



Laboratory for Statistical Artificial Intelligence & Integrative Genomics

How to Go Really Big in AI: Strategies & Principles for Distributed Machine Learning

Eric Xing

epxing@cs.cmu.edu

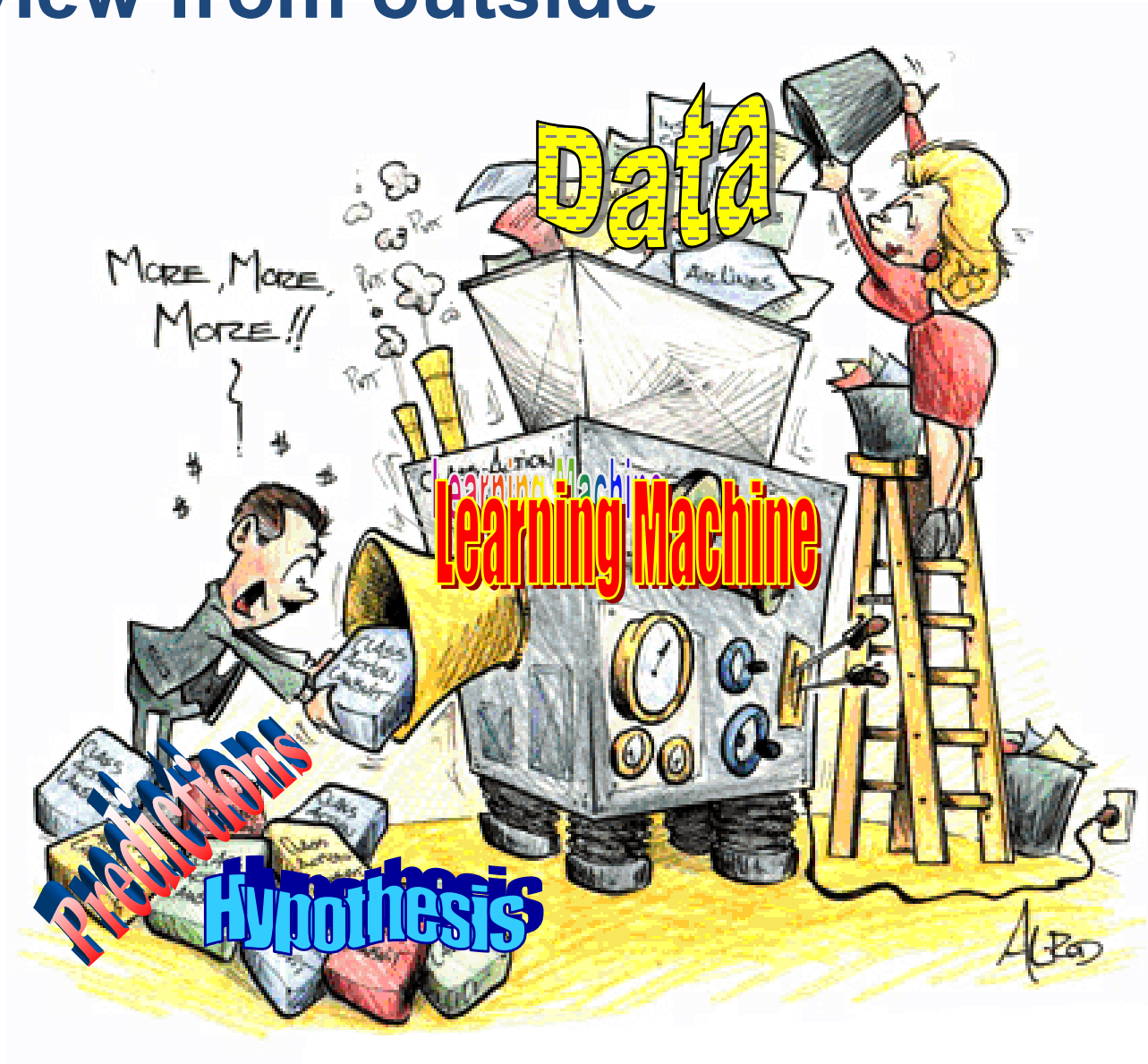
School of Computer Science
Carnegie Mellon University

Acknowledgement:

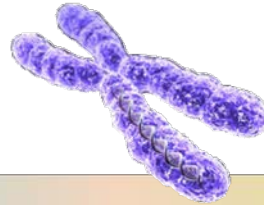
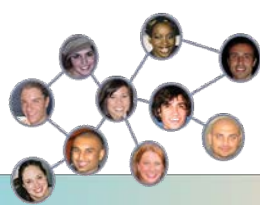
Wei Dai, Qirong Ho, Jin Kyu Kim, Abhimanu Kumar, Seunghak Lee, Jinliang Wei, Pengtao Xie, Yaoliang Yu, Hao Zhang, Xun Zheng
James Cipar, Henggang Cui,
and, Phil Gibbons, Greg Ganger, Garth Gibson



Machine Learning: -- a view from outside



Inside ML ...



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

```

C:\WINDOWS\system32\cmd.exe
C:\>nbtstat
Displays protocol statistics and current TCP/IP connections using NBT
(NetBIOS over TCP/IP).
NBTSTAT [ [-a RemoteName] [-A IP address] [-c] [-n]
          [-r] [-R] [-RR] [-s] [-S] [interval] ]

-a <adapter status> Lists the remote machine's name table given its name
-A <Adapter status> Lists the remote machine's name table given its
                    IP address.
-c <cache> Lists NBT's cache of remote [machine] names and their
-n <names> Lists local NetBIOS names.
-r <resolved> Lists names resolved by broadcast and via WINS
-R <Reload> Purges and reloads the remote cache name table
-S <Sessions> Lists sessions table with the destination IP address
-s <sessions> Lists sessions table converting destination IP
                    addresses to computer NETBIOS names.
-RR <ReleaseRefresh> Sends Name Release packets to WINS and then, starts

RemoteName Remote host machine name.
IP address Dotted decimal representation of the IP address.
interval Redisplays selected statistics, pausing interval seconds
            between each display. Press Ctrl+C to stop redisplaying
            statistics.
  
```

Hardware and infrastructure

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

Massive Data



facebook®

1B+ USERS

30+ PETABYTES



32 million
pages

WIKIPEDIA
The Free Encyclopedia



You Tube

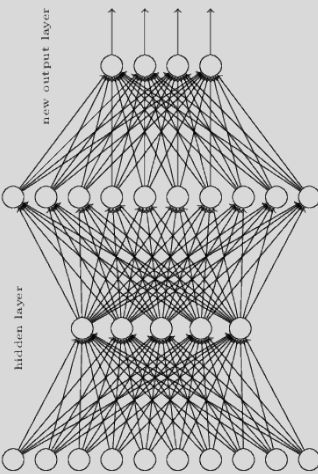
100+ hours video
uploaded every minute



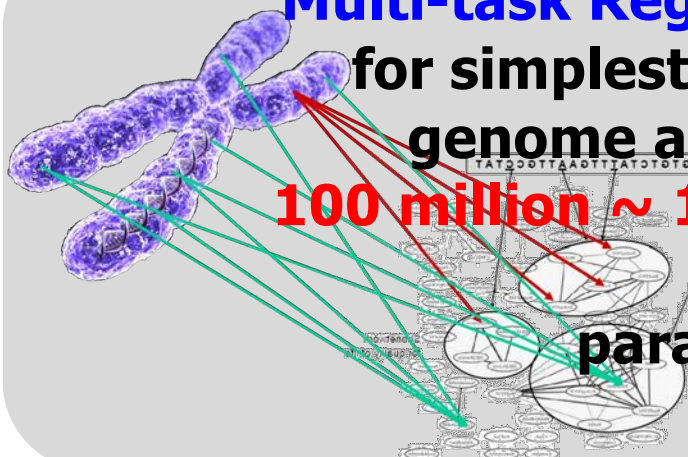
twitter 

645 million users
500 million tweets / day

Growing Model Complexity



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

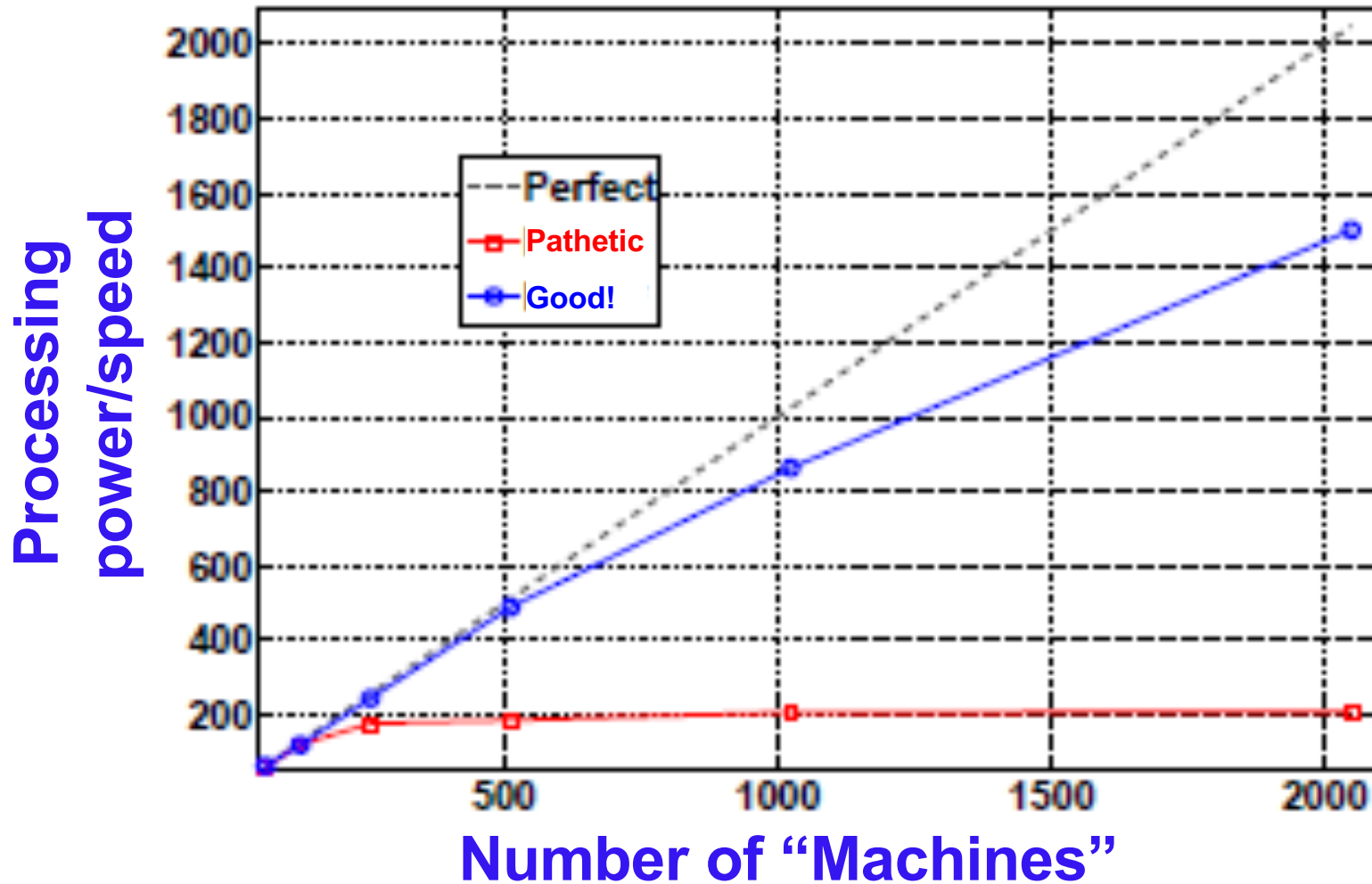


Topic Models for news article analysis:
Up to 1 Trillion model parameters



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

The Scalability Challenge



Why need new Big ML systems?

Today's AI & ML imposes high CAPEX and OPEX

- Example: The **Google Brain** AI & ML system
- **High CAPEX**
 - 1000 machines
 - \$10m+ capital cost (hardware)
 - \$500k+/yr electricity and other costs
- **High OPEX**
 - 3 key scientists (\$1m/year)
 - 10+ engineers (\$2.5m/year)
- **Total 3yr-cost = \$20m+**
- **Small to mid companies and the Academic do not have such luxury**
- **1000 machines only 100x as good as 1 machine!**



Why need some new thinking?

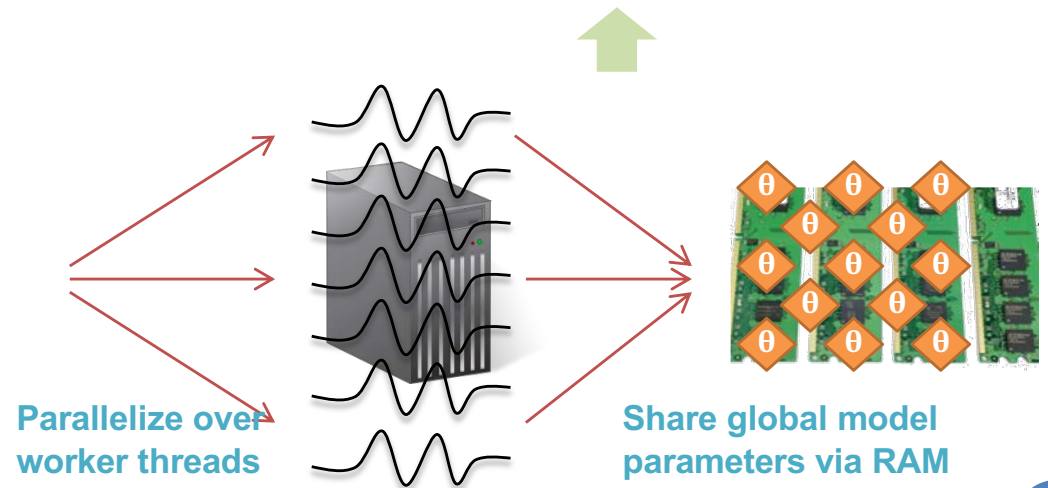
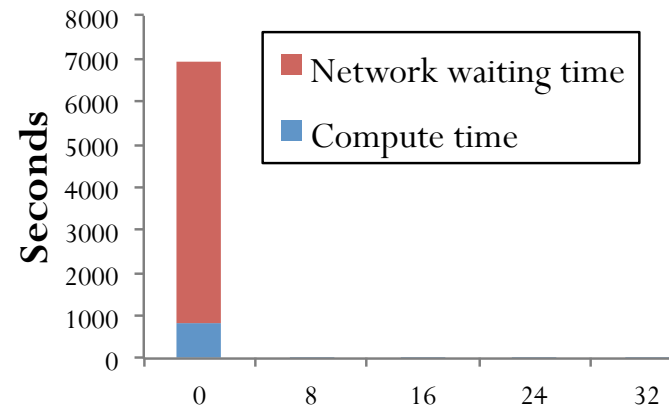
Mler's view

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

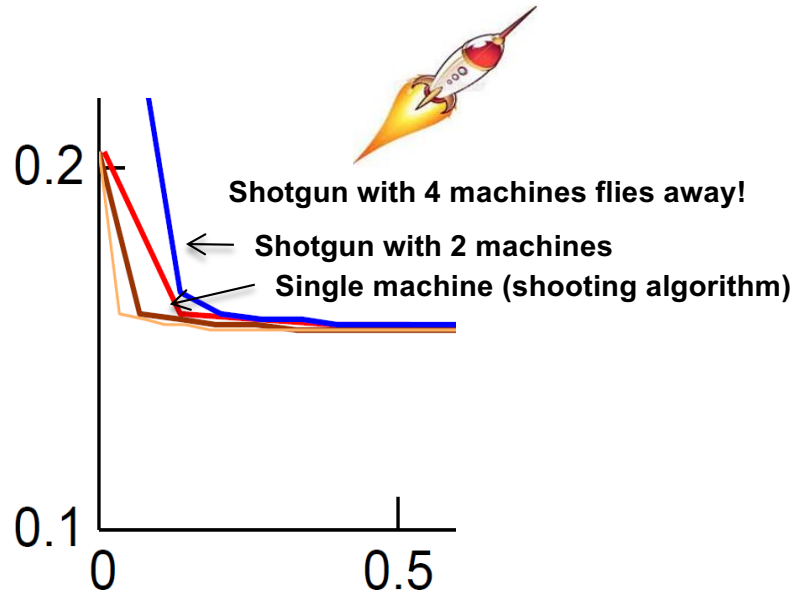
```

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

Compute vs Network
 LDA 32 machines (256 cores)



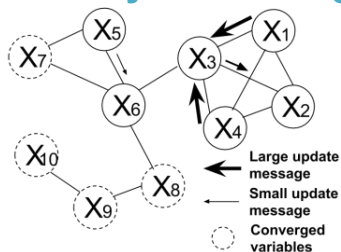
Why need some new thinking?



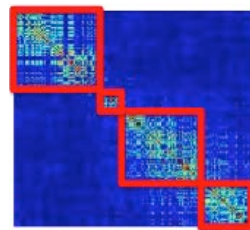
Systems View:

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

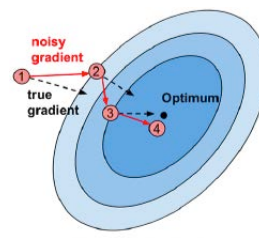
Agonistic of ML properties and objectives in system design



Non-uniform convergence

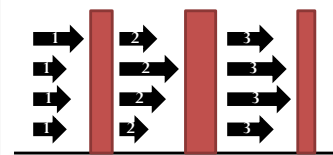


Dynamic structures

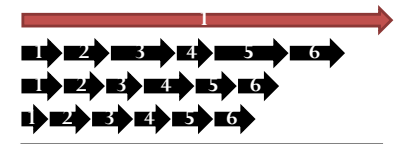


Error tolerance

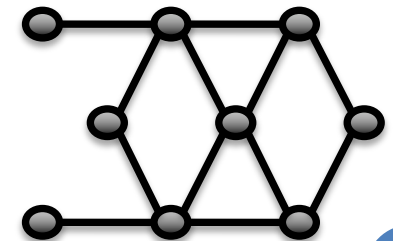
Synchronization model



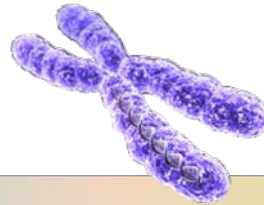
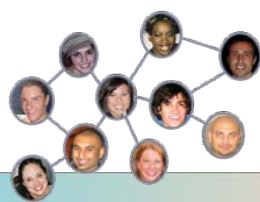
or



Programming model



Existing Solution:



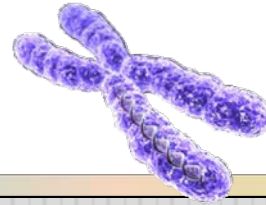
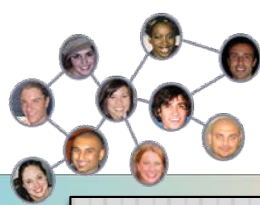
Machine Learning Models/Algorithms

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

Hardware and infrastructure

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

How about this ... [Xing et al., 2015]



Machine Learning Models/Algorithms

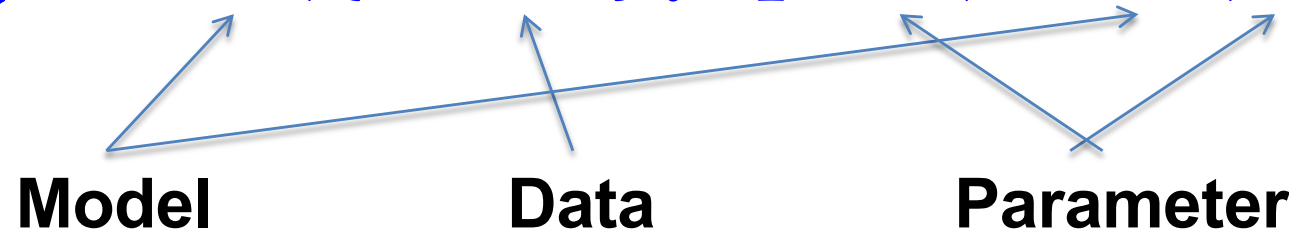
- Graphical Models
- Nonparametric Bayesian Models
- Regularized Bayesian Methods
- Large-Margin
- Deep Learning
- Sparse Coding
- Spectral/Matrix Methods
- Sparse Structured I/O Regression



Hardware and infrastructure

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

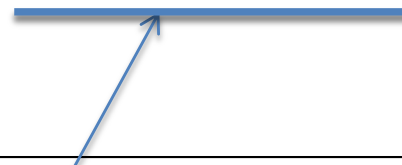
An ML Program

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


Model **Data** **Parameter**

Solved by an iterative convergent algorithm

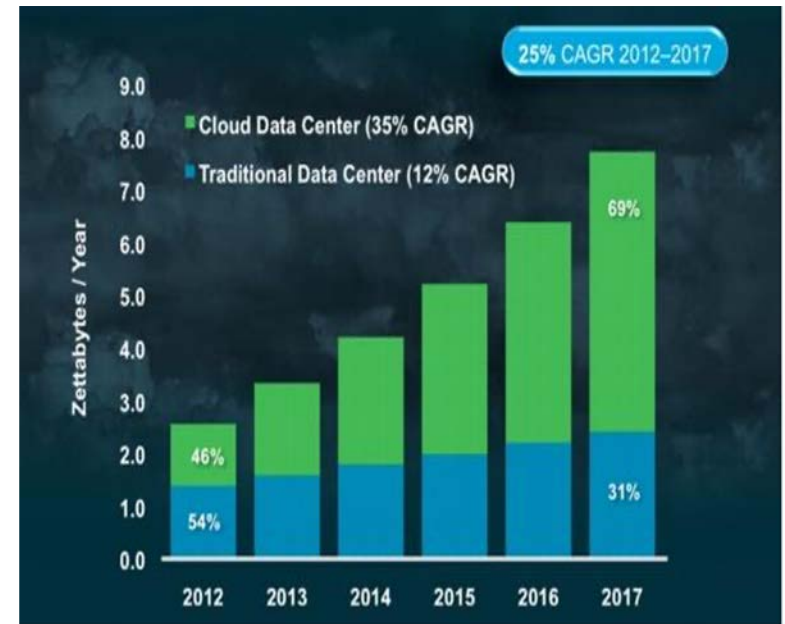
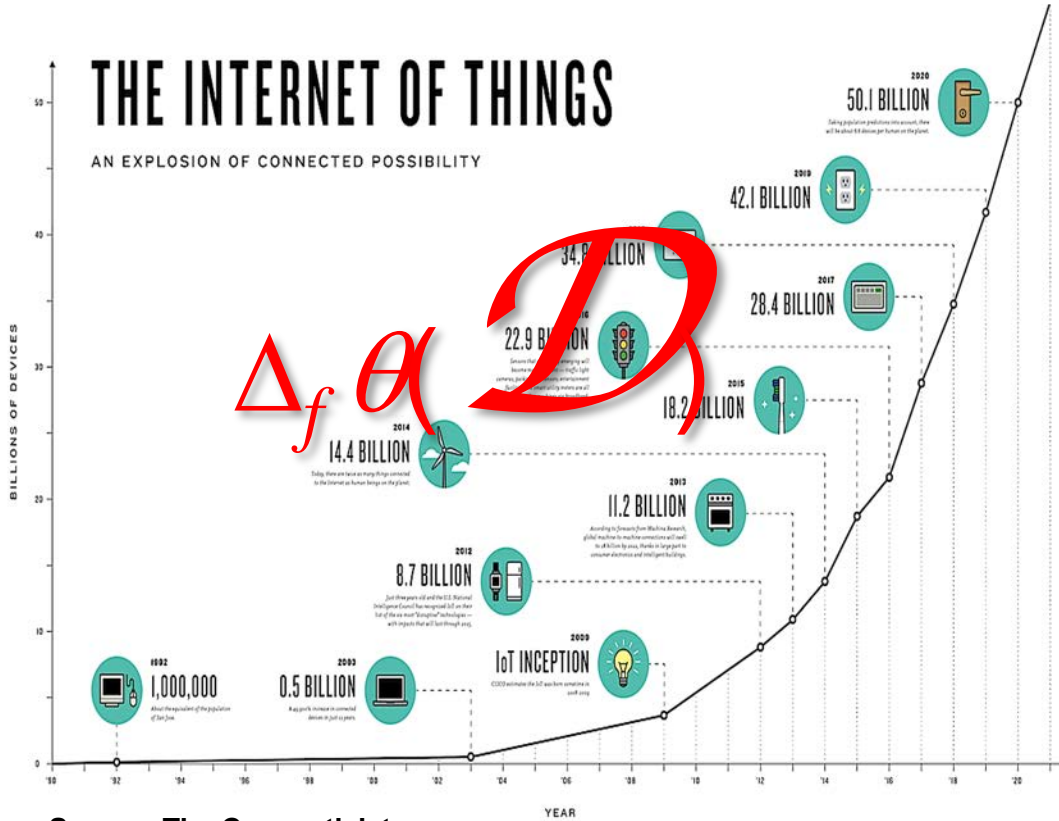
```
for (t = 1 to T) {  
  doThings()  
   $\vec{\theta}^{t+1} = g(\vec{\theta}^t, \Delta_f \vec{\theta}(\mathcal{D}))$   
  doOtherThings()  
}
```



This computation needs to be parallelized!

Challenge #1

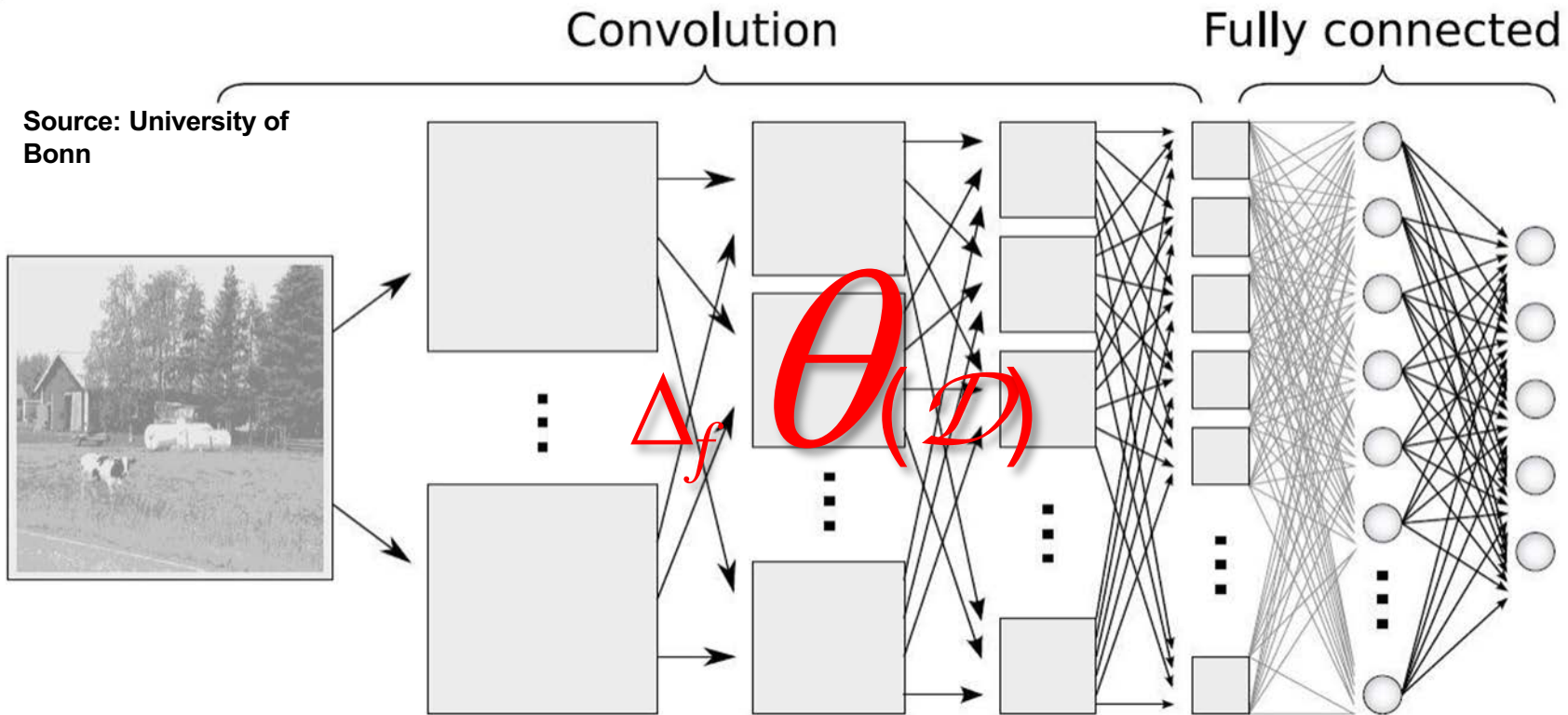
– Massive Data Scale



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

Challenge #2

– Gigantic Model Size

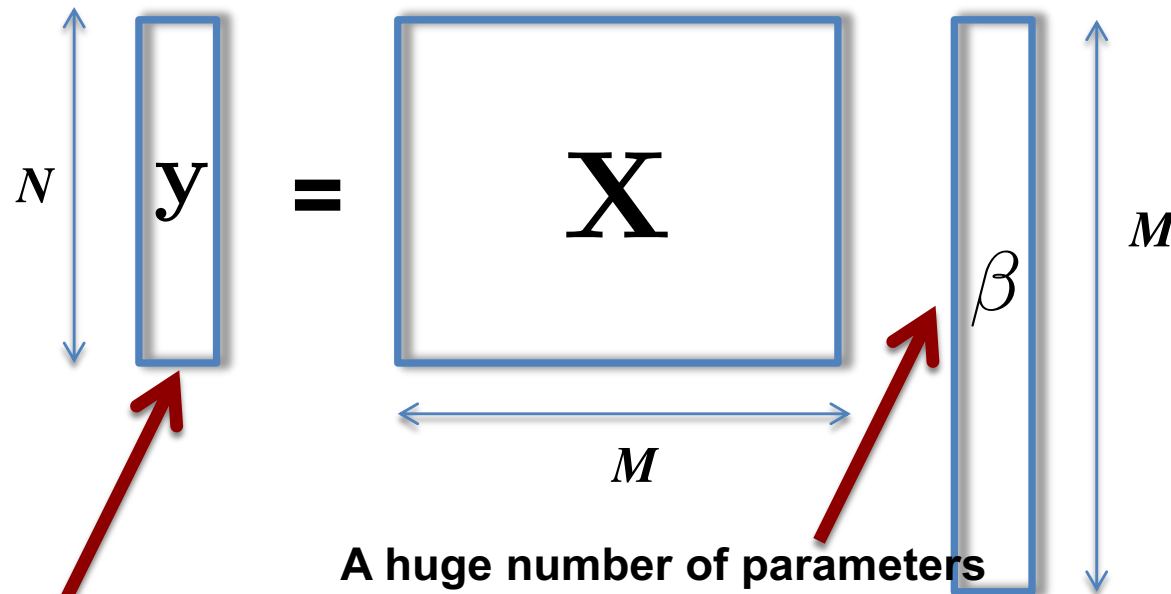


Big Data needs Big Models to extract understanding
But ML models with >1 trillion params also won't fit!

Typical ML Programs (about the “ f ”)

- Optimization programs:

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



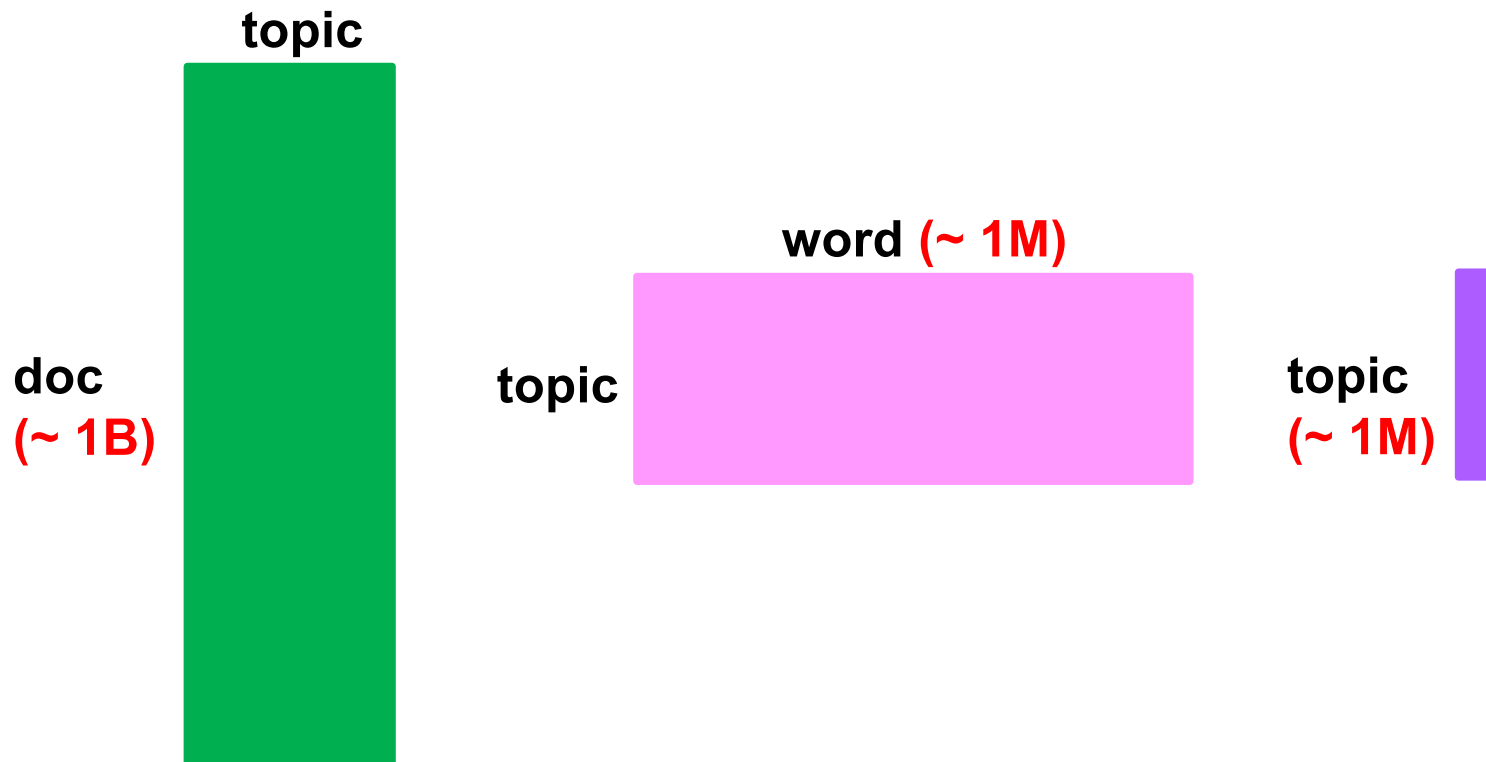
A huge volume of data
 (e.g.) $N = 1\text{B}$

A huge number of parameters
 (e.g.) $M = 1\text{B}$

Typical ML Programs (about the “f”)

- Probabilistic programs

$$z_{di} \sim p(z_{di} = k | \text{rest}) \propto (n_{kd}^{-di} + \alpha_k) \cdot \frac{(n_{kw}^{-di} + \beta_w)}{(n_k^{-di} + \bar{\beta}V)}$$



Algorithmic Accelerations:

- Optimization Algorithms
 - Stochastic gradient descent
 - Coordinate descent
 - Proximal gradient methods --- when \mathcal{L} is not differentiable
 - ISTA, FASTA, Smoothing proximal gradient
 - Proximal average --- complex compound regularizers
 - ADMM --- overlapping constraints
 - ...
- Markov Chain Monte Carlo Algorithms
 - Aliases samplers (constant time high-dimensional sampler)
 - Auxiliary variable methods (inverse Rao-Blackwellization)
 - Embarrassingly Parallel MCMC (sub-posteriors)
 - Parallel Gibbs Sampling
 - Data parallel
 - Model parallel

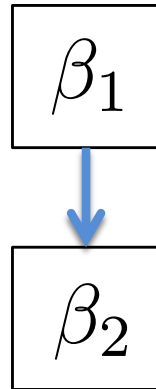
```

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

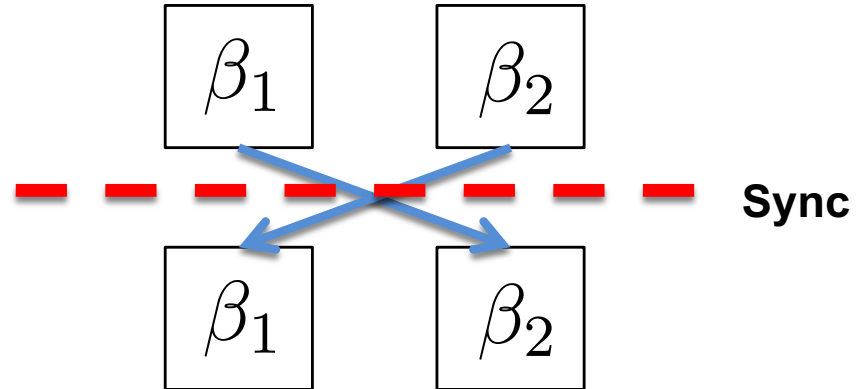
Parallelization Strategies

Usually, we worry ...

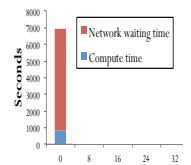
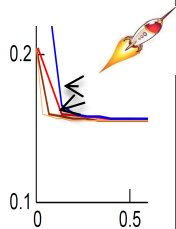
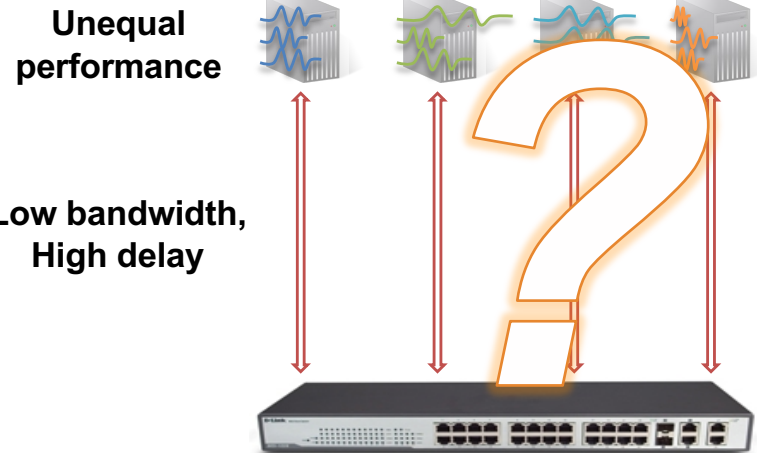
A sequential program



A parallel program



- but assuming an ideal system, e.g.,
 - zero-cost sync,
 - zero-cost fault recovery
 - uniform local progress
 - ...



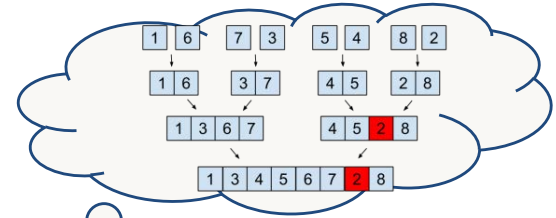
ML Computation vs. Classical Computing Programs



```

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

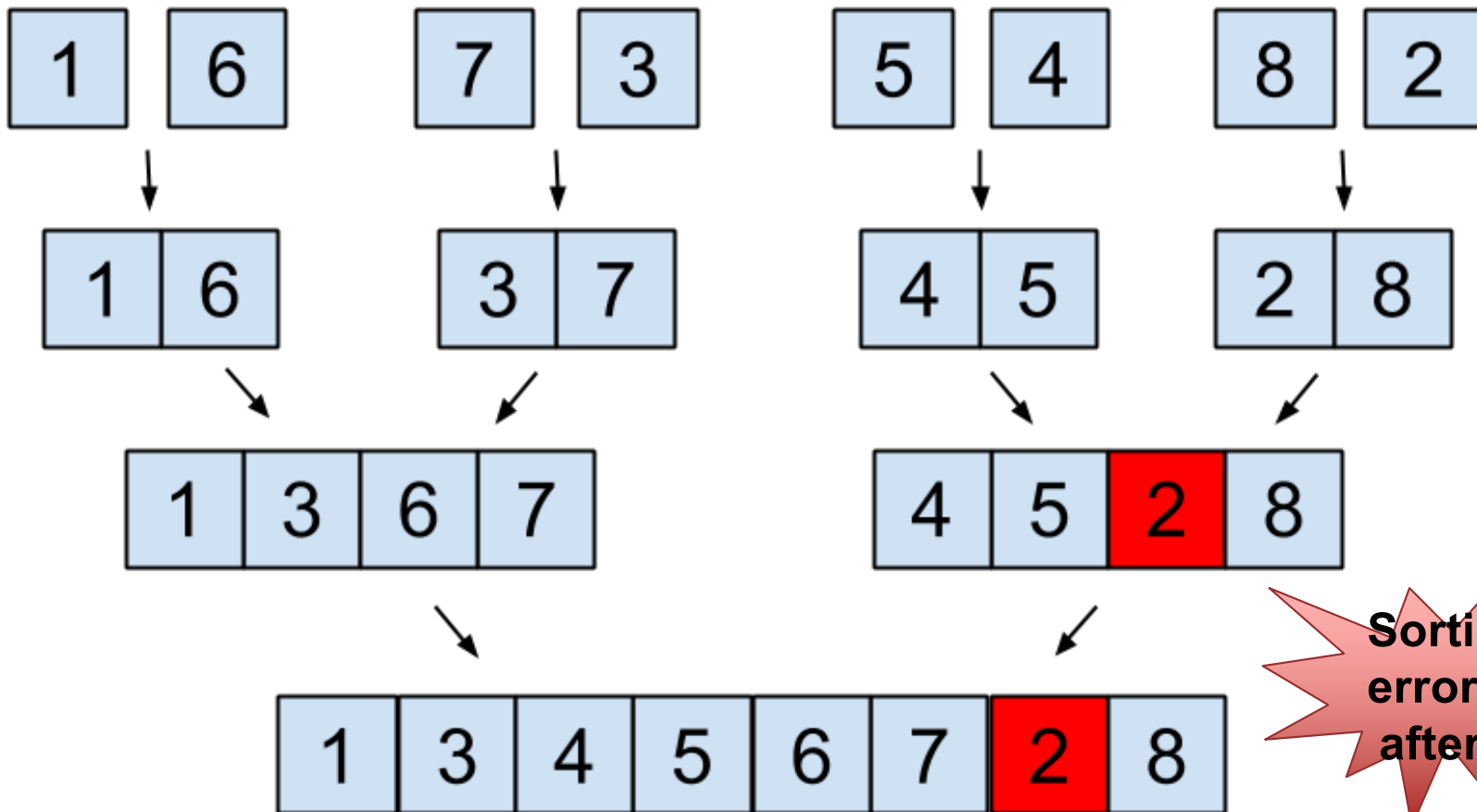
ML Program:
 optimization-centric and
 iterative convergent



Traditional Program:
 operation-centric and
 deterministic

Traditional Data Processing needs operational correctness

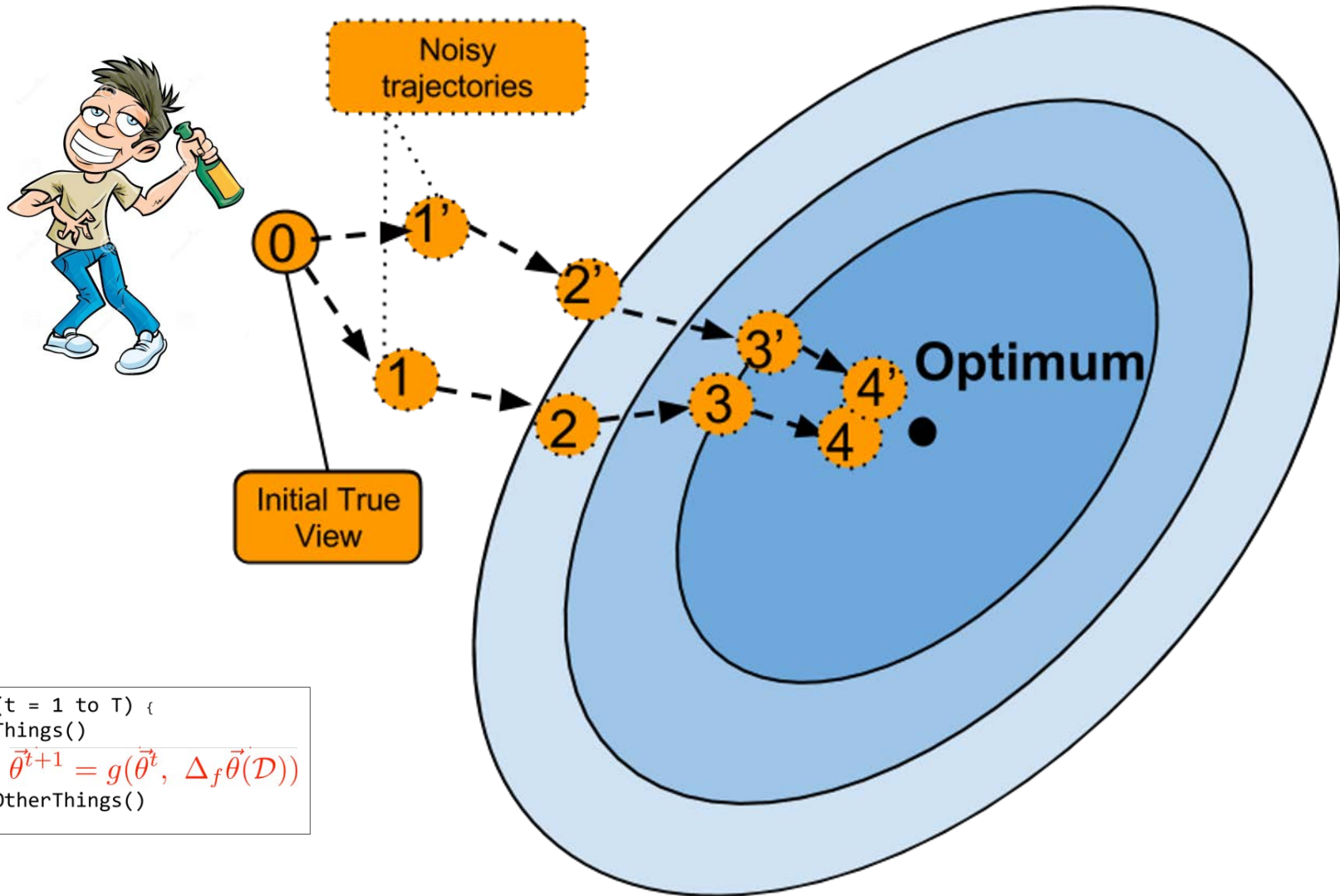
Example: Merge sort



Sorting error: 2 after 5

Error persists and is not corrected 20

ML Algorithms can Self-heal



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

Intrinsic Properties of ML Programs

[Xing et al., 2015]

- ML is algorithmic

- Error rate is low

- Dynamic characteristics

- Non-linear



gent
on



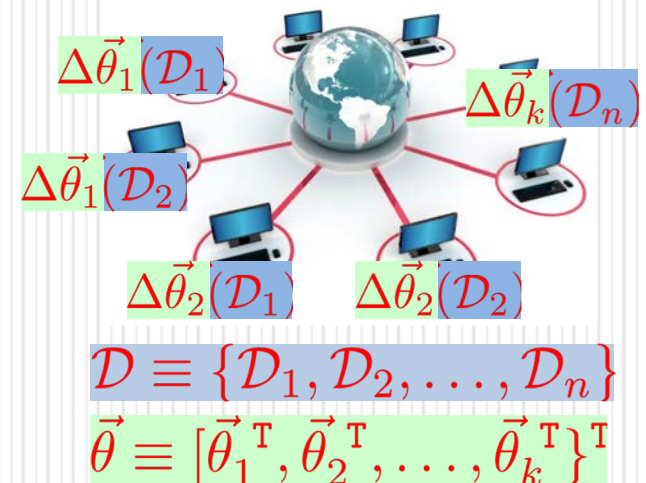
large upd:
message
small upd:
message
converge

- When guard

only

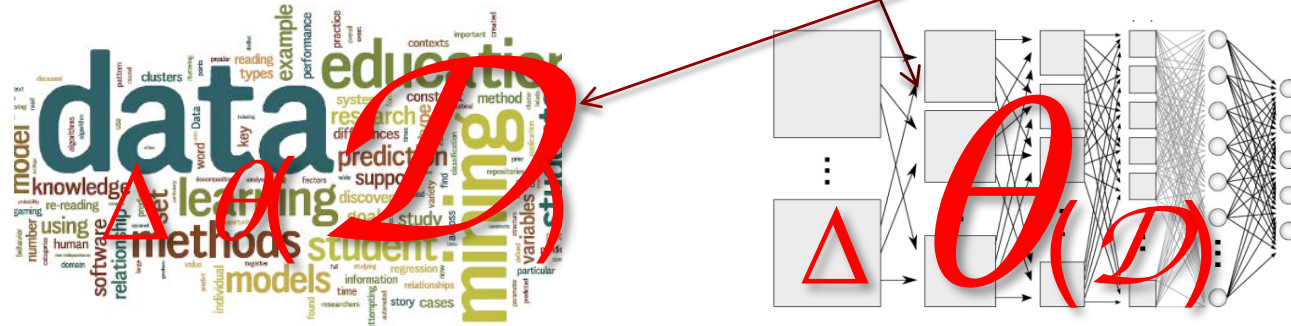
- How do existing Big Data platforms fit the above?

Two Parallel Strategies for ML



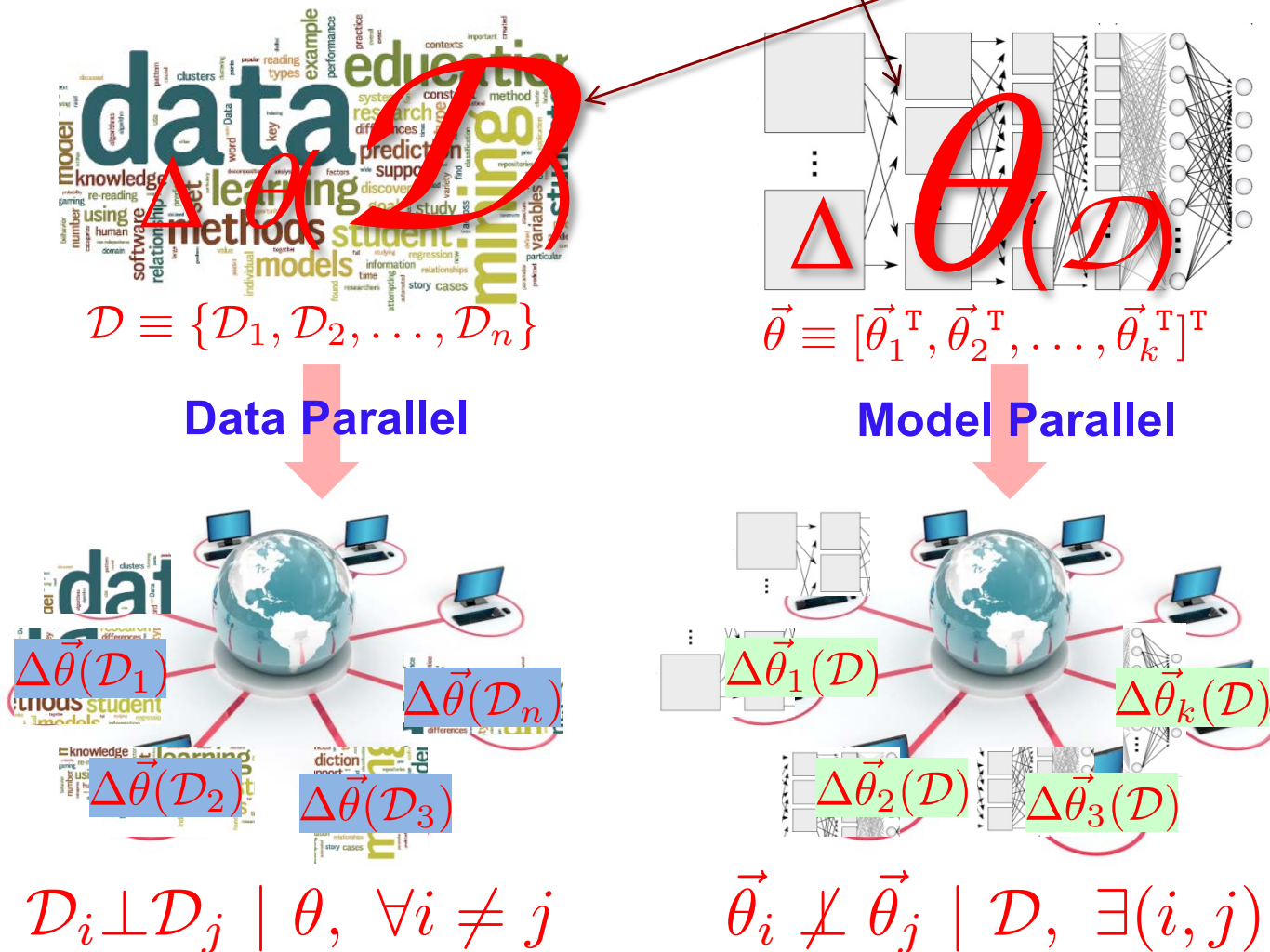
A Dichotomy of Data and Model in ML Programs

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



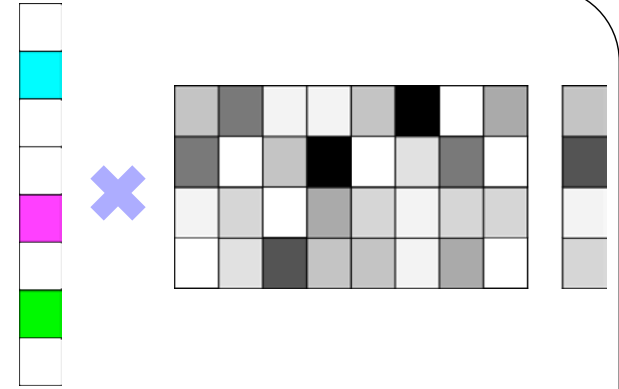
A Dichotomy of Data and Model in ML Programs

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



Optimization Example:

Lasso Regression



- Data, Model

- $D = \{\text{feature matrix } X, \text{ response vector } y\}$
- $\theta = \{\text{parameter vector } \beta\}$

- Objective $L(\theta, D)$

- Least-squares difference between y and $X\beta$: $\sum_{i=1}^N \|y_i - X_i\beta\|_2^2$

- Regularization $\Omega(\theta)$

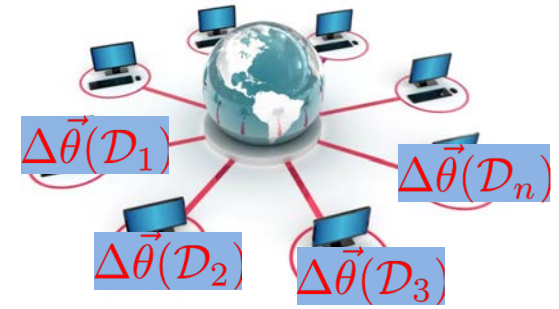
- L_1 penalty on β to encourage sparsity: $\lambda \sum_{j=1}^D |\beta_j|$
- λ is a tuning parameter

- Algorithms

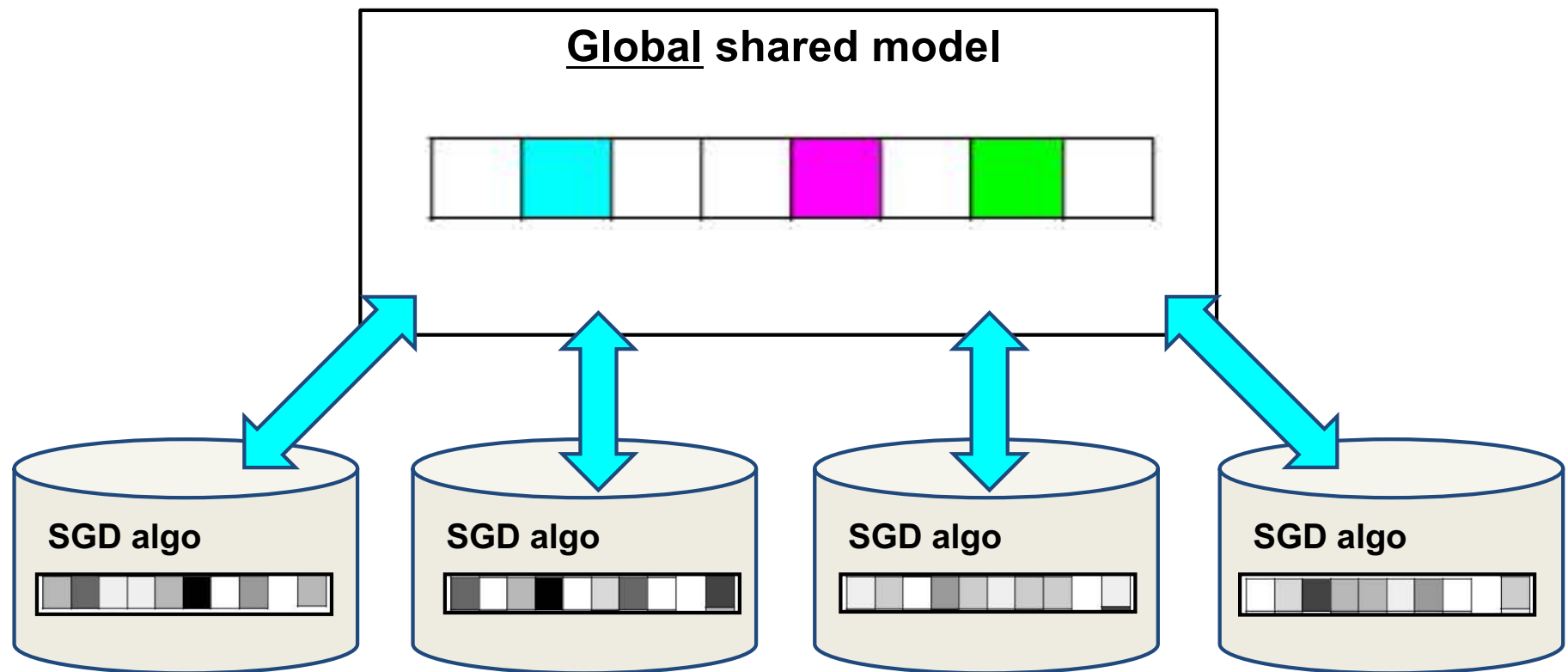
- Coordinate Descent
- Stochastic Proximal Gradient Descent

Data-Parallel Lasso

Proximal SGD:



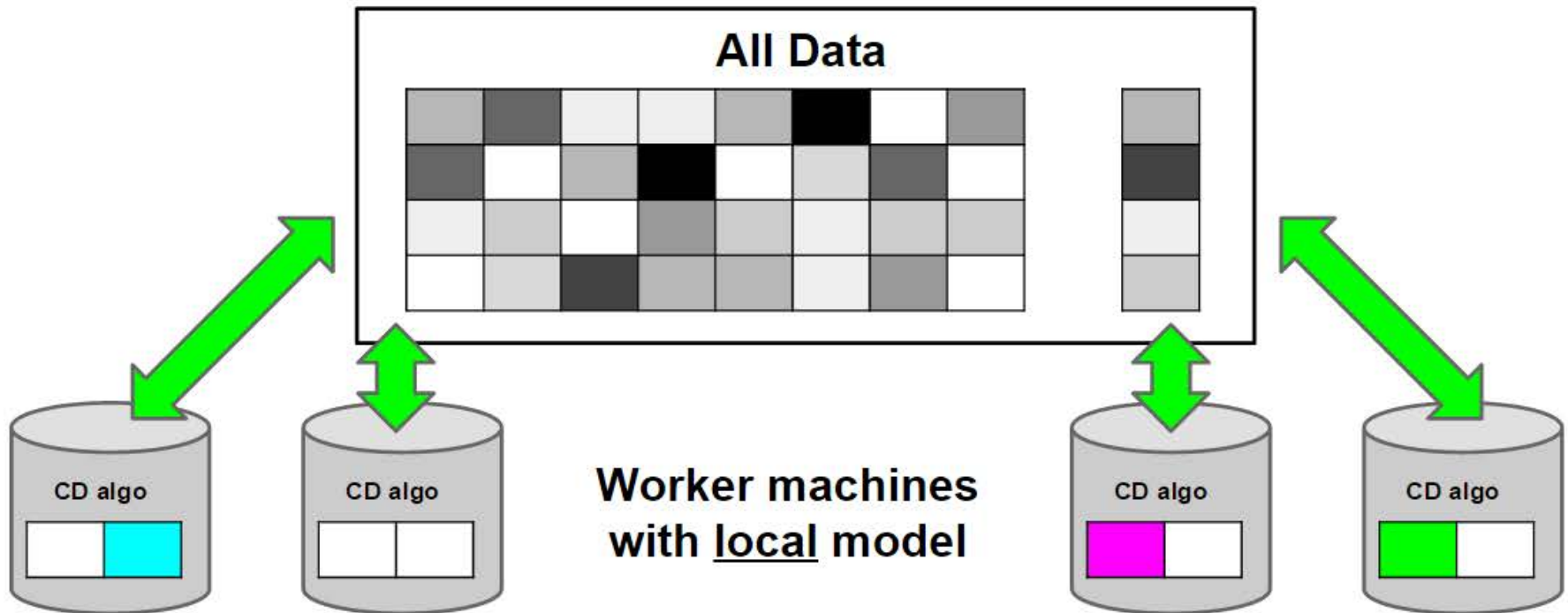
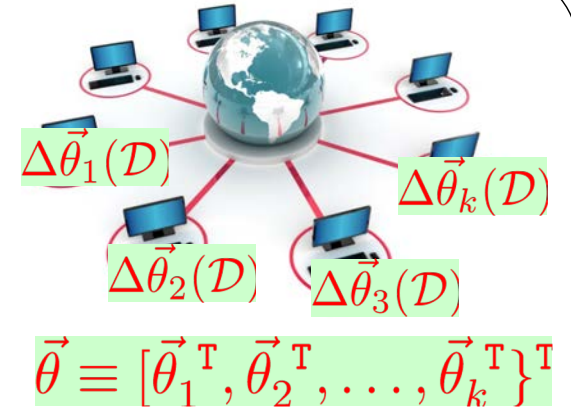
$$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_n\}$$

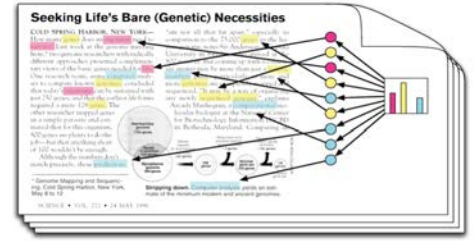


Partition rows of **Feature+Response Matrices**
across workers

Model-Parallel Lasso

Coordinate Descent:





Probabilistic Example:

Topic Models



Model (Topics) = B_k

gene 0.84	brain 0.84
dna 0.82	neuron 0.82
genetic 0.81	nerve 0.81
...	...
life 0.82	data 0.82
evolve 0.81	number 0.82
organism 0.81	computer 0.81
...	...

- Objective $L(\theta, D)$

- Log-likelihood of $D = \{\text{document words } x_{ij}\}$ given unknown $\theta = \{\text{document word topic indicators } z_{ij}, \text{ doc-topic distributions } \delta_i, \text{ topic-word distributions } B_k\}$:

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

- Prior $\rho(\theta)$

- Dirichlet prior on $\theta = \{\text{doc-topic, word-topic distributions}\}$

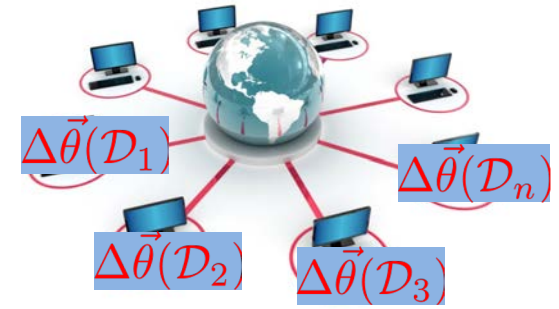
$$\sum_{i=1}^N \ln \mathbb{P}_{\text{Dirichlet}}(\delta_i | \alpha) + \sum_{k=1}^K \ln \mathbb{P}_{\text{Dirichlet}}(B_k | \beta)$$

- α, β are “hyperparameters” that control the Dirichet prior’s strength

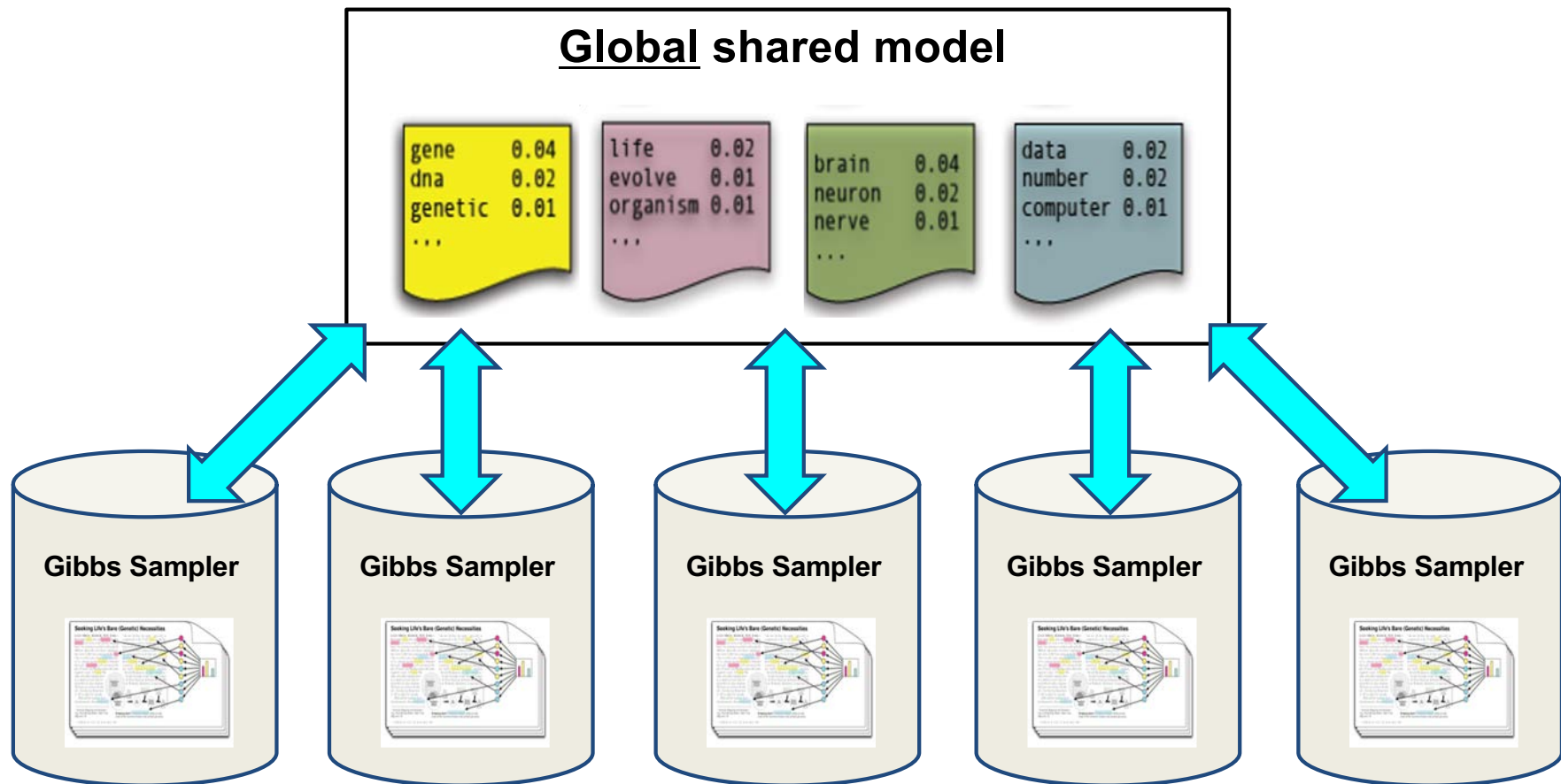
- Algorithm

- Collapsed Gibbs Sampling

Data Parallel Gibbs

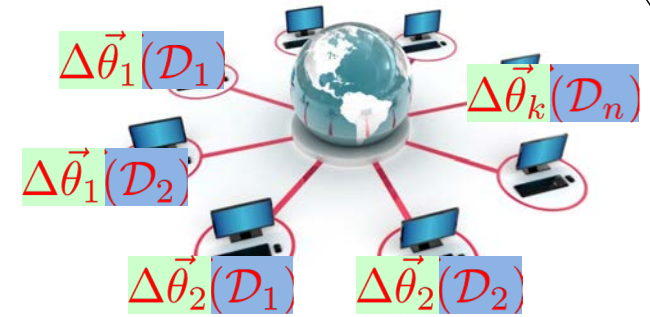
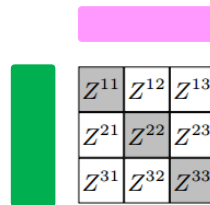


$$\mathcal{D} \equiv \{D_1, D_2, \dots, D_n\}$$



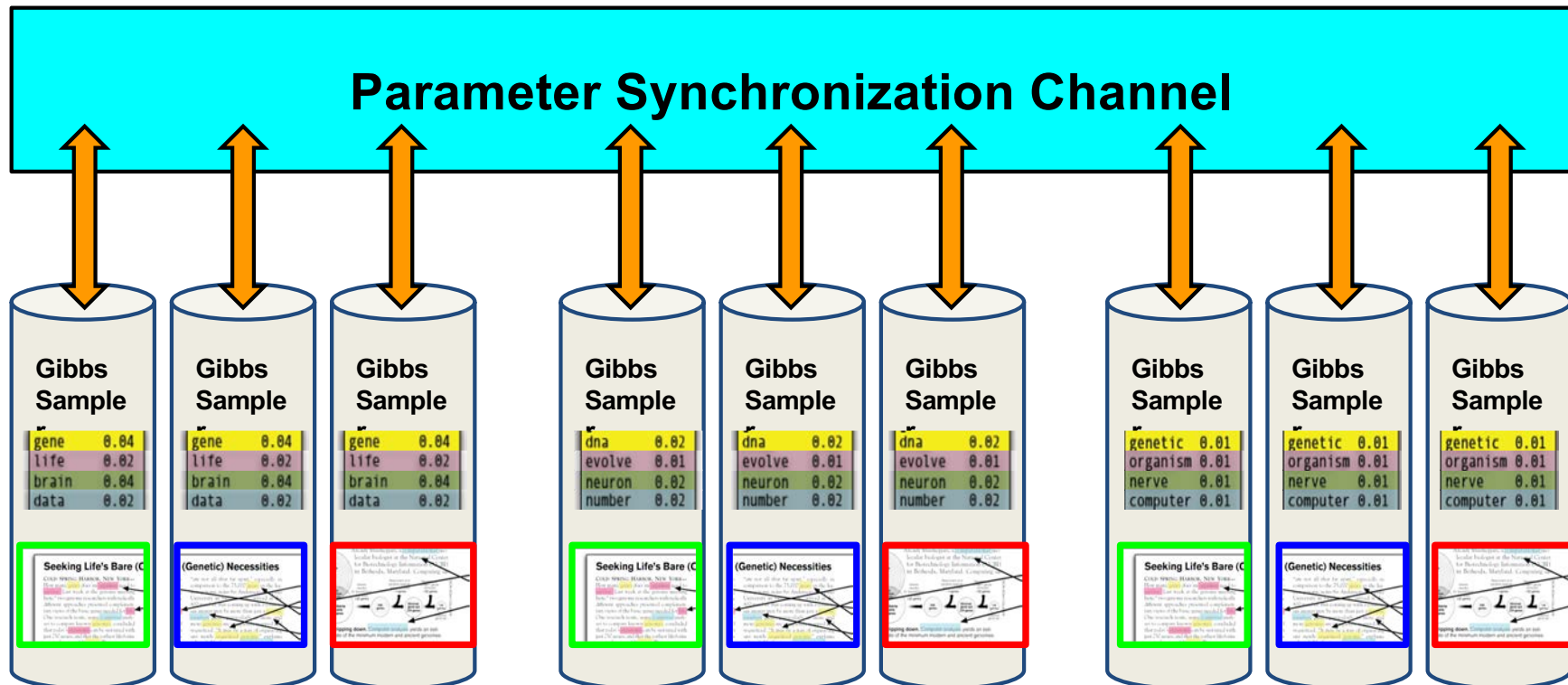
D+M Parallel Gibbs

Pair up vocabulary words with documents, divide across workers



$$D \equiv \{D_1, D_2, \dots, D_n\}$$

$$\vec{\theta} \equiv [\vec{\theta}_1^T, \vec{\theta}_2^T, \dots, \vec{\theta}_k^T]^T$$



What's Next?

First-timer's "Ideal View" of ML

```
global model = (a,b,c,...)  
global data = load(file)
```

```
Update(var a):
```

```
  a = doSomething(data,model)
```

```
Main:
```

```
  do Update() on all var in  
  model until converged
```

Reality of High-performance implementations

Many considerations

- What data batch size?
- How to partition model?
- When to sync up model?
- How to tune step size?
- What order to Update()?

1000s of lines of extra code

Need a System Interface for Parallel ML
– Does ML really Stop at the Ideal View?

4 Principles of ML System Design

How to execute distributed-parallel ML programs?

ML program equations tell us “What to Compute”. But...

1. **How to Distribute?**
2. **How to Bridge Computation and Communication?**
3. **How to Communicate?**
4. **What to Communicate?**



Principles of ML system Design

[Xing et al., to appear 2016]

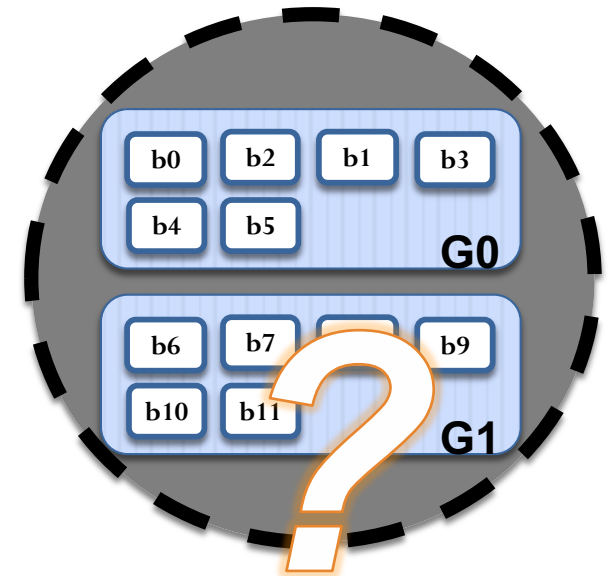
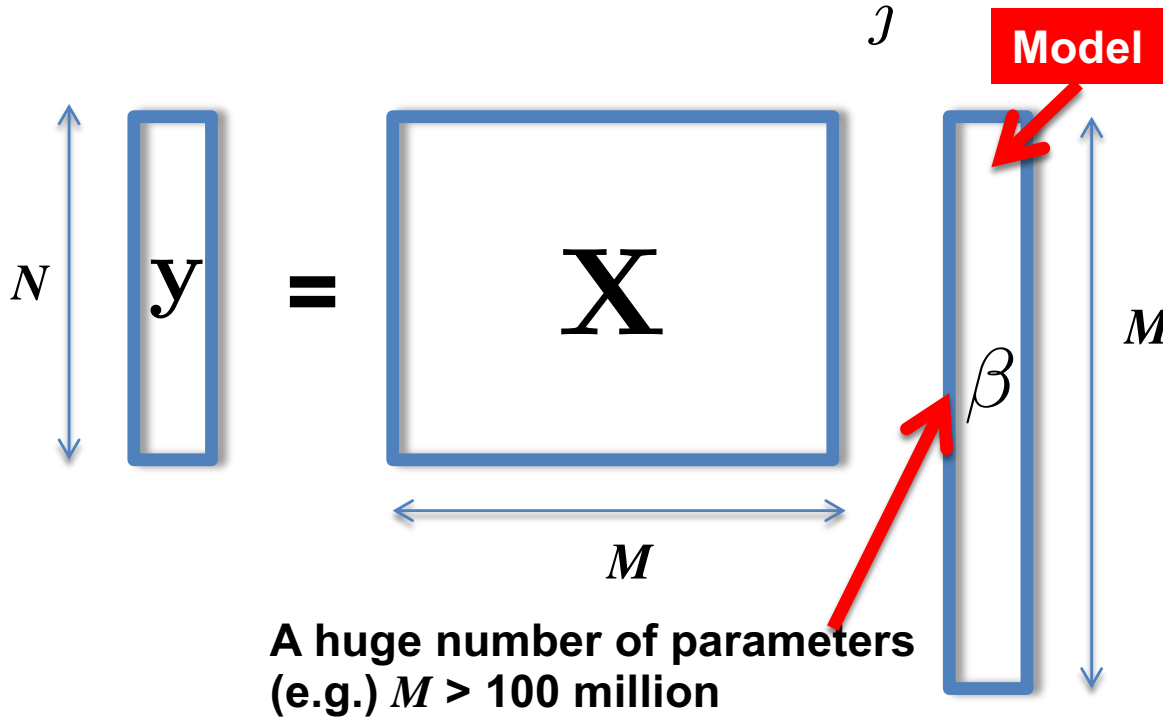
1. How to Distribute: *Scheduling and Balancing workloads*



Example: Model Distribution

Lasso via coordinate descent:

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

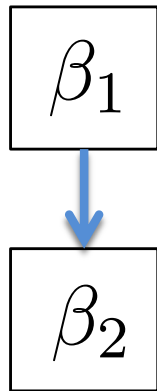


- How to correctly divide computational workload across workers?
- What is the best order to update parameters?

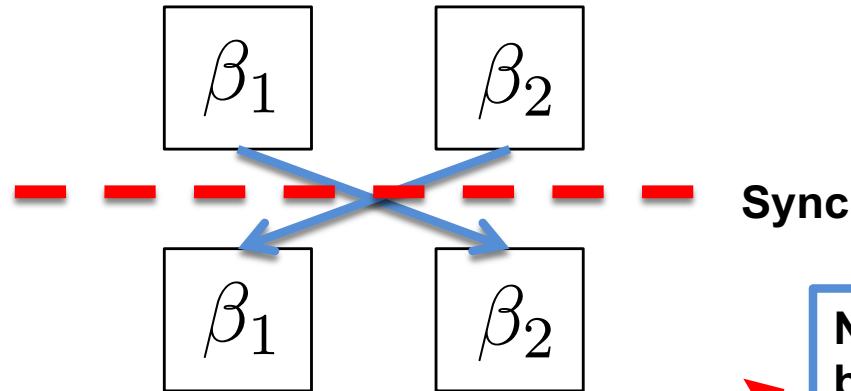
Model Dependencies

- Concurrent updates of β may induce errors

Sequential updates



Concurrent updates



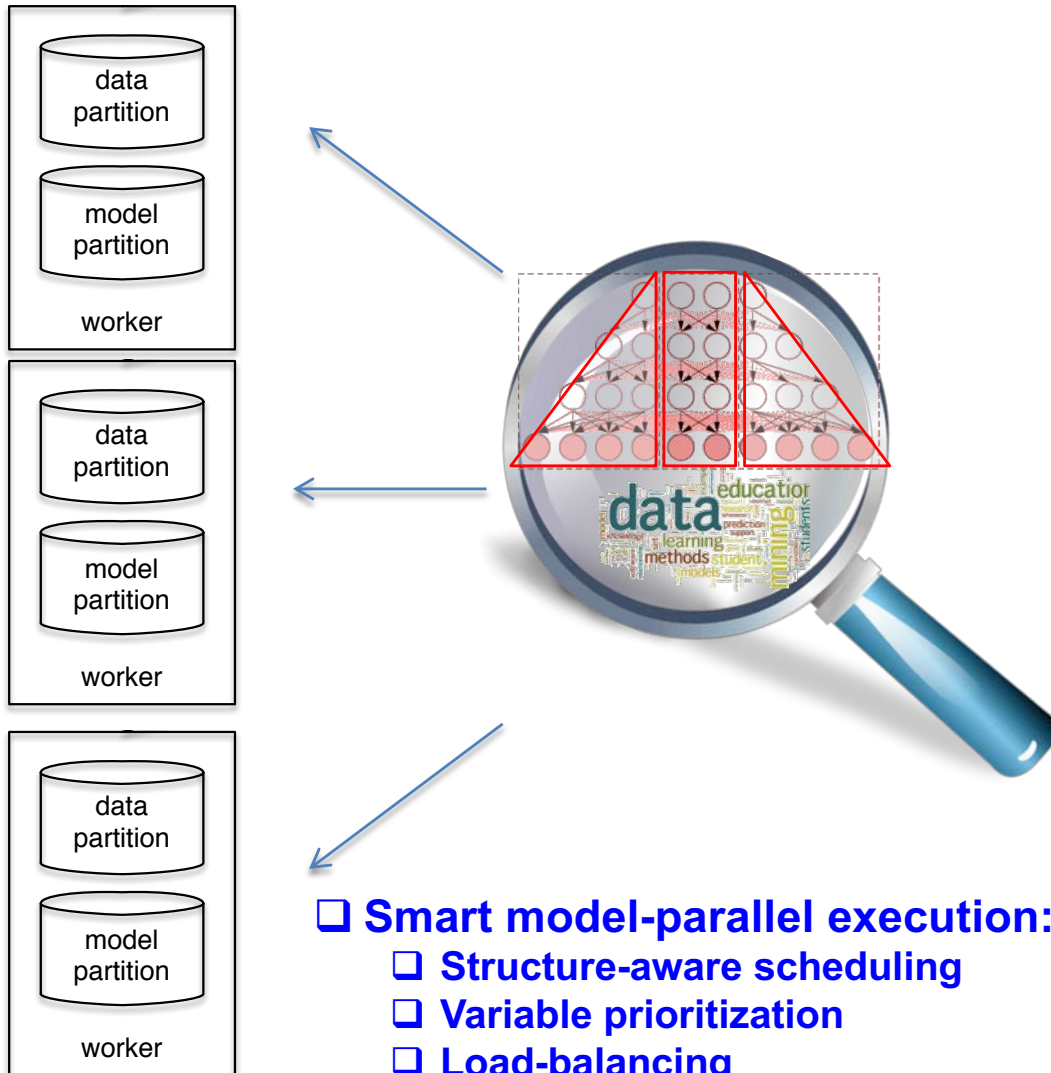
Need to check $\mathbf{x}_1^T \mathbf{x}_2$ before updating parameters

Decreases iteration progress

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

Avoid Dependency Errors via Structure-Aware Parallelization (SAP)

[Lee et al., 2014] [Kim et al., 2016]



```

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
}

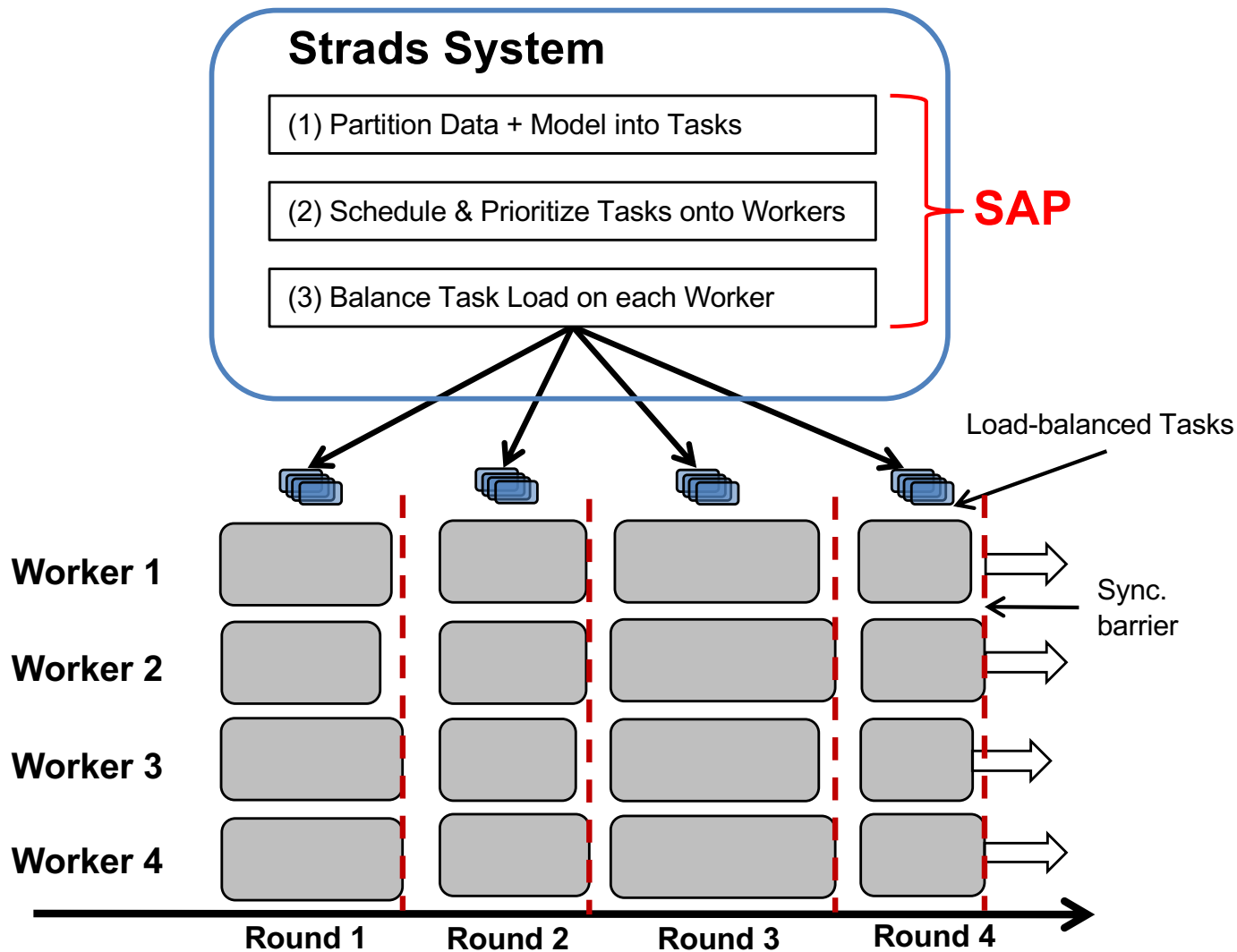
pull(workers = [p], vars = (x[j_1], ..., x[j_U]),
      updates = [z]) {
  // Use partial updates z from workers p to
  // update U vars x[j]. sync() is automatic.
  ...
}
  
```

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

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

A Structure-aware Dynamic Scheduler (Strads)

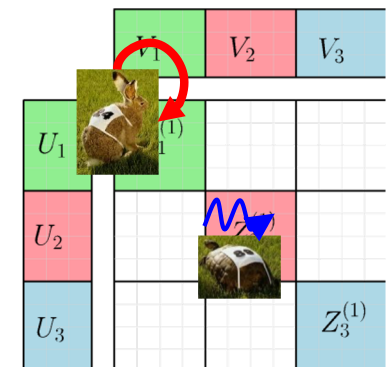
[Lee et al., 2014] [Kim et al., 2016]



- Priority Scheduling

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

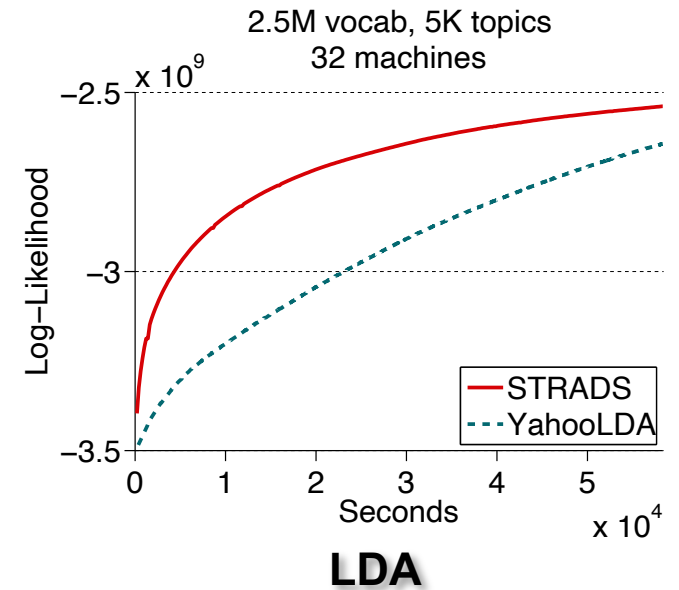
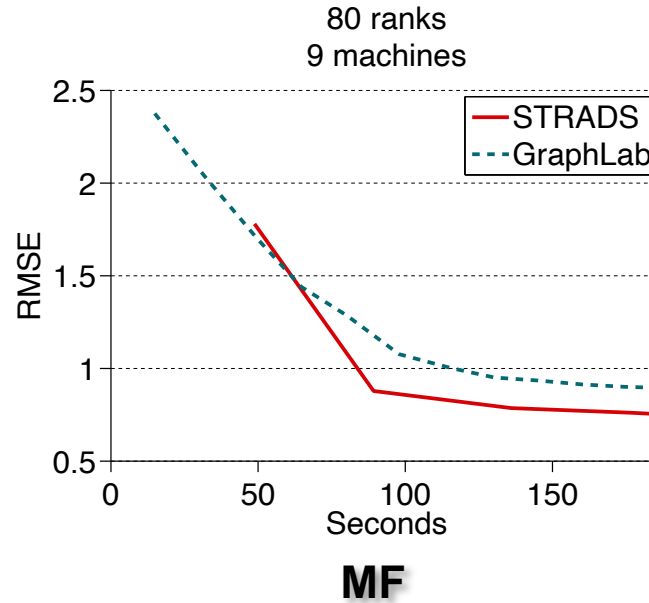
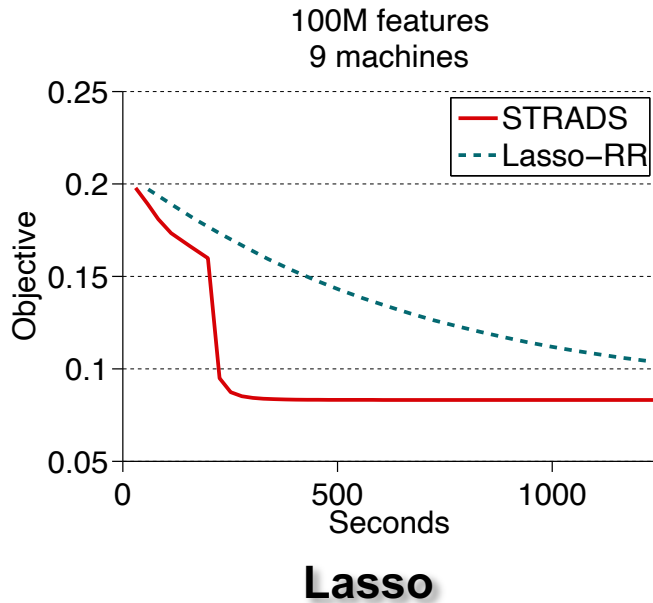
- Block scheduling



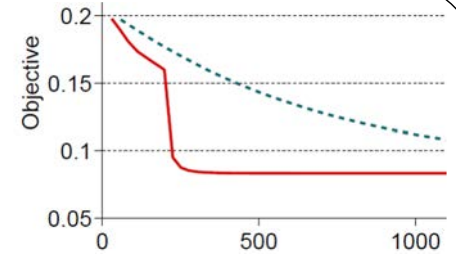
[Kumar, Beutel, Ho and Xing, *Fugue: Slow-worker agnostic distributed learning*, AISTATS 2014]

SAP Scheduling: Faster, Better Convergence across algorithms

- SAP on Strads achieves better speed and objective



SAP gives Near-Ideal Convergence Speed [Xing et al., 2015]



- **Goal:** solve sparse regression problem
 - Via coordinate descent over “SAP blocks” $X^{(1)}, X^{(2)}, \dots, X^{(B)}$
 - $X^{(b)}$ are data columns (features) in block (b)
 - P parallel workers, M -dimensional data
 - $\rho = \text{Spectral Radius}[\text{BlockDiag}[(X^{(1)})^T X^{(1)}, \dots, (X^{(t)})^T X^{(t)}]]$; this block-diagonal matrix quantifies max level of correlation within all SAP blocks $X^{(1)}, X^{(2)}, \dots, X^{(t)}$
- **SAP converges according to**

Gap between current parameter estimate and optimum SAP explicitly minimizes ρ , ensuring as close to $1/P$ convergence as possible

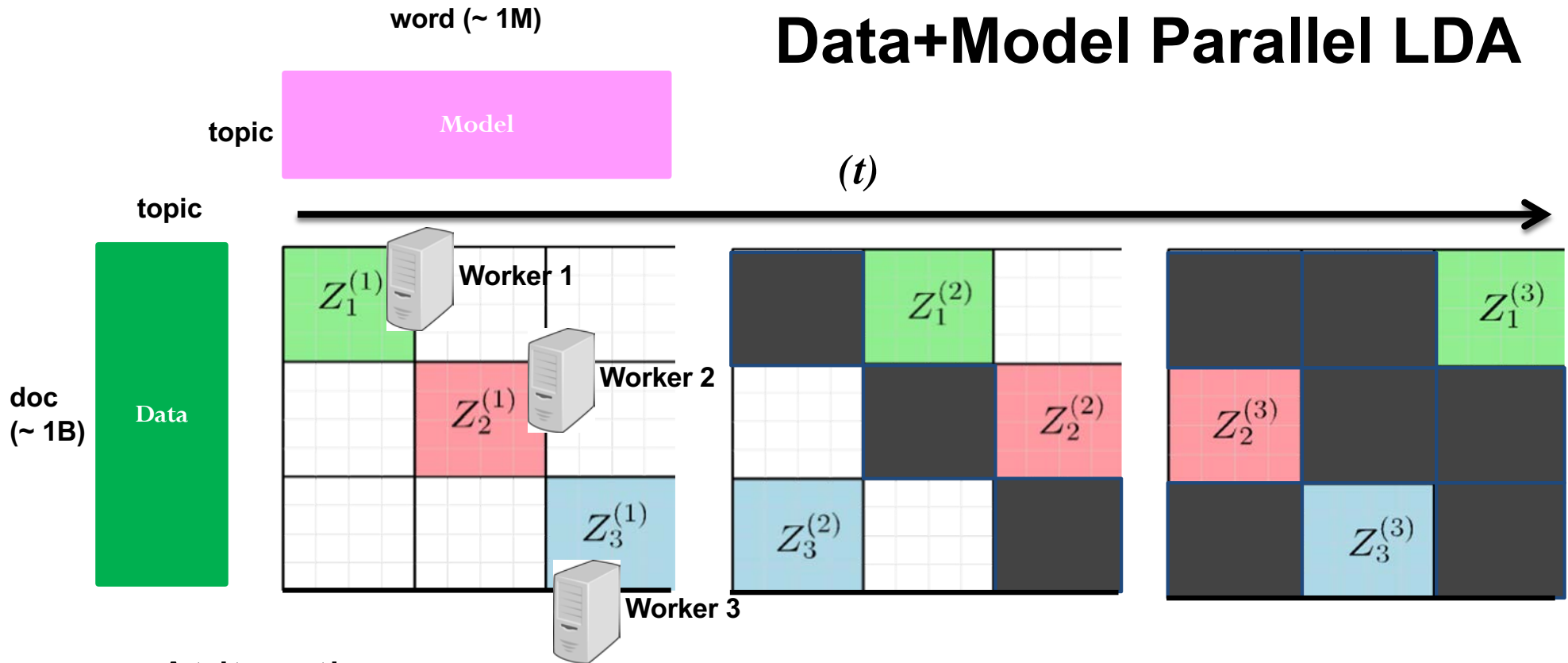
$$\mathbb{E} \left[\overbrace{f(X^{(t)}) - f(X^*)} \right] \leq \frac{\overbrace{\mathcal{O}(M)}}{\underbrace{P - \frac{\mathcal{O}(P^2 \rho)}{M}}_t} \frac{1}{t} = \mathcal{O} \left(\frac{1}{Pt} \right)$$

where t is # of iterations

- **Take-away:** SAP minimizes ρ by searching for feature subsets $X^{(1)}, X^{(2)}, \dots, X^{(B)}$ w/o cross-correlation => as close to P -fold speedup as possible

How to SAP-LDA

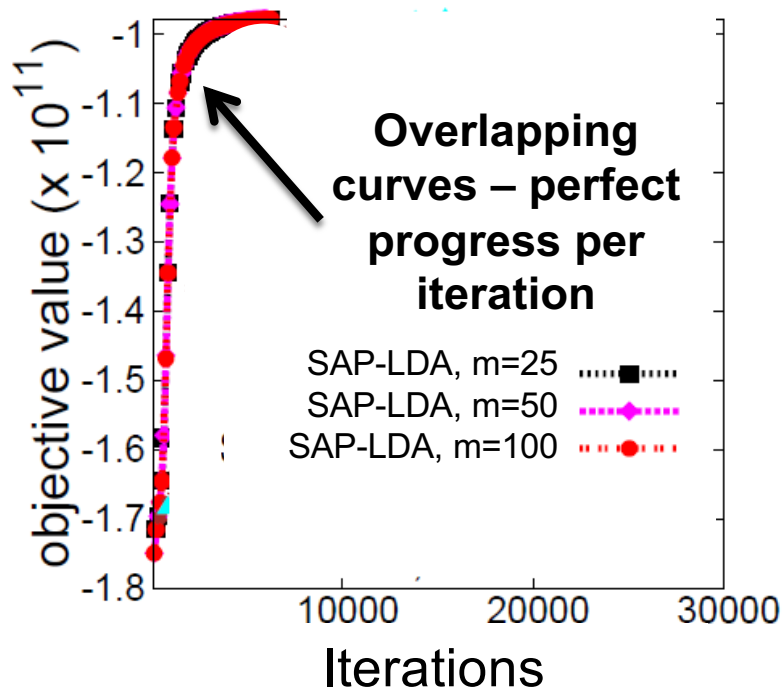
[Zheng et al., to appear 2016]



- At iteration (t) :
 - Worker 1 samples docs+words in $Z_1^{(t)}$
 - Worker 2 $\leftarrow Z_2^{(t)}$, Worker 3 $\leftarrow Z_3^{(t)}$ and so on...
 - Use different-sized $Z_p^{(t)}$ to load balance power-law tokens

Correctly Measuring Parallel Performance [blinded, to appear]

SAP-LDA progress per iteration



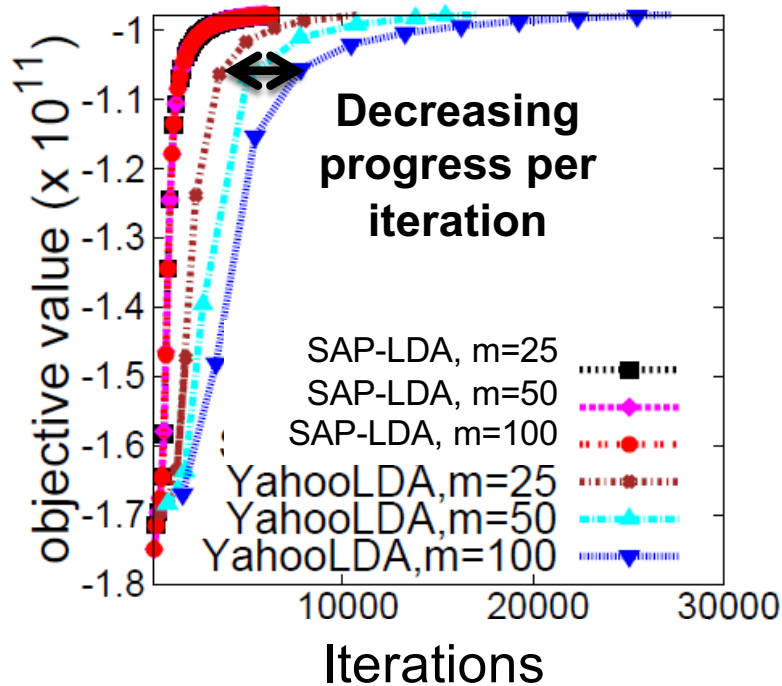
80GB data, 2M words,
1K topics, 100 machines

	SAP-LDA data throughput
25 machines	58.3 M/s (1x)
50 machines	114 M/s (1.96x)
100 machines	204 M/s (3.5x)

- Ideal rate:** progress per iter preserved from 25 \rightarrow 100 machines
 - Thanks to dependency checking
- Near-ideal throughput:** data rate 1x \rightarrow 3.5x from 25 \rightarrow 100 machines
 - Thanks to load balancing
- Convergence Speed = rate x throughput**
 - Therefore near-ideal 3.5x speedup from 25 \rightarrow 100 machines

Correctly Measuring Parallel Performance [blinded, to appear]

YahooLDA progress per iteration



**80GB data, 2M words,
1K topics, 100 machines**

	YahooLDA data throughput
25 machines	39.7 M/s (1x)
50 machines	78 M/s (1.96x)
100 machines	151 M/s (3.8x)

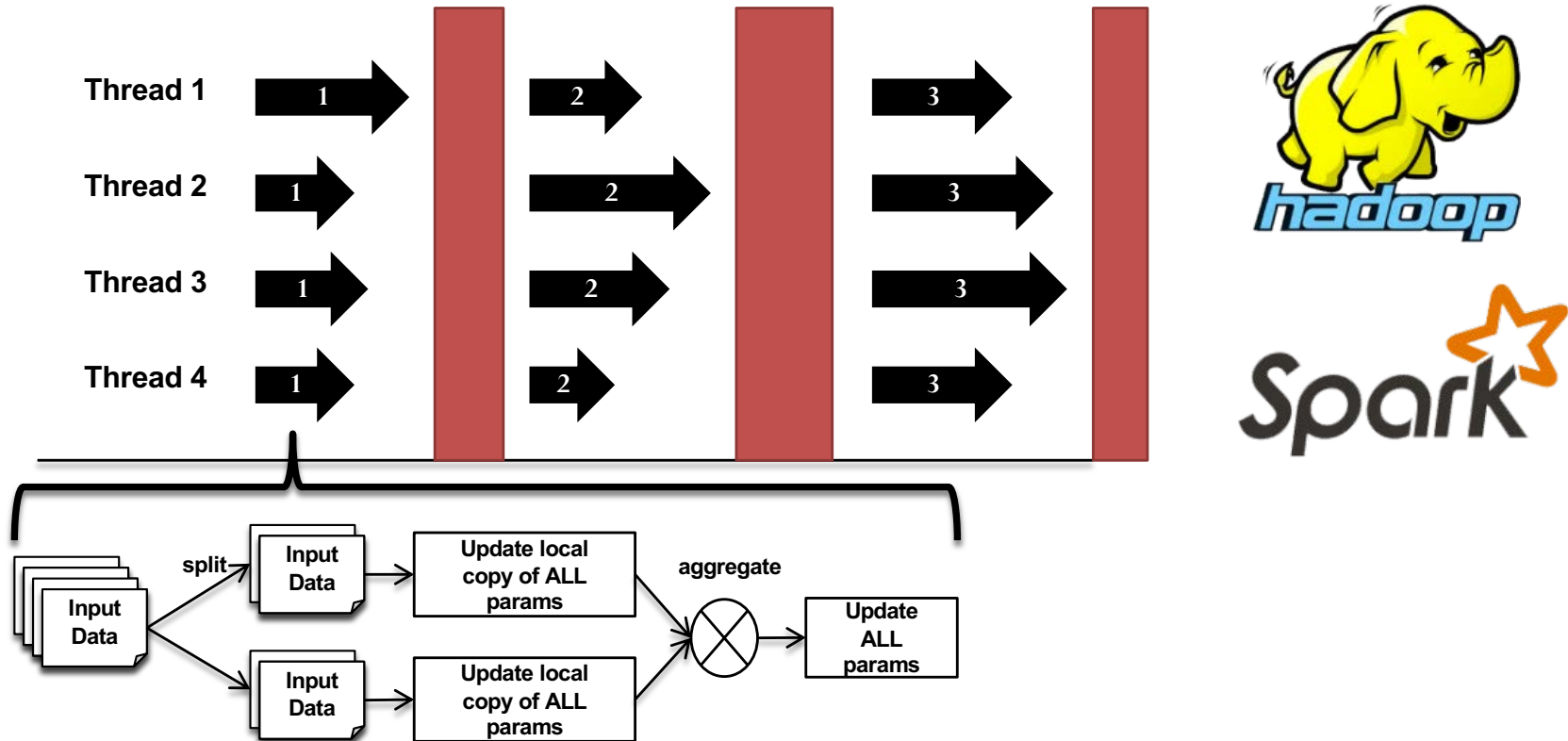
- YahooLDA attains near-ideal throughput (1→3.8x)...
- ... but progress per iteration gets worse with more machines
- **YahooLDA only <2x speedup from 25 → 100 machines**
 - 6.7x slower compared to SAP-LDA

Principles of ML system Design [Xing et al., to appear 2016]

2. How to Bridge Computation and Communication: *Bridging Models and Bounded Asynchrony*

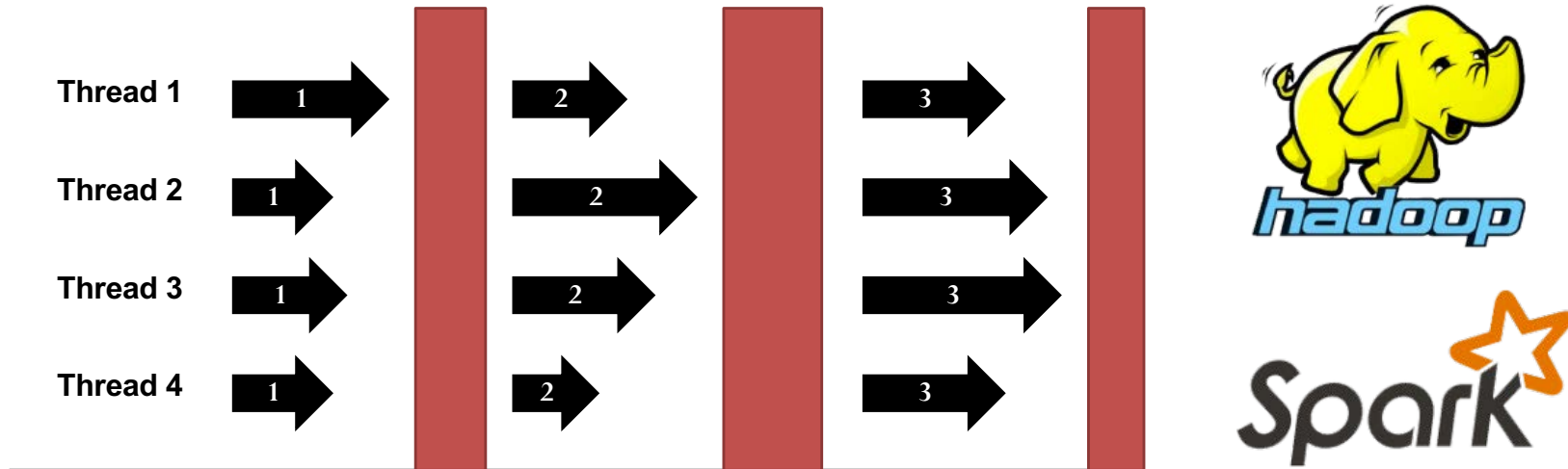


The *Bulk Synchronous Parallel* Bridging Model [Valiant & McColl]



- Perform barrier in order to communicate parameters
- Mimics sequential computation – “serializable” property
- Enjoys same theoretical guarantees as sequential execution

The *Bulk Synchronous Parallel* Bridging Model [Valiant & McColl]



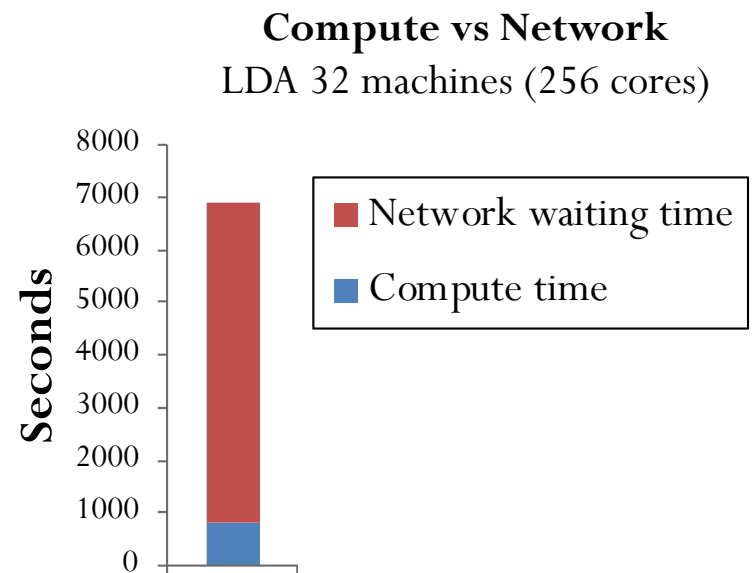
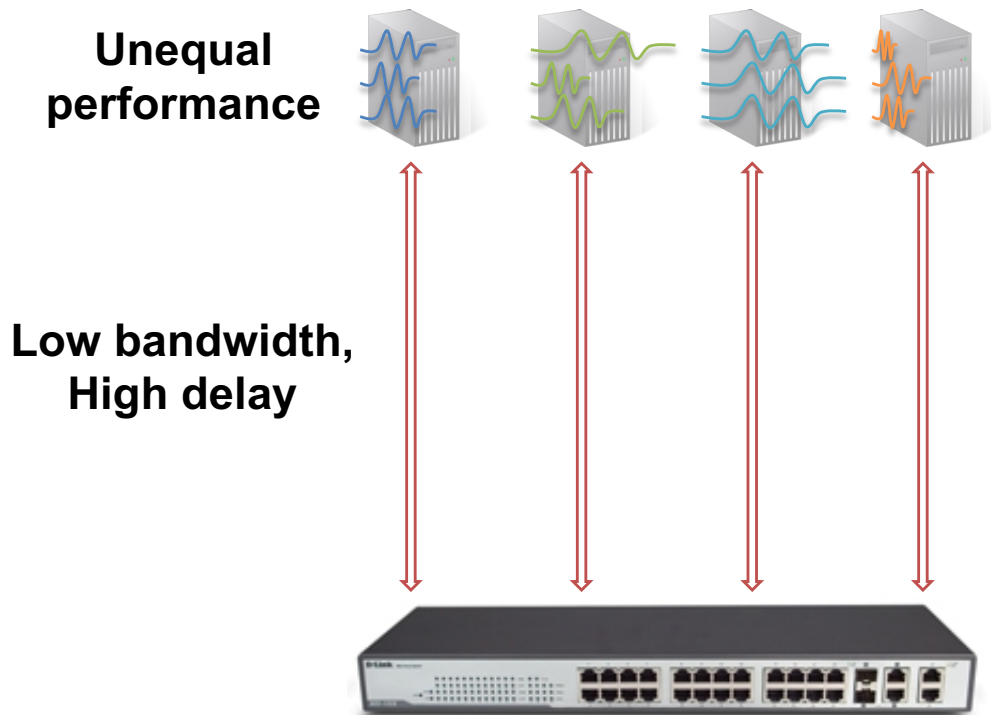
*The success of the von Neumann model of sequential computation is attributable to the fact it is **an efficient bridge between software and hardware...** an analogous bridge is required for parallel computation if that is to become as widely used – **Leslie G. Valiant***

- Numerous implementations since 90s (list by **Bill McColl**):
 - Oxford BSP Toolset ('98), Paderborn University BSP Library ('01), Bulk Synchronous Parallel ML ('03), BSPonMPI ('06), ScientificPython ('07), Apache Hama ('08), Apache Pregel ('09), MulticoreBSP ('11), BSPedupack ('11), Apache Giraph ('11), GoldenOrb ('11), Stanford GPS Project ('11) ...

But There Is No Ideal Distributed System!

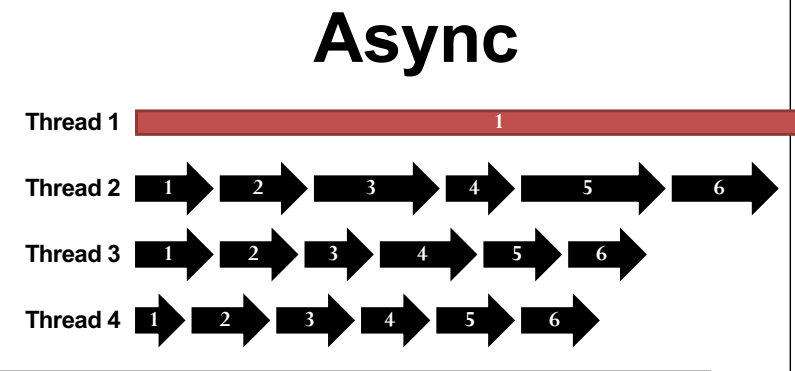
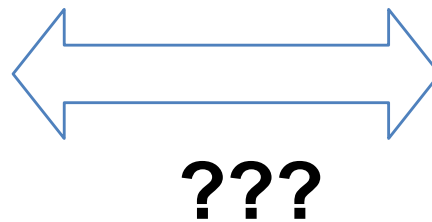
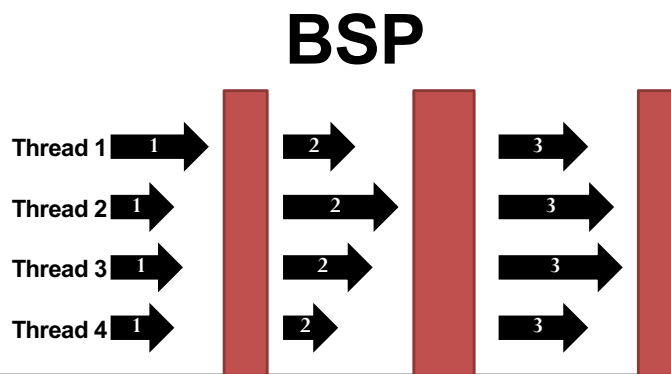
- **Two distributed challenges:**
 - Networks are slow
 - “Identical” machines rarely perform equally

Result: BSP barriers can be slow



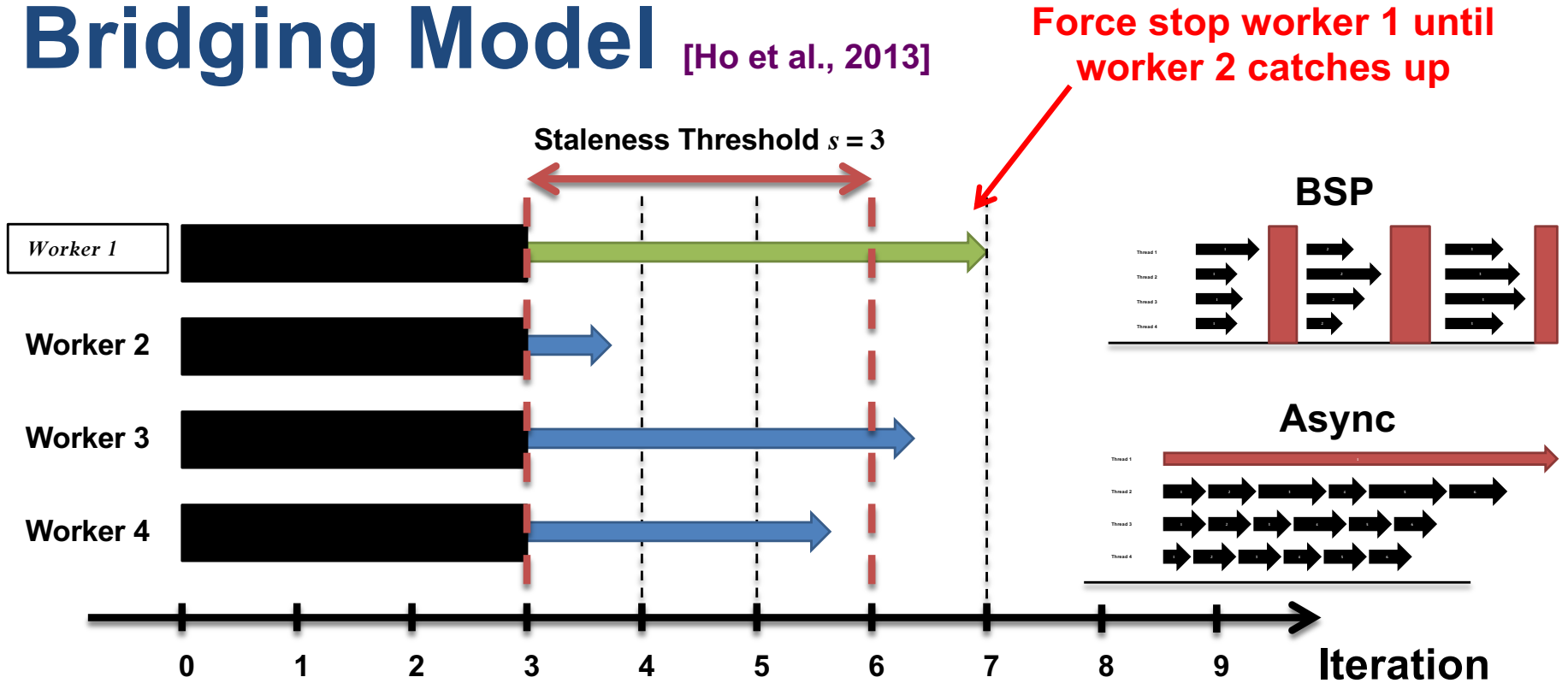
Is there a better way to interleave computation and communication?

- **Safe/slow (BSP) vs. 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?

A Stale Synchronous Parallel Bridging Model [Ho et al., 2013]



Stale Synchronous Parallel (SSP)

- Fastest/slowest workers not allowed to drift $>_s$ iterations apart

Consequence

- Fast like async, yet correct like BSP
- Why? Workers' local view of model parameters "not too stale" (\leq_s iterations old)

Data-Parallel Proximal Gradient under SSP

- Model (e.g. SVM, Lasso ...):

$$\min_{\mathbf{a} \in \mathbb{R}^d} \mathcal{L}(\mathbf{a}, D), \text{ where } \mathcal{L}(\mathbf{a}, D) = f(\mathbf{a}, D) + g(\mathbf{a})$$

data D , model a

- Algorithm:

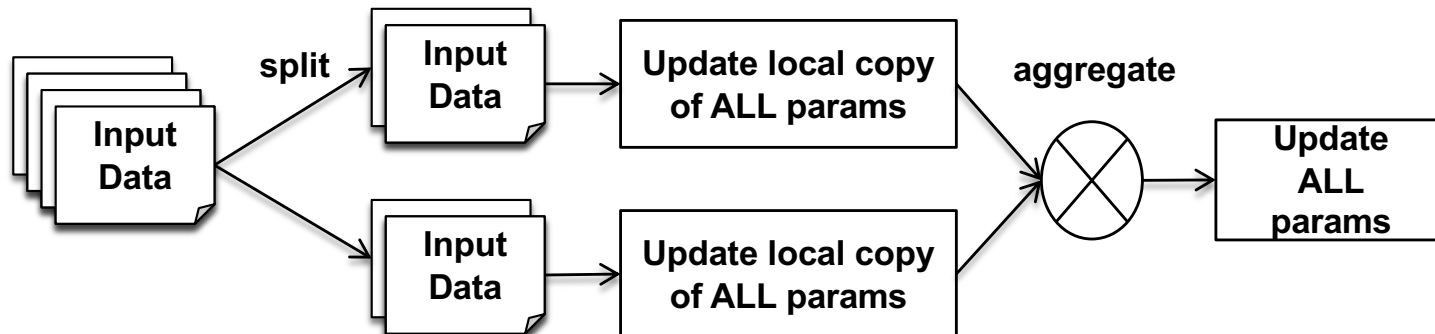
- Update $\mathbf{a}(t) := \underbrace{\text{prox}_g}_{\text{proximal step wrt } g} \left(\mathbf{a}^p(t) - \eta(t) \sum_{(p', t') \in \text{Recv}^p(t)} \overbrace{\Delta(\mathbf{a}^{p'}(t'), D_{p'})}^{\text{sub-update}} \right)$

stale sub-updates $\Delta()$ received by worker p at iteration t

- sub-update $\Delta(\mathbf{a}^p(t), D_p) := \underbrace{\nabla f(\mathbf{a}^p(t), D_p)}_{\text{gradient step wrt } f}$

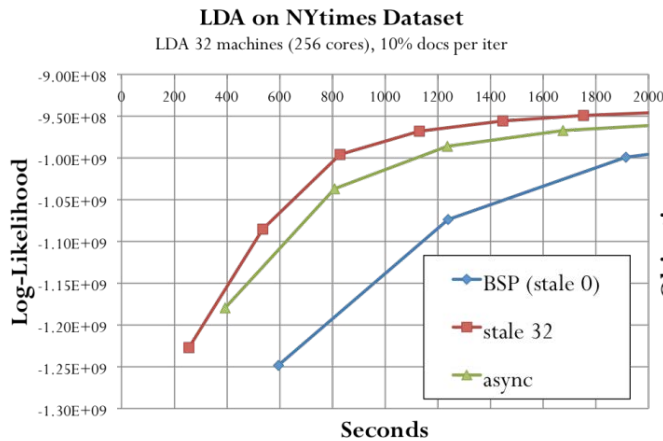
- Data parallel:

- Data D too large to fit in a single worker, divide among P workers

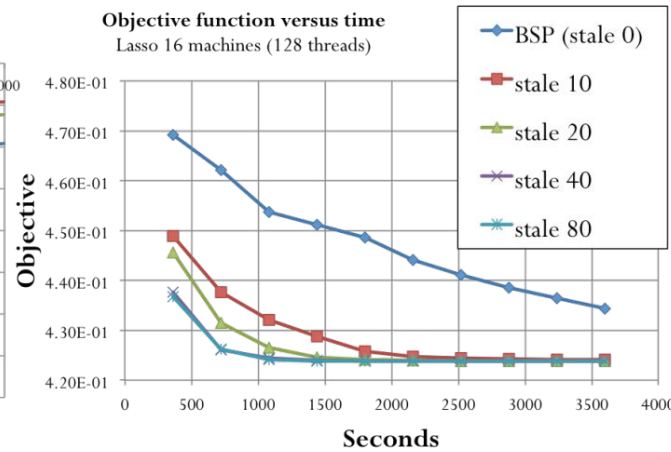


SSP Data-Parallel Async Speed, BSP Guarantee

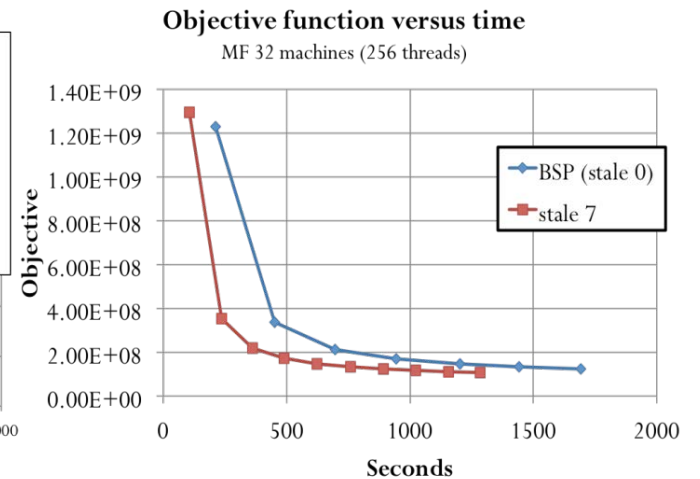
LDA



Lasso



Matrix Fact.



- Massive **Data** Parallelism
- Effective across different algorithms

SSP Data Parallel Convergence Theorem

[Ho et al., 2013, Dai et al., 2015]

Let observed staleness be γ_t

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

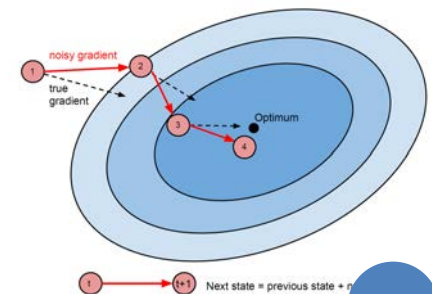
Theorem: Given L-Lipschitz objective f_t and step size h_t ,

$$P \left[\frac{R[X]}{T} - \frac{\mathcal{O}(F^2 + \mu_\gamma L^2)}{\sqrt{T}} \geq \tau \right] \leq \exp \left\{ \frac{-T\tau^2}{\mathcal{O}(\bar{\eta}_T \sigma_\gamma + L^2 s P \tau)} \right\}$$

where

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

Explanation: the distance between true optima and current estimate decreases exponentially with more SSP iterations. *Lower staleness mean, variance $\mu_\gamma, \sigma_\gamma$ improve the convergence rate.*



Model-Parallel Proximal Gradient under SSP

- Model (e.g. SVM, Lasso ...):

$$\min_{\mathbf{a} \in \mathbb{R}^d} \mathcal{L}(\mathbf{a}, D), \text{ where } \mathcal{L}(\mathbf{a}, D) = f(\mathbf{a}, D) + g(\mathbf{a})$$

data D , model a

- Model parallel
 - Model dimension d too large to fit in a single worker
 - Divide model among P workers $\mathbf{a} = (a_1, a_2, \dots, a_P)$.

- Algorithm:

$$\begin{aligned} \forall p, \quad a_p(t+1) &= a_p(t) + \gamma_p(t) \cdot F_p(\mathbf{a}^p(t)) \\ &= a_p(0) + \sum_{k=0}^t \gamma_p(k) \cdot F_p(\mathbf{a}^p(t)) \end{aligned}$$

workers can skip updates

staleness

(local) $\mathbf{a}^p(t) = (a_1(\tau_1^p(t)), \dots, a_P(\tau_P^p(t)))$

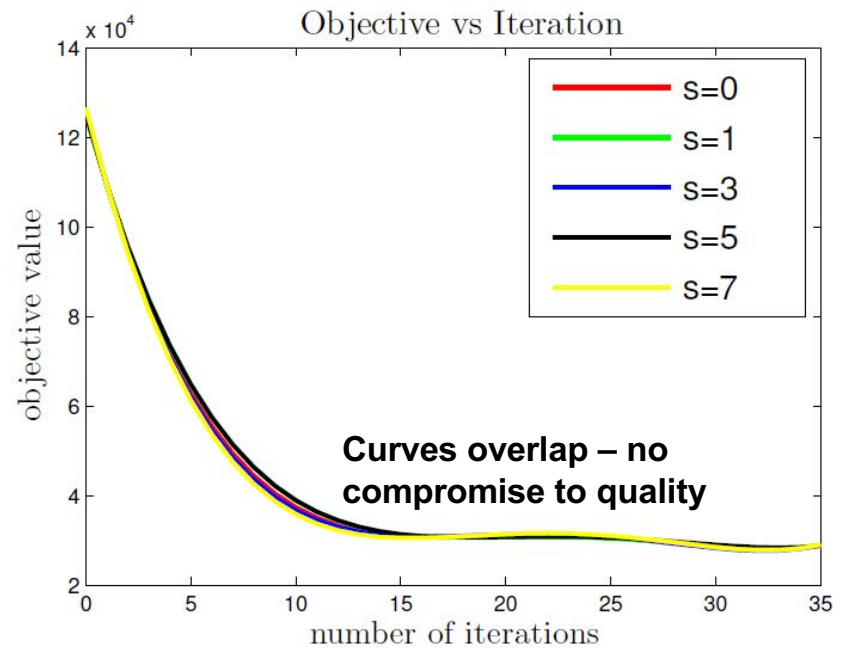
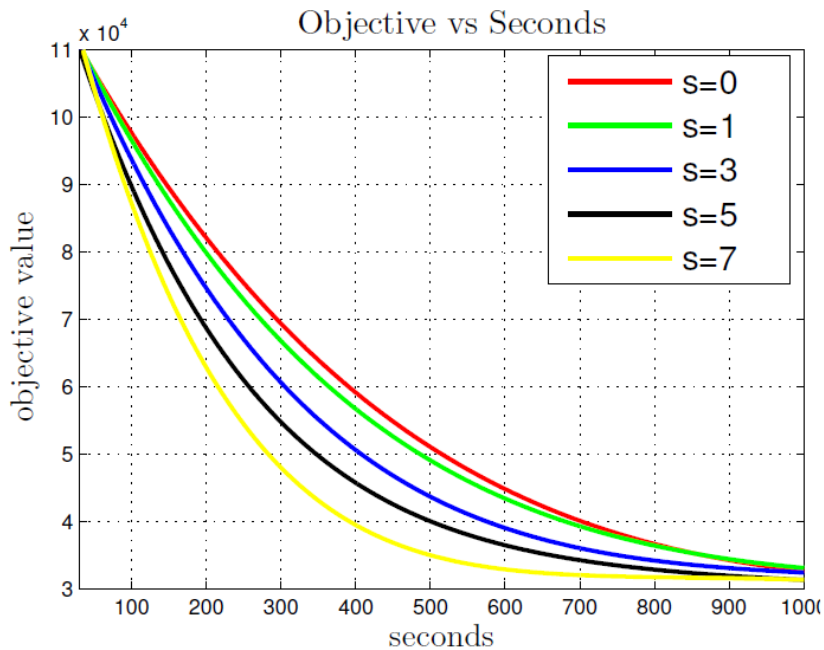
(global) $\mathbf{a}(t) = (a_1(t), \dots, a_P(t))$.

$$\mathbf{a}^p(t+1) := F_p(\mathbf{a}^p(t)) = \underbrace{\text{prox}_{g_p}^\eta}_{\text{proximal step wrt } g} \left(a_p(t) - \underbrace{\eta \nabla_p f(\mathbf{a}^p(t))}_{\text{gradient step wrt } f} \right) - a_p(t)$$

- worker p keeps local copy of the full model (can be avoided for linear models)

SSP Model-Parallel Async Speed, BSP Guarantee

Lasso: 1M samples, 100M features, 100 machines



- Massive **Model** Parallelism
- Effective across different algorithms

SSP Model Parallel Convergence Theorem

[Zhou et al., 2016]

Theorem: Given that the SSP delay is bounded, with appropriate step size and under mild technical conditions, then

$$\sum_{t=0}^{\infty} \|\mathbf{a}(t+1) - \mathbf{a}(t)\| < \infty \quad \sum_{t=0}^{\infty} \|\mathbf{a}^p(t+1) - \mathbf{a}^p(t)\| < \infty$$

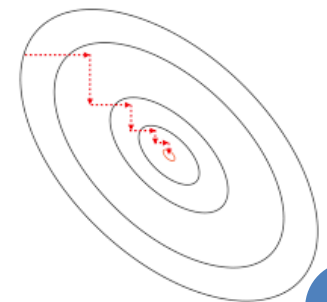
Finite length

In particular, the global and local sequences converge to the same critical point, with rate $O(t^{-1})$:

$$\mathcal{L} \left(\frac{1}{t} \sum_{k=1}^t \mathbf{a}(k) \right) - \inf \mathcal{L} \leq O(t^{-1})$$

Explanation: Finite length guarantees that the algorithm stops (the updates must eventually go to zero).

Furthermore, the algorithm **converges at rate $O(t^{-1})$ to the optimal value; same as BSP model parallel.**



Principles of ML system Design

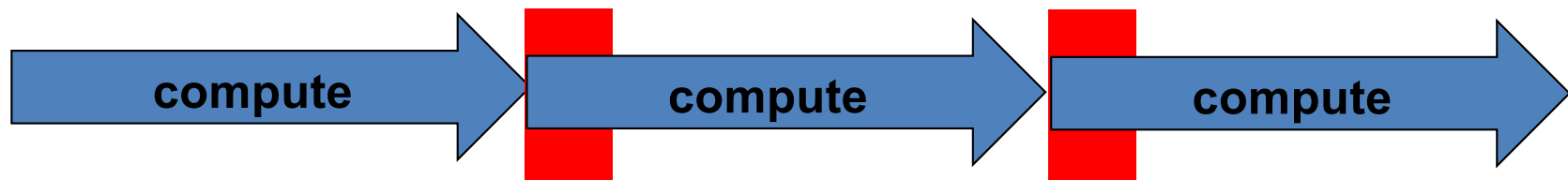
[Xing et al., to appear 2016]

3. How to Communicate: *Managed Communication and Topologies*

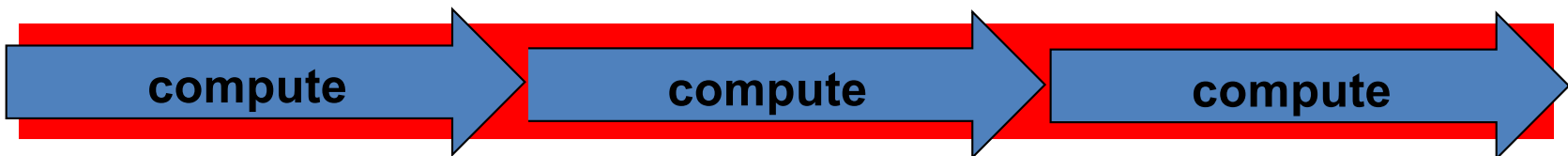


Managed Communication [Wei et al., 2015]

- SSP only
 - Communicates only at iteration boundary
 - Ensures bounded staleness consistency

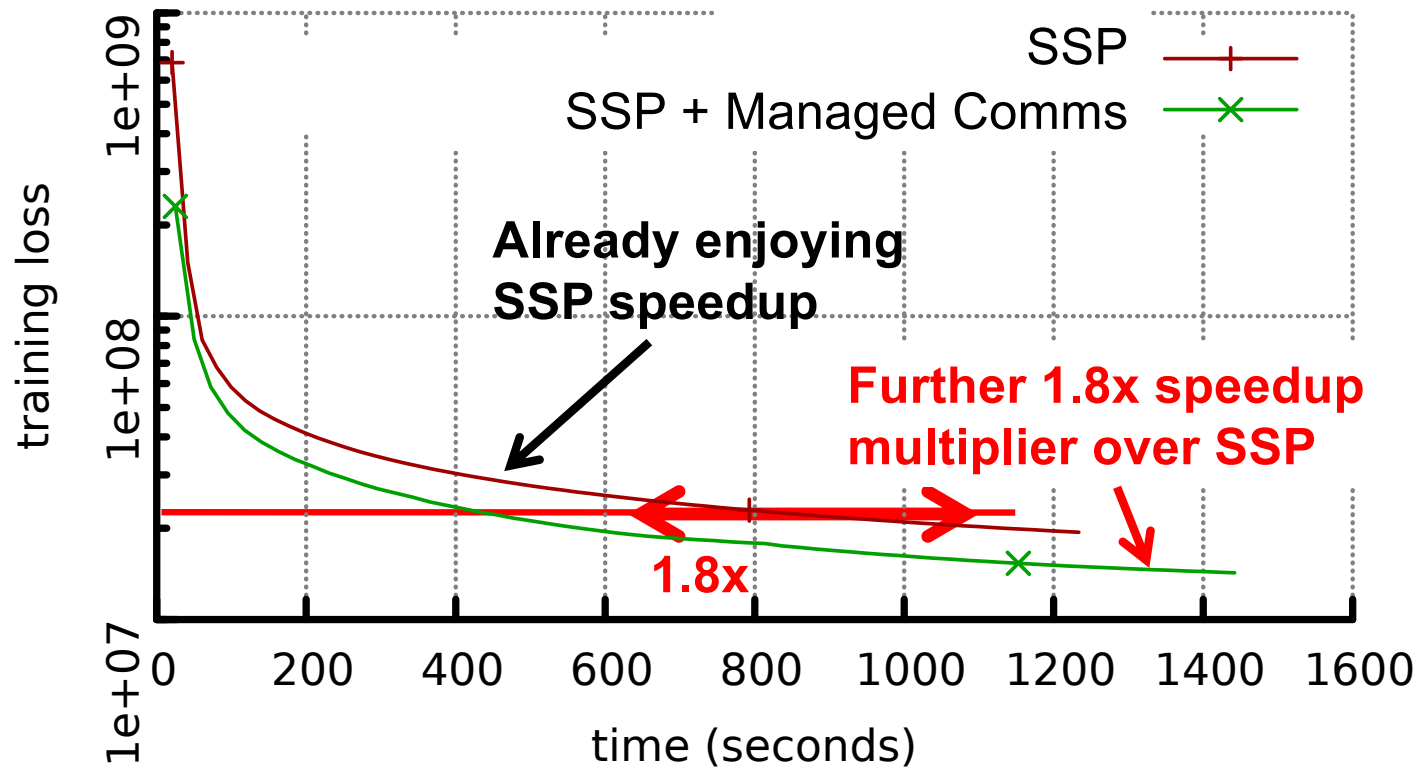


- SSP + Managed Communication
 - Continuous communication/synchronization
 - Update prioritization
 - Same consistency guarantees as SSP



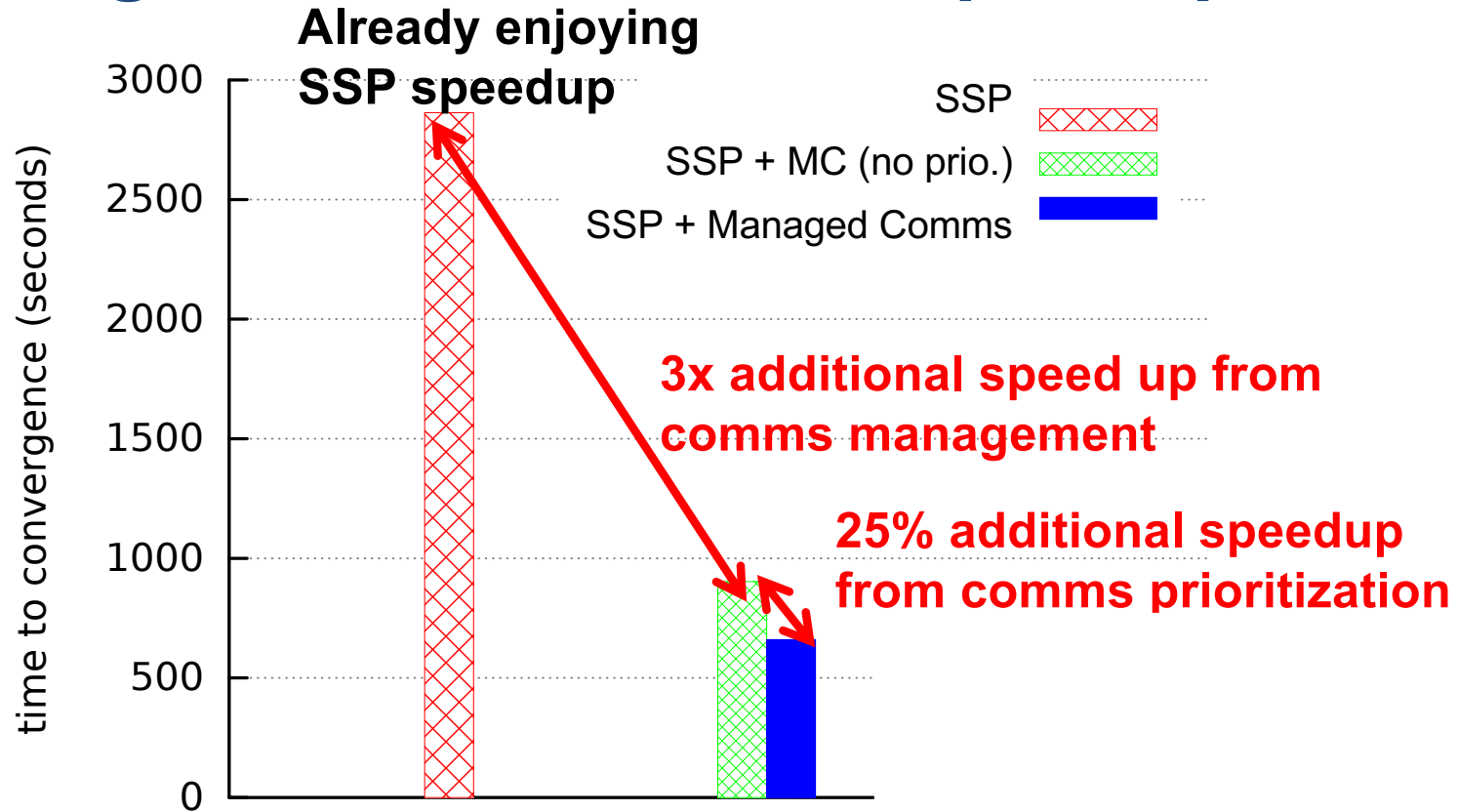
MatrixFact: Managed Communication Speedup

Lower
is better



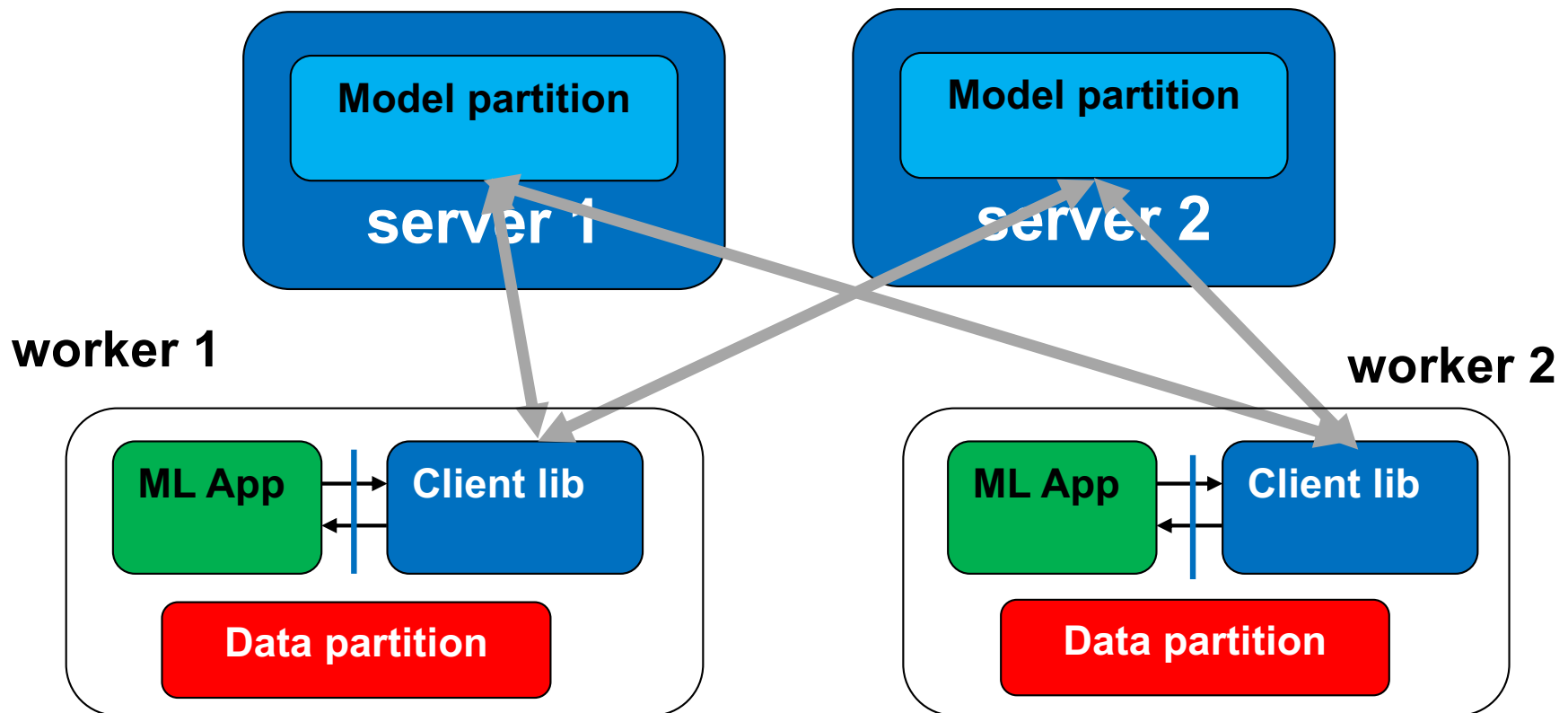
- Matrix Factorization, Netflix data, rank = 400
- 8 machines * 16 cores, 1GbE ethernet

LDA: Managed Communication Speedup



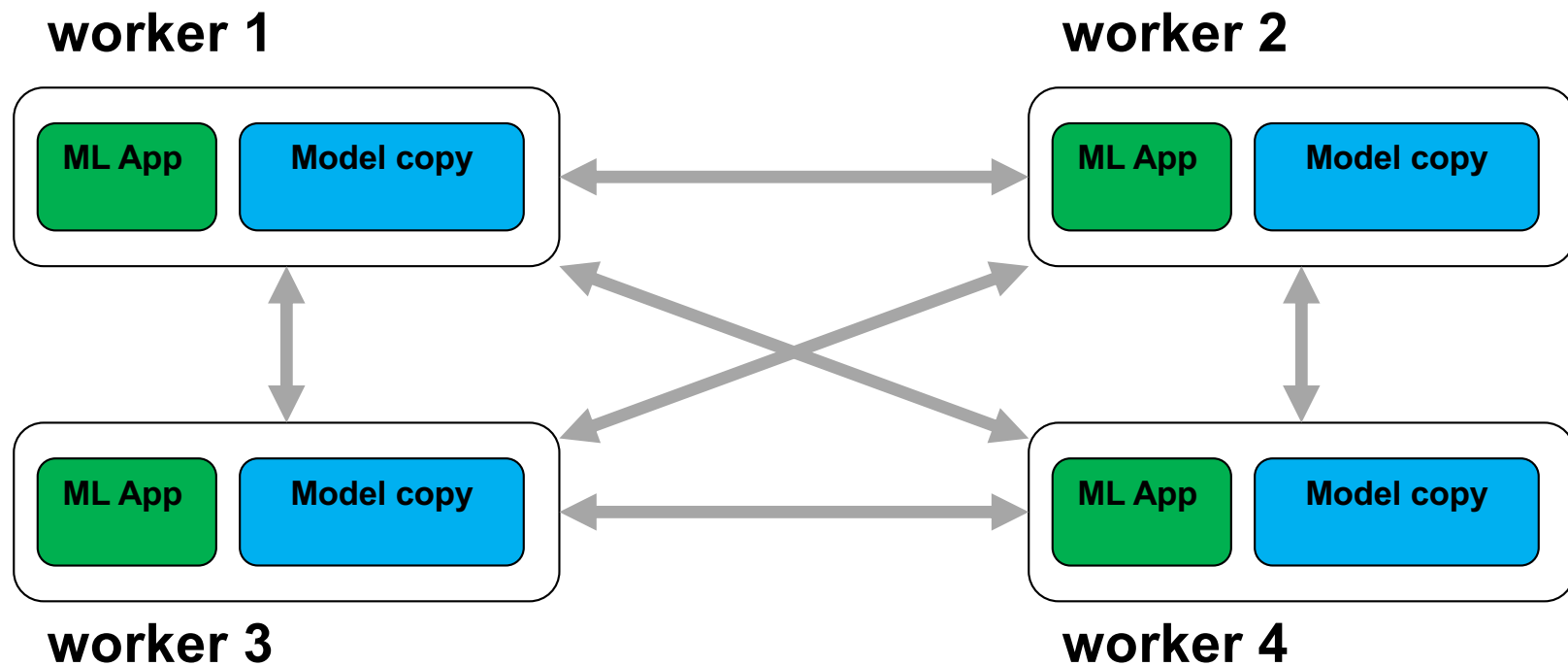
- Latent Dirichlet Allocation, NYTimes, # topics = 1000,
- 16 machines * 16 cores, 1GbE ethernet

Topology: Master-Slave



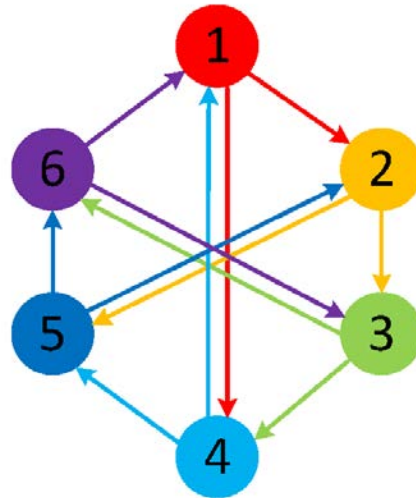
- Used with **centralized storage** paradigm
- Topology = **bipartite graph**: Servers (masters) to Workers (slaves)
- **Disadvantage**: need to code/manage clients and servers separately
- **Advantage**: bipartite topology far smaller than full N^2 P2P connections

Topology: Peer-to-Peer (P2P)



- Used with **decentralized storage** paradigm
- Workers update local parameter view by broadcasting/receiving
- **Disadvantage:** expensive unless updates ΔW are lightweight; expensive for large # of workers
- **Advantage:** only need worker code (no central server code); if ΔW is low rank, comms reduction possible

Halton Sequence Topology [Li et al., 2015]



- Used with **decentralized storage** paradigm
- Like P2P topology, but route messages through many workers
 - e.g. to send message from 1 to 6, use 1->2->3->6
- **Disadvantage:** incur higher SSP staleness due to routing, e.g. 1->2->3->6 = staleness 3
- **Advantage:** support bigger messages; support more machines than P2P topology

Principles of ML system Design

[Xing et al., to appear 2016]

4. What to Communicate: *Exploiting Structure in ML Updates*



Matrix-Parameterized Models (MPMs)

$$\min_W \underbrace{\frac{1}{N} \sum_{i=1}^N f_i(Wa_i; b_i)}_{\text{Loss function}} + \underbrace{h(W)}_{\text{Regularizer}}$$

Matrix parameter W

Distance Metric Learning, Sparse Coding, Distance Metric Learning, Group Lasso, Neural Network, etc.

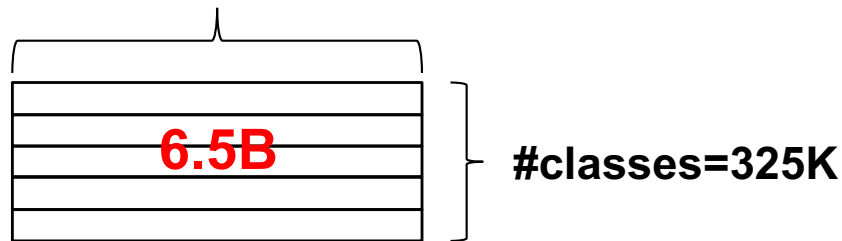
Big MPMs

Billions of params = 10-100 GBs, costly network synchronization

What do we actually need to communicate?

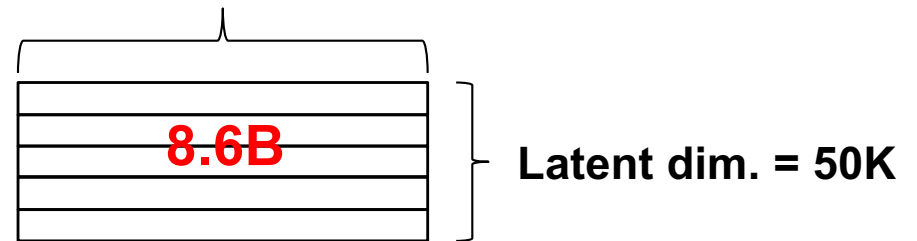
Multiclass Logistic Regression on Wikipedia

Feature dim. = 20K



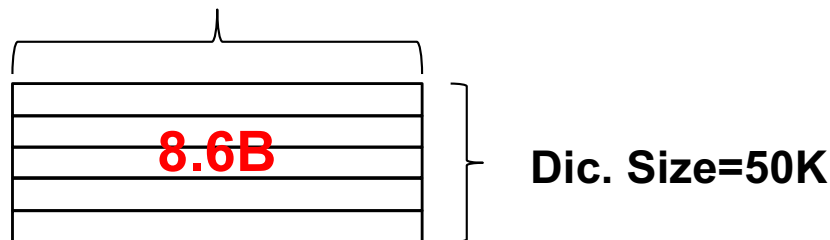
Distance Metric Learning on ImageNet

Feature dim. = 172K



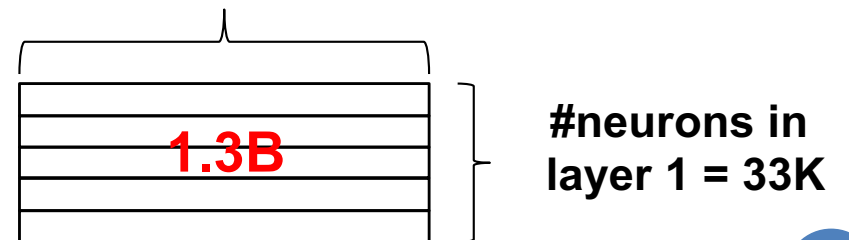
Sparse Coding on ImageNet

Feature dim. = 172K



Neural Network of Google Brain

#neurons in layer 0 = 40K



Full Updates

- Let matrix parameters be W . **Need to send parallel worker updates ΔW to other machines...**
 - Primal stochastic gradient descent (SGD)

$$\min_W \frac{1}{N} \sum_{i=1}^N f_i(Wa_i; b_i) + h(W)$$

$$\Delta W = \frac{\partial f(Wa_i, b_i)}{\partial W}$$

- Stochastic dual coordinate ascent (SDCA)

$$\min_Z \frac{1}{N} \sum_{i=1}^N f_i^*(-z_i) + h^*\left(\frac{1}{N} ZA^T\right)$$

$$\Delta W = (\Delta z_i) a_i$$

Sufficient Factor (SF) Updates

[Xie et al., 2015]

- **Full parameter matrix update ΔW can be computed as outer product of two vectors uv^T (called sufficient factors)**
 - Primal stochastic gradient descent (SGD)

$$\min_W \frac{1}{N} \sum_{i=1}^N f_i(Wa_i; b_i) + h(W)$$

$$\Delta W = uv^T \quad u = \frac{\partial f(Wa_i, b_i)}{\partial (Wa_i)} \quad v = a_i$$

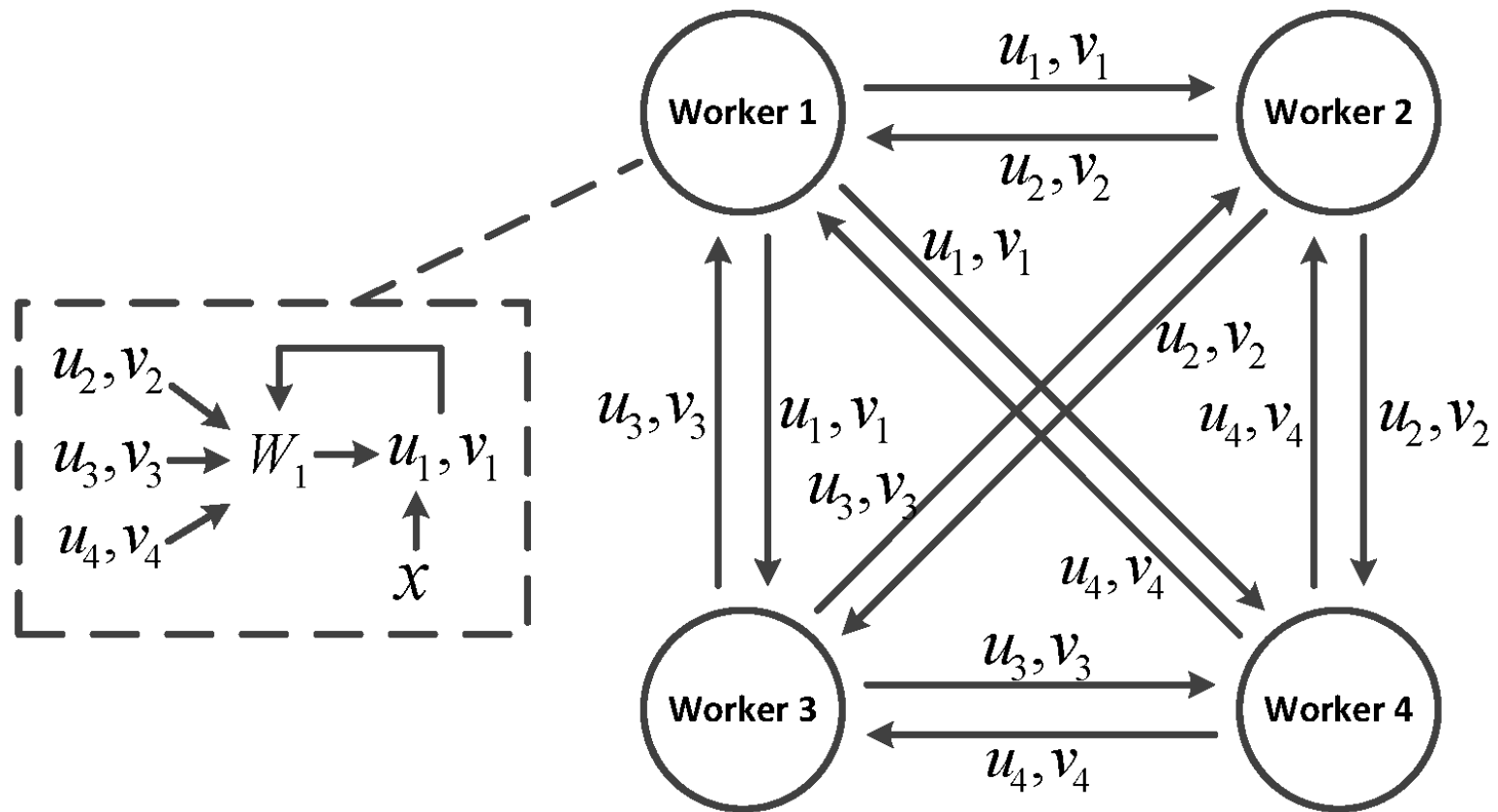
- Stochastic dual coordinate ascent (SDCA)

$$\min_Z \frac{1}{N} \sum_{i=1}^N f_i^*(-z_i) + h^*\left(\frac{1}{N} ZA^T\right)$$

$$\Delta W = uv^T \quad u = \Delta z_i \quad v = a_i$$

- Send the lightweight SF updates (u, v) , instead of the expensive full-matrix ΔW updates!

P2P Topology + SF Updates = Sufficient Factor Broadcasting



SFB Convergence Theorem

[Xie et al., 2015]

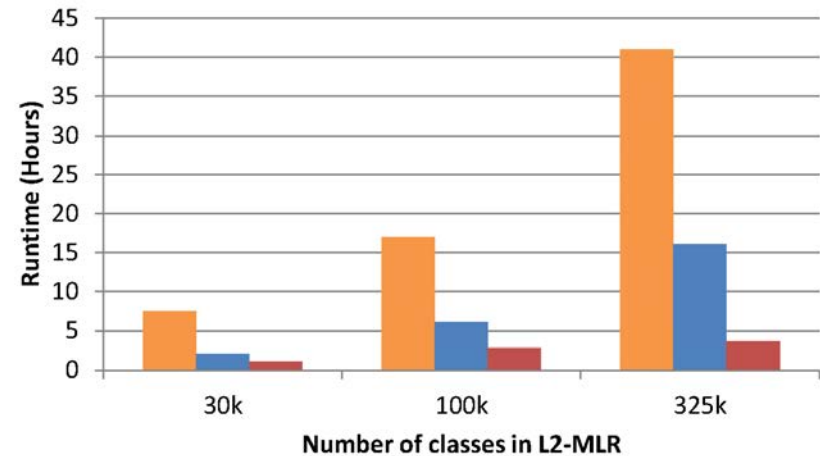
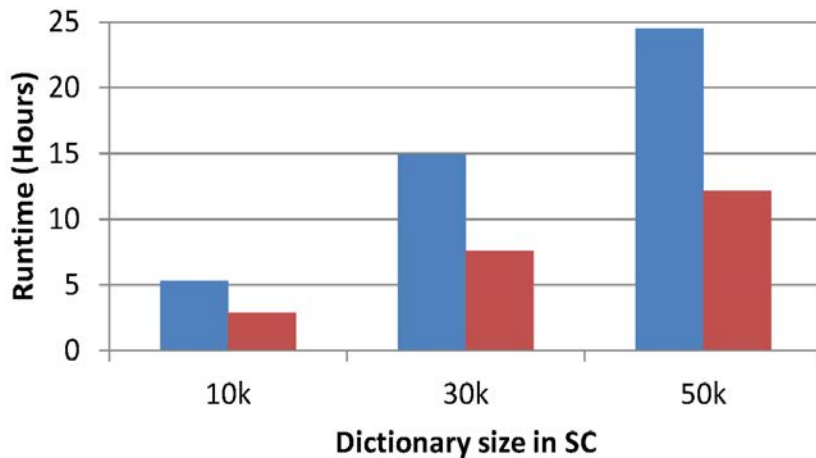
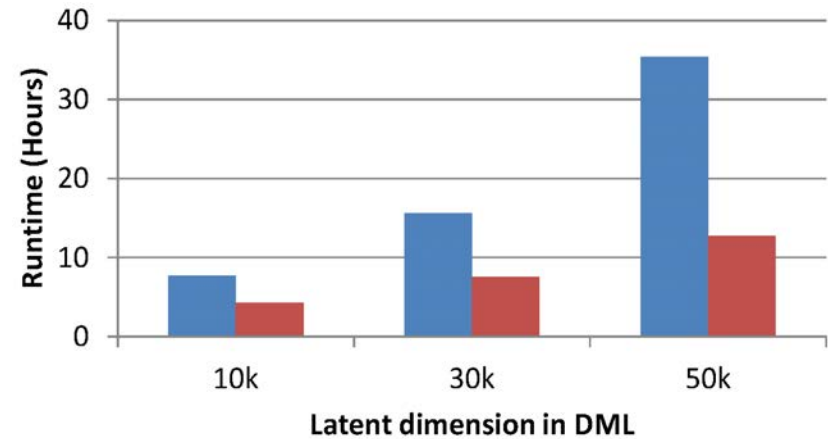
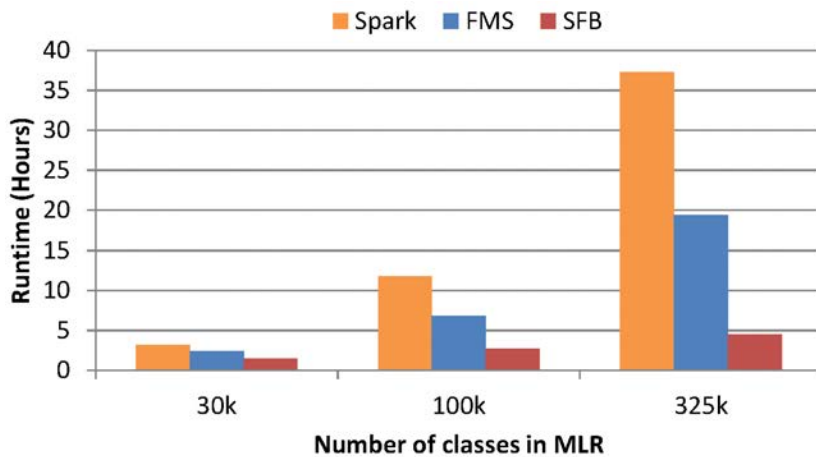
Theorem 1. Let $\{\mathbf{W}_p^c\}$, $p = 1, \dots, P$, and $\{\mathbf{W}^c\}$ be the local sequences and the auxiliary sequence generated by SFB for problem (P) (with $h \equiv 0$), respectively. Under Assumption 1 and set the learning rate $\eta_c^{-1} = \frac{L_F}{2} + 2sL + \sqrt{c}$, then we have

- $\liminf_{c \rightarrow \infty} \mathbb{E} \|\nabla F(\mathbf{W}^c)\| = 0$, hence there exists a subsequence of $\nabla F(\mathbf{W}^c)$ that almost surely vanishes;
- $\lim_{c \rightarrow \infty} \max_p \|\mathbf{W}^c - \mathbf{W}_p^c\| = 0$, i.e. the maximal disagreement between all local sequences and the auxiliary sequence converges to 0 (almost surely);
- There exists a common subsequence of $\{\mathbf{W}_p^c\}$ and $\{\mathbf{W}^c\}$ that converges almost surely to a stationary point of F , with the rate $\min_{c \leq C} \mathbb{E} \|\sum_{p=1}^P \nabla F_p(\mathbf{W}_p^c)\|_2^2 \leq O\left(\frac{(L+L_F)\sigma^2 P s \log C}{\sqrt{C}}\right)$.

Explanation: Parameter copies W_p on different workers p converge to the same optima, *i.e. all workers reach the same (correct) answer.*

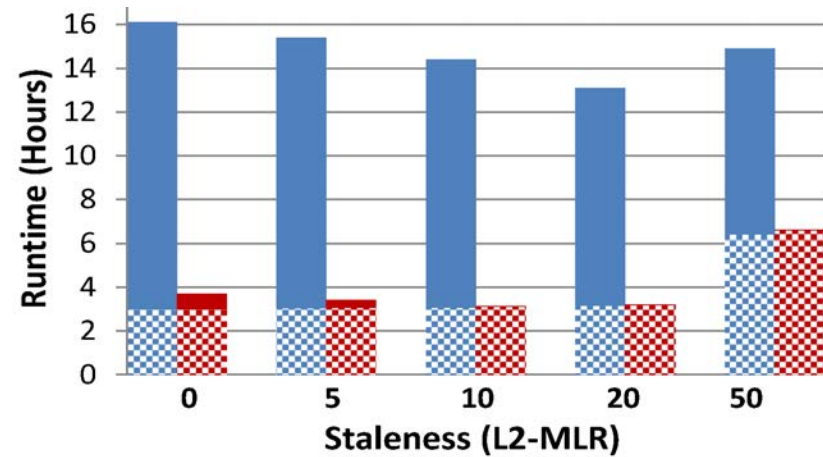
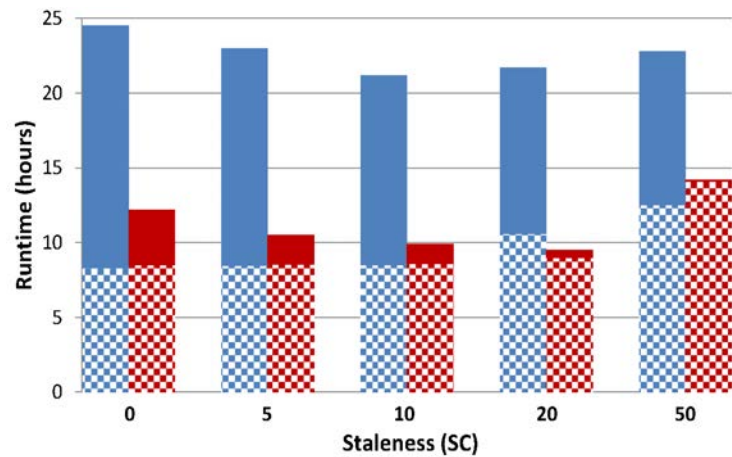
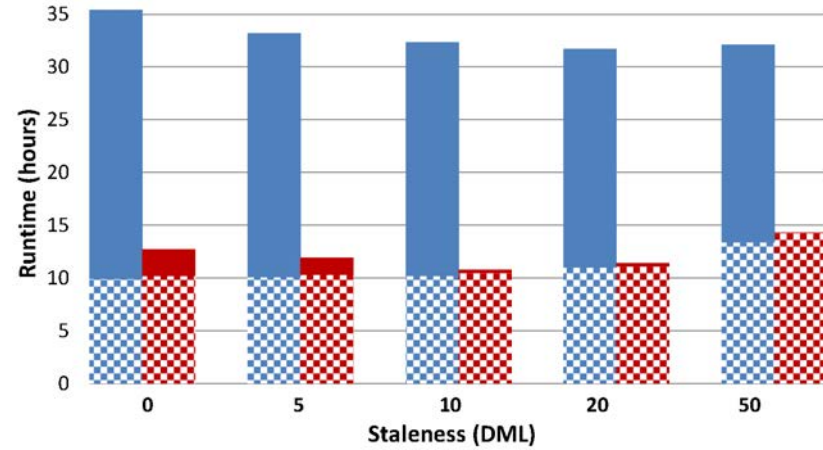
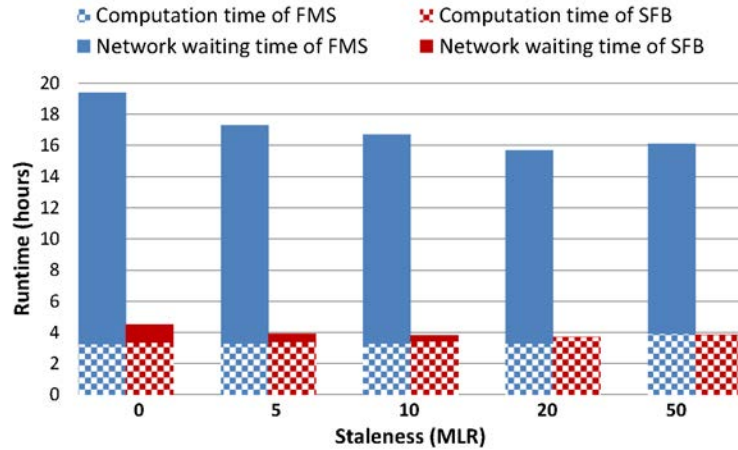
- ✓ Does not need central parameter server or key-value store
- ✓ Works with SSP bridging model (staleness = s)

SF: Convergence Speedup



- Convergence time versus model size, under BSP
- FMS = full matrix updates; SFB = sufficient factor updates

SF: Comm.-Time Reduction



- Computation vs network waiting time
- FMS = full matrix updates; SFB = sufficient factor updates

Summary

1. How to Distribute?

- Structure-Aware Parallelization
- Work Prioritization

2. How to Bridge Computation and Communication?

- BSP Bridging Model
- SSP Bridging Model for Data and Model Parallel

3. How to Communicate?

- Managed comms – interleave comms/compute, prioritized comms
- Parameter Storage: Centralized vs Decentralized
- Communication Topologies: Master-Slave, P2P, Halton Sequence

4. What to Communicate?

- Full Matrix updates
- Sufficient Factor updates
- Hybrid FM+SF updates (as in a DL model)



In Closing: A Distributed Framework for Machine Learning

ML computation can be handled more effectively and economically on a different system architecture

ML Algorithm behavior is different from traditional computing



Flexible and does not need traditional database-style precision



Opportunity for dynamic resource reclamation (CPU, GPU, disk, network)

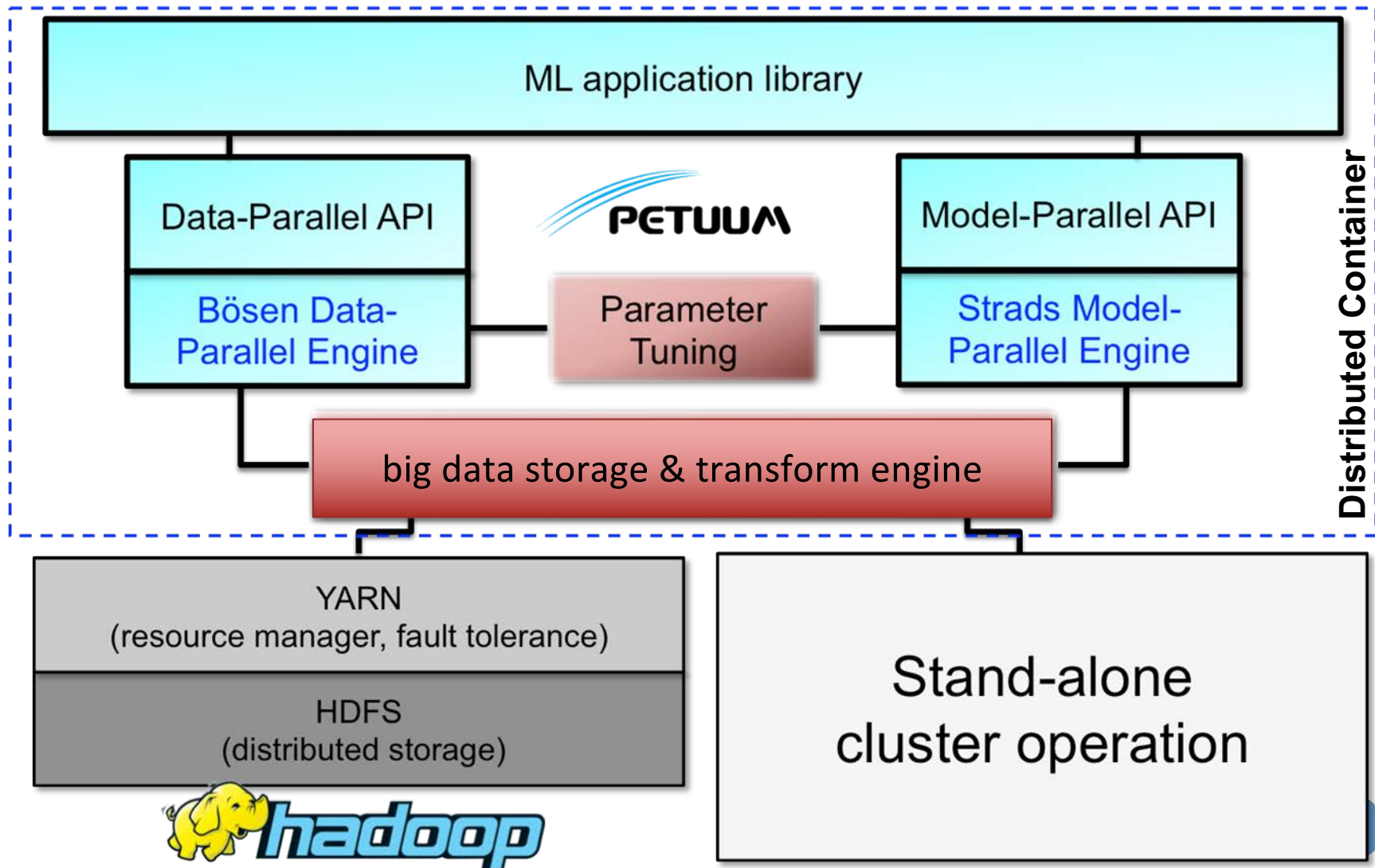


Intelligently-designed workhorse engines can be shared across many ML algorithms

Existing approaches can't take advantage of different AI & ML behavior

- Traditional platforms specialize at supporting database-style workload, incurring expensive error-recovery and network overheads
- Traditional platforms do not perform dynamic resource allocation for fast-completing workloads, wasting CPU ops
- Traditional platforms do not provide sharable workhorse engines, so each vertical application must be developed separately

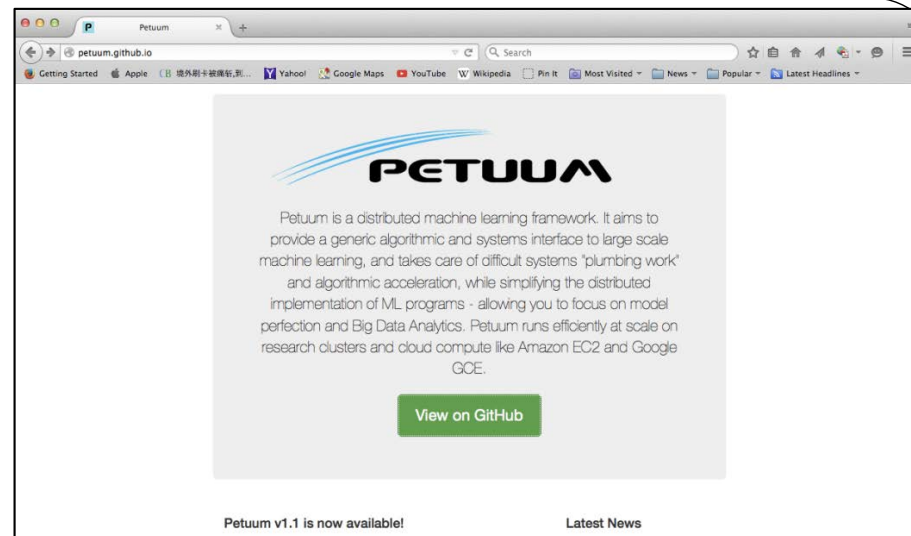
The Petuum Architecture (50,000 feet view)



Major Releases

(petuum.org)

- **Dec 2013: Petuum 0.1**
 - **Initial release**
 - Apps: LDA, matrix factorization
 - System: Bosen (parameter server)
- **March 2014: Petuum 0.2**
 - Apps: LDA, matrix factorization, Lasso
 - System: Strads (model-parallel scheduler)
- **July, 2014: Petuum 0.9**
 - Apps: LDA, matrix factorization, Lasso, Logistic Regression
 - System: large performance improvements
 - Patch releases 0.91 (July 2014), 0.92 (Sept 2014), 0.93 (Dec 2014)
- **Jan 2015: Petuum 1.0**
 - **Many new Apps: MedLDA, NMF, CNN, DML, DNN, DNN speech, Kmeans, MLR, Random forest, Sparse coding**
 - System: more performance improvements
- **July 2015: Petuum 1.1**
 - New Apps: Distributed+GPU CNN, SVM
 - **Big Data Ecosystem Support: Java parameter server (JBosen), HDFS, YARN**



Petuum Speed Advantage

Topic Detection Speed

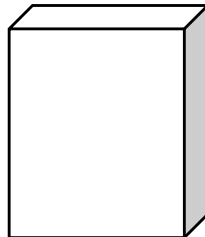
On 128 machines

Spark



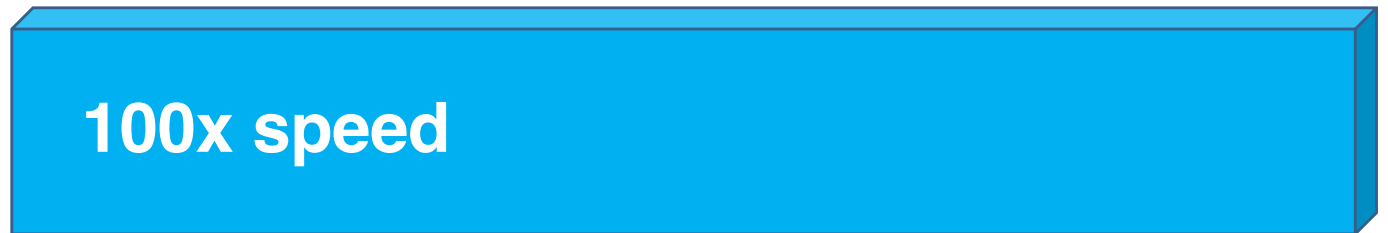
1x speed

Yahoo

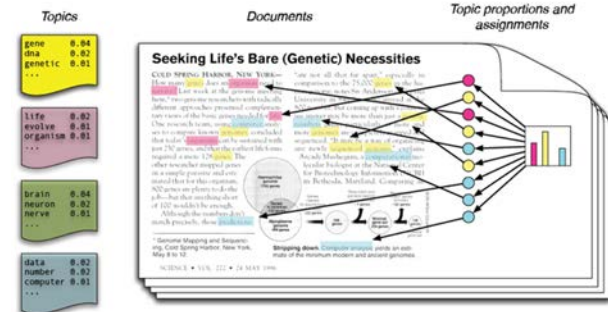


12x speed

Petuum

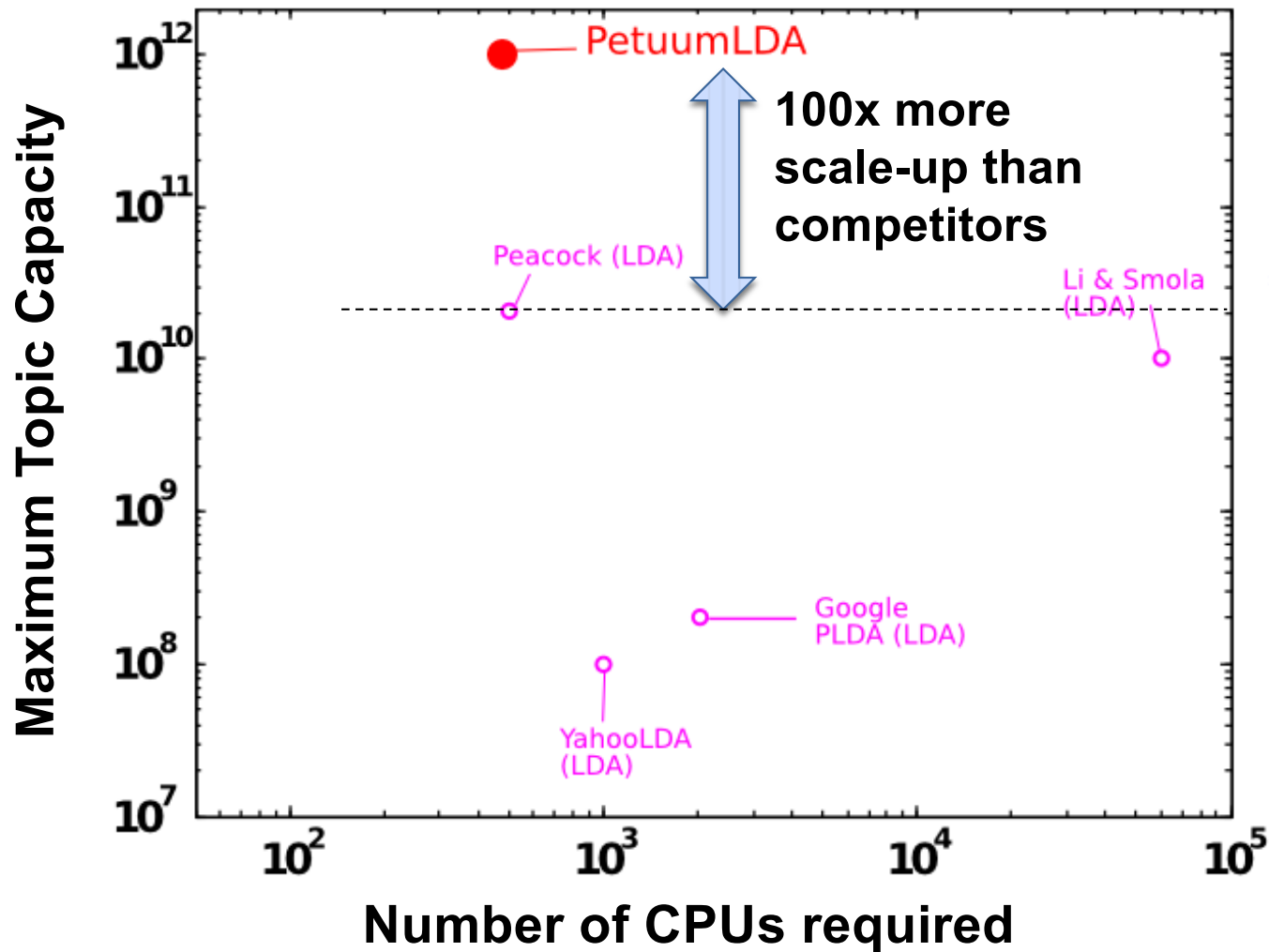


100x speed

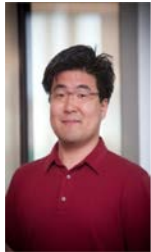


Petuum Size Advantage

Topic Detection Size



Acknowledgements



Jin Kyu Kim



Seunghak Lee



Jinliang Wei



Wei Dai



Pengtao Xie



Xun Zheng



Abhimanu Kumar



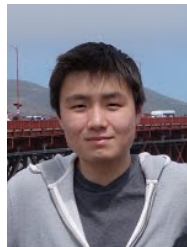
Garth Gibson



Greg Ganger

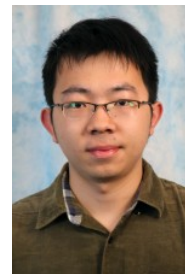


Qirong Ho

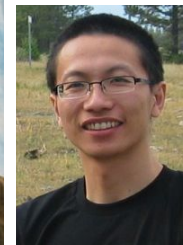


Aurick Qiao

www.sailing.cs.cmu.edu



Hao Zhang



Yaoliang Yu



Phillip Gibbons



James Cipar