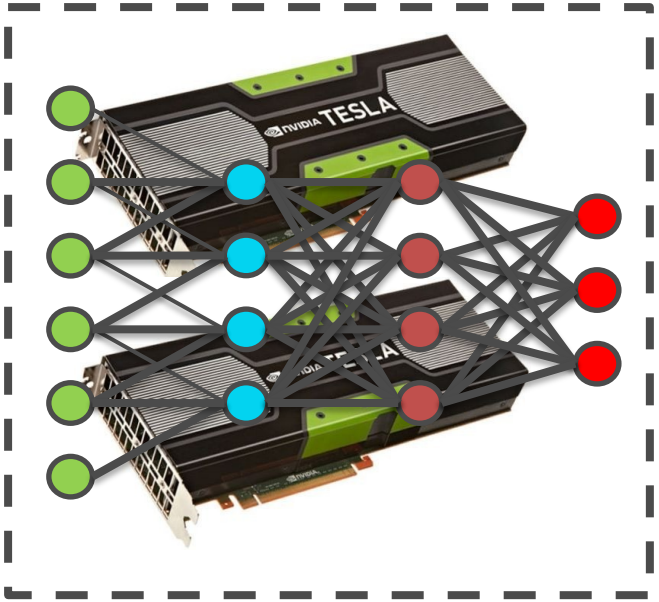
Machine Learning Parallelism Could Be Adaptive, Composable and Automated

Ph.D. Thesis Oral

Hao Zhang

Machine Learning Computation

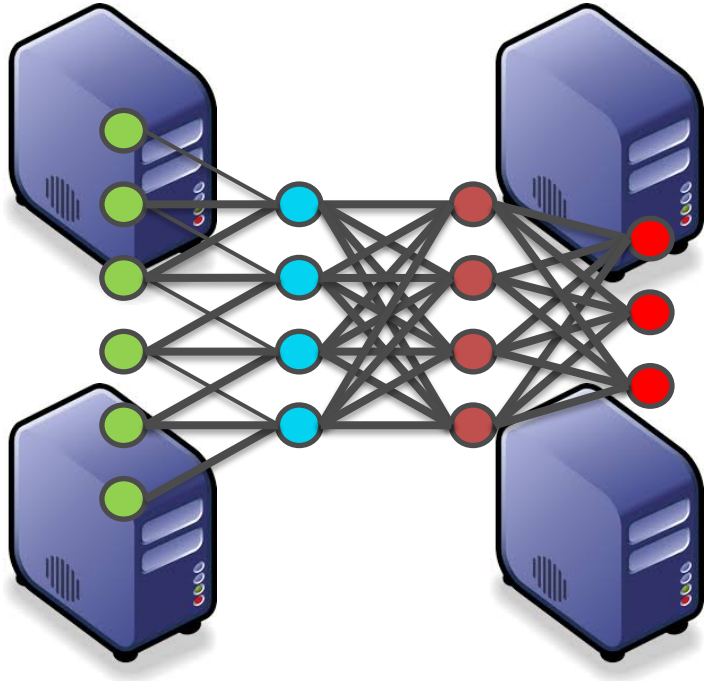
Training data:
images w/ labels



Computational Worker w/ GPUs

Distributed Machine Learning

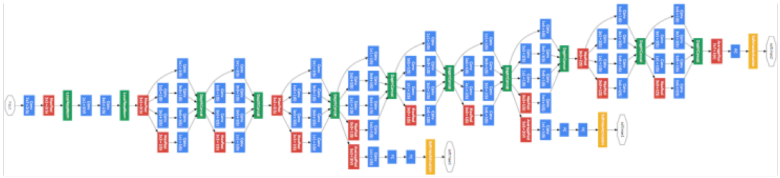
Large scale
Training data

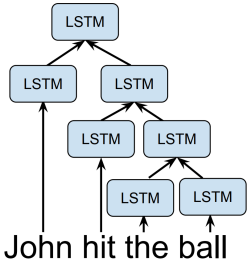


A Cluster of Workers with GPUs

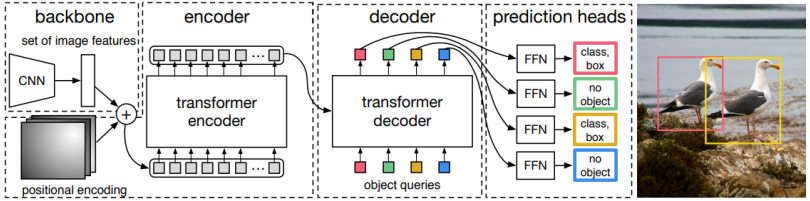
Nowadays ML Models

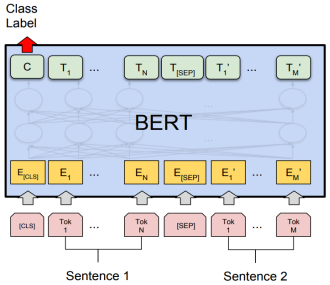
Highly Structured



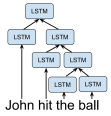


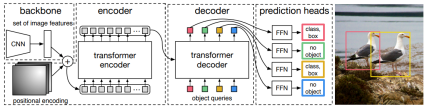
Increasingly Composable

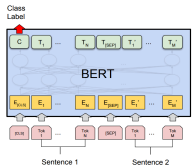




Distribute a Model



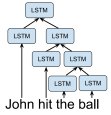


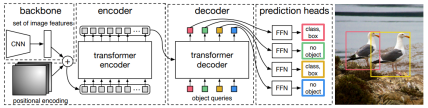


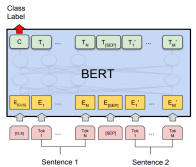
$$\theta^{(t)} = \theta^{(t-1)} + \varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})$$

Model Data

Distribute a Model



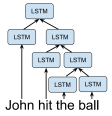


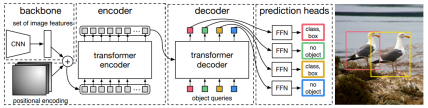


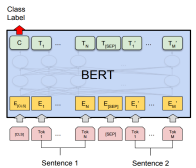
Communication Architecture	Parameter Server	AllReduce/ Collective	P2P	Partial Broadcasting
Consistency Models	Synchronous	Asynchronous	Stale Synchronous	
Partitioning/ Placement	Graph Partitioning	Operator Partitioning	Device Placement	
Encoding/ Decoding	Gradient Quantization	Deep Compression		
Scheduling	WFBP	Pipeline Parallelism	Byte Scheduler	

...

Usability Challenges: Implementations

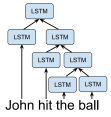


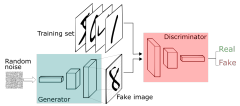


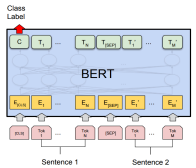


Horovod	Bosen	BytePS	Orpheus
Sergeev et al. 2018	Wei et al. 2015	Peng et al. 2019	Xie et al. 2019
Deep Compression	ColocRL	FlexFlow	Tofu
Han et al. 2016	Mirhoseini et al. 2017	Jia et al. 2019	Wang et al. 2019
tf.distribute .strategy	Mesh-TensorFlow	Poseidon	Parallax
	Shazeer et al. 2018	Zhang et al. 2017	Kim et al. 2019
GeePS	IterStore	Litz	Next…?
Cui et al. 2018	Cui et al. 2014	Qiao et al. 2017	

Usability Challenge: Distinct Interface







Operator Partitioning | Mesh-TensorFlow

```
# tf_images is a tf.Tensor with shape [100, 28, 28] and dtype tf.float32
# tf_labels is a tf.Tensor with shape [100] and dtype tf.int32
graph = mtf.Graph()
mesh = mtf.Mesh(graph, "my_mesh")
batch_dim = mtf.Dimension("batch", 100)
rows_dim = mtf.Dimension("rows", 28)
cols_dim = mtf.Dimension("cols", 28)
hidden_dim = mtf.Dimension("hidden", 1024)
classes_dim = mtf.Dimension("classes", 10)
images = mtf.import_tf_tensor(
    mesh, tf_images, shape=[batch_dim, rows_dim, cols_dim])
labels = mtf.import_tf_tensor(mesh, tf_labels, [batch_dim])
w1 = mtf.get_variable(mesh, "w1", [rows_dim, cols_dim, hidden_dim])
w2 = mtf.get_variable(mesh, "w2", [hidden_dim, classes_dim])
```

Write partitioning layout (“mesh”)

Graph Partitioning | ColocRL

```
with tf.device("/job:local/task:1"):
    first_batch = tf.slice(x, [0], [50])
    mean1 = tf.reduce_mean(first_batch)

with tf.device("/job:local/task:0"):
    second_batch = tf.slice(x, [50], [-1])
    mean2 = tf.reduce_mean(second_batch)
    mean = (mean1 + mean2) / 2


with tf.Session("grpc://localhost:2222") as sess:
    result = sess.run(mean, feed_dict={x: np.random.random(100)})
    print(result)
```

Specify op-device mapping

Collective AllRedue | Horovod

```
# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
# initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)
```

Patch optimizers

Collective AllRedue | PyTorch

```
""" Distributed Synchronous SGD Example """
def run(rank, size):
    torch.manual_seed(1234)
    train_set, bsz = partition_dataset()
    model = Net()
    optimizer = optim.SGD(model.parameters(),
                          lr=0.01, momentum=0.5)

    num_batches = ceil(len(train_set.dataset) / float(bsz))
    for epoch in range(10):
        epoch_loss = 0.0
        for data, target in train_set:
            optimizer.zero_grad()
            output = model(data)
            loss = F.nll_loss(output, target)
            epoch_loss += loss.item()
            loss.backward()
            average_gradients(model)
            optimizer.step()
        print('Rank ', dist.get_rank(), ', epoch ',
              epoch, ': ', epoch_loss / num_batches)
```

Manually average grads

Performance Challenge #1

Lack of sub-model optimization: the overall distribution strategy of a model is the composition of the fittest strategy of each model building block.

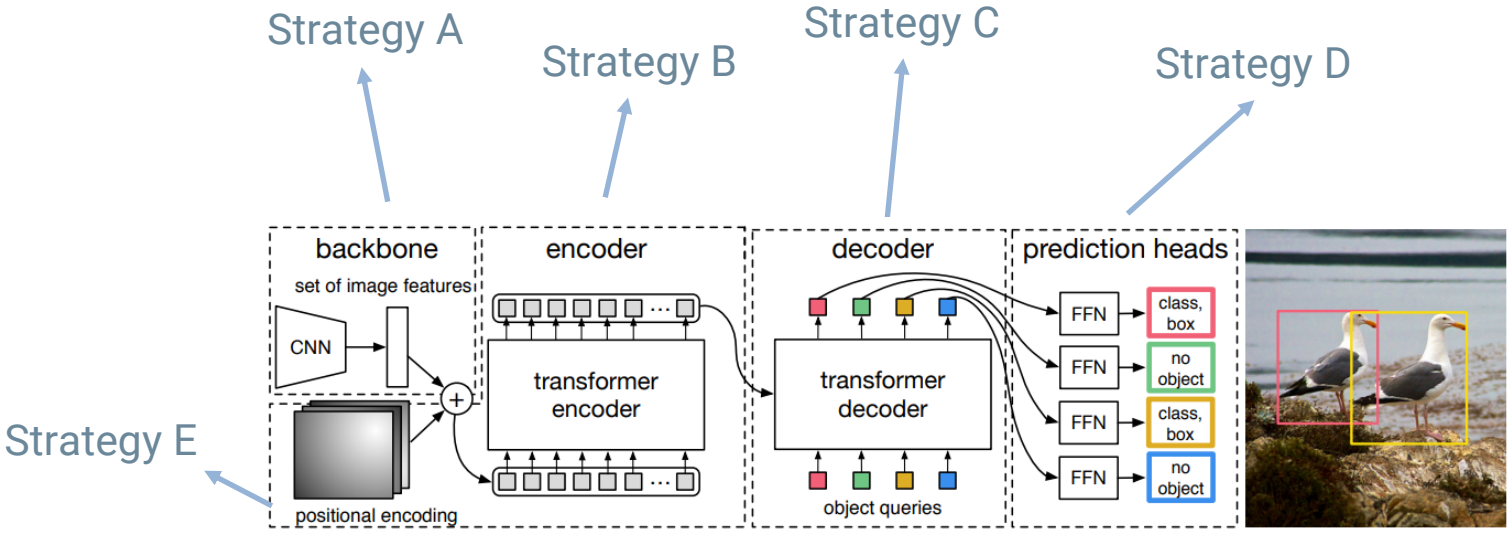
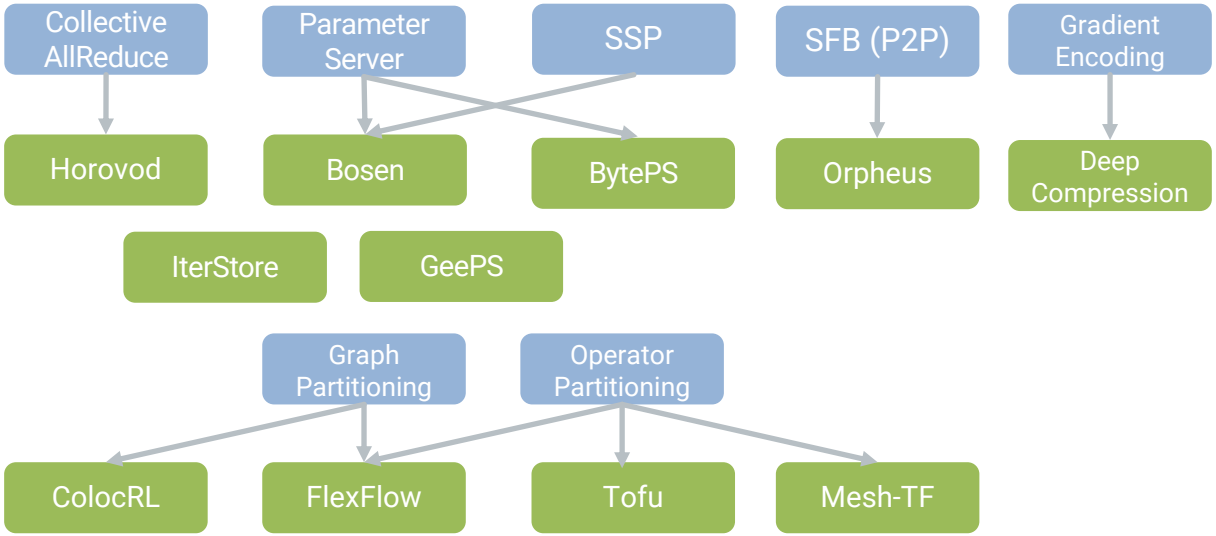


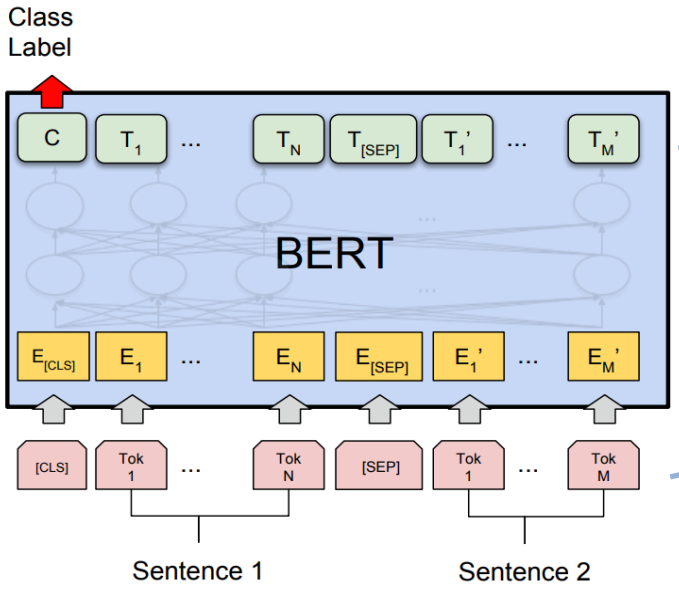
Figure from Carion et al. 2020

Existing Systems are Monolithic



Performance Challenge #2

Co-optimization: Different aspects of ML parallelization need to be considered together.



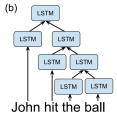
Strategy A

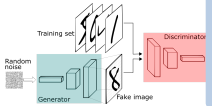
- Communication Architecture: Collective AllReduce?
- Consistency: staleness = 5?
- Gradient enc/dec: no?
- Replication/placement: replicated over all devices ?
- Partitioning: no?

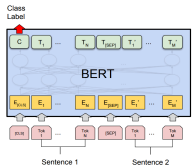
Strategy B

- Communication Architecture: PS?
- Consistency: staleness = 0?
- Gradient enc/dec: 1-bit?
- Replication/placement: GPU:0, GPU:1?
- Partitioning? [5, 1, 1]?

Summary of Challenges







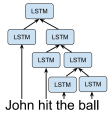
Usability #1: Matching (model, resource) with the *right* distributed strategy is hard.

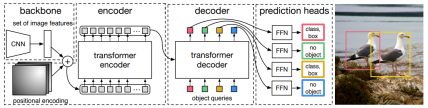
Usability #2: Distinct system implementations and interfaces -> added development overhead.

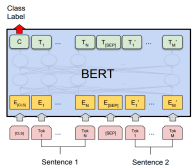
Performance #1: Increasingly more complex model structures -> opportunities sub-model strategy optimization.

Performance #2: Monolithic system design -> *lower-than-expected* performance and lack of cooptimization of multiple parallelization aspects

Goal







Communication Architecture

Consistency Models

Partitioning/ Placement

Encoding/ Decoding

Scheduling

...

Strategy auto-generation

Representation of ML parallelisms

Understanding and unifying various ML parallelisms

Thesis Statement

The structured and composable nature of nowadays ML programs allows for building optimized parallelization strategies that are *adaptive to model and cluster specifications*, *composable from different ML parallelisms*, and *auto-generated via learning-based methods*.

Research Summary

Lead
Co-author
Open source systems
Petuum-proprietary

Thesis Components

Understanding and optimize ML parallelization with *adaptive parallelisms*

Representation and Composability for ML parallelisms

Automating ML parallelization

Thesis-related Work

Communication
Poseidon [ATC'17]

Consistency Model
Impact of staleness [ICLR'19]

Scheduling
Poseidon [ATC'16]

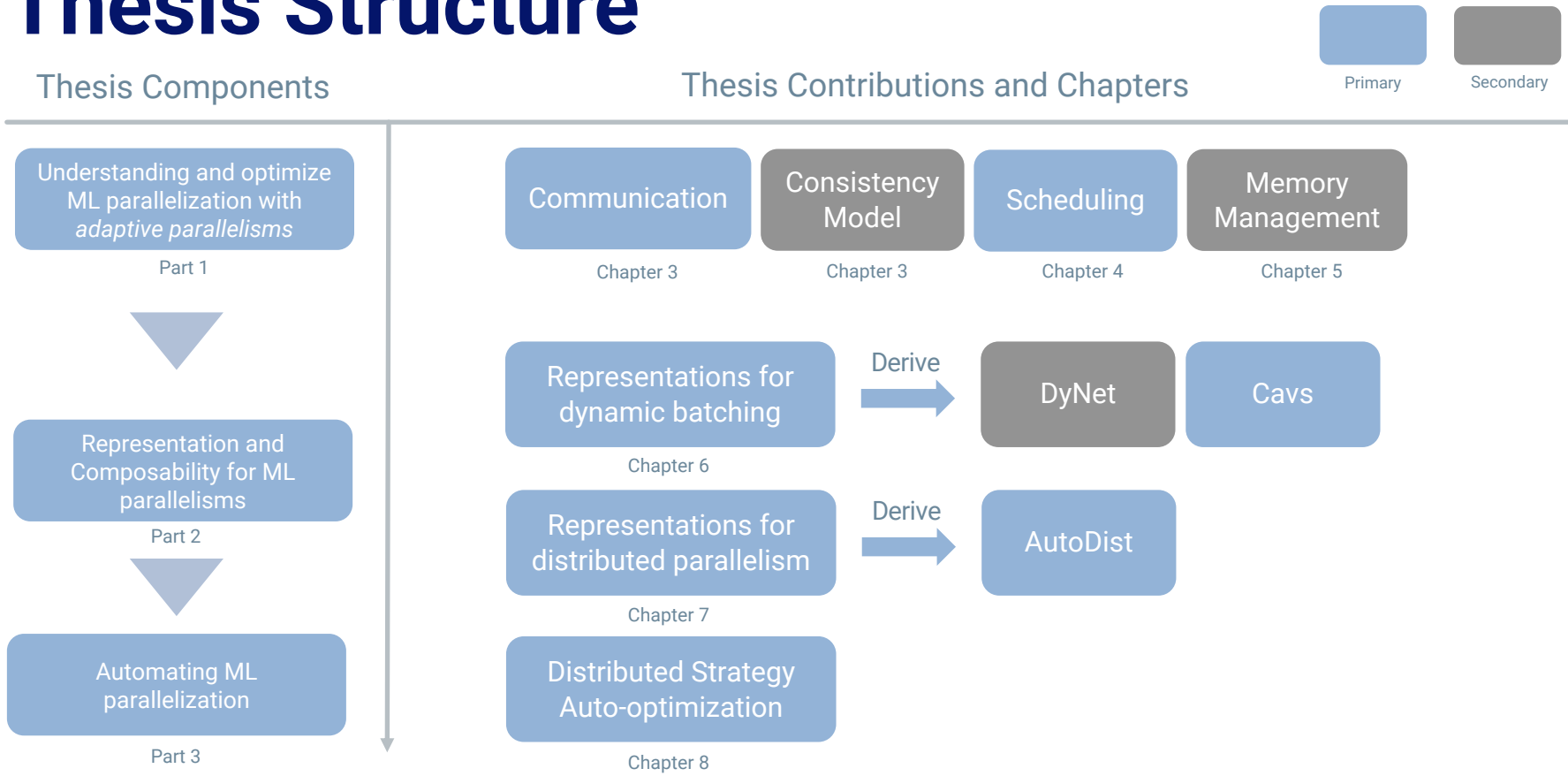
Memory Management
GeePS [Eurosys'16]

Representations for dynamic batching
Derive
DyNet
Cavs
Cavs [ATC'18]

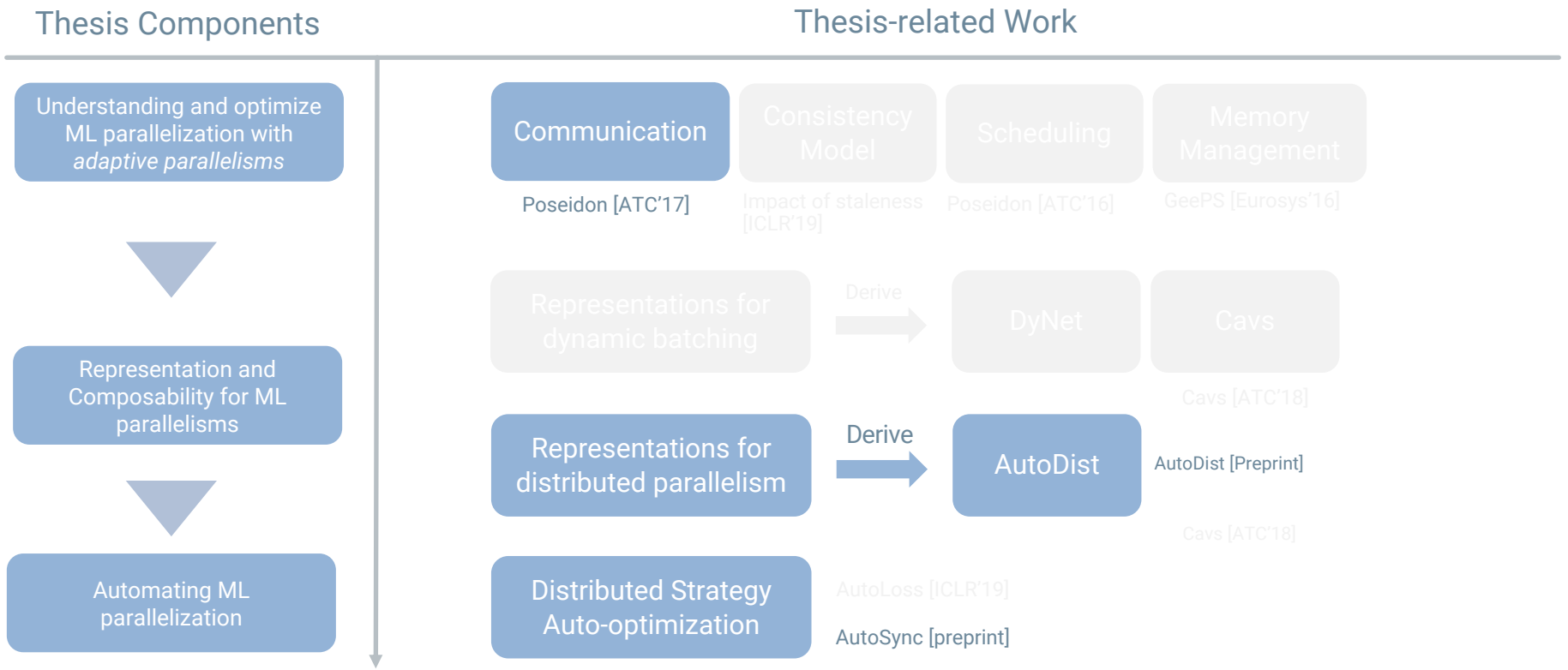
Representations for distributed parallelism
Derive
AutoDist
AutoDist [Preprint]

Distributed Strategy Auto-optimization
AutoLoss [ICLR'19]
AutoSync [preprint]

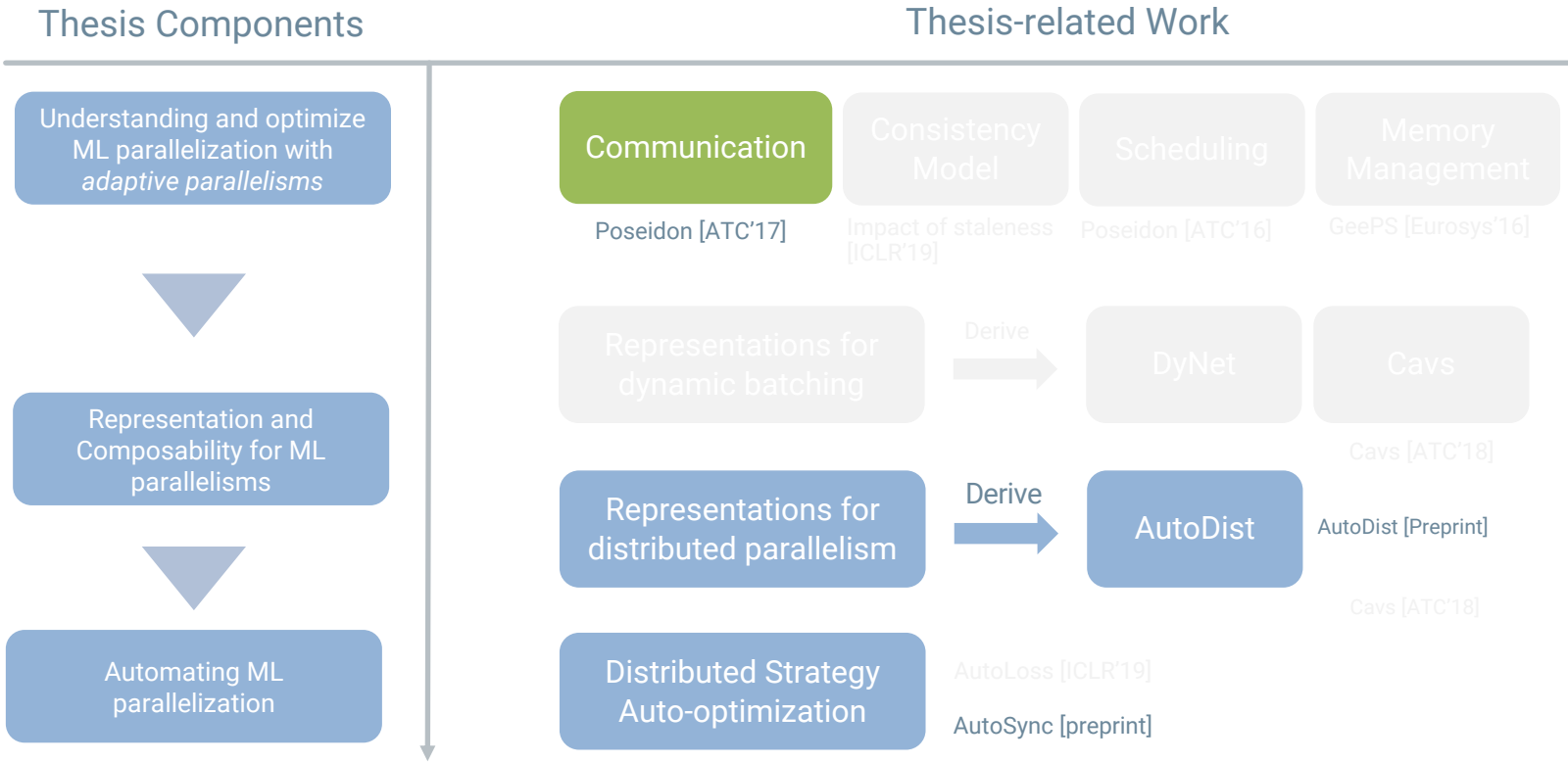
Thesis Structure



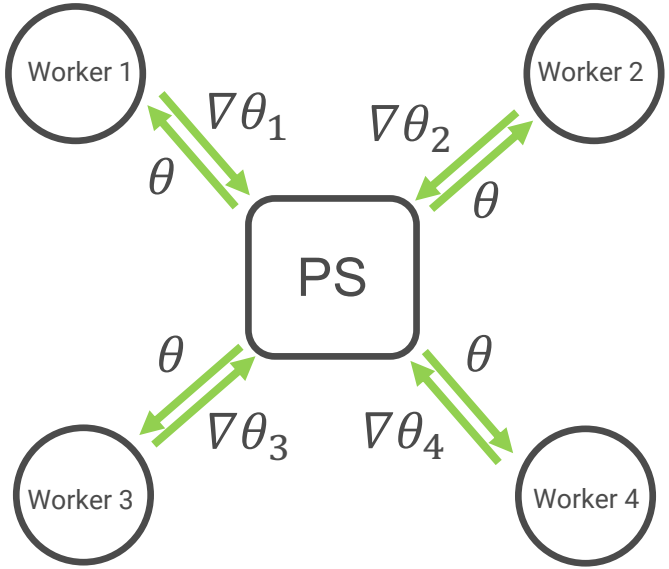
The Rest of Today's Talk



The Rest of Today's Talk



Parameter Server



In total P workers

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

PS collects, aggregates, and applies the gradients, and then broadcast the parameters back to workers

Replicated and happening locally on each worker

Partition data

Problem: Server Bottleneck

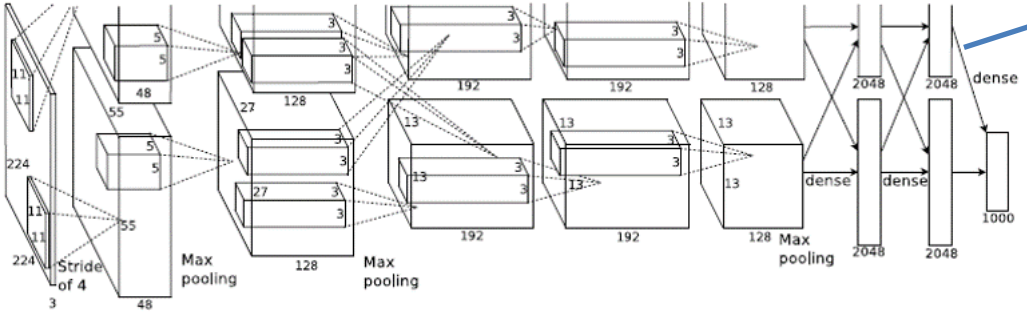
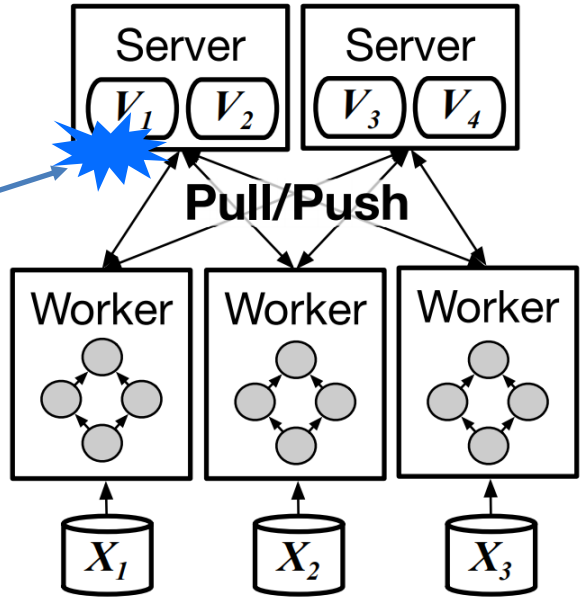
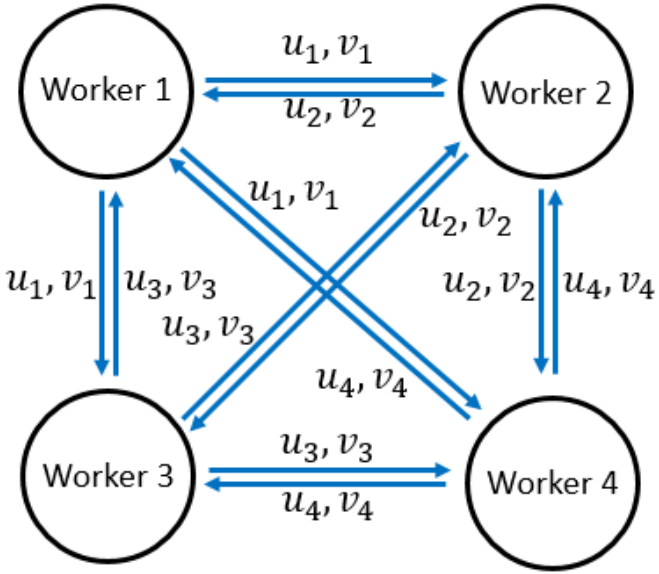


Figure from Krizhevsky et al. 2012



P2P Broadcasting?

- Idea: Send lightweight SF updates (u,v) instead of expensive full-matrix ΔW updates – sufficient factor broadcasting

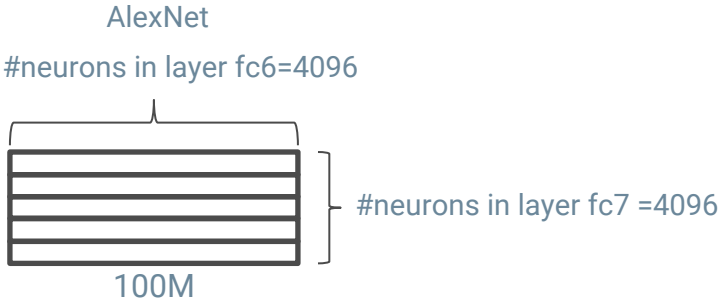


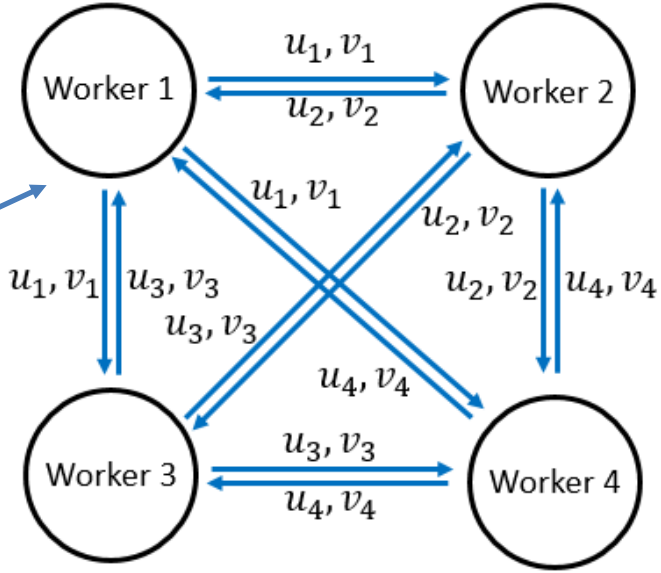
$$\min_W \frac{1}{N}\sum_{i=1}^{N} f_i(Wa_i; b_i) + h(W)$$

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

Full parameter matrix update ΔW can be computed as **outer product of two vectors** uv^T (called sufficient factors)

Problem: Quadratic Overhead





	Size of one message	Number of messages	Network Traffic
P2P SV-Transfer	$O(J+K)$	$O(P^2)$	$O((J+K)P^2)$
Parameter Server	$O(JK)$	$O(P)$	$O(JKP)$

Ring AllReduce

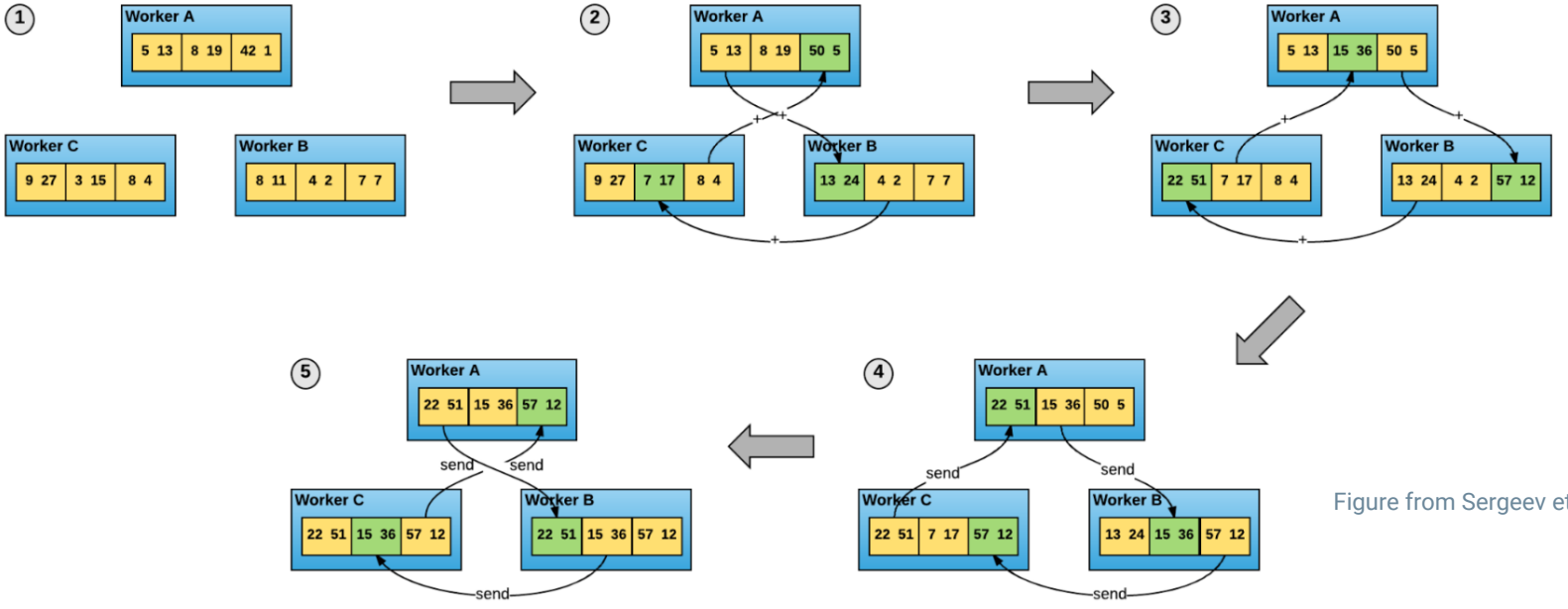
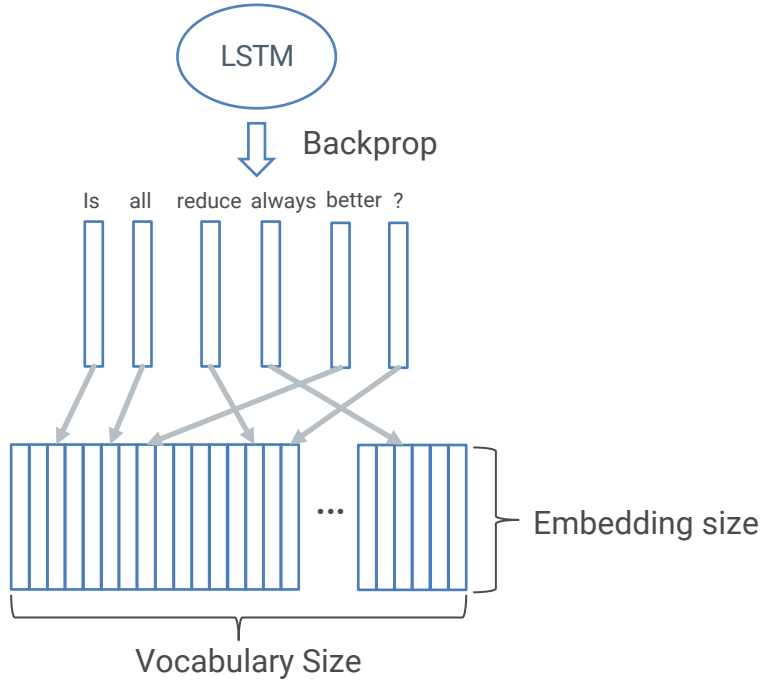


Figure from Sergeev et al. 2018

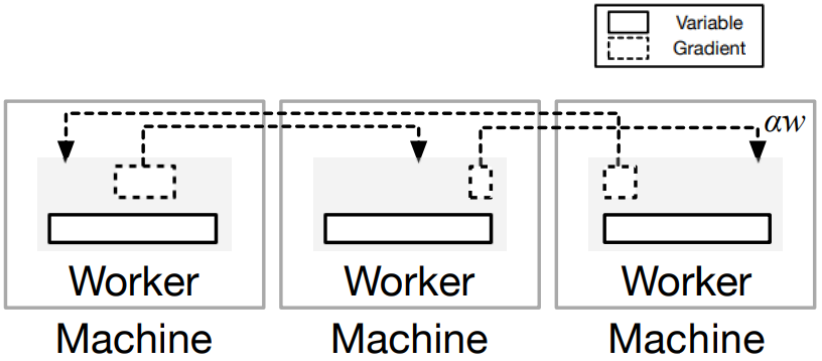
Problem: Sparse Gradients

Sparse Gradients in NLP Models



AllReduce

Figure from Kim et al. 2018



Adaptive Communication

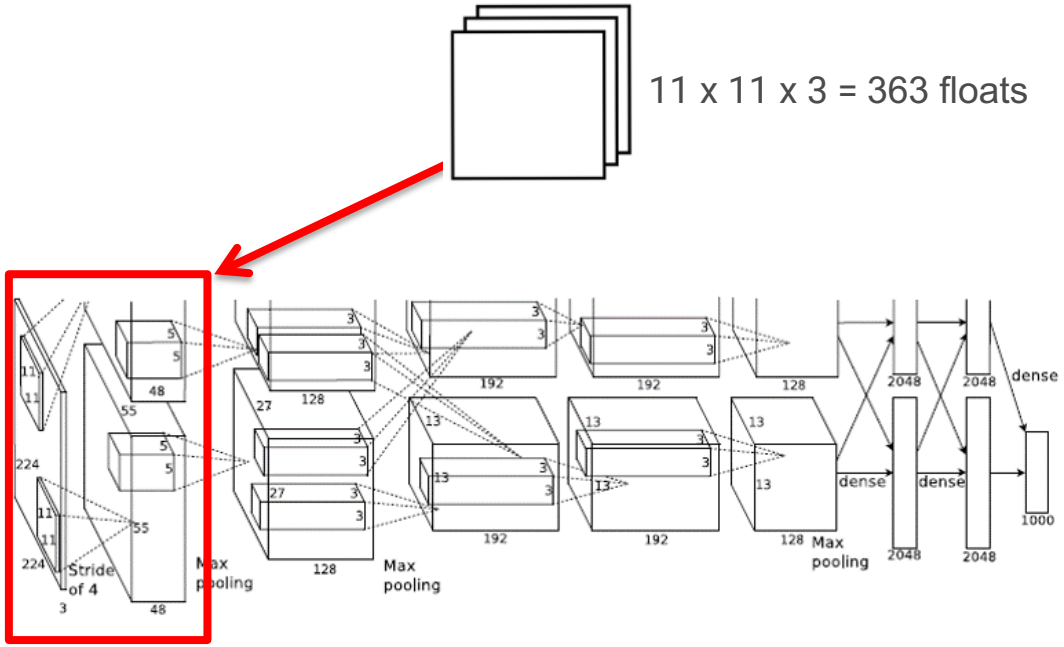


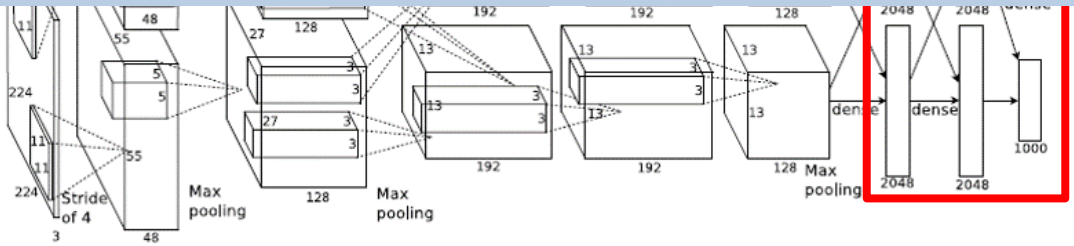
Figure from Krizhevsky et al. 2012

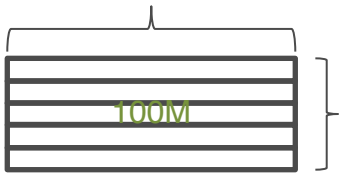
Adaptive Communication

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

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

2x – 8x throughput scalability improvement on a variety of CNNs and clusters configurations with limited bandwidth (1 – 40Gbps).





k (AlexNet)
r fc6=4096

#neurons in layer fc7 =4096

Figure from Krizhevsky et al. 2012

Adaptive Communication

Sparse Gradients in NLP Models

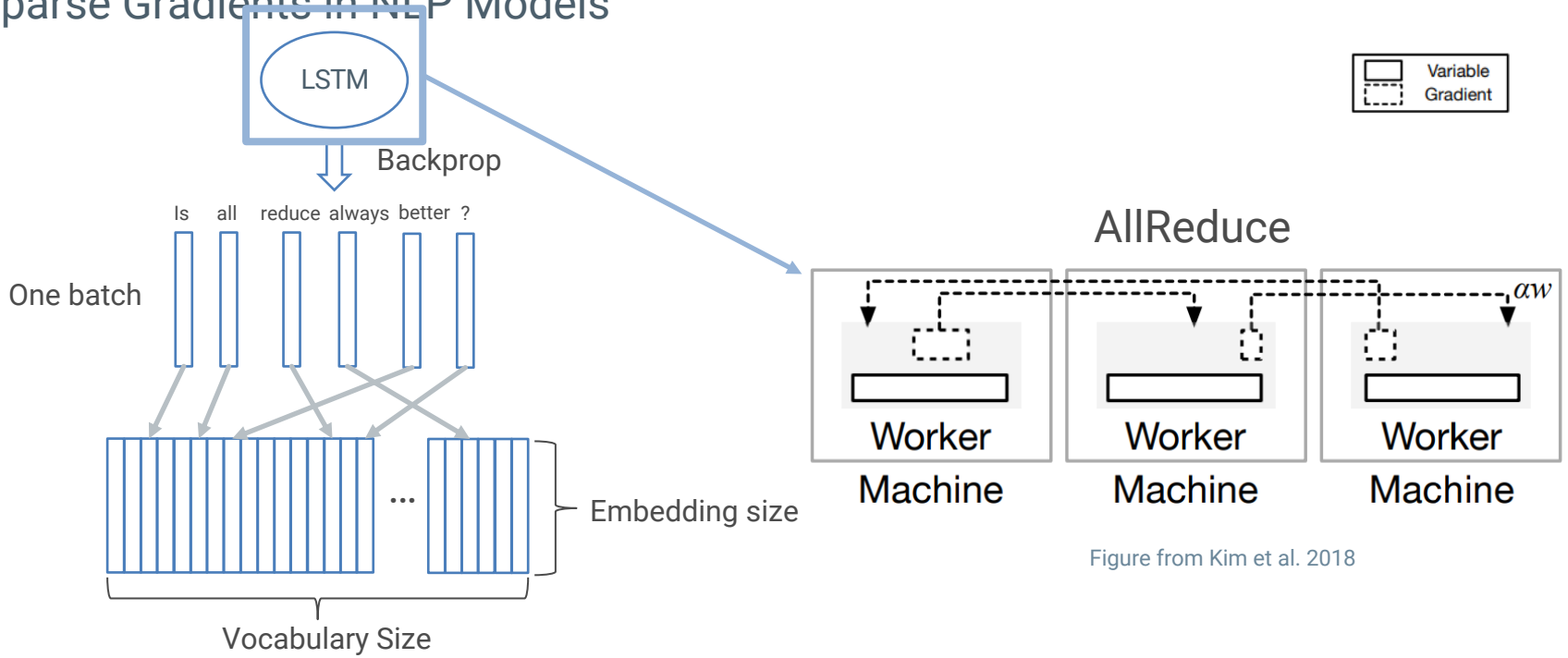


Figure from Kim et al. 2018

Adaptive Communication: Parallax

Sparse Gradients in NLP Models

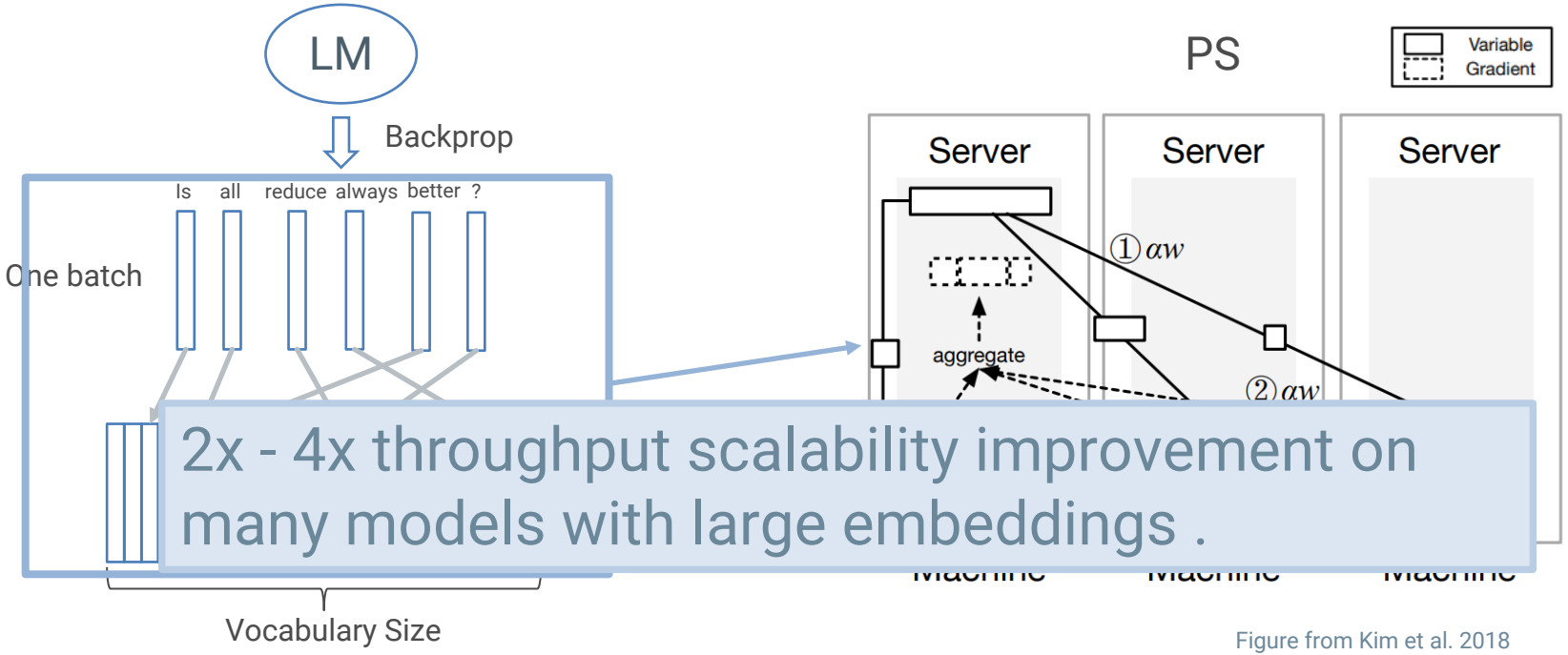
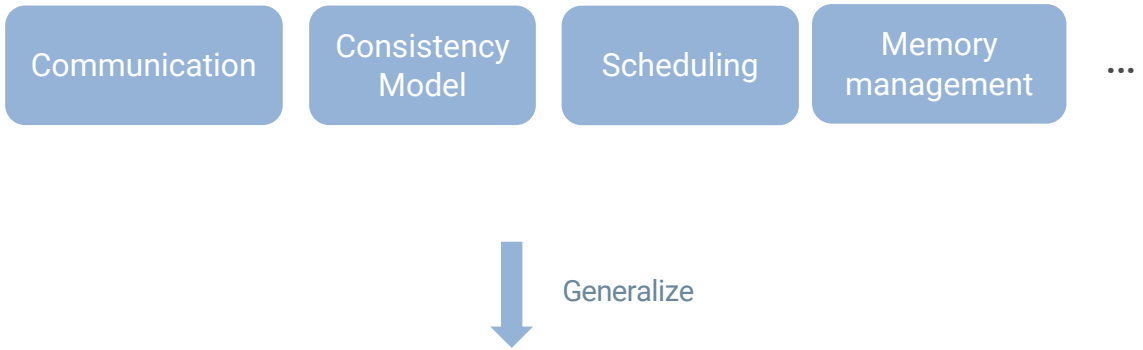


Figure from Kim et al. 2018

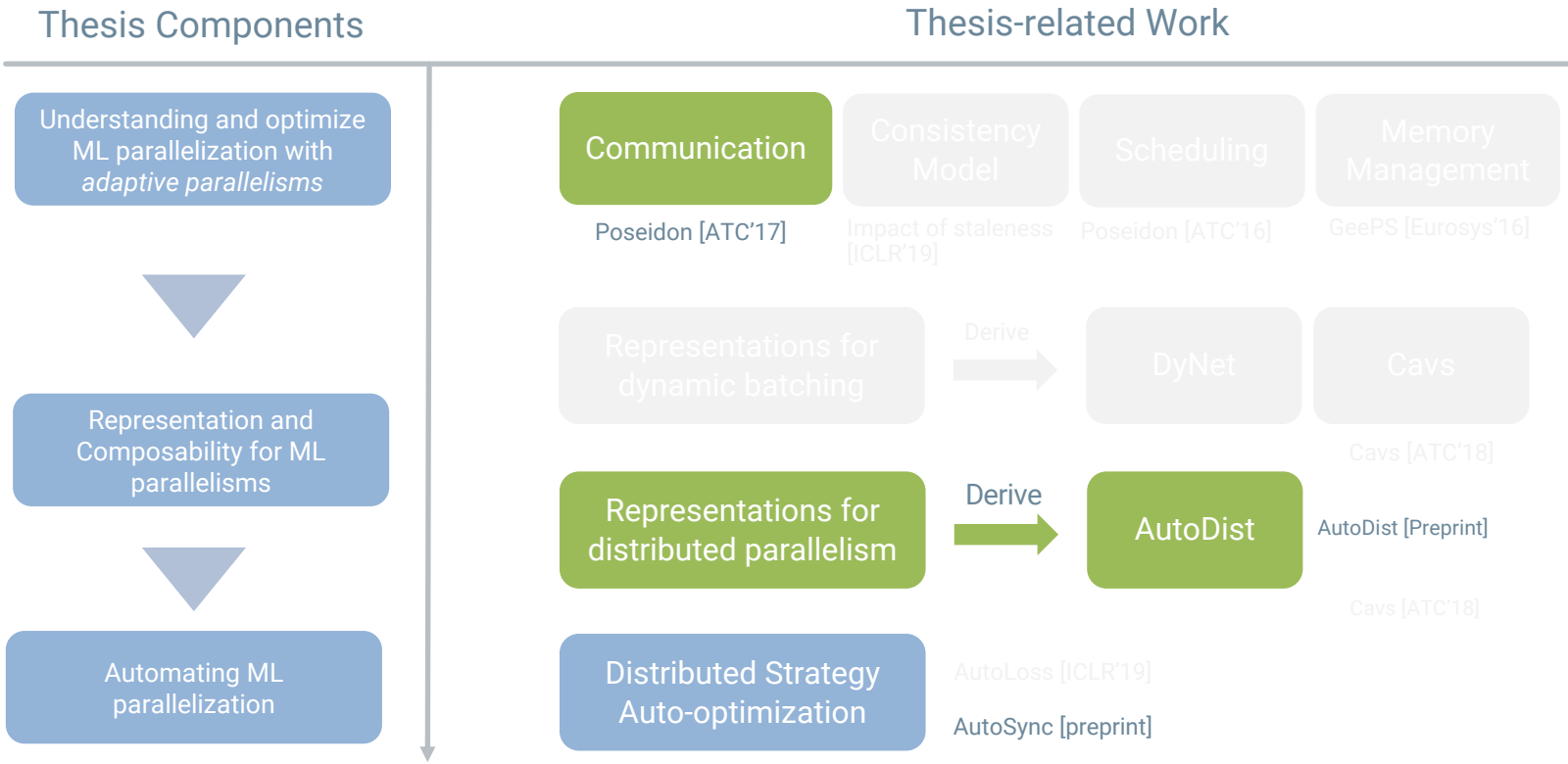
Part I: Main Messages

- Strategies that adapt to specific model properties or resource configurations yield improved performance.
- Adaptiveness might be exploited in many aspects of distributed ML.

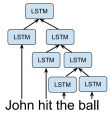
Instantiations of adaptive parallelization strategies

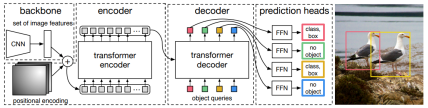


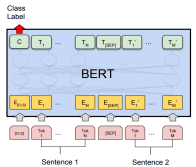
The Rest of Today's Talk



Goal







Communication Architecture

Consistency Models

Partitioning/ Placement

Encoding/ Decoding

Scheduling

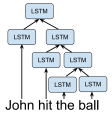
...

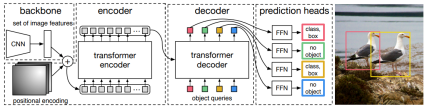
Strategy auto-generation

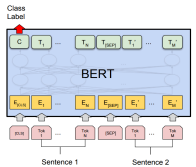
Representation of ML parallelisms

Understanding and unifying various ML parallelisms

An Optimization Perspective





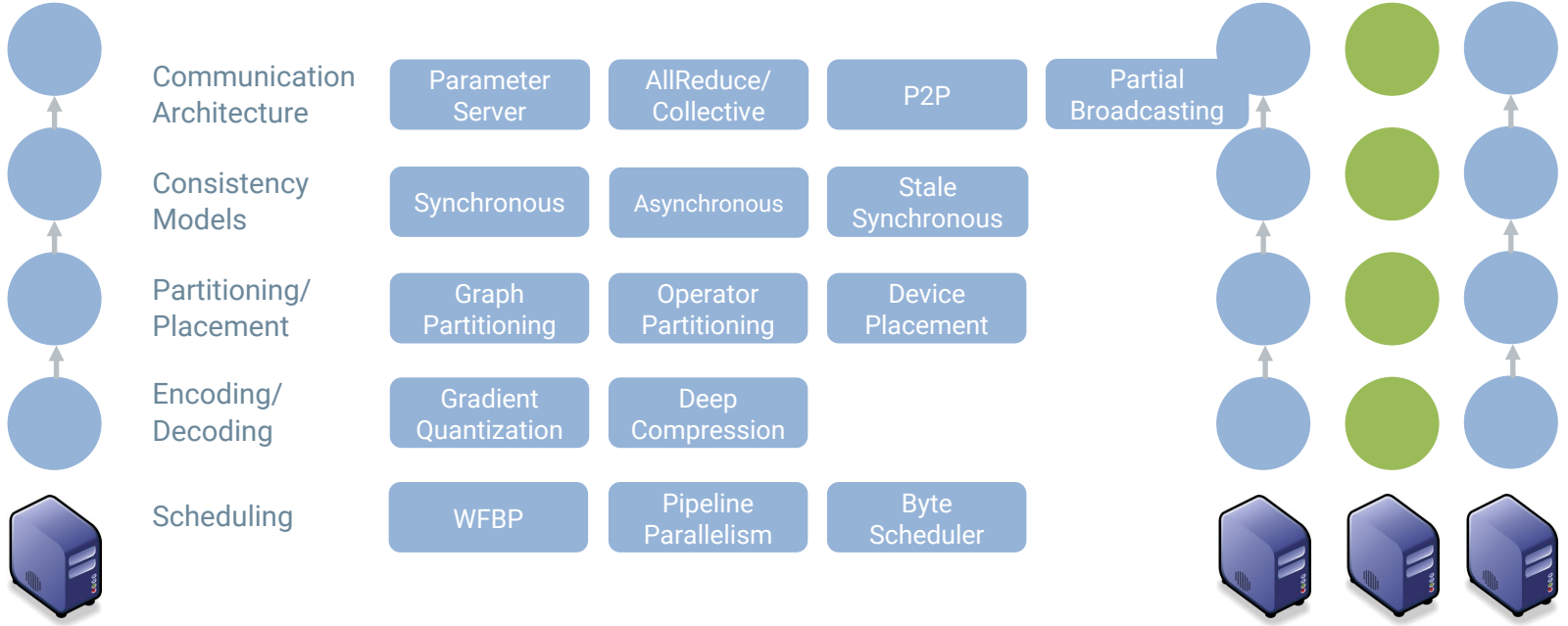


Maximize *Performance*(
strategy
Model & Configurations,
Cluster Topology & Settings)

System throughput/convergence/cloud cost

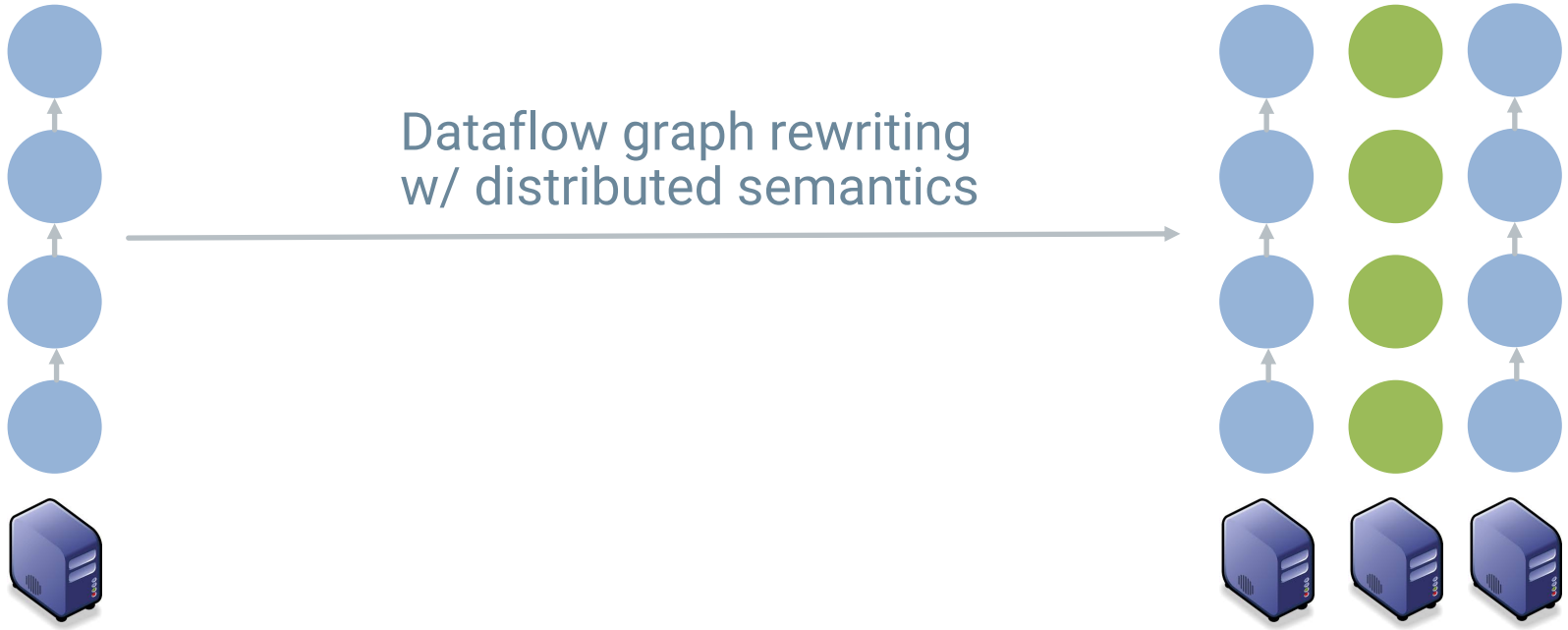
Dataflow graphs

Strategy Representation: Overview

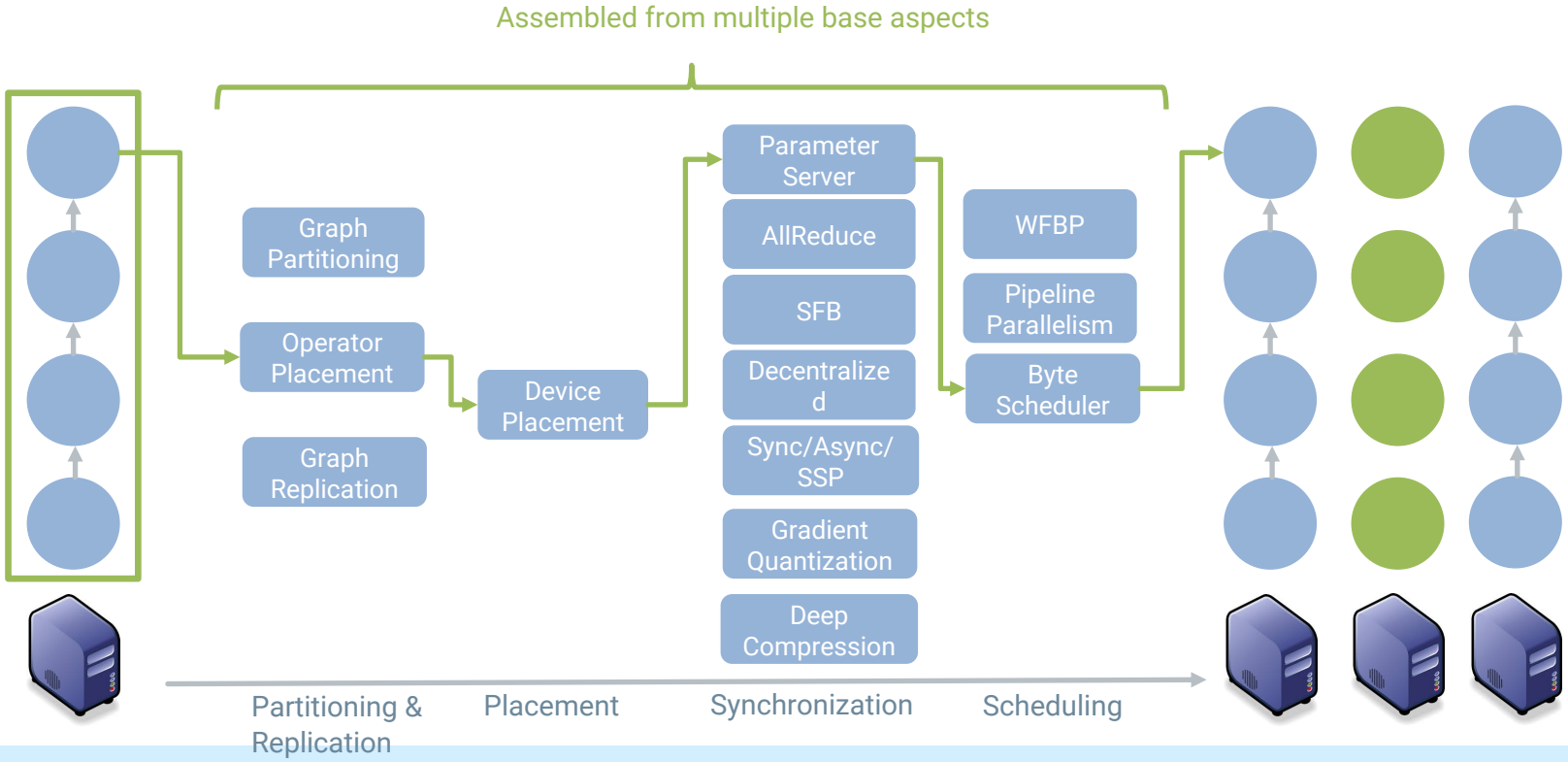


Strategy Representation: Overview

- Idea: dataflow graph rewriting

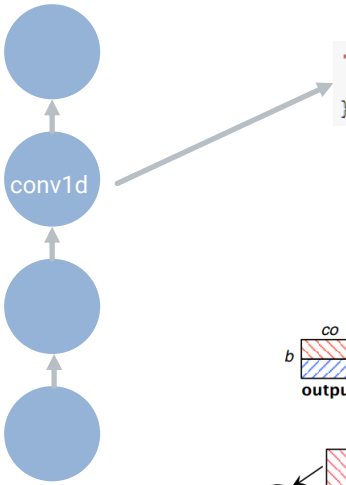


Strategy Representation: Overview



Semantic: Partitioning and Placement

Node Partitioning




```
'conv1d': {
    "partitioning": [1, 2, 0]
}
```

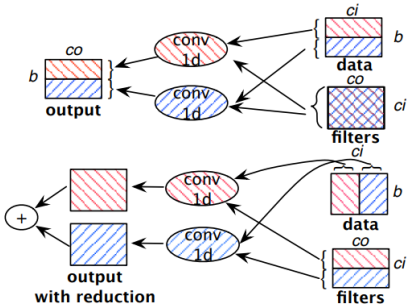
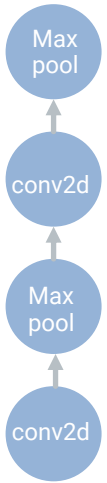


Figure from Wang et al.

Shazeer et al. 2018
Jia, et al. 2019
Wang, et al. 2019
Harlap, etc.

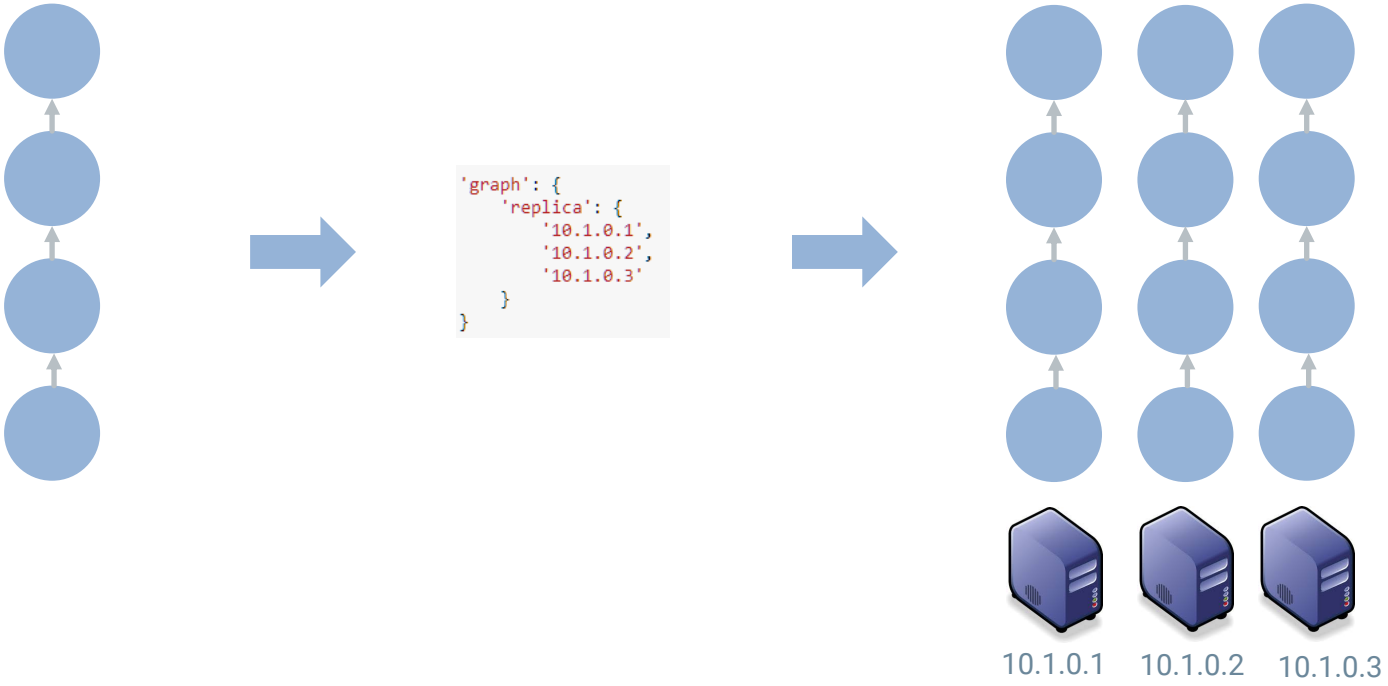
Placement




```
'conv2d': {
    'device': '10.1.0.1:GPU0'
}
```

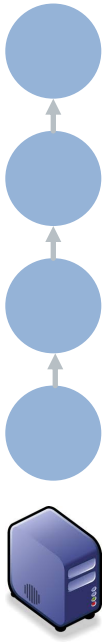
Mirhoseini et al. 2017

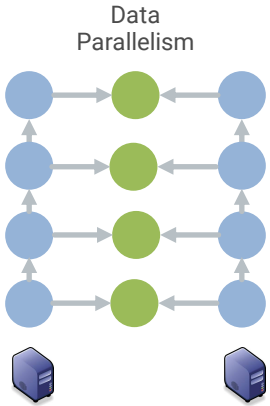
Semantic: Replication

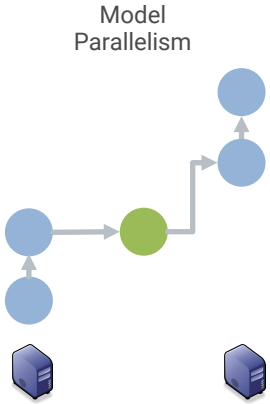


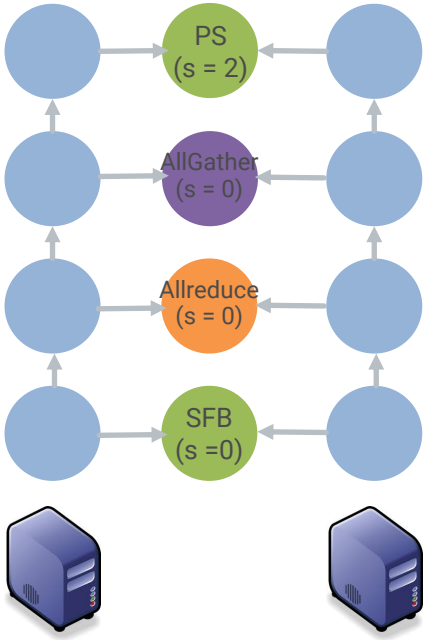
Semantic: Synchronization

Examples

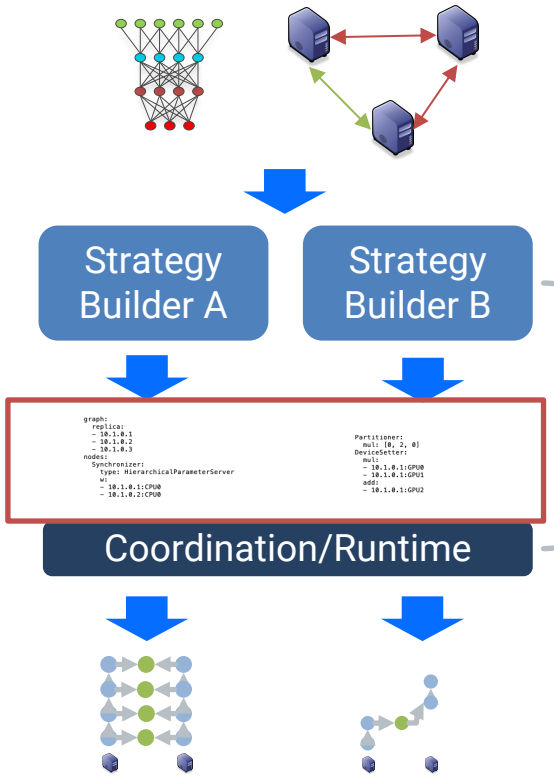



Data
Parallelism


Model
Parallelism


PS
(s = 2)
AllGather
(s = 0)
Allreduce
(s = 0)
SFB
(s =0)

Architecture



AutoDist

Compilation& Strategy

"Kernels"

Coordination

Frameworks

User Interface

StrategyBuilder

Representation

Partitioner

Device Setter

Synchronizer

Replicator

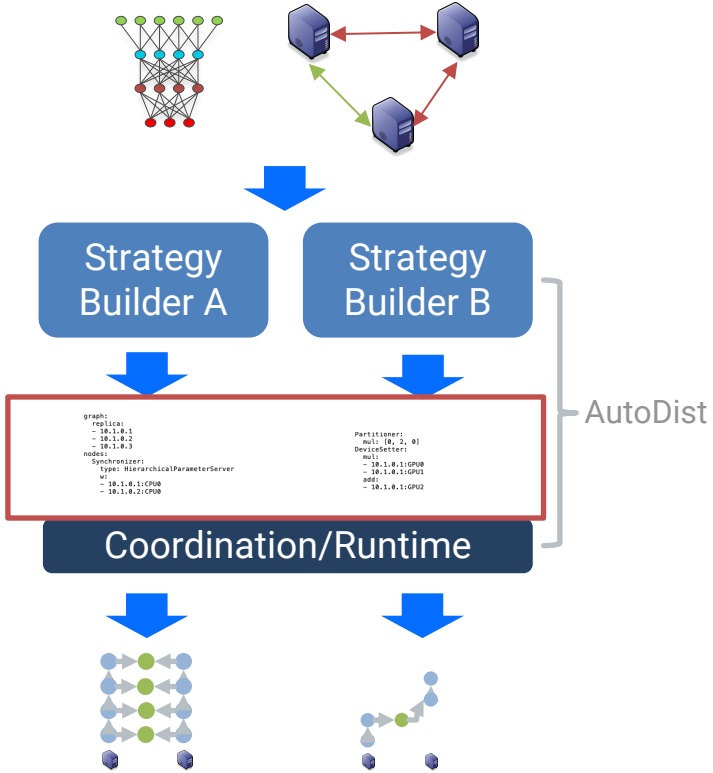
Encoder

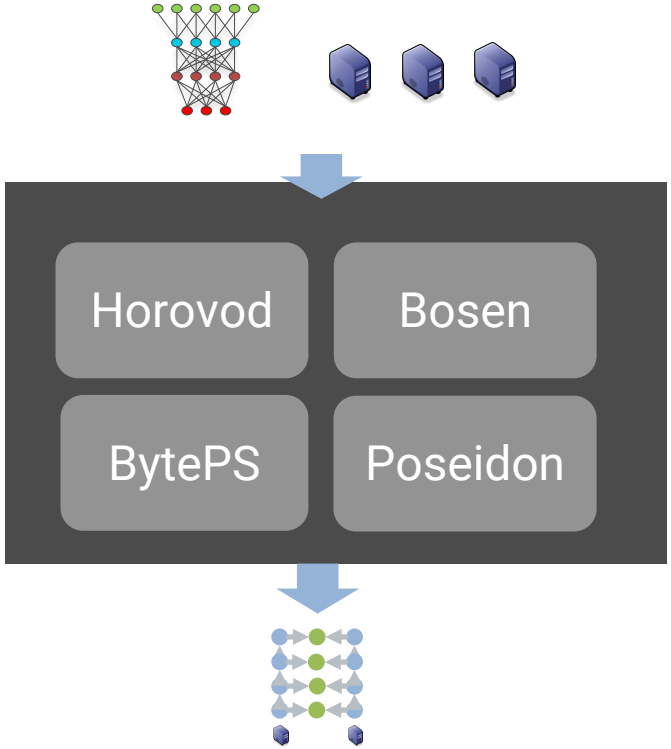
...

Coordinator & Runner

TensorFlow / PyTorch

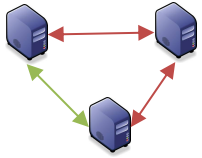
Contrast: Workflows



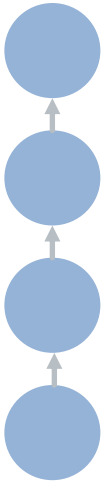


Usage: Strategy

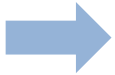
Strategy: model- and resource-dependent expression instructing how to distribute the model.



Cluster spec R



Model graph G



```
graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3
nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    w:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
    b:
    - 10.1.0.1:CPU0
  Partitioner:
    mul: [0, 2, 0]
  DeviceSetter:
    mul:
    - 10.1.0.1:GPU0
    - 10.1.0.1:GPU1
    add:
    - 10.1.0.1:GPU2
```

Strategy S

Usage: Examples

```
train_dataset = build_dataset()
train_iterator = tf.compat.v1.data.make_one_shot_iterator(train_dataset).get_next()

model = build_model()
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.SGD()

def train_step(inputs):
    x, y = inputs
    y_hat = model(x, training=True)
    loss = loss_fn(y, y_hat)
    all_vars = []
    for v in model.trainable_variables:
        all_vars.append(v)
    grads = tf.gradients(loss, all_vars)
    update = optimizer.apply_gradients(zip(grads, all_vars))
    return loss, update

fetches = train_step(train_iterator)
for _ in range(min(10, len(train_images) // BATCH_SIZE * EPOCHS)):
    loss, _ = sess.run(fetches)
    print("train_loss: {loss}")
```

Import

```
from autodist import AutoDist
from autodist.strategy import mystrategy
```

Choose strategy and construct AutoDist object

```
filepath = os.path.join(os.path.dirname(__file__), 'resource_spec.yml')
autodist = AutoDist(resource_spec_file=filepath, strategy_builder=mystrategy)
```

Scope your code

```
with autodist.scope():
    train_dataset = build_dataset()
    train_iterator = tf.compat.v1.data.make_one_shot_iterator(train_dataset).get_next()

    model = build_model()
    loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()
    optimizer = tf.keras.optimizers.SGD()

    def train_step(inputs):
        x, y = inputs
        y_hat = model(x, training=True)
        loss = loss_fn(y, y_hat)
        all_vars = []
        for v in model.trainable_variables:
            all_vars.append(v)
        grads = tf.gradients(loss, all_vars)
        update = optimizer.apply_gradients(zip(grads, all_vars))

        return loss, update

    fetches = train_step(train_iterator)
    sess = autodist.create_distributed_session()
    for _ in range(min(10, len(train_images) // BATCH_SIZE * EPOCHS)):
        loss, _ = sess.run(fetches)
        print(f"train_loss: {loss}")
```

Use AutoDist distributed session instead of native session

Usage: Strategy Builder

Strategy builder: a strategy generation function with expert knowledge.

```python
# Parallax strategy
# Heuristics:
#   use PS for sparse variables,
#   use AllReduce for dense variables
for grad in grads:
    if isinstance(grad, tf.Tensor):
        strategy[grad]['syncer'] = 'AllReduce'
    if isinstance(grad, tf.SparseTensor):
        strategy[grad]['syncer'] = 'PS'
```

Builder

tegyBui

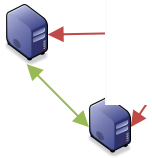
uceBuil

PoseidonBuilder

```python
# Poseidon strategy
# Heuristics:
#   Reduce the grad of FC layers to vectors,
#   and then broadcast vectors
for grad in grads:
    if np.linalg.matrix_rank(grad) == 1:
        strategy[grad]['syncer'] = 'AllReduce'
        strategy[grad]['compressor'] = 'SFB'
    else:
        strategy[grad]['syncer'] = 'PS'
        strategy[grad]['compressor'] = None
```

imeterServer

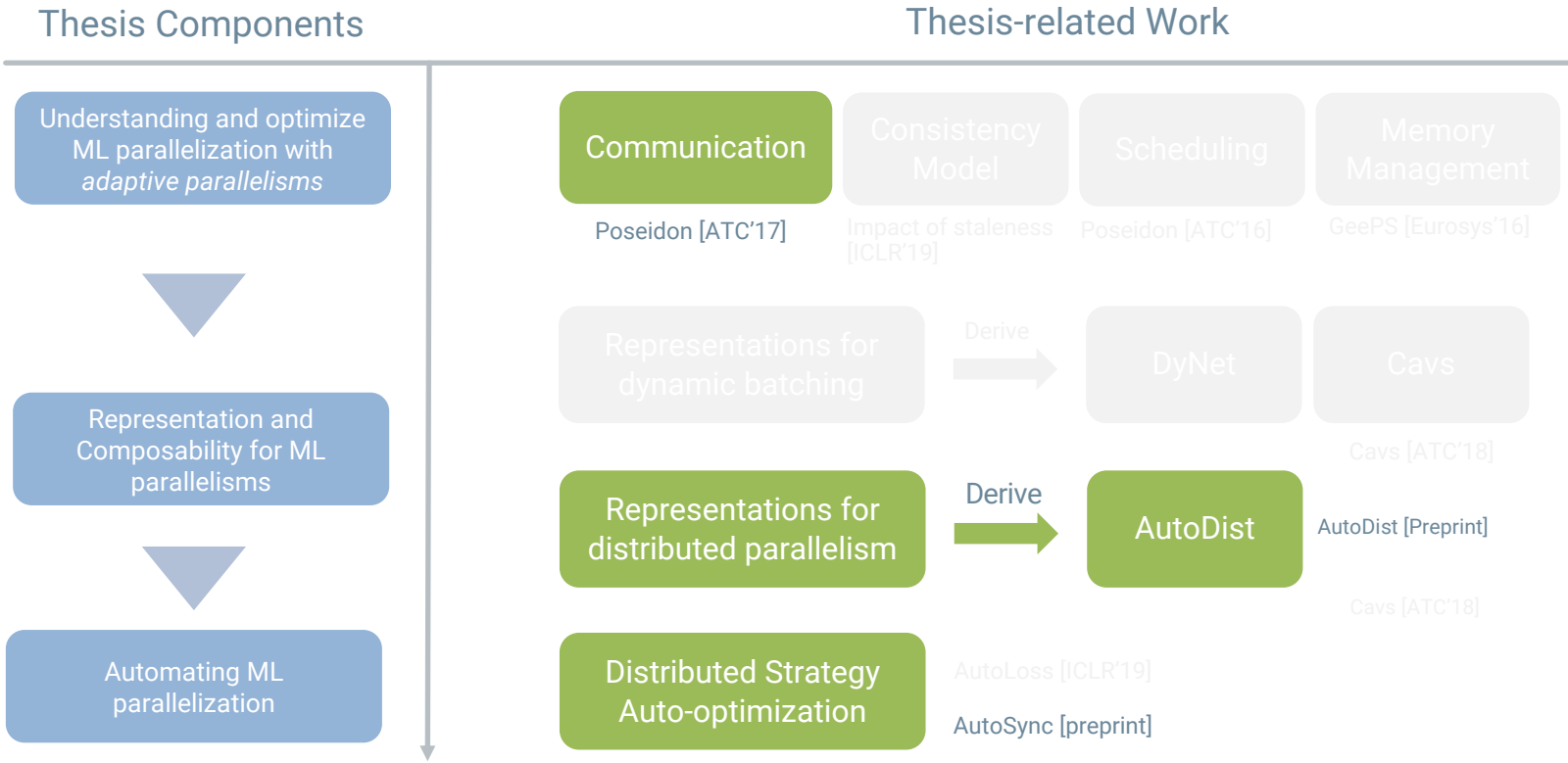
– 10.1.0.1:CPU0



Resource spec

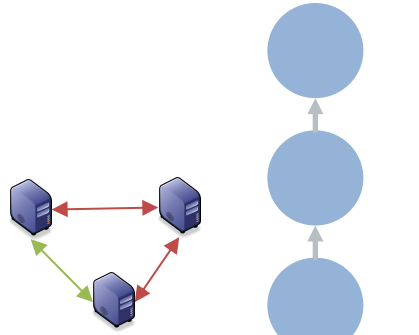
Prebuilt strategy builders report *matched* or *slightly better* performance compared to their corresponded specialized systems (TF runtime).

The Rest of Today's Talk



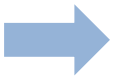
Usage: Strategy Builder

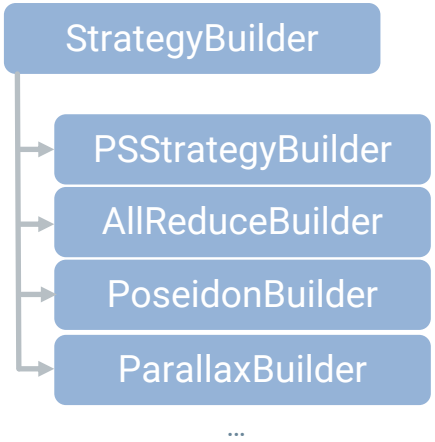
Strategy builder: a strategy generation function with expert knowledge.

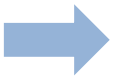


Resource spec R

Model graph G





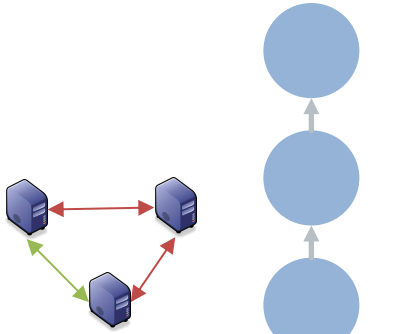


```
graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3
nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    w:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
    b:
    - 10.1.0.1:CPU0
  Partitioner:
    mul: [0, 2, 0]
  DeviceSetter:
    mul:
    - 10.1.0.1:GPU0
    - 10.1.0.1:GPU1
    add:
    - 10.1.0.1:GPU2
```

Strategy S

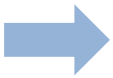
Problem: Design Auto StrategyBuilders

Strategy builder: a strategy generation function auto-optimizes over the strategy space.

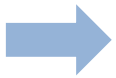


Resource spec R

Model graph G



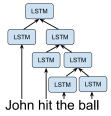
Auto StrategyBuilder

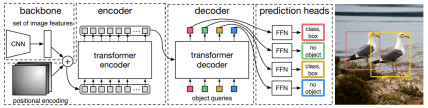


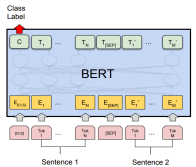
```
graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3
nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    w:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
    b:
    - 10.1.0.1:CPU0
  Partitioner:
    mul: [0, 2, 0]
  DeviceSetter:
    mul:
    - 10.1.0.1:GPU0
    - 10.1.0.1:GPU1
    add:
    - 10.1.0.1:GPU2
```

Strategy S

An Optimization Perspective







System throughput/convergence/cloud cost

Maximize *Performance*(
strategy
Model & Configurations,
Cluster Topology & Settings)

Dataflow graphs

```
graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3
nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    w:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
    b:
    - 10.1.0.1:CPU0
  Partitioner:
    mul: [0, 2, 0]
  DeviceSetter:
    mul:
    - 10.1.0.1:GPU0
    - 10.1.0.1:GPU1
    add:
    - 10.1.0.1:GPU2
```

A Simplified Form

$$\underset{\text{strategy}}{\text{Maximize}} \; \textit{Performance}(\text{Model \& Configurations, Cluster Topology \& Settings})$$

$$\max_{s} t(D, R, s)$$
$$s.t.\ s \in S_{sync}$$

Solving the Problem

Non-continuity

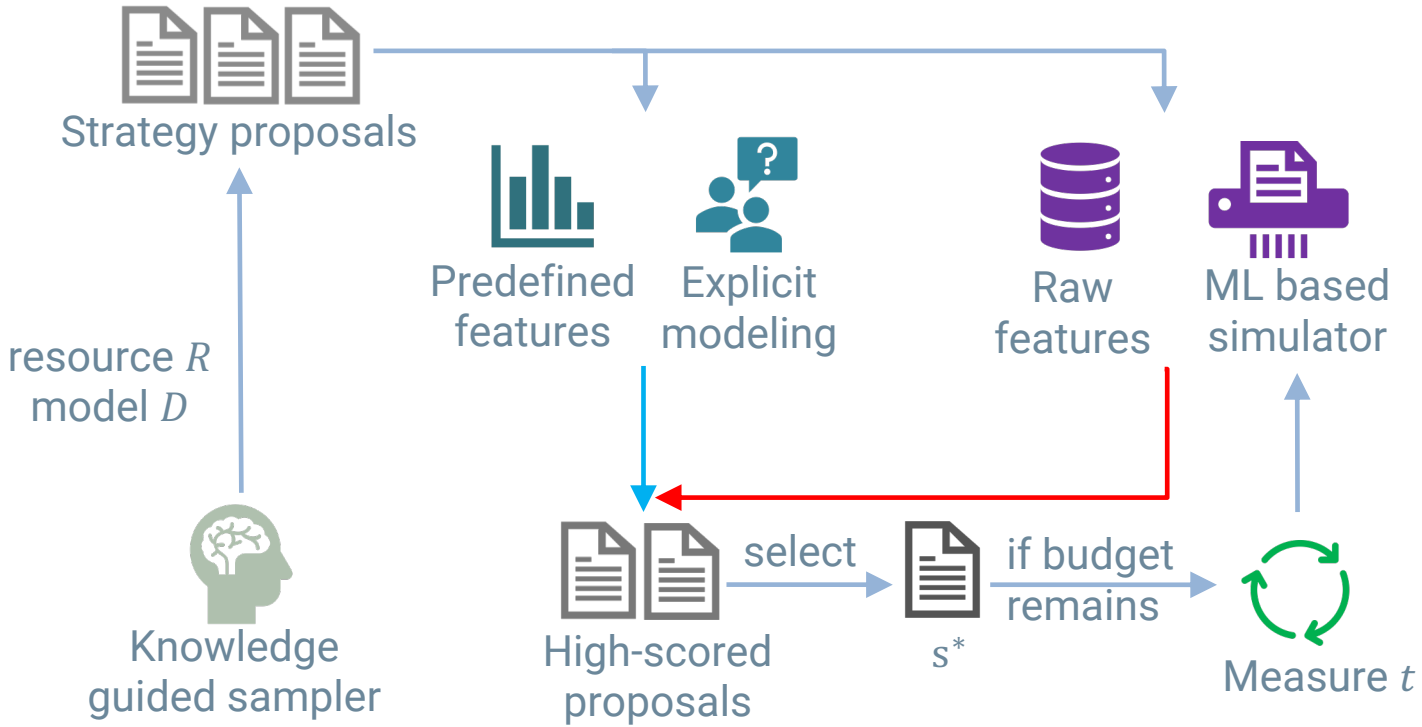
Transparent communication

$$\max_{s} t(D, R, s)$$
$$s.t.\ s \in S_{sync}$$

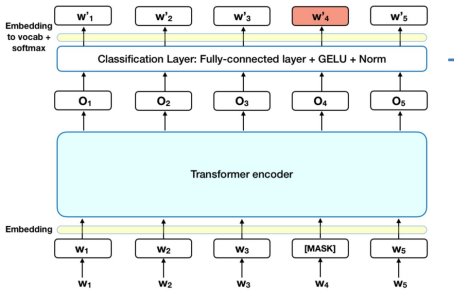
Measuring t is not cheap

Domain-specific prior knowledge

Optimizer


Strategy proposals
Predefined features
Explicit modeling
Raw features
ML based simulator
resource R
model D
Knowledge guided sampler
High-scored proposals
select
s^*
if budget remains
Measure t

Communication Modeling: Features



For each variable +

Raw features:

[variable size, compression type, reduction structure, placement, bandwidth, #replicas]

m_i c_i $\mathbb{I}_p$ $\mathbb{I}_d(j,k)$ $b_{j,k}$ $r_{i,k}^{\mathbb{I}_p}$

$$T_{\text{server}}^{PS}(v_i) = \underbrace{\sum_{k=1}^{w_i} \mathbb{I}_d(j,k) \cdot \frac{c_i(m_i)}{b_{j,k}} \cdot r_{i,k}^{\mathbb{I}_p}}_{\text{network transfer}} + \underbrace{\sum_{k=1}^{w_i} \mathbb{I}_d(j,k) \cdot r_{i,k}^{\mathbb{I}_p} \cdot \phi}_{\text{network overhead}} + \underbrace{\delta}_{\text{GPU kernel latency}},$$

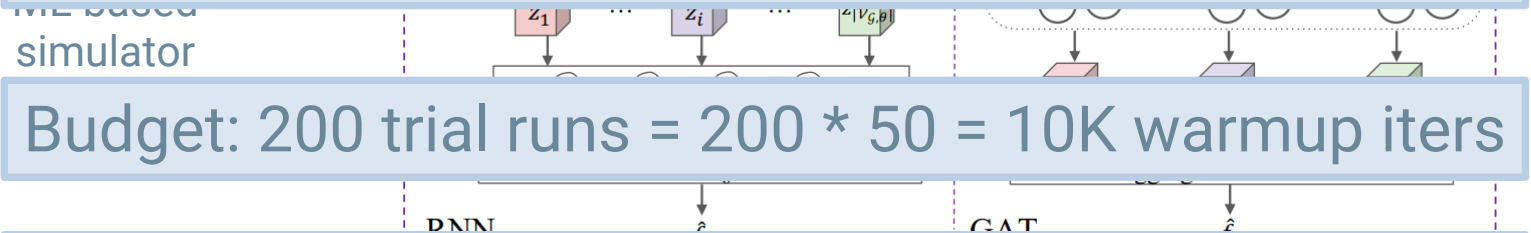
$$T_{sync}^{CC}(v_i) = \underbrace{\mathbb{I}_1 \frac{2(w_i-1)c_i(m_i)}{w_i b_m} + \mathbb{I}_2 \frac{(w_i-1)c_i(m_i)}{w_i b_m} + \mathbb{I}_3 \frac{c_i(m_i)}{b_m}}_{\text{network transfer}} + w_i \cdot \phi + \delta,$$

Predefined features:

[network transfer, network overhead, GPU memory kernel latency]

ML Simulator

Find strategies 1.1 – 1.6x faster than specialized systems, such as BytePS, Horovod, or handwritten strategy builders, on CNNs, transformers, RNNs, and recommendation models (TF runtime).



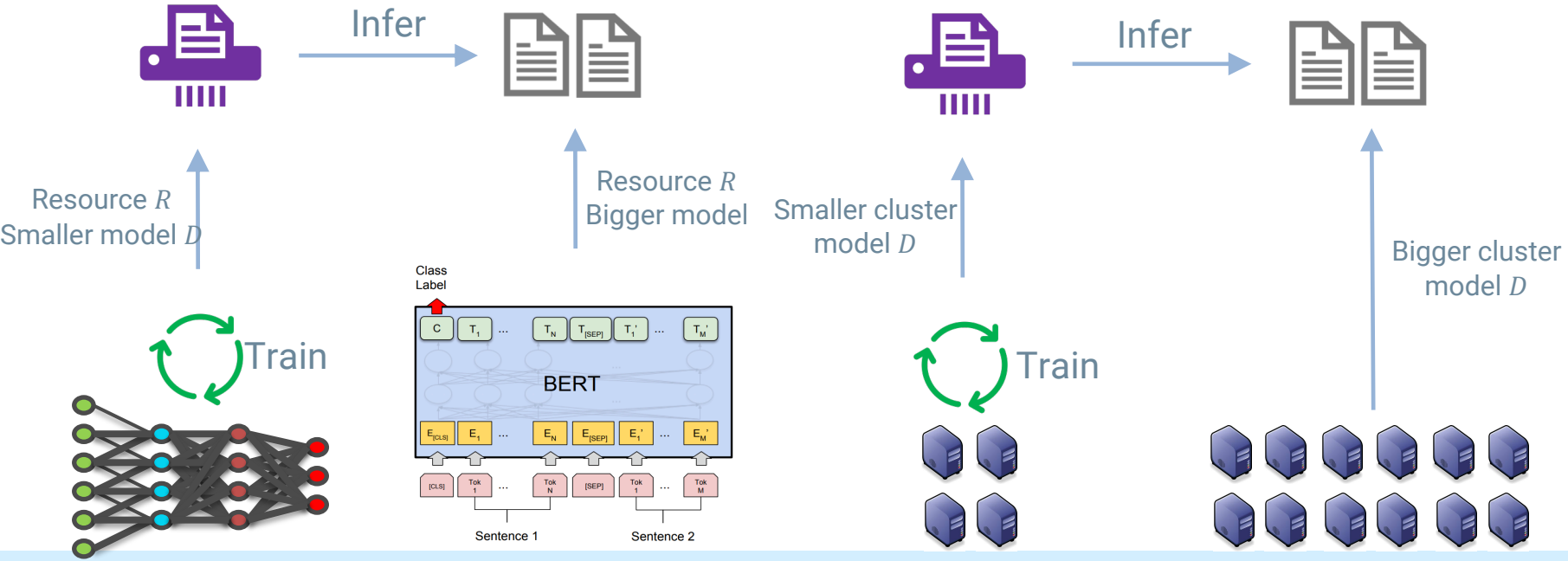
Budget: 200 trial runs = 200 * 50 = 10K warmup iters

Trained simulators exhibit certain transferability.

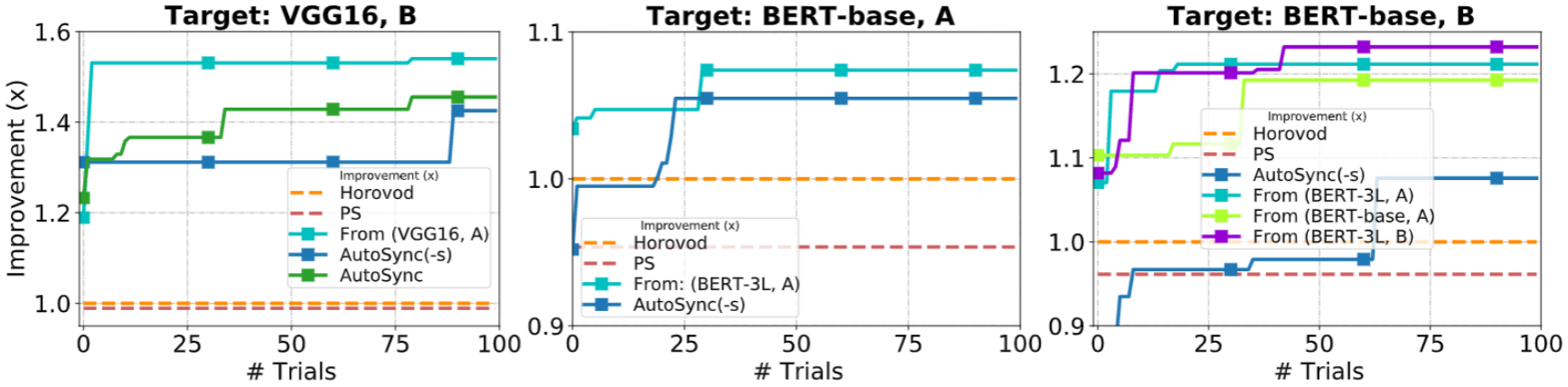
Transfer Trained Simulators

Case #1

Case #2



Transfer Trained Simulators



The Rest of Today's Talk

Research Roadmap

Understanding Individual aspects and optimizations with adaptive parallelisms

Unified Representation of aspects of ML parallelisms

Auto-optimization for Distributed ML

Thesis-related Work

Communication — Poseidon [ATC'17]

Consistency Model — Impact of staleness [ICLR'19]

Scheduling — Poseidon [ATC'16]

Memory Management — GeePS [Eurosys'16]

Resource Allocation — Pollux [Preprint]

Representations for distributed parallelism — Derive → AutoDist — AutoDist [Preprint]

Representations for dynamic batching — Derive → DyNet, Cavs — Cavs [ATC'18]

Distributed Strategy Auto-optimization — AutoLoss [ICLR'19], AutoSync [preprint]

New Takeaways (after Thesis Proposal)

- Offer a rich set of de facto distributed strategies in one system
  - Performance matches corresponded specialized systems
- Unified and simpler interface for distributed ML
- Develop new distributed strategies using Python
- Offer preliminary strategy auto-optimization
  - Performance 1.1x – 1.6x faster than specialized systems
- It was open sourced: https://github.com/petuum/autodist (please try and start)

Summary: Distributed ML Training

Eager, prototyping, development

De facto distributed strategy (a.k.a. strategy builders)

Declarative, massive training, production

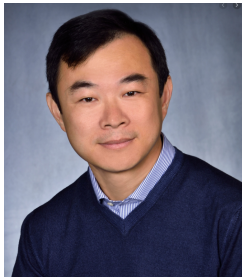
Model/Program IR (active WIP)

Composable parallelism and strategy auto-optimization

Cluster/Resource scheduling, coordination, management, etc.

Acknowledgement

Advisor



Committee members

Collaborators

Collaborators (6 more!)

Thank you

Architecture: “Kernels”

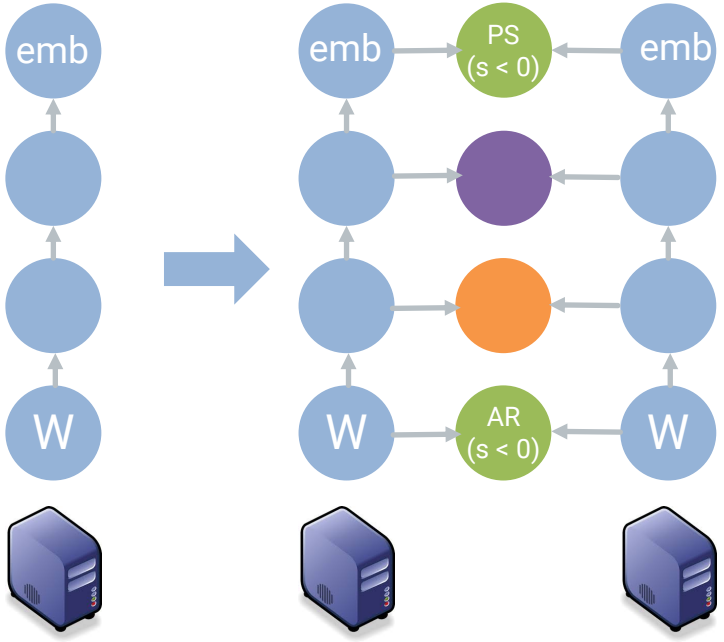
Kernels are implementations of atomic graph transformation operations.

```
'synchronizer': {
    'scheme': 'PS',
    'config': {
        'reduction_destinations': {
            '10.1.0.1:CPU',
            '10.1.0.2:CPU'
        },
        'local_replication': True,
        'staleness': <3,
        'compressor': 'PowerSGD'
    }
}
```

Expression of a PS synchronizer

(generated by strategy builder)

Invoke execution

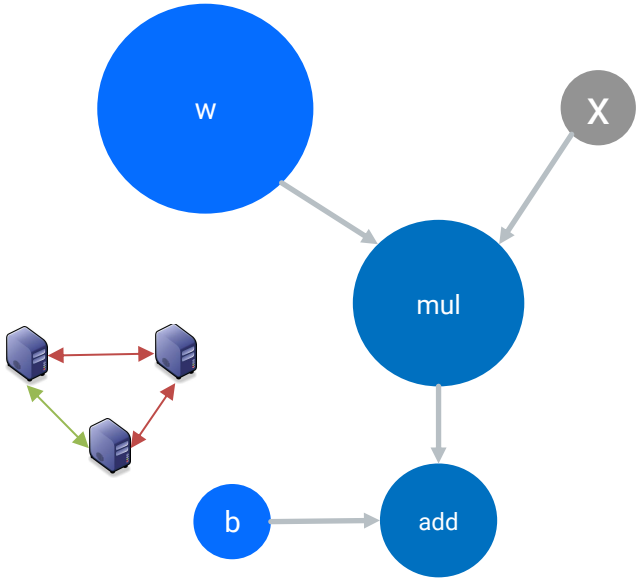


Implementations of transformations

Nowadays ML Frameworks

- *High-level Language APIs:* ML prototyping is made easier
- *Dataflow graph representation:* Composable and powerful representations for ML workloads, allows for various optimizations
- *Multi-layer architecture:* APIs–> graph –> runtime –> kernel - > device

Example: y = Wx + b



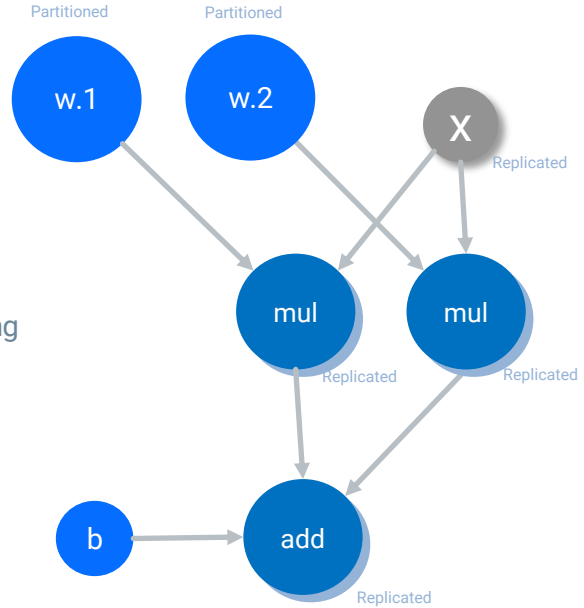
Compile

```
graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3
nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    w:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
    b:
    - 10.1.0.1:CPU0
  Partitioner:
    mul: [0, 2, 0]
  DeviceSetter:
    mul:
    - 10.1.0.1:GPU0
    - 10.1.0.1:GPU1
    add:
    - 10.1.0.1:GPU2
```

Distribution: Apply a Strategy to Transform the Graph

```
graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3
nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    w:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
    b:
    - 10.1.0.1:CPU0
  Partitioner:
    mul: [0, 2, 0]
  DeviceSetter:
    mul:
    - 10.1.0.1:GPU0
    - 10.1.0.1:GPU1
    add:
    - 10.1.0.1:GPU2
```

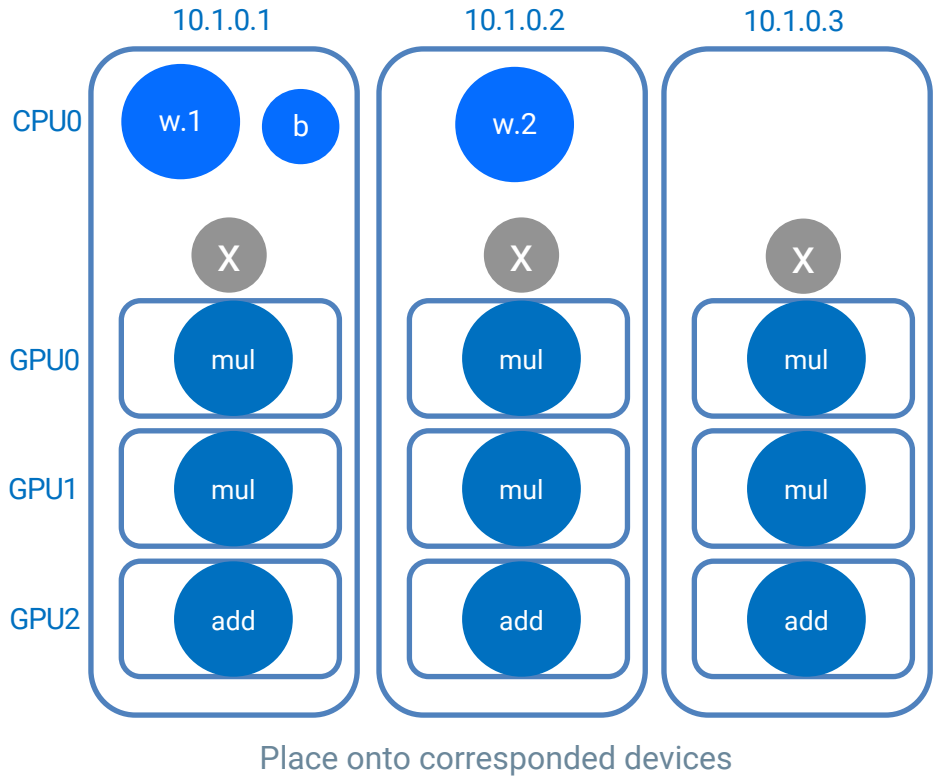
Graph rewriting



Distribution: Apply a Strategy to Transform the Graph

```
graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3
nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    w:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
    b:
    - 10.1.0.1:CPU0
  Partitioner:
    mul: [0, 2, 0]
  DeviceSetter:
    mul:
    - 10.1.0.1:GPU0
    - 10.1.0.1:GPU1
    add:
    - 10.1.0.1:GPU2
```

Execute



Other Work

- Scalable and Auto ML
  - AutoLoss [Zhang, et al. ICLR'19]
- Large-scale Applications
  - Large-scale topic models on twitter data [Zhang, KDD'15]
  - HD-CNN [ICCV'15]
- ML on/with structures
  - Structured GANs [Zhang et al. Neurips'18]
  - Symbolic Graph Reasoning [Neurips'18]
  - Semantic Manipulation [ECCV'18]
  - Visual paragraph generation [ICCV'17]
  - Learning concept taxonomies [ACL'16]