Software-Defined Cloud Systems

Xin Jin

2022.10.14

北京大學
PEKING UNIVERSITY

HotDC 2022

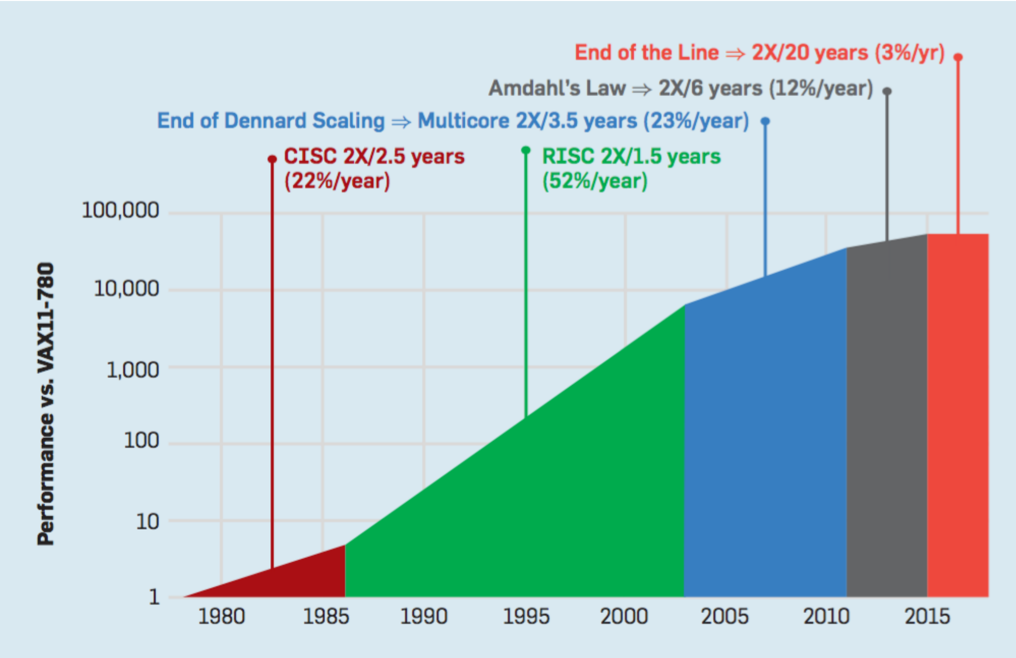
Cloud Systems: Critical Infrastructure of Modern Society

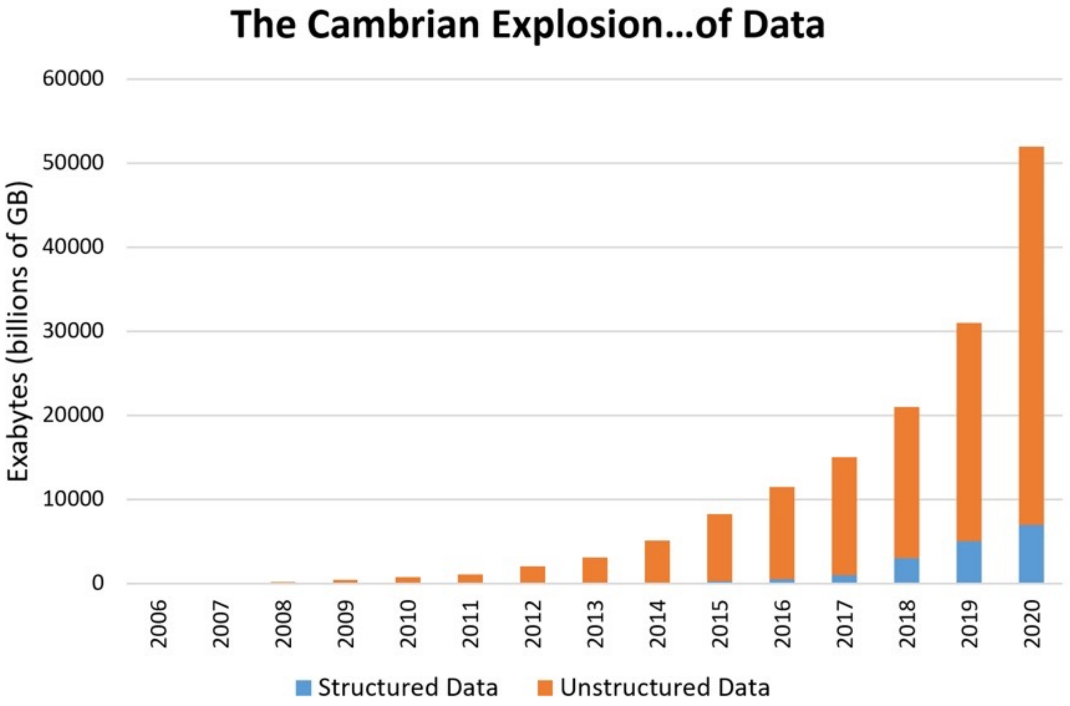
Apps …

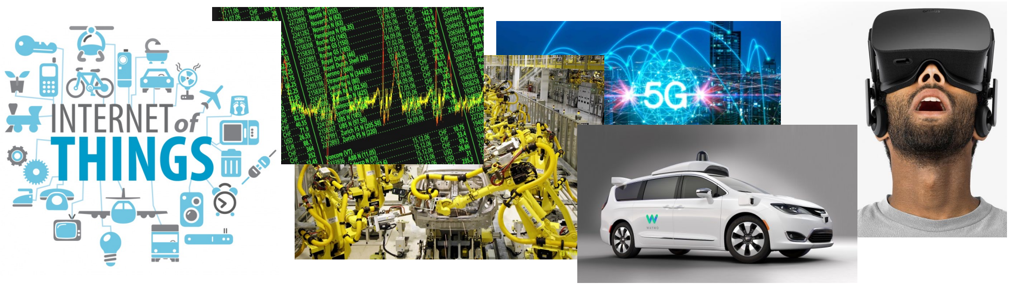
Cloud Systems

End of Moore's Law

Exponentially Growing Demand (more data, apps, services…)





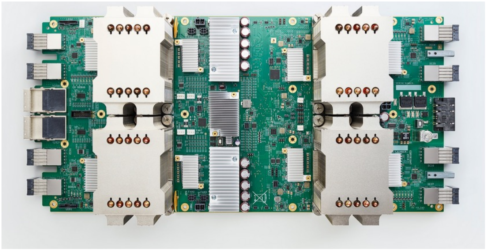


Rise of Domain-Specific Processors

GPU

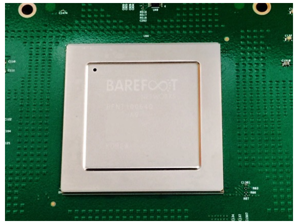
Graphics

TPU



Machine Learning

PISA



Packet Processing

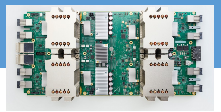
Building Software-Defined Cloud Operating Systems in the Post Moore's Law Era

Apps . . .

Cloud Data Centers

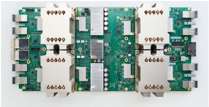
Software-Defined Cloud Systems

Domain-Specific Hardware


Distributed

Traditional OS is CPU-centric for a single node → New challenges are new research opportunities



GPU TPU Tofino Trident DRAM NVM

Domain-Specific Hardware

Our Recent Work on Software-Defined Cloud Systems

Data Plane: Reliability

Meissa: Scalable Network Testing for Programmable Data Planes

SIGCOMM 2022
Amsterdam

Control Plane: Multi-Resource Scheduling

Multi-Resource Interleaving for Deep Learning Training

SIGCOMM 2022
Amsterdam

SIGCOMM 2022
Amsterdam

Meissa: Scalable Network Testing for Programmable Data Planes

Naiqian Zheng, Mengqi Liu, Ennan Zhai, Hongqiang Harry Liu,
Yifan Li, Kaicheng Yang, Xuanzhe Liu, Xin Jin

北京大學 PEKING UNIVERSITY

Alibaba Group

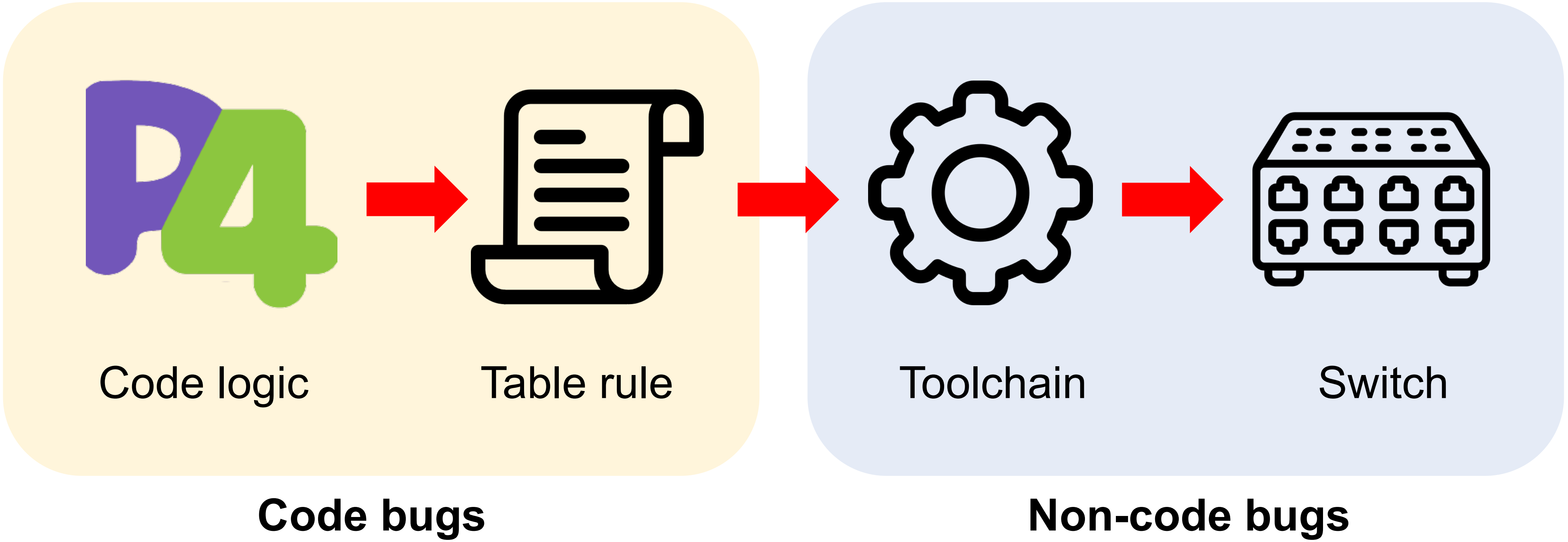
Programmable data planes are buggy

- ✓ Versatility
- ✓ High performance
- ✓ Programmability

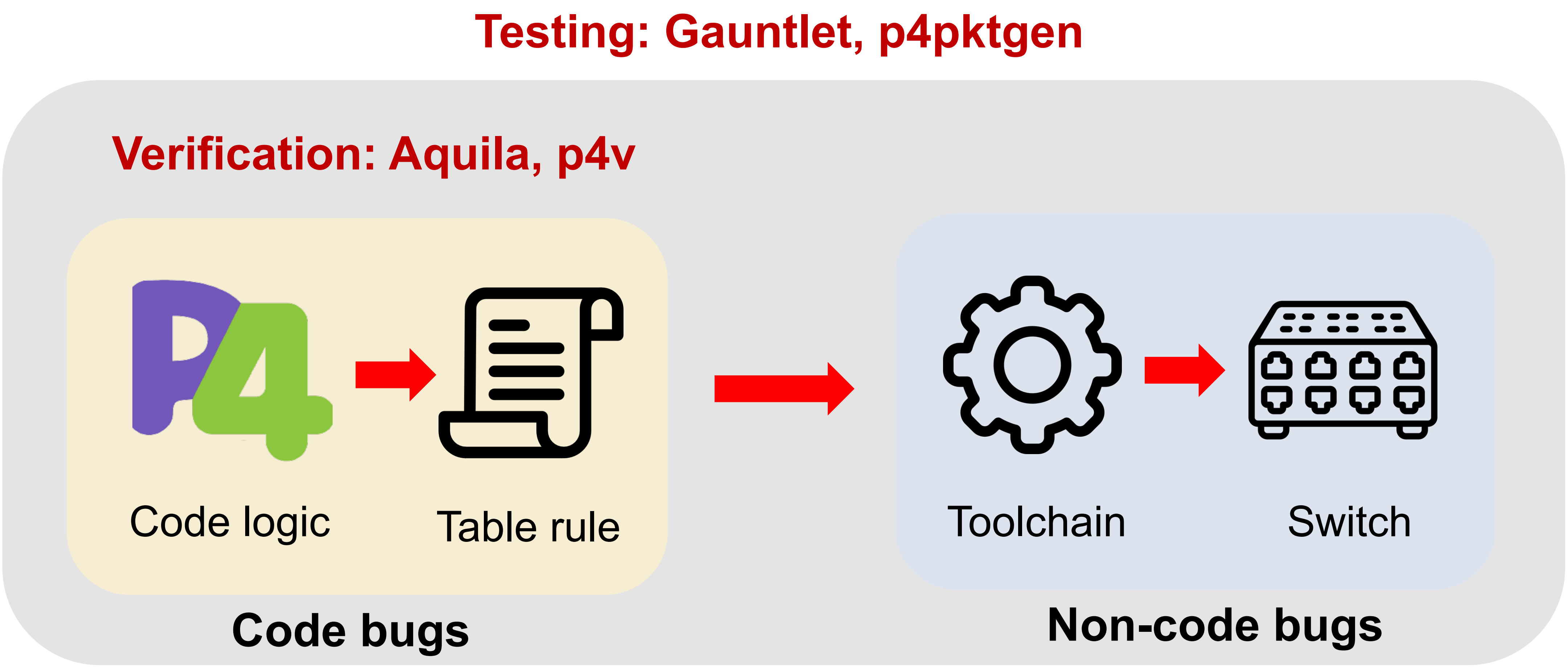
- ! Large control flow
- ! Non-code bugs
- ! Incomprehensive test

✗ Bugs are common with programmable data planes!

Bug taxonomy

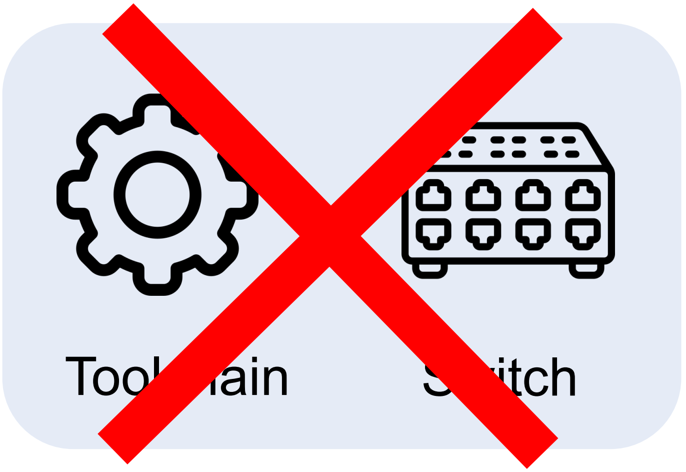


Tools to identify bugs



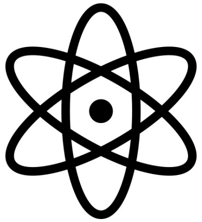
Challenge

Verification: Aquila, p4v



Non-code bugs

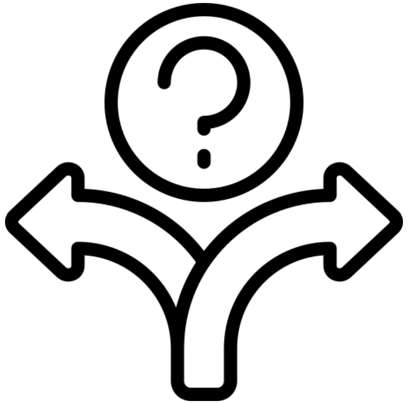
Testing: Gauntlet, p4pktgen



Path Explosion

LOC: $O(10^4)$

# of path: $O(10^{197})$

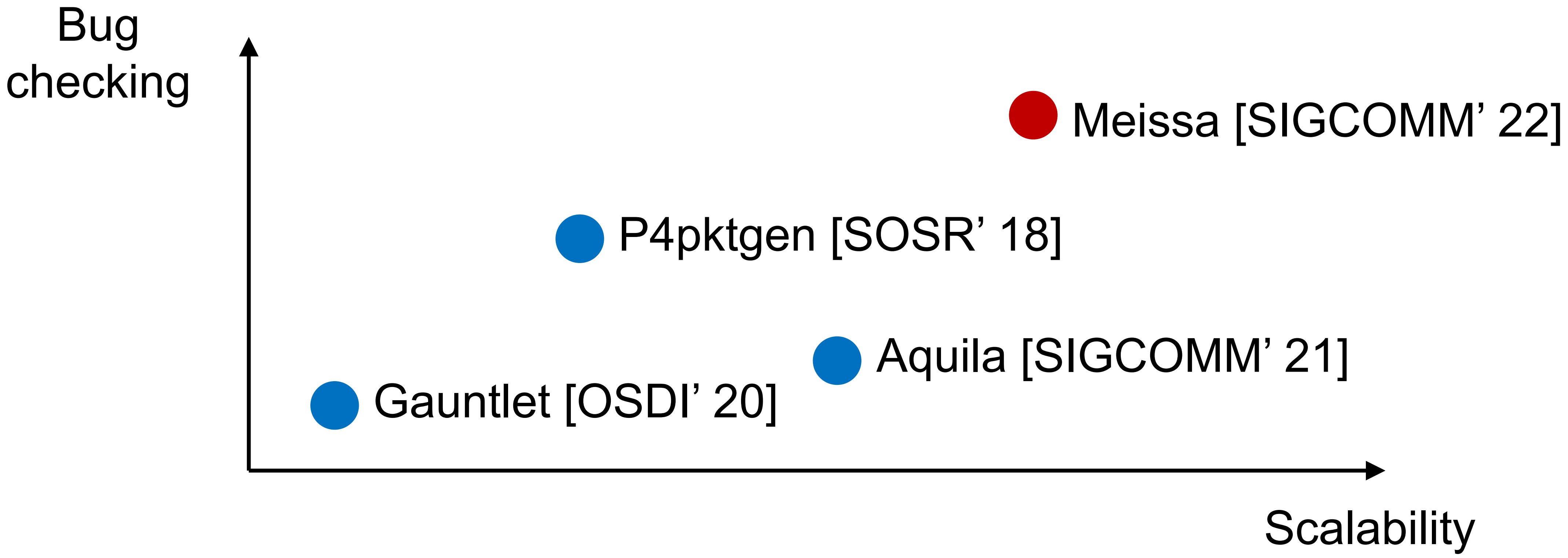


Challenge

Identify both code bugs and non-code bugs

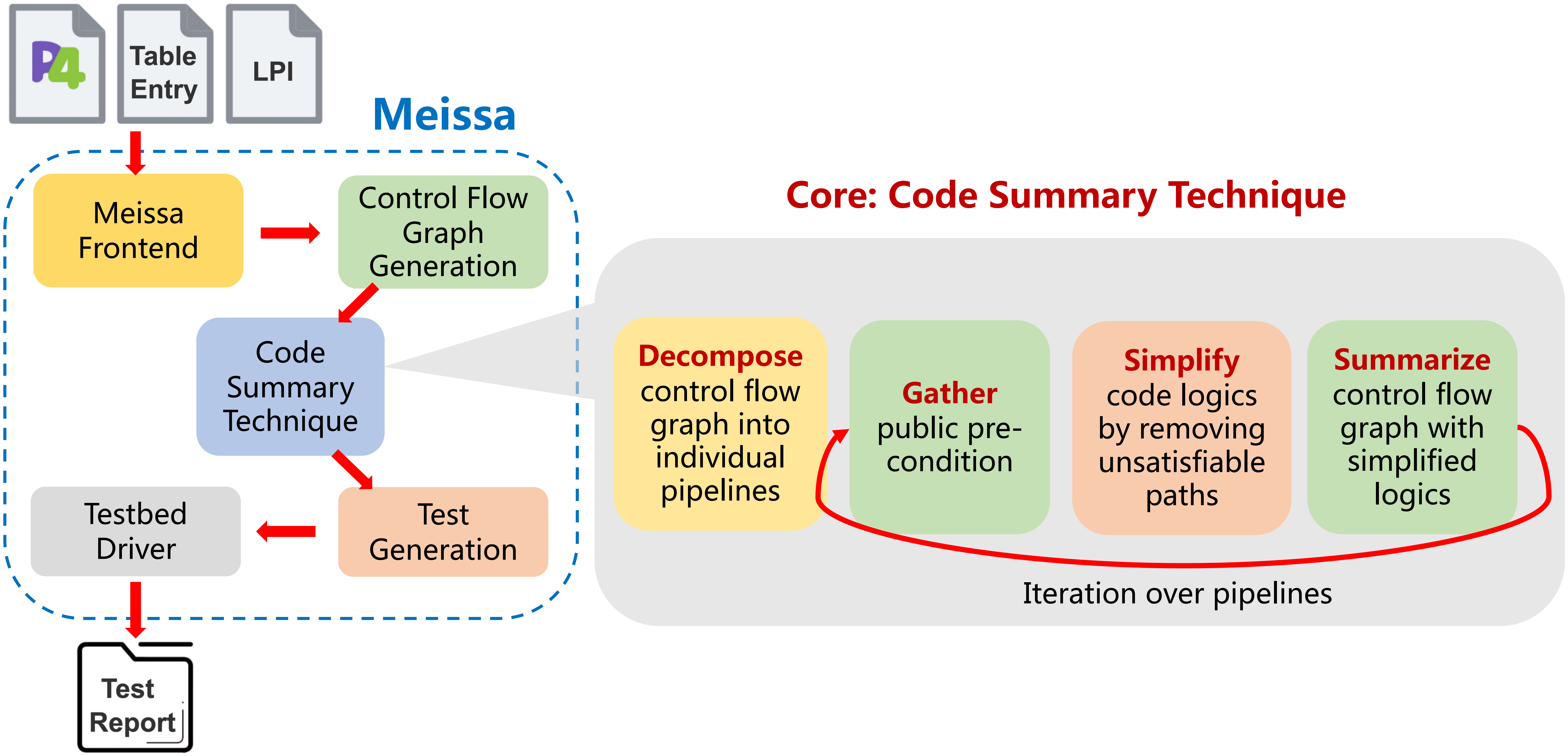
Scale to large large programs

How to detect potential bugs?



Scalable testing with 100% path coverage

Meissa overview



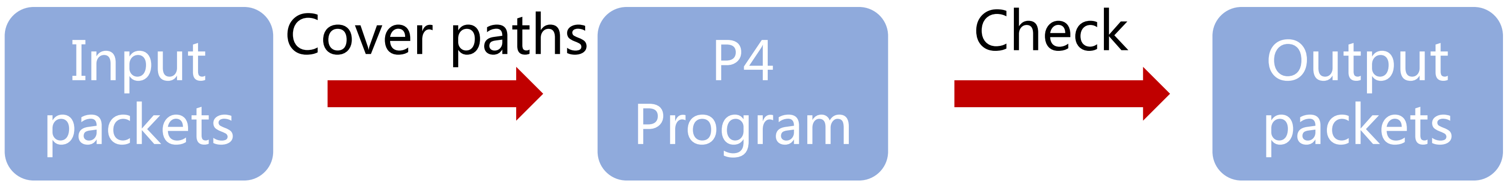
Control flow graph

- The control flow graph(CFG) depicts logics of a P4 program
  - Code
  - Table entry

- The control flow graph consists of two types of nodes
  - Predicate node: judgement, branching
  - Action node: variable assignment

Test generation with symbolic execution

Test generation: input packets which traverse paths in the CFG



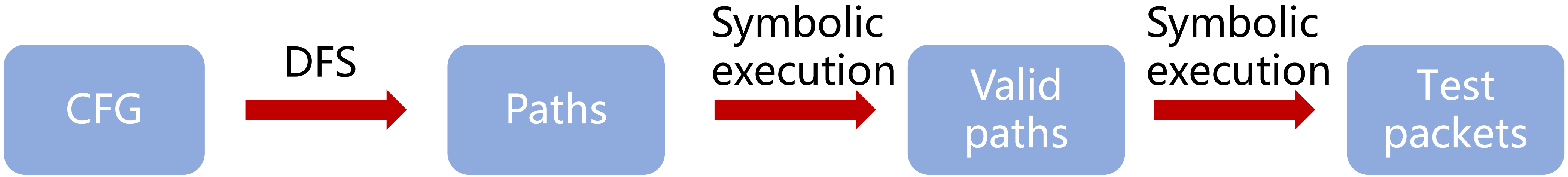
Goal: **100%** path coverage

Goal: get input packets which traverse **all** paths in the CFG

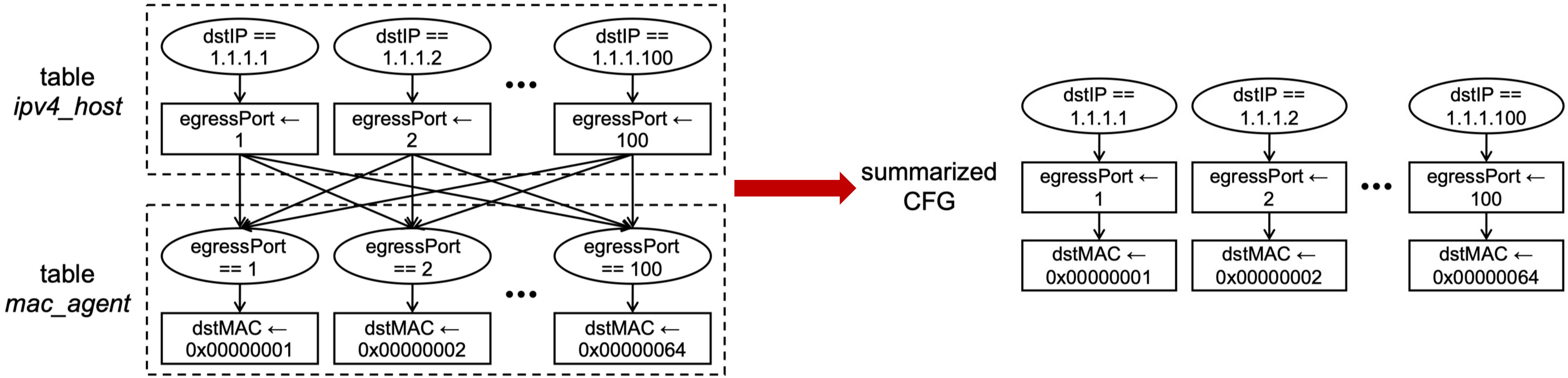
Test generation with symbolic execution

Depth-first search traverses the control flow graph.

Symbolic execution checks the paths' satisfiability.



A lot of redundancy



10000 paths

only **100** valid paths!

Code summary technique

- Code summary eliminates redundancy in advance to speed up DFS.
- Code summary summarizes each pipelines with a succinct representation respectively.
- Code summary gathers succinct summaries of each pipelines into a new simplified CFG.

Summary of an individual pipeline

Techniques:

1. Intra-pipeline redundancy elimination
2. Inter-pipeline public pre-condition filtering

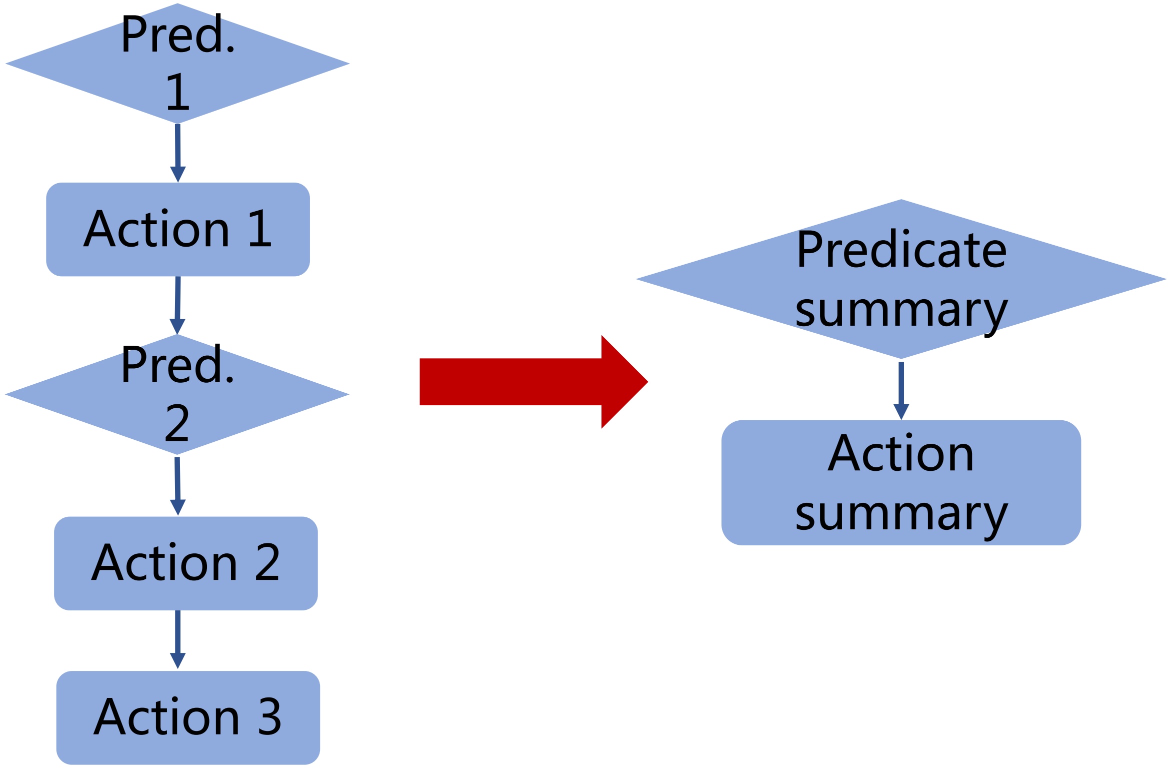
Intra-pipeline redundancy elimination

Goals:

1. Remove invalid paths
2. Shorten valid paths

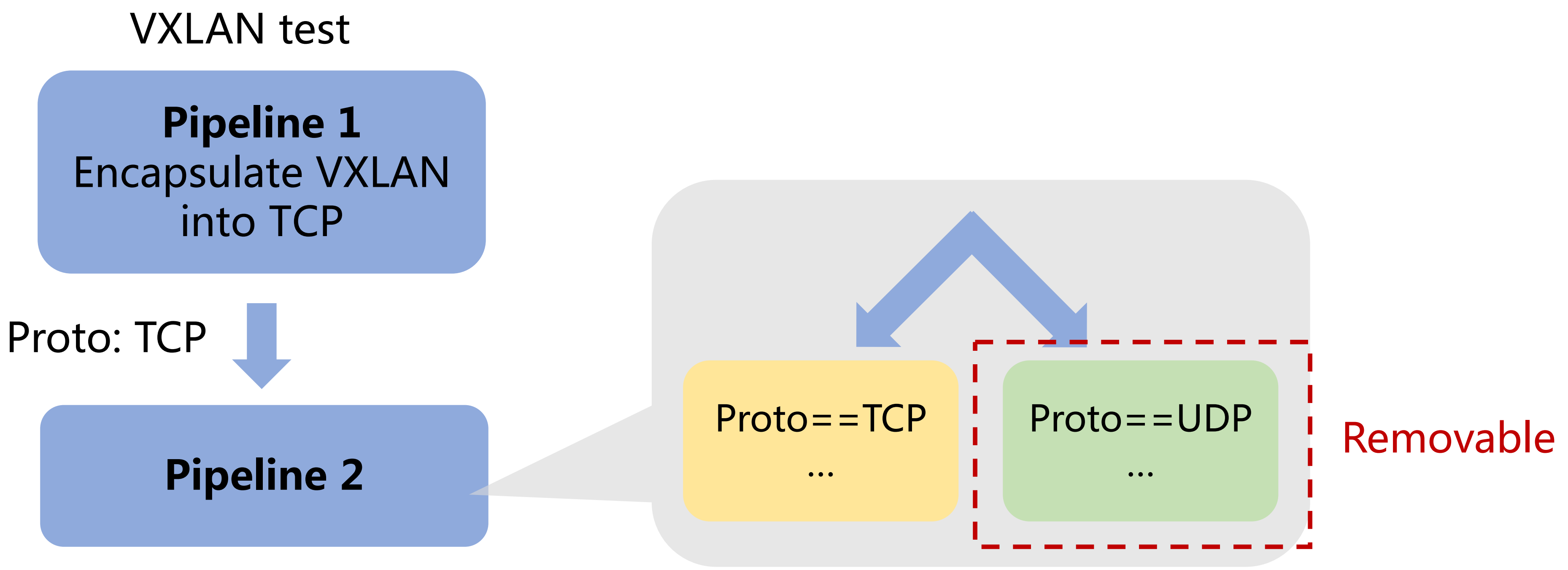
Algorithms:

1. Depth-first search
2. Collect paths' contexts
3. Gather summary nodes



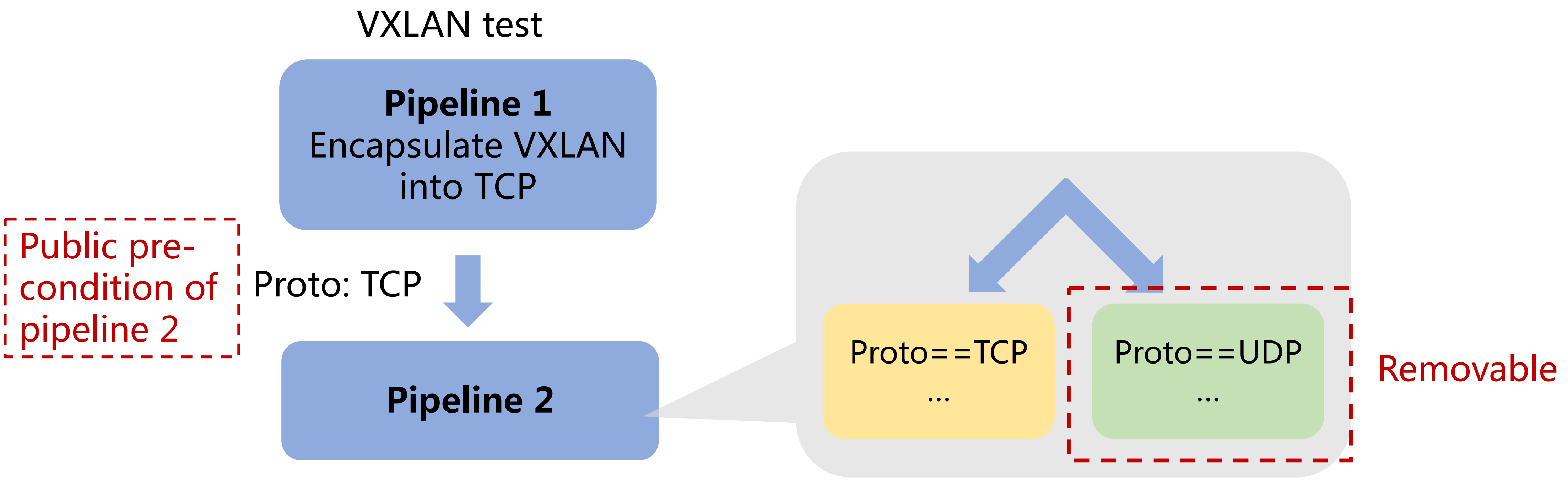
Inter-pipeline public pre-condition filtering

Note: Valid paths after intra-pipeline redundancy elimination may be invalid in the integral CFG.



Inter-pipeline public pre-condition filtering

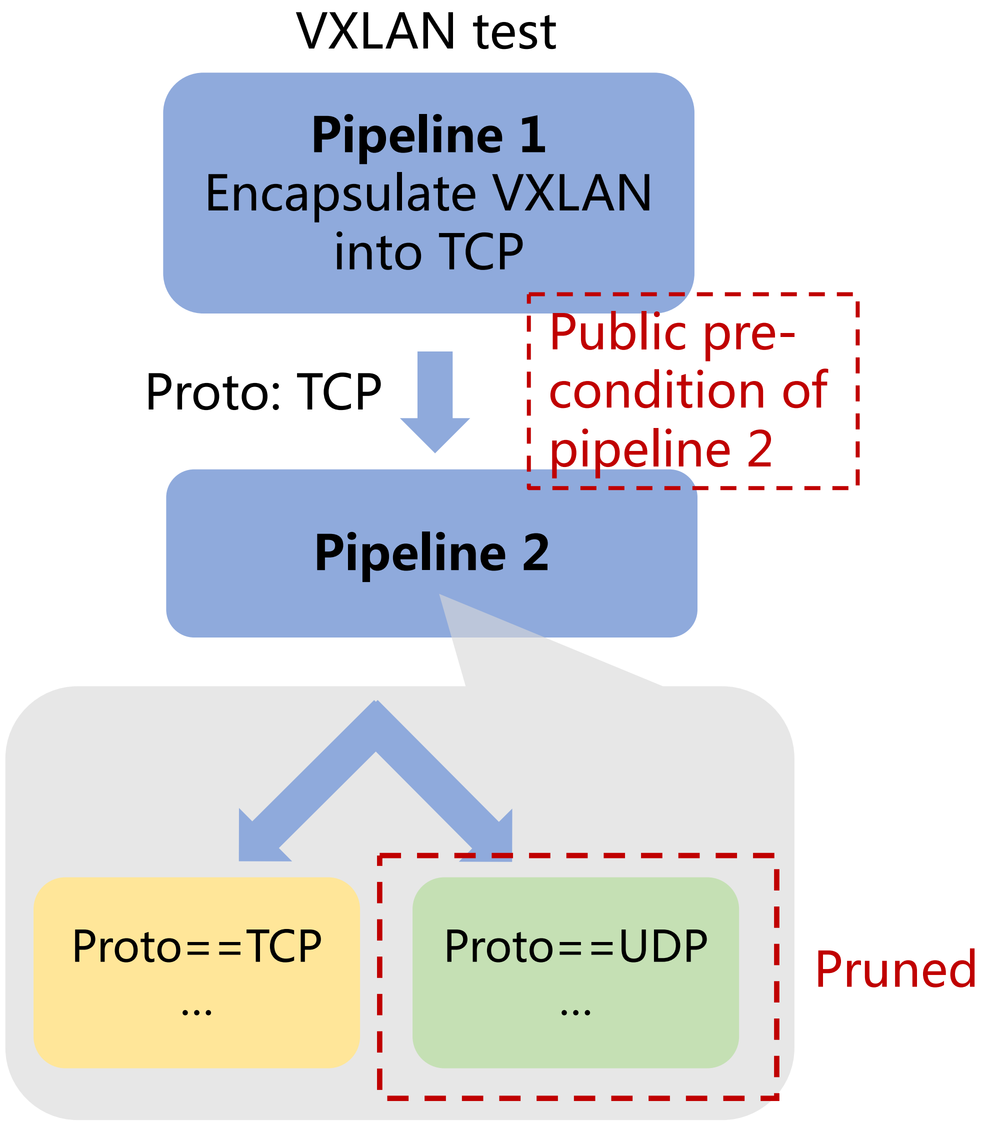
Public pre-condition:
The common conditions shared by the paths from the entry point of the program to the target pipeline.



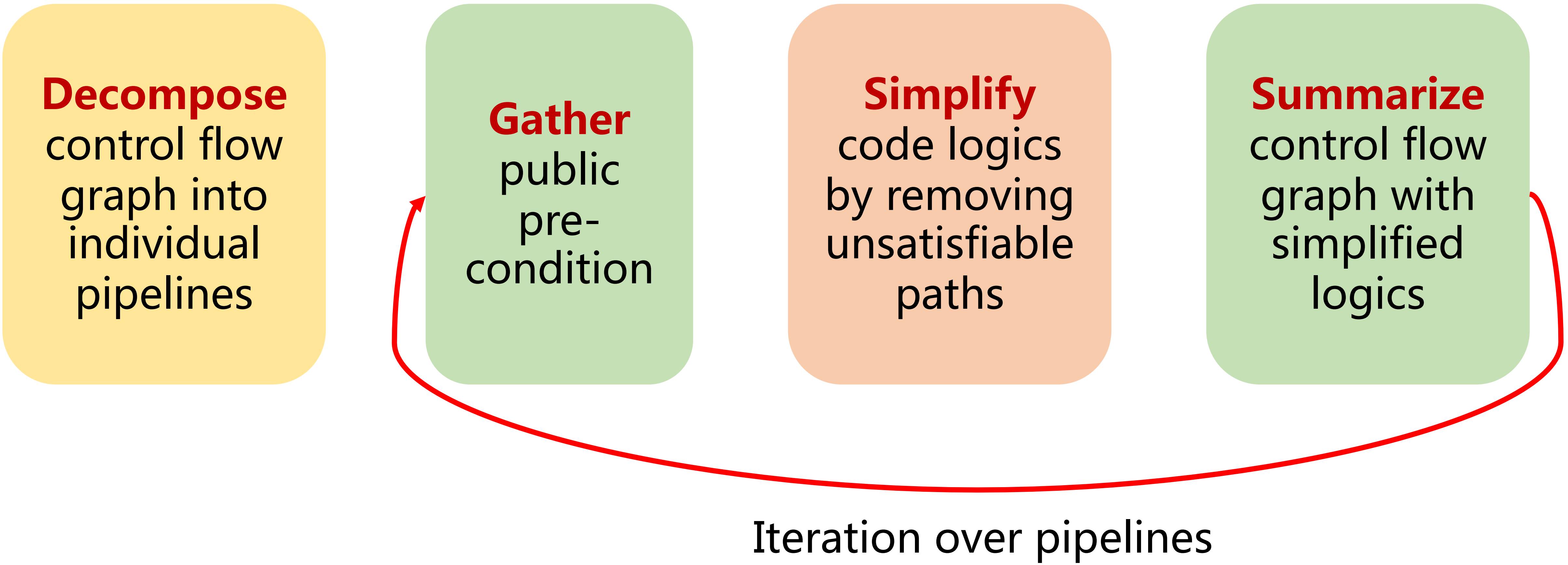
Inter-pipeline public pre-condition filtering

Algorithm:

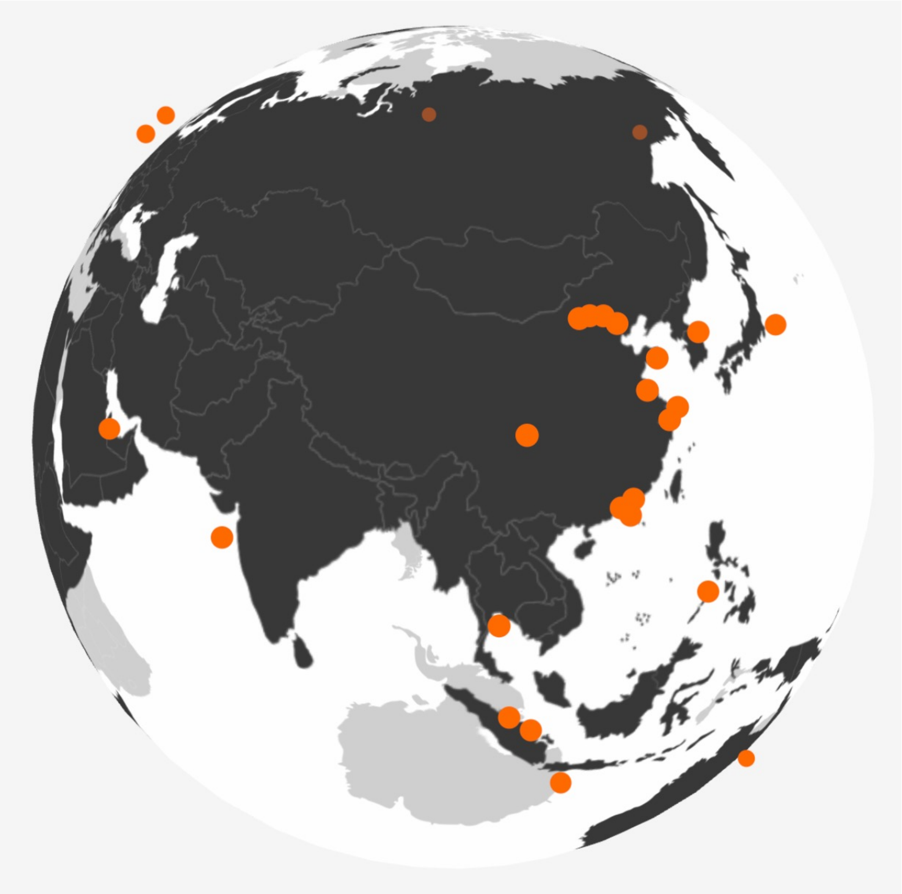
1. Collect all valid paths from entry point of program to the pipeline.
2. Analyze these paths to identify pre-conditions in common
3. Prune paths with public pre-conditions



Code summary technique



Meissa is deployed **globally**



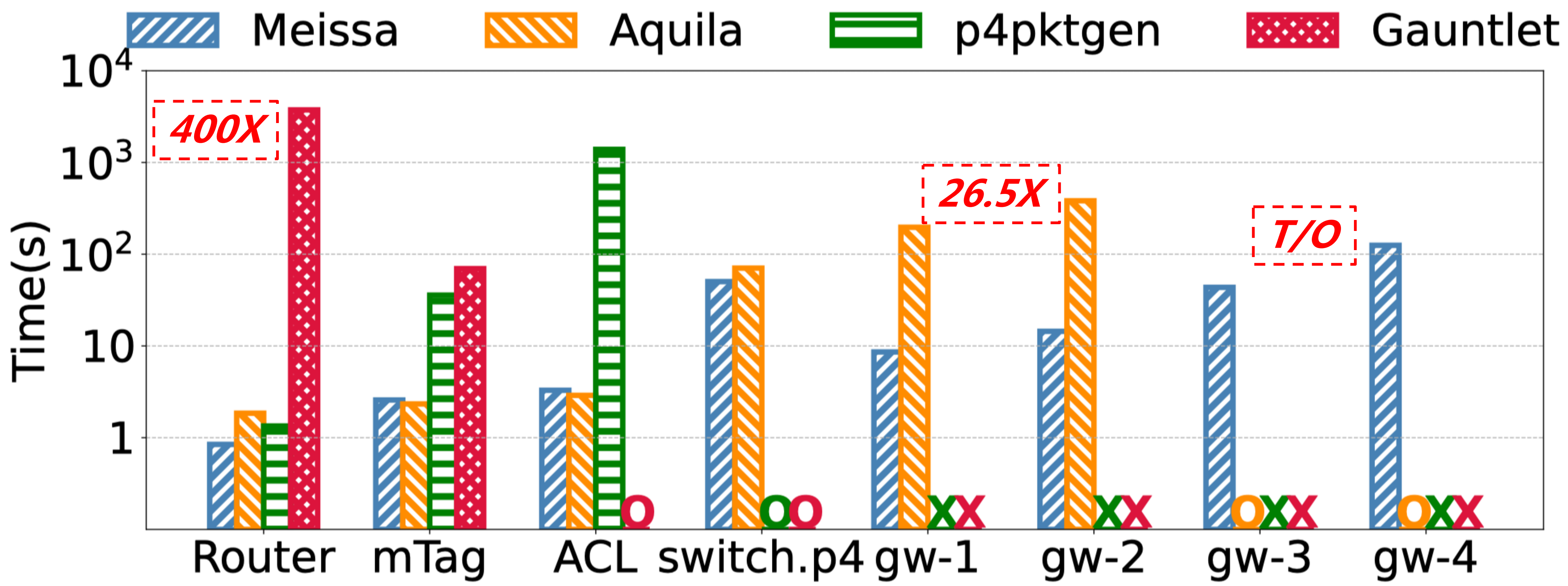
Alibaba Cloud

Since fall 2021, Meissa has been deployed in more than 200 P4 programmable gateways among 4 continents.

Evaluation methodology

	Name	LOC	# of pipelines	# of switches
Open-sourced	Router	256	1	1
	mTag	227	1	1
	ACL	400	1	1
	switch.p4	7086	1	1
Industrial production	gateway-1	>1000	1	1
	gateway-2	>3000	2	1
	gateway-3	>10000	4	1
	gateway-4	>20000	8	2

Scalability



O for time-out, X for non-support

Bug finding ability

Type	Index	Bug	Meissa	p4pktgen	PTA	Gauntlet	Aquila
Code Bugs	1	Routing misconfiguration	✓	✗	✗	✗	✓
	2	Unrestricted ACL rules	✓	✗	✗	✗	✓
	3	Parser wrong logic	✓	✓	✓	✓	✓
	4	Ingress wrong logic	✓	✓	✓	✓	✓
	5	Wrong deparser emit	✓	✗	✓	✗	✓
	6	Checksum fail-to-update	✓	✗	✗	✗	✗
Non-code Bugs	7	p4c frontend bug 2147	✓	✓	✗	✓	✗
	8	p4c frontend bug 2343	✓	✓	✗	✓	✗
	9	bf-p4c backend bug 1	✓	✗	✗	✓	✗
	10	bf-p4c backend bug 3	✓	✗	✗	✓	✗
	11	bf-p4c backend bug 6	✓	✗	✗	✓	✗
	12	bf-p4c backend bug A (incorrect arithmetic comparison)	✓	✗	✗	✗	✗
	13	bf-p4c backend bug B (incorrect assignment)	✓	✗	✗	✗	✗
	14	bf-p4c backend bug C (setValid)	✓	✗	✗	✗	✗
	15	Misuse of optimization pragmas	✓	✗	✗	✗	✗
	16	Missing compilation flags	✓	✗	✗	✗	✗

Unknown bugs

Conclusion

Meissa is a scalable network testing system for programmable data planes.

Meissa leverages a domain specific code summary technique to guarantee full coverage and scalability.

Meissa is developed for programmable switches, but its principals also apply to other programmable data plane devices.

Our Recent Work on Software-Defined Cloud Systems

Data Plane: Reliability

Meissa: Scalable Network Testing for Programmable Data Planes

SIGCOMM 2022
Amsterdam

Control Plane: Multi-Resource Scheduling

Multi-Resource Interleaving for Deep Learning Training

SIGCOMM 2022
Amsterdam

SIGCOMM 2022
Amsterdam

Multi-Resource Interleaving for Deep Learning Training

Yihao Zhao, Yuanqiang Liu, Yanghua Peng,
Yibo Zhu, Xuanzhe Liu, Xin Jin

北京大學
PEKING UNIVERSITY

ByteDance
字节跳动

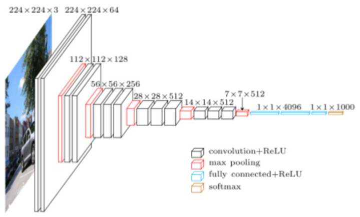
Deep Learning Training

Deep Learning (DL) is popular

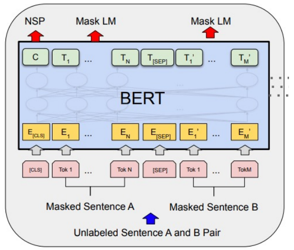
- DL training becomes an important workload in enterprises' clusters

DL training uses **multiple resource types**

- DL training is **iterative**
- DL training is **staged** and each stage mainly uses a specific resource type



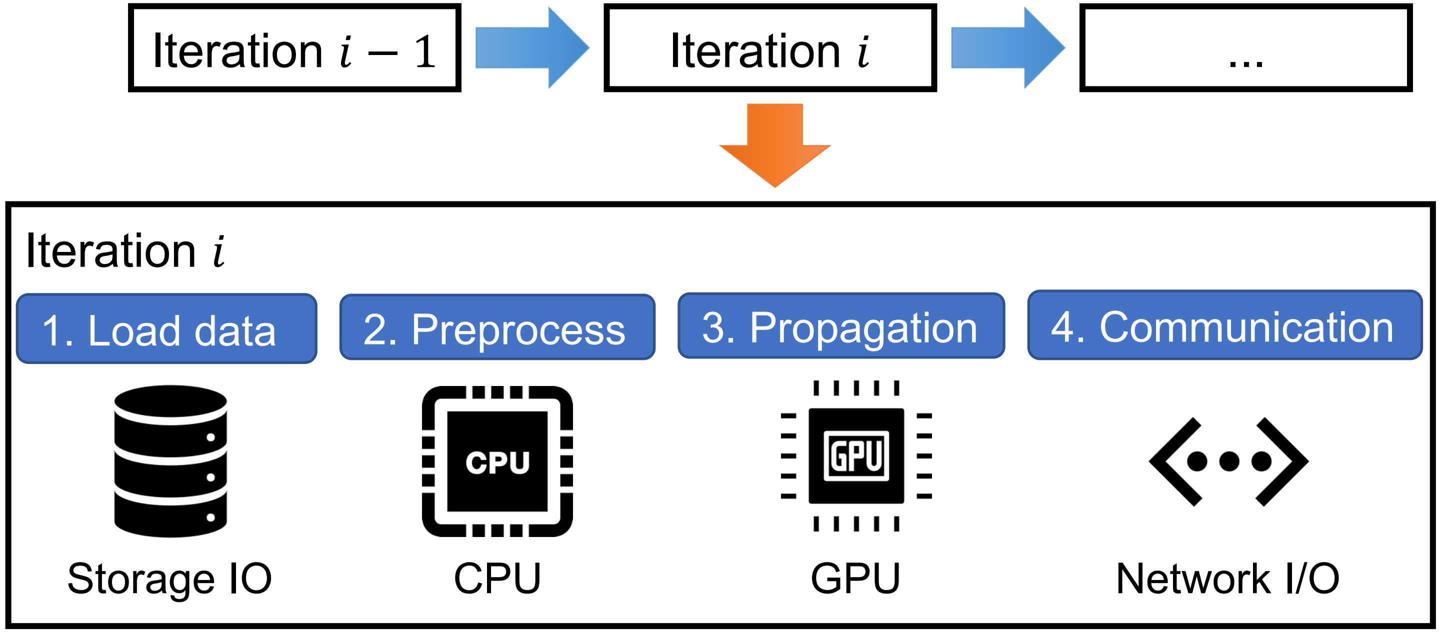
Computer Vision



Natural Language Processing

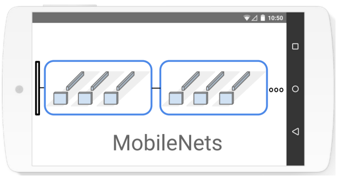
Reinforcement Learning

...



Deep Learning Training

A wide spectrum of DL models varies in resource requirements



...

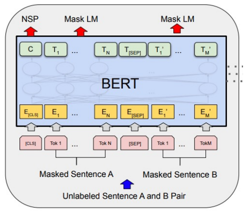
Tiny DL Models

Storage IO Bottlenecked

...

Reinforcement Learning

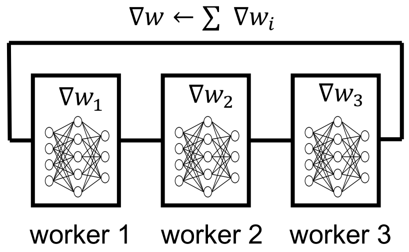
CPU Bottlenecked



...

Natural Language Processing

GPU Bottlenecked



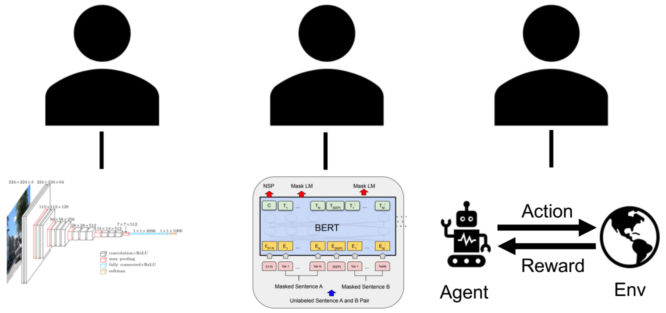
...

Distributed Training

Network IO Bottlenecked

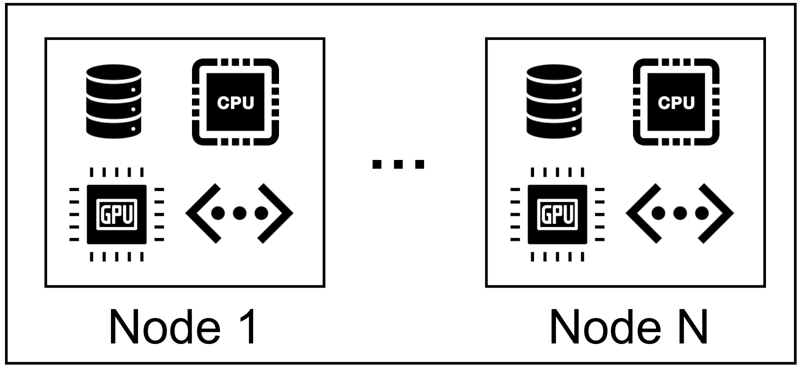
DL Training in Clusters

Users and DL training jobs



Scheduler

Cluster



Goal:

- Minimize finish time
- Maximize resource utilization
- …

Current DL Scheduler:

Most allocate GPUs to a job exclusively

Some explore only GPU sharing

Miss the opportunity of multi-resource sharing!

DL Training in Clusters

Challenges of multi-resource sharing

- Reduce interference among shared DL jobs
- Improve both job and cluster efficiency

Some explore only GPU sharing

Miss the opportunity of multi-resource sharing!

Our approach (Muri)

A DL cluster scheduler that utilizes MUlti-Resource Interleaving to improve job and cluster efficiency

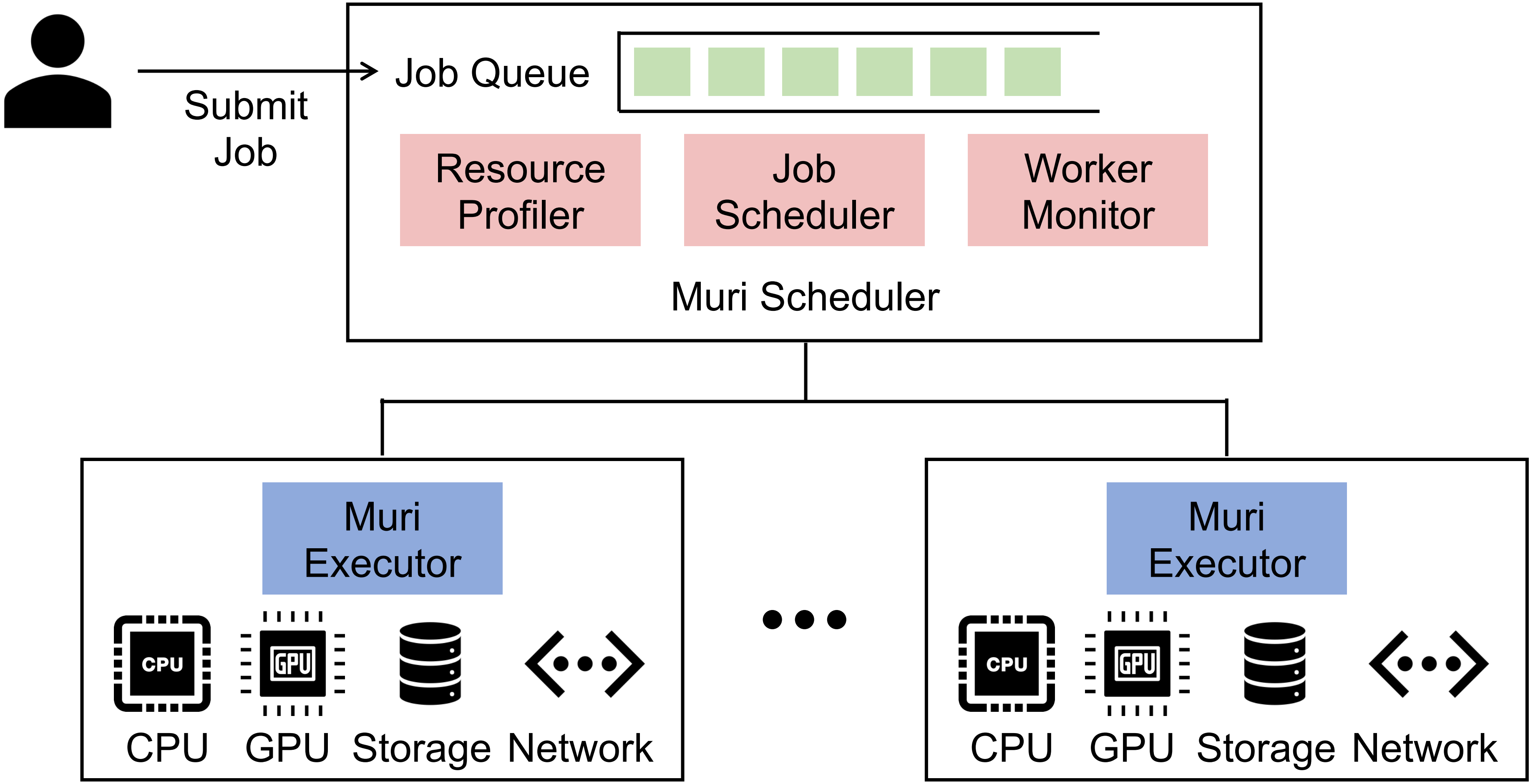
Multi-resource interleaving

- Pack jobs on the same set of resources by interleaving stages in time
- Reduce interference among shared jobs

Blossom-based scheduler

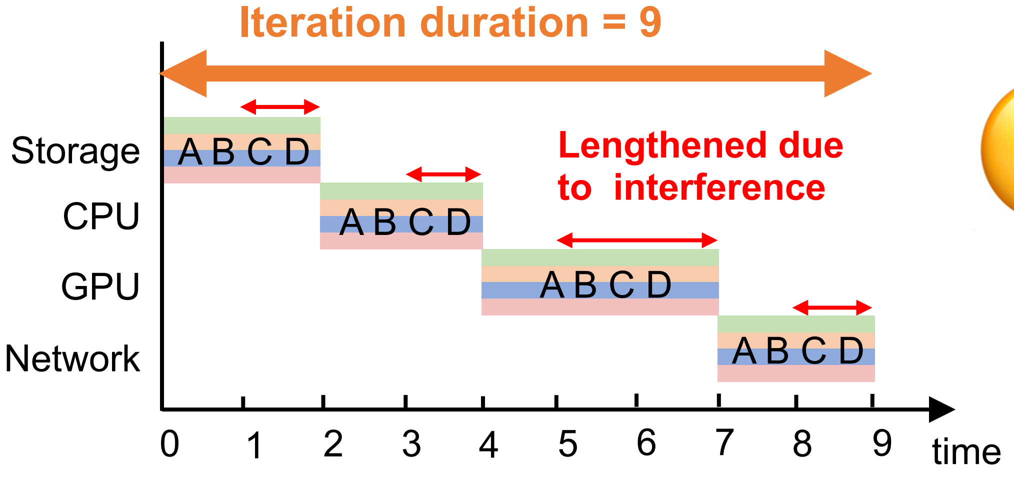
- Assign sharing groups to maximize interleaving efficiency
- Improve both job and cluster efficiency

Muri Architecture



Multi-Resource Sharing

Space sharing



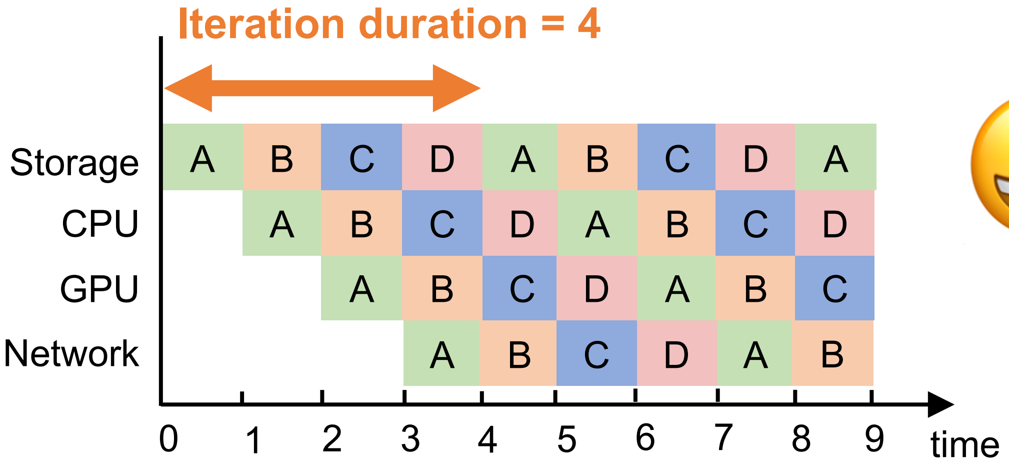
High interference among shared jobs leads to **longer iteration duration**

- At every moment, each resource type on one machine can be used by multiple jobs

Time sharing

Resource utilization

Cost



A B C D: DL training jobs

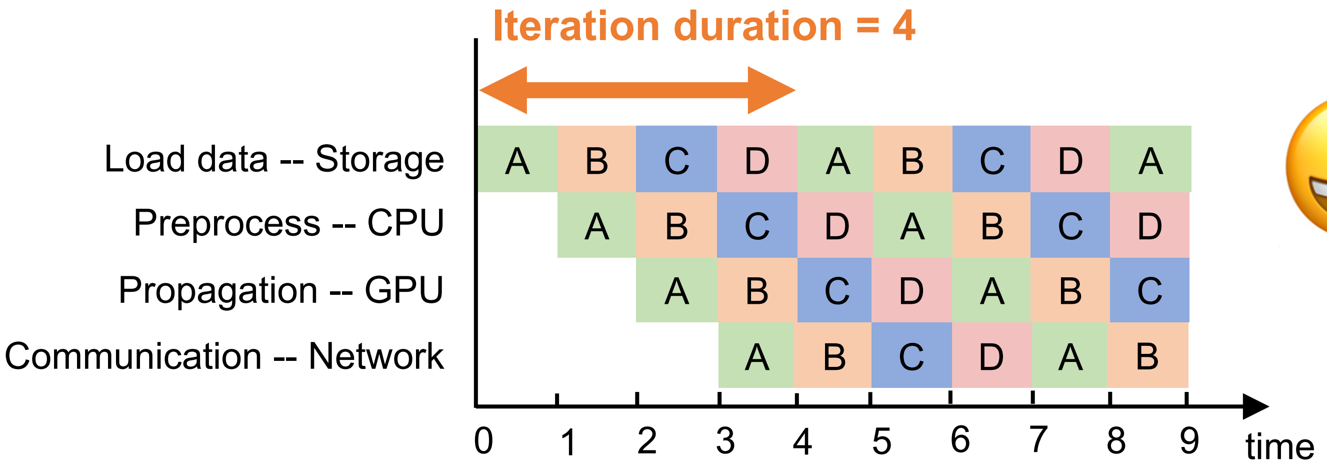
Low interference among shared jobs brings **shorter iteration duration**

- At every moment, each resource type on one machine can be used by only one job

Muri: Multi-Resource Interleaving

Muri exploits fine-grained multi-resource interleaving in time

- Staged pattern of DL training brings inherent stages to interleave
- Iterative pattern of DL training enables low-overhead scheduling decision for interleaving



A B C D: DL training jobs

Low interference among shared jobs brings **shorter iteration duration**

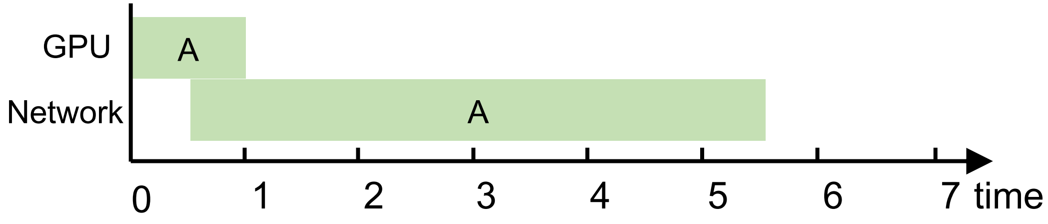
- At every moment, each resource type on one machine can be used by only one job

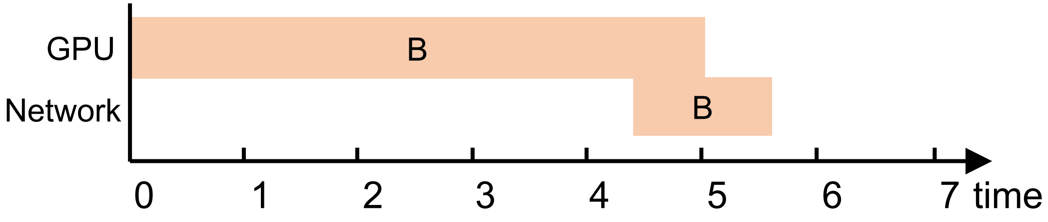
Multi-Resource Interleaving vs. Pipelining

Pipelining

Overlap multiple resources **intra-job**

Throughput when job A and B are run separately: $\frac{1}{5.5+5.5} = \frac{1}{11}$ iterations/s



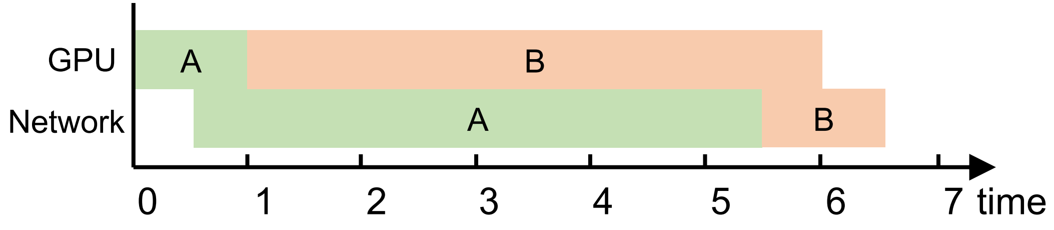


Multi-resource interleaving

Overlap multiple resources **inter-job**

Throughput when job A and B are interleaved: $\frac{1}{\max(5.5,6.5)} = \frac{1}{6.5}$ iterations/s

1.7× higher throughput!



Muri: Capture Interleaving Efficiency

Interleaving efficiency represents how perfect a grouping plan can overlap the resource usage of the jobs

$$\boxed{\gamma} = \frac{1}{k}\sum_{j=0}^{k-1} \boxed{\frac{\sum_{i=0}^{p-1} t_i^j}{T}}$$

Interleaving efficiency

Occupied time ratio of each resource

Iteration duration can be estimated by

$$T = \sum_{j=0}^{k-1} \max_{i=0}^{p-1} t_i^{(i+j) mod\ k}$$

k: the number of resource types

p: the number of jobs in one group

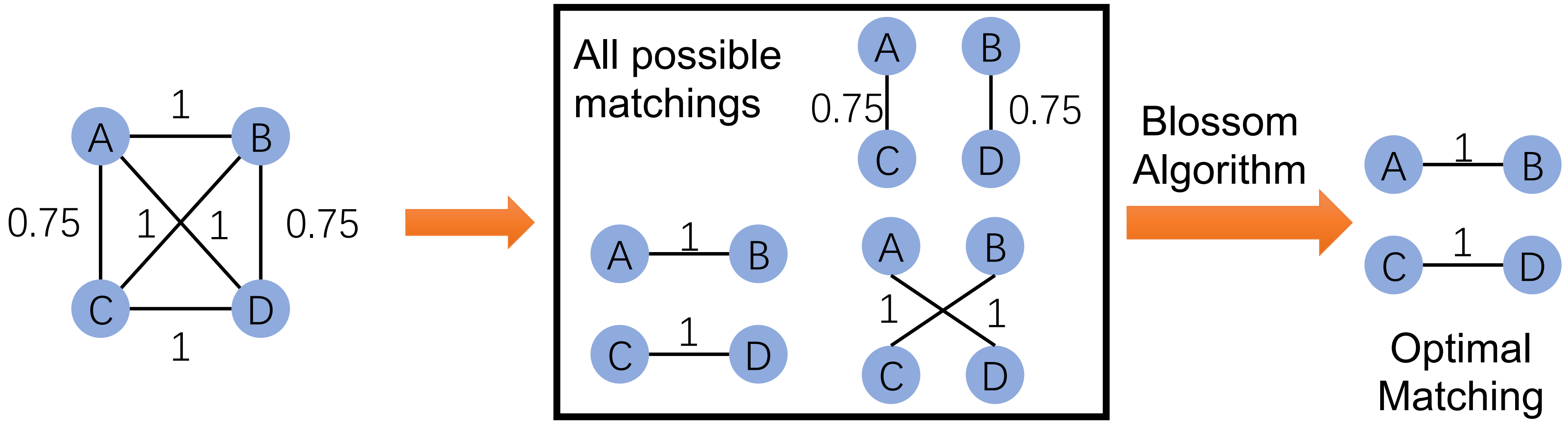
t_i^j: the duration that job i uses resource j

Muri Scheduler: Select Jobs to Interleave

Formulate as a **maximum weighted matching problem** for two resource types

- Node: a group of jobs that are interleaved
- Edge: interleave the jobs in the two nodes
- Edge weight: the interleaving efficiency
- Matching: a grouping plan

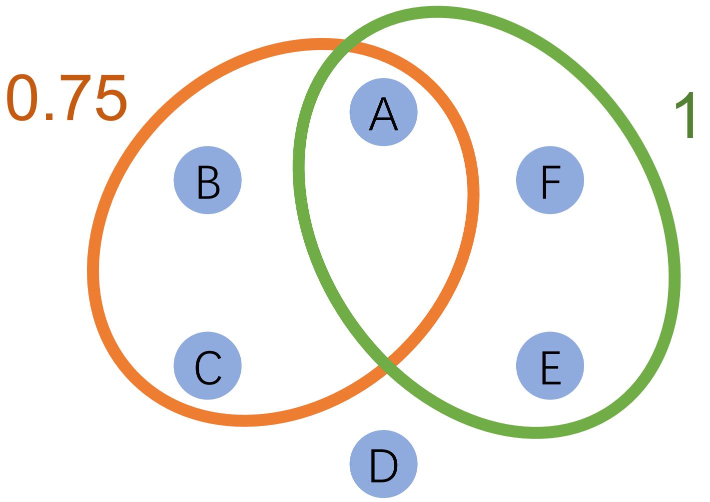
Optimal for two resource types



Muri Scheduler: Select Jobs to Interleave

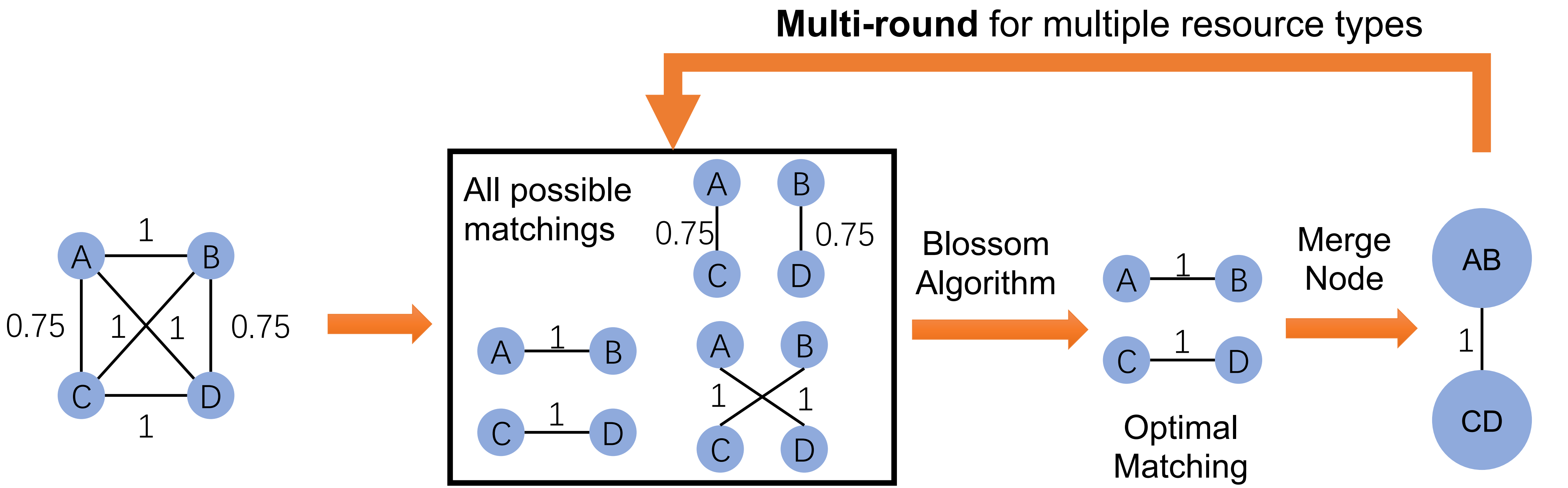
For more than two resource types…

- Maximum weighted k-uniform hypergraph matching
- **NP-Hard!**



Muri Scheduler: Select Jobs to Interleave

Multi-round heuristic algorithm for multiple resource types



Muri: Other Design Details

Handle multi-GPU jobs

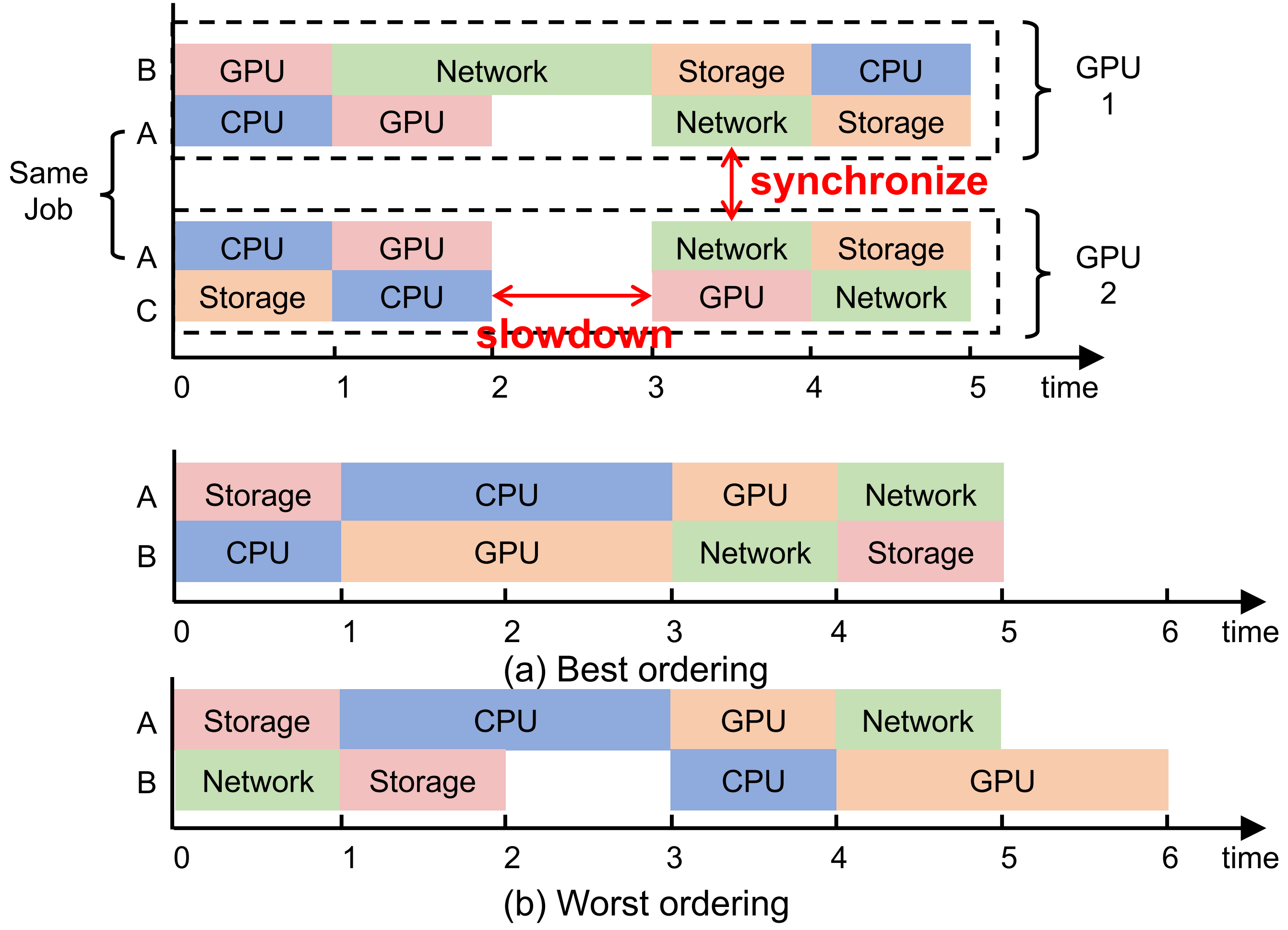
- Only group jobs with the same GPU requirement as intra-job synchronization brings slowdown

Optimize interleaving efficiency

- (a) has interleaving efficiency $\gamma \approx 0.5$
- (b) has interleaving efficiency $\gamma = 0.4$
- Enumerate all orderings of a group as the ordering of jobs affects the interleaving efficiency

Optimize average JCT

- Assign a priority to each job
- SRSF when job durations are known
- 2D-LAS when job durations are unknown



Evaluation

- Implementation: ~7,000 LOC
  - PyTorch 1.8.1
  - CUDA 11.1
- Testbed
  - 64-GPU cluster, NVIDIA Tesla V100 GPU
- Traces
  - Philly Trace from Microsoft [Jeon et al. 2019]
- Models
  - CV: ResNet18, ShuffleNet, VGG16, VGG19
  - NLP: Bert, GPT-2
  - RL: A2C, DQN

Testbed Experiments: Overall Performance

8 nodes w/ 8 GPUs each (V100)
400 DL jobs submitted over 10s days

	SRTF	SRSF	Muri-S
Normalized JCT	2.12	2.03	1
Normalized Makespan	1.56	1.59	1
Normalized 99^{th}%-ile JCT	3.31	3.82	1

Job durations are known

	Tiresias	Themis	Muri-L
Normalized JCT	2.59	3.56	1
Normalized Makespan	1.48	1.47	1
Normalized 99^{th}%-ile JCT	2.54	2.60	1

Job durations are unknown

Testbed Experiments: Overall Performance

8 nodes w/ 8 GPUs each (V100)
400 DL jobs submitted over 10s days

	SRTF	SRSF	Muri-S
Normalized JCT	2.12	2.03	1
Normalized Makespan	1.56	1.59	1
Normalized 99^{th}%-ile JCT	3.31	3.82	1

Job durations are known

Job efficiency

- $> 2\times$ faster average job completion time
- $> 2.5\times$ faster tail job completion time

Cluster efficiency

- $> 1.4\times$ faster makespan

	Tiresias	Themis	Muri-L
Normalized JCT	2.59	3.56	1
Normalized Makespan	1.48	1.47	1
Normalized 99^{th}%-ile JCT	2.54	2.60	1

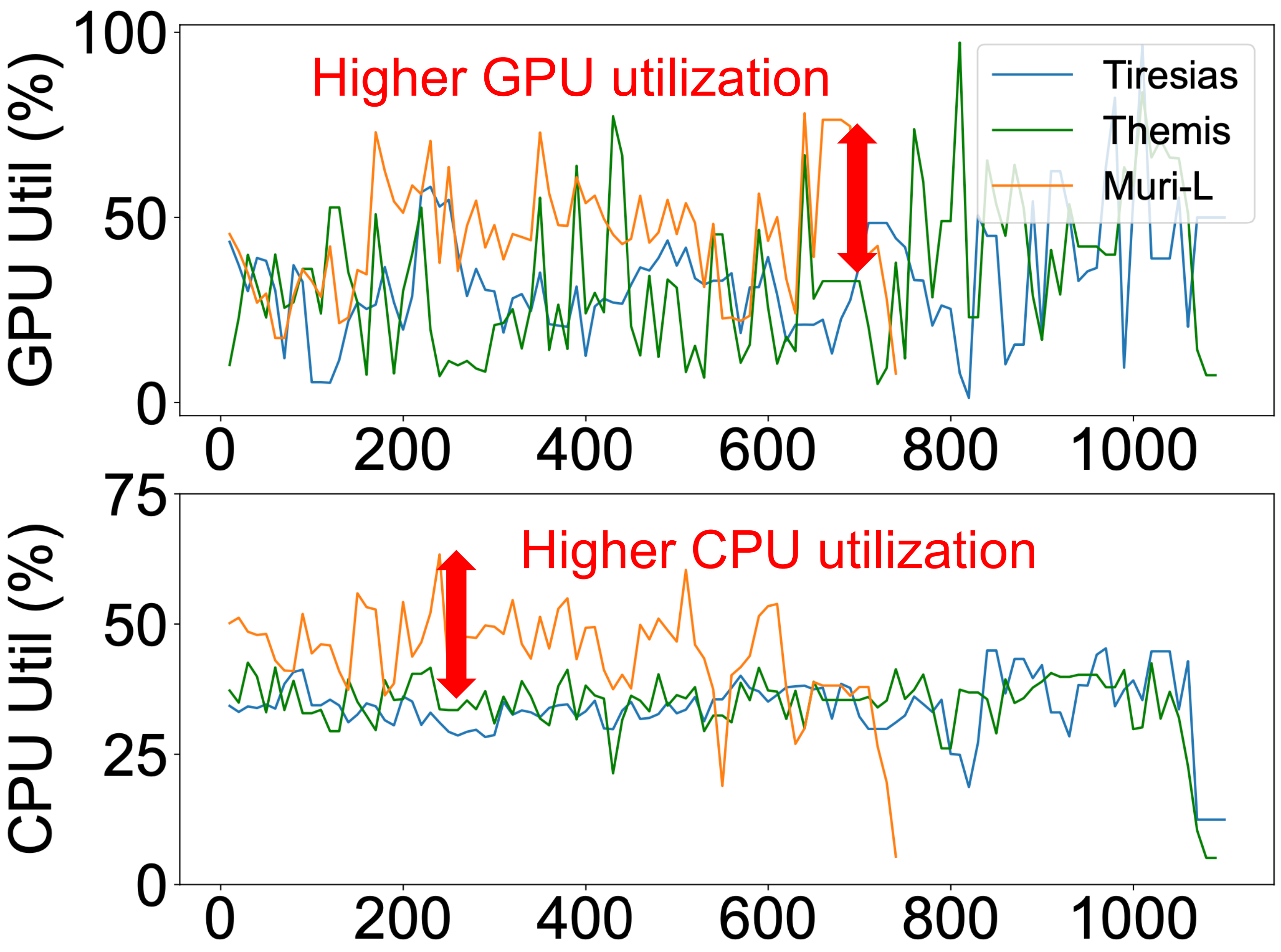
Job durations are unknown

Testbed Experiments: Detailed Metrics

Job durations are unknown

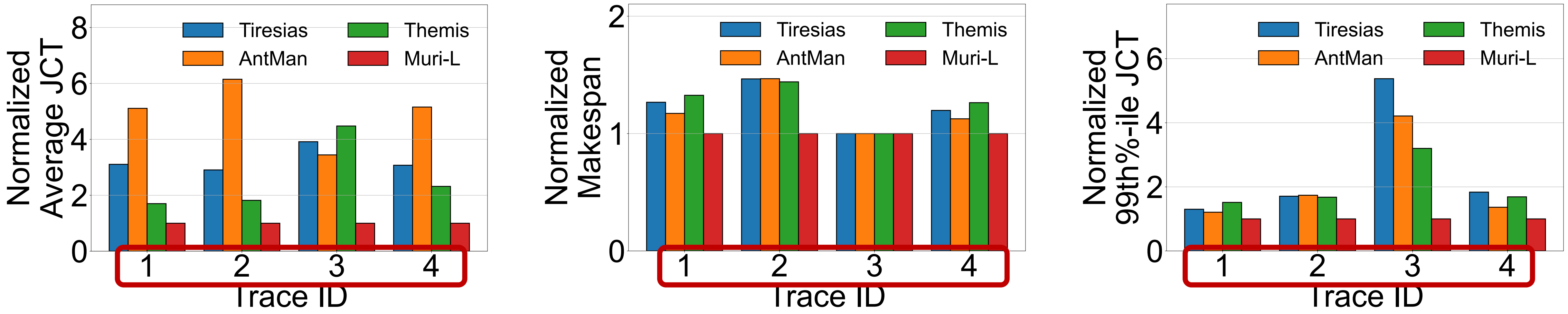
Higher utilization

- **36%** higher average GPU utilization
- **30%** higher average CPU utilization
- Other resources in our paper!



Trace-Driven Simulations

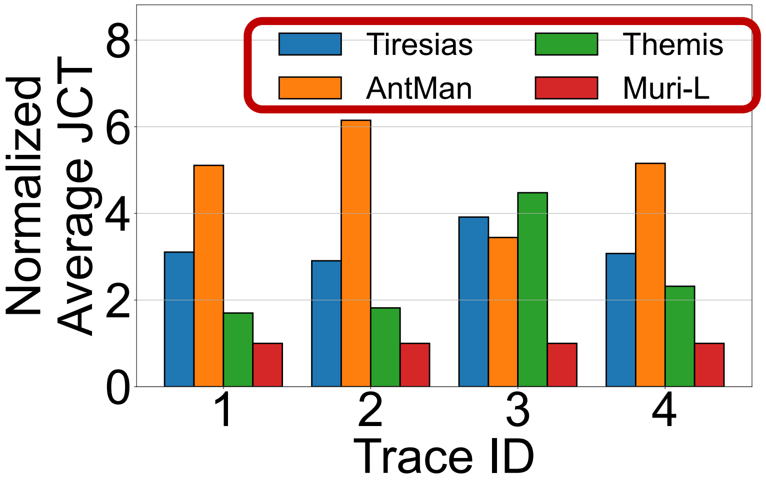
Job durations are unknown



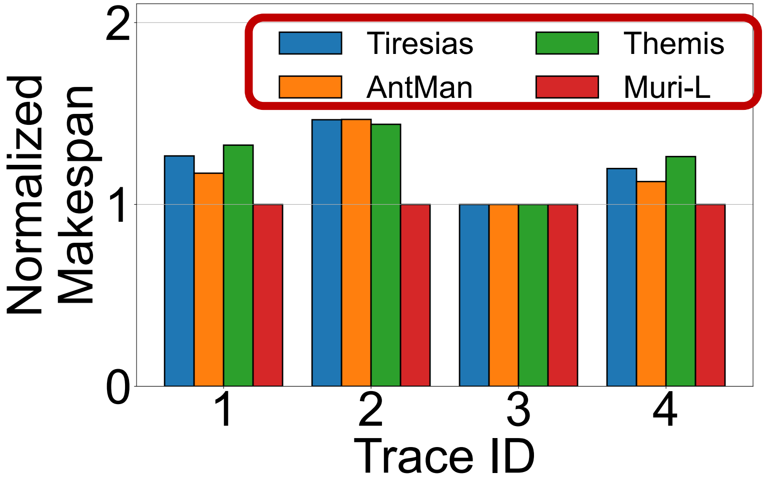
(a) Average JCT (b) Makespan (c) 99^{th}%-ile JCT

Trace-Driven Simulations

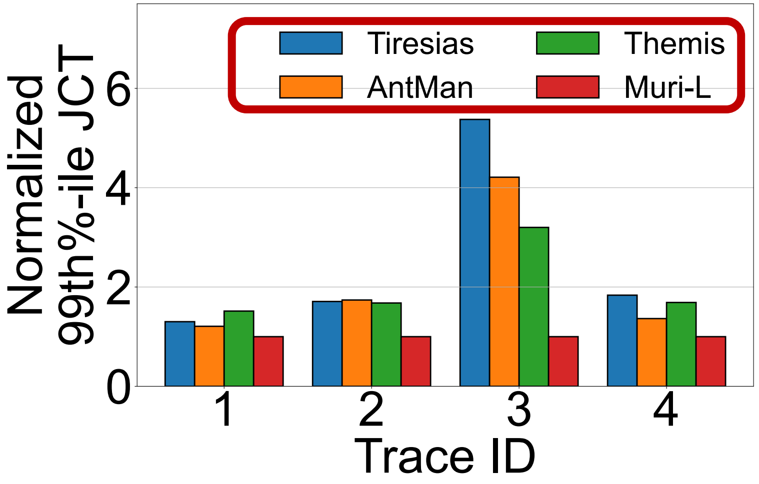
Job durations are unknown



(a) Average JCT



(b) Makespan

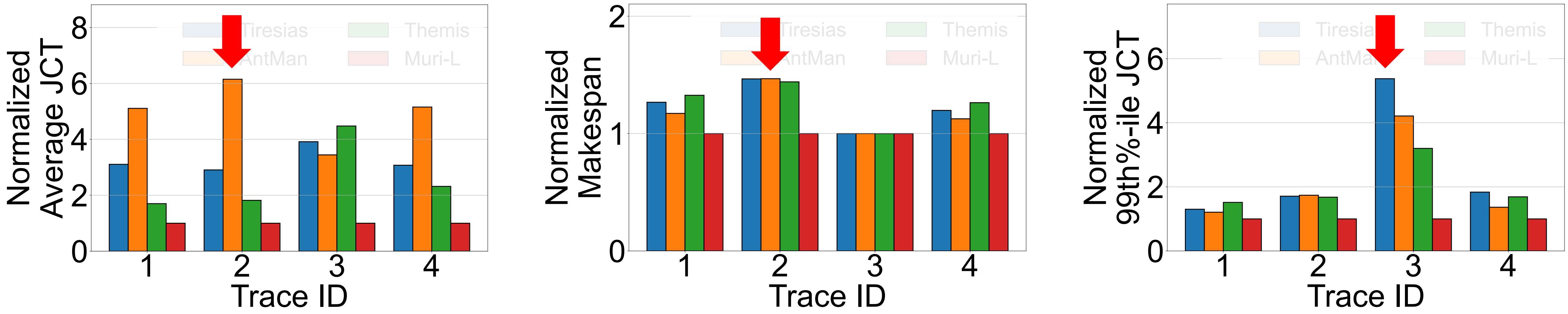


(c) 99^{th}%-ile JCT

Trace-Driven Simulations

Job durations are unknown

Results: improve up to $6.1\times$ avg. JCT, $1.5\times$ makespan, and $5.4\times$ tail JCT



(a) Average JCT (b) Makespan (c) 99th%-ile JCT

More Experiments in our Paper

- Performance when job durations are known
- More detailed metrics
- Analysis of Muri
  - Impact of designs
  - Impact of workload distributions
  - Impact of inaccurate profiling
- …

Conclusion

- Muri: a multi-resource cluster scheduler for DL workloads
  - Introduce multi-resource interleaving to share jobs in *time*
  - Utilize a Blossom-based scheduling algorithm to maximize the interleaving efficiency
- Muri improves average JCT by up to $6.1\times$ and makespan by up to $1.6\times$

Open-sourced at https://github.com/pkusys/Muri

Thanks!

Xin Jin
xinjinpku@pku.edu.cn