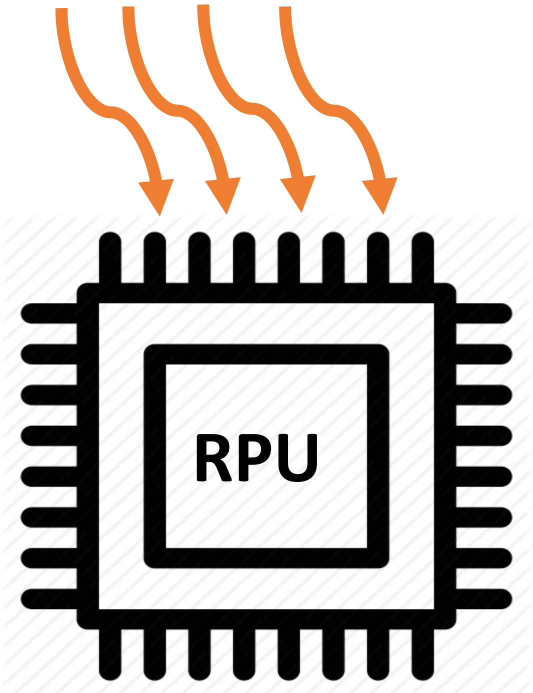
MICRO 2022



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

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

Purdue University

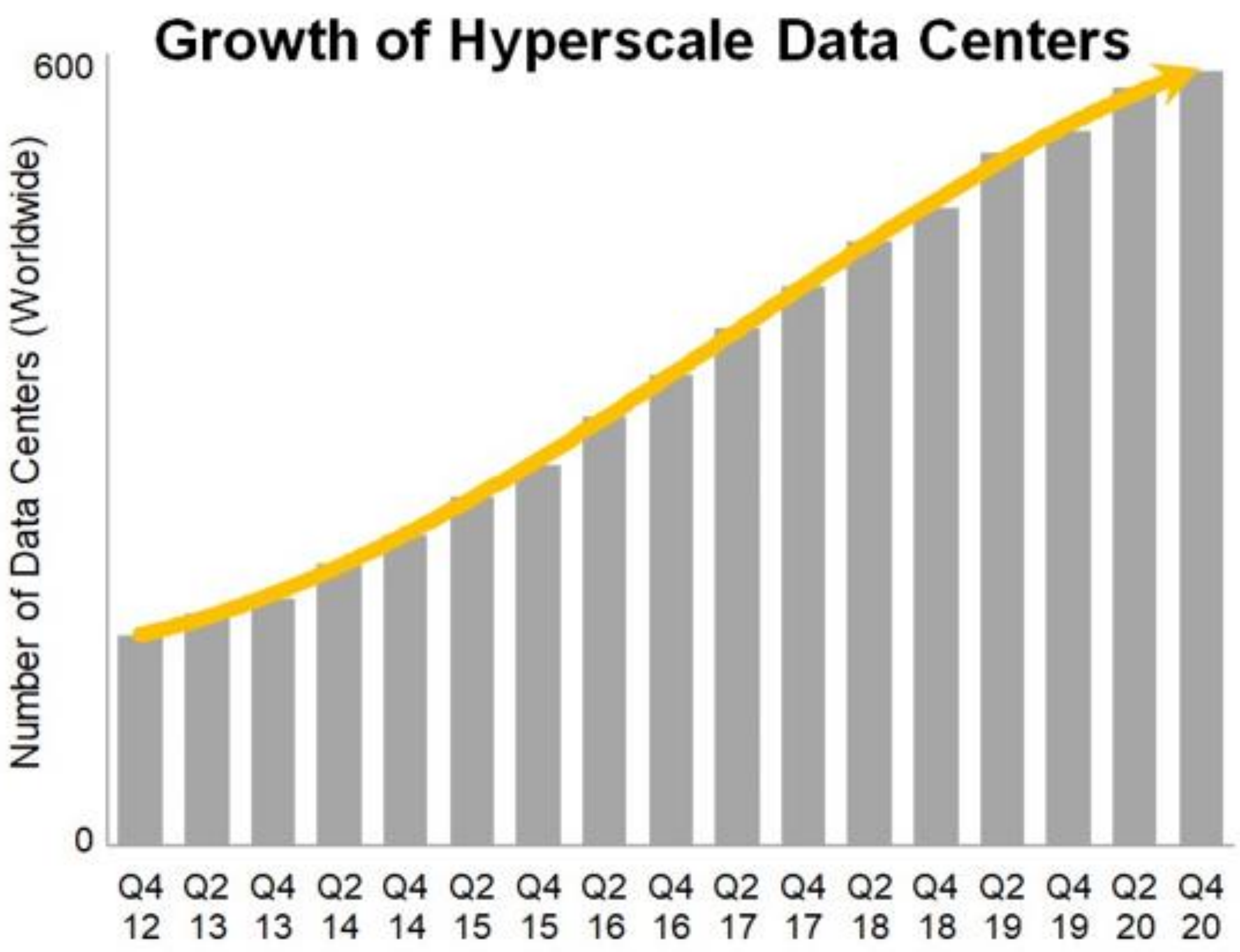
Home page: https://mkhairy.github.io/
Contact: abdallm@purdue.edu

*Currently at AMD Research

10/3/2022

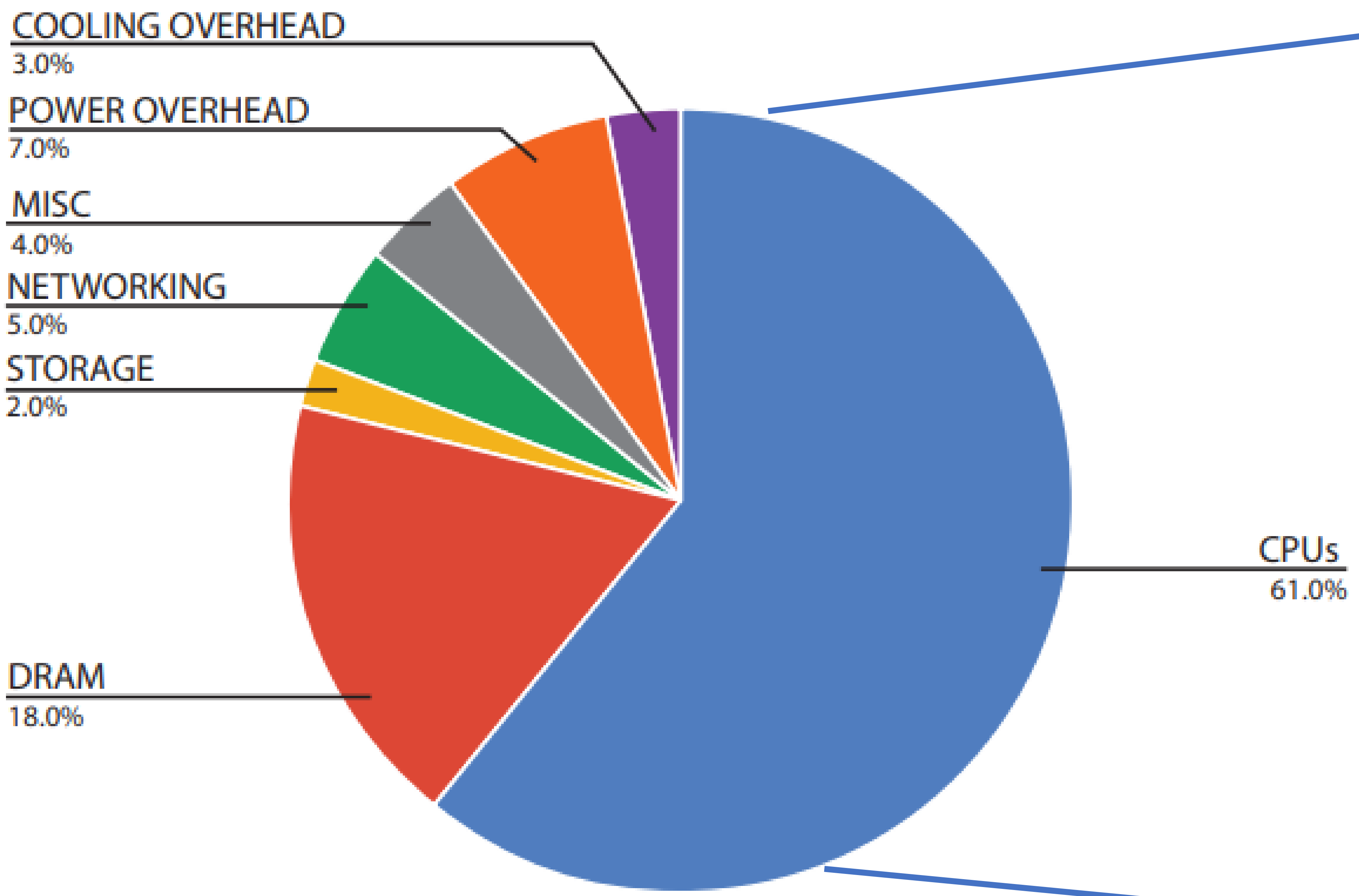
Growth of Hyperscale Data Centers

- The growth of hyperscale data centers has steadily increased in the last decade
- The next era of IoT and AI
- Challenges:
  - Slowing growth of Moore's law
  - High power consumption
  - Large carbon footprint
  - By 2030, the data centers will consume 9% of the total electricity demand

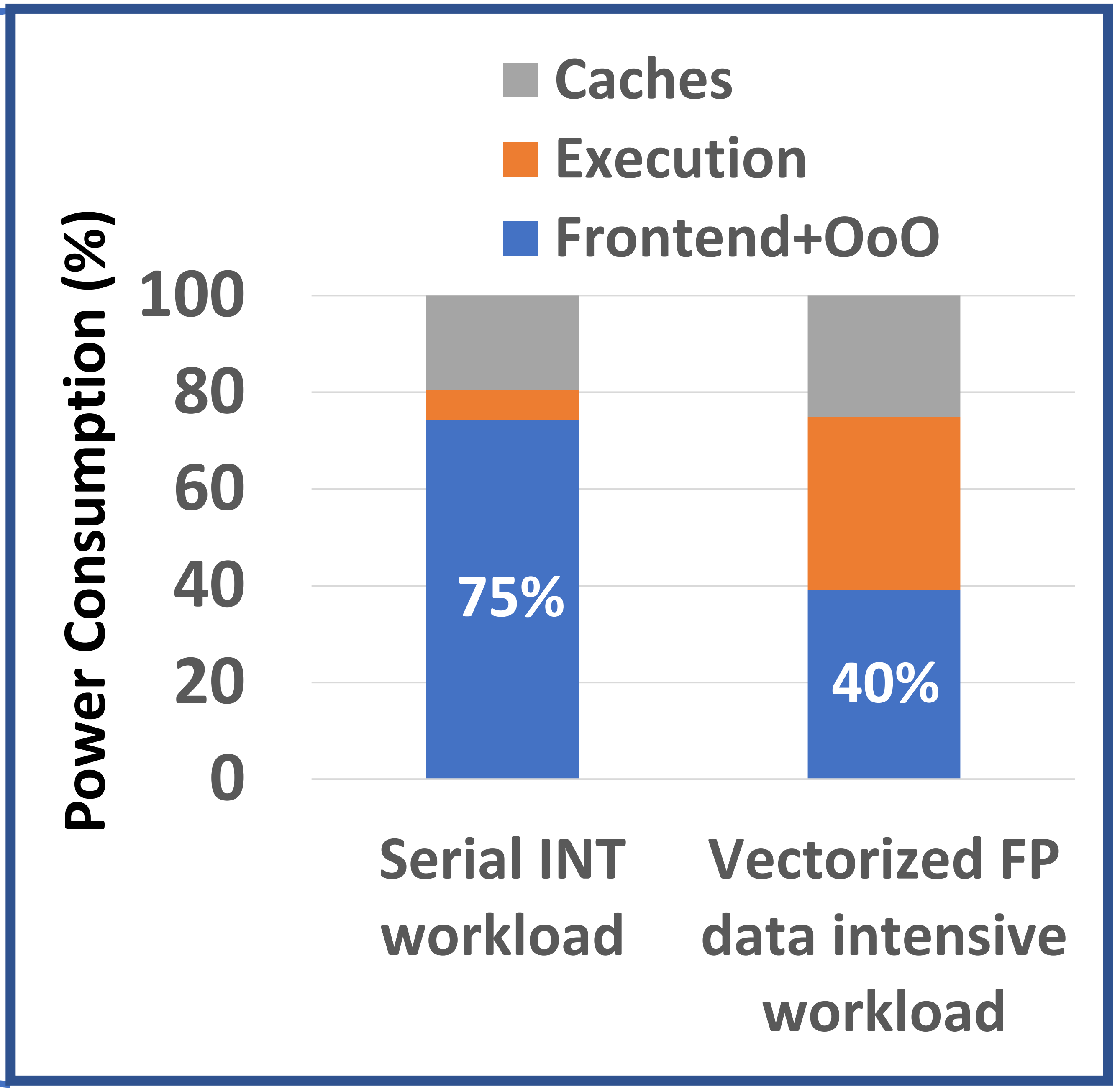


https://www.datacenterknowledge.com/cloud/analysts-there-are-now-more-500-hyperscale-data-centers-world
https://www.nature.com/articles/d41586-018-06610-y

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

Frontend+OoO Overhead is Increasing

16 KB I$
3-wide fetch
1 KB BTB

32 KB I$
4-wide fetch
2 KB BTB

64 KB I$
5-wide fetch
4 KB BTB

Frontend is getting larger

128 ROB entries
6-wide issue

256 ROB entries
8-wide issue

352 ROB entries
12-wide issue

Pipeline OoO is getting more complex

SSE
128-bit

AVX
256-bit

AVX
512-bit

SIMD is getting wider to amortize front+OoO complexity

32-bit ALU

32-bit ALU

32-bit ALU

<u>BUT</u>, scalar units remain the same

As we move forward, the frontend+OoO overhead is getting larger compared to the scalar units

So, this was the Hardware.
What about Software? What kind of Software running in the data centers?

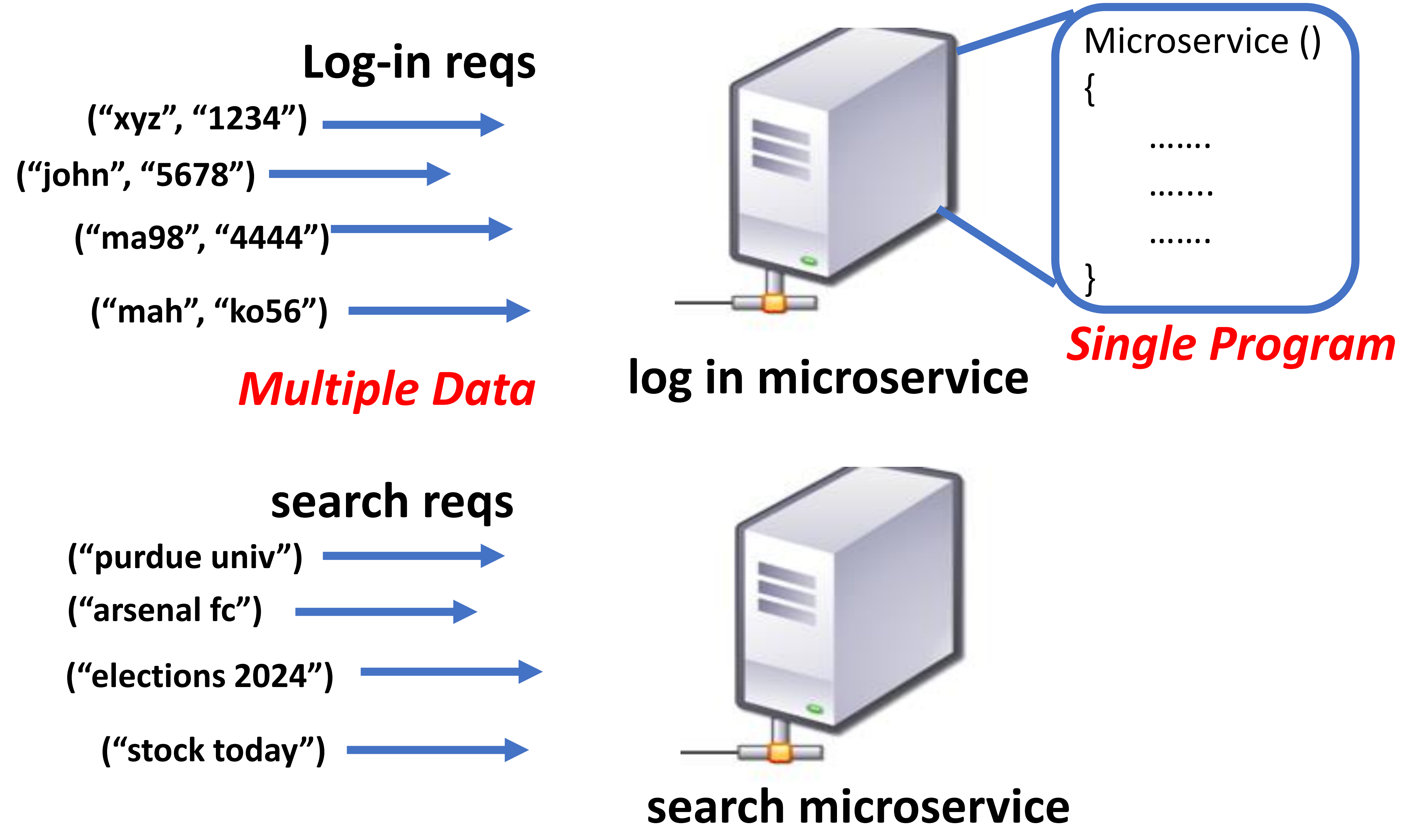
1 Application, Million of Users

Google
facebook
Private Datacenter

Uber
NETFLIX
AWS
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

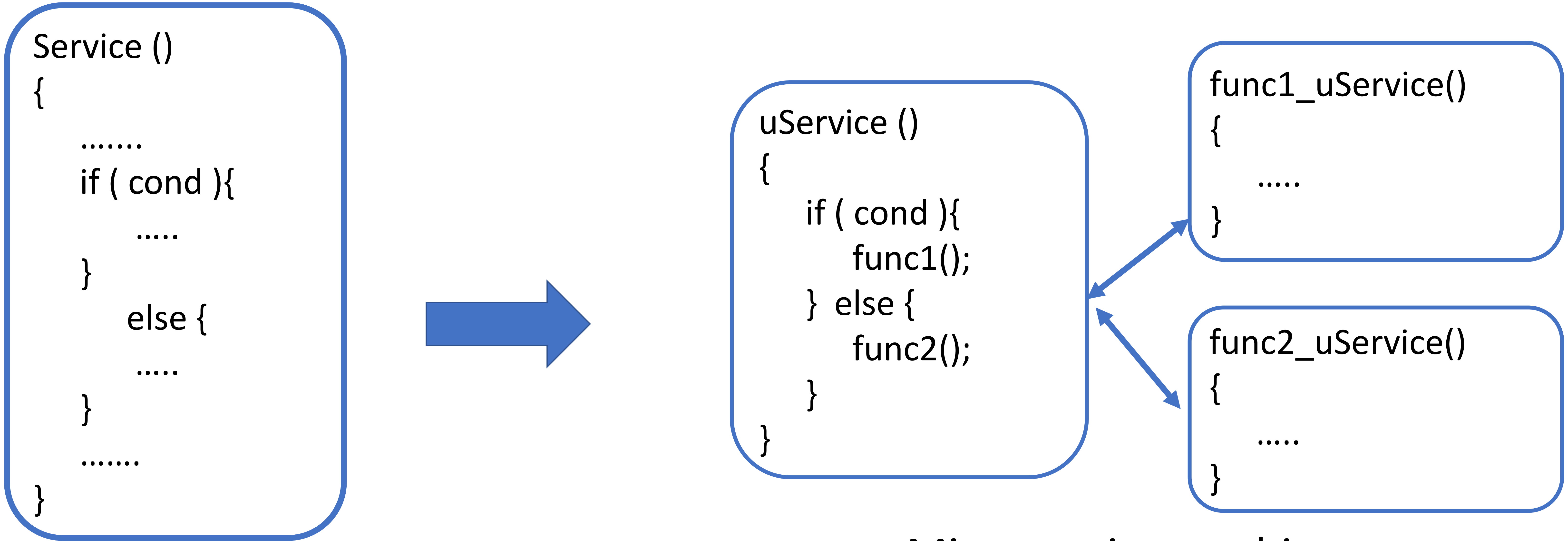
“Slower but energy-efficient wimpy cores only win for general data center workloads if their single-core speed is reasonably close to that of mid-range brawny cores”

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

SIMT-friendly Microservices



Monolithic Service

Microservices architecture
+Smaller cache footprint
+Less divergent

Key Observation#2: Microservices reduce the per-thread cache requirement and minimize control-flow variations between concurrent threads

Batching Optimization

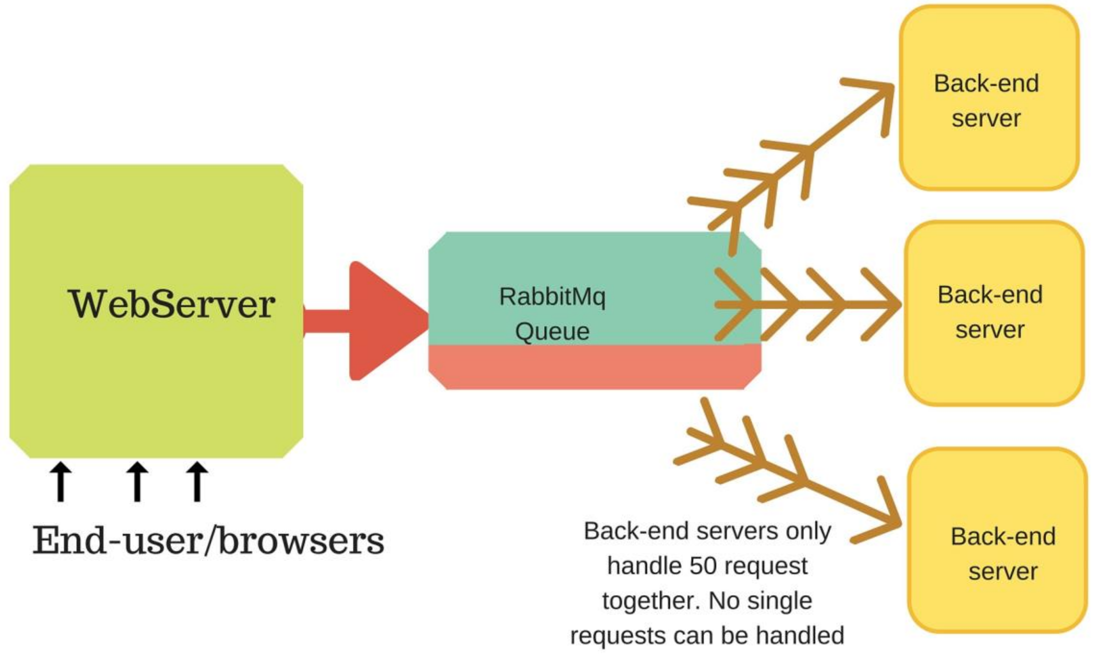
From Google's Production DL Inference

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.

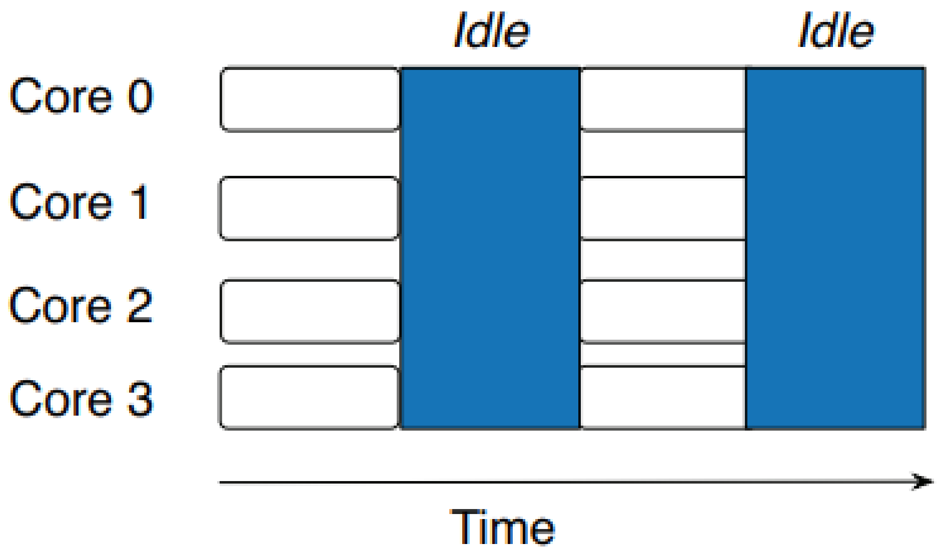
DL Inference Batching

Memcached servers



Network Batching

Power management

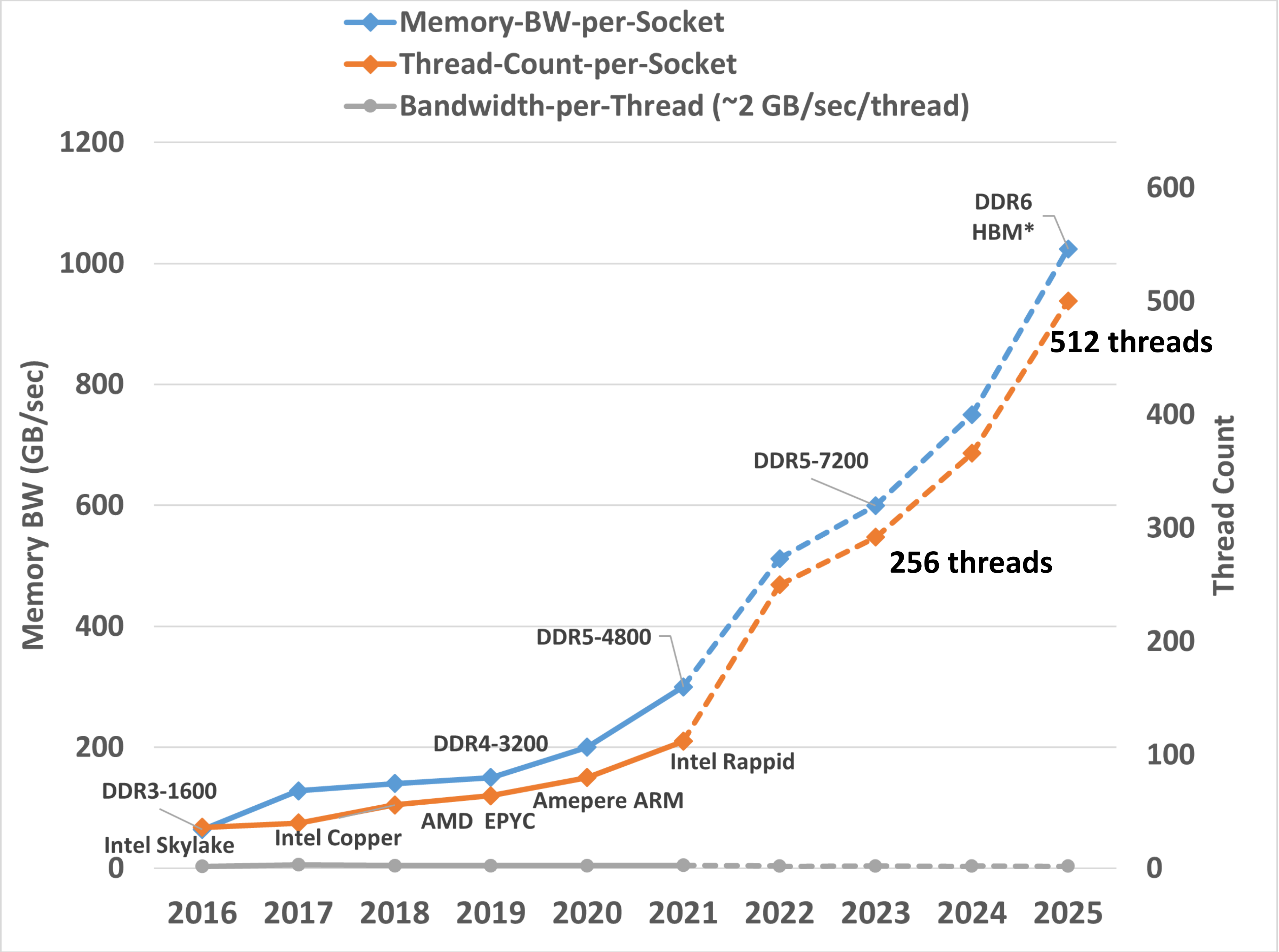


Batching for deep sleep

Key Observation#3: 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*

Off-Chip BW Scaling



Key Observation #4: There is available headroom to increase on-chip throughput (thread count) in the foreseeable future.

How to increase on-chip throughput of CPU?

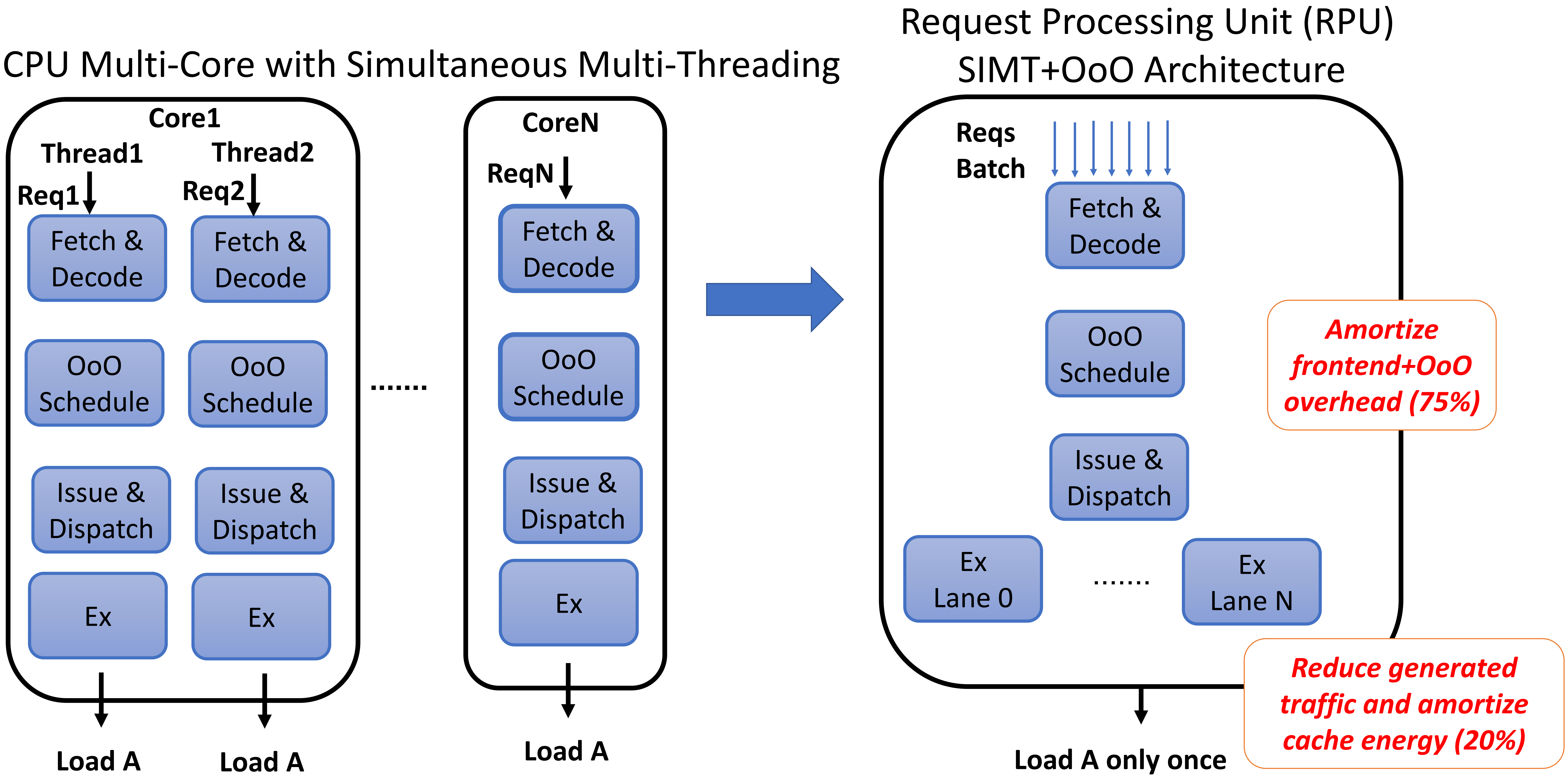
- Direction#1 (industry standard): Add more Chiplets + Cores + SMT ✗

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

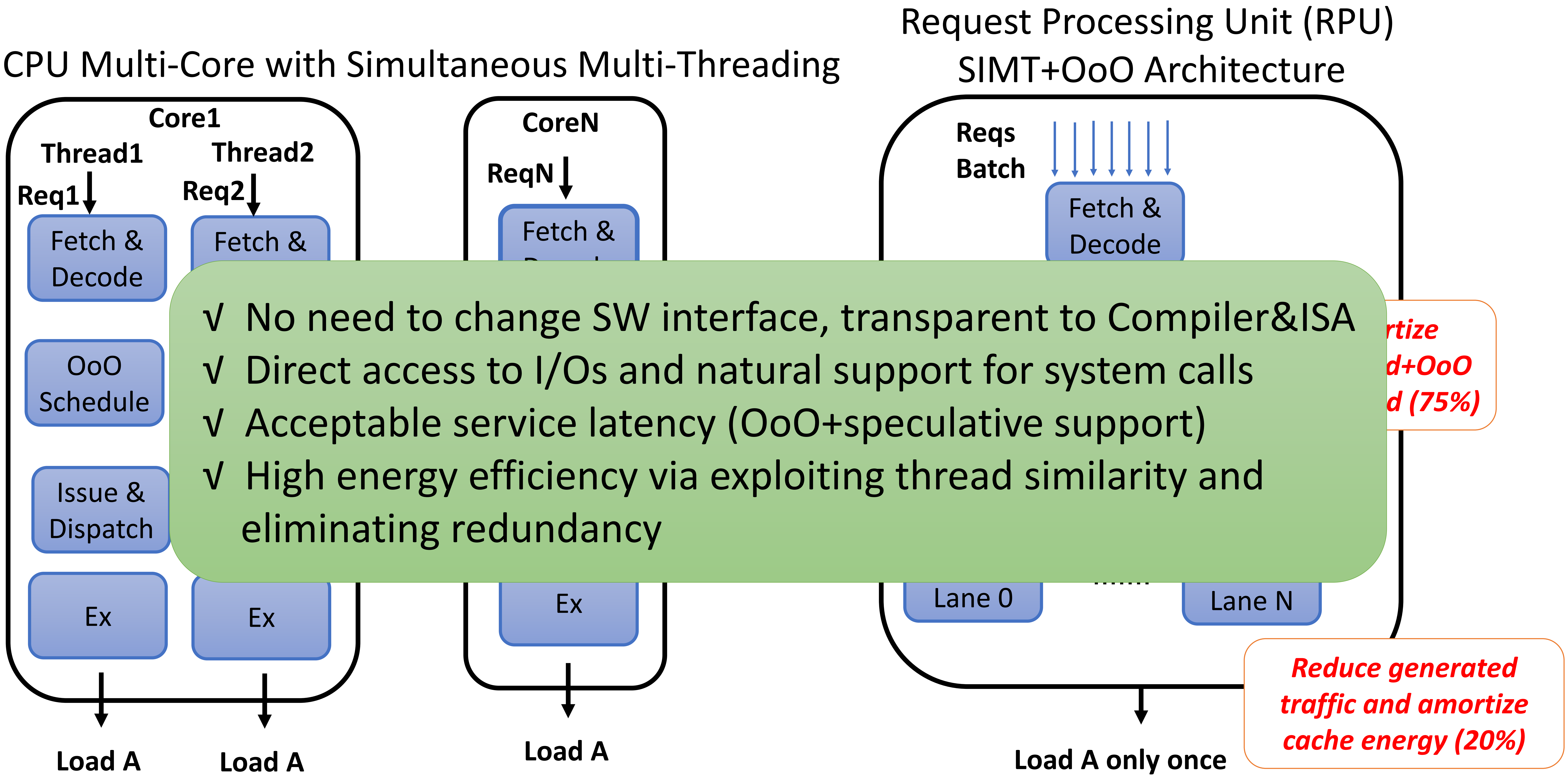
Key Idea

"Let's bring SIMT efficiency to the CPU world!"

SIMT Efficiency

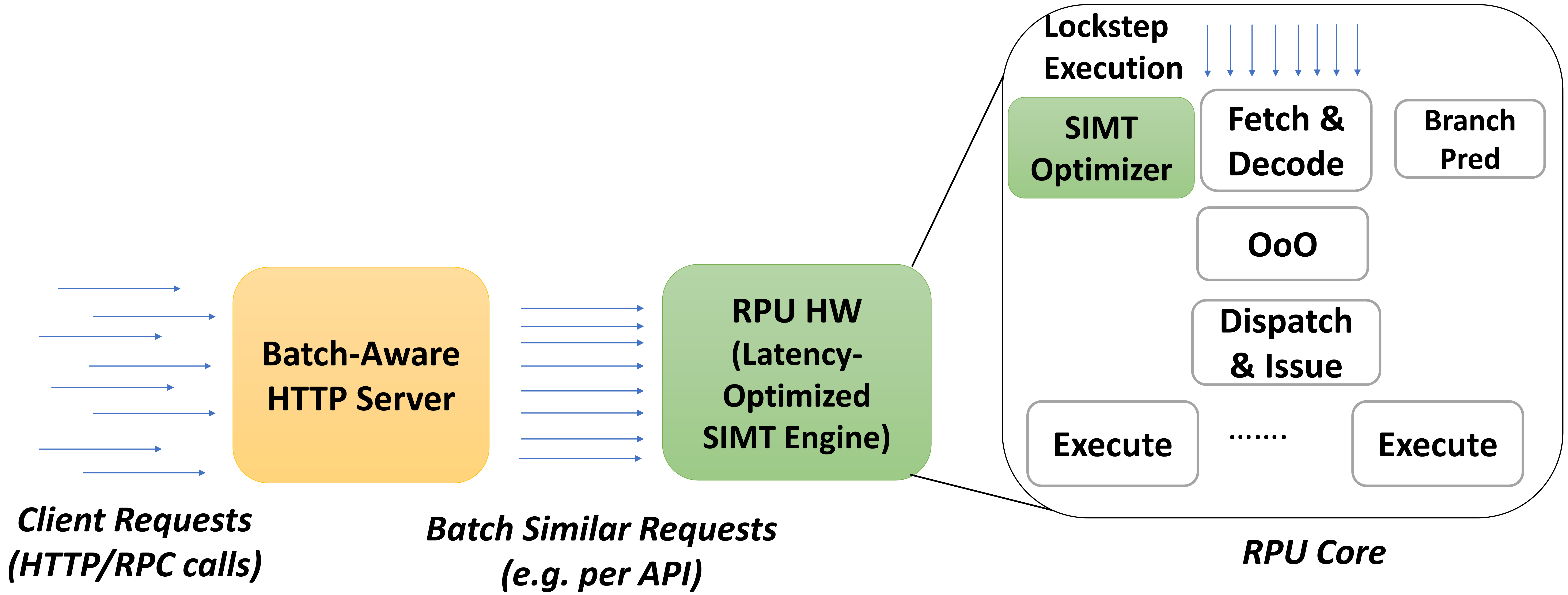


SIMT Efficiency



√ No need to change SW interface, transparent to Compiler&ISA
√ Direct access to I/Os and natural support for system calls
√ Acceptable service latency (OoO+speculative support)
√ High energy efficiency via exploiting thread similarity and eliminating redundancy

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

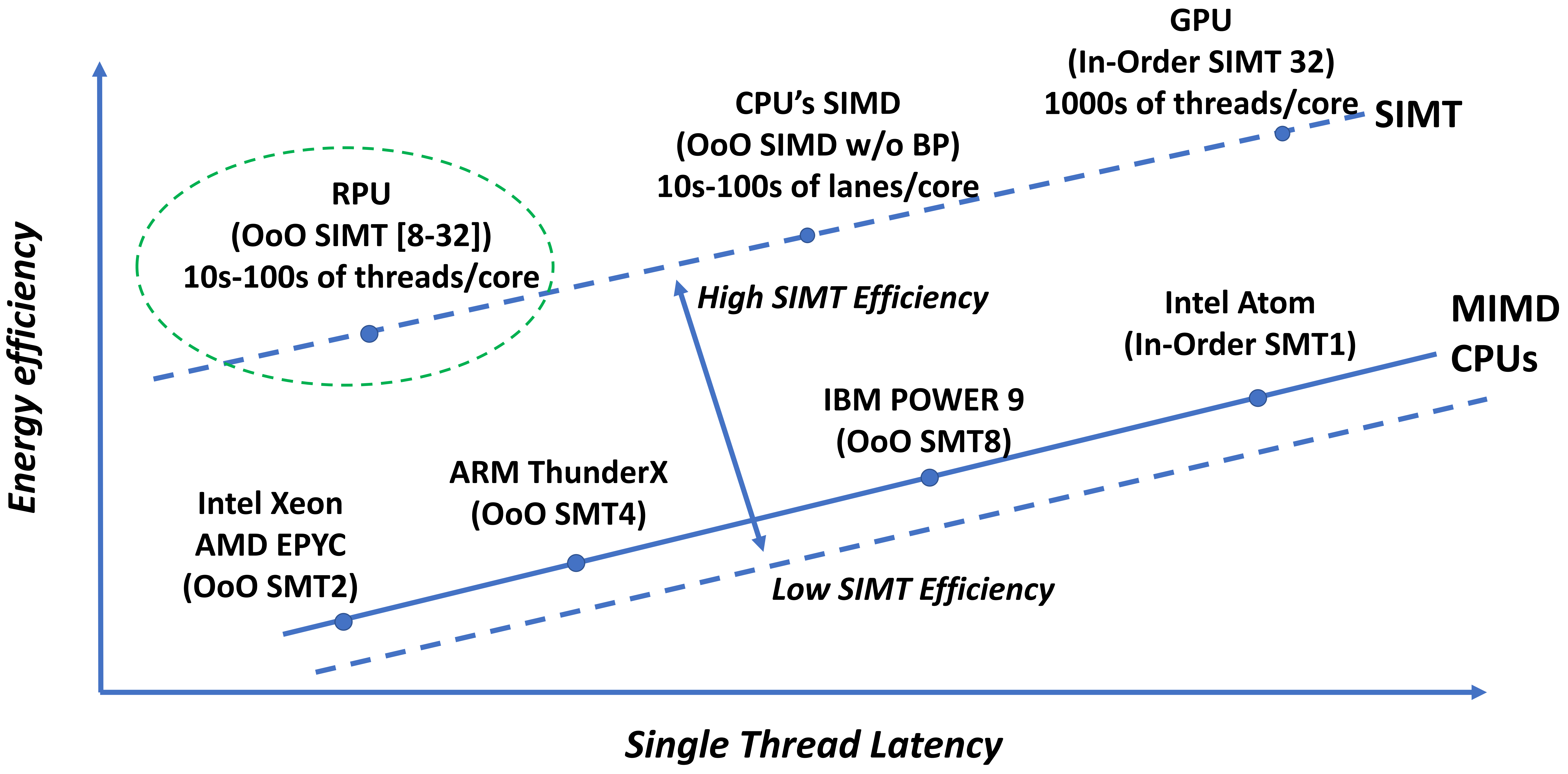
The RPU takes advantage of the latency optimizations and programmability of the CPU

& 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

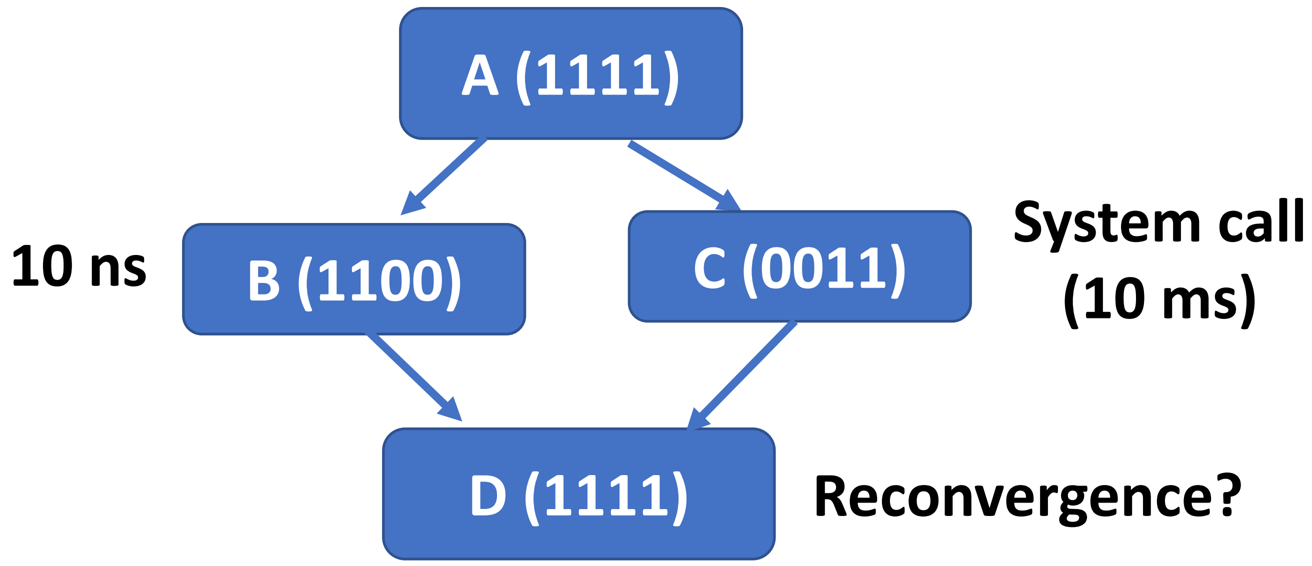
Latency & Energy-Efficiency Tradeoff



Deep Dive into RPU's Challenges

Deep Dive into RPU’s Challenges

- Control Divergence
  - Control divergence wit high latency branch

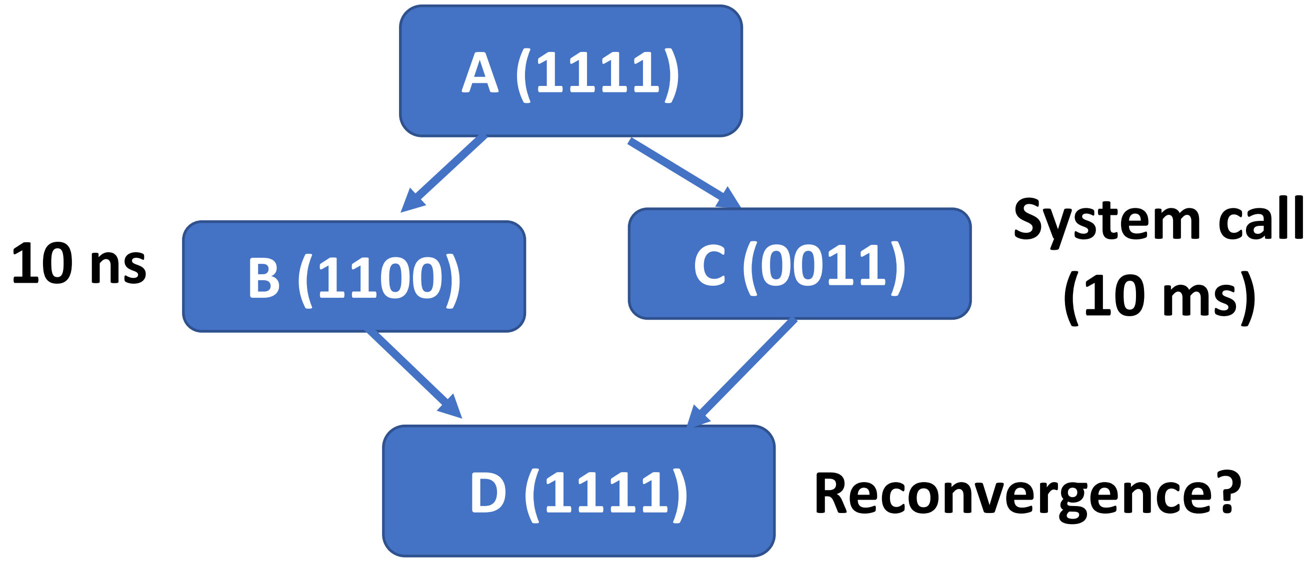


- Memory Divergence
  - Cache Contention & Bank Conflicts

- Larger execution units & cache resources at the backend
  - Higher instruction execution & L1 hit latency

Deep Dive into RPU's Challenges

- Control Divergence
  - Control divergence wit high latency branch



- Memory Divergence
  - Cache Contention & Bank Conflicts

- Larger execution units & cache resources at the backend
  - Higher instruction execution & L1 hit latency

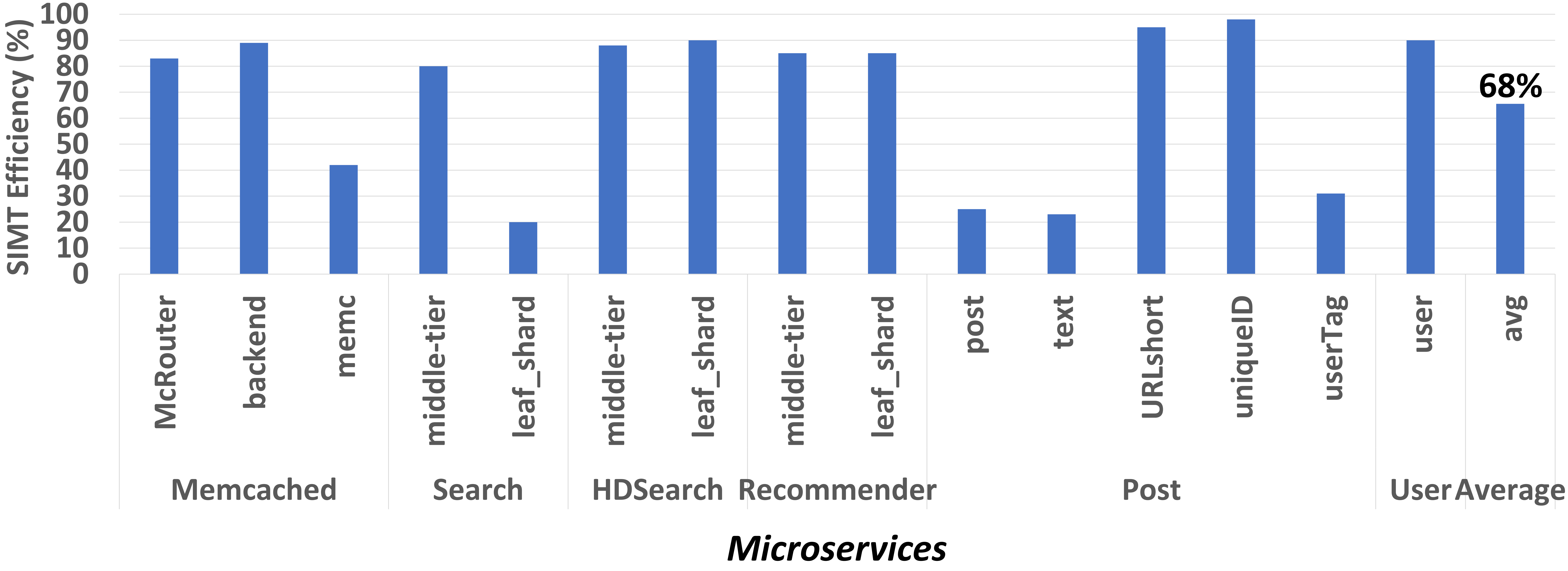
HW/SW Stack

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

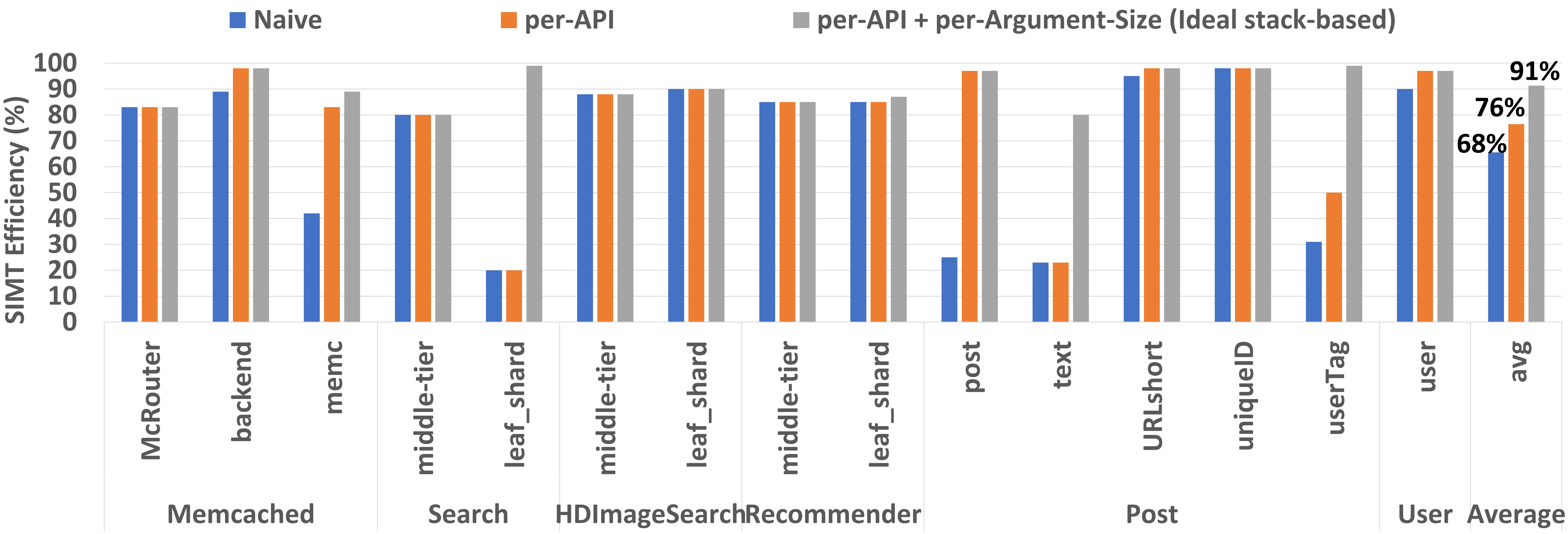
→ Some VM&process management system calls are reimplemented in the RPU driver to be batch-aware

SIMT Control Efficiency

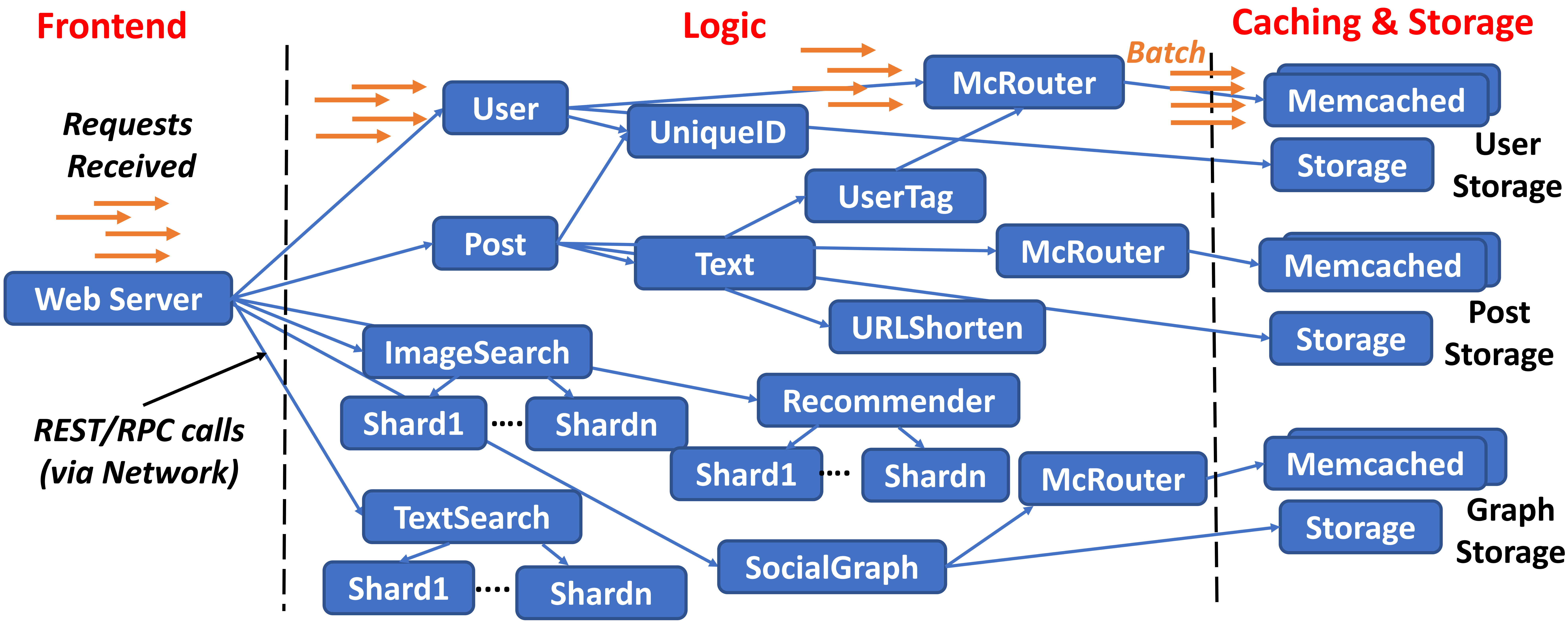


Notes: (1) Batch Size = 32, (2) System Calls are not included, (3) SIMT Eff = scalar-instructions / (batch-instructions * batch-size), (4) fine-grain locking are assumed

SIMT Control Efficiency (Optimized)

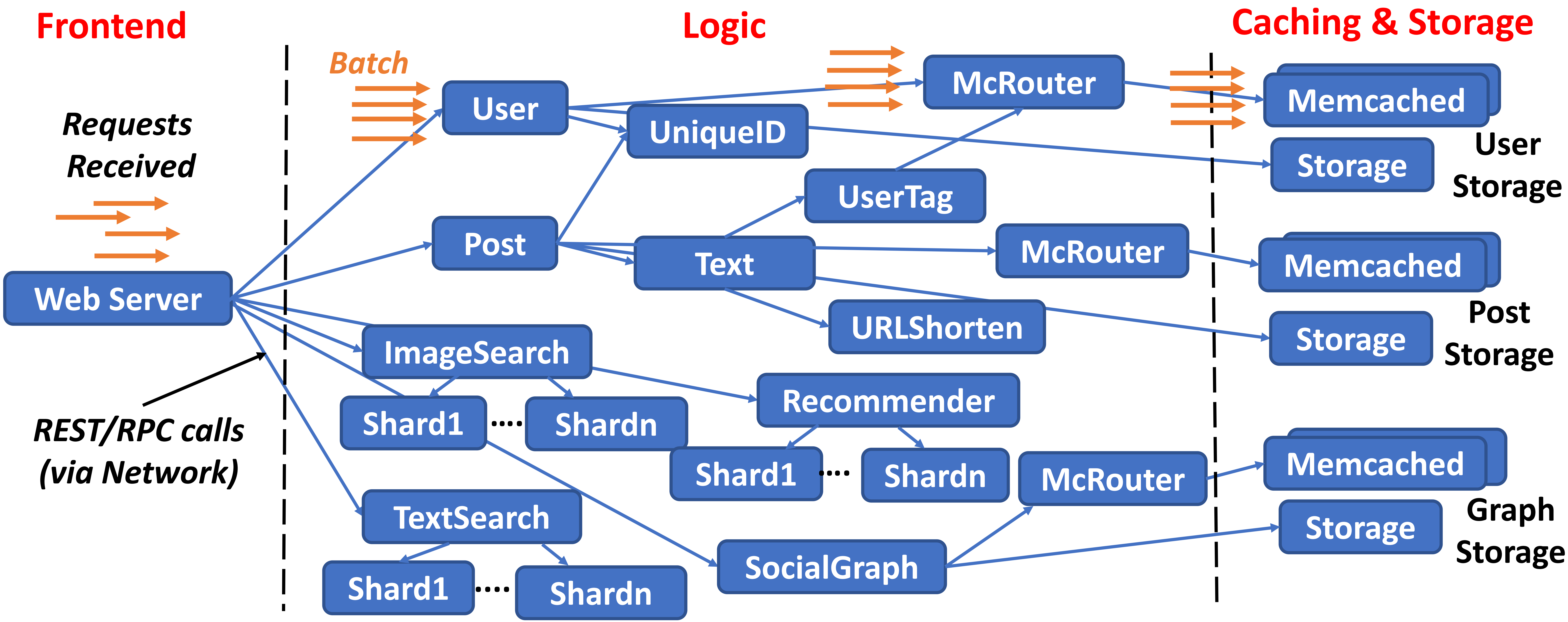


Current System: Selective Batching



Key Observation: Batching is heavily employed in the data center (DL inference, Memcached, ..)

SIMR: System-Level Batching

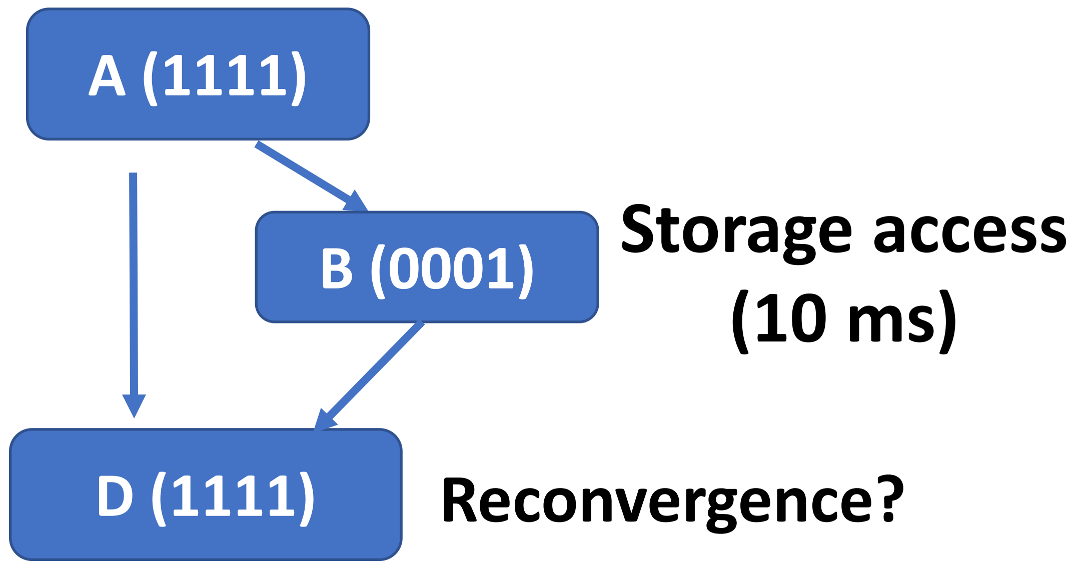


Key Observation: Batching is heavily employed in the data center (DL inference, Memcached, ..)

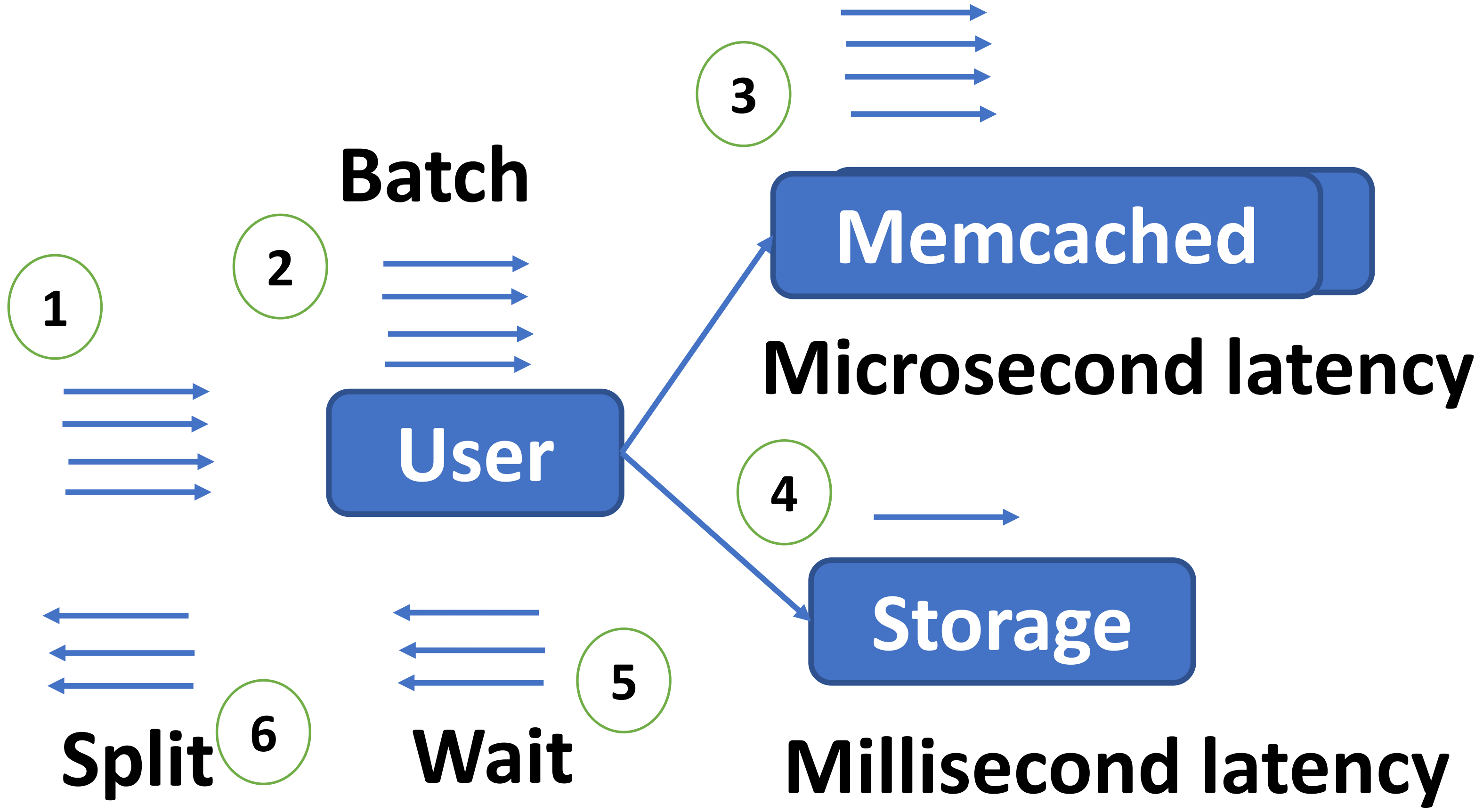
→ Instead of batching individual microservices, we propose batching in all microservices in the graph

System-Level Batch Splitting

```
Procedure get_user(int userid)
   /* first try the cache */
   data = memcached_fetch("userrow:" + userid)
   if not data       /* SIMT Divergence*/
       /* not found : request database */
       data = db_select("SELECT * FROM users
        WHERE userid = ?", userid)
       /* then store in cache until next get */
       memcached_add("userrow:" + userid, data)
   end       /* SIMT Reconvergence Point*/
   return data
```



Control Flow with Active Mask



HW/SW Stack

Webservice (C++, PHP, …)
ARM/x86 compiler
HTTP server
Runtime/libs (pthread, cstdlib, ..)
OS (Process, VM, I/Os)
Multi Core CPU

CPU SW Stack

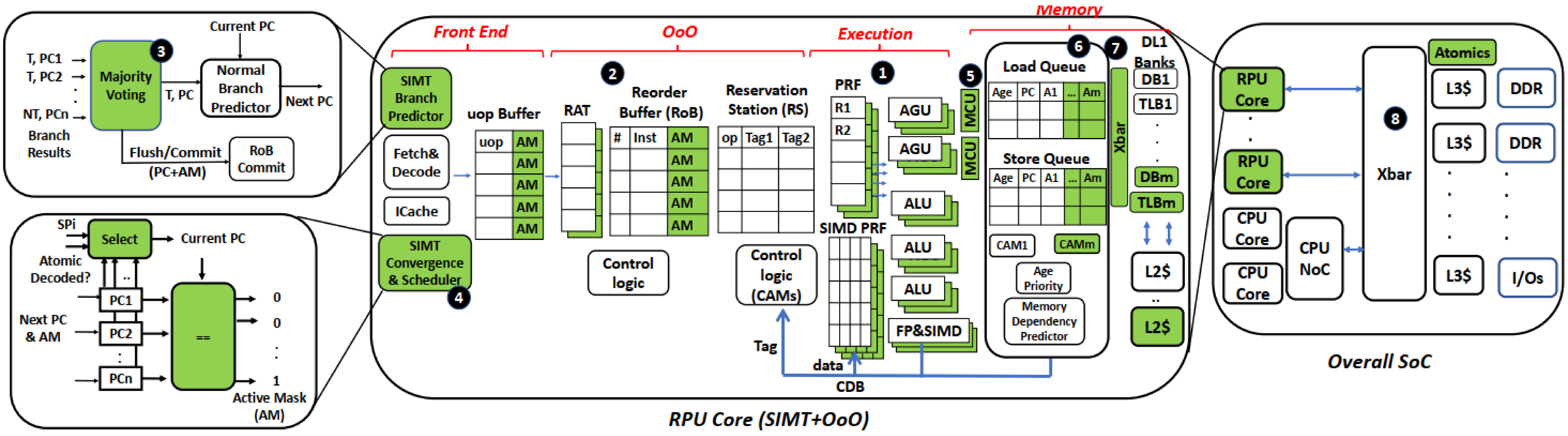
CUDA
CUDA compiler
Nvidia Triton HTTP server
CUDA runtime/libs (cudalib, tensorRT, ..)
OS (I/Os management)
CUDA driver (VM/thread management)
GPU Hardware

GPU SW Stack

Webservice (C++, PHP, …)
ARM/x86 compiler
Batch-aware HTTP server
Runtime/libs (pthread, cstdlib, ..)
OS (I/Os management)
RPU driver (VM/thread management)
RPU Hardware

RPU SW Stack

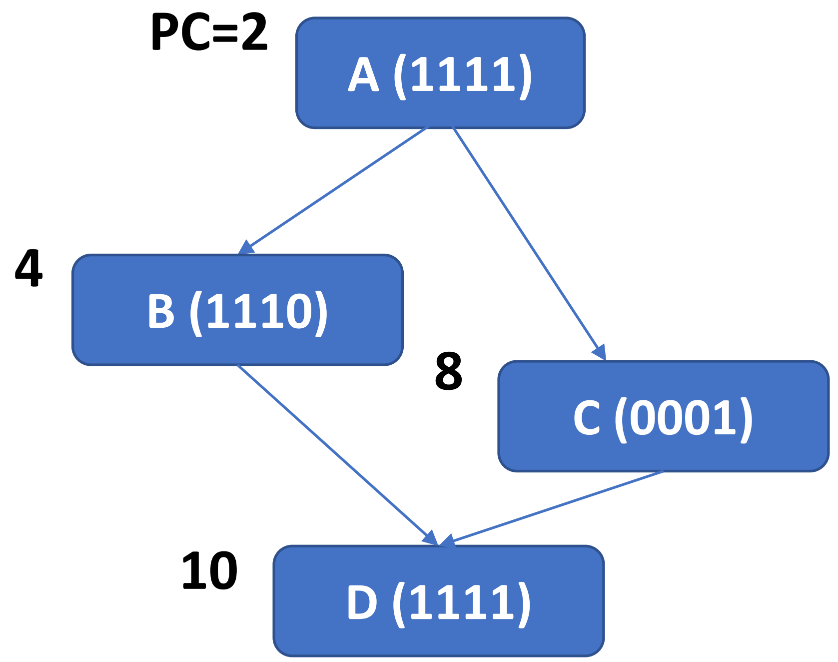
RPU HW


Current PC
T, PC1
T, PC2
NT, PCn
Majority Voting
T, PC
Normal Branch Predictor
Next PC
Branch Results
Flush/Commit (PC+AM)
RoB Commit
SPi
Select
Current PC
Atomic Decoded?
PC1
PC2
PCn
Next PC & AM
==
0
0
1
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

Control Divergence Handling

```
1. / / BBA Basic Block "A"
2. if ( x > 0)
3. {
4.     / / BBB
5. }
6. else
7.  {
8.     / / BBC
9.  }
10. / / BBD
```

Divergent code example



Control Flow with Active Mask

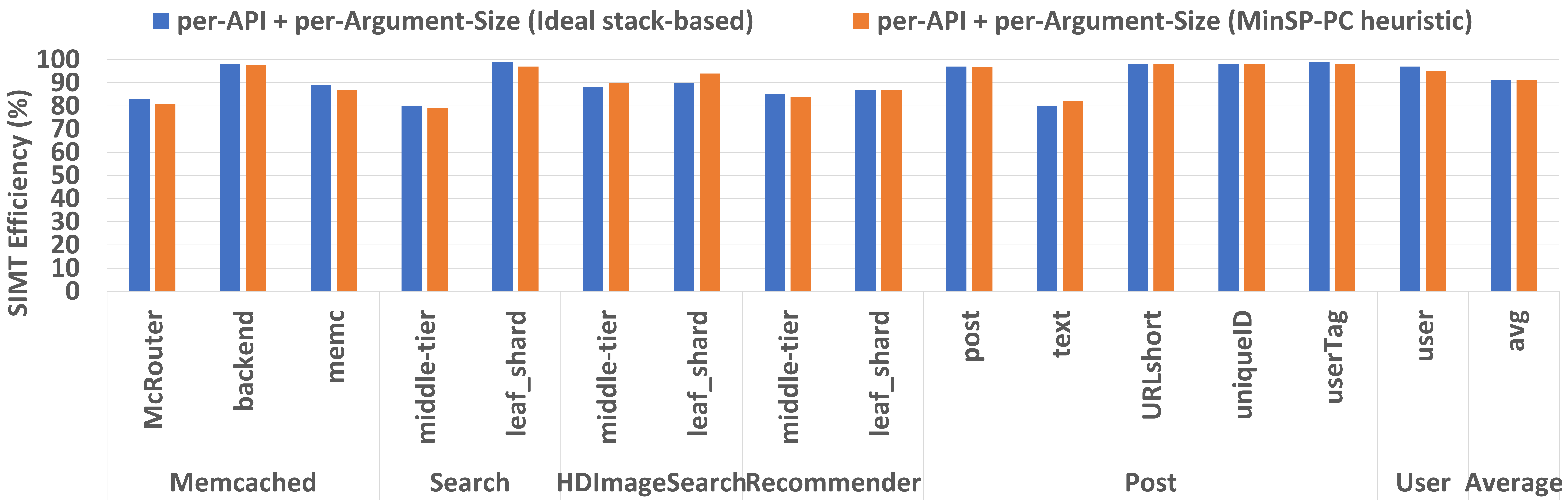
PC1	PC2	PC3	PC4	Current PC (min)	Active mask	Next PC (BP)
2	2	2	2	2	1111	4
4	4	4	4(F)	4	1111	6
6	6	6	8	6	1110	10
10	10	10	8	8	0001	10
10	10	10	10	10	1111	12

MinPC selection policy

Serialize divergent paths

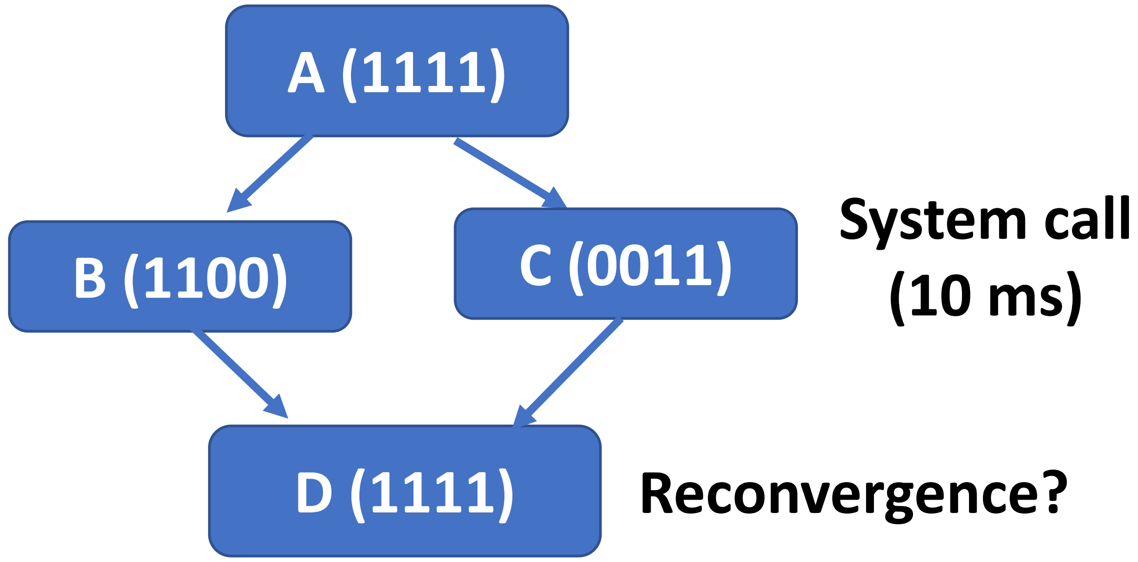
Heuristic-based reconvergence analysis (MinPC policy) – transparent to ISA and compiler

MinSP-PC Heuristic



Deep Dive into RPU's Challenges

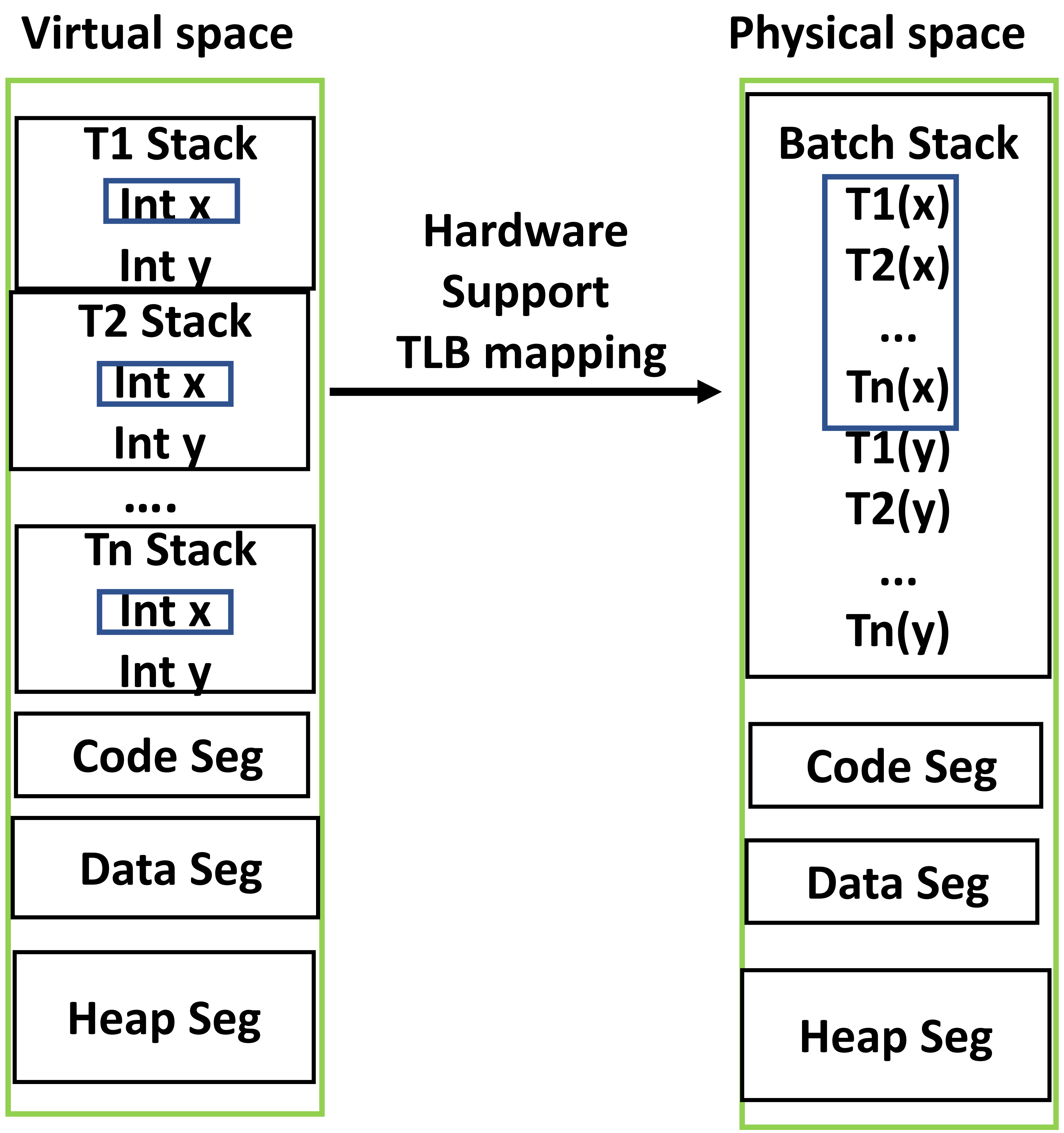
- Control Divergence
  - Control divergence wit high latency branch



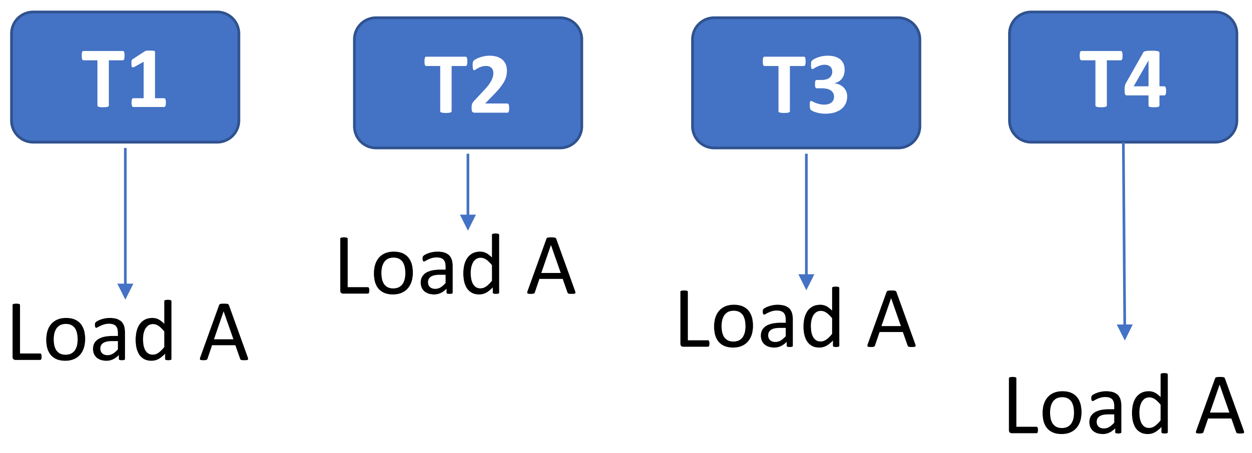
- Memory Divergence
  - Cache Contention & Bank Conflicts

- Larger execution units & cache resources at the backend
  - Higher instruction execution & L1 hit latency

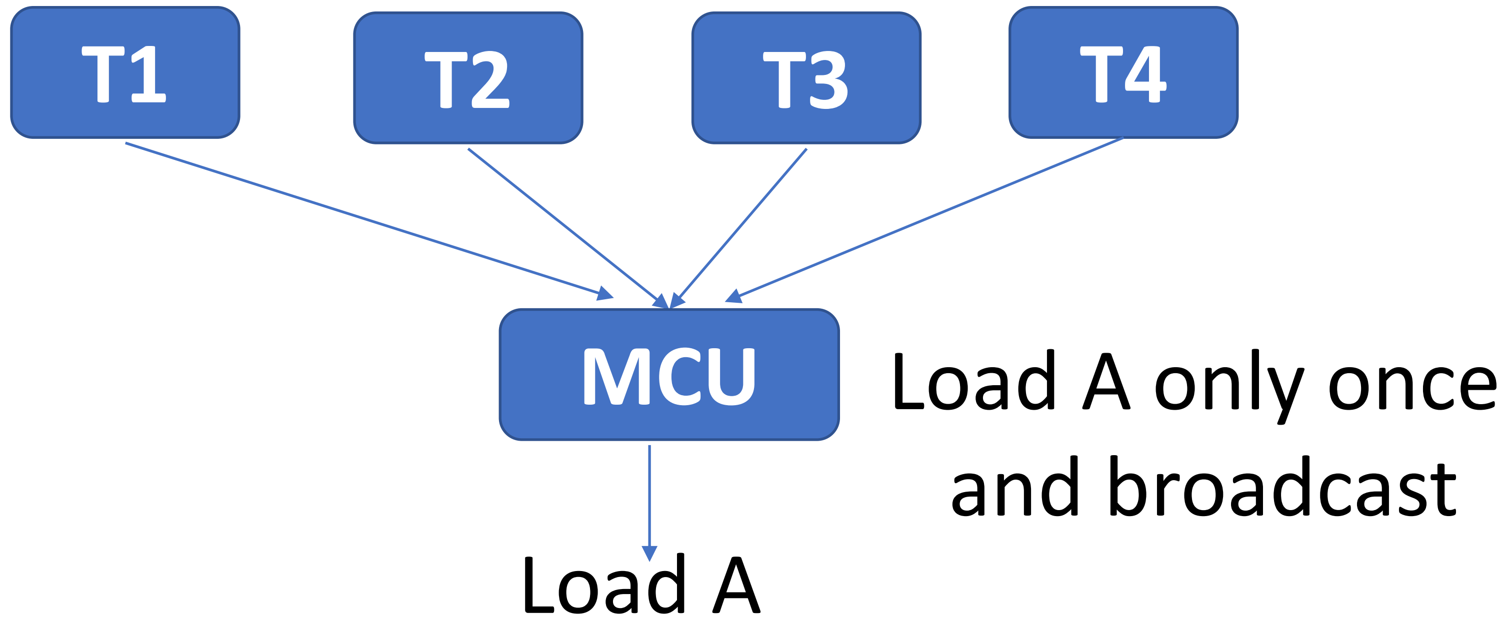
Memory Coalescing Optimizations



Stack segment coalescing with data interleaving



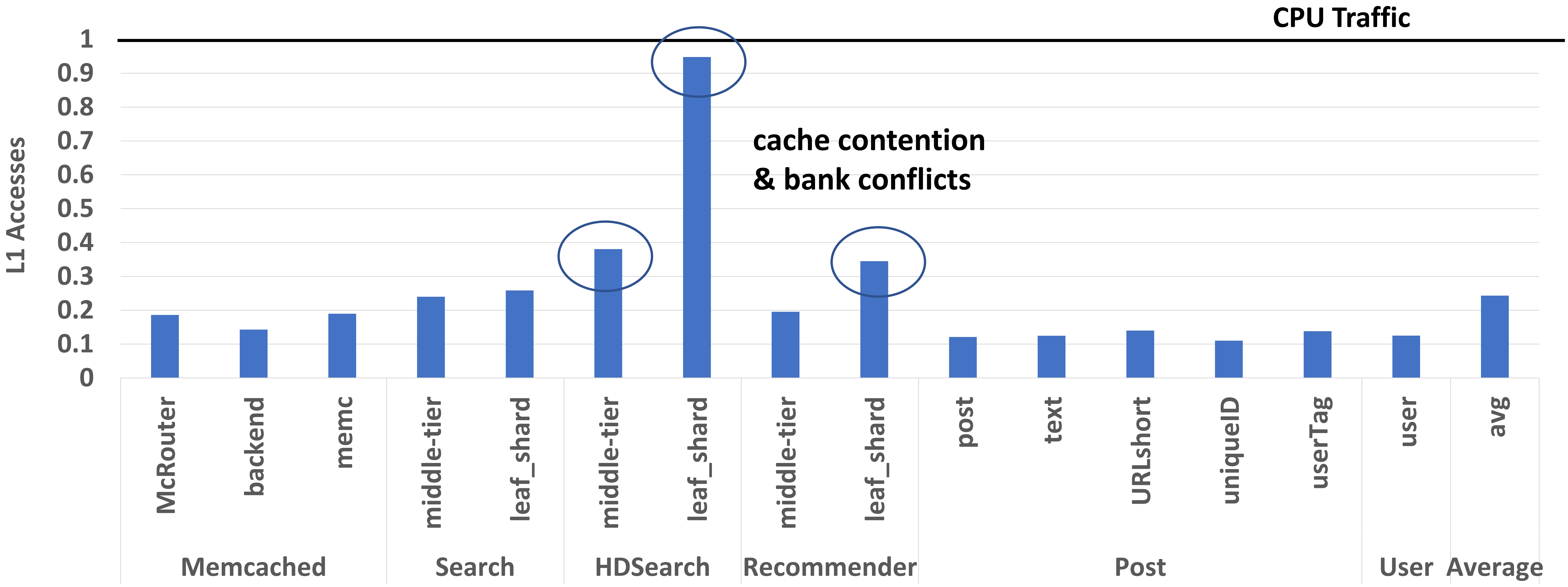
Independent threads execution (CPU)



SIMT execution with MCU

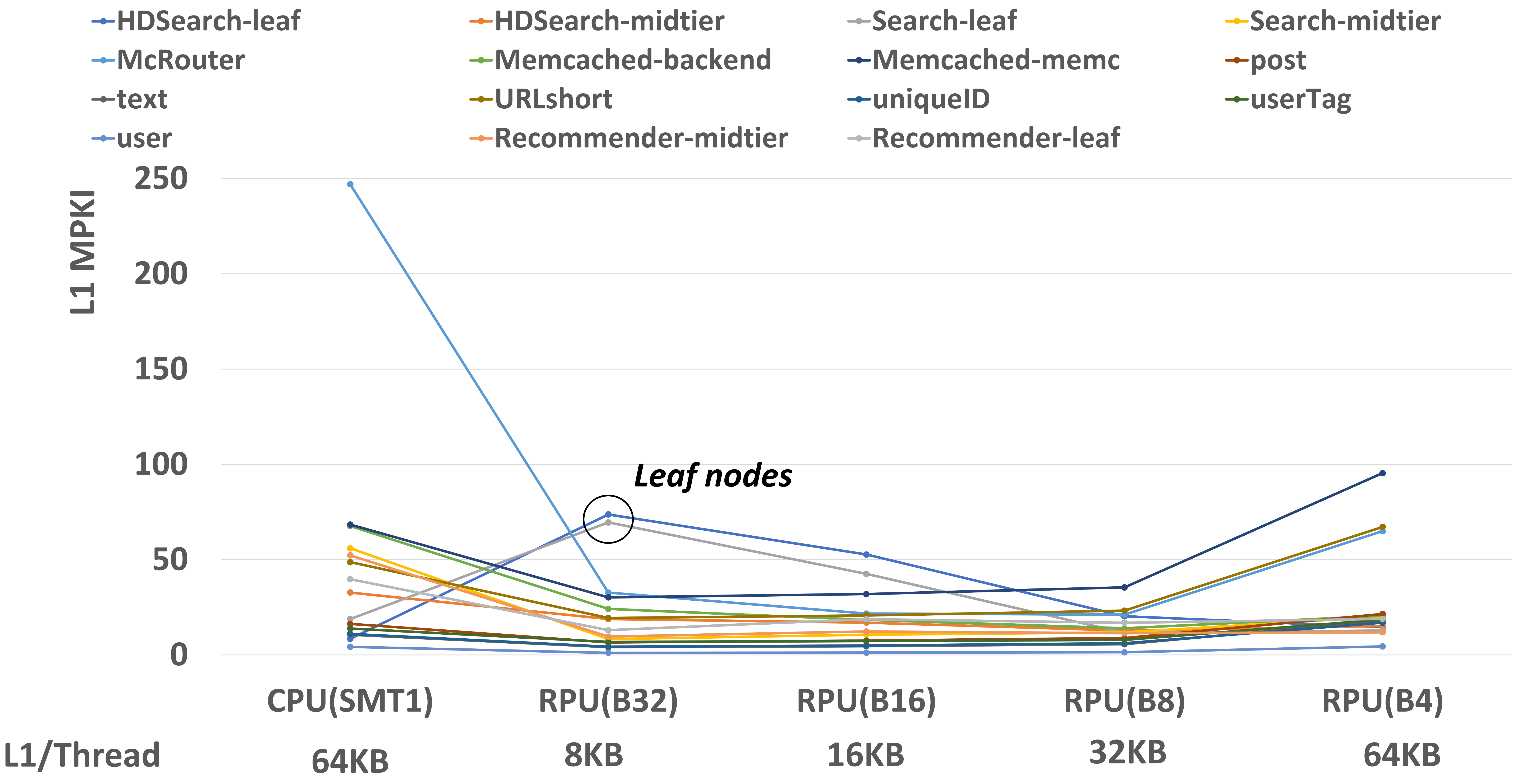
HW memory coalescing unit (MCU) for Heap & Data segments

Traffic Reduction

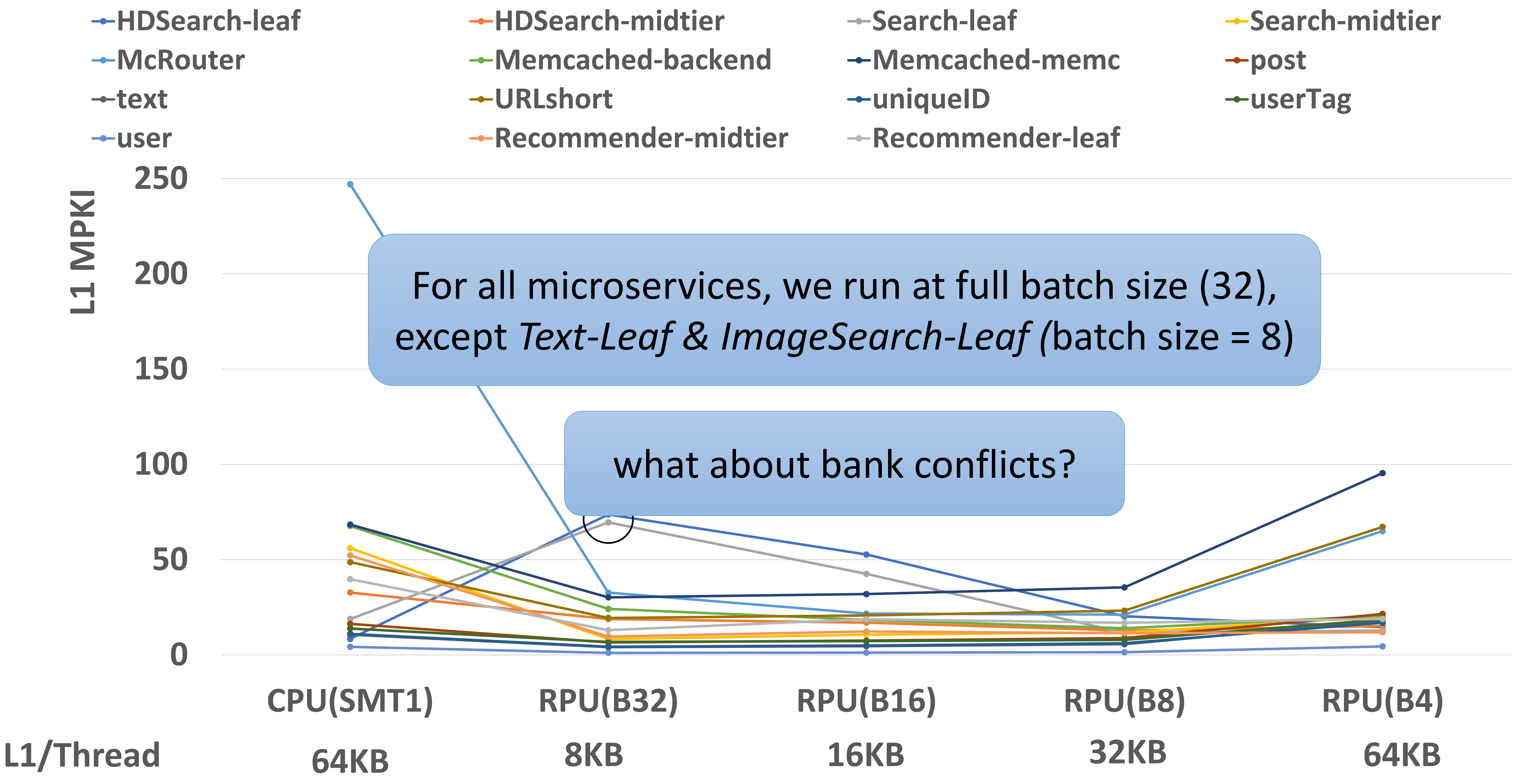


→ **4x traffic reduction compared to CPU**

Batch Size Tuning to Alleviate Cache Contention



Batch Size Tuning to Alleviate Cache Contention



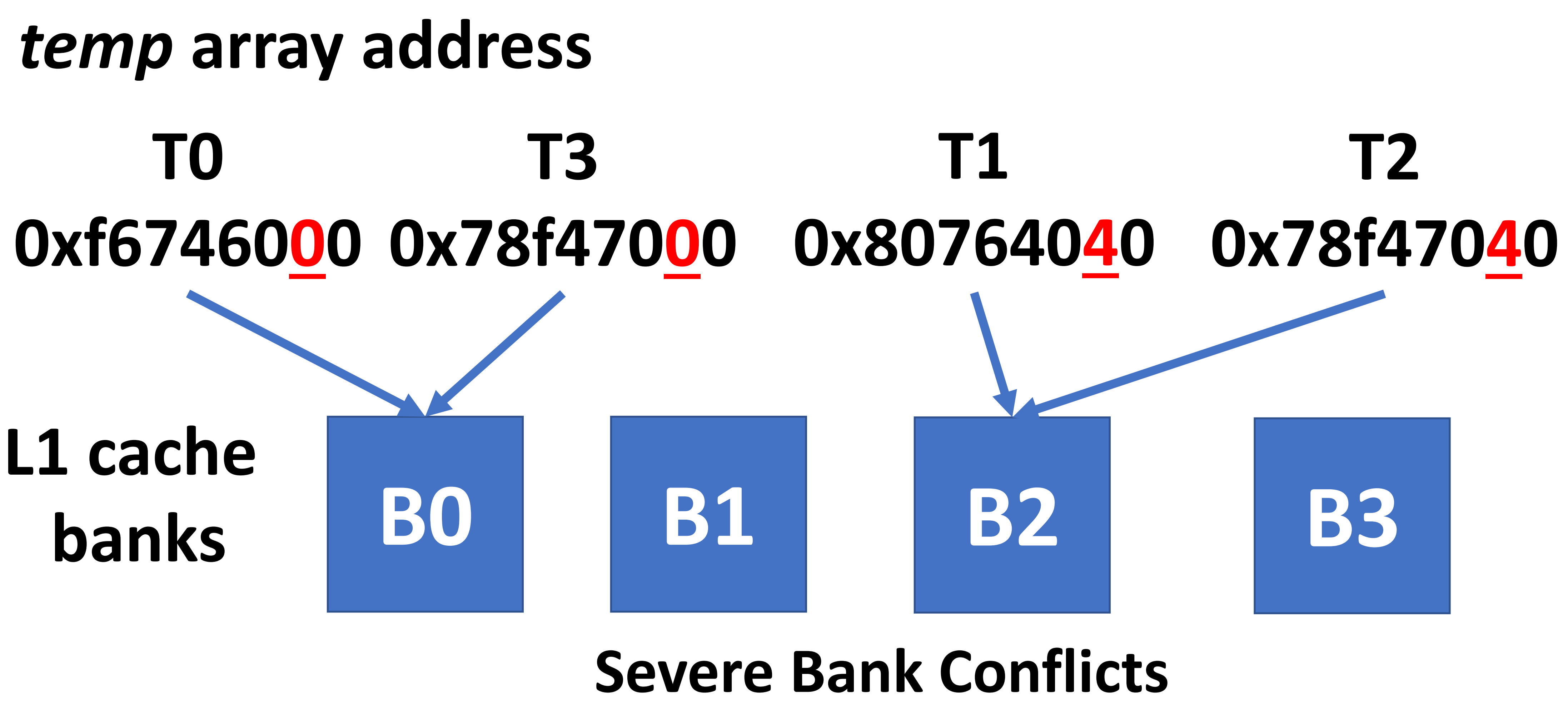
For all microservices, we run at full batch size (32), except *Text-Leaf & ImageSearch-Leaf* (batch size = 8)

what about bank conflicts?

SIMT-Agnostic Memory Allocator

```
Microservice ()
  //Create a private temporary array in the
  // heap segment
  int* temp = new int[n];
   ………..
   for(int i=0; i<n; i++)
      temp[i] = i;     //Write to the temp
   ………..
for(int i=0; i<n; i++)
     sum += temp[i];  //Read from the temp
 ………..
```

Assume data are interleaved every 32B

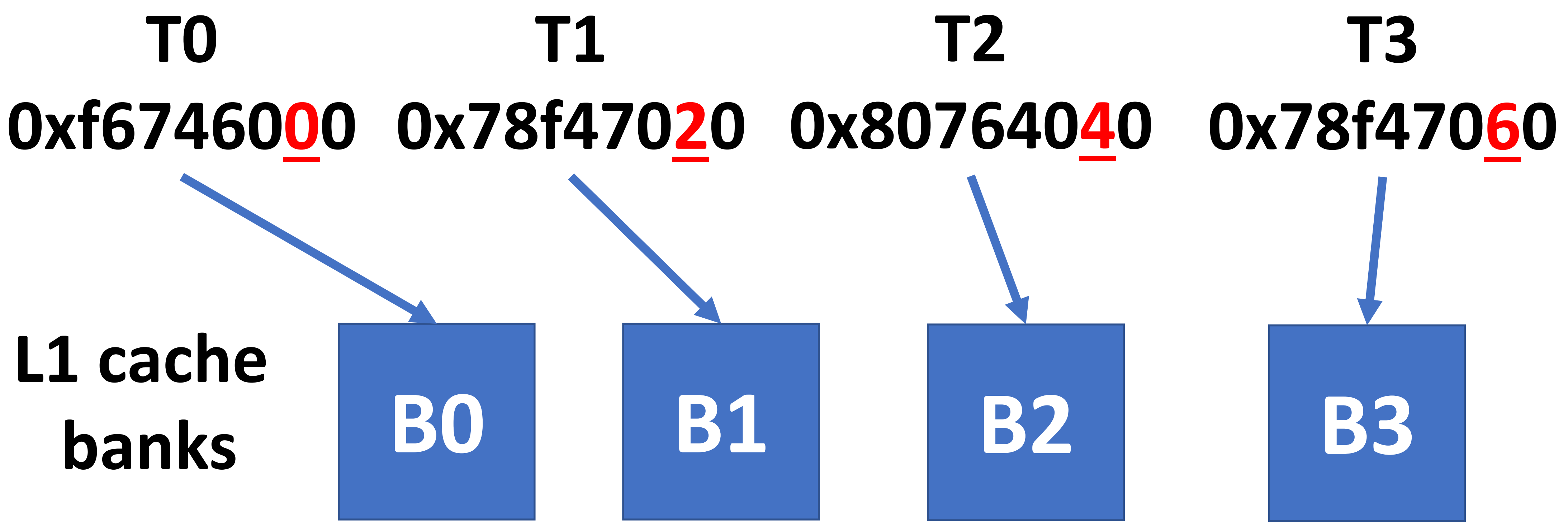


C++ SIMT-Agnostic Memory Allocator

SIMT-Aware Memory Allocator

```
Microservice ()
  //Create a private temporary array in the
  // heap segment
  int* temp = new int[n];
   ..........
   for(int i=0; i<n; i++)
      temp[i] = i;      //Write to the temp
   ..........
for(int i=0; i<n; i++)
      sum += temp[i];  //Read from the temp
 ..........
```

Assume data are interleaved every 32B



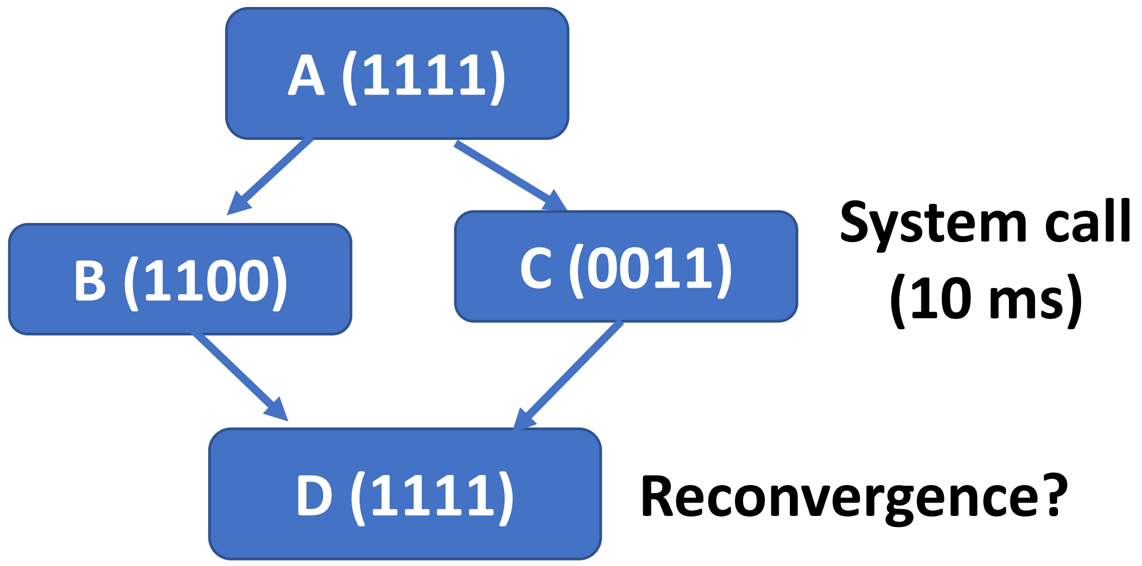
Conflict-free

C++ SIMT-Aware Memory Allocator

→ ensures start_address%(n*tid) = 0

Deep Dive into RPU's Challenges

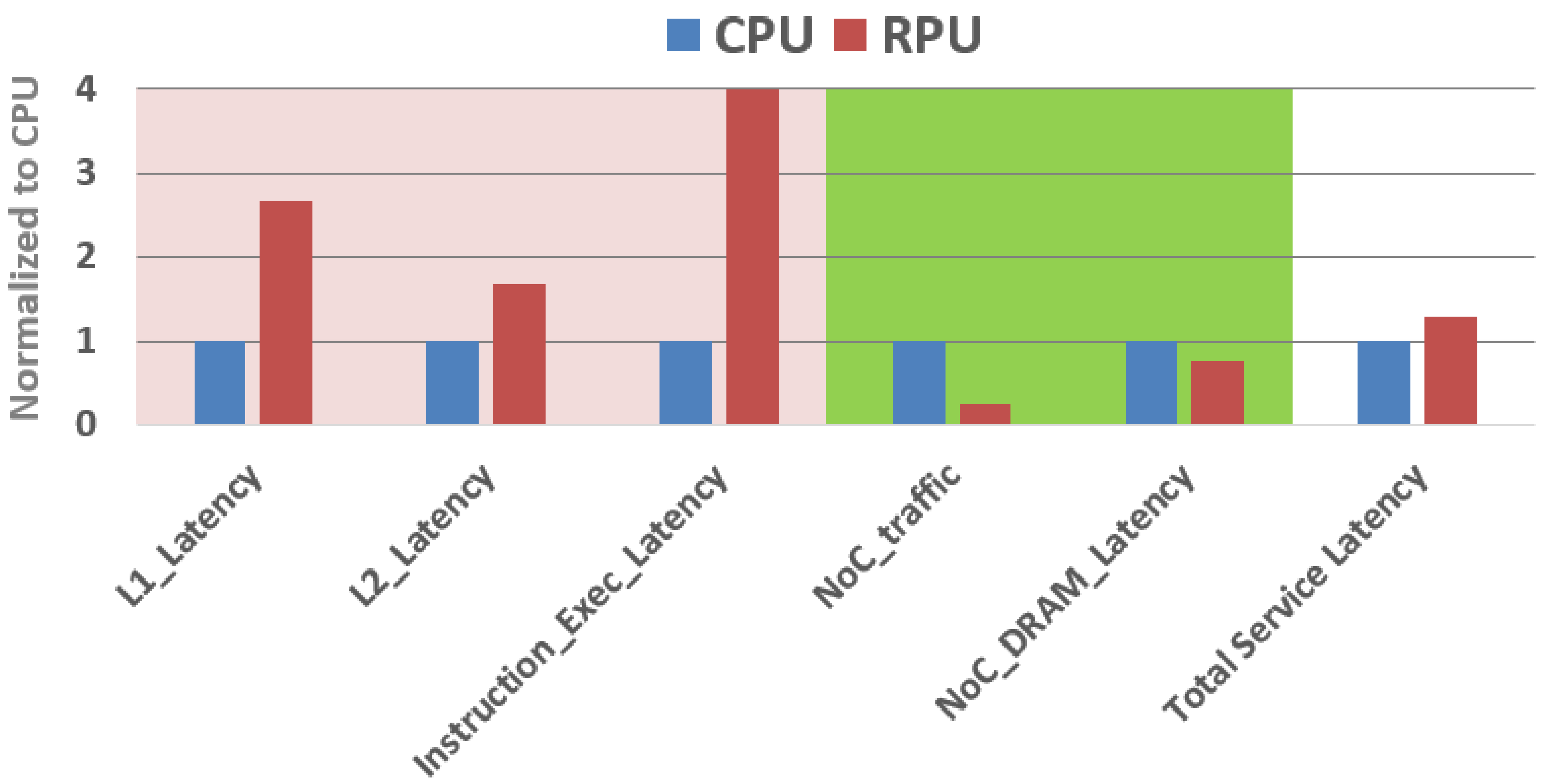
- Control Divergence
  - Control divergence wit high latency branch



- Memory Divergence
  - Cache Contention & Bank Conflicts

- Higher instruction execution & L1 hit latency
  - More execution units & cache resources at the backend

Memory Latency Improvement



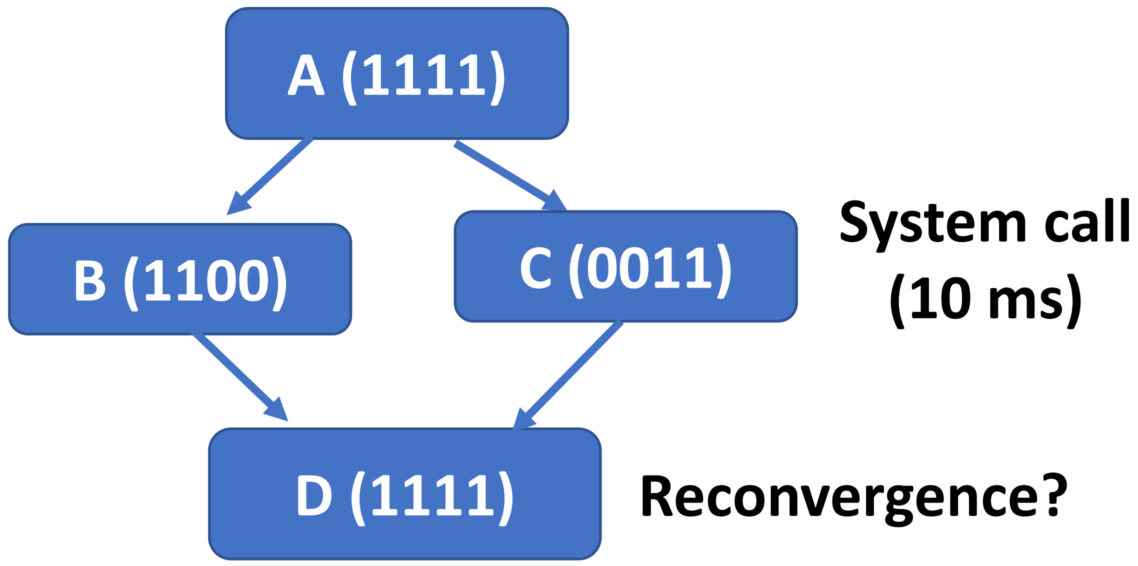
Metrics that contribute to total service latency

→ Memory Latency improvement (due to less traffic and crossbar) helps to offset the latency increases in instructions and cache hits

<u>Recall</u>: data center workloads exhibit a limited IPC and retire rate as they are bounded by memory latency

Deep Dive into RPU's Challenges

- Control Divergence
  - Control divergence wit high latency branch



- Memory Divergence
  - Cache Contention & Bank Conflicts

- Higher instruction execution & L1 hit latency
  - More execution units & cache resources at the backend

Evaluation

- Analytical Model
- Simulation-based evaluation
  - Chip-level evaluation
  - System-level evaluation

Energy Efficiency of CPU vs RPU (Analytical Model)

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

batch size (n) = 8-32

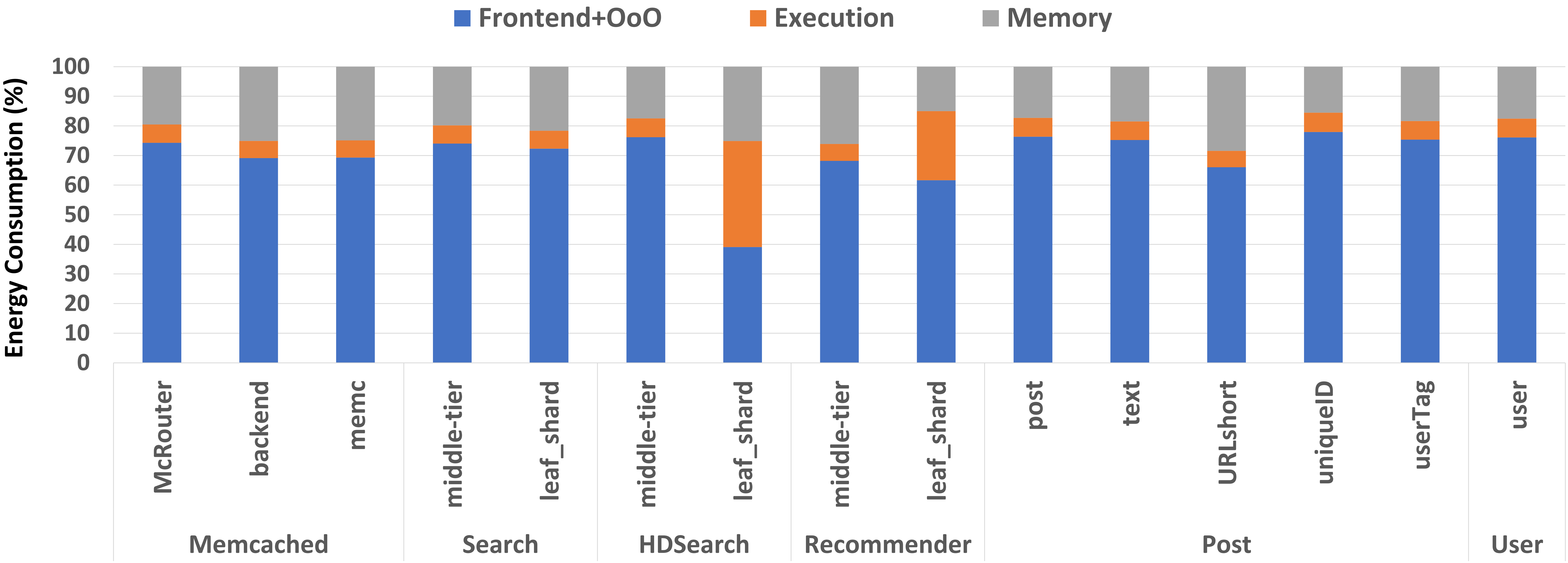
SIMT Efficiency=92%

data locality ratio =75%

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

Dynamic Instrumentation

SIMTec (x86 PIN-based tool) [ISPASS 2022]

traces (w/ & w/o batching)

Chip-level cycle accurate simulator

Accel-Sim [ISCA 2020] & McPAT

Throughput & latency

System-level uservice-interaction simulator

uQsim [ISPASS 2019]

SIMT Efficiency

CPU vs RPU throughput/Watt & latency

End-to-end tail latency & Max throughput

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*

Simulation Configuration

- Baseline: Single threaded CPU and SMT8 CPU
- RPU: SIMT-32 (1 batch)

- We ensure both CPU and RPU have the same pipeline configuration, frequency, and memory resources/thread for SMT8 and our RPU

- CPU & RPU power&area are estimated at the same technology node (7-nm)

Table 4.4. CPU vs RPU Simulated Configuration

Metric	CPU	CPU SMT	RPU
Core Pipeline	8-wide 128-entry OoO	8-wide 128-entry OoO	8-wide 128-entry OoO
Freq	2.5 GHZ	2.5 GHZ	2.5 GHZ
#Cores	98	80	20
Threads/core	1	SMT-8	SIMT-32 (1 batch)
Total Threads	98	640	640
#Lanes	1	1	8
Max IPC/core	8	8	64 (issue x lanes)
ALU/Bra Exec Lat	1-cycle	1-cycle	4-cycle
L1 Inst/core	64KB	64KB	64KB
Reg File/core	2KB	16KB	64KB
L1 Cache	64KB, 8-way, 3 cycles, 1-bank 32B/cycle	64KB, 8-way, 3 cycles, 8-banks 256BB/cycle	256KB, 8-way, 8 cycles, 8-banks 256B/cycle
L2 Cache	512KB, 8-way, 12 cycles, 1-bank	512KB, 8-way, 12-cycles, 2-banks	2MB, 8-way, 20 cycles, 2-banks
DRAM	8x DDR5-3200, 200 GB/sec	10x DDR5-7200, 576 GB/sec	10x DDR5-7200, 576 GB/sec
Interconnect	9x9 Mesh	11x11 Mesh	40x40 Crossbar
OoO entries/thread	128, 8-wide	16, 1-wide	128, 8-wide
L1 capacity/thread	64KB	8KB	8KB
L1B/cycle/thread	32B/cycle	32B/cycle	8B/cycle
memBW/thread	2 GB/sec	0.9 GB/sec	0.9 GB/sec

Per-component Area and Peak Power Estimates

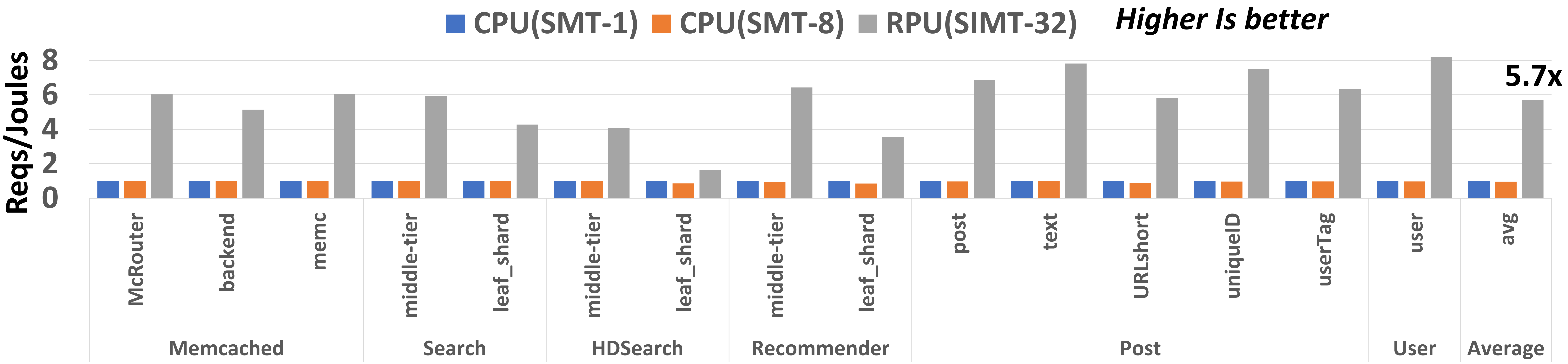
- RPU core is 6.3x larger and consumes 4.5x more peak power than the CPU core; however, the RPU core supports 32x more threads

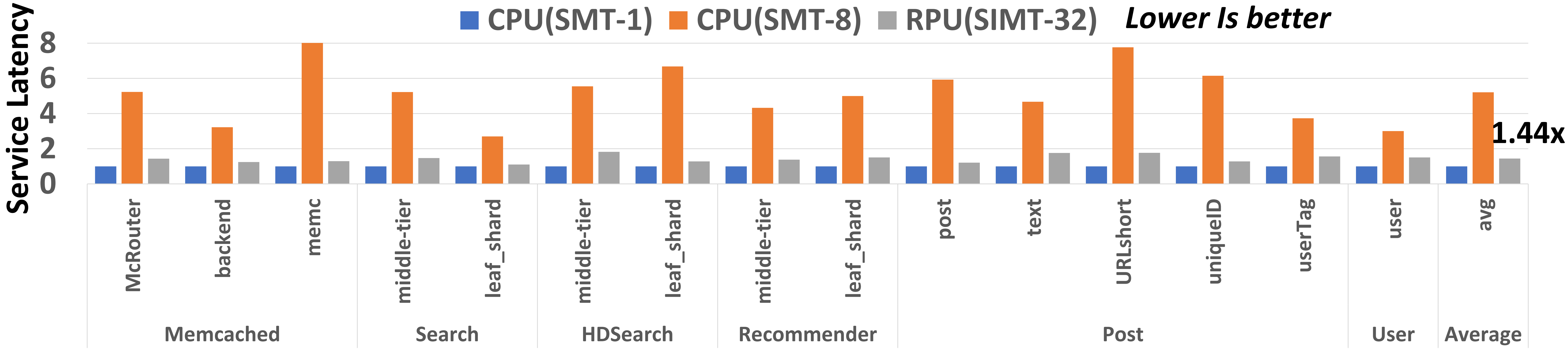
- The additional overhead of the RPU-only structures consume 11.8% of the RPU core.

Table V: Per-component area and peak power estimates

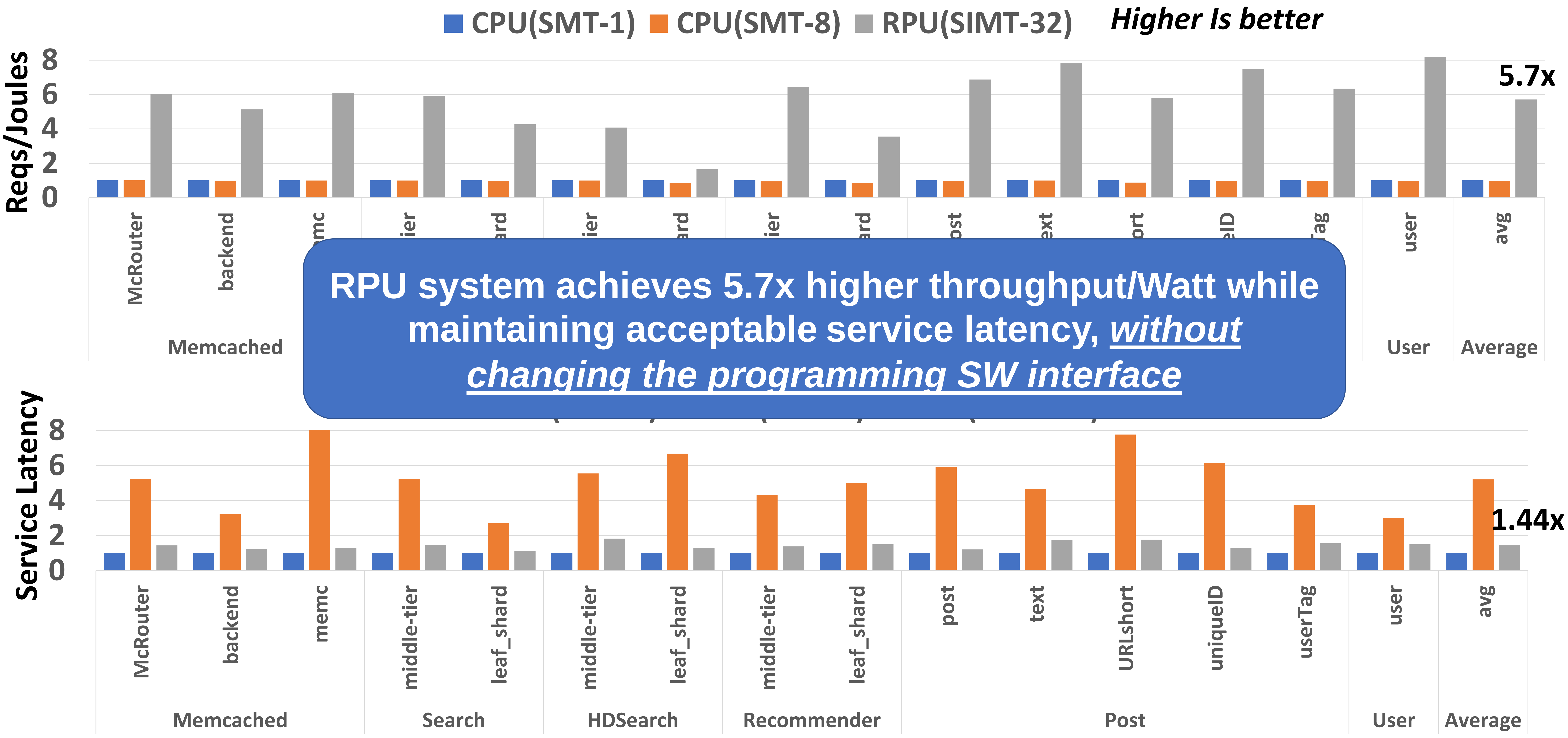
Component	Area				Peak Power			
	CPU		RPU		CPU		RPU	
	mm^2	% Core	mm^2	% Core	Watt	% Core	Watt	% Core
Fetch&Decode	0.27	24.3	0.3	4.3	0.39	15.6	0.4	3.6
Branch Prediction	0.01	0.9	0.01	0.1	0.02	0.8	0.02	0.2
OoO	0.11	9.9	0.17	2.4	0.85	34	1.45	12.9
Register File	0.14	12.6	2.52	35.8	0.49	19.6	4.26	38
Execution Units	0.25	22.5	2.31	32.8	0.34	13.6	2.51	22.4
Load/Store Unit	0.07	6.3	0.34	4.8	0.13	5.2	0.41	3.7
L1 Cache	0.04	3.6	0.22	3.1	0.09	3.6	0.2	1.8
TLB	0.02	1.8	0.08	1.1	0.06	2.4	0.4	3.6
L2 Cache	0.2	18	0.71	10.1	0.13	5.2	0.24	2.1
Majority Voting	0	0	0.02	0.3	0	0	0.03	0.3
SIMT Optimizer	0	0	0.03	0.4	0	0	0.05	0.4
MCU	0	0	0.02	0.3	0	0	0.01	0.1
L1-Xbar	0	0	0.31	4.4	0	0	1.23	11
Total-1core	1.11		7.04		2.5		11.21	
	mm^2	% Chip	mm^2	% Chip	Watt	% Chip	Watt	% Chip
Total-Allcores	108.8	77.2	140.8	81	245	72.5	224.2	73.7
L3 Cache	7.82	5.5	7.82	4.5	0.75	0.2	0.75	0.2
NoC	9.78	6.9	1.72	1	36.52	10.8	7.02	2.3
Memory Ctrl	14.64	10.4	23.59	13.6	6.85	2	19.27	6.3
Static Power					49	14.5	53	17.4
Total Chip	141		173.9		338.1		304.2	

Efficiency and Service Latency Results (Simulation)



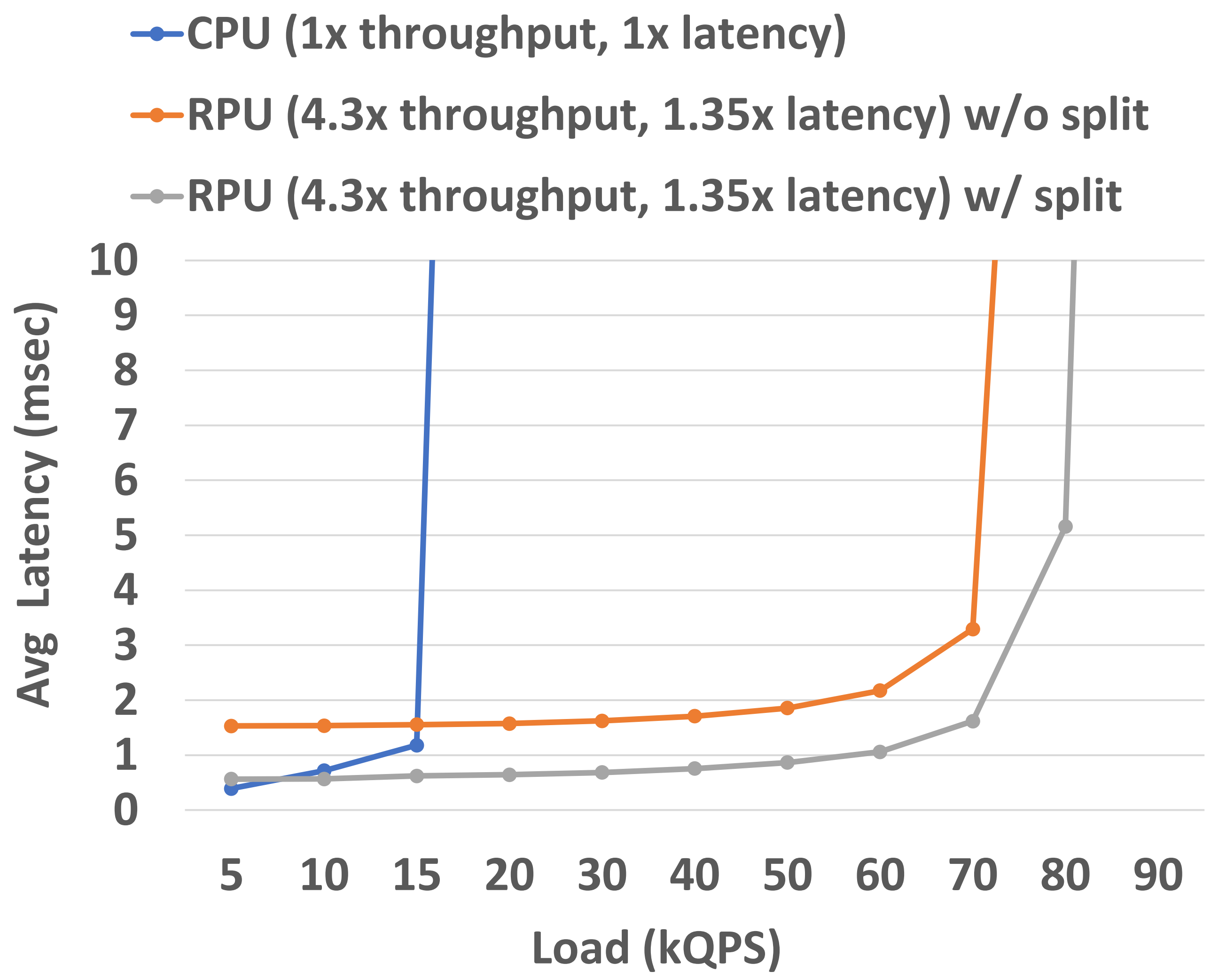


Efficiency and Service Latency Results (Simulation)

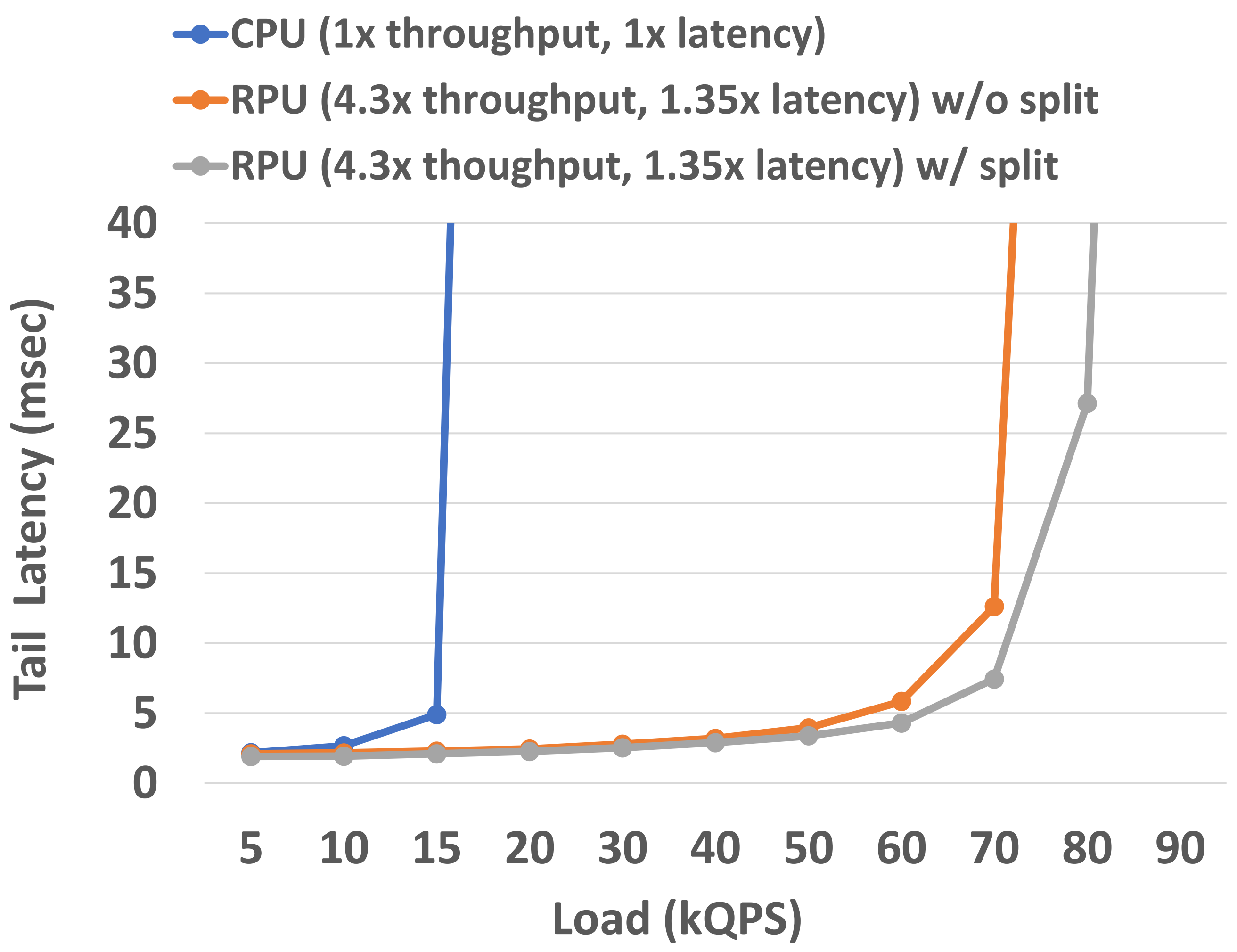


RPU system achieves 5.7x higher throughput/Watt while maintaining acceptable service latency, *__without changing the programming SW interface__*

System-Level Results (uQsim Simulator)



Average latency



99% tail latency

→ RUP's batching overhead is amortized at low and high loads

→ Batch split technique achieves almost <u>the same</u> average and tail latency as CPU system at <u>4x higher</u> throughput

→ Without the batch split technique, we are still able to get a good tail latency

Notes: assume 90% hit rate of Memcached, storage latency = 1 ms & network latency = 60 nsec

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

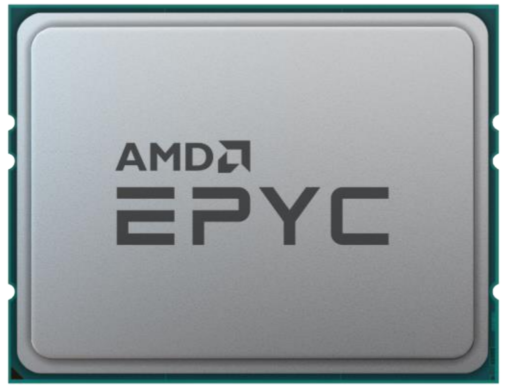
We are very interested in evaluating SIMT control efficiency in proprietary production microservices.

Google

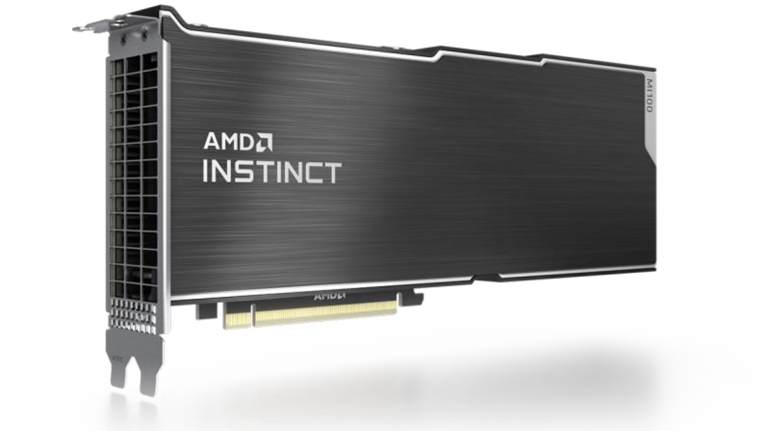
facebook

Thank You!
Q&A?

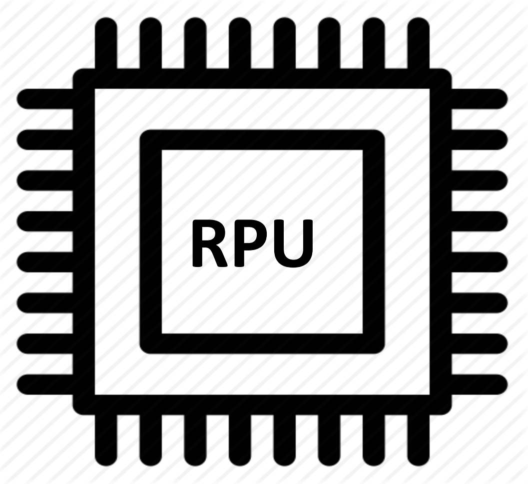
Instruction level parallelism (ILP) &
Thread level parallelism (TLP)



Data level parallelism (DLP)



Request level parallelism (RLP)

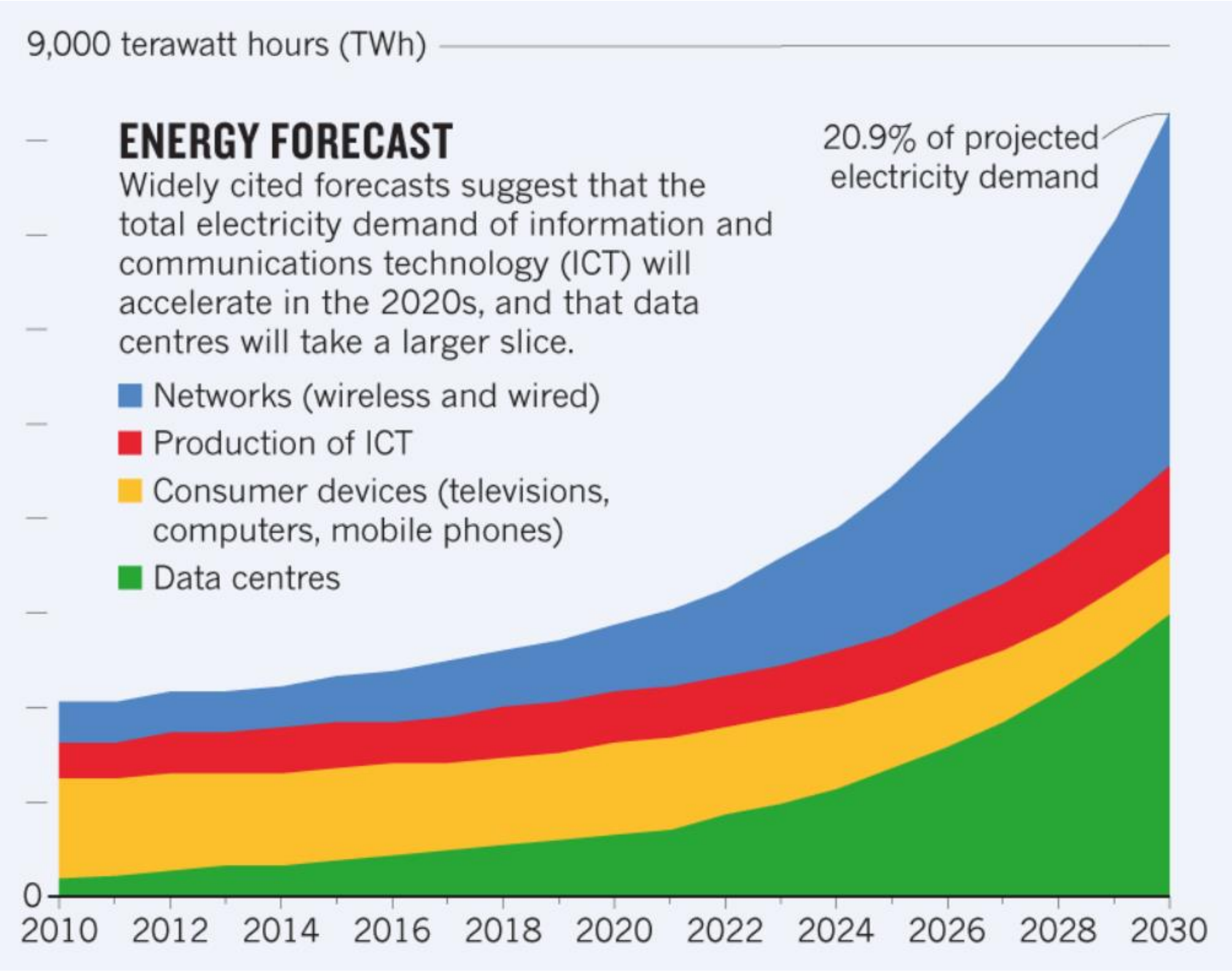


Back-Up Slides

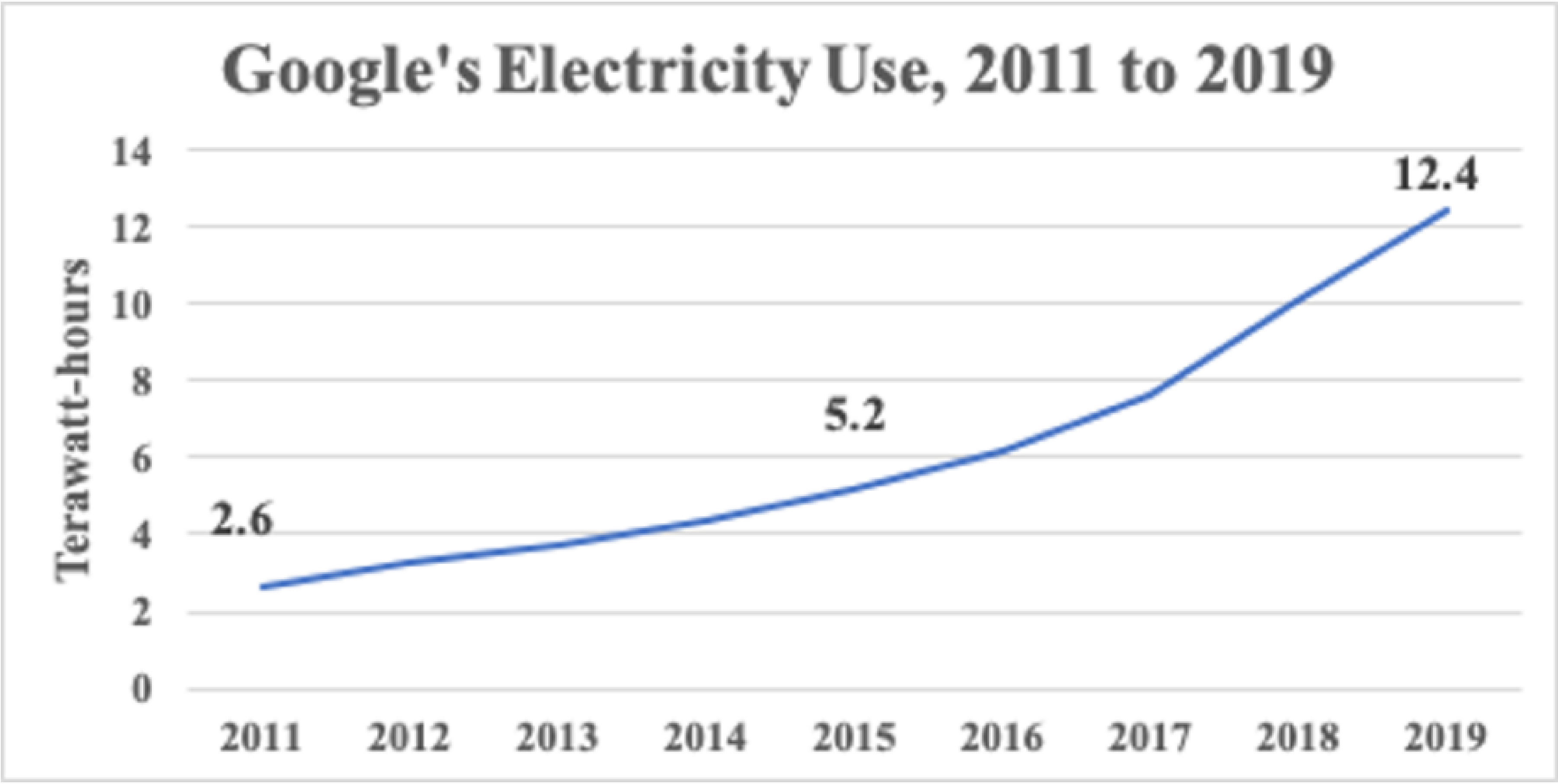
Motivation & Background Slides

Energy Efficiency Crisis

- By 2030, the data centers will consume 9% of the total electricity demand



https://www.nature.com/articles/d41586-018-06610-y



https://robertbryce.com/googles-dominance-fueled-by-zambia-size-amounts-of-electricity/

More Moore!

Google Building More Data Centers for Massive Future Clouds

BY RICH MILLER - DECEMBER 3, 2019 — 1 COMMENT

A Massive Chip Shortage Is Hitting the Entire Semiconductor Industry

By Joel Hruska on December 21, 2020 at 11:15 am | Comments

Moore's Law Is Dead. Now What?

Shrinking transistors have powered 50 years of advances in computing—but now other ways must be found to make computers more capable.

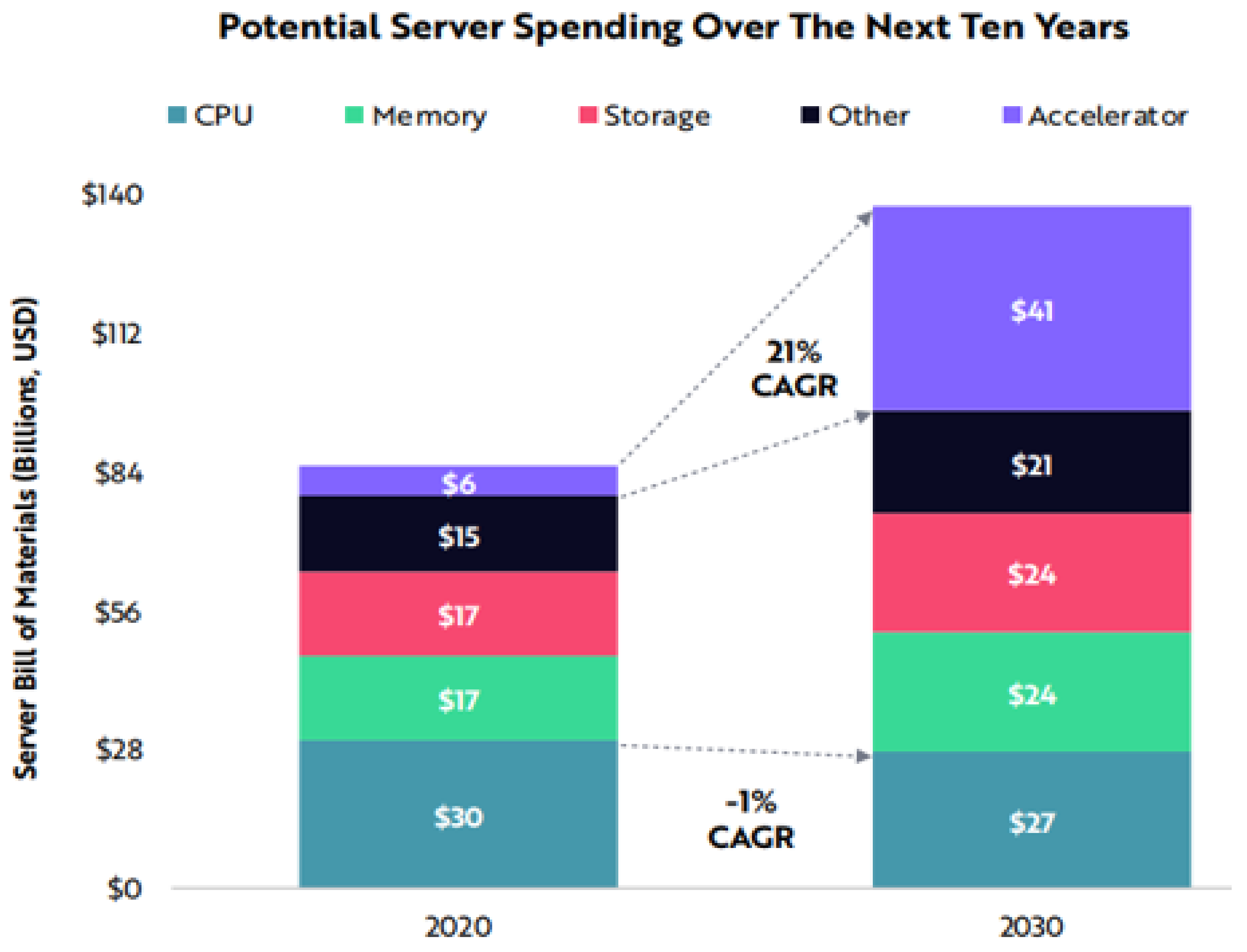
The Chip Shortage Keeps Getting Worse. Why Can't We Just Make More?

By Ian King, Adrian Leung and Demetrios Pogkas

Why data centres are the new frontier in the fight against climate change

https://www.extremetech.com/computing/318554-a-massive-chip-shortage-is-hitting-the-entire-semiconductor-industry
https://www.bloomberg.com/graphics/2021-chip-production-why-hard-to-make-semiconductors/
https://www.marketwatch.com/story/the-semiconductor-shortage-is-here-to-stay-but-it-will-affect-chip-companies-differently-11618678056
https://www.zdnet.com/article/the-global-chip-shortage-is-a-bigger-problem-than-everyone-realised-and-it-will-go-on-for-longer-too/
https://arstechnica.com/cars/2021/05/chip-shortage-continues-us-asks-taiwan-to-prioritize-automakers/

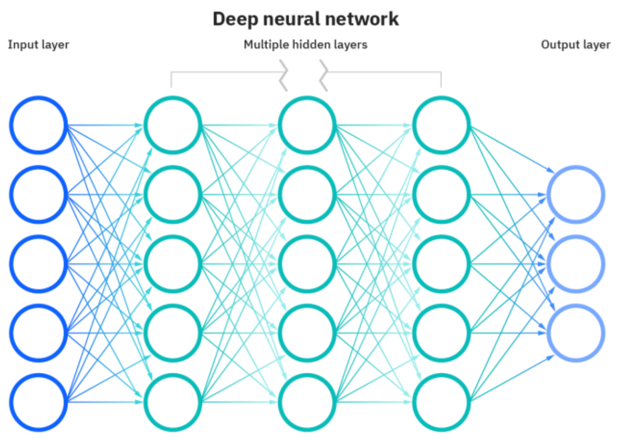
Solution: Hardware/Software Co-Design (Accelerators)



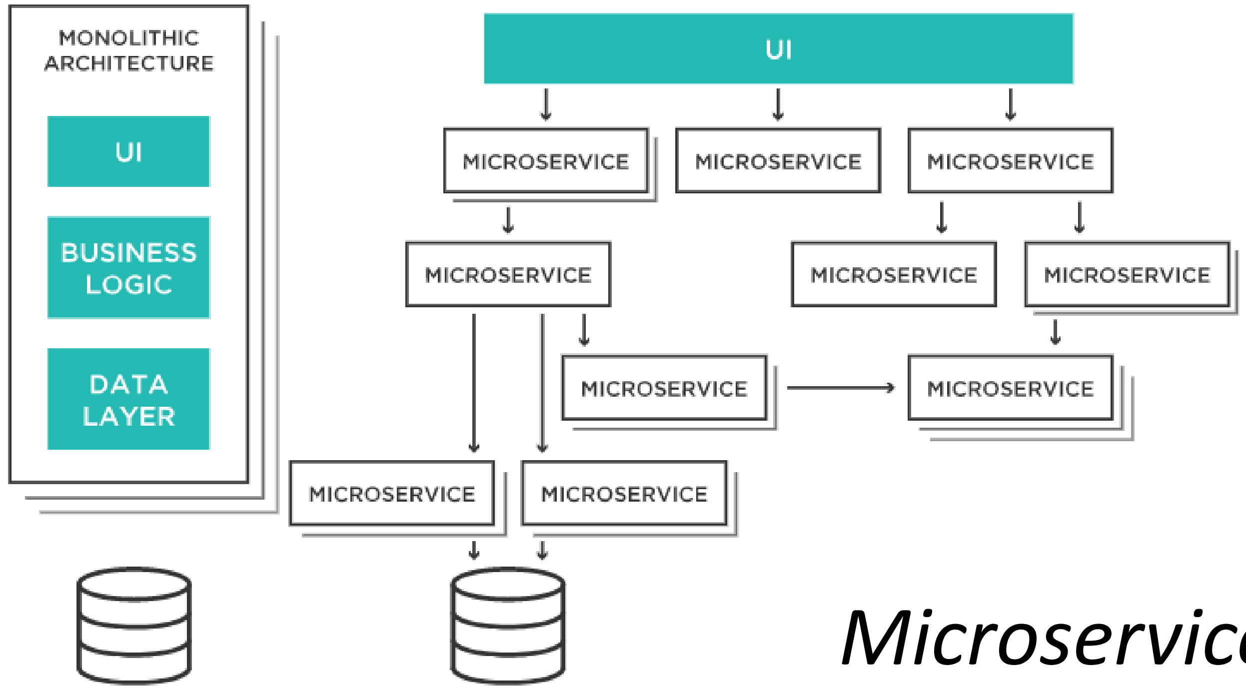
https://research.ark-invest.com/hubfs/1_Download_Files_ARK-Invest/White_Papers/ARK%E2%80%93Invest_BigIdeas_2021.pdf?hsCtaTracking=4e1a031b-7ed7-4fb2-929c-072267eda5fc%7Cee55057a-bc7b-441e-8b96-452ec1efe34c

Solution: Hardware/Software Co-Design (Accelerators)

Software

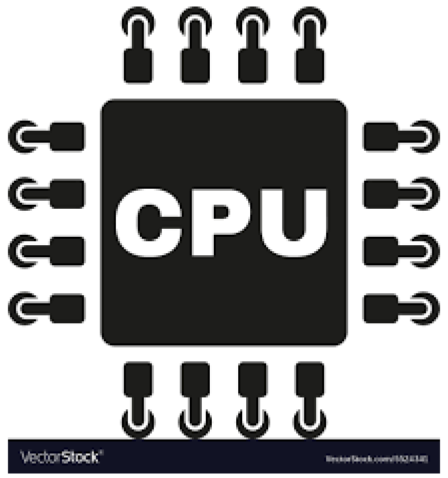


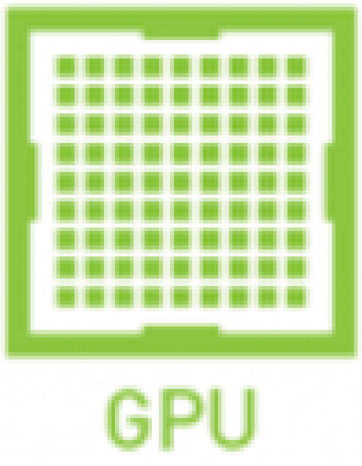
Deep Learning

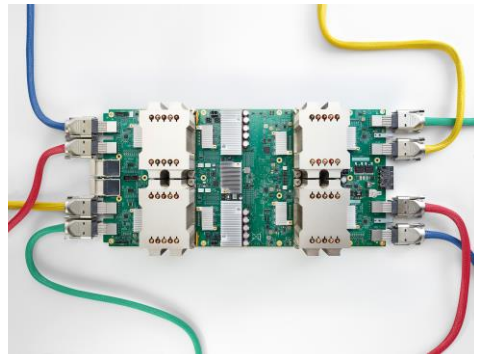


Microservices

Hardware

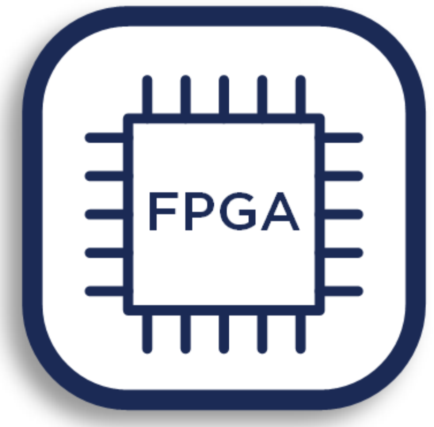


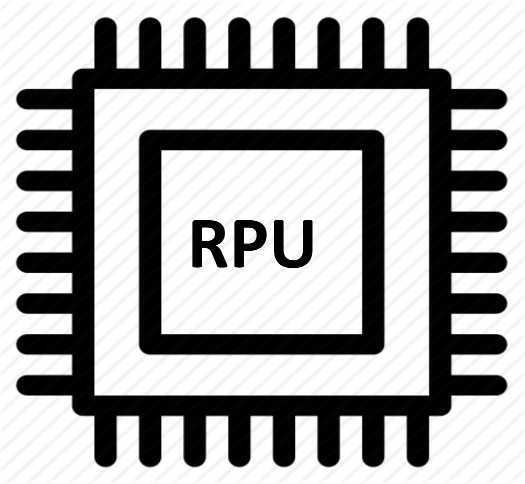




TPU

VCU

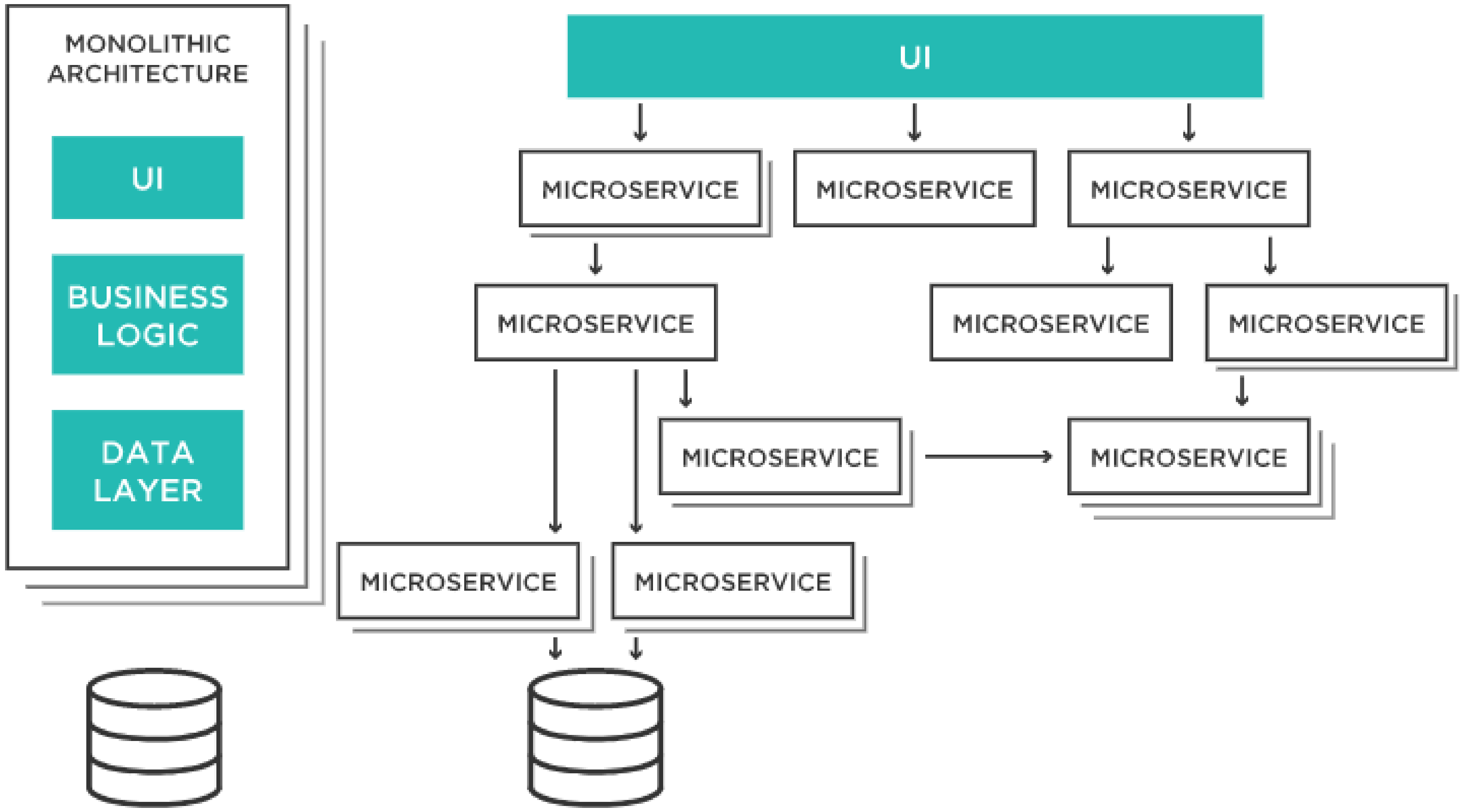




Accelerators

Microservices Architecture

The microservices architecture has become a de facto standard for developing large-scale web applications.



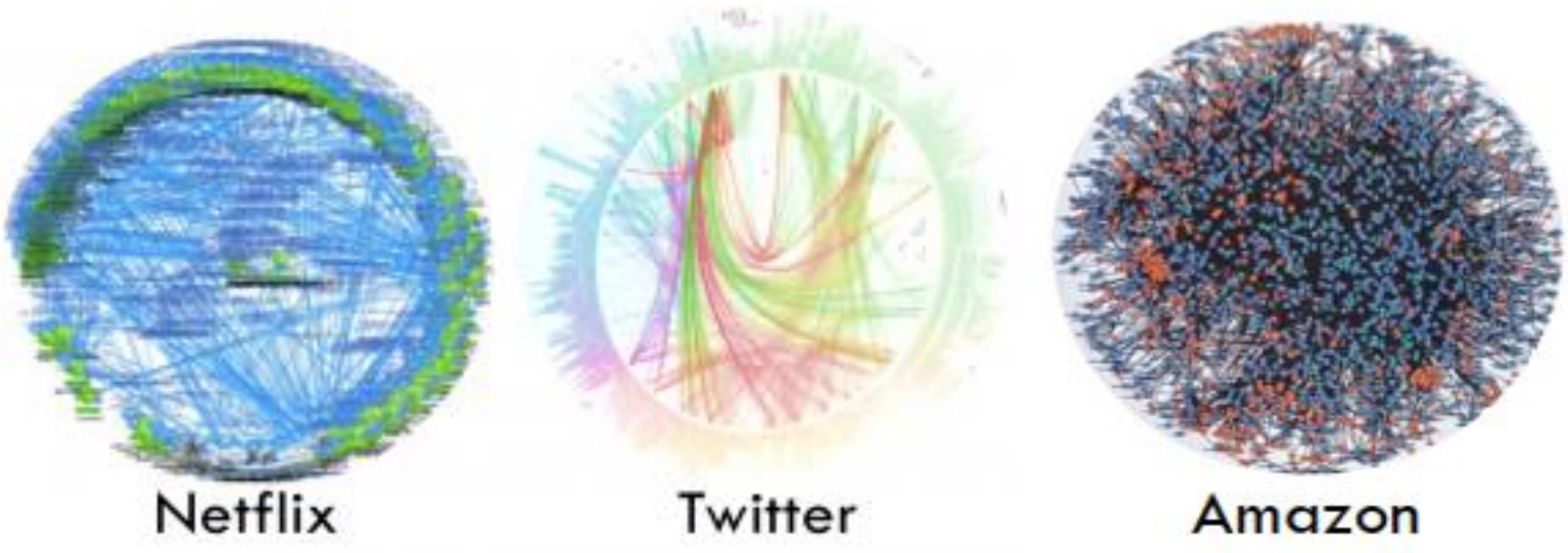
Benefits:
- Scalability
- Modularity
- Easy to maintain/debugging
- Different programming languages
- Loose-coupling, reliability
- Owned by a small team

Drawbacks:
- Network processing overhead
- High context-switching overhead
- Complex cluster management

https://www.tibco.com/reference-center/what-is-microservices-architecture

Yu Gan et al., "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems", ASPLOS 2021

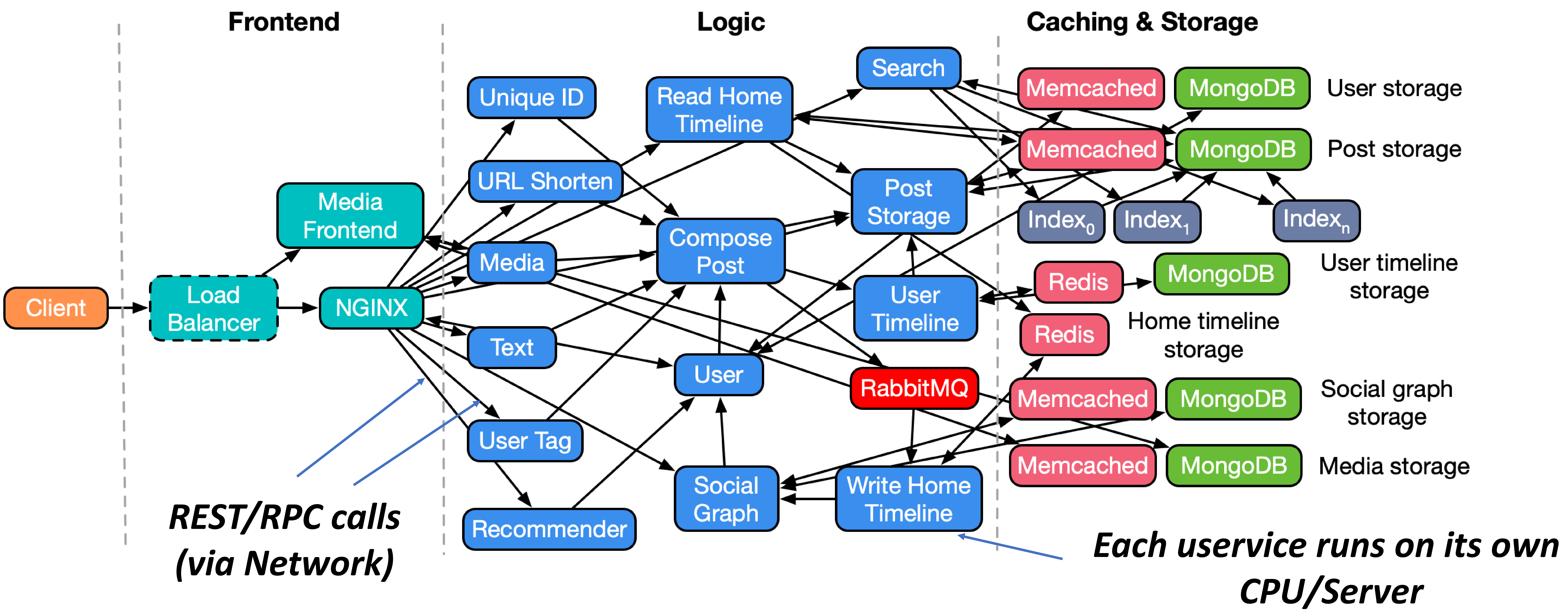
Reality is Much Complex



Microservices graph of large cloud services
Recent increased interest in "*Nanoservices*"

Image source: https://www.sigarch.org/reacting-to-new-trends-in-cloud-software/
https://www.bmc.com/blogs/microservice-vs-nanoservice/
Ibanez, Stephen, et al. "The nanoPU: Redesigning the CPU-Network Interface to Minimize RPC Tail Latency." *arXiv preprint arXiv:2010.12114* (2020).

Microservice Example: SocialNetwork



Yu Gan et al., "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems", ASPLOS 2019

Server Workloads on CPUs

- Ferdman [ASPLOS'14], Grant [HPCA'18], Grant [ISCA'21],…..
- Conclusions: CPUs are inefficient in the datacenter
  - L3 cache & DRAM BW are underutilized (low MLP)
  - ILP is limited (IPC per thread=0.25-1, average is 0.5)
  - L3 cache hit rate is low and hardware data prefetchers are ineffective
  - "Low coherence & core-to-core communication"

→ They suggest an increase in the number of threads on-chip is necessary to better use these resources

Ferdman, Michael, et al. "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware", APSLOS 2014
Ayers, Grant, et al. "Memory Hierarchy for Web Search", HPCA 2018
Ayers, Grant, et al. " AsmDB: Understanding and Mitigating Front-End Stalls in Warehouse-Scale Computers", ISCA 2019
Ayers, Grant, et al. " Classifying Memory Access Patterns for Prefetching ", ASPLOS 2020
Gope, et al. "Architectural Support for Server-Side PHP Processing ", ISCA 2017
Gan, Yu, et al. "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems." ASPLOS 2019
Kanev, Svilen, et al. "Profiling a warehouse-scale computer." .ISCA 2015
Sriraman, Akshitha, et al. "Softsku: Optimizing server architectures for microservice diversity@ scale." ISCA 2019.
Sriraman, Akshitha, et al. "Accelerometer: Understanding acceleration opportunities for data center overheads at hyperscale." ASPLOS 2020.

Observations Summary

- All the requests/threads run the “same” program (SPMD)
- Threads rarely communicate
- The control flow are coherent and less divergent
- Instruction and data footprint is getting smaller
- Batching is heavily used in datacenter services
- We need energy-efficient high-throughput system

What does this look like?

Single Instruction Multiple Threads (SIMT) Or SIMD

Observations Summary

- All the requests/threads run the “same” program (SPMD)
- Threads rarely communicate
- The control flow are coherent and less divergent
- Instruction and data footprint is getting smaller
- Batching is heavily used in datacenter services
- We need energy-efficient high-throughput system

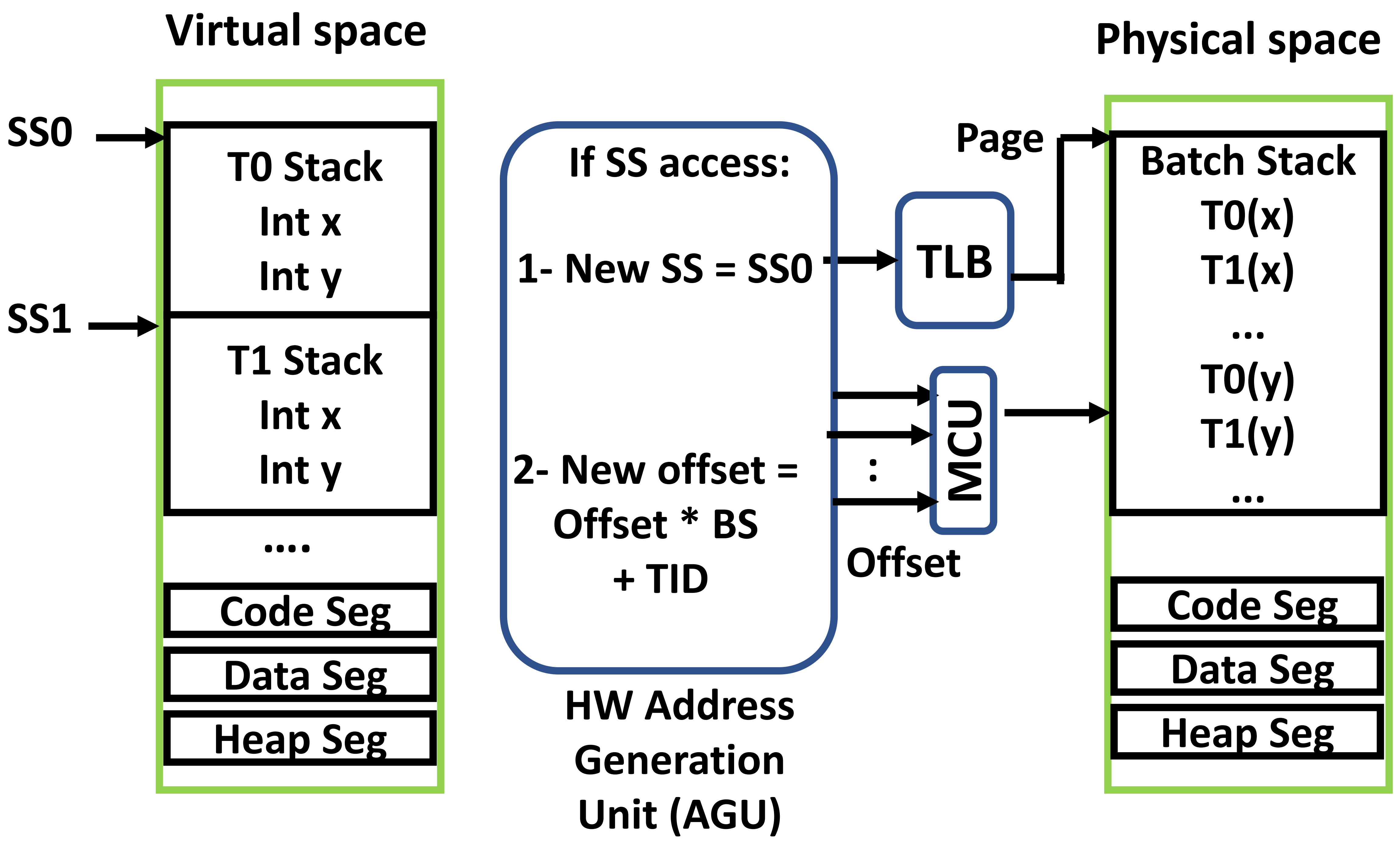
What does this look like?

Single Instruction Multiple Threads (SIMT) Or SIMD

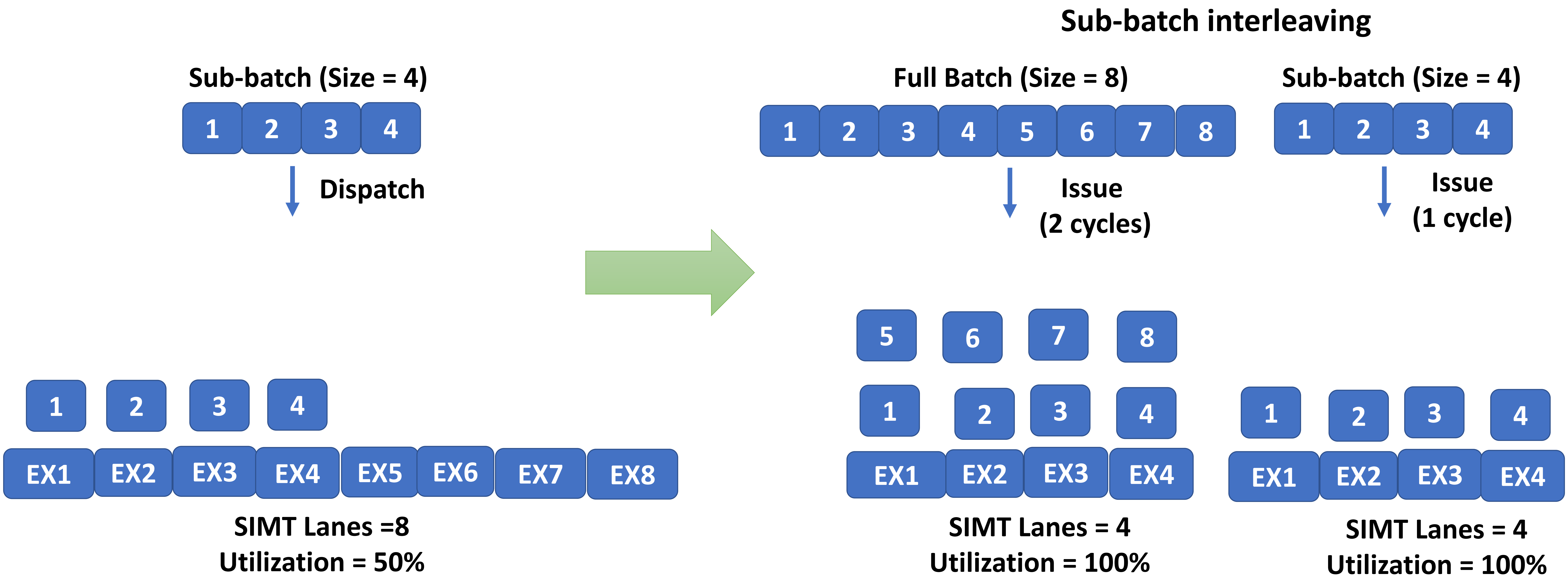
But, wait, what about service latency?

More RPU Hardware Details

Transparent Stack Segment Coalescing



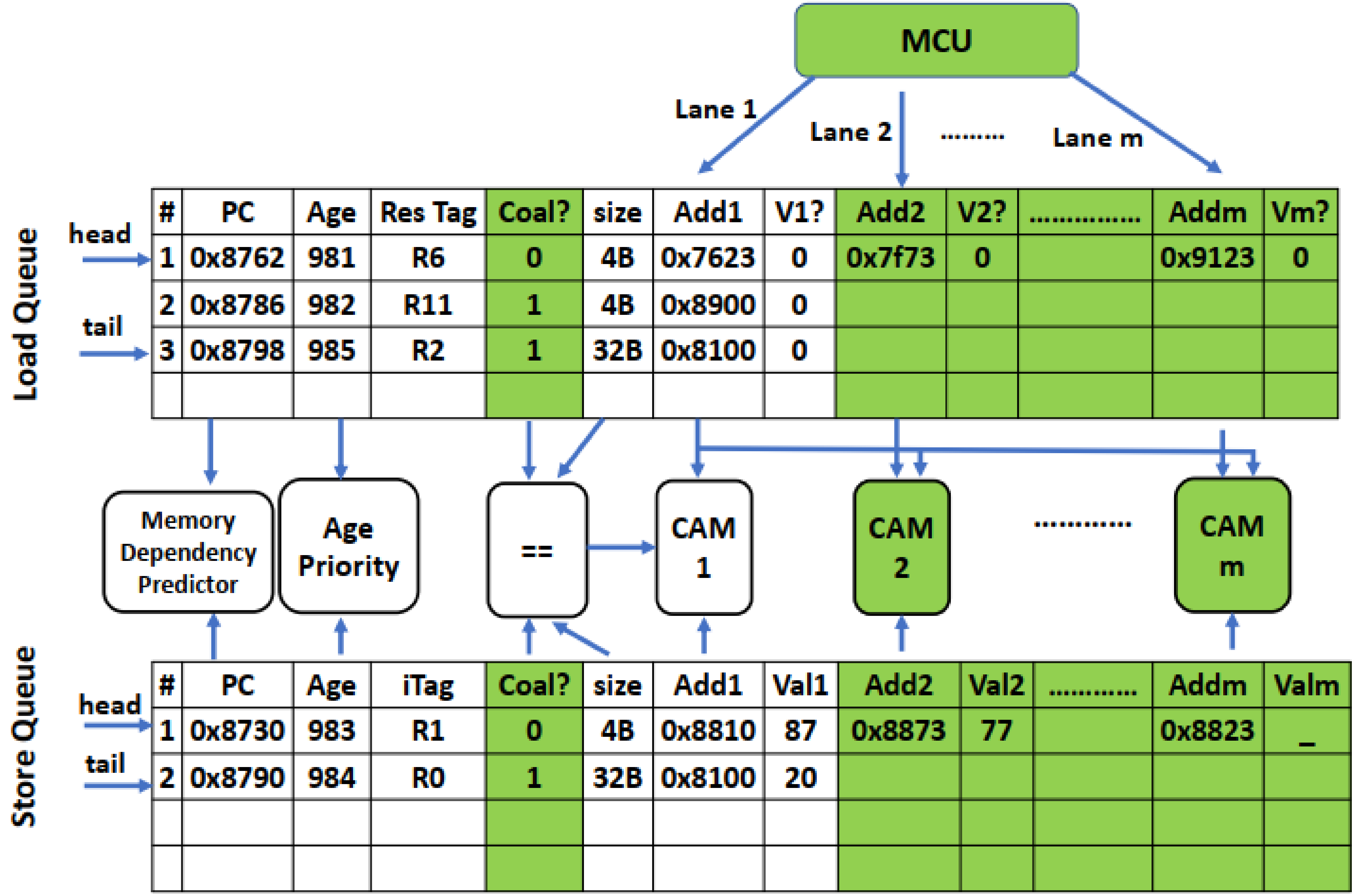
Sub-batch Interleaving



→ Alleviate divergence, exploit deeper pipeline & fully utilize your IPC utilization

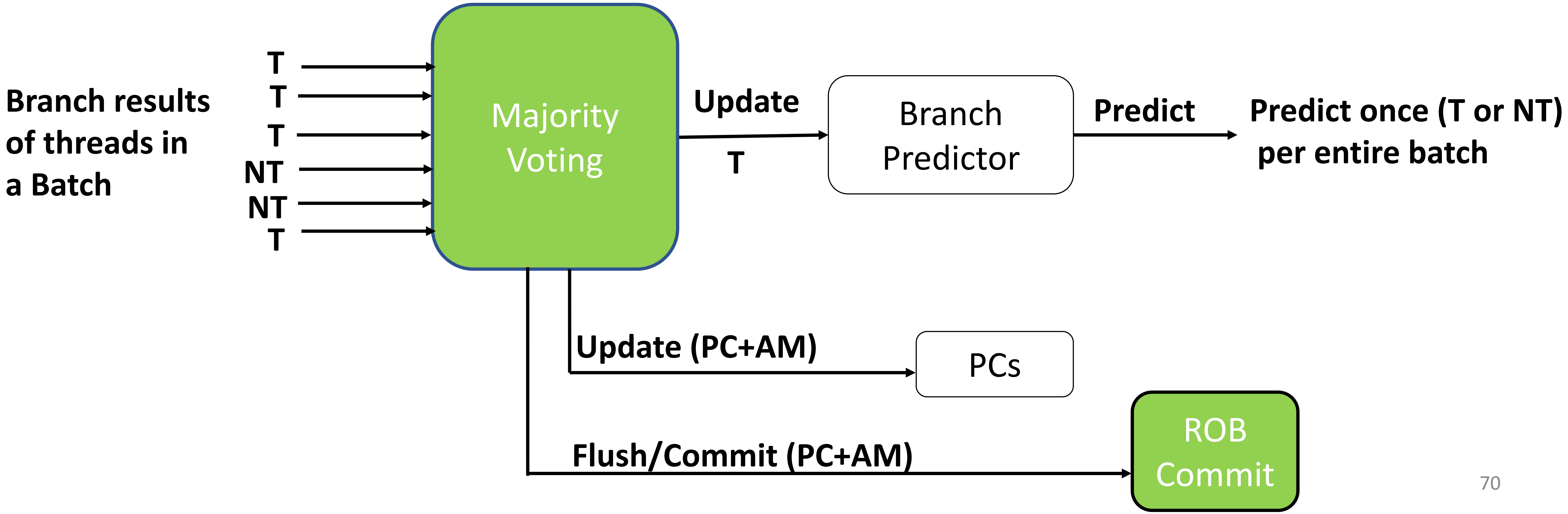
→ In our final RPU configuration, SIMT lanes = 8 & max batch size = 32

RPU's LD/ST Unit

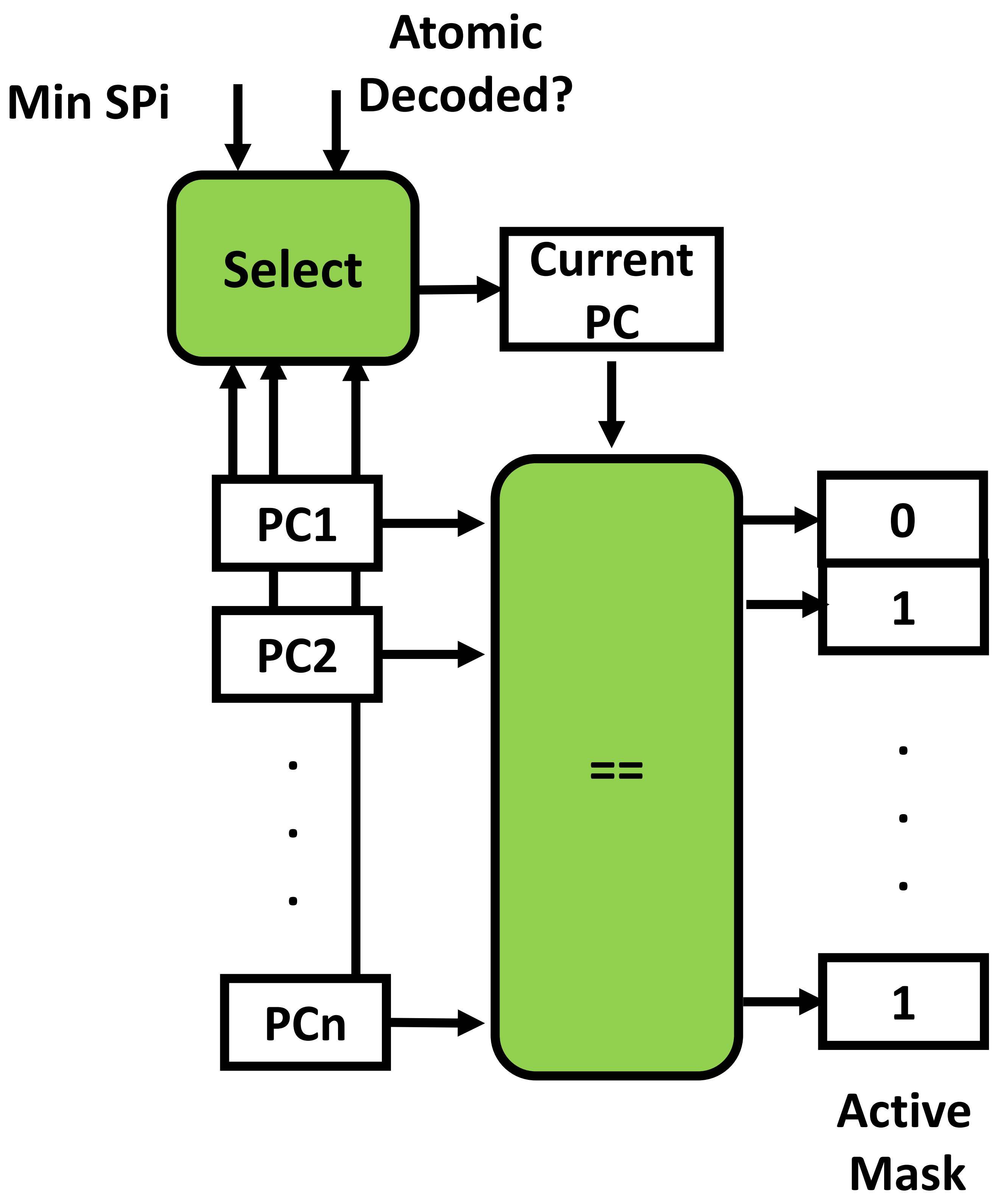


SIMT + Branch Predictor

- The branch predictor operates at the batch granularity, i.e., only one prediction is generated for all the threads in a batch.



Transparent Deadlock-free Stack-less Convergence Optimizer



How to select?

1- Current PC = PCi of Min (SPi)

2- If all SPi are equal

  Current PC = min (PCi)

3- If deadlock detection (a thread X has not update PC for *m* cycles and frequent atomics are decoded)

  → Current PC = X(PC) for *k* cycles

Weak Consistency + NMCA

- Important lesson learned from the GPU space:
  - Traditional coherence/consistency model (MOESI/TSO) does not efficiently scale beyond 100/1K threads
  - You need to relax your consistency model to continue thread scaling
- Good news: (key observations)

  (1) Data Center workloads rarely communicate and exhibit low locks, read-write sharing and overall low coherence traffic

  (2) Multiple copy atomicity (MCA) is not required by most of the data center applications. As eventual consistency is widely adopted

  - Example: For facebook, It is okay for a friend to see the post update before others

So, lets apply more-scalable weak consistency with non multi copy atomicity model (NMCA)

Ferdman, Michael, et al. "Clearing the clouds: a study of emerging scale-out workloads on modern hardware." ASPLOS 2012
Ayers, Grant, et al. "Memory hierarchy for web search." *HPCA 201*

RPU’s Consistency Model

- Weak Consistency + NMCA. What does this mean?
  - Private caches are only guaranteed to be coherent and consistent at barriers & fences
    - Move atomics to L3 cache → negligible performance impact as we have low locks
  - A simple, relaxed, directory-based coherence protocol with no-transient states or invalidation acknowledgments → only ack at barrier (see HMG [HPCA’20])
  - Multiple threads can share the same store queue per core

This relaxed memory model allows RPU to scale the number of threads efficiently, improving thread density by an order of magnitude

- Other good news: some CPU ISAs, like ARMv7 and IBM POWER, already support a weak consistency model with NMCA

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

GPU vs RPU Keywords

GPU	RPU
Grid/Thread Block	SW Batch
Warp	HW Batch
Thread	Thread/Request
Kernel	Service
GPU Core / Streaming MultiProcessor (SM)	RPU Core / Streaming MultiRequest (SM)
Warp Scheduler	Batch Scheduler
Single Instruction Multiple Thread (SIMT)	Single Instruction Multiple Request (SIMR)
CUDA core	Execution lane

CPU Inefficiencies and RPU's Mitigation

Table 4.3. CPU inefficiencies in the data center

Data center characteristics & CPU inefficiency	**RPU's mitigation**
Request similarity [155] & high frontend power consumption [11]	SIMT execution to amortize frontend overhead
Inter-request data sharing [143]	Memory coalescing and an increase in the number of threads sharing private caches
Low coherence/locks [142], [143] and eventual consistency [186]	Weak memory ordering, relaxed coherence with non-memory-copy-atomicity & higher bandwidth core-to-memory interconnect
Low IPC due to frequent frontend stalls and memory latency [29], [32], [141]–[144]	Multi-thread interleaving
DRAM & L3 BW are underutilized, data prefetchers are ineffective [30], [142], [143], [145]	High thread level parallelism (TLP) to fully utilize BW
Microservice/nanoservice have a smaller cache footprint [26]	High TLP and decrease L1&L2 cache capacity/thread

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.

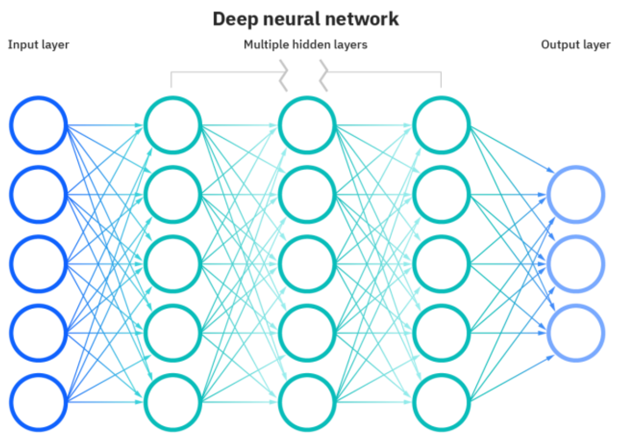
Quoted: “Clearly, datacenter applications limit latency, not batch size. Future DSAs should take advantage of larger batch sizes”

- (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

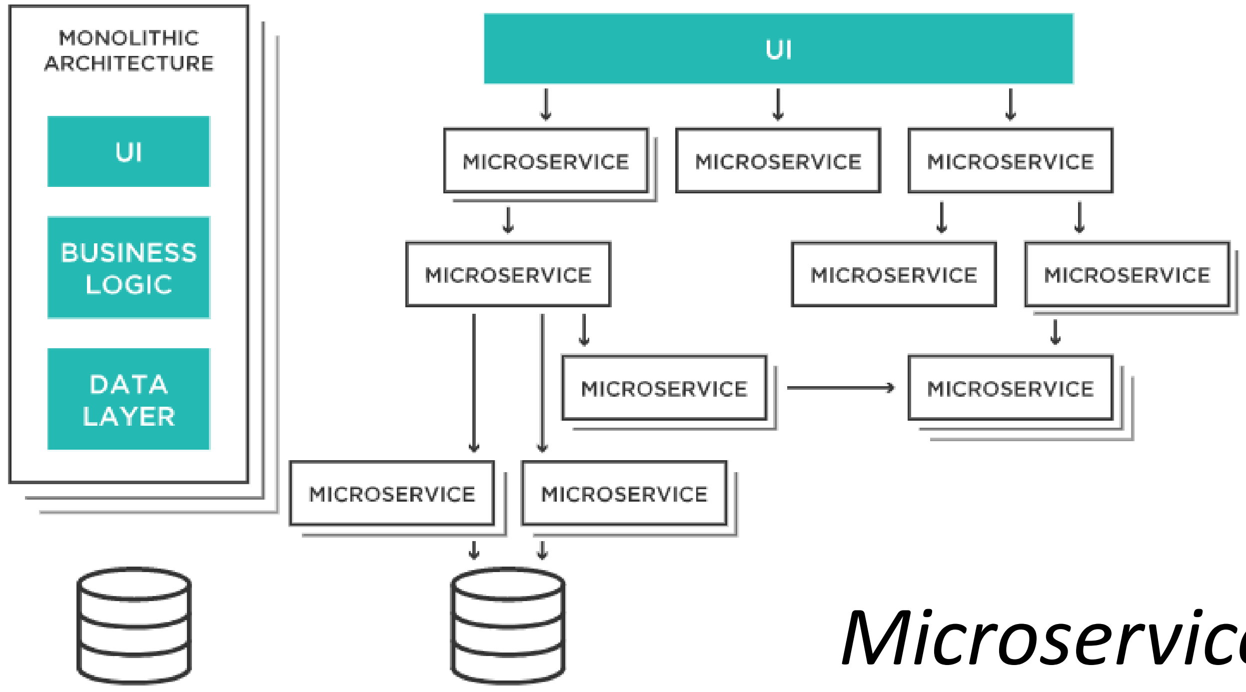
Jouppi, Norman P., et al. "Ten Lessons From Three Generations Shaped Google’s TPUv4i: Industrial Product." *2021 ISCA*

Thank You!
Q&A?

Software

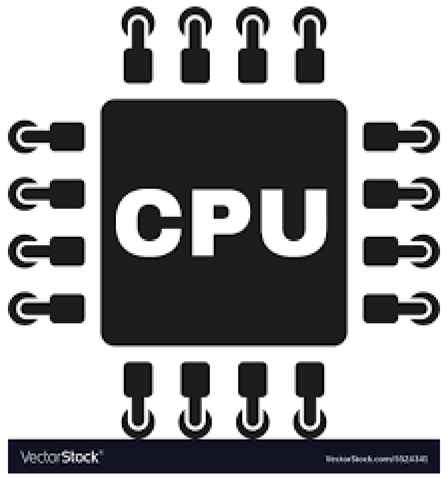


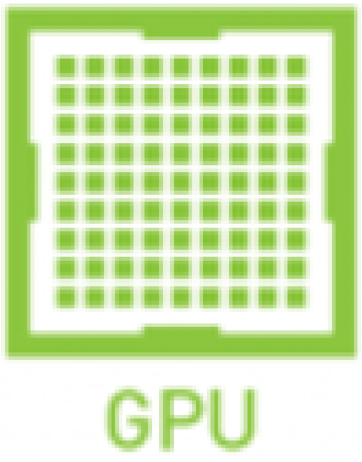
Deep Learning

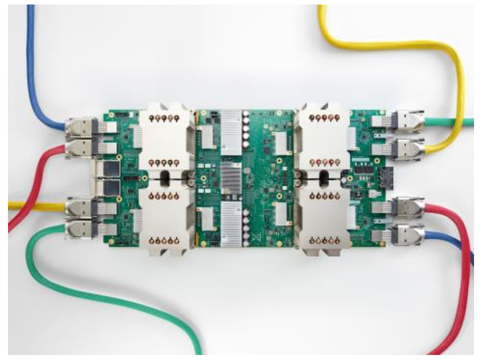


Microservices

Hardware

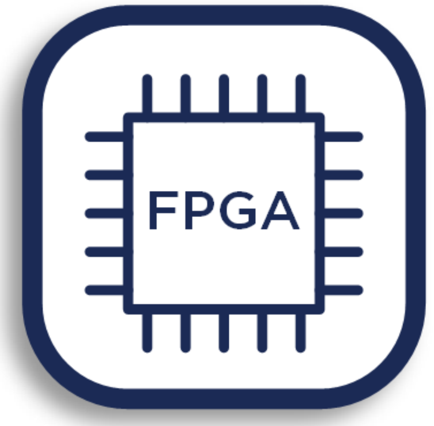


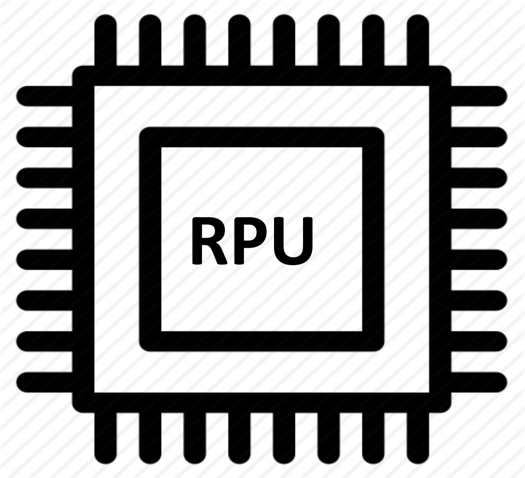




TPU

VCU





Accelerators