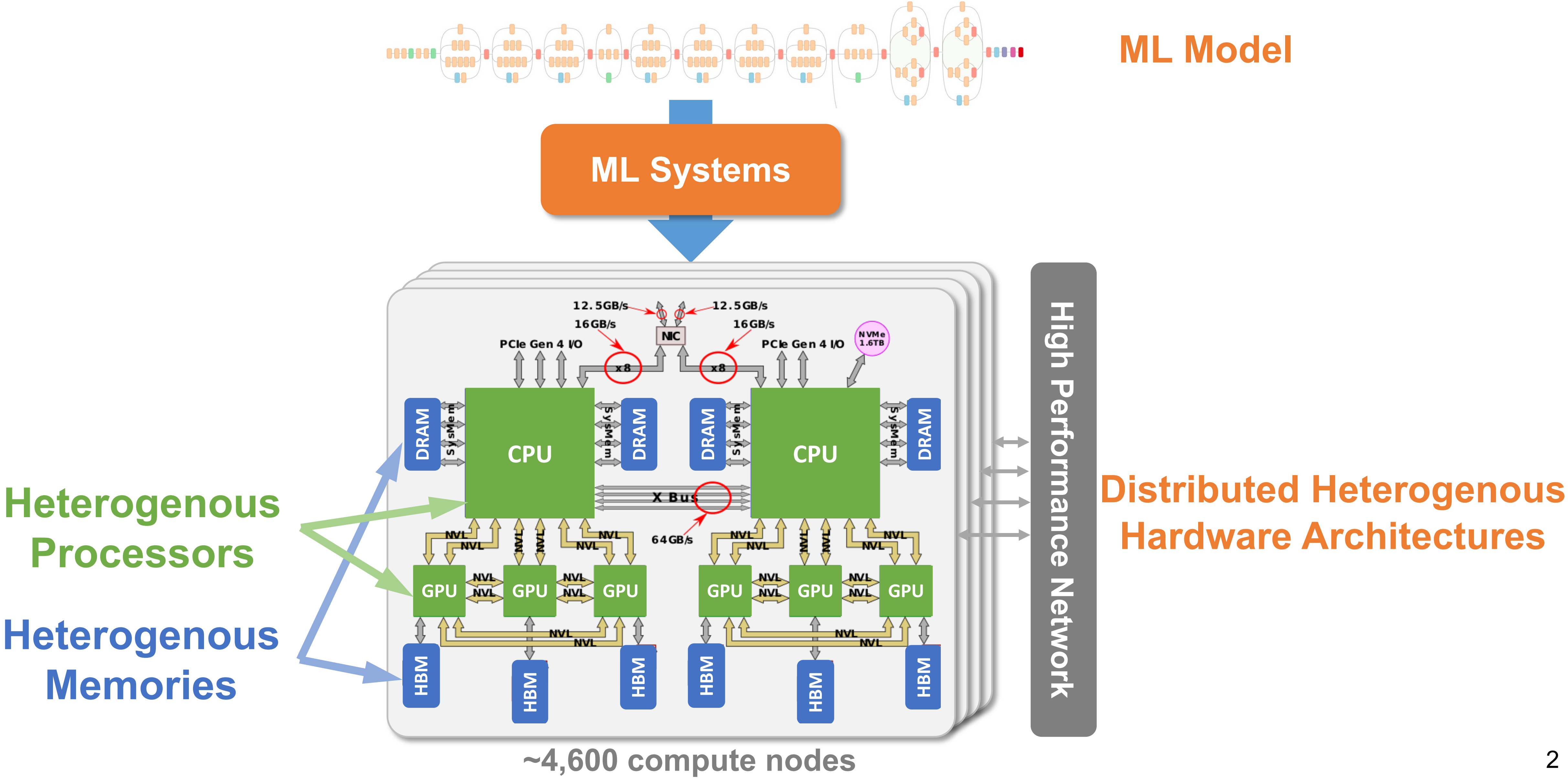
15-442/15-642: Machine Learning Systems

Three Lessons Learned from ML Systems

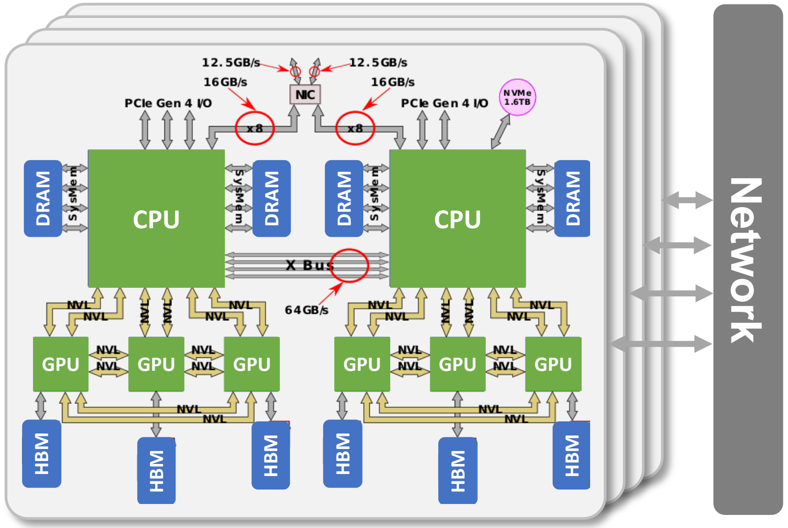
Tianqi Chen and Zhihao Jia
Carnegie Mellon University

ML Systems are a Key Ingredient in ML



Challenges of Building ML Systems

ML Systems



Increasingly diverse models

Large Language Models
Transformers,
Vision Language Models,
Graph Neural Networks,
Mixture of Experts,
Sparse NN,
Dynamic NN,
…

Increasingly heterogeneous hardware

CPUs, GPUs, TPUs,
AI accelerators,
FPGAs, CGRAs,
Programmable networks,
and their combinations
…

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

Faculty

Tianqi Chen
Assistant Professor

Greg Ganger
Professor

Phillip B. Gibbons
Professor

Zhihao Jia
Assistant Professor

Todd C. Mowry
Professor

Ameet Talwalkar
Assistant Professor

Eric Xing
Professor

Xupeng Miao
Postdoc

PhD Students

Daiyaan Arfeen

Pratik Fegade

Bohan Hou

Elias Jääsaari

Byungsoo Jeon

Hongyi Jin

Ruihang Lai

Zikun Li

Gabriele Oliaro

Suhas Jayaram Subramanya

Mengdi Wu

Mingkuan Xu

Zhihao Zhang

Yonghao Zhuang

Zhuoming Chen

Masters Students

Guoyao Li

Sudeep Agarwal

Shreyashri Biswas

Xinhao Cheng

April Yang

Undergraduate Students

Jinqi (Kathryn) Chen

Peiyuan Liao

Alumni

Yue Zhao
Ph.D., 2023. Assistant Professor at USC.

Muyang Li
M.S., 2023. PhD student at MIT

Tony Xi
M.S., 2023. Now at Apple

Zeyu Wang
M.S., 2023. Now at Yahoo

Andrew Gu
M.S., 2022. Now at Meta

Zhongyu Chen
M.S., 2022. Now at LinkedIn

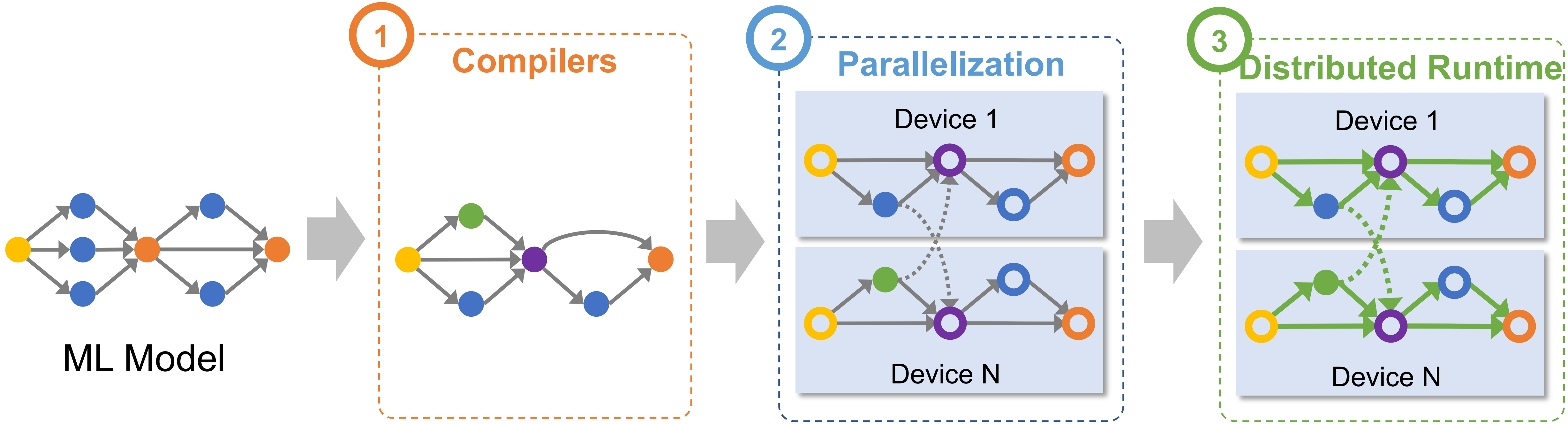
Bowen Chen
M.S., 2022. Now at Annapurna Labs

Neeraj Aggarwal
M.S., 2022. Now at Applied Intuition

catalyst

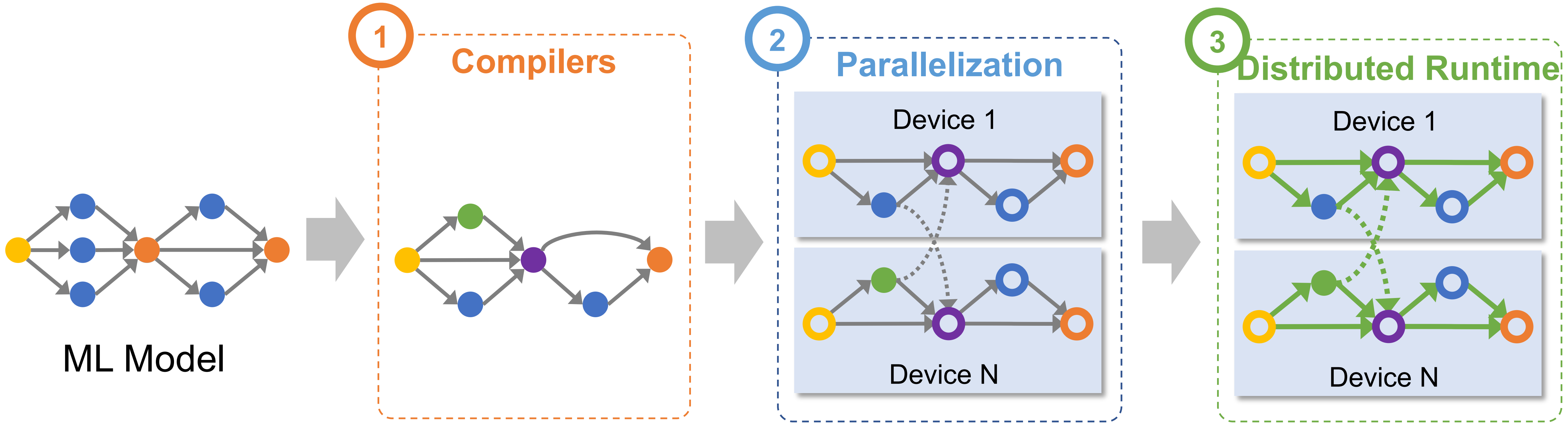
https://catalyst.cs.cmu.edu/

This Lecture:
Three Lessons Learned from Our MLSys Research



1. Automated approaches can offer **3-10x** improvement for most tasks
2. Joint optimization is **critical**
3. Combing systems and ML optimizations is **promising but challenging**

Lesson 1: Automated Approaches Offer 3-10x Improvement



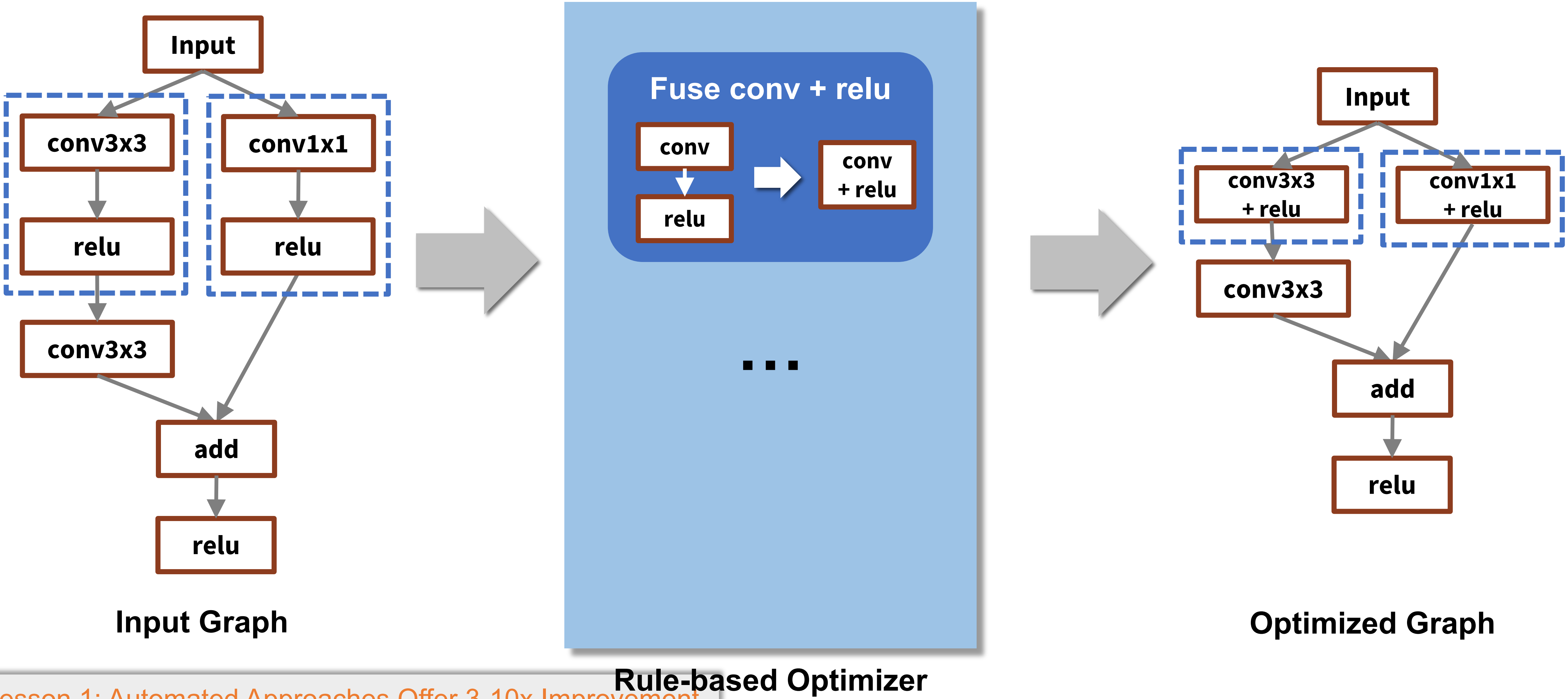
Case Study 1:
TASO: Up to 3.1x

Case Study 2:
FlexFlow: Up to 10x

catalyst

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

Case Study: Current Rule-based Graph Optimizations



Case Study:
Current Rule-based Graph Optimizations

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

Fuse conv + relu

Fuse conv + batch normalization

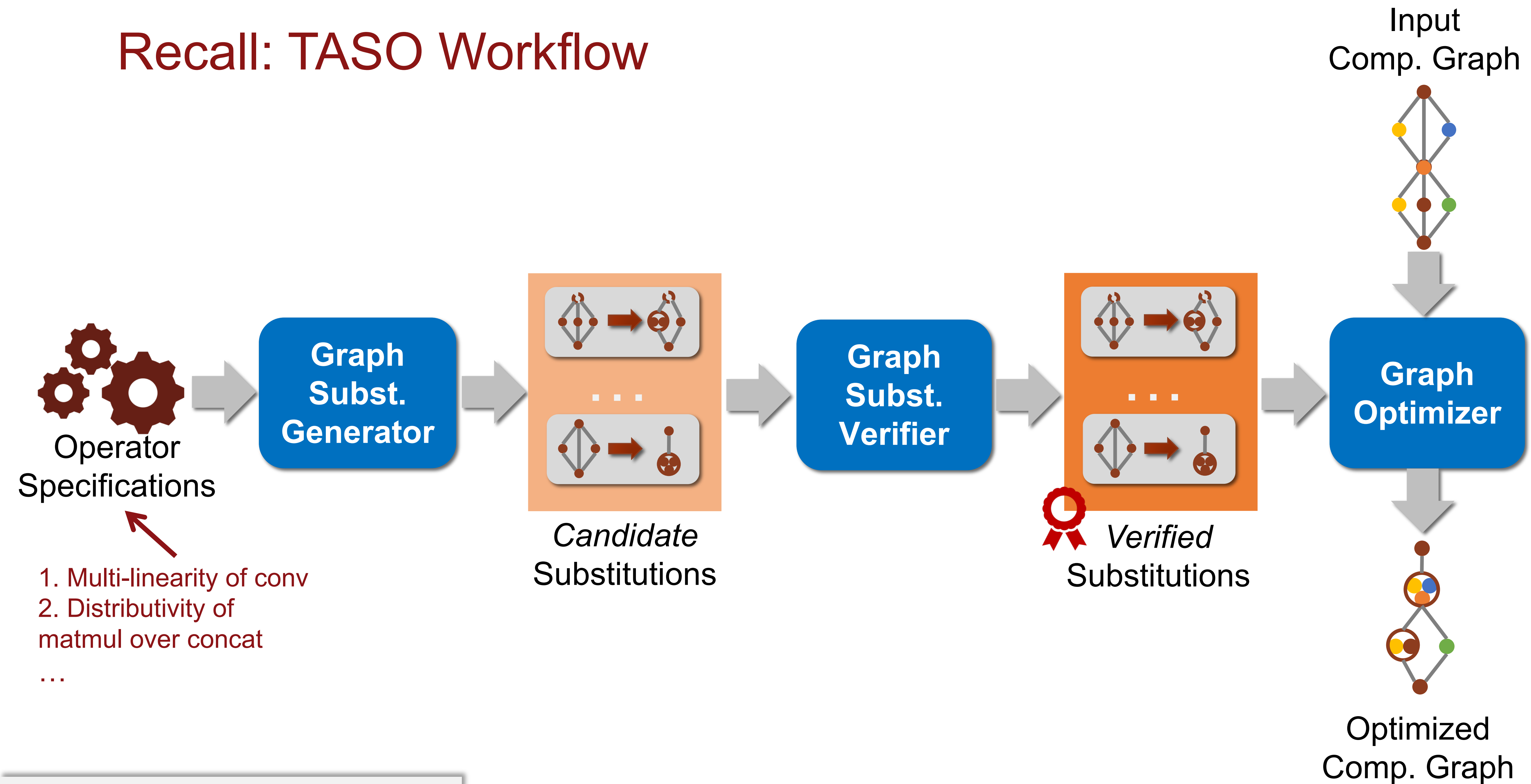
Fuse multi. convs

...

Rule-based Optimizer

```
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
```

Recall: TASO Workflow



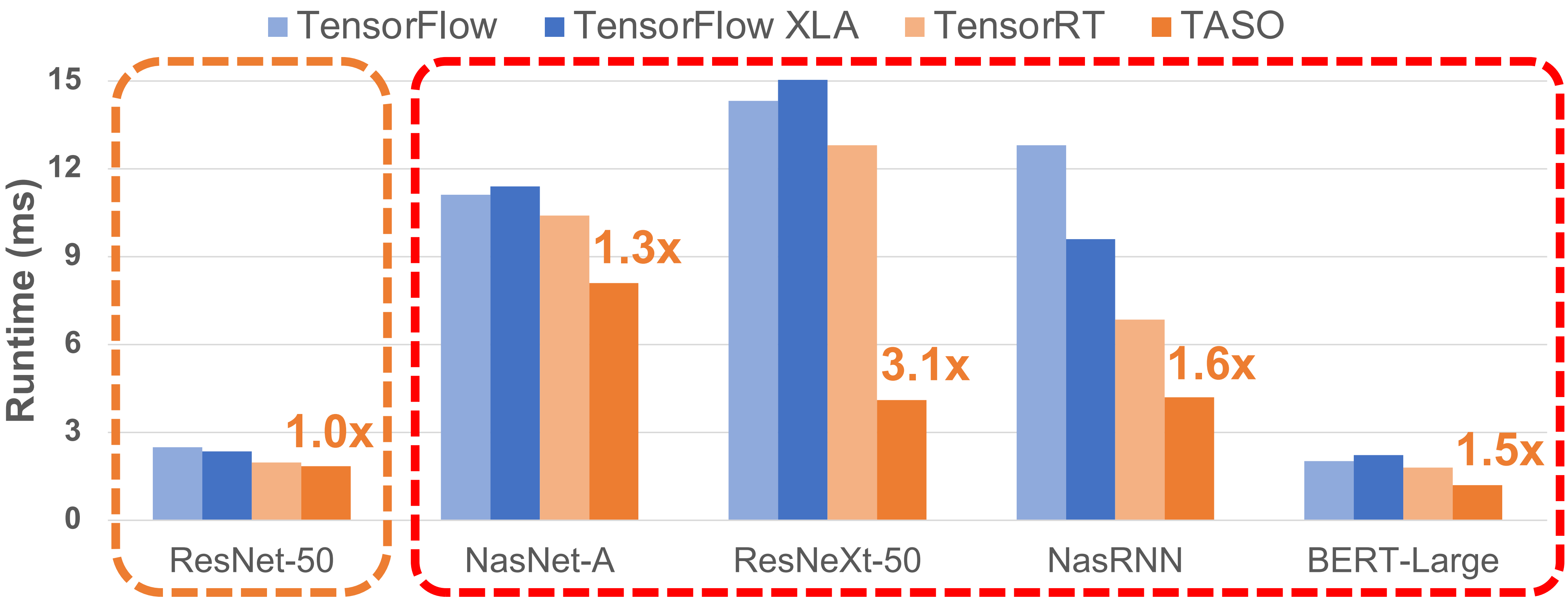
* Lecture 8: Automated Graph Optimizations

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

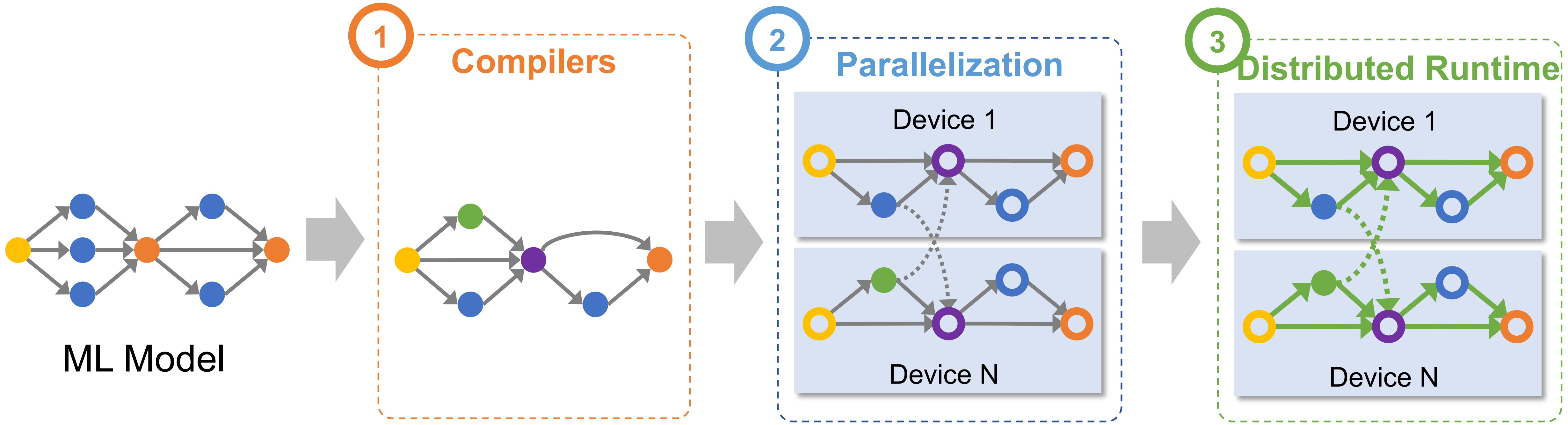
End-to-end Inference Performance (Nvidia V100 GPU)



Competitive on standard models

Larger speedups on emerging models

Lesson 1: Automated Approaches Offer 3-10x Improvement



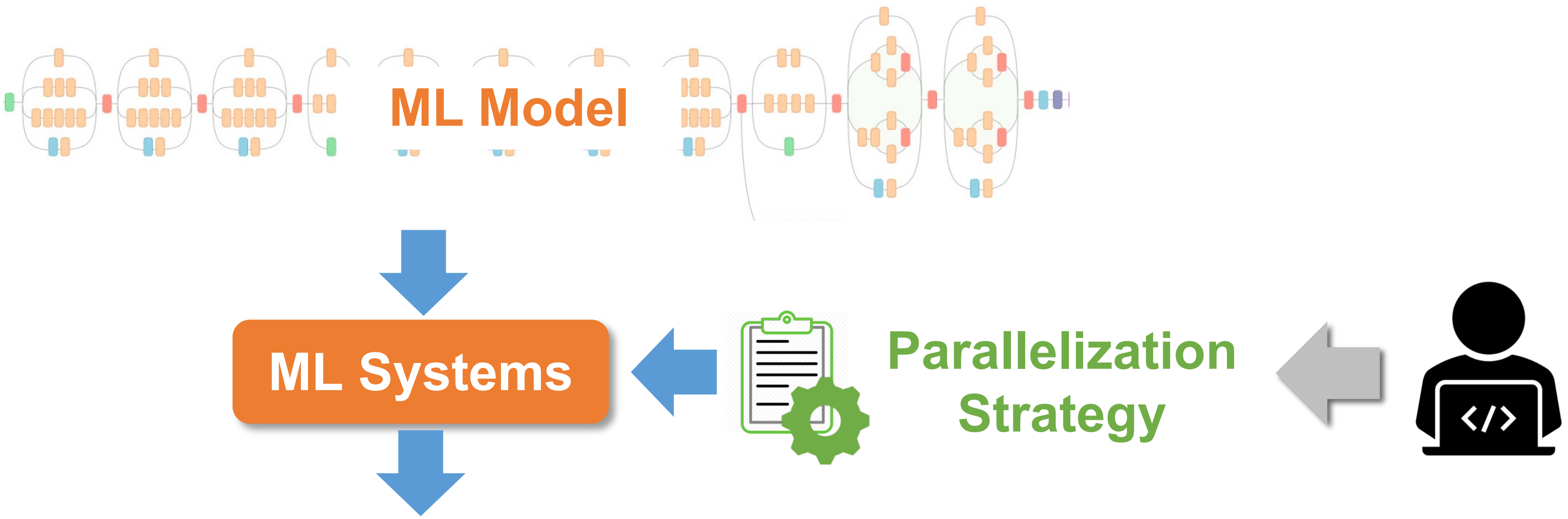
Case Study 1:
TASO: Up to 3.1x

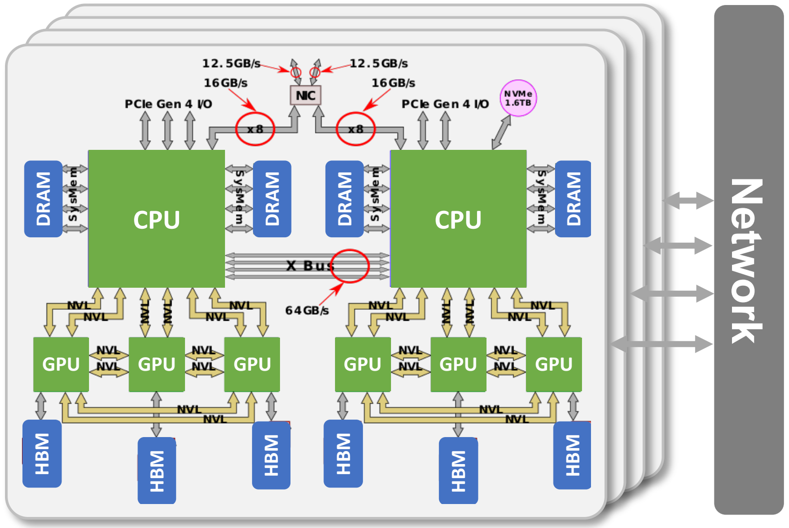
Case Study 2:
FlexFlow: Up to 10x

catalyst

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

Challenges of Parallelizing DNN Training



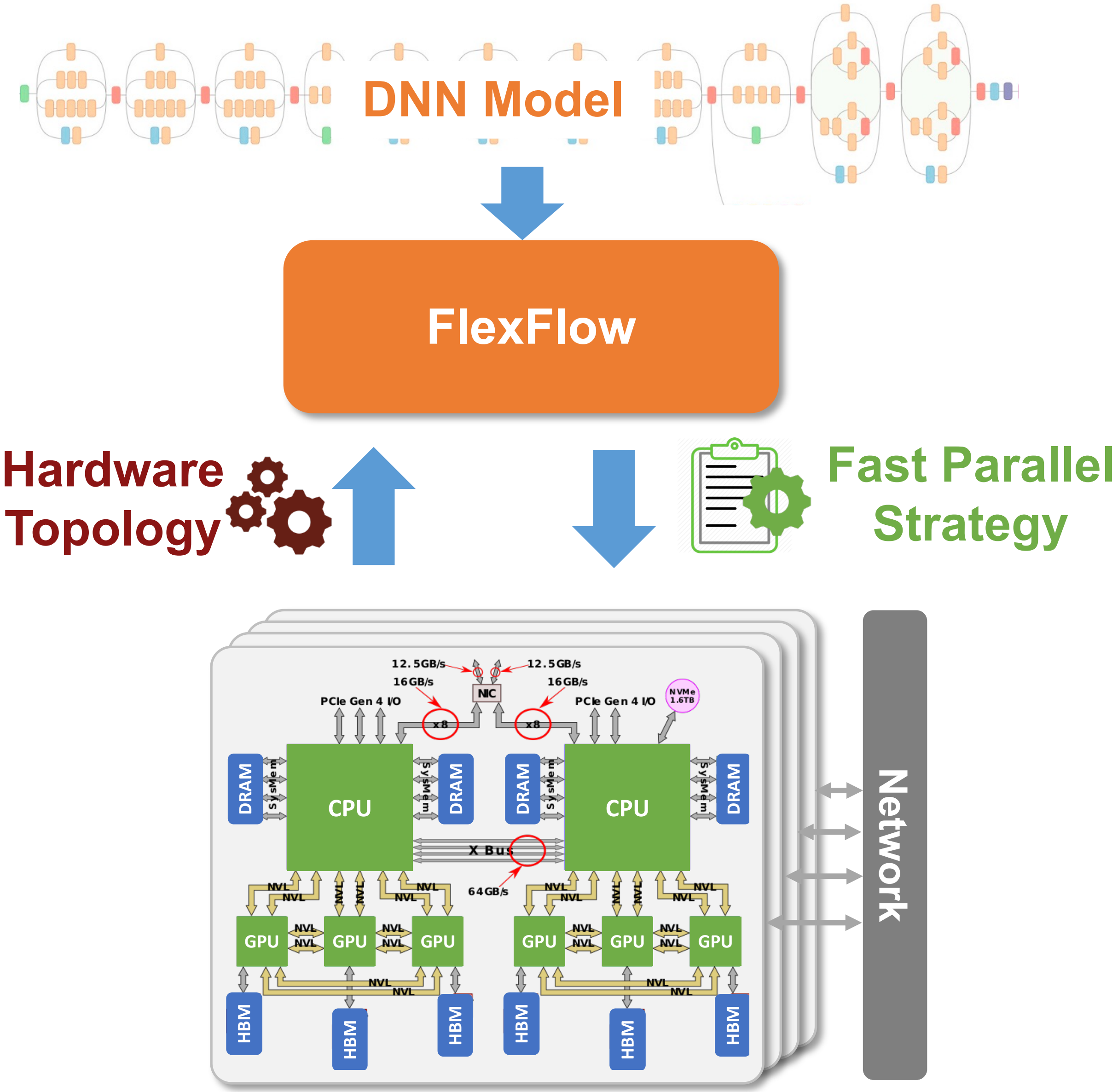


Limitations:

- Hard to manually design and implement
- 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: Searching for Efficient Parallelization Strategies

A **search space** of possible parallelization strategies

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

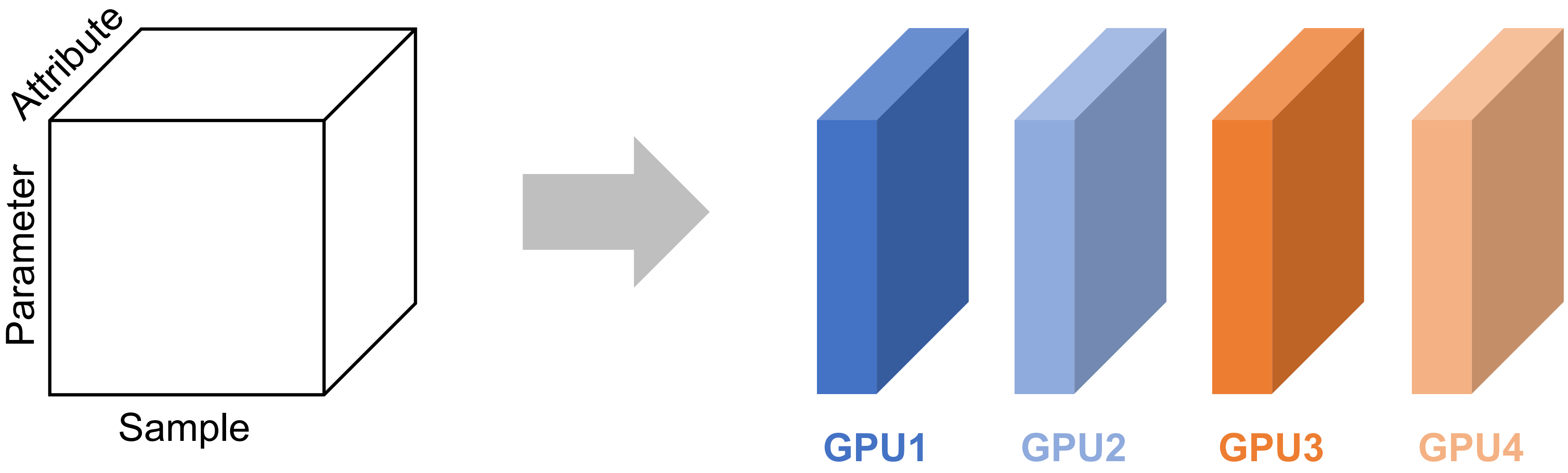
= **Fast and Scalable Parallelization strategies**

The SOAP Search Space

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

The SOAP Search Space

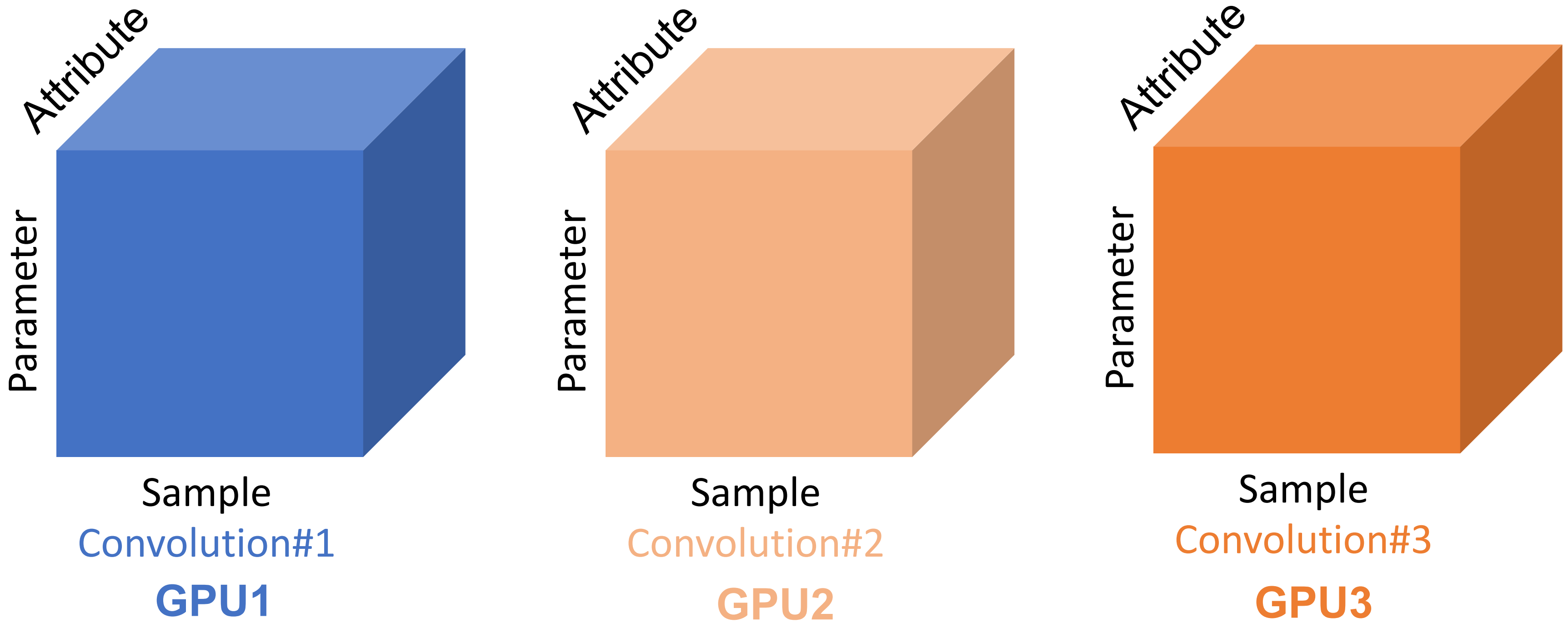
- **S**amples: partitioning training samples (Data Parallelism)
- **O**perators
- **A**ttributes
- **P**arameters



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

The SOAP Search Space

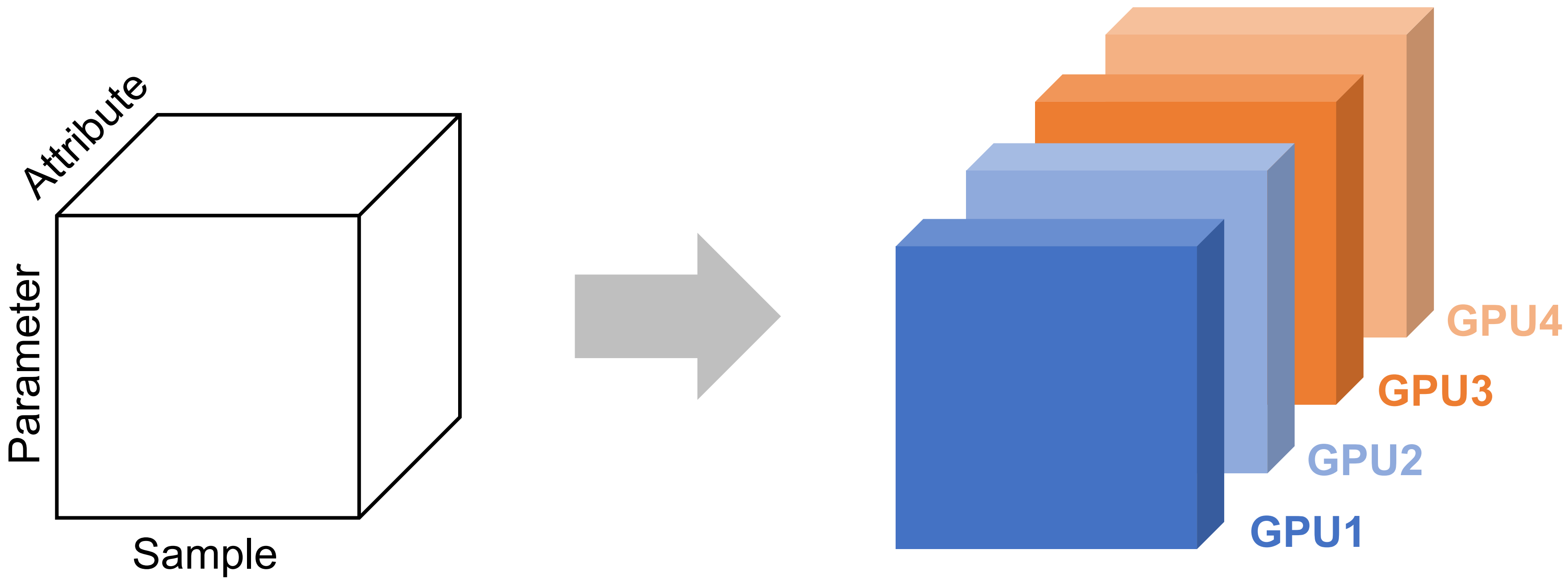
- **S**amples: partitioning training samples (Data Parallelism)
- **O**perators: partitioning ML operators (Model Parallelism)
- **A**ttributes
- **P**arameters



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

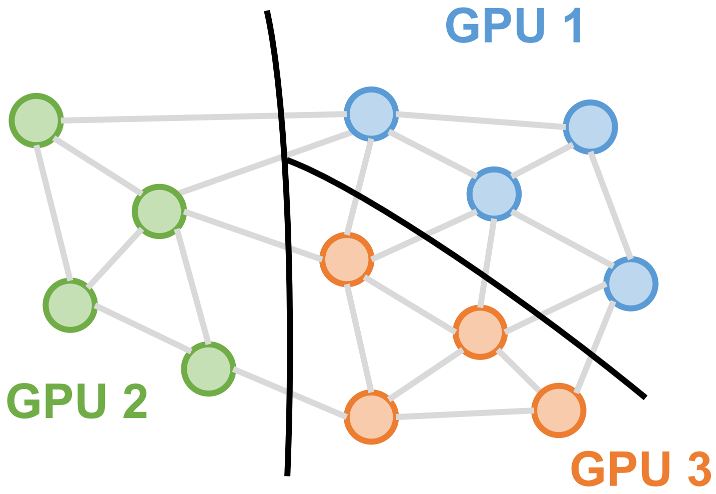


Parallelizing a 1D convolution in ***Attribute***

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

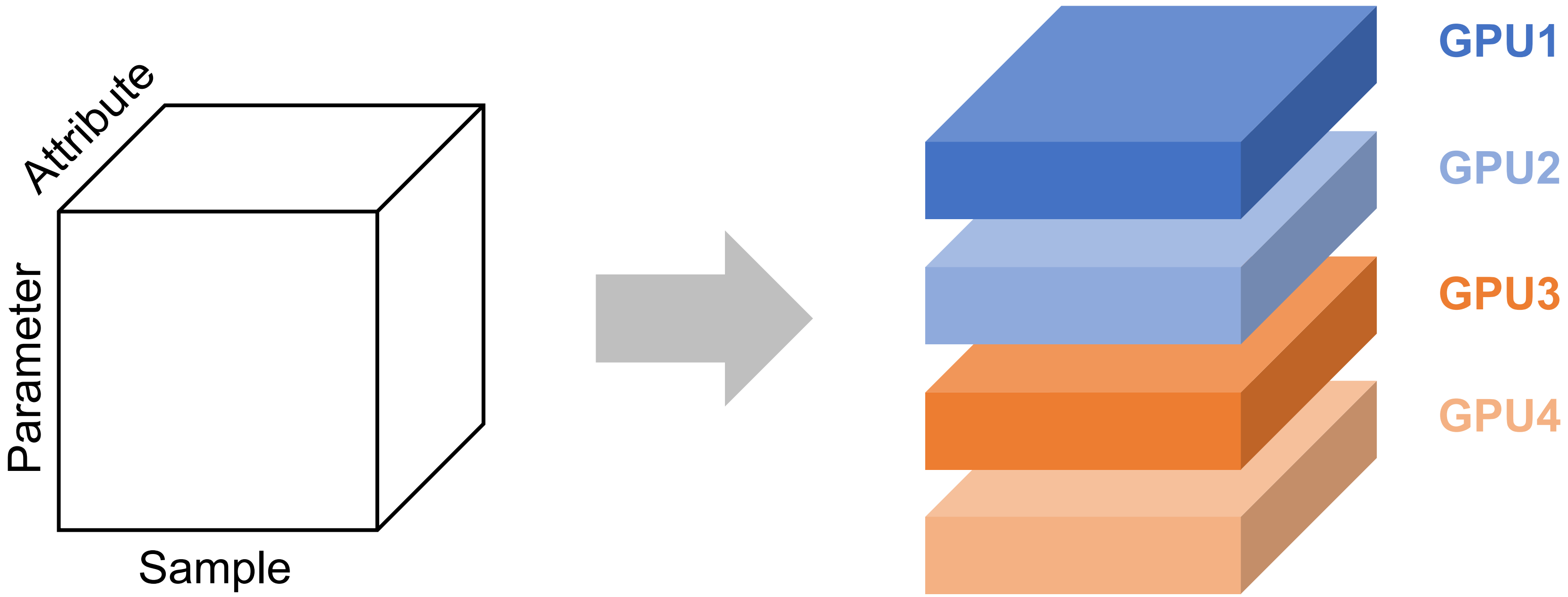
Attribute parallelism for CNNs
(attribute = pixel)



Attribute parallelism for GNNs
(attribute = vertex)

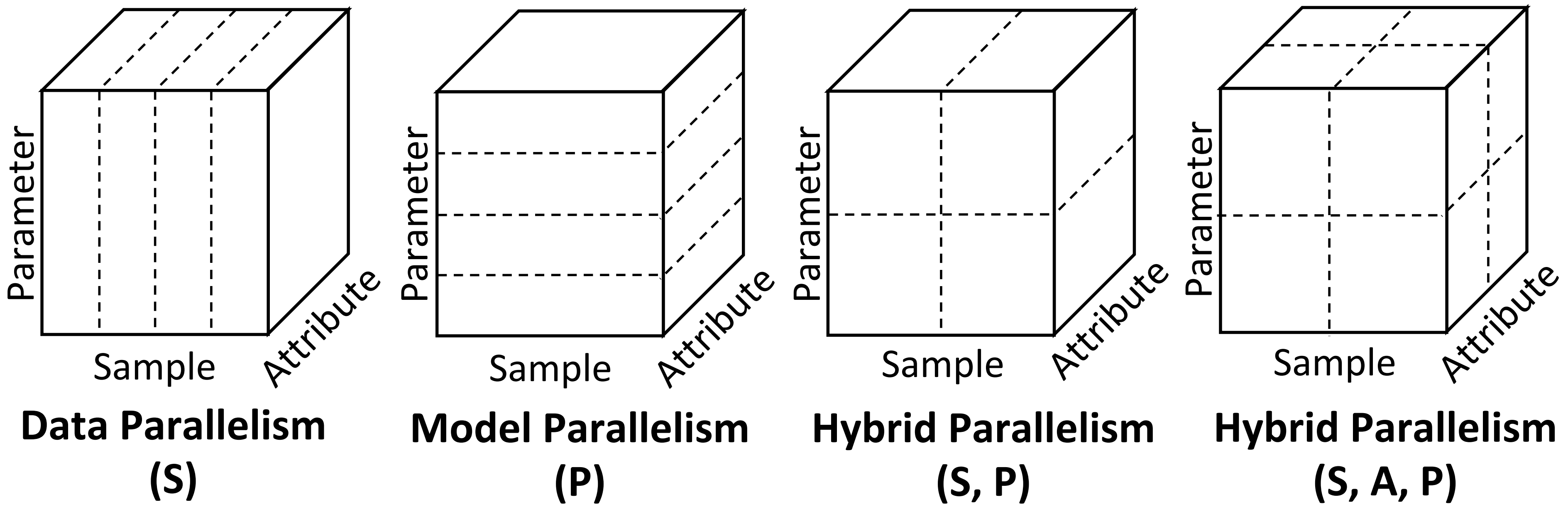
The SOAP Search Space

- **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 (Tensor Model Parallelism)



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

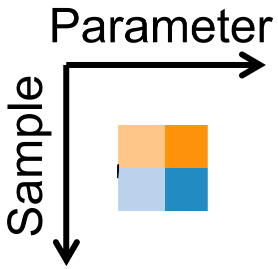
Hybrid Parallelism in SOAP

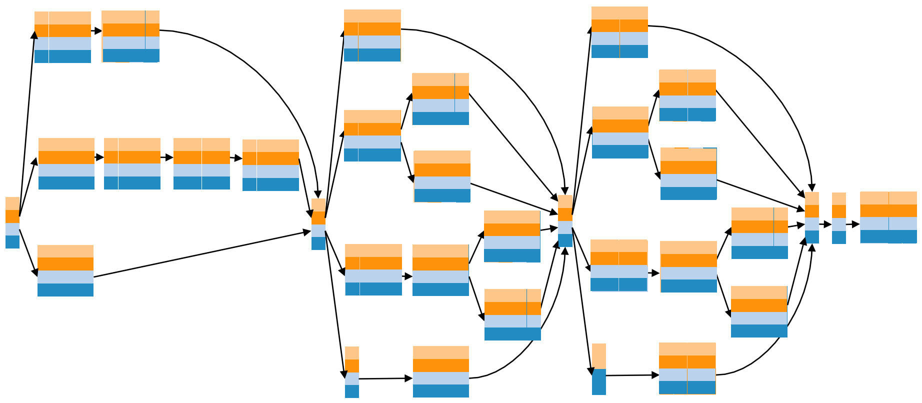


Example parallelization strategies for 1D convolution

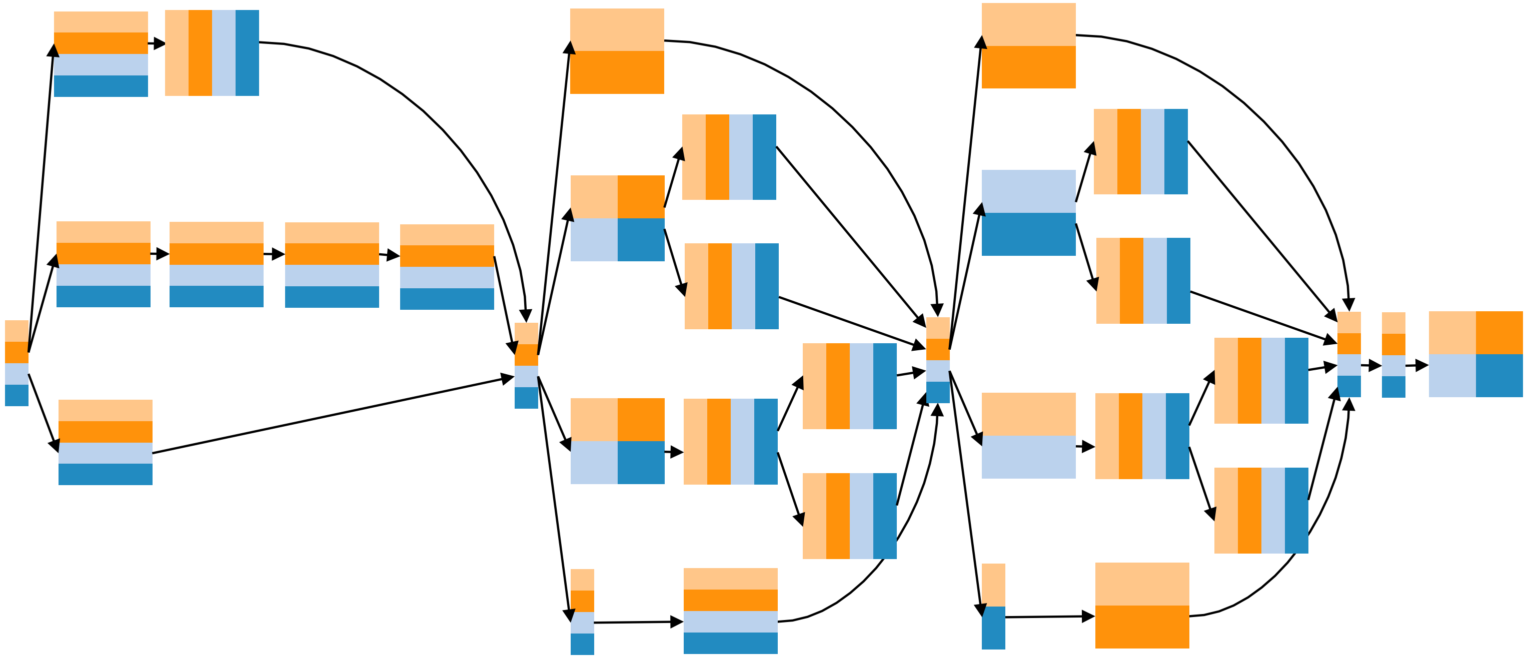
Different strategies perform the same computation.

GPU 1
GPU 2
GPU 3
GPU 4

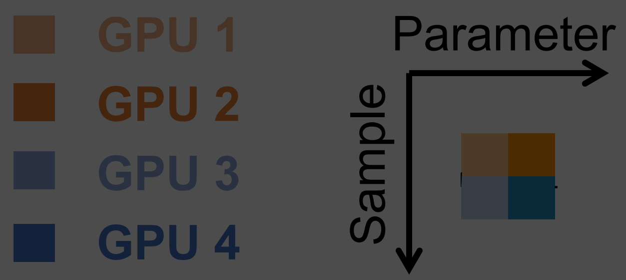


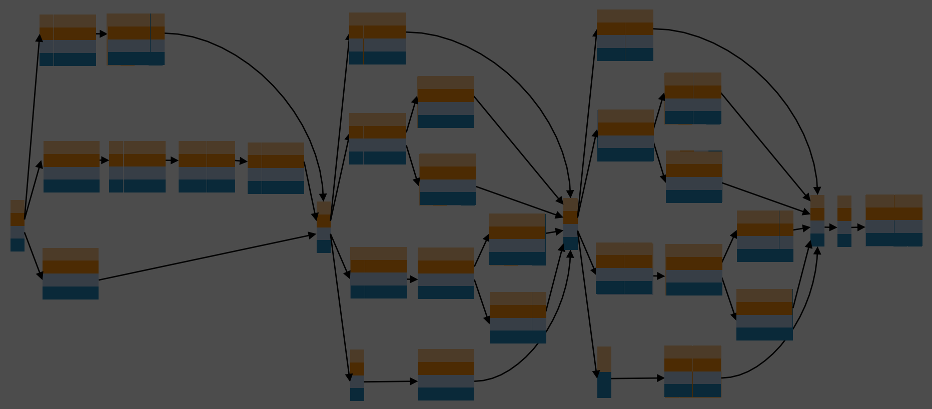


Data parallelism

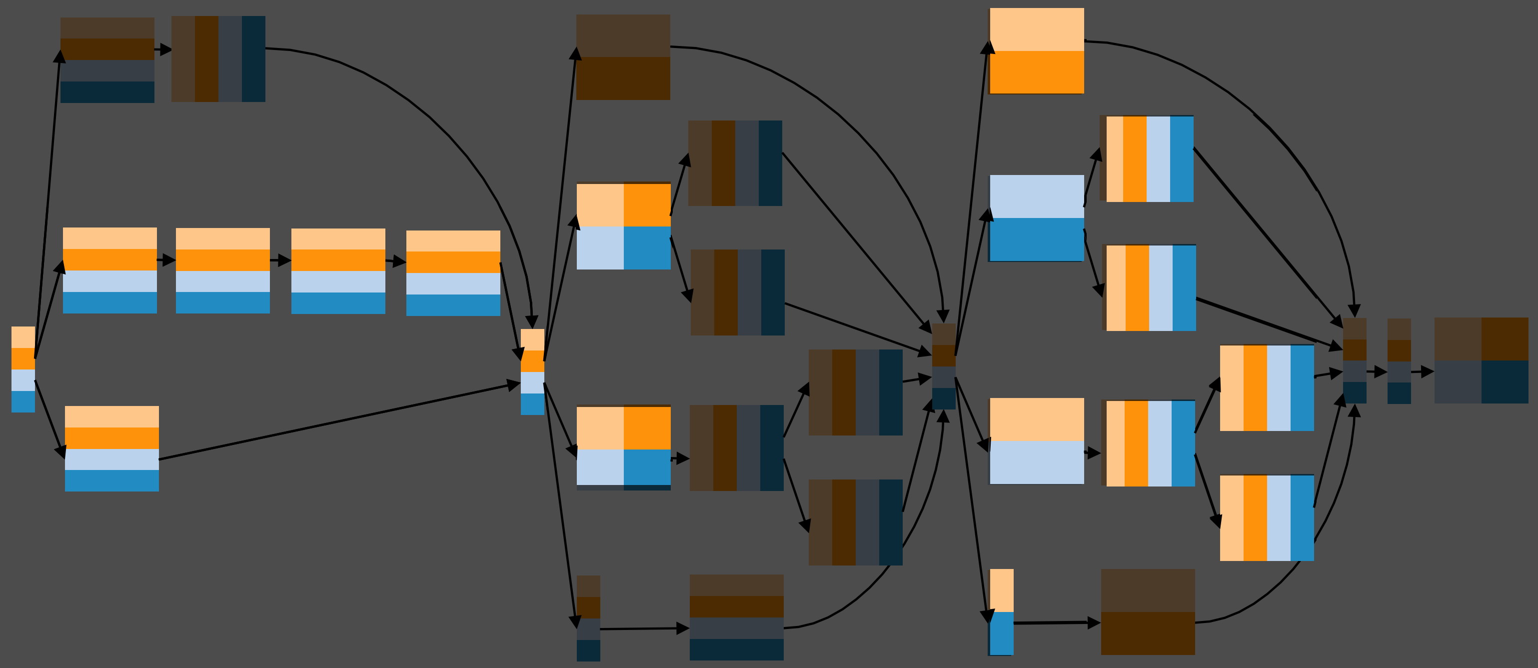


A parallelization strategy in SOAP **(1.2x faster)**





Data parallelism



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

Challenges of Discovering Fast Strategies in SOAP

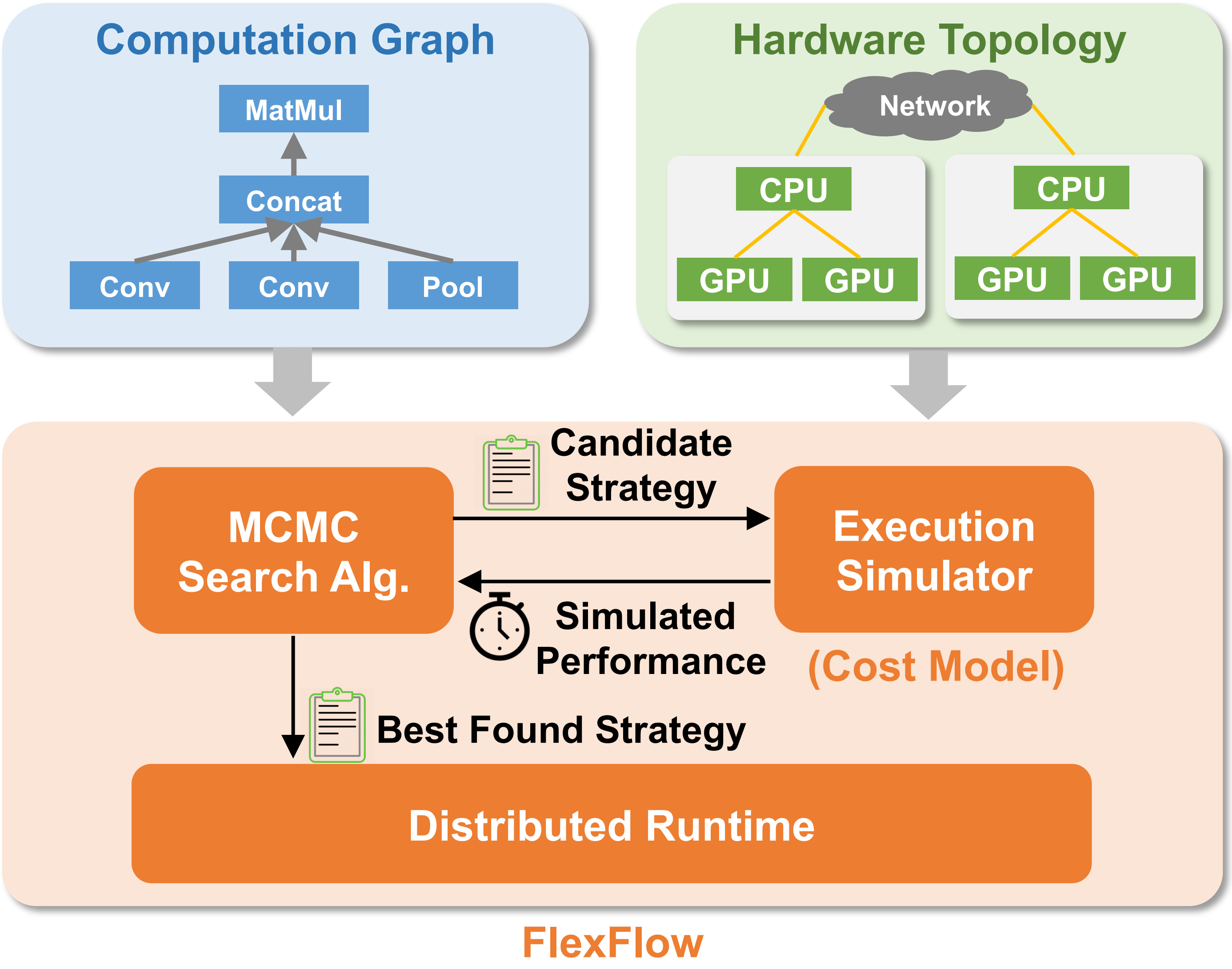
1. SOAP contains billions or more possible strategies

MCMC search algorithm

2. Evaluating a strategy on hardware is too slow

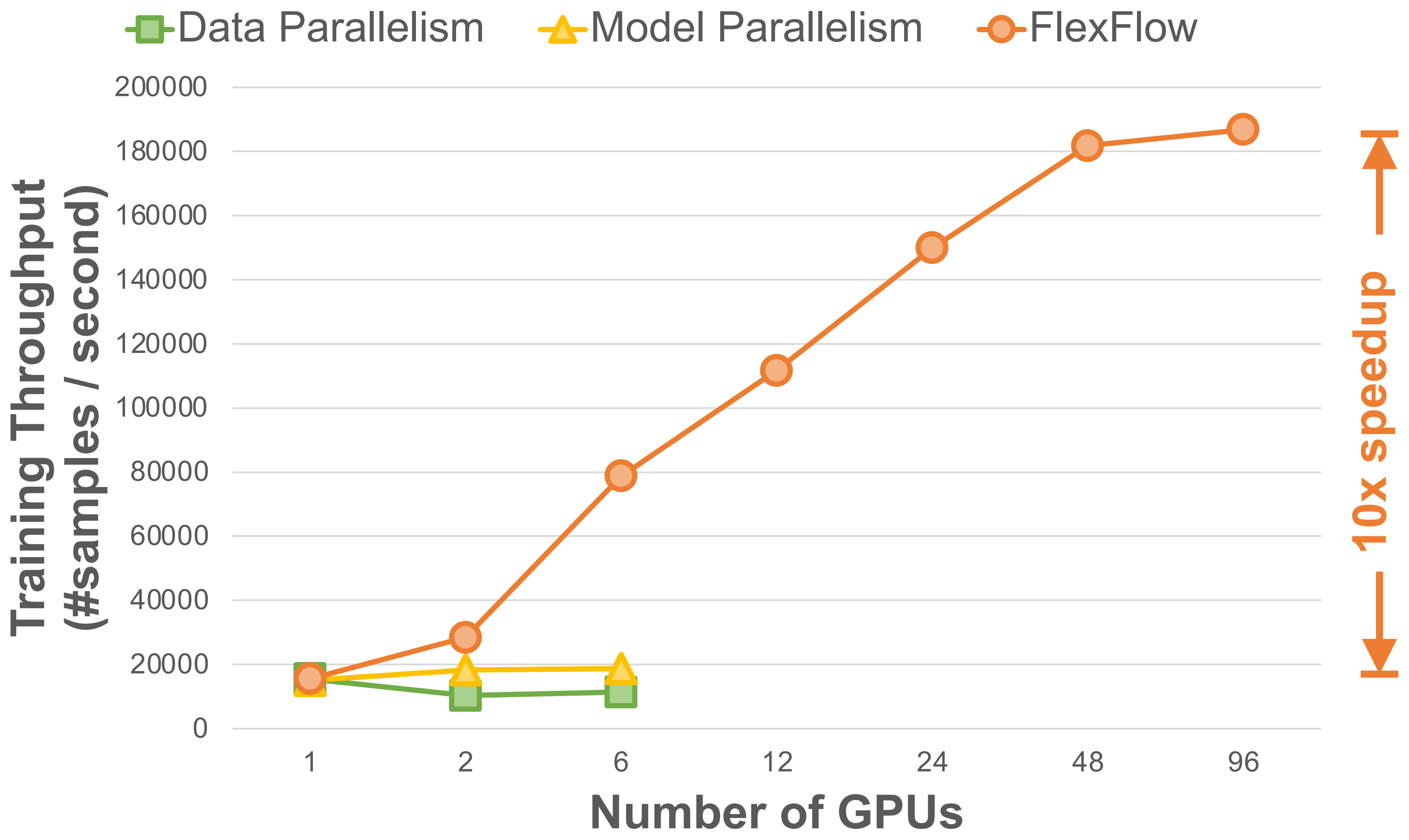
Execution simulator

FlexFlow Overview

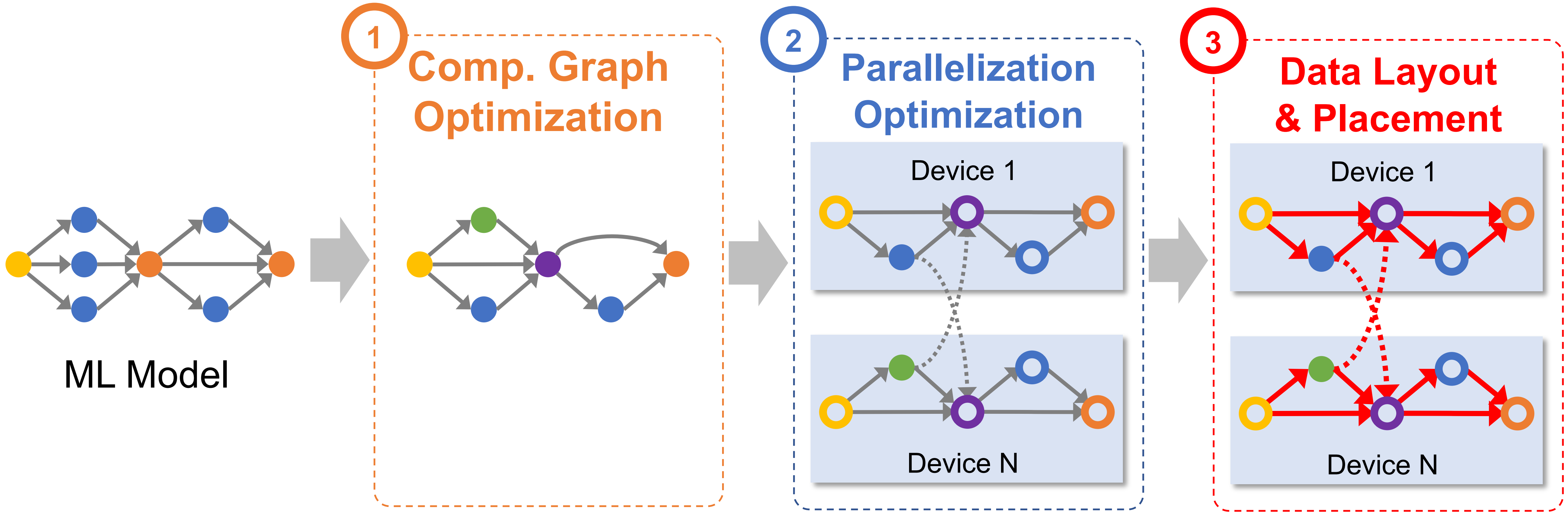


Deep Learning Recommendation Model (DLRM)

A deep learning model for ads recommendation



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

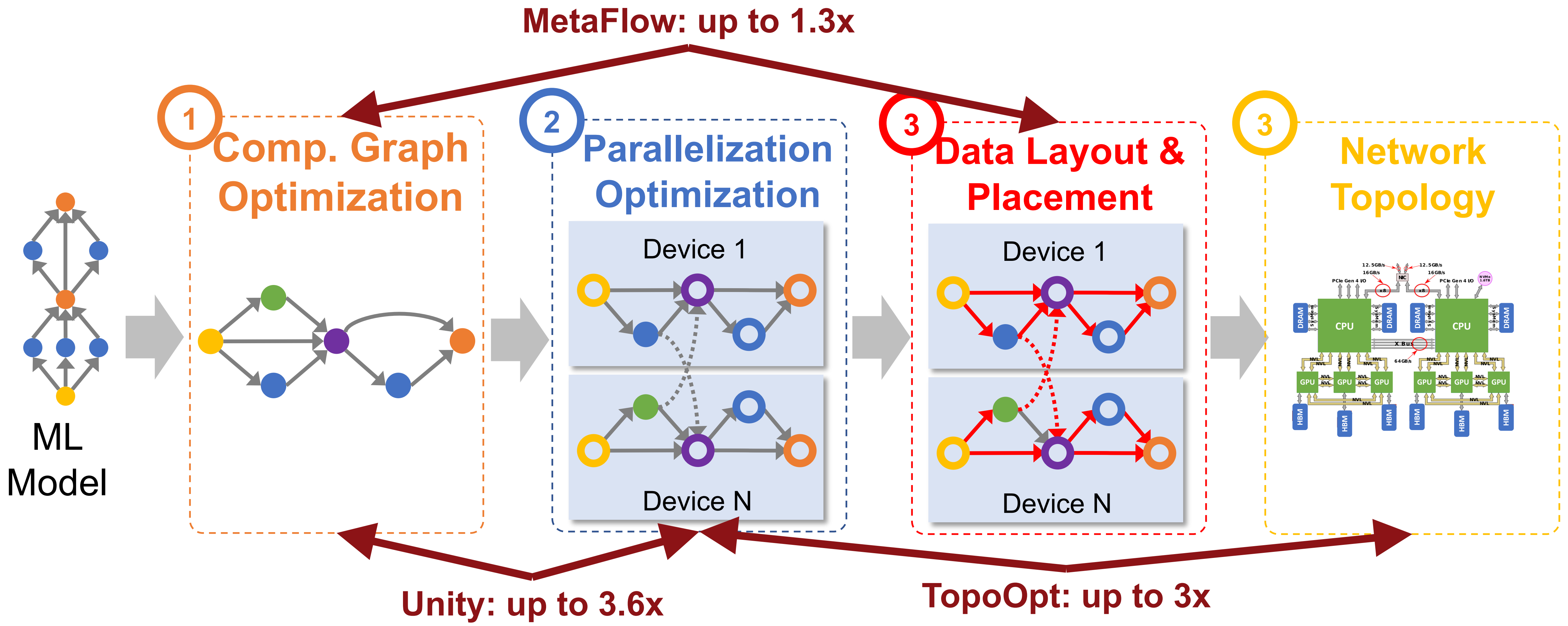
Common Advantages of Automated Approaches

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

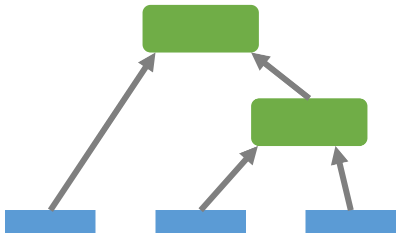
- **Less engineering effort**: code for discovering optimizations is generally much less than manual implementation of these optimizations

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

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

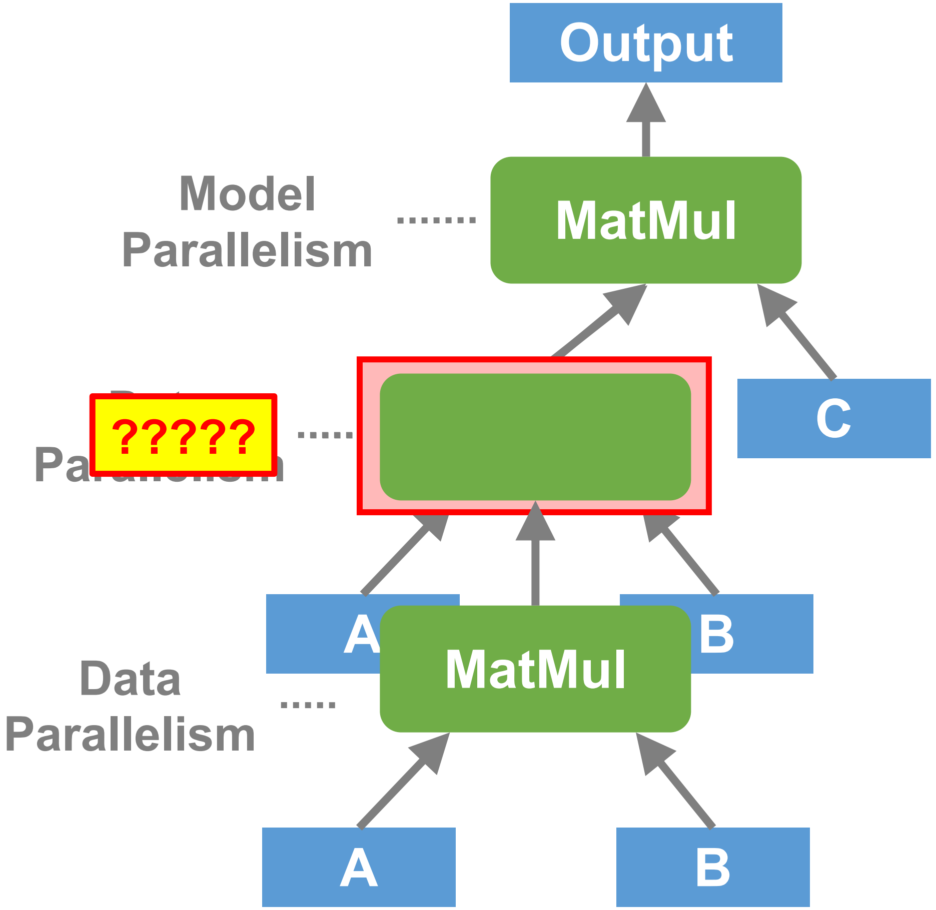


Auto-Parallelization

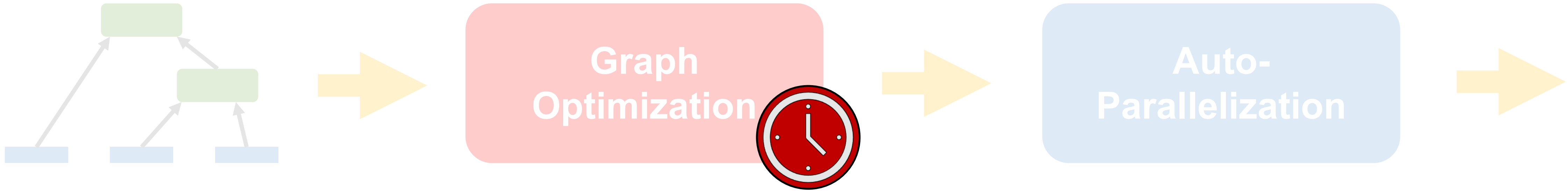
Graph Optimization

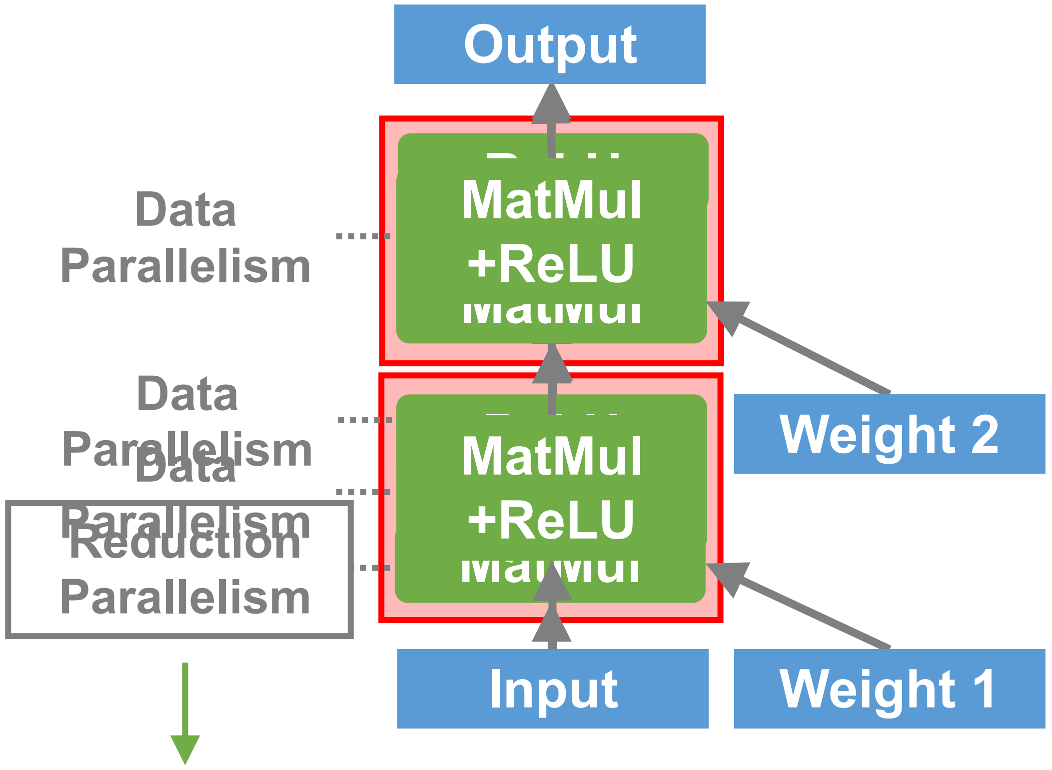
?

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

Representation

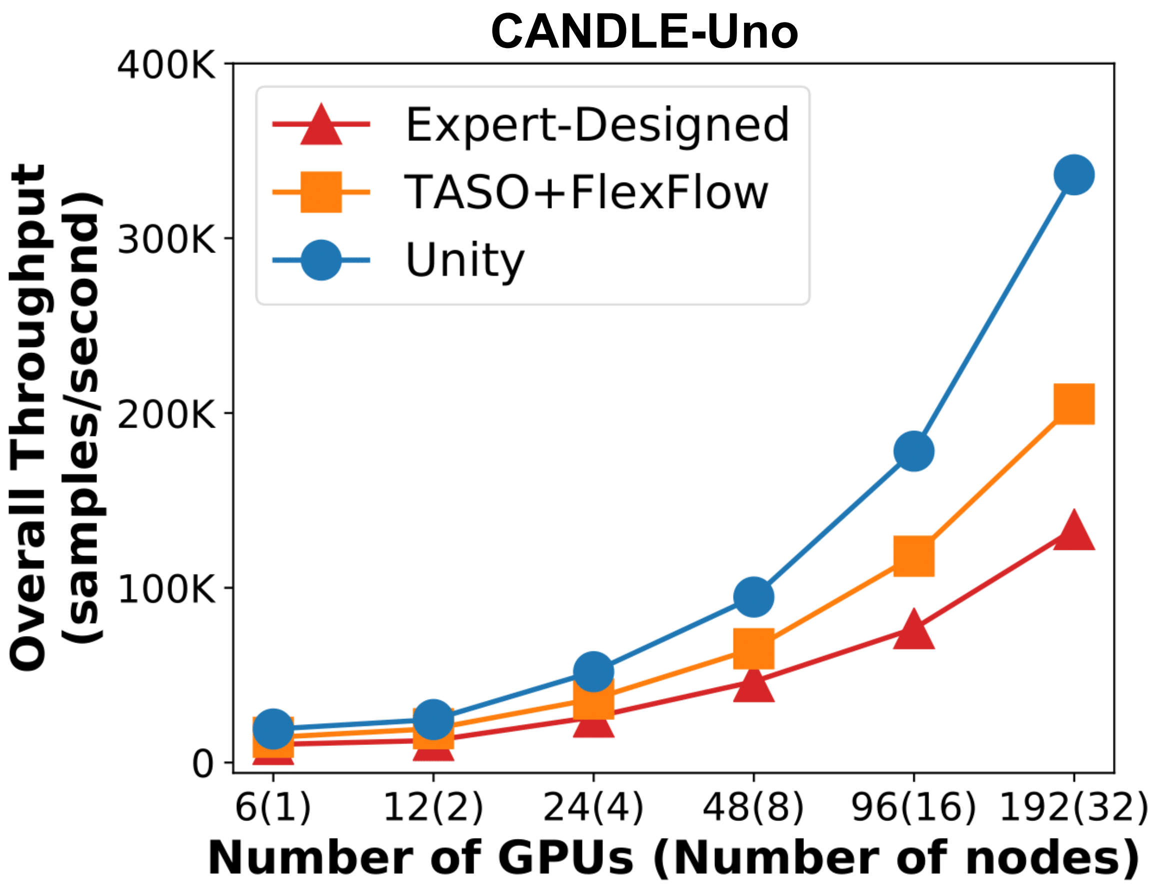
Parallel Computation Graph (PCG)

Unity

Scalability

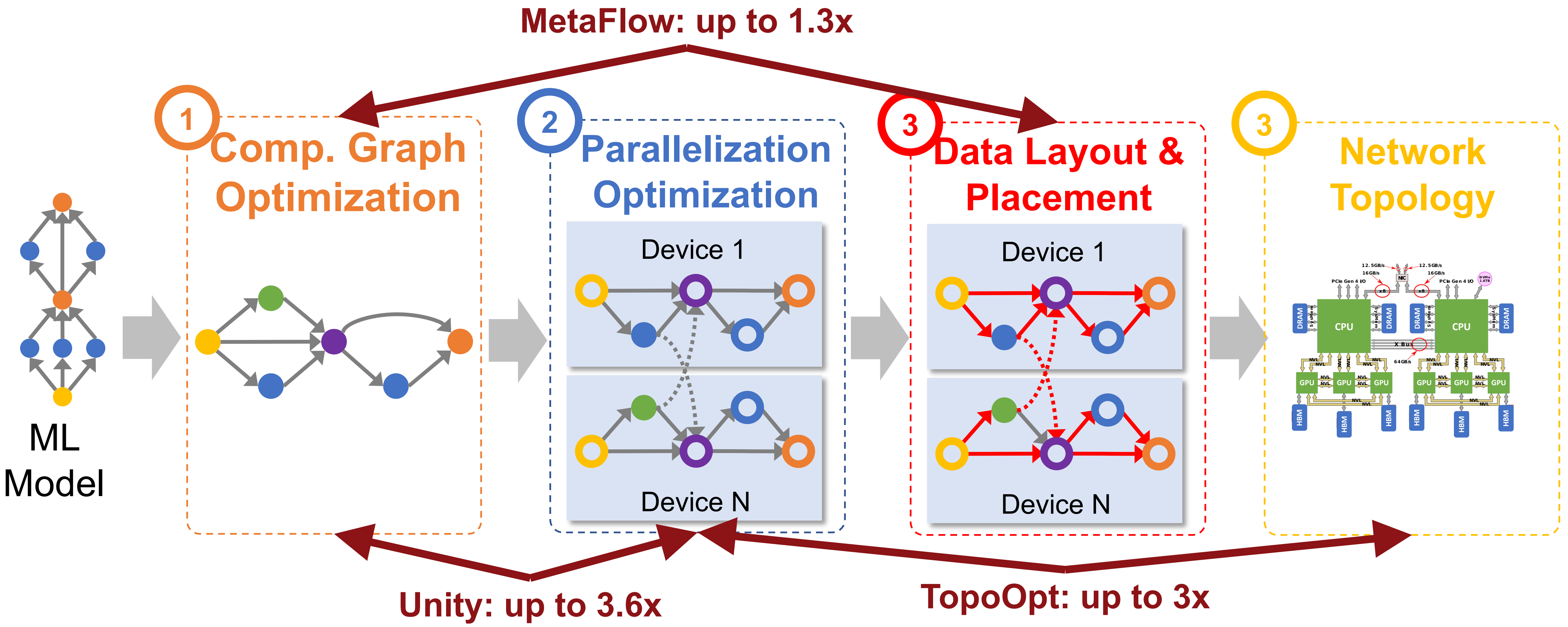
Hierarchical Search Algorithm

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
- Low-Rank Adaptation
- Distillation
- Neural Architecture Search

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

Systems Optimizations

Pro: preserve equivalence

ML Optimizations

Pro: better performance

- Faster ML operators
- Less Computation

Achieve the best of both worlds?

Equivalent Optimizations

Approximate 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**