Pyrrha: Congestion-Root-Based Flow Control to Eliminate Head-of-Line Blocking in Datacenter

Kexin Liu*, Zhaochen Zhang*, Chang Liu, Yizhi Wang, Vamsi Addanki, Stefan Schmid, Qingyue Wang, Wei Chen, Xiaoliang Wang, Jiaqi Zheng, Wenhao Sun, Tao Wu, Ke Meng, Fei Chen, Weiguang Wang, Bingyang Liu, Wanchun Dou, Guihai Chen, **Chen Tian**

CC is Falling for Transient Congestion

Trend1: The high port bandwidth allows sending out more flows within **the first RTT**, even before congestion control could step in.

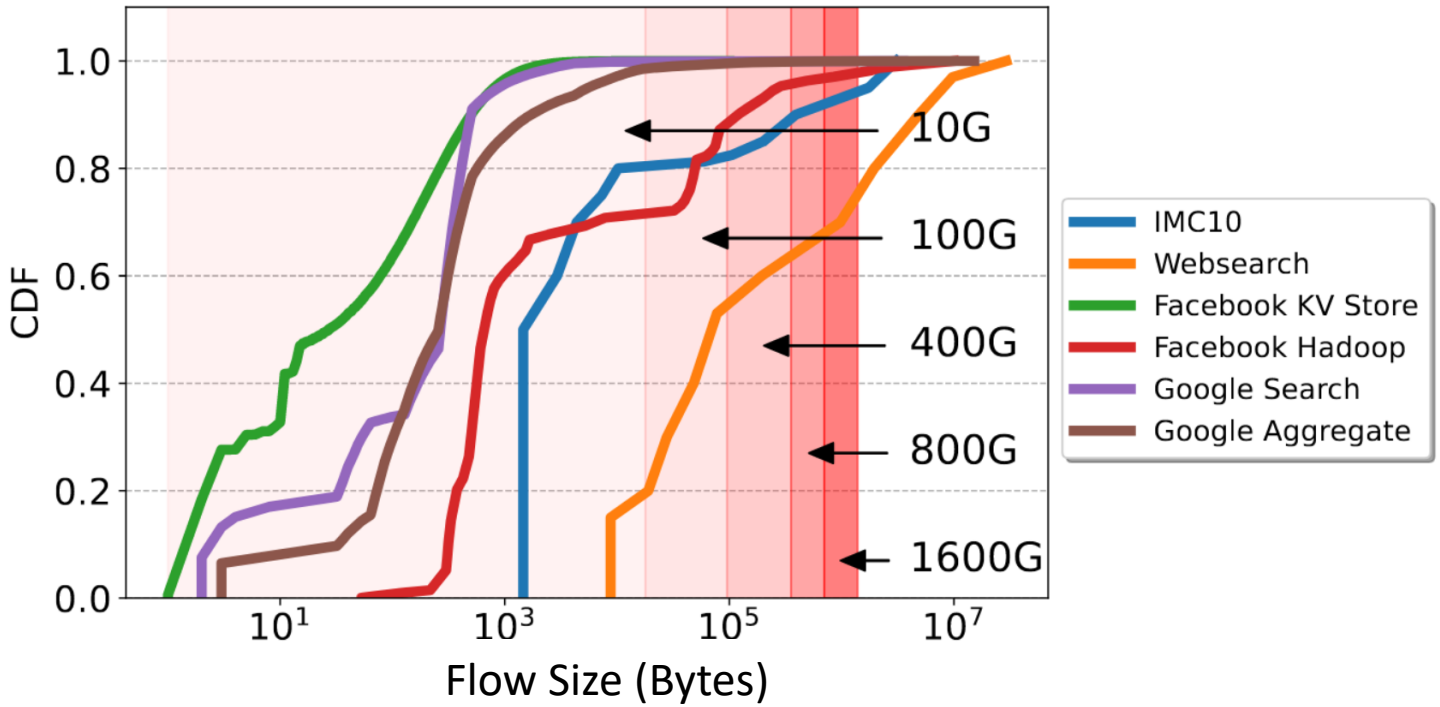
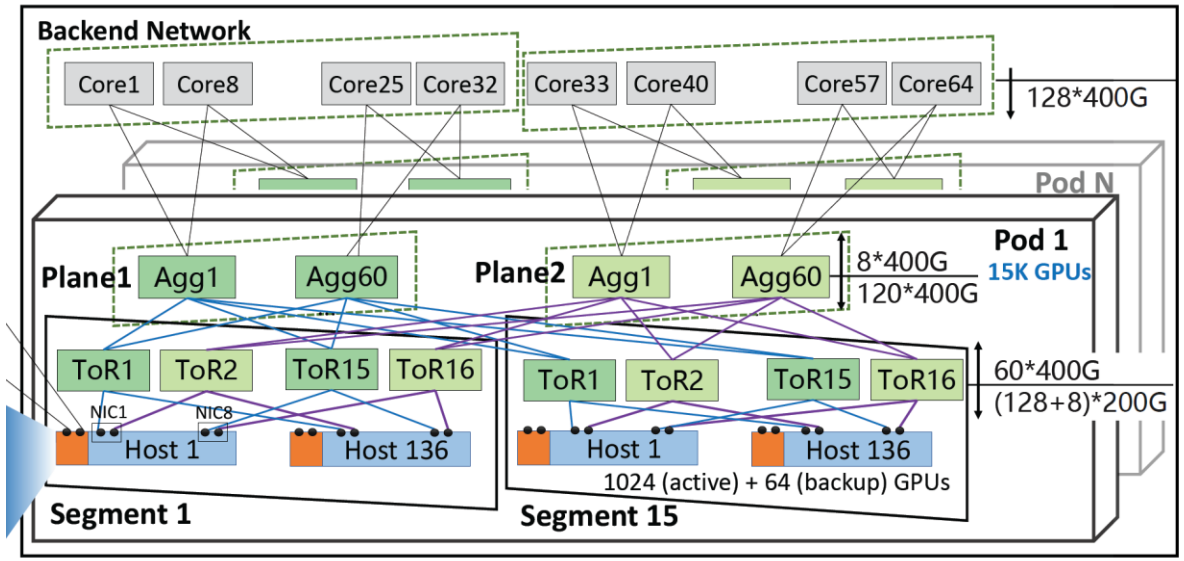


Figure source: Harmony@NSDI '24

CC is Falling for Transient Congestion

Trend2: The growing scale of DCN and the emerging workloads (e.g., distributed training) lead to **more bursty** traffic.



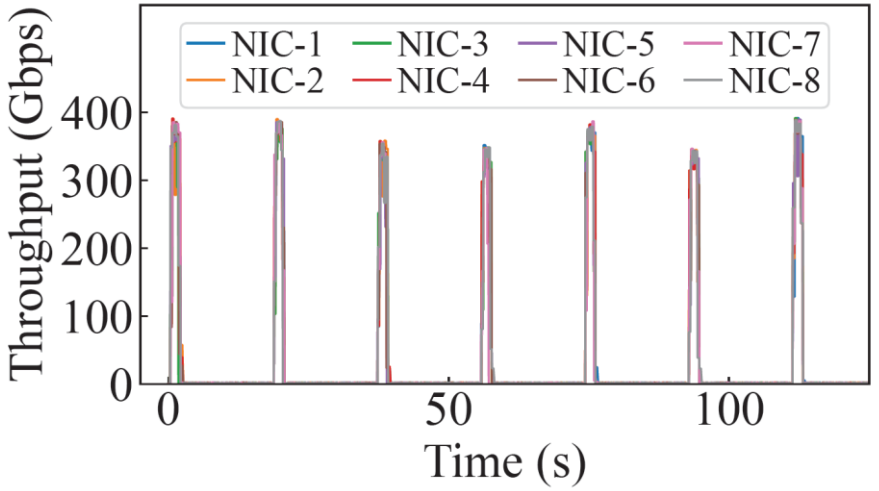


Figure source: HPN@Sigcomm '24

CC is Falling for Transient Congestion

Trend3: The decreasing buffer-to-bandwidth ratio calls for **faster** congestion management for transient congestion.

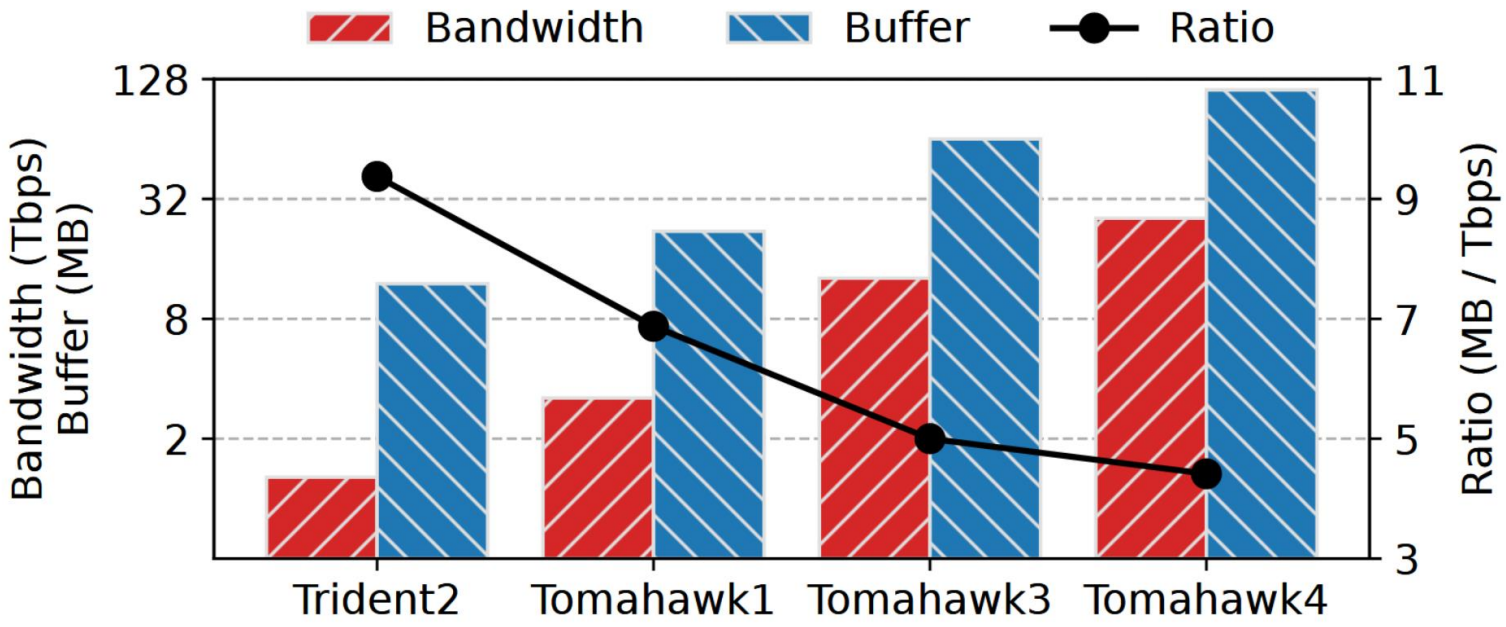


Figure source: PrioPlus@EuroSys '25

CC is Falling for Transient Congestion

Trend3: The decreasing buffer-to-bandwidth ratio calls for **faster** congestion management for transient congestion.

Data center networks need to manage **transient** congestion more quickly.

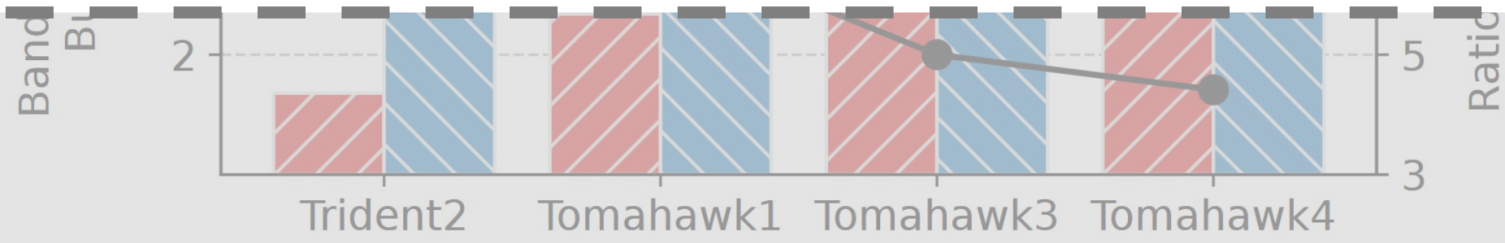
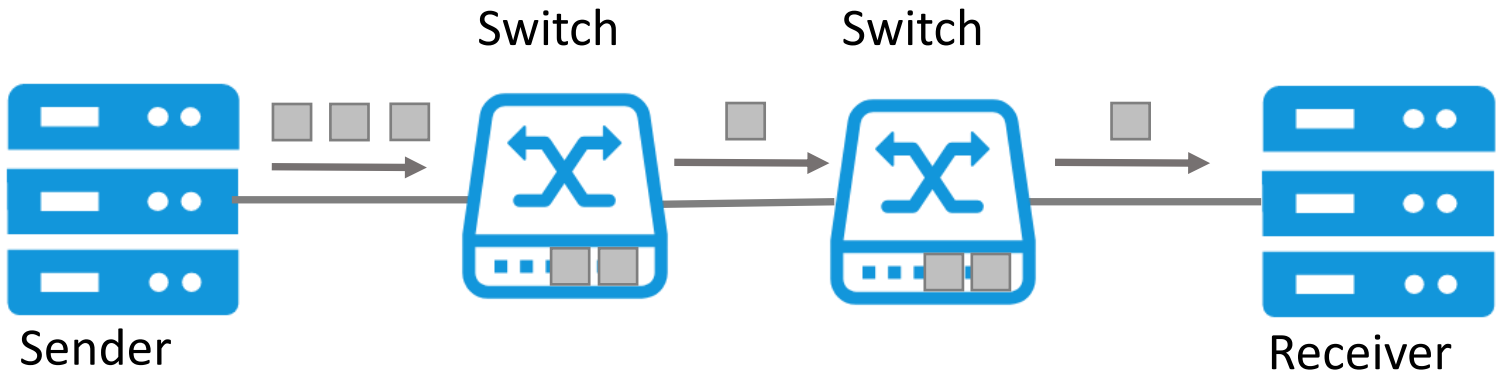


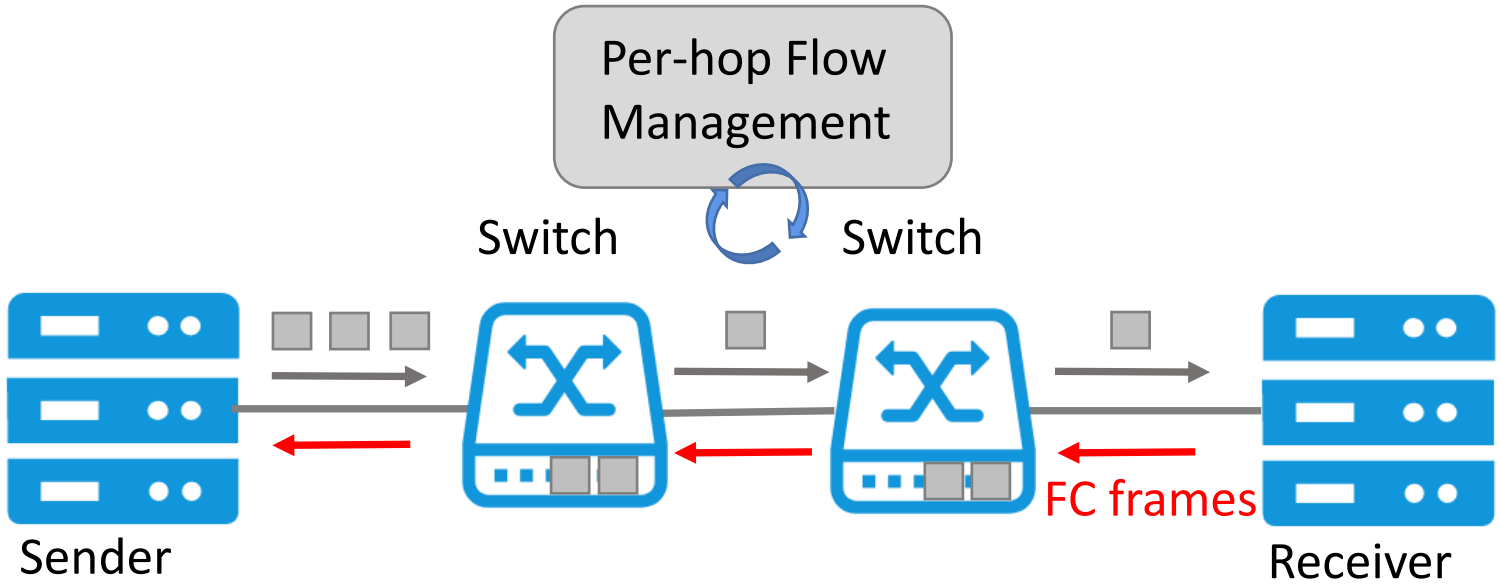
Figure source: PrioPlus@EuroSys '25

Our Vision for Data Center Congestion Management



Our Vision for Data Center Congestion Management

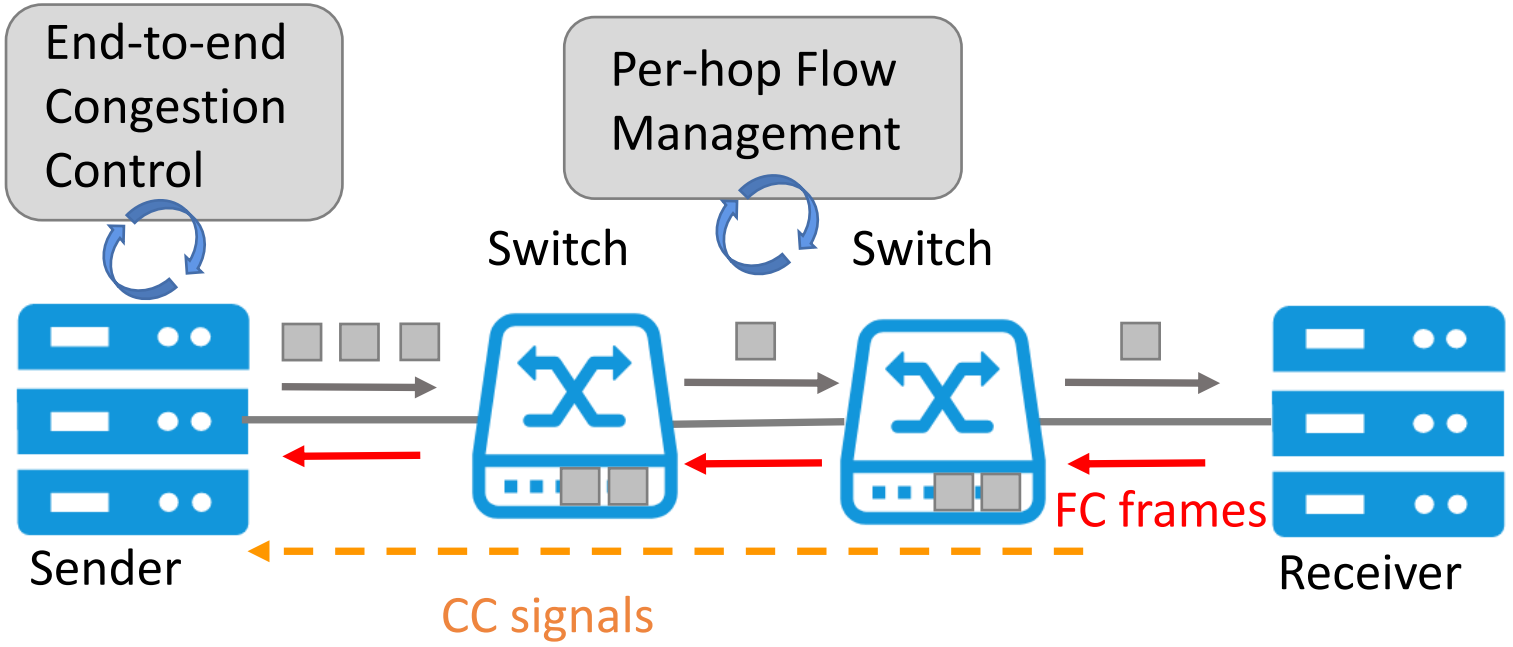
Per-hop FC handles transient congestion.



Our Vision for Data Center Congestion Management

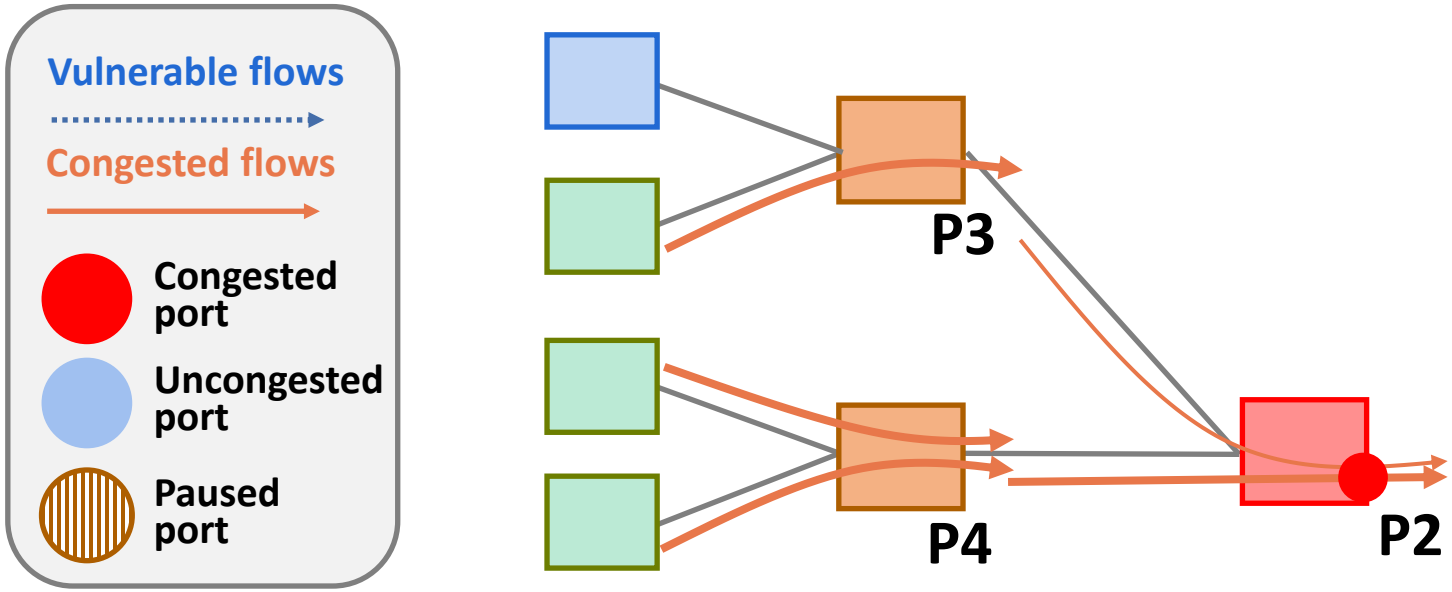
Per-hop FC handles transient congestion.

End-to-end CC handles persistent congestion.



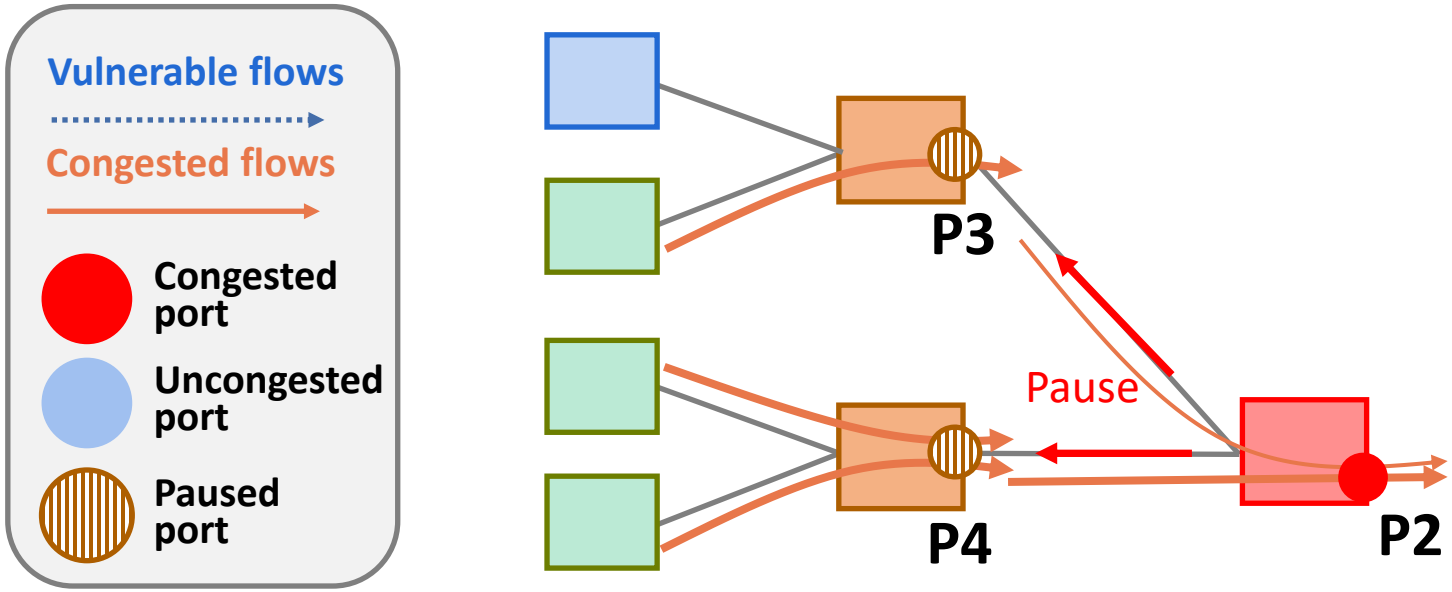
HOL Blocking Problem of Naive FC

Head-of-Line Blocking: a flow is paused innocently by the congested port that it does not pass through.



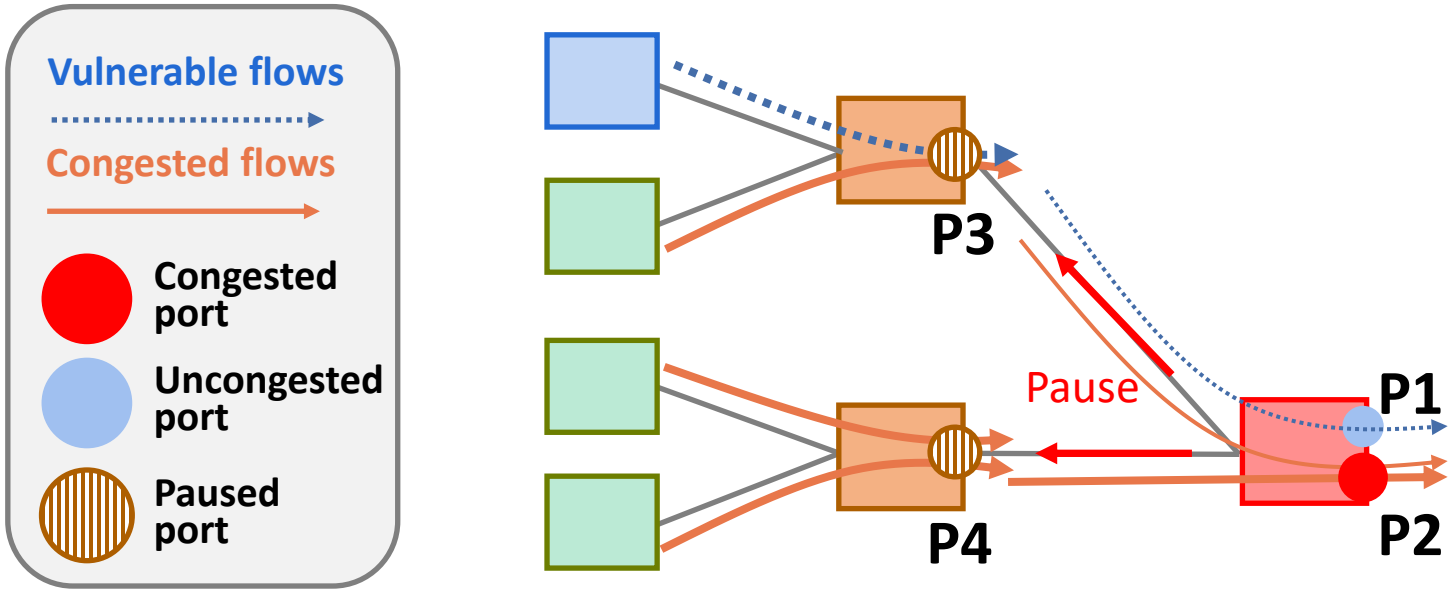
HOL Blocking Problem of Naive FC

Head-of-Line Blocking: a flow is paused innocently by the congested port that it does not pass through.



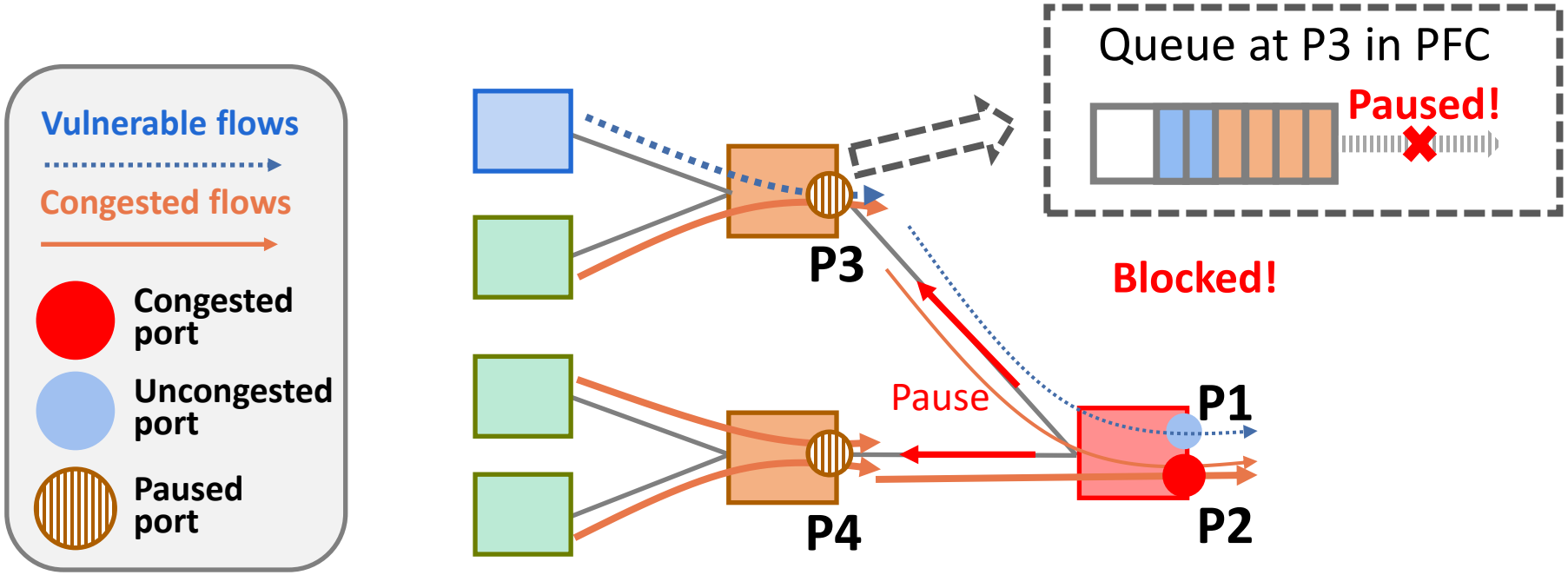
HOL Blocking Problem of Naive FC

Head-of-Line Blocking: a flow is paused innocently by the congested port that it does not pass through.



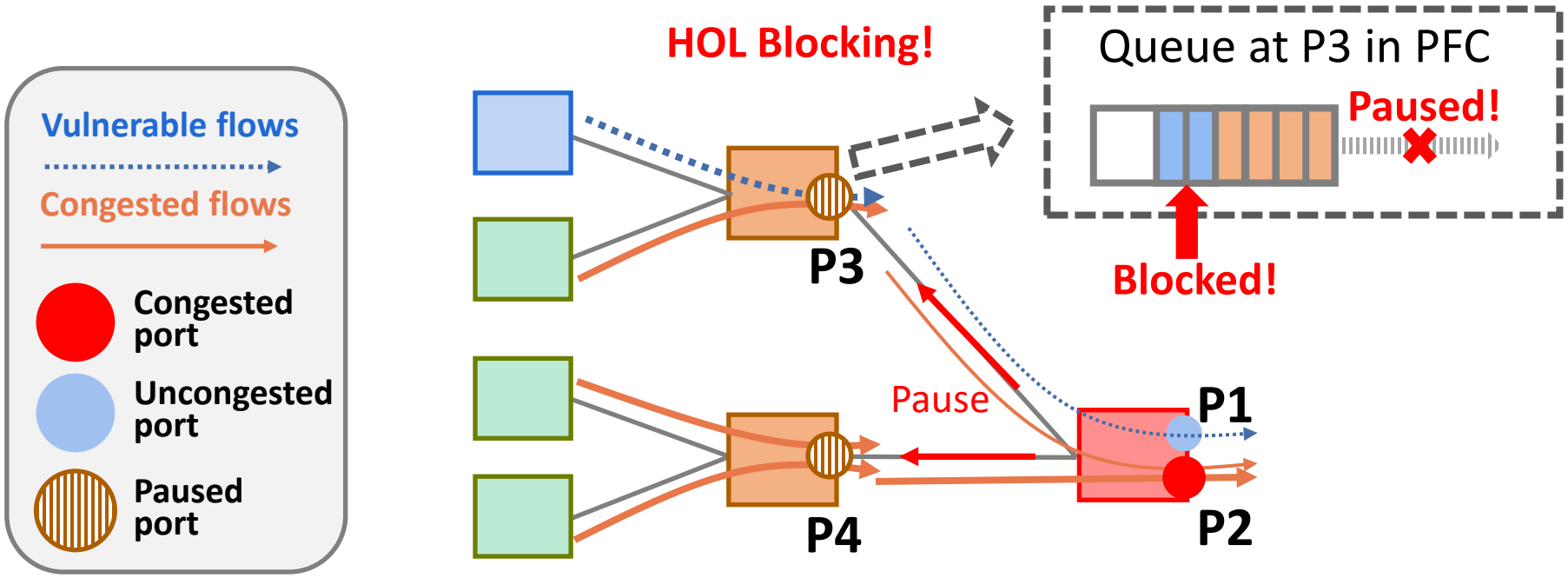
HOL Blocking Problem of Naive FC

Head-of-Line Blocking: a flow is paused innocently by the congested port that it does not pass through.



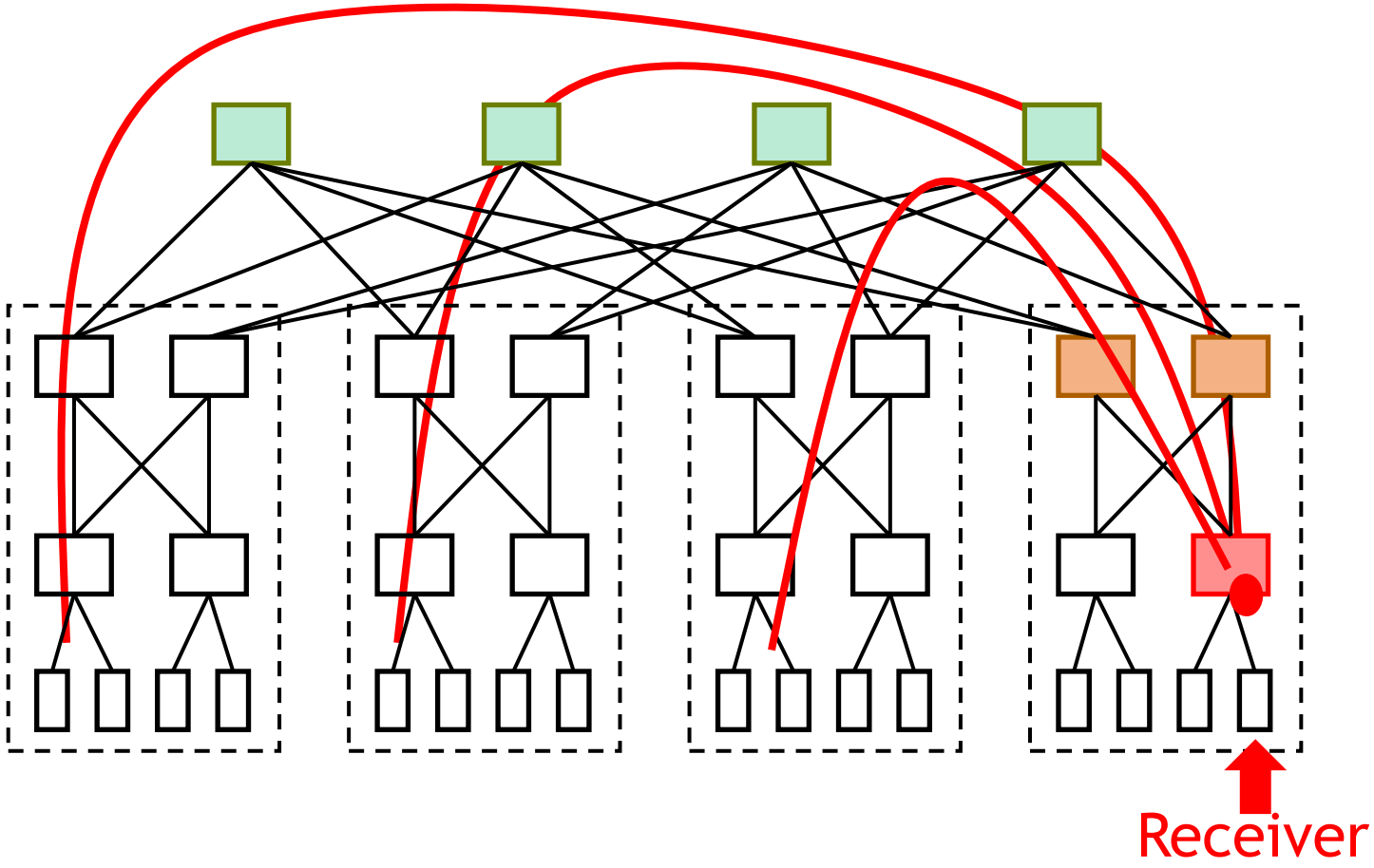
HOL Blocking Problem of Naive FC

Head-of-Line Blocking: a flow is paused innocently by the congested port that it does not pass through.



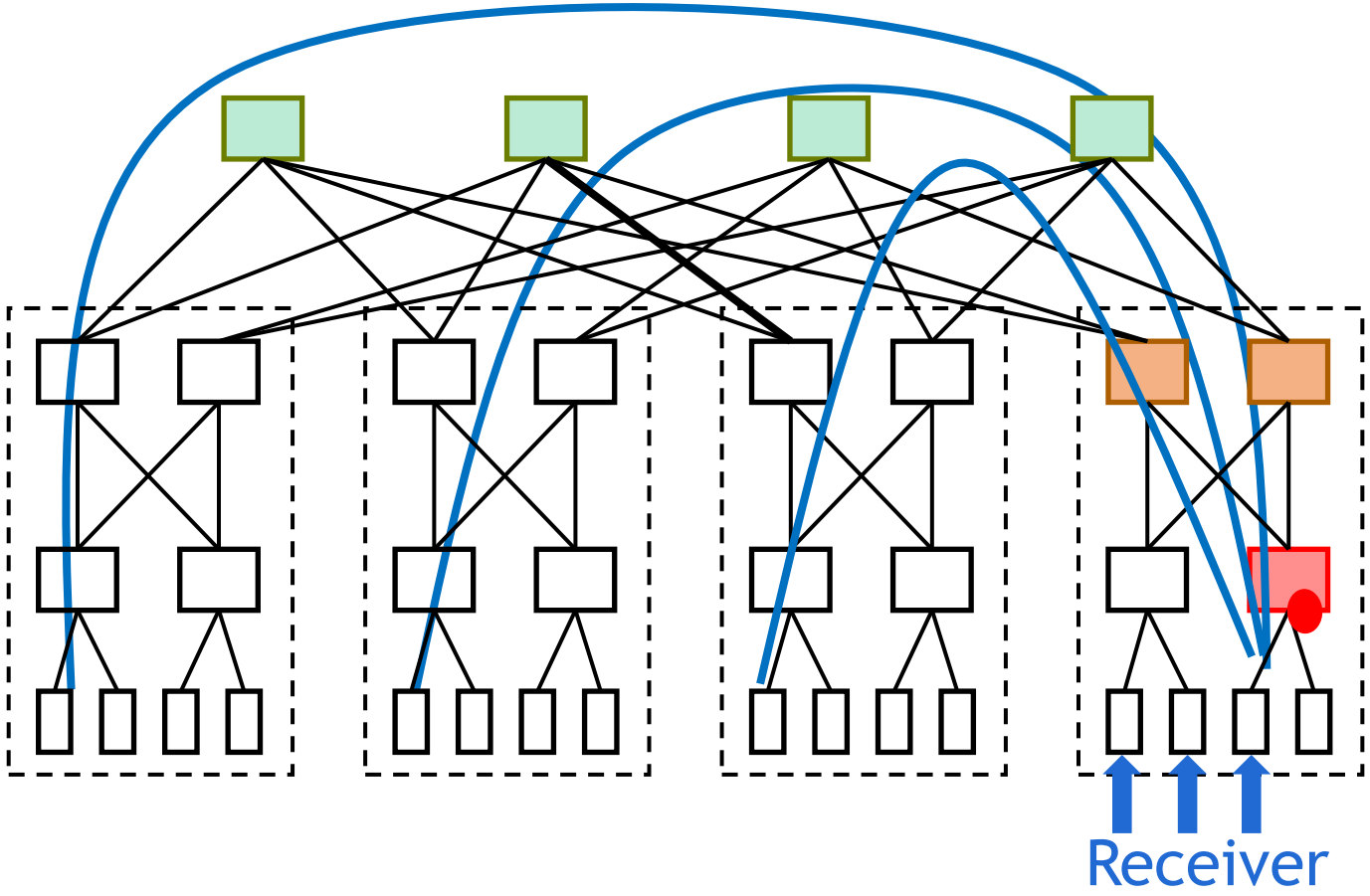
HOL Blocking Problem of Naive FC

Test workload:
continuous incast flows



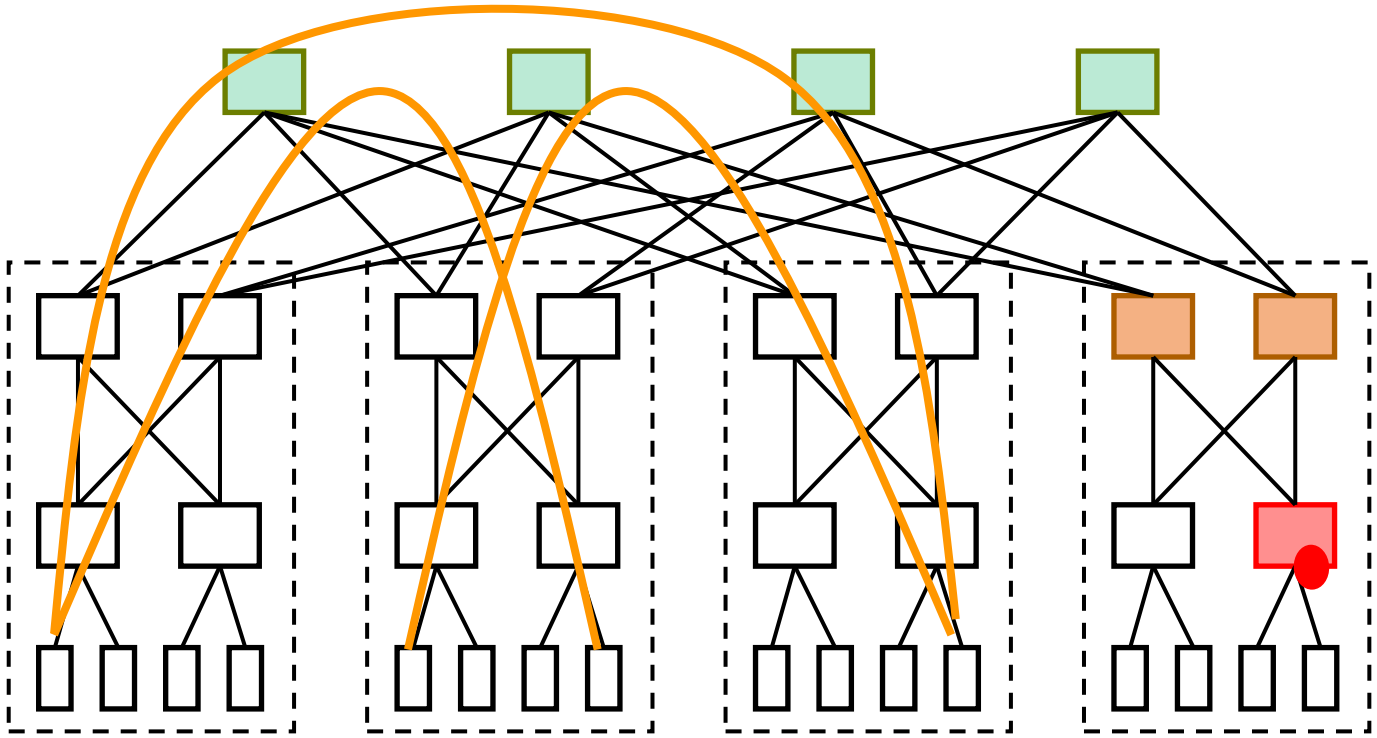
HOL Blocking Problem of Naive FC

Test workload:
continuous incast flows +
vulnerable flows with a total rate of 80 Gbps



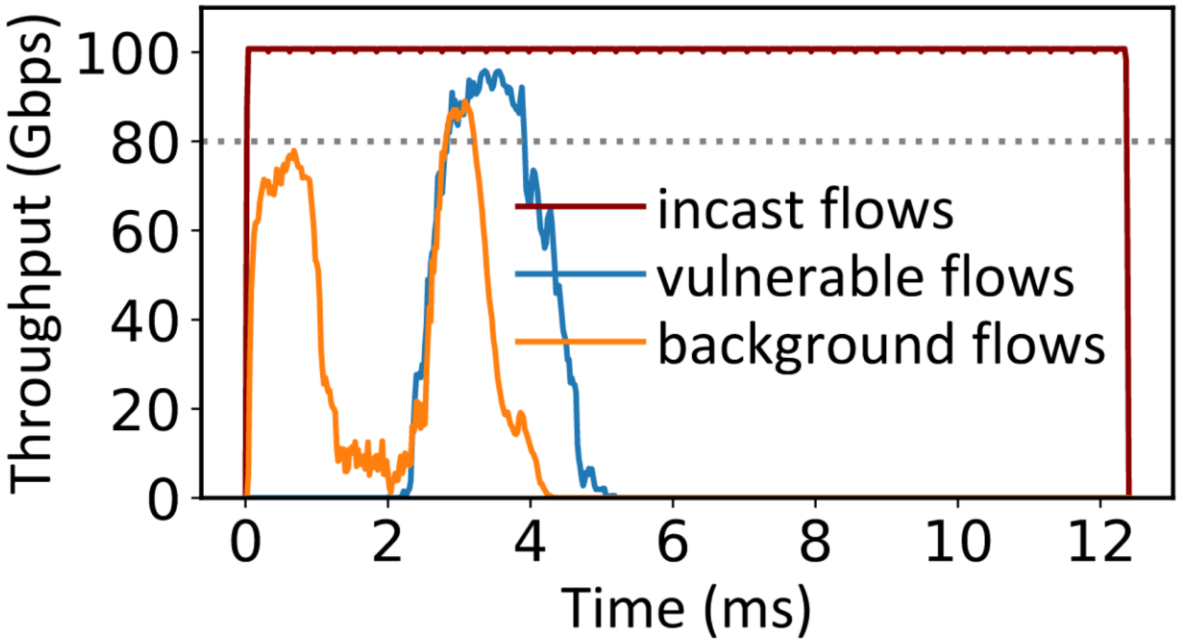
HOL Blocking Problem of Naive FC

Test workload:
continuous incast flows +
vulnerable flows with a total rate of 80 Gbps +
background flows with a total rate of 80 Gbps



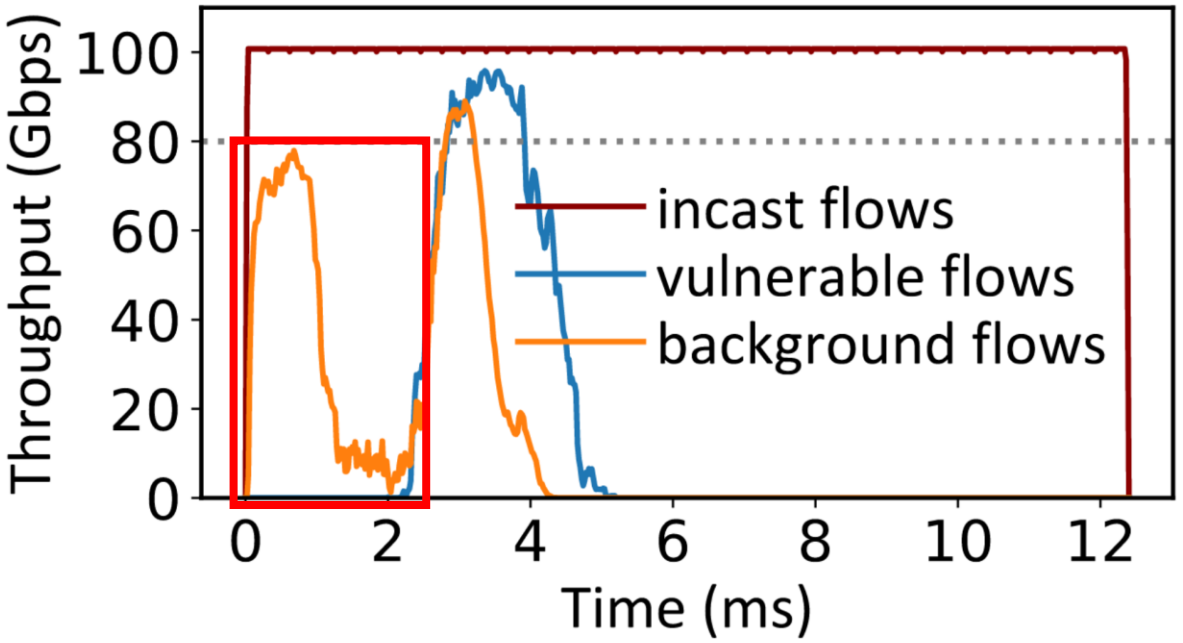
HOL Blocking Problem of Naive FC

PFC: The throughput of vulnerable flows and background flows is severely hurt due to HOL blocking.



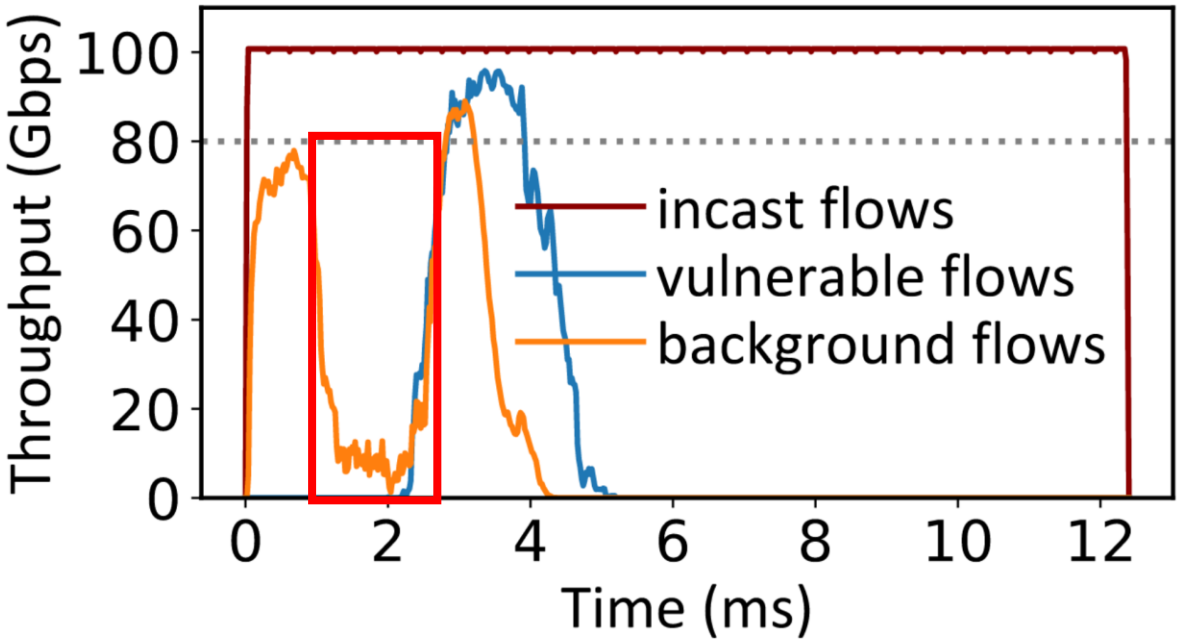
HOL Blocking Problem of Naive FC

PFC: The throughput of vulnerable flows and background flows is severely hurt due to HOL blocking.



HOL Blocking Problem of Naive FC

PFC: The throughput of vulnerable flows and background flows is severely hurt due to HOL blocking.



Idealized FC can Eliminate HOL blocking but is Non-scalable

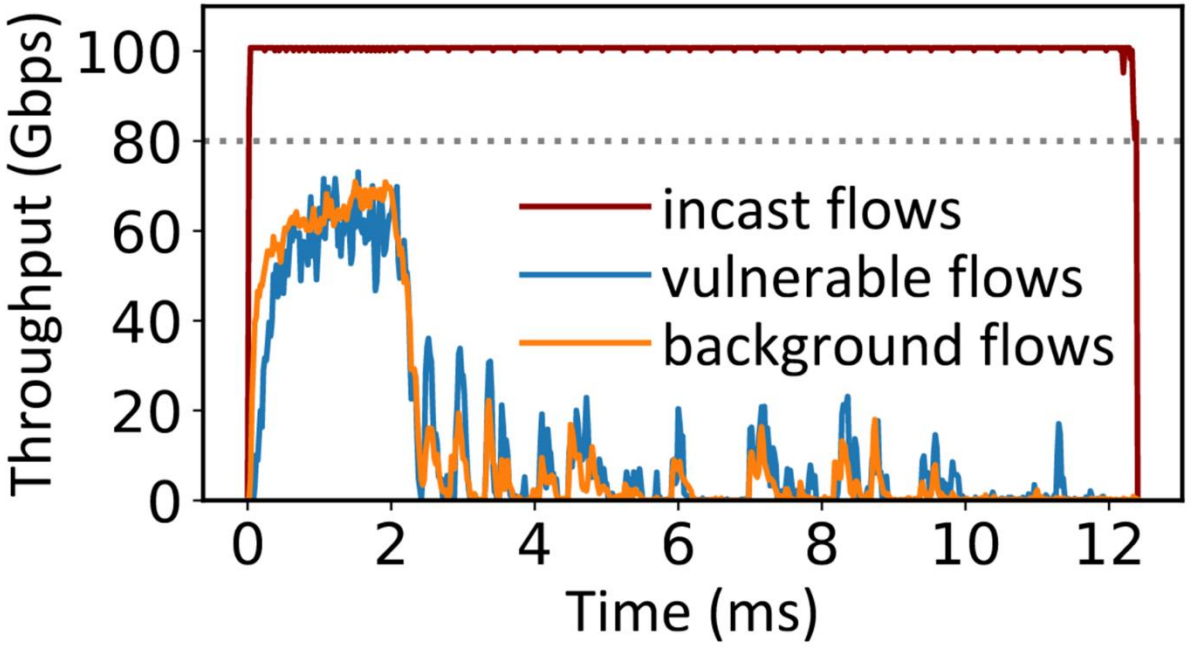
Per-flow-queue FC: Vulnerable and background flows fully utilize the link, at the same time the throughput of incast flows does not downgrade.

Not scalable: There could be thousands of flows that can be observed on a port [1,2].

[1] (EuroSys18) Information rich flow record generation on commodity switches.
[2] (SigMetric24) Lightweight Acquisition and Ranging of Flows in the Data Plane.

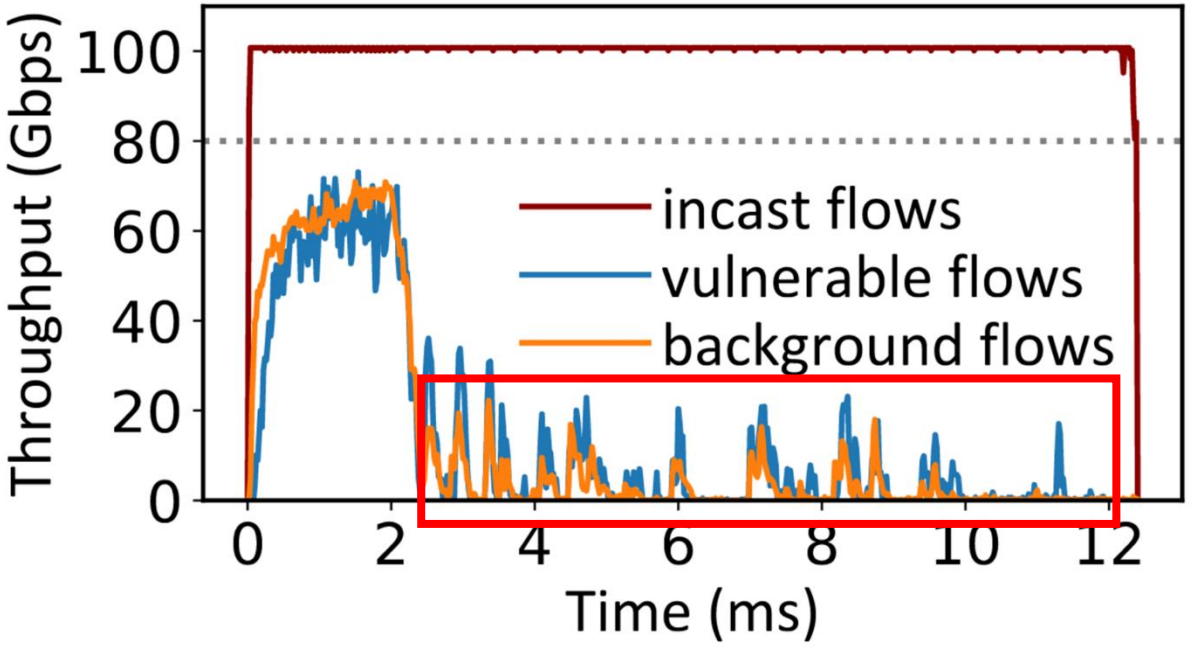
State-of-the-art FC is Flawed

BFC@NSDI22: Vulnerable and background flows sharing the same queue with incast flows are severely hurt.



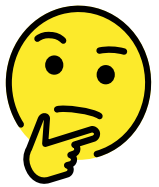
State-of-the-art FC is Flawed

BFC@NSDI22: Vulnerable and background flows sharing the same queue with incast flows are severely hurt.

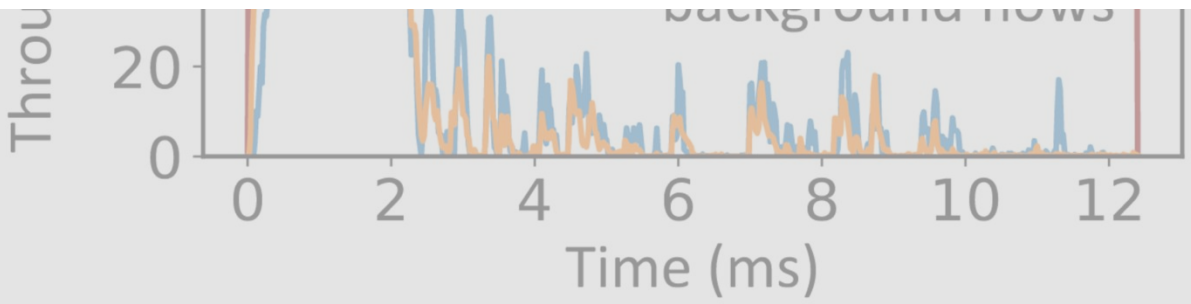


State-of-the-art FC is Flawed

BFC@NSDI22: Vulnerable and background flows sharing the same queue with incast flows are severely hurt.



How to eliminate HOL blocking caused by flow control in the most cost-effective way



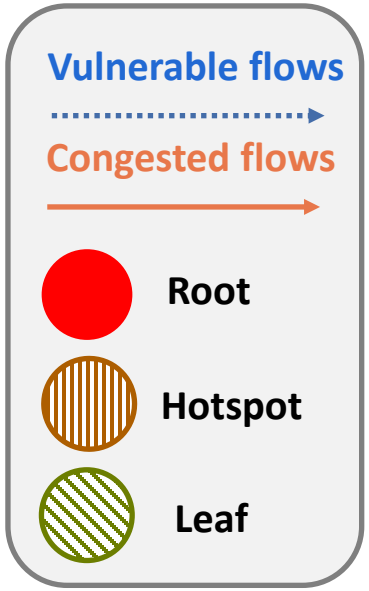
We Need a Better Control Granularity for Per-hop FC

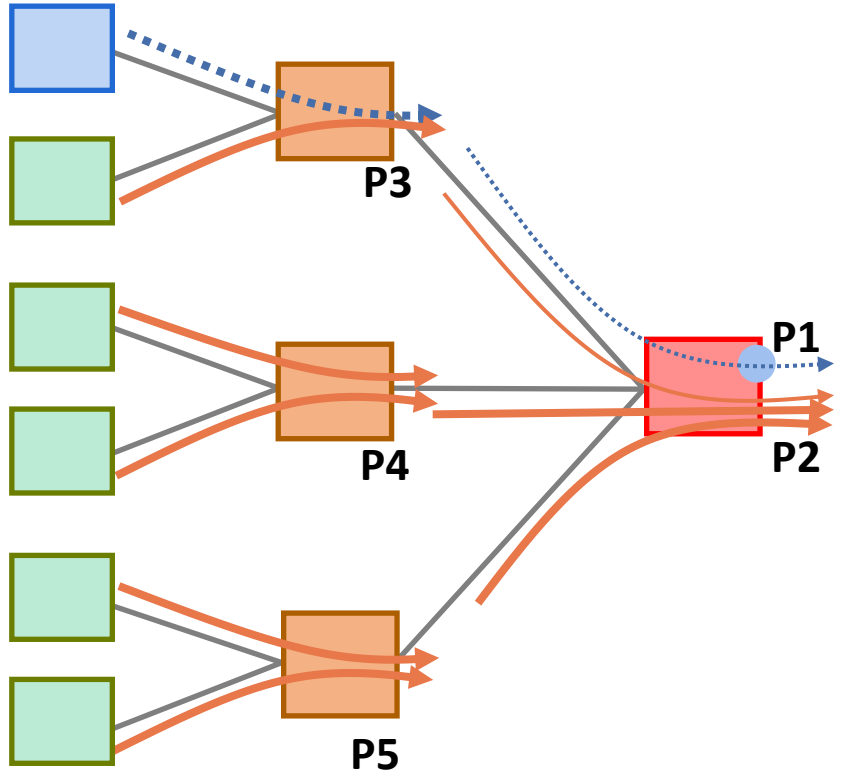
PFC's control granularity:
control the entire priority queue. ⇢ Head-of-line blocking.

Per-flow-queue's control granularity:
control at the flow granularity. ⇢ Non-scalable.

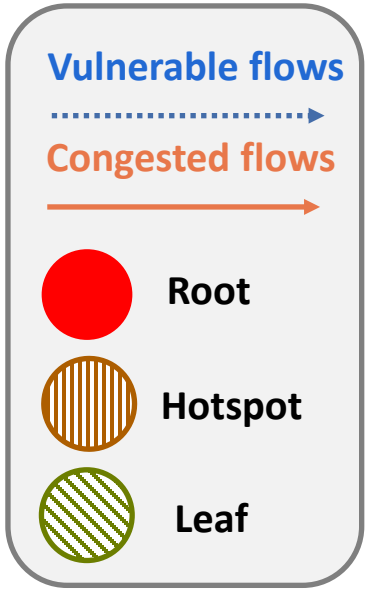
A better control granularity? ⟹ Blocking-free and scalable.

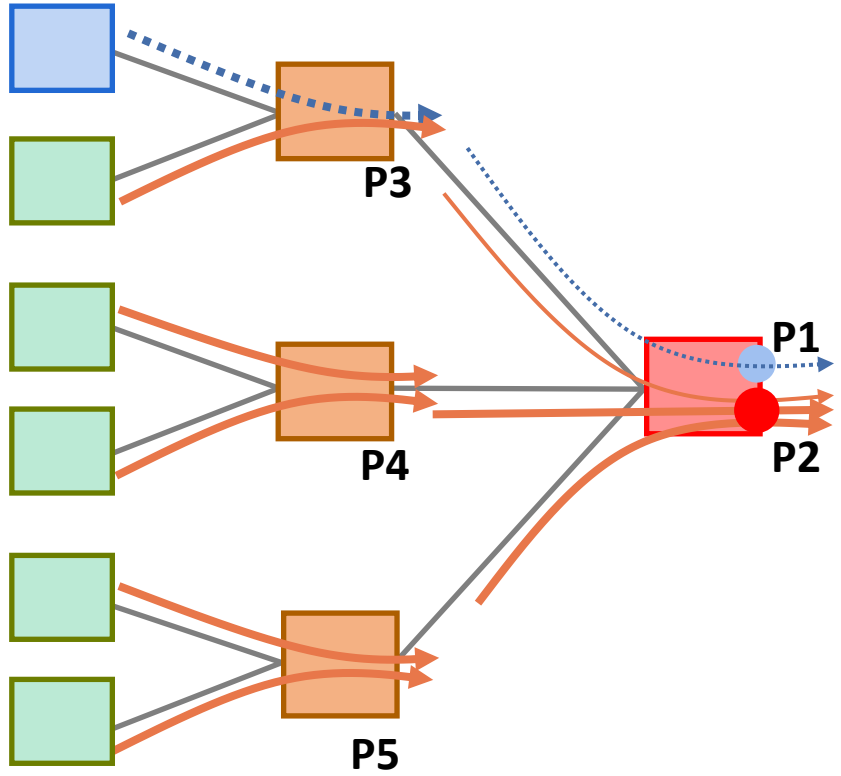
Analysis of Congestion in Per-hop FC



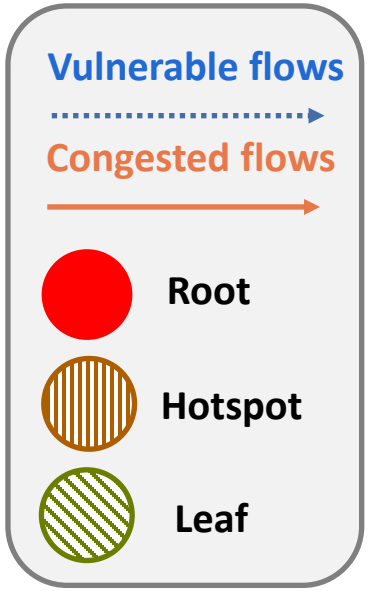


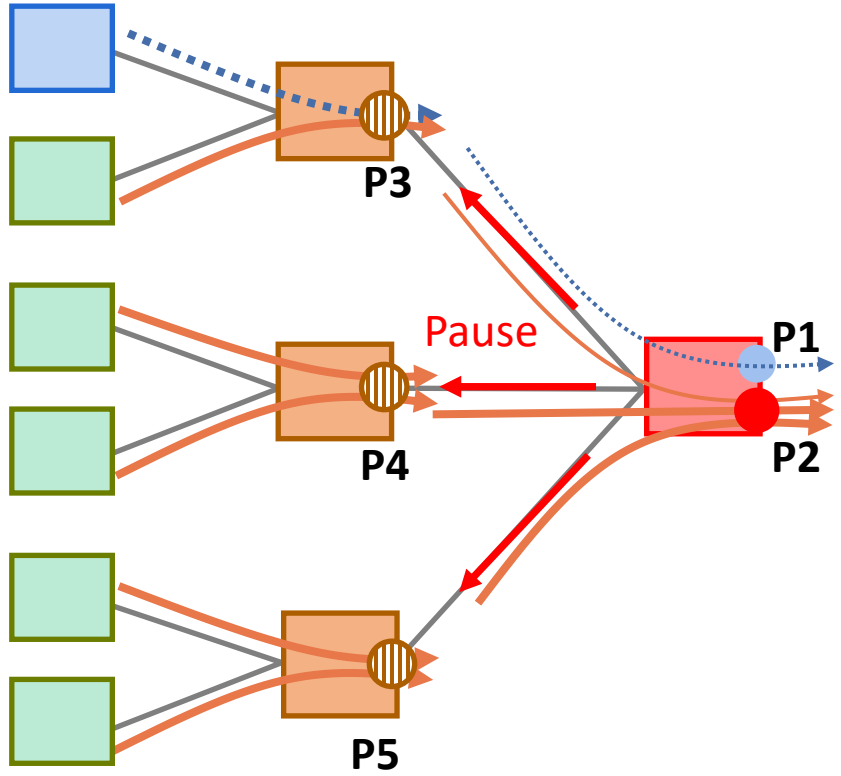
Analysis of Congestion in Per-hop FC



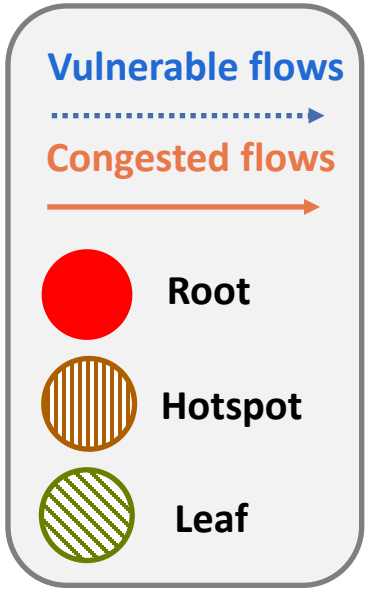


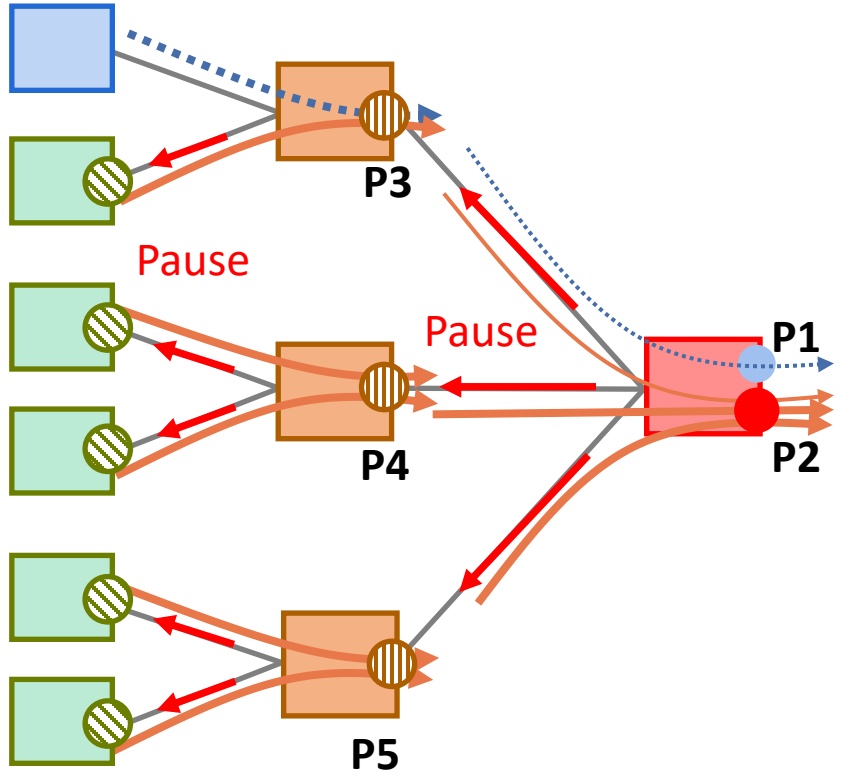
Analysis of Congestion in Per-hop FC



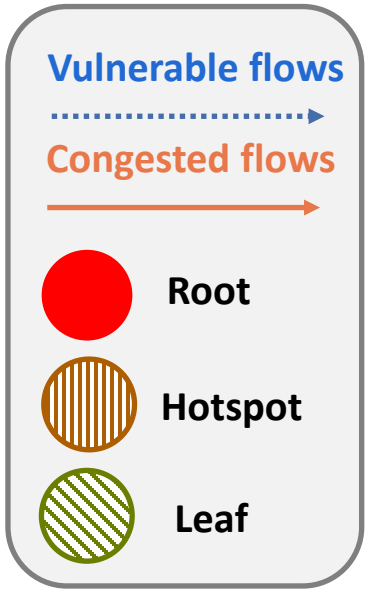


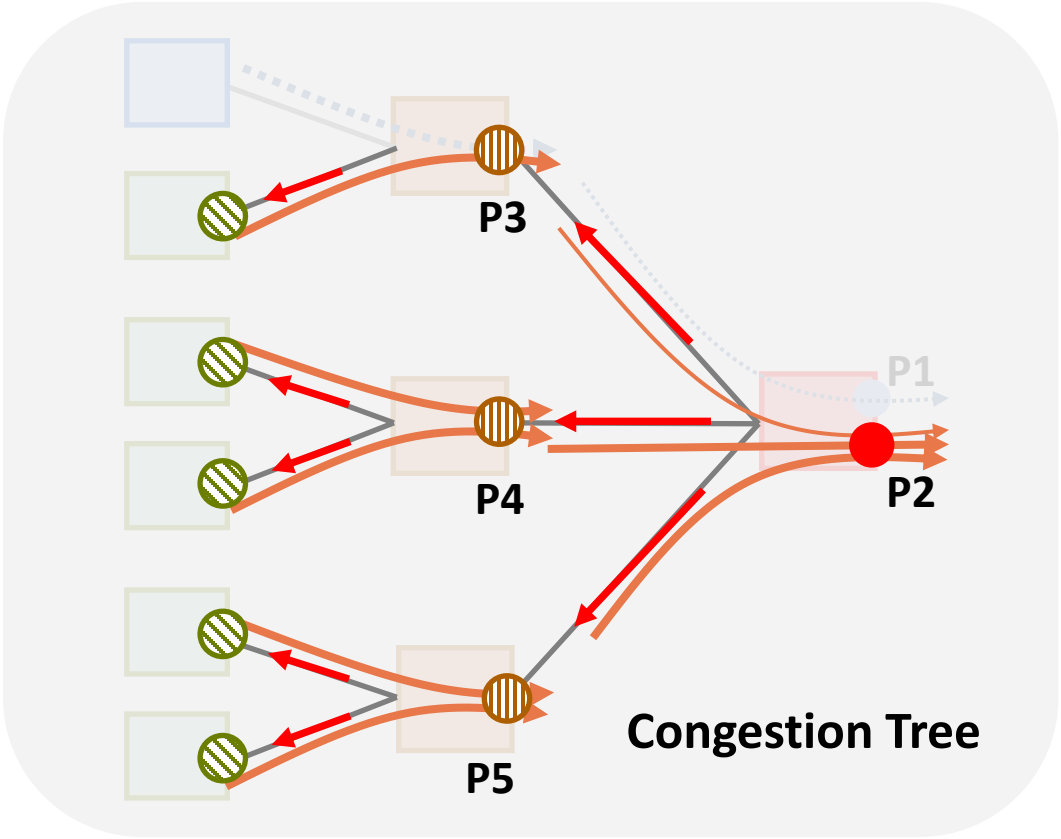
Analysis of Congestion in Per-hop FC



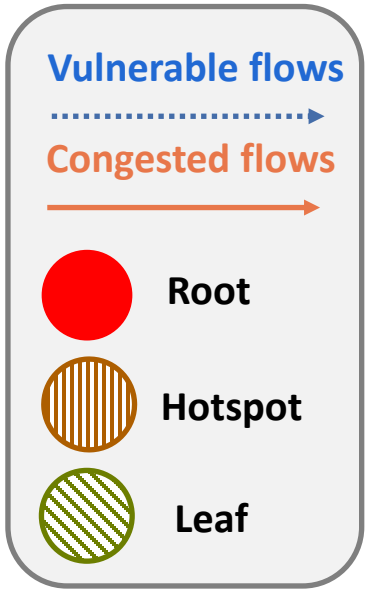


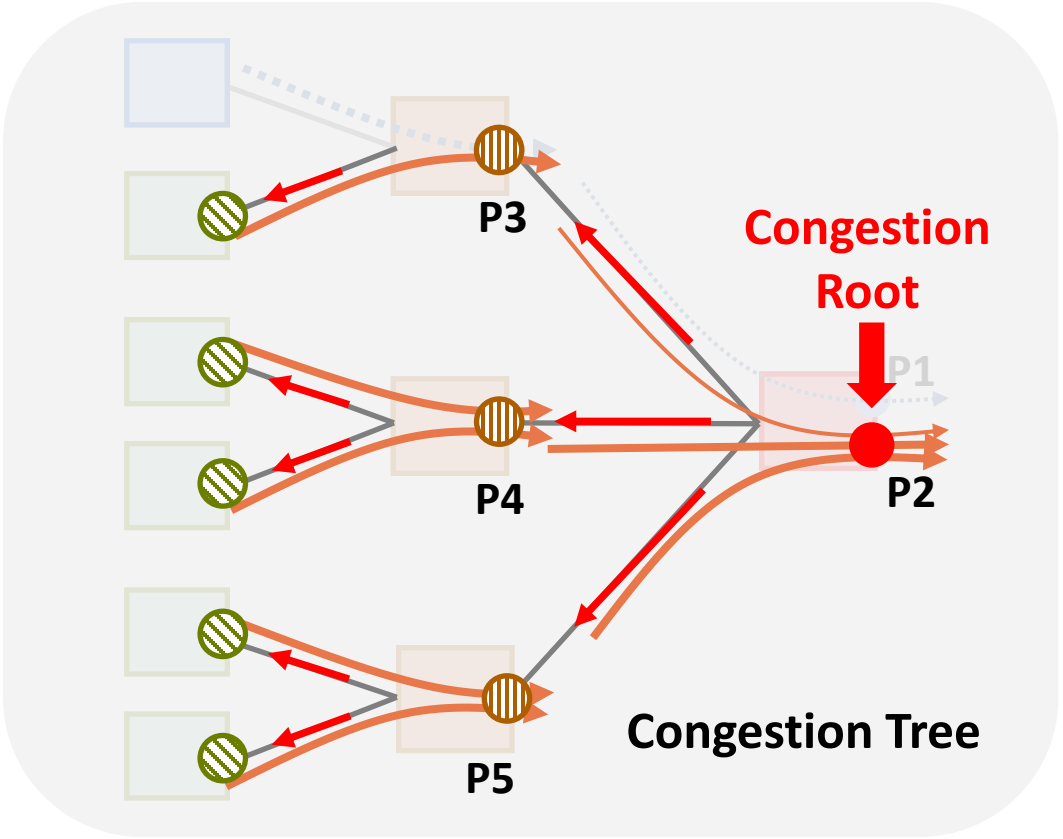
Analysis of Congestion in Per-hop FC





Analysis of Congestion in Per-hop FC





Analysis of Congestion in Per-hop FC

Only control the flows passing through the congestion root.

Blocking-free.

Analysis of Congestion in Per-hop FC

Only control the flows passing through the congestion root. ⟹ **Blocking-free.**

The number of concurrent congestion roots observed on each port is moderate [3]. ⟹ **Scalable.**

[3] (IMC) Network traffic characteristics of data centers in the wild.

Analysis of Congestion in Per-hop FC

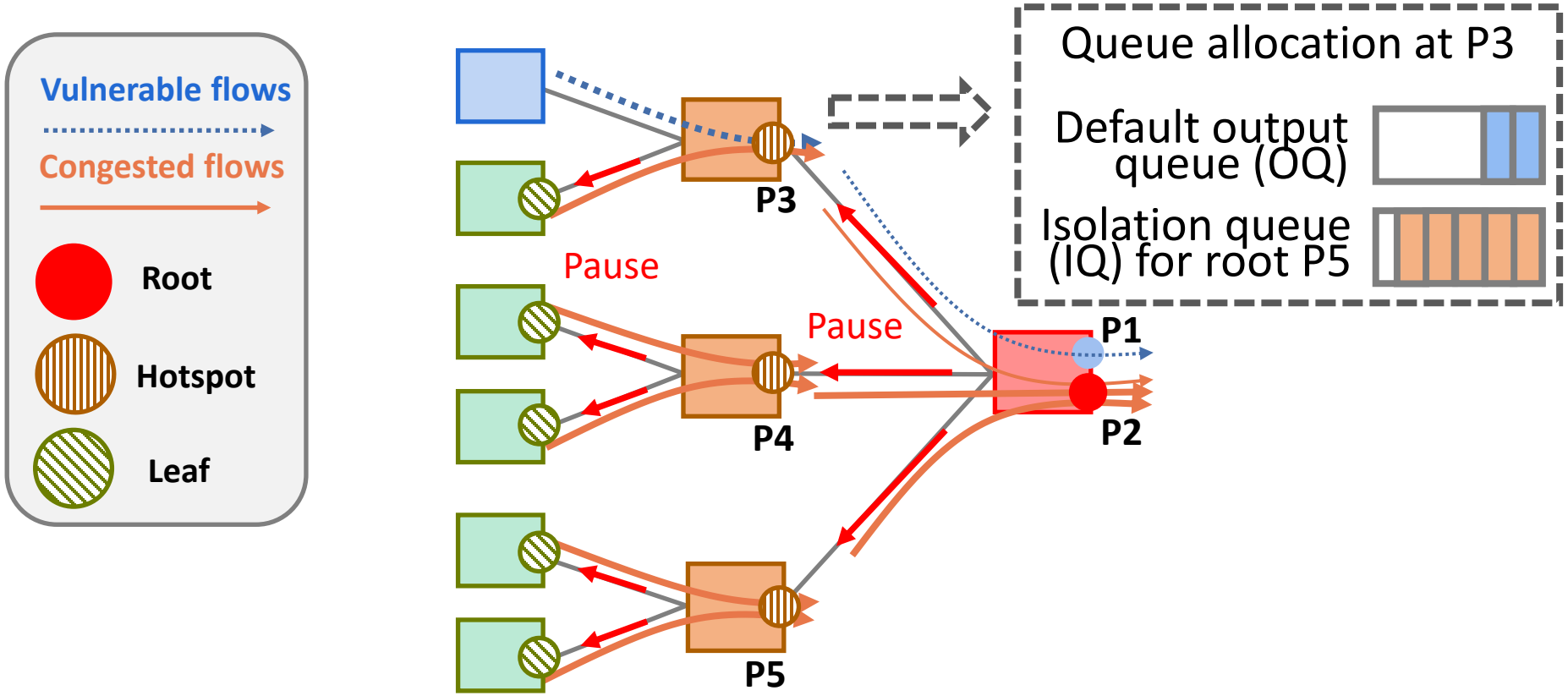
Key insight: Congestion root is the appropriate granularity of flow control!

Only control the flows passing through the congestion root. ⟹ **Blocking-free.**

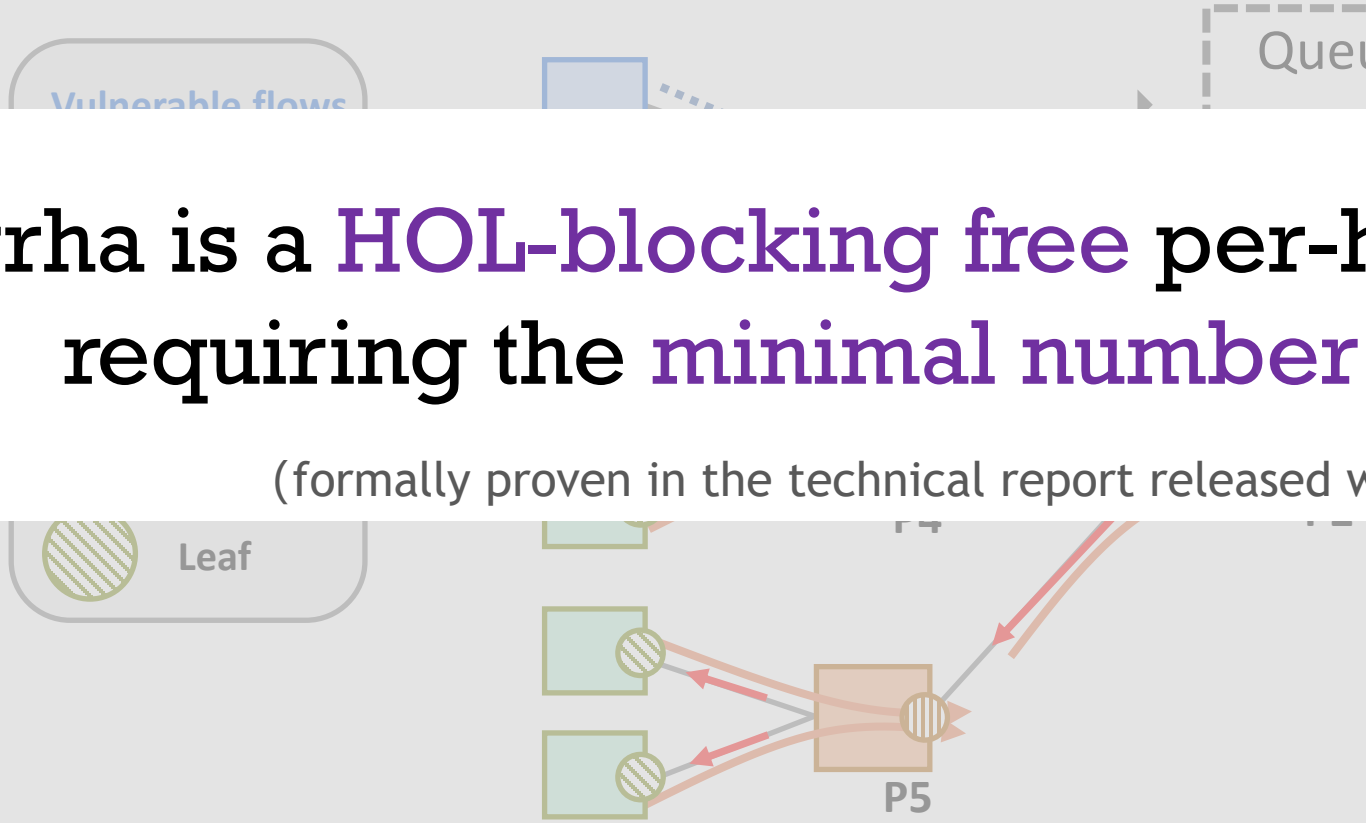
The number of concurrent congestion roots observed on each port is moderate [3]. ⟹ **Scalable.**

[3] (IMC) Network traffic characteristics of data centers in the wild.

Pyrrha: A Congestion-Root-Based Flow Control



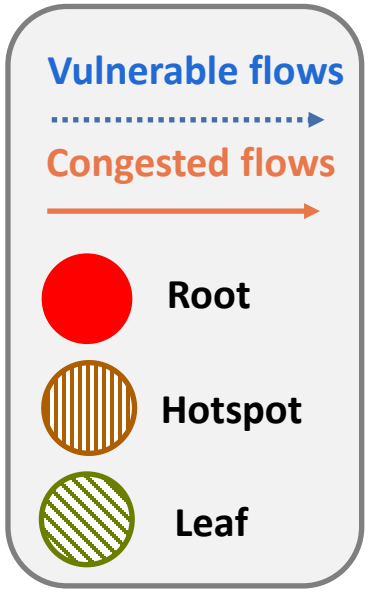
Pyrrha: A Congestion-Root-Based Flow Control

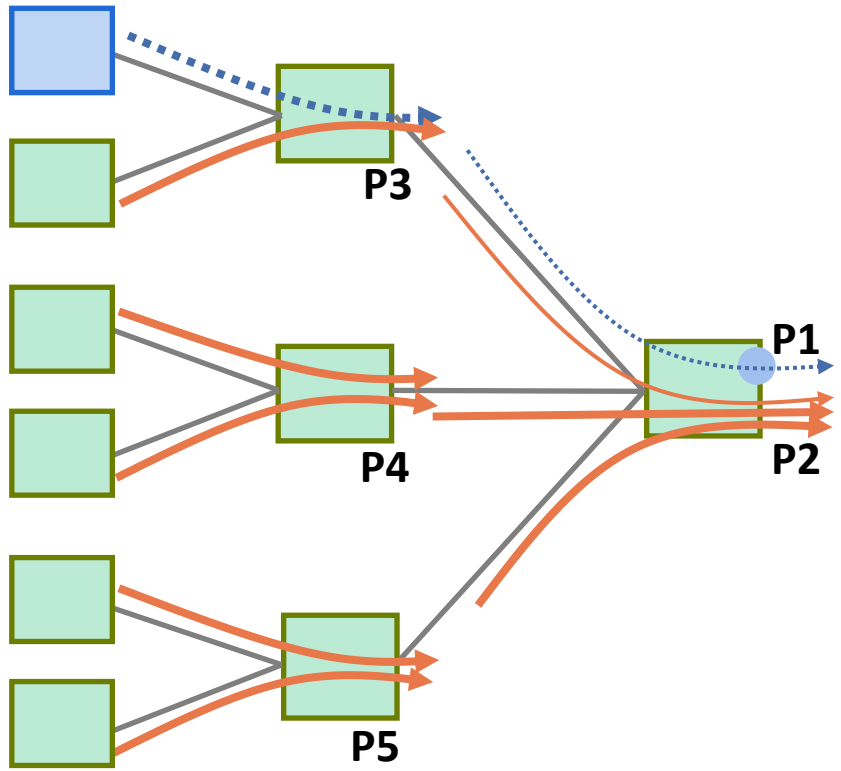


Pyrrha is a HOL-blocking free per-hop FC protocol requiring the minimal number of queues.

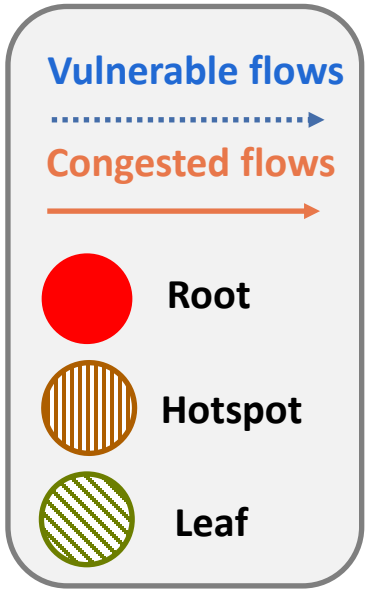
(formally proven in the technical report released with our paper.)

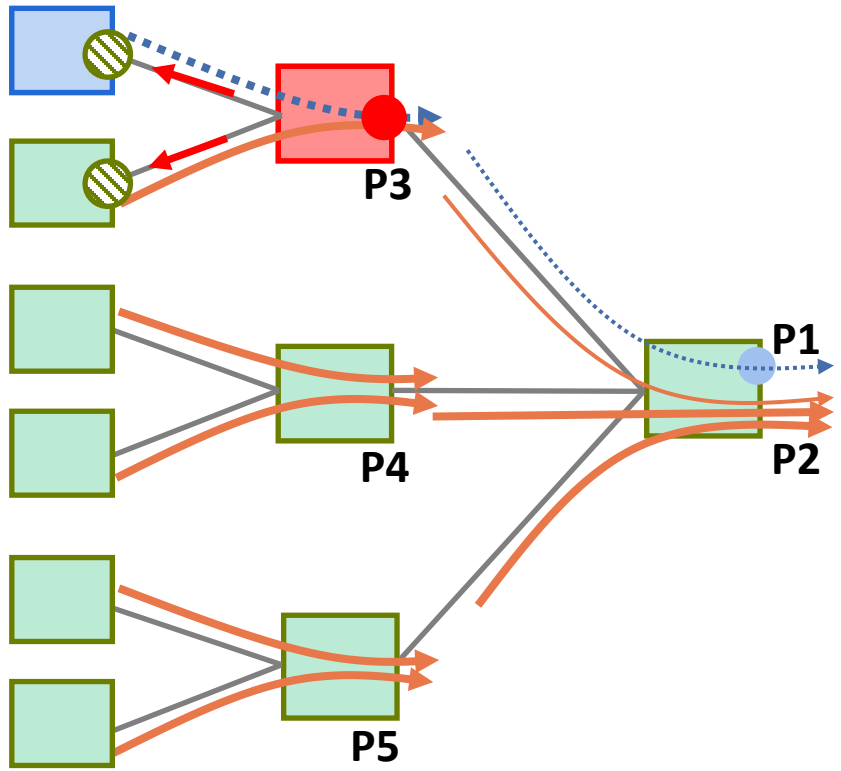
Key Challenge: Identify Congestion Root



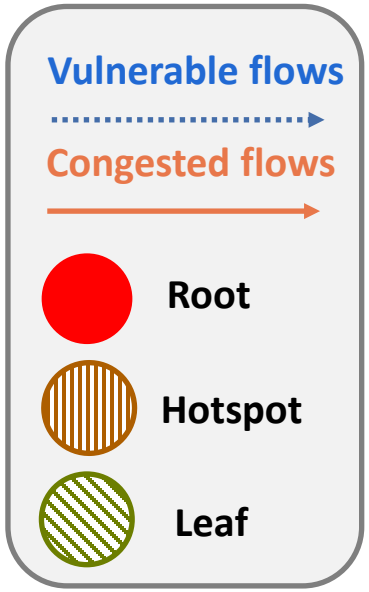


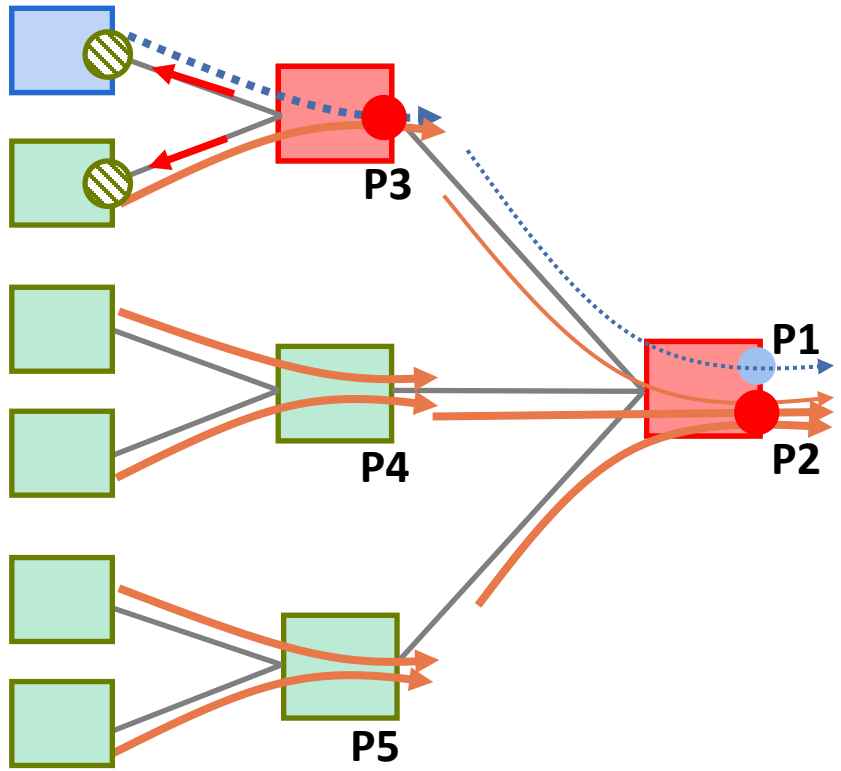
Key Challenge: Identify Congestion Root



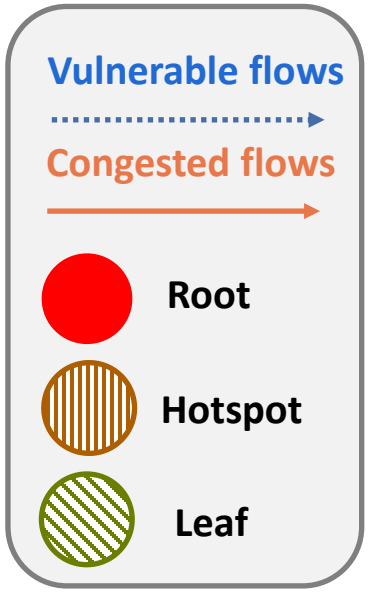


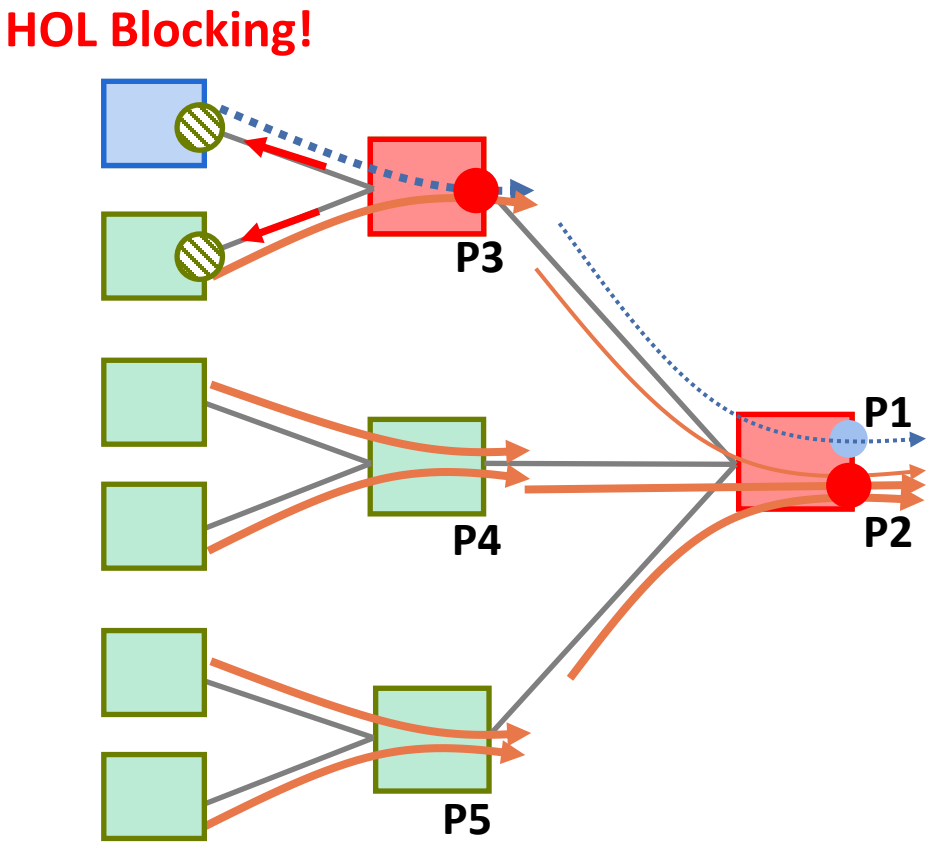
Key Challenge: Identify Congestion Root



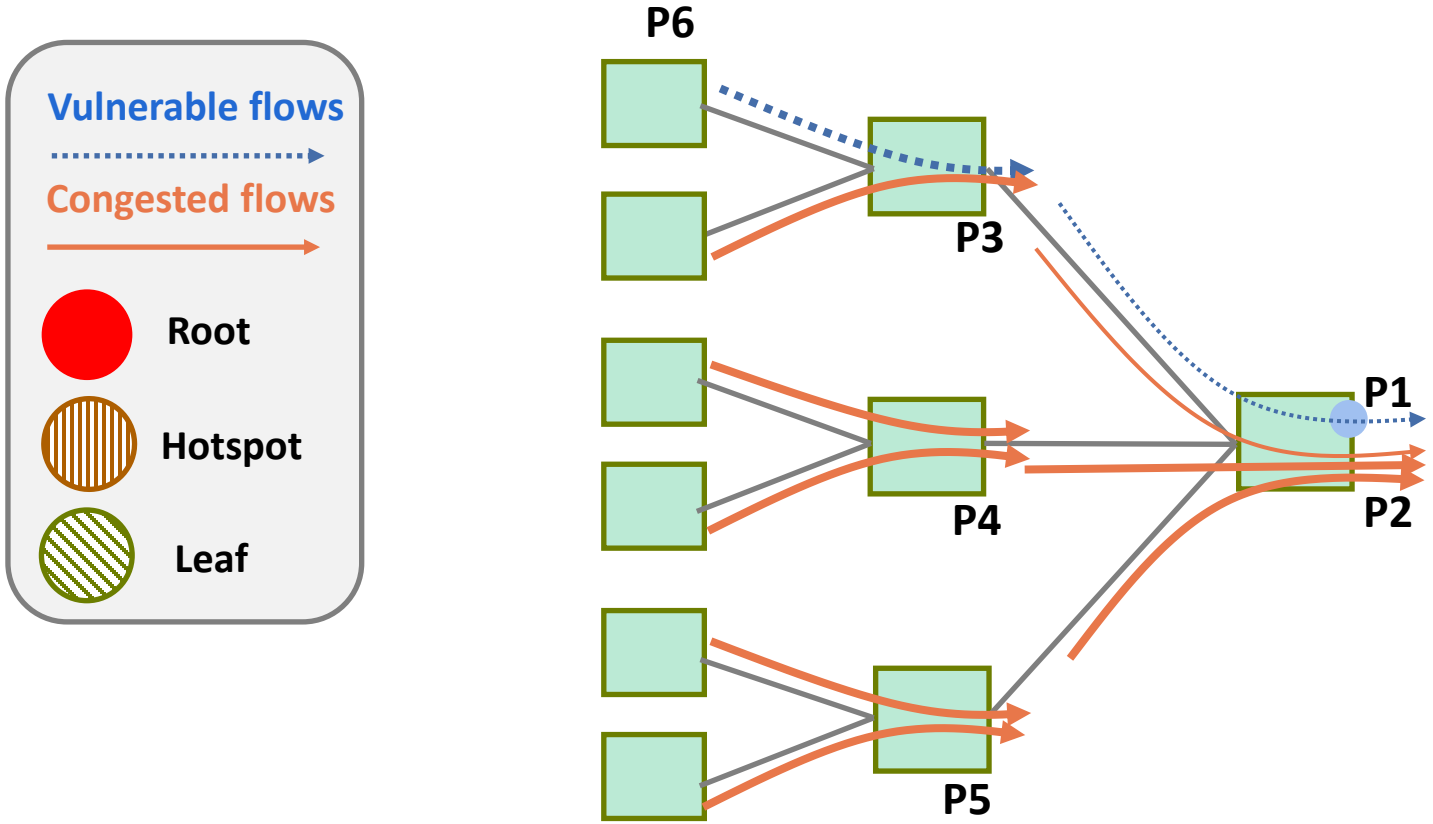


Key Challenge: Identify Congestion Root

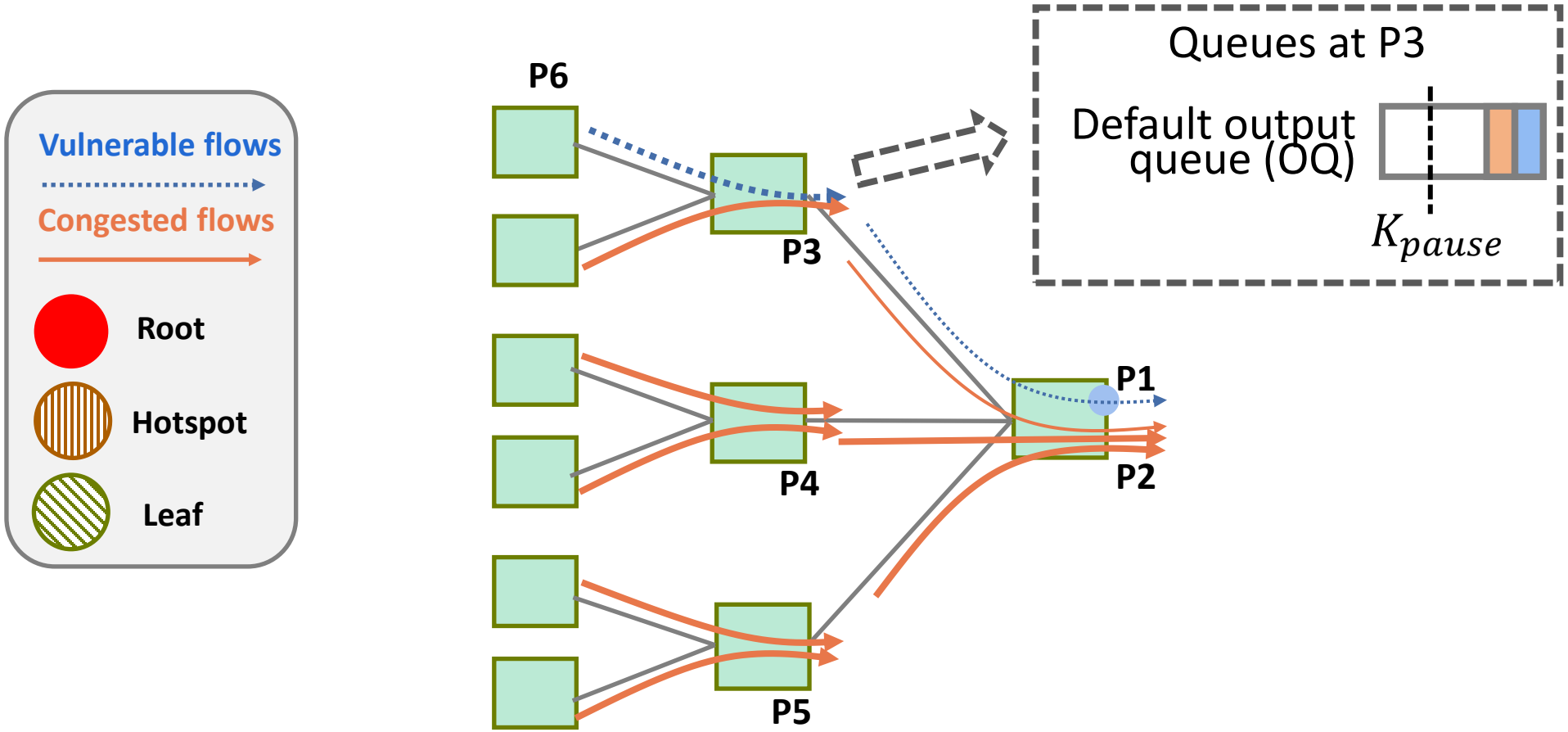




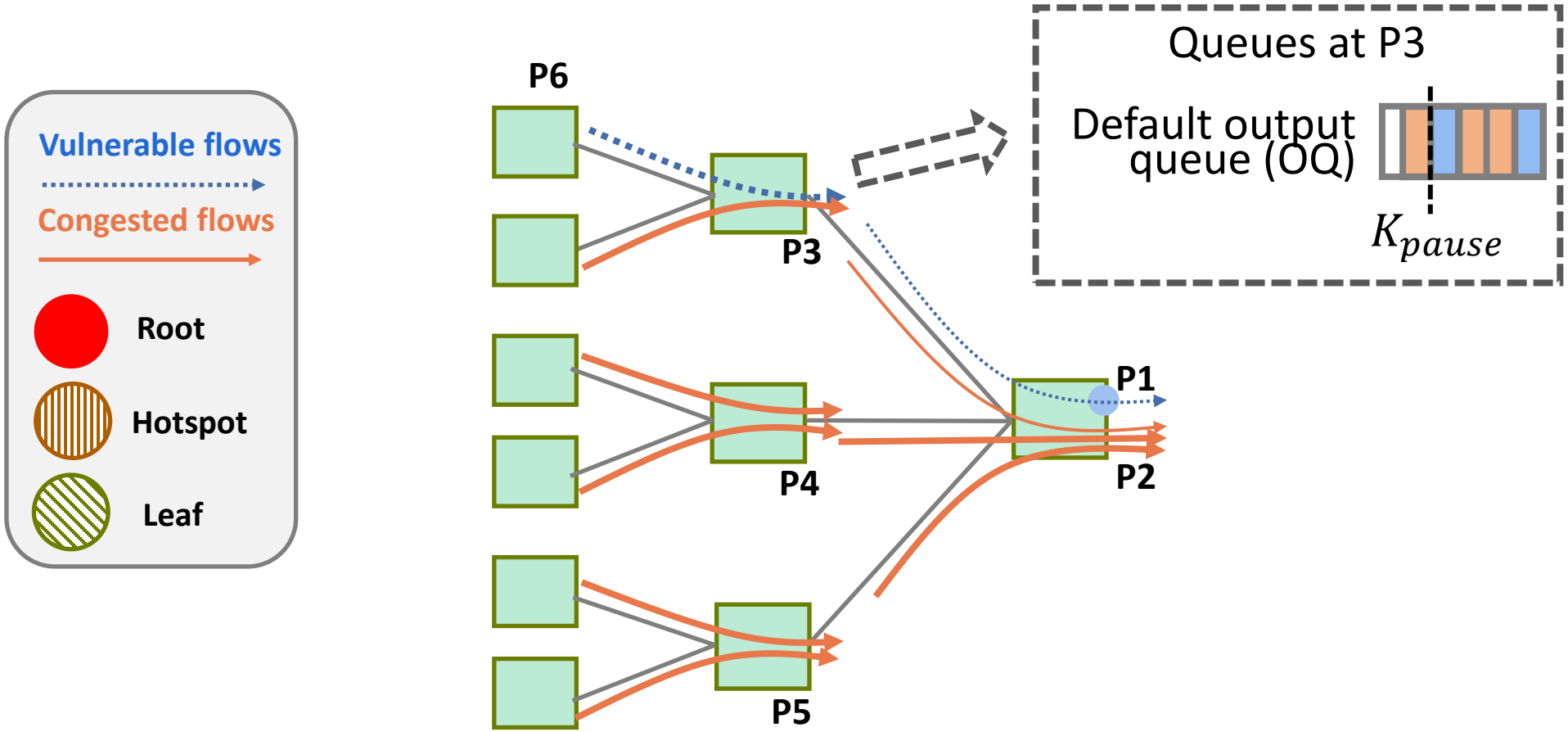
Pyrrha Design: Distributed Congestion Root Election



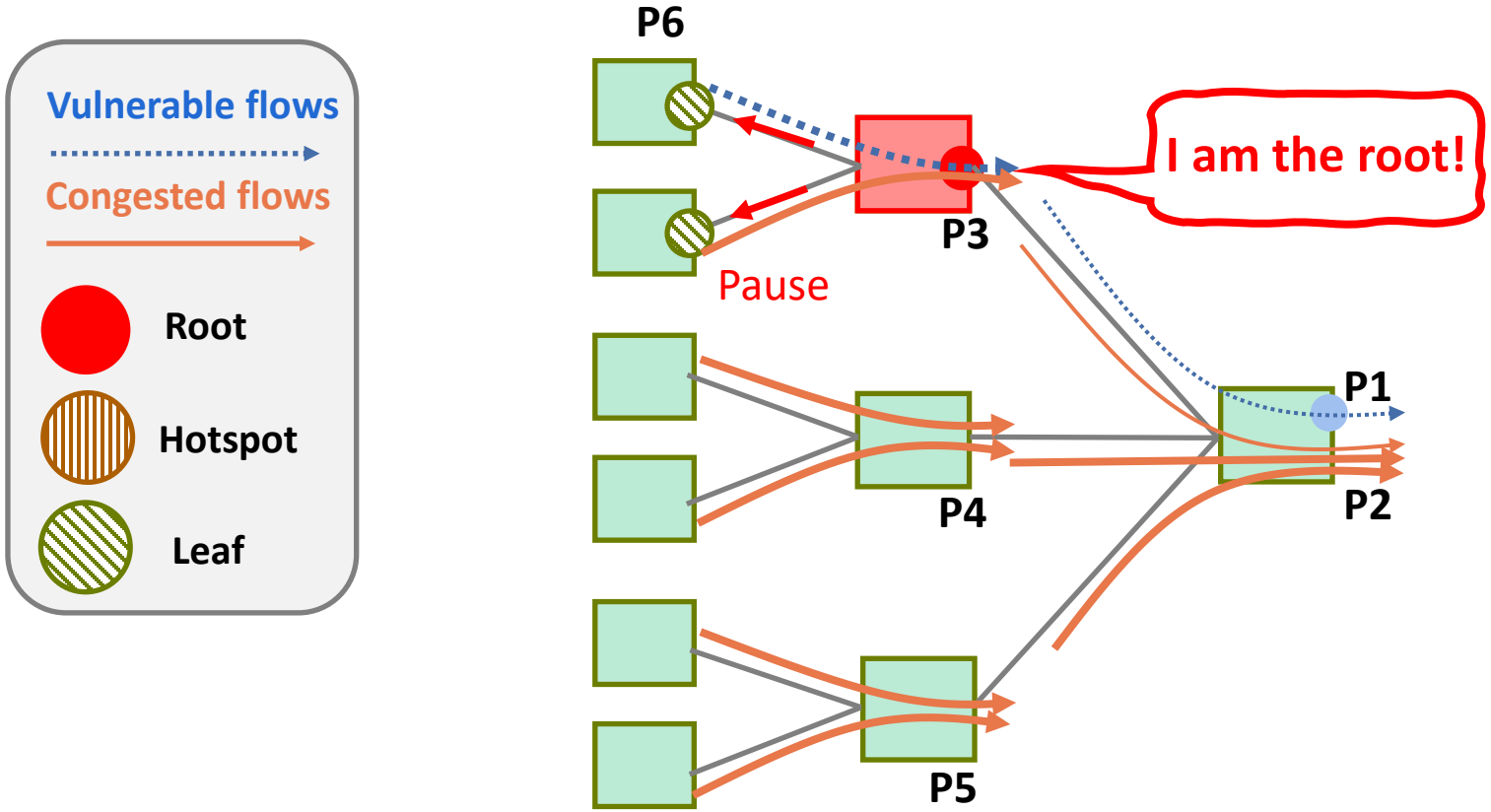
Pyrrha Design: Distributed Congestion Root Election



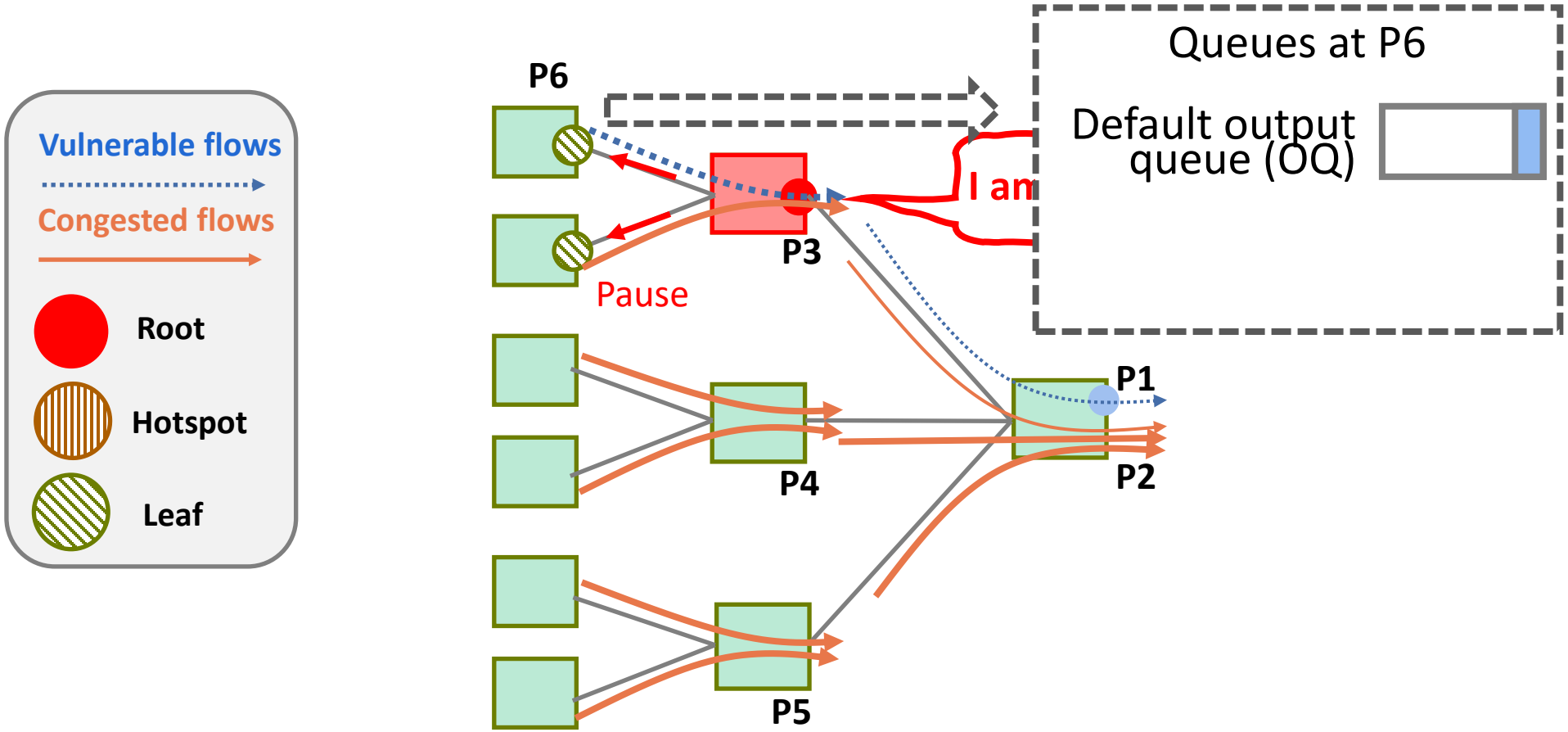
Pyrrha Design: Distributed Congestion Root Election



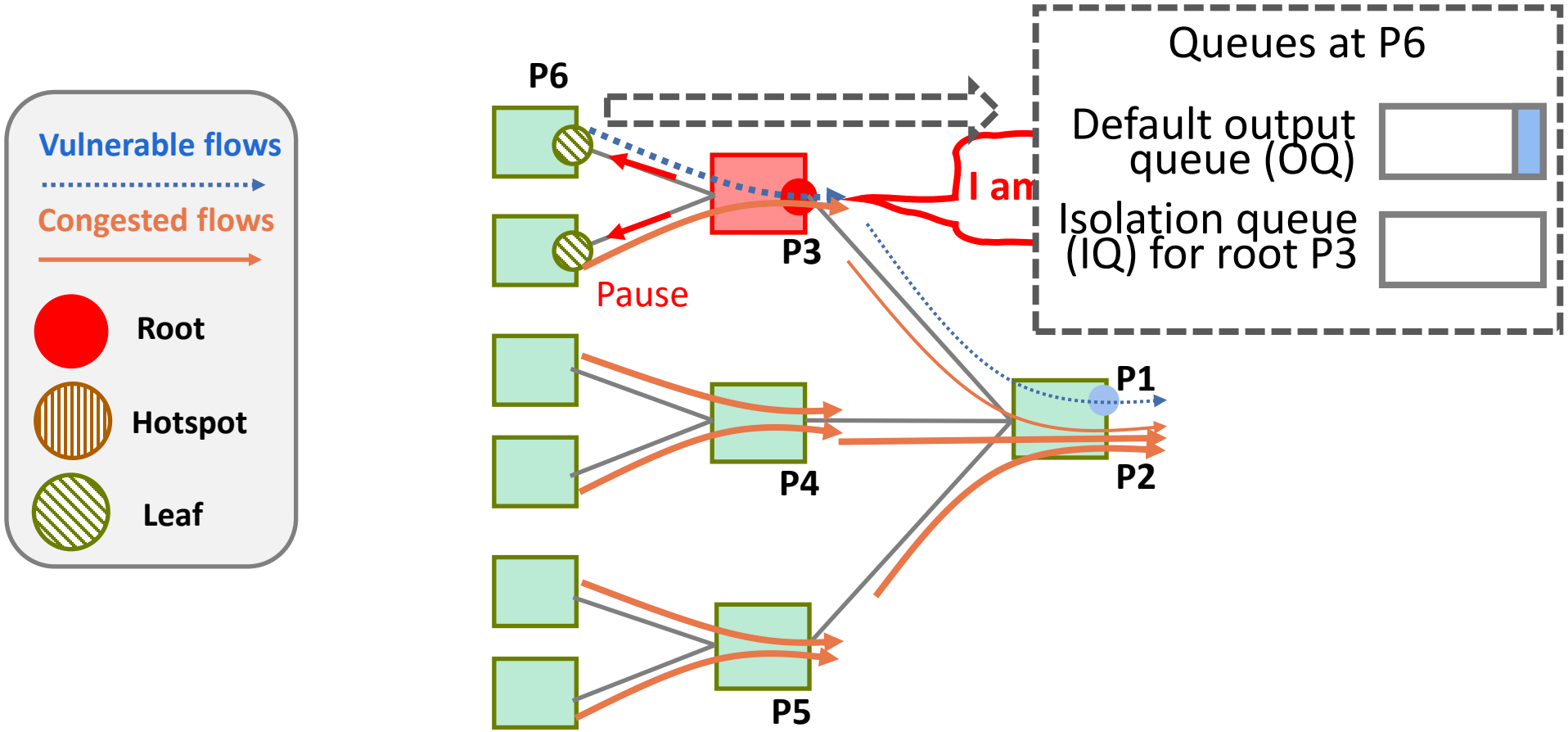
Pyrrha Design: Distributed Congestion Root Election



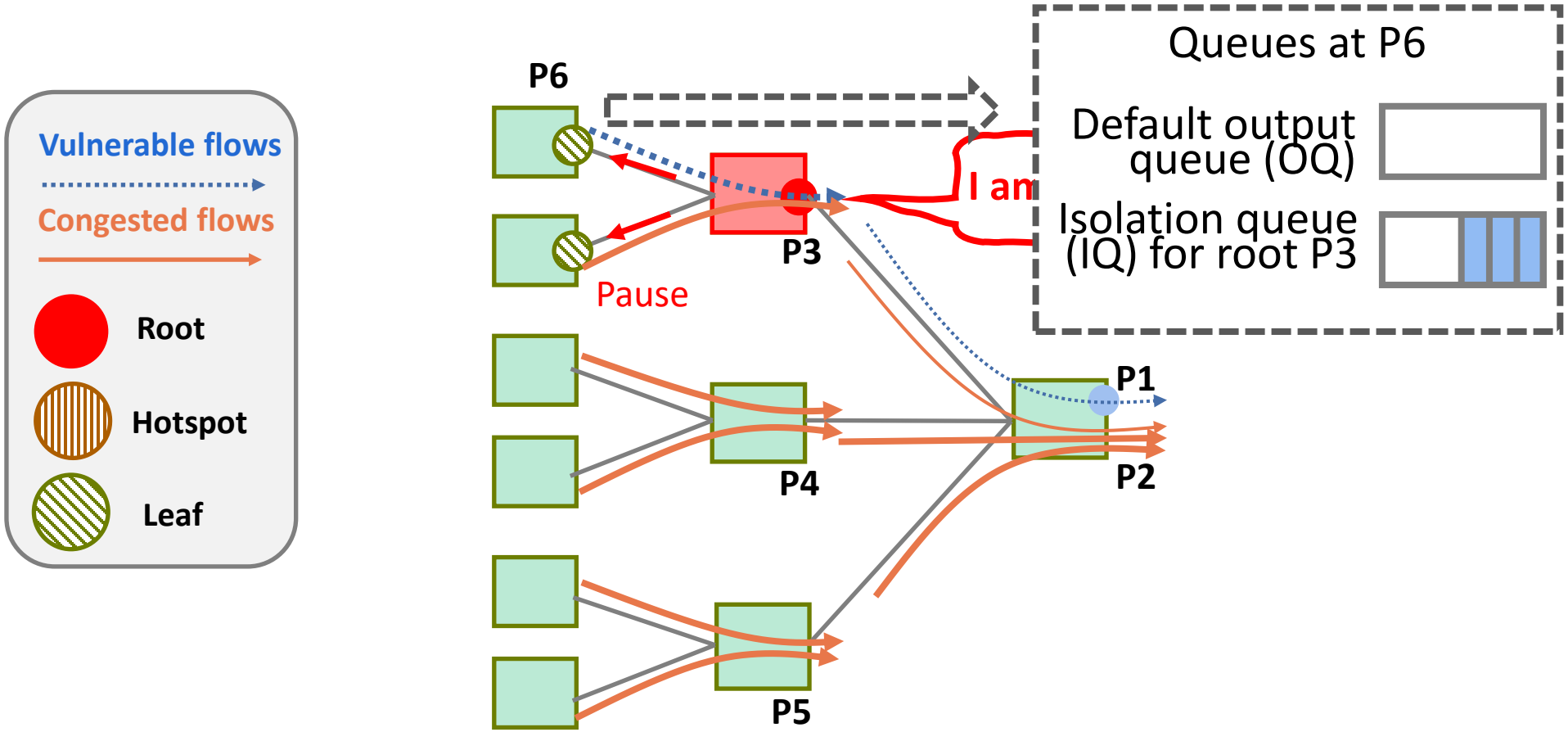
Pyrrha Design: Distributed Congestion Root Election



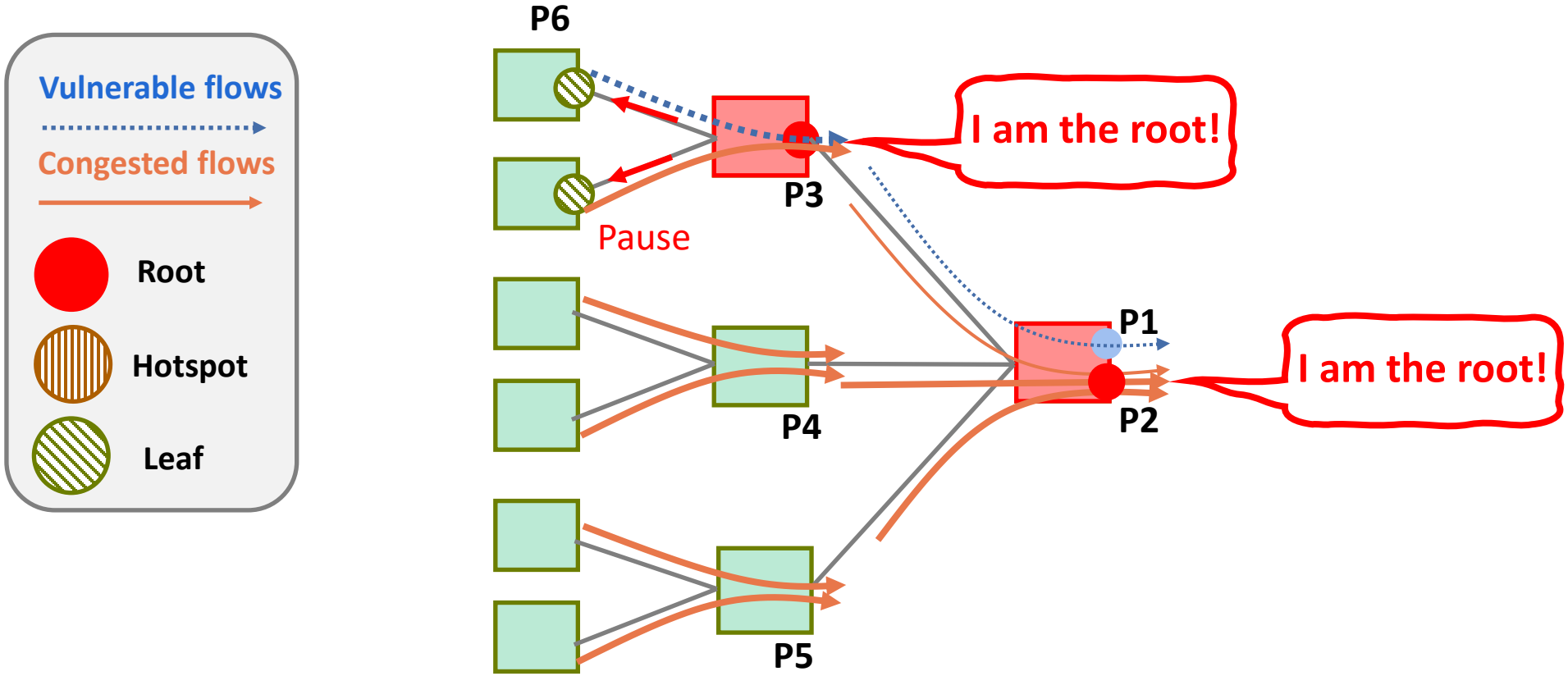
Pyrrha Design: Distributed Congestion Root Election



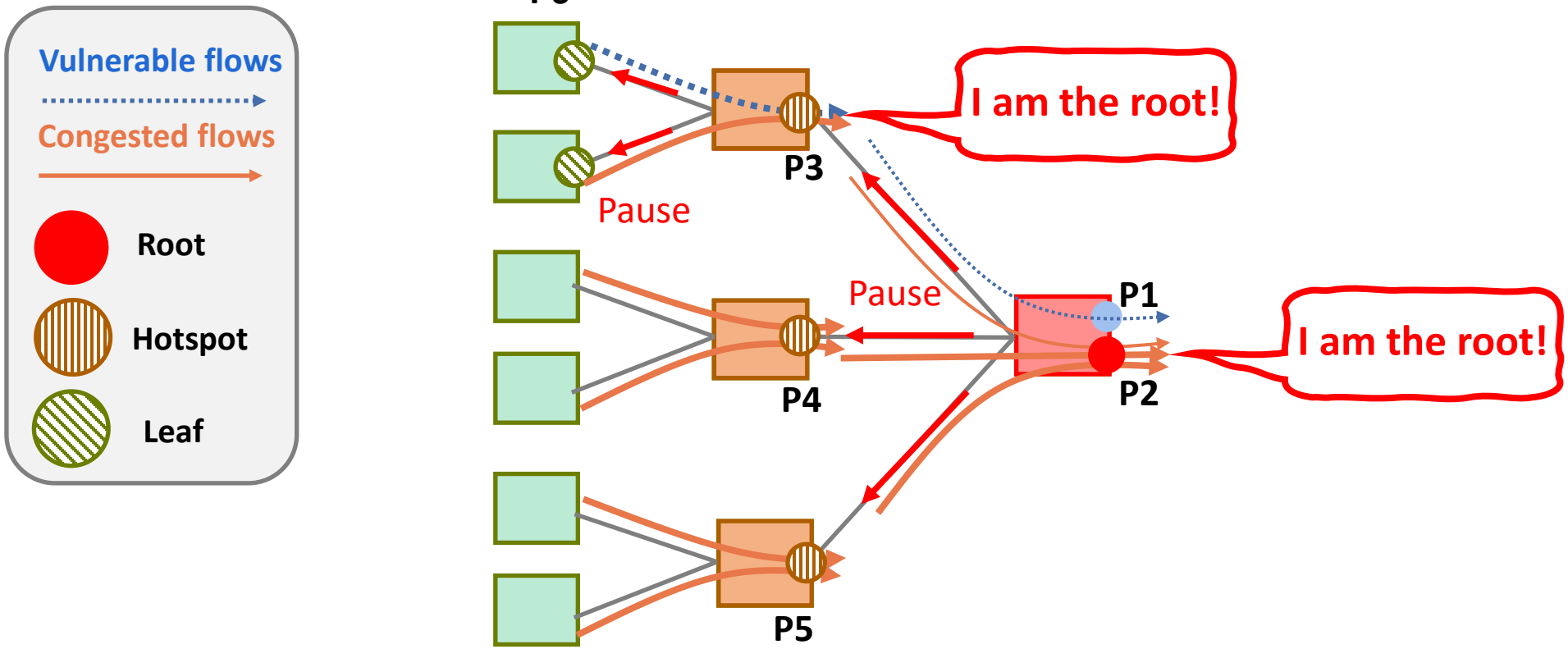
Pyrrha Design: Distributed Congestion Root Election



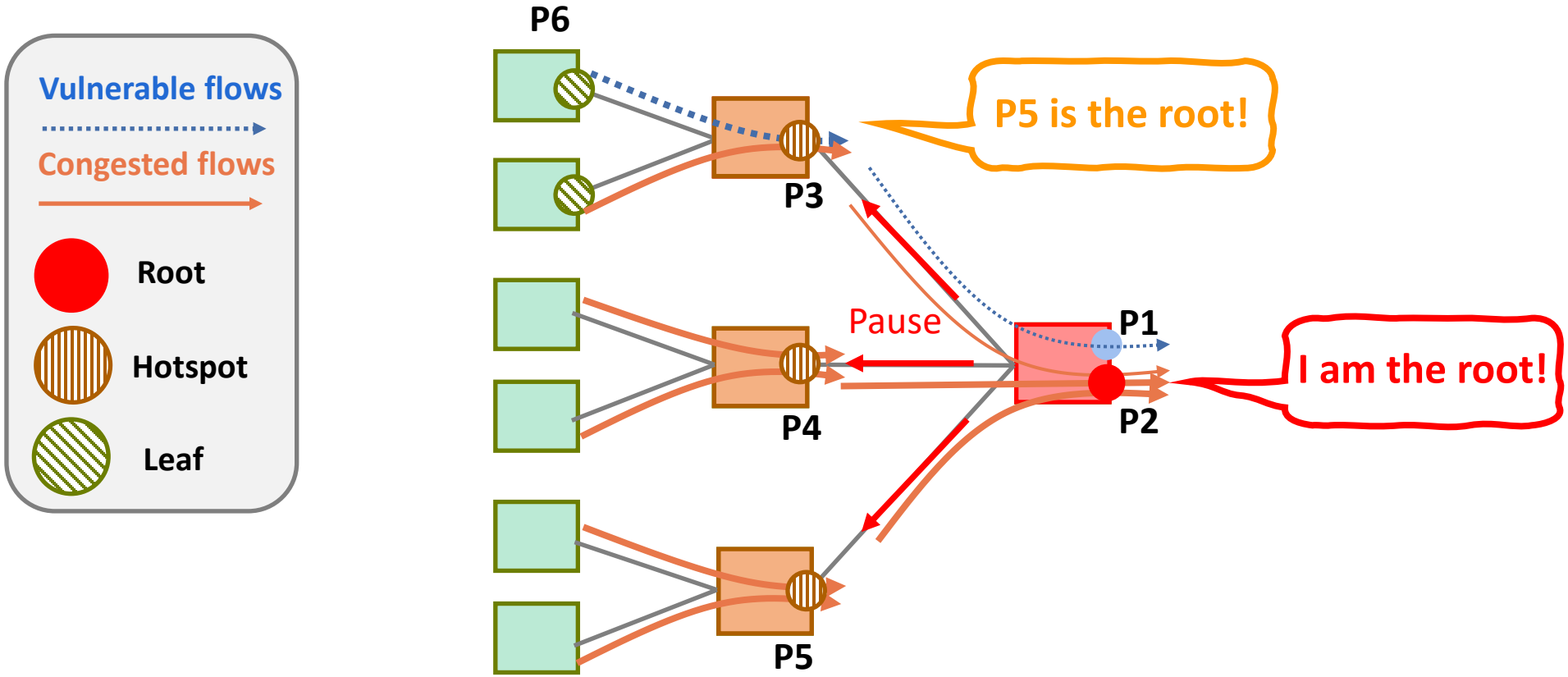
Pyrrha Design: Distributed Congestion Root Election



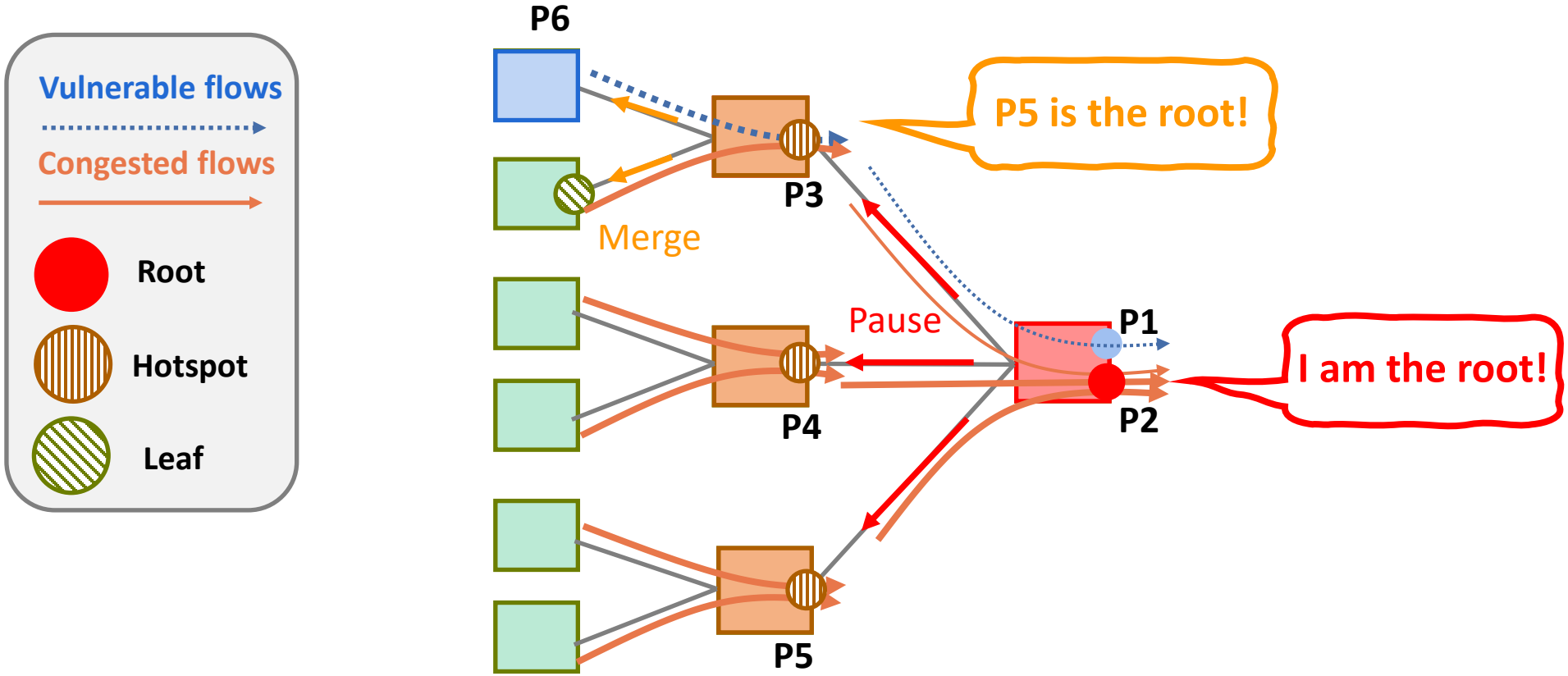
Pyrrha Design: Distributed Congestion Root Election



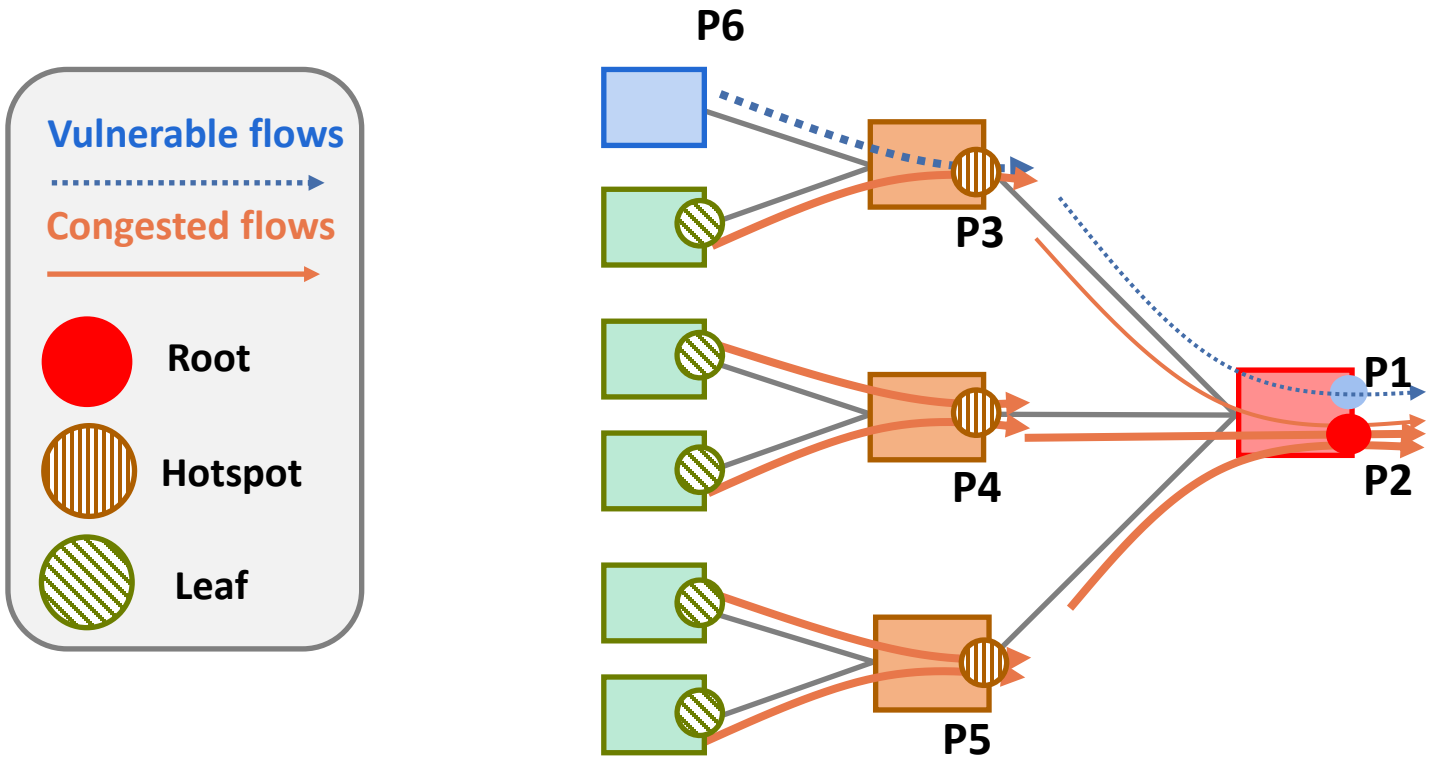
Pyrrha Design: Distributed Congestion Root Election



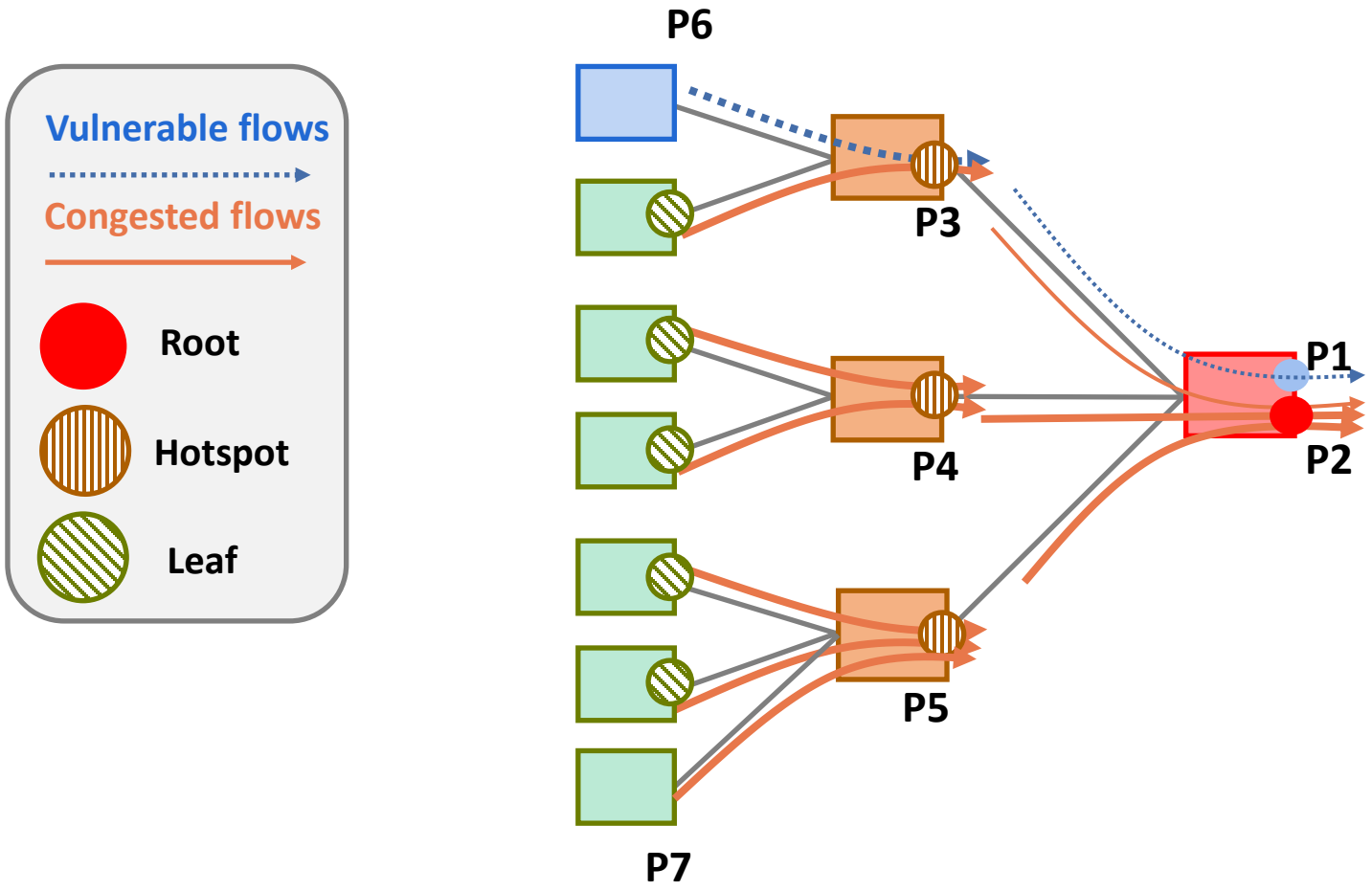
Pyrrha Design: Distributed Congestion Root Election



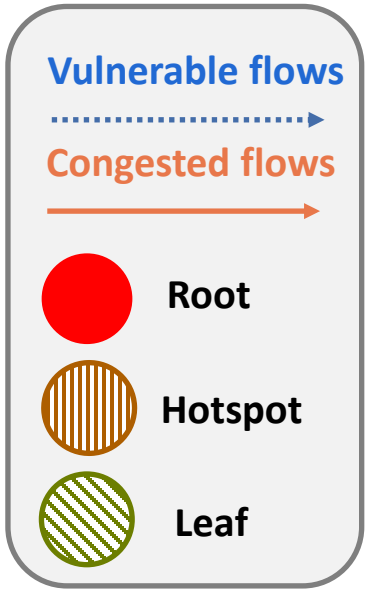
Challenge 2: Identify Congested Flows

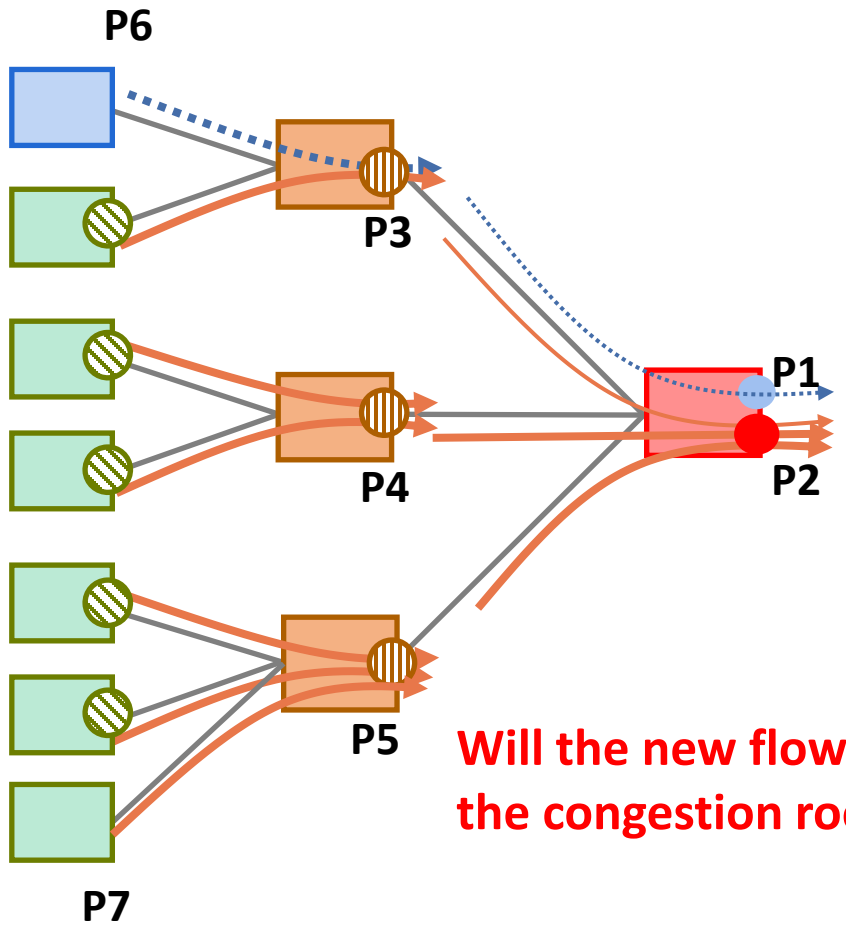


Challenge 2: Identify Congested Flows



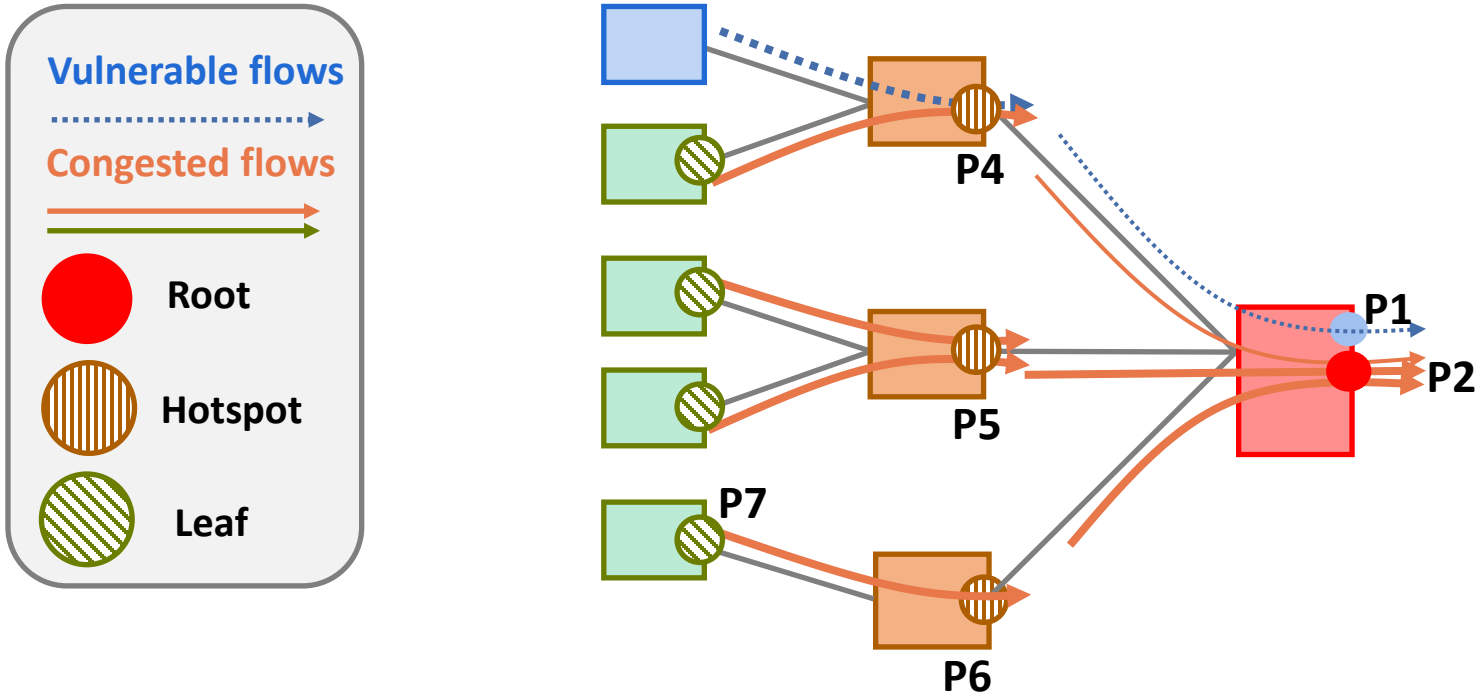
Challenge 2: Identify Congested Flows



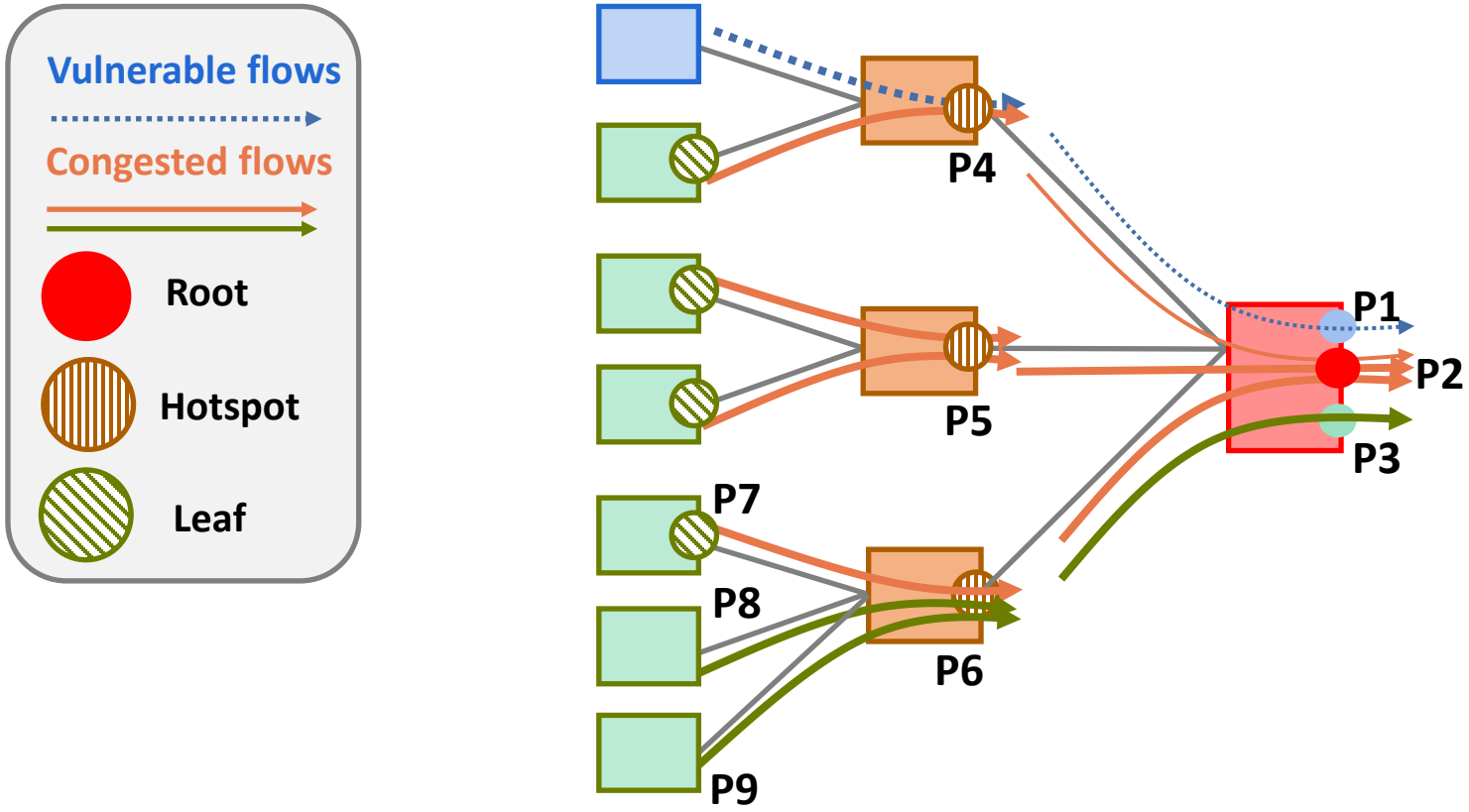


Will the new flow pass through the congestion root P2?

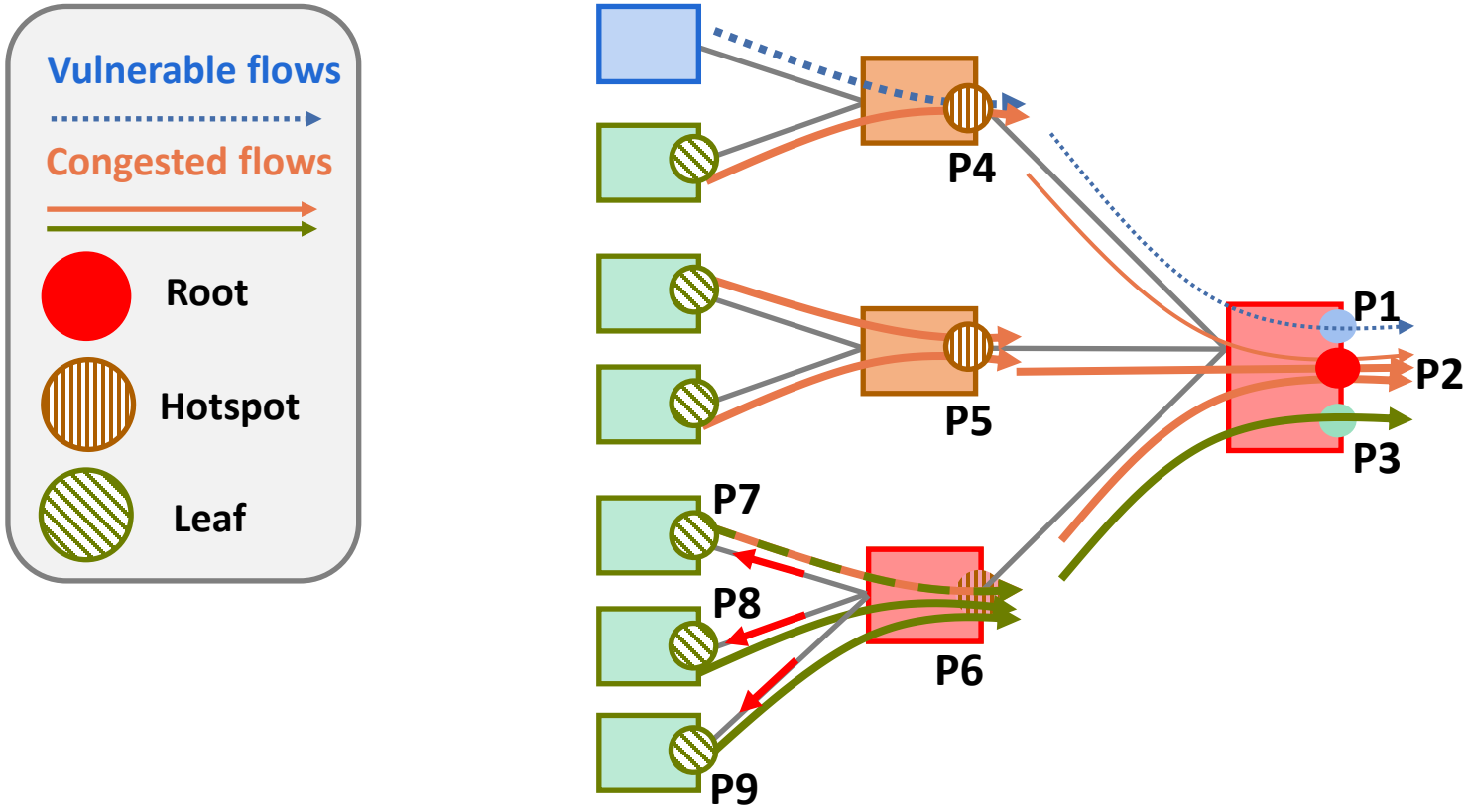
Challenge 3: Handle Tree-tangling Scenarios



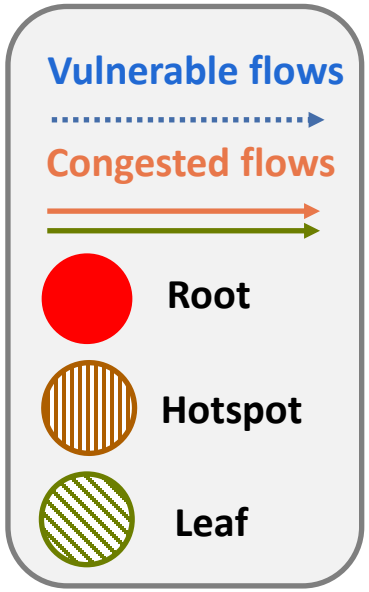
Challenge 3: Handle Tree-tangling Scenarios

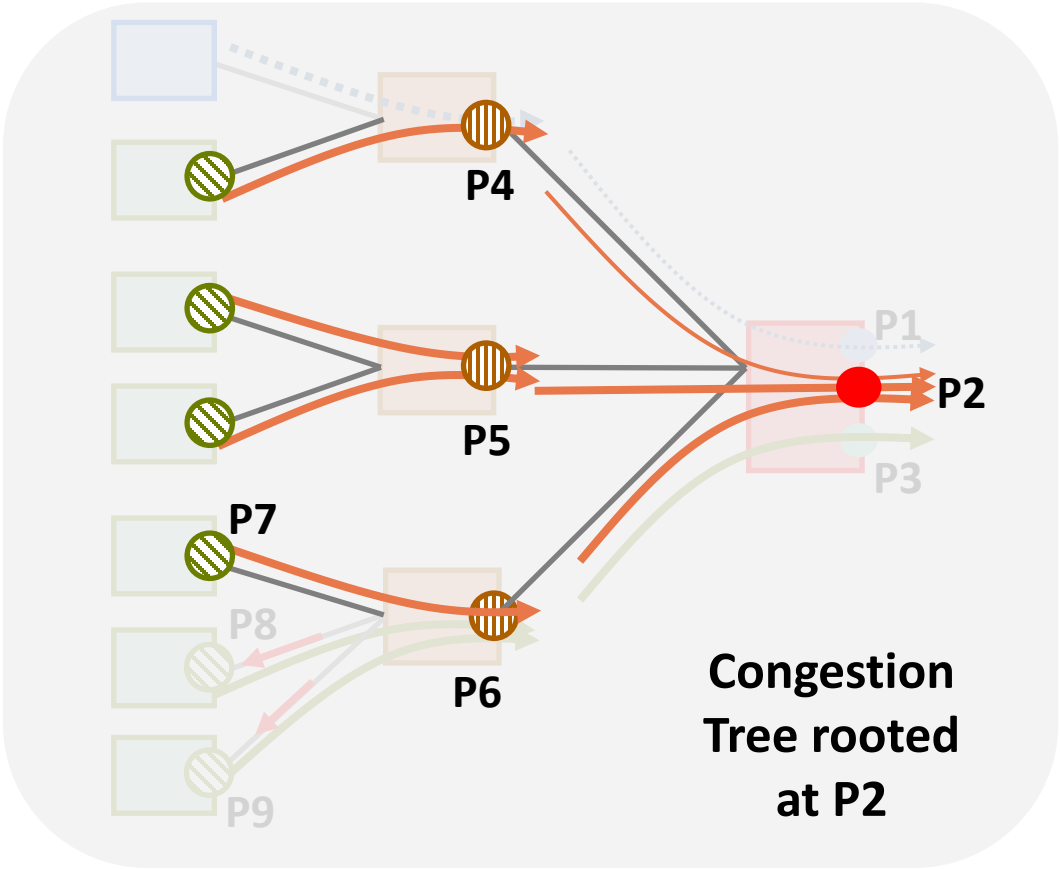


Challenge 3: Handle Tree-tangling Scenarios

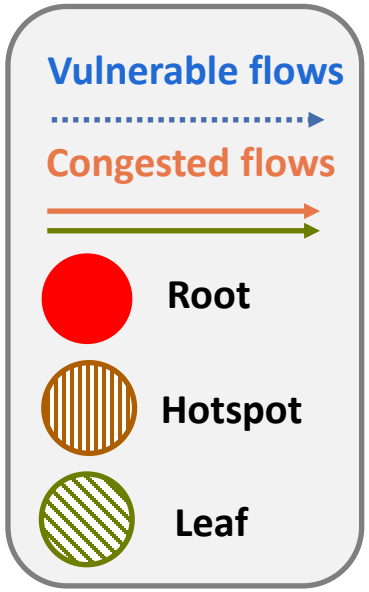


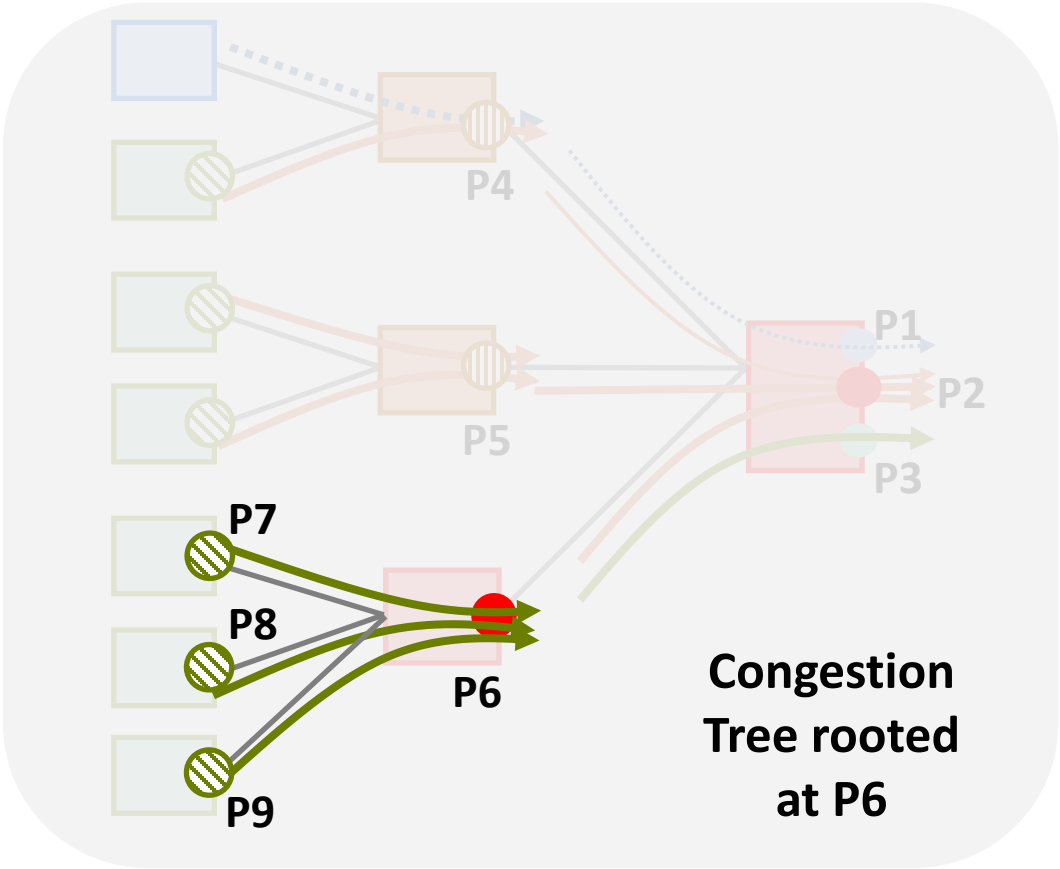
Challenge 3: Handle Tree-tangling Scenarios



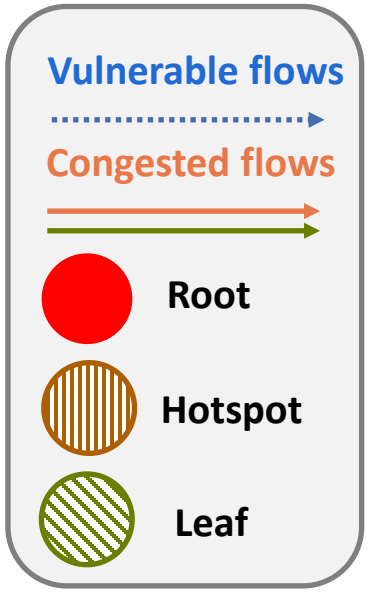


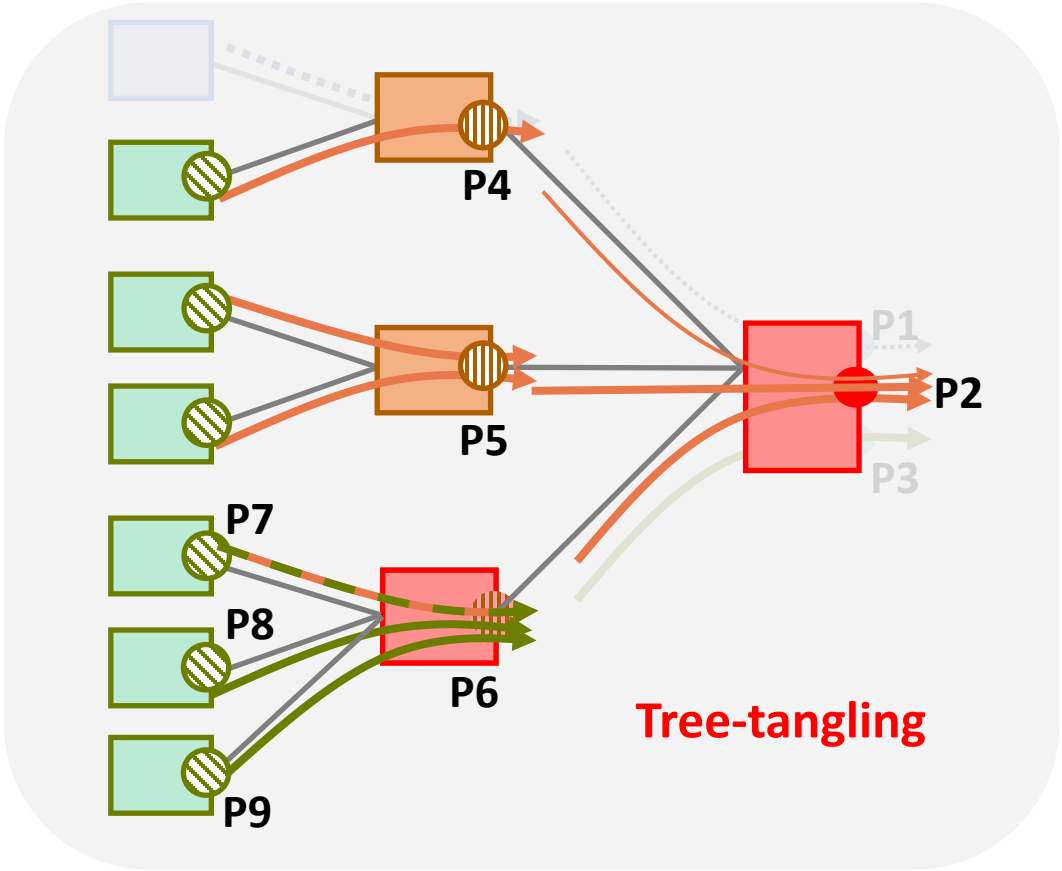
Challenge 3: Handle Tree-tangling Scenarios



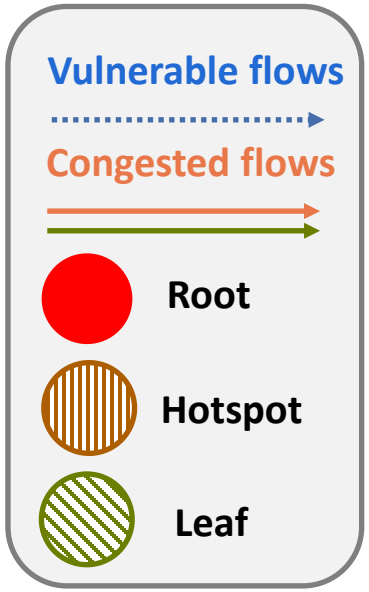


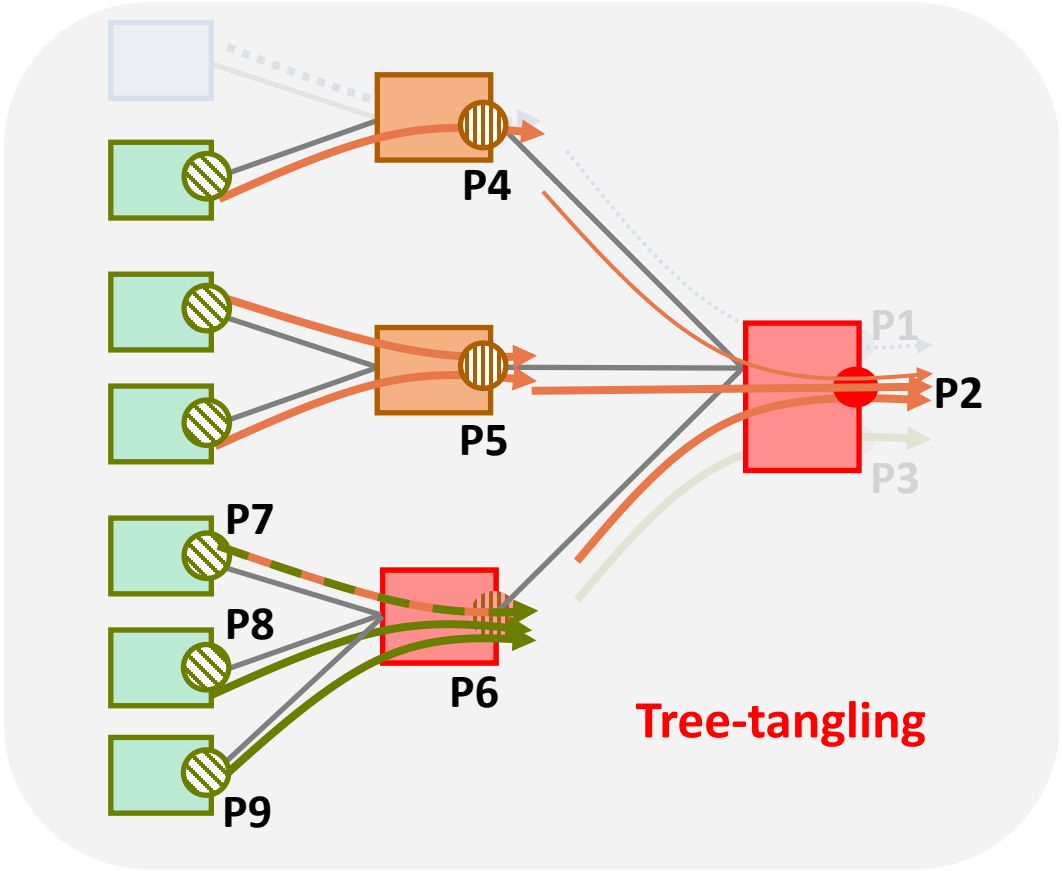
Challenge 3: Handle Tree-tangling Scenarios



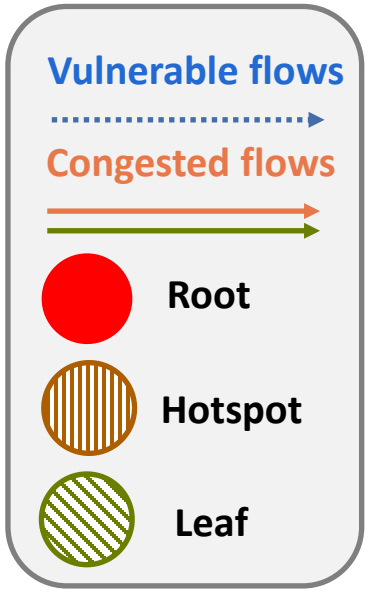


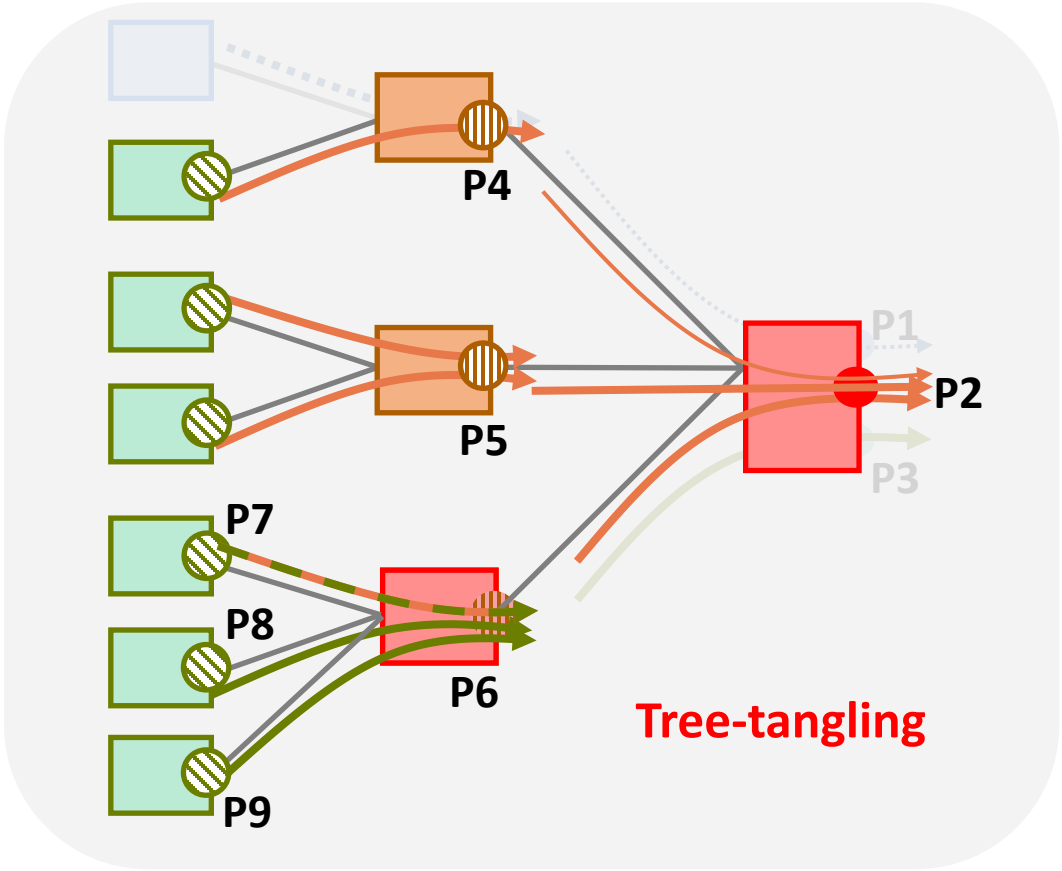
Challenge 3: Handle Tree-tangling Scenarios





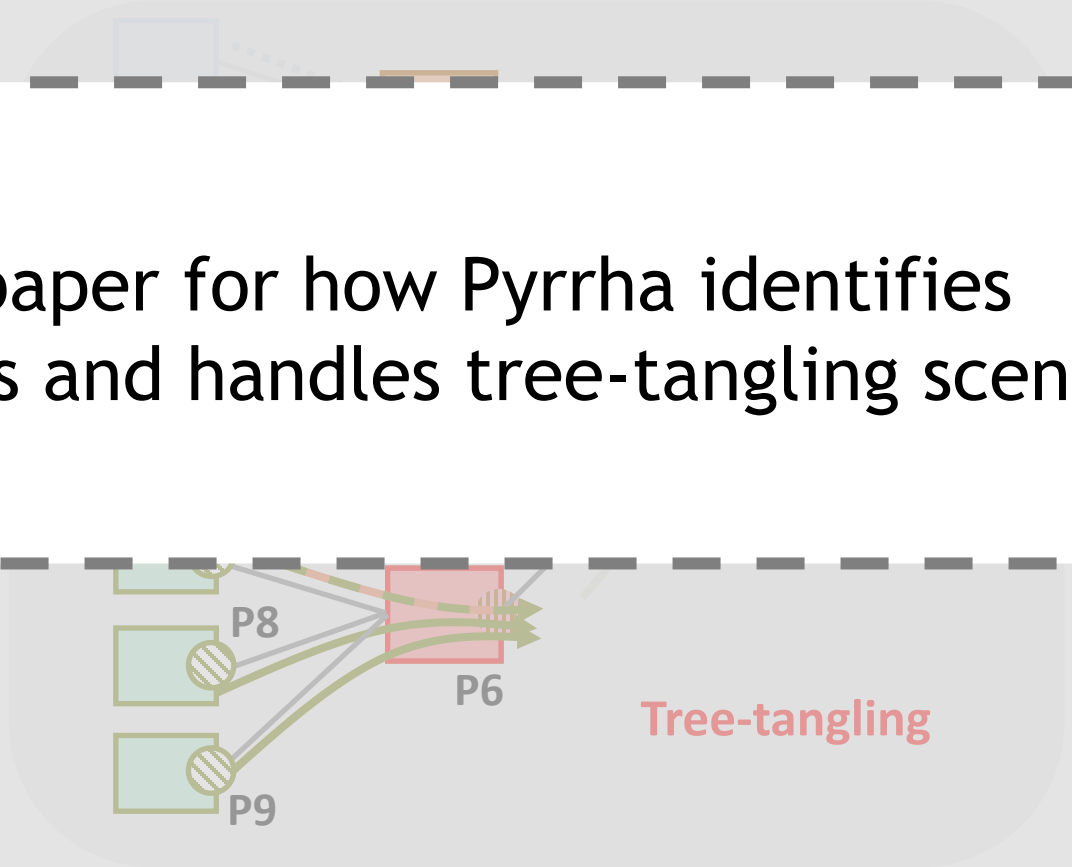
Challenge 3: Handle Tree-tangling Scenarios





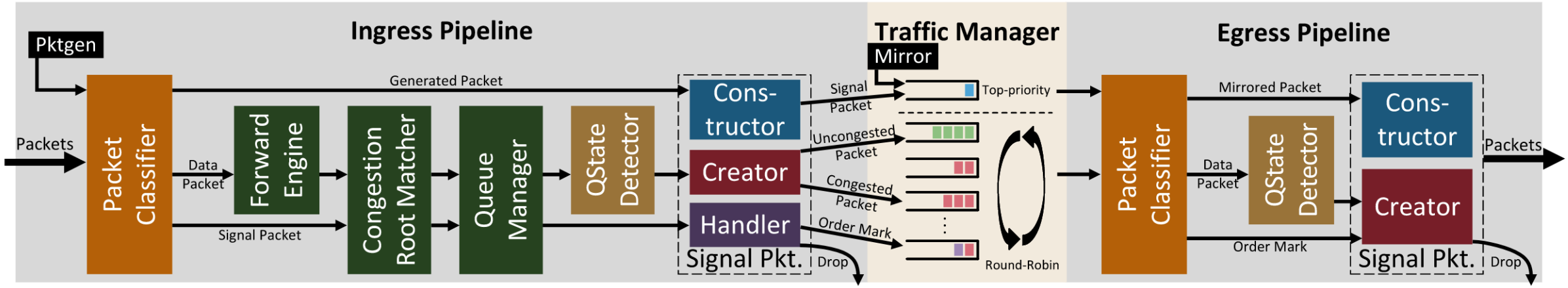
Challenge 3: Handle Tree-tangling Scenarios

Please see our paper for how Pyrrha identifies congestion flows and handles tree-tangling scenarios.



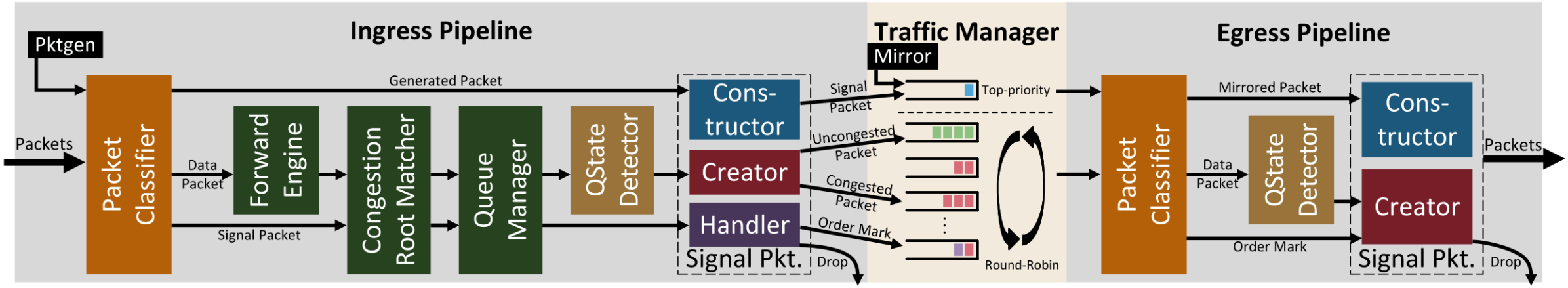
Pyrrha Prototype and Overhead Analysis

We implement a Pyrrha prototype on **Tofino2** with 2.5k lines of P4 code and 2k lines of Python code.



Pyrrha Prototype and Overhead Analysis

We implement a Pyrrha prototype on **Tofino2** with 2.5k lines of P4 code and 2k lines of Python code.

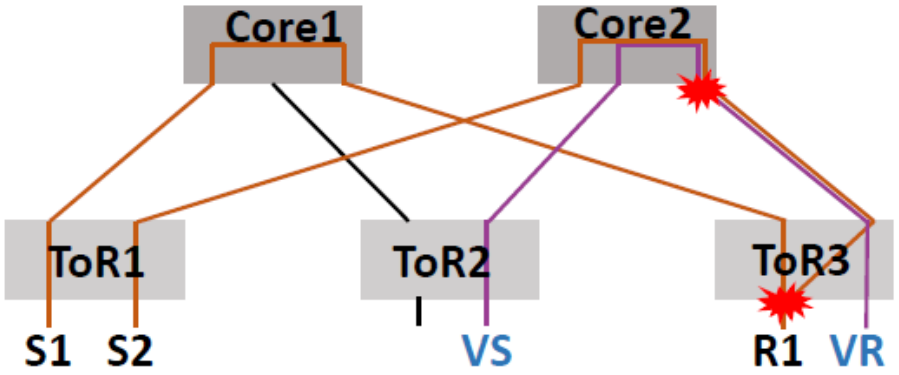


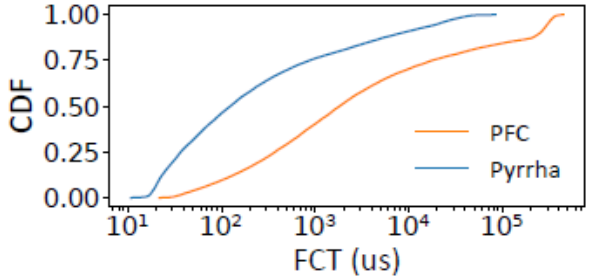
Our Pyrrha prototype can easily scale to a k=36 fat-tree topology (≈10k hosts), with around 11 MB (44.5% of Tofino2) memory resource consumption.

Testbed Evaluation

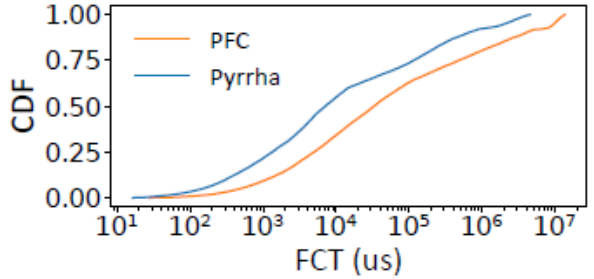
Pyrrha significantly reduces the FCT of vulnerable flows compared with PFC.

Topology: 2-stage clos topology with 2 cores and 3 ToRs.
Workload: Web Server and Web Search.





(a) Vulnerable flows (Web Server).

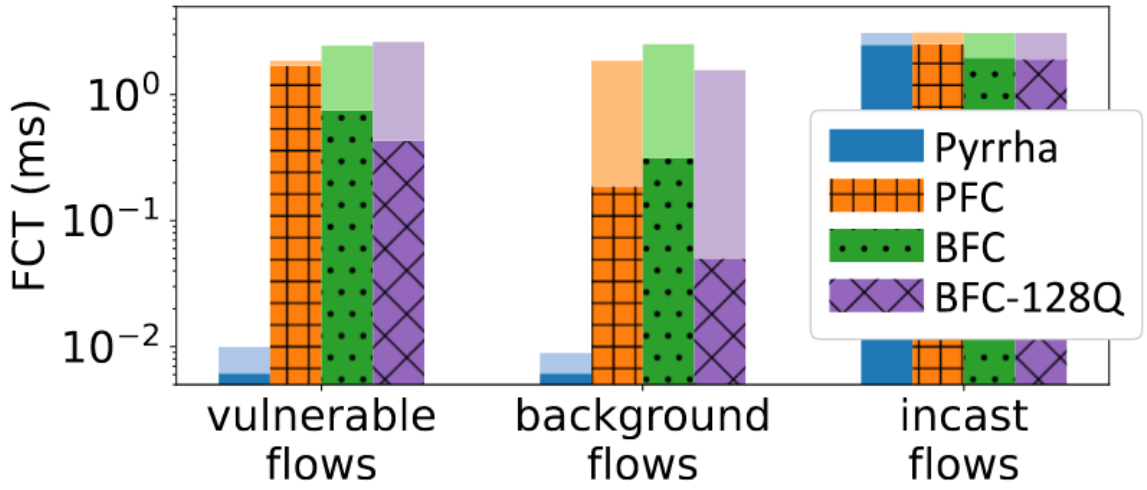


(b) Vulnerable flows (Web Search).

Simulation Evaluation

FCT in incast-mix scenario, compare with other FCs

Topology: 2-stage clos topology with 160 hosts.
Incast flows: periodically generated with average load at 0.5.
Vulnerable and background flows: generated following Poisson process with load at 0.8.

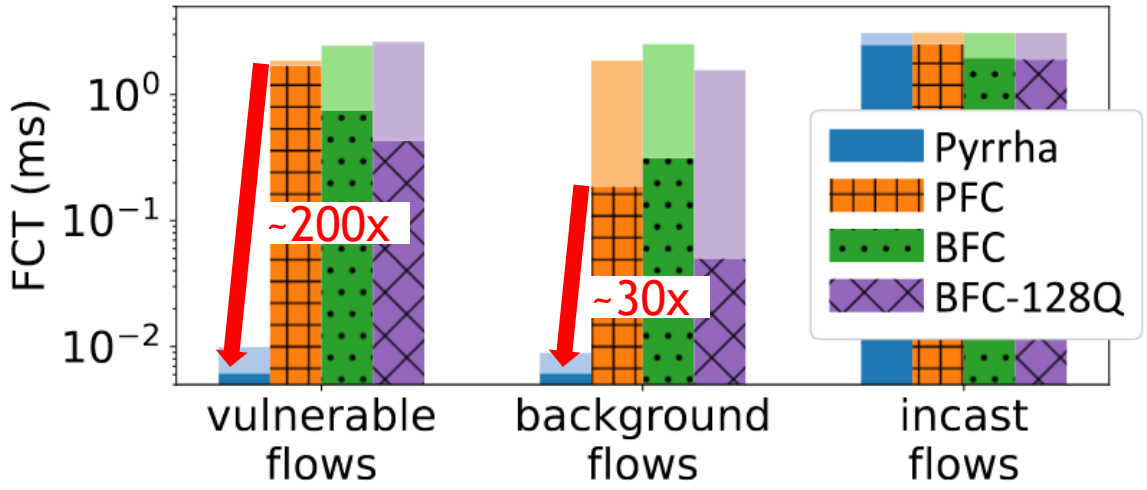


(a) Memcached (overall performance).

Simulation Evaluation

FCT in incast-mix scenario, compare with other FCs

Topology: 2-stage clos topology with 160 hosts.
Incast flows: periodically generated with average load at 0.5.
Vulnerable and background flows: generated following Poisson process with load at 0.8.

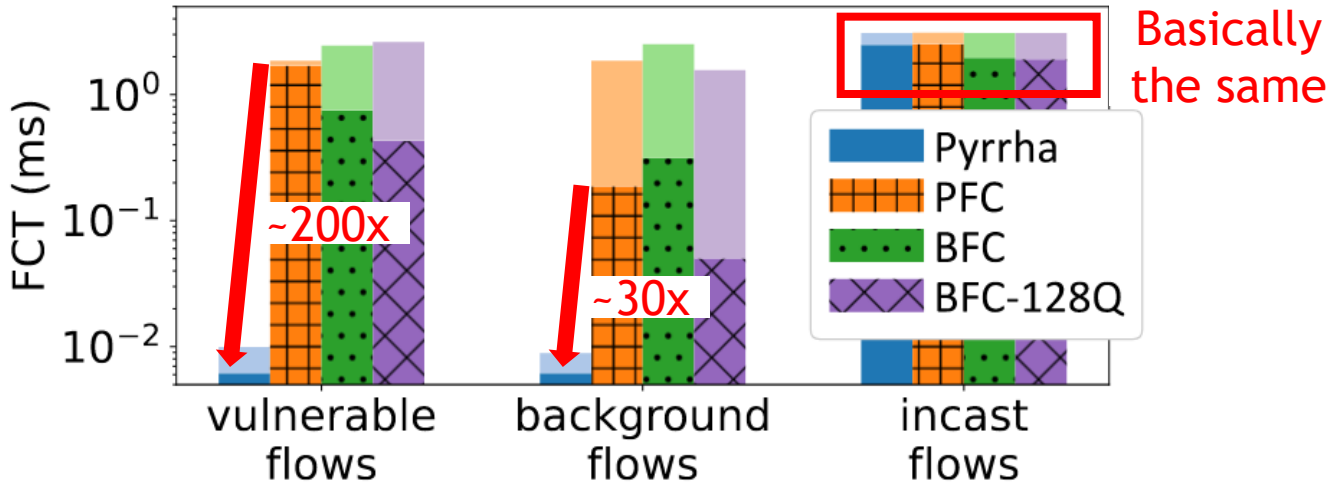


(a) Memcached (overall performance).

Simulation Evaluation

FCT in incast-mix scenario, compare with other FCs

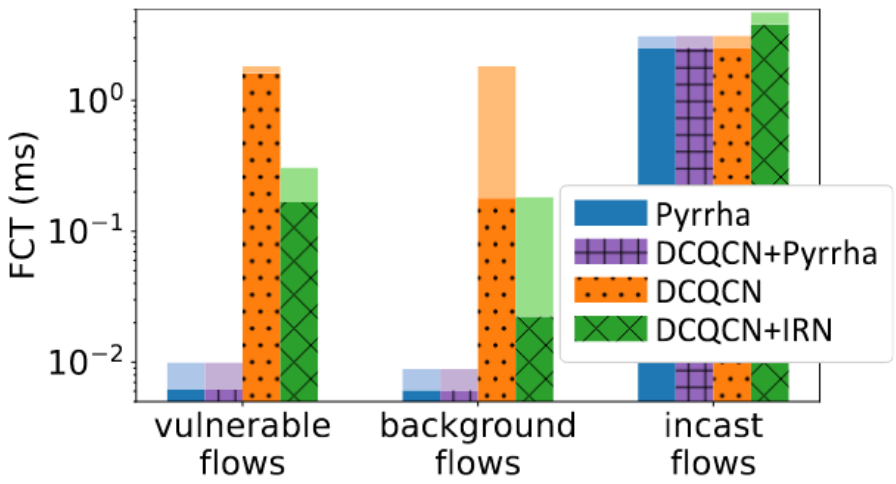
Topology: 2-stage clos topology with 160 hosts.
Incast flows: periodically generated with average load at 0.5.
Vulnerable and background flows: generated following Poisson process with load at 0.8.



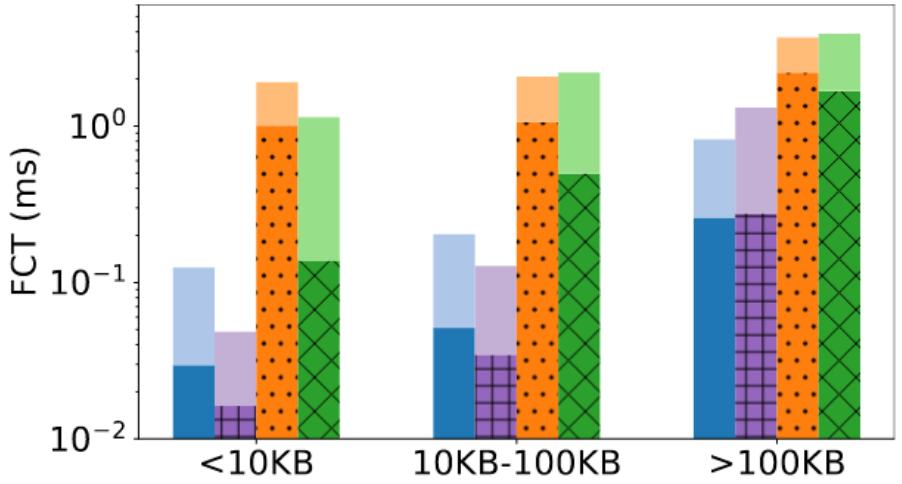
(a) Memcached (overall performance).

Simulation Evaluation

FCT in incast-mix scenario, with or without CC



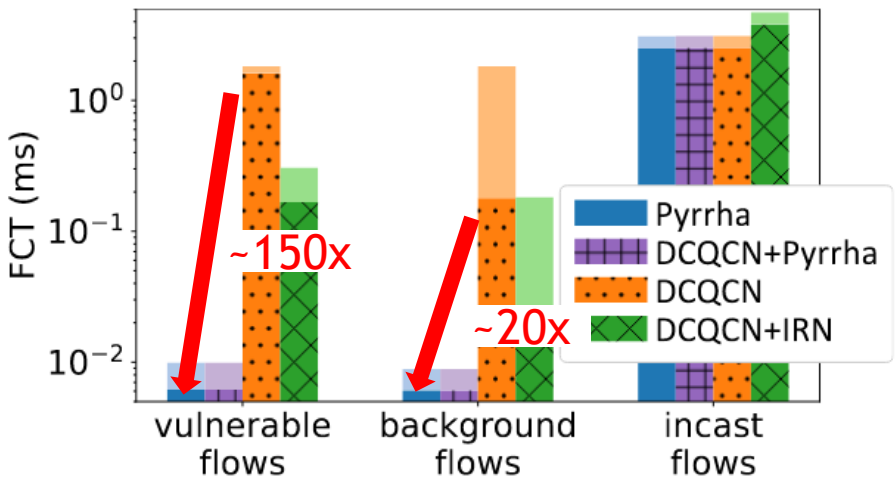
(a) Memcached (overall flows).



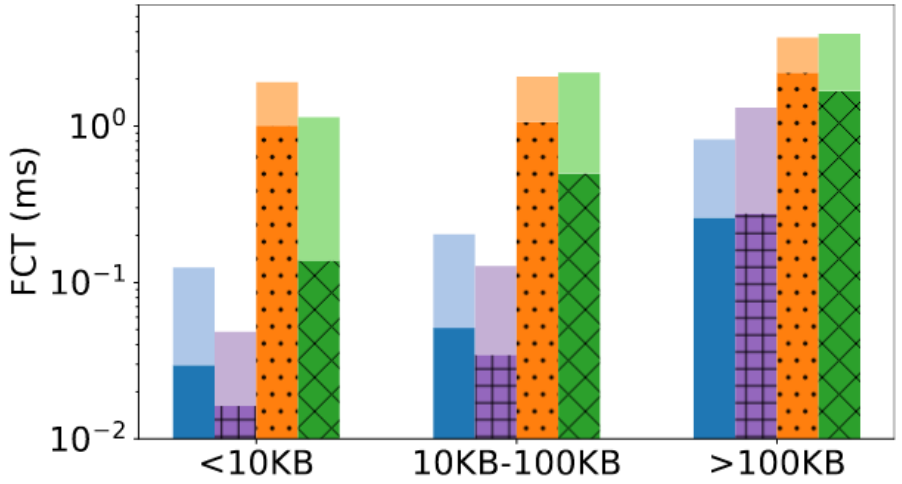
(b) Web Server (vulnerable flows).

Simulation Evaluation

FCT in incast-mix scenario, with or without CC



(a) Memcached (overall flows).

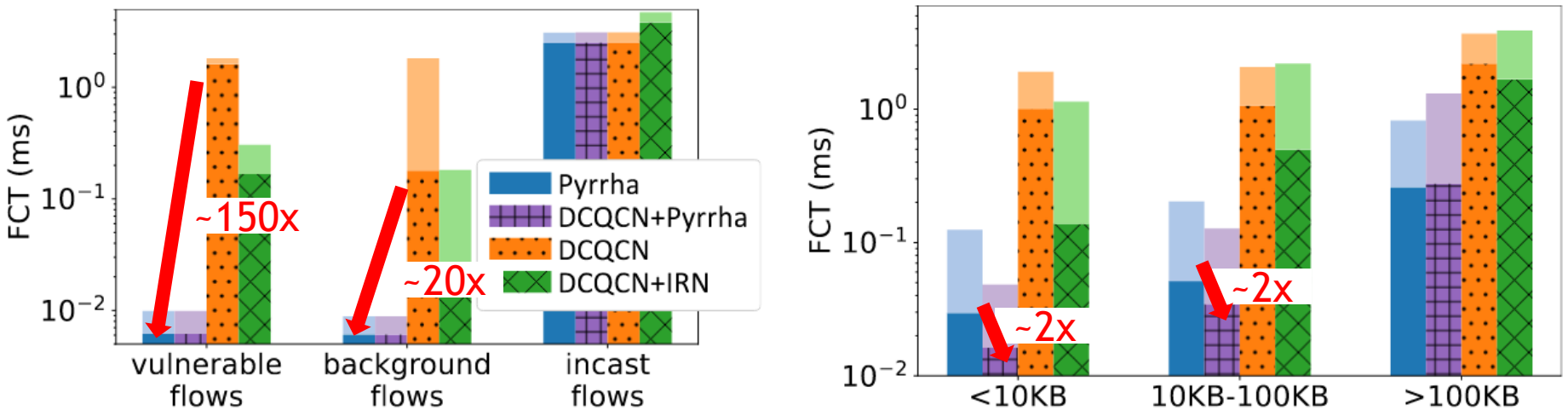


(b) Web Server (vulnerable flows).

Pyrrha significantly benefits uncongested flows compare with pure CC solution.

Simulation Evaluation

FCT in incast-mix scenario, with or without CC



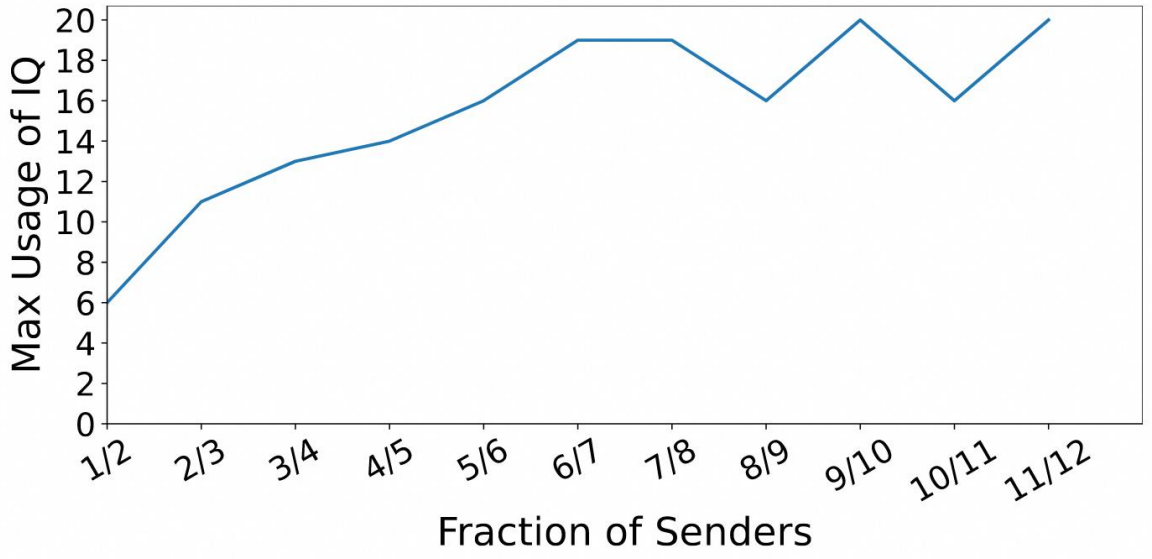
(a) Memcached (overall flows). (b) Web Server (vulnerable flows).

Pyrrha significantly benefits uncongested flows compare with pure CC solution. Pyrrha can cooperate with existing CC to achieve better performance for small flows.

Simulation Evaluation

Pyrrha’s queue usage is moderate

Topology: standard fat-tree with k=16 (1024 hosts).
Most stressful workload: (m − 1) out of m ToRs send traffic simultaneously to the left 1/m of the ToRs.



Conclusion

Pyrrha is a congestion-root-based per-hop flow control, which can control the transmission of flows at a fine granularity without HOL blocking while requiring only a minimal number of queues.

It is time to embrace per-hop flow control to react to congestion promptly!

Thanks !

Find our open-source code on: *https://github.com/NASA-NJU/Pyrrha*