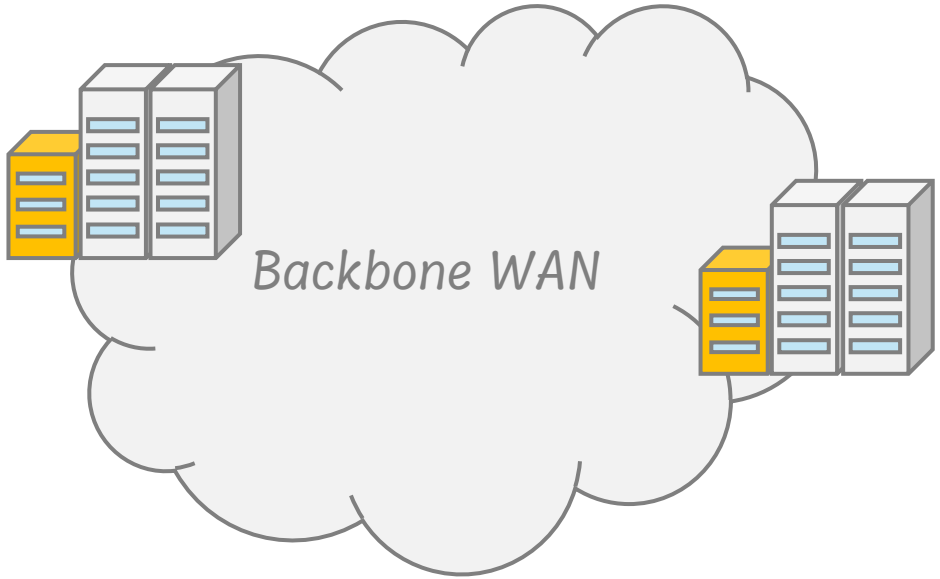
S2: A Distributed Configuration Verifier for Hyper-Scale Networks

Dan Wang, Peng Zhang,
Wenbing Sun, Wenkai Li,
Xing Feng, Hao Li

Jaiwei Chen,
Weirong Jiang,
Yongping Tang

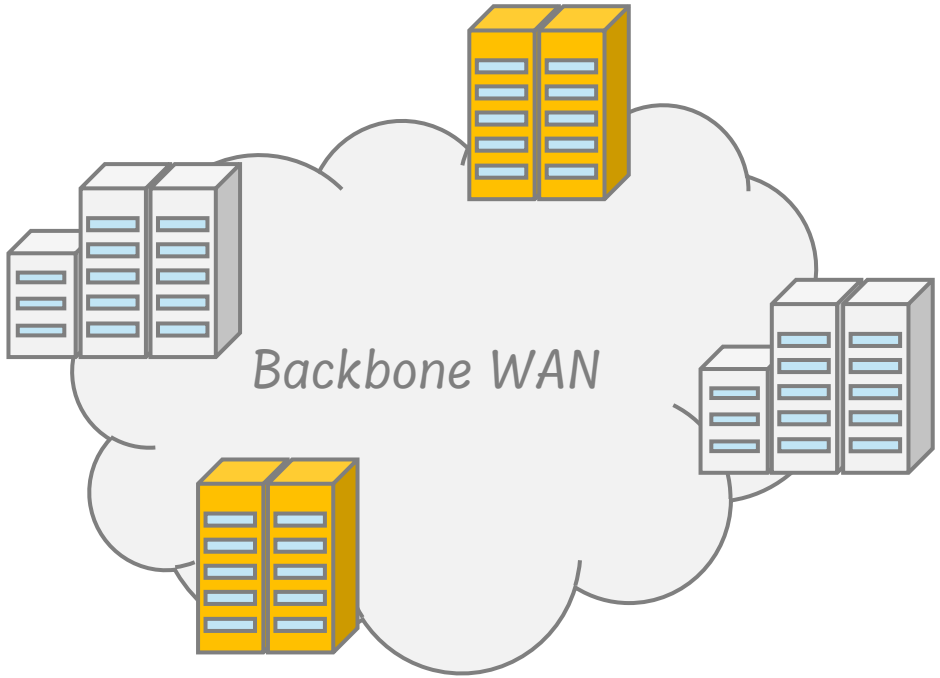
Networks are achieving hyper-scale

- Each DCN is getting larger



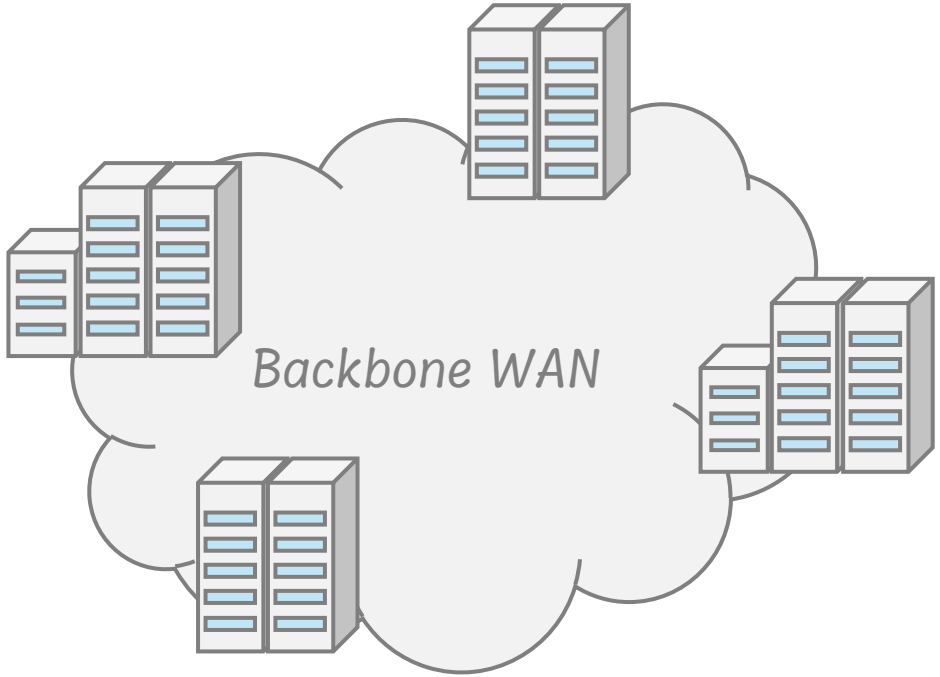
Networks are achieving hyper-scale

- New DCNs are built continuously



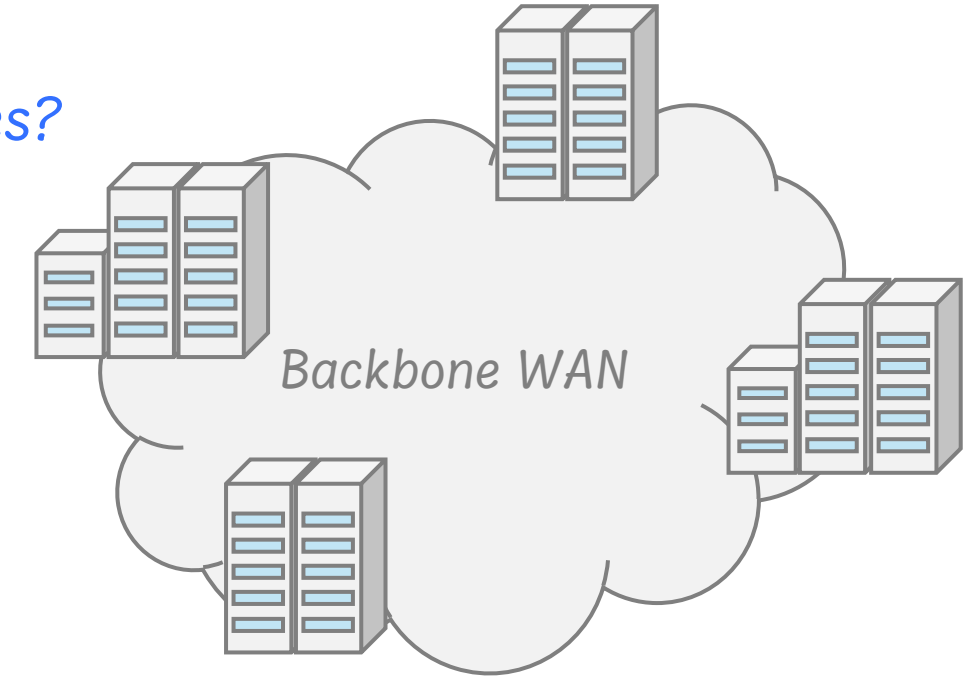
Networks are achieving hyper-scale

!!! >10K switches !!!



Hyper-scale networks are not error-prone?

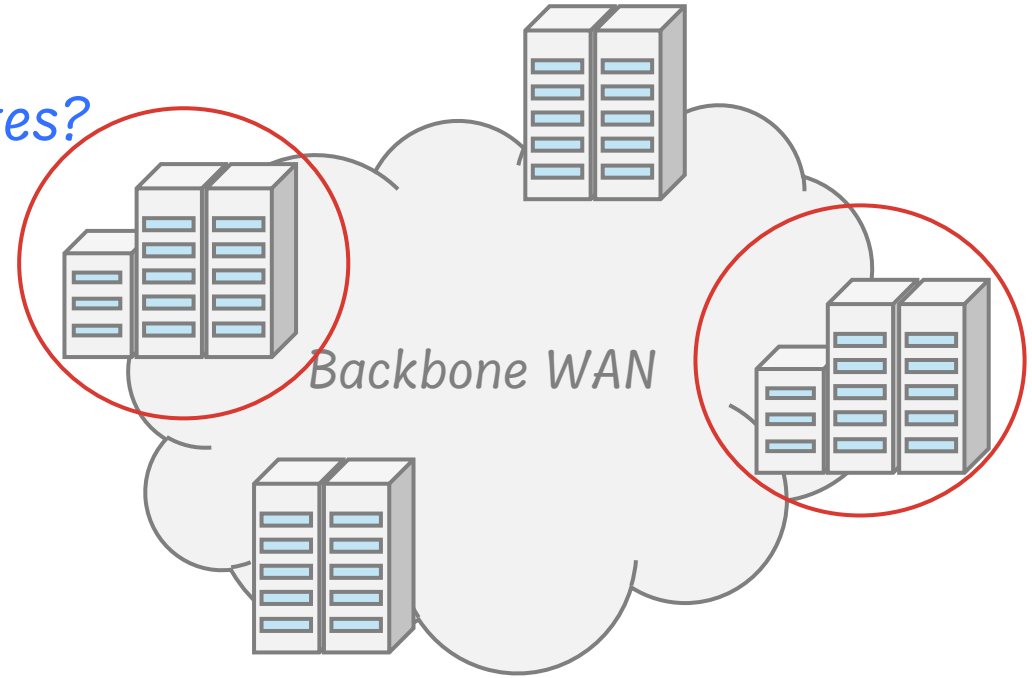
- Well-structured?
- Standard configuration with templates?



>10K switches

Hyper-scale networks are ~~not~~ error-prone!

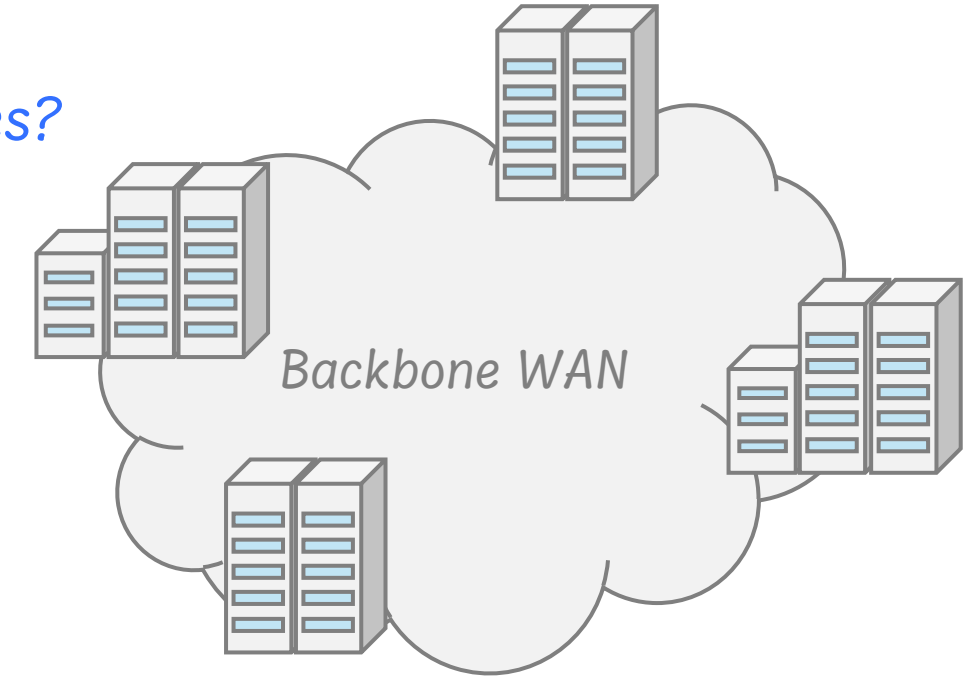
- Well-structured? -> heterogenous
- Standard configuration with templates?



>10K switches

Hyper-scale networks are ~~not~~ error-prone!

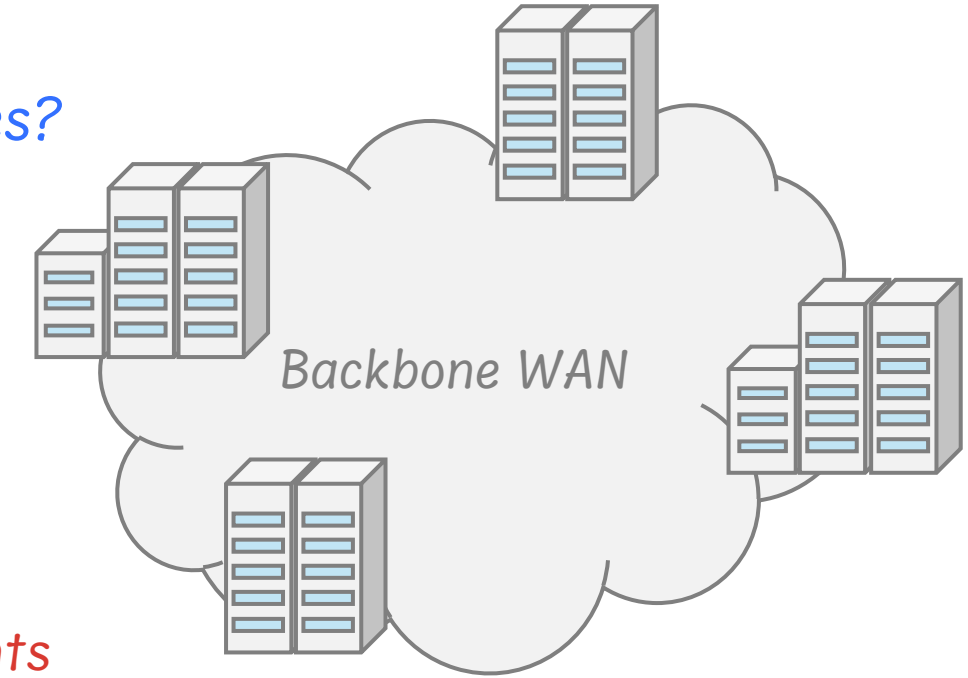
- Well-structured? -> heterogenous
- Standard configuration with templates?
  -> non-standard



>10K switches

Hyper-scale networks are ~~not~~ error-prone!

- Well-structured? -> heterogenous
- Standard configuration with templates?
  -> non-standard
  - > Route aggregation
  - > ECMP with different max path
  - > AS_Path overwrite
  - > Dual stack (IPv4 + IPv6)
  - > Vendor specific behavior -> 30% incidents



>10K switches

Can we verify hyper-scale networks (>10K switches) within a reasonable amount of time (2h)?

Control Plane Verifiers

Simulation-based CPVs	Analysis-based CPVs	Modular CPVs
Batfish [SIGCOMM'23]	ARC [SIGCOMM'16]	Lightyear [SIGCOMM'23]
FastPlane [VMCAI'19]	Minesweeper [SIGCOMM'17]	Timepiece [PLDI'23]
ShapeShifter [POPL'20]	Tiramisu [NSDI'20]	Kirigami [ToN'24]
Bonsai [SIGCOMM'18]		

Control Plane Verifiers

Limited support for network features

Batfish [SIGCOMM'23]
FastPlane [VMCAI'19]
ShapeShifter [POPL'20]
Bonsai [SIGCOMM'18]

Simulation-based CPVs

ARC [SIGCOMM'16]
Minesweeper [SIGCOMM'17]
Tiramisu [NSDI'20]

Analysis-based CPVs

Lightyear [SIGCOMM'23]
Timepiece [PLDI'23]
Kirigami [ToN'24]

Modular CPVs

1. Limited support for properties
2. Require much effort from users

We choose to simulate

Batfish [SIGCOMM'23]
FastPlane [VMCAI'19]
ShapeShifter [POPL'20]
Bonsai [SIGCOMM'18]

Simulation-based CPVs

Existing simulators "scale up"

Batfish [SIGCOMM'23] -> Parallelism

FastPlane [VMCAI'19] -> BGP scheduling

ShapeShifter [POPL'20] -> Abstract Interpretation

Bonsai [SIGCOMM'18] -> Compression

Simulation-based CPVs

Existing simulators "scale up"

Batfish [SIGCOMM'23] -> Parallelism -> #switches >> #cores

FastPlane [VMCAI'19] -> BGP scheduling -> Require monotonicity

ShapeShifter [POPL'20] -> Abstract Interpretation -> loose precision

Bonsai [SIGCOMM'18] -> Compression -> per-prefix compression

Simulation-based CPVs

Existing simulators "scale up"

Batfish [SIGCOMM'23] -> Parallelism -> #switches >> #cores

FastPlane [VMCAI'19] -> BGP scheduling -> Require monotonicity

ShapeShifter [POPL'20] -> Abstract Interpretation -> loose precision

Bonsai [SIGCOMM'18] -> Compression -> per-prefix compression

None reported scaling to >10K switches

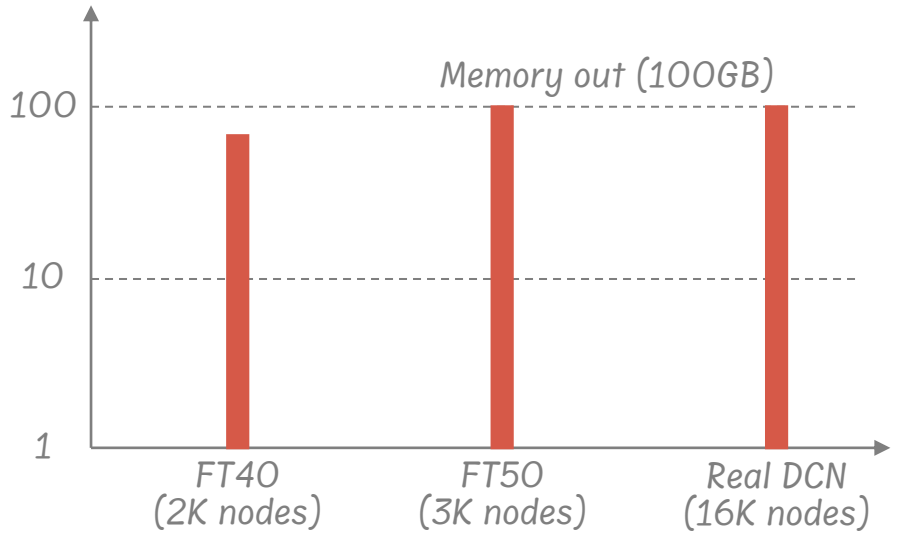
Simulation-based CPVs

Why State-of-the-Art simulators cannot scale to hyper-scale networks?

Simulating hyper-scale networks is challenging

- Memory intensive

FatTreeK, O(K^5) number of routes
FatTree60, around 400 million routes
Real DCN, around 200 (300) million IPv4 (IPv6) routes

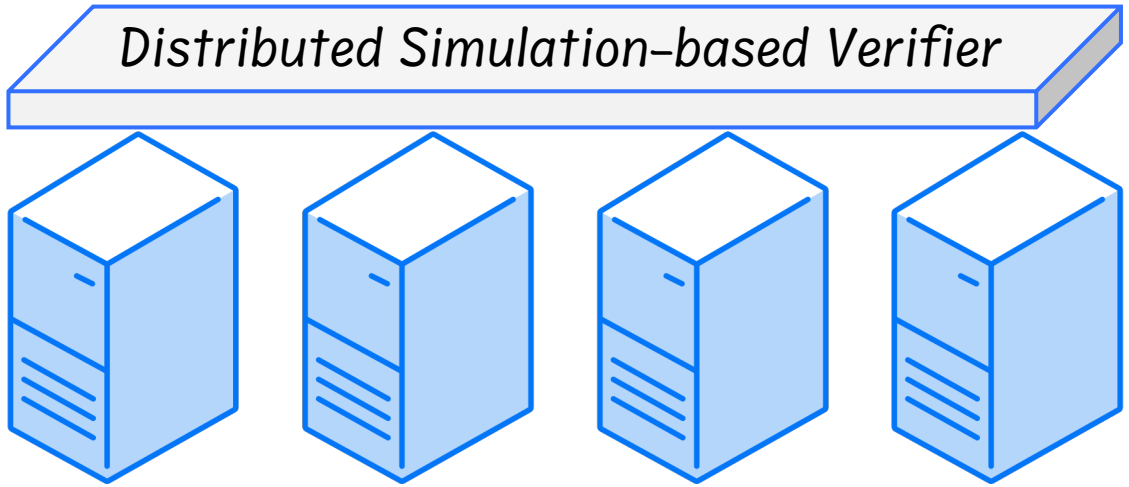


Simulate with Batfish (15 cores, 100GM memory)

- Compute intensive and lacks parallelism

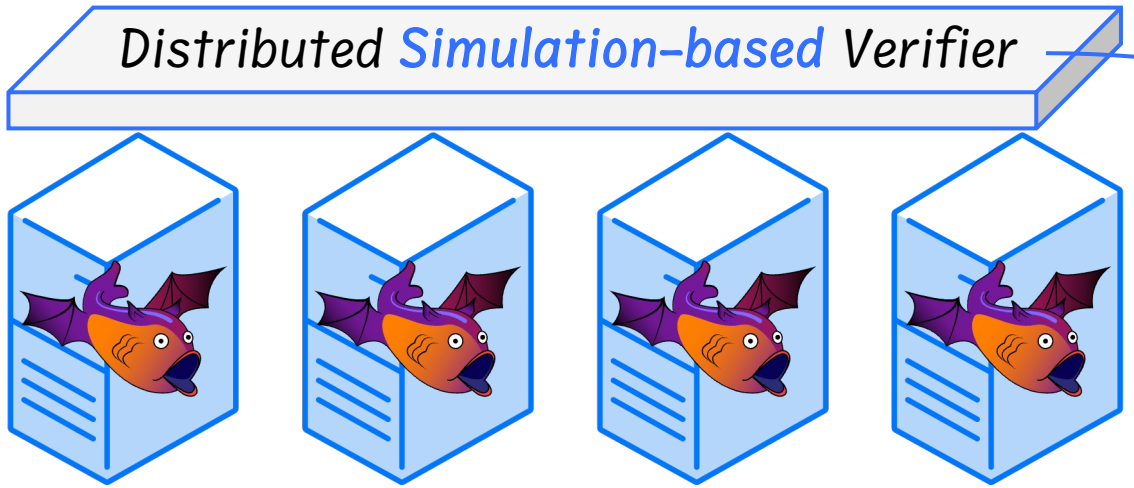
> Control Plane: #switches >> #cores
> Data Plane: non-parallel BDD

We choose to "scale-out"



We choose to "decouple"

the distributed framework from the switch model



Benefit from

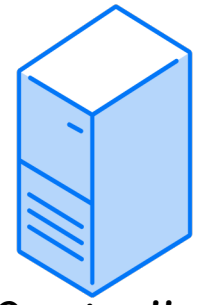
> Maturity

> Continuous evolution

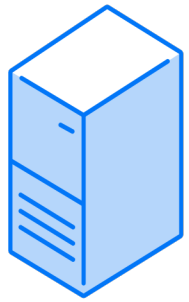
We propose S2

Scalable Simulation-based Verifier

Architecture of S2

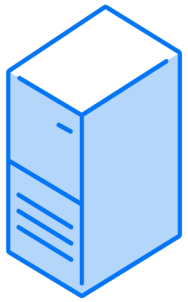


Controller



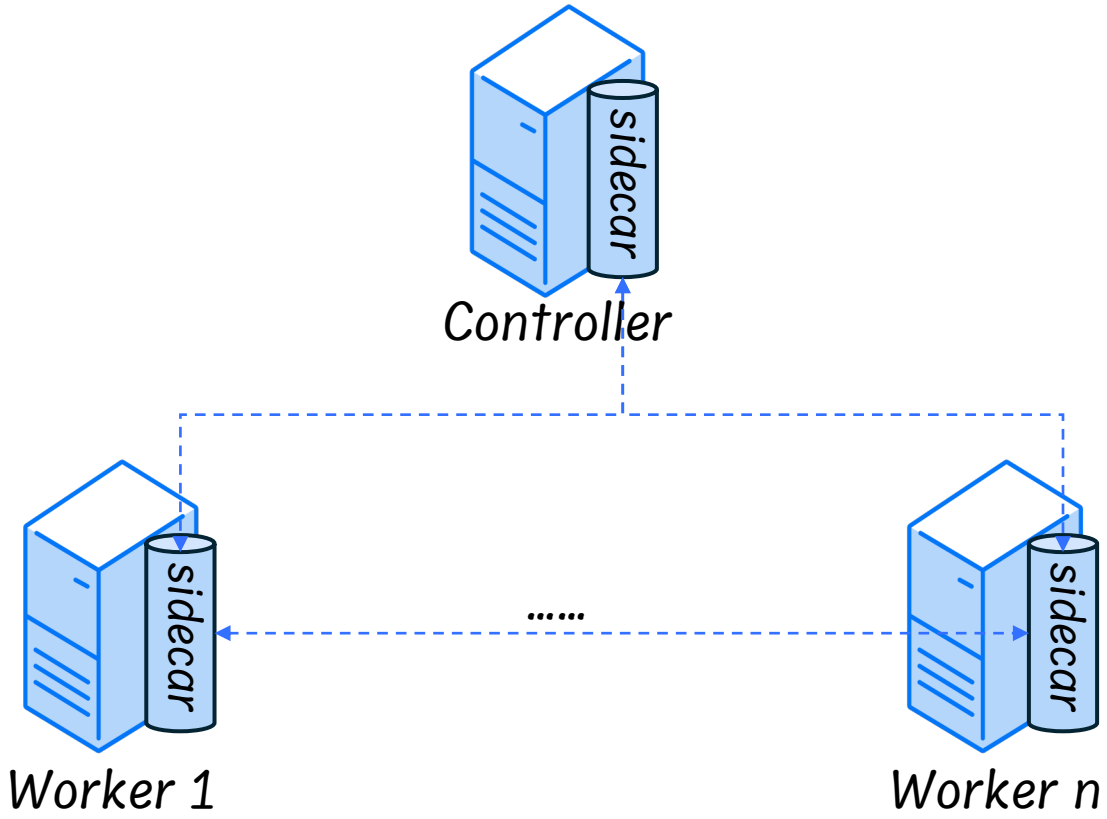
Worker 1

……

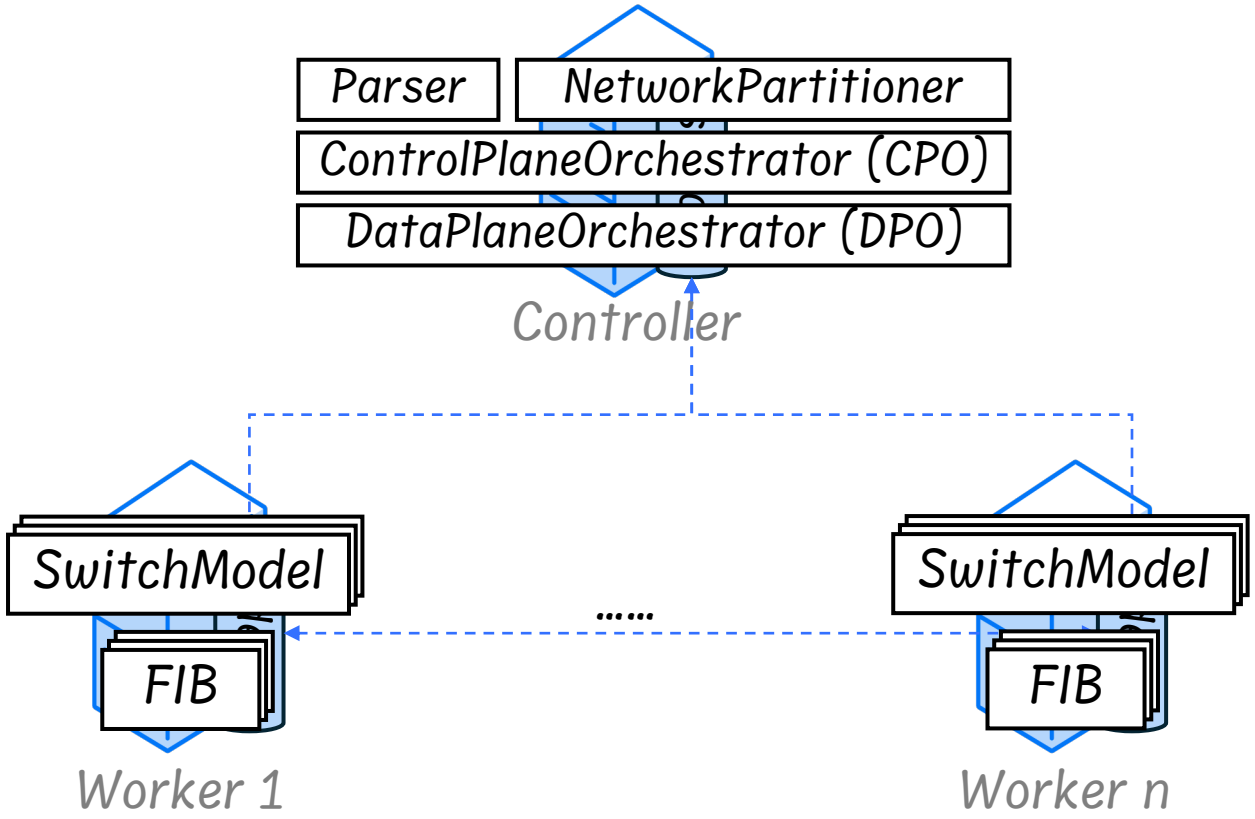


Worker n

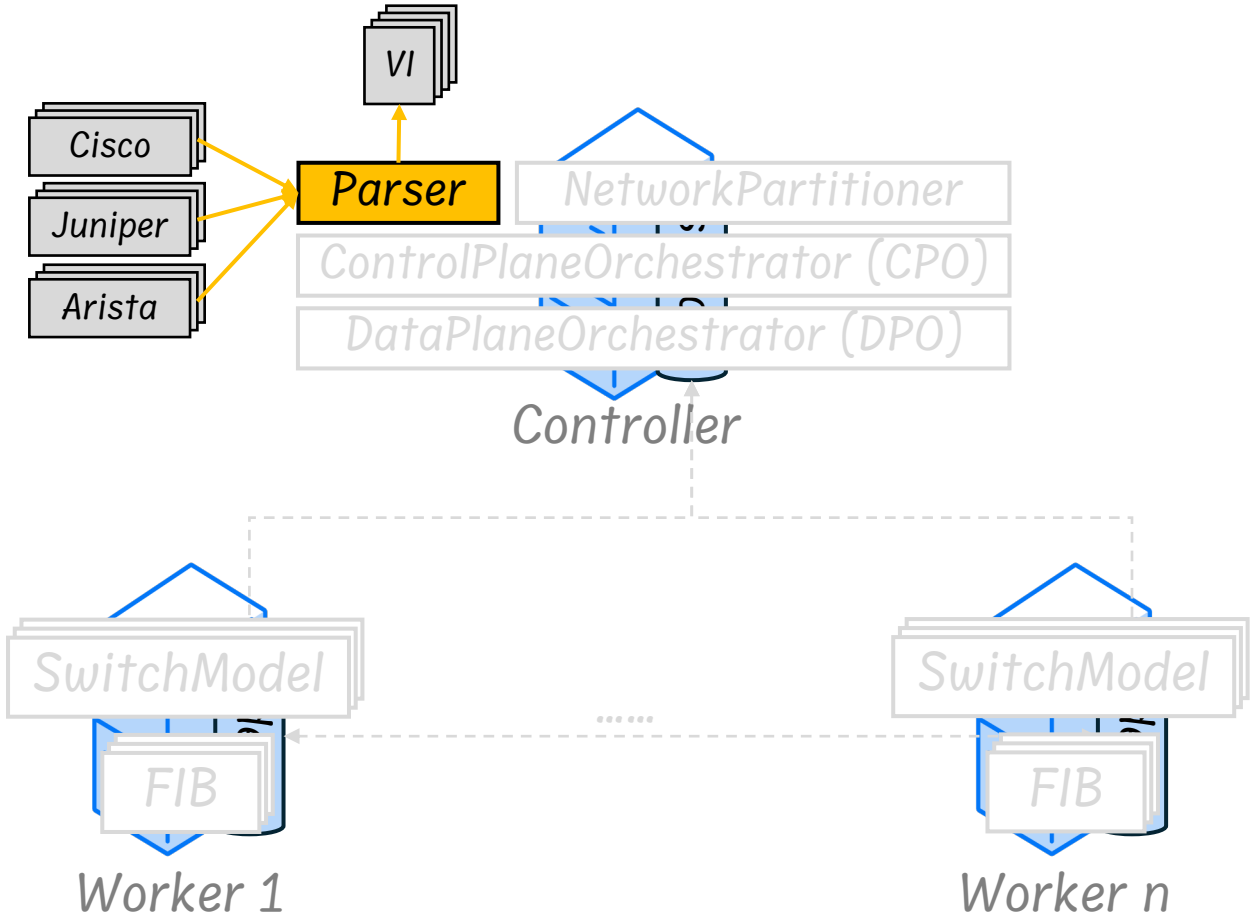
Architecture of S2



Architecture of S2



Workflow of S2



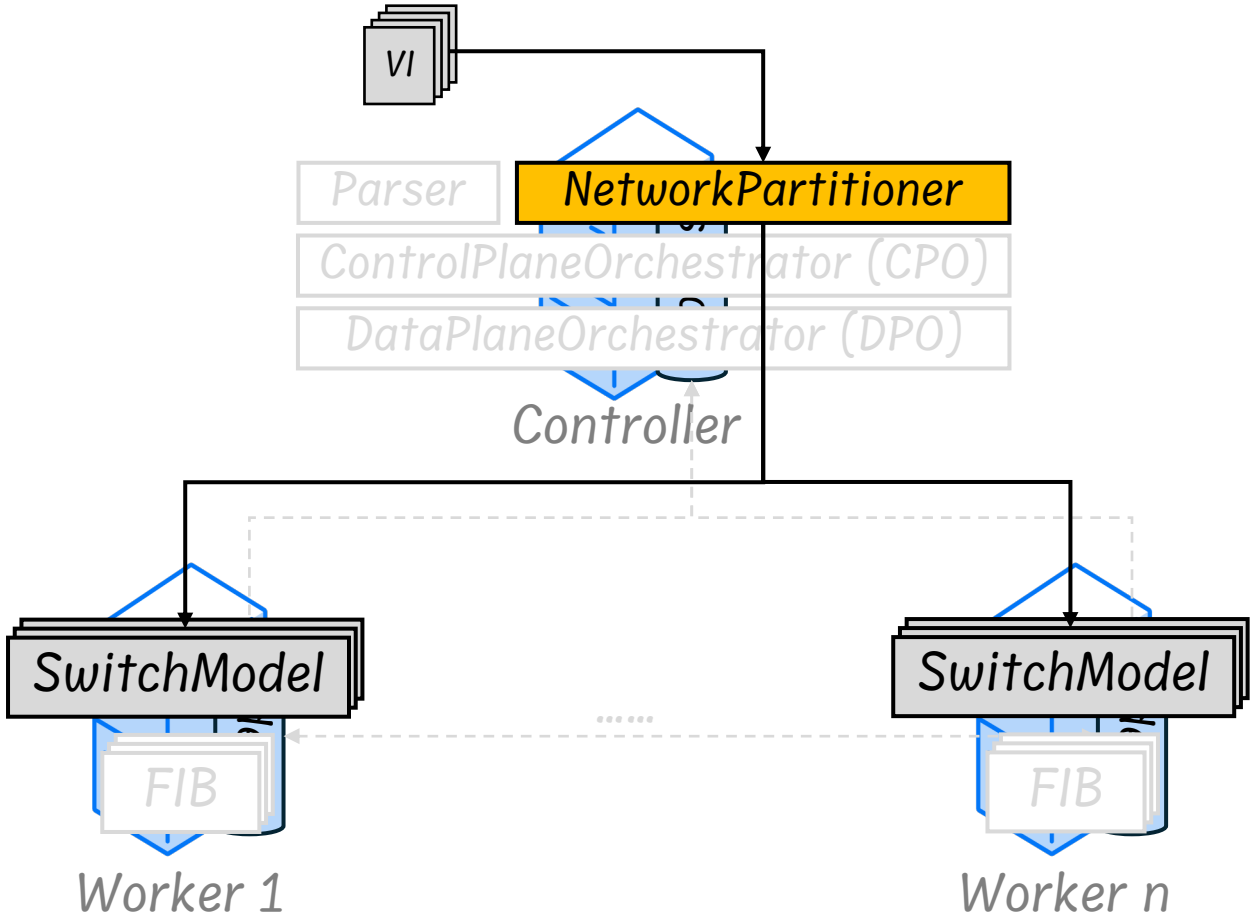
Step1: Parse vendor-specific configuration files into vendor-independent models

Step2: Partition the graph into n segments, one for each worker

Step3: Distributed Control Plane Simulation

Step4: Distributed Data Plane Verification

Workflow of S2



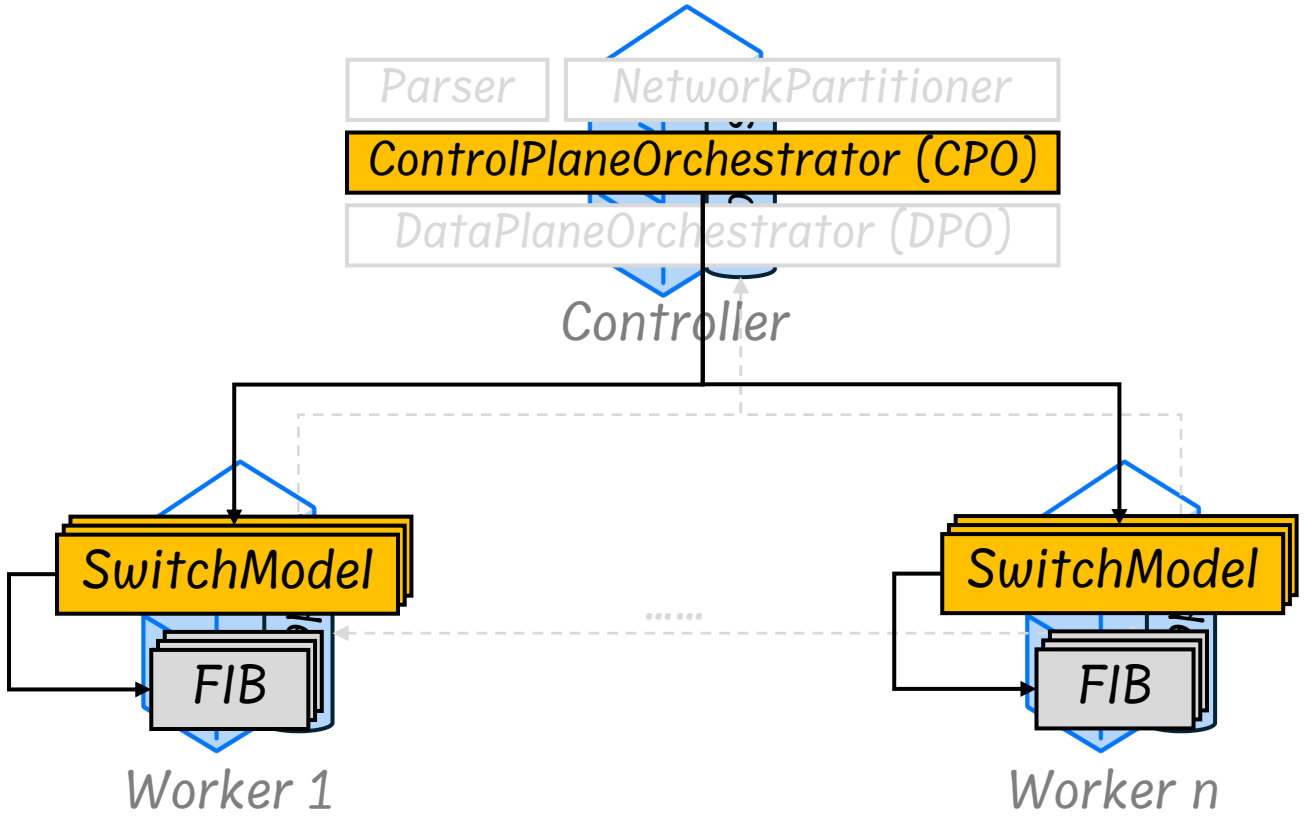
Step1: Parse vendor-specific configuration files into vendor-independent models

Step2: Partition the graph into n segments, one for each worker

Step3: Distributed Control Plane Simulation

Step4: Distributed Data Plane Verification

Workflow of S2



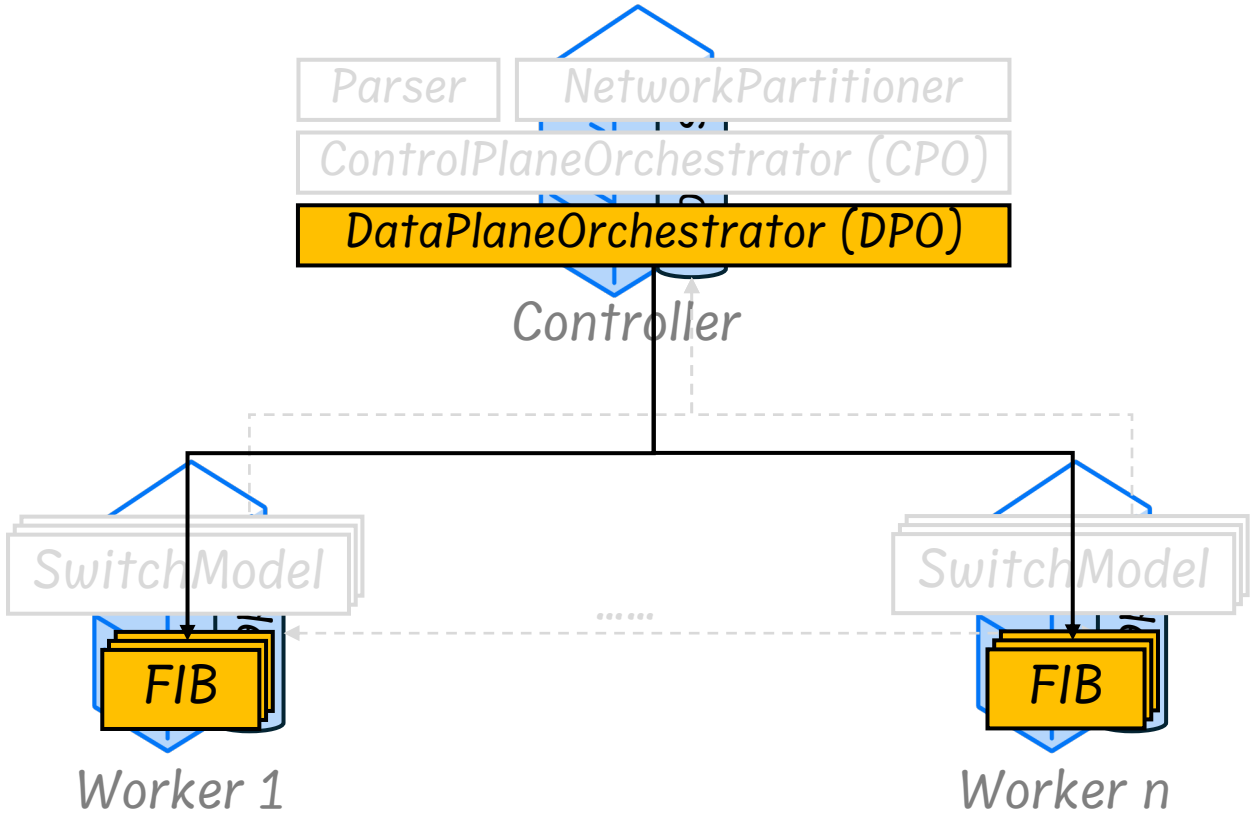
Step1: Parse vendor-specific configuration files into vendor-independent models

Step2: Partition the graph into n segments, one for each worker

Step3: Distributed Control Plane Simulation

Step4: Distributed Data Plane Verification

Workflow of S2



Step1: Parse vendor-specific configuration files into vendor-independent models

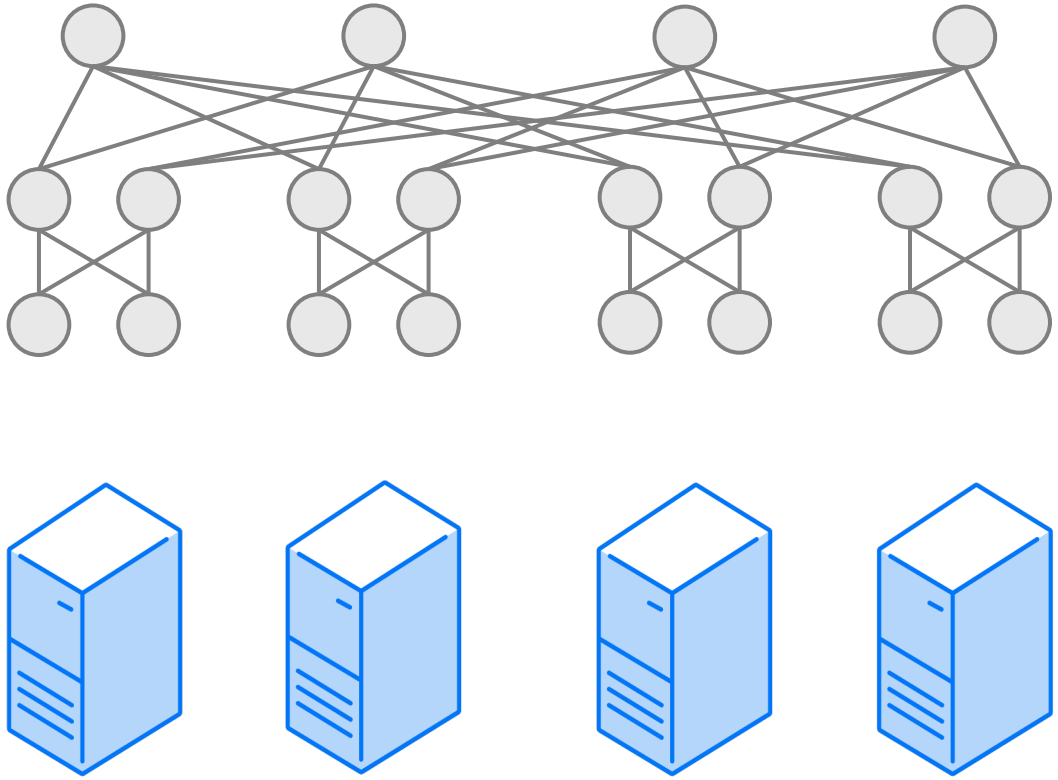
Step2: Partition the graph into n segments, one for each worker

Step3: Distributed Control Plane Simulation

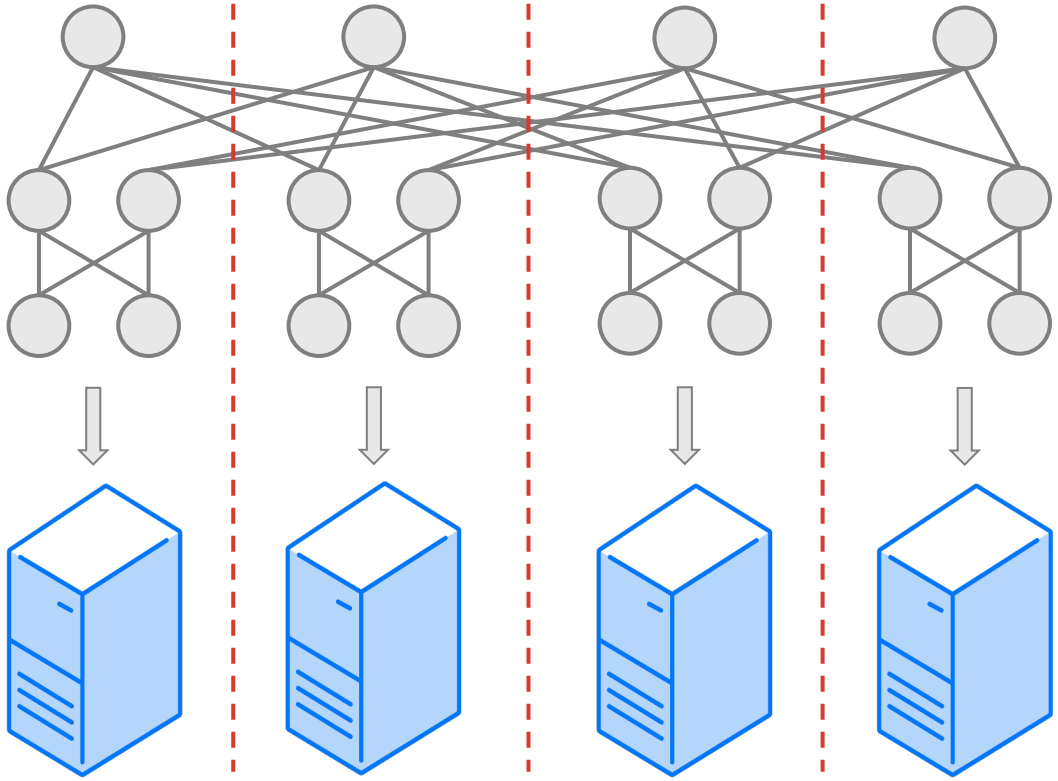
Step4: Distributed Data Plane Verification

Network Partition

Network Partition



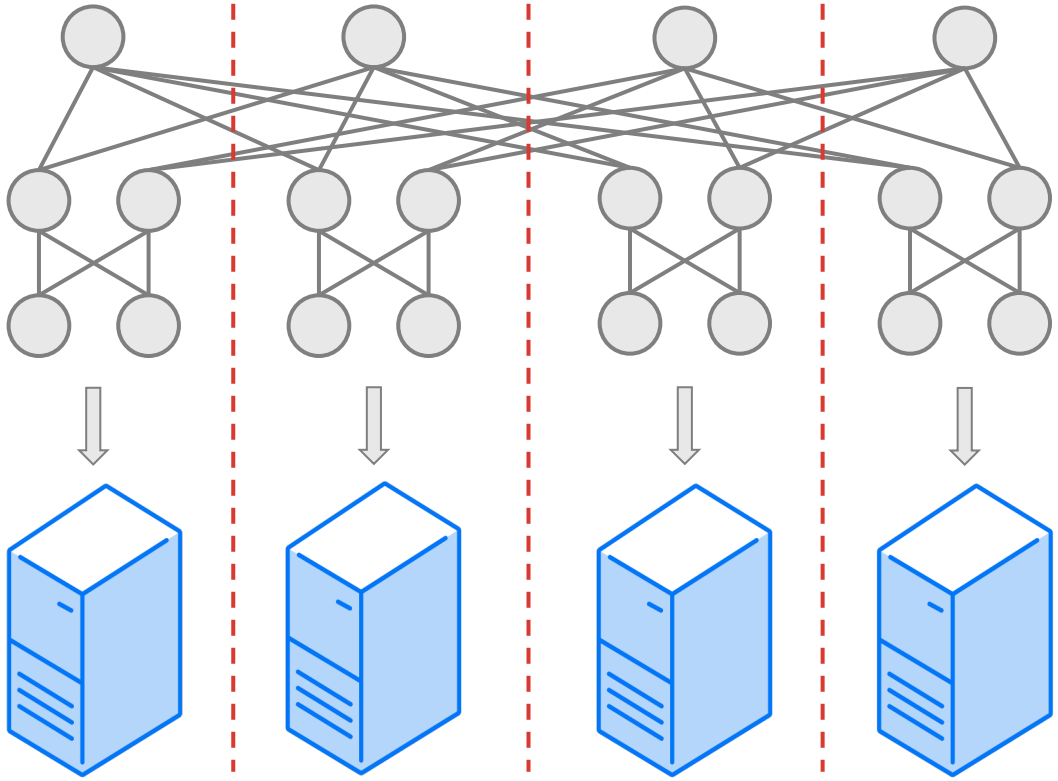
Network Partition



How to partition the network?

1. *Balanced worker workloads*
2. *Minimal cross-worker communication cost*

Network Partition



How to partition the network?

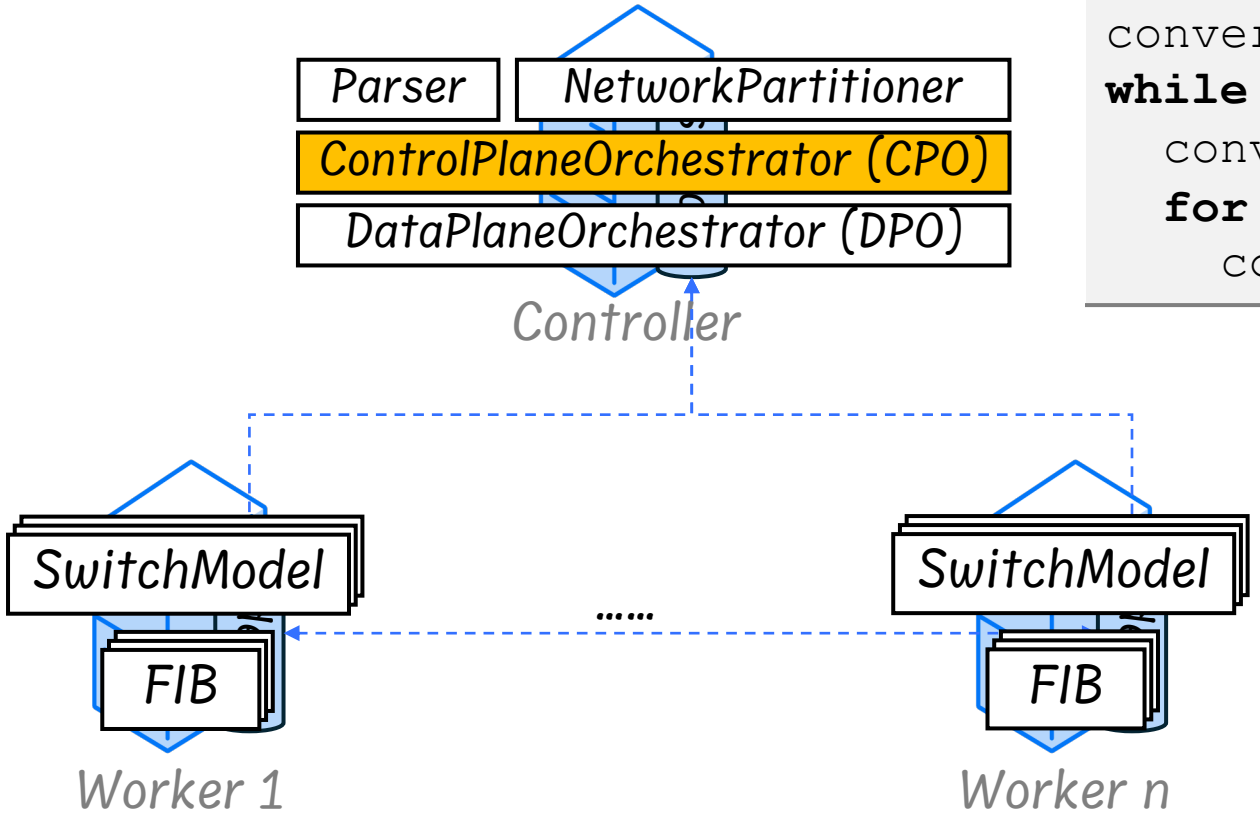
1. Balanced worker workloads
2. Minimal cross-worker communication cost

Number of routes

Graph partitioning

Distributed Control Plane Simulation

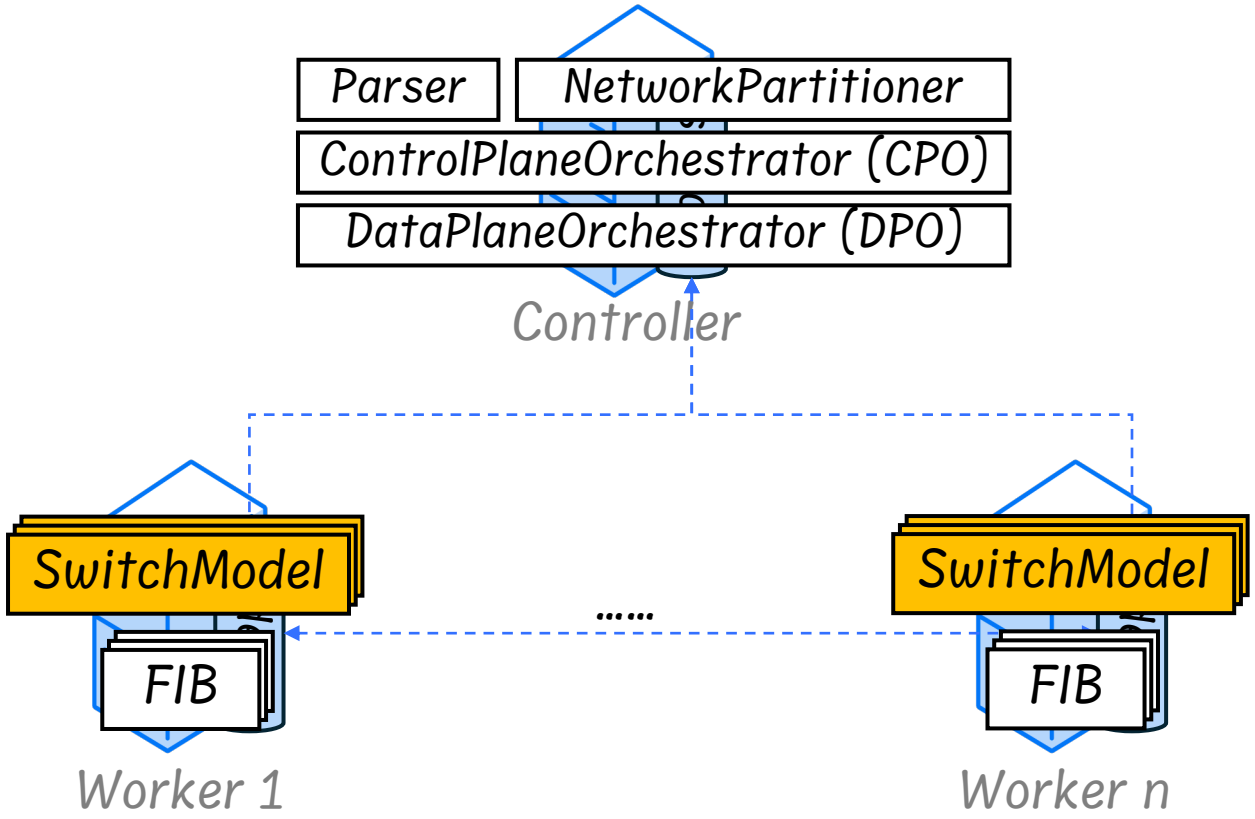
Distributed Control Plane Simulation




```
Fixpoint()
```

```
converged <- false;
while !converged
  converged <- true;
  for w in workers do
    converged <- w.Execute() & converged;
```

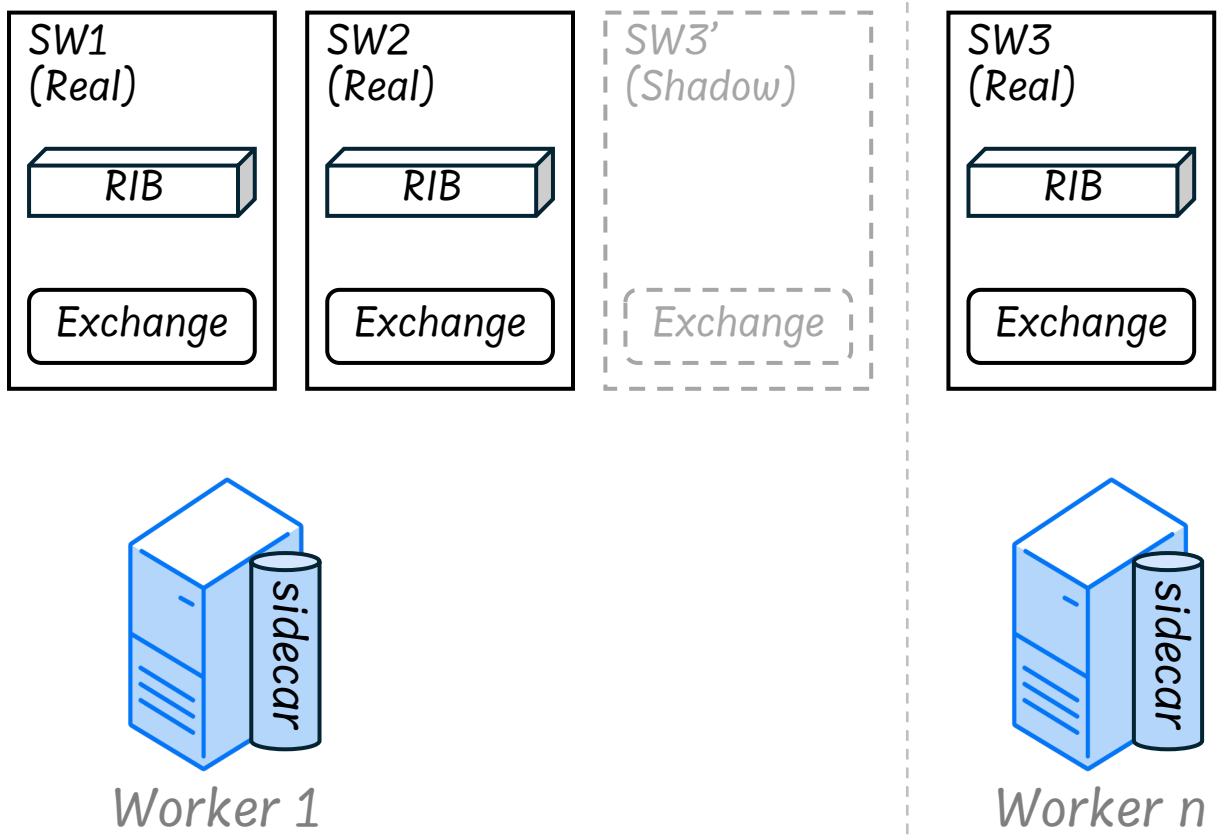
Distributed Control Plane Simulation



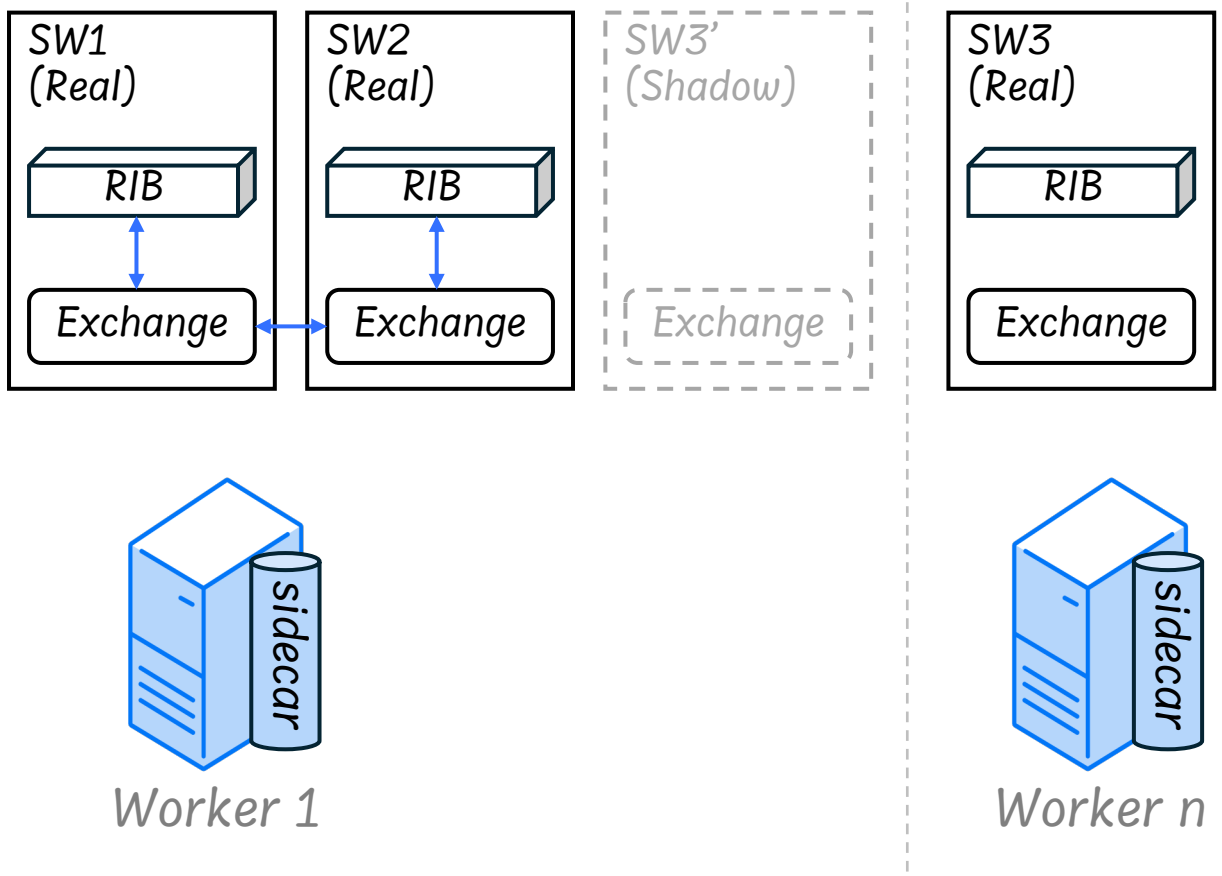

```
Execute()
```

```
for node in w.nodes do
  for neighbor in node.neighbors do
    routes <- neighbor.Exchange(node)
    node.updateRIB(routes)
```

Distributed Control Plane Simulation

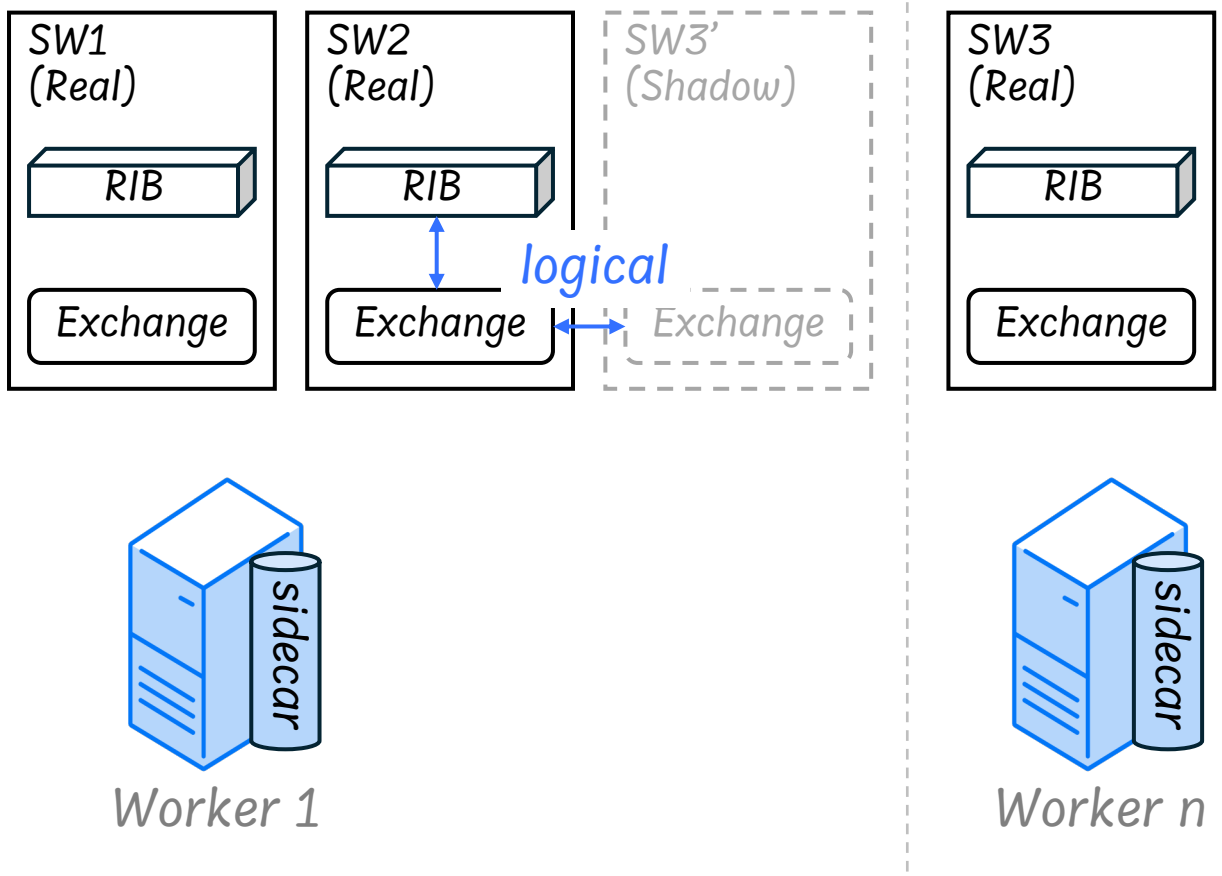


Distributed Control Plane Simulation



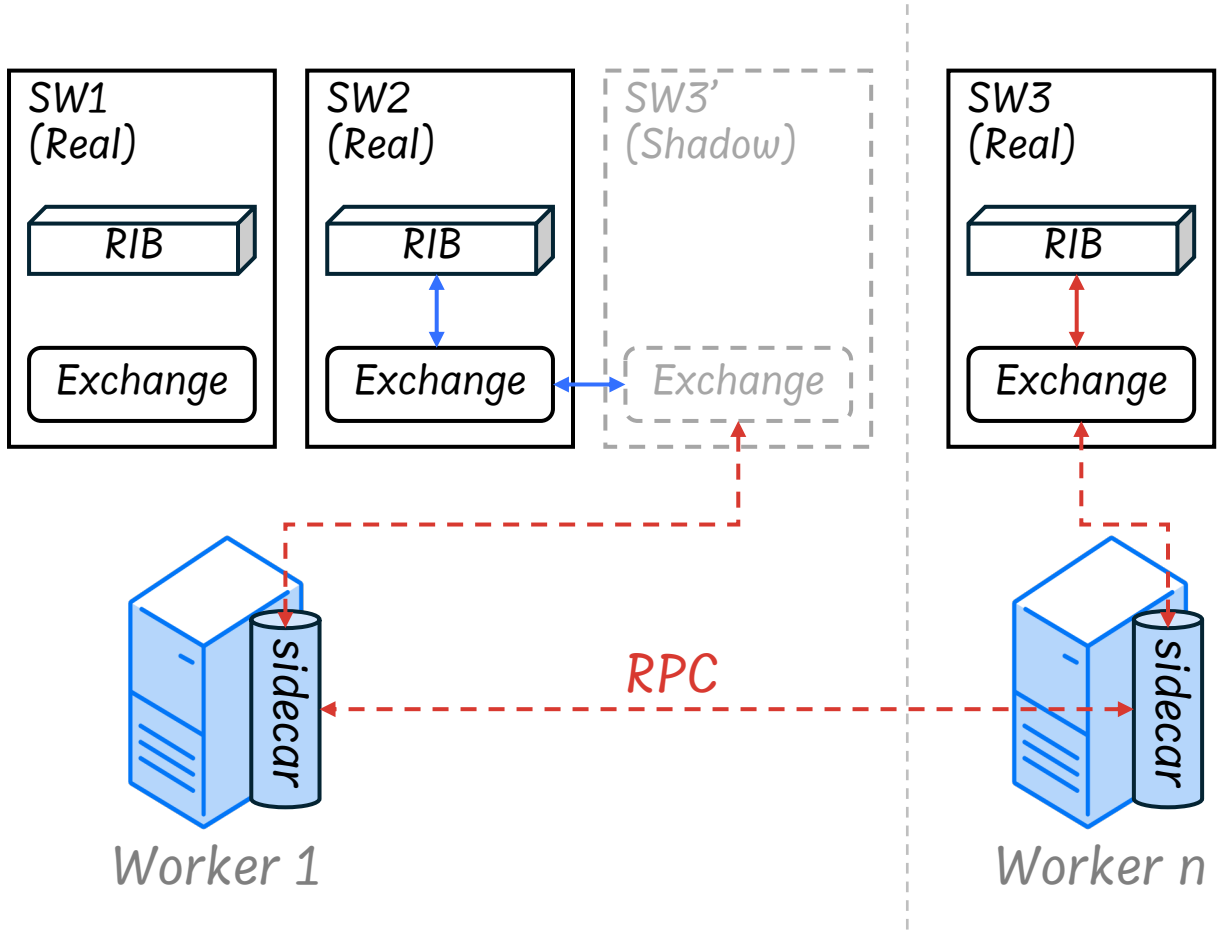
Route exchange between SW1 and SW2

Distributed Control Plane Simulation



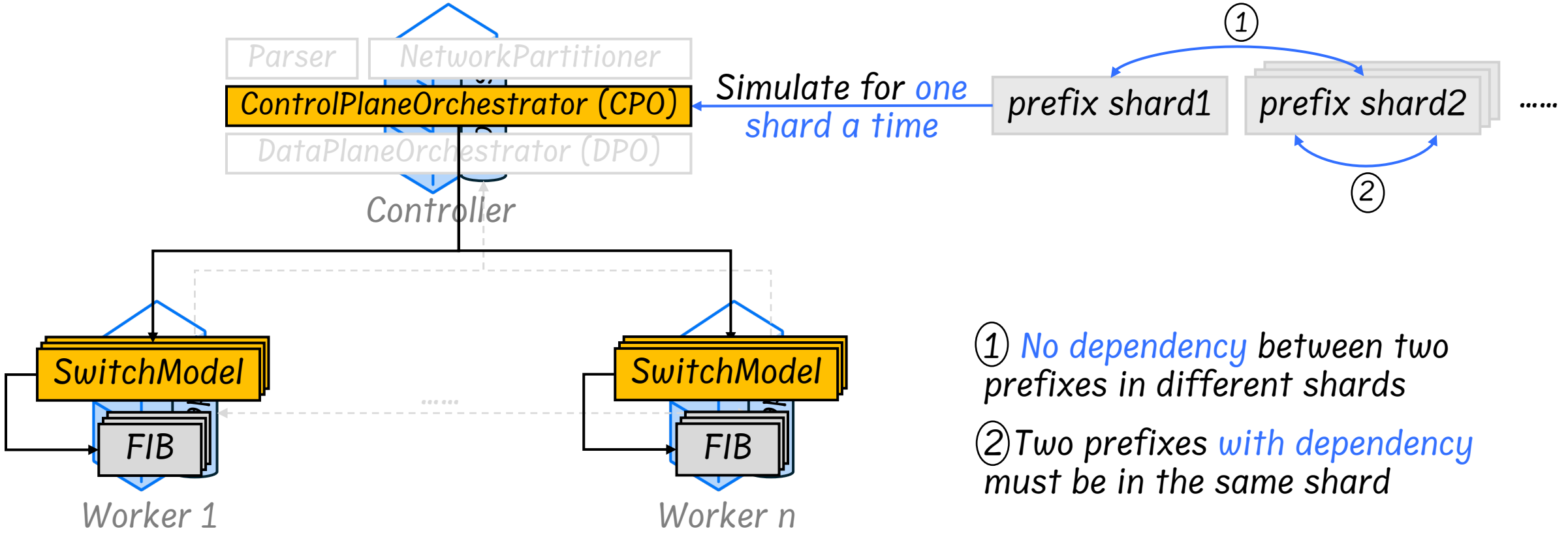
Route exchange between SW2 and SW3

Distributed Control Plane Simulation



Route exchange between SW2 and SW3

Optimization: prefix sharding

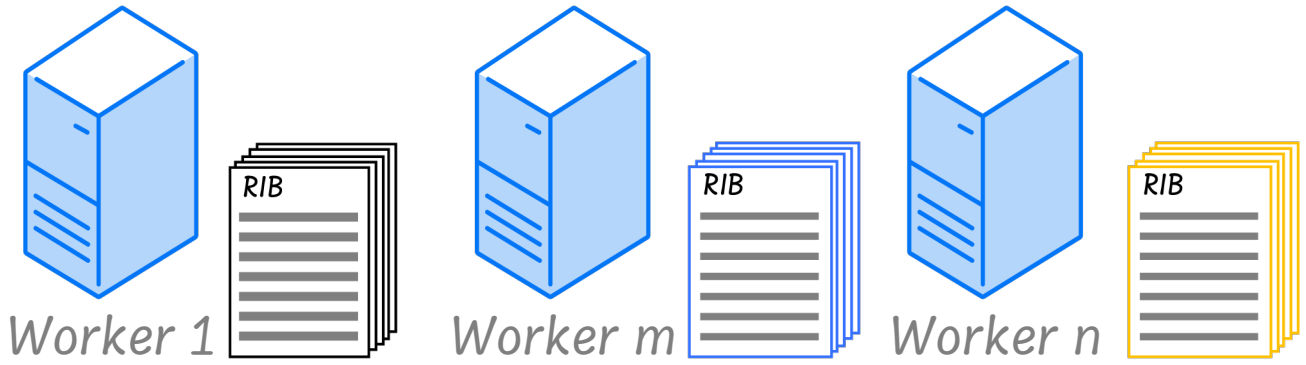


① No dependency between two prefixes in different shards

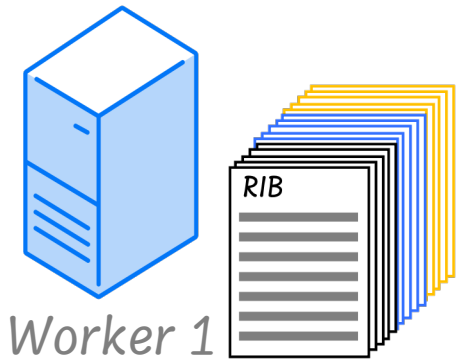
② Two prefixes with dependency must be in the same shard

Data Plane Verification

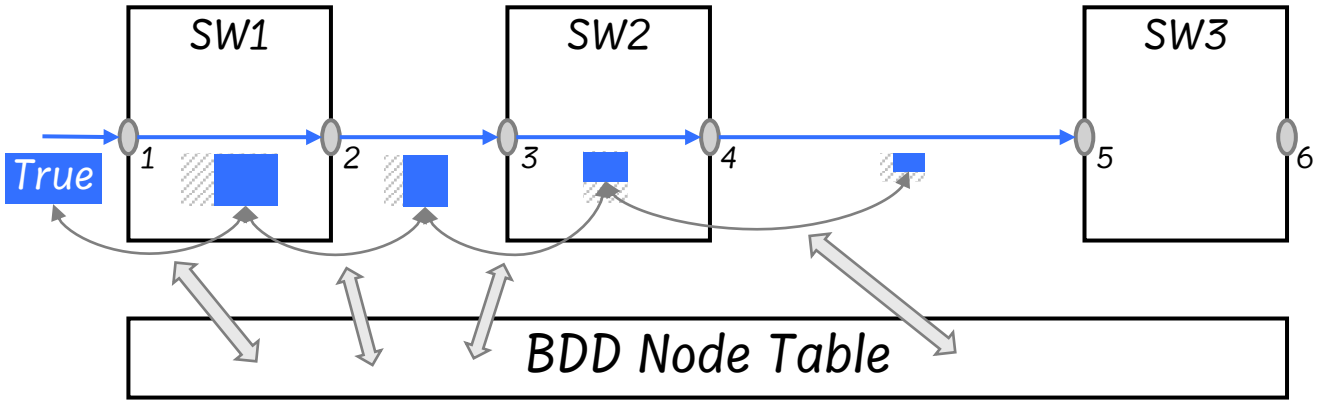
How to perform data plane verification?

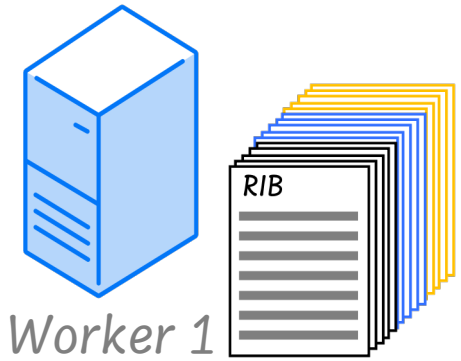


Gather RIBs onto one worker?

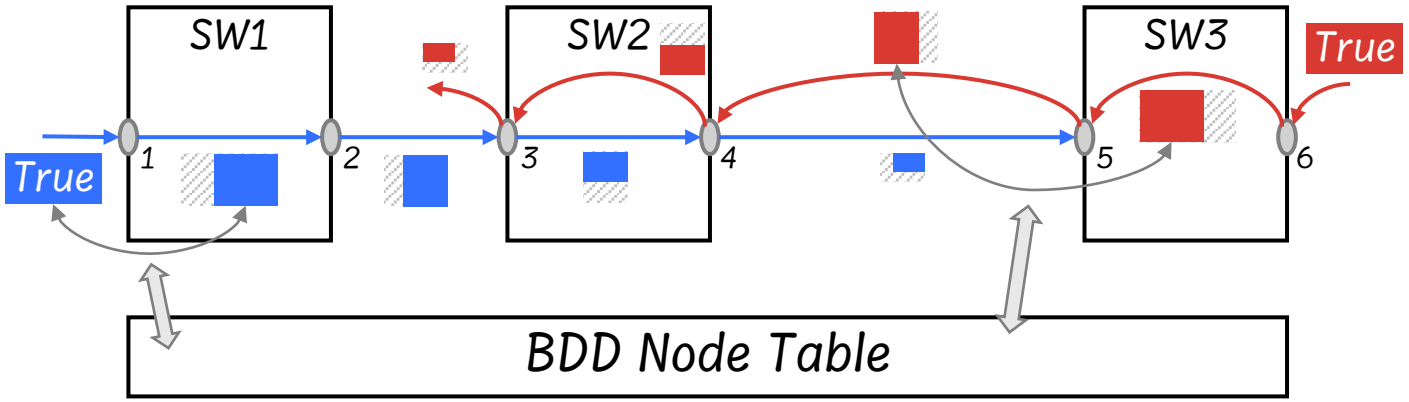


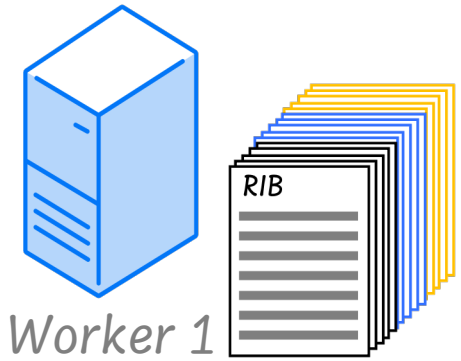
Perform centralized packet forwarding?





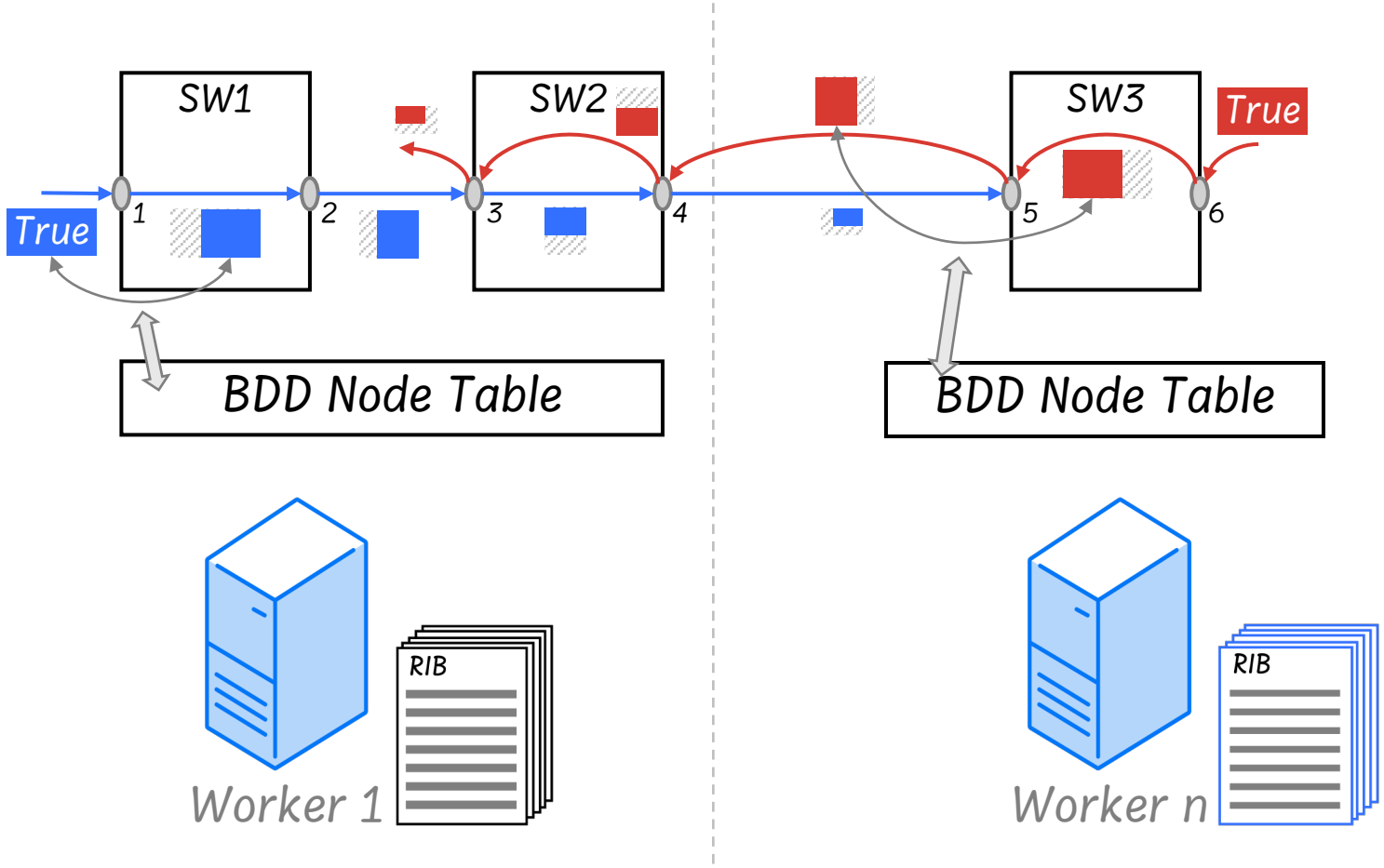
Gather routes and perform centralized DPV?





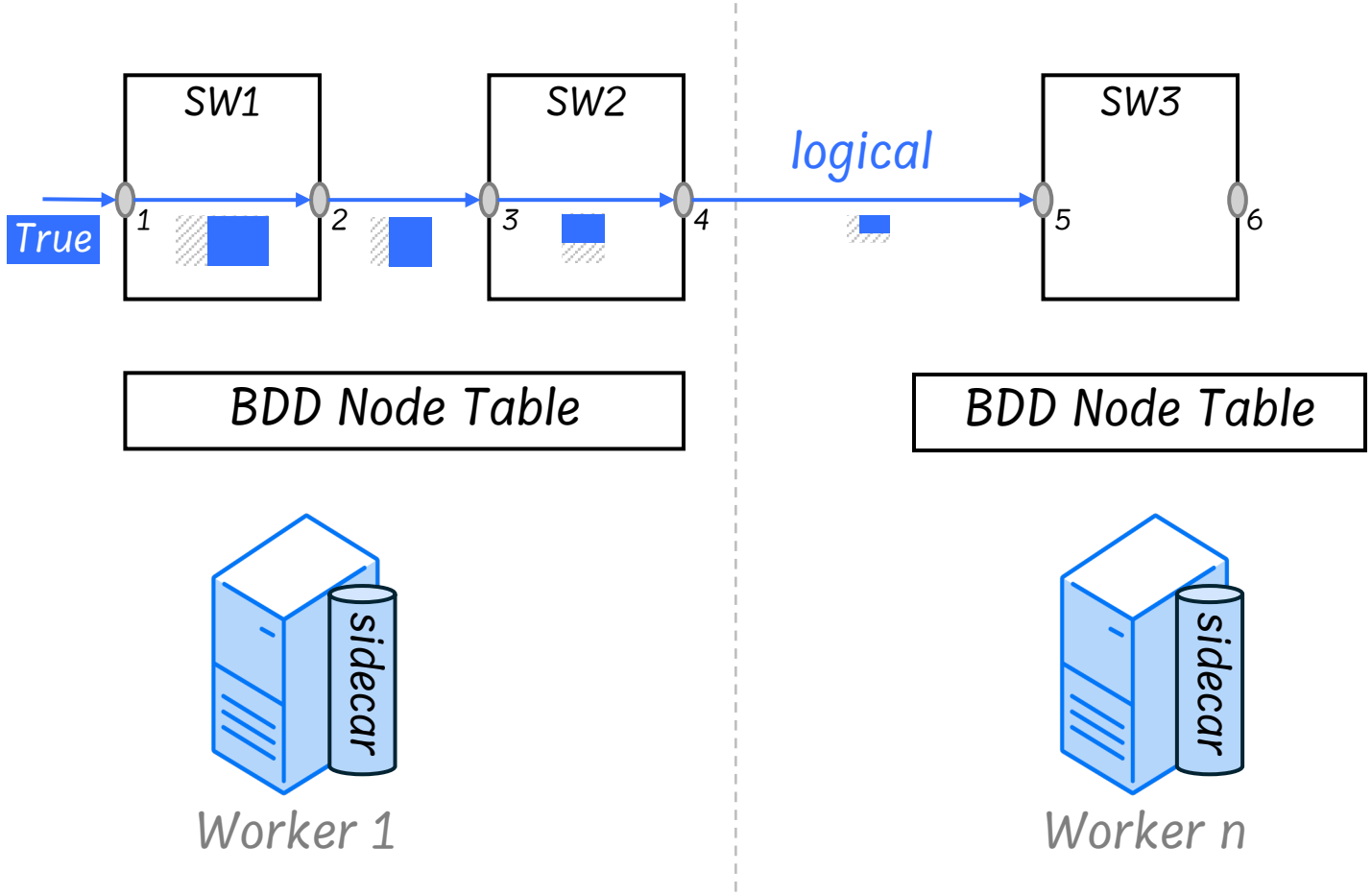
1. *Heavy* memory usage
2. *Sequential* packet forwarding

Perform *distributed* packet forwarding!

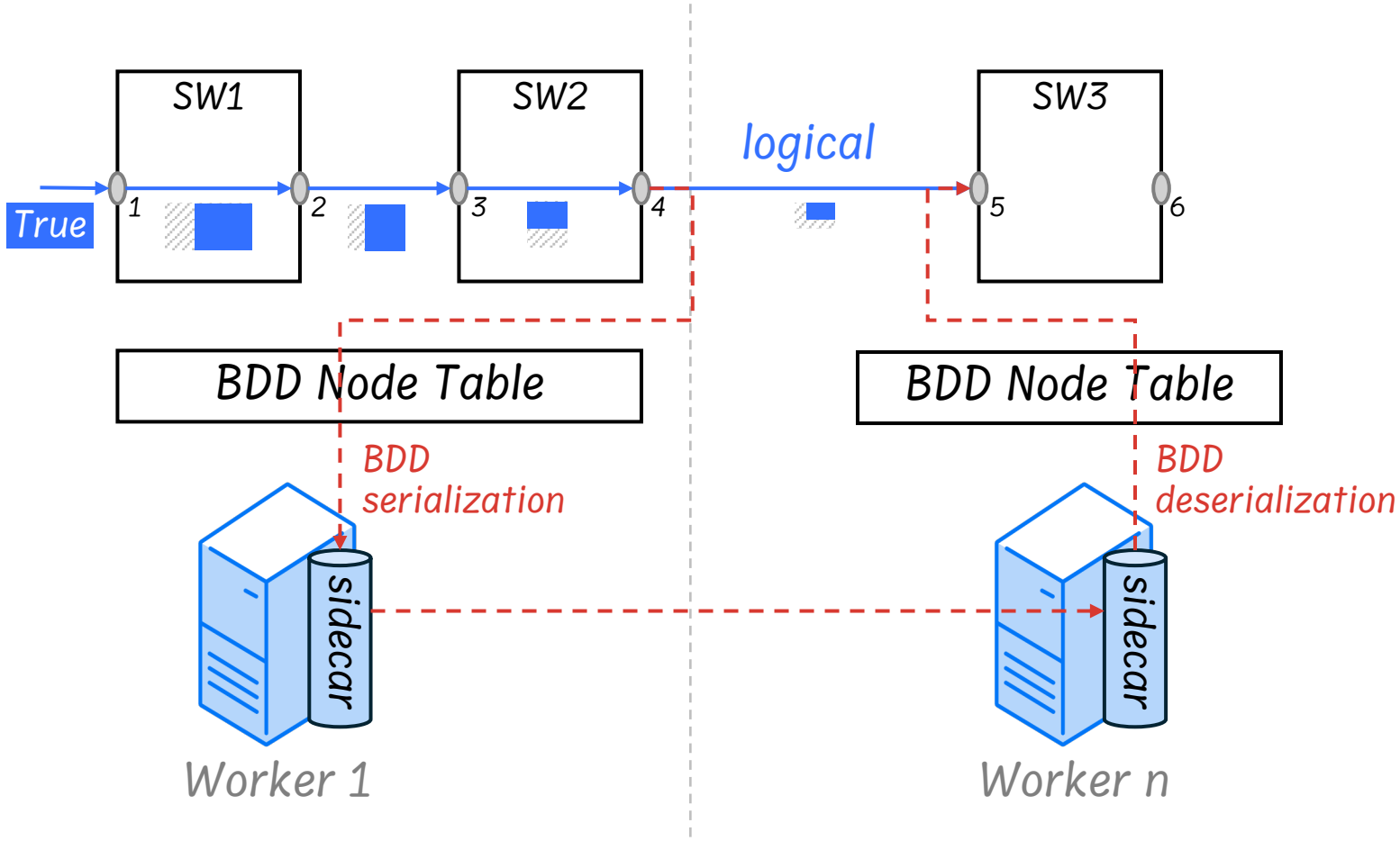


1. *Relatively light* memory usage
2. *Parallel* packet forwarding

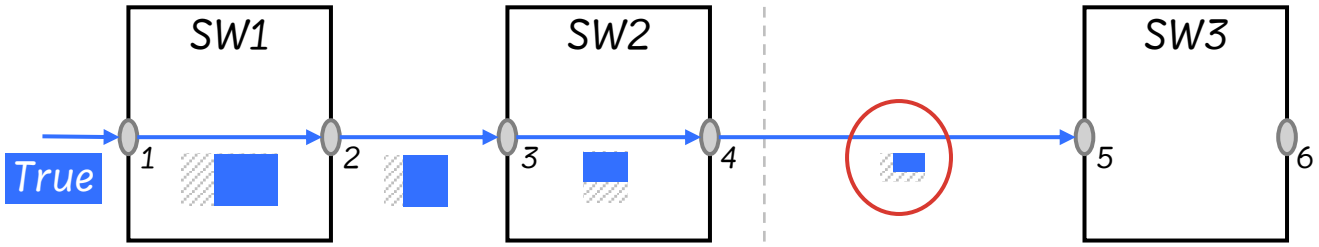
Distributed Data Plane Verification



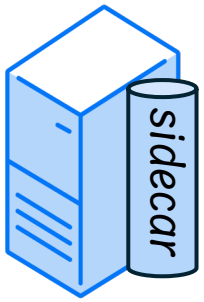
Distributed Data Plane Verification



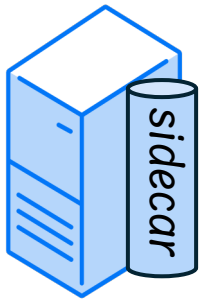
Distributed Data Plane Verification



All packets reachable from SW1 to SW3

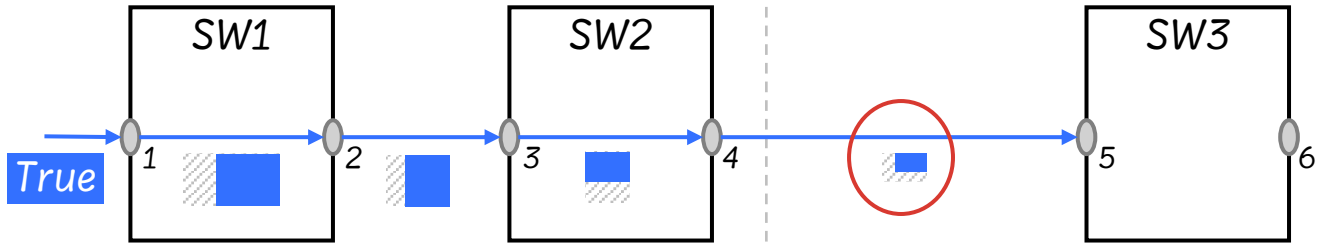


Worker 1

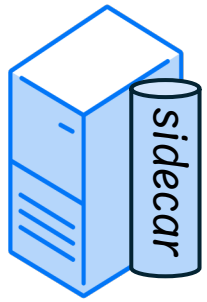


Worker n

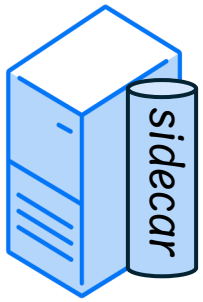
Distributed Data Plane Verification



All packets reachable from SW1 to SW3



Worker 1



Worker n

1. Reachability
2. Waypoint
3. Multi-path consistency
4. Loop/Blackhole

……

Evaluations

Implementation and Evaluation Setup

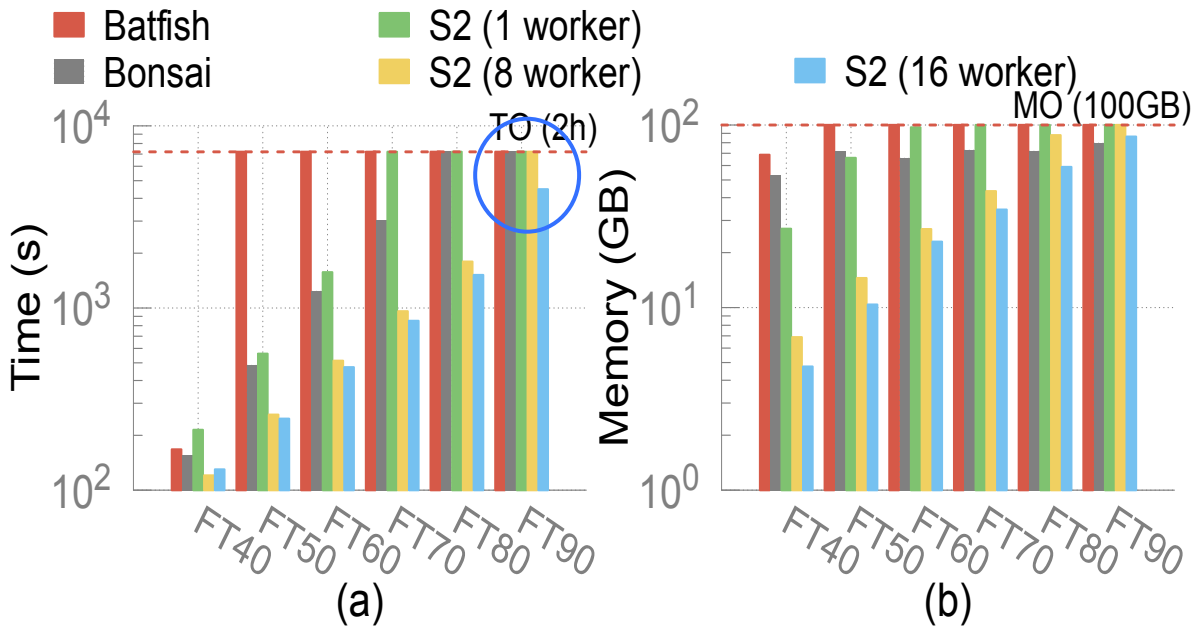
> We implement S2 on top of Batfish
~12K LOC of Java code, only ~500 LOC modification to Batfish

> We use both synthetic FatTrees (BGP, ECMP) and a real DCN (16k switches)

> We use five physical Linux servers: 64-core, 500GB RAM

> We divide each physical Linux server into four logical servers: 15-core, 100GM RAM

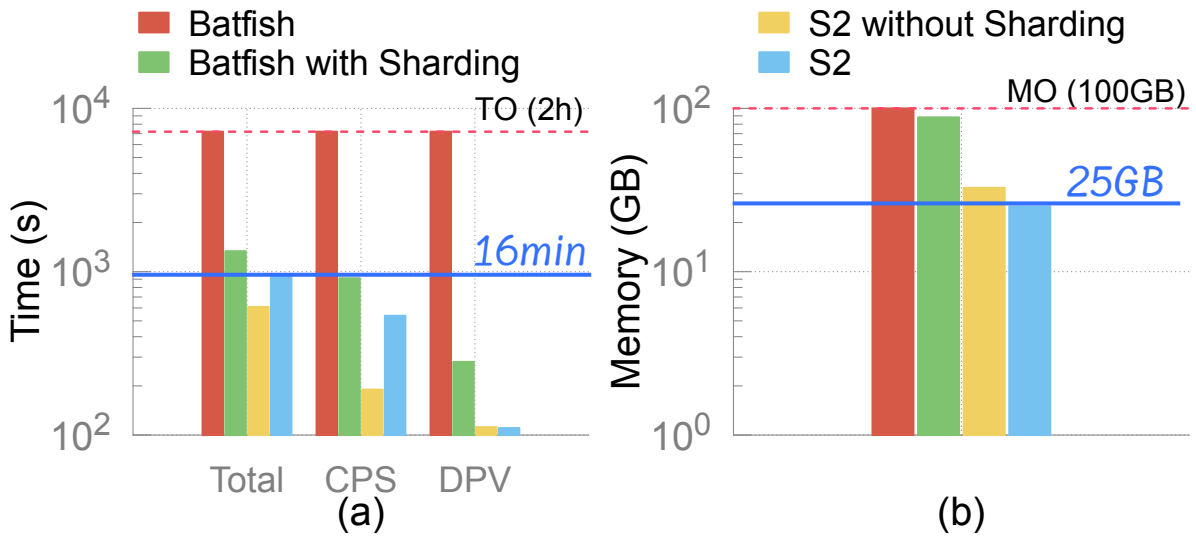
Results for synthetic FatTrees



> S2 (16 worker) is the only one that scales to FT90 (11K nodes)

> S2 scales out with more workers

Results for the real DCN



> The real DCN contains ~16K switches, producing ~200M IPv4 routes

> S2 finishes with ~16 minutes and ~25GB memory

Summary

- Hyper-scale networks are error prone, verifying them is compute- and memory-intensive
- Existing verifiers choose to "scale-up"
- We designed S2, "scale-out" on a distributed architecture
- We built S2 on top of Batfish, it scales to networks with >10K switches within 2 hours
- We will open-source S2 soon

Happy to take your questions