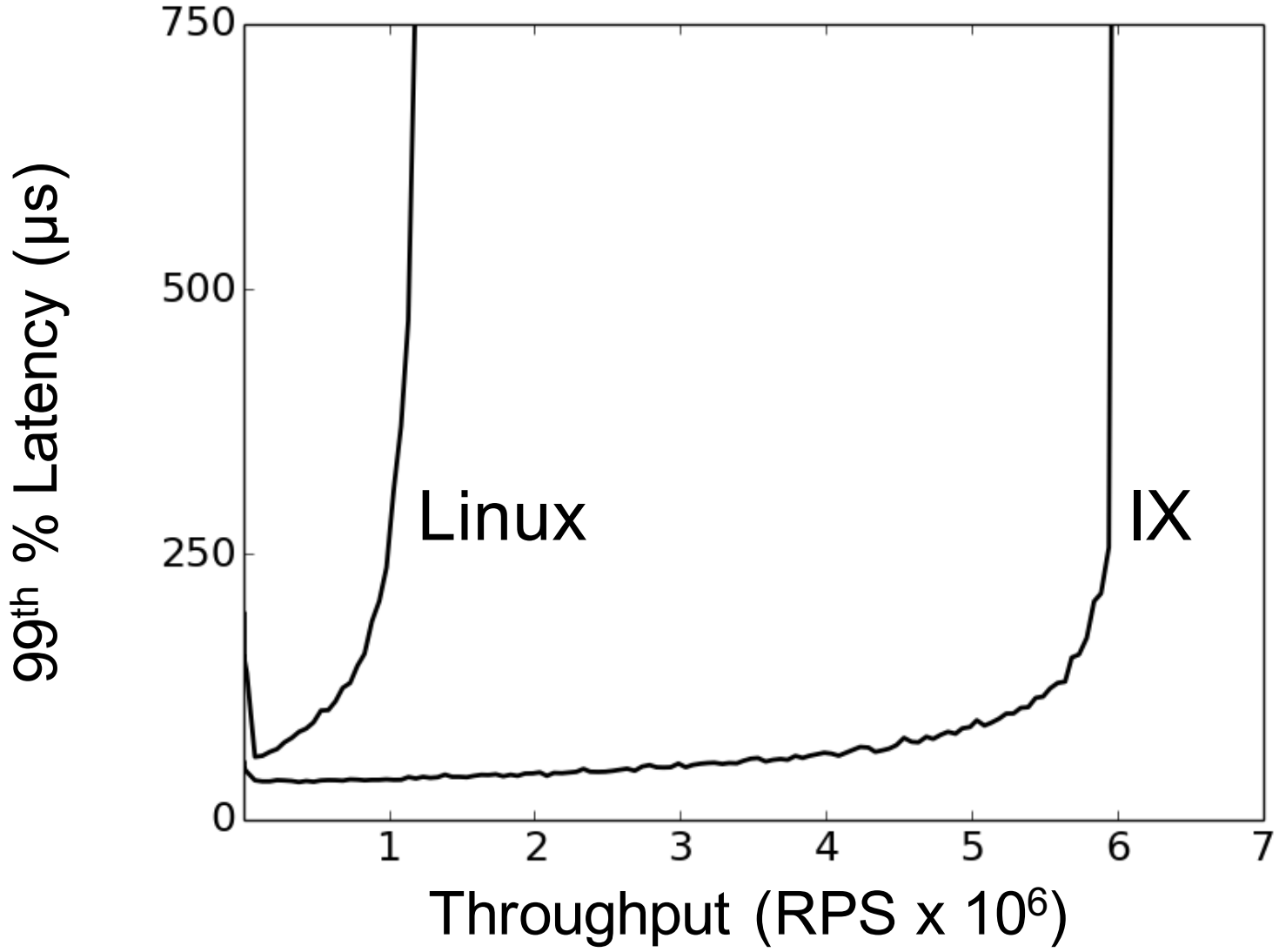
Energy Proportionality and Worload Consolidation for Latency-critical Applications

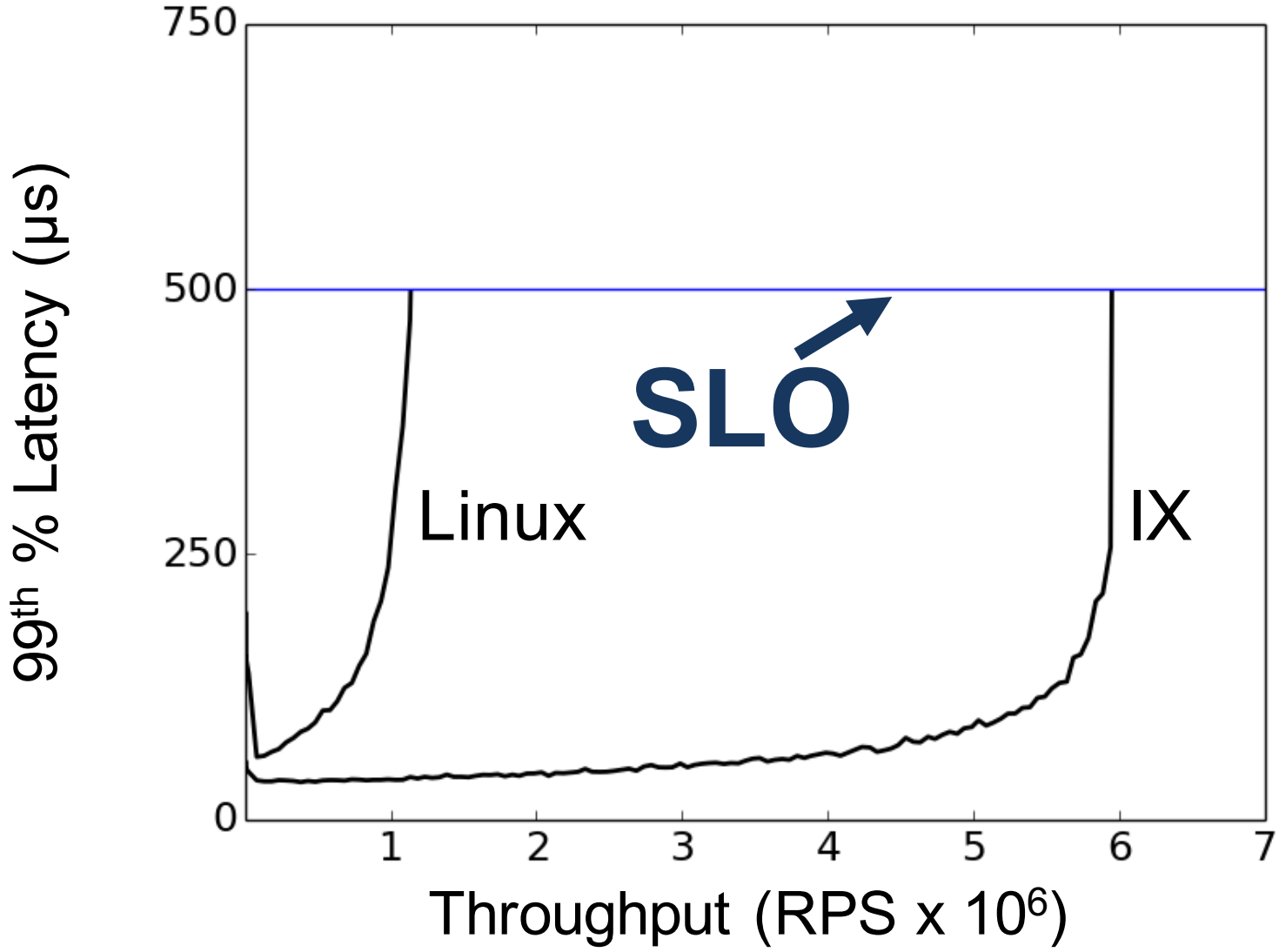
George Prekas, Mia Primorac,
Adam Belay, Christos Kozyrakis,
Edouard Bugnion

Latency-critical applications [OSDI14]



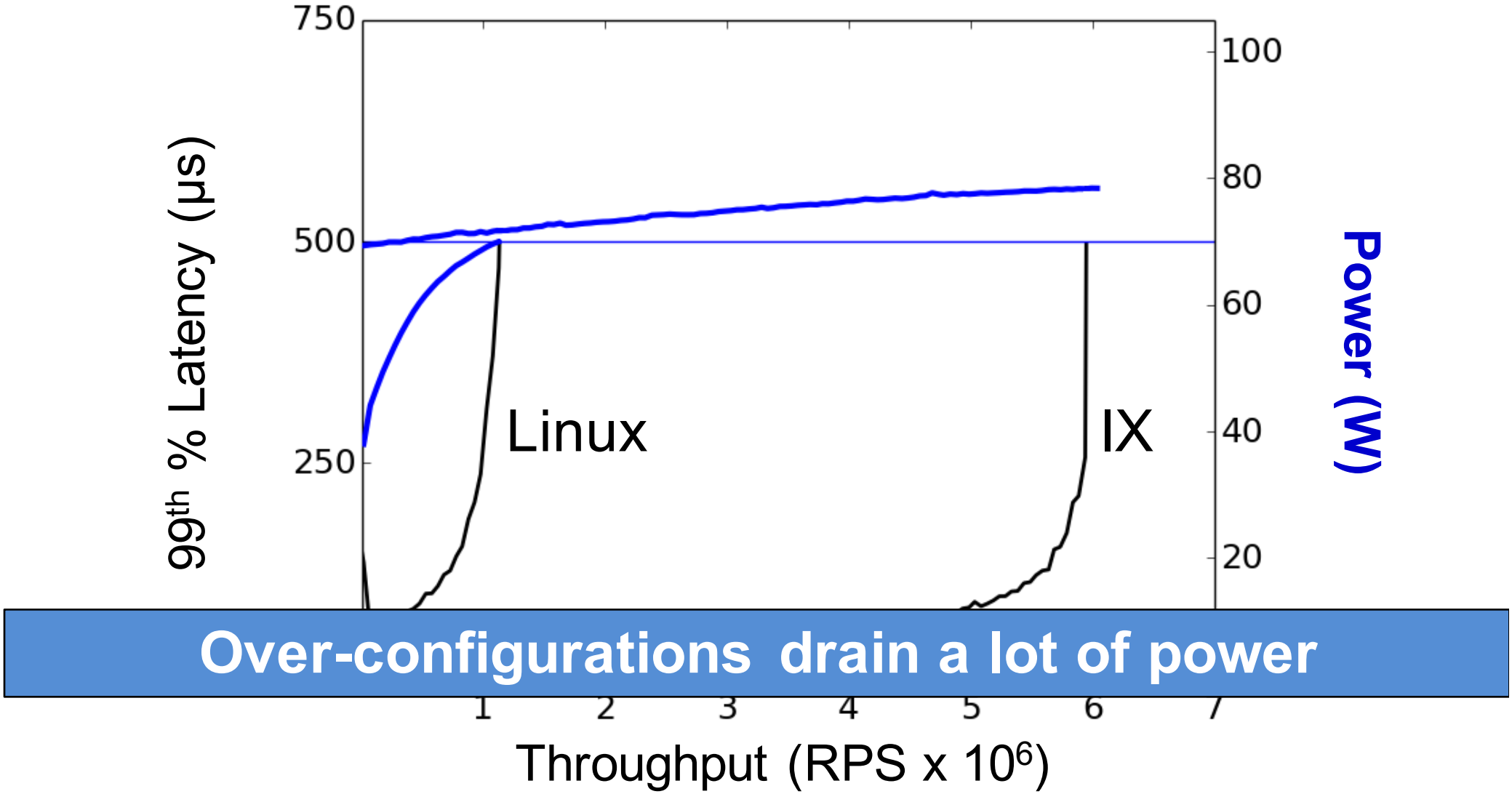
- Memcached, Facebook USR workload, 2752 connections
- Server: Xeon E5-2665 @2.4 Ghz, 8 cores and 16 HTs, Intel x520

Latency-critical applications [OSDI14]



- Memcached, Facebook USR workload, 2752 connections
- Server: Xeon E5-2665 @2.4 Ghz, 8 cores and 16 HTs, Intel x520

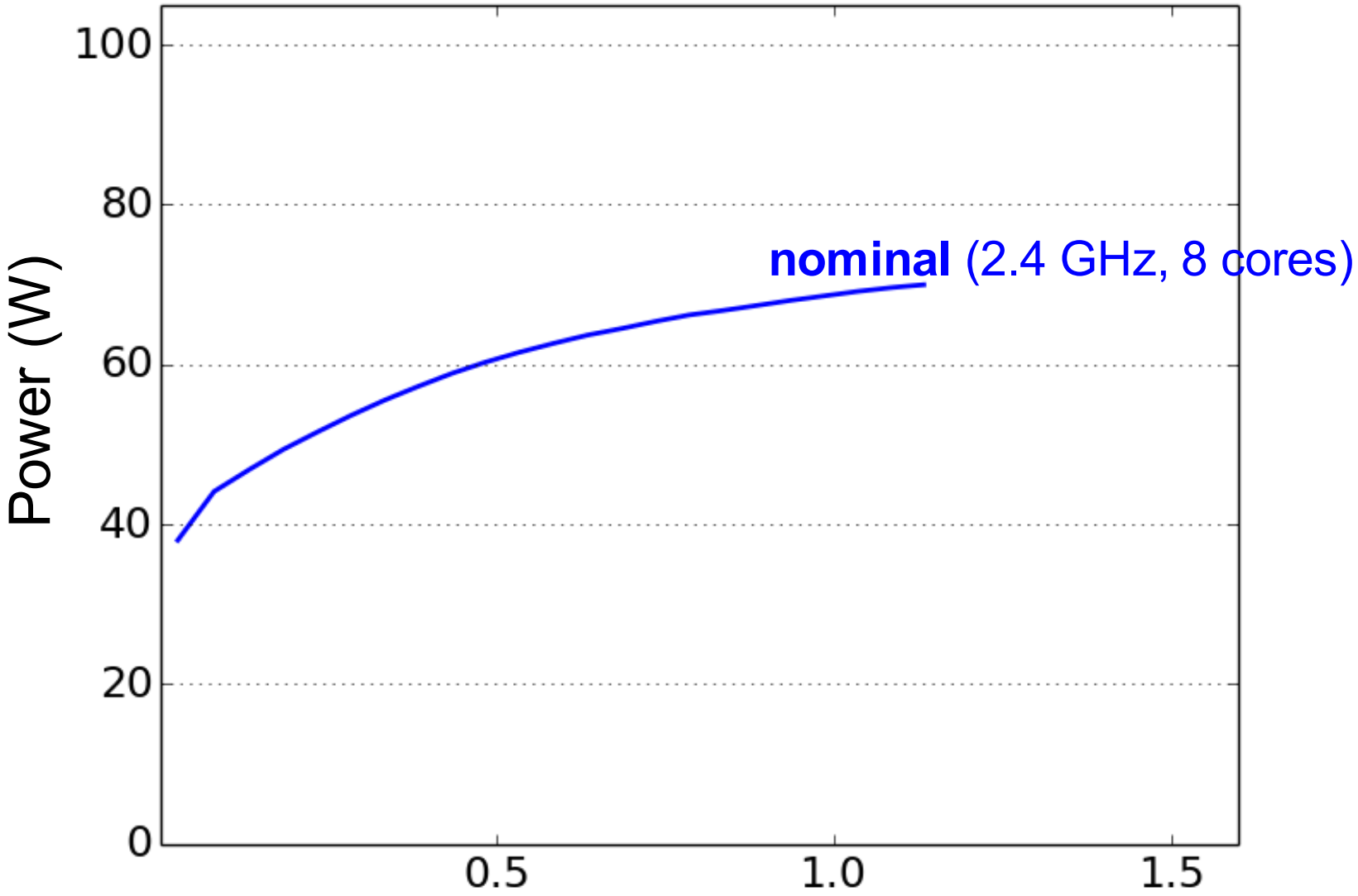
What about energy efficiency?



Over-configurations drain a lot of power

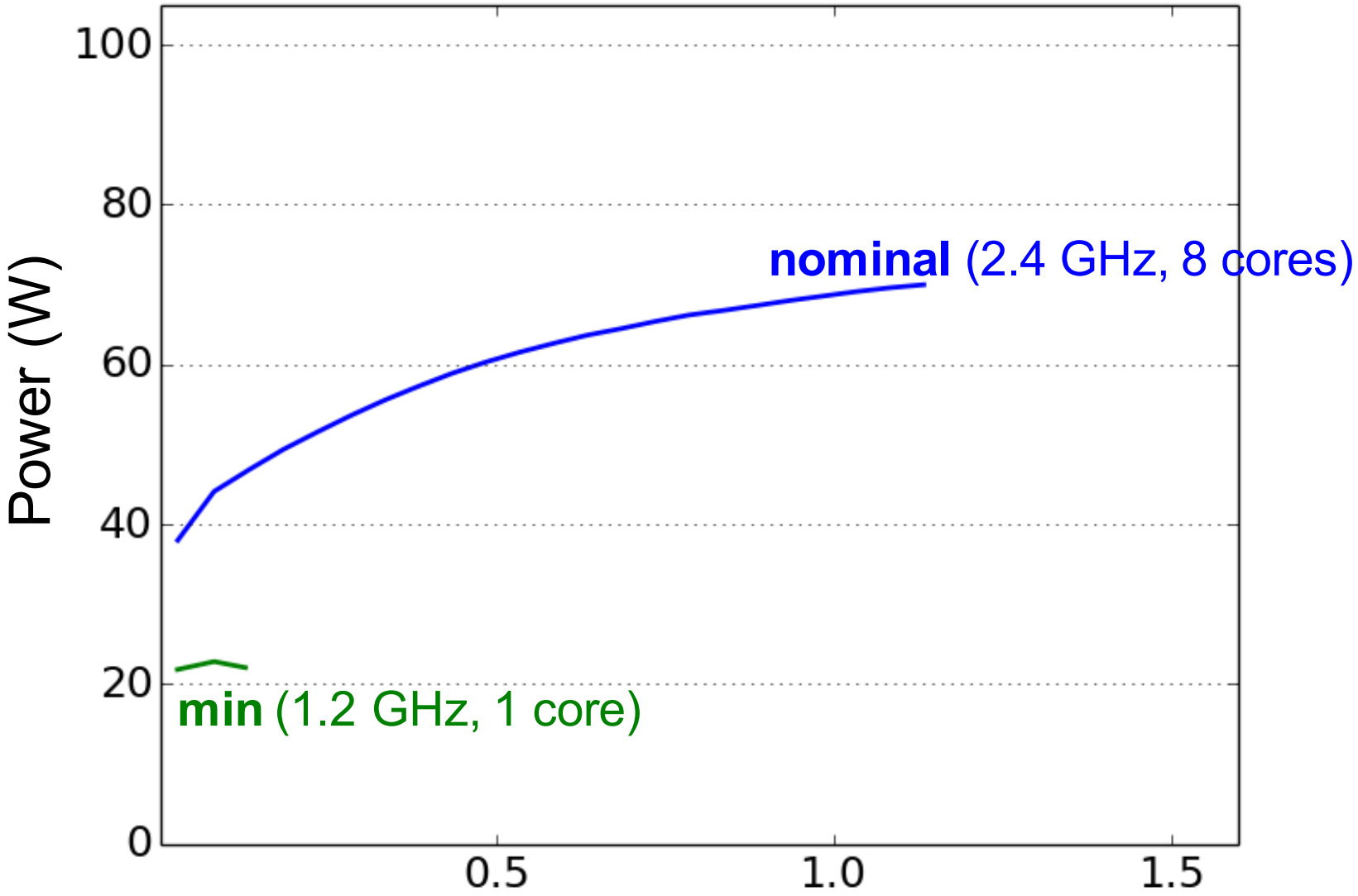
- Memcached, Facebook USR workload, 2752 connections
- Server: Xeon E5-2665 @2.4 Ghz, 8 cores and 16 HTs, Intel x520

Static configurations trade-off



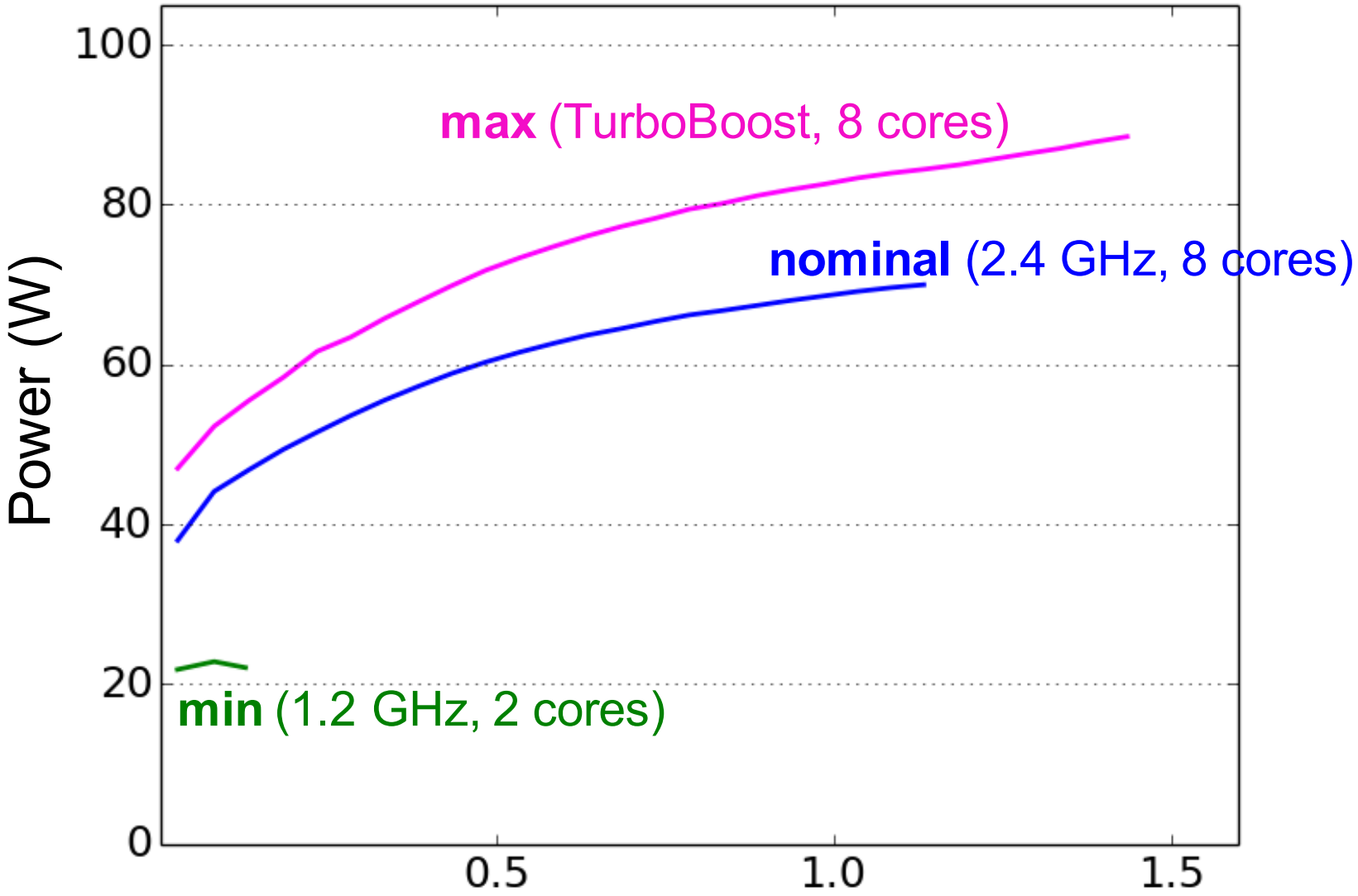
Linux; Memcached MRPS at SLO

Static configurations trade-off



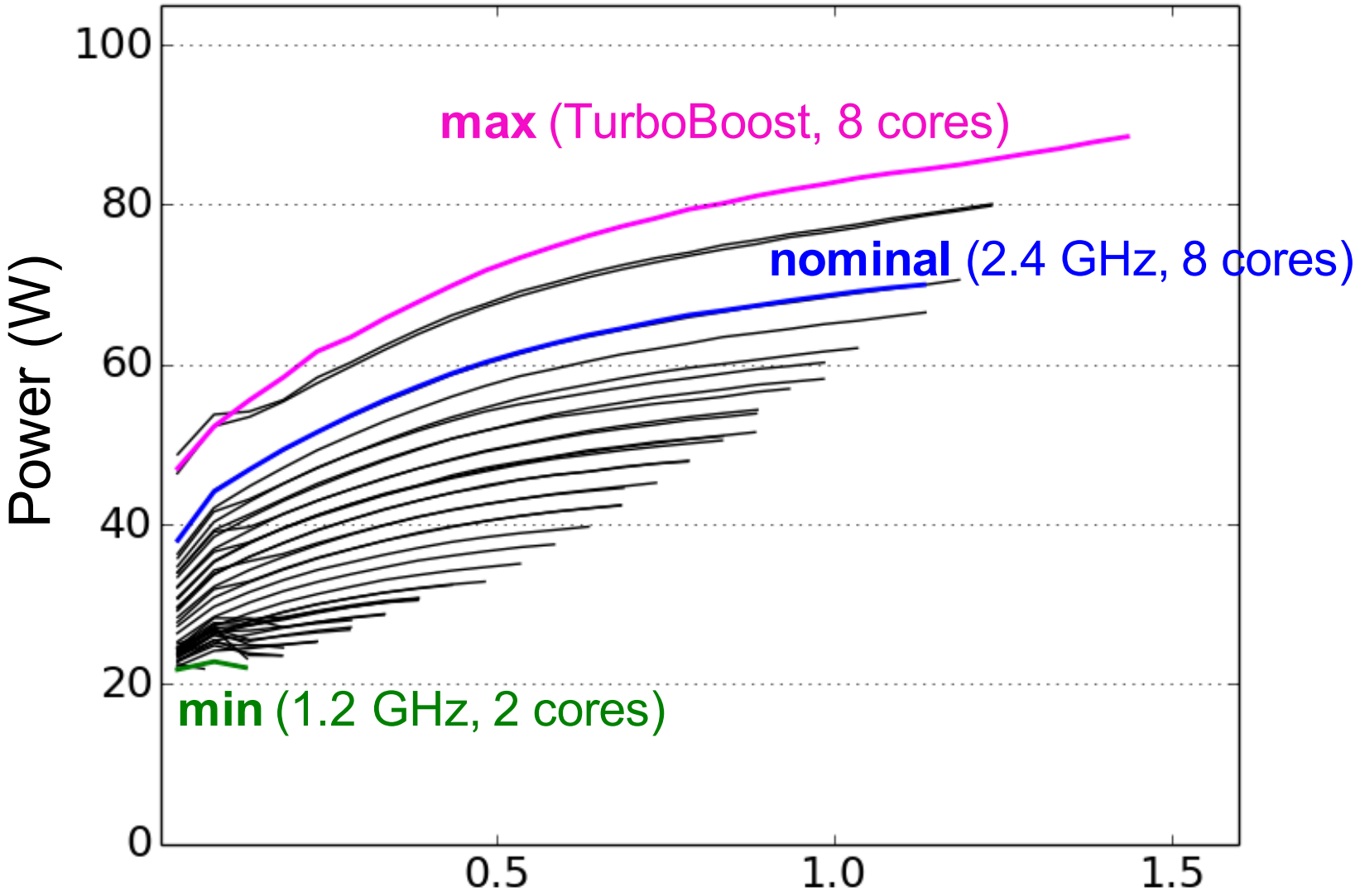
Linux; Memcached MRPS at SLO

Static configurations trade-off



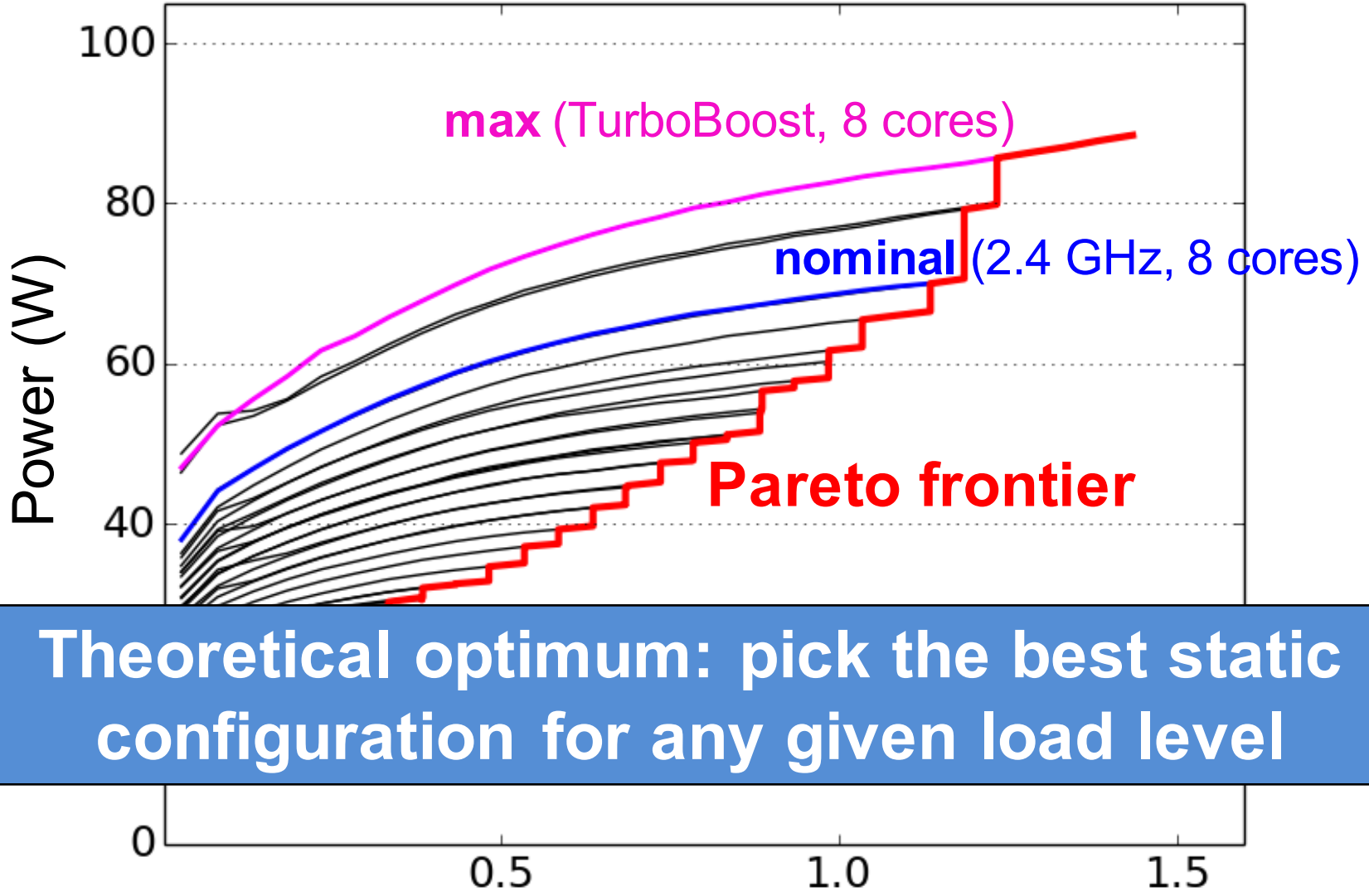
Linux; Memcached MRPS at SLO

Static configurations trade-off



224 static configurations

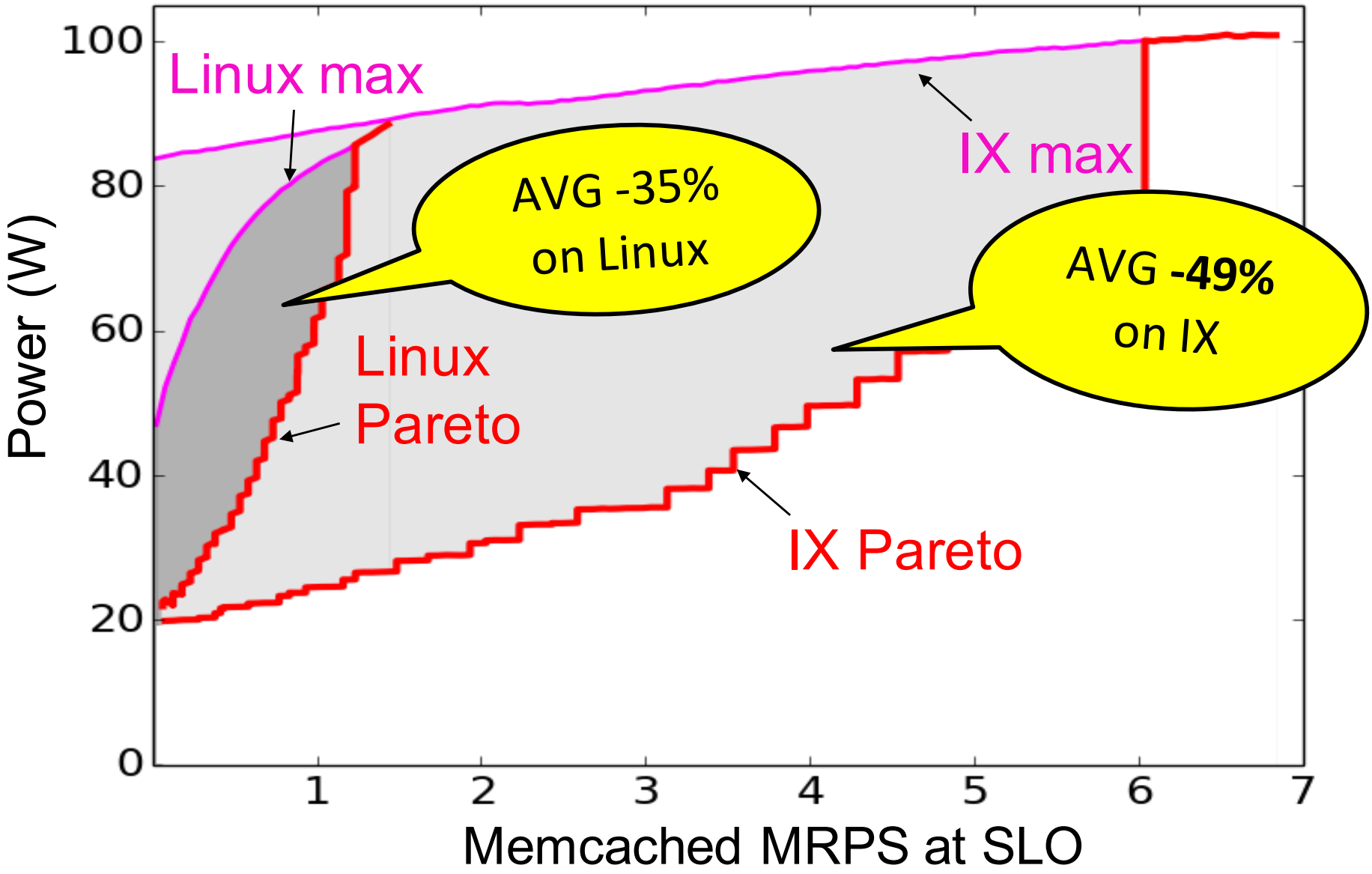
Pareto Frontier



Theoretical optimum: pick the best static configuration for any given load level

Linux; 224 static configurations

Potential energy savings

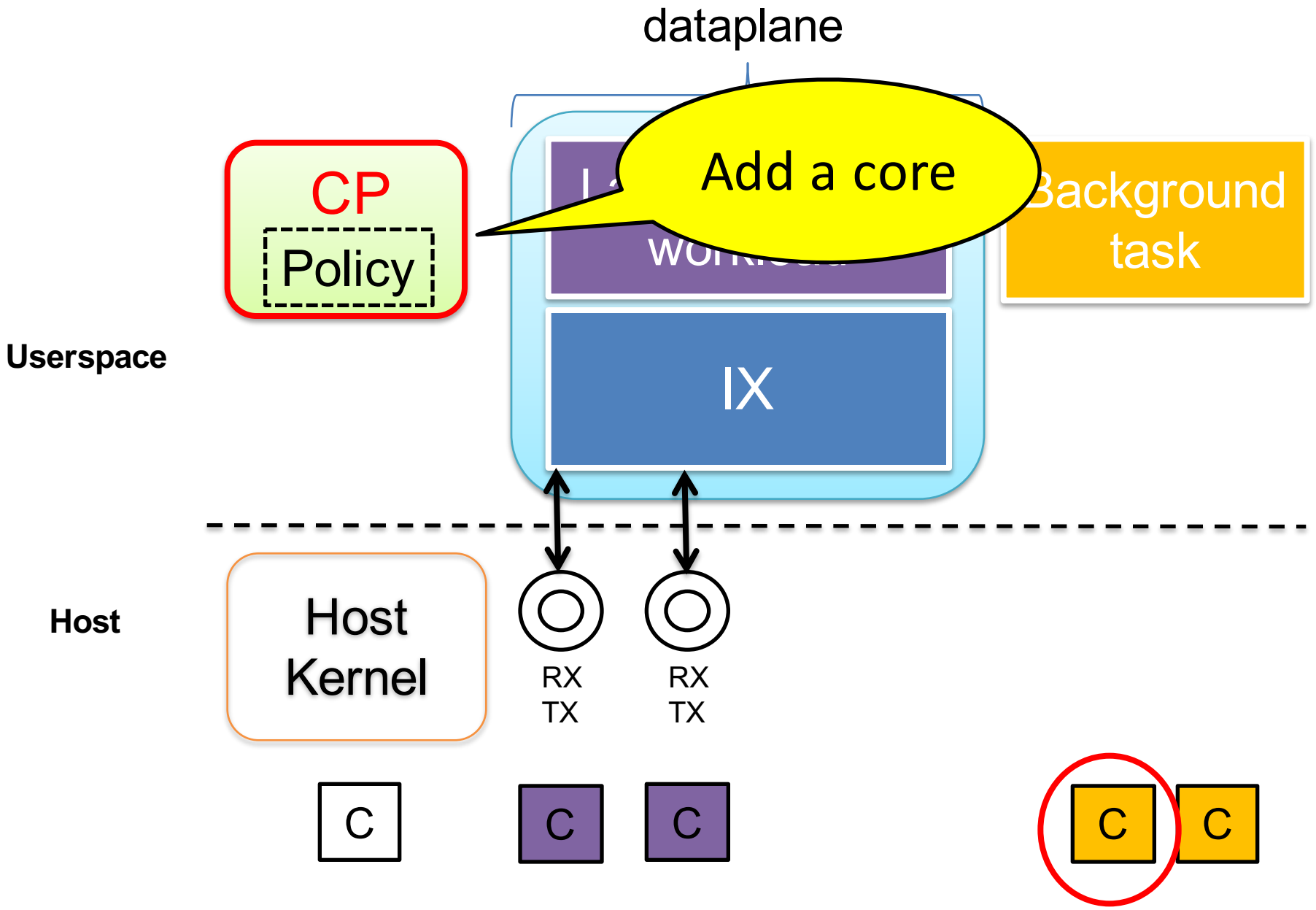


Contributions

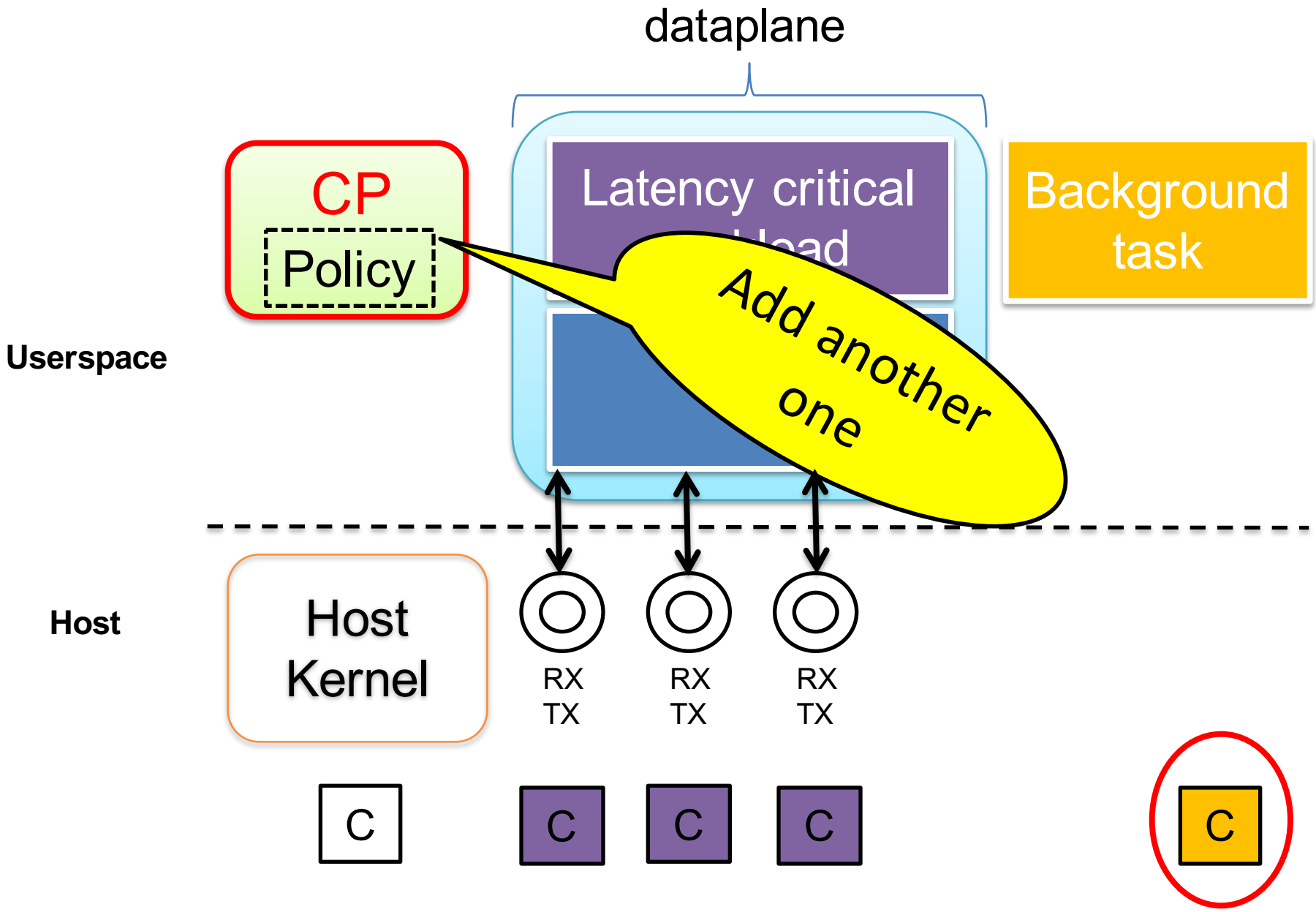
- Dynamic resource controls
  - for low-latency, high-throughput dataplanes
  - Supports energy proportionality and workload consolidation policies

- Evaluation of dynamic resource controls vs.
  - Maximum configuration (static)
  - Pareto-optimal behavior (theoretical bound)

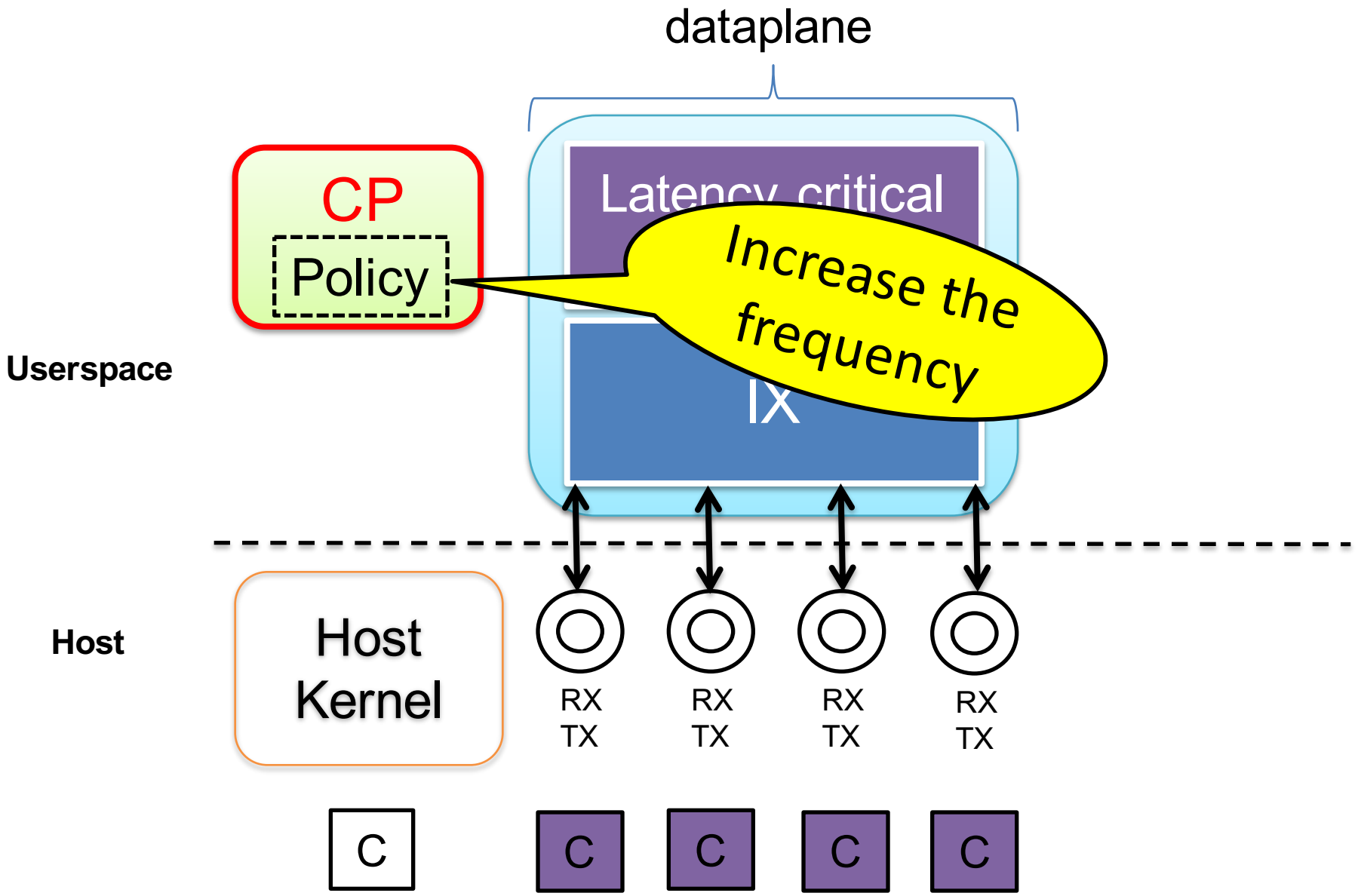
Dynamic resource control



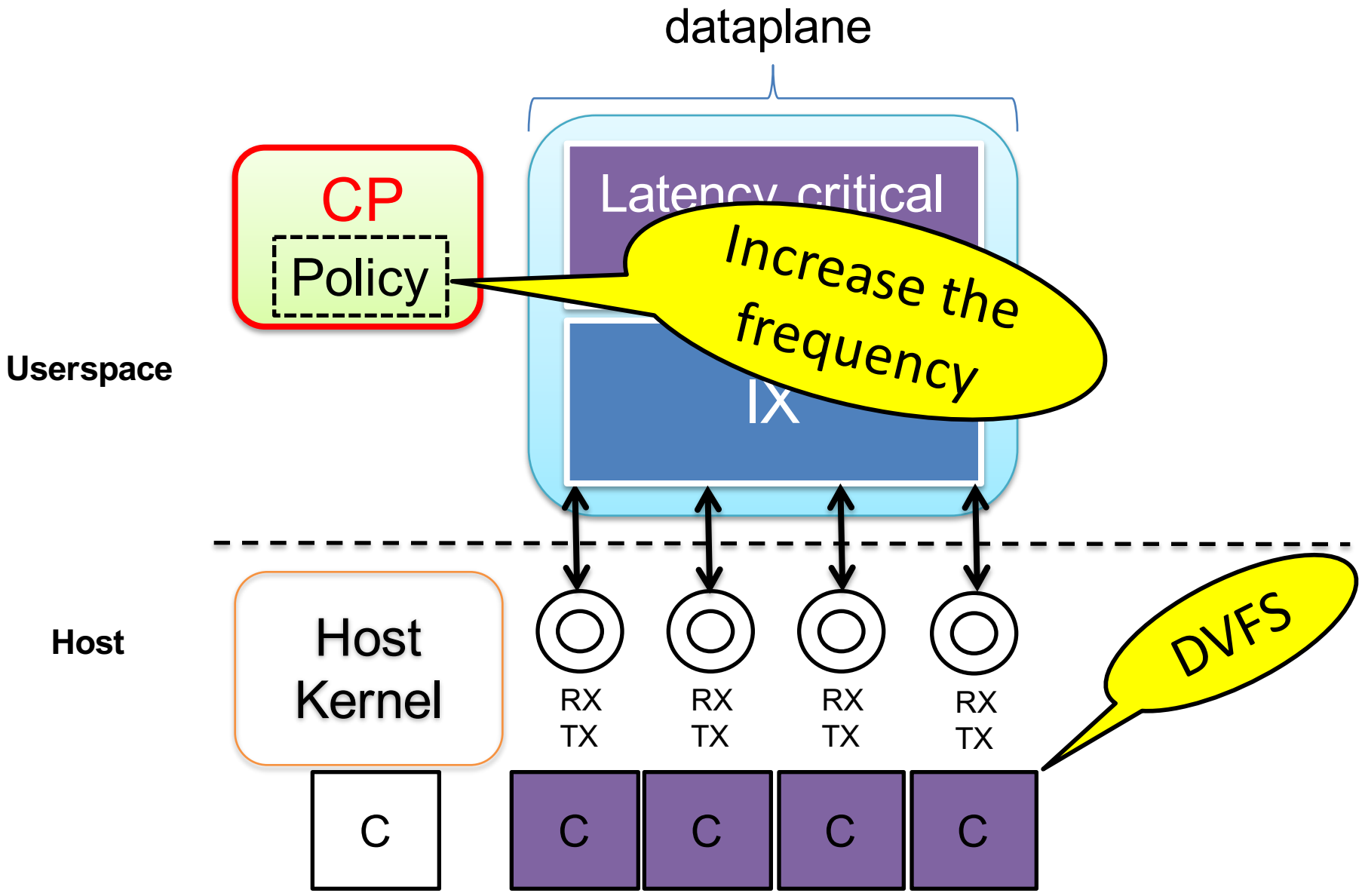
Dynamic resource control



Dynamic resource control



Dynamic resource control



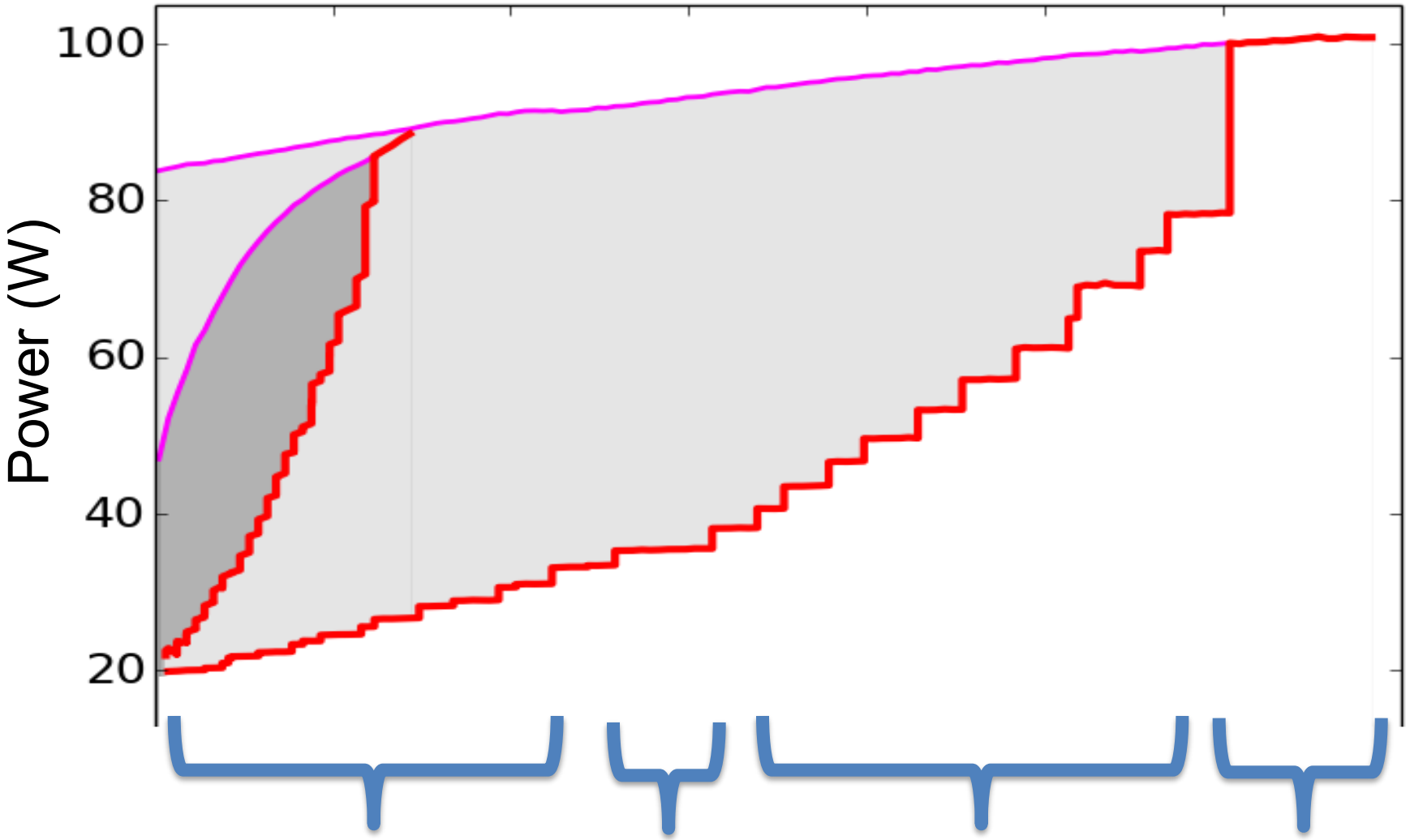
Key challenges

1. Which resources to add/remove ?
   - inferred by Pareto analysis
   - different for energy proportionality and workload consolidation

2. When to add/remove resources ?
   - Need to design a stable control loop
   - Different triggers to add/remove resources

3. How to add/remove cores
   - Fast, TCP-friendly rebalancing mechanism

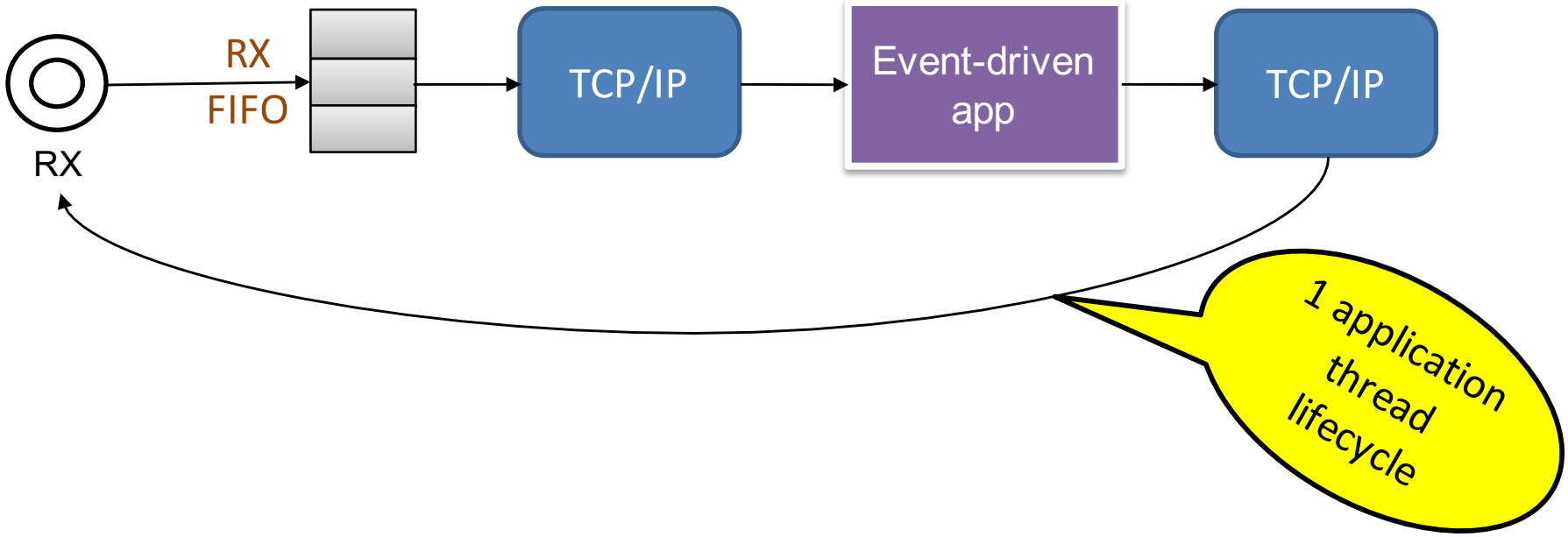
#1: Resource Adjustment Policies



+core +ht +dvfs +Turbo

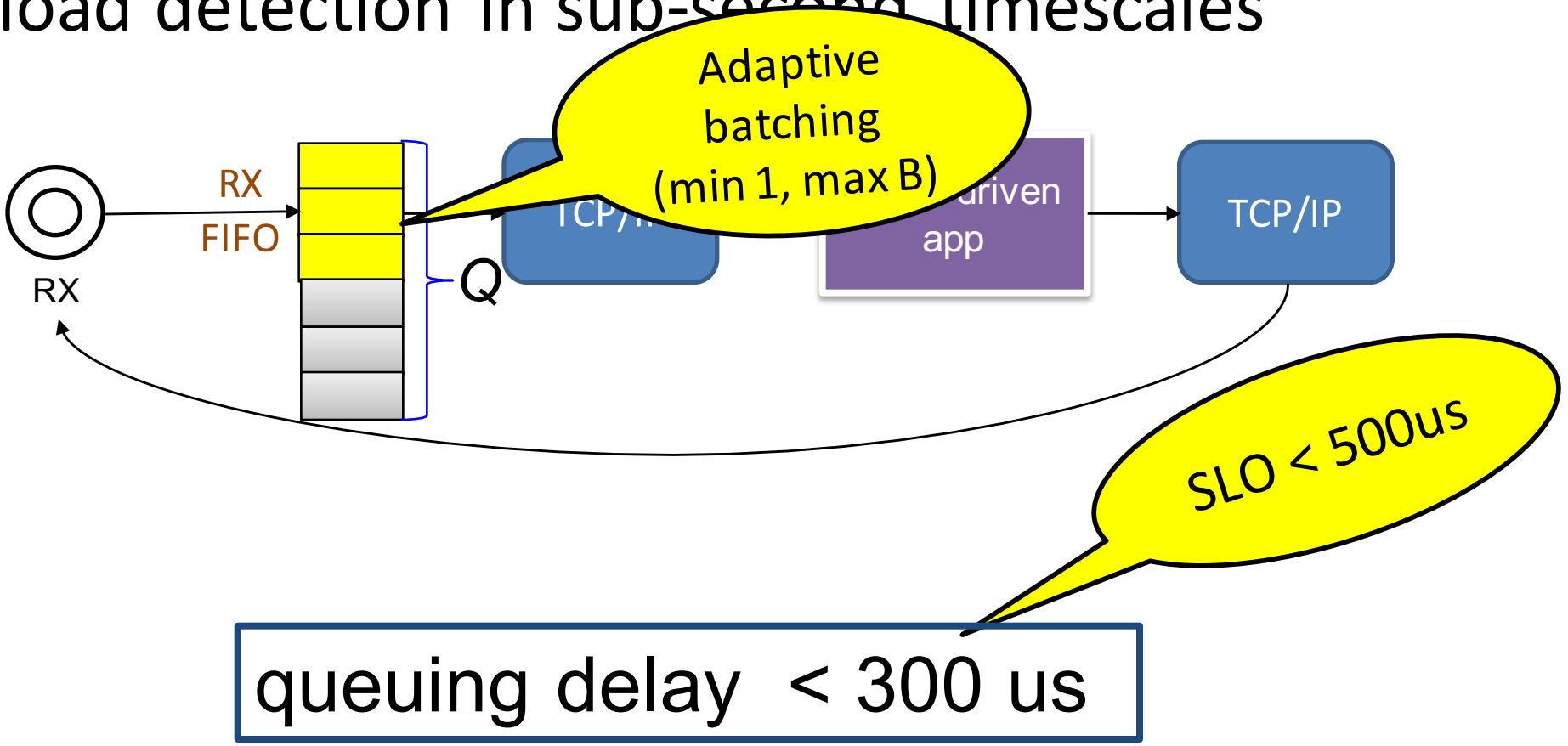
#2: Detection

- Centralized queues provide a single point of load detection in sub-second timescales



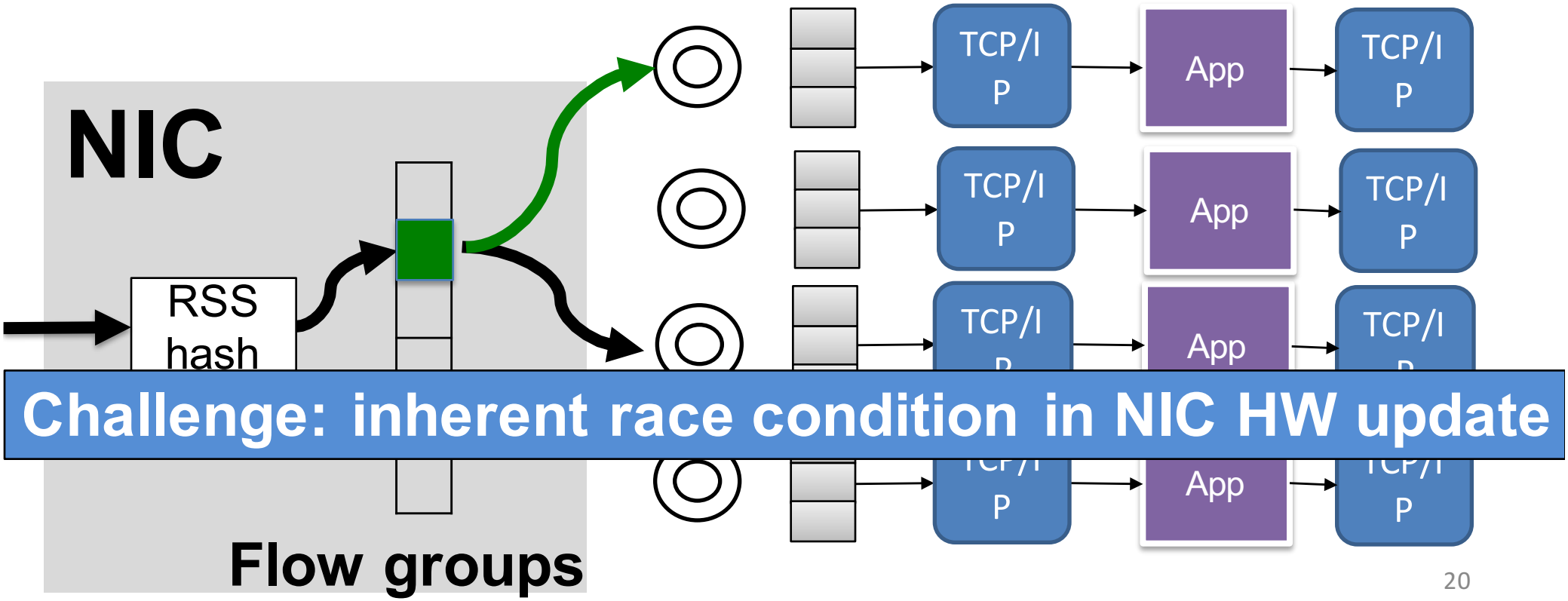
#2 Detection (add)

- Centralized queues provide a single point of load detection in sub-second timescales



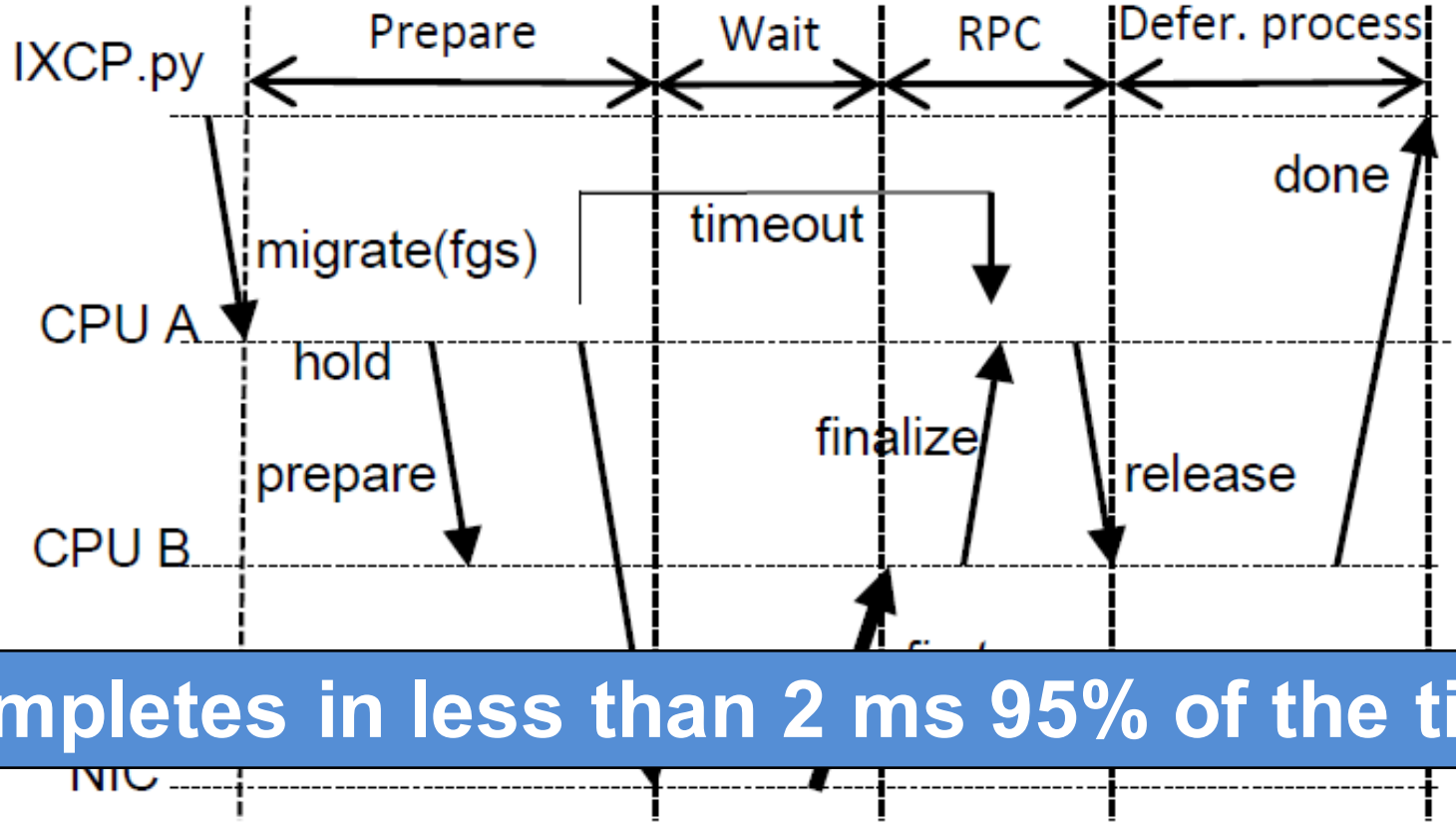
#3: TCP-friendly add/remove core

- Challenge: maintain coherence-free IX design
  - Queues dedicated to cores
  - Lock-free TCP stack



#3: Flow-group Migration Algorithm

- TCP-friendly
- Without dropping or reordering packets



Completes in less than 2 ms 95% of the time

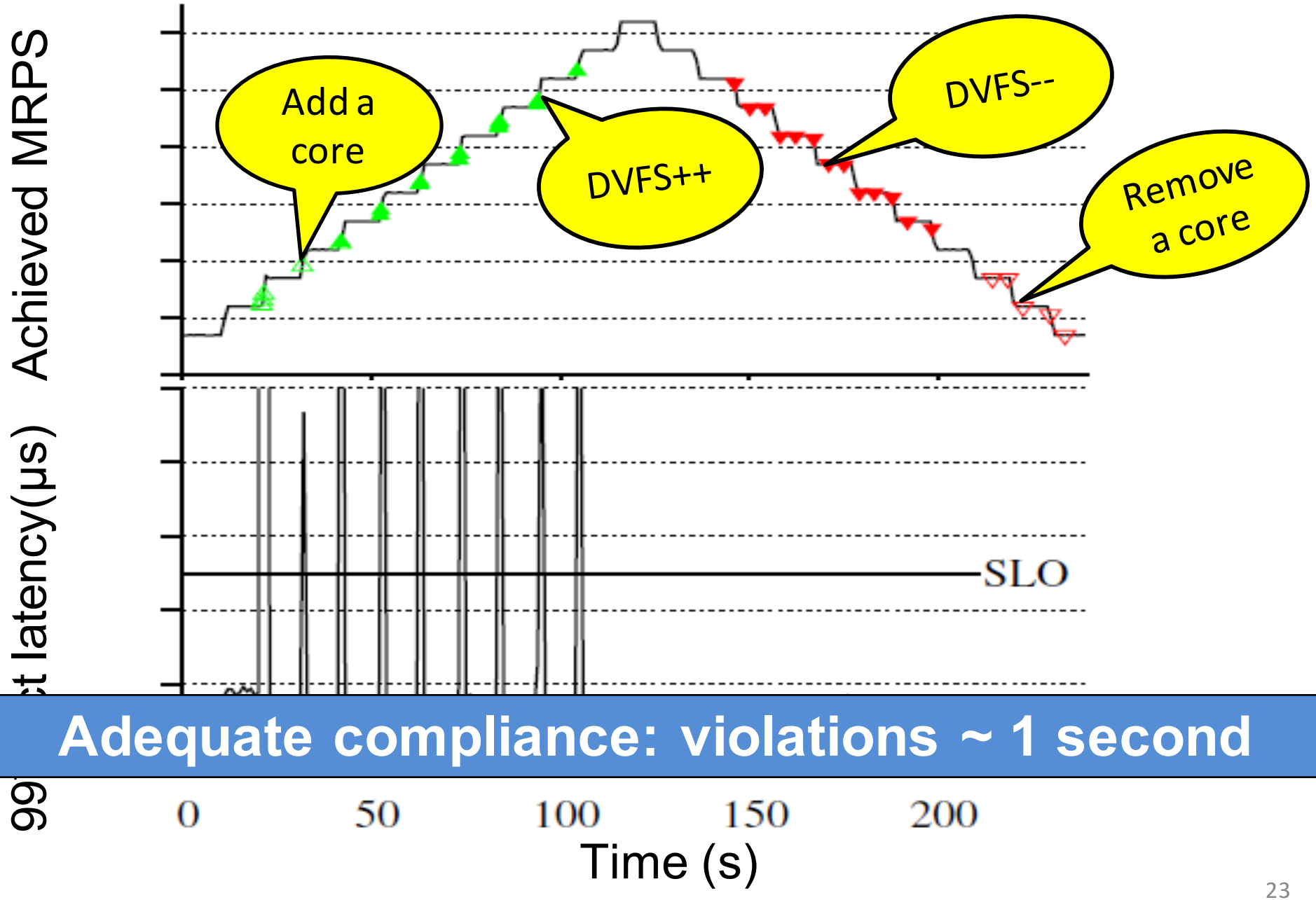
Experimental setup

- Latency-sensitive workload – memcached
  - **Energy prop.**; workload consolidation
- 3 demanding synthetic load patterns:
  - *slope*, ***step***, *sine+noise*
  - *4min cycle time*

- 10 load generating clients + 1 latency measuring client @1 second intervals

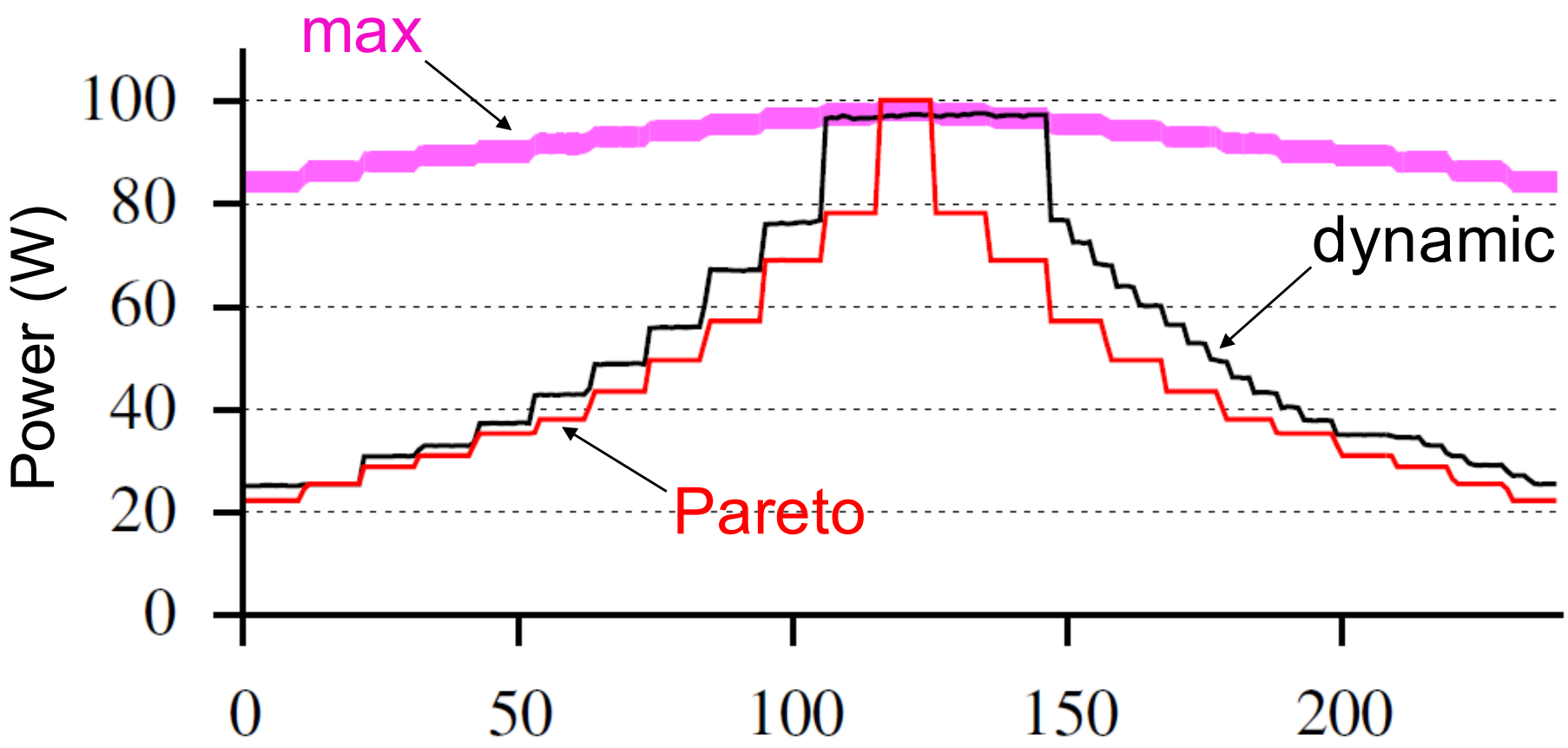
- 2752 connections; Poisson-distribution

Evaluation – step pattern



Adequate compliance: violations ~ 1 second

Evaluation – step pattern



Average: max=91W Dynamic=48W Pareto=41W

Evaluation

		Slope	Step	Sine+noise
lity	Max. power	91	92	94
Pro	Pareto optimal	39	41	45
on ty k)				

Saving 44%-54% of processor energy →
85%-93% of Pareto-optimal bound

Running background job at 32%-46% of their standalone throughput →
82%-92% of the Pareto-optimal bound

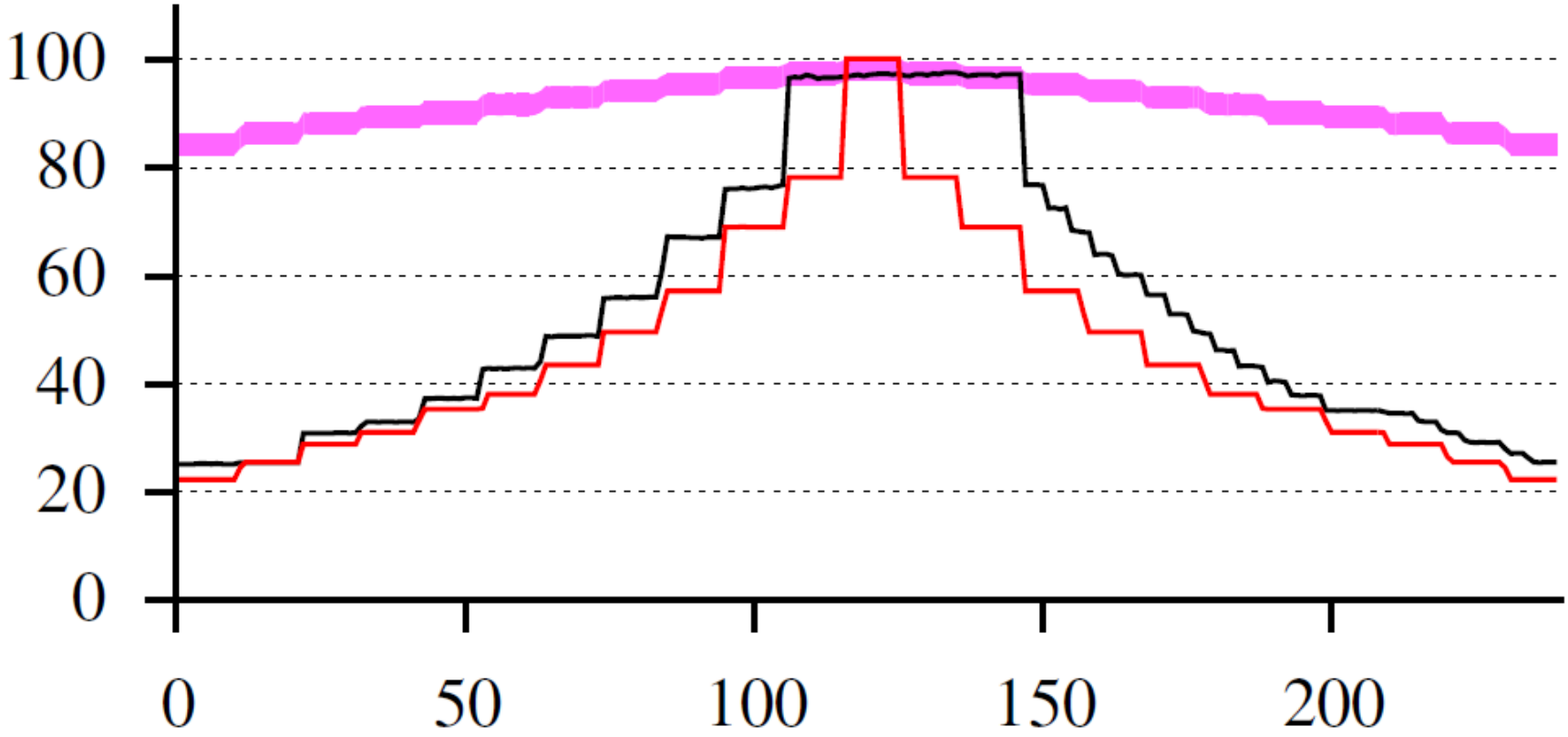
Conclusion

- Real challenges to latency-sensitive applications
  - Maintain service-level objectives **while**
  - Minimize energy consumption **or**
  - Maximize workload consolidation

- Design using Pareto methodology to determine theoretical bound and derive control policies

- Implement dynamic resource controls to IX dataplane operating system

Thank you



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Stanford