School of Computer Science
Carnegie Mellon

Probabilistic Graphical Models

Distributed Algorithms for ML

David (Wei) Dai

Lecture 21, April 5, 2017

Massive Data

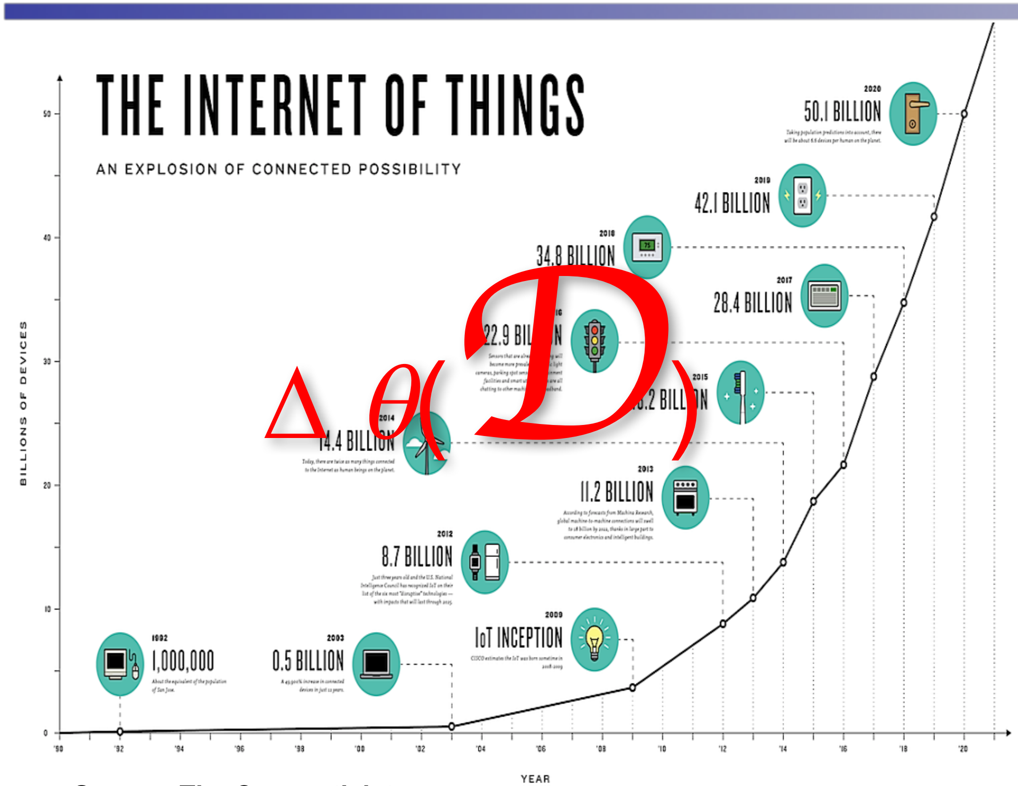
1B+ USERS
30+ PETABYTES

32 million pages

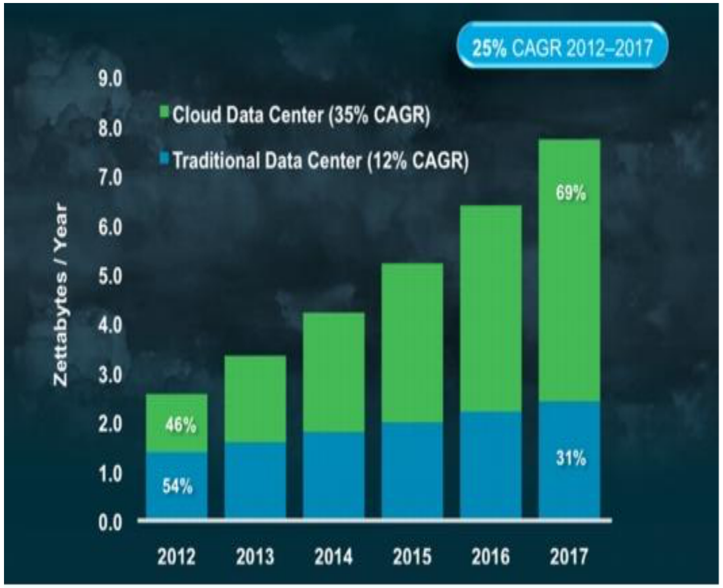
100+ hours video uploaded every minute

645 million users
500 million tweets / day

Challenge 1 – Massive Data Scale



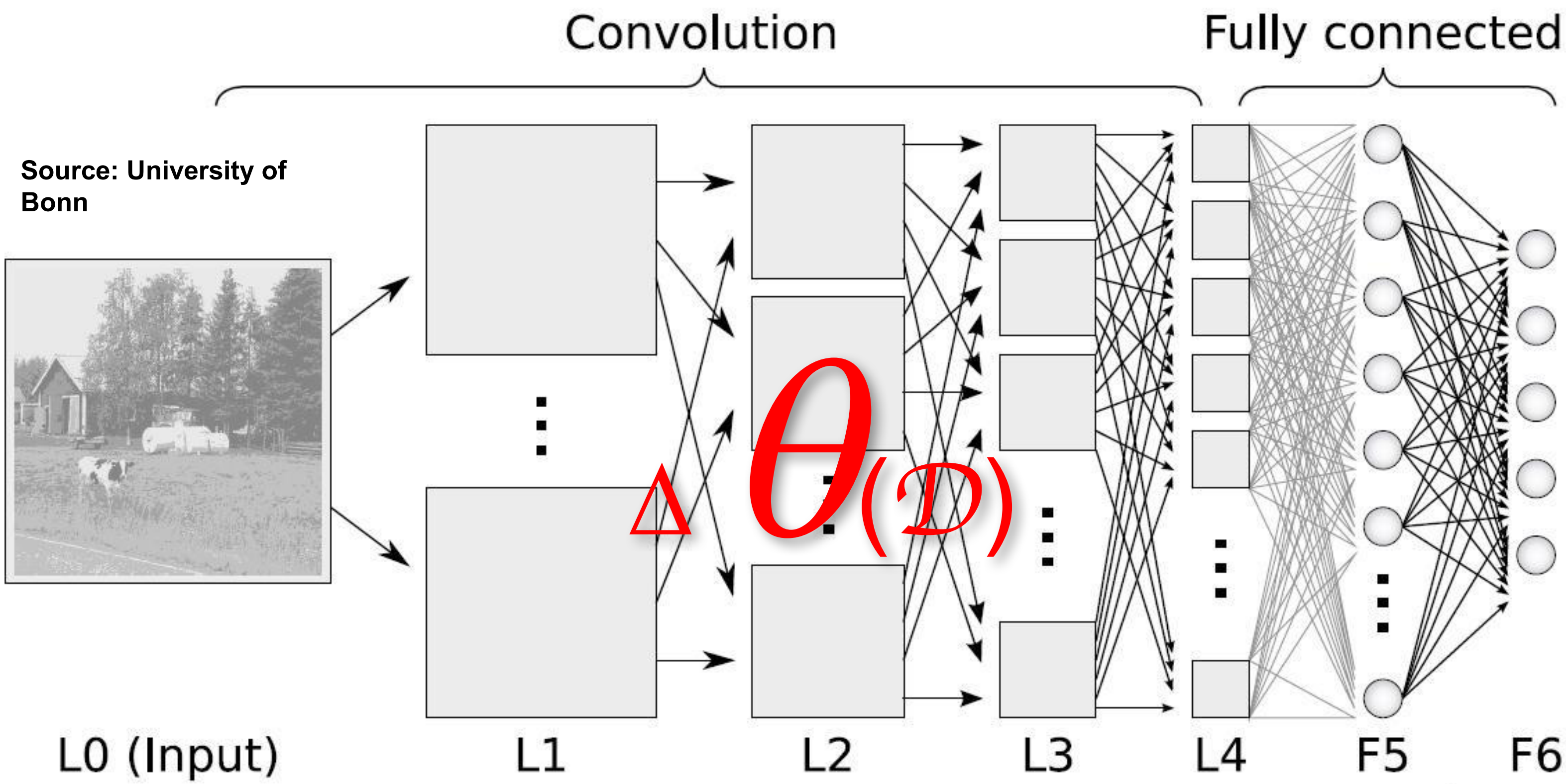
Source: The Connectivist



Source: Cisco Global Cloud Index

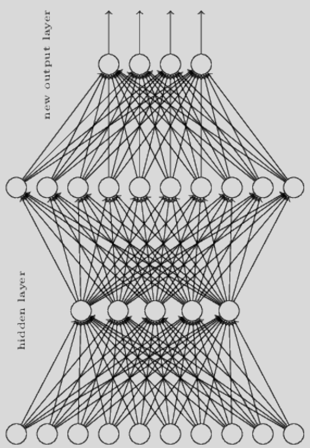
Familiar problem: data from 50B devices, data centers won't fit into memory of single machine

Challenge 2 –
Gigantic Model Size

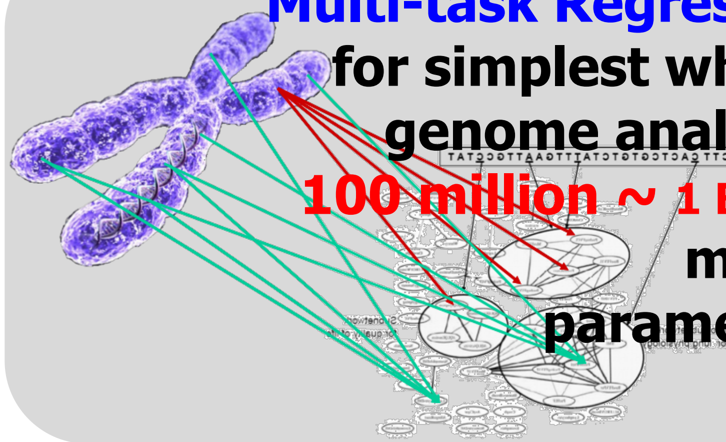


Maybe Big Data needs Big Models to extract understanding?
But models with >1 trillion params also won't fit!

Growing Need for Big and Contemporary ML Programs



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



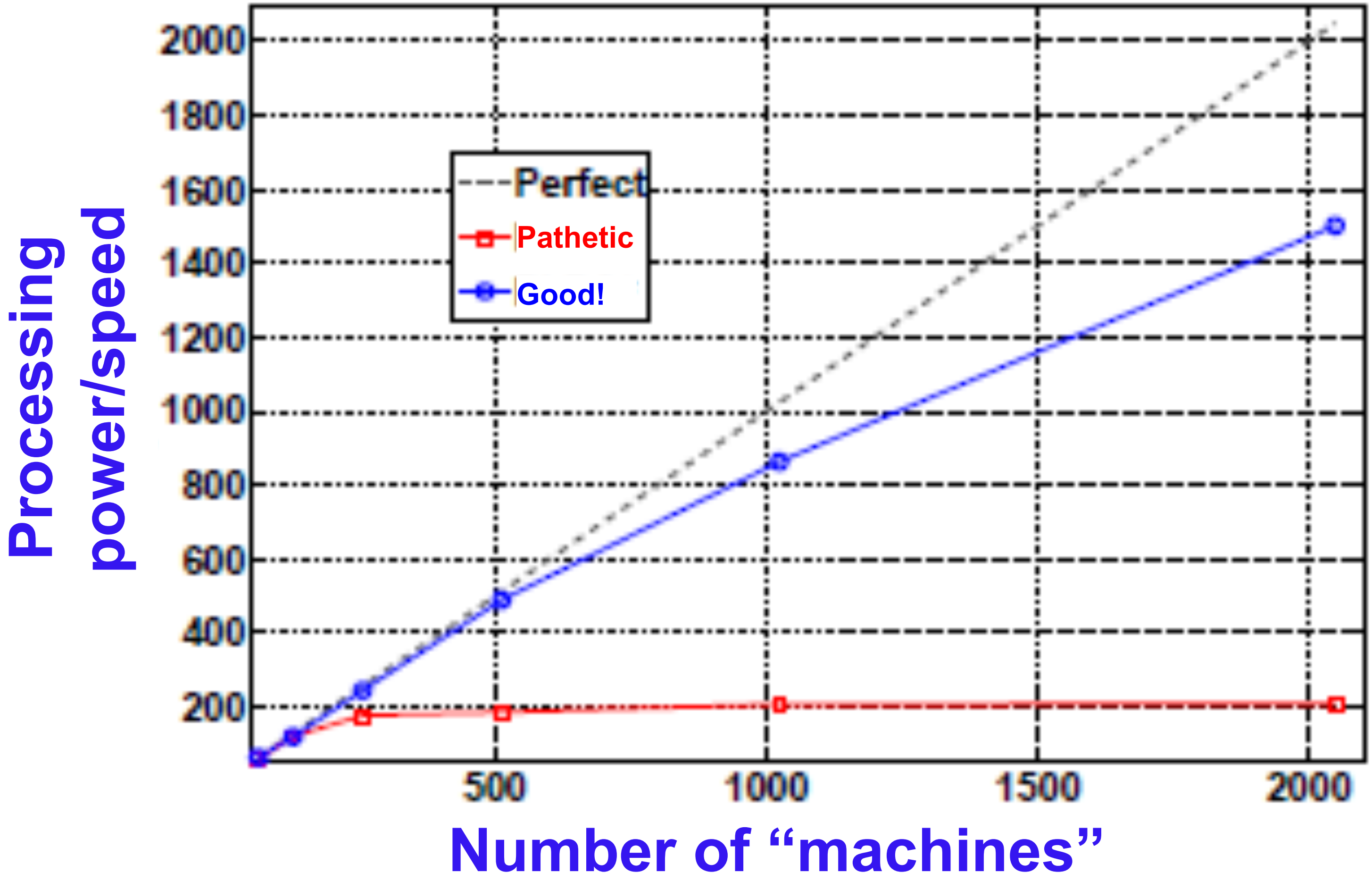
Multi-task Regression
for simplest whole-genome analysis:
100 million ~ 1 Billion
model parameters

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



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()
```

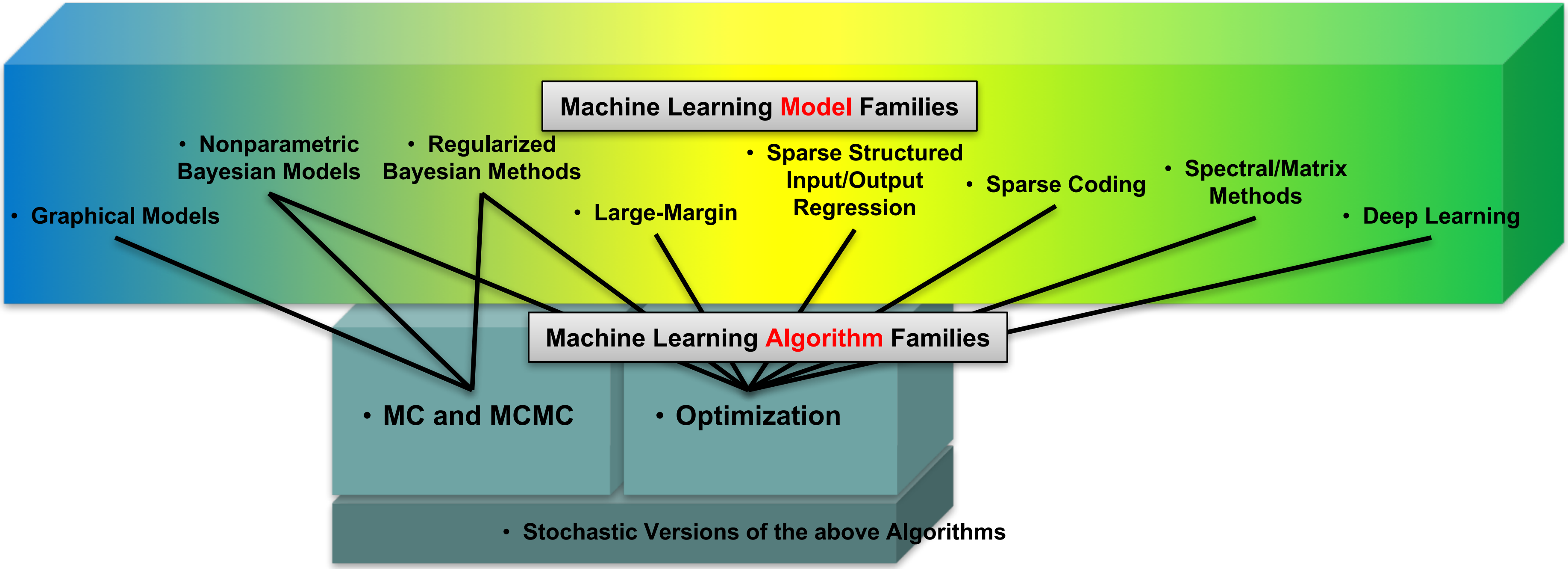
$$\vec{\theta}^{t+1} = g(\vec{\theta}^{t}, \; \Delta_f \vec{\theta}(\mathcal{D}))$$

```
  doOtherThings()
}
```

This computation needs to be scaled up !

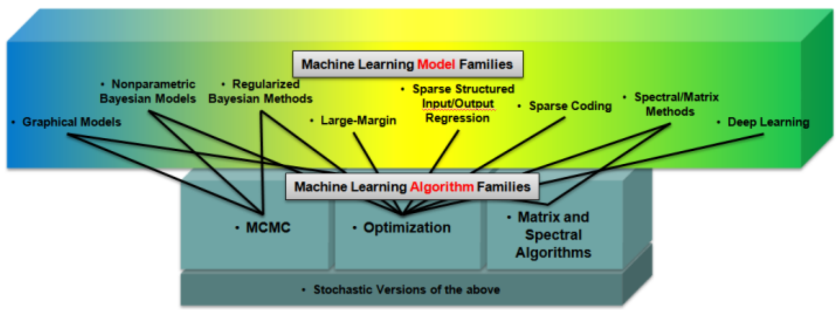
A "Classification" of ML Models and Tools

- 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)

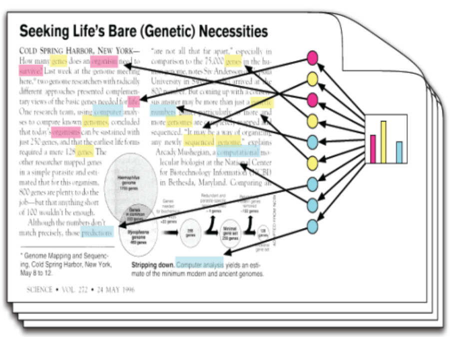


A “Classification” of ML Models and Tools

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

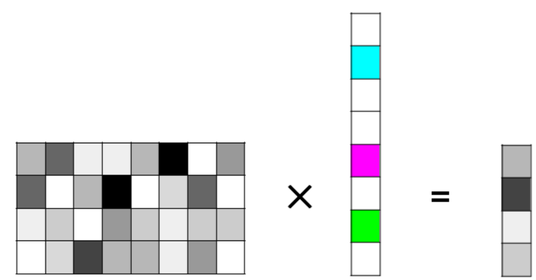


Probabilistic Programs



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

Optimization Programs

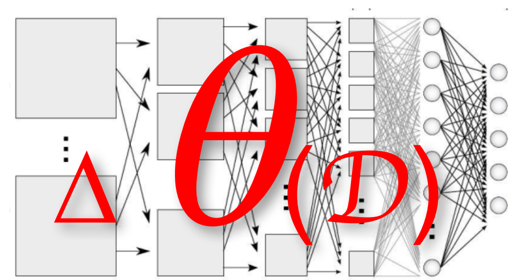


$$\sum_{i=1}^{N} \|y_i - X_i\beta\|_2^2 + \lambda \sum_{j=1}^{D} |\beta_j|$$

Parallelization Strategies

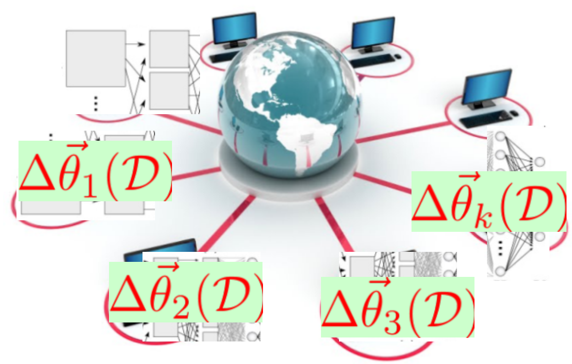
$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

New Model = Old Model + Update(Data)



Data Parallel

Model Parallel



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

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

Outline:
Optimization & MCMC Algorithms

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

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

Optimization Programs

Algorithm I: Stochastic Gradient Descent

- Consider an optimization problem:

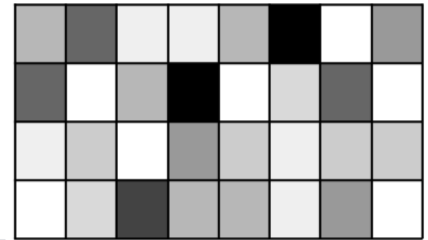
$$\min_x \mathbb{E}\{f(x,d)\}$$

- 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

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

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: $\sum_{i=1}^{N} \|y_i - X_i\beta\|_2^2$

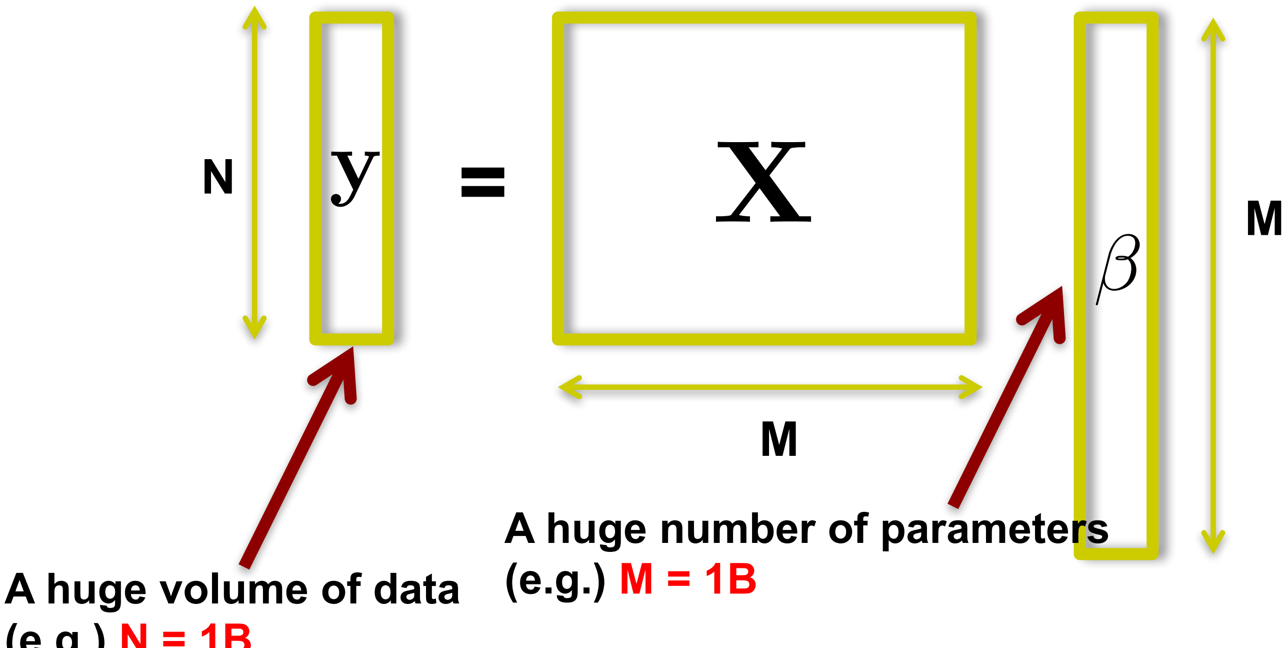
- Regularization r(θ)
  - L1 penalty on β to encourage sparsity: $\lambda \sum_{j=1}^{D} |\beta_j|$
  - λ is a tuning parameter

- Algorithms
  - Coordinate Descent
  - Stochastic Proximal Gradient Descent

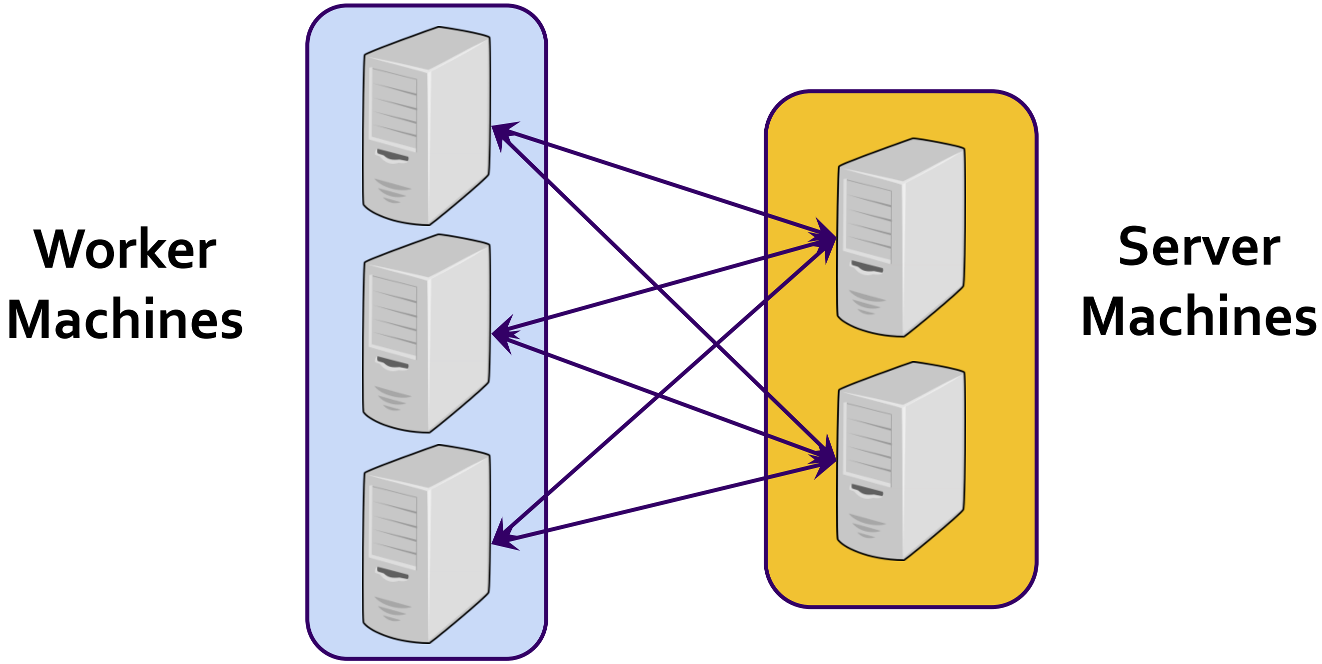
Challenge

- **Optimization programs:**

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



Distributed KV-Store for ML



- Model parameters are stored on PS machines and accessed via key-value interface (distributed shared memory)
- More in the next lecture

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

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

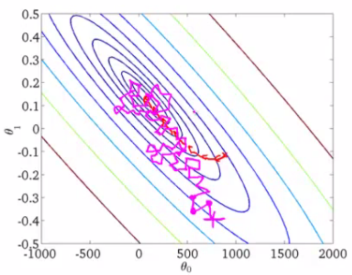
- Step 2: Proximal operator: perform soft thresholding on β

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

  - 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

Stochastic Gradient Descent

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



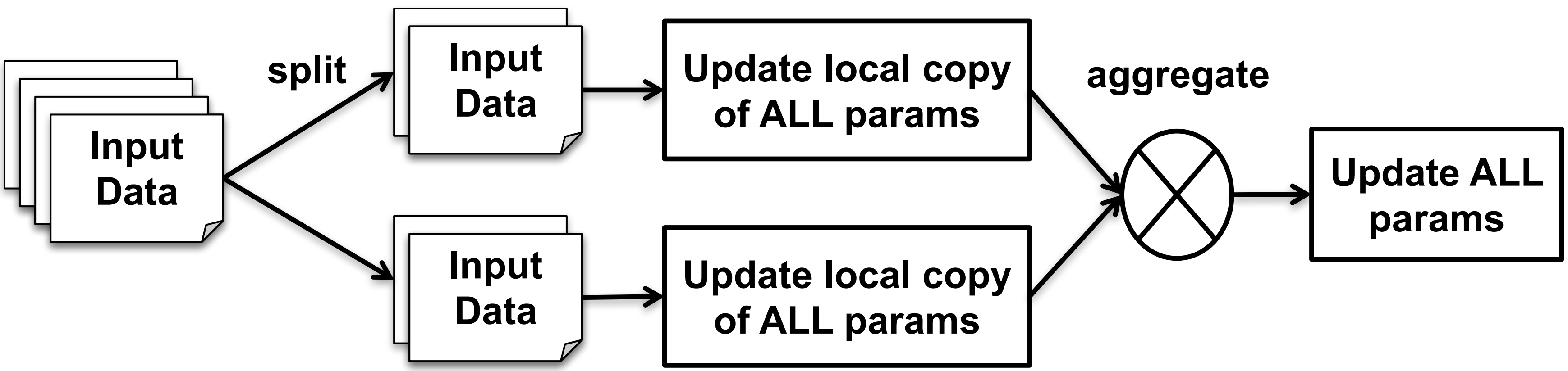
- **Traditional SGD compute gradients based on a single sample**

- **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

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

Learning Rates Problem in SGD

- Stochastic Gradient Descent

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \boldsymbol{g}_t$$

- Learning rate in SGD is difficult to tune.
- Big problem especially when the data is sparse:
  - Assume $\eta_1 = 0.1$

    $\eta_{1000} = 0.01$

  - If dimension 10 is sparse and is 0 for all first 999 minibatches, and only becomes non-zero in the 1000th minibatch.
  - $\boldsymbol{x}^{(10)}$ receives very small update → slow convergence

Adaptive Learning Rates (Adagrad)

- Instead of standard SGD

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \boldsymbol{g}_t$$

- Adagrad updates *each coordinate*

$$x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{\sum_{t'=1}^{t} g_{t',i}^2}} g_{t,i}$$

- Very good for sequential execution
- But with delay, very unstable.
  - Why?

Duchi et al 2011

Adaptive Revision

- Instead of Adagrad

$$x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{\sum_{t'=1}^{t} g_{t',i}^2}} g_{t,i}$$

- AdaRevision uses (approximately)

$$G_t^{\text{bck}} := g_t^2 + 2 g_t g_t^{\text{bck}}$$

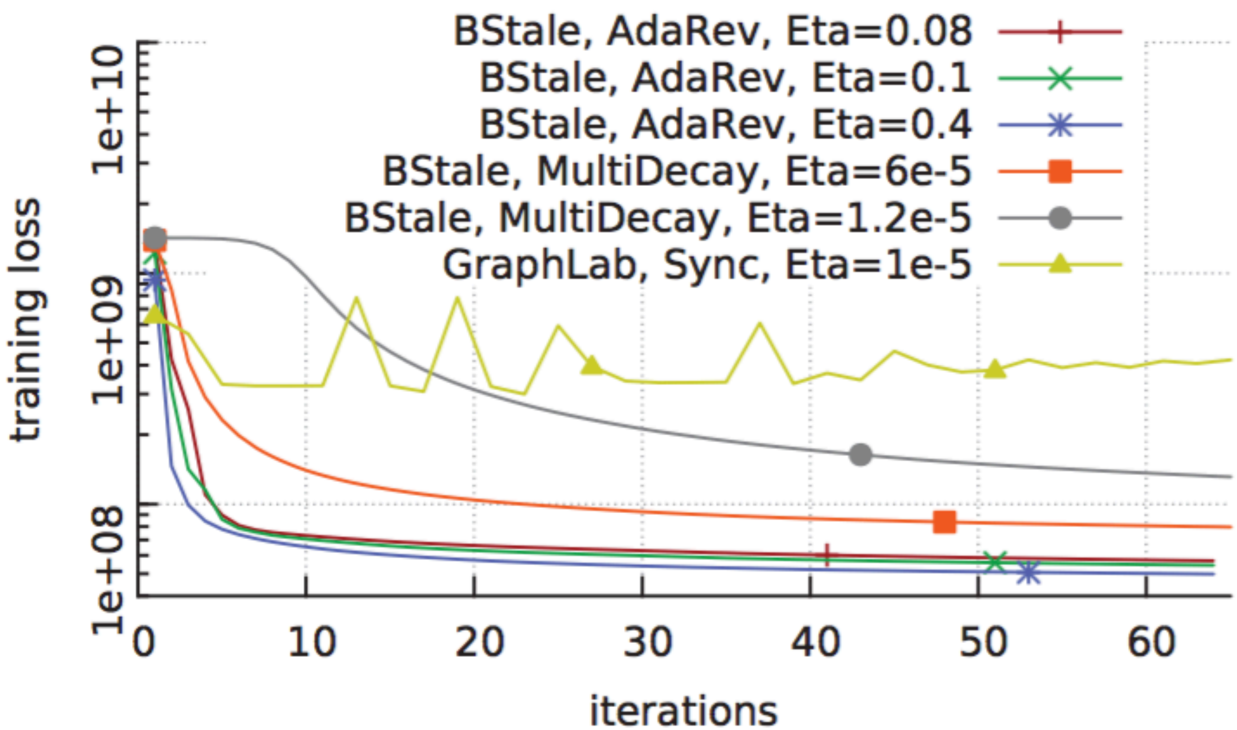
$$g_t^{\text{bck}} = \sum_{t' \in \text{delayed}} g_{t'}$$

$$\eta_t = \frac{\eta}{\sqrt{\sum_{t'} G_{t'}^{\text{bck}}}}$$

McMahan et al 2014

Adaptive Revision

- Adarevision is robust to delay.



Setup: matrix factorization on 16 threads single node

Wei et al 2015

Coordinate Descent
Case study: 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:

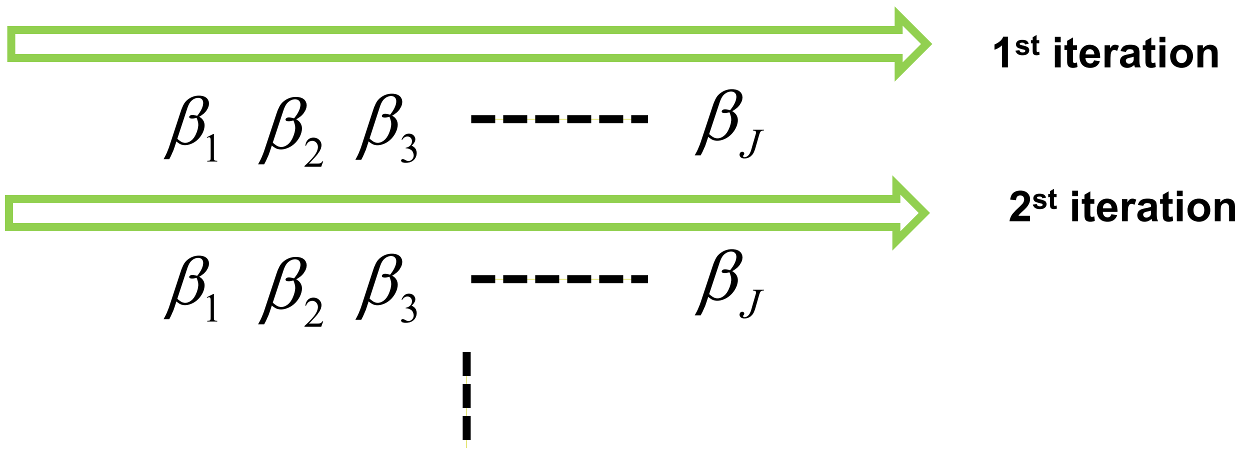
$$\beta_j = S\left\{\mathbf{x}_j^T(\mathbf{y} - \sum_{l \neq j} x_l \beta_l), \lambda\right\}$$

Soft thresholding

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

Coordinate Descent

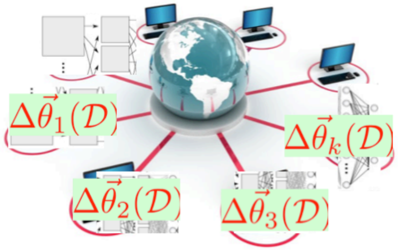
Update each regression coefficient in a cyclic manner



- **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**

Parallel Coordinate Descent

[Bradley et al. 2011]

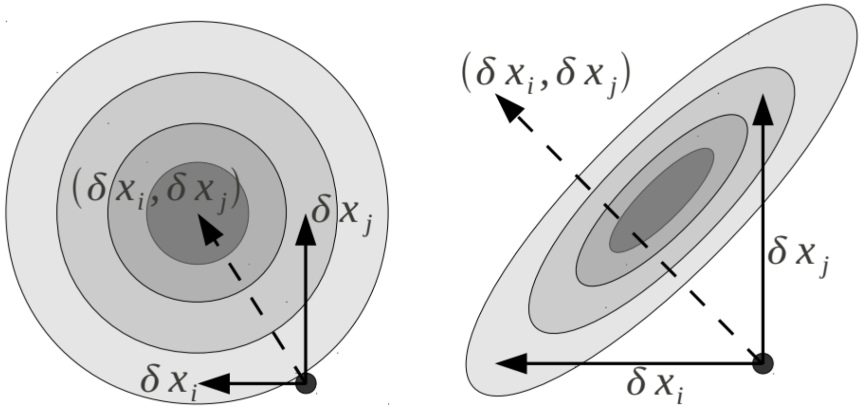


- Shotgun, a parallel coordinate descent algorithm
  - Choose parameters to update at random
  - Update the selected parameters in parallel
  - Iterate until convergence

- 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 $A^T A$
  - 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

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

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

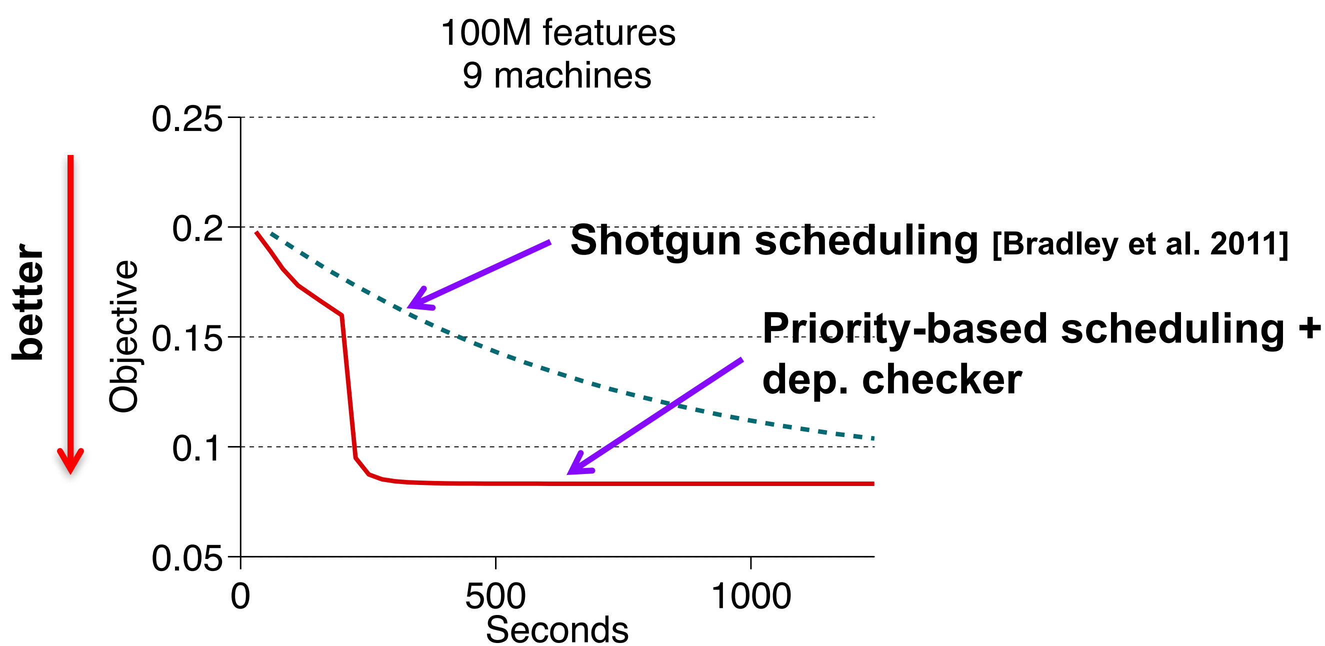
- Update step:
  - For all β_j chosen in Schedule step, in parallel, perform coordinate descent update

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

  - Repeat from Schedule step

Comparison: priority vs. random-scheduling

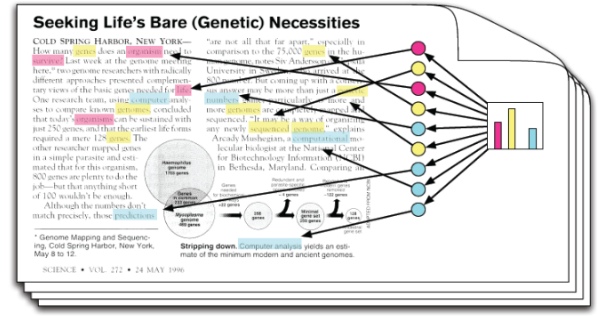
- Priority-based scheduling converges faster than Shotgun (random) scheduling



Probabilistic Programs

Case study: Topic Model (LDA)

Probabilistic Example: Topic Models



- 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}

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

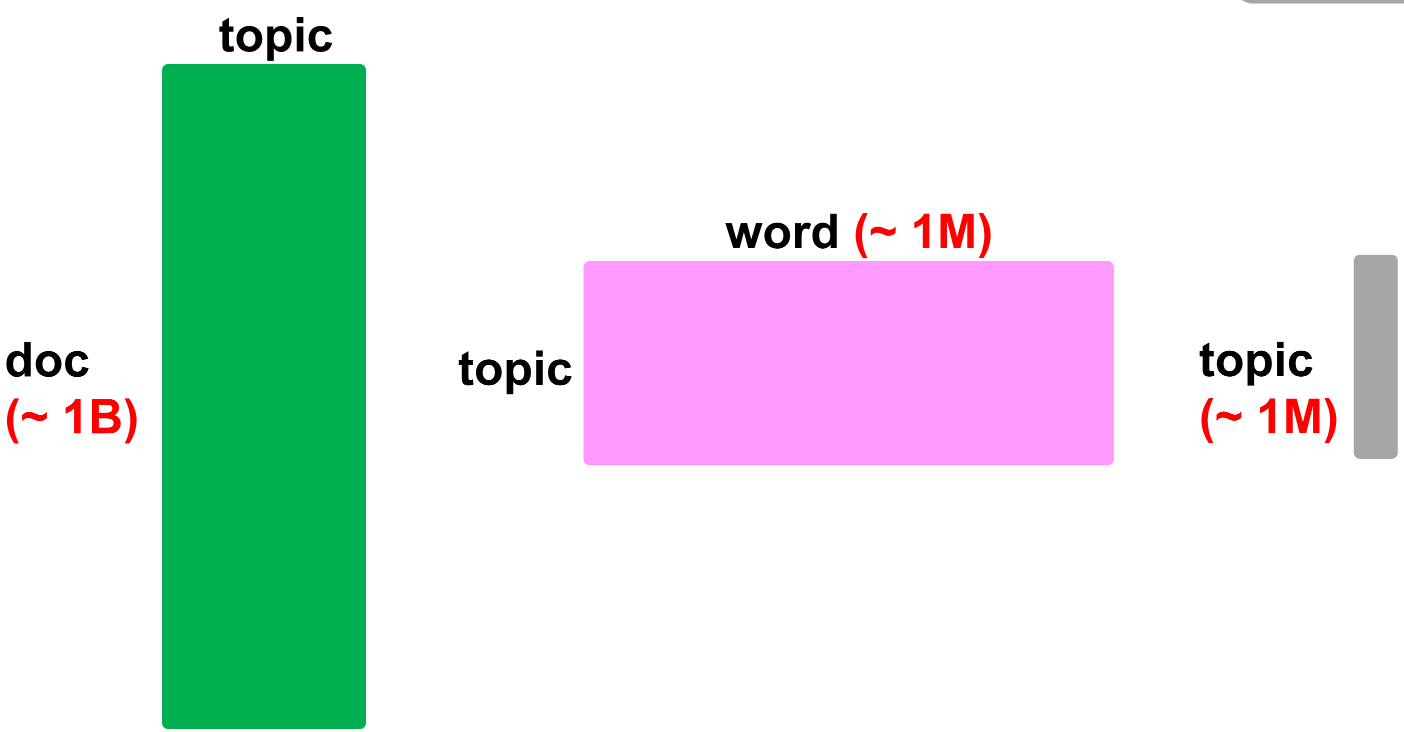
  - α, β are "hyperparameters" that control the Dirichet prior's strength

- Algorithm
  - Collapsed Gibbs Sampling

Challenge

- **Probabilistic programs**

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



Properties of Collapsed Gibbs Sampling (CGS)

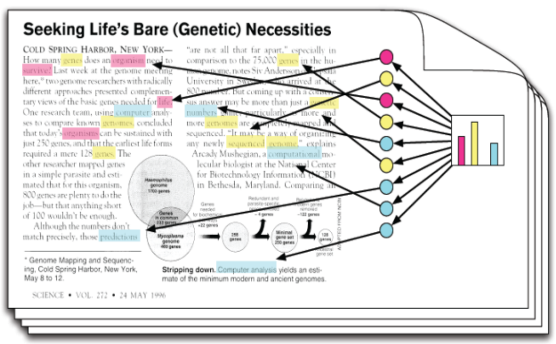
$$p(z_{ij} = k | x_{ij}, \delta_i, B) \propto (\delta_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + B_{k,x_{ij}}}{V\beta + \sum_{v=1}^{V} B_{k,v}}$$

- 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!

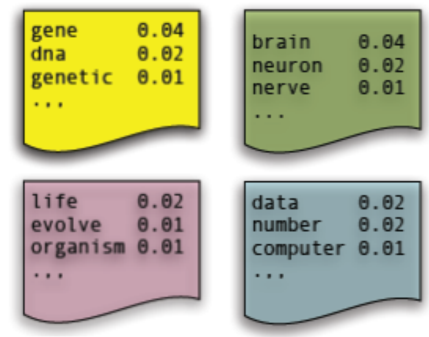
Probabilistic Example: Topic Models

Applications: Natural Language Processing, Information Retrieval

Data (Docs) = x_{ij}



Model (Topics) = B_k



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 $k_{new} = z_{ij}$

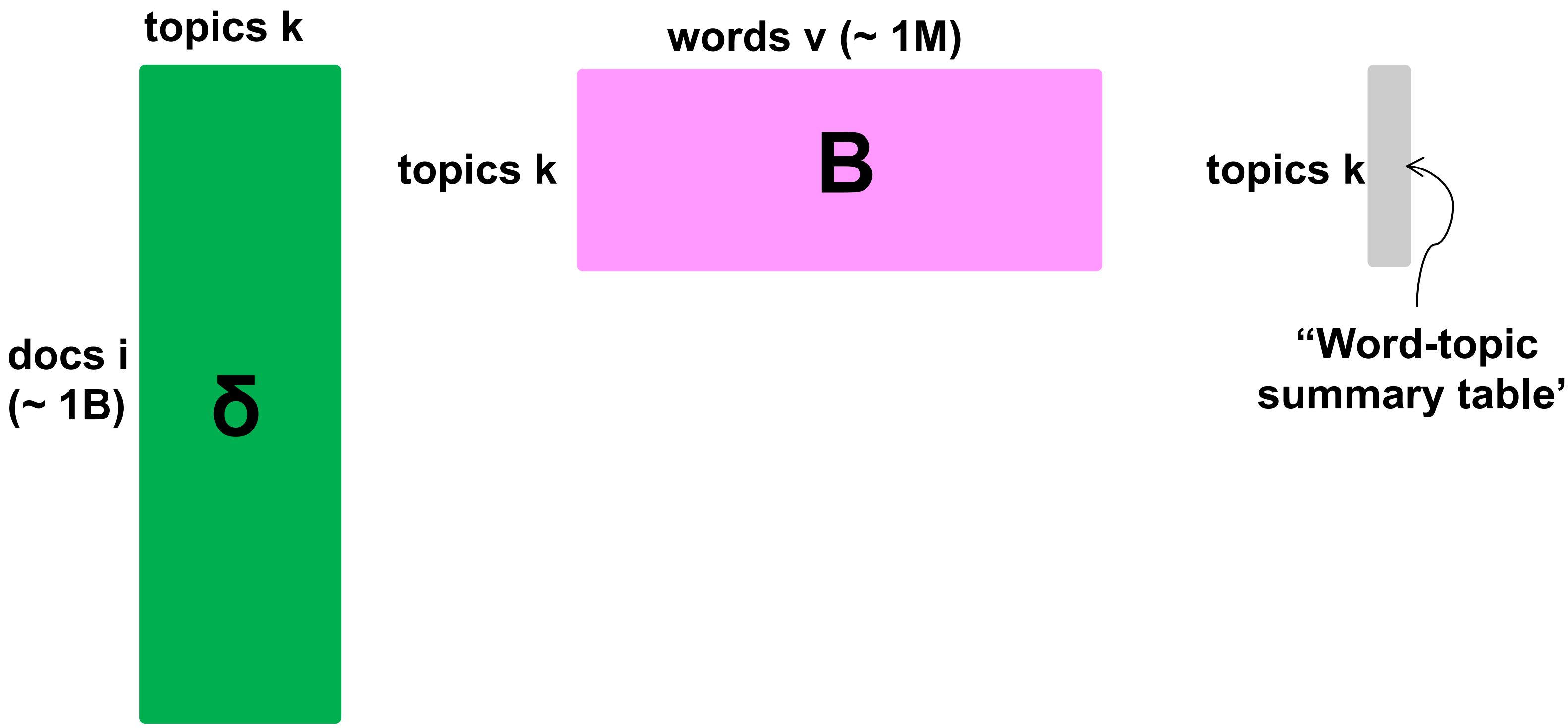
Perform updates to B, δ:

$$B_{k_{old}, w_{ij}} = B_{k_{old}, w_{ij}} - 1$$
$$B_{k_{new}, w_{ij}} = B_{k_{new}, w_{ij}} + 1$$
$$\delta_{i, k_{old}} = \delta_{i, k_{old}} - 1$$
$$\delta_{i, k_{new}} = \delta_{i, k_{new}} + 1$$

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

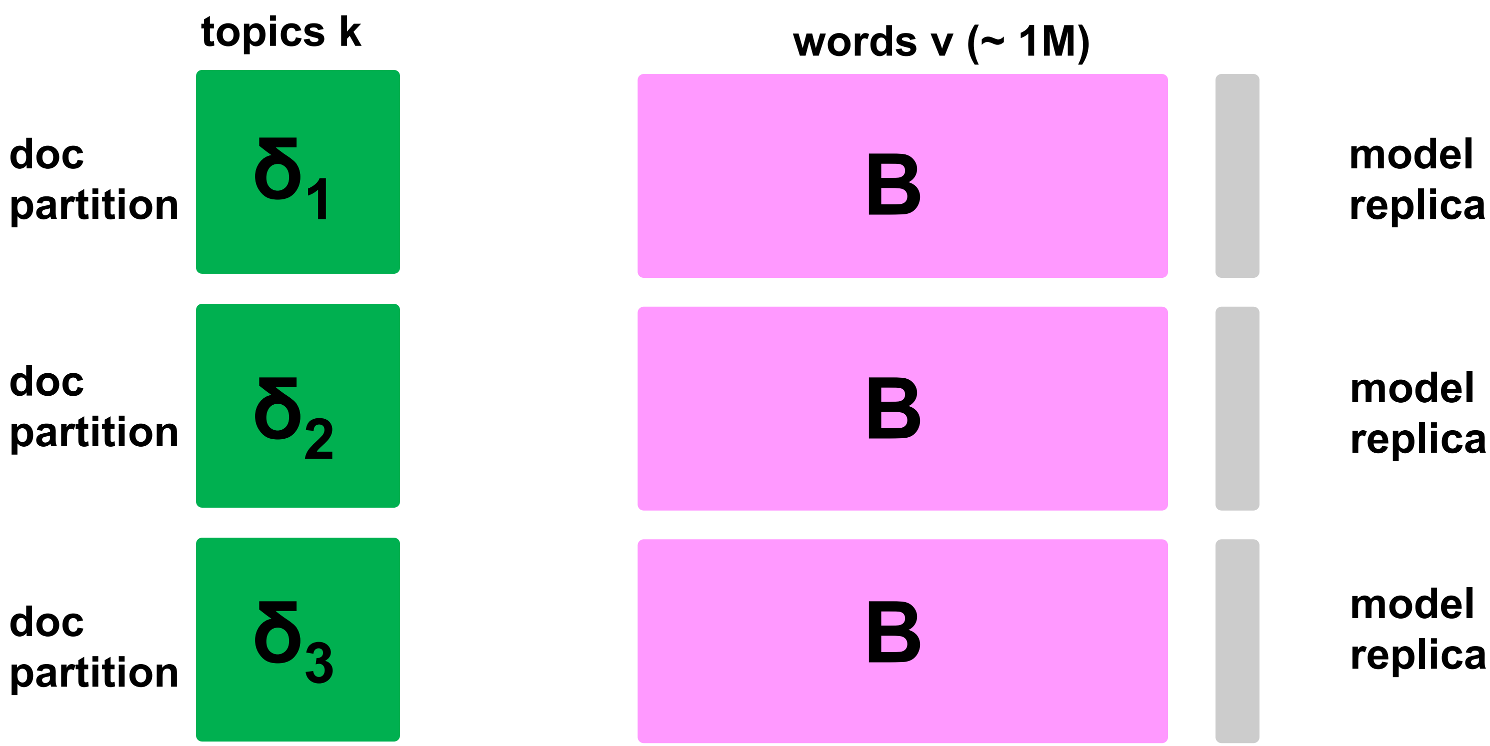
CGS Example: Topic Model sampler

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



Data Parallelization for CGS Topic Model Sampler

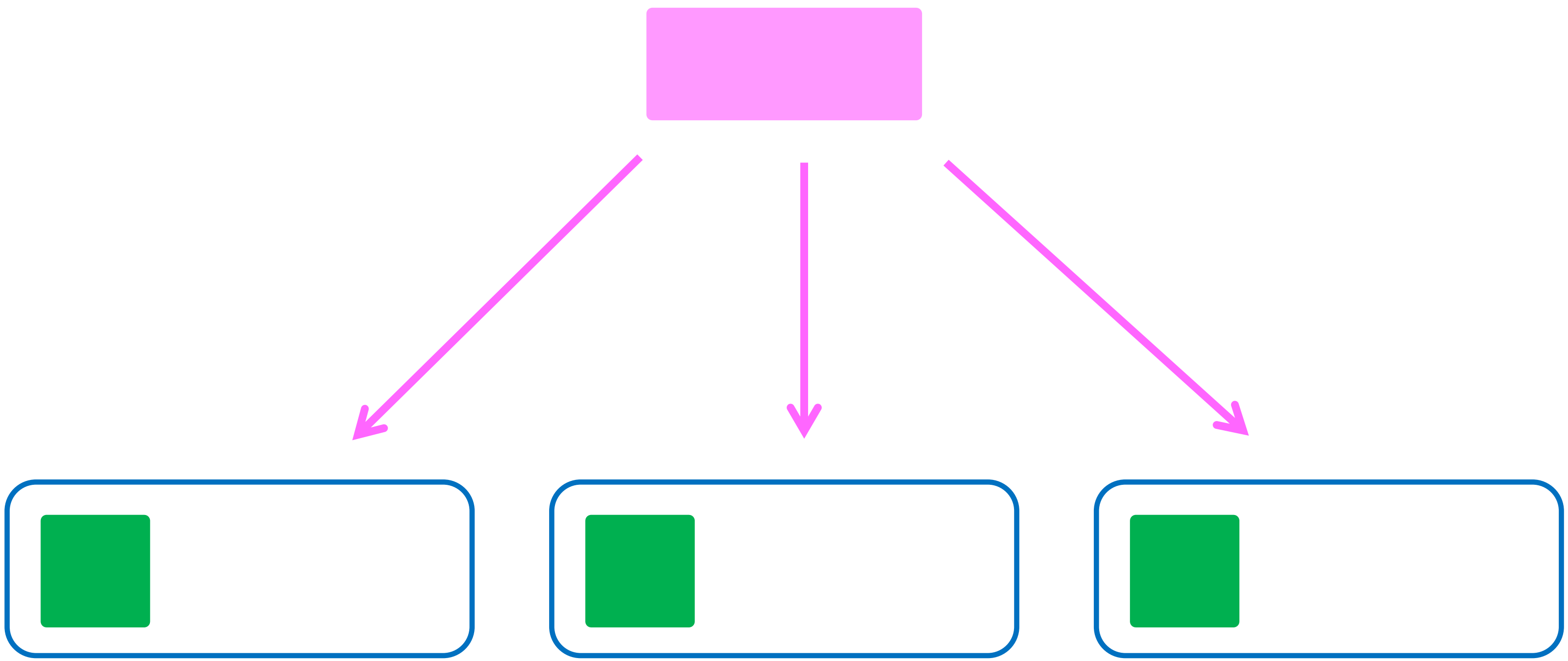
$$p(z_{ij} = k | x_{ij}, \delta_i, B) \propto (\delta_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + B_{k,x_{ij}}}{V\beta + \sum_{v=1}^{V} B_{k,v}}$$



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

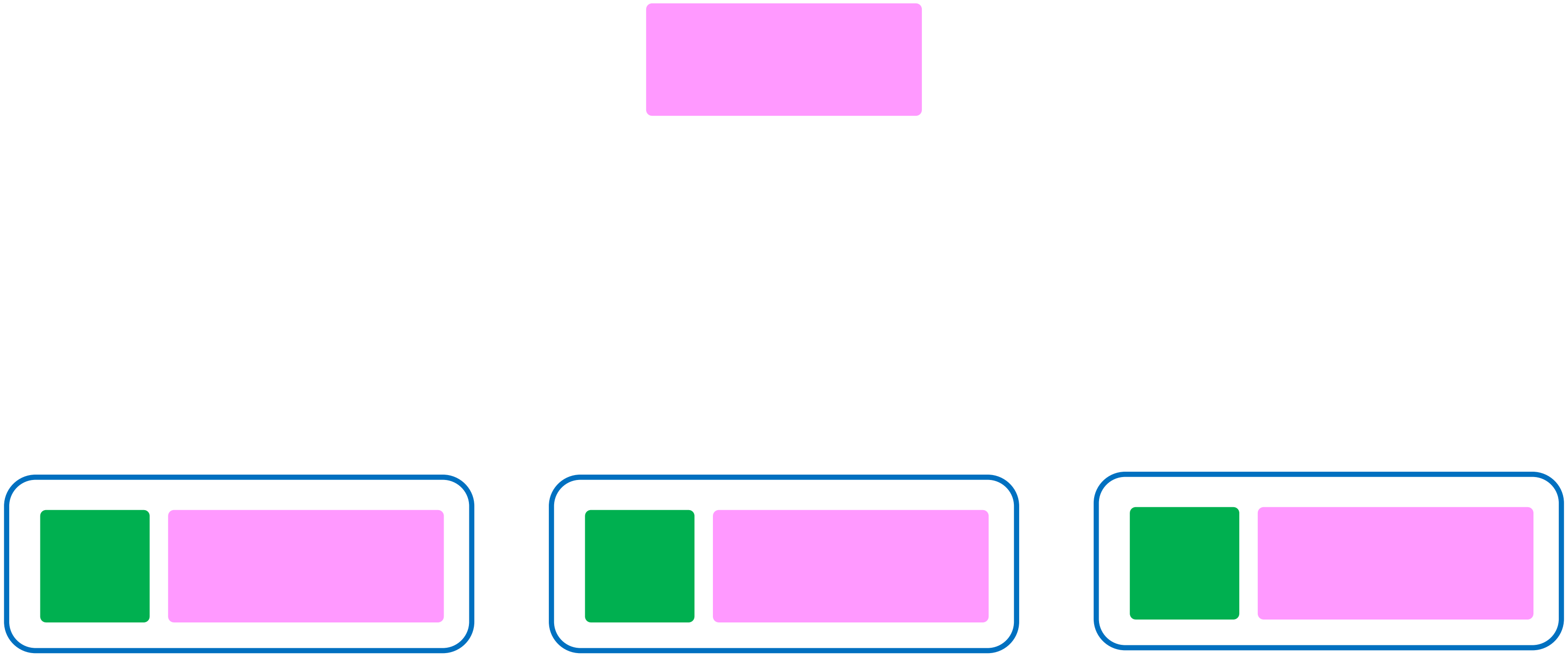
- Step 1: broadcast central model



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

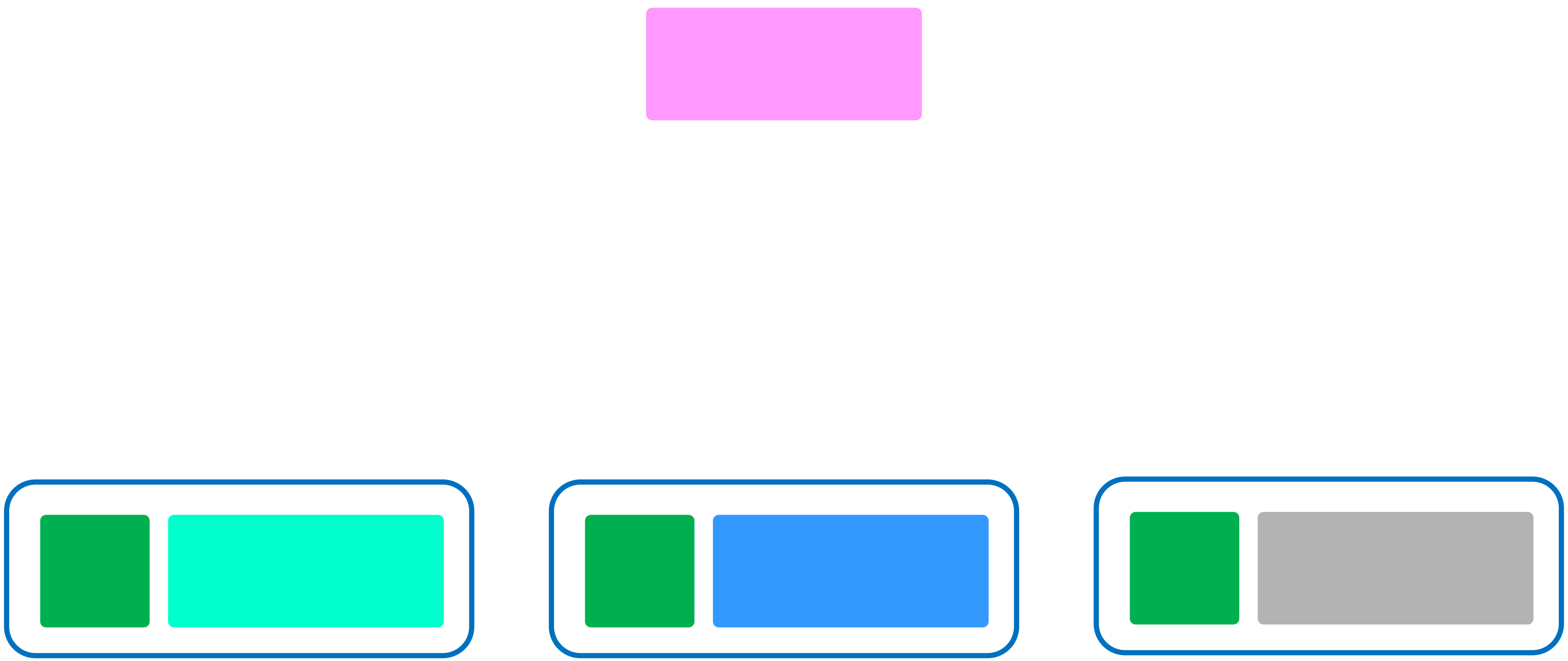
- Step 1: broadcast central model



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

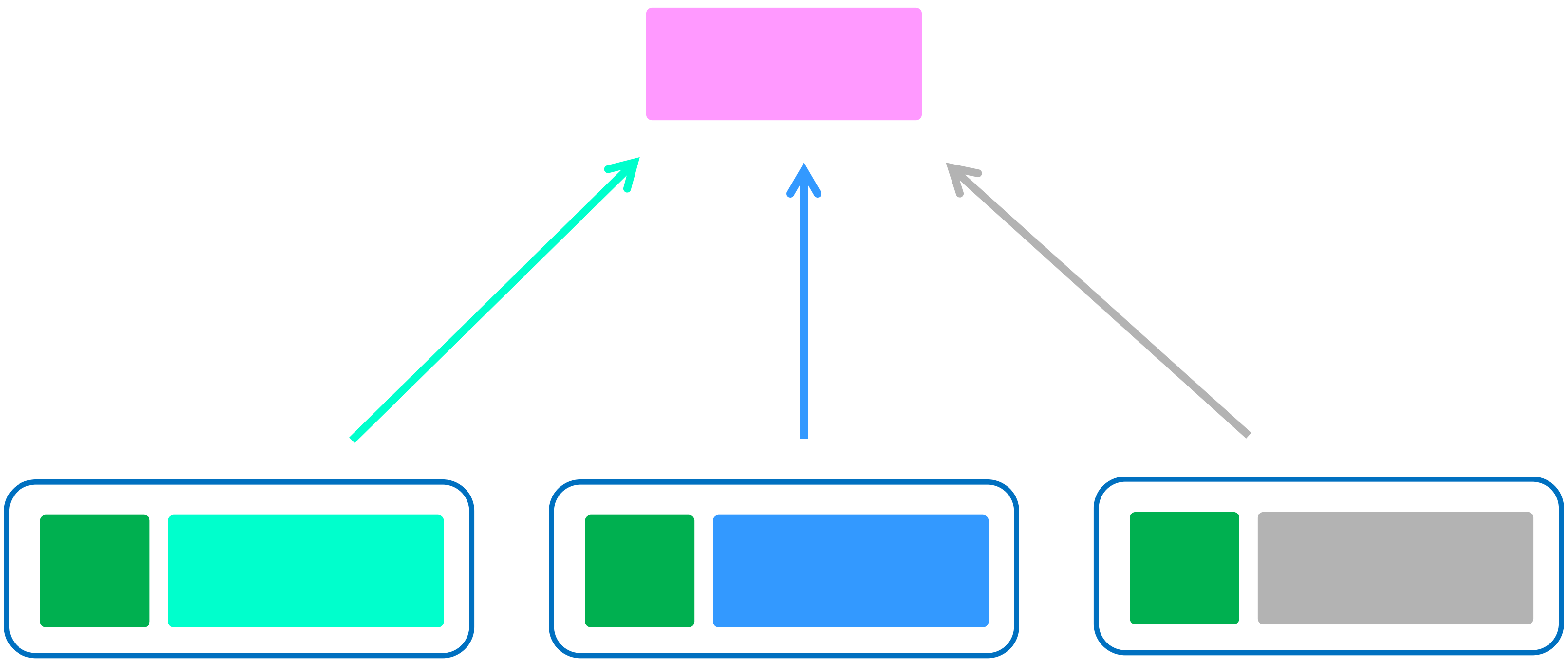
- Step 2: Perform Gibbs sampling in parallel



Data-Parallel Strategy: Approx. Distributed LDA

[Newman et al., 2009]

- Step 3: commit changes back to the central model



Error in data-parallel LDA

- Consider the CGS equation:

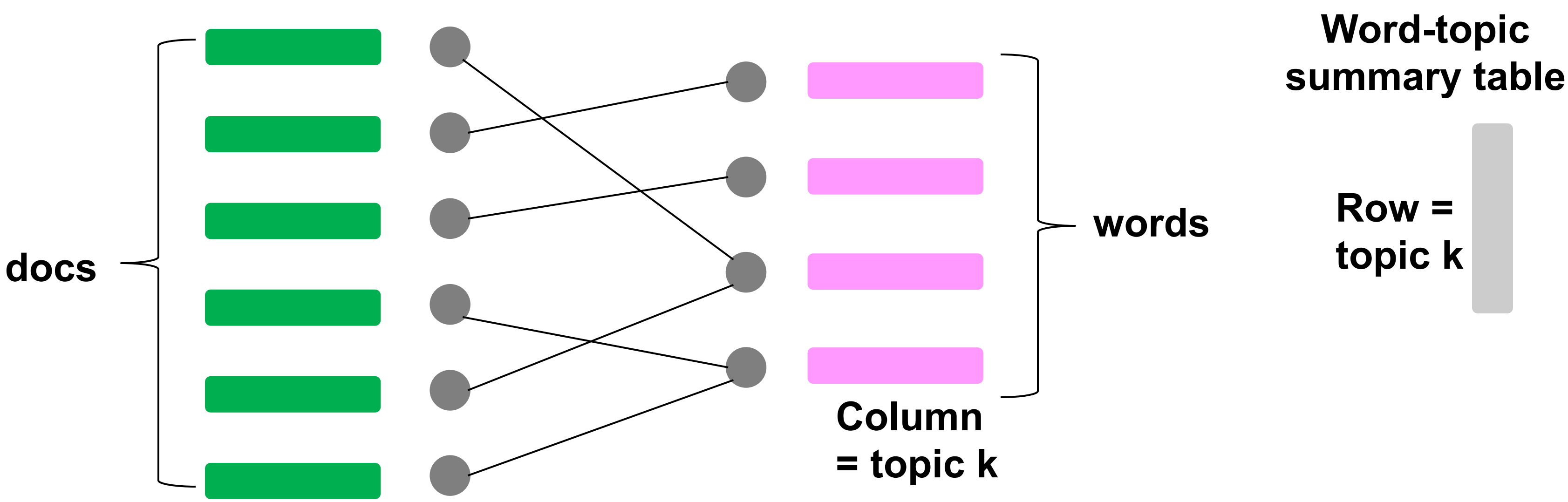
$$p(z_{ij} = k | x_{ij}, \delta_i, B) \propto (\delta_{ik} + \alpha_k) \cdot \frac{\beta_{x_{ij}} + B_{k,x_{ij}}}{V\beta + \sum_{v=1}^{V} B_{k,v}}$$

- 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]

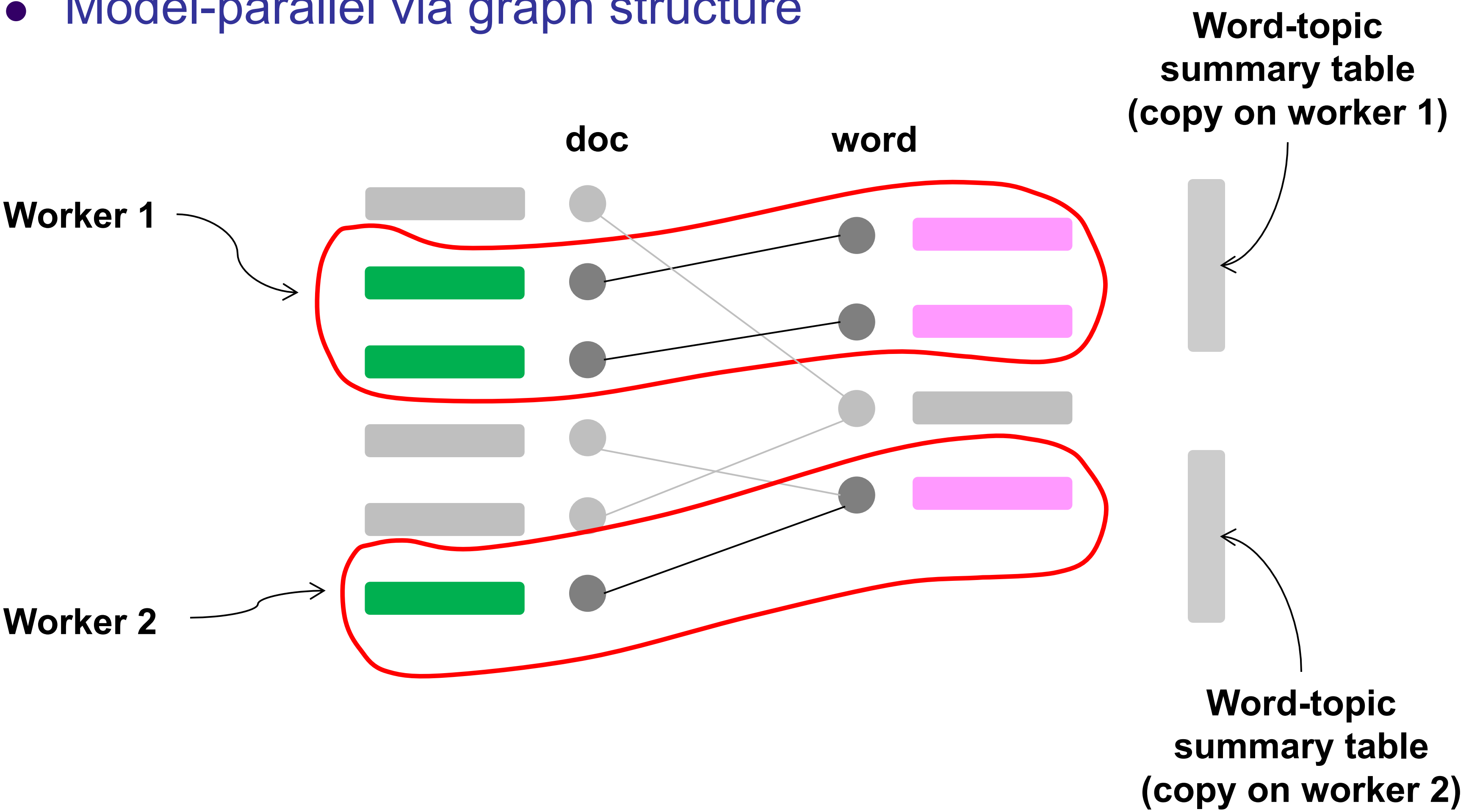
- Think graphically: token = edge

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



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

- Model-parallel via graph structure



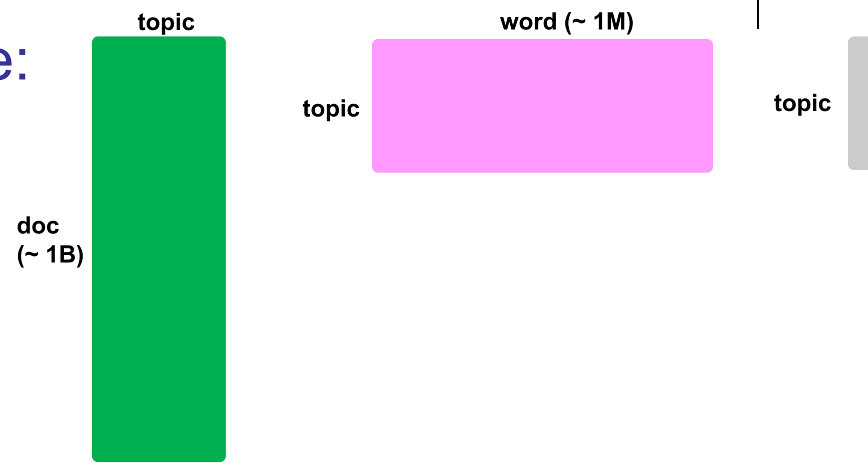
Model-Parallel Strategy 1: GraphLab LDA [Low et al., 2010; Gonzalez et al., 2012]

- 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?

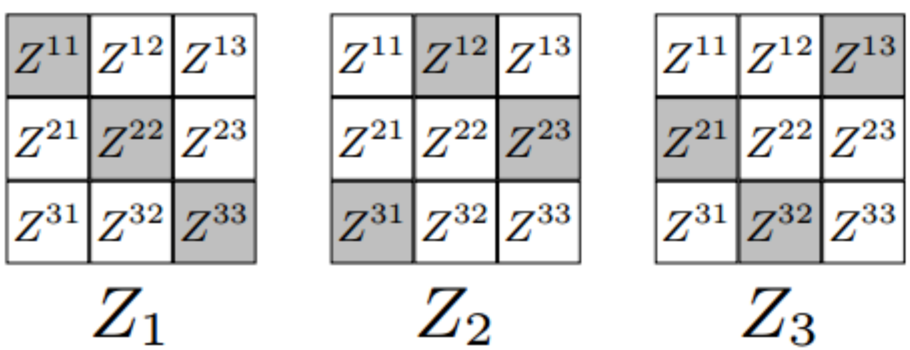
Model-Parallel Strategy 2: LightLDA (Petuum LDA v2)

[Yuan et al., 2015]

- Topic model matrix structure:



- Idea: non-overlapping matrix partition:

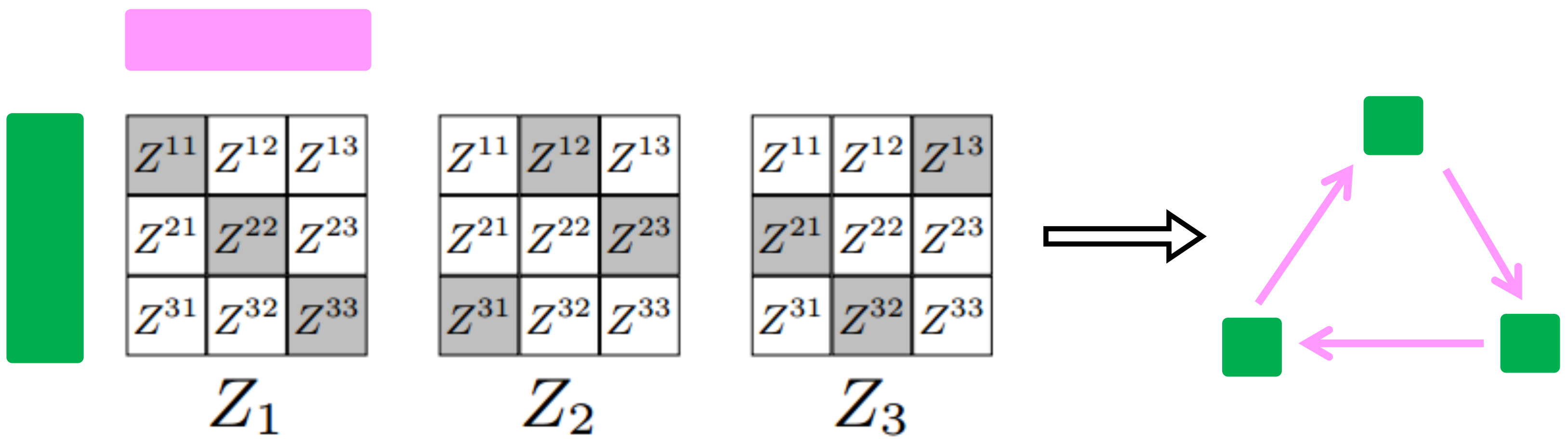


Source: [Gemulla et al., 2011]

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

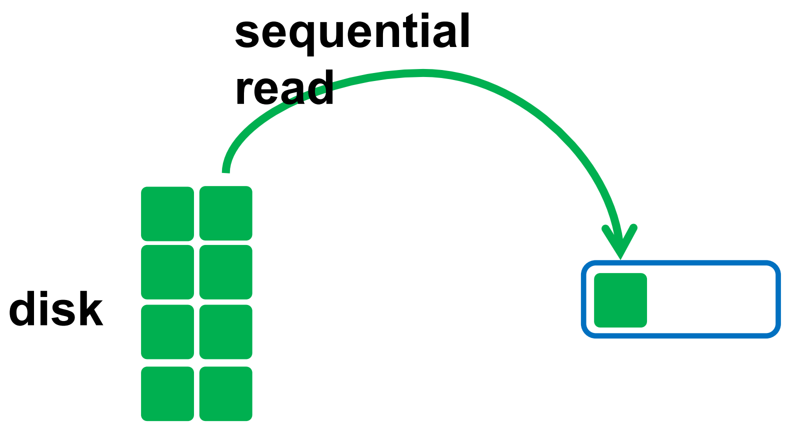


Source: [Gemulla et al., 2011]

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

[Yuan et al., 2015]

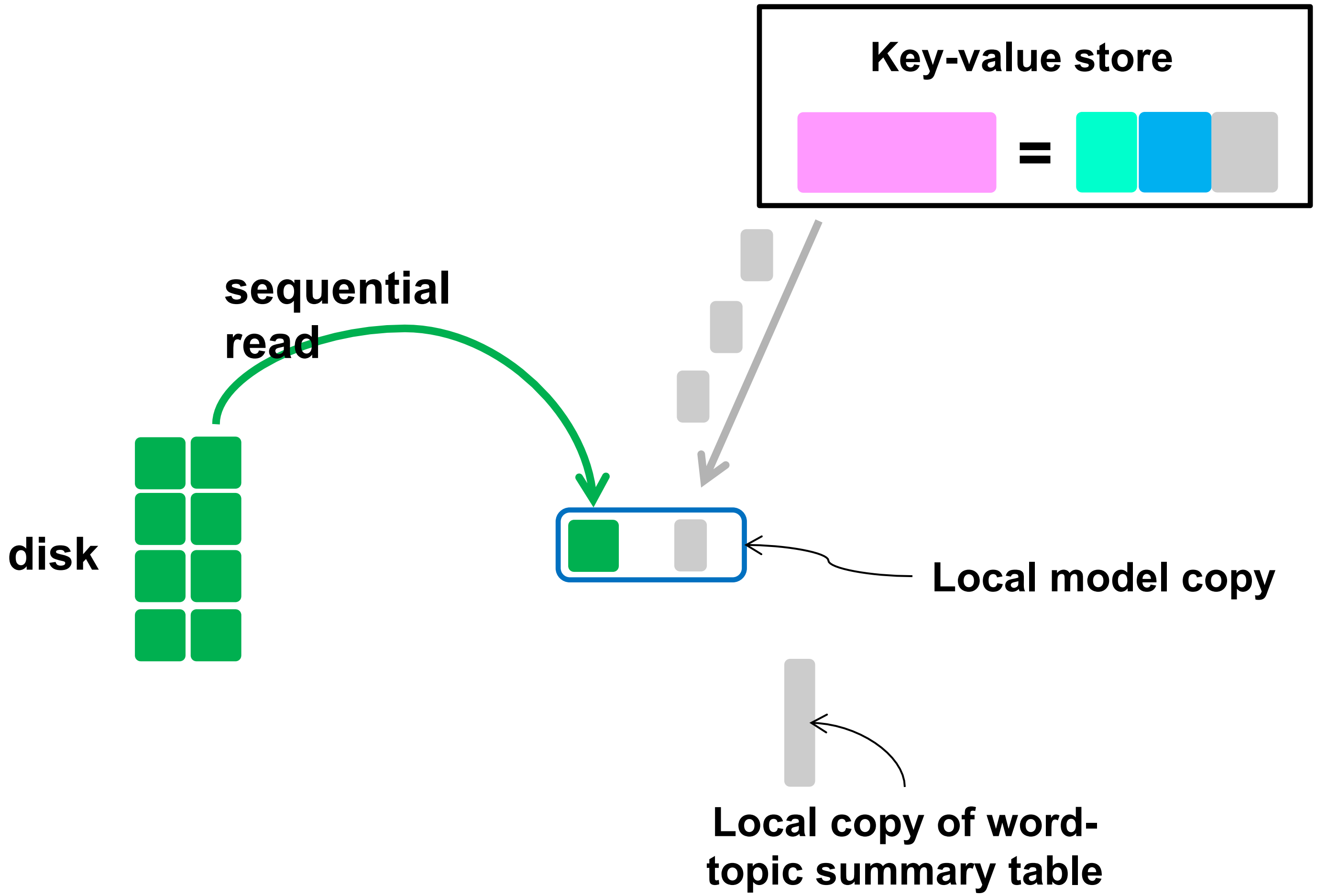
- During preprocessing: determine set of words used in each data block
- Begin training: load each data block from disk



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

[Yuan et al., 2015]

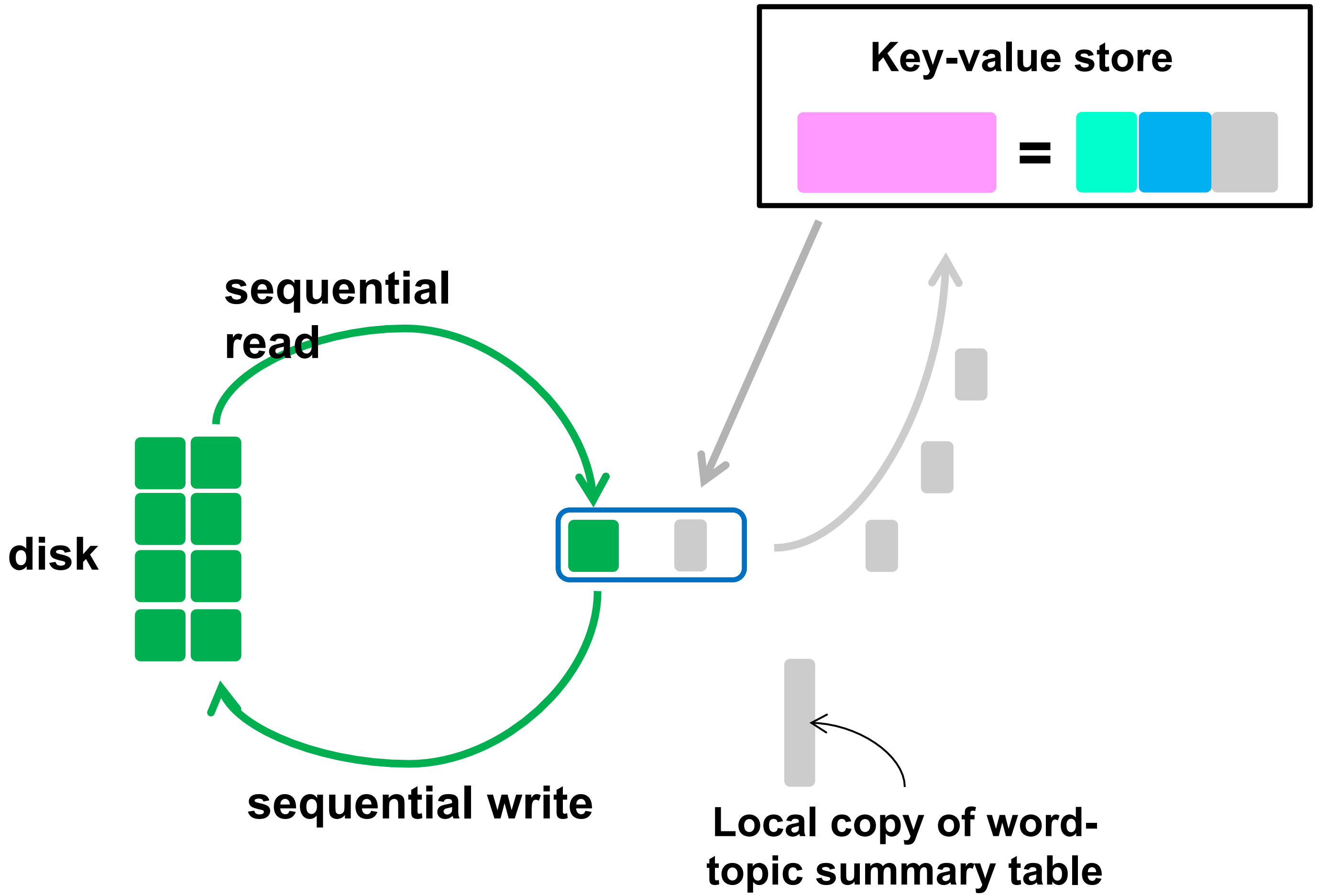
- Pull the set of words from Key-Value store



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:

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

- 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

Summary

- Most ML programs are either optimization or probabilistic programs
  - Optimization programs: SGD, ProxSGD, Coordinate Descent. Example: Lasso
  - Probabilistic programs: Gibbs sampling. Example: Topic model (LDA)
- Key considerations
  - Network delay: how to control error arising from delays?
  - How to partition the problem?
- Two ways to divide ML programs:
  - Data Parallel. Suitable if model parameters can be shared by all workers.
  - Model Parallel: Need to be careful in splitting model (e.g., pick dimensions with low correlations)