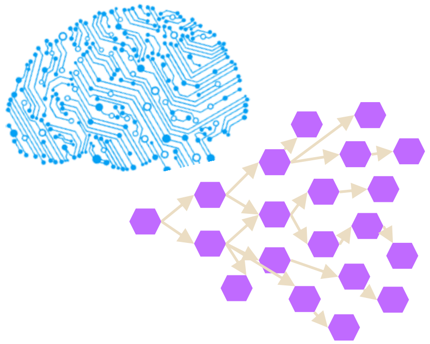


SEER: LEVERAGING BIG DATA TO NAVIGATE THE COMPLEXITY OF PERFORMANCE DEBUGGING IN CLOUD MICROSERVICES

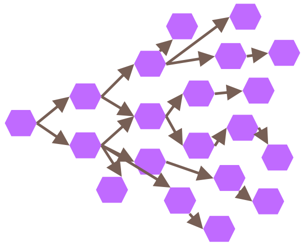
Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng,
Yuan He, Meghna Pancholi, and Christina Delimitrou

Cornell University

ASPLOS – April 15th 2019

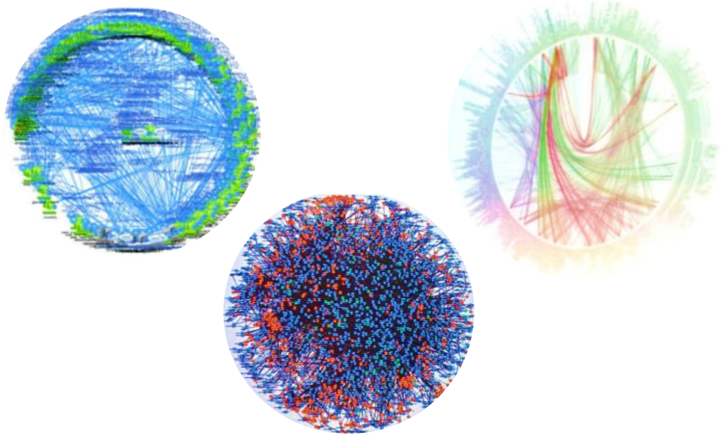
Executive Summary

- From monoliths to microservices:
  - Monoliths → all functionality in a single service
  - Microservices → many single-concerned, loosely-coupled services

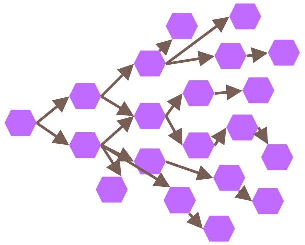


Executive Summary

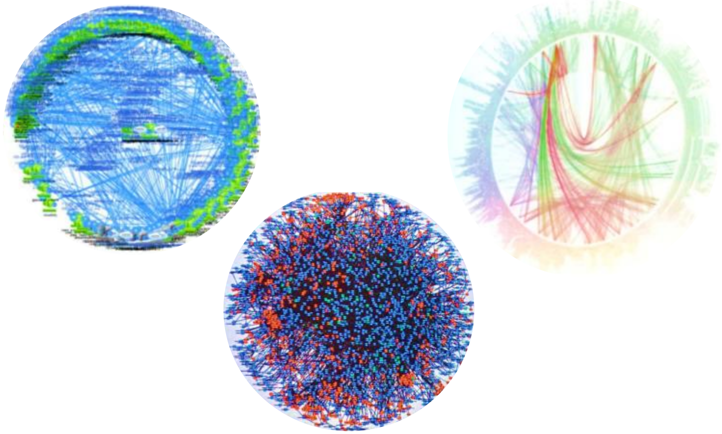
- From monoliths to microservices:
  - Monoliths → all functionality in a single service
  - Microservices → many single-concerned, loosely-coupled services



Executive Summary

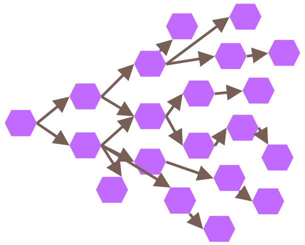


- From monoliths to microservices:
  - Monoliths → all functionality in a single service
  - Microservices → many single-concerned, loosely-coupled services

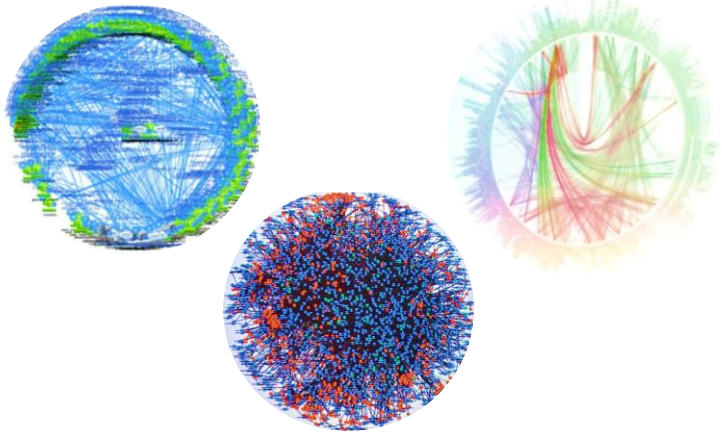


- Microservices implications:
  - Modularity, specialization, faster development
  - Performance unpredictability (us-level QoS), cascading QoS violations → A-posteriori debugging

Executive Summary



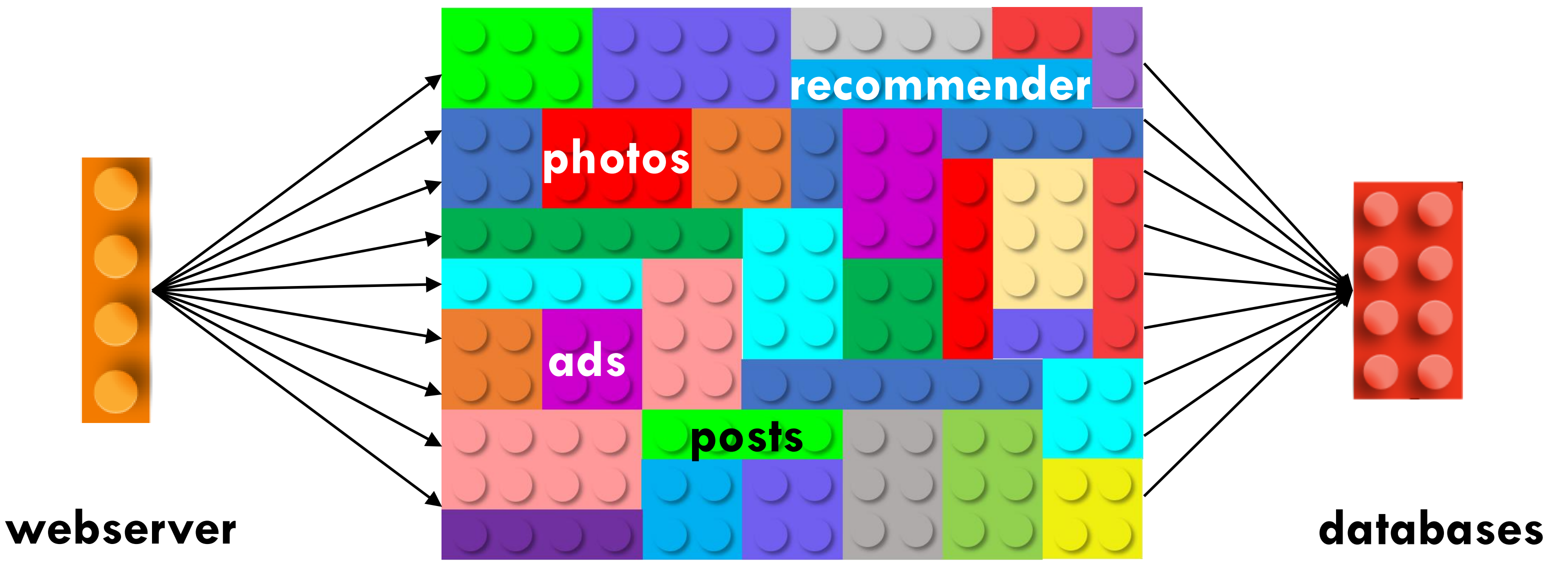
- From monoliths to microservices:
  - Monoliths → all functionality in a single service
  - Microservices → many single-concerned, loosely-coupled services



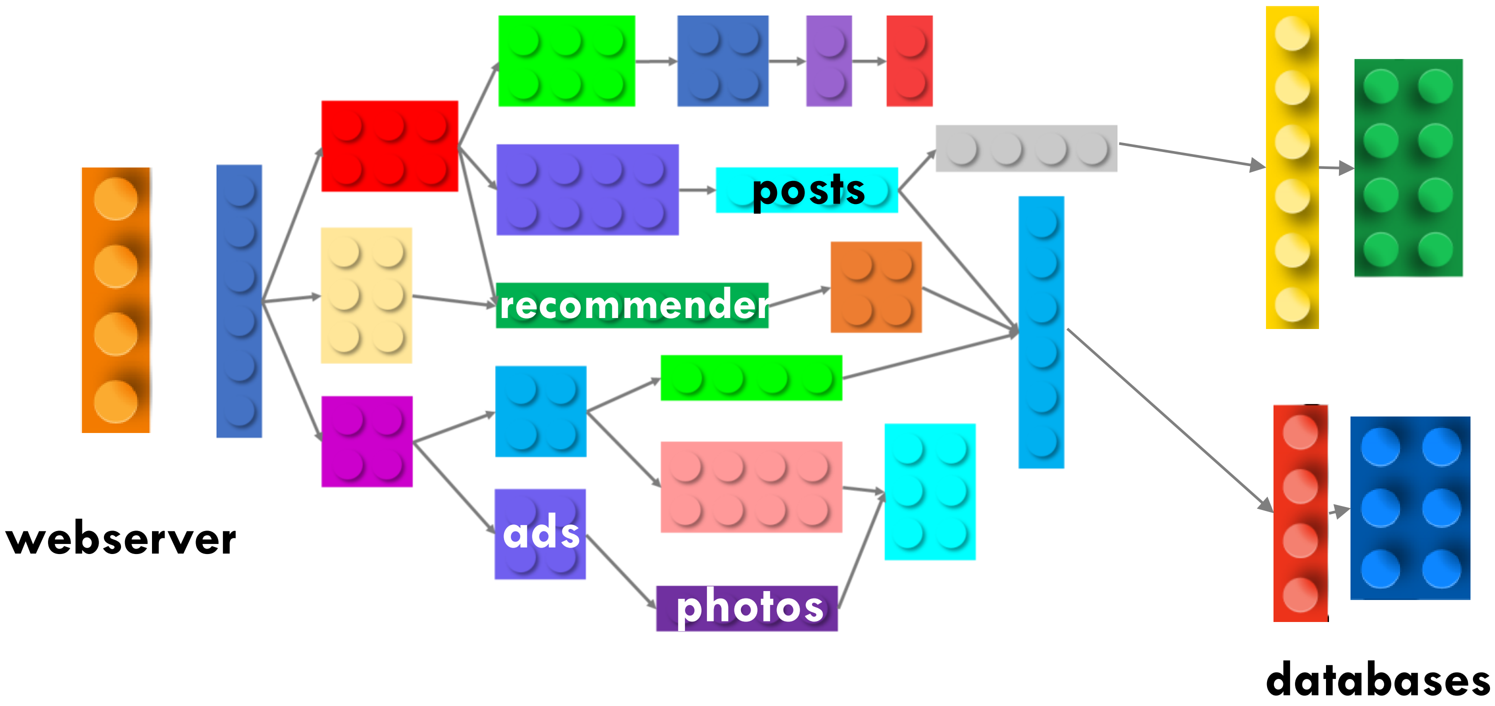
- Microservices implications:
  - Modularity, specialization, faster development
  - Performance unpredictability (us-level QoS), cascading QoS violations → A-posteriori debugging

- Seer: Proactive performance debugging for interactive microservices
  - Leverage DL to anticipate & diagnose root cause of QoS violations
  - >90% accuracy on large-scale end-to-end microservices deployments
  - Avoid unpredictable performance
  - Offer insight to improve microservices design and deployment

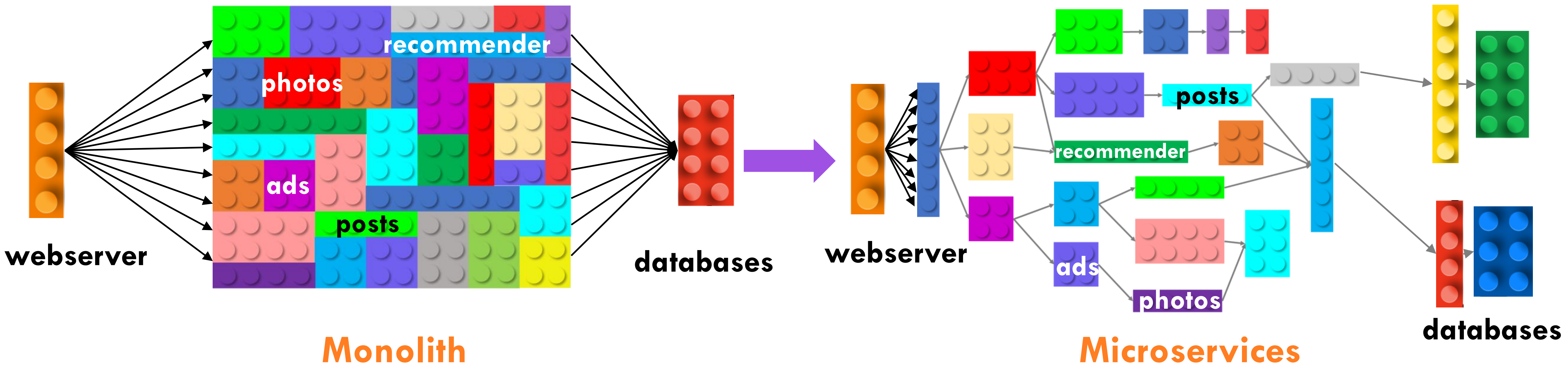
Motivation



Motivation

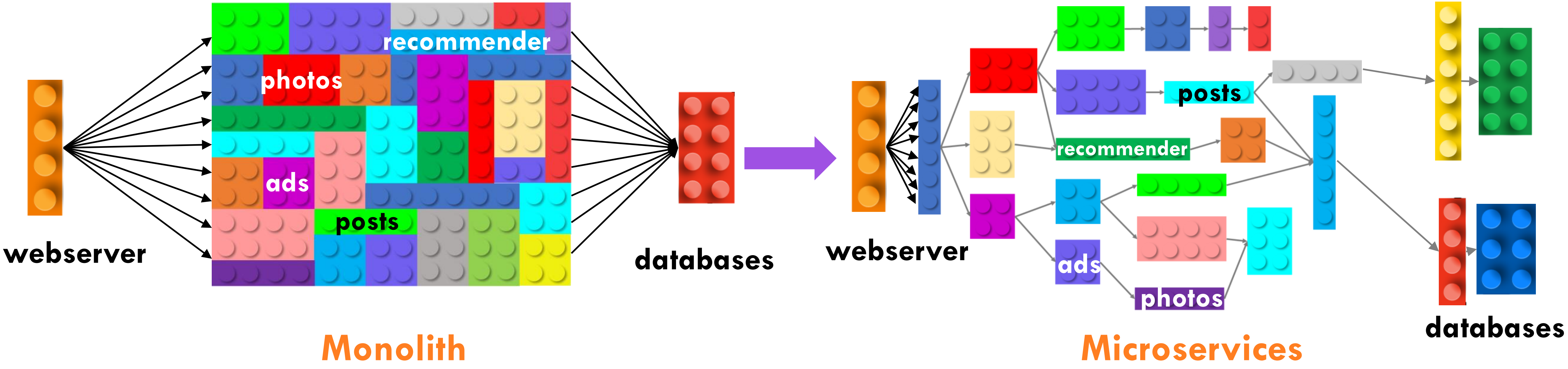


Motivation



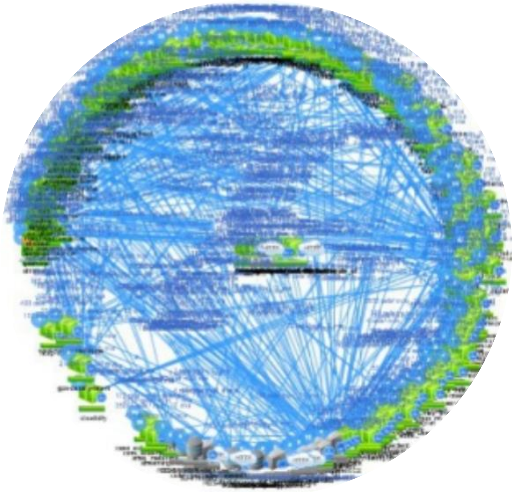
webserver
recommender
photos
ads
posts
databases
Monolith
webserver
posts
recommender
ads
photos
databases
Microservices

Motivation



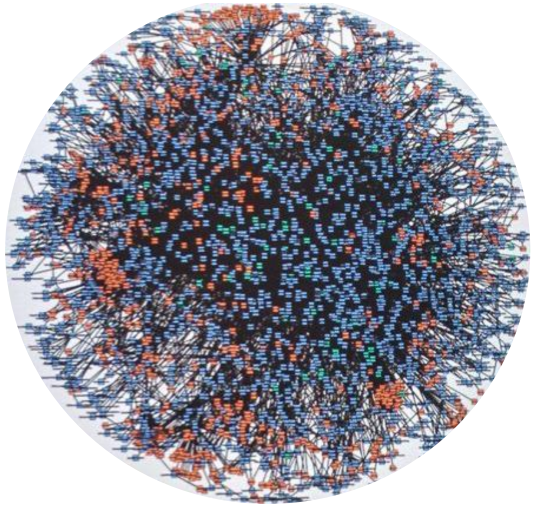
- Advantages of microservices:
  - Modular → easier to understand
  - Speed of development & deployment
  - On-demand provisioning, elasticity
  - Language/framework heterogeneity

Performance Debugging Challenges



Netflix

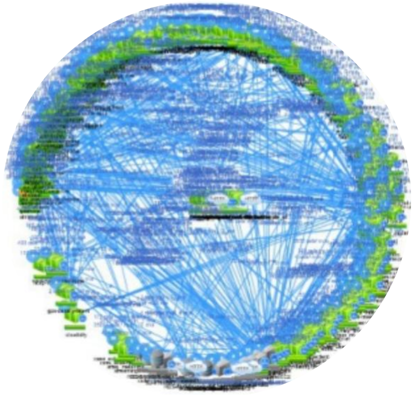
Twitter



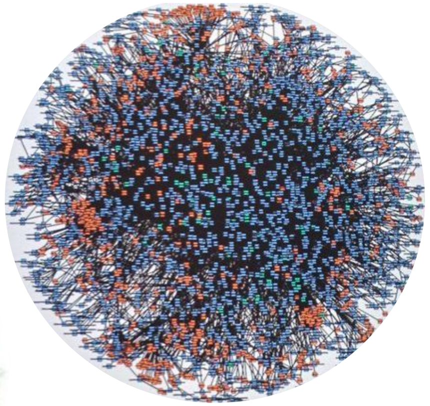
Amazon

- Complicate cluster management & performance debugging
- Dependencies cause cascading QoS violations
- Difficult to isolate root cause of performance unpredictability

Performance Debugging Challenges


Netflix


Twitter

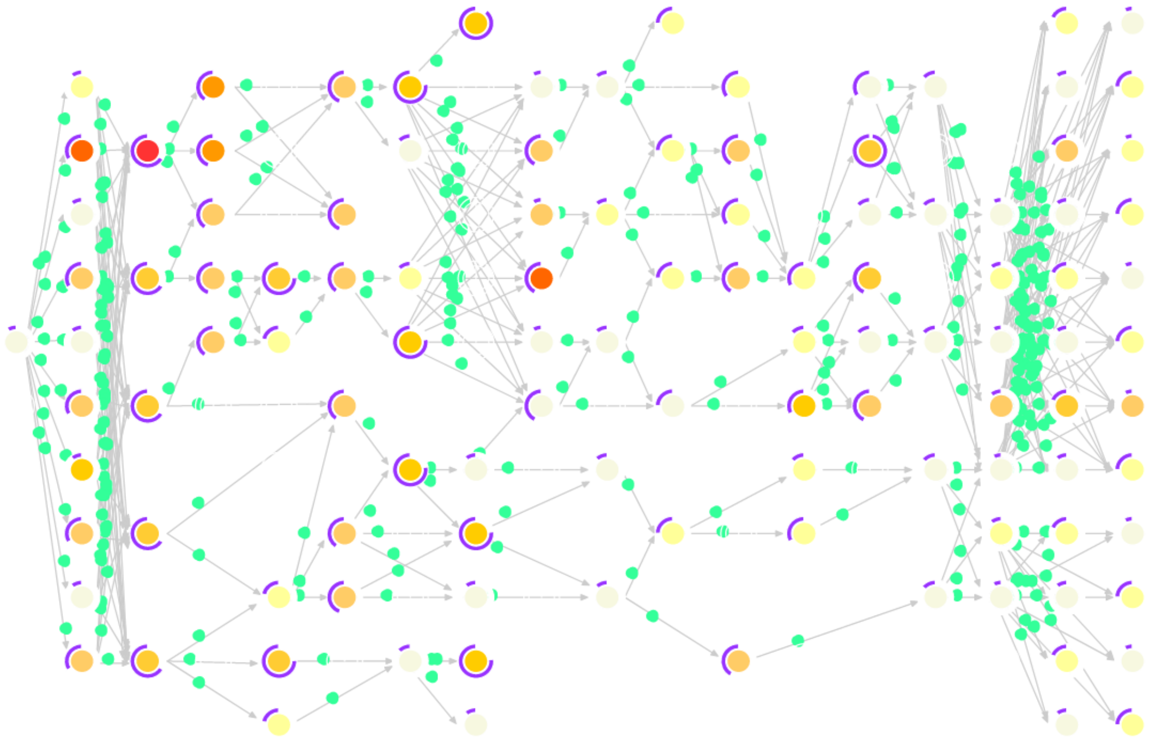

Amazon

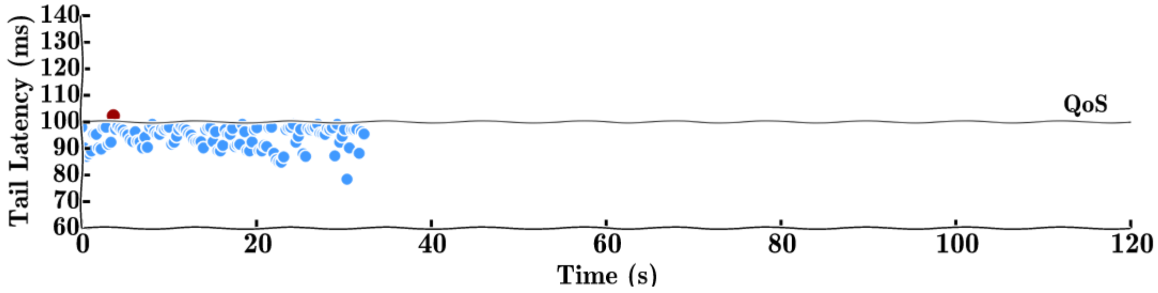
- Complicate cluster management & performance debugging
- Dependencies cause cascading QoS violations
- Difficult to isolate root cause of performance unpredictability

Performance Debugging Challenges

- Dependencies cause cascading QoS violations
- Empirical performance debugging → too slow, bottlenecks propagate
- Long recovery times for performance

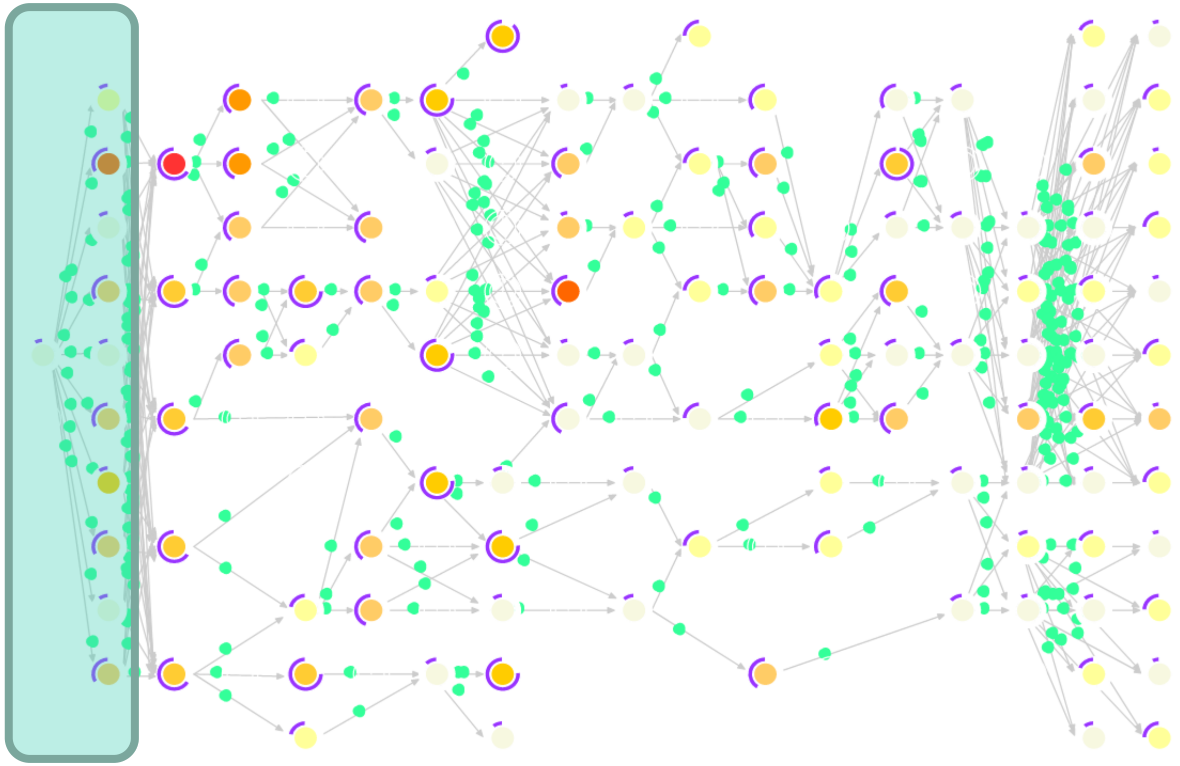
Performance Debugging Challenges

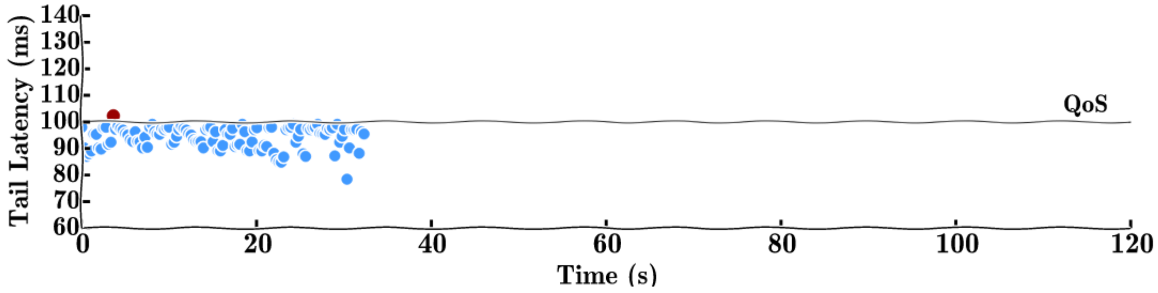


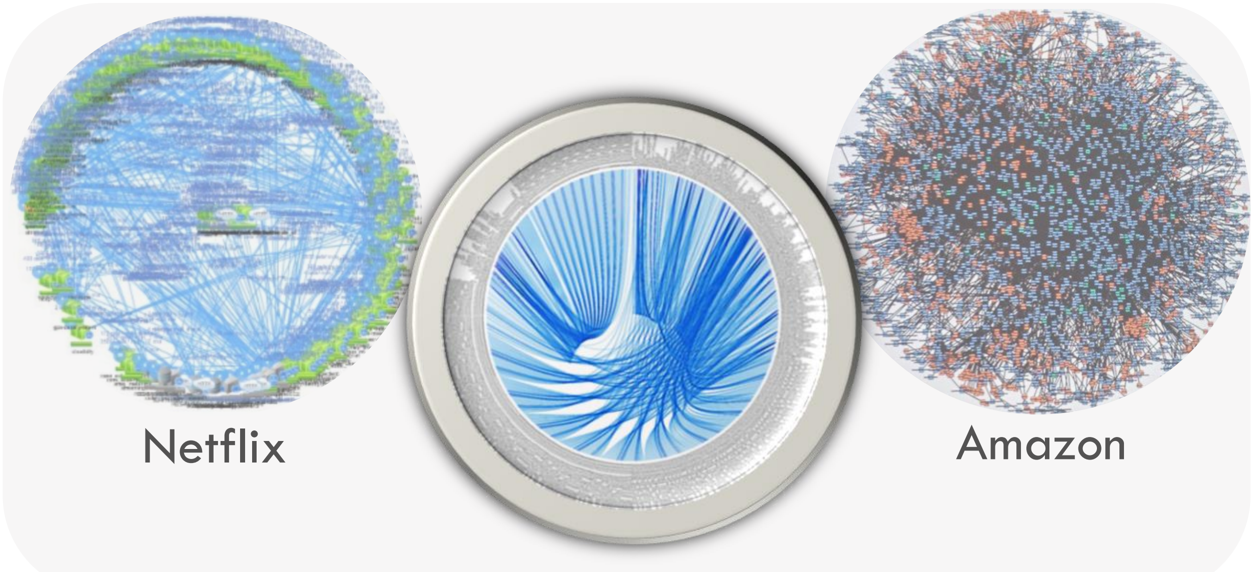


- Dependencies cause cascading QoS violations
- Empirical performance debugging → too slow, bottlenecks propagate
- Long recovery times for performance

Performance Debugging Challenges



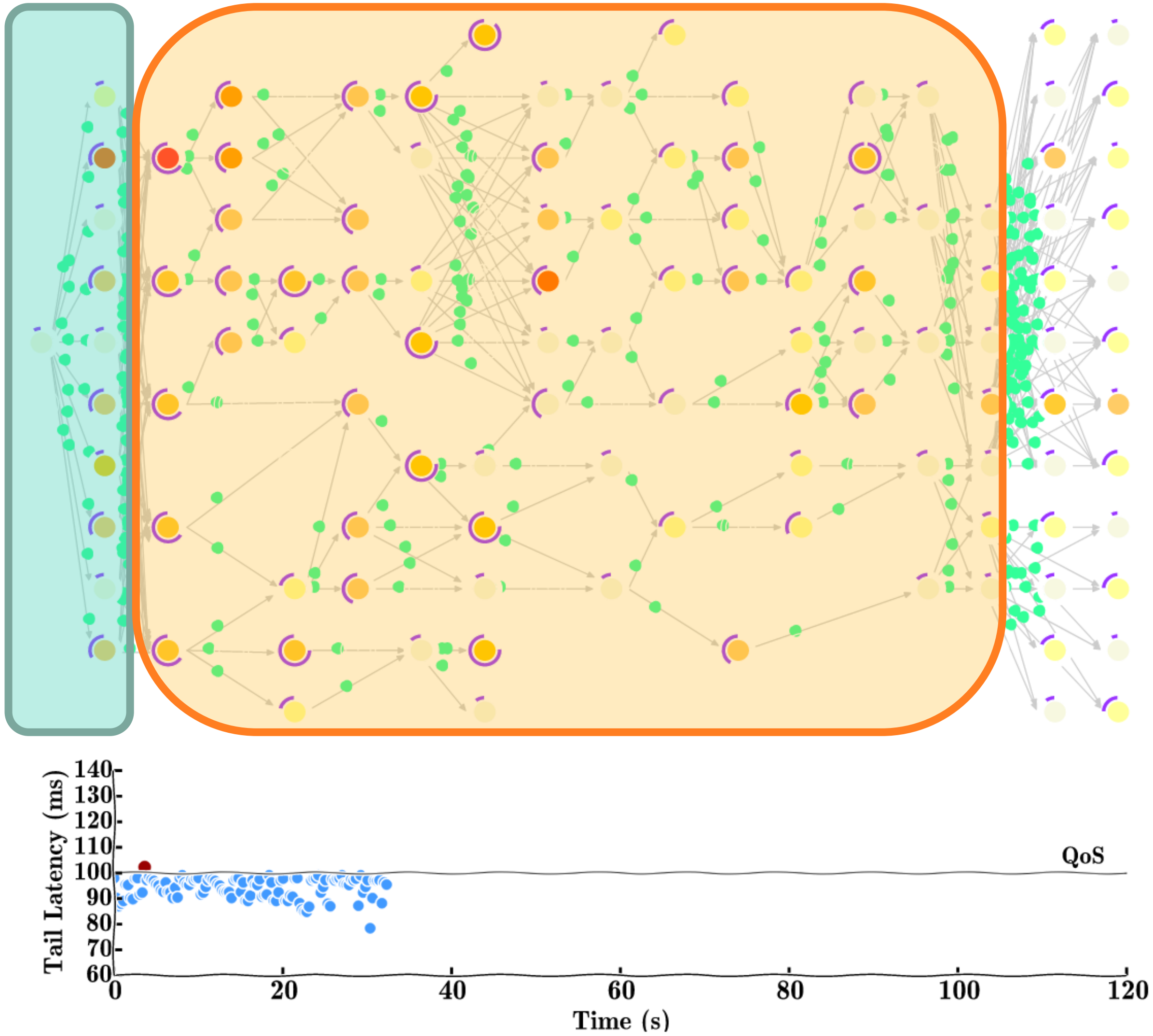




Social Network

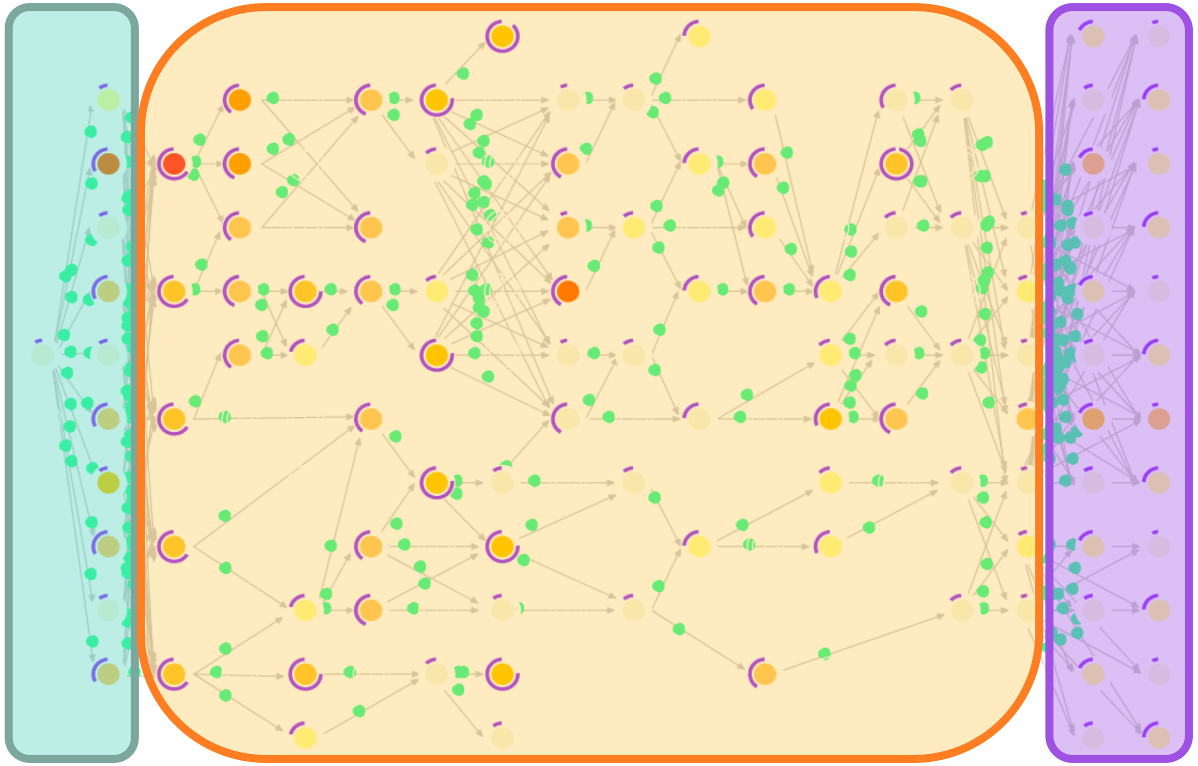
- Dependencies cause cascading QoS violations
- Empirical performance debugging → too slow, bottlenecks propagate
- Long recovery times for performance

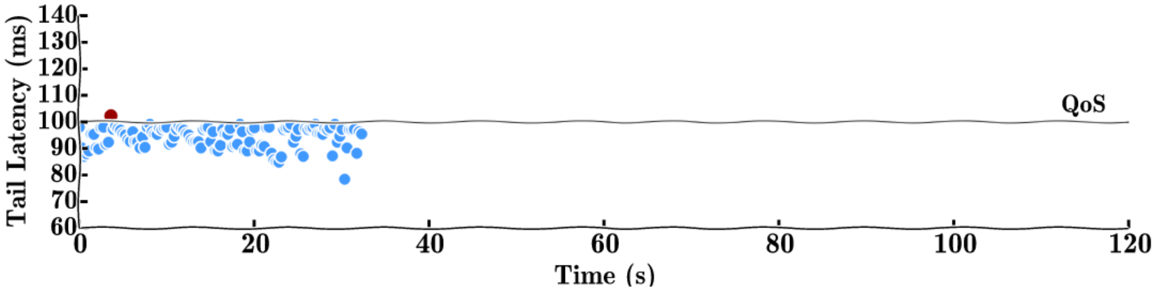
Performance Debugging Challenges



- Dependencies cause cascading QoS violations
- Empirical performance debugging → too slow, bottlenecks propagate
- Long recovery times for performance

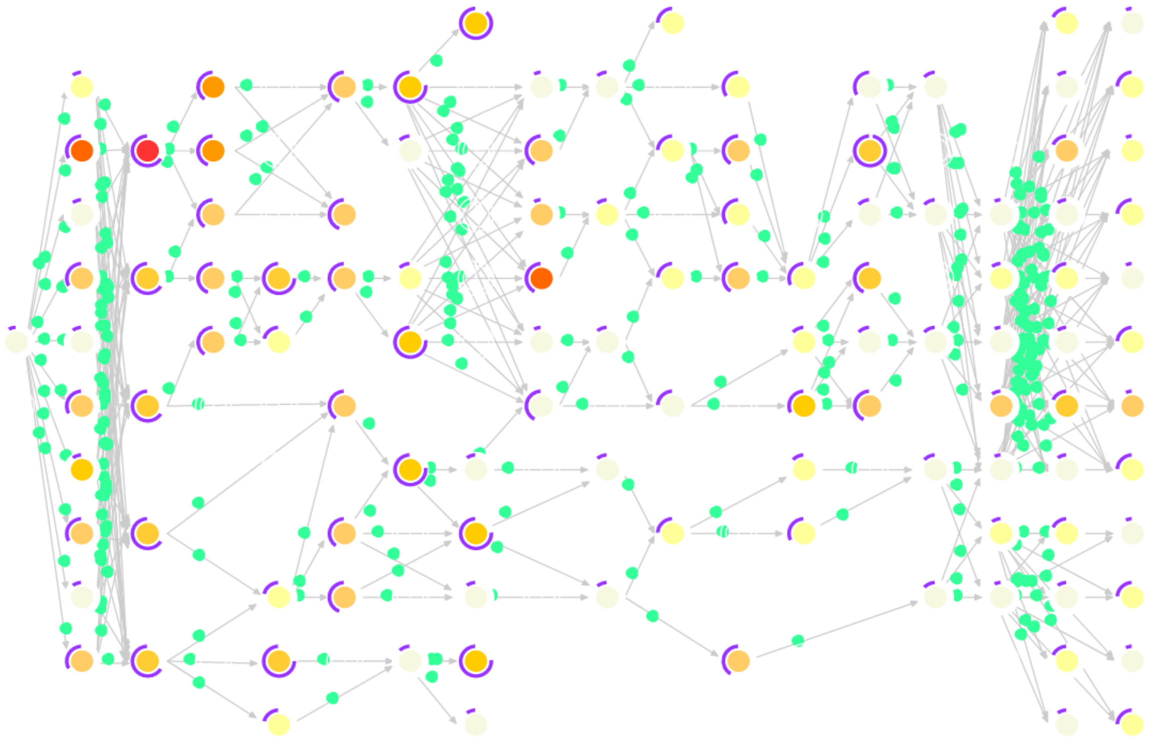
Performance Debugging Challenges

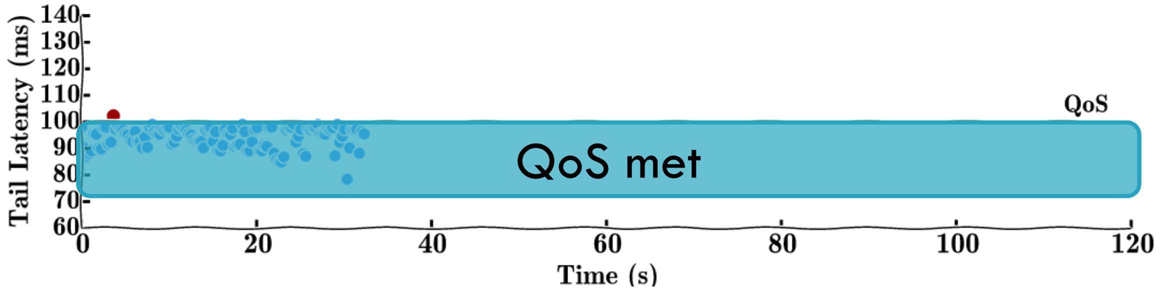




- Dependencies cause cascading QoS violations
- Empirical performance debugging → too slow, bottlenecks propagate
- Long recovery times for performance

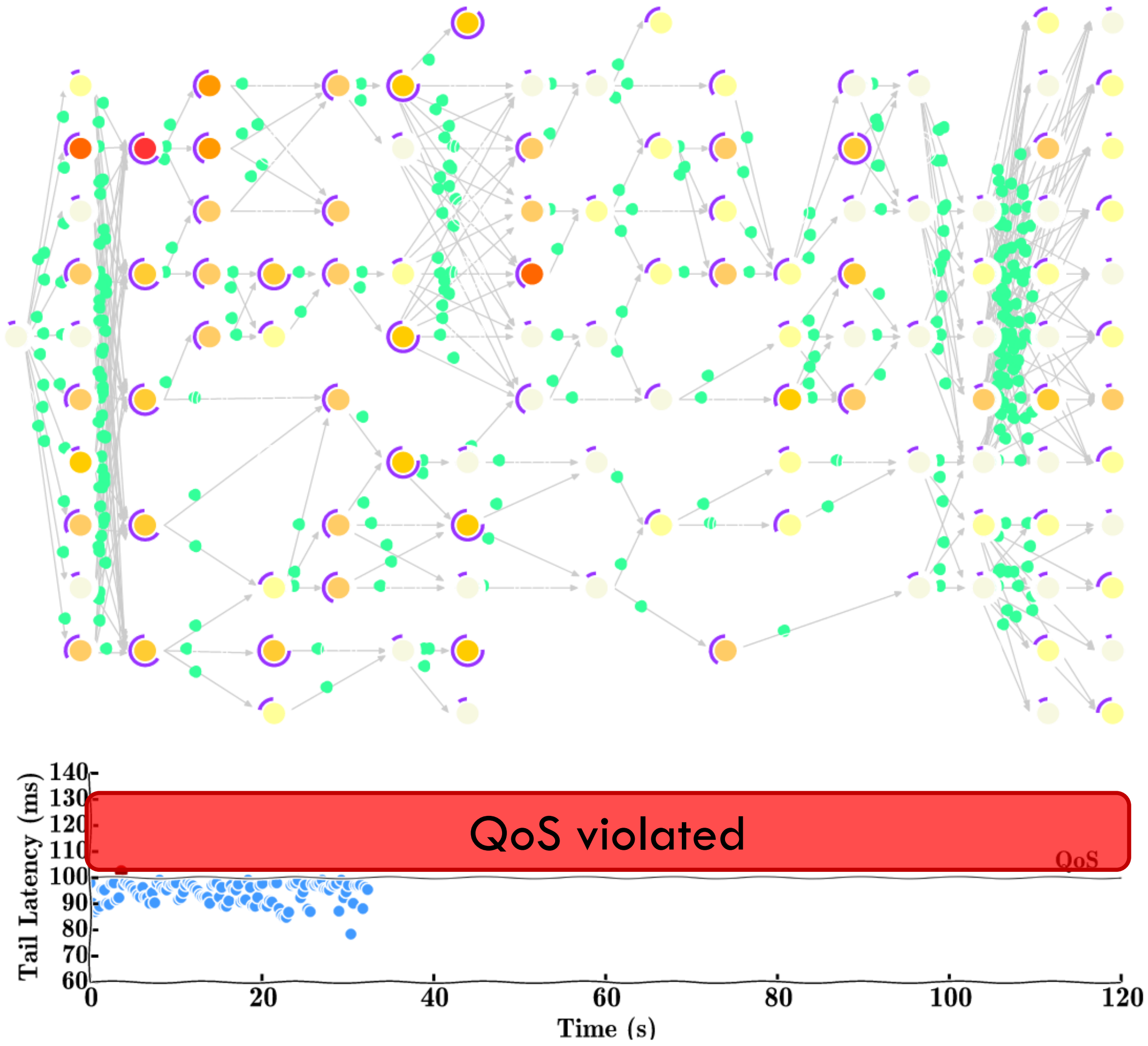
Performance Debugging Challenges





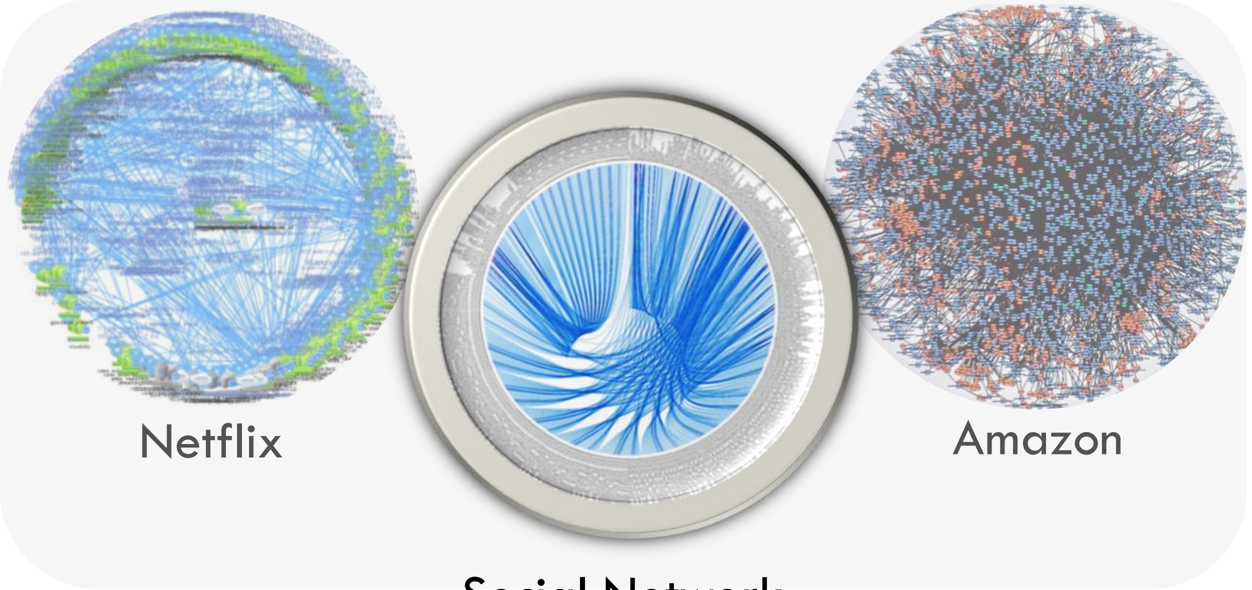
- Dependencies cause cascading QoS violations
- Empirical performance debugging → too slow, bottlenecks propagate
- Long recovery times for performance

Performance Debugging Challenges



- Dependencies cause cascading QoS violations
- Empirical performance debugging → too slow, bottlenecks propagate
- Long recovery times for performance

Performance Debugging Challenges

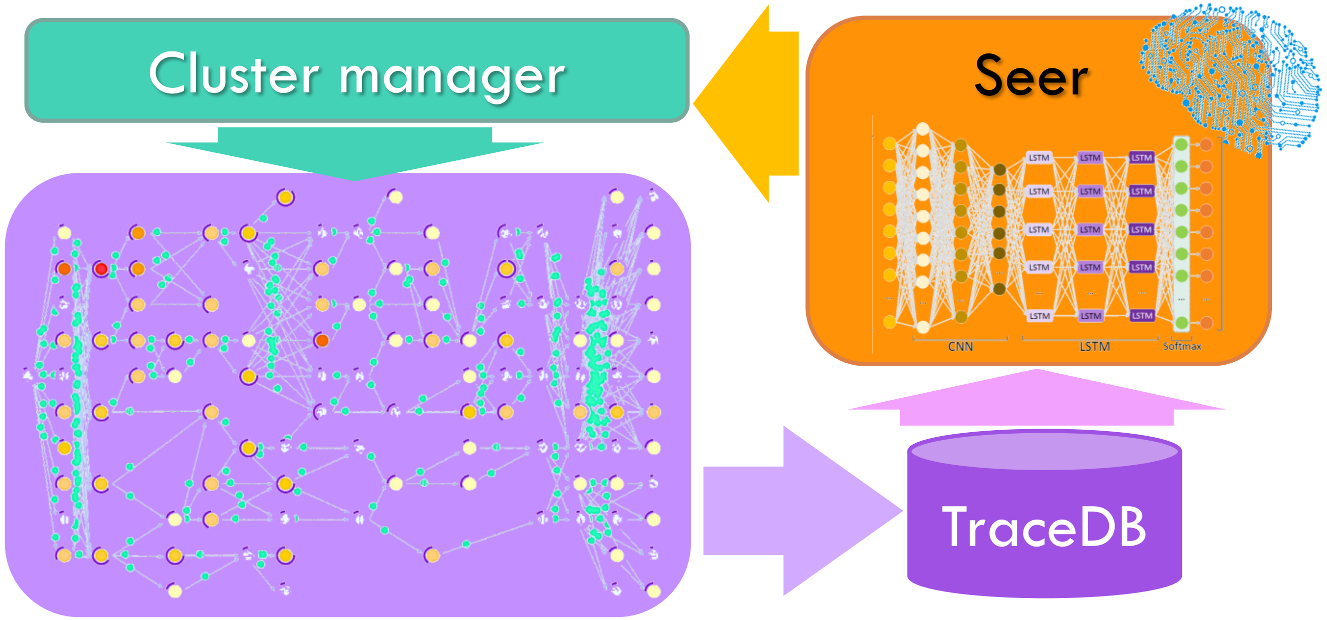


Social Network

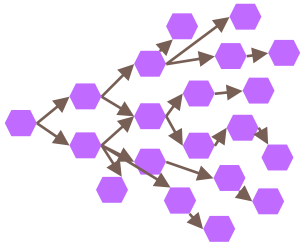
- Dependencies cause cascading QoS violations
- Empirical performance debugging → too slow, bottlenecks propagate
- Long recovery times for performance

Demo: http://www.csl.cornell.edu/~delimitrou/2019.asplos.seer.demo_motivation.mp4

Seer: Proactive Performance Debugging

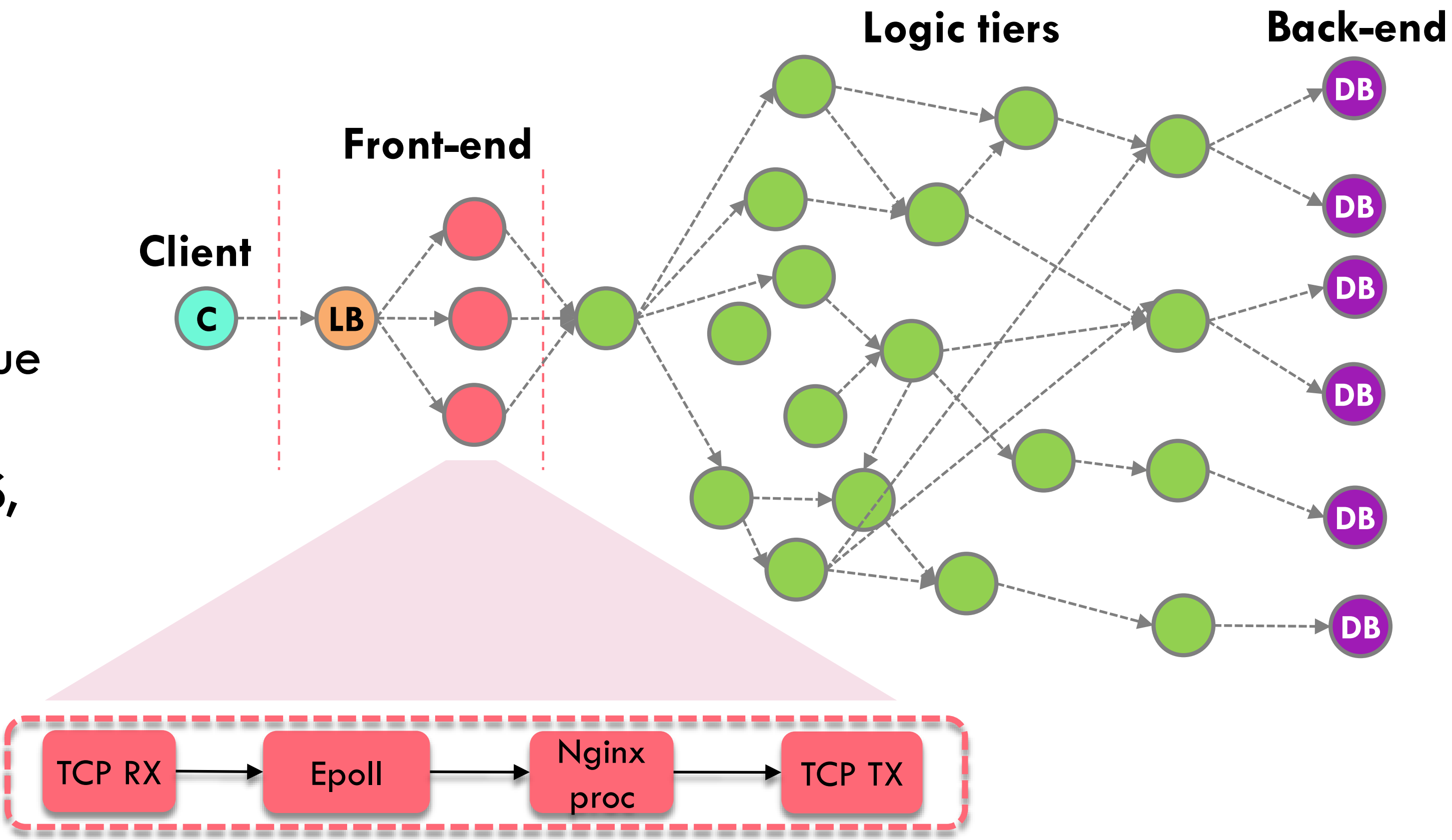


- Use ML to identify the culprit (root cause) of an *upcoming* QoS violation
  - Leverage the massive amount of distributed traces collected over time
  - Use targeted per-server hardware probes to determine the cause of the QoS violation
- Inform cluster manager to take proactive action & prevent QoS violation
  - Need to predict 100s of msec – a few sec in the future



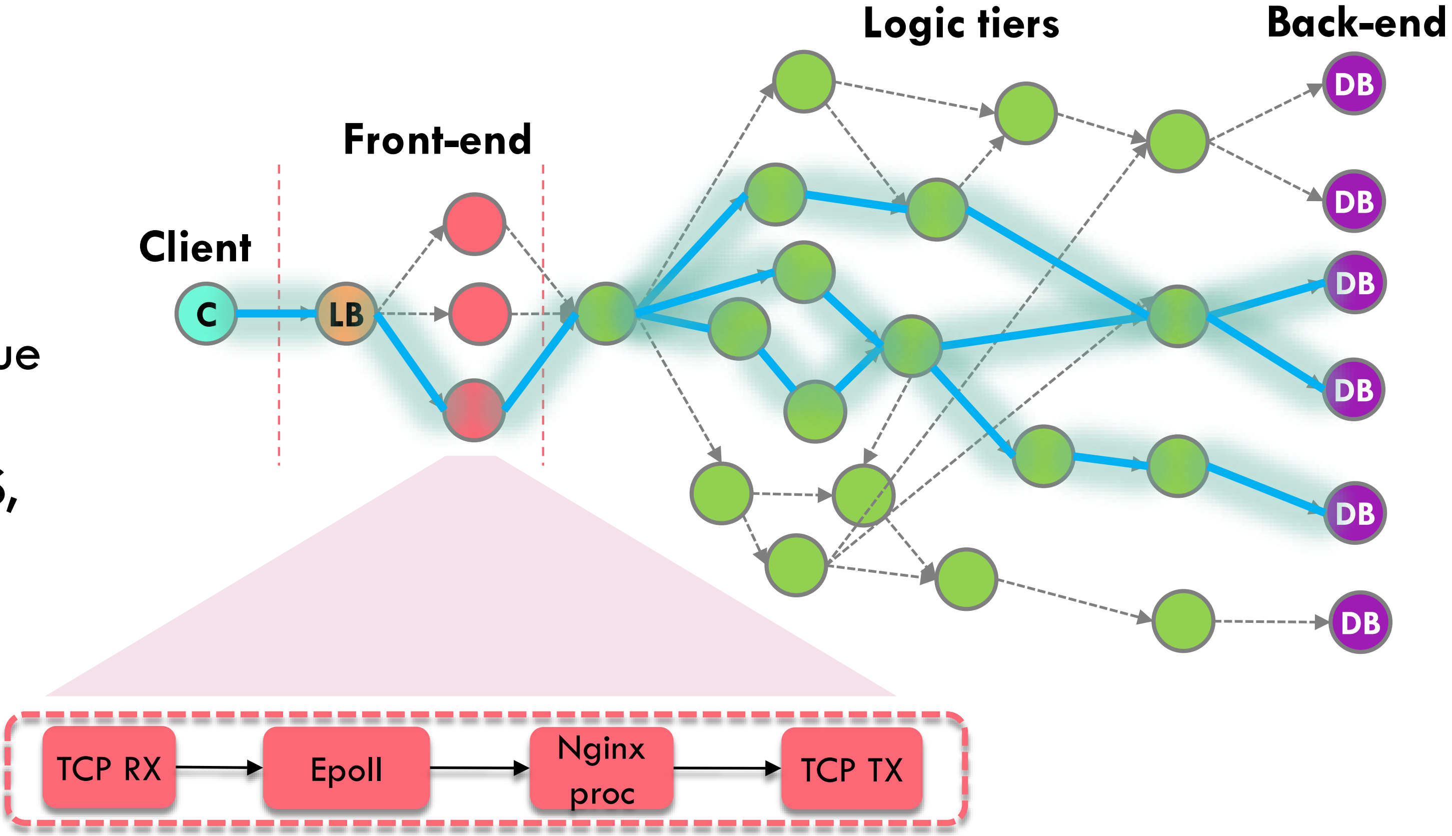
Instrumentation & Tracing

- Two-level tracing
  - Distributed RPC-level tracing
    - Similar to Dapper, Zipkin
    - Per-microservice latencies
    - Inter- and intra-microservice queue lengths
    - Tracing overhead: <0.1% in QPS, <0.2% in 99th %ile latency
  - Per-node hardware monitoring
    - Targeted on nodes with problematic microservices
    - Perf counters & contentious microbenchmarks



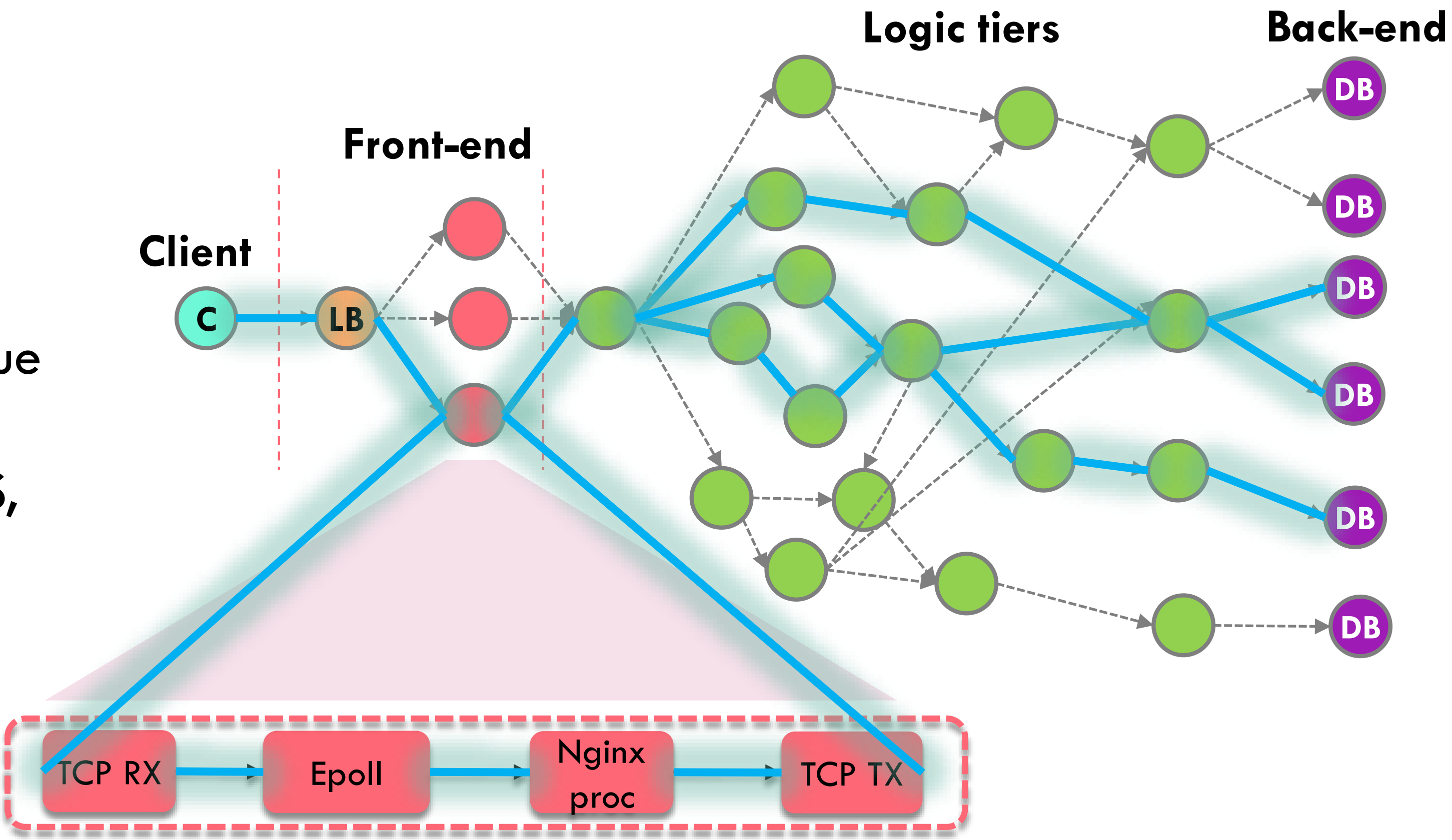
Instrumentation & Tracing

- Two-level tracing
  - Distributed RPC-level tracing
    - Similar to Dapper, Zipkin
    - Per-microservice latencies
    - Inter- and intra-microservice queue lengths
    - Tracing overhead: <0.1% in QPS, <0.2% in 99th %ile latency
  - Per-node hardware monitoring
    - Targeted on nodes with problematic microservices
    - Perf counters & contentious microbenchmarks

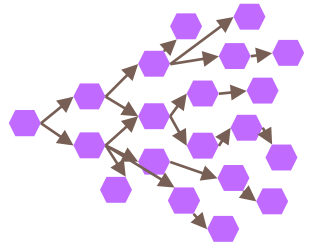


Instrumentation & Tracing

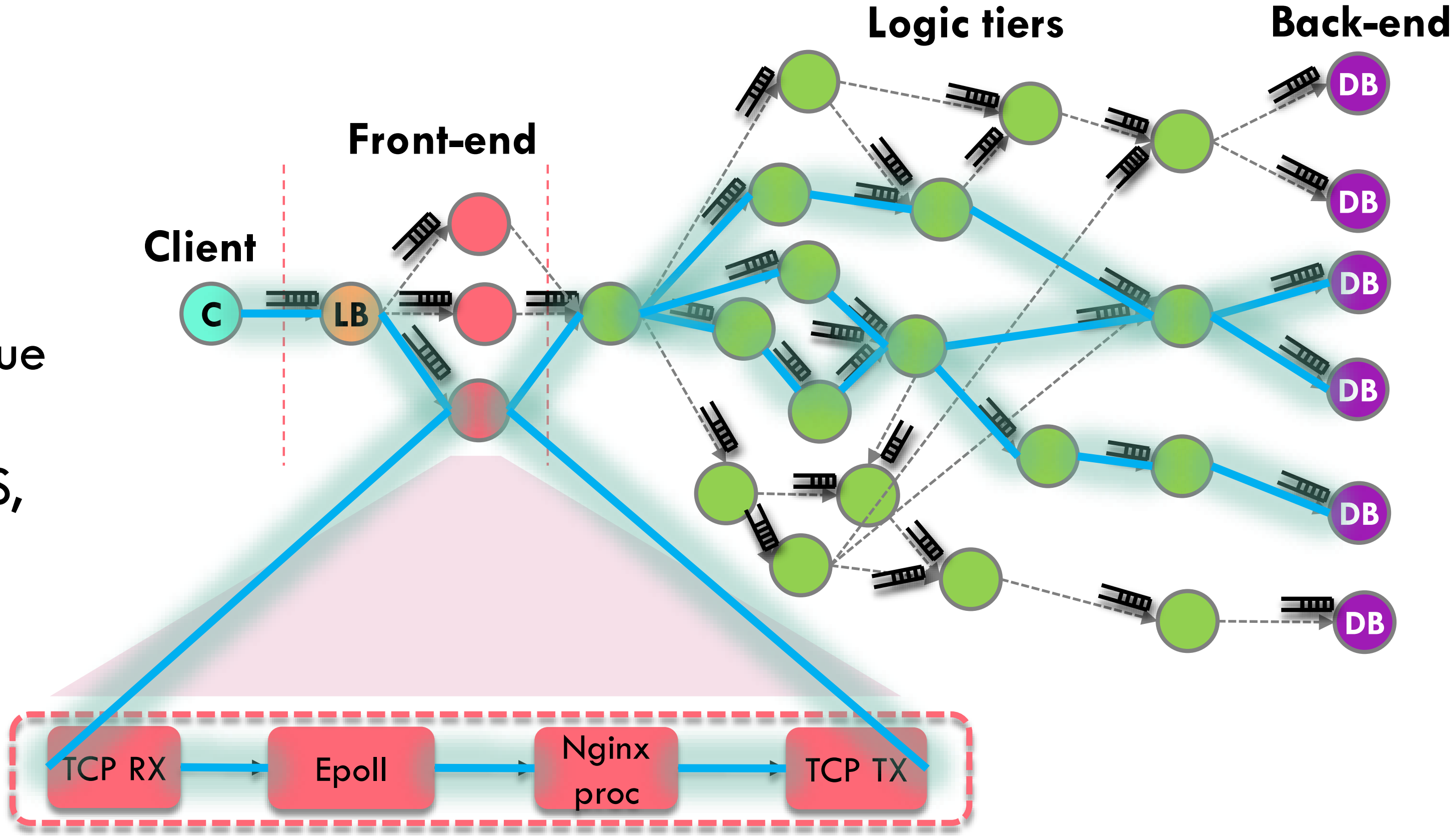
- Two-level tracing
  - Distributed RPC-level tracing
    - Similar to Dapper, Zipkin
    - Per-microservice latencies
    - Inter- and intra-microservice queue lengths
    - Tracing overhead: <0.1% in QPS, <0.2% in 99th %ile latency
  - Per-node hardware monitoring
    - Targeted on nodes with problematic microservices
    - Perf counters & contentious microbenchmarks



Instrumentation & Tracing

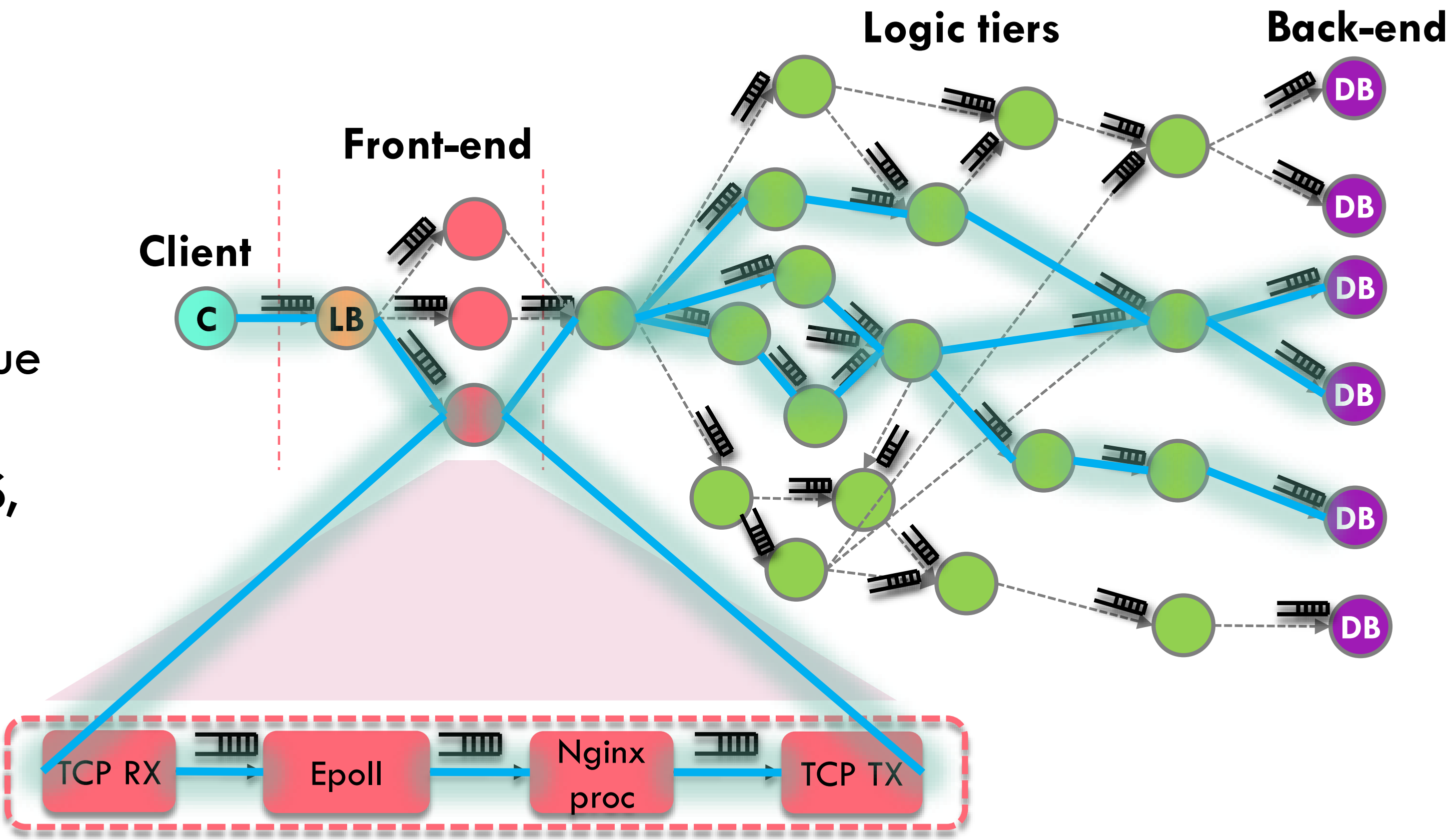


- Two-level tracing
  - Distributed RPC-level tracing
    - Similar to Dapper, Zipkin
    - Per-microservice latencies
    - Inter- and intra-microservice queue lengths
    - Tracing overhead: <0.1% in QPS, <0.2% in 99th %ile latency
  - Per-node hardware monitoring
    - Targeted on nodes with problematic microservices
    - Perf counters & contentious microbenchmarks

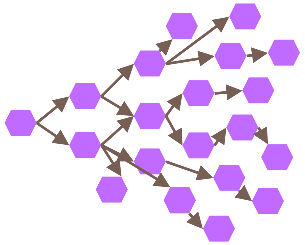


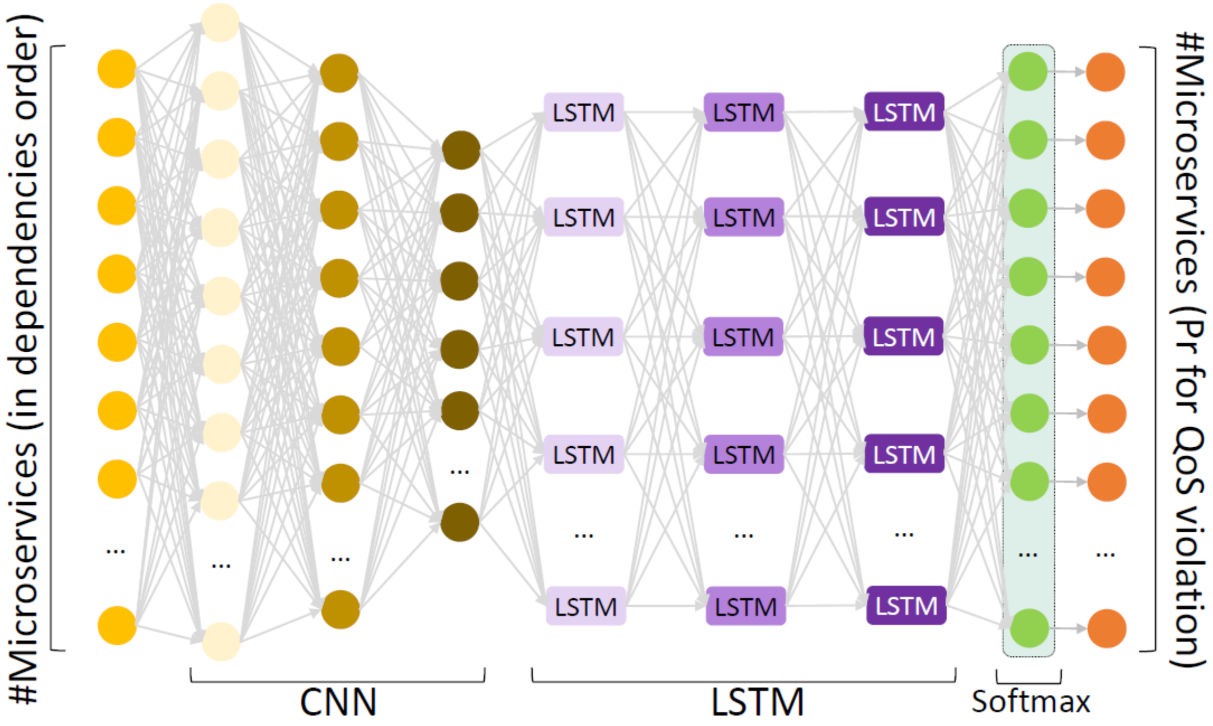
Instrumentation & Tracing

- Two-level tracing
  - Distributed RPC-level tracing
    - Similar to Dapper, Zipkin
    - Per-microservice latencies
    - Inter- and intra-microservice queue lengths
    - Tracing overhead: <0.1% in QPS, <0.2% in 99th %ile latency
  - Per-node hardware monitoring
    - Targeted on nodes with problematic microservices
    - Perf counters & contentious microbenchmarks



DL for Cloud Performance Debugging



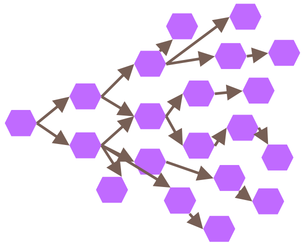


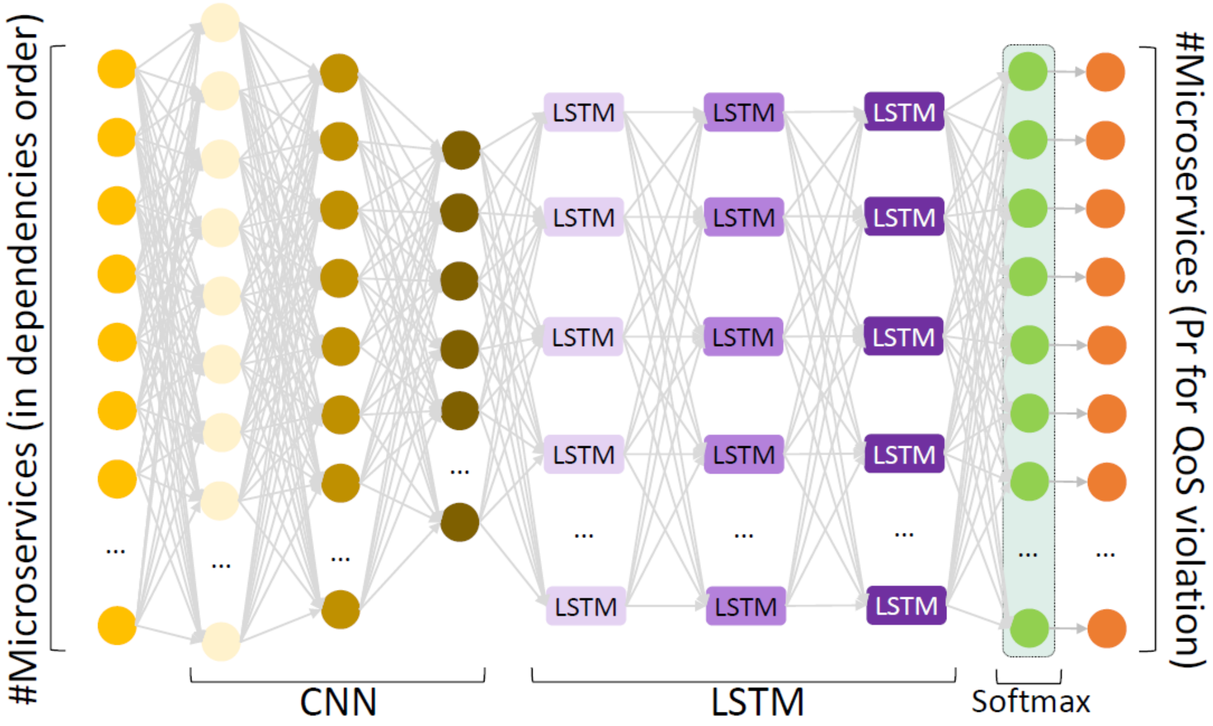
Output signal

Probability that a microservice will initiate a QoS violation in the near future

- Why?
  - Architecture-agnostic
  - Adjusts to changes over time
  - High accuracy, good scalability & fast inference (within window of opportunity)

DL for Cloud Performance Debugging





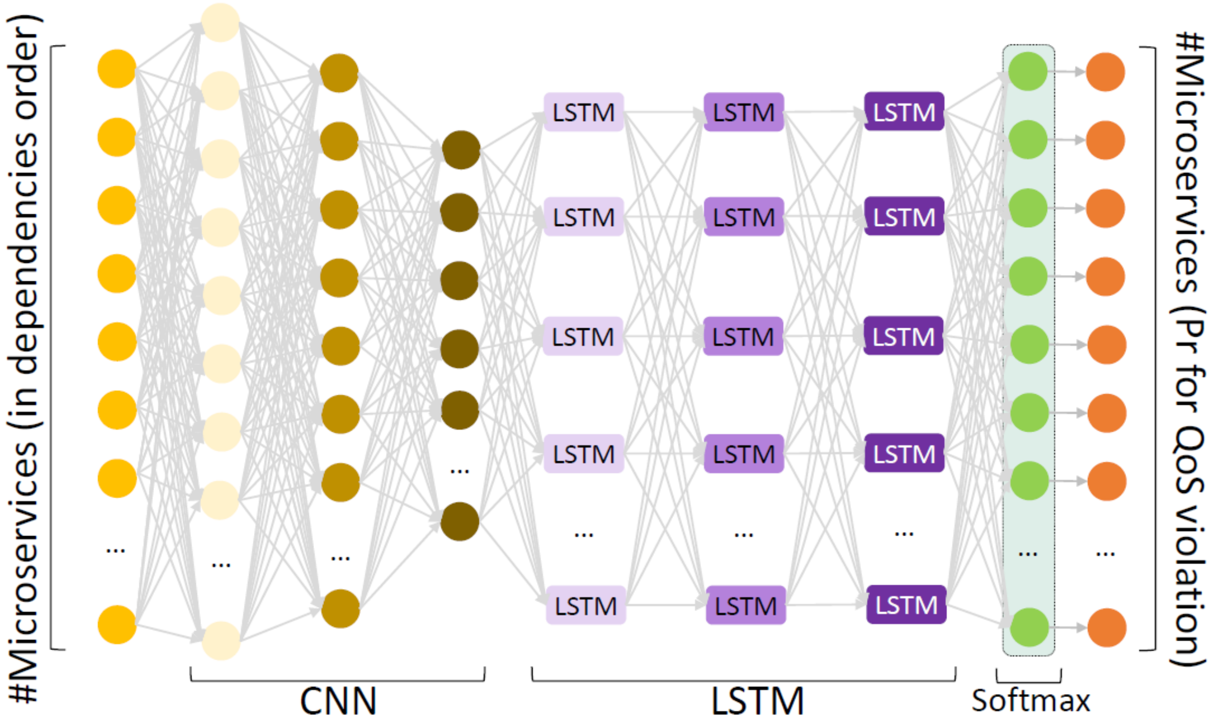
Output signal

Probability that a microservice will initiate a QoS violation in the near future

DL for Cloud Performance Debugging

Input signal

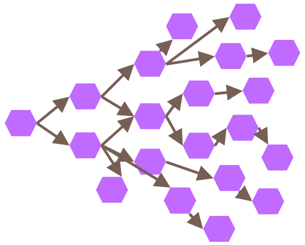
- Container utilization



Output signal

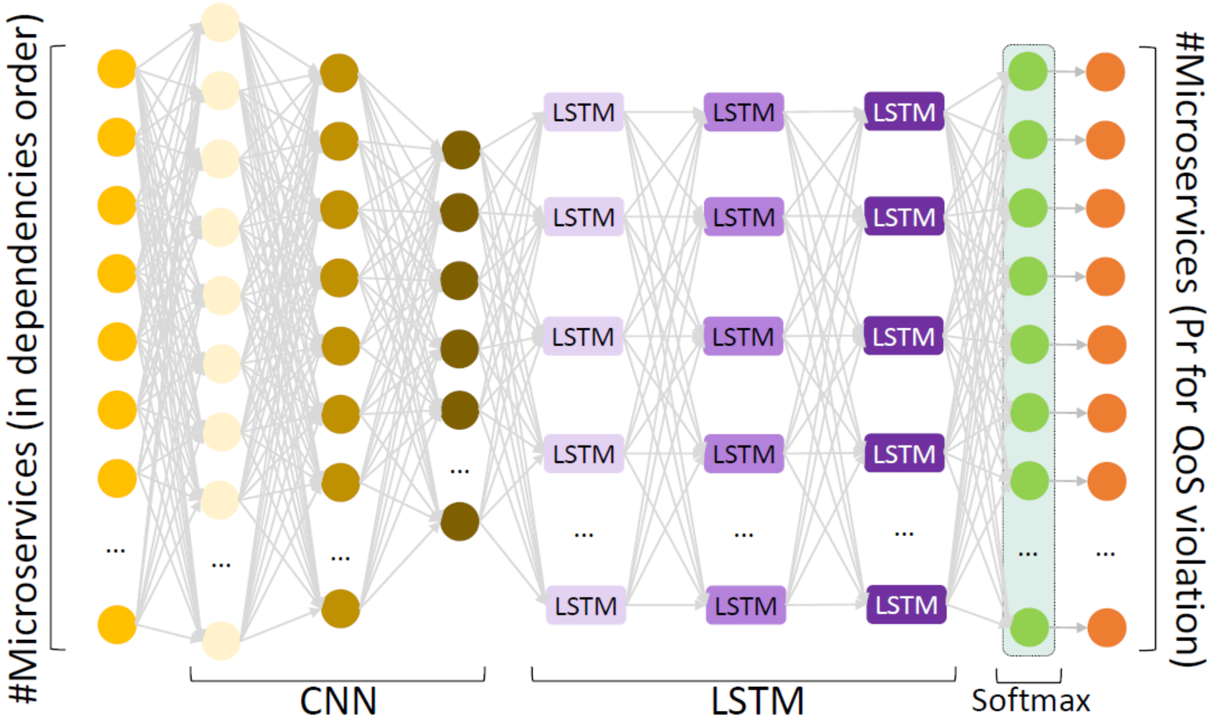
Probability that a microservice will initiate a QoS violation in the near future

DL for Cloud Performance Debugging



Input signal

- Container utilization
- Latency



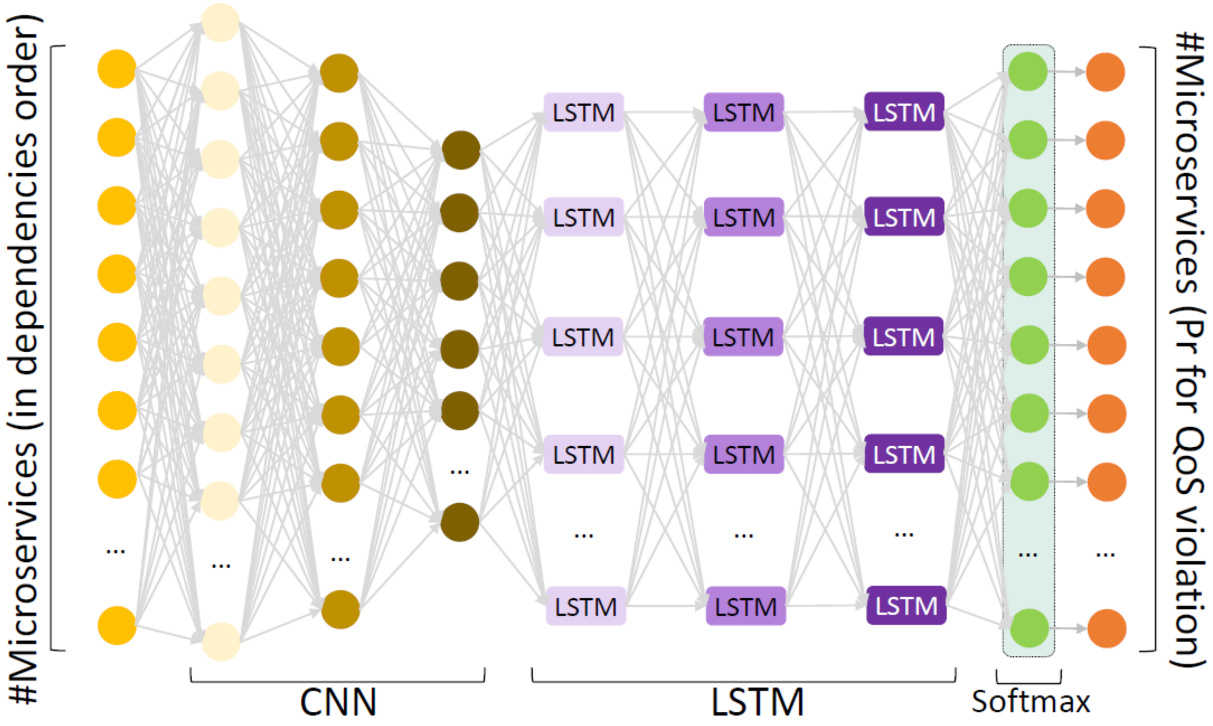
Output signal

Probability that a microservice will initiate a QoS violation in the near future

DL for Cloud Performance Debugging

Input signal

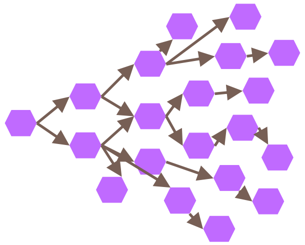
- Container utilization
- Latency
- Queue length



Output signal

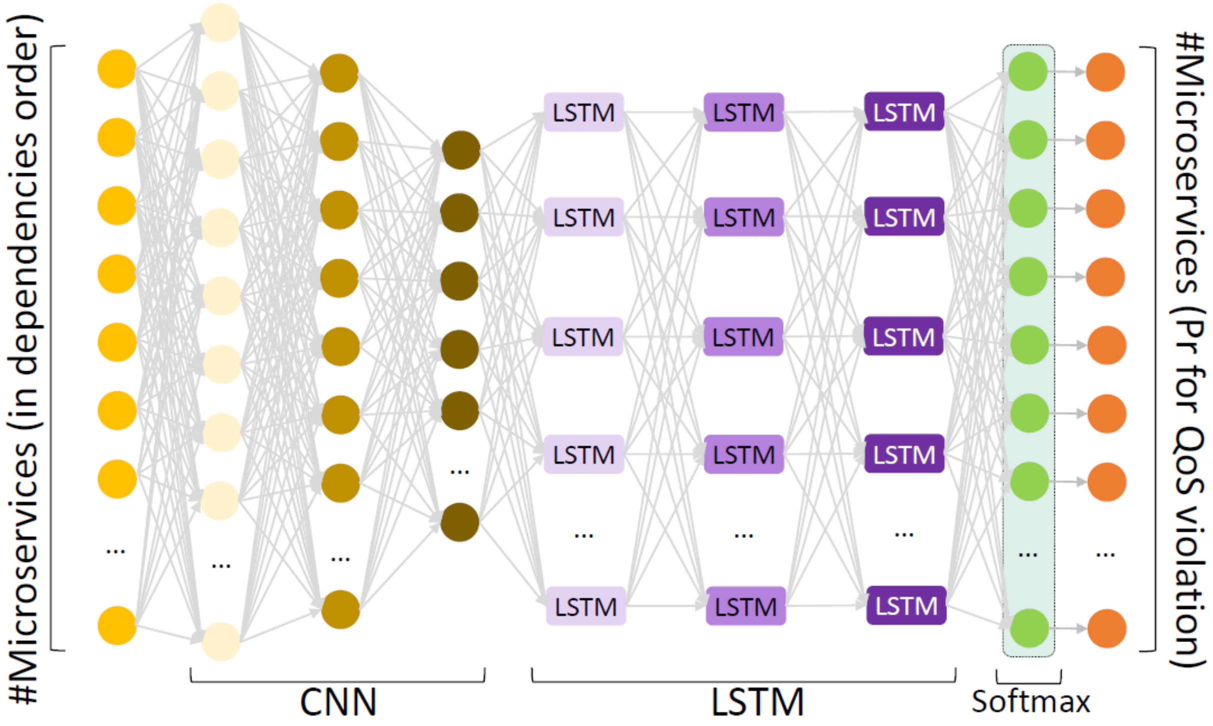
Probability that a microservice will initiate a QoS violation in the near future

DL for Cloud Performance Debugging



Input signal

- Container utilization
- Latency
- Queue length



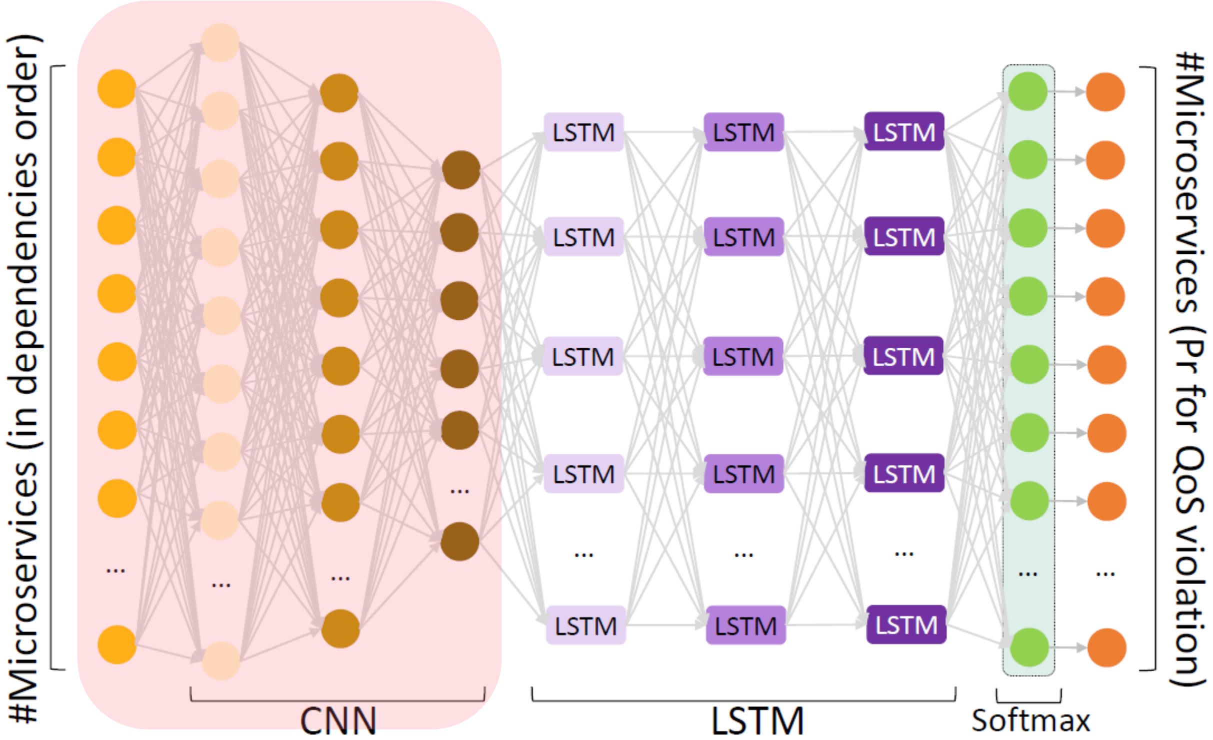
Output signal

Probability that a microservice will initiate a QoS violation in the near future

DL for Cloud Performance Debugging

Input signal

- Container utilization
- Latency
- Queue length



Dimensionality reduction

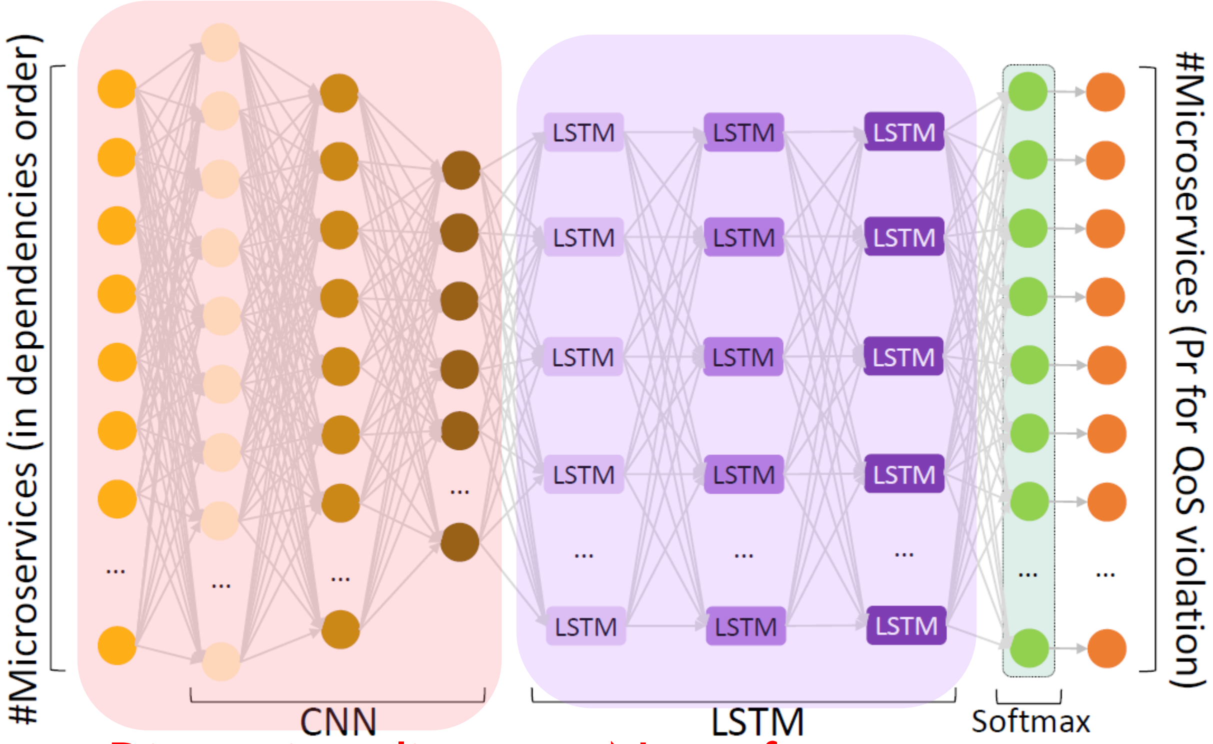
Output signal

Probability that a microservice will initiate a QoS violation in the near future

DL for Cloud Performance Debugging

Input signal

- Container utilization
- Latency
- Queue length



Dimensionality reduction

Near-future prediction

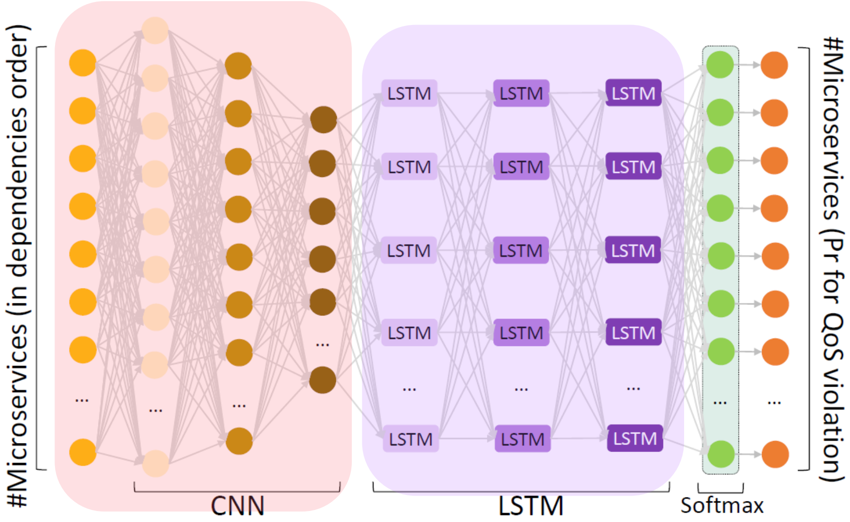
Output signal

Probability that a microservice will initiate a QoS violation in the near future

DL for Cloud Performance Debugging

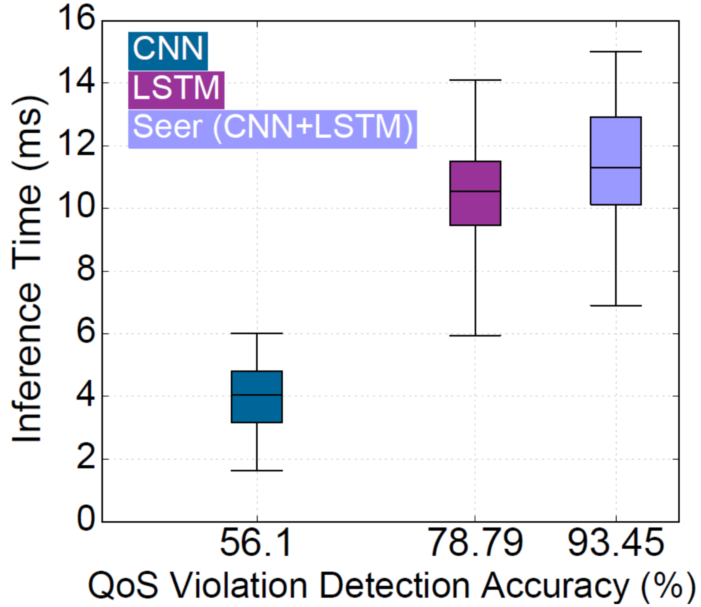
Input signal

- Queue length



Output signal

Probability that a microservice will initiate a QoS violation in the near future



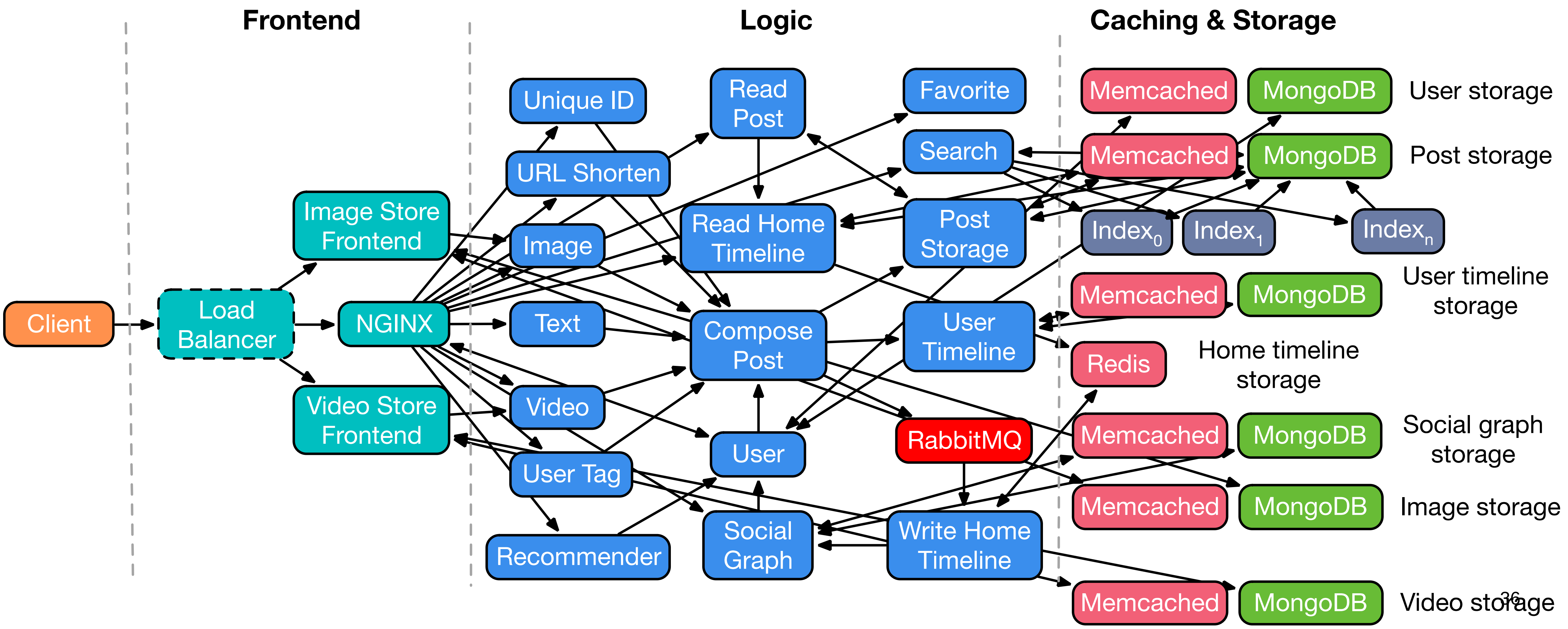
- DNN Configuration
  - CNN: Fast, but cannot effectively predict future
  - LSTM: Higher accuracy, but affected by noisy, non-critical microservices
  - Hybrid network: Highest accuracy, without significantly higher overhead

Methodology

- Training once: slow (hours - days)
  - Across load levels, load distributions, request types
  - Annotated queue traces → inject microbenchmarks to force controlled QoS violations
  - Weight/bias inference with SGD
  - Incremental retraining & dynamically expanding/shrinking in the background
- Inference: continuously streaming traces
- 20-server dedicated heterogeneous cluster
  - Different server configurations
  - 10s of cores, >100GB RAM per server
- 4 end-to-end applications → ~30-40 unique microservices each
  - Social Network, Media Service, E-commerce Site, Banking System

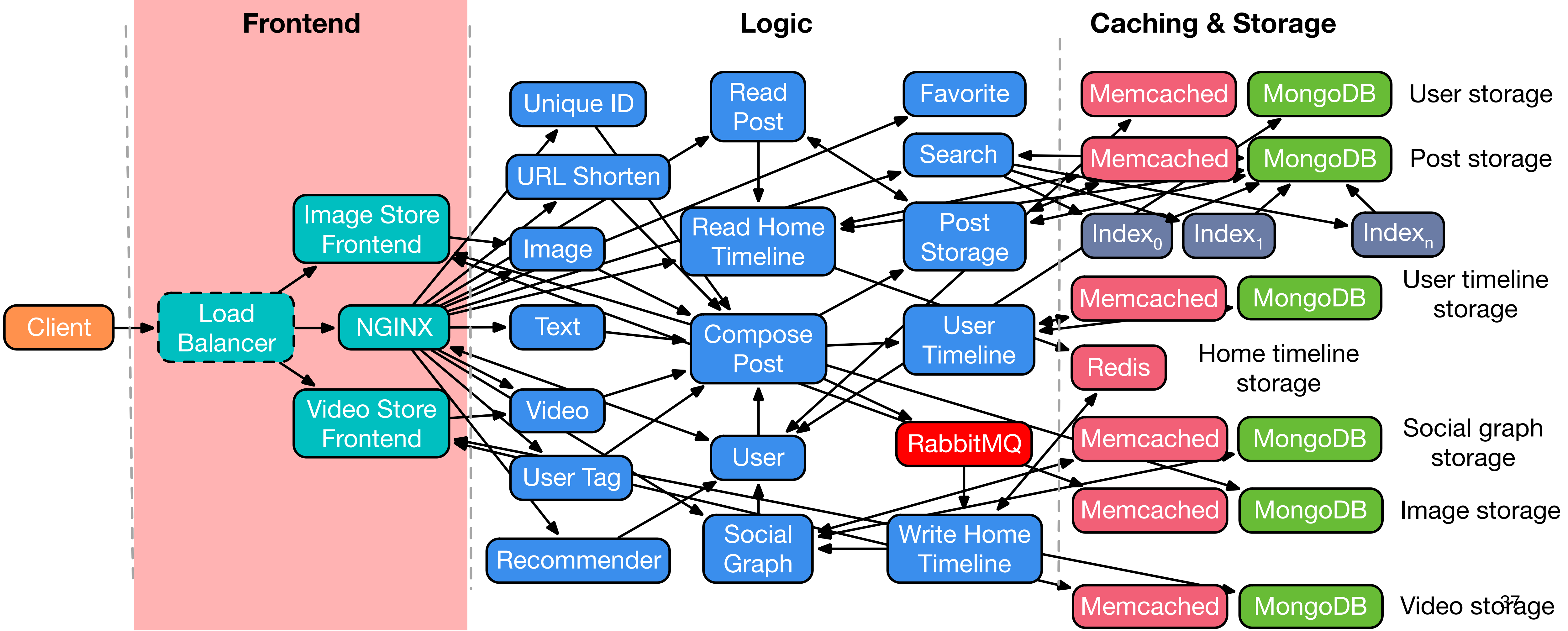
End-to-end Microservices

- Social Network



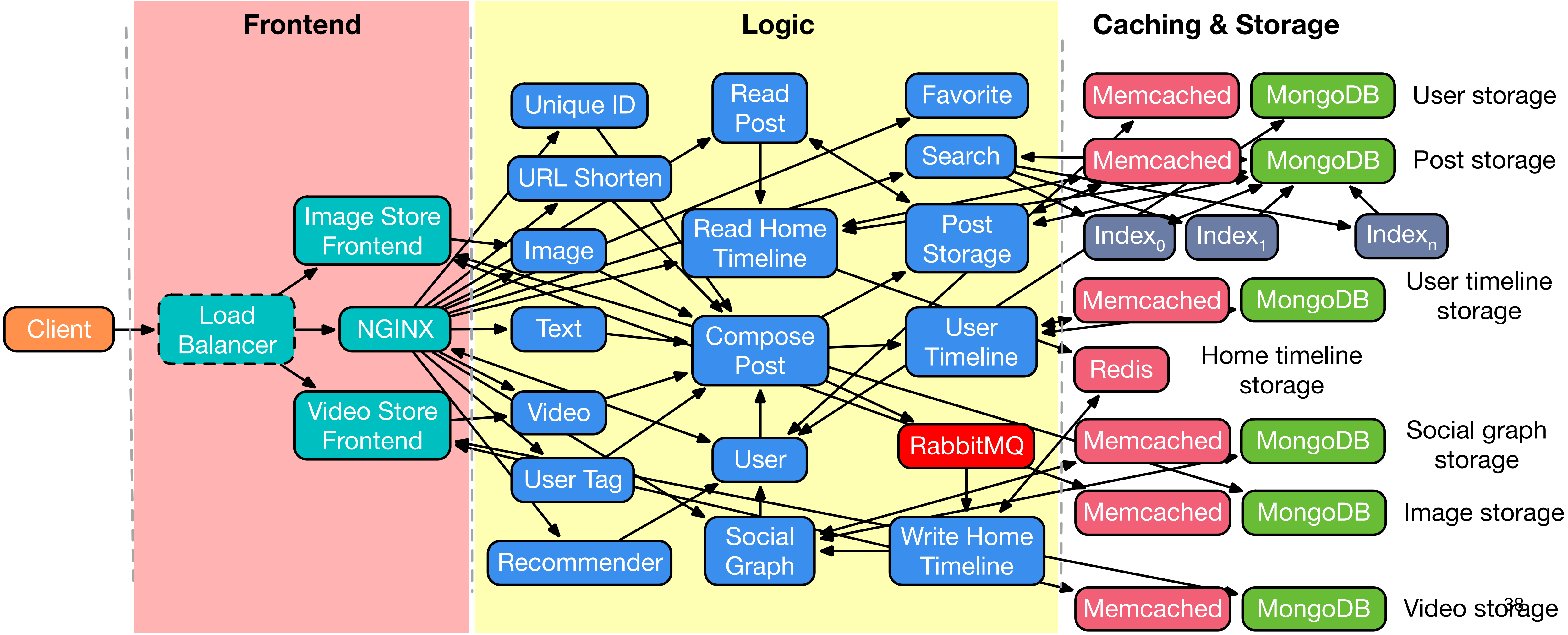
End-to-end Microservices

- Social Network



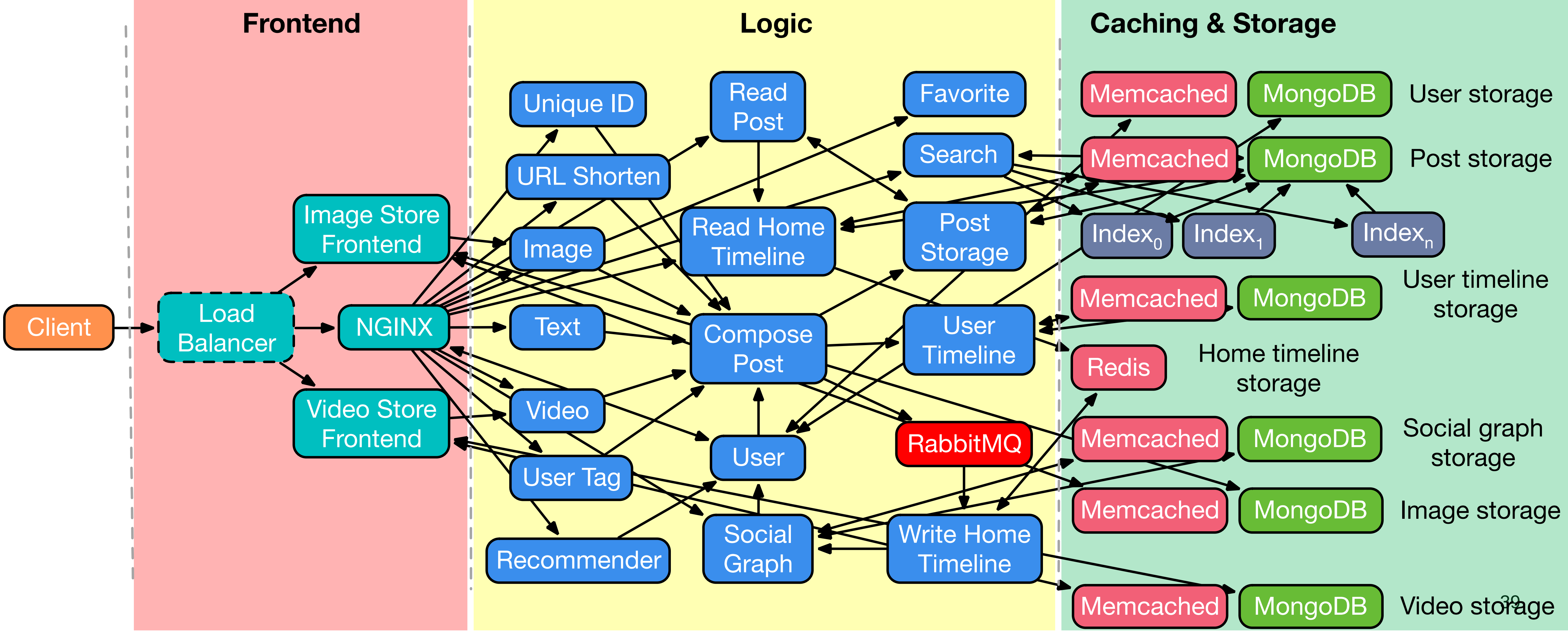
End-to-end Microservices

- Social Network

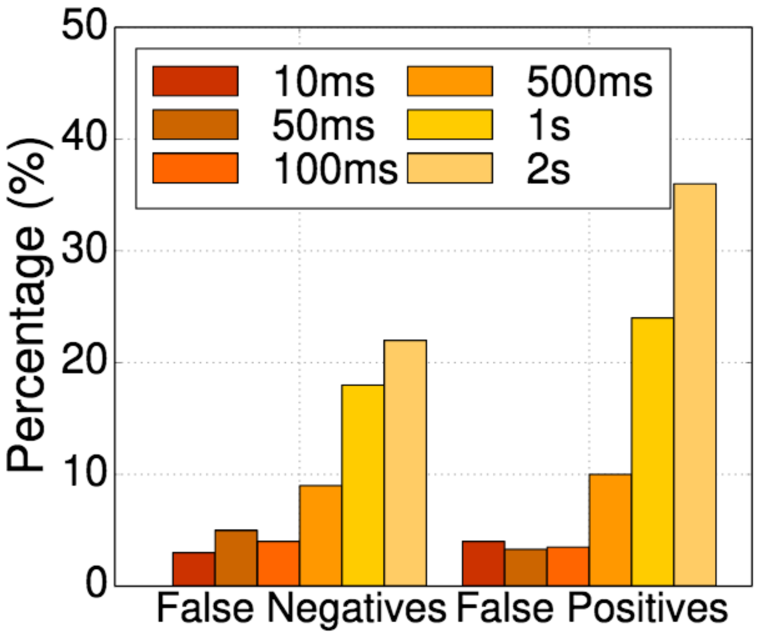


End-to-end Microservices

- Social Network



Validation

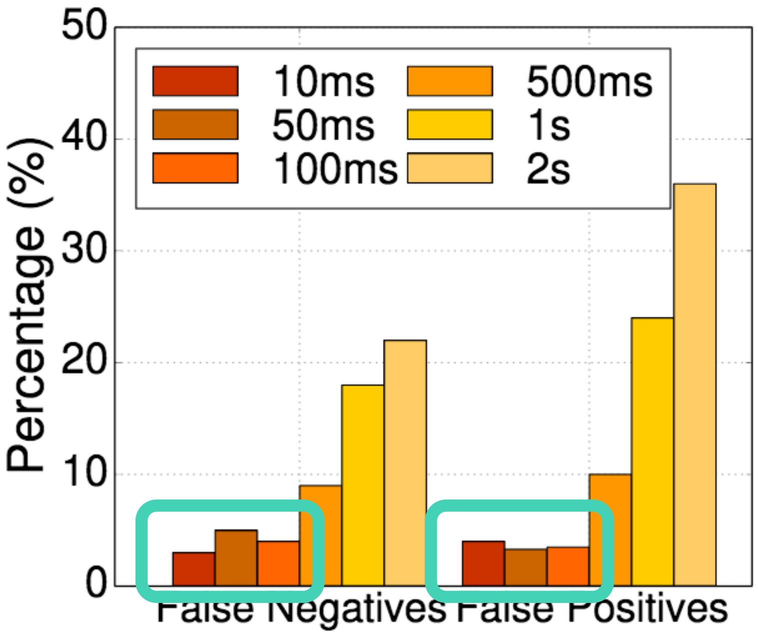


- 50GB input training dataset
  - Accuracy levels off thereafter
- 50ms tracing sampling interval
  - No benefit from finer-grain tracing

91% accuracy in signaling upcoming QoS violations

88% accuracy in attributing QoS violation to correct microservice

Validation

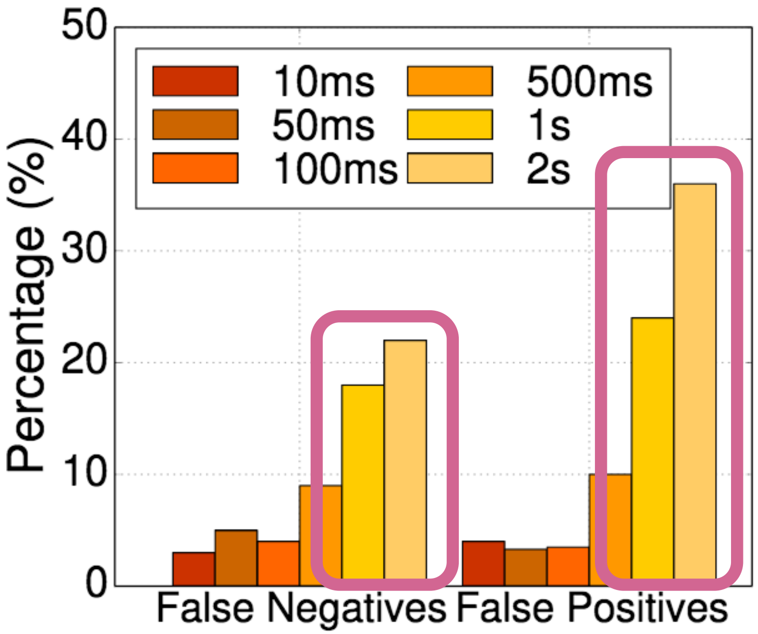


- 50GB input training dataset
  - Accuracy levels off thereafter
- 50ms tracing sampling interval
  - No benefit from finer-grain tracing

91% accuracy in signaling upcoming QoS violations

88% accuracy in attributing QoS violation to correct microservice

Validation

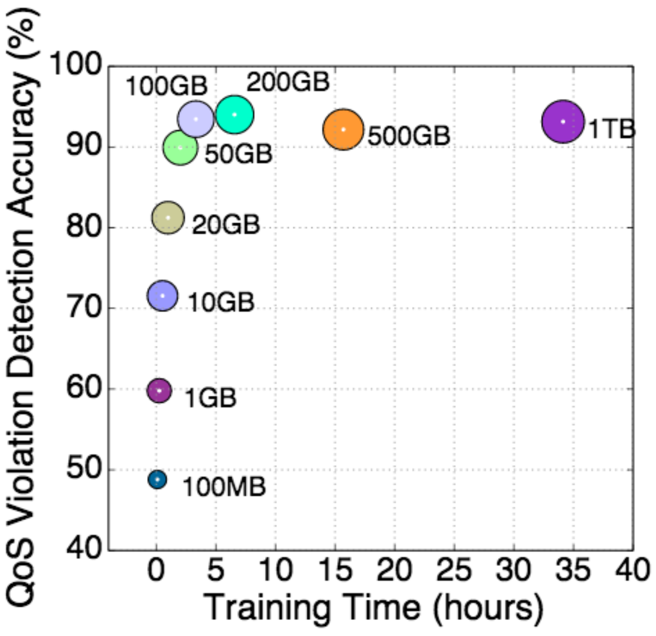


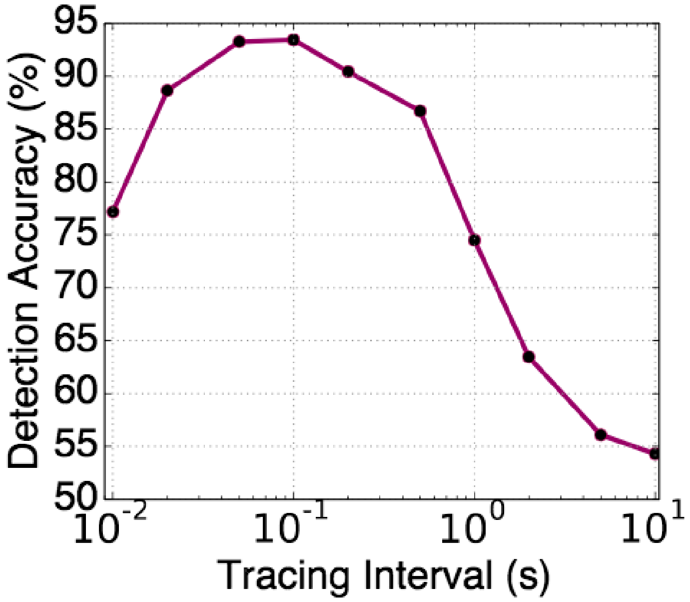
- 50GB input training dataset
  - Accuracy levels off thereafter
- 50ms tracing sampling interval
  - No benefit from finer-grain tracing

91% accuracy in signaling upcoming QoS violations

88% accuracy in attributing QoS violation to correct microservice

Sensitivity Analysis

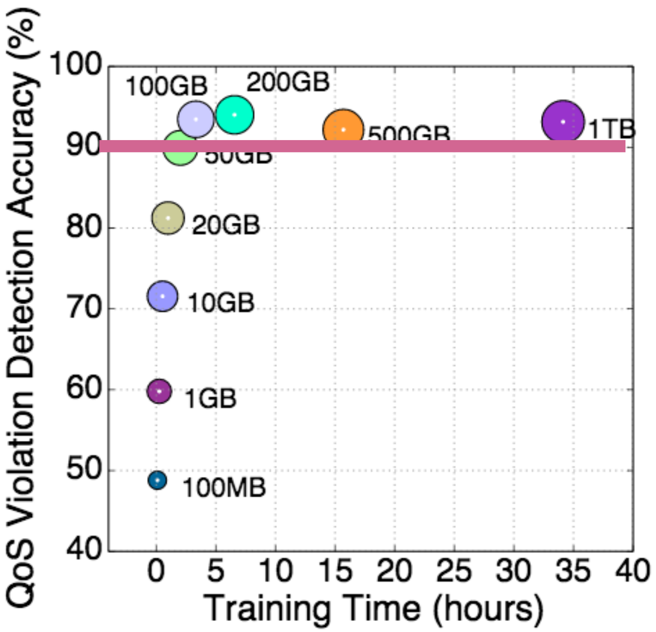


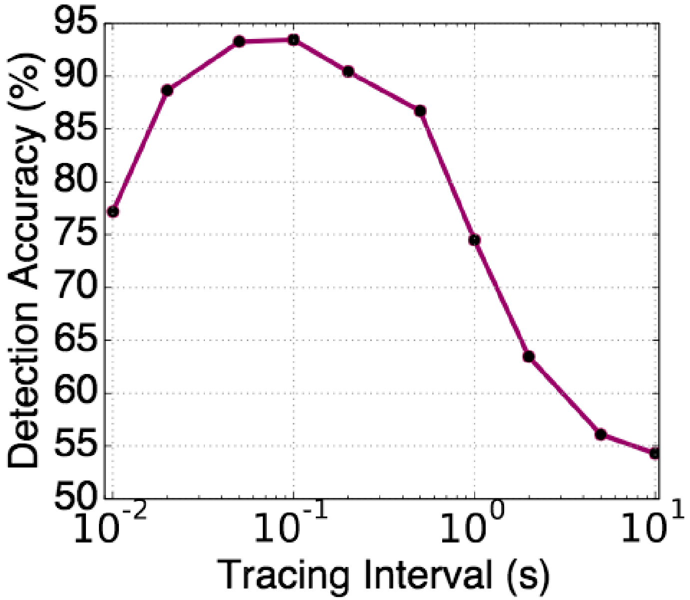


- Large increase in accuracy until ~50GB training set
  - Levels off afterwards
- Large increase in training time after 50GB

- Tracing interval < 500ms → low accuracy
- Tracing interval > 100ms → no further improvement

Sensitivity Analysis

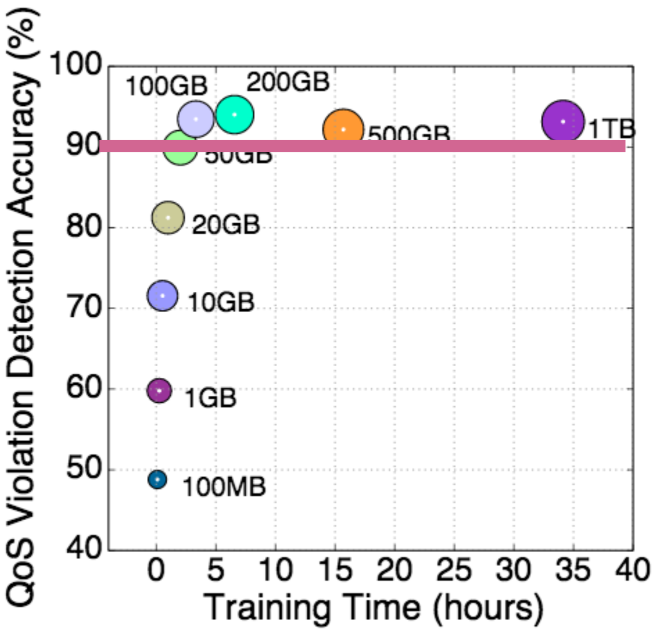


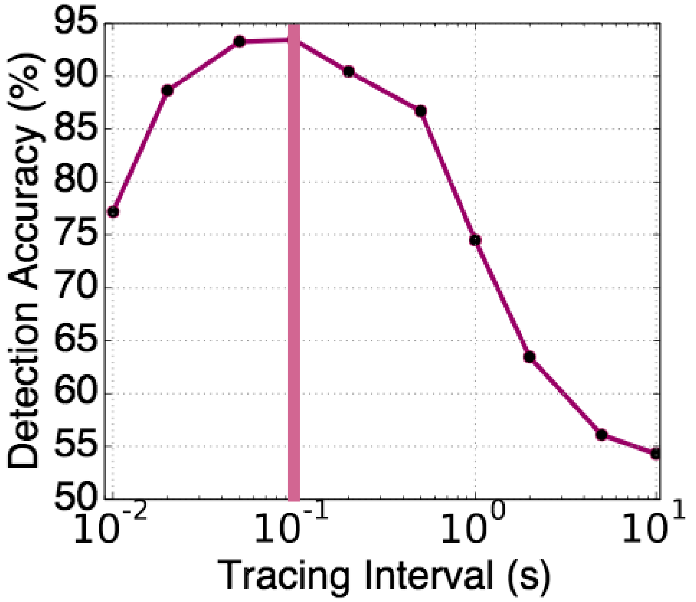


- Large increase in accuracy until ~50GB training set
  - Levels off afterwards
- Large increase in training time after 50GB

- Tracing interval < 500ms → low accuracy
- Tracing interval > 100ms → no further improvement

Sensitivity Analysis

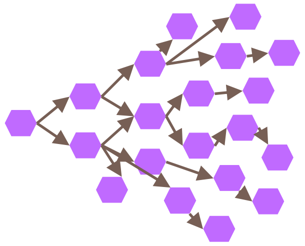




- Large increase in accuracy until ~50GB training set
  - Levels off afterwards
- Large increase in training time after 50GB

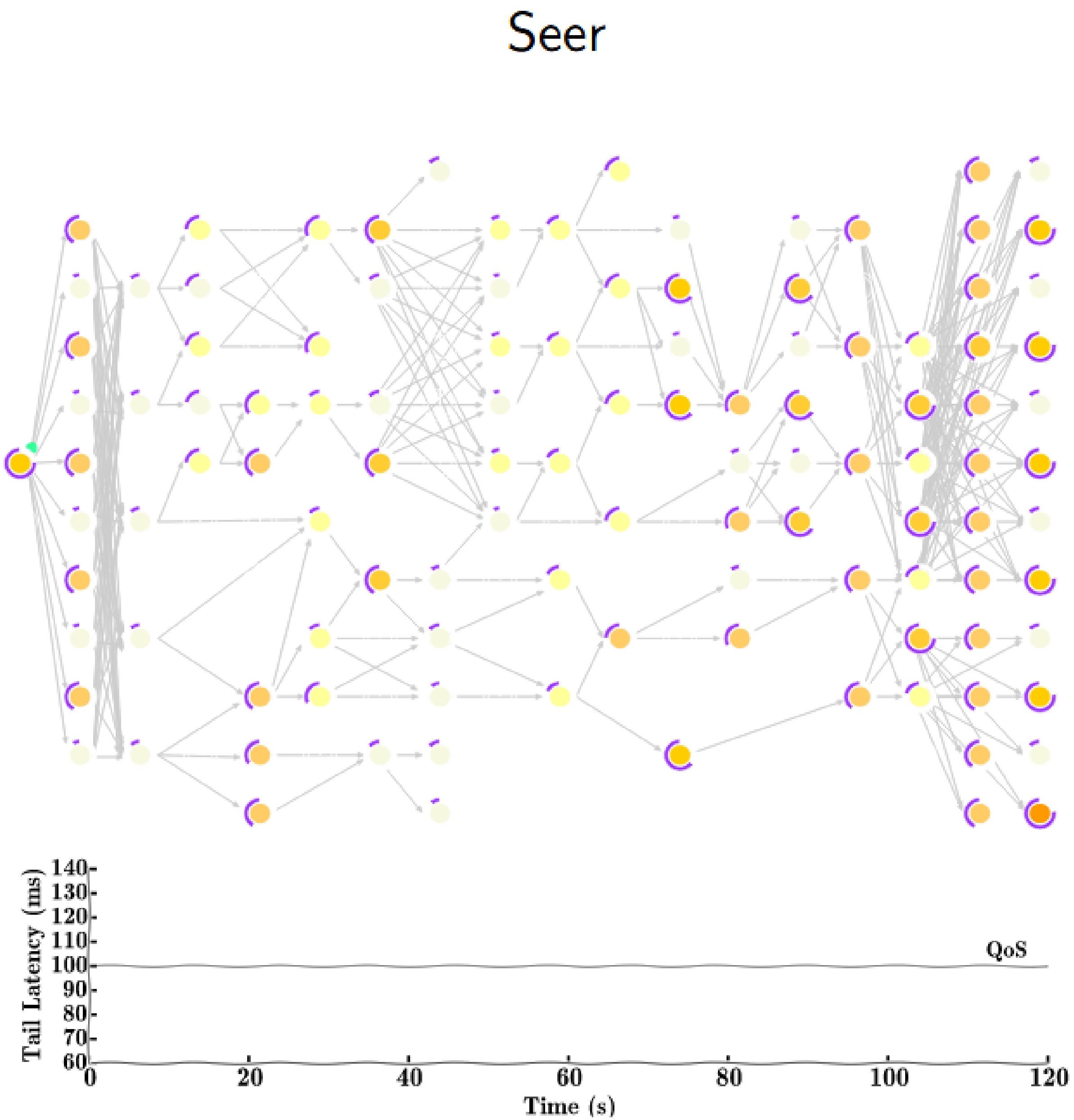
- Tracing interval < 500ms → low accuracy
- Tracing interval > 100ms → no further improvement

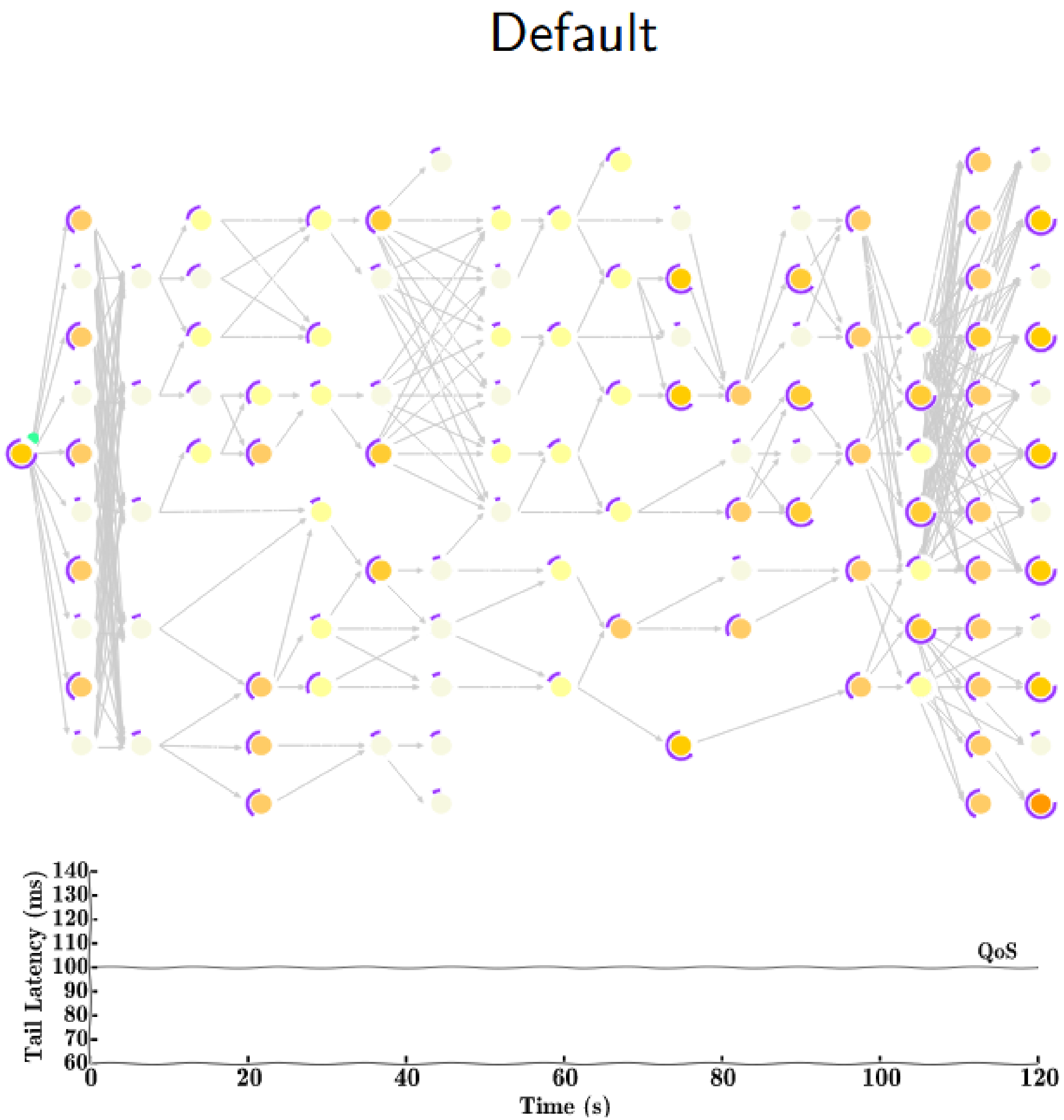
Avoiding QoS Violations



- Identify cause of QoS violation
  - Private cluster: performance counters & utilization monitors
  - Public cluster: contentious microbenchmarks
- Adjust resource allocation
  - RAPL (fine-grain DVFS) & scale-up for CPU contention
  - Cache partitioning (CAT) for cache contention
  - Memory capacity partitioning for memory contention
  - Network bandwidth partitioning (HTB) for net contention
  - Storage bandwidth partitioning for I/O contention
- Application level bugs
  - Human needs to intervene

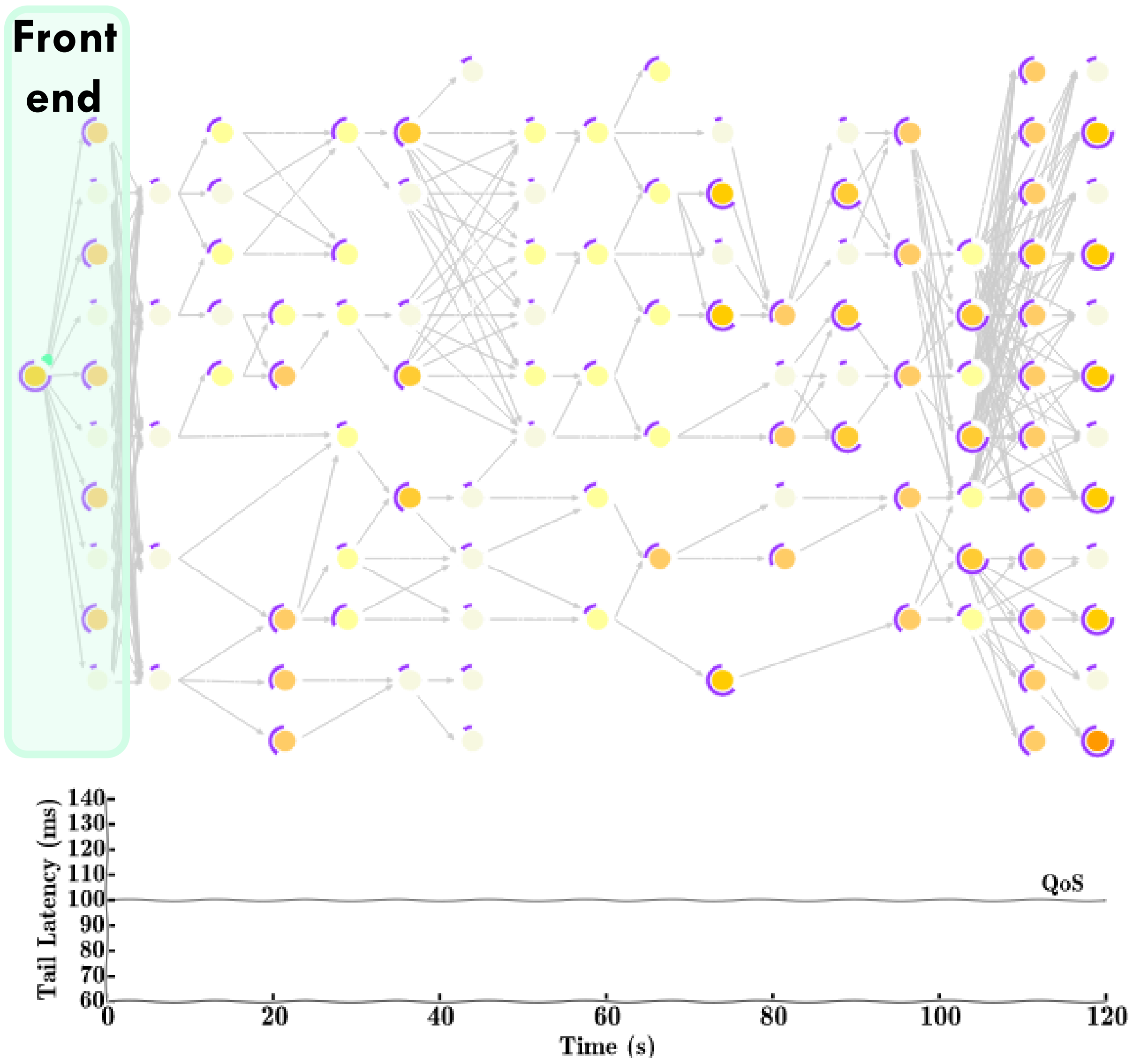
Demo


Seer
Tail Latency (ms)
140
130
120
110
100
90
80
70
60
QoS
0
20
40
60
80
100
120
Time (s)

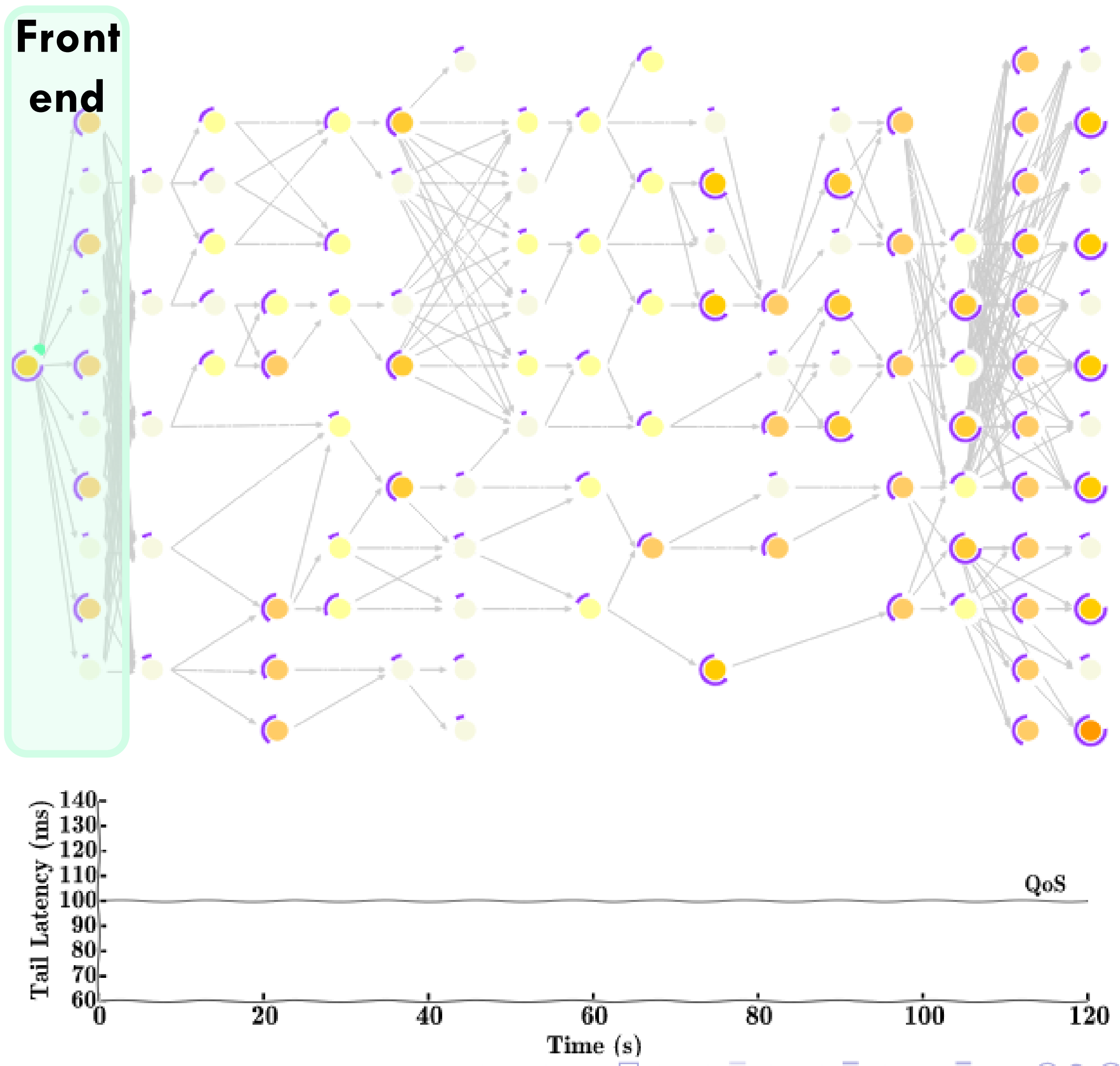

Default
Tail Latency (ms)
140
130
120
110
100
90
80
70
60
QoS
0
20
40
60
80
100
120
Time (s)

Demo

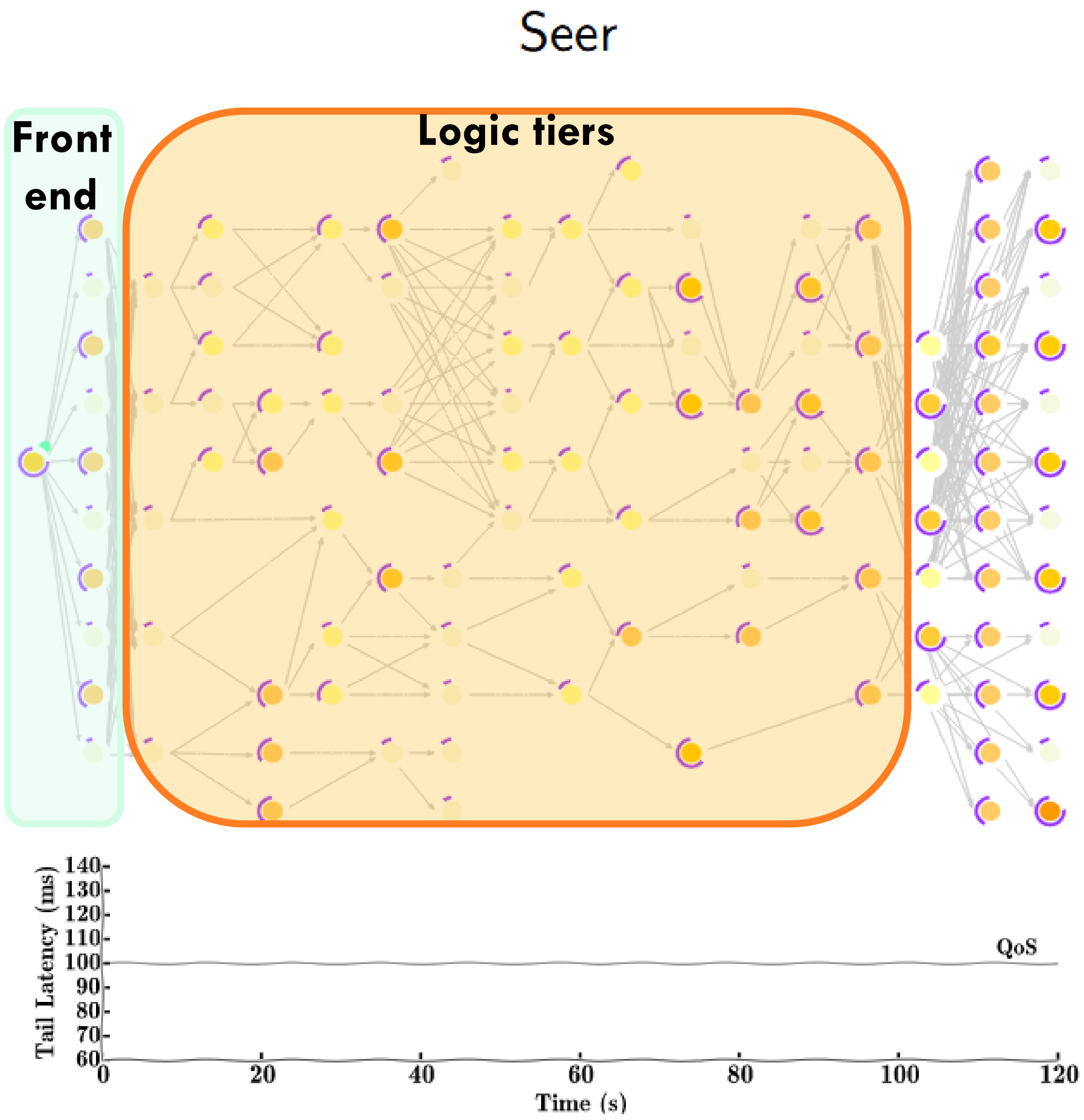
Seer



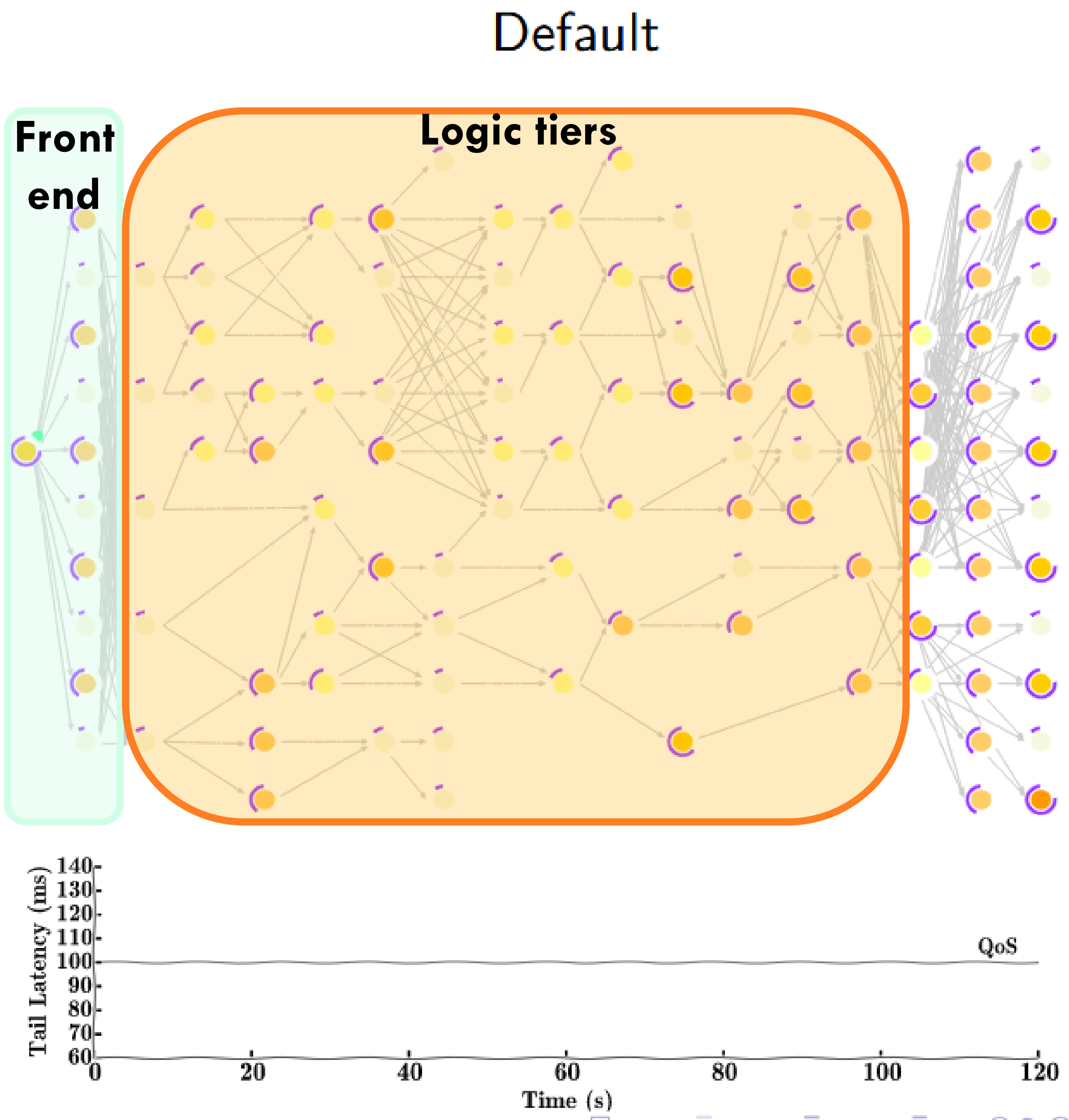
Default



Demo



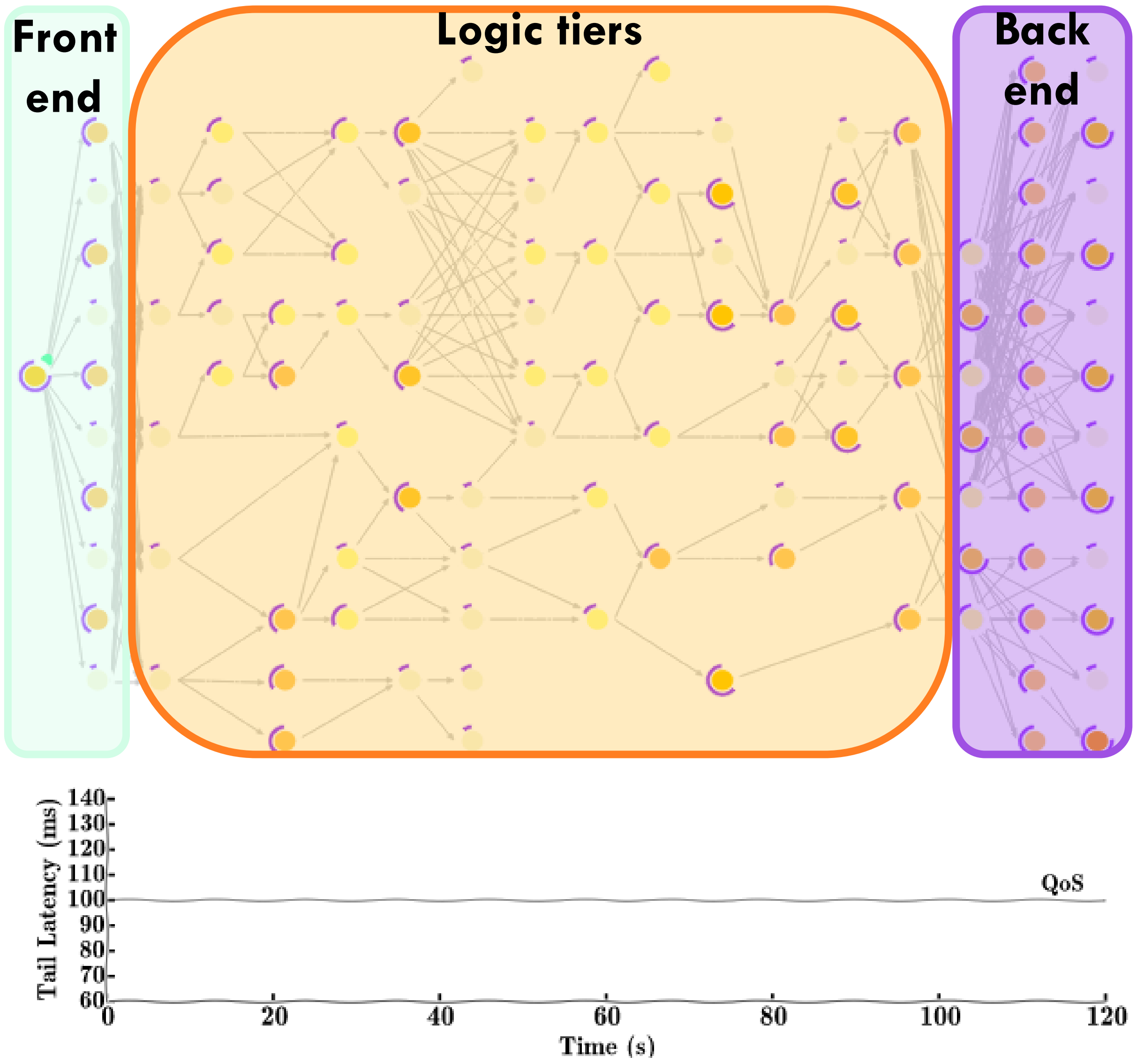
Seer
Front end
Logic tiers
Tail Latency (ms)
140
130
120
110
100
90
80
70
60
QoS
0
20
40
60
80
100
120
Time (s)



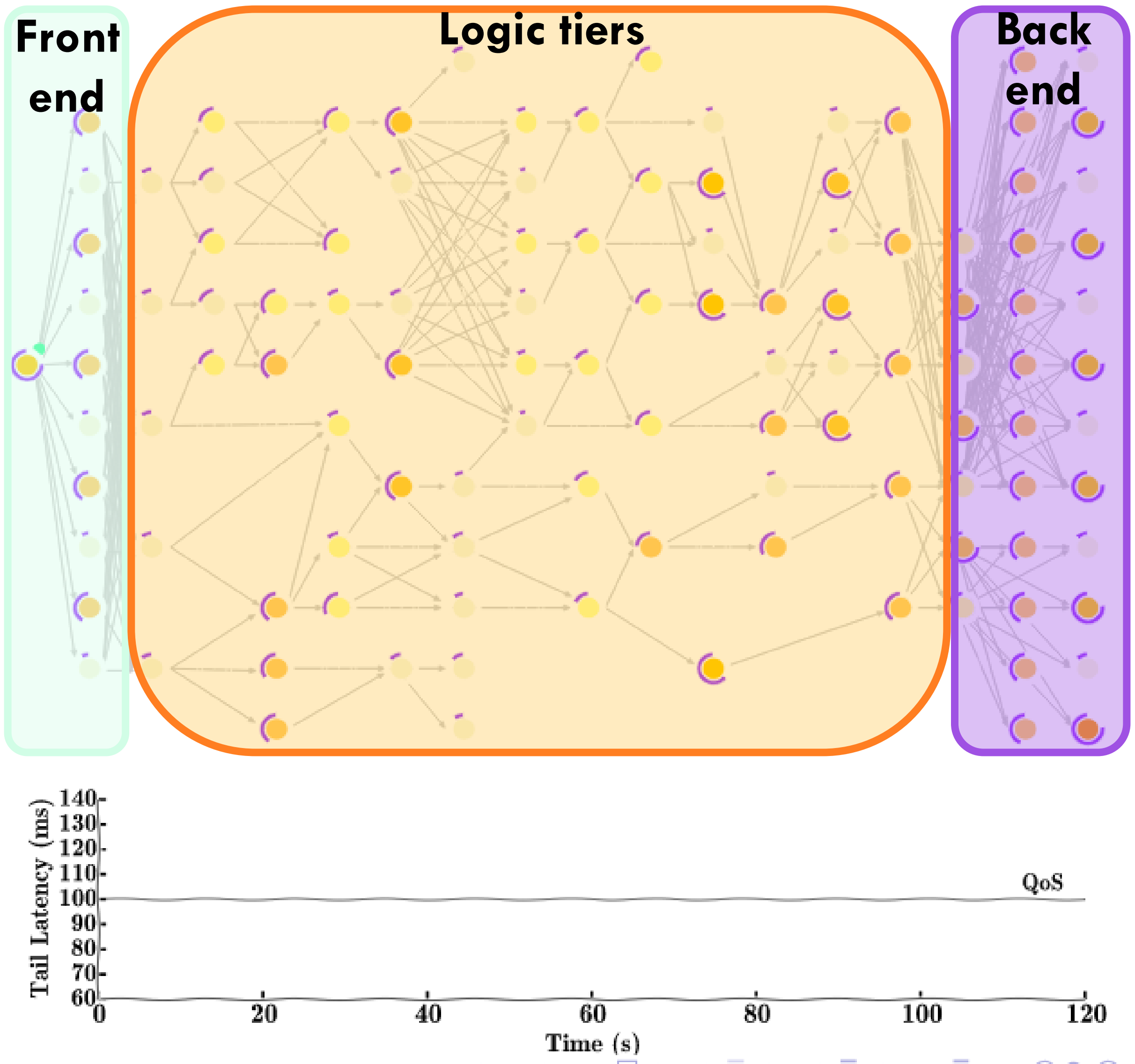
Default
Front end
Logic tiers
Tail Latency (ms)
140
130
120
110
100
90
80
70
60
QoS
0
20
40
60
80
100
120
Time (s)

Demo

Seer



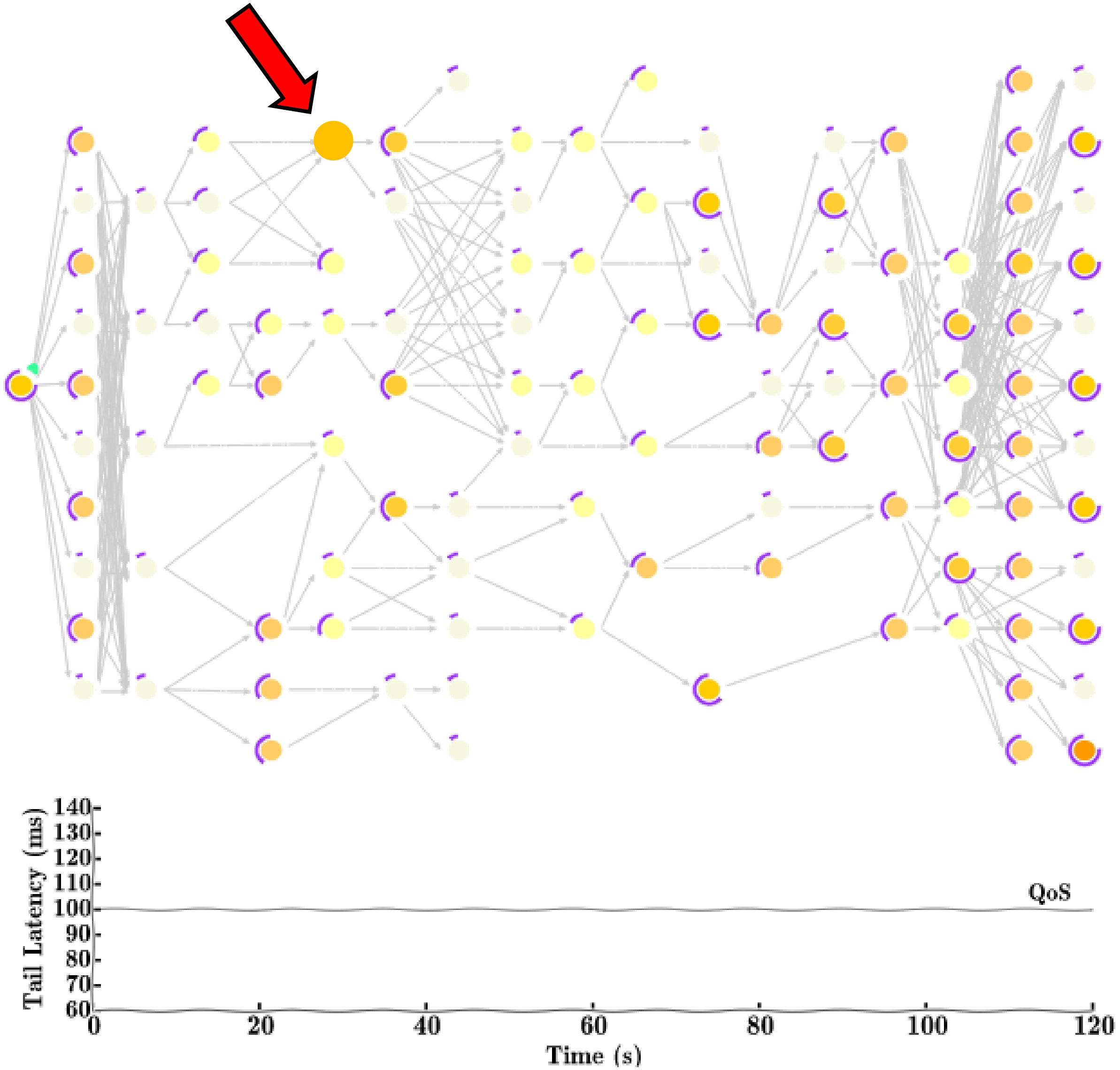
Default



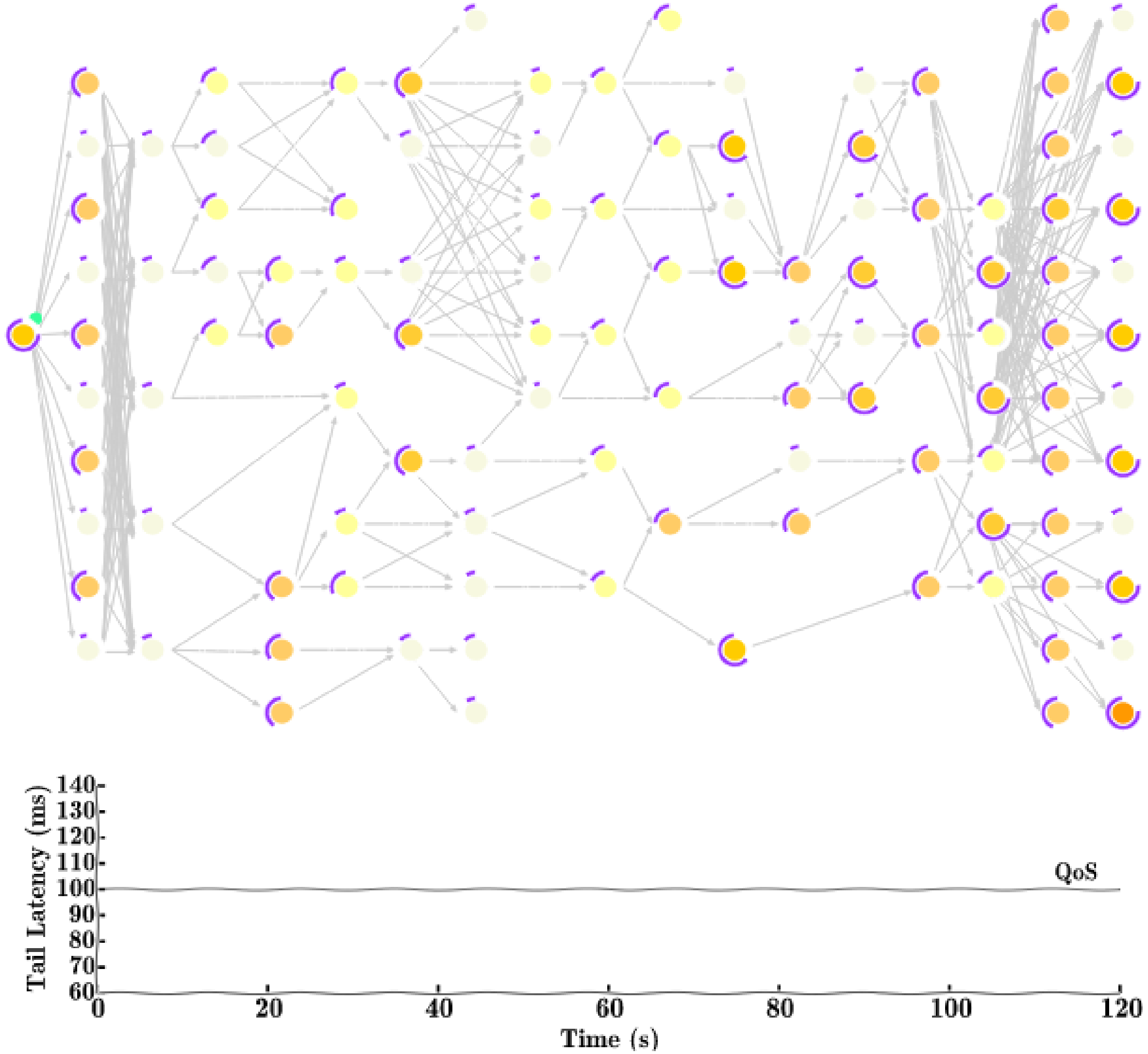
Demo

Queue

Seer

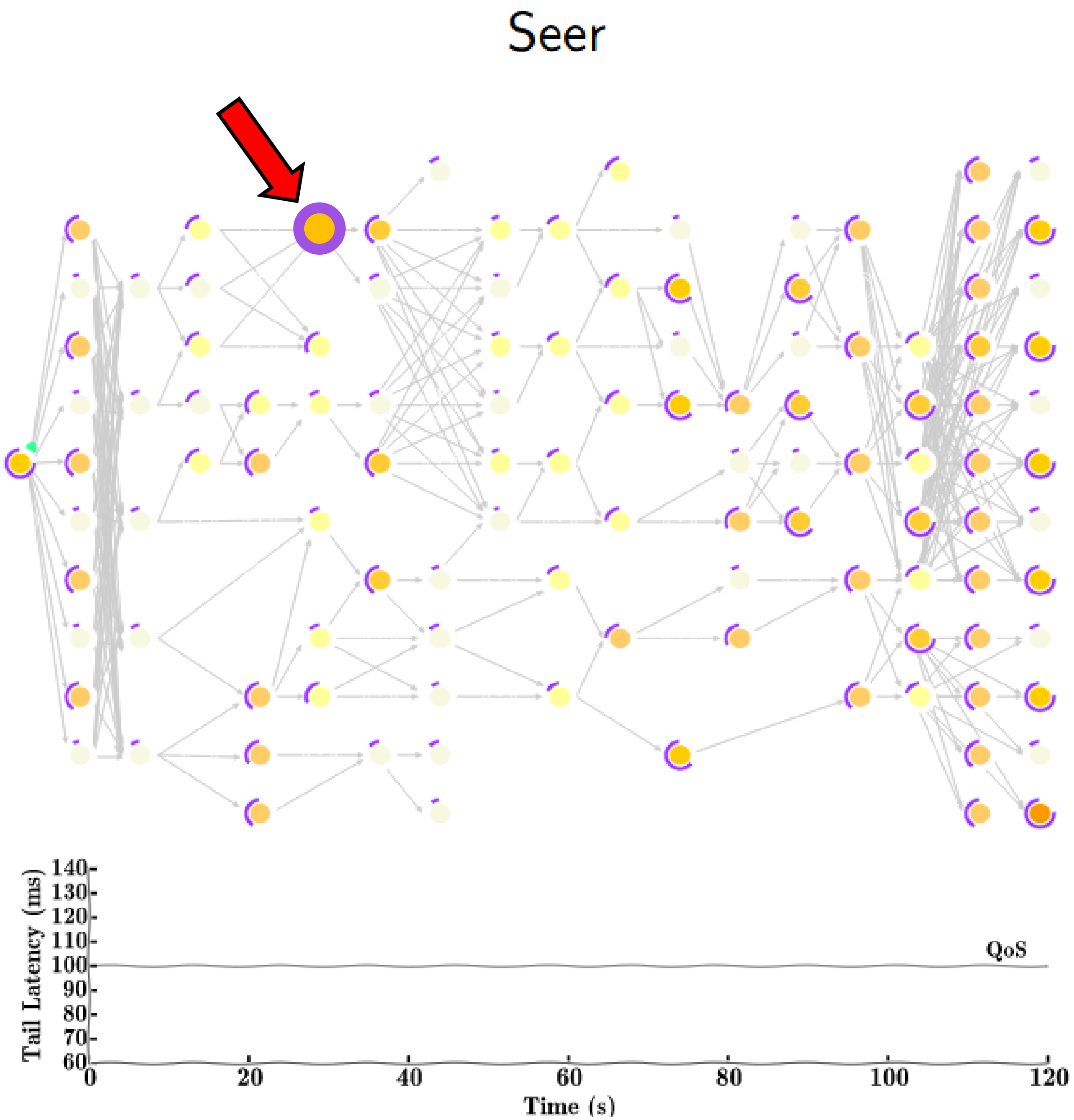


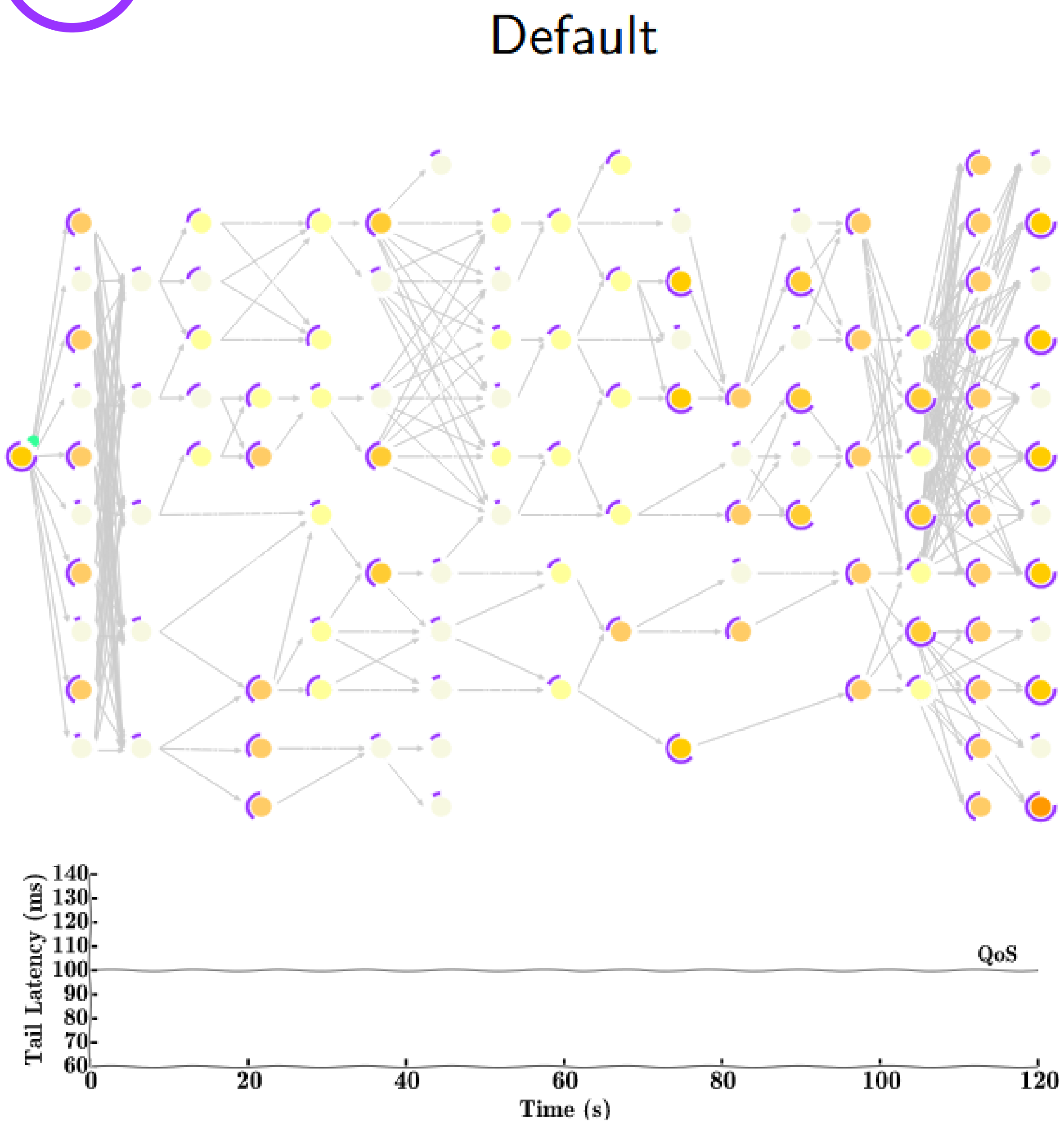
Default



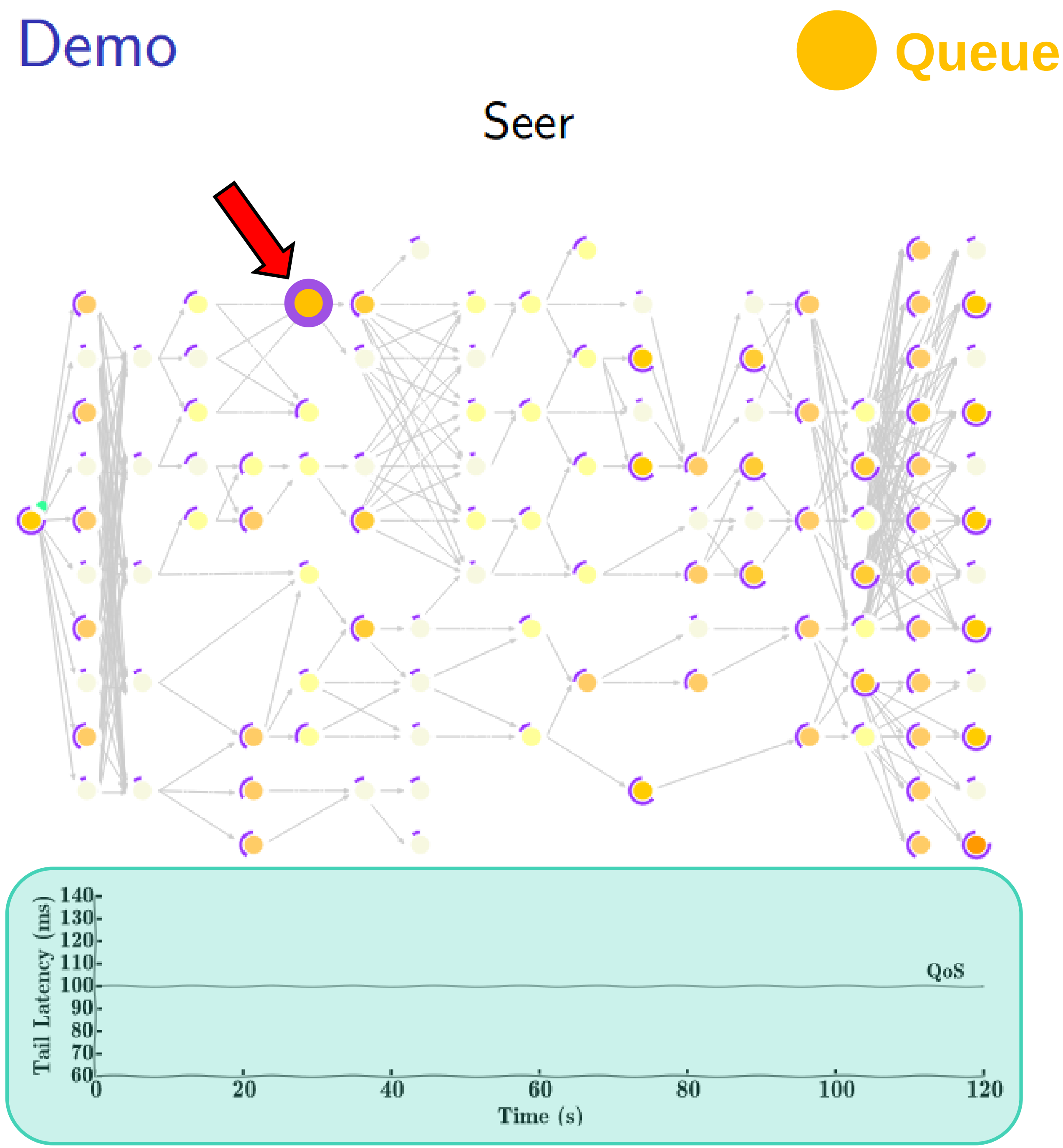
Demo

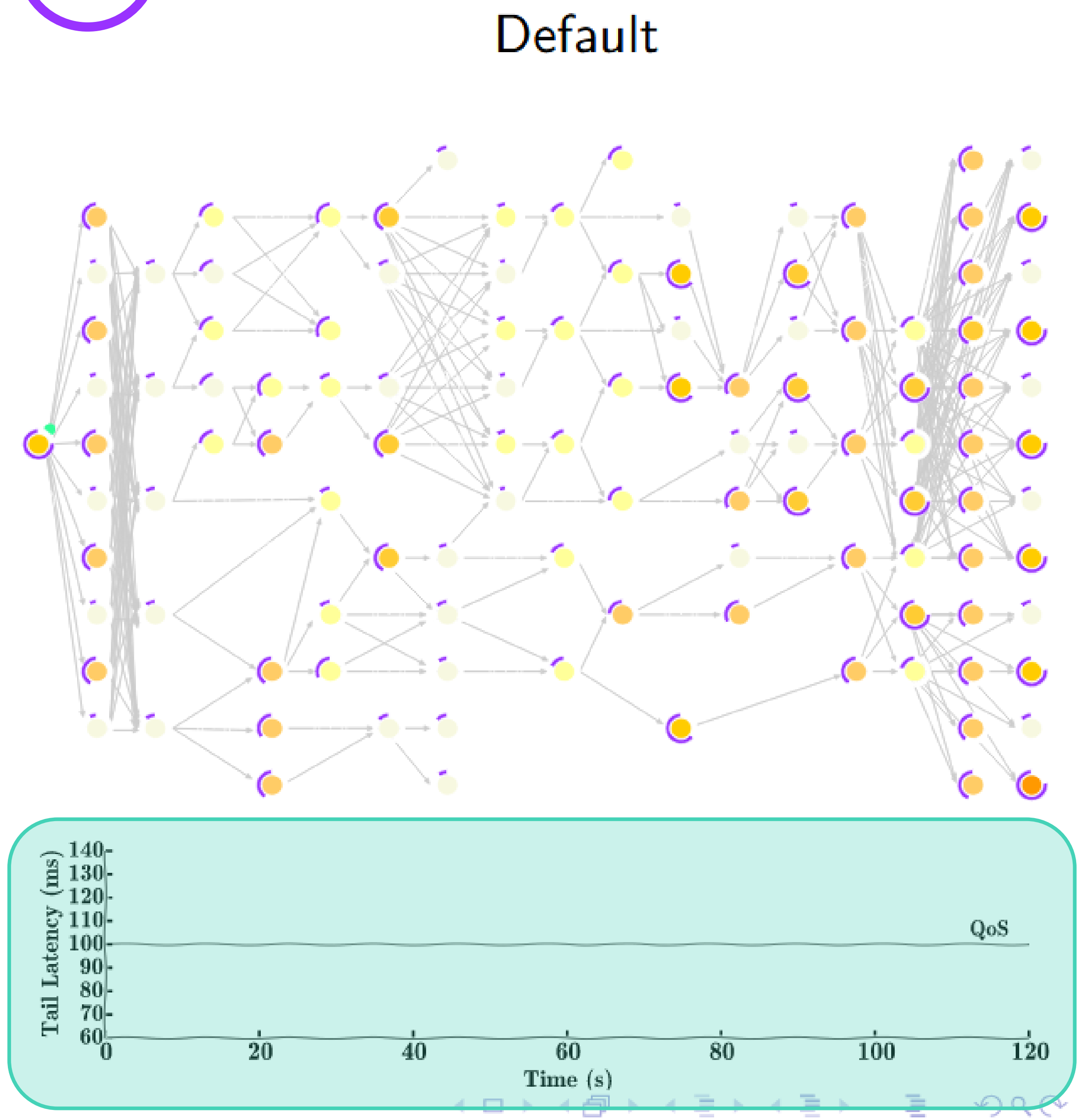
Queue CPU



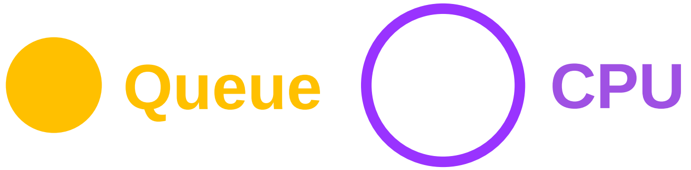


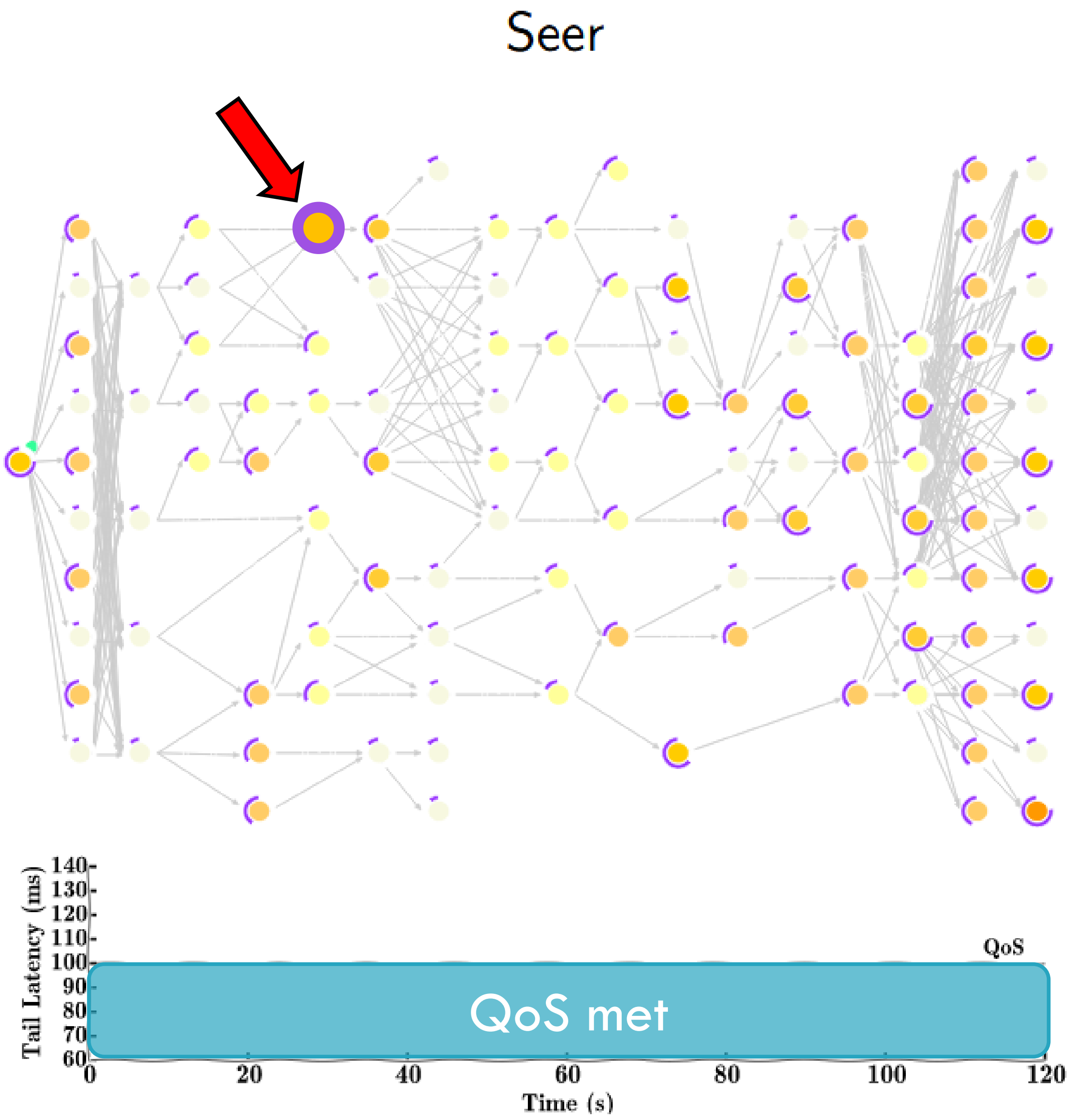
Demo

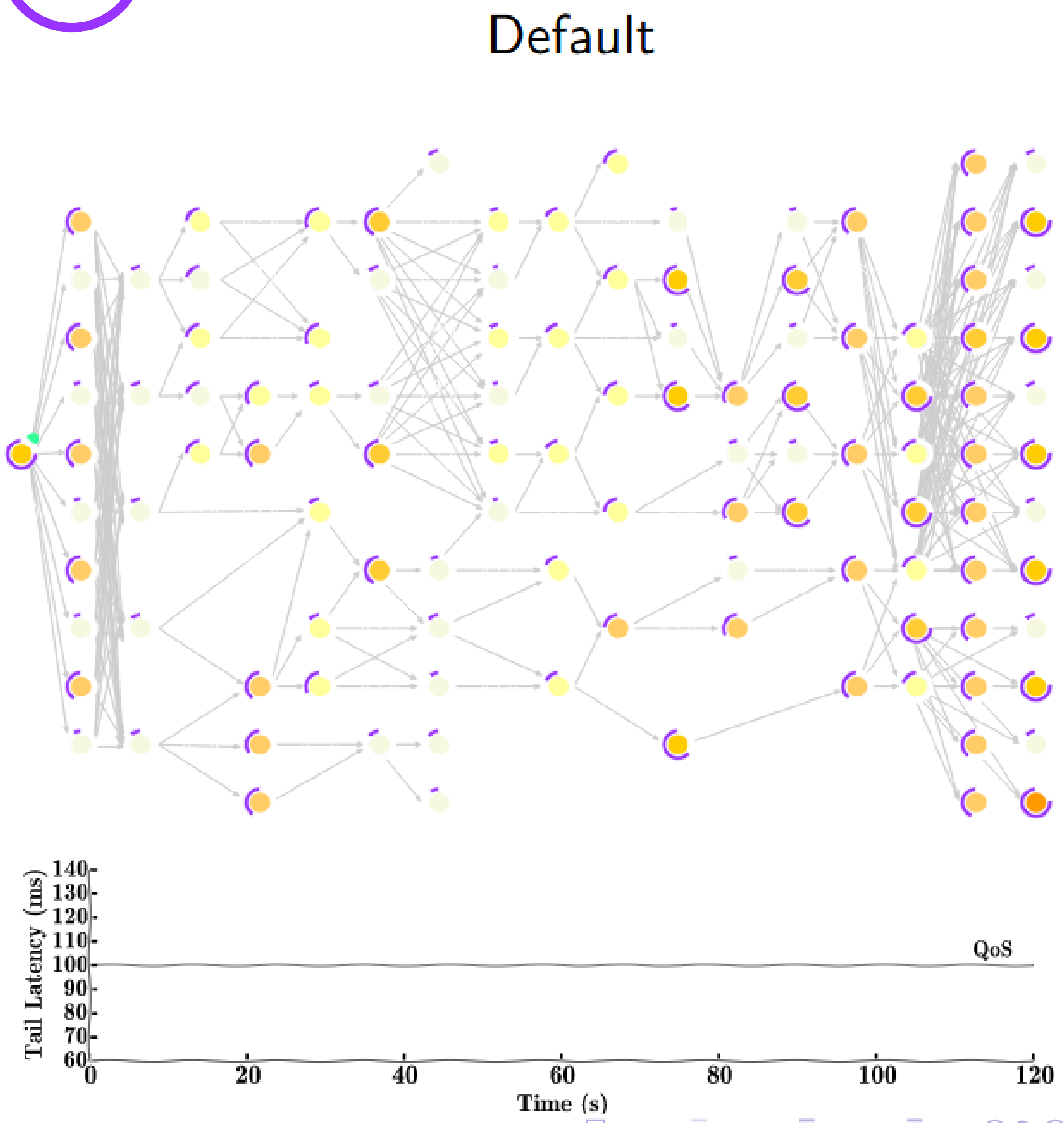

Queue
CPU
Seer
Tail Latency (ms)
140
130
120
110
100
90
80
70
60
QoS
0
20
40
60
80
100
120
Time (s)


Default
Tail Latency (ms)
140
130
120
110
100
90
80
70
60
QoS
0
20
40
60
80
100
120
Time (s)

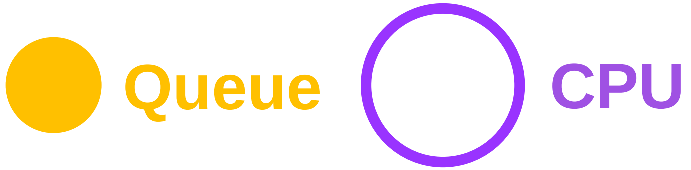
Demo

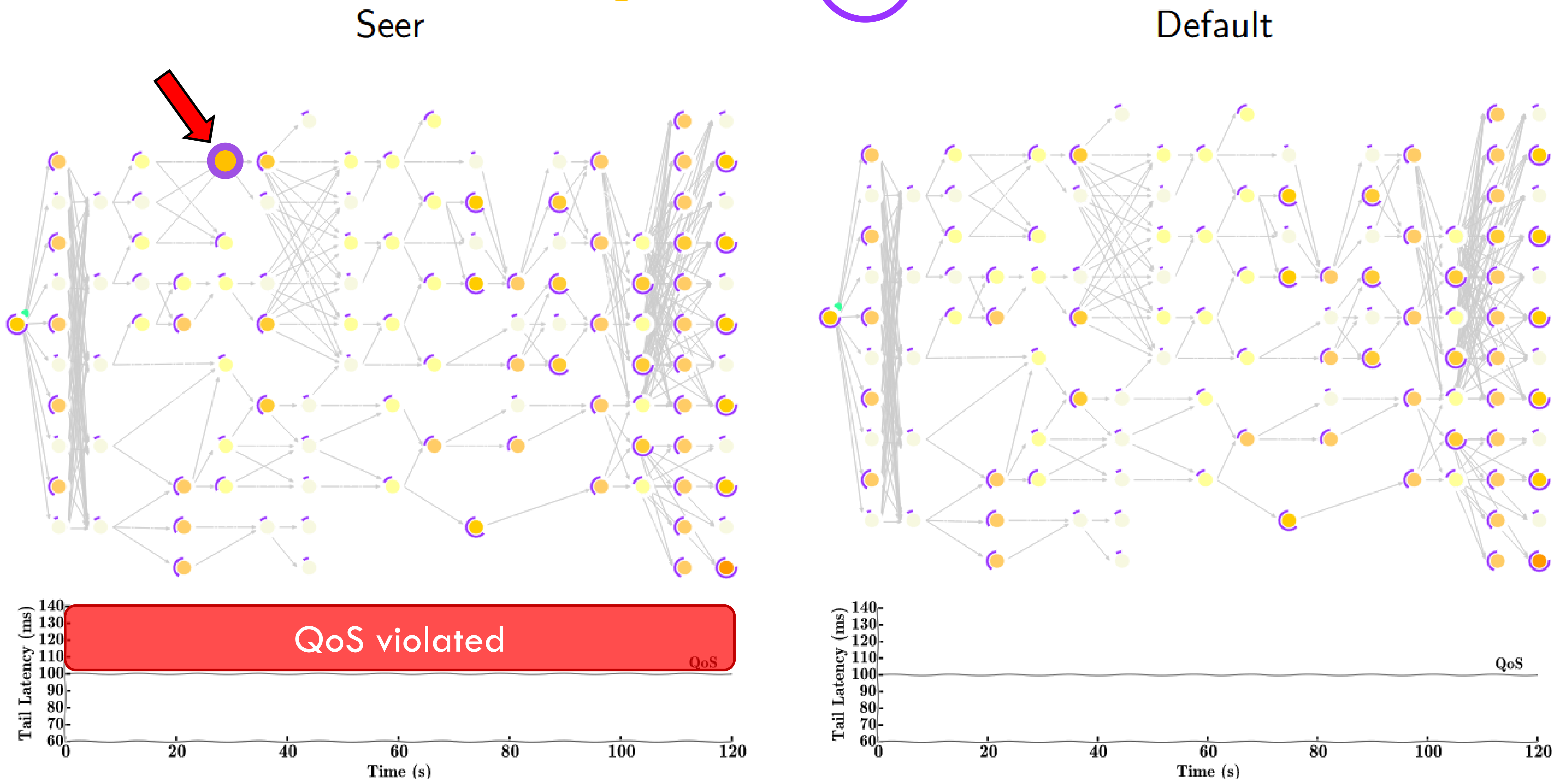

Queue
CPU


Seer
Tail Latency (ms)
140
130
120
110
100
90
80
70
60
0
20
40
60
80
100
120
Time (s)
QoS
QoS met


Default
Tail Latency (ms)
140
130
120
110
100
90
80
70
60
0
20
40
60
80
100
120
Time (s)
QoS

Demo

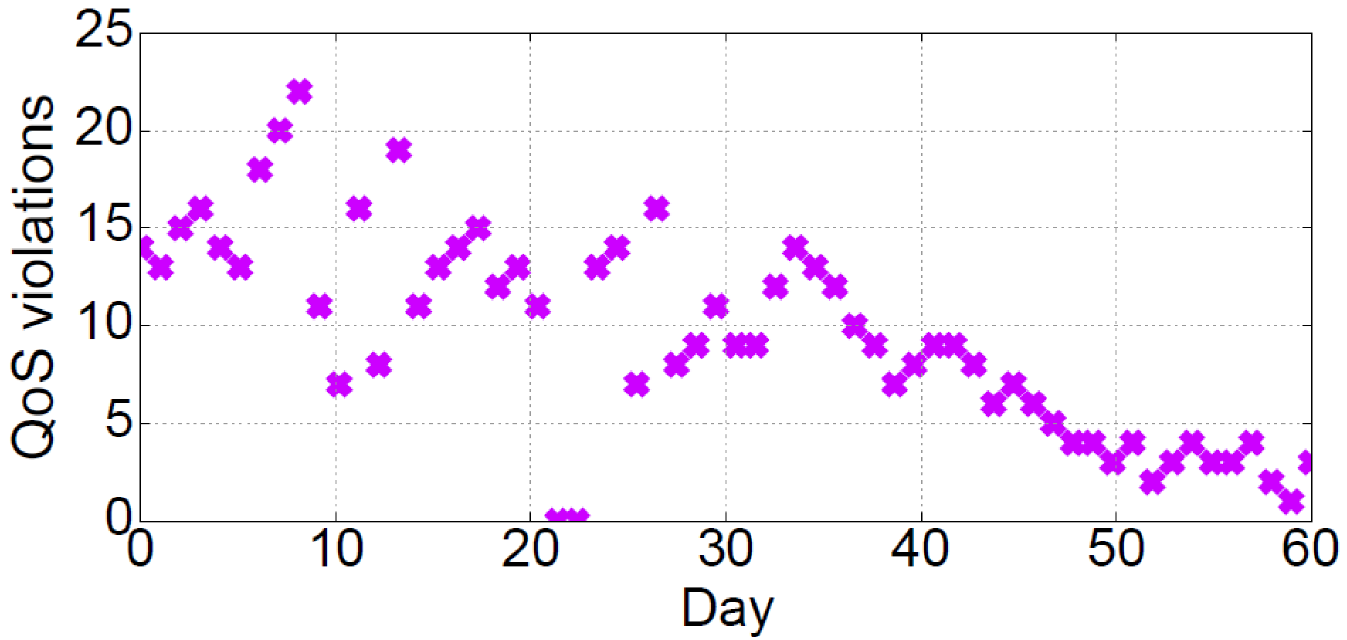

Queue
CPU


Seer
Default
QoS violated
QoS
Tail Latency (ms)
Time (s)
140
130
120
110
100
90
80
70
60
0
20
40
60
80
100
120

Demo

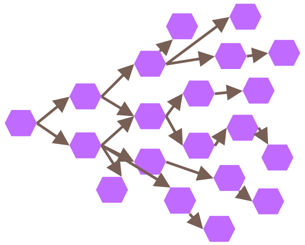
Demo: http://www.csl.cornell.edu/~delimitrou/2019.asplos.seer.demo.mp4

Using ML to Design Better Cloud Systems

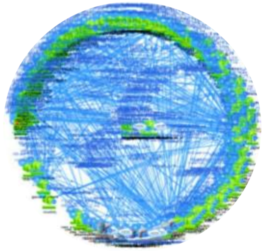


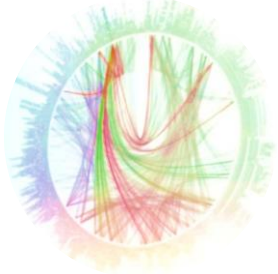
- Large-scale Social Network deployment (~600 users, ~2 months deployment)
- Offload Seer on Google TPU v2 → 24x-118x improvement in training and inference
- Several bugs found (blocking RPCs, livelocks, shared data structs, cyclic dependencies, insufficient resources, etc.)
- Fewer QoS violations over time

Conclusions

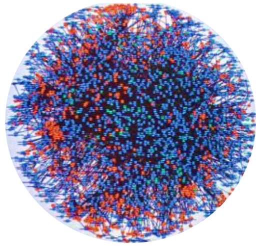


- Microservices become increasingly popular



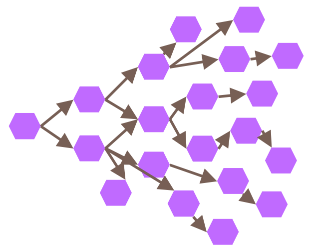


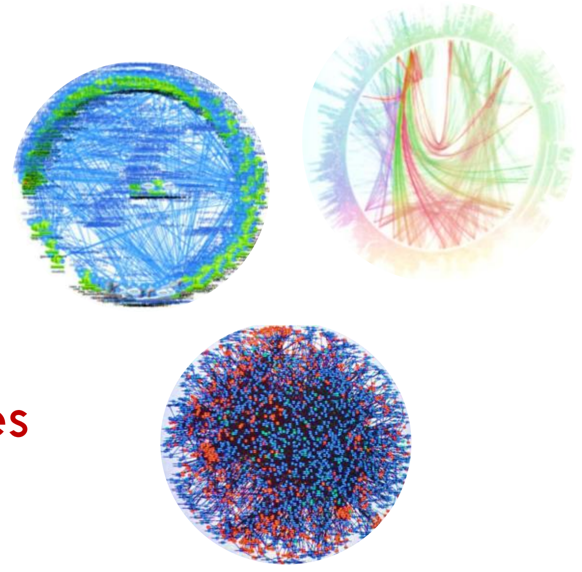
- Traditional performance debugging techniques do not scale and introduce long recovery times



- Seer leverages DL to anticipate QoS violations & find their root causes
  - >90% detection accuracy, avoids 86% of QoS violations
- Provides insight on how to better design and deploy complex microservices
- Practical solutions for systems whose scale make previous empirical solutions impractical

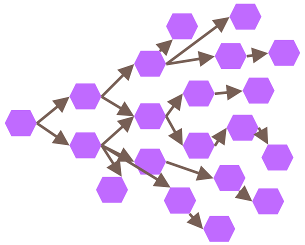
Questions?



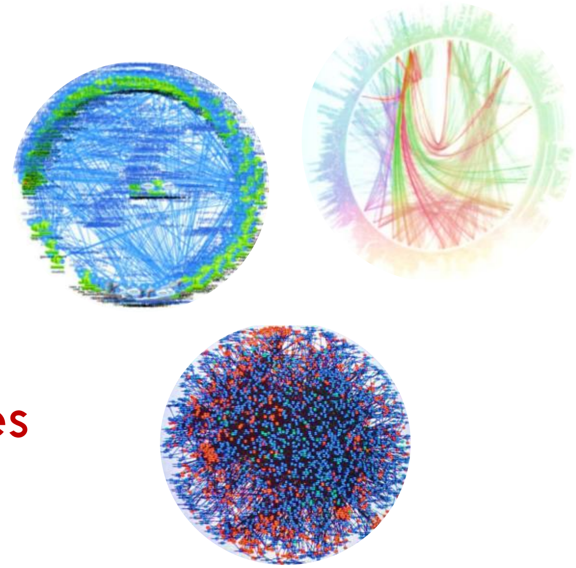


- Microservices become increasingly popular
- Traditional performance debugging techniques do not scale and introduce long recovery times
- Seer leverages DL to anticipate QoS violations & find their root causes
  - >90% detection accuracy, avoids 86% of QoS violations
- Provides insight on how to better design and deploy complex microservices
- Practical solutions for systems whose scale make previous empirical solutions impractical

Questions?



- Microservices become increasingly popular
- Traditional performance debugging techniques do not scale and introduce long recovery times
- Seer leverages DL to anticipate QoS violations & find their root causes
  - >90% detection accuracy, avoids 86% of QoS violations
- Provides insight on how to better design and deploy complex microservices
- Practical solutions for systems whose scale make previous empirical solutions impractical



Thank you!