

Scheduling for Efficient Large-Scale Machine Learning Training

Jinliang Wei

Carnegie Mellon University



Carnegie Mellon University

Computer Science Department

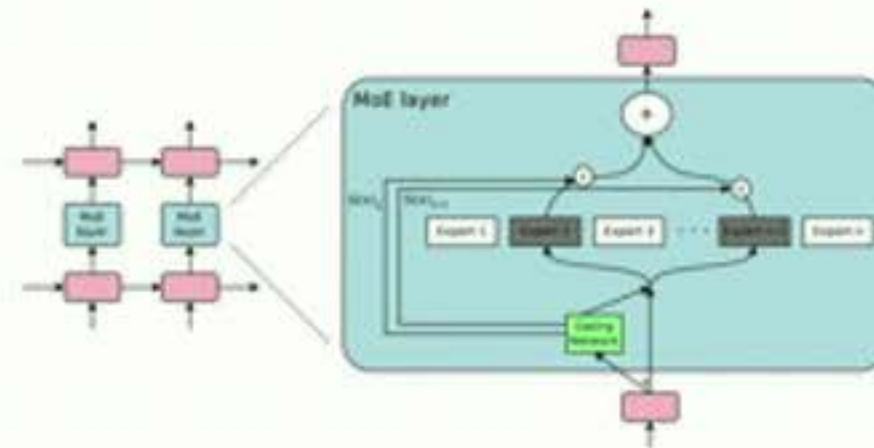
Machine Learning Training: Quest for Efficiency



YouTube-8M Dataset

6.1 Million Video IDs	350,000 Hours of Video	2.6 Billion Audio/Visual Features	3862 Classes	3.0 Avg. Labels / Video
--------------------------	---------------------------	---	-----------------	-------------------------------

Growing data size



Growing model complexity

Challenges:

Machine learning models take long time to train.

Machine learning training consumes large amount of memory.

Implementing parallel/distributed programs is hard.

My Work: More Efficient ML Training via Scheduling

Key Idea:

Leverage general structural properties in ML computation to improve efficiency

My Work: More Efficient ML Training via Scheduling

Key Idea:

Leverage general structural properties in ML computation to improve efficiency

Challenges:

What structural properties are helpful?

My Work: More Efficient ML Training via Scheduling

Key Idea:

Leverage general structural properties in ML computation to improve efficiency

My Work: More Efficient ML Training via Scheduling

Key Idea:

Leverage general structural properties in ML computation to improve efficiency

Challenges:

What structural properties are helpful?

My Work: More Efficient ML Training via Scheduling

Key Idea:

Leverage general structural properties in ML computation to improve efficiency

Challenges:

What structural properties are helpful?

Generalizability across models / algorithms

My Work: More Efficient ML Training via Scheduling

Key Idea:

Leverage general structural properties in ML computation to improve efficiency

Challenges:

What structural properties are helpful?

Generalizability across models / algorithms

How to leverage it with no / little burden to users?

My Work: More Efficient ML Training via Scheduling

Key Idea:

Leverage general structural properties in ML computation to improve efficiency



Systems developed:

Bösen: (parameter server) [SoCC'15]
~20K LoC (C++)

Orion: (auto-parallelization) [EuroSys'19]
~23K LoC (C++, Julia)

Non-trivial work on **TensorFlow core**

Challenges:

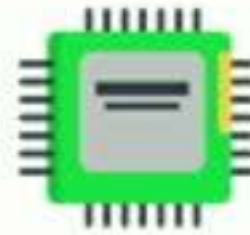
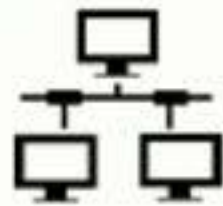
What structural properties are helpful?

Generalizability across models / algorithms

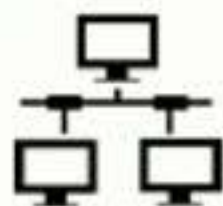
How to leverage it with no / little burden to users?

Scheduling within a Single Training Job

Scheduling within a Single Training Job



Scheduling within a Single Training Job

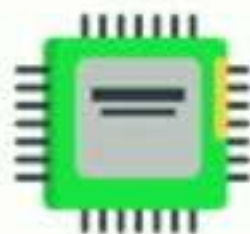


Network Communication

When and what to send?

Lead [SoCC'15, Best paper]

Coauthor [ATC'17] [SysML'19]



Computation

What to compute in parallel?

[EuroSys'19]



Memory Allocation

When and where to allocate?

[In preparation]

Highlights of results:

- Scheduling communication: up to 30% faster convergence
- Scheduling computation: even faster convergence with less programmer effort
- Scheduling memory: 10x bigger model on the same hardware

Background: Serial Machine Learning Training

Sequential learning algorithm, e.g., SGD:

```
repeat until convergence
  foreach mini-batch in dataset
    update model parameters
```

Background: Serial Machine Learning Training

Sequential learning algorithm, e.g., SGD:

```
repeat until convergence ← Many passes over training data
  foreach mini-batch in dataset
    update model parameters
```

Background: Serial Machine Learning Training

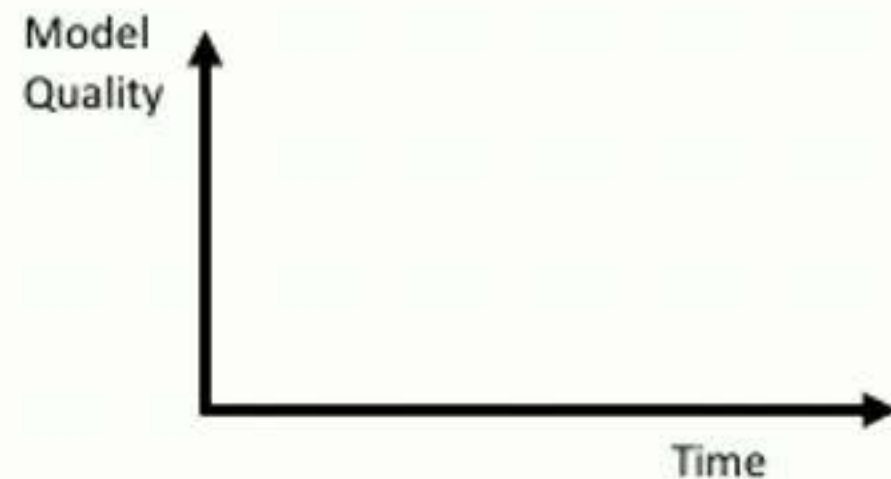
Sequential learning algorithm, e.g., SGD:

```
repeat until convergence ← Many passes over training data
  foreach mini-batch in dataset ← Many updates per data pass
    update model parameters
```

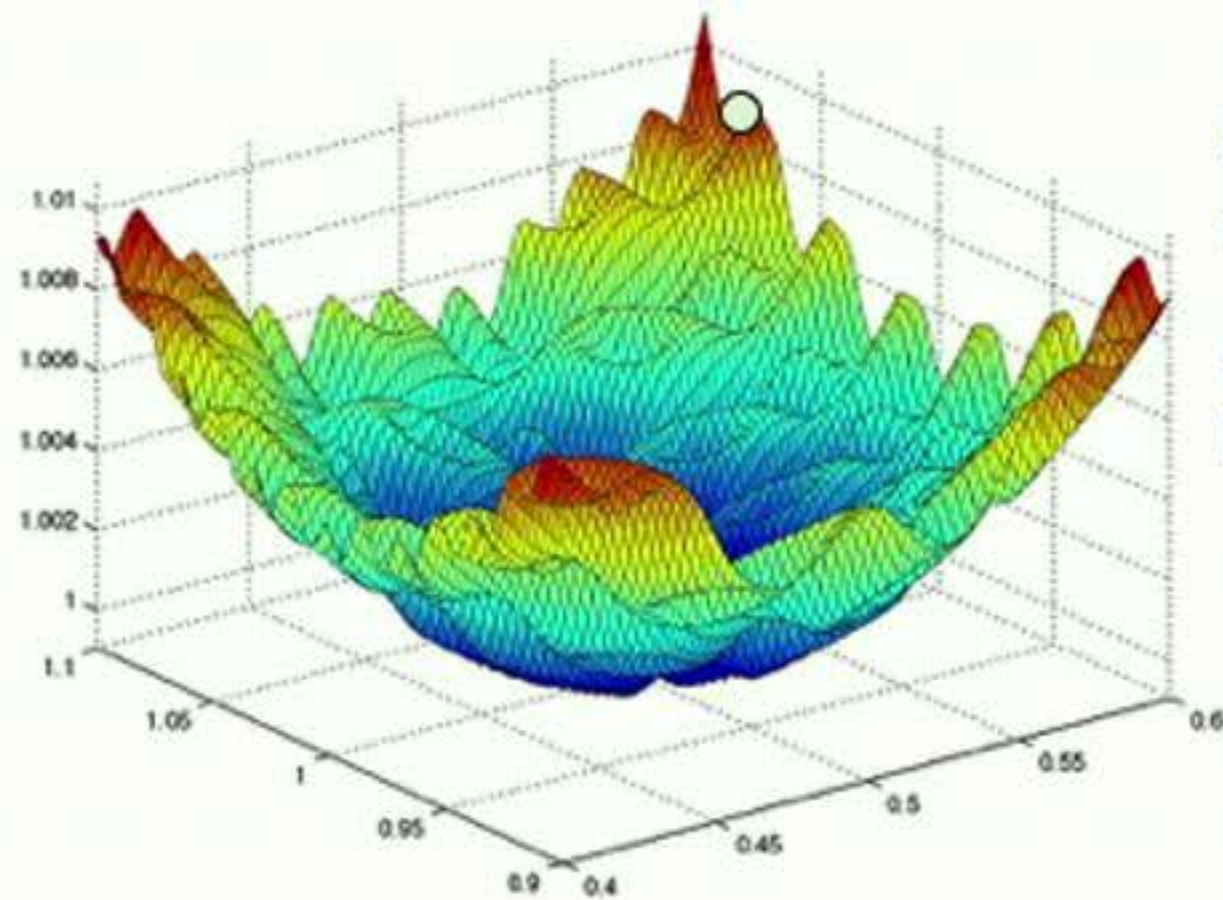
Background: Serial Machine Learning Training

Sequential learning algorithm, e.g., SGD:

```
repeat until convergence ← Many passes over training data
  foreach mini-batch in dataset ← Many updates per data pass
    update model parameters
```



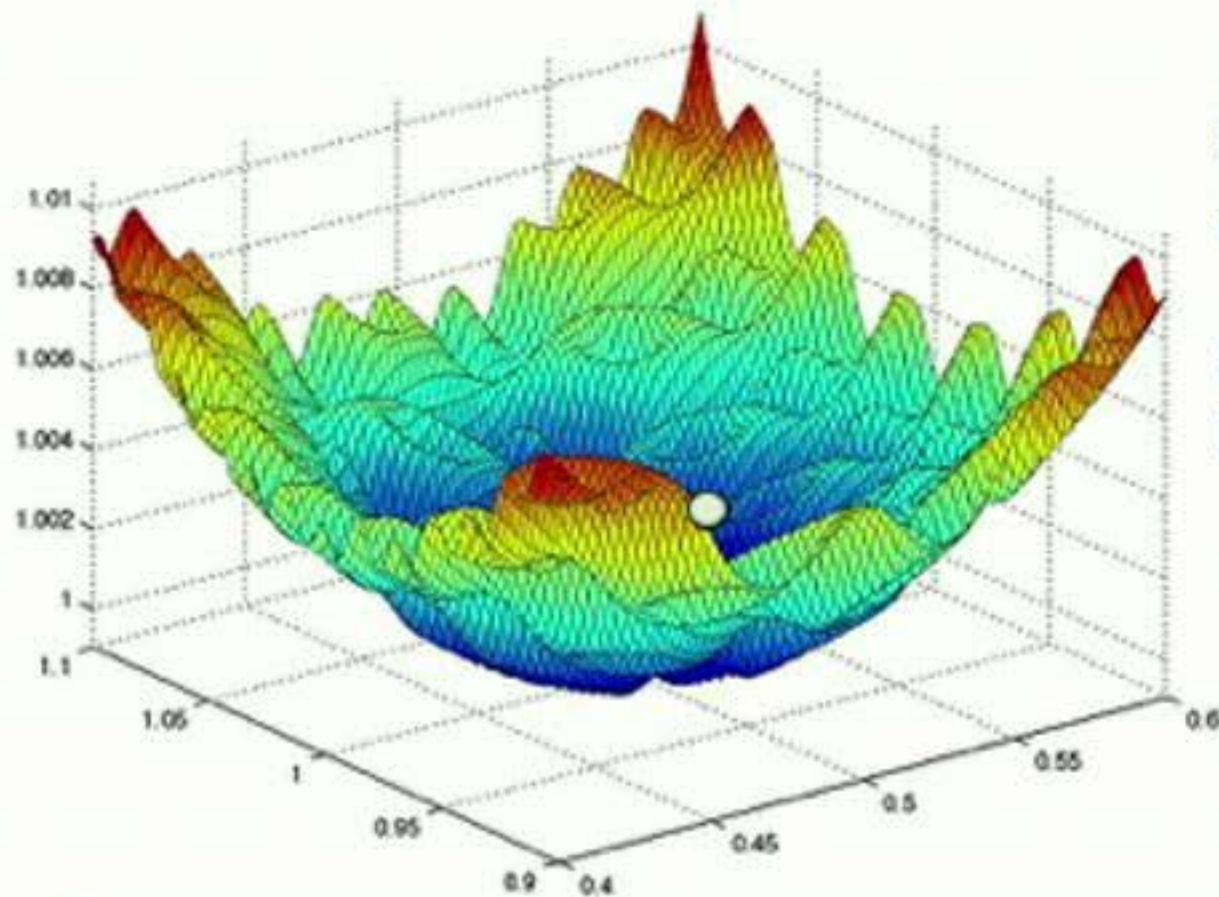
Machine Learning Training Is A Search Process



Stopping criteria (convergence):
achieve a desired model quality (plateau)

Error doesn't mean it's wrong
It often means more steps

Machine Learning Training Is A Search Process

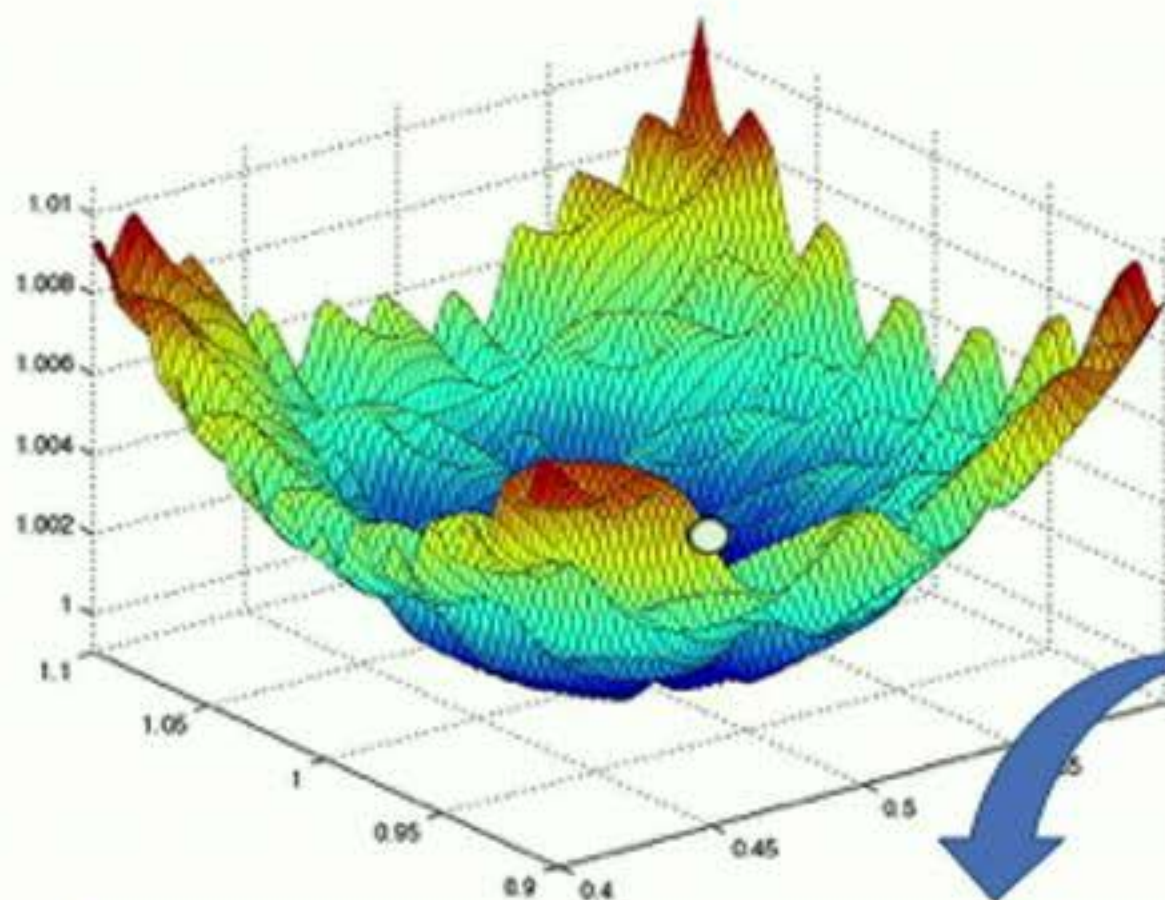


Stopping criteria (convergence):
achieve a desired model quality (plateau)

Error doesn't mean it's wrong
It often means more steps

$$\text{Convergence speed} = \text{samples/sec} * \text{convergence/sample}$$

Machine Learning Training Is A Search Process



Stopping criteria (convergence):
achieve a desired model quality (plateau)

Error doesn't mean it's wrong
It often means more steps

Trade-off is possible

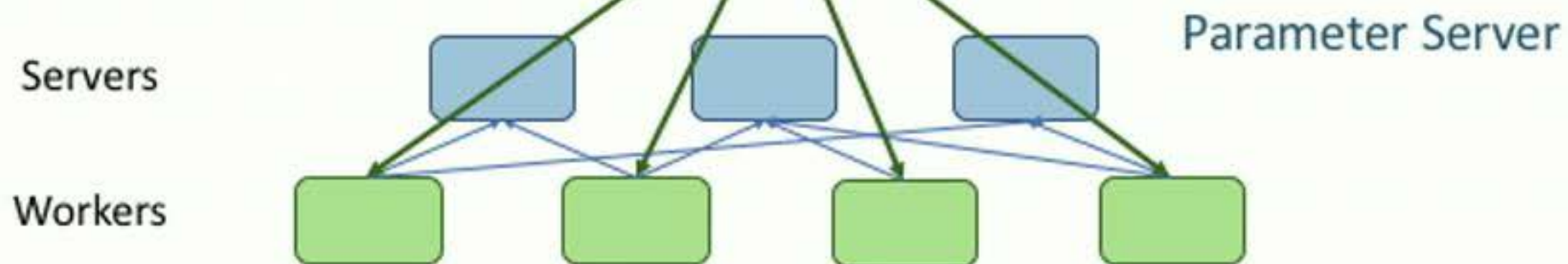
$$\text{Convergence speed} = \text{samples/sec} * \text{convergence/sample}$$

Background: Data Parallelism for Computation Throughput

Simply run some / all mini-batches in parallel, regardless of dependence

repeat until convergence

```
in parallel foreach mini-batch in dataset  
update model parameters
```



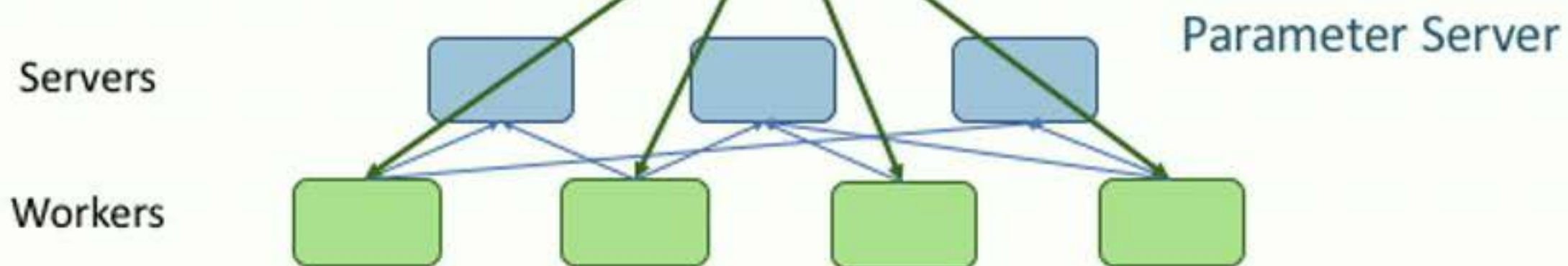
Convergence speed = samples/sec X convergence/sample

Background: Data Parallelism for Computation Throughput

Simply run some / all mini-batches in parallel, regardless of dependence

repeat until convergence

```
in parallel foreach mini-batch in dataset  
update model parameters
```



Convergence speed = samples/sec X convergence/sample

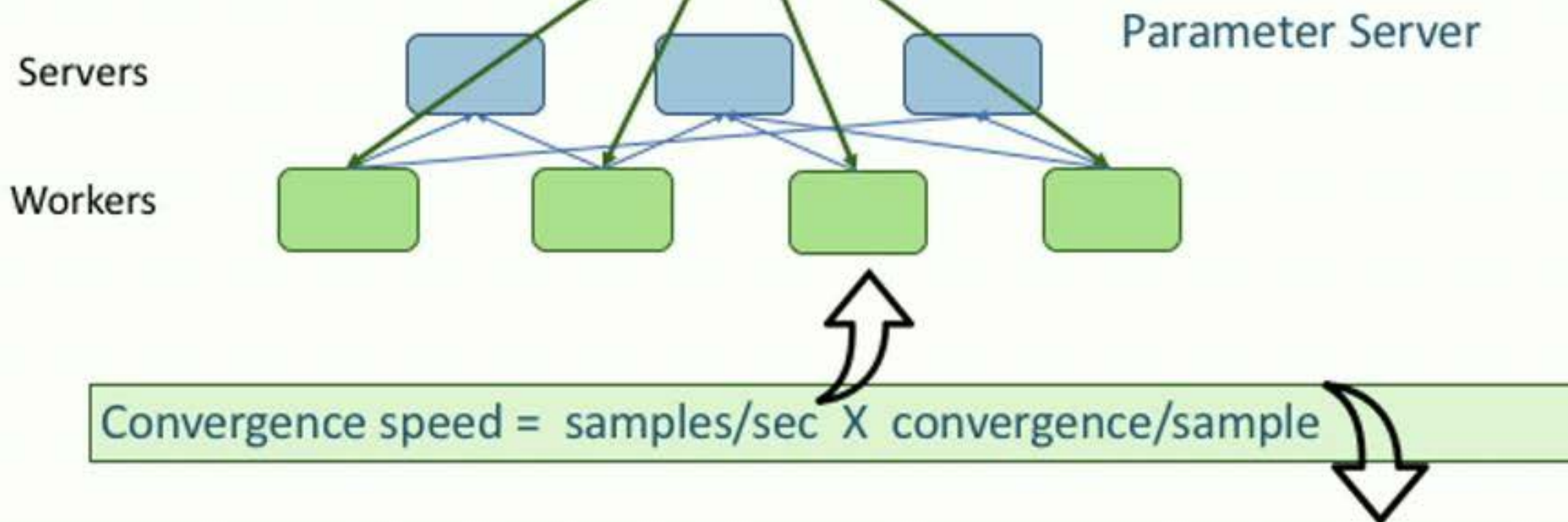
Background: Data Parallelism for Computation Throughput

Simply run some / all mini-batches in parallel, regardless of dependence

repeat until convergence

in parallel foreach mini-batch in dataset

update model parameters



Data Parallelism Does Not Retain The Sequential Semantics

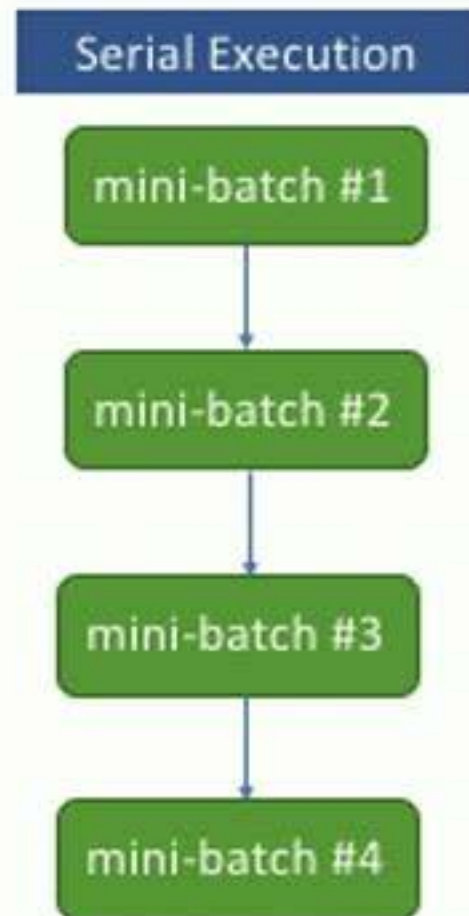
Serial Execution

mini-batch #1

Data Parallelism Does Not Retain The Sequential Semantics



Data Parallelism Does Not Retain The Sequential Semantics



Later iterations observe updates
from earlier iterations

Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Data Parallelism

mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

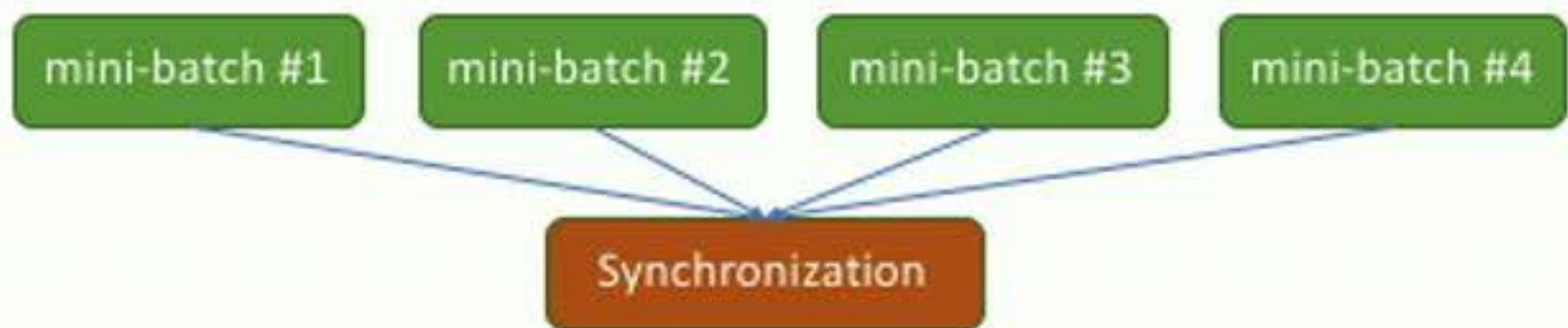
Later iterations observe updates
from earlier iterations

Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution



Data Parallelism



Later iterations observe updates from earlier iterations

Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Later iterations observe updates from earlier iterations

Data Parallelism

mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Synchronization

Inconsistency: parallel iterations do not observe updates from each other

Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Later iterations observe updates from earlier iterations

Data Parallelism

mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Synchronization

W_0 ΔW_1 $W_1 = W_0 + \Delta W_1$

Inconsistency: parallel iterations do not observe updates from each other

Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Later iterations observe updates from earlier iterations

Data Parallelism

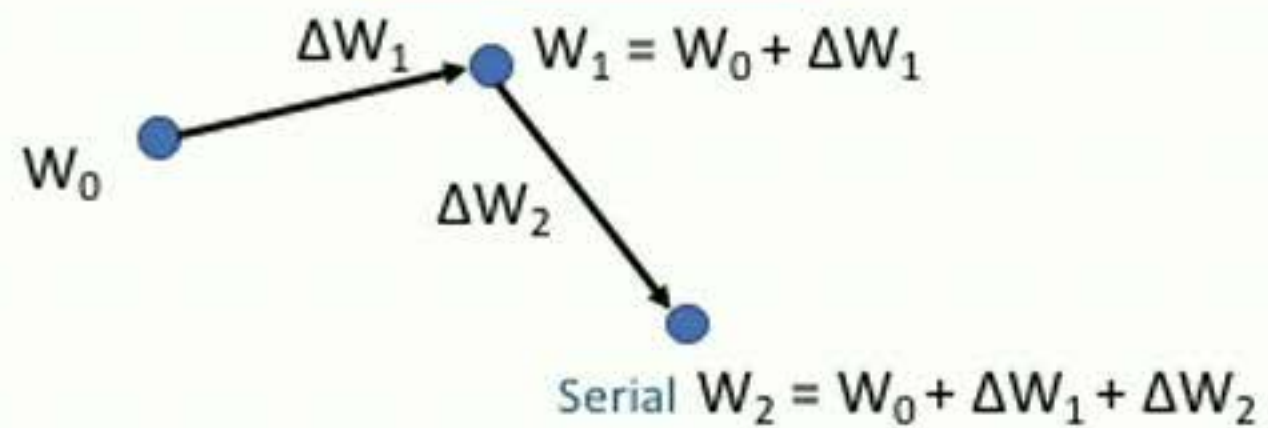
mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Synchronization



Inconsistency: parallel iterations do not observe updates from each other

Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Later iterations observe updates from earlier iterations

Data Parallelism

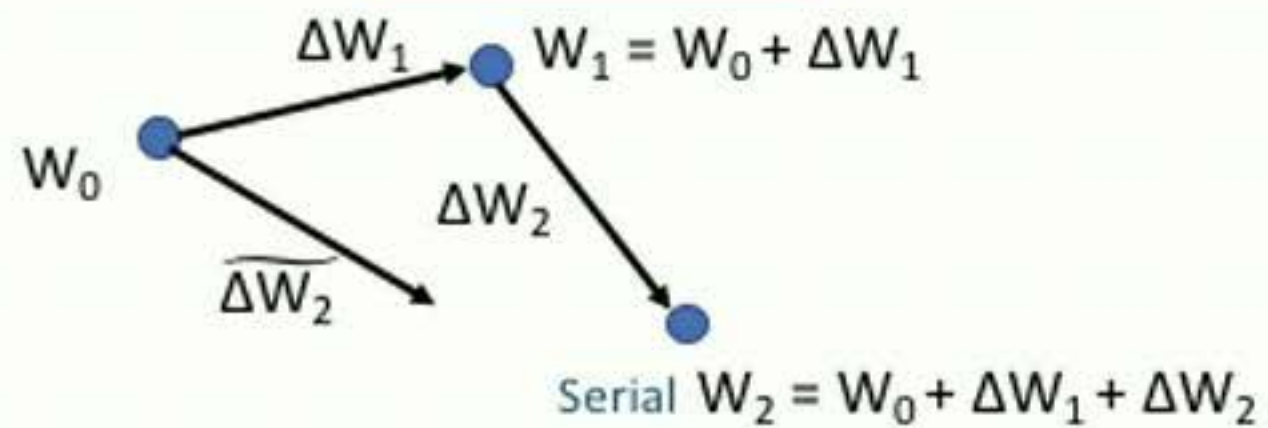
mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Synchronization



Inconsistency: parallel iterations do not observe updates from each other

Data Parallelism Does Not Retain The Sequential Semantics

Serial Execution

mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Later iterations observe updates from earlier iterations

Data Parallelism

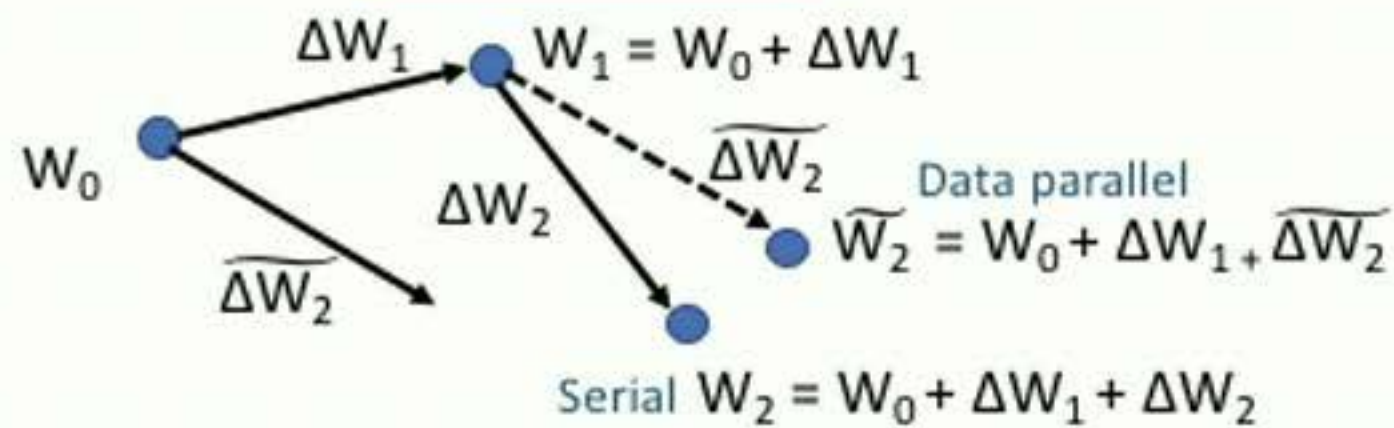
mini-batch #1

mini-batch #2

mini-batch #3

mini-batch #4

Synchronization



Inconsistency: parallel iterations do not observe updates from each other

Background: Sparsity and The Communication Bottleneck

ML models of interest (5~10 years ago):
Simple and highly sparse

- Sparse Logistic Regression
- Latent Dirichlet Allocation (LDA)
- Matrix Factorization (MF) ...

Problem:

Light computation per mini-batch vs. heavy communication

Background: Sparsity and The Communication Bottleneck

ML models of interest (5~10 years ago):
Simple and highly sparse

- Sparse Logistic Regression
- Latent Dirichlet Allocation (LDA)
- Matrix Factorization (MF) ...

Problem:

Light computation per mini-batch vs. heavy communication

Compute

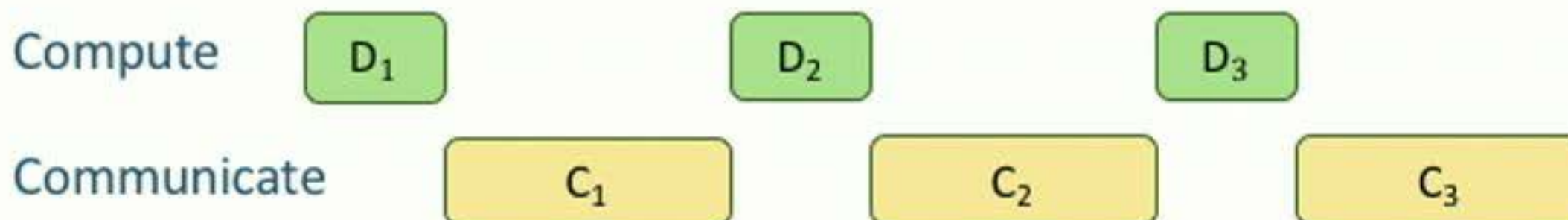
Background: Sparsity and The Communication Bottleneck

ML models of interest (5~10 years ago):
Simple and highly sparse

Sparse Logistic Regression
Latent Dirichlet Allocation (LDA)
Matrix Factorization (MF) ...

Problem:

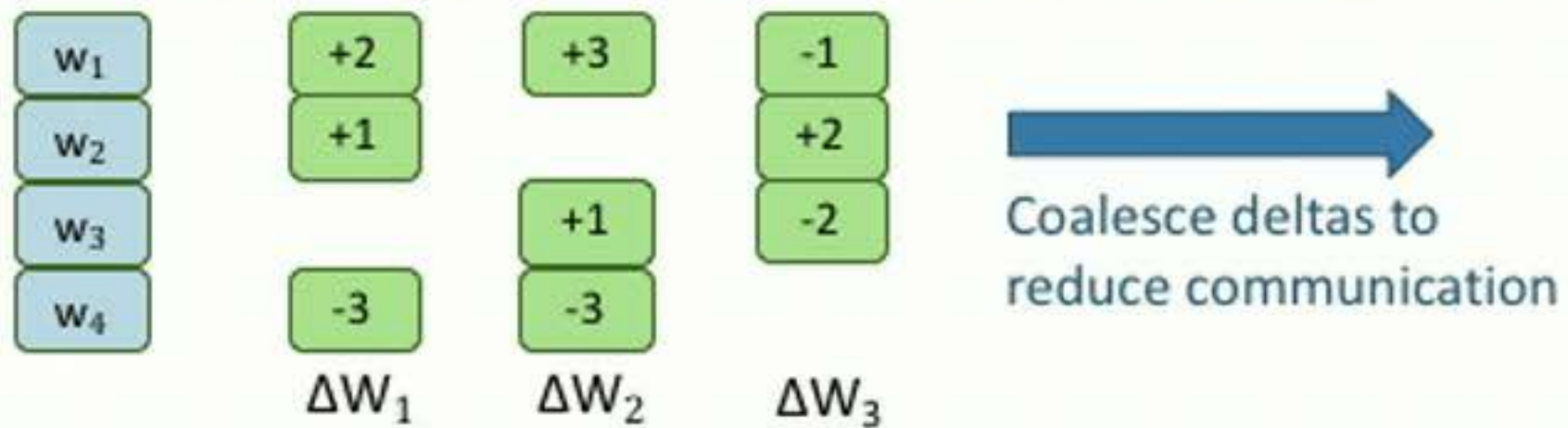
Light computation per mini-batch vs. heavy communication



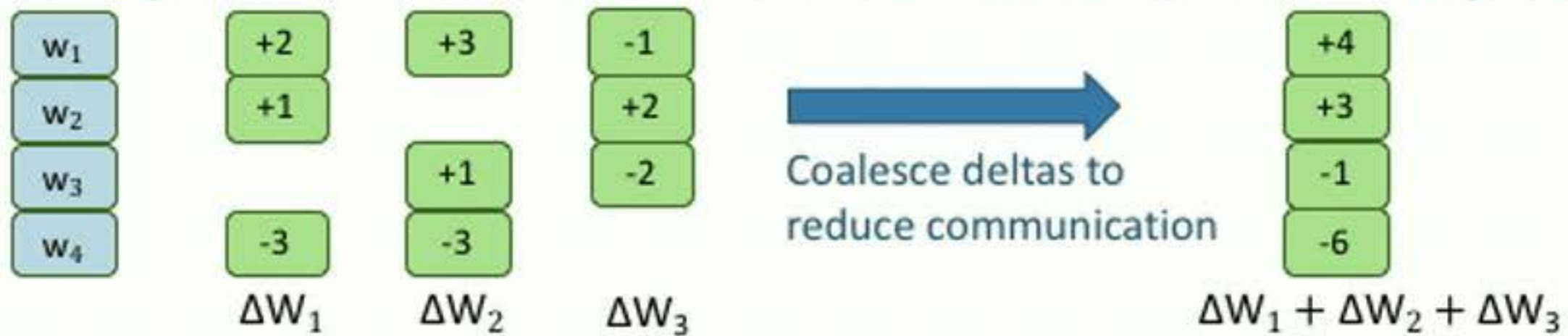
Background: Trade Even More Consistency for Throughput

w_1	+2	+3	-1
w_2	+1		+2
w_3		+1	-2
w_4	-3	-3	
	ΔW_1	ΔW_2	ΔW_3

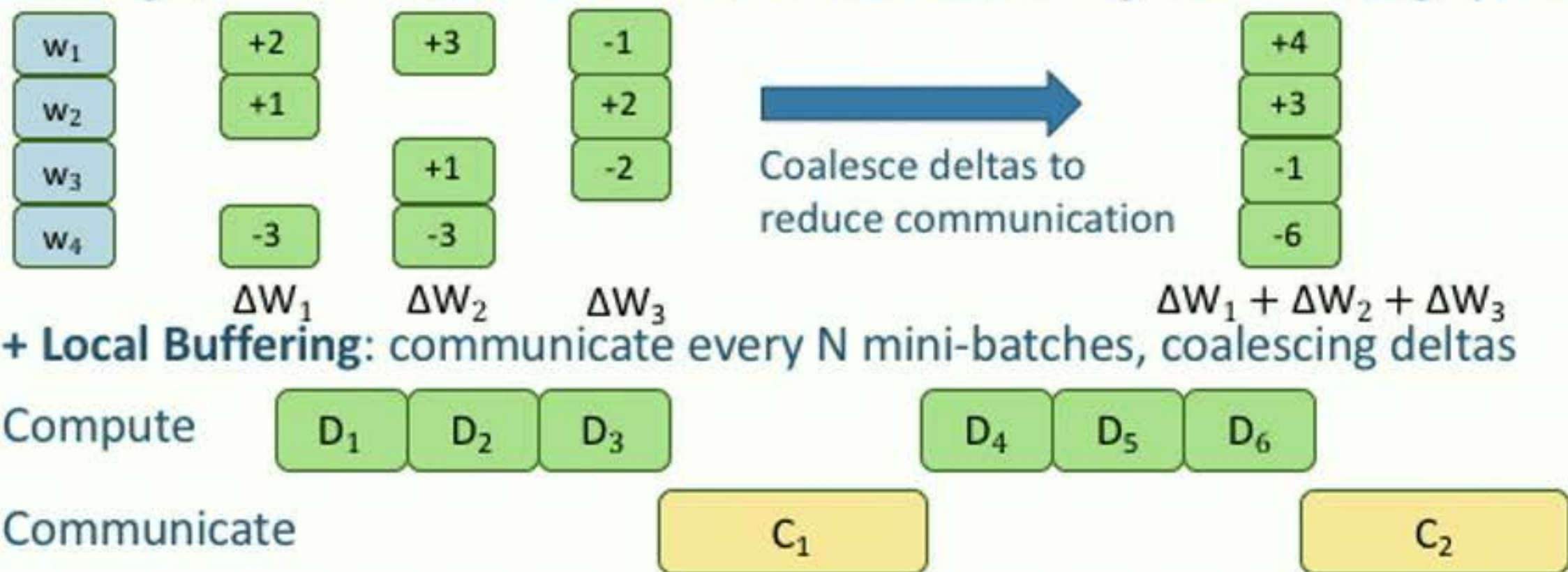
Background: Trade Even More Consistency for Throughput



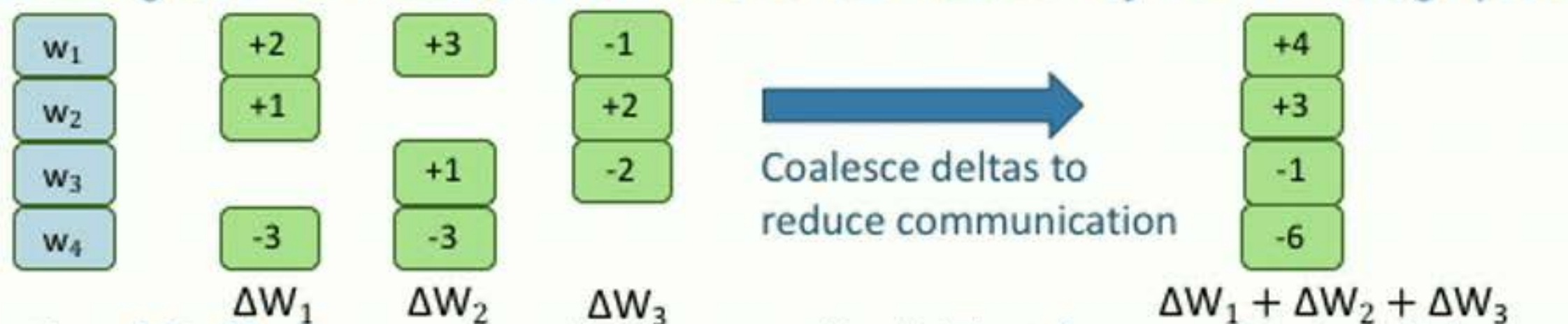
Background: Trade Even More Consistency for Throughput



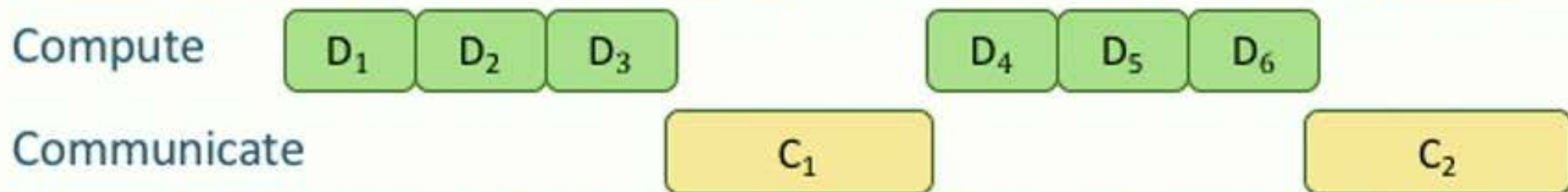
Background: Trade Even More Consistency for Throughput



Background: Trade Even More Consistency for Throughput

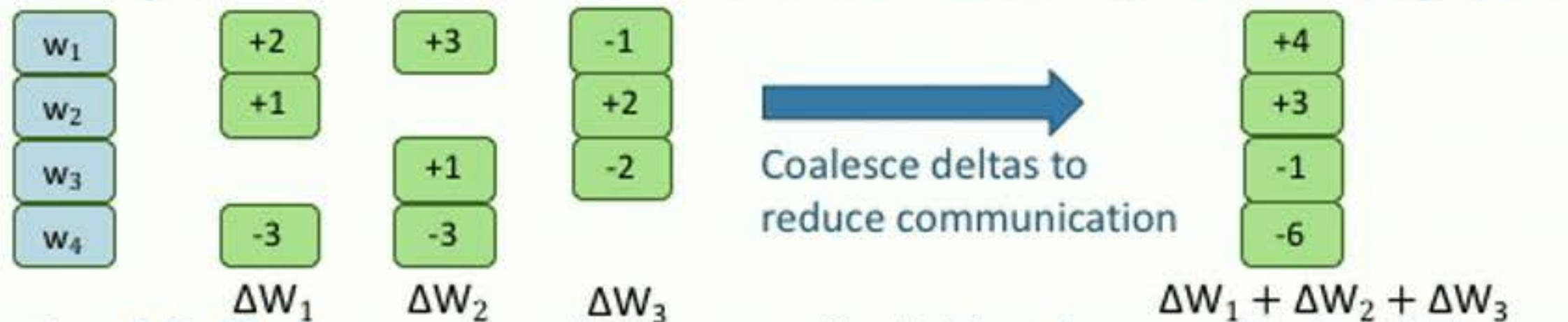


+ **Local Buffering:** communicate every N mini-batches, coalescing deltas

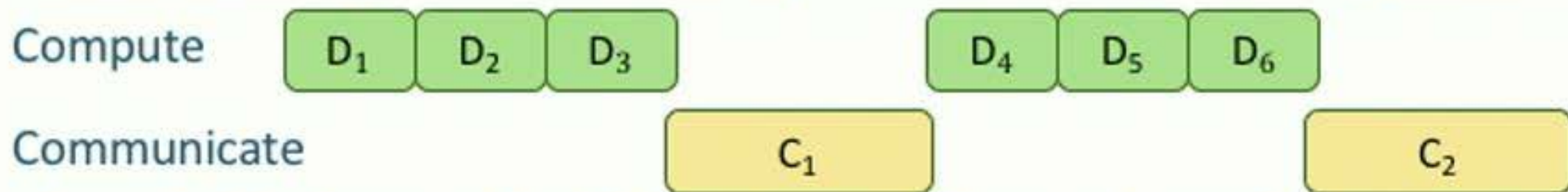


+ **Bounded Staleness:** block iff the fastest is T steps ahead of the slowest

Background: Trade Even More Consistency for Throughput



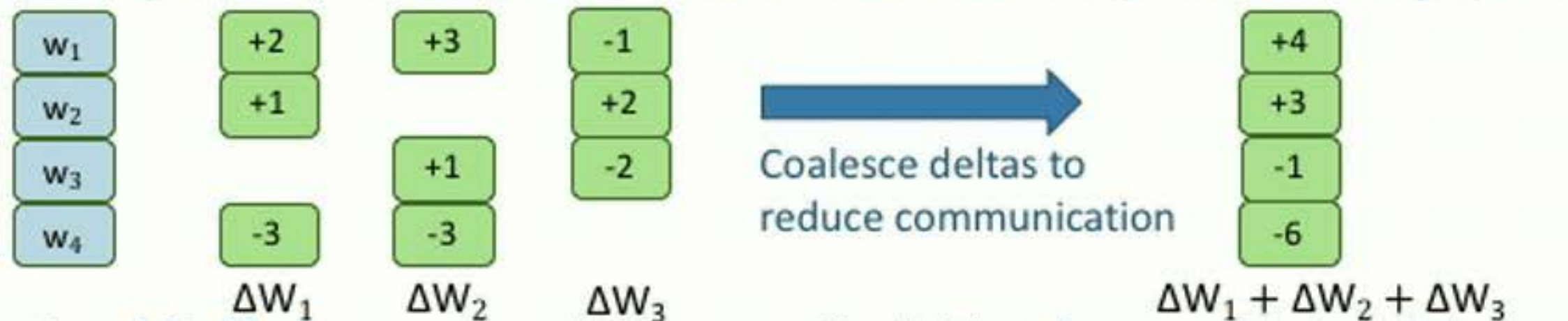
+ **Local Buffering:** communicate every N mini-batches, coalescing deltas



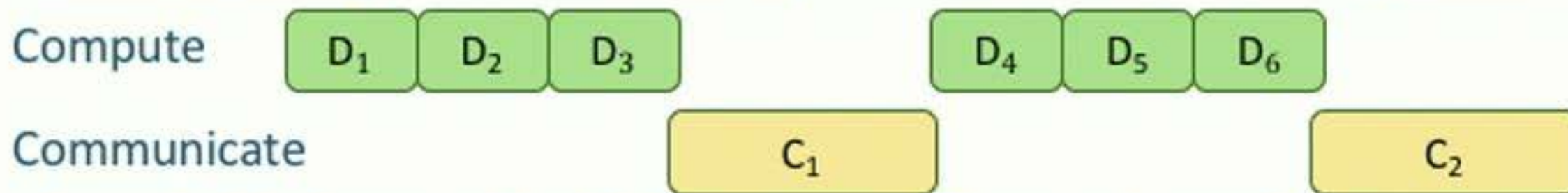
+ **Bounded Staleness:** block iff the fastest is T steps ahead of the slowest

Compute

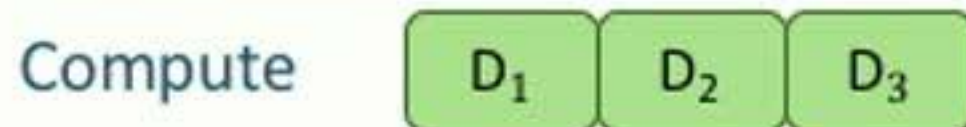
Background: Trade Even More Consistency for Throughput



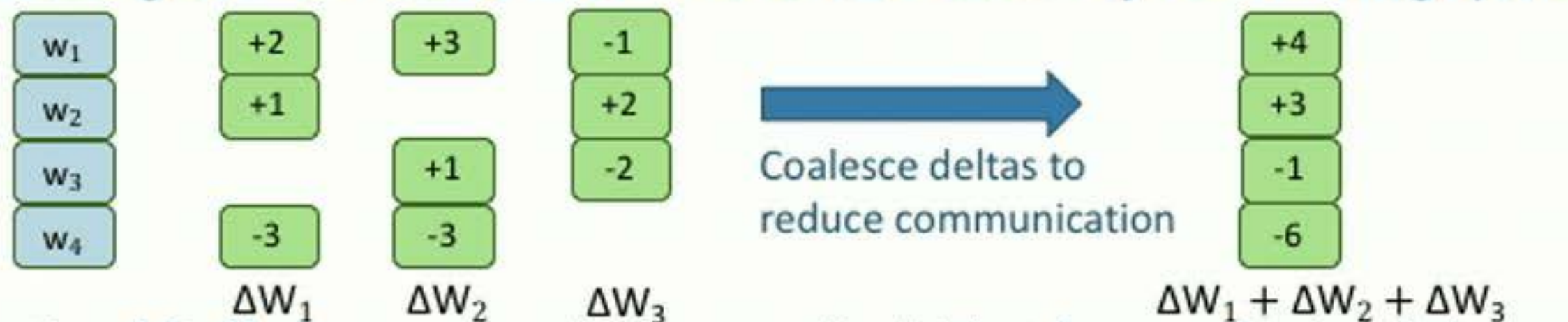
+ **Local Buffering:** communicate every N mini-batches, coalescing deltas



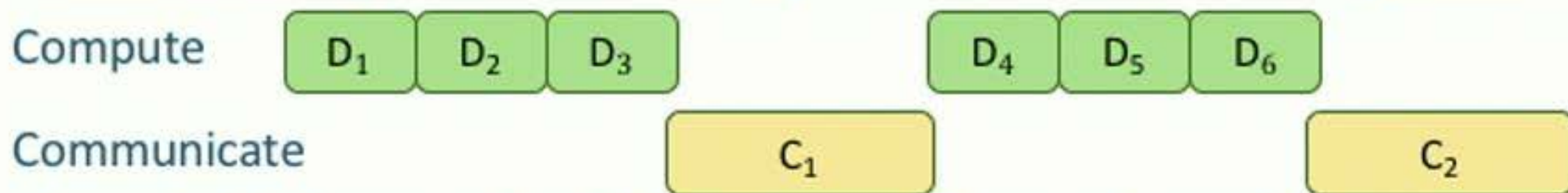
+ **Bounded Staleness:** block iff the fastest is T steps ahead of the slowest



Background: Trade Even More Consistency for Throughput



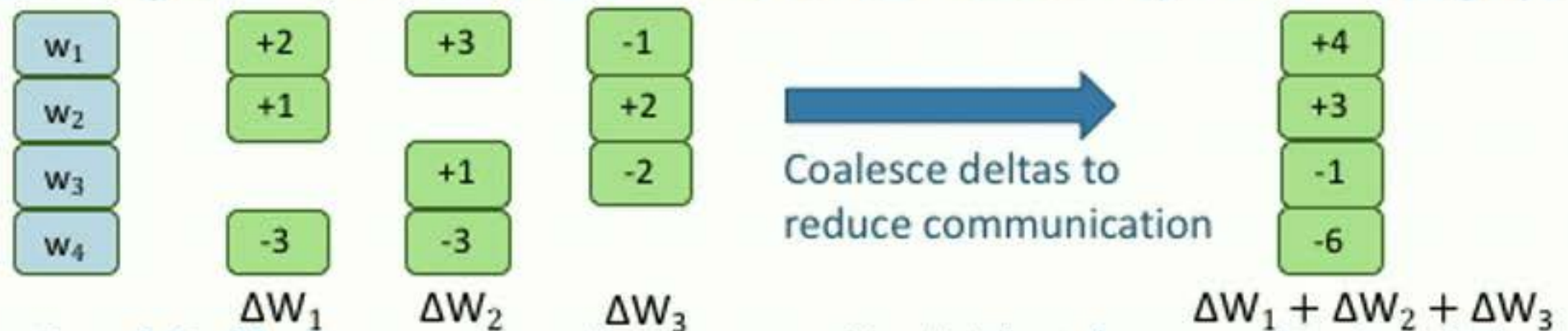
+ **Local Buffering**: communicate every N mini-batches, coalescing deltas



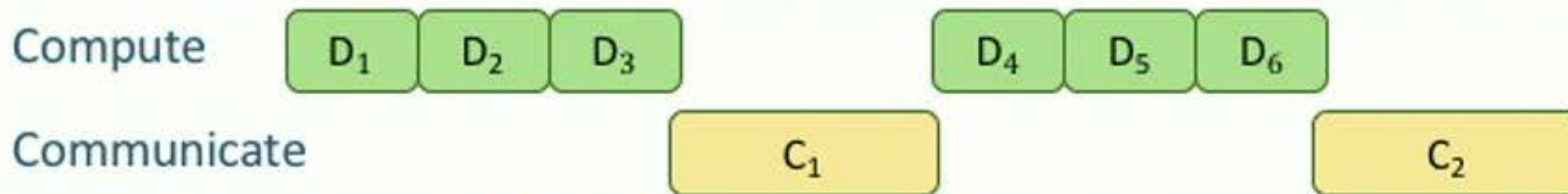
+ **Bounded Staleness**: block iff the fastest is T steps ahead of the slowest



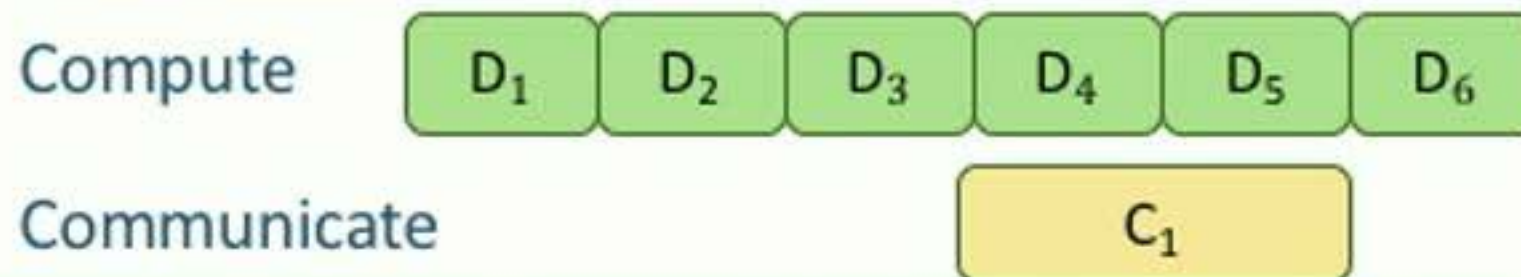
Background: Trade Even More Consistency for Throughput



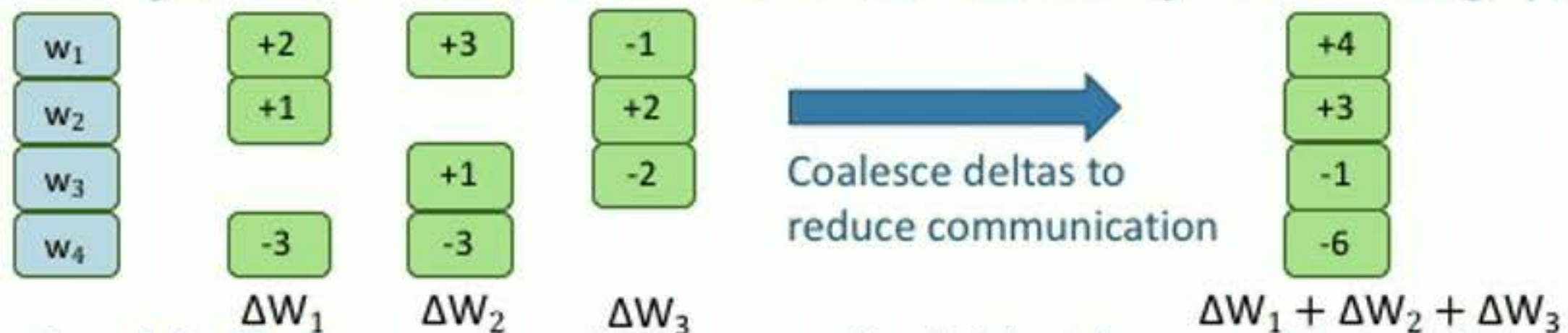
+ **Local Buffering:** communicate every N mini-batches, coalescing deltas



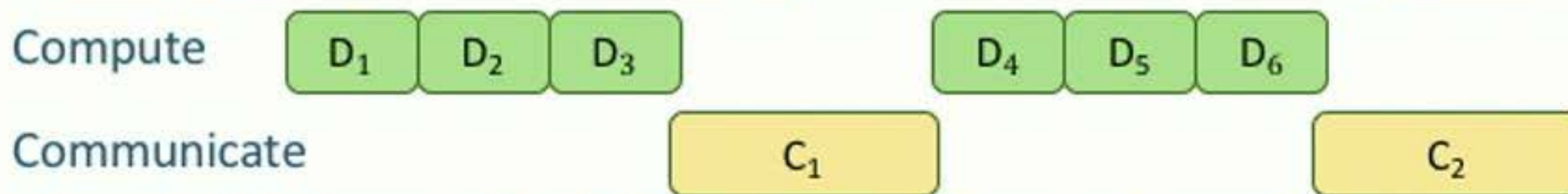
+ **Bounded Staleness:** block iff the fastest is T steps ahead of the slowest



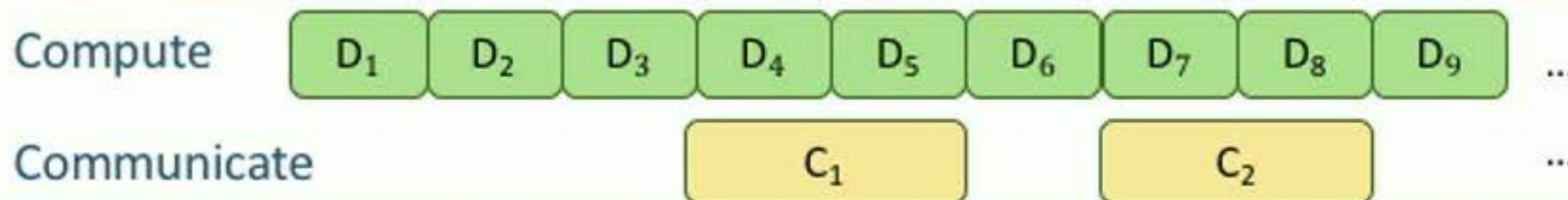
Background: Trade Even More Consistency for Throughput



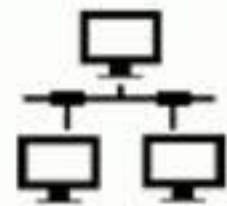
+ **Local Buffering**: communicate every N mini-batches, coalescing deltas



+ **Bounded Staleness**: block iff the fastest is T steps ahead of the slowest



Reduce Inconsistency via Scheduling

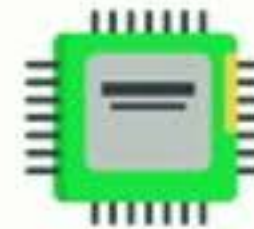


Network Communication

When and what to send?

Publication [SoCC'15, Best paper]

Systems Developed Bösen: parameter server



Computation

What to compute in parallel?

[EuroSys'19]

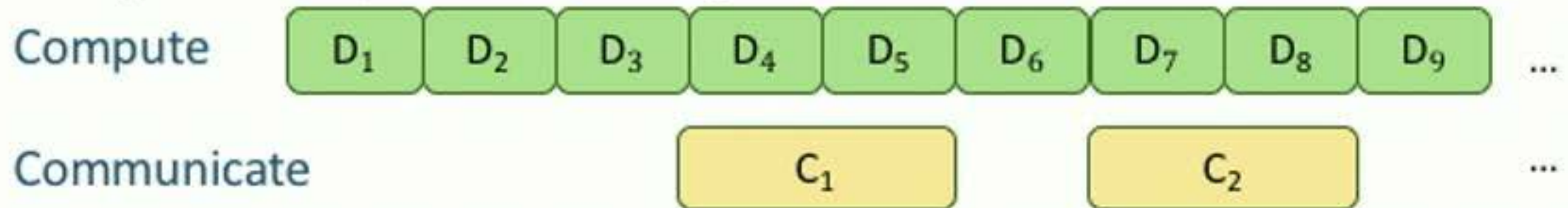
Systems Developed Orion: parallelization framework

Highlights of results:

- Scheduling communication: up to 30% faster convergence
- Scheduling computation: even faster convergence with less programmer effort

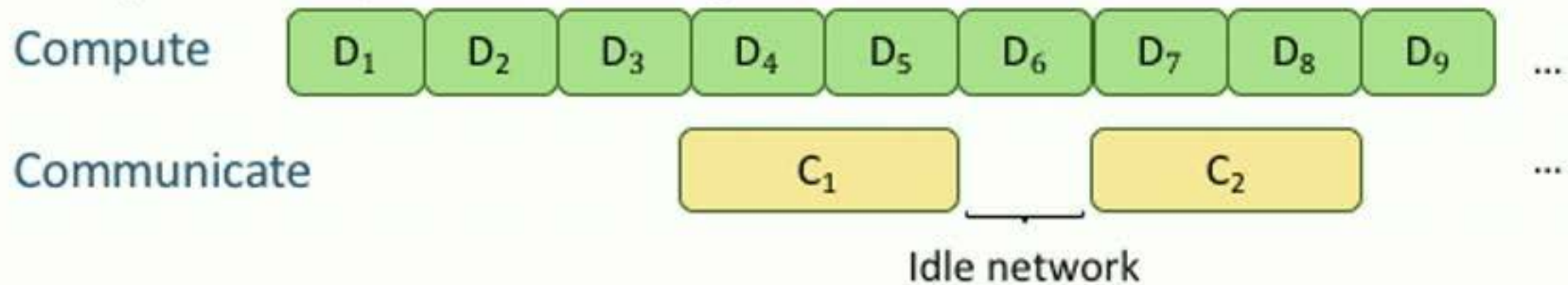
Opportunity: Spare Network Bandwidth

Data parallelism, + local buffering + bounded staleness:



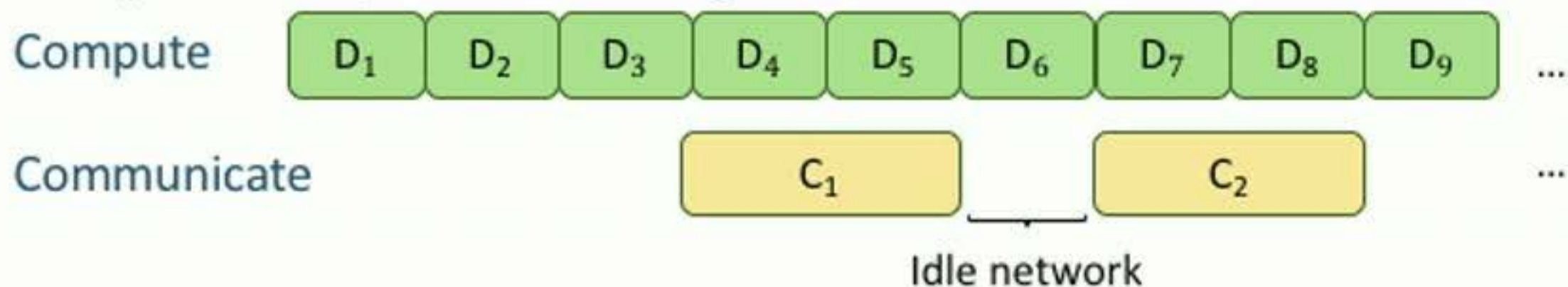
Opportunity: Spare Network Bandwidth

Data parallelism, + local buffering + bounded staleness:

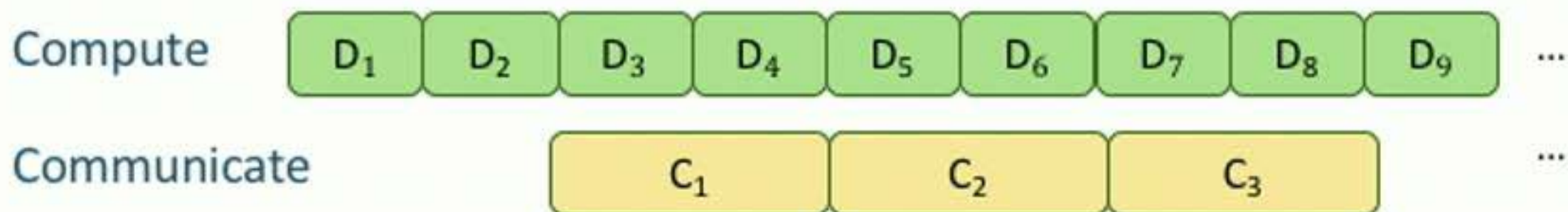


Opportunity: Spare Network Bandwidth

Data parallelism, + local buffering + bounded staleness:

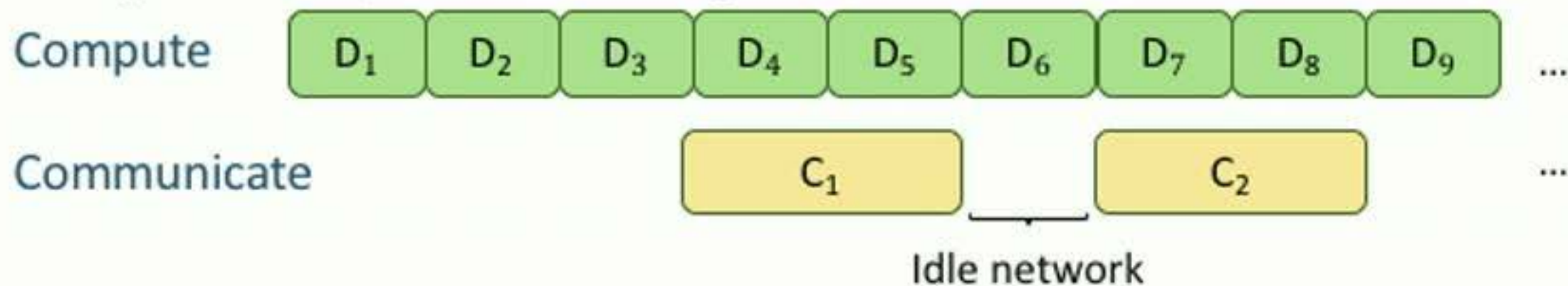


Manually tuning communication frequency:

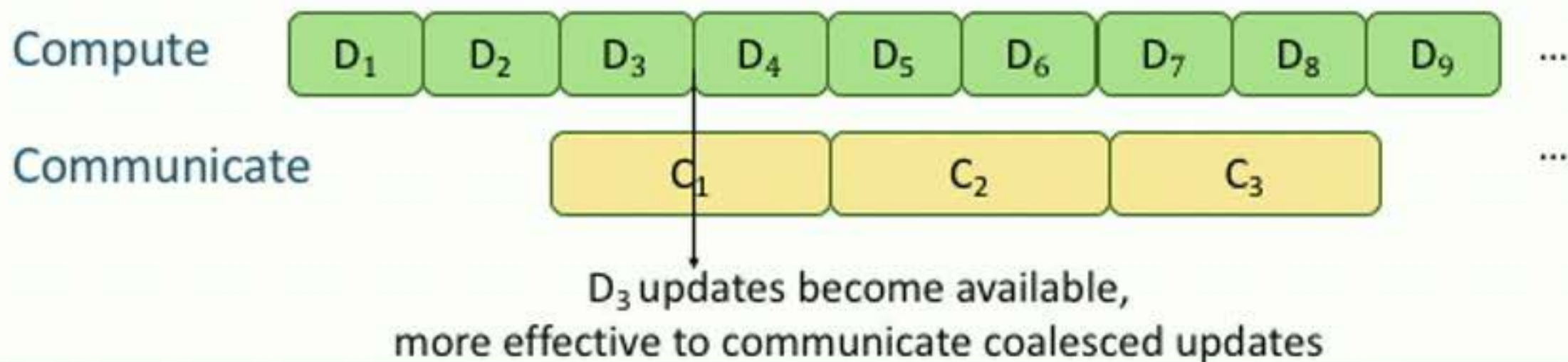


Opportunity: Spare Network Bandwidth

Data parallelism, + local buffering + bounded staleness:

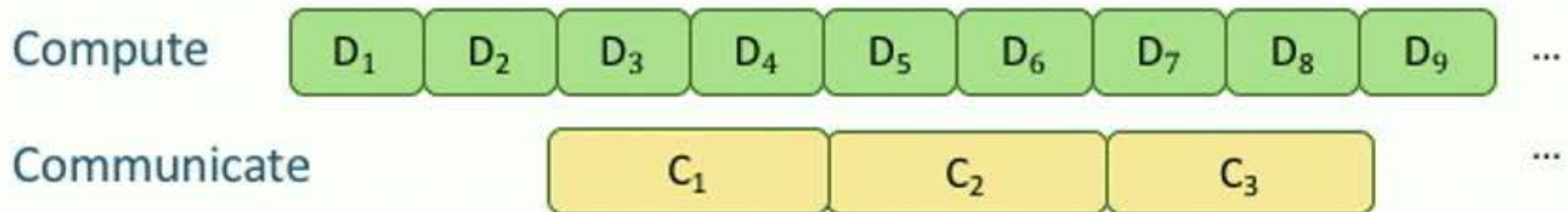


Manually tuning communication frequency:



Fine-Grained Comm. + Prioritization [Wei et al., SoCC'15]

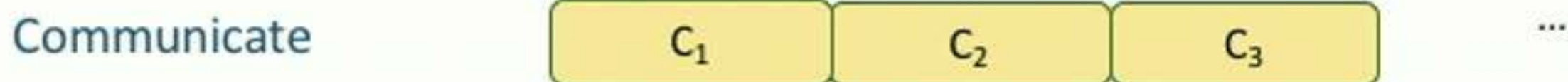
Existing: manually tuned communication frequency



...

Fine-Grained Comm. + Prioritization [Wei et al., SoCC'15]

Existing: manually tuned communication frequency

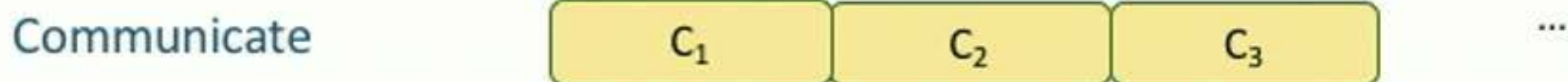


Ours: fine-grained communication

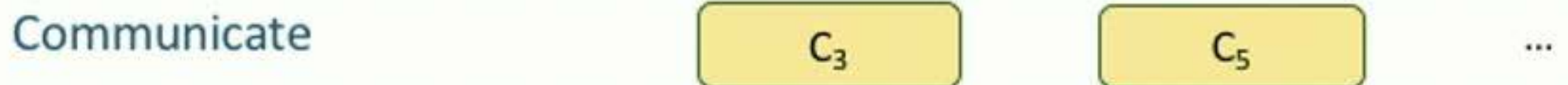


Fine-Grained Comm. + Prioritization [Wei et al., SoCC'15]

Existing: manually tuned communication frequency

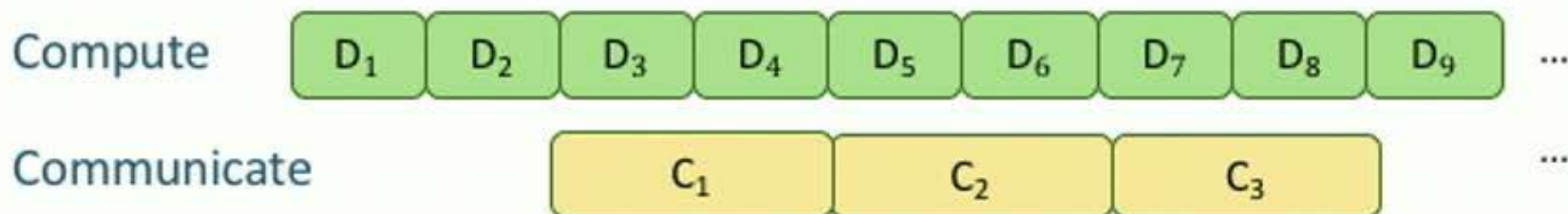


Ours: fine-grained communication

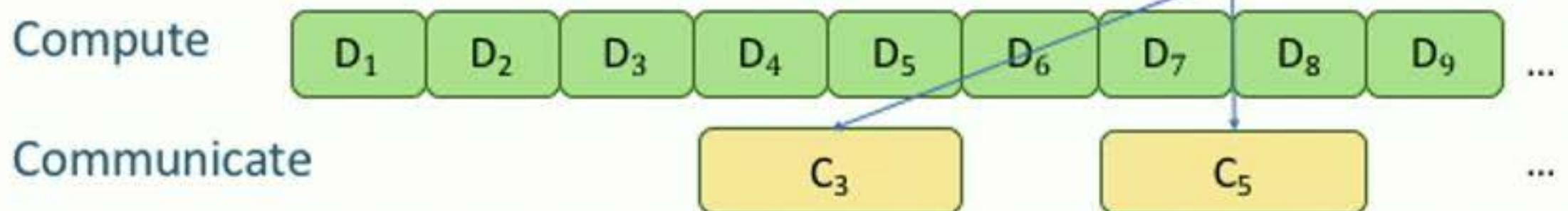


Fine-Grained Comm. + Prioritization [Wei et al., SoCC'15]

Existing: manually tuned communication frequency

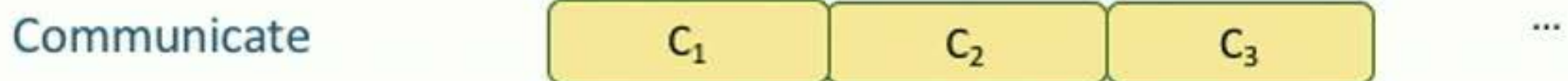


Ours: fine-grained communication Periodic synchronization to ensure convergence

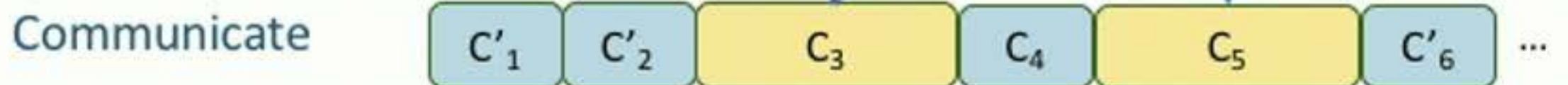


Fine-Grained Comm. + Prioritization [Wei et al., SoCC'15]

Existing: manually tuned communication frequency

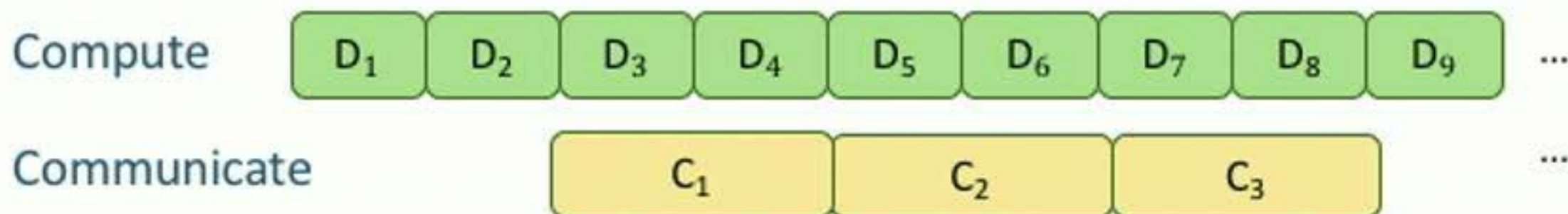


Ours: fine-grained communication Periodic synchronization to ensure convergence

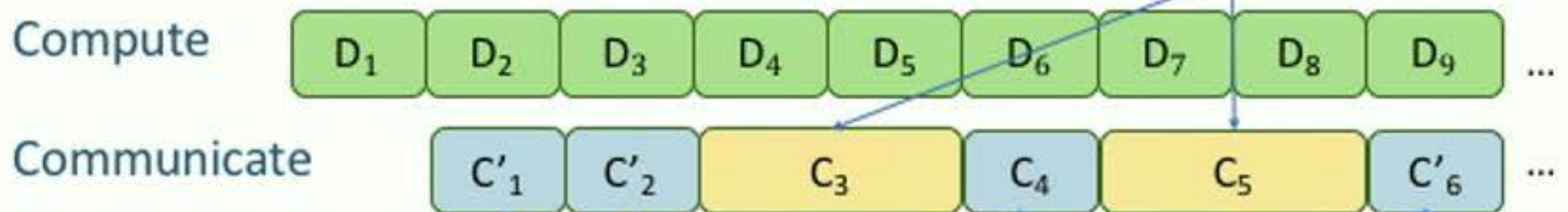


Fine-Grained Comm. + Prioritization [Wei et al., SoCC'15]

Existing: manually tuned communication frequency



Ours: fine-grained communication Periodic synchronization to ensure convergence



Prioritize update communication based on relative magnitude

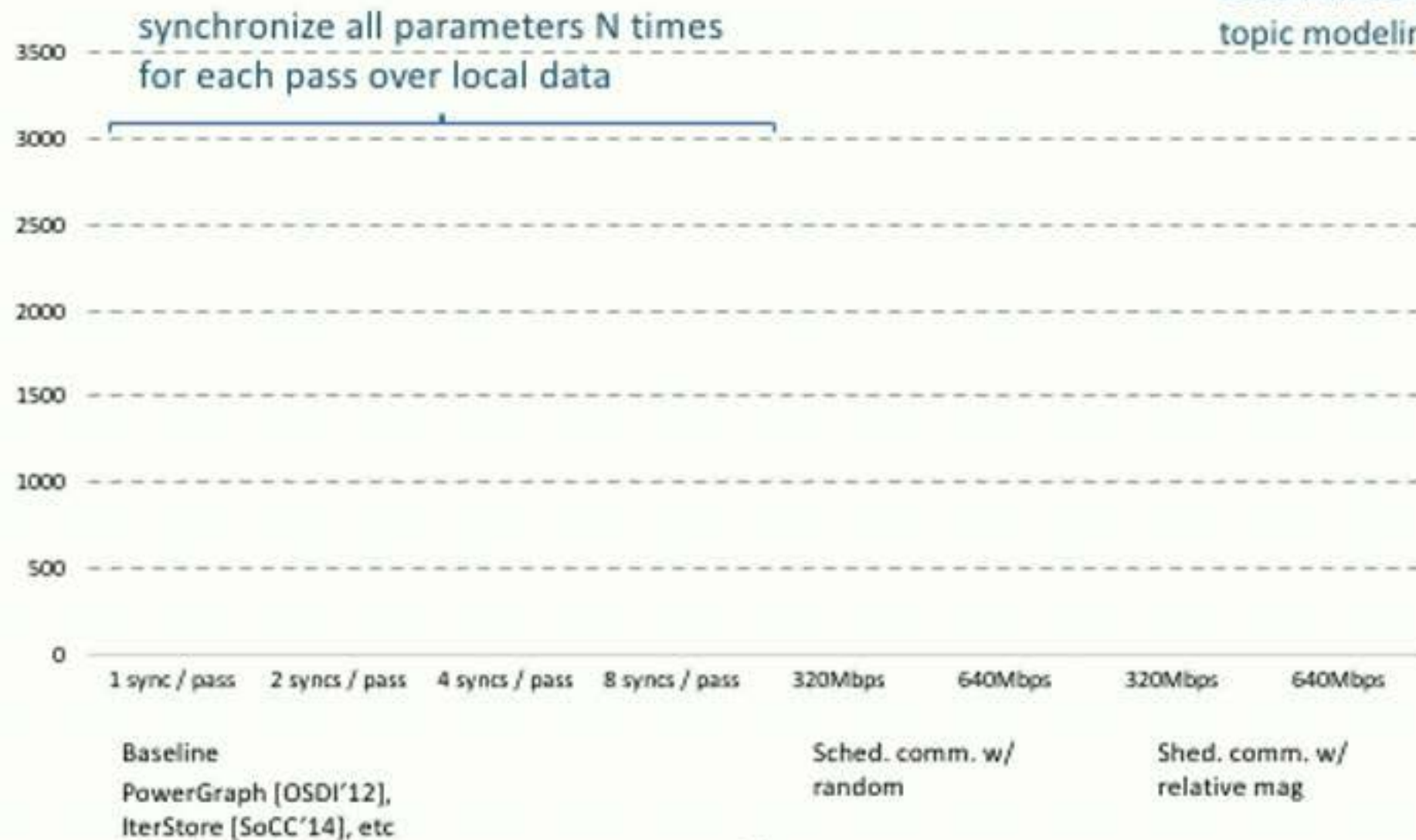
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps,
topic modeling



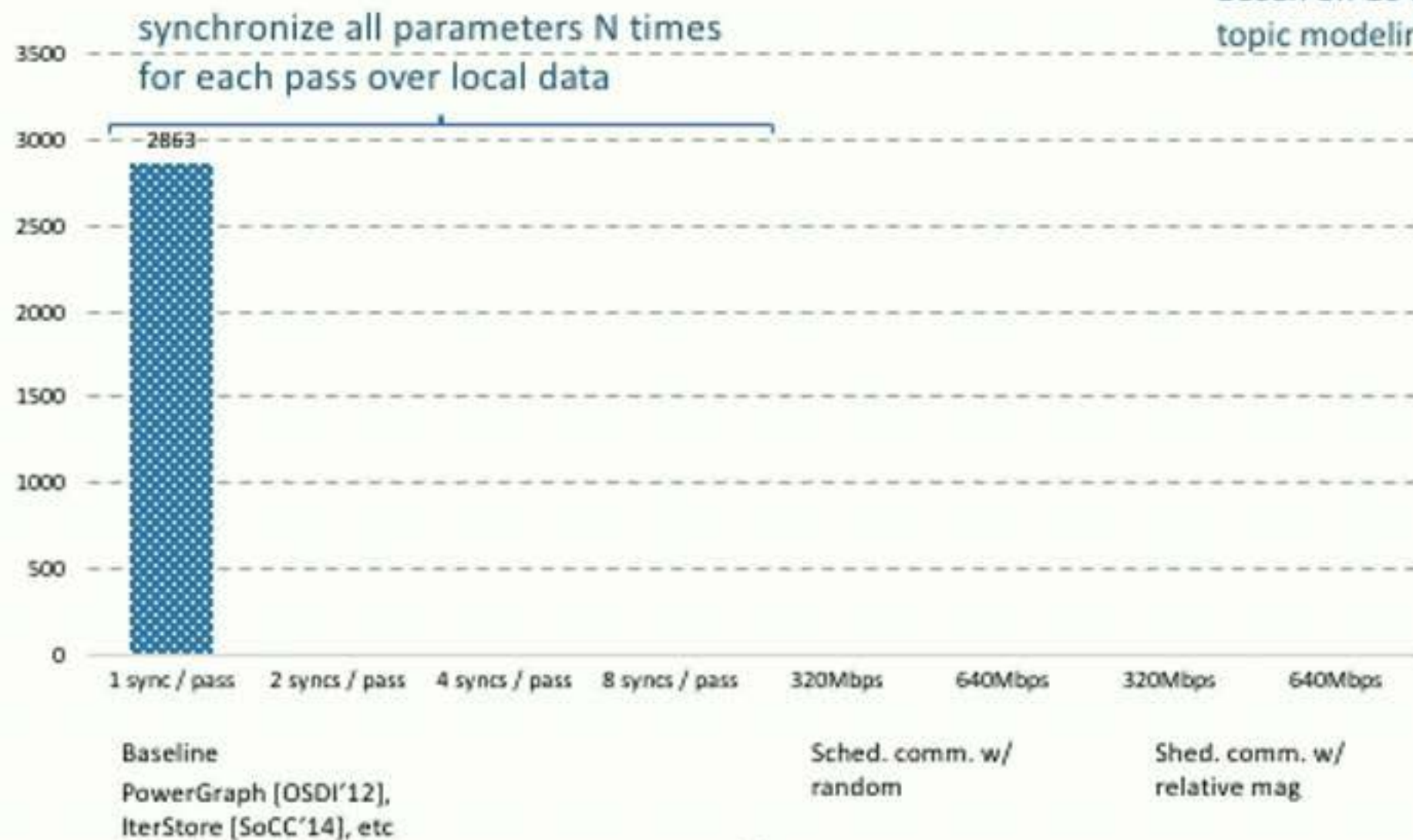
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps,
topic modeling



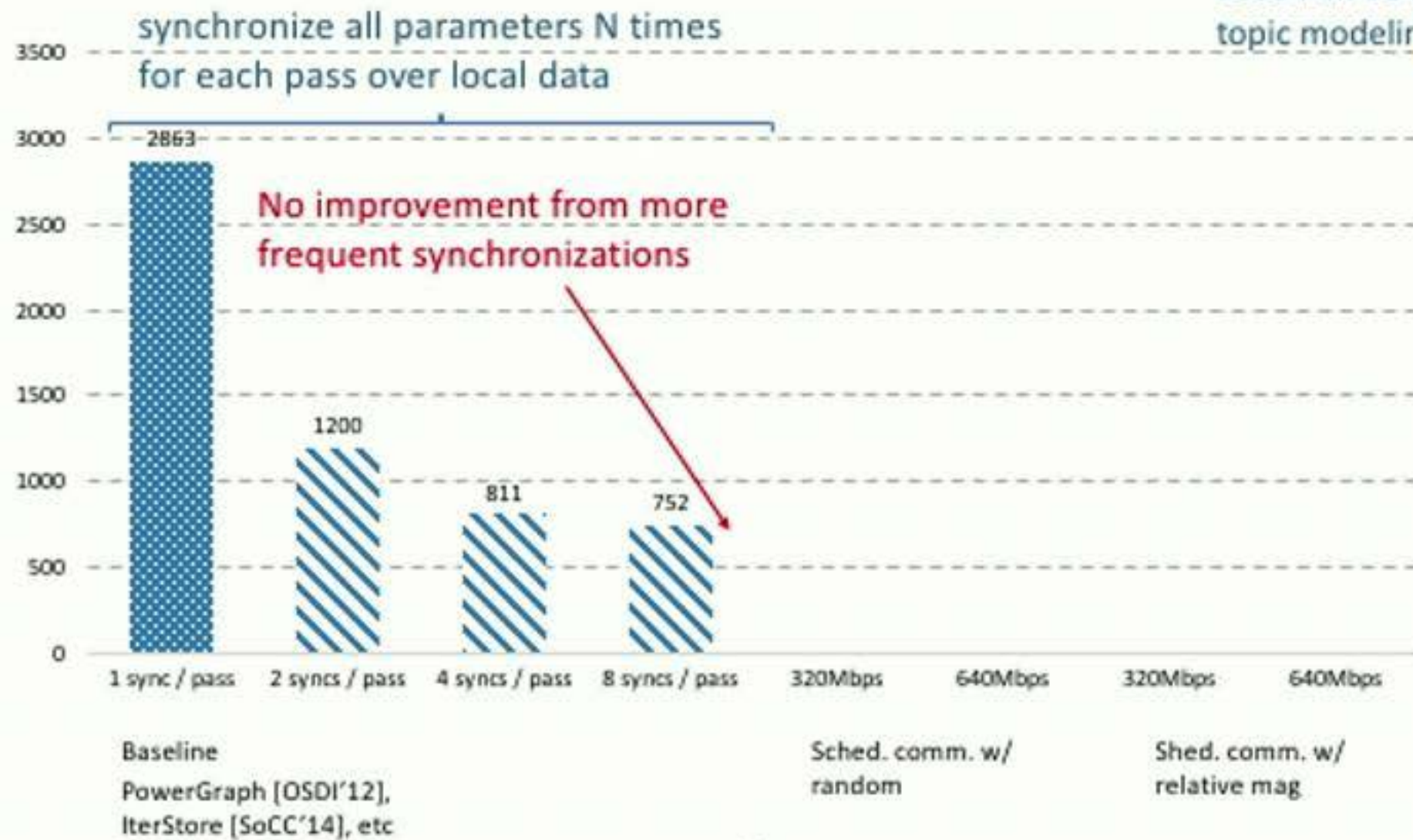
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps,
topic modeling



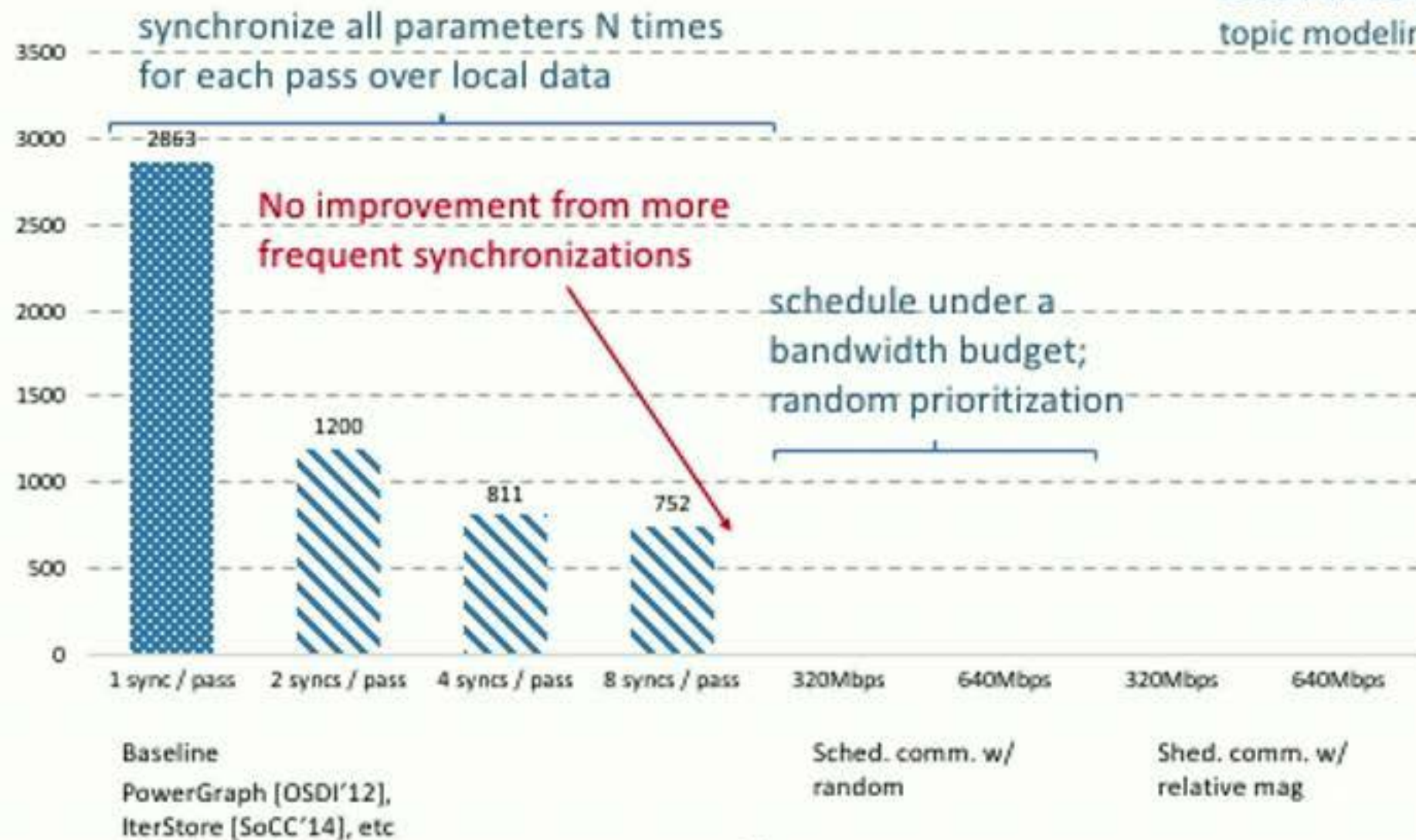
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps,
topic modeling



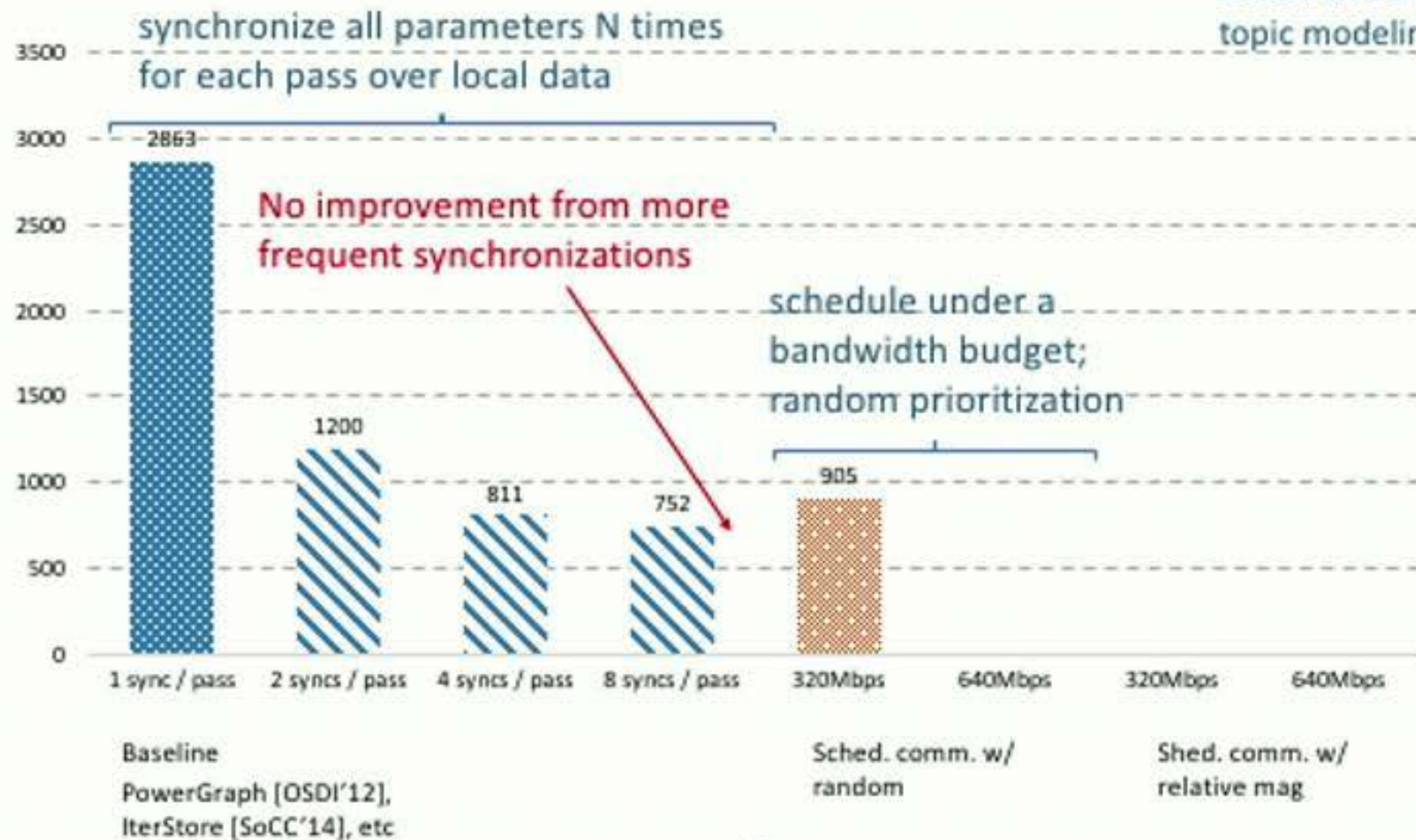
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps,
topic modeling



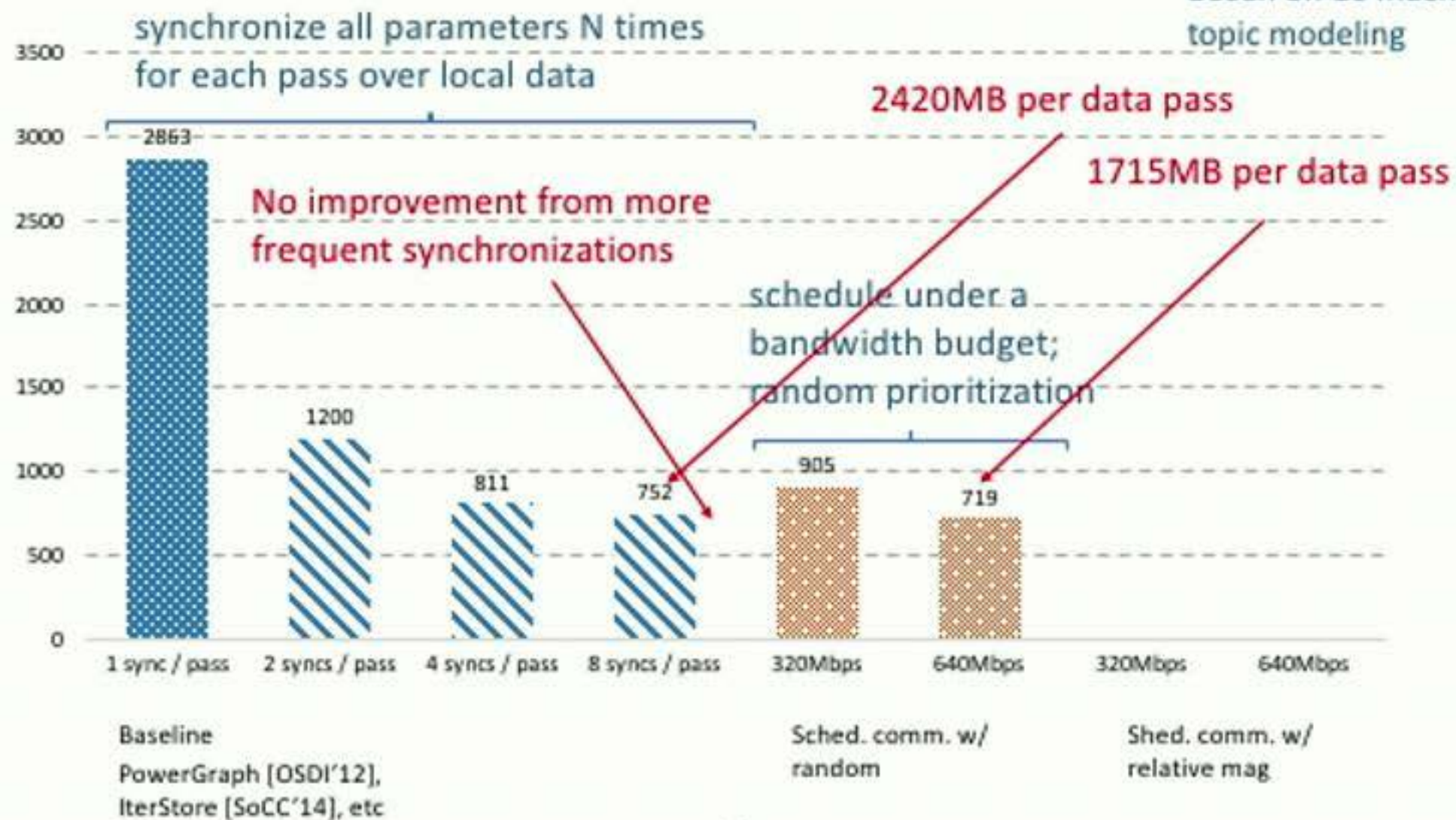
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps,
topic modeling



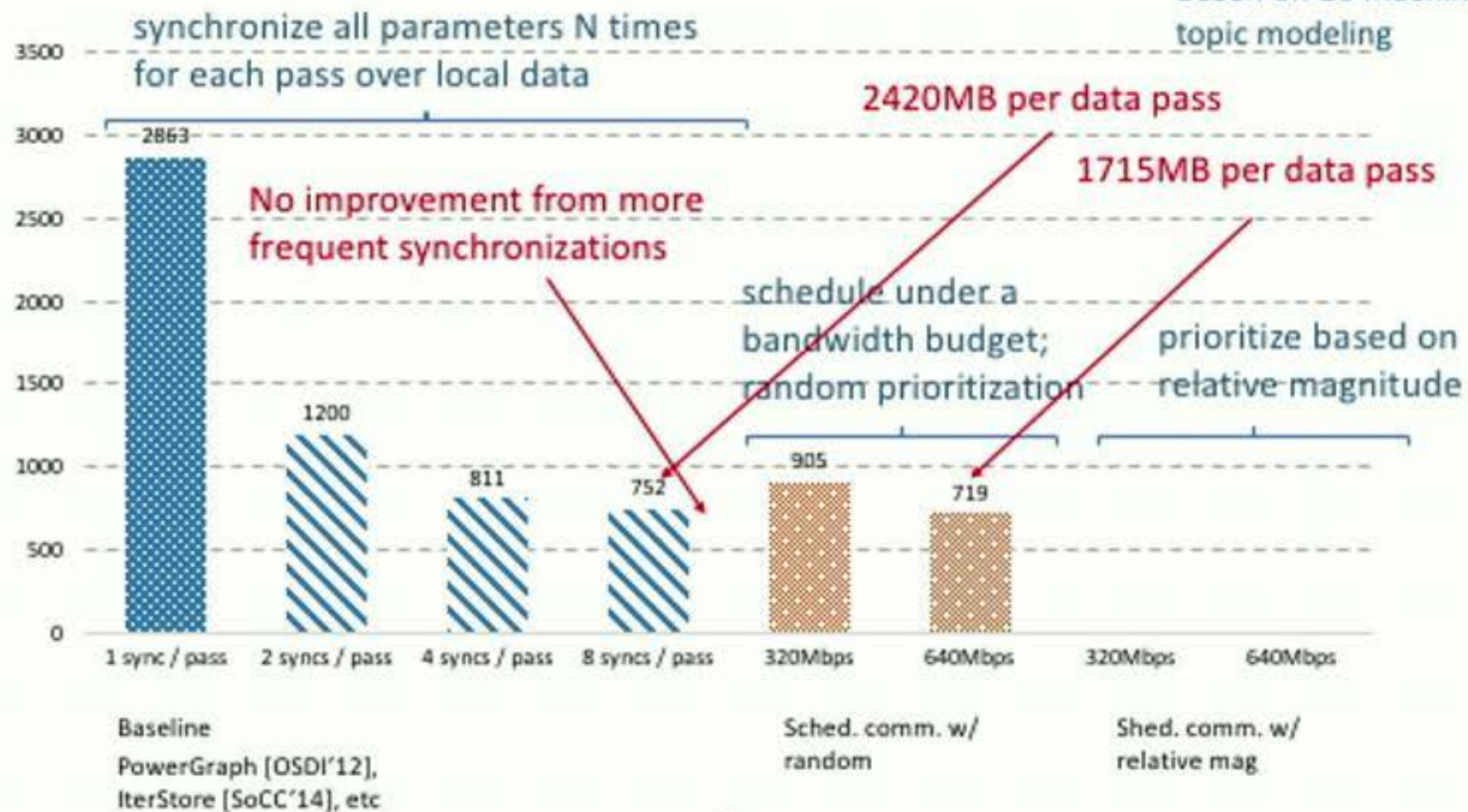
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps,
topic modeling



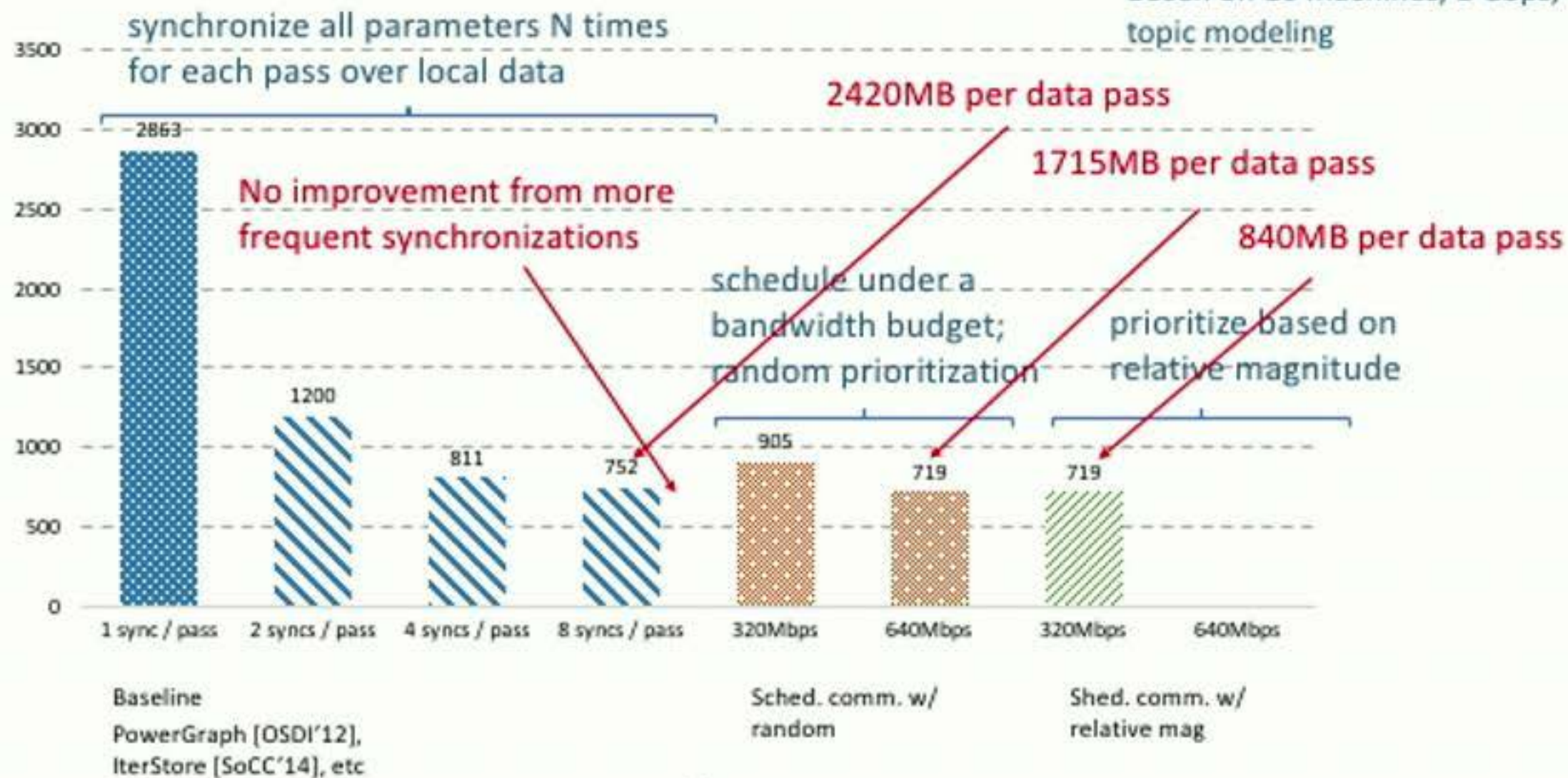
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps,
topic modeling



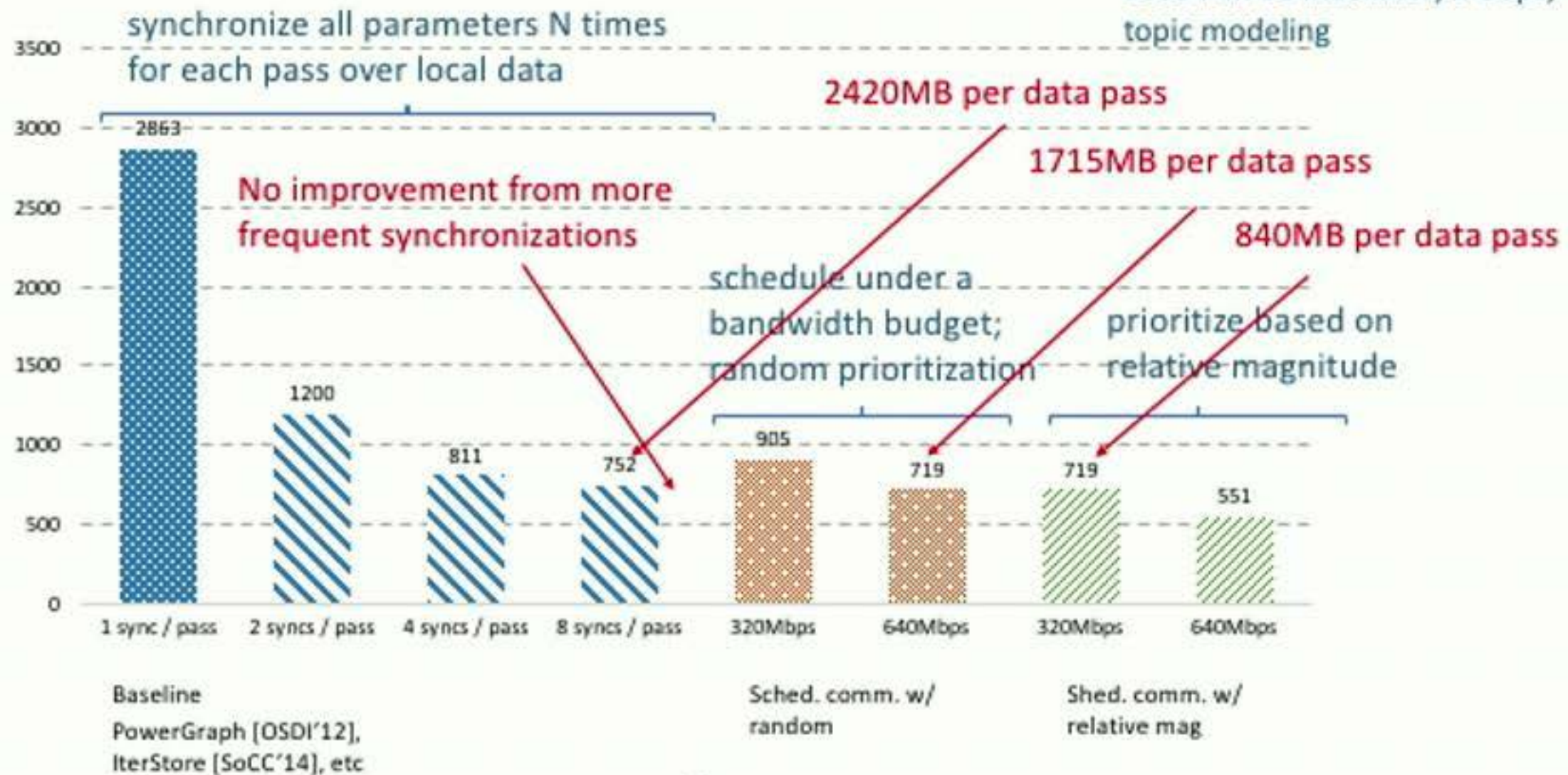
Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps, topic modeling



Experiment Results: Time to Convergence

Bösen on 16 machines, 1 Gbps,
topic modeling



Schedule Computation To Reduce Inconsistency



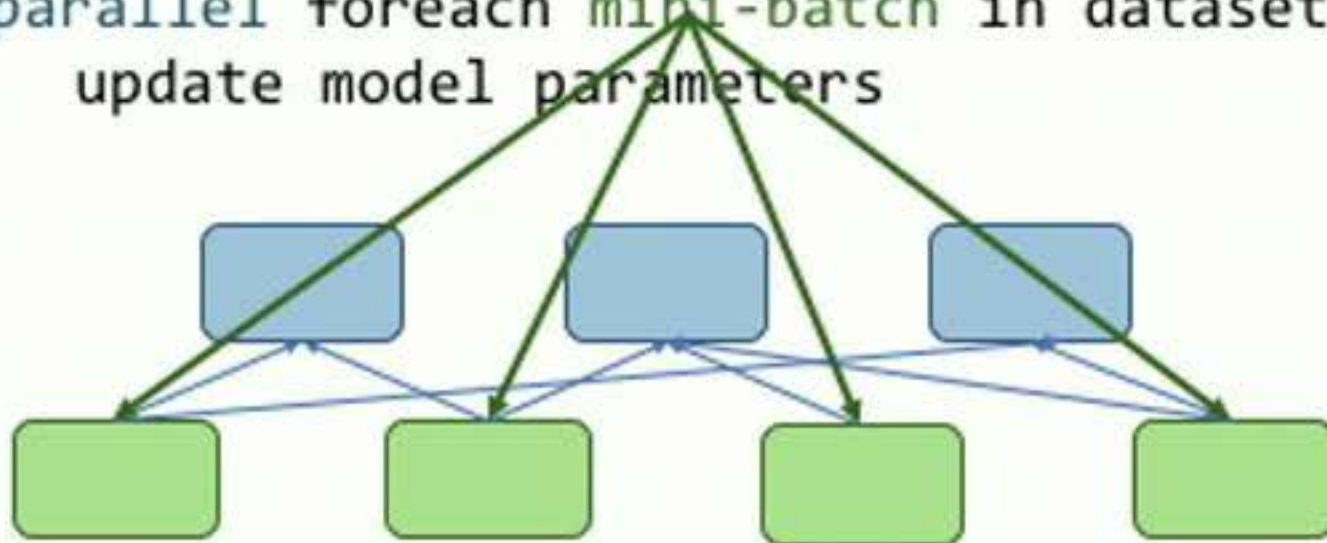
Execute only independent mini-batches in parallel

repeat until convergence

in parallel foreach mini-batch in dataset
update model parameters

Servers

Workers



Structural Sparse Parameter Access In ML

In some models, parameters are accessed based on data sample attributes.

Example:

Model: Matrix Factorization

Application: Recommender systems

Parameters: User Latent Vectors
Item Latent Vectors

Data sample:

UserID	ItemID	Rating
#38	#65	5.0

Structural Sparse Parameter Access In ML

In some models, parameters are accessed based on data sample attributes.

Example:

Model: Matrix Factorization

Application: Recommender systems

Parameters: User Latent Vectors

Item Latent Vectors

Data sample:

UserID	ItemID	Rating
#38	#65	5.0

There exist f_1, f_2, \dots, f_k , such that

if $d_i[f_1] \neq d_j[f_1]$, $d_i[f_2] \neq d_j[f_2]$, ..., and $d_i[f_k] \neq d_j[f_k]$,
 d_i and d_j don't access the same parameters.

Structural Sparse Parameter Access In ML

In some models, parameters are accessed based on data sample attributes.

Example:

Model: Matrix Factorization

Application: Recommender systems

Parameters: User Latent Vectors

Item Latent Vectors

Data sample:

UserID	ItemID	Rating
#38	#65	5.0

There exist f_1, f_2, \dots, f_k , such that

if $d_i[f_1] \neq d_j[f_1]$, $d_i[f_2] \neq d_j[f_2]$, ..., and $d_i[f_k] \neq d_j[f_k]$,
 d_i and d_j don't access the same parameters.

Other examples: topic modeling, gradient boosted trees, etc.

Partition The Dataset for Nonconflicting Accesses

There exist f_1, f_2, \dots, f_k , such that
if $d_i[f_1] \neq d_j[f_1]$, $d_i[f_2] \neq d_j[f_2]$, ..., and $d_i[f_k] \neq d_j[f_k]$,
 d_i and d_j don't access the same parameters.

Partition the dataset by those fields

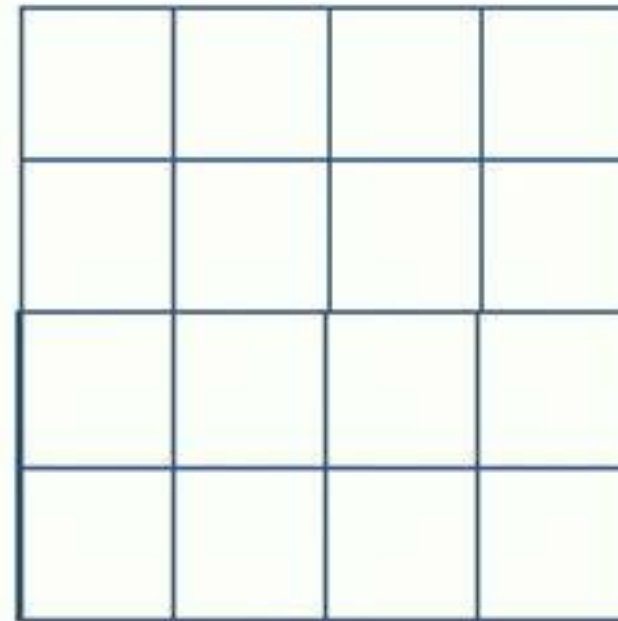


Dataset

Partition The Dataset for Nonconflicting Accesses

There exist f_1, f_2, \dots, f_k , such that
if $d_i[f_1] \neq d_j[f_1]$, $d_i[f_2] \neq d_j[f_2]$, ..., and $d_i[f_k] \neq d_j[f_k]$,
 d_i and d_j don't access the same parameters.

Partition the dataset by those fields



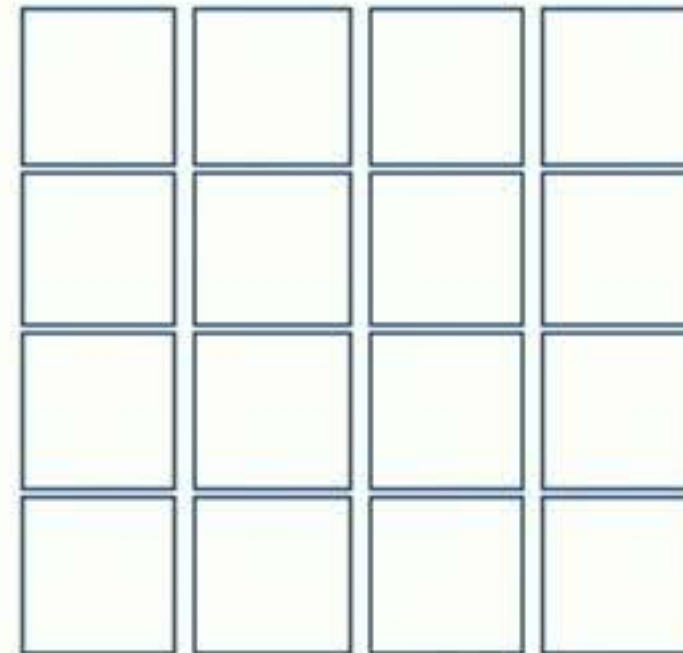
Dataset

Partition The Dataset for Nonconflicting Accesses

There exist f_1, f_2, \dots, f_k , such that

if $d_i[f_1] \neq d_j[f_1]$, $d_i[f_2] \neq d_j[f_2]$, ..., and $d_i[f_k] \neq d_j[f_k]$,
 d_i and d_j don't access the same parameters.

Partition the dataset by those fields



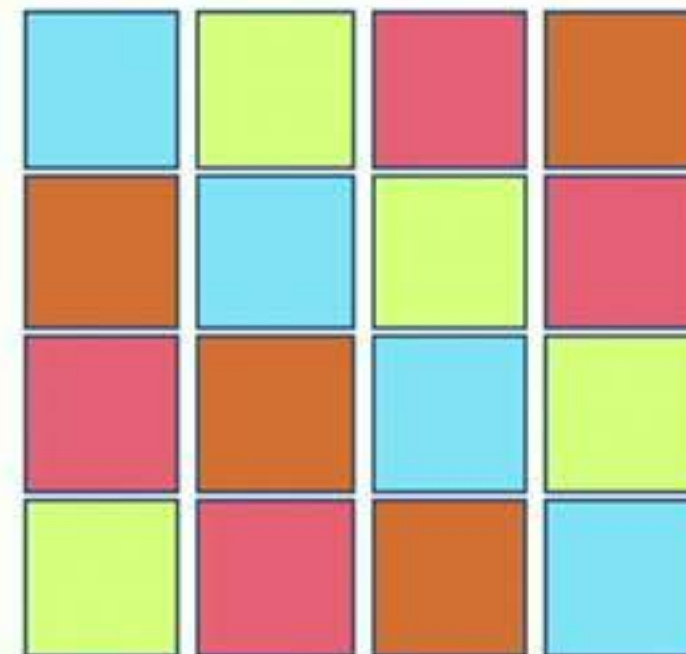
Dataset

Partition The Dataset for Nonconflicting Accesses

There exist f_1, f_2, \dots, f_k , such that
if $d_i[f_1] \neq d_j[f_1]$, $d_i[f_2] \neq d_j[f_2]$, ..., and $d_i[f_k] \neq d_j[f_k]$,
 d_i and d_j don't access the same parameters.

Partition the dataset by those fields

Nonconflicting parameter accesses



Dataset

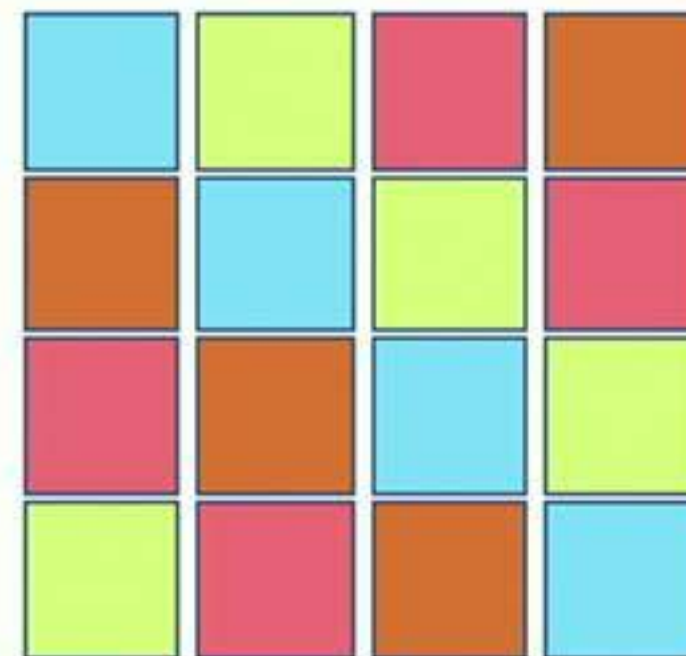
Partition The Dataset for Nonconflicting Accesses

There exist f_1, f_2, \dots, f_k , such that
if $d_i[f_1] \neq d_j[f_1]$, $d_i[f_2] \neq d_j[f_2]$, ..., and $d_i[f_k] \neq d_j[f_k]$,
 d_i and d_j don't access the same parameters.

Partition the dataset by those fields

Nonconflicting parameter accesses

Special case of automatic
parallelizing compilers



Dataset

Challenges for Scheduling Computation

Challenges for Scheduling Computation

Challenge #1: applicable only when certain property holds

Solution: fall back to data parallelism otherwise

Challenges for Scheduling Computation

Challenge #1: applicable only when certain property holds

Solution: fall back to data parallelism otherwise

Challenge #2: implementation requires non-trivial programmer effort

Solution: automatically parallelize serial programs

Orion: Automatic Parallelization [Wei, et al., EuroSys'19]



Our goals:

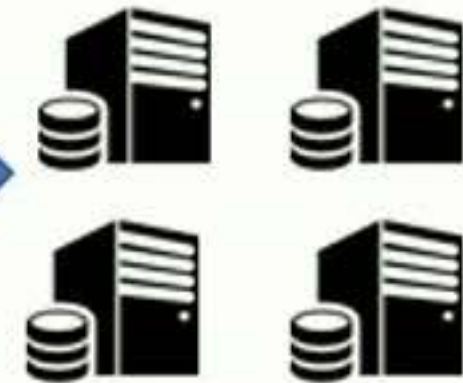
1. A `parallel_for` construct and users implement a serial program;
2. Preserves sequential semantics when possible;
3. Data parallelism otherwise with user permission



Serial ML program in Julia



Orion's abstraction:
A single thread w/ huge memory



Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine

40 Gbps Ethernet

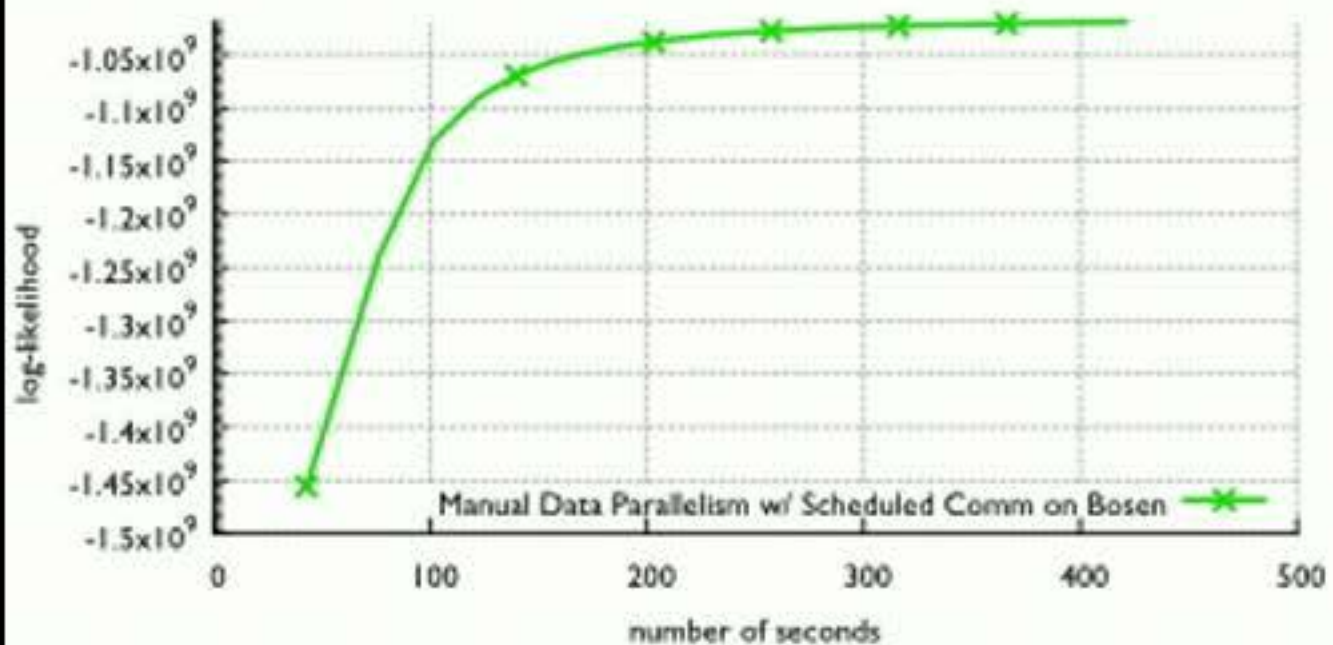
Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm

Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine

40 Gbps Ethernet

Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm

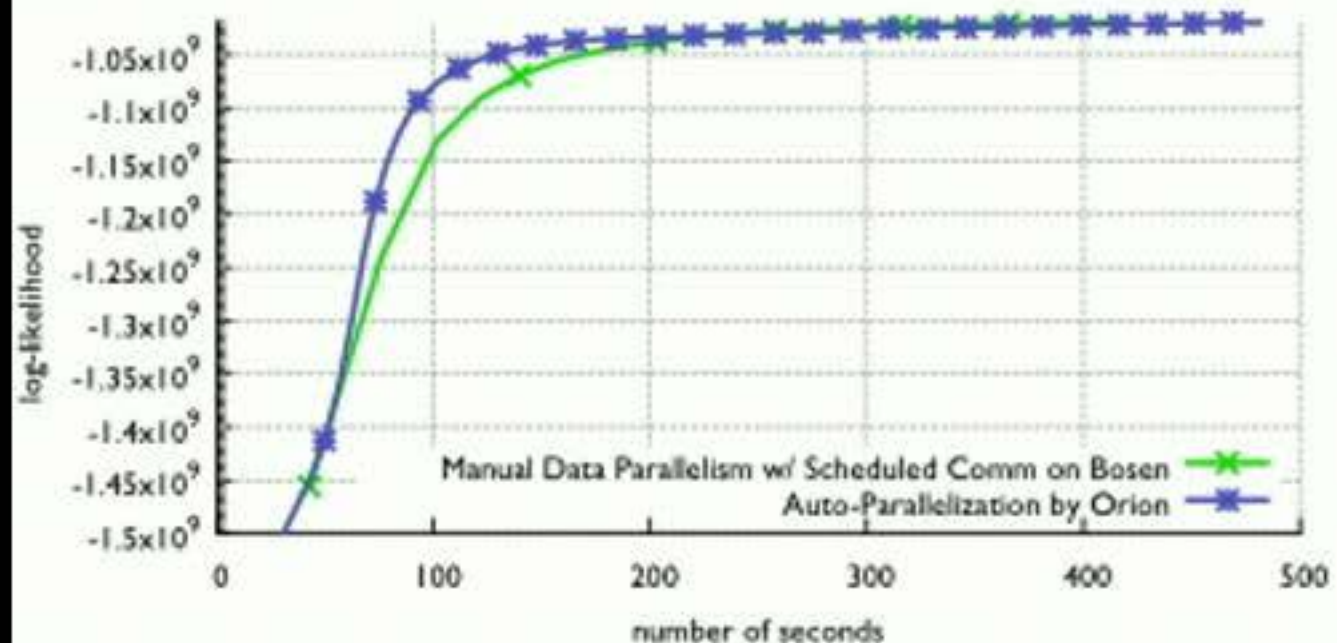


Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine

40 Gbps Ethernet

Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm

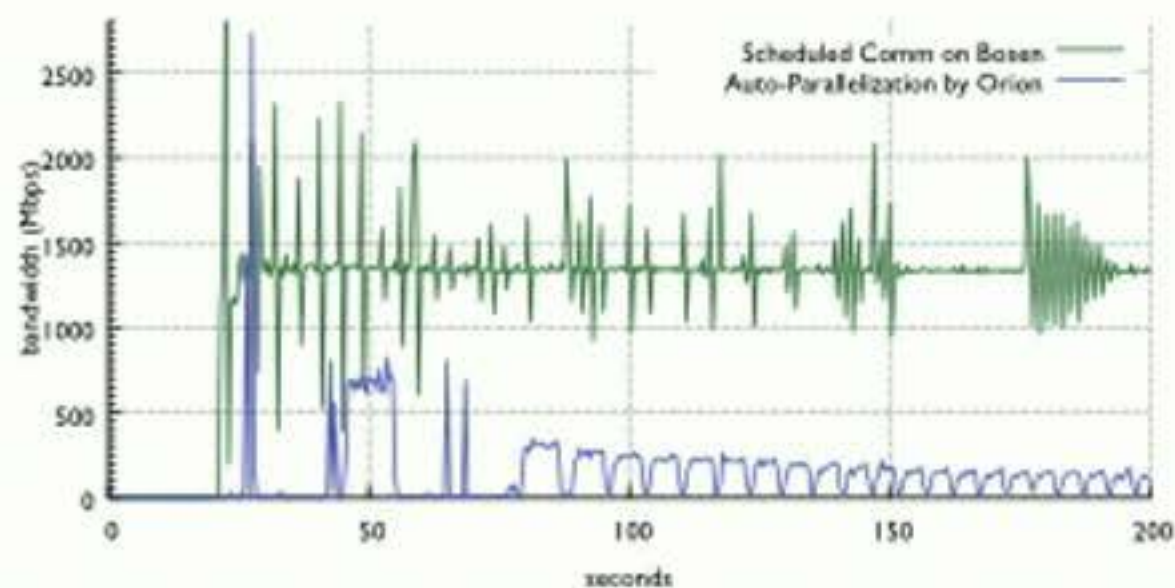
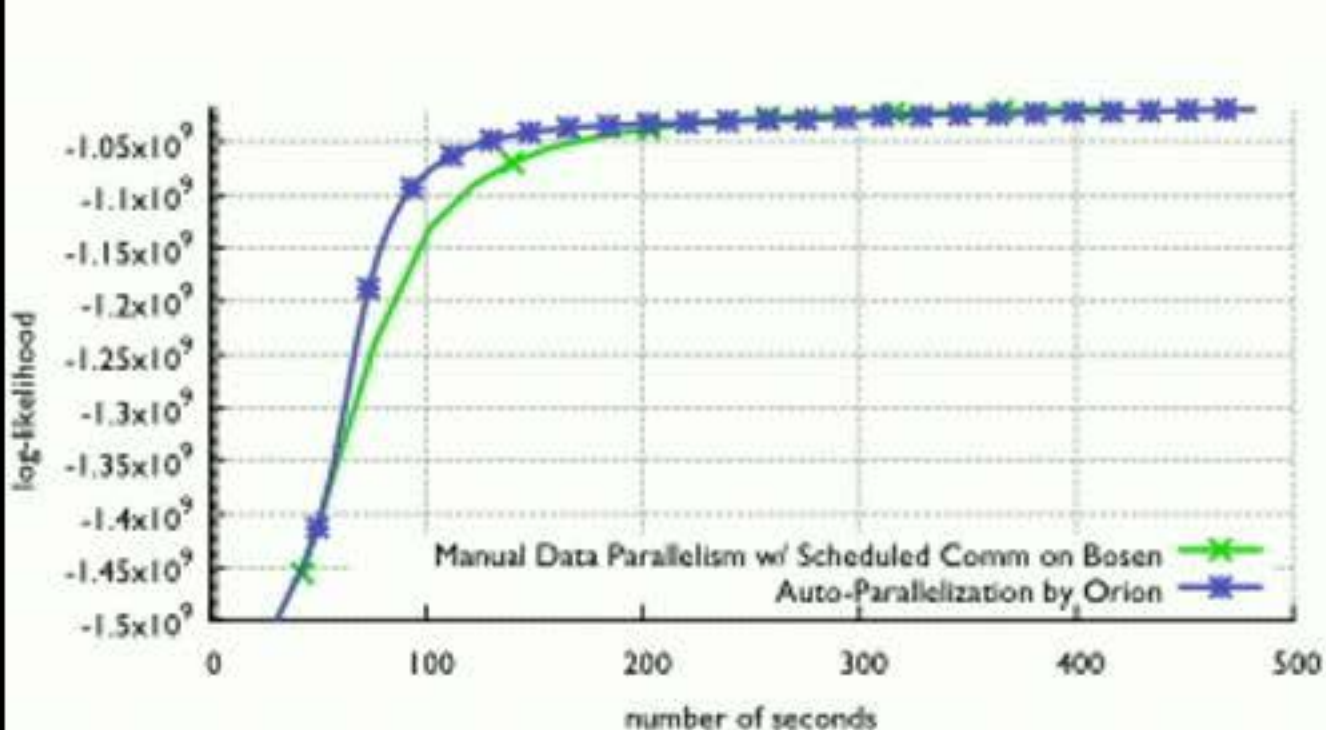


Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine

40 Gbps Ethernet

Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm

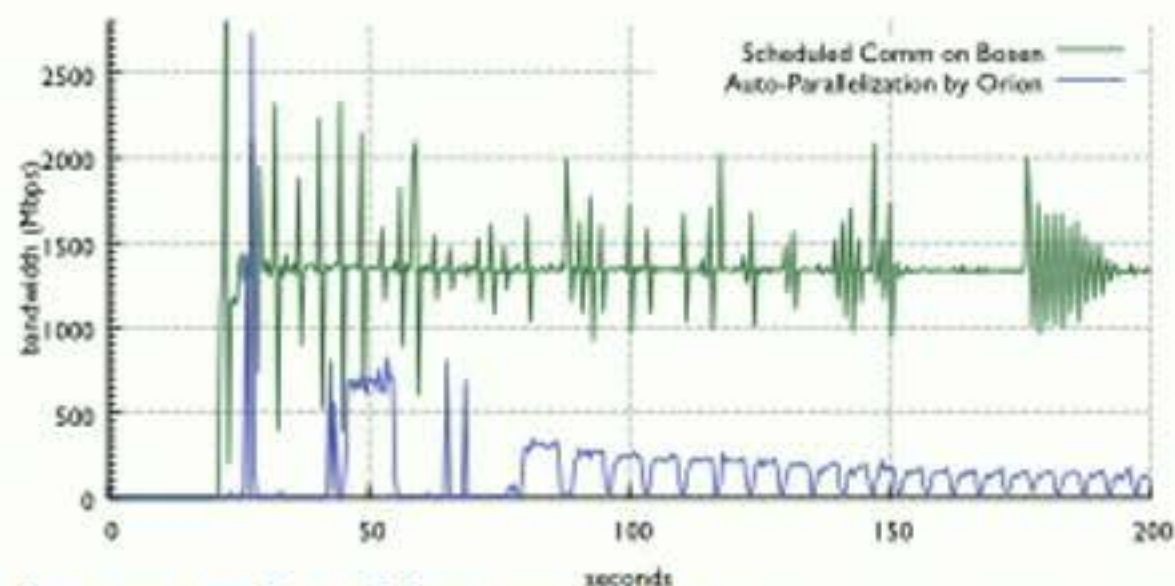
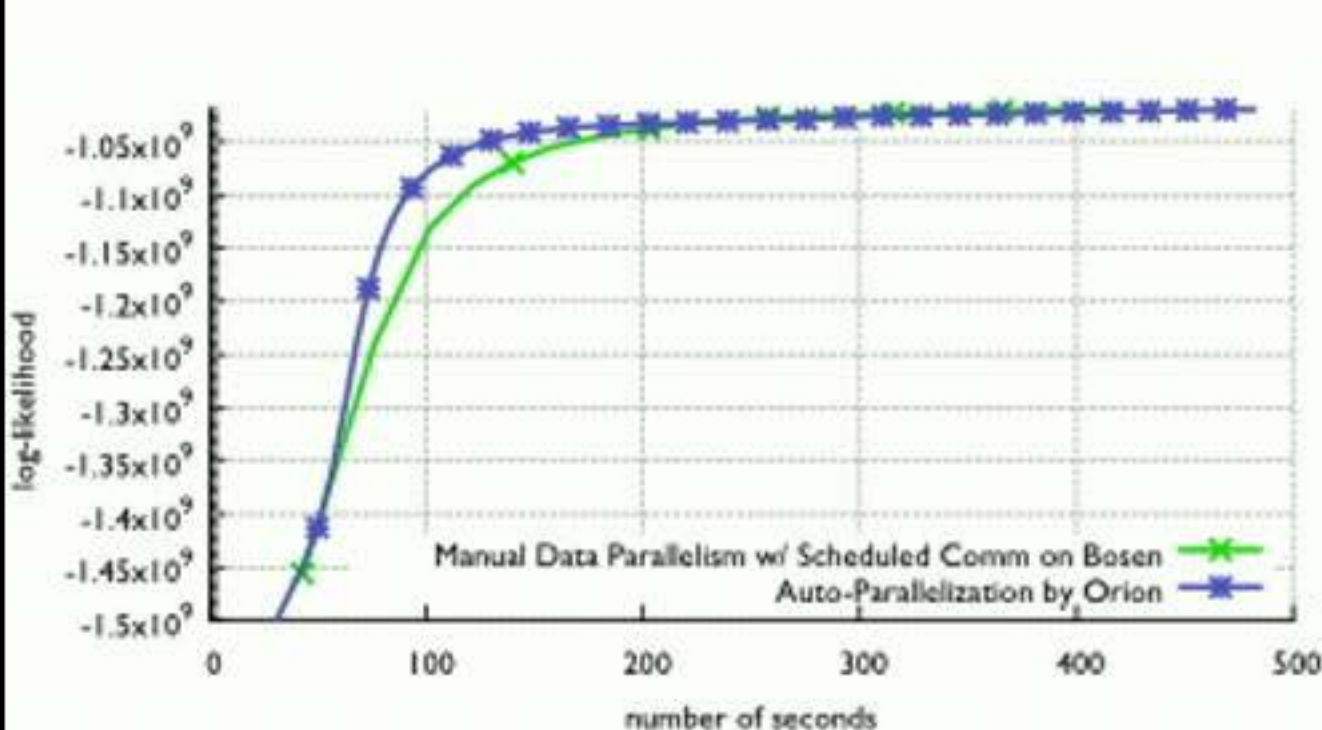


Compare Orion vs. Bösen

12 machines, 32 vCPU cores / machine

40 Gbps Ethernet

Latent Dirichlet Allocation (LDA) for topic modeling + Gibbs sampling algorithm



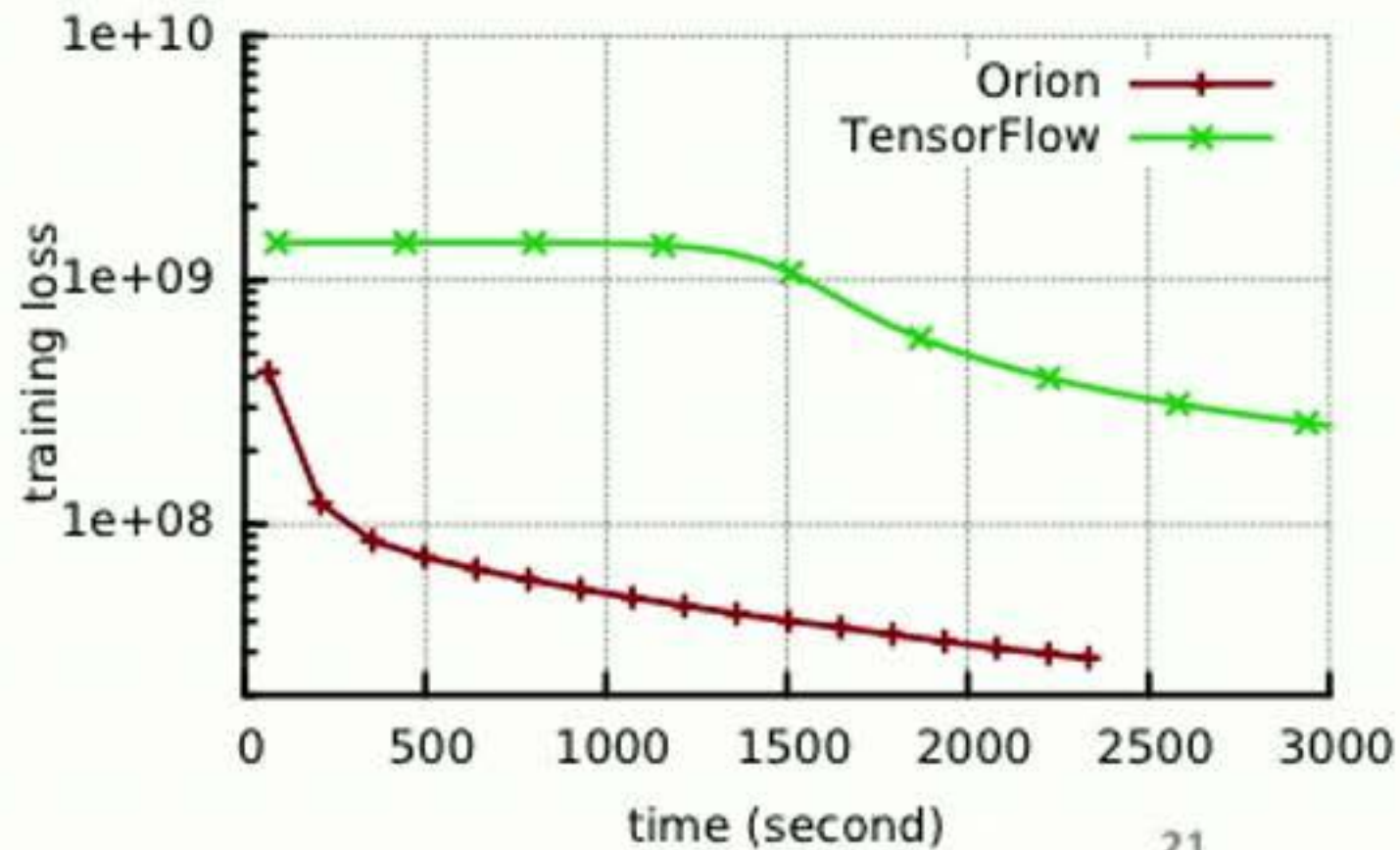
thousands of lines of C++

vs. a few hundreds of lines of Julia

Compare Orion vs. TensorFlow

1 machines, 32 vCPU cores / machine

Matrix Factorization (MF) for recommendations + SGD



TensorFlow suffers due to

- *Data parallelism*
- *Slower per-sample convergence*
- *Poor support for sparsity*
- *2x longer time per iteration*

The Developing View: More And More Complex ML Models

repeat until convergence

 foreach mini-batch in dataset

 update model parameters

We've focused improving computation across mini-batches.

The Machine Learning Trend:

The mini-batch computation is becoming more and more complex

The Developing View: More And More Complex ML Models

repeat until convergence

 foreach mini-batch in dataset

 update model parameters

We've focused improving computation across mini-batches.

The Machine Learning Trend:

The mini-batch computation is becoming more and more complex

Deep Neural Networks:

Heavy computation per mini-batch

Dense parameter access

Synchronize after each mini-batch

The Developing View: More And More Complex ML Models

repeat until convergence

`foreach mini-batch in dataset`

`update model parameters`

We've focused improving computation across mini-batches.

The Machine Learning Trend:

The mini-batch computation is becoming more and more complex

Deep Neural Networks:

Heavy computation per mini-batch

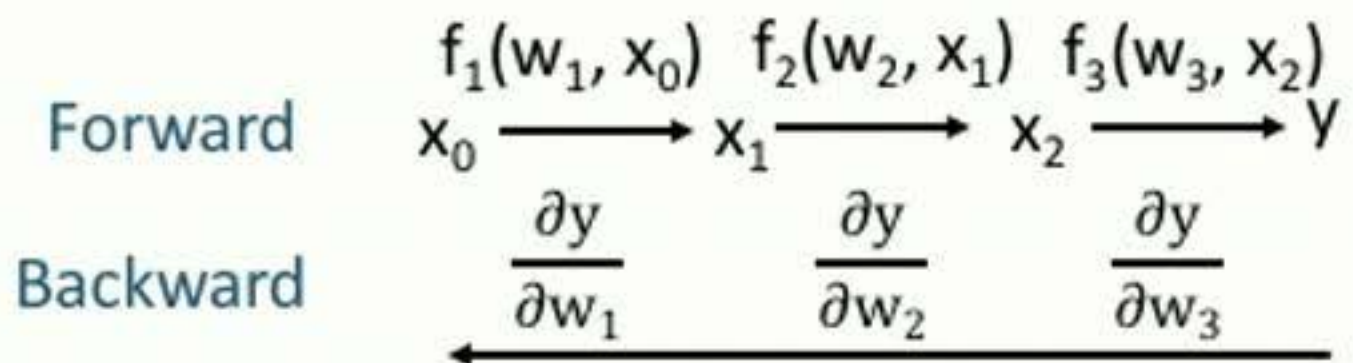
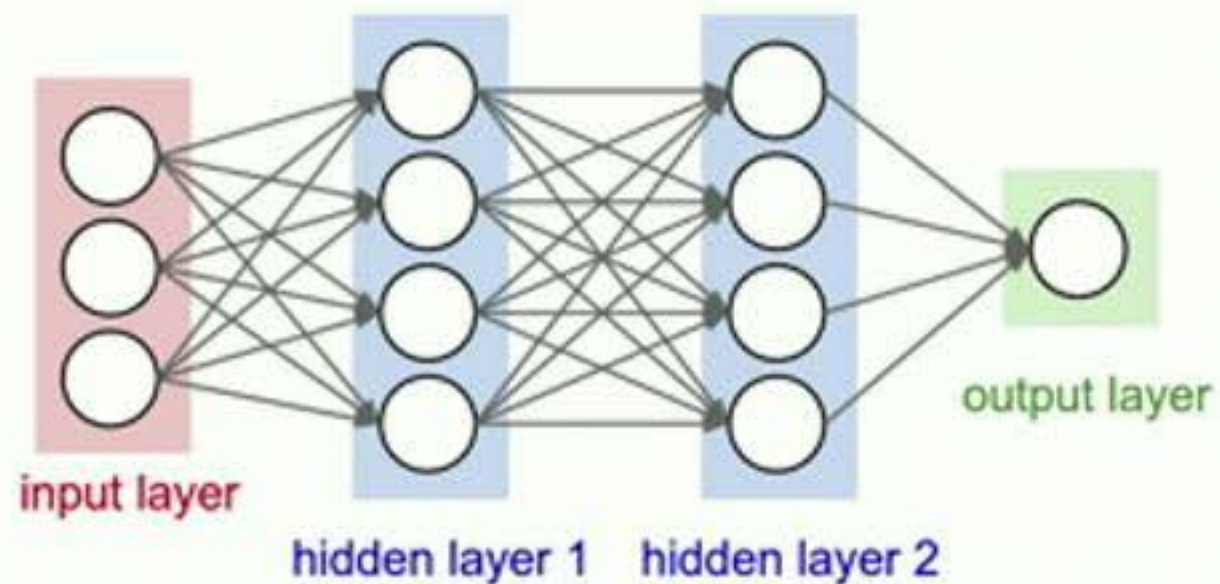
Dense parameter access

Synchronize after each mini-batch

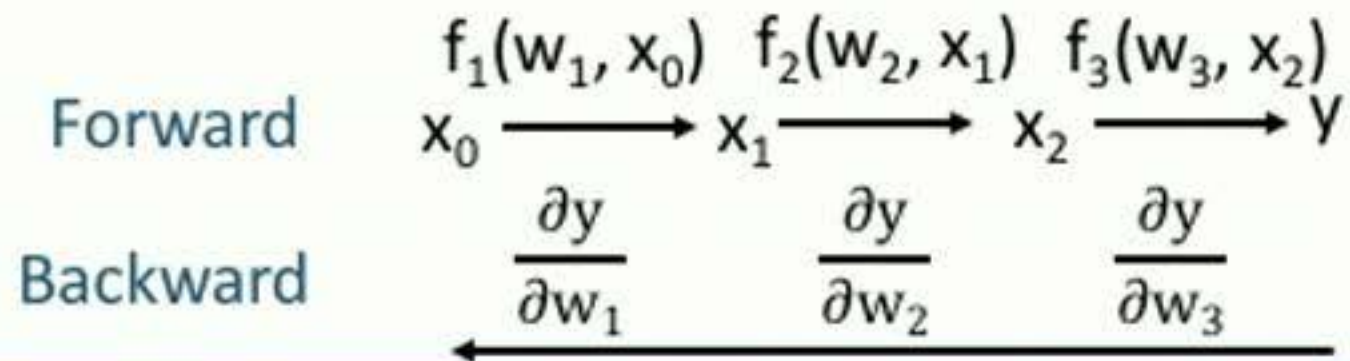
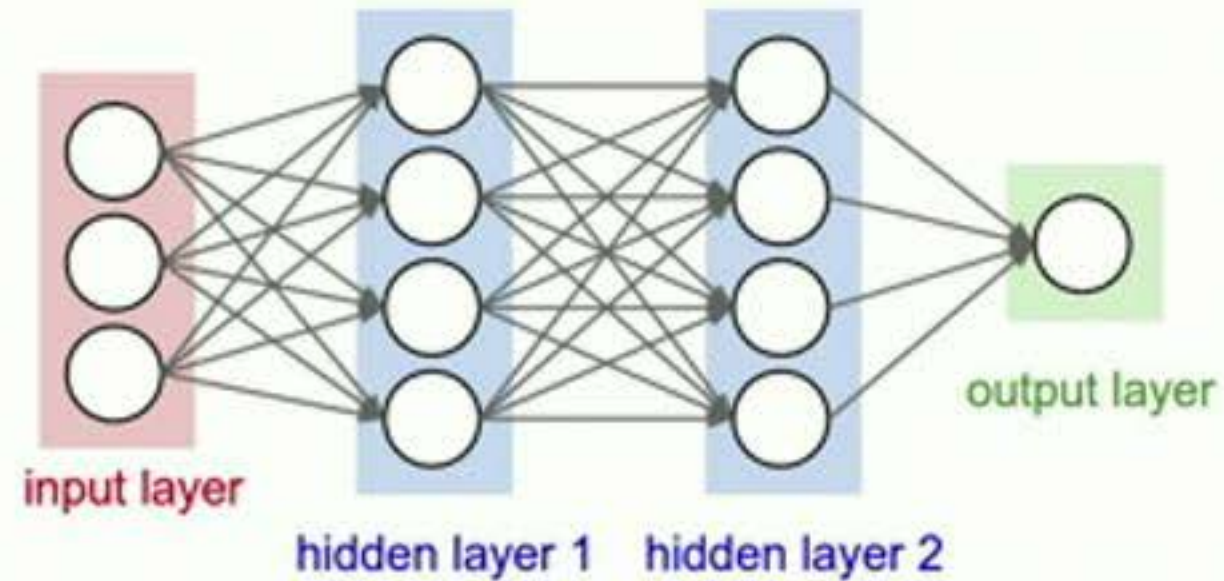
Opportunity:

Improve DNN efficiency without scarifying computation quality

The Mini-Batch Computation of Deep Neural Networks



The Mini-Batch Computation of Deep Neural Networks



Opportunity:

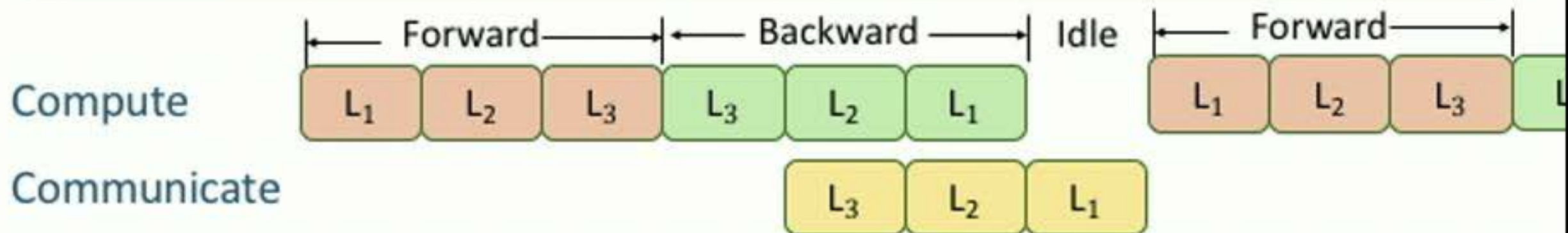
Not all parameters (updates) are needed (generated) at the same time

Schedule Communication Within A Mini-Batch for DNNs



Wait-Free Back Propagation: [Zhang et al., ATC'17] (coauthor)

Send updates layer by layer in the backward order, i.e., as soon as they are generated

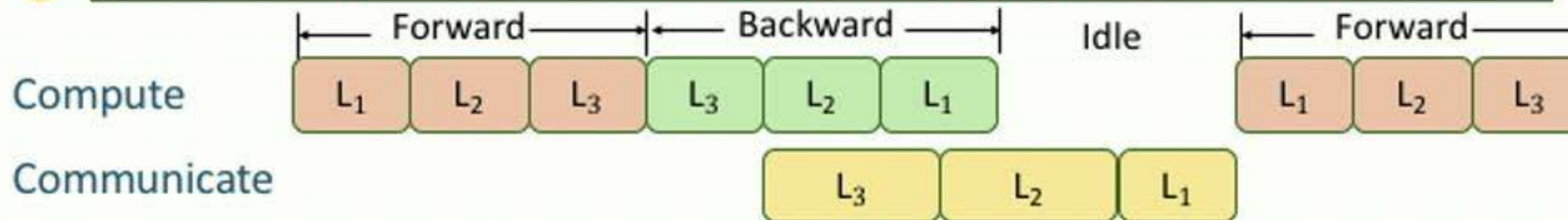


Overlap backward computation with communication within a mini-batch
Ideally, computation is idle only during the first layer's communication

Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



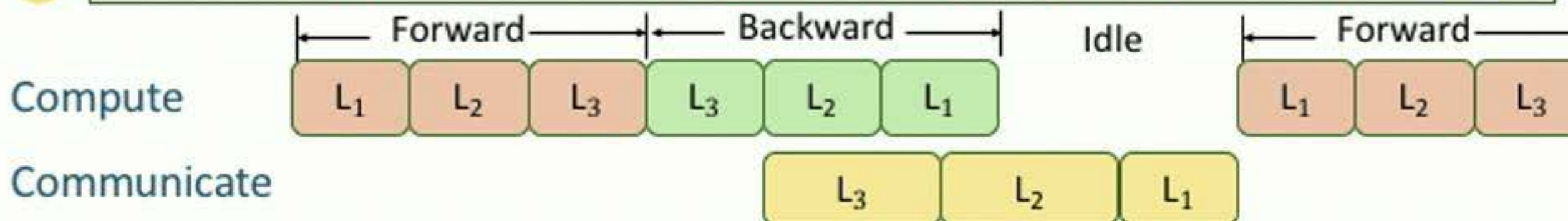
But the first layer's communication could be delayed by the previous layers



Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



But the first layer's communication could be delayed by the previous layers

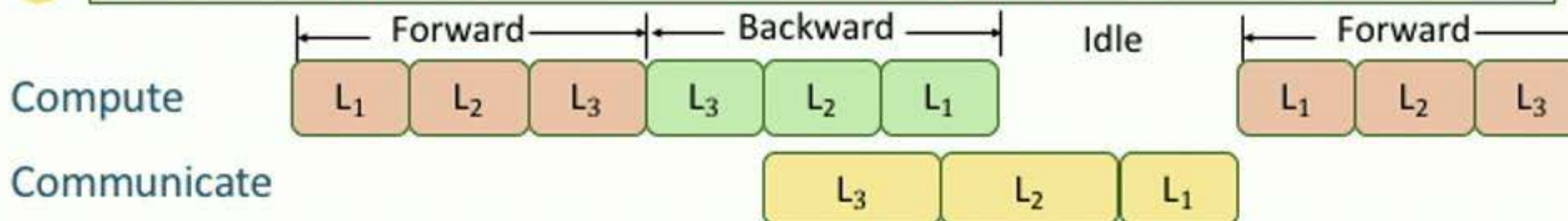


Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed

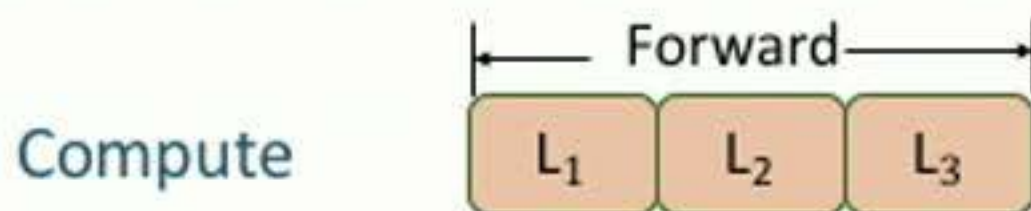
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



But the first layer's communication could be delayed by the previous layers



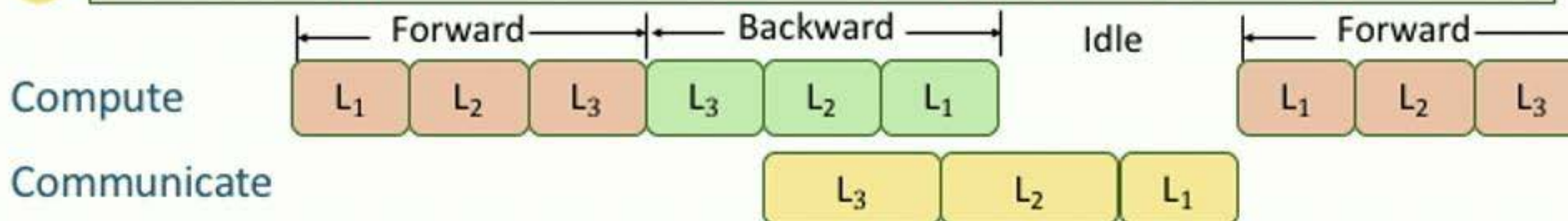
Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



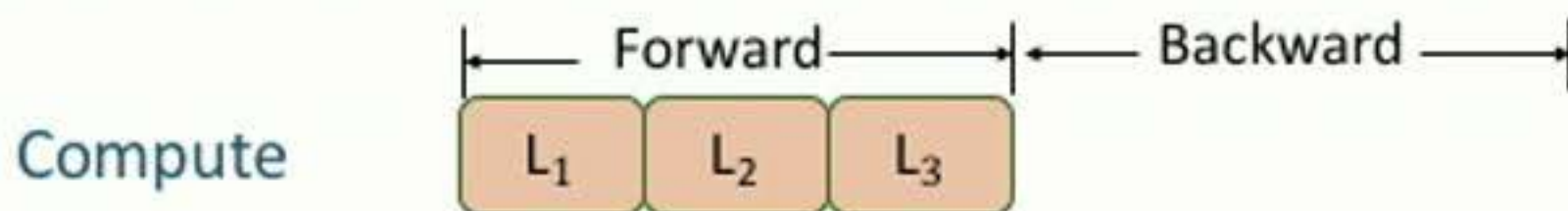
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



But the first layer's communication could be delayed by the previous layers



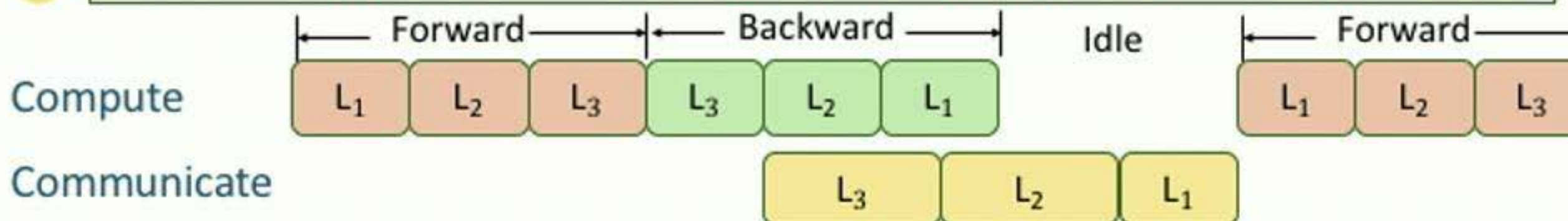
Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



But the first layer's communication could be delayed by the previous layers



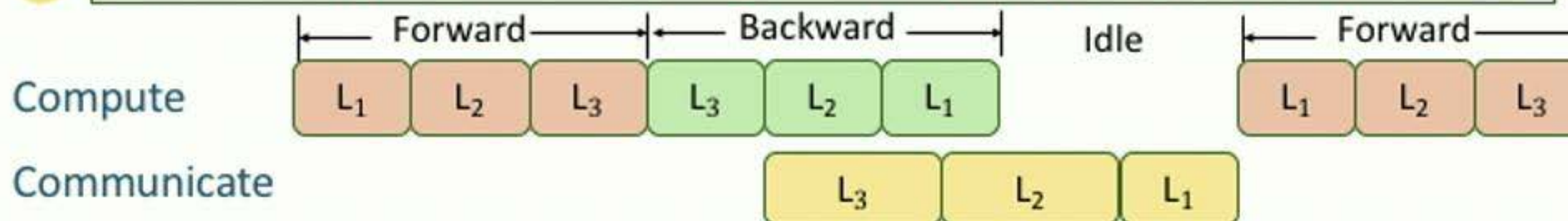
Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



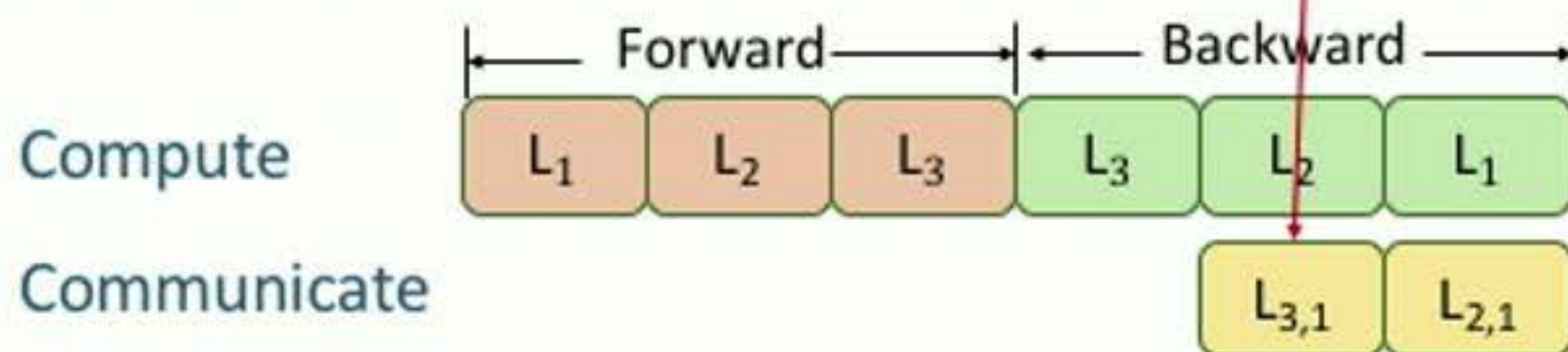
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



But the first layer's communication could be delayed by the previous layers



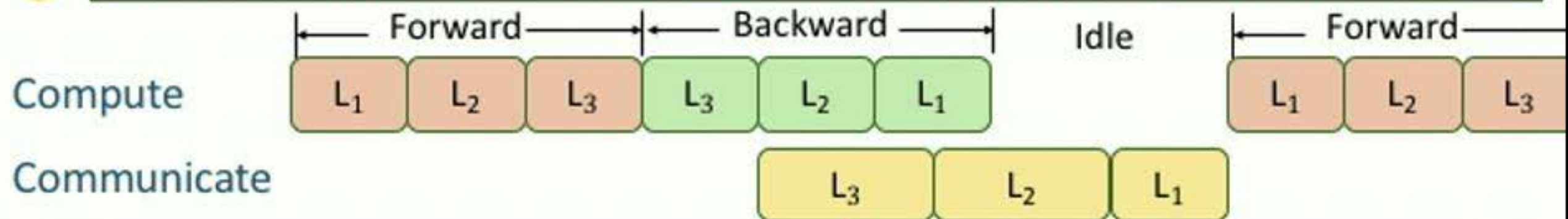
Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



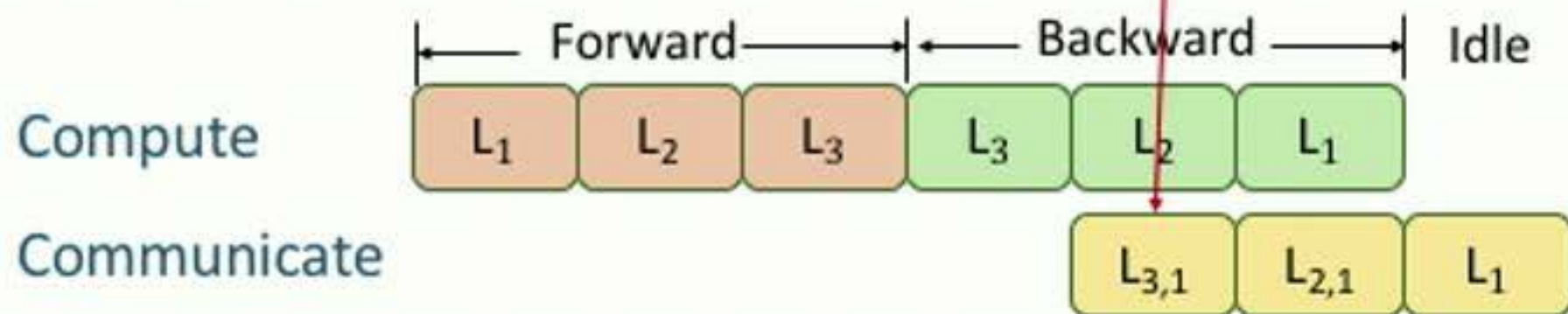
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



But the first layer's communication could be delayed by the previous layers



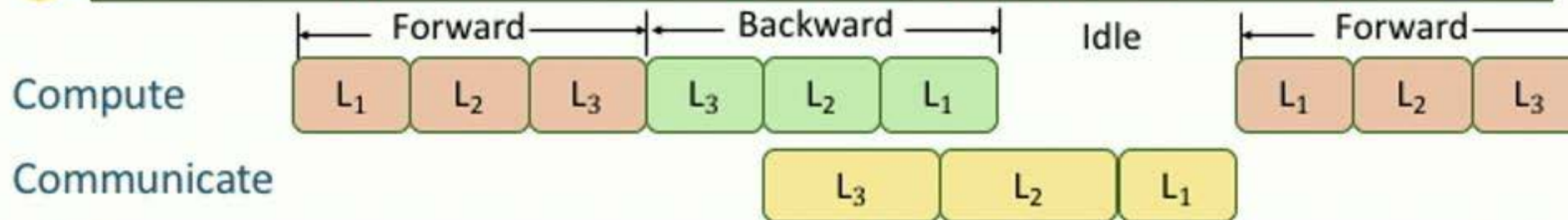
Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



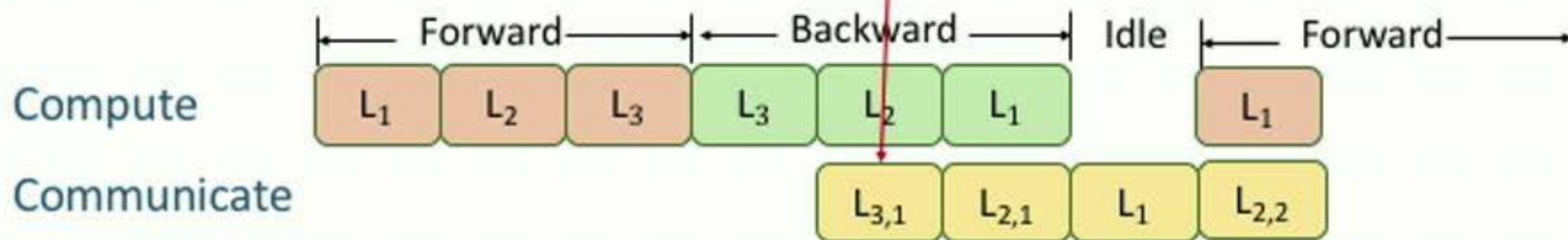
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



But the first layer's communication could be delayed by the previous layers



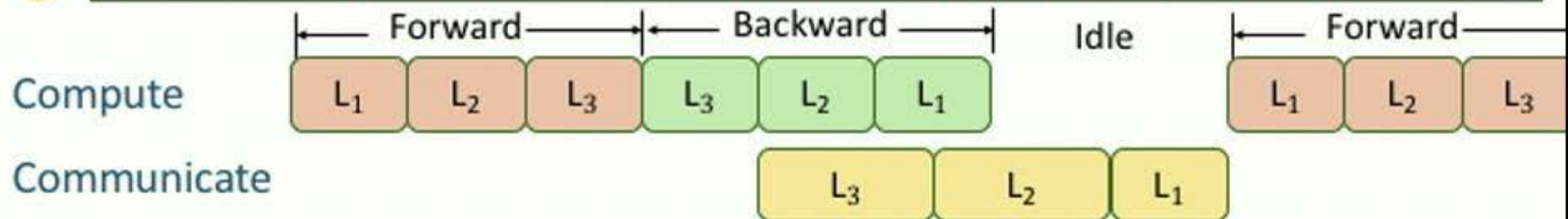
Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



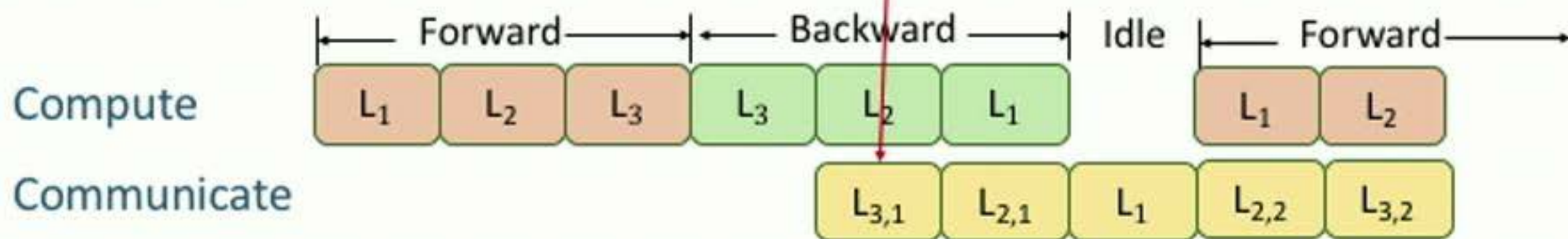
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



But the first layer's communication could be delayed by the previous layers



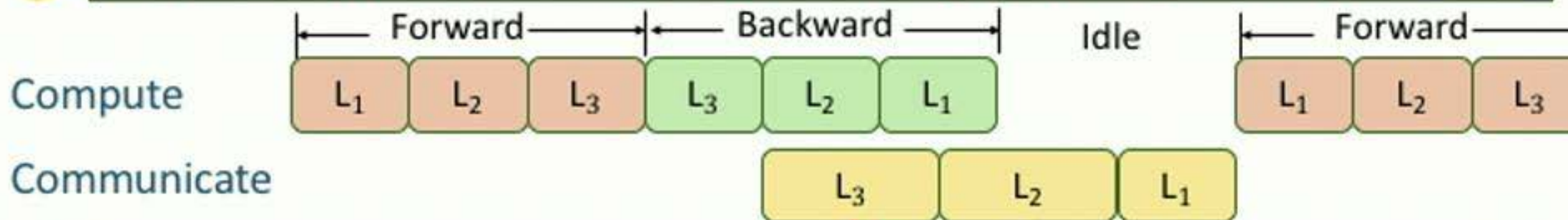
Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



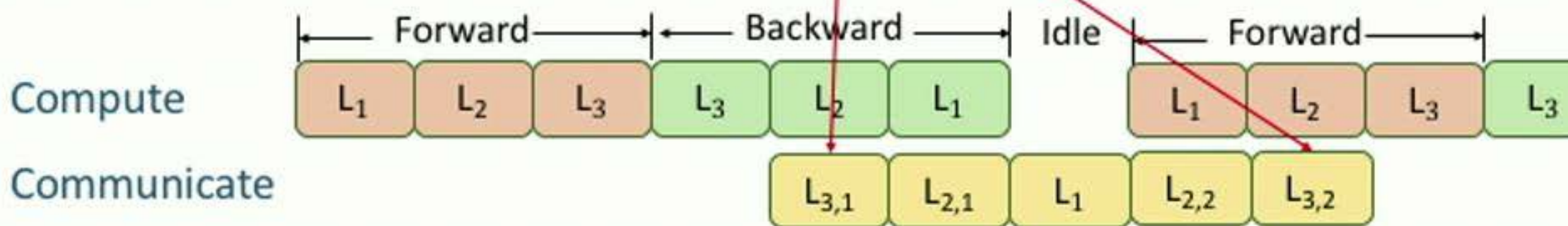
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



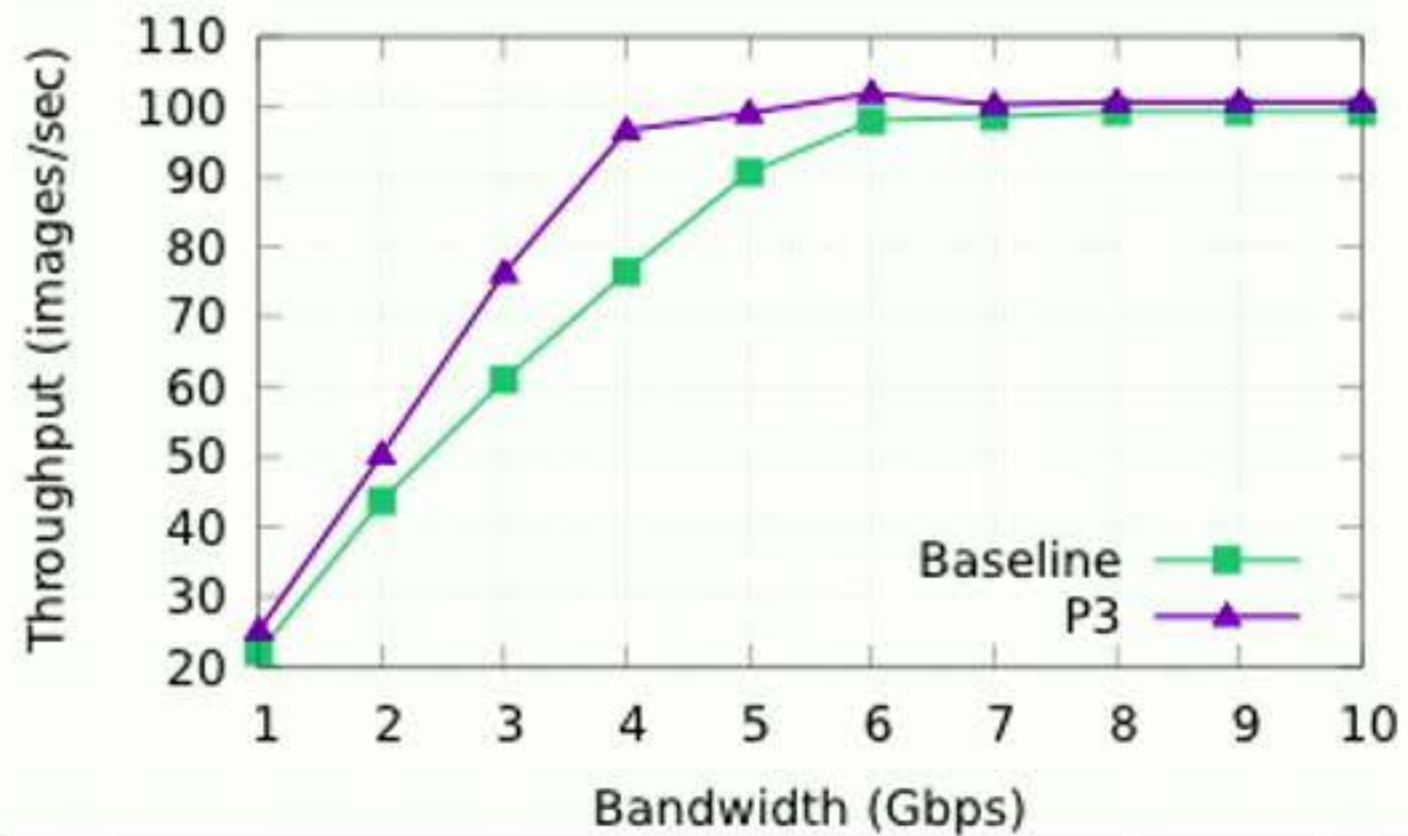
But the first layer's communication could be delayed by the previous layers



Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



P3 Experiment Results on MXNet

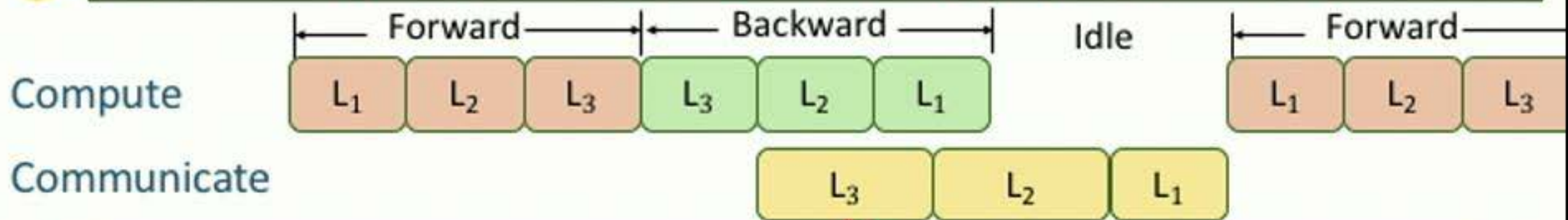


Baseline: MXNet (w/ Wait-Free Backpropagation)
Model: ResNet-50

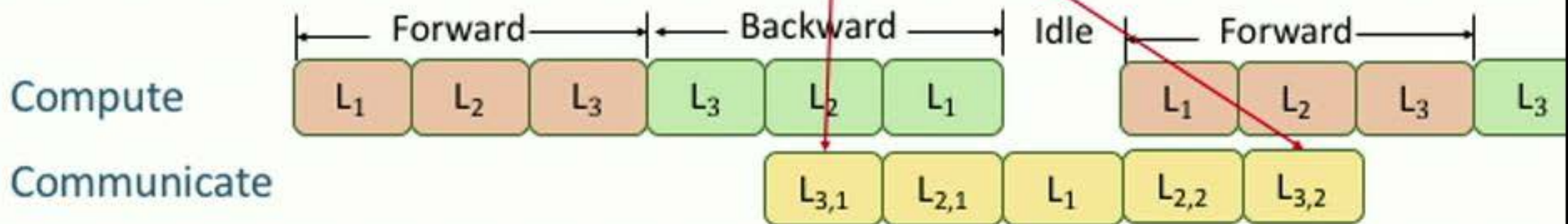
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



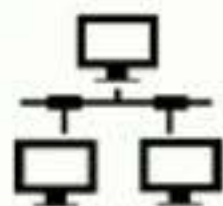
But the first layer's communication could be delayed by the previous layers



Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



Scheduling within a Single Training Job

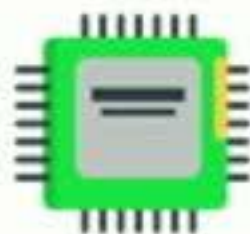


Network Communication

When and what to send?

Lead [SoCC'15, Best paper]

Coauthor [ATC'17] [SysML'19]



Computation

What to compute in parallel?

[EuroSys'19]



Memory Allocation

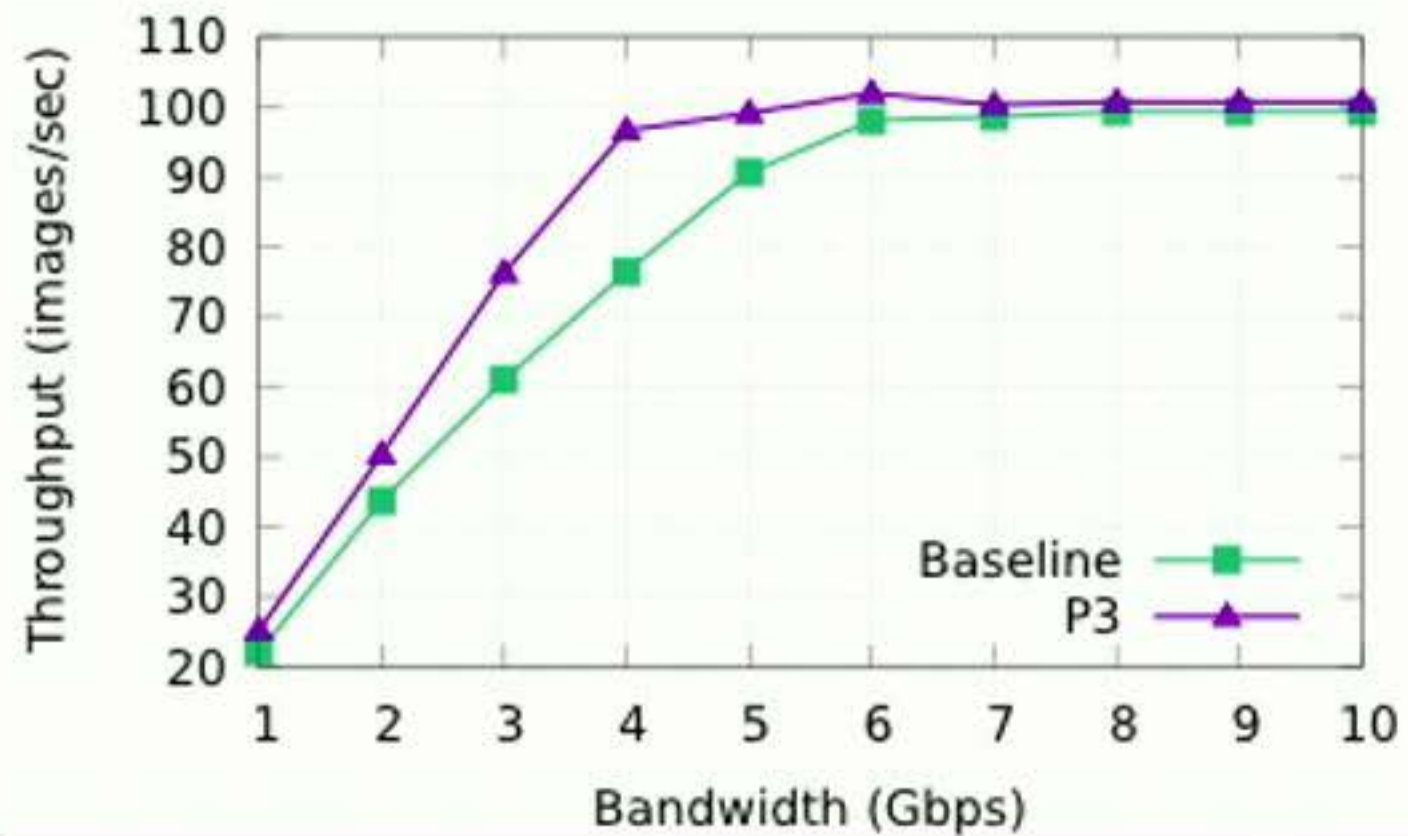
When and where to allocate?

[In preparation]

Highlights of results:

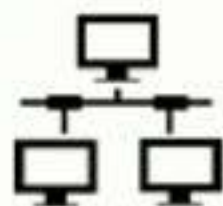
- Scheduling communication: up to 30% faster convergence
- Scheduling computation: even faster convergence with less programmer effort
- **Scheduling memory: 10x bigger model on the same hardware**

P3 Experiment Results on MXNet



Baseline: MXNet (w/ Wait-Free Backpropagation)
Model: ResNet-50

Scheduling within a Single Training Job

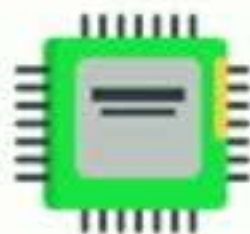


Network Communication

When and what to send?

Lead [SoCC'15, Best paper]

Coauthor [ATC'17] [SysML'19]



Computation

What to compute in parallel?

[EuroSys'19]



Memory Allocation

When and where to allocate?

[In preparation]

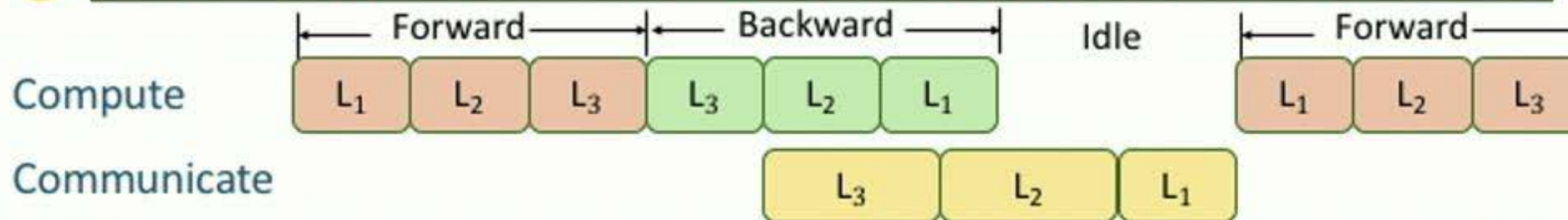
Highlights of results:

- Scheduling communication: up to 30% faster convergence
- Scheduling computation: even faster convergence with less programmer effort
- **Scheduling memory: 10x bigger model on the same hardware**

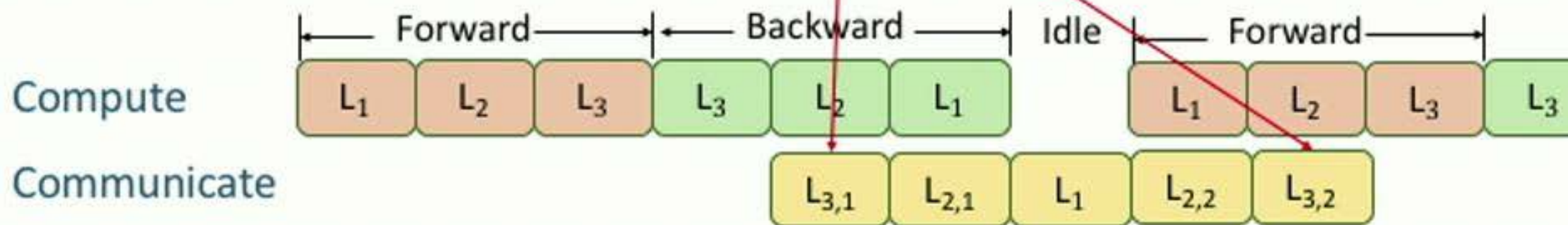
Schedule Comm. Within A Mini-Batch for DNNs (Cont.)



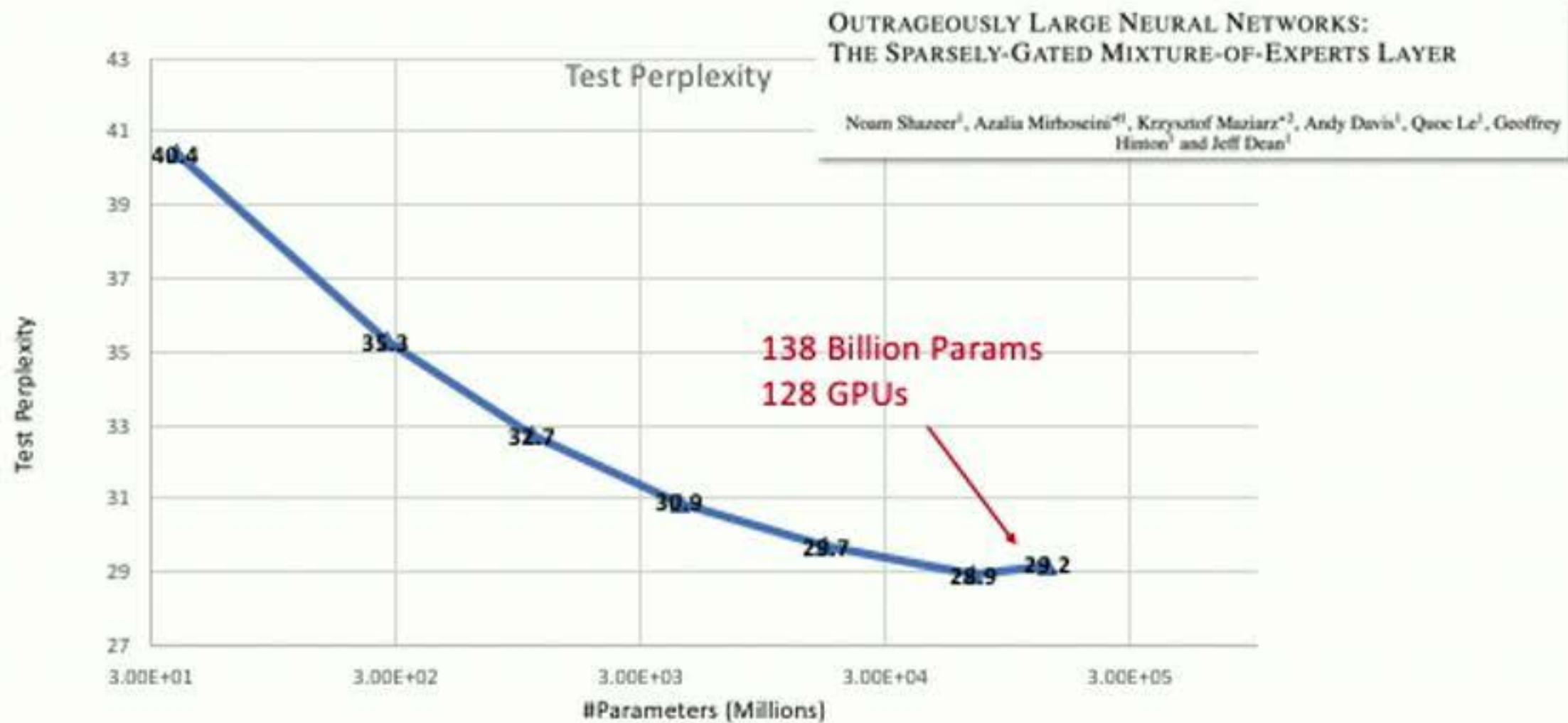
But the first layer's communication could be delayed by the previous layers



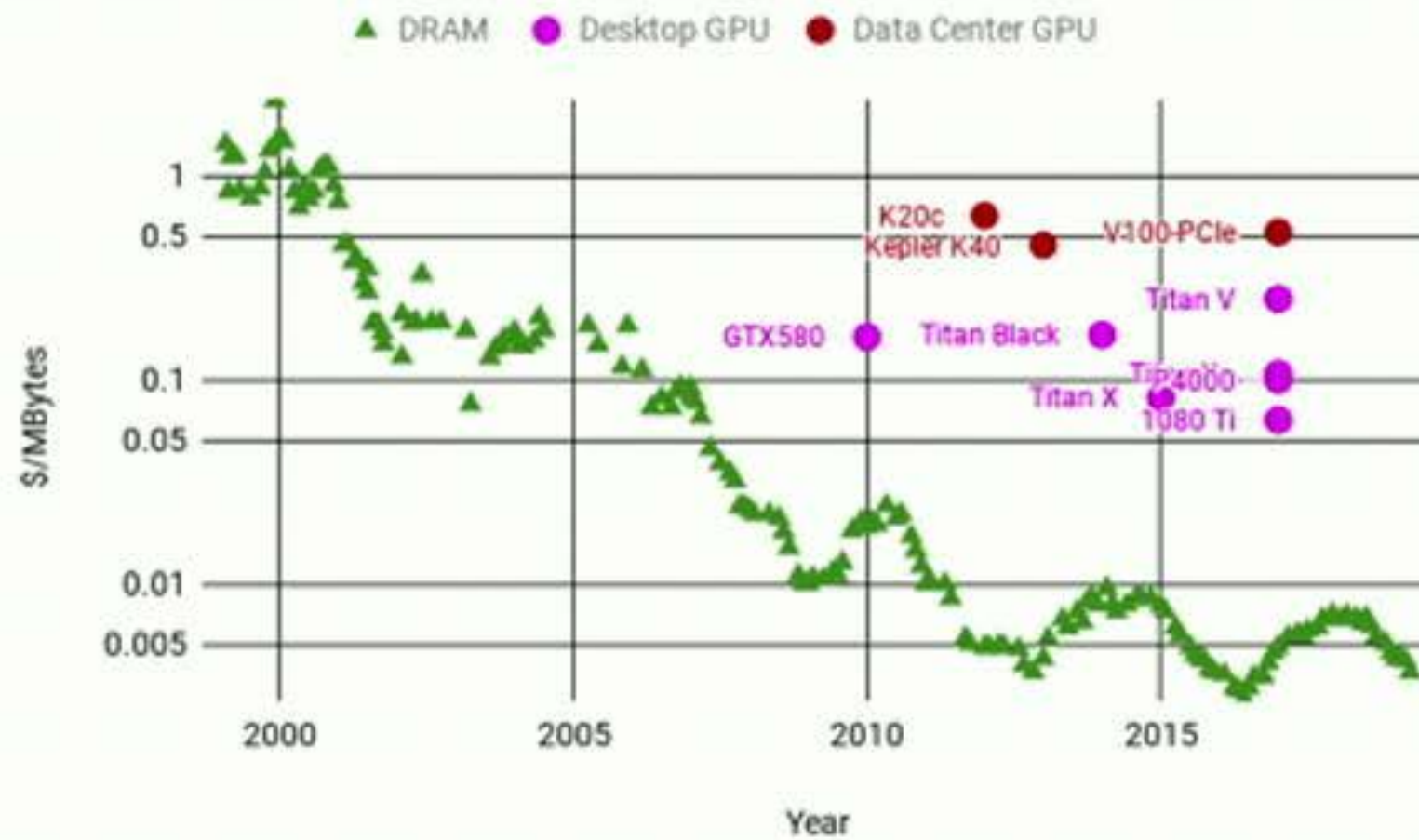
Priority-based Parameter Propagation: [Jayarajan et al., SysML'19] (coauthor)
Prioritize communication based on when the value is needed



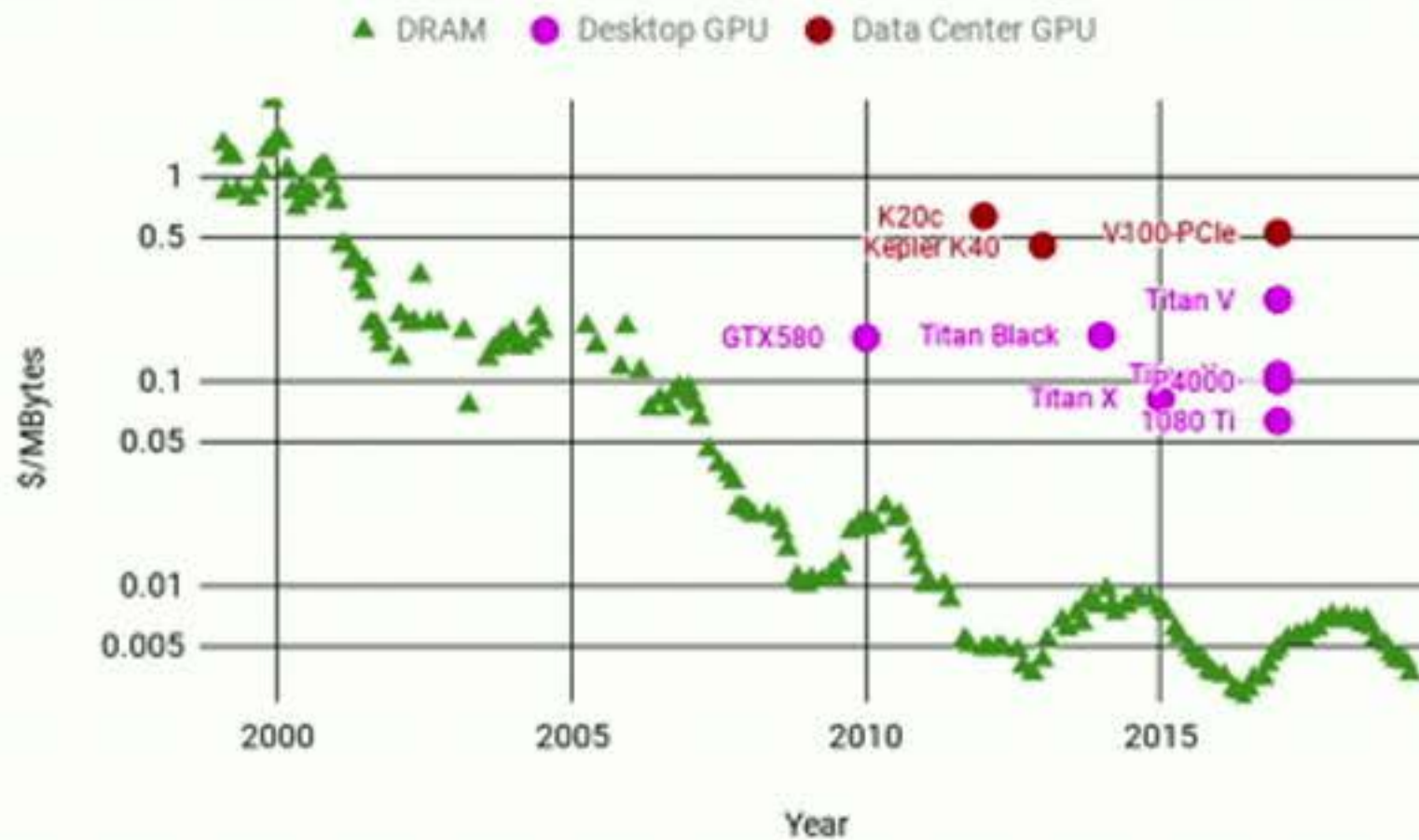
Larger Models Lead To Better Performance



GPU Memory Is Limited And Expensive



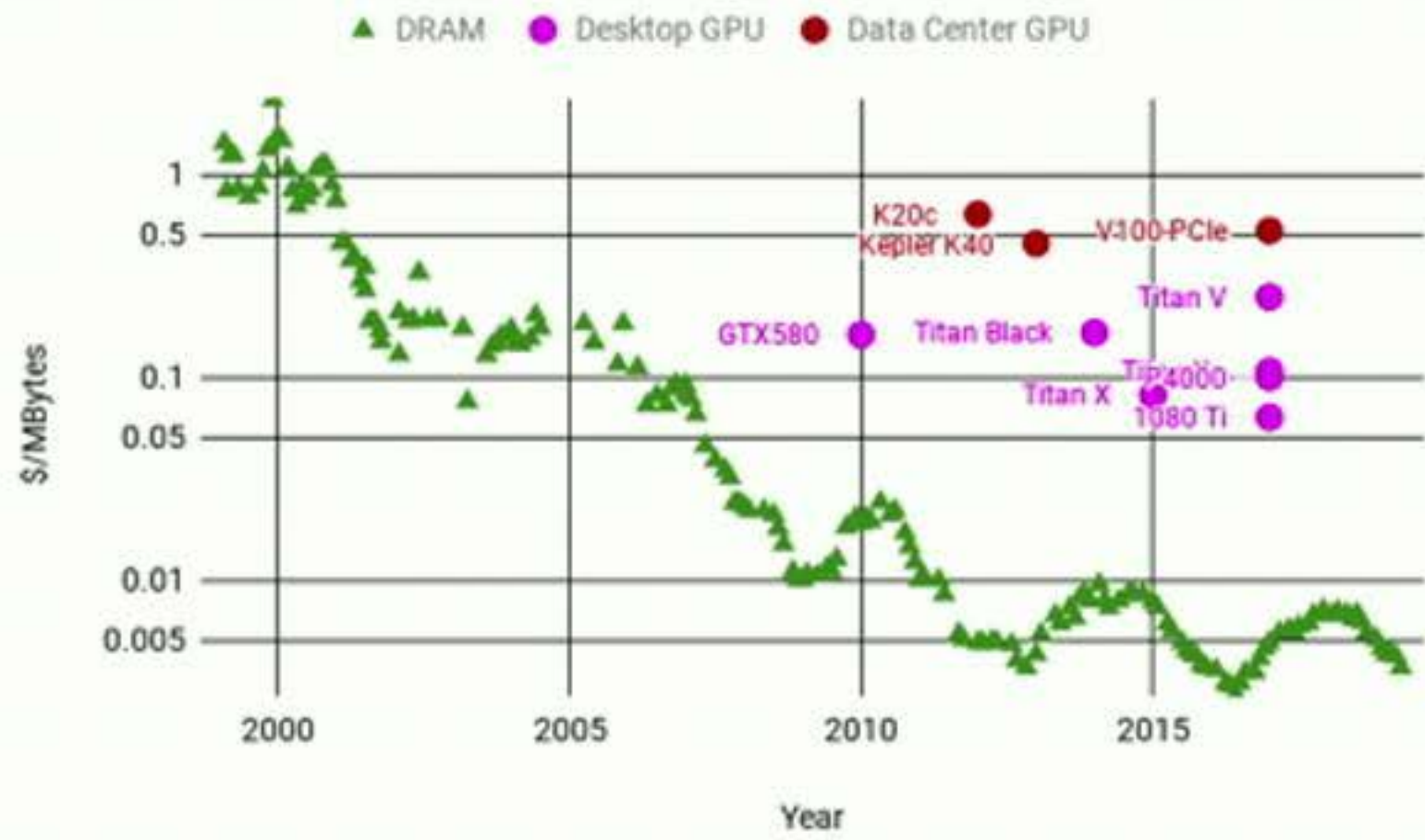
GPU Memory Is Limited And Expensive



Memory Capacity	Price
16GB	\$7399
32GB	\$8799

Nvidia V100 (PCIe) GPU Price
 Source: thinkmate.com
 2019/8/12

GPU Memory Is Limited And Expensive



Memory Capacity	Price
16GB	\$7399
32GB	\$8799

Nvidia V100 (PCIe) GPU Price
 Source: thinkmate.com
 2019/8/12

\$0.085 per extra MB

Many Previous Works on Improving Memory Efficiency

Gradient checkpointing (leveraging recomputation)

Training Deep Nets with Sublinear Memory Cost [Chen et al., arXiv'16]

Memory-Efficient Backpropagation Through Time [Gruslys et al., arXiv'16]

...

Memory swapping (leveraging cheaper host memory)

Dynamic Control Flow in Large-Scale Machine Learning [Yu, EuroSys'19]

vDNN: virtualized deep neural networks for scalable, memory-efficient neural network design [Rhu et al., MICRO'16]

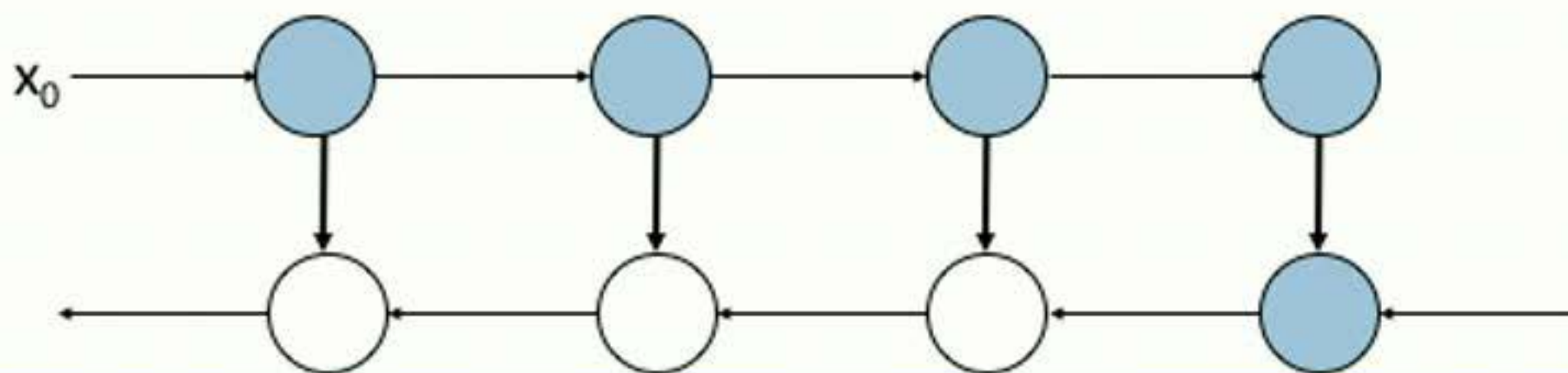
Training Deeper Models by GPU Memory Optimization on TensorFlow [Meng et al., MLSys'17]

Superneurons: dynamic GPU memory management for training deep neural networks [Wang et al., PPOPP'18]

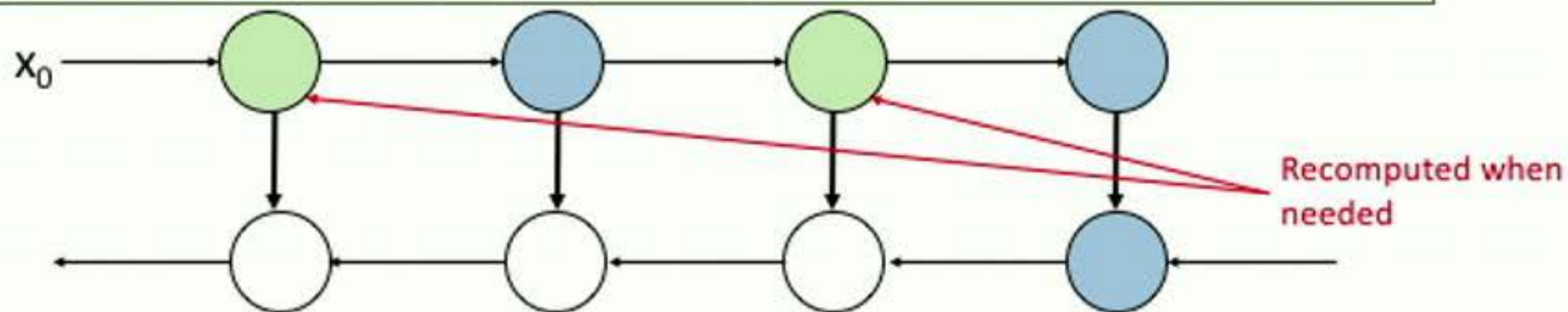
TensorFlow Grapper memory optimizer

Background: Gradient Checkpointing

Original computation graph for backpropagation, $O(N)$ memory cost

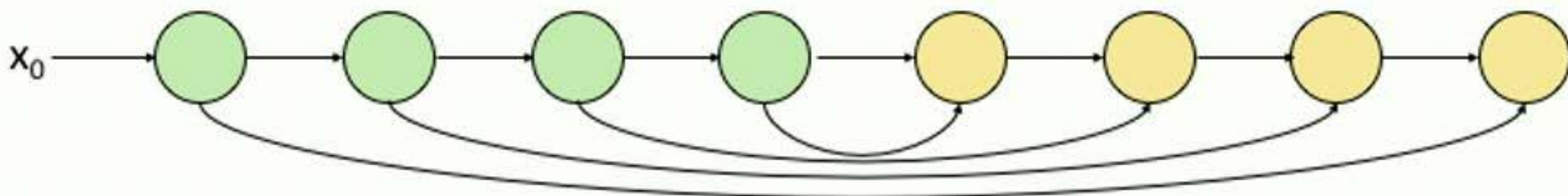


With gradient checkpointing, $O(\sqrt{N})$ memory cost

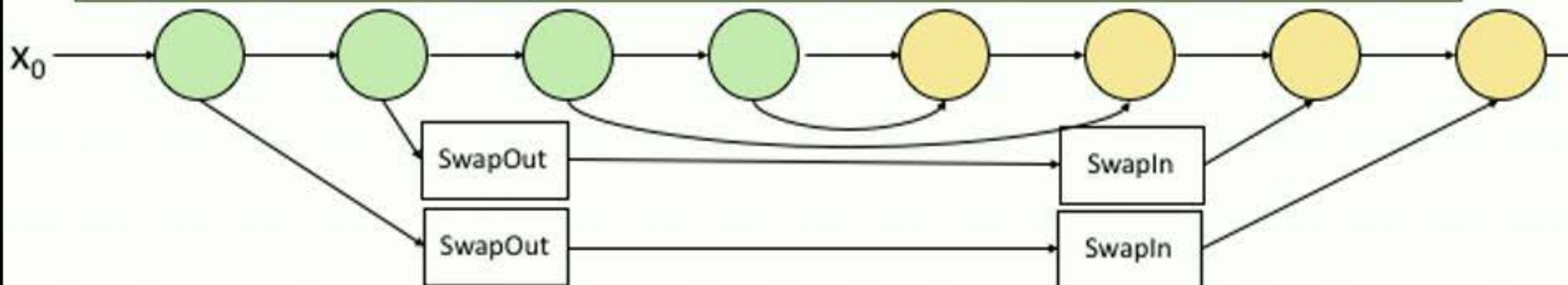


Background: Memory Swapping

Original computation graph for backpropagation, $O(M)$ memory cost



With memory swapping, $O(1)$ memory cost



They Work Well for Linear Graphs

Most nodes are “graph separator nodes”: removing each one separates the graph into two disjoint subgraphs

Gradient checkpointing: easy to determine which nodes to checkpoint.

Limited freedom regarding scheduling

Memory swapping: easy to determine what and when to swap

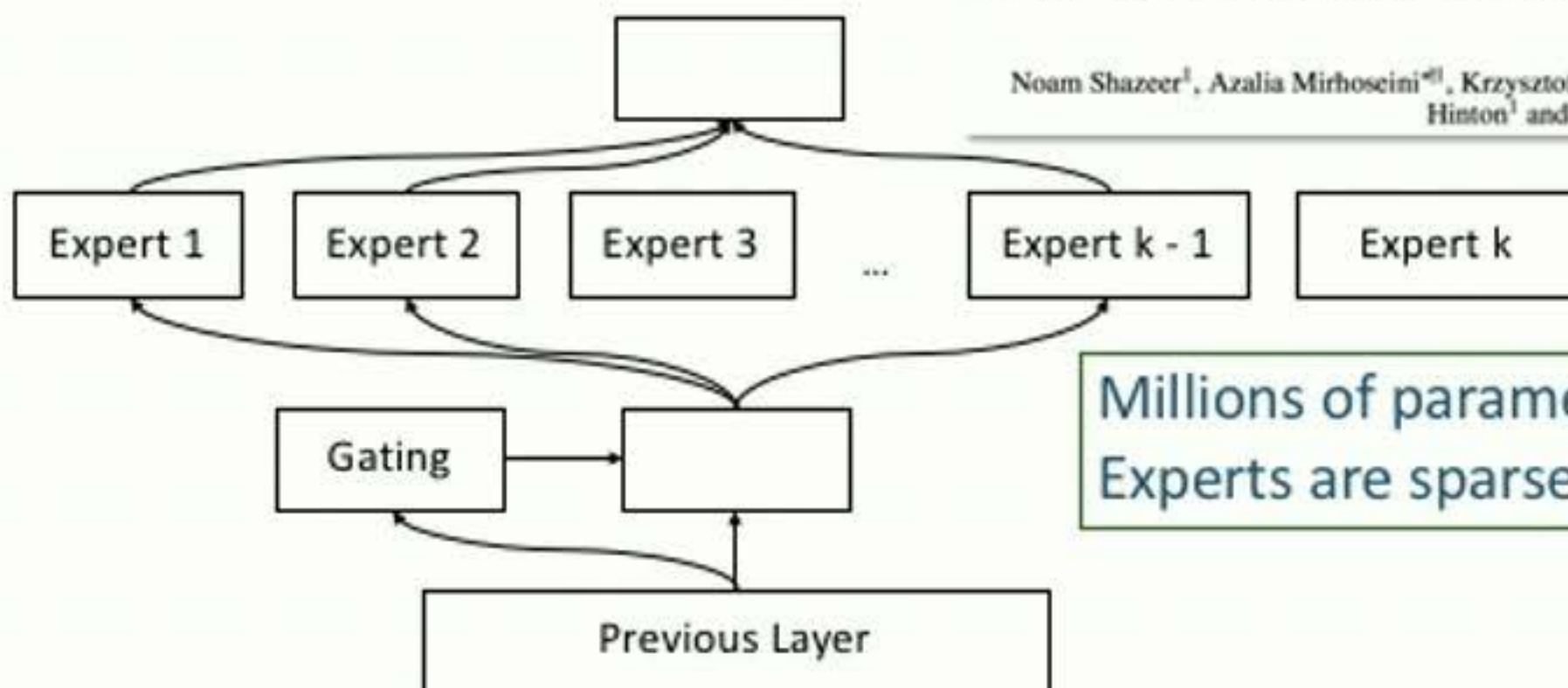
Problem: many neural network graphs are not linear!

Some layers have an excessive amount of parallelism.

Emerging Non-linear Neural Networks

OUTRAGEOUSLY LARGE NEURAL NETWORKS: THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

Noam Shazeer¹, Azalia Mirhoseini^{†1}, Krzysztof Maziarz^{*2}, Andy Davis¹, Quoc Le¹, Geoffrey Hinton¹ and Jeff Dean¹



Millions of parameters per expert.
Experts are sparsely activated.

Goal: General Memory-Efficient DL On TensorFlow



Linear and nonlinear computation graphs

Implement and evaluate on TensorFlow

Transparent to applications.

Existing memory optimizations for TensorFlow:

Gradient checkpointing (Bolotov et al., GitHub'17):

Limited to linear graphs; requires non-trivial changes to application program

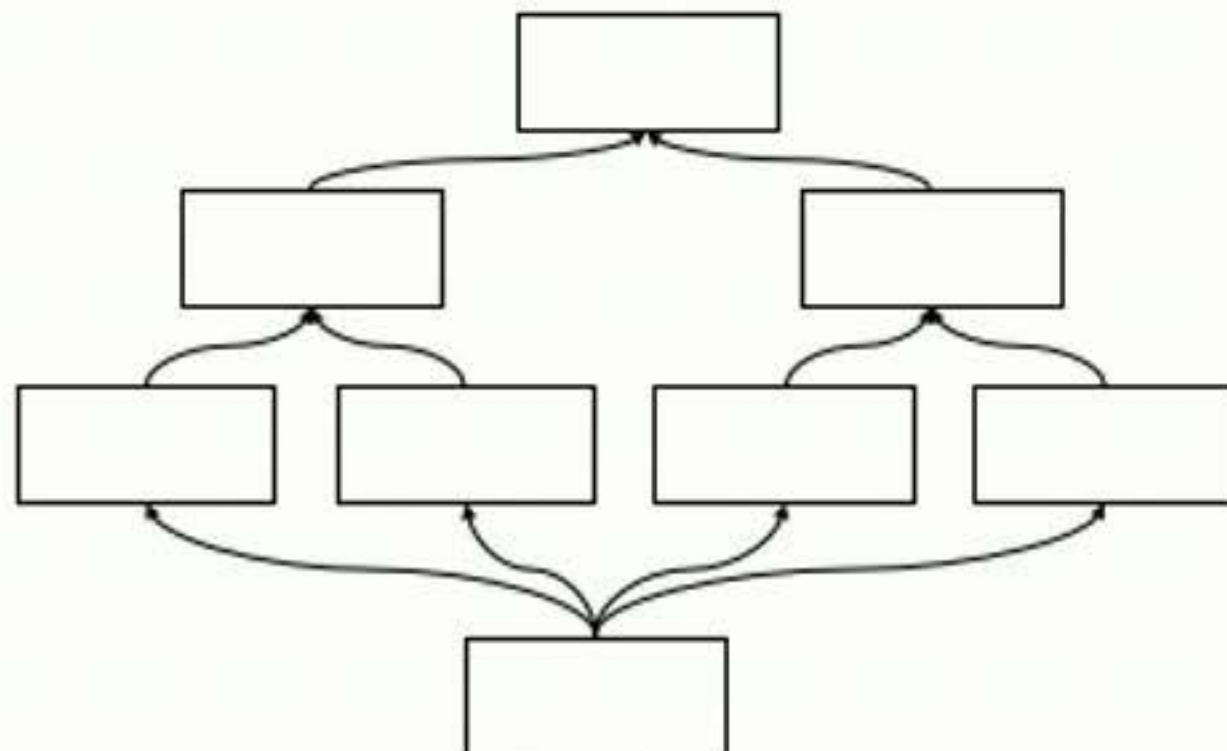
Grappler memory swapping pass:

Limited to linear graphs

WhileLoop memory swapping ([Yuan et al., EuroSys'18]):

Operation specific memory reduction

Idea #1: Limit Memory Consumption by Limiting Parallelism



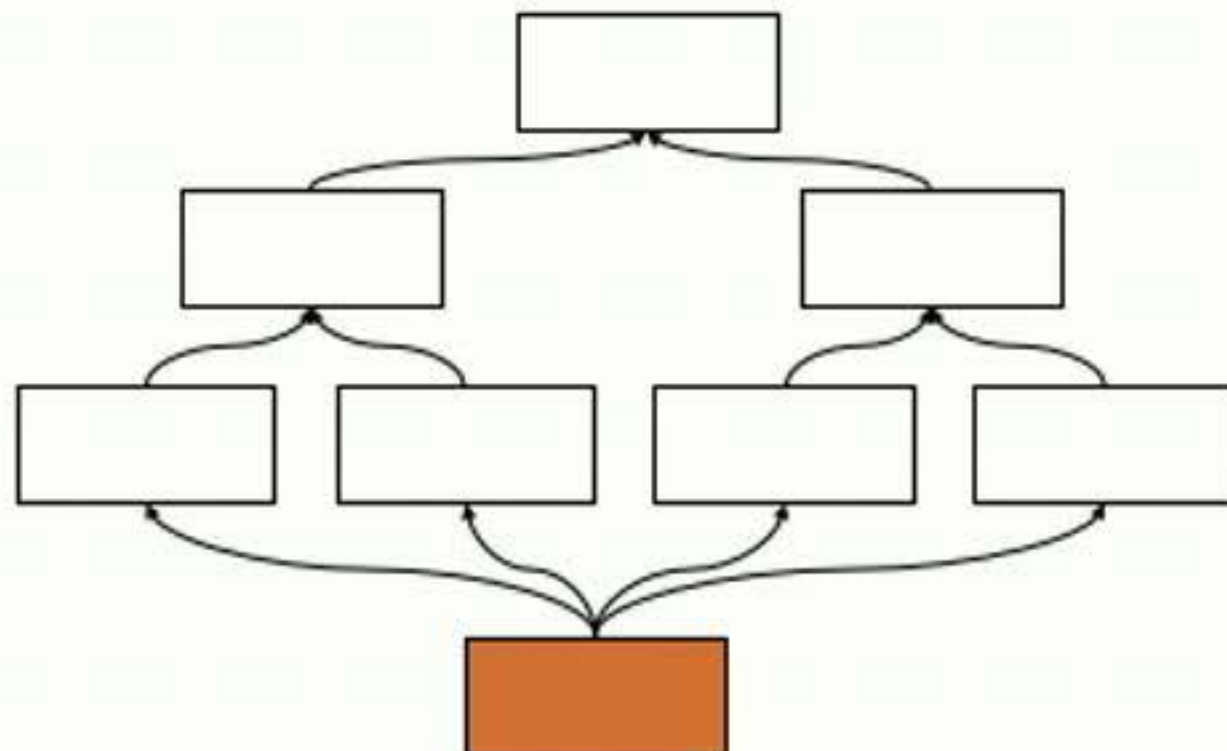
TensorFlow

Breath-first traversal

Max. parallelism

Max. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



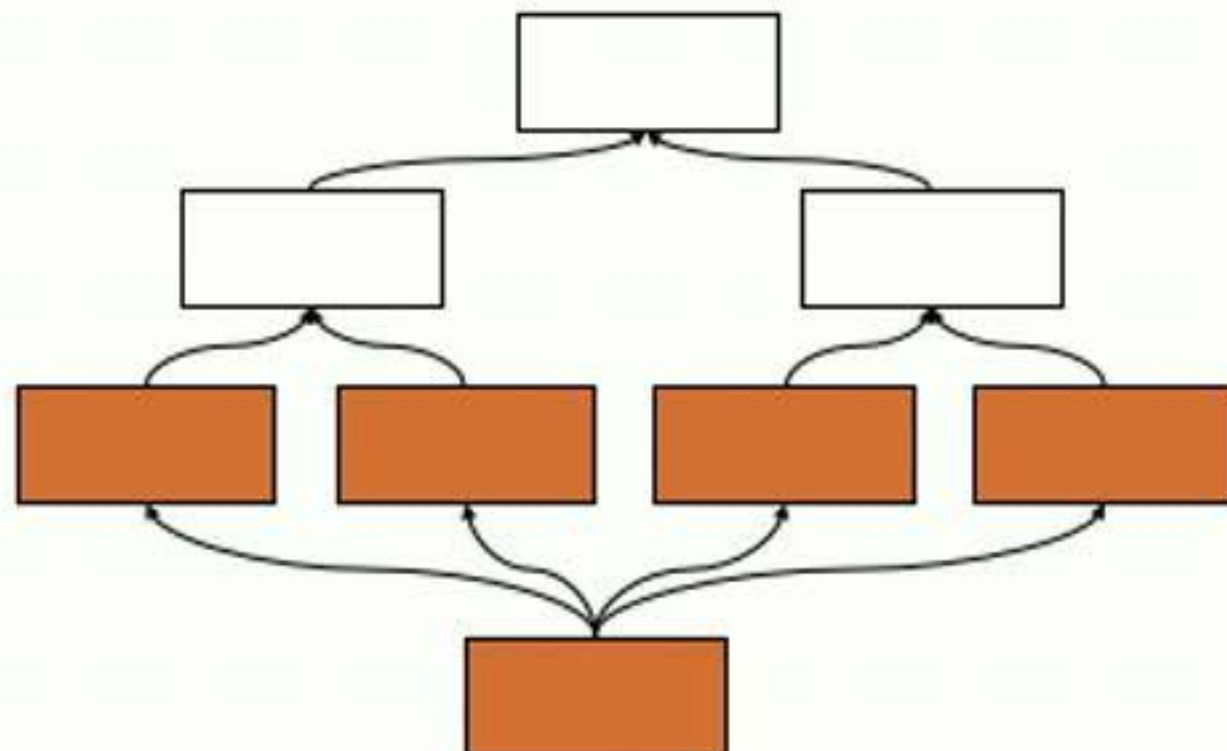
TensorFlow

Breath-first traversal

Max. parallelism

Max. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



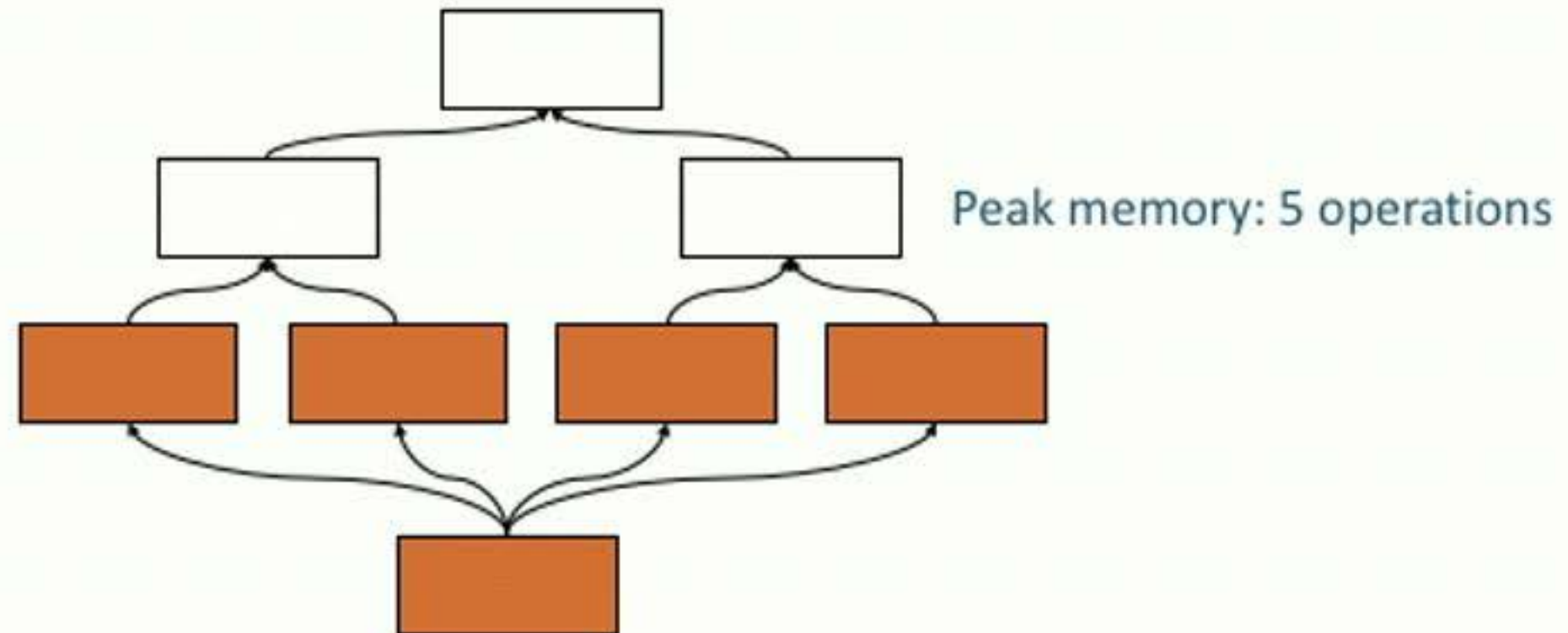
TensorFlow

Breath-first traversal

Max. parallelism

Max. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



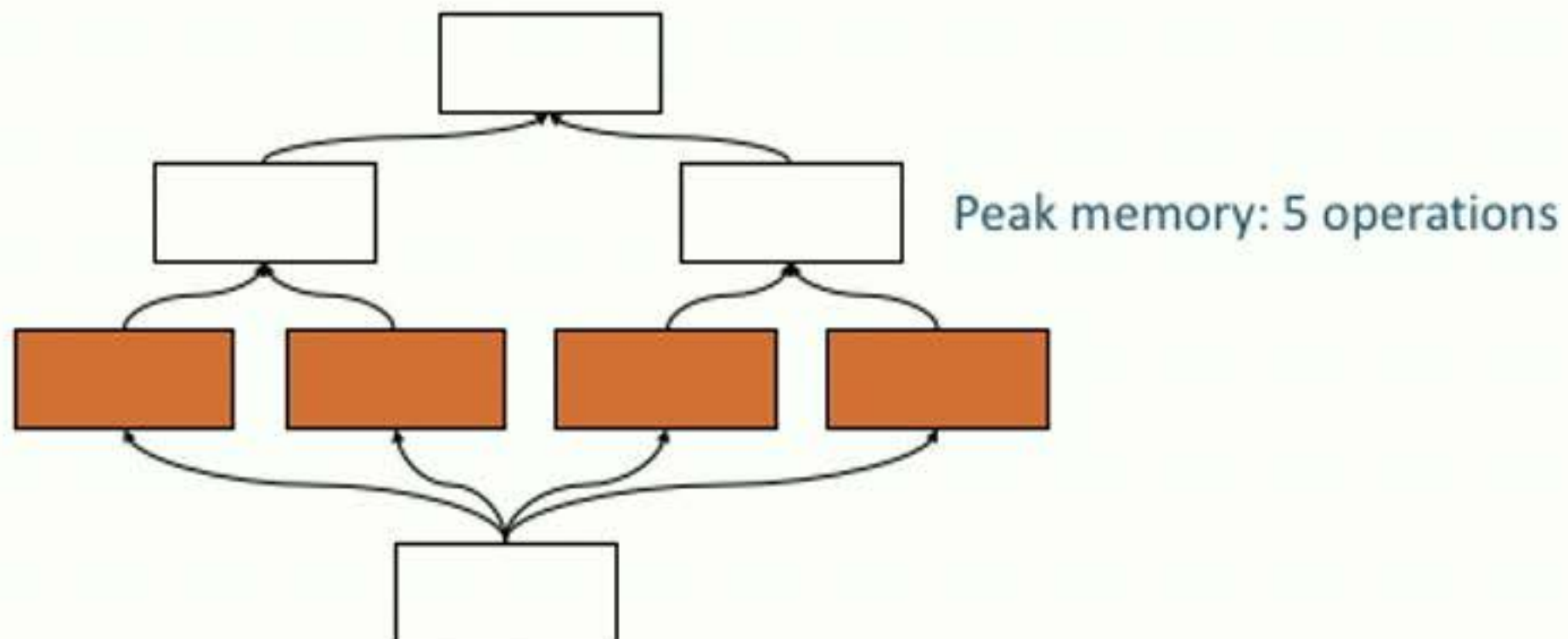
TensorFlow

Breath-first traversal

Max. parallelism

Max. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



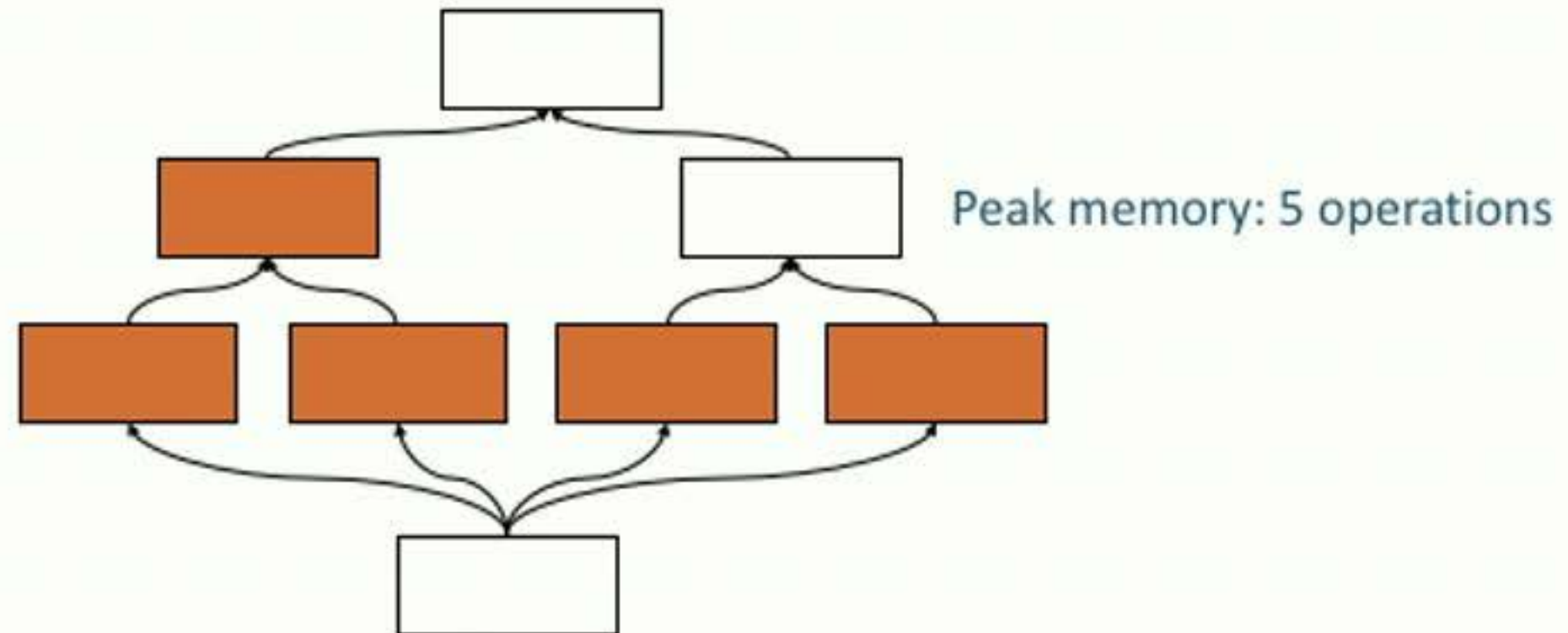
TensorFlow

Breath-first traversal

Max. parallelism

Max. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



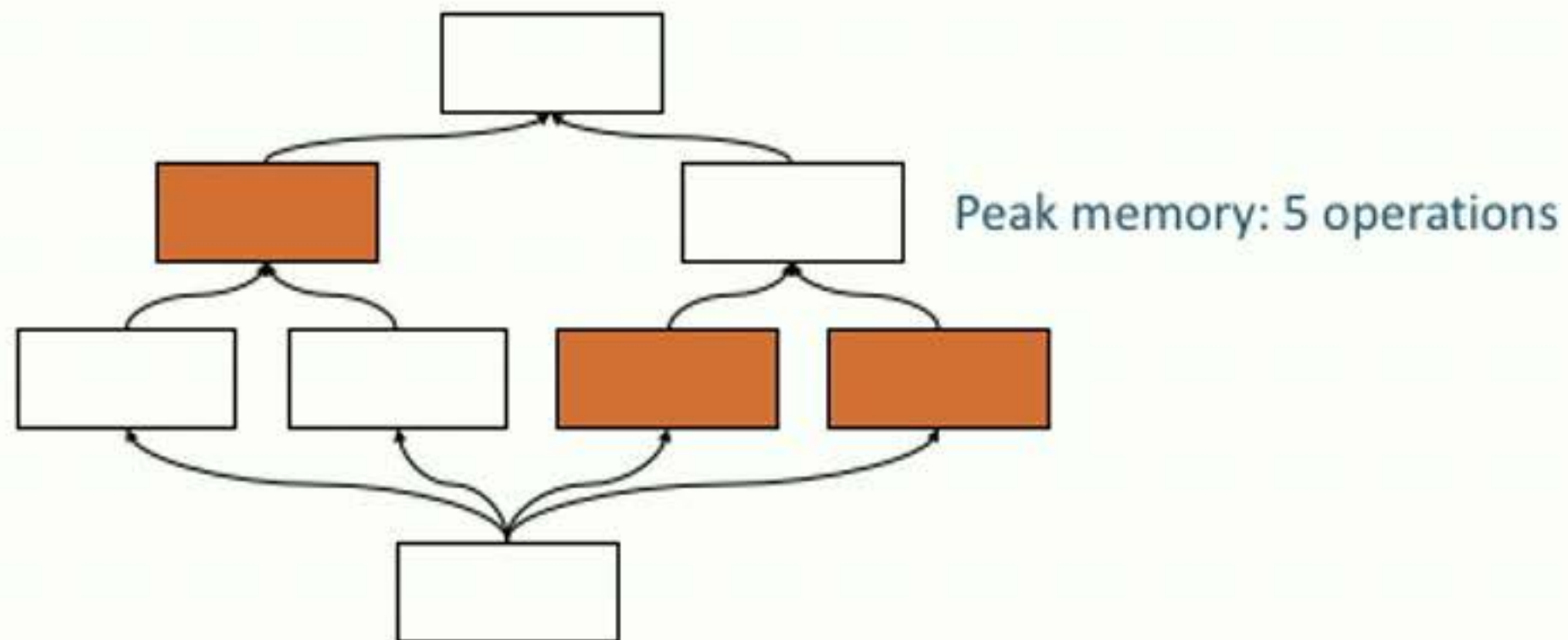
TensorFlow

Breath-first traversal

Max. parallelism

Max. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



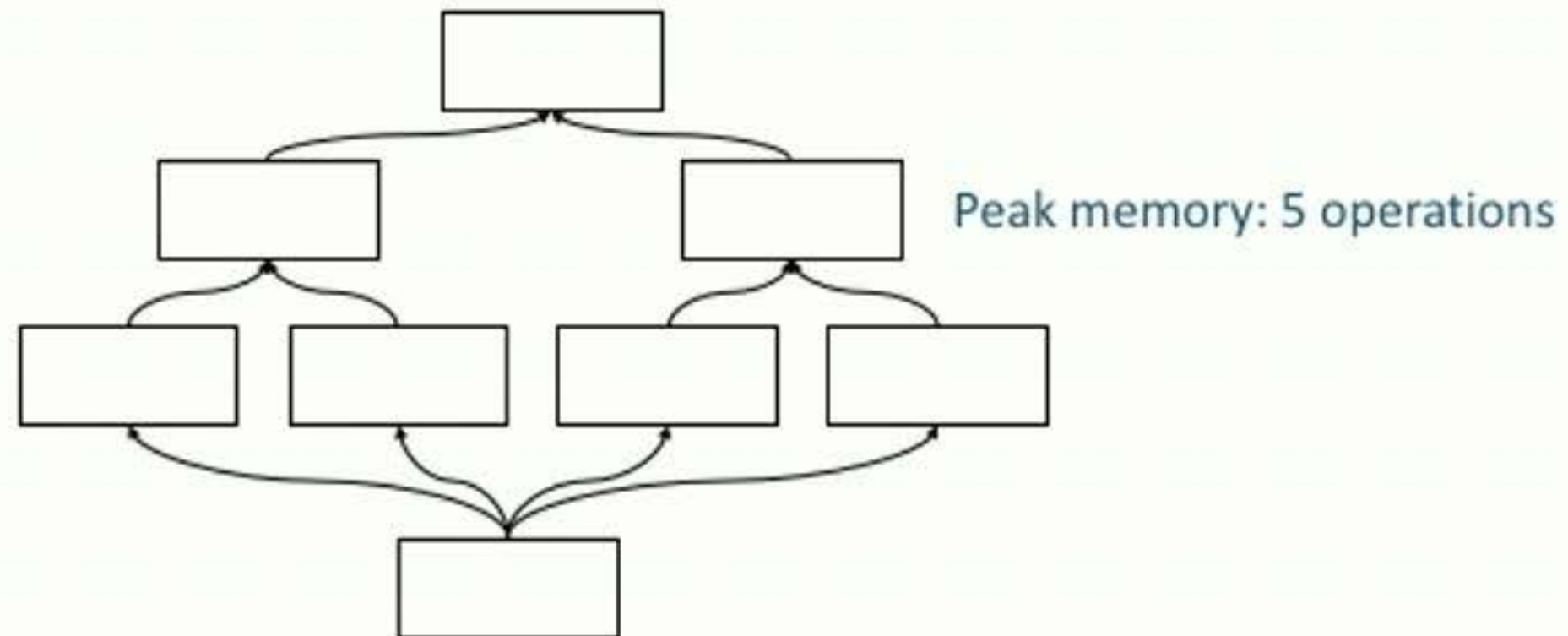
TensorFlow

Breath-first traversal

Max. parallelism

Max. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



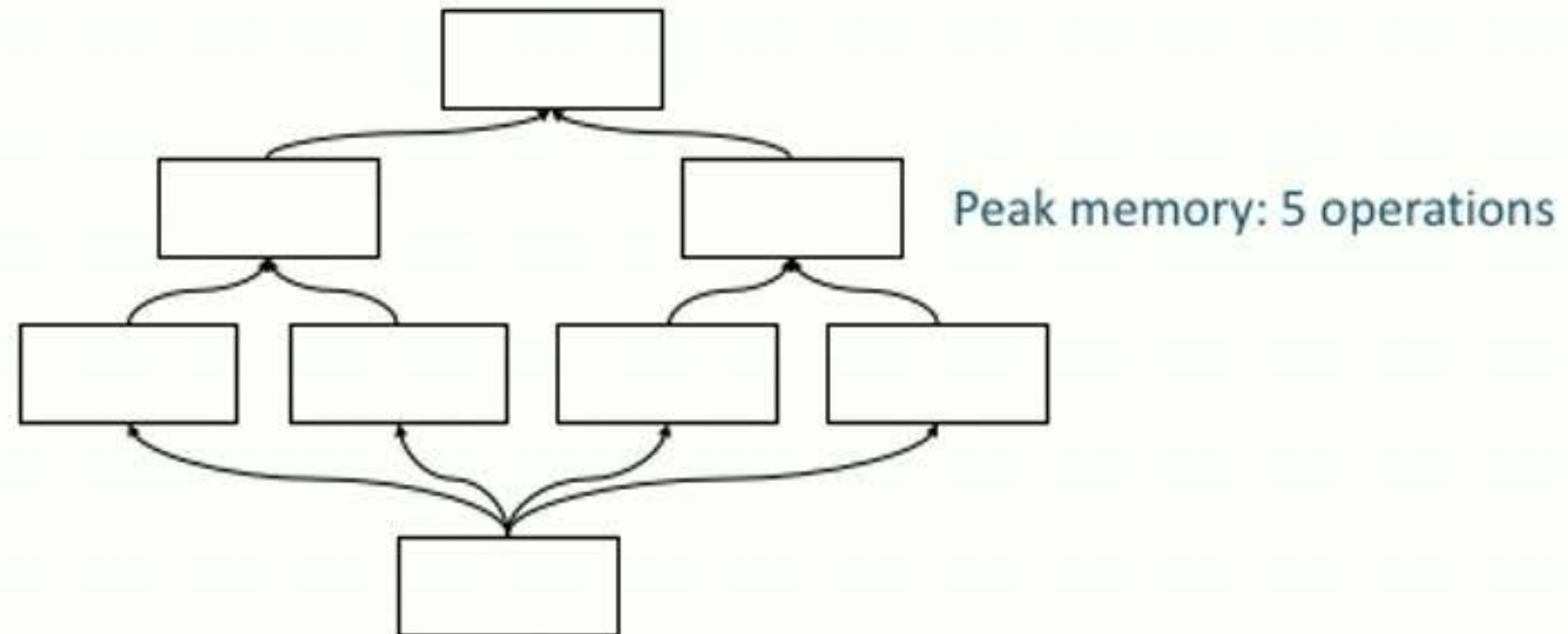
TensorFlow

Breath-first traversal

Max. parallelism

Max. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



TensorFlow

Breath-first traversal

Max. parallelism

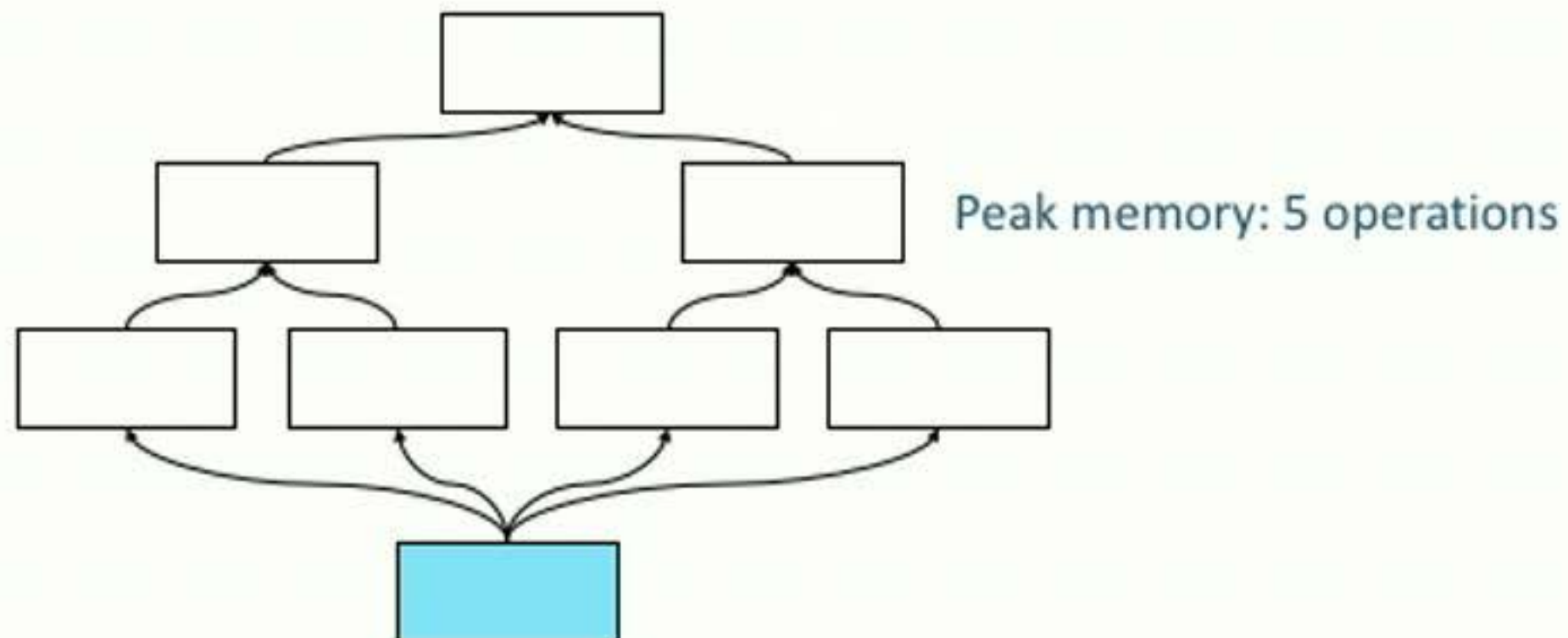
Max. memory

Linearize the graph

No parallelism

Min. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



TensorFlow

Breath-first traversal

Max. parallelism

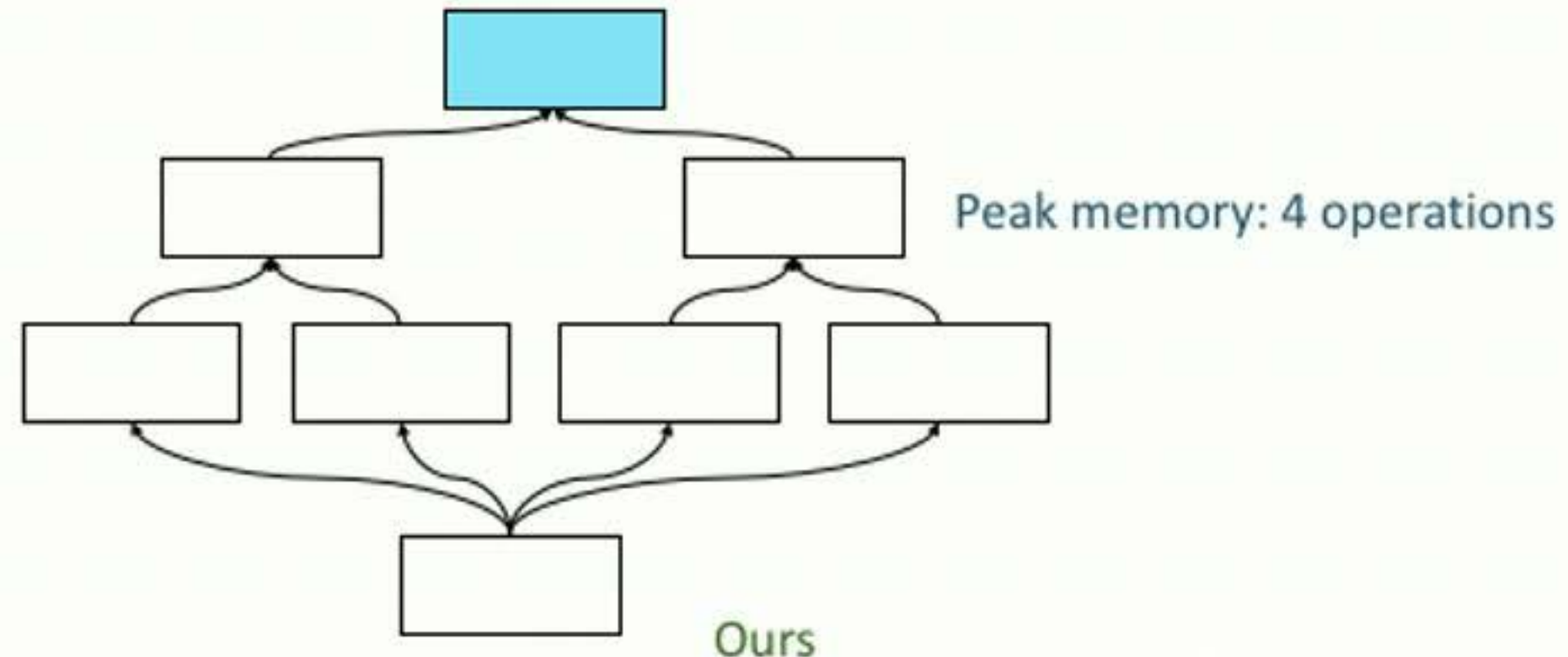
Max. memory

Linearize the graph

No parallelism

Min. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



TensorFlow

Breath-first traversal

Max. parallelism
Max. memory

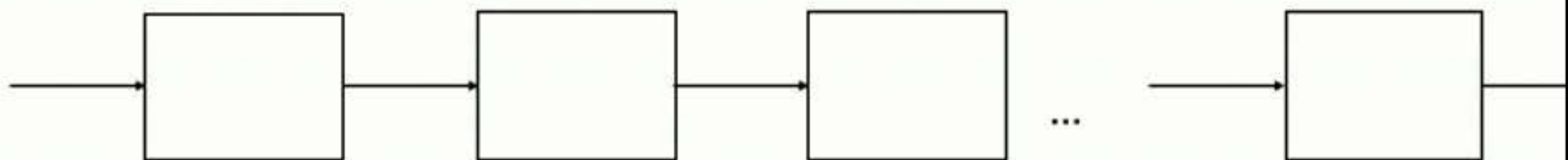
Partition the graph,
Linearize among partitions
Parallelism with partitions

Control parallelism vs. memory

Linearize the graph

No parallelism
Min. memory

Idea #2: Offload GPU Tensors To Host Memory



Transformer

Use MoE as the Feed Forward layer

12 MoEs

32 experts per MoE

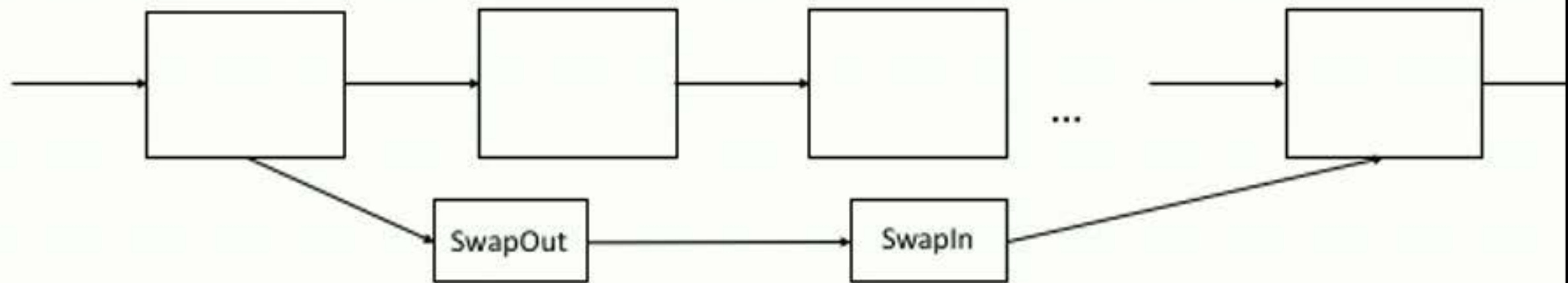
2M params per expert

~800M parameters total

Peak memory:

9.5GB to 6.8GB

Idea #2: Offload GPU Tensors To Host Memory



Transformer

Use MoE as the Feed Forward layer

12 MoEs

32 experts per MoE

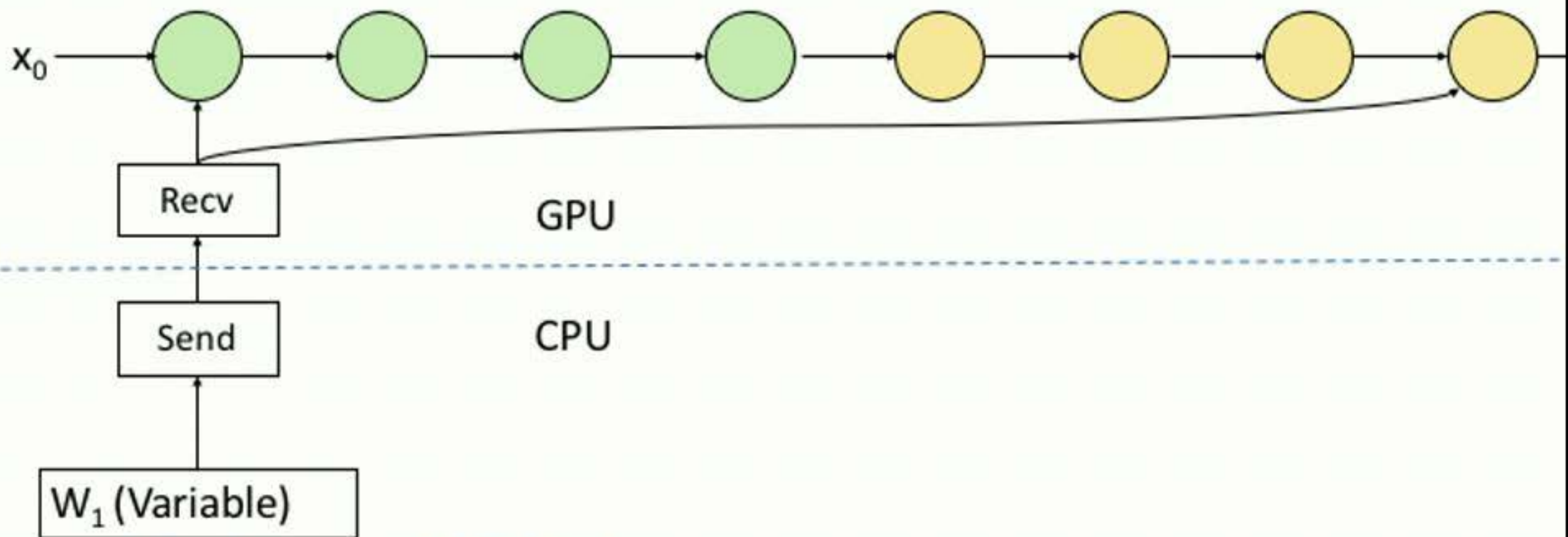
2M params per expert

~800M parameters total

Peak memory:

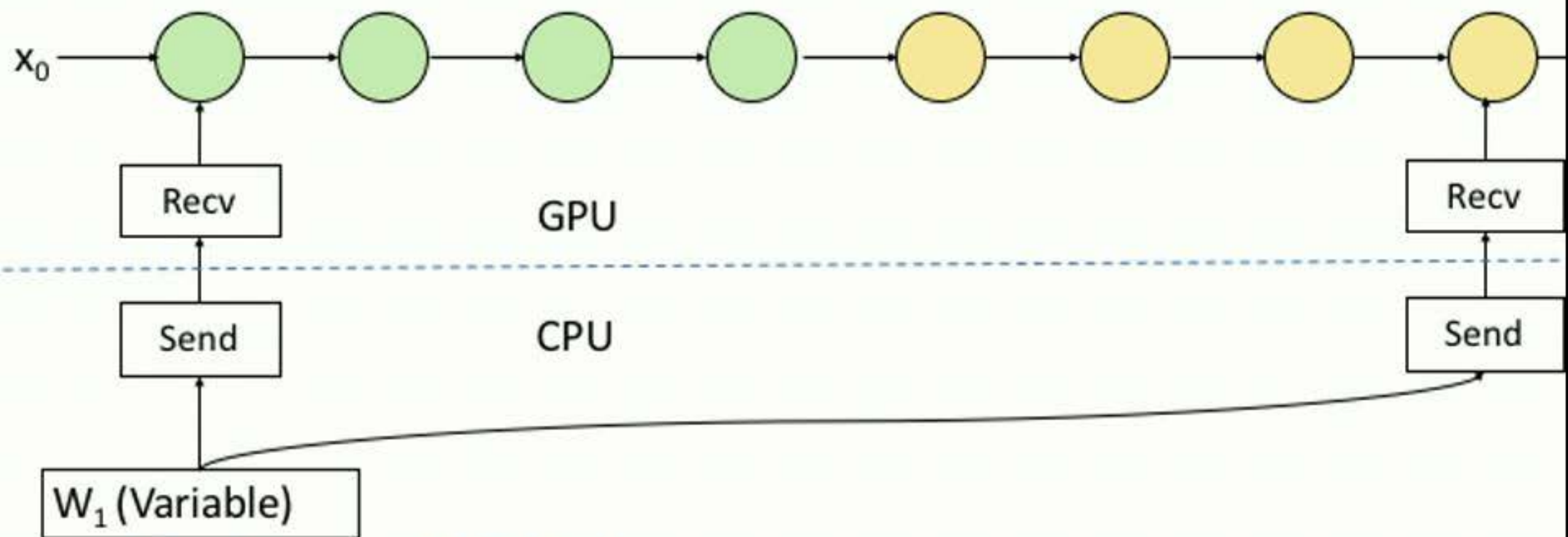
9.5GB to 6.8GB

Idea #3: Place Persistent Tensors on Host Memory & Send To GPU Only When Needed



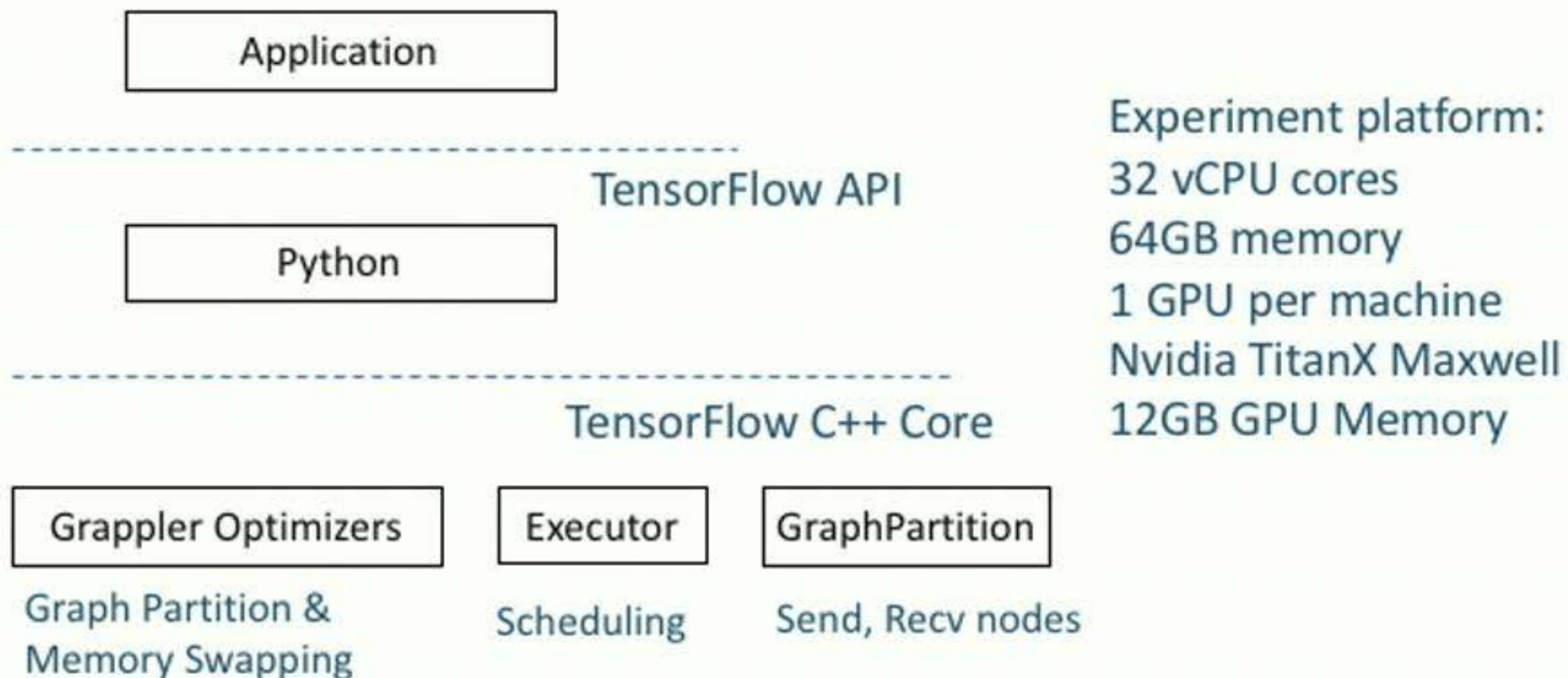
Transformer w/ MoE Peak memory:
6.8GB to 3.3GB

Idea #3: Place Persistent Tensors on Host Memory & Send To GPU Only When Needed

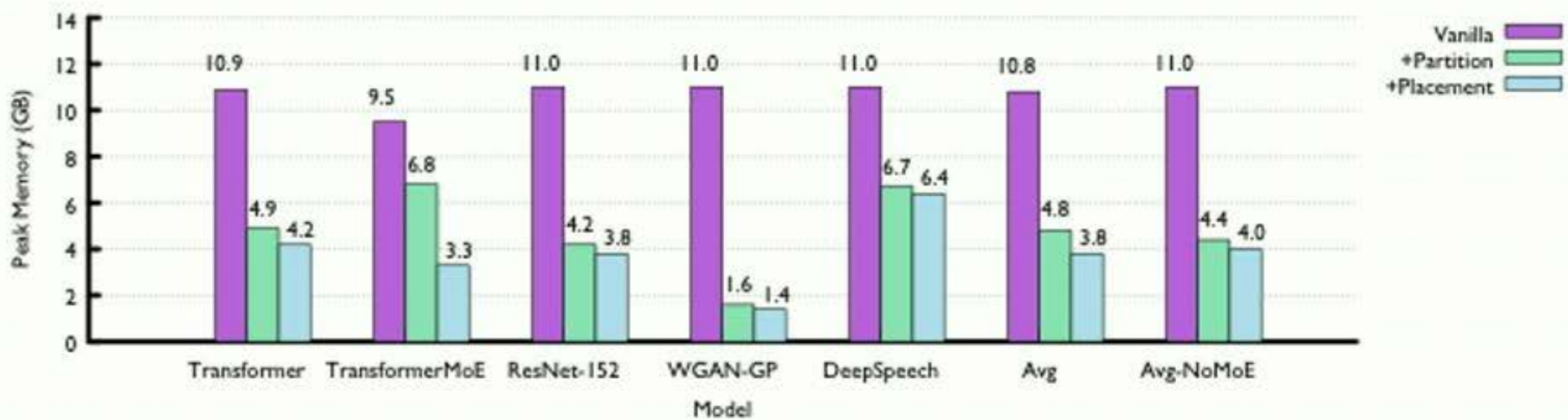


Transformer w/ MoE Peak memory:
6.8GB to 3.3GB

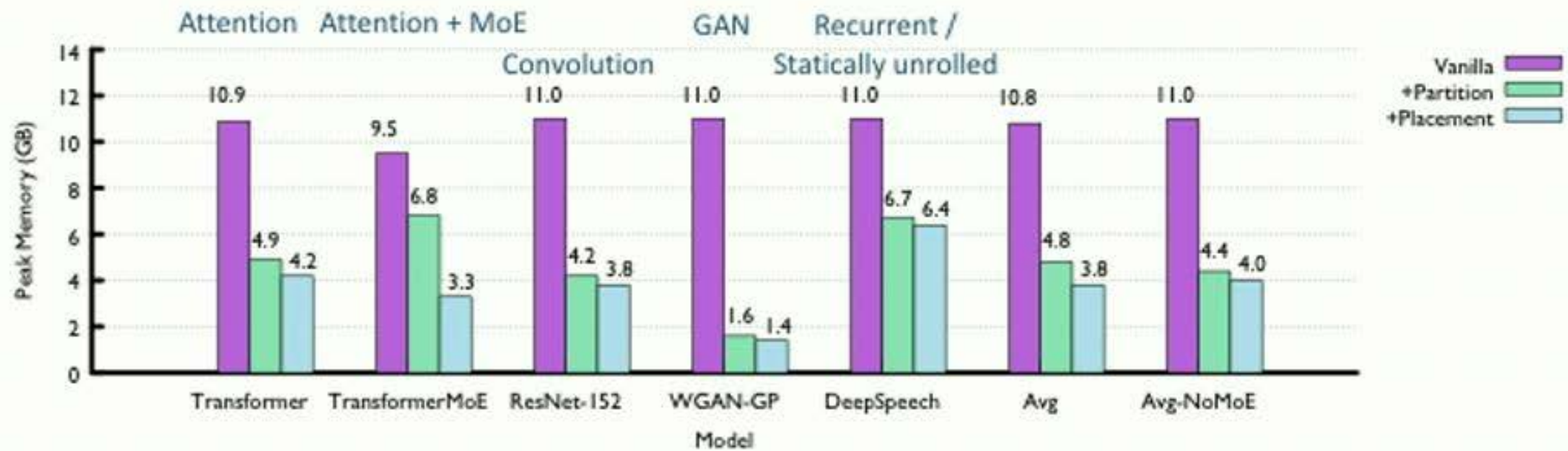
Implementation & Experiment Setup



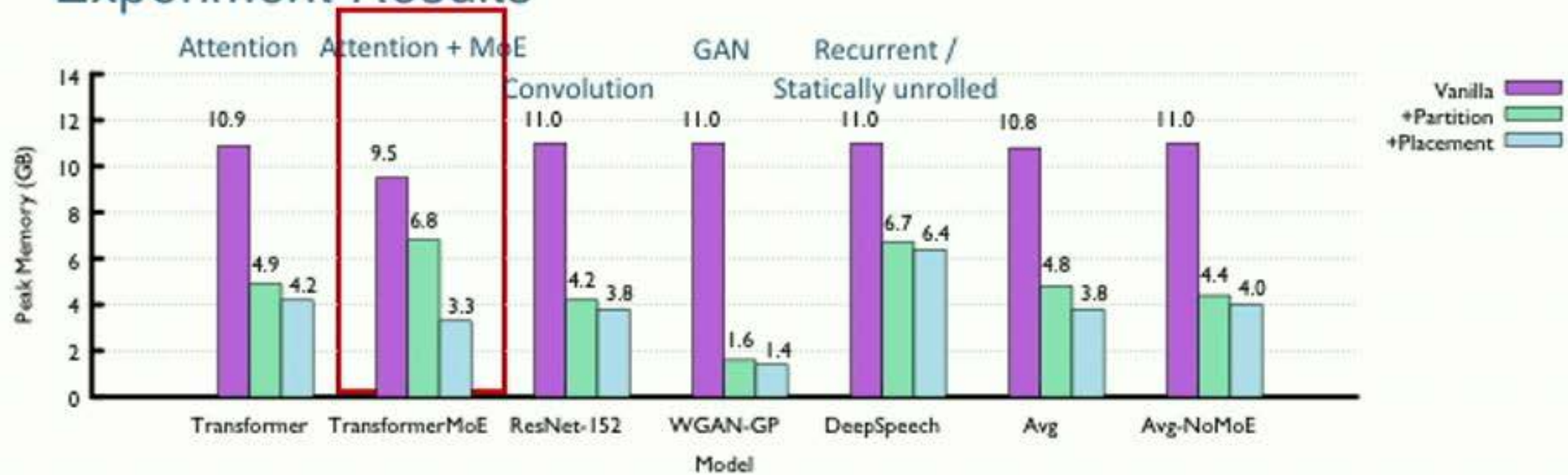
Experiment Results



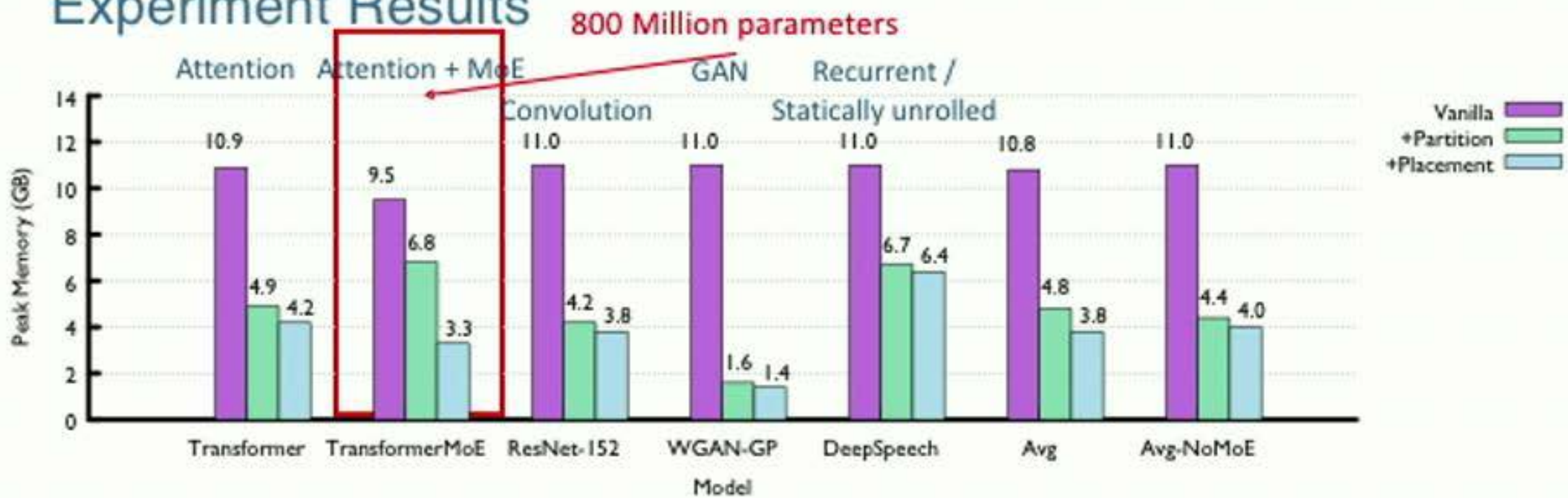
Experiment Results



Experiment Results

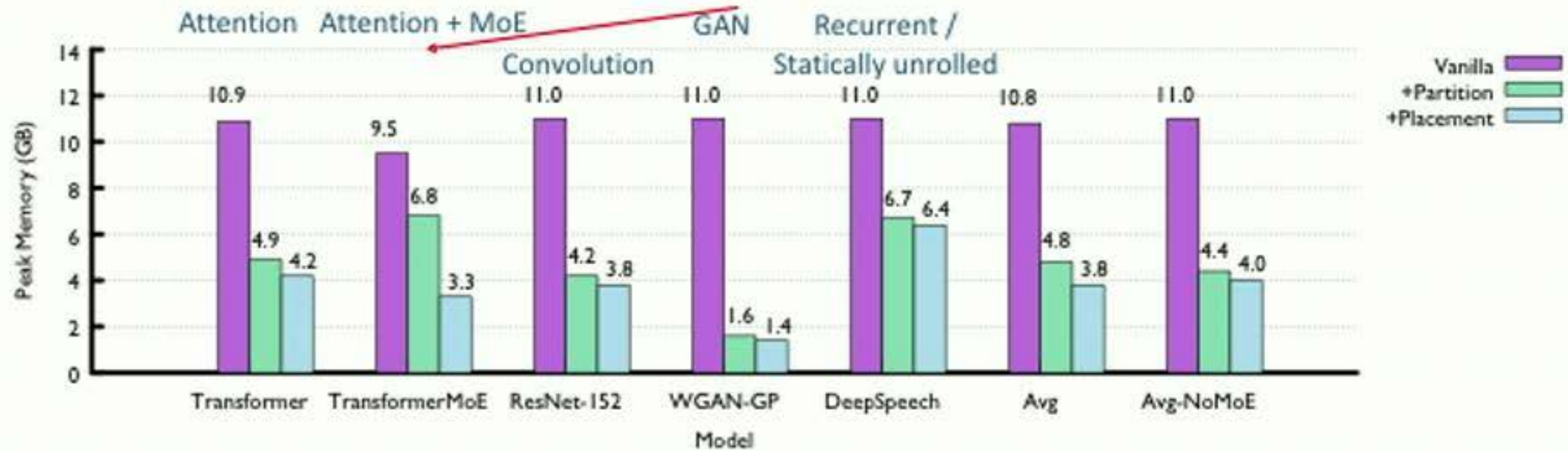


Experiment Results



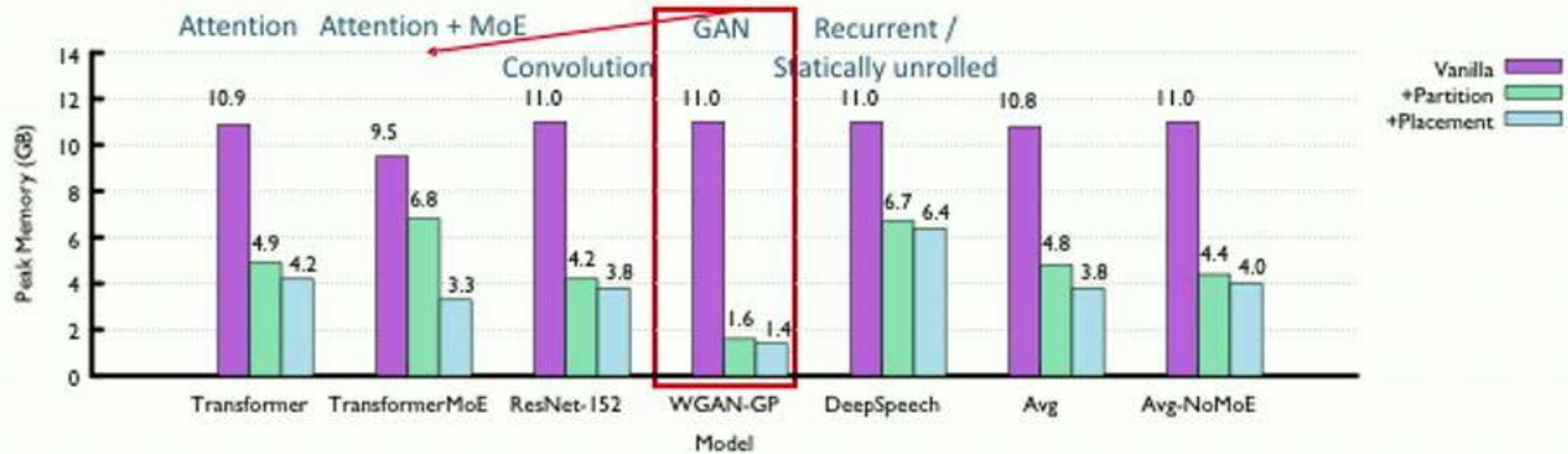
Experiment Results

800 Million parameters



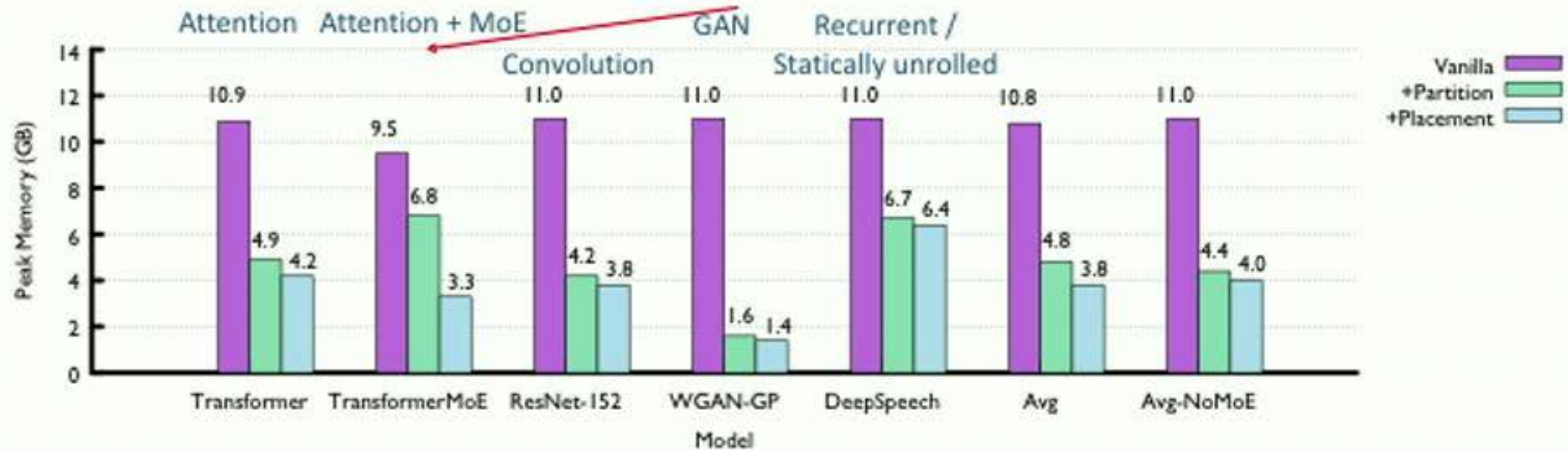
Experiment Results

800 Million parameters



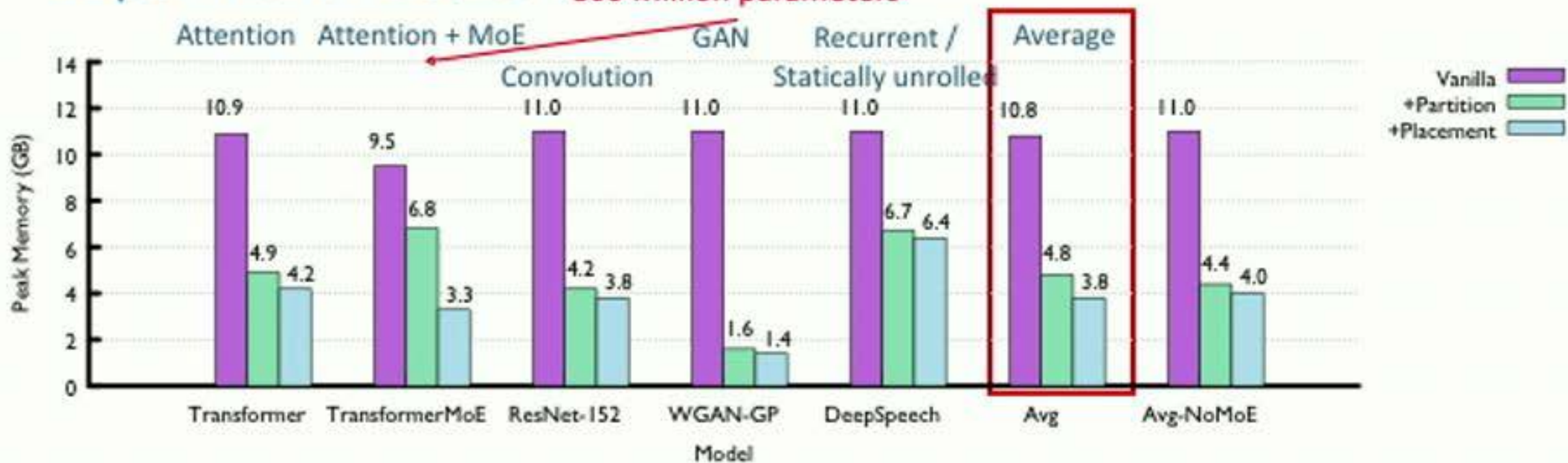
Experiment Results

800 Million parameters



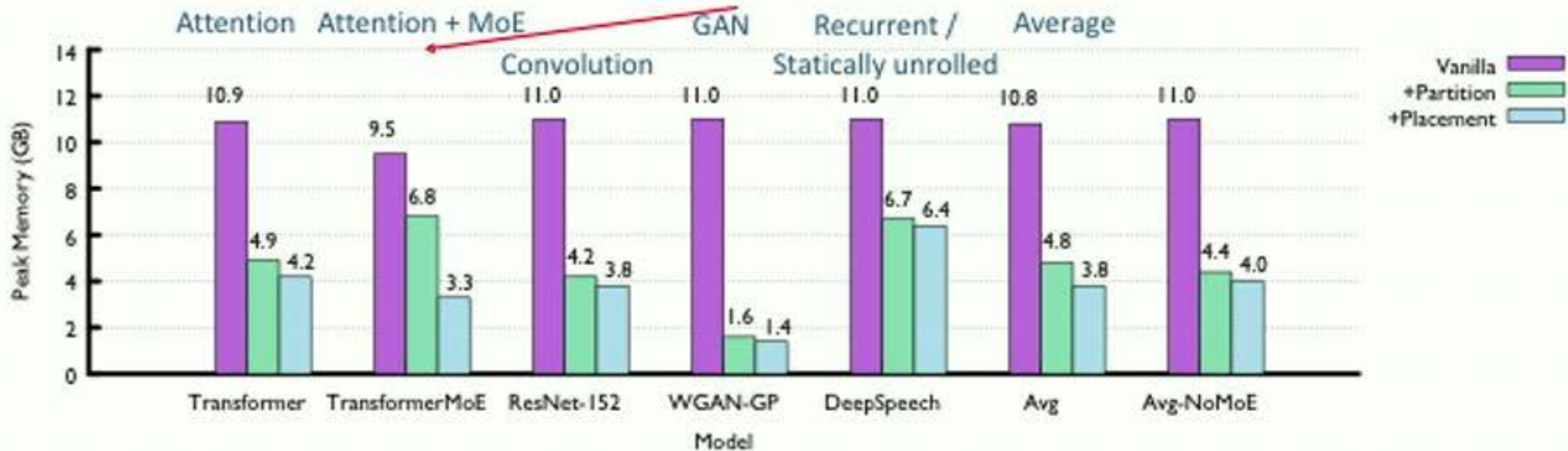
Experiment Results

800 Million parameters



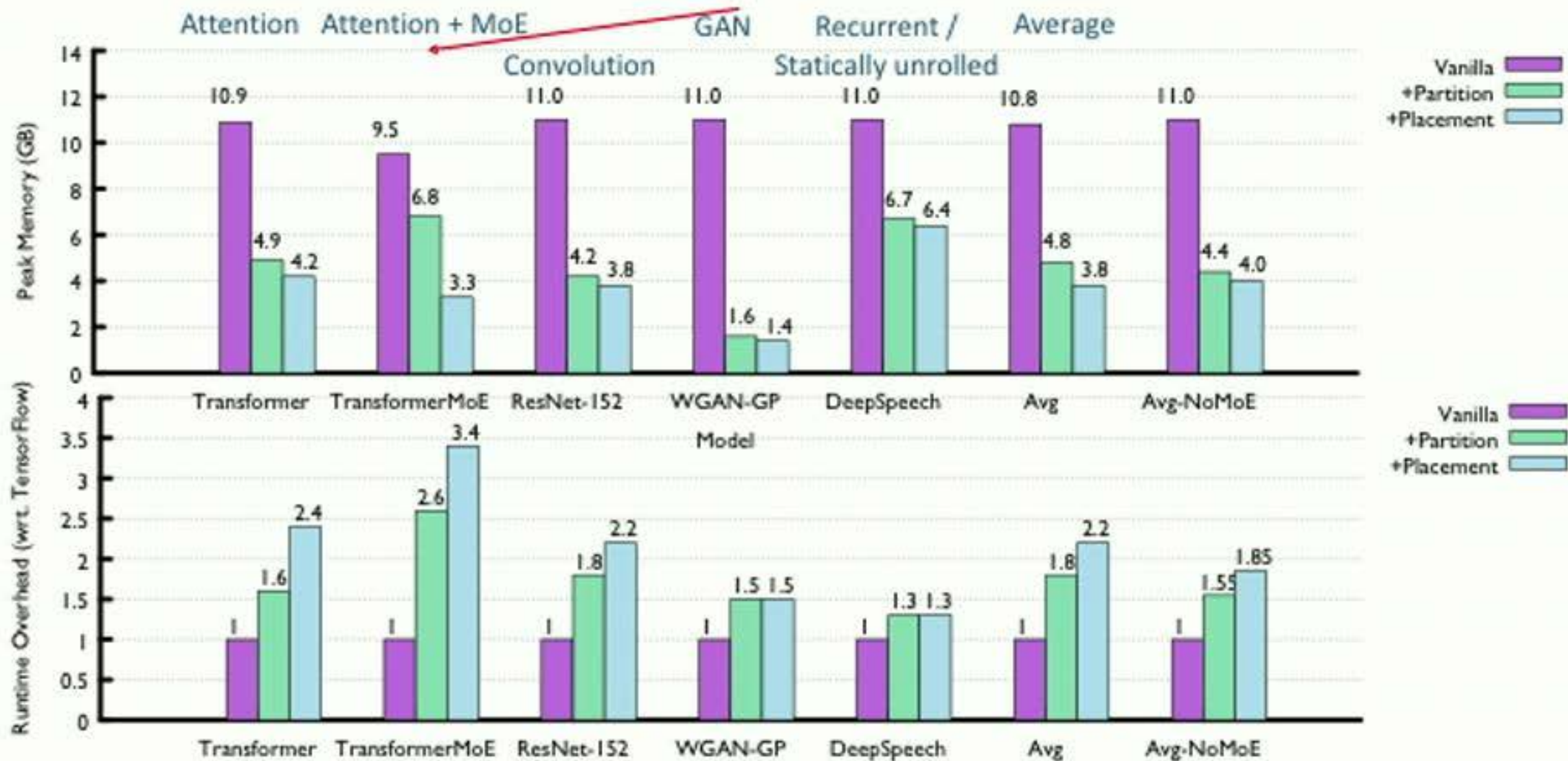
Experiment Results

800 Million parameters



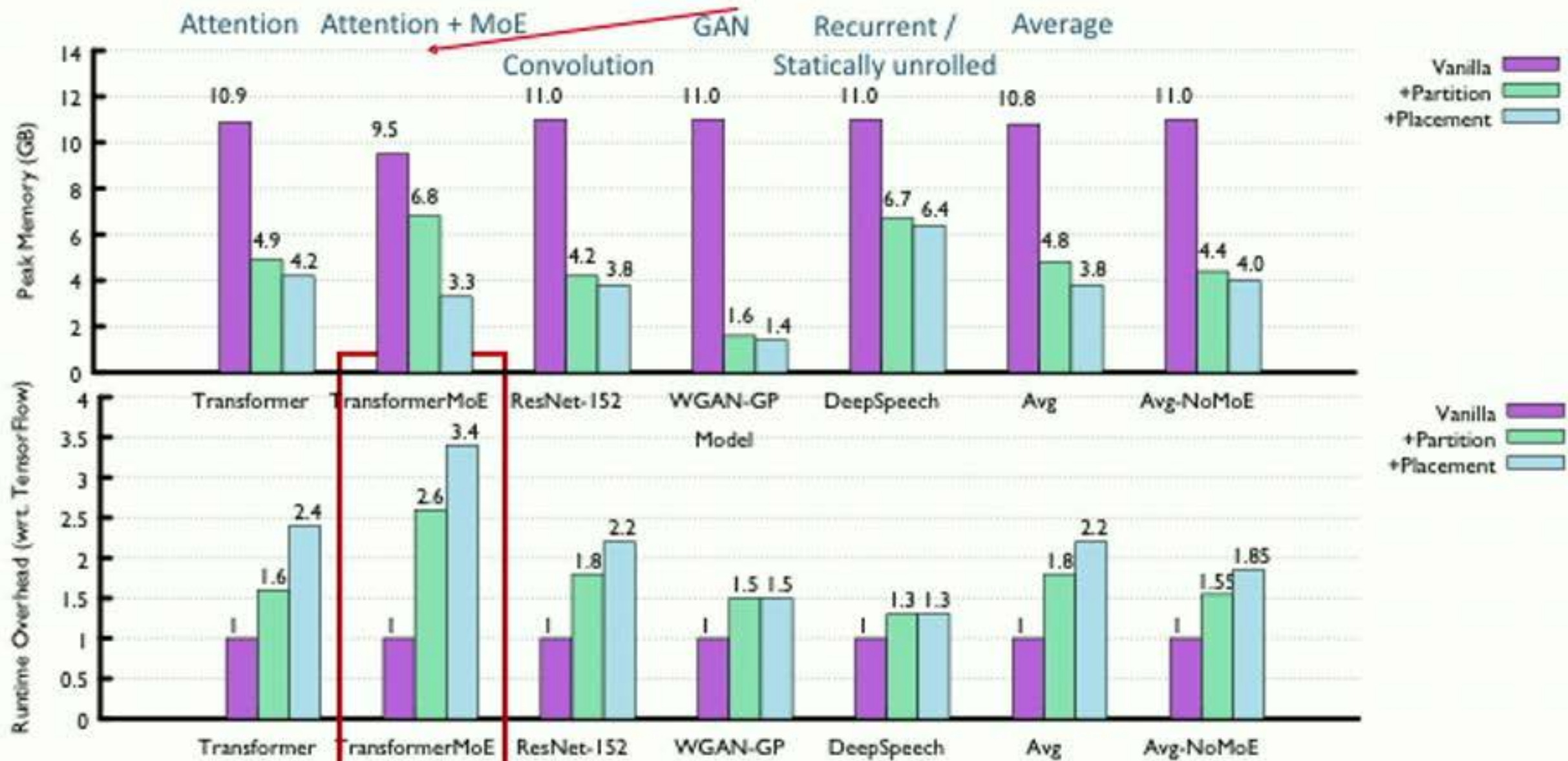
Experiment Results

800 Million parameters



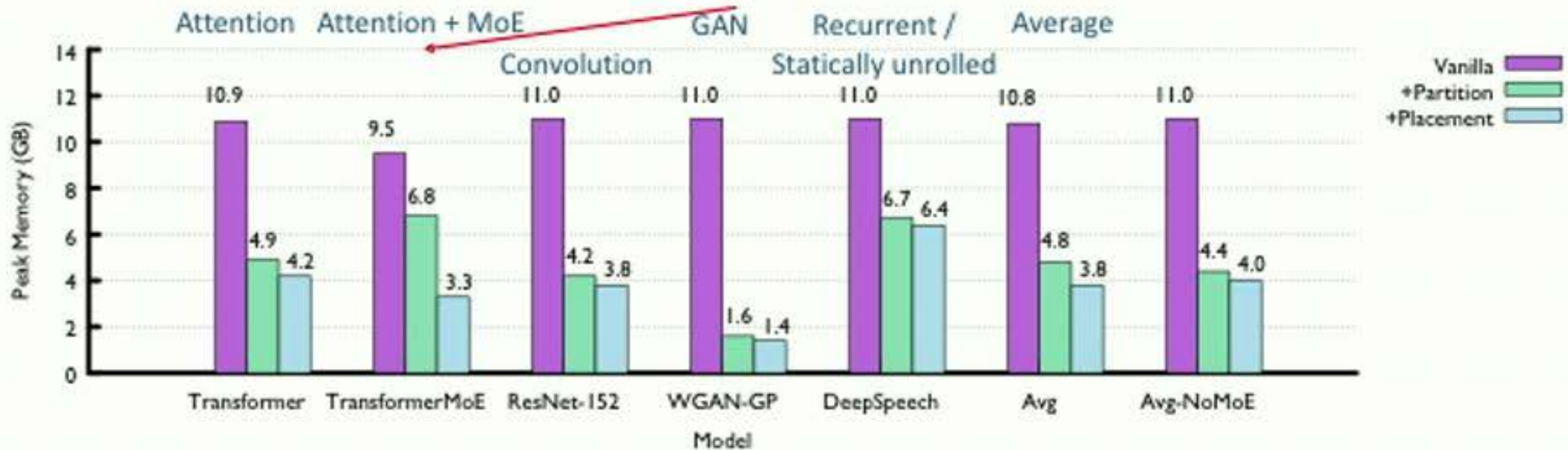
Experiment Results

800 Million parameters



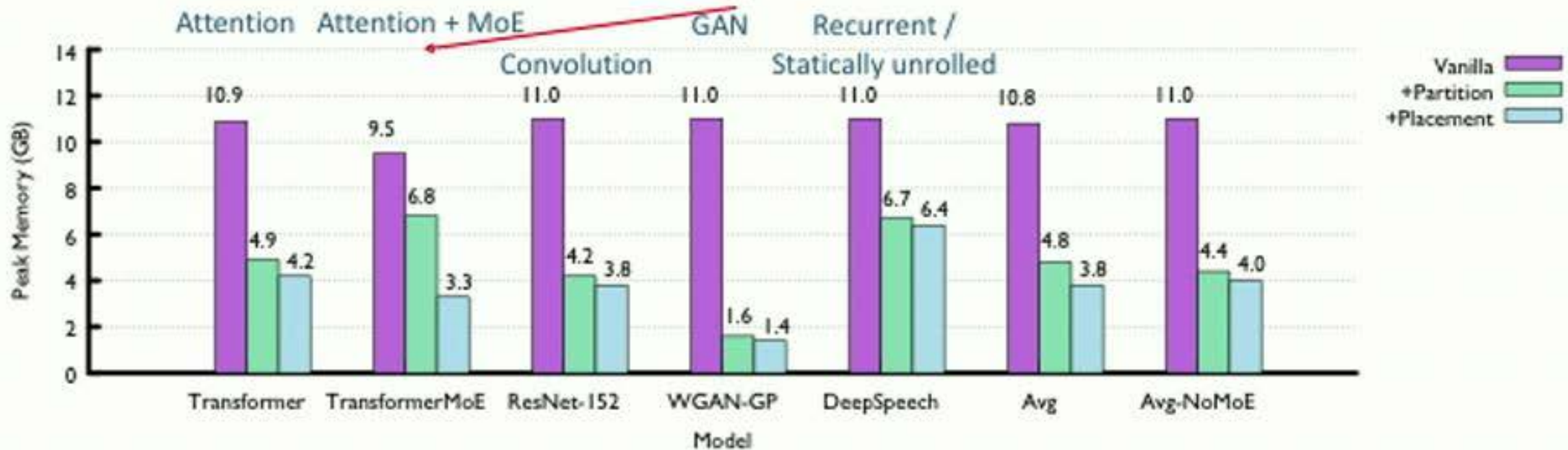
Experiment Results

800 Million parameters

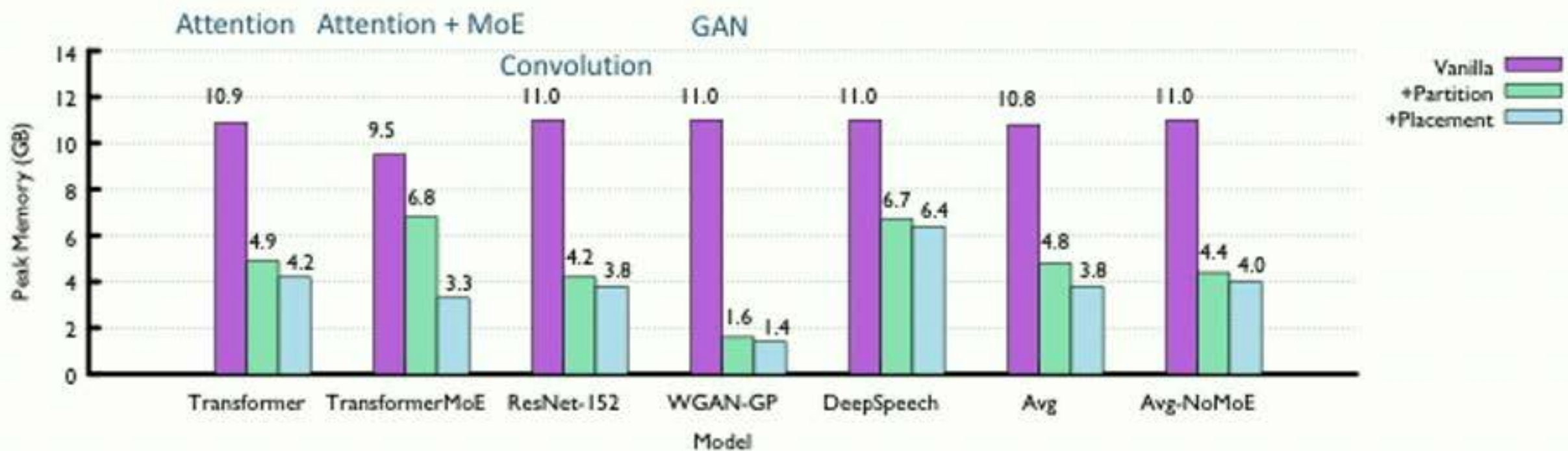


Experiment Results

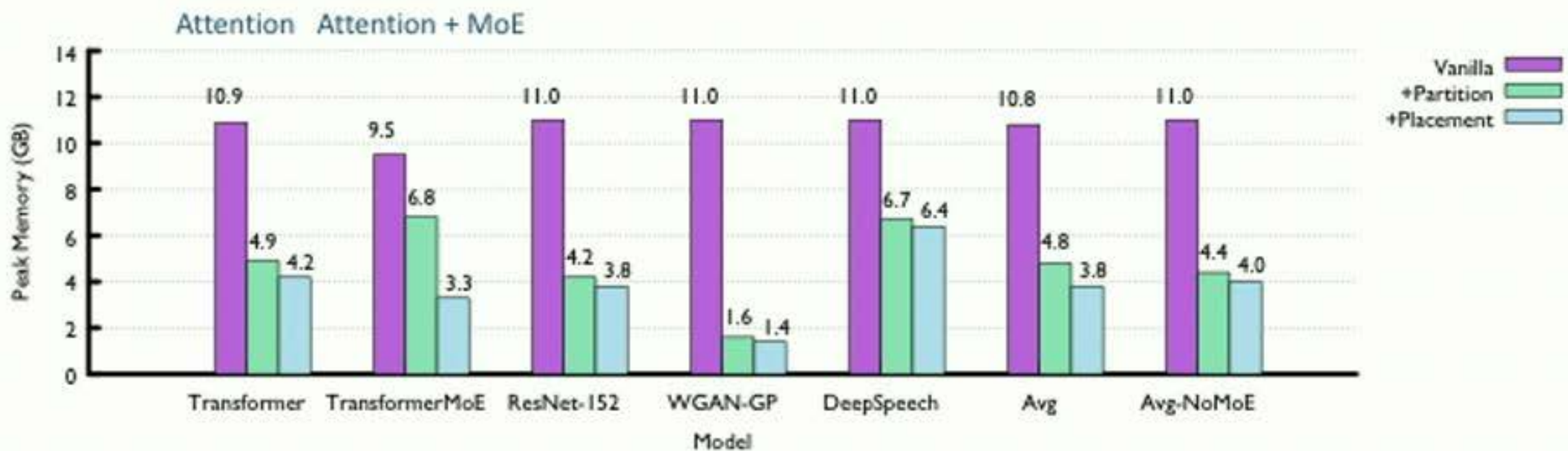
800 Million parameters



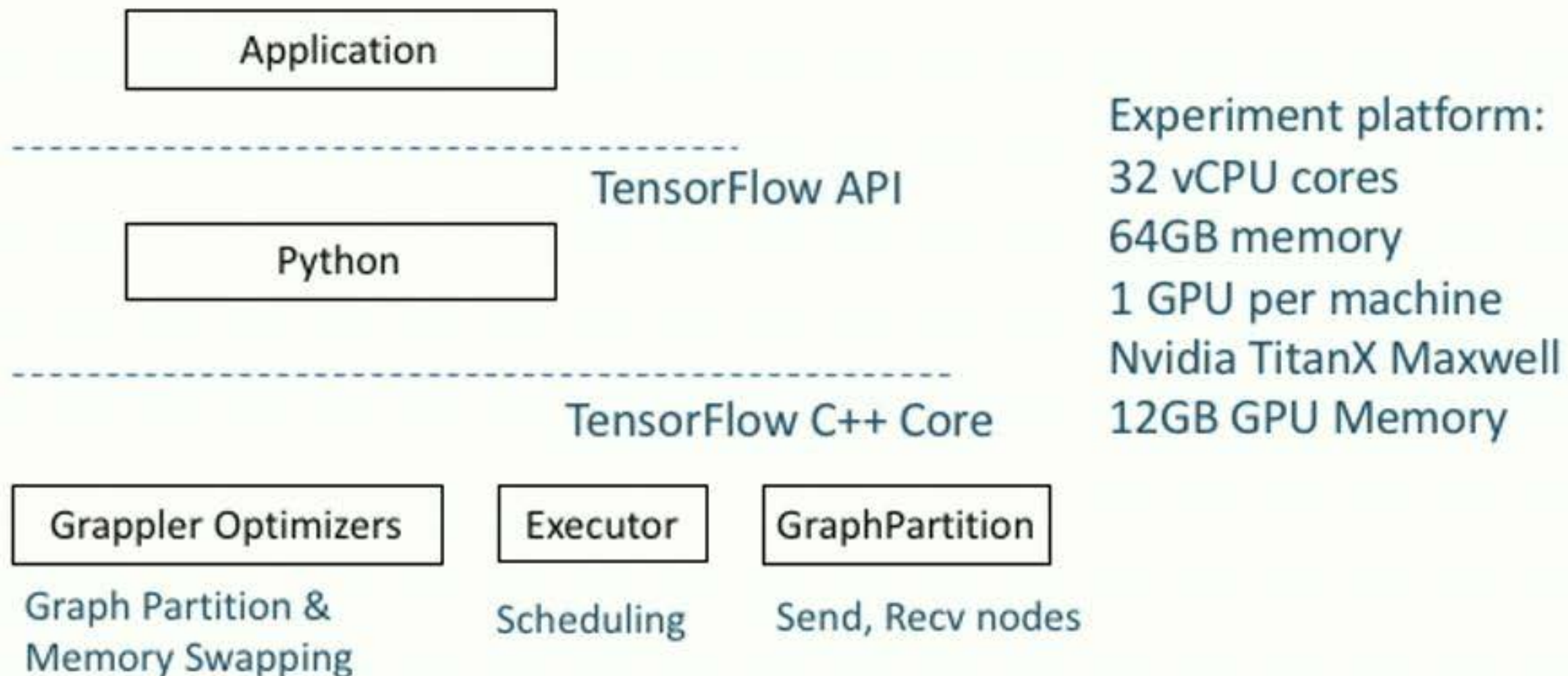
Experiment Results



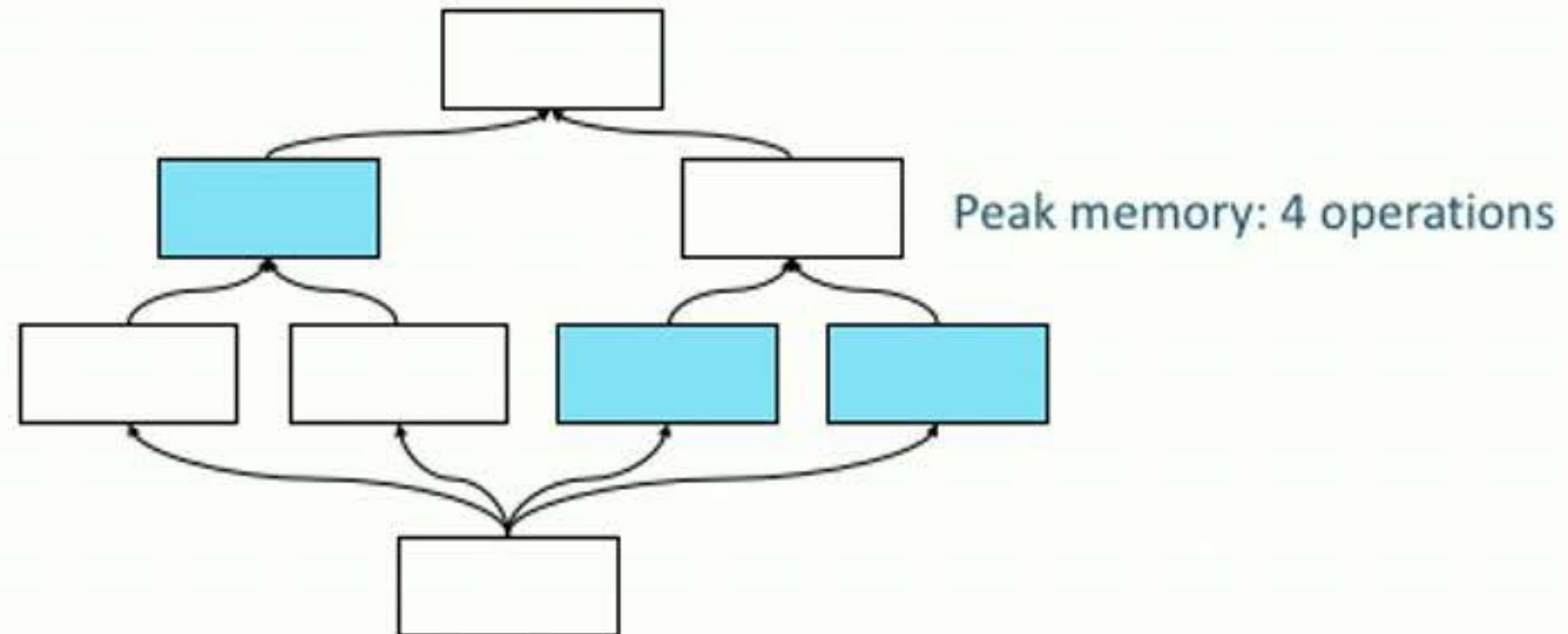
Experiment Results



Implementation & Experiment Setup



Idea #1: Limit Memory Consumption by Limiting Parallelism



TensorFlow

Breath-first traversal

Max. parallelism

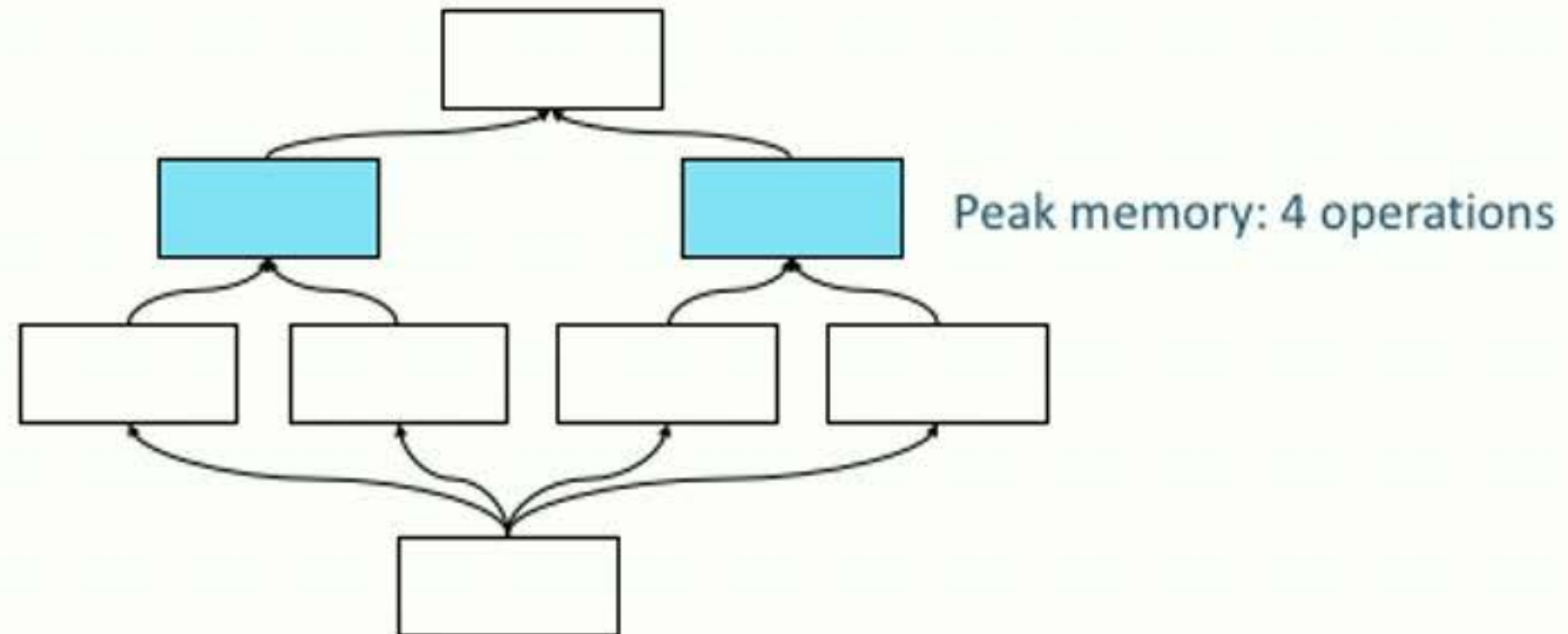
Max. memory

Linearize the graph

No parallelism

Min. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



TensorFlow

Breath-first traversal

Max. parallelism

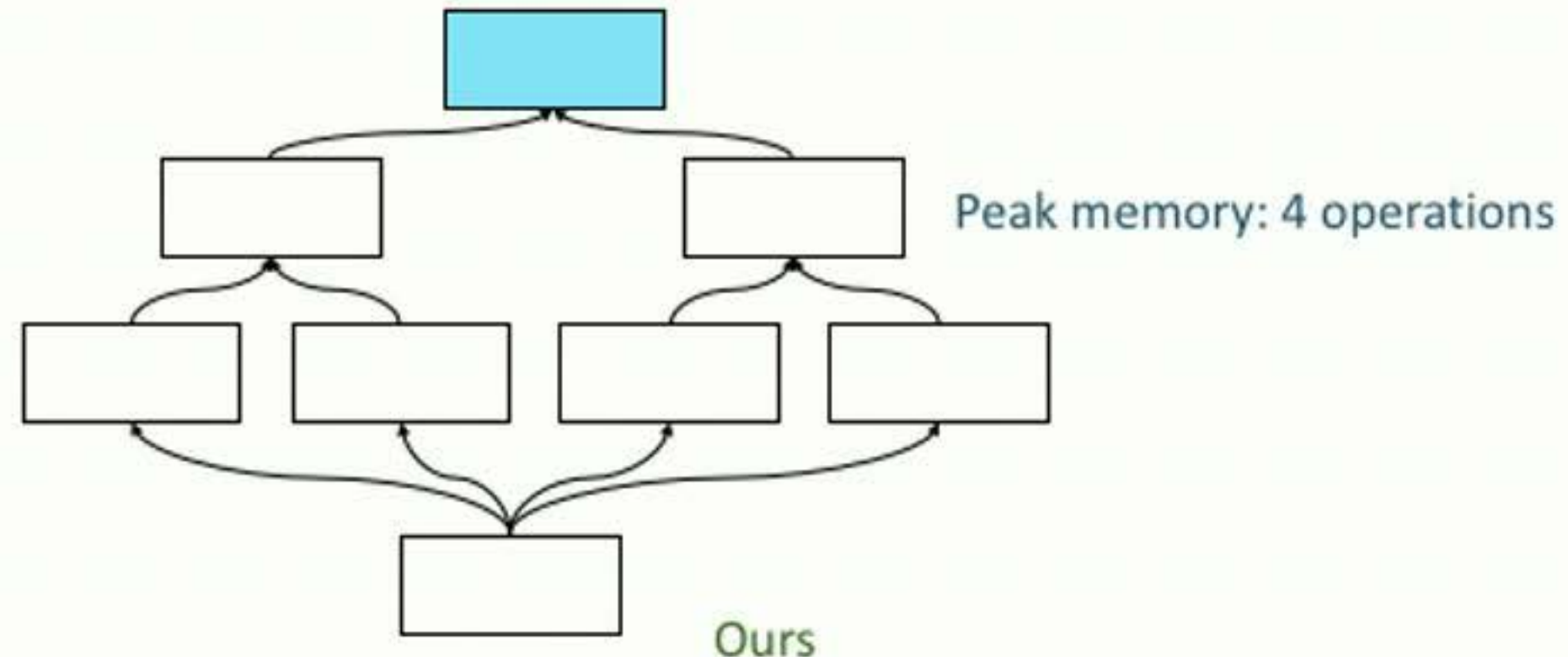
Max. memory

Linearize the graph

No parallelism

Min. memory

Idea #1: Limit Memory Consumption by Limiting Parallelism



TensorFlow

Breath-first traversal

Max. parallelism
Max. memory

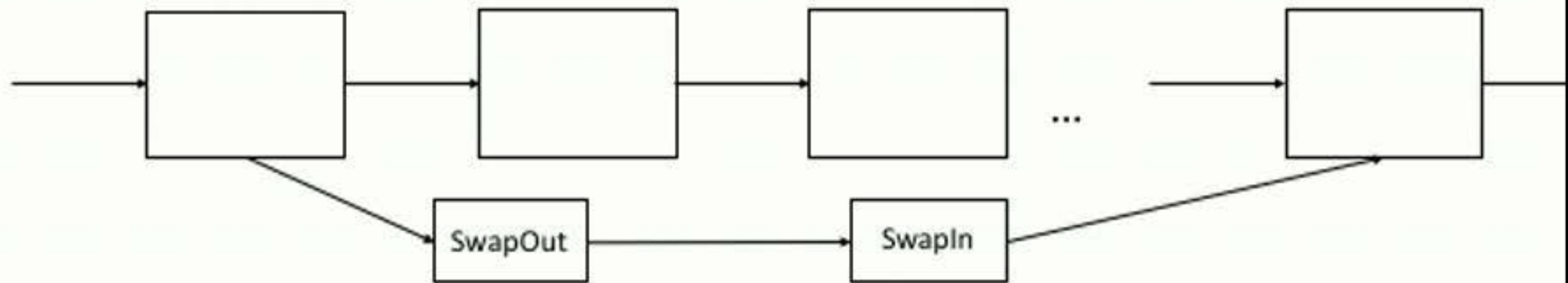
Partition the graph,
Linearize among partitions
Parallelism with partitions

Control parallelism vs. memory

Linearize the graph

No parallelism
Min. memory

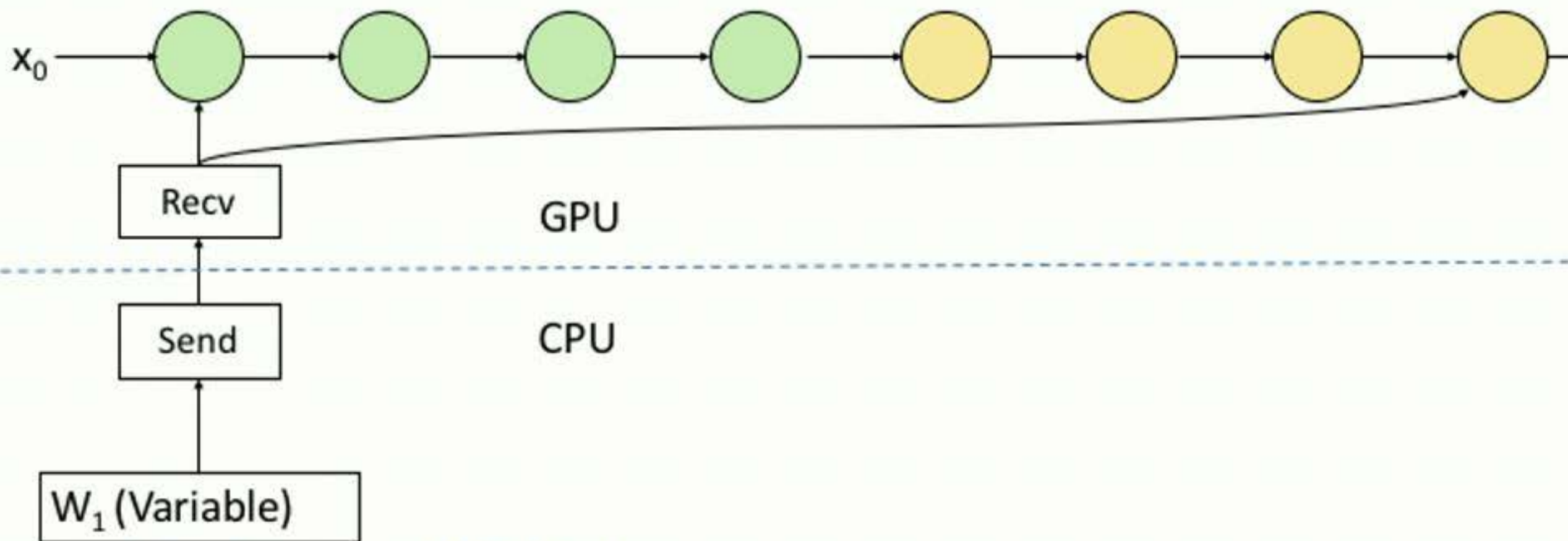
Idea #2: Offload GPU Tensors To Host Memory



Transformer
Use MoE as the Feed Forward layer
12 MoEs
32 experts per MoE
2M params per expert
~800M parameters total

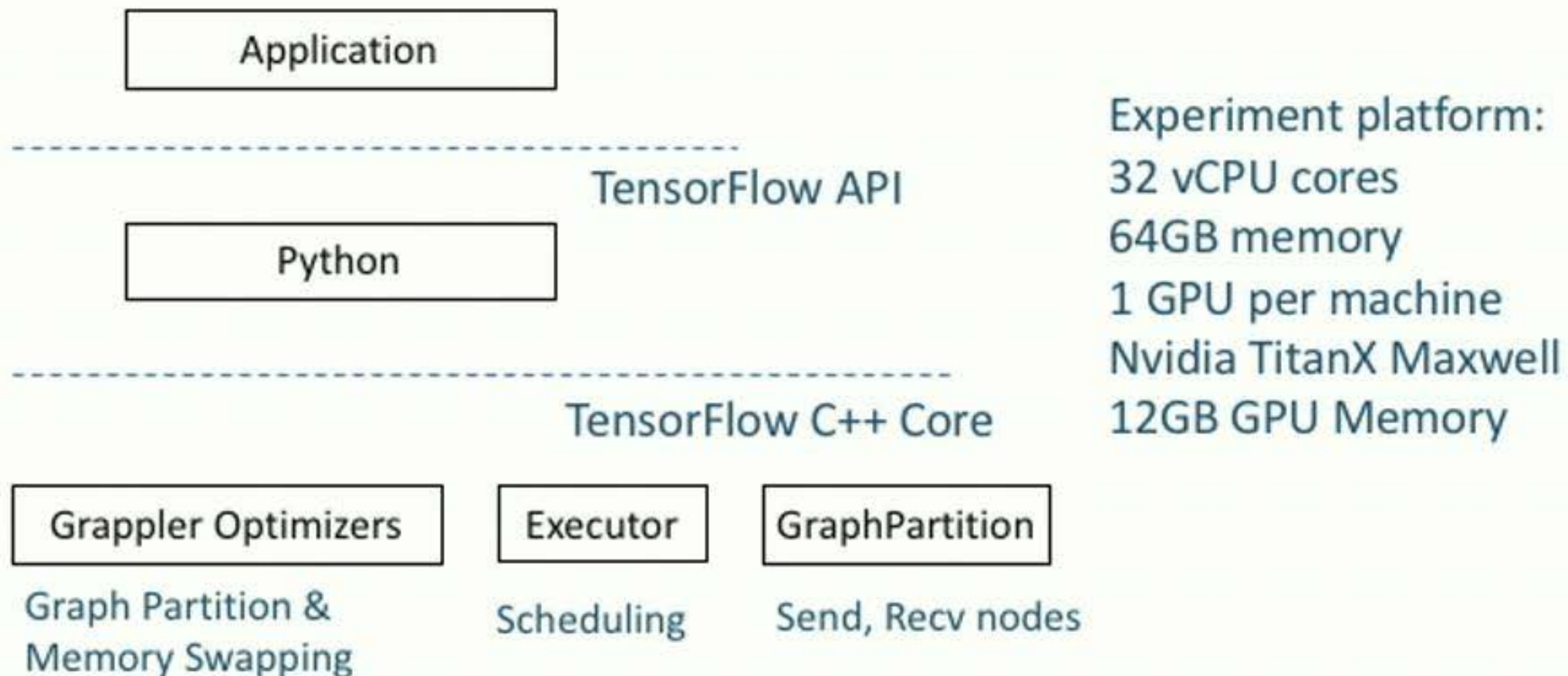
Peak memory:
9.5GB to 6.8GB

Idea #3: Place Persistent Tensors on Host Memory & Send To GPU Only When Needed



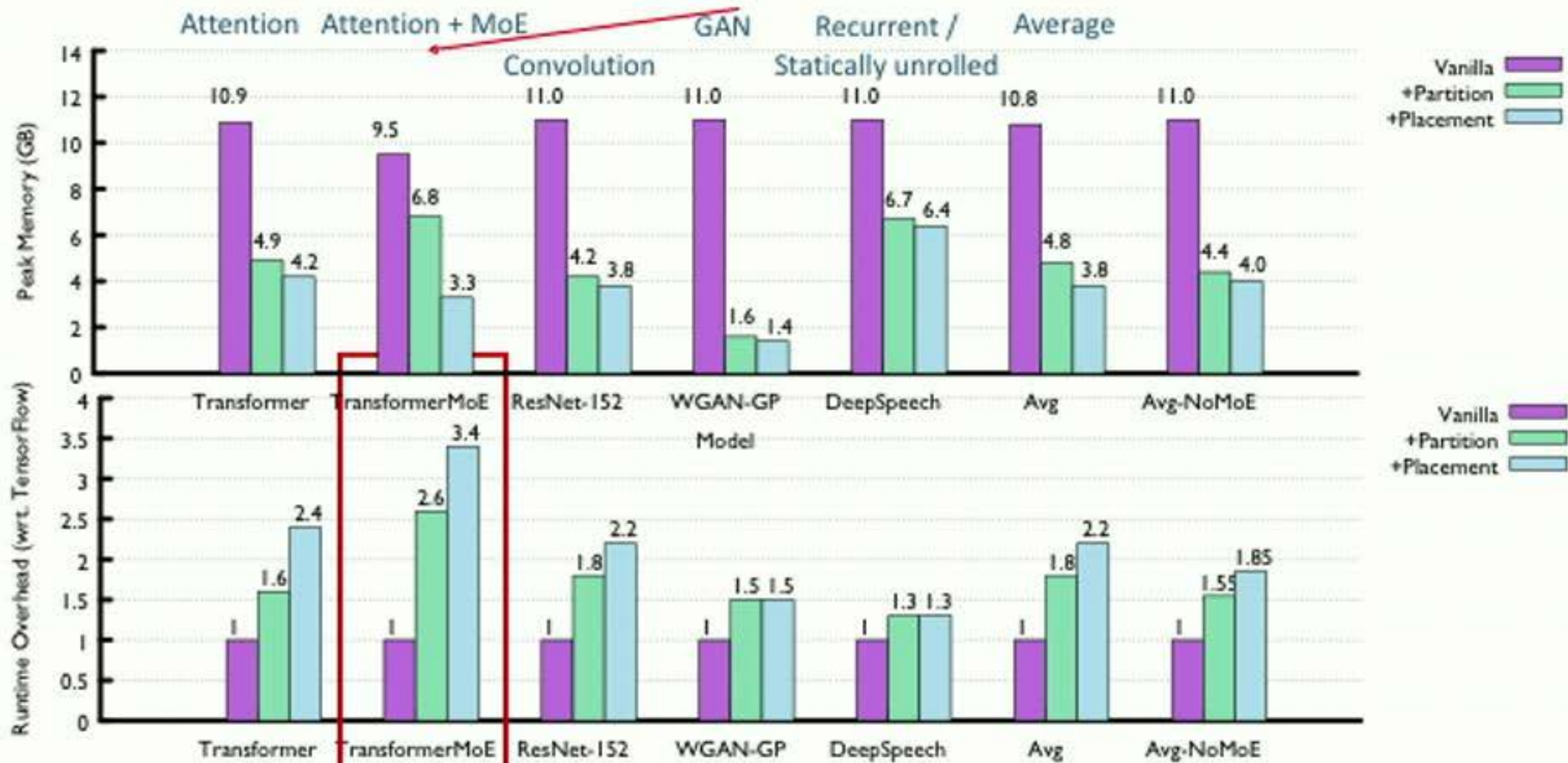
Transformer w/ MoE Peak memory:
6.8GB to 3.3GB

Implementation & Experiment Setup



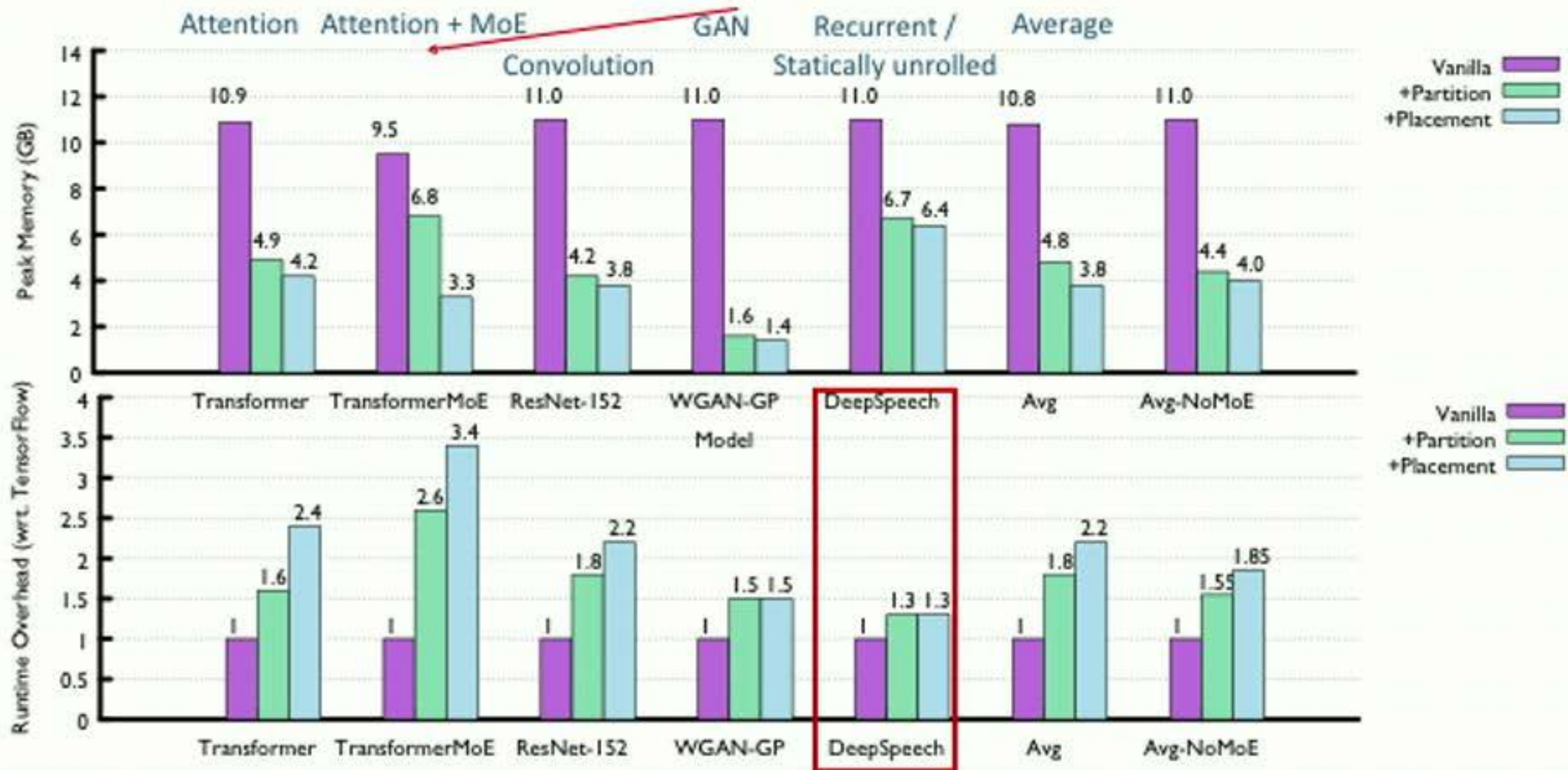
Experiment Results

800 Million parameters



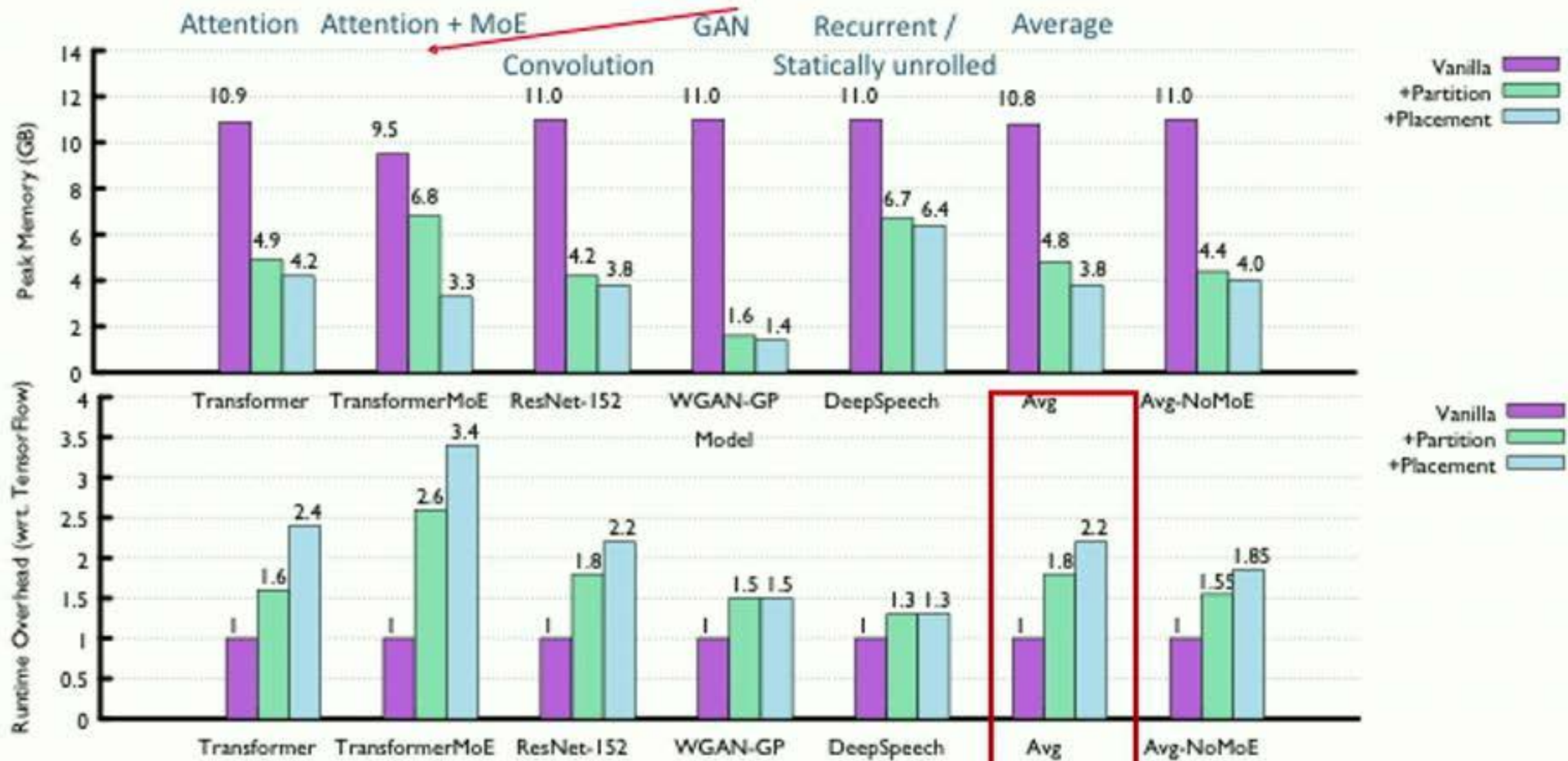
Experiment Results

800 Million parameters



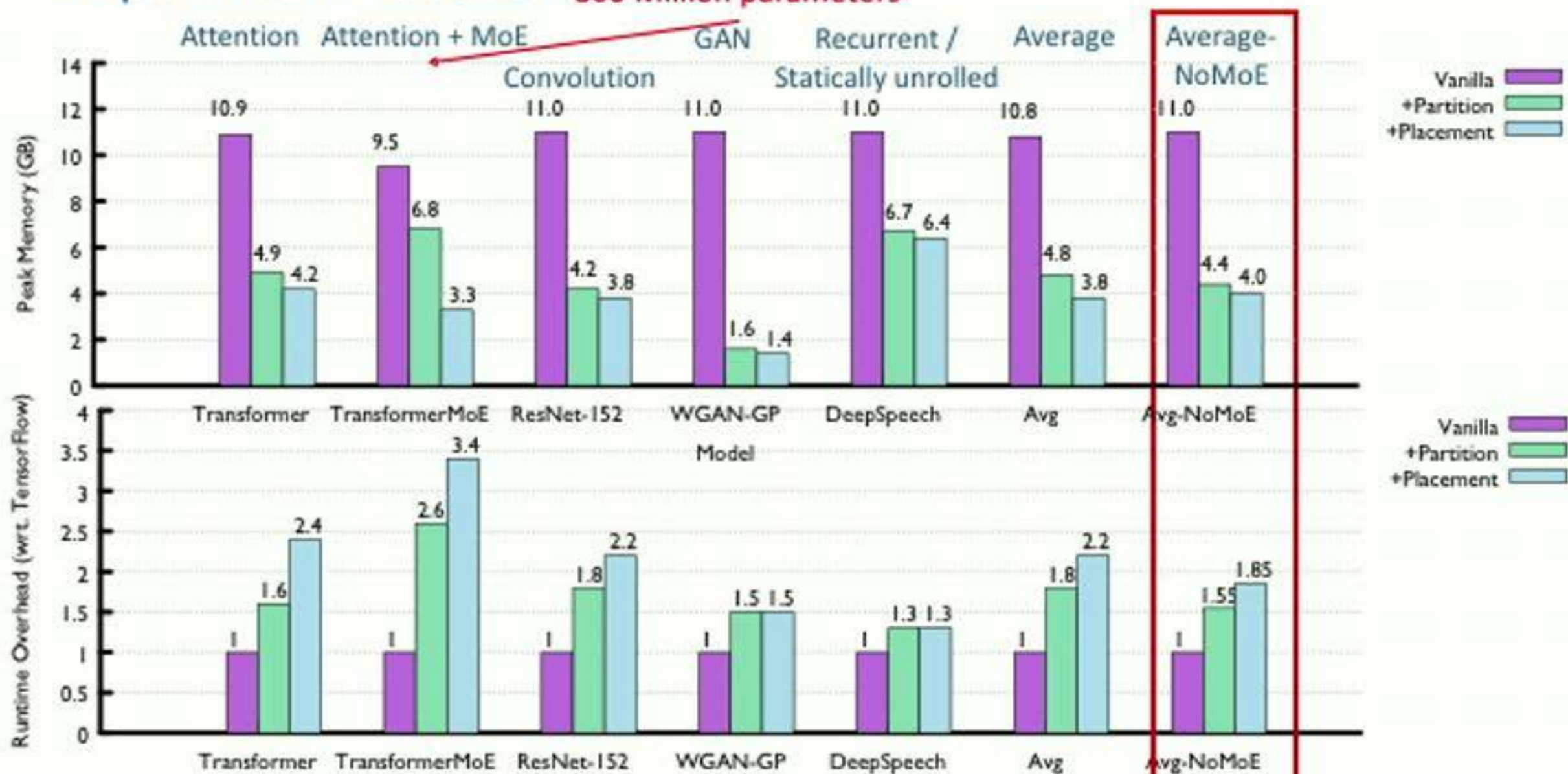
Experiment Results

800 Million parameters



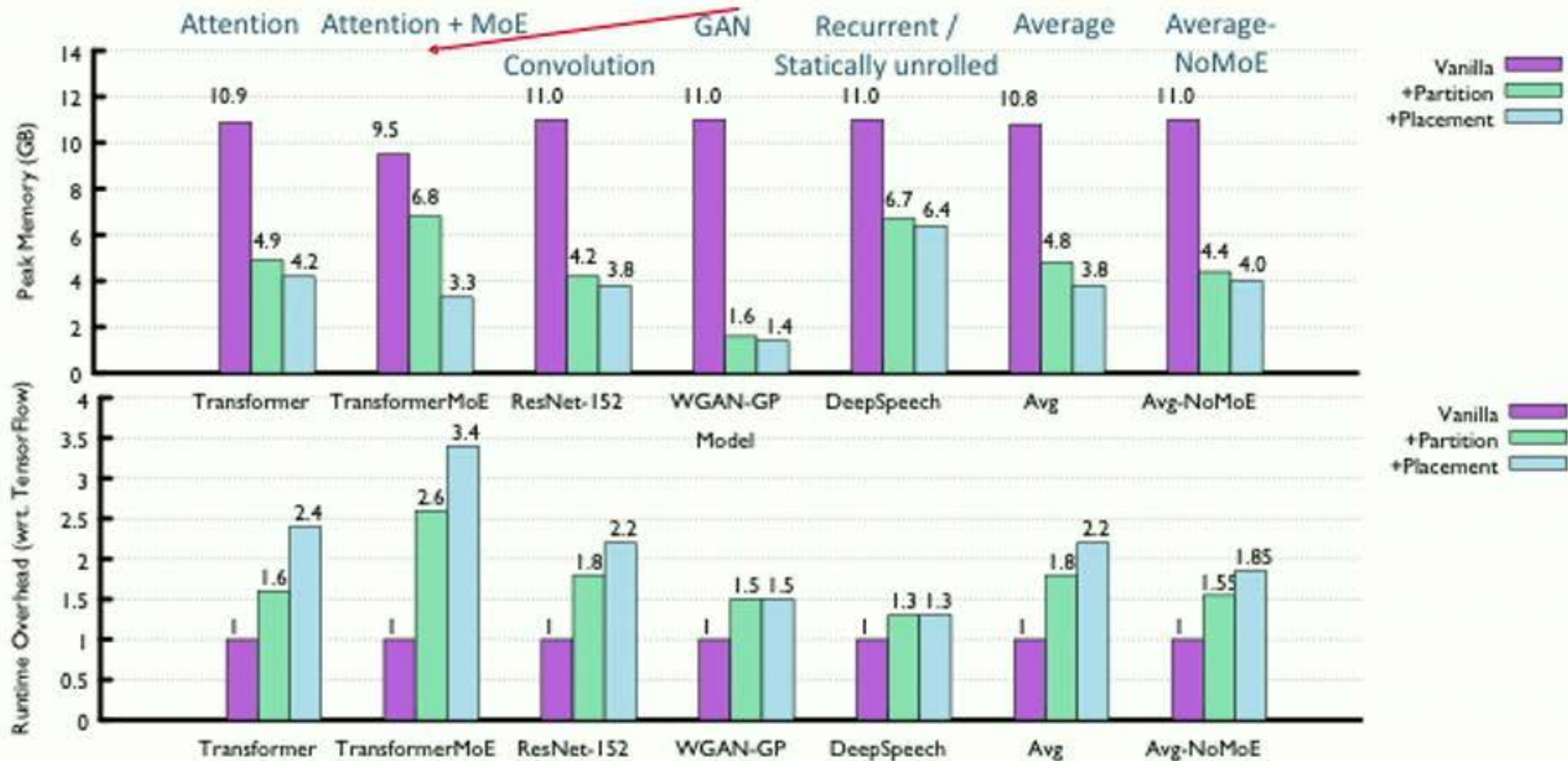
Experiment Results

800 Million parameters



Experiment Results

800 Million parameters



Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput
TensorFlow	4	0.24 Billion	19.0
TensorFlowMem	48	2.5 Billion	1.9

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput
TensorFlow	4	0.24 Billion	19.0
TensorFlowMem	48	2.5 Billion	1.9

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Batch	TensorFlow	TensorFlowMem
16	504	1916
32	235	1001

Maximum ResNet Depth

Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput	Batch	TensorFlow	TensorFlowMem
TensorFlow	4	0.24 Billion	19.0	16	504	1916
TensorFlowMem	48	2.5 Billion	1.9	32	235	1001

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Maximum ResNet Depth

System	#Machines	#Experts / MoE	#Parameters
TensorFlow	4	128	3 Billion
TensorFlow	16	208	5 Billion
TensorFlowMem	4	256	6 Billion
TensorFlowMem + Optimized MoE	4	512	12 Billion

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput	Batch	TensorFlow	TensorFlowMem
TensorFlow	4	0.24 Billion	19.0	16	504	1916
TensorFlowMem	48	2.5 Billion	1.9	32	235	1001

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Maximum ResNet Depth

System	#Machines	#Experts / MoE	#Parameters
TensorFlow	4	128	3 Billion
TensorFlow	16	208	5 Billion
TensorFlowMem	4	256	6 Billion
TensorFlowMem + Optimized MoE	4	512	12 Billion

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput	Batch	TensorFlow	TensorFlowMem
TensorFlow	4	0.24 Billion	19.0	16	504	1916
TensorFlowMem	48	2.5 Billion	1.9	32	235	1001

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Maximum ResNet Depth

System	#Machines	#Experts / MoE	#Parameters
TensorFlow	4	128	3 Billion
TensorFlow	16	208	5 Billion
TensorFlowMem	4	256	6 Billion
TensorFlowMem + Optimized MoE	4	512	12 Billion

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput	Batch	TensorFlow	TensorFlowMem
TensorFlow	4	0.24 Billion	19.0	16	504	1916
TensorFlowMem	48	2.5 Billion	1.9	32	235	1001

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Maximum ResNet Depth

System	#Machines	#Experts / MoE	#Parameters
TensorFlow	4	128	3 Billion
TensorFlow	16	208	5 Billion
TensorFlowMem	4	256	6 Billion
TensorFlowMem + Optimized MoE	4	512	12 Billion

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Partition big tensors

Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput	Batch	TensorFlow	TensorFlowMem
TensorFlow	4	0.24 Billion	19.0	16	504	1916
TensorFlowMem	48	2.5 Billion	1.9	32	235	1001

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Maximum ResNet Depth

System	#Machines	#Experts / MoE	#Parameters
TensorFlow	4	128	3 Billion
TensorFlow	16	208	5 Billion
TensorFlowMem	4	256	6 Billion
TensorFlowMem + Optimized MoE	4	512	12 Billion

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput	Batch	TensorFlow	TensorFlowMem
TensorFlow	4	0.24 Billion	19.0	16	504	1916
TensorFlowMem	48	2.5 Billion	1.9	32	235	1001

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Maximum ResNet Depth

System	#Machines	#Experts / MoE	#Parameters
TensorFlow	4	128	3 Billion
TensorFlow	16	208	5 Billion
TensorFlowMem	4	256	6 Billion
TensorFlowMem + Optimized MoE	4	512	12 Billion

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Partition big tensors

Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput	Batch	TensorFlow	TensorFlowMem
TensorFlow	4	0.24 Billion	19.0	16	504	1916
TensorFlowMem	48	2.5 Billion	1.9	32	235	1001

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Maximum ResNet Depth

System	#Machines	#Experts / MoE	#Parameters
TensorFlow	4	128	3 Billion
TensorFlow	16	208	5 Billion
TensorFlowMem	4	256	6 Billion
TensorFlowMem + Optimized MoE	4	512	12 Billion

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Scaling Model Size

System	#Experts / MoE	#Parameters	Throughput	Batch	TensorFlow	TensorFlowMem
TensorFlow	4	0.24 Billion	19.0	16	504	1916
TensorFlowMem	48	2.5 Billion	1.9	32	235	1001

Transformer w/ MoE

12 MoEs, 4M parameters per expert

Maximum ResNet Depth

System	#Machines	#Experts / MoE	#Parameters
TensorFlow	4	128	3 Billion
TensorFlow	16	208	5 Billion
TensorFlowMem	4	256	6 Billion
TensorFlowMem + Optimized MoE	4	512	12 Billion

Distributed Transformer w/ MoE

12 MoEs, 2M parameters per expert

Partition big tensors

Recurrent Neural Networks –Scaling Sequence Length

Sequence Length	100	200	400	500	800
TensorFlow	1.15	2.31	4.65	OOM	OOM
TensorFlowMem	1.56	3.03	6.03	N/A	12.01

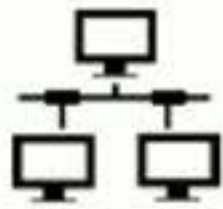
Mozilla DeepSpeech, statically unrolled RNN

Mini-batch size = 128

Time per mini-batch (seconds)

Summary

Summary

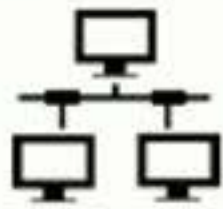


Fine-grained communication

Prioritization based on relative magnitude

Prioritization based on when values are used

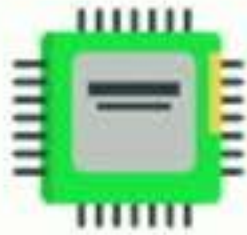
Summary



Fine-grained communication

Prioritization based on relative magnitude

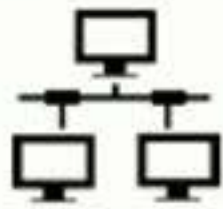
Prioritization based on when values are used



Statically analyze memory accesses

Schedule independent computation in parallel

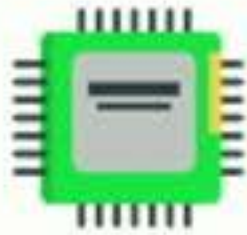
Summary



Fine-grained communication

Prioritization based on relative magnitude

Prioritization based on when values are used



Statically analyze memory accesses

Schedule independent computation in parallel

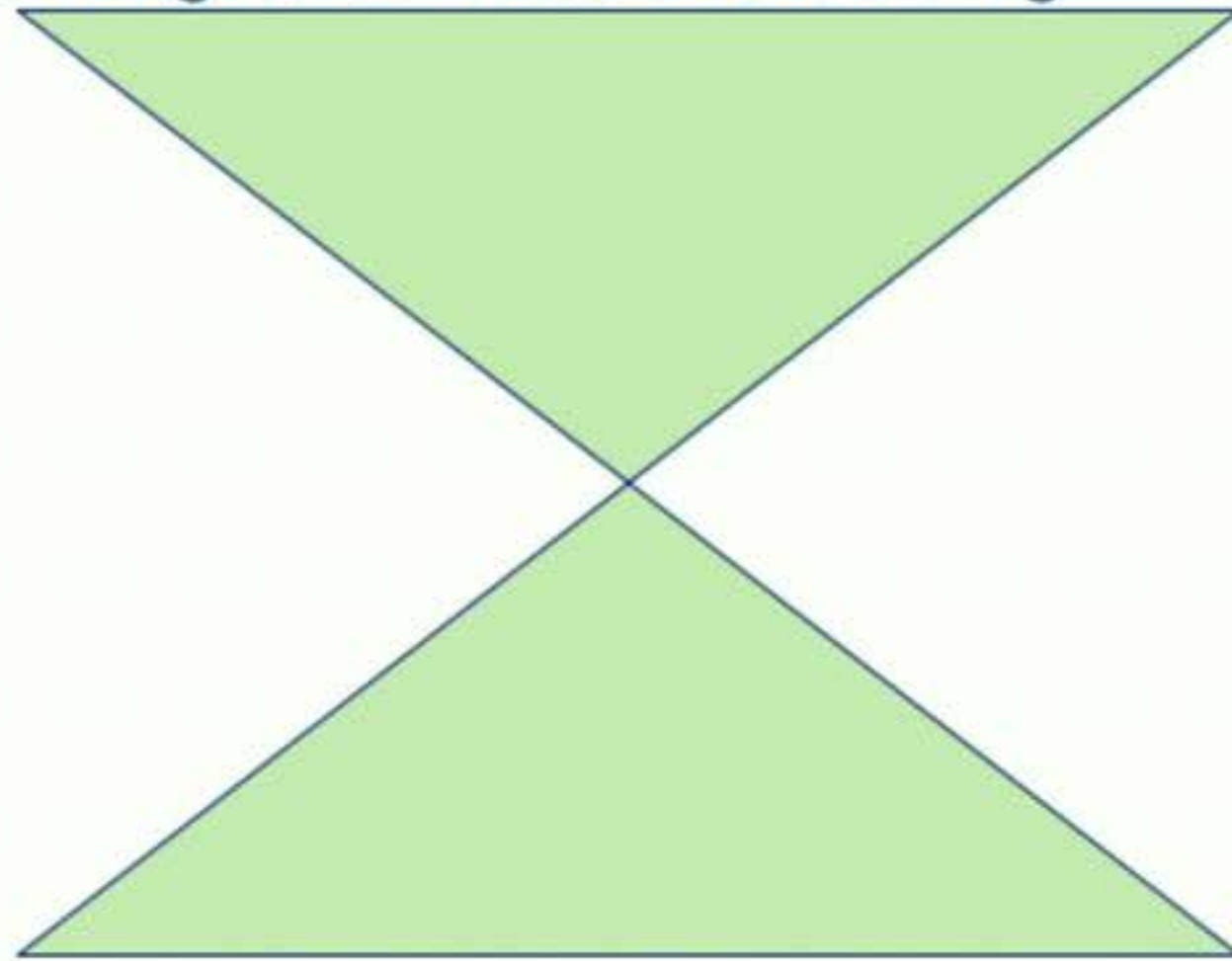


Partitioned computation graph

Leverage cheap host memory

Machine Learning Is Still Fast Advancing

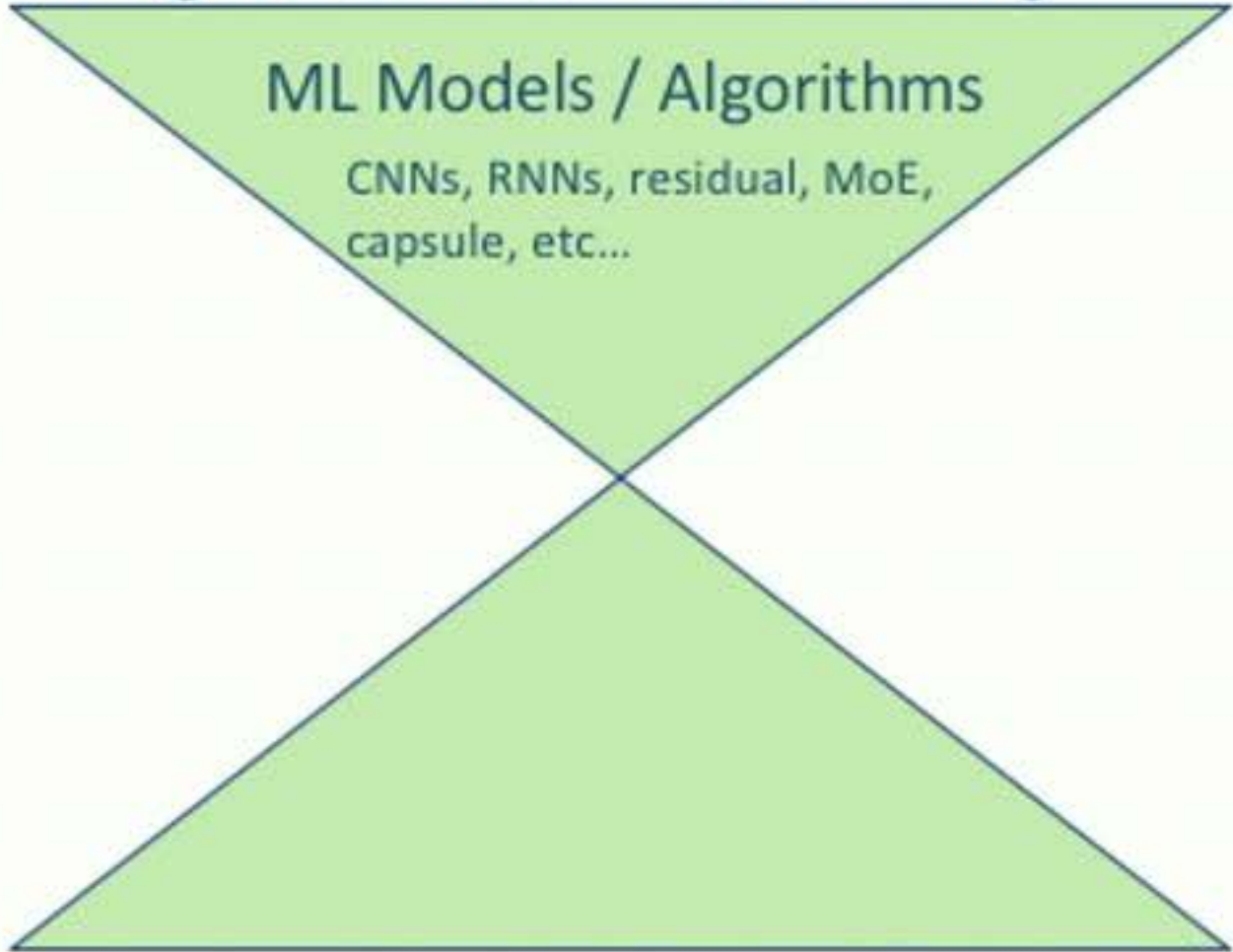
Machine Learning Is Still Fast Advancing



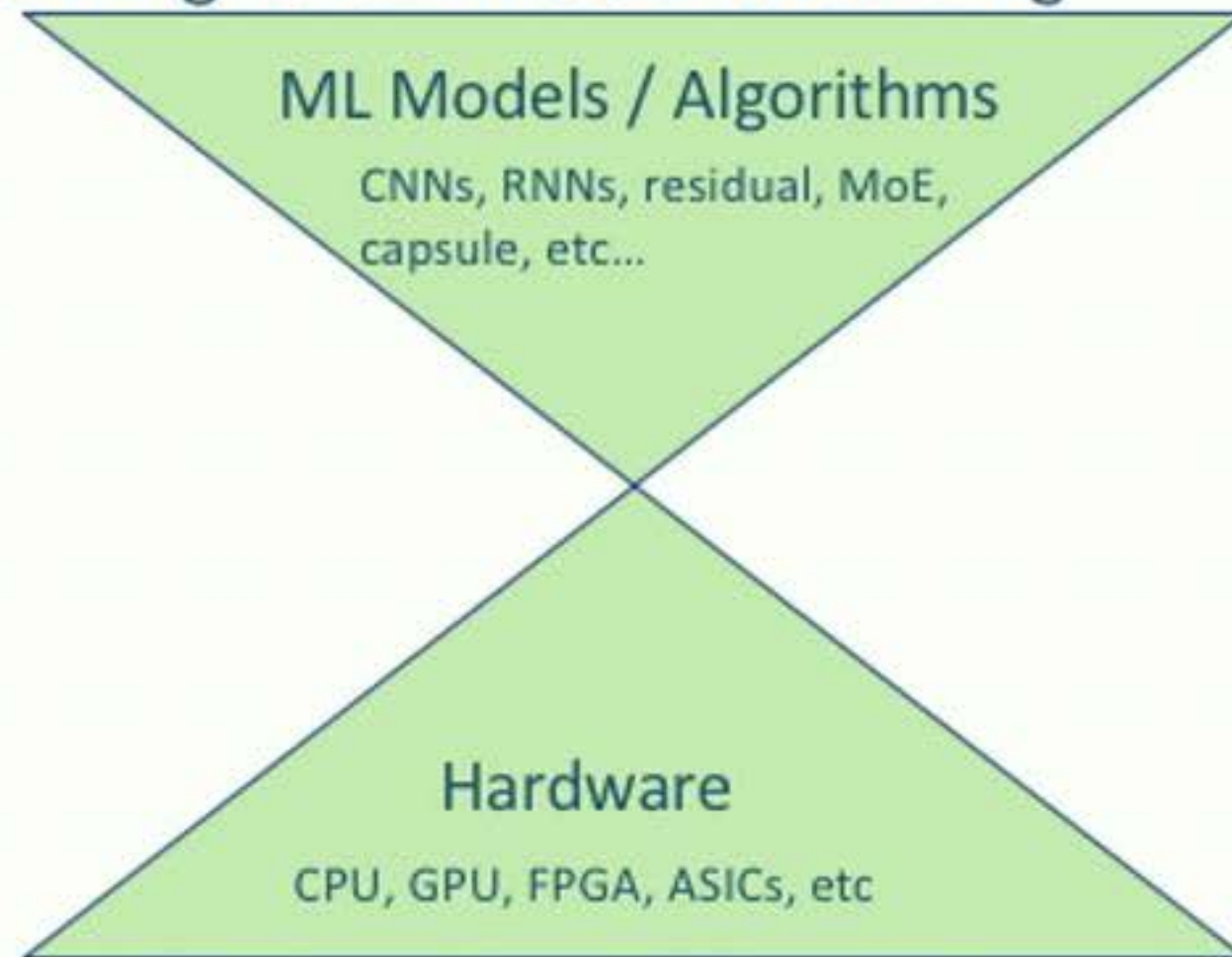
Machine Learning Is Still Fast Advancing

ML Models / Algorithms

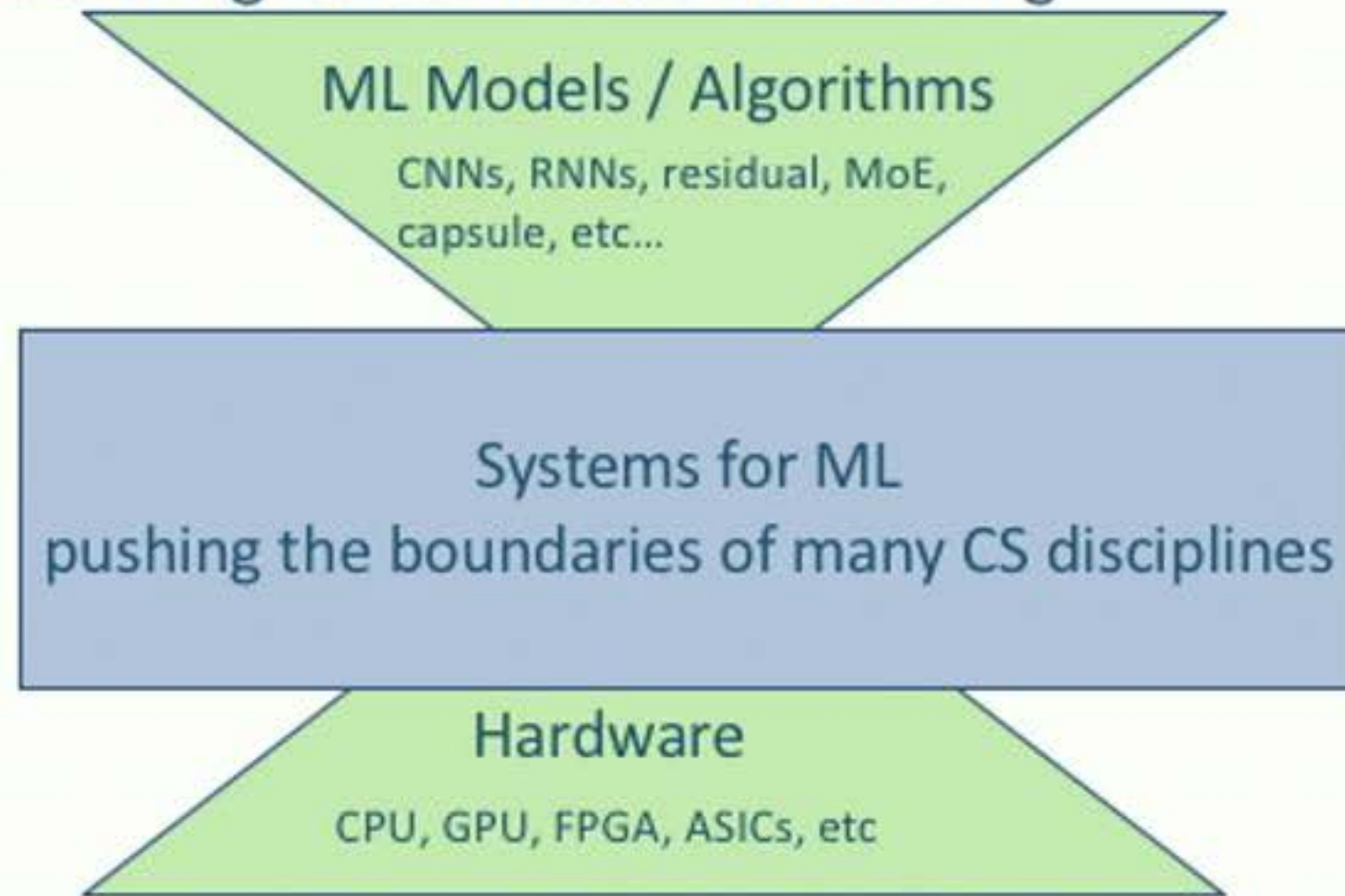
CNNs, RNNs, residual, MoE,
capsule, etc...



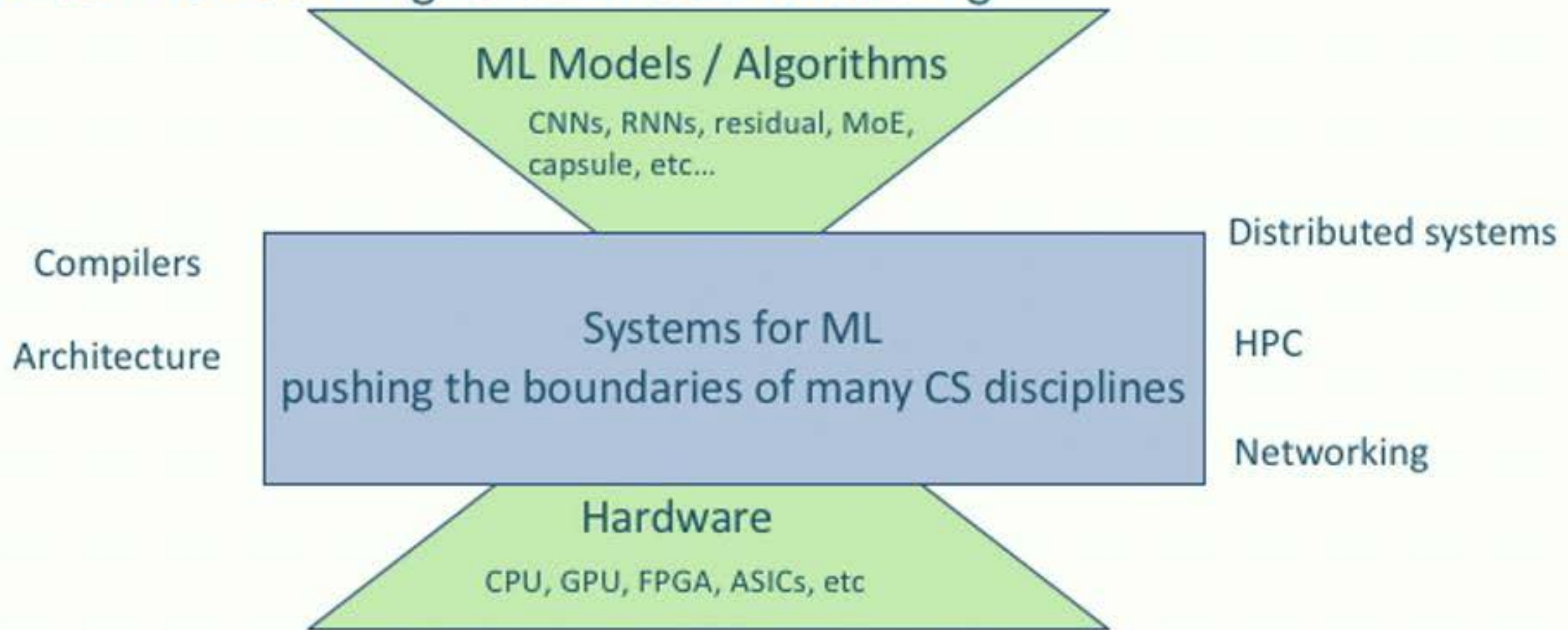
Machine Learning Is Still Fast Advancing



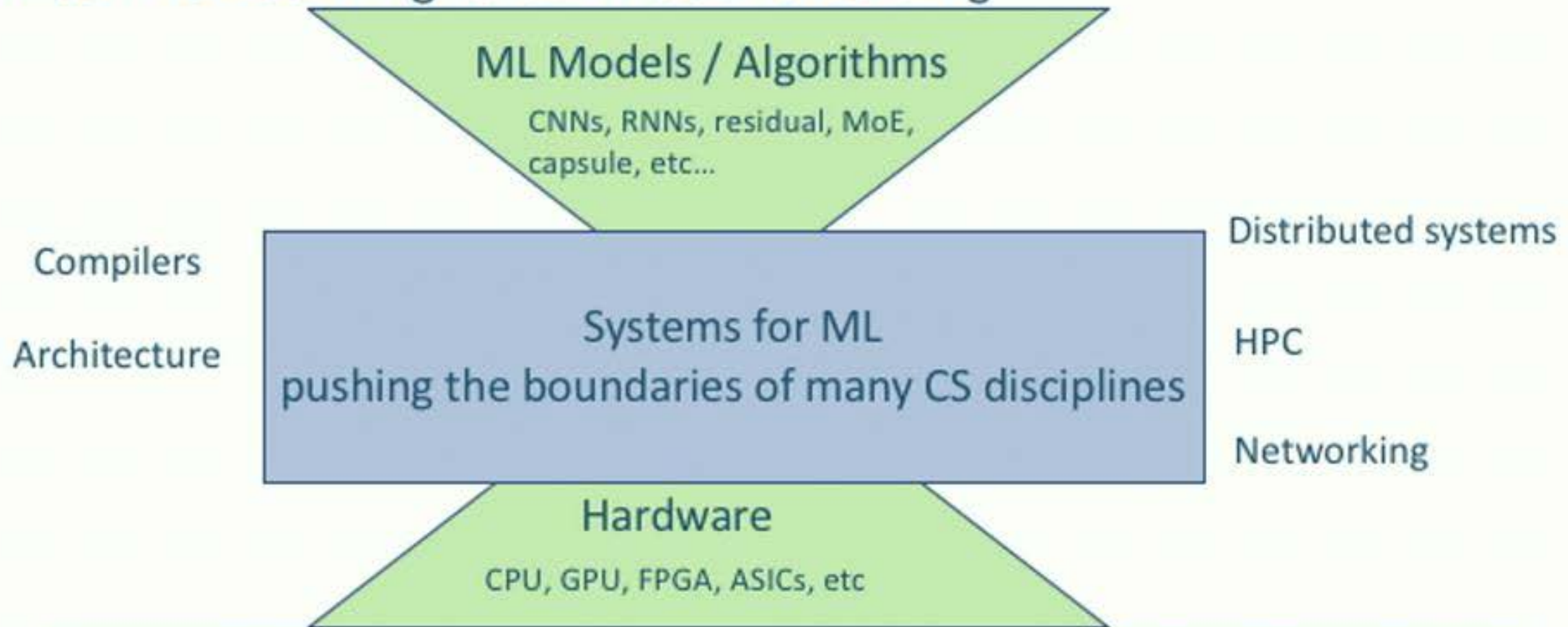
Machine Learning Is Still Fast Advancing



Machine Learning Is Still Fast Advancing



Machine Learning Is Still Fast Advancing



How to support the expanding ML computation?

How to take advantage of new hardware?

Future Directions

Future Directions

Programming support and compilation

Future Directions

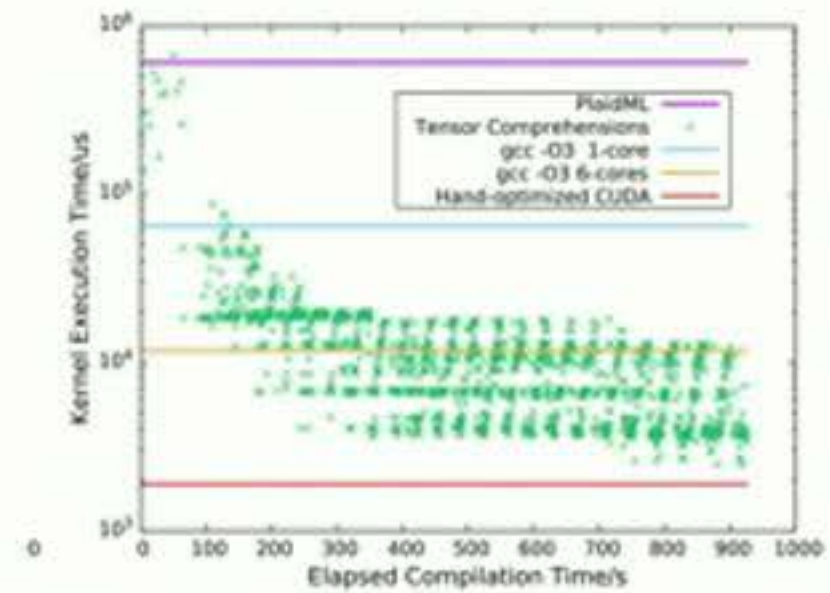
Programming support and compilation

- New operations, e.g., capsule?

Future Directions

Programming support and compilation

- New operations, e.g., capsule?



(c) Capsule Kernel

Future Directions

Programming support and compilation

- New operations, e.g., capsule?
- New control flow primitives, e.g., functions?

Future Directions

Programming support and compilation

- New operations, e.g., capsule?
- New control flow primitives, e.g., functions?
- New hardware, e.g., ASICs

Future Directions

Programming support and compilation

- New operations, e.g., capsule?
- New control flow primitives, e.g., functions?
- New hardware, e.g., ASICs

Model parallelism

- Operation partitioning
- Device placement, even dynamic placement for dynamic control flow

Future Directions

Programming support and compilation

- New operations, e.g., capsule?
- New control flow primitives, e.g., functions?
- New hardware, e.g., ASICs

Model parallelism

- Operation partitioning
- Device placement, even dynamic placement for dynamic control flow

ML-driven optimizations for ML systems

- Complex design space

Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

Techniques

Trade-off

Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

Techniques

Scheduling

Trade-off

Degree of Parallelism

Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

Techniques

Scheduling

Gradient checkpointing & **Constant folding**

Trade-off

Degree of Parallelism

Computation

Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

Techniques

Scheduling

Gradient checkpointing & **Constant folding**

Memory swapping & Device placement

Trade-off

Degree of Parallelism

Computation

Communication

Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

Techniques

Scheduling

Gradient checkpointing & **Constant folding**

Memory swapping & Device placement

Quantization

Trade-off

Degree of Parallelism

Computation

Communication

Accuracy

Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

Techniques

Trade-off

Scheduling

Degree of Parallelism

Gradient checkpointing & **Constant folding**

Computation

Memory swapping & Device placement

Communication

Quantization

Accuracy

Challenges:

- 1) Scheduling is NP-complete;
- 2) Best configuration depends on the program and hardware;
- 3) Techniques are inter-dependent

Example: Fast & Memory-Efficient Deep Learning

Many ways to reduce memory consumption, with different trade-offs

Techniques

Trade-off

Scheduling

Degree of Parallelism

Gradient checkpointing & **Constant folding**

Computation

Memory swapping & Device placement

Communication

Quantization

Accuracy

Challenges:

- 1) Scheduling is NP-complete;
- 2) Best configuration depends on the program and hardware;
- 3) Techniques are inter-dependent



Minimize training time subject to memory constraints?