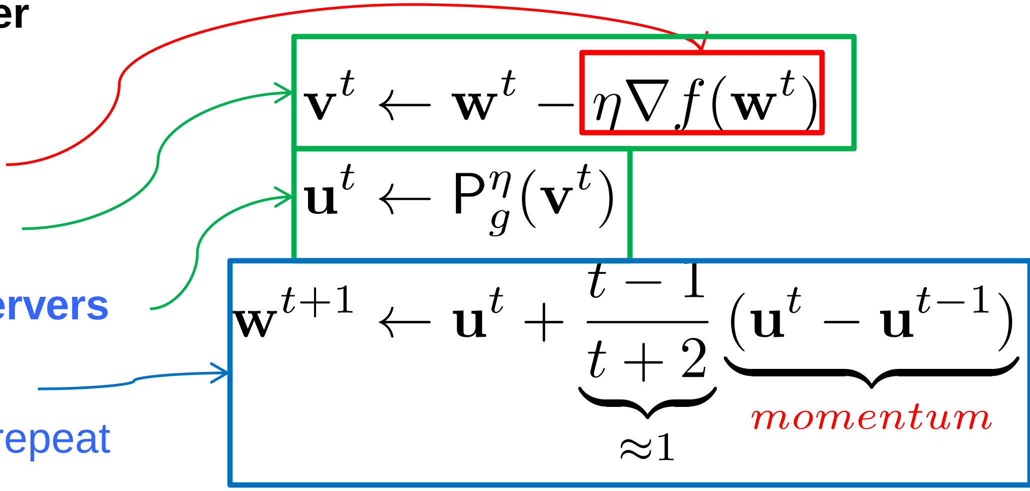
Accelerated Proximal Gradient

$$\mathbf{v}^t \leftarrow \mathbf{w}^t - \eta \nabla f(\mathbf{w}^t)$$
$$\mathbf{u}^t \leftarrow \mathsf{P}_g^\eta(\mathbf{v}^t)$$
$$\mathbf{w}^{t+1} \leftarrow \mathbf{u}^t + \underbrace{\frac{t-1}{t+2}}_{\approx 1} \underbrace{(\mathbf{u}^t - \mathbf{u}^{t-1})}_{momentum}$$

$$\mathsf{P}_g^\eta(w) := \arg\min_z \tfrac{1}{2\eta}\|w - z\|_2^2 + g(z)$$

Parallel (Accelerated) PG

- Bulk Synchronous Parallel Accelerated PG (exact)
  - **[Chen and Ozdaglar, 2012]**
- Asynchronous Parallel (non-accelerated) PG (inexact)
  - **[Li et al., 2014] Parameter Server**
- General strategy:
  1. Compute gradients on **workers**
  2. Aggregate gradients on **servers**
  3. Compute proximal operator on **servers**
  4. Compute momentum on **servers**
  5. Send result $\mathbf{w}^{t+1}$ to **workers** and repeat



$$\mathbf{v}^t \leftarrow \mathbf{w}^t - \boxed{\eta \nabla f(\mathbf{w}^t)}$$

$$\mathbf{u}^t \leftarrow \mathrm{P}_g^{\eta}(\mathbf{v}^t)$$

$$\mathbf{w}^{t+1} \leftarrow \mathbf{u}^t + \underbrace{\frac{t-1}{t+2}}_{\approx 1} \underbrace{(\mathbf{u}^t - \mathbf{u}^{t-1})}_{momentum}$$

- Can apply Hogwild-style asynchronous updates to non-accelerated PG, for empirical speedup
  - Open question: what about accelerated PG? What happens theoretically and empirically to accelerated momentum under asynchrony?

When Objective Is Not Smooth:

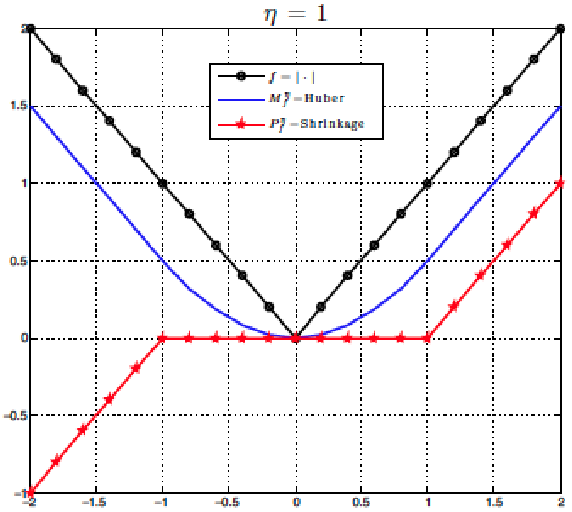
-- Moreau Envelope Smoothing

- So far need *f* to have Lipschitz cont grad, obtained $O(1/t^2)$
- What if not ?
- Can use subgradient, with diminishing step size ⇨ O(1/sqrt(t))
  - Huge gap !!
- Smoothing comes into rescue, if *f* itself is H-Lipschitz cont
  - Approx f with something nicer, like Taylor expansion in calculus 101
- Replace *f* with its Moreau envelope function

$$\mathrm{M}_f^\eta(w) := \min_z \frac{1}{2\eta}\|w - z\|_2^2 + f(z)$$

$$\boxed{\textbf{Prop.}\ \forall w\ , 0 \le f(w) - \mathrm{M}_f^\eta(w) \le \eta H^2/2}$$

  - f(w) = |w|, envelope M_f^η is Huber's func (blue curve)
  - Minimizer gives the proximal map P_f^η (red curve)



Smoothing Proximal Gradient

[Chen et al., 2012]

- Use Moreau envelope as smooth approximation
  - Rich and long history in convex analysis **[Moreau, 1965; Attouch, 1984]**
- Inspired by proximal point alg **[Martinet, 1970; Rockafellar, 1976]**
  - Proximal point alg = PG, when $f \equiv 0$
- Rediscovered in **[Nesterov, 2005]**, led to SPG **[Chen et al., 2012]**

$$\min_{\mathbf{w}} f(\mathbf{w}) + g(\mathbf{w})$$ ⇦ **original**

approx. ⇨ $$\approx \min_{\mathbf{w}} \mathsf{M}_f^\eta(\mathbf{w}) + g(\mathbf{w})$$

- With $\eta = O(1/t)$, SPG converges at $O(1/(\eta t^2)) = O(1/t)$
- Improves subgradient $O(1/\sqrt{t})$
- Requires both efficient P_f^η and P_g^η

Smoothing Proximal Gradient

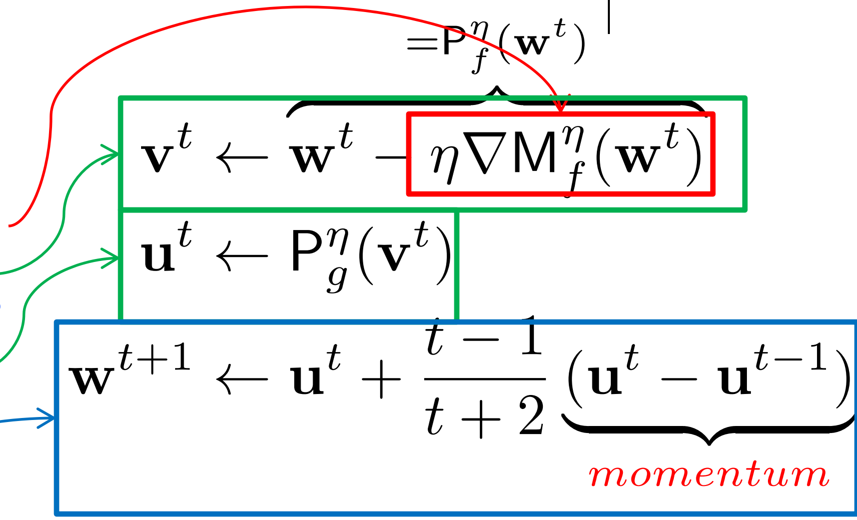
$$\mathbf{v}^t \leftarrow \overbrace{\mathbf{w}^t - \eta\nabla \mathsf{M}_f^\eta(\mathbf{w}^t)}^{=\mathsf{P}_f^\eta(\mathbf{w}^t)}$$

$$\mathbf{u}^t \leftarrow \mathsf{P}_g^\eta(\mathbf{v}^t)$$

$$\mathbf{w}^{t+1} \leftarrow \mathbf{u}^t + \frac{t-1}{t+2}\underbrace{(\mathbf{u}^t - \mathbf{u}^{t-1})}_{\mathit{momentum}}$$

Parallel SPG?

- No known work yet
- Possible strategy:
  1. Compute smoothed gradients on **workers**
  2. Aggregate smoothed gradients on **servers**
  3. Compute proximal operator on **servers**
  4. Compute momentum on **servers**
  5. Send result **w**$^{t+1}$ to **workers** and repeat



$$\mathbf{v}^t \leftarrow \overbrace{\mathbf{w}^t - \boxed{\eta \nabla \mathrm{M}_f^\eta(\mathbf{w}^t)}}^{=\mathrm{P}_f^\eta(\mathbf{w}^t)}$$

$$\mathbf{u}^t \leftarrow \mathrm{P}_g^\eta(\mathbf{v}^t)$$

$$\mathbf{w}^{t+1} \leftarrow \mathbf{u}^t + \frac{t-1}{t+2}\underbrace{(\mathbf{u}^t - \mathbf{u}^{t-1})}_{momentum}$$

- The above strategy is exact under Bulk Synchronous Parallel (just like accelerated PG).
  - Not clear how asynchronous updates impact smoothing+momentum
  - Open research topic

When Variables Are Coupled:

-- Algorithm IV: ADMM

Canonical form: $$\min_{w,z} \overbrace{f(w)+g(z)}^{\text{☺ uncoupled}}, \quad \text{s.t.} \quad \overbrace{Aw+Bz=c}^{\text{☹ coupled}},$$

where $w \in \mathbb{R}^m, z \in \mathbb{R}^p, A : \mathbb{R}^m \to \mathbb{R}^q, B : \mathbb{R}^p \to \mathbb{R}^q, c \in \mathbb{R}^q$

- Numerically challenging because
  - Function f or g nonsmooth or constrained (i.e., can take value ∞)
  - Linear constraint couples the variables w and z
  - Large scale, interior point methods NA
- Naively alternating x and z does not work
  - Min w^2 s.t. w + z = 1; optimum clearly is w = 0
  - Start with say w = 1 → z = 0 → w = 1 → z = 0 …
- However, without coupling, can solve separately w and z
  - Idea: try to decouple vars in the constraint!

Example: Empirical Risk Minimization (ERM)

$$\min_w g(w) + \overbrace{\sum_{i=1}^{n} f_i(w)}^{\text{☹ coupled}}$$

- Each i corresponds to a training point (x_i, y_i)
- Loss f_i measures the fitness of the model parameter w
  - least squares: $f_i(w) = (y_i - w^\top x_i)^2$
  - support vector machines: $f_i(w) = (1 - y_i w^\top x_i)_+$
  - boosting: $f_i(w) = \exp(-y_i w^\top x_i)$
  - logistic regression: $f_i(w) = \log(1 + \exp(-y_i w^\top x_i))$
- g is the regularization function, e.g. $\lambda_n \|w\|_2^2$ or $\lambda_n \|w\|_1$
- Vars coupled in obj, but not in constraint (none)
  - Reformulate: transfer coupling from obj to constraint
  - Arrive at canonical form, allow unified treatment later

How to: variable duplication

- Duplicate variables to achieve canonical form

$$\min_{w} g(w) + \sum_{i=1}^{n} f_i(w)$$

$$\Big\downarrow \quad v = [w_1, \dots, w_n]^\top$$

$$\min_{v,z} \quad g(z) + \underbrace{\sum\nolimits_i f_i(w_i),}_{f(v)} \quad \text{s.t.} \quad \underbrace{w_i = z, \forall i}_{v-[I,\dots,I]^\top z=0}$$

- Global consensus constraint: $\forall i, w_i = z$
  - All w_i must (eventually) agree
- Downside: many extra variables, increase problem size
  - Implicitly maintain duplicated variables

Augmented Lagrangian

Canonical form: $$\min_{\mathbf{w},\mathbf{z}} f(\mathbf{w}) + g(\mathbf{z}), \quad \text{s.t.} \quad A\mathbf{w} + B\mathbf{z} = \mathbf{c},$$

where $\mathbf{w} \in \mathbb{R}^m, \mathbf{z} \in \mathbb{R}^p, A : \mathbb{R}^m \to \mathbb{R}^q, B : \mathbb{R}^p \to \mathbb{R}^q, \mathbf{c} \in \mathbb{R}^q$

- Intro Lagrangian multiplier $\boldsymbol{\lambda}$ to decouple variables

$$\min_{\mathbf{w},\mathbf{z}} \max_{\boldsymbol{\lambda}} \underbrace{f(\mathbf{w}) + g(\mathbf{z}) + \boldsymbol{\lambda}^\top (A\mathbf{w} + B\mathbf{z} - \mathbf{c}) + \frac{\mu}{2} \|A\mathbf{w} + B\mathbf{z} - \mathbf{c}\|_2^2}_{L_\mu(\mathbf{w},\mathbf{z};\boldsymbol{\lambda})}$$

  - L_μ : augmented Lagrangian
  - More complicated min-max problem, but no coupling constraints

Algorithm IV: ADMM

$$\min_{\mathbf{w},\mathbf{z}} \max_{\boldsymbol{\lambda}} \underbrace{f(\mathbf{w}) + g(\mathbf{z}) + \boldsymbol{\lambda}^\top (A\mathbf{w} + B\mathbf{z} - \mathbf{c}) + \frac{\mu}{2}\|A\mathbf{w} + B\mathbf{z} - \mathbf{c}\|_2^2}_{L_\mu(\mathbf{w},\mathbf{z};\boldsymbol{\lambda})}$$

- Fix dual $\boldsymbol{\lambda}$, block coordinate descent on primal w, z

$$\mathbf{w}^{t+1} \leftarrow \arg\min_{\mathbf{w}} L_\mu(\mathbf{w}, \mathbf{z}^t; \boldsymbol{\lambda}^t) \equiv f(\mathbf{w}) + \frac{\mu}{2}\|A\mathbf{w} + B\mathbf{z}^t - \mathbf{c} + \boldsymbol{\lambda}^t/\mu\|^2$$

$$\mathbf{z}^{t+1} \leftarrow \arg\min_{\mathbf{z}} L_\mu(\mathbf{w}^{t+1}, \mathbf{z}; \boldsymbol{\lambda}^t) \equiv g(\mathbf{z}) + \frac{\mu}{2}\|A\mathbf{w}^{t+1} + B\mathbf{z} - \mathbf{c} + \boldsymbol{\lambda}^t/\mu\|^2$$

- Fix primal w, z, gradient ascent on dual $\boldsymbol{\lambda}$

$$\boldsymbol{\lambda}^{t+1} \leftarrow \boldsymbol{\lambda}^t + \eta(A\mathbf{w}^{t+1} + B\mathbf{z}^{t+1} - \mathbf{c})$$

- Step size η can be large, e.g. $\eta = \mu$
  - Usually rescale $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}/\eta$ to remove η

Row partition (data parallel)

$$\min_z g(z) + \sum_{i=1}^{n} f_i(A_i z - c_i)$$

- each i corresponds to a (block of) training data A_i
- all summands f_i share the same global variable z
- all ERM in this form: SVM, lasso, logistic regression, etc.
- parallellize by duplicating z into w_1, … w_n

$$\min_{\mathbf{w}=[\mathbf{w}_1,\ldots,\mathbf{w}_n],\mathbf{z}} \underbrace{g(\mathbf{z})}_{\textbf{server}} + \sum_i \underbrace{f_i(A_i\mathbf{w}_i - \mathbf{c})}_{\textbf{worker machine i}}, \quad \text{s.t.} \quad \mathbf{z} - \mathbf{w}_i = 0, \forall i$$

- Exact Synchronization (bulk sync parallel) needed

Column partition (model parallel)

$$\min_{\mathbf{w}} f\Big(\sum_{j=1}^{p} A_j w_j - c\Big) + \sum_{j=1}^{p} g_j(w_j)$$

- in columns data $A = [A_1, \ldots, A_p]$, variables $\mathbf{w} = [w_1, \ldots, w_p]$
- Each function g_j have its own variable w_j
- All variables w_j coupled in f
- parallelize by adding auxiliary variable $\mathbf{z} = [z_1, \ldots, z_p]$

$$\min_{\mathbf{w},\mathbf{z}} f\underbrace{(\sum_j z_j - c)}_{\textbf{server}} + \sum_j \underbrace{g_j(w_j)}_{\textbf{worker machine j}}, \quad \text{s.t.} \quad A_j w_j - z_j = 0, \forall j$$

- Exact Synchronization (bulk sync parallel) needed

Asynchronous Parallel ADMM

[Zhang & Kwok, 2014]

- Only simplified consensus problem being studied:

$$\min_{\mathbf{w}=[\mathbf{w}_1,\ldots,\mathbf{w}_n],\mathbf{z}} \quad \sum_{i=1}^{n} f_i(\mathbf{w}_i), \quad \text{s.t.} \quad \mathbf{w}_i - \mathbf{z} = 0, \forall i$$

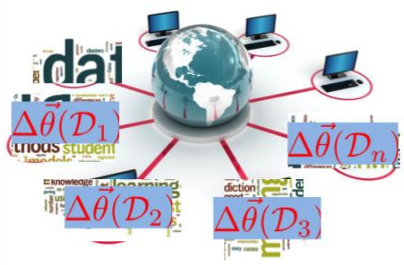
- Can distribute the primal updates for each w_i

$$(\mathbf{w}_1, \ldots, \mathbf{w}_n) \leftarrow \arg\min_{\mathbf{w}} L_\mu(\mathbf{w}, \mathbf{z}; \boldsymbol{\lambda})$$

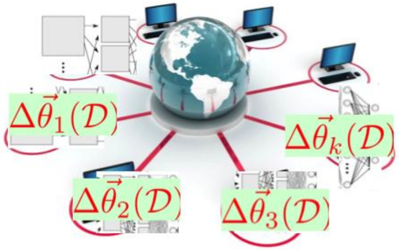
- But dual update $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \sum_i \mathbf{w}_i - \mathbf{z}$ can happen only after all primal updates – barrier bottleneck
- How to alleviate the barrier bottleneck?
  - Asynchronously execute dual update after seeing s out of n primal updates
  - Condition: no machine is too far behind
    - Can be achieved with bounded staleness **[Ho et al., 2013]**
  - Asynchronous convergence proved in **[Zhang & Kwok, 2014]**

Outline: Optimization & MCMC Algorithms

- Optimization Algorithms
  - Stochastic gradient descent
  - Coordinate descent
  - Proximal gradient methods
    - ISTA, FASTA, Smoothing proximal gradient
  - ADMM


$\Delta\vec{\theta}(\mathcal{D}_1)$
$\Delta\vec{\theta}(\mathcal{D}_n)$
$\Delta\vec{\theta}(\mathcal{D}_2)$
$\Delta\vec{\theta}(\mathcal{D}_3)$

- Markov Chain Monte Carlo Algorithms
  - Auxiliary Variable methods
  - Embarrassingly Parallel MCMC
  - Parallel Gibbs Sampling
    - Data parallel
    - Model parallel


$\Delta\vec{\theta}_1(\mathcal{D})$
$\Delta\vec{\theta}_k(\mathcal{D})$
$\Delta\vec{\theta}_2(\mathcal{D})$
$\Delta\vec{\theta}_3(\mathcal{D})$

Example Probabilistic Program: Topic Models

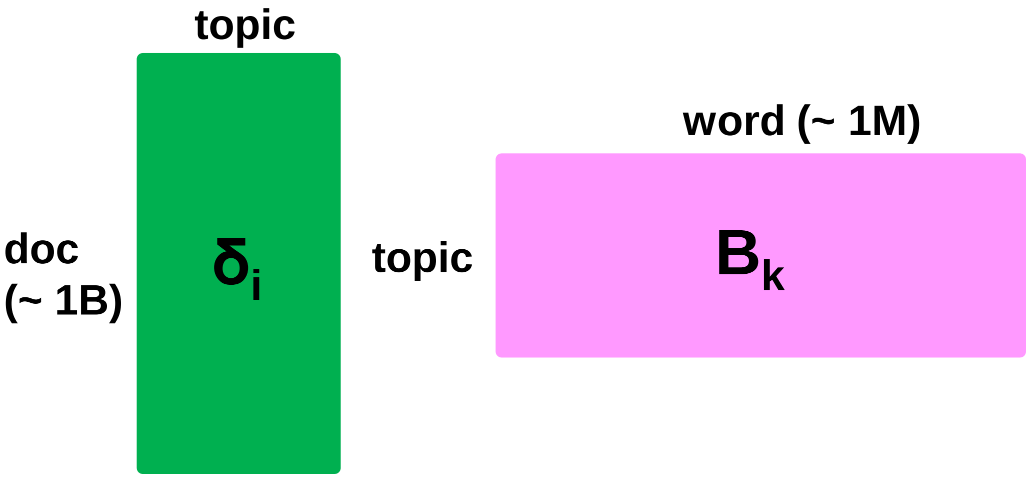
$$\sum_{i=1}^{N}\sum_{j=1}^{N_i} \ln \mathbb{P}_{Categorical}(x_{ij} \mid z_{ij}, B) + \sum_{i=1}^{N}\sum_{j=1}^{N_i} \ln \mathbb{P}_{Categorical}(z_{ij} \mid \delta_i)$$

Generative model of data

$$+ \sum_{i=1}^{N} \ln \mathbb{P}_{Dirichlet}(\delta_i \mid \alpha) + \sum_{i=k}^{K} \ln \mathbb{P}_{Dirichlet}(B_k \mid \beta)$$

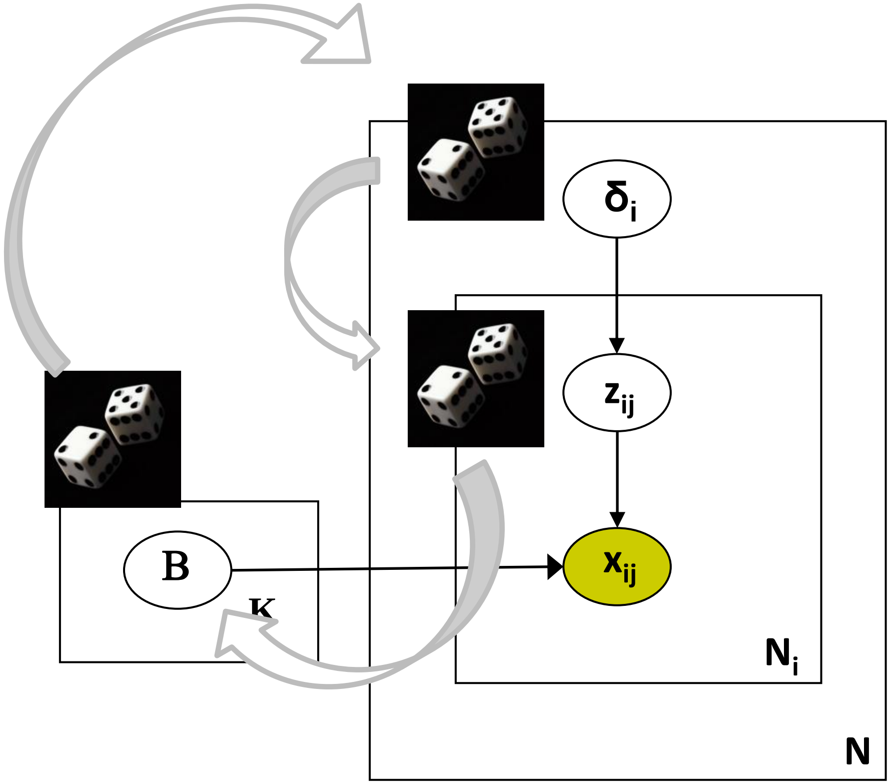
Priors on parameters

- Generative model
  - Fit topics to each word x_{ij} in each doc i
  - Uses categorical distributions with parameters δ and B
- Parameter priors
  - Induce sparsity in δ and B
  - Can also incorporate structure
    - E.g. asymmetric prior

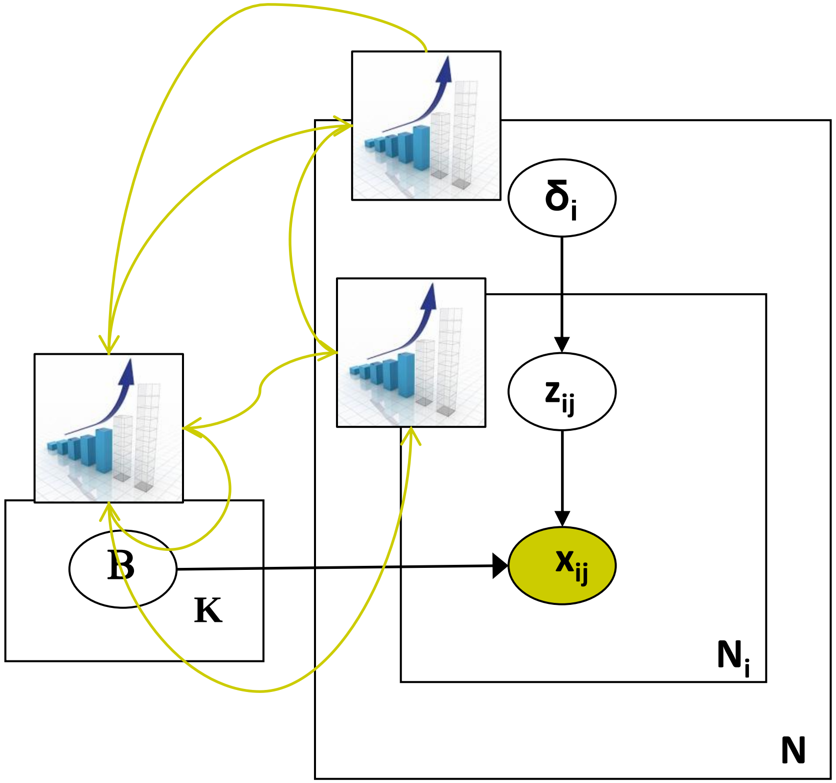


Inference for Probabilistic Programs: MCMC and SVI

Markov Chain Monte Carlo:
Randomly sample each variable in sequence
Next set of slides on this

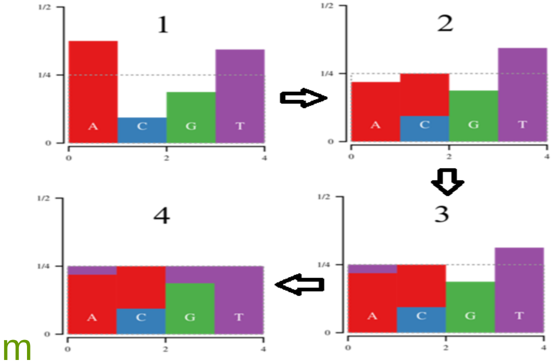


Variational Inference:
Gradient ascent on variables
Can be treated as an optimization problem

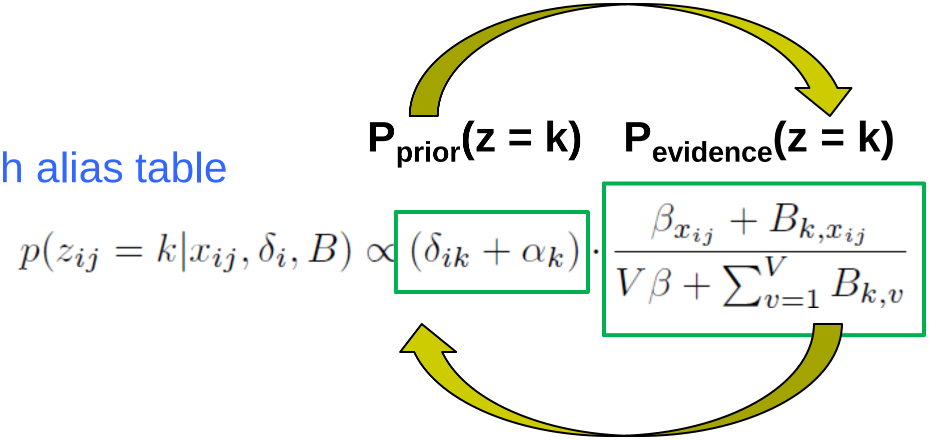


Preliminaries: Speeding up sequential MCMC

- Technique 1: Alias tables
  - Sample from categorical distribution in amortized O(1)
  - “Throw darts at a dartboard”
  - Ex: probability distribution [0.5, 0.25, 0.25]
    - => alias table {1, 1, 2, 3} => draw from table uniformly at random



- Technique 2: Cyclic Metropolis Hastings **[Yuan et al., 2015]**
  - Exploit Bayesian form P(z=k) = $P_{evidence}(k)$ * $P_{prior}(k)$
    - Propose z_1 from $P_{evidence}(k)$
    - Accept/Reject z_1
    - Propose z_2 from $P_{prior}(k)$
    - Accept/Reject z_2 … repeat
  - $P_{prior}(k)$, $P_{evi}(k)$ cheap to compute with alias table



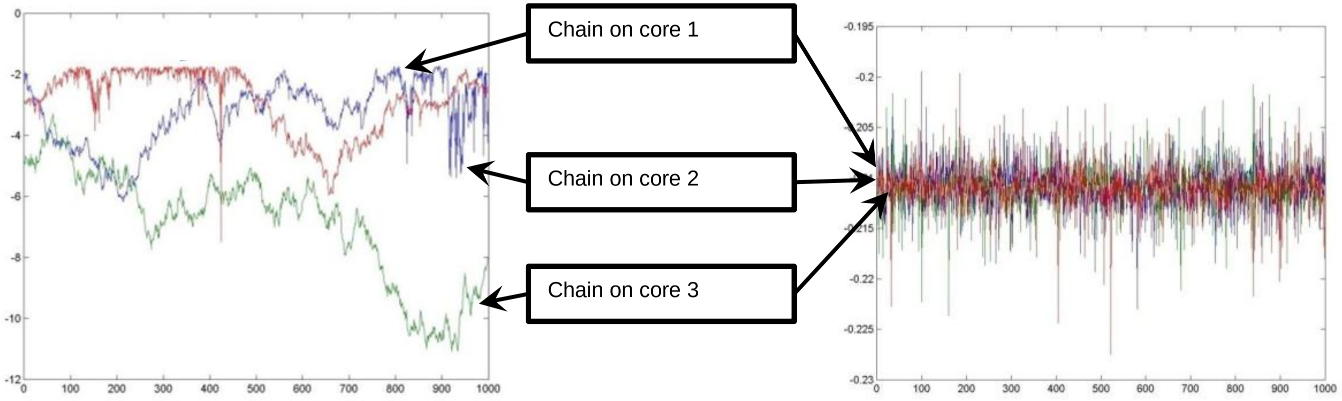
$P_{prior}(z = k)$ **$P_{evidence}(z = k)$**

$$p(z_{ij} = k | x_{ij}, \delta_i, B) \propto \boxed{(\delta_{ik} + \alpha_k)} \cdot \boxed{\frac{\beta_{x_{ij}} + B_{k,x_{ij}}}{V\beta + \sum_{v=1}^{V} B_{k,v}}}$$

- Other speedup techniques
  - Stochastic Gradient MCMC
  - Stochastic Variational Inference

Parallel and Distributed MCMC: Classic methods

- Classic parallel MCMC solution 1
  - Take multiple chains in parallel, take average/consensus between chains.
    - But what if each chain is very slow to converge?
    - Need full dataset on each process – no data parallelism!



Chain on core 1

Chain on core 2

Chain on core 3

Not converged

Converged

Parallel and Distributed MCMC: Classic methods

- Classic parallel MCMC solution 2
  - Sequential Importance Sampling
  - Rewrite distribution over n variables as telescoping product over proposals q():

$$r(x_{1:n}) = r_1(x_1)\prod_{k=2}^{n}\alpha_k(x_{1:k}) \quad \text{where} \quad \alpha_n(x_{1:n}) = \frac{P'_n(x_{1:n})}{P'_{n-1}(x_{1:n-1})q_n(x_n \mid x_{1:n-1})}$$

  - SIS algorithm:
    - Parallel draw samples $x^i_n \sim q_n(x_n|x^i_{1:n-1})$
    - Parallel compute unnorm. wgts. $r^i_n = r^i_{n-1}\alpha_n(x^i_{1:n}) = r^i_{n-1}\frac{P'_n(x^i_{1:n})}{P'_{n-1}(x^i_{1:n-1})q_n(x^i_n \mid x^i_{1:n-1})}$
    - Compute normalized weights w^i_n by normalizing r^i_n
  - Drawback: variance of SIS samples increases exponentially with n
    - Need resampling + take many chains to control variance

- **Let us look at newer solutions to parallel MCMC…**

Solution I: Induced Independence via Auxiliary Variables [Dubey et al. 2013, 2014]

- Auxiliary Variable Inference: reformulate model as P independent models
  - Example below: **Dirichlet Process** for mixture models
  - Also applies to **Hierarchical Dirichlet Process** for topic models

- AV model (left) equivalent to standard DP model (right)

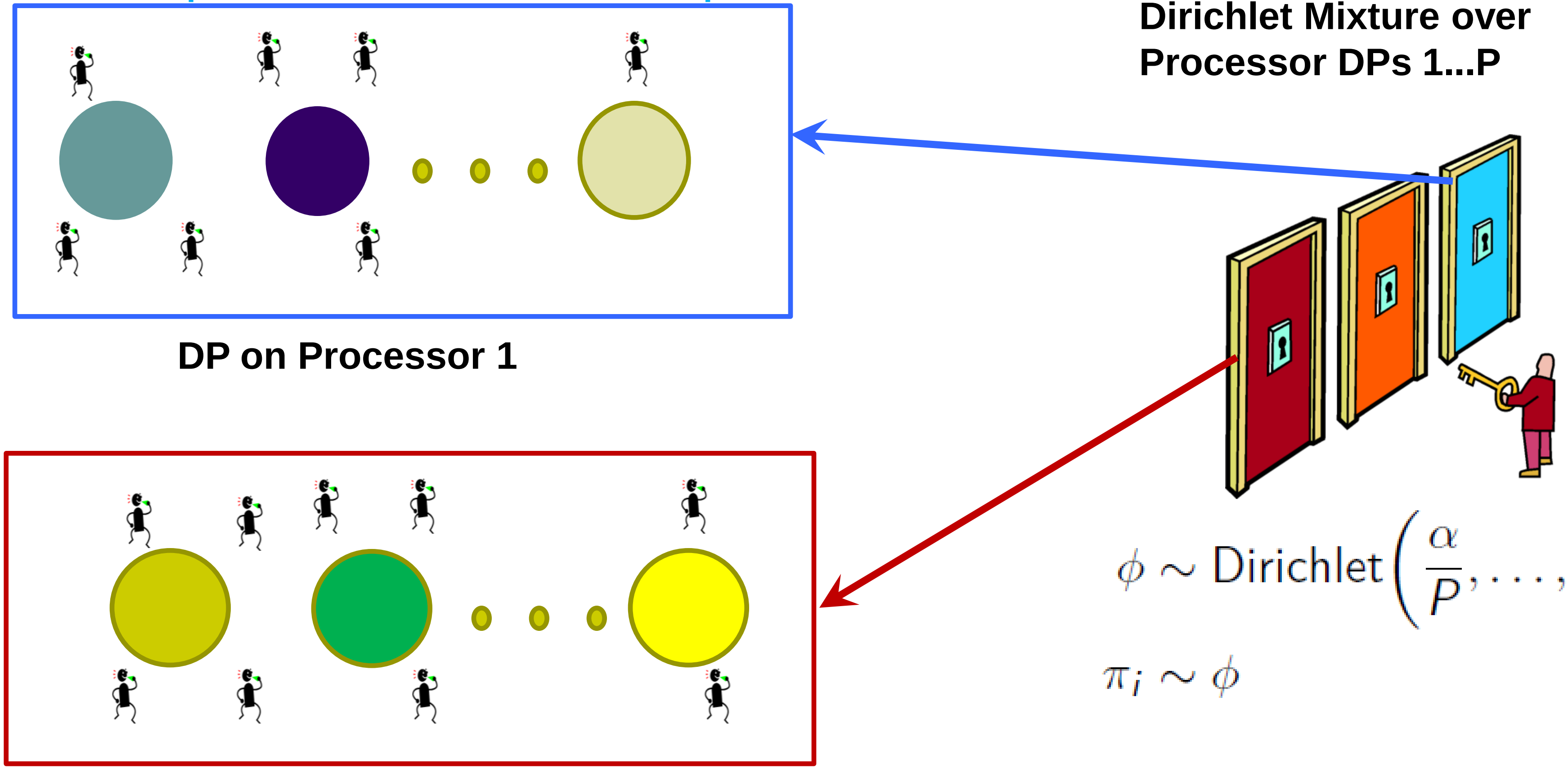
$$\begin{aligned}
&D_j \sim \mathrm{DP}\left(\frac{\alpha}{P}, H\right), \quad j = 1, \ldots, P \\
&\phi \sim \mathrm{Dirichlet}\left(\frac{\alpha}{P}, \ldots, \frac{\alpha}{P}\right) \\
&\pi_i \sim \phi \\
&\theta_i \sim D_{\pi_i} \\
&x_i \sim f(\theta_i), \quad i = 1, \ldots, N.
\end{aligned}$$

⟺

$$\begin{aligned}
D &\sim \mathrm{DP}(\alpha, H)\,, \\
\theta_i &\sim D\,, \\
x_i &\sim f(\theta_i)
\end{aligned}$$

Solution I: Induced Independence via Auxiliary Variables [Dubey et al., 2013, 2014]

- Why does it work? A mixture over Dirichlet processes is equivalent to a Dirichlet processes



Dirichlet Mixture over Processor DPs 1…P

DP on Processor 1

DP on Processor P

$$\phi \sim \text{Dirichlet}\left(\frac{\alpha}{P}, \ldots, \frac{\alpha}{P}\right)$$

$$\pi_i \sim \phi$$

Solution I: Induced Independence via Auxiliary Variables [Dubey et al., 2013, 2014]

- Parallel inference algorithm:
  - Initialization: assign data randomly across P Dirichlet Processes; assign each Dirichlet Process to one worker p=1..P
  - Repeat until convergence:
    - Each worker performs Gibbs sampling on local data within its DP
    - Each worker swaps its DP's clusters with other workers, via Metropolis-Hastings:
      - For each cluster c, propose a new DP q=1..P
      - Compute proposal probability of c moving to p
      - Acceptance ratio depends on cluster size

- Can be done asynchronously in parallel without affecting performance

Solution II: Embarrassingly Parallel (but correct) MCMC [Neiswanger et al., 2014]

- High-level idea:
  - Run MCMC in parallel on data subsets; no communication between machines.
  - Combine samples from machines to construct full posterior distribution samples.

- Objective: recover full posterior distribution

$$p(\theta|x^N) \propto p(\theta)p(x^N|\theta) = p(\theta)\prod_{i=1}^N p(x_i|\theta)$$

- Definitions:
  - Partition data into M subsets $\{x^{n_1}, \ldots, x^{n_M}\}$
  - Define m-th machine's "subposterior" to be $p_m(\theta) \propto p(\theta)^{\frac{1}{M}} p(x^{n_m}|\theta)$
    - Subposterior: "The posterior given a subset of the observations with an underweighted prior".

Embarassingly Parallel MCMC

- Algorithm
  1. For m=1…M independently in parallel, draw samples from each subposterior p_m
  2. Estimate subposterior density product $p_1 \cdots p_M(\theta) \propto p(\theta | x^N)$ (and thus the full posterior $p(\theta | x^N)$) by "combining subposterior samples"

- "Combine subposterior samples" via nonparametric estimation
  1. Given T samples $\{\theta_{t_m}^m\}_{t_m=1}^T$ from each subposterior p_m :
     - Construct Kernel Density Estimate (Gaussian kernel, bandwidth h):

$$\widehat{p}_m(\theta) = \frac{1}{T} \sum_{t_m=1}^{T} \frac{1}{h^d} K\left(\frac{\|\theta - \theta_{t_m}^m\|}{h}\right) = \frac{1}{T} \sum_{t_m=1}^{T} \mathcal{N}_d(\theta | \theta_{t_m}^m, h^2 I_d)$$

  2. Combine subposterior KDEs:

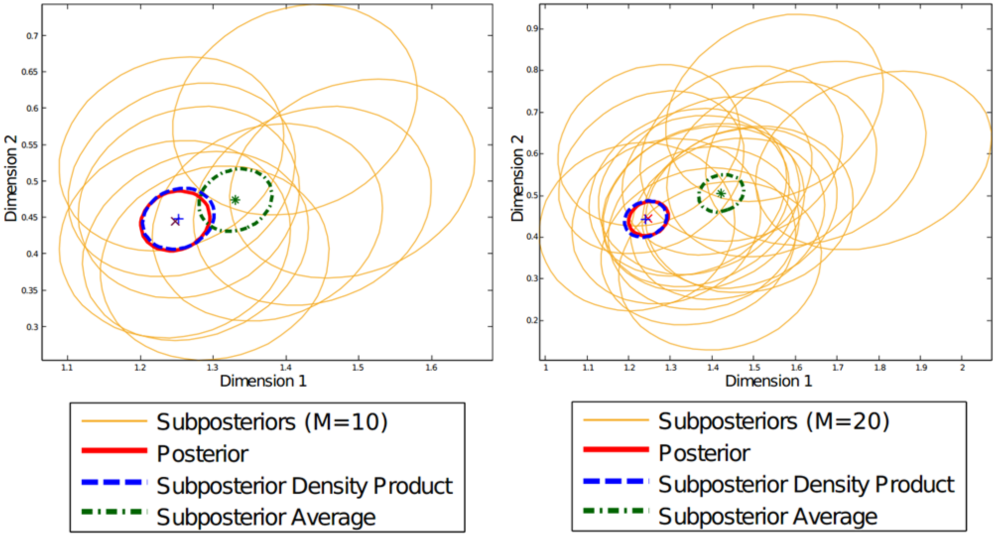
$$\widehat{p_1 \cdots p_M}(\theta) = \widehat{p}_1 \cdots \widehat{p}_M(\theta) = \frac{1}{T^M} \prod_{m=1}^{M} \sum_{t_m=1}^{T} \mathcal{N}_d(\theta | \theta_{t_m}^m, h^2 I_d) \quad \propto \sum_{t_1=1}^{T} \cdots \sum_{t_M=1}^{T} w_{t\cdot} \mathcal{N}_d\left(\theta \middle| \bar{\theta}_{t\cdot}, \frac{h^2}{M} I_d\right)$$

     - where

$$\bar{\theta}_{t\cdot} = \frac{1}{M} \sum_{m=1}^{M} \theta_{t_m}^m \qquad w_{t\cdot} = \prod_{m=1}^{M} \mathcal{N}_d\left(\theta_{t_m}^m | \bar{\theta}_{t\cdot}, h^2 I_d\right)$$

Embarassingly Parallel MCMC

- Simulations:
  - More subposteriors = tighter estimates
  - EPMCMC recovers correct parameter
  - Naïve subposterior averaging does not!



© Eric Xing @ CMU, 2015

Solution III: Parallel Gibbs Sampling

- Many MCMC algorithms
  - Sequential Monte Carlo **[Canini et al., 2009]**
  - Hybrid VB-Gibbs **[Mimno et al., 2012]**
  - Langevin Monte Carlo **[Patterson et al., 2013]**
  - …

- Common choice in tech/internet industry:
  - Collapsed Gibbs sampling **[Griffiths and Steyvers, 2004]**
  - e.g. topic model Collapsed Gibbs sampler:

$$p(z_{ij} = k | x_{ij}, \delta_i, B) \propto (\delta_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + B_{k,x_{ij}}}{V\beta + \sum_{v=1}^{V} B_{k,v}}$$

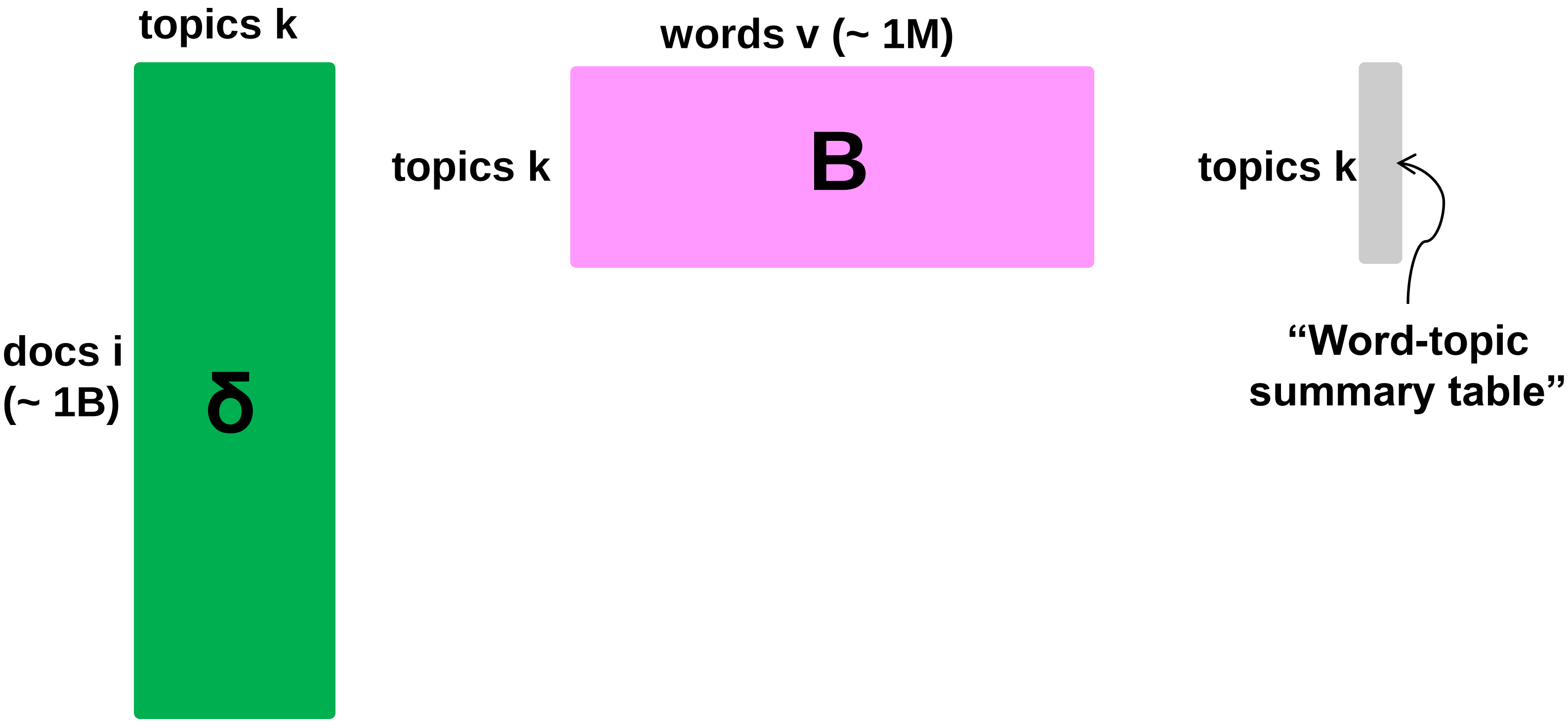
Properties of Collapsed Gibbs Sampling (CGS)

$$p(z_{ij} = k | x_{ij}, \delta_i, B) \propto (\delta_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + B_{k,x_{ij}}}{V\beta + \sum_{v=1}^{V} B_{k,v}}$$

- Simple equation: easy for system engineers to scale up
- Good theoretical properties
  - Rao-Blackwell theorem guarantees CGS sampler has lower variance (better stability) than naïve Gibbs sampling
- Empirically robust
  - Errors in δ, B do not affect final stationary distribution by much
- Updates are sparse: fewer parameters to send over network
- Model parameters δ, B are sparse: less memory used
  - If it were dense, even 1M word * 10K topic ≈ 40GB already!

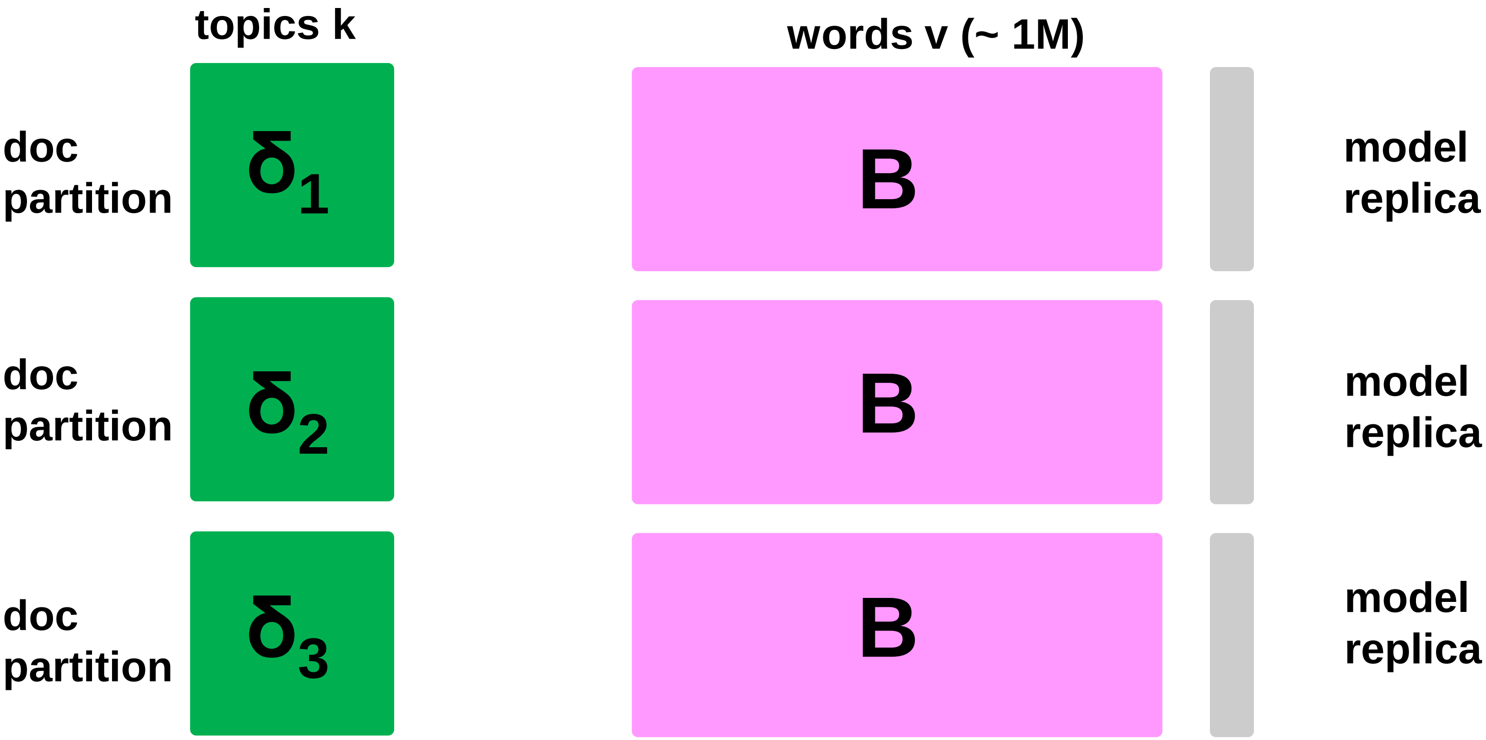
CGS Example: Topic Model sampler

$$p(z_{ij} = k | x_{ij}, \delta_i, B) \propto (\boxed{\delta}_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + \boxed{B}_{k,x_{ij}}}{V\beta + \boxed{\sum_{v=1}^{V} B_{k,v}}}$$



Data Parallelization for CGS Topic Model Sampler

$$p(z_{ij}=k|x_{ij},\delta_i,B) \propto (\boxed{\delta}_{ik}+\alpha_k)\cdot\frac{\beta_{x_{ij}}+\boxed{B}_{k,x_{ij}}}{V\beta+\boxed{\sum_{v=1}^{V} B_{k,v}}}$$



topics k

words v (~ 1M)

doc partition δ_1 B model replica

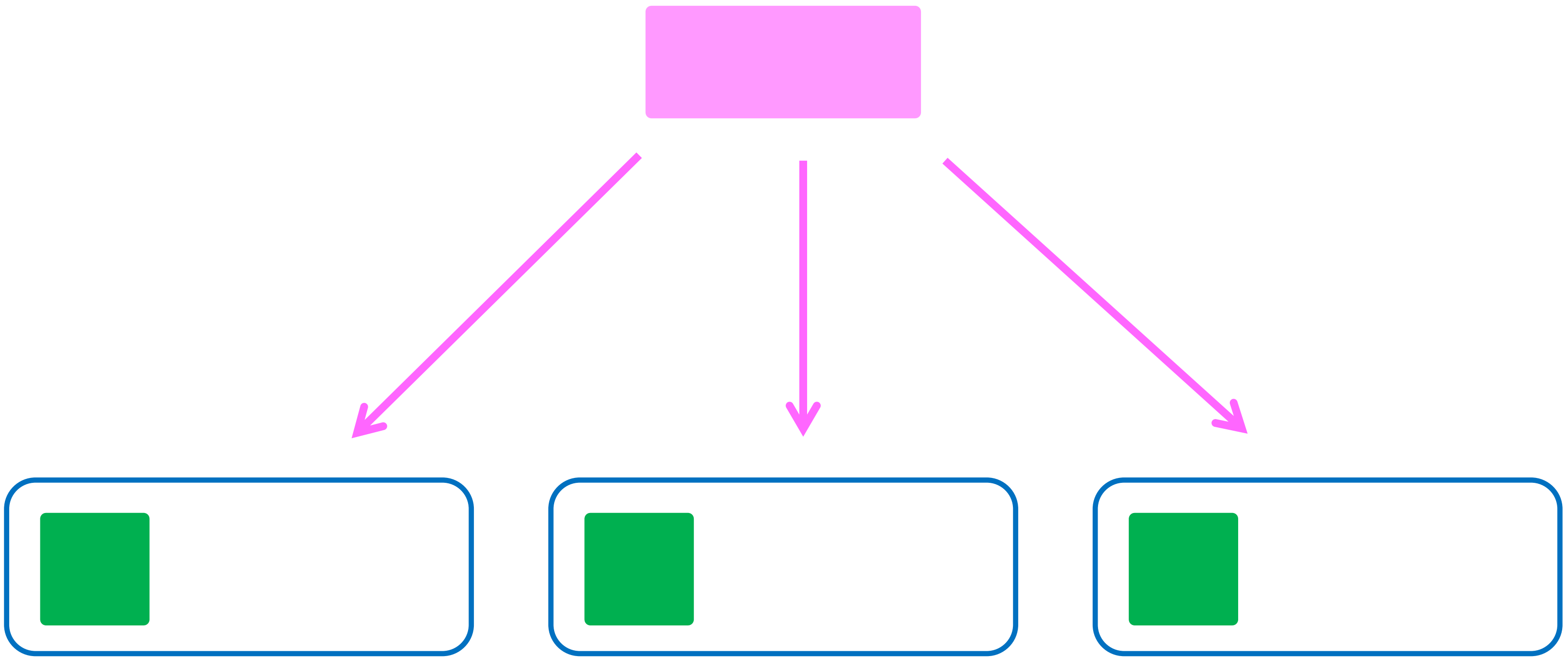
doc partition δ_2 B model replica

doc partition δ_3 B model replica

Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

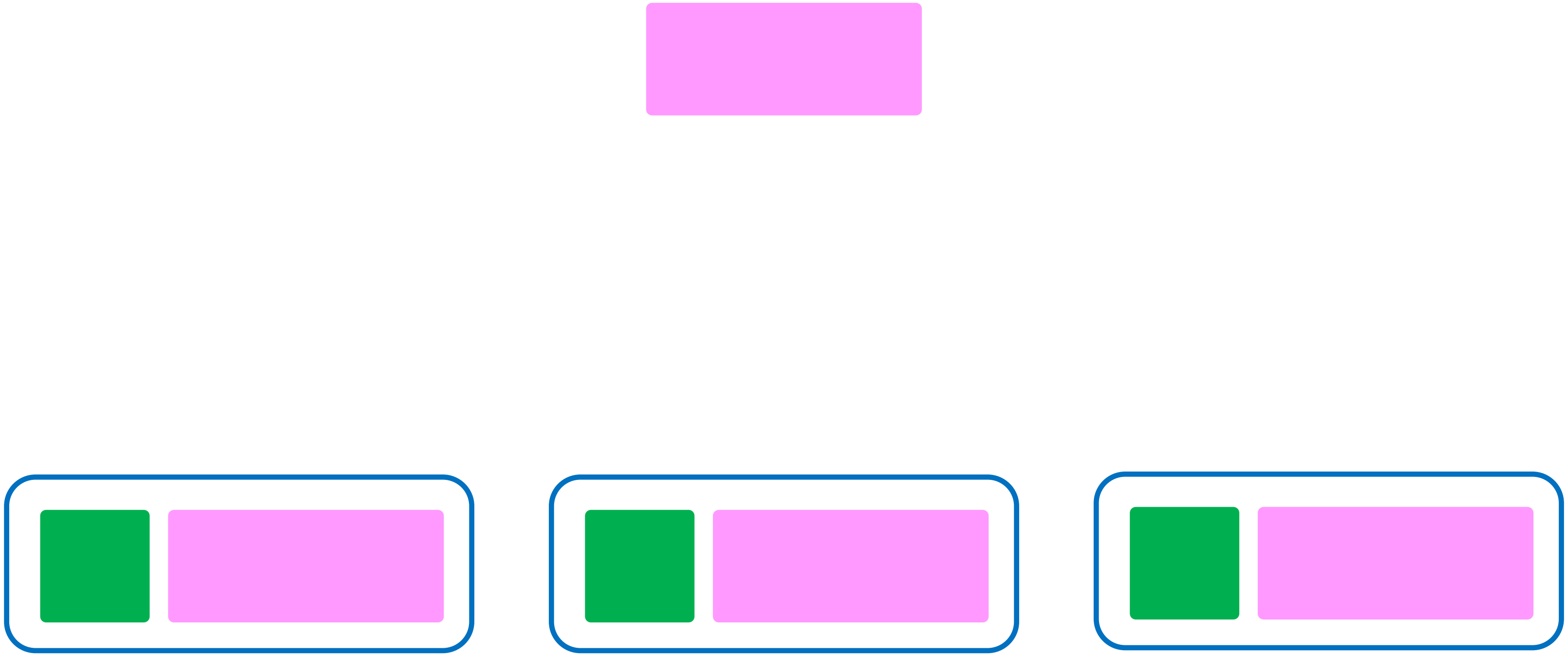
- Step 1: broadcast central model



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

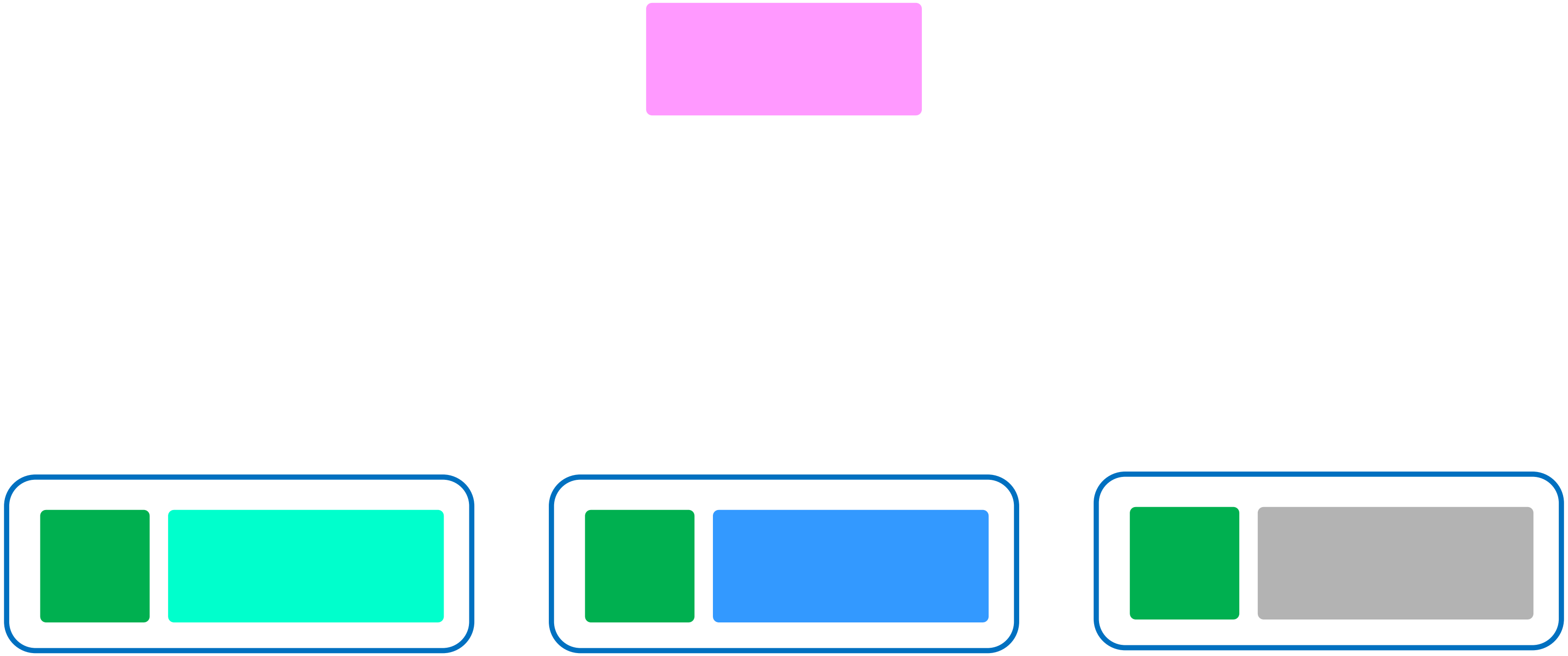
- Step 1: broadcast central model



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

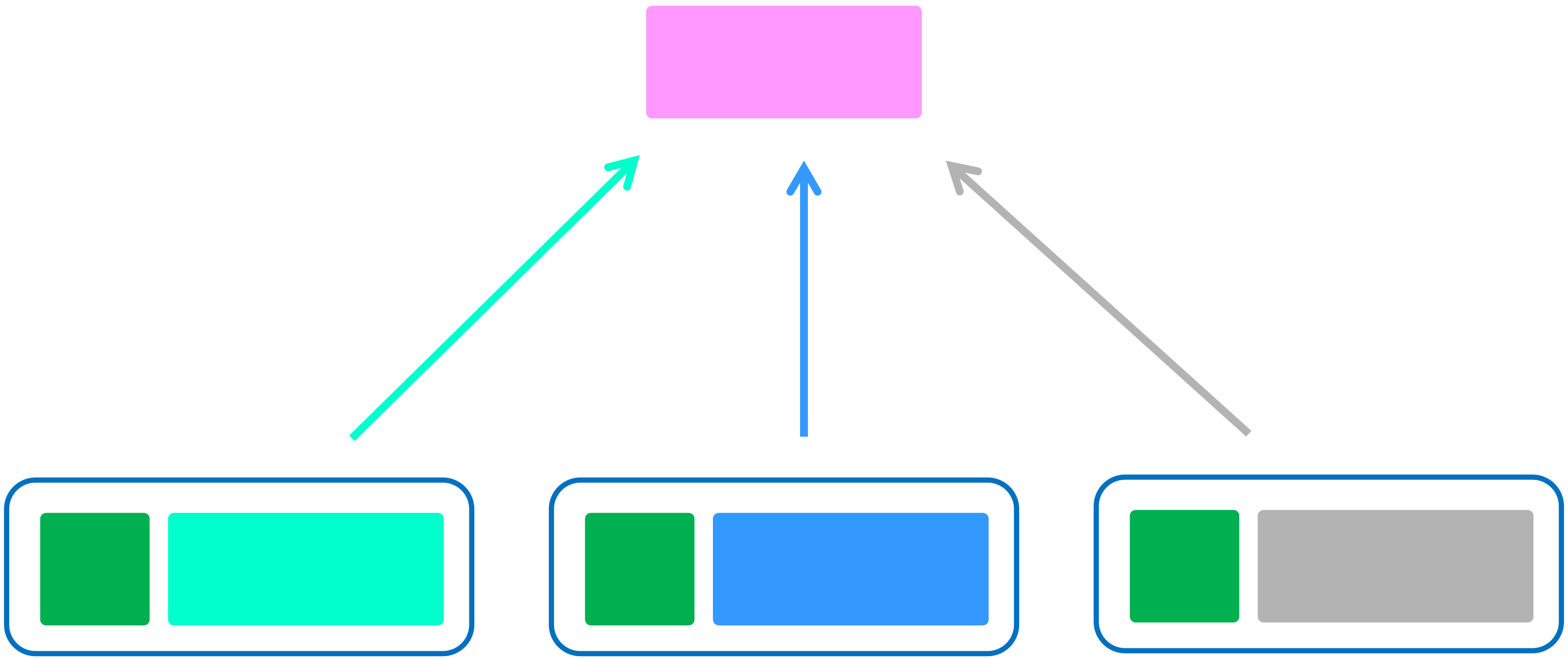
- Step 2: Perform Gibbs sampling in parallel



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

- Step 3: commit changes back to the central model



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

- Approximate
  - Convergence not guaranteed – Markov Chain ergodicity broken
  - Results generally “good enough” for industrial use
- Bulk synchronous parallel
  - CPU cycles are wasted while synchronizing the model
  - Asynchronous and bounded-asynchronous extensions possible **[Smola et al., 2010; Ahmed et al., 2012, Dai et al., 2015]**
- How to overlap communication and computation for better efficiency?

Error in data-parallel LDA

- Consider the CGS equation:

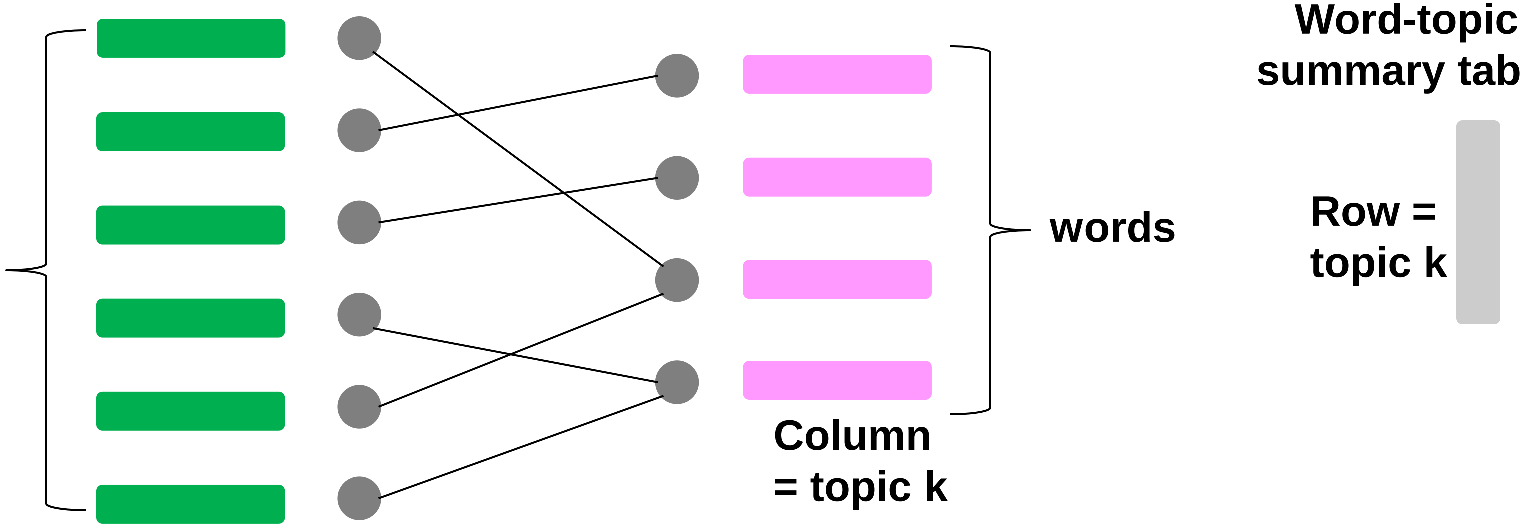
$$p(z_{ij} = k|x_{ij}, \delta_i, B) \propto (\boxed{\delta}_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + \boxed{B}_{k,x_{ij}}}{V\beta + \boxed{\sum_{v=1}^{V} B_{k,v}}}$$

- Data-parallelism incurs error in B (the pink box) and the summation term (the gray box)
  - Both quantities are duplicated onto workers; their values become stale as sampling proceeds
  - True even for bulk synchronous parallel execution!
- Asynchrony helps somewhat
  - Communicate very frequently to reduce staleness
- Is there a better solution?

Model-Parallel Strategy 1: GraphLab LDA [Low et al., 2010; Gonzalez et al., 2012]

- Think graphically: token = edge

$$p(z_{ij}=k|x_{ij},\delta_i,B) \propto (\boxed{\delta}_{ik}+\alpha_k)\cdot\frac{\beta_{x_{ij}}+\boxed{B}_{k,x_{ij}}}{V\beta+\boxed{\sum_{v=1}^{V} B_{k,v}}}$$



docs

Column = topic k

words

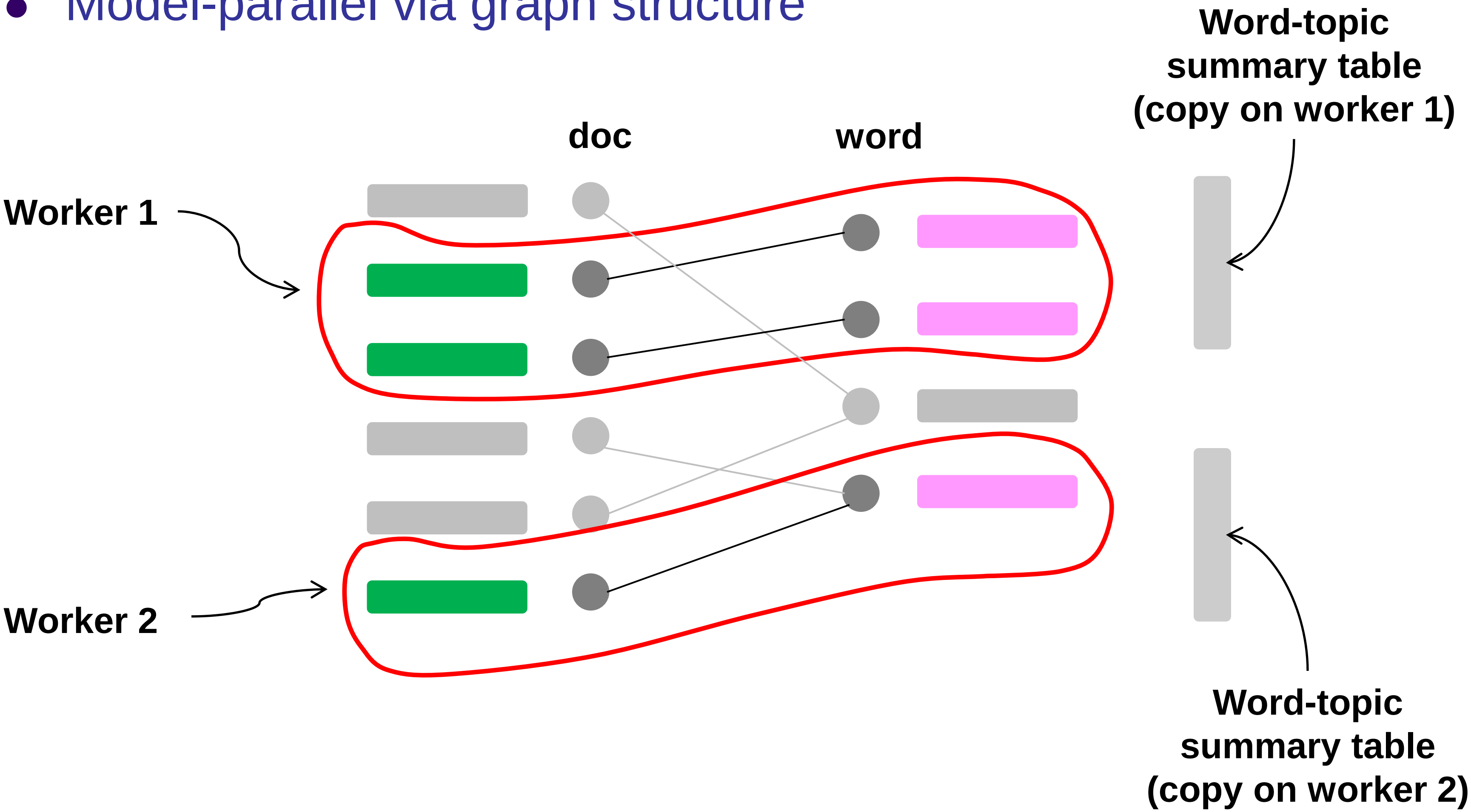
Column = topic k

Word-topic summary table

Row = topic k

Model-Parallel Strategy 1: GraphLab LDA [Low et al., 2010; Gonzalez et al., 2012]

- Model-parallel via graph structure



Model-Parallel Strategy 1: GraphLab LDA [Low et al., 2010; Gonzalez et al., 2012]

- Asynchronous communication
  - Overlaps computation and communication – iterations are faster

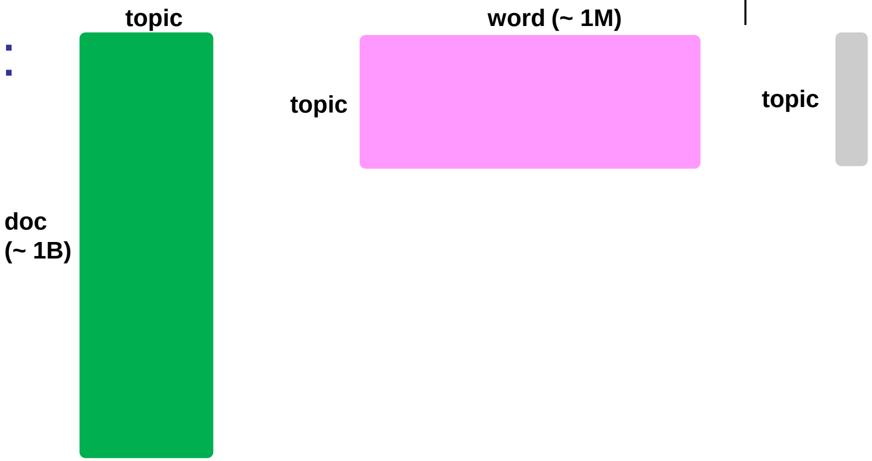
- Model-parallelism means each machine only stores a subset of statistics
  - Less memory usage if implemented well

- Drawback: need to convert problem into a graph
  - Vertex-cut duplicates lots of vertices, canceling out savings
- Are there other ways to partition the problem?

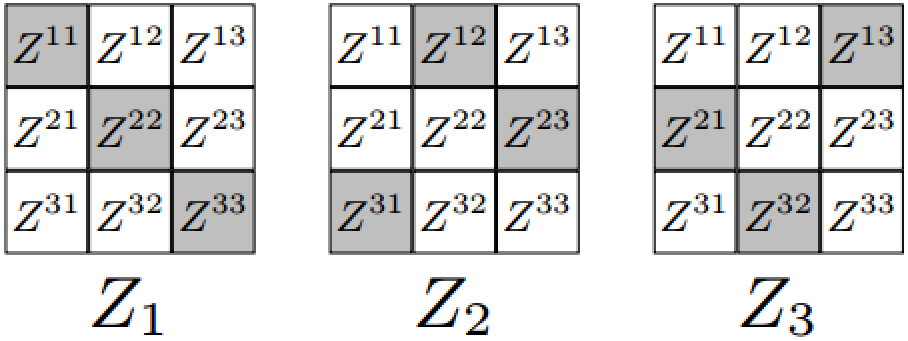
Model-Parallel Strategy 2: LightLDA (Petuum LDA v2)

[Yuan et al., 2015]

- Topic model matrix structure:



- Idea: non-overlapping matrix partition:

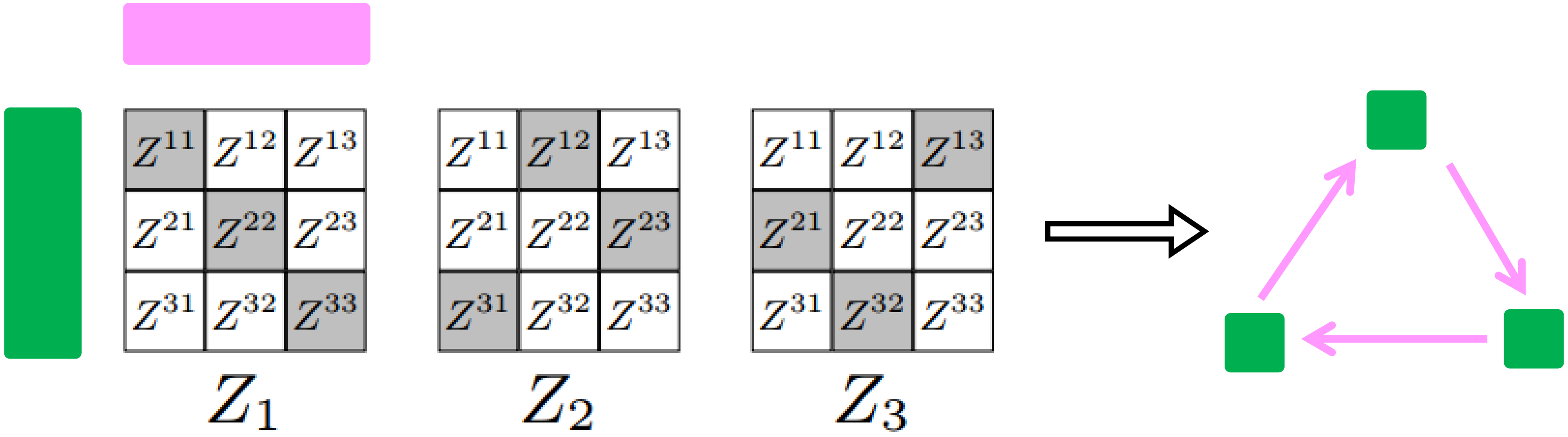

Z^{11} Z^{12} Z^{13}
Z^{21} Z^{22} Z^{23}
Z^{31} Z^{32} Z^{33}
Z_1
Z^{11} Z^{12} Z^{13}
Z^{21} Z^{22} Z^{23}
Z^{31} Z^{32} Z^{33}
Z_2
Z^{11} Z^{12} Z^{13}
Z^{21} Z^{22} Z^{23}
Z^{31} Z^{32} Z^{33}
Z_3

Source: [Gemulla et al., 2011]

Model-Parallel Strategy 2: LightLDA (Petuum LDA v2)

[Yuan et al., 2015]

- Non-overlapping partition of the word count matrix
- Fix data at machines, send model to machines as needed



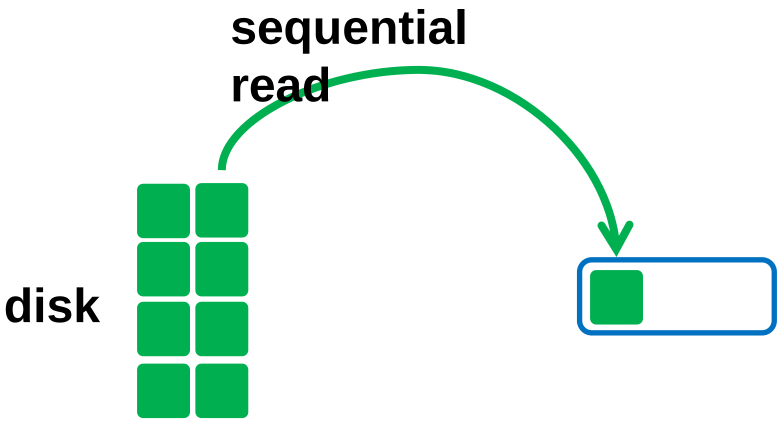
z^{11} z^{12} z^{13}
z^{21} z^{22} z^{23}
z^{31} z^{32} z^{33}
Z_1
z^{11} z^{12} z^{13}
z^{21} z^{22} z^{23}
z^{31} z^{32} z^{33}
Z_2
z^{11} z^{12} z^{13}
z^{21} z^{22} z^{23}
z^{31} z^{32} z^{33}
Z_3

Source: [Gemulla et al., 2011]

Model-Parallel Strategy 2: LightLDA (Petuum LDA v2)

[Yuan et al., 2015]

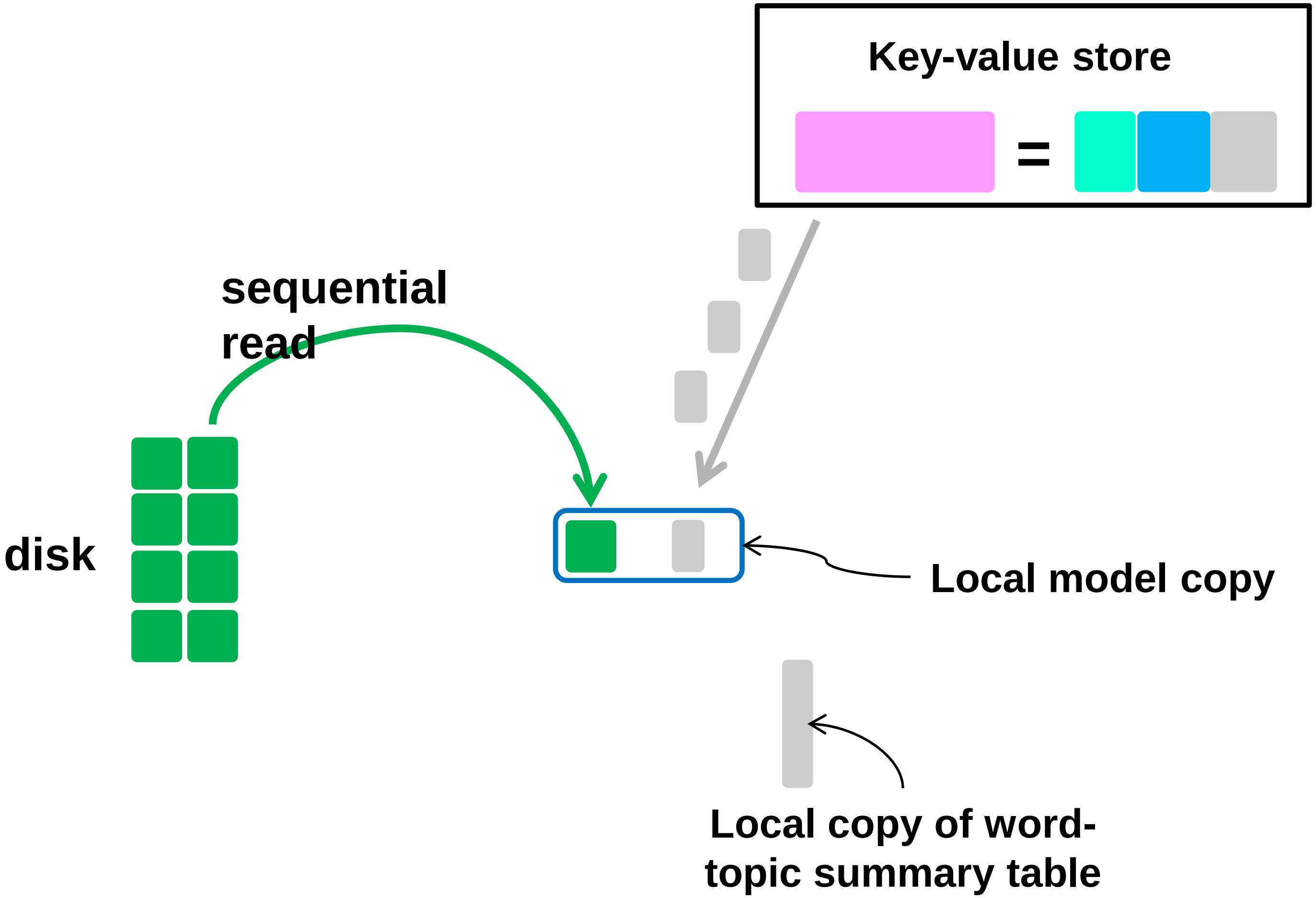
- During preprocessing: determine set of words used in each data block 🟩
- Begin training: load each data block from disk



Model-Parallel Strategy 2: LightLDA (Petuum LDA v2)

[Yuan et al., 2015]

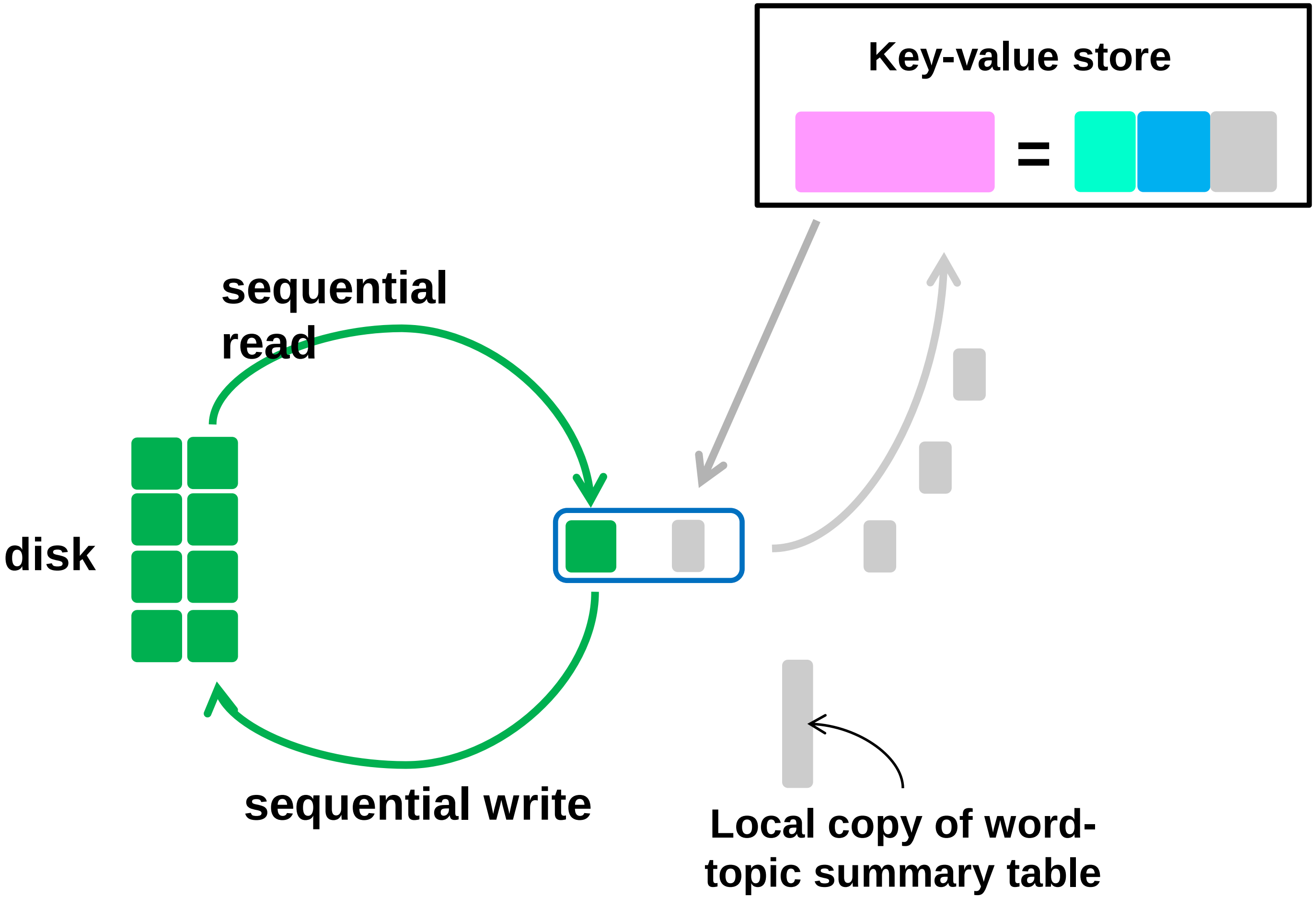
- Pull the set of words from Key-Value store



Model-Parallel Strategy 2: LightLDA (Petuum LDA v2)

[Yuan et al., 2015]

- Sample, write result to disk, send changes back to KV-store



Model-Parallel Strategy 2: LightLDA (Petuum LDA v2)

[Yuan et al., 2015]

- Model-parallel advantage: disjoint words/docs on each machine
  - Gibbs sampling almost equivalent to sequential case
  - More accurate than data-parallel LDA
  - Fast, asynchronous execution possible
- Compared to GraphLab LDA:
  - Simple partitioning strategy – less system overheads, easier to implement
  - Need to be careful about load imbalance (some docs will touch a particular word more times than others)
    - Solution: pre-group documents by word frequency

Error in model-parallel LDA

- Recall the CGS equation:

$$p(z_{ij}=k|x_{ij},\delta_i,B) \propto (\boxed{\delta}_{ik}+\alpha_k)\cdot \frac{\beta_{x_{ij}}+\boxed{B}_{k,x_{ij}}}{V\beta+\boxed{\sum_{v=1}^{V} B_{k,v}}}$$

- Model-parallelism only has error in summation term (gray box)
  - Summation term is very large for Big Data (billions of docs) => error negligible
  - Compared to data-parallelism: error due to B (pink box) eliminated

Distributed ML Algorithms Summary

- Many parallel algorithms for both Optimization and MCMC
- They share common parallelization themes
  - **Embarrassingly parallel:** combine results from multiple independent problems, e.g. PSGD, EP-MCMC
  - **Stochastic over data:** approximate functions/ gradients with expectation over subset of data, then parallelize over data subsets, e.g. SGD
  - **Model-parallel:** parallelize over model variables, e.g. Coordinate Descent
  - **Auxiliary variables:** decompose problem by decoupling dependent variables, e.g. ADMM, Auxiliary Variable MCMC
- Considerations
  - **Regularizers, model structure:** may need sequential proximal or projection step, e.g. Stochastic Proximal Gradient
  - **Data partitioning:** for data-parallel, how to split data over machines?
  - **Model partitioning:** for model-parallel, how to split model over machines? Need to be careful as model variables are not necessarily independent of each other.

Implementing Distributed ML Algorithms

- Implementing high-performance distributed ML is not easy
- If not careful, can end up slower than single machine!
  - System bottlenecks (load imbalance, network bandwidth & latency) are not trivial to engineer around
- Even if algorithm is theoretically sound and has attractive properties, still need to pay attention to system aspects
  - Bandwidth (communication volume limits)
  - Latency (communication timing limits)
  - Data and Model partitioning (machine memory limitation, also affects comms volume)
  - Data and Model scheduling (affects convergence rate, comms volume & timing)
  - **Non-ideal systems behavior:** uneven machine performance, other cluster users

Implementing Distributed ML Algorithms

- A number of ad-hoc or partial solutions, but sometimes lacking theoretical analysis
  - **Major barrier:** hard to analyze solutions because algorithm/systems sometimes not fully/transparently described in papers
  - **Possible solution:** a universal language and principles for design could facilitate theoretical analysis of existing and new solutions

- Let us look at some open-source platforms, which distributed ML algorithms can be implemented upon

Open-Source Platforms for Distributed ML

Modern Systems for Big ML

- Just now: data-, model-parallel ML algorithms for optimization, MCMC
- One could write distributed implementations from scratch
- Perhaps better to use an existing open source platform?

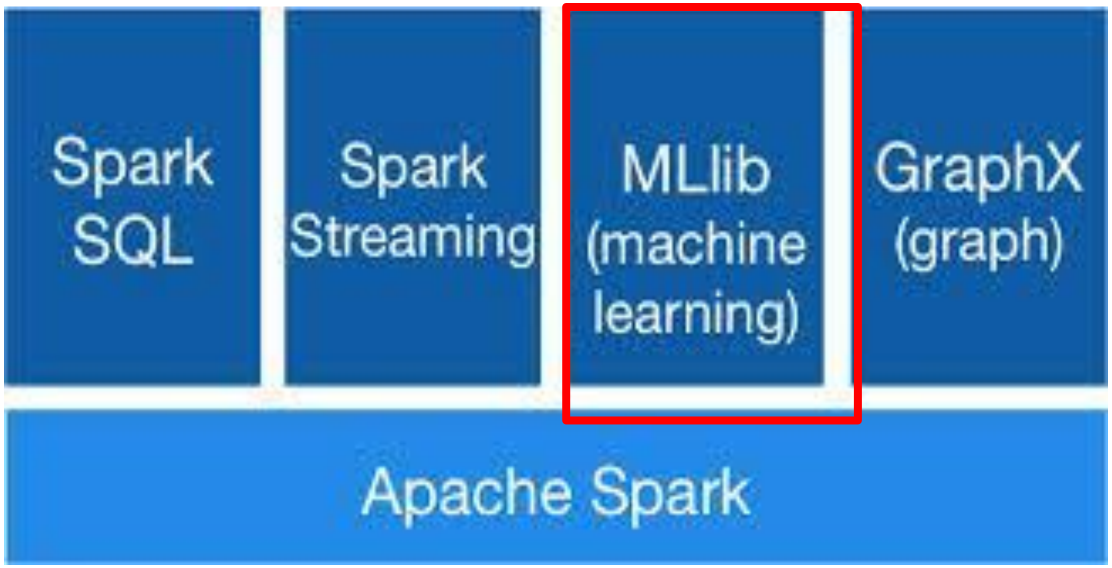
GraphLab

Spark

PETUUM

Spark Overview [Zaharia et al., 2010]

- General-purpose system for Big Data processing
  - Shell/interpreter for Matlab/R-like analytics
- MLlib = Spark's ready-to-run ML library
  - Implemented on Spark's API

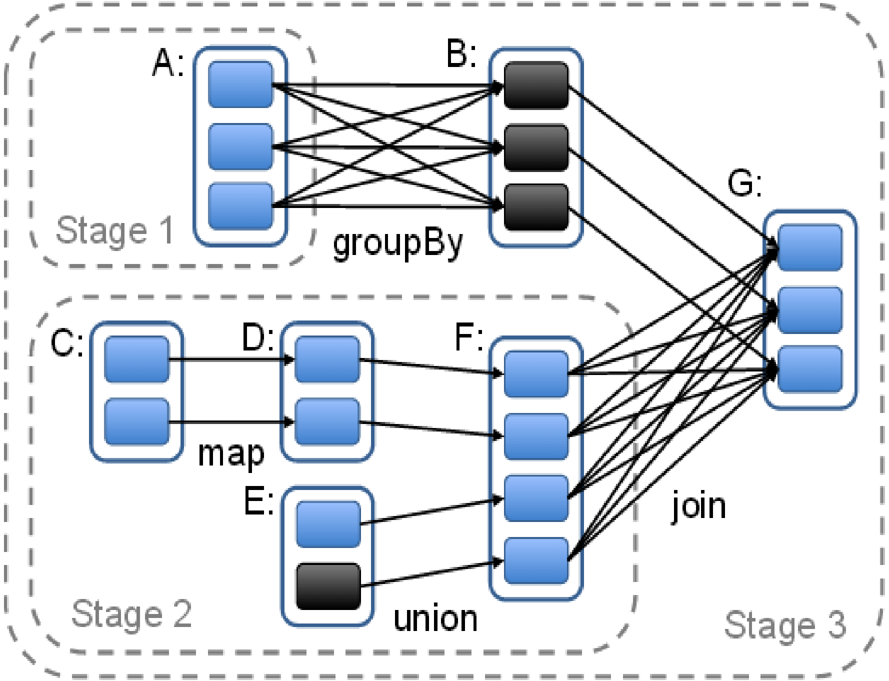


Spark Overview [Zaharia et al., 2010]

- MLlib algorithms (v1.4)
  - Classification and regression
    - linear models (SVMs, logistic regression, linear regression)
    - naive Bayes
    - decision trees
    - ensembles of trees (Random Forests and Gradient-Boosted Trees)
    - isotonic regression
  - Collaborative filtering
    - alternating least squares (ALS)
  - Clustering
    - k-means
    - Gaussian mixture
    - power iteration clustering (PIC)
    - latent Dirichlet allocation (LDA)
    - streaming k-means
  - Dimensionality reduction
    - singular value decomposition (SVD)
    - principal component analysis (PCA)

Spark Overview [Zaharia et al., 2010]

- Key feature: Resilient Distributed Datasets (RDDs)
  - Data processing = lineage graph of transforms
  - RDDs = nodes
  - Transforms = edges



Source: Zaharia et al. (2012)

Spark Overview [Zaharia et al., 2010]

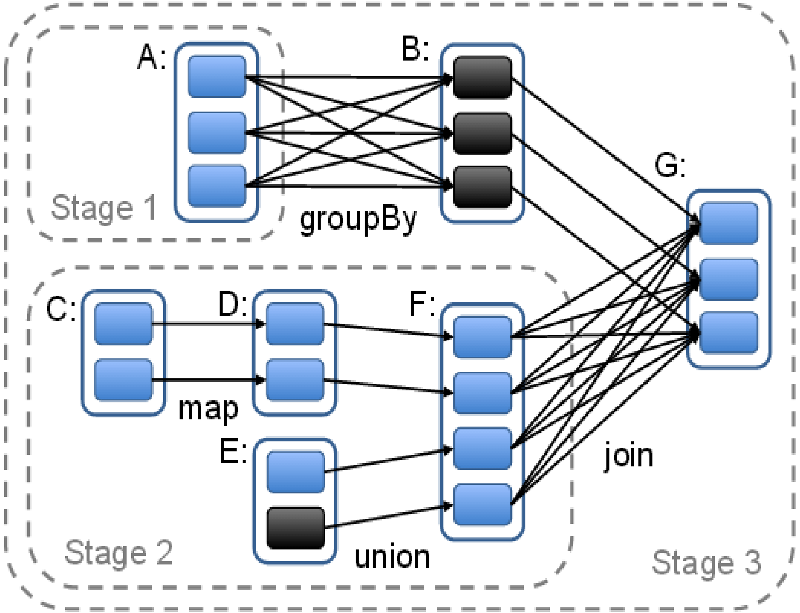
- RDD-based programming model
  - Similar in spirit to Hadoop Mapreduce
  - Functional style: manipulate RDDs via “transformations”, “actions”
    - E.g. map is a transformation, reduce is an action
  - Example: load file, count total number of characters

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

  - Other transformations and actions:
    - union(), intersection(), distinct()
    - count(), first(), take(), foreach()
    - …
  - Can specify if an RDD should be “persisted” to disk
    - Allows for faster recovery during cluster faults

Spark Overview [Zaharia et al., 2010]

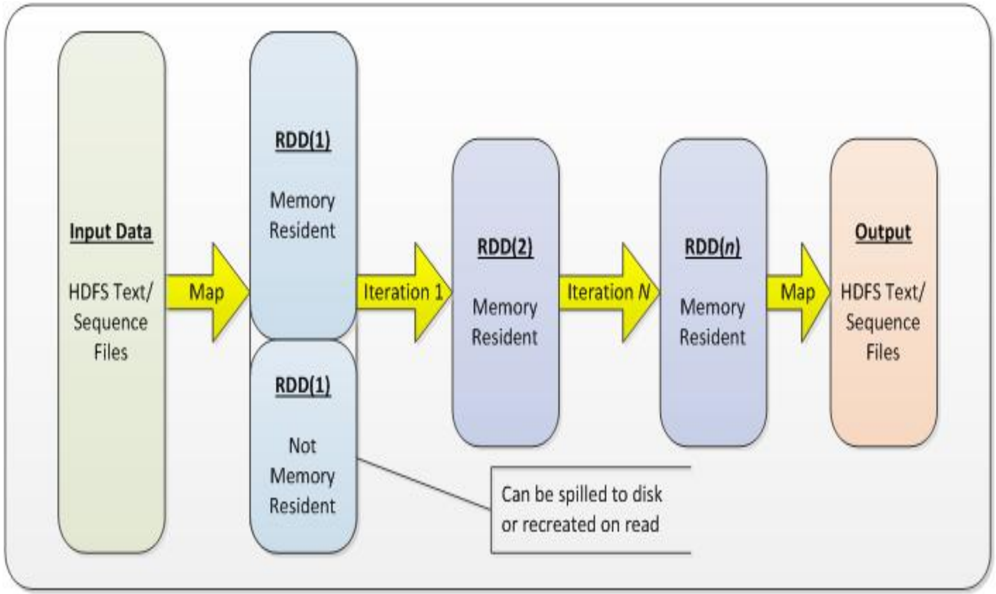
- Benefits of Spark:
  - Fault tolerant - RDDs immutable, just re-compute from lineage
  - Cacheable - keep some RDDs in RAM
    - Faster than Hadoop MR at iterative algorithms
  - Supports MapReduce as special case

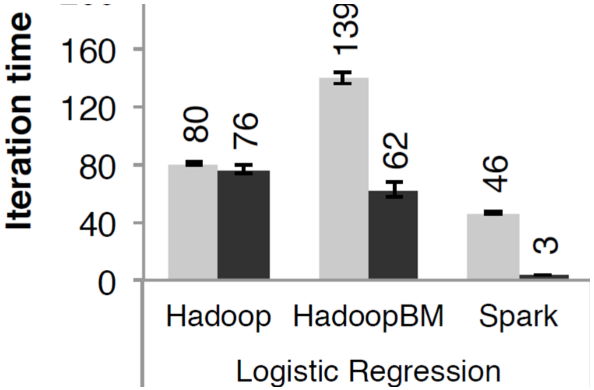


Source: Zaharia et al. (2012)

Spark: Faster MapR on Data-Parallel

- Spark's solution: **Resilient Distributed Datasets (RDDs)**
  - Input data → load as RDD → apply transforms → output result
  - RDD transforms strict superset of MapR
  - RDDs cached in memory, avoid disk I/O



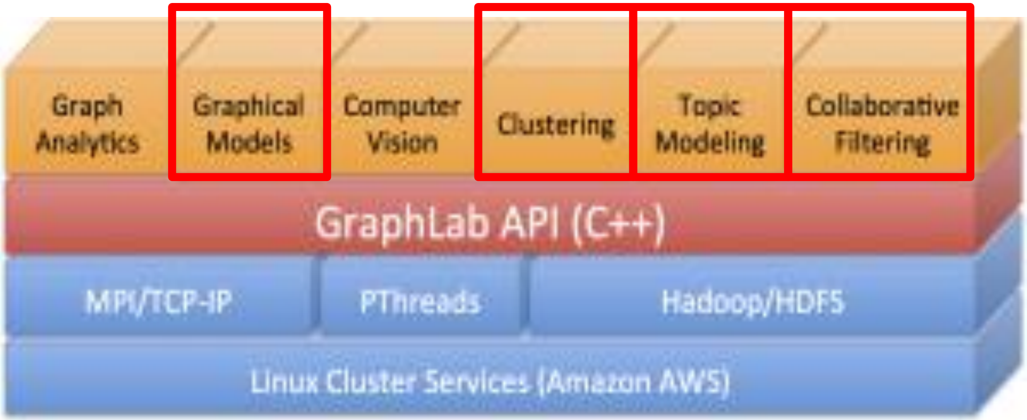


- **Spark ML library supports data-parallel ML algos, like Hadoop**
  - Spark and Hadoop: comparable first iter timings…
  - But Spark's later iters are much faster

Source: ebaytechblog.com

GraphLab Overview [Low et al., 2012]

- Known as “GraphLab PowerGraph v2.2”
  - Different from commercial software “GraphLab Create” by Dato.com, who formerly developed PowerGraph v2.2
- System for Graph Programming
  - Think of ML algos as graph algos
- Comes with ready-to-run “toolkits”
  - ML-centric toolkits: clustering, collaborative filtering, topic modeling, graphical models



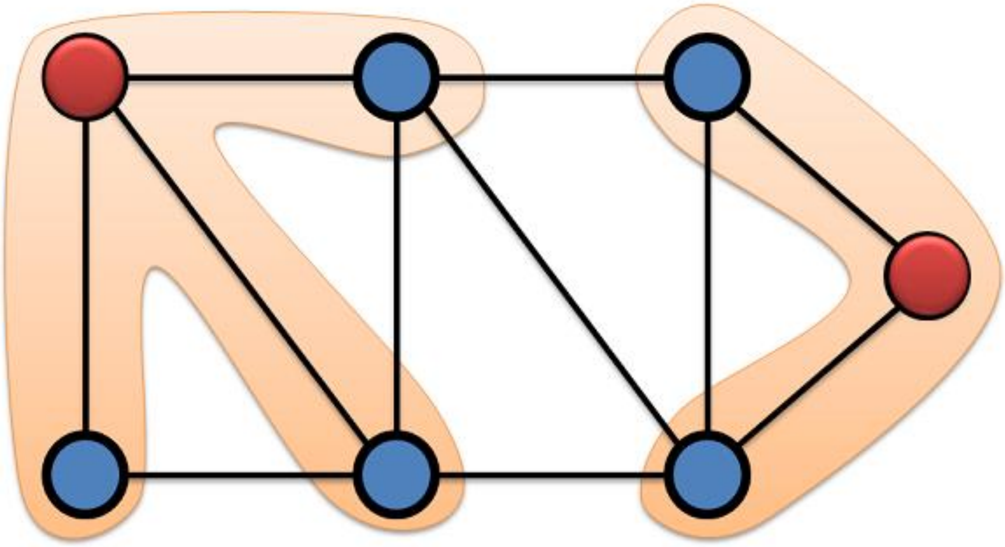
© Eric Xing @ CMU, 2015

GraphLab Overview [Low et al., 2012]

- ML-related toolkits
  - Clustering
    - K-means
    - Spectral
  - Collaborative Filtering
    - Matrix Factorization (including Non-negative, L1/L2-regularized)
  - Graphical Models
    - Factor graphs
    - Belief propagation algorithm
  - Topic Modeling
    - LDA
- Other toolkits available for computer vision, graph analytics, linear systems

GraphLab Overview [Low et al., 2012]

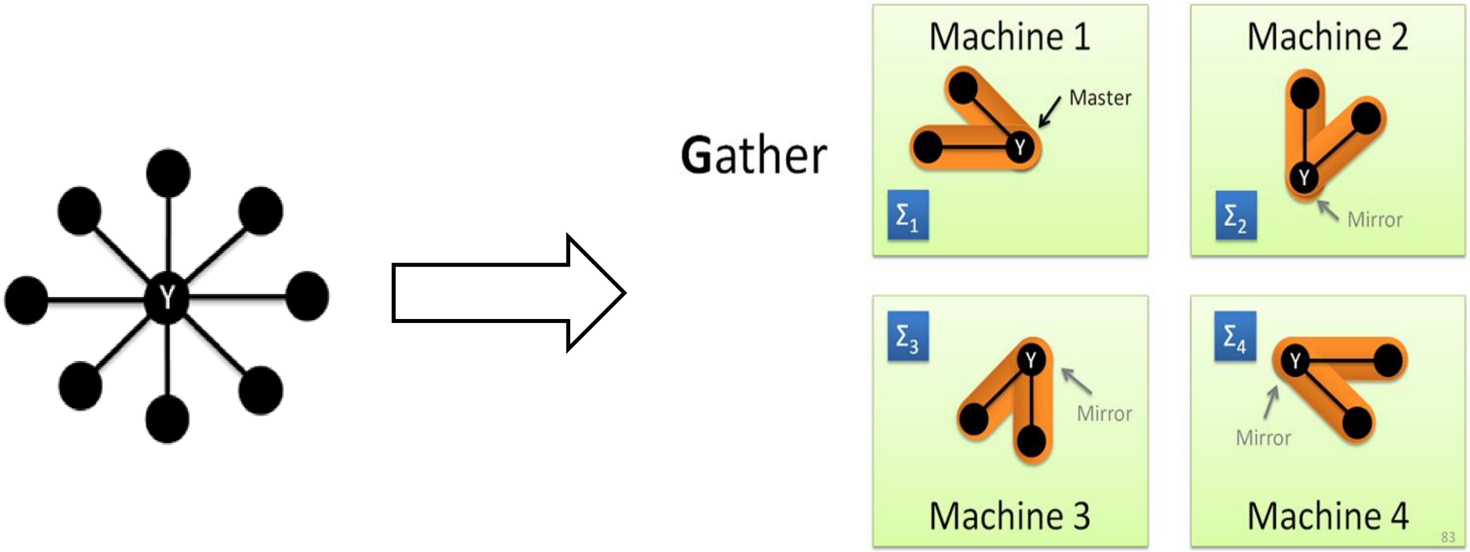
- Key feature: Gather-Apply-Scatter Programming Model
  - Write ML algos as vertex programs
  - Run vertex programs in parallel on each graph node
  - Graph nodes, edges can have data, parameters



Source: Gonzalez (2012)

GraphLab Overview [Low et al., 2012]

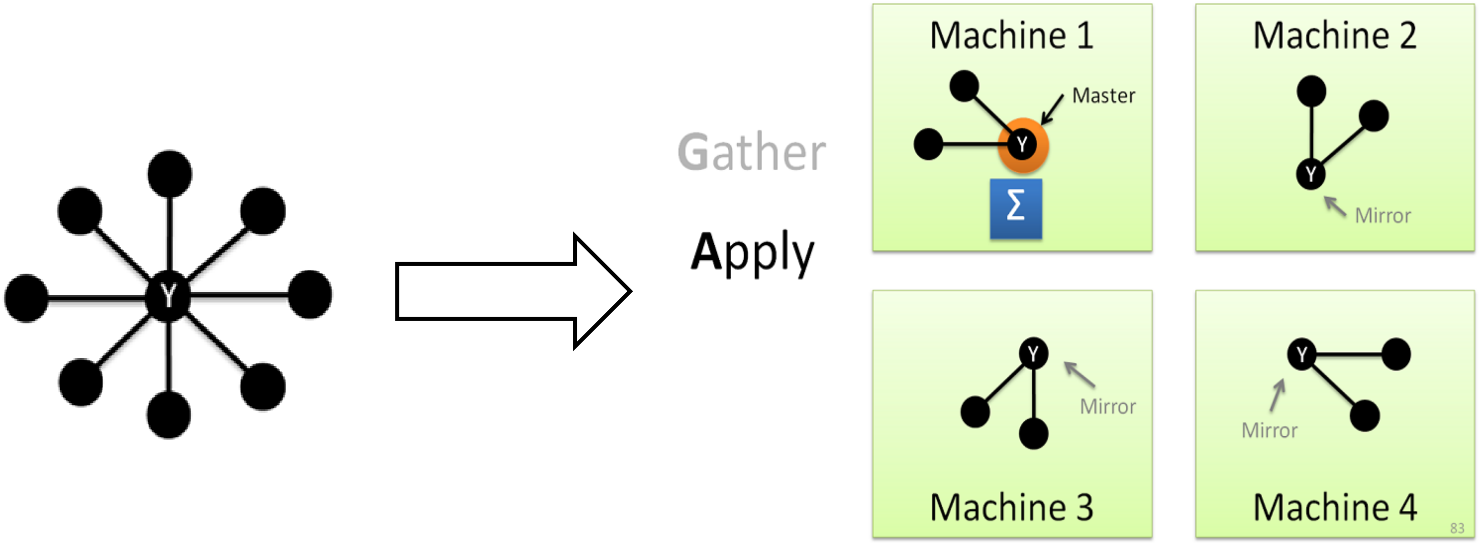
- Programming Model: GAS Vertex Programs
  - **1) Gather():** Accumulate data, params from my neighbors + edges
  - 2) Apply(): Transform output of Gather(), write to myself
  - 3) Scatter(): Transform output of Gather(), Apply(), write to my edges



Source: Gonzalez (2012)

GraphLab Overview [Low et al., 2012]

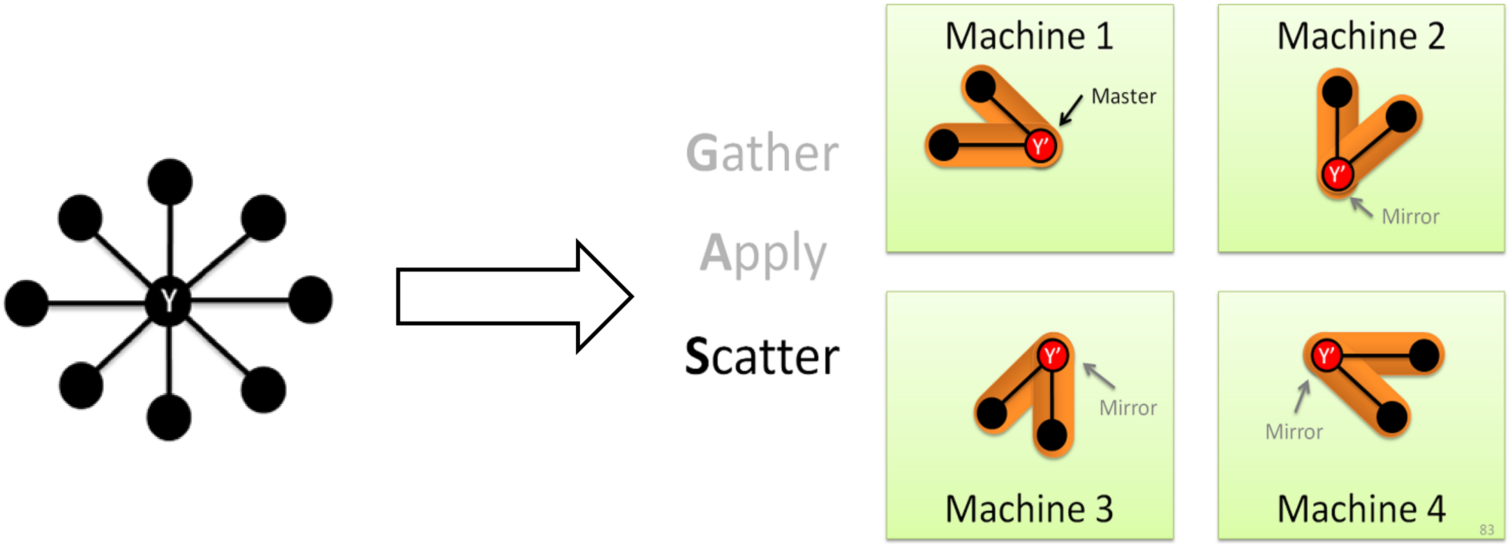
- Programming Model: GAS Vertex Programs
  - 1) Gather(): Accumulate data, params from my neighbors + edges
  - **2) Apply():** Transform output of Gather(), write to myself
  - 3) Scatter(): Transform output of Gather(), Apply(), write to my edges



Source: Gonzalez (2012)

GraphLab Overview [Low et al., 2012]

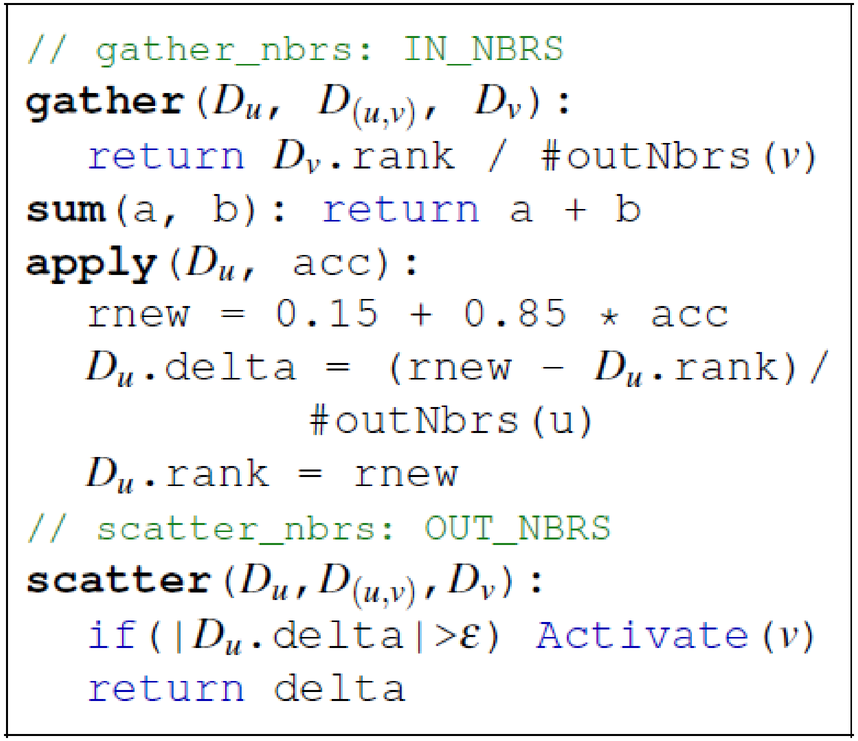
- Programming Model: GAS Vertex Programs
  - 1) Gather(): Accumulate data, params from my neighbors + edges
  - 2) Apply(): Transform output of Gather(), write to myself
  - **3) Scatter():** Transform output of Gather(), Apply(), write to my edges



Source: Gonzalez (2012)

GraphLab Overview [Low et al., 2012]

- Example GAS program: Pagerank
  - Programmer implements gather(), apply(), scatter() functions

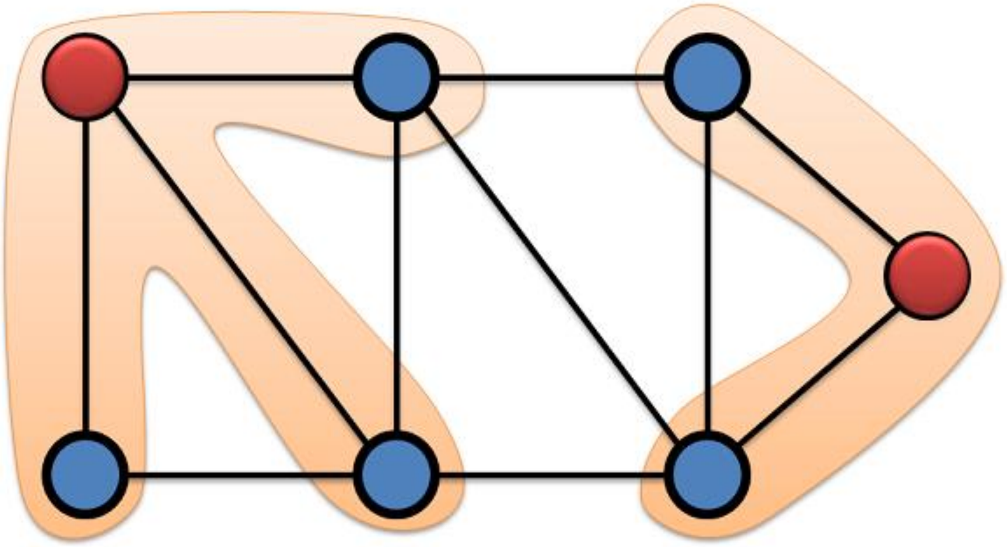


```
// gather_nbrs: IN_NBRS
gather($D_u$, $D_{(u,v)}$, $D_v$):
   return $D_v$.rank / #outNbrs($v$)
sum(a, b): return a + b
apply($D_u$, acc):
   rnew = 0.15 + 0.85 * acc
   $D_u$.delta = (rnew - $D_u$.rank)/
                 #outNbrs(u)
   $D_u$.rank = rnew
// scatter_nbrs: OUT_NBRS
scatter($D_u$,$D_{(u,v)}$,$D_v$):
   if(|$D_u$.delta|>$\varepsilon$) Activate($v$)
   return delta
```

Source: Gonzalez et al. (OSDI 2012)

GraphLab Overview [Low et al., 2012]

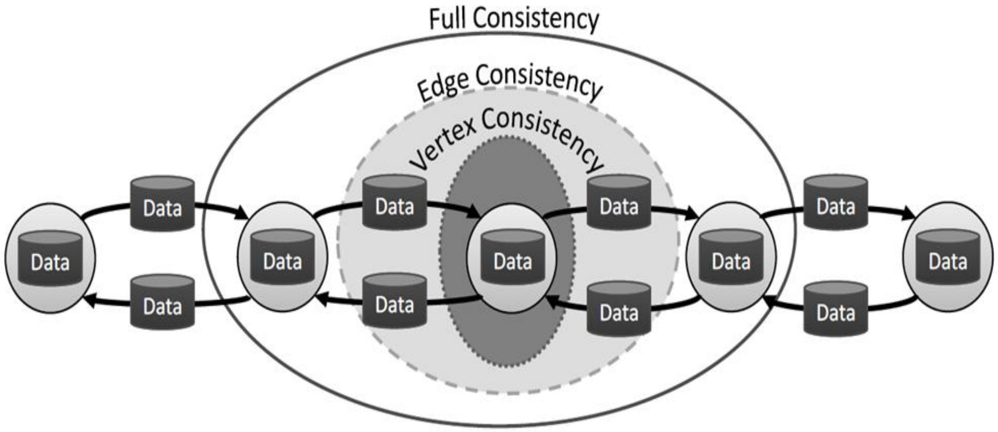
- Benefits of Graphlab
  - Supports asynchronous execution - fast, avoids straggler problems
  - Edge-cut partitioning - scales to large, power-law graphs
  - Graph-correctness - for ML, more fine-grained than MapR-correctness

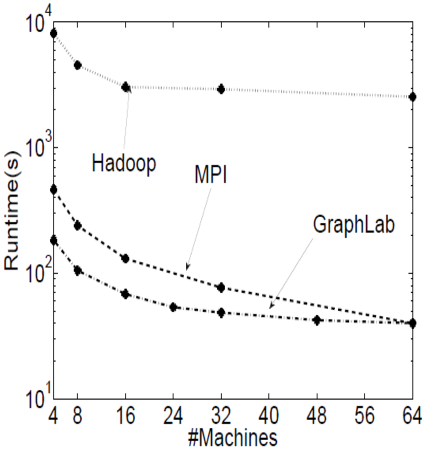


Source: Gonzalez (2012)

GraphLab: Model-Parallel via Graphs

- GraphLab **Graph consistency models**
  - Guide search for “ideal” model-parallel execution order
  - ML algo correct if input graph has all dependencies





- GraphLab supports asynchronous (no-waiting) execution
  - Correctness enforced by graph consistency model
  - Result: GraphLab graph-parallel ML much faster than Hadoop

Source: Low et al. (2010)

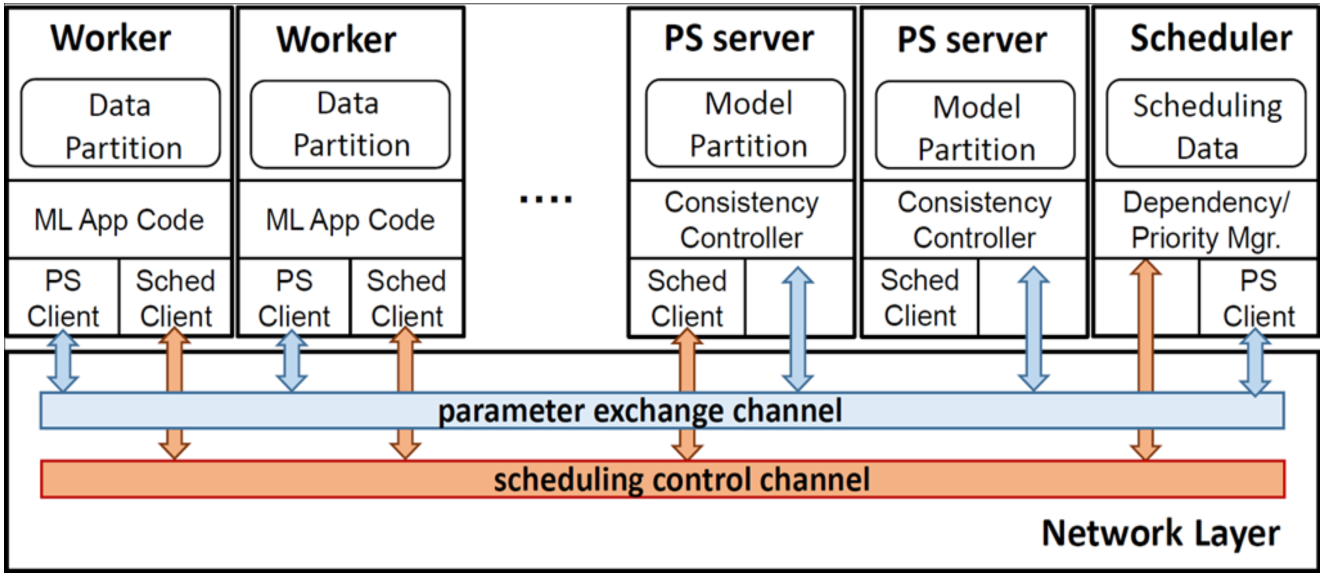
PETUUM

A New Framework for Large Scale Parallel Machine Learning

(Petuum.org)

Petuum Overview [Xing et al., 2015]

- Key modules
  - **Key-value store** (Parameter Server) for data-parallel ML algos
  - **Scheduler** for model-parallel ML algos
- Program ML algos in iterative-convergent style
  - ML algo = (1) write update equations + (2) iterate eqns via schedule

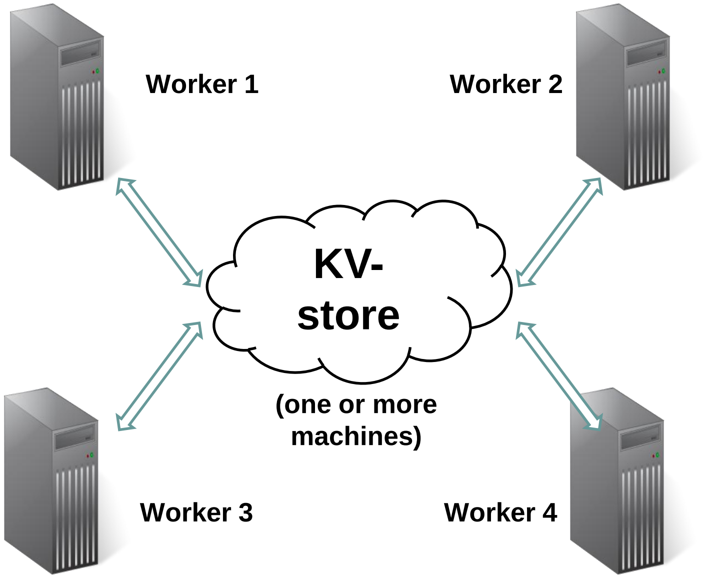


Petuum Overview [Xing et al., 2015]

- ML Library (Petuum v1.1):
  - Topic Modeling
    - LDA
    - MedLDA (supervised topic models)
  - Deep Learning
    - Fully-connected DNN
    - Convolutional Neural Network
  - Matrix Factorization
    - Least-squares Collaborative Filtering (with regularization)
    - Non-negative Matrix Factorization
    - Sparse Coding
  - Regression
    - Lasso Regression
  - Metric Learning
    - Distance Metric Learning
  - Clustering
    - K-means
  - Classification
    - Random Forest
    - Logistic Regression and SVM
    - Multi-class Logistic Regression

Petuum Overview [Xing et al., 2015]

- Key-Value store (Parameter Server)
  - Enables data-parallelism
  - A type of Distributed Shared Memory (DSM)
    - Model parameters globally shared across workers
  - Programming: replace local variables with PS calls



Single Machine

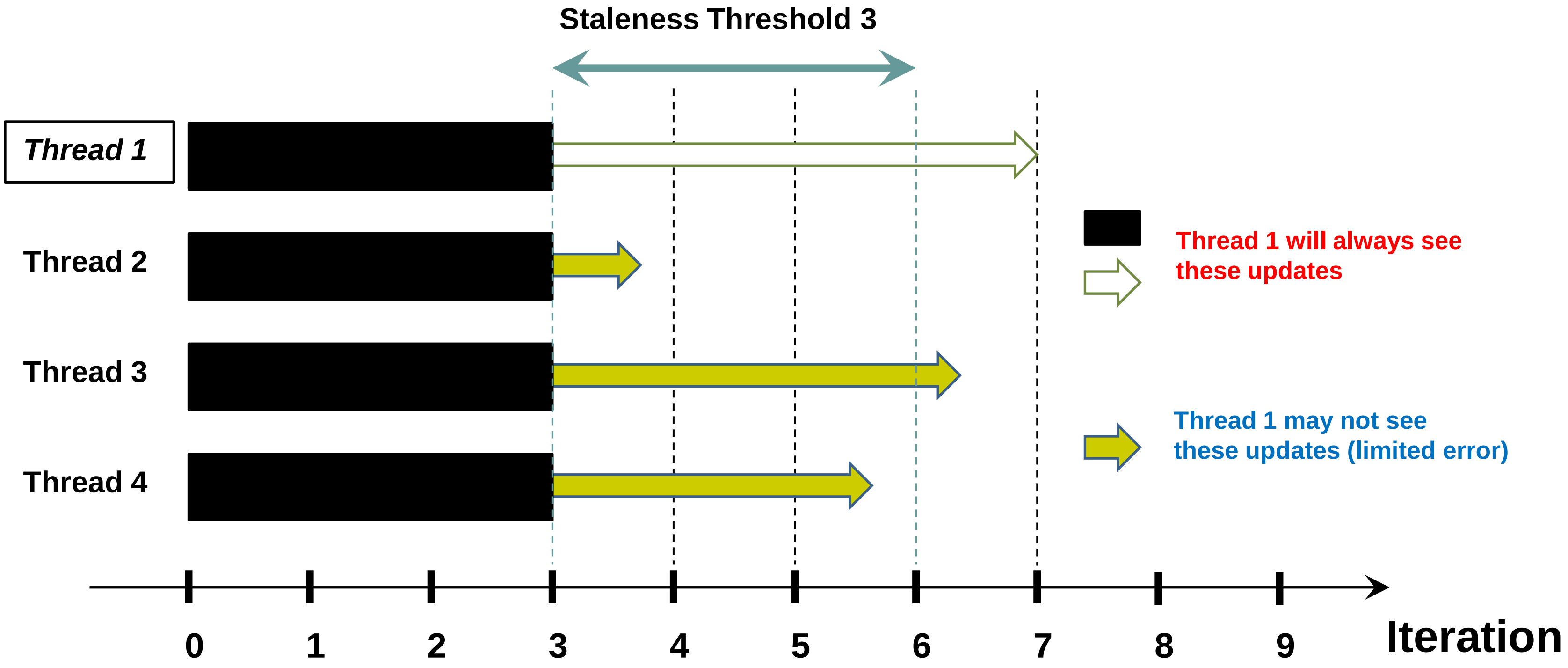
```
ProcessDataPoint(i) {
  for j = 1 to M {
    old = model[j]
    delta = f(model,data(i))
    model[j] += delta
  }
}
```

Distributed with PS

```
ProcessDataPoint(i) {
  for j = 1 to M {
    old = PS.read(model,j)
    delta = f(model,data(i))
    PS.inc(model,j,delta)
  }
}
```

Petuum Overview [Xing et al., 2015]

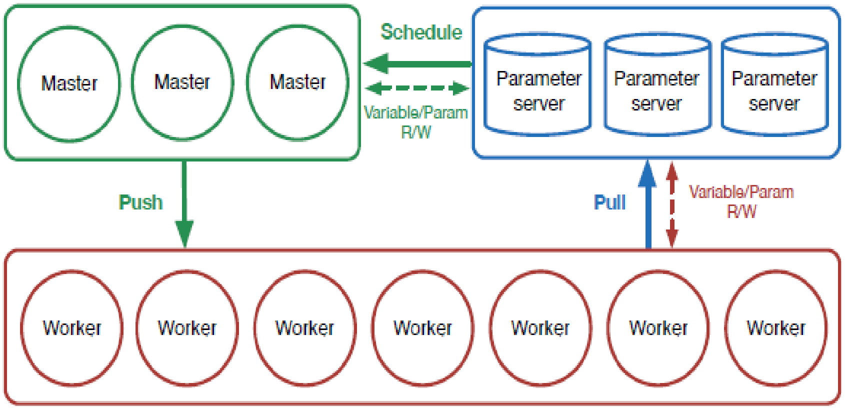
- Key-Value store features:
  - ML-tailored consistency model: Stale Synchronous Parallel (SSP)
  - Asynchronous-like speed
  - Bulk Synchronous Parallel-like correctness guarantees for ML

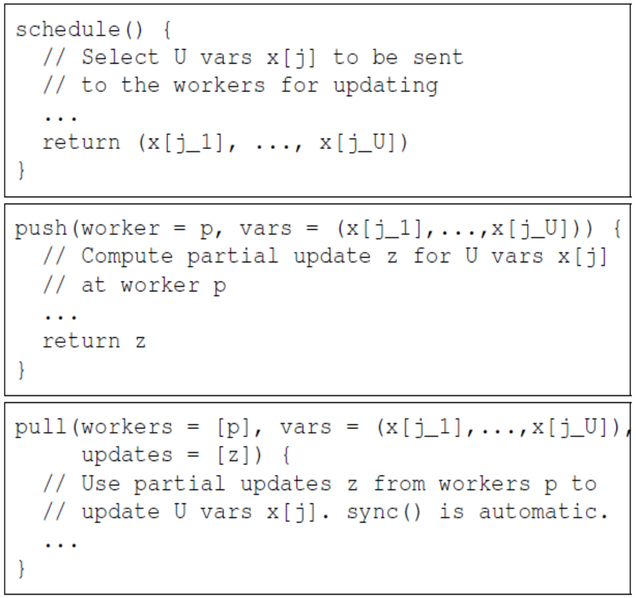

Staleness Threshold 3
Thread 1
Thread 2
Thread 3
Thread 4
Thread 1 will always see these updates
Thread 1 may not see these updates (limited error)
0
1
2
3
4
5
6
7
8
9
Iteration

© Eric Xing @ CMU, 2015

Petuum Overview [Xing et al., 2015]

- Scheduler
  - Enables correct model-parallelism
  - Can analyze ML model structure for best execution order
  - Programming: schedule(), push(), pull() abstraction





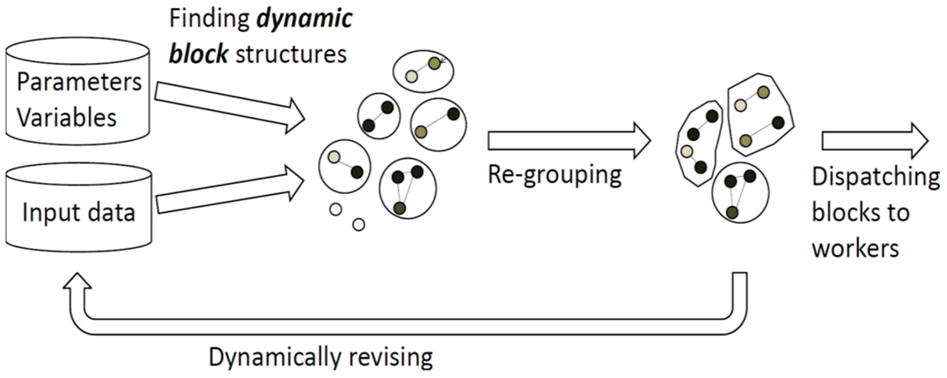
```
schedule() {
  // Select U vars x[j] to be sent
  // to the workers for updating
  ...
  return (x[j_1], ..., x[j_U])
}
```

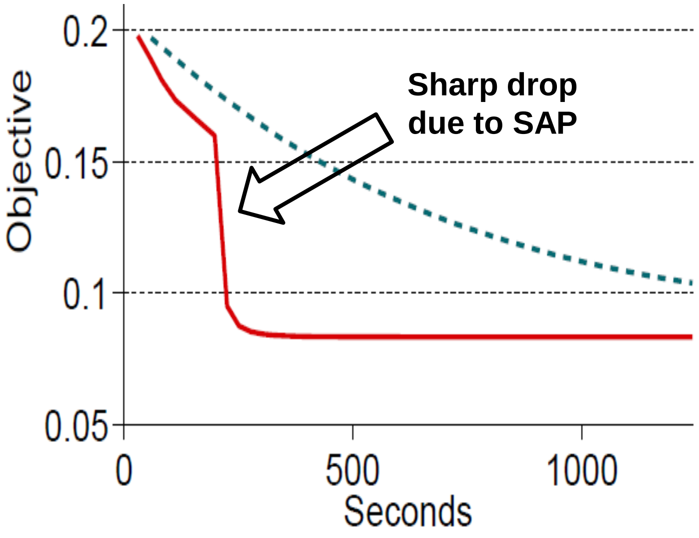
```
push(worker = p, vars = (x[j_1],...,x[j_U])) {
  // Compute partial update z for U vars x[j]
  // at worker p
  ...
  return z
}
```

```
pull(workers = [p], vars = (x[j_1],...,x[j_U]),
     updates = [z]) {
  // Use partial updates z from workers p to
  // update U vars x[j]. sync() is automatic.
  ...
}
```

Petuum Overview [Xing et al., 2015]

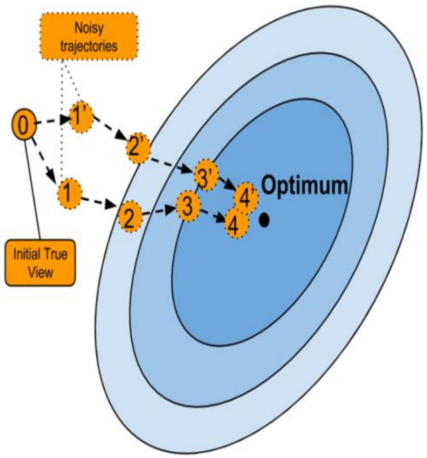
- Scheduler benefits:
  - ML scheduling engine: Structure-Aware Parallelization (SAP)
  - Scheduled ML algos require less computation to finish

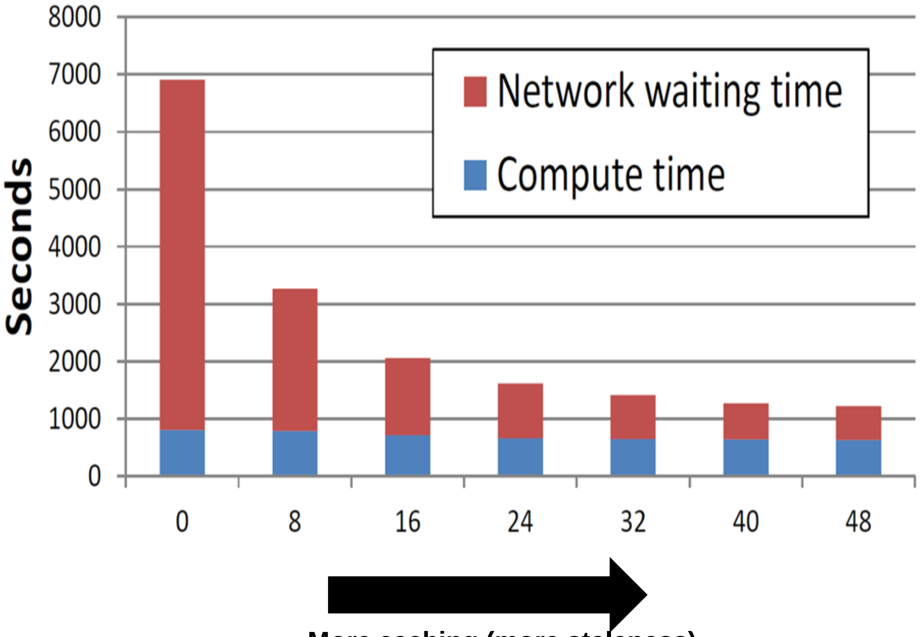

Parameters
Variables
Input data
Finding ***dynamic block*** structures
Re-grouping
Dispatching blocks to workers
Dynamically revising



Petuum: ML props = 1st-class citizen

- Error tolerance via Stale Sync Parallel KV-store
  - System Insight 1: ML algos bottleneck on network comms
  - System Insight 2: More caching => less comms => faster execution

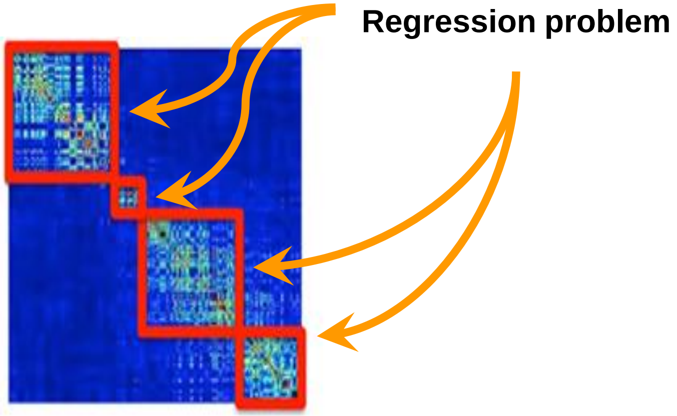


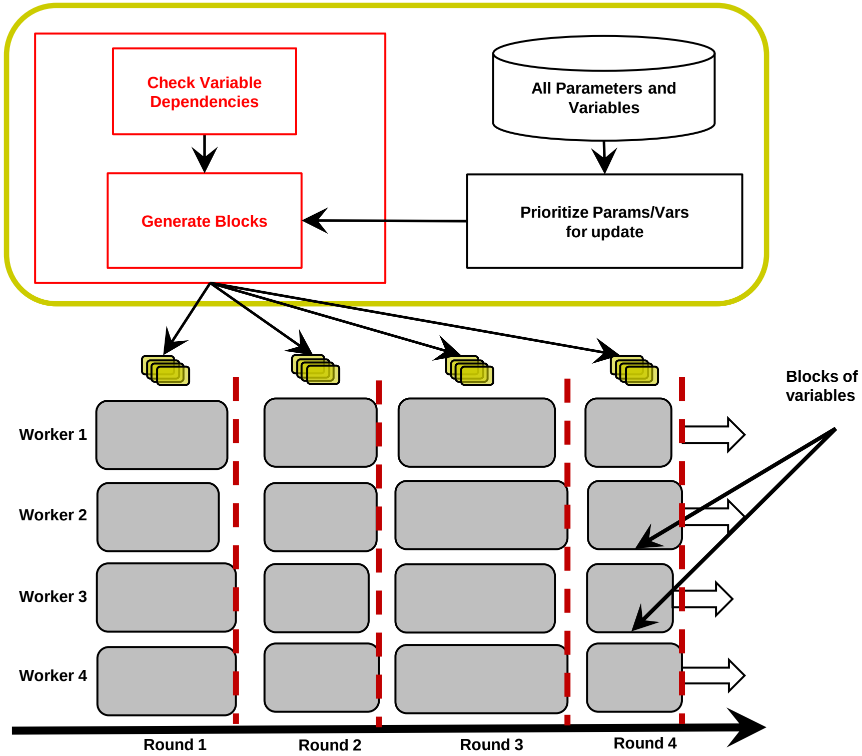


More caching (more staleness)

Petuum:
ML props = 1st-class citizen

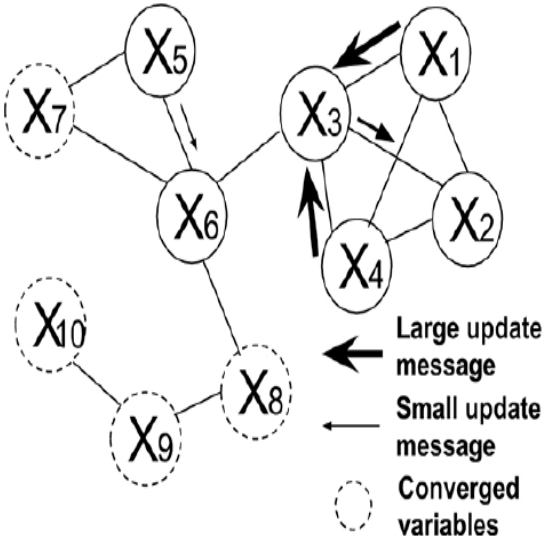
- Harness Block dependency structure via Scheduler
  - System Insight 1: Pipeline scheduler to hide latency
  - System Insight 2: Load-balance blocks to prevent stragglers

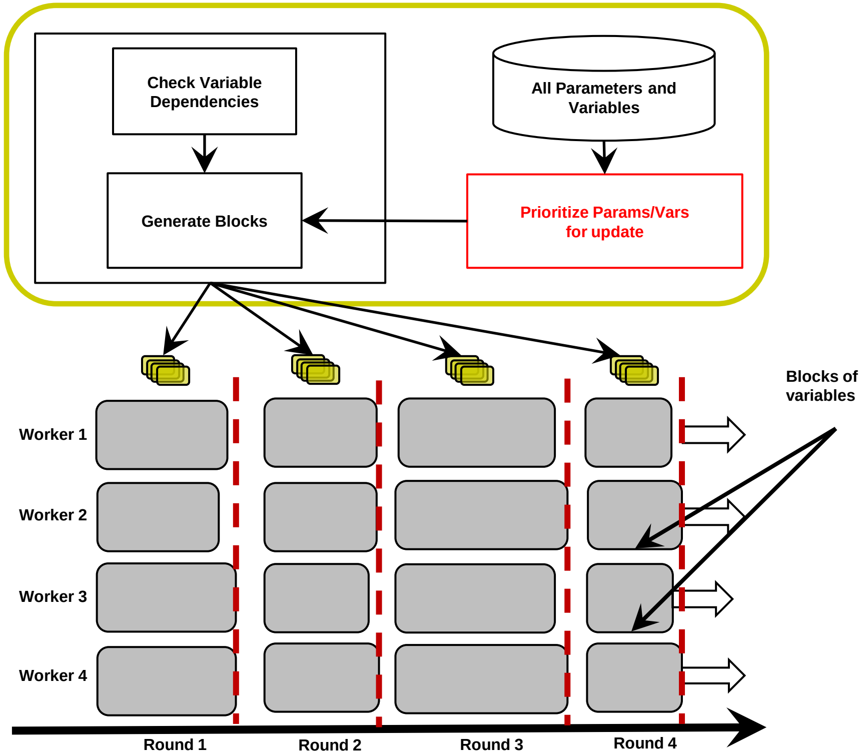




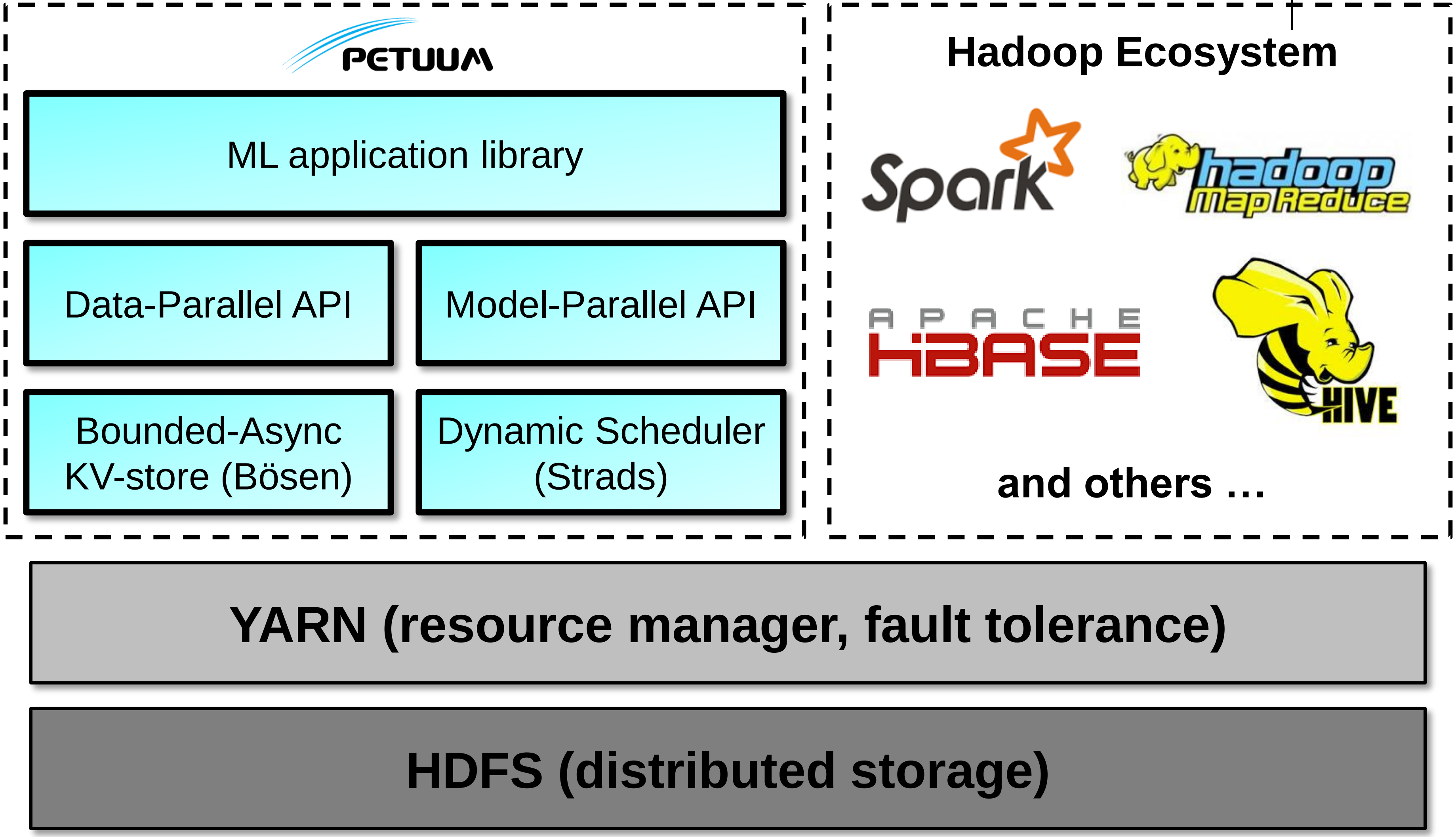
Petuum: ML props = 1st-class citizen

- Exploit Uneven Convergence via Prioritizer
  - System Insight 1: Prioritize small # of vars => fewer deps to check
  - System Insight 2: Lowers computational cost of Scheduling





Petuum Architecture and Hadoop Ecosystem Integration



ML application library

Data-Parallel API

Model-Parallel API

Bounded-Async KV-store (Bösen)

Dynamic Scheduler (Strads)

Hadoop Ecosystem

and others …

YARN (resource manager, fault tolerance)

HDFS (distributed storage)

ML Programming Interface: Needs and Considerations

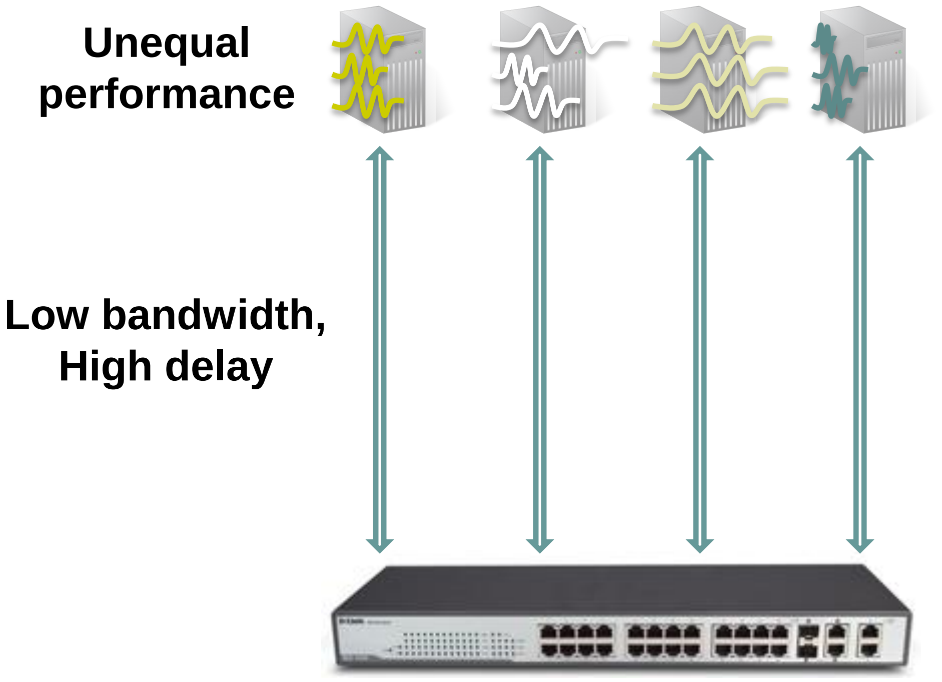
- An ideal ML programming interface should make it easy to write correct data-parallel, model-parallel ML programs

- What can be abstracted away?
  - Abstract away inter-worker communication/synchronization:
    - Automatic consistency models; bandwidth management through distributed shared memory
  - Abstract scheduling away from update equations:
    - Easy to change scheduling strategy, or use dynamic schedules
  - Abstract away worker management:
    - Let ML system decide optimal number and configuration of workers
  - Ideally, reduce programmer burden to just 3 things:
    - Declare model, write updates, write schedule

Systems, Architectures for Distributed ML

There Is No Ideal Distributed System!

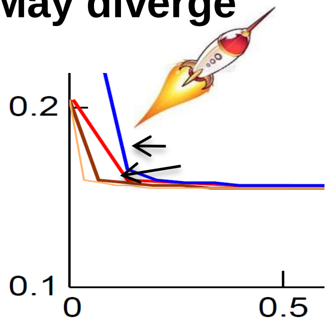
- Not quite that easy…
- **Two distributed challenges:**
  - Networks are slow
  - “Identical” machines rarely perform equally



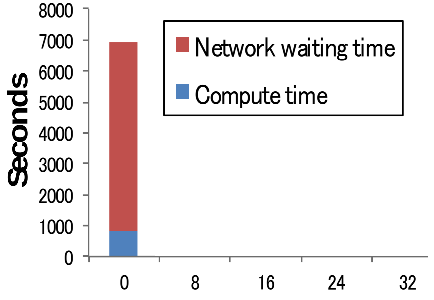
Unequal performance

Low bandwidth, High delay

Async execution: May diverge



BSP execution: Long sync time



Issue: How to approach distributed systems?

- Idealist view
  - Start with simplified view of distributed systems; develop elaborate theory
- Issues being explored:
  - Information theoretic lower bounds for communication **[Zhang et al. 2013]**
  - Provably correct distributed architectures, with mild assumptions **[Langford et al. 2009, Duchi and Agarwal 2011]**
- How can we build practical solutions using these ideas?

- Pragmatist view
  - Start with real-world, complex distributed systems, and develop a combination of theoretical guarantees and empirical evidence
- Issues being explored:
  - Fault tolerance and recovery **[Zaharia et al. 2012, Spark, Li et al. 2014]**
  - Impact of stragglers and delays on inference, and robust solutions **[Ho et al. 2013, Dai et al. 2015, Petuum, Li et al. 2014]**
  - Scheduling of inference computations for massive speedups **[Low et al. 2012, GraphLab, Kim et al. 2014, Petuum]**
- How can we connect these phenomena to theoretical inference correctness and speed?

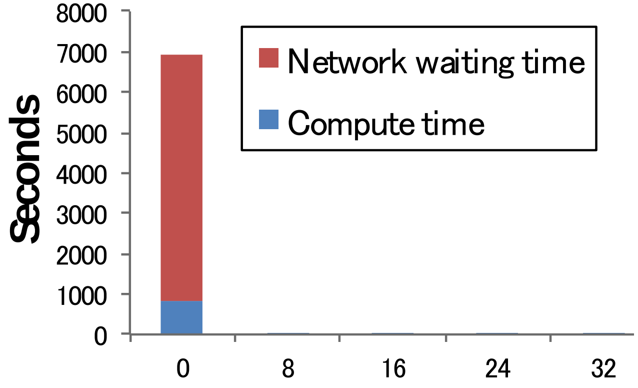
Why need new Big ML systems?

MLer’s view

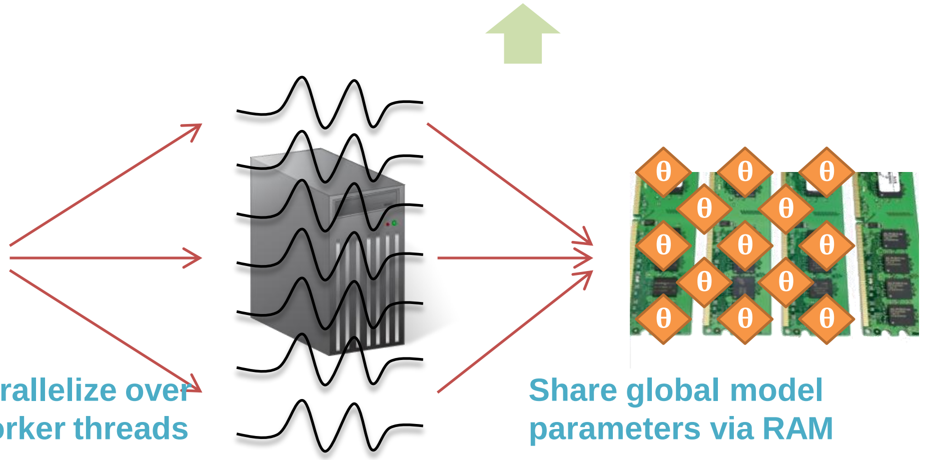
- Focus on
  - Correctness
  - fewer iteration to converge,
- but assuming an ideal system, e.g.,
  - zero-cost sync,
  - uniform local progress

Compute vs Network

LDA 32 machines (256 cores)



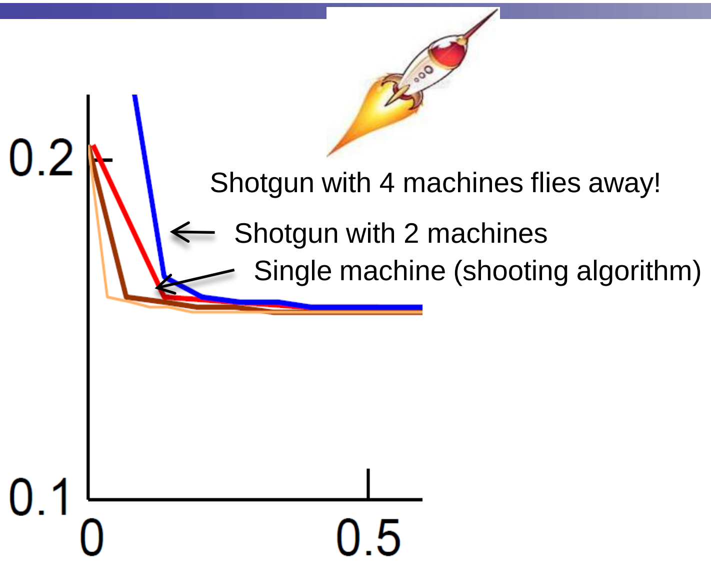
```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```



Parallelize over worker threads

Share global model parameters via RAM

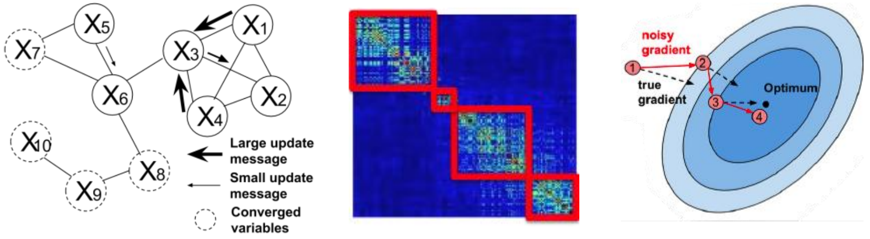
Why need new Big ML systems?



Systems View:

- Focus on
  - high iteration throughput (more iter per sec)
  - strong fault-tolerant atomic operations,
- but assume ML algo is a black box
  - ML algos “still work” under different execution models
  - “easy to rewrite” in chosen abstraction

Agonistic of ML properties and objectives **in system design**

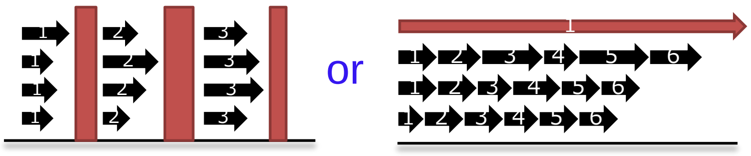


Non-uniform convergence

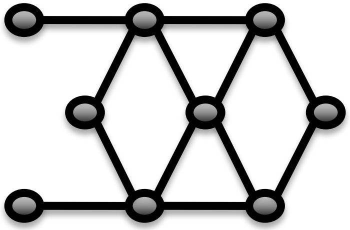
Dynamic structures

Error tolerance

Synchronization model



Programming model



Why need new Big ML systems?

MLer's view

- Focus on
  - Correctness
  - fewer iteration to converge,
- but assuming an ideal system, e.g.,
  - zero-cost sync,
  - uniform local progress

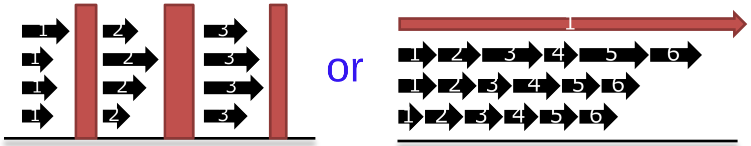
```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```

Oversimplify systems issues

- **need machines to perform consistently**
- **need lots of synchronization**
- **or even try not to communicate at all**

Systems View:

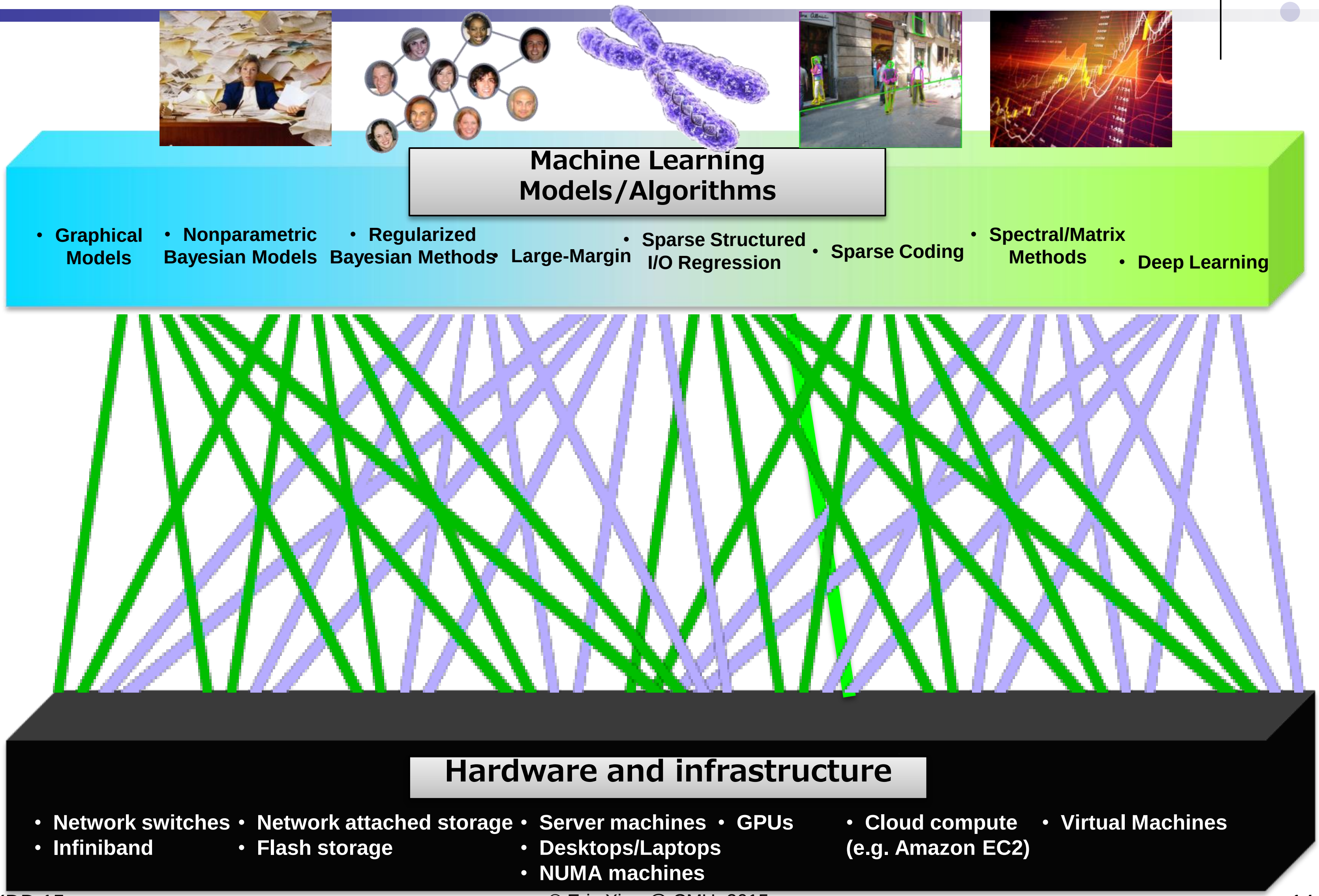
- Focus on
  - high iteration throughput (more iter per sec)
  - strong fault-tolerant atomic operations,
- but assume ML algo is a black box
  - ML algos "still work" under different execution models
  - "easy to rewrite" in chosen abstraction



Oversimplify ML issues and/or ignore ML opportunities

- **ML algos "just work" without proof**
- **Conversion of ML algos across different program models (graph programs, RDD) is easy**

Solution:



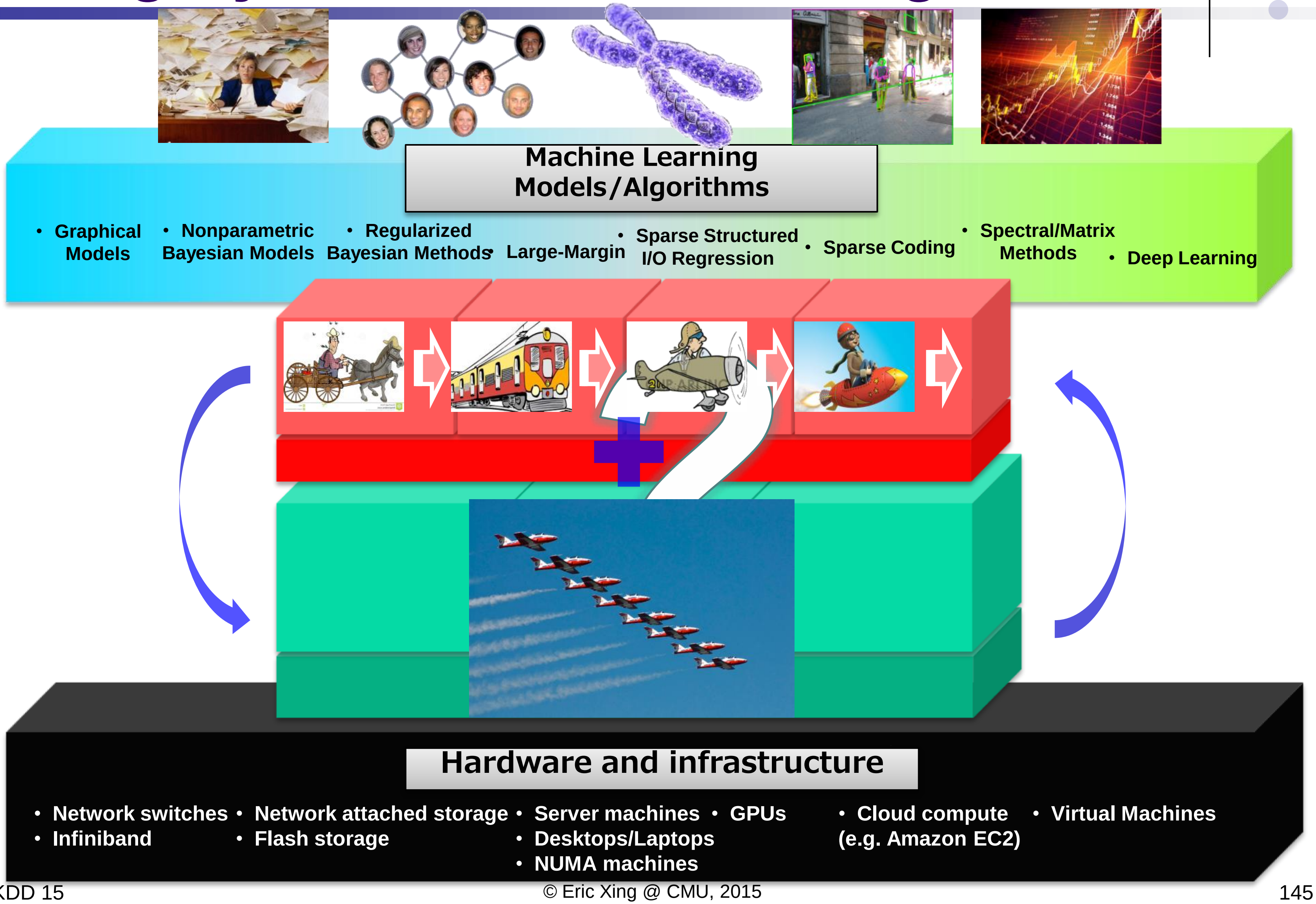
Machine Learning Models/Algorithms

- Graphical Models
- Nonparametric Bayesian Models
- Regularized Bayesian Methods
- Large-Margin
- Sparse Structured I/O Regression
- Sparse Coding
- Spectral/Matrix Methods
- Deep Learning

Hardware and infrastructure

- Network switches
- Infiniband
- Network attached storage
- Flash storage
- Server machines
- Desktops/Laptops
- NUMA machines
- GPUs
- Cloud compute (e.g. Amazon EC2)
- Virtual Machines

Solution:
An Alg/Sys INTERFACE for Big ML



Machine Learning Models/Algorithms

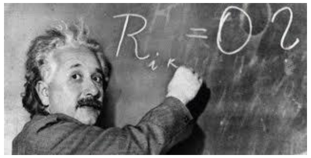
- Graphical Models
- Nonparametric Bayesian Models
- Regularized Bayesian Methods
- Large-Margin
- Sparse Structured I/O Regression
- Sparse Coding
- Spectral/Matrix Methods
- Deep Learning

Hardware and infrastructure

- Network switches
- Infiniband
- Network attached storage
- Flash storage
- Server machines
- Desktops/Laptops
- NUMA machines
- GPUs
- Cloud compute (e.g. Amazon EC2)
- Virtual Machines

The Big-ML "Stack" - More than just software

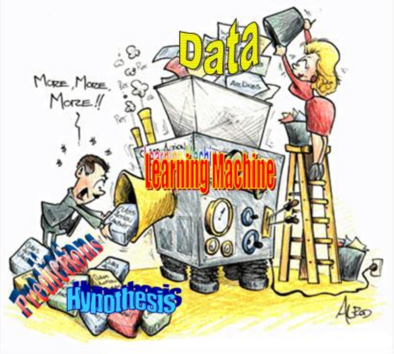
Theory: Degree of parallelism, convergence analysis, sub-sample complexity …

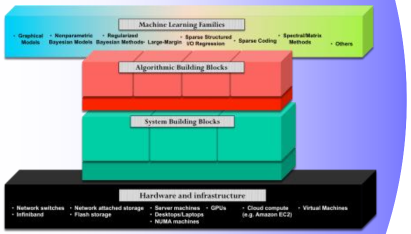


Representation: Compact and informative features

Model: Generic building blocks: loss functions, structures, constraints, priors …

Algorithm: Parallelizable and stochastic MCMC, VI, Opt, Spectrum …



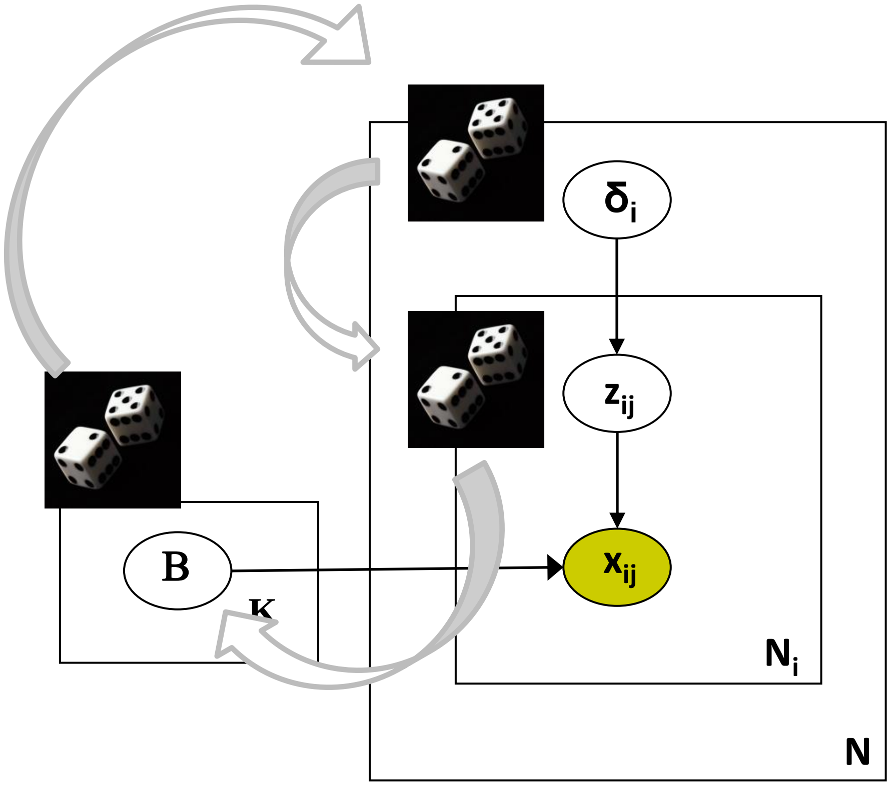

Machine Learning Families
· Graphical Models
· Nonparametric Bayesian Models
· Regularized Bayesian Methods
· Large-Margin
· Sparse Structured I/O Regression
· Sparse Coding
· Spectral/Matrix Methods
· Others
Algorithmic Building Blocks
System Building Blocks
Hardware and infrastructure
· Network switches
· Infiniband
· Network attached storage
· Flash storage
· Server machines
· Desktops/Laptops
· NUMA machines
· GPUs
· Cloud compute (e.g. Amazon EC2)
· Virtual Machines

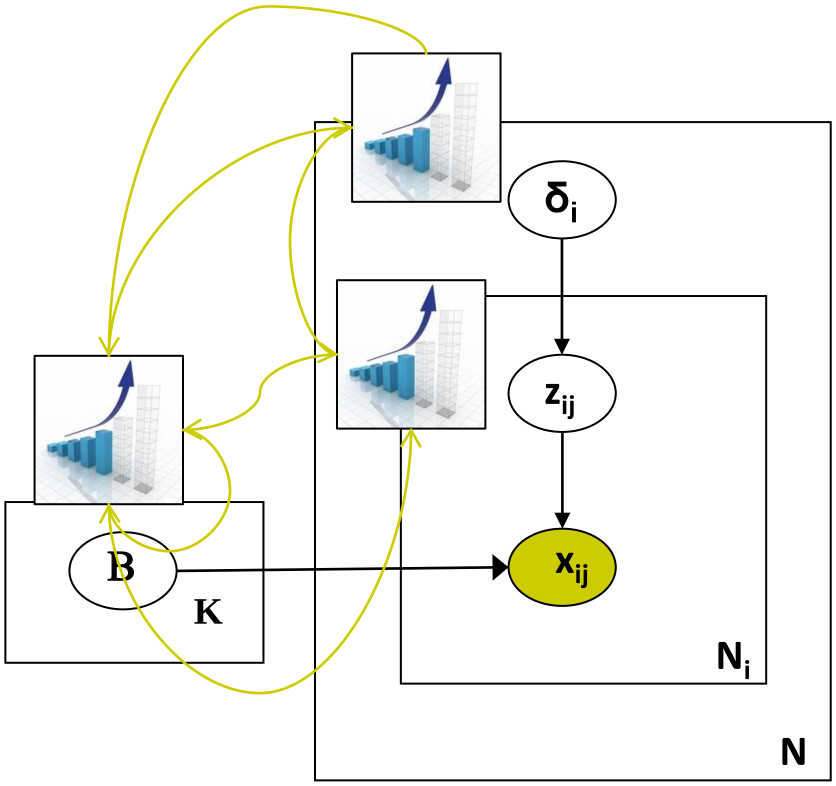
Programming model & Interface: High: Matlab/R
Medium: C/JAVA
Low: MPI

System: Distributed architecture: DFS, KV-store, task scheduler…

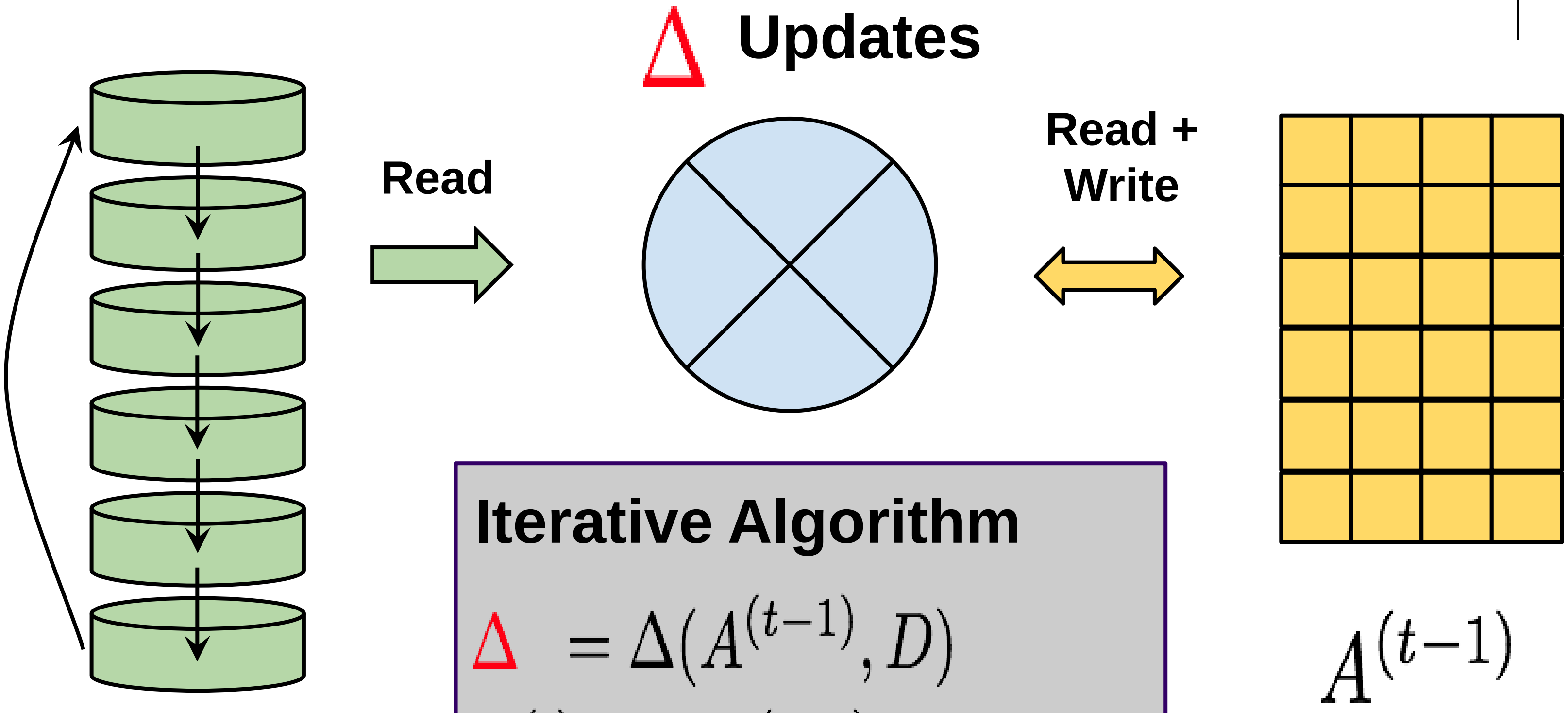
Hardware: GPU, flash storage, cloud …

ML algorithms are Iterative-Convergent





A General Picture of ML Iterative-Convergent Algorithms



D

Data

Iterative Algorithm

$$\Delta = \Delta(A^{(t-1)}, D)$$

$$A^{(t)} = F(A^{(t-1)}, \Delta)$$

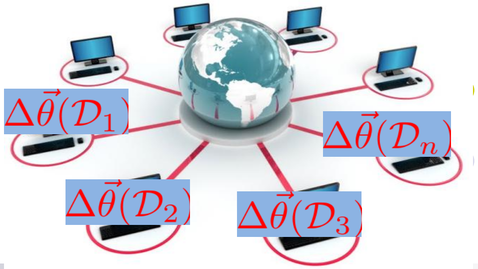
$F()$ Aggregate + Transform

Δ Intermediate Updates

$A^{(t-1)}$

Model Parameters at iteration (t-1)

Issues with Hadoop and I-C ML Algorithms?



$\Delta\vec{\theta}(\mathcal{D}_1)$
$\Delta\vec{\theta}(\mathcal{D}_n)$
$\Delta\vec{\theta}(\mathcal{D}_2)$
$\Delta\vec{\theta}(\mathcal{D}_3)$

$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$

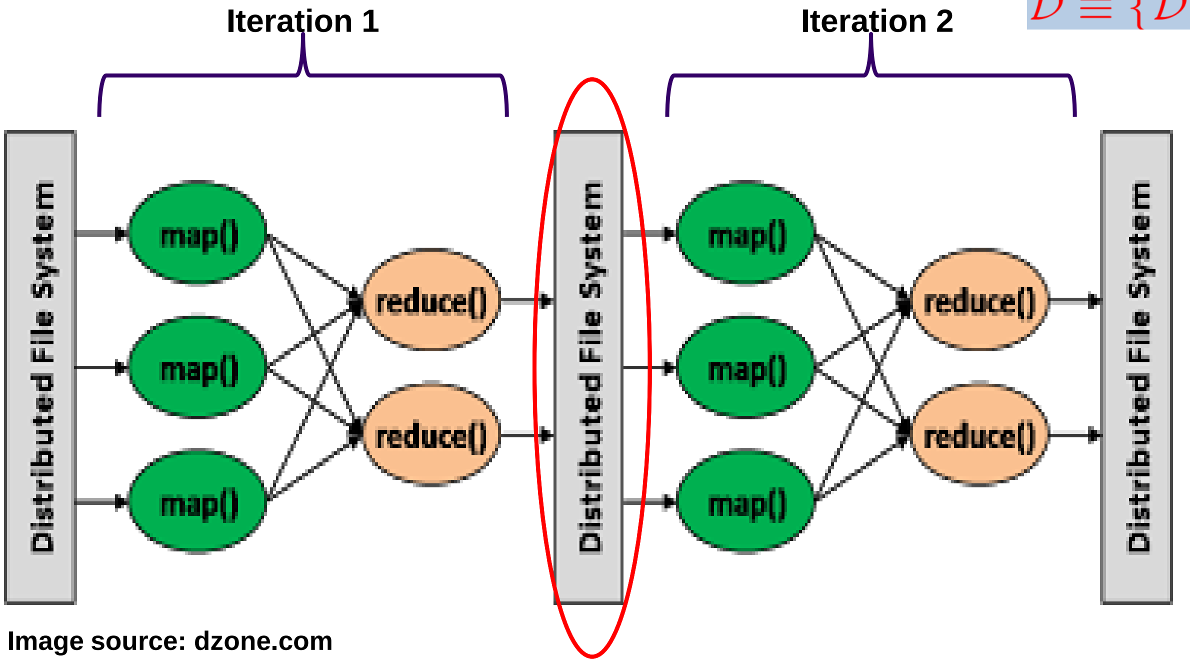


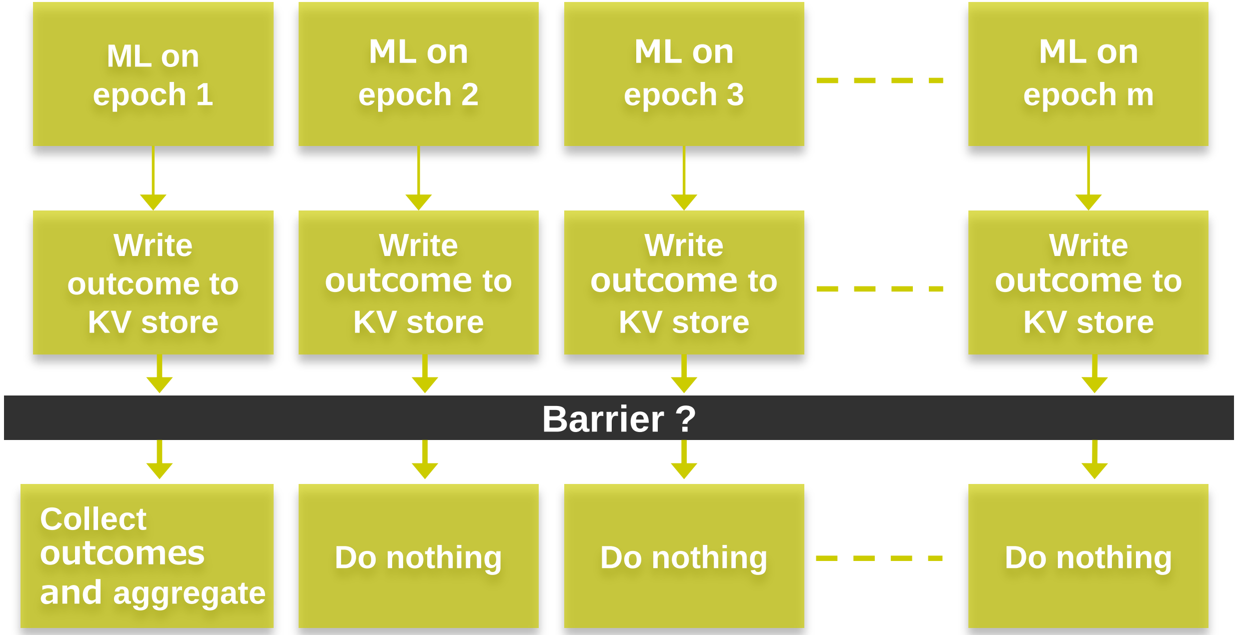
Image source: dzone.com

HDFS Bottleneck

Naïve MapReduce not best for ML

- Hadoop can execute iterative-convergent, data-parallel ML...
  - map() to distribute data samples i, compute update $\Delta(D_i)$
  - reduce() to combine updates $\Delta(D_i)$
  - Iterative ML algo = repeat map()+reduce() again and again
- But reduce() writes to HDFS before starting next iteration's map() - very slow iterations!

Good Parallelization Strategy is important



ML on epoch 1

ML on epoch 2

ML on epoch 3

ML on epoch m

Write outcome to KV store

Write outcome to KV store

Write outcome to KV store

Write outcome to KV store

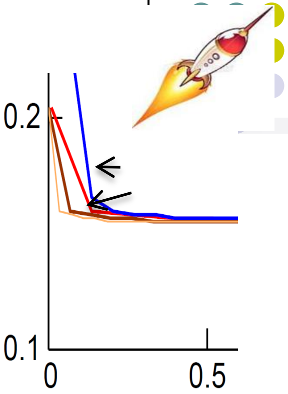
Barrier ?

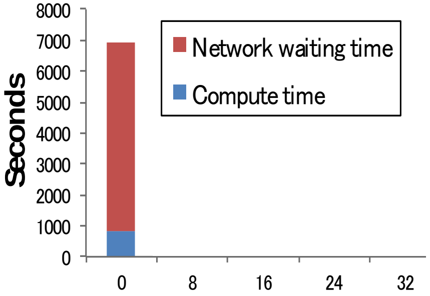
Collect outcomes and aggregate

Do nothing

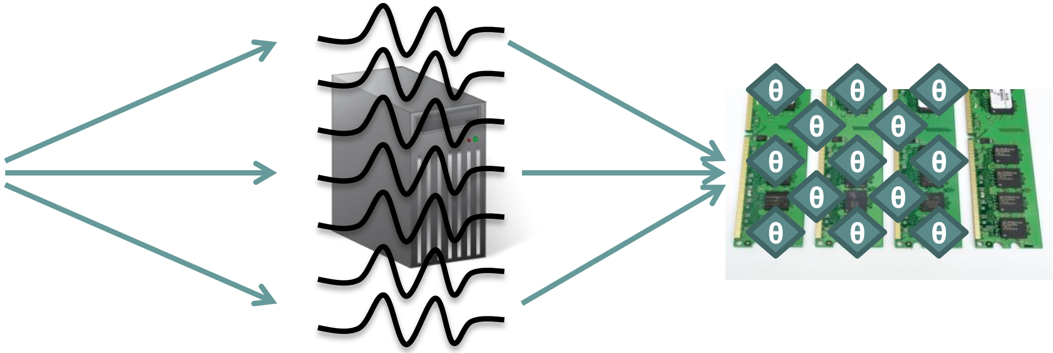
Do nothing

Do nothing

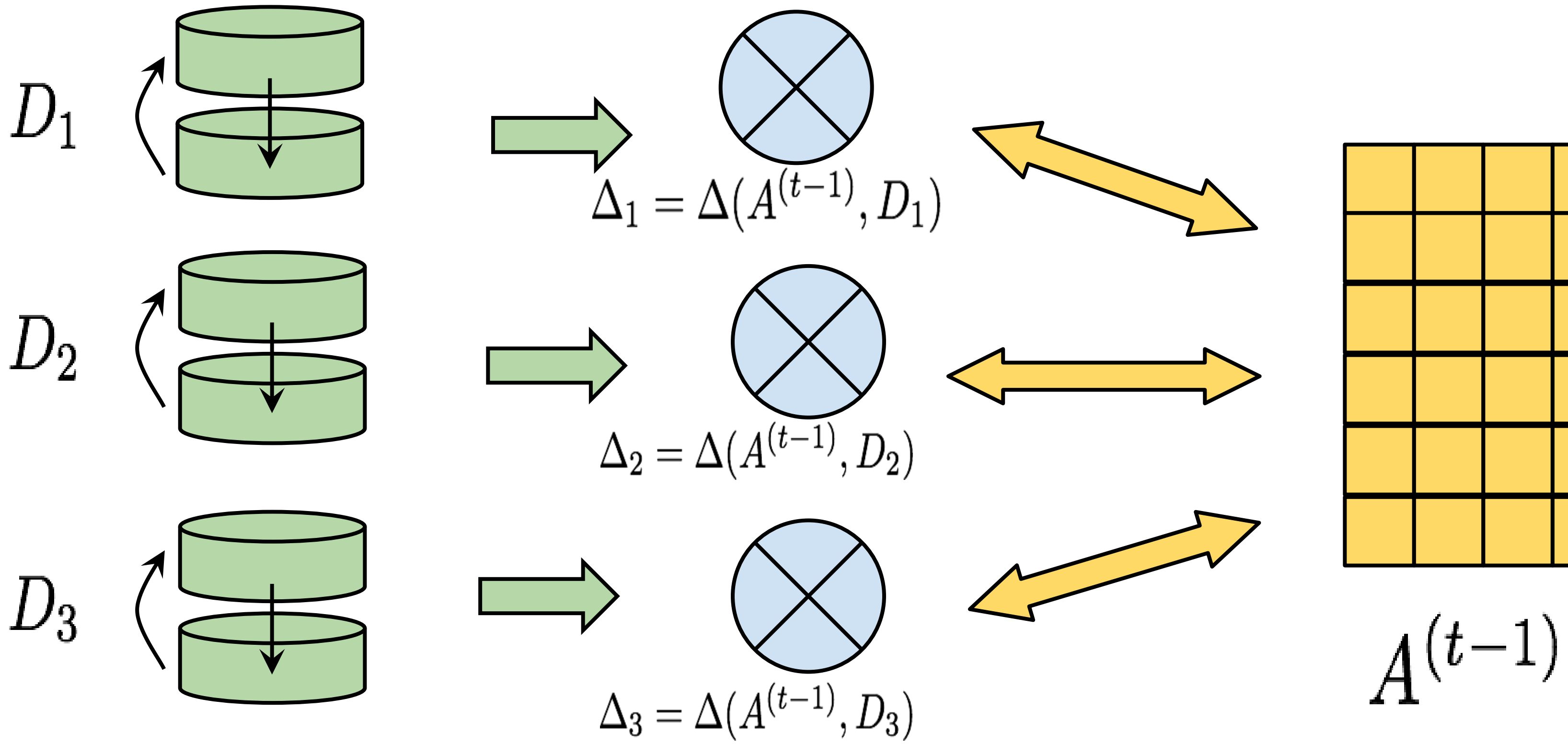




```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```



Data Parallelism



D_1

D_2

D_3

$$\Delta_1 = \Delta(A^{(t-1)}, D_1)$$

$$\Delta_2 = \Delta(A^{(t-1)}, D_2)$$

$$\Delta_3 = \Delta(A^{(t-1)}, D_3)$$

$A^{(t-1)}$

Additive Updates

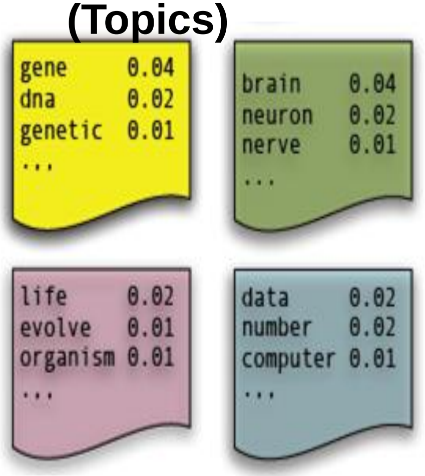
$$\Delta = \sum_{p=1}^{3} \Delta_p$$

$$A^{(t)} = F(A^{(t-1)}, \Delta)$$

Example Data Parallel: Topic Models

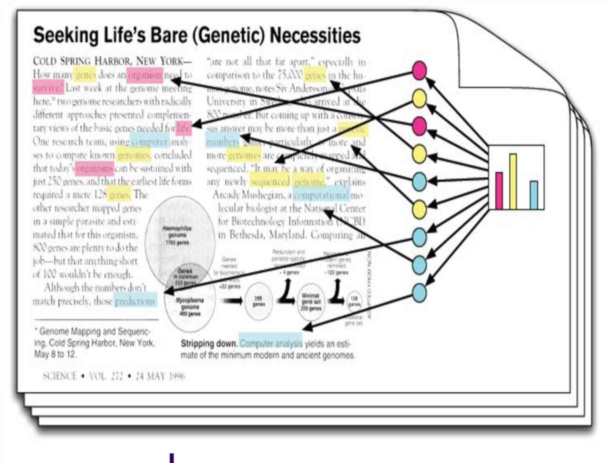
BIG DATA (billions of docs)

Model (Topics)



Update (MCMC algo)

Data (Docs)



Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK— How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 468 genes
Genes needed for biochemical
+22 genes
256 genes
Redundant and parasite-specific
−4 genes
Minimal gene set 250 genes
Redundant genes removed
−122 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

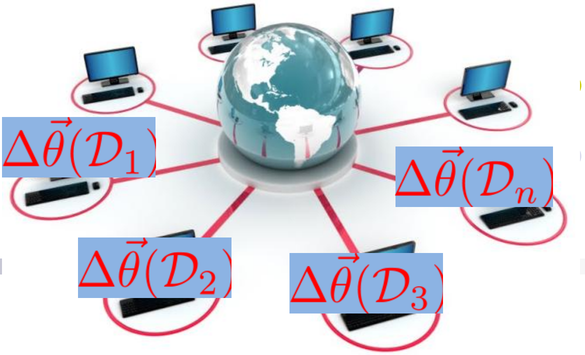
Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

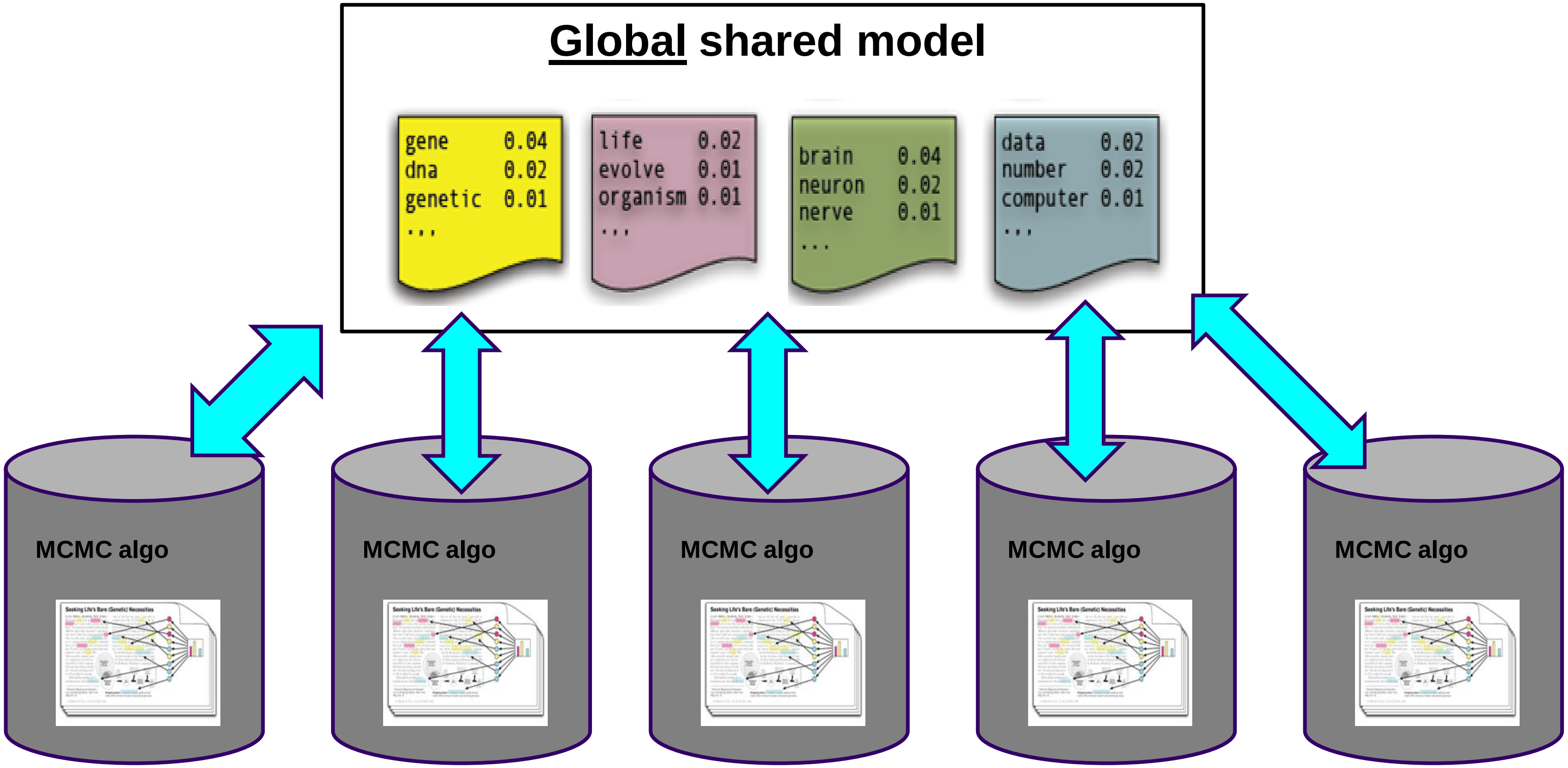
$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(D)$$

Example Data Parallel: Topic Models



$\Delta\vec{\theta}(\mathcal{D}_1)$ $\Delta\vec{\theta}(\mathcal{D}_n)$ $\Delta\vec{\theta}(\mathcal{D}_2)$ $\Delta\vec{\theta}(\mathcal{D}_3)$

$$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$$



Global shared model

```
gene     0.04
dna      0.02
genetic  0.01
...
```

```
life     0.02
evolve   0.01
organism 0.01
...
```

```
brain    0.04
neuron   0.02
nerve    0.01
...
```

```
data     0.02
number   0.02
computer 0.01
...
```

MCMC algo

MCMC algo

MCMC algo

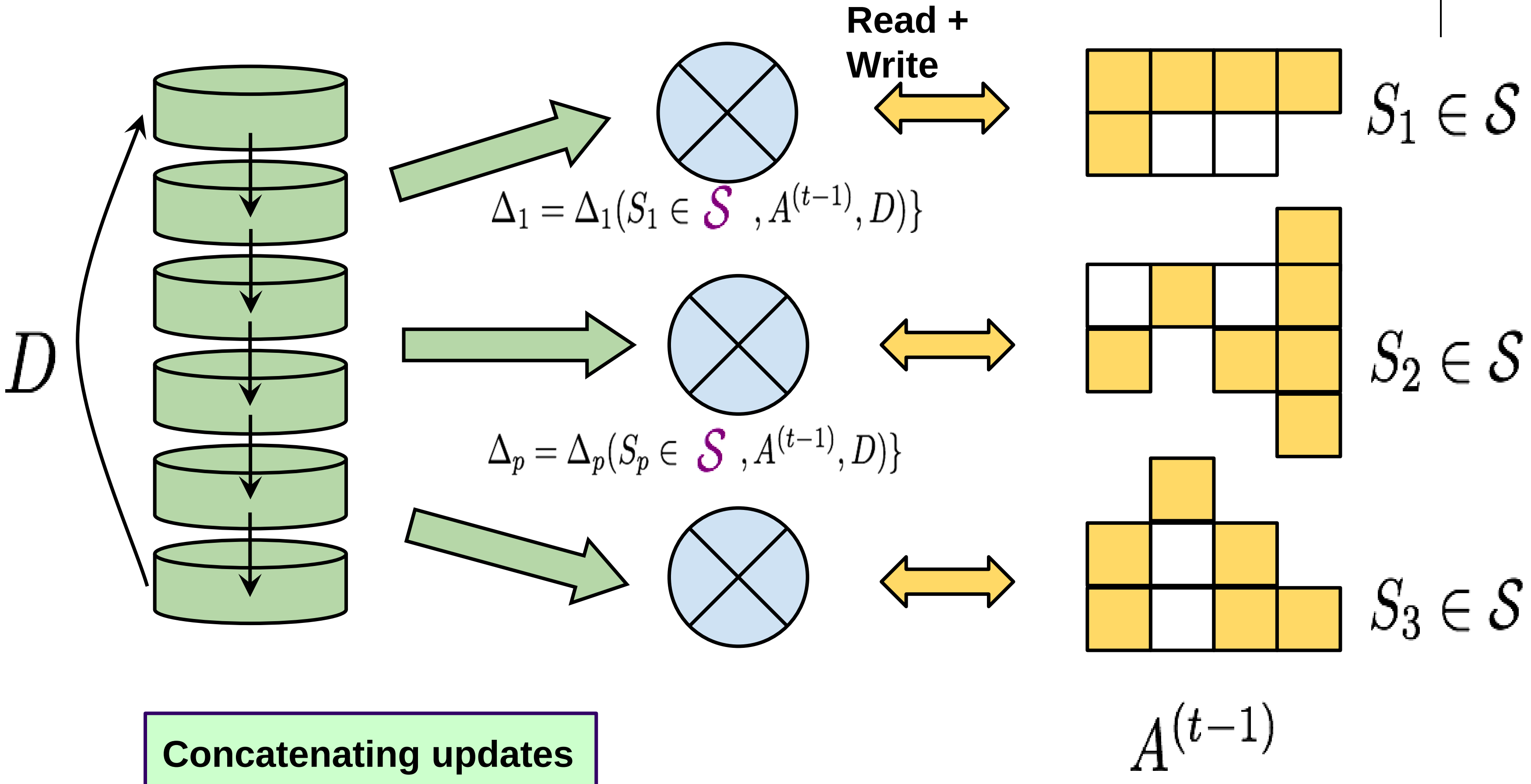
MCMC algo

MCMC algo

Model Parallelism

Scheduling Function

$\mathcal{S} = \mathcal{S}(A^{(t-1)}, D)$


Read + Write
D
$\Delta_1 = \Delta_1(S_1 \in \mathcal{S}\ , A^{(t-1)}, D)\}$
$\Delta_p = \Delta_p(S_p \in \mathcal{S}\ , A^{(t-1)}, D)\}$
$S_1 \in \mathcal{S}$
$S_2 \in \mathcal{S}$
$S_3 \in \mathcal{S}$
$A^{(t-1)}$

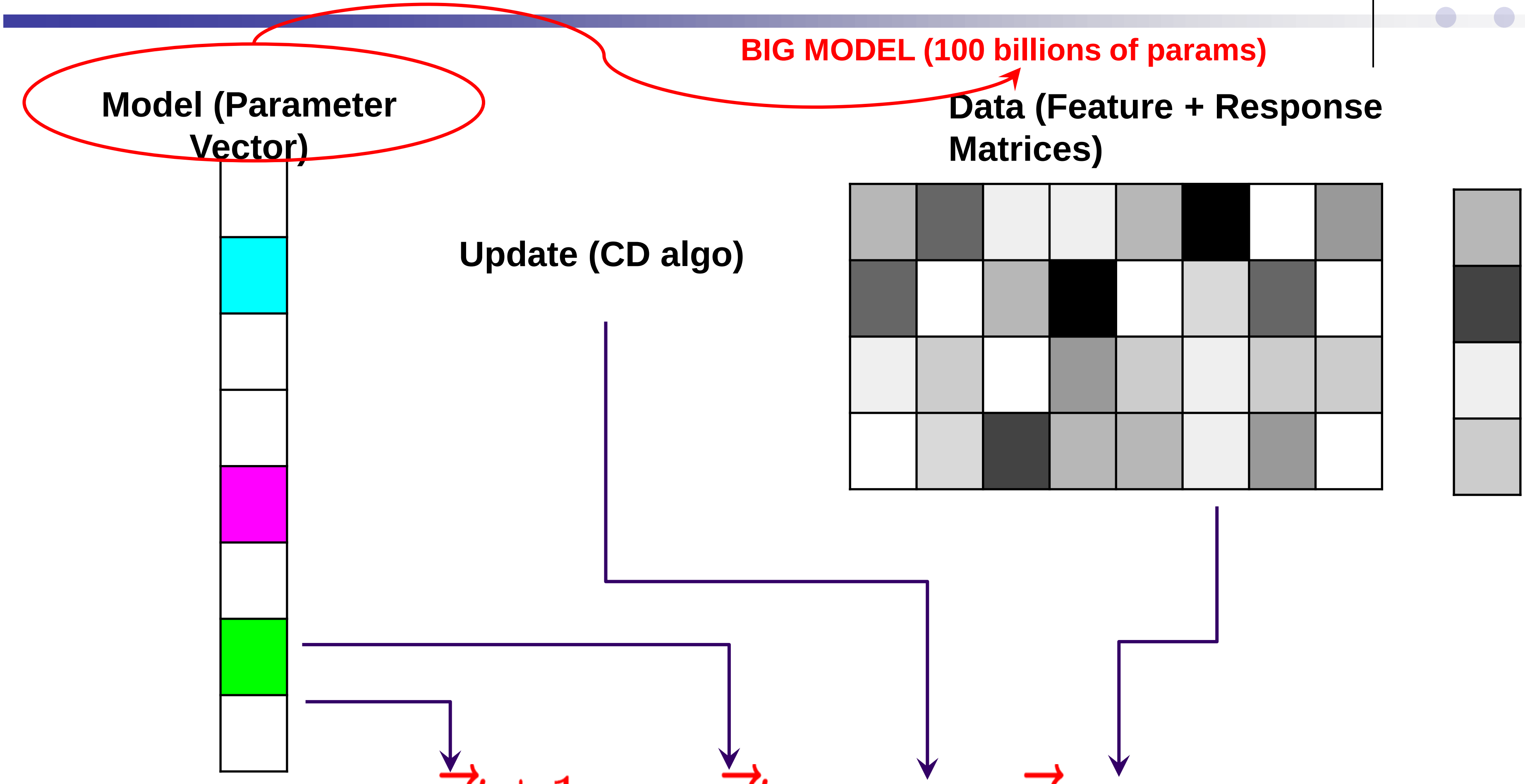
Concatenating updates

$$\Delta = \{\Delta_p\}$$

$$A^{(t)} = F(A^{(t-1)}, \Delta)$$

□ **model parameters not updated in this iteration**

Example Model Parallel: Lasso Regression



BIG MODEL (100 billions of params)

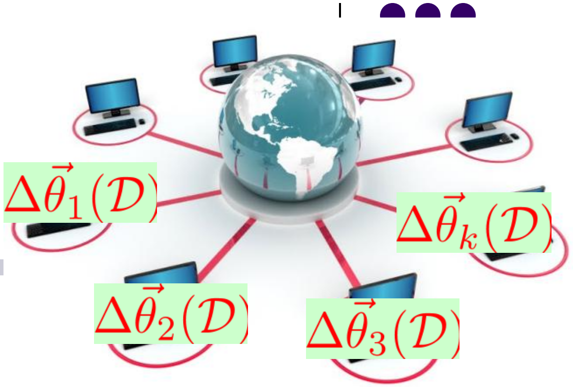
Model (Parameter Vector)

Data (Feature + Response Matrices)

Update (CD algo)

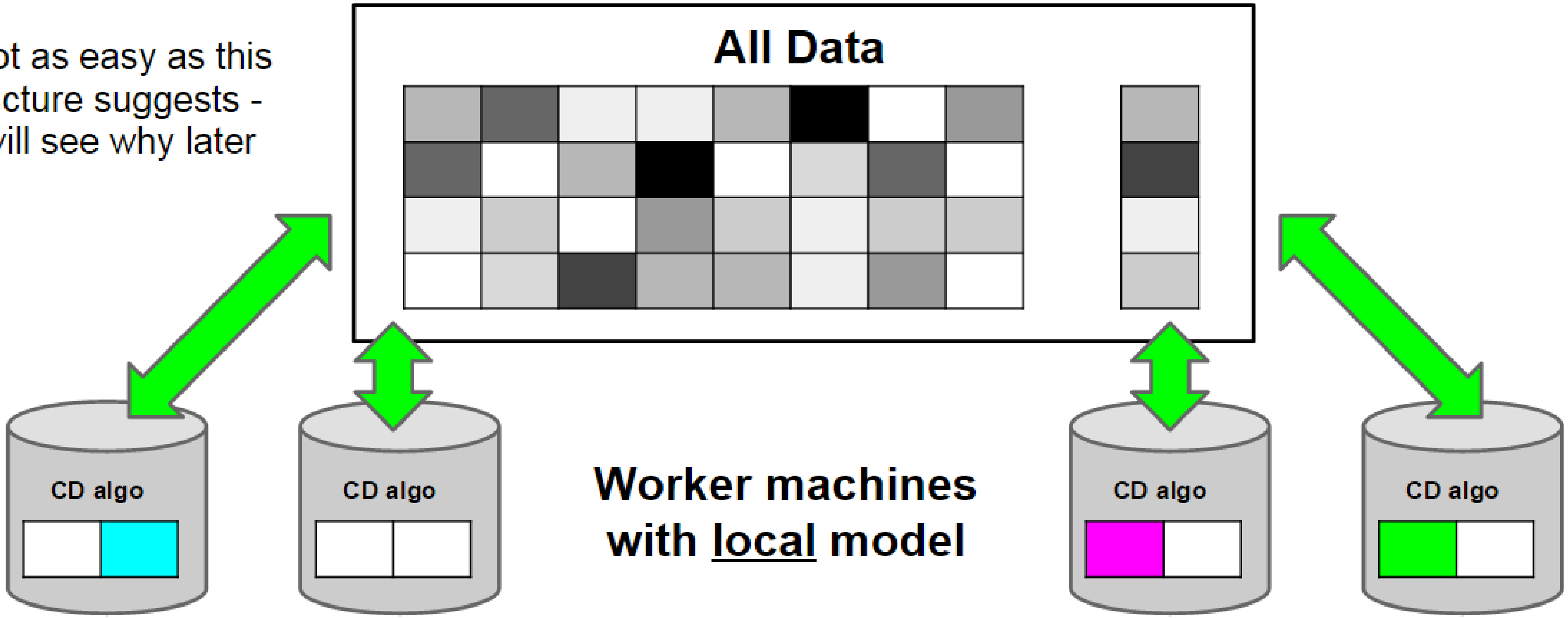
$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

Example Model Parallel: Lasso Regression


$\Delta\vec{\theta}_1(\mathcal{D})$
$\Delta\vec{\theta}_k(\mathcal{D})$
$\Delta\vec{\theta}_2(\mathcal{D})$
$\Delta\vec{\theta}_3(\mathcal{D})$

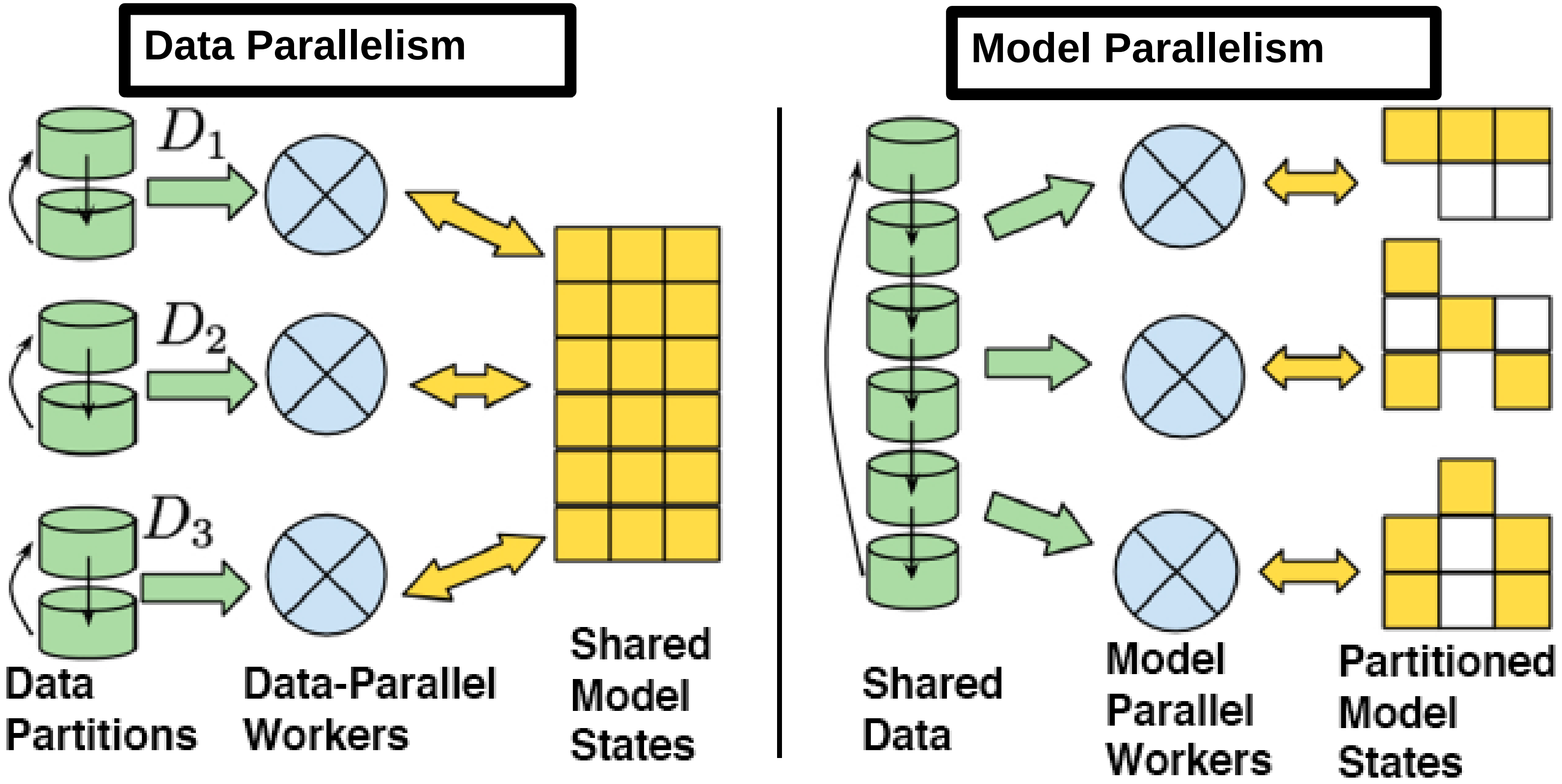
$$\vec{\theta} \equiv [\vec{\theta}_1^{\,\mathrm{T}}, \vec{\theta}_2^{\,\mathrm{T}}, \ldots, \vec{\theta}_k^{\,\mathrm{T}}\}^{\mathrm{T}}$$

Not as easy as this picture suggests - will see why later



Worker machines with <u>local</u> model

A Dichotomy of Data and Model in ML Programs



Data Parallelism

D_1

D_2

D_3

Data Partitions

Data-Parallel Workers

Shared Model States

Model Parallelism

Shared Data

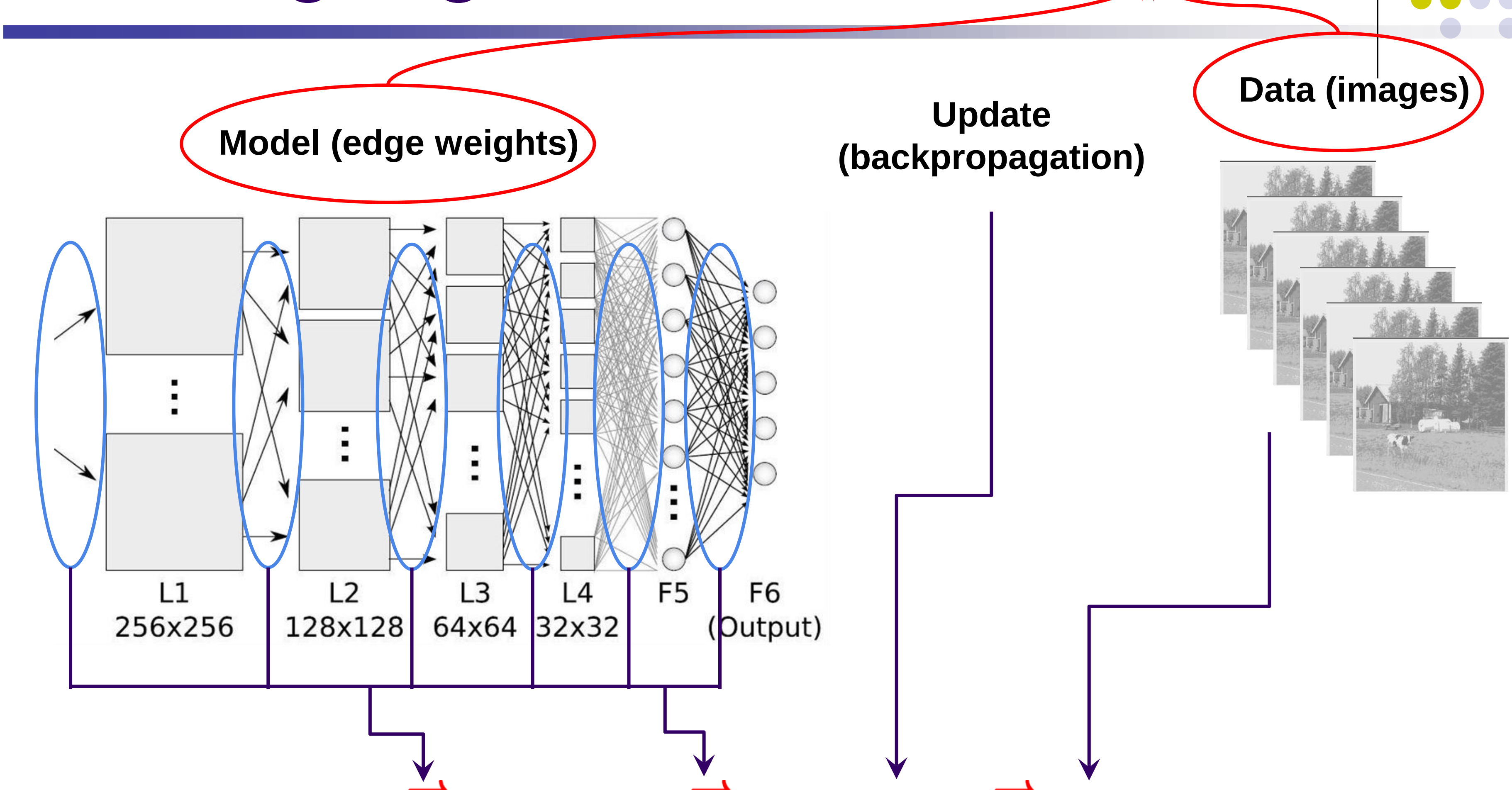
Model Parallel Workers

Partitioned Model States

$$\mathcal{D}_i \perp \mathcal{D}_j \mid \theta, \; \forall i \neq j \qquad \vec{\theta}_i \not\perp \vec{\theta}_j \mid \mathcal{D}, \; \exists (i,j)$$

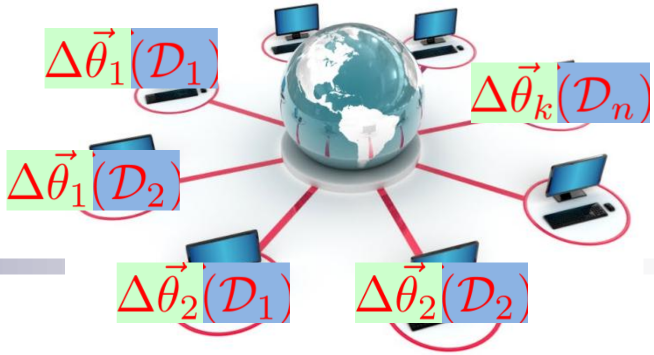
Data+Model Parallel: Solving Big Data+Model

Data & Model both big!
Millions of images,
Billions of weights
What to do?



$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

Data+Model Parallel: Solving Big Data+Model

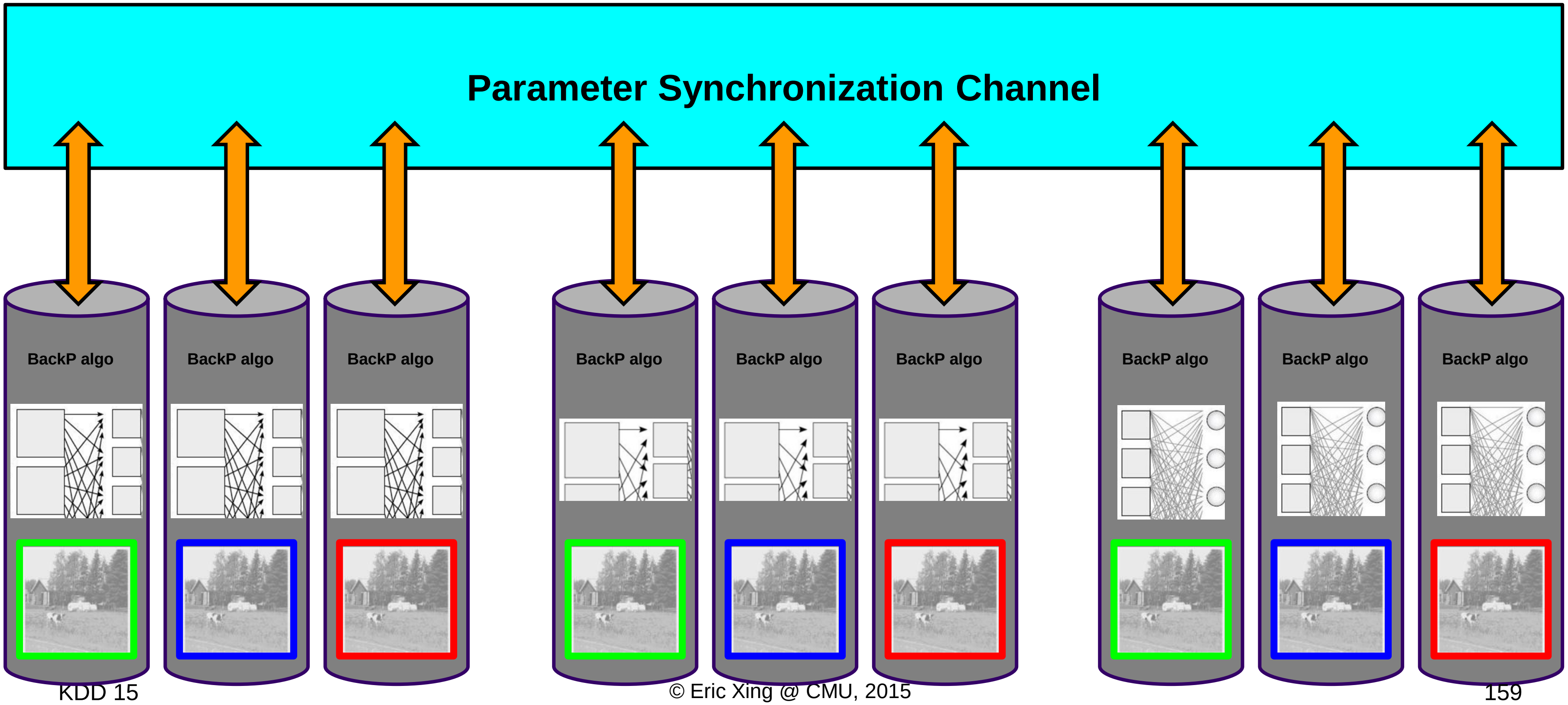


$\Delta\vec{\theta}_1(\mathcal{D}_1)$
$\Delta\vec{\theta}_k(\mathcal{D}_n)$
$\Delta\vec{\theta}_1(\mathcal{D}_2)$
$\Delta\vec{\theta}_2(\mathcal{D}_1)$
$\Delta\vec{\theta}_2(\mathcal{D}_2)$

Tackle Deep Learning scalability challenges by combining data+model parallelism

$$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$$

$$\vec{\theta} \equiv [\vec{\theta}_1^{\mathsf{T}}, \vec{\theta}_2^{\mathsf{T}}, \ldots, \vec{\theta}_k^{\mathsf{T}}\}^{\mathsf{T}}$$



Parameter Synchronization Channel

BackP algo

BackP algo

BackP algo

BackP algo

BackP algo

BackP algo

BackP algo

BackP algo

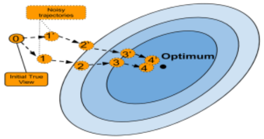
BackP algo

How difficult is data/model-parallelism?

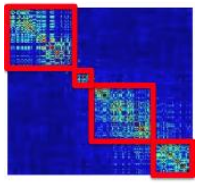
- Certain mathematical conditions must be met
- Data-parallelism generally OK when data IID (independent, identically distributed)
  - Very close to serial execution, in most cases
- Naive Model-parallelism doesn’t work
  - NOT equivalent to serial execution of ML algo
  - Need carefully designed schedule

Intrinsic Properties of ML Programs

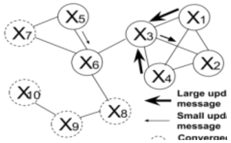
- ML is **optimization-centric**, and admits an **iterative convergent** algorithmic solution rather than a one-step closed form solution
  - **Error tolerance**: often robust against limited errors in intermediate calculations



  - **Dynamic structural dependency**: changing correlations between model parameters critical to efficient parallelization



  - **Non-uniform convergence**: parameters can converge in very different number of steps



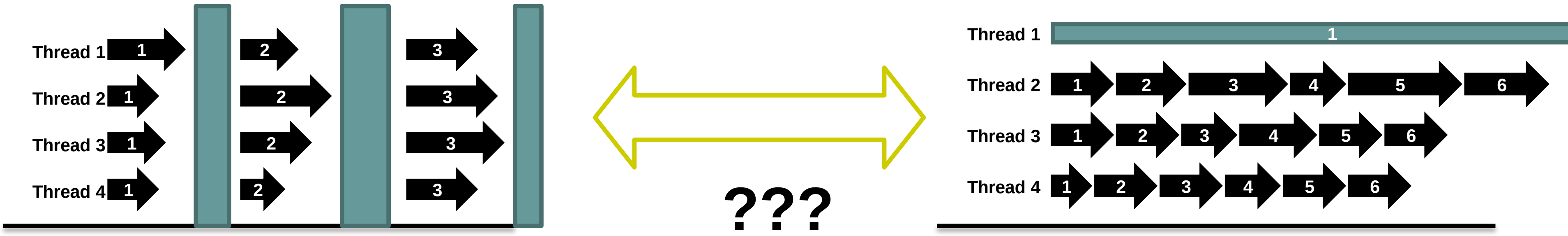
- Whereas traditional programs are **transaction-centric**, thus only guaranteed by **atomic correctness** at every step

- Most existing platforms (e.g., Spark, GraphLab) have not yet systematically explore and exploit above properties

Challenges in Data Parallelism

- **Existing ways are either safe/slow (BSP), or fast/risky (Async)**
- **Challenge 1: Need “Partial” synchronicity**
  - Spread network comms evenly (don’t sync unless needed)
  - Threads usually shouldn’t wait – but mustn’t drift too far apart!
- **Challenge 2: Need straggler tolerance**
  - Slow threads must somehow catch up

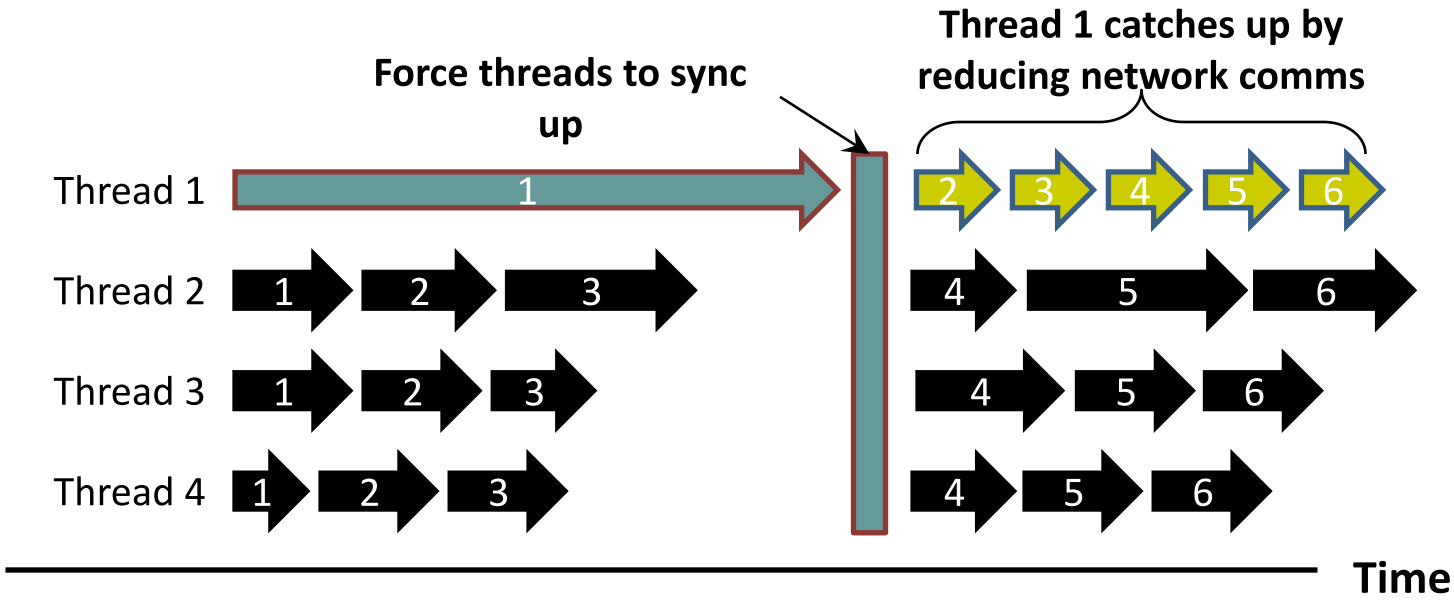
BSP **Async**



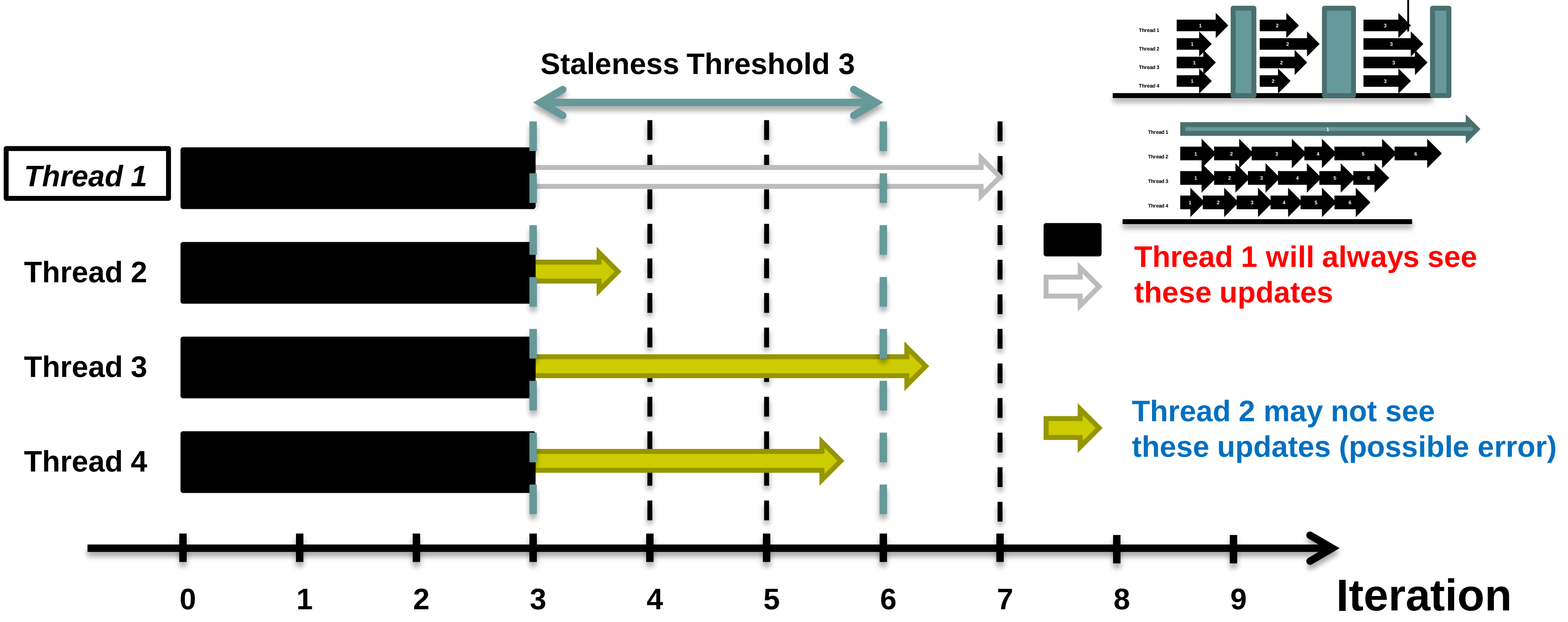
Is persistent memory really necessary for ML?

Is there a middle ground for data-parallel consistency?

- **Challenge 1: “Partial” synchronicity**
  - Spread network comms evenly (don’t sync unless needed)
  - Threads usually shouldn’t wait – but mustn’t drift too far apart!
- **Challenge 2: Straggler tolerance**
  - Slow threads must somehow catch up



High-Performance Consistency Models for Fast Data-Parallelism [Ho et al., 2013]



Thread 1 will always see these updates

Thread 2 may not see these updates (possible error)

Stale Synchronous Parallel (SSP), a “bounded-asycnhronous” model

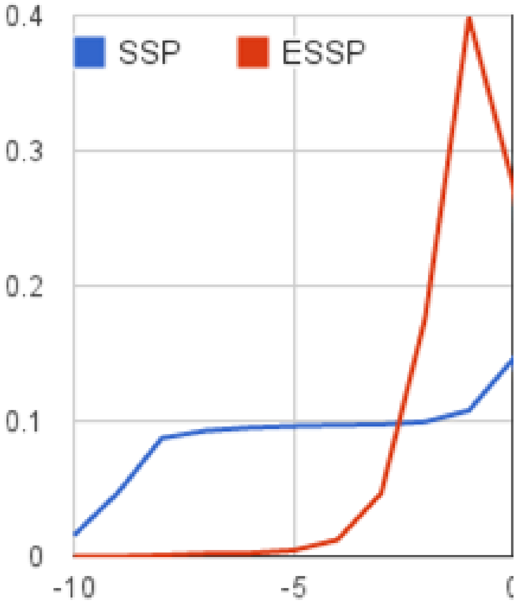
- Allow threads to run at their own pace, without synchronization
- Fastest/slowest threads not allowed to drift >S iterations apart
- Threads cache local (stale) versions of the parameters, to reduce network syncing

Consequence:

- Asynchronous-like speed, BSP-like ML correctness guarantees
- Guaranteed age bound (staleness) on reads
- Contrast: no-age-guarantee Eventual Consistency seen in Cassandra, Memcached

Improving Bounded-Async via Eager Updates [Dai et al., 2015]

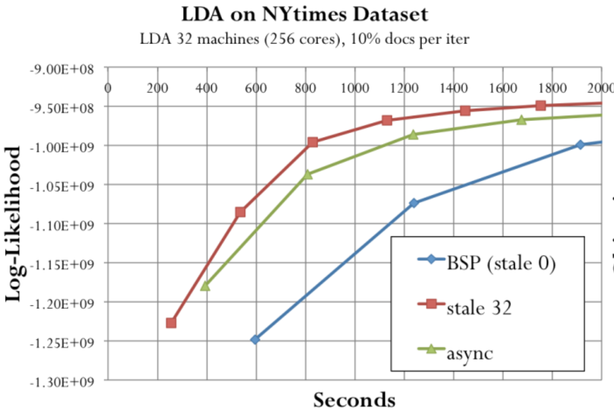
- Eager SSP (ESSP) protocol
  - Use spare bandwidth to push fresh parameters sooner
- Figure: difference in stale reads between SSP and ESSP
  - ESSP has fewer stale reads; lower staleness variance
  - Faster, more stable convergence (theorems later)



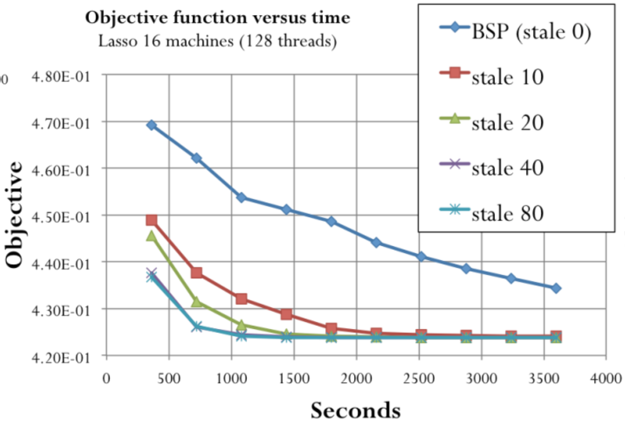
Enjoys Async Speed, yet BSP Guarantee, across algorithms

- Scale up Data Parallelism without being limited by long BSP synchronization time

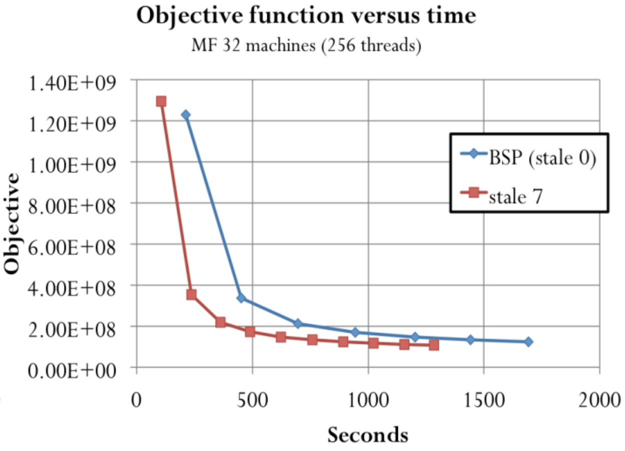
- Effective across different algorithms, e.g. LDA, Lasso, Matrix Factorization:



LDA



LASSO

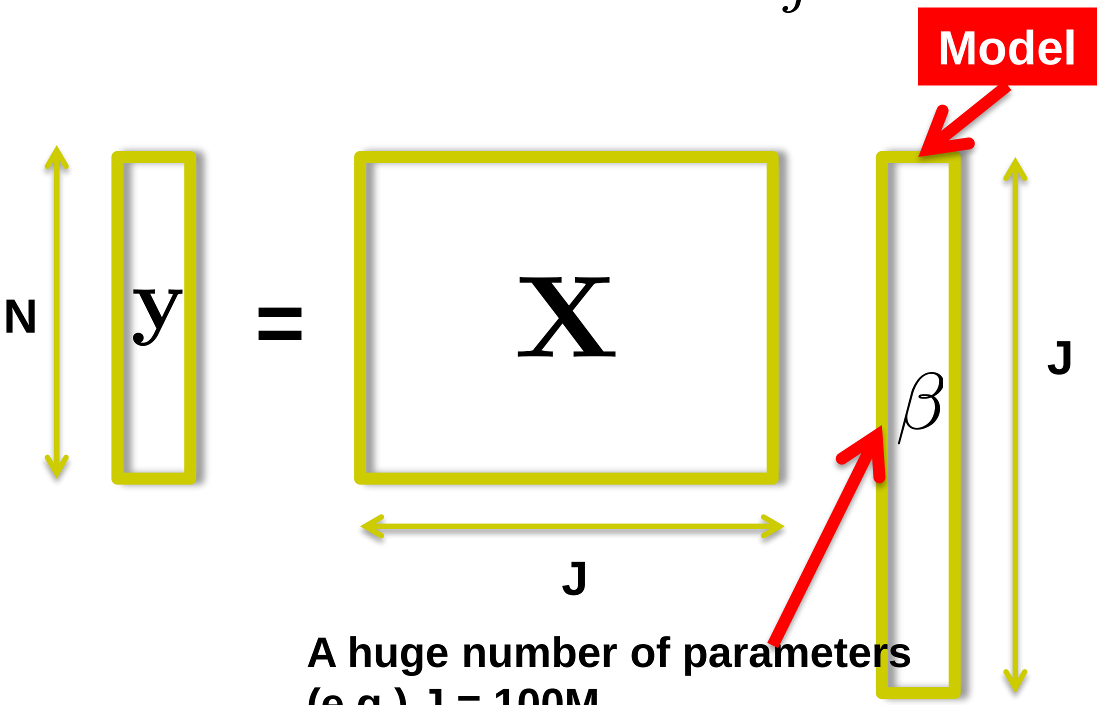


Matrix Fact.

Challenges in Model Parallelism

- Recall Lasso regression:

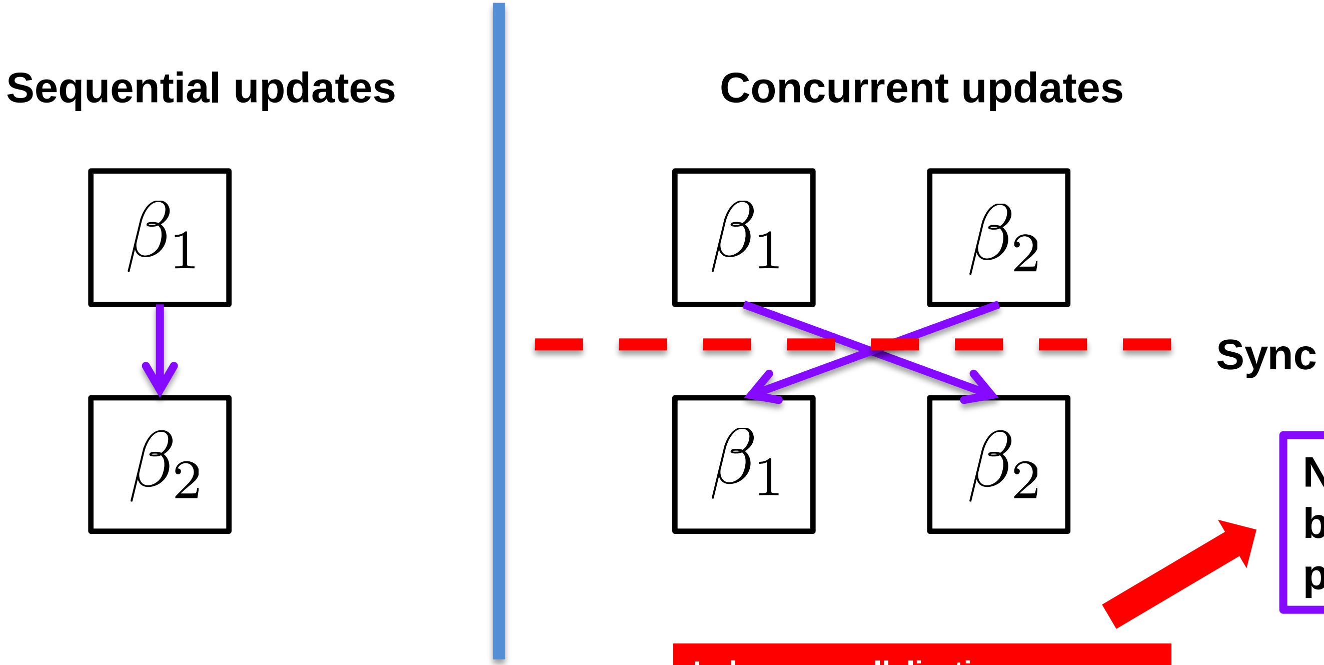
$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_j |\beta_j|$$



A huge number of parameters
(e.g.) J = 100M

Challenge 1: Model Dependencies

- Concurrent updates of β may induce errors

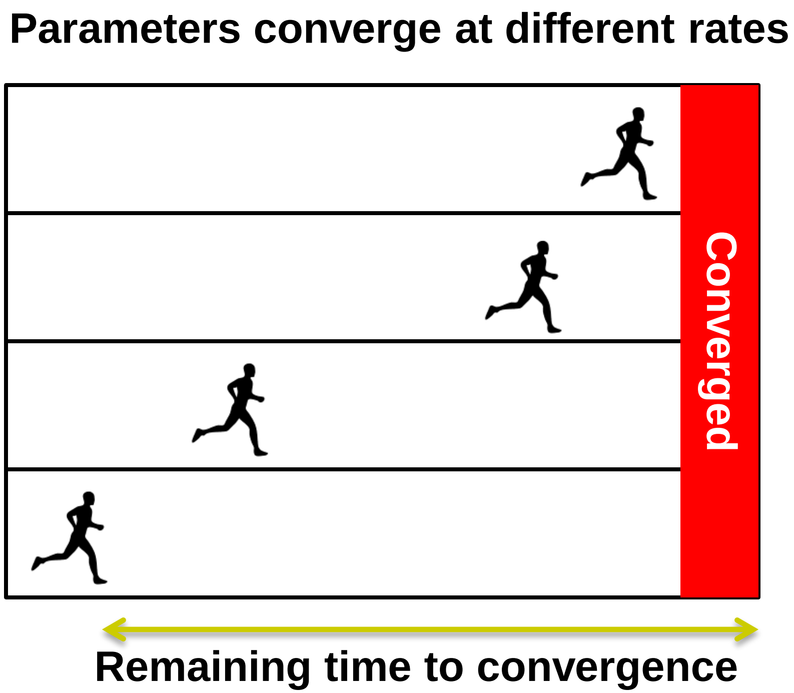


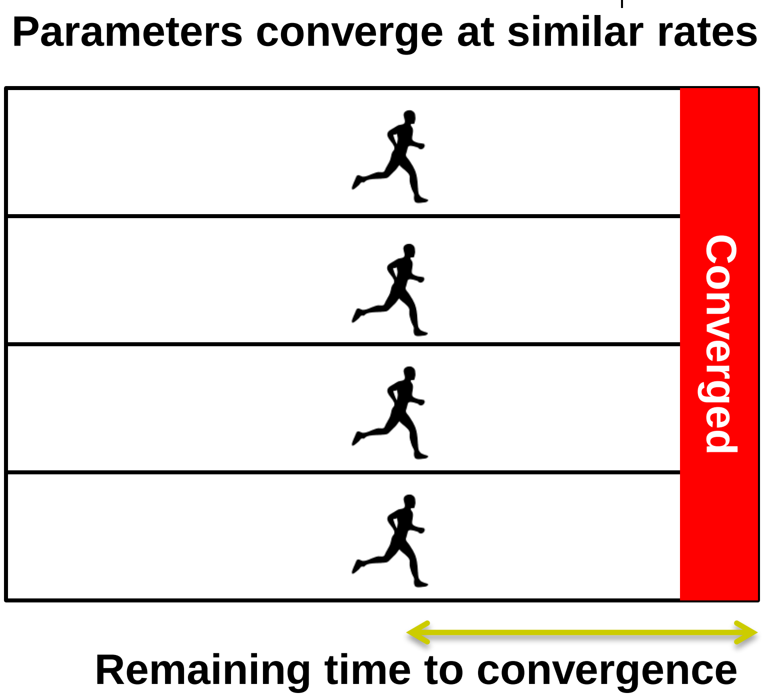
Need to check $x_1^T x_2$ before updating parameters

Induces parallelization error

$$\beta_1^{(t)} \leftarrow S(\mathbf{x}_1^T \mathbf{y} - \boxed{\mathbf{x}_1^T \mathbf{x}_2 \beta_2^{(t-1)}}, \lambda)$$

Challenge 2: Uneven Convergence Rate on Parameters


Parameters converge at different rates
Converged
Remaining time to convergence

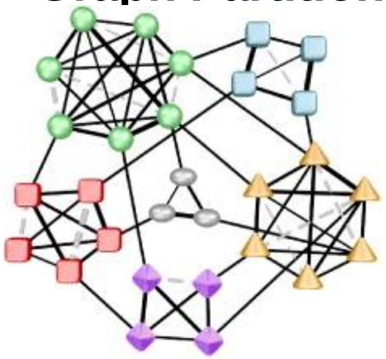

Parameters converge at similar rates
Converged
Remaining time to convergence

- Convergence time determined by slowest parameters
- How to make slowest parameters converge more quickly?

Is there a middle ground for model-parallel consistency?

- **Existing ways are either safe but slow, or fast but risky**
- **Challenge 1: need approximate but fast model partition**
  - Full representation of data/model, and explicitly compute all dependencies via graph cut is not feasible
- **Challenge 2: need dynamic load balancing**
  - Capture and explore transient model dependencies
  - Explore uneven parameter convergence

Graph Partition

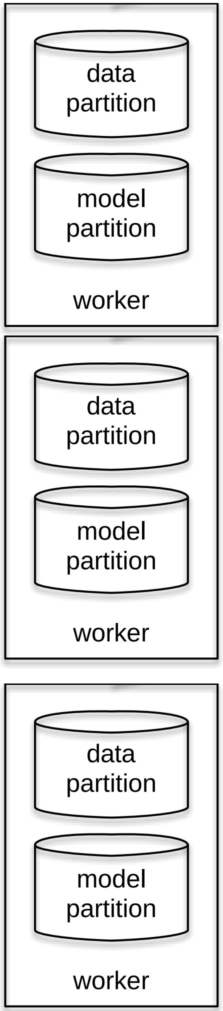


???

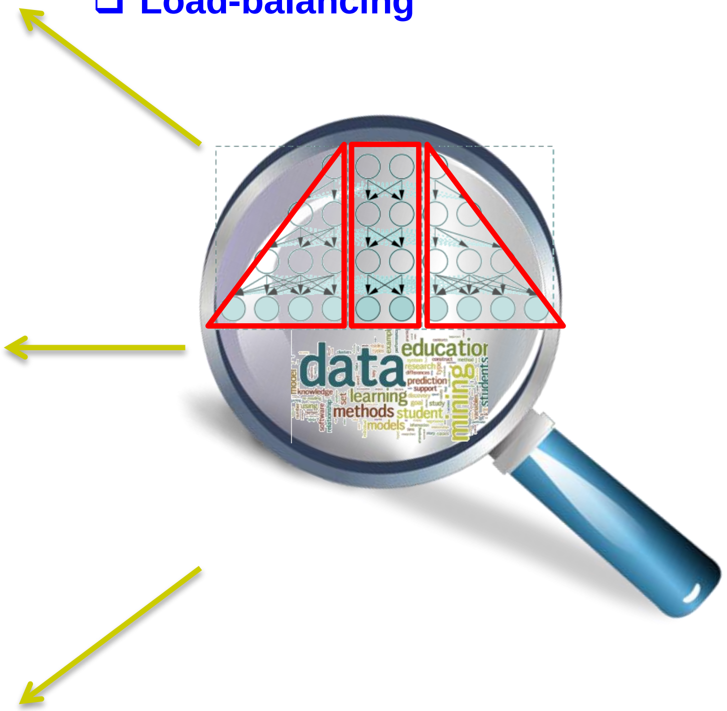
Is full consistency really necessary for ML?

Random Partition

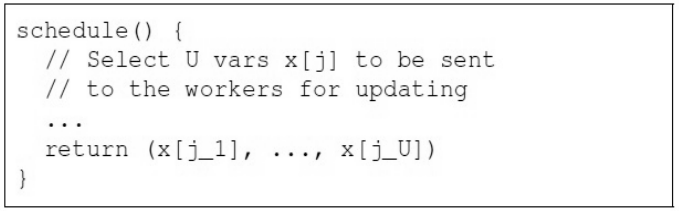
Structure-Aware Parallelization (SAP) [Lee et al., 2014; Kumar et al., 2014]



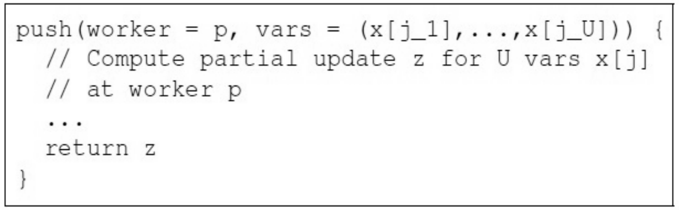
- ❑ **Careful model-parallel execution:**
  - ❑ **Structure-aware scheduling**
  - ❑ **Variable prioritization**
  - ❑ **Load-balancing**



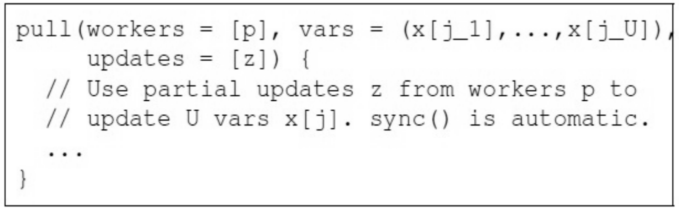
- ❑ **Simple programming:**
  - ❑ **Schedule()**
  - ❑ **Push()**
  - ❑ **Pull()**



```
schedule() {
  // Select U vars x[j] to be sent
  // to the workers for updating
  ...
  return (x[j_1], ..., x[j_U])
}
```



```
push(worker = p, vars = (x[j_1],...,x[j_U])) {
  // Compute partial update z for U vars x[j]
  // at worker p
  ...
  return z
}
```

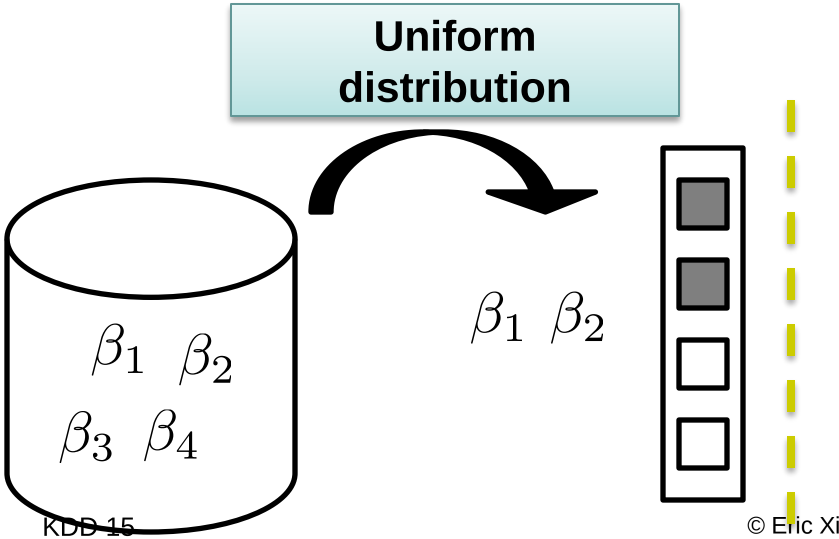


```
pull(workers = [p], vars = (x[j_1],...,x[j_U]),
     updates = [z]) {
  // Use partial updates z from workers p to
  // update U vars x[j]. sync() is automatic.
  ...
}
```

Schedule 1: Priority-based [Lee et al., 2014]

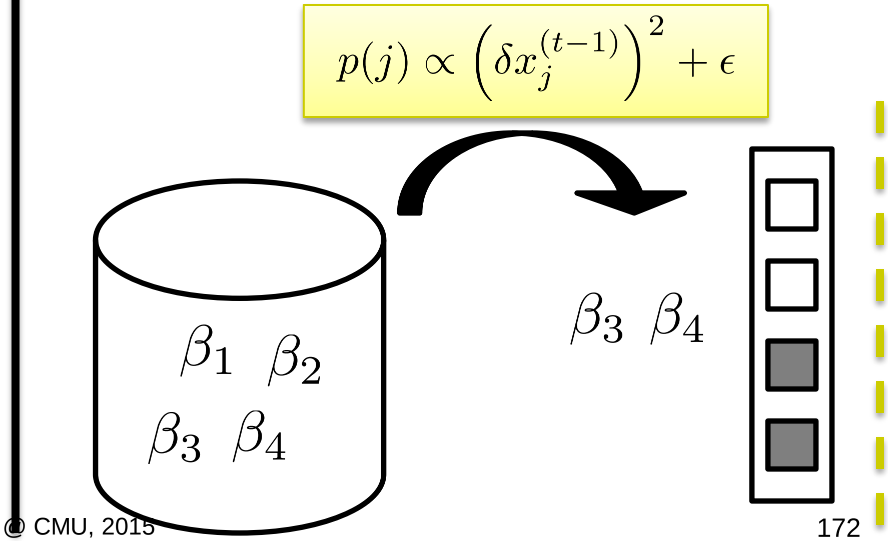
- Choose params to update based on convergence progress
  - Example: sample params with probability proportional to their recent change
  - Approximately maximizes the convergence progress per round

Shotgun [Bradley et al. 2011]



Uniform distribution

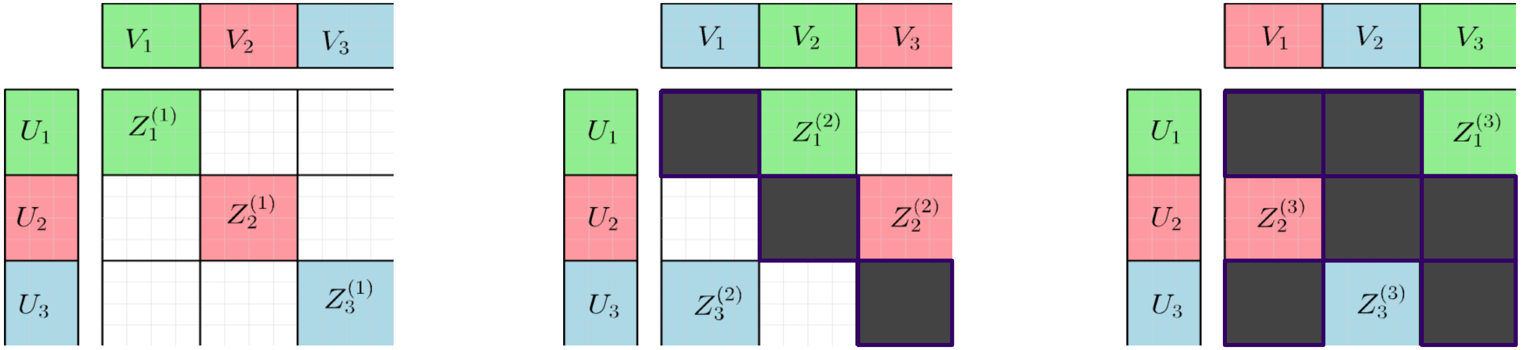
Priority-based scheduling



$$p(j) \propto \left(\delta x_j^{(t-1)}\right)^2 + \epsilon$$

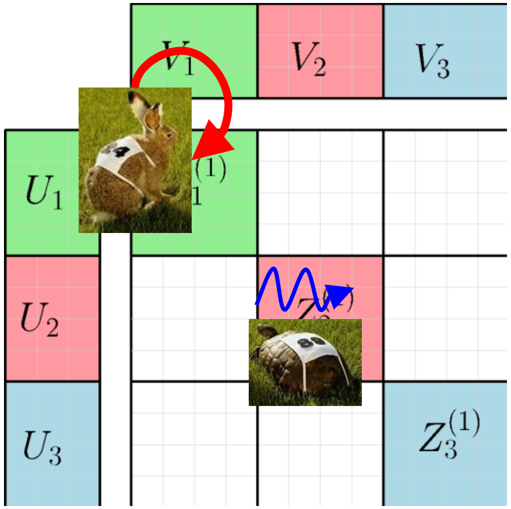
Schedule 2: Block-based (with load balancing) [Kumar et al., 2014]

Partition data & model into *d* × *d* blocks
Run different-colored blocks in parallel

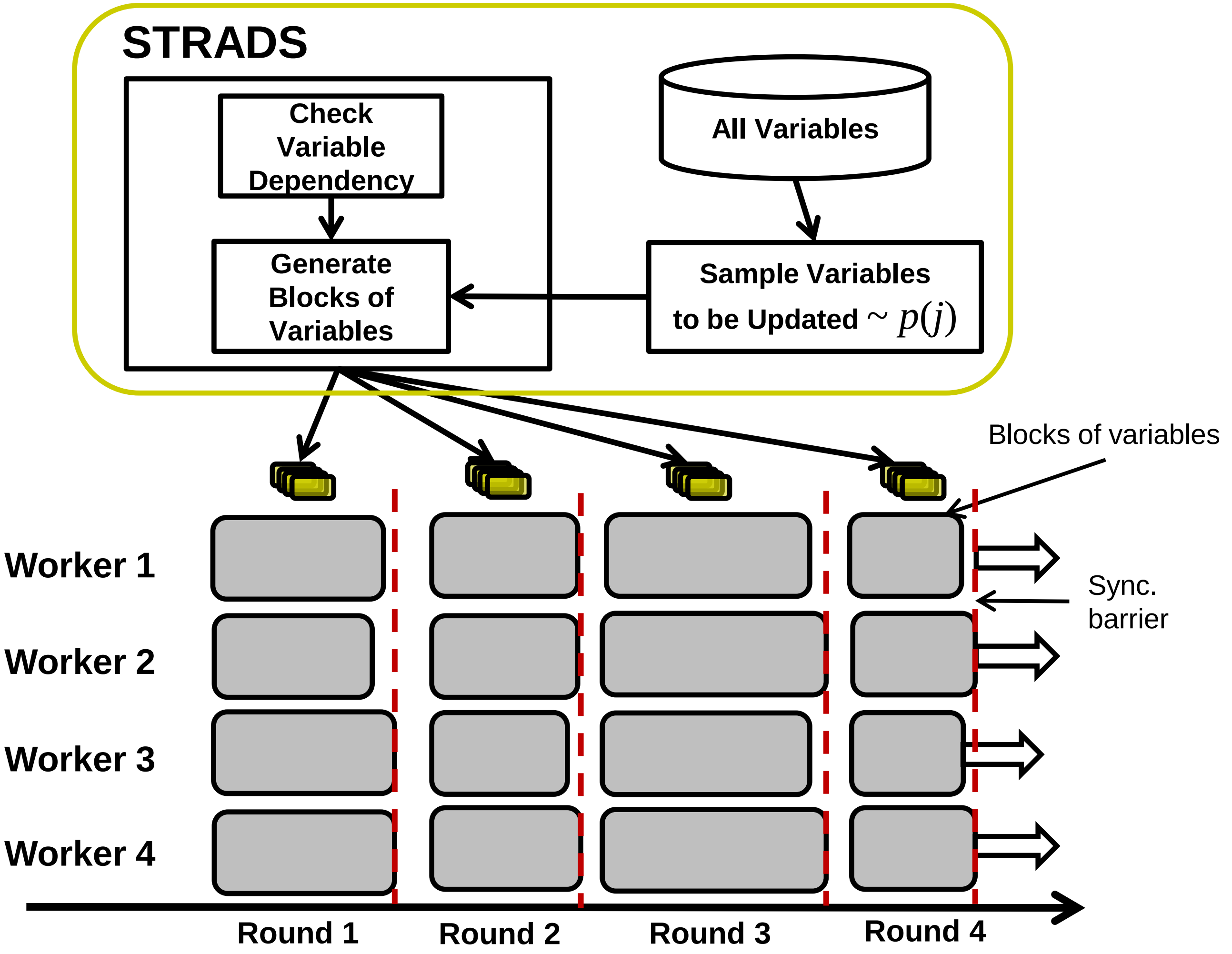


V_1 V_2 V_3
U_1 U_2 U_3
$Z_1^{(1)}$ $Z_2^{(1)}$ $Z_3^{(1)}$
V_1 V_2 V_3
U_1 U_2 U_3
$Z_1^{(2)}$ $Z_2^{(2)}$ $Z_3^{(2)}$
V_1 V_2 V_3
U_1 U_2 U_3
$Z_1^{(3)}$ $Z_2^{(3)}$ $Z_3^{(3)}$

Blocks with less data/para or experience less straggling run more iterations
Automatic load-balancing + better convergence



Structure-aware Dynamic Scheduler (STRADS) [Lee et al., 2014, Kumar et al., 2014]



STRADS

Check Variable Dependency

All Variables

Generate Blocks of Variables

Sample Variables to be Updated $\sim p(j)$

Blocks of variables

Worker 1

Worker 2

Worker 3

Worker 4

Sync. barrier

Round 1

Round 2

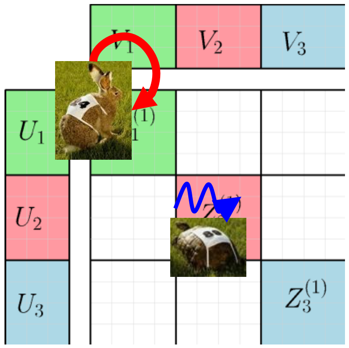
Round 3

Round 4

- **Priority Scheduling**

$$\{\beta_j\} \sim \left(\delta\beta_j^{(t-1)}\right)^2 + \eta$$

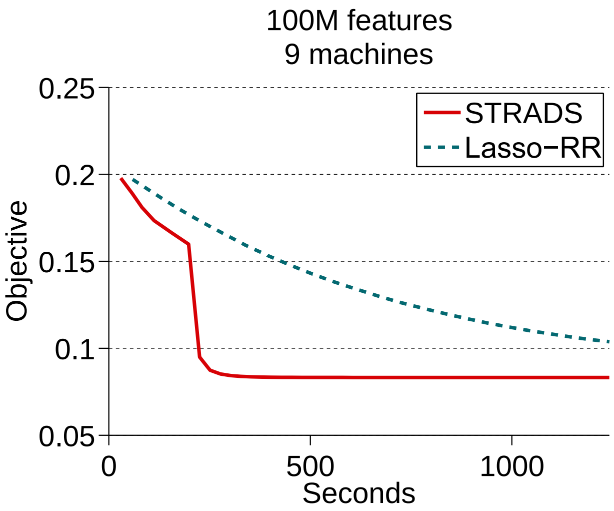
- **Block scheduling**



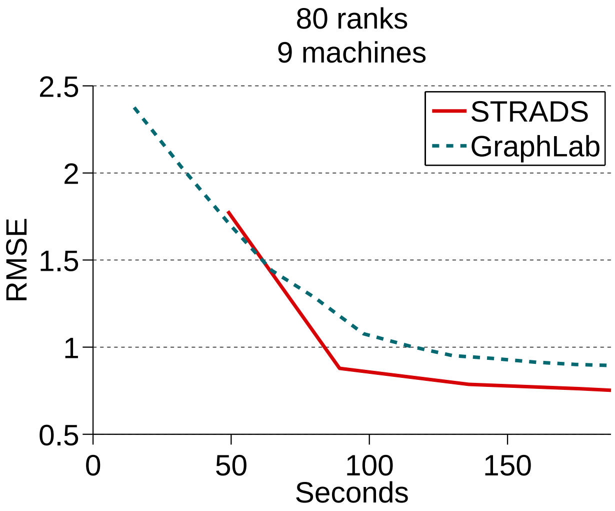
*[Kumar, Beutel, Ho and Xing, **Fugue: Slow-worker agnostic distributed learning**, AISTATS 2014]*

Avoids dependent parallel updates, attains near-ideal convergence speed

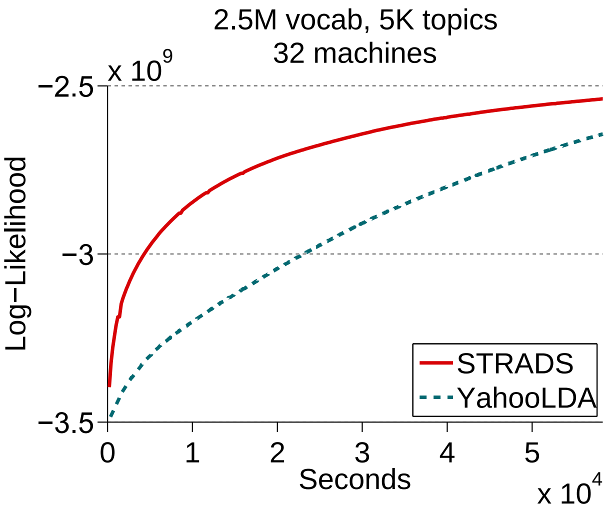
- STRADS+SAP achieves better speed and objective



Lasso



MF

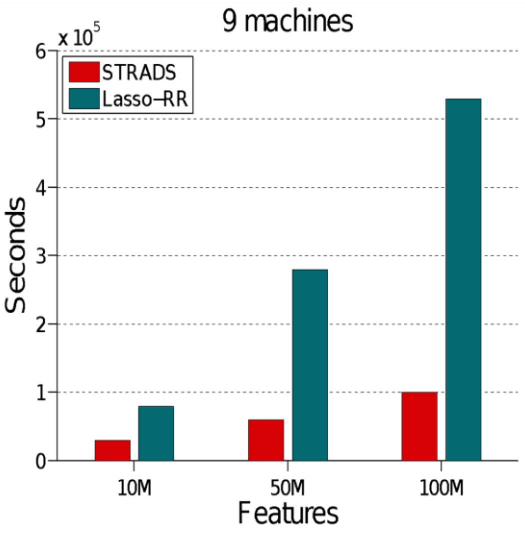


LDA

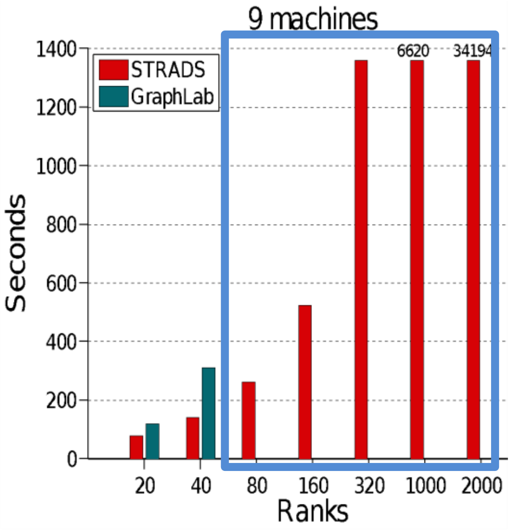
Efficient for large models

- Model is partitioned => can run larger models on same hardware

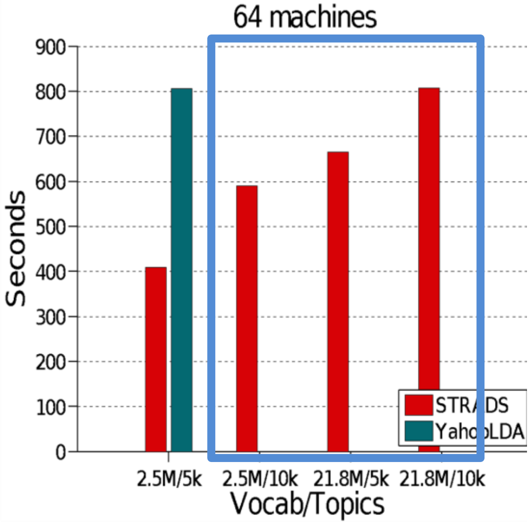
Lasso



MF



LDA



Theory of Real Distributed ML Systems

Why study parallel ML theory?

- What sequential guarantees still hold in parallel setting?
  - Under what conditions?
- Growing body of literature for “ideal” parallel systems
  - Serializable– equivalent to single-machine execution in some sense
  - Focused on per-iteration analysis
    - Abstract away computational/comms cost
    - Predicting real-world running time requires these costs to be put back
- “Real-world” parallel systems a work in progress
  - Asynchronous or bounded-async approaches can empirically work better than synchronous approaches
    - Need additional theoretical analysis to understand why
    - Async => no serializability… why does it still work?
  - Parallelization requires data and/or model partitioning… many strategies exist
    - Want partitioning strategies that are provably correct
    - Need to determine when/where independence is violated, and what impact such violation has on algorithm correctness

Challenges in real-world distributed systems

- Real-world systems need asynchronous execution and load balancing
  - Synchronous system: load imbalances => slow workers => waiting at barriers
  - Need load balancing to reduce load at slow workers
  - Need asynchronous execution so faster workers can proceed without waiting
- Solution 1: key-value stores
  - Automatically manages communication with bounded asynchronous guarantees
- Solution 2: scheduling systems
  - Automatically balances workload across workers; also performs prioritization and dependency checking

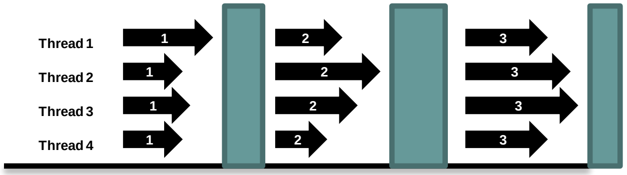
© Eric Xing @ CMU, 2015

Communication strategies

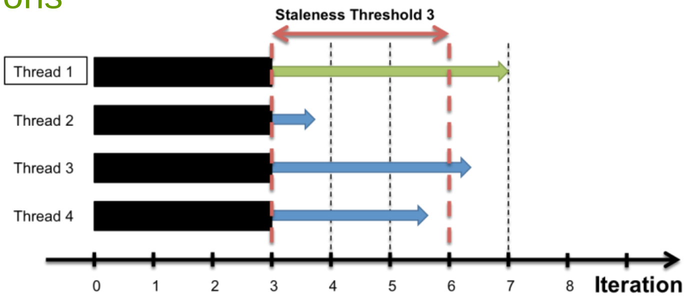
- Data parallel
  - Partition data across workers
    - Or fetch small batches of data in an online/streaming fashion
  - Communicate model as needed to workers
    - e.g. key-value store with bounded asynchronous model – theoretical consequences?
- Model parallel
  - Partition model across workers
    - Model partitions can change dynamically during execution – theoretical consequences?
  - Send data to workers as needed (e.g. from shared database)
    - Or place full copy of data on each worker (since data is immutable)
- Data + Model parallel?
  - Partition both data and model across workers
  - Wide space of strategies; need to reduce model and data communication
    - Reduce model communication by exploiting independence between variables
    - Reduce data and model communication via broadcast strategies, e.g. Halton sequence

Bridging Models for Parallel Programming

- Bulk Synchronous Parallel **[Valiant, 1990]** is a bridging model
  - Bridging model specifies how/when parallel workers should compute, and how/when workers should communicate
  - Key concept: barriers
    - No communication before barrier, only computation
    - No computation inside barrier, only communication



  - Computation is "serializable" – many sequential theoretical guarantees can be applied with no modification
- Bounded Asynchronous Parallel (BAP) bridging model
  - Key concept: bounded staleness **[Ho et al., 2013; Dai et al., 2015]**
    - Workers re-use old version of parameters, up to s iterations old – no need to barrier
    - Workers wait if parameter version older than s iterations



© Eric Xing @ CMU, 2015

Types of Convegence Guarantees

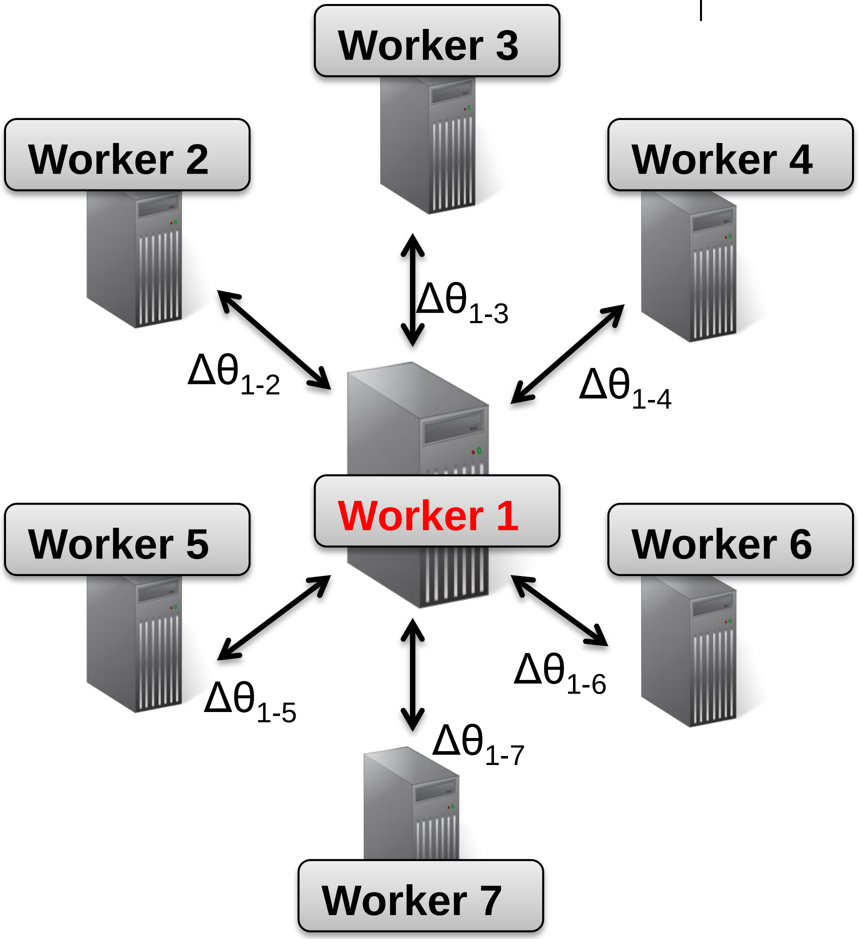
- Regret/Expectation bounds on parameters
  - Better bounds => better convergence progress per iteration
- Probabilistic bounds on parameters
  - Similar meaning to regret/expectation bounds, usually stronger in guarantee
- Variance bounds on parameters
  - Lower variance => higher stability near optimum => easier to determine convergence
- For data parallel?
- For Model parallel?
- For Data + model parallel?

BAP Data Parallel: Can we do value-bounding?

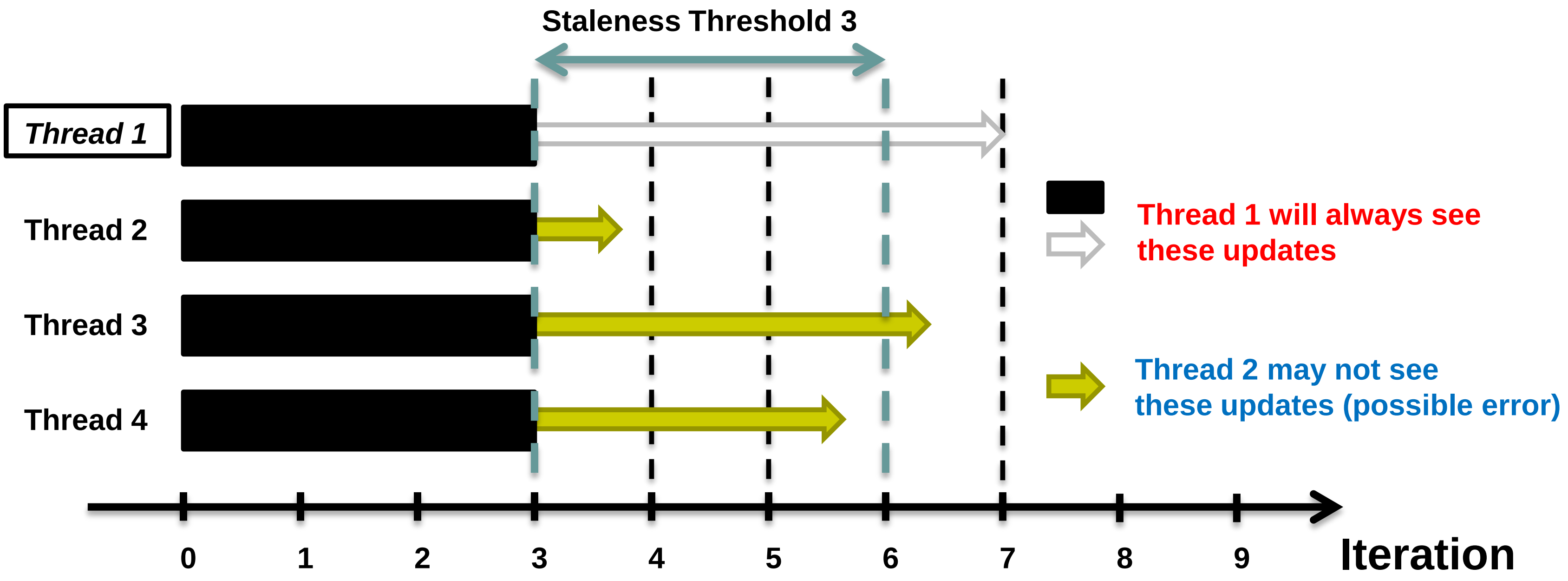
- **Idea:** limit model parameter difference $\Delta\theta_{i-j} = ||\theta_i - \theta_j||$ between machines i,j to < a threshold

- Does not work in practice!
  - To guarantee that $\Delta\theta_{i-j}$ has not exceeded the threshold, machines must wait to communicate with each other
  - No improvement over synchronous execution!

- Rather than controlling parameter difference via magnitude, what about via iteration count?
  - This is the (E)SSP communication model…



BAP Data Parallel: (E)SSP model [Ho et al., 2013; Dai et al., 2015]


Staleness Threshold 3
Thread 1
Thread 2
Thread 3
Thread 4
Thread 1 will always see these updates
Thread 2 may not see these updates (possible error)
0
1
2
3
4
5
6
7
8
9
Iteration

Stale Synchronous Parallel (SSP)

- Allow threads to run at their own pace, without synchronization
- Fastest/slowest threads not allowed to drift >S iterations apart
- Threads cache local (stale) versions of the parameters, to reduce network syncing

Consequence:

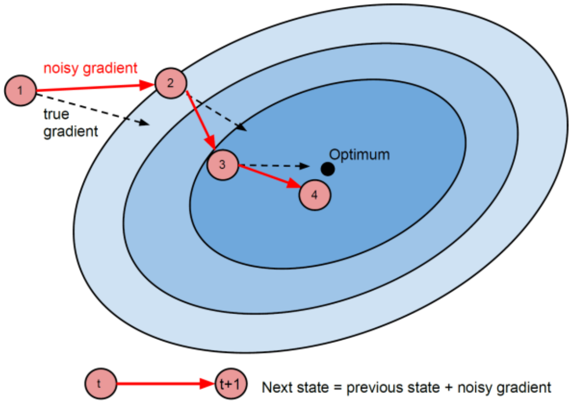
- Asynchronous-like speed, BSP-like ML correctness guarantees
- Guaranteed age bound (staleness) on reads
- Contrast: no-age-guarantee Eventual Consistency seen in Cassandra, Memcached

BAP Data Parallel: (E)SSP Regret Bound [Ho et al., 2013]

- **Goal:** minimize convex $f(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x})$

  (Example: Stochastic Gradient)

  - ***L***-Lipschitz, problem diameter bounded by ***F***²
  - Staleness ***s***, using ***P*** threads across all machines
  - Use step size $\eta_t = \frac{\sigma}{\sqrt{t}}$ with $\sigma = \frac{F}{L\sqrt{2(s+1)P}}$



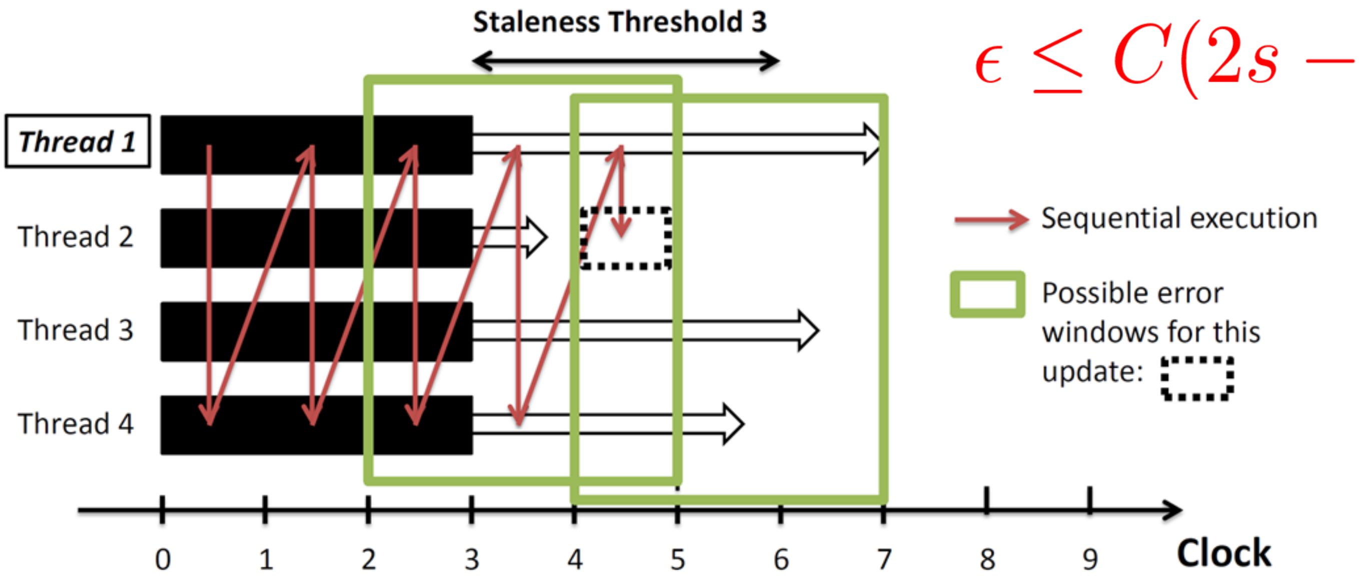
- **(E)SSP converges according to**
  - Where ***T*** is the number of iterations

$$R[\mathbf{X}] := \overbrace{\left[\frac{1}{T}\sum_{t=1}^{T} f_t(\tilde{\mathbf{x}}_t)\right] - f(\mathbf{x}^*)}^{\textbf{Difference between SSP estimate and true optimum}} \leq 4FL\sqrt{\frac{2(s+1)P}{T}}$$

- Note the RHS interrelation between (***L***, ***F***) and (***s***, ***P***)
  - An interaction between model and systems parameters
- Stronger guarantees on means and variances can also be proven

Intuition: Why does (E)SSP converge?

SSP approximates sequential execution



$$\epsilon \leq C(2s - 1)$$

- Number of missing updates bounded
  - Partial, but bounded, loss of serializability
- Hence numeric error in parameter also bounded
- Later in this tutorial – formal theorem

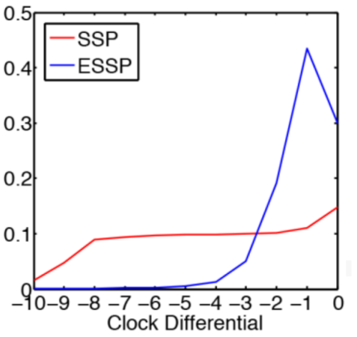
SSP versus ESSP: What is the difference?

- ESSP is a systems improvement over SSP communication
  - Same maximum staleness guarantee as SSP
  - Whereas SSP waits until the last second to communicate…
  - … ESSP communicates updates as early as possible

- What impact does ESSP have on convergence speed and stability?

BAP Data Parallel: (E)SSP Probability Bound

[Dai et al., 2015]



Let real staleness observed by system be γ_t

Let its mean, variance be $\mu_\gamma = \mathbb{E}[\gamma_t]$, $\sigma_\gamma = var(\gamma_t)$

Theorem: Given L-Lipschitz objective *f*$_t$ and stepsize h$_t$,

$$P\left[\underbrace{\frac{R\left[X\right]}{T}}_{\text{Gap between current estimate and optimum}} - \frac{1}{\sqrt{T}}\left(\eta L^2 + \frac{F^2}{\eta} + \underbrace{2\eta L^2 \mu_\gamma}_{\text{Penalty due to high avg. staleness } u_{stale}}\right) \geq \tau\right] \leq \exp\left\{\frac{-T\tau^2}{\underbrace{2\bar{\eta}_T \sigma_\gamma}_{\text{Penalty due to high staleness var. } \sigma_{stale}} + \frac{2}{3}\eta L^2 (2s+1) P\tau}\right\}$$

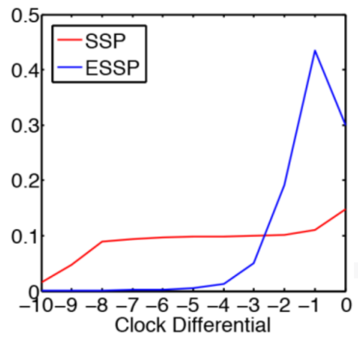
$$R[X] := \sum_{t=1}^{T} f_t(\tilde{x}_t) - f(x^*) \qquad \bar{\eta}_T = \frac{\eta^2 L^4 (\ln T + 1)}{T} = o(T)$$

Explanation: the (E)SSP distance between true optima and current estimate decreases exponentially with more iterations. *Lower staleness mean, variance μ_γ, σ_γ improve the convergence rate.*

Take-away: controlling staleness mean μ_γ, variance σ_γ (on top of max staleness s) is needed for faster ML convergence, which ESSP does.

BAP Data Parallel: (E)SSP Variance Bound

[Dai et al., 2015]



Theorem: the variance in the (E)SSP estimate is

$$\begin{aligned}\mathrm{Var}_{\,t+1} = \mathrm{Var}_{\,t} - 2\eta_t cov(\boldsymbol{x}_t, \mathbb{E}^{\Delta_t}[\boldsymbol{g}_t]) + \mathcal{O}(\eta_t \xi_t) \\ + \mathcal{O}(\eta_t^2 \rho_t^2) + \mathcal{O}^*_{\gamma_t}\end{aligned}$$

where

$$cov(\boldsymbol{a}, \boldsymbol{b}) := \mathbb{E}[\boldsymbol{a}^T \boldsymbol{b}] - \mathbb{E}[\boldsymbol{a}^T]\mathbb{E}[\boldsymbol{b}]$$

and $\mathcal{O}^*_{\gamma_t}$ represents 5th order or higher terms in γ_t

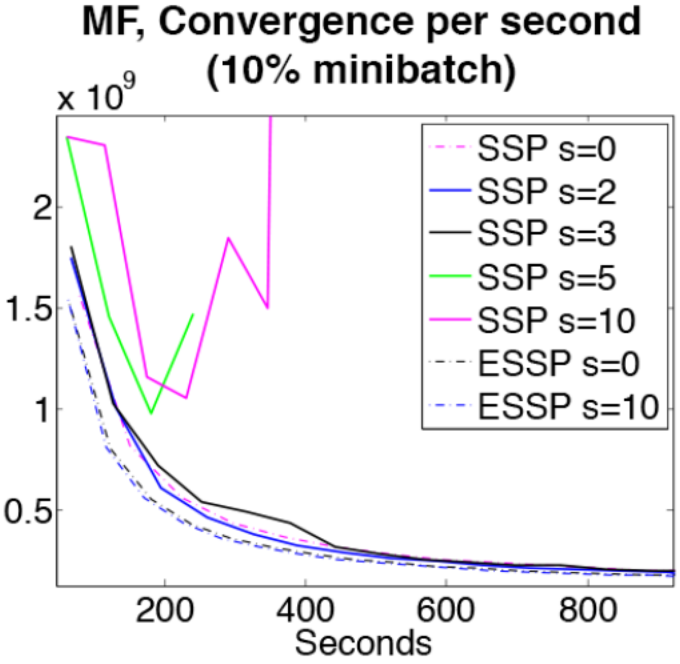
Explanation: The variance in the (E)SSP parameter estimate monotonically decreases when close to an optimum.

Lower (E)SSP staleness γ_t *=> Lower variance in parameter => Less oscillation in parameter => More confidence in estimate quality and stopping criterion.*

Take-away: Lower average staleness (via ESSP) not only improves convergence speed, but also yields better parameter estimates

ESSP vs SSP: Increased stability helps empirical performance

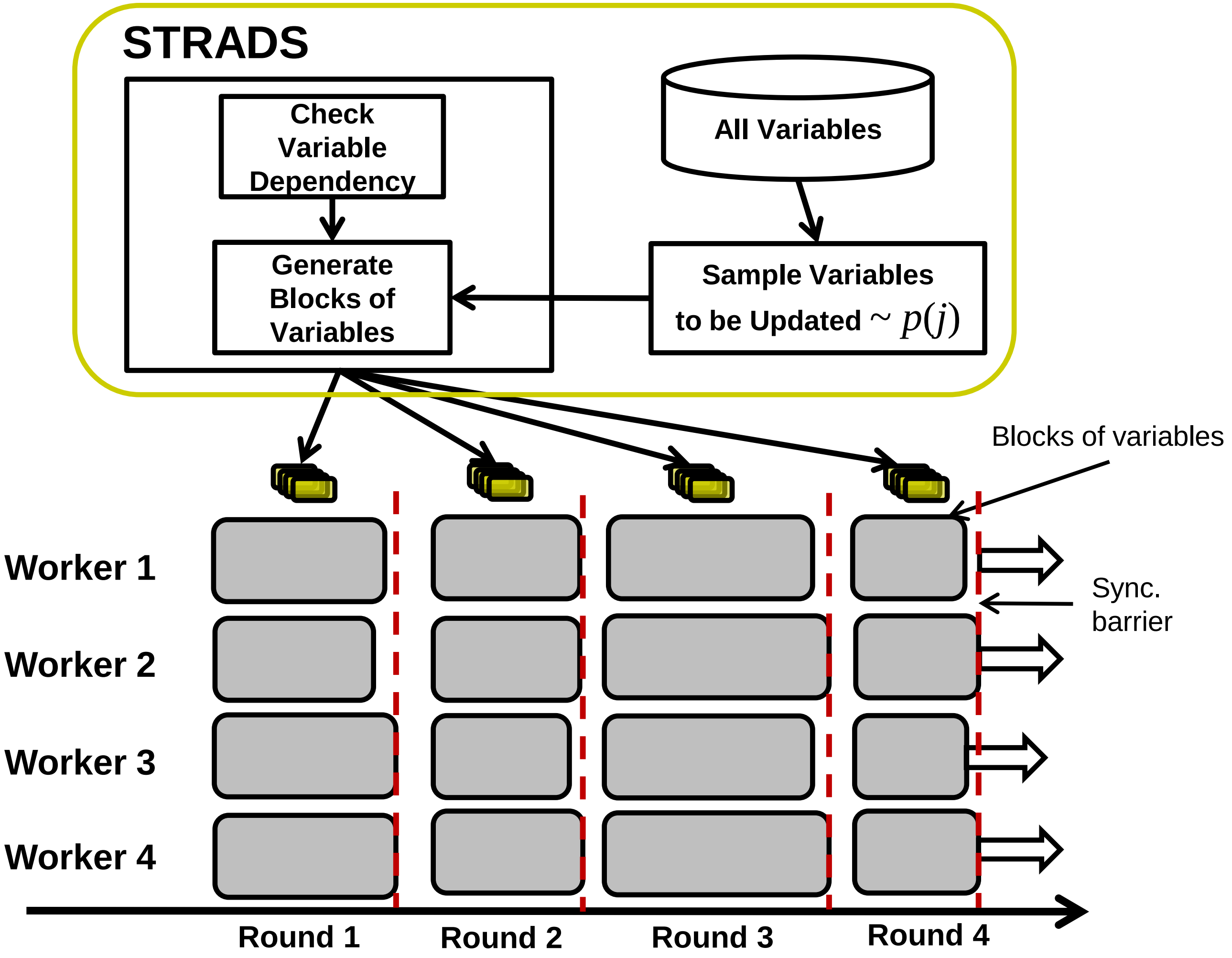
- Low-staleness SSP and ESSP converge equally well
- But at higher staleness, ESSP is more stable than SSP
  - ESSP communicates updates early, whereas SSP waits until the last second
  - ESSP better suited to real-world clusters, with straggler and multi-user issues



© Eric Xing @ CMU, 2015

Scheduled Model Parallel: Dynamic/Block Scheduling

[Lee et al. 2014, Kumar et al. 2014]



STRADS

Check Variable Dependency

Generate Blocks of Variables

All Variables

Sample Variables to be Updated $\sim p(j)$

Blocks of variables

Worker 1

Worker 2

Worker 3

Worker 4

Sync. barrier

Round 1

Round 2

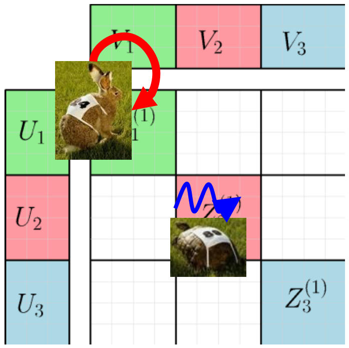
Round 3

Round 4

- **Priority Scheduling**

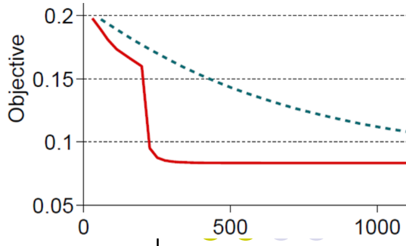
$$\{\beta_j\} \sim \left(\delta\beta_j^{(t-1)}\right)^2 + \eta$$

- **Block scheduling**



Scheduled Model Parallel:
Dynamic Scheduling Expectation Bound

[Lee et al. 2014]



- **Goal:** solve sparse regression problem $\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_j |\beta_j|$
  - Via coordinate descent over "SAP blocks" $X^{(1)}, X^{(2)}, \ldots, X^{(B)}$
    - $X^{(b)}$ are the data columns (features) in block (b)
  - P parallel workers, M-dimensional data
  - ρ = Spectral Radius[BlockDiag[$(X^{(1)})^T X^{(1)}, \ldots, (X^{(t)})^T X^{(t)}$]]; this block-diagonal matrix quantifies the maximum level of correlation (and hence problem difficulty) within all the SAP blocks $X^{(1)}, X^{(2)}, \ldots, X^{(t)}$
- **SAP converges according to**
  - Where t is # of iterations

$$\mathbb{E}\left[\overbrace{f(X^{(t)}) - f(X^*)}^{\text{Gap between current parameter estimate and optimum}}\right] \le \frac{\overbrace{\mathcal{O}(M)}^{\text{SAP explicitly minimizes } \rho \text{, ensuring as close to } 1/P \text{ convergence as possible}}}{P - \boxed{\frac{\mathcal{O}(P^2\rho)}{M}}} \frac{1}{t} = \mathcal{O}\left(\frac{1}{Pt}\right)$$

- **Take-away:** SAP minimizes ρ by searching for feature subsets $X^{(1)}, X^{(2)}, \ldots, X^{(B)}$ without cross-correlation => as close to P-fold speedup as possible

Scheduled Model Parallel:

Dynamic Scheduling Expectation Bound is near-ideal

[Xing et al. 2015]

Let $S^{ideal}()$ be an ideal model-parallel schedule

Let $\beta_{ideal}^{(t)}$ be the parameter trajectory due to ideal scheduling

Let $\beta_{dyn}^{(t)}$ be the parameter trajectory due to SAP scheduling

Theorem: After *t* iterations, we have

$$E[||\beta_{ideal}^{(t)} - \beta_{dyn}^{(t)}||] \leq C \frac{2M}{(t+1)^2} \mathbf{X}^\top \mathbf{X}$$

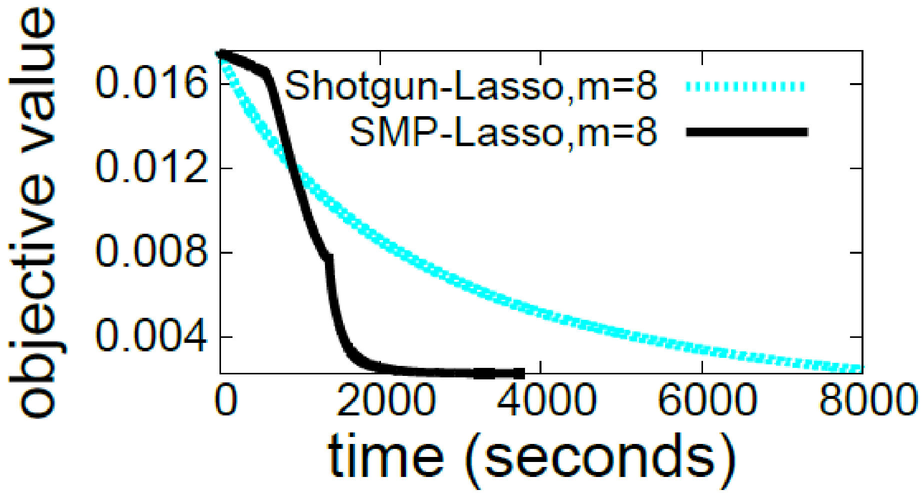
Explanation: *Under dynamic scheduling, algorithmic progress is nearly as good as ideal model-parallelism.*

Intuitively, this is because both ideal and SAP model-parallelism minimize the parameter dependencies between parallel workers.

Scheduled Model Parallel:

Dynamic Scheduling Empirical Performance

- Dynamic Scheduling for Lasso regression (SMP-Lasso): almost-ideal convergence rate, much faster than random scheduling (Shotgun-Lasso)



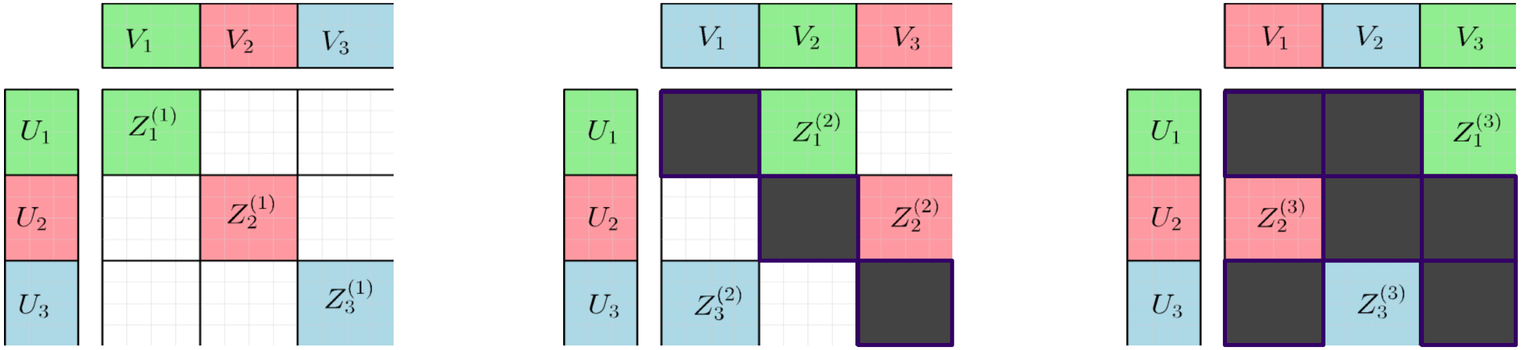
© Eric Xing @ CMU, 2015

Scheduled Data+Model Parallel:

Block-based Scheduling (with load balancing)

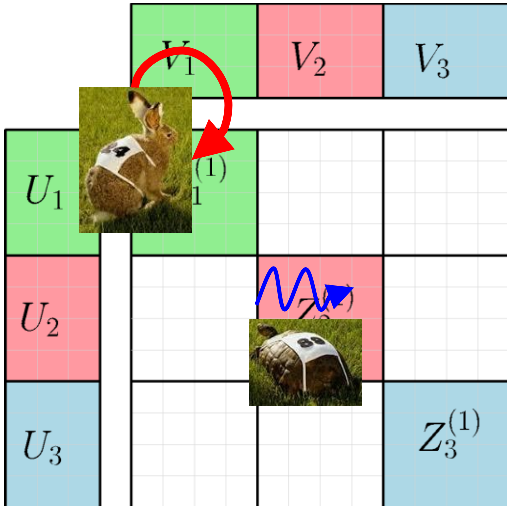
[Kumar et al. 2014]

Partition data & model into *d* × *d* blocks
Run different-colored blocks in parallel



V_1 V_2 V_3
U_1 U_2 U_3
$Z_1^{(1)}$ $Z_2^{(1)}$ $Z_3^{(1)}$
V_1 V_2 V_3
U_1 U_2 U_3
$Z_1^{(2)}$ $Z_2^{(2)}$ $Z_3^{(2)}$
V_1 V_2 V_3
U_1 U_2 U_3
$Z_1^{(3)}$ $Z_2^{(3)}$ $Z_3^{(3)}$

Blocks with less data/para or experience less straggling run more iterations
Automatic load-balancing + better convergence



Scheduled Data+Model Parallel:

Block-based Scheduling Variance Bound 1

[Kumar et al. 2014]

- Variance between iterations S_n+1 and S_n is:

$$
\begin{aligned}
&Var(\Psi_{S_{n+1}}) \\
&=Var(\Psi_{S_n}) - \boxed{2\eta_{S_n}} \sum_{i=1}^{w} n_i \Omega_0^i Var(\psi_{S_n}^i) \\
&\quad - \boxed{2\eta_{S_n}} \sum_{i=1}^{w} n_i \Omega_0^i CoVar(\psi_{S_n}^i, \bar{\delta}_{S_n}^i) + \boxed{\eta_{S_n}^2} \sum_{i=1}^{w} n_i \Omega_1^i + \boxed{\mathcal{O}(\Delta_{S_n})}
\end{aligned}
$$

- Explanation:
  - higher order terms (red) are negligible
  - => parameter variance decreases every iteration
- Every iteration, the parameter estimates become more stable

Scheduled Data+Model Parallel:

Block-based Scheduling Variance Bound 2

[Kumar et al. 2014]

- Intra-block variance: Within blocks, suppose we update the parameters ψ using n_i data points. Then, variance of ψ after those n_i updates is:

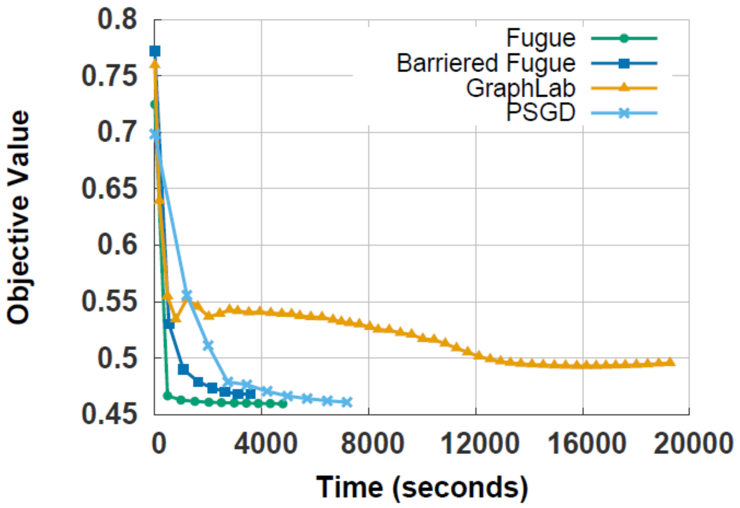
$$
\begin{aligned}
Var(\psi^{t+n_i}) =& Var(\psi^t) - 2\eta_t n_i \Omega_0 (Var(\psi^t)) \\
& - 2\eta_t n_i \Omega_0 CoVar(\psi_t, \bar{\delta}_t) + \boxed{\eta_t^2 n_i \Omega_1} \\
& \boxed{+ \underbrace{\mathcal{O}(\eta_t^2 \rho_t) + \mathcal{O}(\eta_t \rho_t^2) + \mathcal{O}(\eta_t^3) + \mathcal{O}(\eta_t^2 \rho_t^2)}_{\Delta_t}}
\end{aligned}
$$

- Explanation:
  - Higher order terms (red) are negligible
  - => doing more updates within each block decreases parameter variance, leading to more stable convergence
- Load balancing by doing extra updates is effective

Scheduled Data+Model Parallel:
Block-Scheduling Empirical Performance

- Slow-worker Agnostic Block-Scheduling (Fugue) faster than:
  - Embarrassingly Parallel SGD (PSGD)
  - Non slow-worker Agnostic Block-Scheduling (Barriered Fugue)
- Slow-worker Agnostic Block-Scheduling converges to a better optimum than asynchronous GraphLab
  - Reason: more stable convergence due to block-scheduling

- Task: Imagenet Dictionary Learning
  - 630k images, 1k features

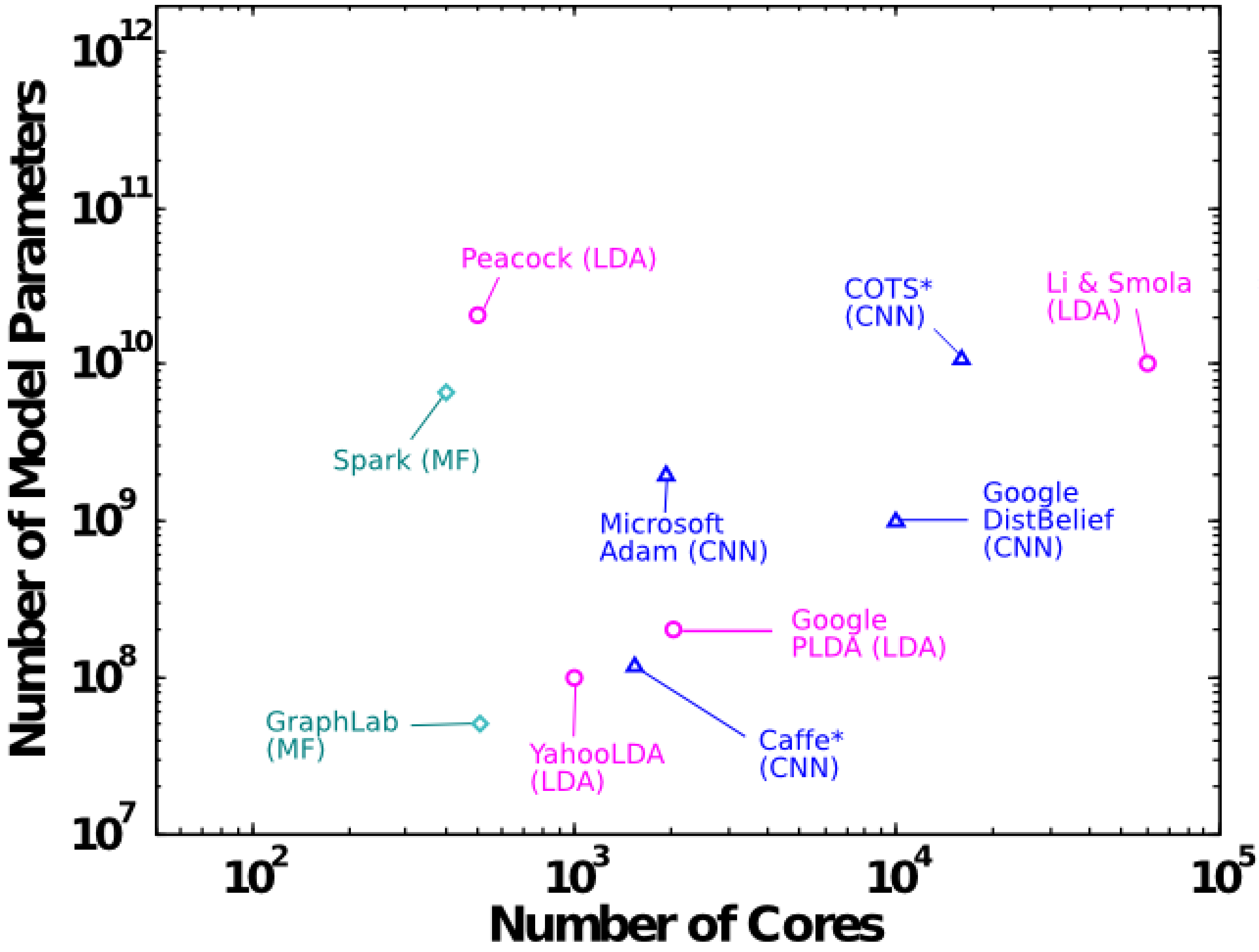


BAP Model-Parallel Guarantees

- Model-parallel under synchronous setting:
  - Dynamic scheduling
  - Slow-worker block-based scheduling
- Synchronous slow-worker problem solved by:
  - Load balancing (for dynamic scheduling)
  - Allow additional iters while waiting for other workers (slow-worker scheduling)
- Work in progress: theoretical guarantees for bounded-async model-parallel execution
  - Intuition: model-parallel sub-problems are nearly independent (thanks to scheduling)
  - Perhaps better per-iteration convergence than bounded-async data-parallel learning?

Open Research Issues and Topics

The Landscape of Big ML



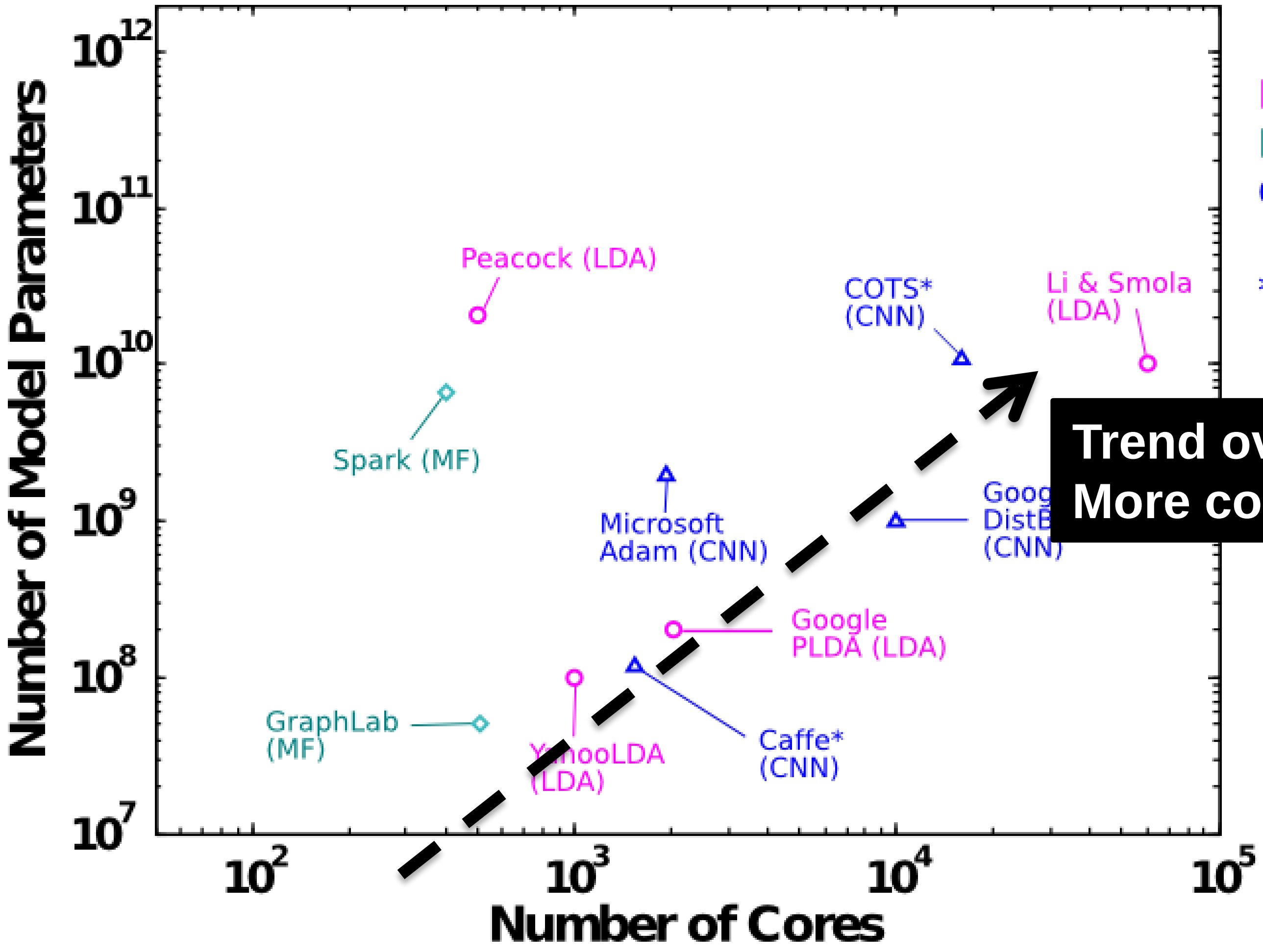
LDA - Topic Model

MF - Matrix Factorization

CNN - Convolutional Neural Network

*GPU cores

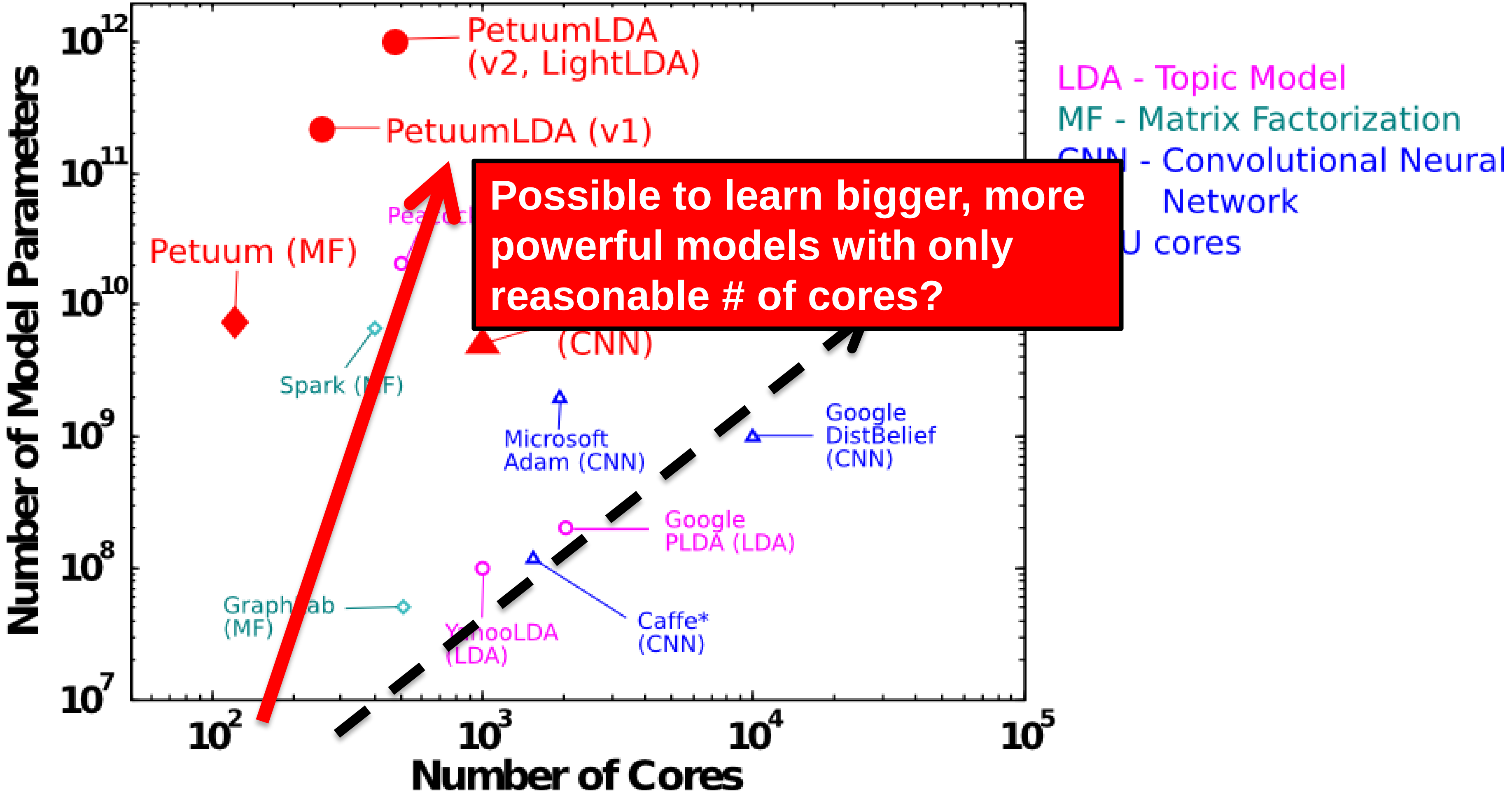
The Landscape of Big ML



LDA - Topic Model
MF - Matrix Factorization
CNN - Convolutional Neural Network
*GPU cores

Trend over last 5 years:
More cores, bigger models

The Landscape of Big ML



Possible to learn bigger, more powerful models with only reasonable # of cores?

Issue: When is Big Data useful?

- Negative examples
  - “Simple” regression and classification models, with fixed parameter size
  - **Intuition:** decrease estimator variance has diminishing returns with more data. Estimator eventually becomes “good enough”, and additional data/computation is unnecessary
- Positive examples
  - Topic models (internet/tech industry)
  - DNNs (Google, Baidu, Microsoft, Facebook, etc.)
  - Collaborative filtering (internet/tech industry)
  - Personalized models
  - Industry practitioners sometimes increase model size with more data
- Conjecture: how much data is useful really depends on model size/capacity

Issue: Are Big Models useful?

- In theory
  - Possibly, but be careful not to over-extend
- Beware “statistical strength”
  - *“When you have large amounts of data, your appetite for hypotheses tends to get even larger. And if it’s growing faster than the statistical strength of the data, then many of your inferences are likely to be false. They are likely to be white noise.”* **–Michael Jordan**

- In practice
  - Some success stories - could there be theory justification?
- Many topics in topic models
  - Capture long-tail effects of interest; improved real-world task performance
- Many parameters in DNNs
  - Improved accuracy in vision and speech tasks
  - Publicly-visible success (e.g. Google Brain)

Issue: Inference Algorithms, or Inference Systems?

- View: focus on inference algorithm
- Scale up by refining the algorithm
  - Given fixed computation, finish inference faster
- Some examples
  - Quasi-Newton algorithms for optimization
  - Fast Gibbs samplers for topic models **[Yao et al. 2009, Li et al. 2014, Yuan et al. 2015, Zheng et al, 2015]**
  - Locality sensitive hashing for graphical models **[Ahmed et al. 2012]**

- View: focus on distributed systems for inference
- Scale up by using more machines
  - Not trivial: real clusters are imperfect and unreliable; Hadoop not a fix-all
- Some examples
  - Spark
  - GraphLab
  - Petuum

Issue: Theoretical Guarantees and Empirical Performance

- View: establishing theoretical guarantees gives practitioners confidence
  - Motivated by empirical science, where guarantees are paramount
- Example: Lasso sparsistency and consistency **[Wainwright, 2009]**
  - Theory predicts how many samples n needed for a Lasso problem with p dimensions and k non-zero elements
  - Simulation experiments show very close match with theory
  - Is there a way to analyze more complex models?

- View: empirical, industrial evidence can provide strong driving force for experimental research
  - Motivated by industrial practice, particularly at internet companies
- Example: AB testing in industry
  - Principled means of testing new algorithms, feature engineering; by experimenting on user base
  - Determine if new method makes a significant difference to click-through rate, user adoption, etc.

Open research topics

- Future of data-, model-parallelism, and other ML properties
  - New properties, principles still undiscovered
  - Potential to accelerate ML beyond naive strategies
- Deep analysis of BigML systems still limited to few ML algos
  - Model of ML execution under error due to imperfect system?
- How to express more ML algorithms in table form (Spark, Petuum), or graph form (GraphLab)
  - Tree-structured algorithms? Infinite-dimensional Bayesian nonparametrics?
  - What are the key elements of a generic ML programming interface?

Acknowledgements

SAILING LAB

Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

Jin Kyu Kim

Seunghak Lee

Jinliang Wei

Wei Dai

Pengtao Xie

Xun Zheng

Abhimanu Kumar

www.sailing.cs.cmu.edu

$$$:

Garth Gibson

Gregory R. Ganger
Professor Electrical & Computer Engineering
Director of the Parallel Data Lab
Carnegie Mellon University
Greg Ganger

Phillip Gibbons

James Cipar

Thank You!