Lifting Synchronization Barriers

ECE 826

Dimitris Papailiopoulos

Today

- Stragglers in Synchronous Distributed Optimization

- Lifting Synchronization Barriers

- HogWild!

Stochastic Gradient Descent

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}; \mathbf{z}_i)$$

loss for data point i

- **Idea** ('50s, '60s [Robbins, Monro], [Widrow, Hoff]):

  Sample a data point + locally optimize.

SGD: *An Über-algorithm*

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \gamma \cdot \nabla \ell(\mathbf{w}_k; \mathbf{z}_{i_k})$$

Stochastic Gradient Descent

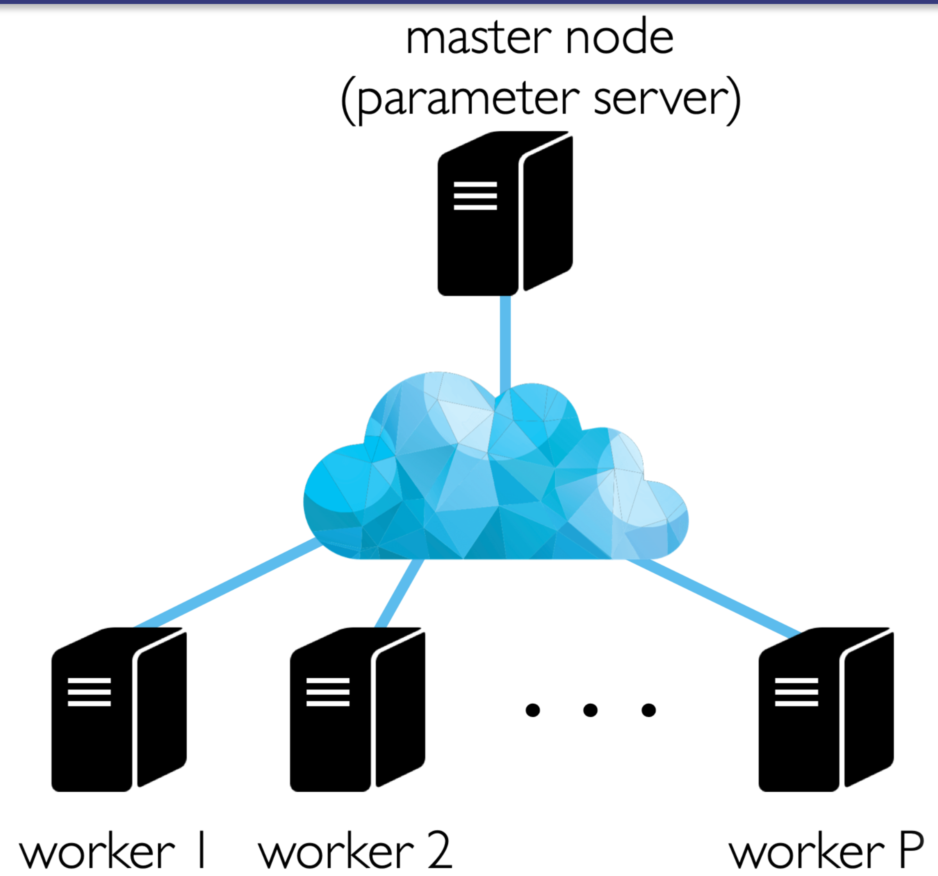
SGD can take years on large nlp models even on a single high end GPU

Goal:

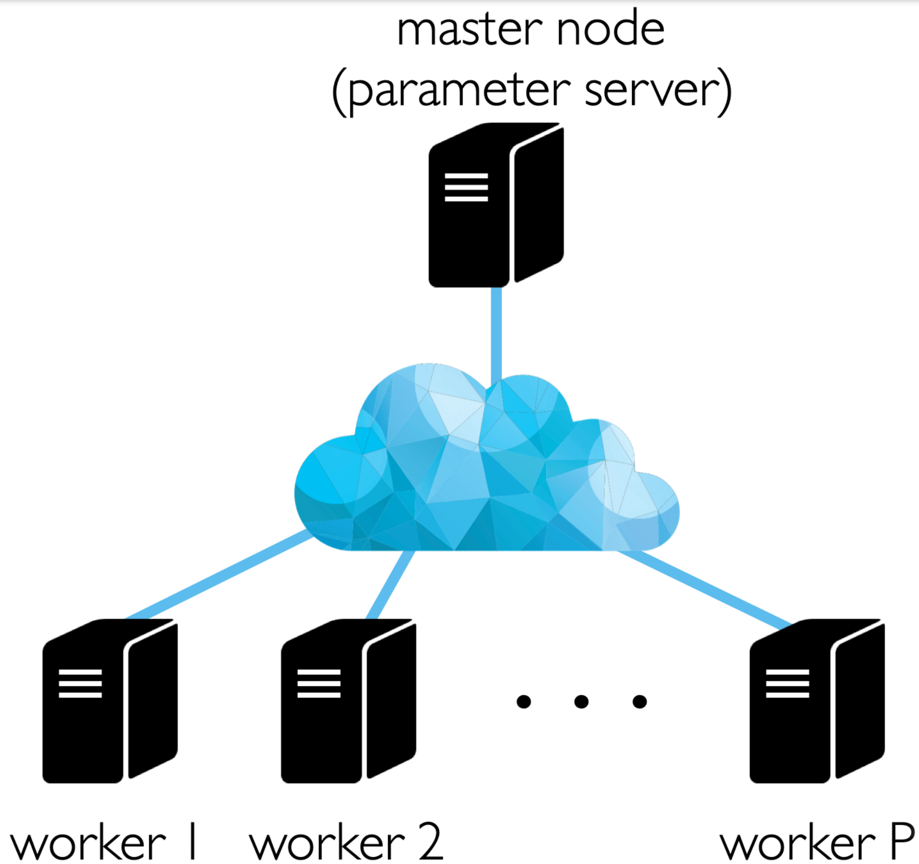
Speed up Machine Learning

Scaling Up SGD

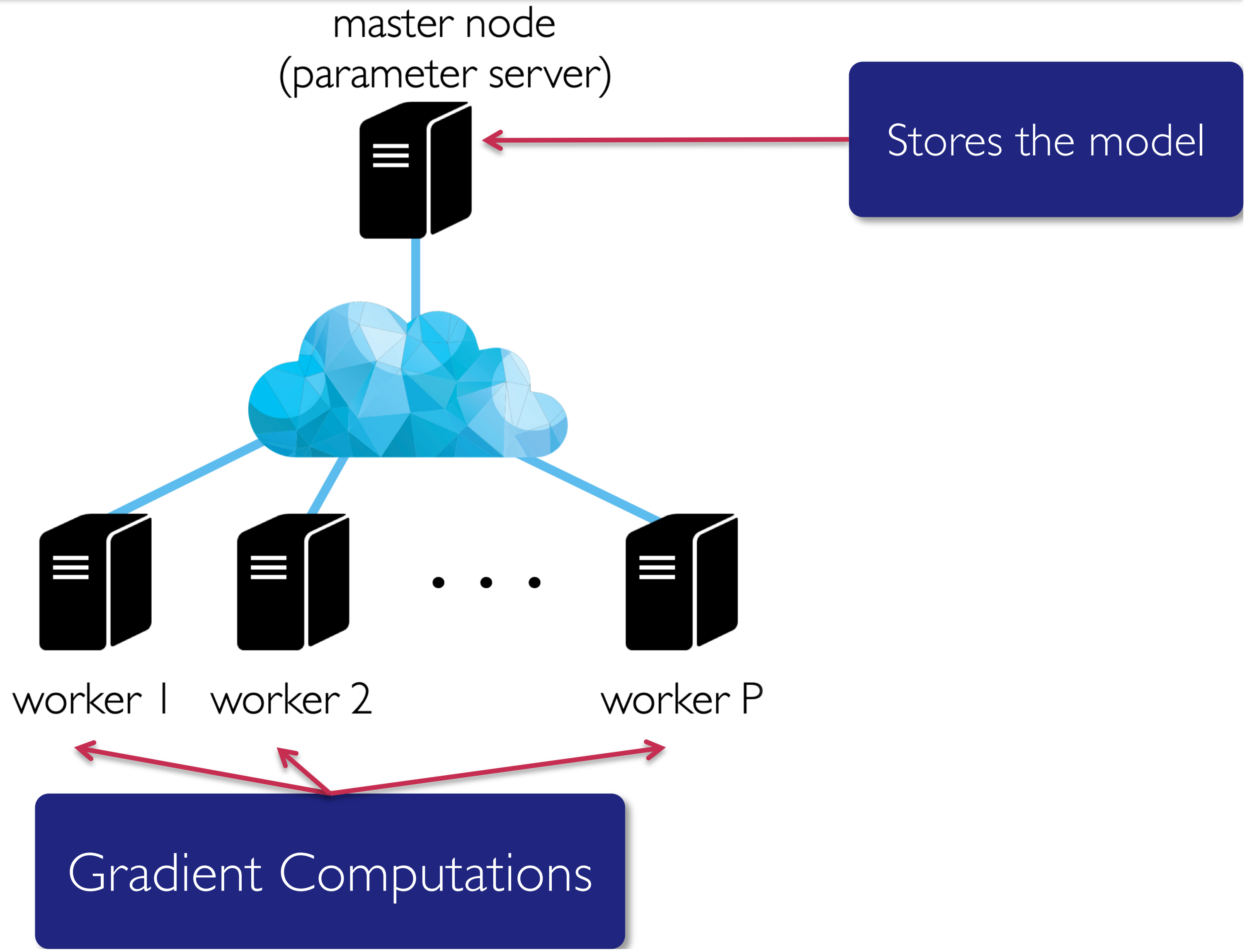
Synchronous computation



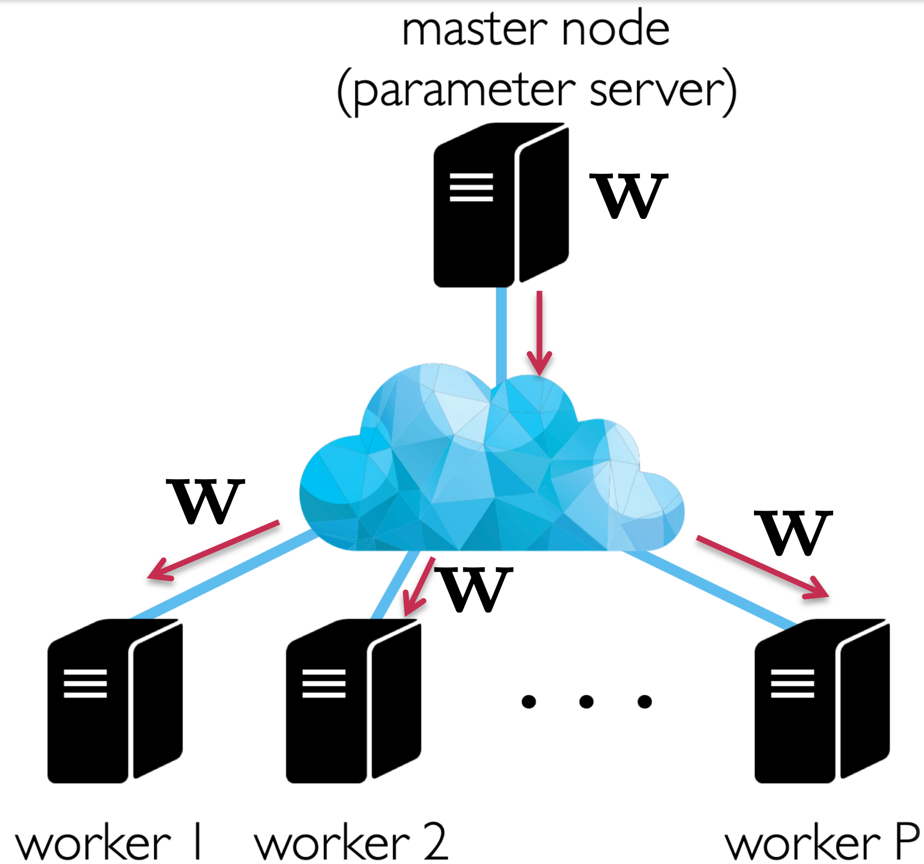
Algorithm of choice: minibatch SGD



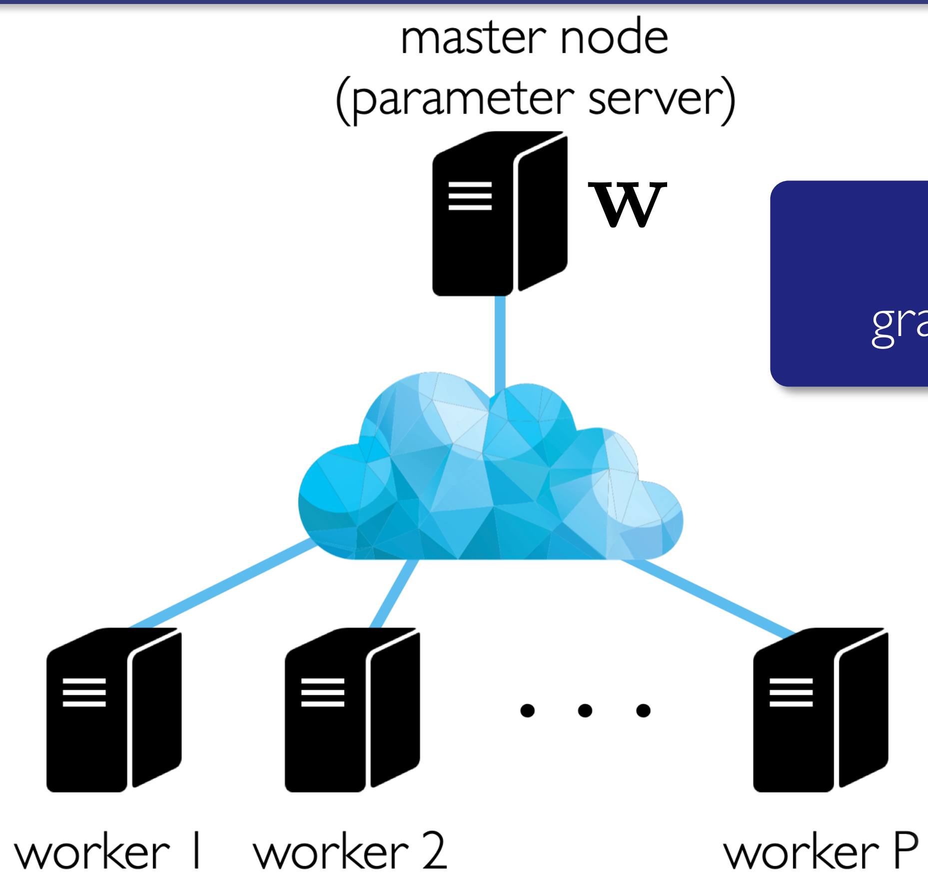
Algorithm of choice: minibatch SGD



Algorithm of choice: minibatch SGD



Algorithm of choice: minibatch SGD

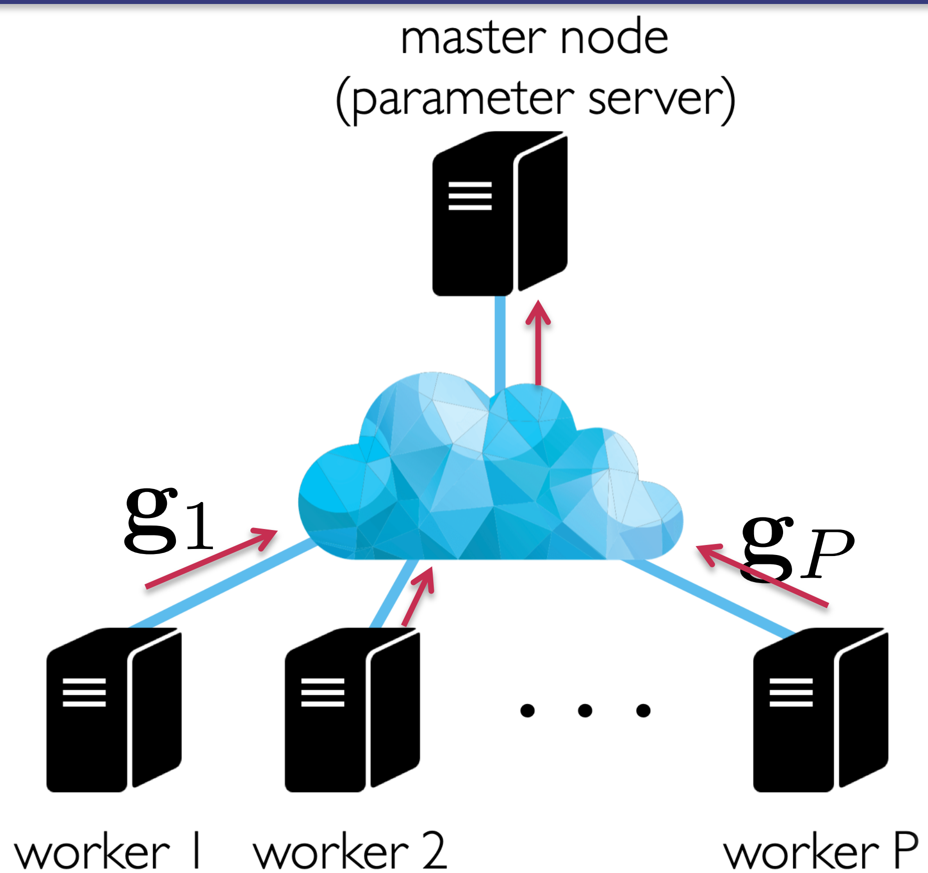


B = batch size,
gradients / iteration

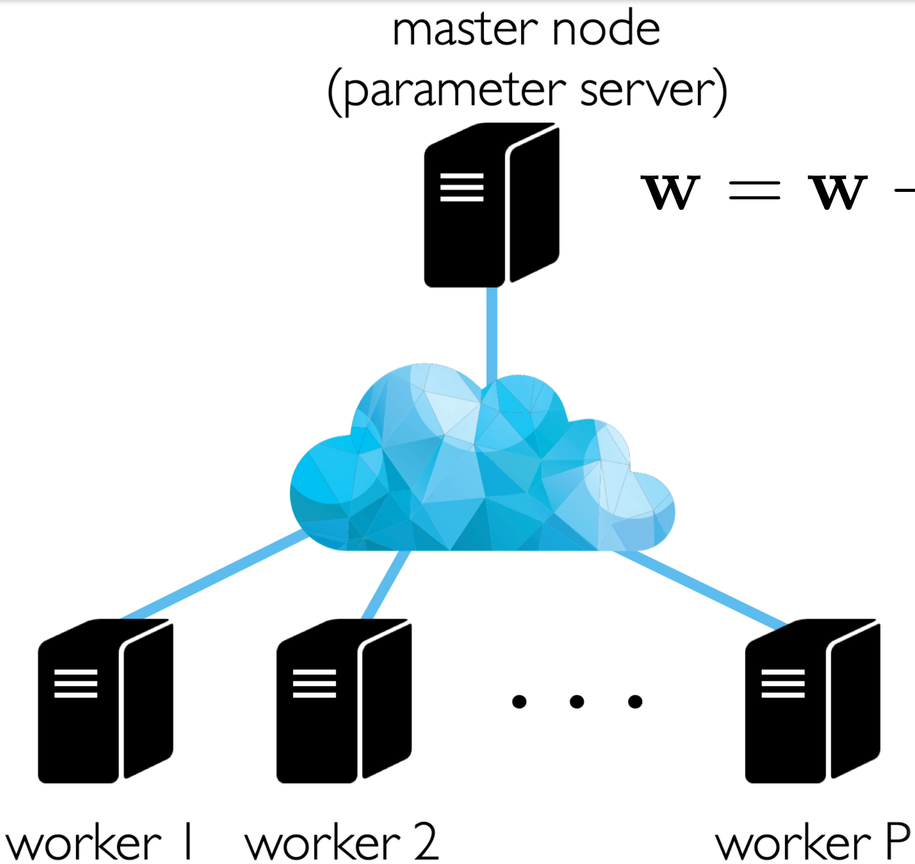
$$\mathbf{g}_1 = \sum_{i \in \mathcal{S}_1} \nabla \ell(\,(\mathbf{x}_i, y_i); \mathbf{w}\,)$$

$$\mathbf{g}_P = \sum_{i \in \mathcal{S}_P} \nabla \ell(\,(\mathbf{x}_i, y_i); \mathbf{w}\,)$$

Algorithm of choice: minibatch SGD

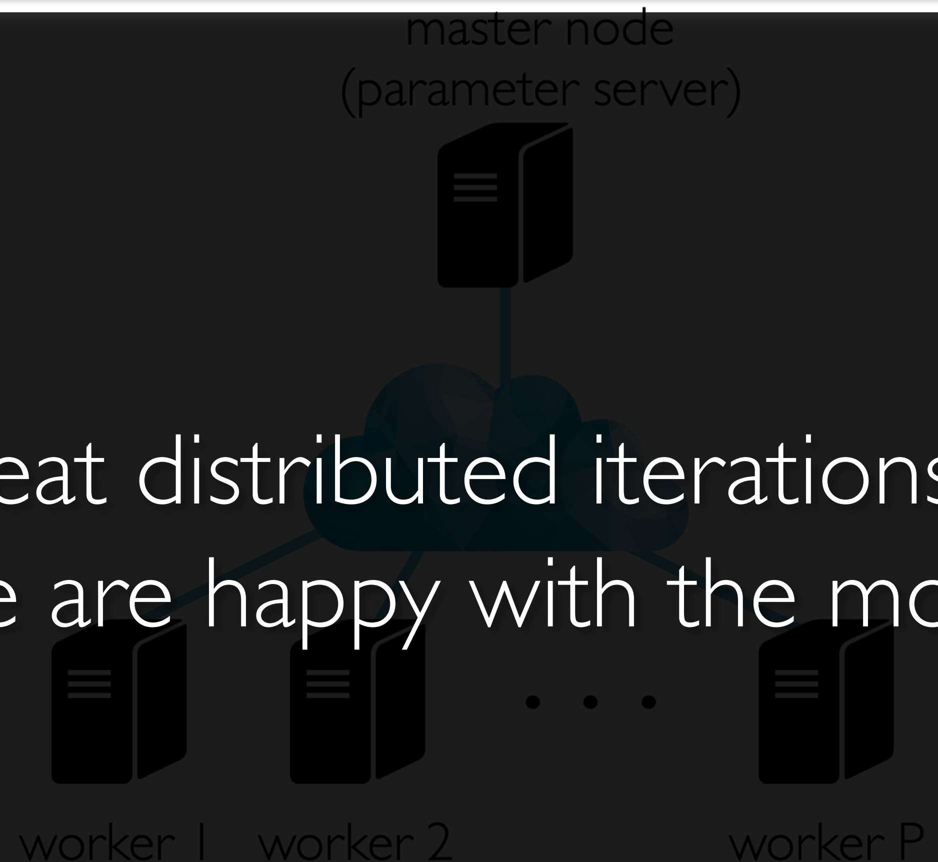


Algorithm of choice: minibatch SGD



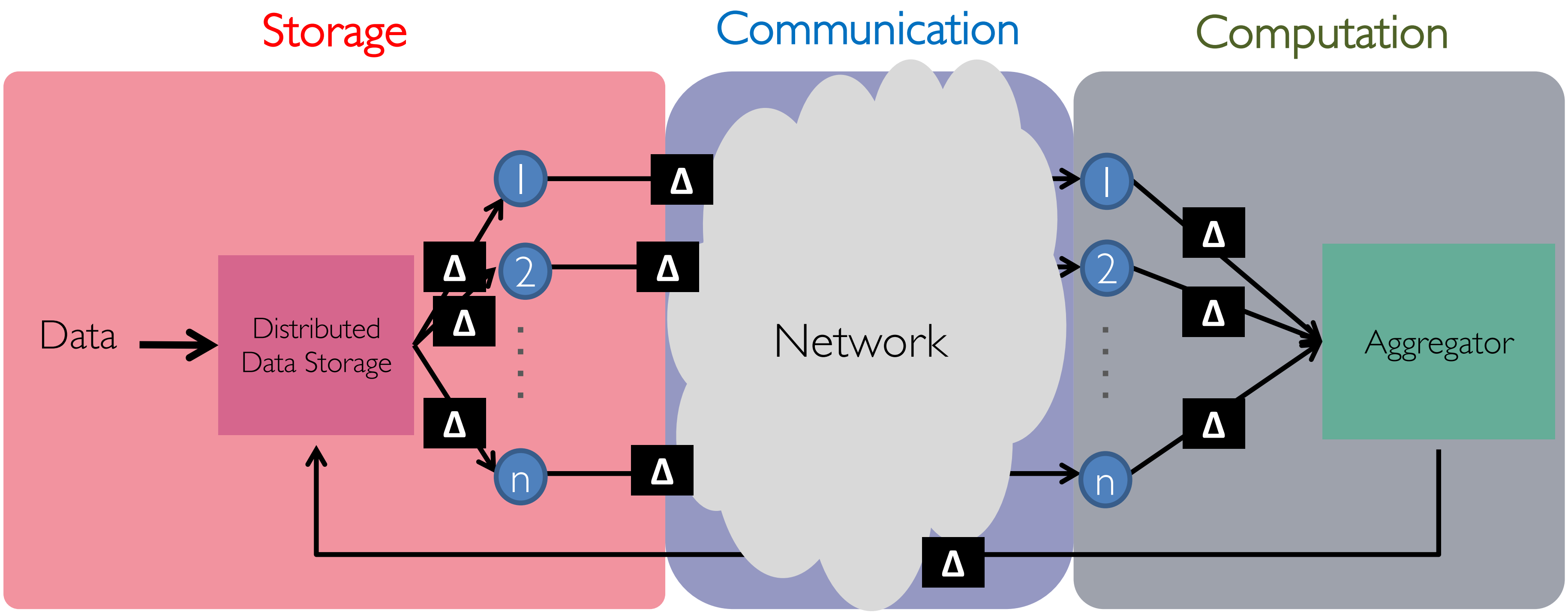
$$\mathbf{w} = \mathbf{w} - \gamma \sum_{i=1}^{P} \mathbf{g}_i$$

Algorithm of choice: minibatch SGD



Repeat distributed iterations until we are happy with the model

Large-scale Distributed Machine Learning Systems



"The scale and complexity of modern Web services make it ***infeasible*** *to eliminate all latency variability."*

Jeff Dean, Google.

Stragglers

- Ideal compute time per node ~ O(total_time/P)
- But there is a lot of randomness:
  - Network/Comm Delays
  - Node/HW Failures
  - Resource Sharing

- What if time per node is a random variable:

  $$X = \text{constant} + \text{Exp}(\lambda)$$

Lemma:

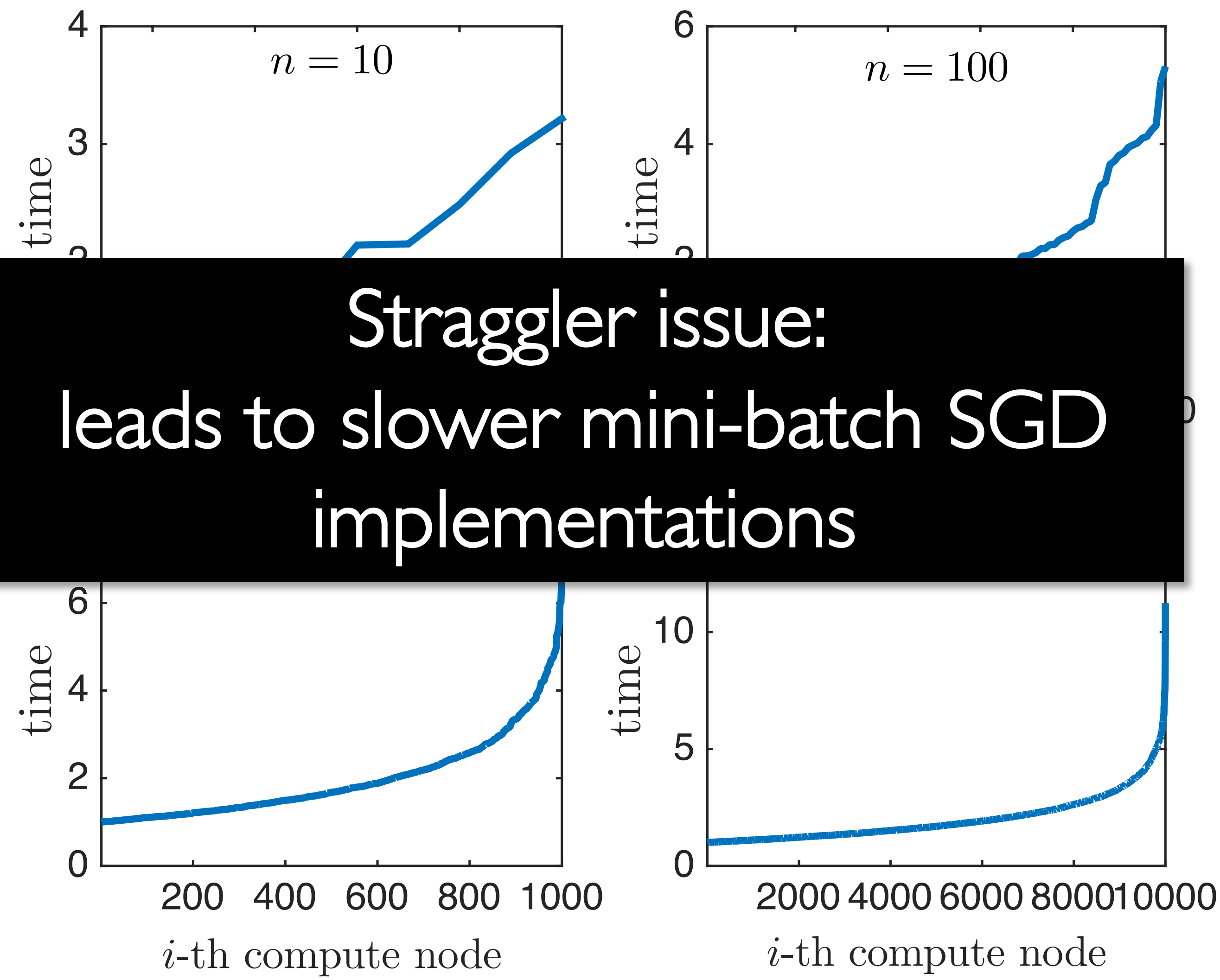
$$\mathbb{E}\{X_{(i)}\} = 1 + \frac{1}{\lambda} \sum_{n-i+1}^{n} \frac{1}{i}$$

Remark:

Slowest node is log(n) times slower than fastest

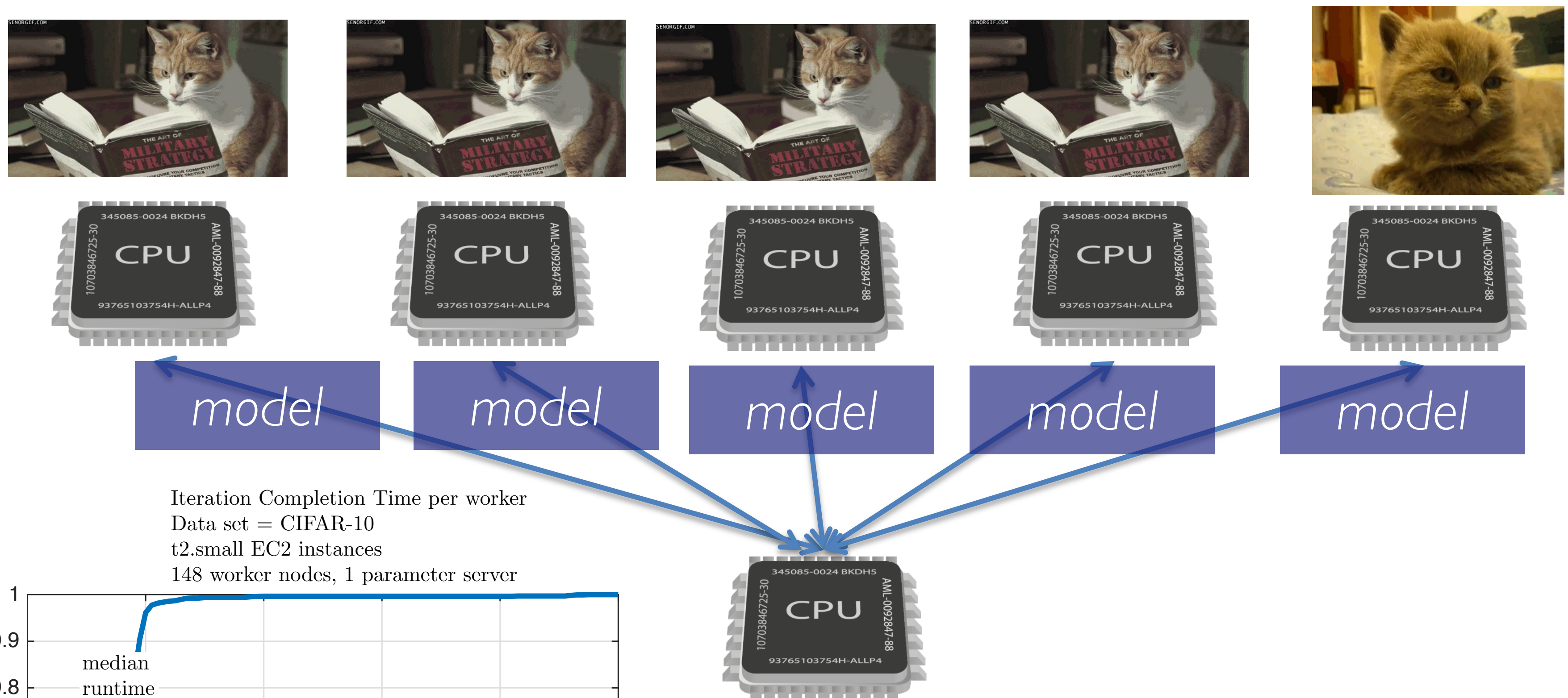
Simulation

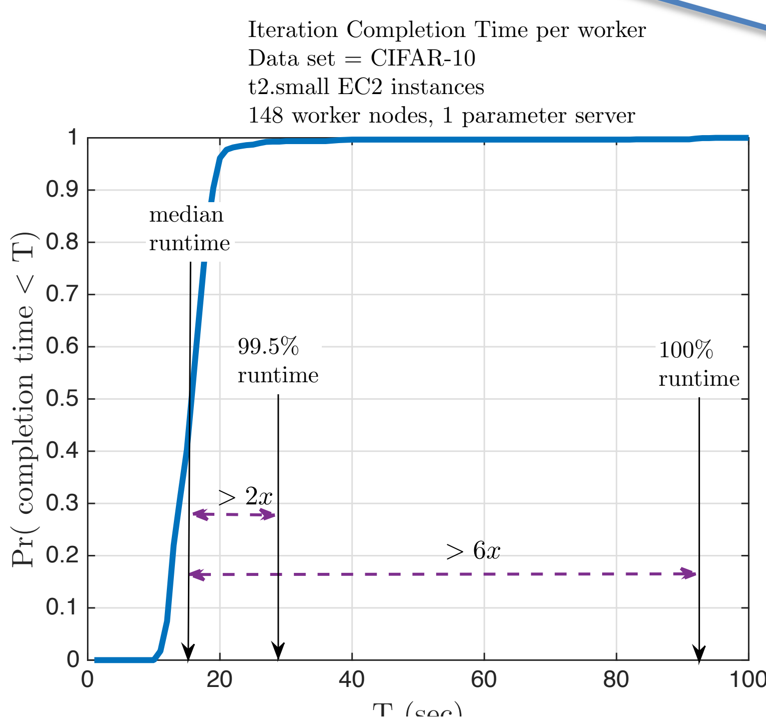
- X(t) = 1+ Exp(0.5), n = 10, 100, 1000, 1000



Straggler issue:
leads to slower mini-batch SGD implementations

Bottleneck: Straggling Learners

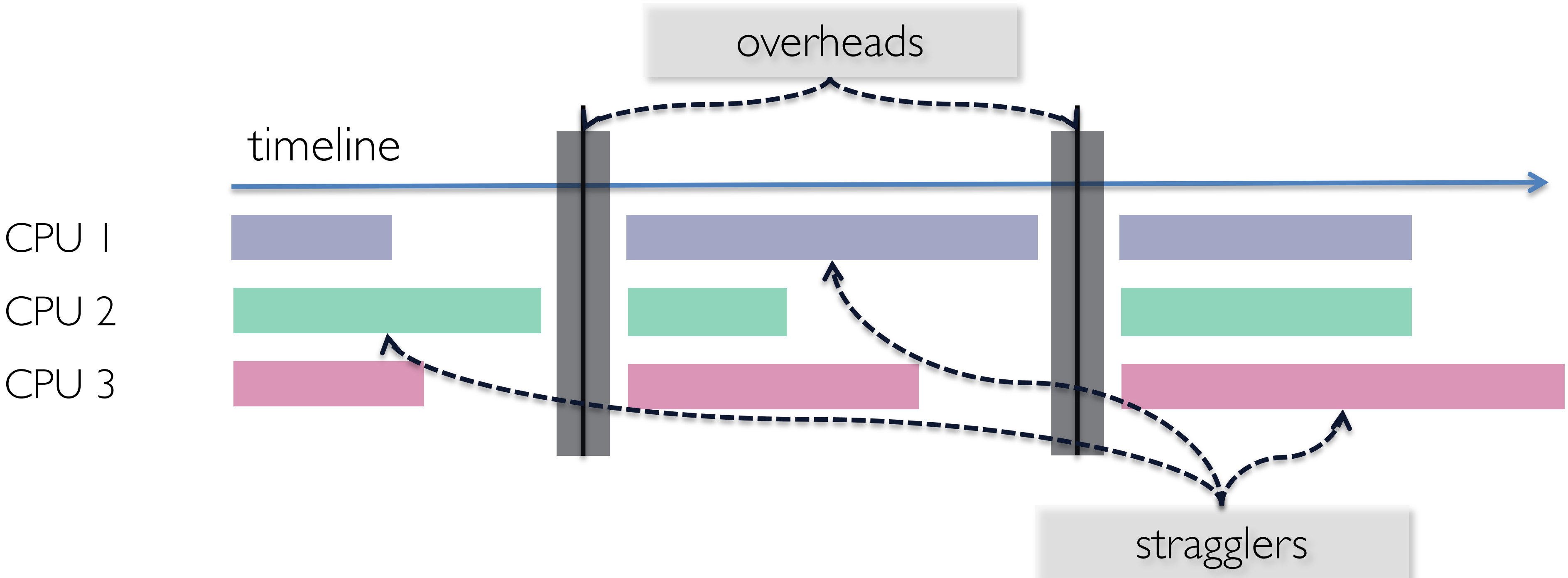




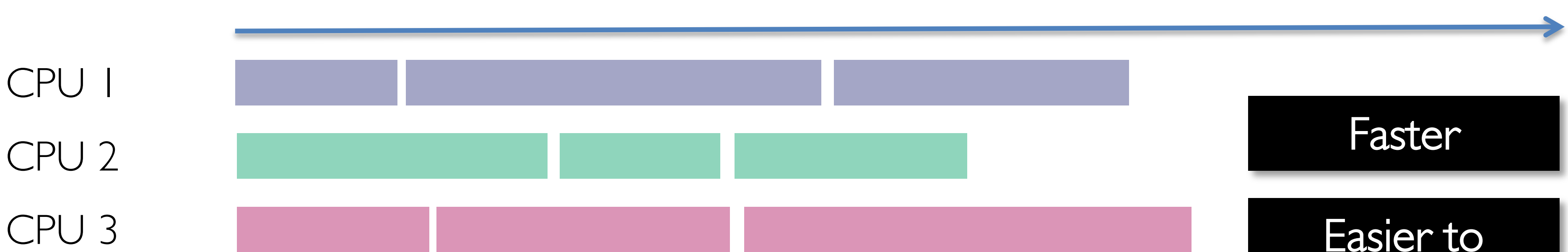
Can we "robustify" distributed ML against stragglers?

Measured on Amazon AWS

A case against Synchronization



Asynchronous World



Faster

Easier to Implement

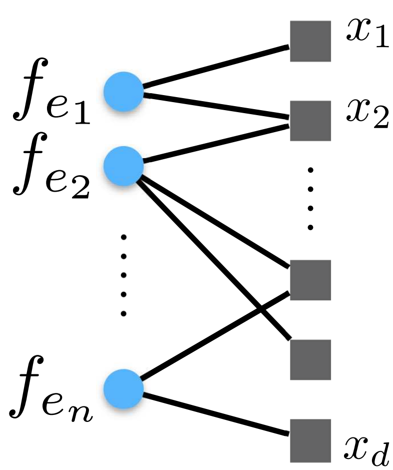
Asynchronous SGD on Sparse Functions

SGD on sparse functions

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

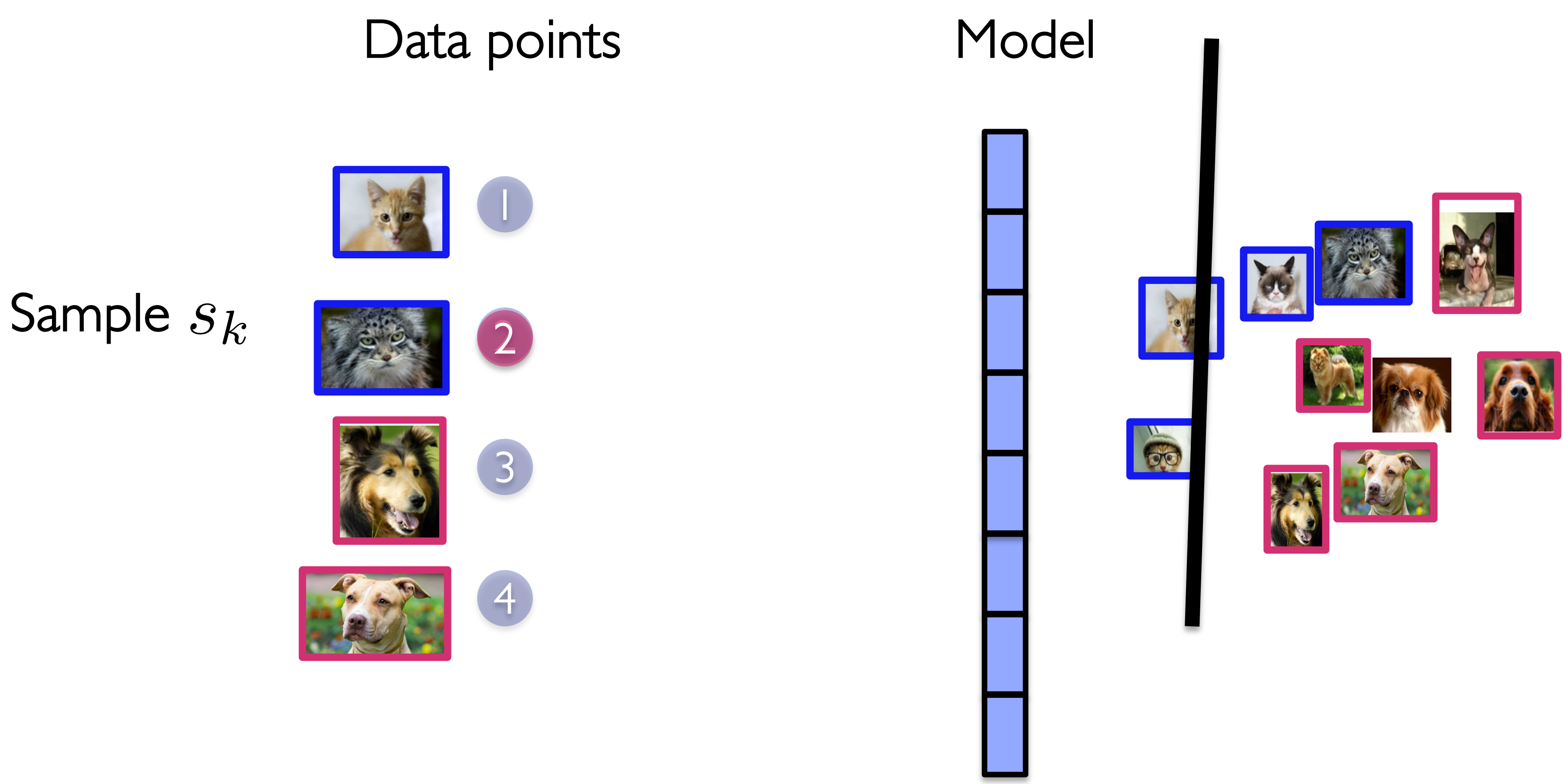
- Def:
  Hyperedge e = the subset of variables that f_e depends on

- The function-variable graph



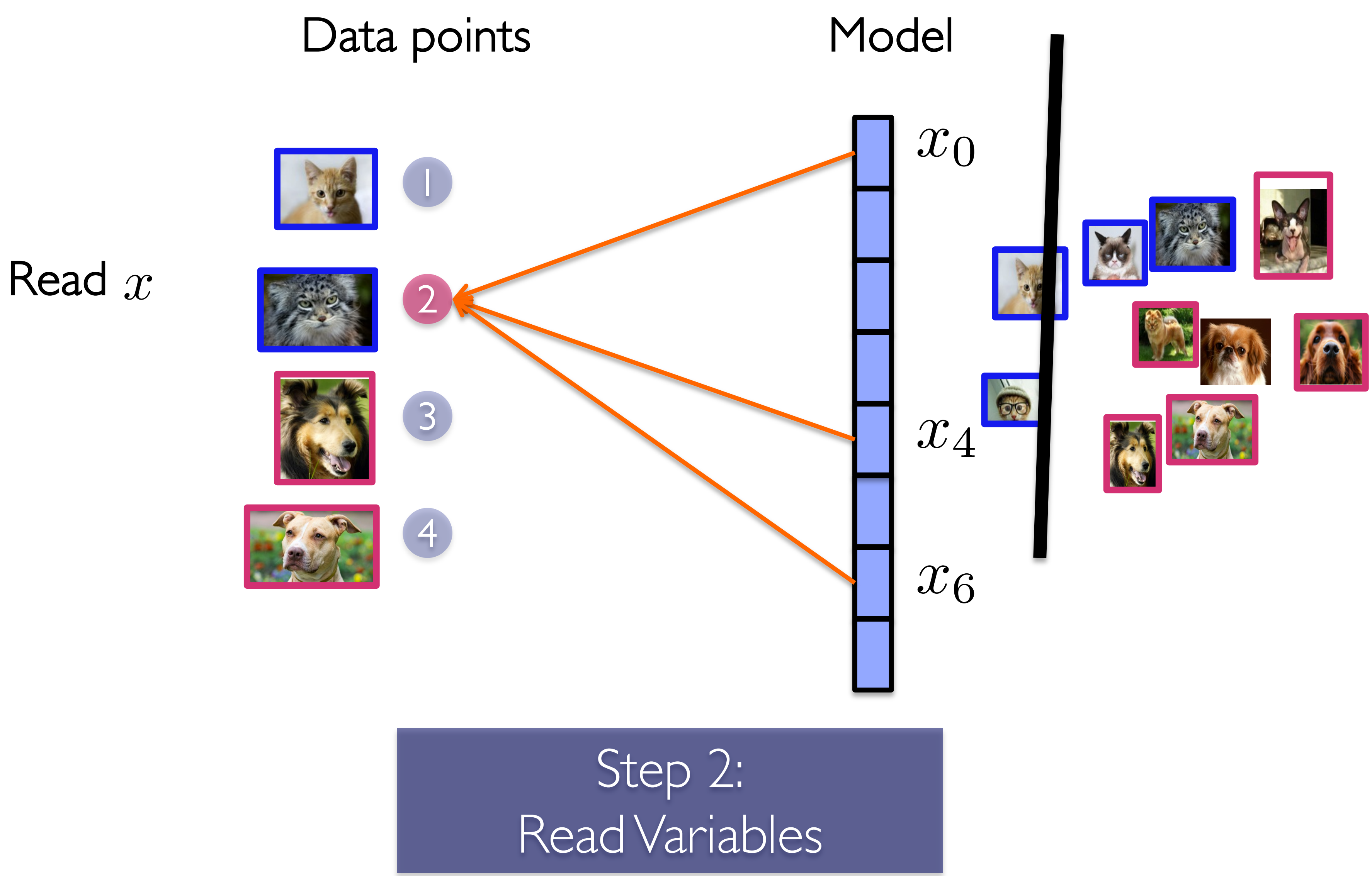
Matrix Fact./Comp.
Graph cuts
Graph/text Classification
Topic Modeling
Dropout
…

SGD on sparse functions



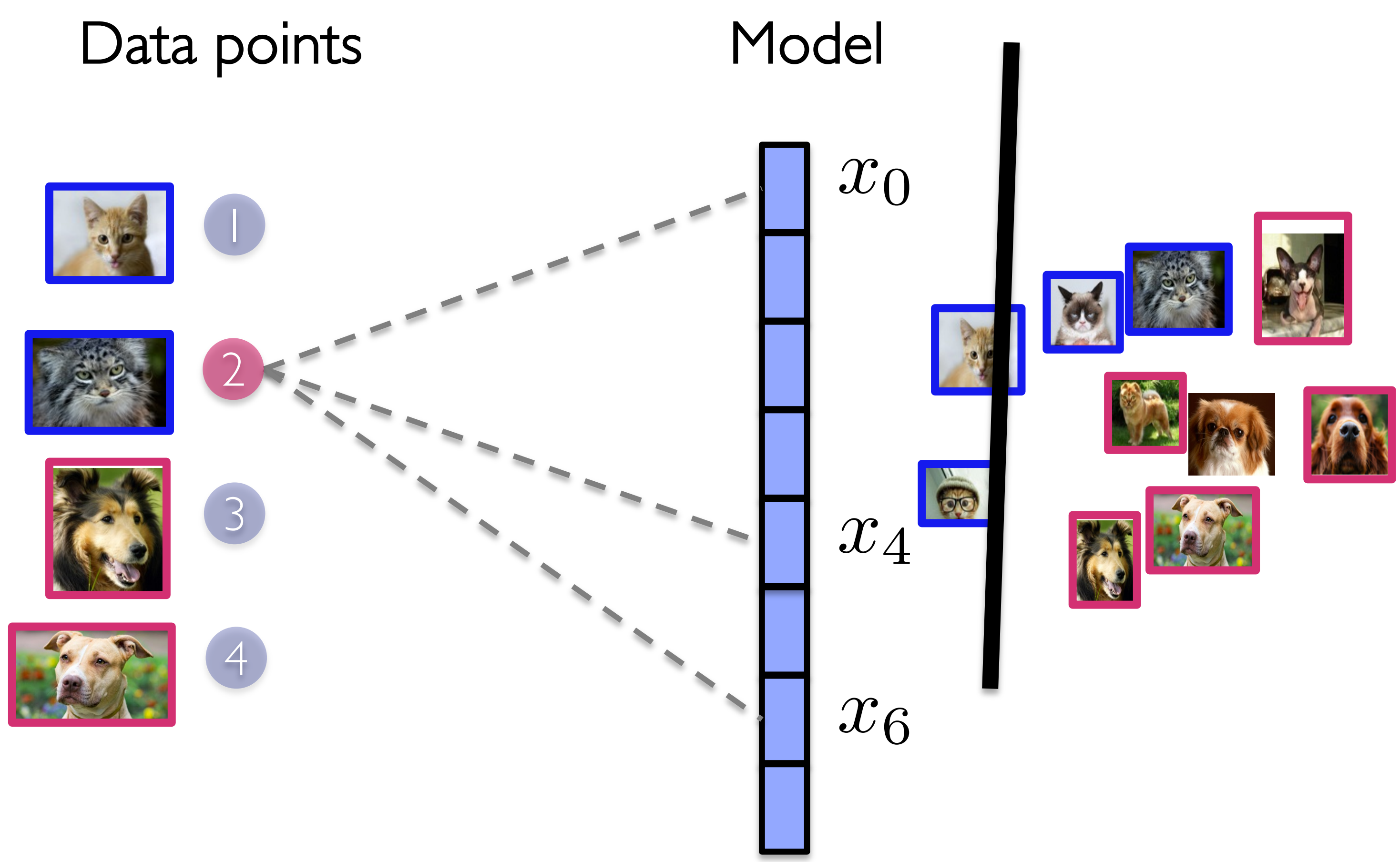
Step 1:
Pick random data point

SGD on sparse functions



SGD on sparse functions

Compute grad.
of local loss
$\nabla f_{s_k}(x)$

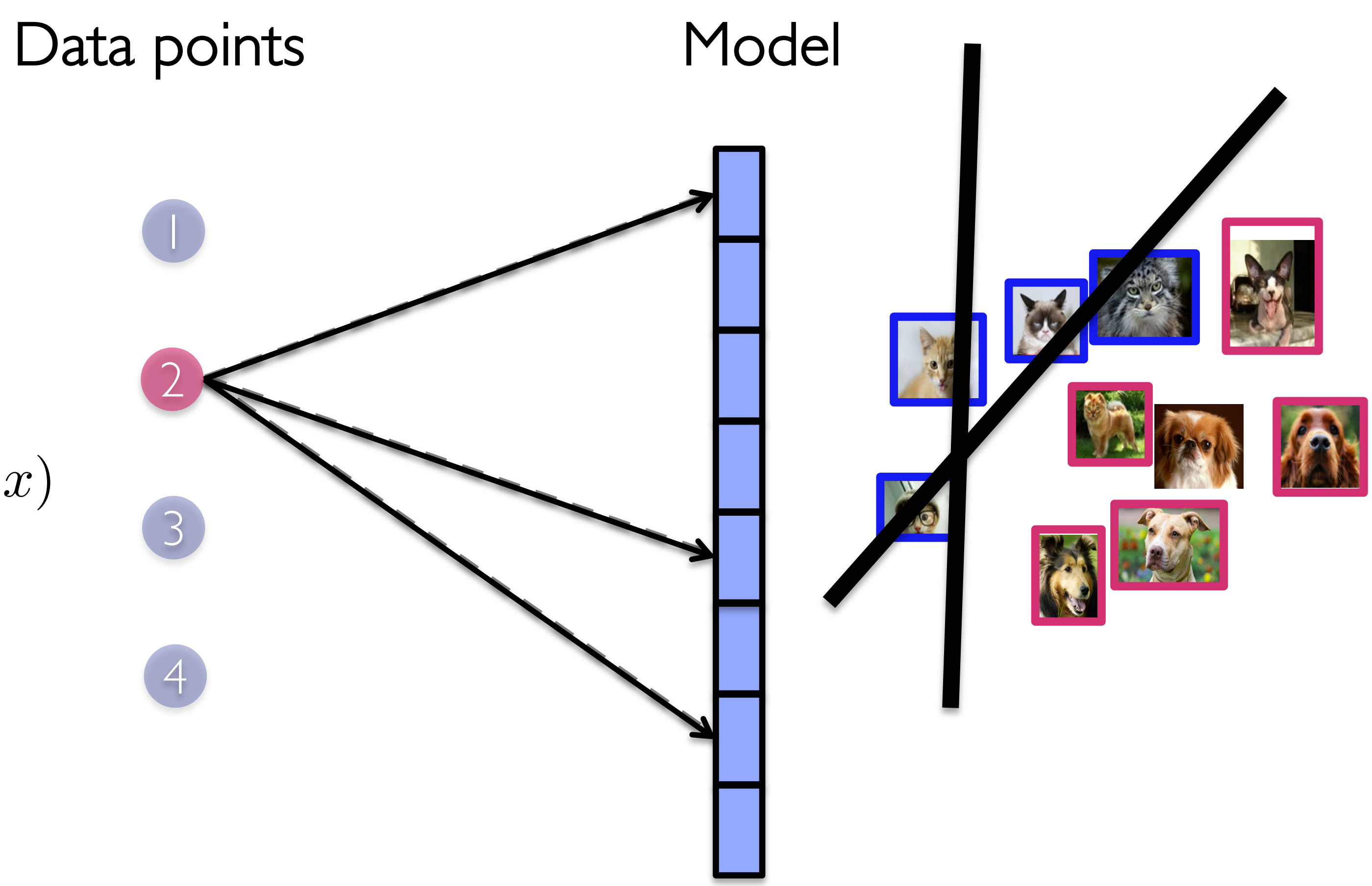


Step3:
Compute local function

SGD on sparse functions

Update Model

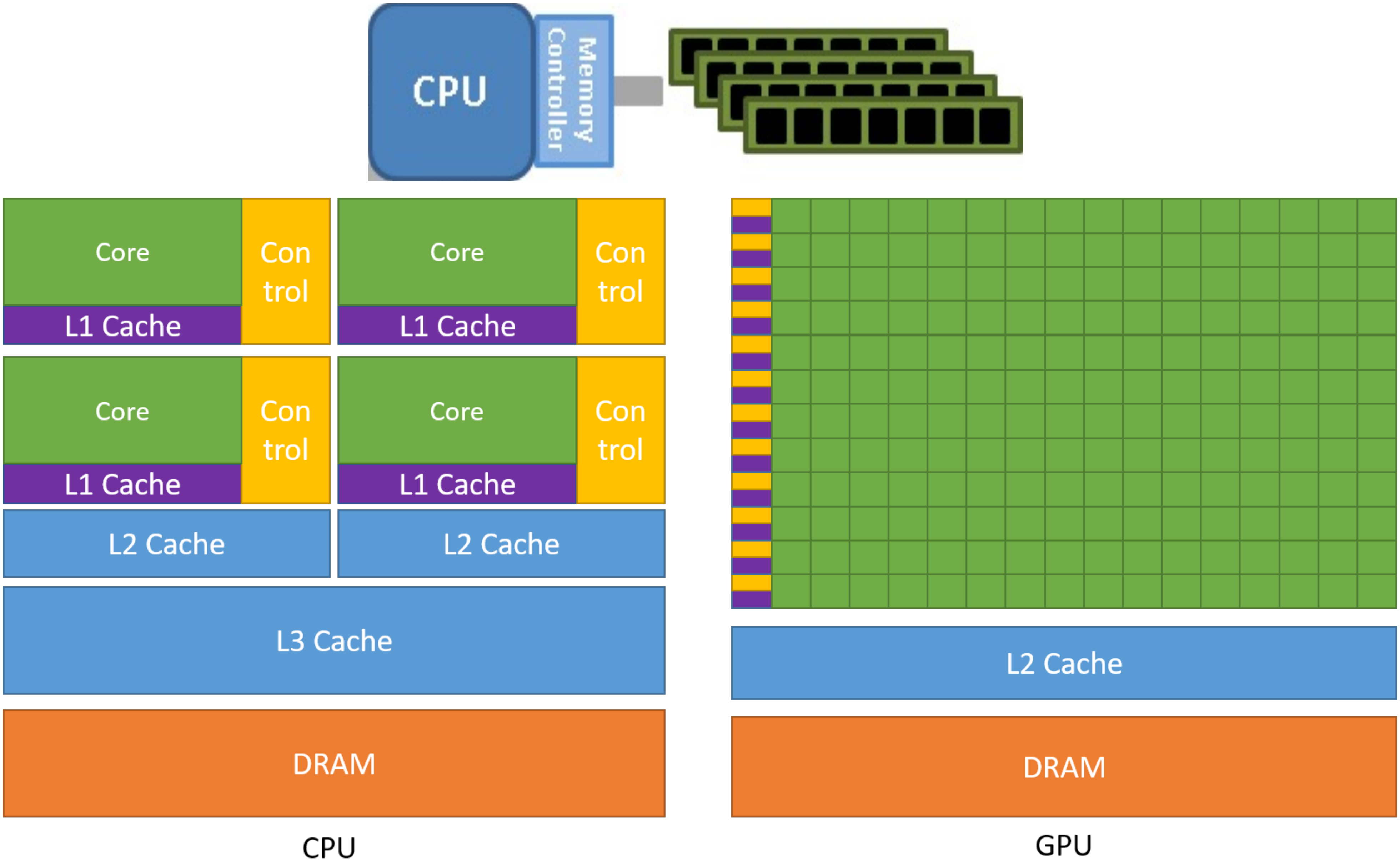
$$x \leftarrow x - \gamma_k \cdot \nabla f_{s_k}(x)$$



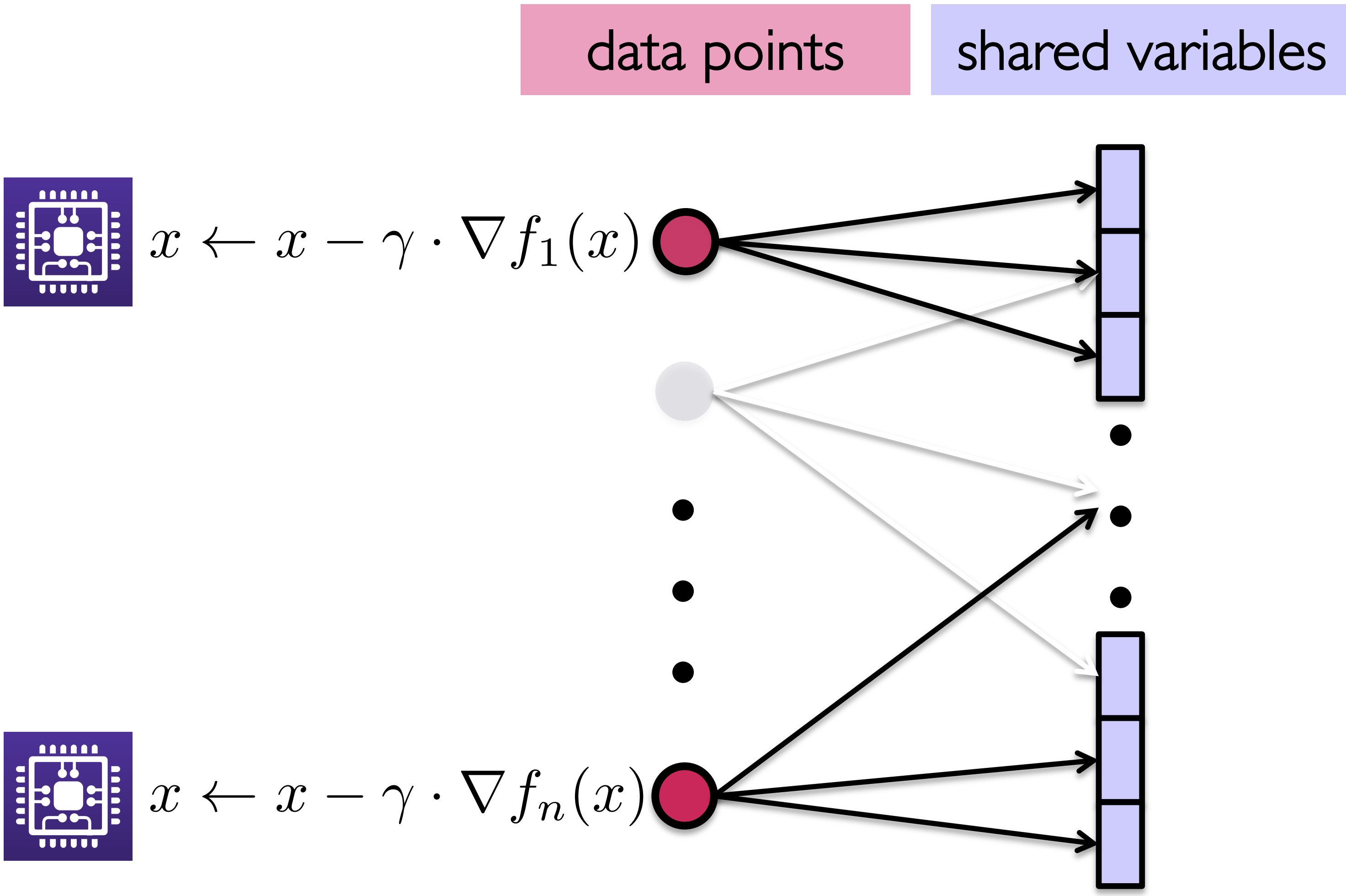
Step4:
Update Model

Parallelizing Sparse SGD on shared memory architectures

Single Machine, Multi-core

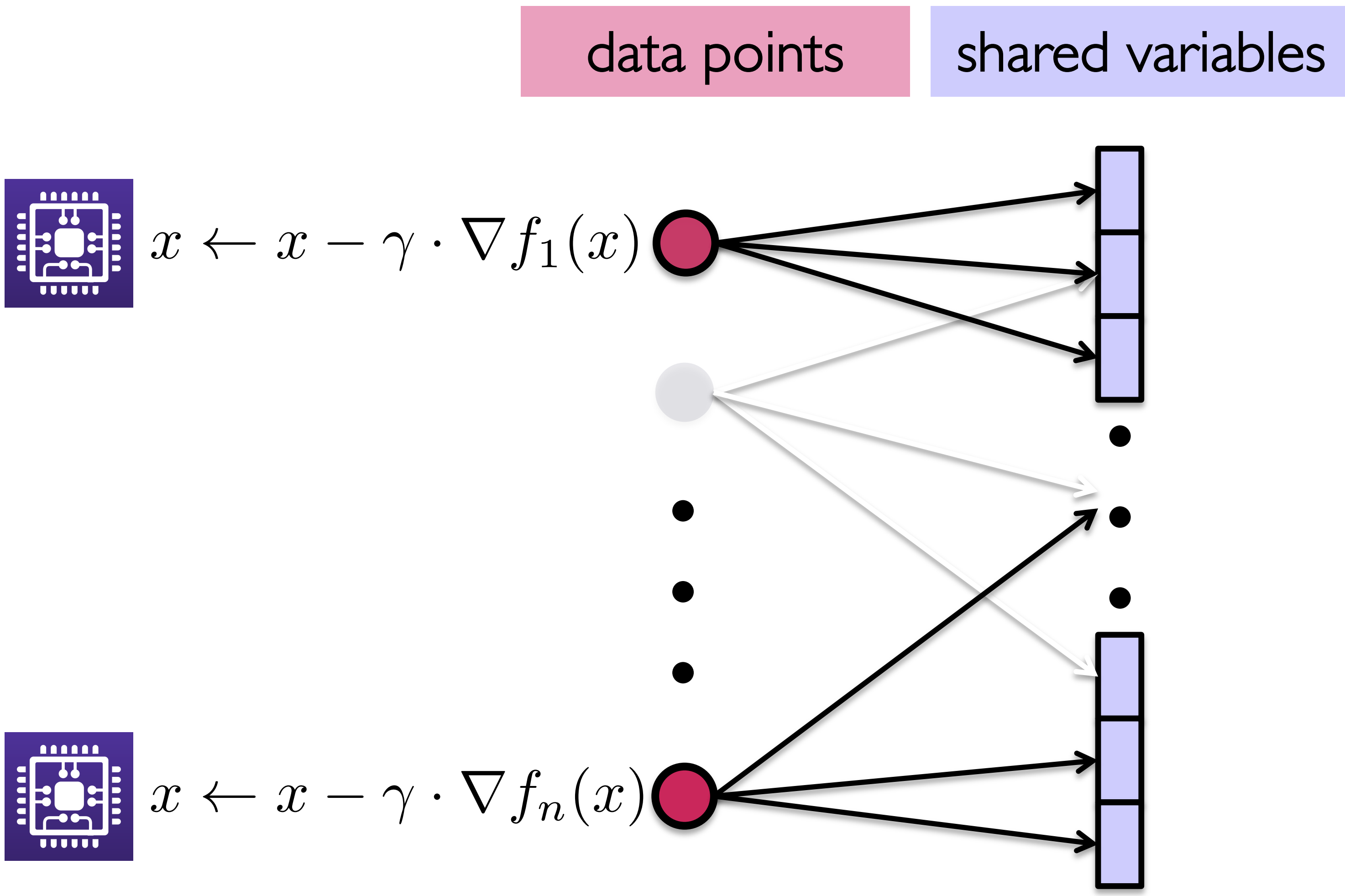


Challenges in Parallel SGD



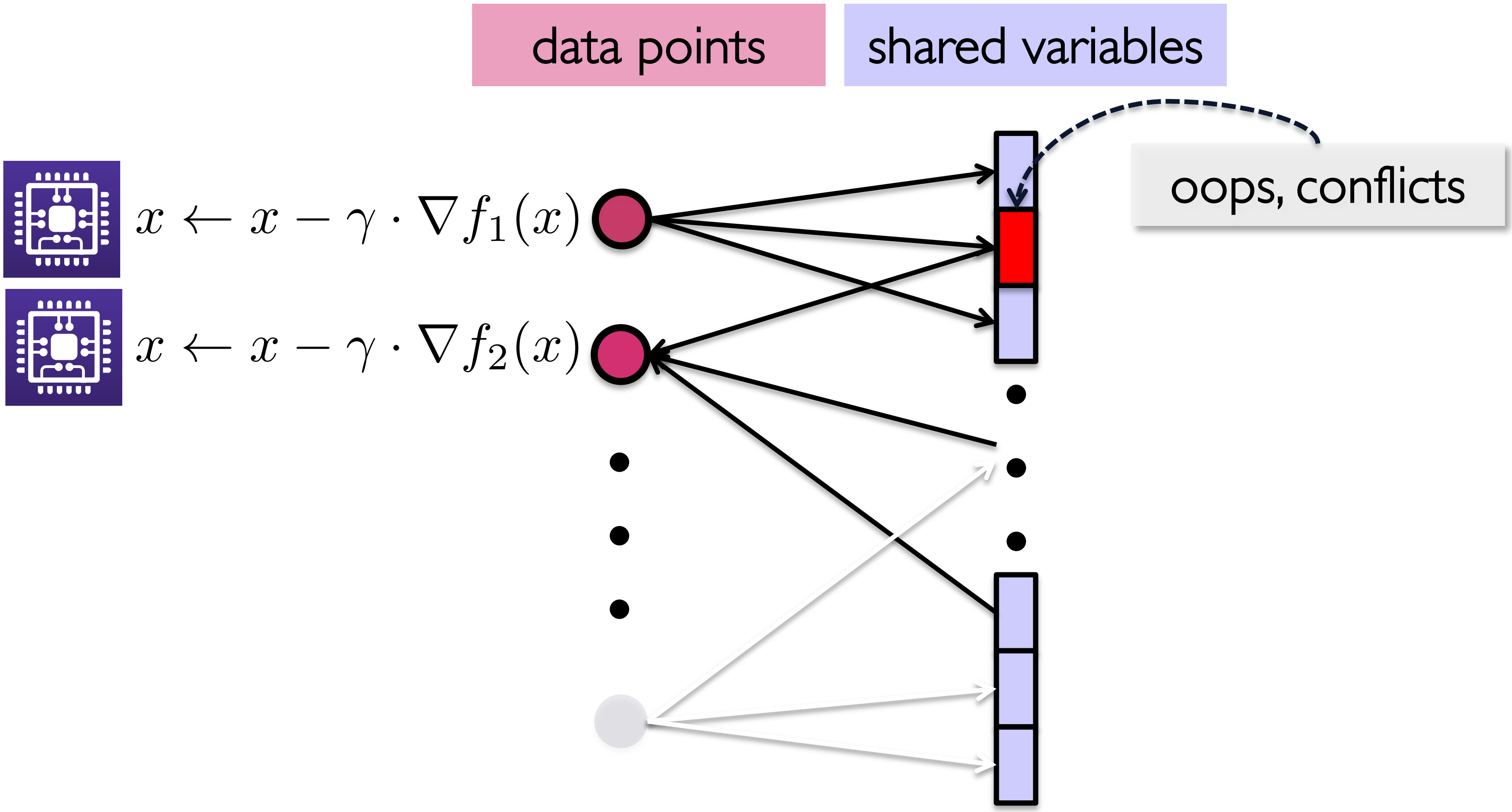
No conflict =>
2 parallel iterations = 2 serial iterations

Challenges in Parallel SGD



No conflict => Speedup

Challenges in Parallel SGD



What should we do for conflicts?

Approach 1: *Coordinate or Lock*

Approach 2: *Don't Care (Lock-free Async.)*

Prior to 2011 Work

Long line of theoretical work since the 60s
[Chazan, Miranker, 1969]

Foundational work on Asynchronous Optimization
Master/Worker model [Tsitsiklis, Bertsekas, 1986, 1989]

Recent hardware/software advances renewed the interest
Round-robin approach [Zinkevich, Langford, Smola, 2009]
Average Runs [Zinkevich et al., 2009],
Average Gradients [Duchi et al, Dekel et al. 2010]

Many based on "Coordinate" or "Lock" approach

Why Coordinate or Lock?

Issue: Synchronization and comm. overheads

HOGWILD! 2011

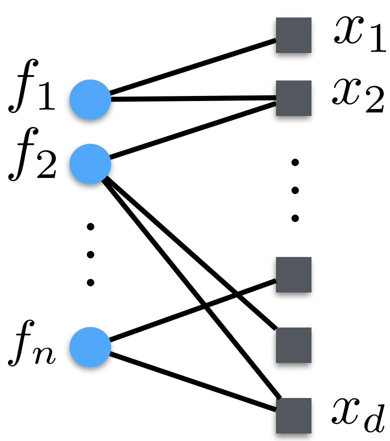
"Run parallel lock-free SGD without synchronization"


Niu


Recht


Ré


Wright



Each processor in parallel

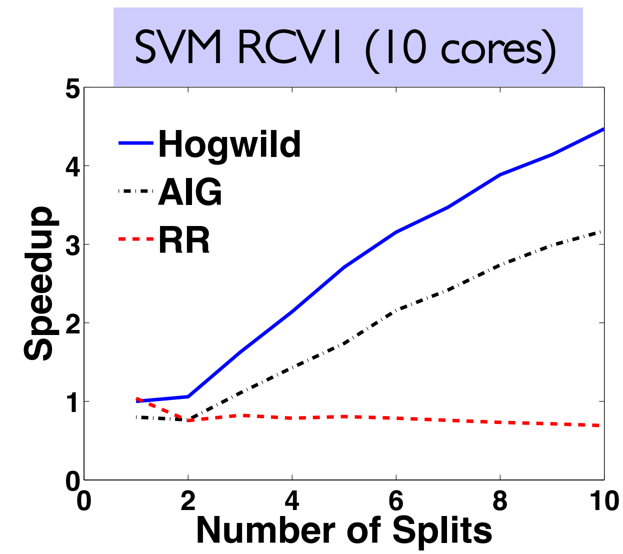
sample function f_i

$x =$ read shared memory

$g = -\gamma \cdot \nabla f_i(x)$

for v in the support of f **do**

$x_v \leftarrow x_v + g_v$



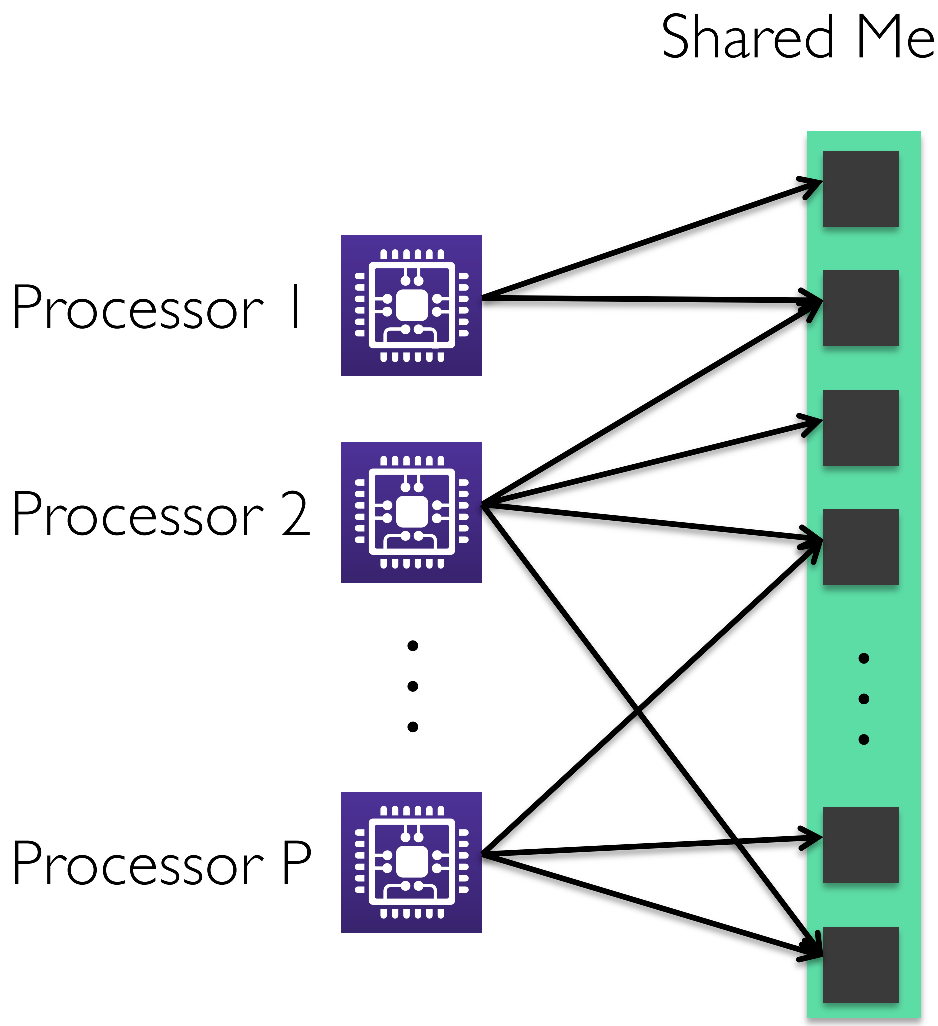
<u>Impact</u>

Google Downpour SGD, Microsoft Project Adam *use HOGWILD!*

Renewed interest on async. optimization

Challenges in Analysis

Challenges in Hogwild!



Issues:
1. updates can be old
2. results can overwritten

Incompatible with classic SGD analysis

How to Analyze Hogwild?

- Measure of performance

$$\text{worst case speedup} = \frac{\text{bound on \#iter of SGD to } \epsilon}{\text{bound on \#iter of Parallel SGD to } \epsilon}$$

Goal of a Hogwild Analysis

Prove that **Parallel SGD** and **Serial SGD** have similar convergence rates for given number of samples

Assumption:
random sampling of gradients yields a nearly optimal load balance
(if number of cores not too many)

How to Analyze Hogwild?

- [Niu, Recht, Re, and Wright, 2011] the first analysis of Hogwild! *Issues:*

  - *many impractical assumptions*
  - *simplified read/write model*
    [consistent reads, single coordinate updates, …]
  - *lengthy derivations*

- Many Async. algorithms follow using similar assumptions, and/or analysis:
  [Duchi et al, 2011], [Liu et al, 2014, 2015], [Avron et al. 2014],
  [De Sa et al, 2015], [Lian et al., 2015], [Peng et al., 2015]

How to Analyze?

[redacted]

General Framework for Asynchronous Lock-free Algorithms?

[redacted]

Analyzing Asynchronous Schemes

A Noisy Lens for Asynchronous Algorithms

Main Idea

Noisy viewpoint:

Asynchronous(*Algo.*(INPUT)) $\equiv$ Serial(*Algo.*(INPUT + Noise)

Perturbed Iterate Analysis for Asynchronous Stochastic Optimization

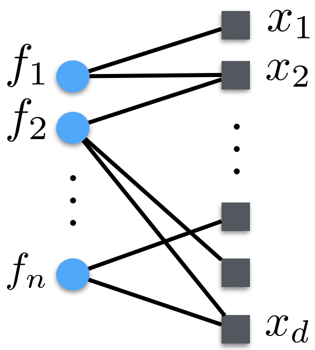
[Mania, Pan, **P**, Recht, Ramchandran, Jordan, 2015]

Joint work with





HOGWILD! as noisy SGD



Each processor in parallel

sample function f_i

$x =$ read shared memory

$g = -\gamma \cdot \nabla f_i(x)$

for v in the support of f **do**

$\quad x_v \leftarrow x_v + g_v$

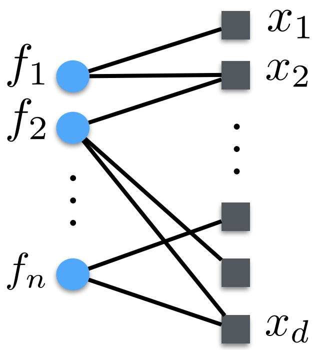
- <u>Def:</u> s_k is the k-th sampled data point

- <u>Fact:</u> Cores don't read "actual" iterates x_k but "noisy iterates" $\hat{x}_k$

- After *T* processed samples, the contents of RAM are:
  (atomic writes + commutativity)

Ex.

HOGWILD! as noisy SGD



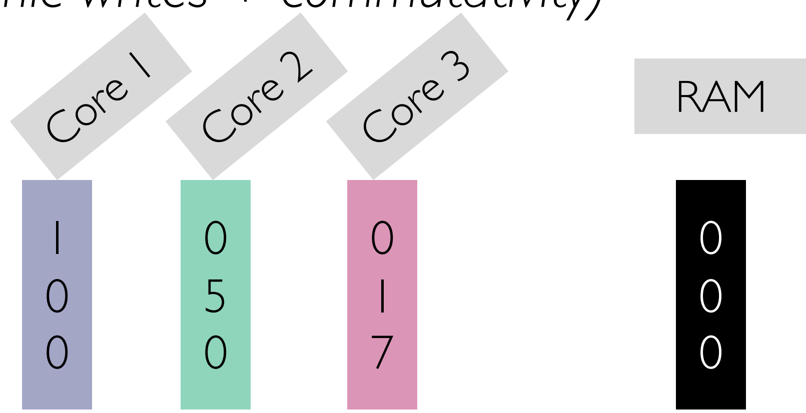
Each processor in parallel

sample function f_i
$x =$ read shared memory
$g = -\gamma \cdot \nabla f_i(x)$
for v in the support of f **do**
$\quad x_v \leftarrow x_v + g_v$

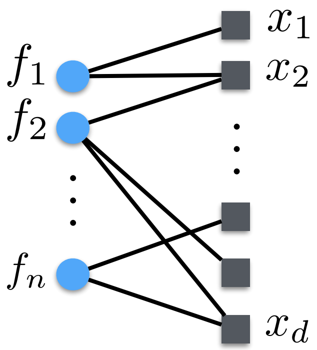
- <u>Def:</u> s_k is the k-th sampled data point
- <u>Fact:</u> Cores don't read "actual" iterates x_k but "noisy iterates" $\hat{x}_k$

- After *T* processed samples, the contents of RAM are:
  (atomic writes + commutativity)

Ex.



HOGWILD! as noisy SGD



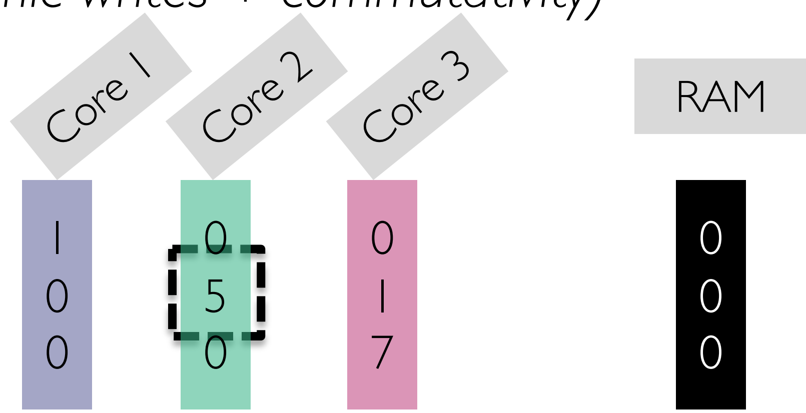
Each processor in parallel

sample function f_i
$x =$ read shared memory
$g = -\gamma \cdot \nabla f_i(x)$
for v in the support of f **do**
$x_v \leftarrow x_v + g_v$

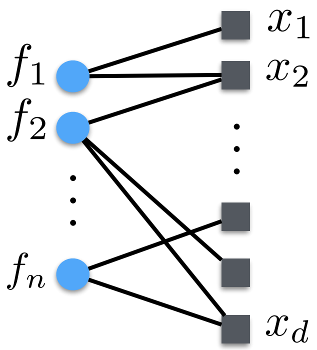
- Def: s_k is the k-th sampled data point
- Fact: Cores don't read "actual" iterates x_k but "noisy iterates" $\hat{x}_k$

- After *T* processed samples, the contents of RAM are:
  (atomic writes + commutativity)

Ex.



HOGWILD! as noisy SGD



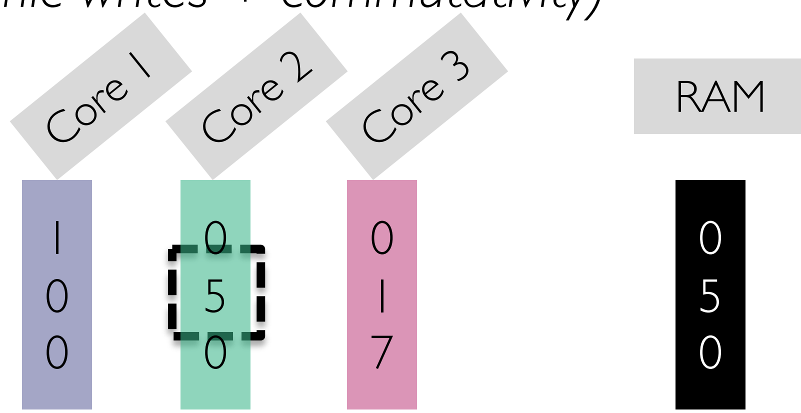
Each processor in parallel

sample function f_i
$x =$ read shared memory
$g = -\gamma \cdot \nabla f_i(x)$
for v in the support of f **do**
$\quad x_v \leftarrow x_v + g_v$

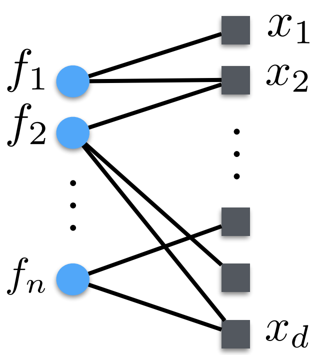
- <u>Def:</u> s_k is the k-th sampled data point
- <u>Fact:</u> Cores don't read "actual" iterates x_k but "noisy iterates" $\hat{x}_k$

- After T processed samples, the contents of RAM are: *(atomic writes + commutativity)*

Ex.



HOGWILD! as noisy SGD

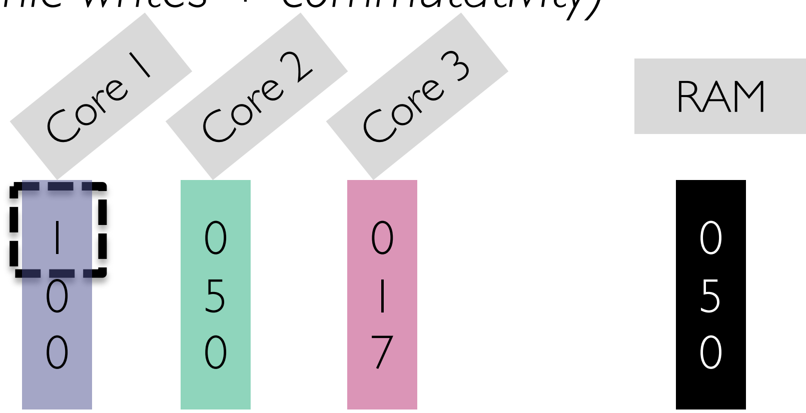


Each processor in parallel

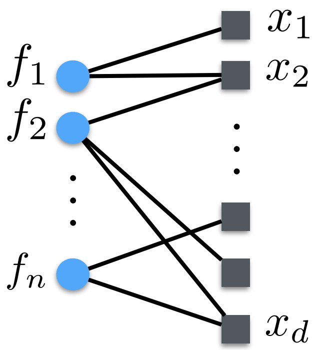
sample function f_i
$x =$ read shared memory
$g = -\gamma \cdot \nabla f_i(x)$
for v in the support of f **do**
  $x_v \leftarrow x_v + g_v$

- Def: s_k is the k-th sampled data point
- Fact: Cores don't read "actual" iterates x_k but "noisy iterates" $\hat{x}_k$
- After *T* processed samples, the contents of RAM are: *(atomic writes + commutativity)*

Ex.



HOGWILD! as noisy SGD



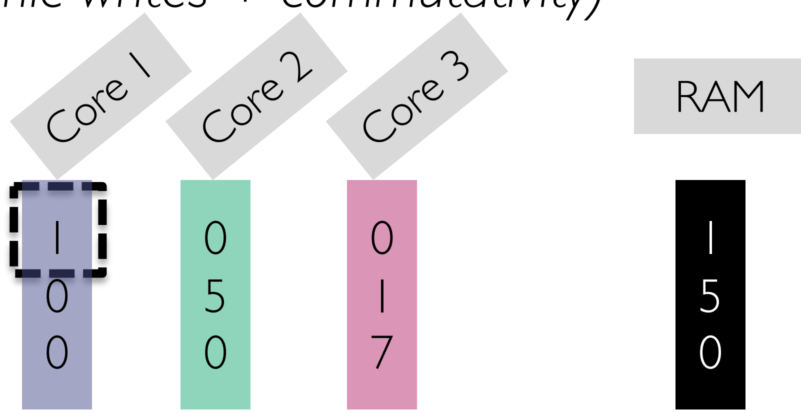
Each processor in parallel

sample function f_i
$x =$ read shared memory
$g = -\gamma \cdot \nabla f_i(x)$
for v in the support of f **do**
$\quad x_v \leftarrow x_v + g_v$

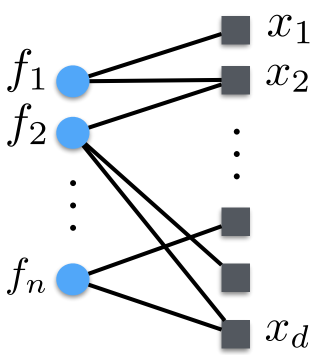
- Def: s_k is the k-th sampled data point
- Fact: Cores don't read "actual" iterates x_k but "noisy iterates" $\hat{x}_k$

- After *T* processed samples, the contents of RAM are:
  (atomic writes + commutativity)

Ex.



HOGWILD! as noisy SGD



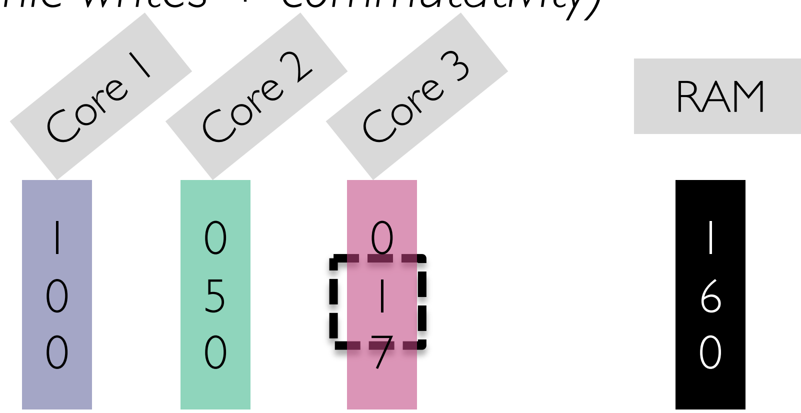
Each processor in parallel

sample function f_i
x = read shared memory
$g = -\gamma \cdot \nabla f_i(x)$
for v in the support of f **do**
$\quad x_v \leftarrow x_v + g_v$

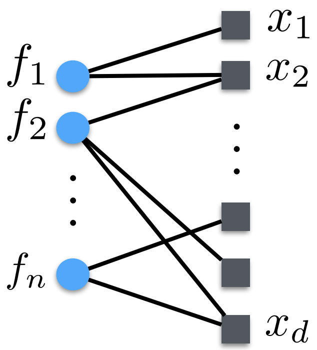
- <u>Def:</u> s_k is the k-th sampled data point
- <u>Fact:</u> Cores don't read "actual" iterates x_k but "noisy iterates" $\hat{x}_k$

- After *T* processed samples, the contents of RAM are: *(atomic writes + commutativity)*

Ex.



HOGWILD! as noisy SGD



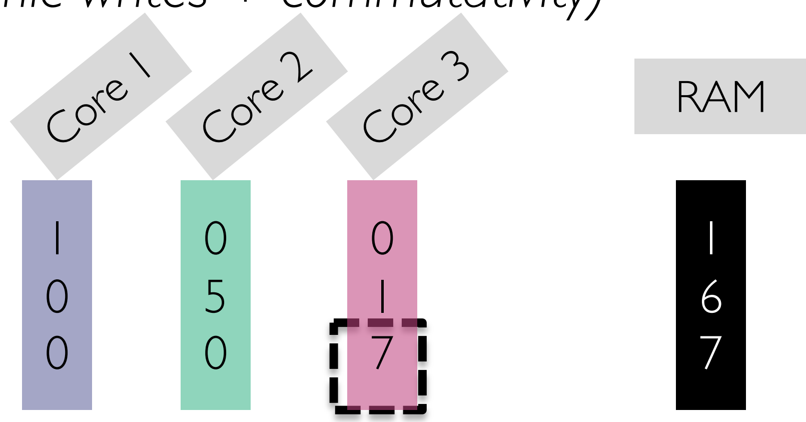
Each processor in parallel

sample function f_i
$x =$ read shared memory
$g = -\gamma \cdot \nabla f_i(x)$
for v in the support of f **do**
  $x_v \leftarrow x_v + g_v$

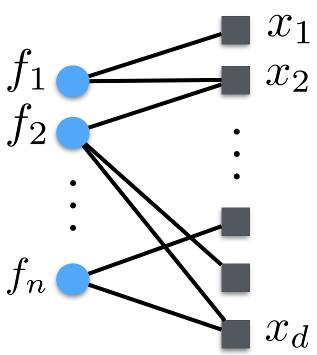
- <u>Def:</u> s_k is the k-th sampled data point
- <u>Fact:</u> Cores don't read "actual" iterates x_k but "noisy iterates" $\hat{x}_k$

- After *T* processed samples, the contents of RAM are:
  (atomic writes + commutativity)

Ex.



HOGWILD! as noisy SGD



Each processor in parallel

sample function f_i
$x =$ read shared memory
$g = -\gamma \cdot \nabla f_i(x)$
for v in the support of f **do**
$\quad x_v \leftarrow x_v + g_v$

- <u>Def:</u> s_k is the k-th sampled data point
- <u>Fact:</u> Cores don't read "actual" iterates x_k but "noisy iterates" $\hat{x}_k$

- After *T* processed samples, the contents of RAM are: *(atomic writes + commutativity)*

$$x_0 - \gamma \cdot \nabla f_{s_0}(\hat{x}_0) - \ldots - \gamma \cdot \nabla f_{s_{T-1}}(\hat{x}_{T-1})$$

Main Questions:

1) Where does noise come from?
2) How strong is it?

Convergence Rates for Noisy SGD

We want to analyze noisy SGD

$$x_{k+1} = x_k - \gamma \cdot \nabla f_{s_k}(\hat{x}_k)$$

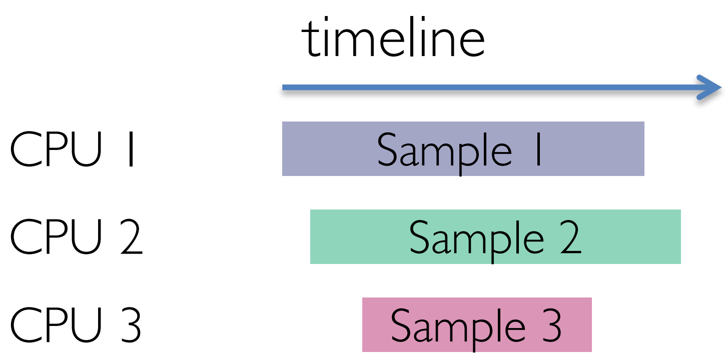
Elementary analysis (using m-strong convexity assumption on *f*):

$$\mathbb{E}\{\|x_{k+1} - x^*\|^2\} \leq (1 - \gamma \cdot m) \cdot \mathbb{E}\{\|x_k - x^*\|^2\} + \gamma^2 \cdot \mathbb{E}\{\|\nabla f_{s_k}(\hat{x}_k)\|^2\}$$

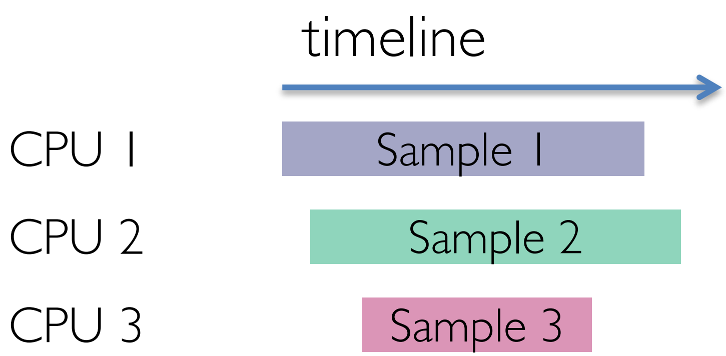
Simple Lemma:
if both terms = $O(\gamma^2 M^2)$,
Noisy SGD gets same rates as SGD *(up to multiplicative constants)*

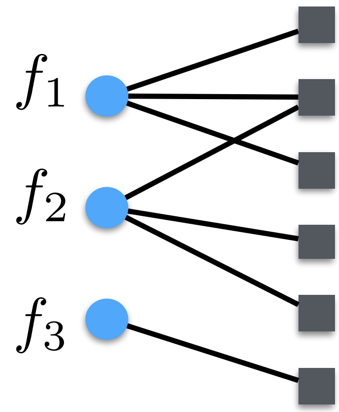
So.. is asynchrony noise small?

Understanding Asynchrony Noise

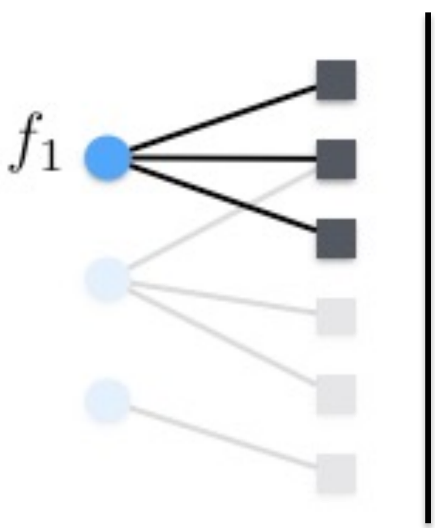


Understanding Asynchrony Noise

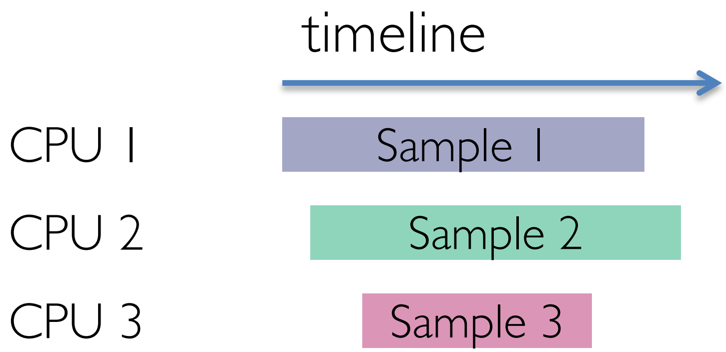


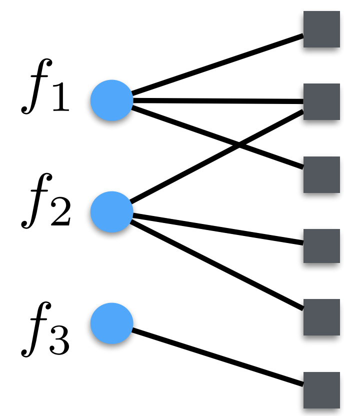


"Serialized" Processing Timeline

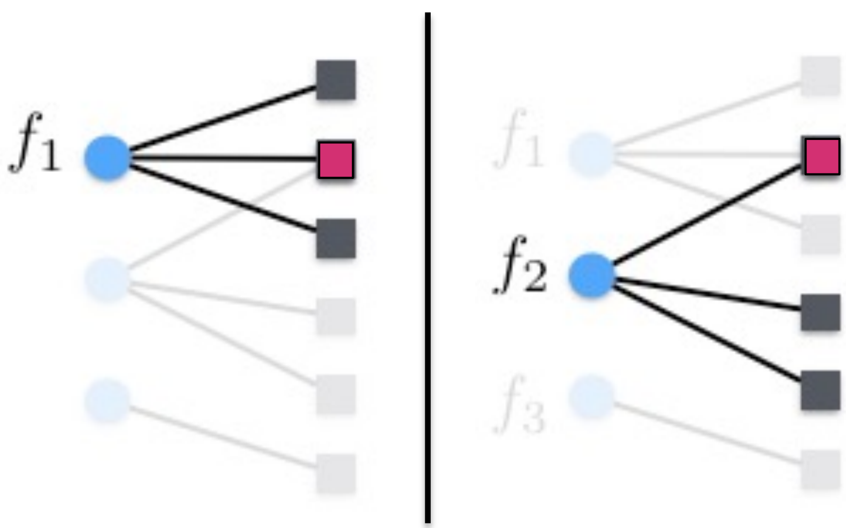


Understanding Asynchrony Noise

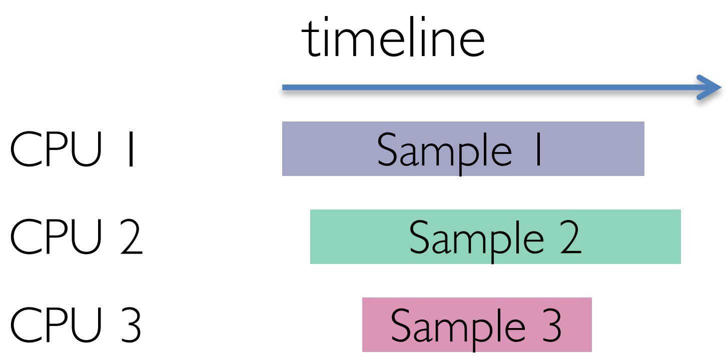


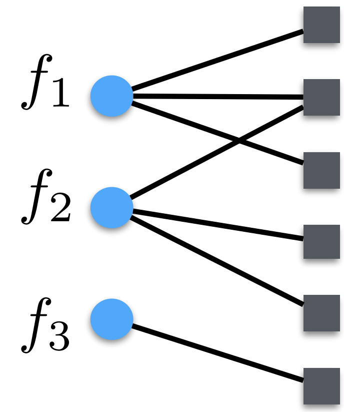


“Serialized” Processing Timeline

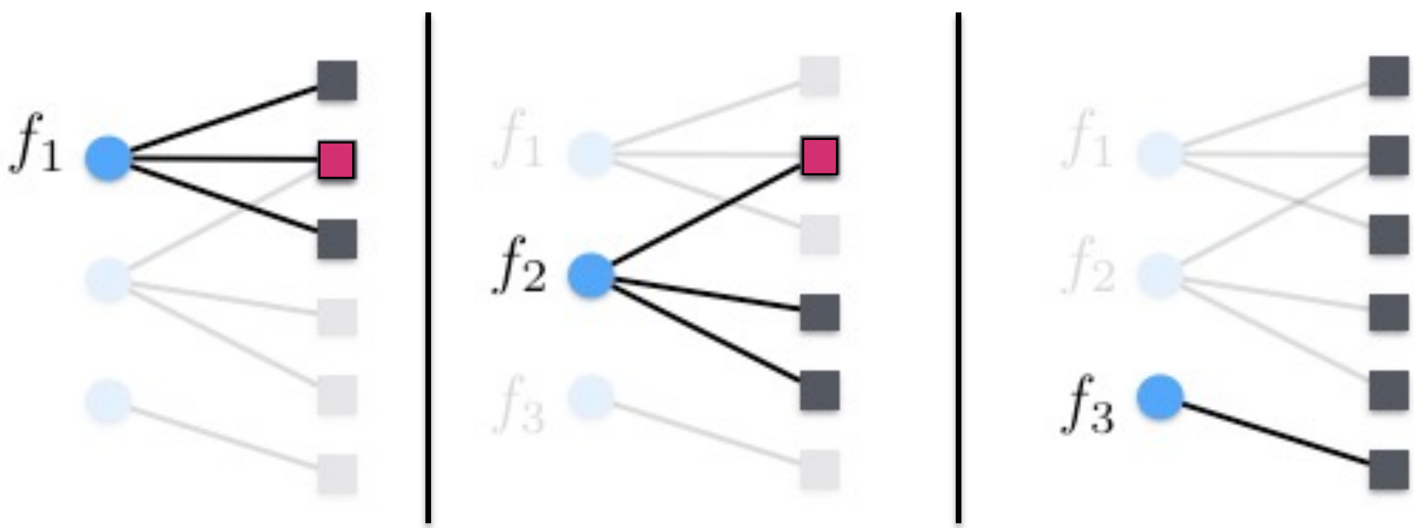


Understanding Asynchrony Noise

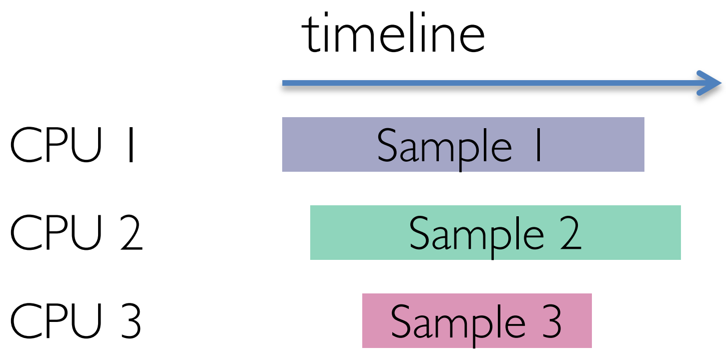


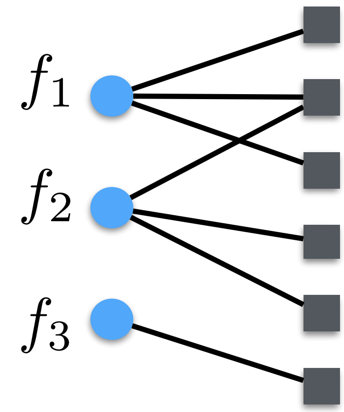


"Serialized" Processing Timeline

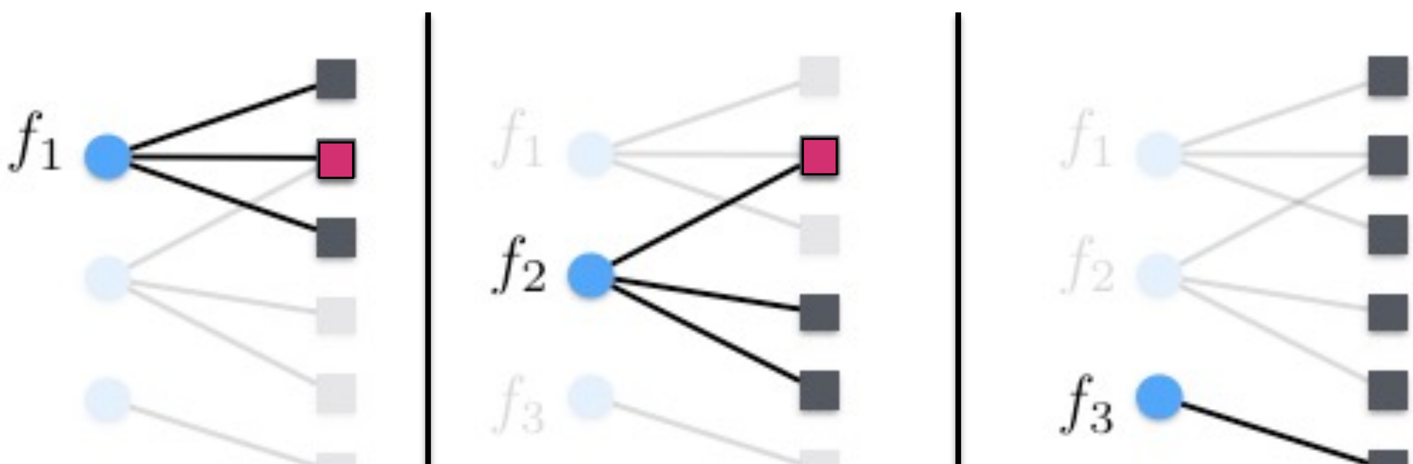


Understanding Asynchrony Noise





"Serialized" Processing Timeline



Asynchrony noise is <u>combinatorial</u>

coordinates in conflict can be as noisy as possible.

(no generative model assumptions)

Convergence Rates for Hogwild!

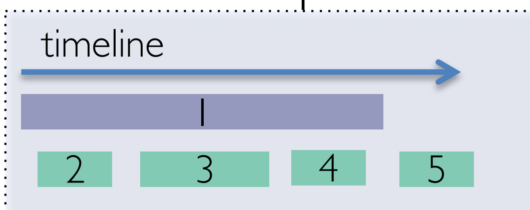
- Let's now analyze "noisy" SGD:

$$x_{k+1} = x_k - \gamma \cdot \nabla f_{s_k}(\hat{x}_k)$$

- *Assumption:* no more than τ samples processed, while a core is processing one

Eg, τ = 3

while 1 is being processed no more than 3 updates occur



Important Note:

If s_i is done before s_k is sampled:

its gradient contribution is recorded in shared RAM,

when a thread starts working on s_k

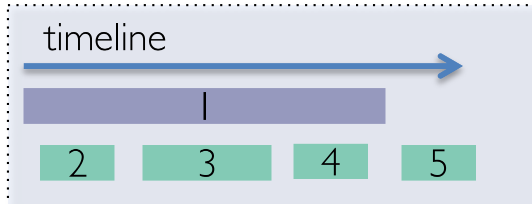
If s_i overlaps in time with s_k (i.e., the two samples are concurrently processed) :

its gradient contribution is only partially recorded in shared RAM,

when a thread starts working on s_k

Convergence Rates for Hogwild!

- *Assumption:* no more than τ samples processed, while a core is processing one



- For each sample s_k

Any difference between $\hat{x}_k$ and x_k caused only by samples that "overlap" with s_k

Therefore

- If s_i is sampled before s_k it *might* overlap with s_k iff $i \geq k - \tau$
- If s_i is sampled after s_k, it *might* overlap with s_k iff $i \leq k + \tau$

Hence:

$$\hat{x}_k - x_k = \sum_{i=k-\tau, i \neq k}^{k+\tau} \gamma \cdot S_i^j \nabla f_{s_i}(\hat{x}_i)$$

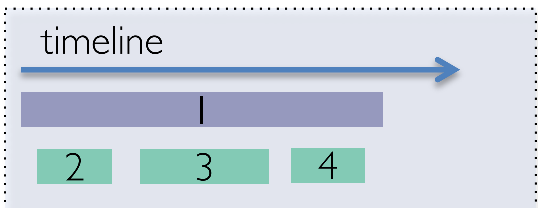
$S_i^j = \text{diagonal with entries in } \{-1, 0, 1\}$

Convergence Rates for Hogwild!

Let's now analyze "noisy" SGD:

$$x_{k+1} = x_k - \gamma \cdot \nabla f_{s_k}(\hat{x}_k)$$

<u>*Assumption:*</u> no more than τ samples processed, while a core is processing one



$$\hat{x}_k - x_k = \sum_{i=k-\tau, i \neq k}^{k+\tau} \gamma \cdot S_i^j \nabla f_{s_i}(\hat{x}_i)$$

Elementary analysis (using m-strong convexity assumption on *f*):

$$\mathbb{E}\{\|x_{k+1} - x^*\|^2\} \leq (1 - \gamma \cdot m) \cdot \mathbb{E}\{\|x_k - x^*\|^2\} + \gamma^2 \cdot \mathbb{E}\{\|\nabla f_{s_k}(\hat{x}_k)\|^2\}$$

Lemma:

if = O( )

Noisy SGD gets same rates as SGD *(up to multiplicative constants)*

Q: Is asynchrony noise that small?

Asynchrony Noise

The main thing we need to bound

$$\gamma\mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\} = \gamma^2 \mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle$$

Reminder: we need it smaller than $\gamma^2 M^2$

Asynchrony Noise

The main thing we need to bound

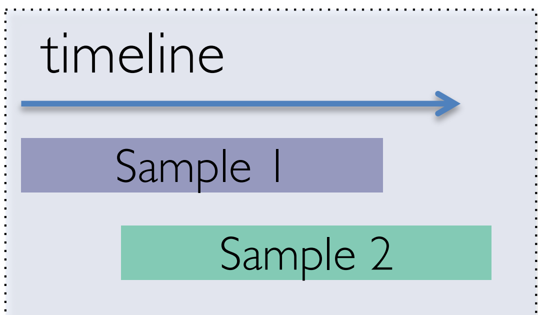
$$\boxed{\gamma\mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}} = \gamma^2\mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k)\right\rangle$$

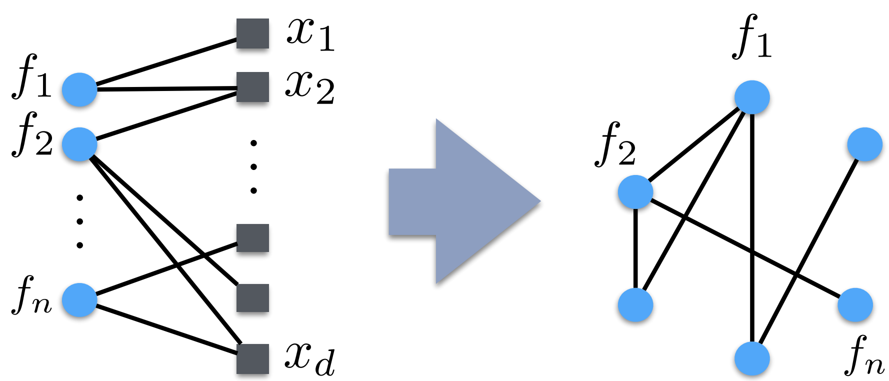
Reminder: we need it smaller than $\gamma^2 M^2$

Note: Asynchrony causes error if sampled grads overlap.

Simple Idea:

Samples might be concurrently processed, but they only "interfere" if they are talking to the same variables:





If the interference is "rare" the noise term should be small

Asynchrony Noise

The main thing we need to bound

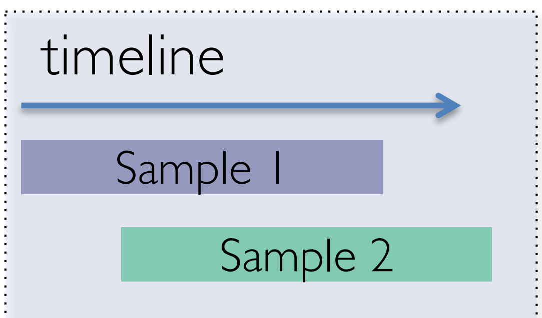
$$\boxed{\gamma\mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}} = \gamma^2\mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle$$

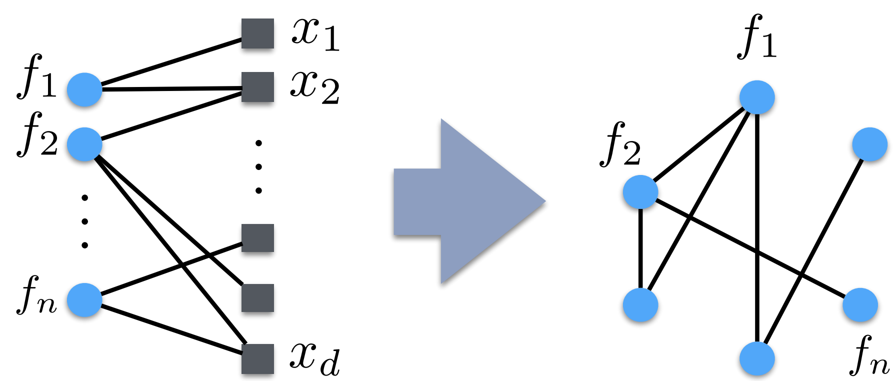
Reminder: we need it smaller than $\gamma^2 M^2$

Note: Asynchrony causes error if sampled grads overlap.

<u>*Simple Idea:*</u>

Samples might be concurrently processed, but they only "interfere" if they are talking to the same variables:





Asynchrony Noise

The main thing we need to bound

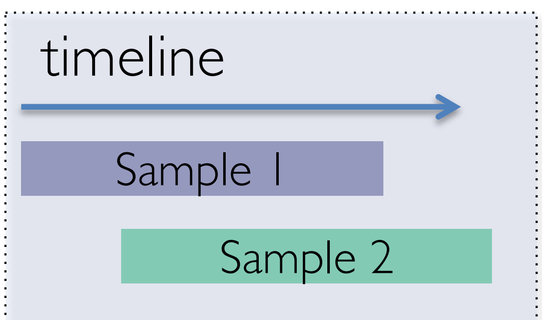
$$\boxed{\gamma \mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}} = \gamma^2 \mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k)\right\rangle$$

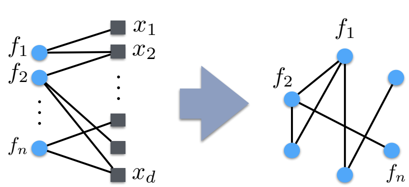
Reminder: we need it smaller than $\gamma^2 M^2$

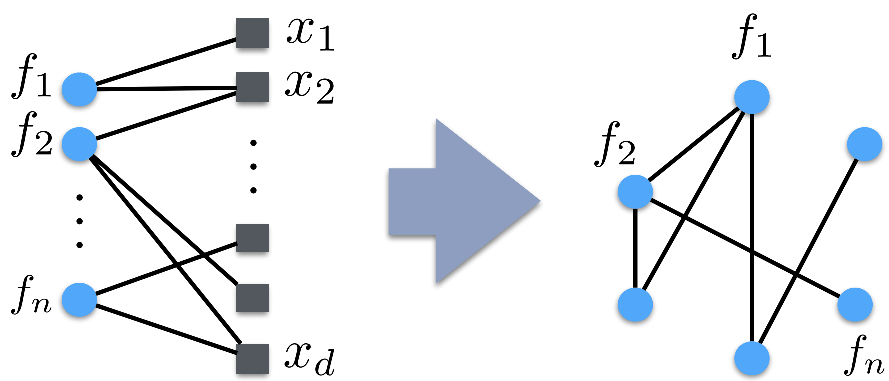
Note: Asynchrony causes error if sampled grads overlap.

Simple Idea:

Samples might be concurrently processed, but they only "interfere" if they are talking to the same variables:







Asynchrony Noise

The main thing we need to bound

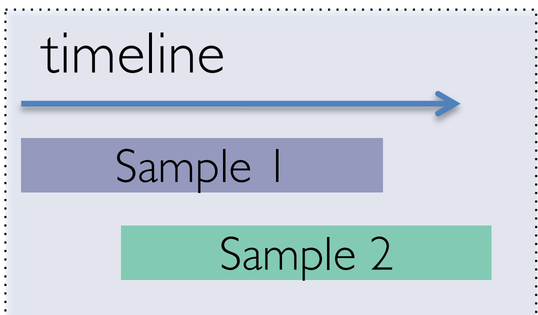
$$\boxed{\gamma\mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}} = \gamma^2\mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle$$

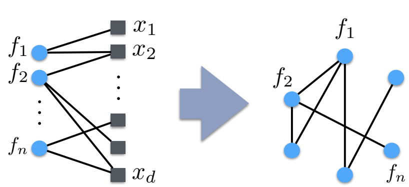
Reminder: we need it smaller than $\gamma^2 M^2$

Note: Asynchrony causes error if sampled grads overlap.

Simple Idea:

Samples might be concurrently processed, but they only "interfere" if they are talking to the same variables:





Bad Event

If the functions sampled share variables
$$\langle \nabla f_{s_i}(x_1), \nabla f_{s_j}(y_2)\rangle \neq 0$$

Good Event

If the functions sampled do not share variables
$$\langle \nabla f_{s_i}(x_1), \nabla f_{s_j}(y_2)\rangle = 0$$

Asynchrony Noise

The main thing we need to bound

$$\boxed{\gamma \mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}} = \gamma^2 \mathbb{E} \left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle$$

Reminder: we need it smaller than $\gamma^2 M^2$

Note: Asynchrony causes error if sampled grads overlap.

$$\gamma^2 \left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle \leq \gamma^2 \left| \left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} \cdot S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle \right|$$

Cauchy-Schwarz

$$a \cdot b \leq \frac{a^2 + b^2}{2}$$

$$\|\nabla f_s(x)\|^2 \leq M^2$$

Asynchrony Noise

The main thing we need to bound

$$\boxed{\gamma \mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}} = \gamma^2 \mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle$$

Reminder: we need it smaller than $\gamma^2 M^2$

Note: Asynchrony causes error if sampled grads overlap.

$$\gamma^2 \mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle \leq \gamma^2 \mathbb{E}\left\{ \sum_{i=k-\tau, i\neq k}^{k+\tau} M^2 \cdot \mathbf{1}_{s_i \cap s_k = 0} \right\}$$

What is $\mathbf{1}_{s_i \cap s_k = 0}$?

Indicator: Does sample *i* overlap with sample *k*?

Asynchrony Noise

The main thing we need to bound

$$\boxed{\gamma \mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}} = \gamma^2 \mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle$$

Reminder: we need it smaller than $\gamma^2 M^2$

Note: Asynchrony causes error if sampled grads overlap.

$$\gamma^2 \mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle \leq \gamma^2 \mathbb{E}\left\{ \sum_{i=k-\tau, i\neq k}^{k+\tau} M^2 \cdot \mathbf{1}_{s_i \cap s_k = 0} \right\}$$

$$\leq \gamma^2 \cdot 2\tau \cdot M^2 \cdot \mathbb{E}\{\mathbf{1}_{s_i \cap s_k = 0}\}$$

What is $\mathbb{E}\{\mathbf{1}_{s_i \cap s_k = 0}\}$?

The probability that sample *i* overlaps with sample *k*

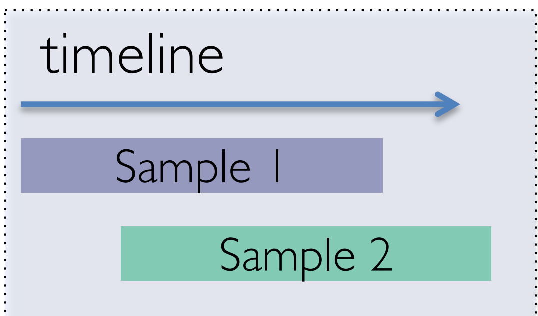
Asynchrony Noise

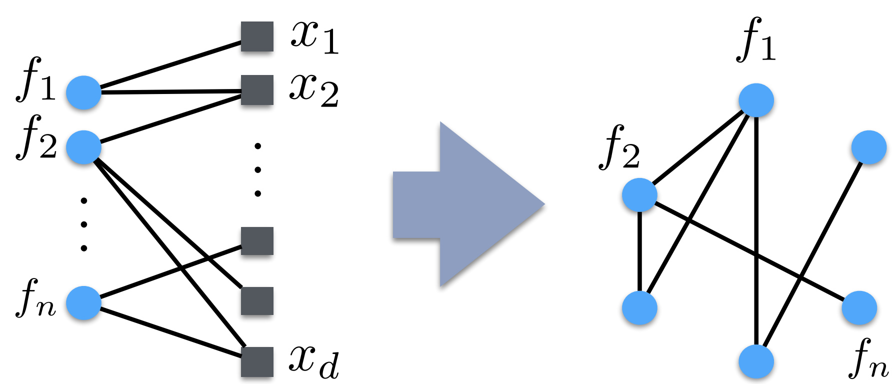
The main thing we need to bound

$$\gamma\mathbb{E}\{\langle x_k-\hat{x}_k,\nabla f_{s_k}(\hat{x}_k)\rangle\} = \gamma^2\mathbb{E}\left\langle \sum_{i=k-\tau,\, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k)\right\rangle$$

Reminder: we need it smaller than $\gamma^2 M^2$

Note: Asynchrony causes error if sampled grads overlap.





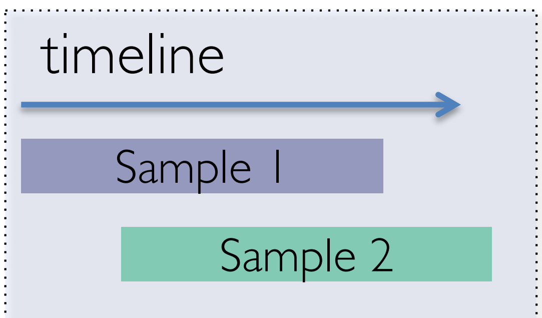
Asynchrony Noise

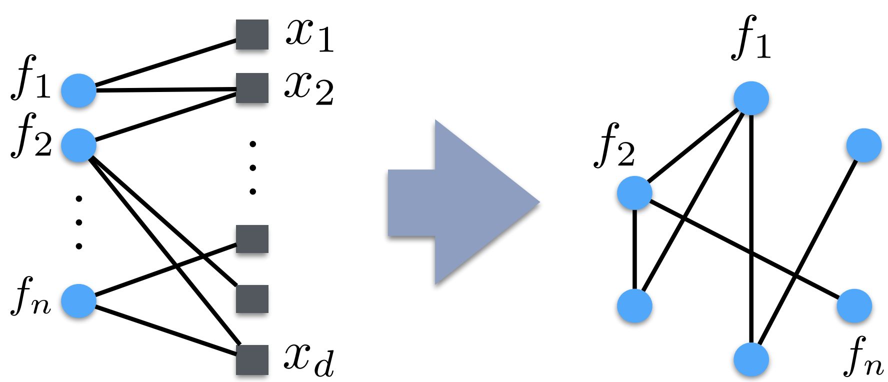
The main thing we need to bound

$$\boxed{\gamma \mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}} = \gamma^2 \mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k)\right\rangle$$

Reminder: we need it smaller than $\gamma^2 M^2$

Note: Asynchrony causes error if sampled grads overlap.





$$\boxed{\text{Pr(two samples conflict)} = \frac{\Delta_{\text{av}}}{n}}$$

Asynchrony Noise

The main thing we need to bound

$$\boxed{\gamma\mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}} = \gamma^2\mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k)\right\rangle$$

Reminder: we need it smaller than $\gamma^2 M^2$

Note: Asynchrony causes error if sampled grads overlap.

$$\gamma^2\mathbb{E}\left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^j \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k)\right\rangle \leq \gamma^2 \cdot 2\tau \cdot M^2 \cdot \mathbb{E}\{\mathbf{1}_{s_i \cap s_k \neq 0}\}$$

The noise term is below $\gamma^2 M^2$ when $\tau \leq \dfrac{n}{2\Delta_{\text{av}}}$

Convergence Rates for Hogwild!

$$x_{k+1} = x_k - \gamma \cdot \nabla f_{s_k}(\hat{x}_k)$$

Reminder of Noisy SGD Rates:

$$\mathbb{E}\{\|x_{k+1} - x^*\|^2\} \leq (1 - \gamma \cdot m) \cdot \mathbb{E}\{\|x_k - x^*\|^2\} + \gamma^2 \cdot \mathbb{E}\{\|\nabla f_{s_k}(\hat{x}_k)\|^2\}$$
$$+ 2\gamma m \cdot \mathbb{E}\{\|x_k - \hat{x}_k\|^2\} + 2\gamma \cdot \mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}$$

Lemma:

if = O( )

Noisy SGD gets same rates as SGD *(up to multiplicative constants)*

Hogwild Rates: Proof Recap

Hogwild is equivalent to a noisy serial SGD

asynchrony noise affects rates, but if bounded, not by much

When core delay is less than $\tau \leq \dfrac{n}{2\Delta_{\text{av}}}$, noise does not affect convergence

Hogwild! Achieves linear speedups

*=in terms of worst case convergence

Convergence of Hogwild

THEOREM 3.4. *If the number of samples that overlap in time with a single sample during the execution of* HOGWILD! *is bounded as*

$$\tau = \mathcal{O}\left(\min\left\{\frac{n}{\overline{\Delta}_C}, \frac{M^2}{\epsilon m^2}\right\}\right),$$

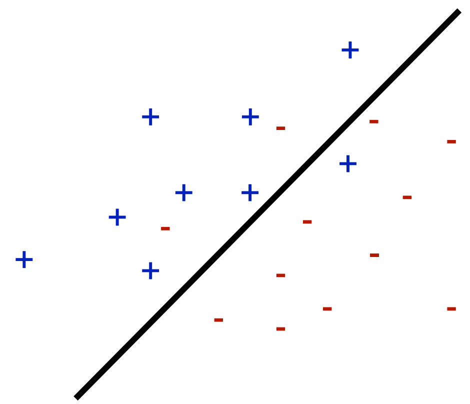
HOGWILD!, *with step size* $\gamma = \frac{\epsilon m}{2M^2}$, *reaches an accuracy of* $\mathbb{E}\|\mathbf{x}_k - \mathbf{x}^*\|^2 \leq \epsilon$ *after*

$$T \geq \mathcal{O}(1)\frac{M^2 \log\left(\frac{a_0}{\epsilon}\right)}{\epsilon m^2}$$

iterations.

Examples of Sparse Problems

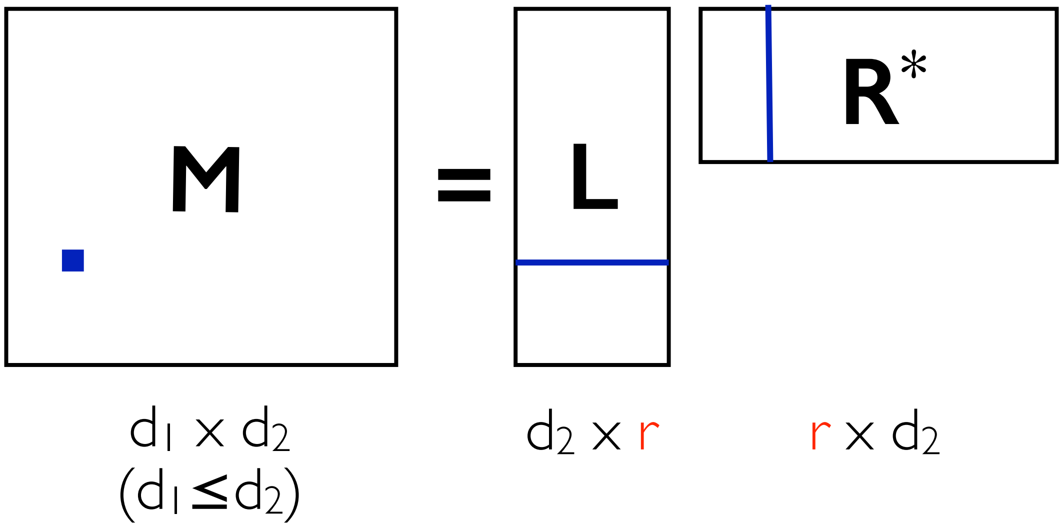
Sparse Support Vector Machines

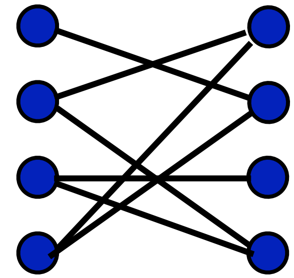


sparse vectors

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

Matrix Completion

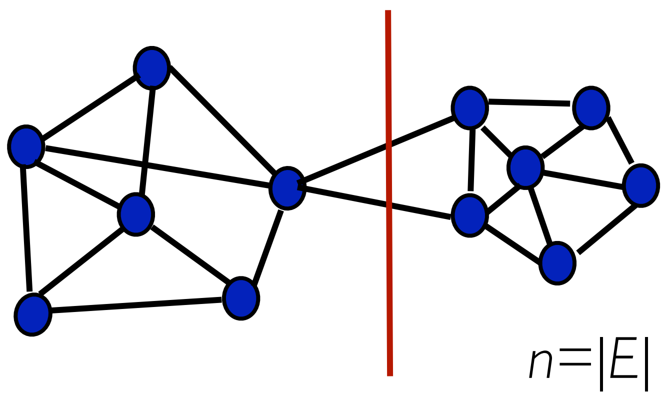




Entries Specified on set *E* (with |*E*|=*n*)

$$\mathrm{minimize}_{(\mathbf{L},\mathbf{R})} \sum_{(u,v)\in E} \left\{ (\mathbf{L}_u \mathbf{R}_v^T - M_{uv})^2 + \mu_u \|\mathbf{L}_u\|_F^2 + \mu_v \|\mathbf{R}_v\|_F^2 \right\}$$

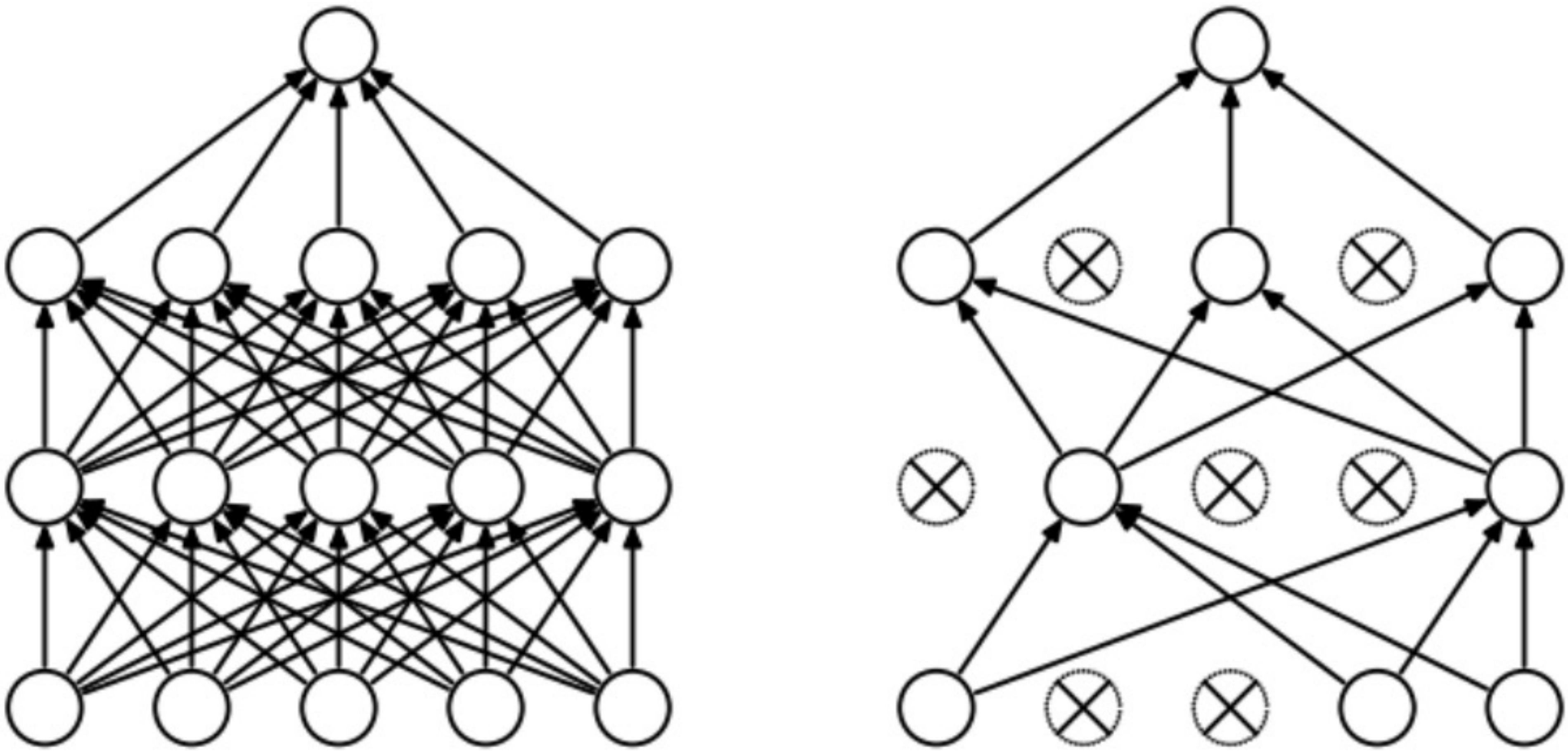
Graph Cuts



- Image Segmentation
- Entity Resolution
- Topic Modeling

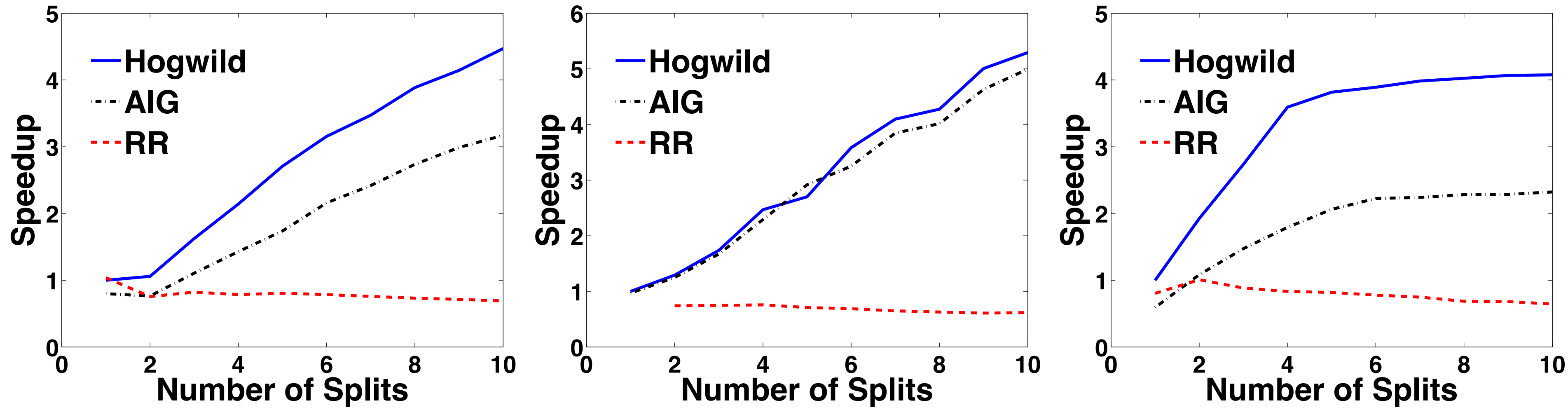
$$\begin{array}{ll} \text{minimize}_x & \sum_{(u,v)\in E} w_{uv} \|x_u - x_v\|_1 \\ \text{subject to} & \mathbf{1}_K^T x_v = 1\,,\ x_v \geq 0\,, \quad \text{for } v = 1, \ldots, D \end{array}$$

Sparsified BackProp



(a) Standard Neural Net　　(b) After applying dropout.

Speedups



SVM
RCV1

MC
Netflix

CUTS
Abdomen

Experiments run on 12 core machine

10 cores for gradients, 1 core for data shuffling

Open Problems

Open Problems: Part 1

Assumptions: Sparsity + convexity => linear speedups

O.P. :
Hogwild! On Dense Problems

Maybe we should featurize dense ML Problems,
so that updates are **sparse**

O.P. :
Fundamental Trade-off
Sparsity vs Learning Quality?

Open Problems: Part 2

- What we proved:

$$\text{worst case speedup} = \frac{\text{bound on \#iter of SGD to } \epsilon}{\text{bound on \#iter of Parallel SGD to } \epsilon}$$

- What we really care about:

$$\text{speedup} = \frac{\text{Time of serial } \mathcal{A} \text{ to accuraccy } \epsilon}{\text{Time of parallel } \mathcal{A} \text{ to accuraccy } \epsilon}$$

O.P. :
True Speedup Proofs for Hogwild

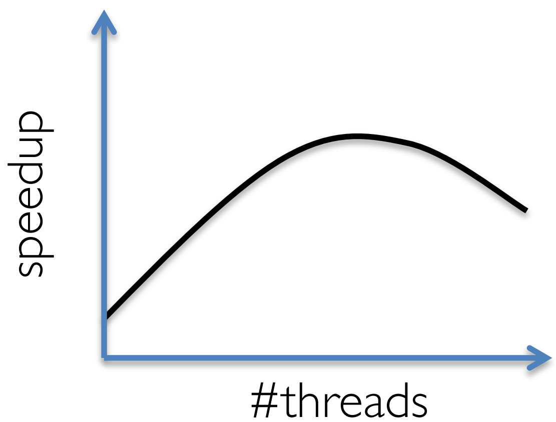
Holy Grail

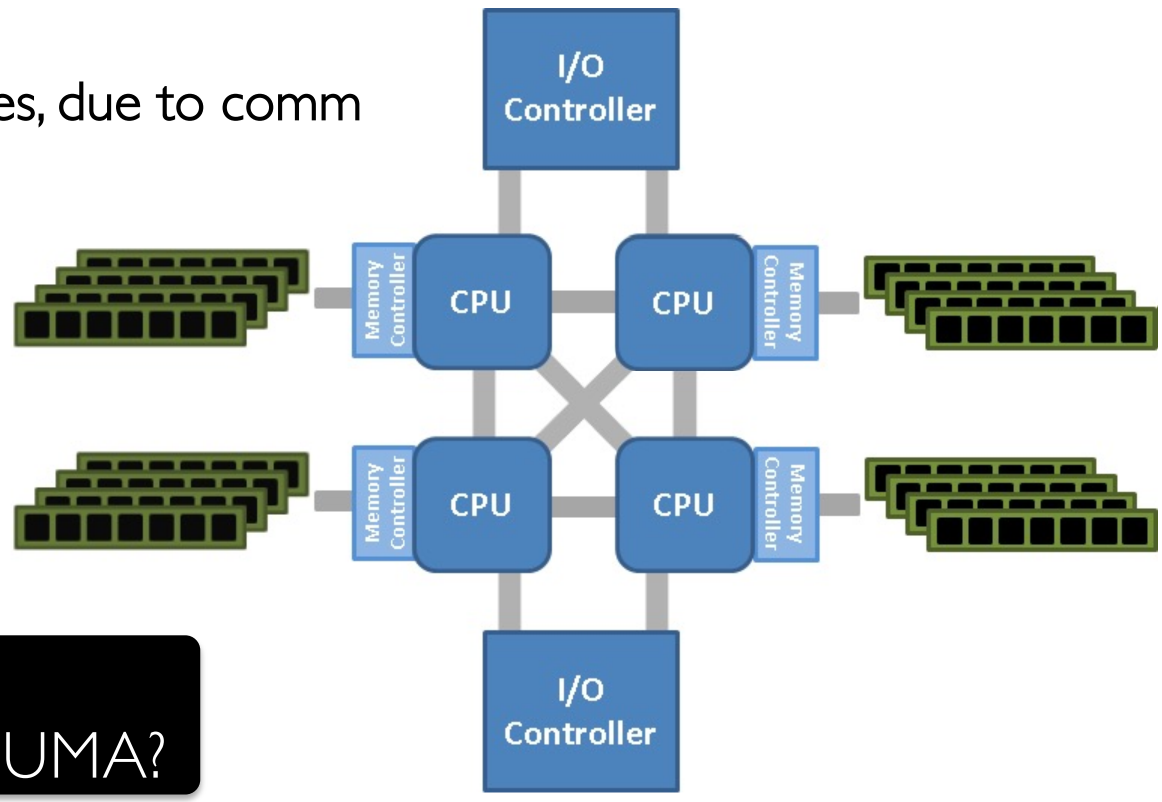
O.P. :
Guarantees for Nonconvex Problems?

Open Problems: Part 3

Hogwild! Algorithms great for Shared Memory Systems

- Issues when scaling across nodes, due to comm





O.P. :
How to provably scale on NUMA?

- Similar Issues for Distributed:

O.P. :
Sync vs Async is still open

Reproducible Models

Reproducibility

- HOGWILD! Models are not reproducible
- Each training session has inherent "system" randomness
- Does not allow to recreate models if needed
- Barrier for accountability and reproducibility
- How can we resolve it?

Reproducibility

Serial Equialence

$$A_{\text{serial}}(S, \pi) = A_{\text{parallel}}(S, \pi)$$

For all Data sets S
For all data order **π** (data points can be arbitrarily repeated)

<u>Main advantage:</u>
- we only need to "prove" speedups
- Convergence proofs inherited directly from serial

<u>Main Issue:</u>
- Serial equivalence too strict
- Cannot guarantee any speedups in the general case

Serial Equivalence

- When is serial equivalence feasible?

- What algorithmic patterns allow for efficient serial equivalent?

- Can a serial equivalent parallel algorithm ever be competitive with Hogwild?

Reading List

- Bertsekas, D.P., 1983. Distributed asynchronous computation of fixed points. Mathematical Programming, 27(1), pp.107-120. Vancouver

- Tsitsiklis, J., Bertsekas, D. and Athans, M., 1986. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. IEEE transactions on automatic control, 31(9), pp.803-812.

- Recht, B., Re, C., Wright, S. and Niu, F., 2011. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. Advances in neural information processing systems, 24.

- Liu, J., Wright, S., Ré, C., Bittorf, V. and Sridhar, S., 2014, June. An asynchronous parallel stochastic coordinate descent algorithm. In International Conference on Machine Learning (pp. 469-477). PMLR.

- Mania, Horia, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael Jordan. "Perturbed Iterate Analysis for Asynchronous Stochastic Optimization." (2019).

- J Reddi, S., Hefny, A., Sra, S., Poczos, B. and Smola, A.J., 2015. On variance reduction in stochastic gradient descent and its asynchronous variants. Advances in neural information processing systems, 28.