Reconciling High Efficiency with Low Latency in the Datacenter

David Lo

PhD Oral Defense

April 7th, 2015

Fun facts about datacenters











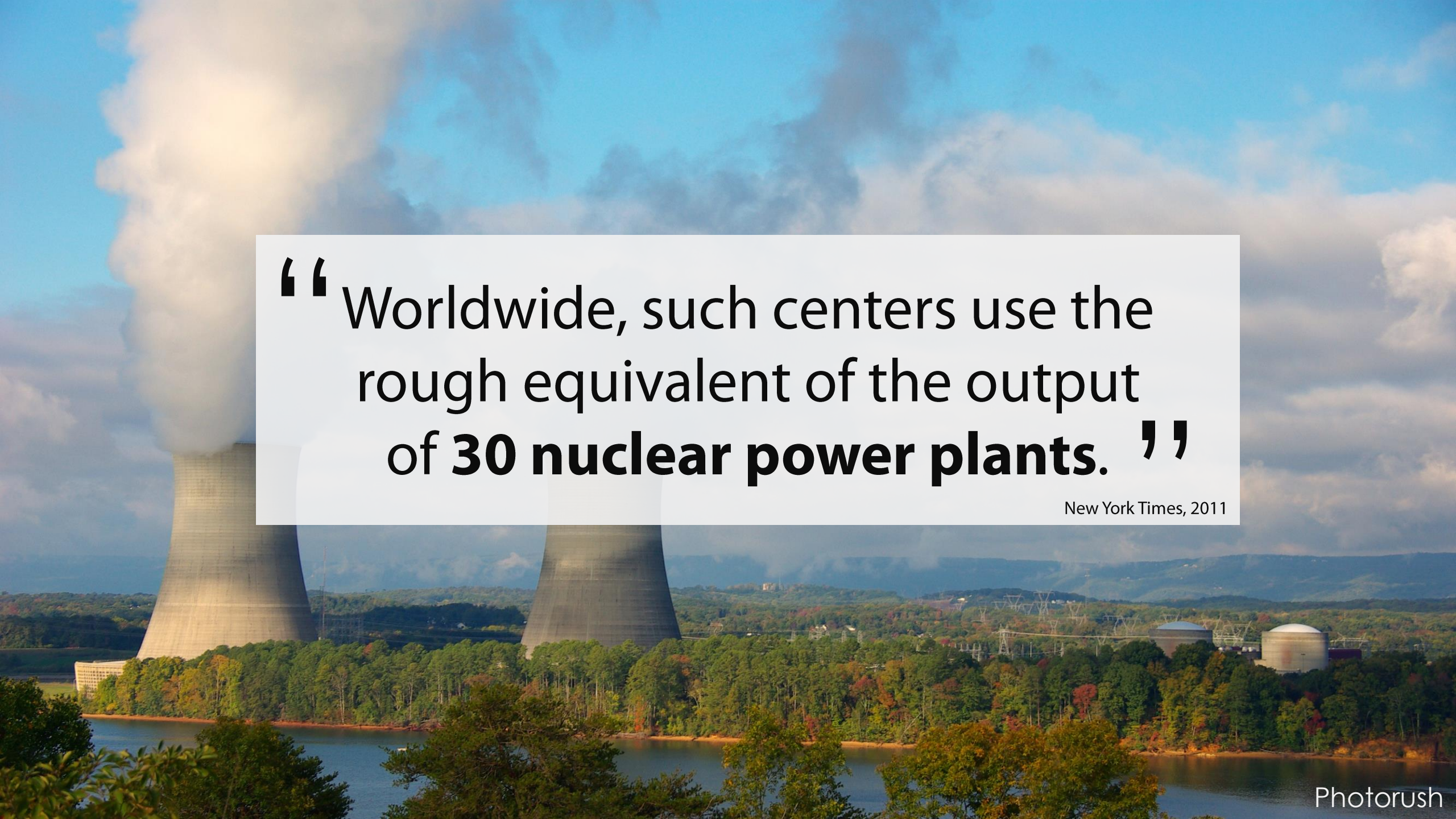



Microsoft Azure





“ Apple Inc. plans to invest **$2 billion** to build a data center… ”
Wall Street Journal, 2015
Google


“Worldwide, such centers use the rough equivalent of the output of **30 nuclear power plants**.”
New York Times, 2011
Photorush

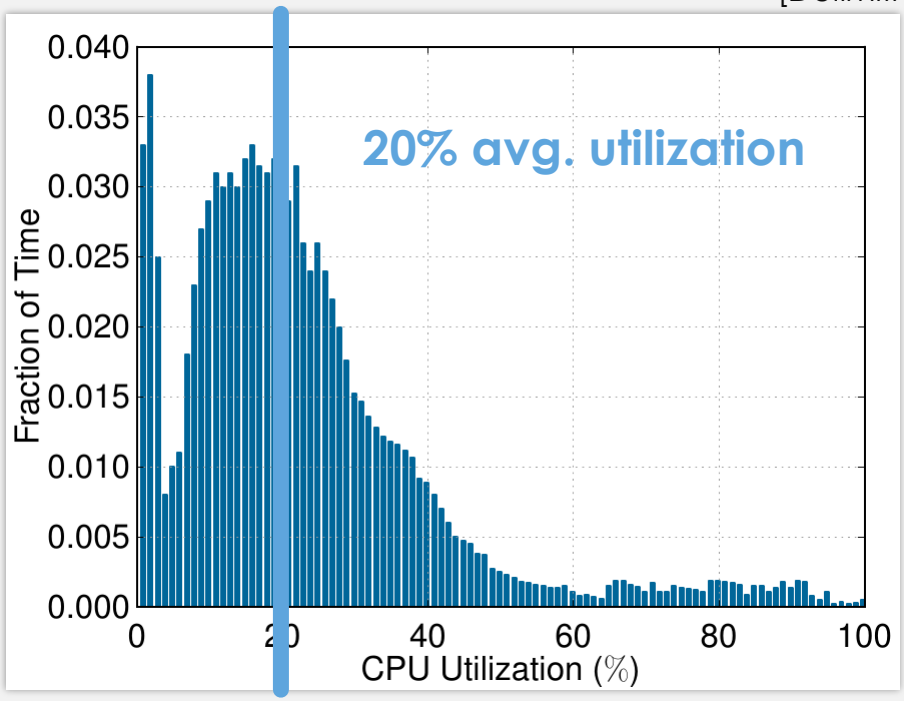

“ Overall data center workloads will nearly **double** from 2013 to 2018 ”
Cisco, 2013
Google

Potential roadblocks for future growth

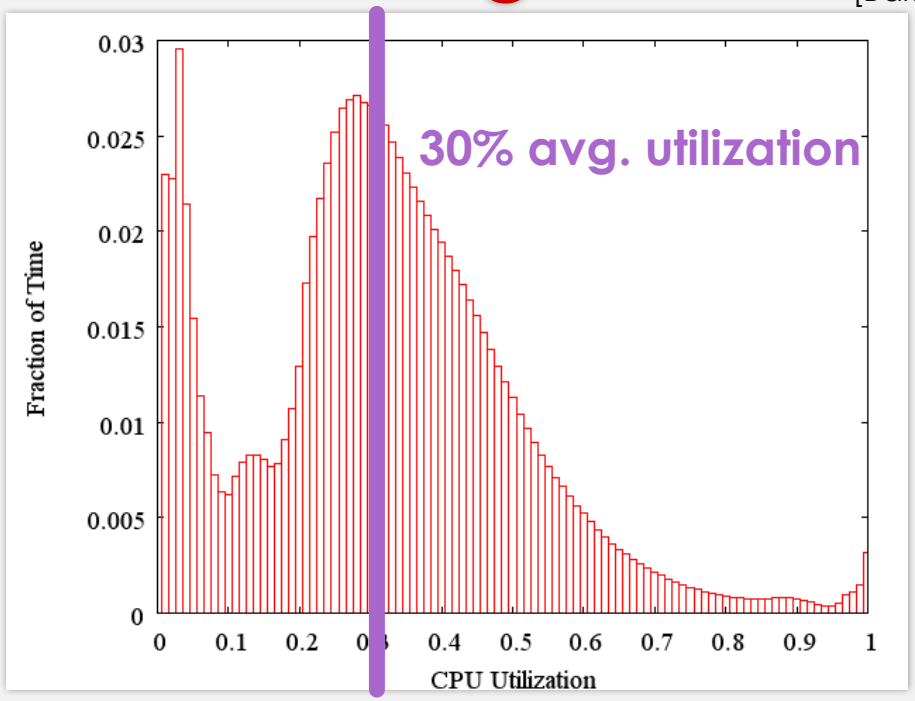
- Cost reduction
  - ~~Switch to commodity servers~~ — One time trick
  - ~~Improved power delivery & cooling~~ — Diminishing returns
- Capability scaling
  - ~~More datacenters~~ — >$2B per DC
  - ~~More servers per datacenter~~ — @60MW per DC
  - ~~CPU vendors will make better chips~~ — End of voltage scaling

But the datacenters are poorly utilized!

Twitter [Delimitrou'14]

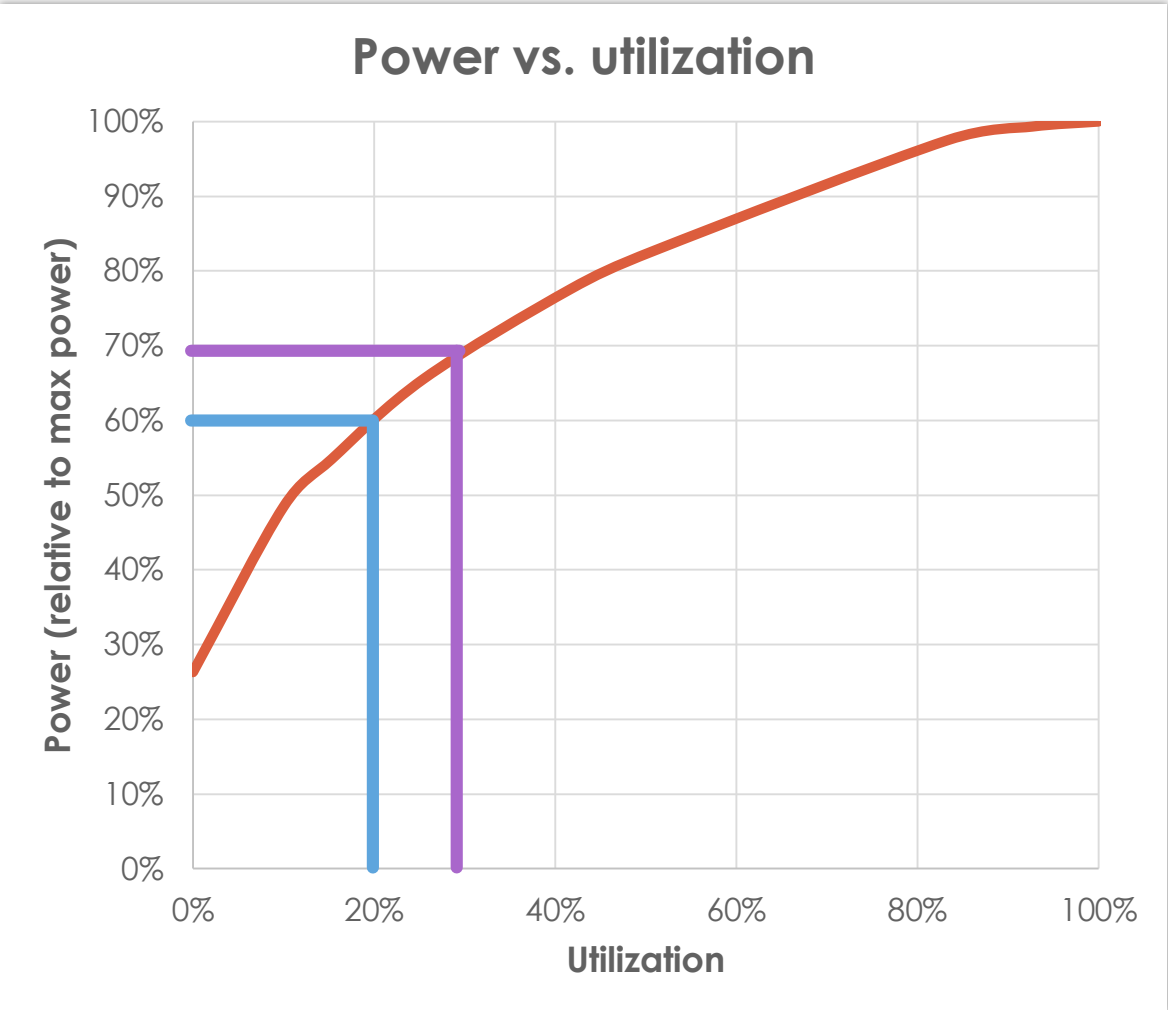


Google [Barroso'09]



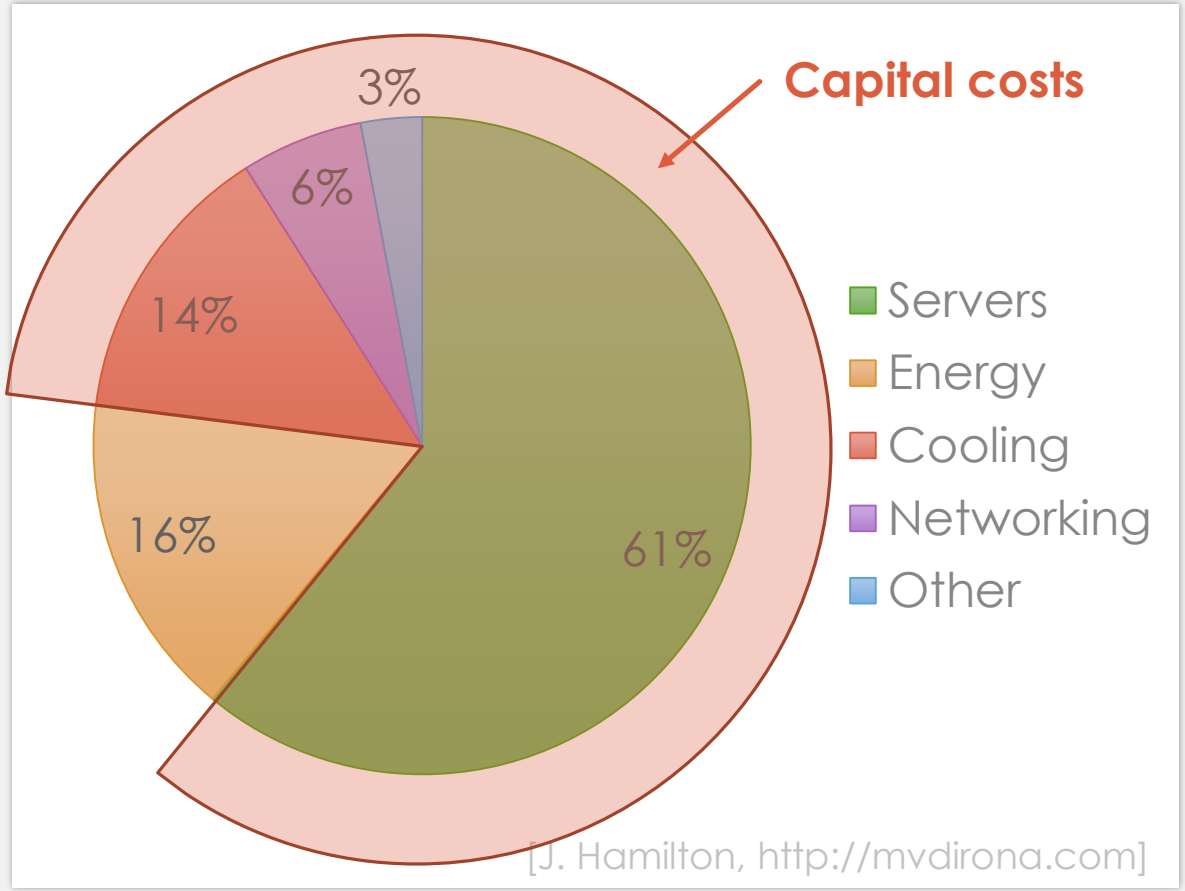
- Low utilization in large-scale clouds, even with automated management systems

Which is bad for energy efficiency



- Servers still use lots of power when not running at peak
- Google: 70% peak power
- Twitter: 60% peak power
- Using 30 nuclear plants, but should only be using 10-15

Bad for cost efficiency



- Most of the costs are in capital equipment
- Google: 59% wasted $
- Twitter: 67% wasted $
- > $1B potential waste on one datacenter

What datacenters need to do better

- Better energy efficiency
  - Lessen/eliminate wasted power at lower utilizations
- Better resource efficiency
  - Improve utilization in the datacenter

Outline

- Causes of low efficiency in the datacenter

- My contributions towards improving efficiency
  - Autoturbo: improving energy efficiency [HPCA '14]
  - OLDIsim: scale-out benchmark [Released by Google]
  - PEGASUS: improving energy efficiency [ISCA '14]
  - Heracles: improving resource efficiency [To appear ISCA '15]

Causes of low efficiency in datacenters

The applications running in a datacenter

- Batch workloads
  - Examples: analytics, log processing, training neural networks
  - Throughput oriented

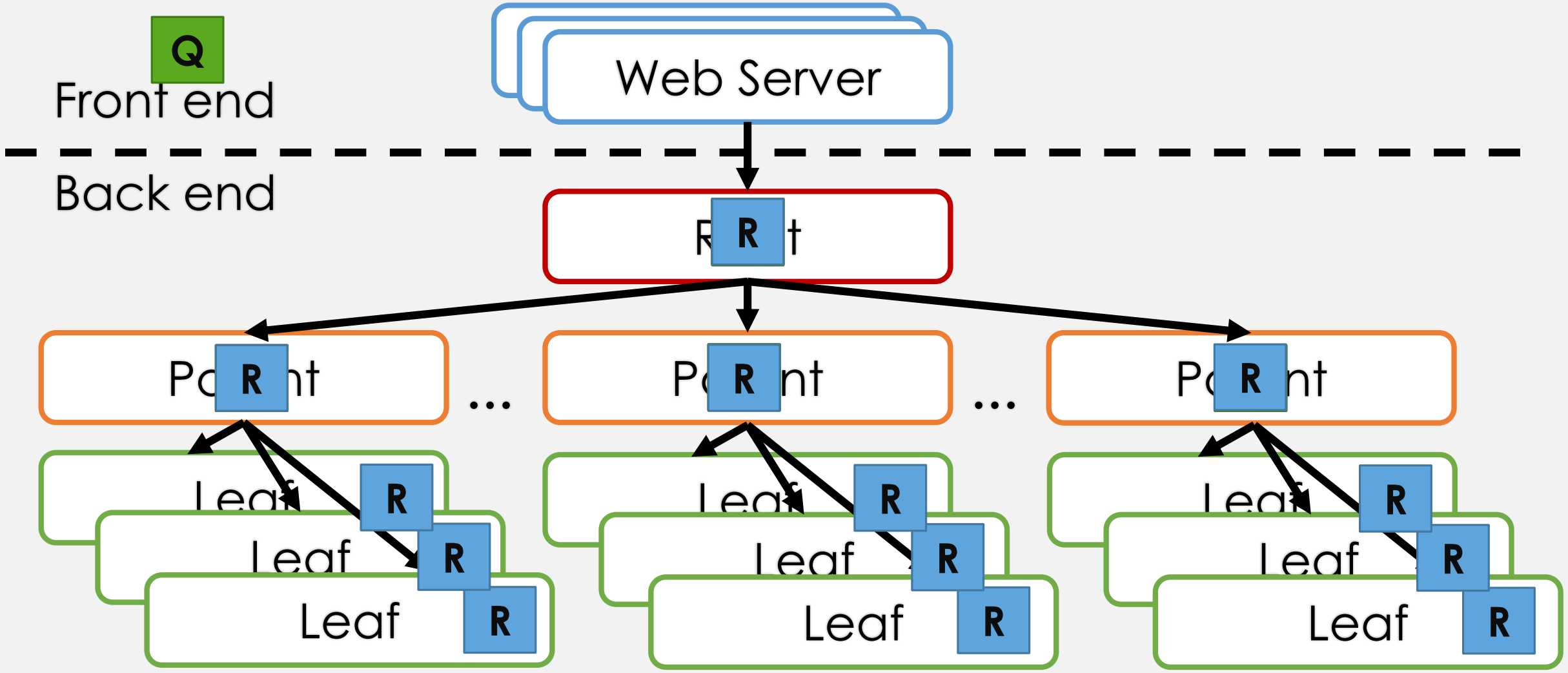
- Latency-critical workloads
  - Examples: Google.com, Facebook.com, Twitter.com
  - User-facing, extremely sensitive to latency

Largest class of latency-critical workloads

- On-line Data Intensive (**OLDI**) workloads are user-facing workloads that mine massive datasets across many servers
  - Strict Service Level Objectives (**SLO**): e.g. 99%-ile tail latency is 5ms
  - High fan-out with large distributed state

- A substantial class of workloads in the datacenter
  - Driven by growth in web services and cloud computing
  - Google, Facebook, Twitter, RAMCloud, etc.

An example OLDI workload: websearch



Why is utilization so low?

- **Underutilization is structurally ingrained in OLDI workloads**

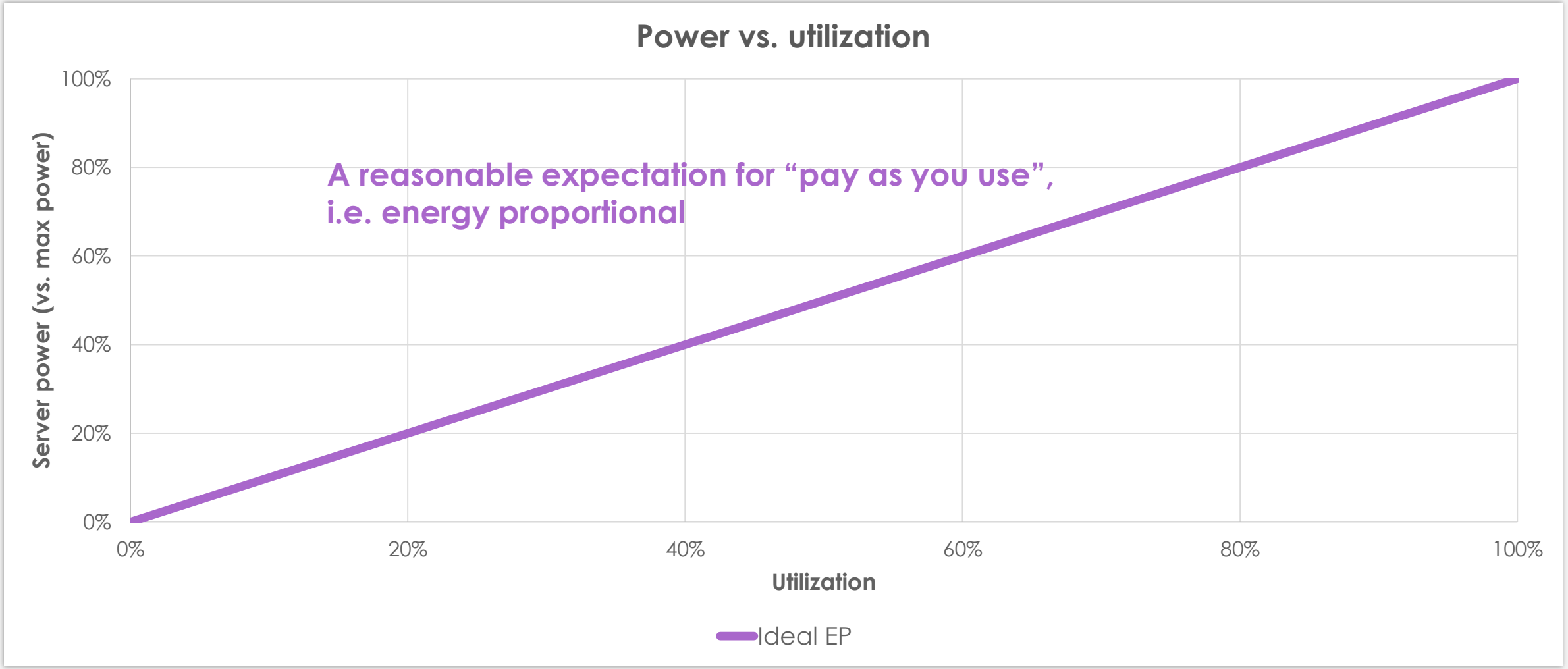
- Diurnal user traffic (e.g. users are sleeping at 3am)
- Planning for high traffic events (e.g. Black Friday)

} Fundamental

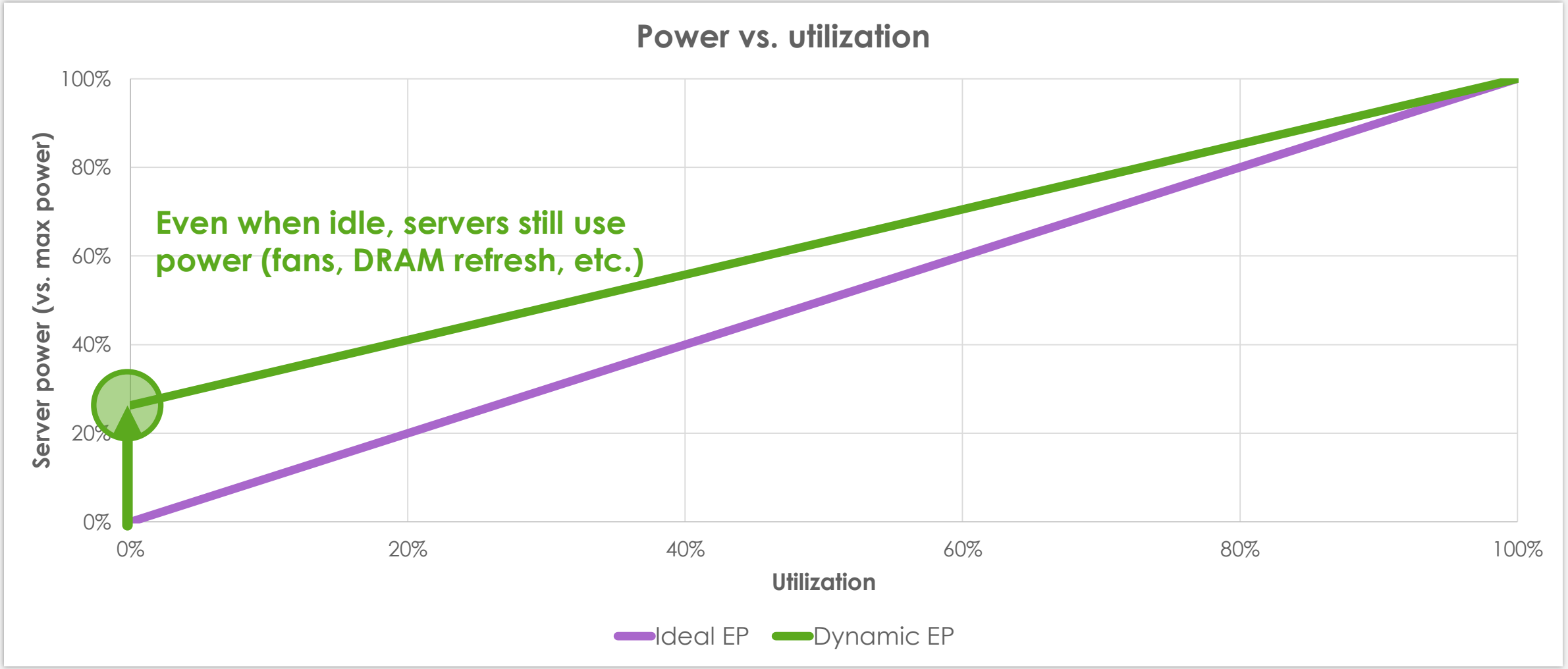
- OLDI workloads are run on dedicated servers

} By design

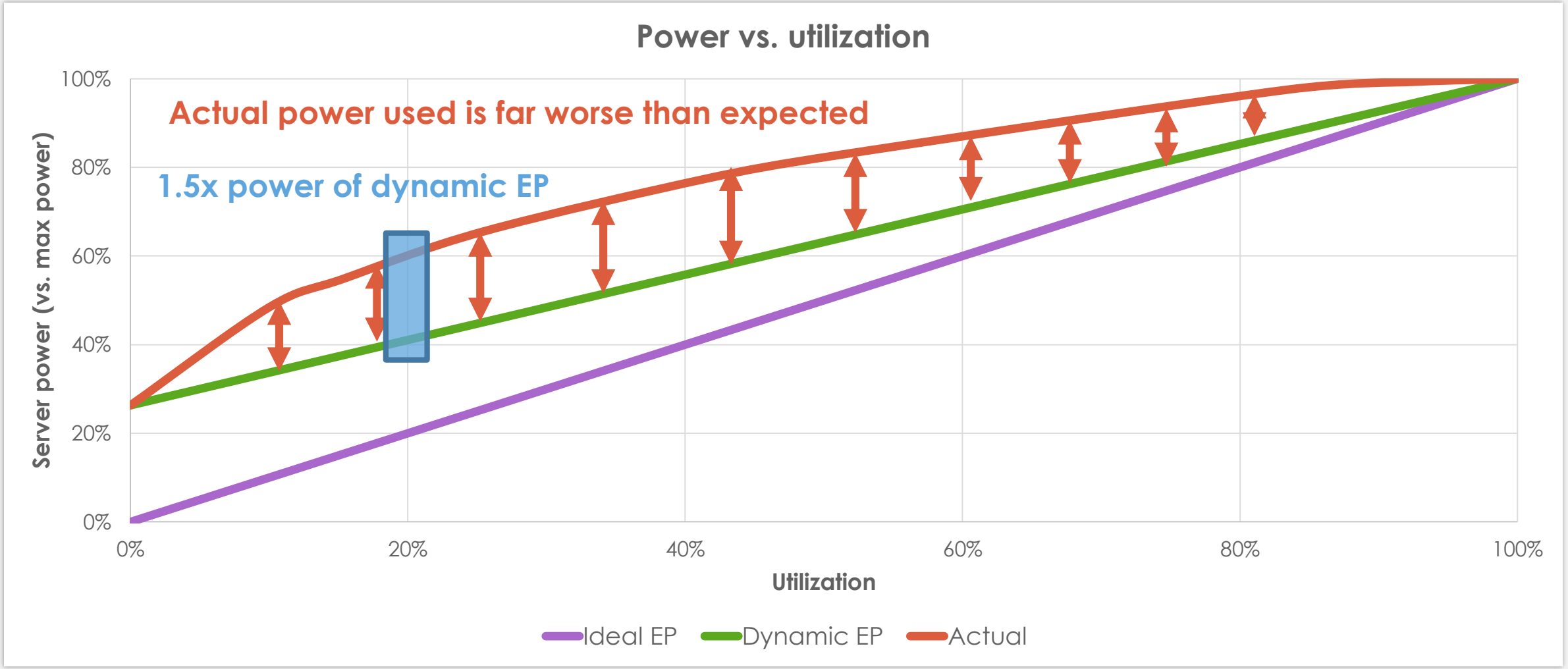
Poor energy efficiency at low utilizations



Poor energy efficiency at low utilizations



Poor energy efficiency at low utilizations



“Common wisdoms” regarding efficiency

- Stuck with low server utilization and poor energy efficiency

- **Claim 1**: hard to improve energy efficiency
  - Power management techniques cause latency violations
- **Claim 2**: hard to increase utilization
  - Co-locating workloads also causes latency violations

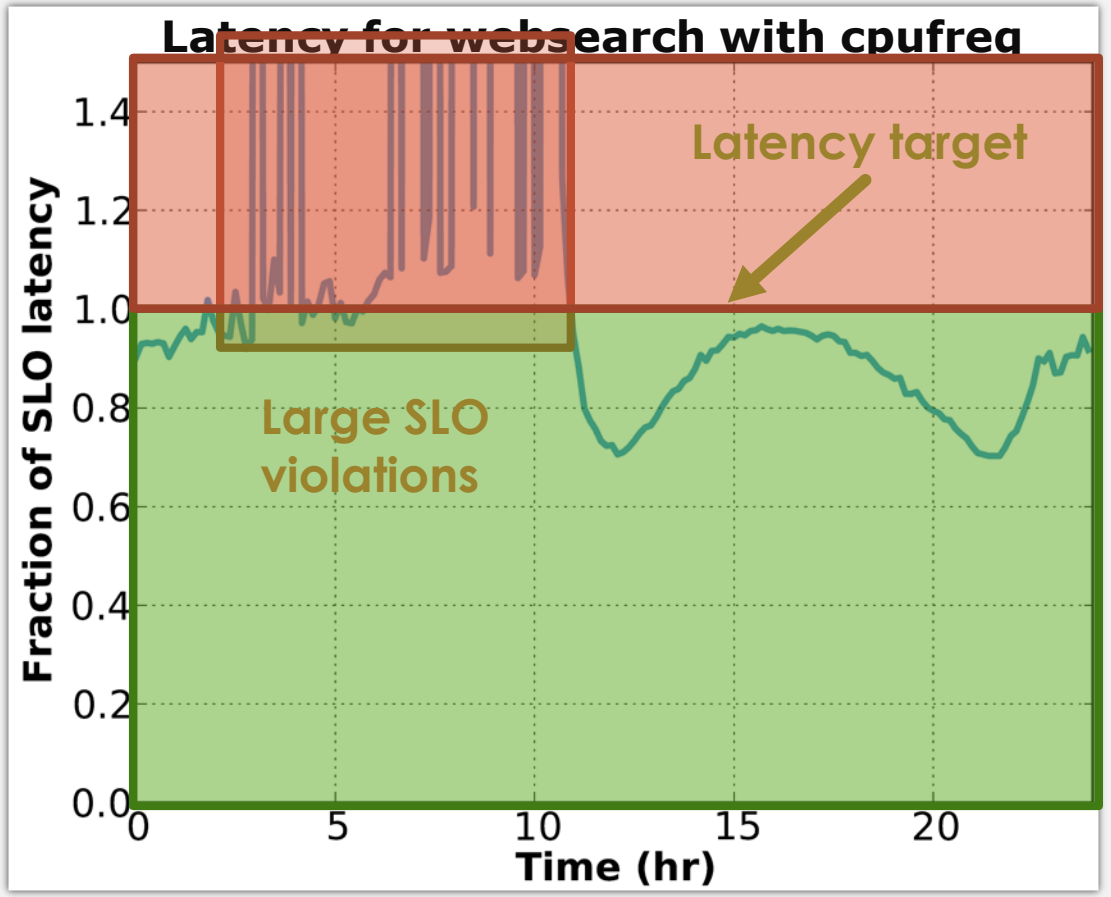
Claim 1: hard to improve energy efficiency

- Many proposed approaches for batch/real-time workloads do not work for latency-critical workloads

- Batch: system idle modes [PowerNap ASPLOS'09, Blink ASPLOS'11]
  - Not enough system idleness, even at low loads
  - Creating idleness with batching violates SLO [Meisner ISCA '11]
- Batch/real-time: power scaling [Isci MICRO'06, Pillai SOSP'01]
  - Mismatch between needs of real-time/batch and latency-critical

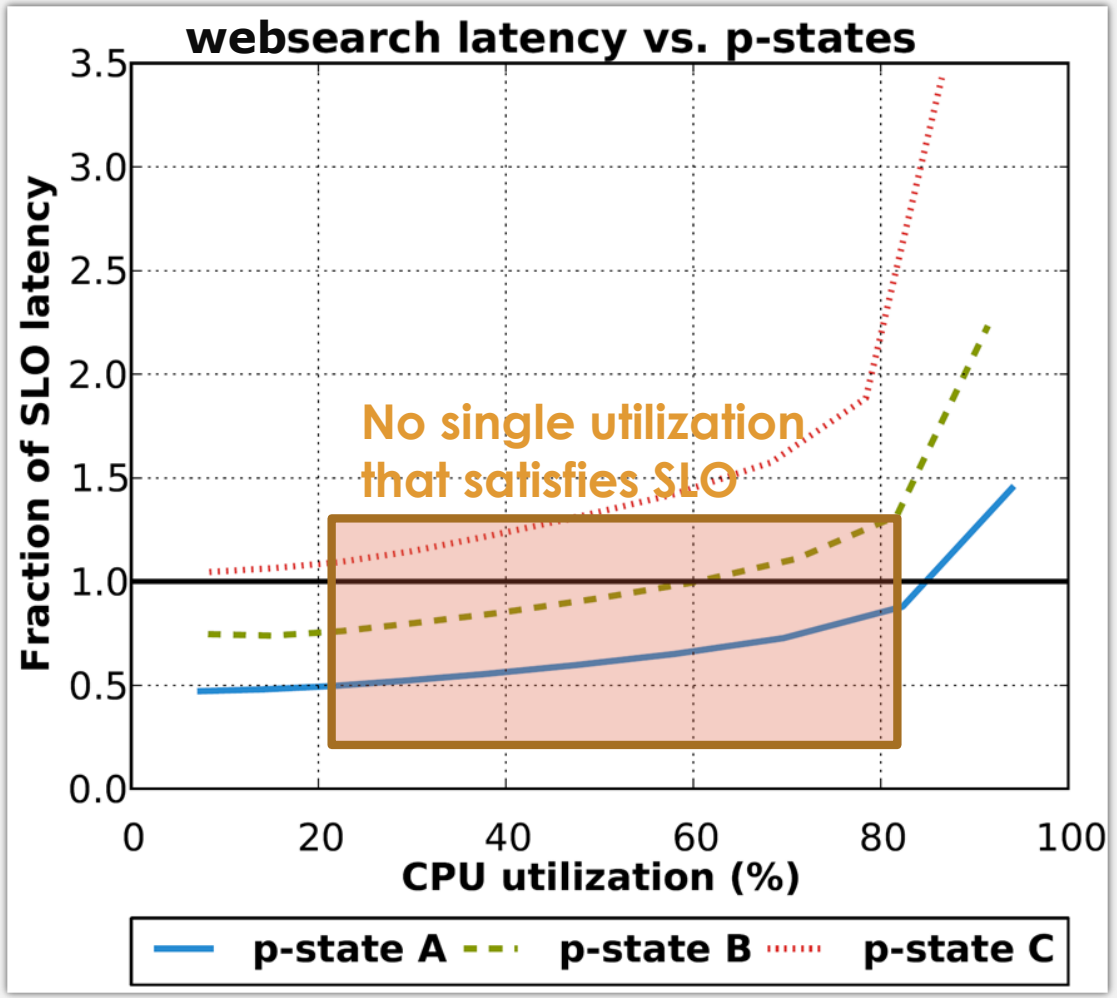
Power scaling is not suitable

- Cpufreq power governor:
  - Uses CPU utilization to determine p-state
  - Keeps utilization at 20-95%
- Run Google websearch with cpufreq power governor and measure latency

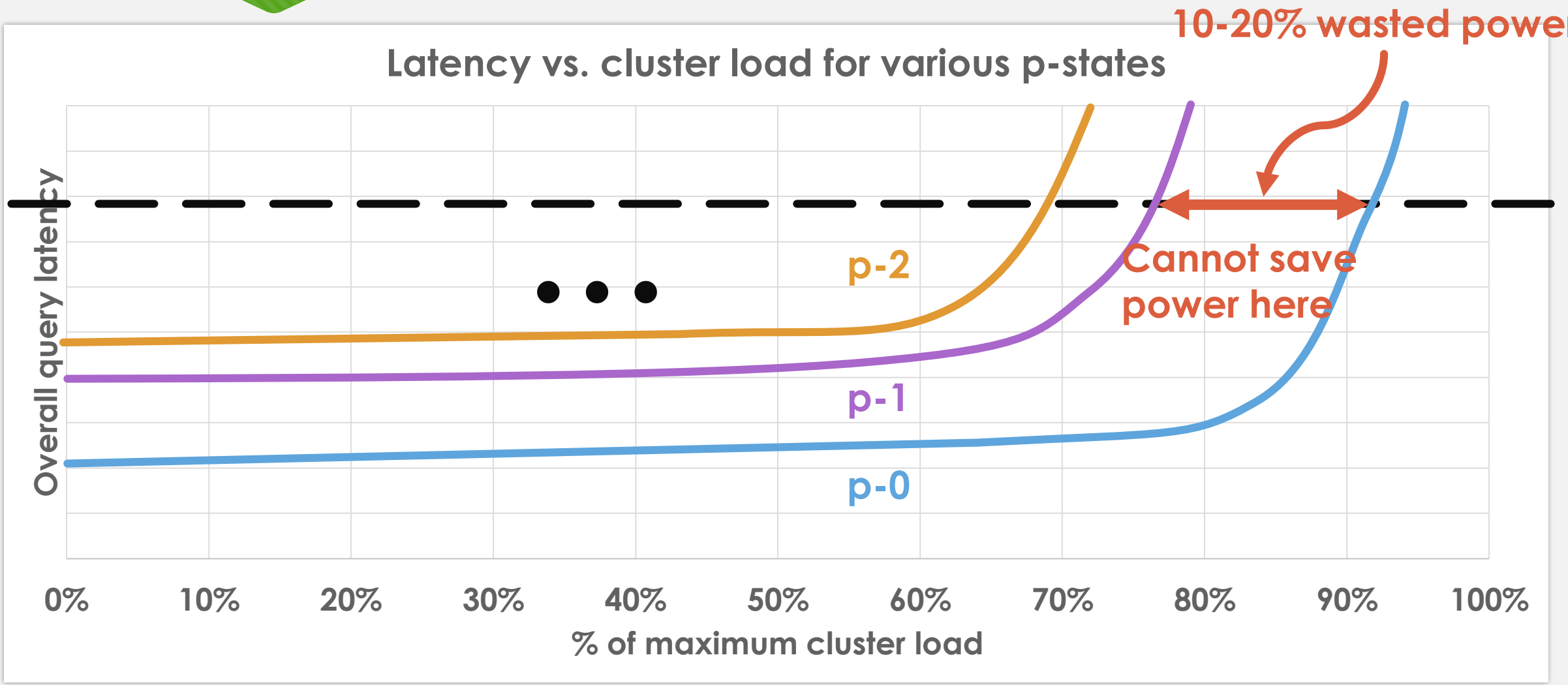


Why all cpufreq-like schemes won't work

- CPU utilization is a poor proxy for workload latency
- In the most extreme case: even a CPU utilization of 10% still leads to SLO violation



p-states are not fine grained enough

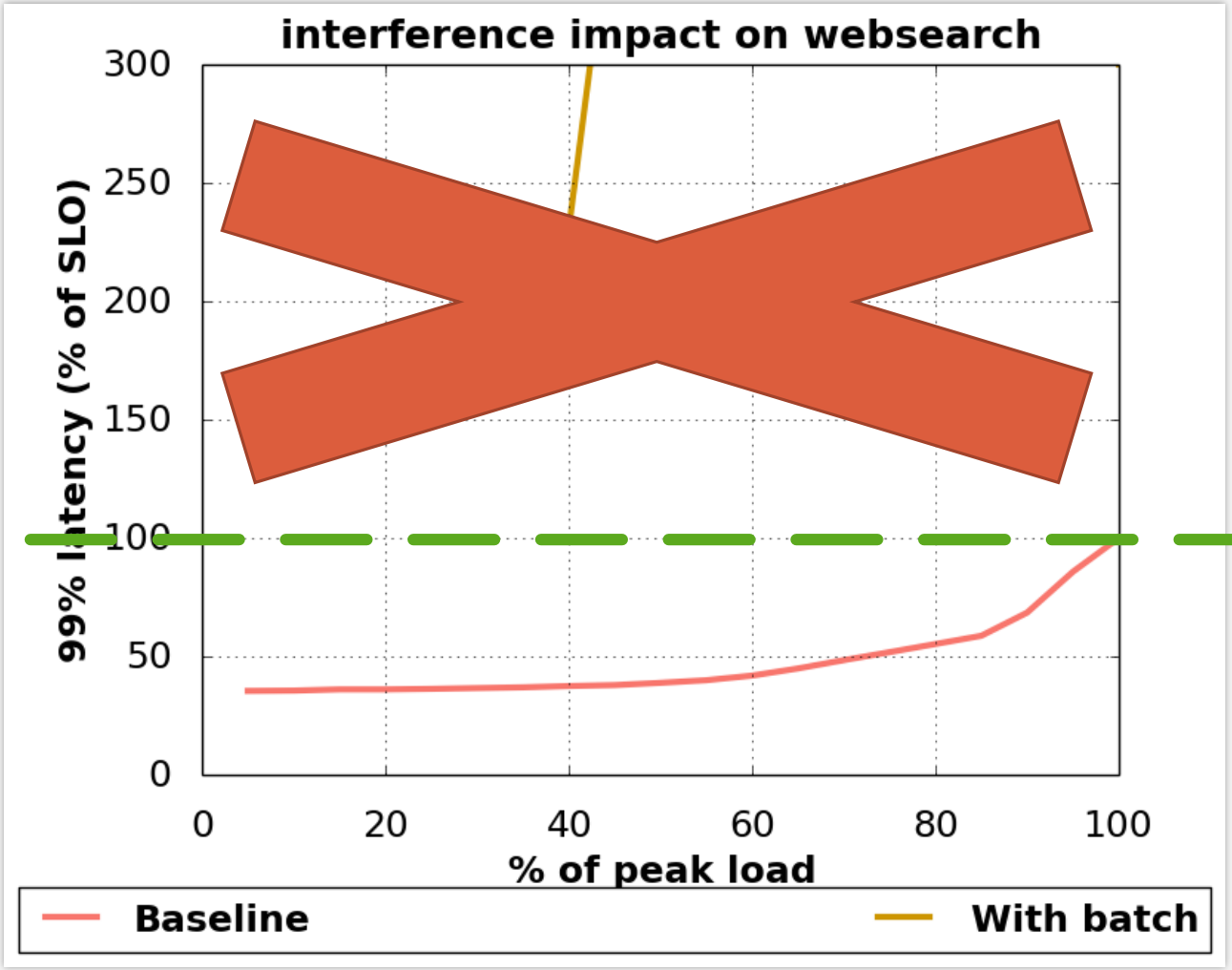


Claim 2: hard to increase utilization

- **Proposal**: during periods of low utilization, fill in spare capacity with extra work (oversubscription)
  - Works for batch workloads

- Unfortunately, latency-critical workloads do not mix well with other workloads
  - Interference on shared resources can cause SLO latency violations

Interference impact on Google websearch



SLO latency

Cannot co-locate workload at any load!

Want to have our cake and eat it too

- "Common wisdom" suggests that low latency and high energy efficiency and utilization are incompatible

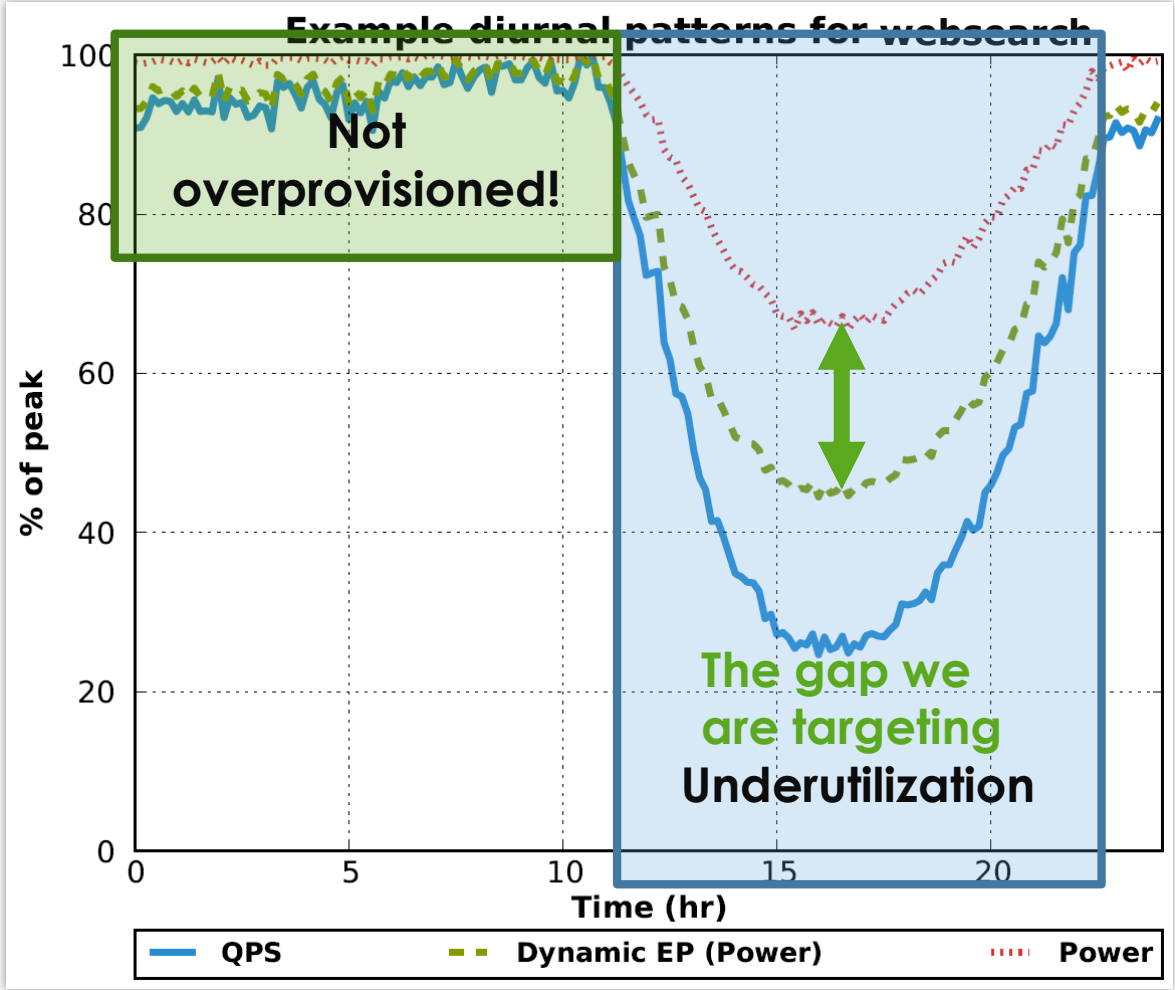
- Show that this is not the case, one step at a time
  - PEGASUS: energy efficiency
  - Heracles: high utilization

PEGASUS

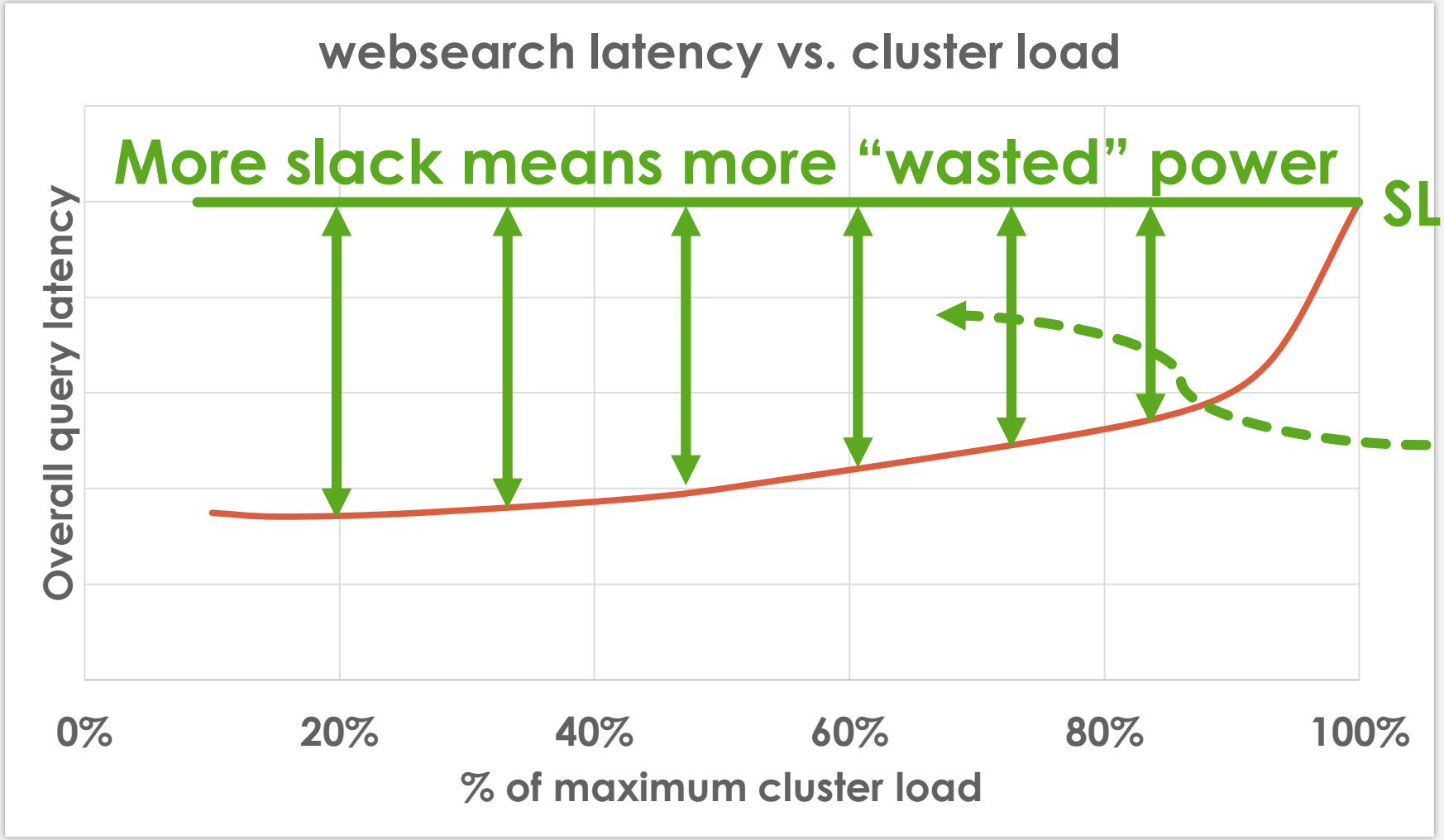
Reconciling low latency with energy efficiency

Opportunities for better energy efficiency

- Diurnal variation in cluster load and power for search across a 24 hour period
- Cluster not fully utilized half the time
- Gap between measured power and EP curves represent potential savings



The extra power is hiding in the latency



Too much latency slack is harmful

- **Beating the end-to-end SLO is no better than meeting it**
  - The end-user only cares if the web page takes a long time to load
  - If the page loads in 0.25sec vs. 0.50sec, user does not notice

- **Better target: keep the latency the same at all utilizations (iso-latency)**

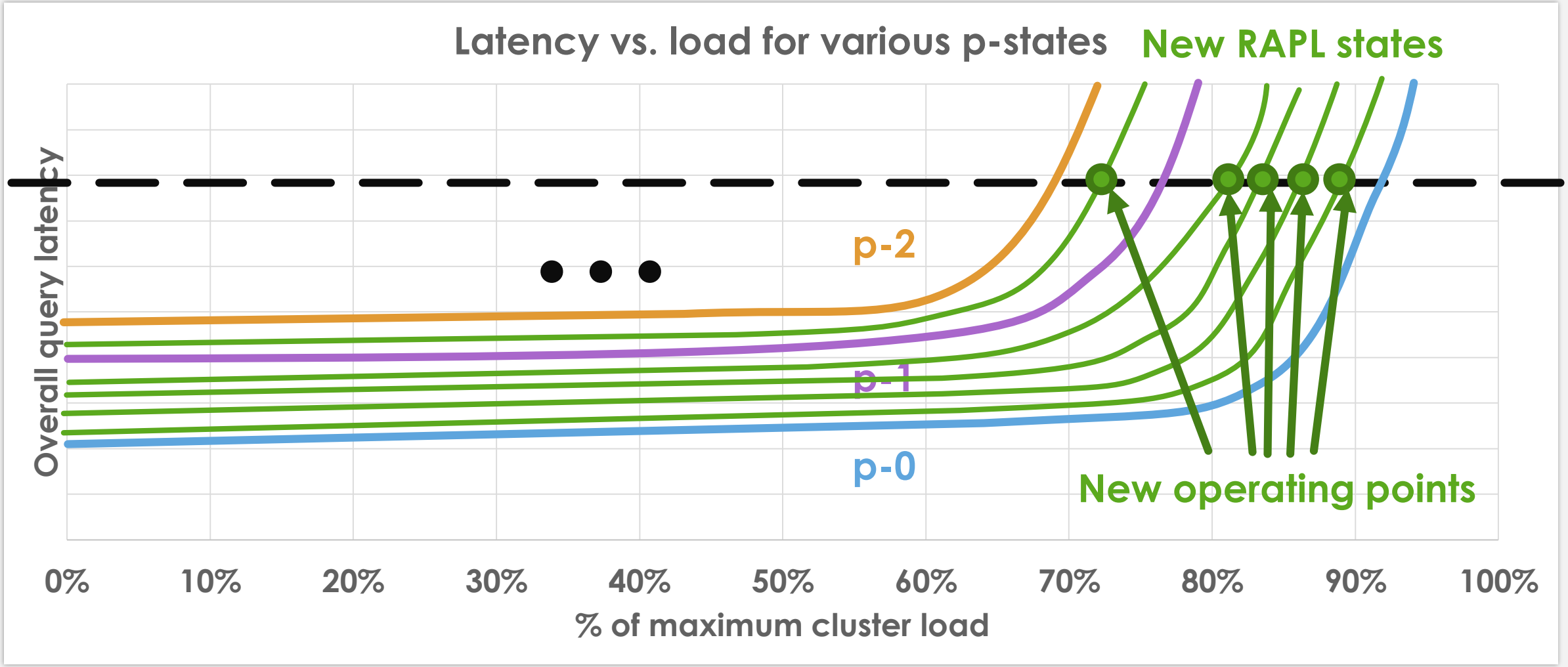
Iso-latency power management

- **Idea:** Trade end-to-end latency slack for power savings
- Use power management mechanisms to keep the workload performing just well enough to avoid SLO violations
  - **Need end-to-end latency feedback from workload**
    - Most OLDI workloads have ways of measuring this ✓
  - **Need fine-grained power management mechanisms**
    - ???

RAPL: a fine-grained mechanism

- RAPL: **R**unning **A**verage **P**ower **L**imit
- **Fine-grained**: power limit increments as small as 0.125W
- **Fast**: <1ms delay to apply new limit
- **Effective**: Dynamic Voltage Frequency Scaling (DVFS) behind the scenes to meet the power limit
  - More fine-grained than p-states
  - Can even modulate between multiples of base clock frequencies

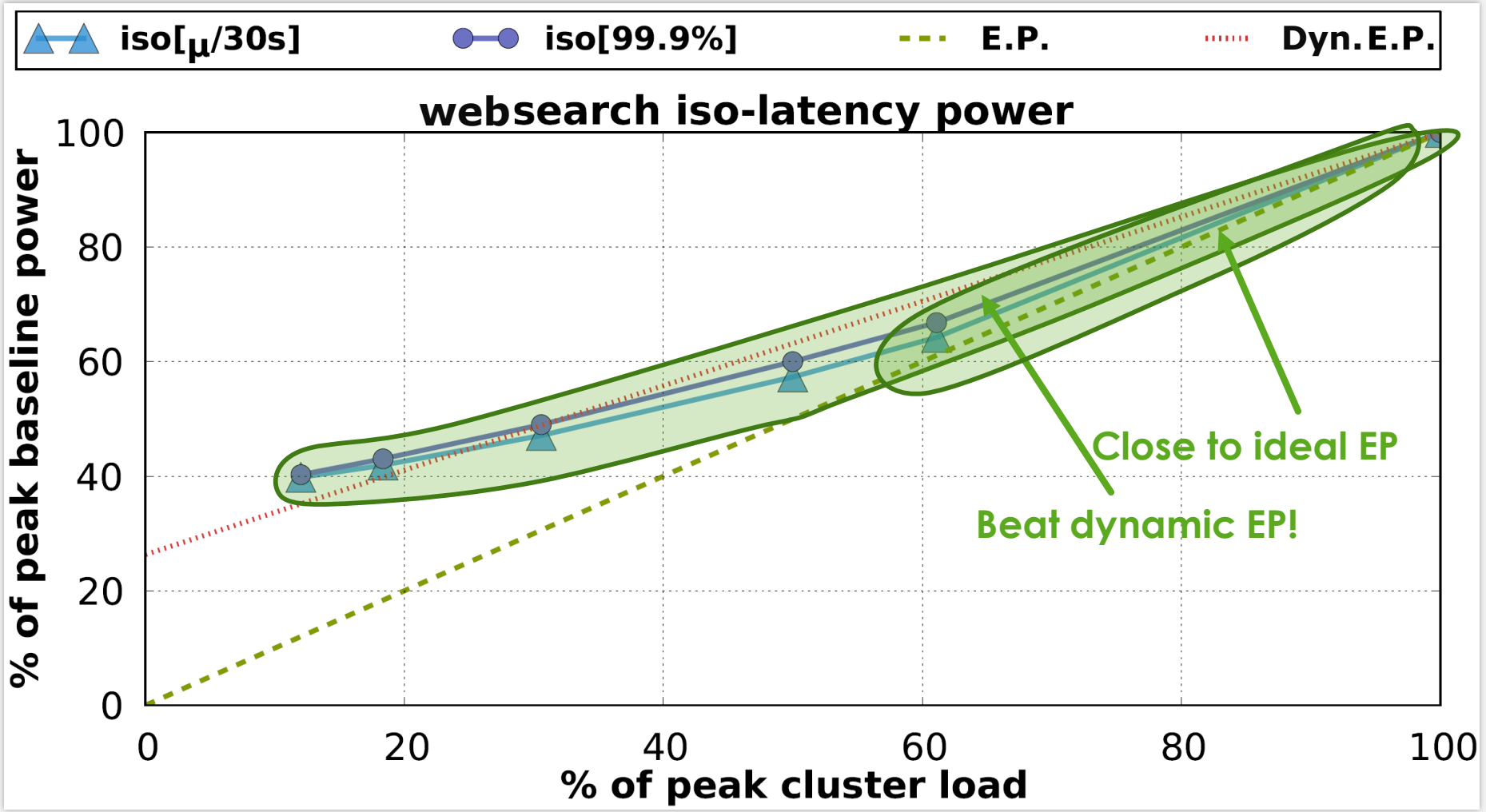
Advantages of fine-grain control



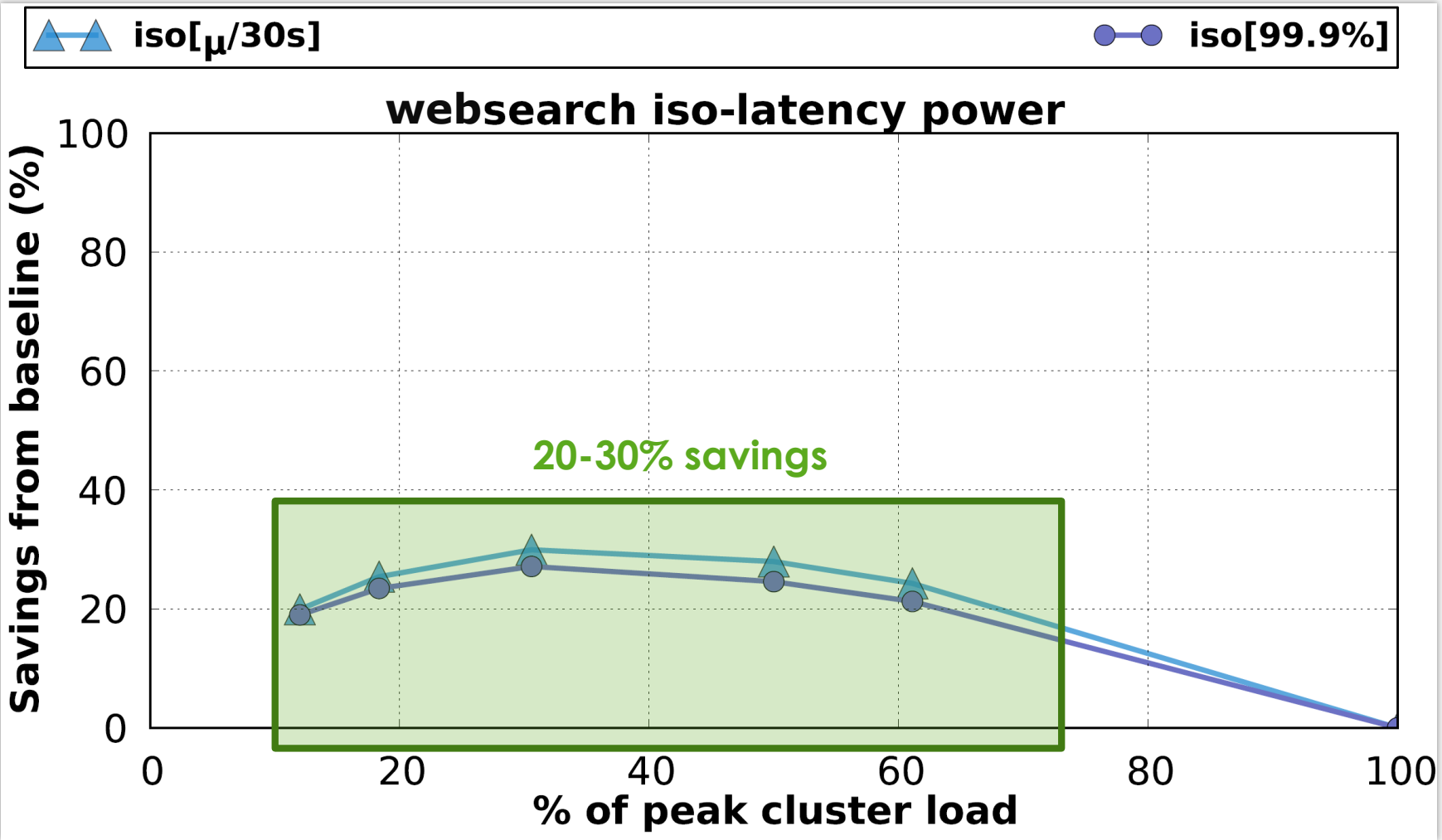
Evaluation of iso-latency potential

- Use static oracle to find lowest power that satisfies SLO
- Do this for several Google workloads
  - **websearch**: Google websearch
  - **memkeyval**: In-memory key-value store
- Measure **total system power** before and after iso-latency
- Do this for varying strictness of end-to-end SLOs
  - 30sec moving average, 99.9%-ile latency

Iso-latency potential: power

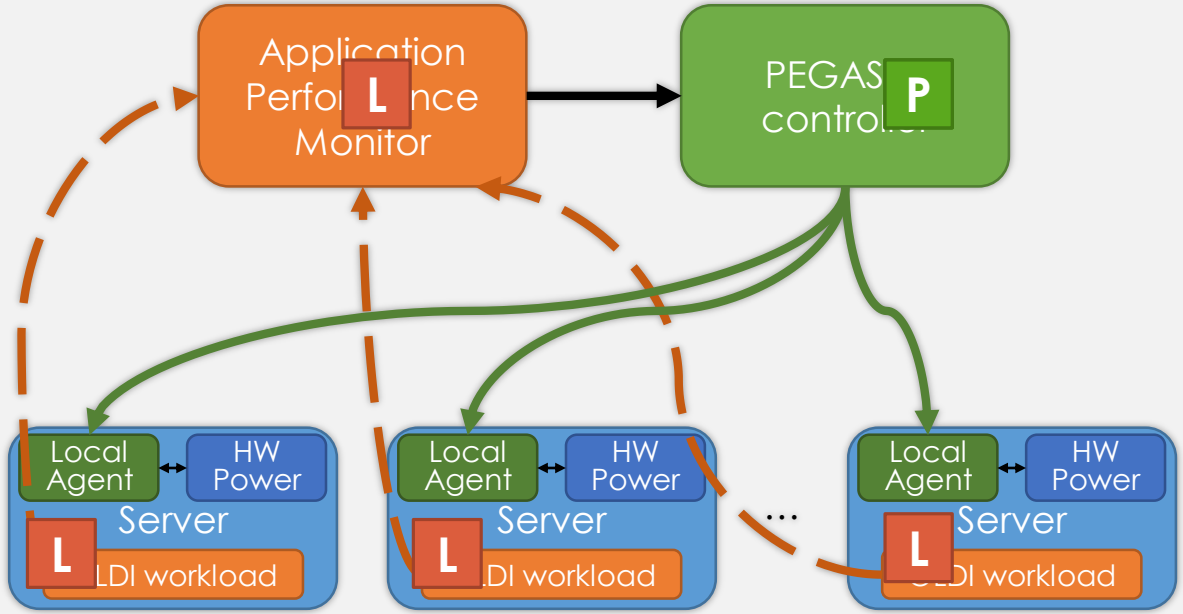


Iso-latency potential: power savings



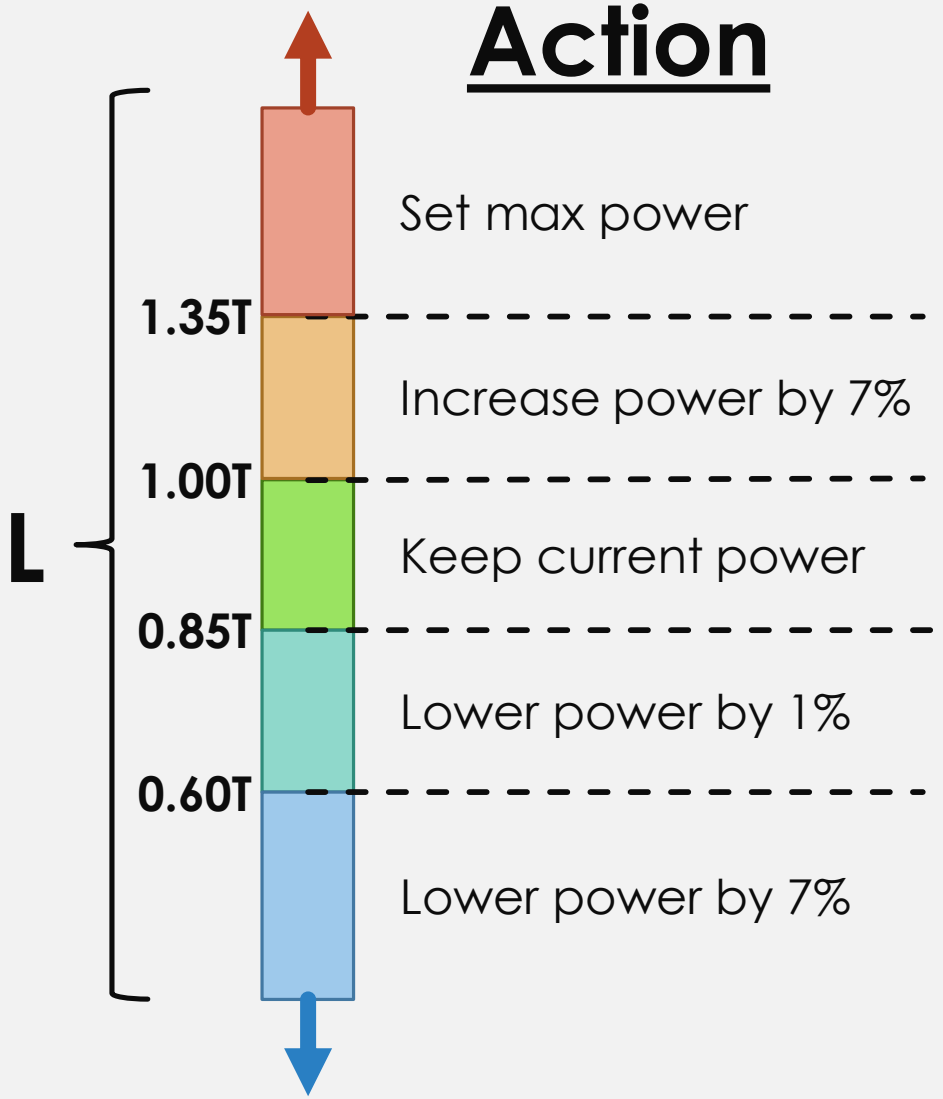
PEGASUS description

- Real-time dynamic controller for **iso-latency**
- Use RAPL as knob for power
- Measures latency slack and sets **uniform** power limit across all servers
- Power is set by workload specific policy
- ~500 lines of code



Example PEGASUS policy for websearch

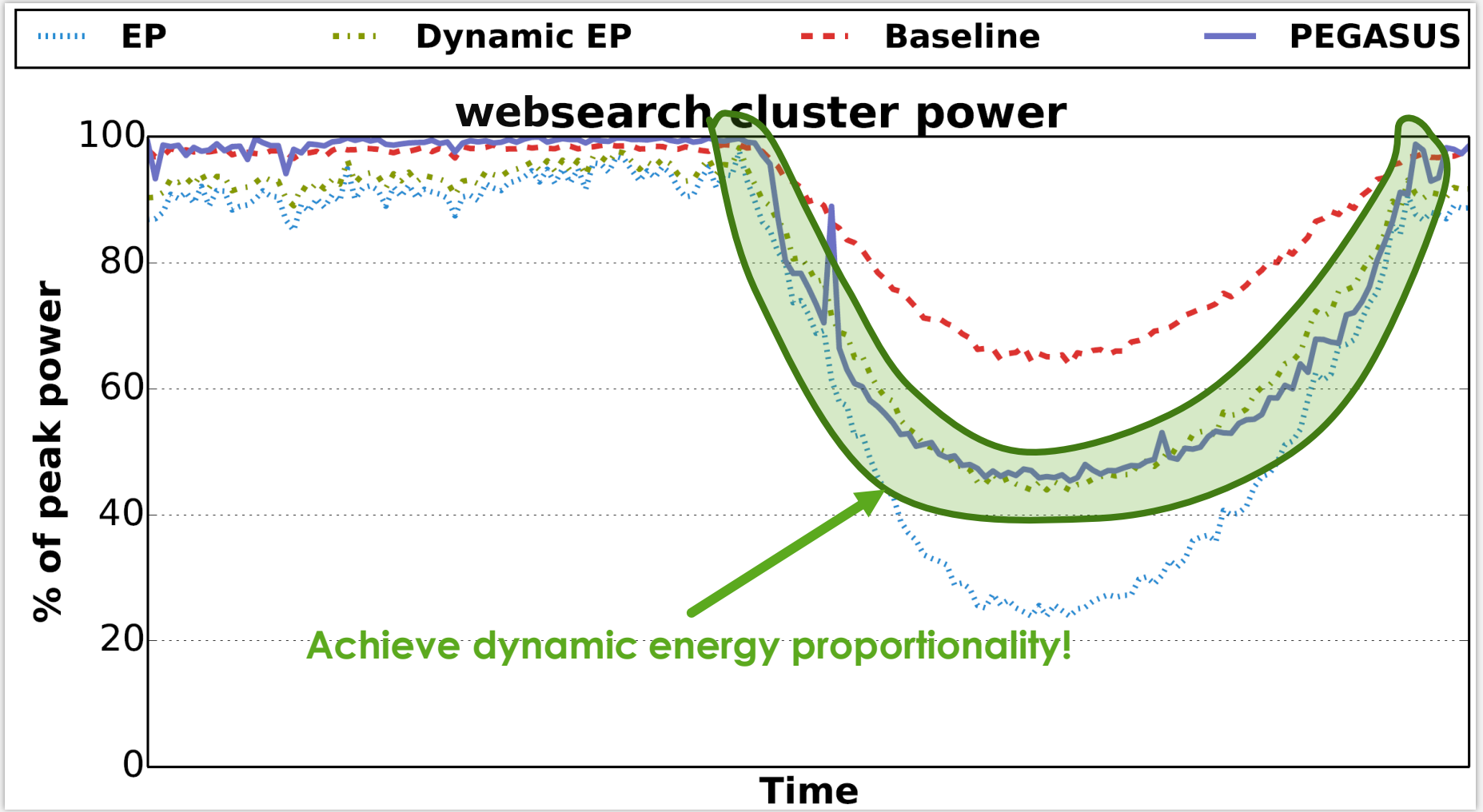
- **L** = Measured instant latency
- **T** = SLO target
- Use instant latency for quick corrections
- Violating SLO latency triggers fail-safe
- Constants determined through empirical optimization



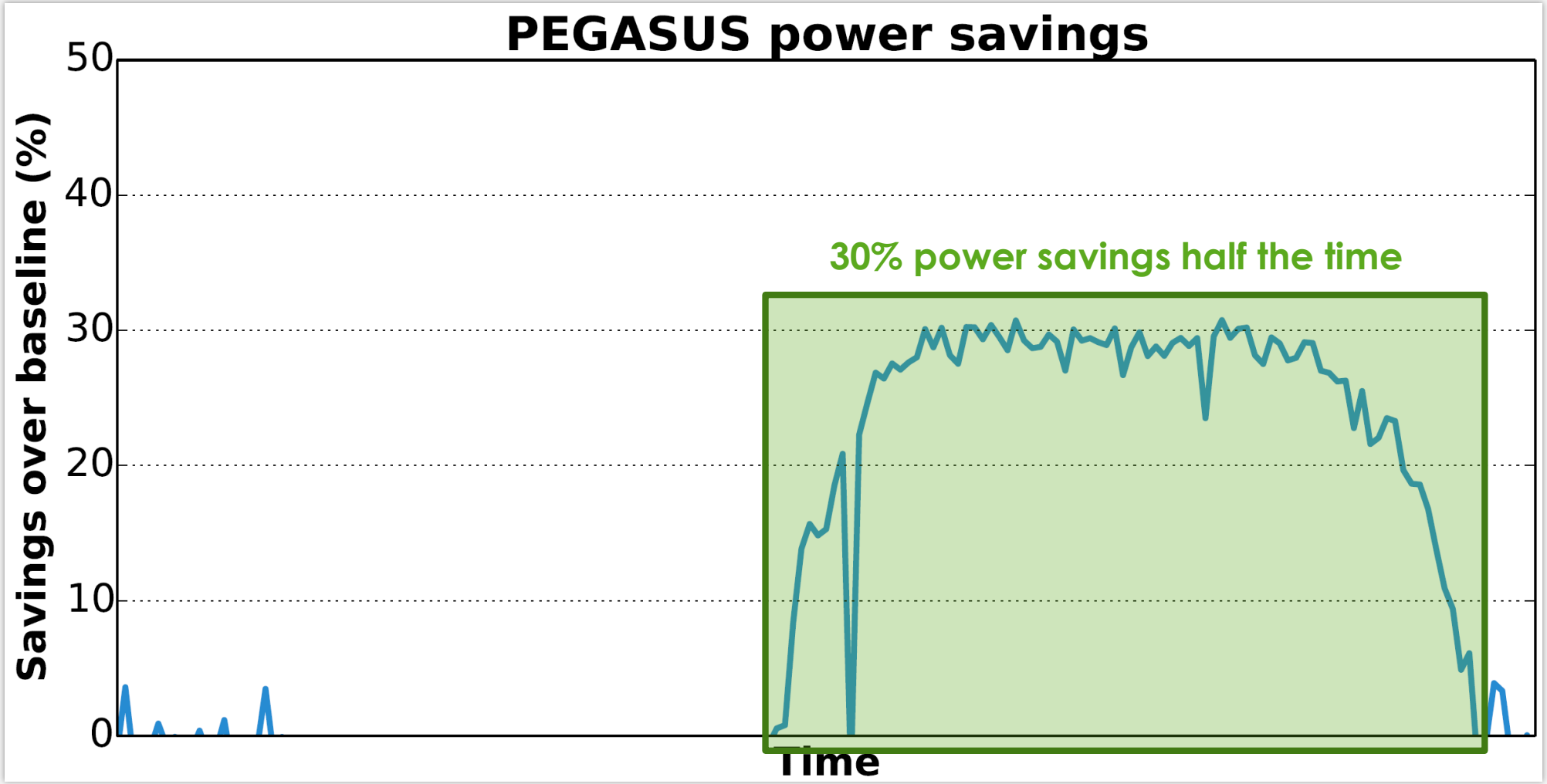
PEGASUS evaluation methodology

- Workload parameters
  - **SLO metric**: 30 second average latency [μ/30s]
  - Traffic pattern and user queries derived from anonymized search logs
  - Index derived from production search index
- Evaluate on several cluster sizes
  - **Small**: tens of machines, use full 24hr trace
  - **Production**: thousands of machines, use 12hr portion
- Measure full cluster power and SLO latency

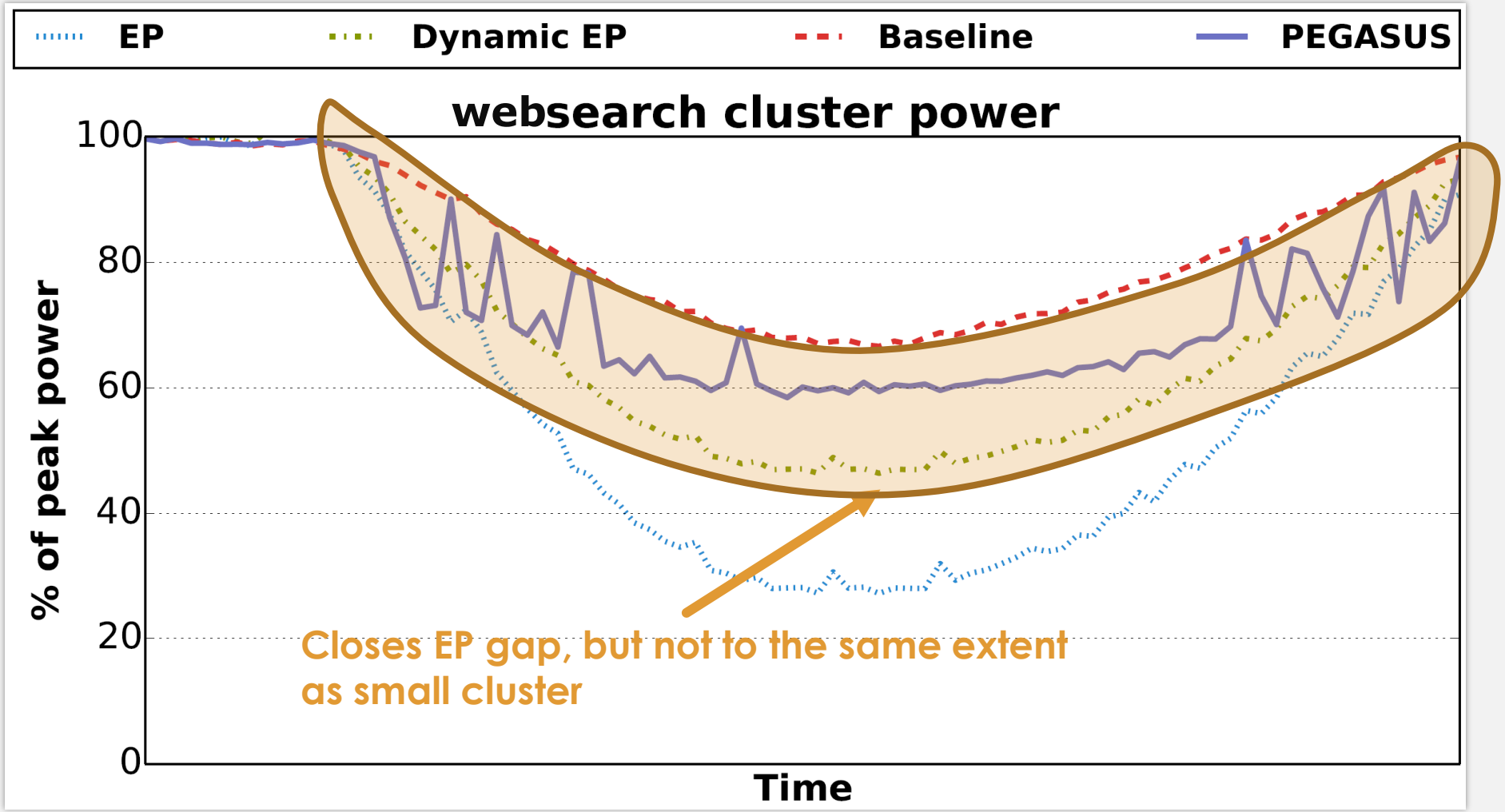
Small cluster results: power over time



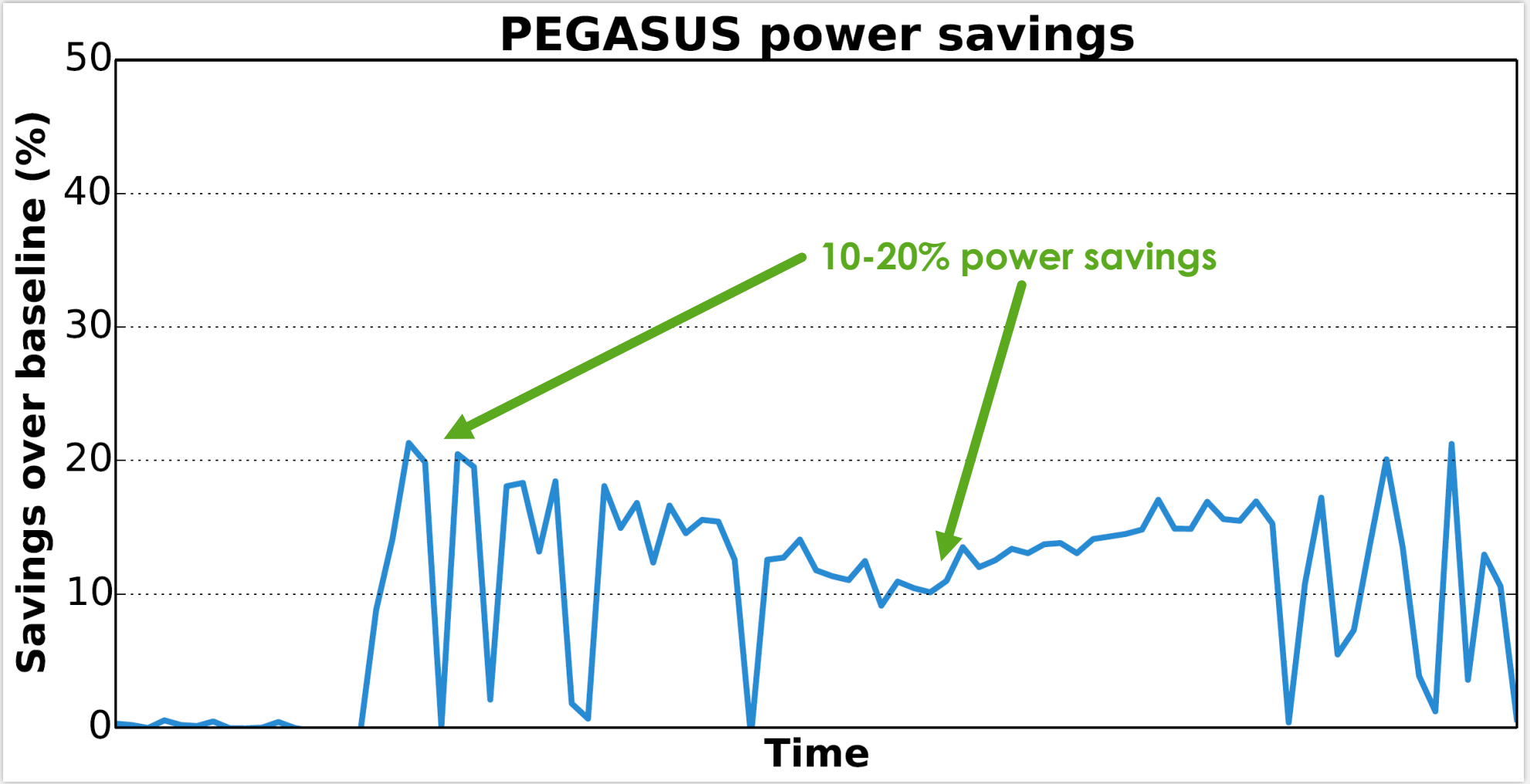
Small cluster results: power comparison



Production cluster results: power over time



Production cluster results: power comparison

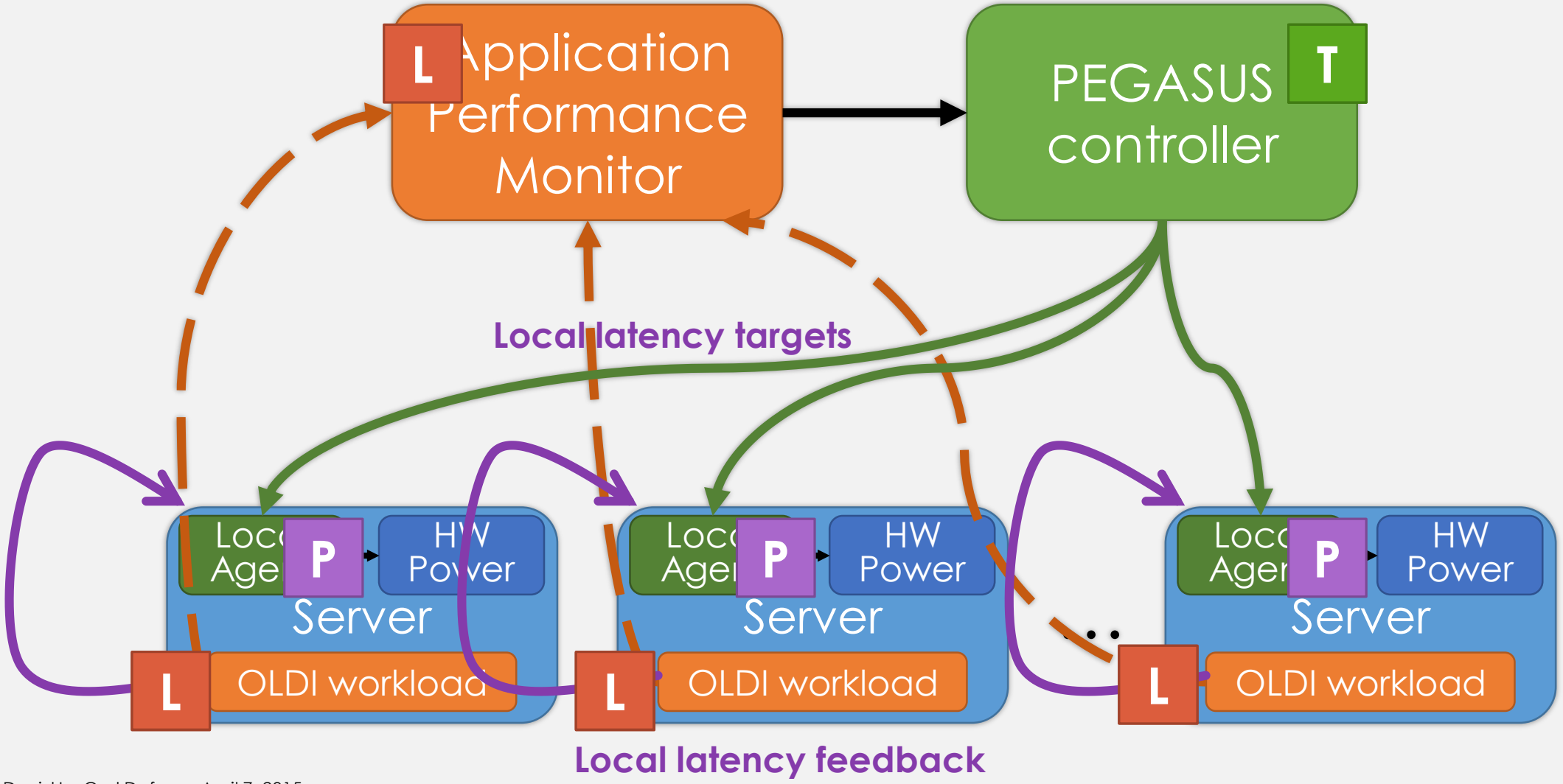


Improving PEGASUS scalability

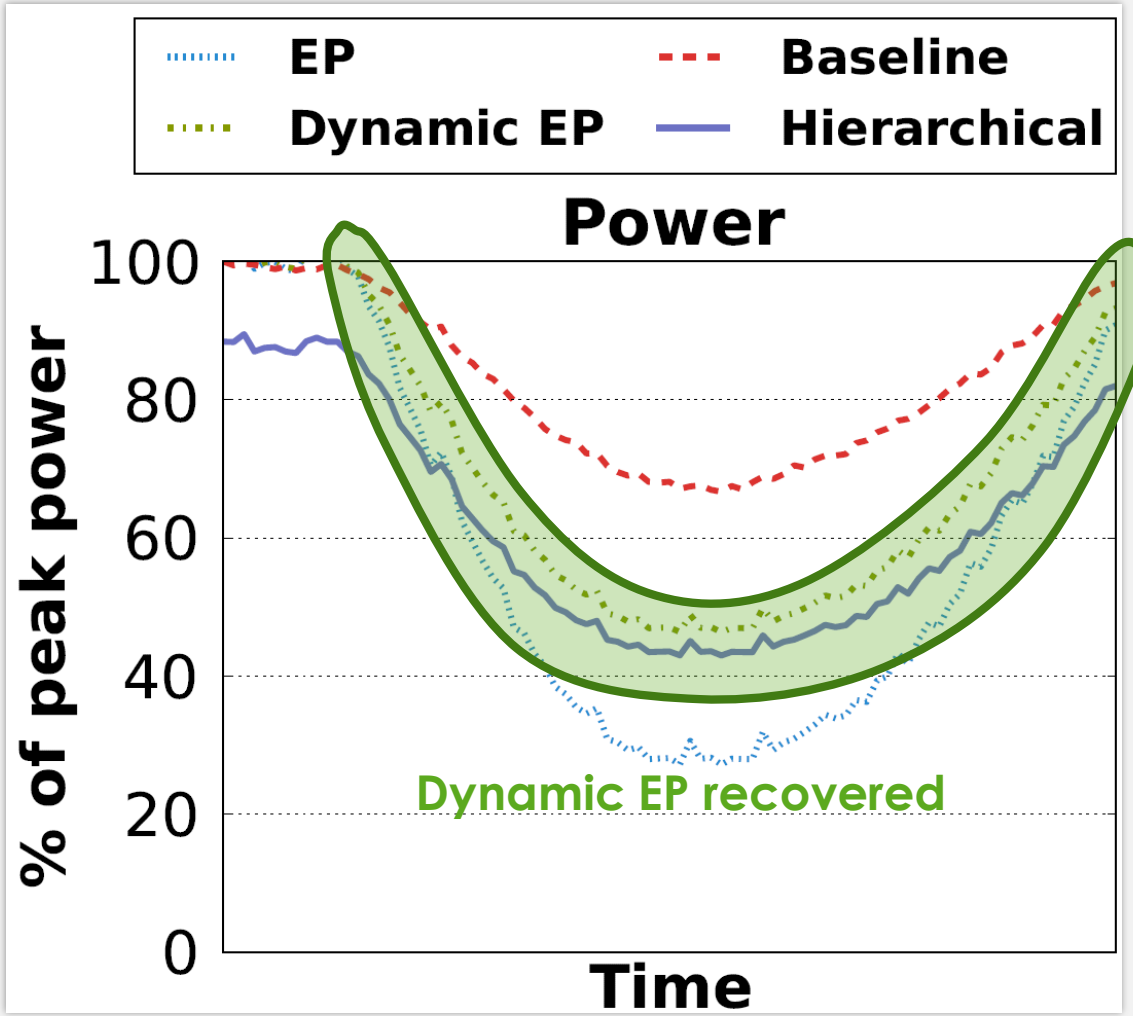
- Production cluster sees **"tail at scale"** for server utilization
  - At peak load, 0.2% nodes at 100% load while 50% nodes at <85% load
  - Caused by popular queries hitting a few shards
  - **Issue:** Hot nodes set lower bound on power limits for everyone

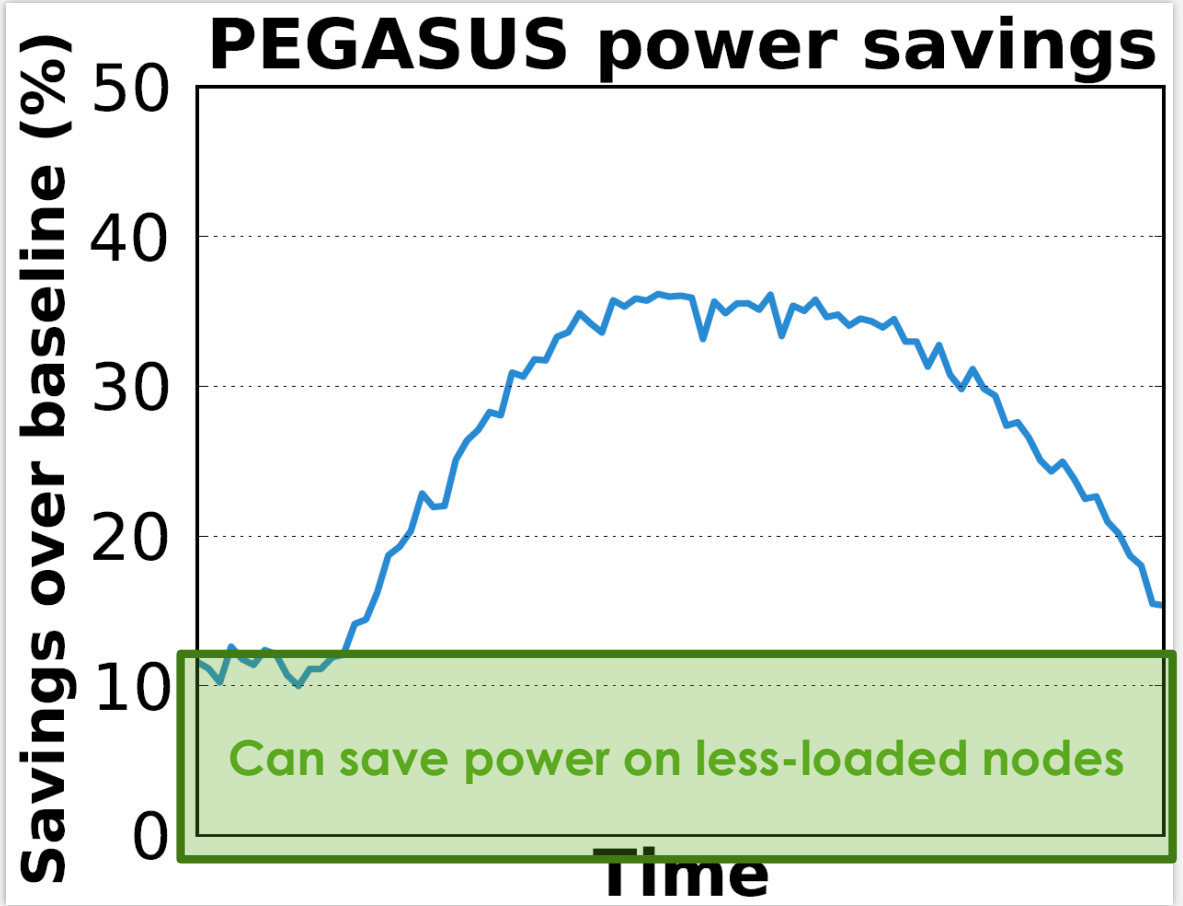
- **Idea**: hierarchical control
  - **Global:** sets latency targets instead of power limits
  - **Local:** decides amount of power needed to meet target latency

Hierarchical PEGASUS design



Estimated hierarchical PEGASUS results





PEGASUS summary

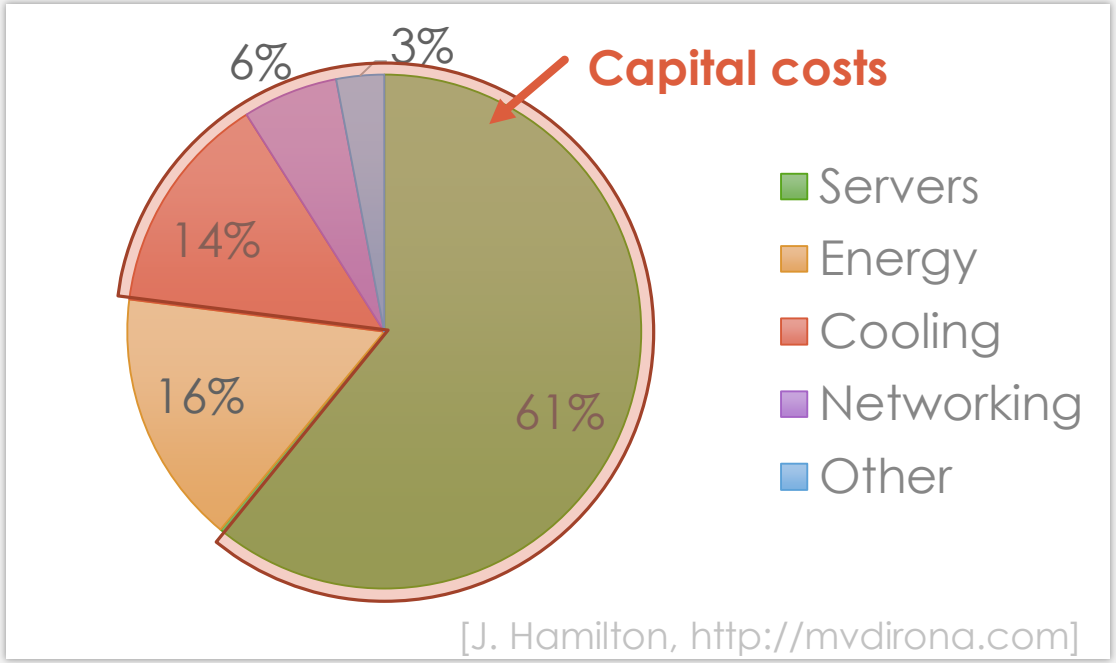
- Halfway there to fully energy proportional systems
- **Iso-latency:** Use SLO metrics and fine-grained power control
  - Save up to 30% power
  - Meet/exceed energy proportionality targets
- PEGASUS achieves **iso-latency** benefits
  - Up to 20% savings on production cluster
  - Be aware of tail at scale effects

Heracles

Reconciling low latency with high utilization

The need for resource efficiency

- Energy efficiency is good but resource efficiency is better
  - Vast majority of costs are non-energy costs

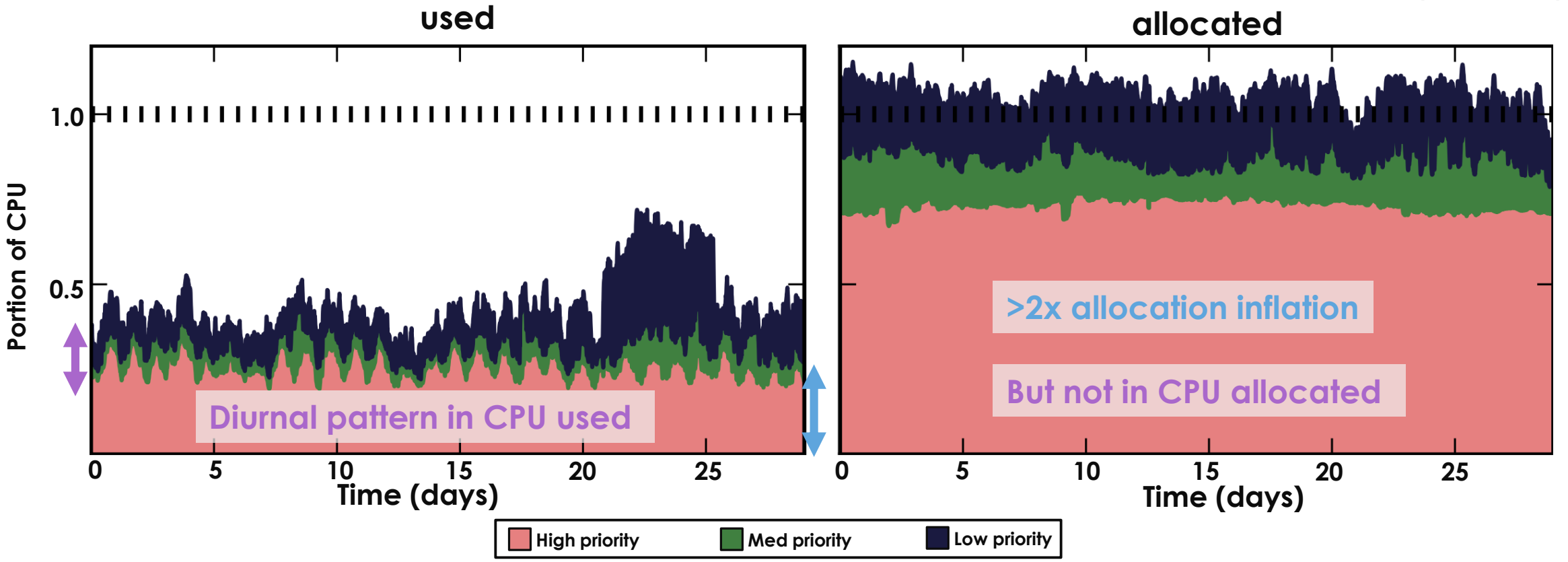


  - Increases capability of existing hardware

Underutilization even with cluster management

Study of utilization on 12,000 node Google cluster

[Reiss SOCC '12]

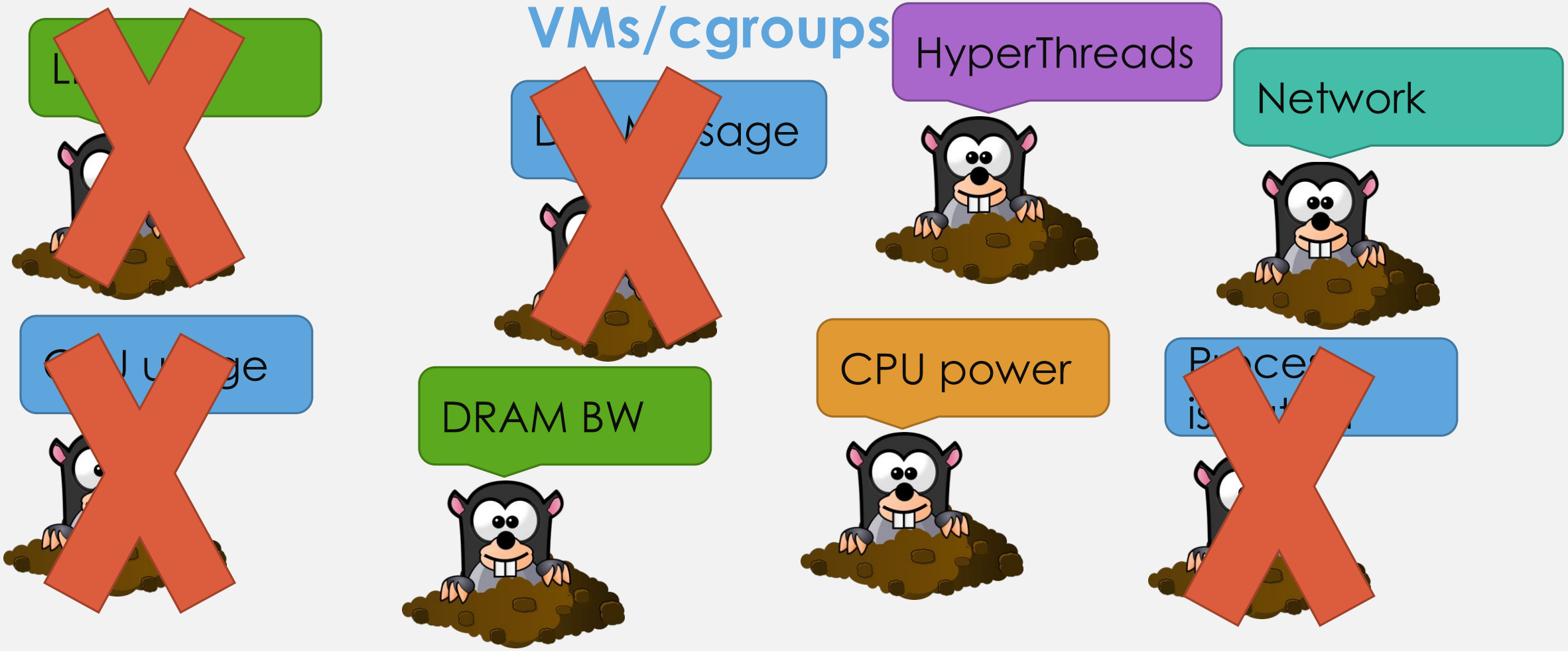


Underutilization is a structural problem

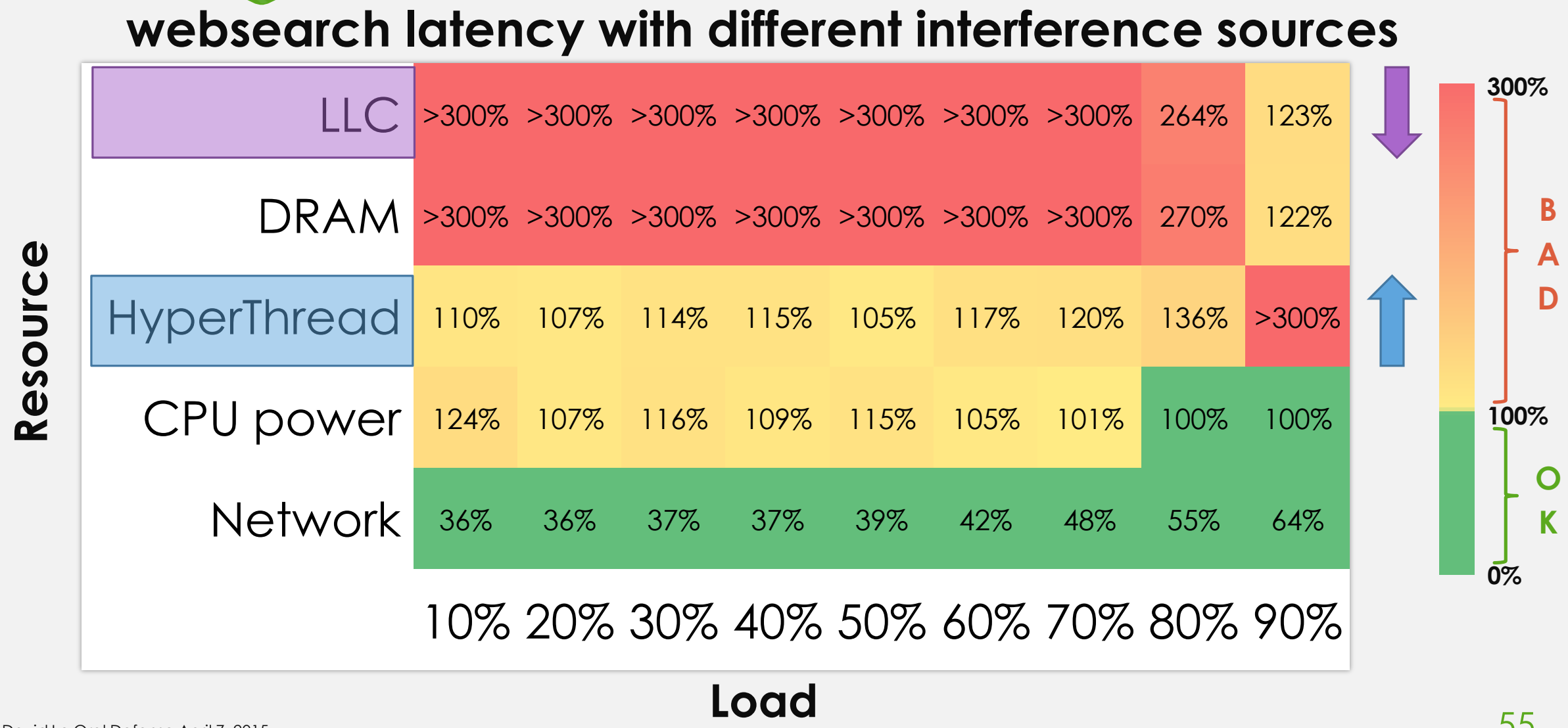
- Effectively an infinite supply of low priority batch jobs
  - Analytics, machine learning, etc.
- High priority jobs are allocated 2x the resources actually used

- Reasons for overprovisioning:
  - Diurnal user traffic
  - Planning for high traffic events
  - Avoid interference from other workloads

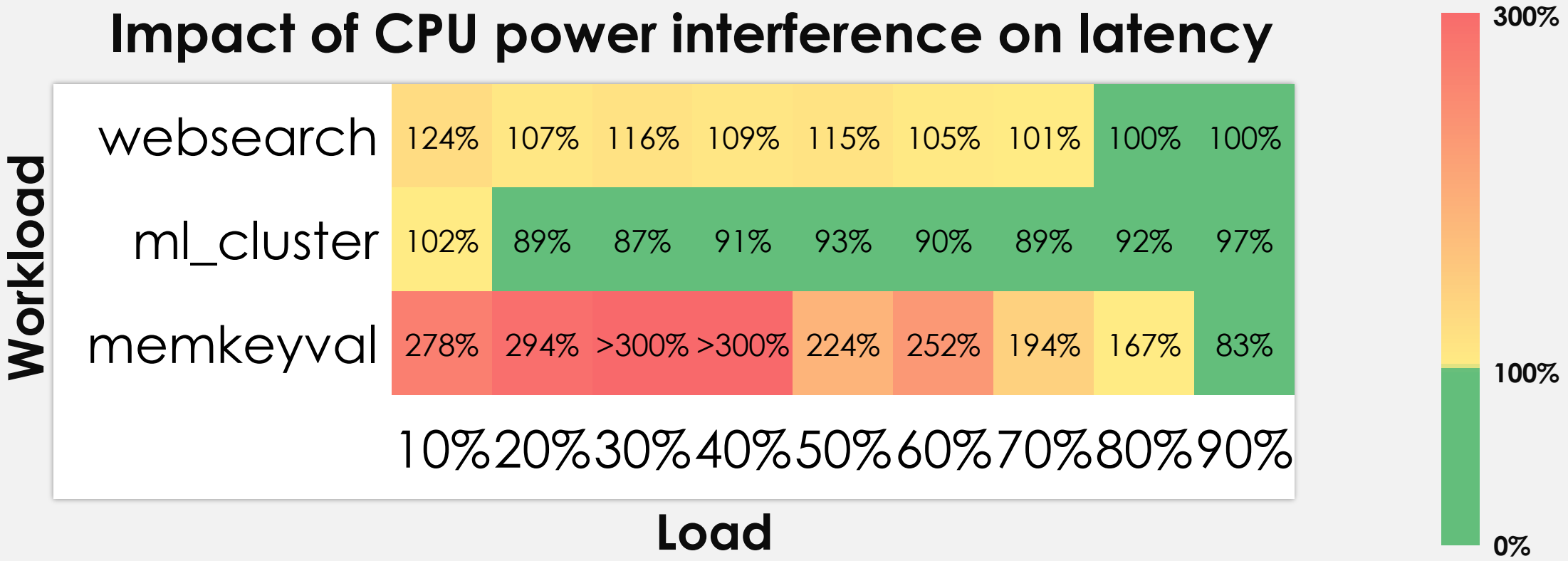
Lots of sources of interference



Interference is different based on resource



And is different based on workload

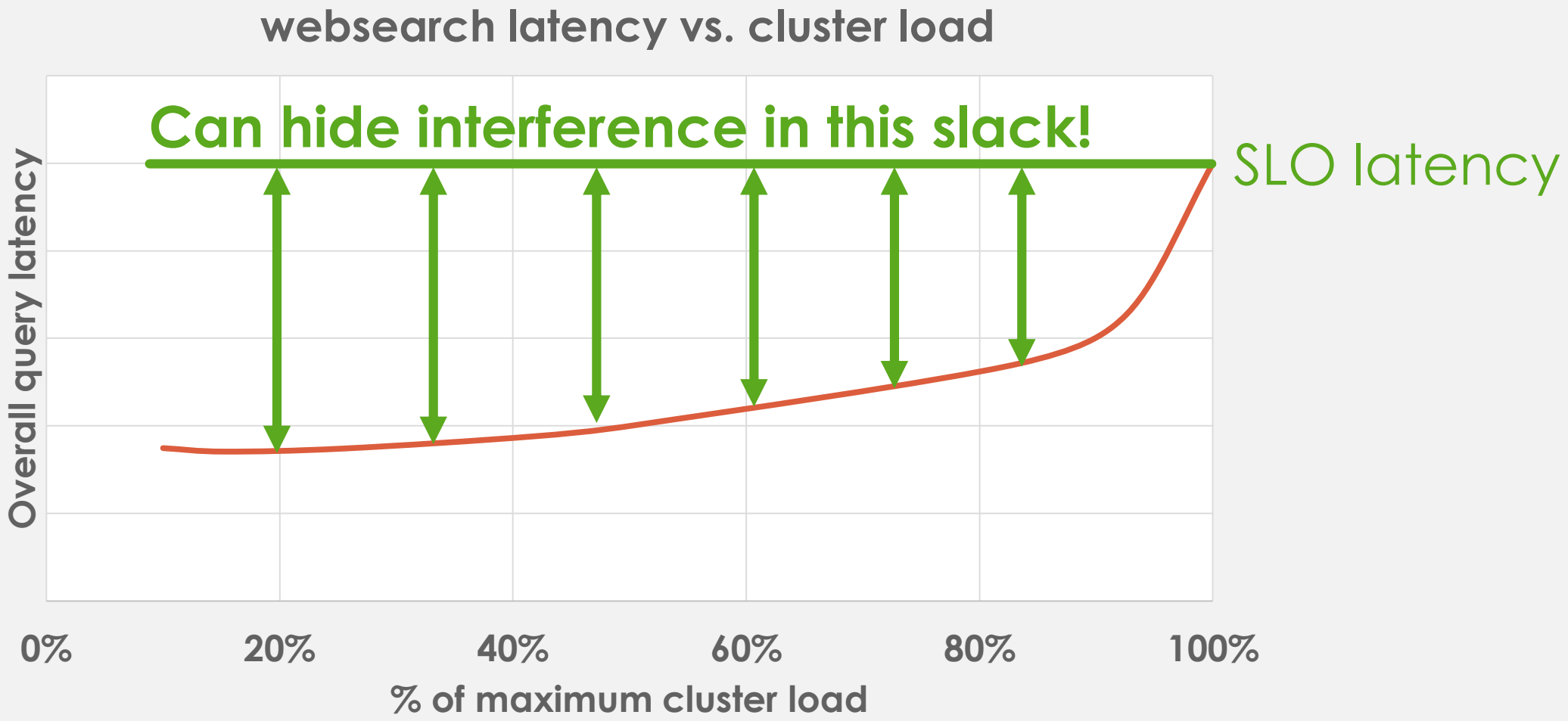


Need dynamic control scheme!

Heracles in a nutshell

- Oversubscription by co-locating a **Best Effort (BE)** job with a **Latency Critical (LC)** job, such as OLDI workloads
- Uses hardware and software knobs to mitigate interference
- Uses latency information from the application to adjust knobs
- Runs locally on each node, complements cluster manager

Use iso-latency to tolerate some interference



Knobs to mitigate interference

- **CPU (HT/L1/L2)**: isolate different workloads to different cores
- **LLC**: hardware cache partitioning
- **DRAM BW**: software monitoring and scheduling
  - Intuition: each core can only issue so many requests/sec
- **Network**: rate limiting in Linux kernel with HTBs
- **CPU power**: per-core DVFS

But how should the knobs be set?

- This looks like an optimization problem
  - **Objective**: maximize utilization
  - **Constraints**: preserve SLO of latency critical application

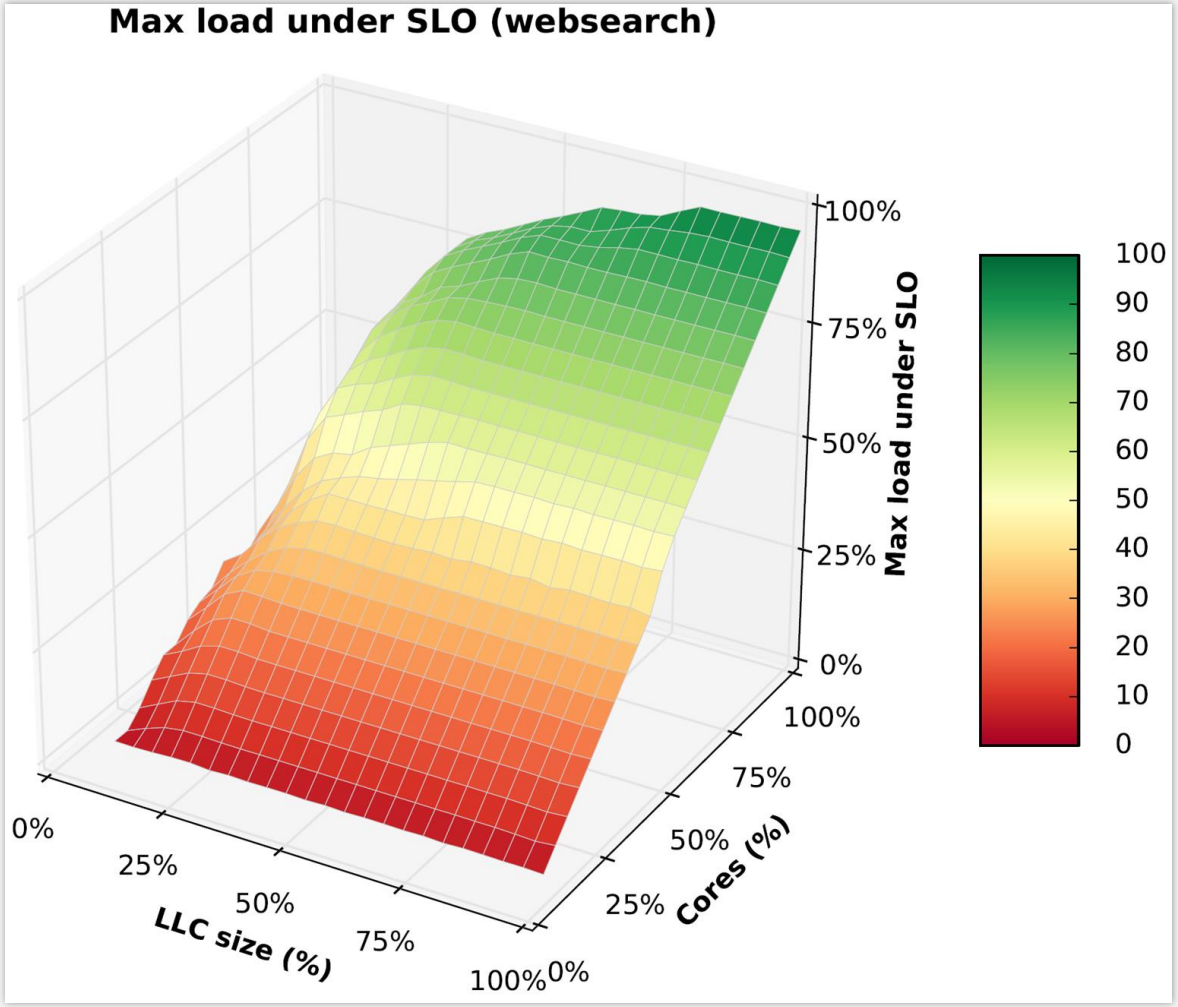
- **Challenge: 5-dimensional formulation!**

Decoupling interference sources

- **Observation**: latency violations occur when a shared resource is extremely loaded
  - High demand for resource causes significant contention
  - LC workload is unable to obtain its required allocation

- **Insight:** assume independent interference under 2 conditions
  - LC workload is not starved for any resource
  - Each resource has enough slack to absorb bursts

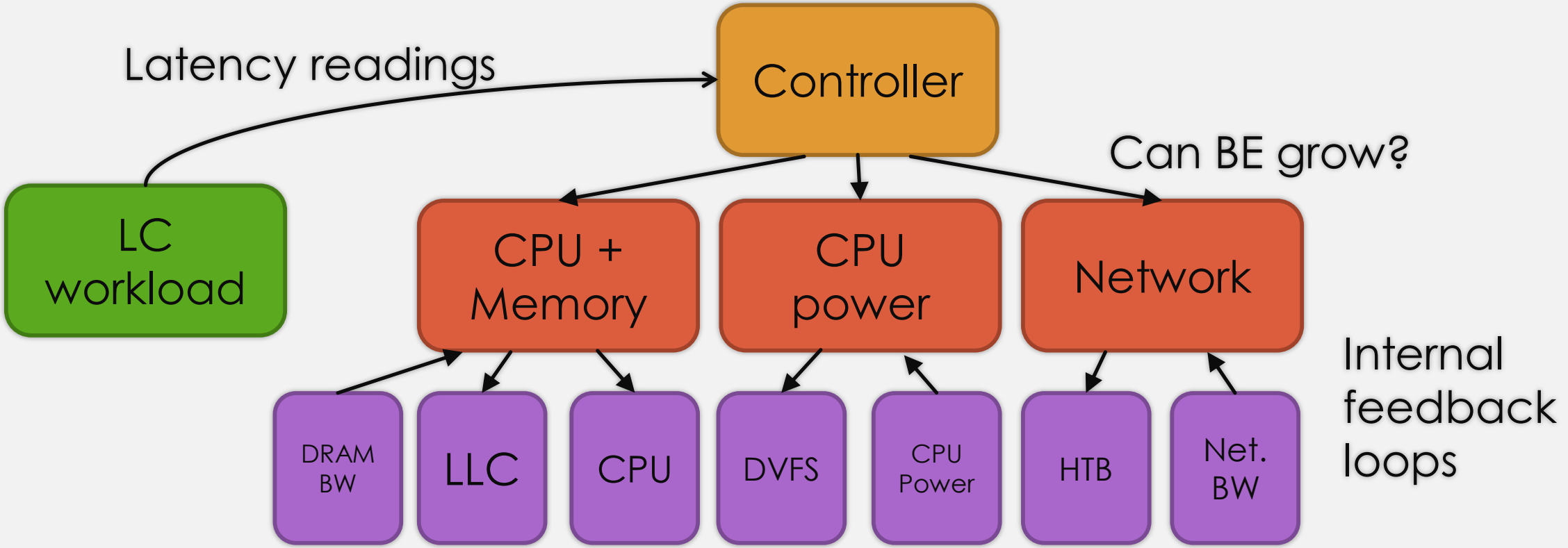
Gradient descent find optimal allocation



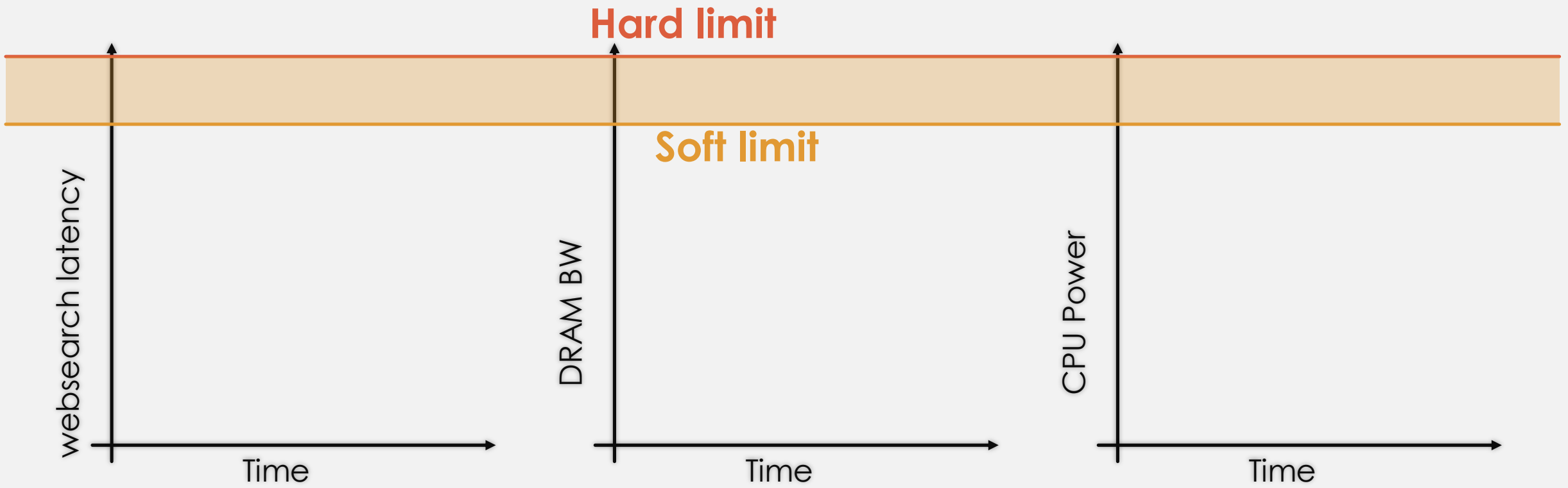
- Performance as a function of resources is convex for benchmarked workloads
- Use of gradient descent is guaranteed to produce optimality

Heracles: shared resource manager

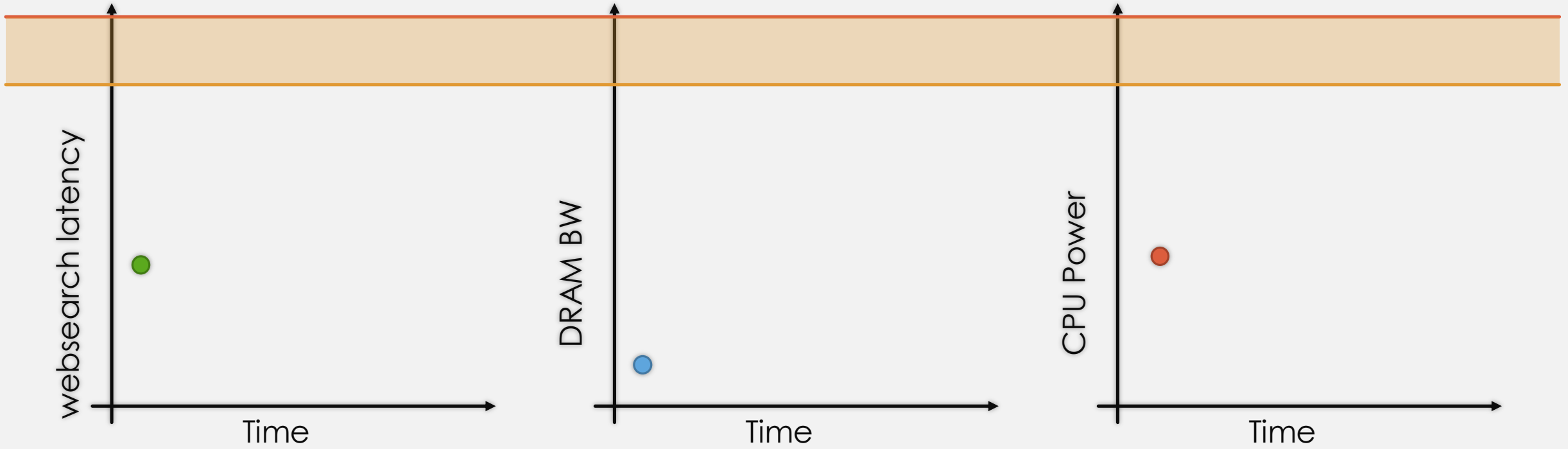
- **Goal**: meet SLO, keep BE from saturating shared resource
- Runs on each machine, ~1k lines of code



Example Heracles run

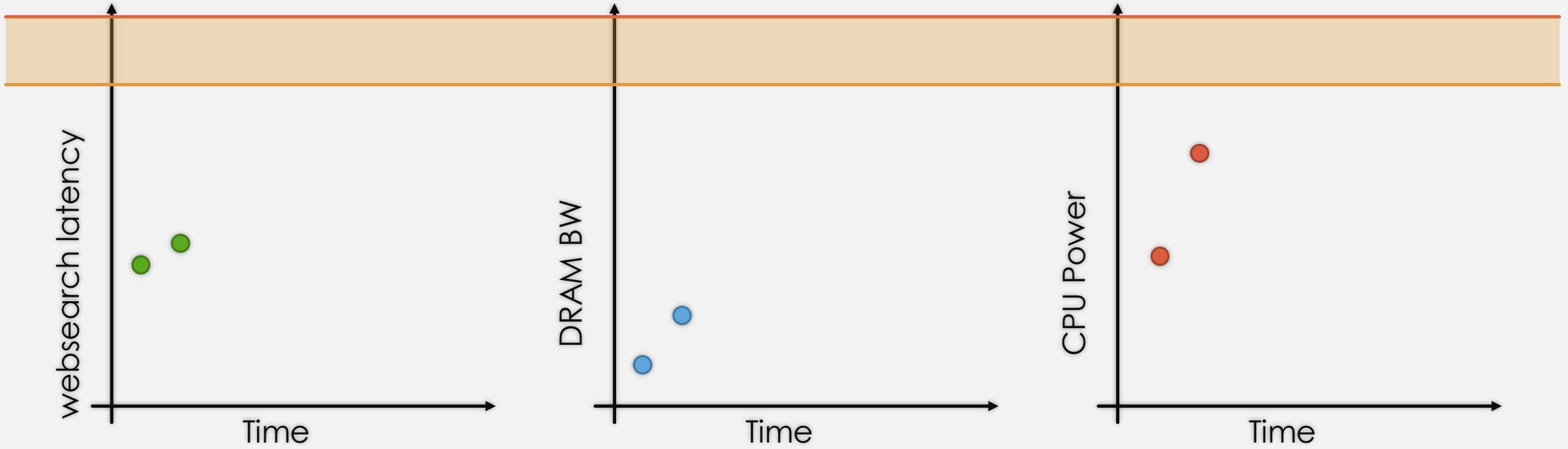


Example Heracles run



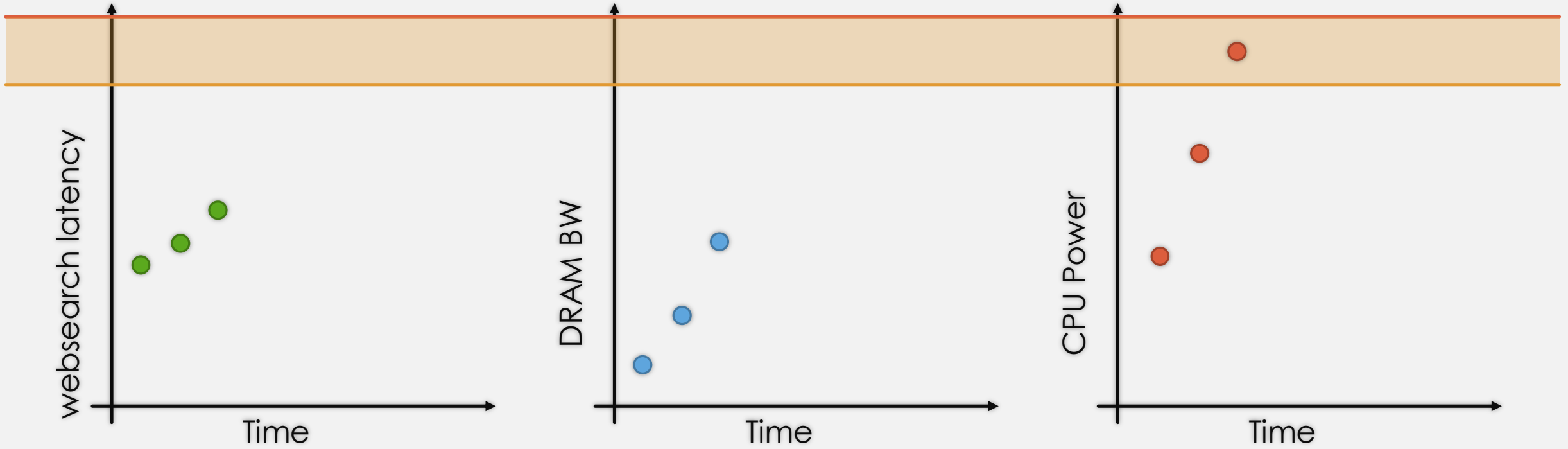
Core Allocation

Example Heracles run



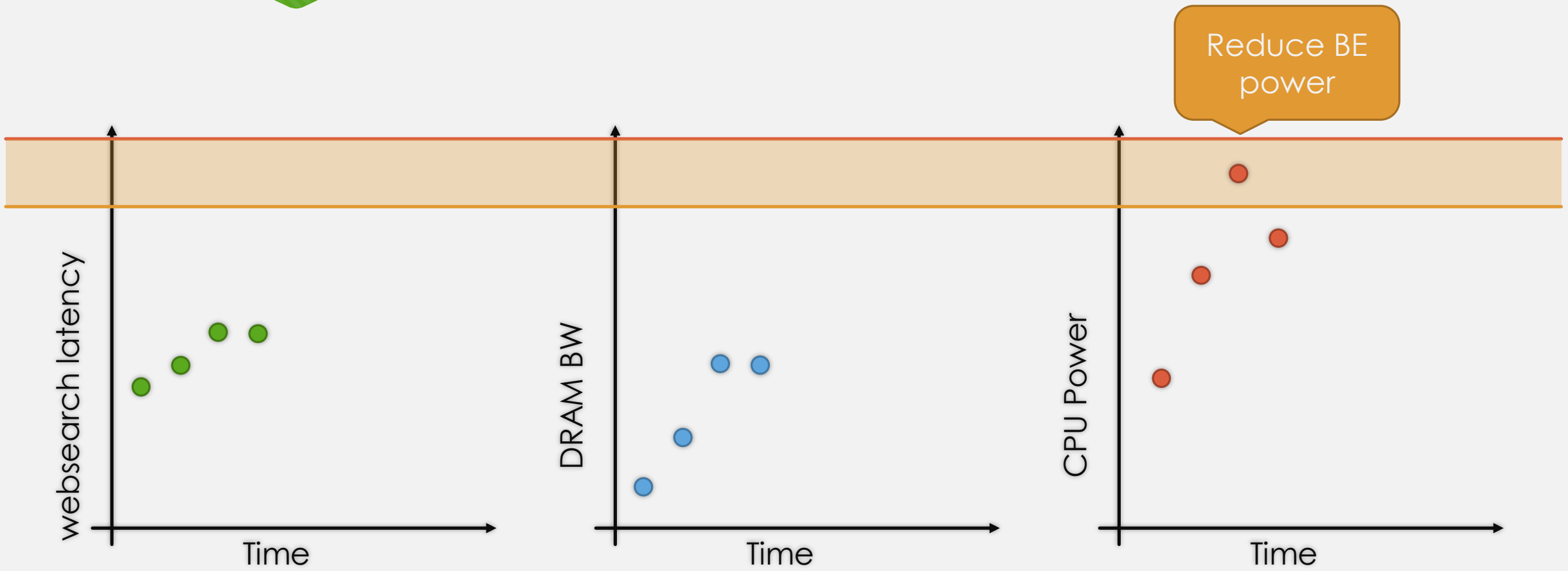
Core Allocation

Example Heracles run



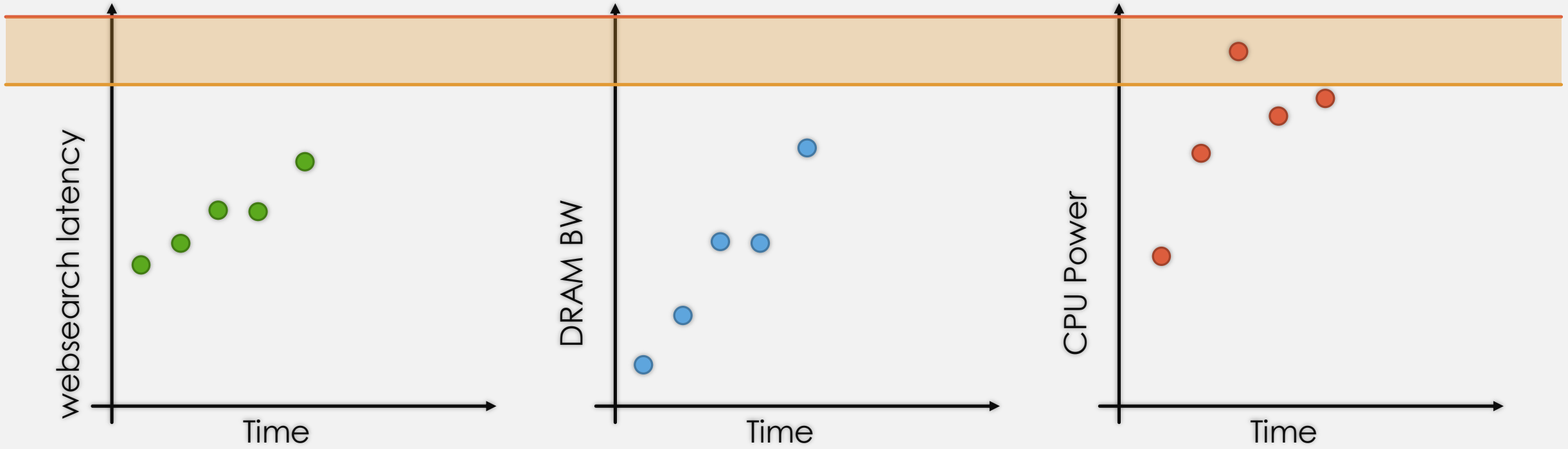
Core Allocation

Example Heracles run



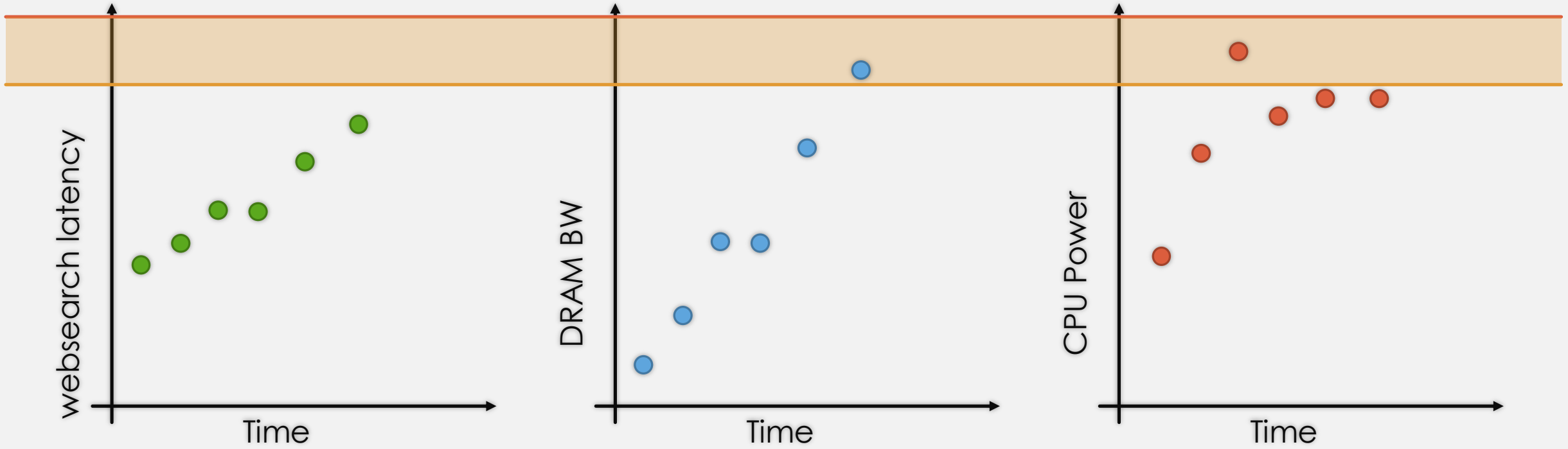
Core Allocation

Example Heracles run

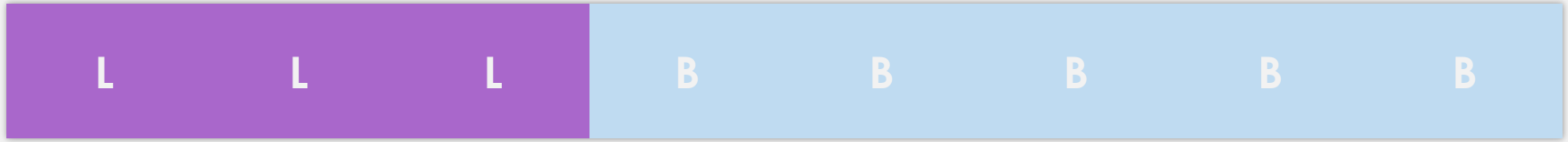


Core Allocation

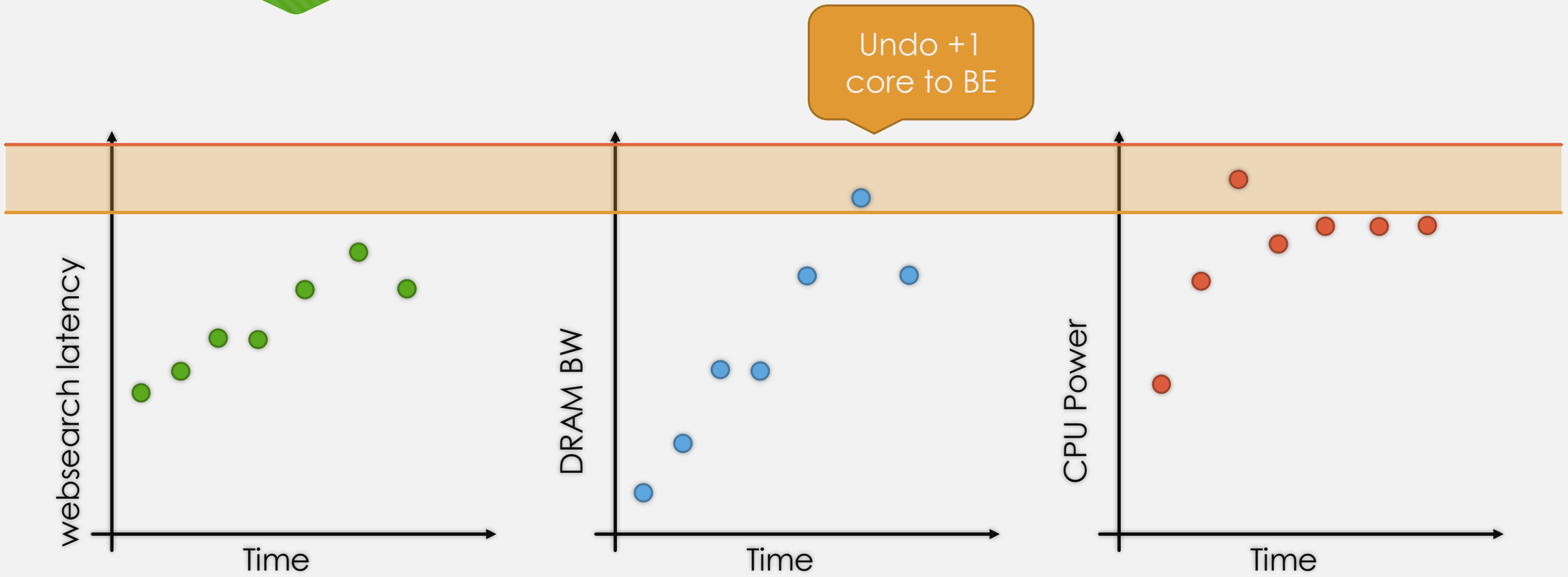
Example Heracles run



Core Allocation



Example Heracles run



Core Allocation

Latency Critical OLDI workloads

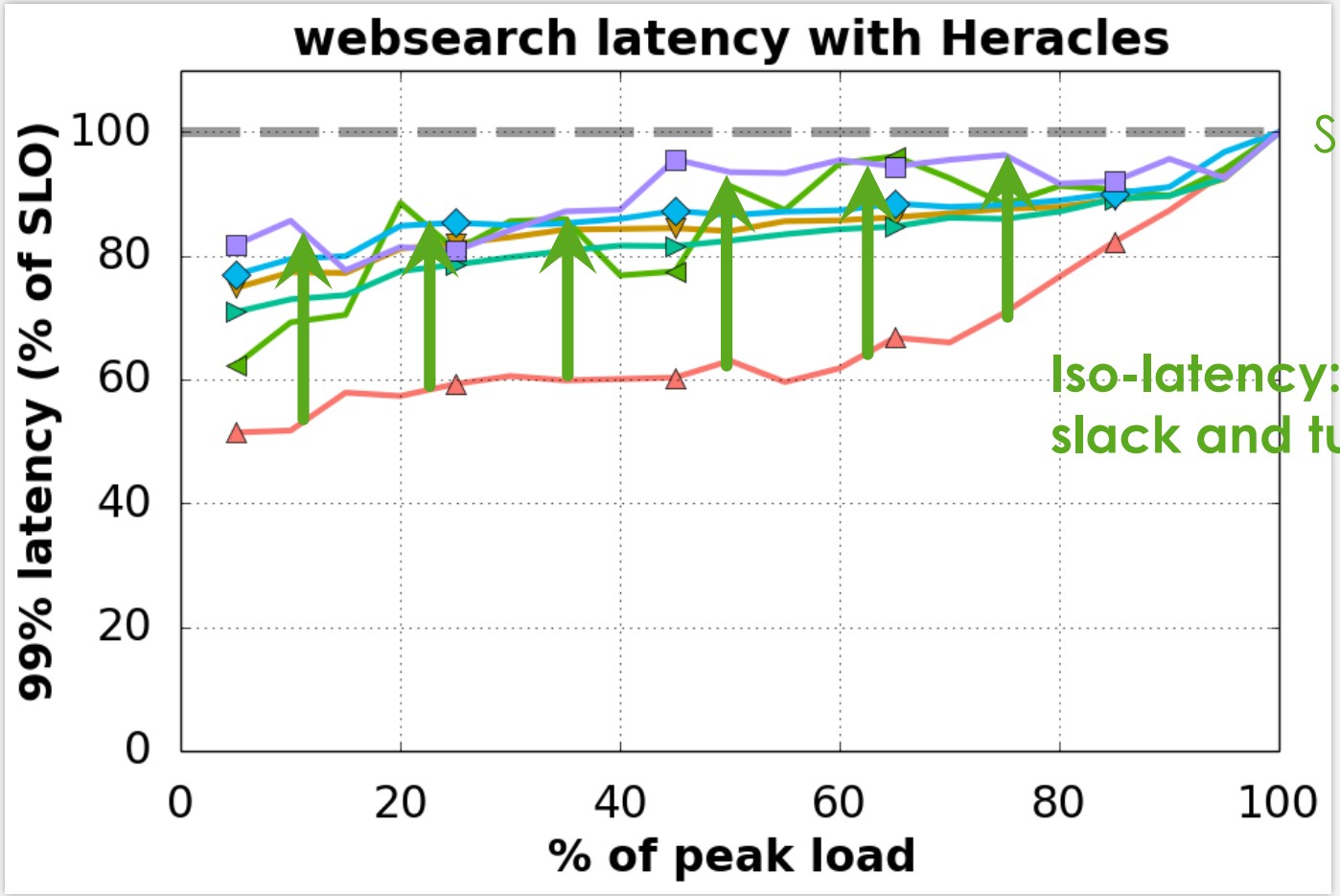
- **websearch**
  - Leaf node, document retrieval/scoring
  - 99%-ile latency SLO of tens of milliseconds
- **ml_cluster**
  - Machine learning for text clustering
  - 95%-ile latency SLO of tens of milliseconds
- **memkeyval**
  - In-memory key-value store
  - 99%-ile latency SLO of hundreds of microseconds

Best Effort jobs

- **stream-LLC**: LLC antagonist
- **stream-DRAM**: DRAM BW antagonist
- **cpu_pwr**: CPU power antagonist
- **brain**: deep learning (LLC, DRAM, CPU, CPU power)
- **streetview**: image stitching (DRAM BW)

Run Heracles on real hardware, measure latency and utilization

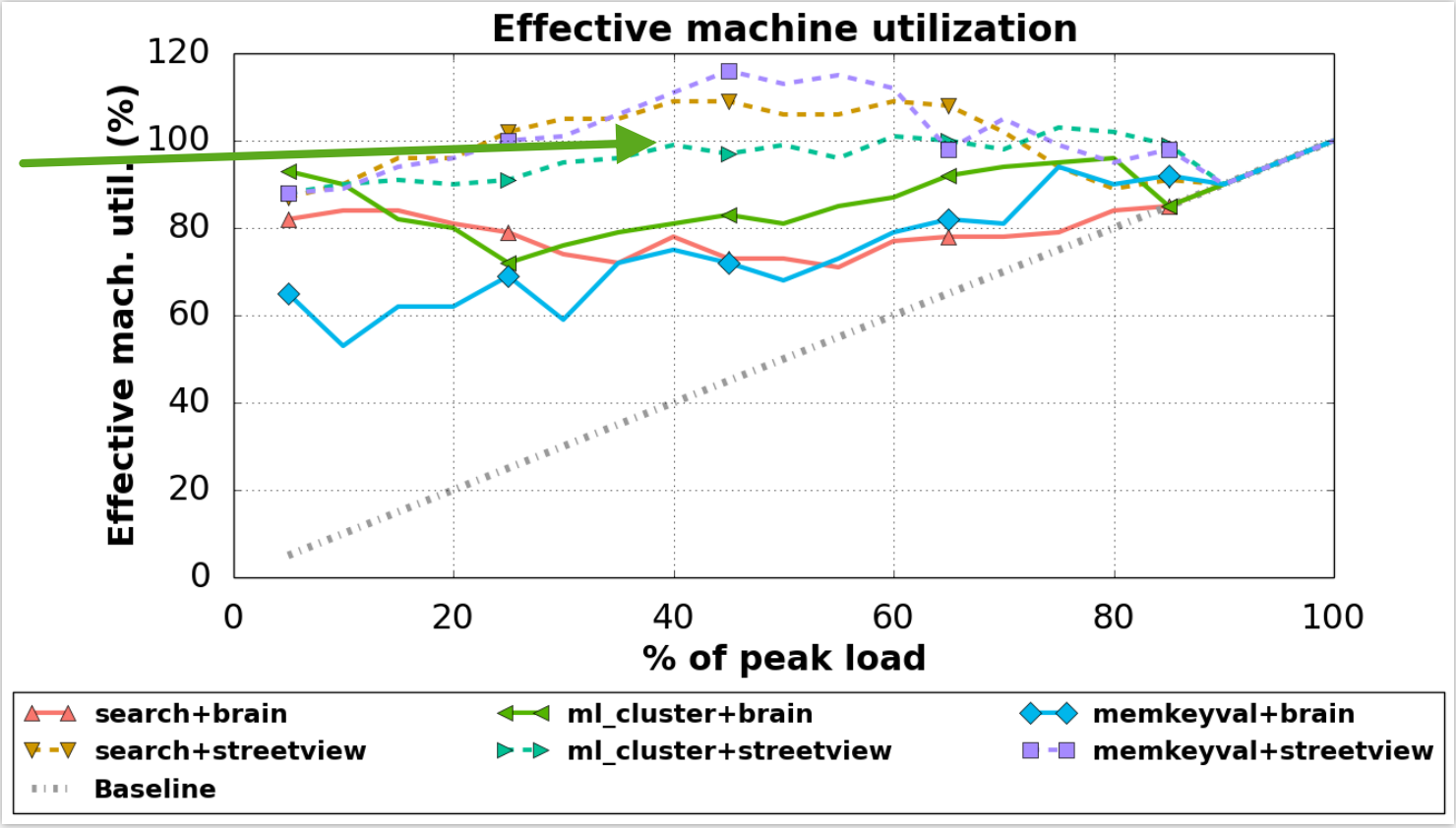
Latency validation: do no harm



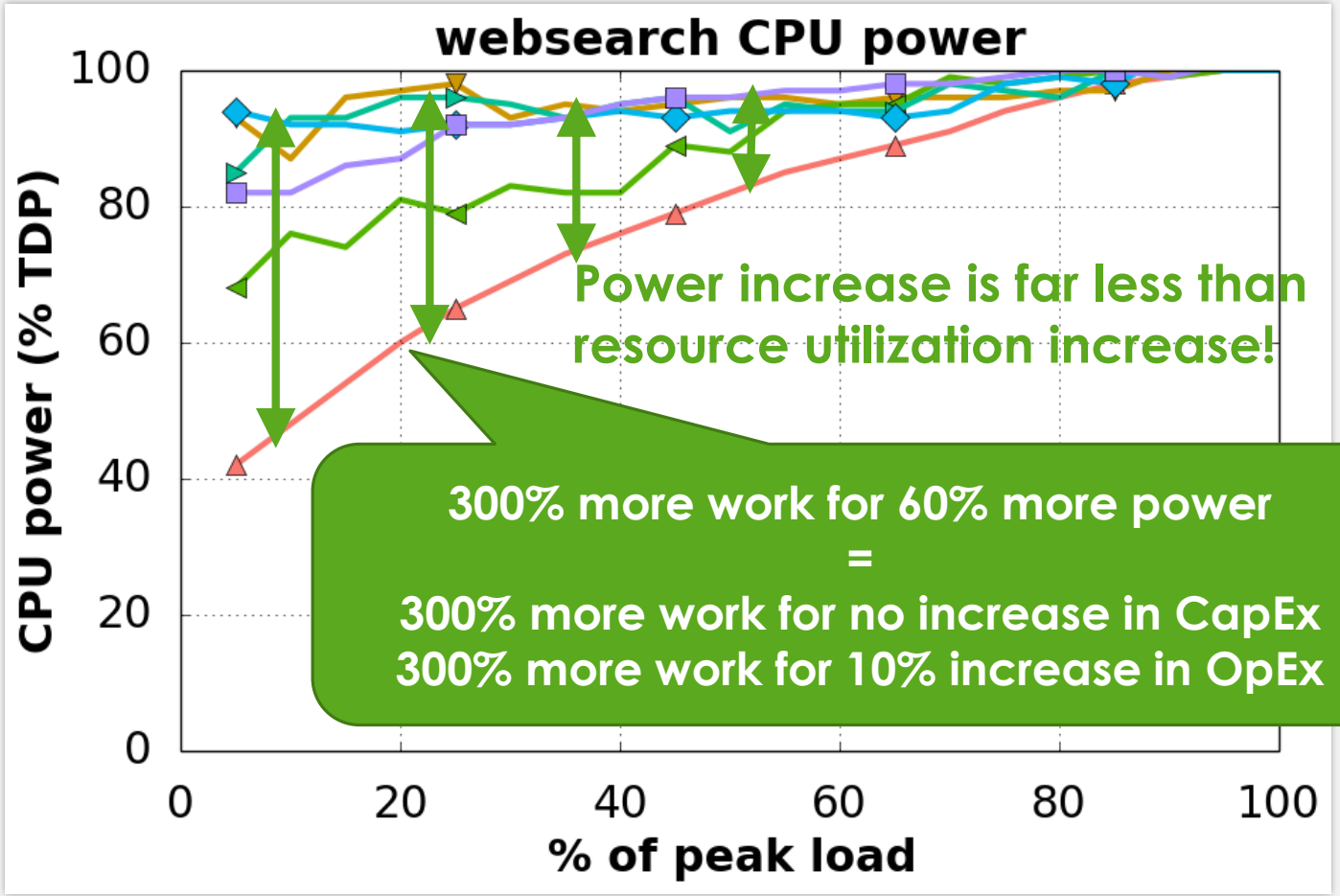
Putting it together: resource efficiency

Effective Machine Utilization = (LC load) + (% BE throughput)

Better than 100% is due to better binpacking

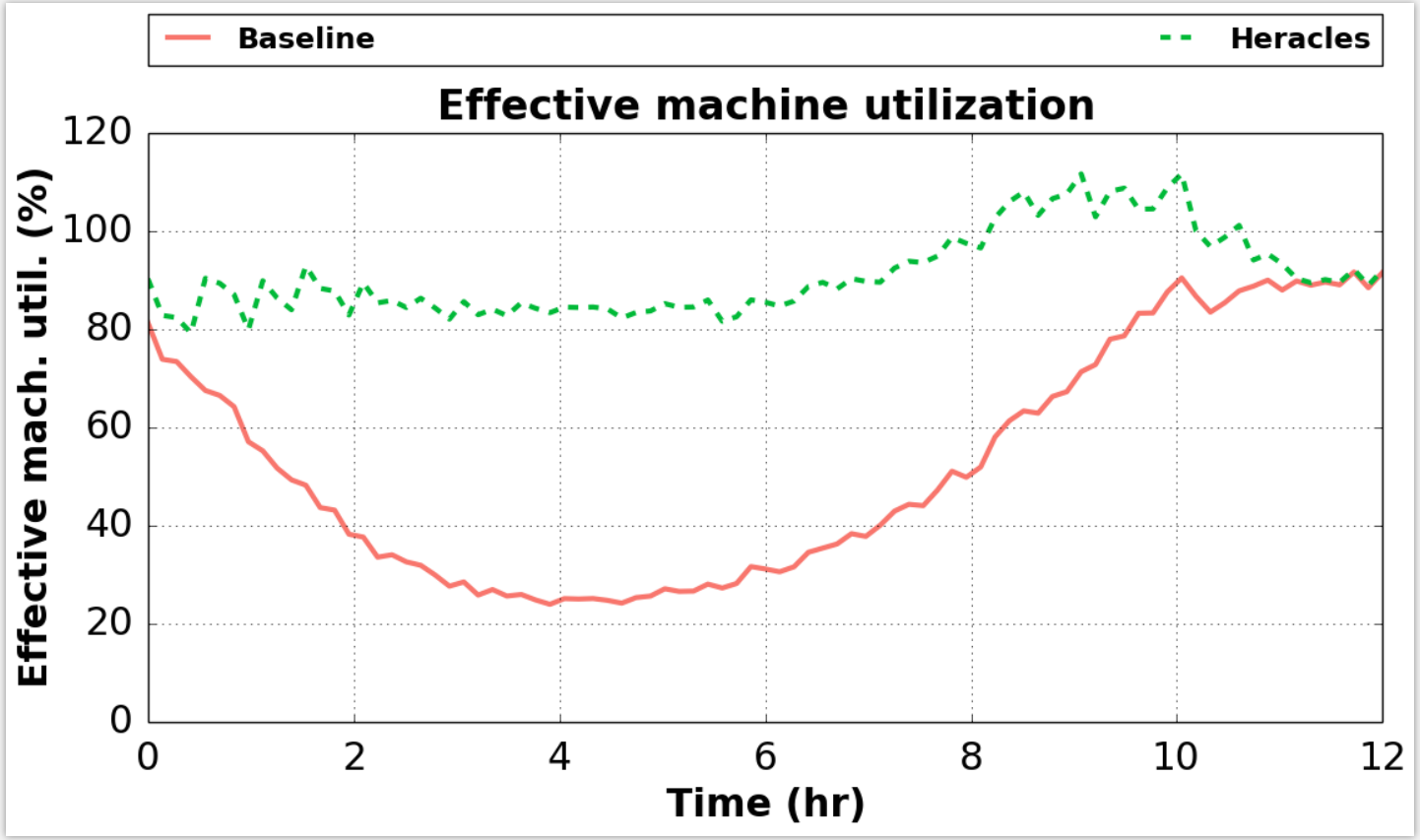


Bonus: energy efficiency too!



Minicluster results

- Use load trace for off-peak hours on websearch minicluster



Related work

- Interference-aware cluster management
  - Paragon [ASPLOS'13], Quasar [ASPLOS'14], BubbleUp [MICRO'11], BubbleFlux [MICRO'12], CPI2 [Eurosys'13]
  - **Complementary, as Heracles enables more co-location opportunities**

- Hardware/software partitioning mechanisms
  - Cache: Vantage [ISCA'11], Ubik [ASPLOS'14], UCP [MICRO'06]
  - Iyer et al [SIGMETRICS'07], memory hierarchy QoS
  - **Lots of individual mechanisms available, Heracles puts them together**

Heracles summary

- Increasing utilization is key to improving datacenter efficiency
- Lots of knobs to control lots of sources of interference
  - Need coordinated policy to find optimal settings
- Heracles significantly increases utilization
  - Achieves average of 90% utilization for Google workloads
  - Potential increase of >300% in cost efficiency

Contributions recap

- Autoturbo: improving energy efficiency
- OLDIsim: scale-out benchmark
- PEGASUS: improving energy efficiency
  - Achieves dynamic energy proportionality for latency-critical jobs
  - Saves up to 30% power during low loads
- Heracles: improving resource efficiency
  - Enables high utilization even in the face of tight SLO constraints
  - Achieves 90% utilization without causing latency violations

Acknowledgements

- **Thesis advisor:** Prof. Kozyrakis
- **Examiners:** Prof. Rosenblum, Prof. Olukotun, Prof. Darve, Prof. Horowitz
- **Google:** Liqun, Rama, Partha, Luiz, …
- **Research group:** Daniel, Christina, Jacob, Adam, …
- **Parents:** Wei Jen and Mei Lien
- **Caroline Suen**
- **Friends**

Thanks for listening!

Fin.