Distributed Asynchronous Online Learning for Natural Language Processing

Kevin Gimpel Dipanjan Das Noah A. Smith

Introduction

- Two recent lines of research in speeding up large learning problems:
  - Parallel/distributed computing
  - Online (and mini-batch) learning algorithms:

    stochastic gradient descent, perceptron, MIRA, stepwise EM

- How can we bring together the benefits of parallel computing and online learning?

Introduction

- We use **asynchronous** algorithms
  (Nedic, Bertsekas, and Borkar, 2001;
  Langford, Smola, and Zinkevich, 2009)

- We apply them to structured prediction tasks:
  - Supervised learning
  - Unsupervised learning with both convex and non-convex objectives

- Asynchronous learning speeds convergence and works best with small mini-batches

ARK lti Carnegie Mellon

Problem Setting

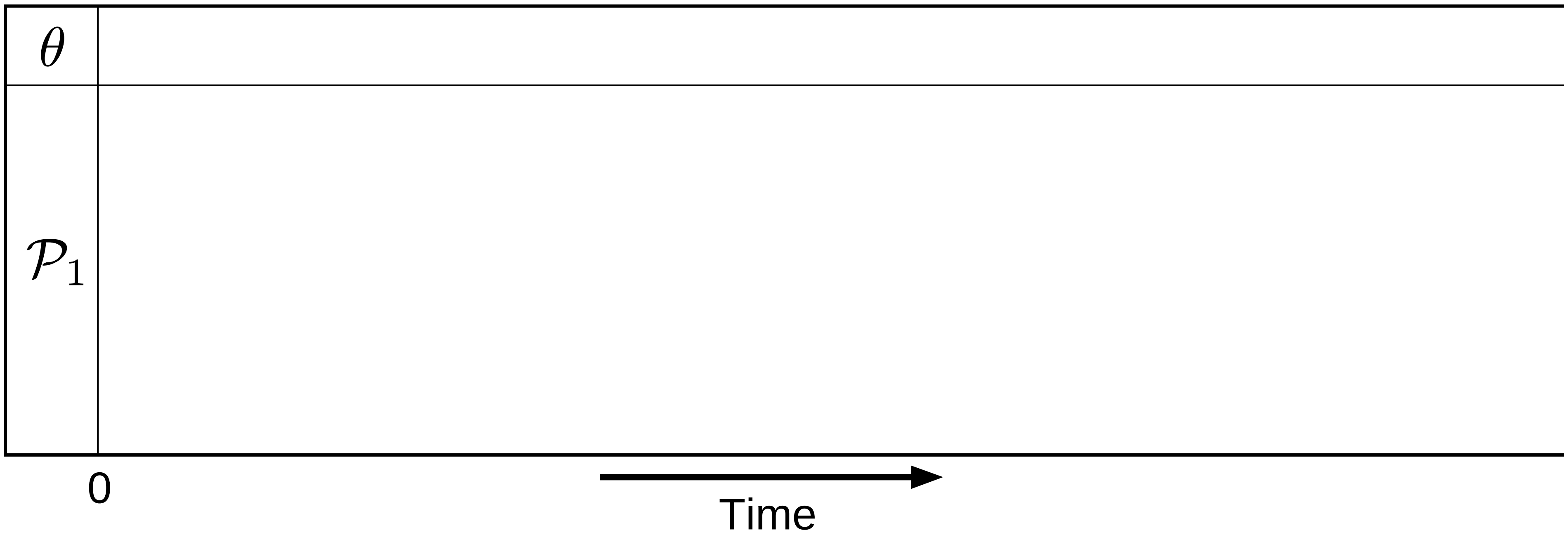
- Iterative learning
  - Moderate to large numbers of training examples
  - Expensive inference procedures for each example
  - For concreteness, we start with gradient-based optimization

- Single machine with multiple processors
  - Exploit shared memory for parameters, lexicons, feature caches, etc.
  - Maintain one master copy of model parameters

Single-Processor Batch Learning

Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Dataset:	$\mathcal{D}$

Single-Processor Batch Learning

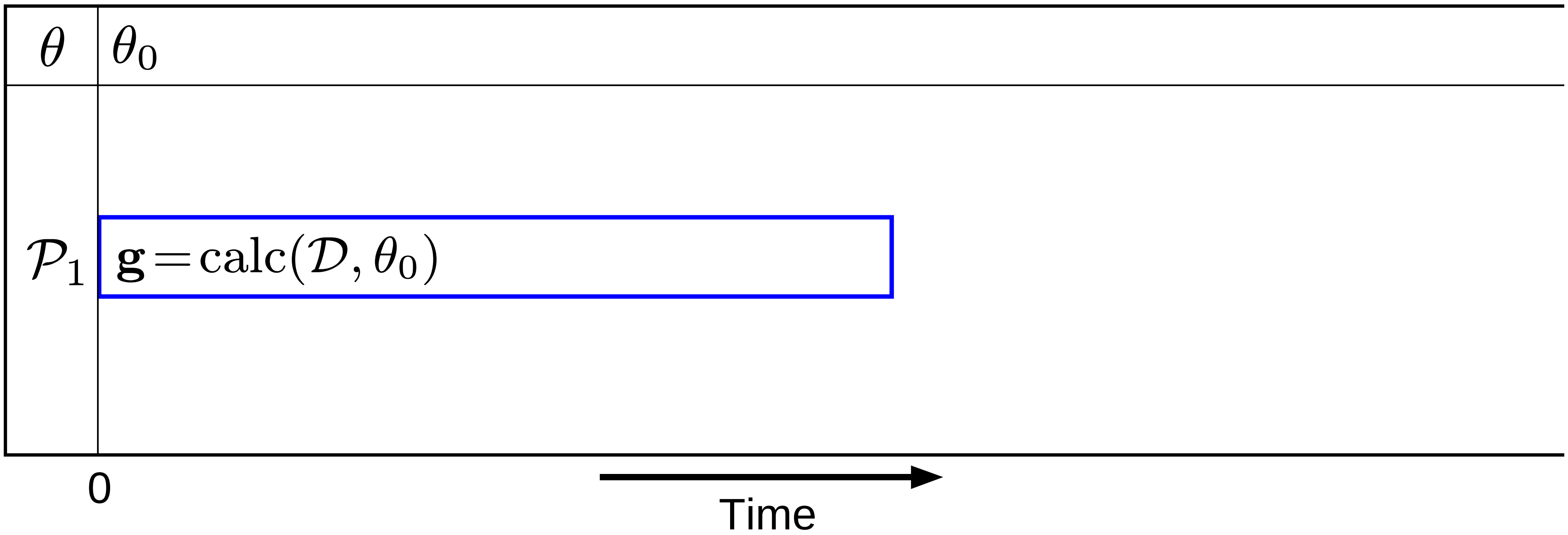


Parameters: θ_t

Processors: $\mathcal{P}_i$

Dataset: $\mathcal{D}$

Single-Processor Batch Learning

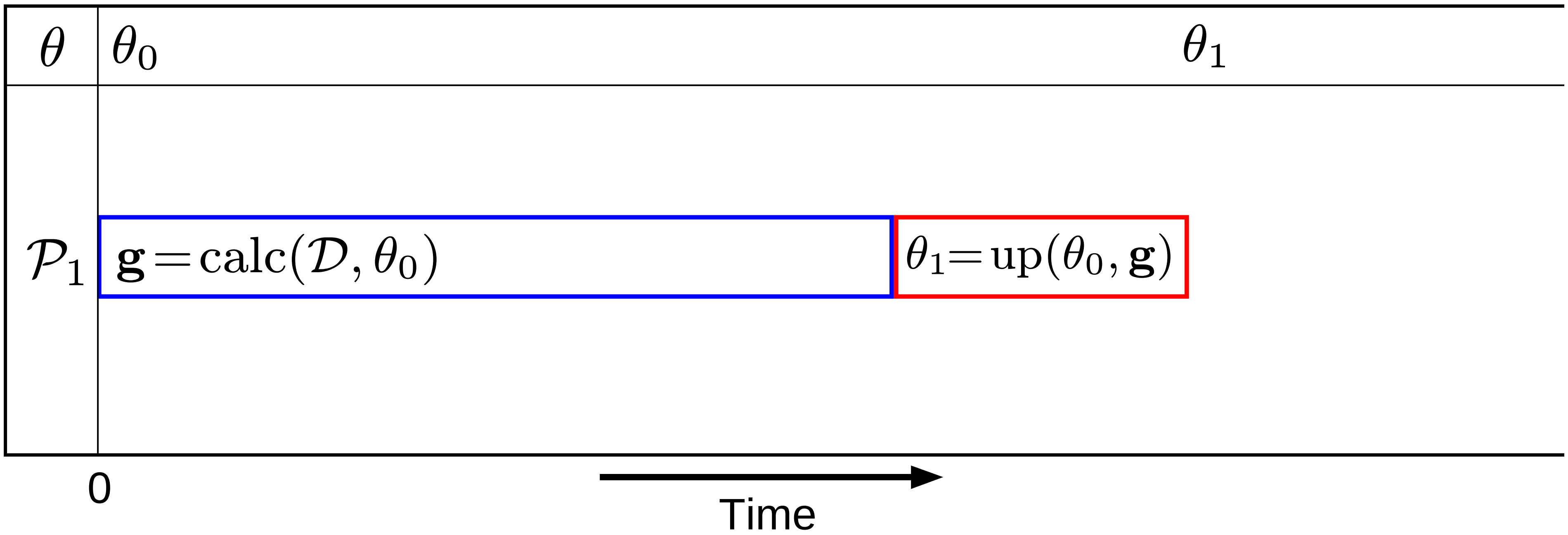


$\mathbf{g} = \text{calc}(\mathcal{D}, \theta)$:

Calculate gradient $\mathbf{g}$ on data $\mathcal{D}$ using parameters θ

Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Dataset:	$\mathcal{D}$

Single-Processor Batch Learning



$\mathbf{g} = \text{calc}(\mathcal{D}, \theta)$:

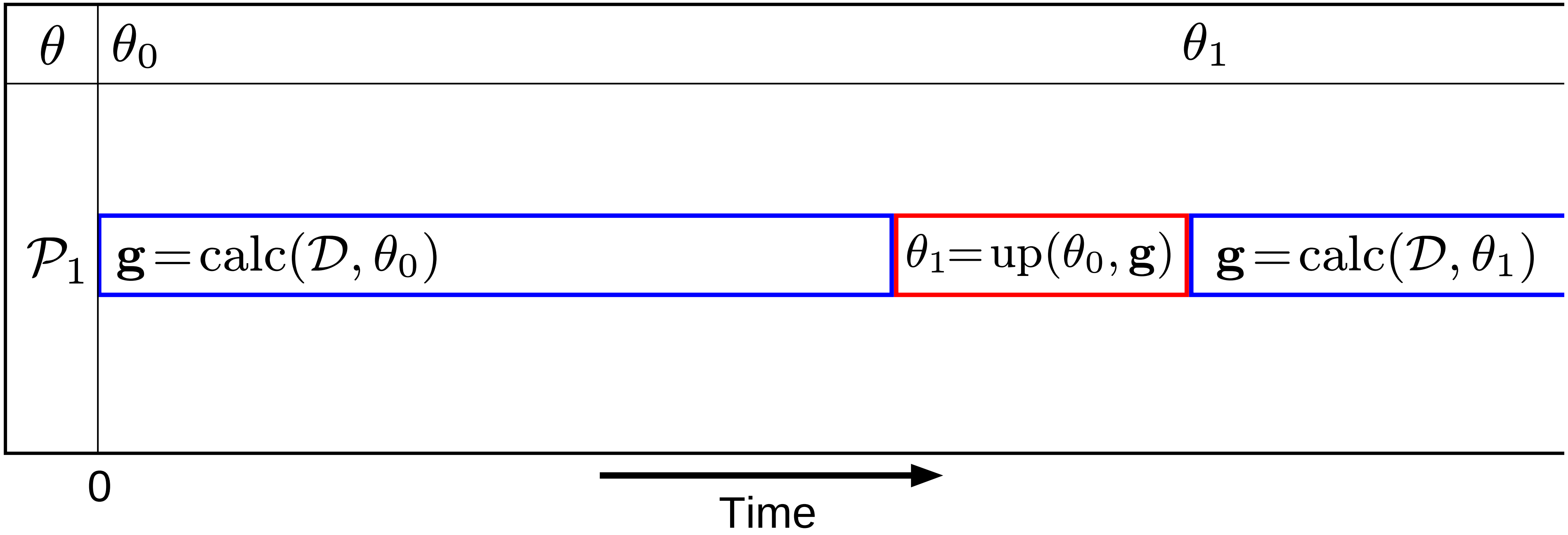
Calculate gradient $\mathbf{g}$ on data $\mathcal{D}$ using parameters θ

$\theta_1 = \text{up}(\theta_0, \mathbf{g})$:

Update θ_0 using gradient $\mathbf{g}$ to obtain θ_1

Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Dataset:	$\mathcal{D}$

Single-Processor Batch Learning



$\mathbf{g} = \text{calc}(\mathcal{D}, \theta)$:

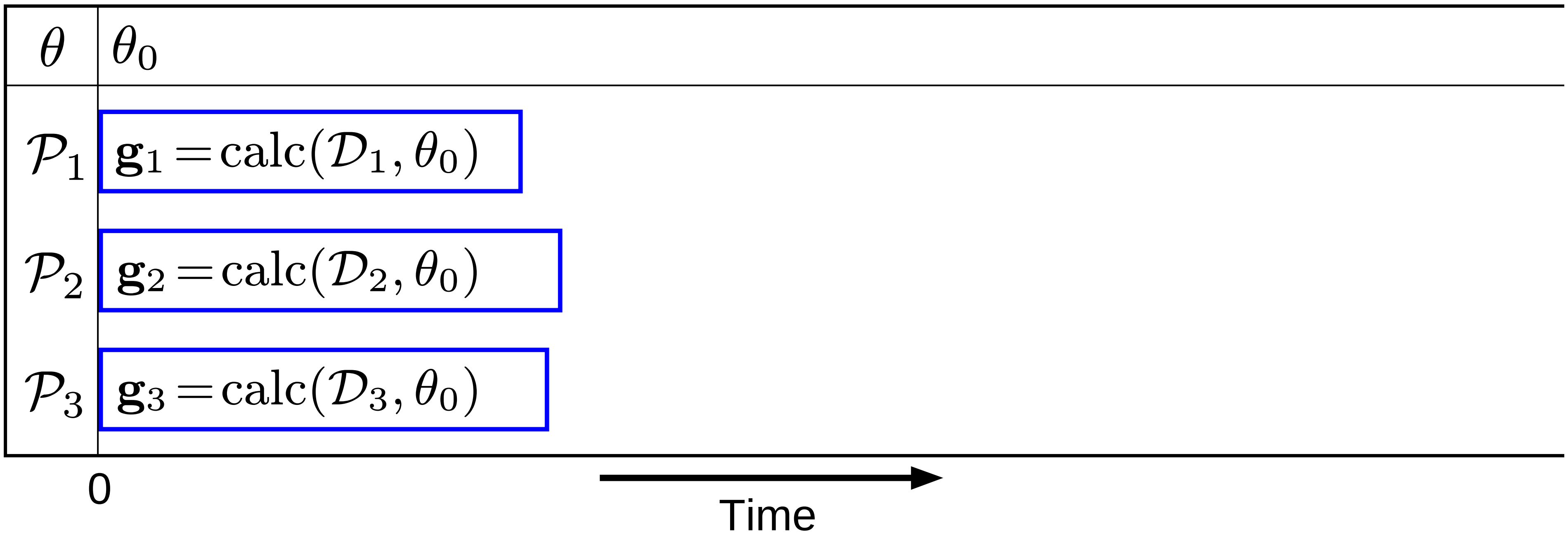
Calculate gradient $\mathbf{g}$ on data $\mathcal{D}$ using parameters θ

$\theta_1 = \text{up}(\theta_0, \mathbf{g})$:

Update θ_0 using gradient $\mathbf{g}$ to obtain θ_1

Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Dataset:	$\mathcal{D}$

Parallel Batch Learning



- Divide data into parts, compute gradient on parts in parallel

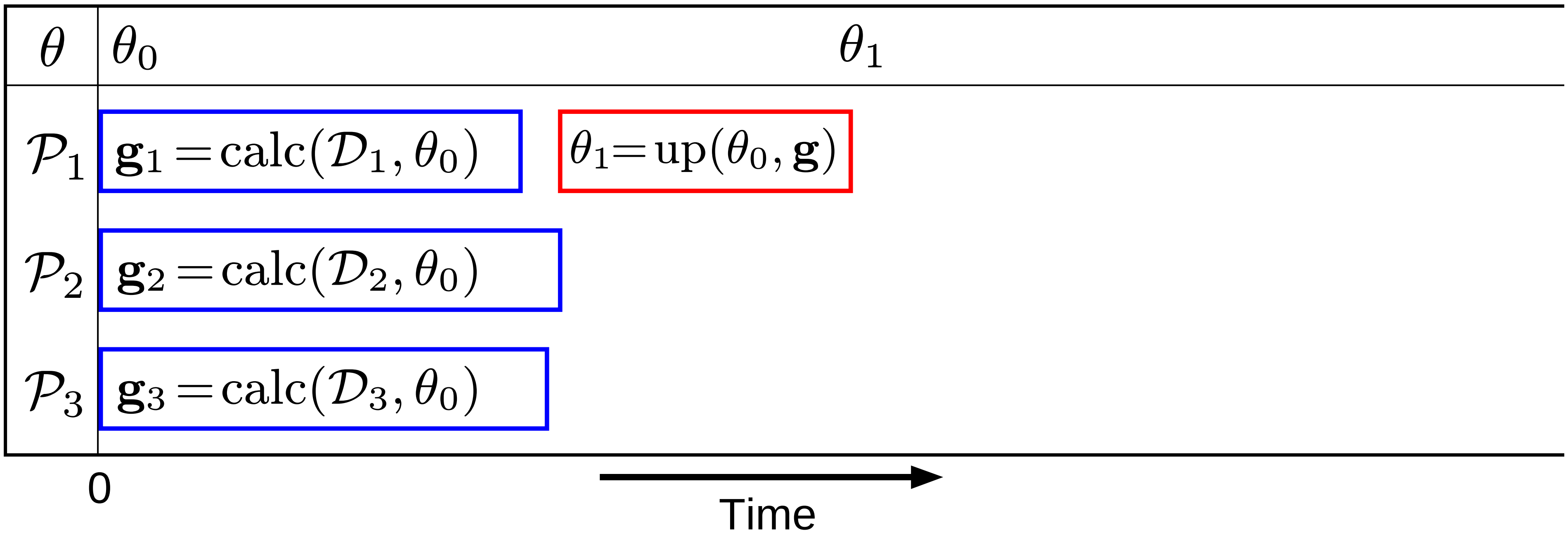
Parameters: θ_t

Processors: $\mathcal{P}_i$

Dataset: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

Parallel Batch Learning



- Divide data into parts, compute gradient on parts in parallel
- One processor updates parameters

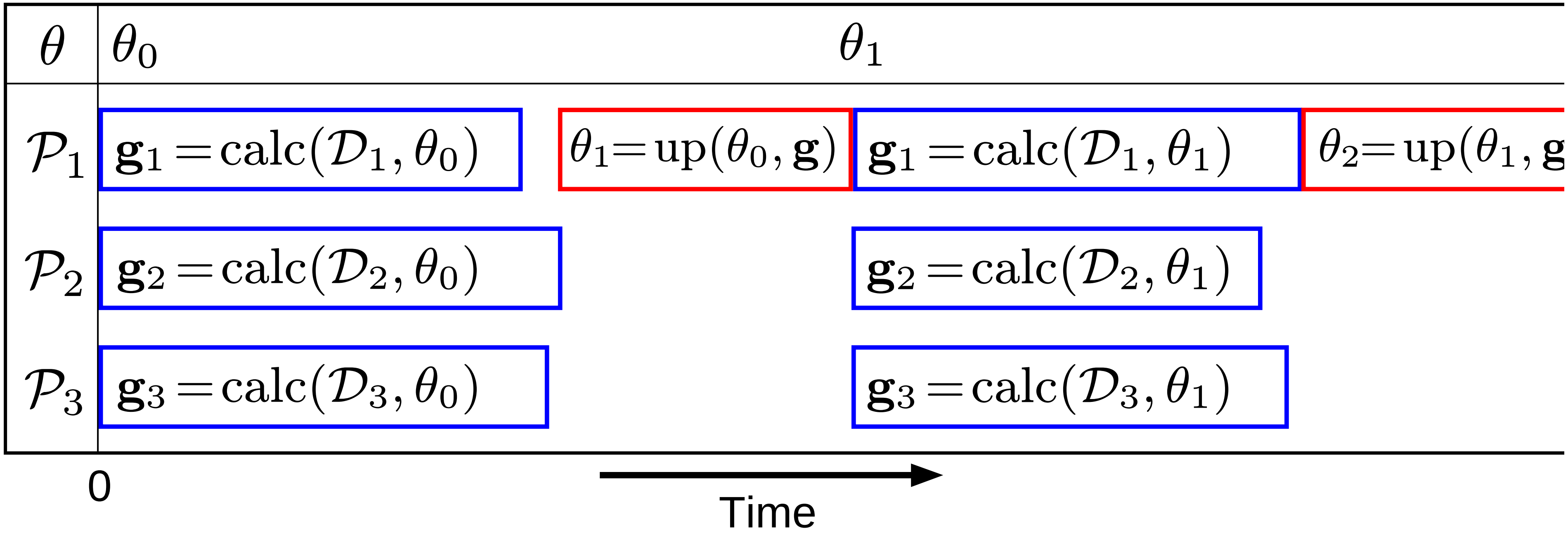
Parameters: θ_t

Processors: $\mathcal{P}_i$

Dataset: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

Parallel Batch Learning



- Divide data into parts, compute gradient on parts in parallel
- One processor updates parameters

Parameters: θ_t

Processors: $\mathcal{P}_i$

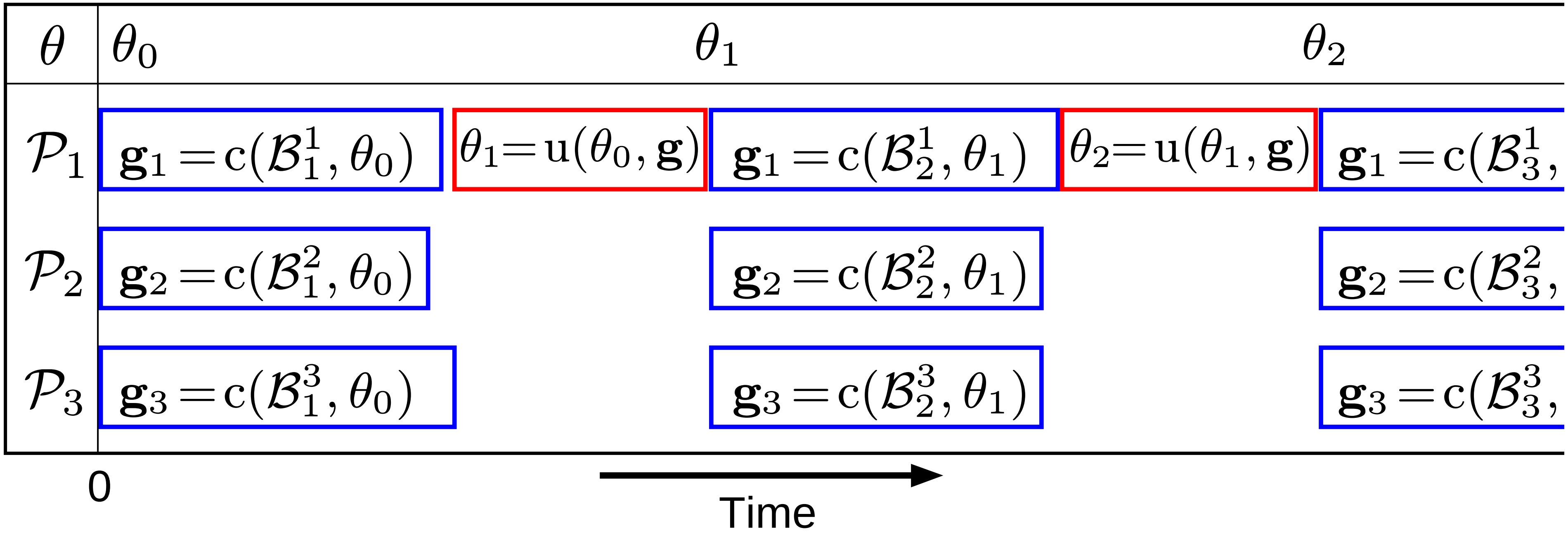
Dataset: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

ARK lti Carnegie Mellon

Parallel Synchronous Mini-Batch Learning

Finkel, Kleeman, and Manning (2008)



- Same architecture, just more frequent updates

Parameters: θ_t

Processors: $\mathcal{P}_i$

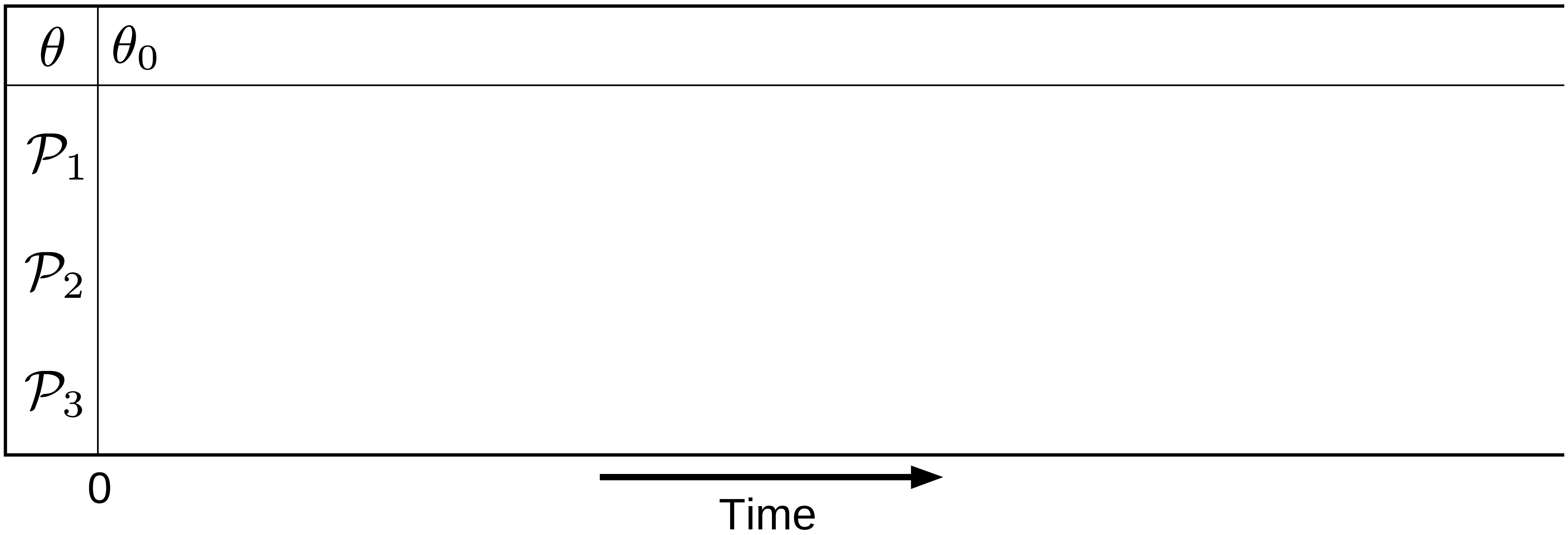
Mini-batches: $\mathcal{B}_t = \mathcal{B}_t^1 \cup \mathcal{B}_t^2 \cup \mathcal{B}_t^3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

ARK lti Carnegie Mellon

Parallel Asynchronous Mini-Batch Learning

Nedic, Bertsekas, and Borkar (2001)



Parameters: θ_t

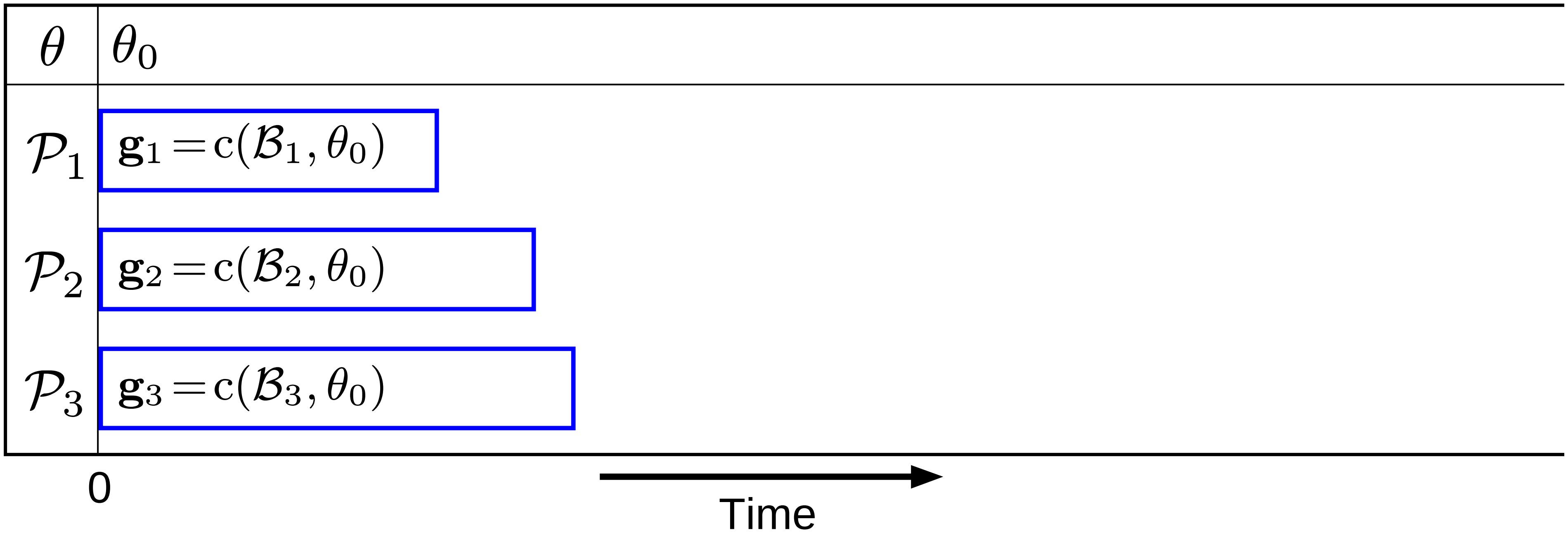
Processors: $\mathcal{P}_i$

Mini-batches: $\mathcal{B}_j$

Gradient: $\mathbf{g}_k$

Parallel Asynchronous Mini-Batch Learning

Nedic, Bertsekas, and Borkar (2001)



Parameters: θ_t

Processors: $\mathcal{P}_i$

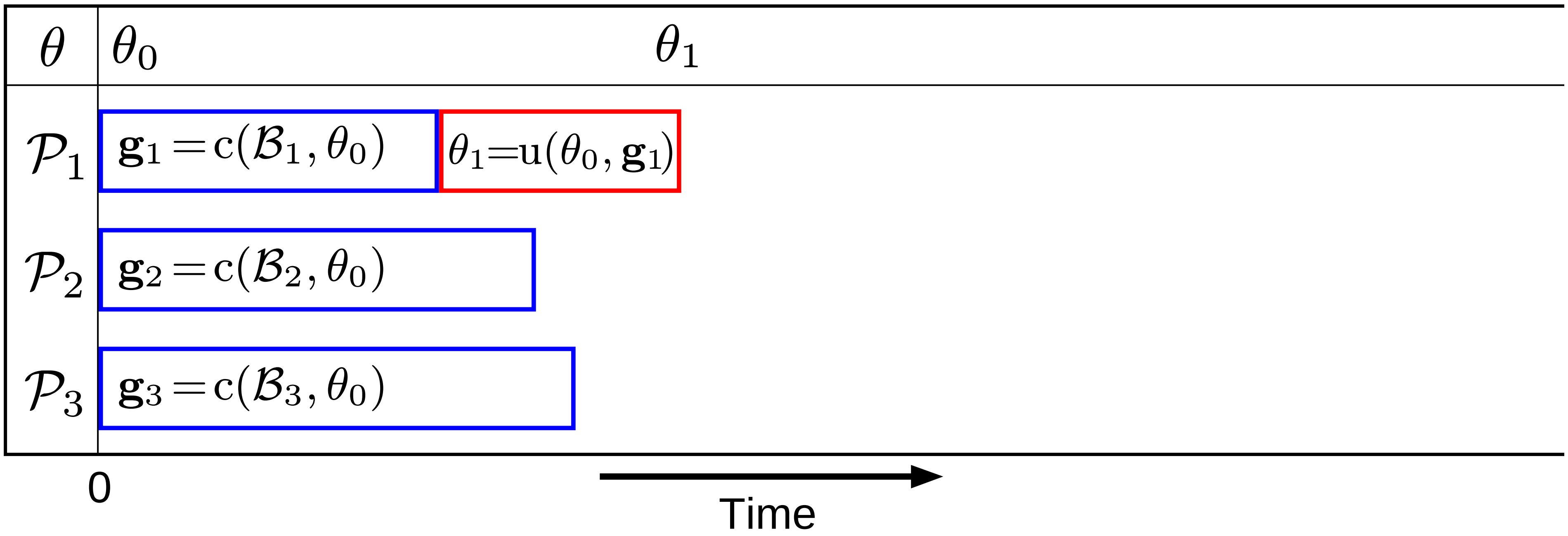
Mini-batches: $\mathcal{B}_j$

Gradient: $\mathbf{g}_k$

ARK lti Carnegie Mellon

Parallel Asynchronous Mini-Batch Learning

Nedic, Bertsekas, and Borkar (2001)



Parameters: θ_t

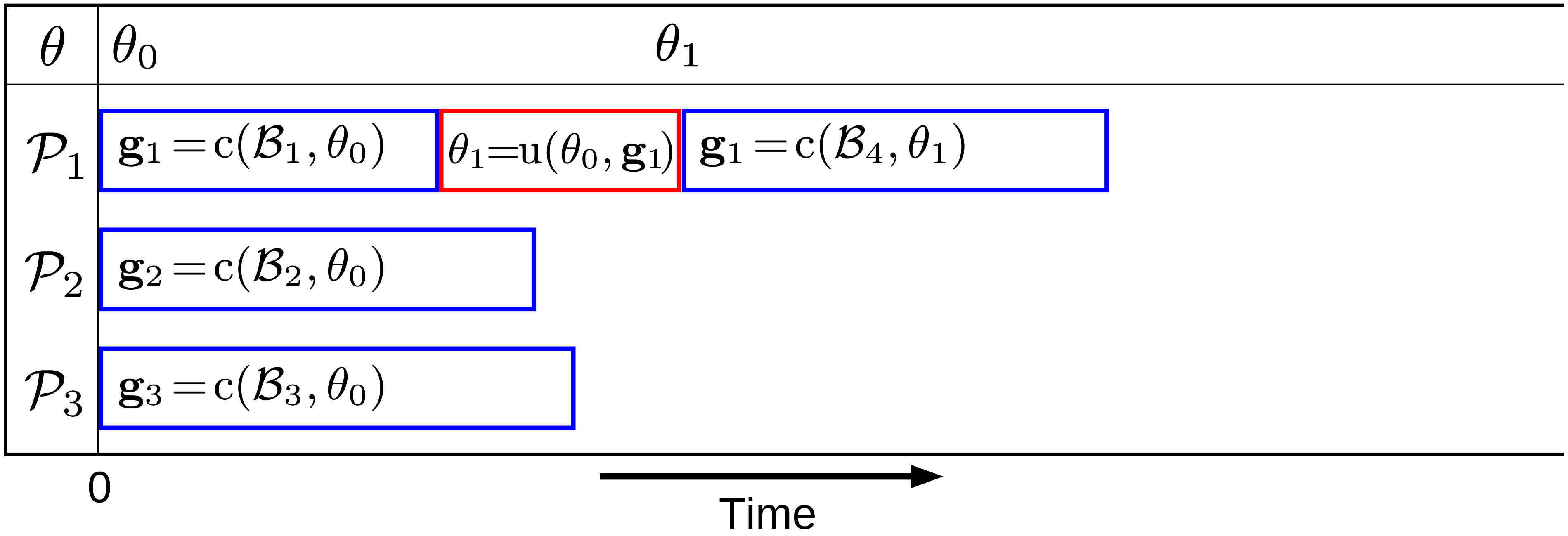
Processors: $\mathcal{P}_i$

Mini-batches: $\mathcal{B}_j$

Gradient: $\mathbf{g}_k$

Parallel Asynchronous Mini-Batch Learning

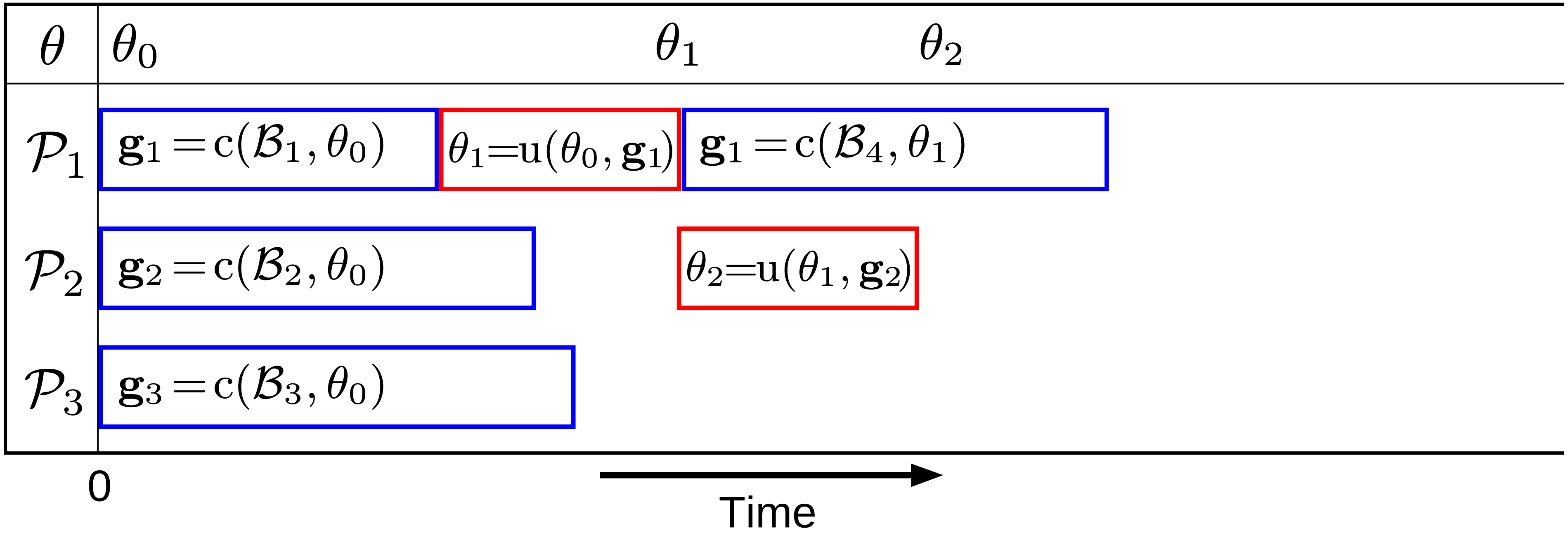
Nedic, Bertsekas, and Borkar (2001)



Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Mini-batches:	$\mathcal{B}_j$
Gradient:	$\mathbf{g}_k$

Parallel Asynchronous Mini-Batch Learning

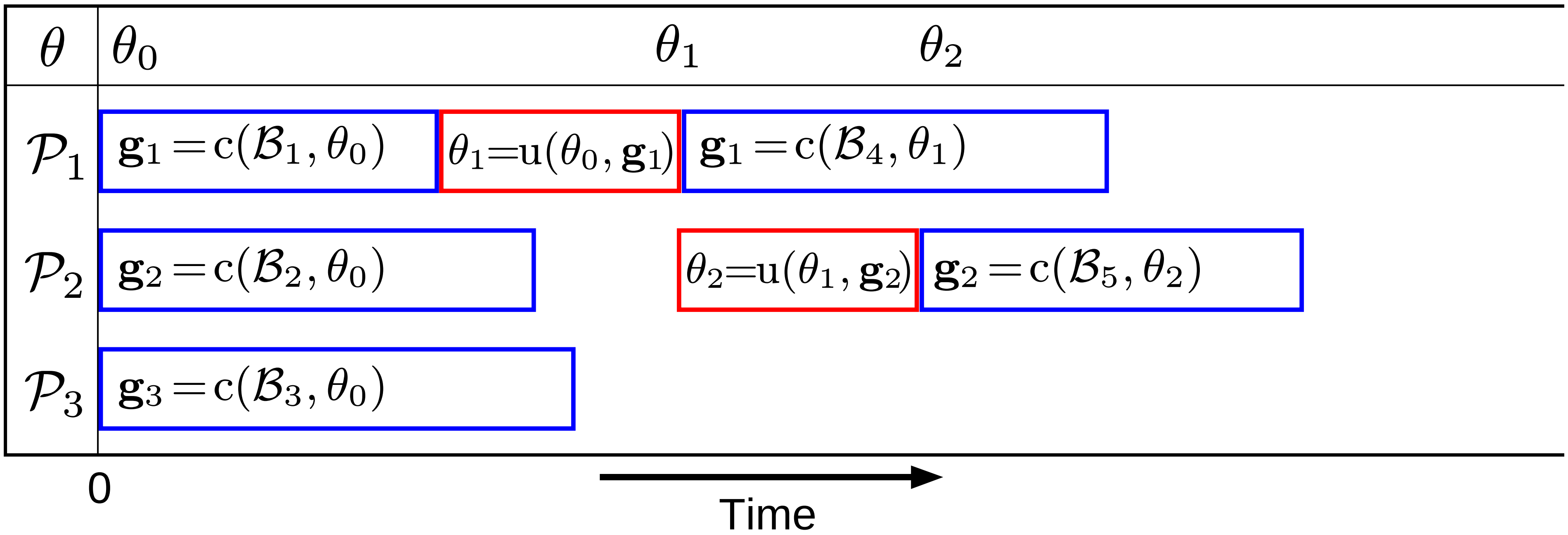
Nedic, Bertsekas, and Borkar (2001)



Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Mini-batches:	$\mathcal{B}_j$
Gradient:	$\mathbf{g}_k$

Parallel Asynchronous Mini-Batch Learning

Nedic, Bertsekas, and Borkar (2001)



Parameters: θ_t

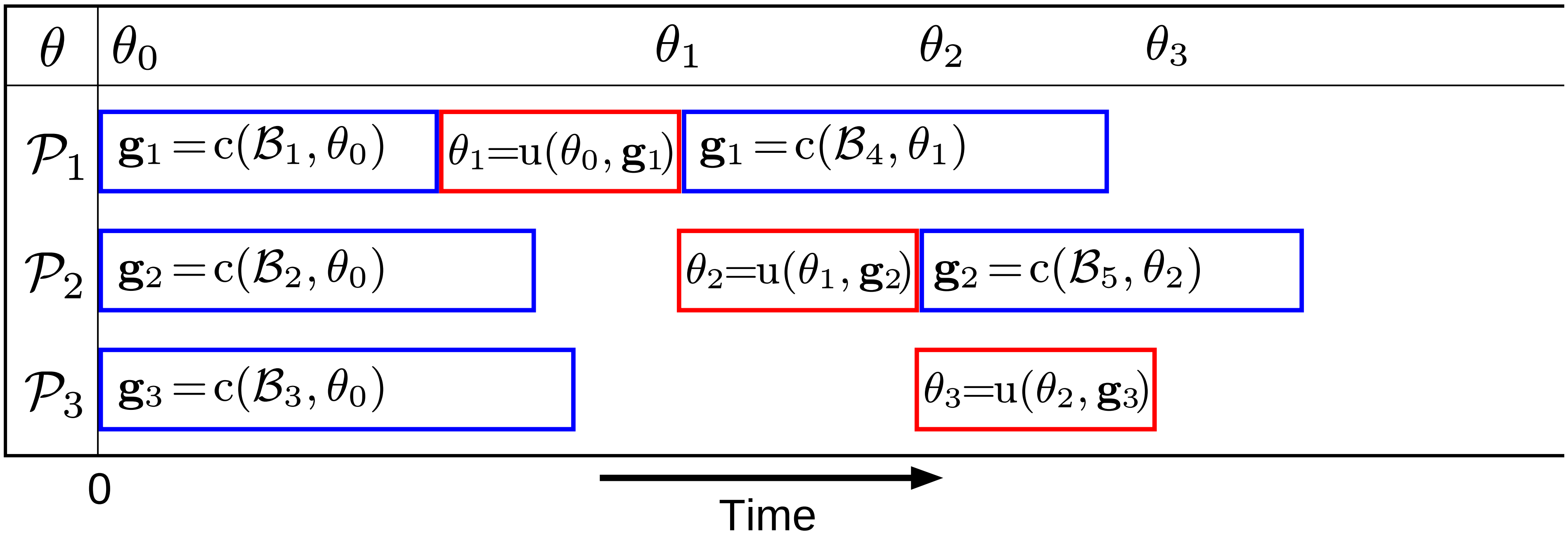
Processors: $\mathcal{P}_i$

Mini-batches: $\mathcal{B}_j$

Gradient: $\mathbf{g}_k$

Parallel Asynchronous Mini-Batch Learning

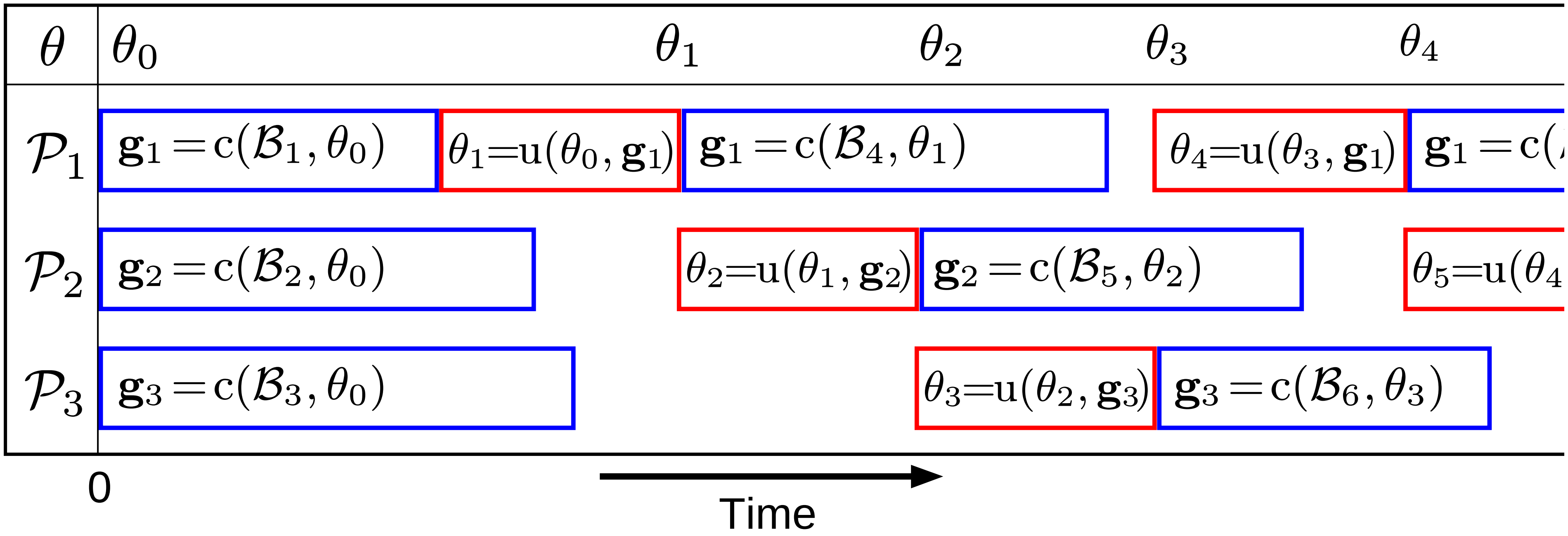
Nedic, Bertsekas, and Borkar (2001)



Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Mini-batches:	$\mathcal{B}_j$
Gradient:	$\mathbf{g}_k$

Parallel Asynchronous Mini-Batch Learning

Nedic, Bertsekas, and Borkar (2001)



- Gradients computed using stale parameters
- Increased processor utilization
- Only idle time caused by lock for updating parameters

Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Mini-batches:	$\mathcal{B}_j$
Gradient:	$\mathbf{g}_k$

ARK lti Carnegie Mellon

Theoretical Results

- How does the use of stale parameters affect convergence?

- Convergence results exist for convex optimization using stochastic gradient descent
  - Convergence guaranteed when max delay is bounded (Nedic, Bertsekas, and Borkar, 2001)
  - Convergence rates linear in max delay (Langford, Smola, and Zinkevich, 2009)

Experiments

Task	Model	Method	Convex?	$\|\mathcal{D}\|$	$\|\theta\|$	m
Named-Entity Recognition	CRF	Stochastic Gradient Descent	Y	15k	1.3M	4
Word Alignment	IBM Model 1	Stepwise EM	Y	300k	14.2M	10k
Unsupervised Part-of-Speech Tagging	HMM	Stepwise EM	N	42k	2M	4

- To compare algorithms, we use wall clock time (with a dedicated 4-processor machine)
- m = mini-batch size

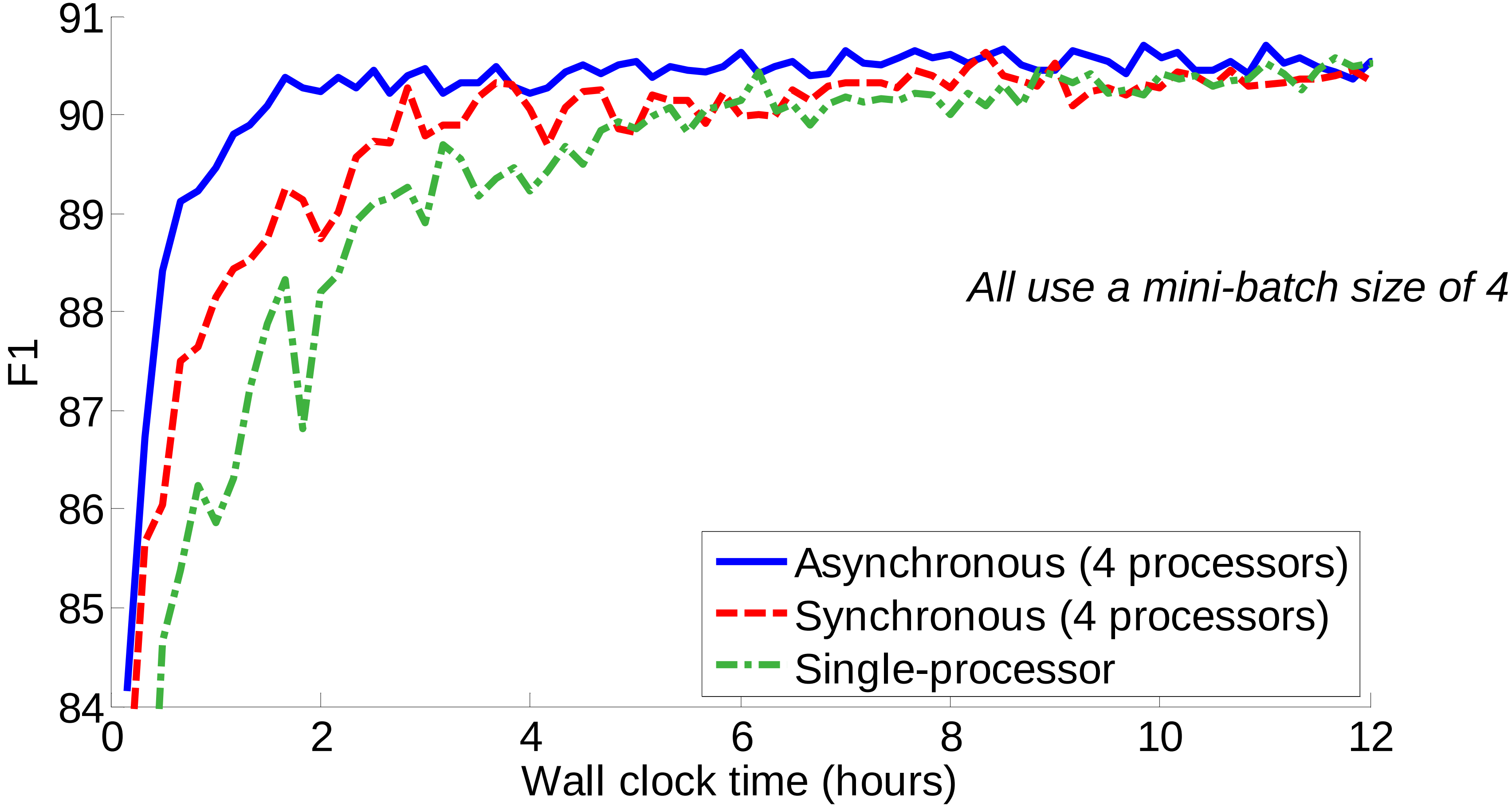
Experiments

Task	Model	Method	Convex?	$\lvert\mathcal{D}\rvert$	$\lvert\theta\rvert$	m
Named-Entity Recognition	CRF	Stochastic Gradient Descent	Y	15k	1.3M	4

- CoNLL 2003 English data
- Label each token with entity type (person, location, organization, or miscellaneous) or non-entity
- We show convergence in F1 on development data

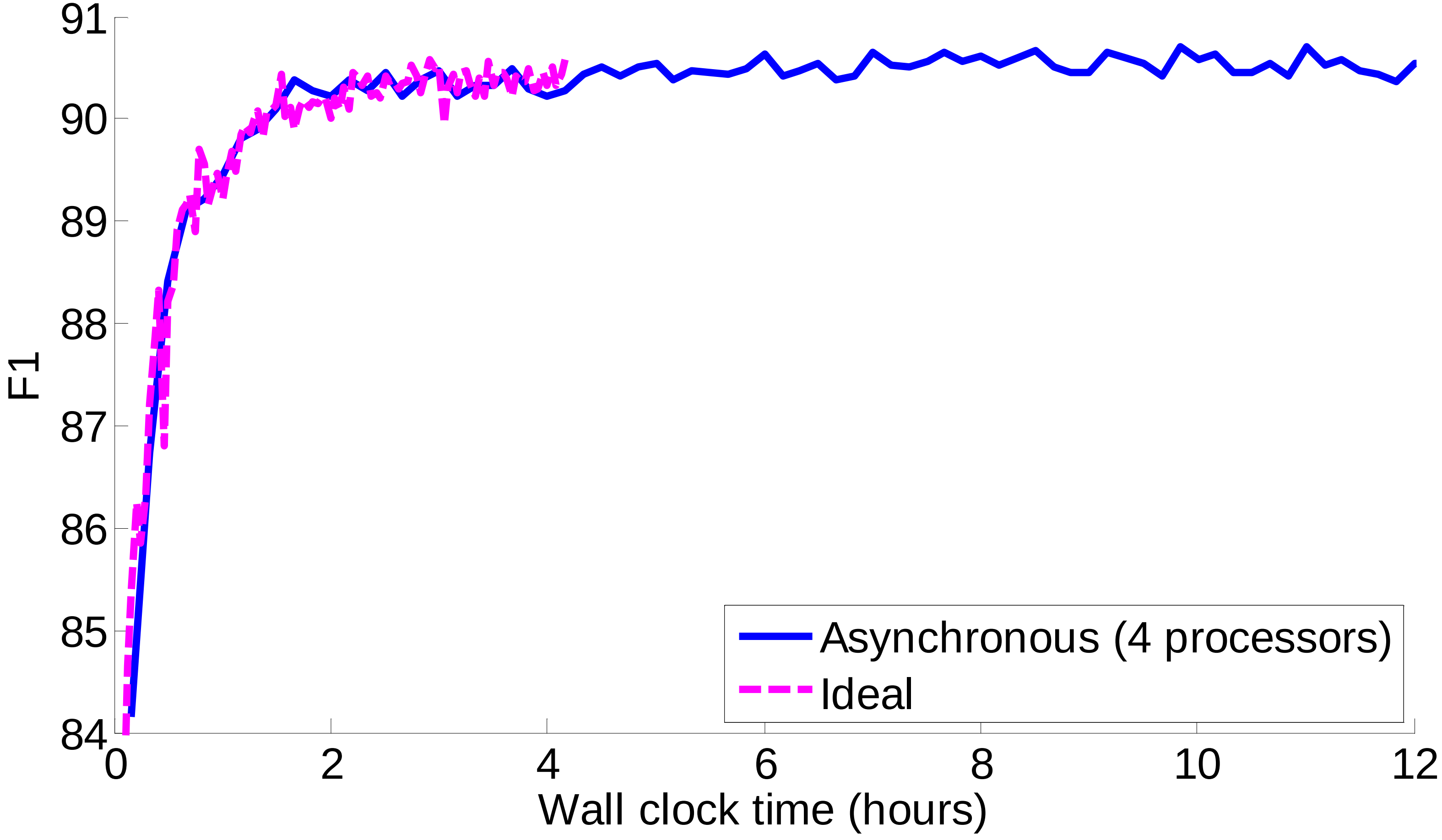
ARK lti Carnegie Mellon

Asynchronous Updating Speeds Convergence



Carnegie Mellon

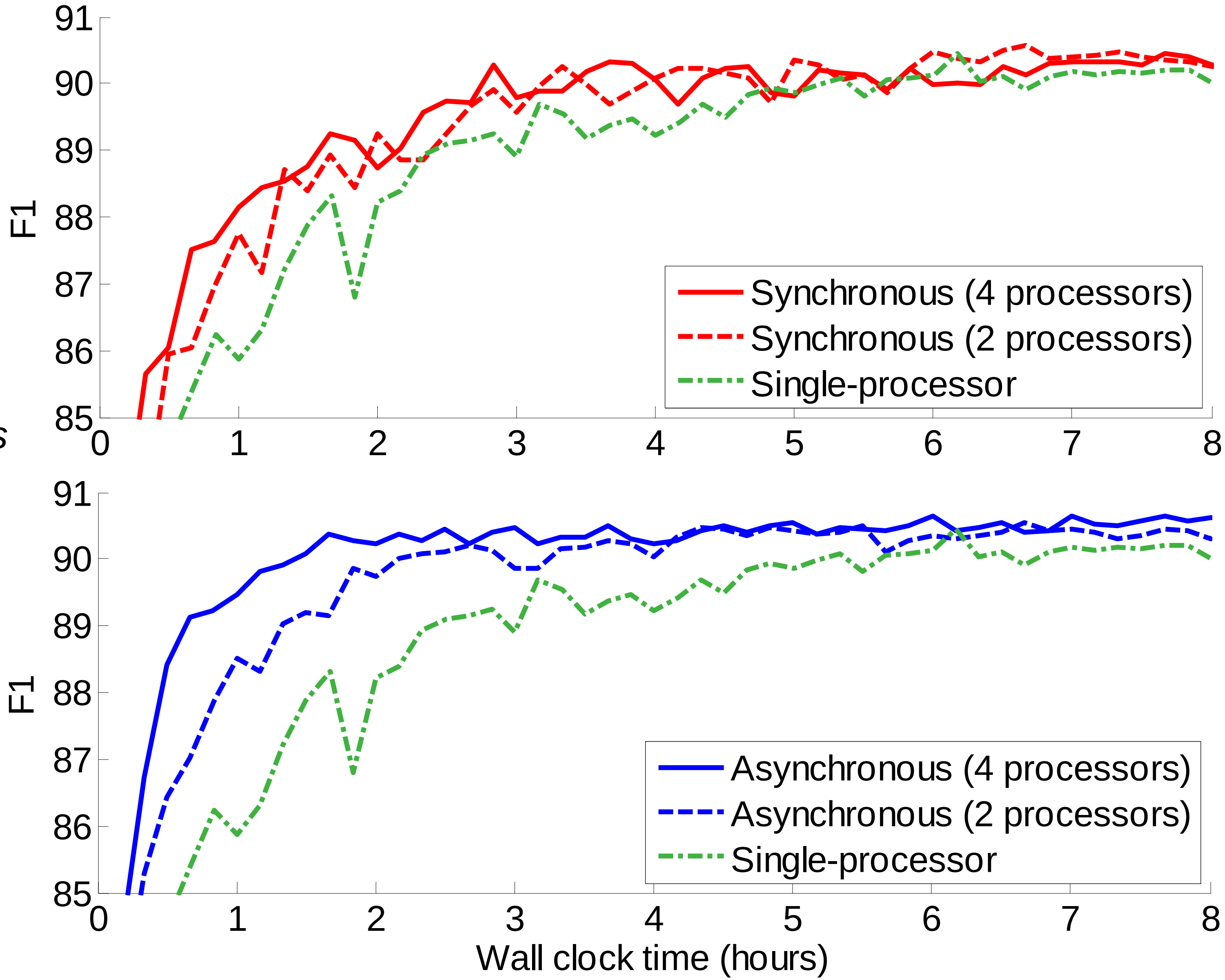
Comparison with Ideal Speed-up



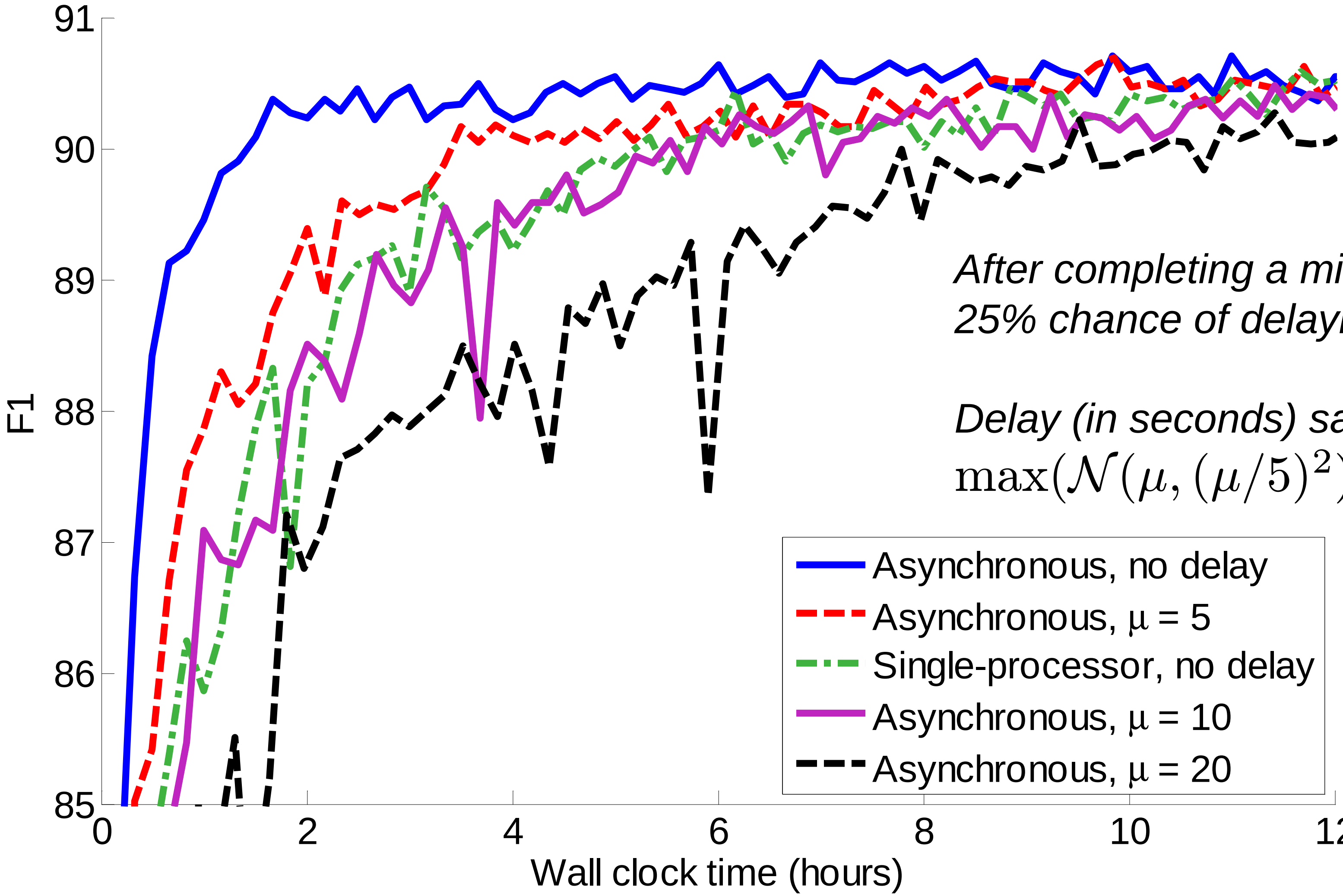
Why Does Asynchronous Converge Faster?

- Processors are kept in near-constant use
- Synchronous SGD leads to idle processors → need for load-balancing

Clearer improvement for asynchronous algorithms when increasing number of processors



Artificial Delays



After completing a mini-batch, 25% chance of delaying

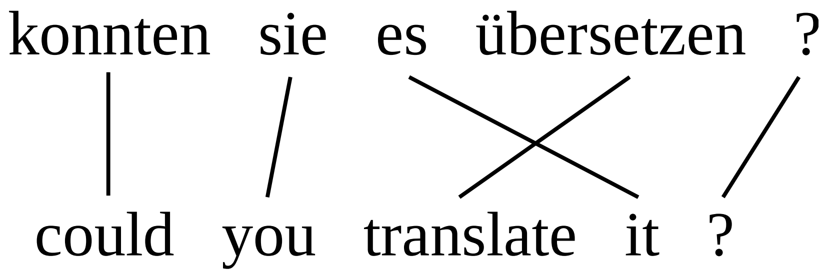
Delay (in seconds) sampled from
$$\max(\mathcal{N}(\mu, (\mu/5)^2), 0)$$

Avg. time per mini-batch = 0.62 s

Experiments

Task	Model	Method	Convex?	$\|\mathcal{D}\|$	$\|\theta\|$	$\boldsymbol{m}$
Word Alignment	IBM Model 1	Stepwise EM	Y	300k	14.2M	10k

- Given parallel sentences, draw links between words:



- We show convergence in log-likelihood (convergence in AER is similar)

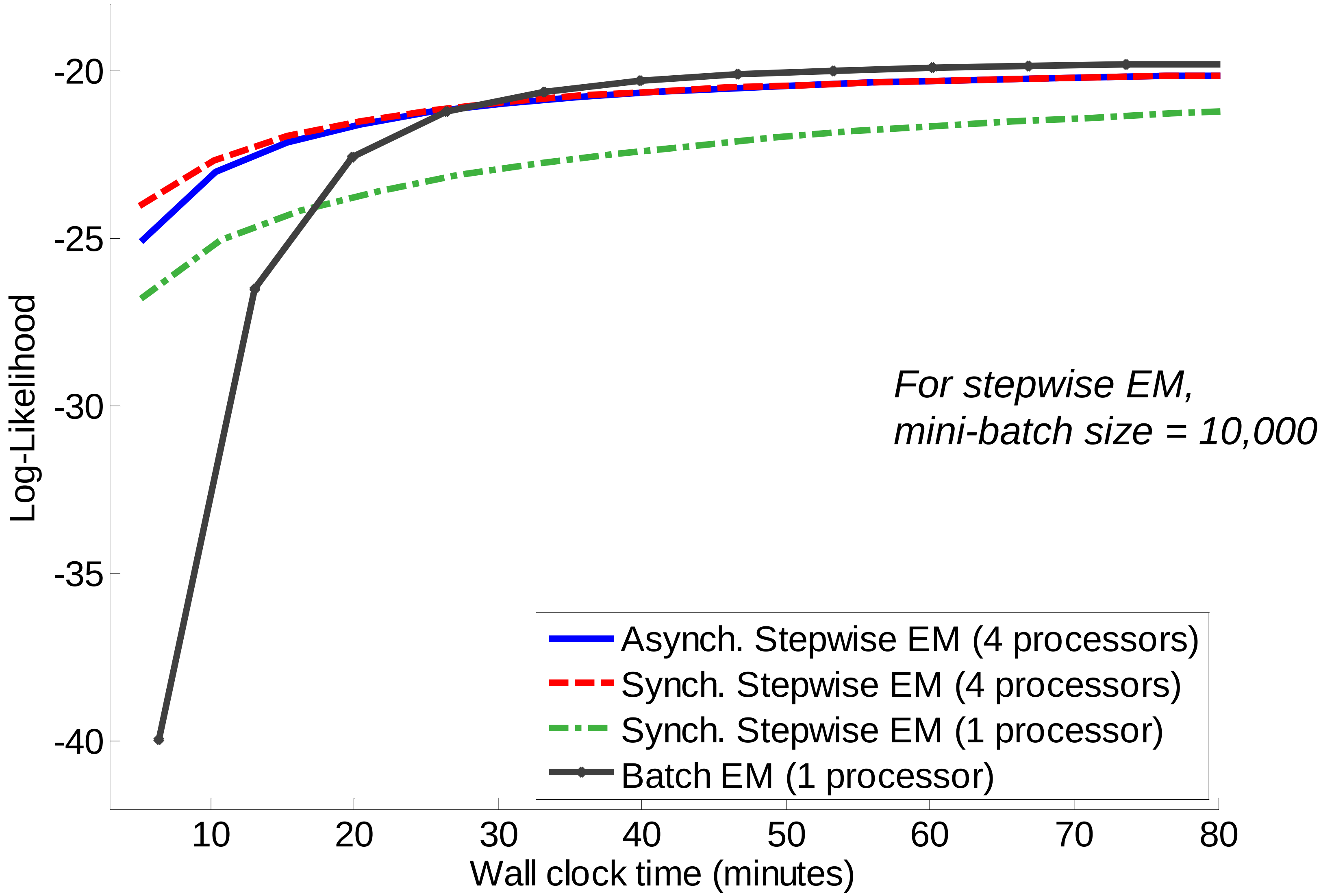
Stepwise EM

(Sato and Ishii, 2000; Cappe and Moulines, 2009)

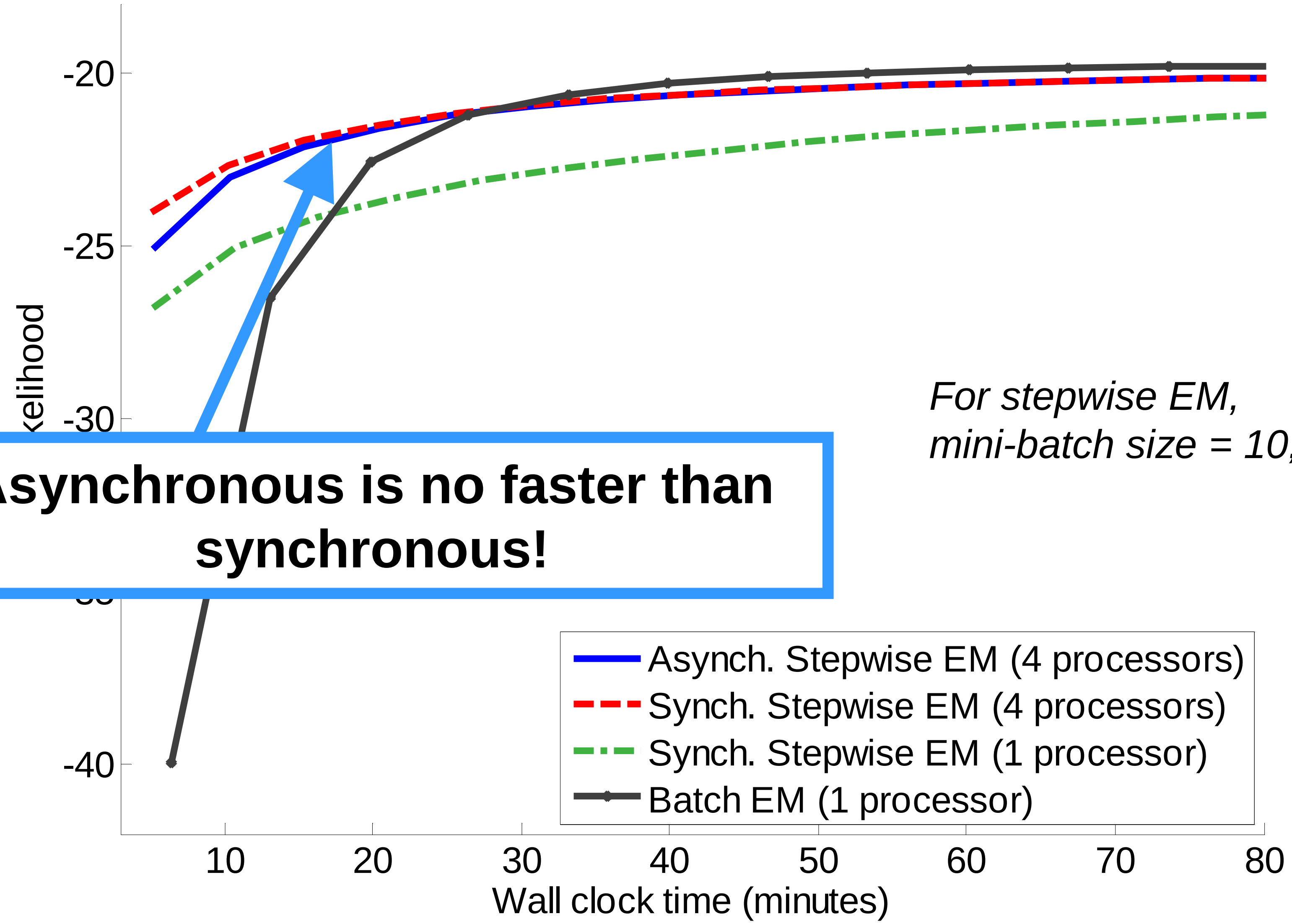
- Similar to stochastic gradient descent in the space of sufficient statistics, with a particular scaling of the update
- More efficient than incremental EM

  (Neal and Hinton, 1998)
- Found to converge much faster than batch EM (Liang and Klein, 2009)

Word Alignment Results



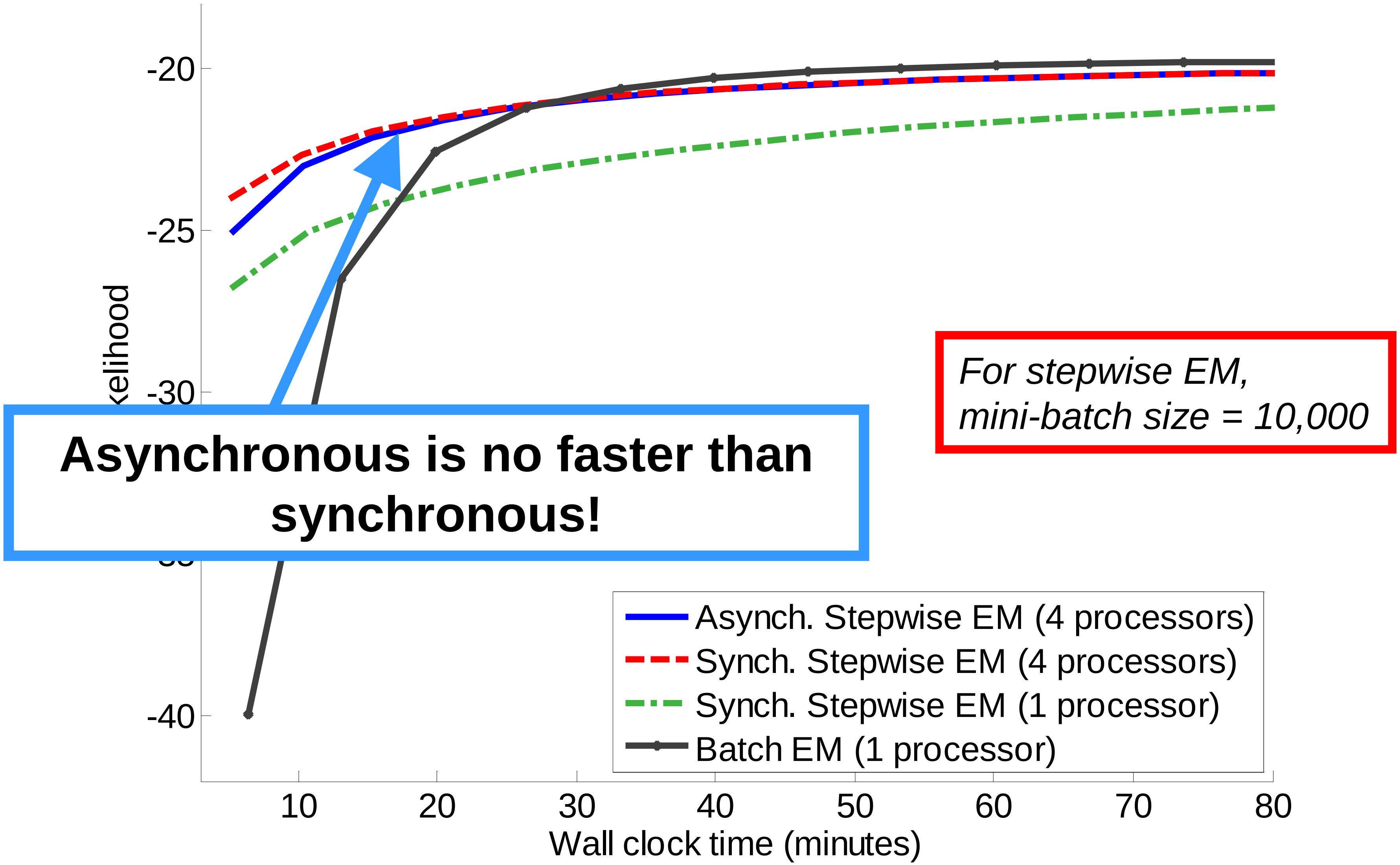
Word Alignment Results



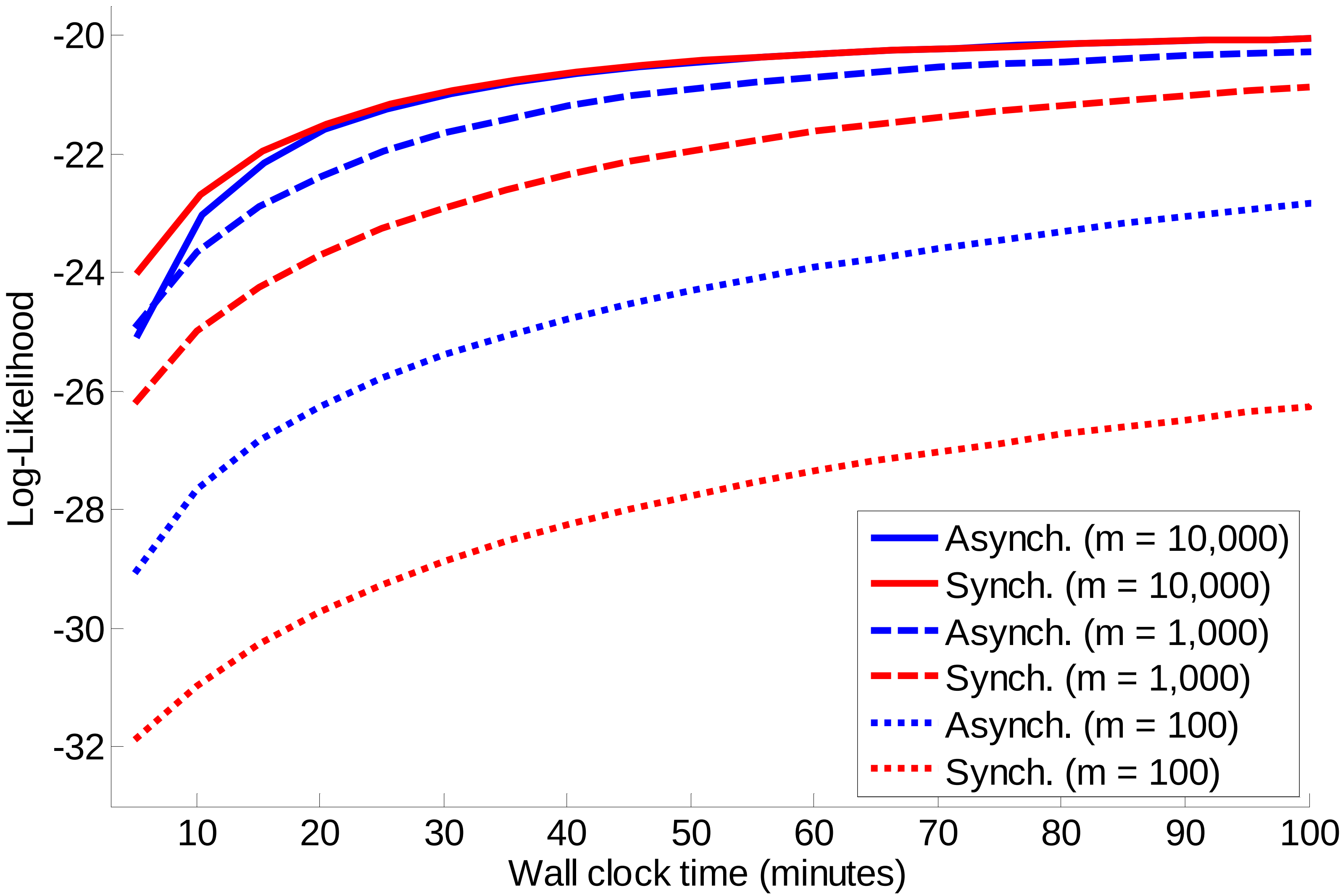
For stepwise EM,
mini-batch size = 10,000

Asynchronous is no faster than synchronous!

Word Alignment Results

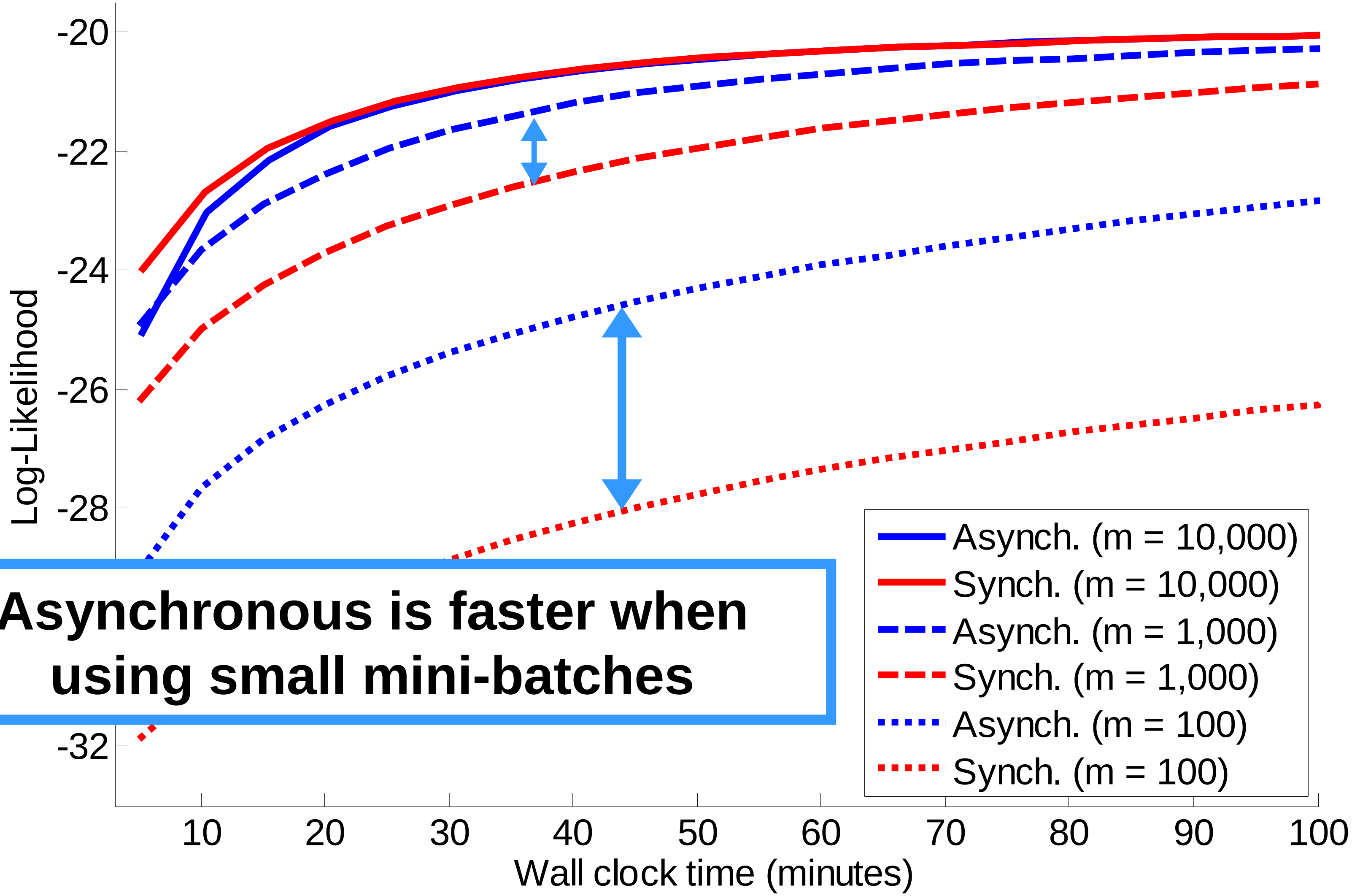


Comparing Mini-Batch Sizes



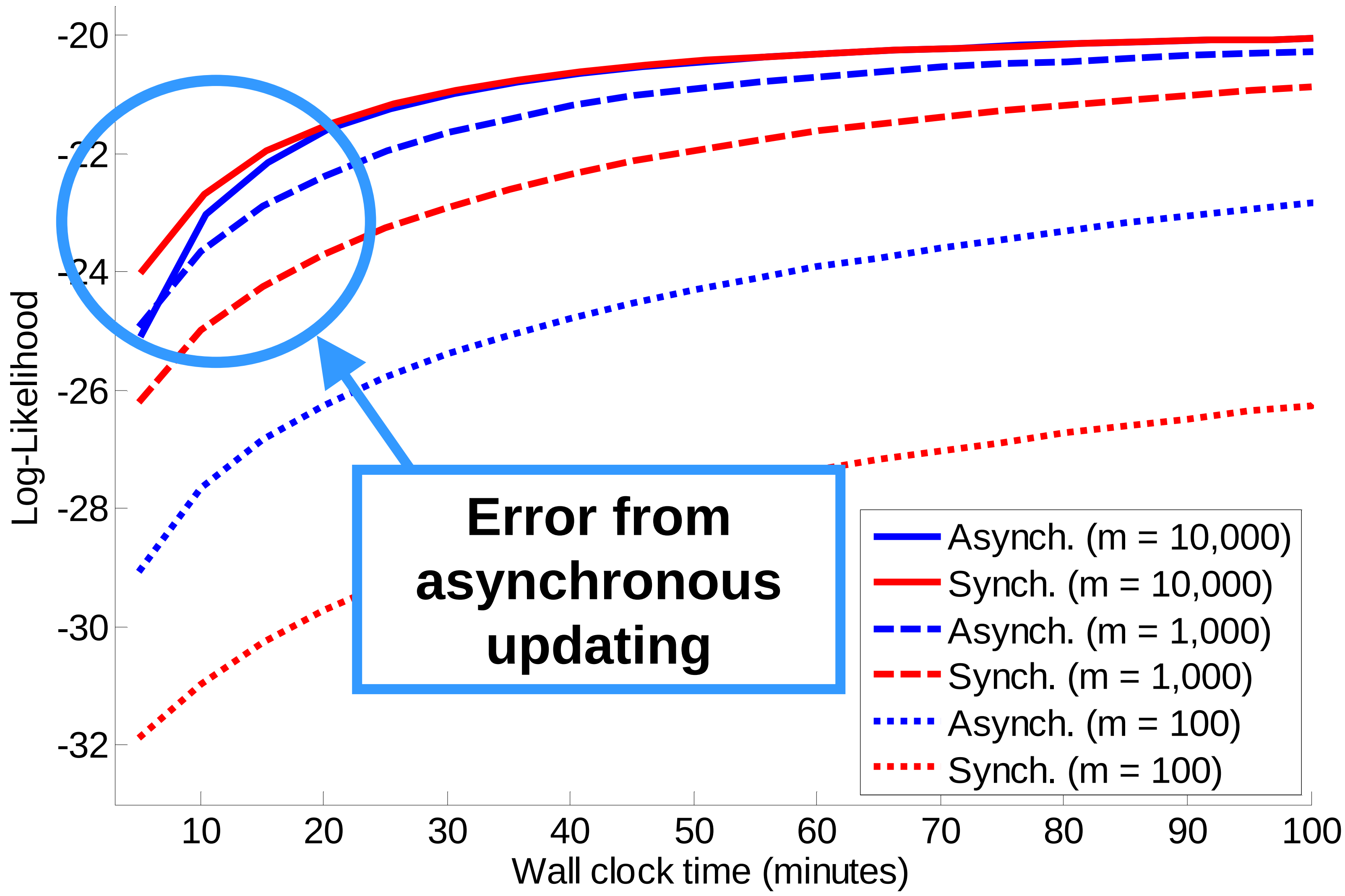
Carnegie Mellon

Comparing Mini-Batch Sizes



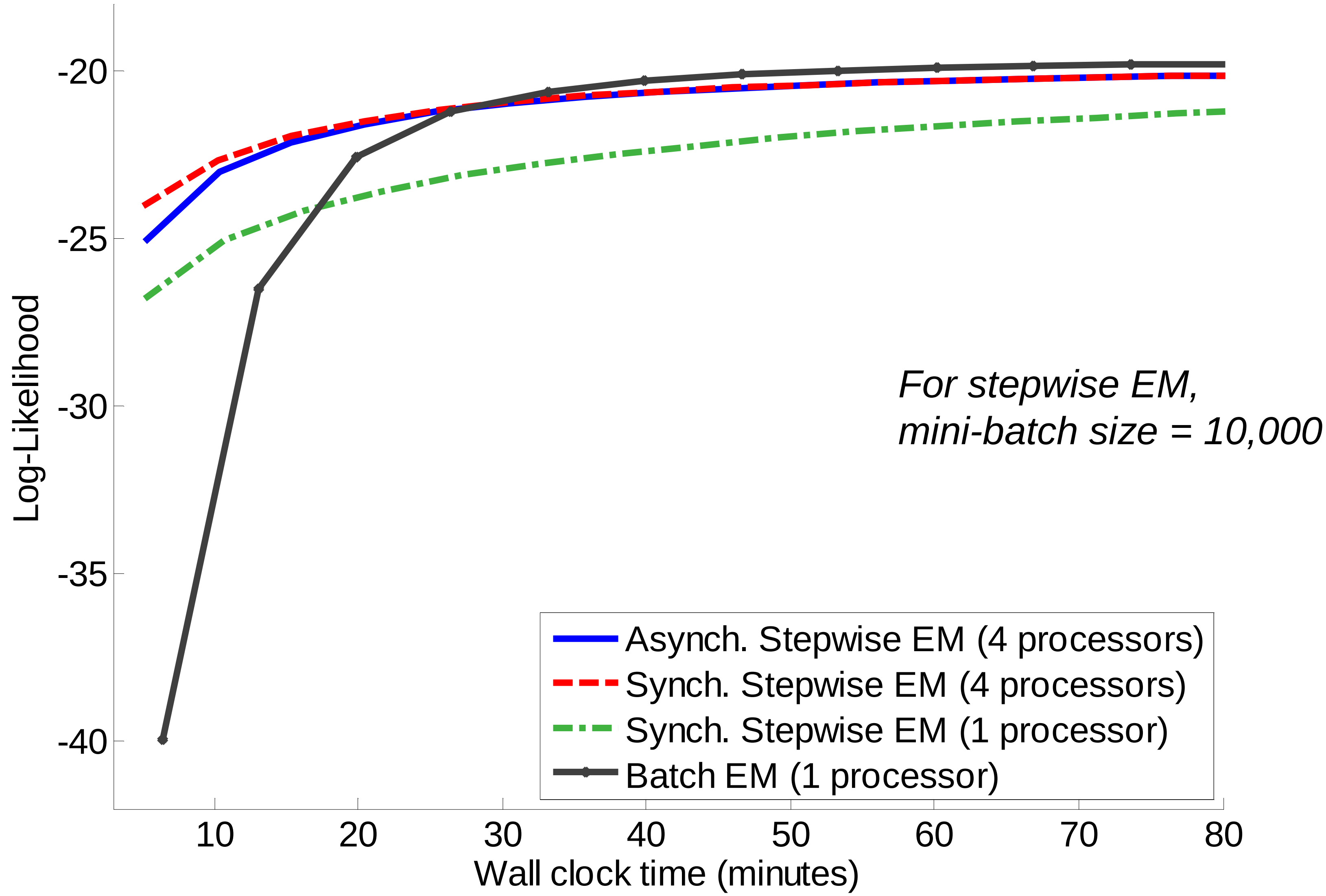
Asynchronous is faster when using small mini-batches

Comparing Mini-Batch Sizes

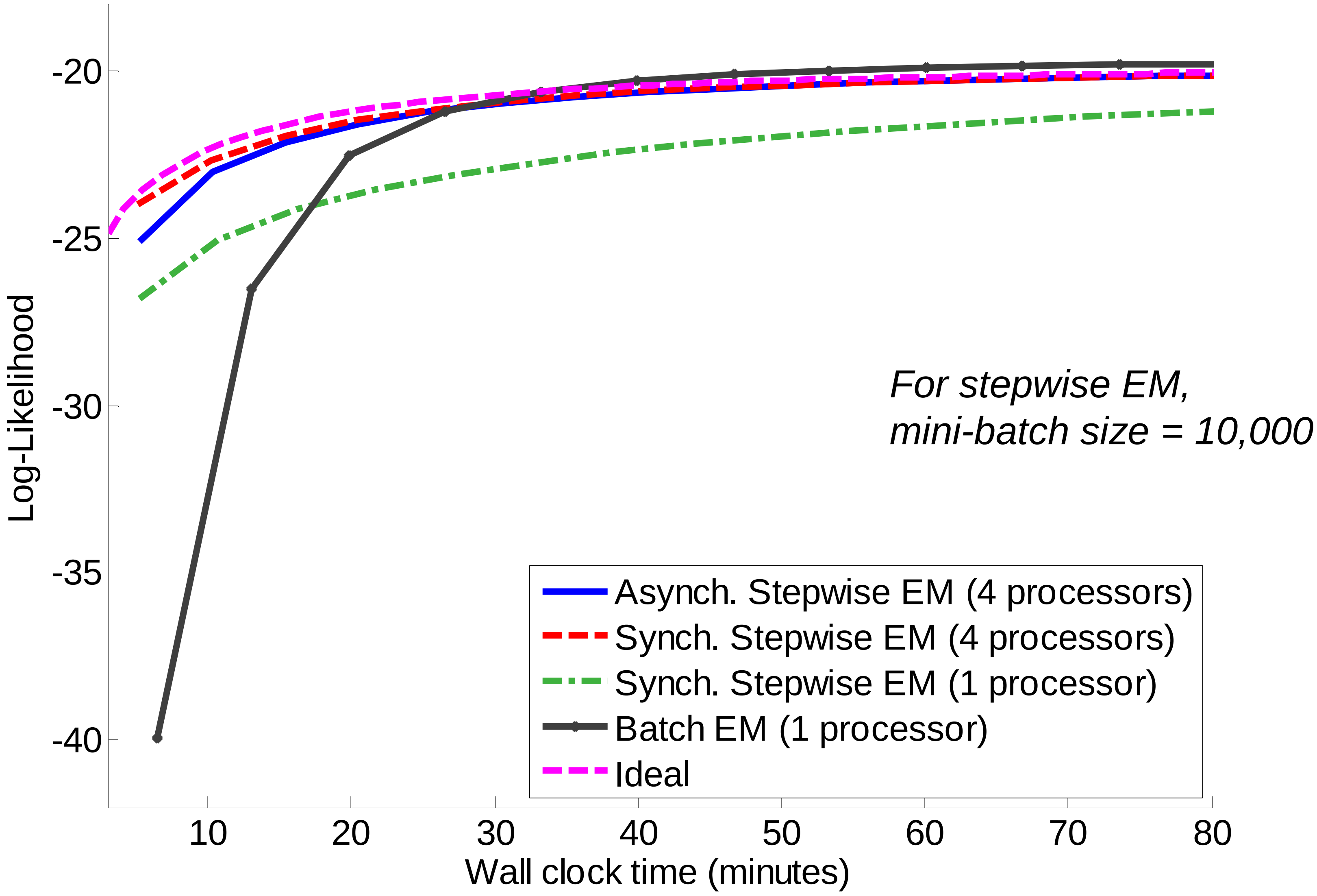


ARK lti Carnegie Mellon

Word Alignment Results



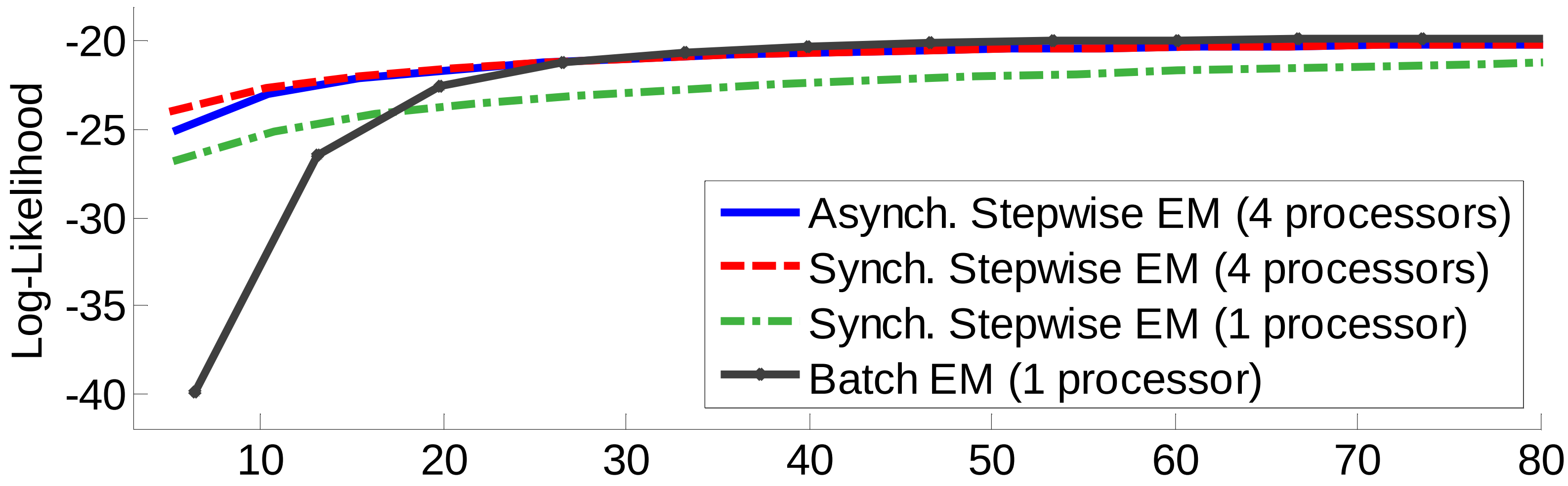
Comparison with Ideal Speed-up



MapReduce?

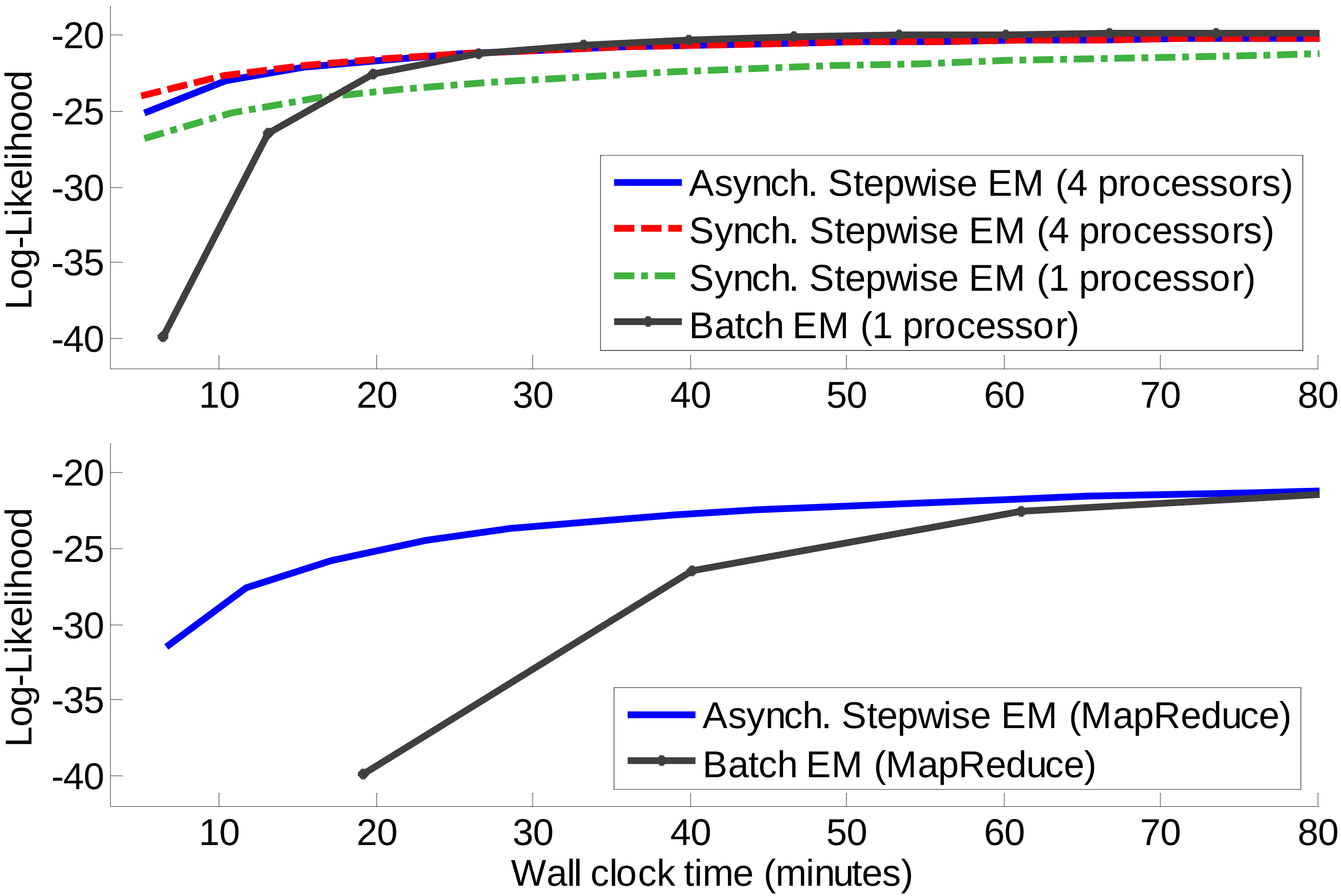
- We also ran these algorithms on a large MapReduce cluster (M45 from Yahoo!)
- Batch EM
  - Each iteration is one MapReduce job, using 24 mappers and 1 reducer
- Asynchronous Stepwise EM
  - 4 mini-batches processed simultaneously, each run as a MapReduce job
  - Each uses 6 mappers and 1 reducer

MapReduce?



Log-Likelihood
-20
-25
-30
-35
-40
10
20
30
40
50
60
70
80
Asynch. Stepwise EM (4 processors)
Synch. Stepwise EM (4 processors)
Synch. Stepwise EM (1 processor)
Batch EM (1 processor)

MapReduce?

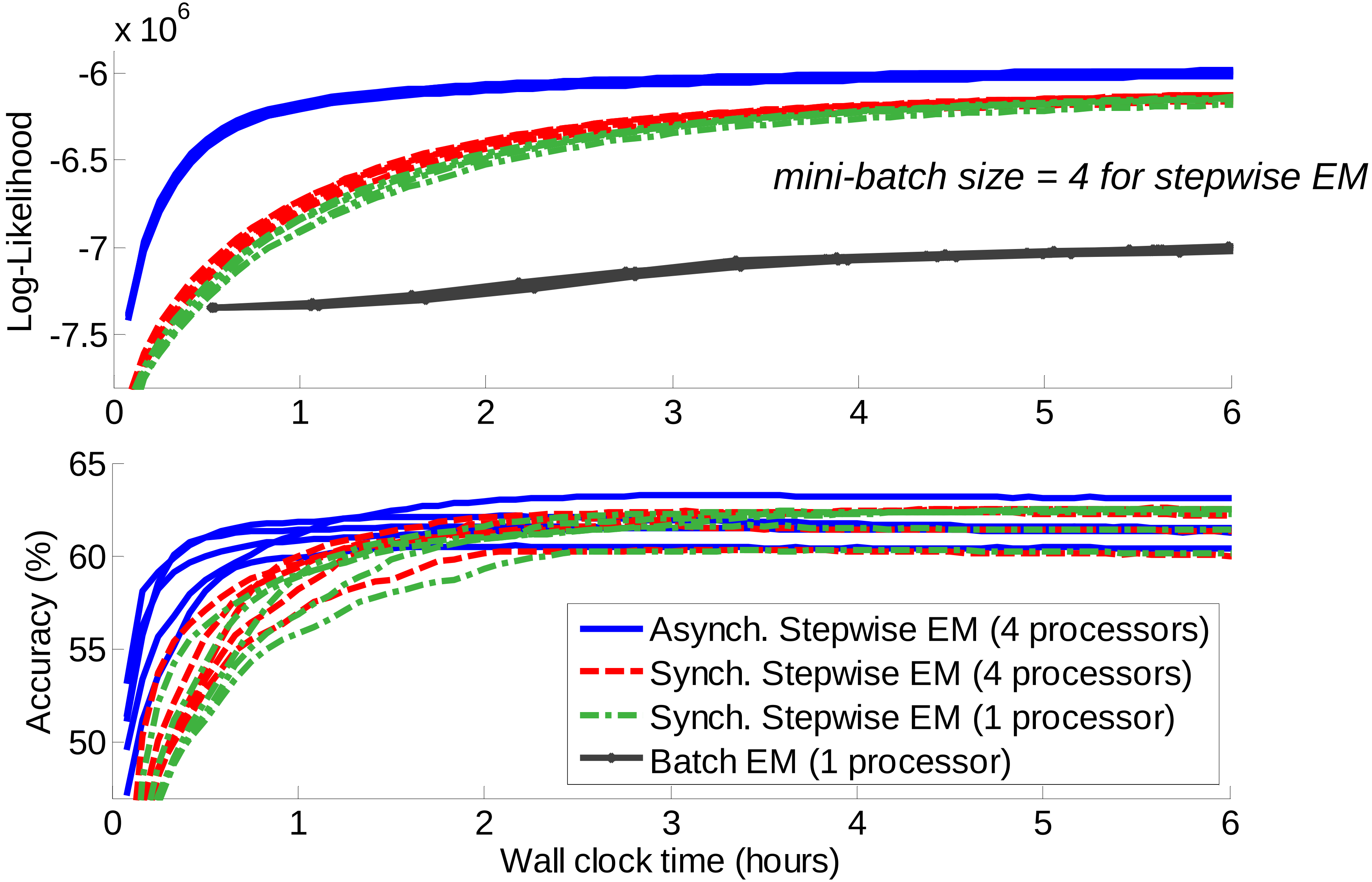

Log-Likelihood
-20
-25
-30
-35
-40
10
20
30
40
50
60
70
80
Asynch. Stepwise EM (4 processors)
Synch. Stepwise EM (4 processors)
Synch. Stepwise EM (1 processor)
Batch EM (1 processor)
Asynch. Stepwise EM (MapReduce)
Batch EM (MapReduce)
Wall clock time (minutes)

Experiments

Task	Model	Method	Convex?	$\lvert\mathcal{D}\rvert$	$\lvert\theta\rvert$	m
Unsupervised Part-of-Speech Tagging	HMM	Stepwise EM	N	42k	2M	4

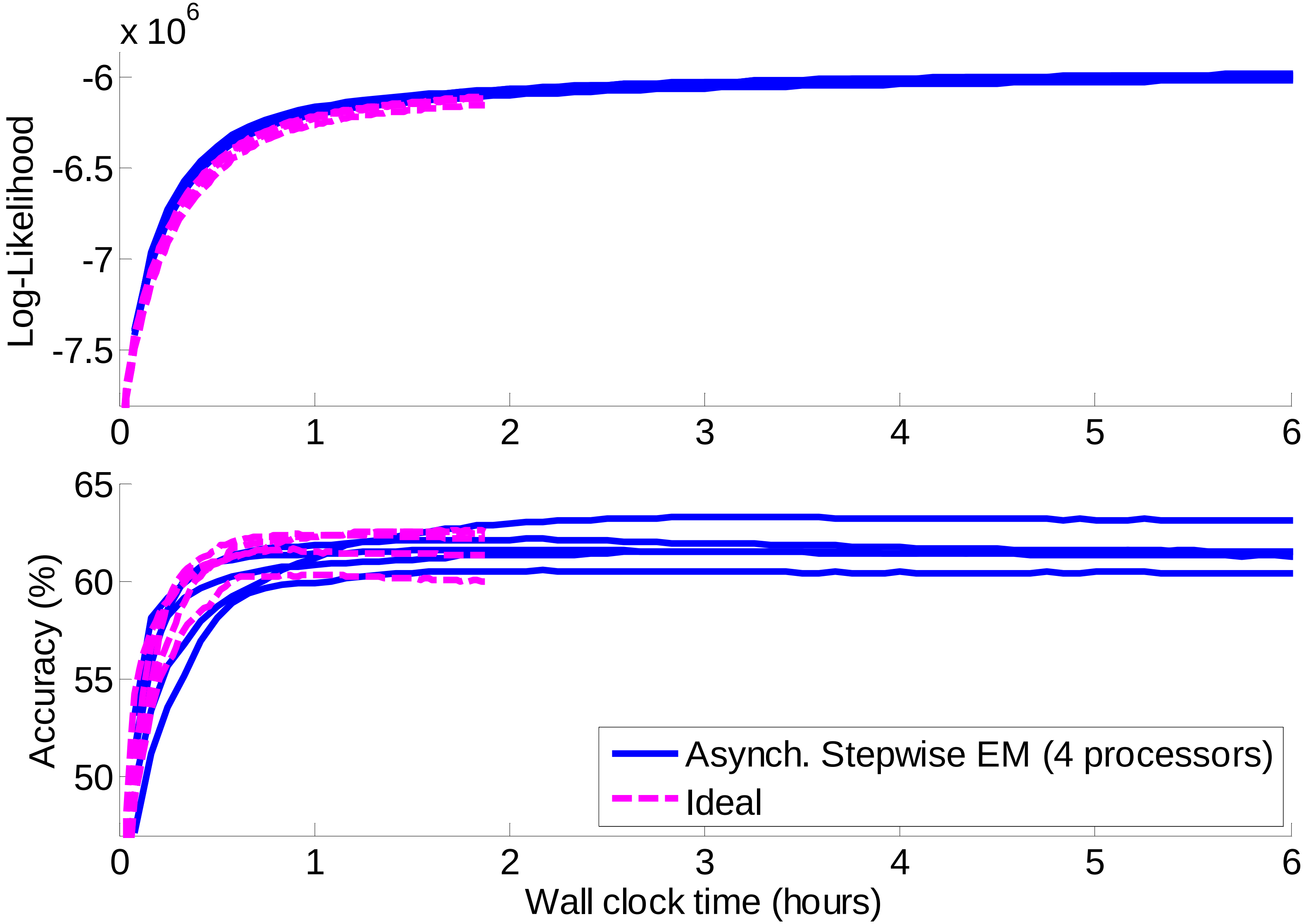
- Bigram HMM with 45 states
- We plot convergence in likelihood and many-to-1 accuracy

Part-of-Speech Tagging Results

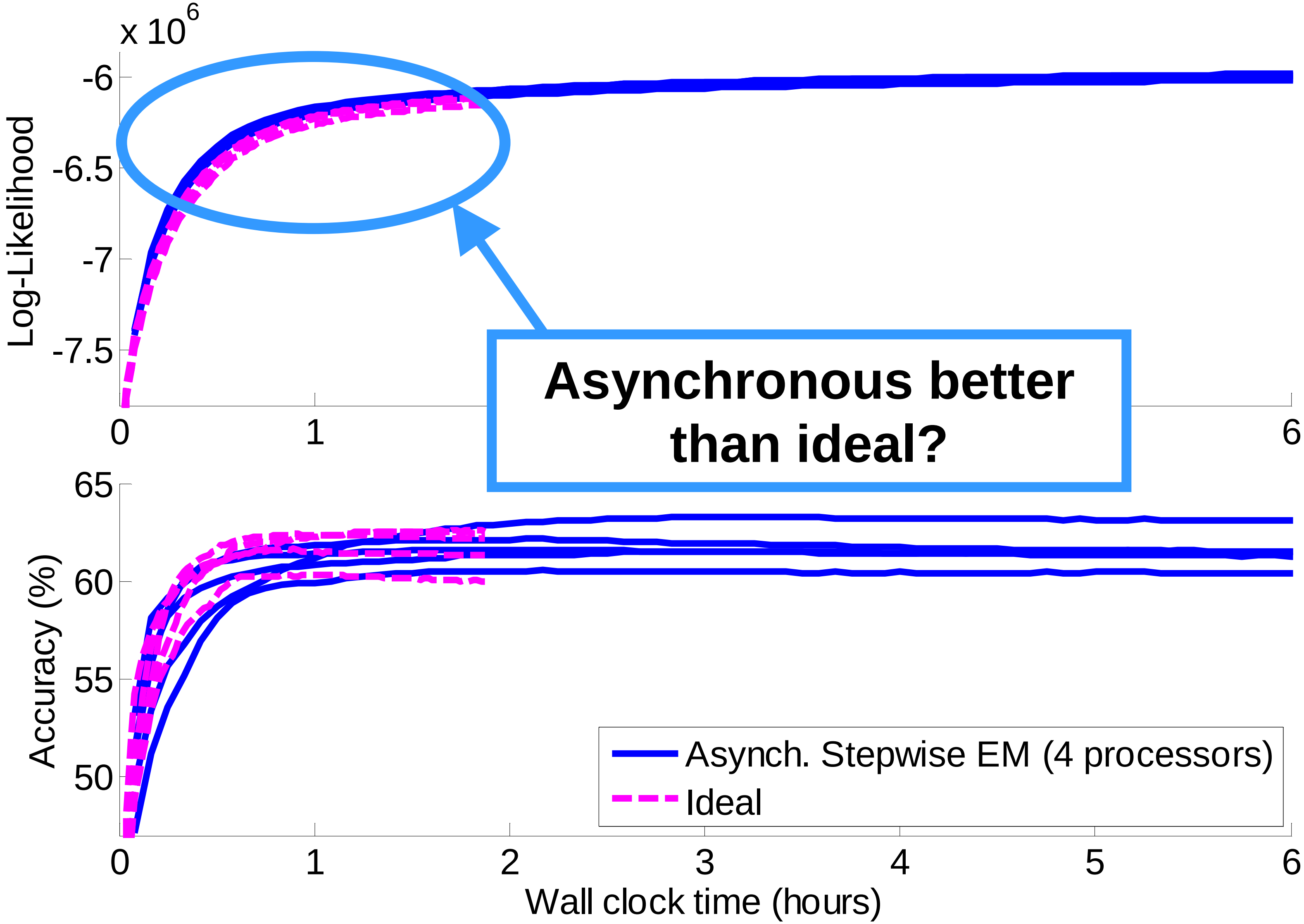


ARK lti Carnegie Mellon

Comparison with Ideal



Comparison with Ideal



Conclusions and Future Work

- Asynchronous algorithms speed convergence and do not introduce additional error
- Effective for unsupervised learning and non-convex objectives
- If your problem works well with small mini-batches, try this!

- Future work
  - Theoretical results for non-convex case
  - Explore effects of increasing number of processors
  - New architectures (maintain multiple copies of θ)

Thanks!