LSU College of Engineering
School of Electrical Engineering & Computer Science

PathFence: Reducing Cross-Path Dependencies in Microservices

Xuhang Gu and Qingyang Wang

Louisiana State University

From Monolithic N-tier to Microservices

N-tier applications

Microservice applications

From Monolithic N-tier to Microservices

N-tier applications

Coupled & Central

web tier

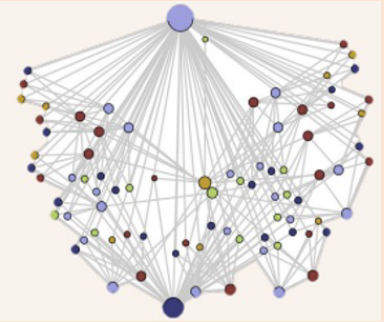
Business tier

Database tier

Shifting to

Microservice applications

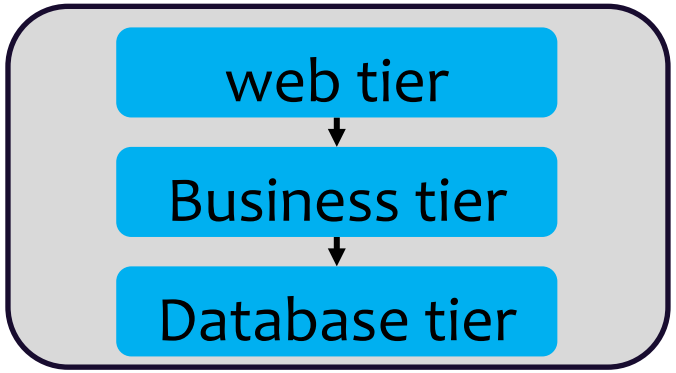
Decoupled & Distributed



From Monolithic N-tier to Microservices

N-tier applications

Coupled & Central

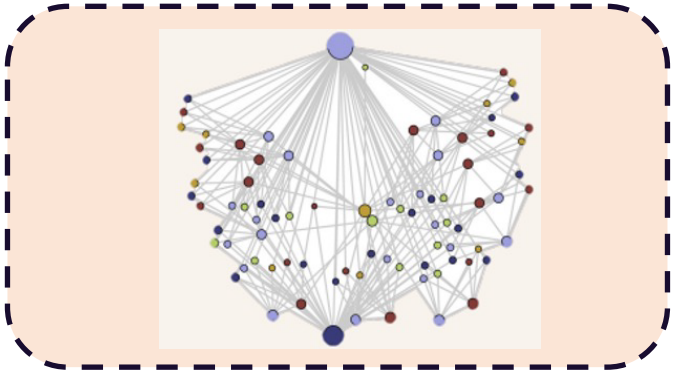


- X Centralized deployment
- X Coarse management
- X fixed programing language

Shifting to

Microservice applications

Decoupled & Distributed

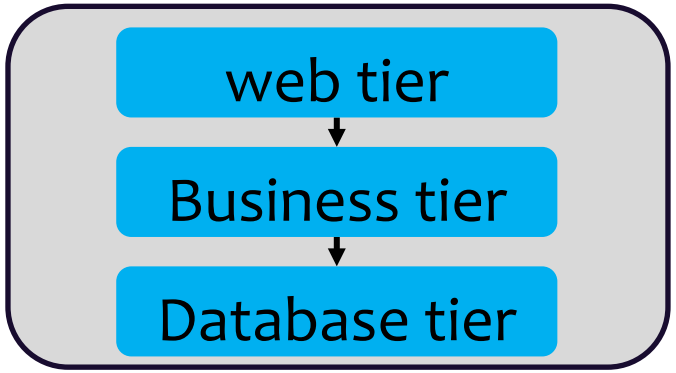


- ✓ Independent development
- ✓ Fine-grained management
- ✓ Flexible programming languages

From Monolithic N-tier to Microservices

N-tier applications

Coupled & Central

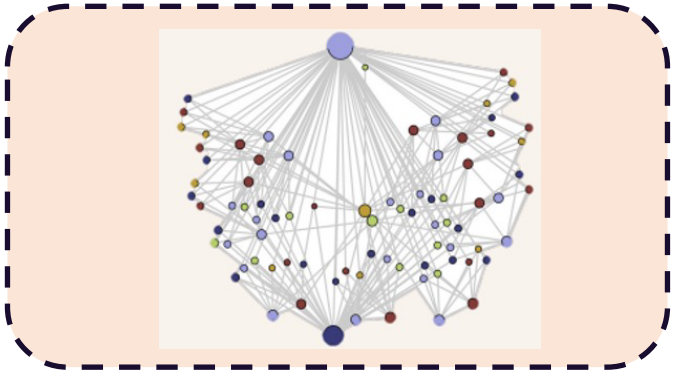


- X Centralized deployment
- X Coarse management
- X fixed programing language

Shifting to

Microservice applications

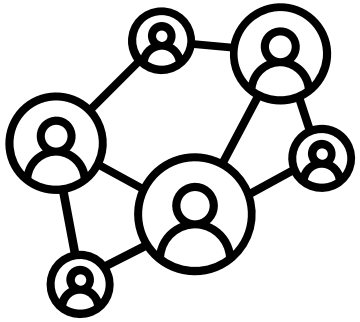
Decoupled & Distributed



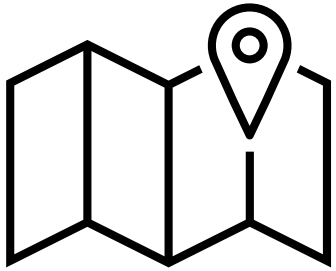
- ✓ Independent development
- ✓ Fine-grained management
- ✓ Flexible programming languages
- X Complex inter-service dependencies

Stringent Latency Requirement of Online Services

E-commerce



Social Media

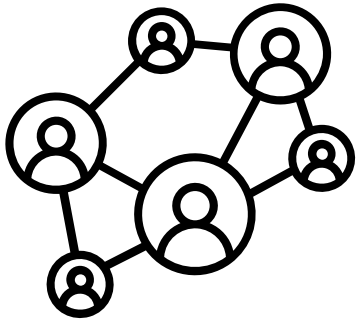


Online Navigation

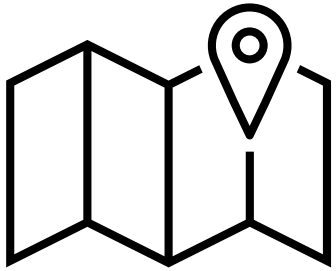
Stringent Latency Requirement of Online Services

- **Amazon:** 100ms latency increase → $0.7 billion loss –[Kohavi et al. *Computer' 07]*

E-commerce



Social Media

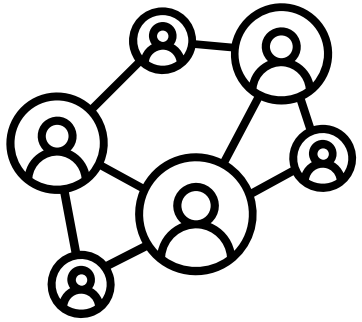


Online Navigation

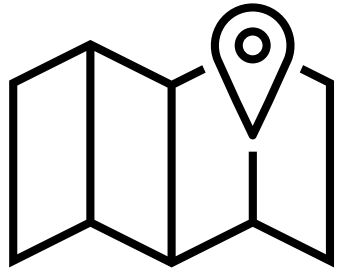
Stringent Latency Requirement of Online Services

- **Amazon:** 100ms latency increase → $0.7 billion loss –*[Kohavi et al. Computer' 07]*
- **Alibaba:** adopting real-time data analytics for Singles' Day advertising → a 30% increase in purchase conversion rate –*[Jiang et al. AliReport'17]*

E-commerce



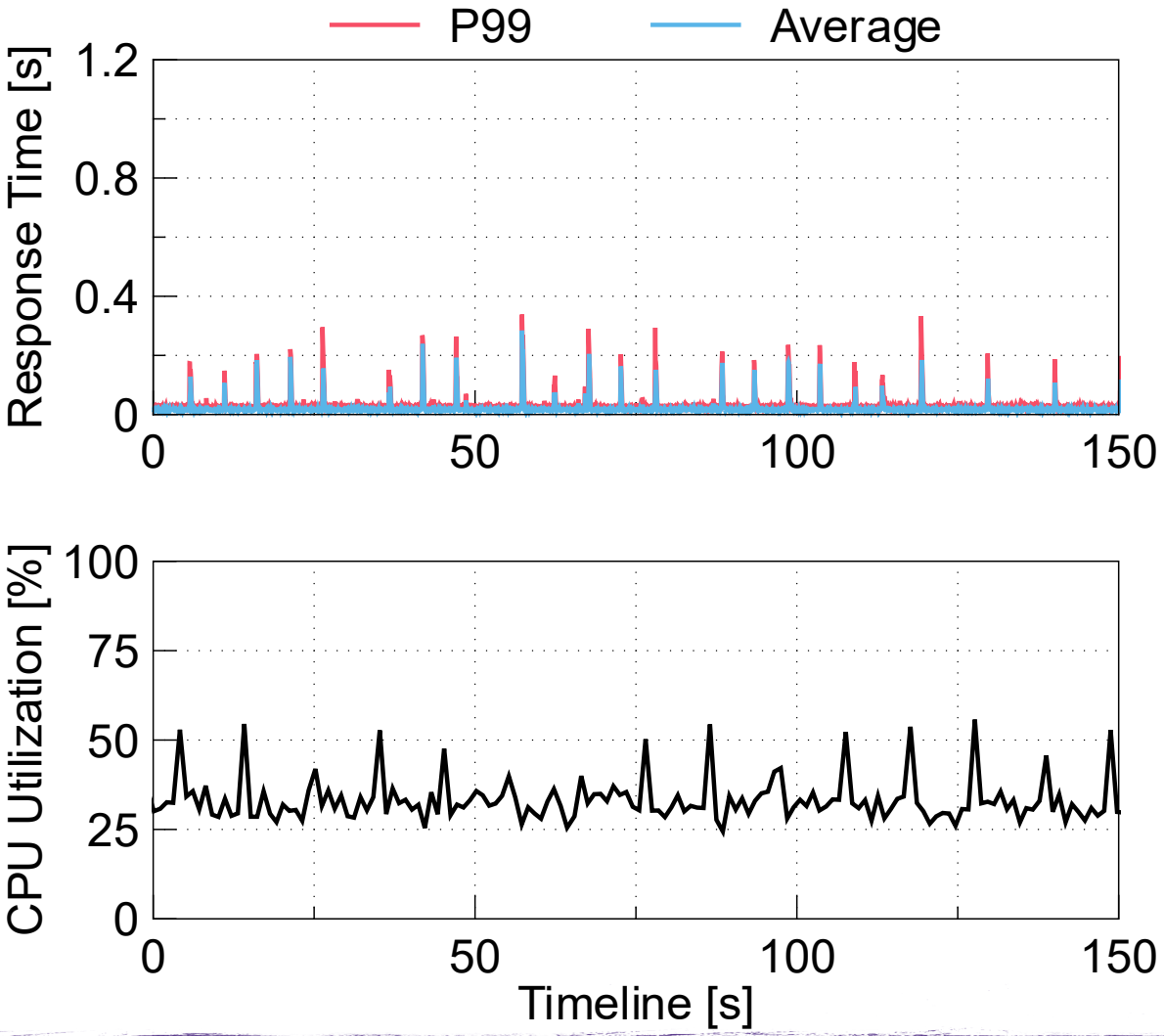
Social Media



Online Navigation

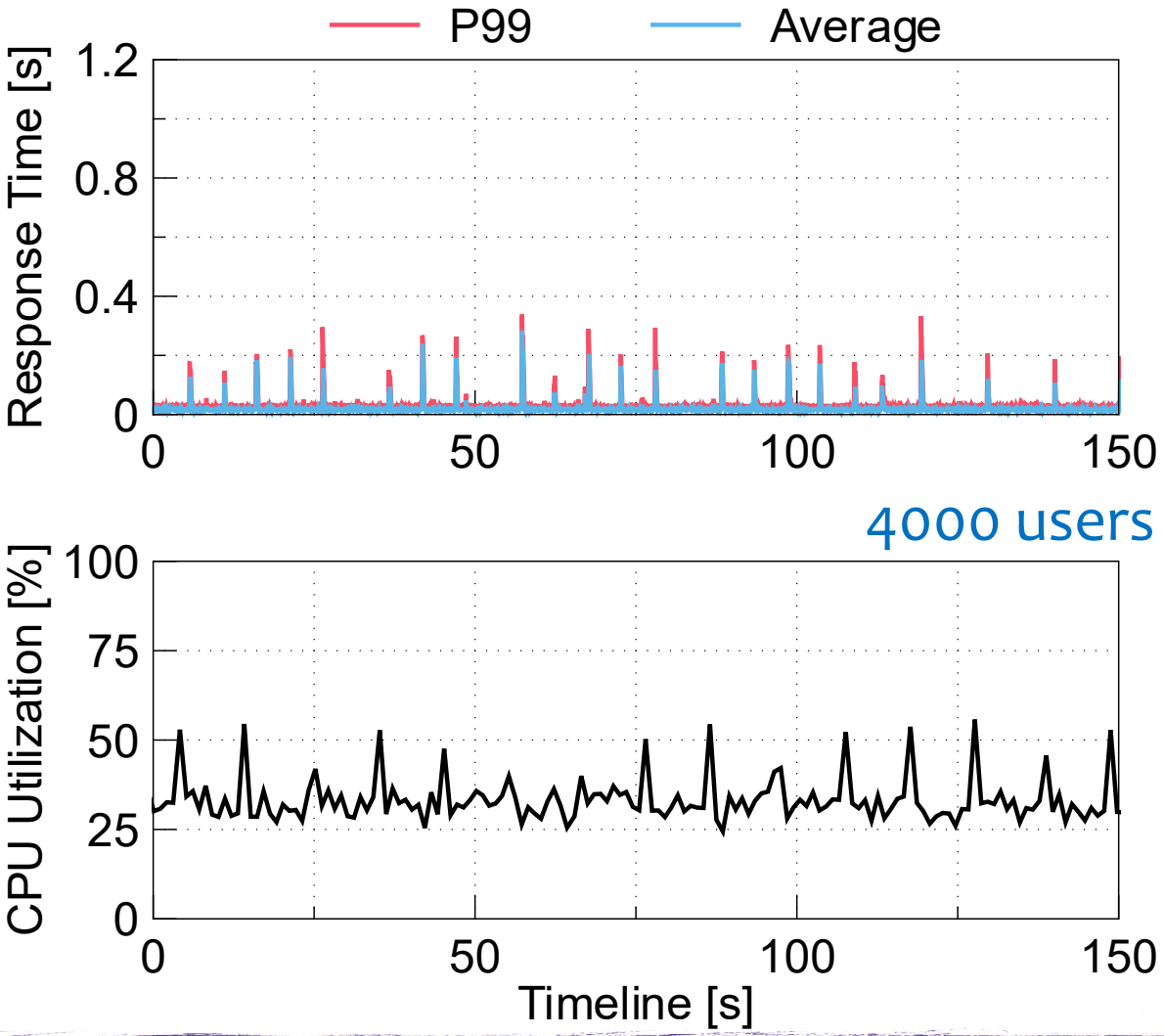
Research Problem

- The challenge: Fragile response time stability in cloud microservices



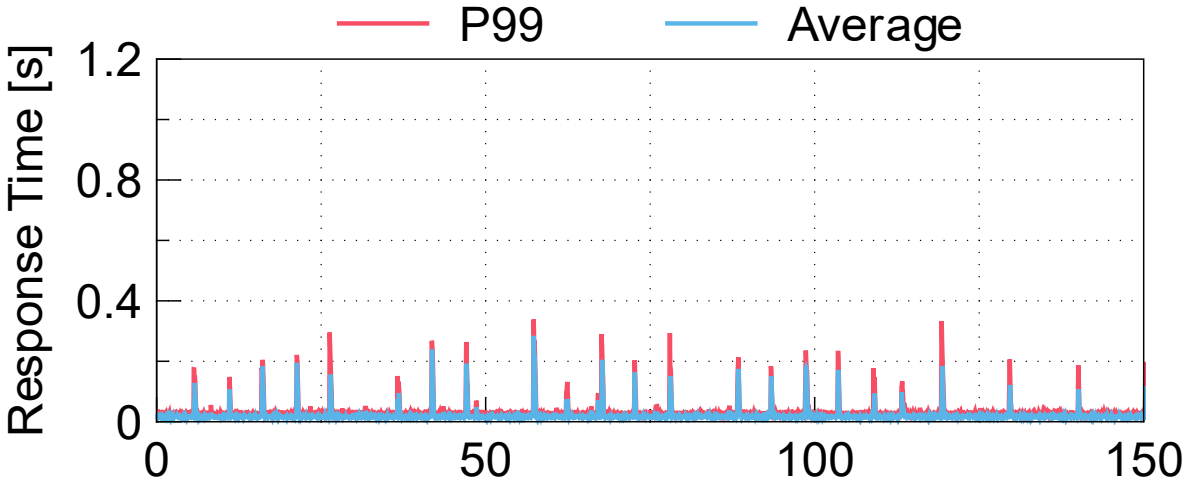
Research Problem

- The challenge: Fragile response time stability in cloud microservices

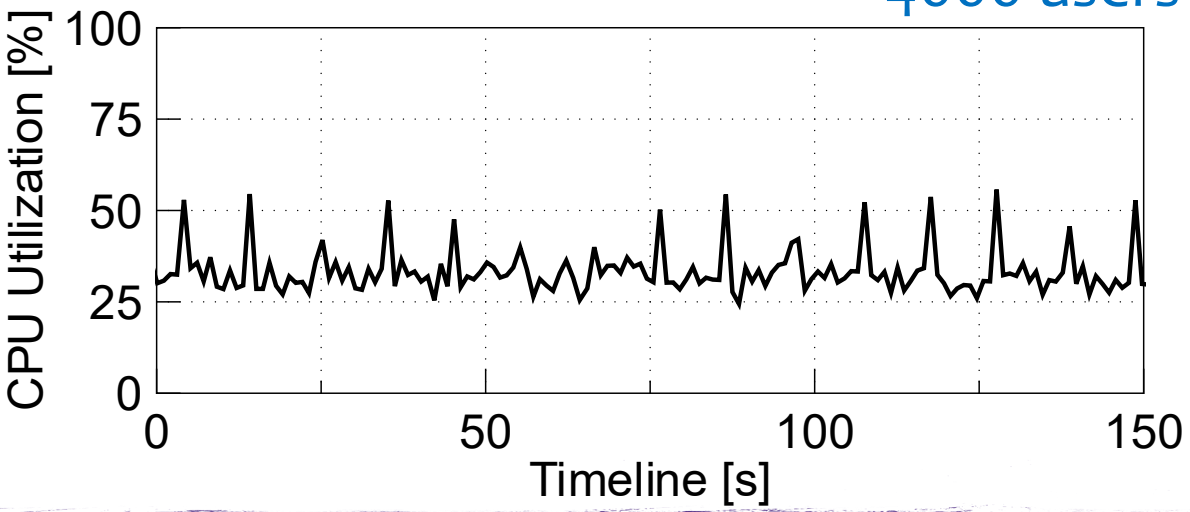


Research Problem

- The challenge: Fragile response time stability in cloud microservices

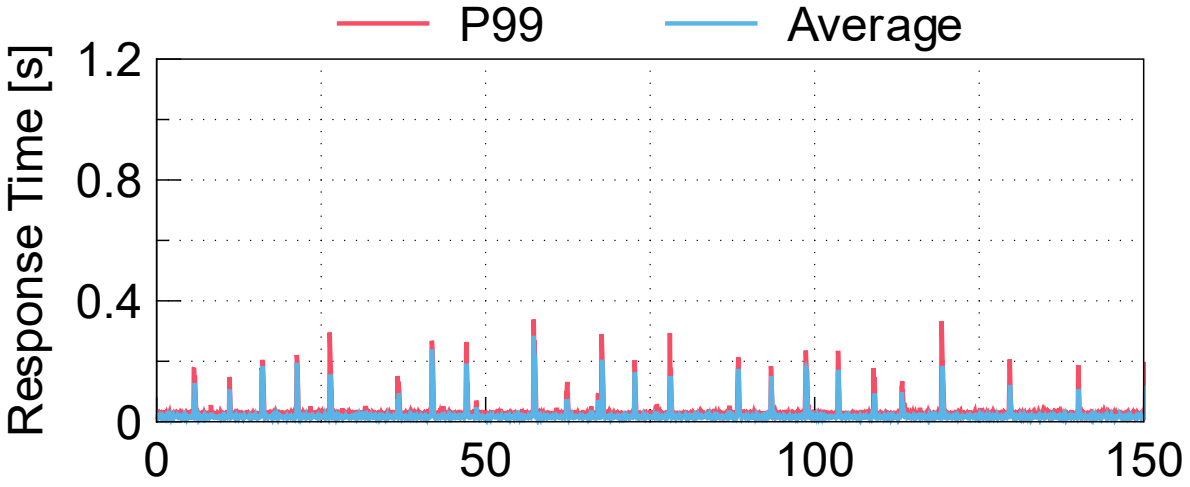


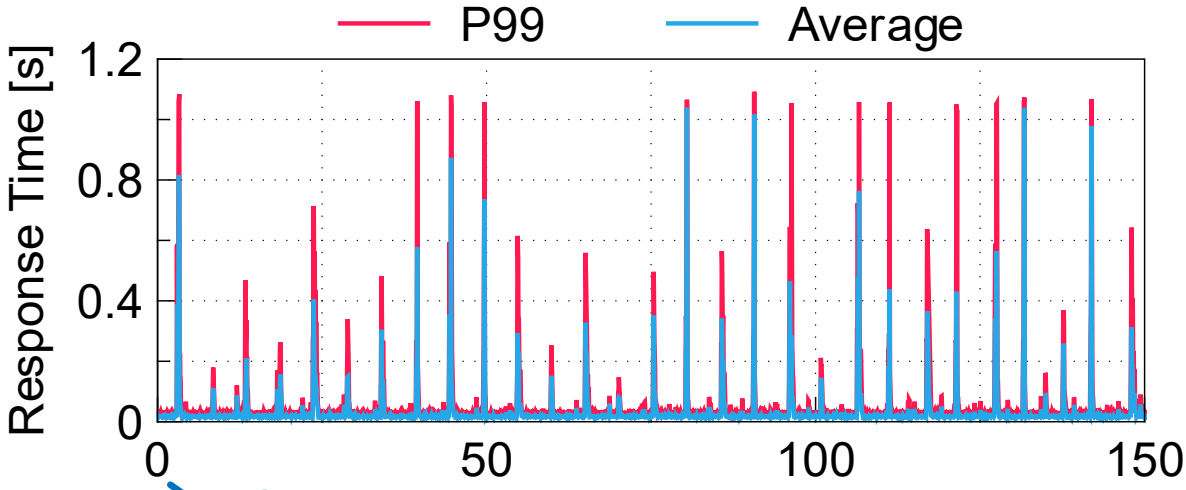
4000 users → 8000 users



Research Problem

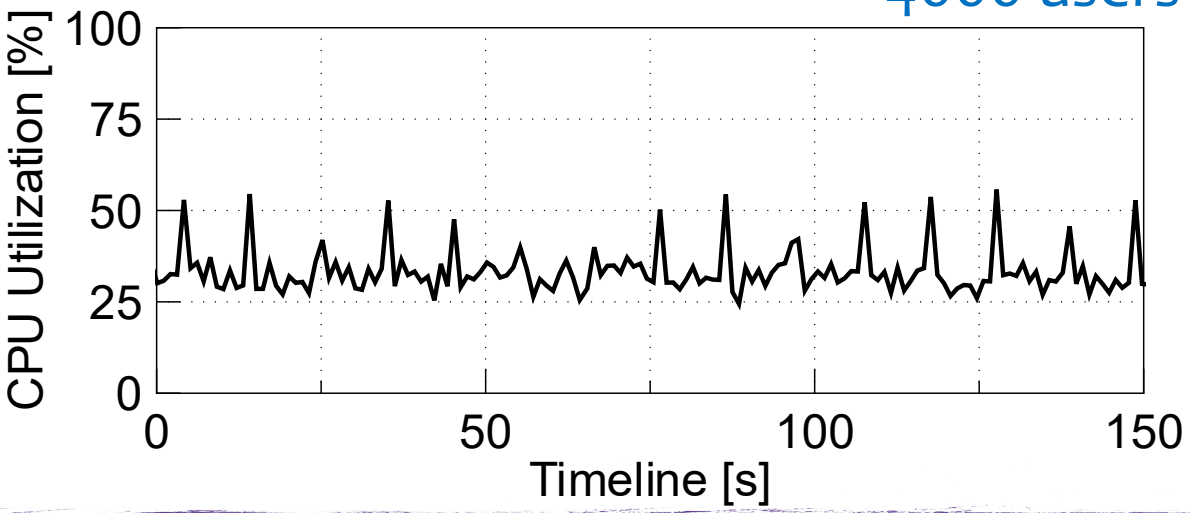
- The challenge: Fragile response time stability in cloud microservices

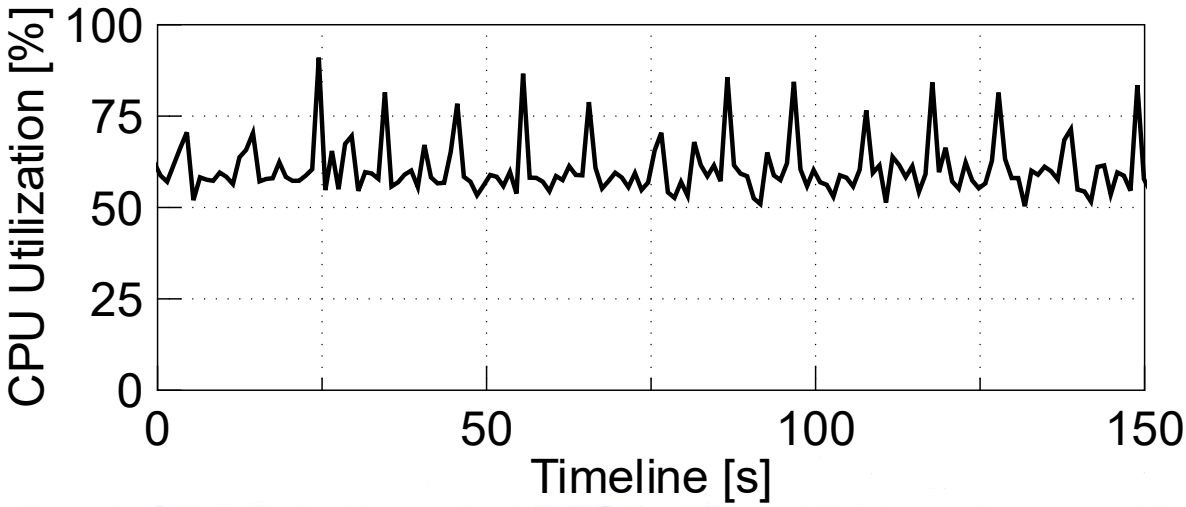




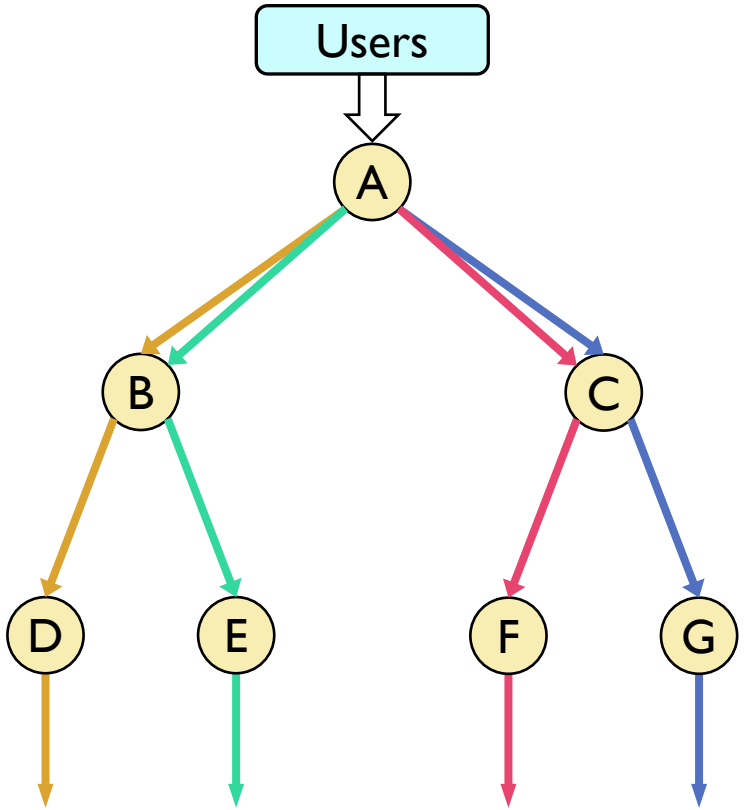
4000 users

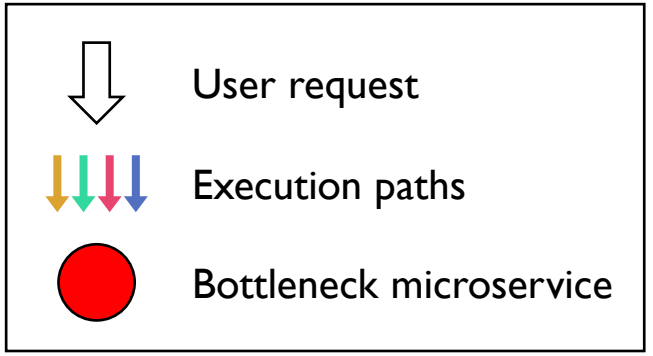
8000 users



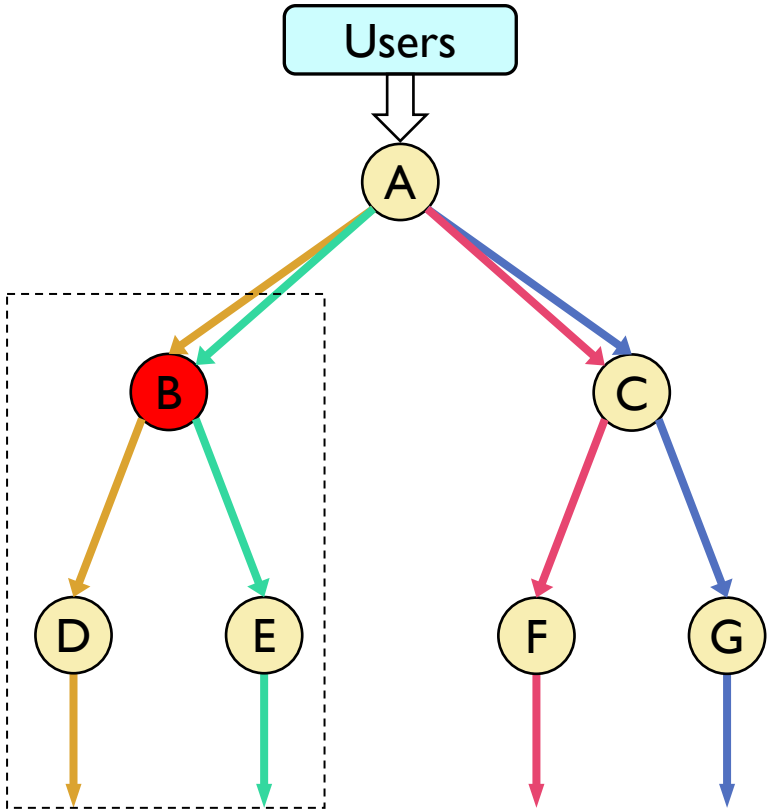


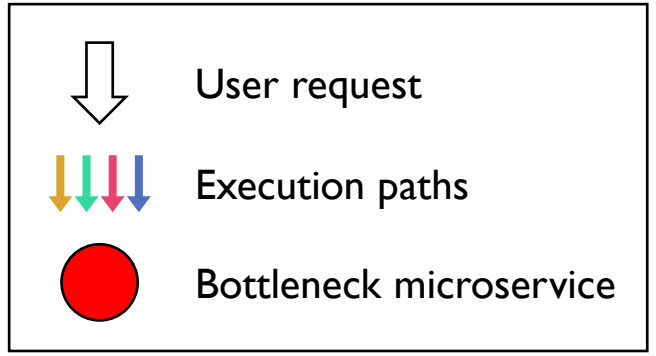
Cross-Path Dependencies Among Different Execution Paths





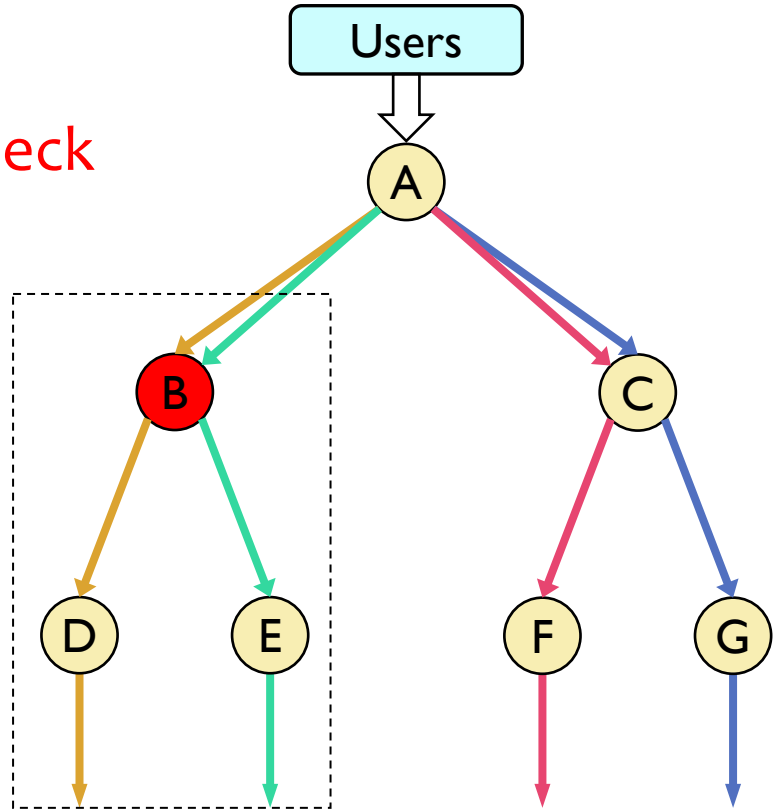
Cross-Path Dependencies Among Different Execution Paths

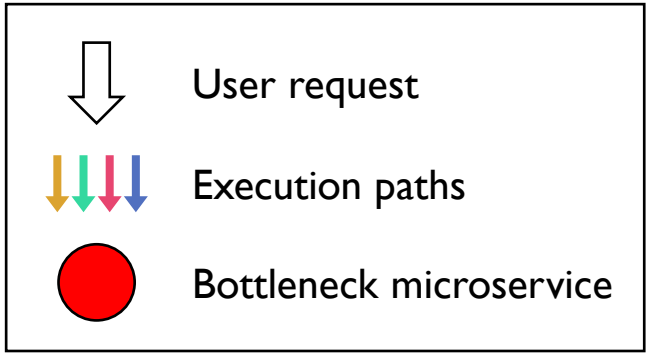




Cross-Path Dependencies Among Different Execution Paths

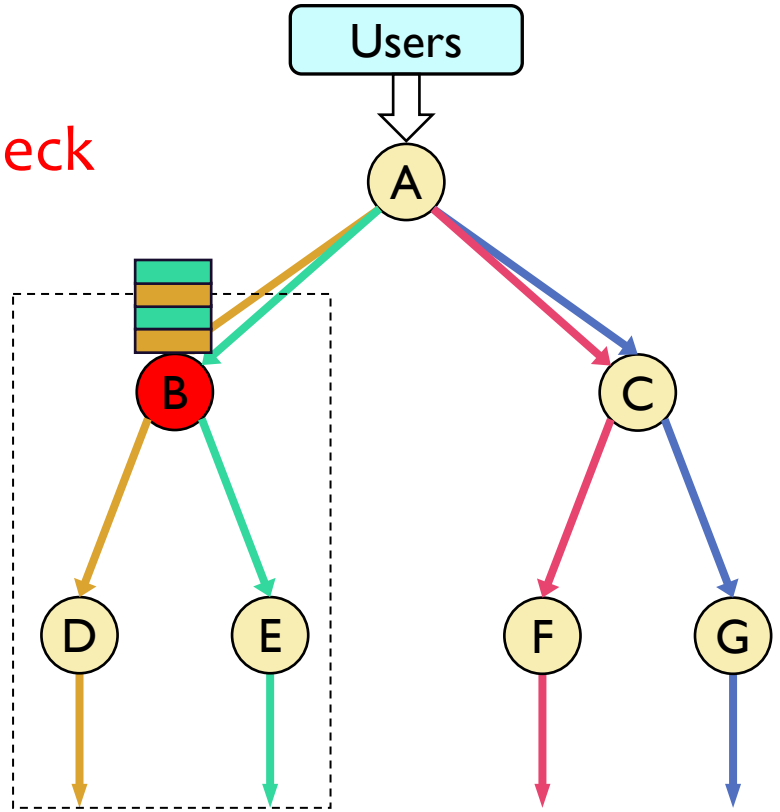
1. Shared bottleneck microservice

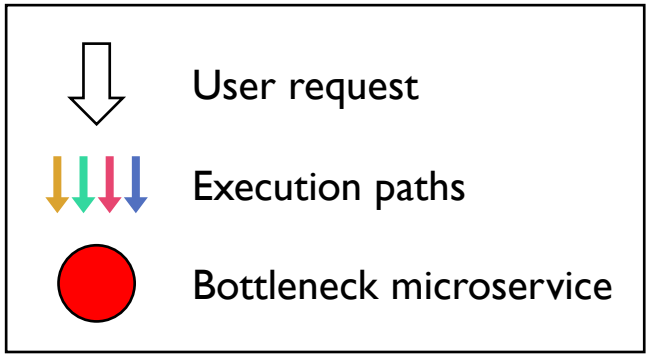




Cross-Path Dependencies Among Different Execution Paths

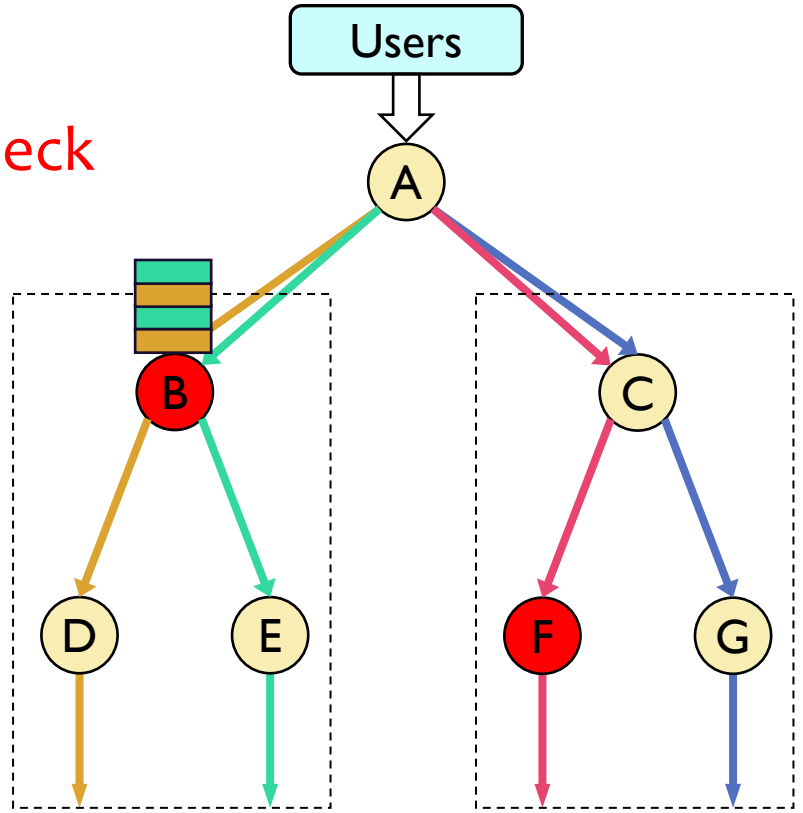
1. Shared bottleneck microservice

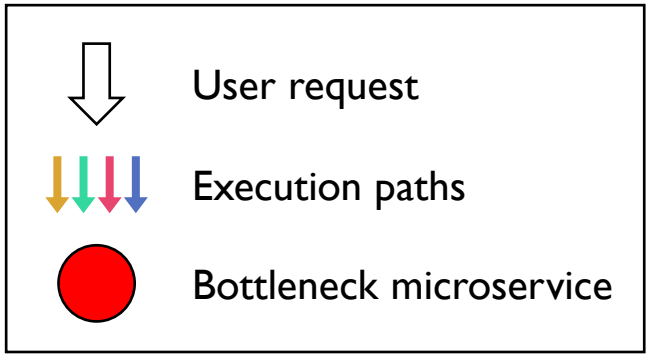




Cross-Path Dependencies Among Different Execution Paths

1. Shared bottleneck microservice

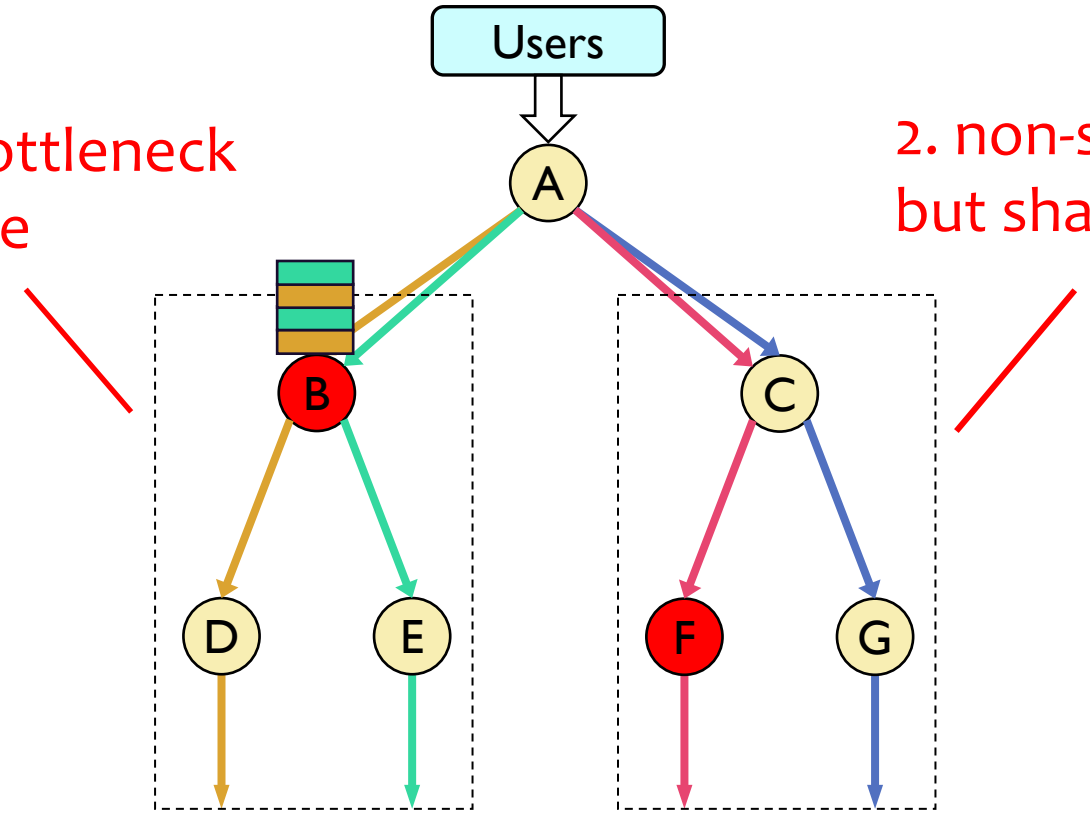


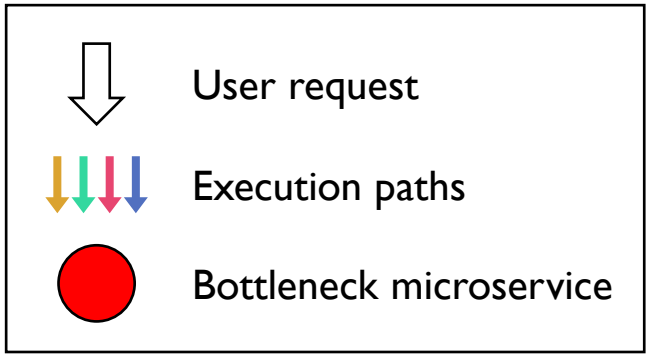


Cross-Path Dependencies Among Different Execution Paths

1. Shared bottleneck microservice

2. non-shared bottleneck microservice, but shared upstream service

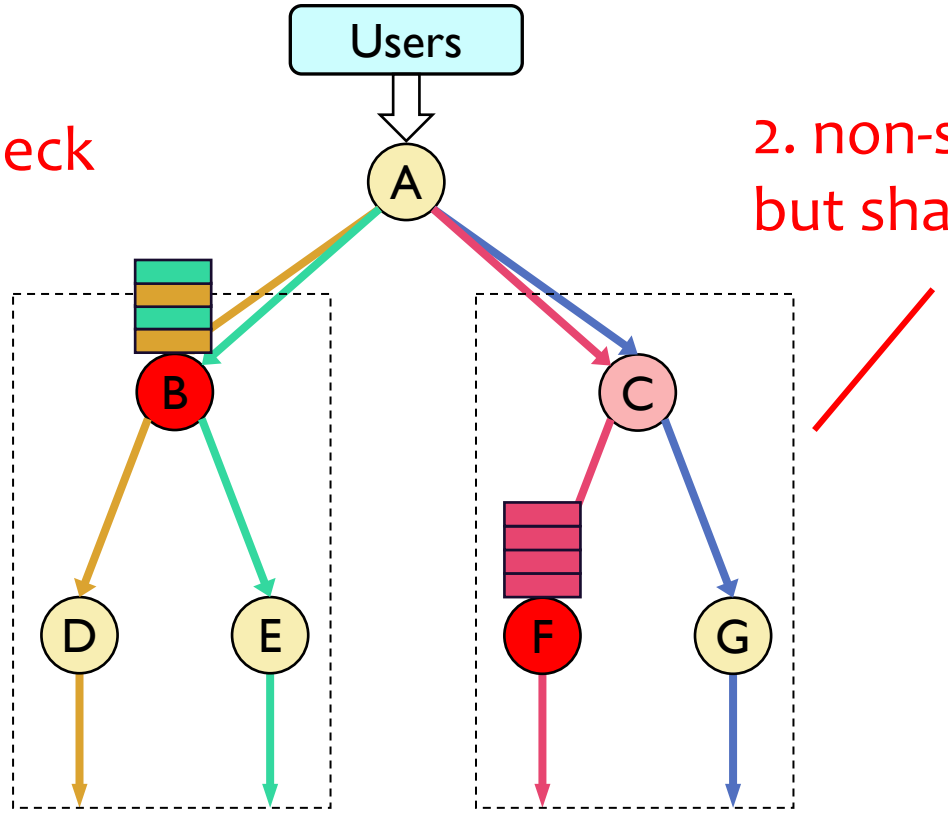


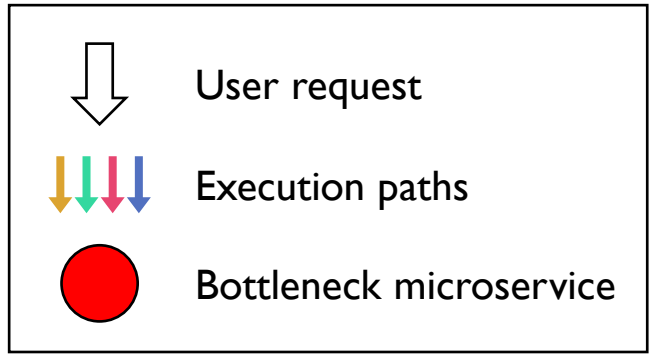


Cross-Path Dependencies Among Different Execution Paths

1. Shared bottleneck microservice

2. non-shared bottleneck microservice, but shared upstream service

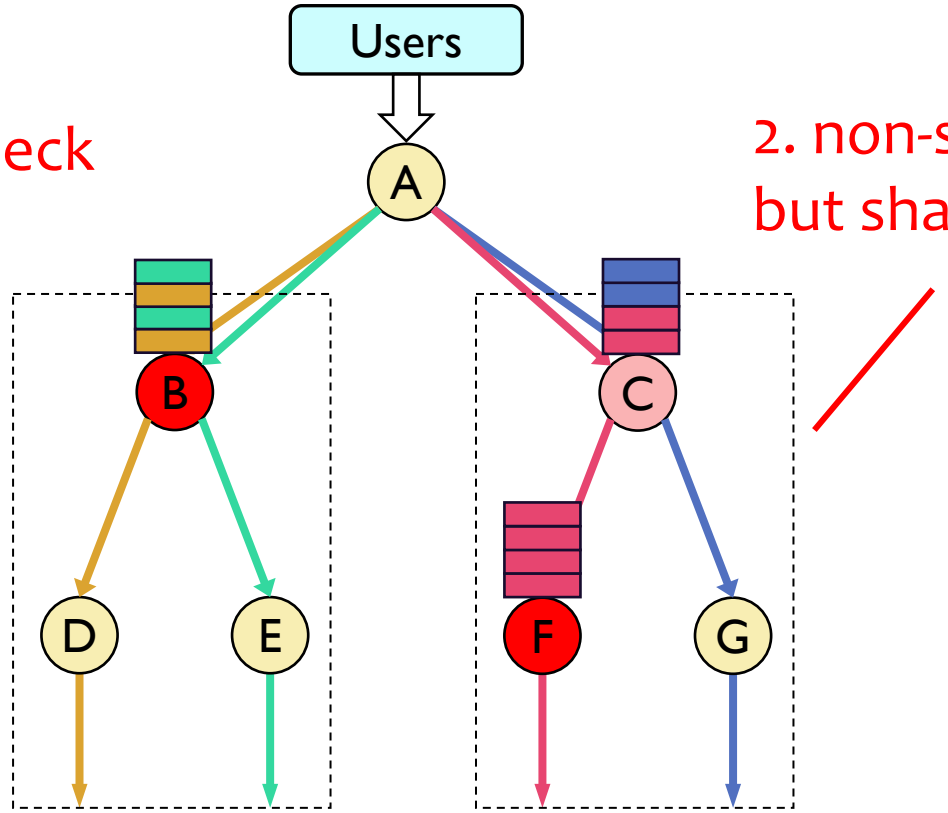


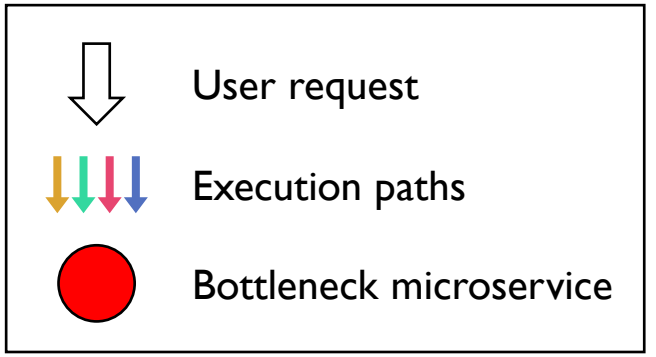


Cross-Path Dependencies Among Different Execution Paths

1. Shared bottleneck microservice

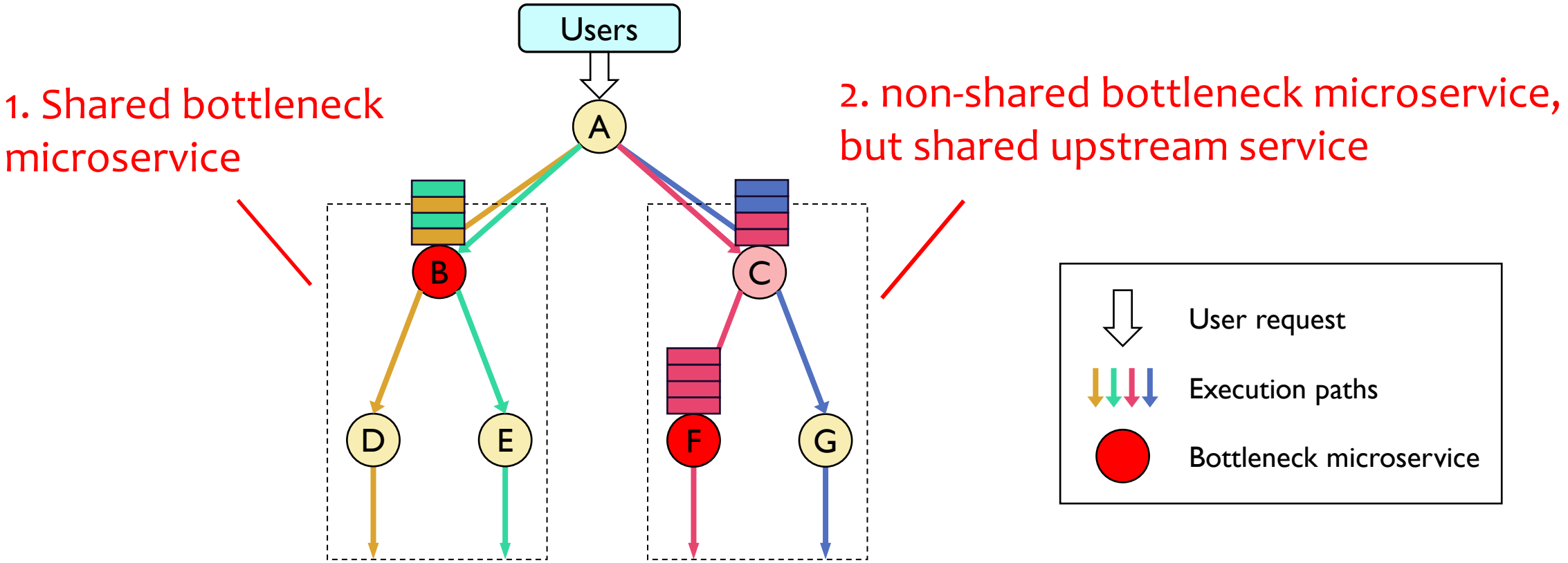
2. non-shared bottleneck microservice, but shared upstream service





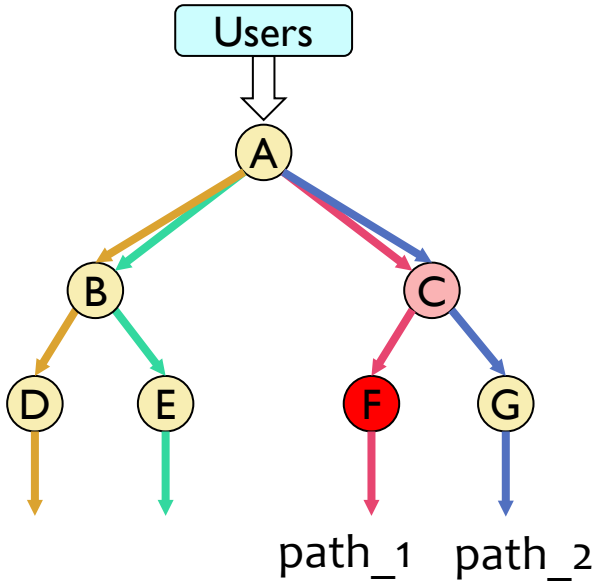
Cross-Path Dependencies Among Different Execution Paths

- Two typical cross-path dependencies



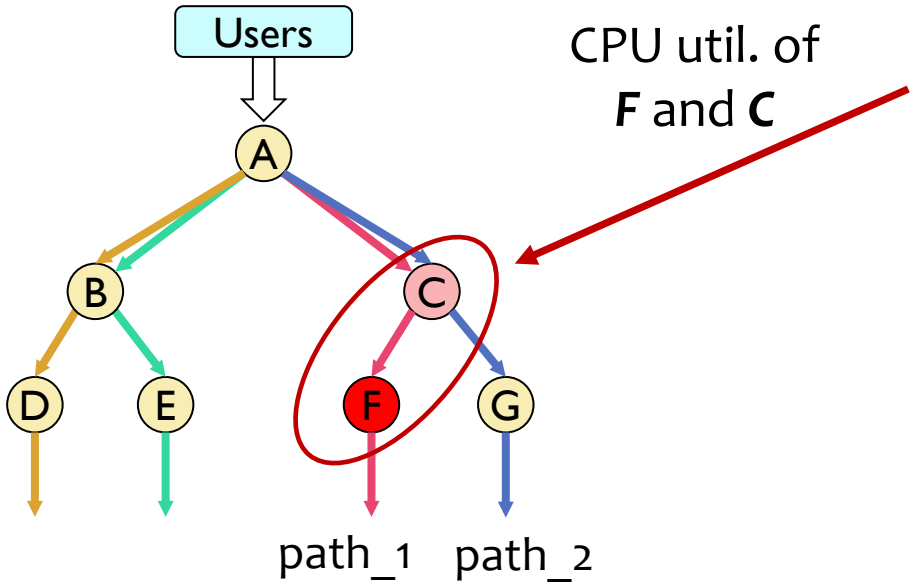
Cross-Path Dependencies → Long Response Time

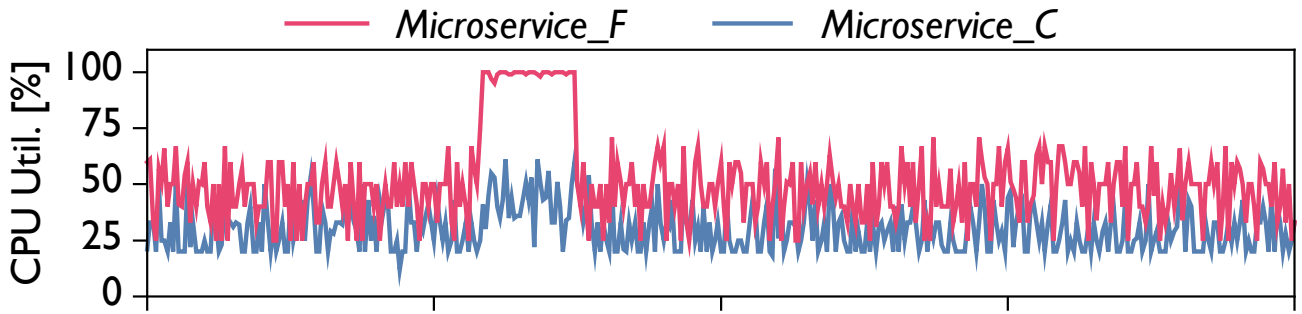
- Cross-path dependencies exist due to shared microservice
  - bottleneck on one path → performance interference to many other paths



Cross-Path Dependencies → Long Response Time

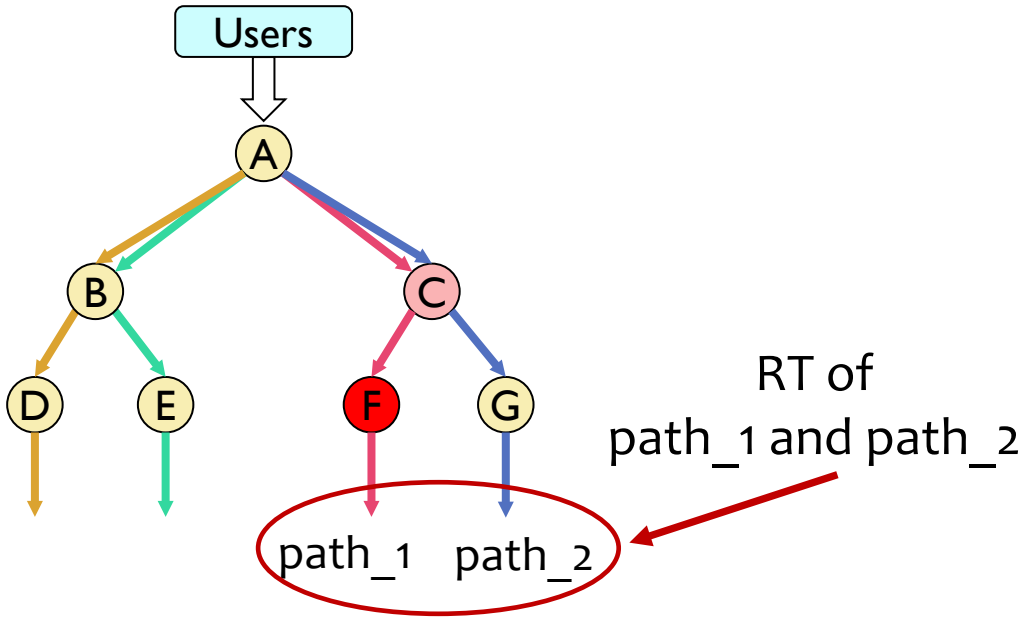
- Cross-path dependencies exist due to shared microservice
  - bottleneck on one path → performance interference to many other paths

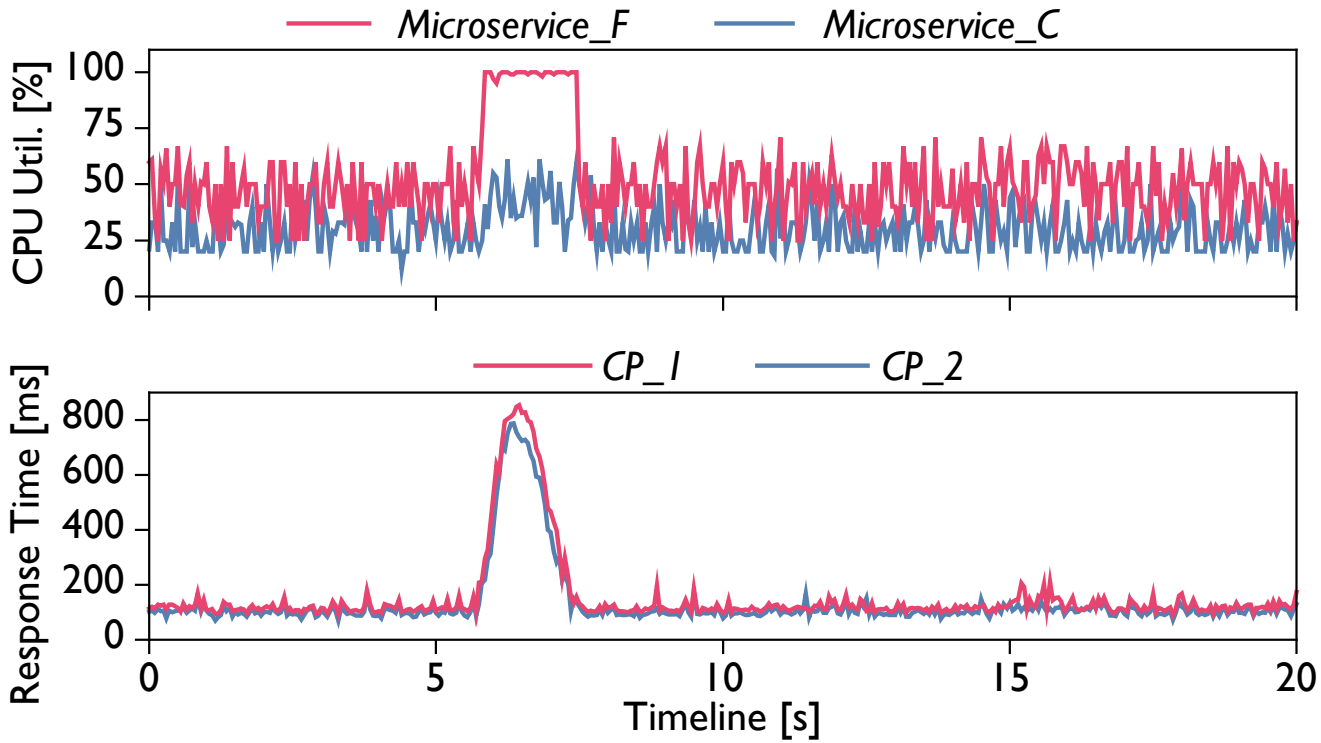




Cross-Path Dependencies → Long Response Time

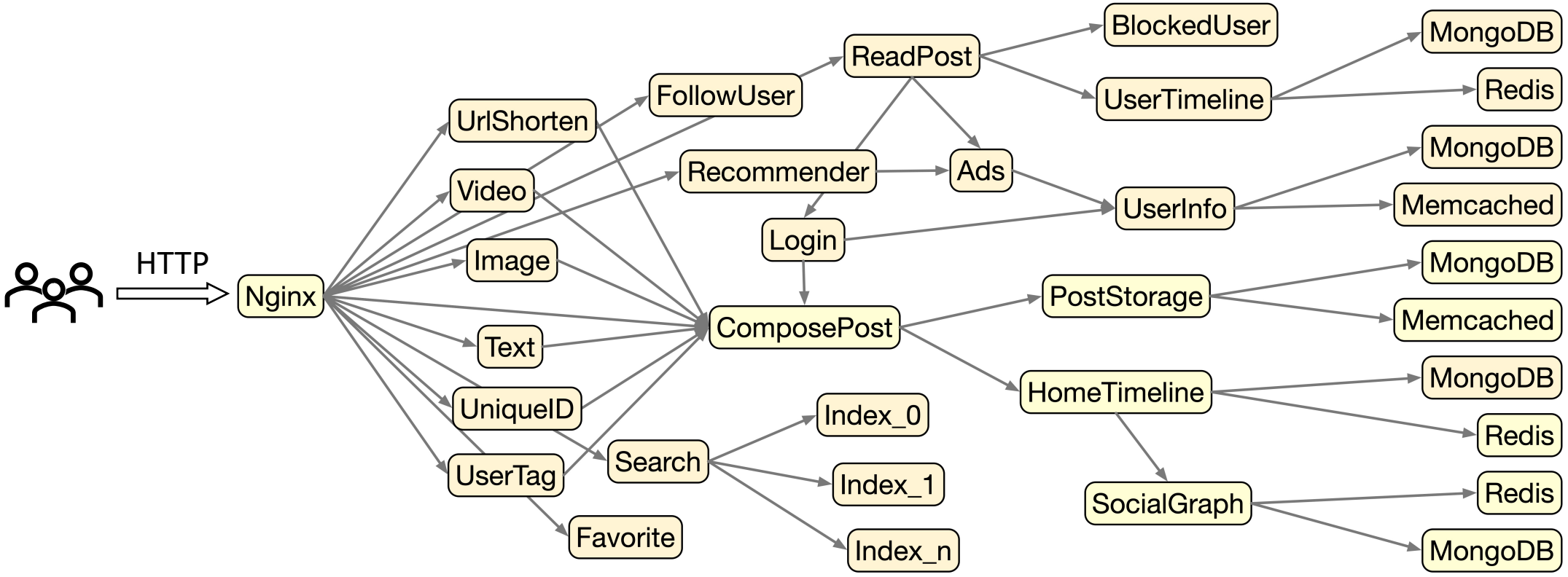
- Cross-path dependencies exist due to shared microservice
  - bottleneck on one path → performance interference to many other paths





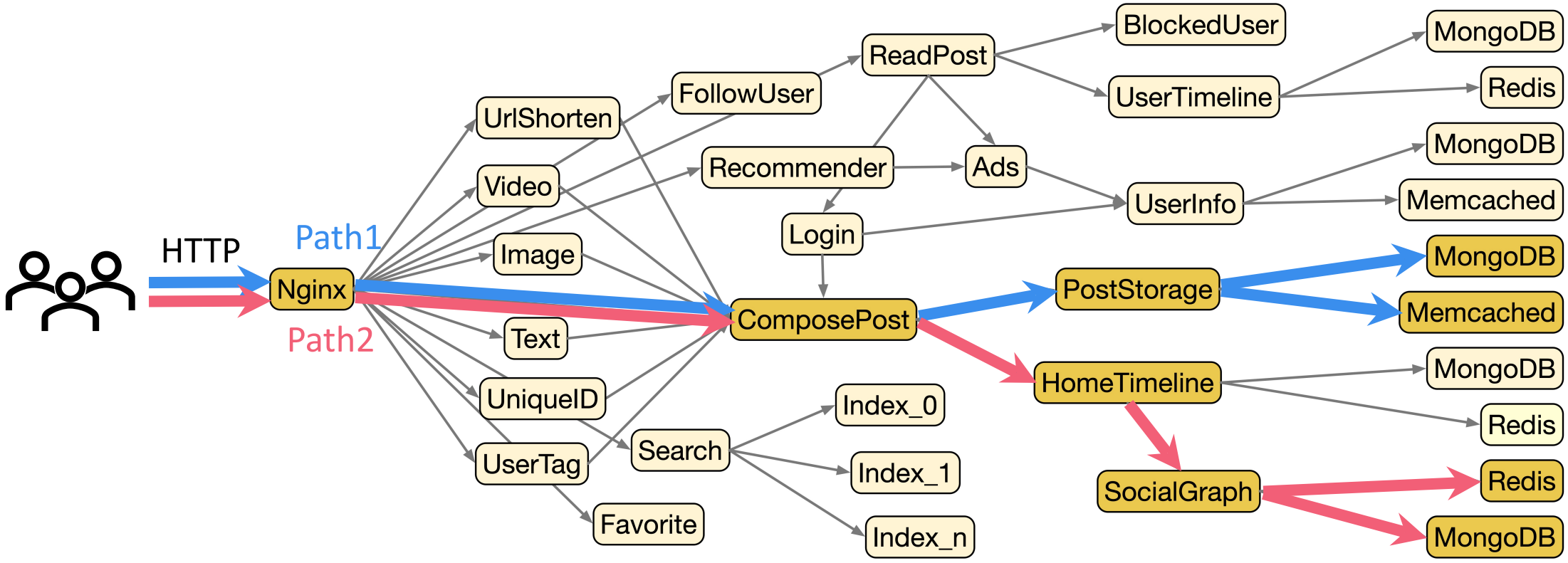
Case Study: Experimental Setup

- **SocialNetwork** microservices –[Gan et al. *ASPLOS' 19*]
- **36** unique component microservices
- More than **20** different execution paths supported



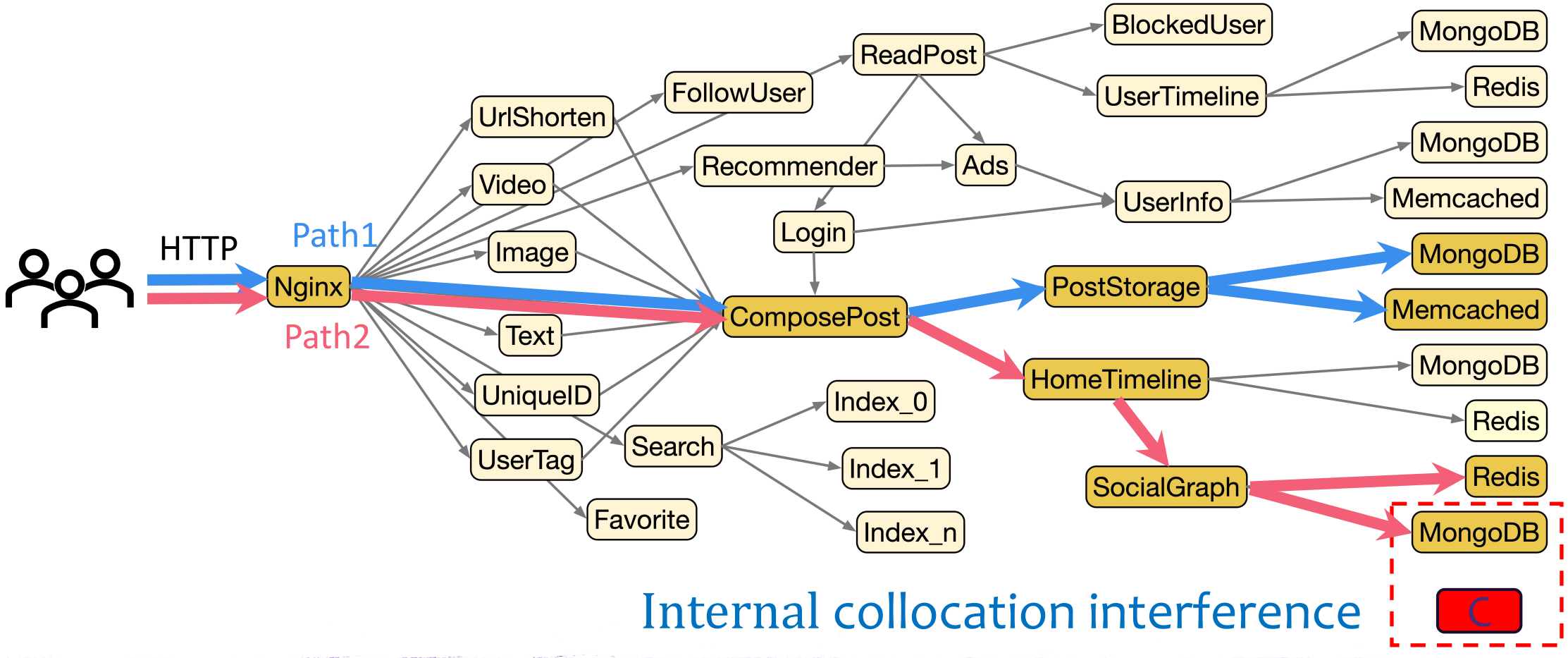
Case Study: Bottleneck Caused by Noisy Neighbor

- Two representative execution paths
- Collocated CPU-intensive neighbor



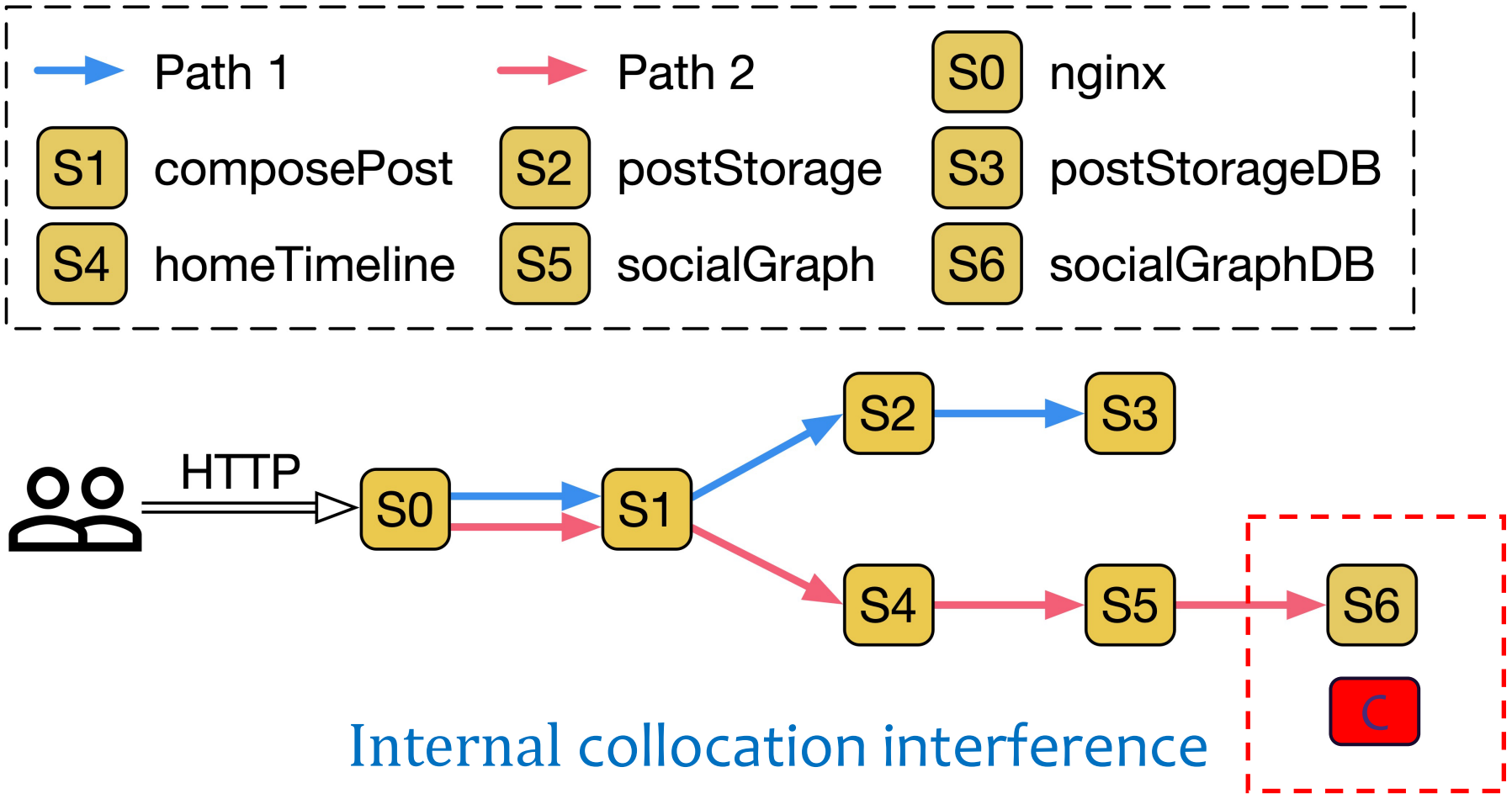
Case Study: Bottleneck Caused by Noisy Neighbor

- Two representative execution paths
- Collocated CPU-intensive neighbor

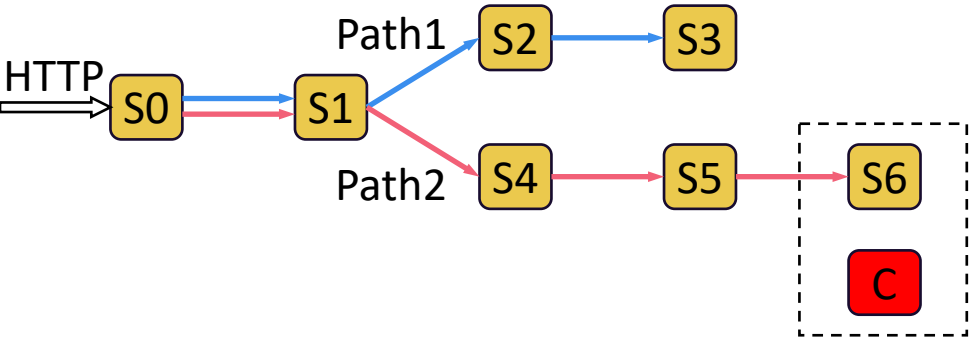


Case Study: Bottleneck Caused by Noisy Neighbor

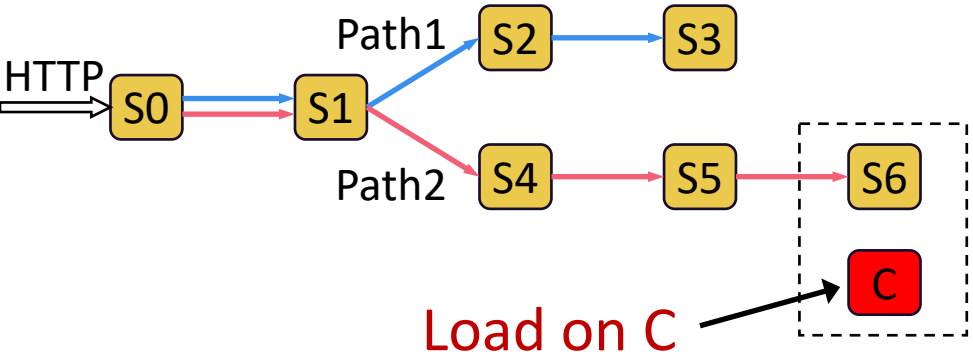
- Two representative execution paths
- Collocated CPU-intensive neighbor



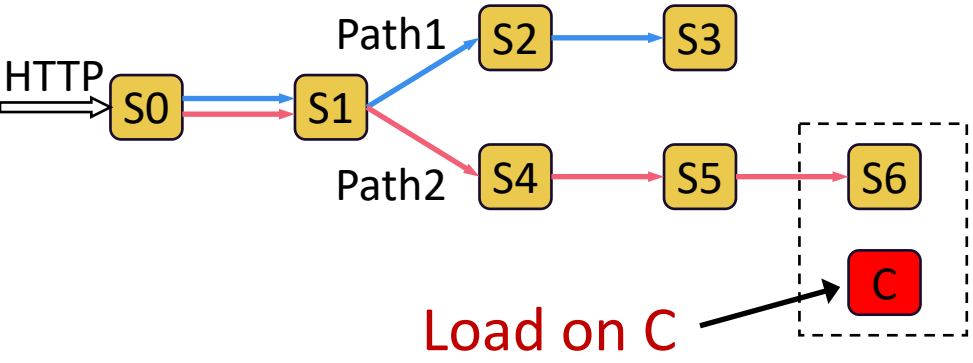
Noisy Neighbor Interference → CPU Saturation on S6

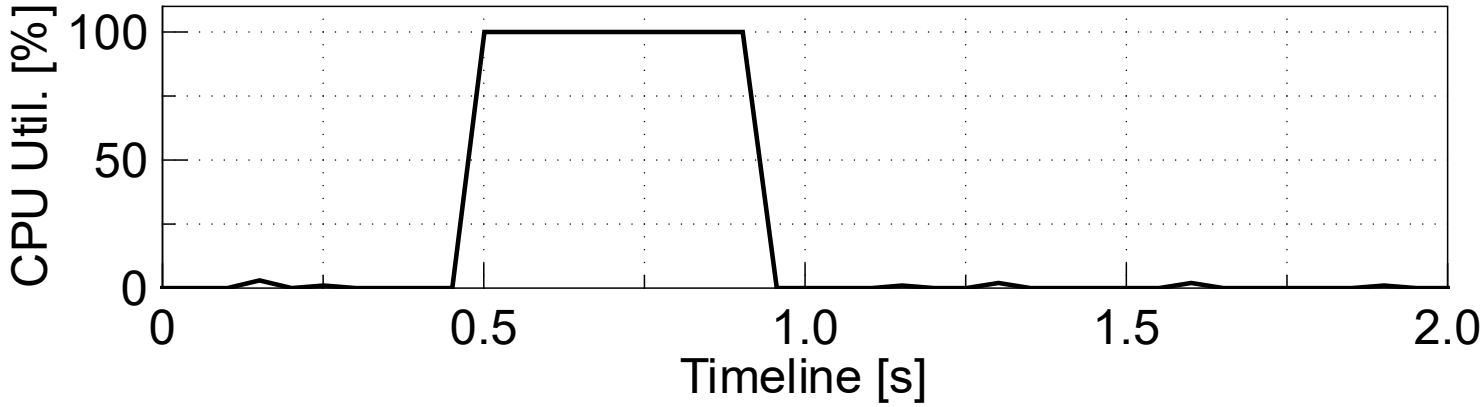


Noisy Neighbor Interference → CPU Saturation on S6

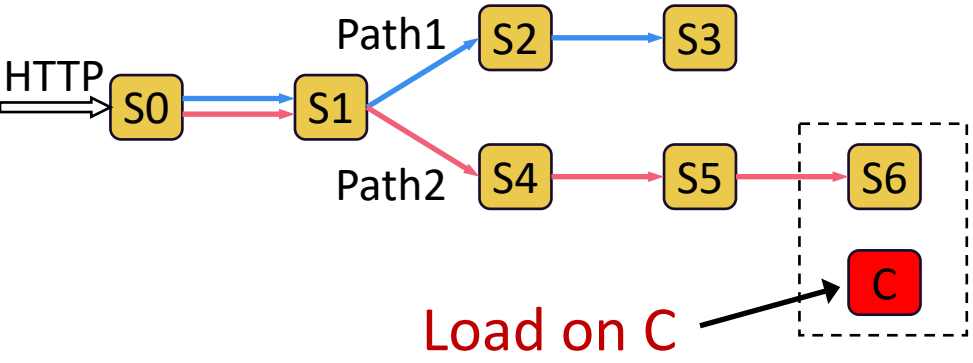


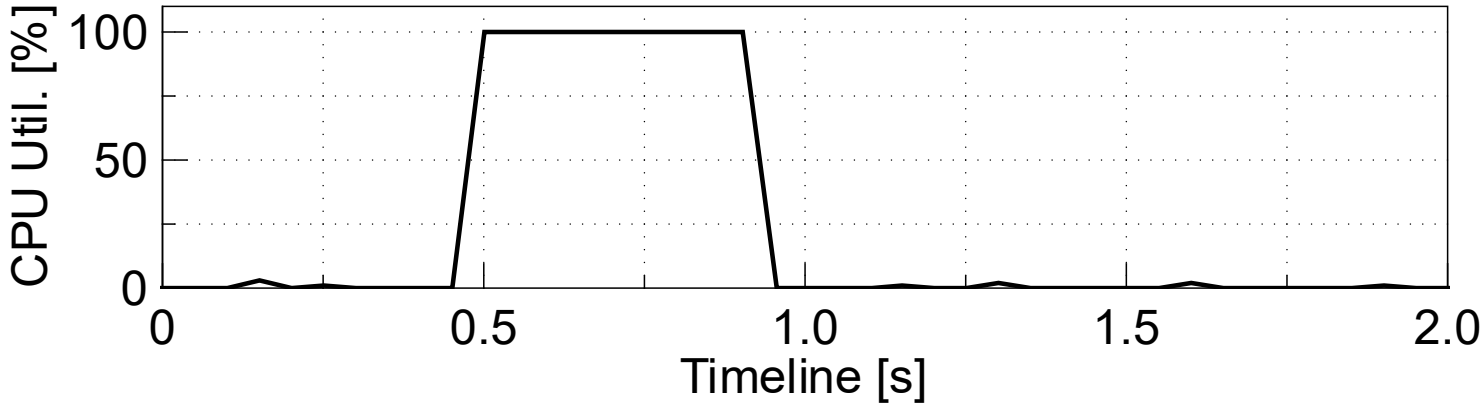
Noisy Neighbor Interference → CPU Saturation on S6

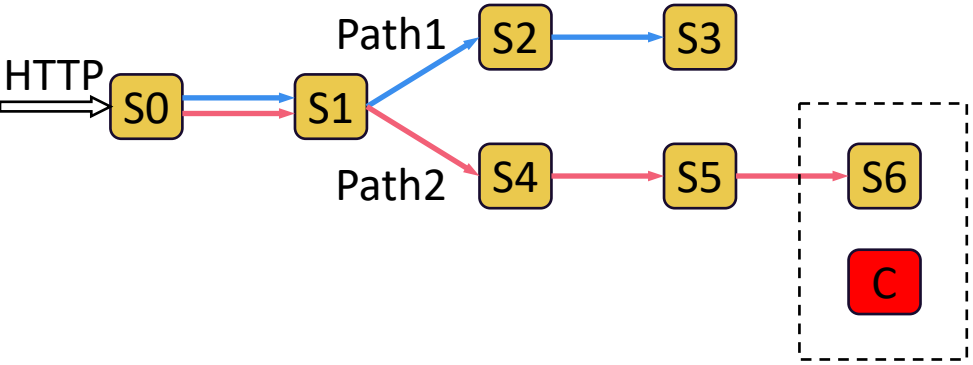




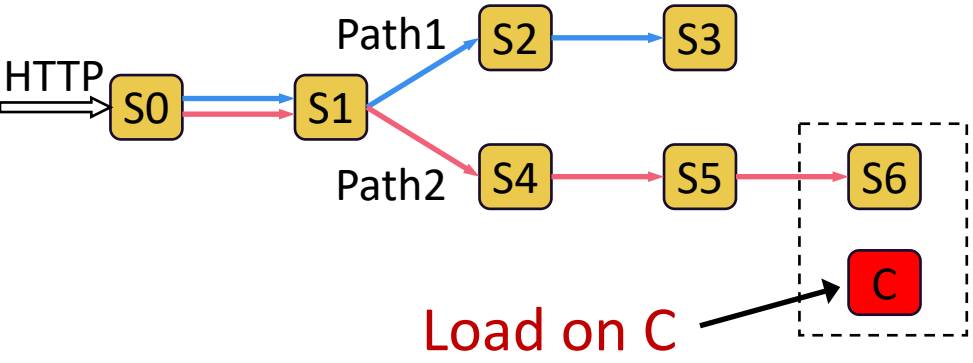
Noisy Neighbor Interference → CPU Saturation on S6

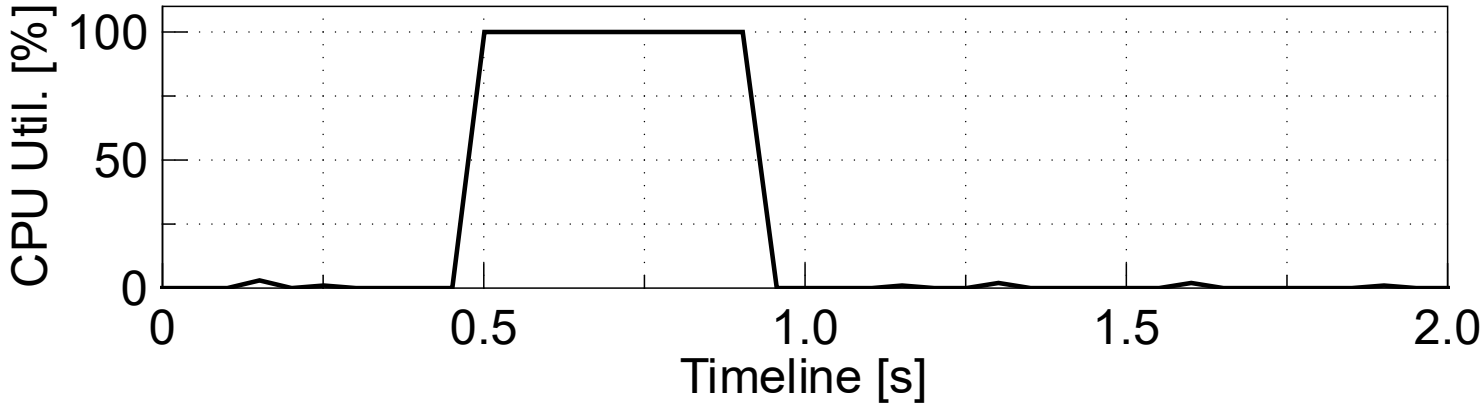


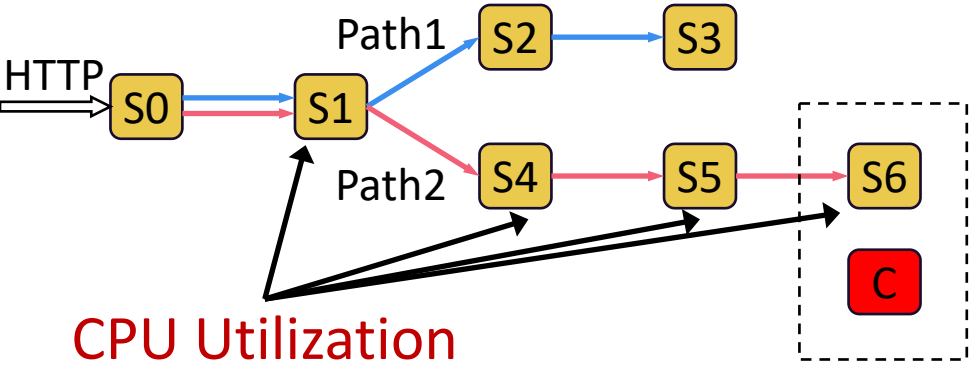




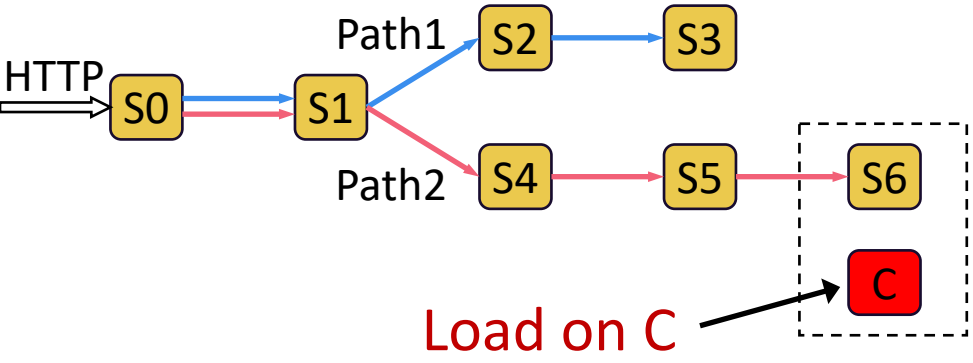
Noisy Neighbor Interference → CPU Saturation on S6

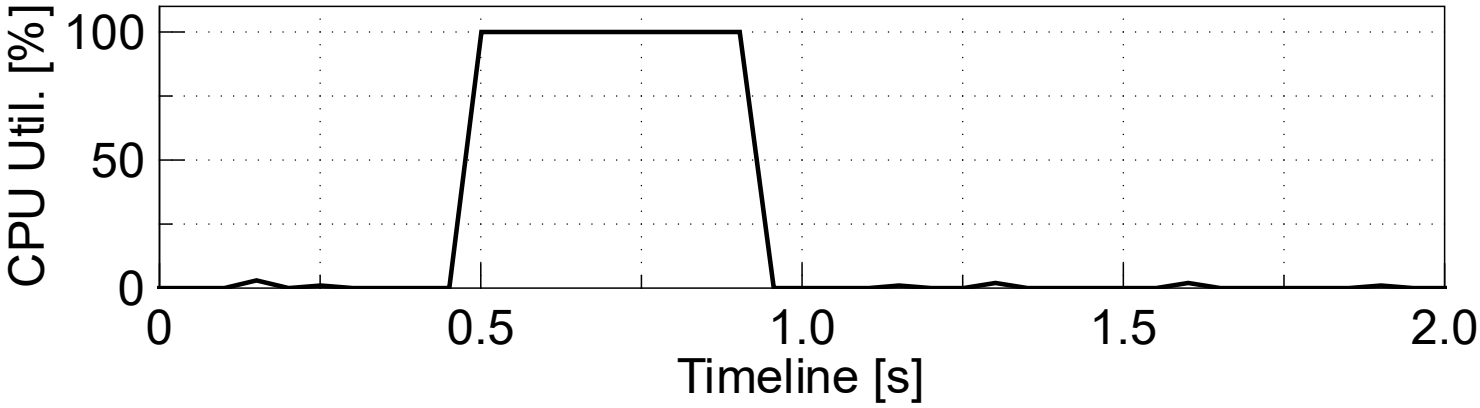


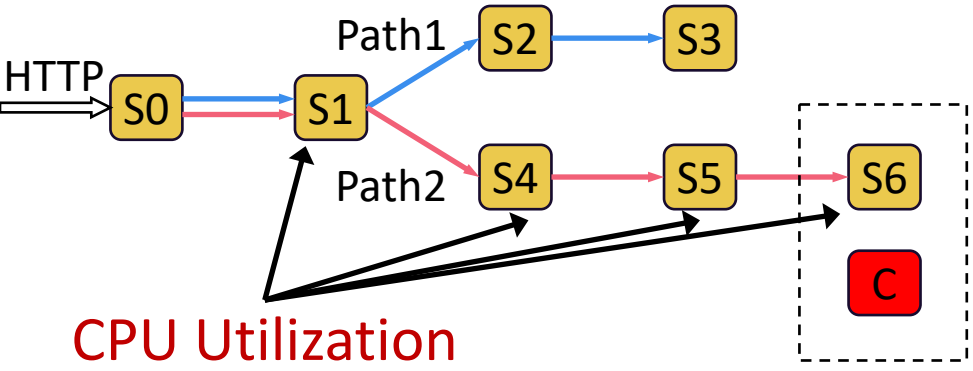


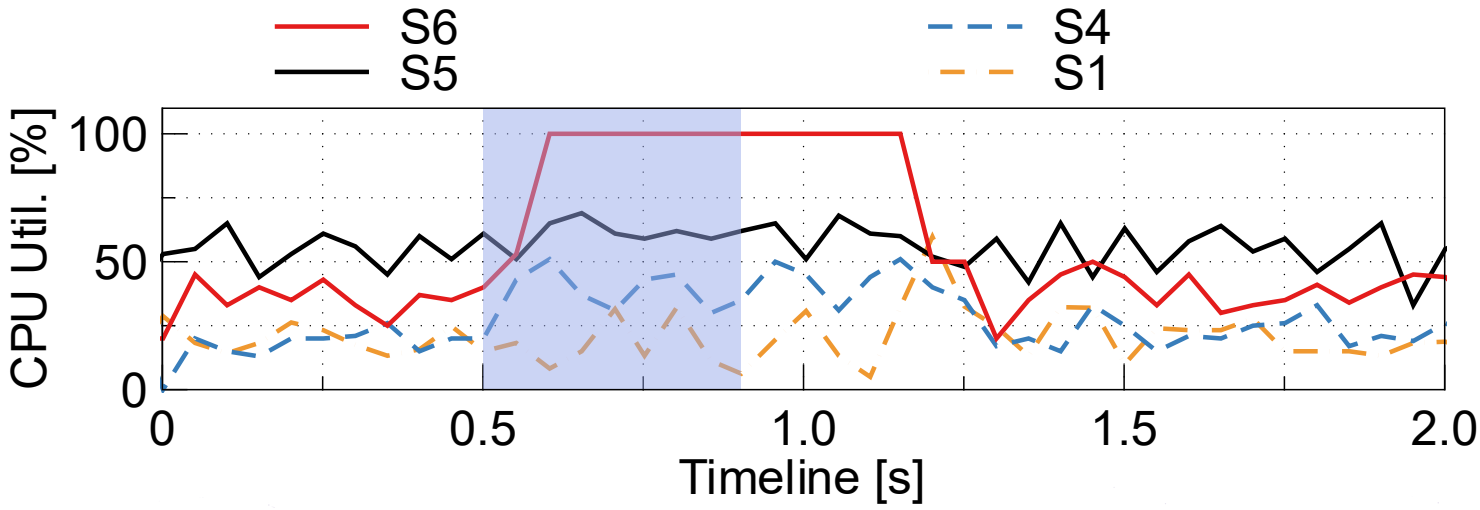


Noisy Neighbor Interference → CPU Saturation on S6

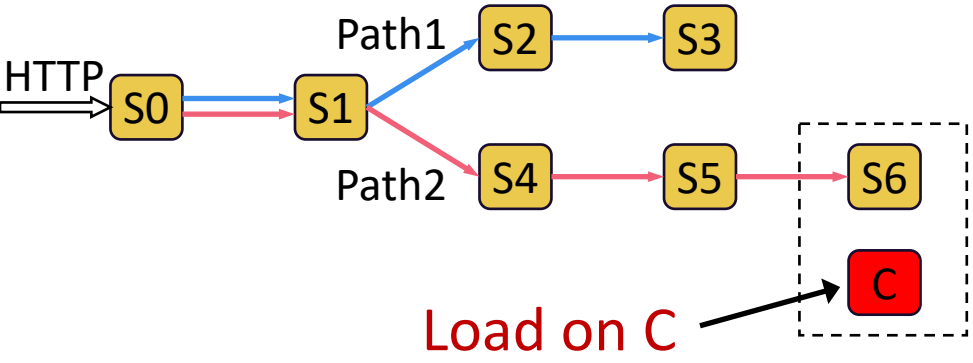


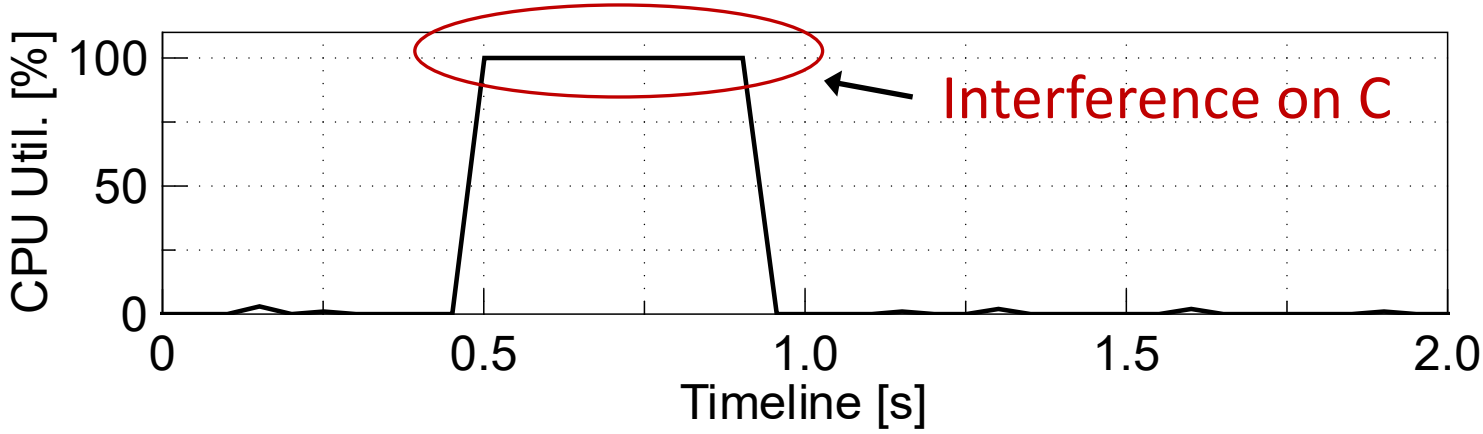


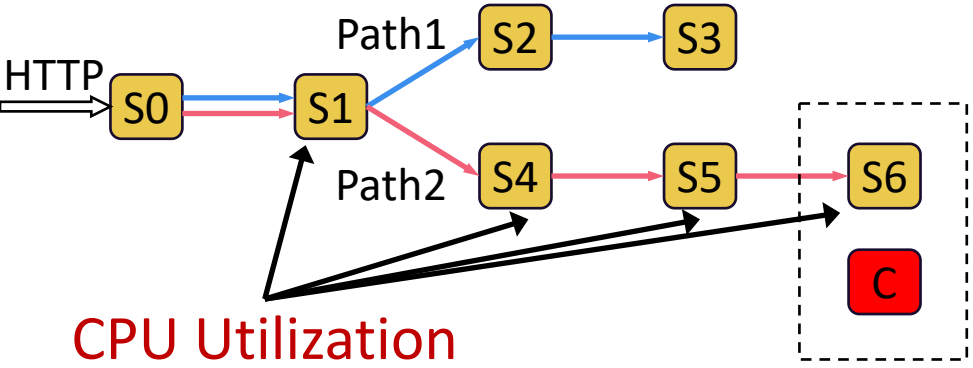


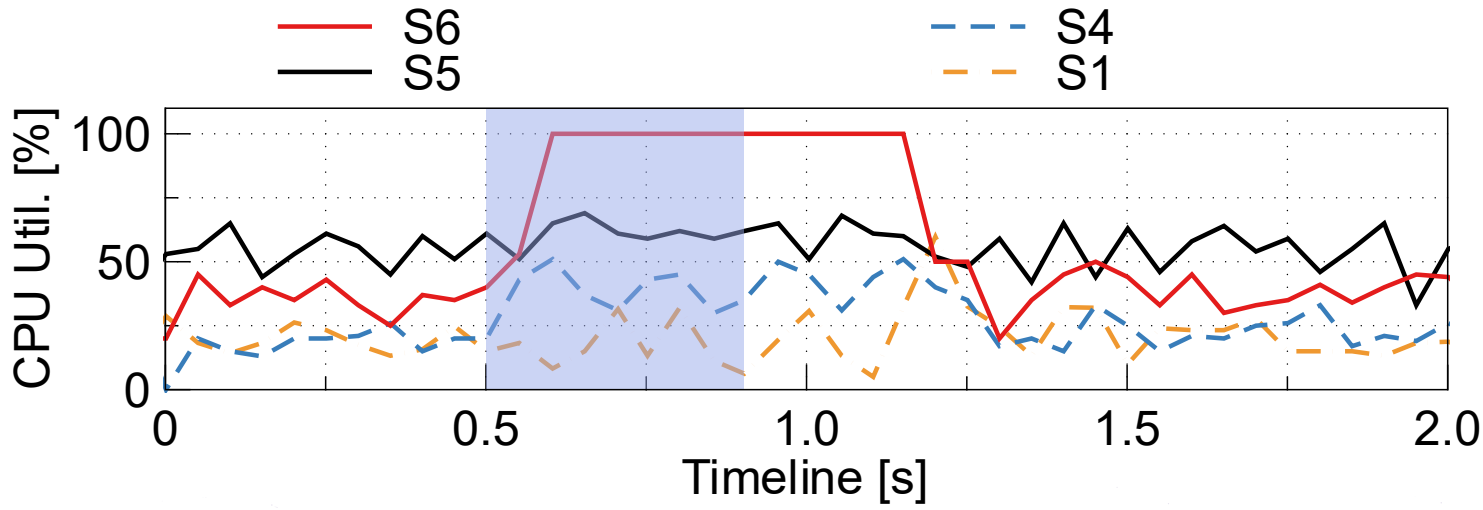


Noisy Neighbor Interference → CPU Saturation on S6

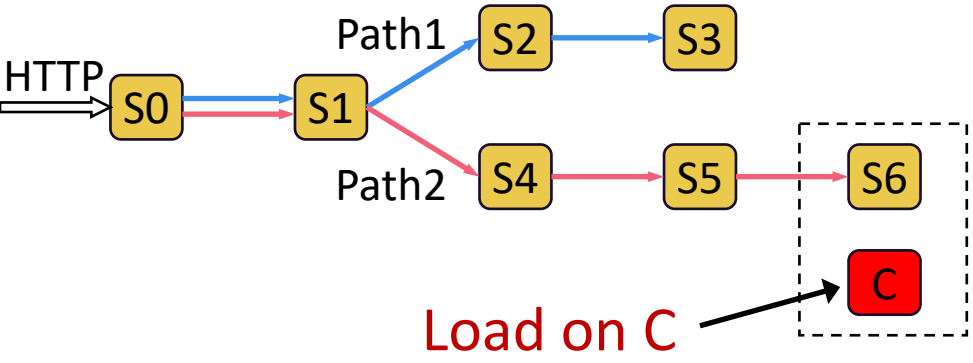


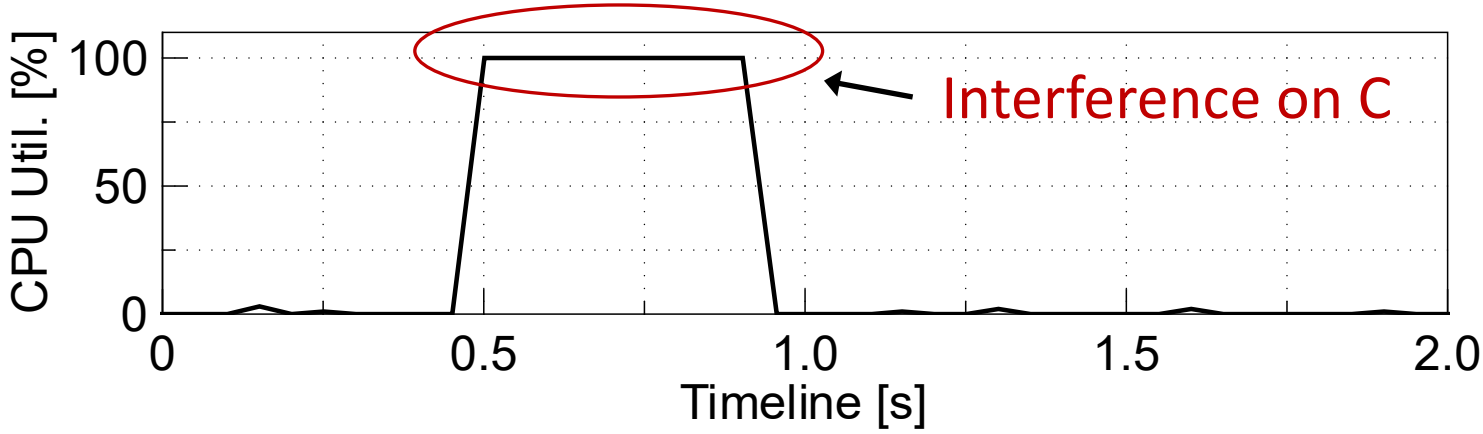


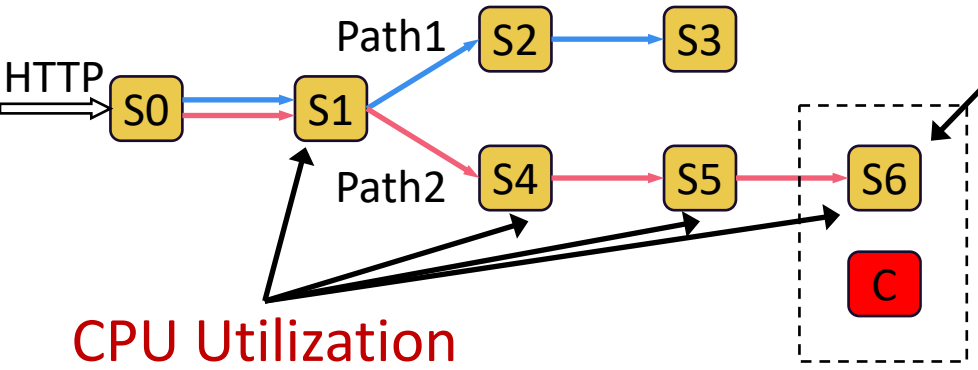


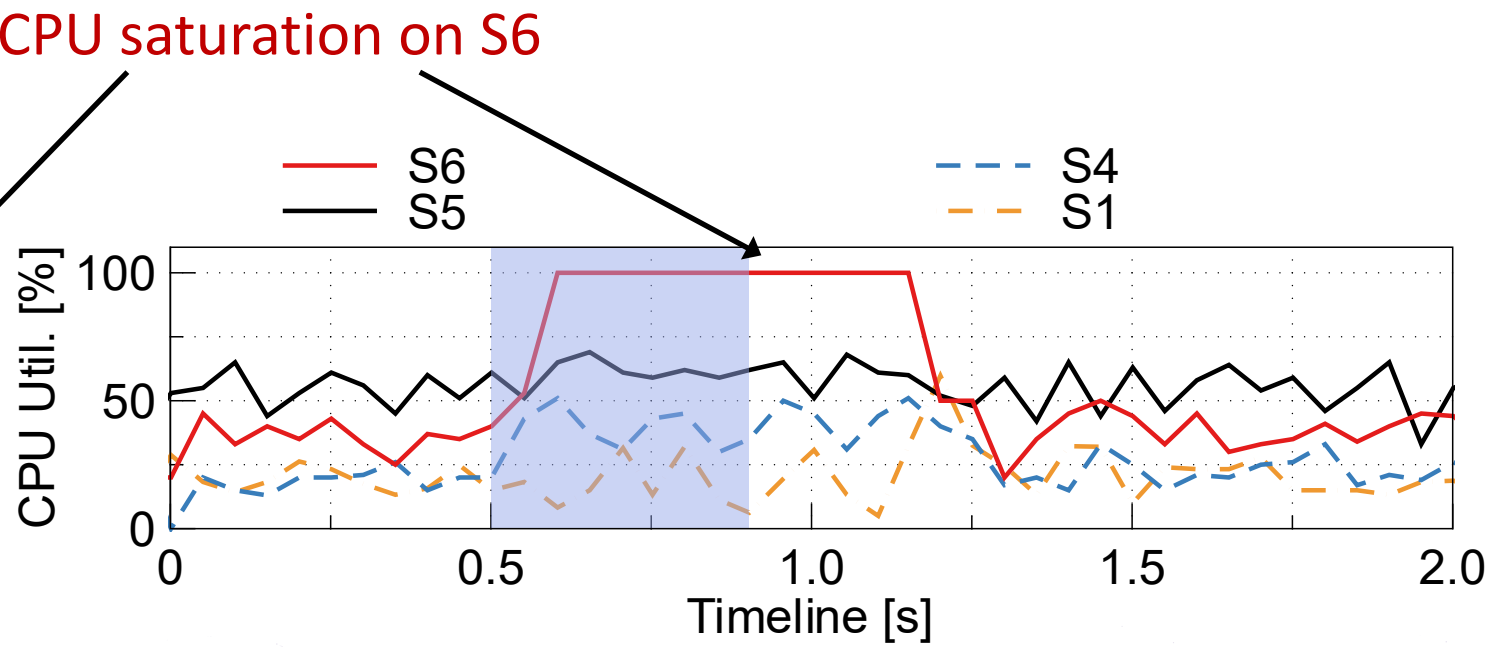


Noisy Neighbor Interference → CPU Saturation on S6

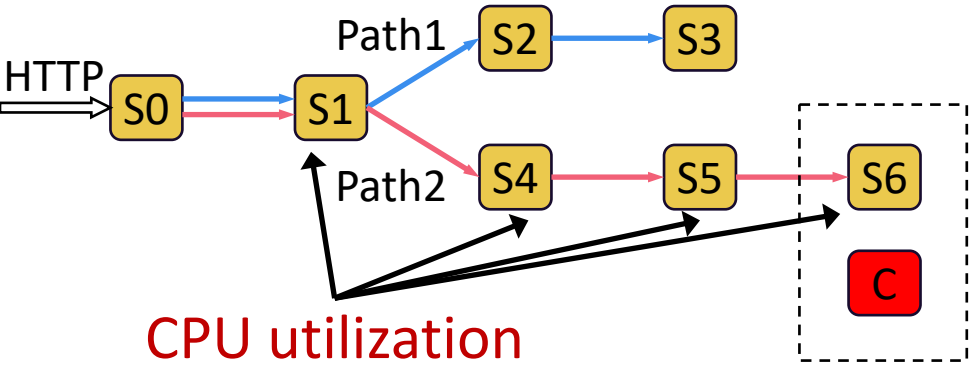


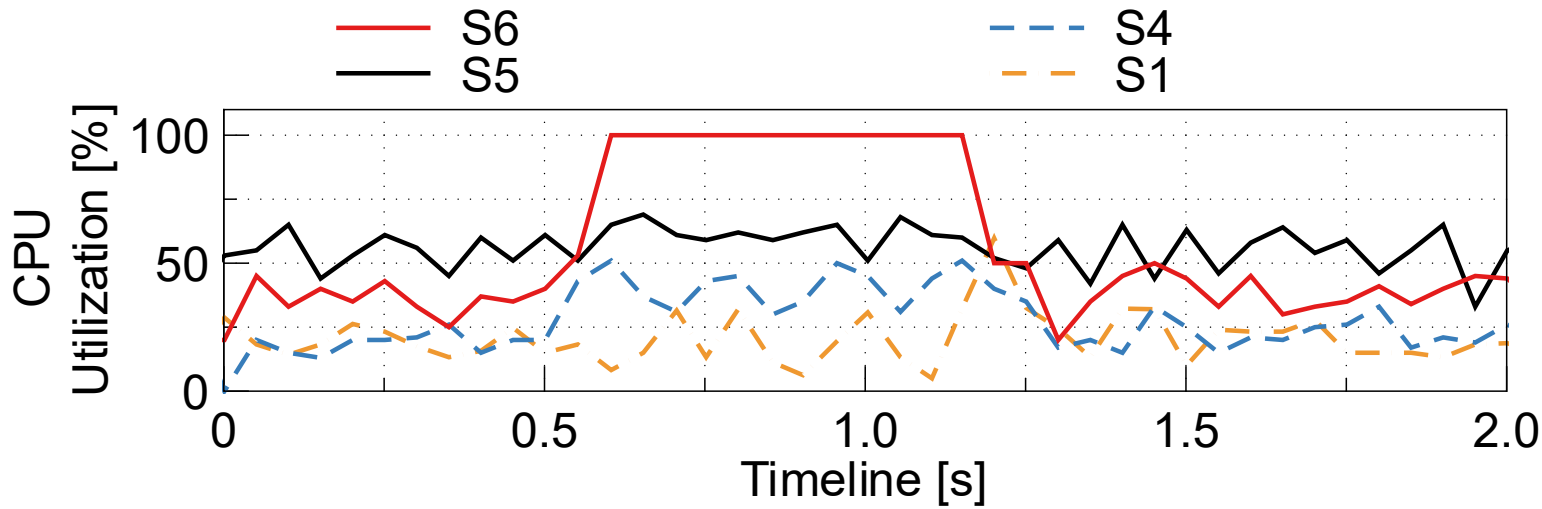




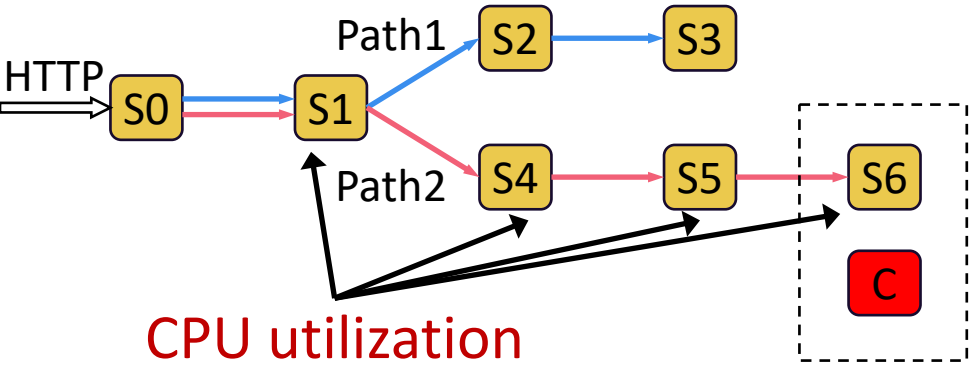


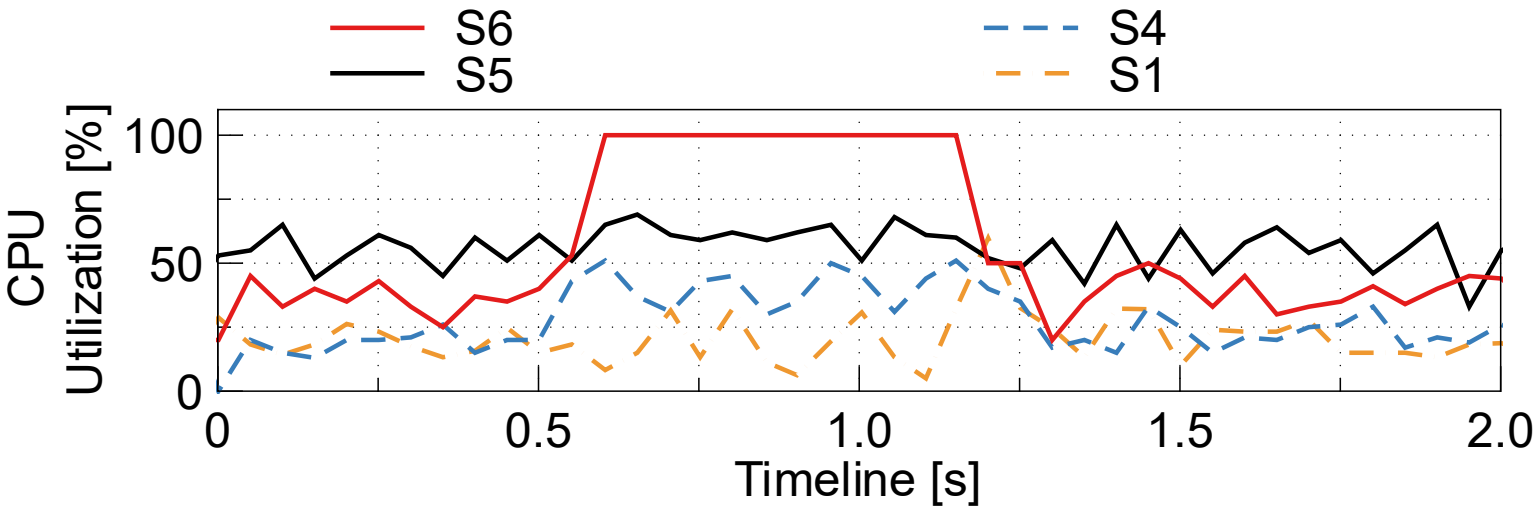
CPU Saturation on S6 → Cross-Service Queue Overflow

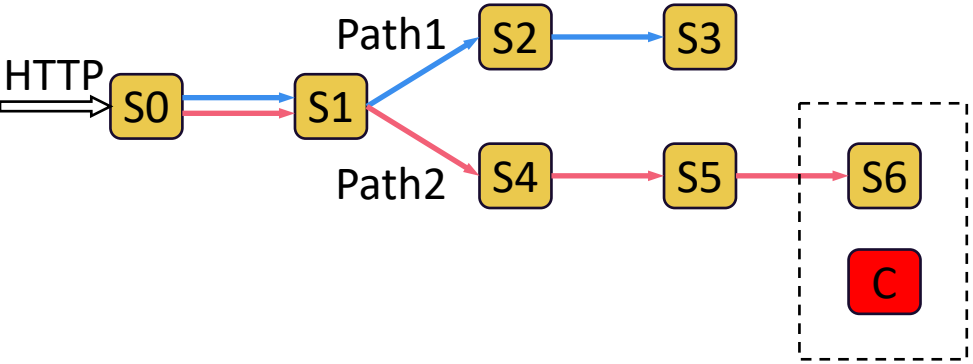




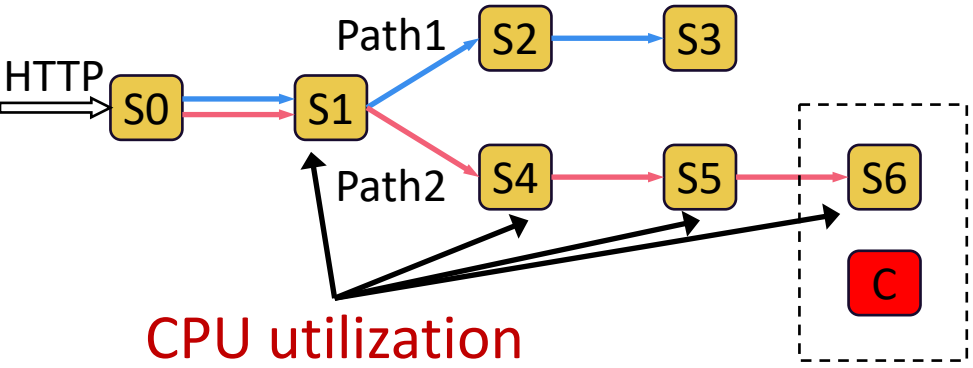
CPU Saturation on S6 → Cross-Service Queue Overflow



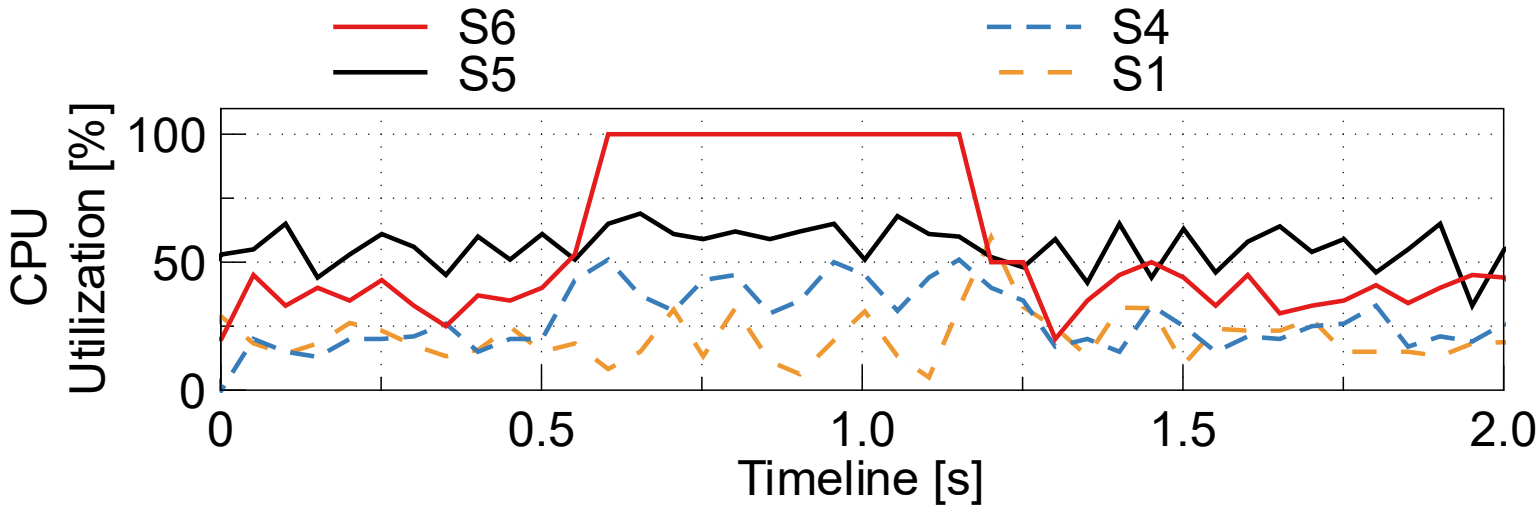


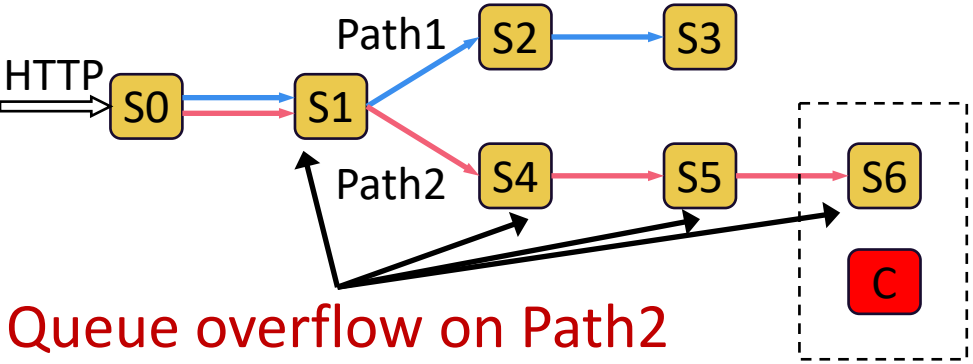


CPU Saturation on S6 → Cross-Service Queue Overflow



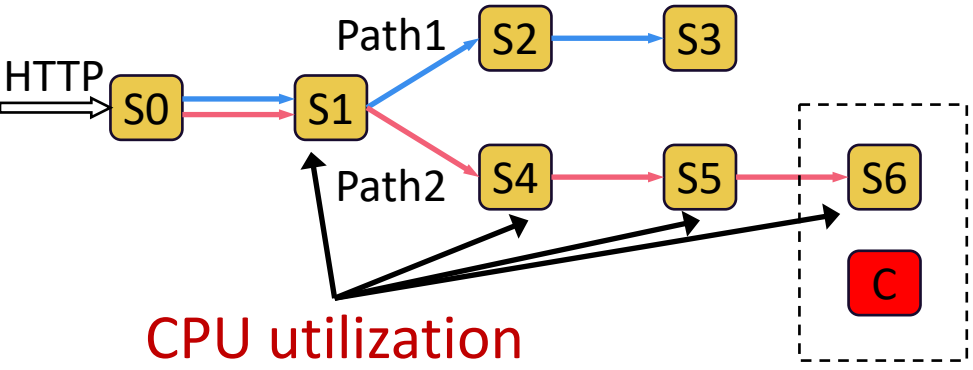
CPU utilization



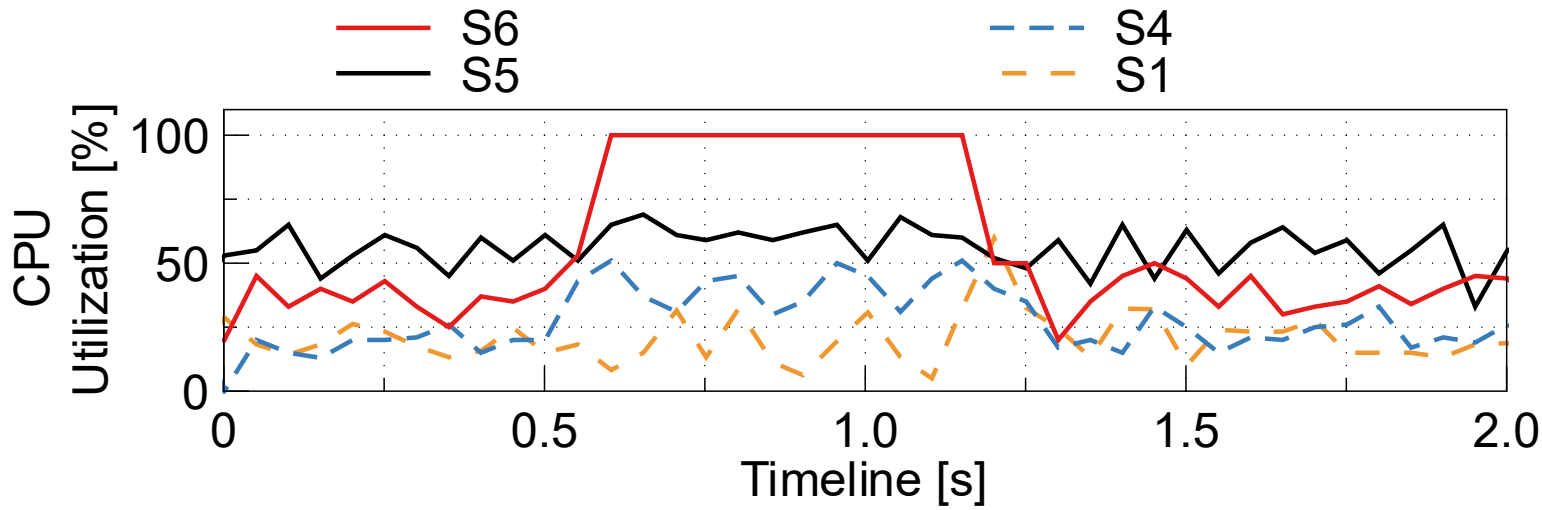


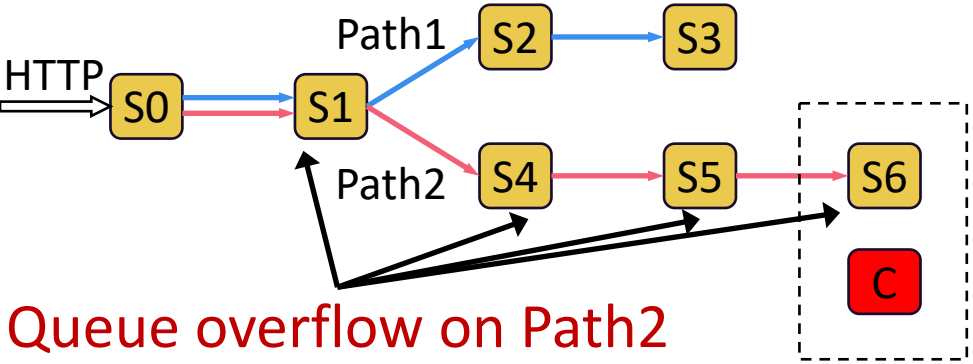
Queue overflow on Path2

CPU Saturation on S6 → Cross-Service Queue Overflow

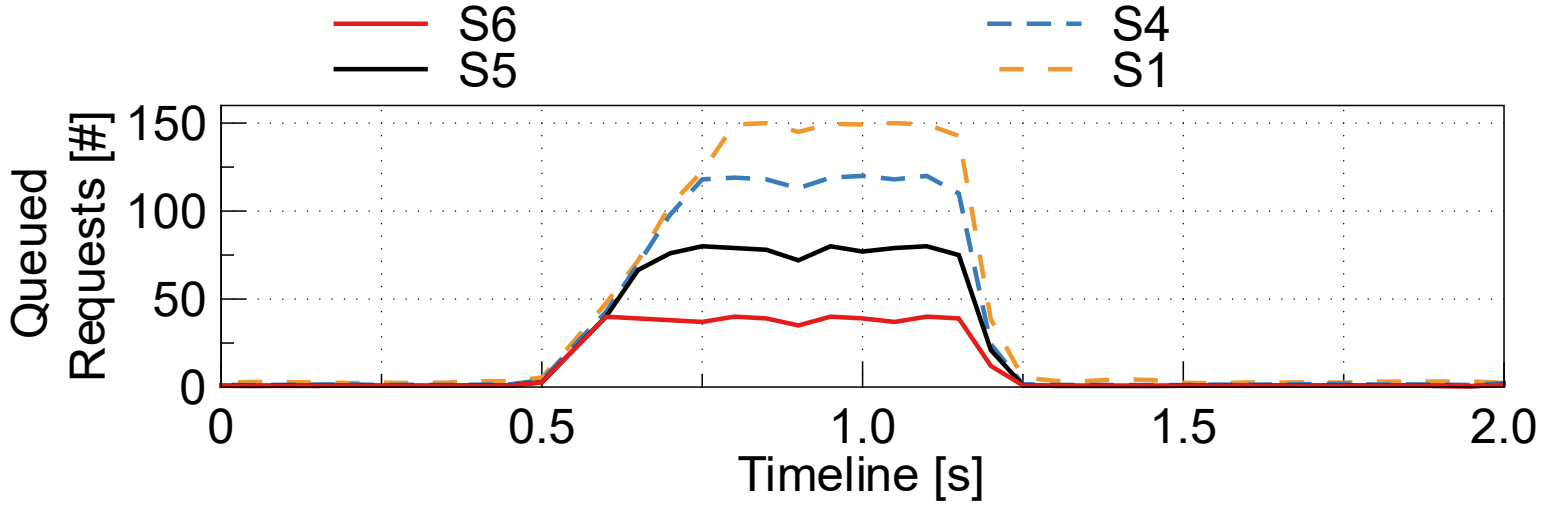


CPU utilization

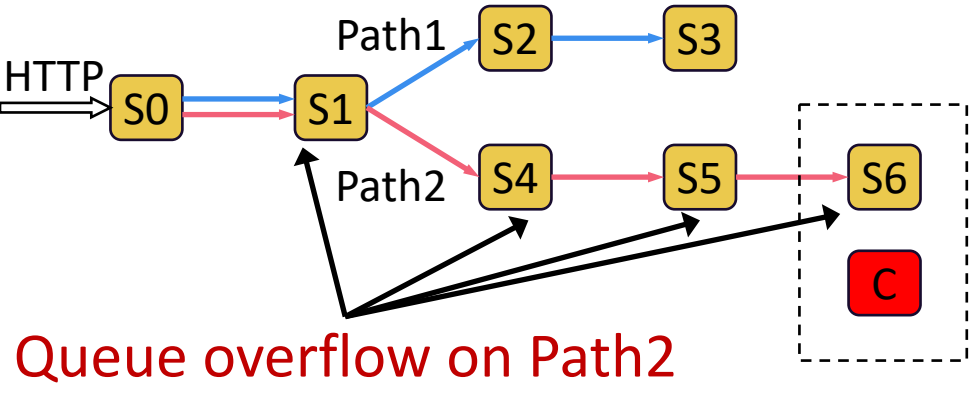




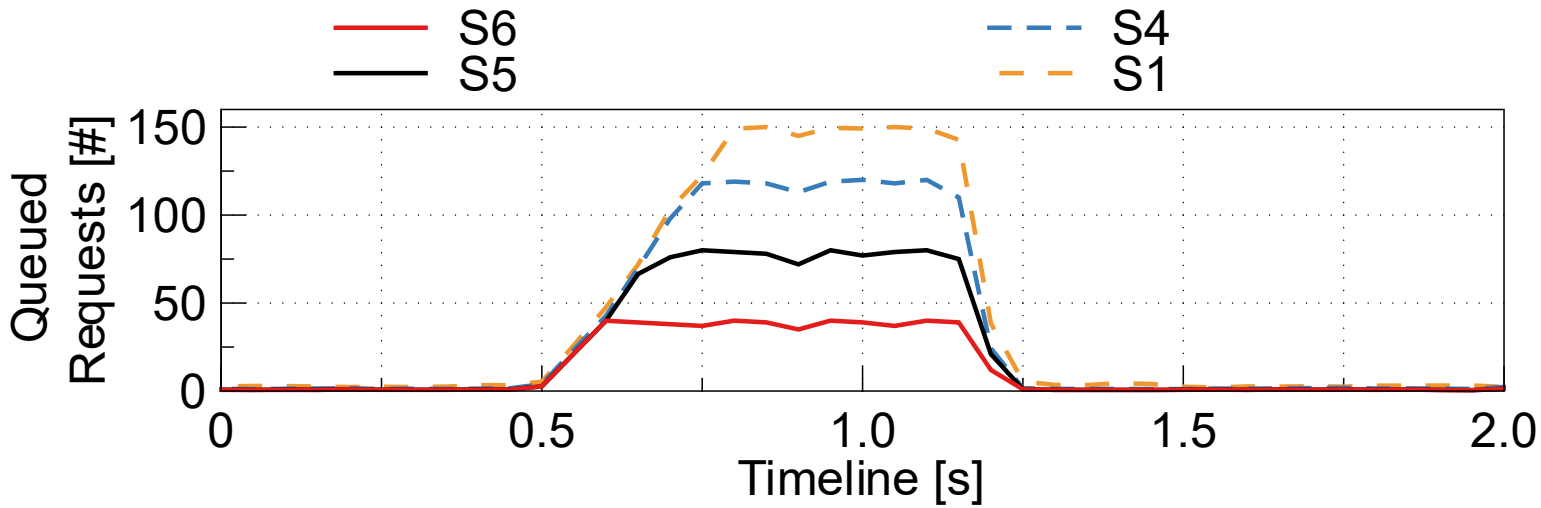
Queue overflow on Path2



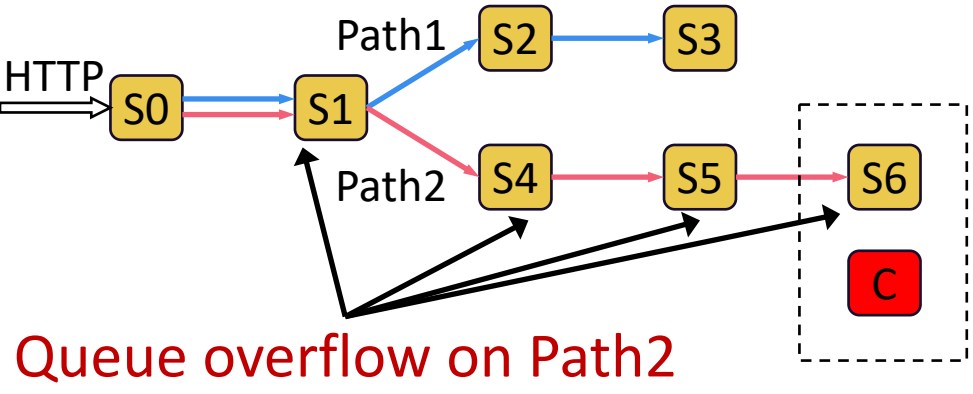
Cross-Service Queue Overflow → Worker Thread Contention



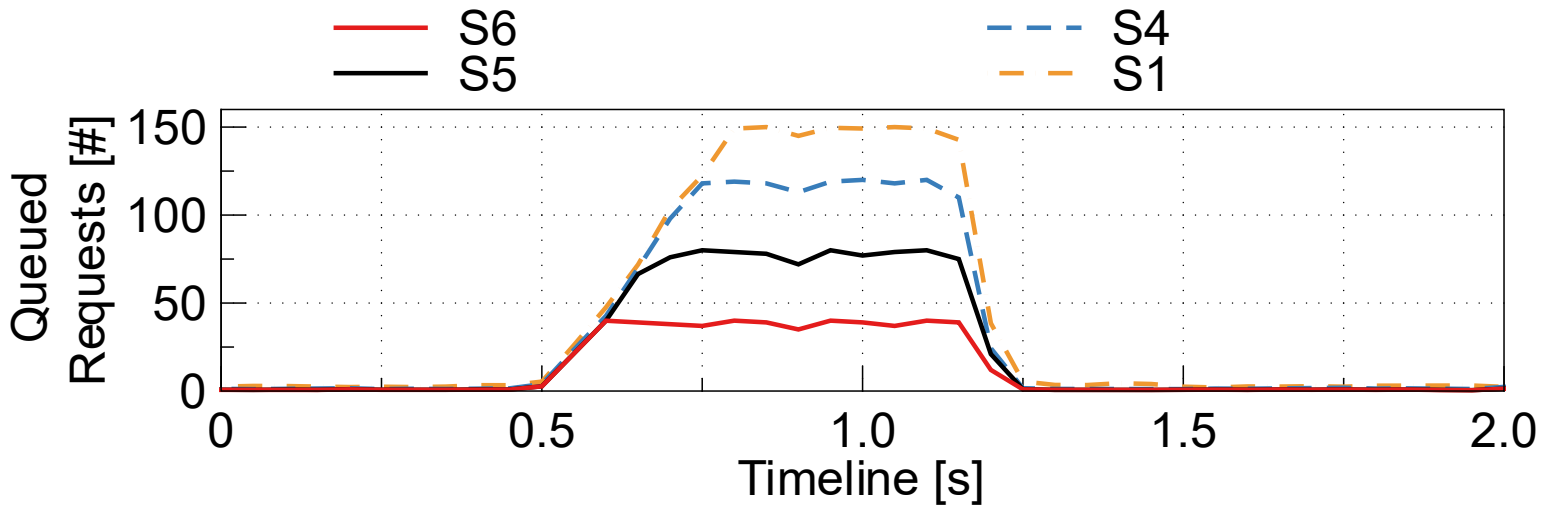
Queue overflow on Path2

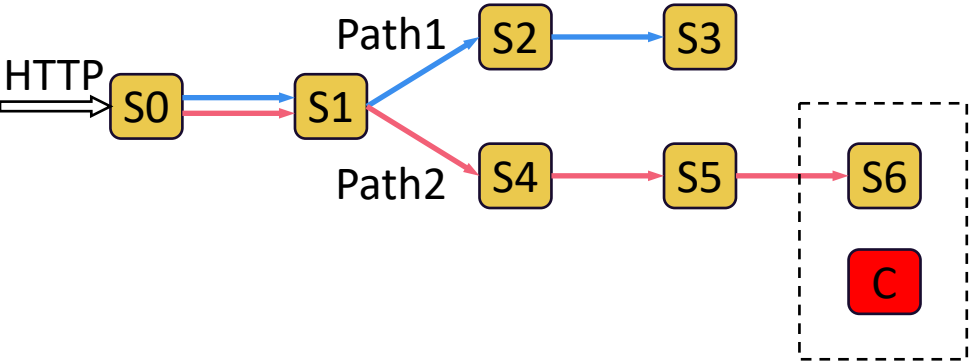


Cross-Service Queue Overflow → Worker Thread Contention

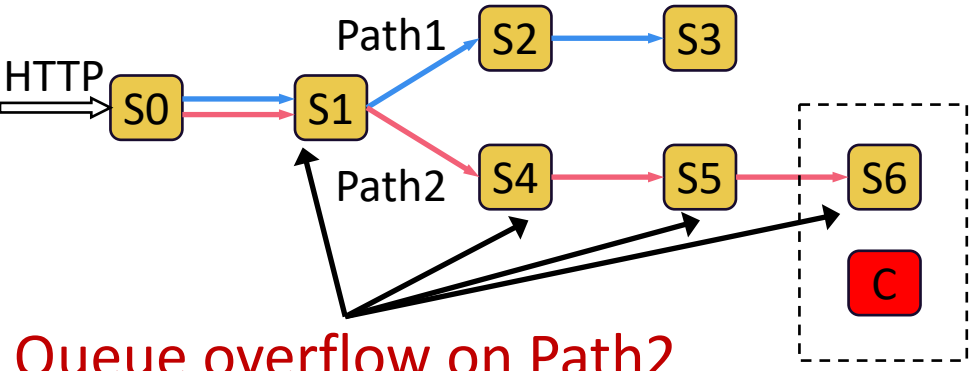


Queue overflow on Path2

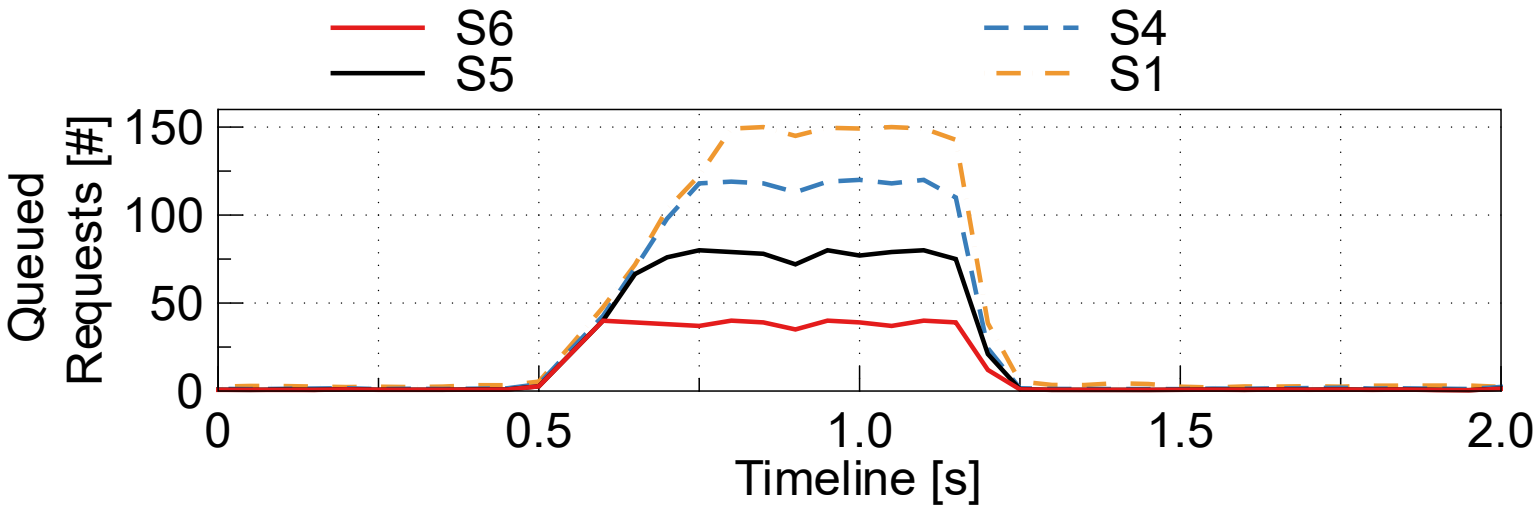


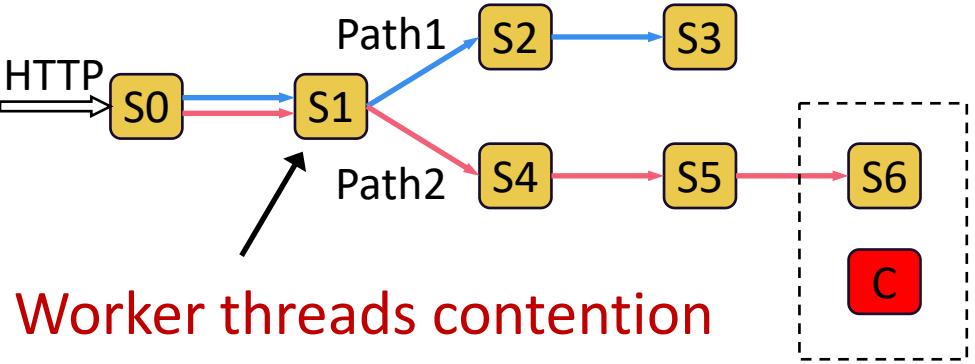


Cross-Service Queue Overflow → Worker Thread Contention



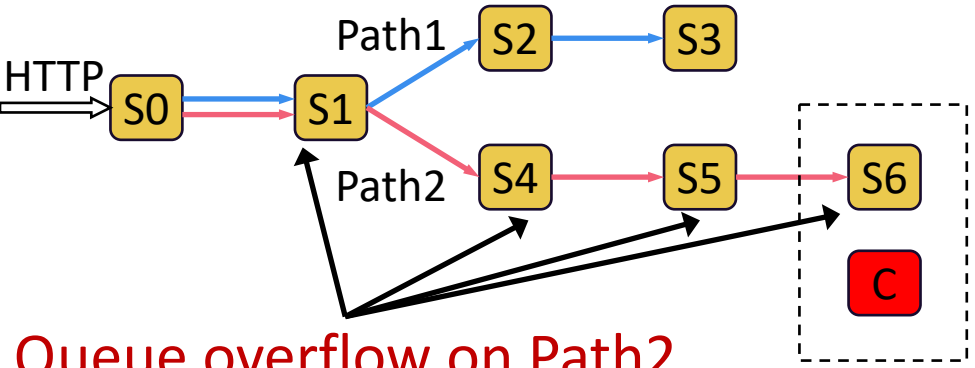
Queue overflow on Path2



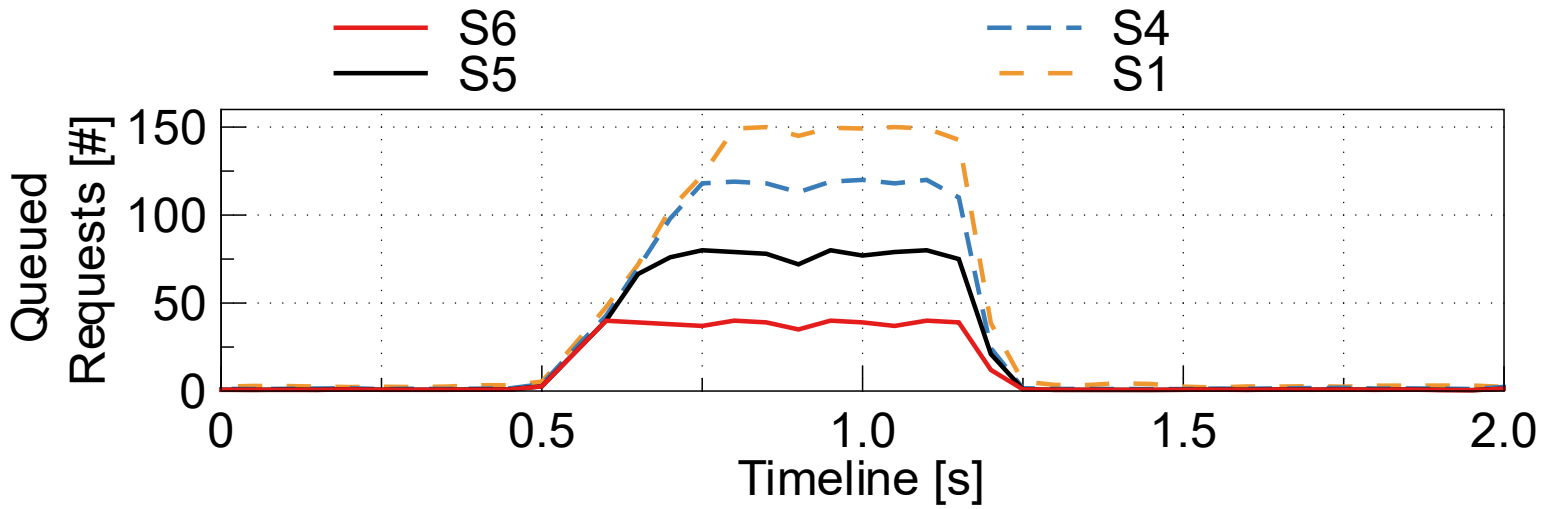


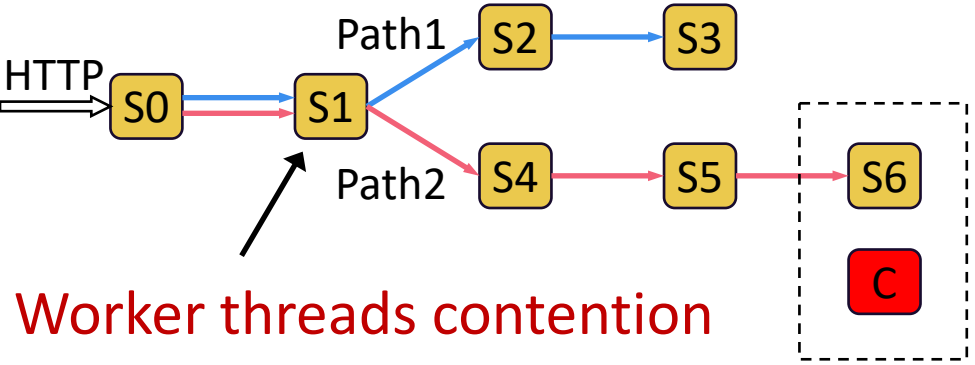
Worker threads contention on S1

Cross-Service Queue Overflow → Worker Thread Contention

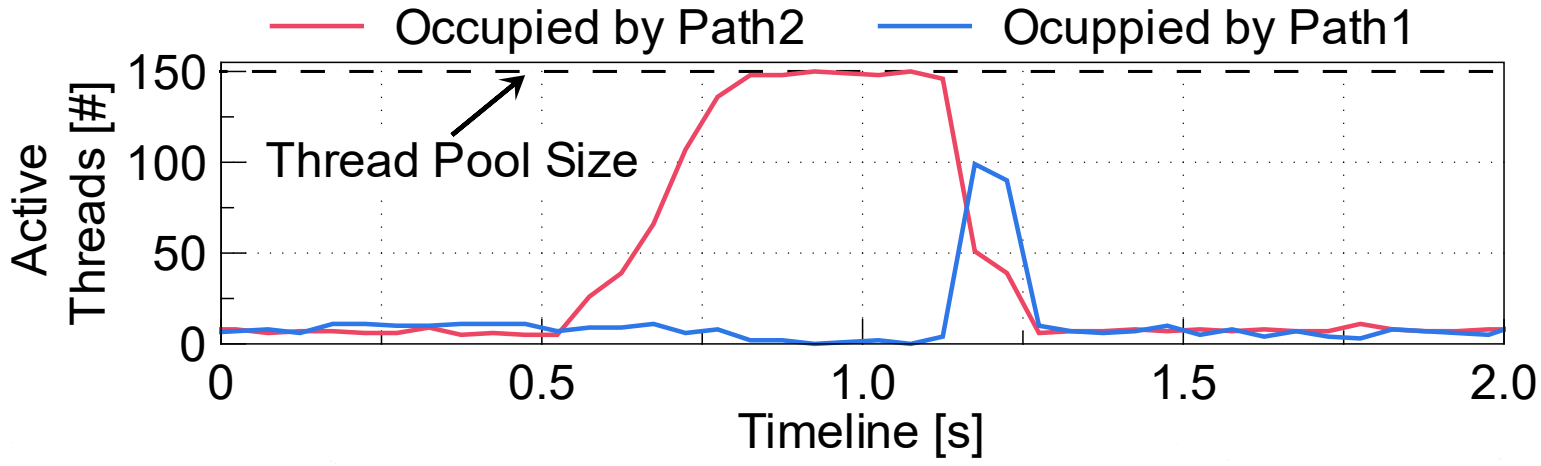


Queue overflow on Path2

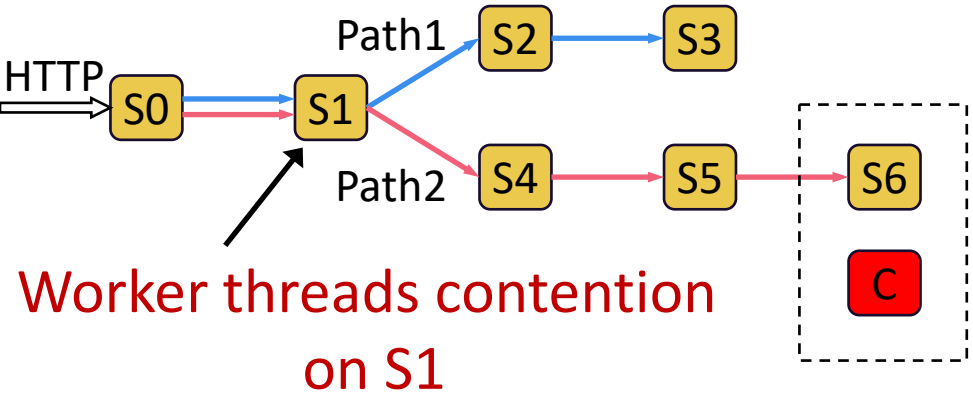


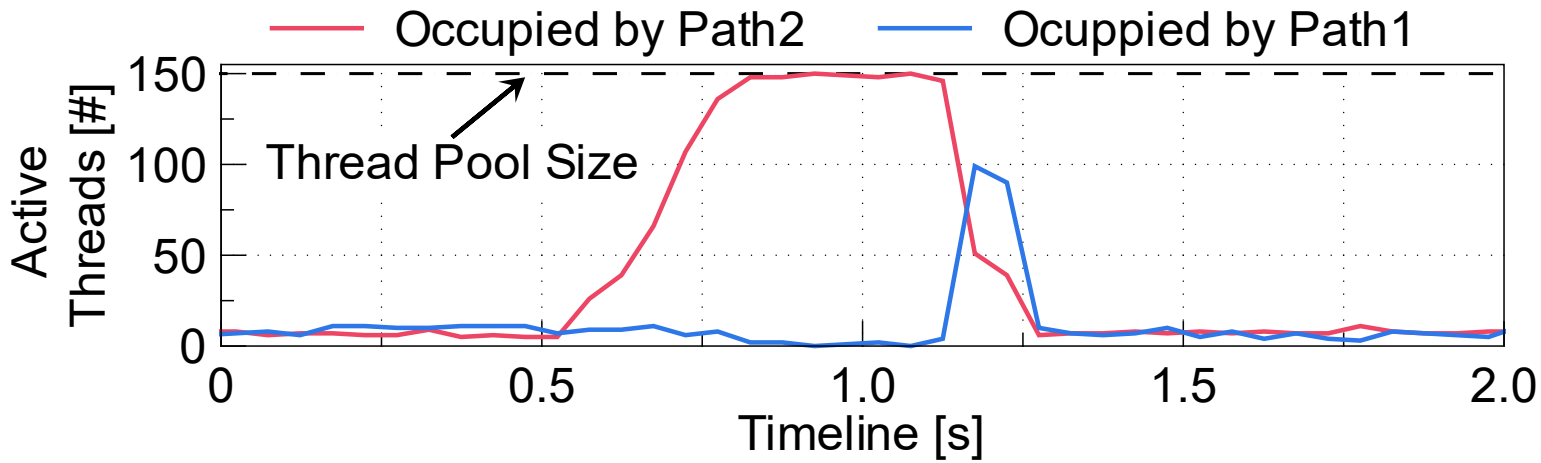


Worker threads contention on S1

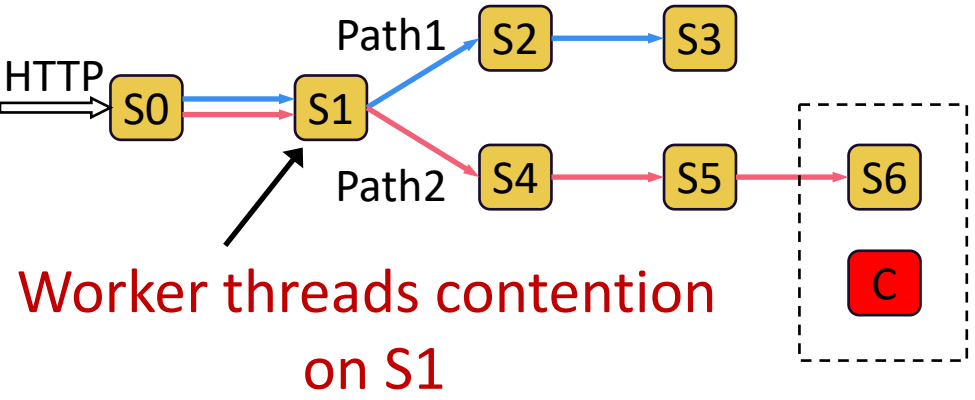


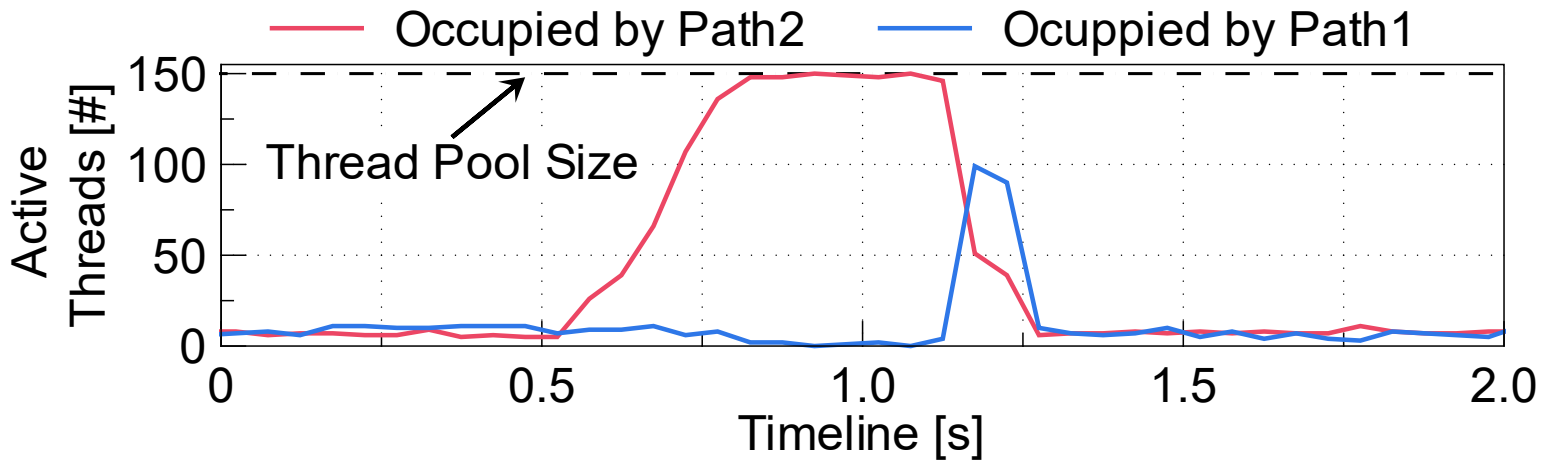
Worker Thread Contention → Long Response Time

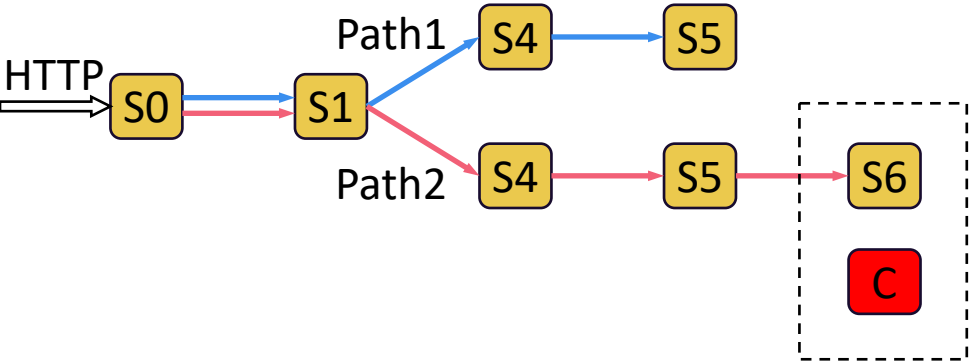




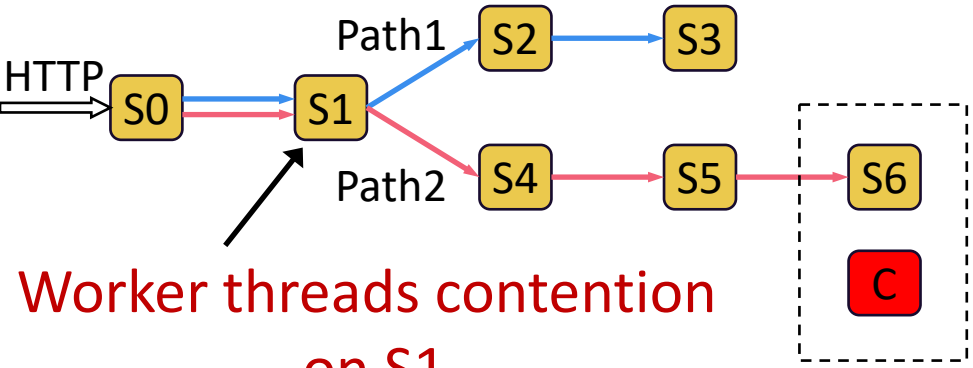
Worker Thread Contention → Long Response Time



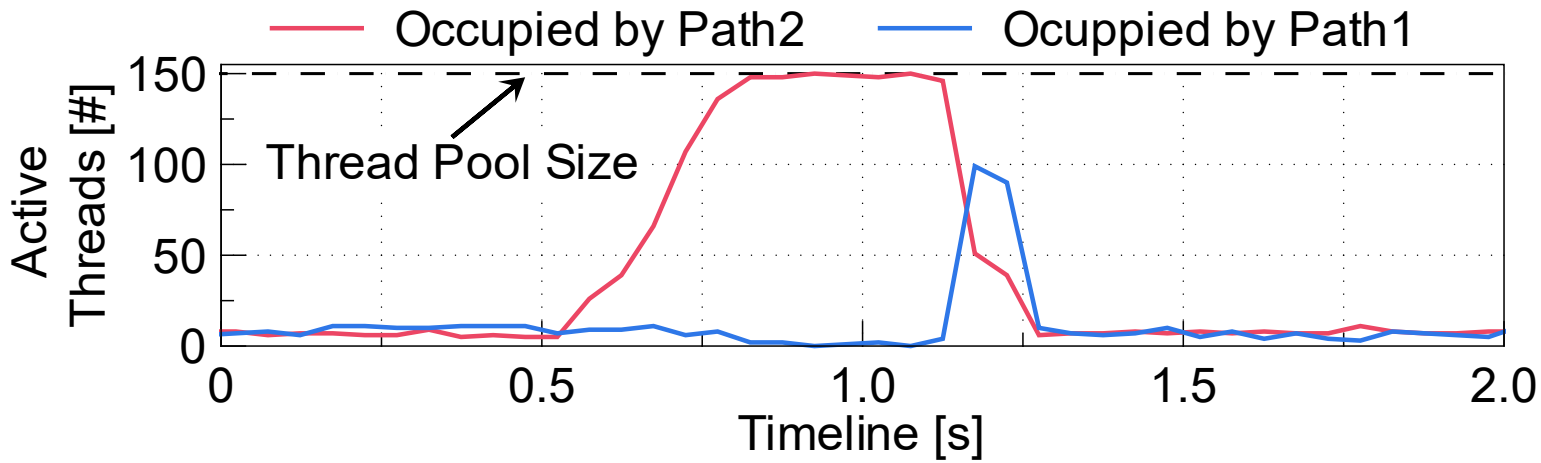


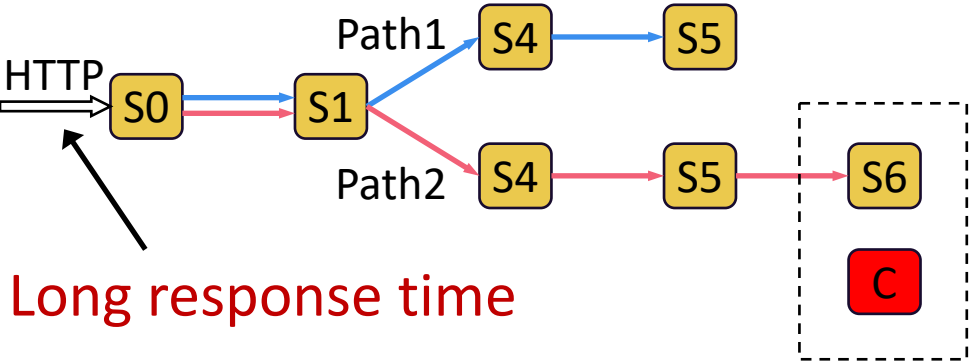


Worker Thread Contention → Long Response Time



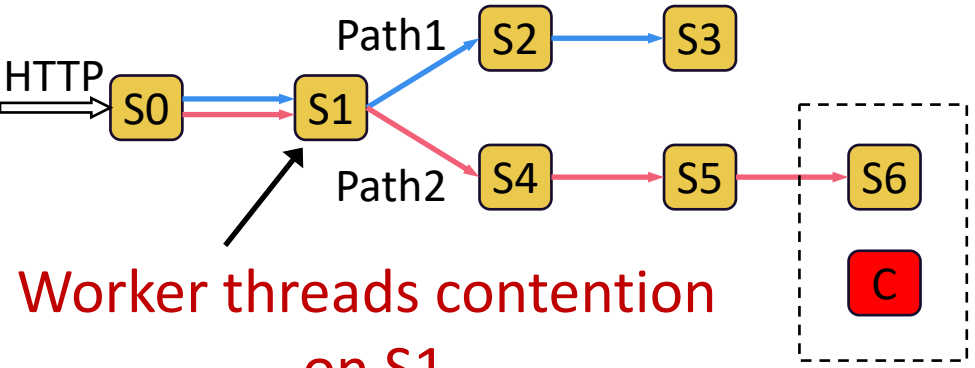
Worker threads contention on S1



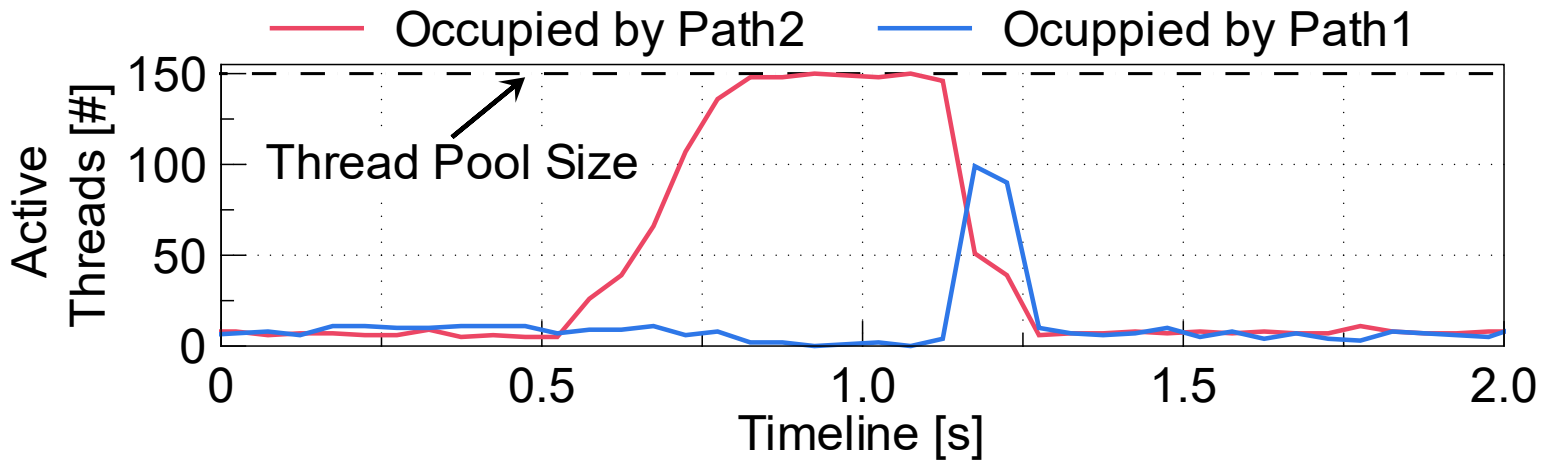


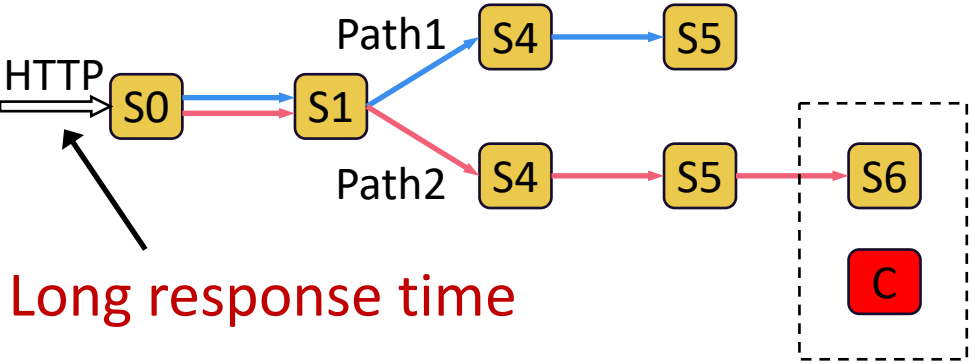
Long response time

Worker Thread Contention → Long Response Time

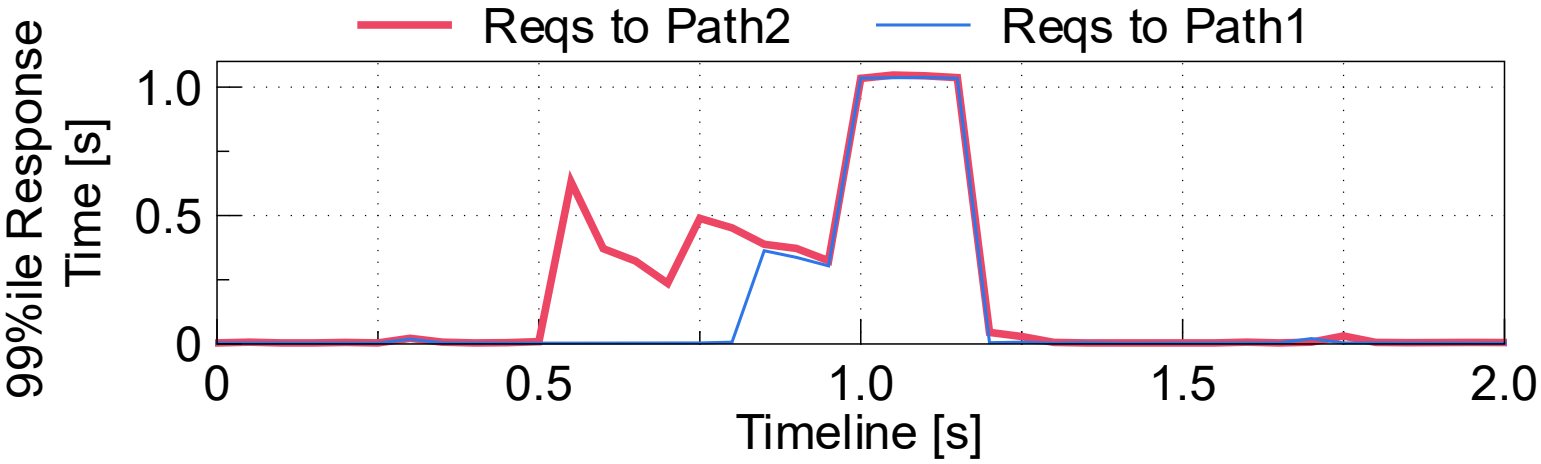


Worker threads contention on S1





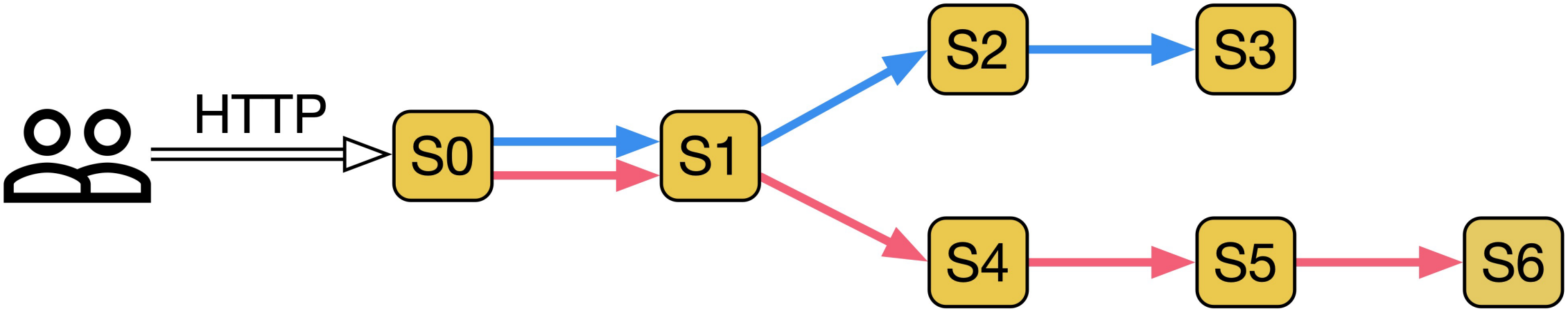
Long response time



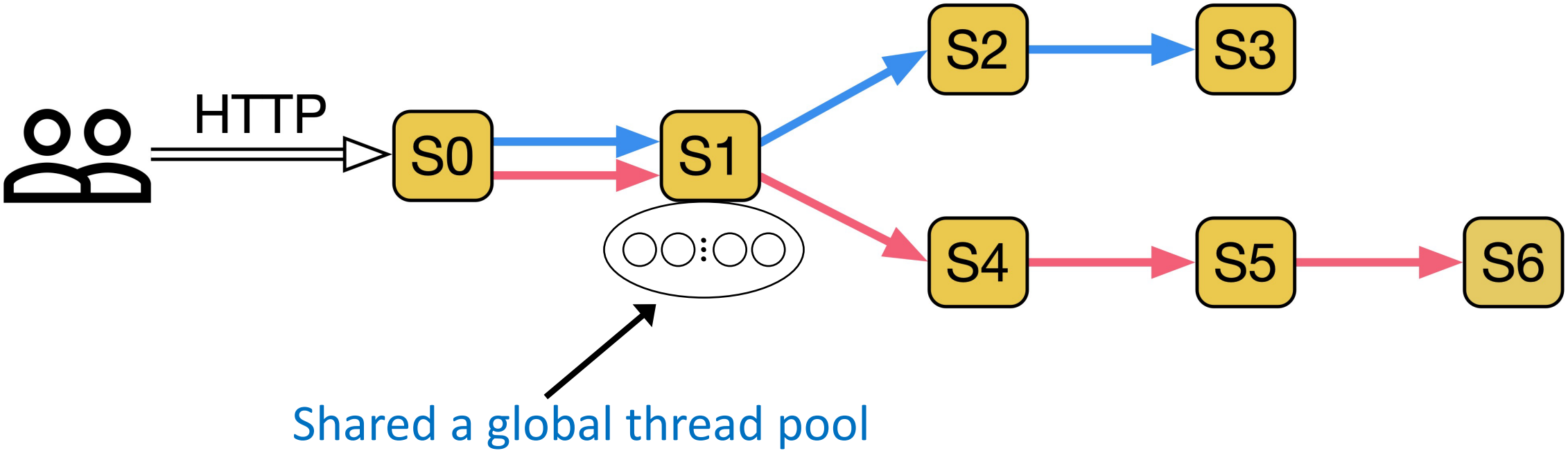
Other Sources of Bottleneck in Microservices

- **Bursty workload** from external users
- **Collocation interference** from internal containers or processes
- **Self-maintenance activities** within services
  - e.g., garbage collection (GC), log flushing
- **Potential bottleneck resources:**
  - Hardware resources (e.g., CPU, disk, memory bandwidth)
  - Soft resources (e.g., threads, database connection pool)

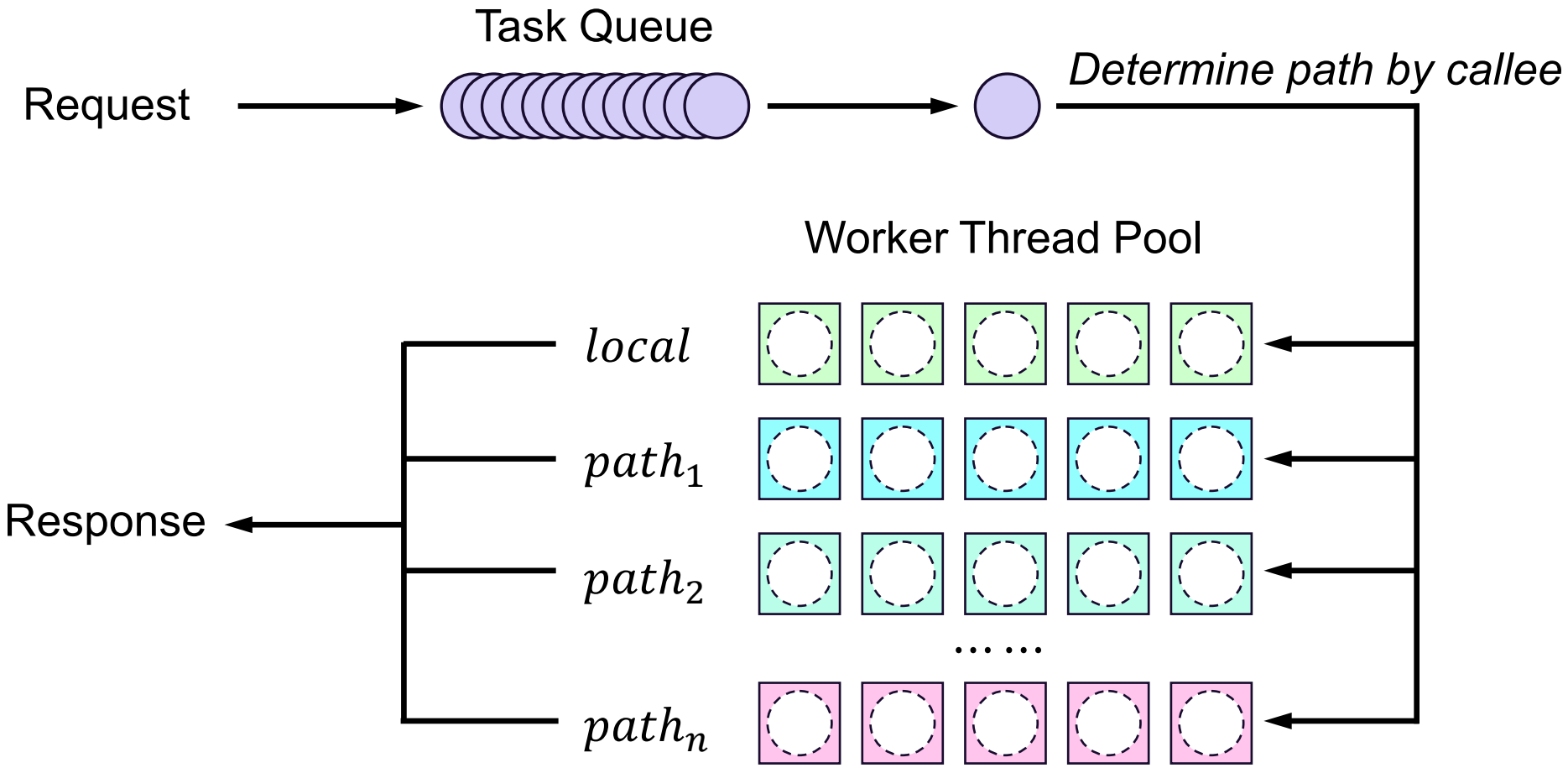
Root Cause: Shared Global Thread Pool



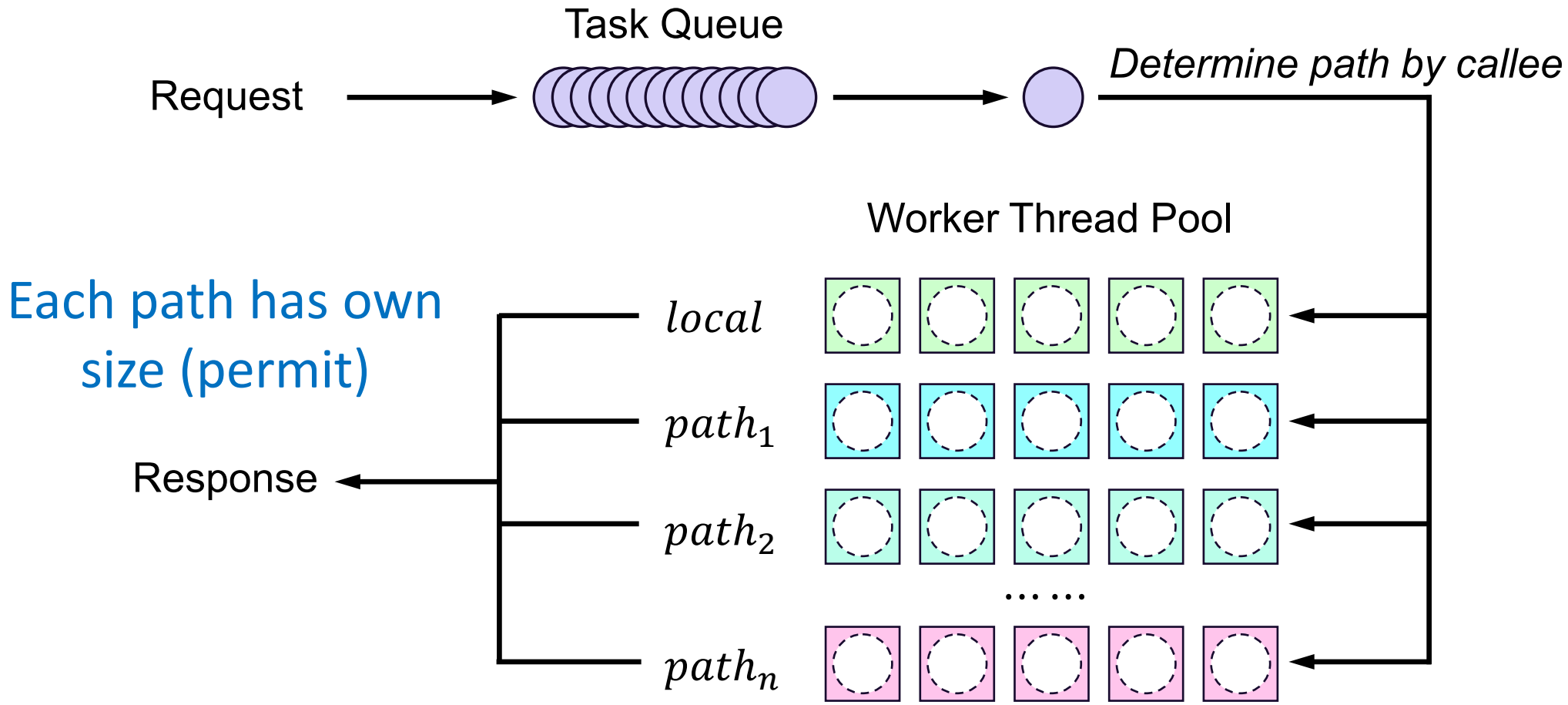
Root Cause: Shared Global Thread Pool



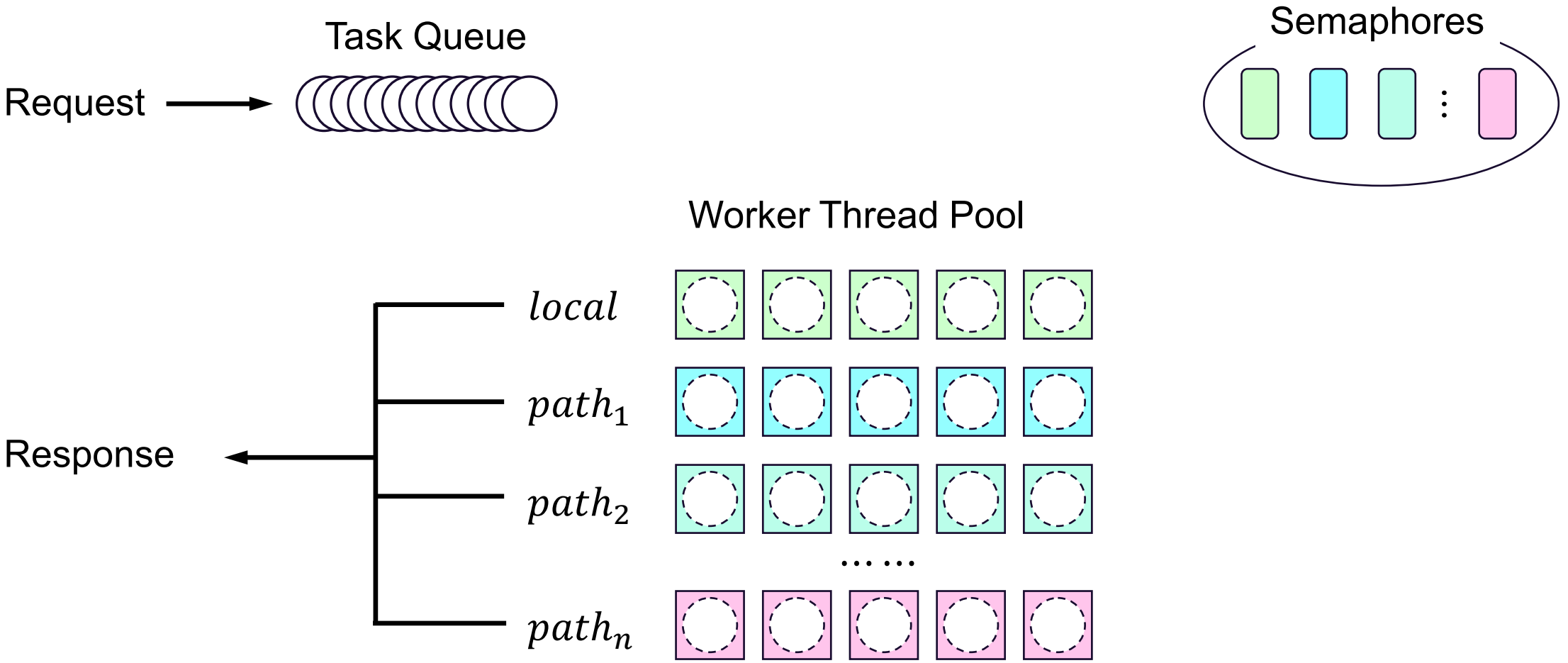
PathFence: Path-Aware Thread Scheduling



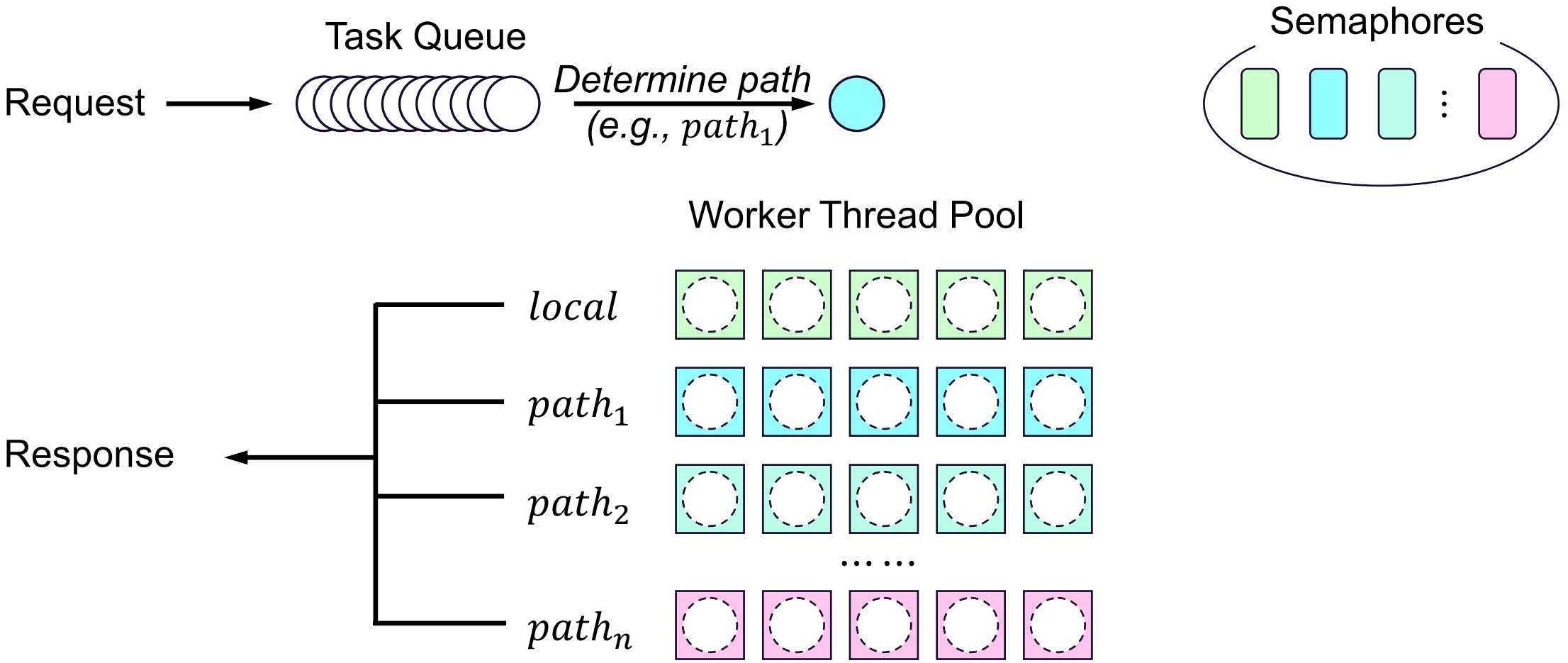
PathFence: Path-Aware Thread Scheduling



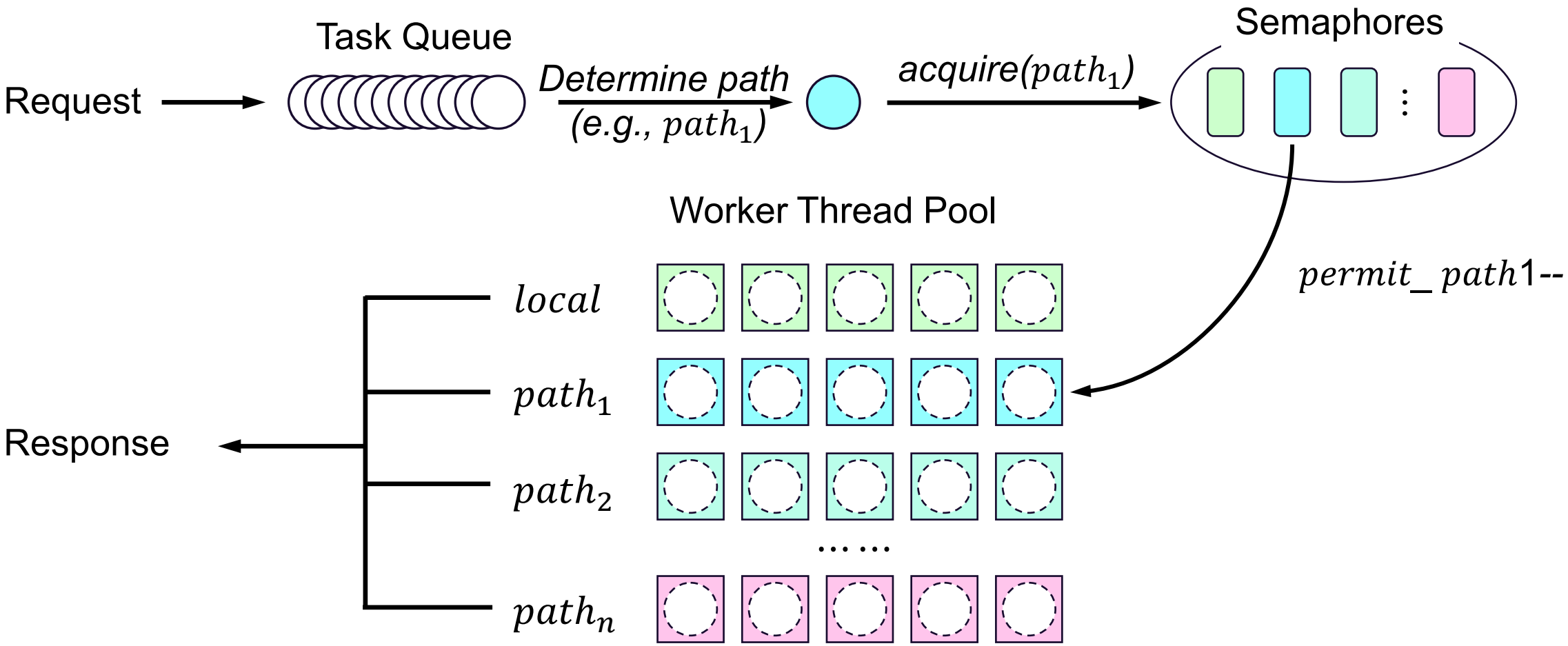
Semaphores-Based Thread Management



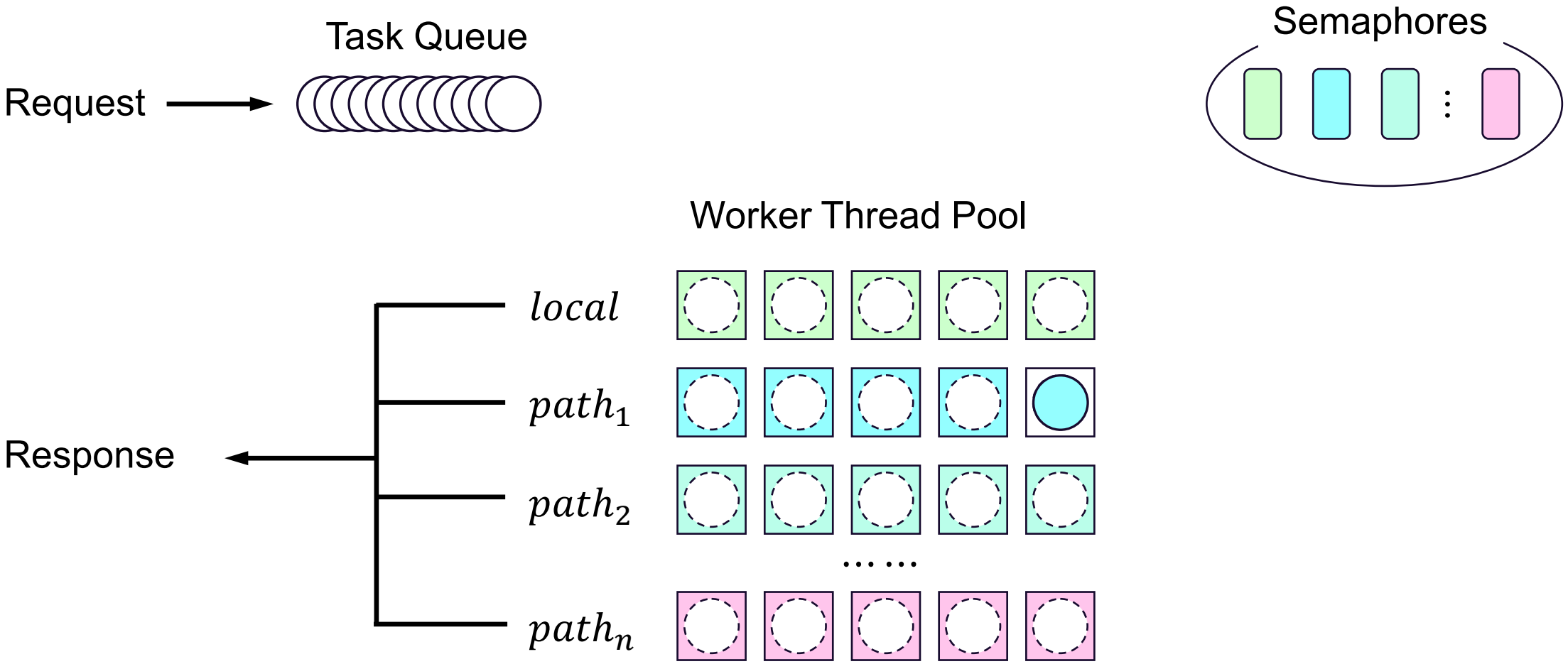
Semaphores-Based Thread Management



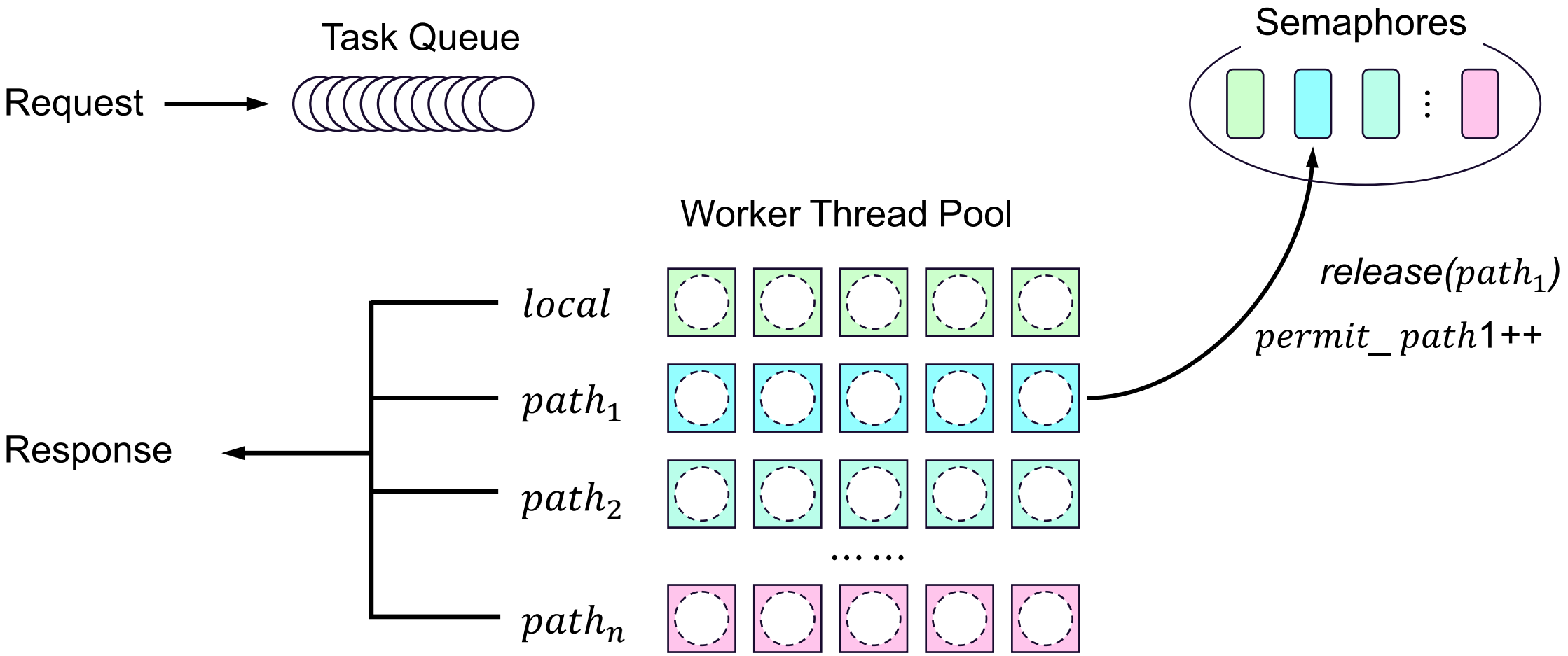
Semaphores-Based Thread Management



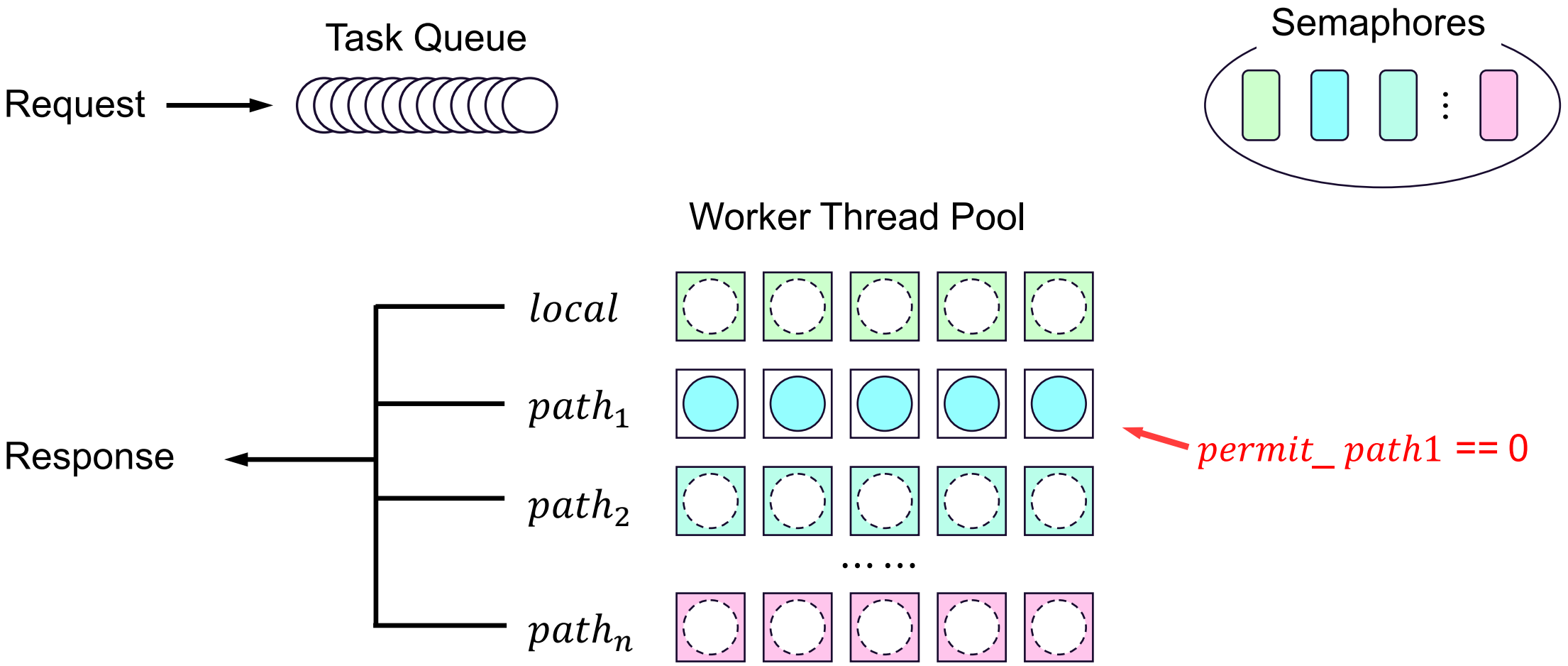
Semaphores-Based Thread Management



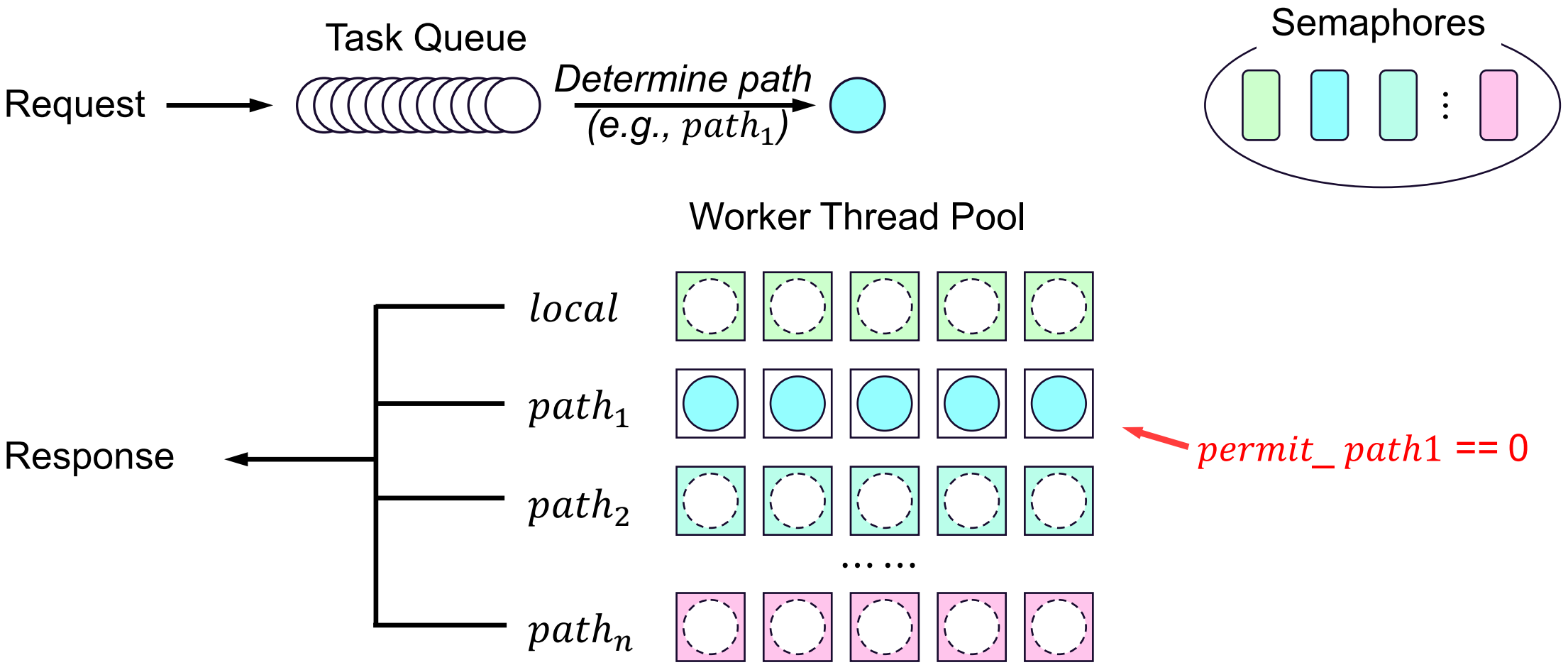
Semaphores-Based Thread Management



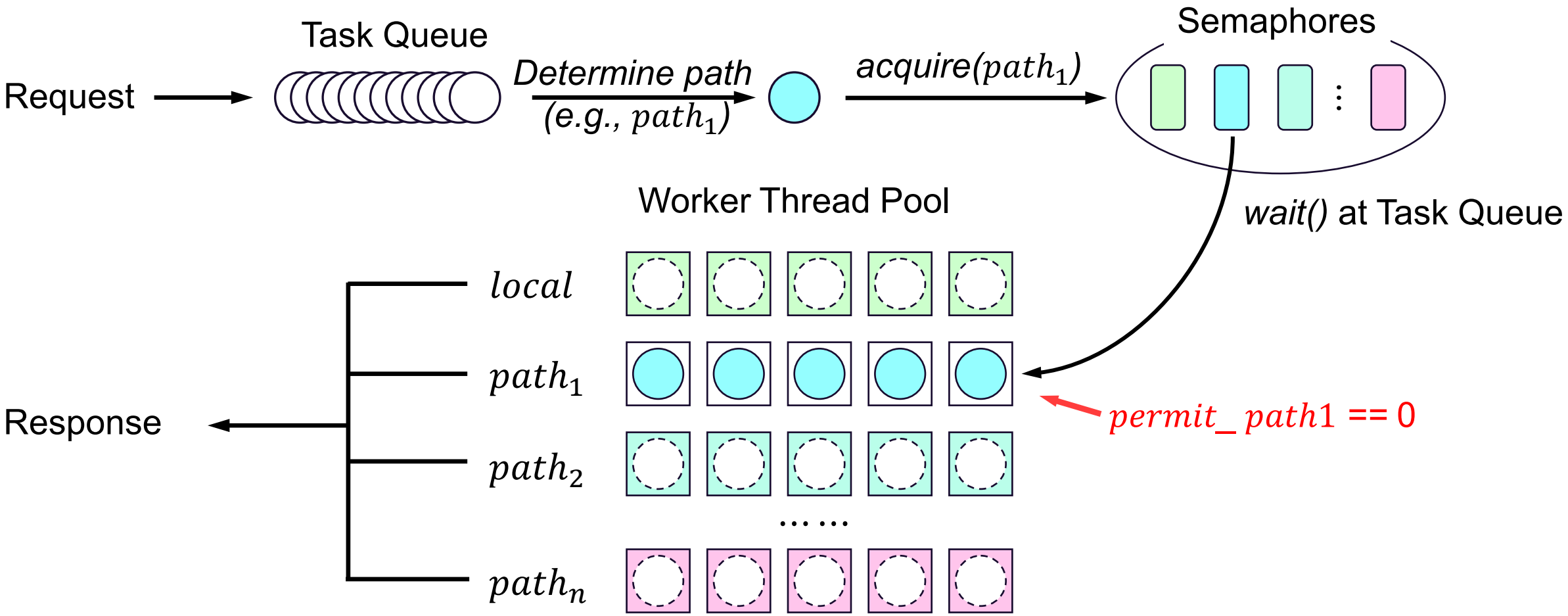
Eliminating Cross-Path Blocking



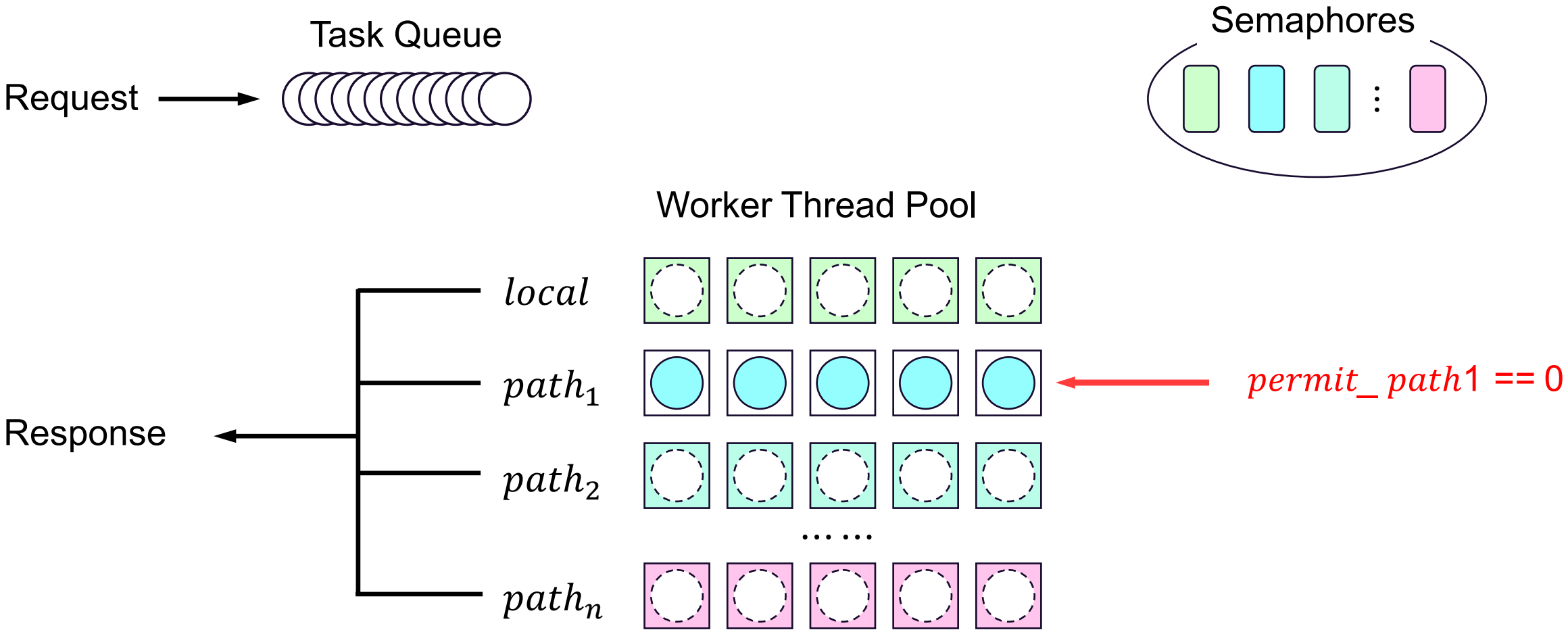
Eliminating Cross-Path Blocking



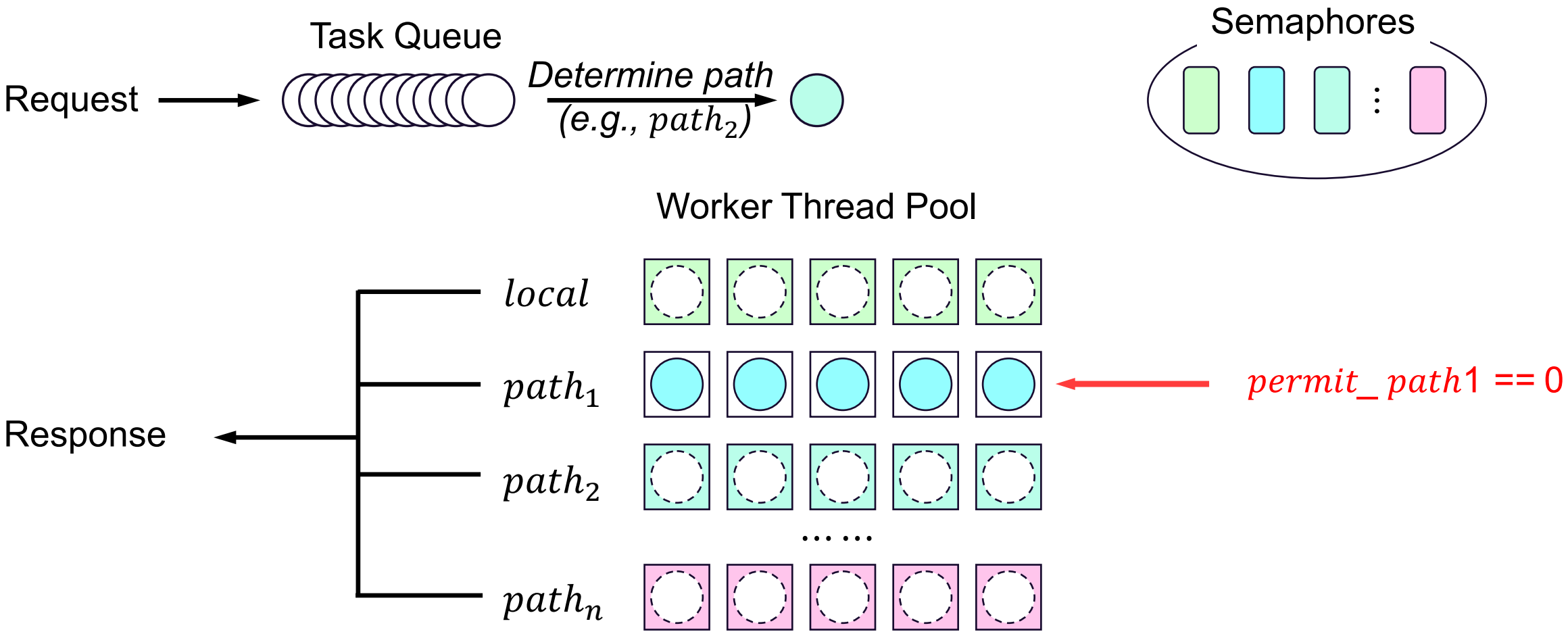
Eliminating Cross-Path Blocking



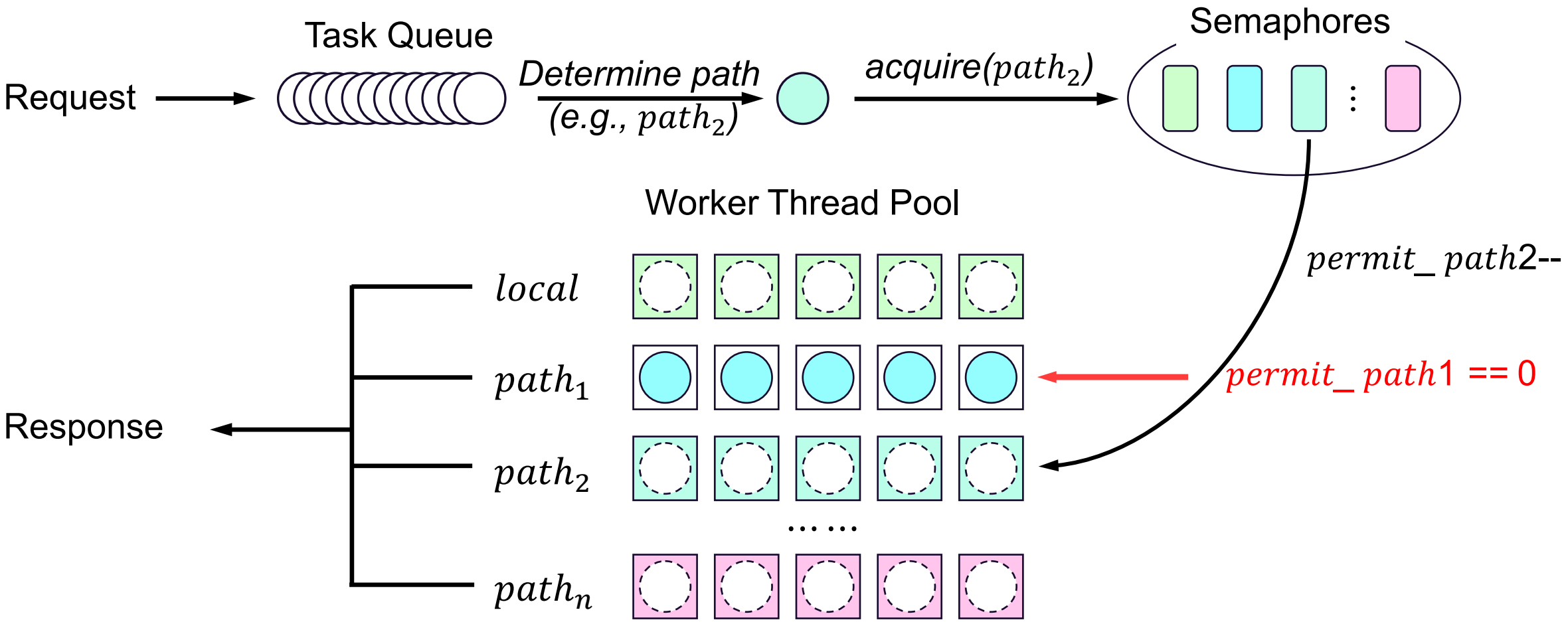
Eliminating Cross-Path Blocking



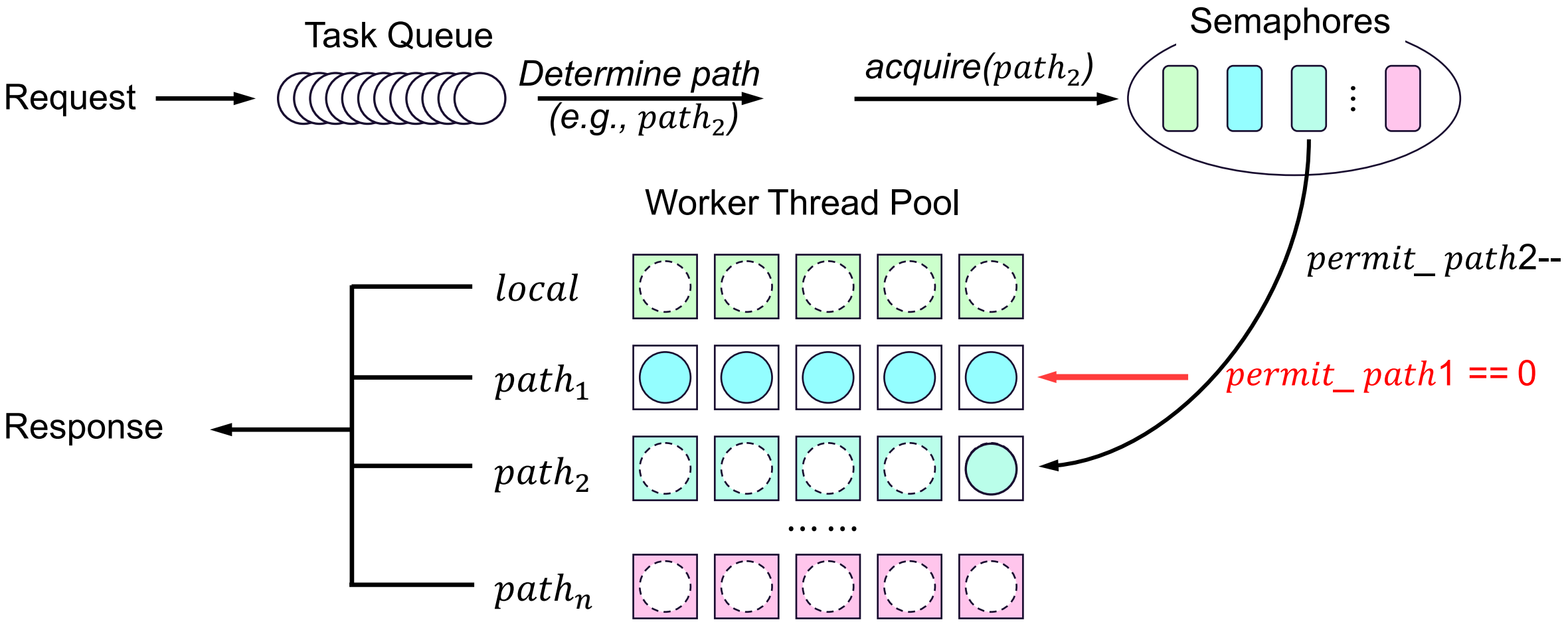
Eliminating Cross-Path Blocking



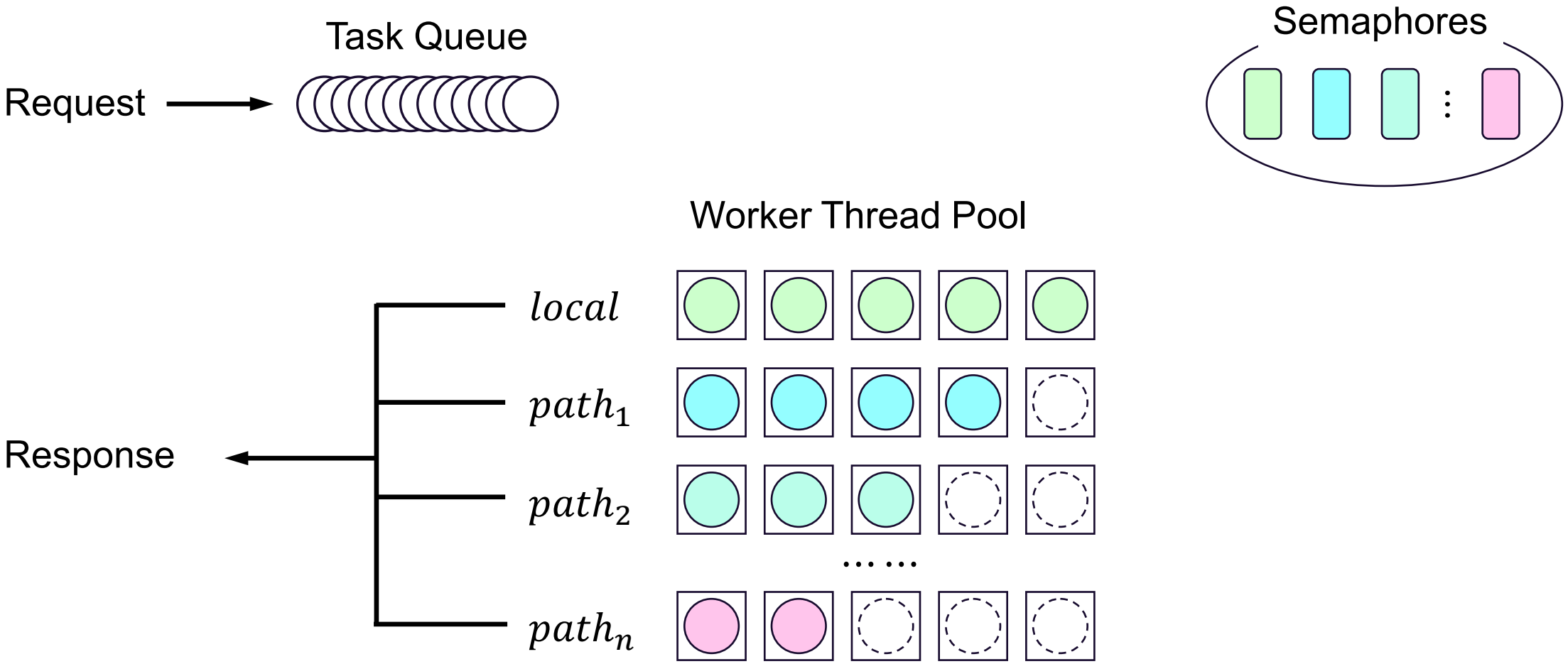
Eliminating Cross-Path Blocking



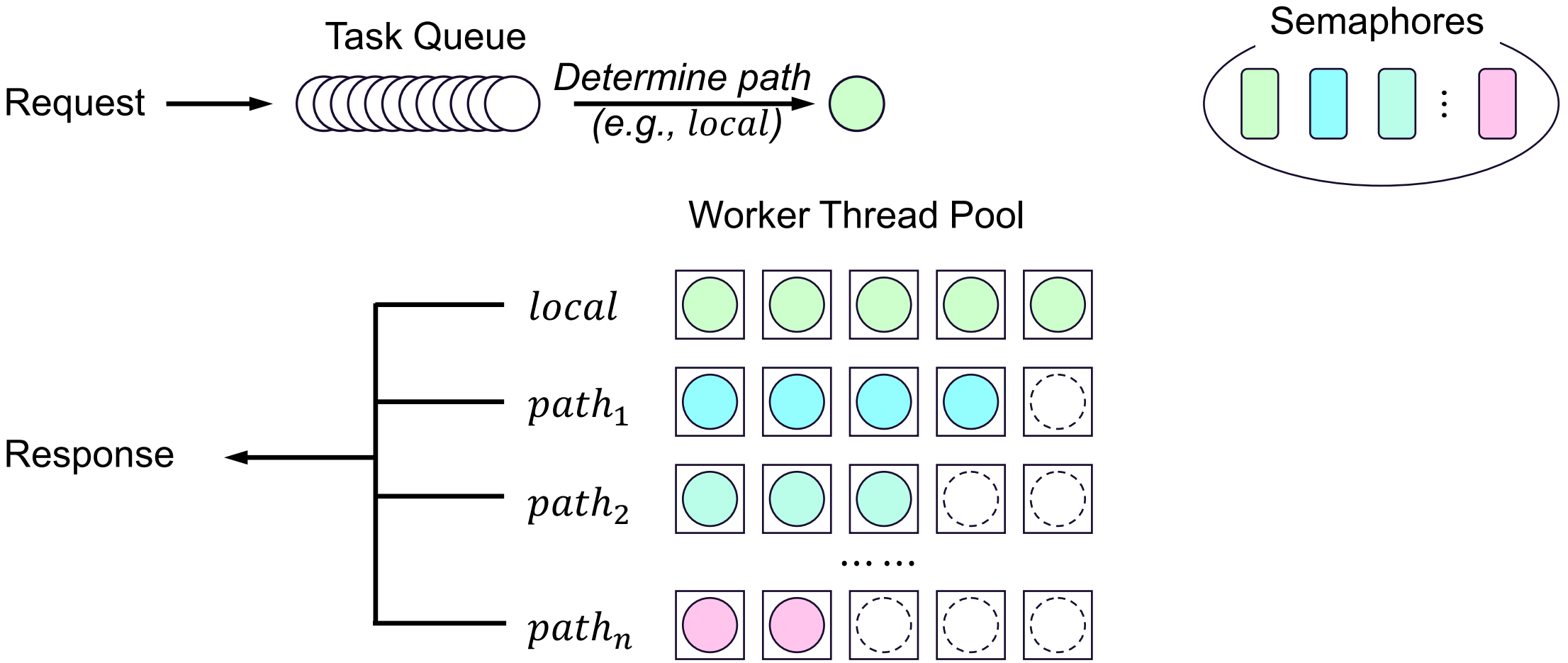
Eliminating Cross-Path Blocking



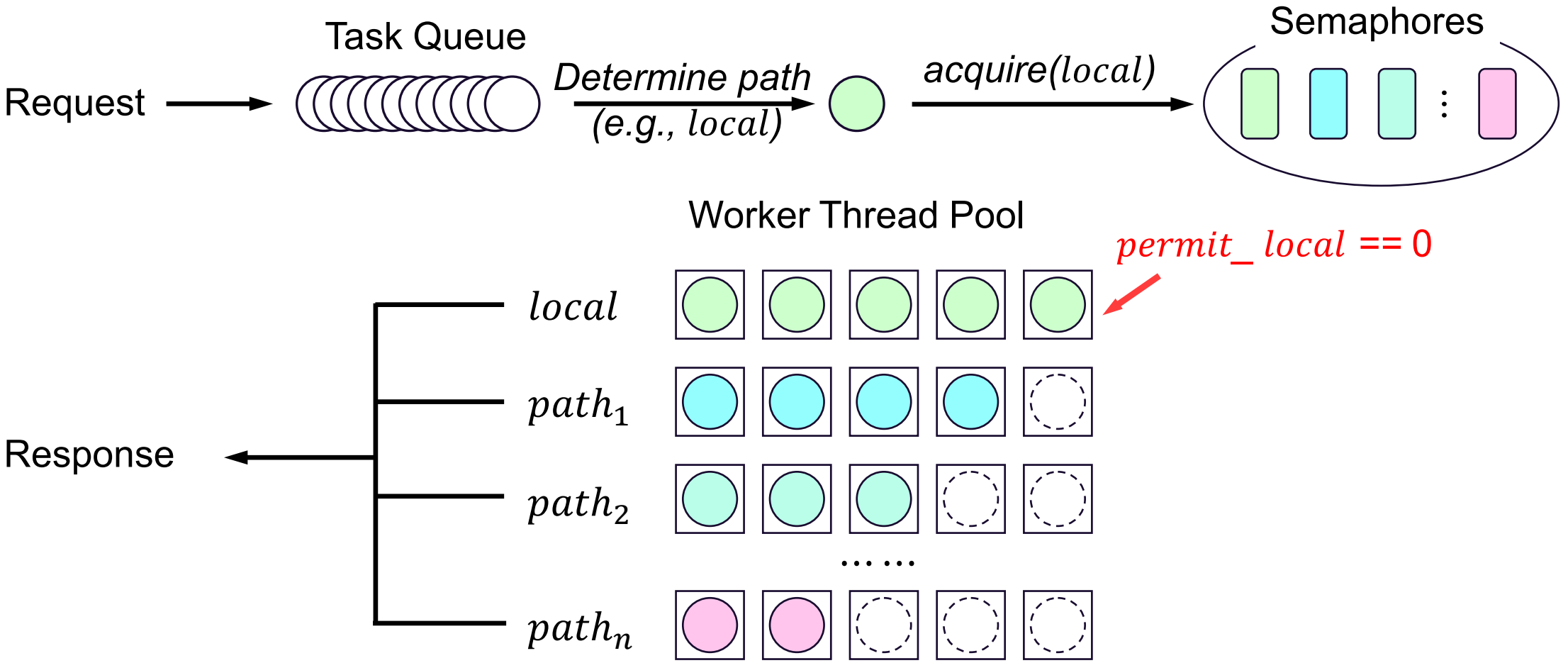
Prioritize Local Requests



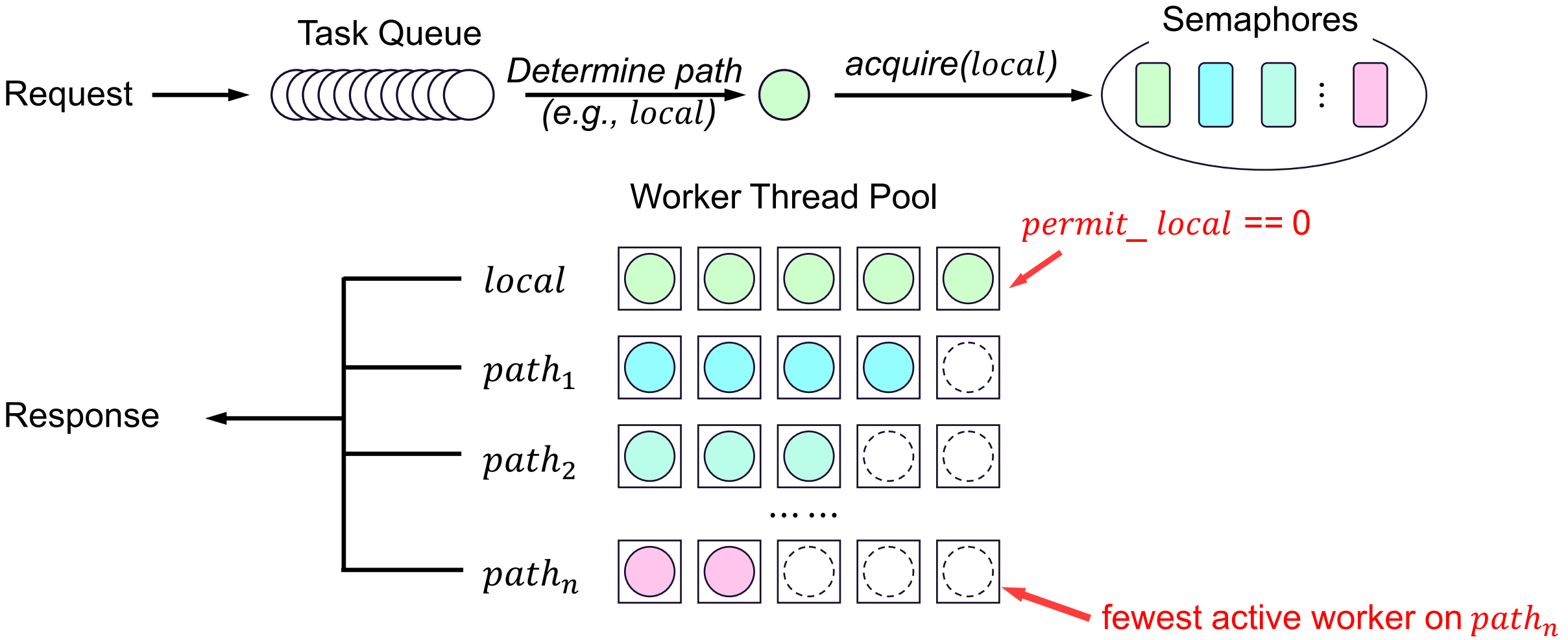
Prioritize Local Requests



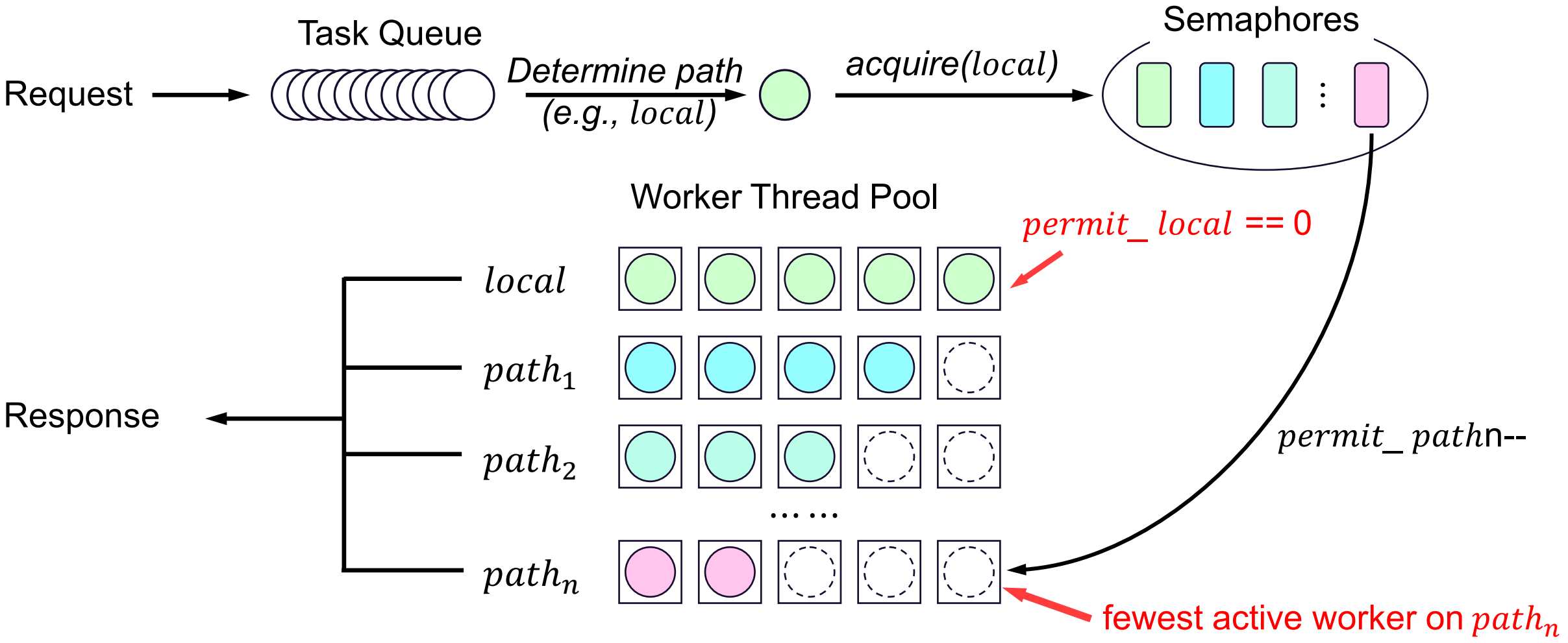
Prioritize Local Requests



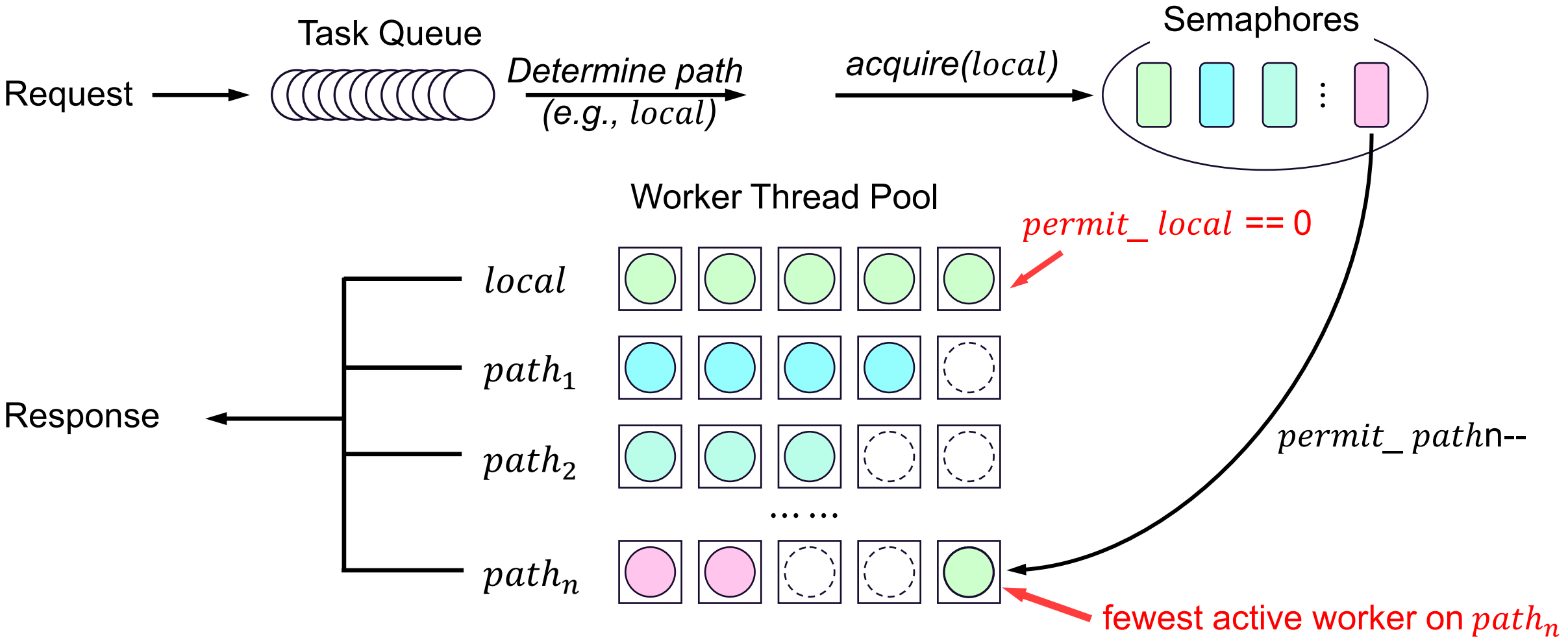
Prioritize Local Requests



Prioritize Local Requests

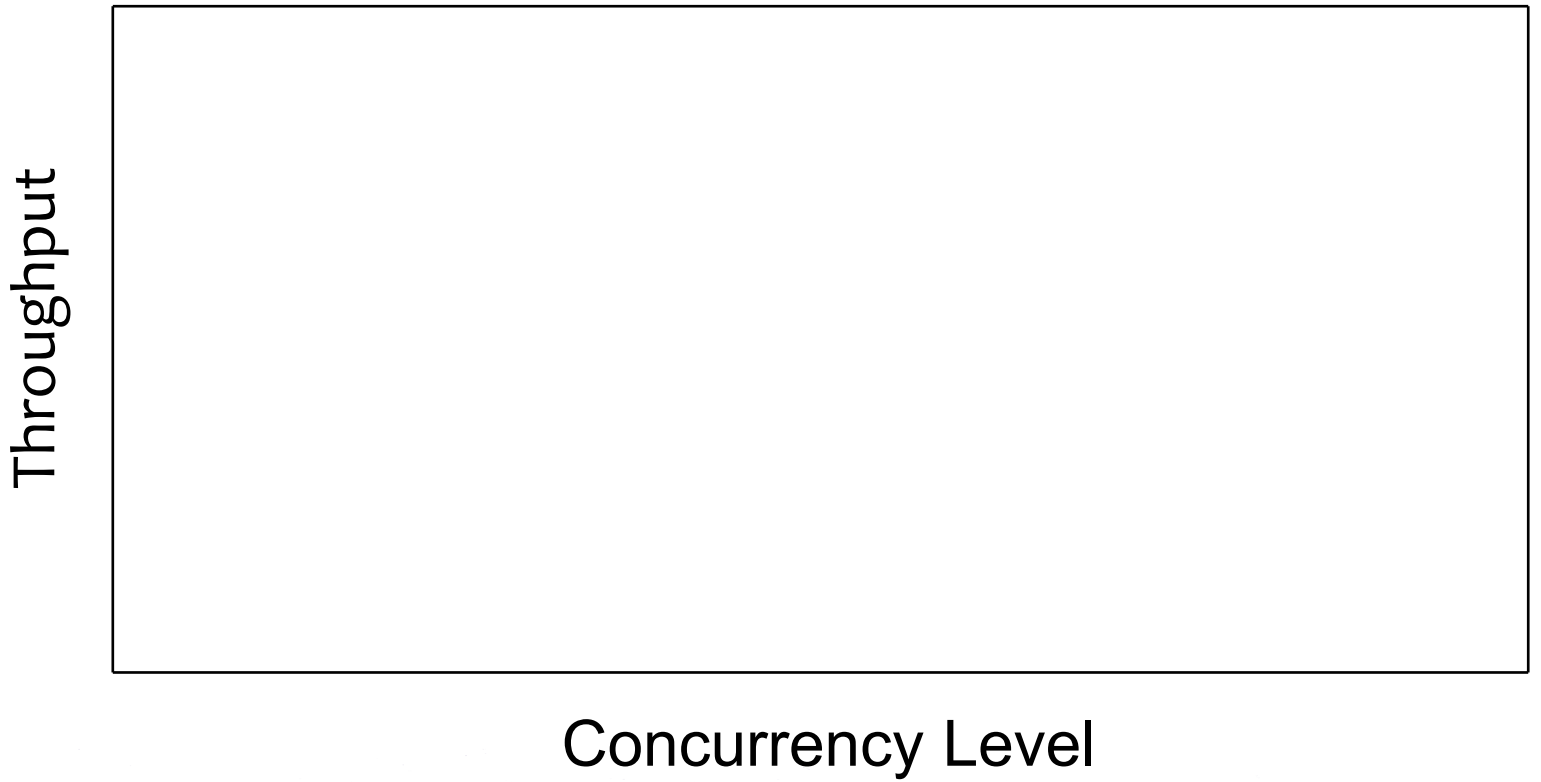


Prioritize Local Requests



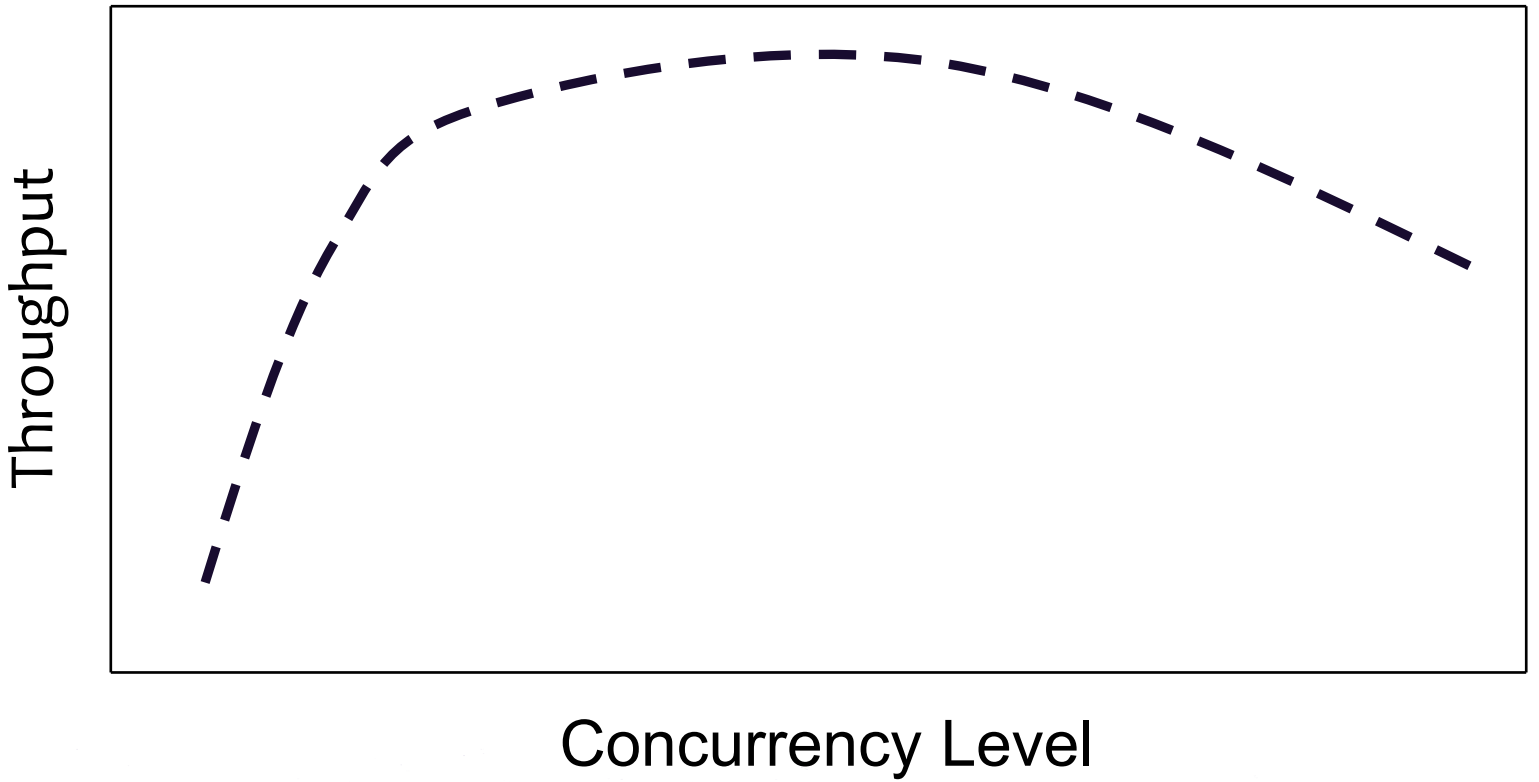
Determine Optimal Concurrency Level for Each Path

- Offline profiling + online updating
- Inspired by existing works
  - SCT model –[Liu et al. *IPDPS' 20*]
  - μTune –[Sriraman et al. *OSDI' 18*]



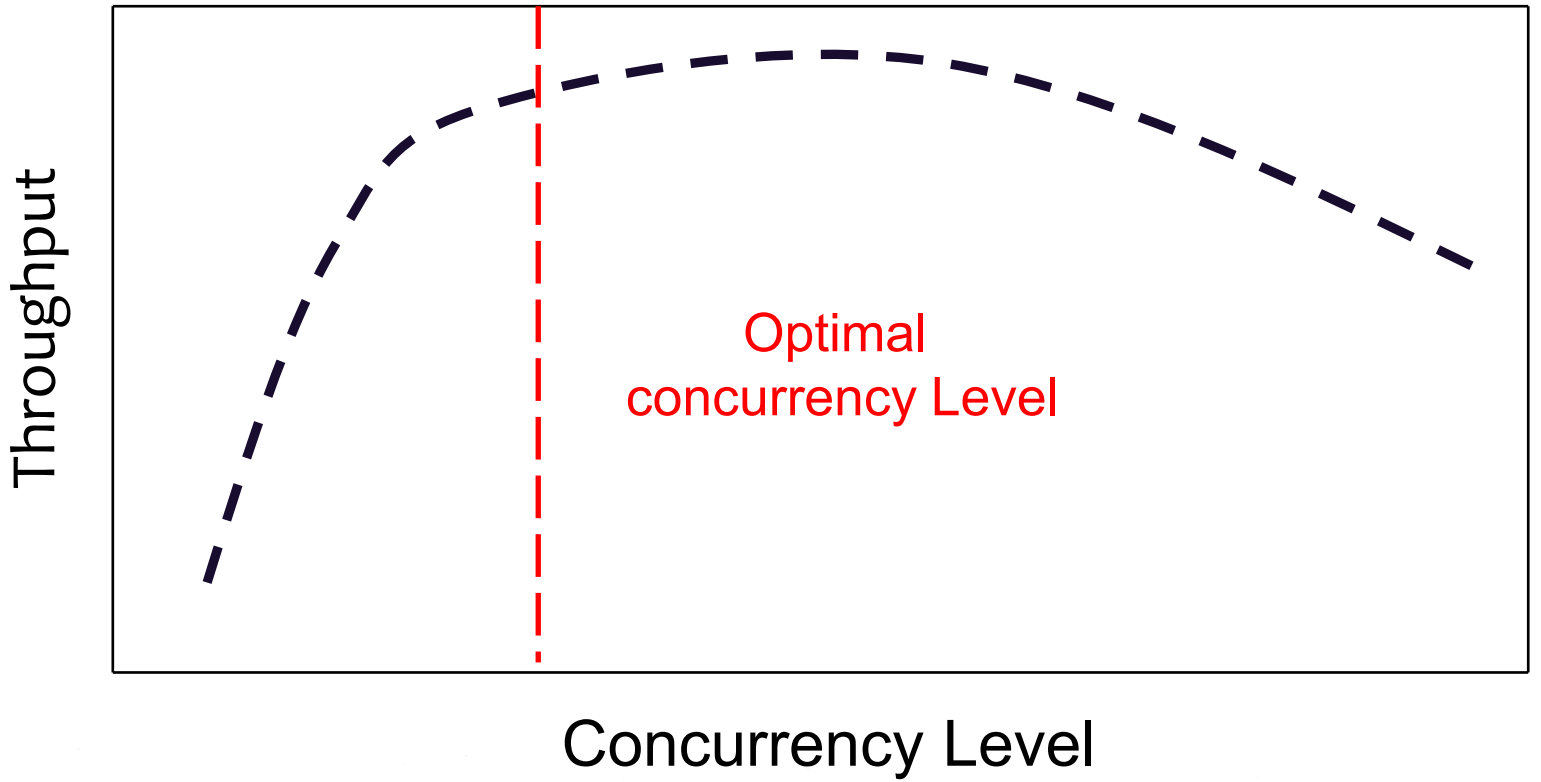
Determine Optimal Concurrency Level for Each Path

- Offline profiling + online updating
- Inspired by existing works
  - SCT model –[Liu et al. *IPDPS' 20*]
  - μTune –[Sriraman et al. *OSDI' 18*]



Determine Optimal Concurrency Level for Each Path

- Offline profiling + online updating
- Inspired by existing works
  - SCT model –[Liu et al. *IPDPS' 20*]
  - μTune –[Sriraman et al. *OSDI' 18*]

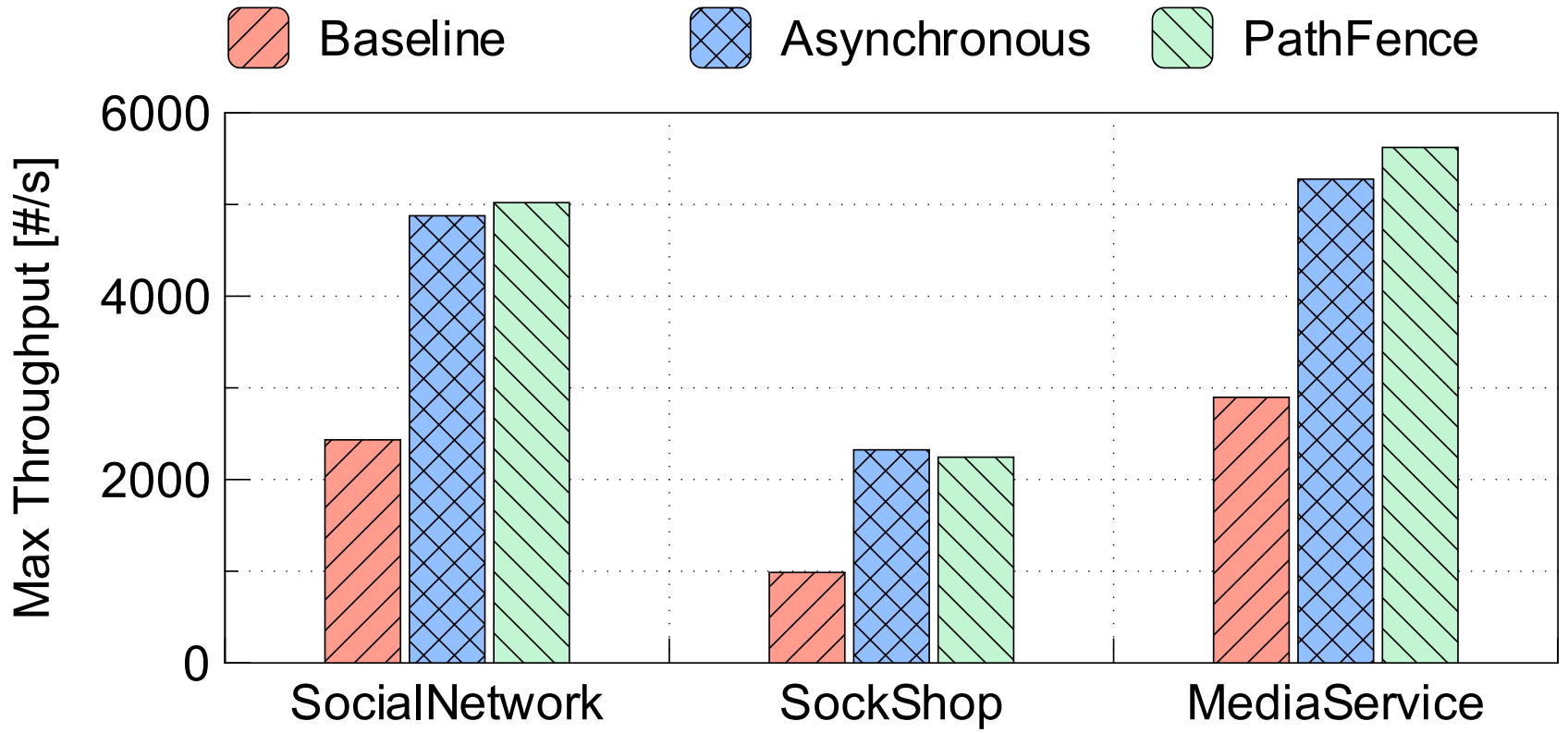


Implementation and Evaluation

- Implemented with **Apache Thrift** in C++ and **SpringBoot** in Java
- Deployed on a **Kubernetes cluster** of 20 physical nodes
- Evaluated with **3 real-world workload traces** and 3 **microservices benchmarks**
- Compare with state-of-the-art (**Asynchronous invocation**)

Results

- Increases the average system throughput by up to 2x
- Reduces the average tail latency by up to 80%
- Reduces the number of dropped/timed out requests by over 90%



Results

- Increases the average system throughput by up to 2x
- Reduces the average tail latency by up to 80%
- Reduces the number of dropped/timed out requests by over 90%

App	Workload Variations	Average Response Time [ms]			99th Percentile Response Time [ms]			Dropped Requests [#]		
		Base	Async.	Our	Base	Async.	Our	Base	Async.	Our
SS	Large	249	171	153	3013	378	394	1833	0	0
	Quick	217	165	158	2374	319	327	797	0	0
	Slow	208	161	143	1178	293	299	273	0	0
SN	Large	314	255	223	7045	1094	1037	5981	173	189
	Quick	289	253	218	3097	1075	1021	2973	134	181
	Slow	273	244	252	1329	598	633	533	0	0
MS	Large	445	397	283	3470	1233	1092	3987	239	204
	Quick	416	311	332	3511	1204	997	2429	131	87
	Slow	391	297	286	1573	871	453	1266	0	0

Conclusion

Conclusion

- **Empirical study** of **response time instability**
  - Cross-path dependencies

Conclusion

- **Empirical study** of **response time instability**
  - Cross-path dependencies
- **Common factors** cause resource contentions
  - Internal collocation interference
  - External bursty workload

Conclusion

- **Empirical study** of **response time instability**
  - Cross-path dependencies
- **Common factors** cause resource contentions
  - Internal collocation interference
  - External bursty workload
- **Path-awareness** soft resources (i.e., threads) scheduling
  - Avoiding soft resource contention
  - Reducing cross-path dependencies

Conclusion

- **Empirical study** of **response time instability**
  - Cross-path dependencies
- **Common factors** cause resource contentions
  - Internal collocation interference
  - External bursty workload
- **Path-awareness** soft resources (i.e., threads) scheduling
  - Avoiding soft resource contention
  - Reducing cross-path dependencies
- PathFence **achieve high performance** as Asynchronous invocation
  - Significantly reducing programming overhead

Thank you!

Check out the full paper for more details!
(xgu5@lsu.edu)