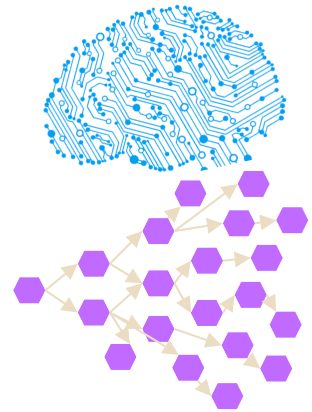


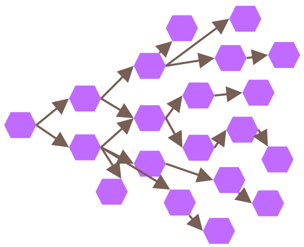
The Hardware & Software Implications of Microservices and How Big Data Can Help

Christina Delimitrou

Cornell University

with Yu Gan, Yanqi Zhang, Shuang Chen, Neeraj Kulkarni, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Ankitha Shetty, Nayan Katarki, Brett Clancy, Chris Colen, Dailun Cheng, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Meghna Pancholi, Siyuan Hu

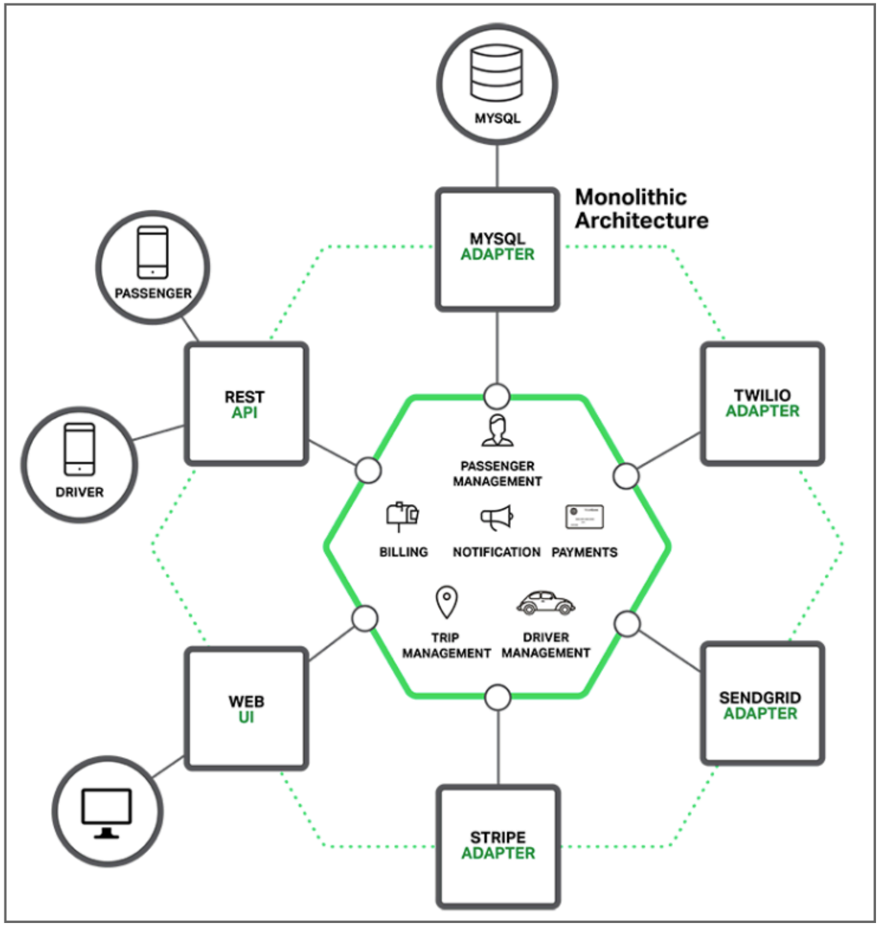
Executive Summary

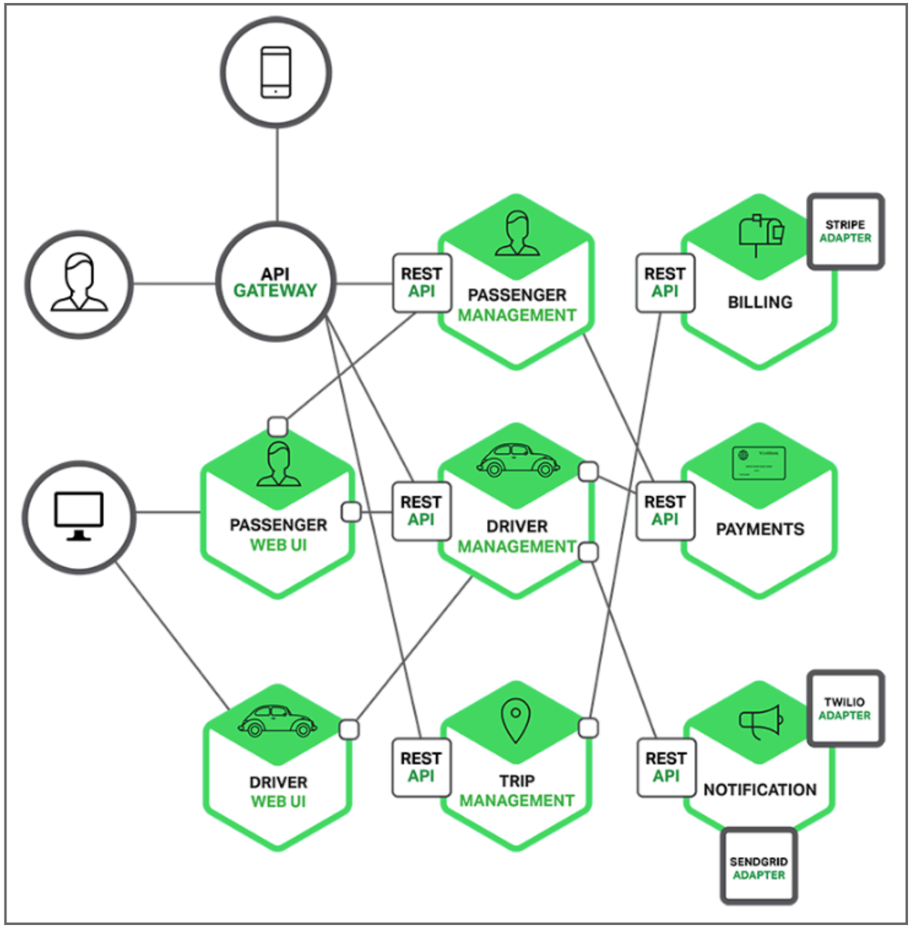


- Shift from monoliths to microservices:
  - Modularity, specialization, simplicity, accelerated development
  - Change assumptions about datacenter server design
  - Complicate scheduling and resource management
  - Amplify tail@scale effects

- Revisit architectural design decisions for microservices
- Highlight management challenges of microservices
- Motivate the need for data-driven approaches for systems whose scale & complexity keeps increasing

From Monoliths to Microservices





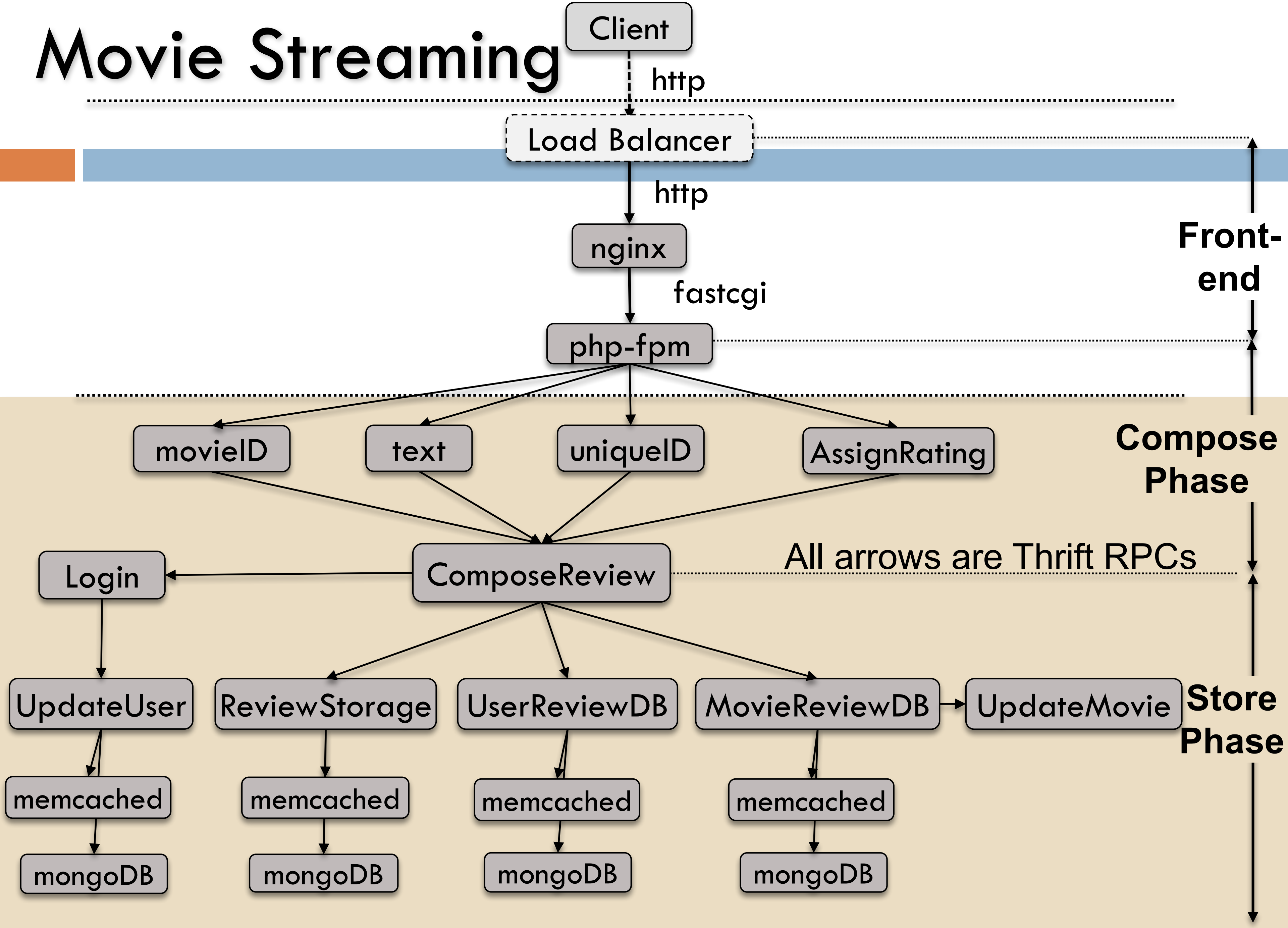
Motivation

- Advantages of microservices:
  - Ease & speed of code development & deployment
  - Security, error isolation
  - PL/framework heterogeneity
- Challenges of microservices:
  - Change server design assumptions
  - Complicate resource management → dependencies
  - Amplify tail-at-scale effects
  - More sensitive to performance unpredictability
  - No representative end-to-end apps with microservices

An End-to-End Suite for Cloud & IoT Microservices

- 4 end-to-end applications using popular open-source microservices → ~30-40 microservices per app
  - Social Network
  - Movie Reviewing/Renting/Streaming
  - E-commerce
  - Drone control service
- Programming languages and frameworks:
  - node.js, Python, C/C++, Java/Javascript, Scala, PHP, and Go
  - Nginx, memcached, MongoDB, CockroachDB, Mahout, Xapian
  - Apache Thrift RPC, RESTful APIs
  - Docker containers
  - Lightweight RPC-level distributed tracing

Movie Streaming

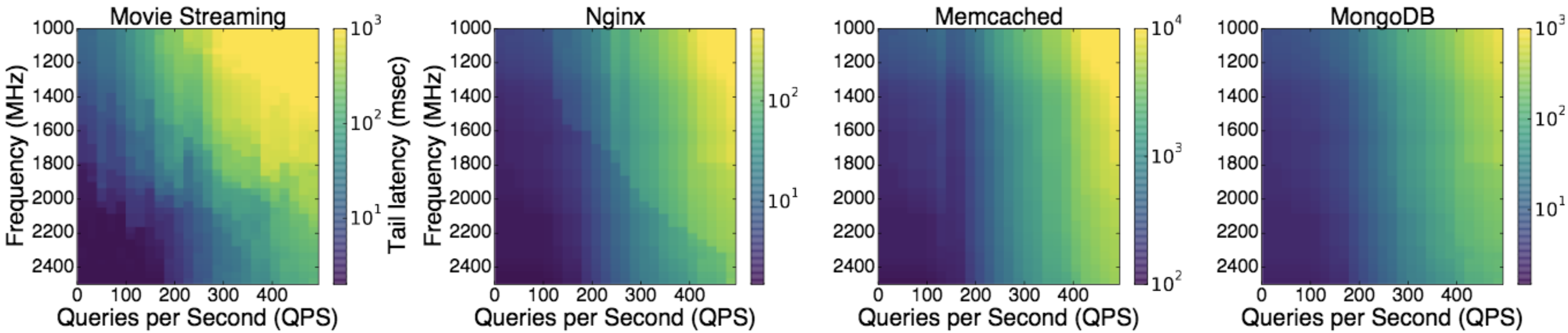


Movie Streaming

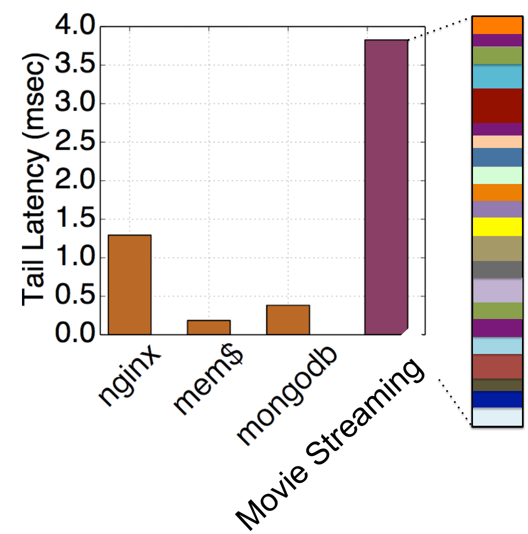
- Browse movie info (movie plot, photos, videos, cast, stats, etc.)
- ML widgets:
  - Recommender for movies to watch
  - Recommender for ads
- User authentication/Payment
- Search:
  - Xapian: search movie DB
- Analytics:
  - Mahout: user analytics based on input stored in HDFS
  - Spark MLlib: in-memory ML analytics

Architectural Implications

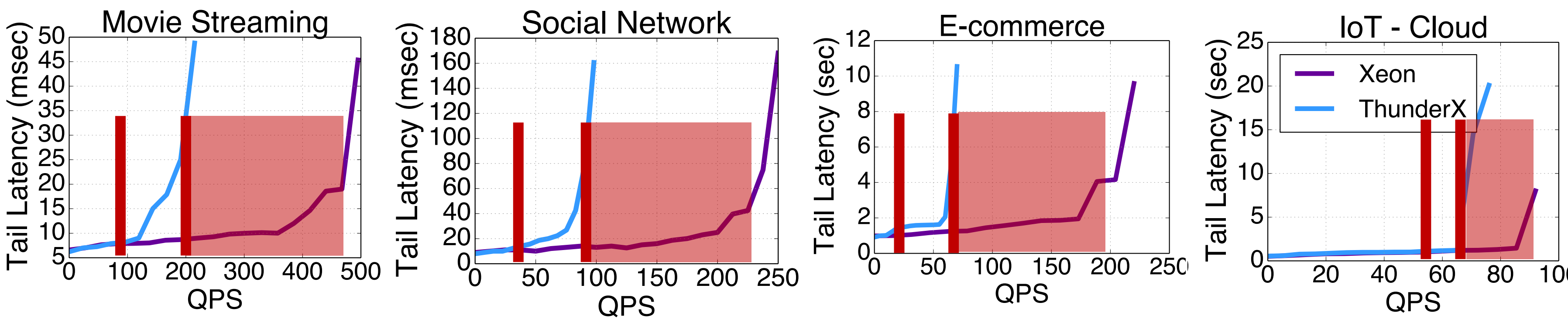
[CAL'18]



- Big vs. small servers:
  - Power management using RAPL
  - More pressure on single-thread performance, low tail latency

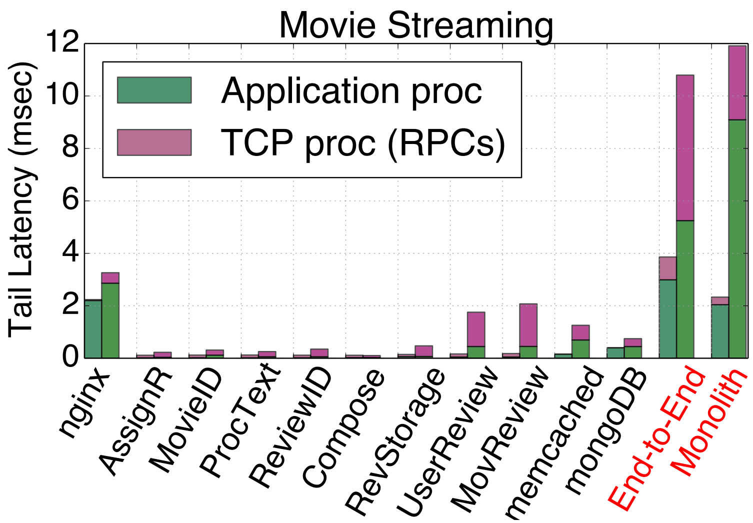


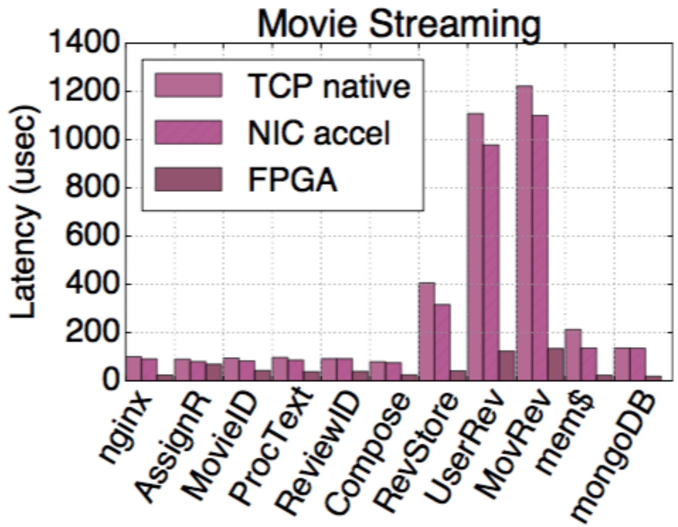
Architectural Implications



- Big vs. small servers:
  - Power management using RAPL
  - More pressure on single-thread performance, low tail latency
  - Low-power SoCs, e.g., Cavium ThunderX2
  - Similar latency, but earlier saturation

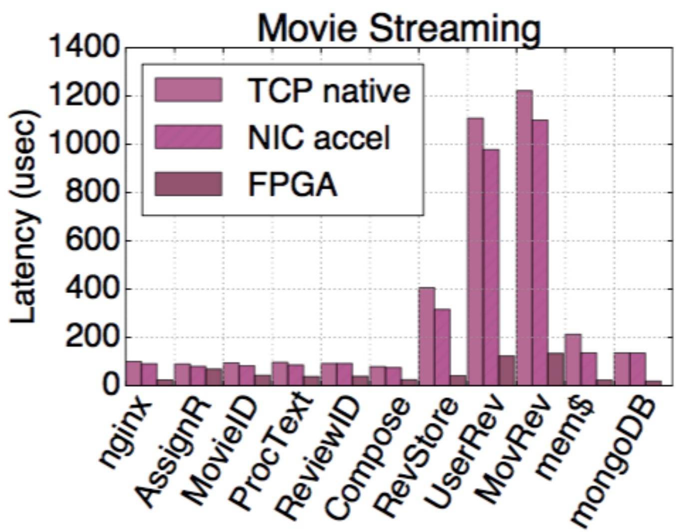
Architectural Implications

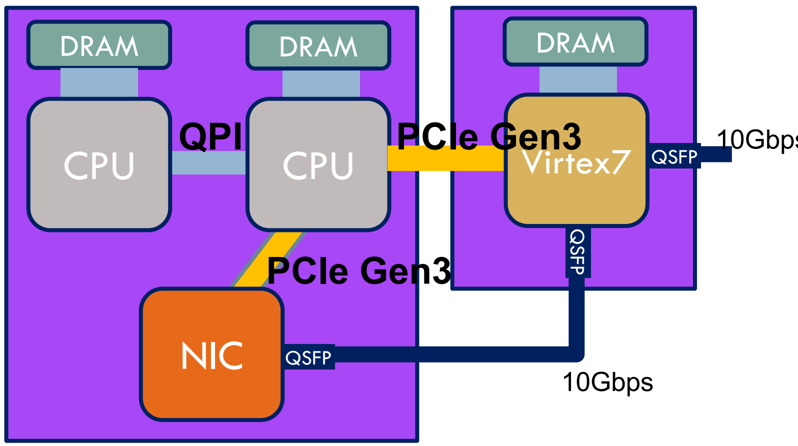




- Computation:Communication ratio:
  - Monolithic service → 70:30 @ high load
  - Microservices → 50:50 @ high load

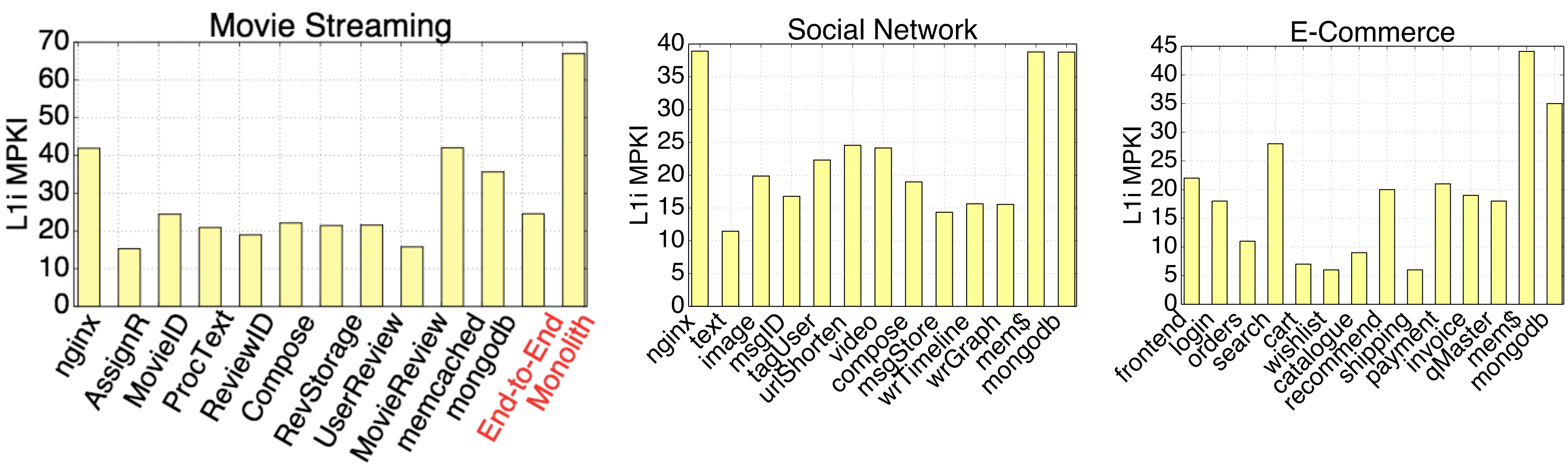
Architectural Implications





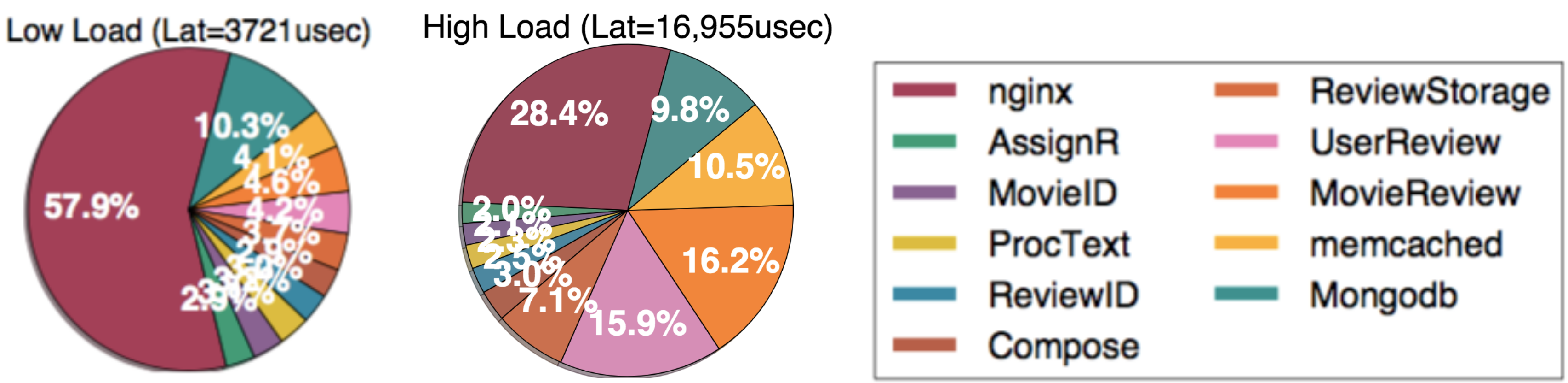
- Computation:Communication ratio:
  - Monolithic service → 70:30 @ high load
  - Microservices → 50:50 @ high load
  - RPC/REST acceleration → NIC offloads, FPGAs

Architectural Implications



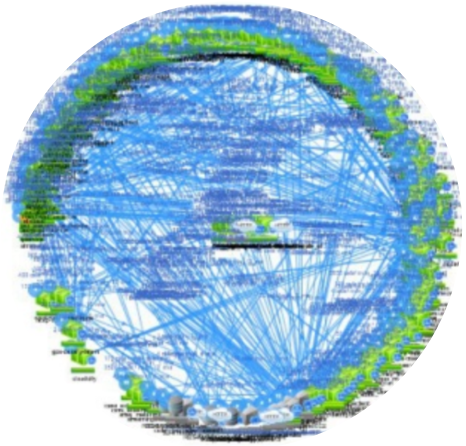
- L1-i cache pressure:
  - Monoliths → Large code footprints → L1i thrashing
  - Microservices → Small footprint/microservice
    - Assuming dedicated cores

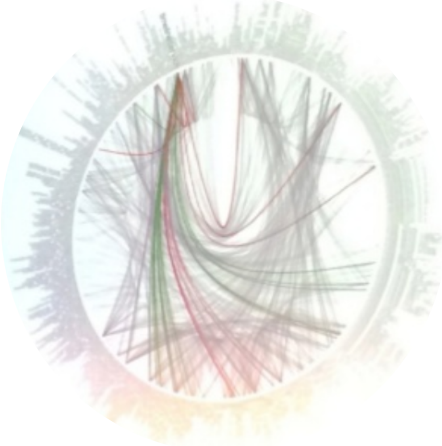
End-to-End Latency Breakdown

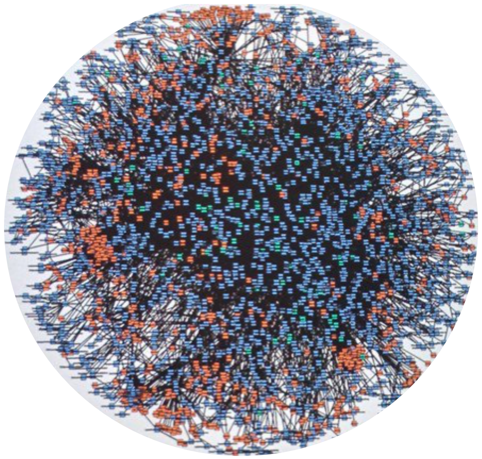


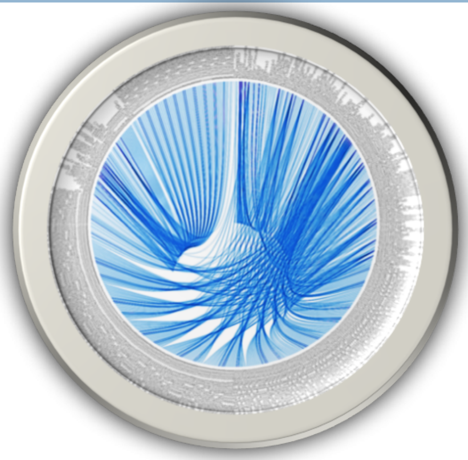
- Post-rightsizing (resource ratios to avoid glaring bottlenecks)
- Bottlenecks shift with load
- Need online, dynamic decisions

Resource Management Implications


Netflix

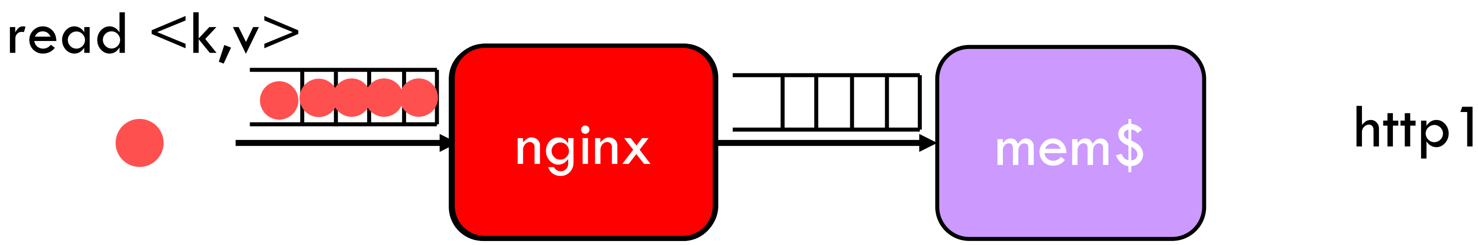

Twitter


Amazon

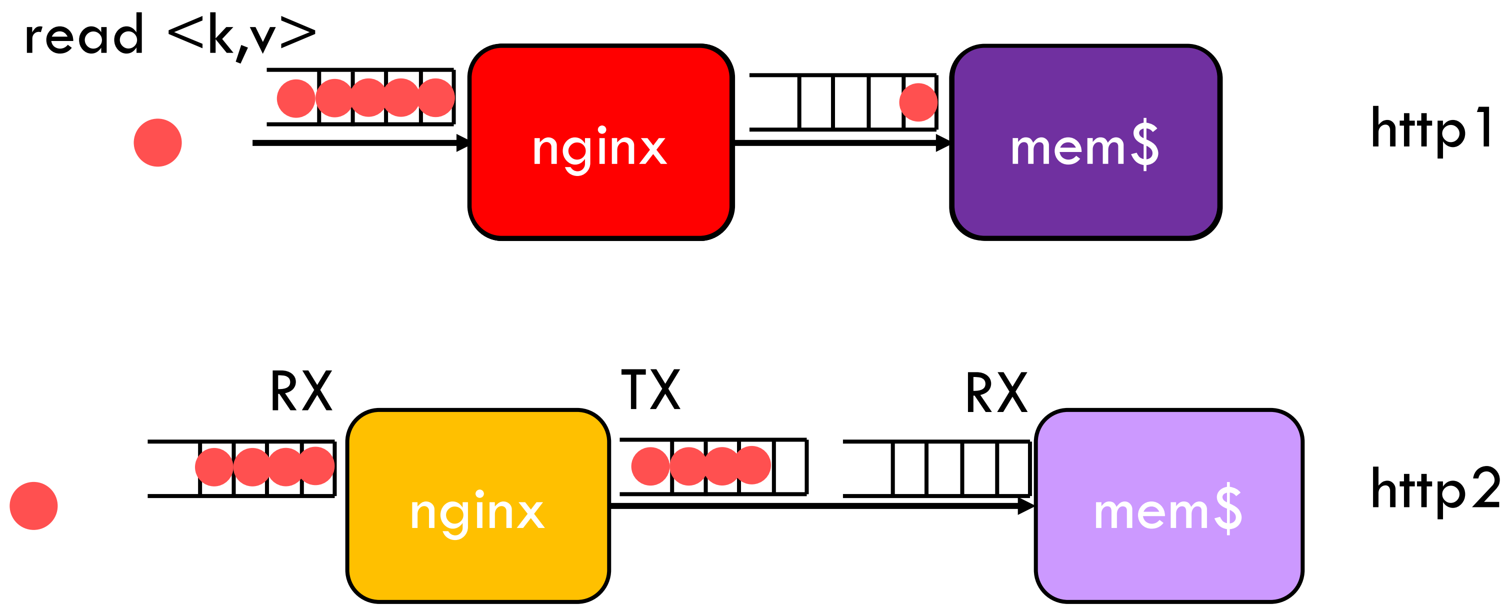

Movie Streaming

- Challenges of microservices:
  - Change server design assumptions
  - Dependencies complicate resource management

Dependencies & Backpressure



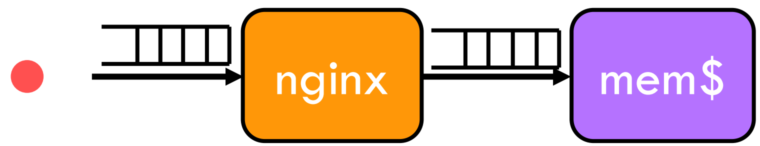
Dependencies & Backpressure

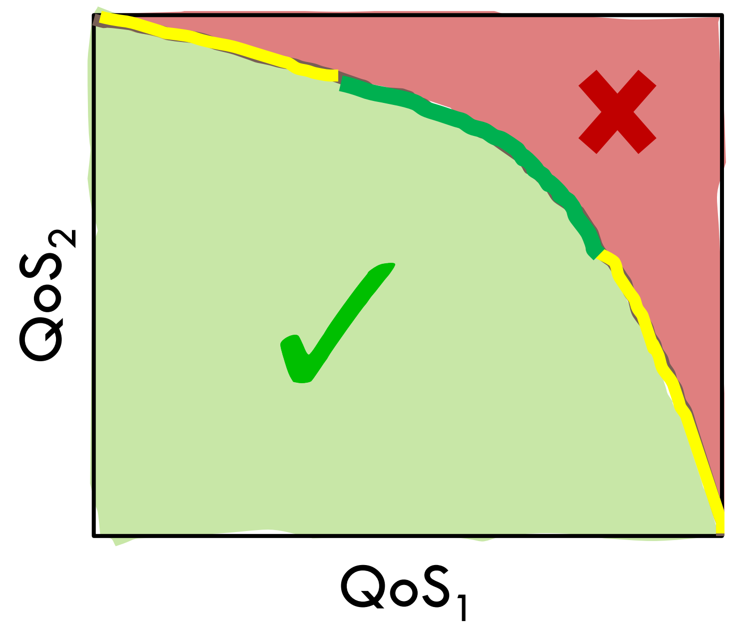


- Traditional techniques like autoscale may help/penalize the wrong microservice
- Dependencies change at runtime → difficult to infer impact

Determine Per-Tier QoS

- Queueing models

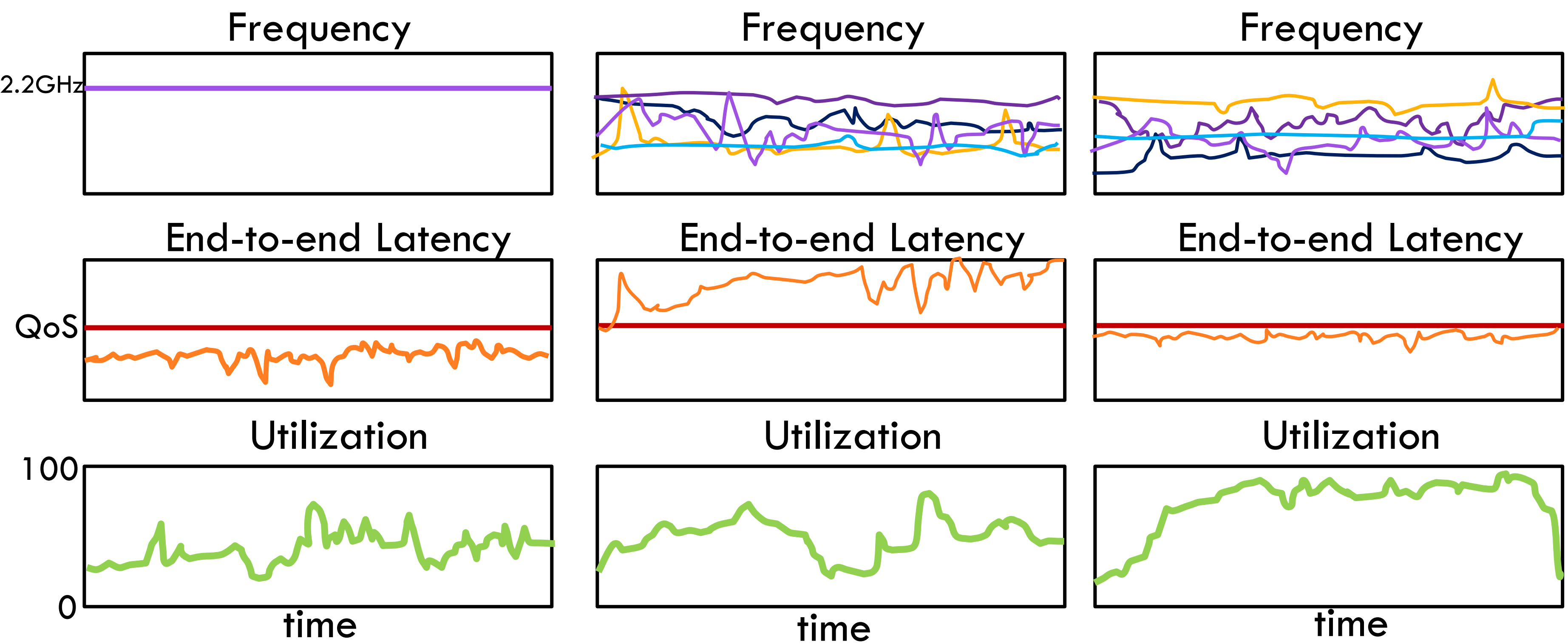




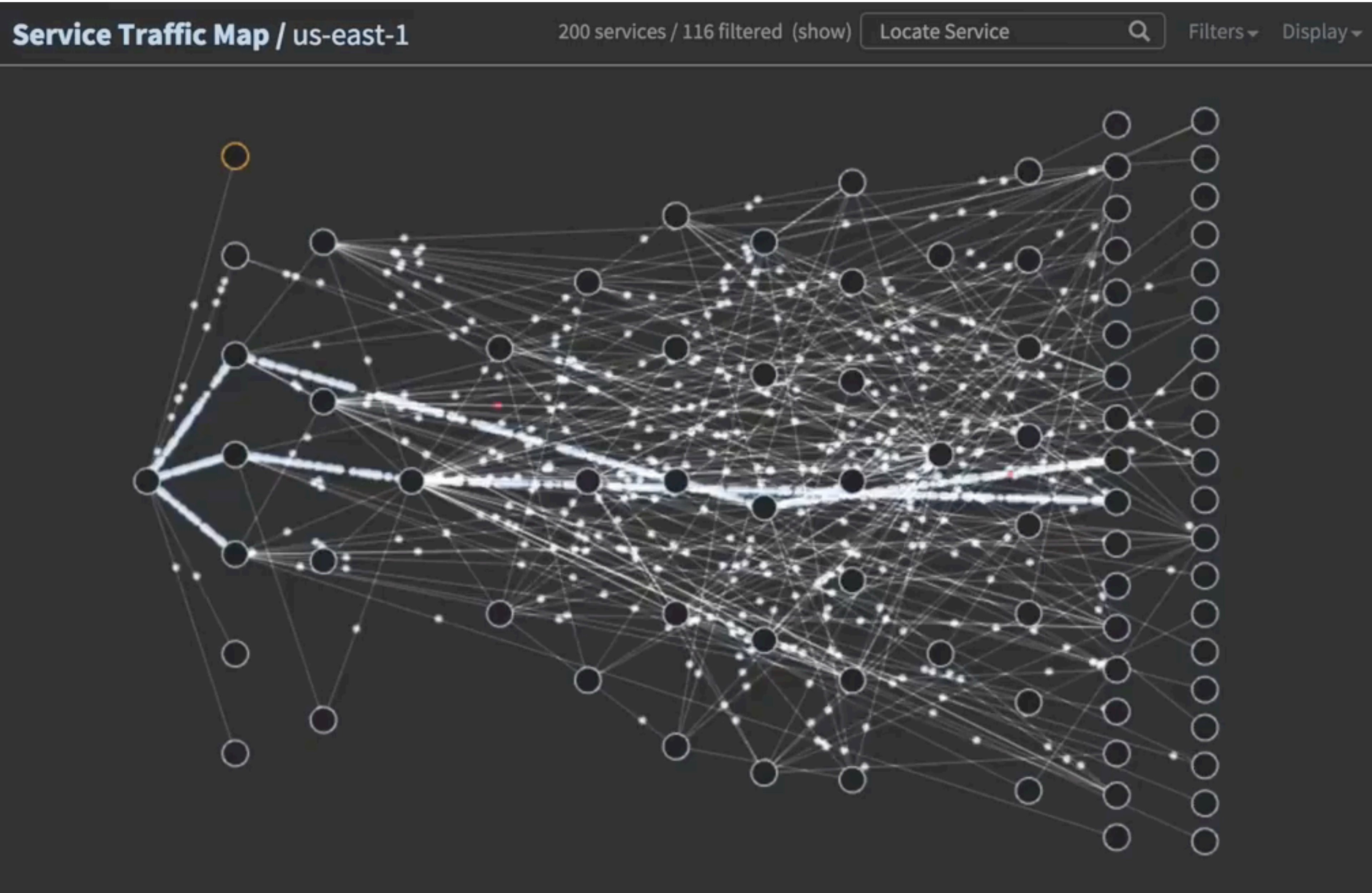
- Queueing network simulation
  - Complex microservices graphs, blocking, cyclic dependencies, etc.

Power Management for Microservices

- Two types of latency slack:
  - Microservices off the critical path
  - Microservices on the critical path with relaxed QoS



Scalability Challenges



Tail at Scale Effects

- Microservices add an extra dimension to tail at scale effects
  - A single slow microservice affects end-to-end latency
  - Much more pressure on performance predictability & availability
  - Monitoring at the edge
- Determining per-tier QoS for 10000s of microservices is intractable
  - Scalable data-driven approach
- Need for online performance debugging

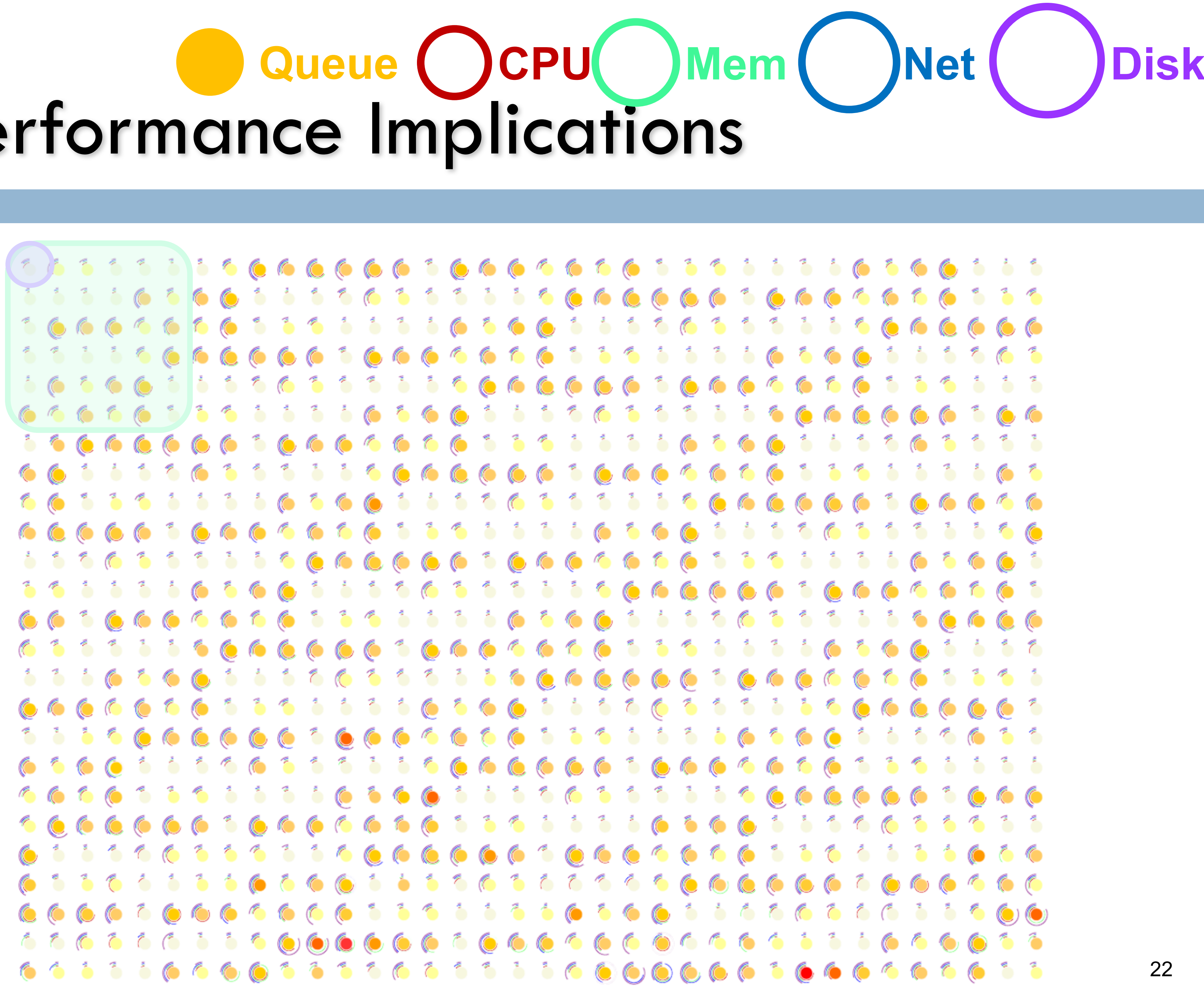
Proactive Performance Debugging

- Dependencies between microservices → propagate & amplify QoS violations
  - Finding the culprit of a QoS violation is difficult
  - Post-QoS violation, returning to nominal operation is hard

- Anticipating QoS violations & identifying culprits

- Seer: Data-driven Performance Debugging for Microservices
  - Combines lightweight RPC-level distributed tracing with hardware monitoring
  - Leverages scalable deep learning to signal QoS violations with enough slack to apply corrective action

Performance Implications



Queue CPU Mem Net Disk

Performance Implications

Seer: Data-Driven Performance Debugging

[HotCloud'18]

- Leverage the massive amount of traces collected over time

1. Apply online, practical data mining techniques that identify the culprit of an *upcoming* QoS violation

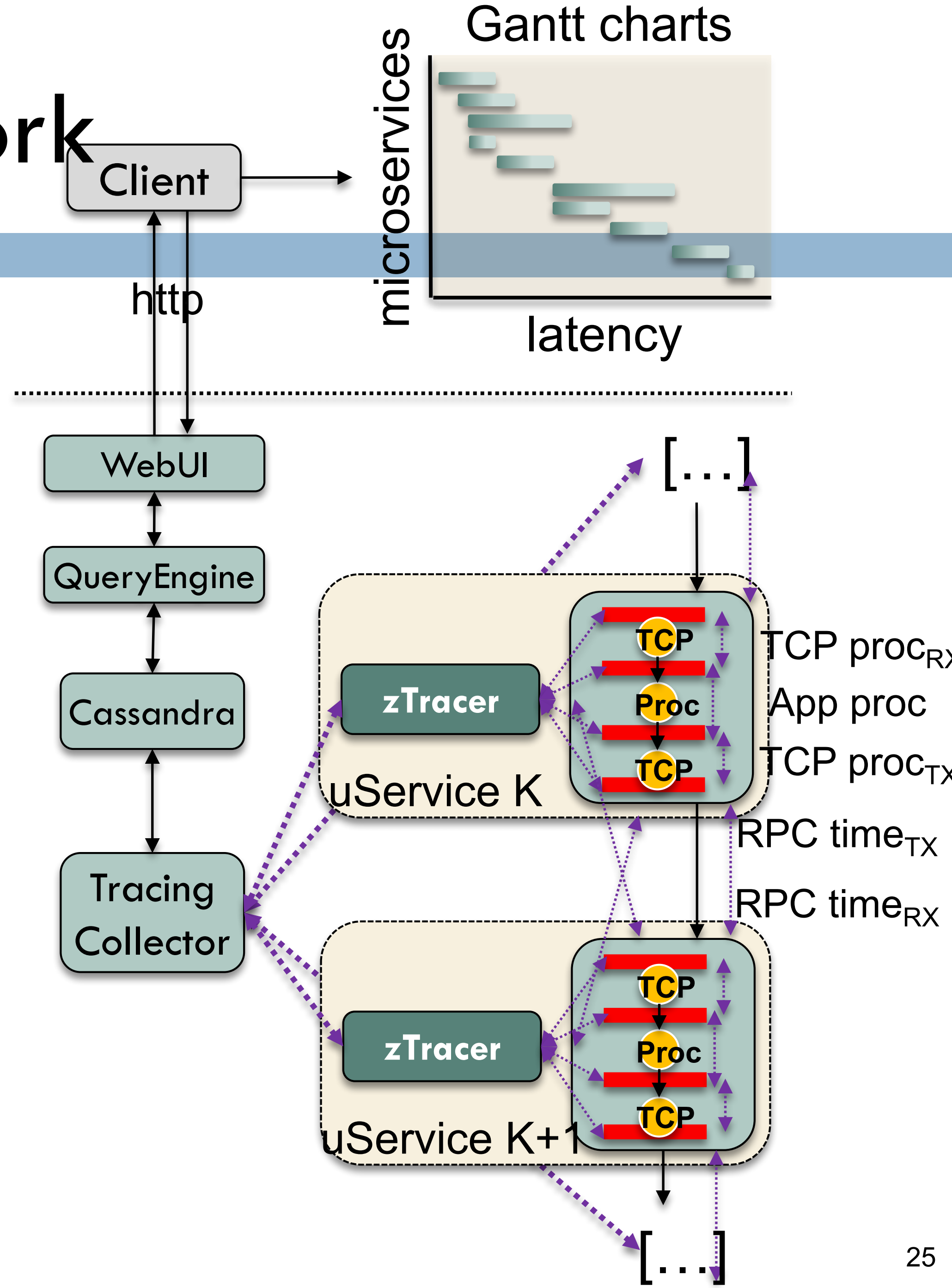
2. Use per-server hardware monitoring to determine the cause of the QoS violation

3. Take corrective action to prevent the QoS violation from occurring

- Need to predict 100s of msec – a few sec in the future

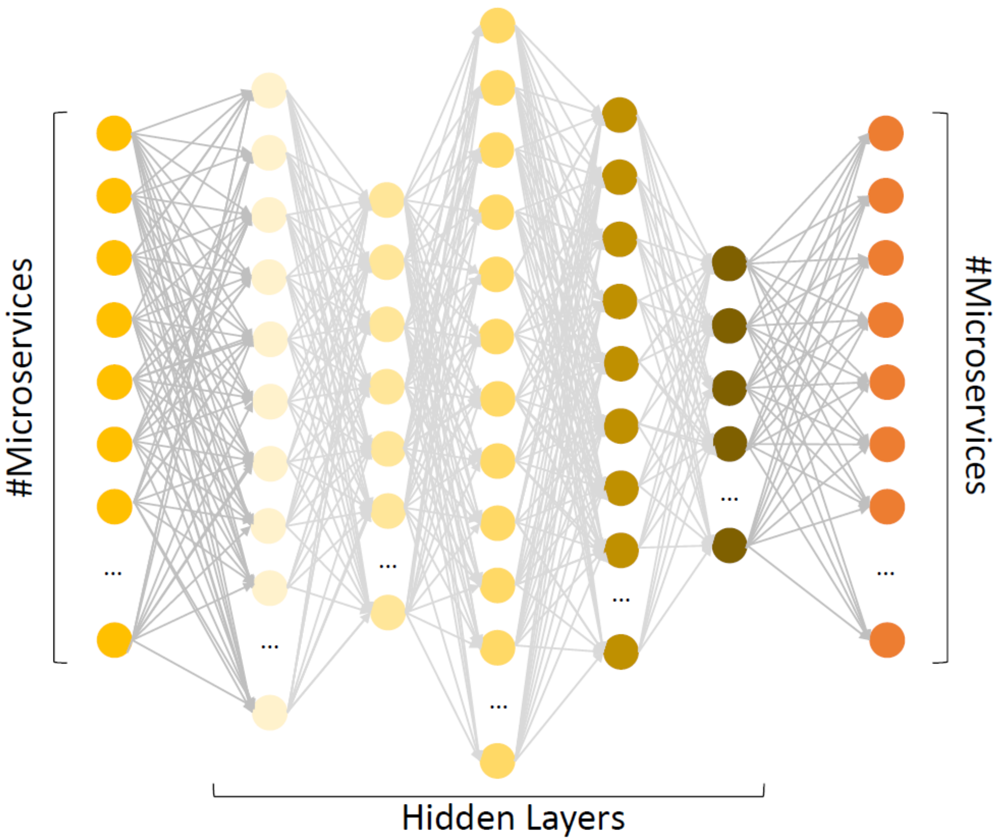
Tracing Framework

- RPC level tracing
- Based on Apache Thrift
- Timestamp start-end for each microservice
- Store in centralized DB (Cassandra)
- Record all requests → No sampling
- Overhead: <0.1% in throughput and <0.2% in tail latency



Deep Learning to the Rescue

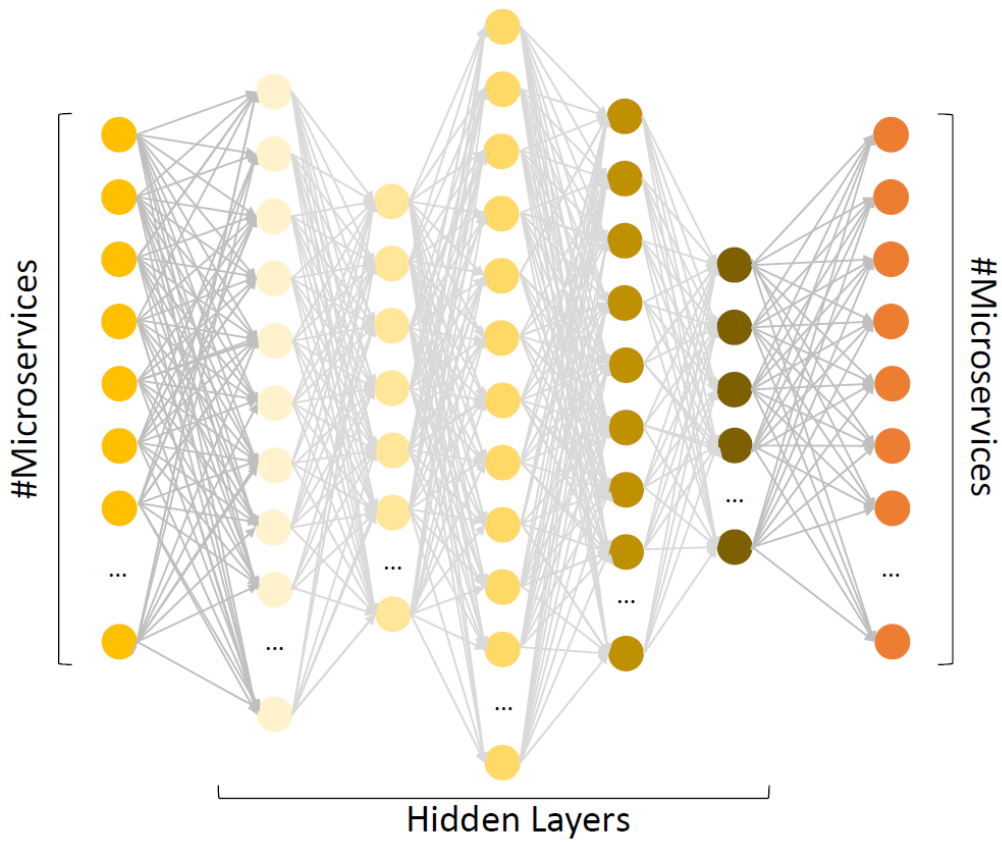
- Why?
  - Architecture-agnostic
  - Adjusts to changes in dependencies over time
  - High accuracy, good scalability
  - Inference within the required window



DNN Configuration

Input signal

- Container utilization
- Latency
- Queue depth



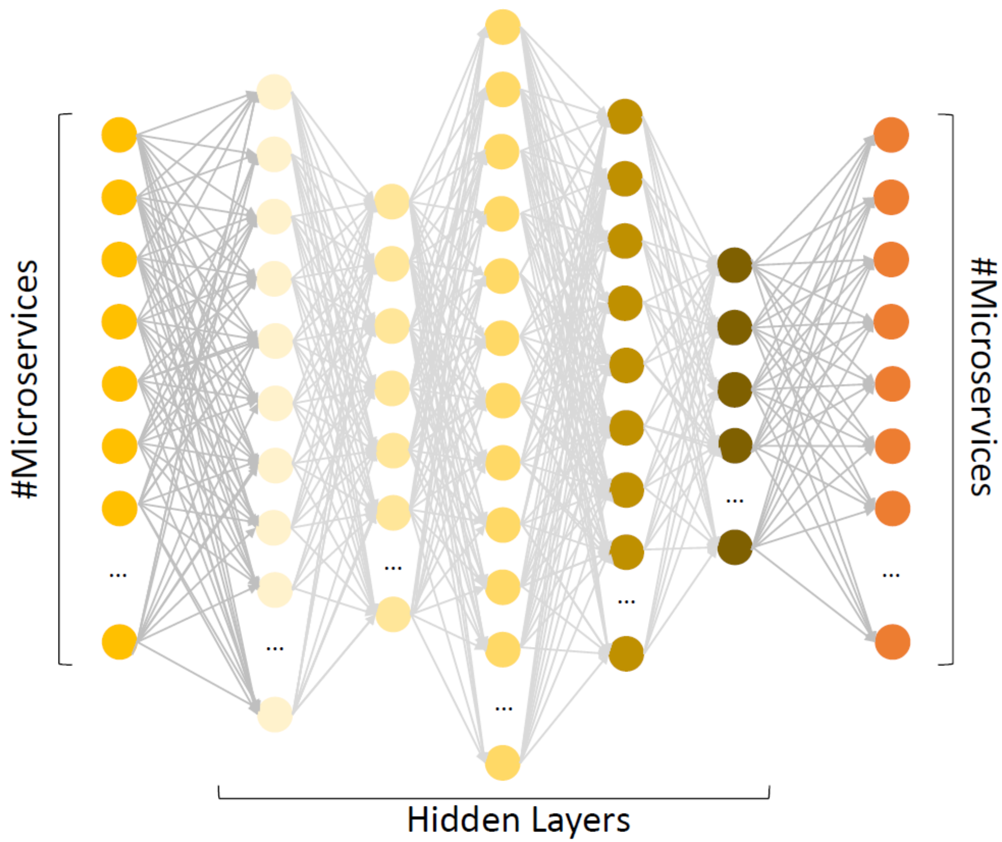
Output signal

Which microservice will cause a QoS violation in the near future?

DNN Configuration

Input signal

- Container utilization
- Latency
- Queue depth



Output signal

Which microservice will cause a QoS violation in the near future?

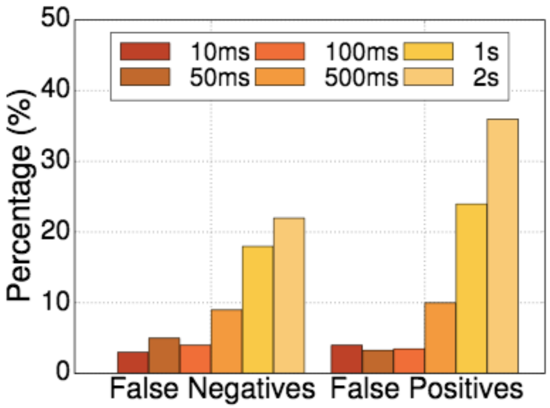
DNN Configuration

- Training once: slow (hours - days)
  - Across load levels, load distributions, request types
  - Distributed queue traces, annotated with QoS violations
  - Weight/bias inference with SGD
  - Retraining in the background

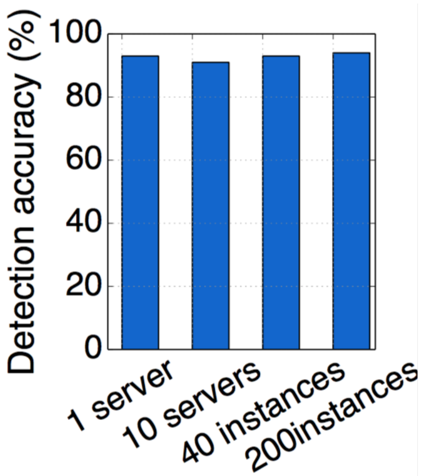
- Inference continuously: streaming trace data

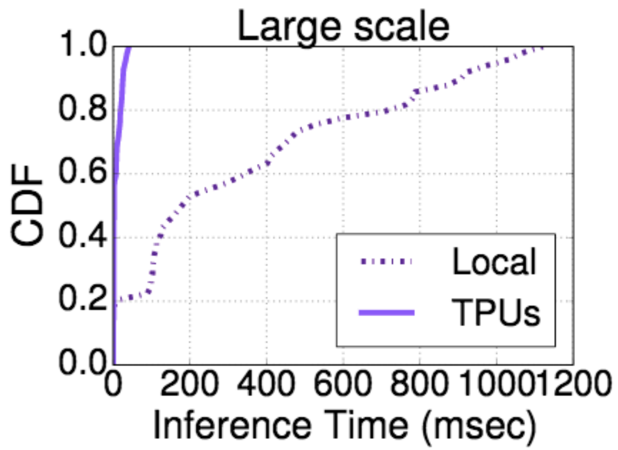
93% accuracy in signaling upcoming QoS violations

91% accuracy in attributing QoS violation to correct microservice



DNN Configuration



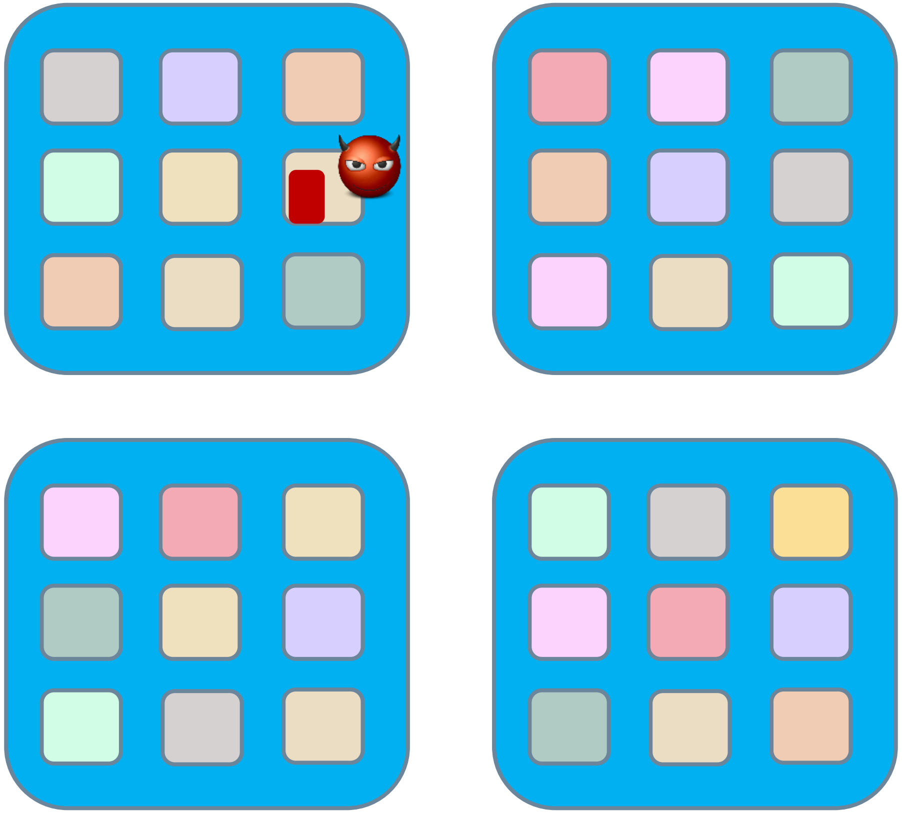


Accuracy stable or increasing with cluster size

- Challenges:
  - In large clusters inference too slow to prevent QoS violations
  - Offload on TPUs, 10-100x improvement; 10ms for 90th %ile inference
  - Fast enough for most corrective actions to take effect (net bw partitioning, RAPL, cache partitioning, scale-up/out, etc.)

Experimental Setup

- 40 dedicated servers
- ~1000 single-concerned containers
- Machine utilization 80-85%
- Inject interference to cause QoS violation
  - Using microbenchmarks (CPU, cache, memory, network, disk I/O)



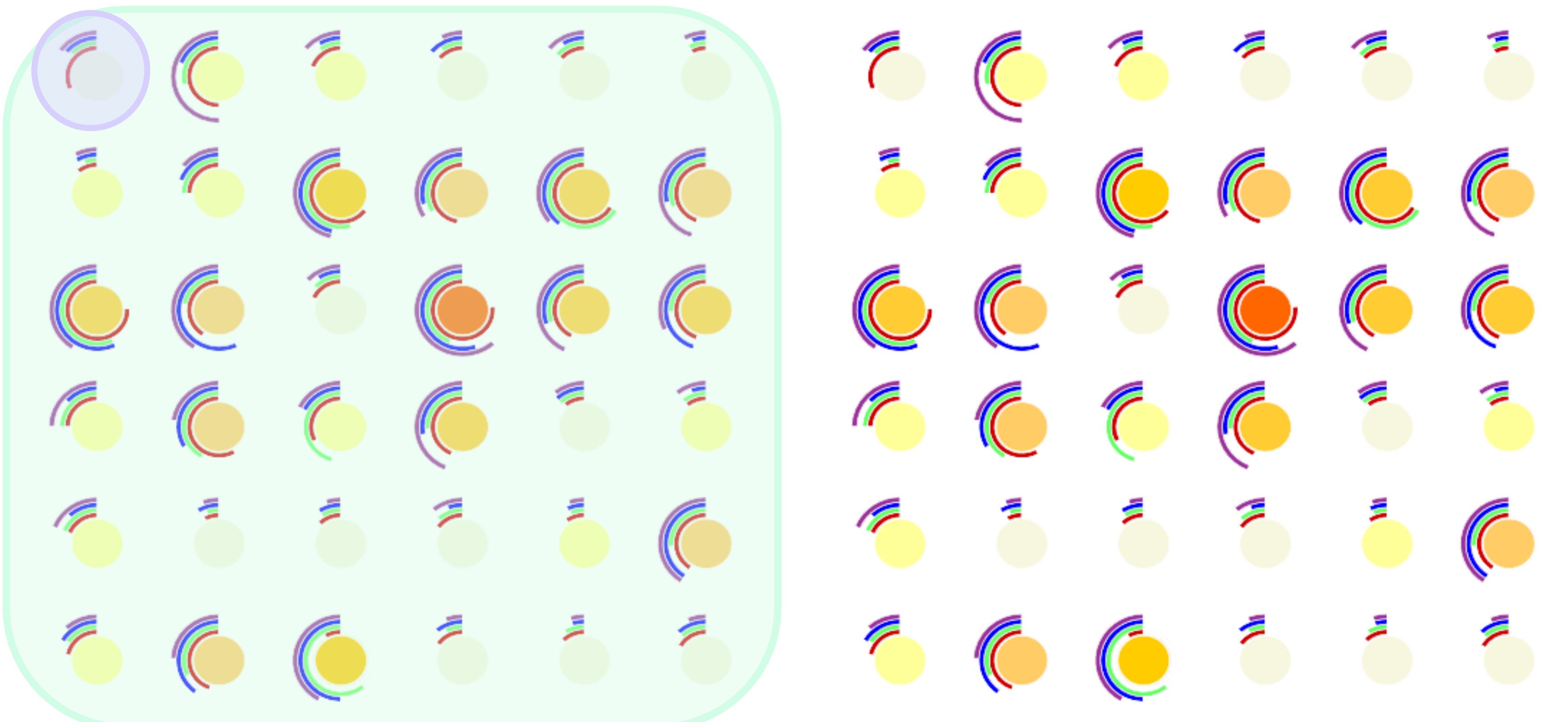
Restoring QoS

- 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

Demo

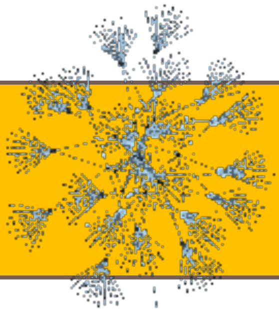
Queue CPU Mem Net Disk

Seer Default

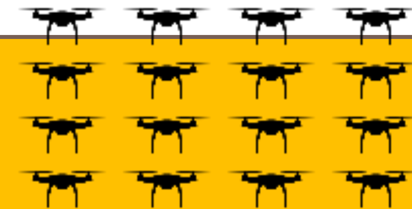


Demo

A New Cloud Stack



λ Applications



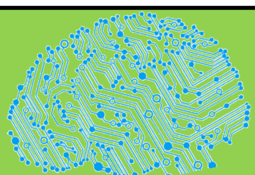
CAL’18a

Serverless

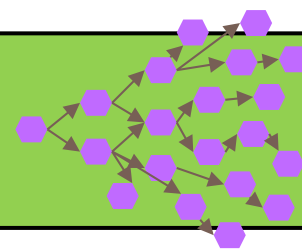
Programming frameworks

fission

in submission



Cluster management



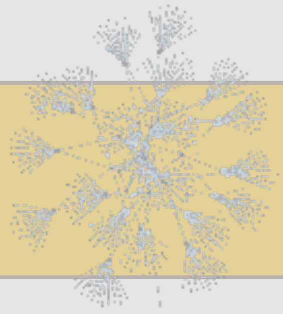
CAL’18b, HotCloud’18, in submission

Hardware design

Scalable isolation

in submission

A New Cloud Stack



Applications

CAL’18a

Serverless

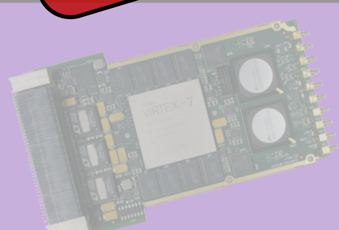
Programming frameworks

in submission

Data-driven approaches can help navigate the increasing complexity of the cloud

Cluster management

CAL’18b, HotCloud’18, in submission



Hardware design

Scalable isolation

Thank you!

in submission