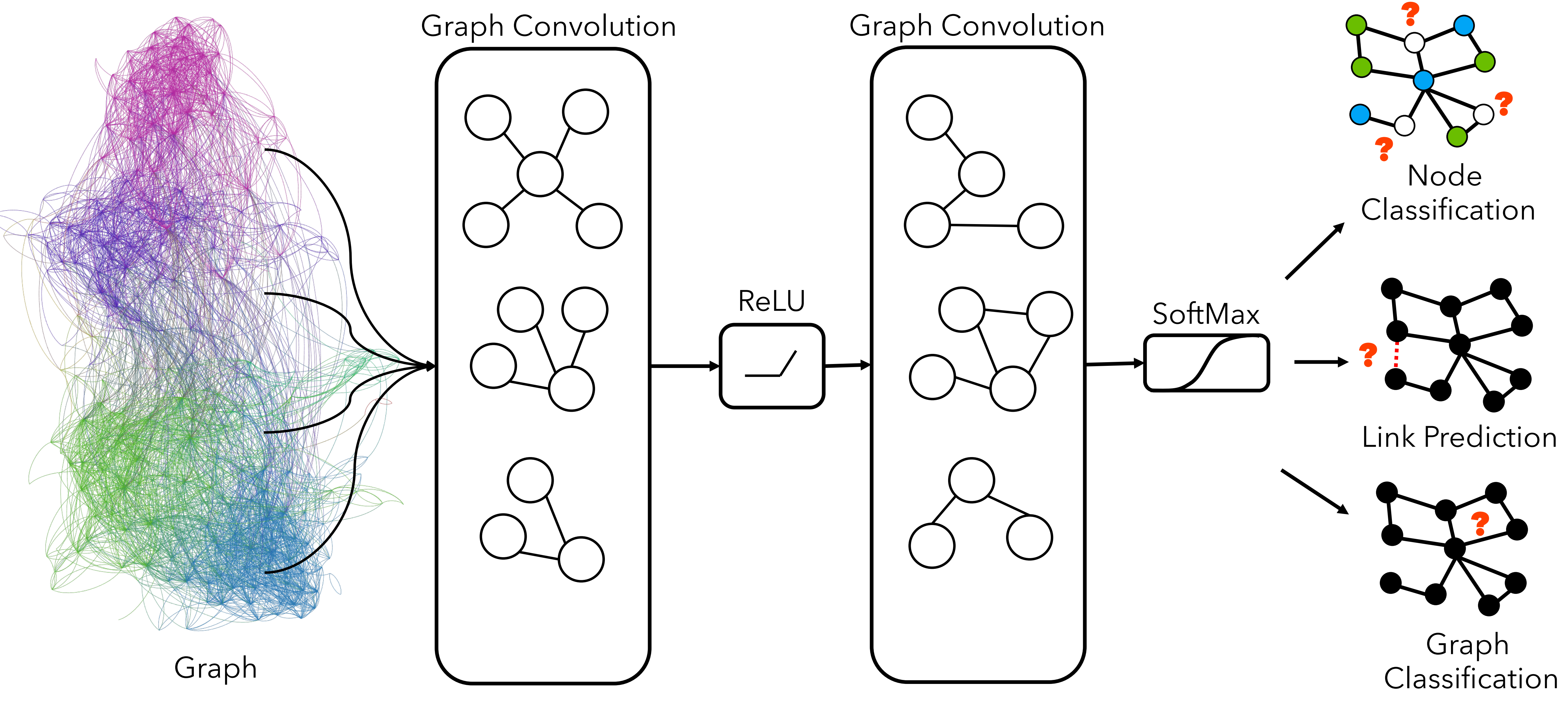
P^3: Distributed Deep Graph Learning at Scale

Swapnil Gandhi, Anand Iyer
Microsoft Research

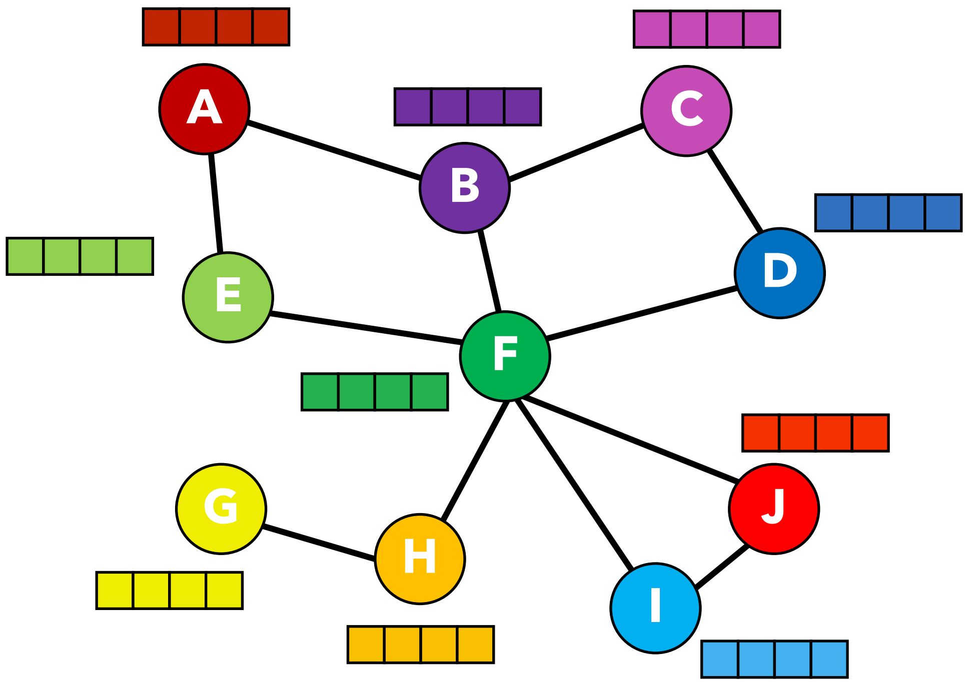
OSDI 2021

Graph **Neural Networks**

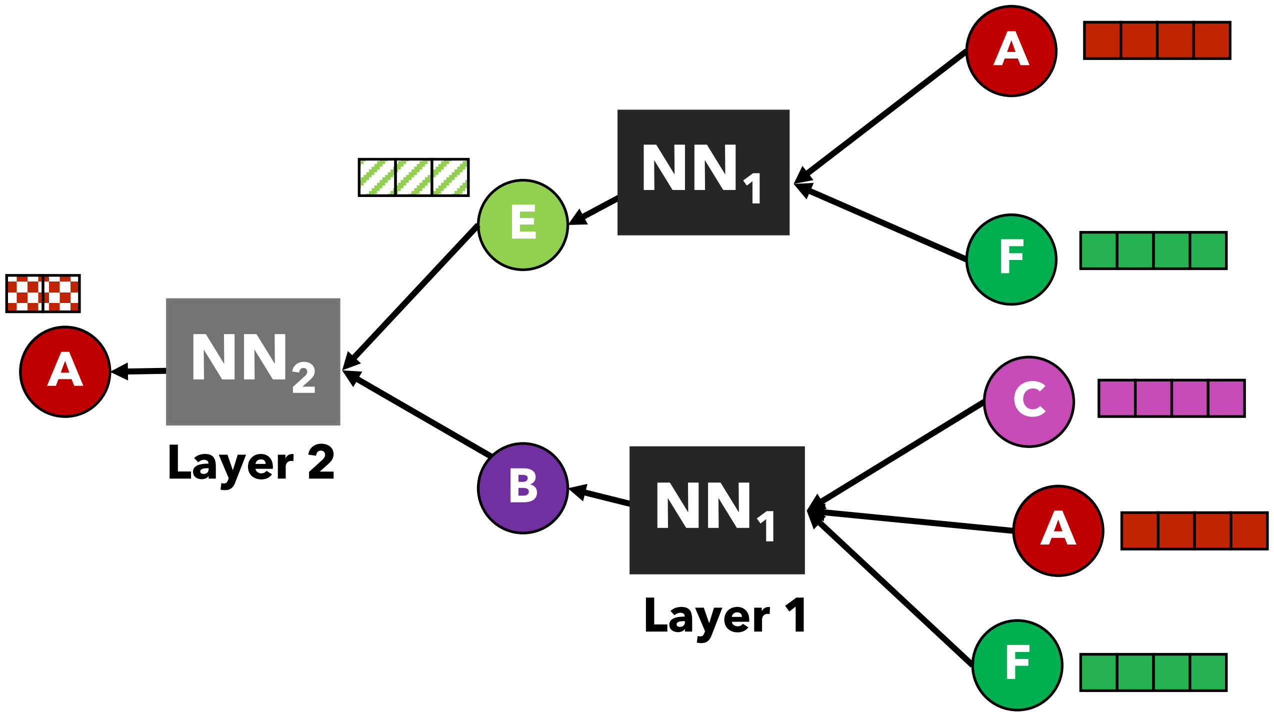


Graph **Neural Networks**

Graph Structure: **What** (to propagate?)
Neural Network: **How** (information is transformed)



Input Graph



2-hop computation graph of A

GNN **Training**

Large Graphs

Millions of nodes,
billions of edges

Hundreds or
thousands of features

New Models

Several proposals: GCN,
GAT, GIN, …

More **sophisticated**,
complex architectures

Significant interest in **distributed** GNN training

Distributed Graph Neural Networks

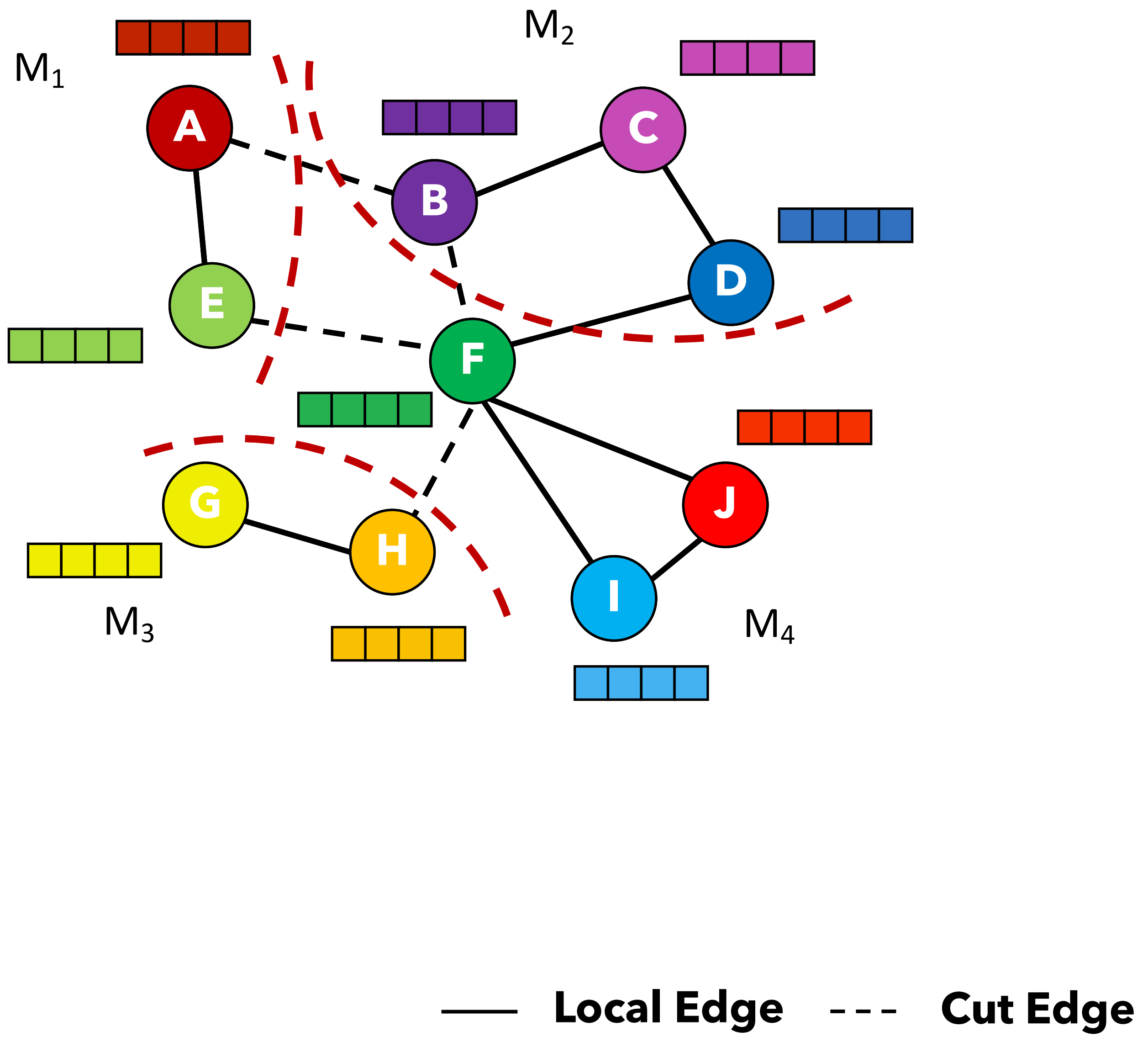
=

Distributed Graph Processing Techniques

+

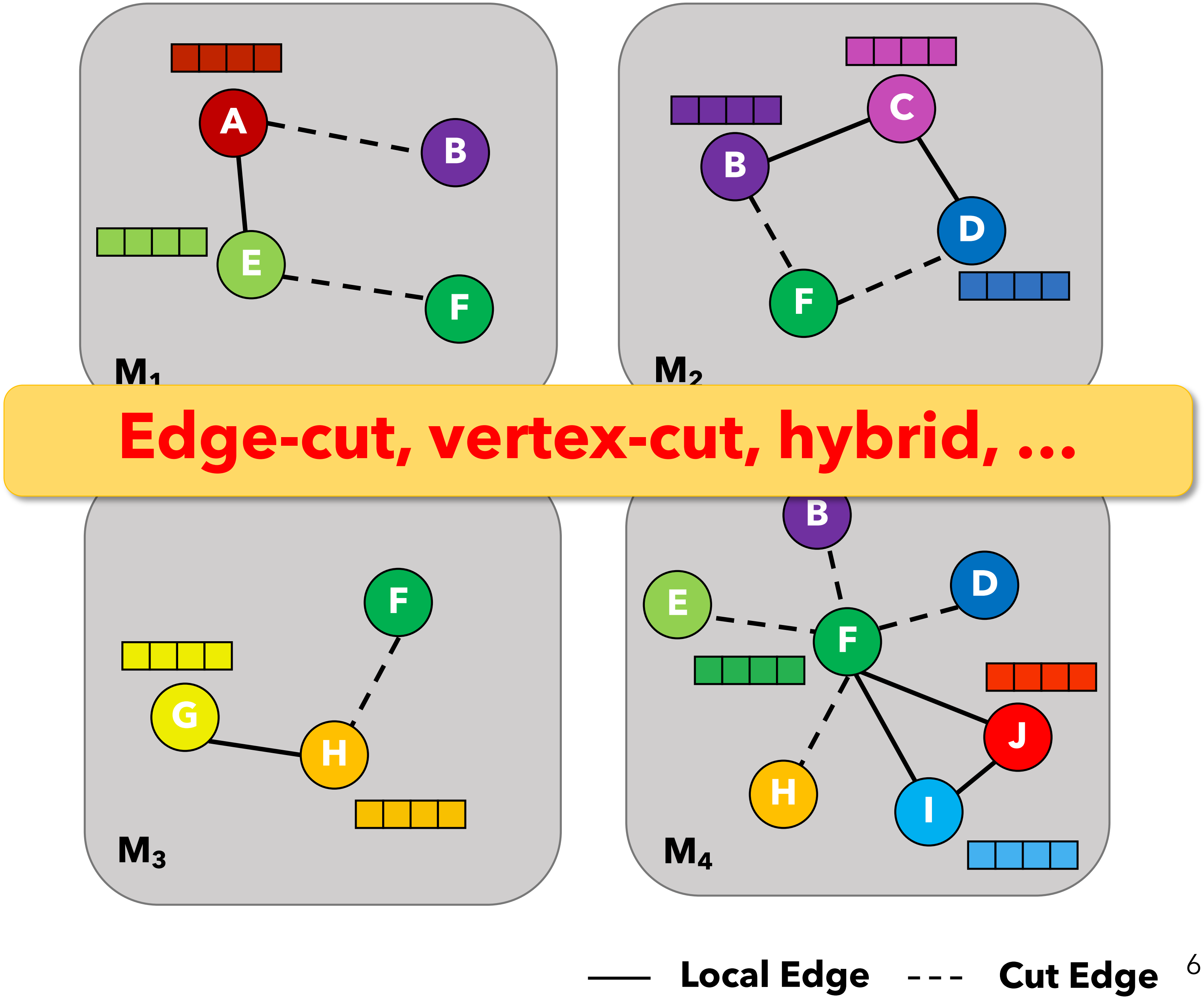
Distributed Neural Network Techniques

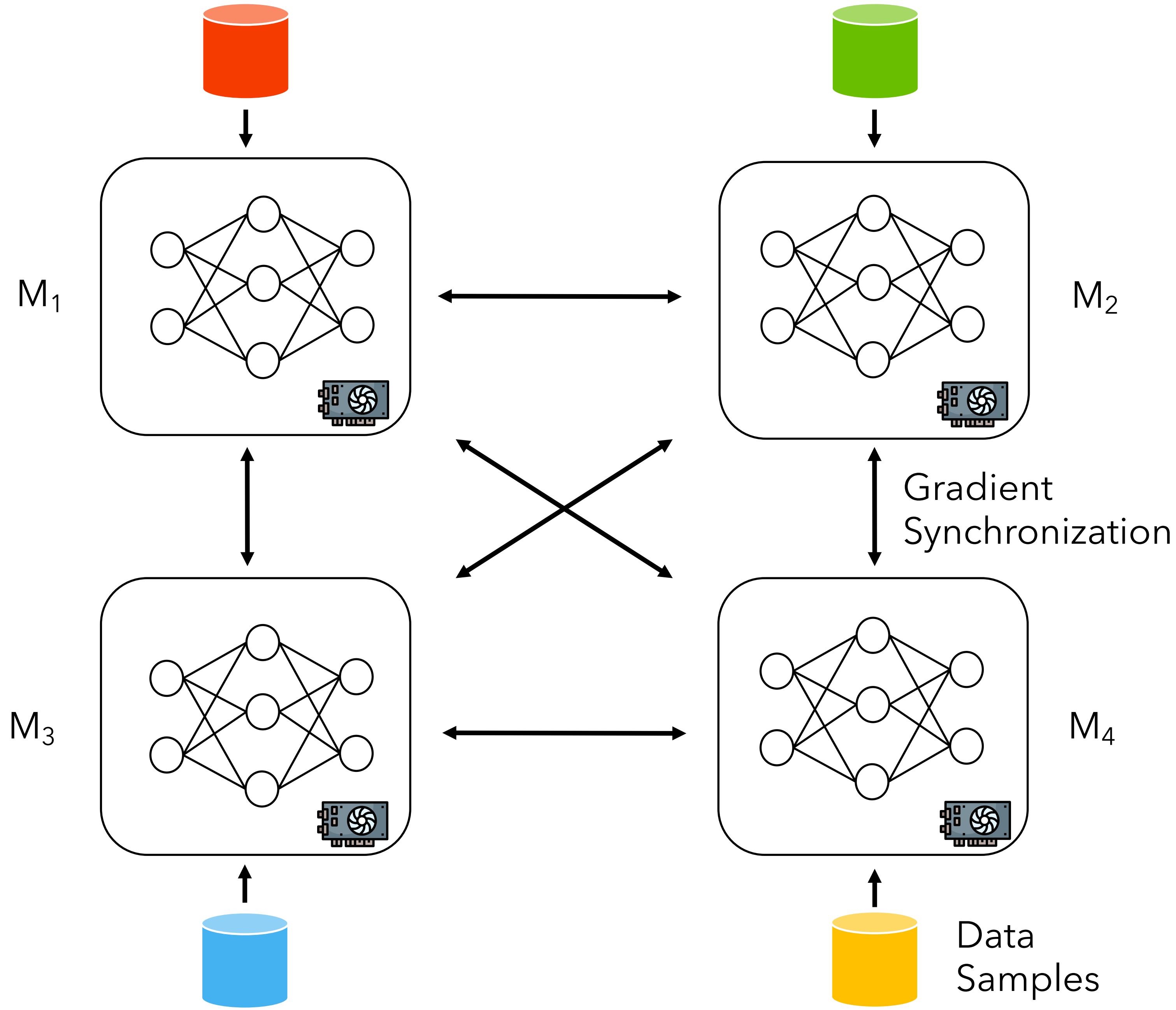
Graph Partitioning



Distributed Graph Processing Techniques

Graph Partitioning

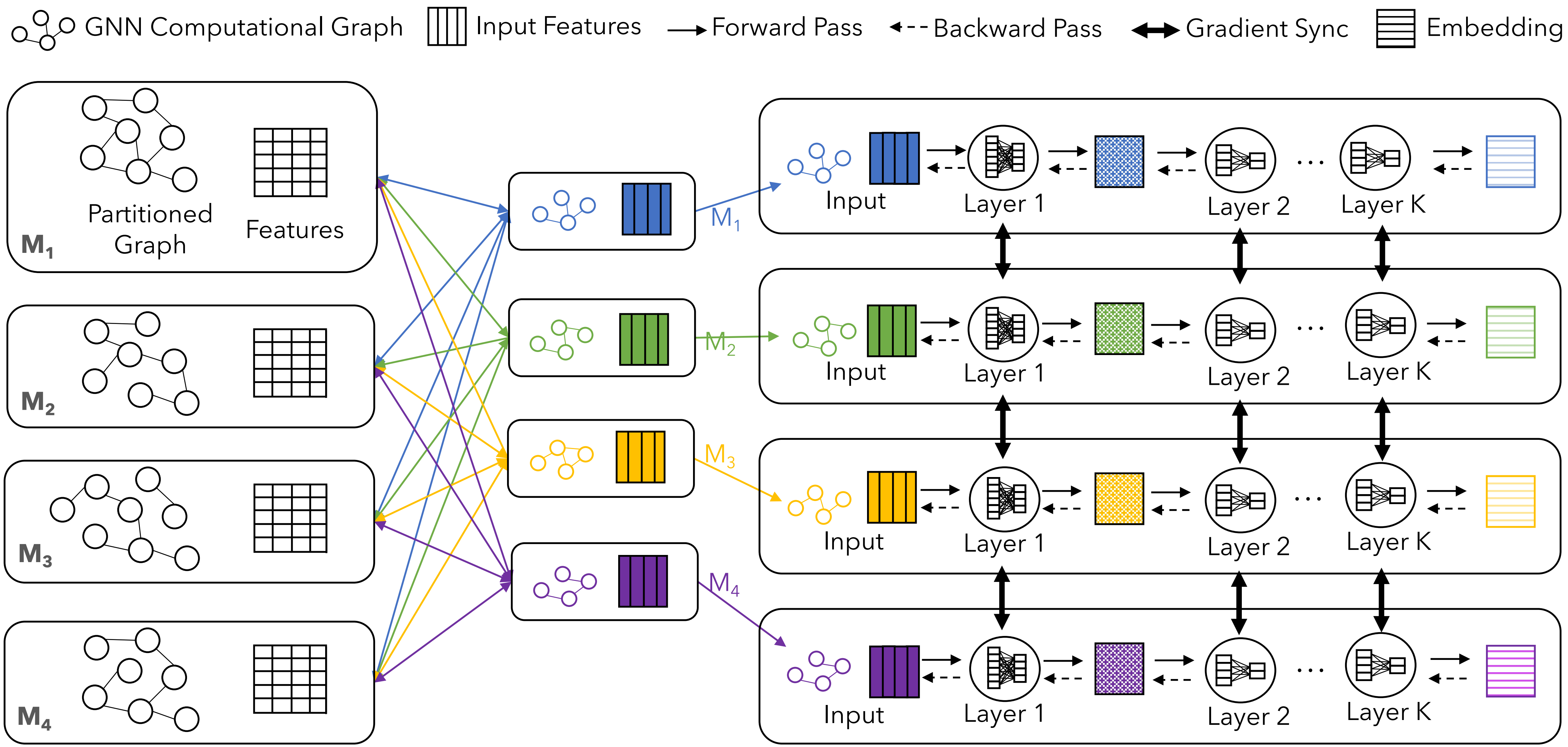




Data-parallel Training

Distributed Neural Network Processing Techniques

Distributed Graph Neural Networks



Network overhead **dominates** epoch time, rendering **GPUs underutilized ~80%** of the time

Partitioning is **ineffective**, and in many cases **counterproductive**

P^3 proposes **push-pull parallelism**, a new technique for distributed GNN training that effectively ***eliminates*** these overheads

P^3: Pipelined Push Pull

Feature movements cause dominant network traffic

Graph structure can be **compactly** represented

Reduce data communicated by avoiding feature movement

Existing systems consider graph & features **indivisible**

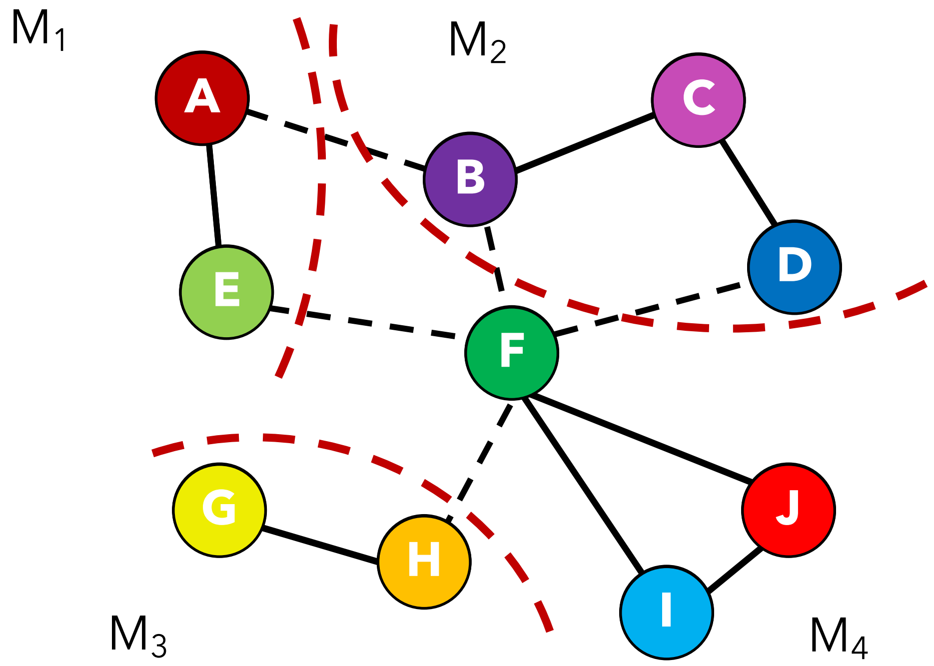
Partition them as **single-entity**

Independent Partitioning of Graph & Features

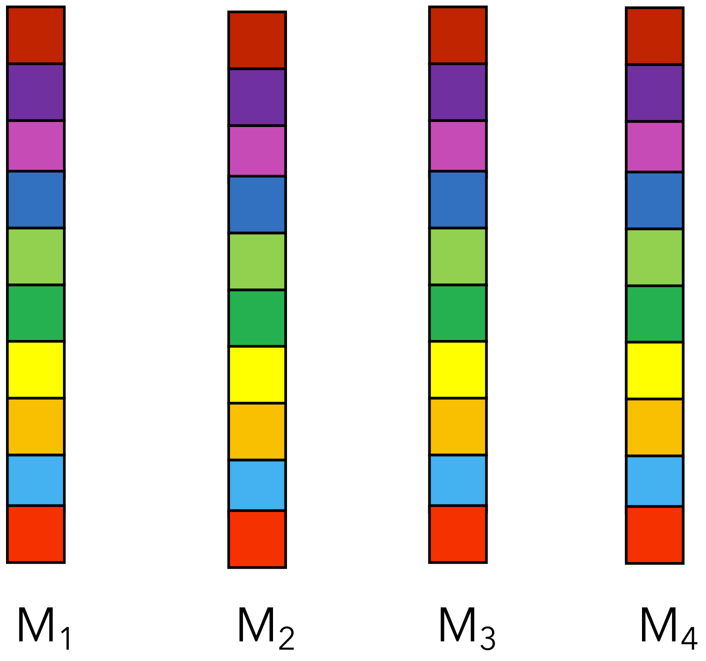
Independent Partitioning

Graph Structure: Partitioned using **random hash func**

Features: Partitioned along **feature dimension**



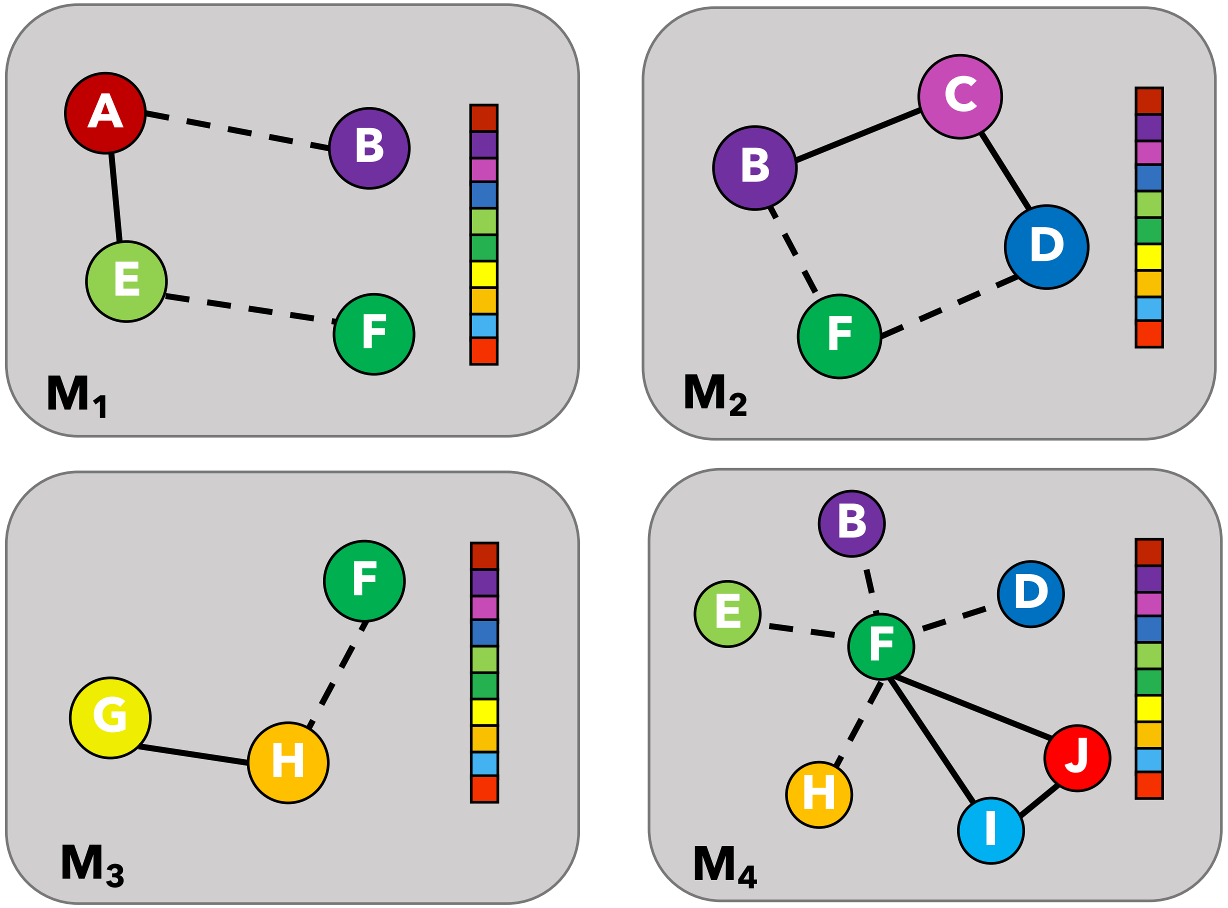
Graph Structure



Features

Independent Partitioning

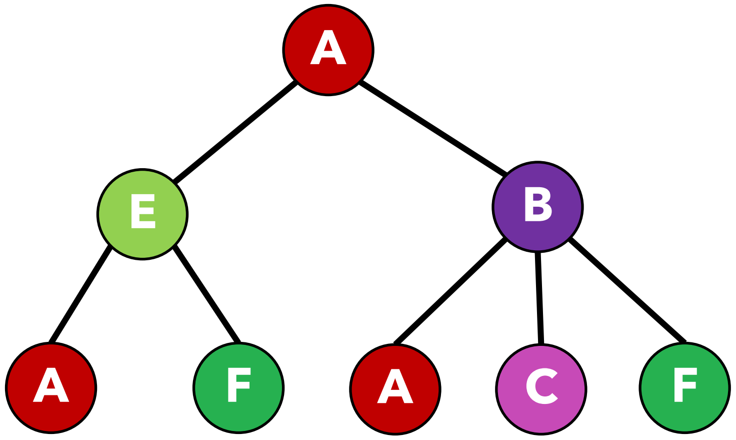
Graph Structure: Partitioned using **random hash func**
Features: Partitioned along **feature dimension**



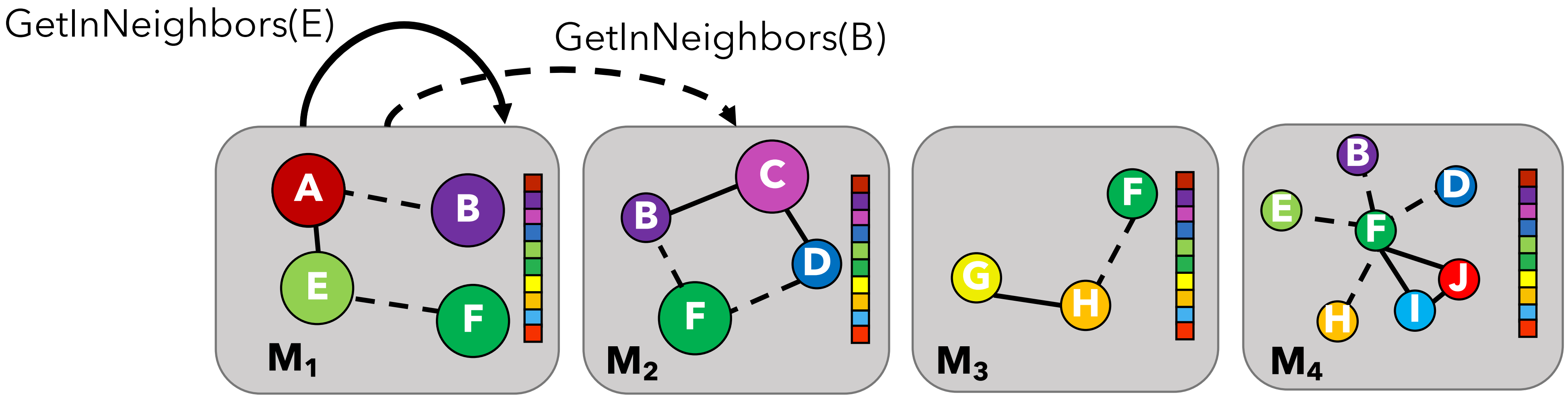
Computation Graph Creation

→ Local ⋯→ Remote

Structure moved over network

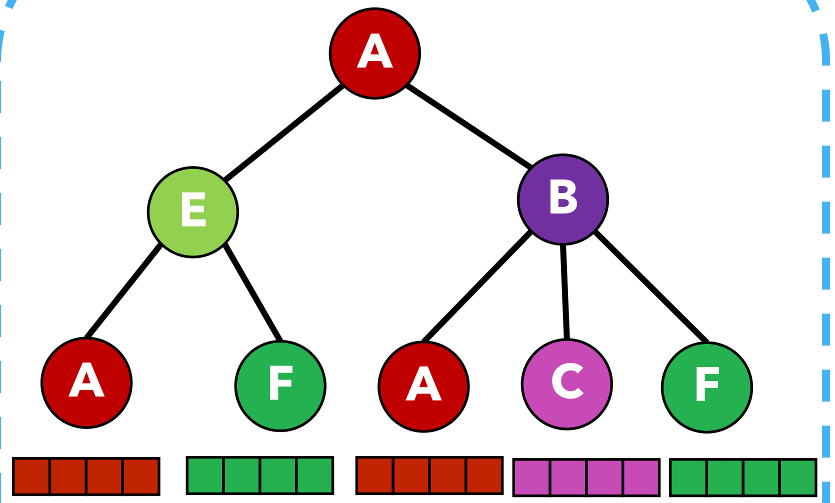


Features not moved over network



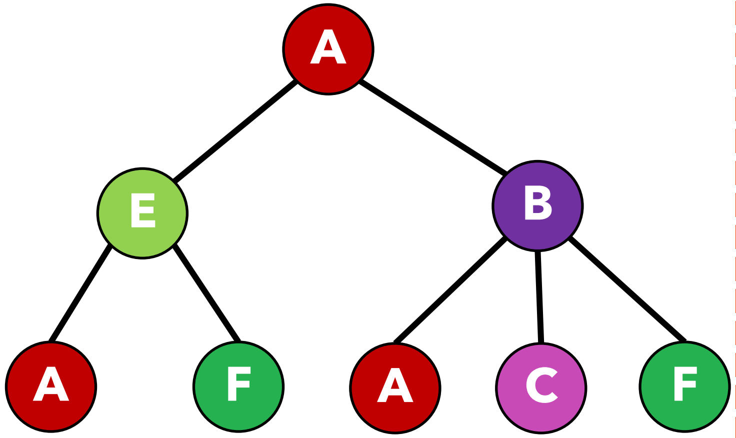
Results in **significant reduction** in data communication

Hybrid Parallelism



Existing Systems

Use Data Parallelism

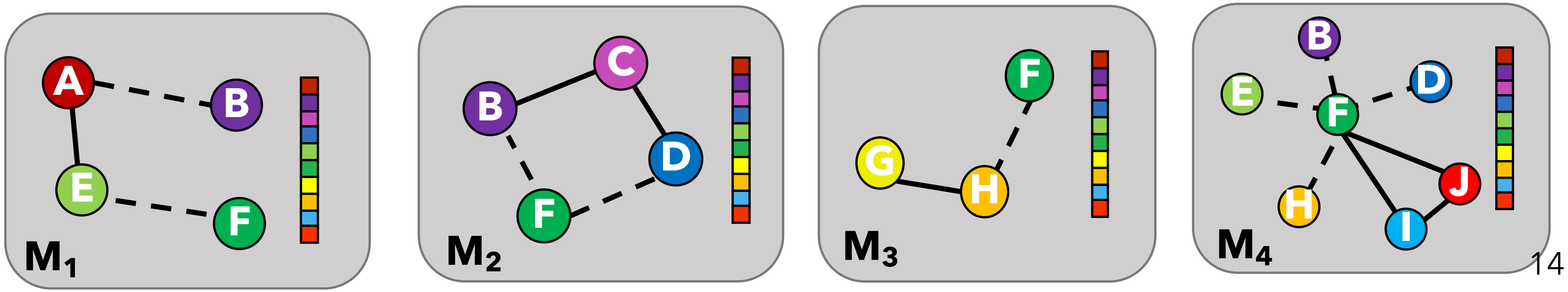


P^3

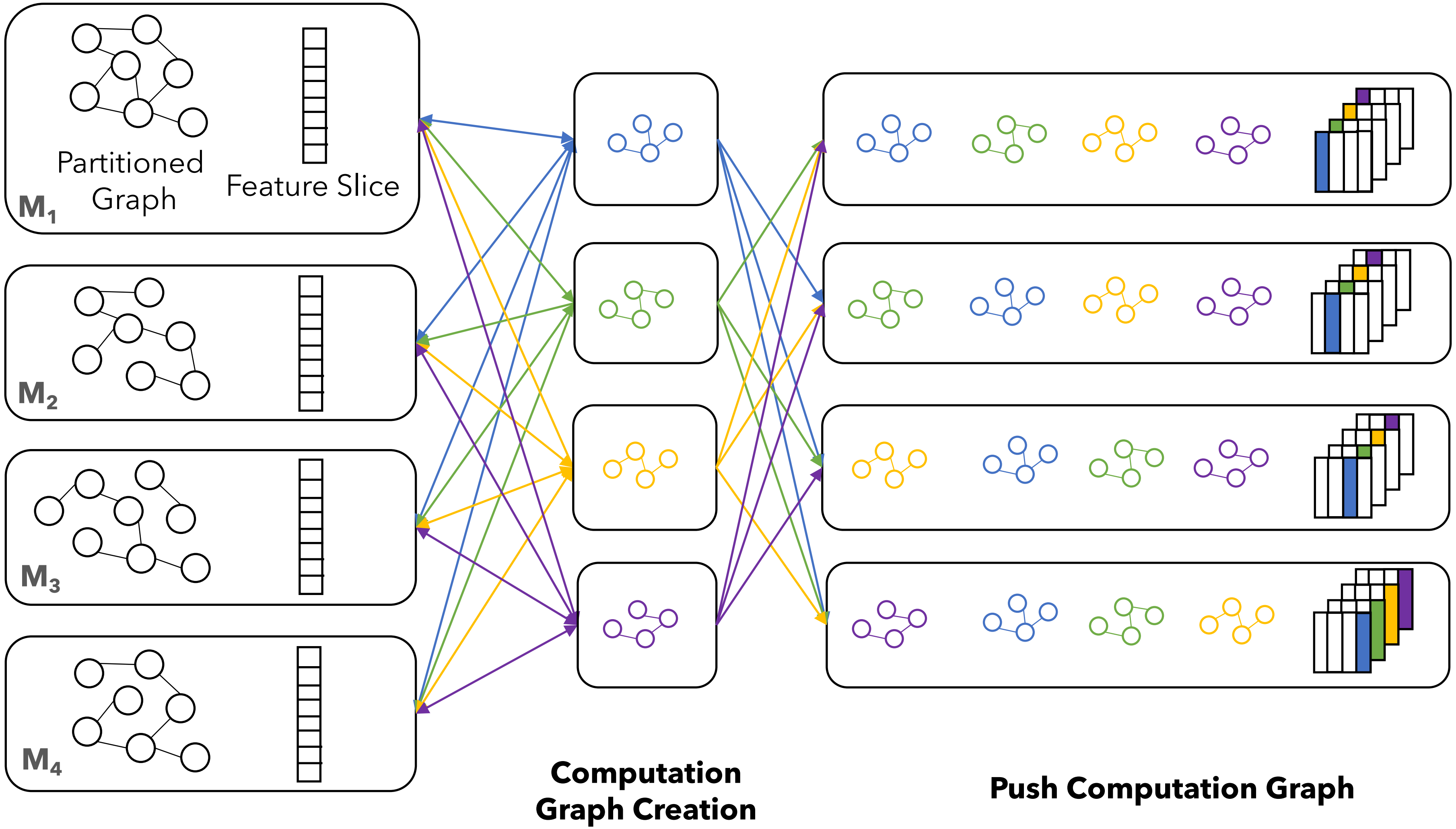
Cannot use data parallelism

Model parallelism incurs overheads

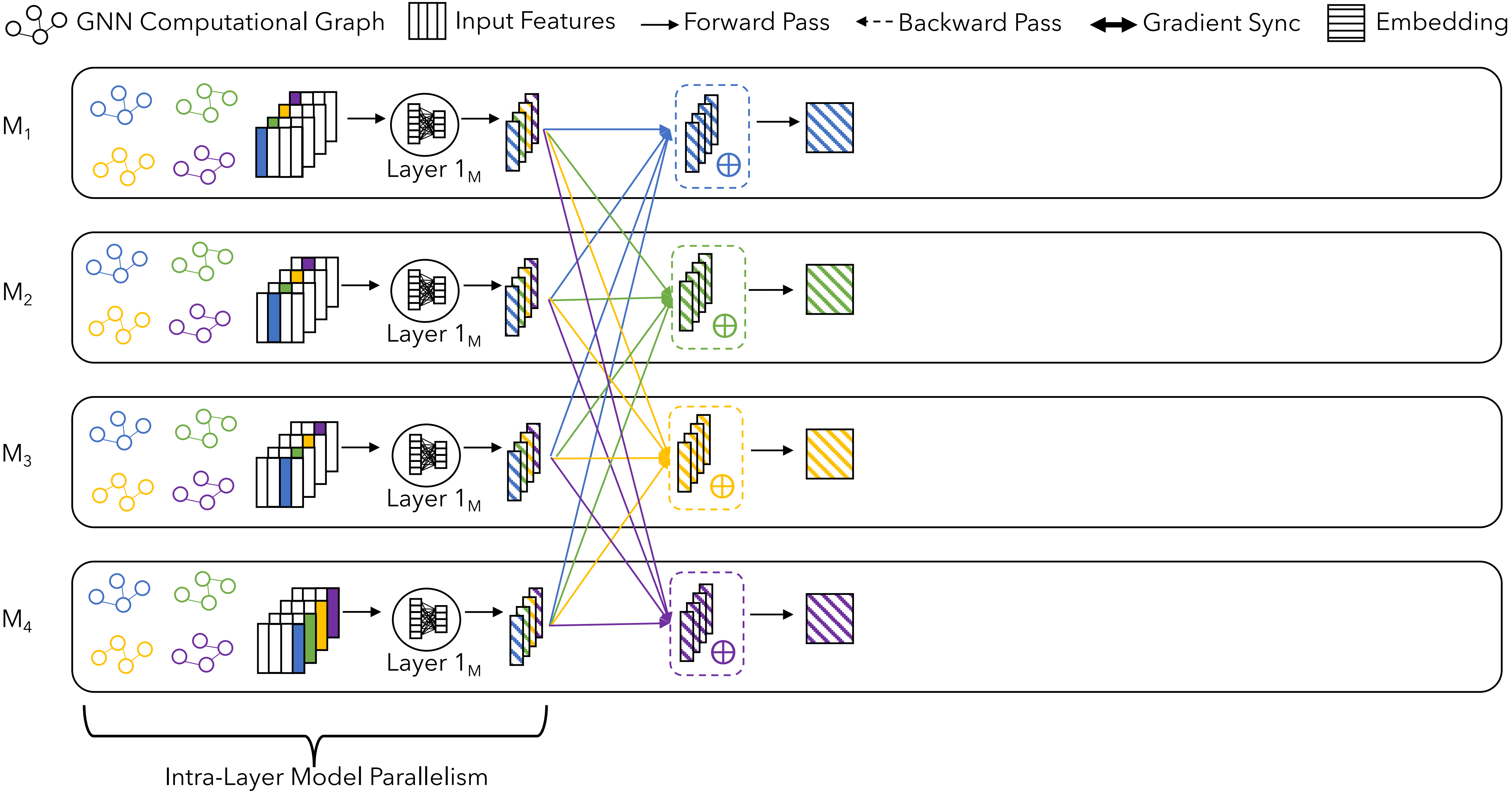
P^3 combines data and model parallelism



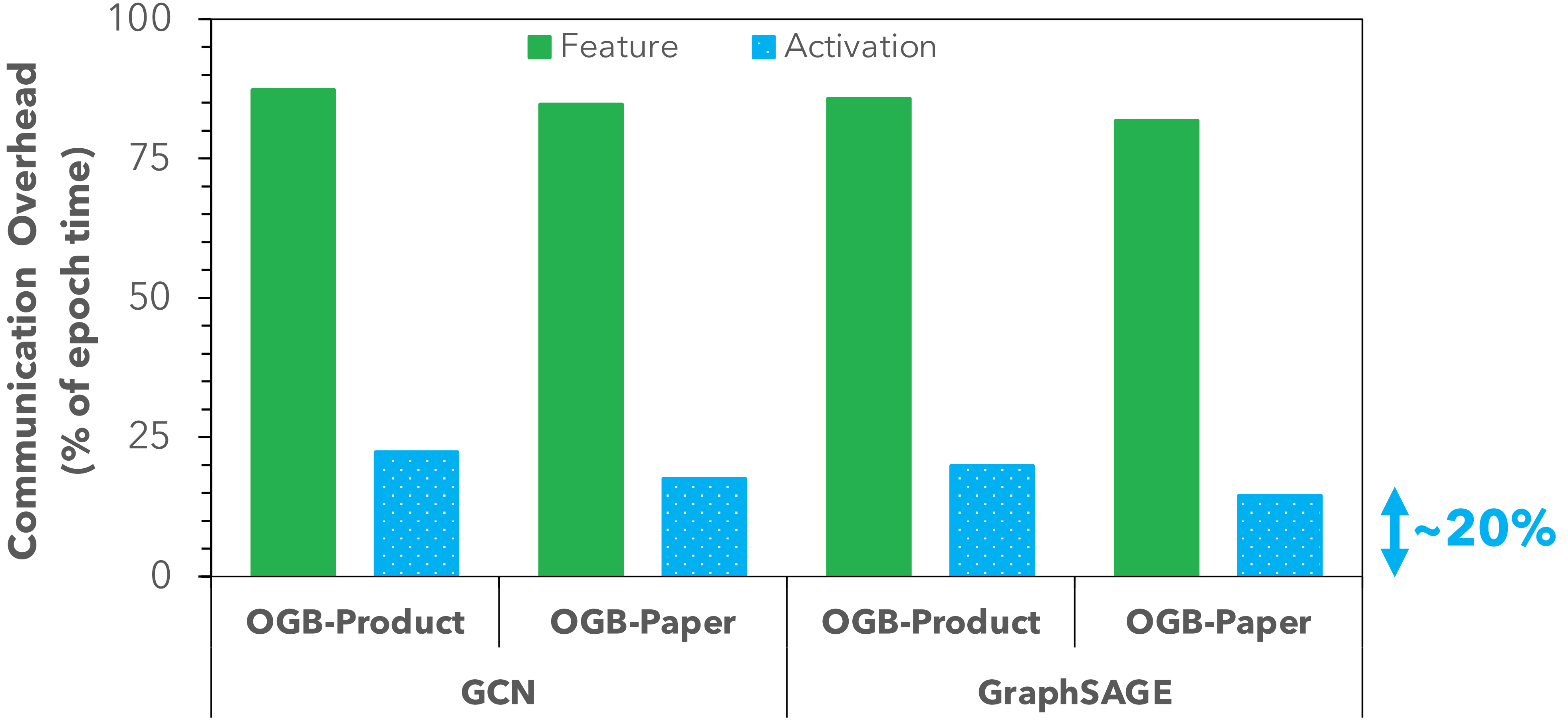
P^3: **Distributed** Graph Neural Networks



P^3: **Distributed** Graph Neural Networks

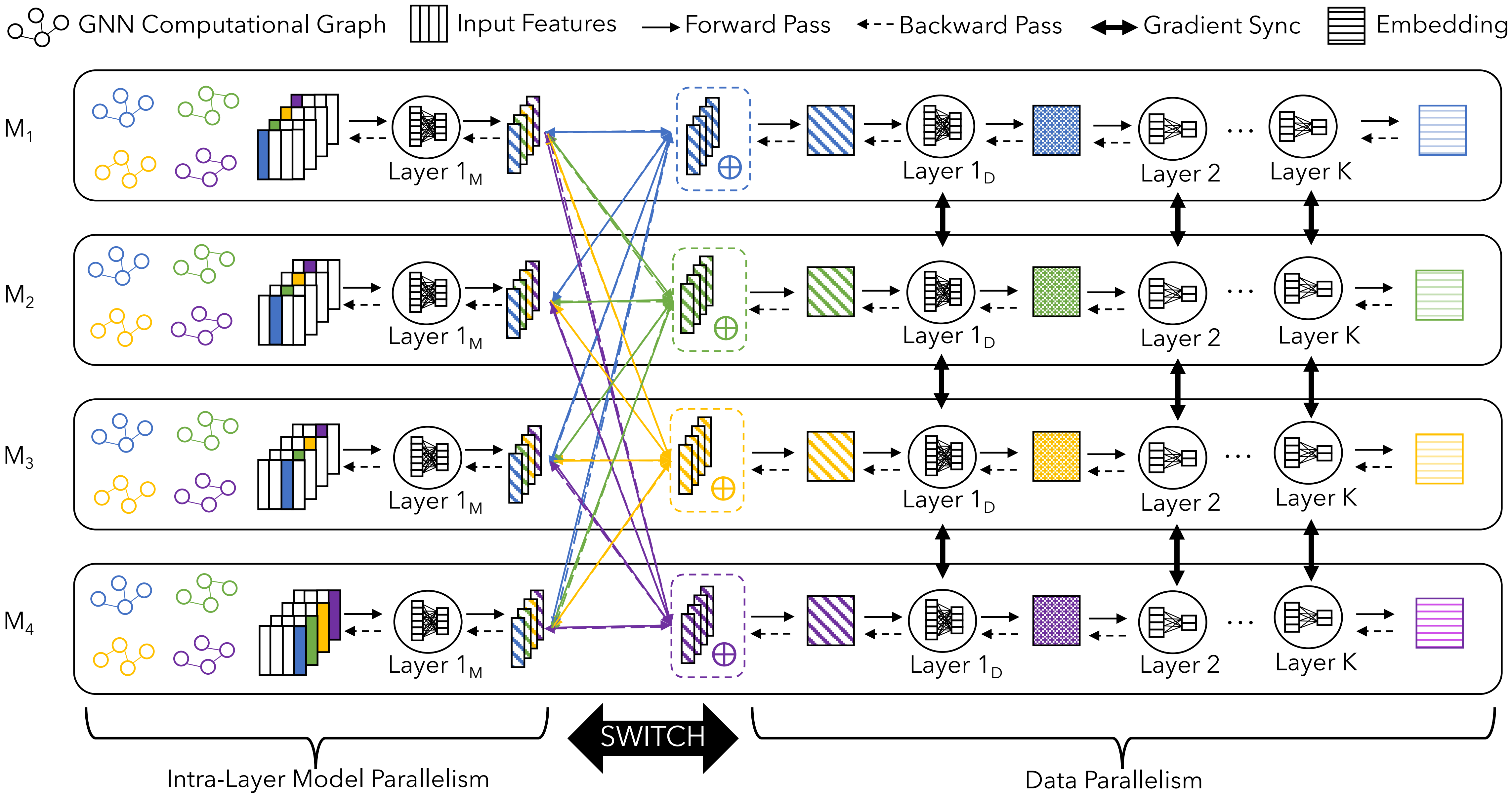


GNNs typically use **small** hidden dimensions → **small** intermediate activations



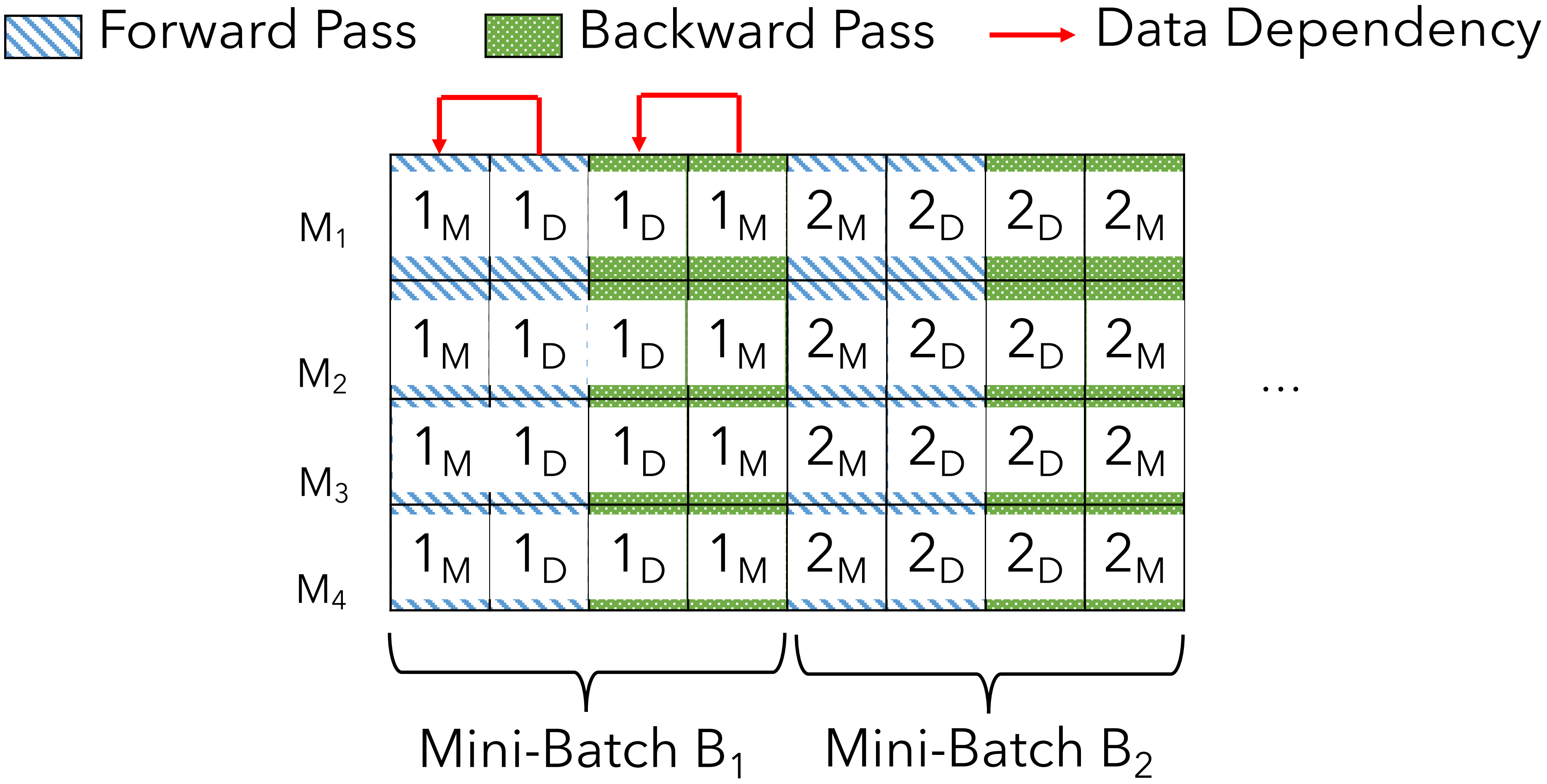
Enables **faster** training across range of GNN models and datasets

P^3: **Distributed** Graph Neural Networks



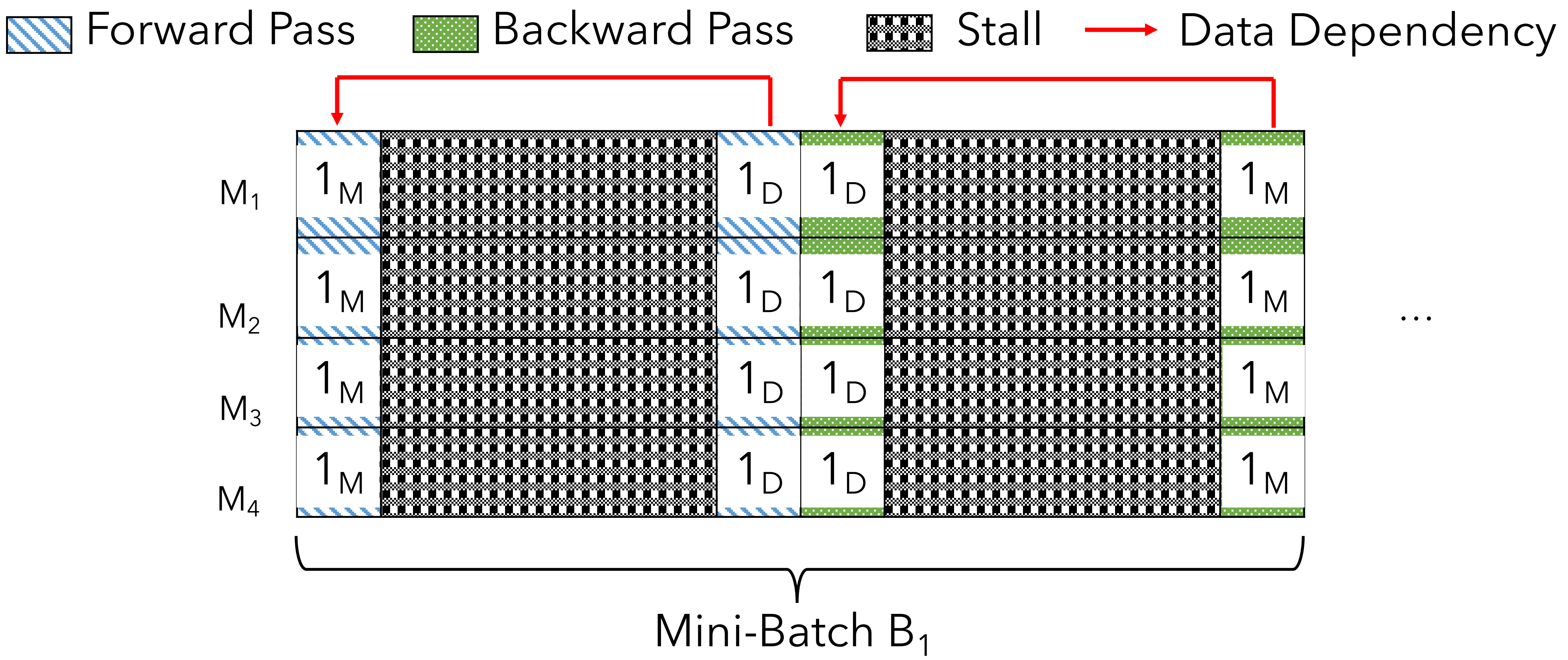
P^3: GNN **Training**

Hybrid Parallelism requires **communication** in **forward** and **backward** pass



P^3: GNN **Training**

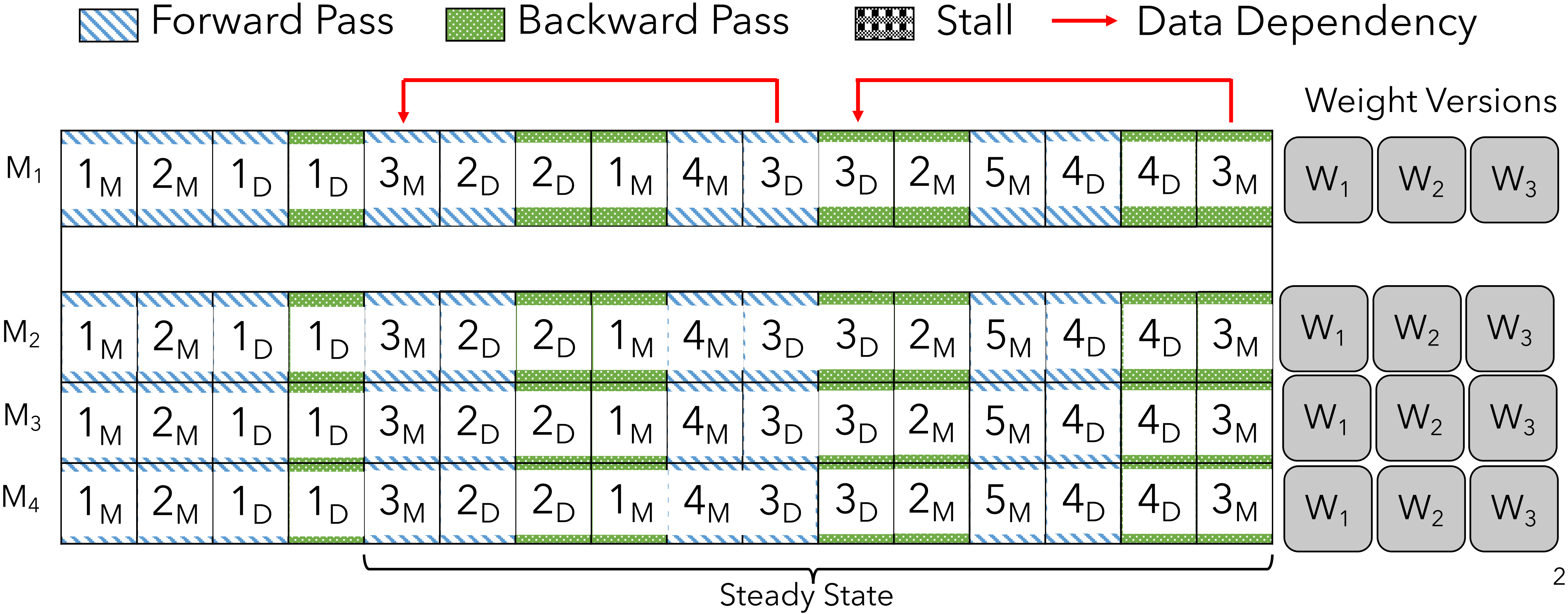
Communication results in **computation stall**



P^3: **Pipelining**

Overlaps **computation** with **communication**

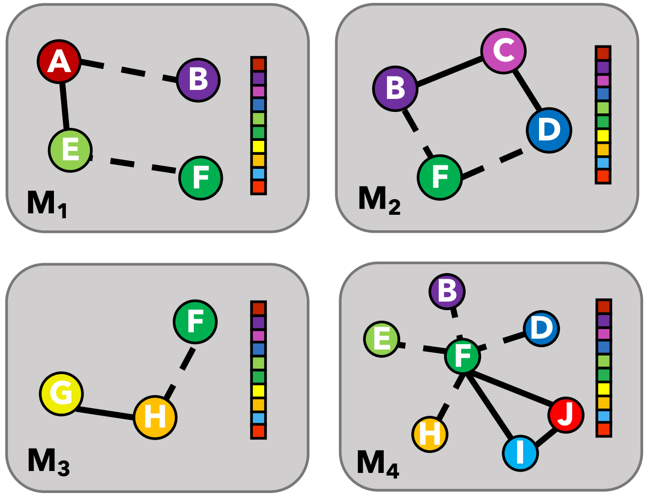
Improves performance by up to **50%**



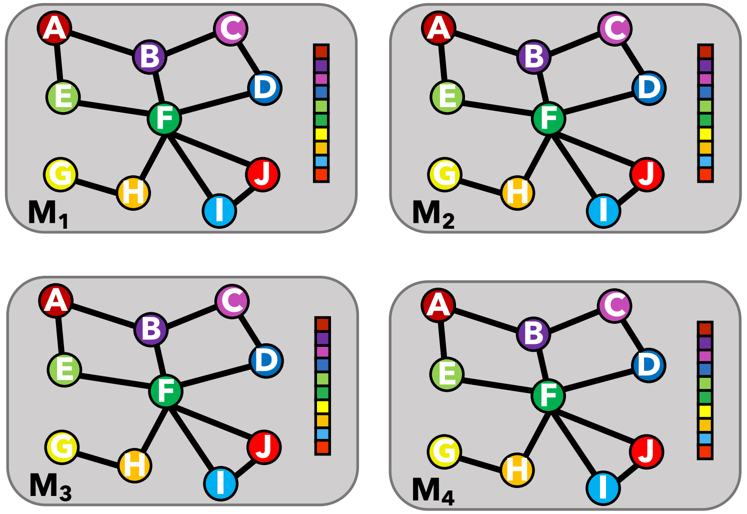
P3: **Caching**

Cache **graph structure** and/or **features**

Improves performance by up to **1.7x**



Structure : Partitioned
Feature : Partitioned

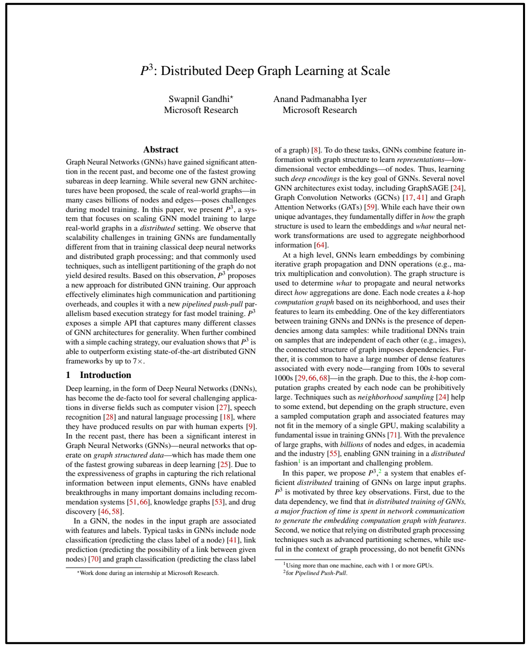


Structure : Cached
Feature : Partitioned

P^3: **API**

P^3 exposes a **simple API** for developers

`partition()`	Independently partition graph and features, and cache if possible
`scatter()`	Generate message vector
`gather()`	Aggregate message vector
`transform()`	Compute partial activation
`sync()`	Accumulate partial activation
`apply()`	Compute output activation



P^3: Distributed Deep Graph Learning at Scale

Swapnil Gandhi* — Microsoft Research
Anand Padmanabha Iyer — Microsoft Research

Abstract

Graph Neural Networks (GNNs) have gained significant attention in the recent past, and become one of the fastest growing subareas in deep learning. While several new GNN architectures have been proposed, the scale of real-world graphs—in many cases billions of nodes and edges—poses challenges during model training. In this paper, we present P^3, a system that focuses on scaling GNN model training to large real-world graphs in a *distributed* setting. We observe that scalability challenges in training GNNs are fundamentally different from that in training classical deep neural networks and distributed graph processing; and that commonly used techniques, such as intelligent partitioning of the graph do not yield desired results. Based on this observation, P^3 proposes a new approach for distributed GNN training. Our approach effectively eliminates high communication and partitioning overheads, and couples it with a new *pipelined push-pull* parallelism based execution strategy for fast model training. P^3 exposes a simple API that captures many different classes of GNN architectures for generality. When further combined with a simple caching strategy, our evaluation shows that P^3 is able to outperform existing state-of-the-art distributed GNN frameworks by up to 7×.

1 Introduction

Deep learning, in the form of Deep Neural Networks (DNNs), has become the de-facto tool for several challenging applications in diverse fields such as computer vision [27], speech recognition [28] and natural language processing [18], where they have produced results on par with human experts [9]. In the recent past, there has been a significant interest in Graph Neural Networks (GNNs)—neural networks that operate on *graph structured data*—which has made them one of the fastest growing subareas in deep learning [25]. Due to the expressiveness of graphs in capturing the rich relational information between input elements, GNNs have enabled breakthroughs in many important domains including recommendation systems [51, 66], knowledge graphs [53], and drug discovery [46, 58].

In a GNN, the nodes in the input graph are associated with features and labels. Typical tasks in GNNs include node classification (predicting the class label of a node) [41], link prediction (predicting the possibility of a link between given nodes) [70] and graph classification (predicting the class label of a graph) [8]. To do these tasks, GNNs combine feature information with graph structure to learn *representations*—low-dimensional vector embeddings—of nodes. Thus, learning such *deep encodings* is the key goal of GNNs. Several novel GNN architectures exist today, including GraphSAGE [24], Graph Convolution Networks (GCNs) [17, 41] and Graph Attention Networks (GATs) [59]. While each have their own unique advantages, they fundamentally differ in *how* the graph structure is used to learn the embeddings and *what* neural network transformations are used to aggregate neighborhood information [64].

At a high level, GNNs learn embeddings by combining iterative graph propagation and DNN operations (e.g., matrix multiplication and convolution). The graph structure is used to determine *what* to propagate and neural networks direct *how* aggregations are done. Each node creates a k-hop *computation graph* based on its neighborhood, and uses their features to learn its embedding. One of the key differentiators between training GNNs and DNNs is the presence of dependencies among data samples: while traditional DNNs train on samples that are independent of each other (e.g., images), the connected structure of graph imposes dependencies. Further, it is common to have a large number of dense features associated with every node—ranging from 100s to several 1000s [29, 66, 68]—in the graph. Due to this, the k-hop computation graphs created by each node can be prohibitively large. Techniques such as *neighborhood sampling* [24] help to some extend, but depending on the graph structure, even a sampled computation graph and associated features may not fit in the memory of a single GPU, making scalability a fundamental issue in training GNNs [71]. With the prevalence of large graphs, with *billions* of nodes and edges, in academia and the industry [55], enabling GNN training in a *distributed* fashion[1] is an important and challenging problem.

In this paper, we propose P^3,[2] a system that enables efficient *distributed* training of GNNs on large input graphs. P^3 is motivated by three key observations. First, due to the data dependency, we find that *in distributed training of GNNs, a major fraction of time is spent in network communication to generate the embedding computation graph with features*. Second, we notice that relying on distributed graph processing techniques such as advanced partitioning schemes, while useful in the context of graph processing, do not benefit GNNs

*Work done during an internship at Microsoft Research.

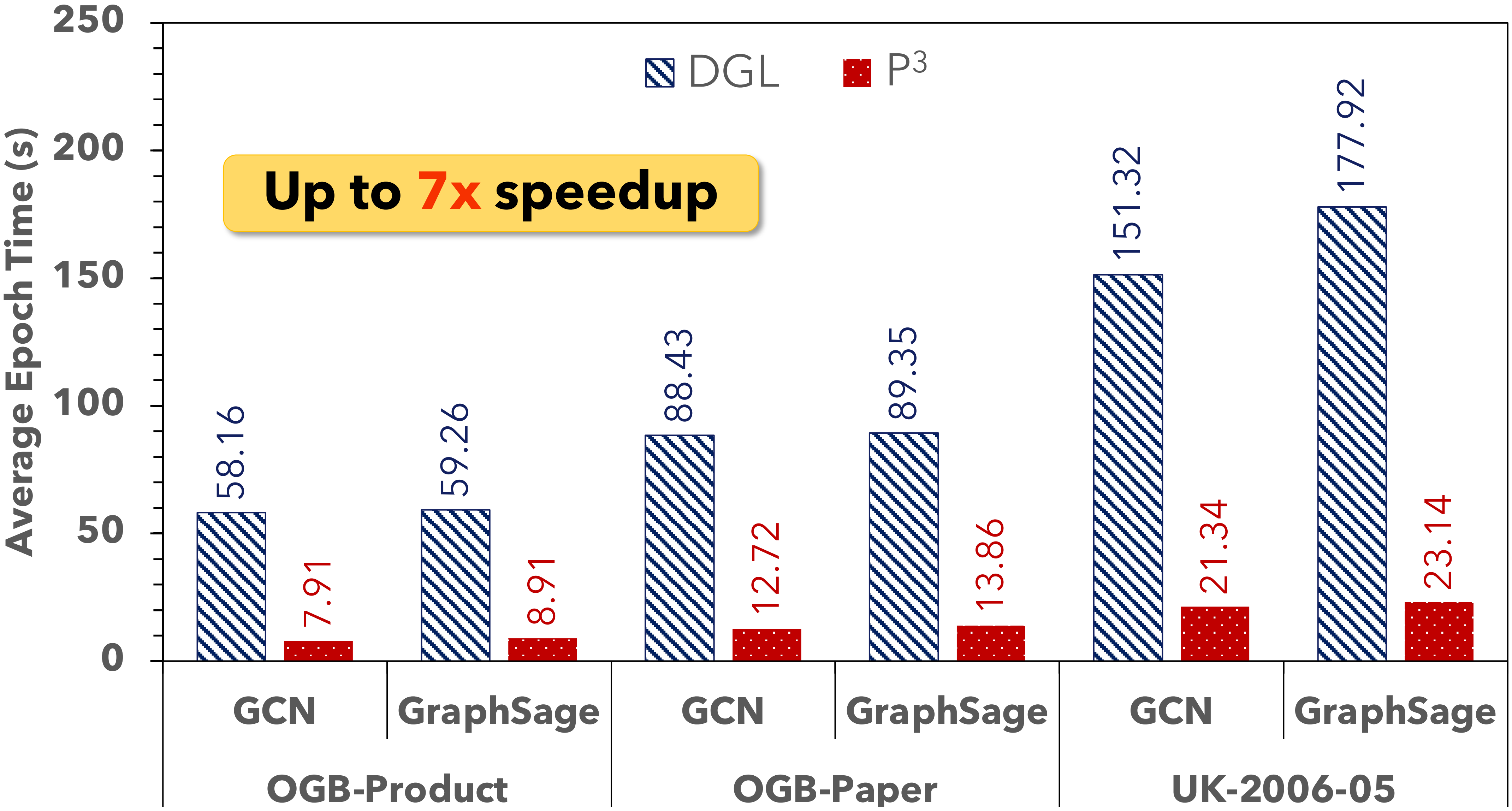
[1] Using more than one machine, each with 1 or more GPUs.

[2] for *Pipelined Push-Pull*.

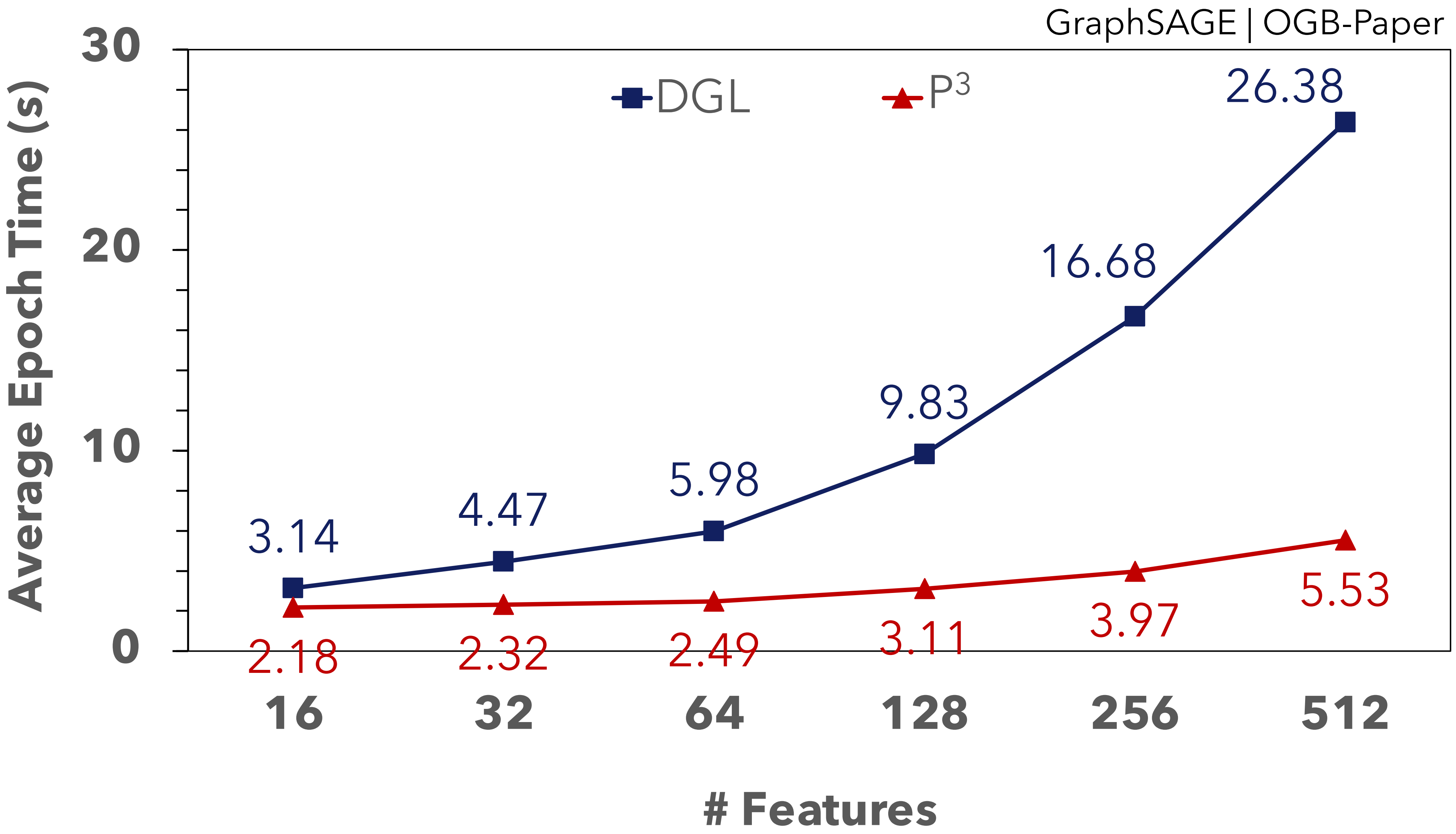
Implementation & **Evaluation**

- Implemented on Deep Graph Library (DGL) v0.5
  - Uses PyTorch v1.6
- Evaluated using 16 NVIDIA Tesla P100 GPUs
  - **OGN-Product:** 123M edges, |F|=100
  - **OGN-Paper:** 1.6B edges, |F|=128
  - **UK-2006-05**: 2.9B edges, |F|=256
- Several GNN architectures
  - GCN [NeurIPS ‘16]
  - GraphSAGE [NeurIPS ‘17]

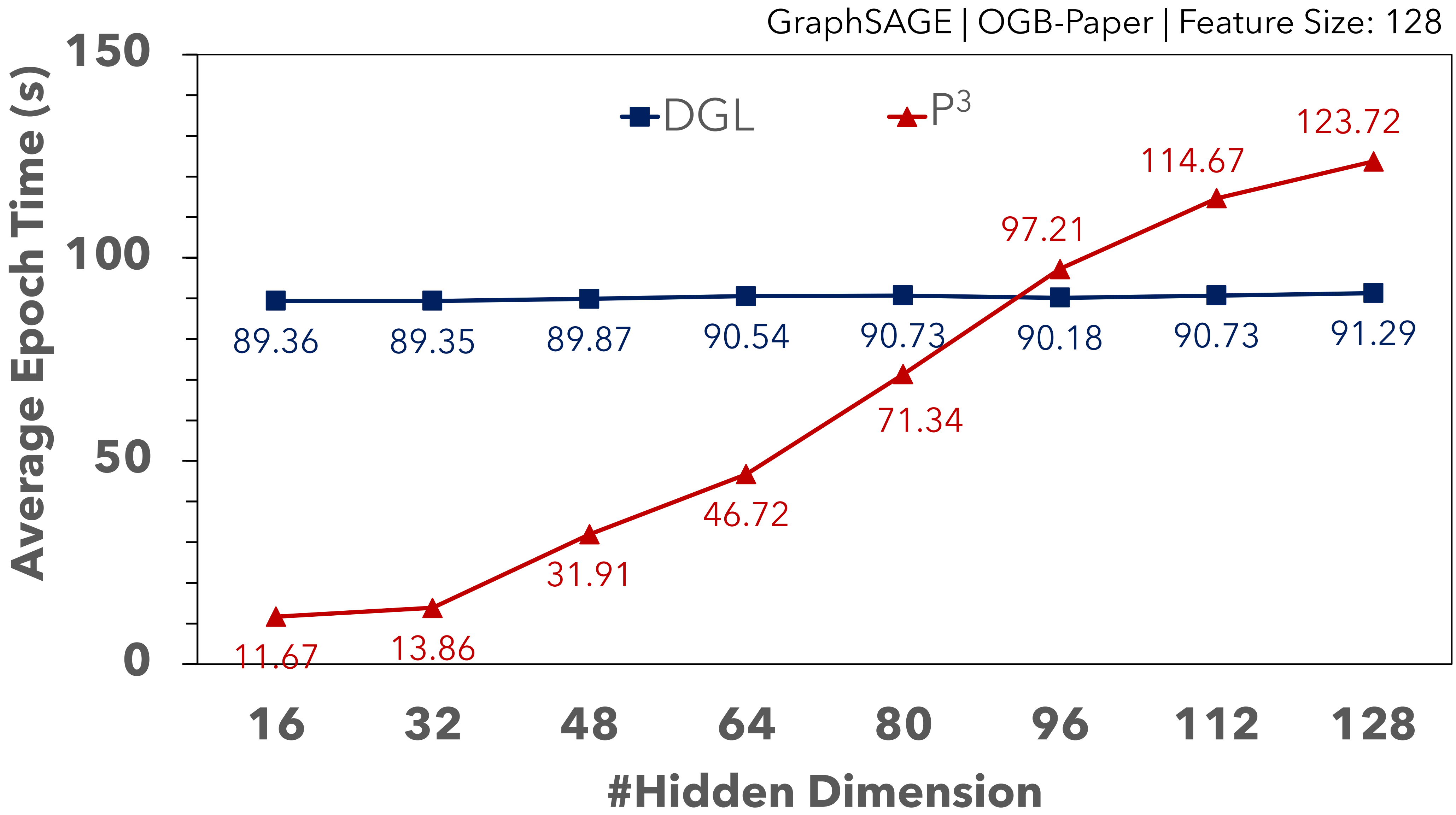
P^3 Performance



P³ Scaling

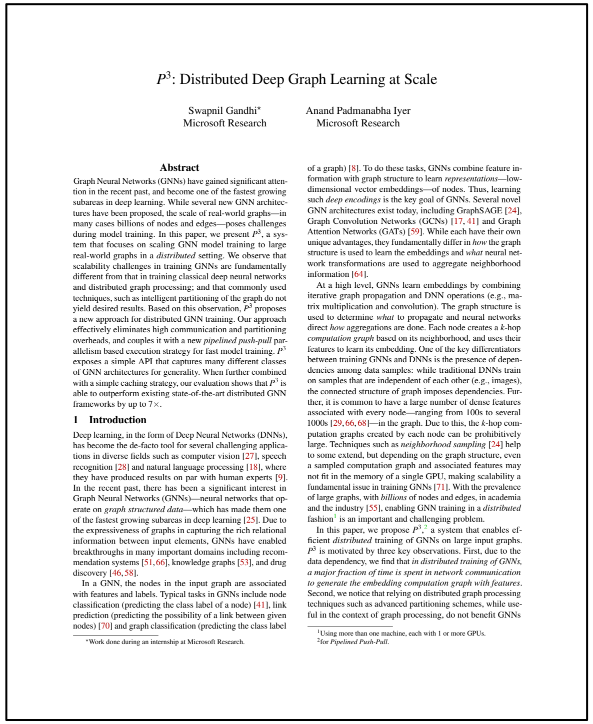

GraphSAGE | OGB-Paper
DGL
P³
Average Epoch Time (s)
30
20
10
0
26.38
16.68
9.83
5.98
4.47
3.14
2.18
2.32
2.49
3.11
3.97
5.53
16
32
64
128
256
512
Features

P^3 Shortcomings



More evaluation in the paper

- GNN Models: GAT [ICLR'18], SGCN [ICML'19]
- Larger Datasets: UK-Union (|E|=5.5B), Facebook (|E|=10B)
- Study impact of
  - Sampling
  - Partitioning Strategies
  - Number of Layers
  - Pipelining
  - Caching
- Scaling Characteristics
- Comparison with ROC [MLSys'20]



P^3: Distributed Deep Graph Learning at Scale

Swapnil Gandhi* — Microsoft Research
Anand Padmanabha Iyer — Microsoft Research

Abstract

Graph Neural Networks (GNNs) have gained significant attention in the recent past, and become one of the fastest growing subareas in deep learning. While several new GNN architectures have been proposed, the scale of real-world graphs—in many cases billions of nodes and edges—poses challenges during model training. In this paper, we present P^3, a system that focuses on scaling GNN model training to large real-world graphs in a *distributed* setting. We observe that scalability challenges in training GNNs are fundamentally different from that in training classical deep neural networks and distributed graph processing; and that commonly used techniques, such as intelligent partitioning of the graph do not yield desired results. Based on this observation, P^3 proposes a new approach for distributed GNN training. Our approach effectively eliminates high communication and partitioning overheads, and couples it with a new *pipelined push-pull* parallelism based execution strategy for fast model training. P^3 exposes a simple API that captures many different classes of GNN architectures for generality. When further combined with a simple caching strategy, our evaluation shows that P^3 is able to outperform existing state-of-the-art distributed GNN frameworks by up to 7×.

1 Introduction

Deep learning, in the form of Deep Neural Networks (DNNs), has become the de-facto tool for several challenging applications in diverse fields such as computer vision [27], speech recognition [28] and natural language processing [18], where they have produced results on par with human experts [9]. In the recent past, there has been a significant interest in Graph Neural Networks (GNNs)—neural networks that operate on *graph structured data*—which has made them one of the fastest growing subareas in deep learning [25]. Due to the expressiveness of graphs in capturing the rich relational information between input elements, GNNs have enabled breakthroughs in many important domains including recommendation systems [51,66], knowledge graphs [53], and drug discovery [46,58].

In a GNN, the nodes in the input graph are associated with features and labels. Typical tasks in GNNs include node classification (predicting the class label of a node) [41], link prediction (predicting the possibility of a link between given nodes) [70] and graph classification (predicting the class label of a graph) [8]. To do these tasks, GNNs combine feature information with graph structure to learn *representations*—low-dimensional vector embeddings—of nodes. Thus, learning such *deep encodings* is the key goal of GNNs. Several novel GNN architectures exist today, including GraphSAGE [24], Graph Convolution Networks (GCNs) [17, 41] and Graph Attention Networks (GATs) [59]. While each have their own unique advantages, they fundamentally differ in *how* the graph structure is used to learn the embeddings and *what* neural network transformations are used to aggregate neighborhood information [64].

At a high level, GNNs learn embeddings by combining iterative graph propagation and DNN operations (e.g., matrix multiplication and convolution). The graph structure is used to determine *what* to propagate and neural networks direct *how* aggregations are done. Each node creates a k-hop *computation graph* based on its neighborhood, and uses their features to learn its embedding. One of the key differentiators between training GNNs and DNNs is the presence of dependencies among data samples: while traditional DNNs train on samples that are independent of each other (e.g., images), the connected structure of graph imposes dependencies. Further, it is common to have a large number of dense features associated with every node—ranging from 100s to several 1000s [29,66,68]—in the graph. Due to this, the k-hop computation graphs created by each node can be prohibitively large. Techniques such as *neighborhood sampling* [24] help to some extend, but depending on the graph structure, even a sampled computation graph and associated features may not fit in the memory of a single GPU, making scalability a fundamental issue in training GNNs [71]. With the prevalence of large graphs, with *billions* of nodes and edges, in academia and the industry [55], enabling GNN training in a *distributed* fashion[1] is an important and challenging problem.

In this paper, we propose P^3,[2] a system that enables efficient *distributed* training of GNNs on large input graphs. P^3 is motivated by three key observations. First, due to the data dependency, we find that *in distributed training of GNNs, a major fraction of time is spent in network communication to generate the embedding computation graph with features*. Second, we notice that relying on distributed graph processing techniques such as advanced partitioning schemes, while useful in the context of graph processing, do not benefit GNNs

*Work done during an internship at Microsoft Research.

[1] Using more than one machine, each with 1 or more GPUs.
[2] for *Pipelined Push-Pull*.

Takeaway

- Distributed training of graph neural networks increasingly important
  - Frameworks = Graph Processing + DNN Training
  - Incur high network communication and partitioning overhead
- P^3 eliminates the overheads with distributed GNN training
  - **Independent partitioning** of graph structure and features
  - **Hybrid parallelism** combined with **pipelining** and **caching**
  - Simple **API** for users

Thank you!

https://swapnilgandhi.com