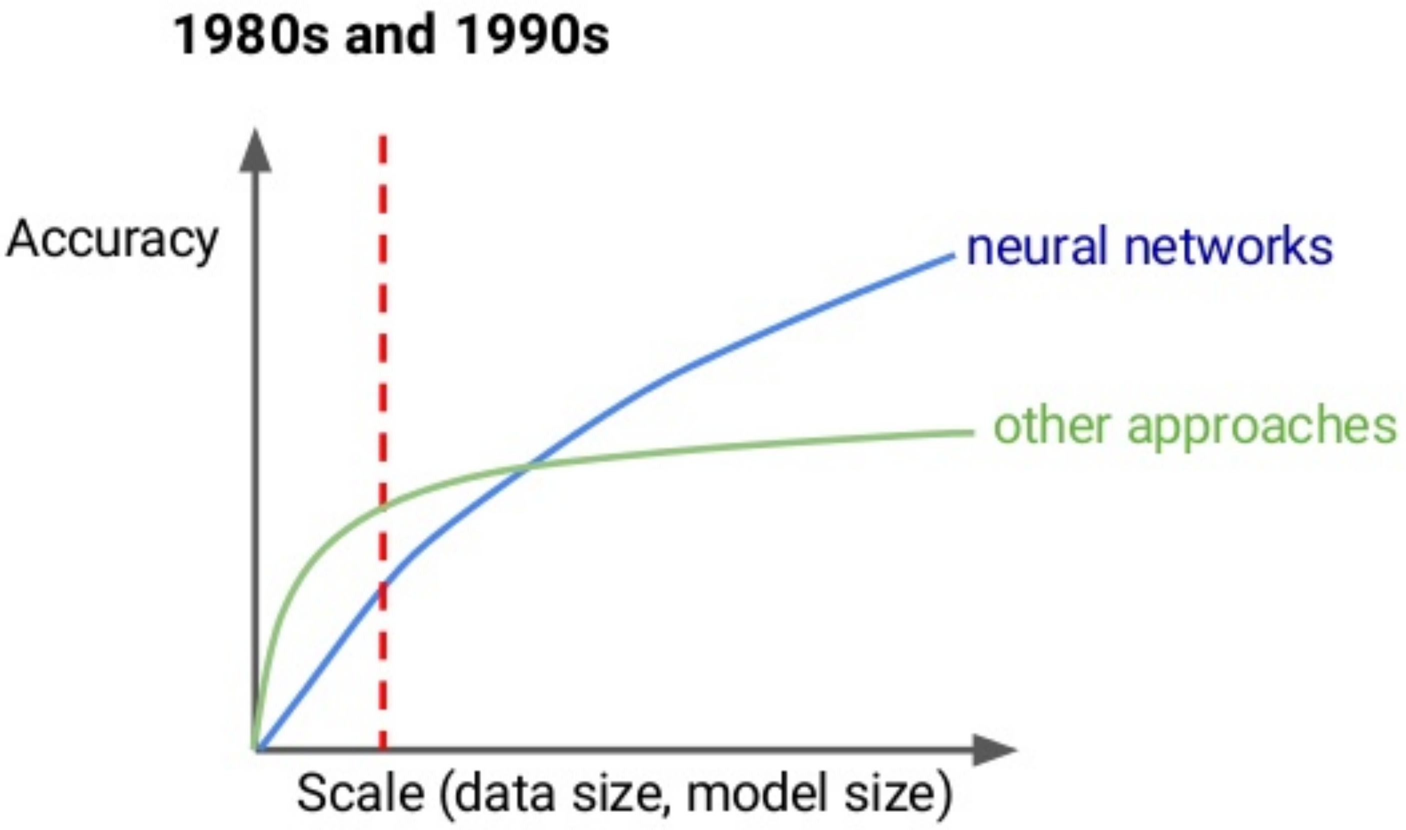
Automatically Discovering Systems Optimizations for Deep Learning

Zhihao Jia

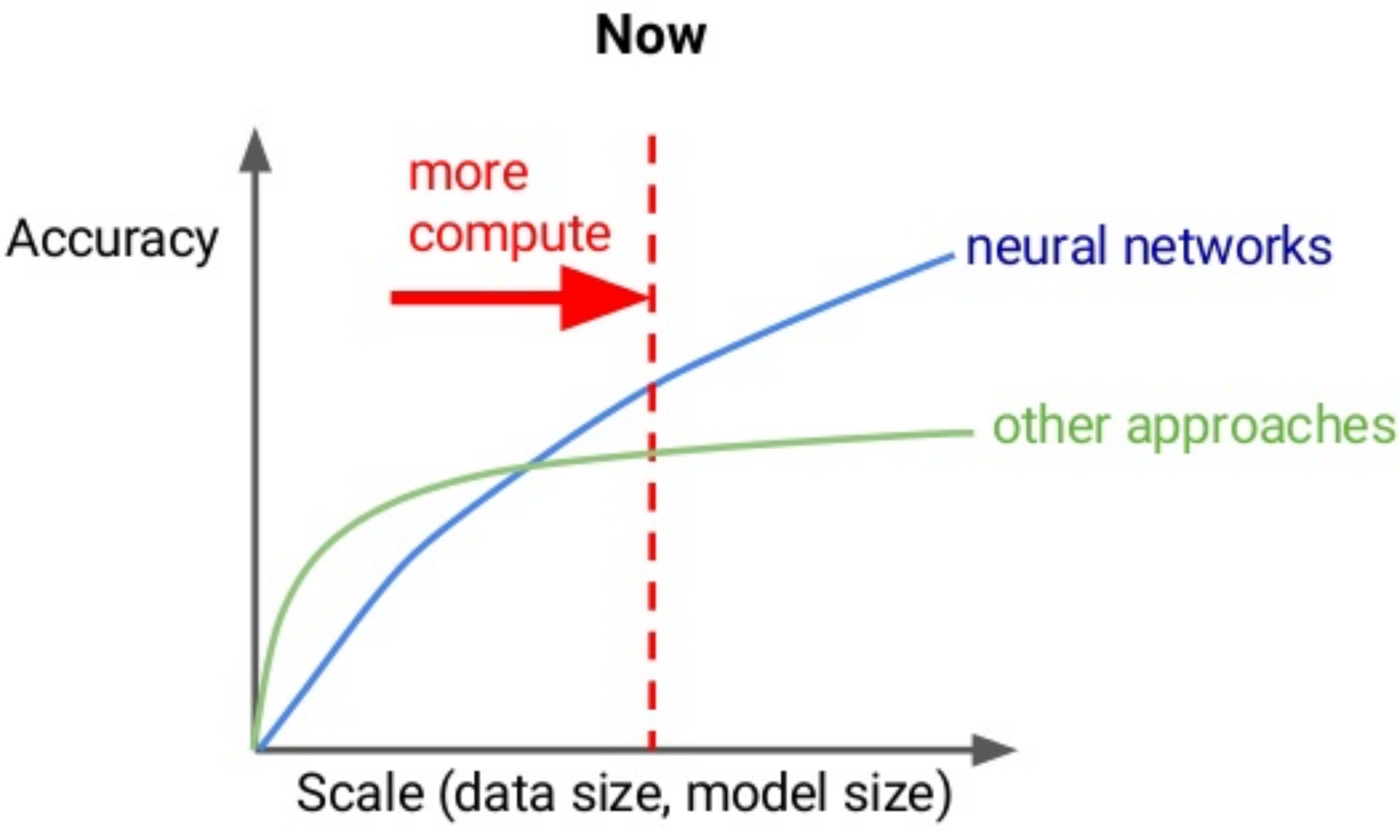
CMU and Facebook

The Rise of ML and Neural Networks

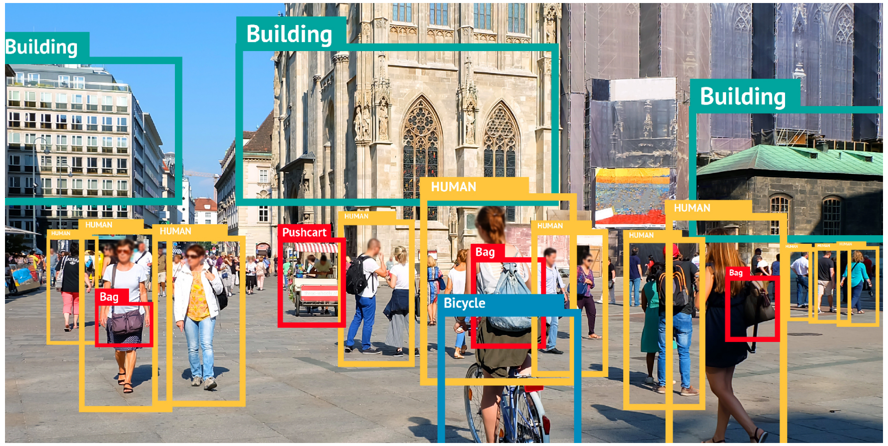


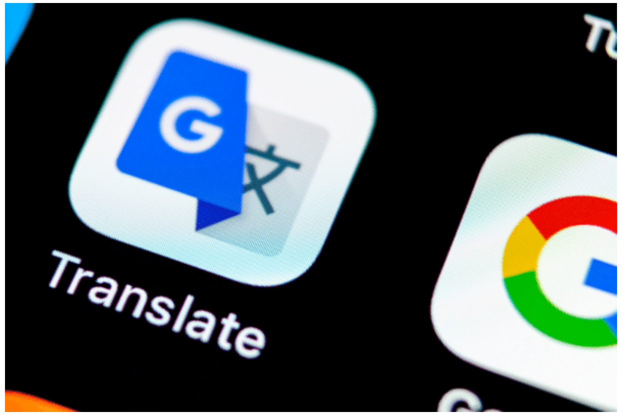
Adapted from Jeff Dean, HotChips 2017

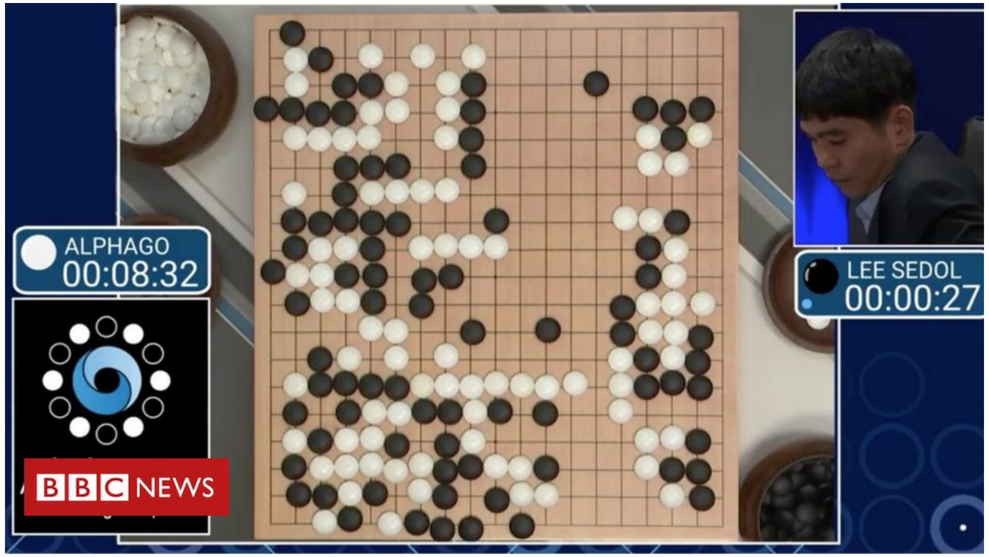
The Rise of ML and Neural Networks



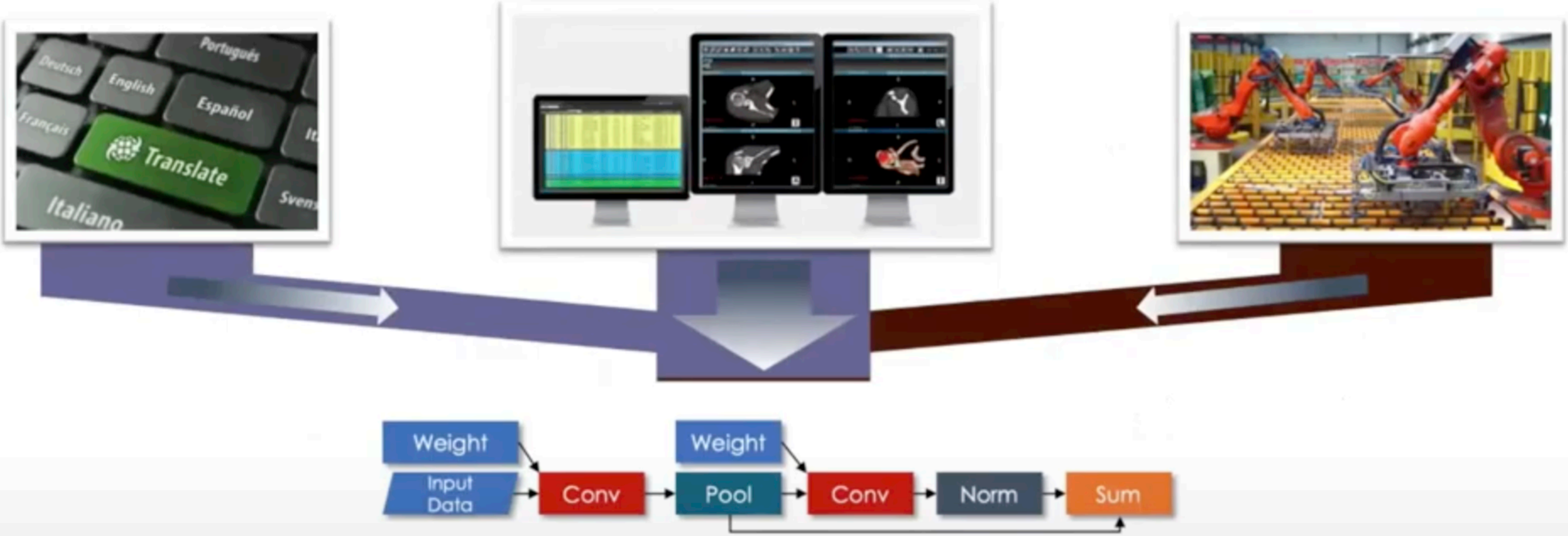
Adapted from Jeff Dean, HotChips 2017


Building
Building
Building
HUMAN
HUMAN
Pushcart
Bag
Bag
Bag
Bicycle


Translate


ALPHAGO
00:08:32
LEE SEDOL
00:00:27
BBC NEWS

Deep Neural Networks for Machine Translation

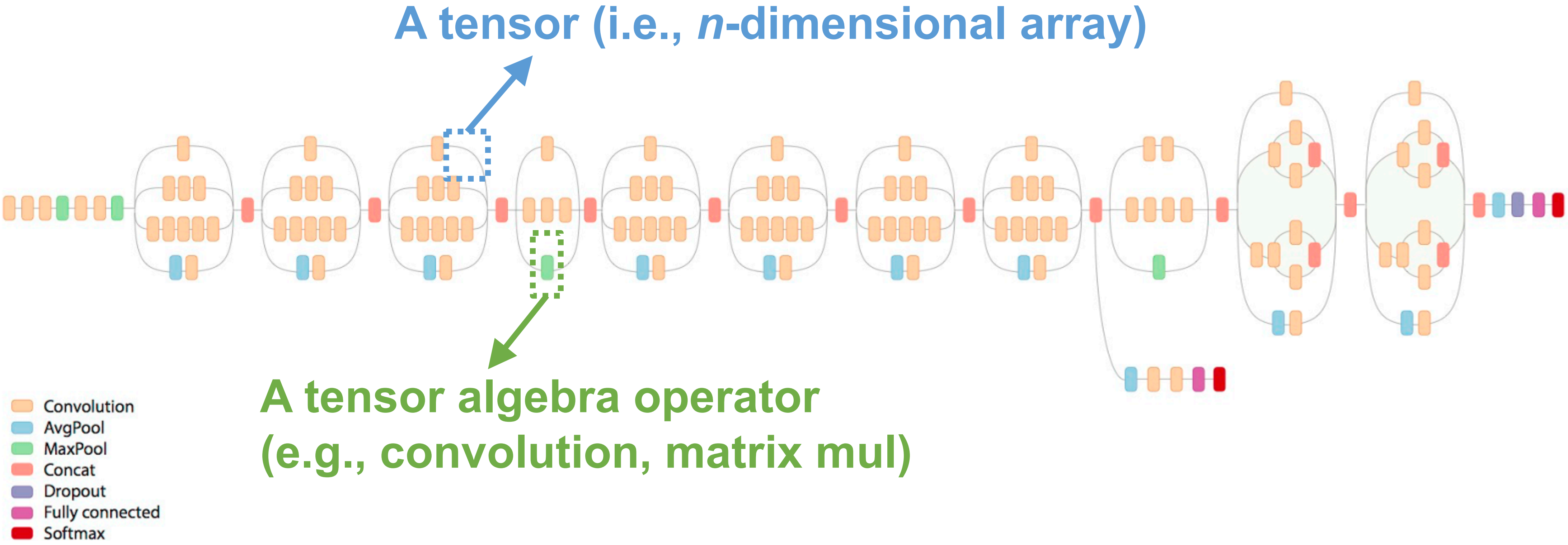


1000x Productivity

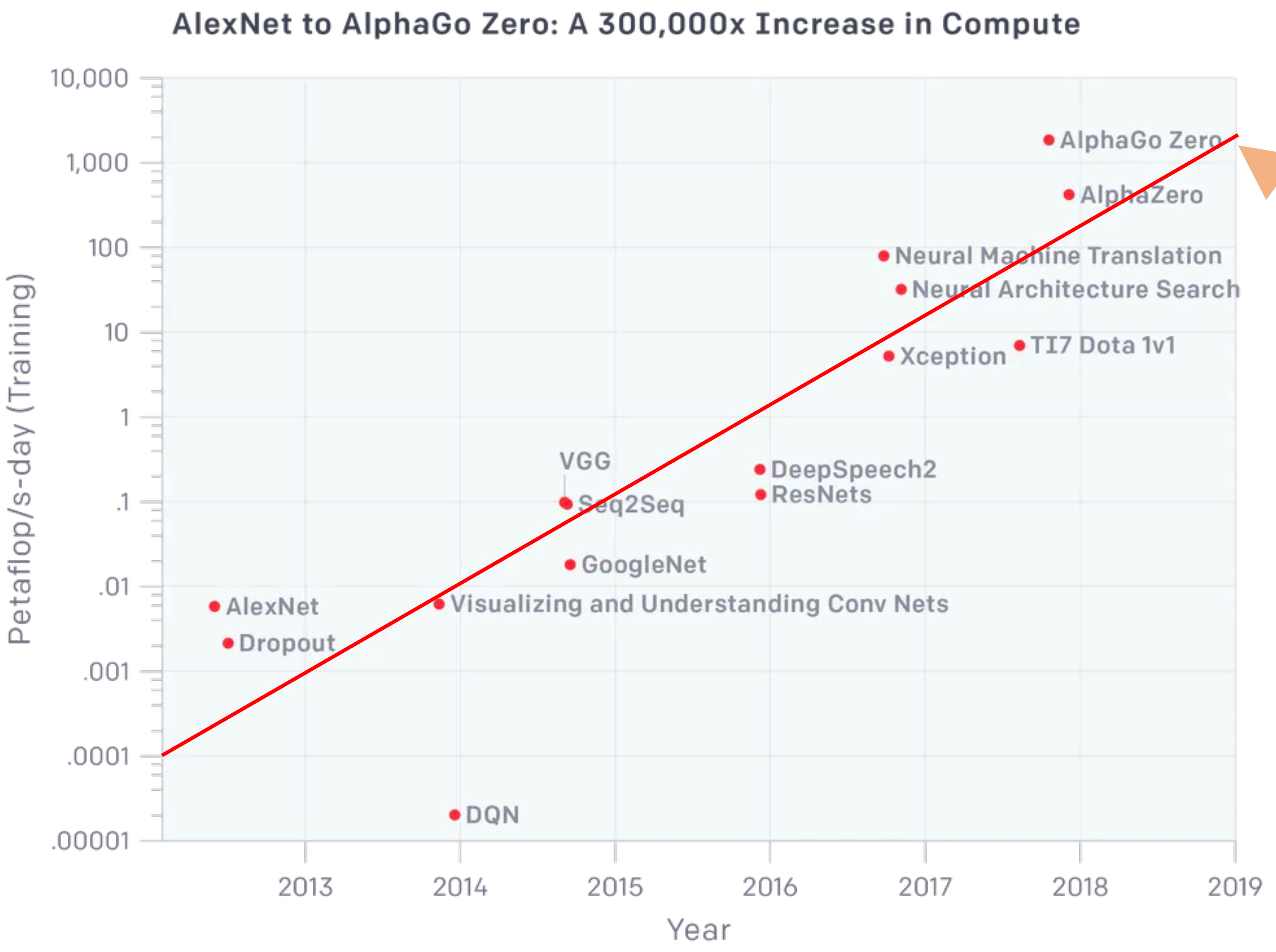
Google shrinks language translation code
from 500k imperative LoC to **500 lines of dataflow**

What is a Deep Neural Network?

- Collection of simple trainable mathematical units that work together to solve complicated tasks



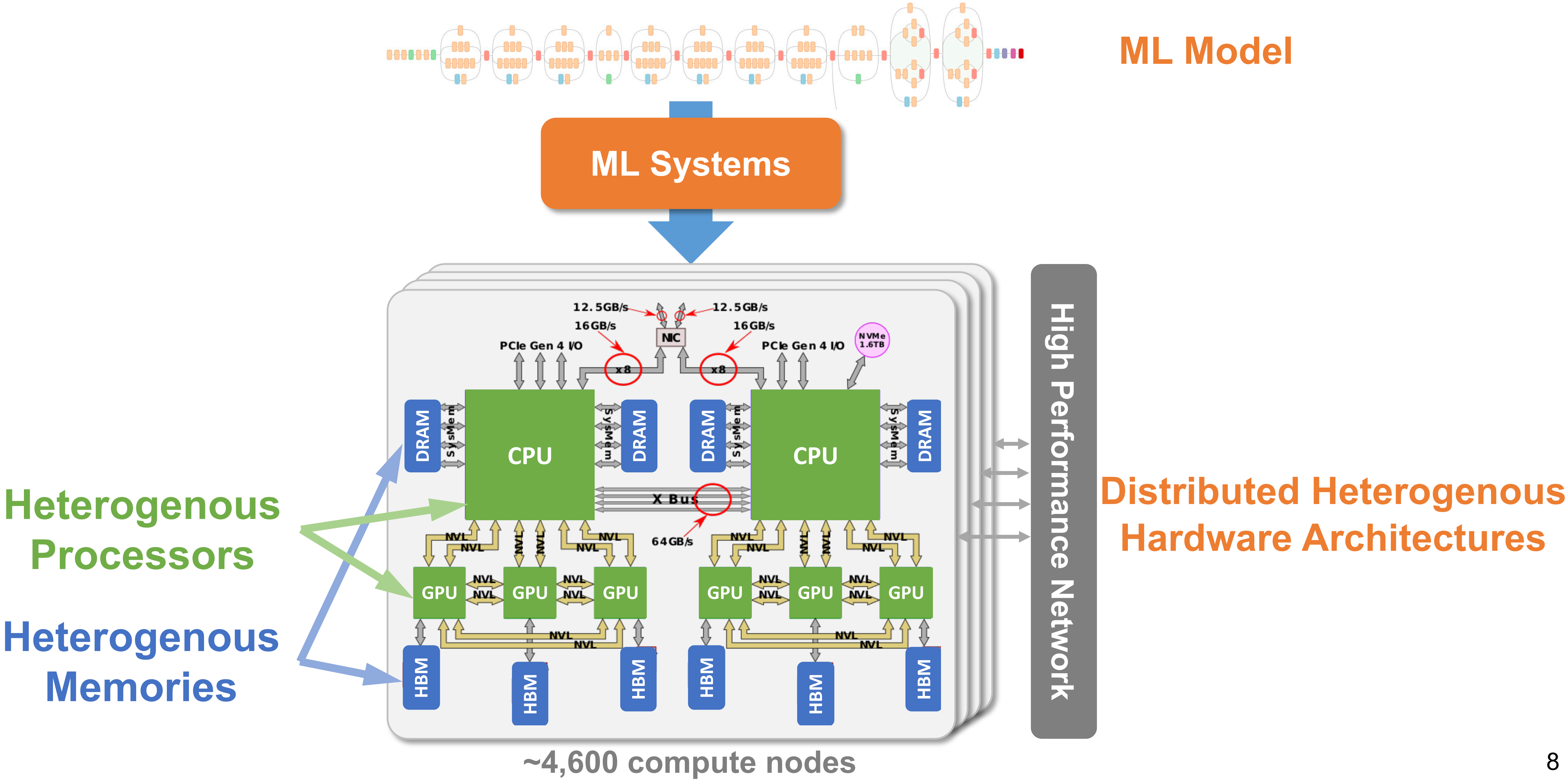
ML Computation is Increasing Exponentially



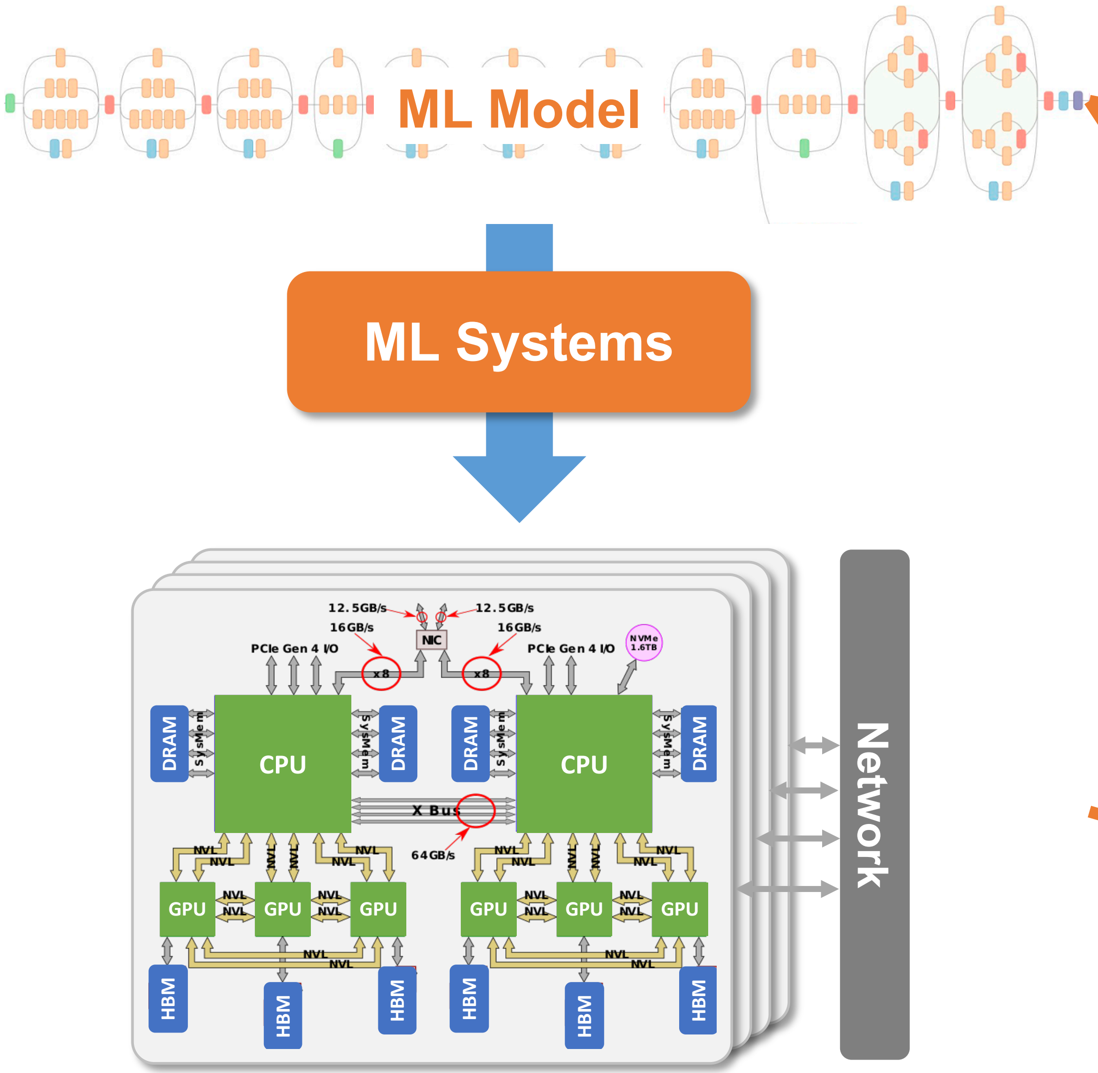
Amount of computation in largest ML training doubles **every 3.4 months**

[OpenAI Blog, 2018]

ML Systems are a Key Ingredient in ML



Challenges of Building ML Systems



Massively Parallel

Tensor algebra is parallelizable in many dimensions

New ML Operators

Continuously introduced into ML systems

Heterogenous Hardware

Different processor kinds and complex memory hierarchy

CMU Automated Learning Systems Lab

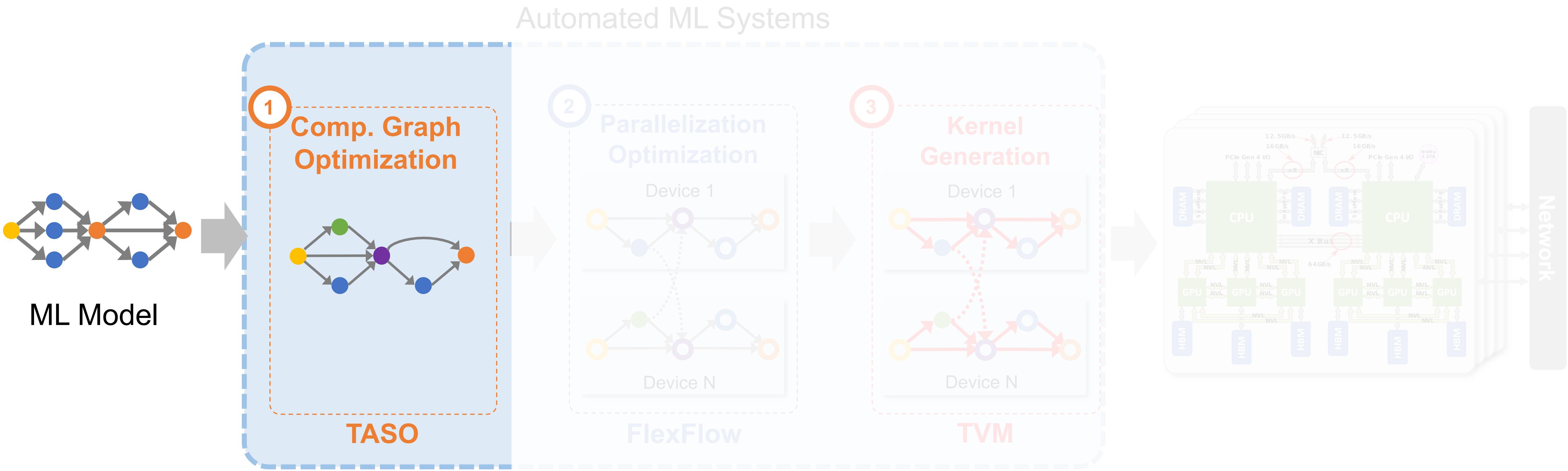
Mission: Automate the design and optimization of ML systems by leveraging

1. Statistical and mathematical properties of ML algorithms
2. Domain knowledge of modern hardware platforms

catalyst

https://catalyst.cs.cmu.edu/

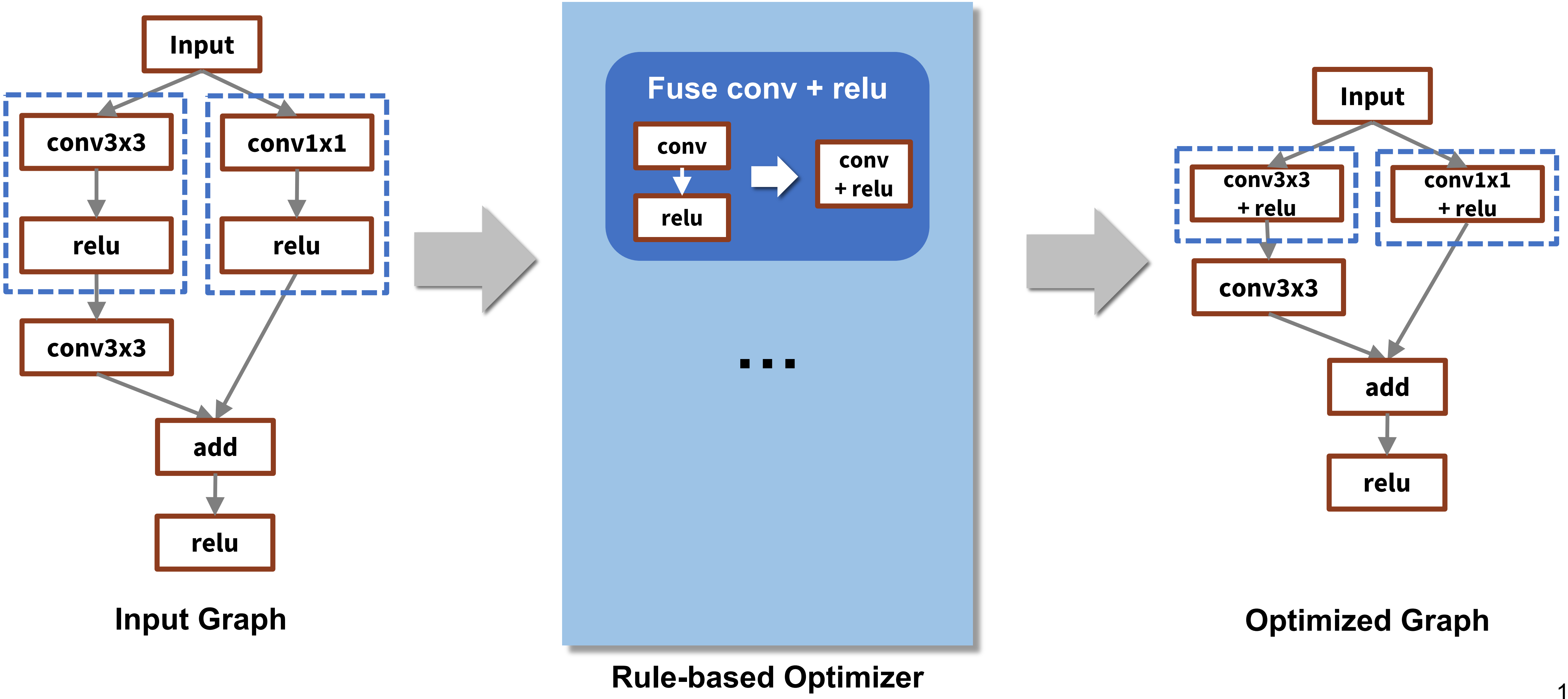
CMU Automated Learning Systems Lab



catalyst

https://catalyst.cs.cmu.edu/

Current Rule-based Graph Optimizations



Current Rule-based Graph Optimizations

TensorFlow currently includes ~<u>200</u> rules (~<u>53,000</u> LOC)

Fuse conv + relu

Fuse conv + batch normalization

Fuse multi. convs

...

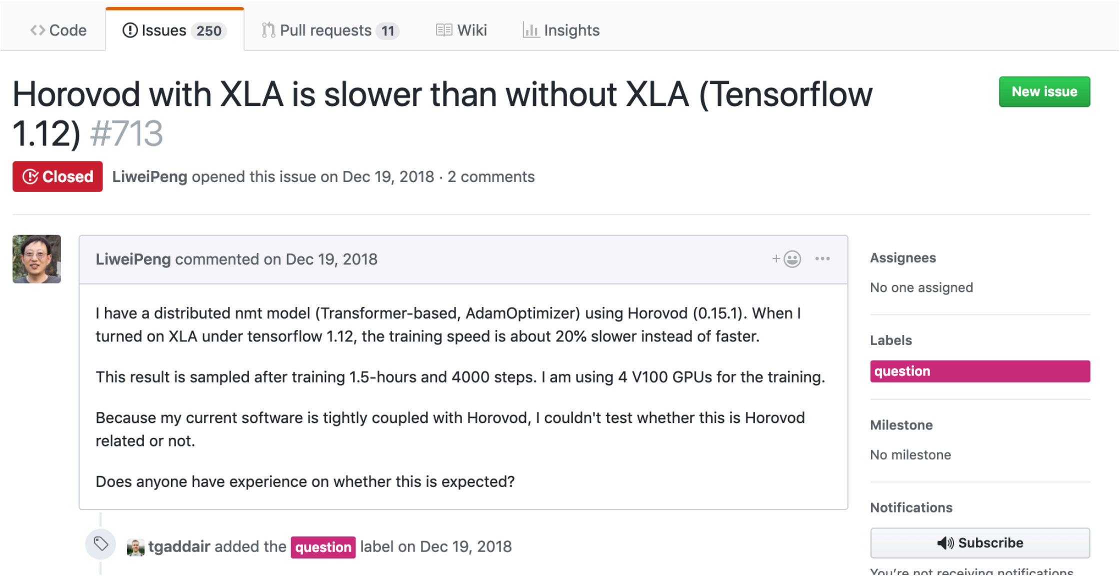
Rule-based Optimizer

```cpp
namespace tensorflow {
namespace graph_transforms {

// Converts Conv2D or MatMul ops followed by column-wise Muls into equivalent
// ops with the Mul baked into the convolution weights, to save computation
// during inference.
Status FoldBatchNorms(const GraphDef& input_graph_def,
                      const TransformFuncContext& context,
                      GraphDef* output_graph_def) {
  GraphDef replaced_graph_def;
  TF_RETURN_IF_ERROR(ReplaceMatchingOpTypes(
      input_graph_def,  // clang-format off
      {"Mul",                // mul_node
        {
          {"Conv2D|MatMul|DepthwiseConv2dNative",  // conv_node
            {
              {"*"},         // input_node
              {"Const"},     // weights_node
            }
          },
          {"Const"},         // mul_values_node
        }
      },  // clang-format on
      [](const NodeMatch& match, const std::set<string>& input_nodes,
         const std::set<string>& output_nodes,
         std::vector<NodeDef>* new_nodes) {
        // Find all the nodes we expect in the subgraph.
        const NodeDef& mul_node = match.node;
        const NodeDef& conv_node = match.inputs[0].node;
        const NodeDef& input_node = match.inputs[0].inputs[0].node;
        const NodeDef& weights_node = match.inputs[0].inputs[1].node;
        const NodeDef& mul_values_node = match.inputs[1].node;

        // Check that nodes that we use are not used somewhere else.
        for (const auto& node : {conv_node, weights_node, mul_values_node}) {
          if (output_nodes.count(node.name())) {
            // Return original nodes.
            new_nodes->insert(new_nodes->end(),
                              {mul_node, conv_node, input_node, weights_node,
                               mul_values_node});
            return Status::OK();
          }
        }

        Tensor weights = GetNodeTensorAttr(weights_node, "value");
        Tensor mul_values = GetNodeTensorAttr(mul_values_node, "value");

        // Make sure all the inputs really are vectors, with as many entries as
        // there are columns in the weights.
        int64 weights_cols;
        if (conv_node.op() == "Conv2D") {
          weights_cols = weights.shape().dim_size(3);
        } else if (conv_node.op() == "DepthwiseConv2dNative") {
          weights_cols =
              weights.shape().dim_size(2) * weights.shape().dim_size(3);
        } else {
          weights_cols = weights.shape().dim_size(1);
        }
        if ((mul_values.shape().dims() != 1) ||
            (mul_values.shape().dim_size(0) != weights_cols)) {
          return errors::InvalidArgument(
              "Mul constant input to batch norm has bad shape: ",
              mul_values.shape().DebugString());
        }

        // Multiply the original weights by the scale vector.
        auto weights_vector = weights.flat<float>();
        Tensor scaled_weights(DT_FLOAT, weights.shape());
        auto scaled_weights_vector = scaled_weights.flat<float>();
        for (int64 row = 0; row < weights_vector.dimension(0); ++row) {
          scaled_weights_vector(row) =
              weights_vector(row) *
              mul_values.flat<float>()(row % weights_cols);
        }

        // Construct the new nodes.
        NodeDef scaled_weights_node;
        scaled_weights_node.set_op("Const");
        scaled_weights_node.set_name(weights_node.name());
        SetNodeAttr("dtype", DT_FLOAT, &scaled_weights_node);
        SetNodeTensorAttr<float>("value", scaled_weights, &scaled_weights_node);
        new_nodes->push_back(scaled_weights_node);

        new_nodes->push_back(input_node);

        NodeDef new_conv_node;
        new_conv_node = conv_node;
        new_conv_node.set_name(mul_node.name());
        new_nodes->push_back(new_conv_node);

        return Status::OK();
      },
      {}, &replaced_graph_def));
  *output_graph_def = replaced_graph_def;
  return Status::OK();
}

REGISTER_GRAPH_TRANSFORM("fold_batch_norms", FoldBatchNorms);

}  // namespace graph_transforms
}  // namespace tensorflow
```

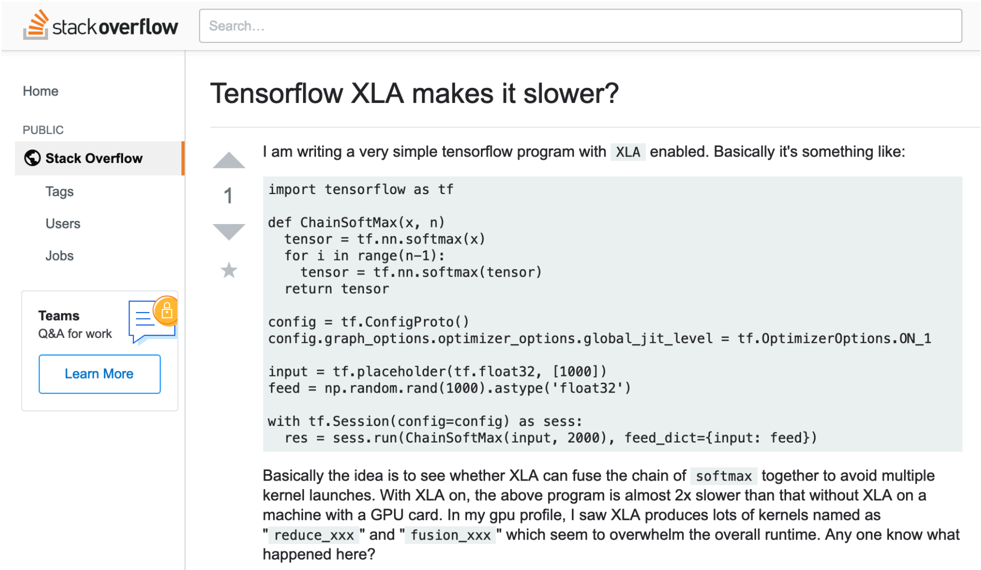
Limitations of Rule-based Optimizations

Robustness

Experts' heuristics do not apply to all models/hardware



When I turned on XLA (TensorFlow's graph optimizer), the training speed is **about 20% slower**



With XLA, my program is **almost 2x slower than** without XLA

Limitations of Rule-based Optimizations

Robustness
Experts' heuristics do not apply to all models/hardware

Scalability
New operators and graph structures require more rules

TensorFlow currently uses ~4K LOC to optimize convolution

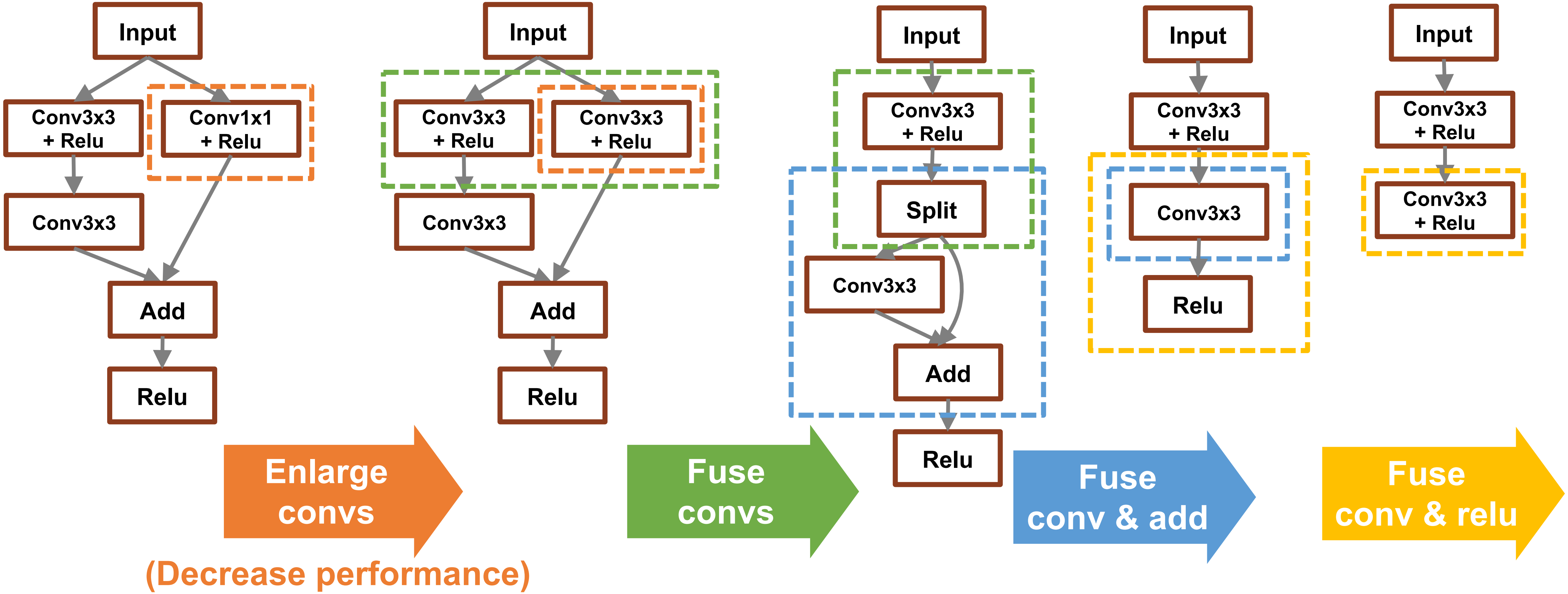
Limitations of Rule-based Optimizations

Robustness
Experts' heuristics do not apply to all models/hardware

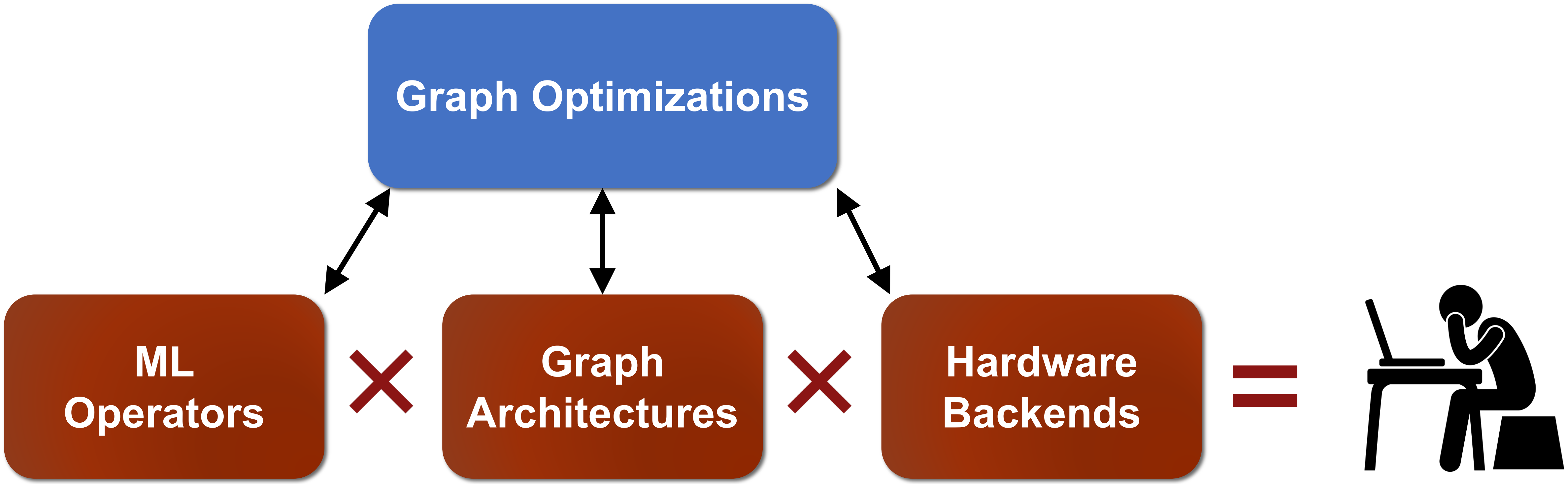
Scalability
New operators and graph structures require more rules

Performance
Miss subtle optimizations for specific models/hardware

Motivating Example (ResNet)



The final graph is <u>30% faster</u> on V100 but <u>10% slower</u> on K80.



Infeasible to manually design graph optimizations for all cases

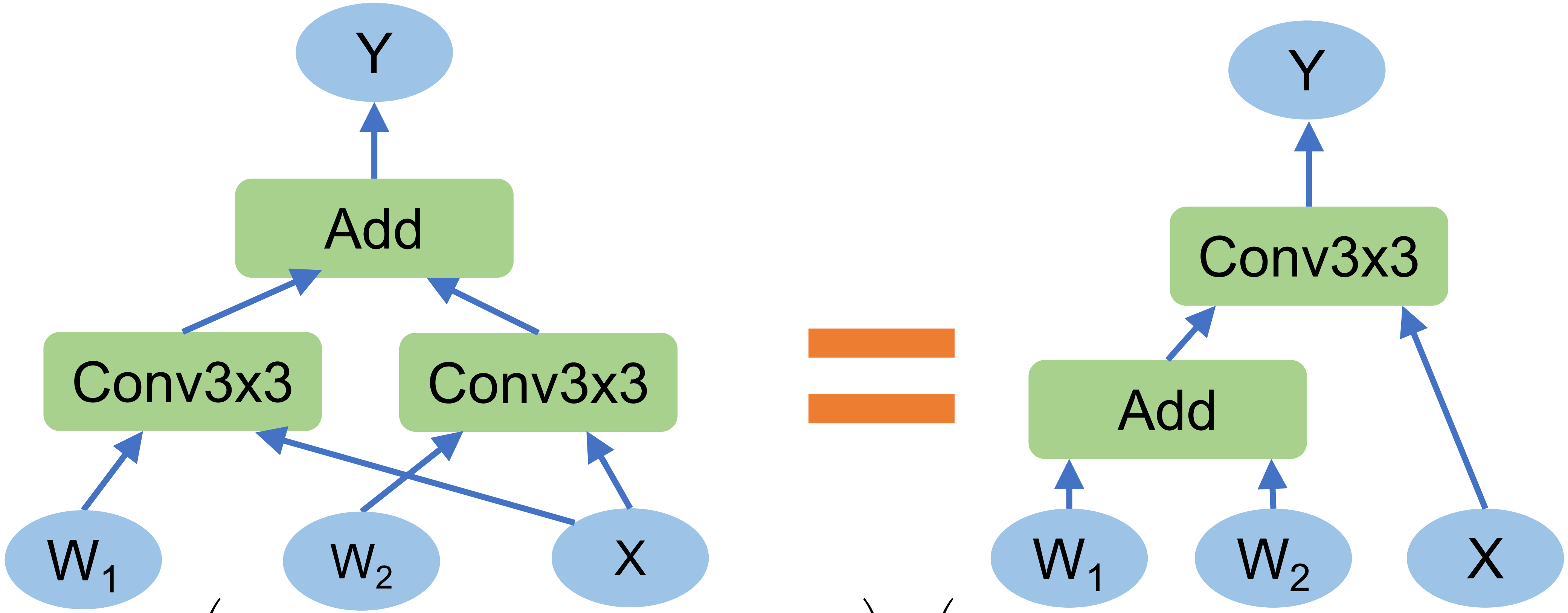
Is it possible to generate them automatically?

TASO: Tensor Algebra SuperOptimizer

Key idea: replace manually-designed graph optimizations with ***automated generation and verification*** of graph substitutions for tensor algebra

- **Less engineering effort:** **53,000** LOC for manual graph optimizations in TensorFlow → **1,400** LOC in TASO
- **Better performance:** outperform existing optimizers by up to **3x**
- **Stronger correctness:** formally verify all generated substitutions

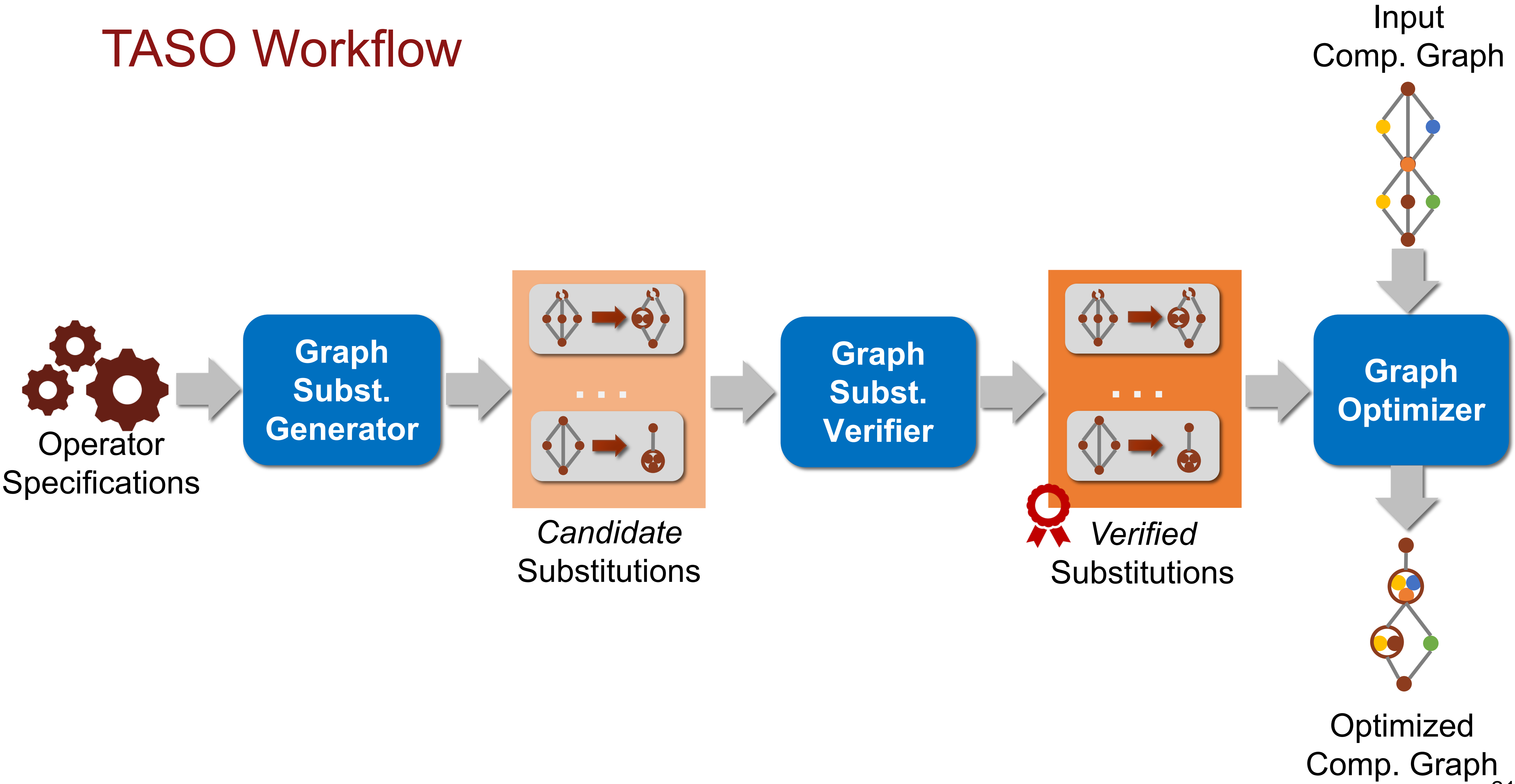
Graph Substitution



$$Y(n,c,h,w) = \left(\sum_{d,u,v} X(n,d,h+u,w+v) * W1(c,d,u,v)\right) + \left(\sum_{d,u,v} X(n,d,h+u,w+v) * W2(c,d,u,v)\right)$$

$$\Leftrightarrow Y(n,c,h,w) = \sum_{d,u,v} X(n,d,h+u,w+v) * \left(\left(W_1\,(c,d,u,v) + W_2\,(c,d,u,v)\right)\right.$$

TASO Workflow



Key Challenges

1. How to generate potential substitutions?

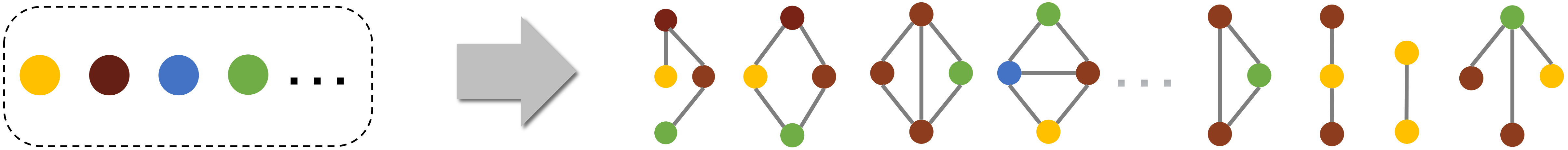
Graph fingerprints

2. How to verify their correctness?

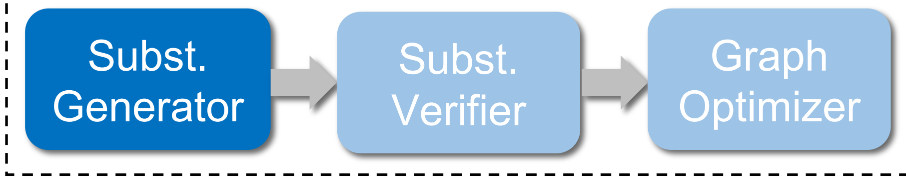
Operator specifications + theorem prover

Graph Substitution Generator

Enumerate **all possible** graphs up to a fixed size using available operators

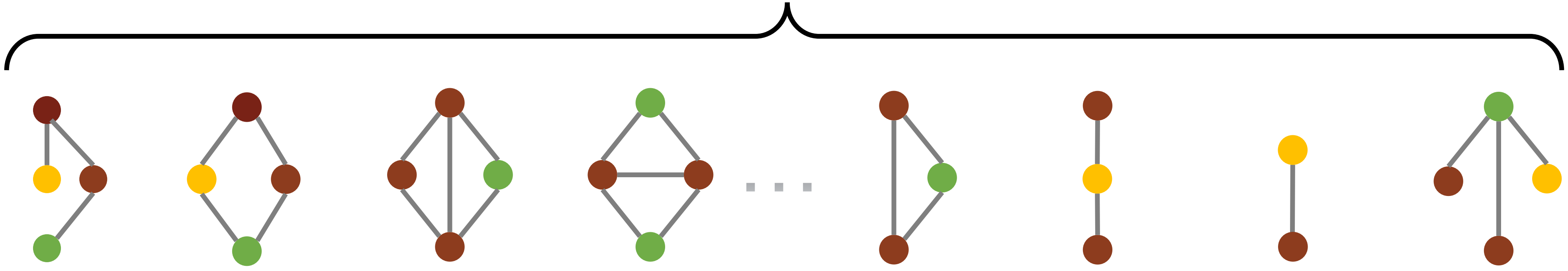


Operators supported by hardware backend



Graph Substitution Generator

66M graphs with up to **4** operators



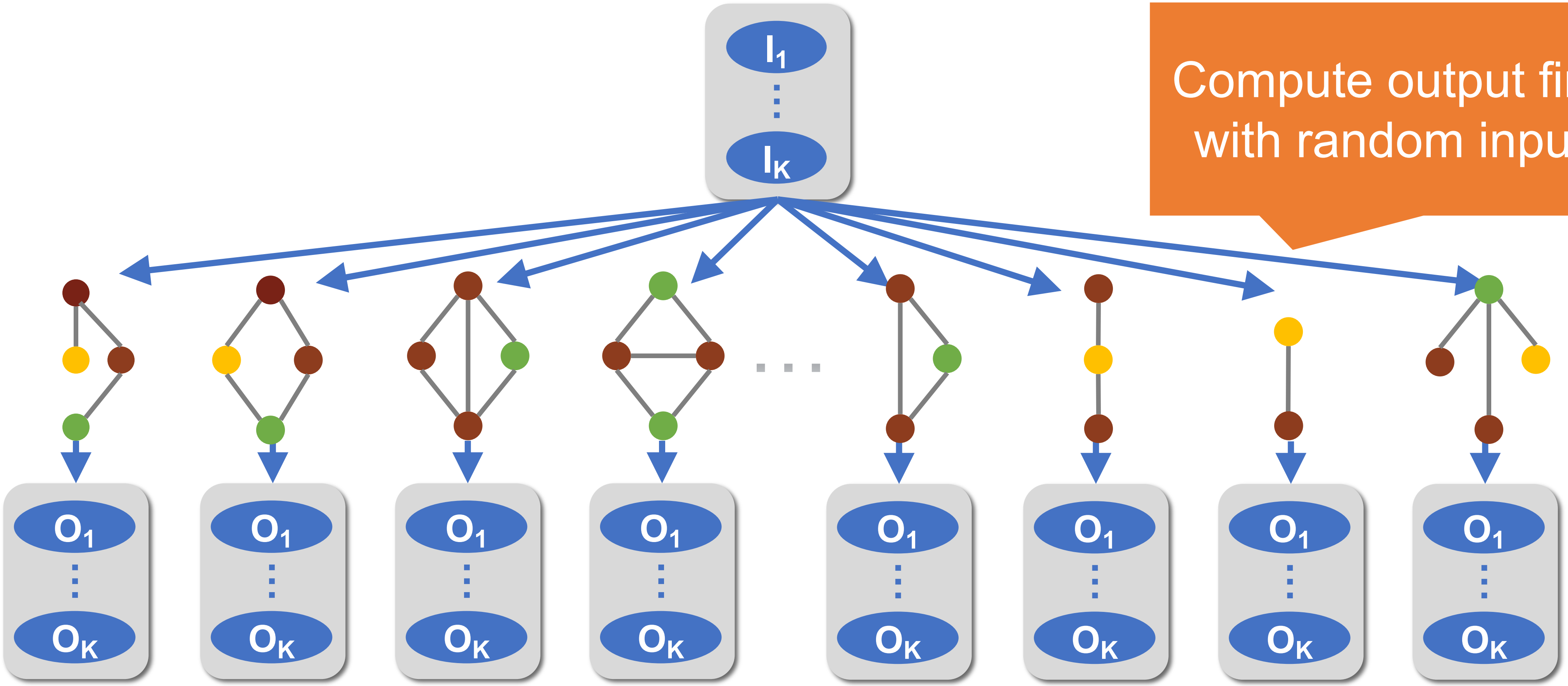
A substitution = a pair of equivalent graphs

Explicitly considering all pairs does not scale

Subst. Generator → Subst. Verifier → Graph Optimizer

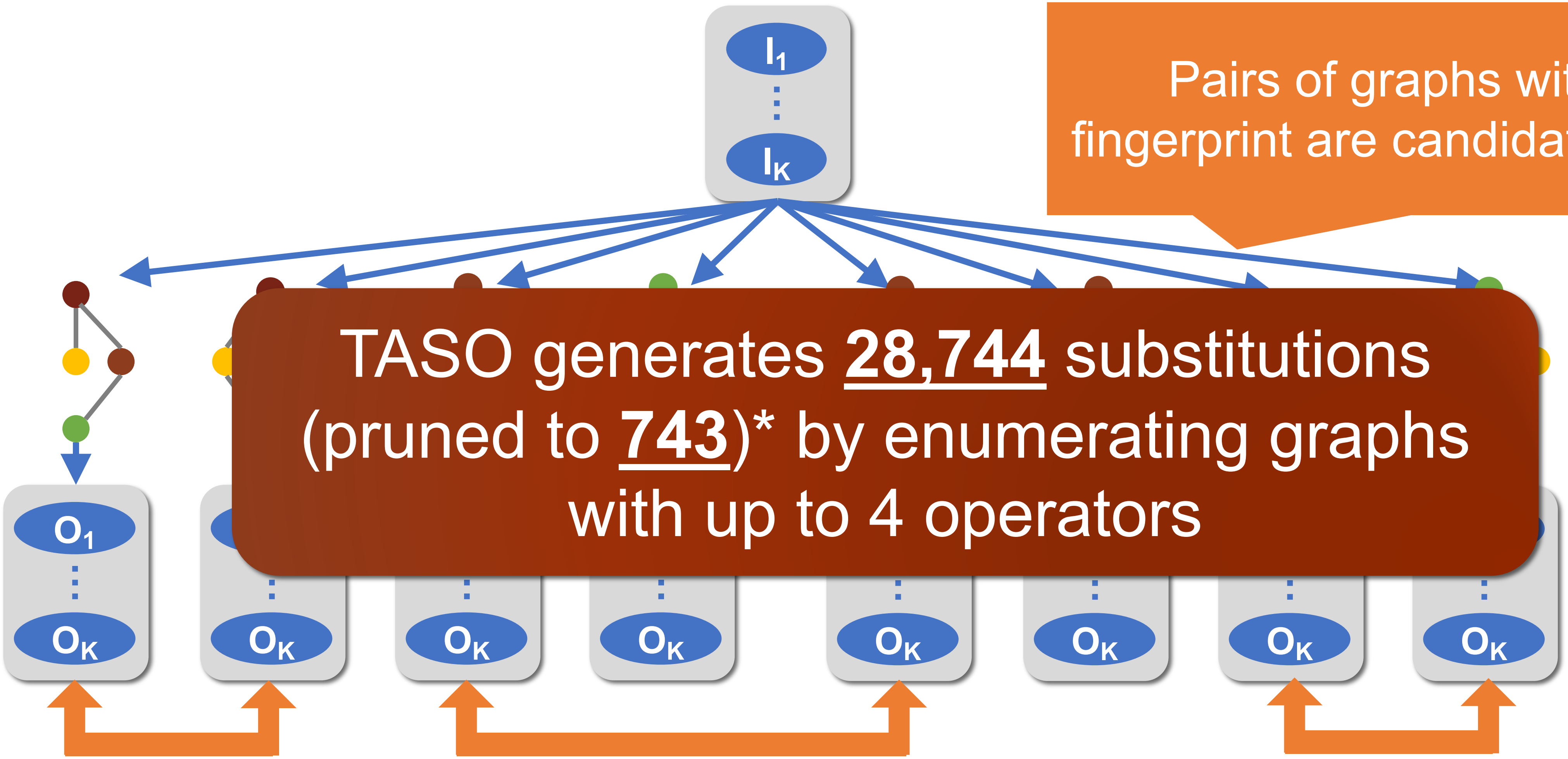
Graph Substitution Generator

Compute output fingerprints with random input tensors



Graph Substitution Generator

Pairs of graphs with identical fingerprint are candidate substitutions

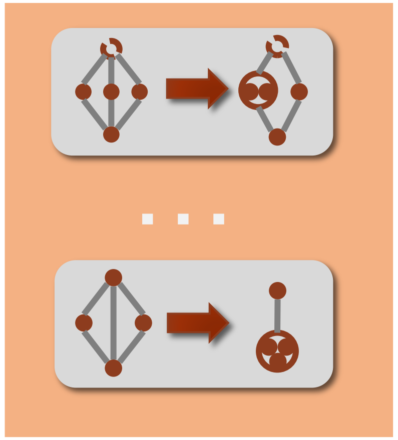


TASO generates **28,744** substitutions (pruned to **743**)* by enumerating graphs with up to 4 operators

*Pruning details available in **Z. Jia et al. SOSP’19**

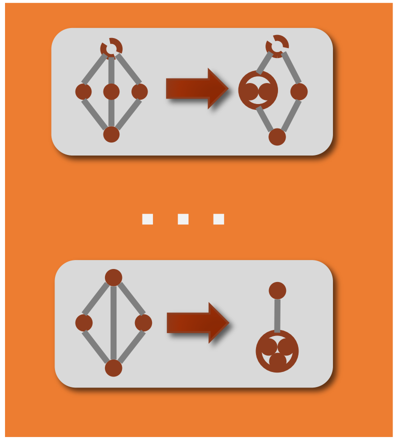
Subst. Generator → Subst. Verifier → Graph Optimizer

Graph Substitution Verifier



Candidate
Substitutions

Graph Subst. Verifier



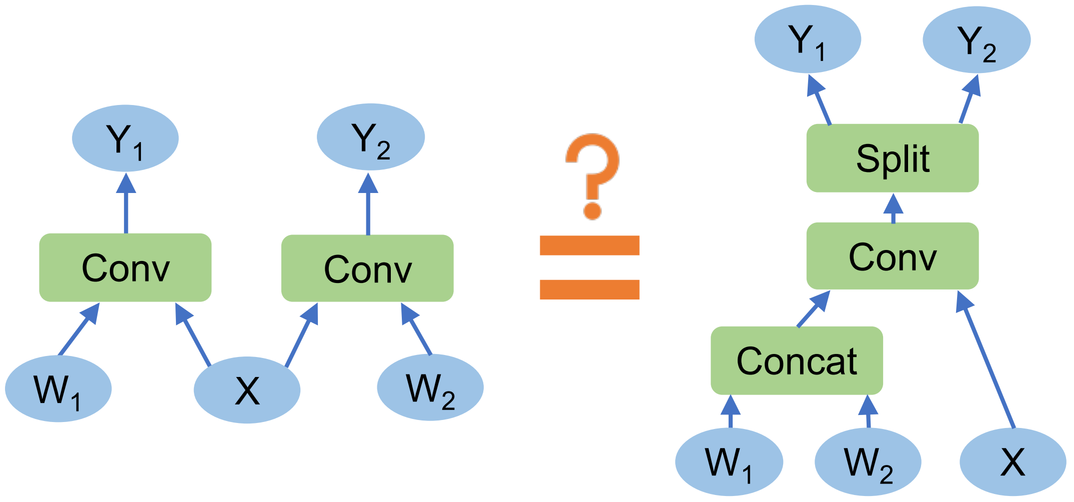
Verified
Substitutions

P1. conv is distributive over concatenation
P2. conv is bilinear
…
Pn.

$$\forall x, w_1, w_2 .$$
$$Conv(x, Concat(w_1, w_2)) = Concat(Conv(x, w_1), Conv(x, w_2))$$

Operator Specifications

Verification Workflow



$(Conv(x, w_1), Conv\ (x, w_2))$

$Split(Conv(x, Concat(w_1, w_2)))$

$$\forall x, w_1, w_2\,.\quad \big(Conv(x, w_1), Conv(x, w_2)\big) = Split\Big(Conv\big(x, Concat(w_1, w_2)\big)\Big)$$

Automated Theorem Prover

P1. $\forall x, w_1, w_2\,.$ $Conv\big(x, Concat(w_1, w_2)\big) = Concat\big(Conv(x, w_1), Conv(x, w_2)\big)$

P2. …

Operator Specifications

Verification Effort

TASO generates all **743** substitutions in 5 minutes, and verifies them against **43** operator properties in 10 minutes

Supporting a new operator requires **a few hours** of human effort to specify its properties

Operator specifications in TASO ≈ **1,400** LOC
Manual graph optimizations in TensorFlow ≈ **53,000** LOC

Operator Property	Comment
$\forall x,y,z.\ \mathtt{ewadd}(x,\mathtt{ewadd}(y,z)) = \mathtt{ewadd}(\mathtt{ewadd}(x,y),z)$	ewadd is associative
$\forall x,y.\ \mathtt{ewadd}(x,y) = \mathtt{ewadd}(y,x)$	ewadd is commutative
$\forall x,y,z.\ \mathtt{ewmul}(x,\mathtt{ewmul}(y,z)) = \mathtt{ewmul}(\mathtt{ewmul}(x,y),z)$	ewmul is associative
$\forall x,y.\ \mathtt{ewmul}(x,y) = \mathtt{ewmul}(y,x)$	ewmul is commutative
$\forall x,y,z.\ \mathtt{ewmul}(\mathtt{ewadd}(x,y),z) = \mathtt{ewadd}(\mathtt{ewmul}(x,z),\mathtt{ewmul}(y,z))$	distributivity
$\forall x,y,w.\ \mathtt{smul}(\mathtt{smul}(x,y),w) = \mathtt{smul}(x,\mathtt{smul}(y,w))$	smul is associative
$\forall x,y,w.\ \mathtt{smul}(\mathtt{ewadd}(x,y),w) = \mathtt{ewadd}(\mathtt{smul}(x,w),\mathtt{smul}(y,w))$	distributivity
	ommutativity
	is its own inverse
	ommutativity
	ommutativity
	ommutativity
	associative
	inear
	inear
	d transpose
	linear
$\forall s,p,x,y,w.\ \mathtt{smul}(\mathtt{conv}(s,p,\mathsf{A}_{\mathsf{none}},x,y),w) = \mathtt{conv}(s,p,\mathsf{A}_{\mathsf{none}},\mathtt{smul}(x,w),y)$	conv is bilinear
$\forall s,p,x,y,z.\ \mathtt{conv}(s,p,\mathsf{A}_{\mathsf{none}},x,\mathtt{ewadd}(y,z)) = \mathtt{ewadd}(\mathtt{conv}(s,p,\mathsf{A}_{\mathsf{none}},x,y),\mathtt{conv}(s,p,\mathsf{A}_{\mathsf{none}},x,z))$	conv is bilinear
	linear
	nvolution kernel
	$\mathsf{A}_{\mathsf{relu}}$ applies relu
	ommutativity
	conv. with $\mathsf{C}_{\mathsf{pool}}$
	rnel
	atrix
	entity
$\forall a,x,y.\ \mathtt{split}_0(a,\mathtt{concat}(a,x,y)) = x$	split definition
	ition
	f concatenation
	ommutativity
	ommutativity
	ommutativity
	ommutativity
	ion and transpose
	ion and matrix mul.
	ion and matrix mul.
	tion and conv.
	concatenation and conv.
$\forall s,p,x,y,z,w.\ \mathtt{conv}(s,p,\mathsf{A}_{\mathsf{none}},\mathtt{concat}(1,x,z),\mathtt{concat}(1,y,w)) = \mathtt{ewadd}(\mathtt{conv}(s,p,\mathsf{A}_{\mathsf{none}},x,y),\mathtt{conv}(s,p,\mathsf{A}_{\mathsf{none}},z,w))$	concatenation and conv.
$\forall k,s,p,x,y.\ \mathtt{concat}(1,\mathtt{pool}_{\mathsf{avg}}(k,s,p,x),\mathtt{pool}_{\mathsf{avg}}(k,s,p,y)) = \mathtt{pool}_{\mathsf{avg}}(k,s,p,\mathtt{concat}(1,x,y))$	concatenation and pooling
$\forall k,s,p,x,y.\ \mathtt{concat}(0,\mathtt{pool}_{\mathsf{max}}(k,s,p,x),\mathtt{pool}_{\mathsf{max}}(k,s,p,y)) = \mathtt{pool}_{\mathsf{max}}(k,s,p,\mathtt{concat}(0,x,y))$	concatenation and pooling
$\forall k,s,p,x,y.\ \mathtt{concat}(1,\mathtt{pool}_{\mathsf{max}}(k,s,p,x),\mathtt{pool}_{\mathsf{max}}(k,s,p,y)) = \mathtt{pool}_{\mathsf{max}}(k,s,p,\mathtt{concat}(1,x,y))$	concatenation and pooling

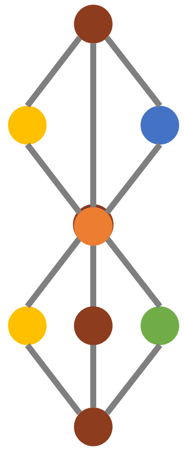
Search-Based Graph Optimizer

Cost Model[1]

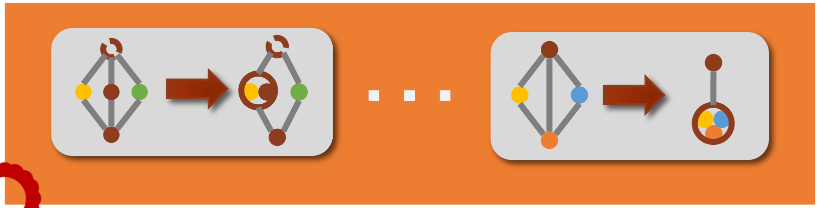
- Based on individual operators' cost
- Measure the cost of each operator on hardware

Search-Based Graph Optimizer

- **Cost-based backtracking search**
- Optimizing an ML model takes less than **10** minutes



Input
Comp. Graph

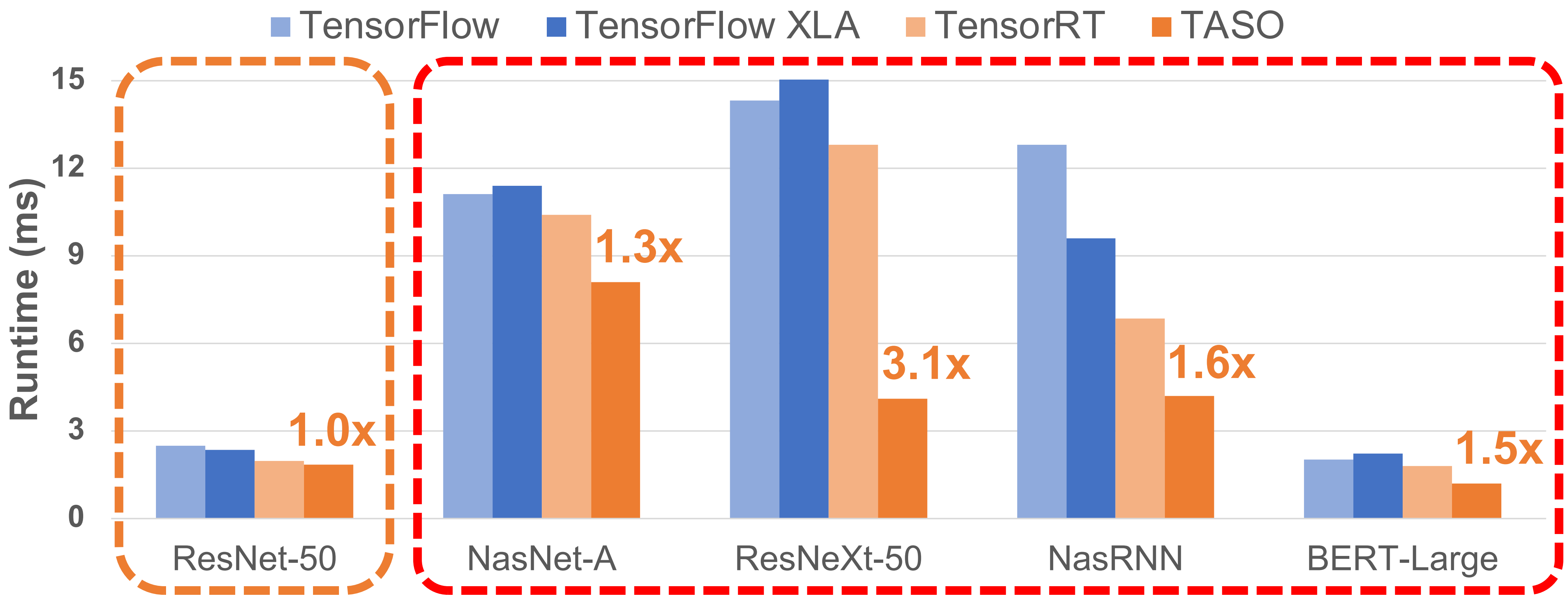


Verified Substitutions

Optimized
Comp. Graph

1. Exploring Hidden Dimensions in Parallelizing Convolutional Neural Networks. ICML'18.

End-to-end Inference Performance (Nvidia V100 GPU)



Competitive on standard models

Larger speedups on emerging models

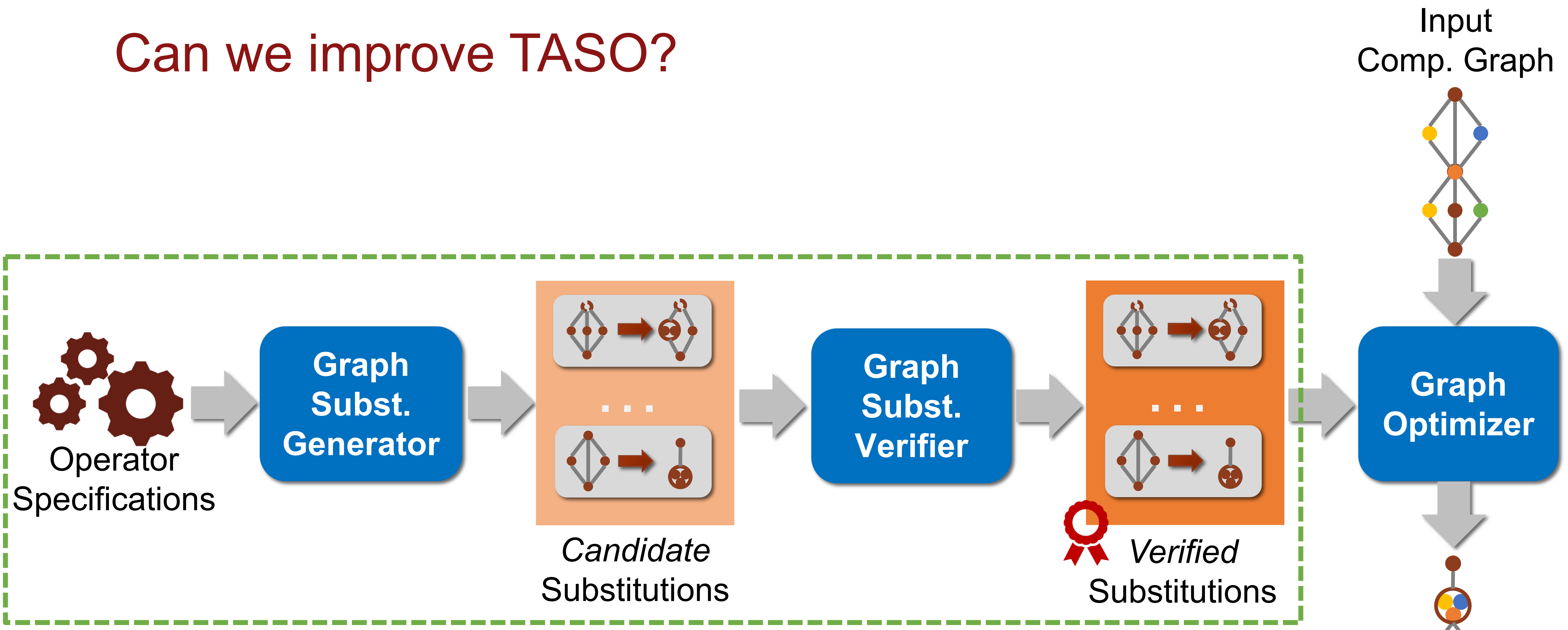
TASO

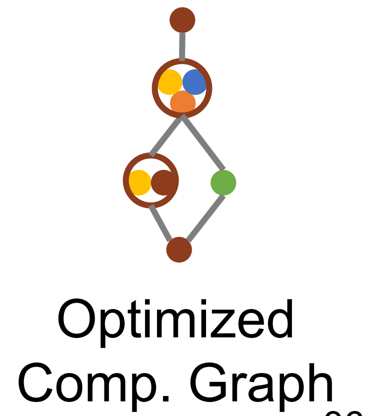
First DNN graph optimizer that automatically generates substitutions

- Less engineering effort
- Better runtime performance
- Stronger correctness guarantee

1. TASO: Optimizing Deep Learning Computation with Automated Generation of Graph Substitutions. SOSP'19.
2. Optimizing DNN Computation with Relaxed Graph Substitutions. MLSys'19.
3. Exploring Hidden Dimensions in Parallelizing Convolutional Neural Networks. ICML'18.

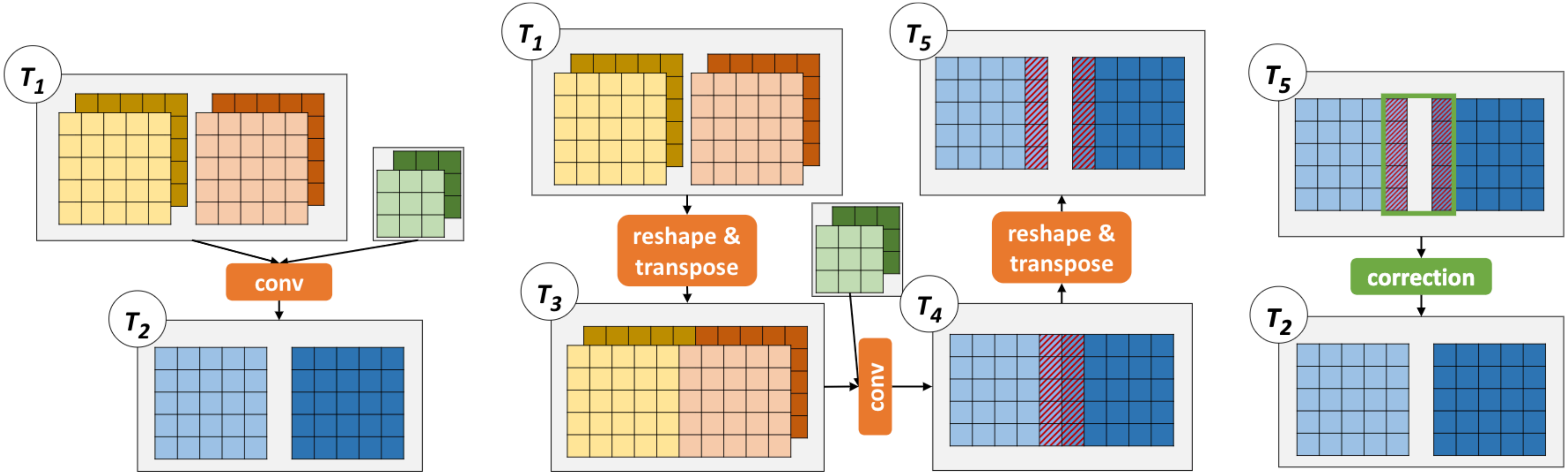
Can we improve TASO?





TASO only discovers fully equivalent substs.
Can we use substs that are partially equivalent and correct the results afterwards?

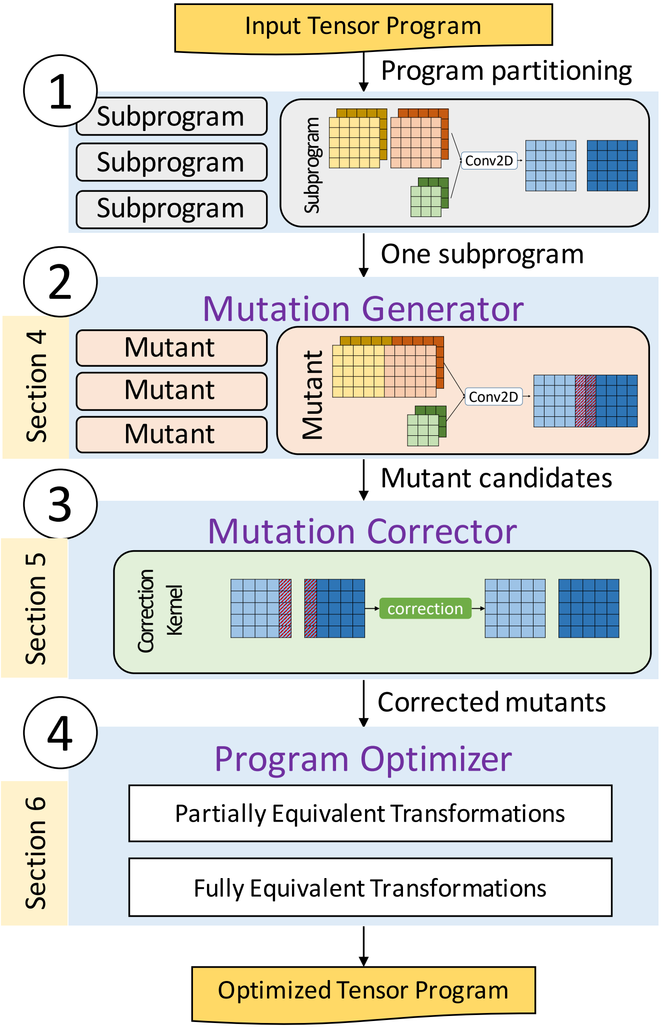
Motivating Example: Partially Equivalent Substs



(a) Input program. (b) A partially equivalent transformation. (c) Correcting results.

PET: Partially Equivalent Substs and Auto Corrections

Up to 2.5x faster than TASO



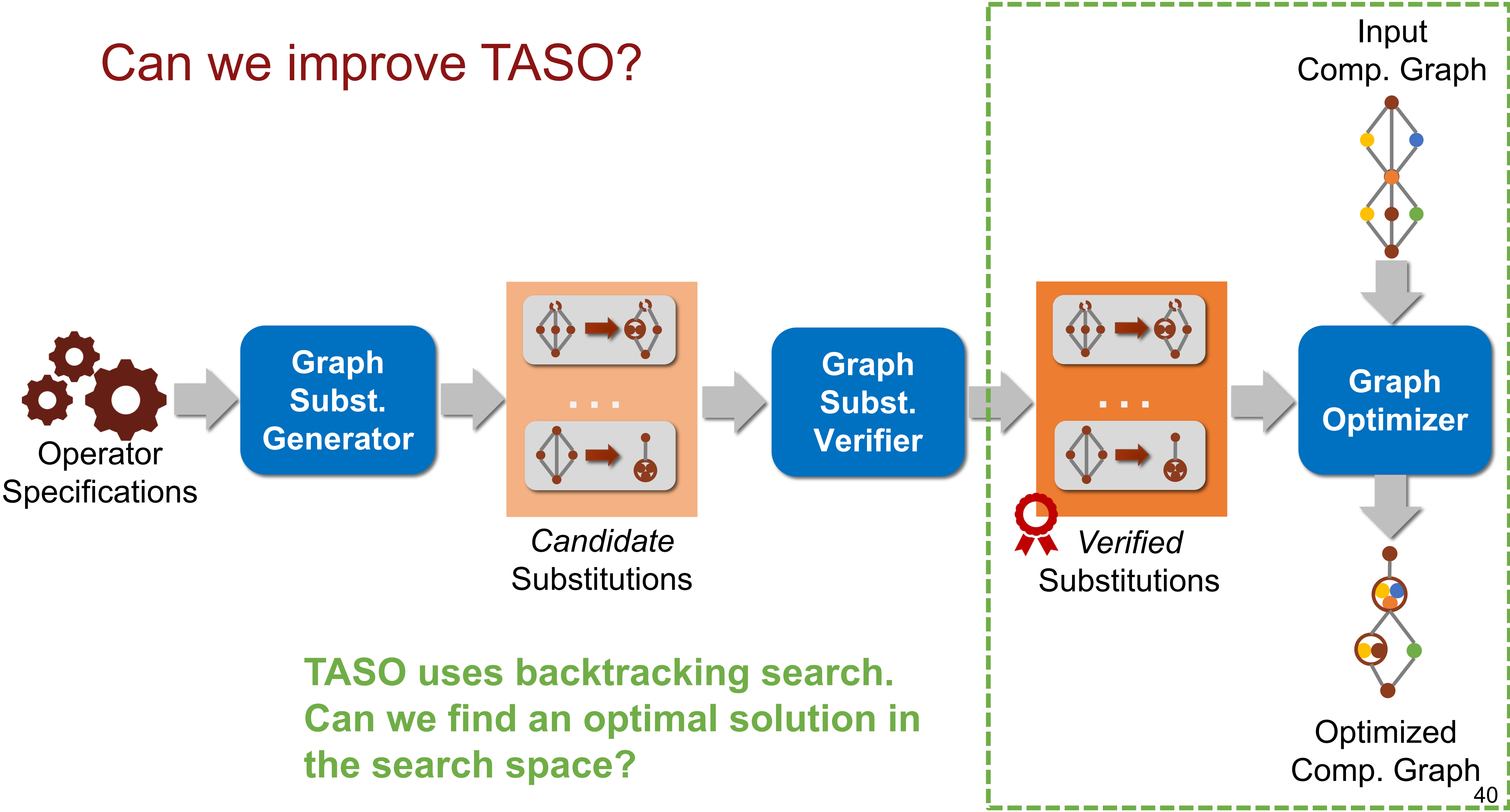
Generate both fully and partially equivalent substs

Auto-correct substs to maintain end-to-end equivalance

Optimize DNNs using both fully and partially equivalent substs

Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections. H. Wang et al.

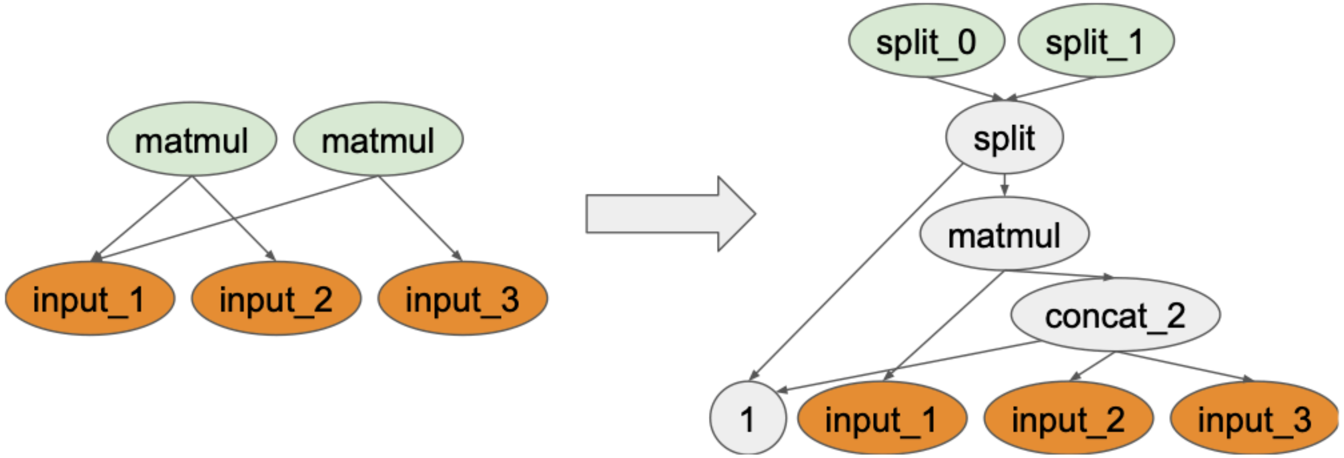
Can we improve TASO?



TASO uses backtracking search. Can we find an optimal solution in the search space?

Equality Saturation for TASO

- Key idea: a new representation that can express all possible computation graphs at once



Source: (matmul ?input$_1$?input$_2$), (matmul ?input$_1$?input$_3$)
Target: (split$_0$ (split 1 (matmul ?input$_1$ (concat$_2$ 1 ?input$_2$?input$_3$)))),

(split$_1$ (split 1 (matmul ?input$_1$ (concat$_2$ 1 ?input$_2$?input$_3$))))

The optimal graph is **16%** better.
The search is **48x** faster.

EQUALITY SATURATION FOR TENSOR GRAPH SUPEROPTIMIZATION. Y. Yang et al.

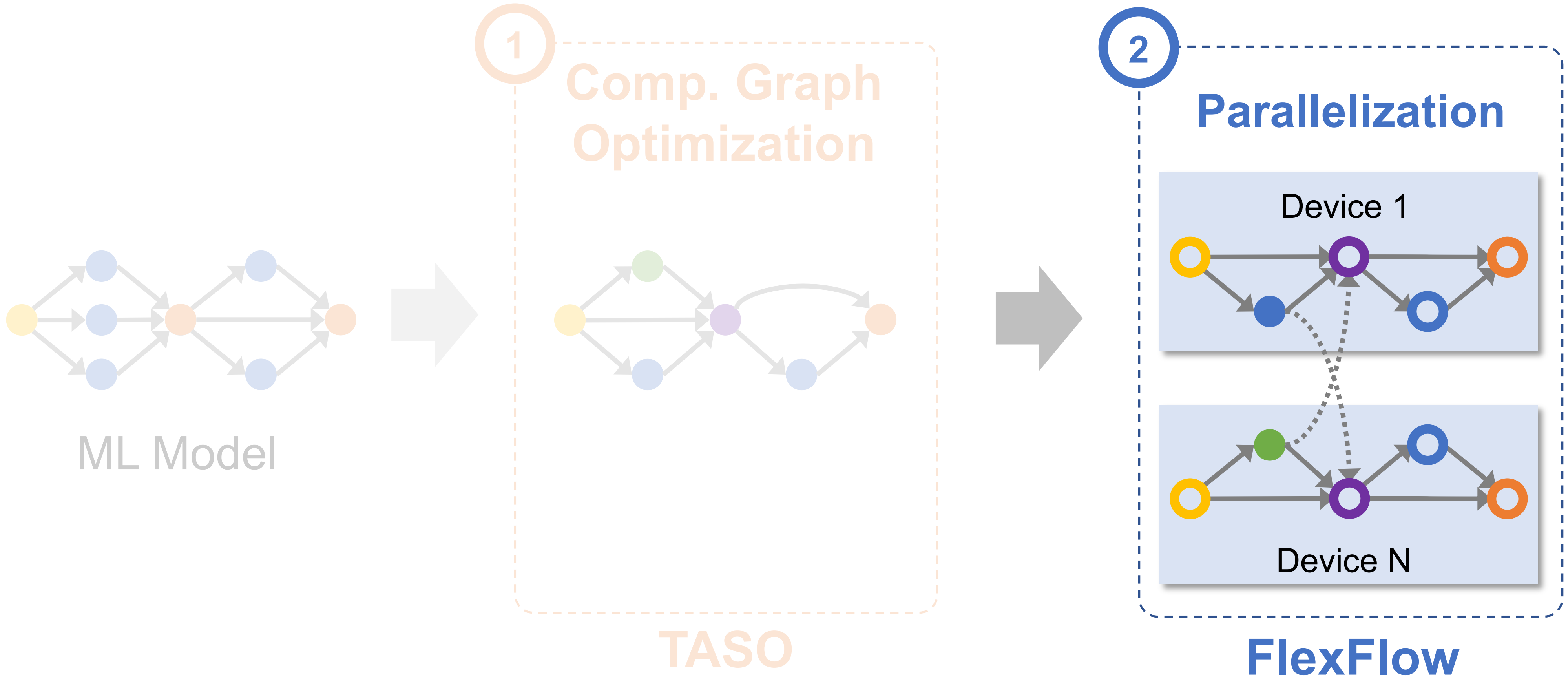
Can we apply TASO to other problem domains?

- **Cross-optimizations between ML and DB operations**
  - Why: ML and DB operations are optimized separately in today's DB systems
  - How: automatically generate co-optimizations of linear algebra and relational algebra operations

- **Optimizing Compilers for Quantum Computing**
  - Why: today's quantum machines support different sets of instructions -> impossible to manually design optimizations for all quantum architectures
  - How: automatically generate quantum program transformations given a set of instructions

- **Others?**

Automated Discovery of ML Optimizations

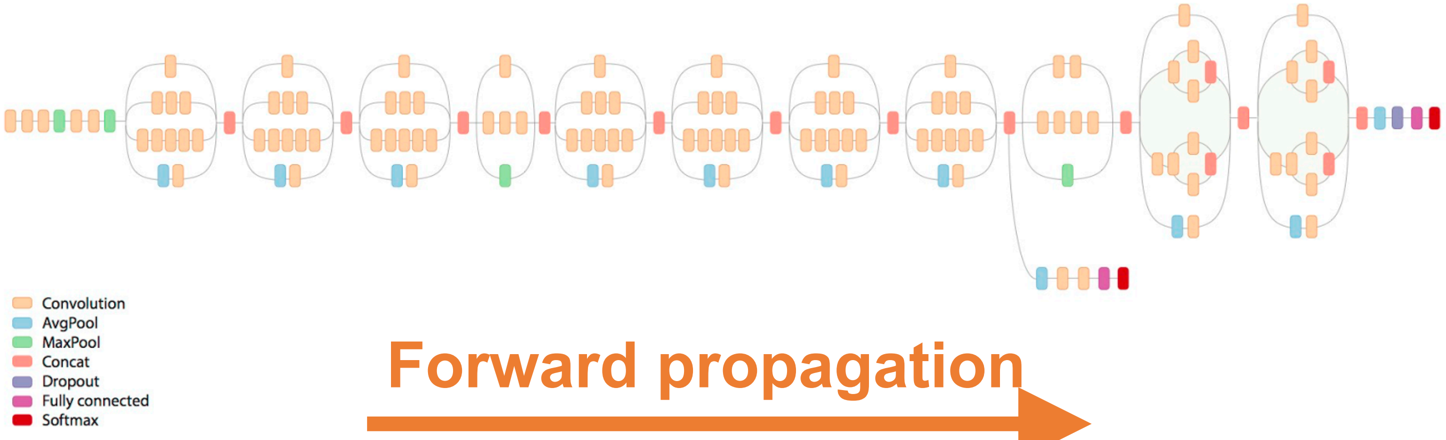


Stochastic Gradient Descent (SGD)

Train ML models through many iterations of 3 stages

1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce error for each trainable weight
3. **Weight update**: use the loss value to update model weights

Model inputs



Model prediction

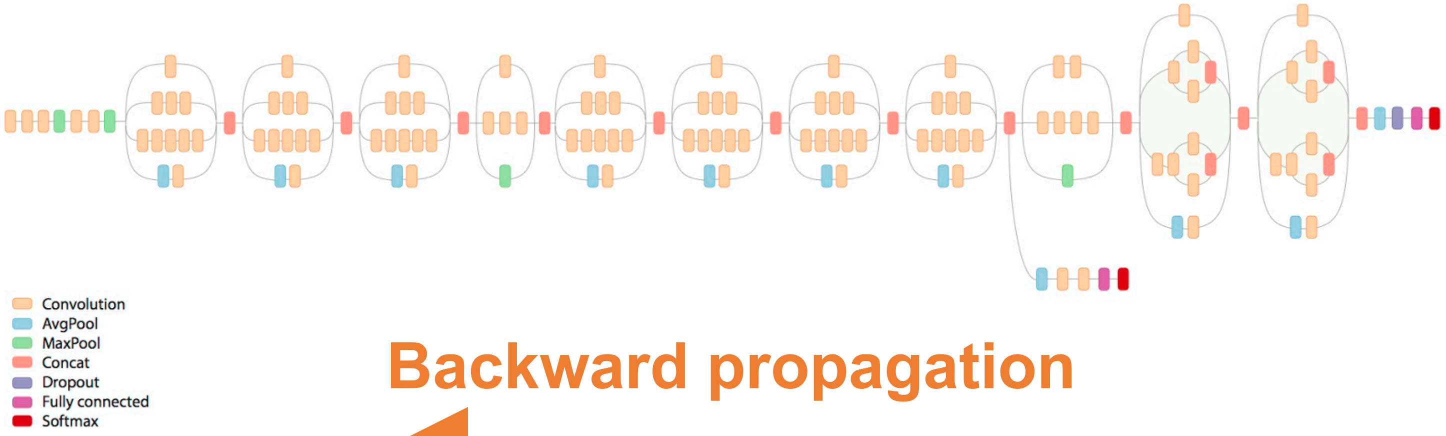
Forward propagation

Stochastic Gradient Descent (SGD)

Train ML models through many iterations of 3 stages

1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce error for each trainable weight
3. **Weight update**: use the loss value to update model weights

Model inputs



Model prediction

Backward propagation

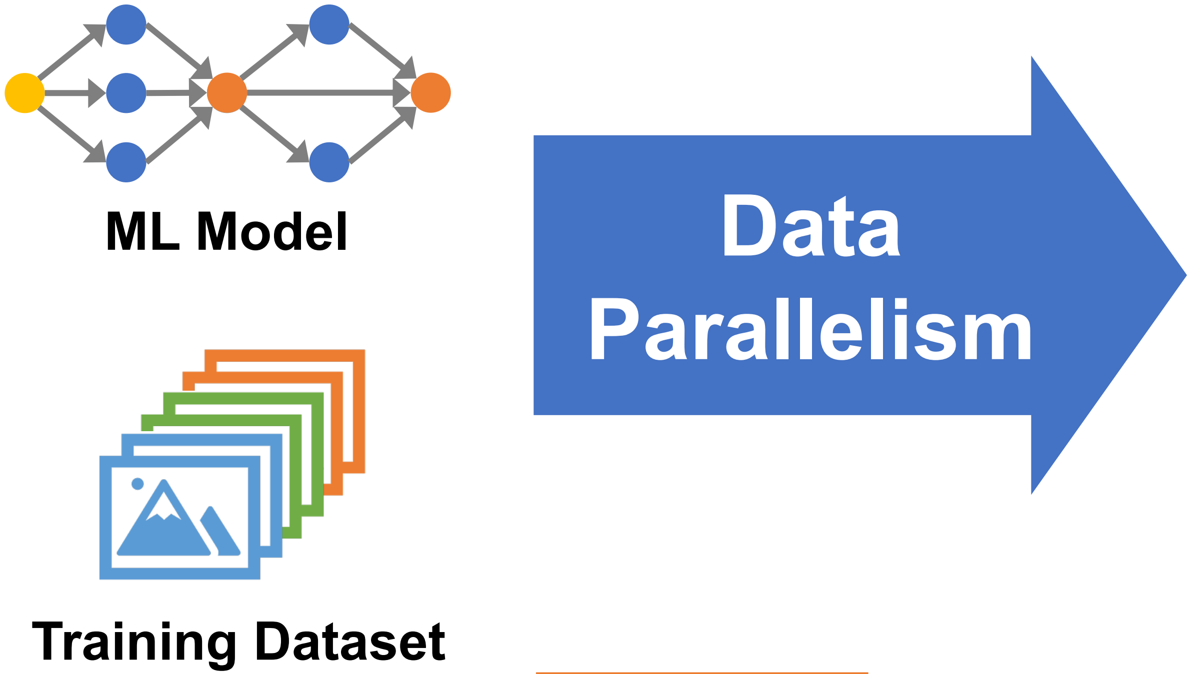
Stochastic Gradient Descent (SGD)

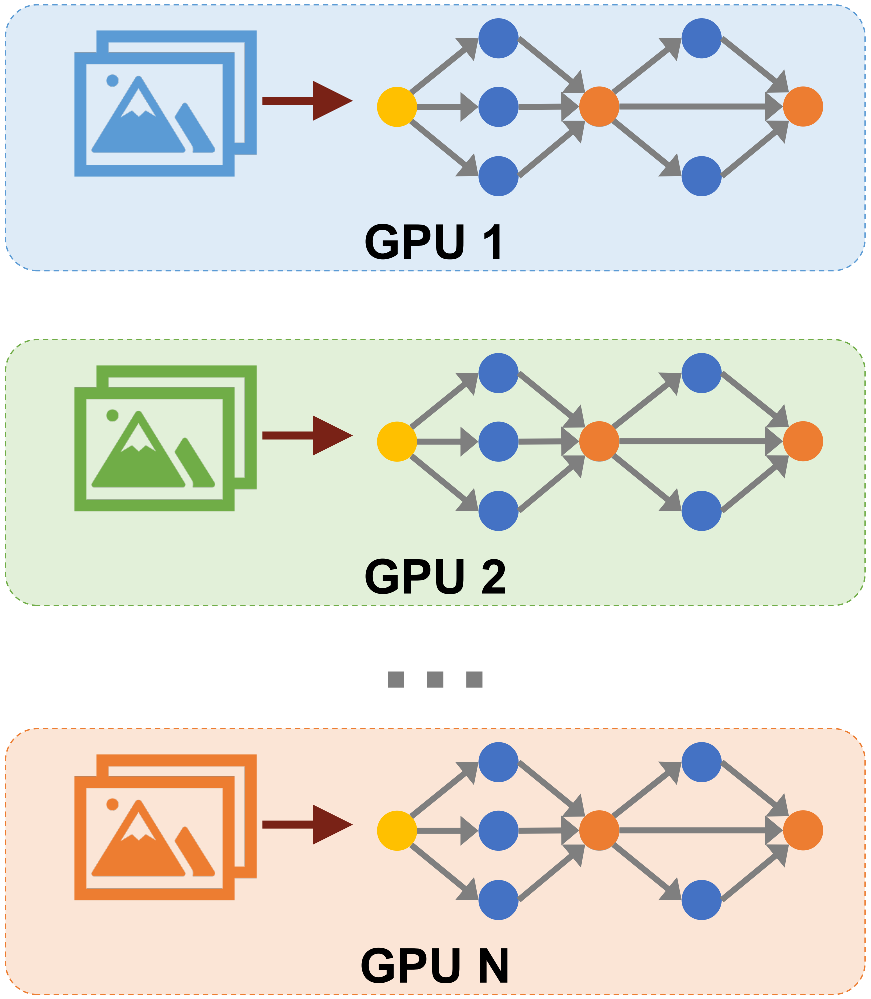
Train ML models through many iterations of 3 stages

1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce error for each trainable weight
3. **Weight update**: use the loss value to update model weights

$$w_i \coloneqq w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \sum_{j=1}^{n} \nabla L_j(w_i)$$

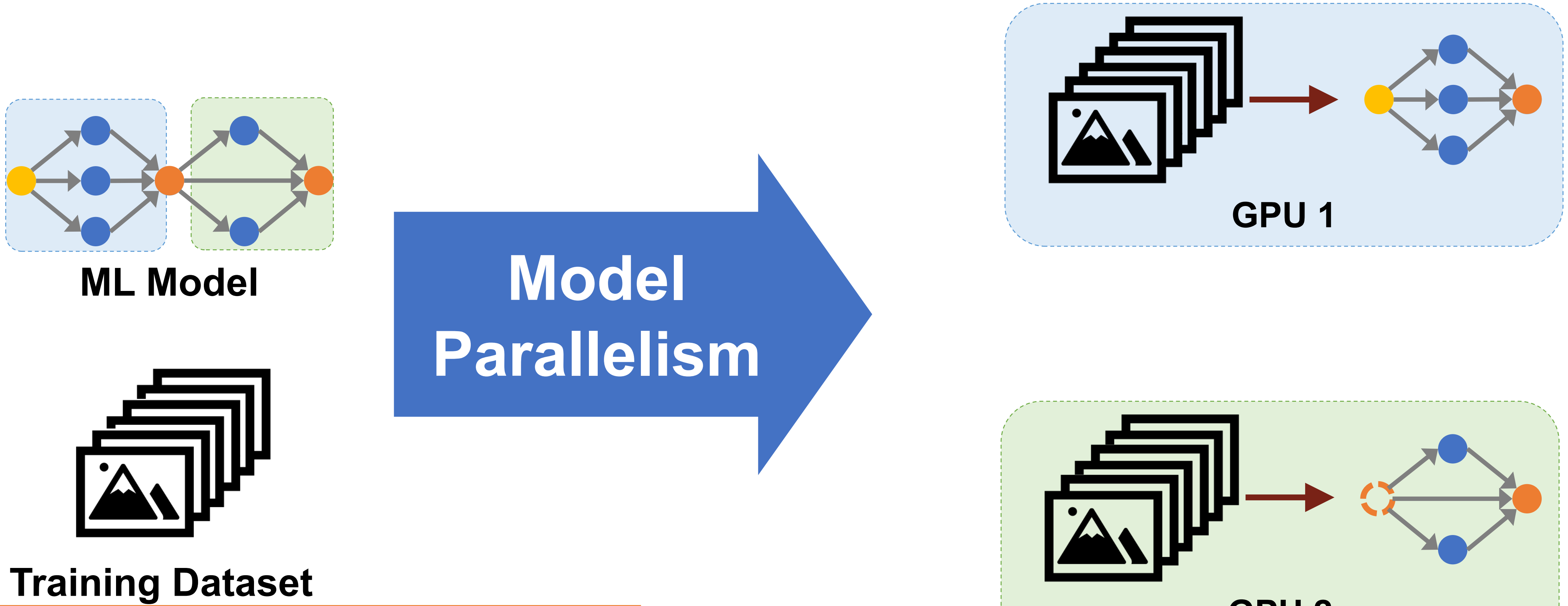
Current Strategies to Parallelize ML Training: Data and Model Parallelism





$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \boxed{\sum_{j=1}^{n} \nabla L_j(w_i)}$$

Current Strategies to Parallelize ML Training: Data and Model Parallelism



$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \sum_{j=1}^{n} \nabla L_j(w_i)$$

Are there strategies beyond data/model parallelism?
Can we discover fast ones automatically?

FlexFlow: Automated Search for Fast Strategies

Define a **search space** of possible parallelization strategies

\+

A **cost model** and a **search algorithm**

=

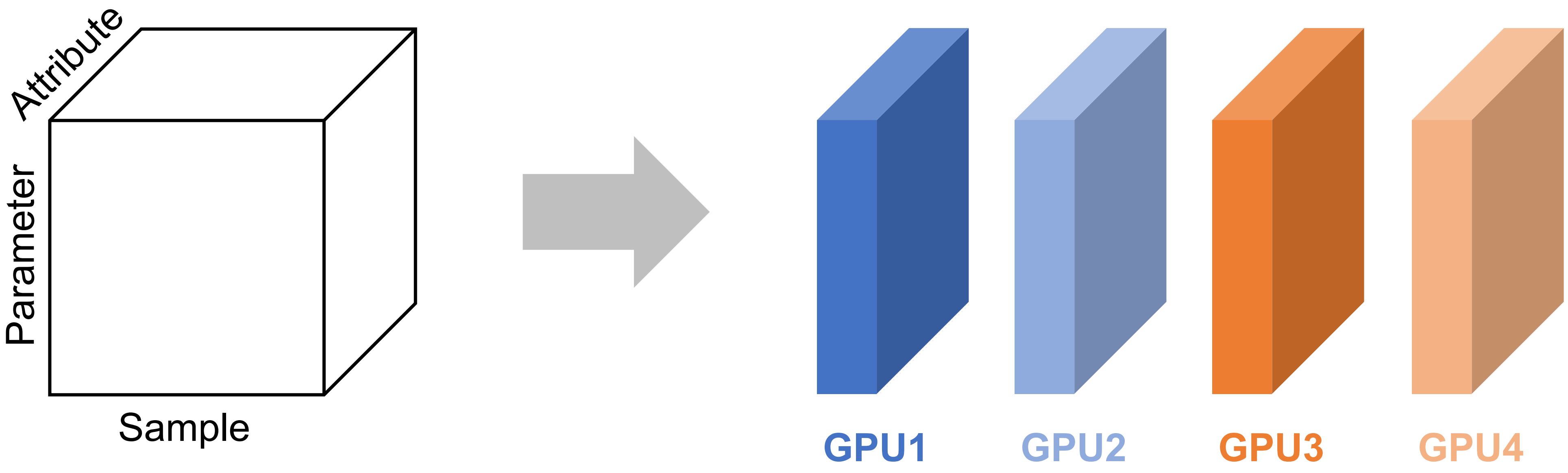
Optimized Parallelization strategies

The SOAP Search Space

- **S**amples
- **O**perators
- **A**ttributes
- **P**arameters

The SOAP Search Space

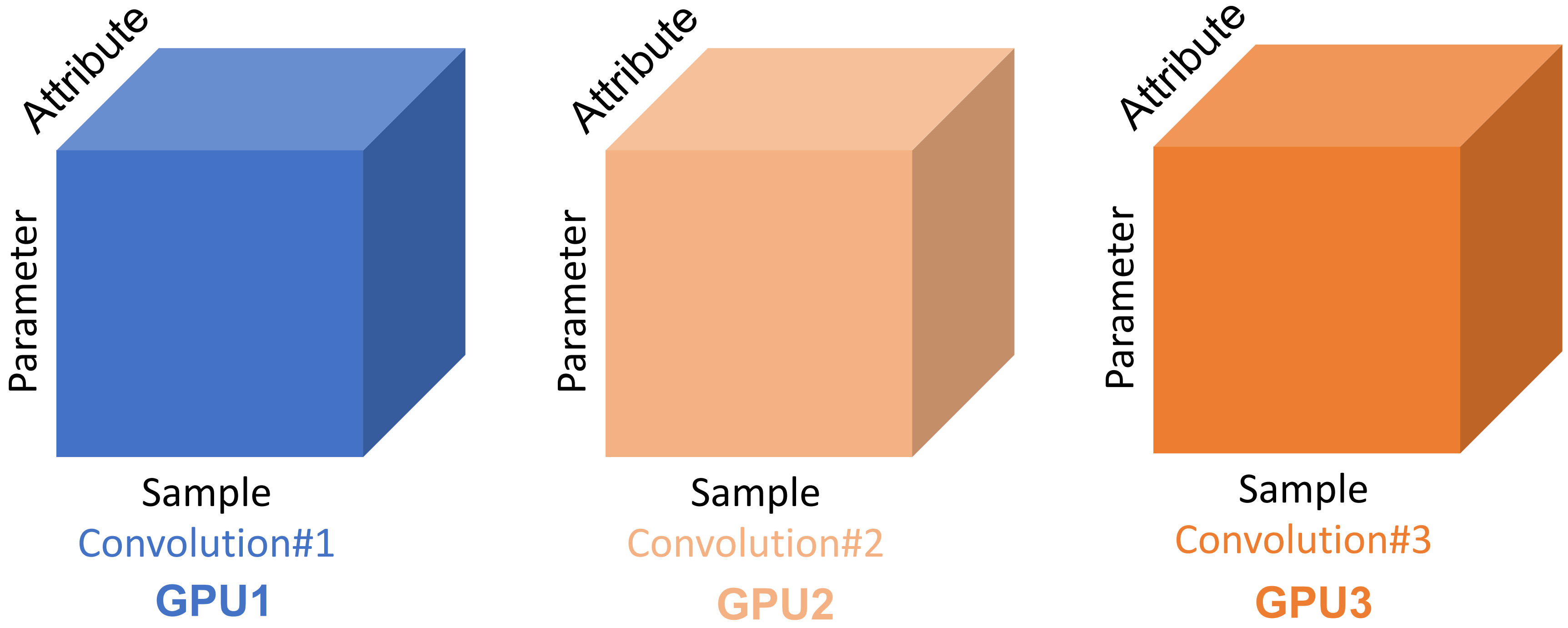
- **S**amples: partitioning training samples (Data Parallelism)
- **O**perators
- **A**ttributes
- **P**arameters



Parallelizing a 1D convolution in ***Sample***

The SOAP Search Space

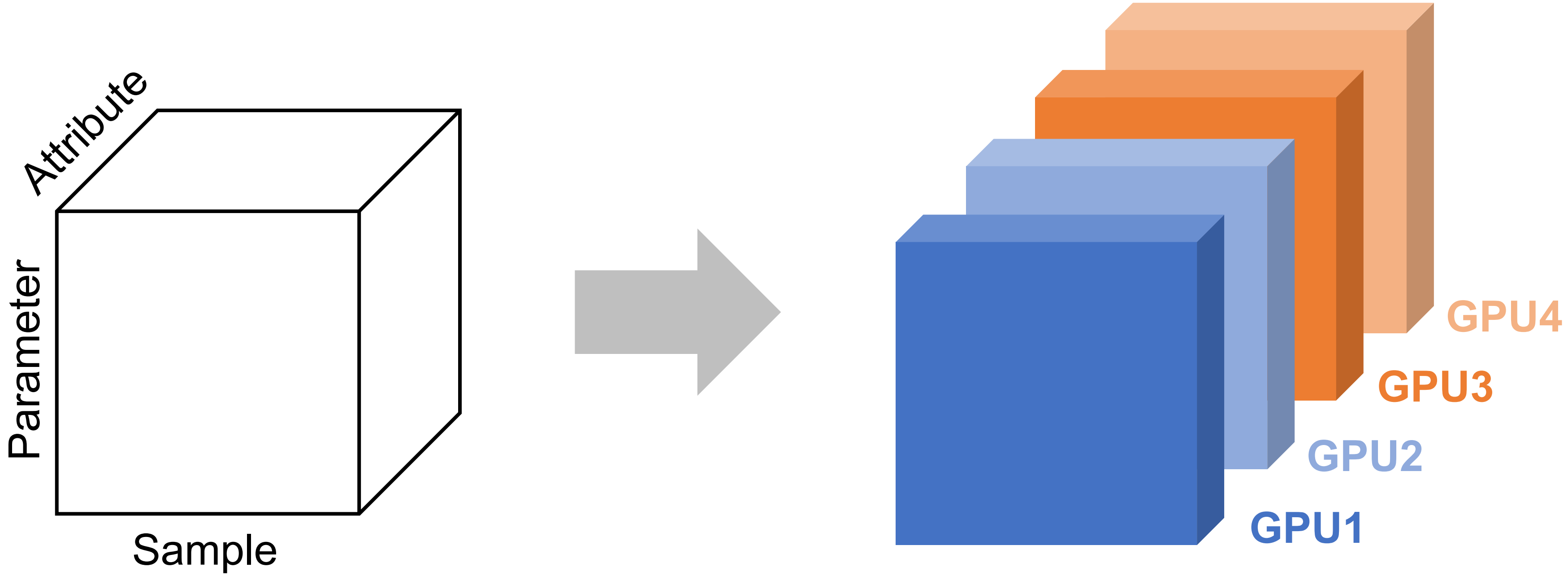
- **S**amples: partitioning training samples (Data Parallelism)
- **O**perators: partitioning ML operators (Model Parallelism)
- **A**ttributes
- **P**arameters



Parallelizing multiple convolutions in ***Operator***

The SOAP Search Space

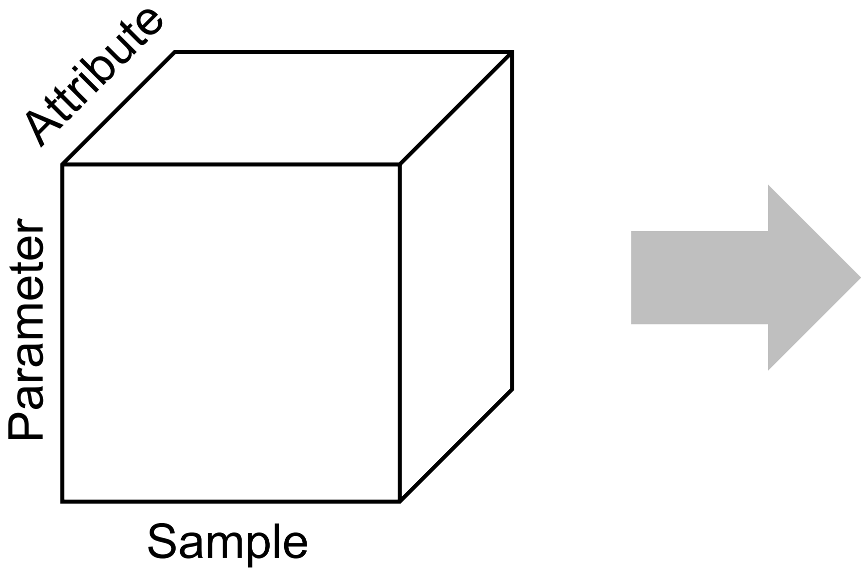
- **S**amples: partitioning training samples (Data Parallelism)
- **O**perators: partitioning ML operators (Model Parallelism)
- **A**ttributes: partitioning attributes in a sample (e.g., pixels)
- **P**arameters

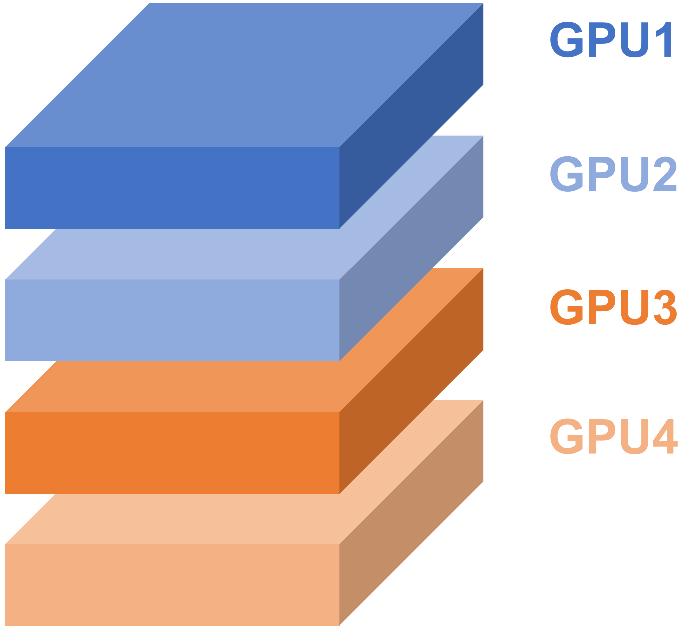


Parallelizing a 1D convolution in ***Attribute***

The SOAP Search Space

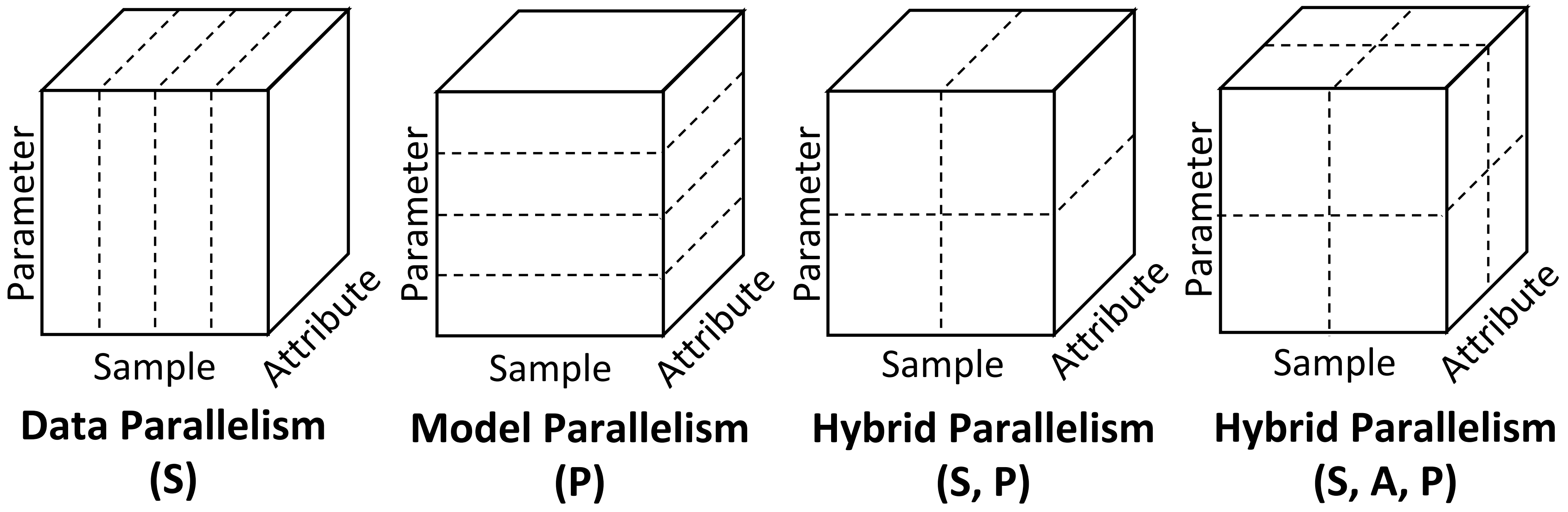
- **S**amples: partitioning training samples (Data Parallelism)
- **O**perators: partitioning ML operators (Model Parallelism)
- **A**ttributes: partitioning attributes in a sample (e.g., pixels)
- **P**arameters: partitioning parameters in an operator





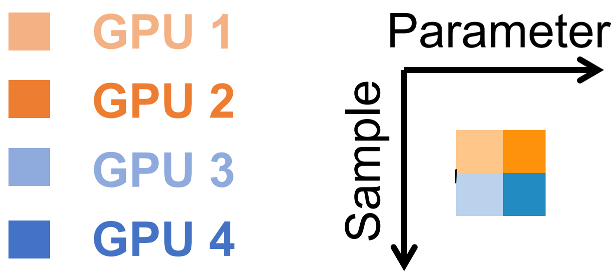
Parallelizing a 1D convolution in ***Parameter***

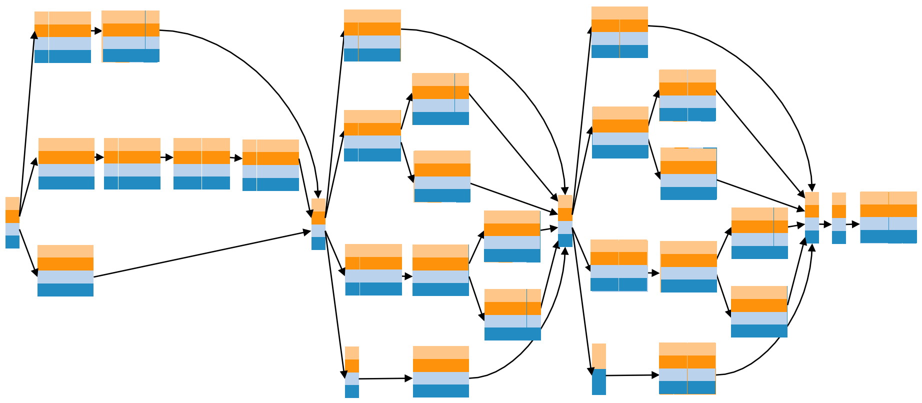
Hybrid Parallelism in SOAP



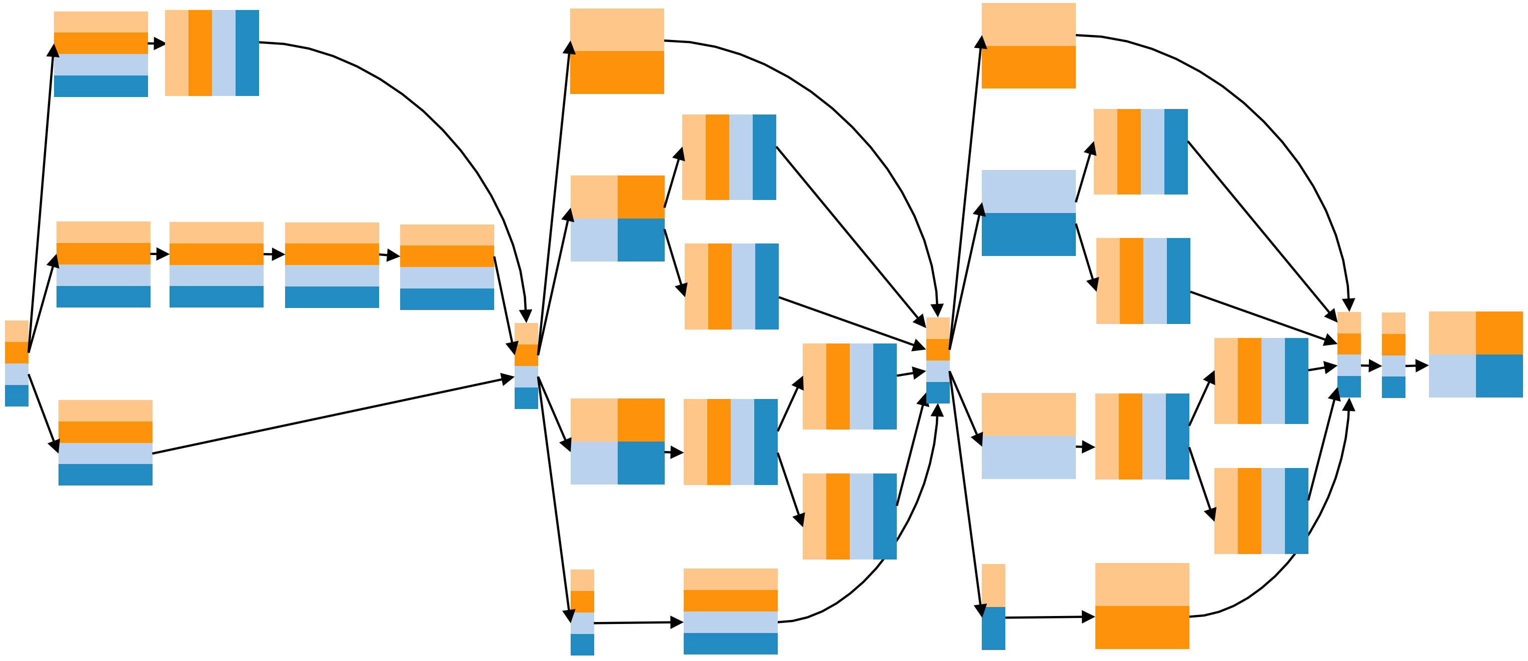
Example parallelization strategies for 1D convolution

Different strategies perform the same computation.

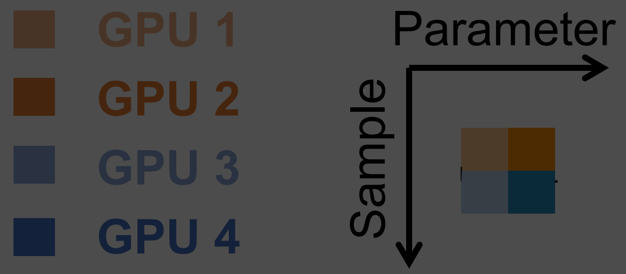


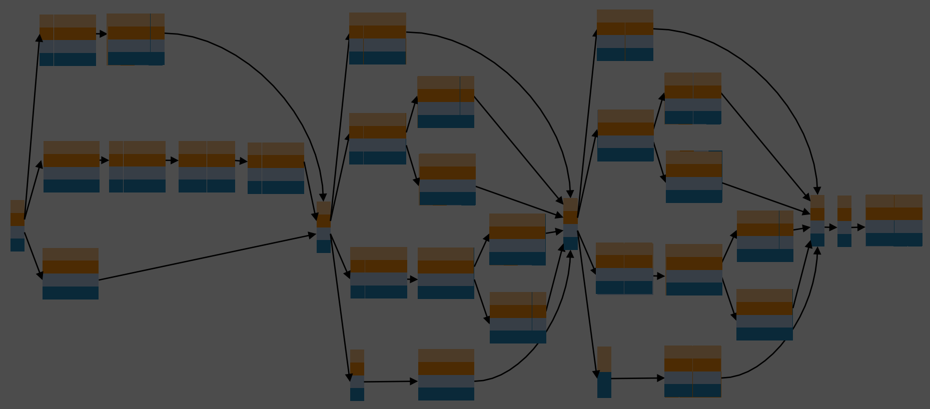


Data parallelism

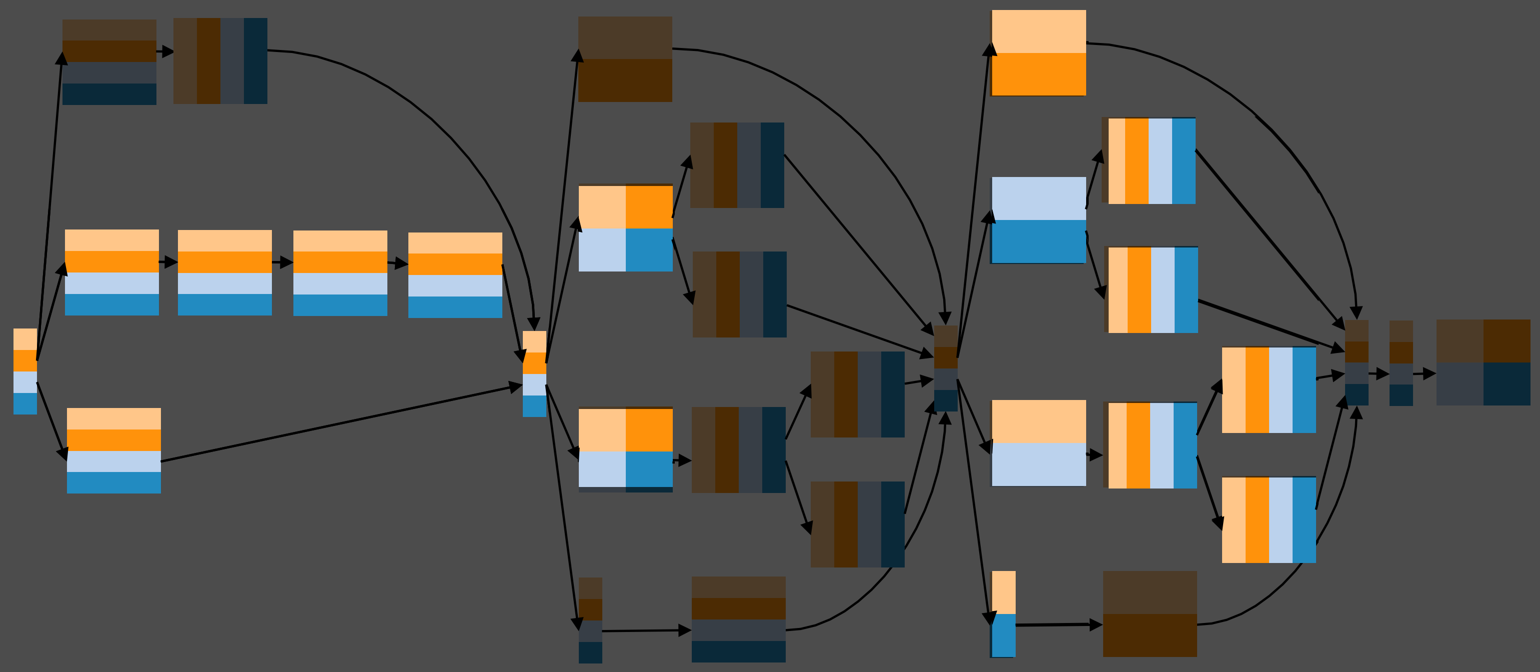


A parallelization strategy in SOAP **(1.2x faster)**





Data parallelism



A parallelization strategy in SOAP **(1.2x faster)**

Challenges of Discovering Fast Strategies in SOAP

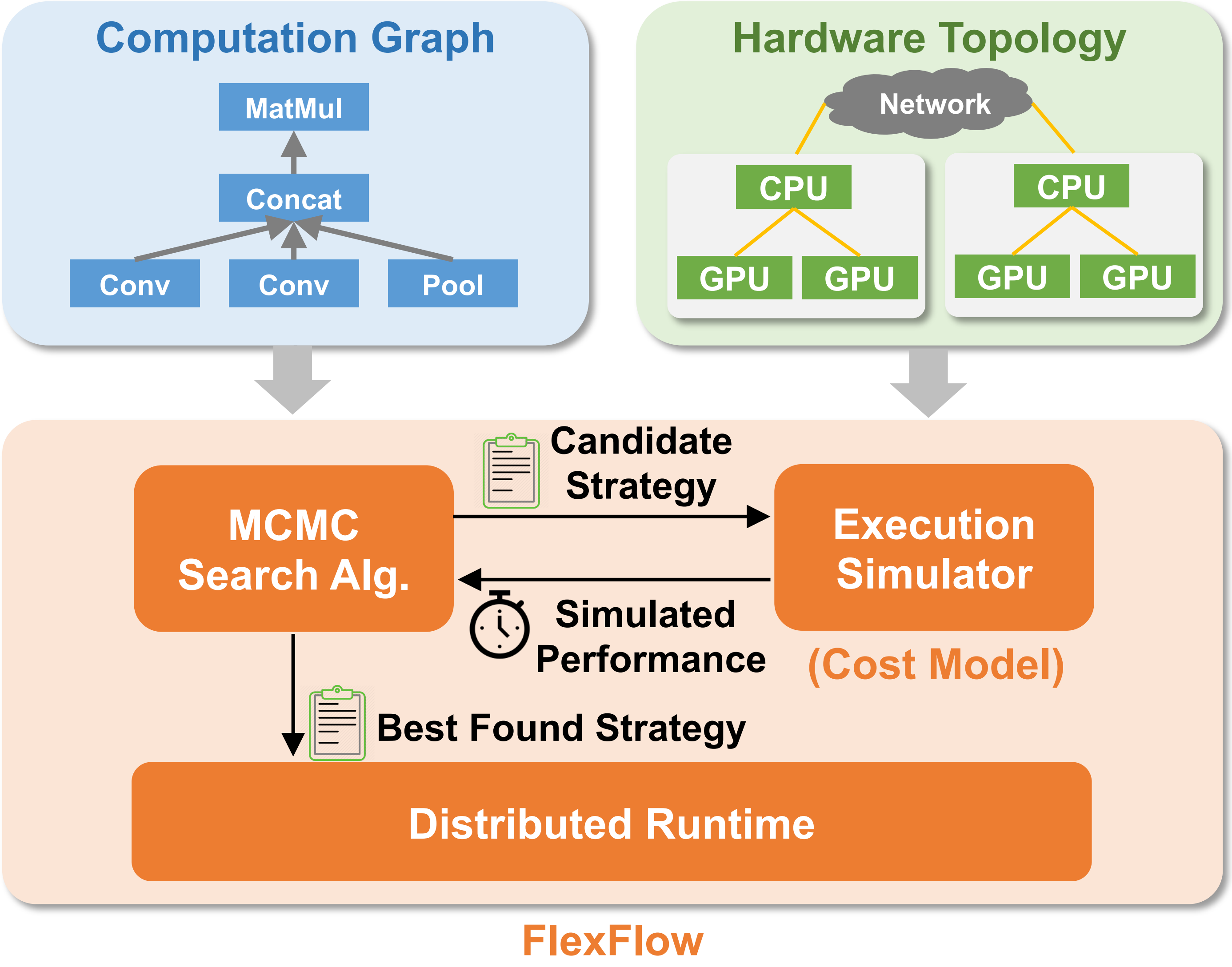
1. SOAP contains billions or more possible strategies

MCMC search algorithm

2. Evaluating a strategy on hardware is too slow

Execution simulator

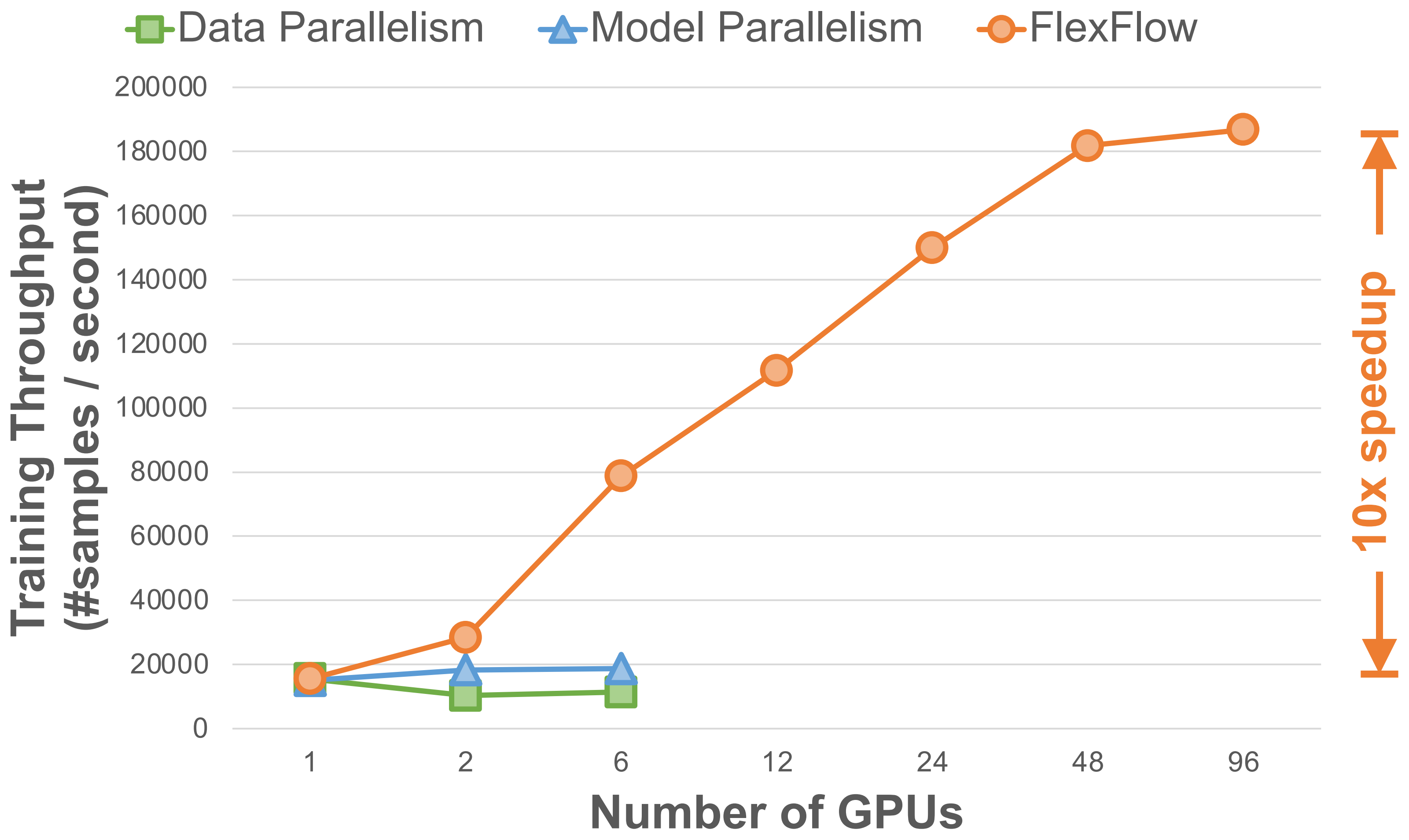
FlexFlow Overview



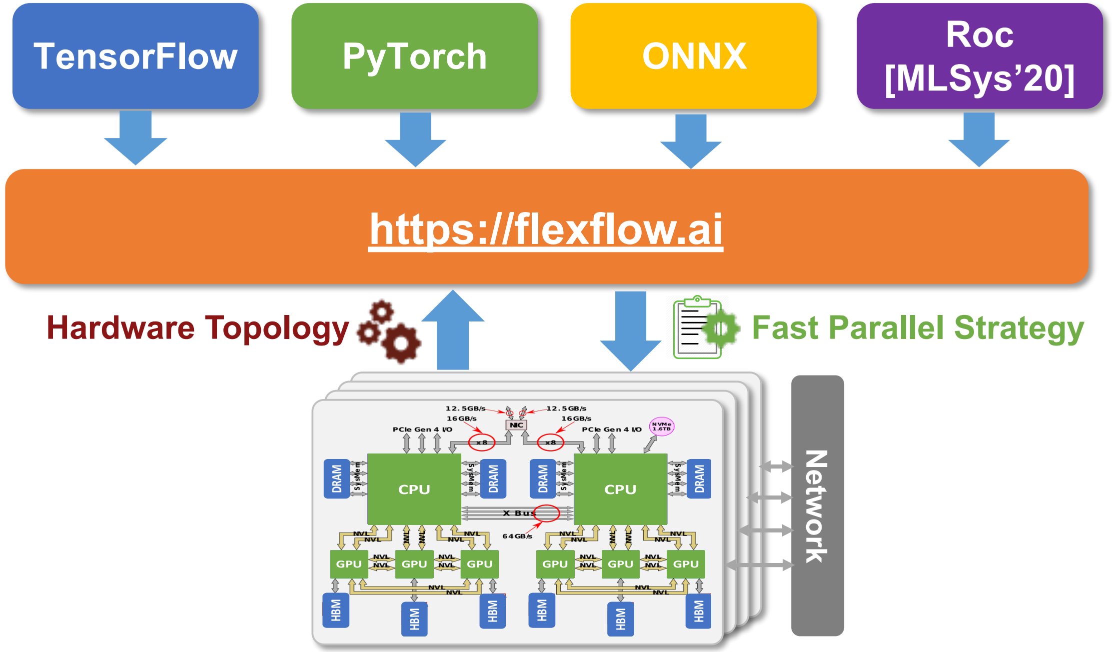
Deep Learning Recommendation Model (DLRM)

facebook

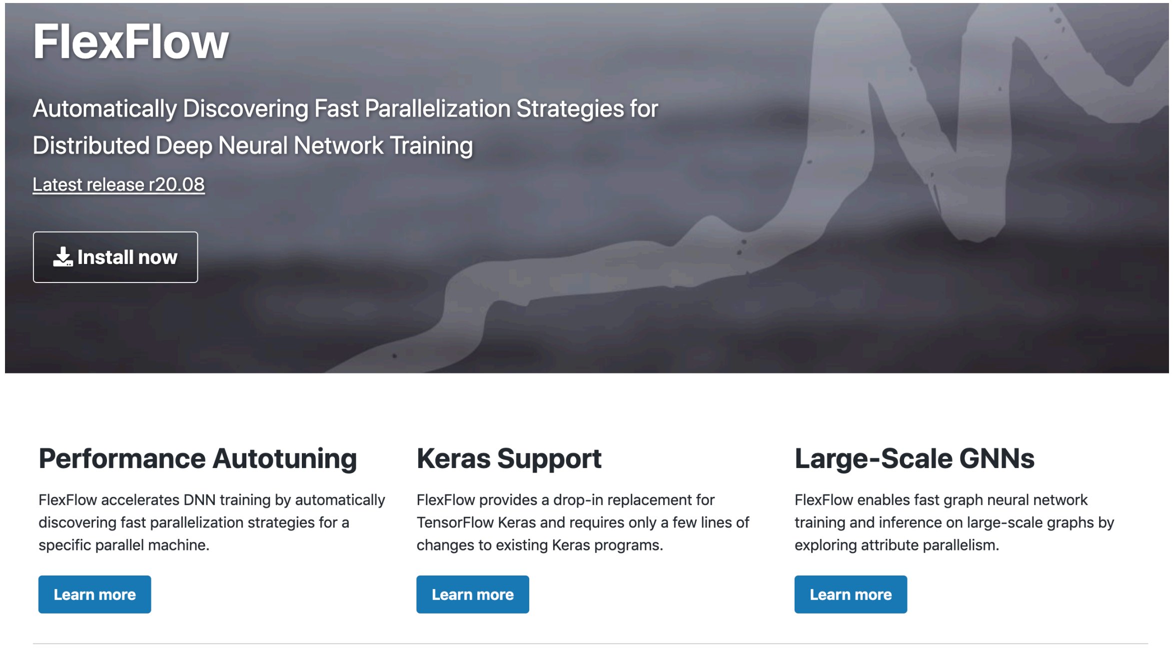
A deep learning model for ads recommendation



FlexFlow: Automatically Discover Fast Parallelization for DNNs

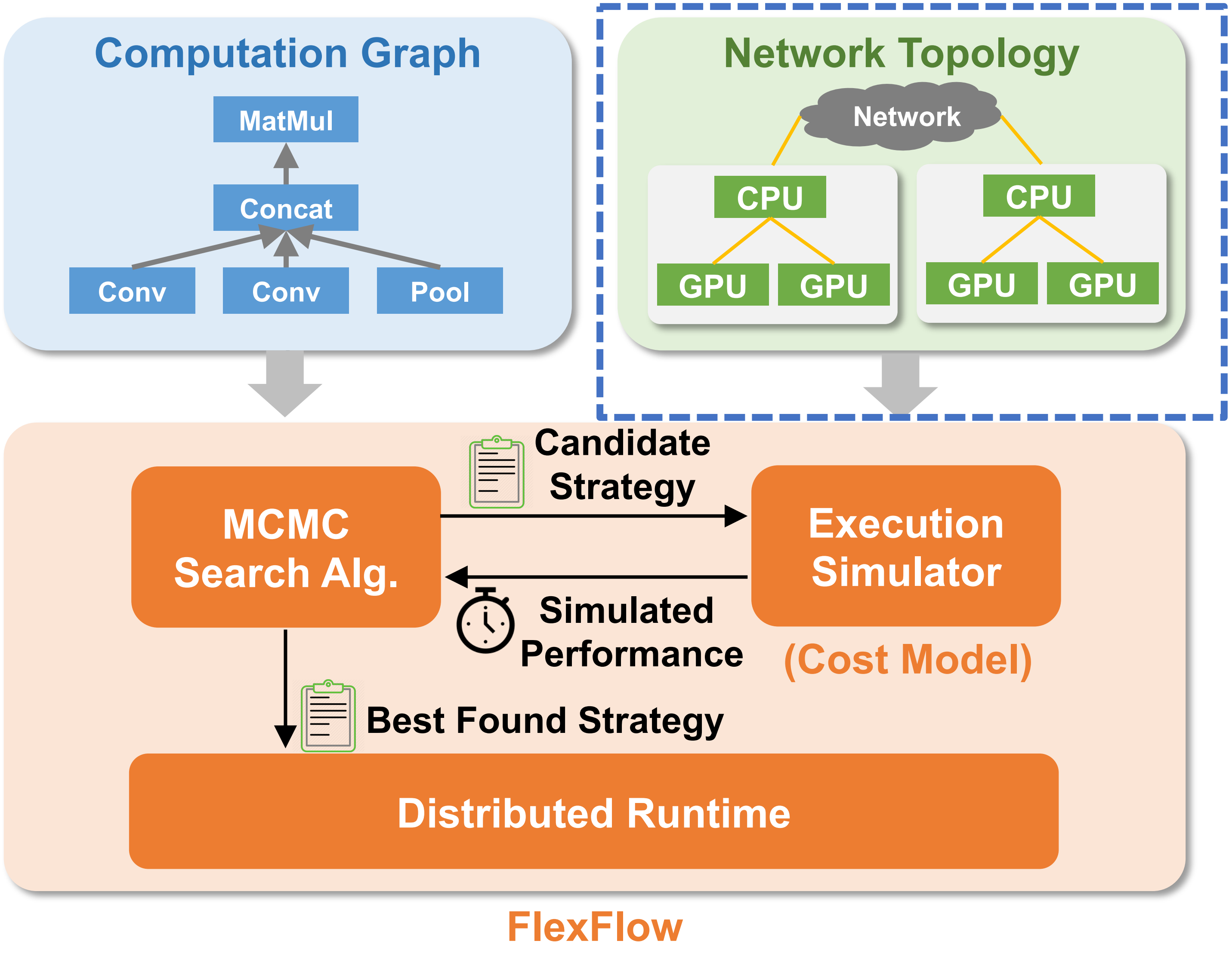


https://flexflow.ai



Los Alamos NATIONAL LABORATORY EST.1943
Carnegie Mellon University
perspecta LABS
catalyst

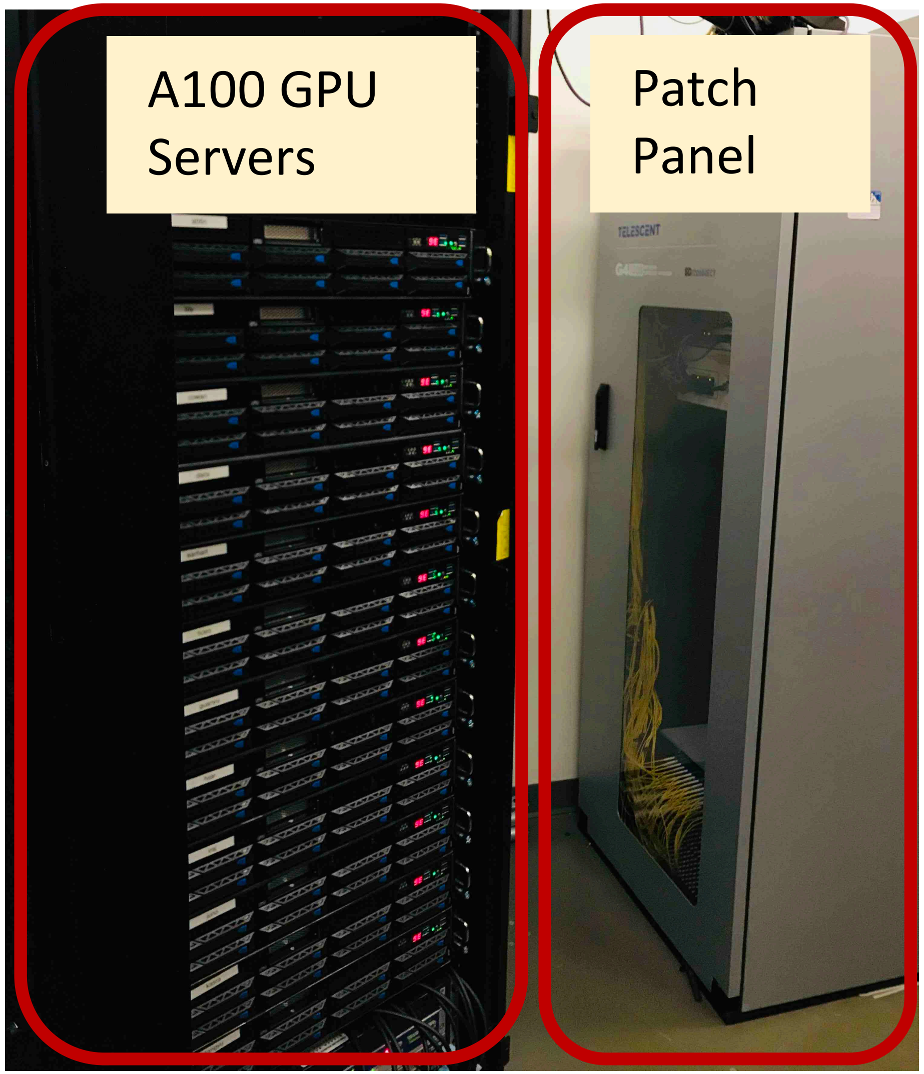
Can we improve FlexFlow?

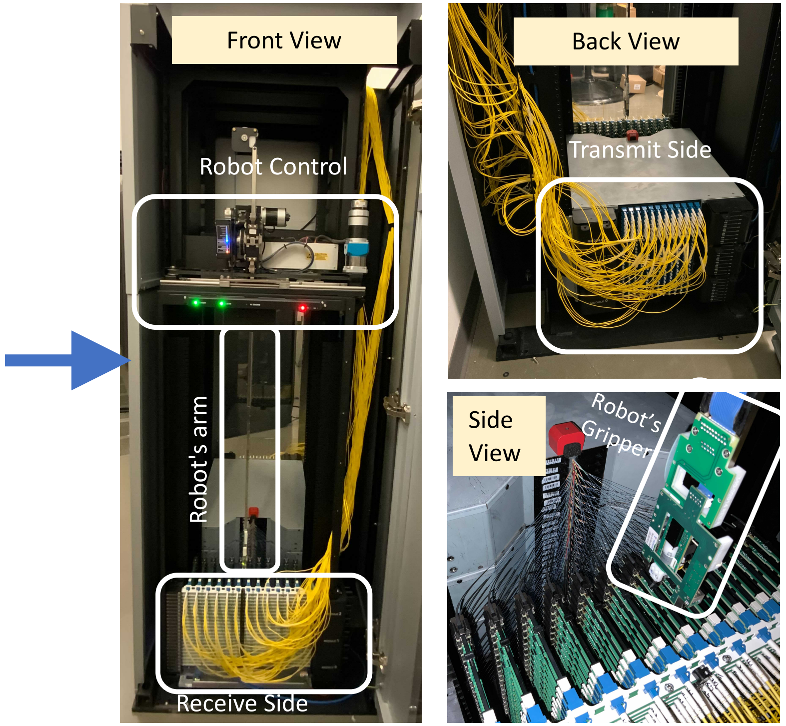


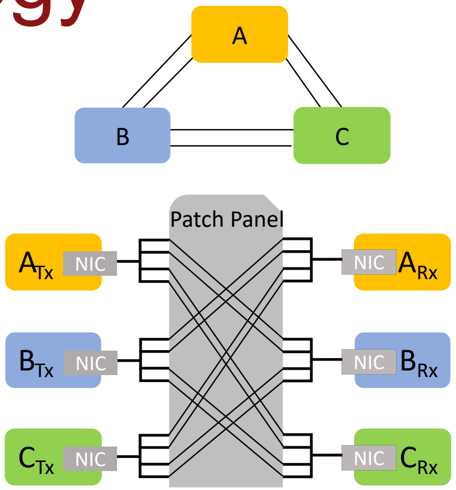
FlexFlow takes network topology as an input.

Can we co-optimize network topology and parallelization strategies?

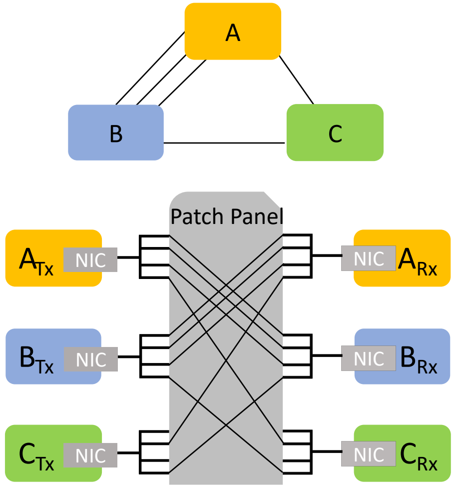
Testbed with Reconfigurable Network Topology







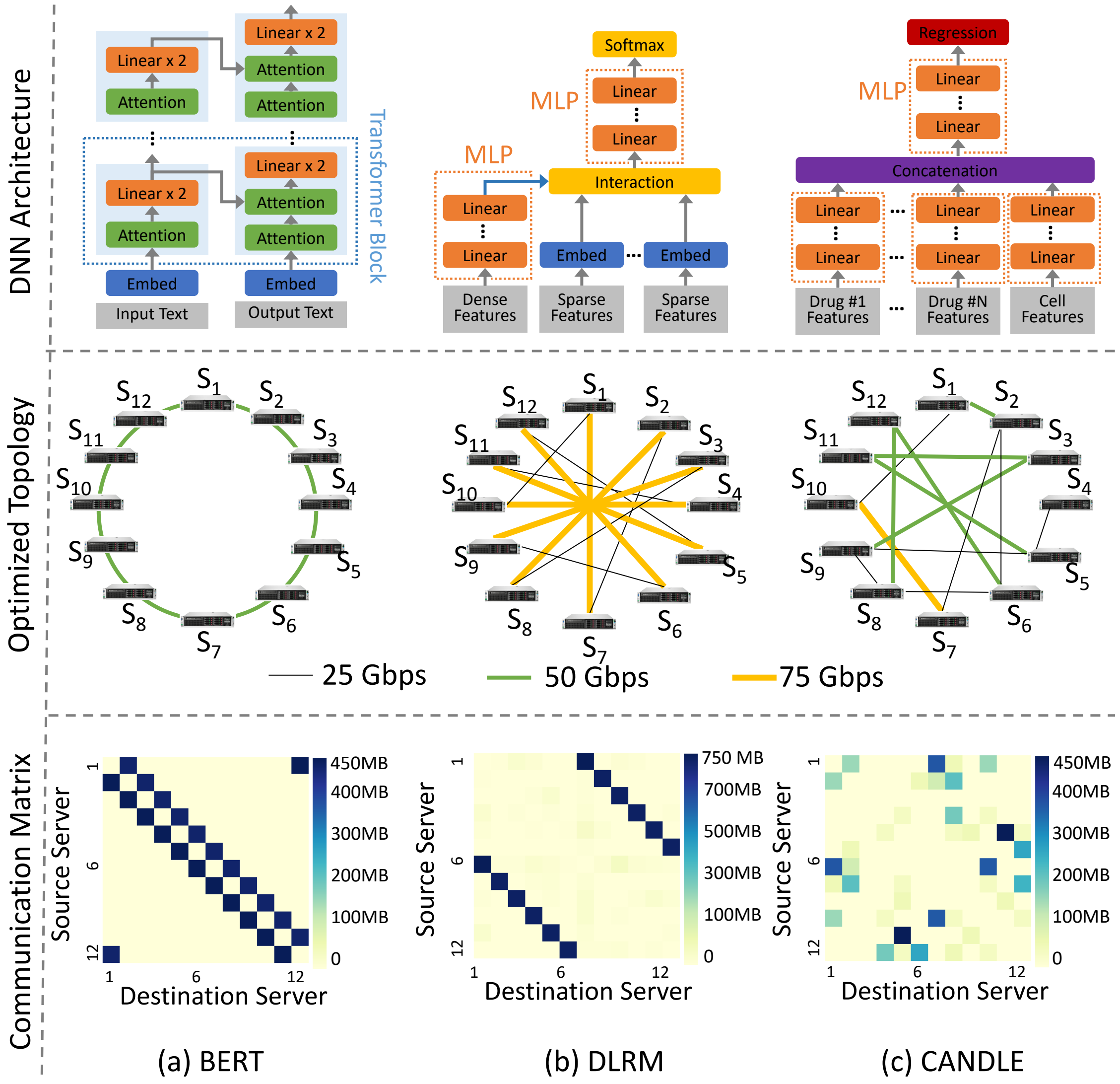
(a)



(b)

Reconfigurable network topology

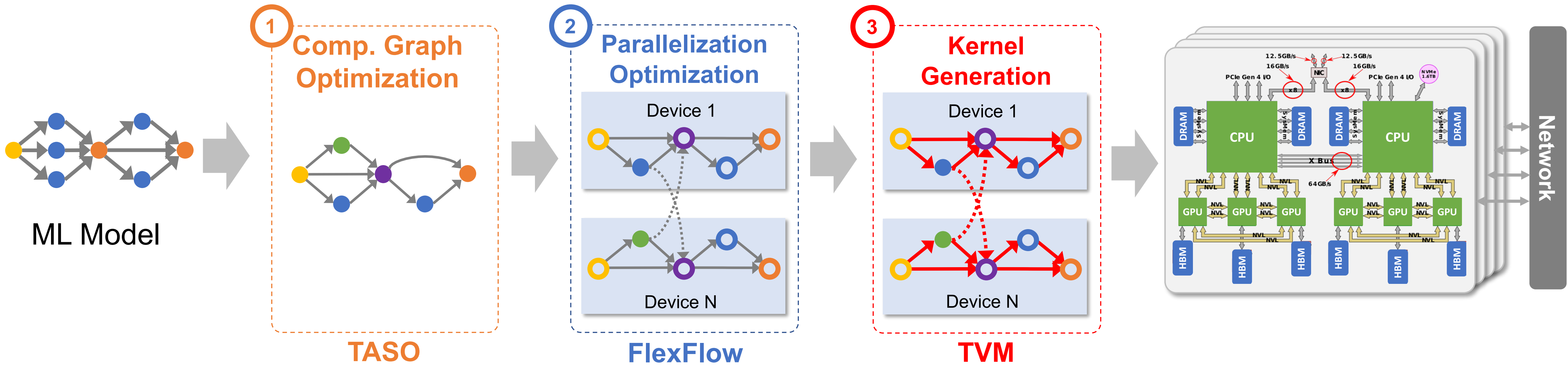
Joint Optimization of Parallelization Strategy and Network Topology



Up to 5.7x faster than FlexFlow w/ Fat-tree interconnect

Optimizing the Network Topology for Distributed DNN Training. W. Wang et al.

Automated Machine Learning Systems



catalyst

https://catalyst.cs.cmu.edu/