MXFaaS: Resource Sharing in Serverless Environments for Parallelism and Efficiency

ISCA 2023

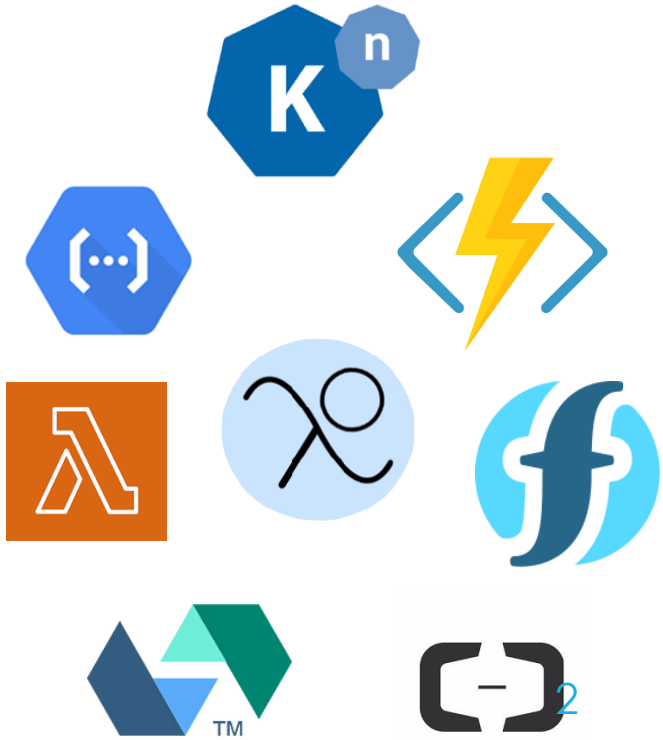
Jovan Stojkovic, Tianyin Xu, Hubertus Franke*, Josep Torrellas

University of Illinois at Urbana-Champaign

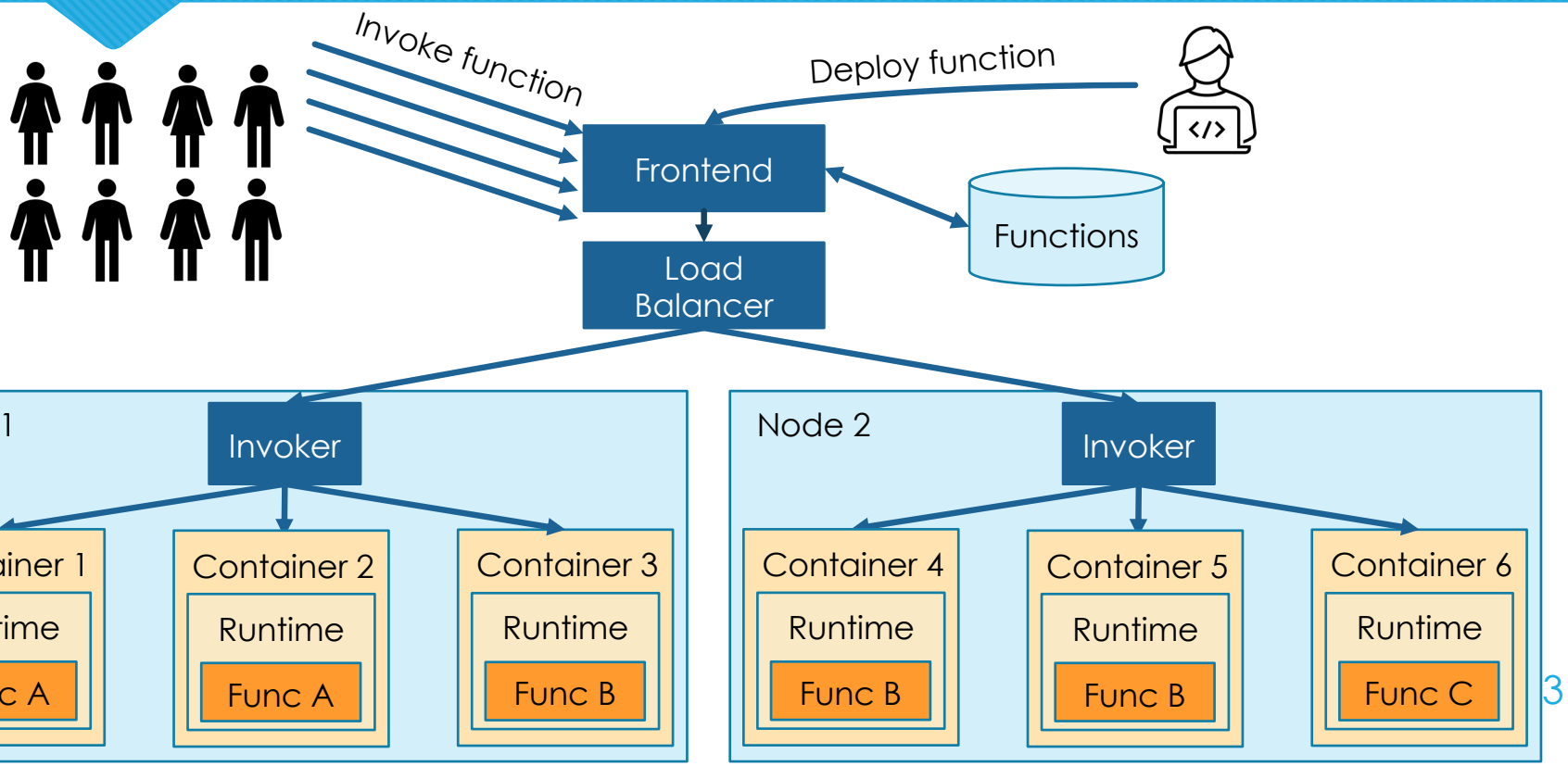
*IBM Research

What is serverless computing?

- Serverless computing is a popular cloud paradigm
  - Users deploy applications, providers provision resources
- Many benefits
  - Simple and modular programming
  - Automatic resource scaling
  - Pay-as-you-go model
- AWS Lambda, Microsoft Azure, Google Cloud, IBM Cloud



How serverless computing works?

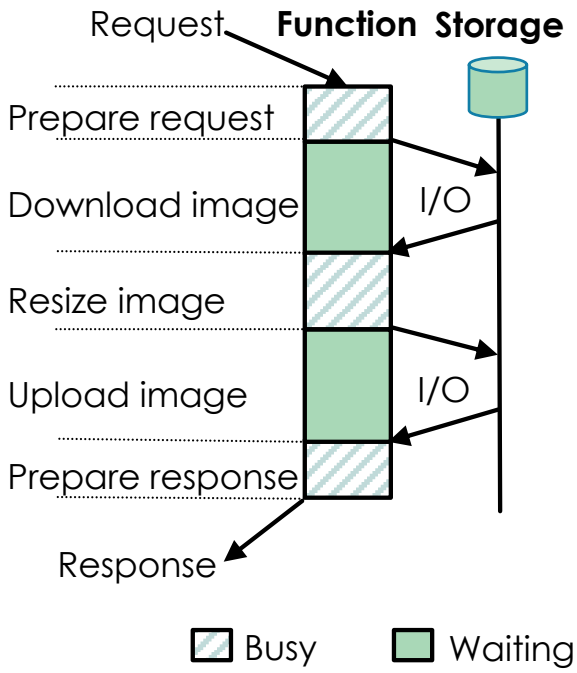


Contributions

- Architectural characterization of serverless environments
- Propose **MXFaaS** – novel serverless platform based on resource sharing/multiplexing across function invocations
  - Efficient use of processor cycles, I/O bandwidth, and memory state
- Average speedup 5.2X, tail latency reduction 7.4X, throughput improvement 4.8X

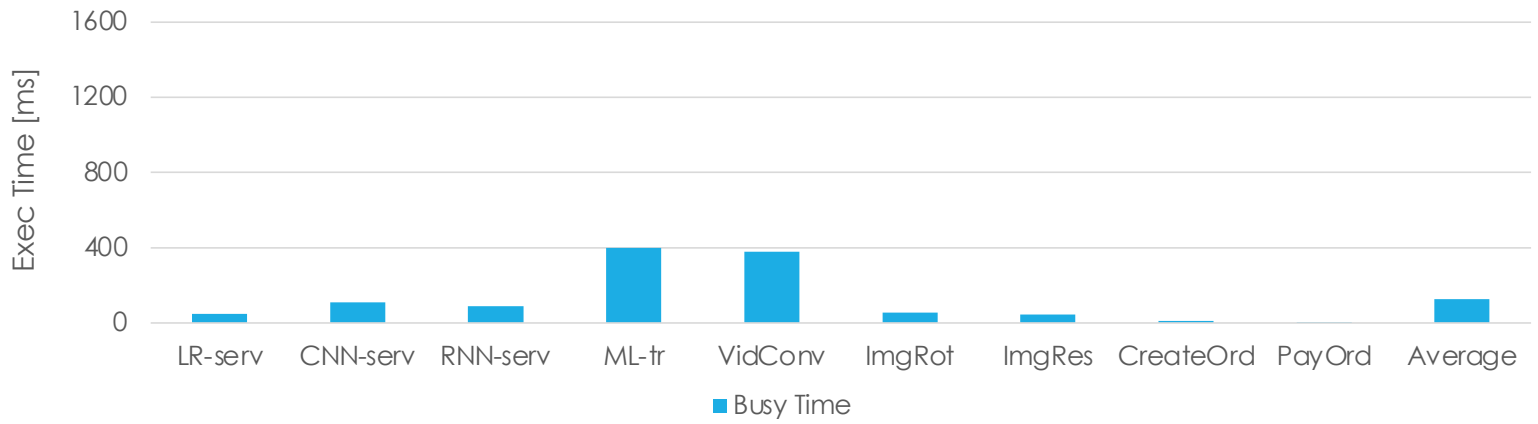
Idle Time Dominates Function Execution

- Functions synchronously accessing remote storage



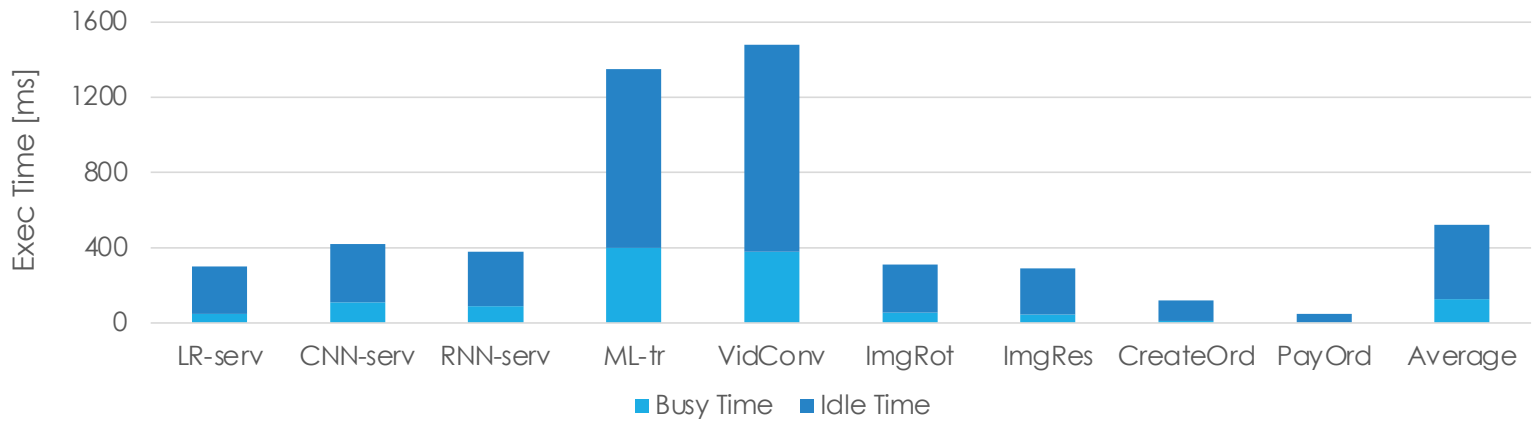
Idle Time Dominates Function Execution

- Functions synchronously accessing remote storage



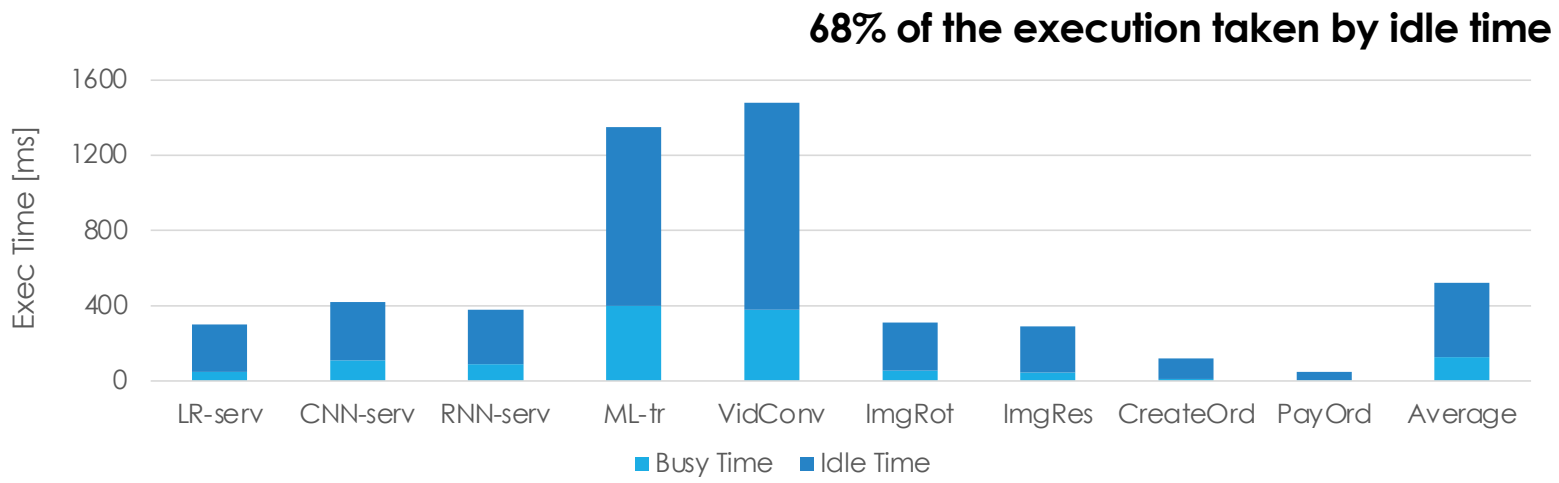
Idle Time Dominates Function Execution

- Functions synchronously accessing remote storage



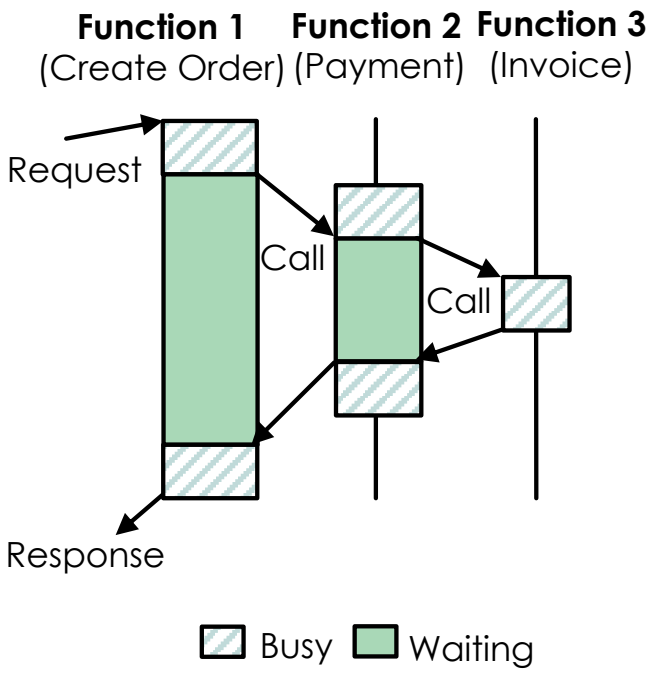
Idle Time Dominates Function Execution

- Functions synchronously accessing remote storage



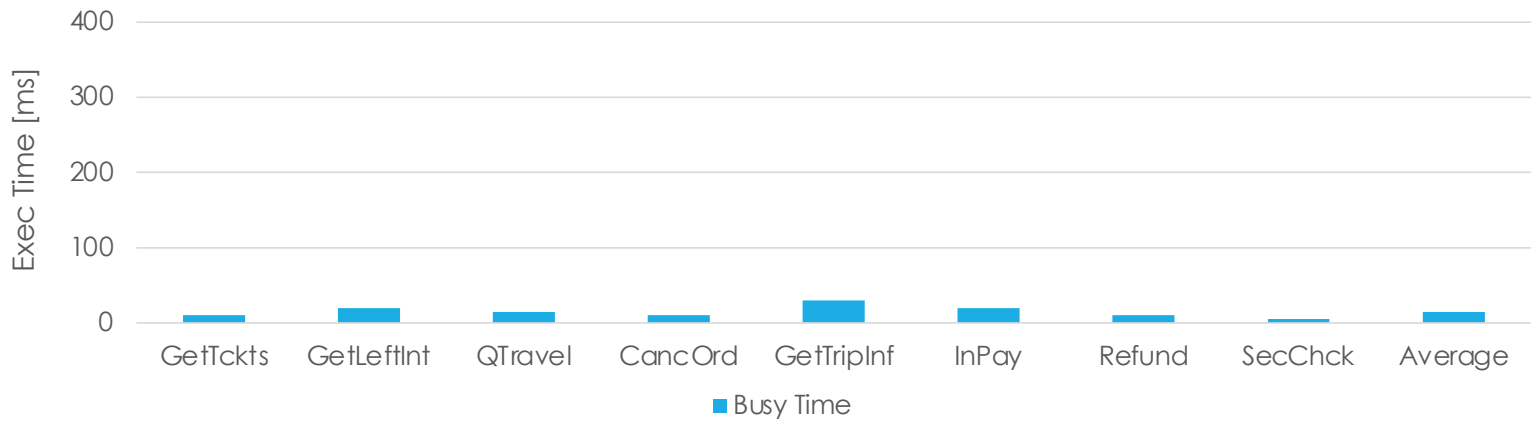
Idle Time Dominates Function Execution

- Functions synchronously invoking other functions



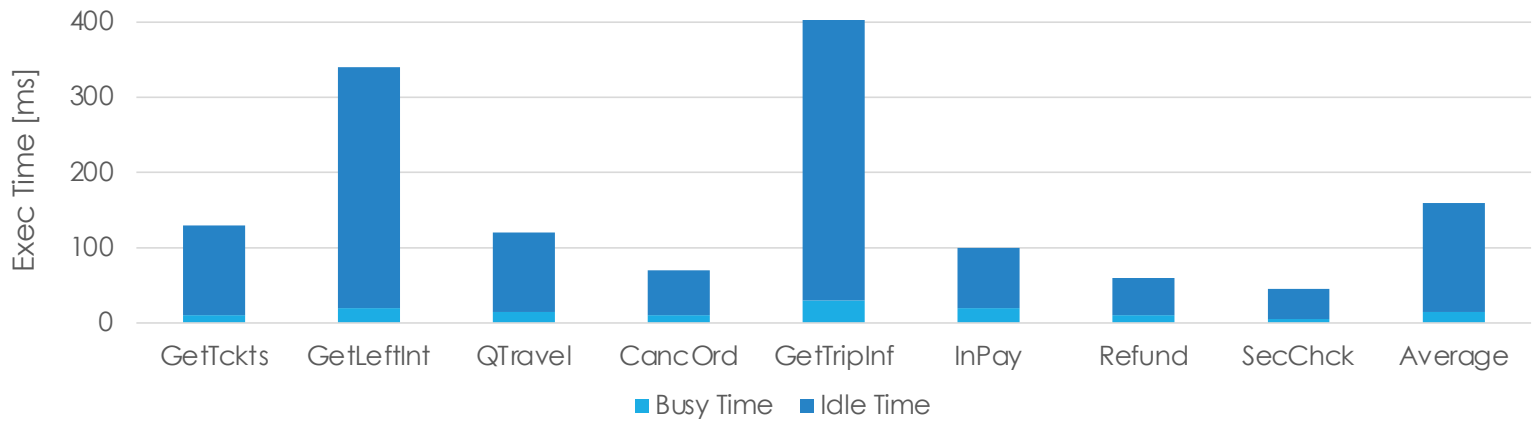
Idle Time Dominates Function Execution

- Functions synchronously invoking other functions



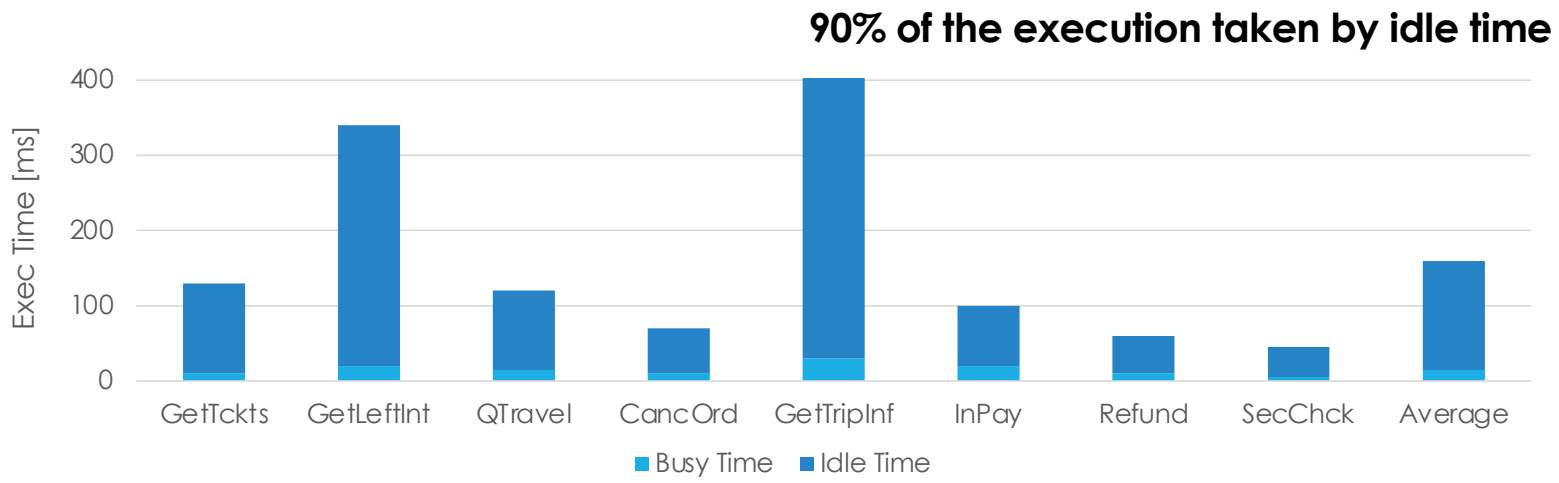
Idle Time Dominates Function Execution

- Functions synchronously invoking other functions



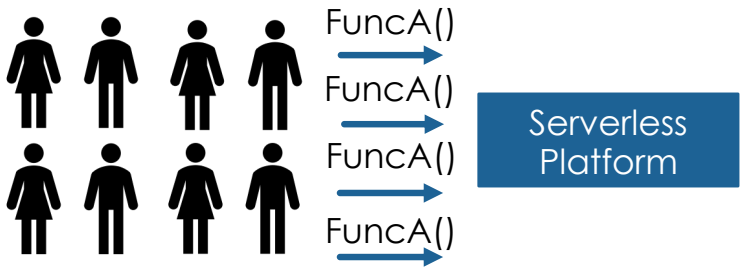
Idle Time Dominates Function Execution

- Functions synchronously invoking other functions



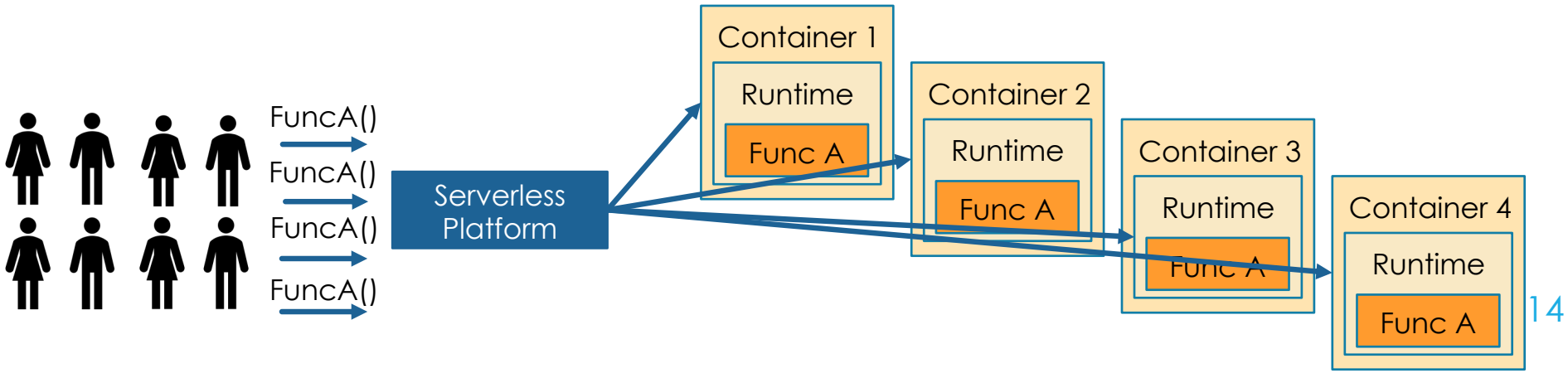
Same-Function Invocations are Bursty

- Serverless computing model promotes autoscaling and request-level parallelism



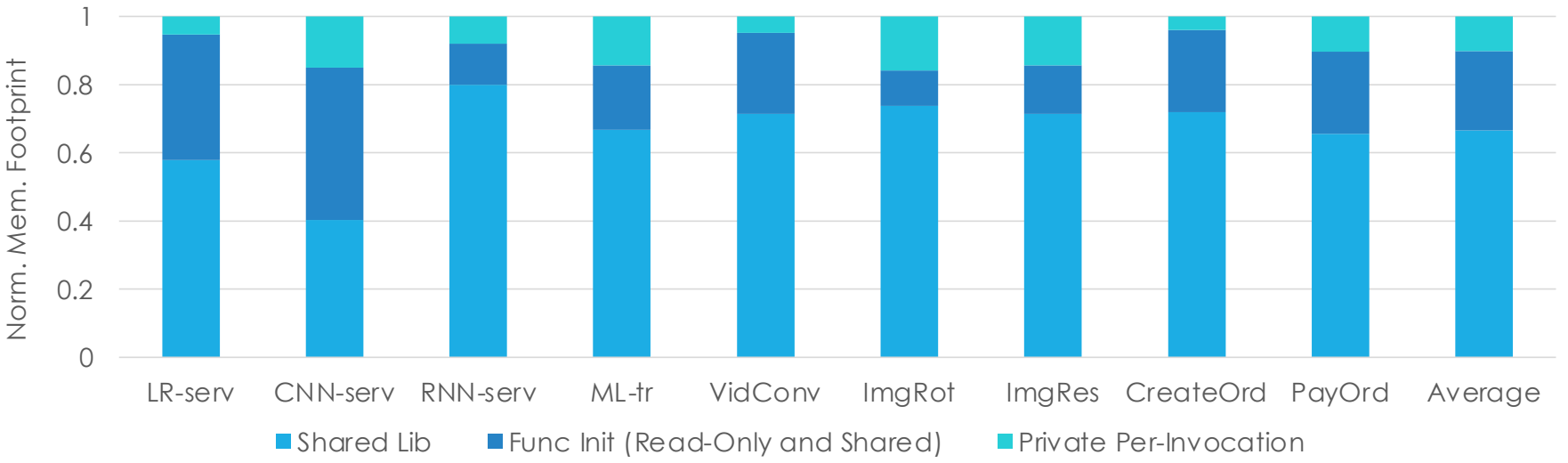
Same-Function Invocations are Bursty

- Current systems handle burstiness inefficiently and spawn many concurrent containers



Implications of Bursty Function Invocations: 1. In-memory State Replication

- Same-function invocations frequently access the same data and instructions
- Shared libraries and function specific read-only data are replicated in memory



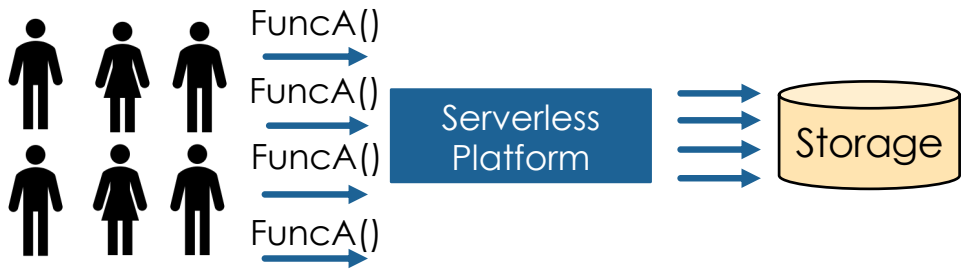
Implications of Bursty Function Invocations: 2. Pressure on Remote Storage

- Same-function invocations execute the same code

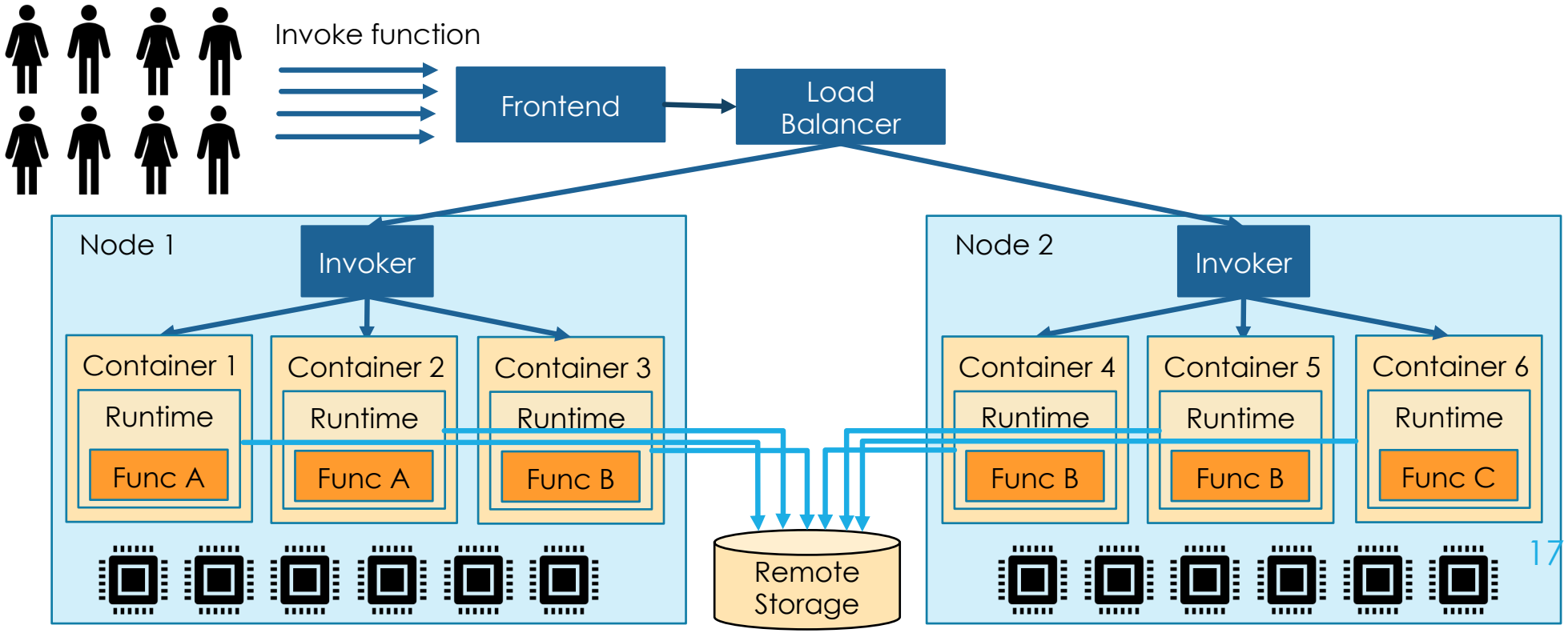
  → access the same storage area for same/similar data

- Invocations bursty

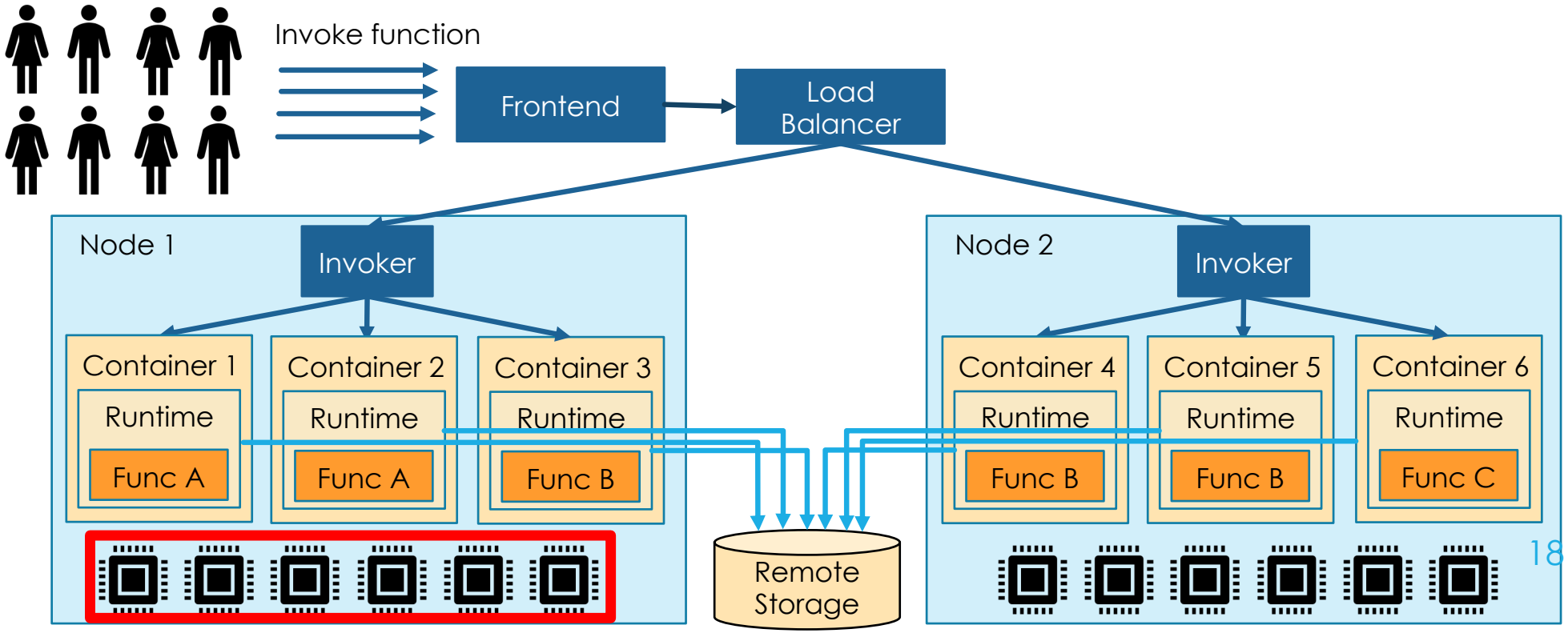
  → concurrent requests to the remote storage



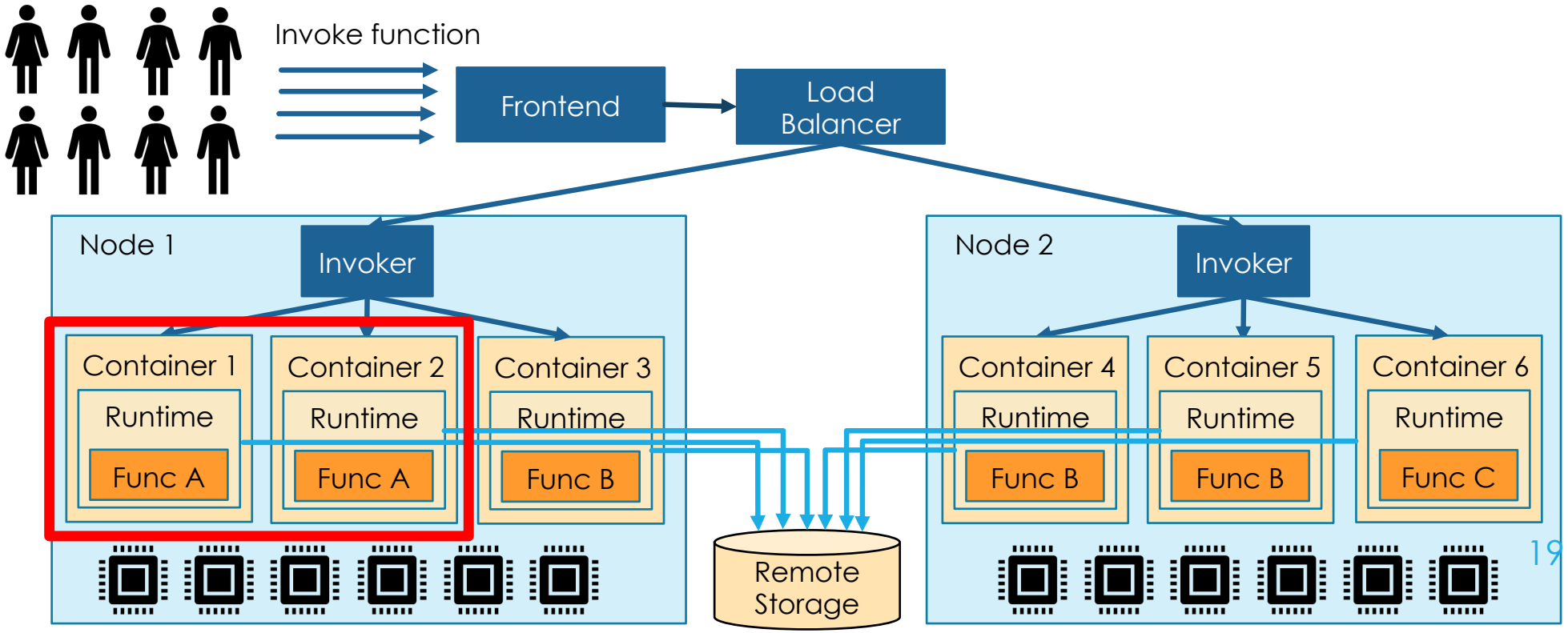
Conventional systems are not optimized for bursty invocation patterns



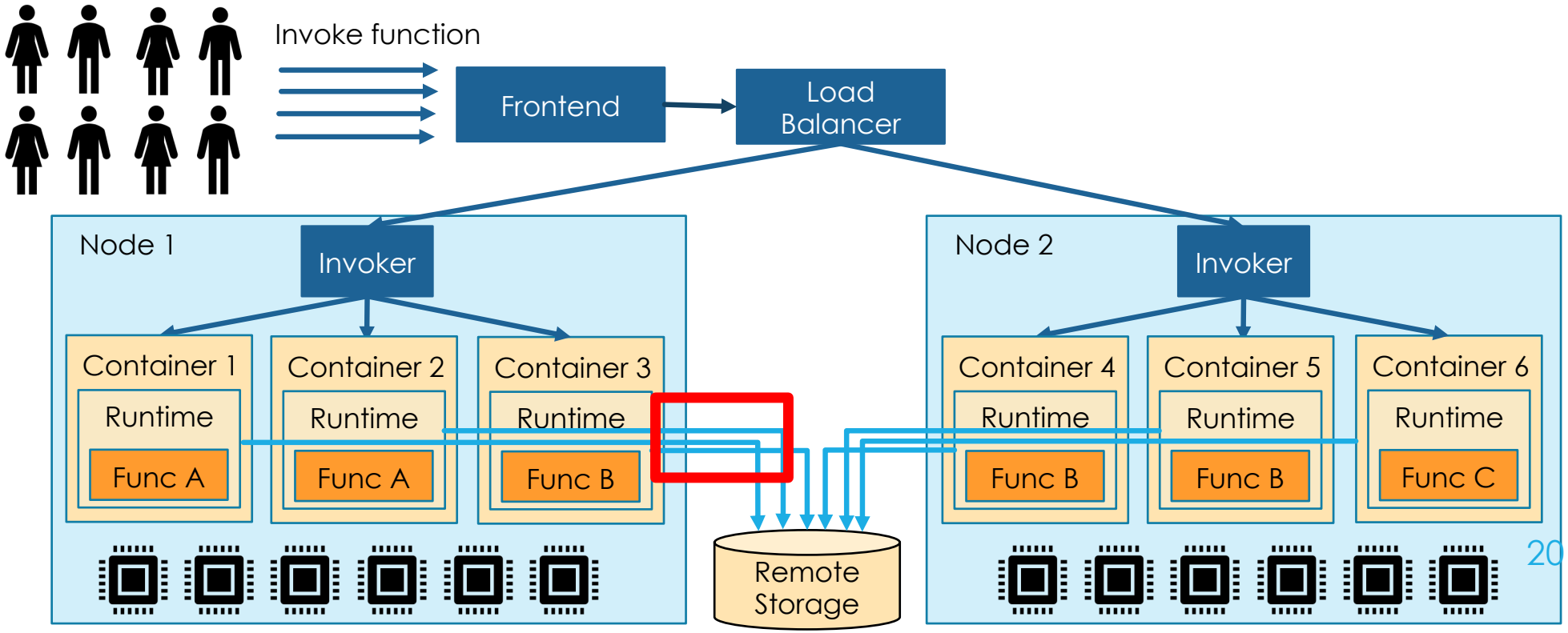
Conventional systems are not optimized for bursty invocation patterns



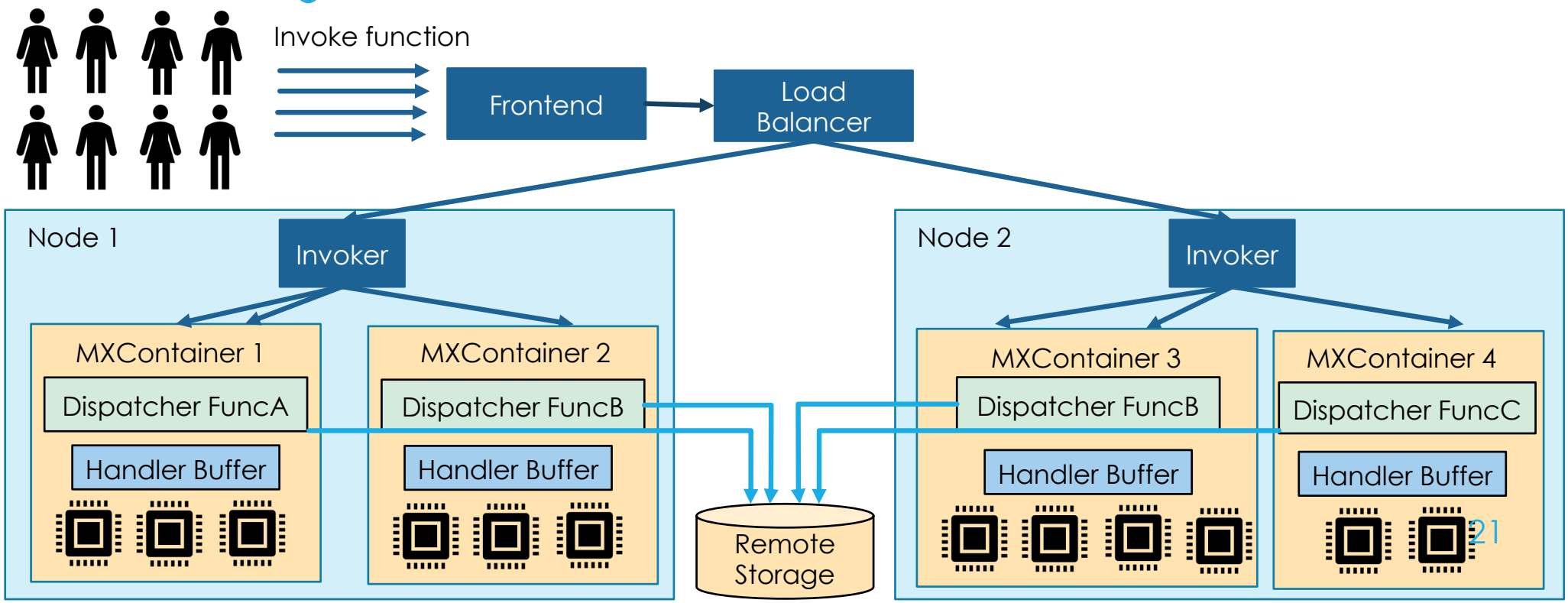
Conventional systems are not optimized for bursty invocation patterns



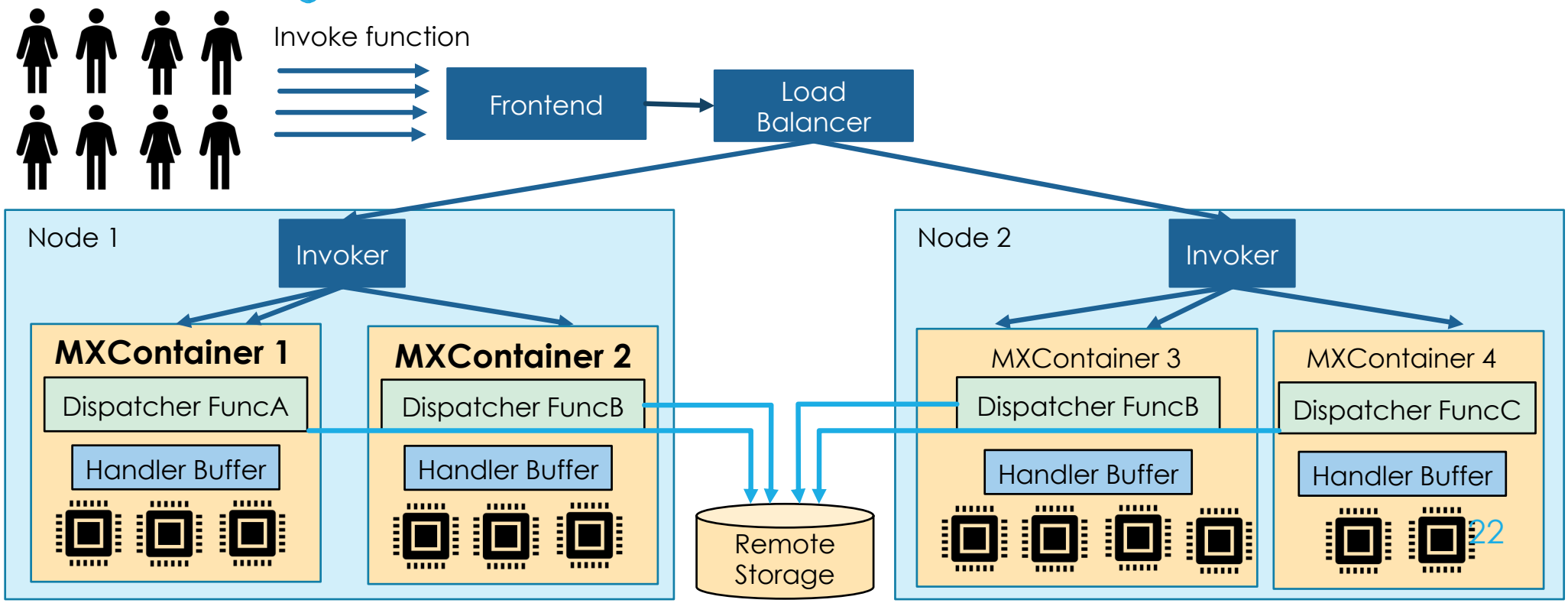
Conventional systems are not optimized for bursty invocation patterns



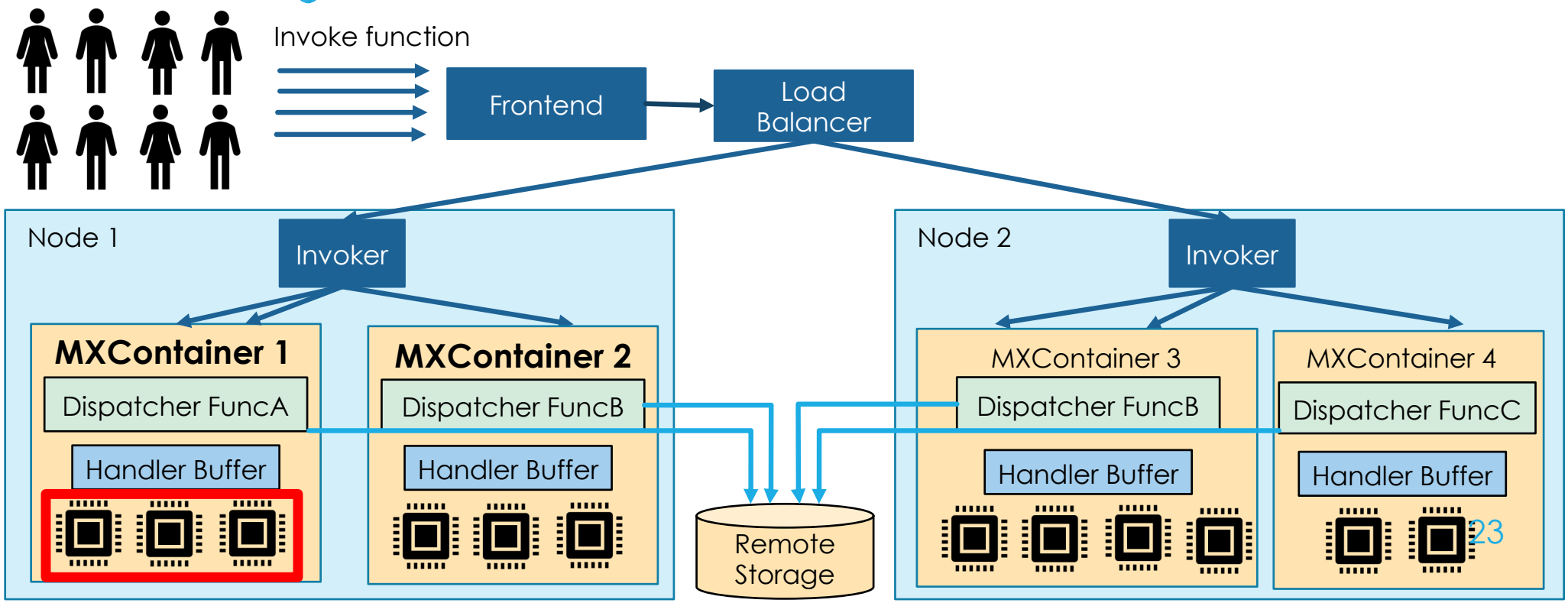
MXFaaS: Resource Sharing for Parallelism and Efficiency



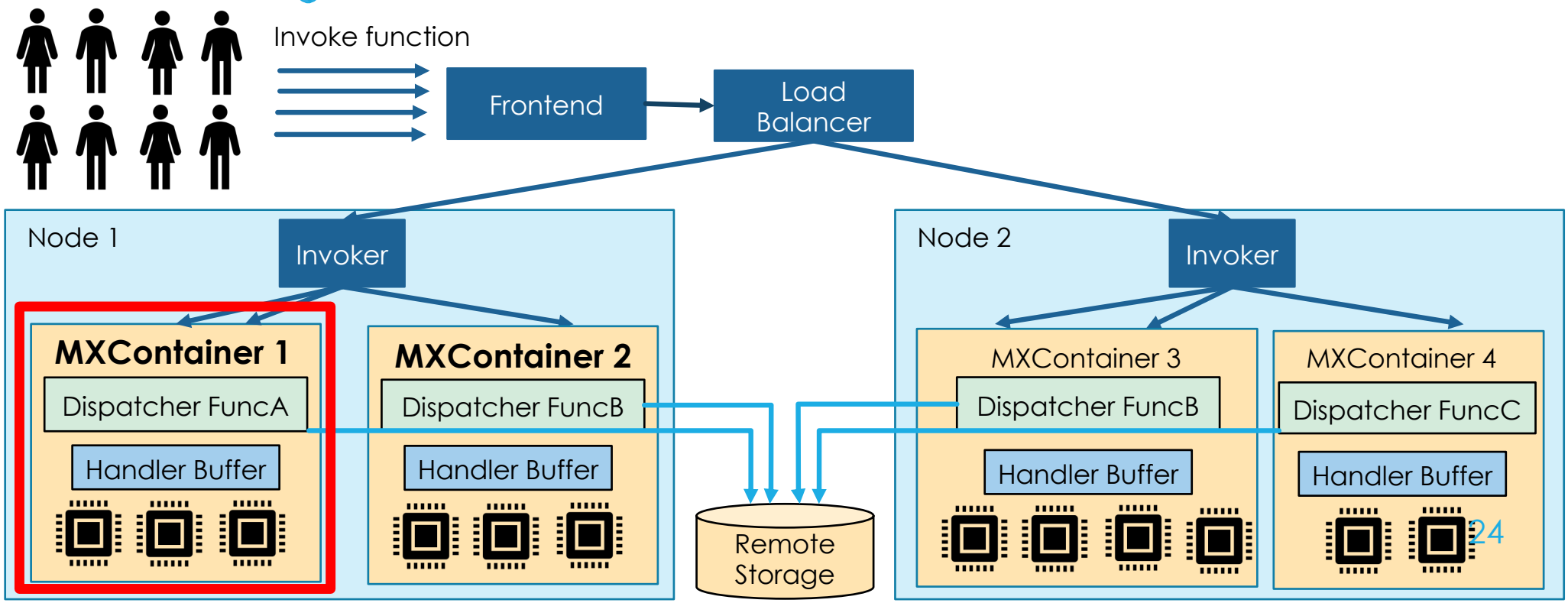
MXFaaS: Resource Sharing for Parallelism and Efficiency



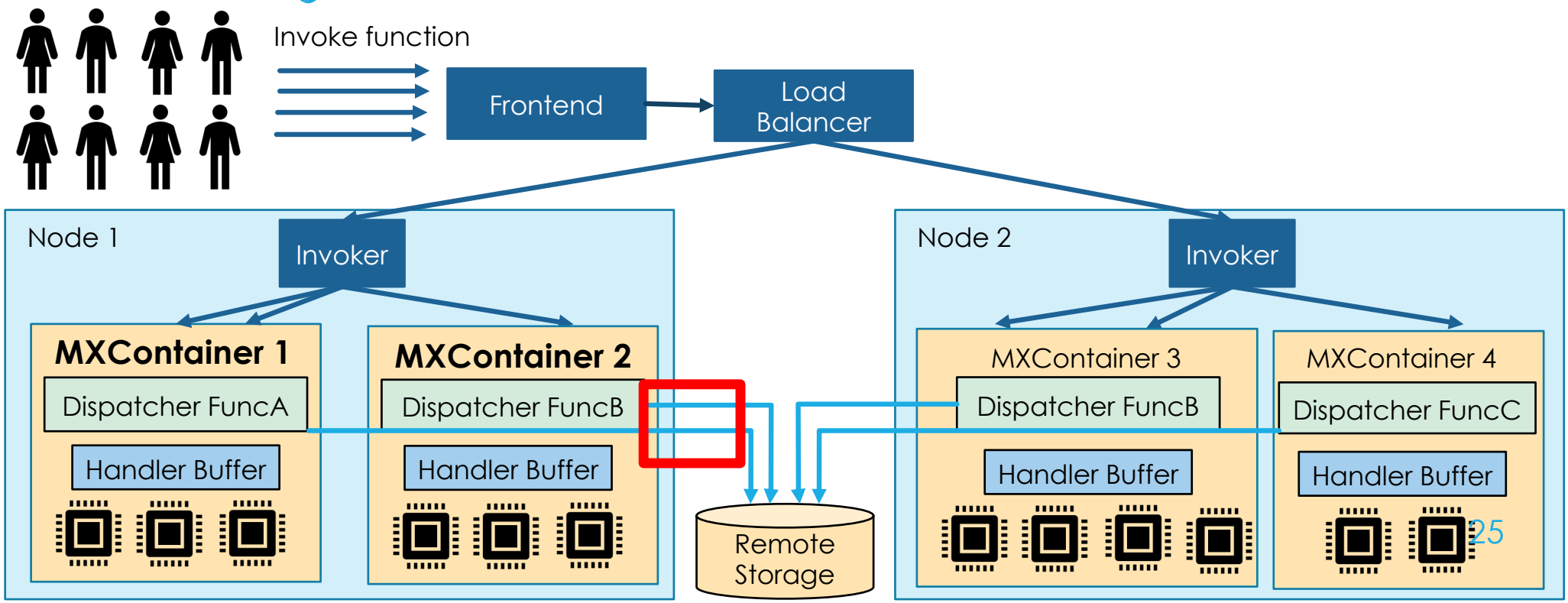
MXFaaS: Resource Sharing for Parallelism and Efficiency



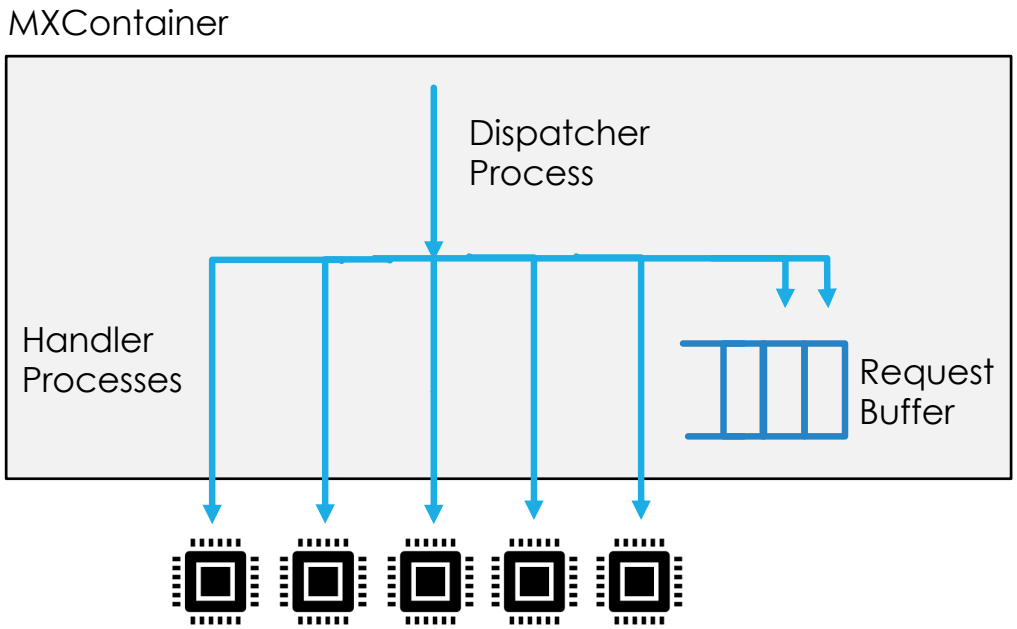
MXFaaS: Resource Sharing for Parallelism and Efficiency



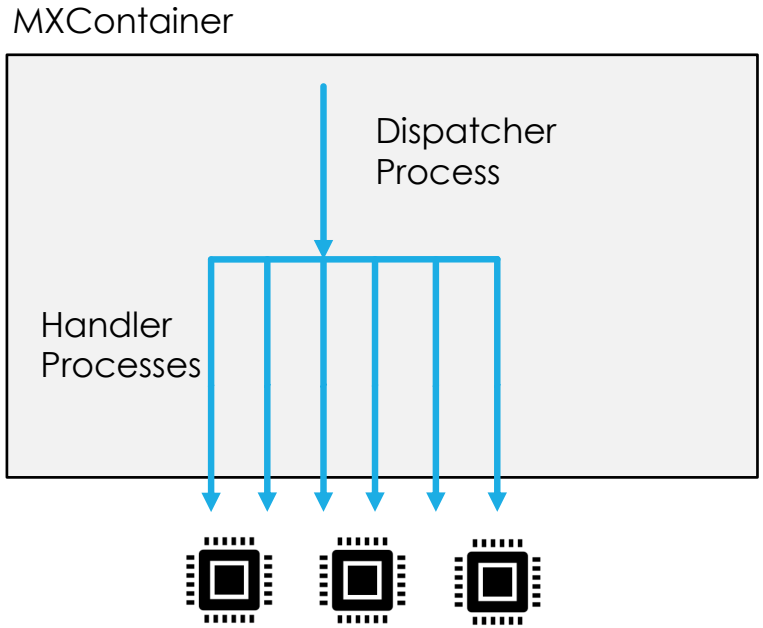
MXFaaS: Resource Sharing for Parallelism and Efficiency



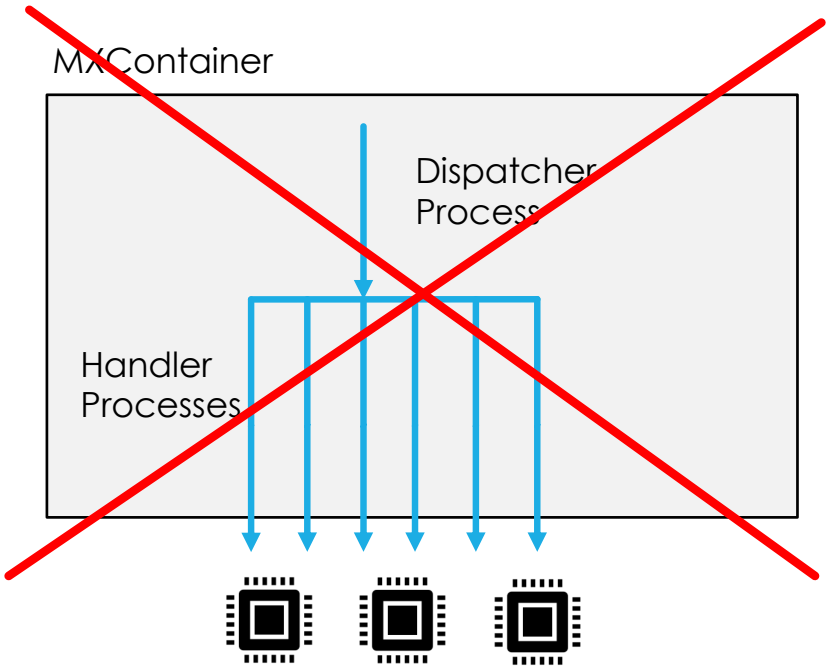
Structure of an MXContainer



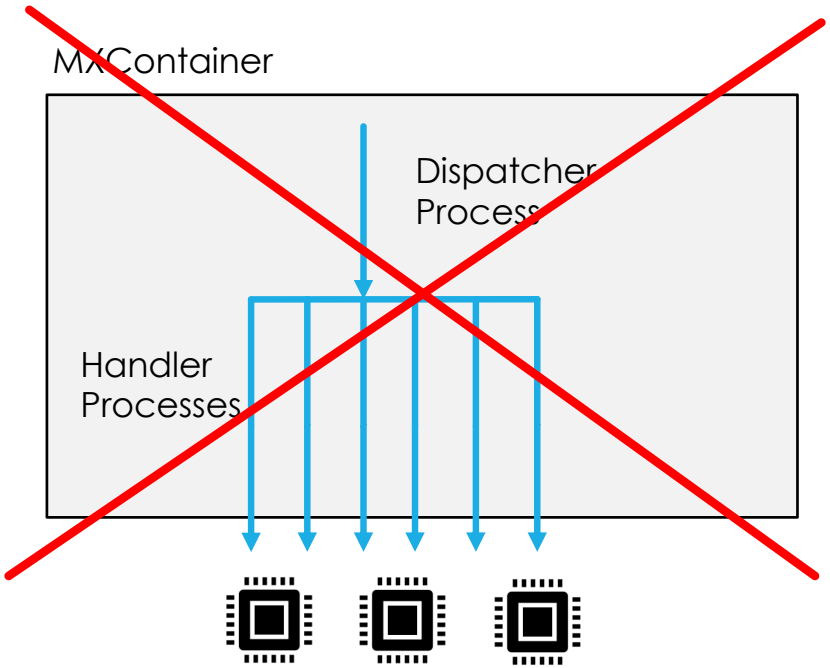
Naïve Request Scheduling

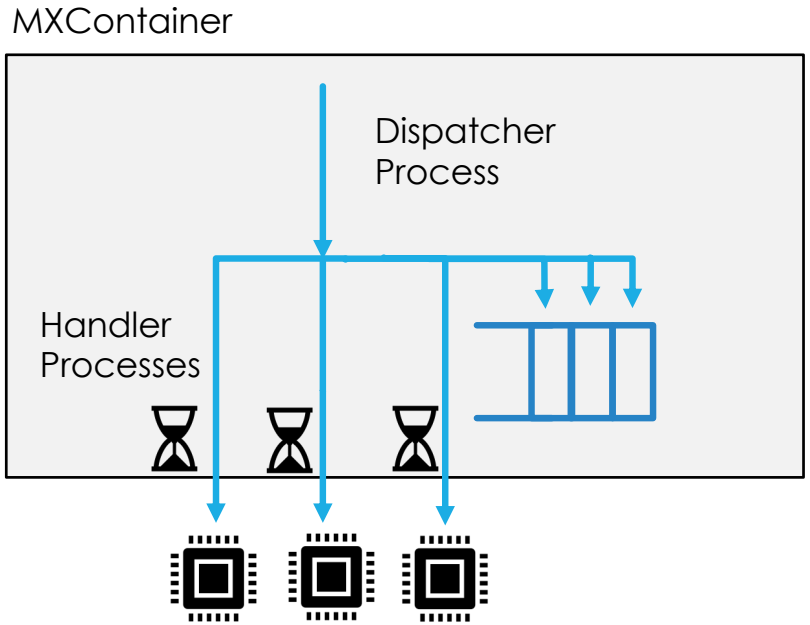


Naïve Request Scheduling

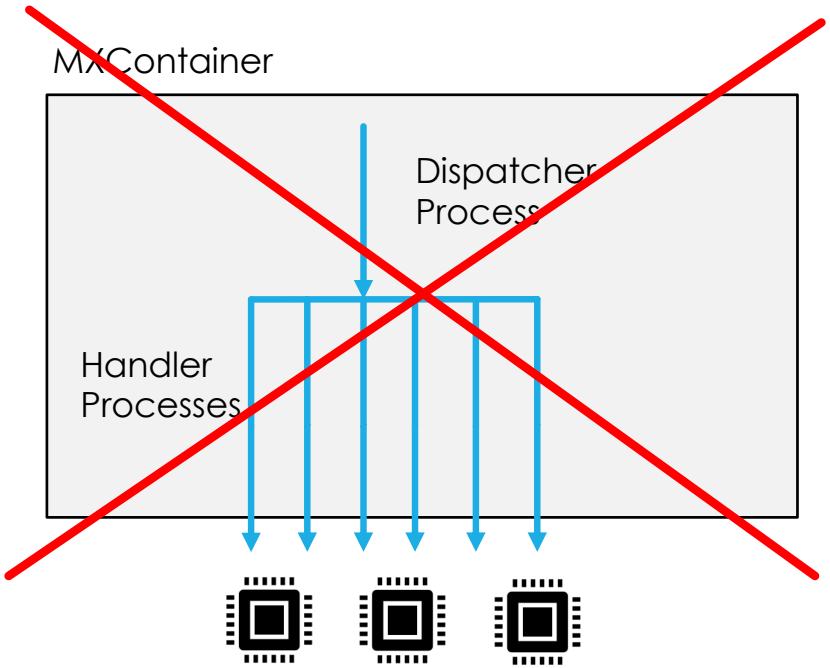


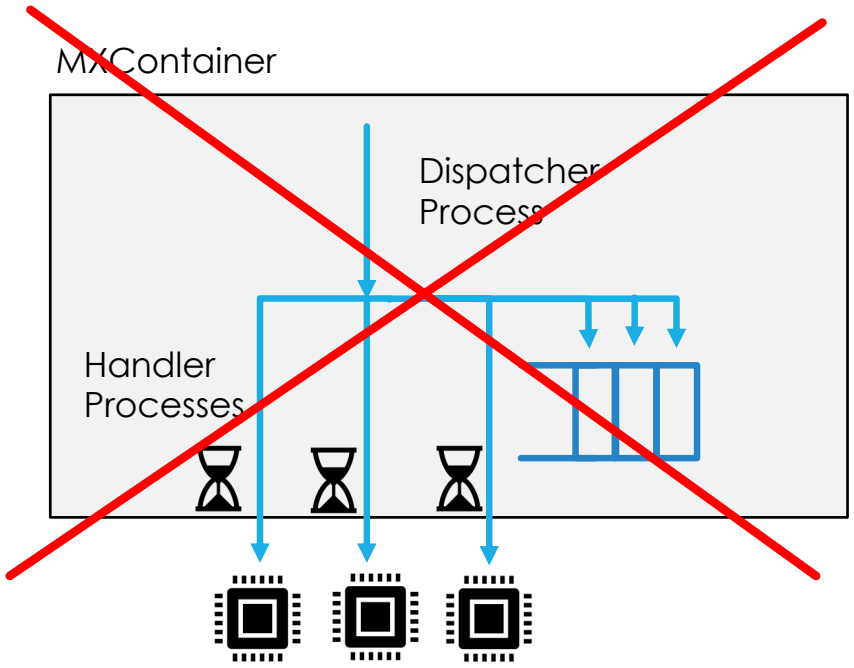
Naïve Request Scheduling





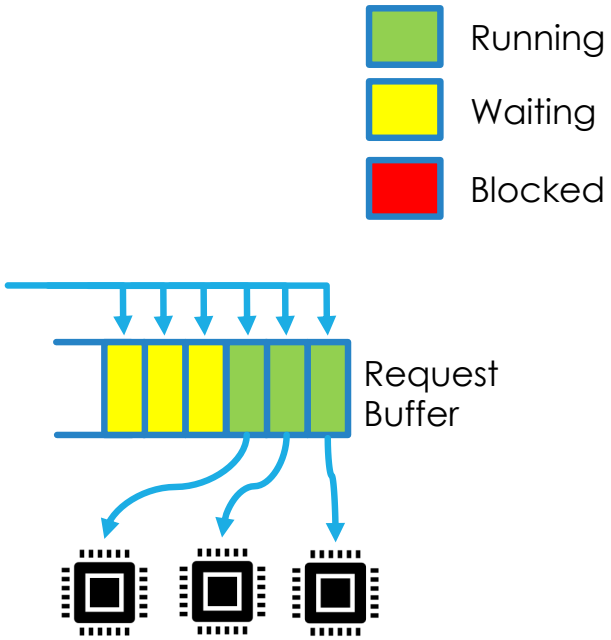
Naïve Request Scheduling





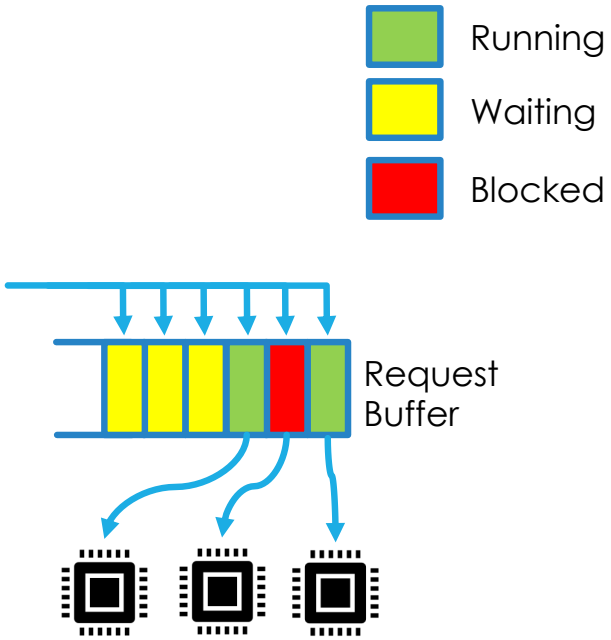
Sharing CPU Cycles with MXContainers

- Multiplexing CPU with ***smart handler scheduling***



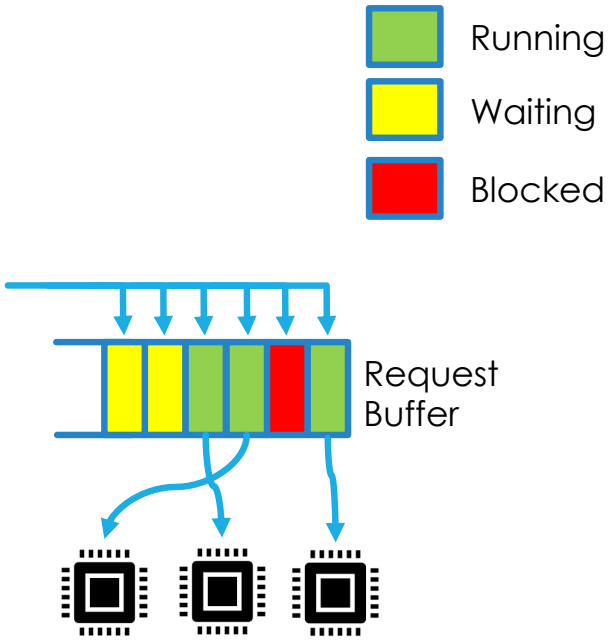
Sharing CPU Cycles with MXContainers

- Multiplexing CPU with ***smart handler scheduling***



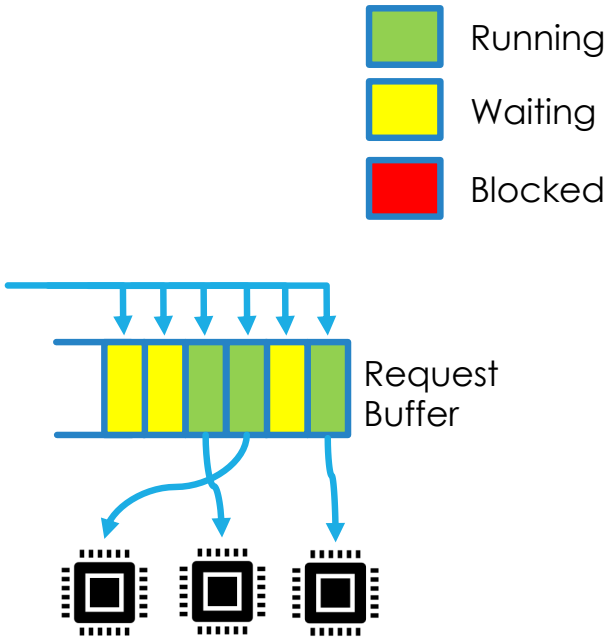
Sharing CPU Cycles with MXContainers

- Multiplexing CPU with ***smart handler scheduling***



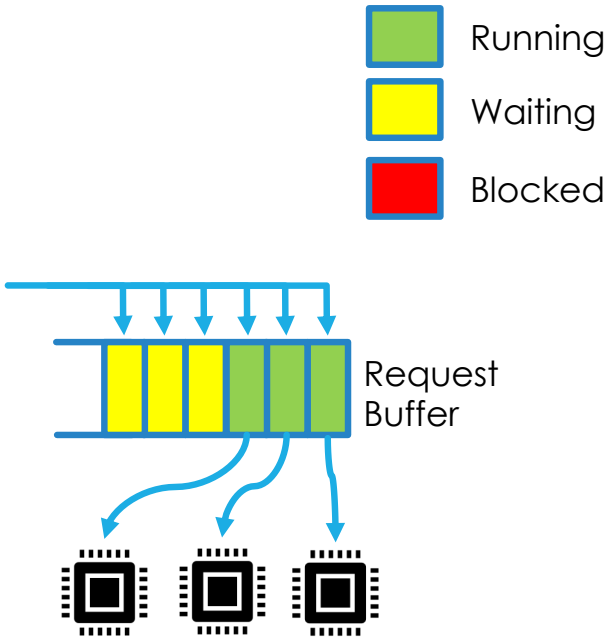
Sharing CPU Cycles with MXContainers

- Multiplexing CPU with ***smart handler scheduling***



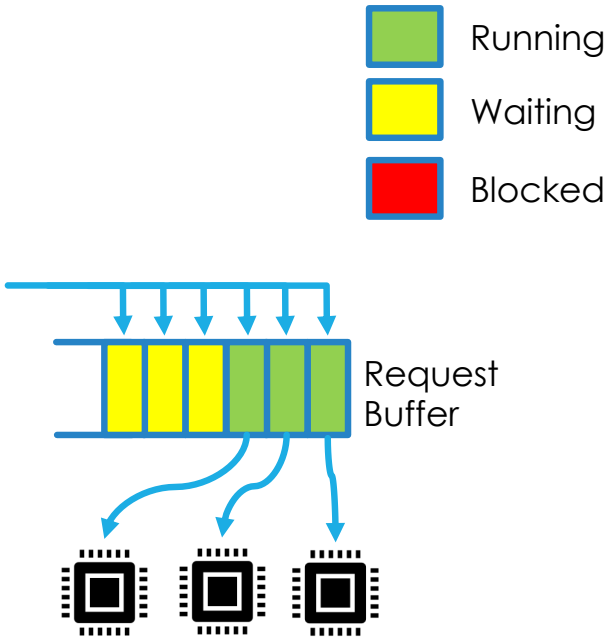
Sharing CPU Cycles with MXContainers

- Multiplexing CPU with ***smart handler scheduling***



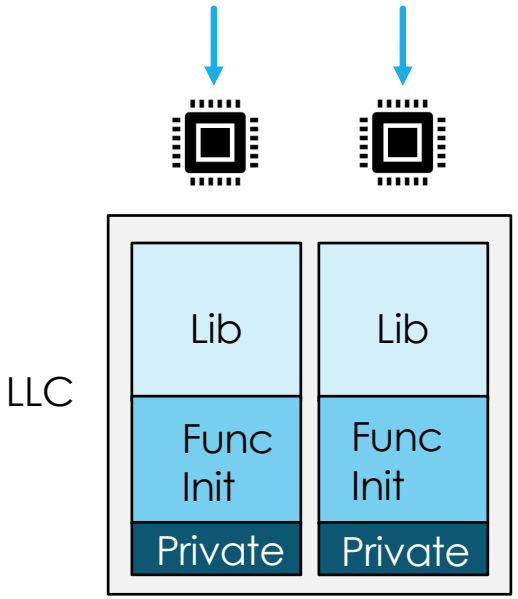
Sharing CPU Cycles with MXContainers

- Multiplexing CPU with ***smart handler scheduling***
  - ➔ High utilization of CPU cycles + low tail latency



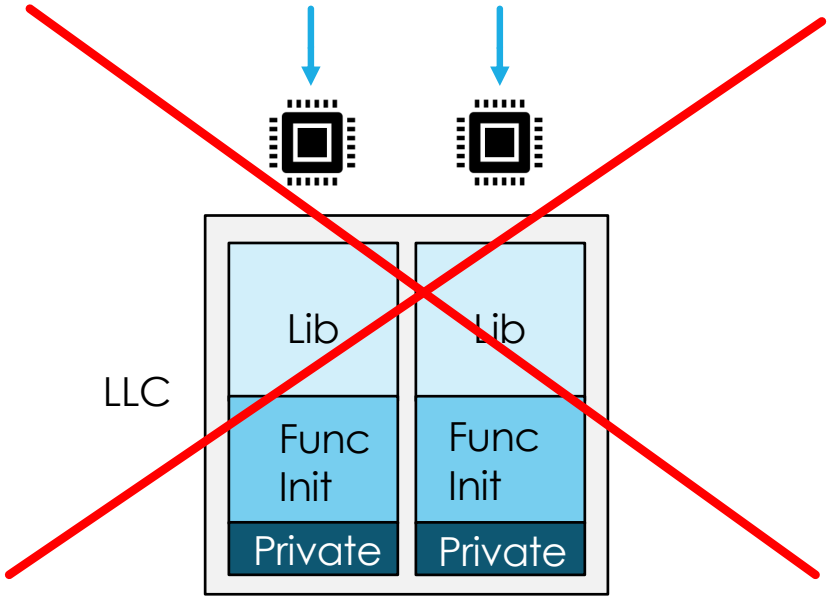
Sharing Memory and Processor State with MXContainers

- Conventional: replicated memory state



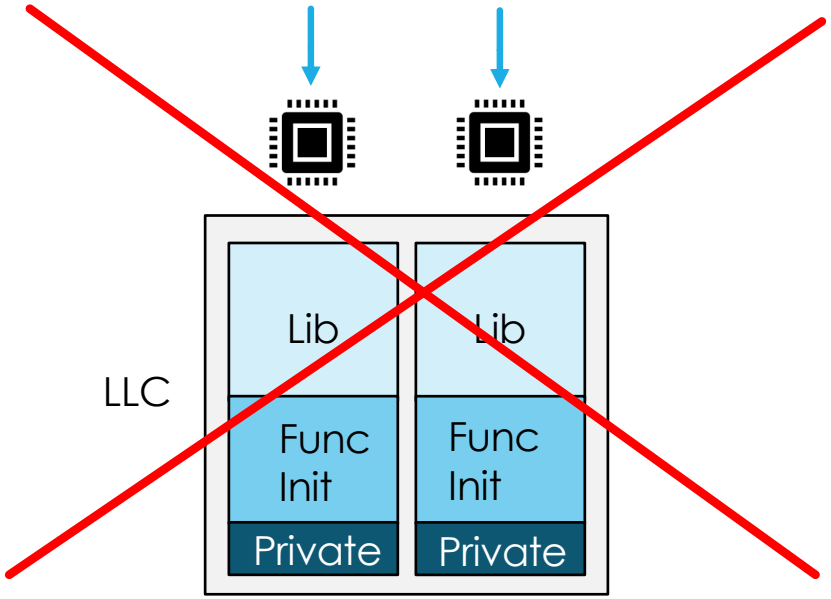
Sharing Memory and Processor State with MXContainers

- Conventional: replicated memory state
  wasted memory



Sharing Memory and Processor State with MXContainers

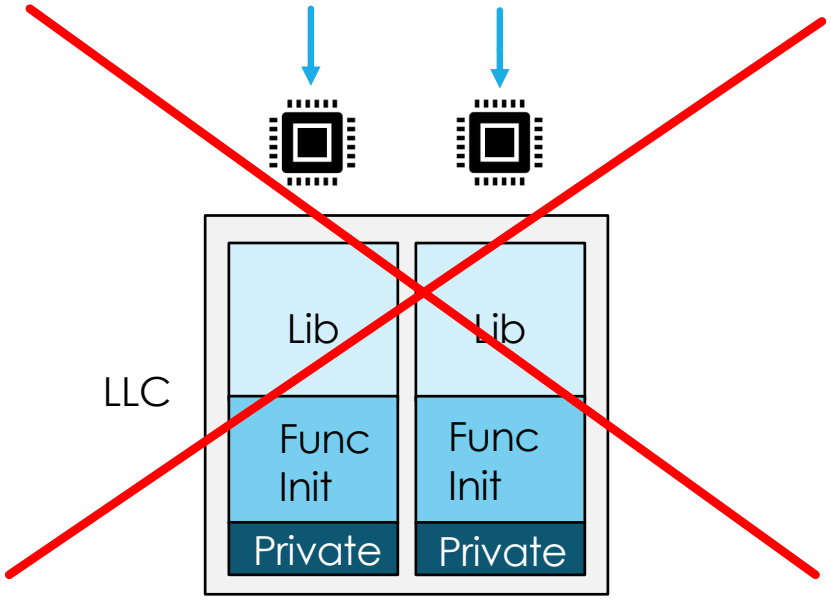
- Conventional: replicated memory state
  wasted memory



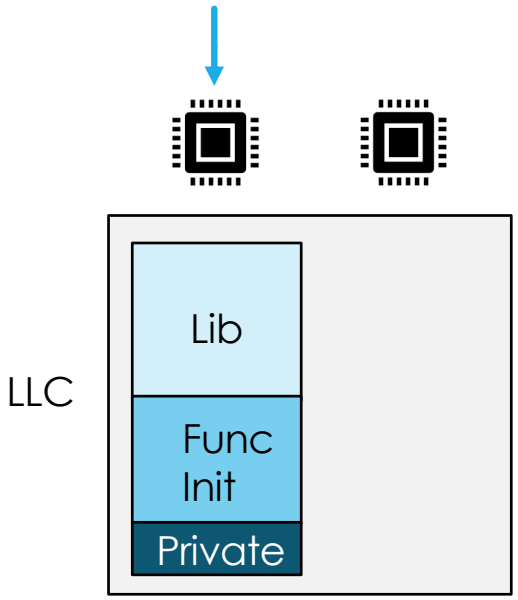
- MXFaaS: sharing memory state

Sharing Memory and Processor State with MXContainers

- Conventional: replicated memory state wasted memory

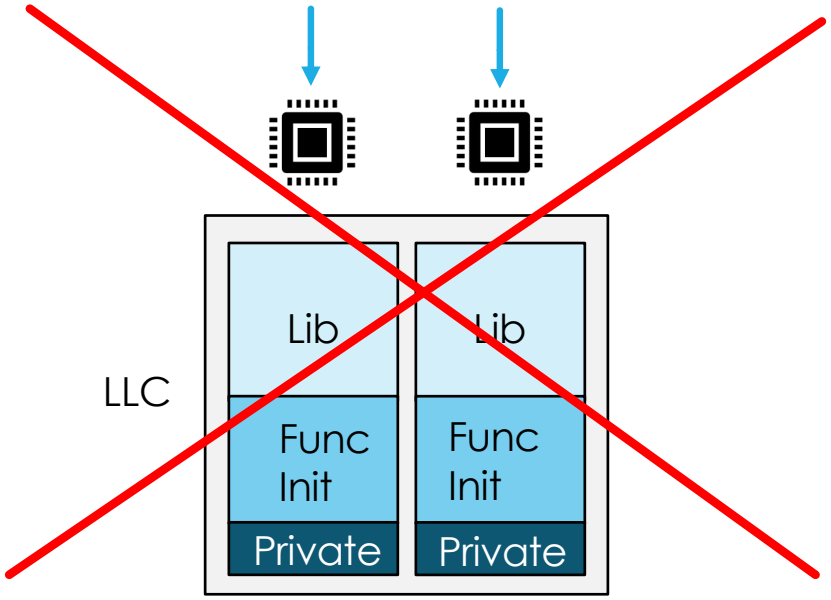


- MXFaaS: sharing memory state

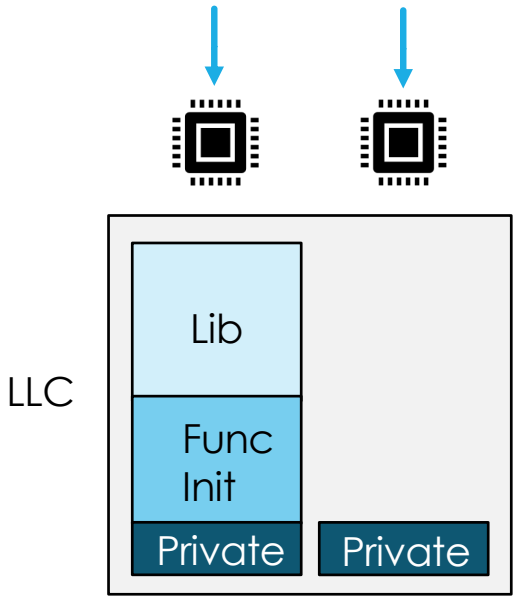


Sharing Memory and Processor State with MXContainers

- Conventional: replicated memory state
  wasted memory

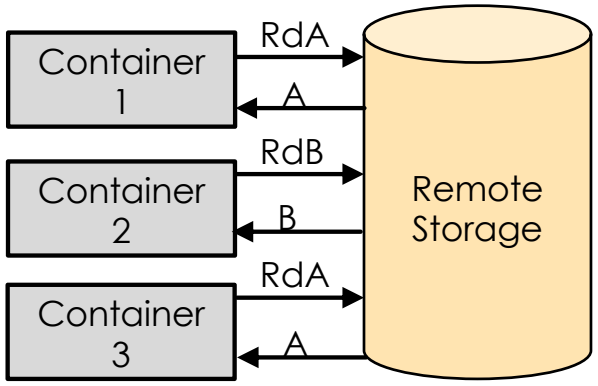


- MXFaaS: sharing memory state
  efficient memory use



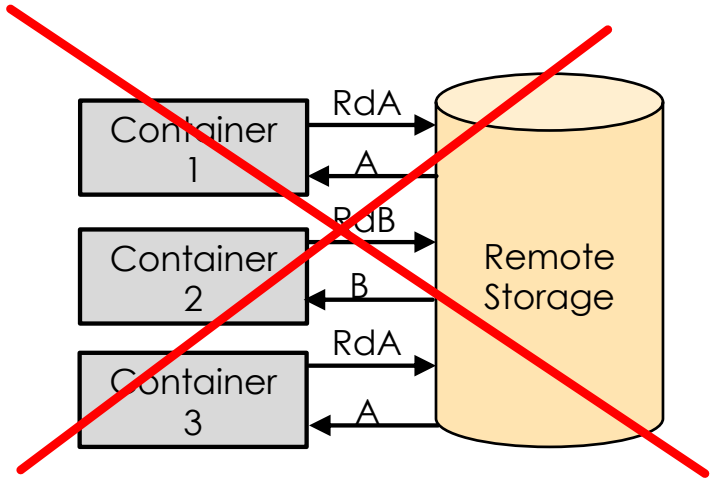
Conventional I/O Bandwidth Use

- Conventional: each requests creates independent connection with the remote storage



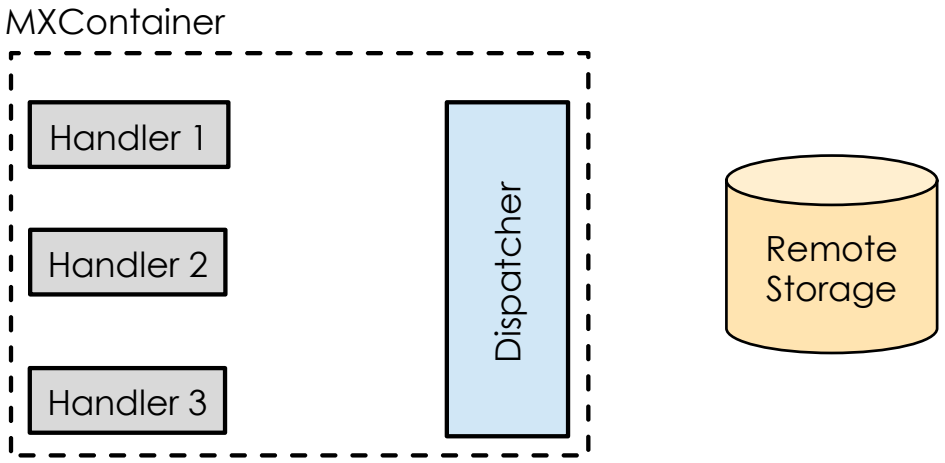
Conventional I/O Bandwidth Use

- Conventional: each requests creates independent connection with the remote storage



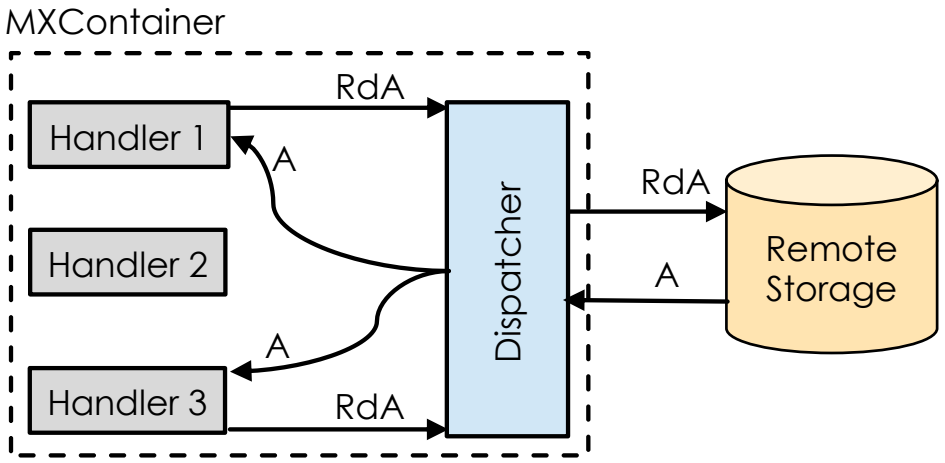
Sharing I/O Bandwidth with MXContainers

- MXFaaS: coalescing remote storage accesses
  - Same data: software Miss Status Holding Table (MSHT)



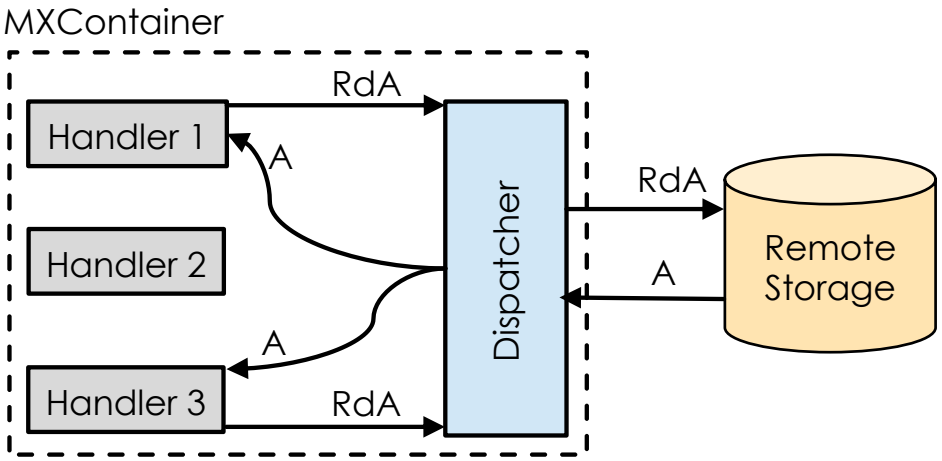
Sharing I/O Bandwidth with MXContainers

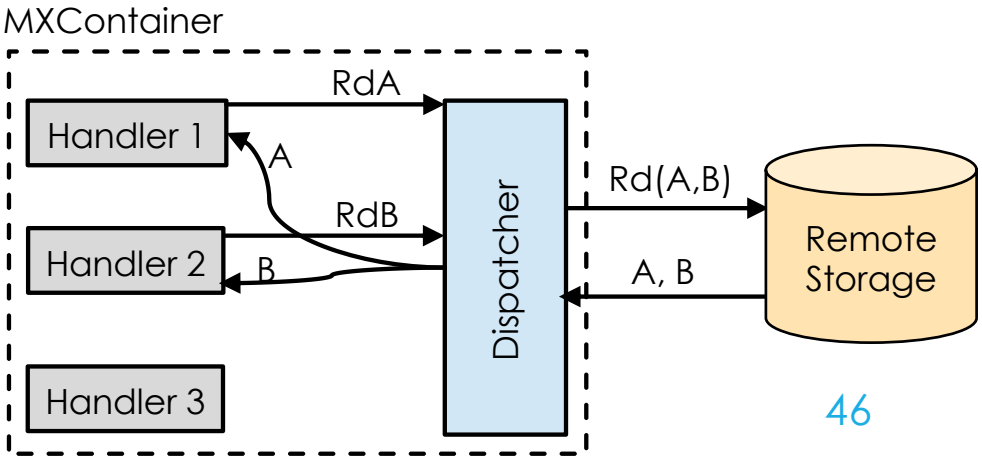
- MXFaaS: coalescing remote storage accesses
  - Same data: software Miss Status Holding Table (MSHT)



Sharing I/O Bandwidth with MXContainers

- MXFaaS: coalescing remote storage accesses
  - Same data: software Miss Status Holding Table (MSHT)
  - Different data: combine scalars into a single vector request



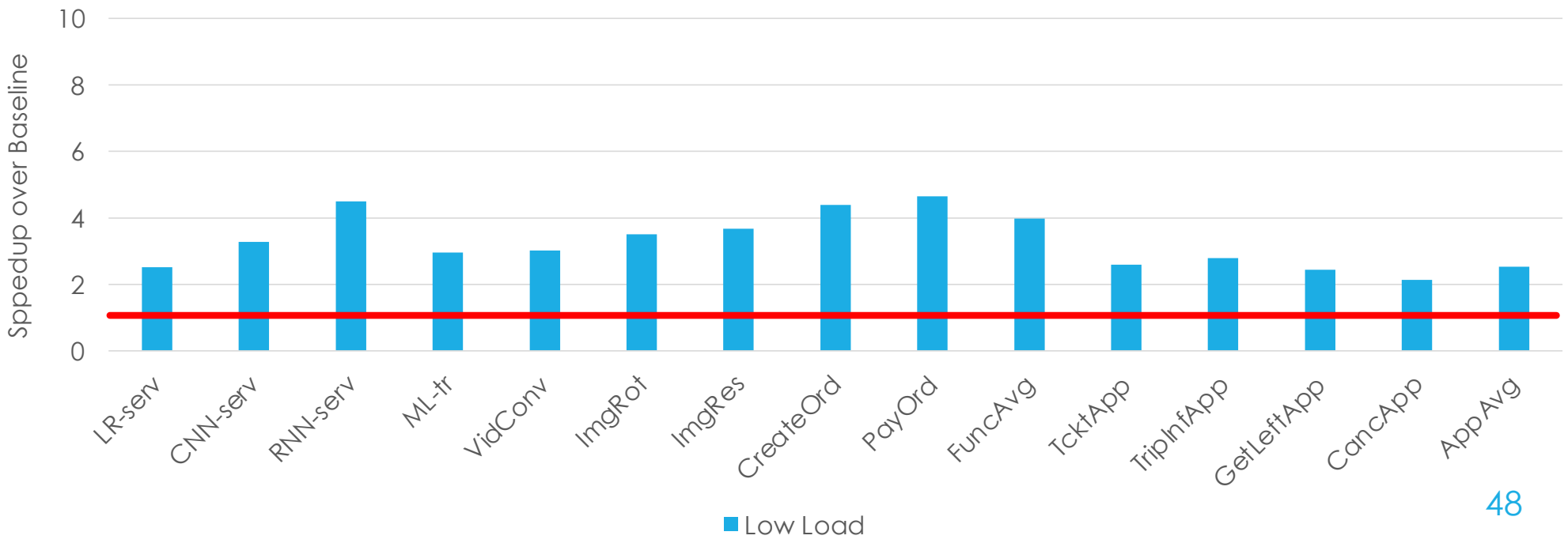


Evaluation Methodology

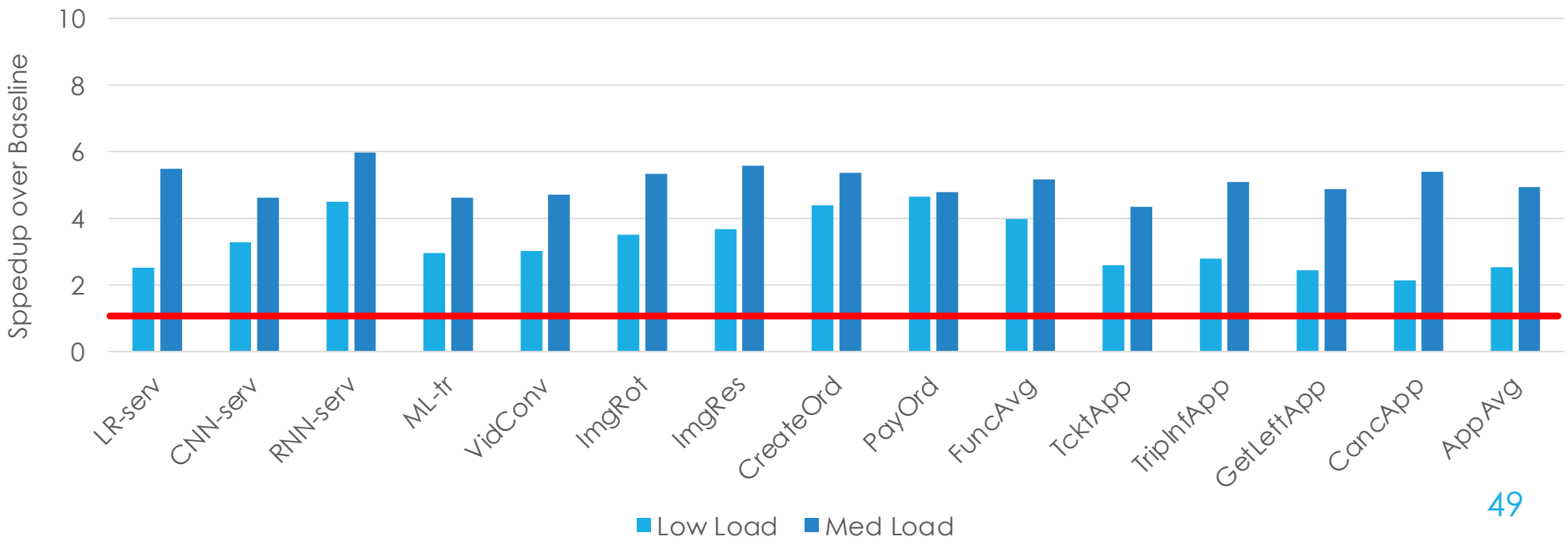
- Cluster with 15 AMD Epyc servers (24 cores, each)
- Platforms: OpenWhisk and KNative
- Baseline: state-of-the-art research platform (Nightcore ASPLOS'21)
- Various functions and applications from two benchmark suites
- 3 system loads
  - Low (450 RPS*)
  - Medium (1000 RPS)
  - High (1800 RPS)

* RPS = Requests Per Second

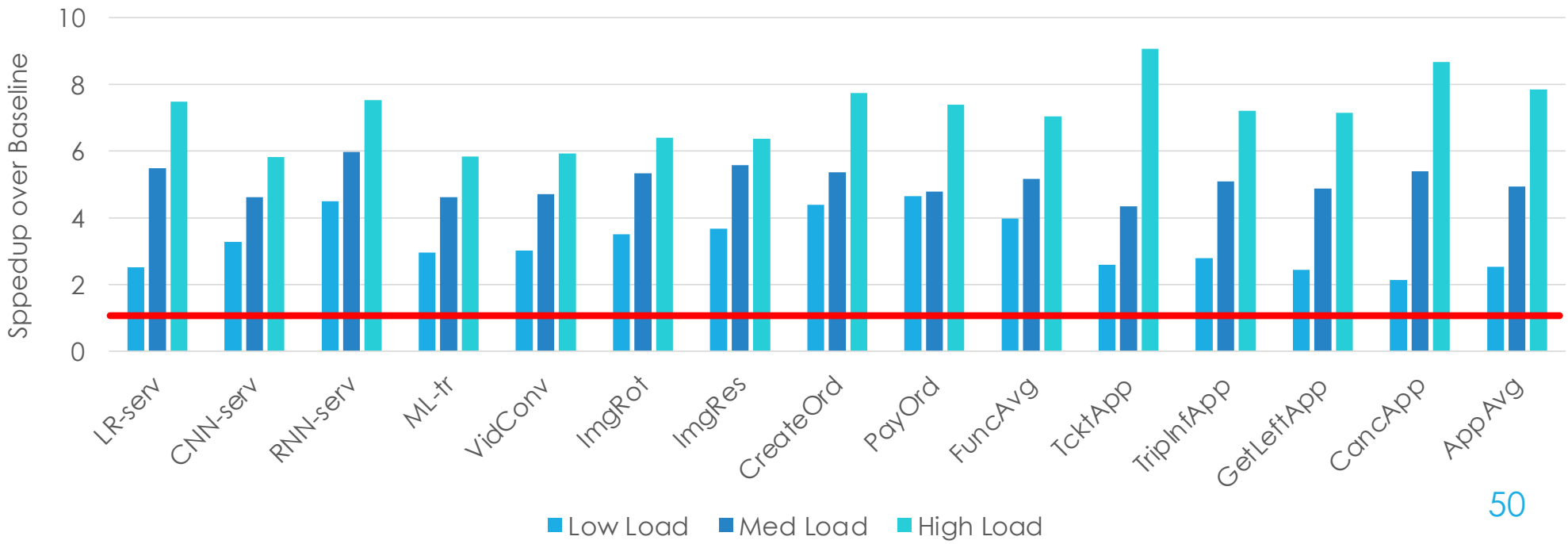
MXFaaS Delivers High Speedups



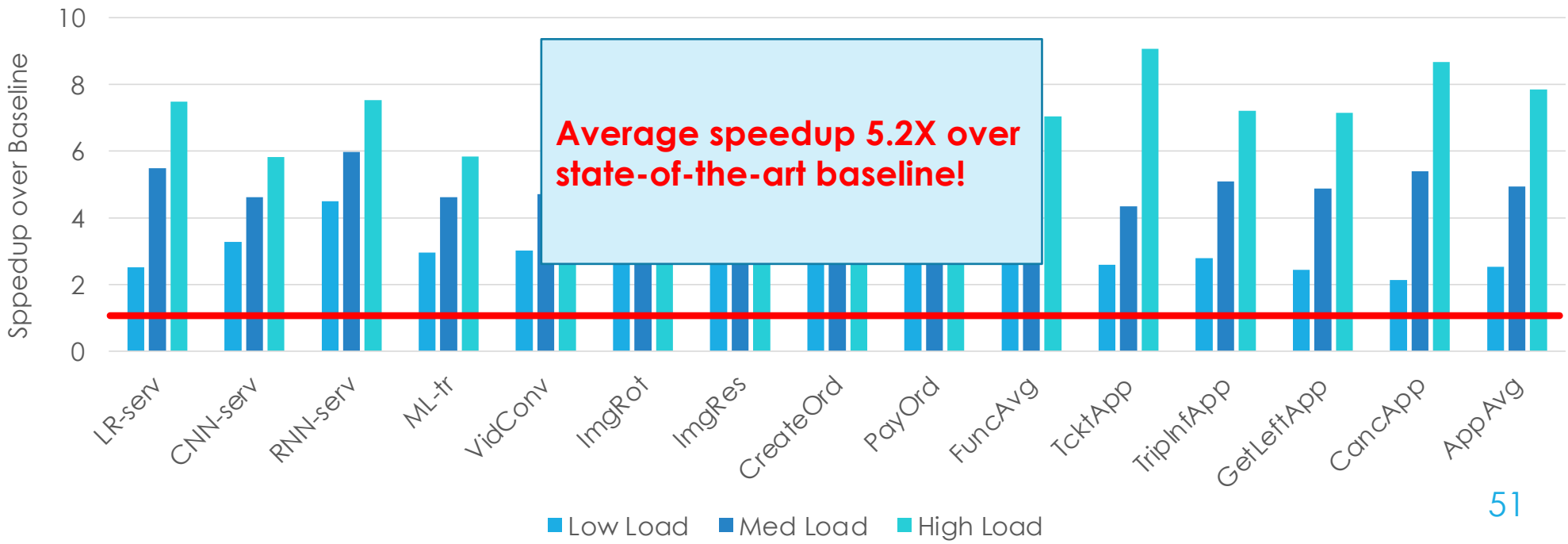
MXFaaS Delivers High Speedups



MXFaaS Delivers High Speedups



MXFaaS Delivers High Speedups



Conclusion

- Serverless computing brings benefits, but its current execution is inefficient
- Propose **MXFaaS** – novel serverless execution model based on resource multiplexing/sharing
  - MXContainers enables sharing CPU cycles, I/O bandwidth and processor/memory state across function invoc
- Average speedup 5.2X, tail latency reduction 7.4X, throughput improvement 4.8X

MXFaaS: Resource Sharing in Serverless Environments for Parallelism and Efficiency

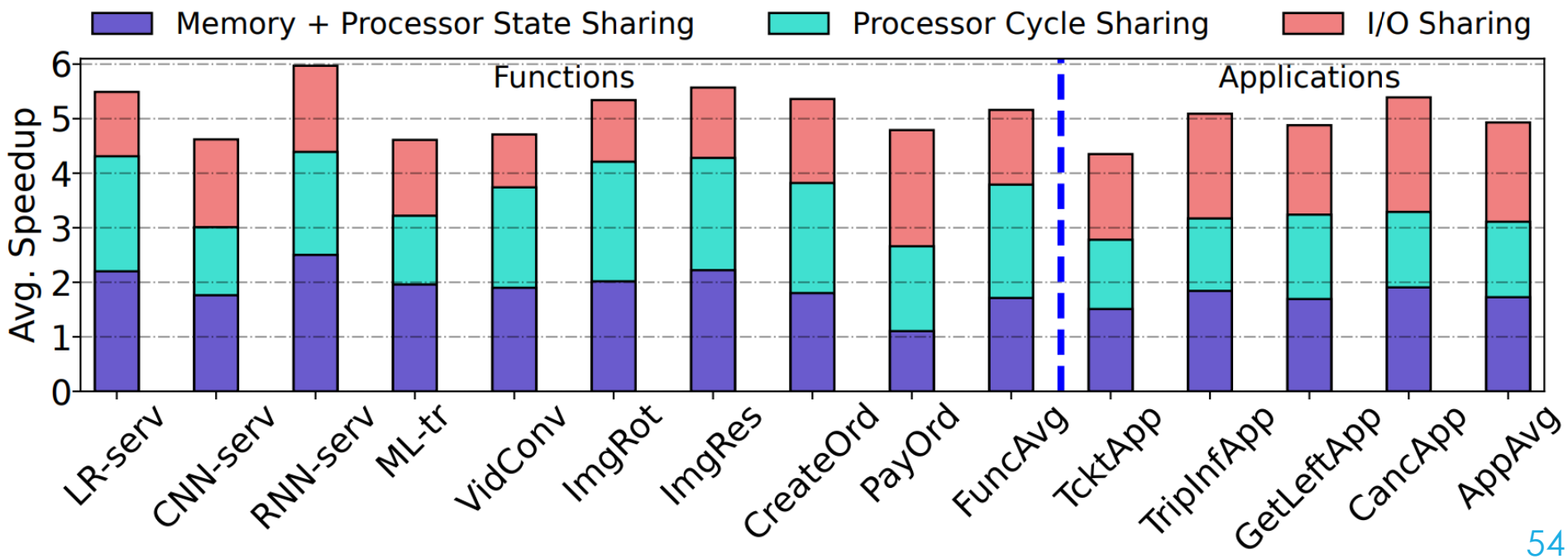
ISCA 2023

Jovan Stojkovic, Tianyin Xu, Hubertus Franke*, Josep Torrellas

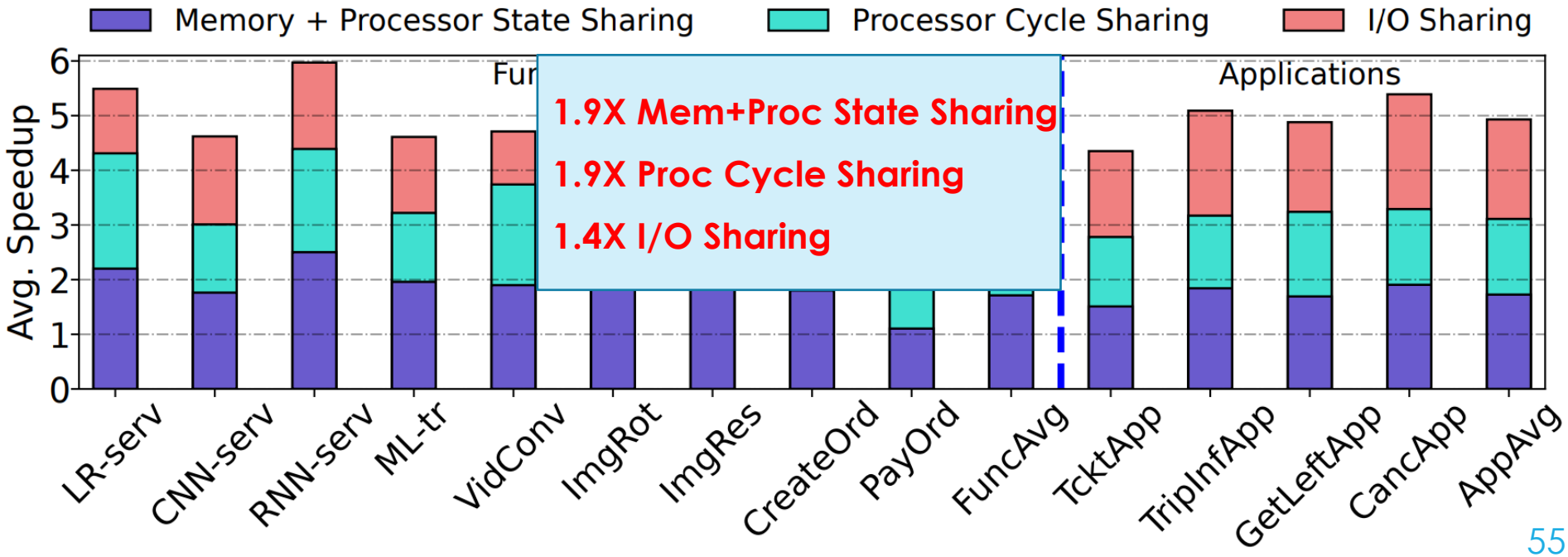
University of Illinois at Urbana-Champaign

*IBM Research

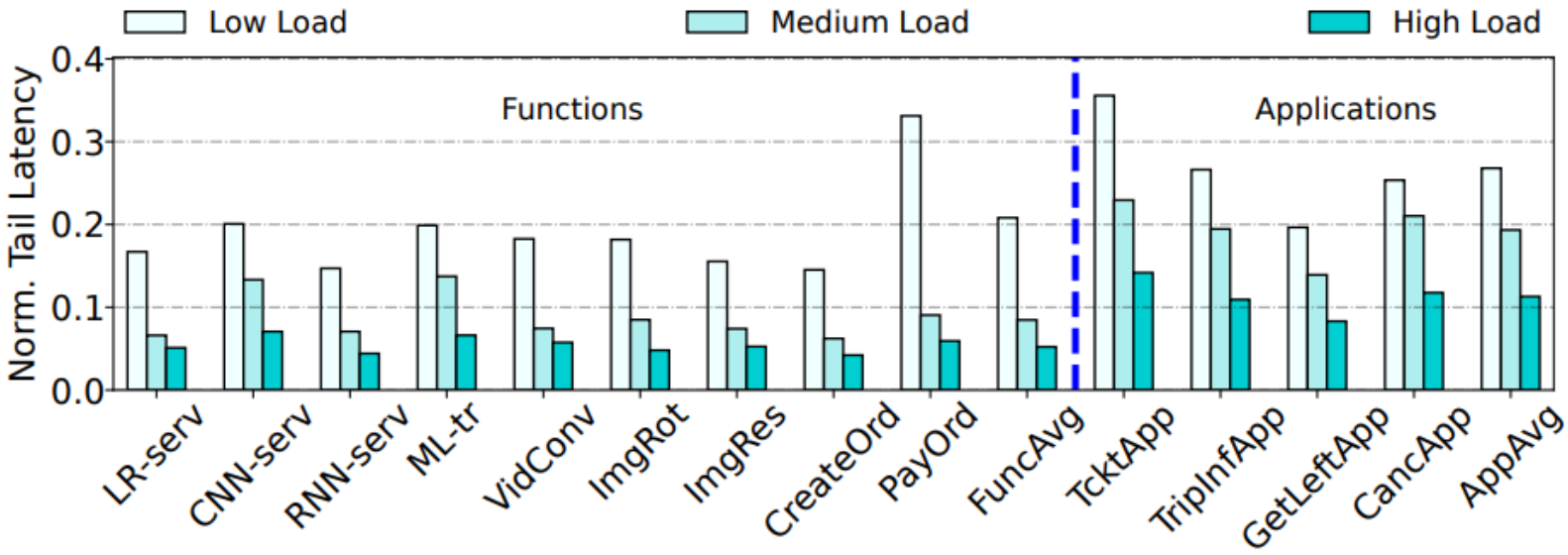
Speedup Breakdown



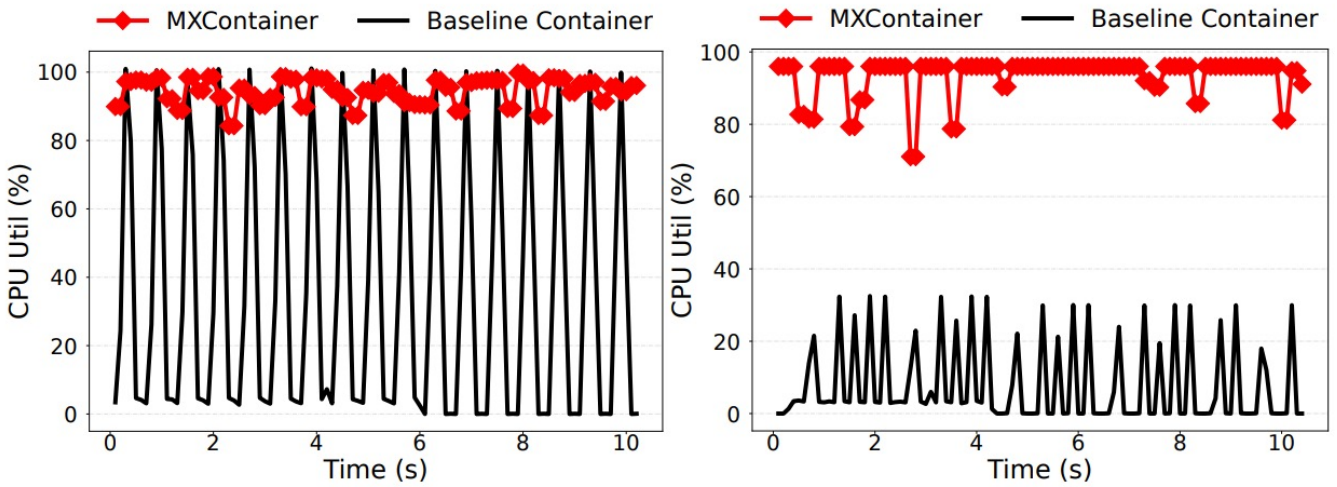
Speedup Breakdown



Tail Latency Reduction



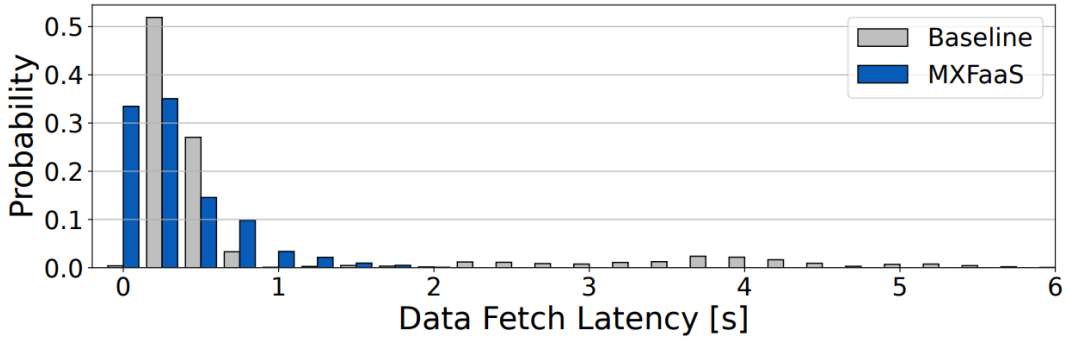
CPU Utilization

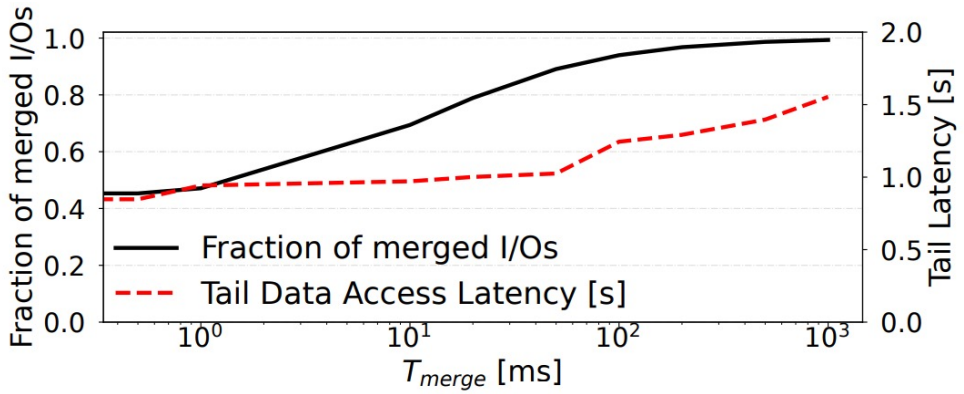


(a) CNN-serv (least idle workload) (b) CreateOrd (most idle workload)

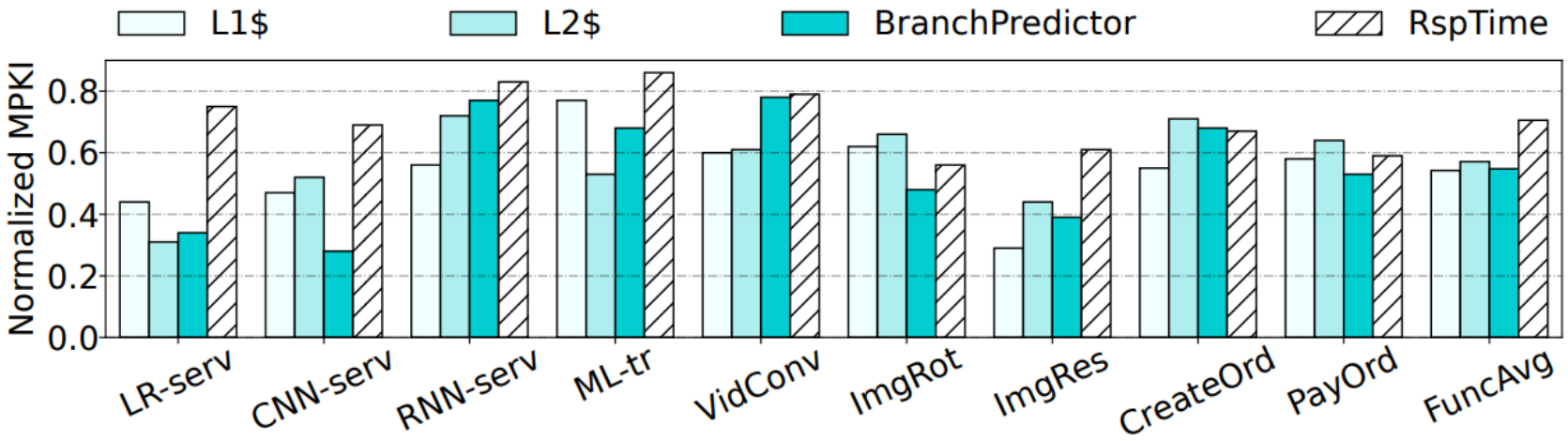
Figure 12: Container CPU utilization over time.

I/O Bandwidth Savings





Microarchitectural State Reuse



MXFaaS Scalability