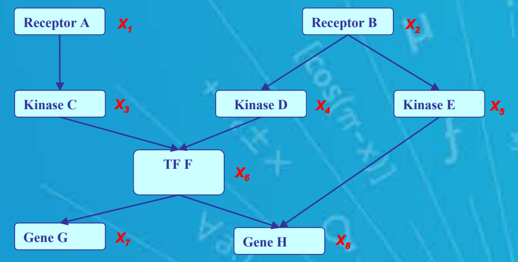
Probabilistic Graphical Models

Scalable algorithms and systems for learning, inference and prediction

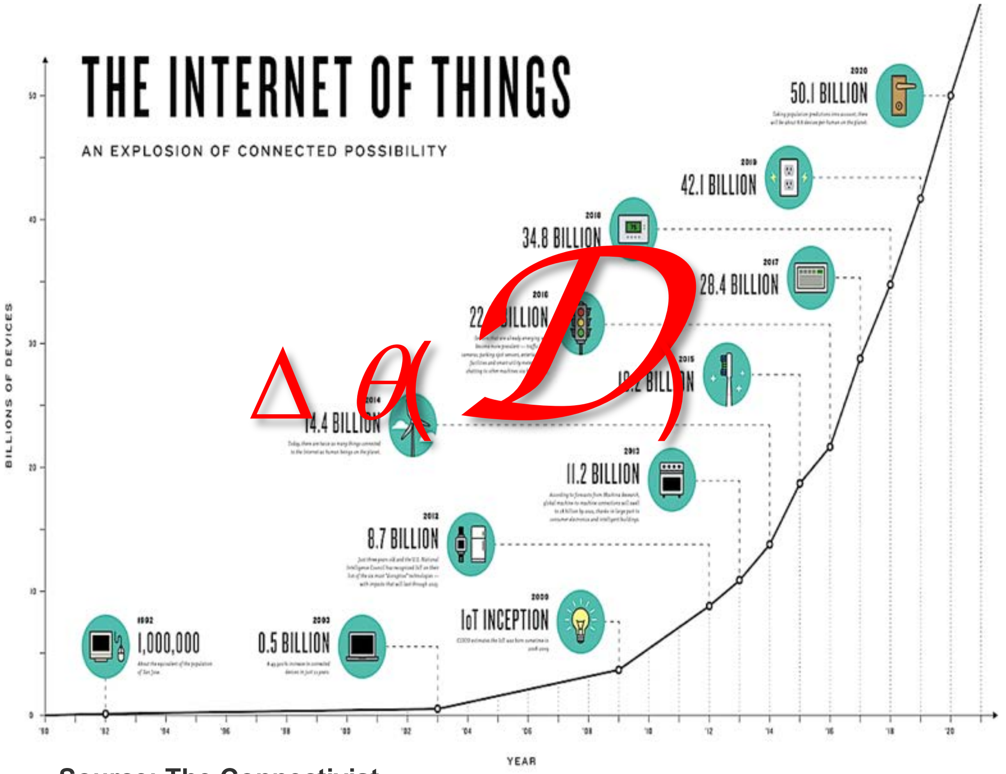
Qirong Ho

Lecture 26, April 22, 2020



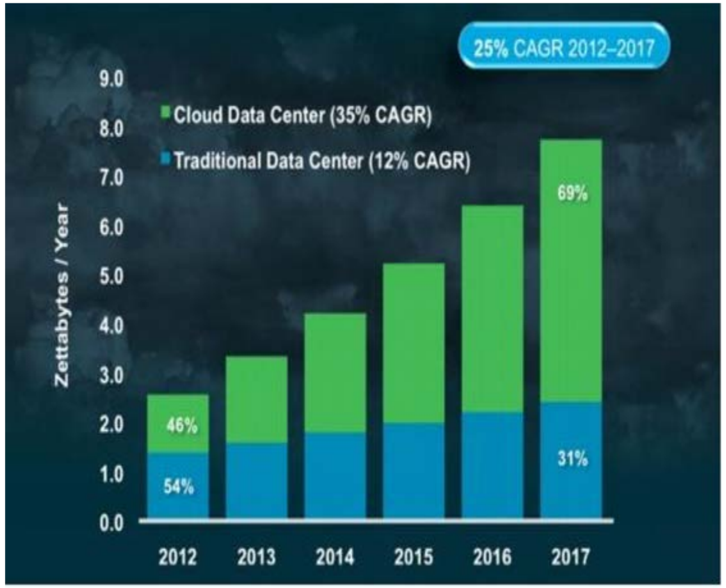
Reading: see class homepage

Challenge 1 – Massive Data Scale



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

Source: The Connectivist



Source: Cisco Global Cloud Index

Challenge 1 – Massive Data Scale

facebook

1B+ USERS
30+ PETABYTES

WIKIPEDIA
The Free Encyclopedia

32 million
pages

YouTube

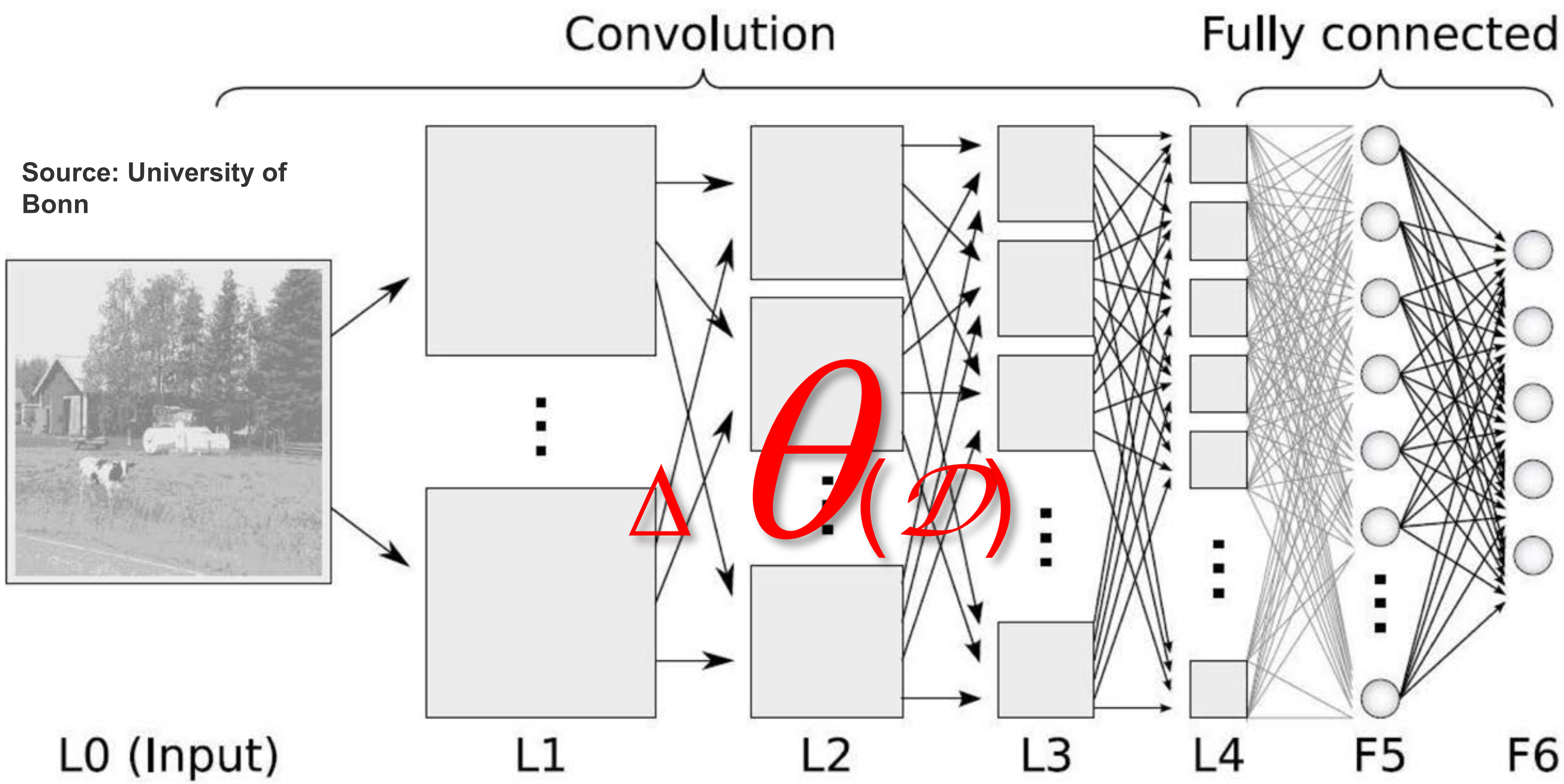
100+ hours video
uploaded every minute

twitter

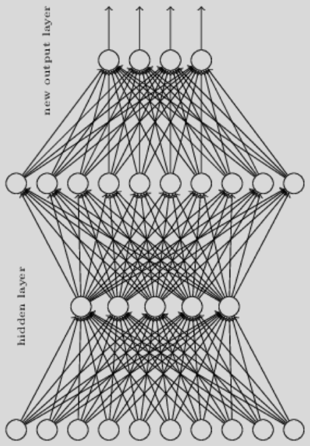
645 million users
500 million tweets / day

Challenge 2 – Gigantic Model Size

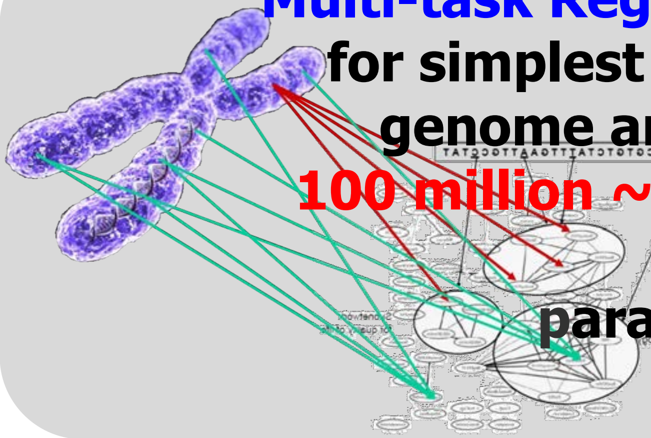
Source: University of Bonn



Challenge 2 – Gigantic Model Size



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



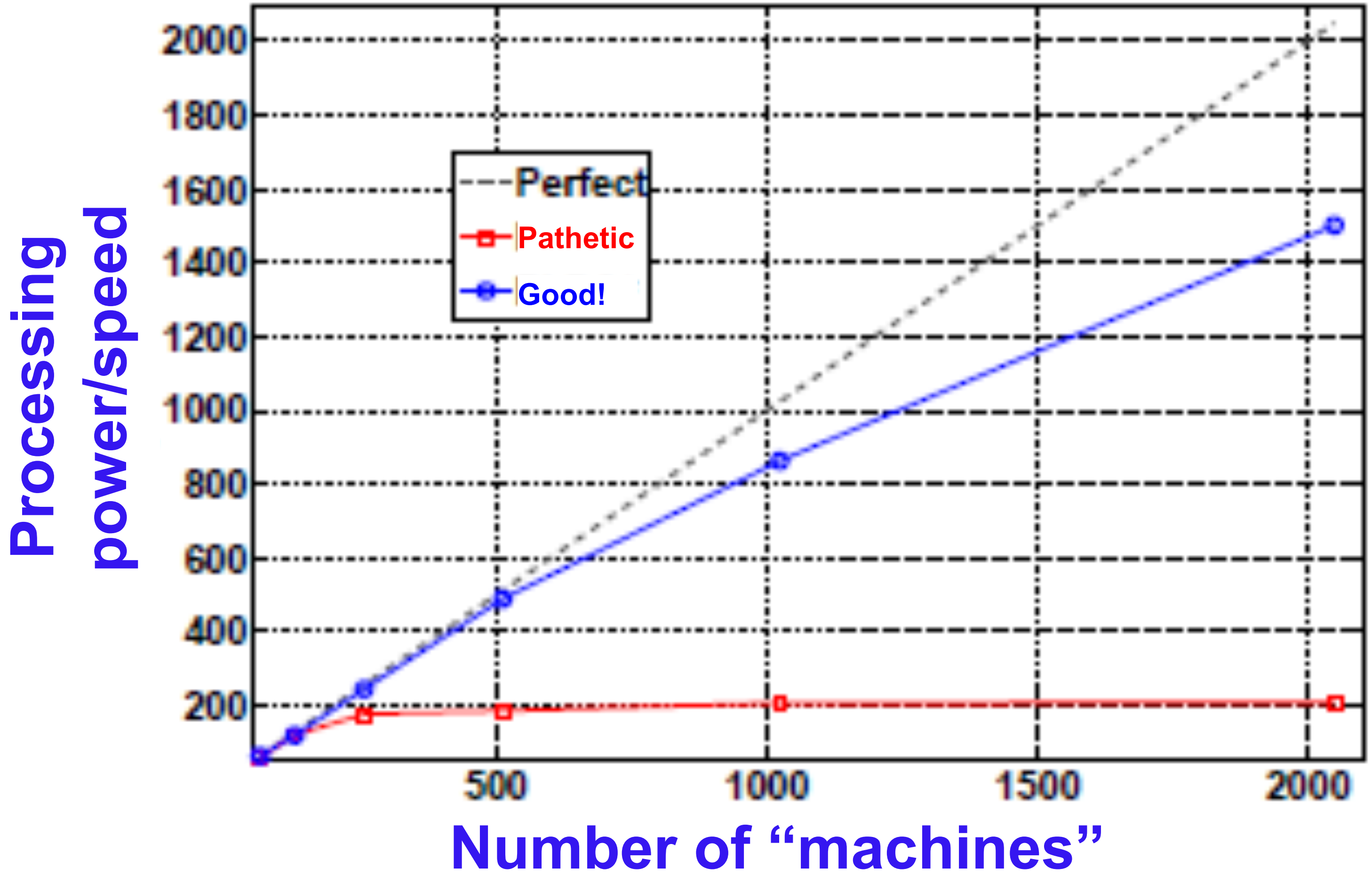
Multi-task Regression
for simplest whole-genome analysis:
100 million ~ 1 Billion
model parameters

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

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

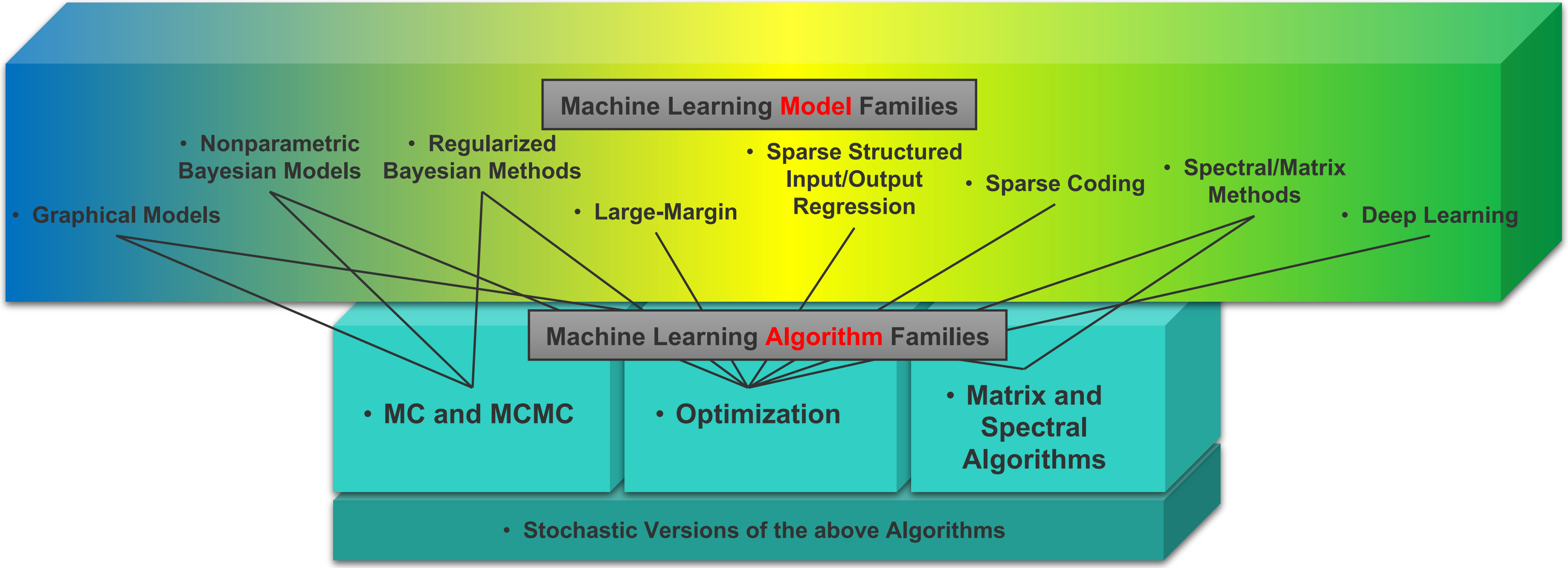
NETFLIX

The Scalability Challenge



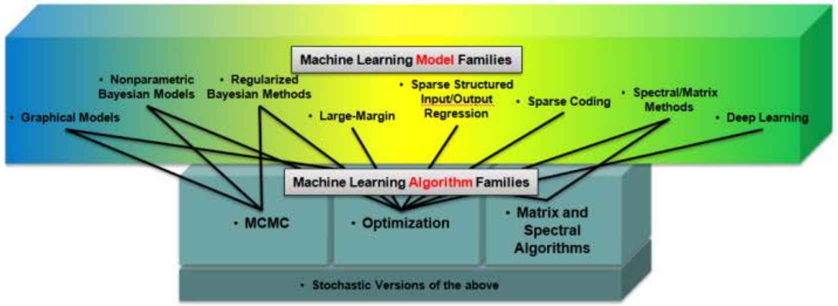
A "Classification" of ML Models and Tools

- 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)

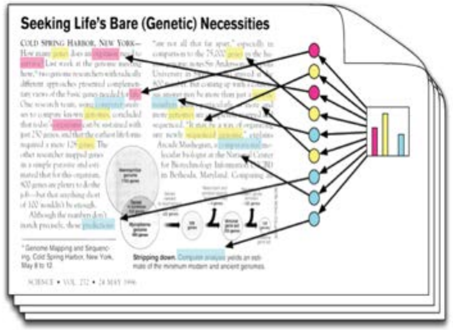


A “Classification” of ML Models and Tools

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

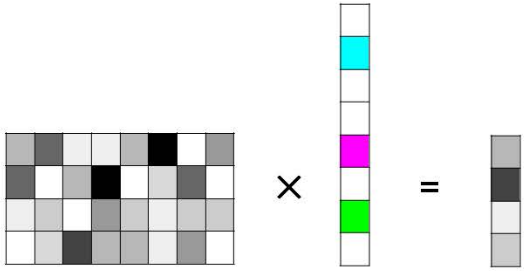


Probabilistic Programs



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

Optimization Programs

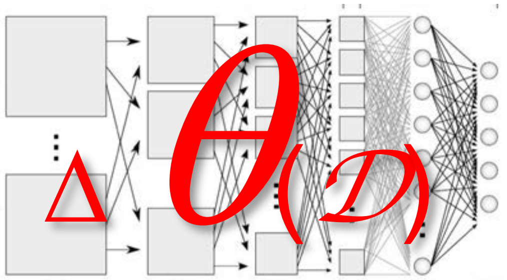


$$\sum_{i=1}^{N}\|y_i - X_i\beta\|_2^2 + \lambda\sum_{j=1}^{D}|\beta_j|$$

Iterative-convergent view of ML

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

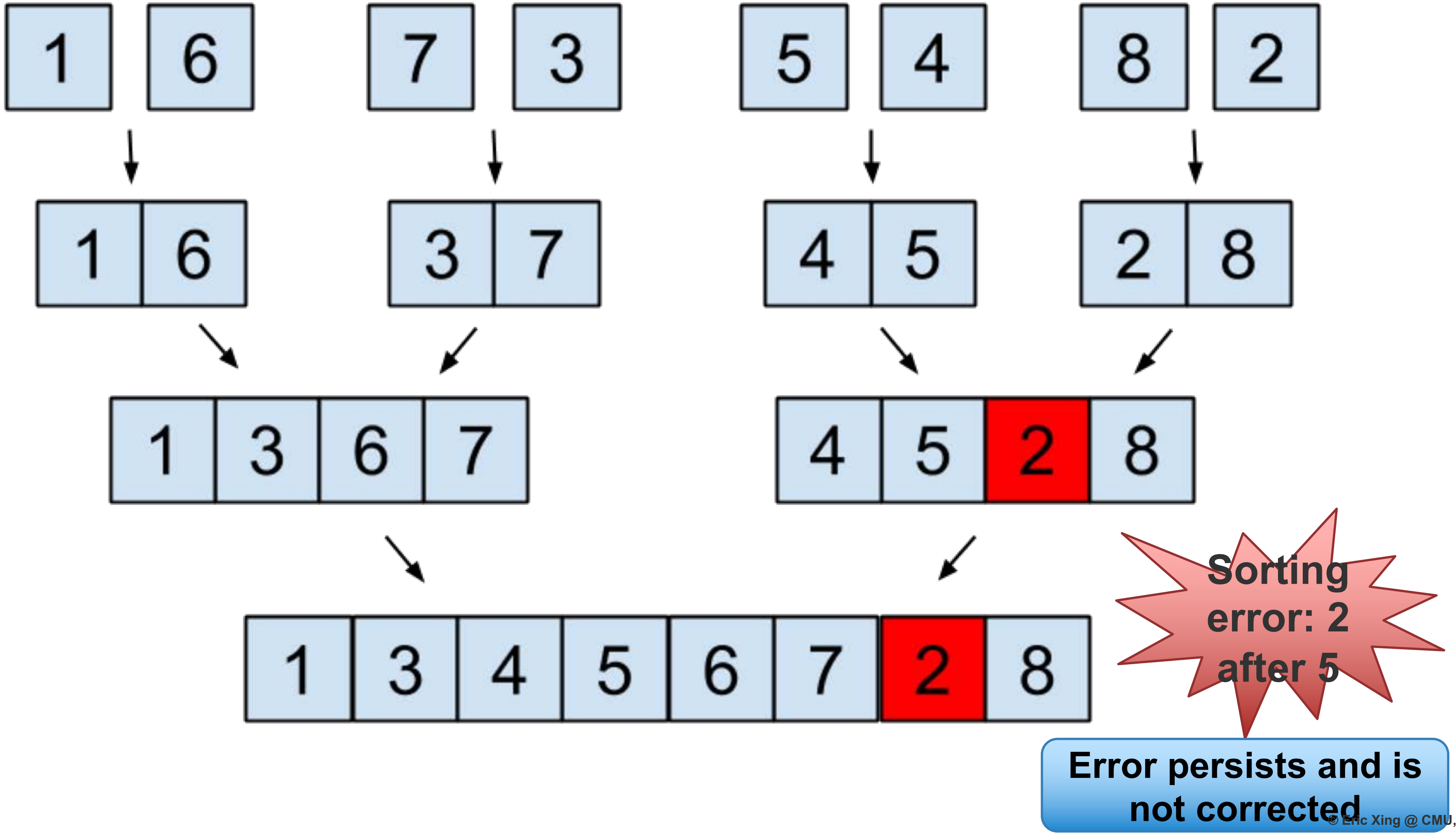
New Model = Old Model + Update(Data)



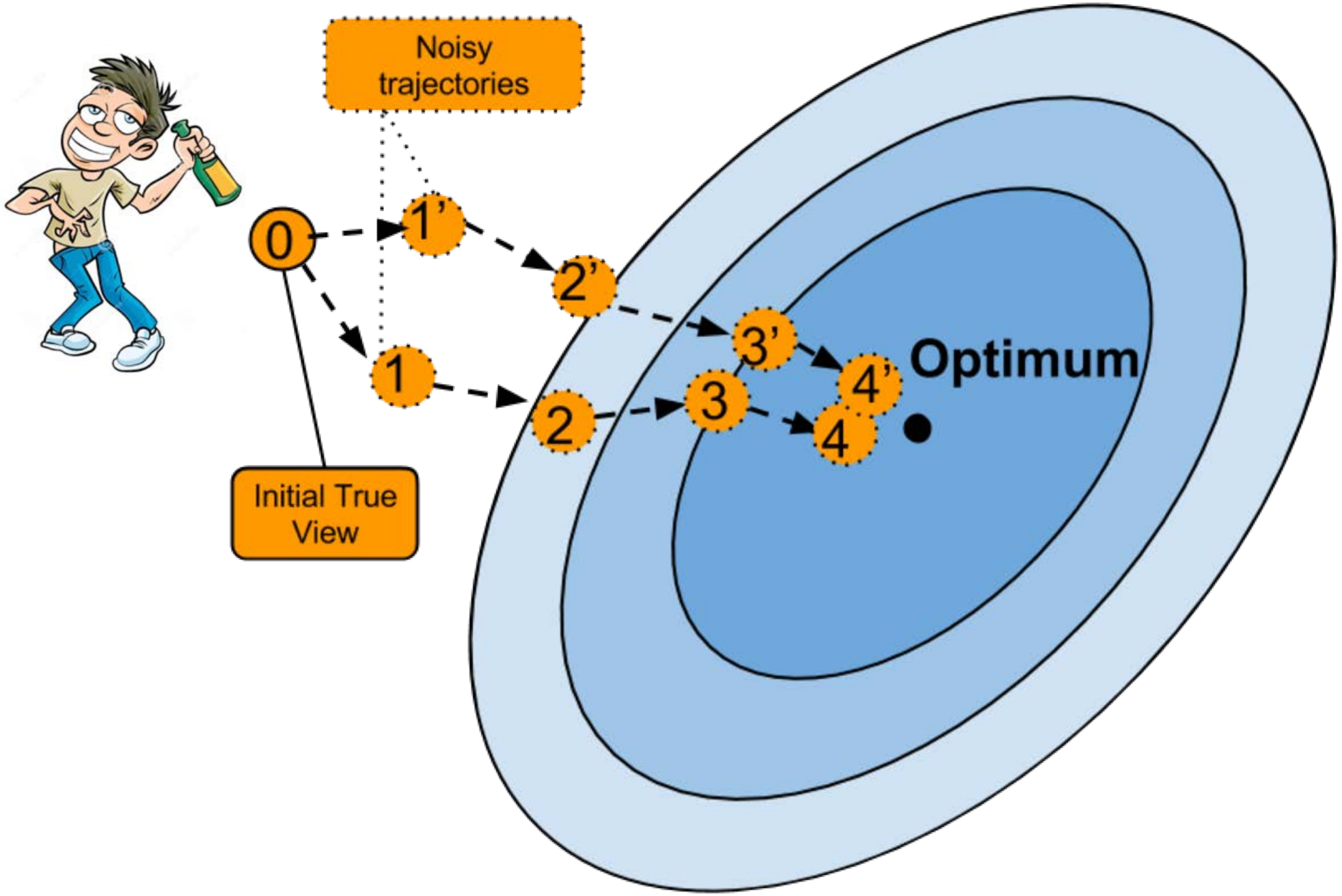
- ML models solved via iterative-convergent ML algorithms
  - Iterative-convergent algorithms repeat until θ is stationary. Examples:
    - Probabilistic programs: MC, MCMC, Variational Inference
    - Optimization programs: Stochastic Gradient Descent, ADMM, Proximal Methods, Coordinate Descent

Most algorithms need operational correctness …

Example: Merge sort



… but ML Algorithms can Self-heal



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()
```

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

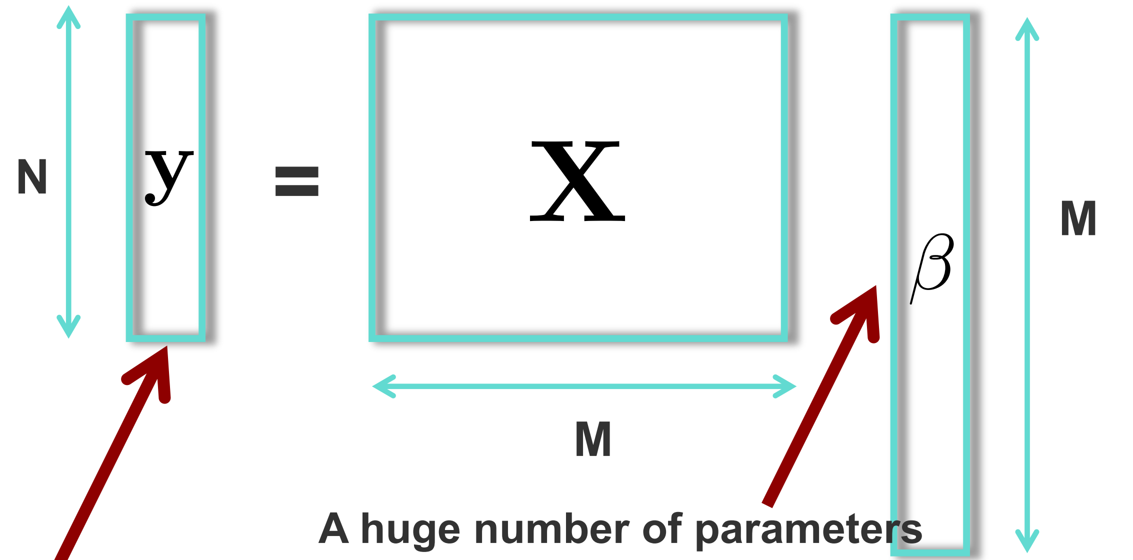
```
  doOtherThings()
}
```

This computation needs to be parallelized!

Challenge

- Optimization programs:

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



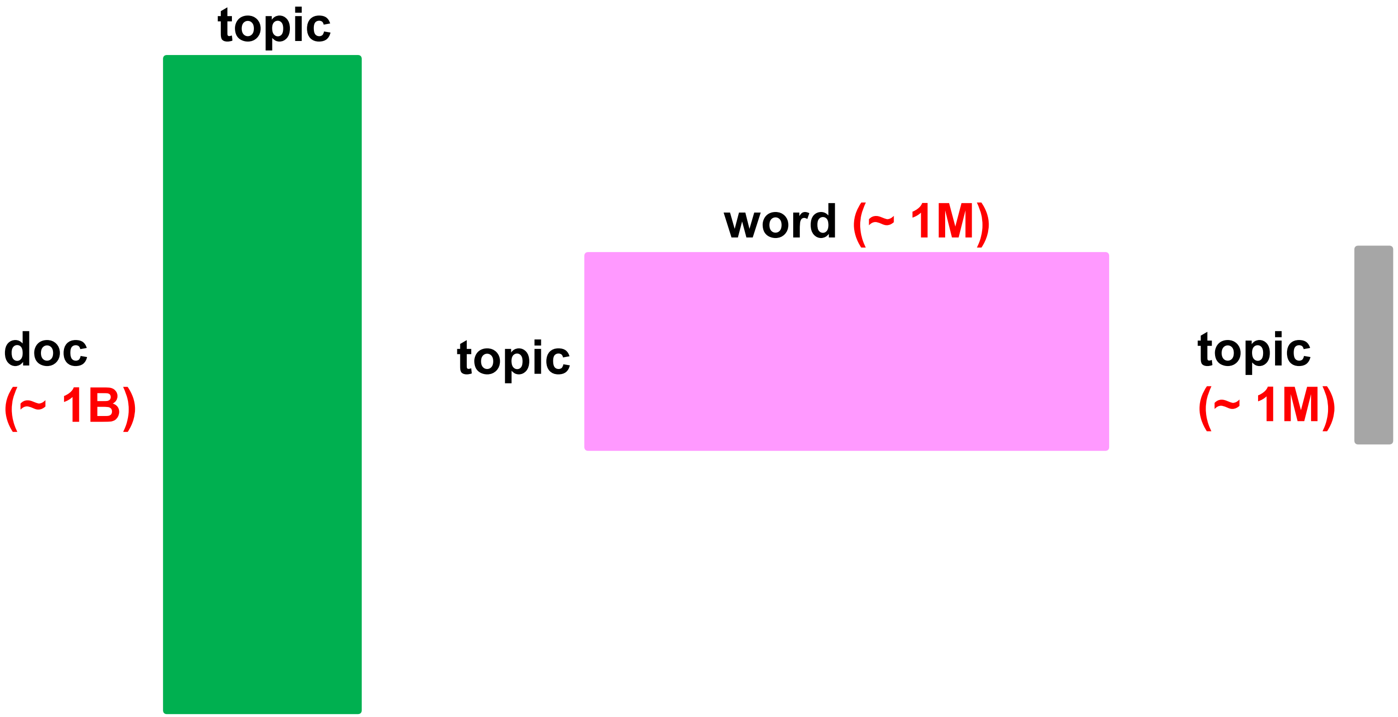
A huge volume of data (e.g.) N = 1B

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

Challenge

- Probabilistic programs

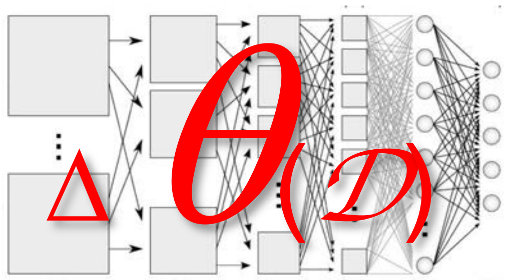
$$z_{ij} \sim p(z_{ij} = k | x_{ij}, \delta_i, B) \propto (\boxed{\delta}_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + \boxed{B}_{k,x_{ij}}}{V\beta + \boxed{\sum_{v=1}^{V} B_{k,v}}}$$



Parallelization Strategies

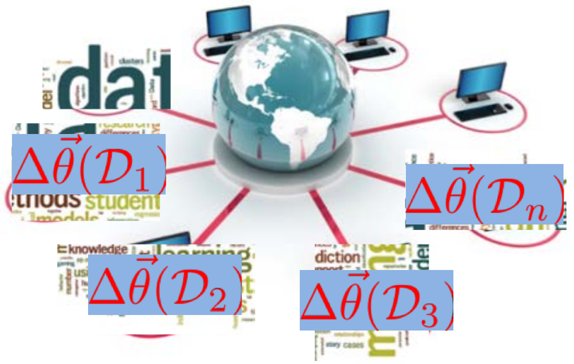
$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

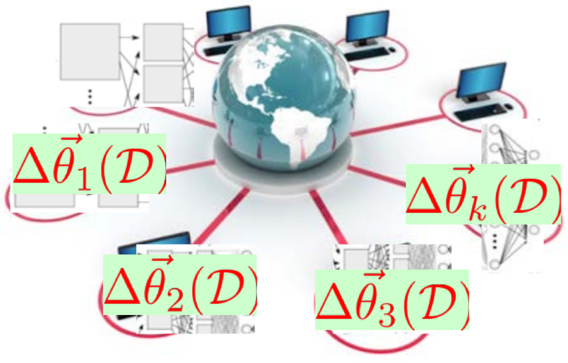
New Model = Old Model + Update(Data)



Data Parallel

Model Parallel





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

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

Optimization & MCMC Algorithms

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

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

Example Optimization Program: Sparse Linear Regression

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

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 $\Omega(\boldsymbol{\beta})$:

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

Sparsity

$$\Omega_{group}(\boldsymbol{\beta}) = \sum_{\mathbf{g}\in G}\|\boldsymbol{\beta}_{\mathbf{g}}\|_2$$

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

$$\Omega_{tree}(\boldsymbol{\beta})$$

$$\Omega_{overlap}(\boldsymbol{\beta})$$

Structured sparsity (sparsity + structured information)

© Eric Xing @ CMU, 2005-2020

Algorithm I:
Stochastic Gradient Descent

- Consider an optimization problem:

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

- Classical gradient descent:

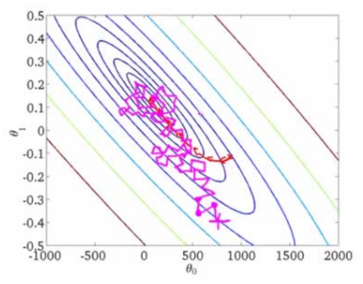
$$x^{(t+1)} \leftarrow x^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_x f(x^{(t)}, d_i)$$

- 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



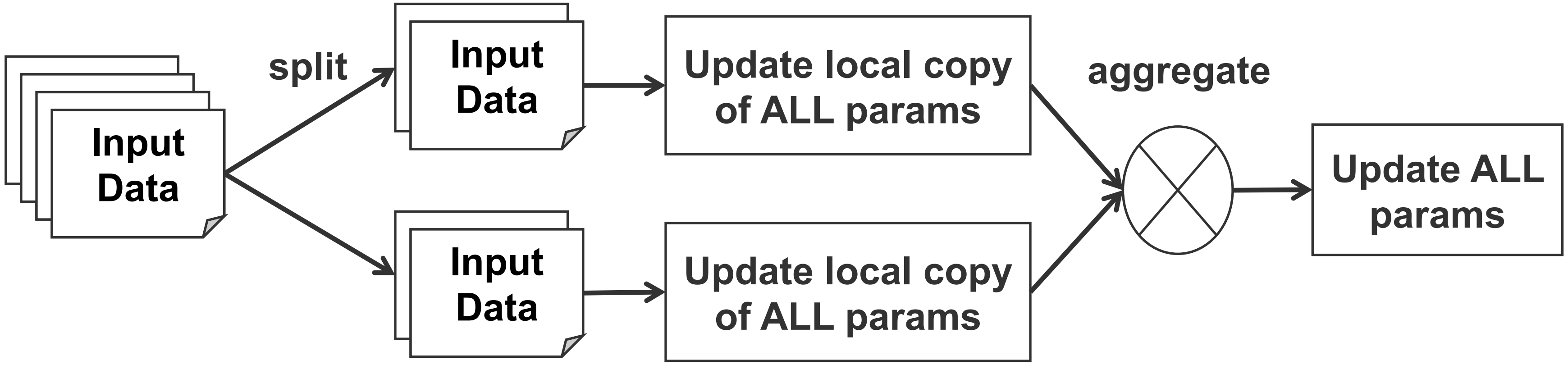
- Traditional SGD compute gradients based on a single sample

- 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

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

- Parallel SGD [Zinkevich et al., 2010]



- 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 f_t and step size η_t,

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

  - where $R[X] := \sum_{t=1}^{T} f_t(\tilde{x}_t) - f(x^*)$
  - 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

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

  - 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

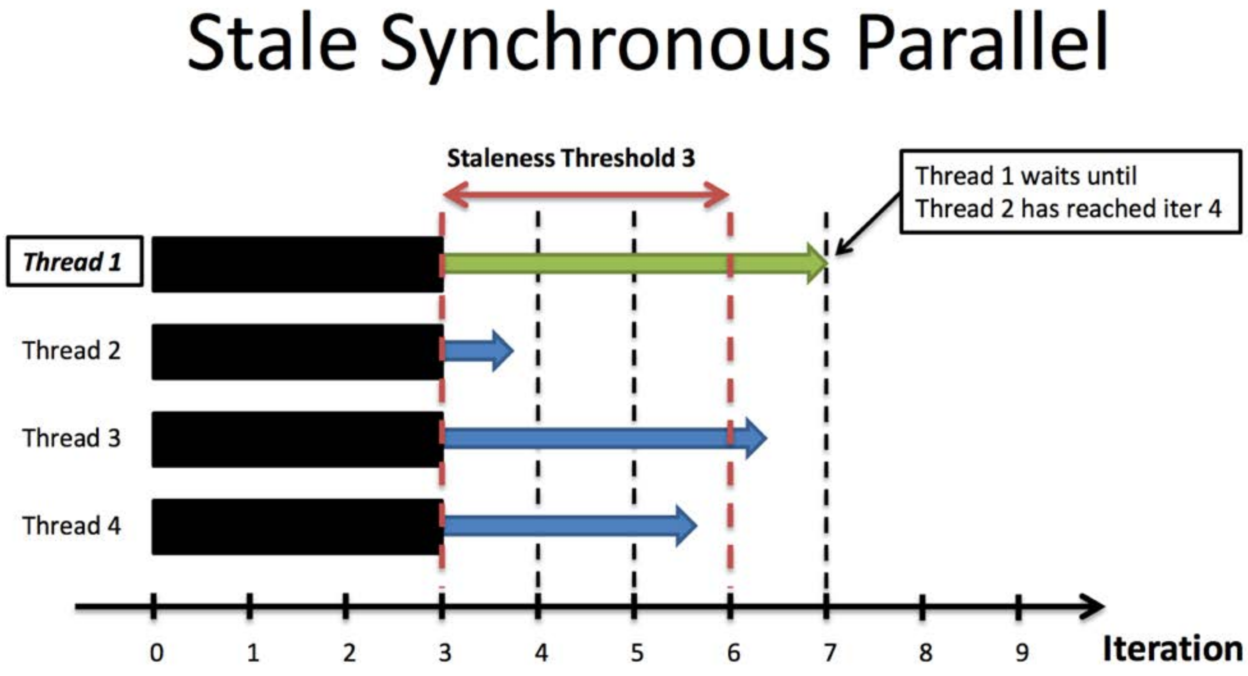
- Intuition: variance of the parameter estimate decreases near the optimum
  - But delay ε_t increases parameter variance => instability during convergence
- Distributed systems have much higher average delay, compared to single machine

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



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

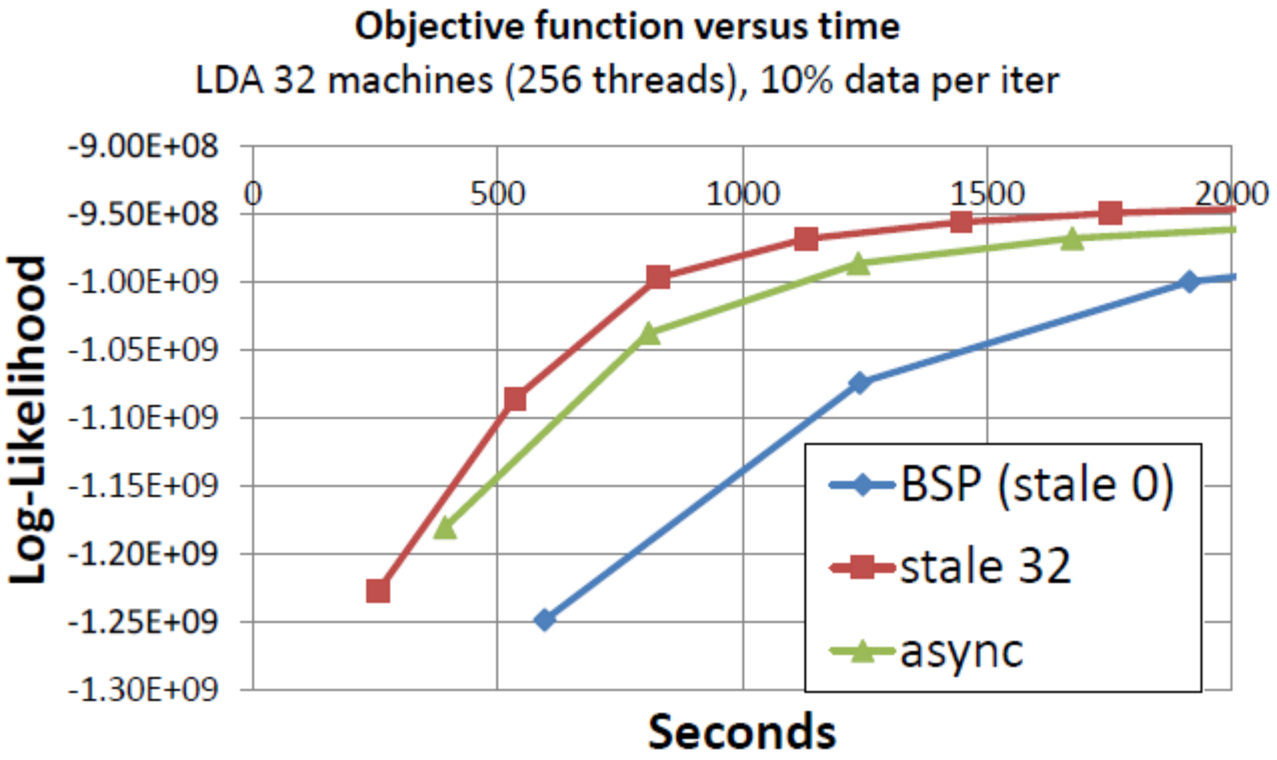
$$\beta^{(t)} = \beta^{(t-1)} + 2(y_i - X_i\beta^{(t-1)})X_i^\top$$

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

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

  - 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

$\beta_1 \quad \beta_2 \quad \beta_3 \quad ------ \quad \beta_J$

2st iteration

$\beta_1 \quad \beta_2 \quad \beta_3 \quad ------ \quad \beta_J$

- **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

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

- 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

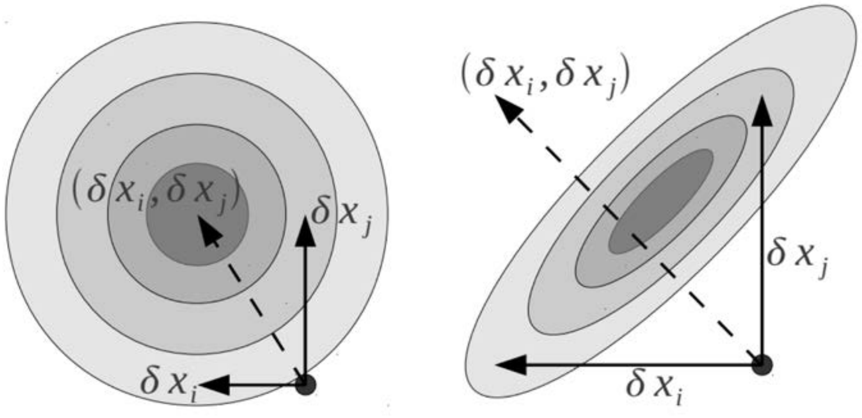
[Bradley et al. 2011]

- 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 $P \leq \frac{d}{2\rho}$ 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_i 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

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

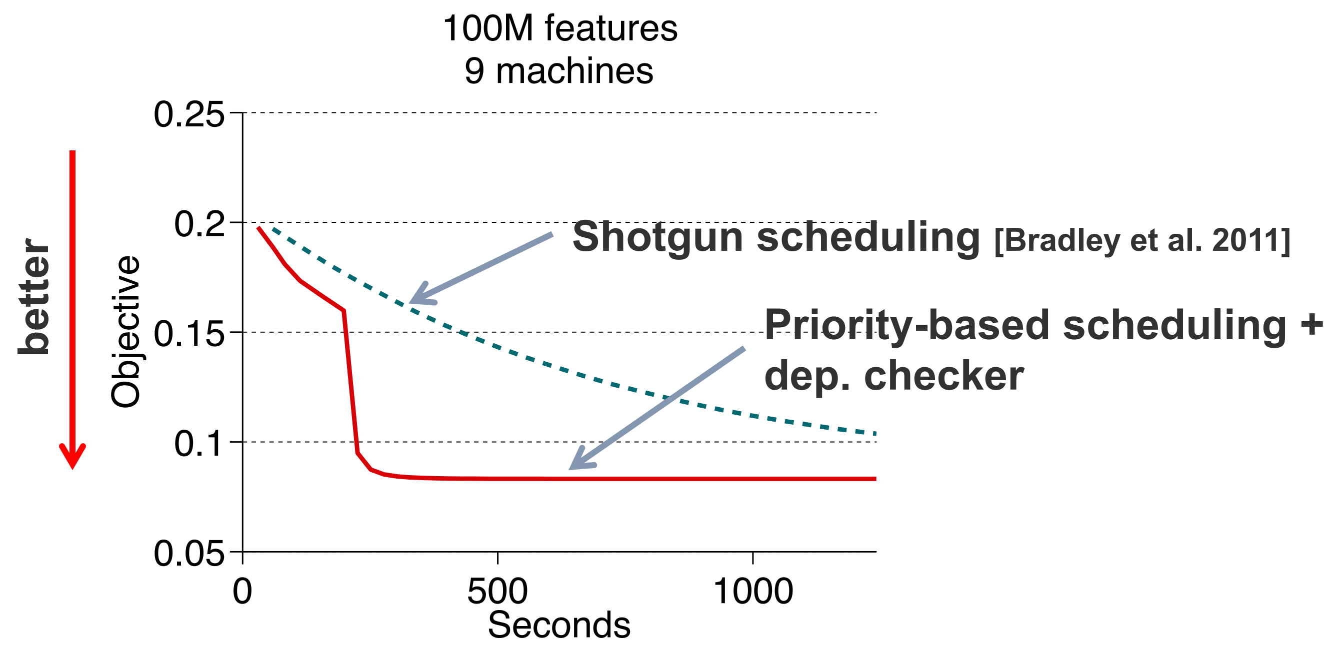
- Update step:
  - For all β_j chosen in Schedule step, in parallel, perform coordinate descent update

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

  - 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
  - …

© Eric Xing @ CMU, 2005-2020

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.**

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

Background: 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): $\mathsf{P}_{\|\cdot\|_1+\|\cdot\|_{\mathsf{tv}}}^\eta(\mathbf{w}) = \mathsf{P}_{\|\cdot\|_1}^\eta\left(\mathsf{P}_{\|\cdot\|_{\mathsf{tv}}}^\eta(\mathbf{w})\right)$

Improvement #1: 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

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

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

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 w^{t+1} to **workers** and repeat

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

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

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

- 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?

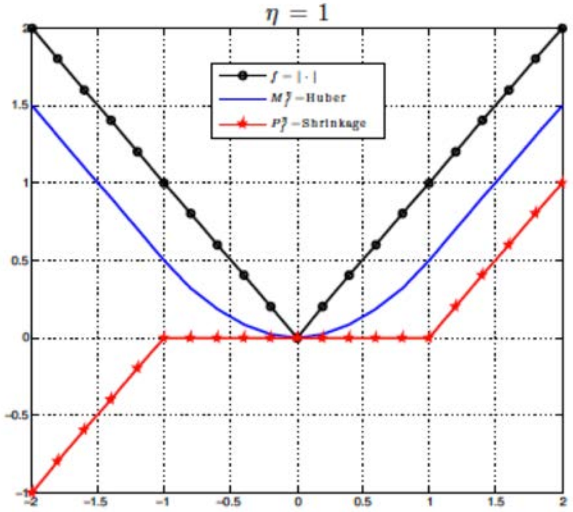
Improvement #2:
Non-Smooth Objectives: Moreau Envelope Smoothing

- 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

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

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

  - 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]

$$\min_{\mathbf{w}} f(\mathbf{w}) + g(\mathbf{w}) \quad \Leftarrow \text{original}$$

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

- 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

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

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

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

Parallel SPG?

- Difficulty: Gradients replaced by $\mathsf{P}^{\eta}_{f}(\mathbf{w}^{t})$

- Requires $\mathsf{P}^{\eta}_{f}(\mathbf{w}^{t})$ to be parallelizable
  - Assuming this can be done, then:
  1. Parallel-compute $\mathsf{P}^{\eta}_{f}(\mathbf{w}^{t})$ on **workers**
  2. Aggregate on **servers**
  3. Compute proximal operator on **servers**
  4. Compute momentum on **servers**
  5. Send result w^{t+1} to **workers** and repeat

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

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

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

- Above strategy is exact under Bulk Synchronous Parallel (just like accelerated PG)
  - Not clear how asynchronous updates impact smoothing+momentum
  - Not clear which $\mathsf{P}^{\eta}_{f}(\mathbf{w}^{t})$ can be parallelized
  - Open research topic

Optimization & MCMC Algorithms

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

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

Example Probabilistic Program: Topic Models

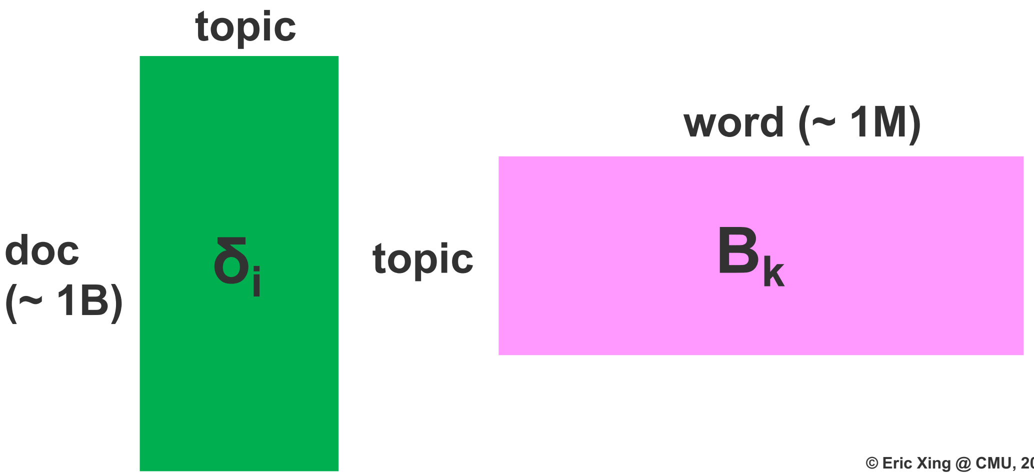
$$\sum_{i=1}^{N}\sum_{j=1}^{N_i} \ln \mathbb{P}_{Categorical}(x_{ij} \mid z_{ij}, B) + \sum_{i=1}^{N}\sum_{j=1}^{N_i} \ln \mathbb{P}_{Categorical}(z_{ij} \mid \delta_i)$$

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

Generative model of data

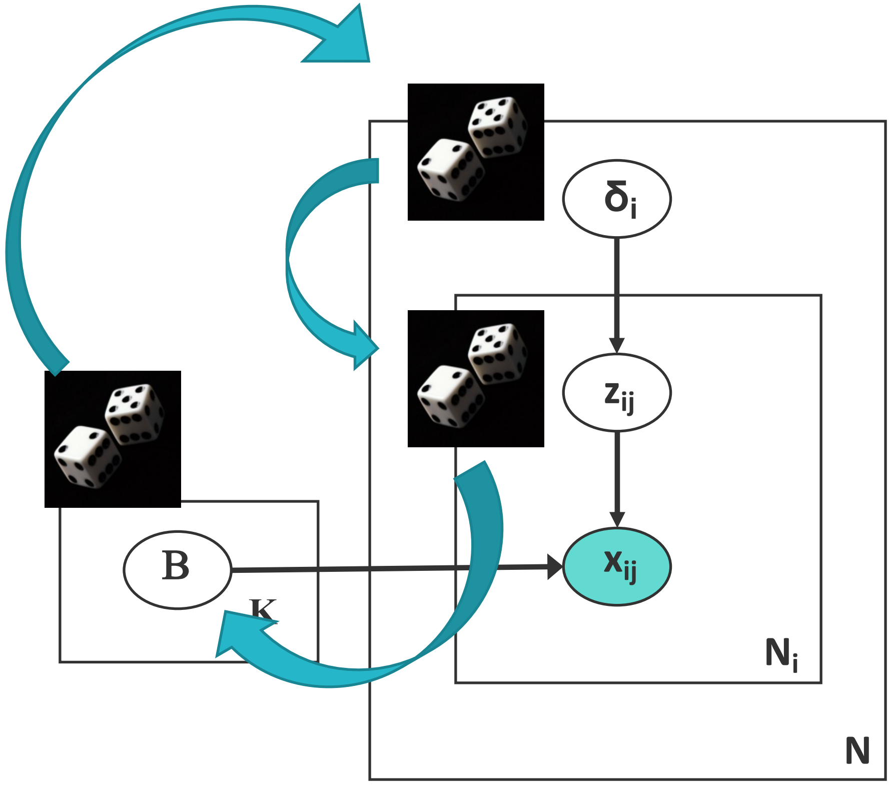
Priors on parameters

- 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

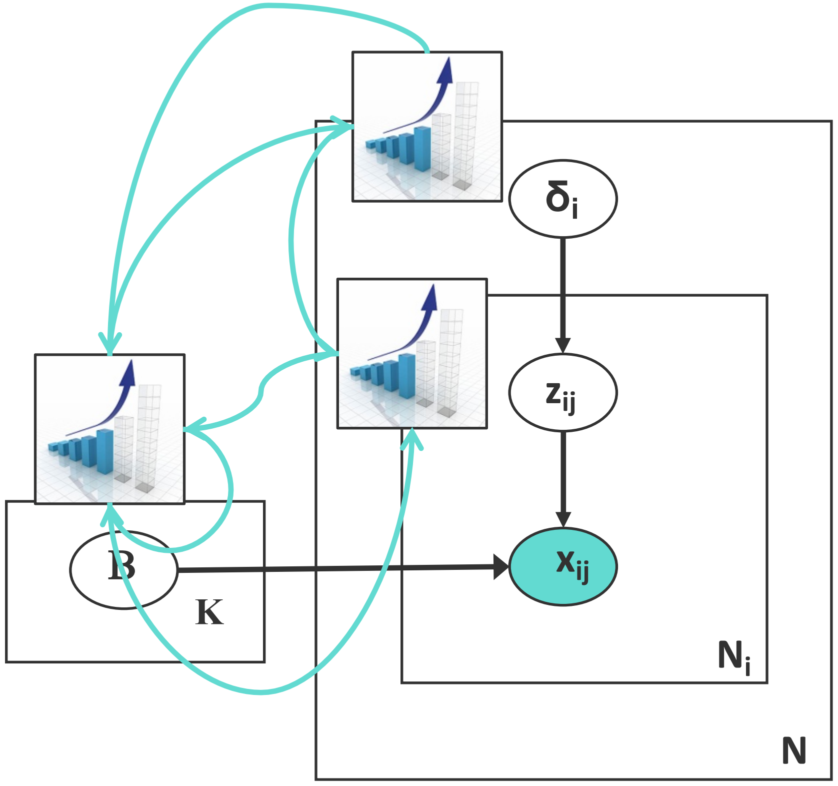


Inference for Probabilistic Programs: MCMC and SVI

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

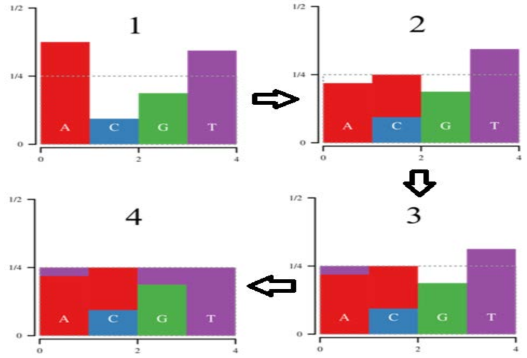


Variational Inference:
Gradient ascent on variables
Can be treated as an optimization problem



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



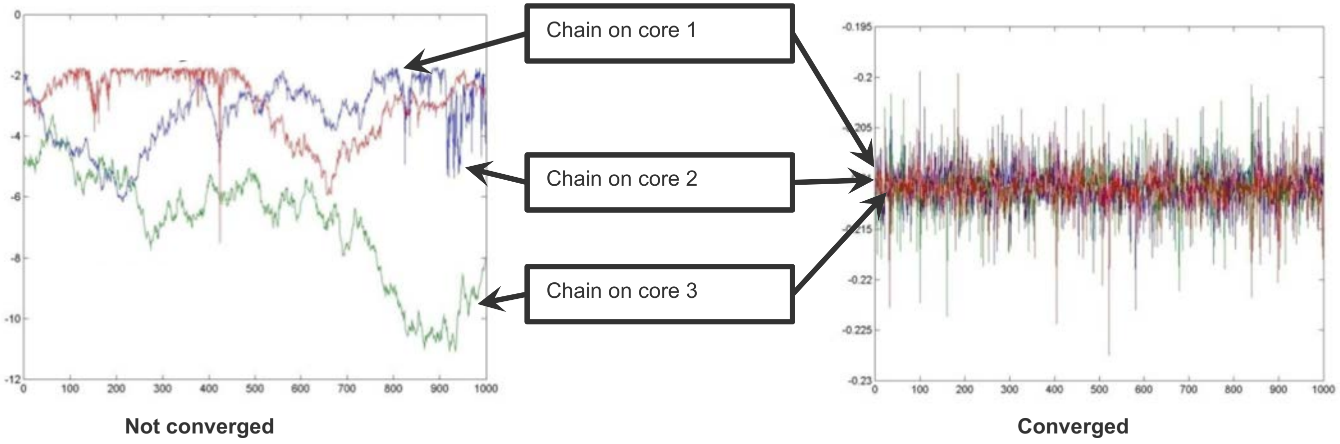
- Technique 2: Cyclic Metropolis Hastings [Yuan et al., 2015]
  - Exploit Bayesian form $P(z=k) = P_{evidence}(k) * P_{prior}(k)$
    - Propose z_1 from $P_{evidence}(k)$
    - Accept/Reject z_1
    - Propose z_2 from $P_{prior}(k)$
    - Accept/Reject z_2 … repeat
  - $P_{prior}(k)$, $P_{evi}(k)$ cheap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

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

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

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!



Parallel and Distributed MCMC: Classic methods

- Classic parallel MCMC solution 2
  - Sequential Importance Sampling (SIS)
  - 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)

$$D_j \sim \mathrm{DP}\left(\frac{\alpha}{P}, H\right), \quad j = 1, \ldots, P$$

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

$$\pi_i \sim \phi$$

$$\theta_i \sim D_{\pi_i}$$

$$x_i \sim f(\theta_i), \quad i = 1, \ldots, N.$$

$$D \sim \mathrm{DP}(\alpha, H),$$

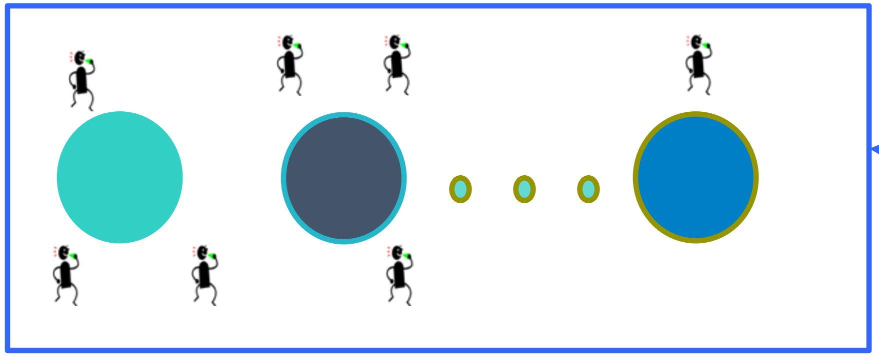
$$\theta_i \sim D,$$

$$x_i \sim f(\theta_i)$$

Solution I: Induced Independence via Auxiliary Variables

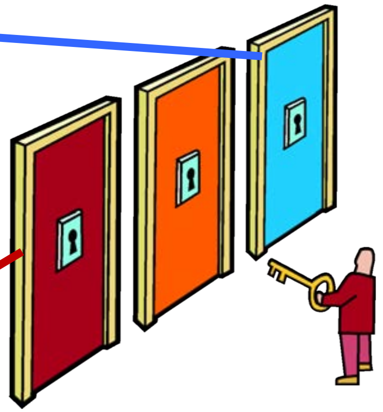
[Dubey et al. 2013, 2014]

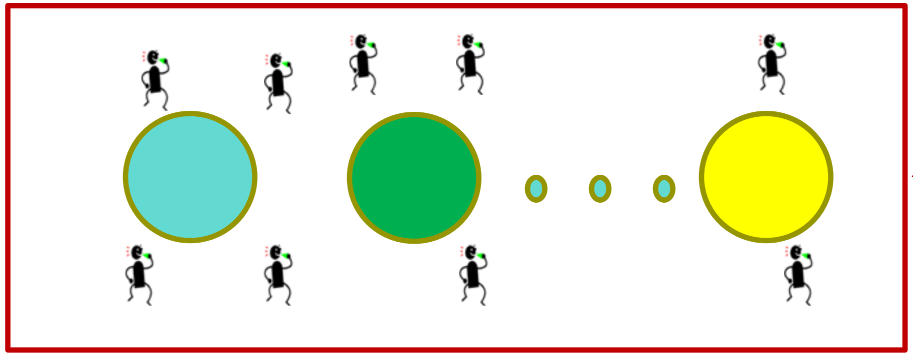
- Why does it work? A mixture over Dirichlet processes is equivalent to a Dirichlet processes



DP on Processor 1

Dirichlet Mixture over Processor DPs 1...P





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".

Embarrassingly 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:

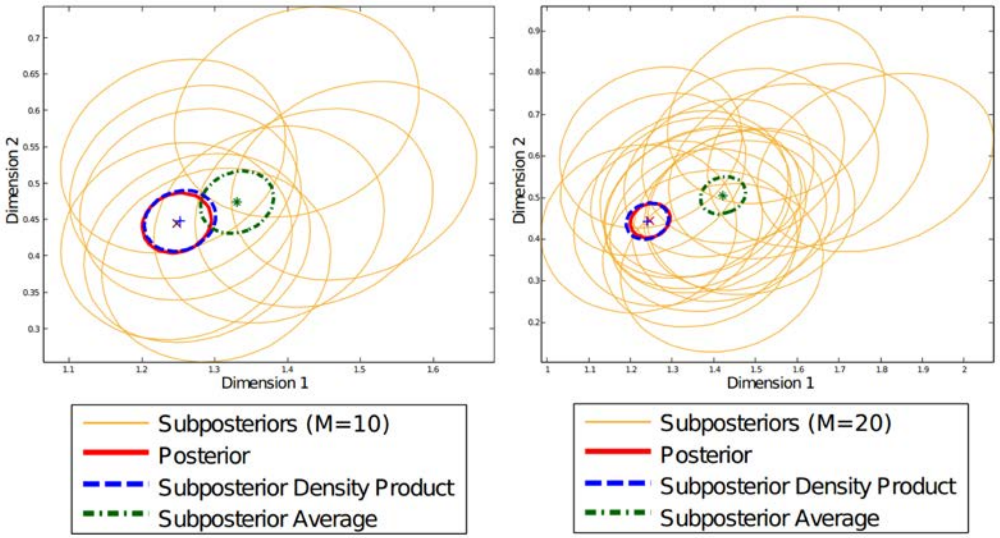
$$\widehat{p_1 \cdots p_M}(\theta) = \widehat{p}_1 \cdots \widehat{p}_M(\theta) = \frac{1}{T^M}\prod_{m=1}^{M}\sum_{t_m=1}^{T}\mathcal{N}_d(\theta|\theta_{t_m}^m, h^2 I_d) \quad \propto \sum_{t_1=1}^{T}\cdots\sum_{t_M=1}^{T} w_{t\cdot}\mathcal{N}_d\left(\theta\middle|\bar{\theta}_{t\cdot}, \frac{h^2}{M}I_d\right)$$

     - where

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

Embarrassingly Parallel MCMC

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



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:

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

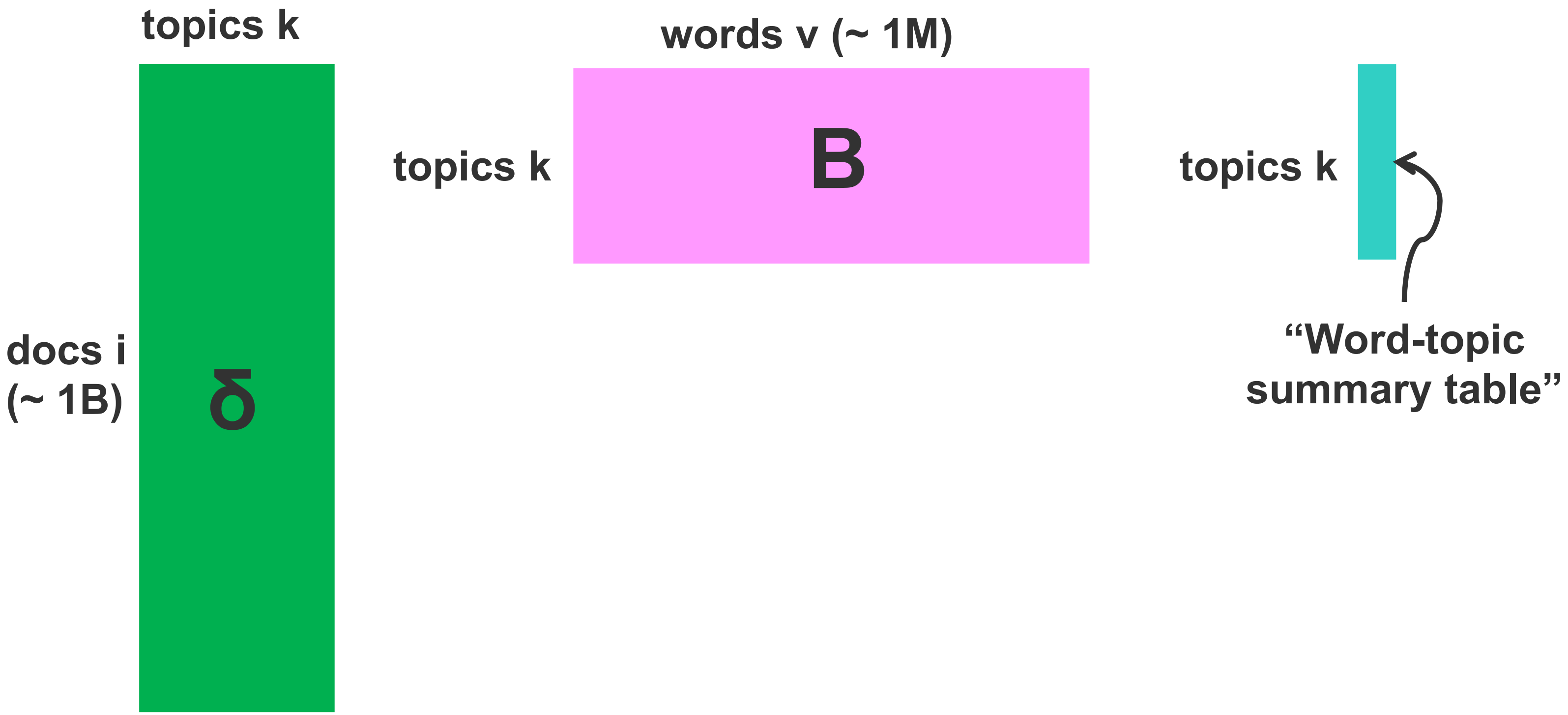
Properties of Collapsed Gibbs Sampling (CGS)

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

- 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!

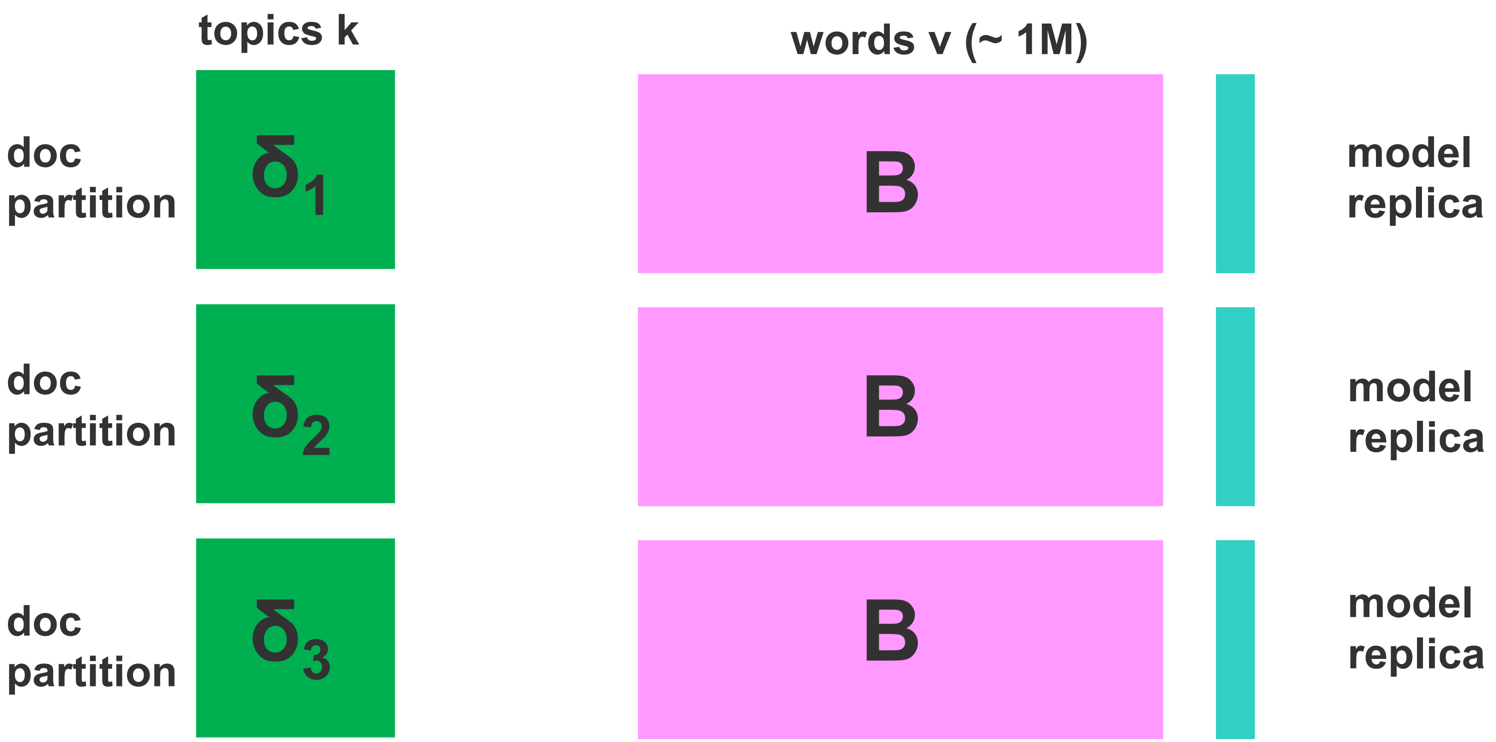
CGS Example: Topic Model sampler

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



Data Parallelization for CGS Topic Model Sampler

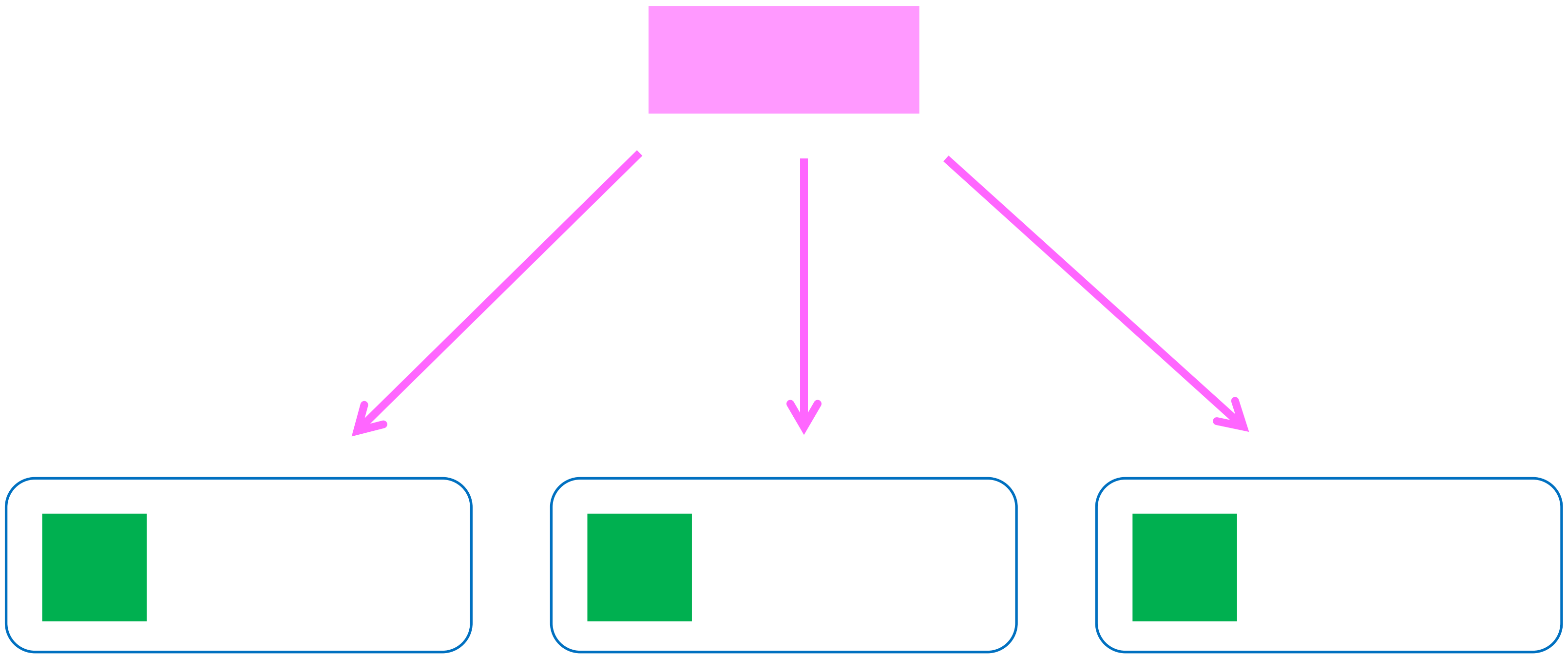
$$p(z_{ij} = k|x_{ij}, \delta_i, B) \propto (\boxed{\delta}_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + \boxed{B}_{k,x_{ij}}}{V\beta + \boxed{\sum_{v=1}^{V} B_{k,v}}}$$



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

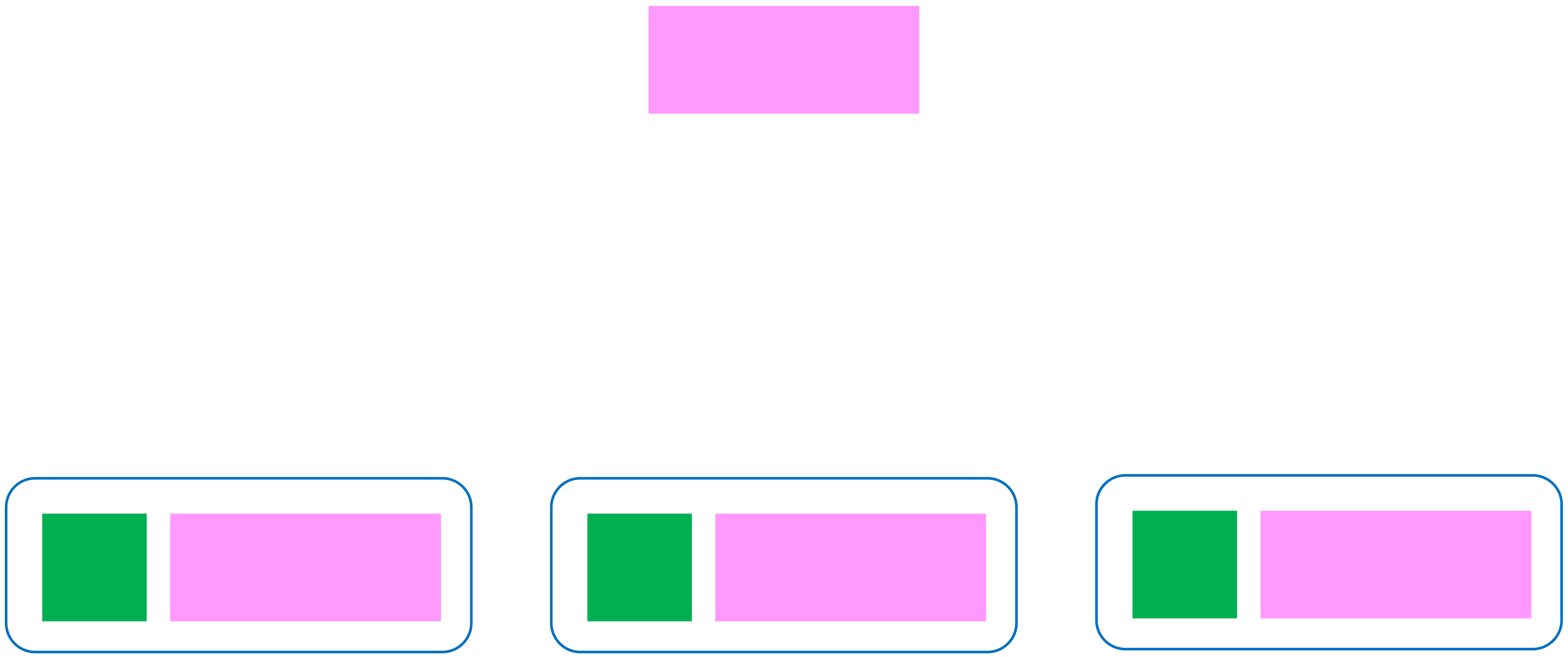
- Step 1: broadcast central model



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

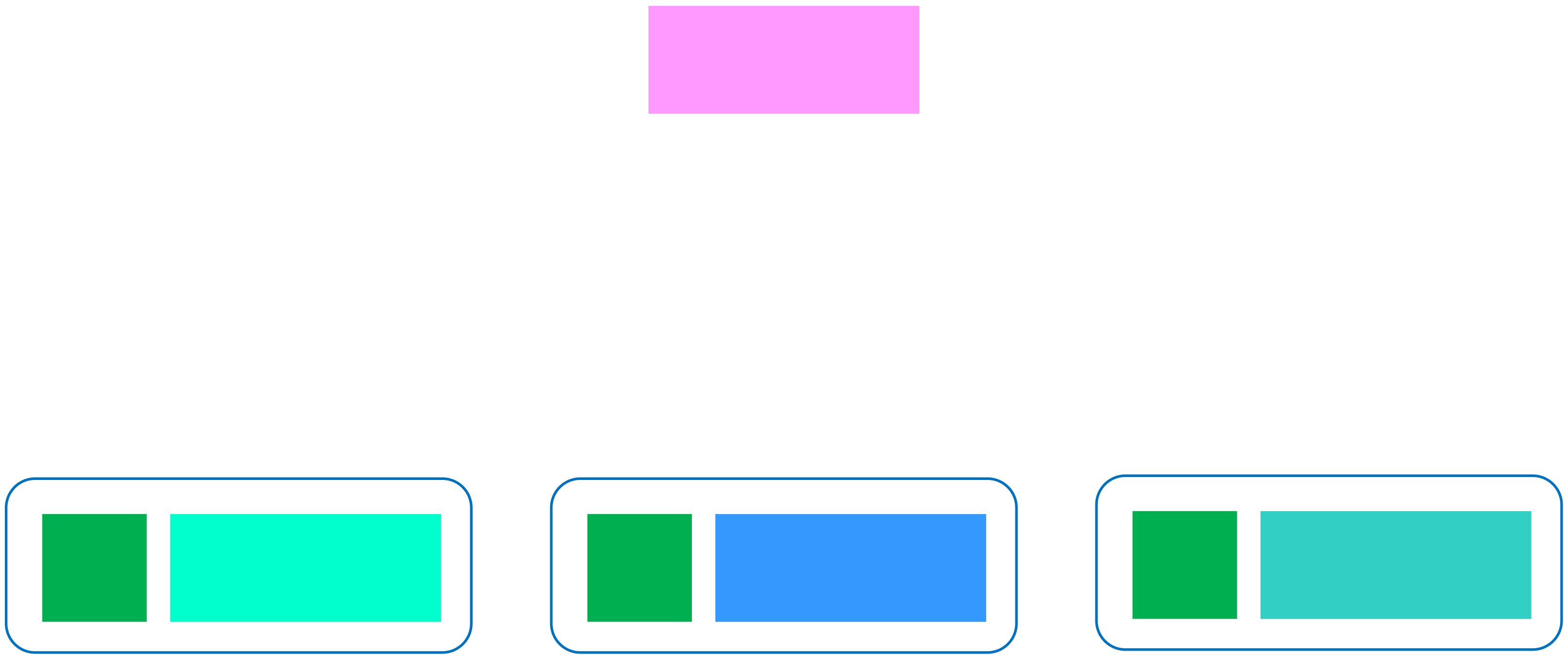
- Step 1: broadcast central model



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

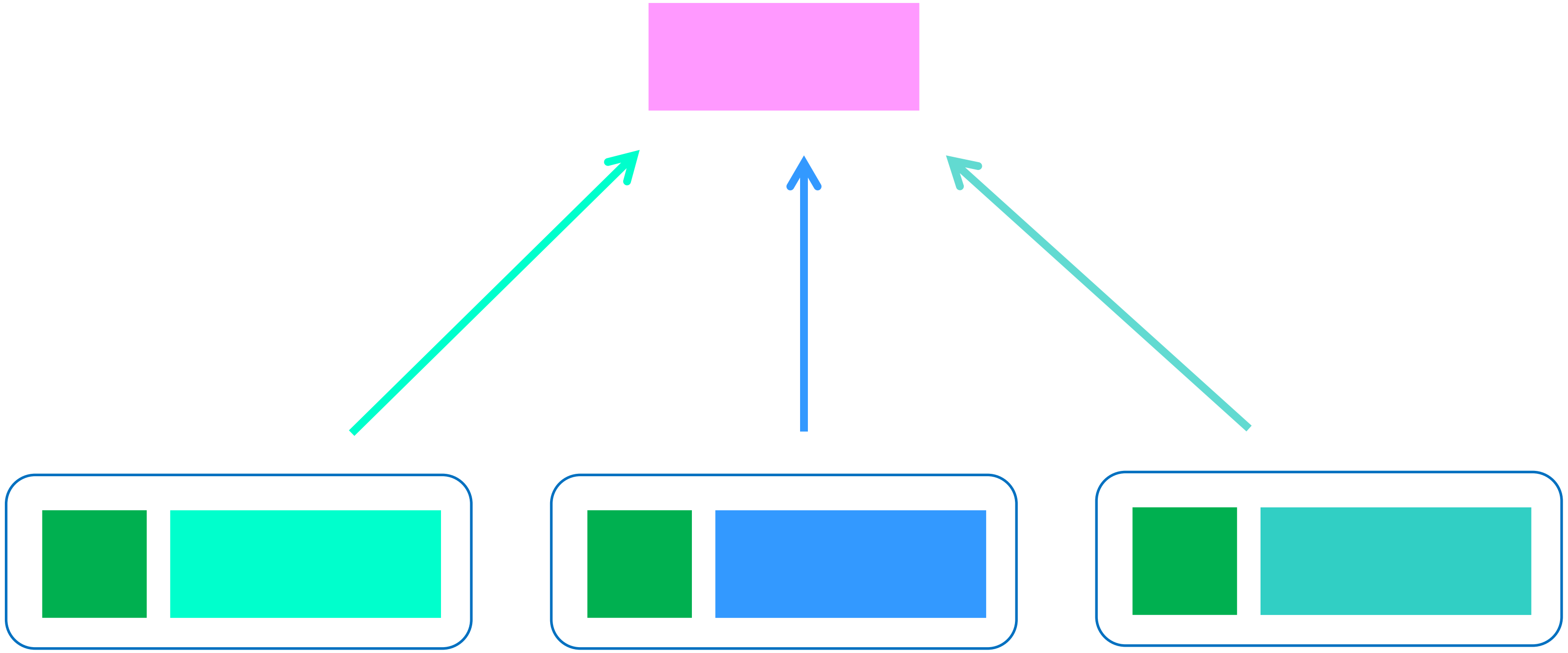
- Step 2: Perform Gibbs sampling in parallel



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:

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

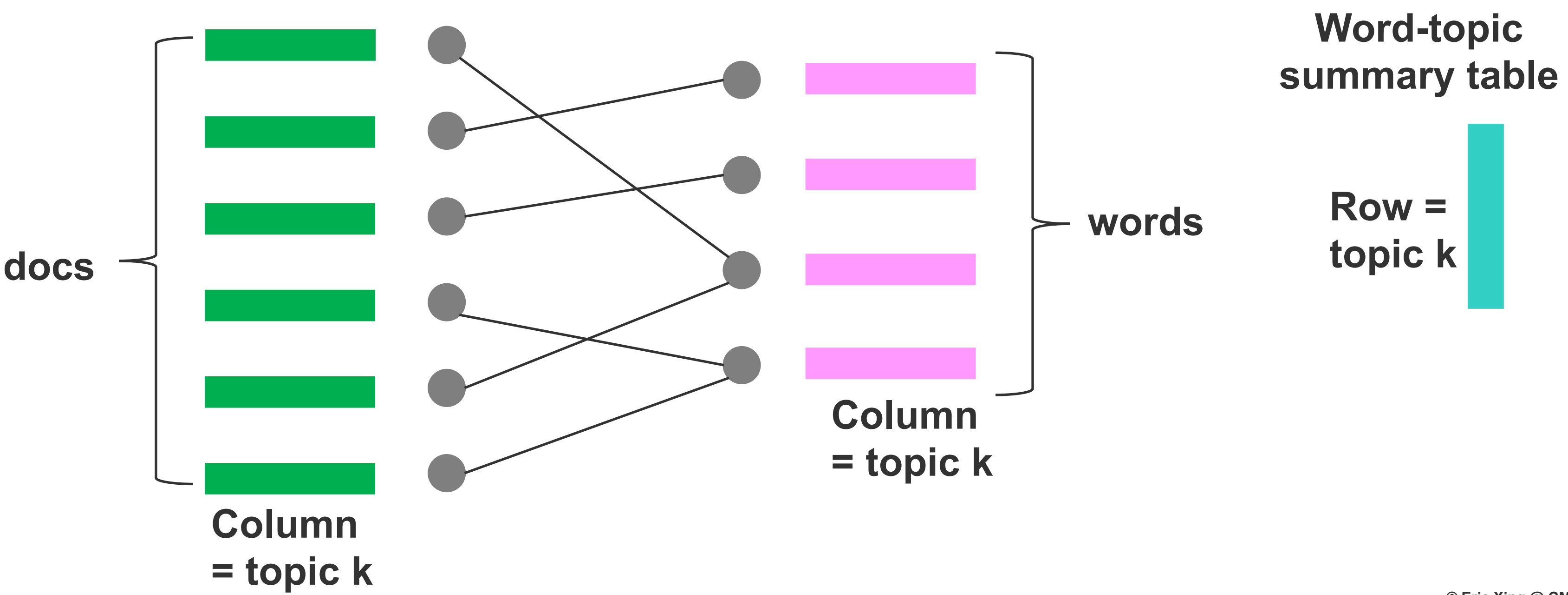
- Data-parallelism incurs error in B (the pink box) and the summation term (the gray box)
  - Both quantities are duplicated onto workers; their values become stale as sampling proceeds
  - True even for bulk synchronous parallel execution!
- Asynchrony helps somewhat
  - Communicate very frequently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

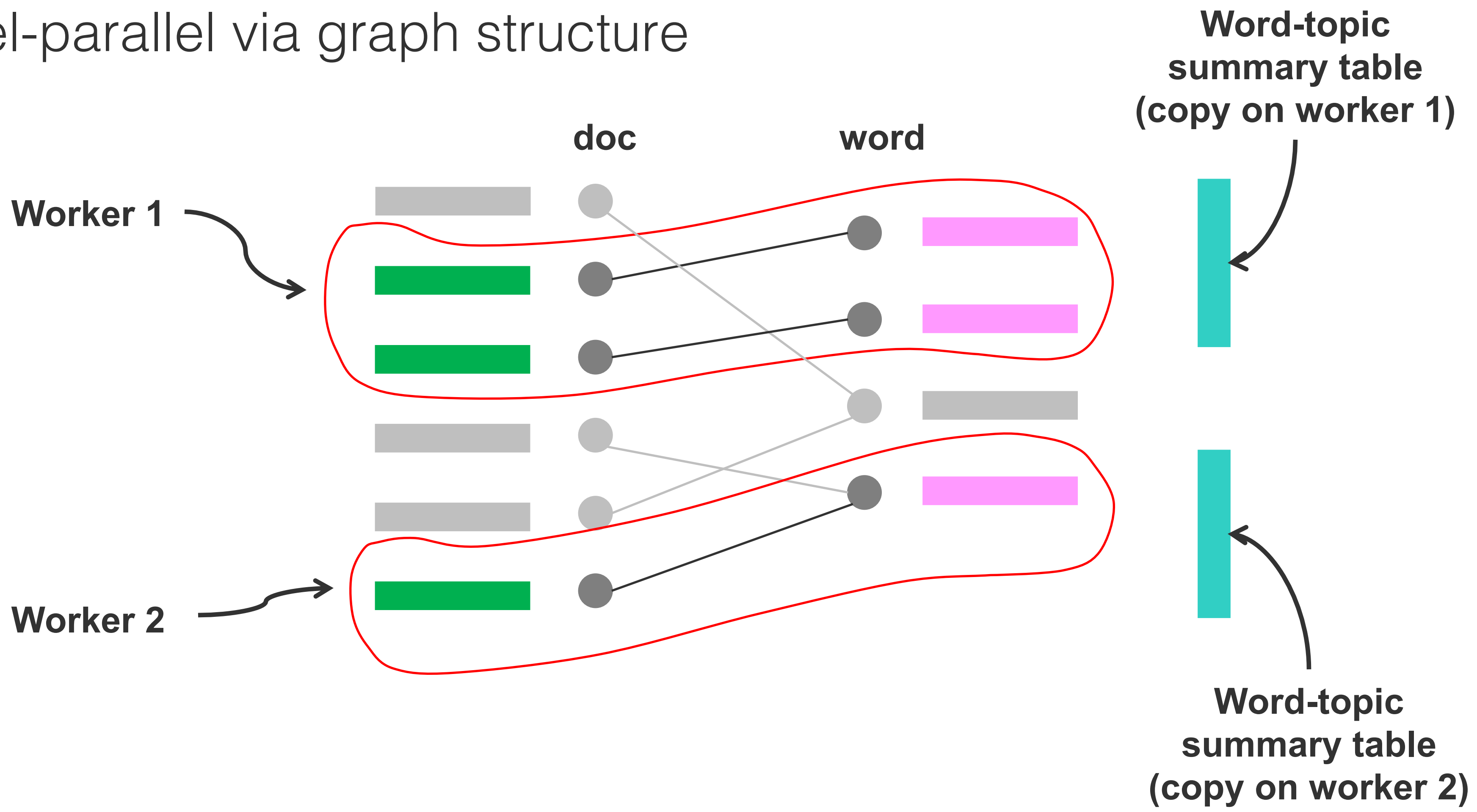
$$p(z_{ij} = k | x_{ij}, \delta_i, B) \propto (\boxed{\delta}_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + \boxed{B}_{k,x_{ij}}}{V\beta + \boxed{\sum_{v=1}^{V} B_{k,v}}}$$



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

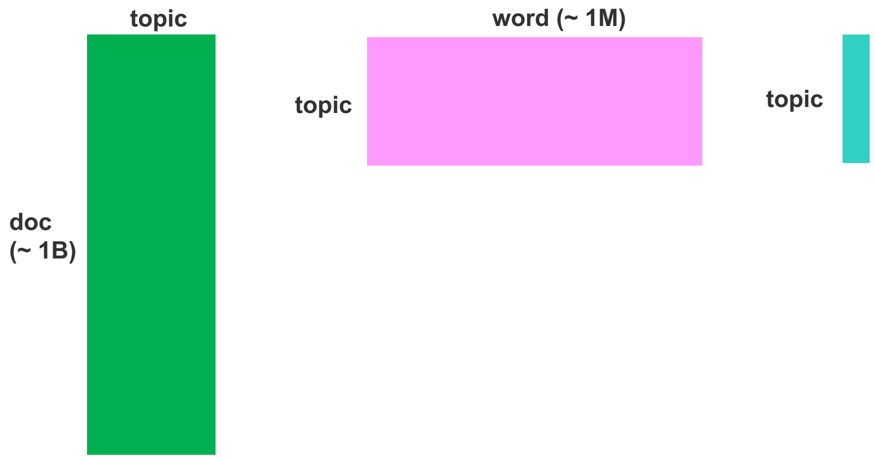
[Low et al., 2010; Gonzalez et al., 2012]

- 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?

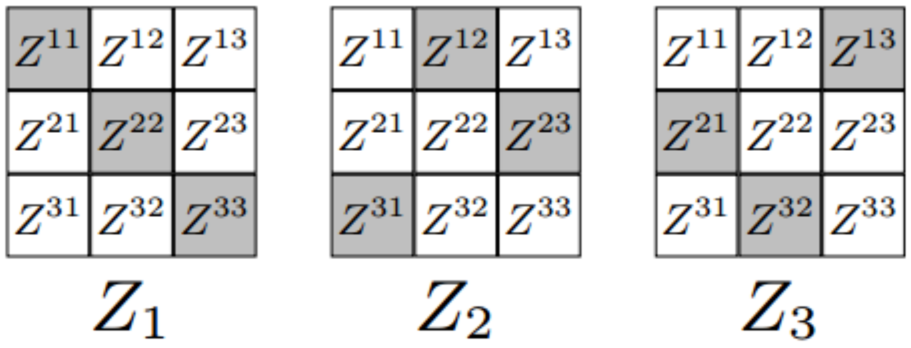
Model-Parallel Strategy 2: LightLDA (Petuum LDA v2)

[Yuan et al., 2015]

- Topic model matrix structure:



- Idea: non-overlapping matrix partition:

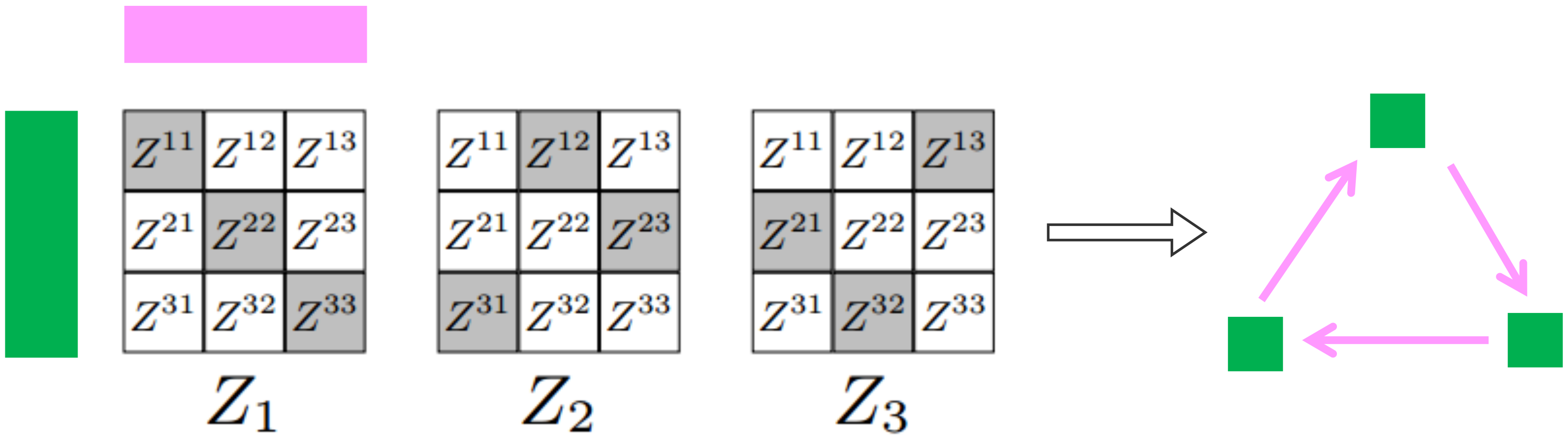


Source: [Gemulla et al., 2011]

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

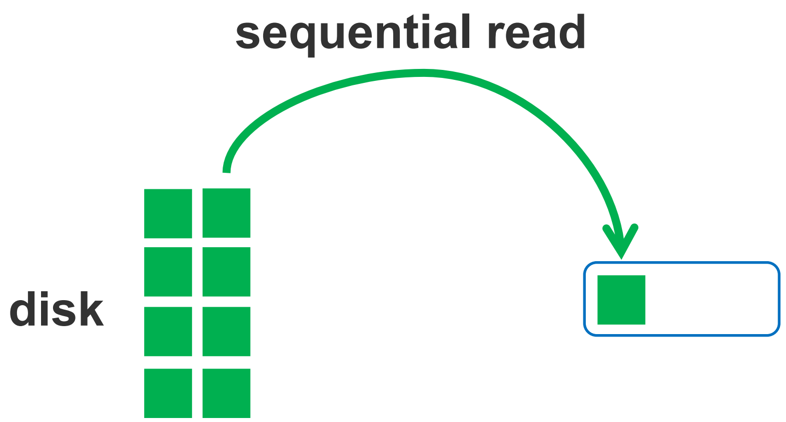


Source: [Gemulla et al., 2011]

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

[Yuan et al., 2015]

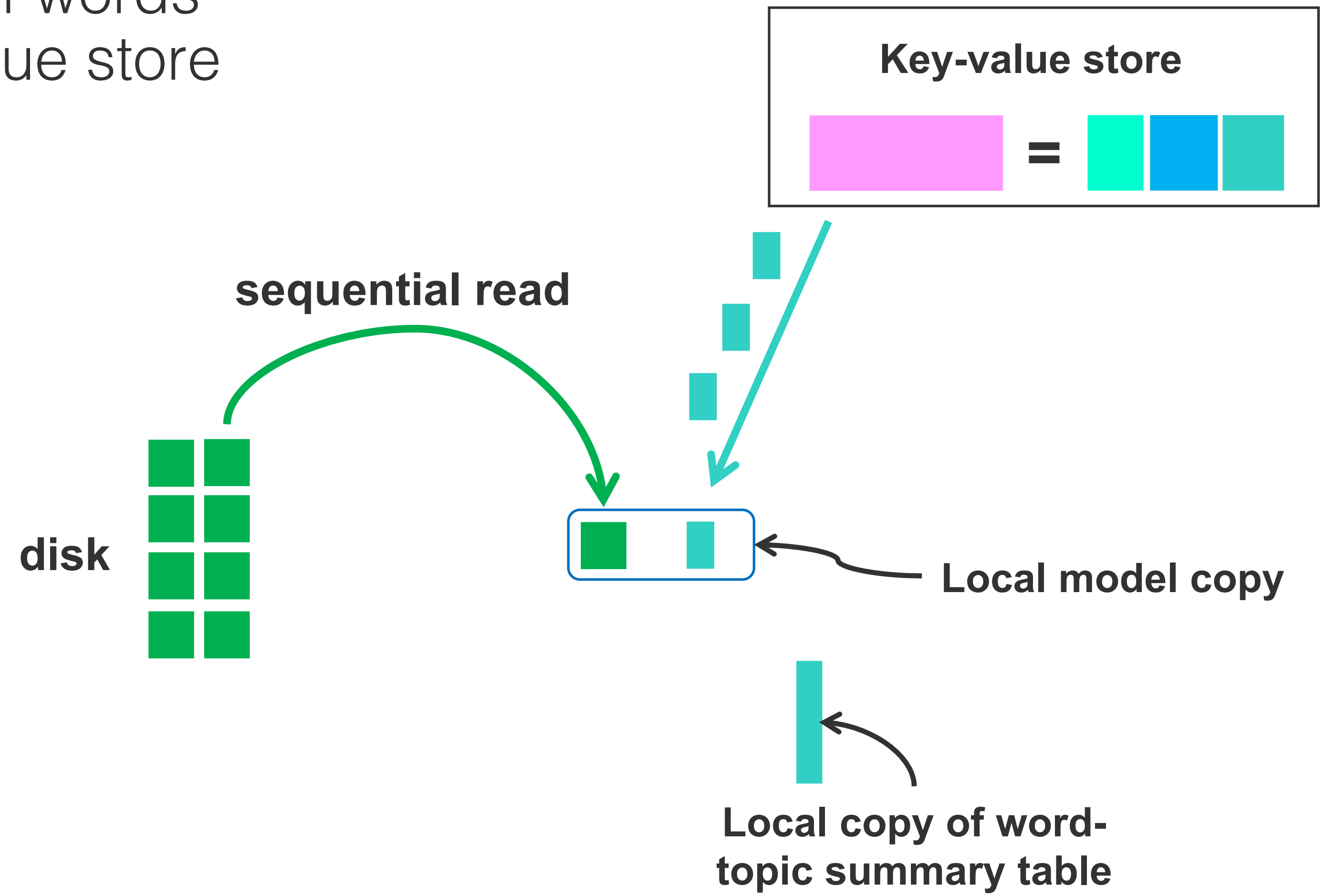
- During preprocessing: determine set of words used in each data block
- Begin training: load each data block from disk



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

[Yuan et al., 2015]

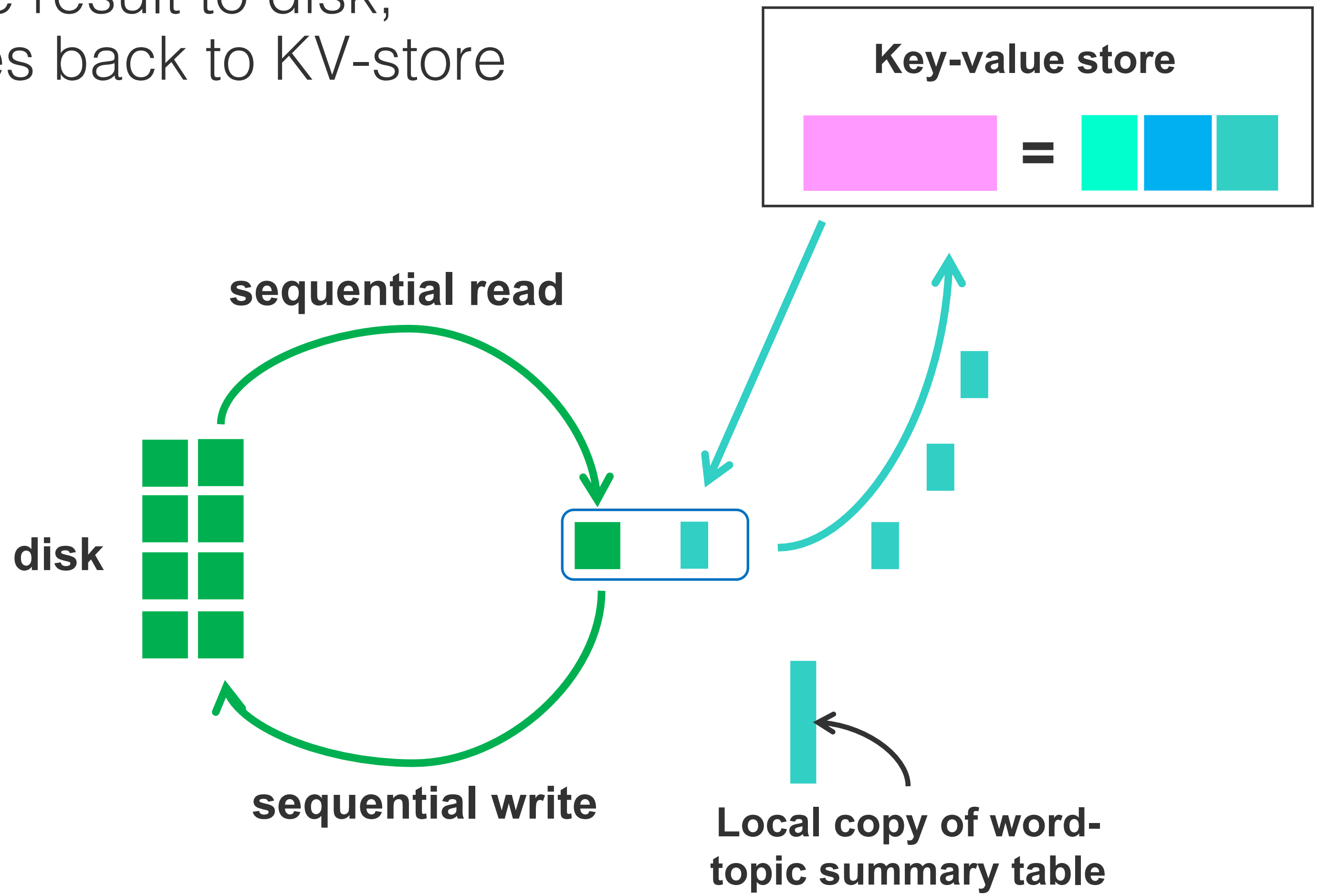
- Pull the set of words from Key-Value store



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

- Parallel algos for Optimization and MCMC share common 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.

Part 2: Distributed Systems for ML

Distributed Systems for ML

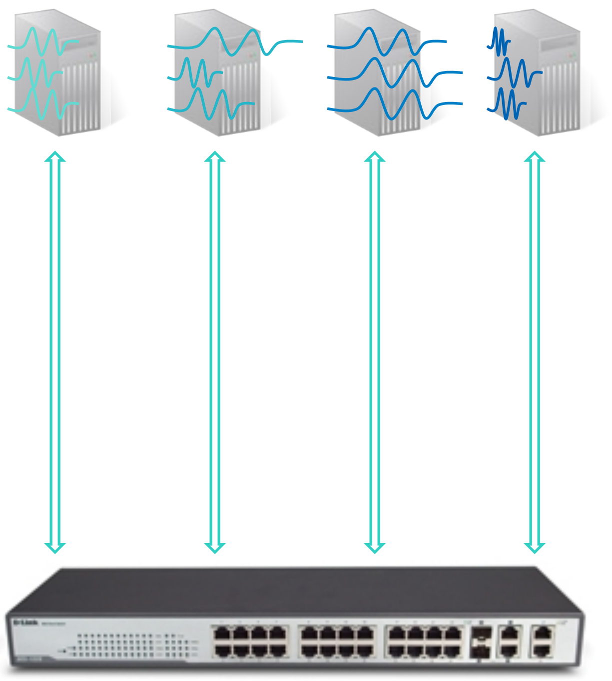
- Just now: Exploit algorithmic and mathematical properties of ML learning and inference algorithms, to create efficient distributed ML algorithms
  - Once model has been learnt, prediction is (usually) embarrassingly parallel – given n machines, duplicate the learnt model and give each machine $1/n$ of the samples to be predicted
- What about the systems properties of real-world machines?

There Is No Ideal Distributed System!

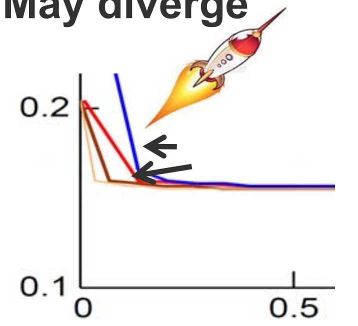
- Two distributed challenges:
  - Networks are (relatively) slow
  - “Identical” machines rarely perform equally

Unequal performance

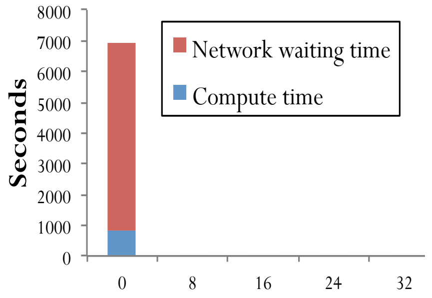
Low bandwidth, High delay



Async execution: May diverge



BSP execution: Long sync time



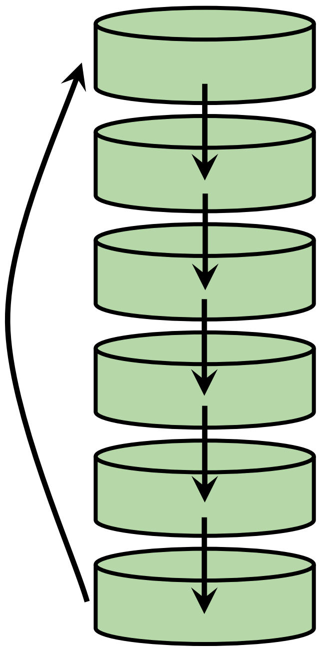
There Is No Ideal Distributed System!

- 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

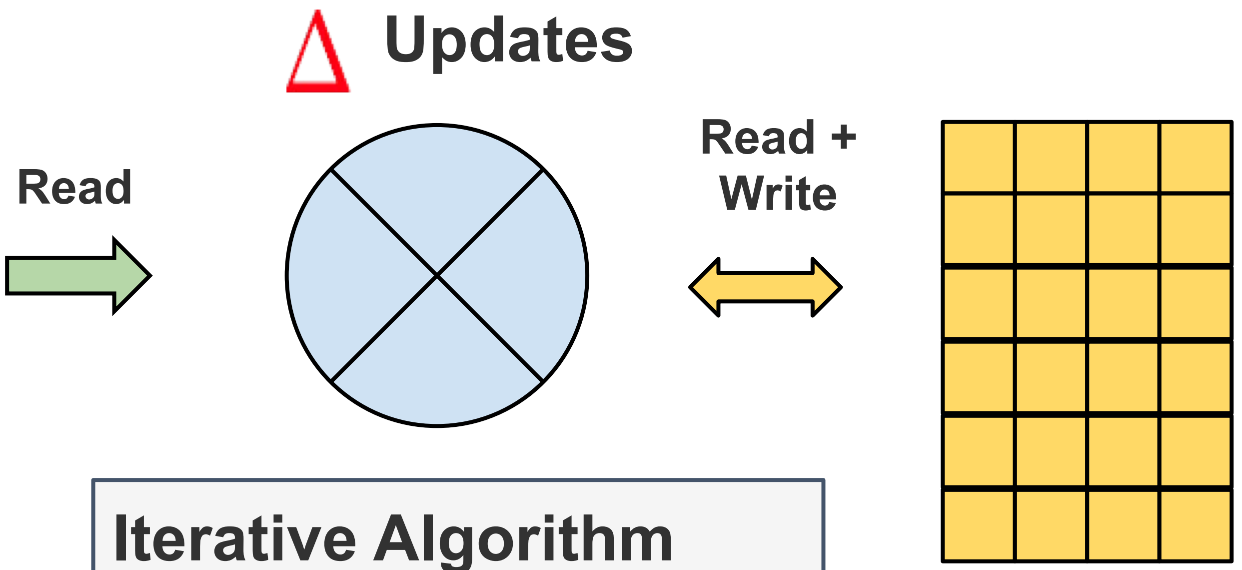
© Eric Xing @ CMU, 2005-2020

A General Picture of ML Iterative-Convergent Algorithms



D

Data



$A^{(t-1)}$

Model Parameters at iteration (t-1)

Iterative Algorithm

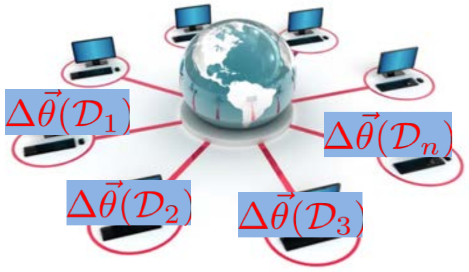
$\Delta \;\; = \Delta(A^{(t-1)}, D)$

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

$F()$ Aggregate + Transform

Δ Intermediate Updates

Issues with Hadoop and I-C ML Algorithms?



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

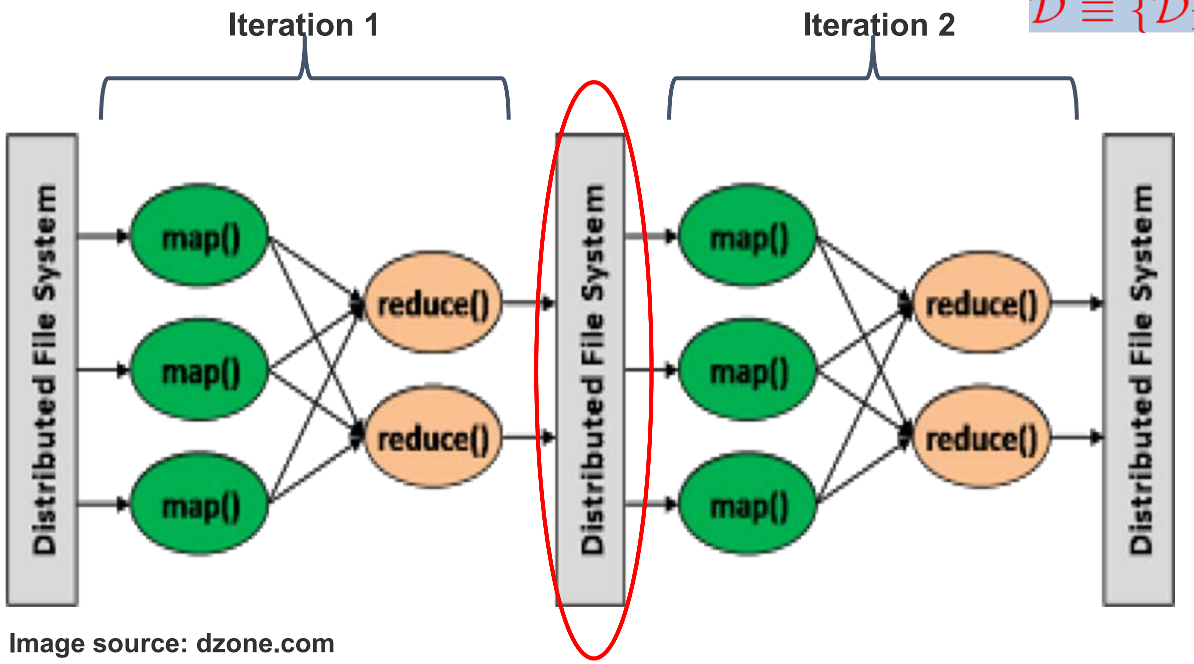


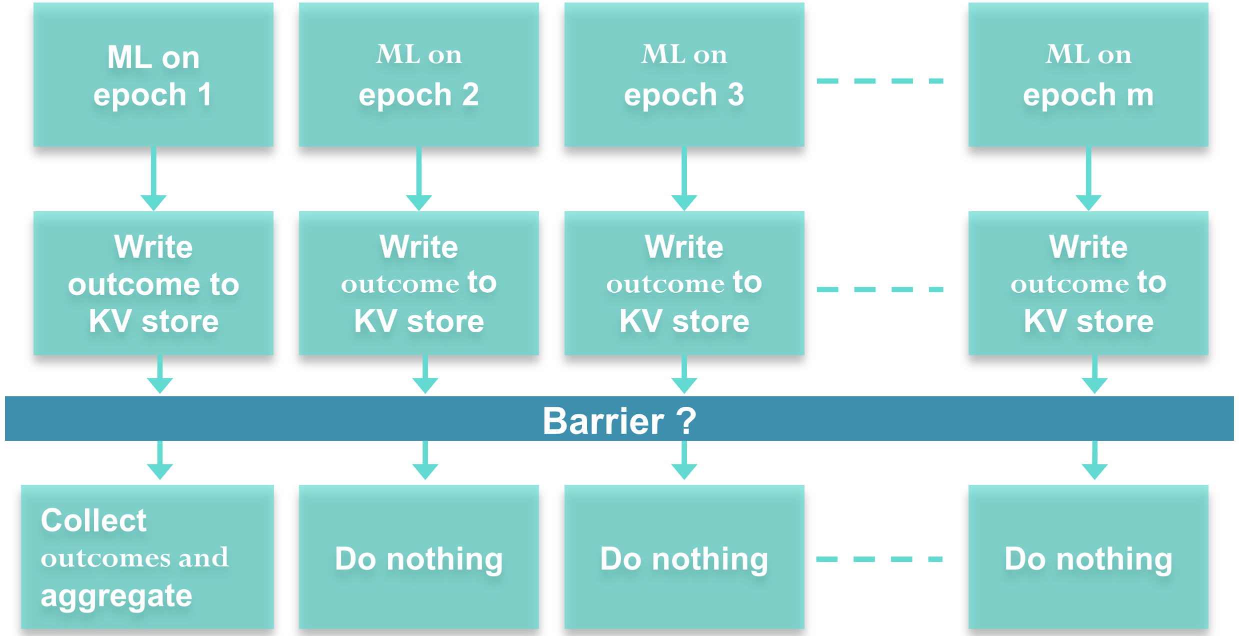
Image source: dzone.com

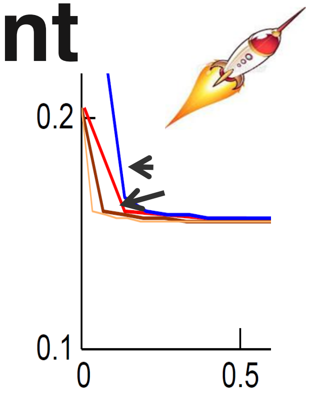
HDFS Bottleneck

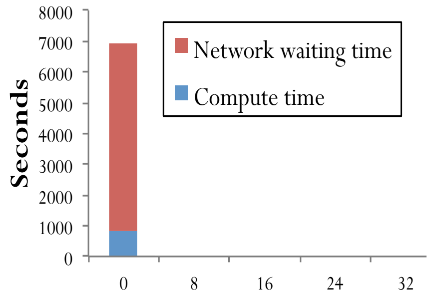
Naïve MapReduce not best for ML

- 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!

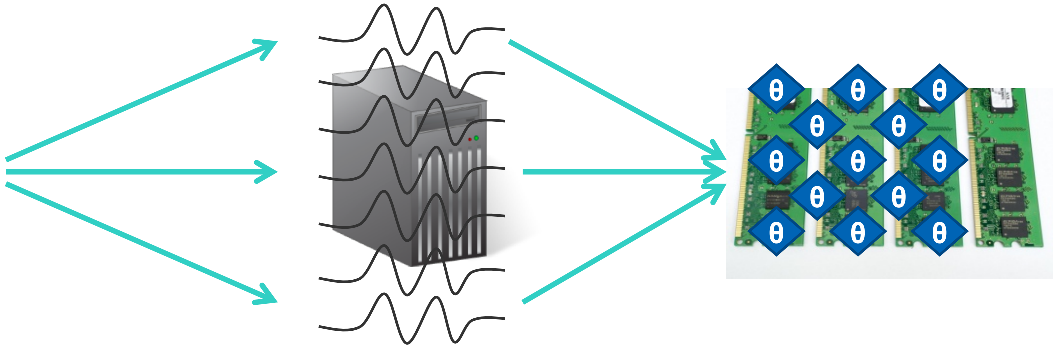
Good Parallelization Strategy is Important



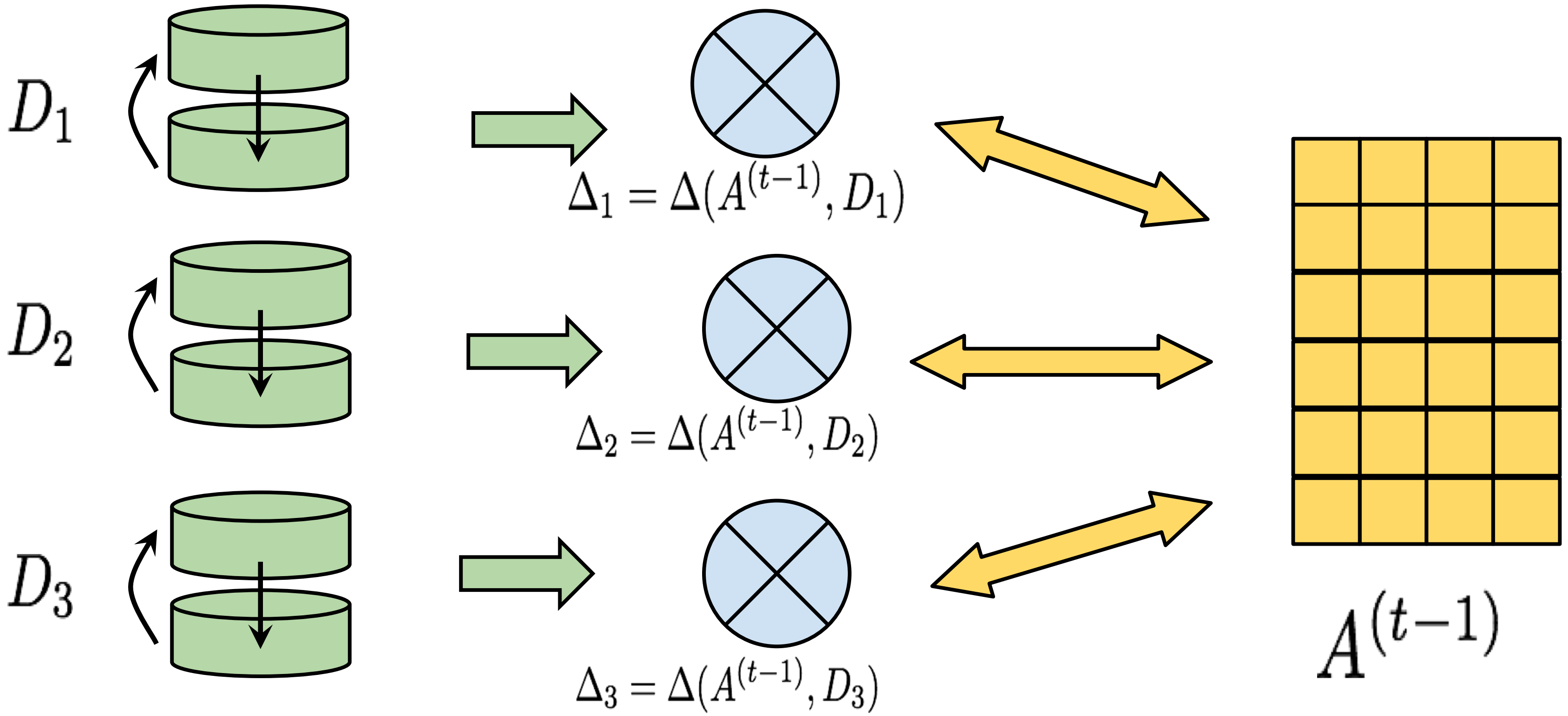





```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```



Data Parallelism



Additive Updates

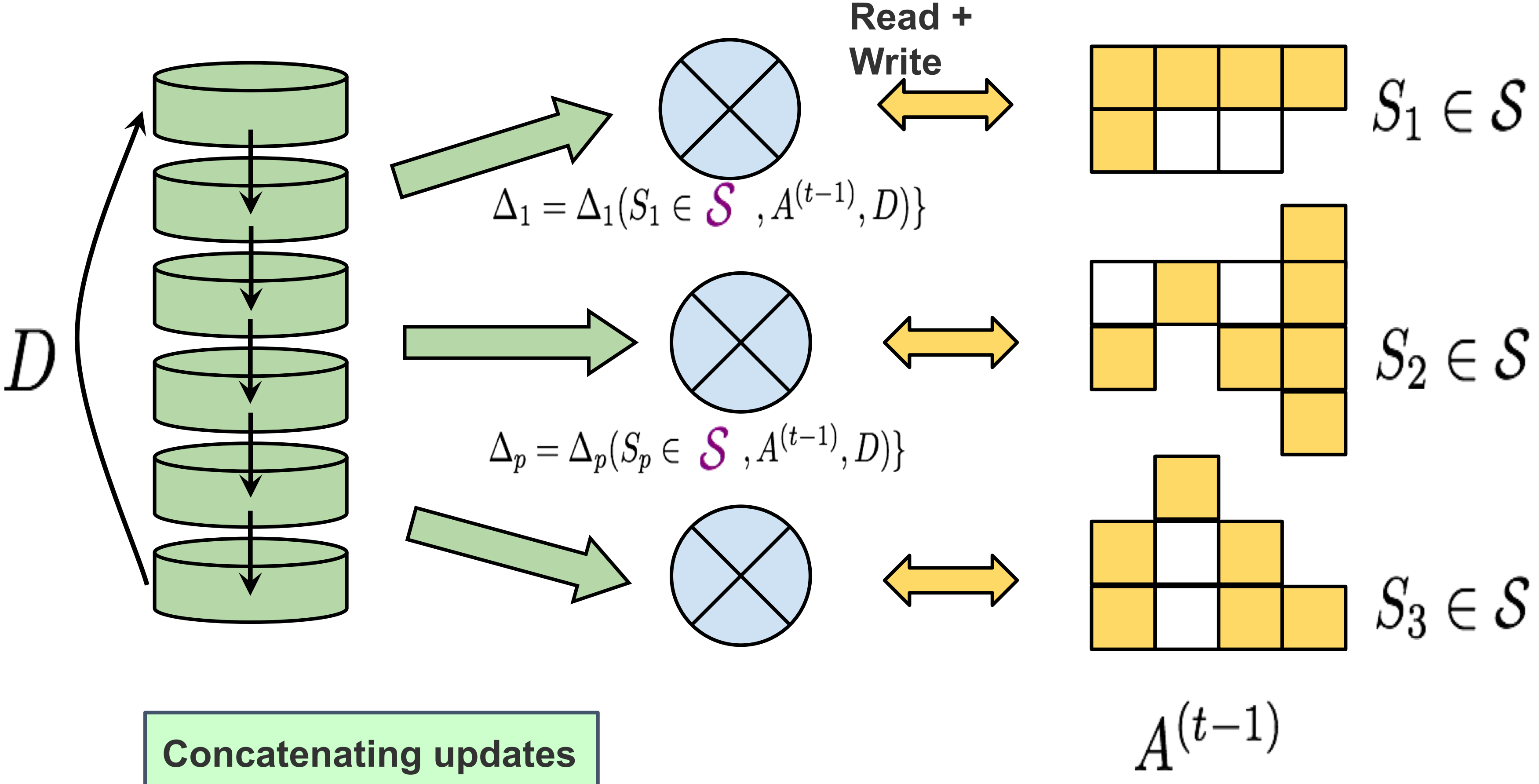
$$\Delta = \sum_{p=1}^{3} \Delta_p$$

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

Model Parallelism

Scheduling Function

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



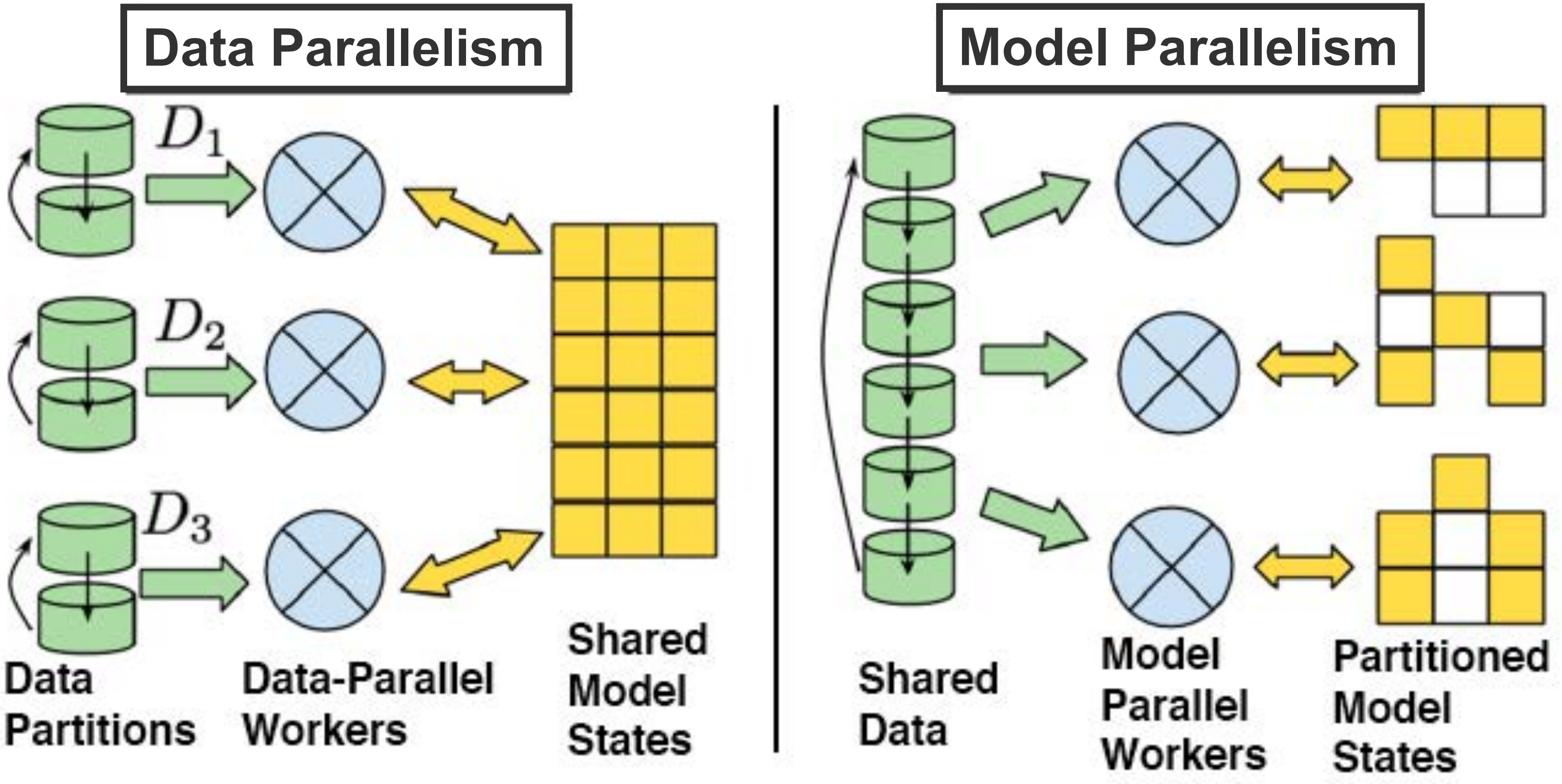
Concatenating updates

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

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

☐ model parameters not updated in this iteration

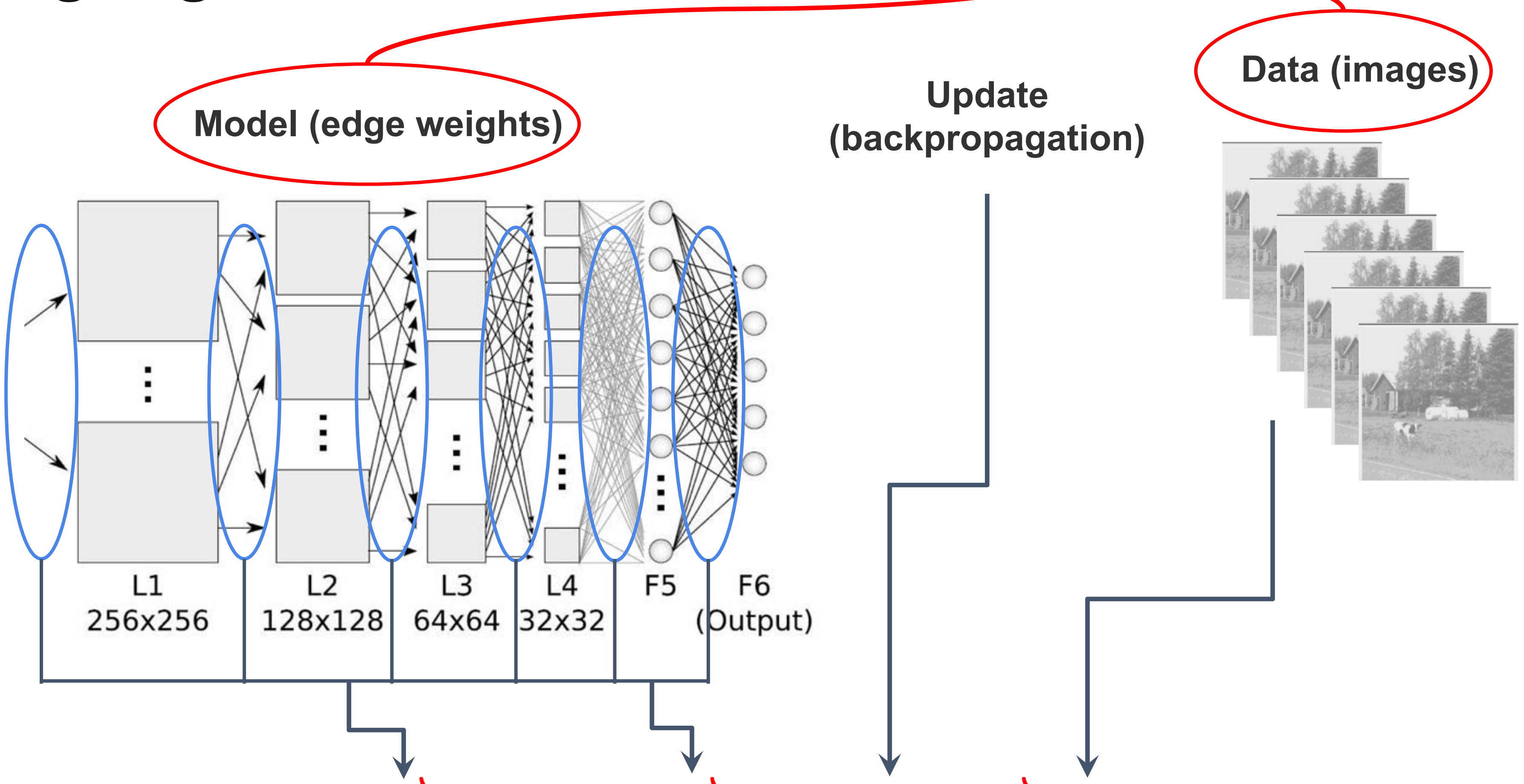
A Dichotomy of Data and Model in ML Programs



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

Data+Model Parallel: Solving Big Data+Model

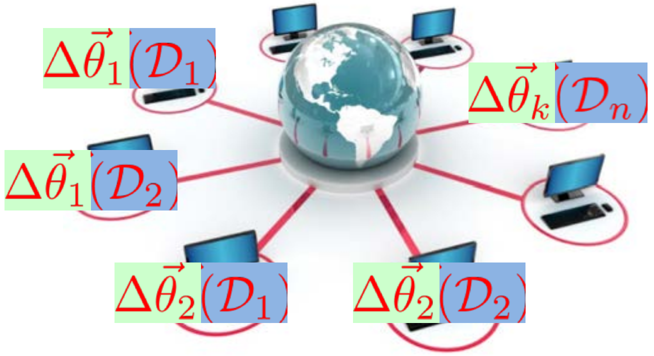
Data & Model both big!
Millions of images,
Billions of weights
What to do?



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

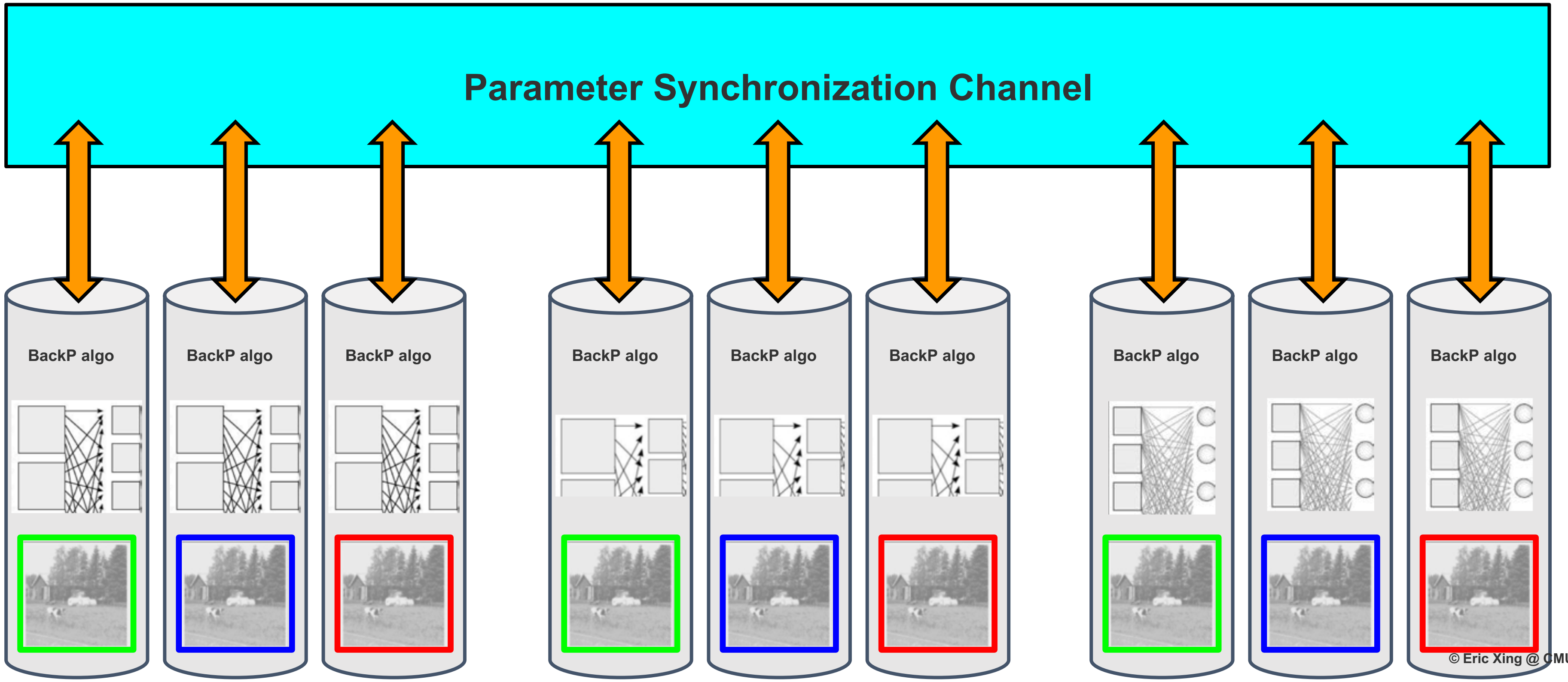
Data+Model Parallel: Solving Big Data+Model

Tackle Deep Learning scalability challenges by combining data+model parallelism



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

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



How difficult is data/model-parallelism?

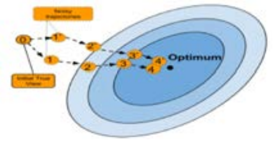
- Certain mathematical conditions must be met

- Data-parallelism generally OK when data IID (independent, identically distributed)
  - Very close to serial execution, in most cases

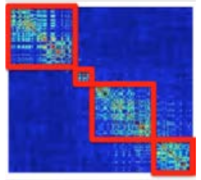
- Naive Model-parallelism won't work
  - NOT equivalent to serial execution of ML algo
  - Need carefully designed schedule

Intrinsic Properties of ML Programs

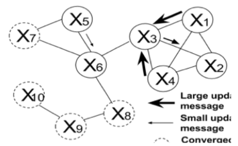
- ML is optimization-centric, and admits an iterative convergent algorithmic solution rather than a one-step closed form solution
  - Error tolerance: often robust against limited errors in intermediate calculations



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



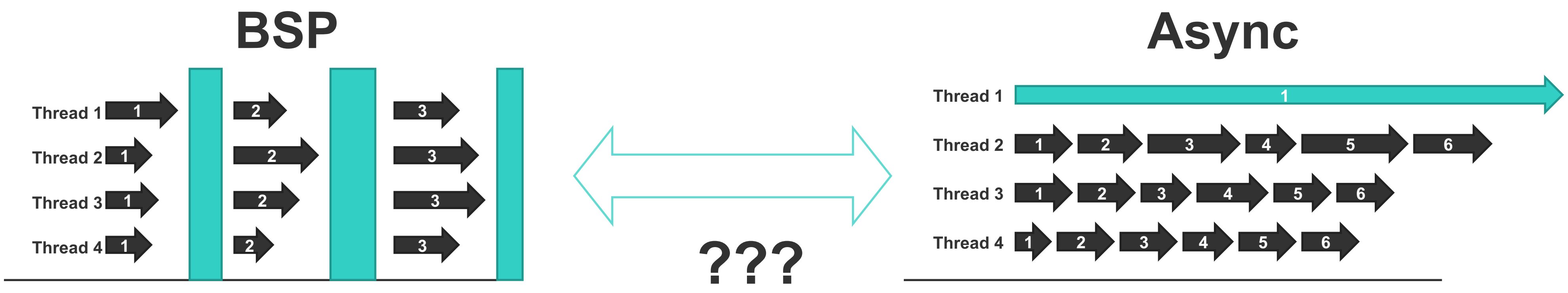
  - Non-uniform convergence: parameters can converge in very different number of steps



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

Challenges in Data Parallelism

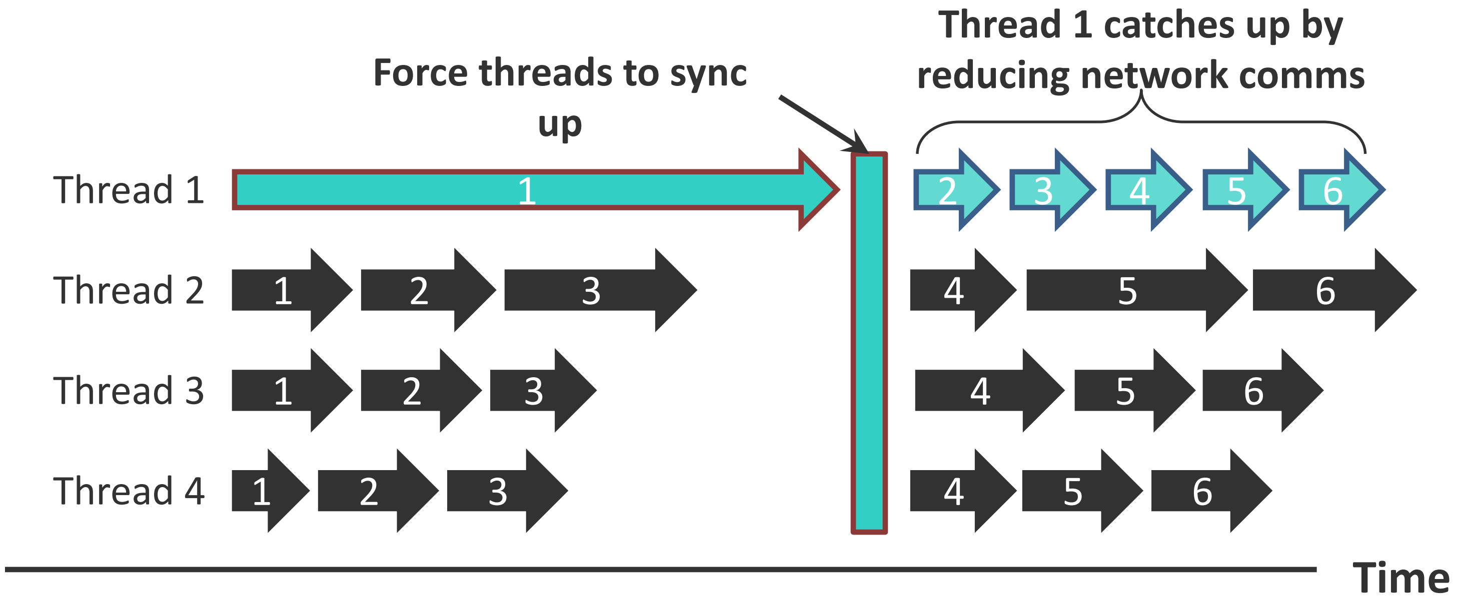
- Existing ways are either safe/slow (BSP), or fast/risky (Async)
- Challenge 1: Need “Partial” synchronicity
  - Spread network comms evenly (don’t sync unless needed)
  - Threads usually shouldn’t wait – but mustn’t drift too far apart!
- Challenge 2: Need straggler tolerance
  - Slow threads must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



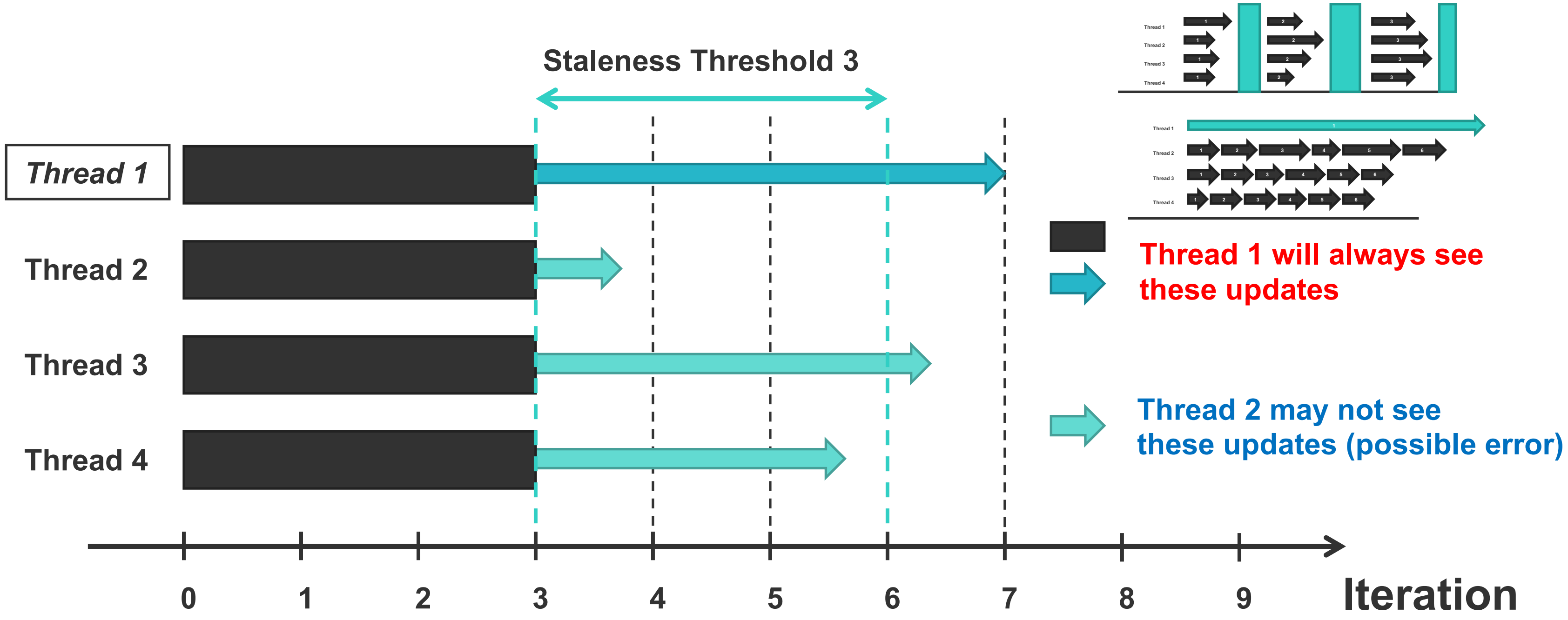
Is persistent memory really necessary for ML?

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]



Stale Synchronous Parallel (SSP), a "bounded-asynchronous" 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:

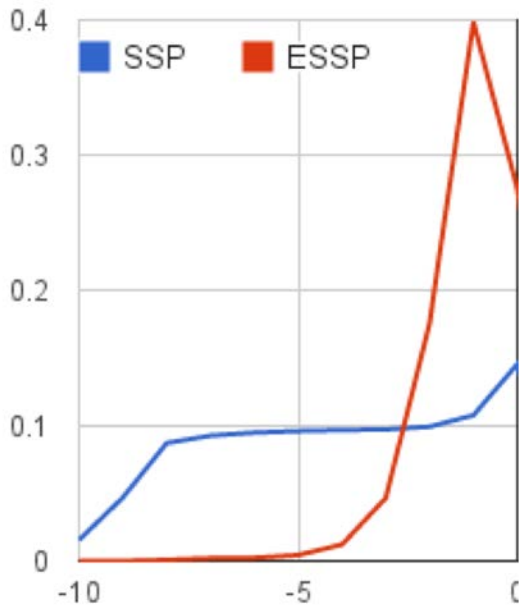
- Asynchronous-like speed, BSP-like ML correctness guarantees
- Guaranteed age bound (staleness) on reads
- Contrast: no-age-guarantee Eventual Consistency seen in Cassandra, Memcached

© Eric Xing @ CMU, 2005-2020

Improving Bounded-Async via Eager Updates

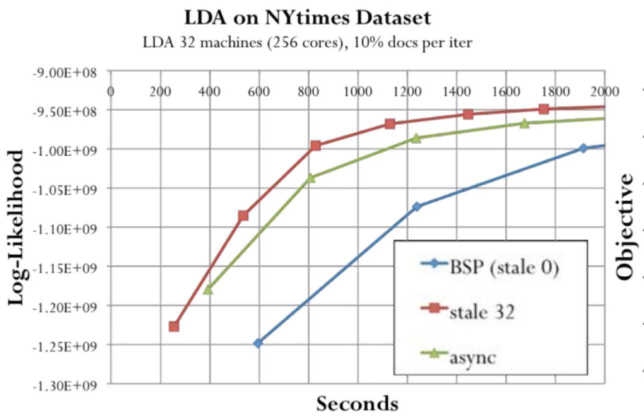
[Dai et al., 2015]

- 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)

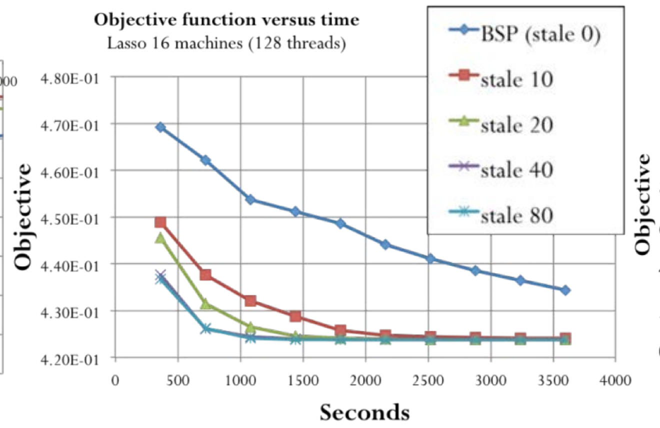


Async Speed + BSP-like Guarantees, across algorithms

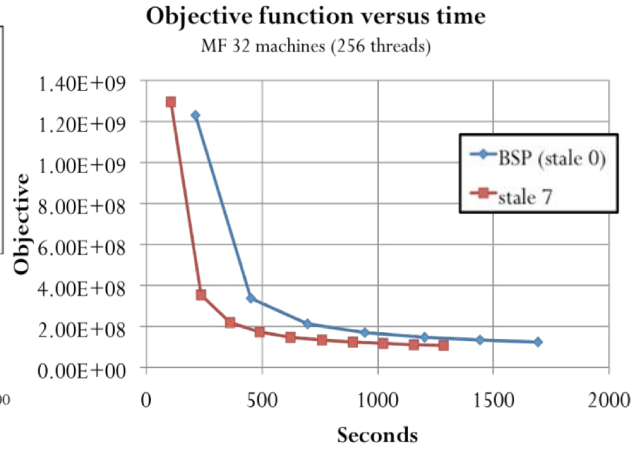
- Scale up Data Parallelism without long BSP synchronization time
- Effective across multiple algorithms, e.g. LDA, Lasso, Matrix Factorization:



LDA



LASSO

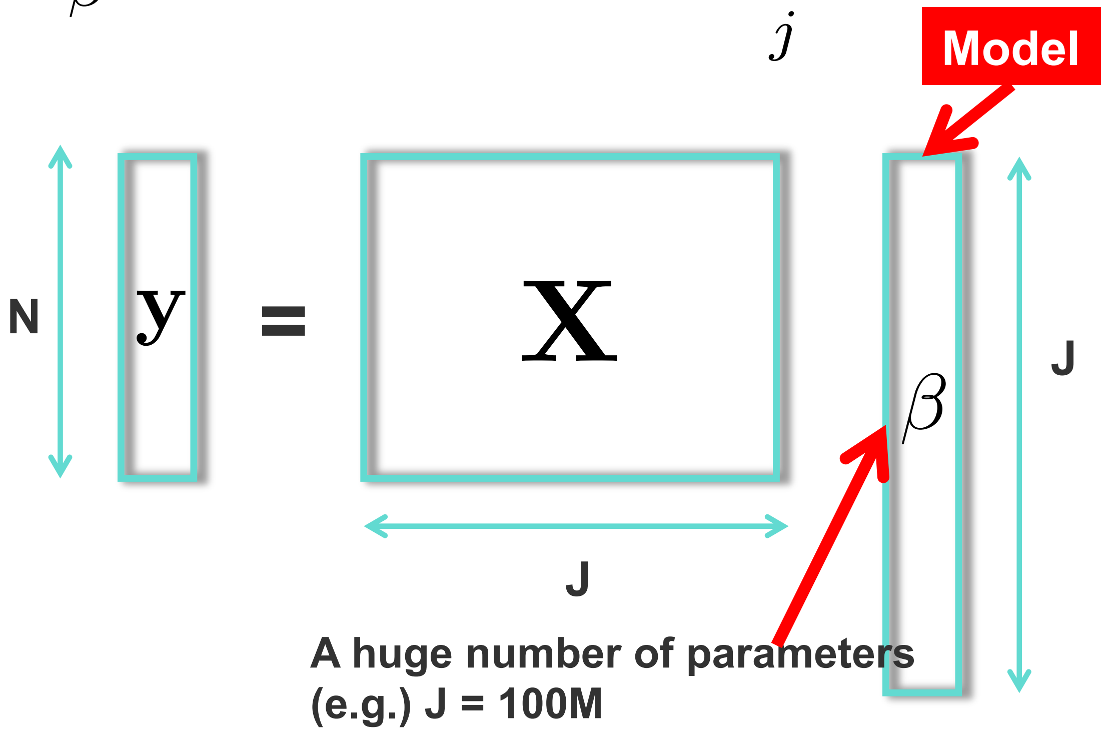


Matrix Fact.

Challenges in Model Parallelism

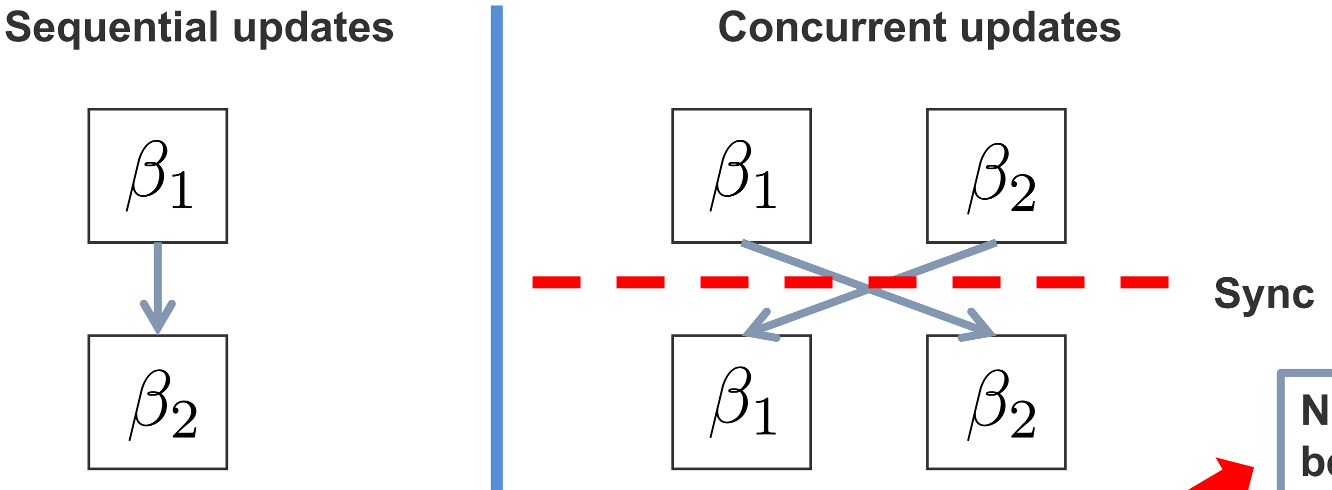
- Recall Lasso regression:

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



Challenge 1: Model Dependencies

- Concurrent updates of β may induce errors

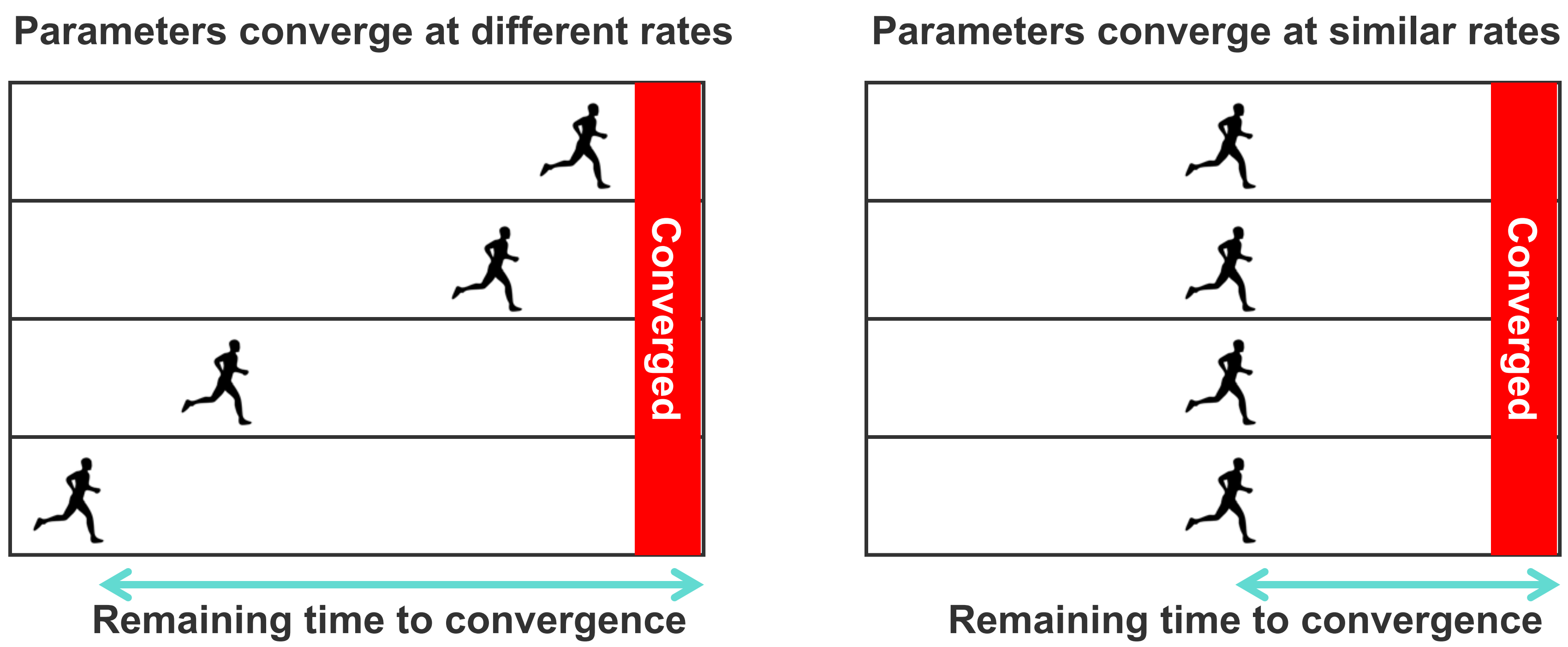


Need to check $\mathbf{x}_1^T\mathbf{x}_2$ before updating parameters

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

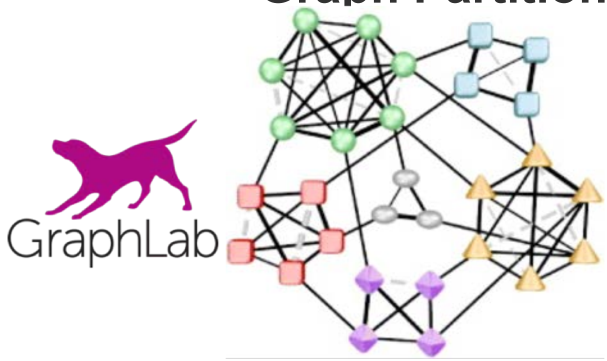


- Time-to-convergence determined by slowest parameters
- How to make slowest parameters converge quicker?

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



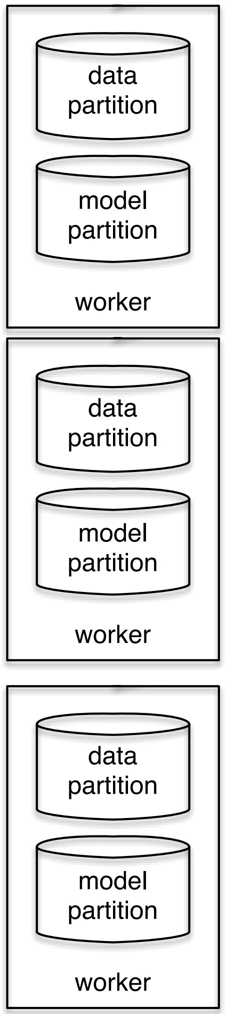
???

Is full consistency really necessary for ML?

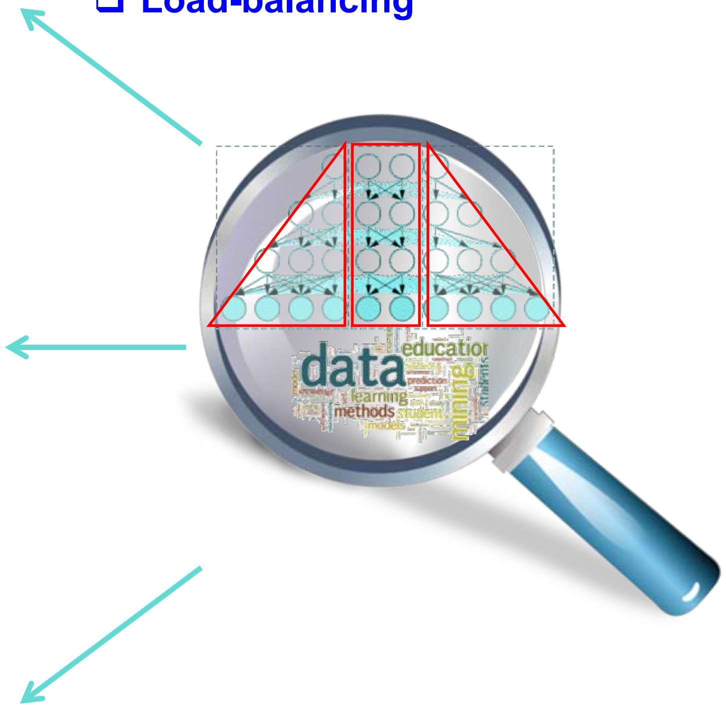
Random Partition

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()

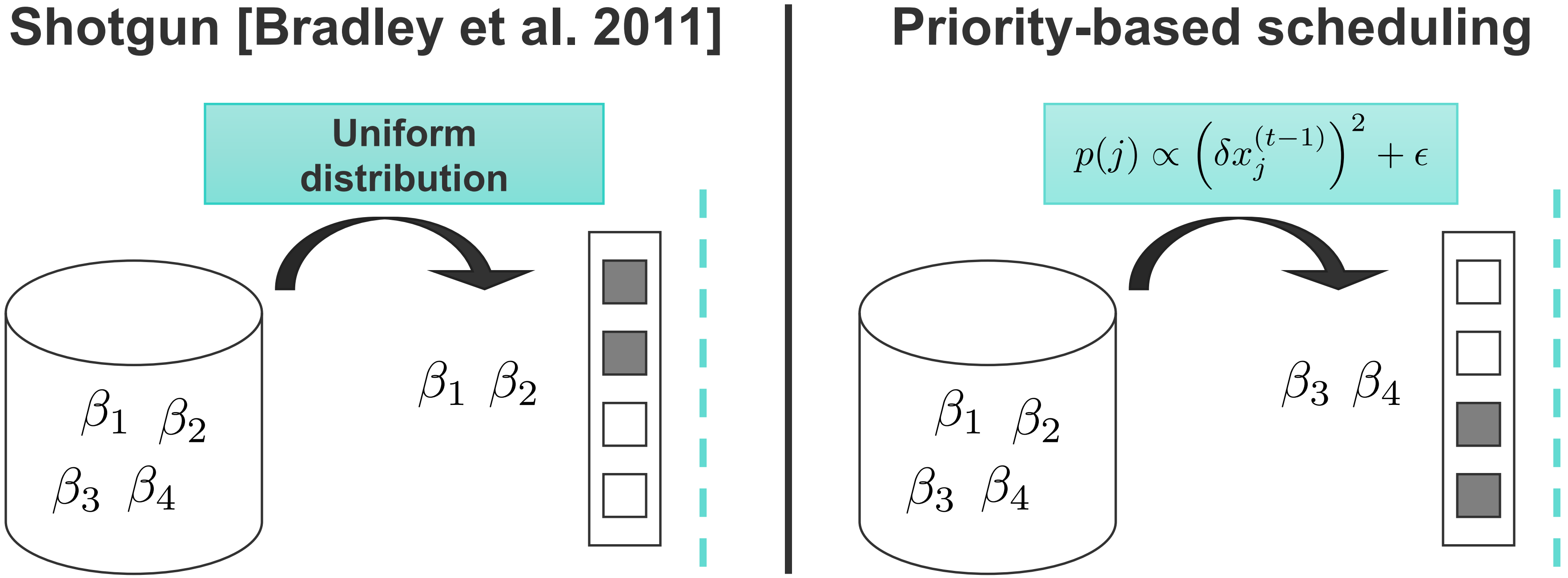
```
schedule() {
  // Select U vars x[j] to be sent
  // to the workers for updating
  ...
  return (x[j_1], ..., x[j_U])
}
```

```
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]

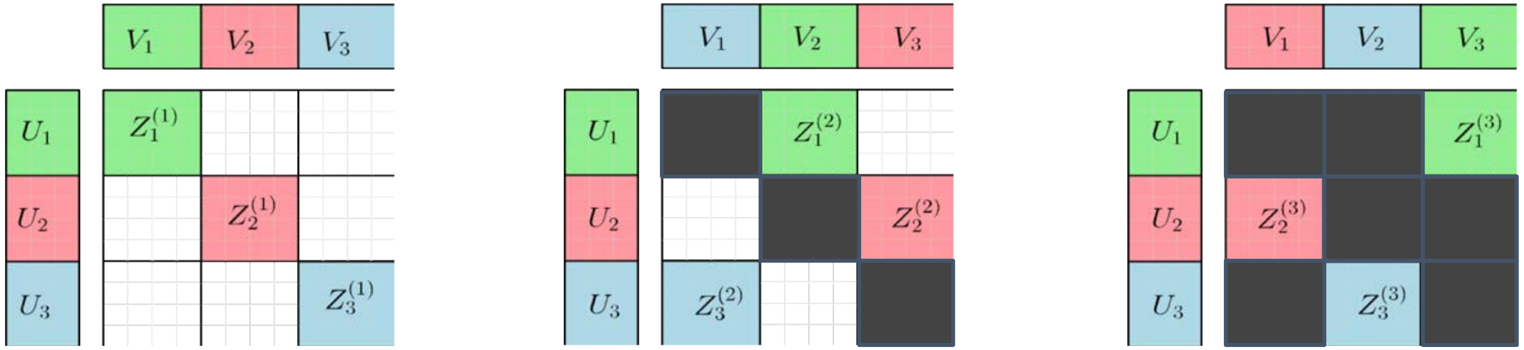
- Choose params to update based on convergence progress
  - Example: sample params with probability proportional to their recent change
  - Approximately maximizes the convergence progress per round



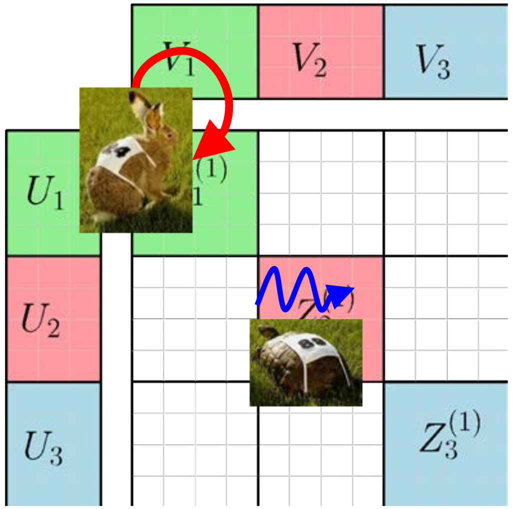
Schedule 2: Block-based (with load balancing)

[Kumar et al., 2014]

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

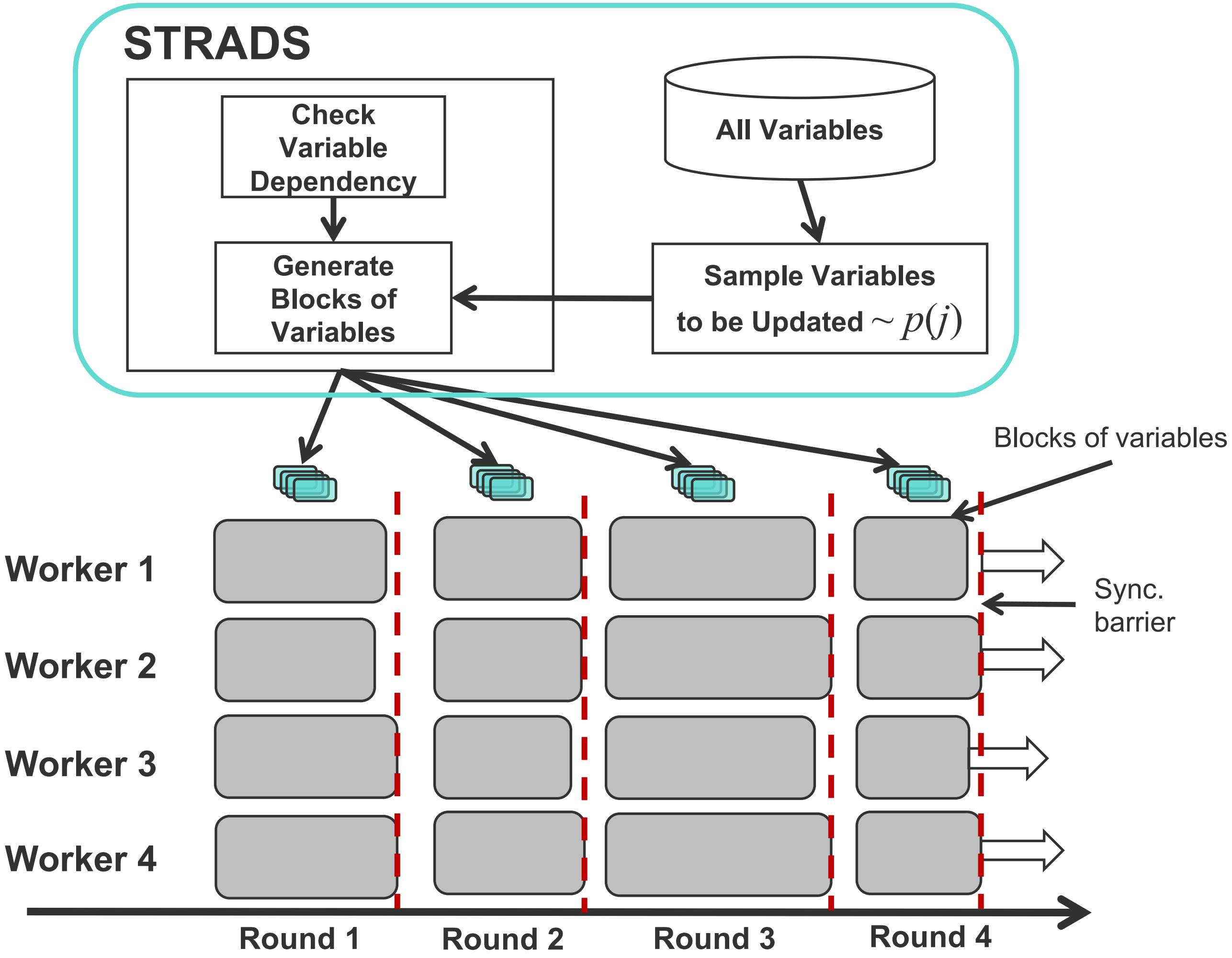


Blocks with less data/para or experience less straggling run more iterations
Automatic load-balancing + better convergence



Structure-aware Dynamic Scheduler (STRADS)

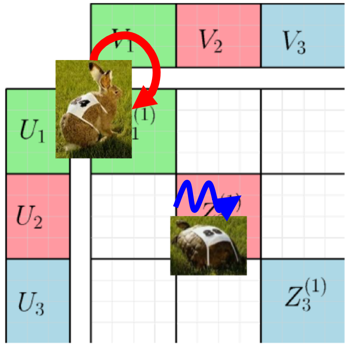
[Lee et al., 2014, Kumar et al., 2014]



- **Priority Scheduling**

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

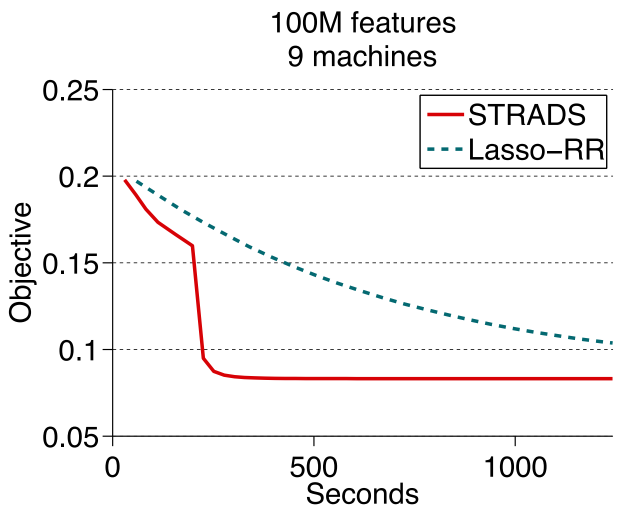
- **Block scheduling**



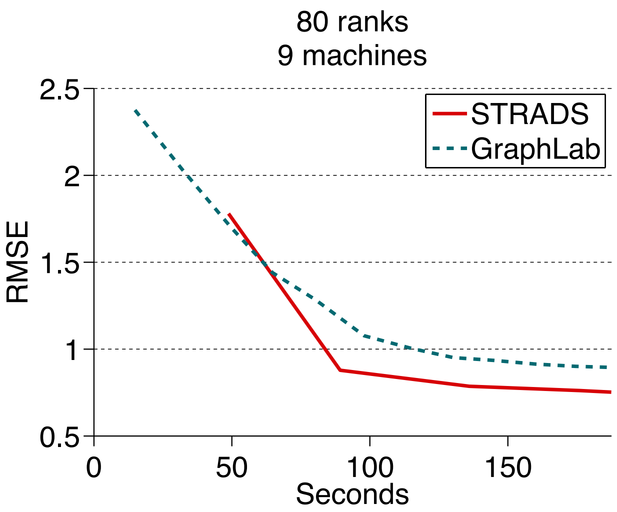
[Kumar, Beutel, Ho and Xing, ***Fugue: Slow-worker agnostic distributed learning***, AISTATS 2014]

Avoids dependent parallel updates, attains near-ideal convergence speed

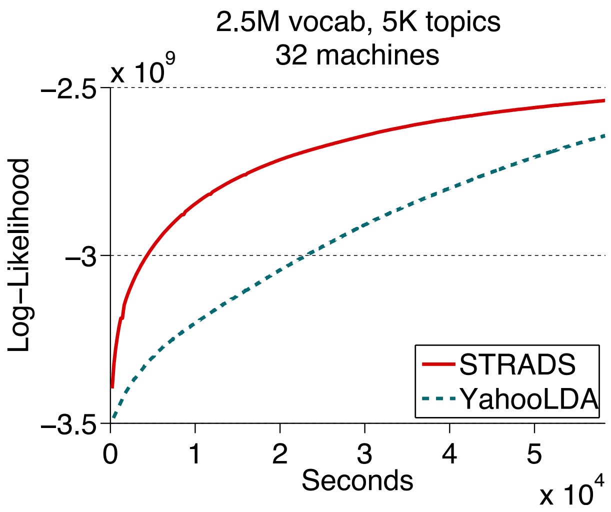
- STRADS+SAP achieves better speed and objective



Lasso



MF

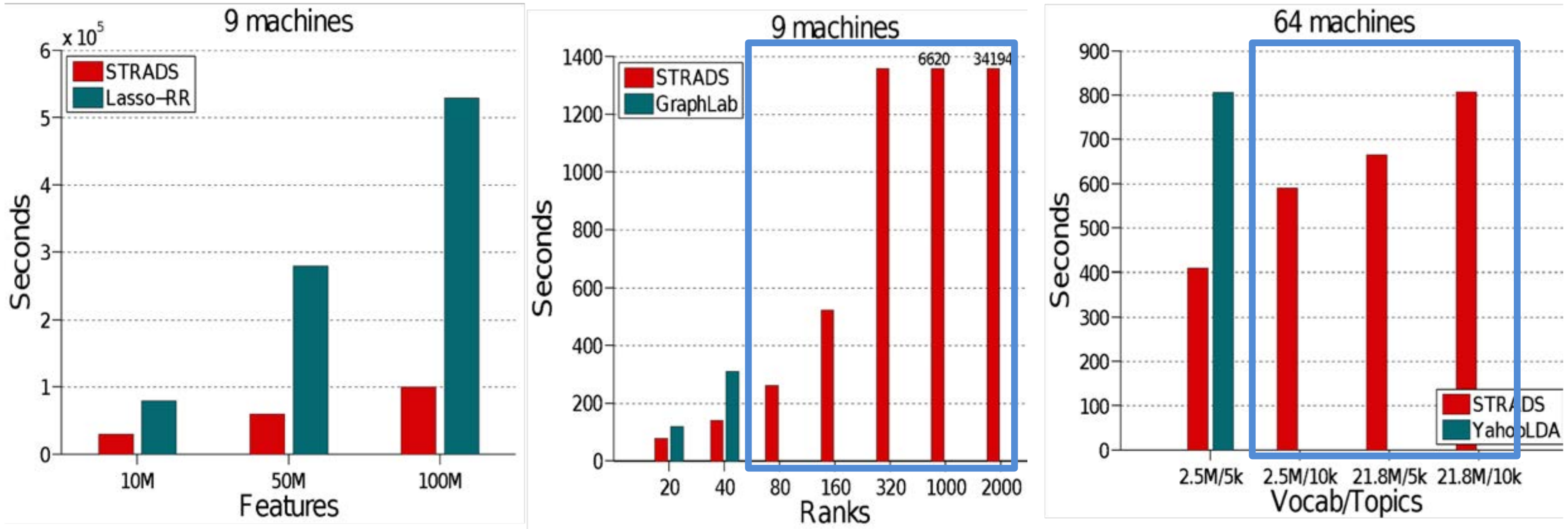


LDA

Efficient for large models

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

Lasso **MF** **LDA**

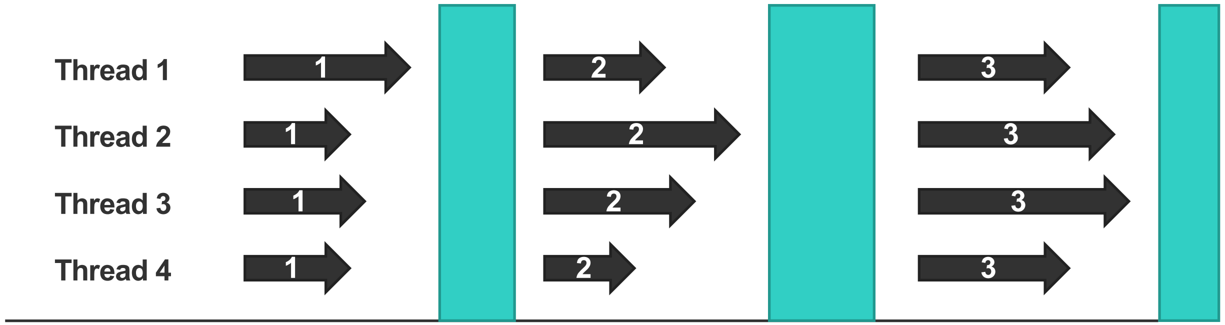


Theory of real-world distributed ML systems

- What guarantees still hold in parallel setting? Under what conditions?
- Computational and communications costs cannot be ignored
  - Real-world ML running time is heavily influenced by them
- Asynchronous or bounded-async approaches can empirically work better than synchronous approaches
  - Async => no serializability… why does it still work?
- Parallelization requires data and/or model partitioning
  - Want partitioning strategies that are provably correct
  - When/where is independence violated? What is the impact on algorithm correctness?

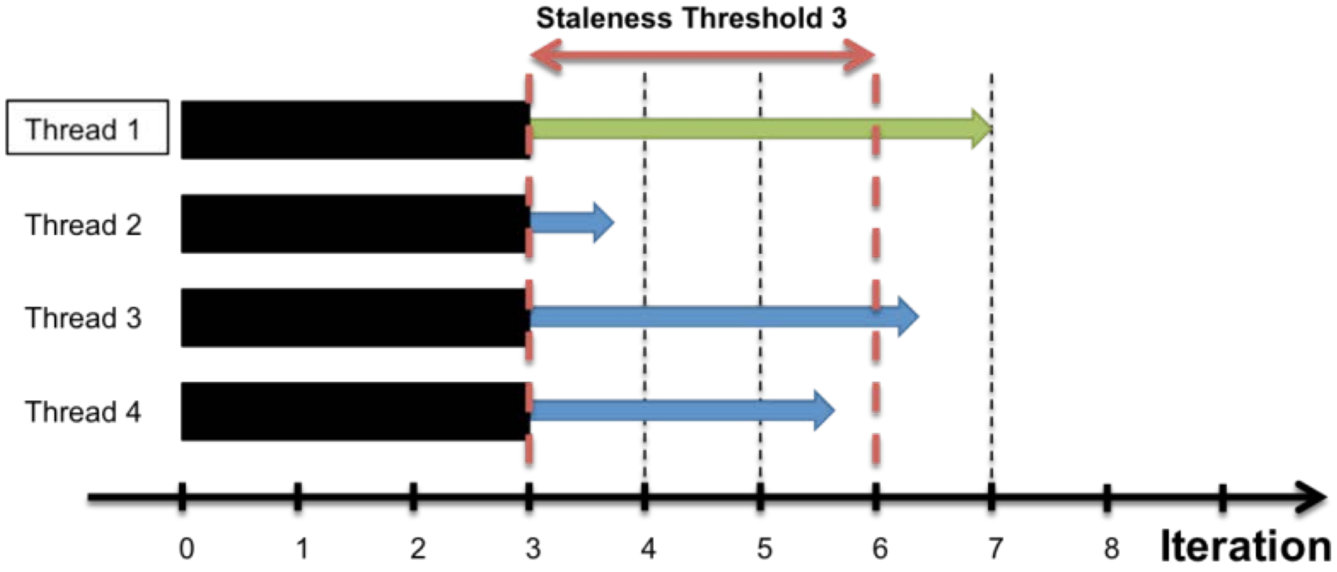
Background: Bridging Models for Parallel Programming

- 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



Background: Bridging Models for Parallel Programming

- 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



Background: Types of Convergence 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

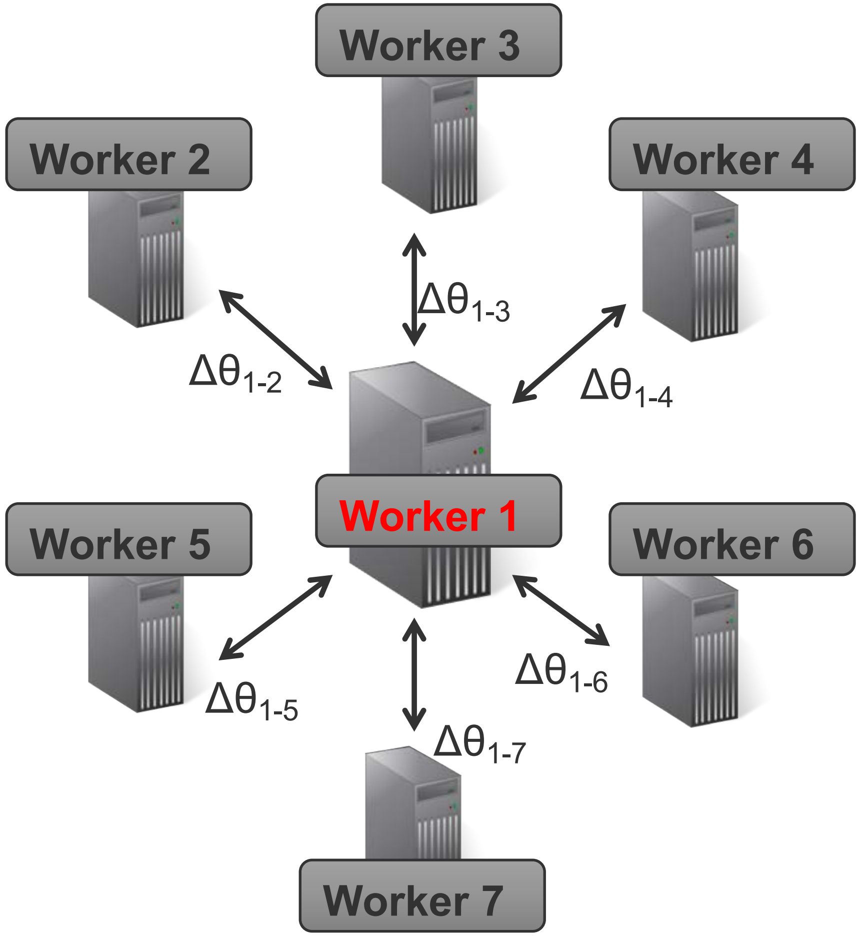
- Guarantees can be for Data-parallel, Model-parallel, or Data+Model-parallel

BAP Data Parallel: Why can't we do value-bounding?

- Seemingly-natural Idea: limit model parameter difference $\Delta\theta_{i\text{-}j} = ||\theta_i - \theta_j||$ between machines i,j to not exceed a given threshold

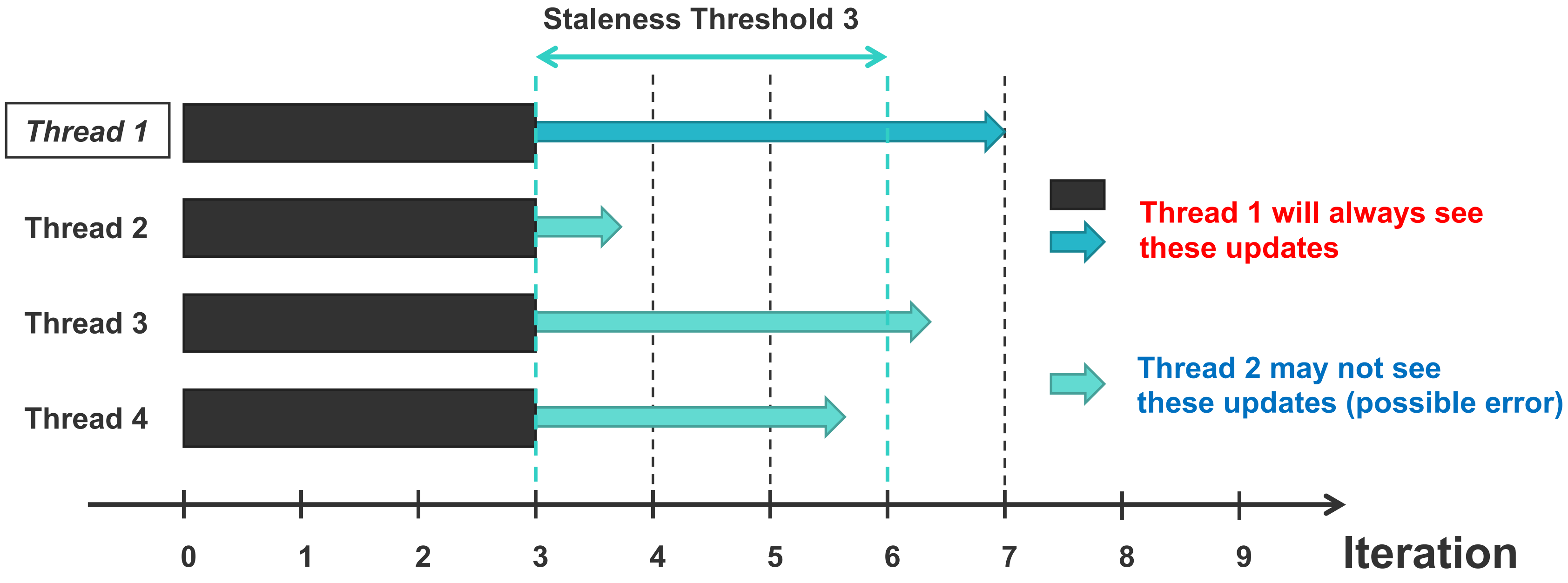
- Not practical!
  - To guarantee that $\Delta\theta_{i\text{-}j}$ has not exceeded the threshold, machines must wait to communicate with each other
  - No improvement over synchronous execution!

- 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]



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

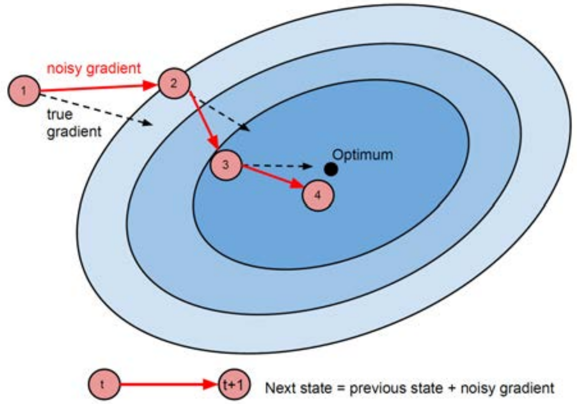
Consequence:

- 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]

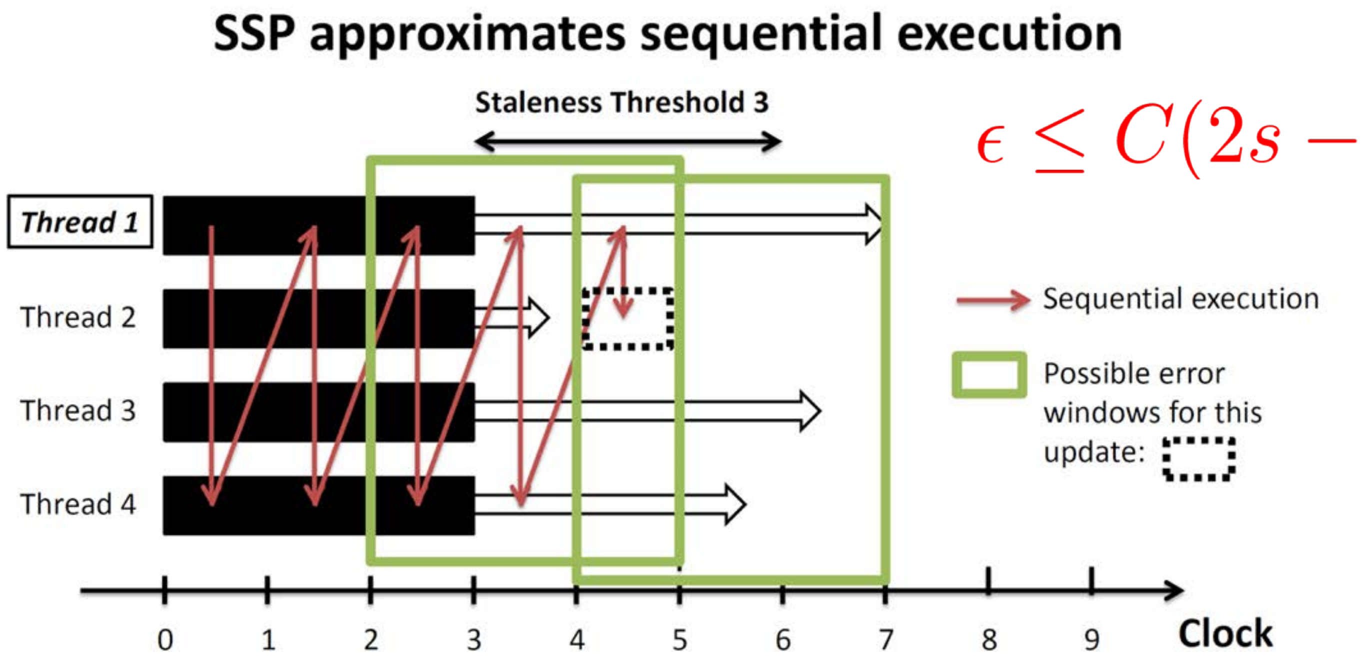
- Goal: minimize convex $f(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x})$
  (Example: Stochastic Gradient)
  - L-Lipschitz, problem diameter bounded by F^2
  - Staleness s, using P threads across all machines
  - Use step size $\eta_t = \frac{\sigma}{\sqrt{t}}$ with $\sigma = \frac{F}{L\sqrt{2(s+1)P}}$
- (E)SSP converges according to
  - Where T is the number of iterations



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

- 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?



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

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

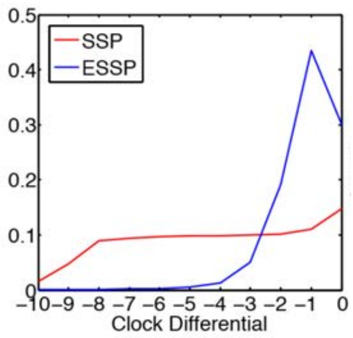
SSP versus ESSP: What is the difference?

- 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,

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

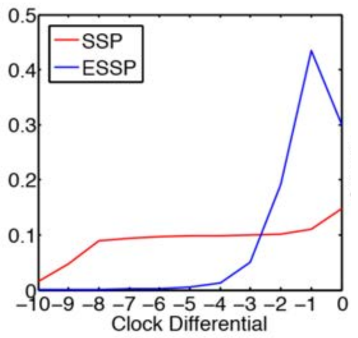
$$R[X] := \sum_{t=1}^{T} f_t(\tilde{x}_t) - f(x^*) \qquad \bar{\eta}_T = \frac{\eta^2 L^4(\ln T + 1)}{T} = o(T)$$

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

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

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

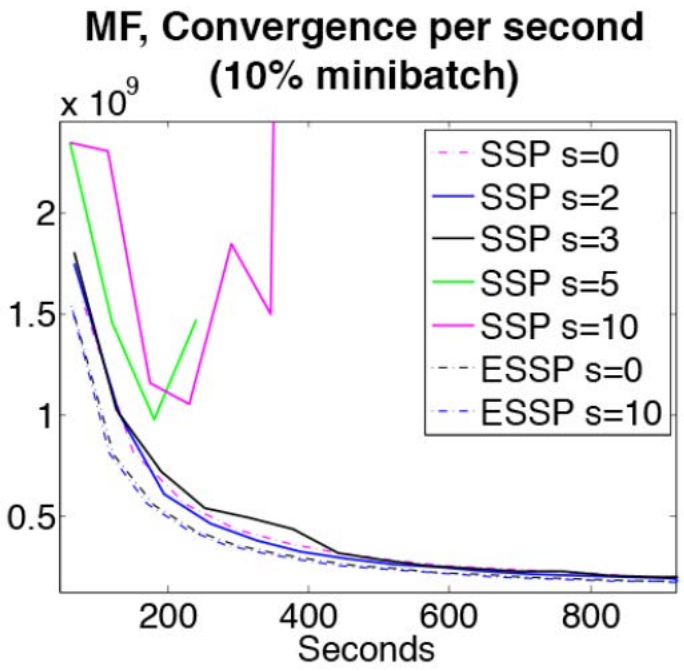
Explanation: The variance in the (E)SSP parameter estimate monotonically decreases when close to an optimum.

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: higher 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



BAP and Model-parallel?

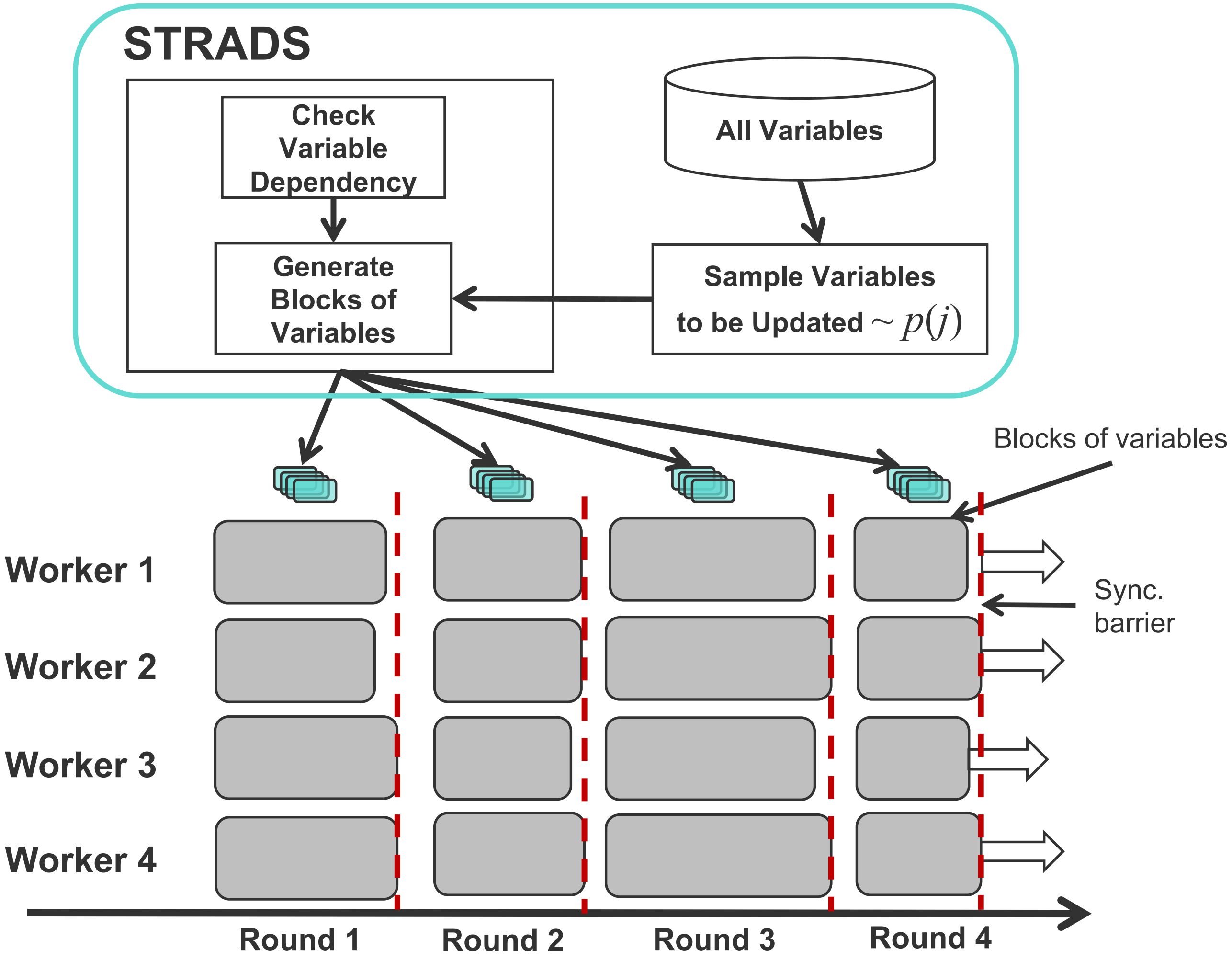
- Further Reading
  - *On Convergence of Model Parallel Proximal Gradient Algorithm for Stale Synchronous Parallel System*, Zhou et al., AISTATS 2016

- Intuition
  - Model-parallel sub-problems become nearly independent with proper scheduling
  - Has similarities to Hogwild [Recht et al., 2011], but…
    - Hogwild relies on atomic operations for consistency – only practical for single-machine
    - BAP+Model-parallel relies on BAP for consistency – implementable for real-world distributed systems

- Potentially better per-iteration convergence than BAP data-parallel

Scheduled Model Parallel: Dynamic/Block Scheduling

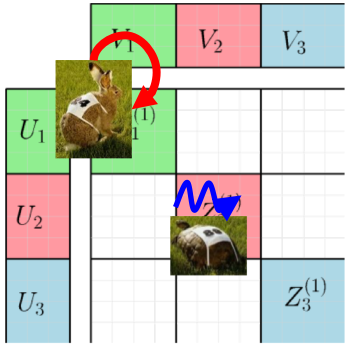
[Lee et al. 2014, Kumar et al. 2014]



- **Priority Scheduling**

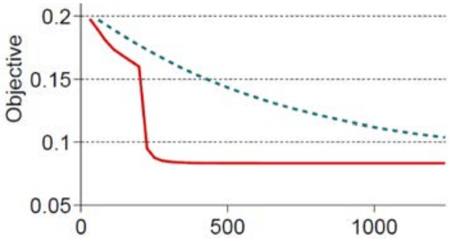
$$\{\beta_j\} \sim \left(\delta\beta_j^{(t-1)}\right)^2 + \eta$$

- **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

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

- 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 $\beta^{(t)}_{ideal}$ be the parameter trajectory due to ideal scheduling

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

Theorem: After *t* iterations, we have

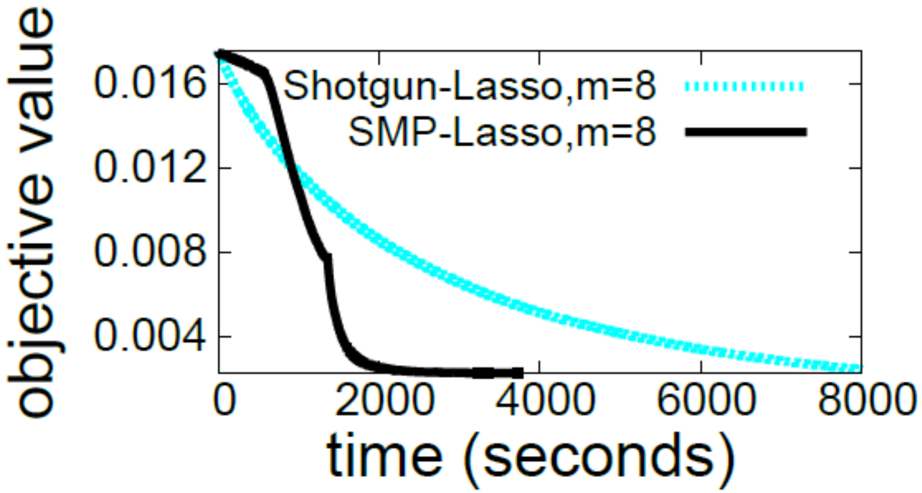
$$E[|\beta^{(t)}_{ideal} - \beta^{(t)}_{dyn}|] \leq C \frac{2M}{(t+1)^2} \mathbf{X}^\top \mathbf{X}$$

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)

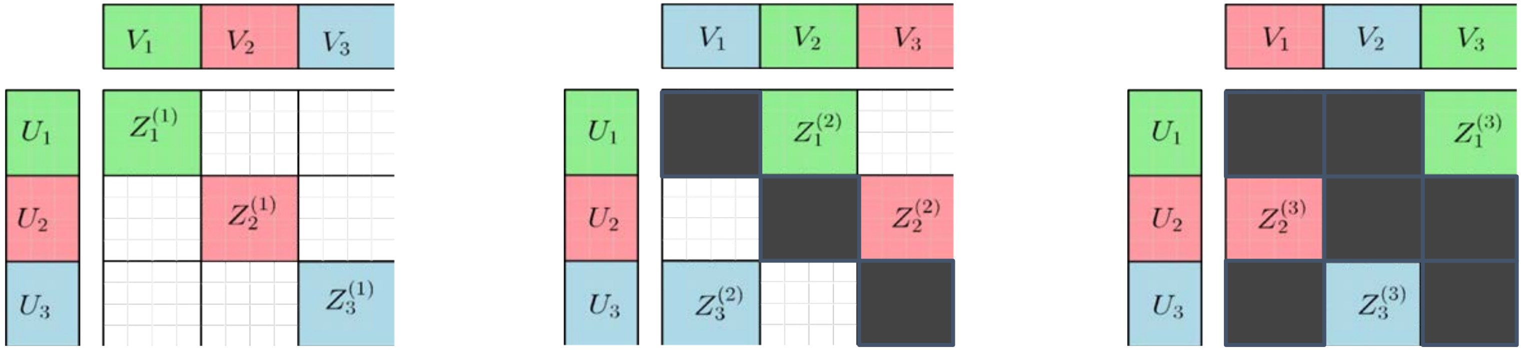


Scheduled Data+Model Parallel: Block-based Scheduling (with load balancing)

[Kumar et al. 2014]

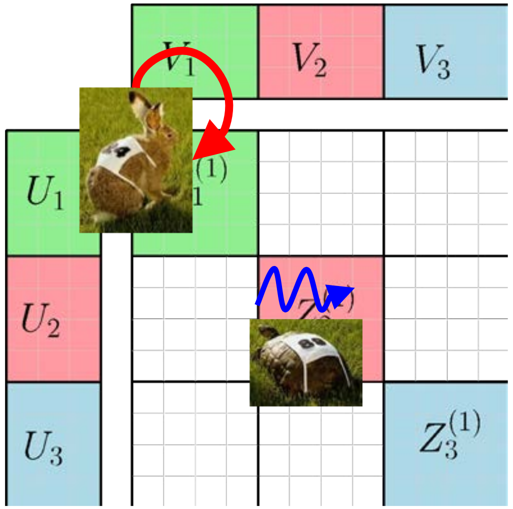
Partition data & model into *d* × *d* blocks

Run different-colored blocks in parallel



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:

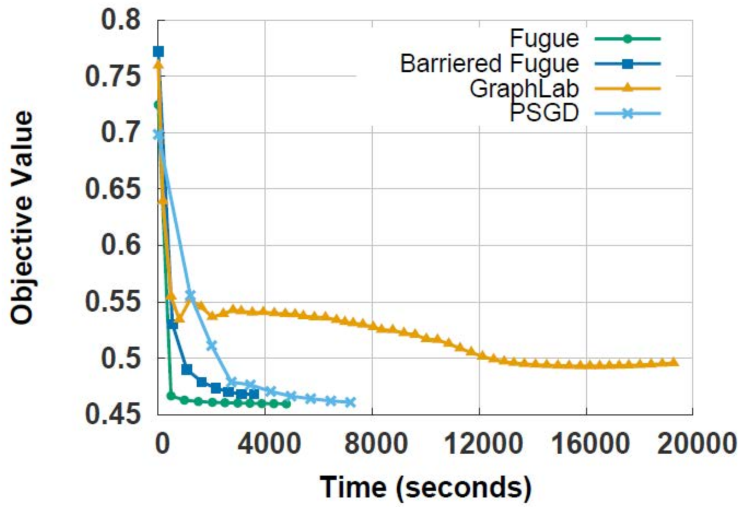
$$Var(\psi^{t+n_i}) = Var(\psi^t) - 2\eta_t n_i \Omega_0 (Var(\psi^t)) - 2\eta_t n_i \Omega_0 CoVar(\psi_t, \bar{\delta}_t) + \boxed{\eta_t^2 n_i \Omega_1} \boxed{+ \underbrace{\mathcal{O}(\eta_t^2 \rho_t) + \mathcal{O}(\eta_t \rho_t^2) + \mathcal{O}(\eta_t^3) + \mathcal{O}(\eta_t^2 \rho_t^2)}_{\Delta_t}}$$

- 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



Distributed ML Systems – Summary

- Real-world distributed systems are never ideal
  - Slow communication, uneven computation speed
  - Naïve Bulk Synchronous Parallel (BSP) can be slower than non-parallel implementation!
- Solution 1: Bounded-Asynchronous Parallel (BAP)
  - Exploit properties of ML algorithm convergence
  - Stale communication mitigates non-idealness in distributed systems
  - Applicable to data-parallel and model-parallel strategies
- Solution 2: Scheduled Model Parallelism (SMP)
  - Exploit ML model structural properties
  - Re-ordering of computation mitigates non-idealness in distributed systems
  - SMP is a model-parallel strategy that is compatible with data-parallelism
- Theoretical analysis
  - Convergence guarantees exist for BAP, SMP
  - Rates are influenced by
    - ML model/algorithm properties: learning rates and model structure
    - Distributed systems properties: number of parallel machines, staleness