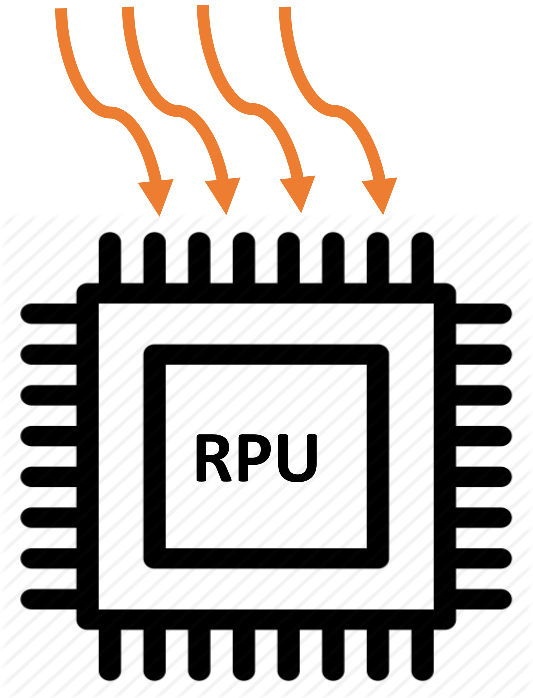
MICRO 2022



SIMR: Single Instruction Multiple Request Processing for Energy-Efficient Data Center Microservices

Mahmoud Khairy*, Ahmad Alawneh, Aaron Barnes, and Timothy G. Rogers

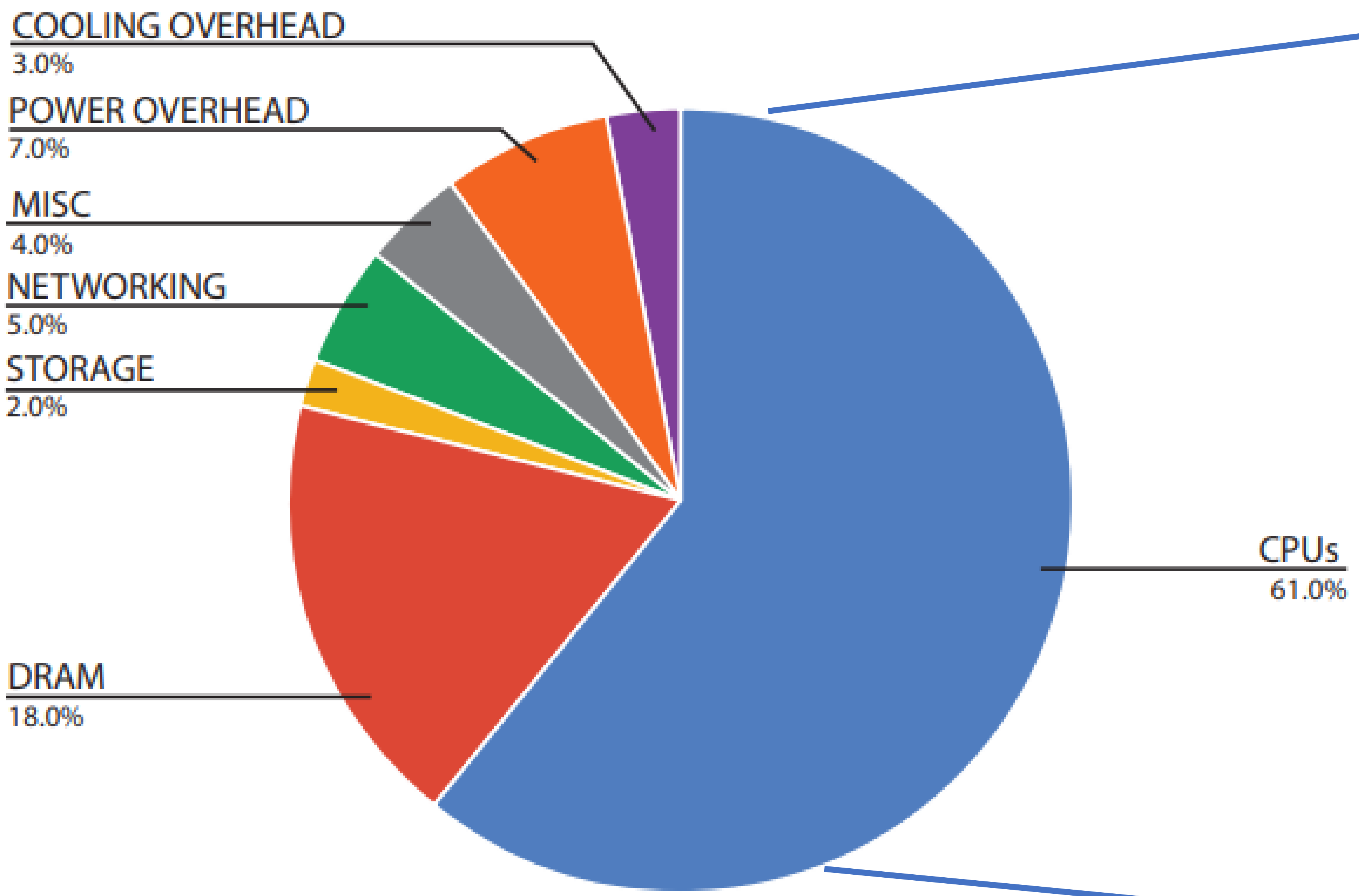
Purdue University

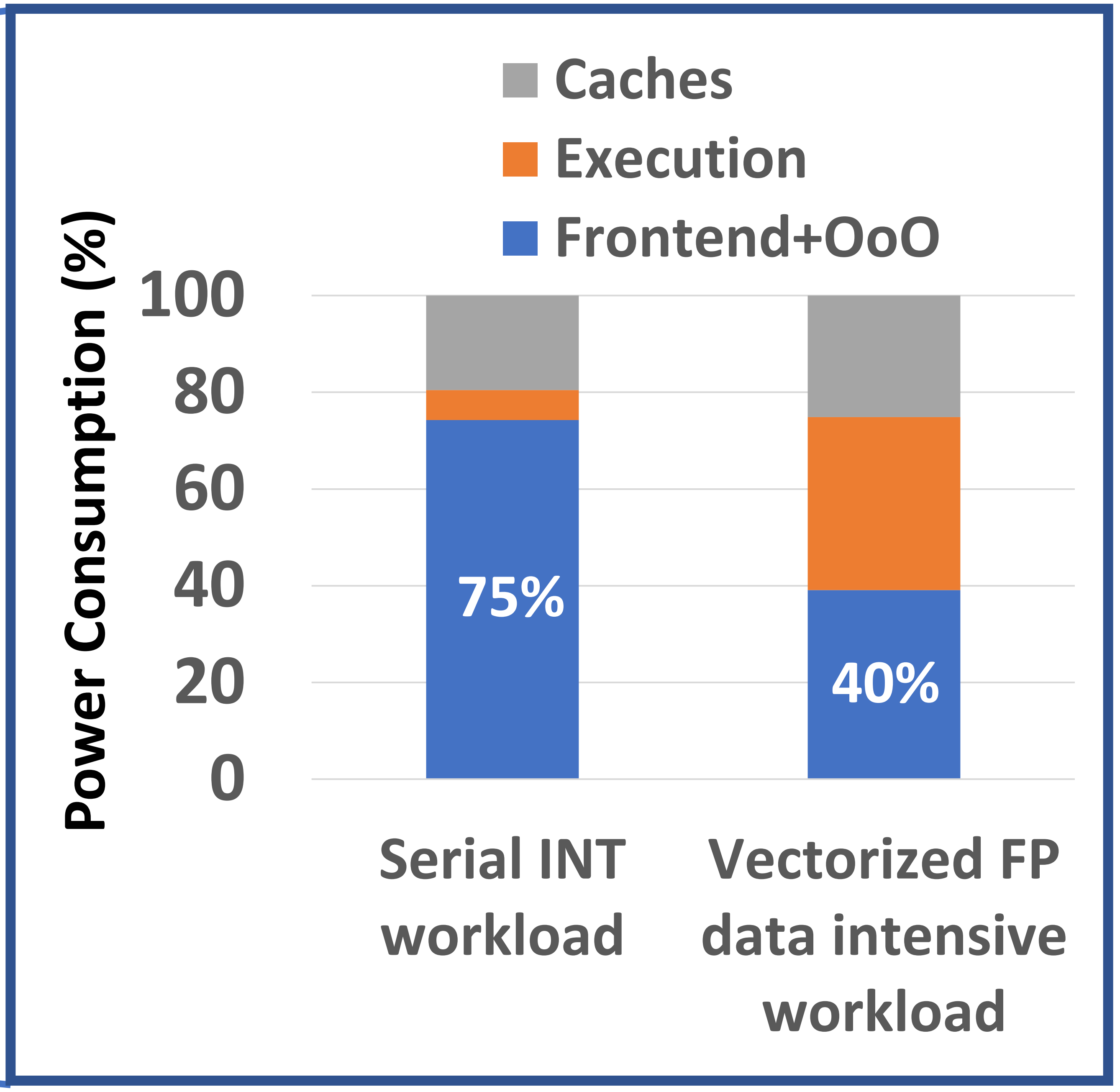
Home page: https://mkhairy.github.io/
Contact: abdallm@purdue.edu

*Currently at AMD Research

10/3/2022

Datacenter Power Breakdown





Datacenter Power Breakdown (from Google)

CPU Power Breakdown

25-45% of datacenter power is consumed in CPU’s instruction supply (frontend & OoO)

Barroso, Luiz André, and Urs Hölzle. "The datacenter as a computer: An introduction to the design of warehouse-scale machines." *Synthesis lectures on computer architecture*. 2018
Haj-Yihia, Jawad, et al. "Fine-grain power breakdown of modern out-of-order cores and its implications on skylake-based systems." *ACM TACO* 2016
Notes: in the TACO paper, Execution includes ALU+Reg+OoO. In the fig above, we exclude the OoO and add it to the frontend. Caches power include dynamic L1/L2/L3 cache power.. The numbers are collected with McPAT

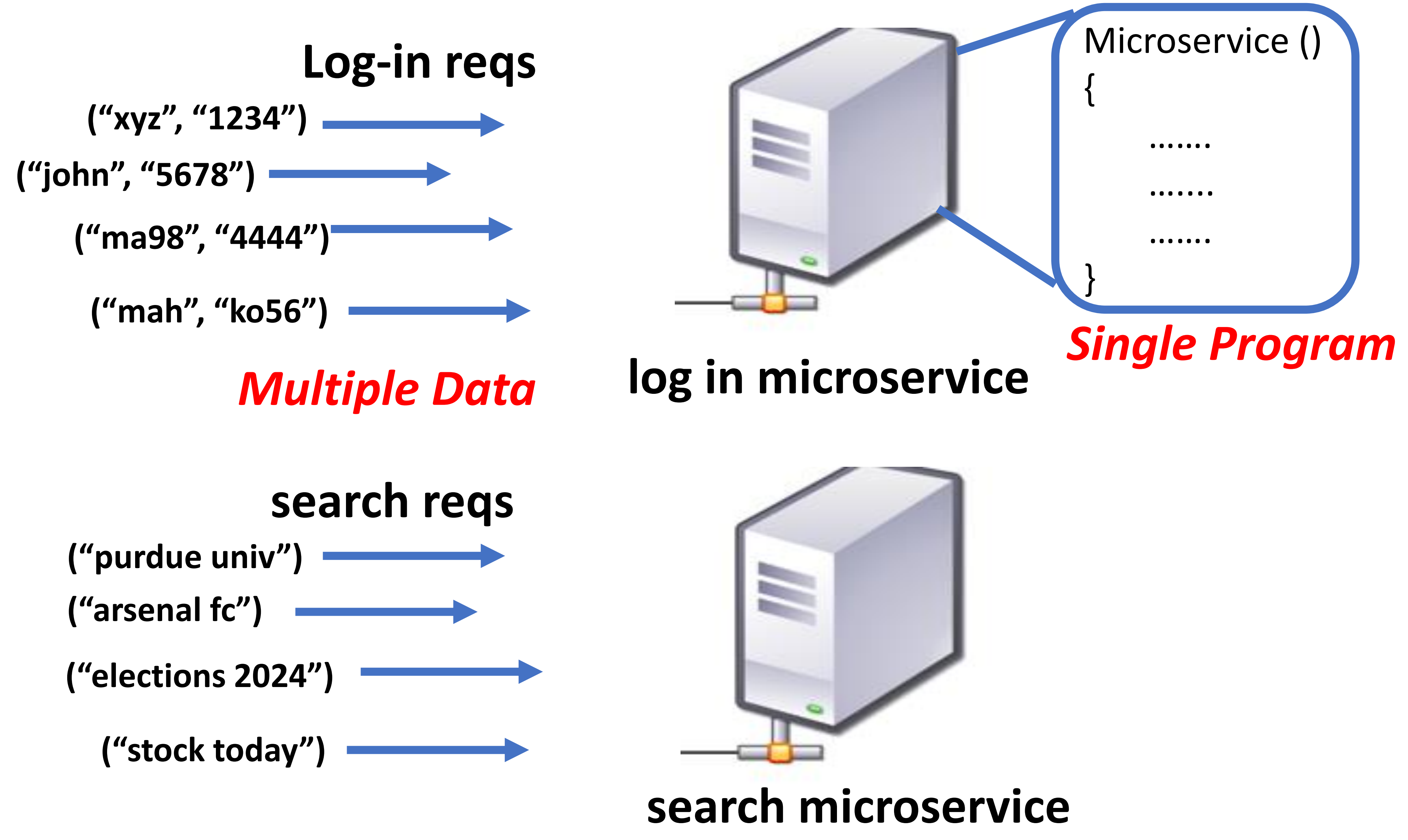
1 Application, Millions of Users

Private Datacenter

Public Datacenter

"Similar" Request-Level Parallelism

1000s of independent requests are all running the same code



Key Observation #1: Single Program Multiple Data (SPMD) are abundant in the datacenters

Server Workloads on GPU's

- **Key Idea:** Exploit SPMD by batching requests and run them on GPU's Single Instruction Multiple Thread (SIMT) or CPU's SIMD
- **Advantage**: Significant energy efficiency (throughput/watts) vs multi-threaded CPU
- **Drawbacks**:
  - (1) Hindering programmability (C++/PHP vs CUDA/OpenCL)
  - (2) Limited system calls support
  - (3) High service latency (10-6000x)
    - GPUs tradeoff single threaded optimizations (OoO, speculative execution, etc.) in favor of excessive multithreading
    - In SIMD, relying on branch predicates & fine grain context

Rhythm: Harnessing Data Parallel Hardware for Server Workloads

Sandeep R Agrawal, Duke University, sandeep@cs.duke.edu
Valentin Pistol, Duke University, pistol@cs.duke.edu
Jun Pang, Duke University, pangjun@cs.duke.edu
John Tran, NVIDIA, johntran@nvidia.com
David Tarjan *, NVIDIA
Alvin R Lebeck, Duke University, alvy@cs.duke.edu

Rhythm, ASPLOS 2014

MemcachedGPU: Scaling-up Scale-out Key-value Stores

Tayler H. Hetherington, The University of British Columbia, taylerh@ece.ubc.ca
Mike O'Connor, NVIDIA & UT-Austin, moconnor@nvidia.com
Tor M. Aamodt, The University of British Columbia, aamodt@ece.ubc.ca

MemcachedGPU, SoCC 2015

ispc: A SPMD Compiler for High-Performance CPU Programming

Matt Pharr, Intel Corporation, matt.pharr@intel.com
William R. Mark, Intel Corporation, william.r.mark@intel.com

ispc, InPar 2012

Recall: GPUs and SIMDs were designed to execute data parallel portion (i.e., loops) not the entire application

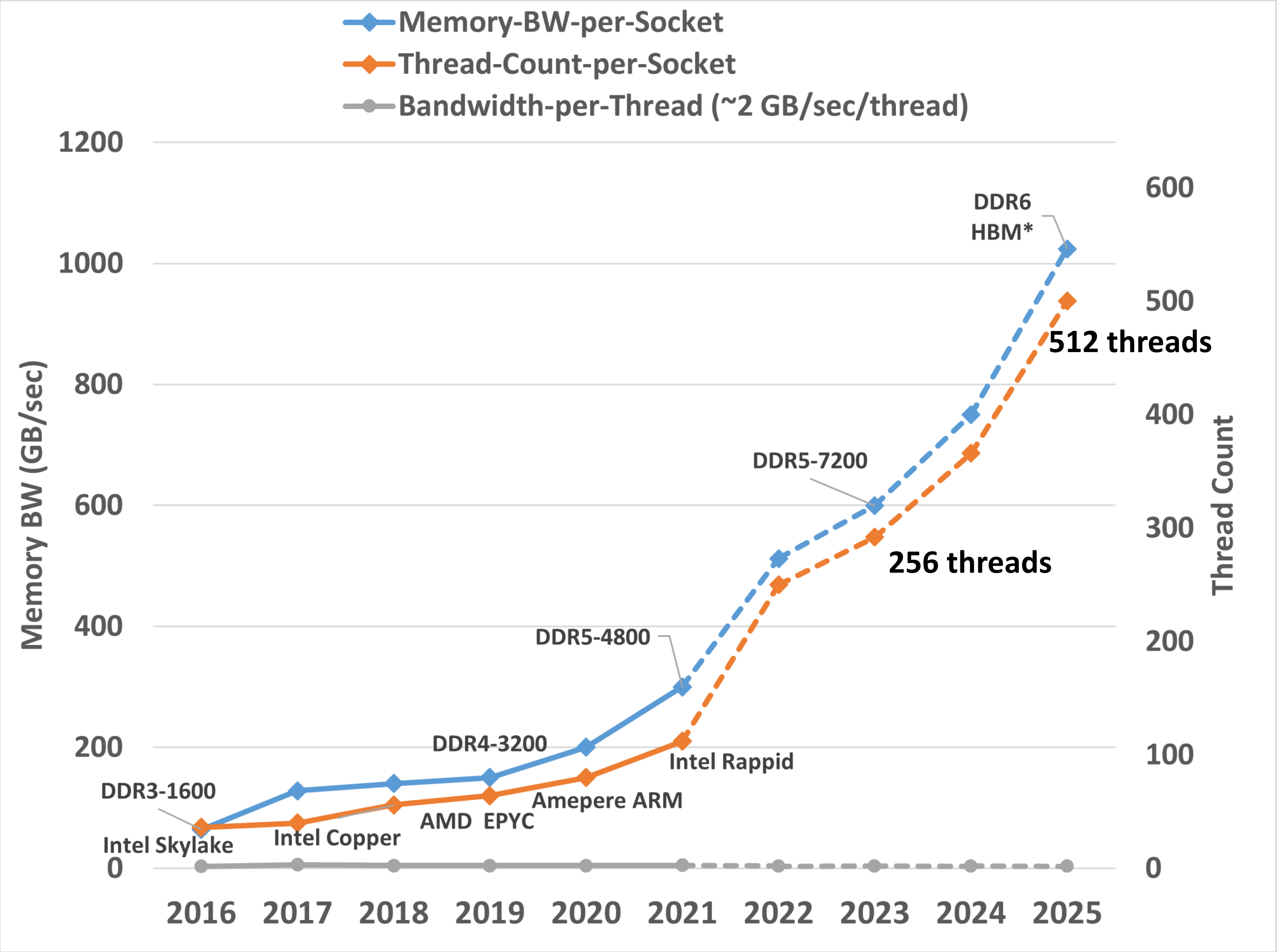
“Slower but energy-efficient wimpy cores only win for general data center workloads if their single-core speed is reasonably close to that of mid-range brawny cores”

Up to 2x slower latency can be tolerated by data center providers

Urs Hölzle
Google SVP

Barroso, Luiz André, and Urs Hölzle. "The datacenter as a computer: An introduction to the design of warehouse-scale machines." *Synthesis lectures on computer architecture*. 2018
Hölzle, Urs. "Brawny cores still beat wimpy cores, most of the time." IEEE MICRO 2010

Off-Chip BW Scaling



Key Observation #2: There is available headroom to increase on-chip throughput (thread count) in the foreseeable future.

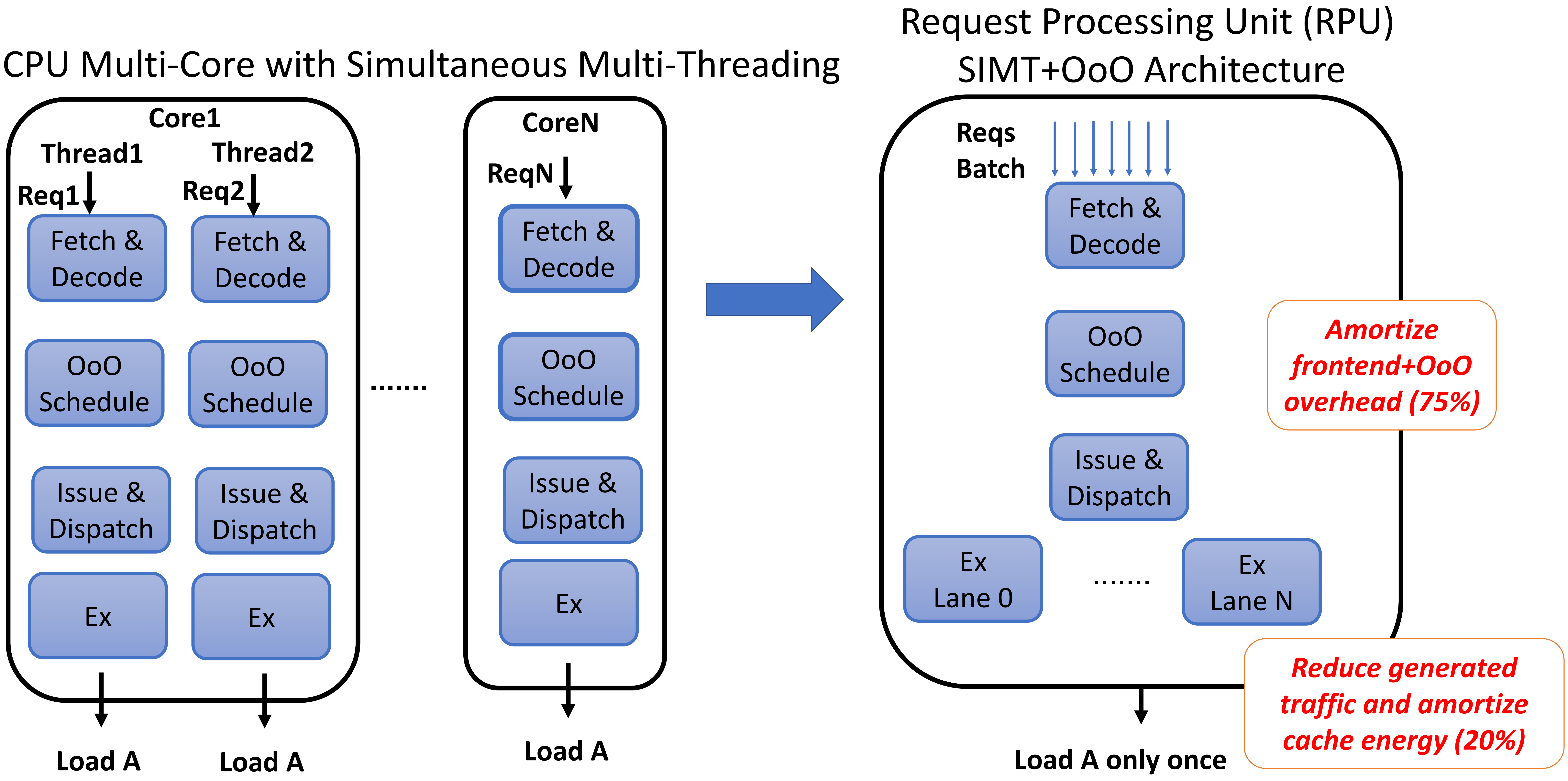
How to increase on-chip throughput of CPU?

- Direction#1 (industry standard): Add more Chiplets + Cores + SMT ✗

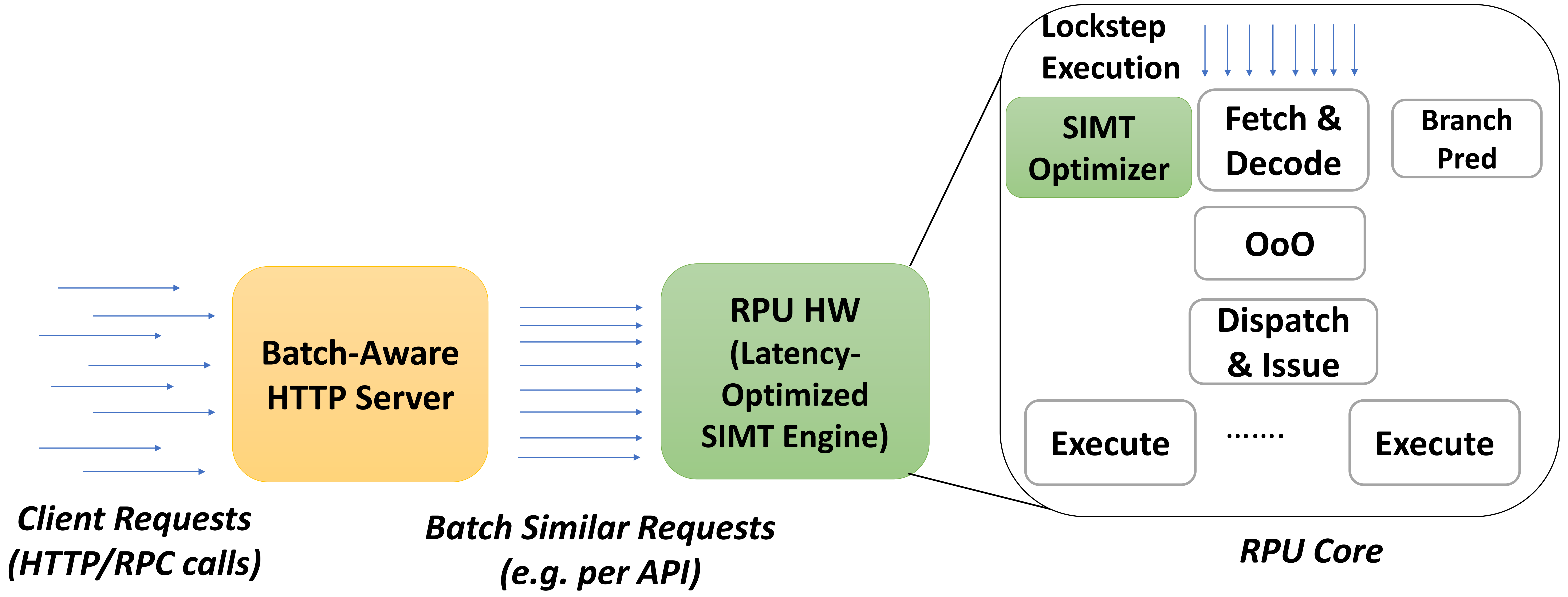
- Direction#2 (this work): Move to *SIMT* ✓
  - More energy efficient (throughput/watts)
  - Cost-effective (throughput/area)
  - Better scalability

“Let’s bring SIMT efficiency to the CPU world!”

SIMT Efficiency



SIMR System Overview



CPU vs GPU vs RPU

Metric	CPU	GPU	RPU
Core model	OoO	In-Order	OoO
Programming	General-Purpose	CUDA/OpenCL	General-Purpose
ISA	x86/ARM	HSAIL/PTX	x86/ARM
System Calls Support	Yes	No	Yes
Thread grain	Coarse grain	Fine grain	Coarse grain
Threads per core	Low (1-8)	Massive (2K)	Moderate (8-32)
Thread model	SMT	SIMT	SIMT
Consistency	Variant	Weak+NMCA*	Weak+NMCA*
Interconnect	Mesh/Ring	Crossbar	Crossbar

The RPU takes advantage of the latency optimizations and programmability of the CPU

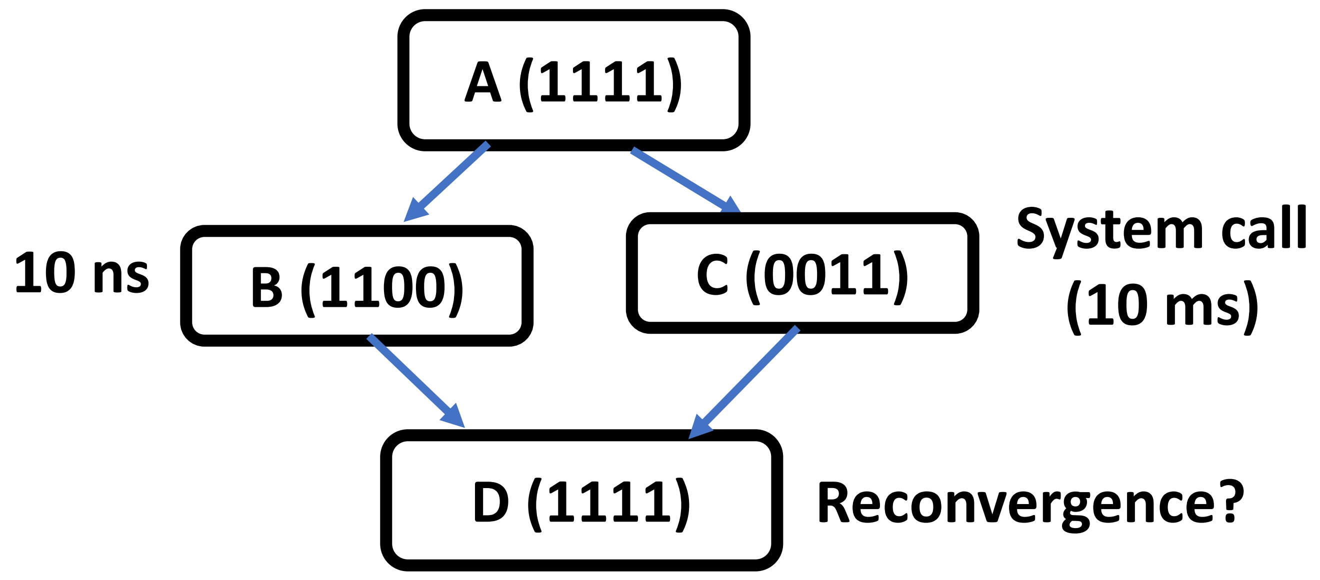
& SIMT efficiency and memory model scalability of the GPU

*NMCA: non-multi copy atomicity

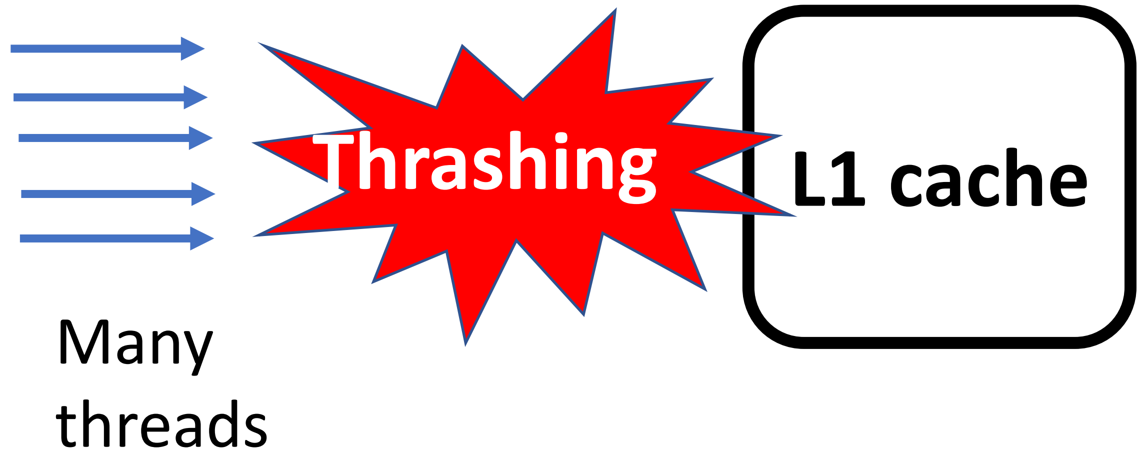
Ren, Xiaowei, et al. “HMG: Extending cache coherence protocols across modern hierarchical multi-GPU systems." HPCA 2020

RPU's Challenges

- Control Divergence
  - Challenge: Control divergence with high latency path
  - Solution: Optimized batching & System-level batch split



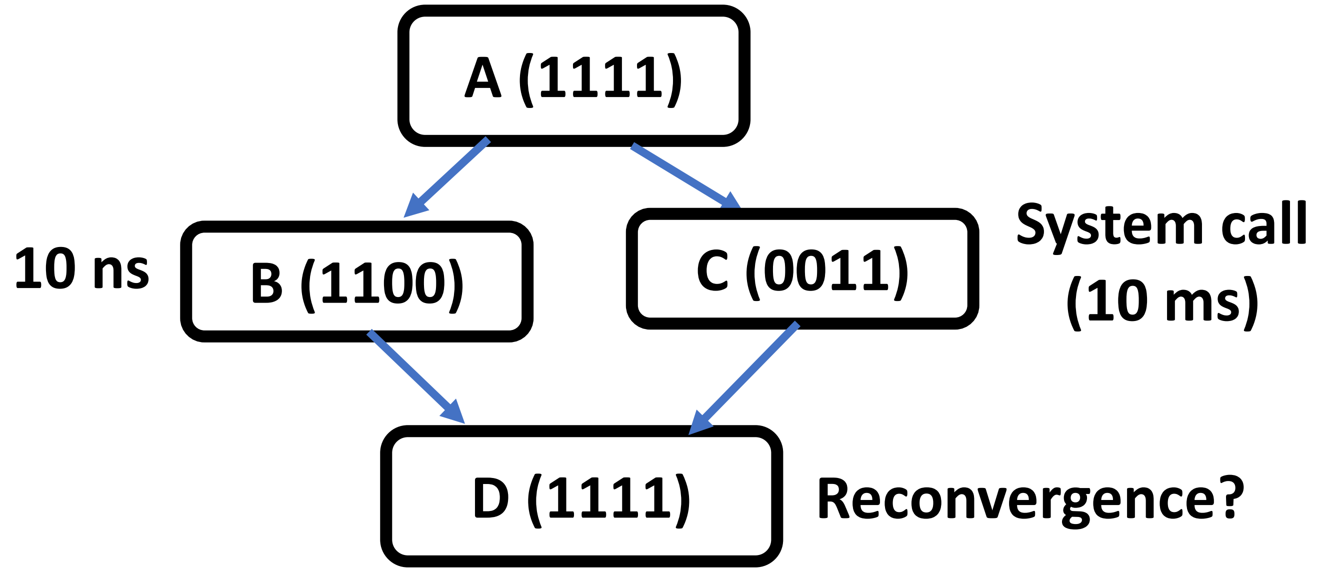
- Memory Divergence
  - Challenge: Cache/TLB contention & bank conflicts
  - Solution: Batch tuning, stack/memory coalescing and SIMR-aware memory allocation



- Larger execution units & cache resources
  - Challenge: Higher instruction execution & L1 hit latency
  - Solution: Exploit low IPC, less generated traffic and employ sub-batching interleaving

RPU’s Challenges

- Control Divergence
  - <u>Challenge</u>: Control divergence with high latency path
  - <u>Solution</u>: Optimized batching & System-level batch split

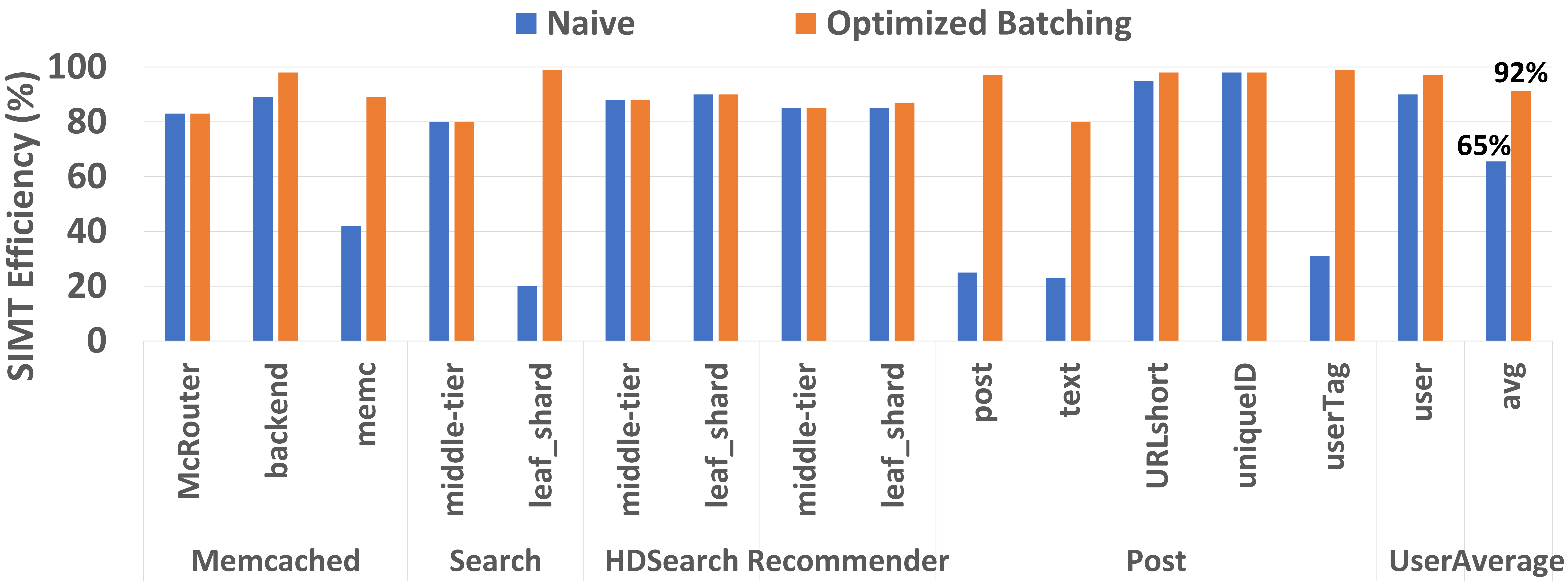


- Mem
  - C
  - <u>Solution</u>: Batch tuning, stack/memory coalescing and SIMR-aware memory allocation

Read more details in the paper on how we address these challenges

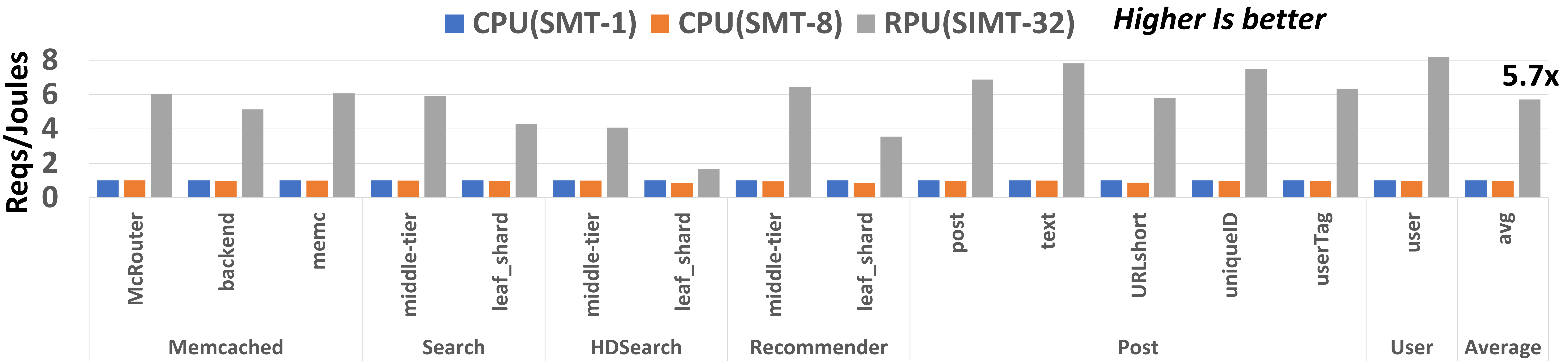
- Larger execution units & cache resources
  - <u>Challenge</u>: Higher instruction execution & L1 hit latency
  - <u>Solution</u>: Exploit low IPC, less generated traffic and employ sub-batching interleaving

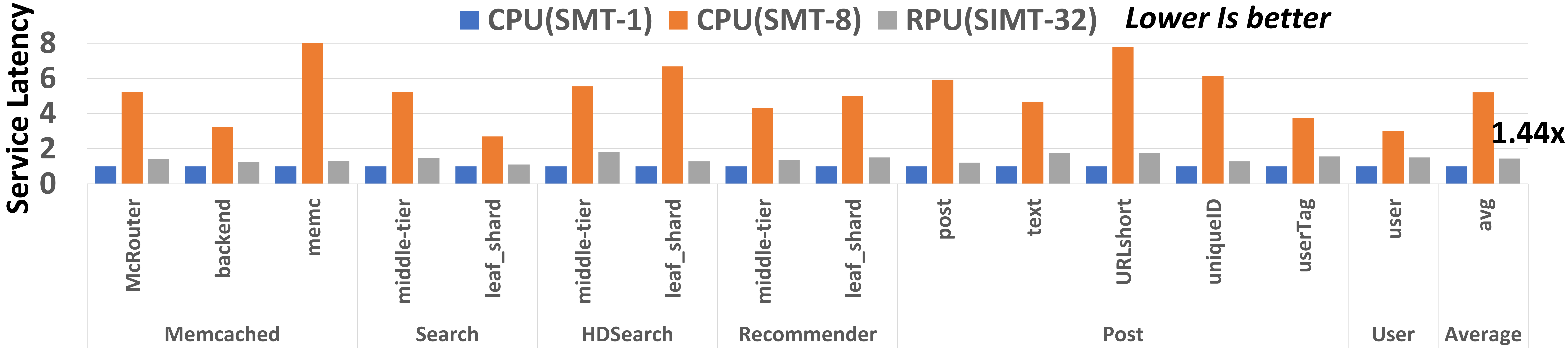
SIMT Control Efficiency



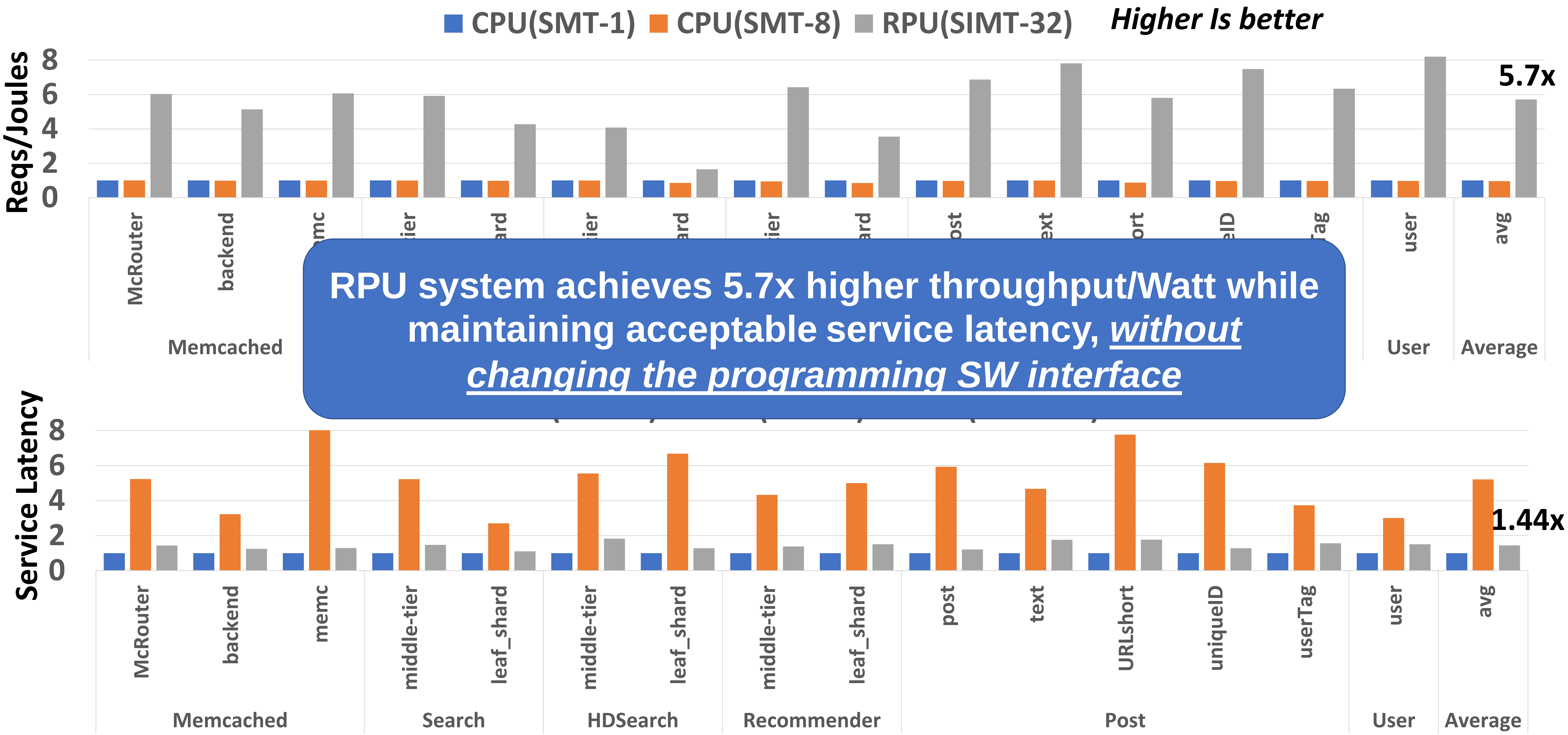
Notes: (1) Batch Size = 32 & #batches=75, (2) System Calls are not traced, (3) SIMT Eff = scalar-instructions / (batch-instructions * batch-size), (4) fine-grain locking are assumed. Other assumptions are included in the paper.

Efficiency and Service Latency Results (Simulation)





Efficiency and Service Latency Results (Simulation)



RPU system achieves 5.7x higher throughput/Watt while maintaining acceptable service latency, *without changing the programming SW interface*

Summary

- *Request Similarity* is abundant in the data center.
- We start with *OoO CPU* design and augment it with *SIMT execution* to maximize chip utilization and exploit the similarity.
- We co-design the software stack to support *batching* and awareness of SIMT execution.

SIMT efficiency is high in the open-source microservices we study.

DeathStarBench

μSuite: A Benchmark Suite for Microservices

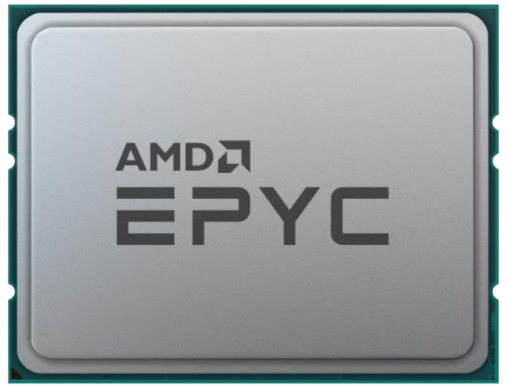
We are very interested in evaluating SIMT control efficiency in proprietary production microservices.

Google

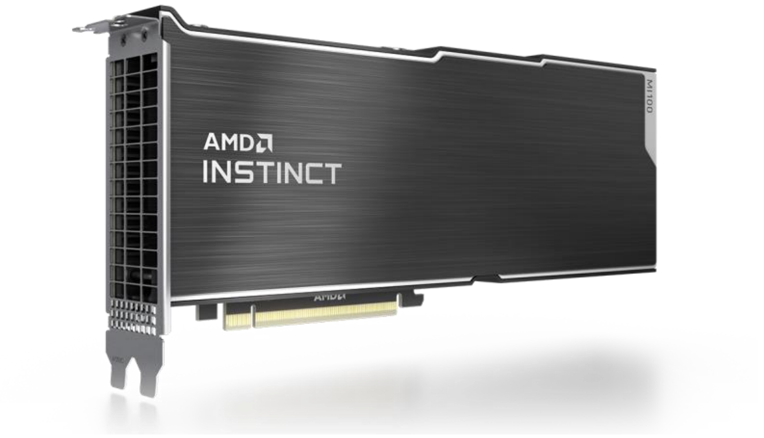
facebook

Thank You!
Q&A?

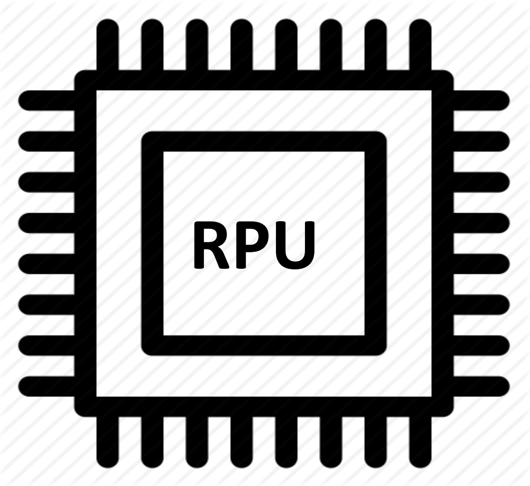
Instruction level parallelism (ILP) &
Thread level parallelism (TLP)



Data level parallelism (DLP)

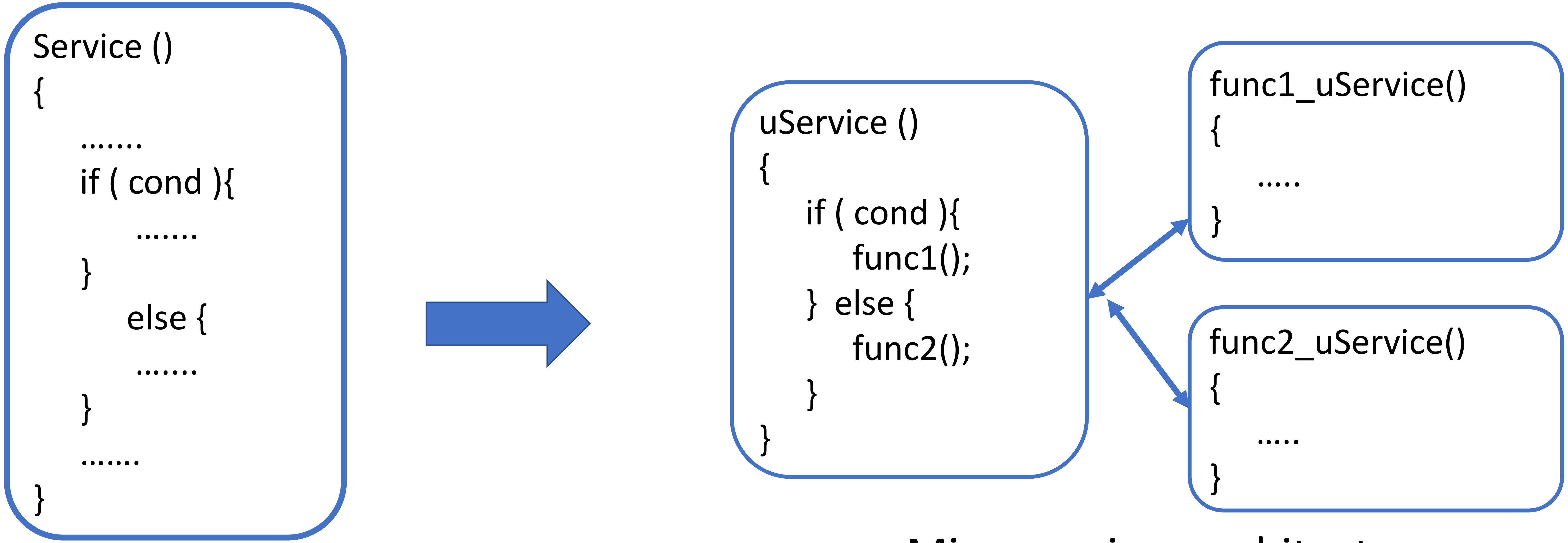


Request level parallelism (RLP)



Backup Slides

SIMT-friendly Microservices



Monolithic Service

Microservices architecture

+Smaller cache footprint

+Less divergent

Key Observation#3: Microservices reduce the per-thread cache requirement and minimize control-flow variations between concurrent threads

Batching Optimization

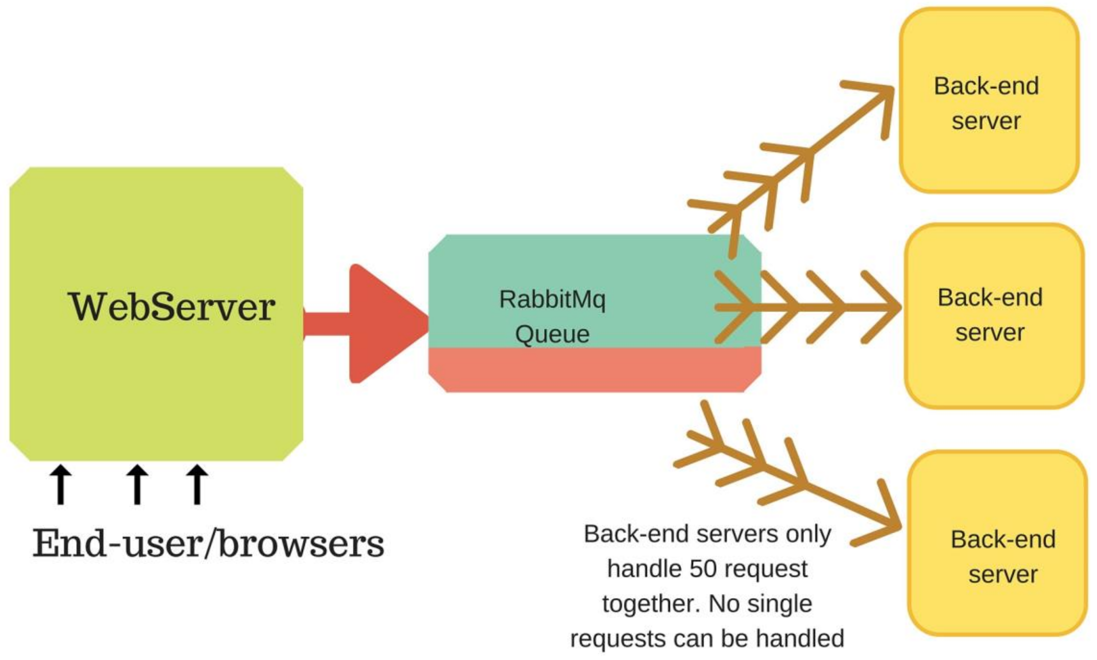
From Google's Production DL Inference

Production						MLPerf 0.7		
DNN	ms	batch	DNN	ms	batch	DNN	ms	batch
MLP0	7	200	RNN0	60	8	Resnet50	15	16
MLP1	20	168	RNN1	10	32	SSD	100	4
CNN0	10	8	BERT0	5	128	GNMT	250	16
CNN1	32	32	BERT1	10	64			

Table 5. Latency limit in ms and batch size picked for TPUv4i.

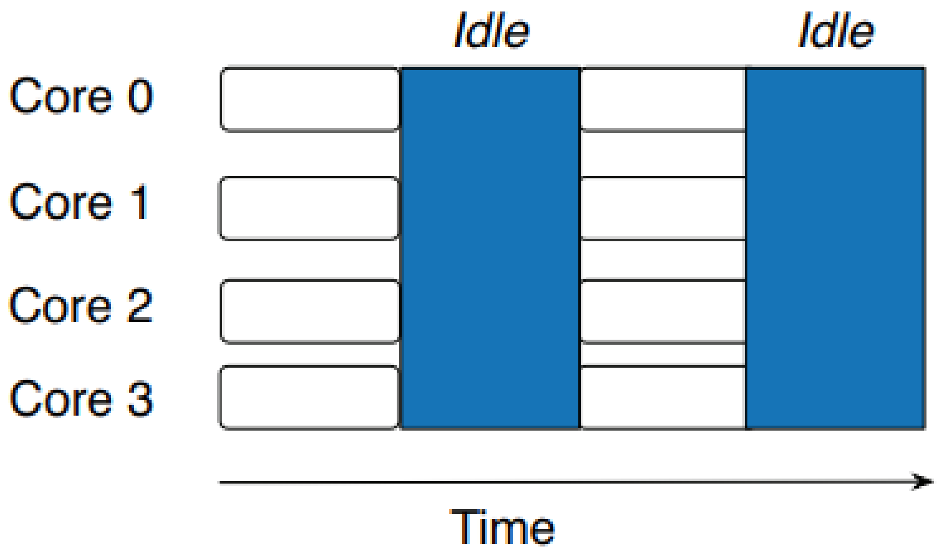
DL Inference Batching

Memcached servers



Network Batching

Power management

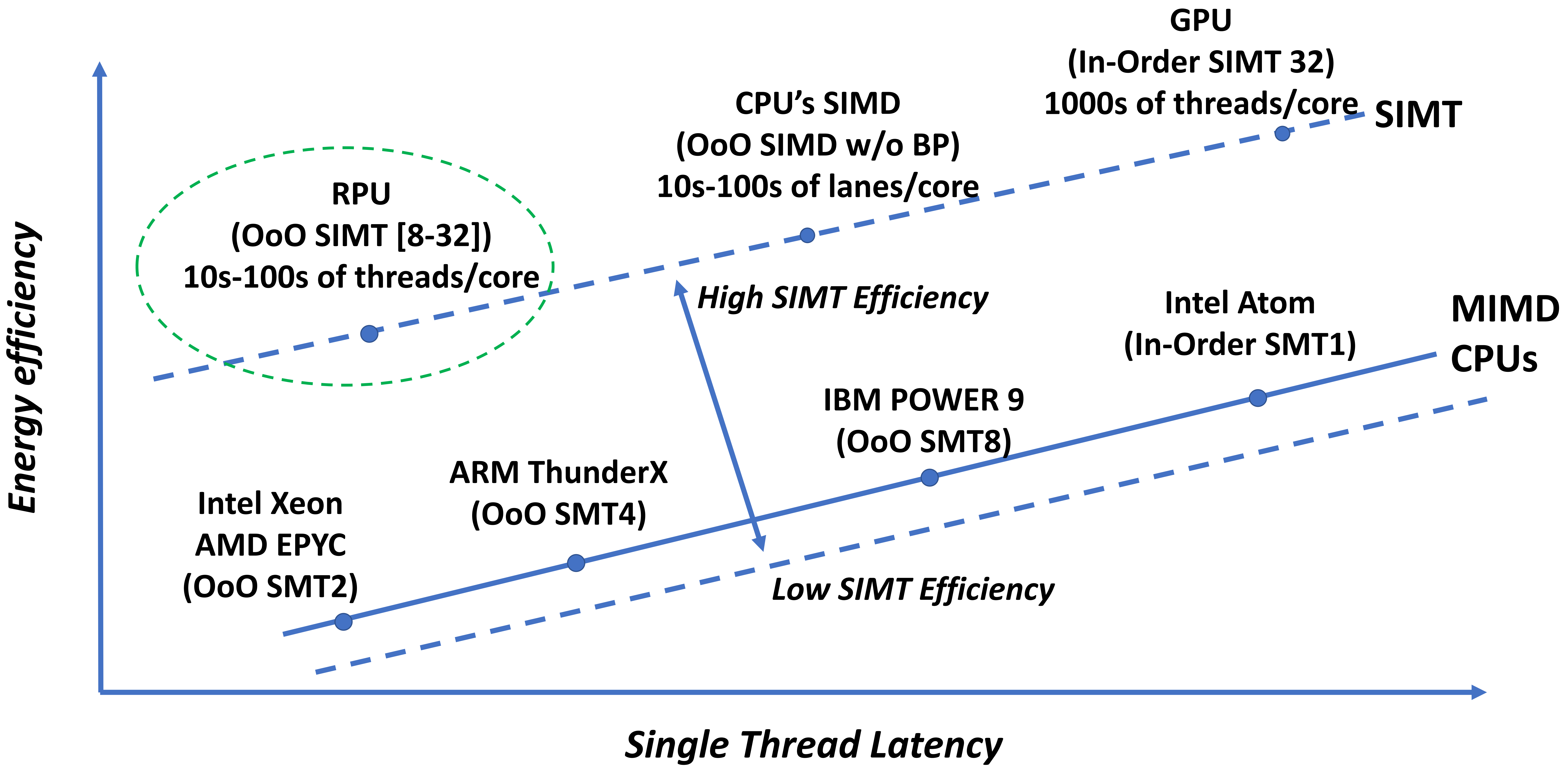


Batching for deep sleep

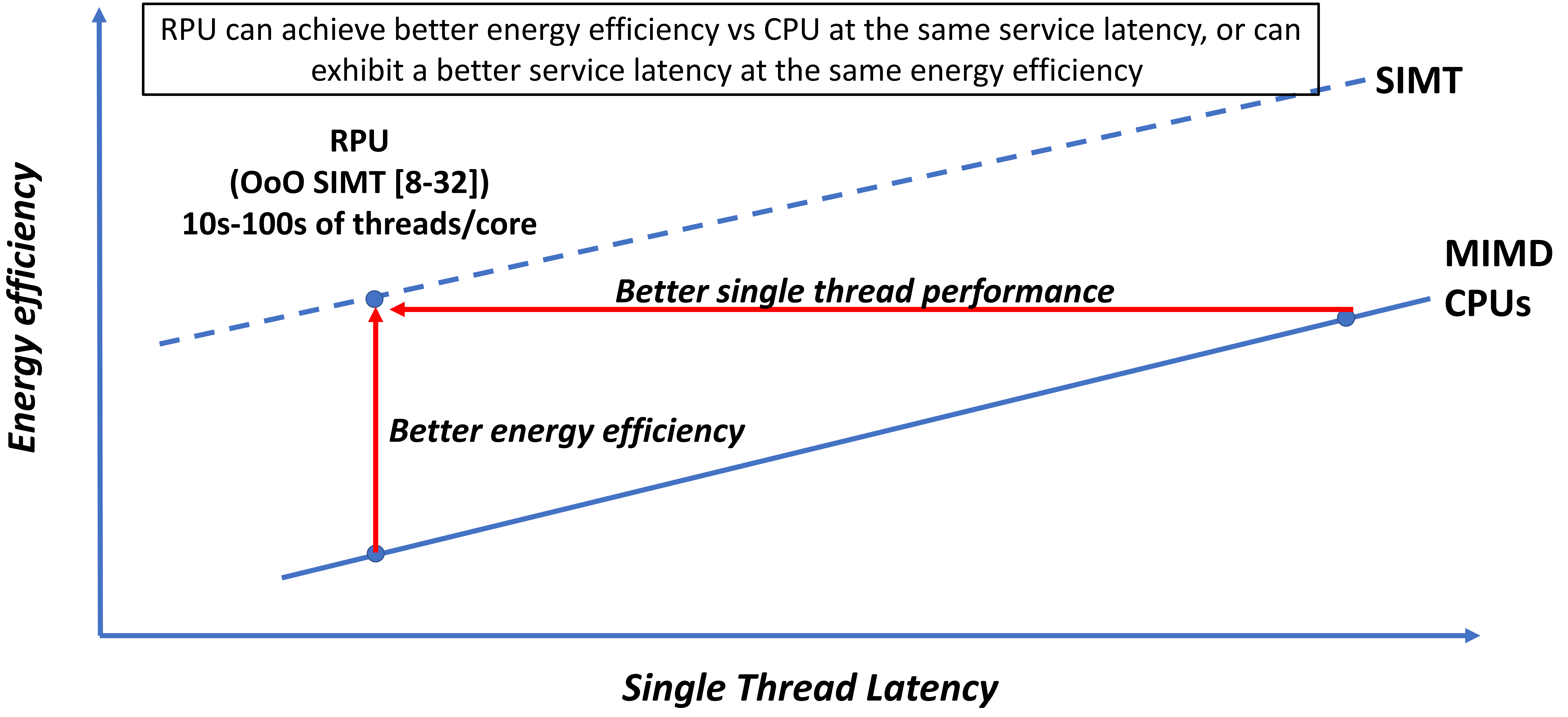
Key Observation#4: Modern data centers already rely on request batching heavily

Jouppi, Norman P., et al. "Ten Lessons From Three Generations Shaped Google's TPUv4i: Industrial Product." *2021 ISCA*
https://memcached.org/blog/nvm-multidisk/
Meisner, David, and Thomas F. Wenisch. "Dreamweaver: architectural support for deep sleep." *ASPLOS 2012*

Latency & Energy-Efficiency Tradeoff



Latency & Energy-Efficiency Tradeoff



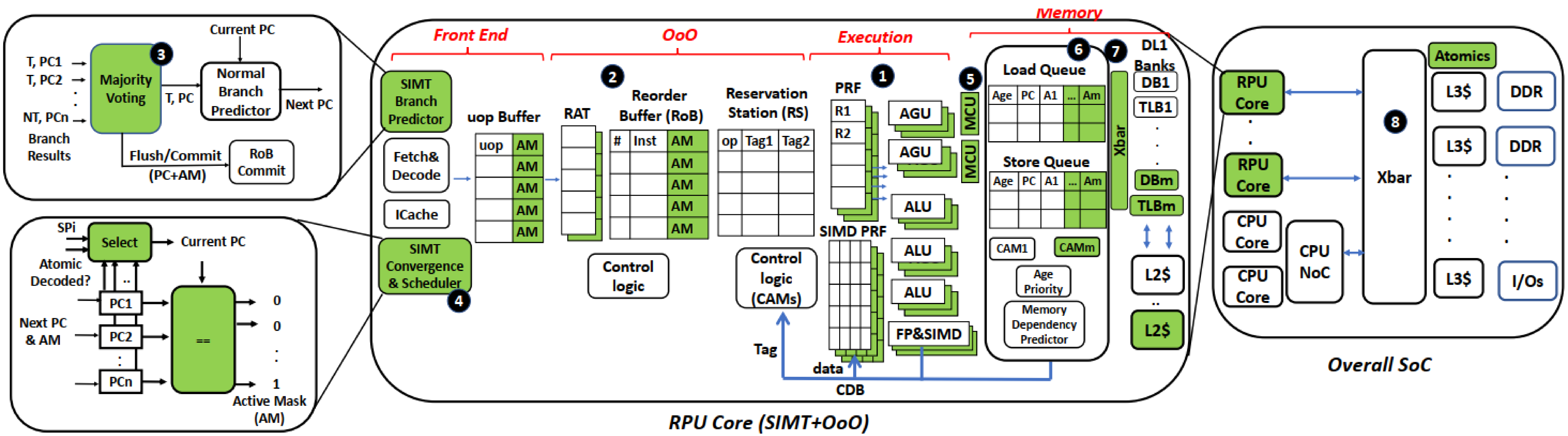
HW/SW Stack

CPU SW Stack	GPU SW Stack	RPU SW Stack
Webservice (C++, PHP, …)	CUDA	Webservice (C++, PHP, …)
ARM/x86 compiler	CUDA compiler	ARM/x86 compiler
HTTP server	Nvidia Triton HTTP server	Batch-aware HTTP server
Runtime/libs (pthread, cstdlib, ..)	CUDA runtime/libs (cudalib, tensorRT, ..)	Runtime/libs (pthread, cstdlib, ..)
OS (Process, VM, I/Os)	OS (I/Os management)	OS (I/Os management)
	CUDA driver (VM/thread management)	RPU driver (VM/thread management)
Multi Core CPU	GPU Hardware	RPU Hardware

→ For RPU, we keep the SW programming interface as in the CPU

→ Some VM&process management system calls are reimplemented in the RPU driver to be batch-aware

RPU HW



Current PC
T, PC1
T, PC2
NT, PCn
Majority Voting
Normal Branch Predictor
T, PC
Next PC
Branch Results
Flush/Commit (PC+AM)
RoB Commit
SPi
Select
Current PC
Atomic Decoded?
Next PC & AM
PC1
PC2
PCn
==
Active Mask (AM)
Front End
OoO
Execution
Memory
SIMT Branch Predictor
Fetch& Decode
ICache
SIMT Convergence & Scheduler
uop Buffer
uop
AM
RAT
Reorder Buffer (RoB)
#
Inst
Reservation Station (RS)
op
Tag1
Tag2
Control logic
Control logic (CAMs)
Tag
PRF
R1
R2
SIMD PRF
data
CDB
AGU
ALU
FP&SIMD
MCU
Load Queue
Age
PC
A1
...
Am
Store Queue
CAM1
CAMm
Age Priority
Memory Dependency Predictor
Xbar
DL1 Banks
DB1
TLB1
DBm
TLBm
L2$
RPU Core (SIMT+OoO)
RPU Core
CPU Core
CPU NoC
Xbar
Atomics
L3$
DDR
I/Os
Overall SoC

Energy Efficiency of CPU vs RPU (Analytical Model)

$$\frac{\boldsymbol{CPU\ Energy}}{\boldsymbol{RPU\ Energy}} = \frac{\boldsymbol{Execution\ Energy + Memory\ system\ Energy + Front\_OoO\ Energy + Static\ Energy}}{\boldsymbol{Execution\ Energy + (1-r)\,(Memory\ system\ Energy) + \frac{1}{n\ *\ eff}\,\underbrace{[\,Front\_OoO\ Energy + r * Memory\ system\ Energy + Static\ Energy]}_{\text{Amortized factors = 50-90\%}} + SIMT\_Overhead}}$$

data locality ratio =75%

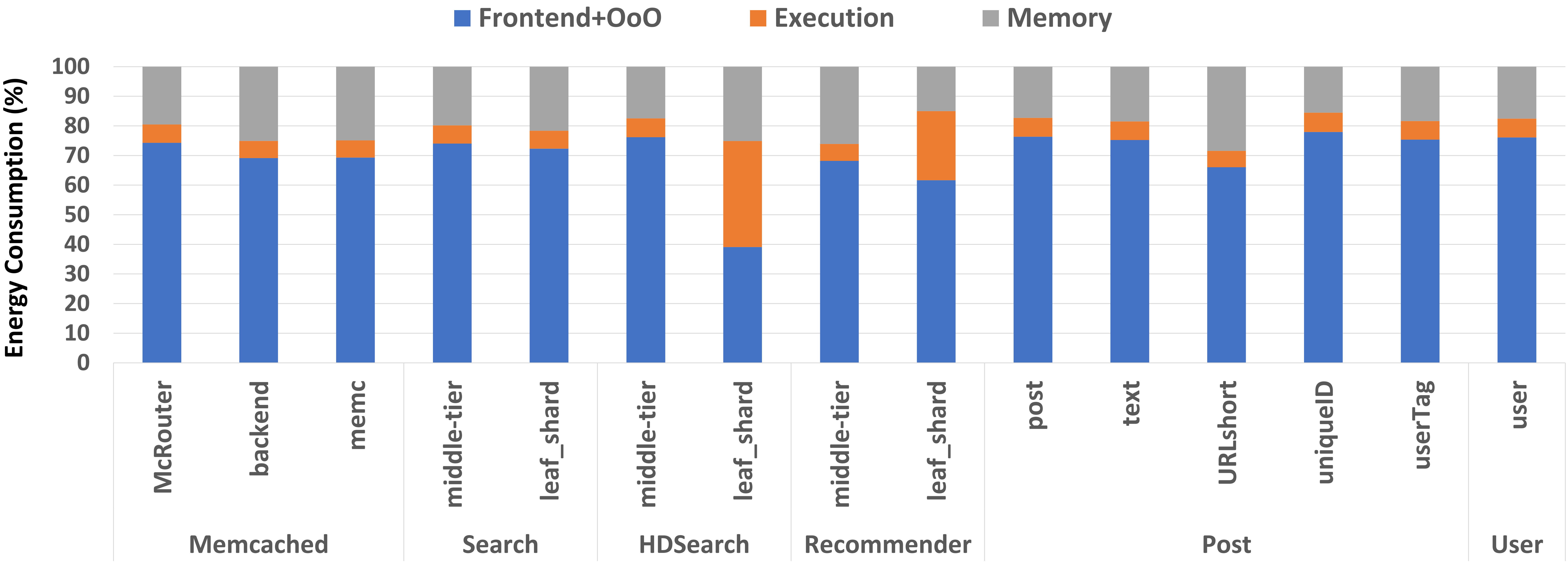
batch size (n) = 8-32

SIMT Efficiency=92%

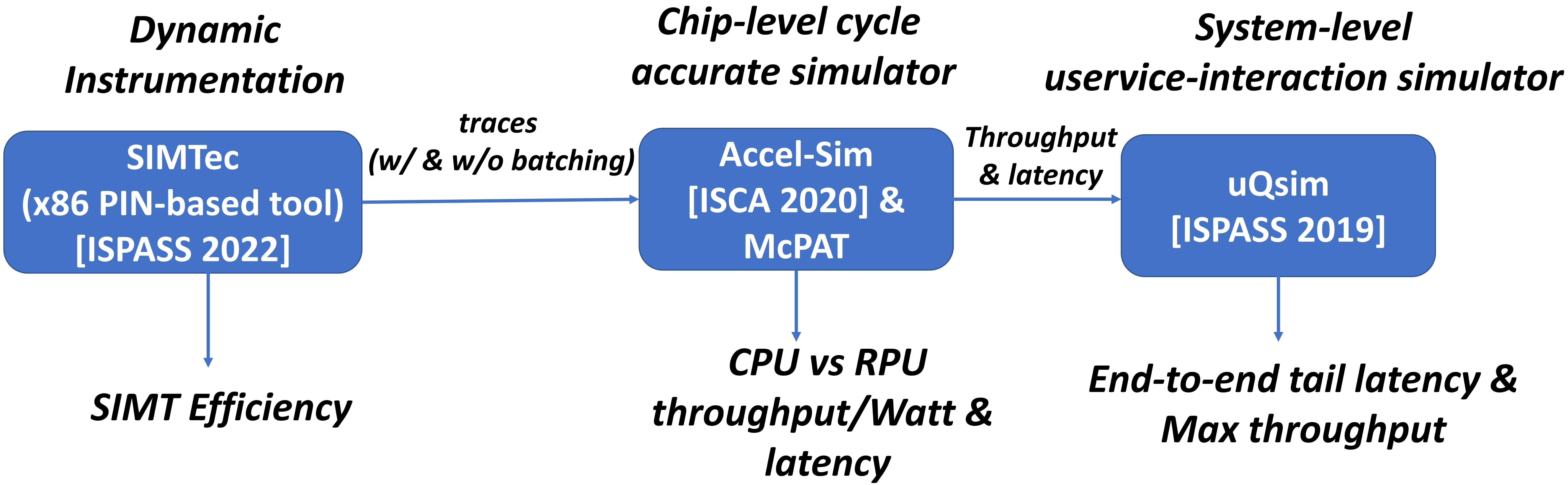
Larger L1/L2
MCUs
Active mask
etc.

→ *an anticipated 2-10x energy efficiency gain can be achieved with RPU vs CPU*

CPU Dynamic Energy Breakdown



Experimental Setup



Workloads: Social Network Microservices

Microsuite [IISWC 2018], DeathStarBench [ASPLOS 2020] and In-house benchmarks

Libraries: c++ stdlib, Intel MKL, OpenSSL, FLANN, Pthread, zlib, protobuf, gRPC and MLPack, …

Khairy, Mahmoud, et al. "Accel-Sim: An extensible simulation framework for validated GPU modeling." *ISCA 2020*
Zhang, Yanqi, Yu Gan, and Christina Delimitrou. "uqSim: Scalable and Validated Simulation of Cloud Microservices." ISPASS 2019
Alawneh, Ahmad , et al. "A SIMT Analyzer for Multi-Threaded CPU Applications." ISPASS 2022
Sriraman, Akshitha, and Thomas F. Wenisch. "µ suite: a benchmark suite for microservices." IISWC 2018
Gan, Yu, et al. "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems." ASPLOS 2019
Li, Sheng, et al. "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures." *MICRO 2009*

Batching Opportunity for Facebook Services

- To amortize batching overhead, you either need:
  - (1) High service latency, with low traffic so service latency will amortize batching **OR**
  - (2) High traffic, with low service latency so high traffic will amortize batching **OR**
  - (3) High traffic and high service latency (ideal case)
- Let’s take a look at Facebook in-production services:

μservice	Throughput (QPS)	Req. latency	Insn./query
Web	O (100)	O (ms)	O (10^6)
Feed1	O (1000)	O (ms)	O (10^9)
Feed2	O (10)	O (s)	O (10^9)
Ads1	O (10)	O (ms)	O (10^9)
Ads2	O (100)	O (ms)	O (10^9)
Cache1	O (100K)	O (μs)	O (10^3)
Cache2	O (100K)	O (μs)	O (10^3)

Low traffic but high latency

Low latency but high traffic

Note: I was not able to calculate the exact batching overhead as the exact numbers are not shown and SLA (P99 latency) is not specified.

Sriraman, Akshitha, Abhishek Dhanotia, and Thomas F. Wenisch. "Softsku: Optimizing server architectures for microservice diversity@ scale.“ASPLOS 2019

Batching Opportunity for Google Services

- (1) From Google in-production ML inference services:
  - Batching is widely used for DL inference with size = 8-20 reqs based on traffic and latency

Production						*MLPerf 0.7*		
DNN	*ms*	*batch*	*DNN*	*ms*	*batch*	*DNN*	*ms*	*batch*
MLP0	7	200	RNN0	60	8	Resnet50	15	16
MLP1	20	168	RNN1	10	32	SSD	100	4
CNN0	10	8	BERT0	5	128	GNMT	250	16
CNN1	32	32	BERT1	10	64			

Table 5. Latency limit in ms and batch size picked for TPUv4i.

Quoted: “Clearly, datacenter applications limit latency, not batch size. Future DSAs should take advantage of larger batch sizes”

- (2) Further, Google search service has a high service latency (~10s ms) and high traffic (~100K QPS), so they are a good candidate for batching

Jouppi, Norman P., et al. "Ten Lessons From Three Generations Shaped Google’s TPUv4i: Industrial Product." *2021 ISCA*