Automatically Discovering ML Optimizations

Zhihao Jia

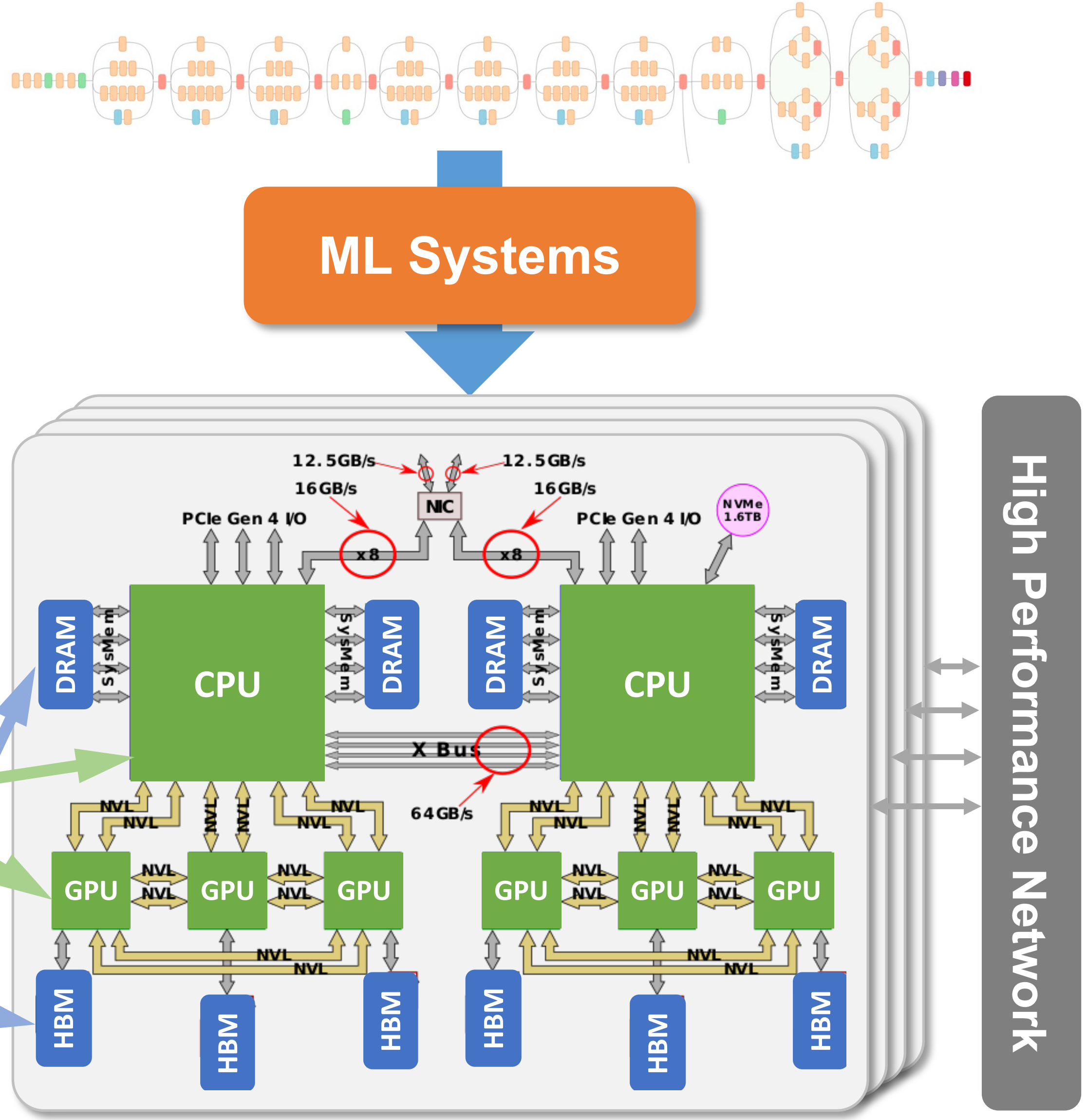
Computer Science Department

Carnegie Mellon University

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

My Research: Systems for ML

ML Model



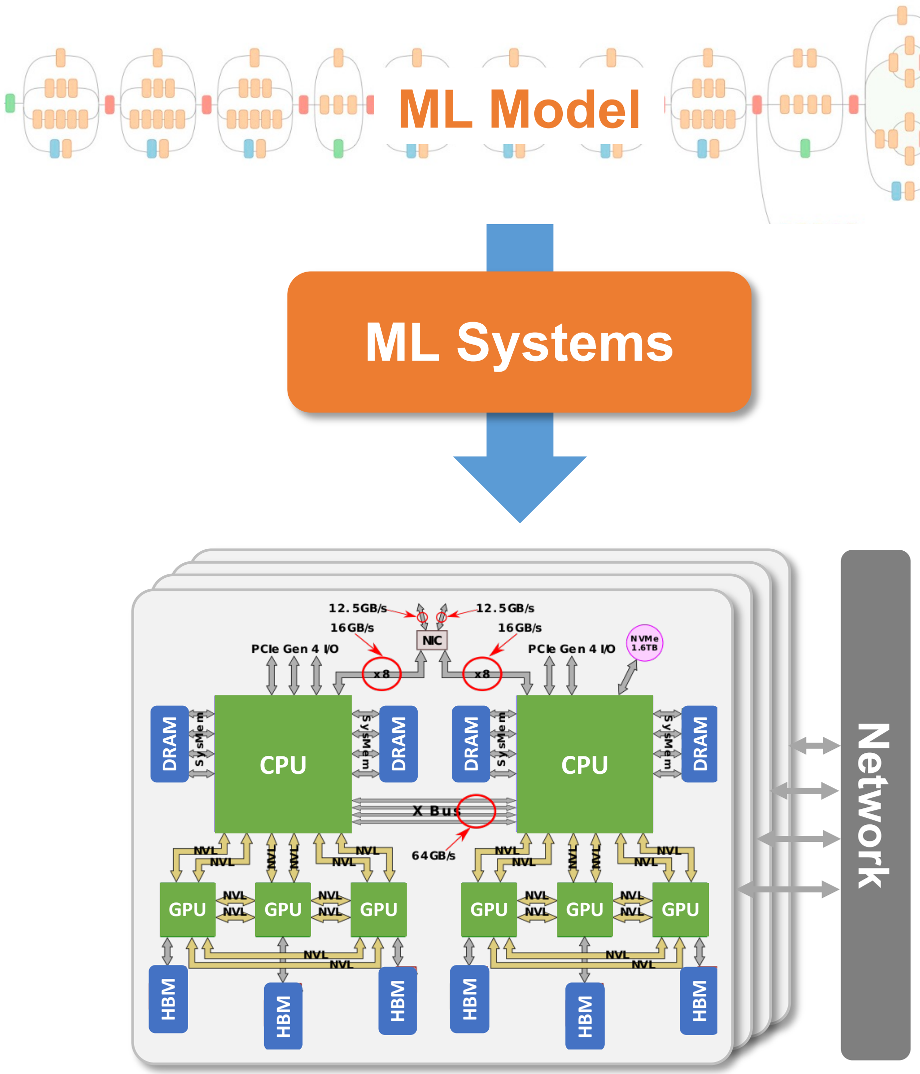
Distributed Heterogenous Hardware Architectures

Heterogenous Processors

Heterogenous Memories

~4,600 compute nodes

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

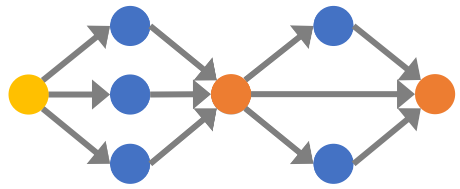
Mission: Automate the design and optimization of ML systems by leveraging

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

catalyst

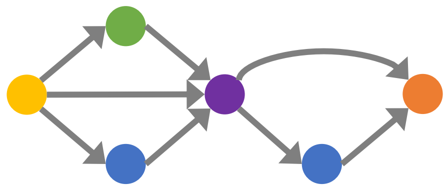
https://catalyst.cs.cmu.edu/

Our Research: Automated Discovery of ML Optimizations

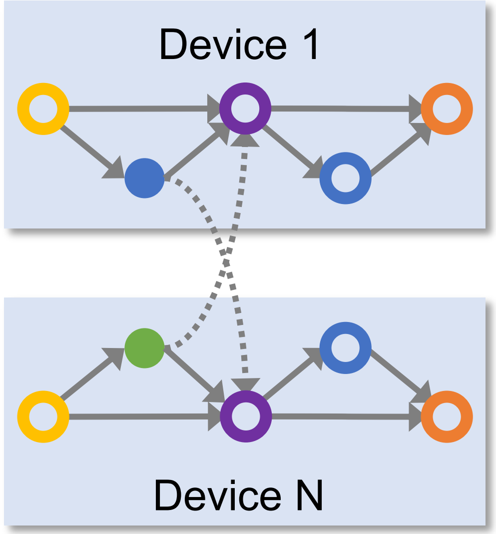


ML Model

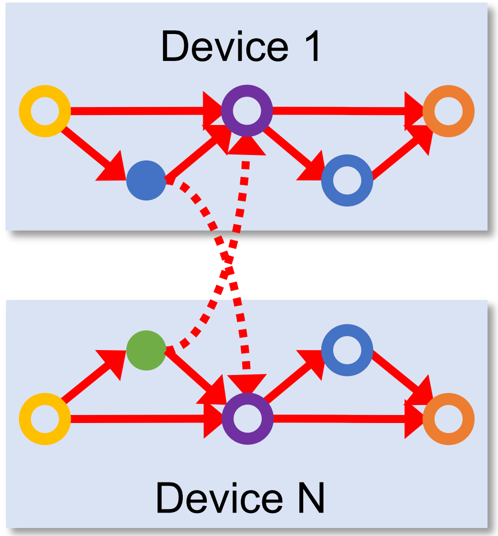
1 **Comp. Graph Optimization**



2 **Parallelization Optimization**



3 **Data Layout & Placement**



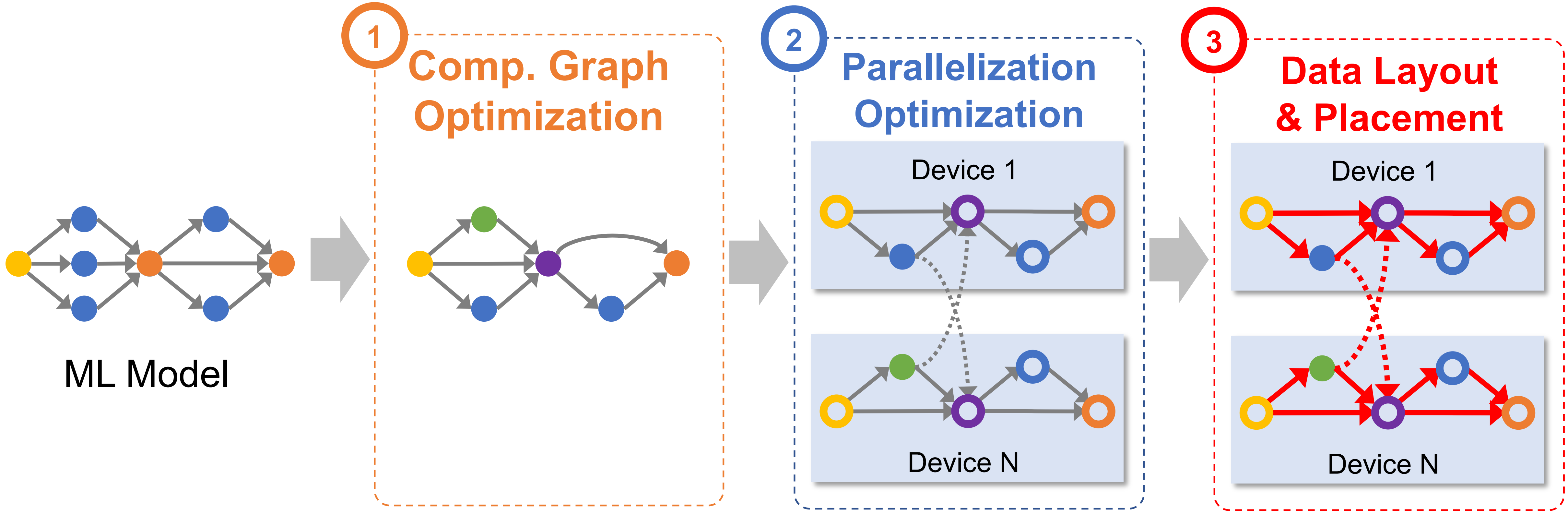
TASO [SOSP'19]
PET [OSDI'21]
Automated generation of graph optimizations

FlexFlow [MLSys'19]
Unity [OSDI'22]
Automated discovery of fast parallelization strategies

Lux [VLDB'18]
Roc [MLSys'20]
Automated data layout and placement optimizations

catalyst

Lesson 1: Automated Approaches Offer 3-10x Improvement



TASO: Up to 3.1x
PET: Up to 2.5x

FlexFlow: Up to 10x
Unity: Up to 3.6x

Lux: Up to 10x
Roc: Up to 4.1x

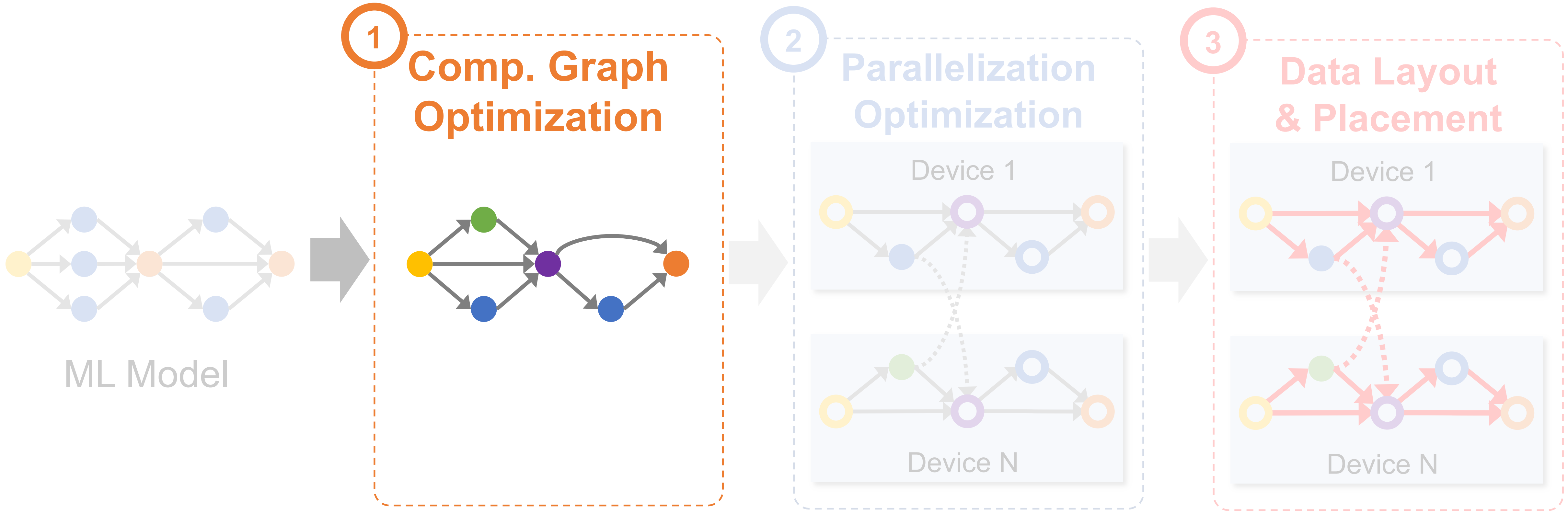
Advantages of Automated Approaches

- **Better runtime performance:** discovering novel optimizations hard to manually designed, 3-10x speedup over manual optimizations

- **Less engineering effort**: code for discovering optimizations is generally much less than manual implementation of these optimizations

- **Stronger correctness guarantees**: using formal verification techniques

Our Research: Automated Discovery of ML Optimizations

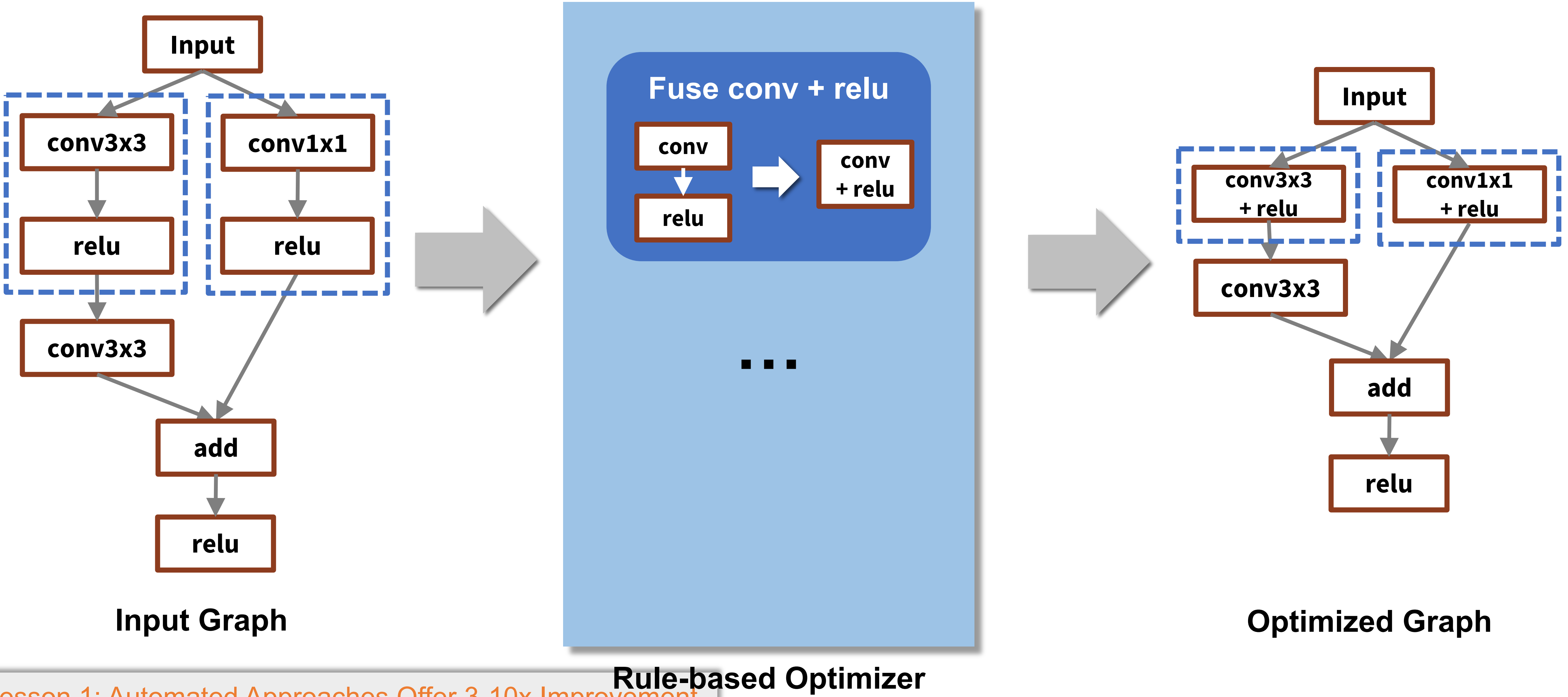


TASO [SOSP'19]
PET [OSDI'21]
Automated generation of graph optimizations

FlexFlow [MLSys'19]
Unity [OSDI'22]
Automated discovery of fast parallelization strategies

Lux [VLDB'18]
Roc [MLSys'20]
Automated data layout and placement optimizations

Current Rule-based Graph Optimizations



Current Rule-based Graph Optimizations

TensorFlow currently includes ~200 rules (~53,000 LOC)

Fuse conv + relu

Fuse conv + batch normalization

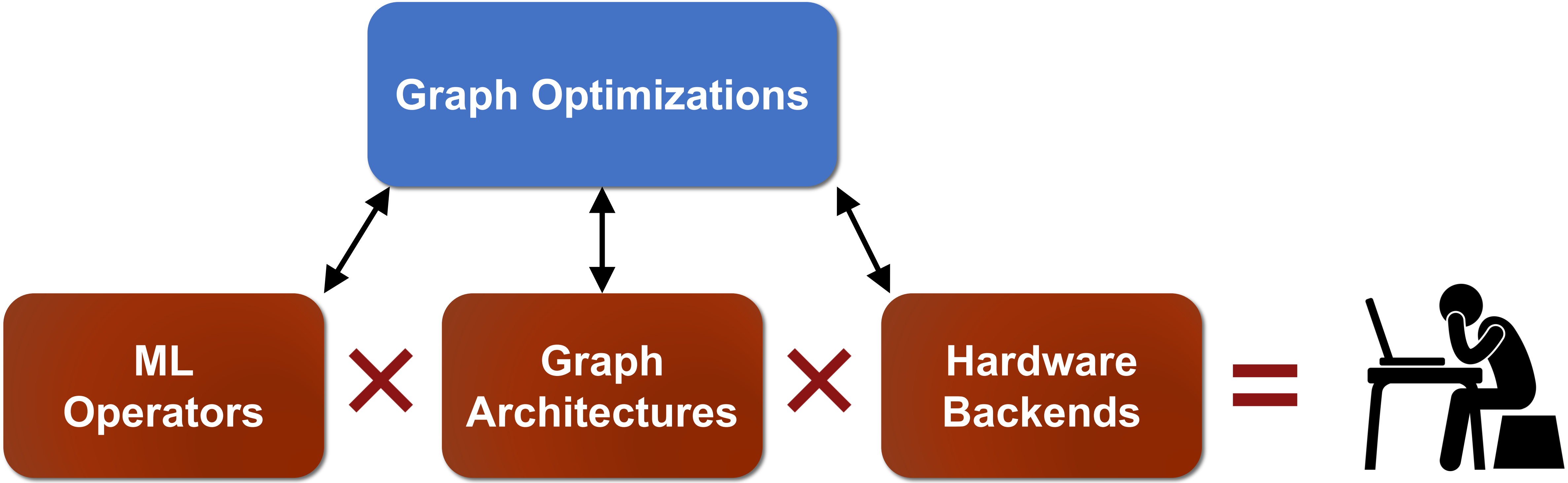
Fuse multi. convs

...

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
```

Infeasible to Manually Design Graph Optimizations

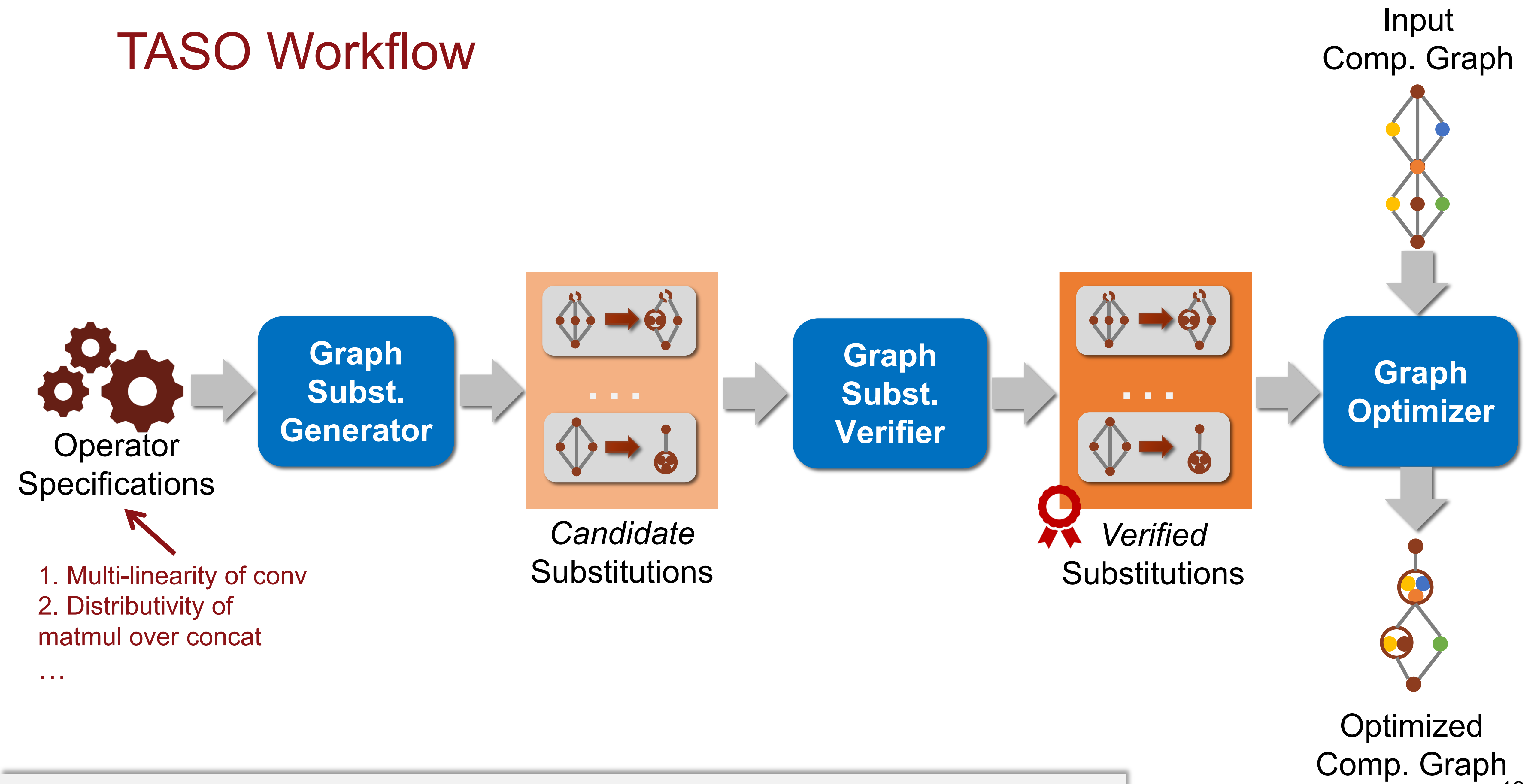


TASO: Tensor Algebra SuperOptimizer

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

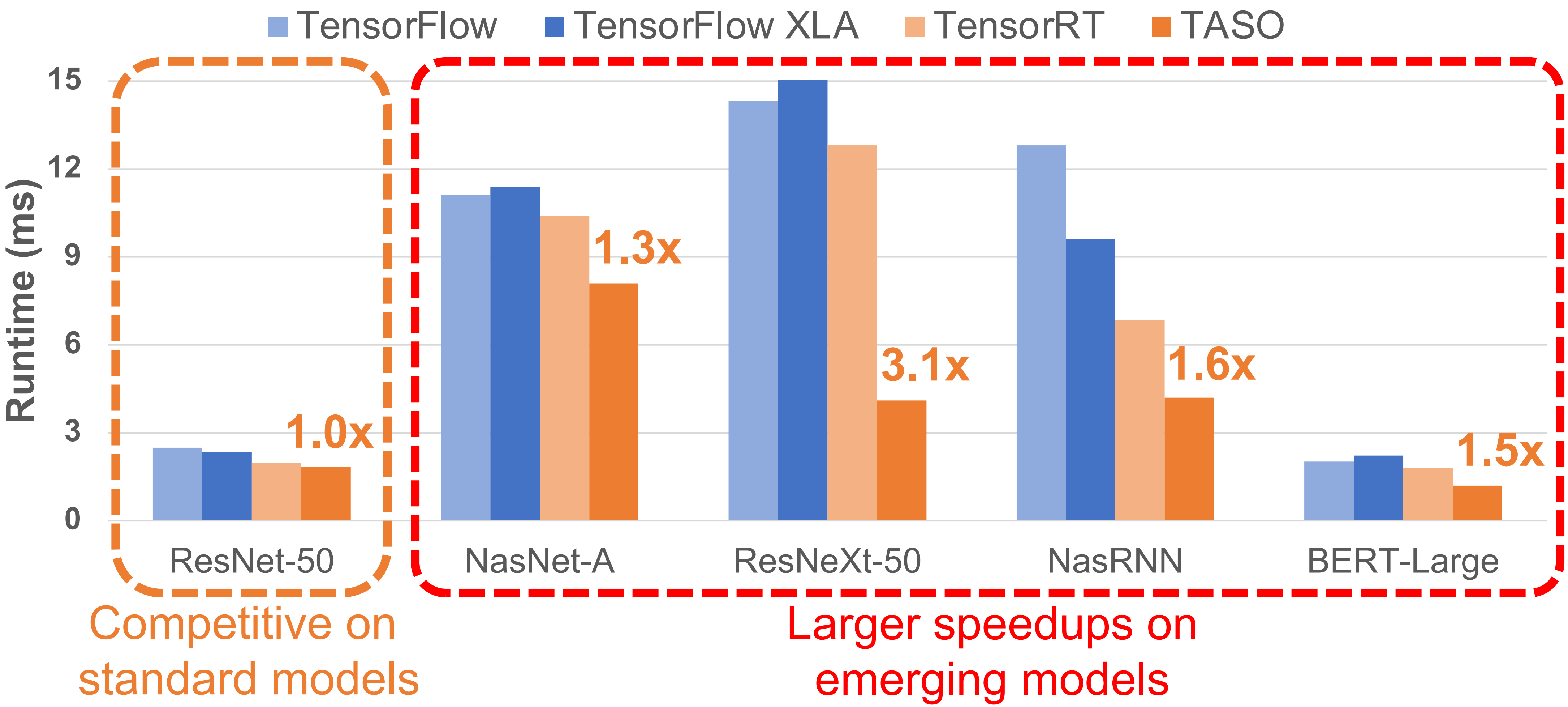
- **Less engineering effort:** **53,000** LOC for manual graph optimizations in TensorFlow → **1,400** LOC in TASO
- **Better performance:** outperform existing optimizers by up to **3x**
- **Stronger correctness:** formally verify all generated substitutions

TASO Workflow



TASO: Optimizing Deep Learning Computation with Automated Generation of Graph Substitutions. SOSP’19.

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



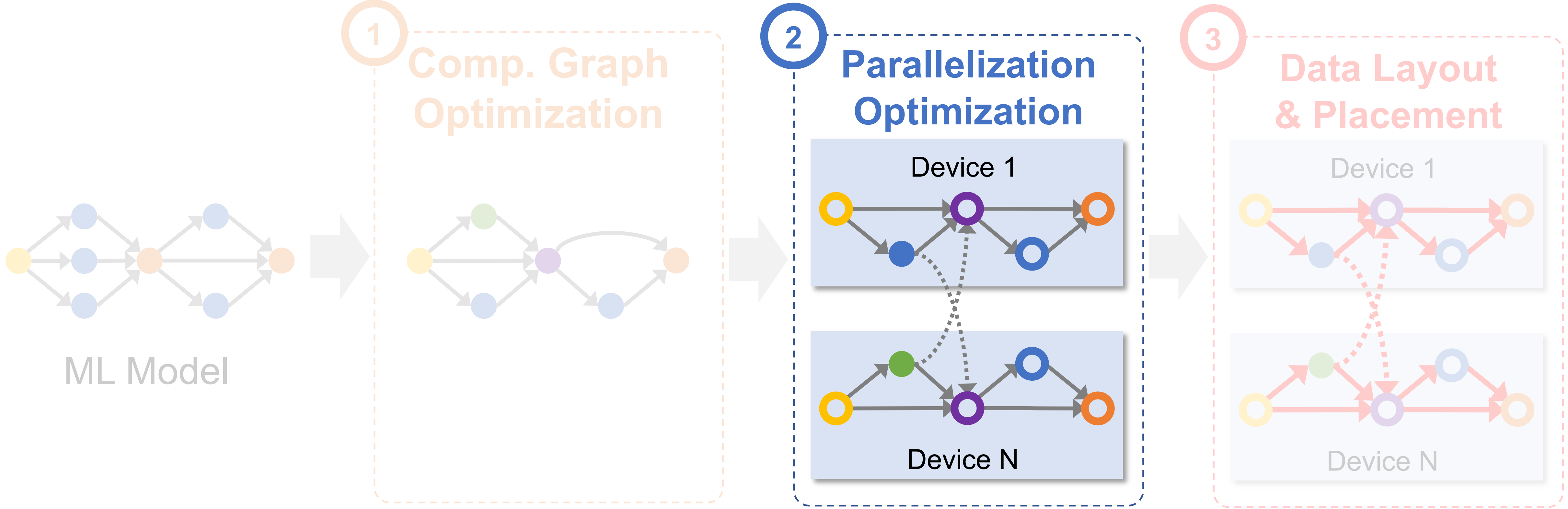
TASO

First DNN graph optimizer that automatically generates substitutions

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

1. PET: Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections. OSDI'21.
2. TASO: Optimizing Deep Learning Computation with Automated Generation of Graph Substitutions. SOSP'19.
3. Optimizing DNN Computation with Relaxed Graph Substitutions. MLSys'19.
4. Exploring Hidden Dimensions in Parallelizing Convolutional Neural Networks. ICML'18.

Our Research: Automated Discovery of ML Optimizations



TASO [SOSP'19]
PET [OSDI'21]
Automated generation of graph optimizations

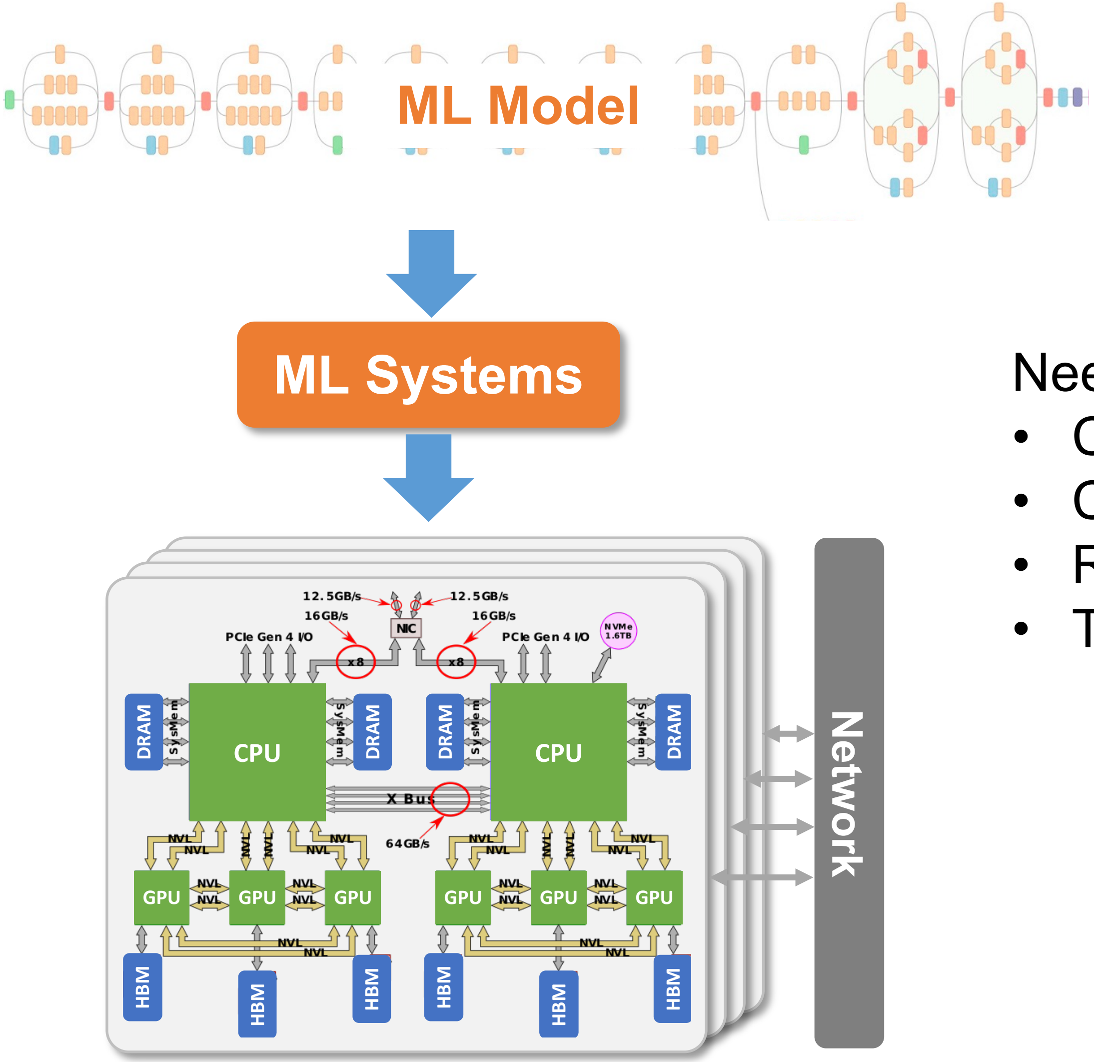
FlexFlow [MLSys'19]
Unity [OSDI'22]
Automated discovery of fast parallelization strategies

Lux [VLDB'18]
Roc [MLSys'20]
Automated data layout and placement optimizations

catalyst

Lesson 1: Automated Approaches Offer 3-10x Improvement

Challenges of Parallelizing DNN Training

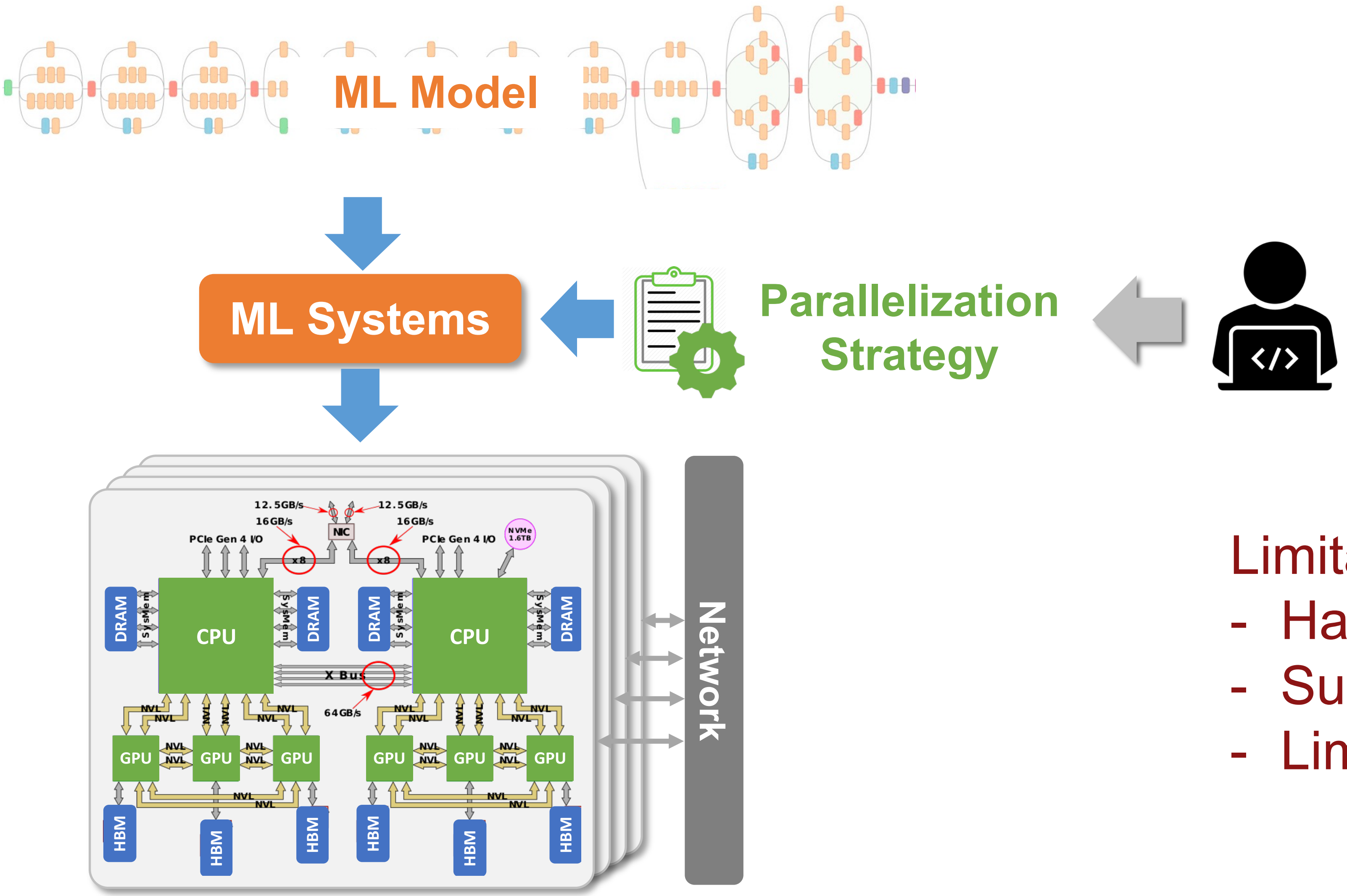


Need to simultaneously consider:

- Computation cost
- Communication overhead
- Resource usage
- Task scheduling

Lesson 1: Automated Approaches Offer 3-10x Improvement

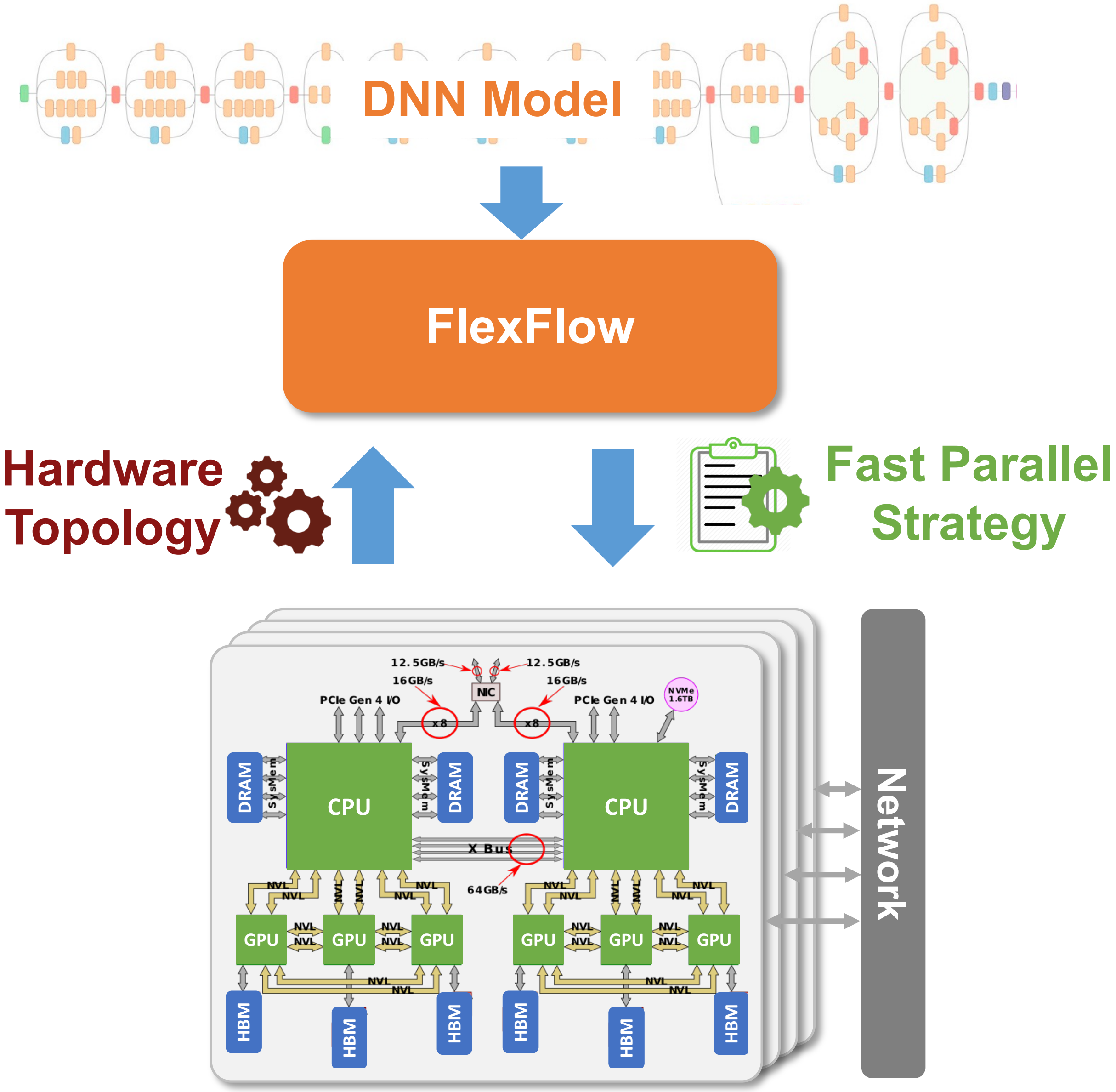
Current Systems Rely on Manually Designed Strategies



Limitations:
- Hard to manually design
- Suboptimal performance
- Limited portability

Lesson 1: Automated Approaches Offer 3-10x Improvement

FlexFlow: Automatically Optimizing DNN Parallelization

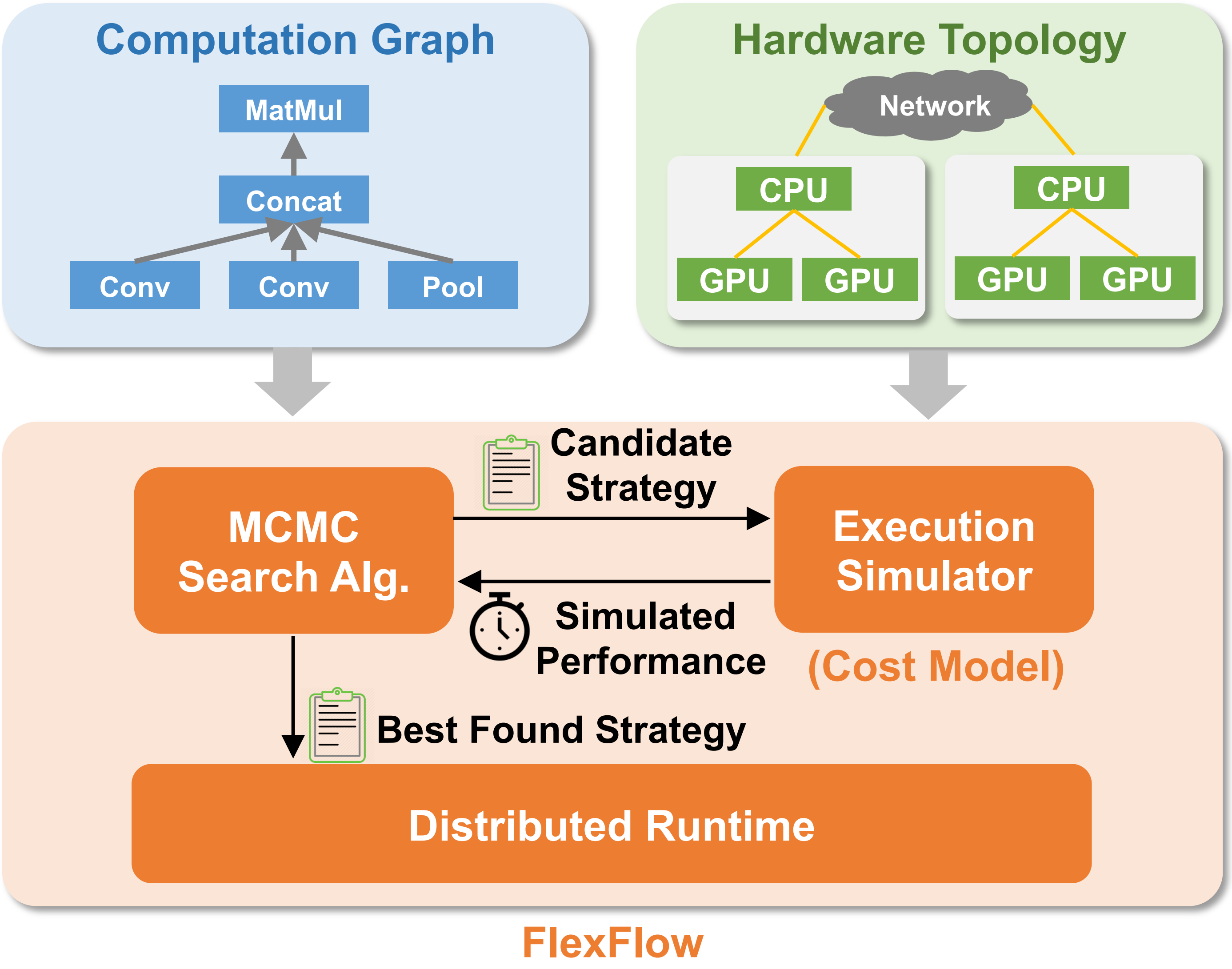


Better Performance
Up to 10x faster than manually designed strategies

Fast Deployment
Minutes of automated search to discover performant strategies

No Manual Effort
Automatically find strategies for new DNN models or hardware platforms

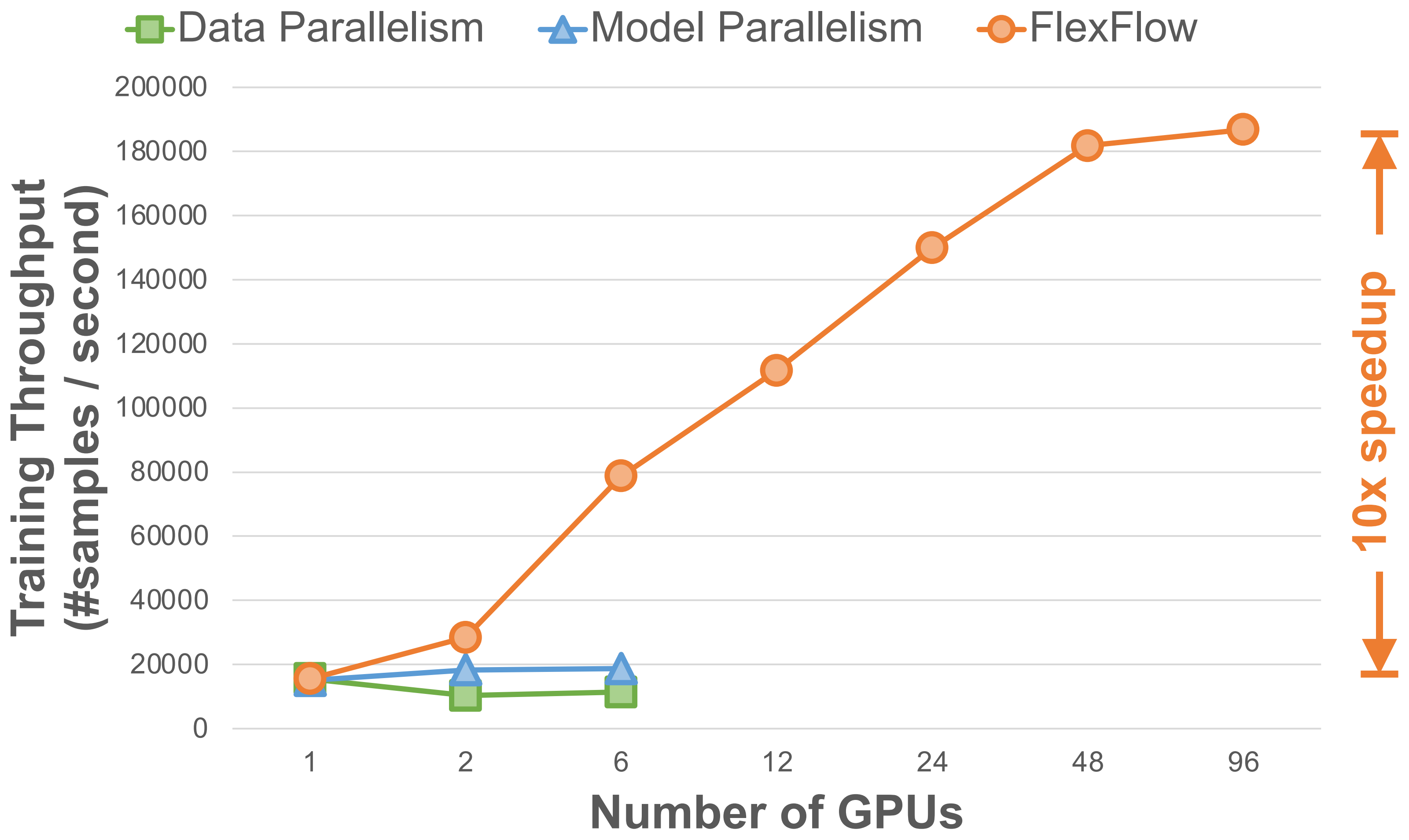
FlexFlow Overview



Deep Learning Recommendation Model (DLRM)

facebook

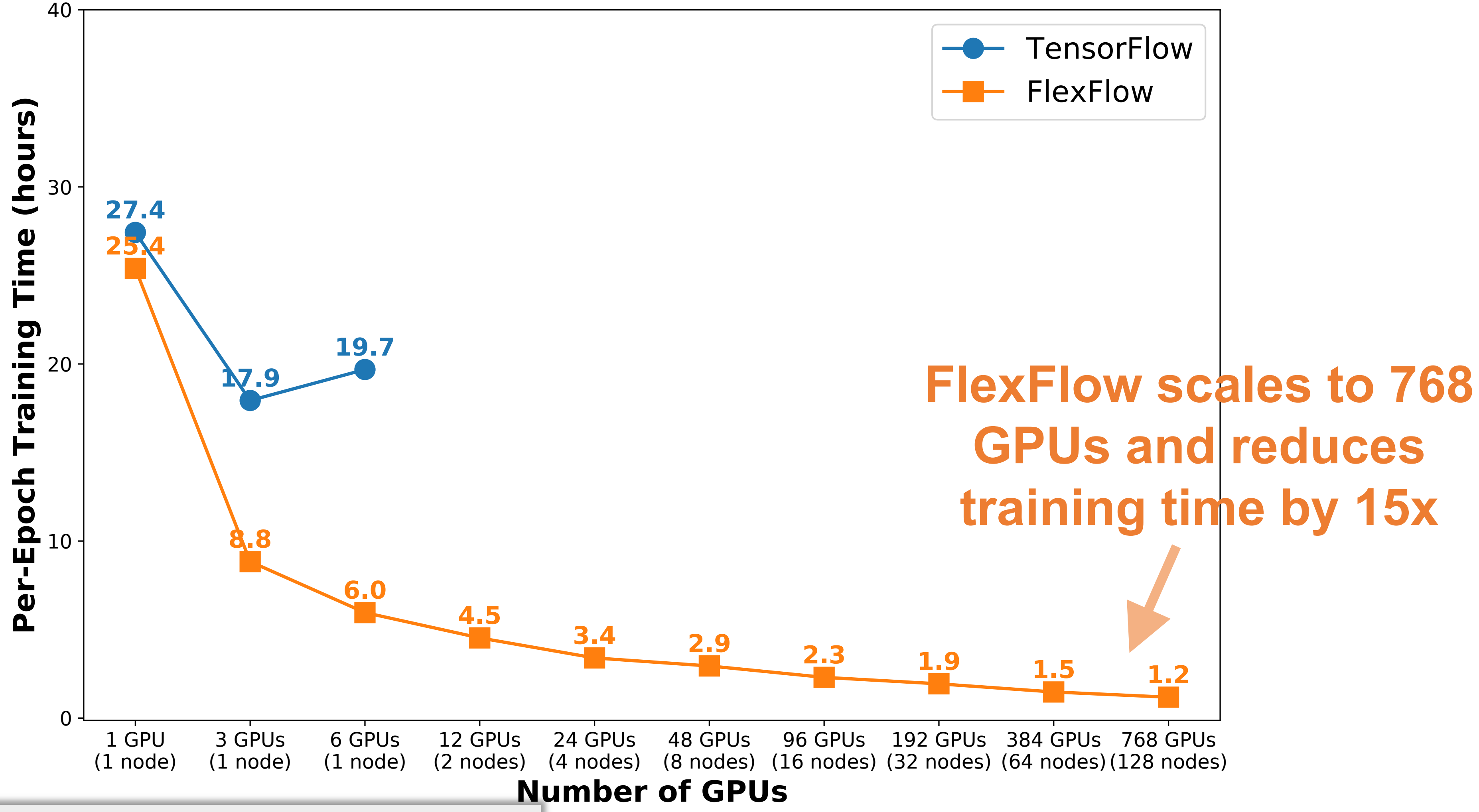
A deep learning model for ads recommendation



Lesson 1: Automated Approaches Offer 3-10x Improvement

ECP-CANDLE Training Performance

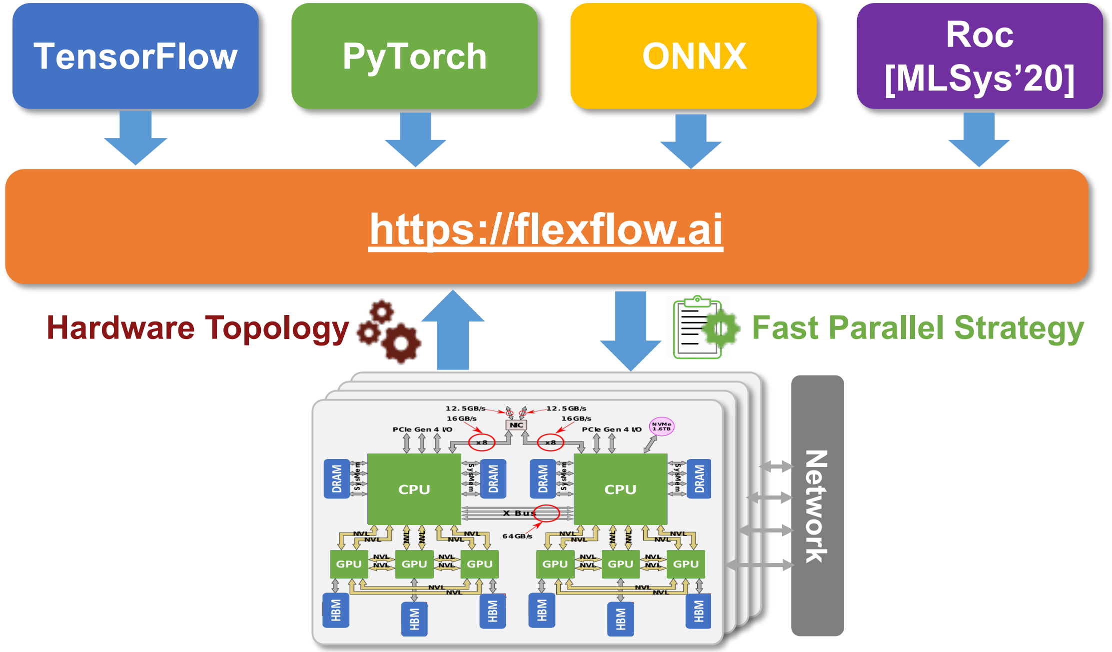
A deep learning model for precision medicine

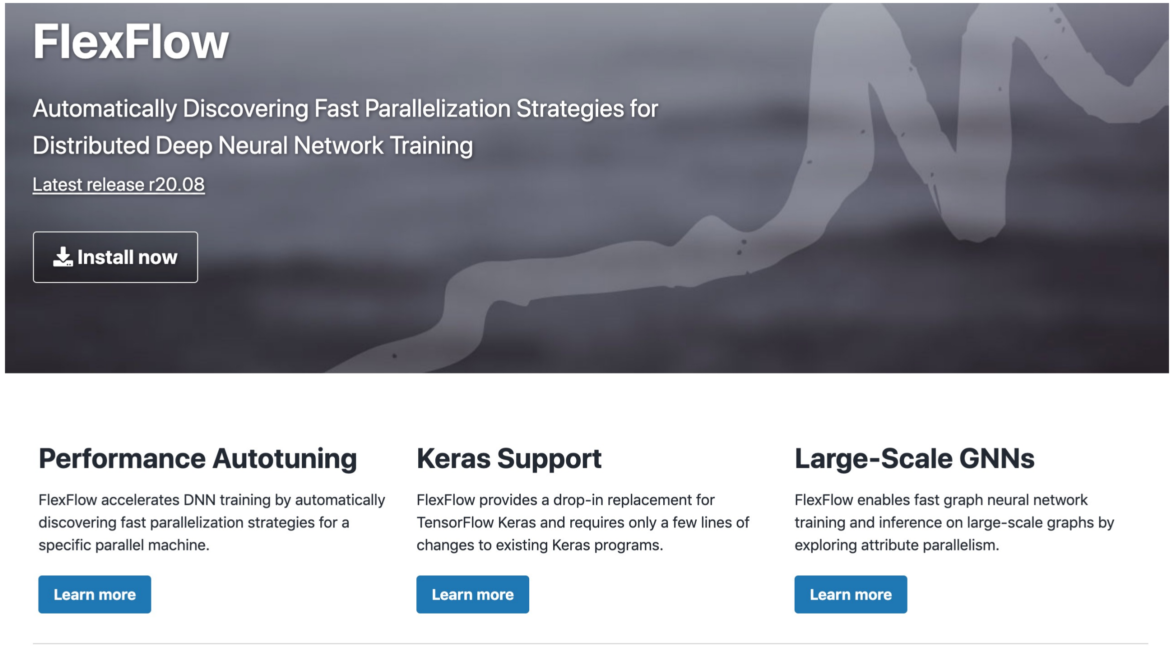


Lesson 1: Automated Approaches Offer 3-10x Improvement

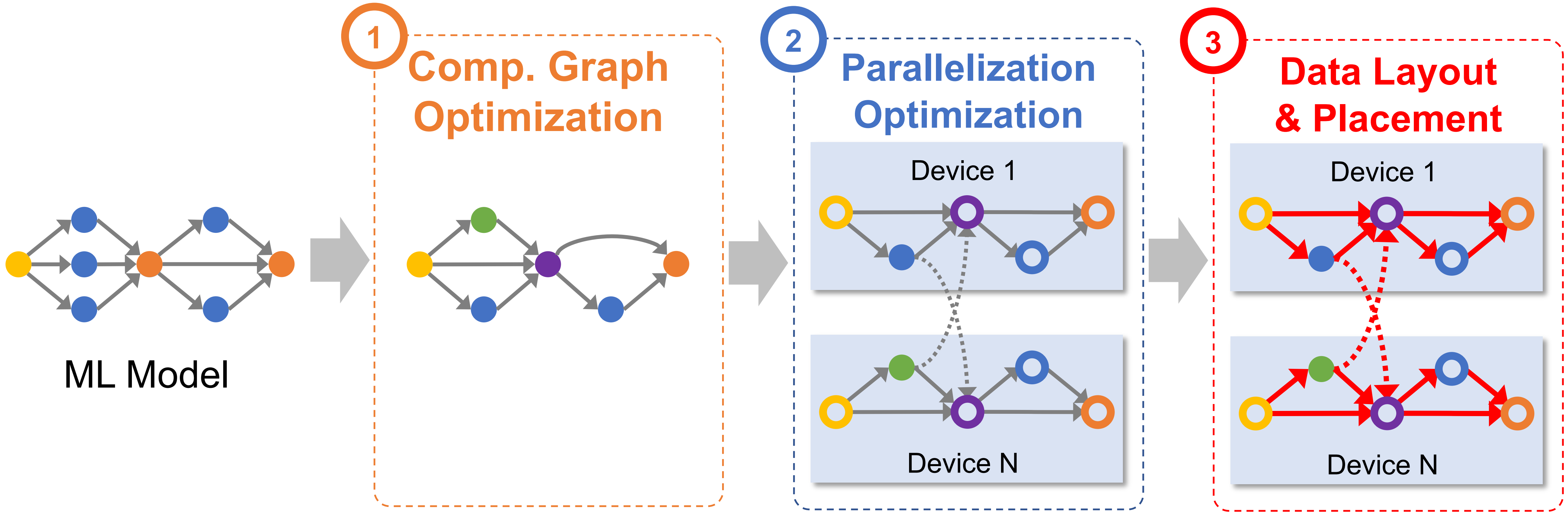
FlexFlow: Automatically Discovering Fast and Scalable DNN Parallelization Strategies

https://flexflow.ai





Lesson 1: Automated Approaches Offer 3-10x Improvement

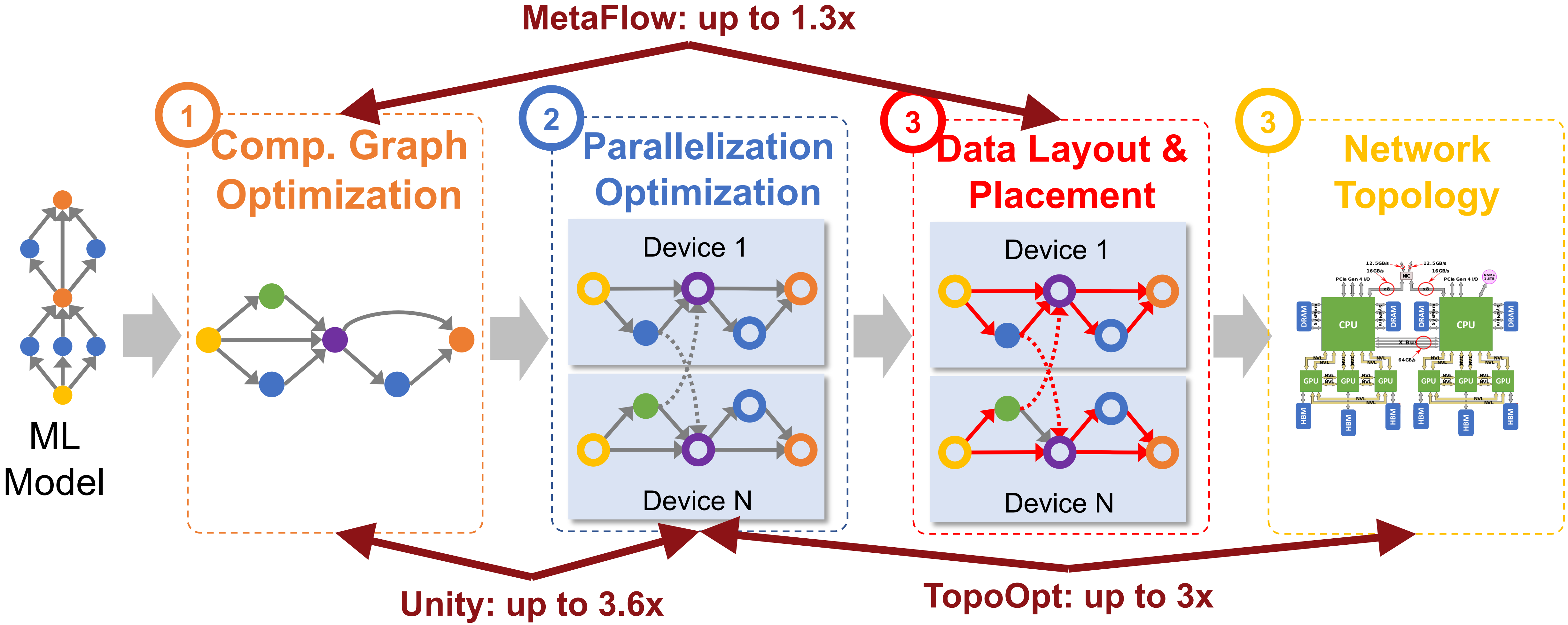


TASO: Up to 3.1x
PET: Up to 2.5x

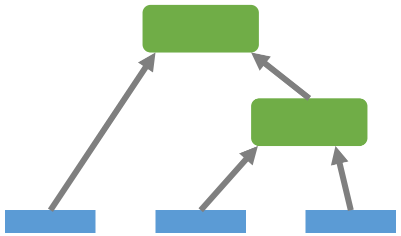
FlexFlow: Up to 10x
Unity: Up to 3.6x

Lux: Up to 10x
Roc: Up to 4.1x

Lesson 2: Joint Optimization is Critical to Performance



1. Unity: Accelerating DNN Training Through Joint Optimization of Algebraic Transformations and Parallelization. OSDI'22.
2. TopoOpt: Optimizing the Network Topology for Distributed DNN Training. NSDI'23.
3. MetaFlow: Optimizing DNN Computation with Relaxed Graph Substitutions. MLSys'19

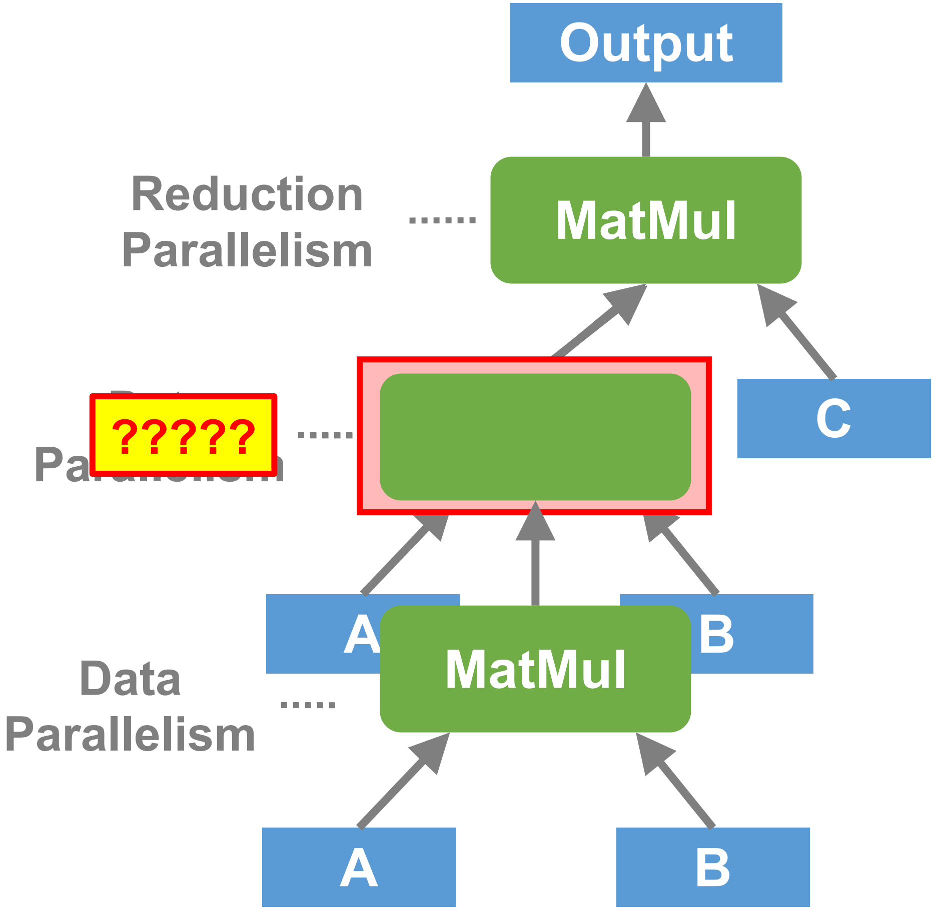


Auto-Parallelization

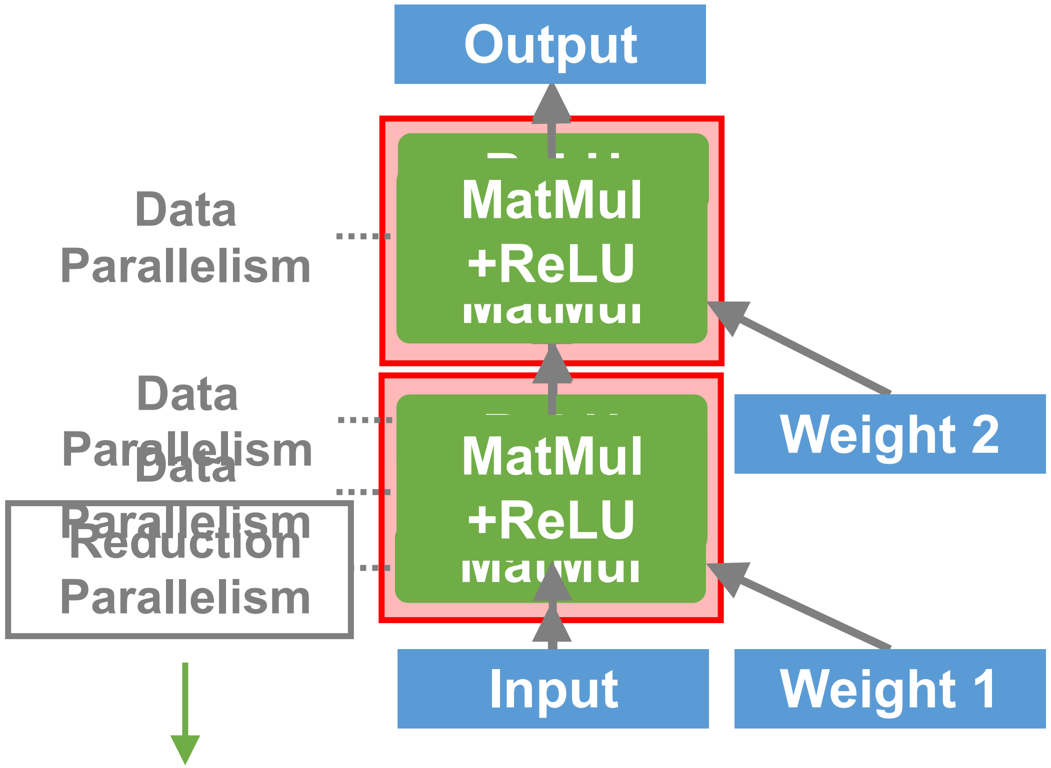
Graph Optimization

?

Lesson 2: Joint Optimization is Critical to Performance



Lesson 2: Joint Optimization is Critical to Performance



≈ 6× less communication!

Lesson 2: Joint Optimization is Critical to Performance

1. Representation
2. Scalability

Unity

Representation

Representation

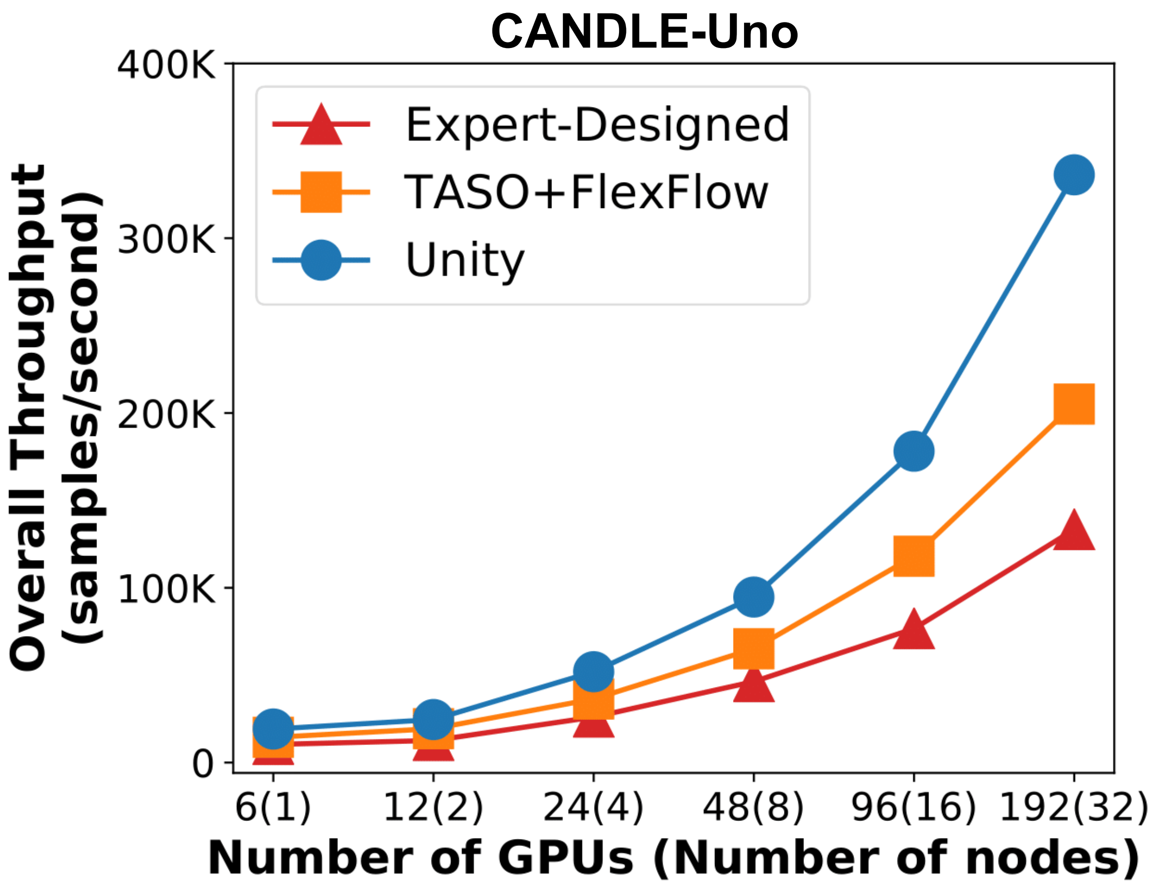
Parallel Computation Graph (PCG)

Scalability

Scalability

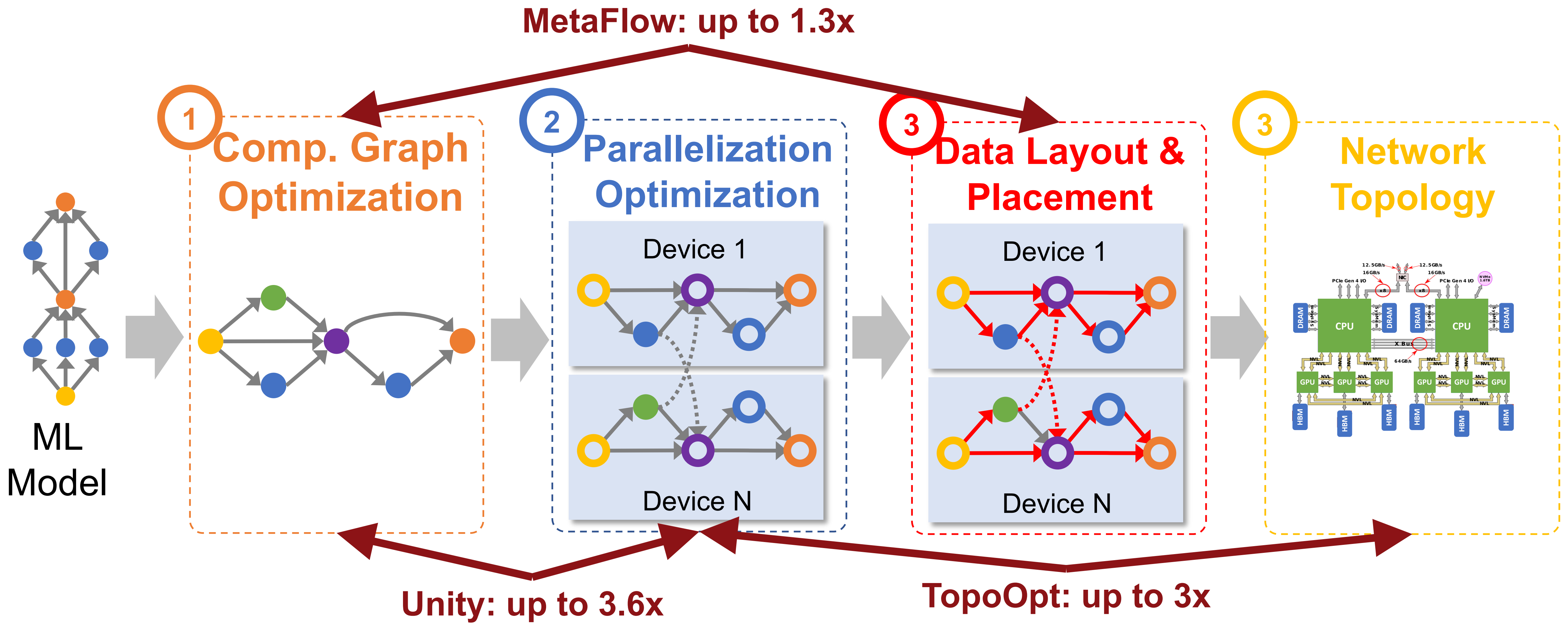
Hierarchical Search Algorithm

Joint Optimization Enables Better Performance and Scalability



Lesson 2: Joint Optimization is Critical to Performance

Lesson 2: Joint Optimization is Critical to Performance



1. Unity: Accelerating DNN Training Through Joint Optimization of Algebraic Transformations and Parallelization. OSDI’22.
2. TopoOpt: Optimizing the Network Topology for Distributed DNN Training. NSDI’23.
3. MetaFlow: Optimizing DNN Computation with Relaxed Graph Substitutions. MLSys’19

Lesson 3: Combining ML and Systems Optimizations is Promising but Challenging

Systems Optimizations

- Graph Transformations
- Auto Parallelization
- Kernel Generation
- Data Layout and Placement

ML Optimizations

- Quantization
- Pruning
- Distillation
- Neural Architecture Search

Lesson 3: Combining ML and Systems Optimizations is Promising but Challenging

Systems Optimizations

Pro: preserve functionality

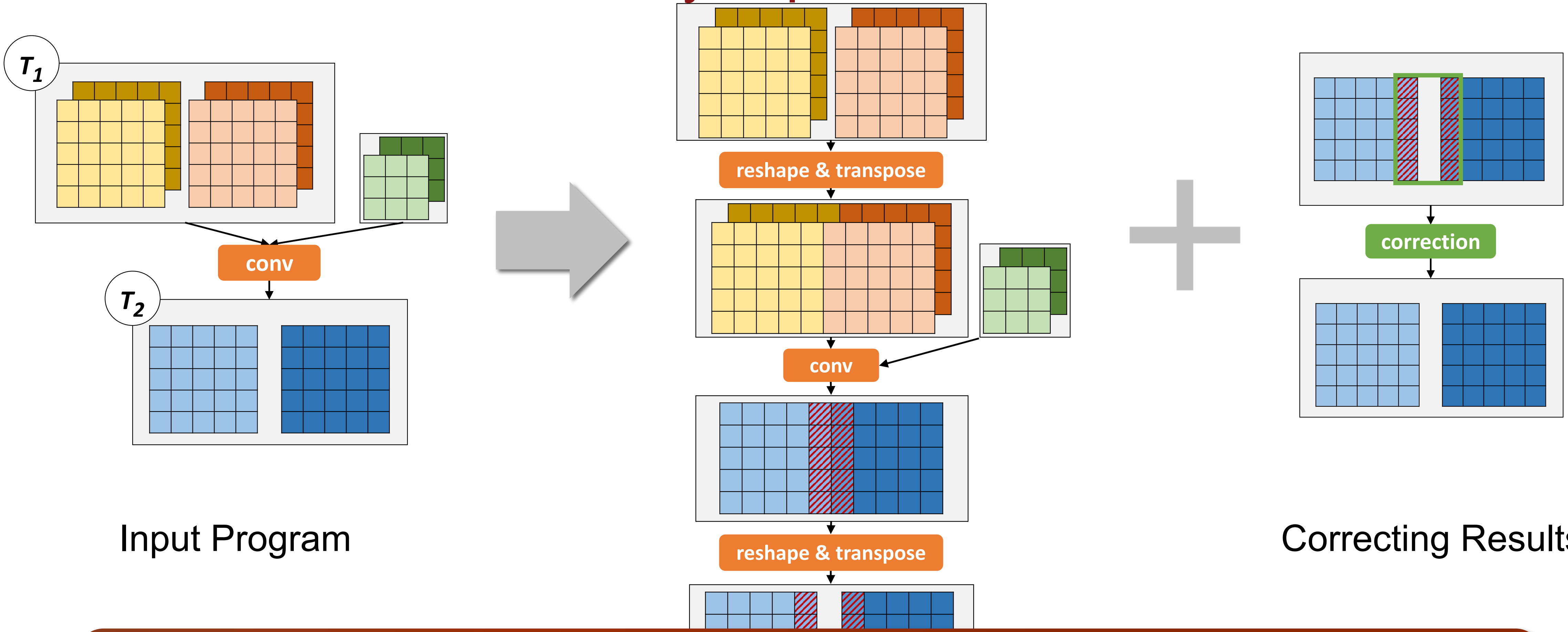
ML Optimizations

Pro: better performance

- Faster ML operators
- Less Computation

Achieve the best of both worlds?

Hidden Treasure: Partially Equivalent Transformations



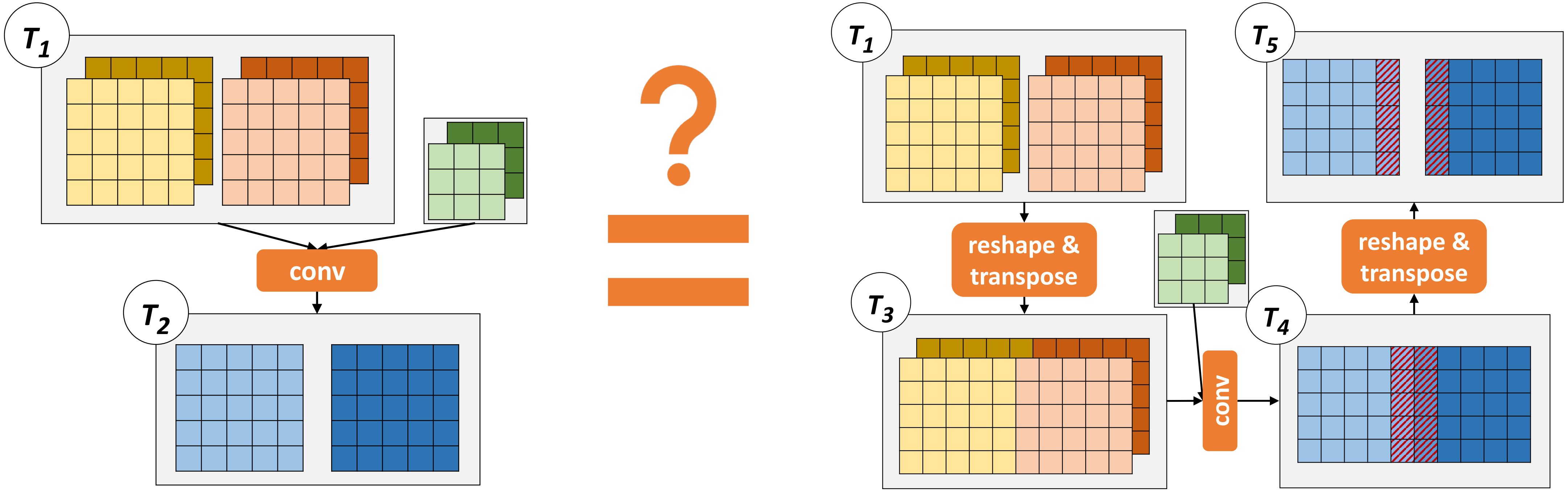
Input Program

Correcting Results

- Transformation and correction lead to **1.2x** speedup for ResNet-18
- Correction preserves end-to-end equivalence

PET: Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections. OSDI'21

Hidden Treasure: Partially Equivalent Transformations



1. Which part of the computation is not equivalent?
2. How to correct the results?

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

PET

- **Tensor program optimizer** with partially equivalent transformations and automated corrections

- **Larger optimization space** by combining fully and partially equivalent transformations
- **Better performance**: outperform existing optimizers by up to **2.5x**
- **Correctness**: automated corrections to preserve end-to-end equivalence

Equivalent Optimizations

Partially-Equivalent Optimizations

Non-Equivalent Optimizations

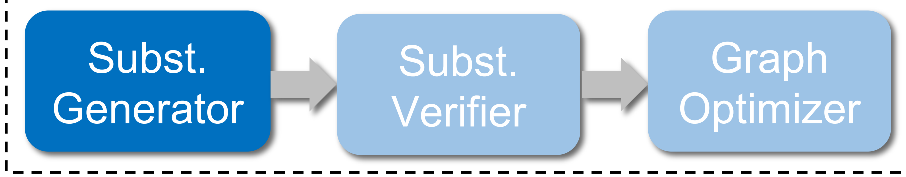
- Runtime Performance
- Predictive Performance

Three Lessons

1. Automated approaches can offer **3-10x** improvement on most tasks

2. Joint optimization is **critical**

3. Combing systems and ML optimizations is **promising but challenging**

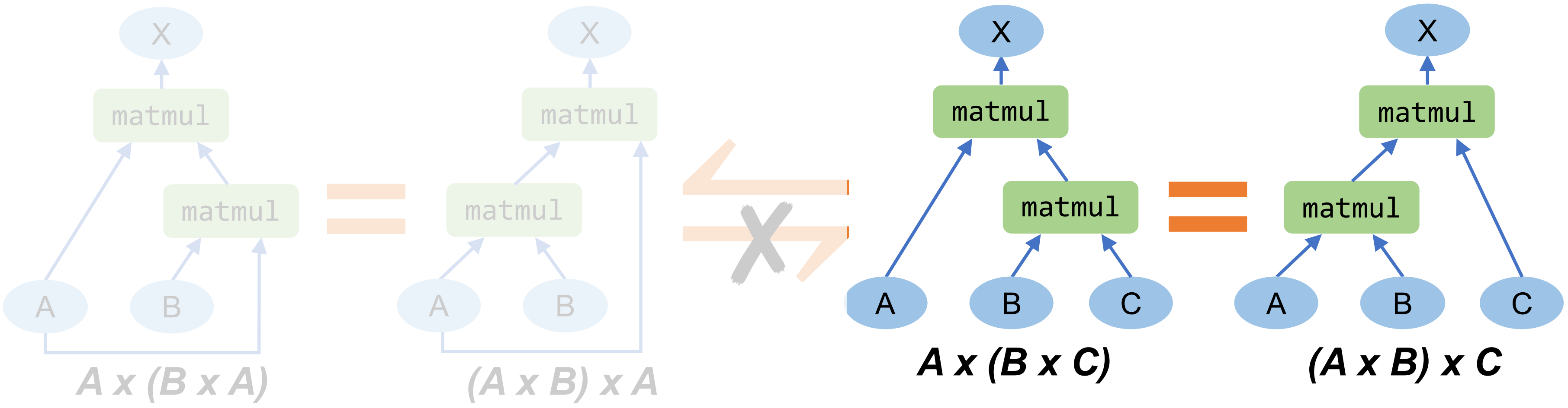


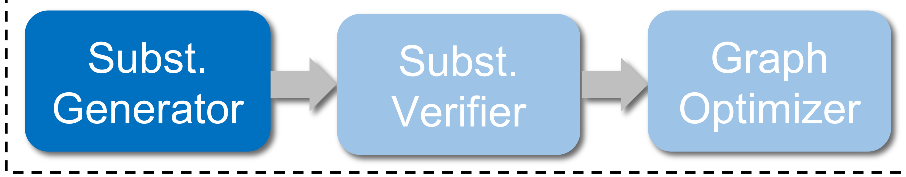
Pruning Redundant Substitutions

28,744 substitutions

Input Tensor Renaming

17,346 substitutions





Pruning Redundant Substitutions

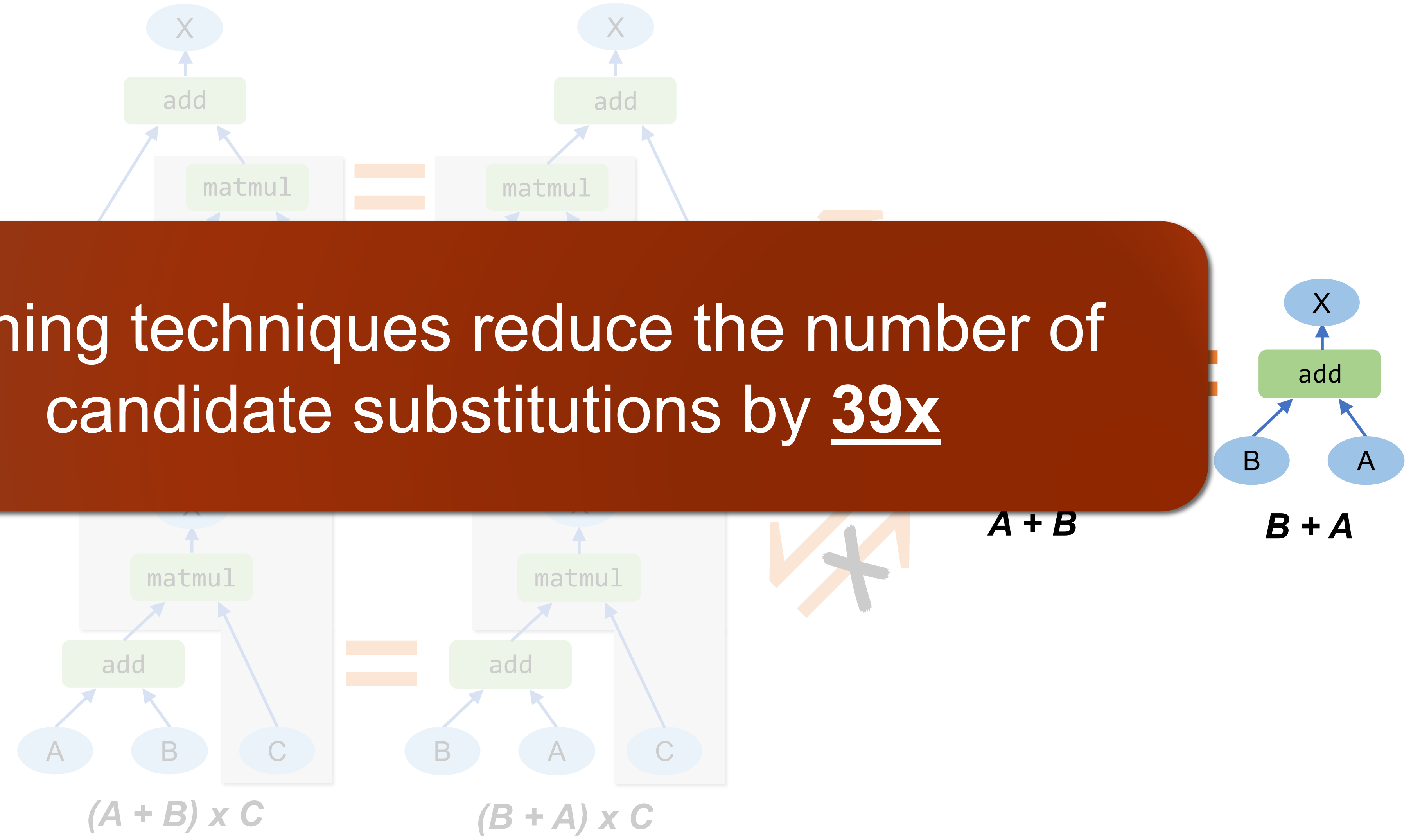
28,744 substitutions

Input Tensor Renamin

17,346 subst

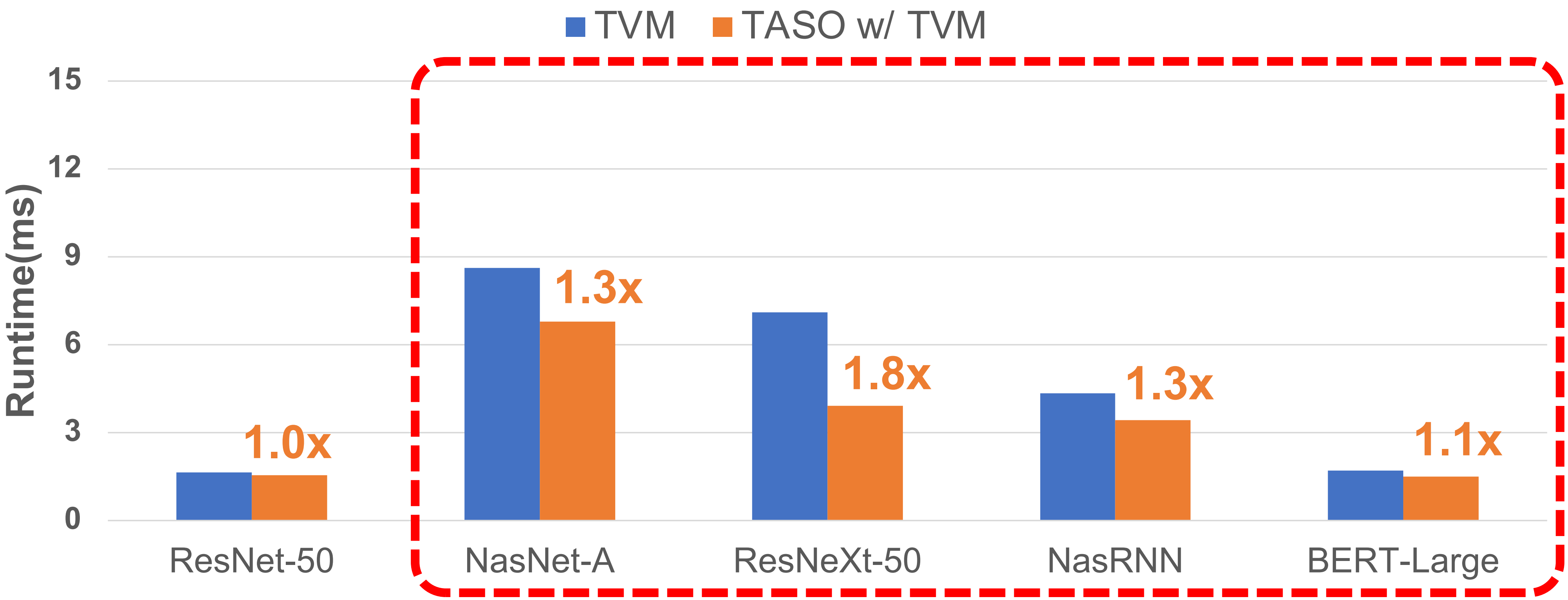
Common Subgraphs

743 substitutions



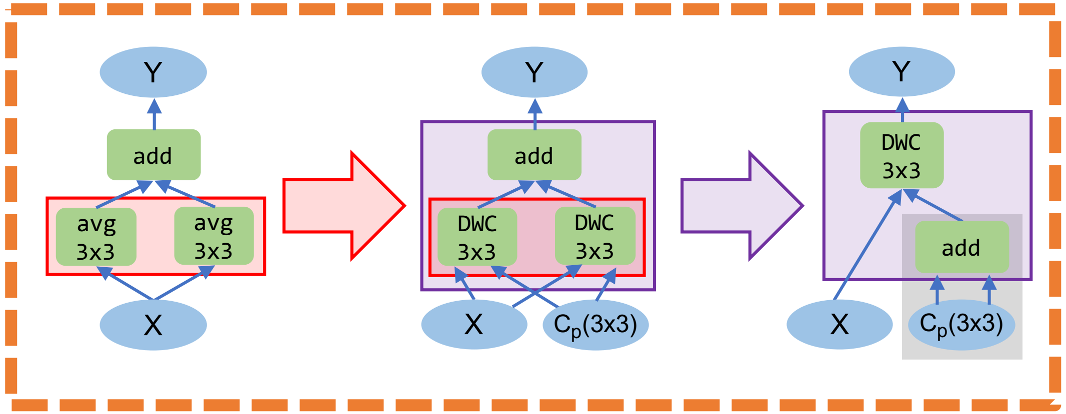
Pruning techniques reduce the number of candidate substitutions by **39x**

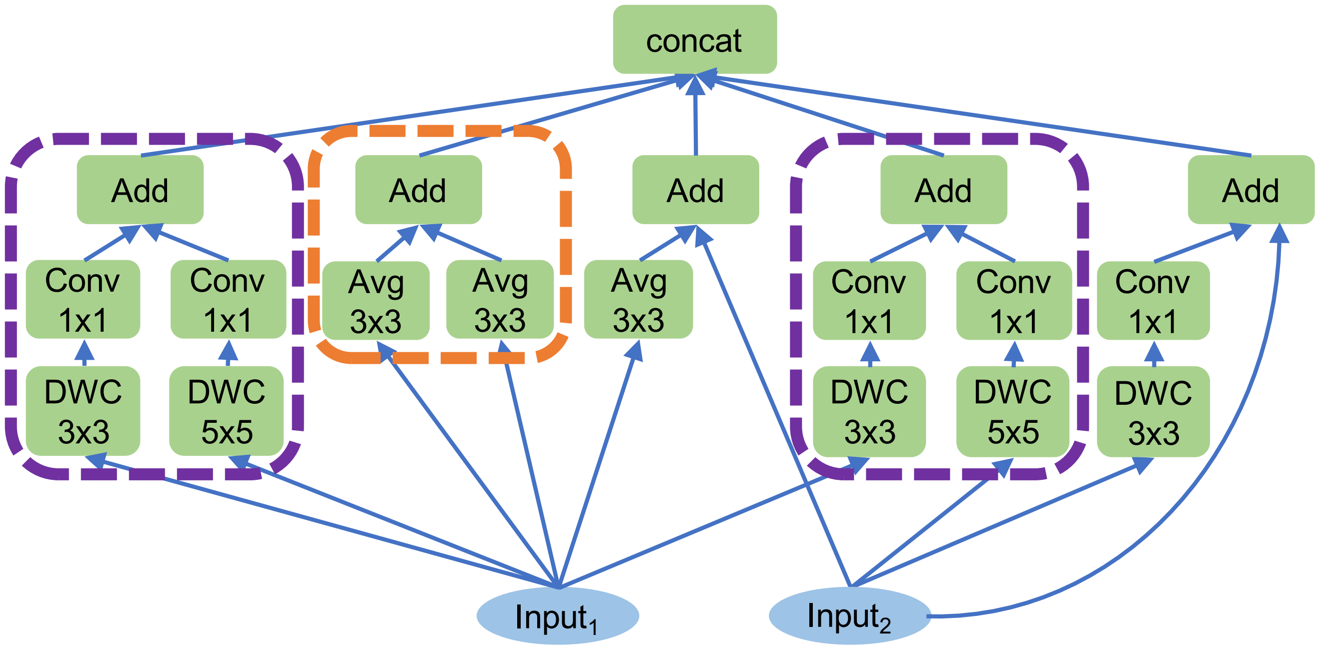
End-to-end Inference Performance (TVM)



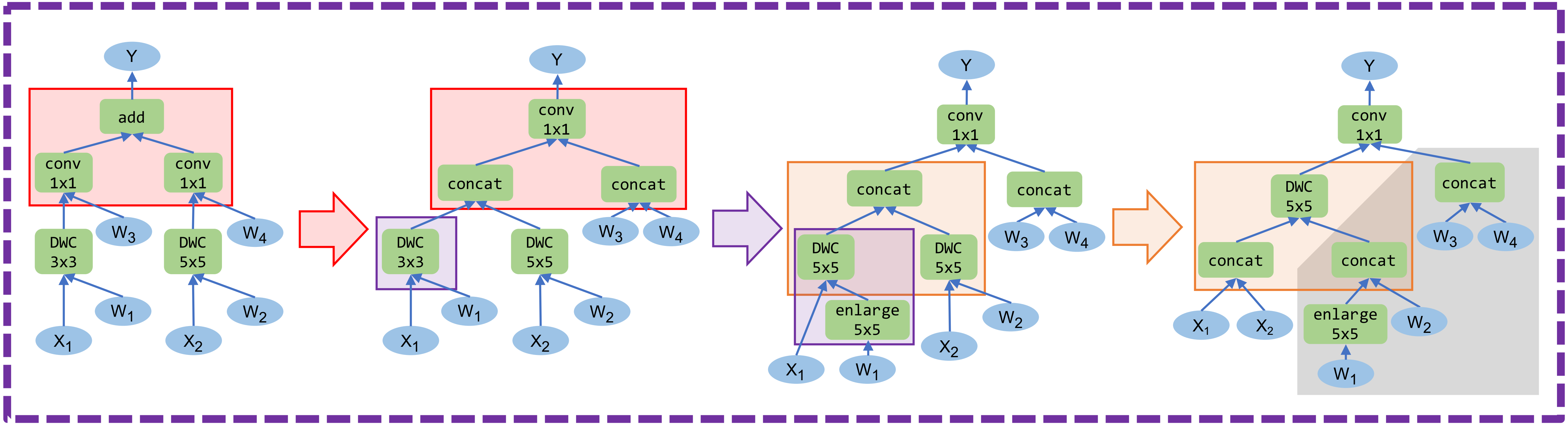
Larger speedups on emerging models

Case Study: NASNet





*DWC: depth-wise convolution



Heatmap of Used Substitutions

Not covered in TensorFlow

How many times a subst. is used to optimize a DNN

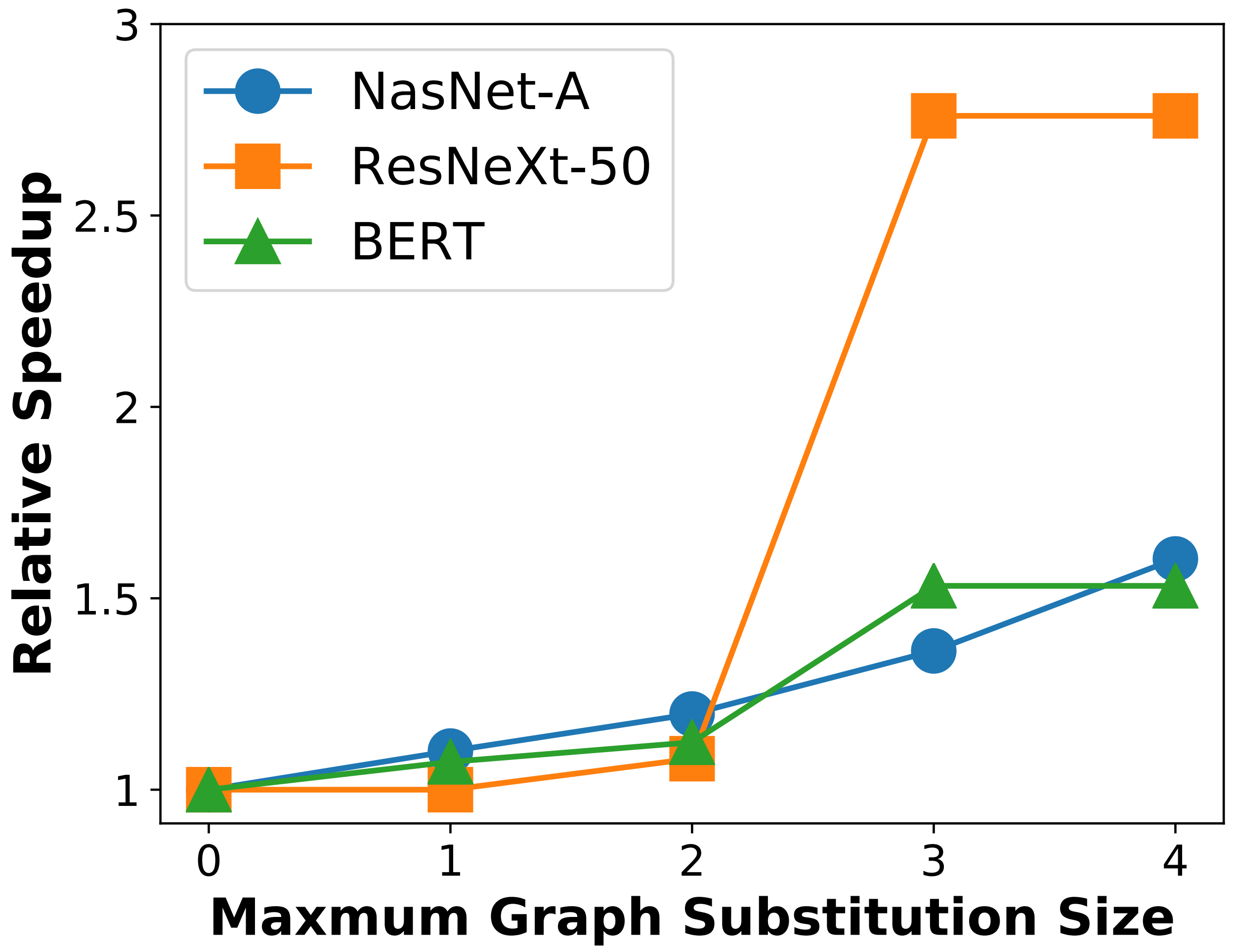
	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21	s22	s23	s24	s25	s26	s27
ResNet50	29	6	29	6	1	5	5																				
NasNet-A	3					41	41	14	32	3	15	19	19	9	10												
ResNeXt-50	25	5	25					1								23	3	7	5	7							
NasRNN																					10	40	20	20	10		
BERT																							24	24		12	36

Colour scale: 0, 10, 20, 30, 40

Different DNN models require different substitutions.

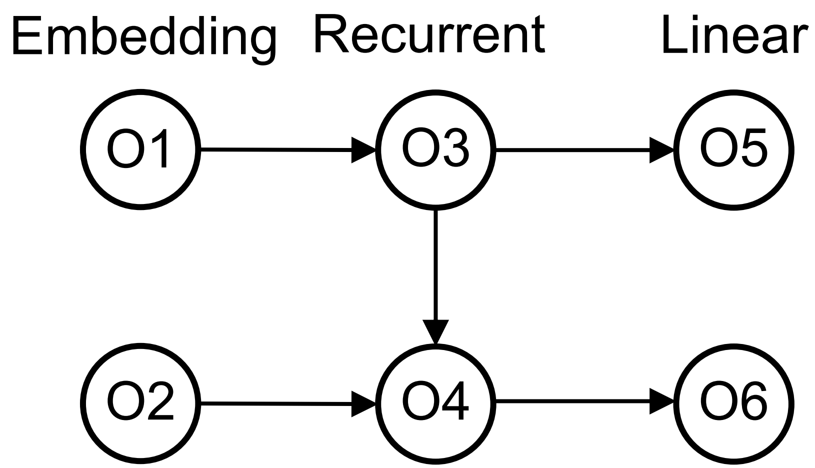
Scalability Analysis

Max Num. of Operators	Mem. to Cache Fingerprints
1	**0.9 KB**
2	**35.8 KB**
3	**6.9 MB**
4	**5.35 GB**



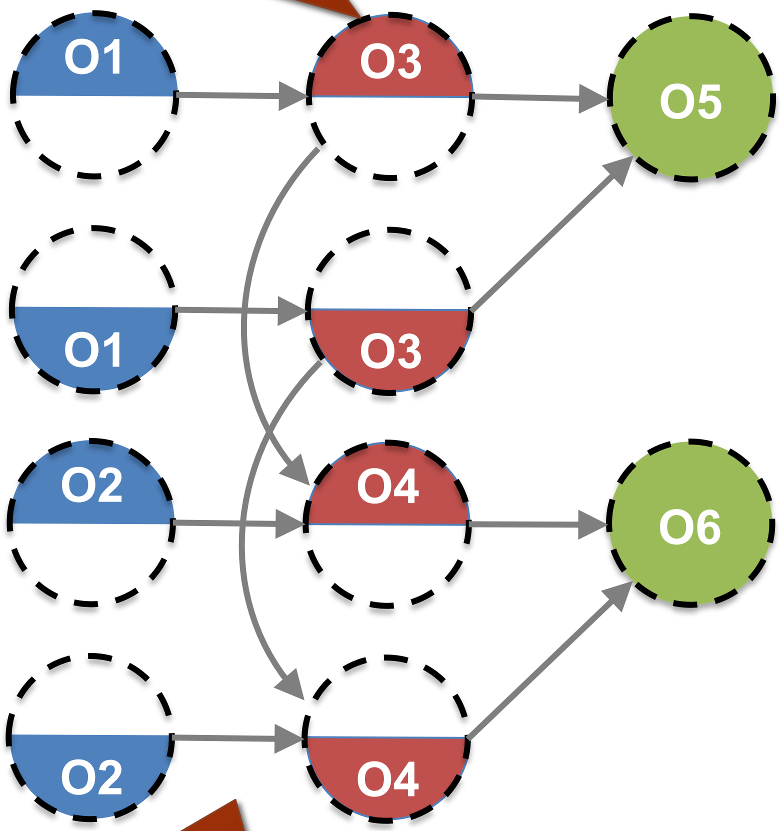
Execution Simulator

Task run time ≈ measurements on operators



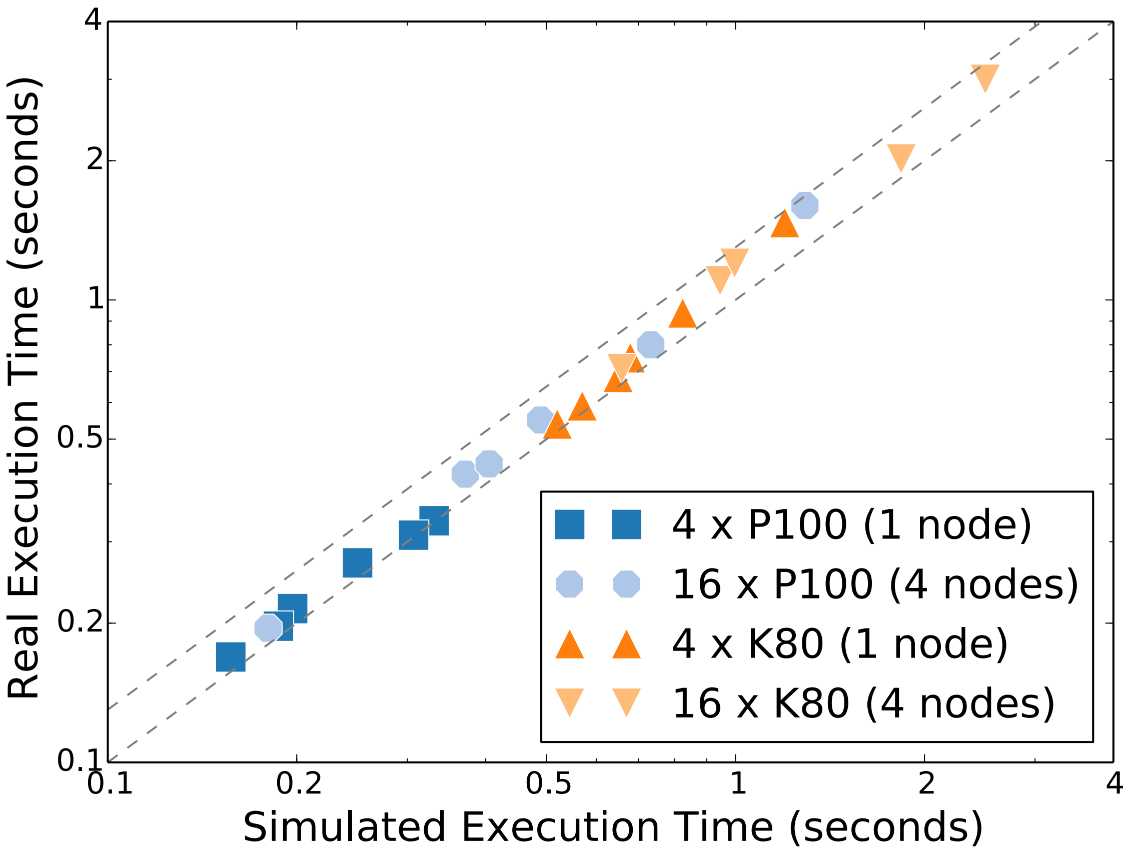
ML Architecture

Parallelization Strategy	
O1, O2	Degree(sample) = 2 **GPU1**
O3, O4	Degree(sample) = 2 **GPU2**
O5, O6	Degree(sample) = 1 **GPU3**



Data transfer time ≈ tensor size / channel bandwidth

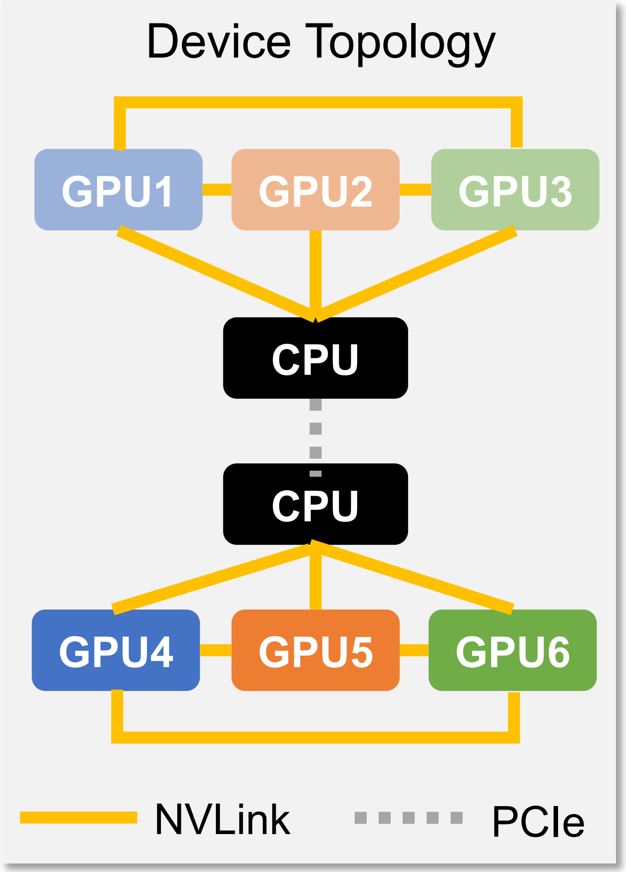
Execution Simulator

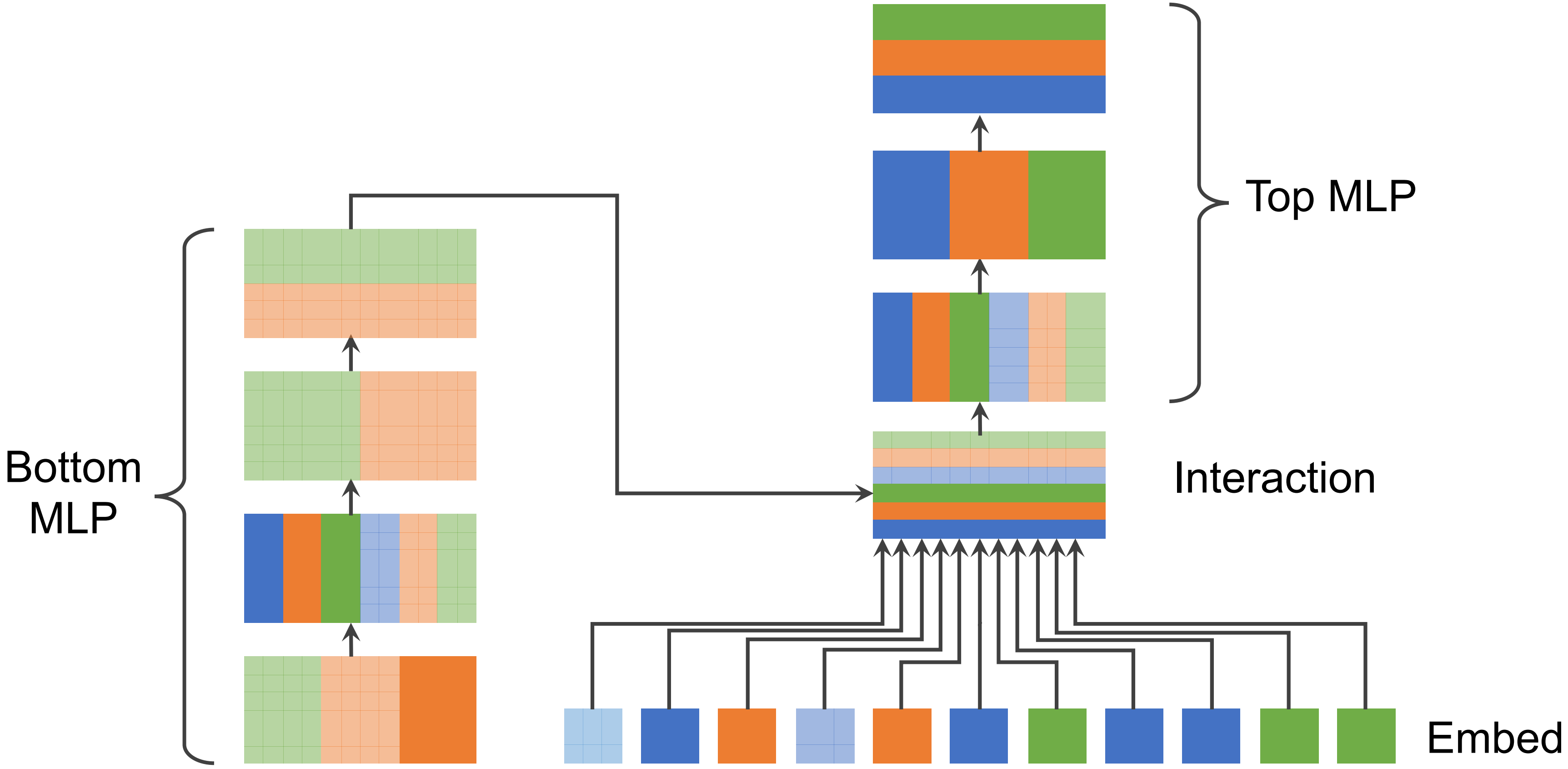


Relative difference between simulated and actual execution time is less than 30%

Simulated execution time preserves real execution time ordering

Case Study





Layers	Communication	Compute	Strategy
Embed	Heavy	Light	Parallelism across **Operators**
Interaction	Light	Heavy	Parallelism across **Samples**
MLP	Heavy	Heavy	Parallelism across **Samples** and **Parameters**

FlexFlow Impact

Facebook uses FlexFlow to train production ML models. Increase training throughput by **10x**.

Used by LANL to train ML models for precision medicine. Reduce training time from **days** to **hours**.

Improve the **accuracy**, **scalability**, and **performance** of graph neural networks.