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

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

Purdue University

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

Growth of Hyperscale Data Centers
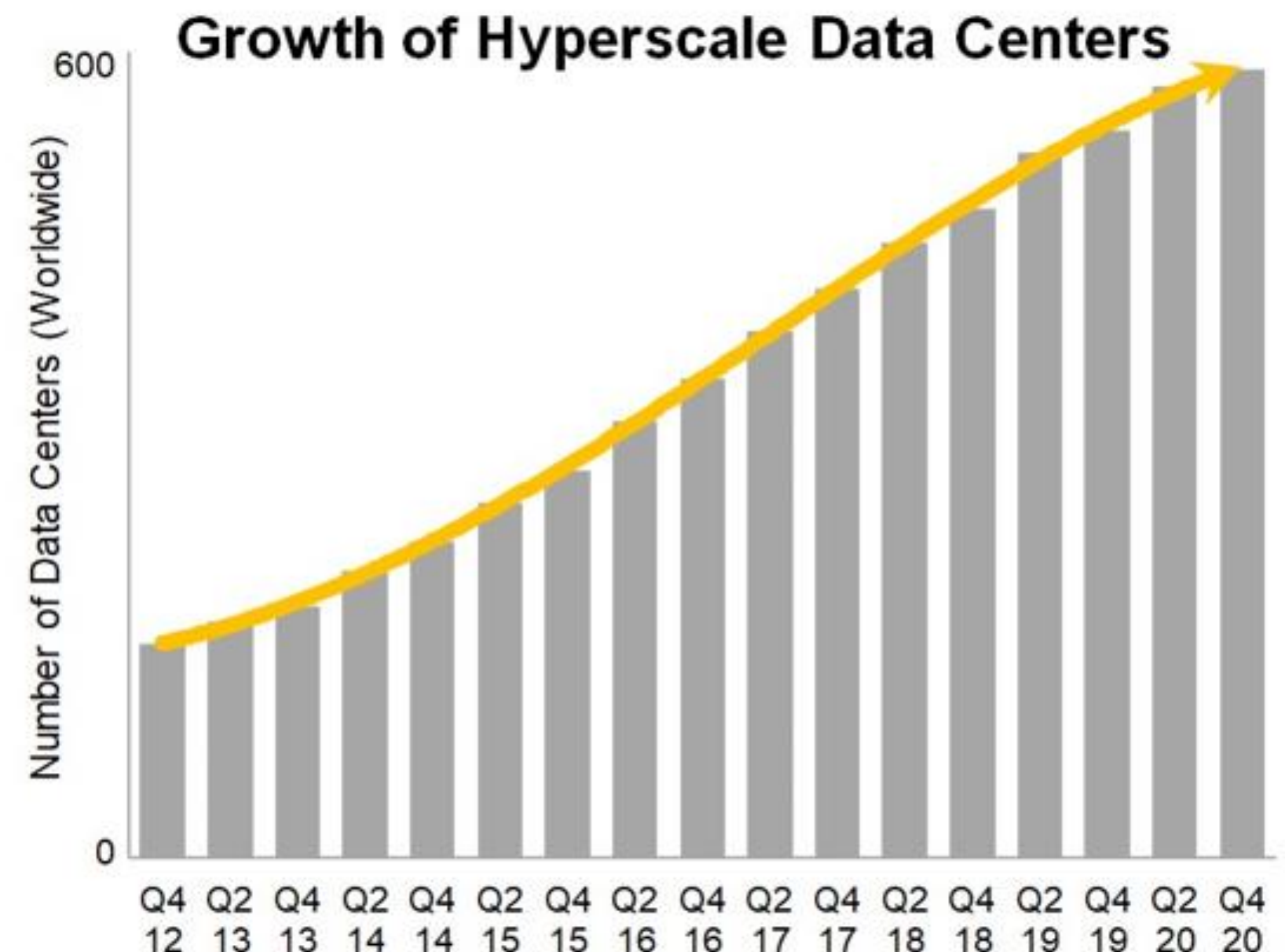
Datacenter Power Breakdown
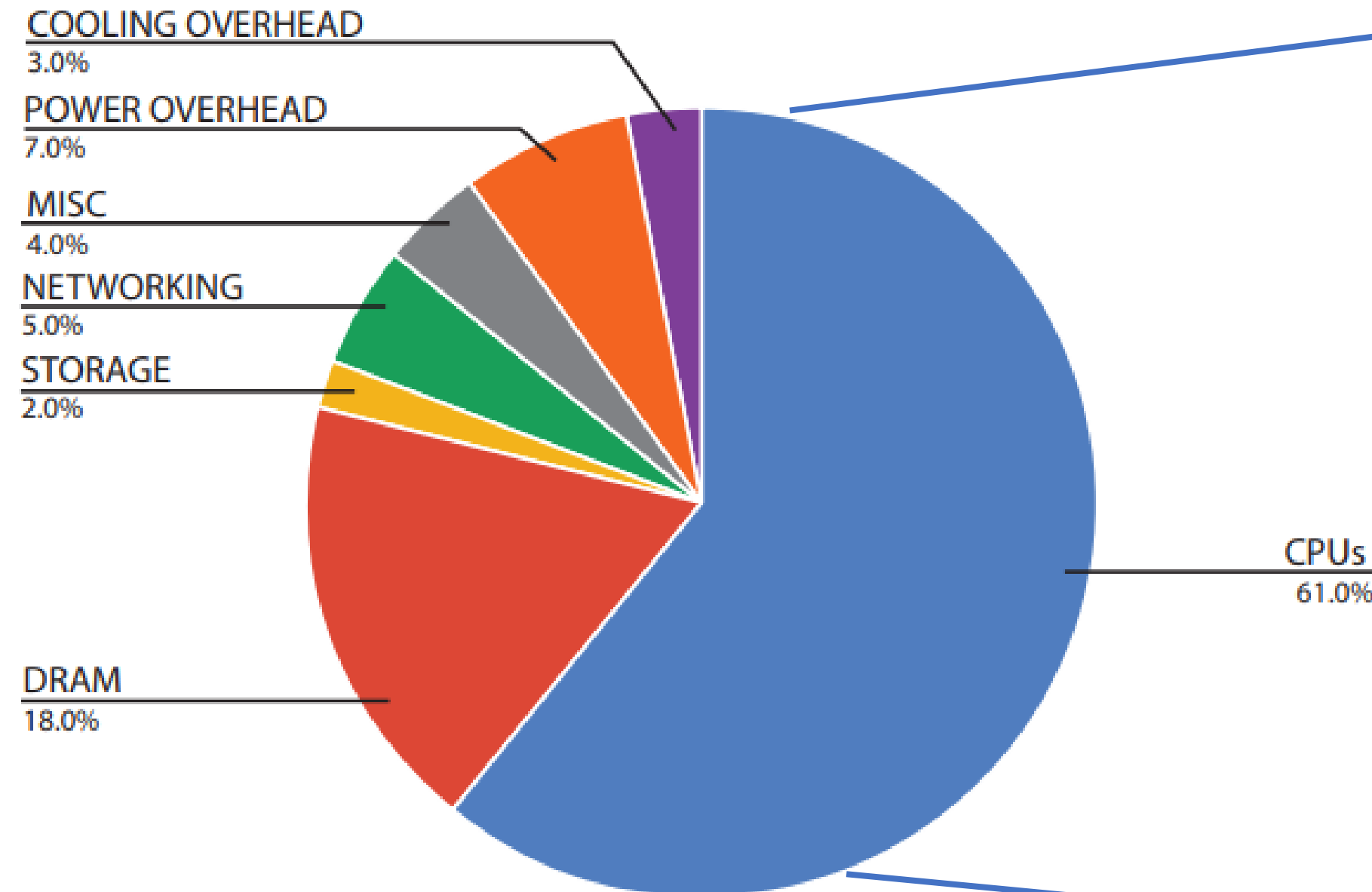
Datacenter Power Breakdown (from Google)
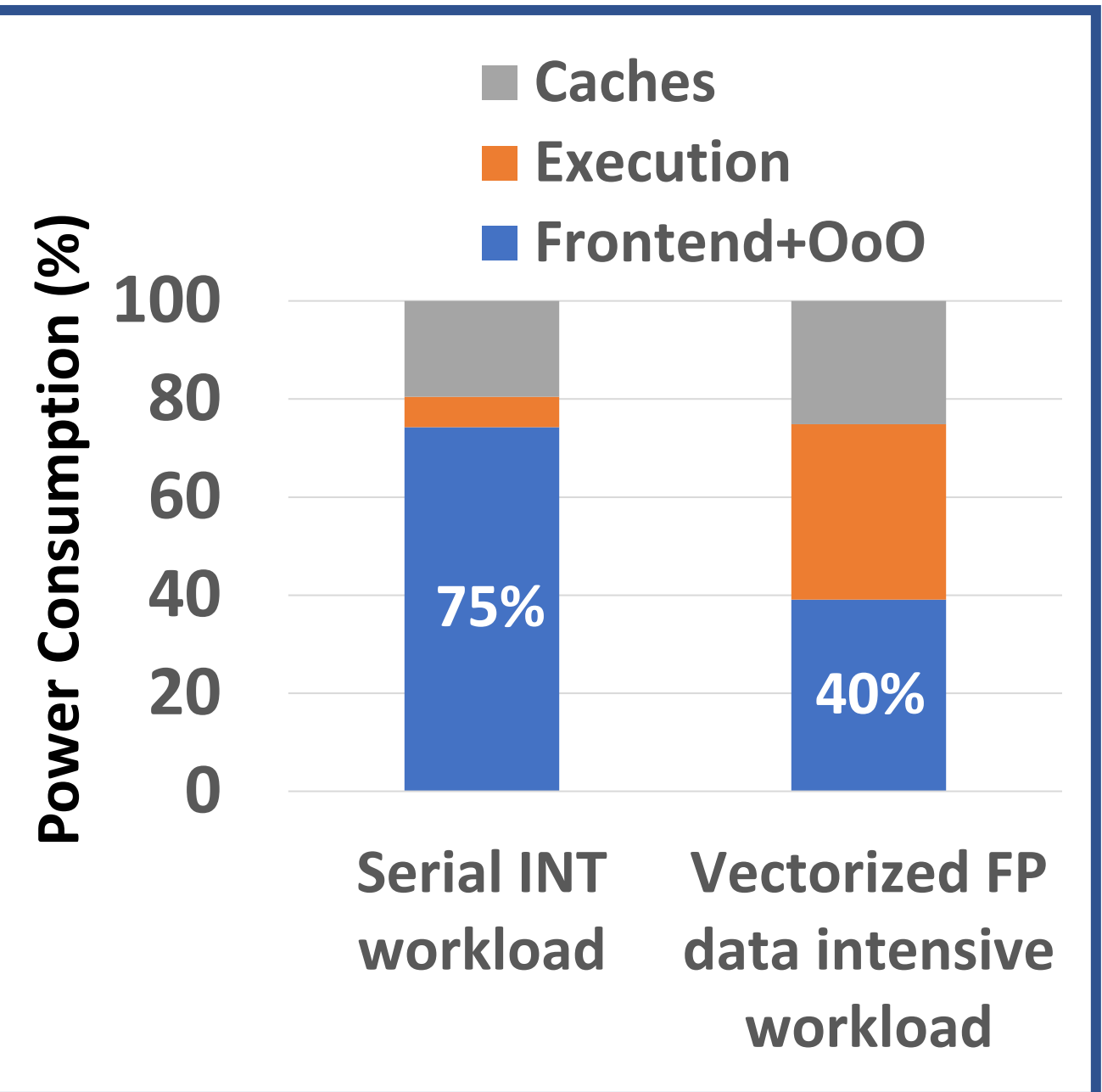
CPU Power Breakdown

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
<u>Notes</u>: 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, Million of Users

Google

facebook

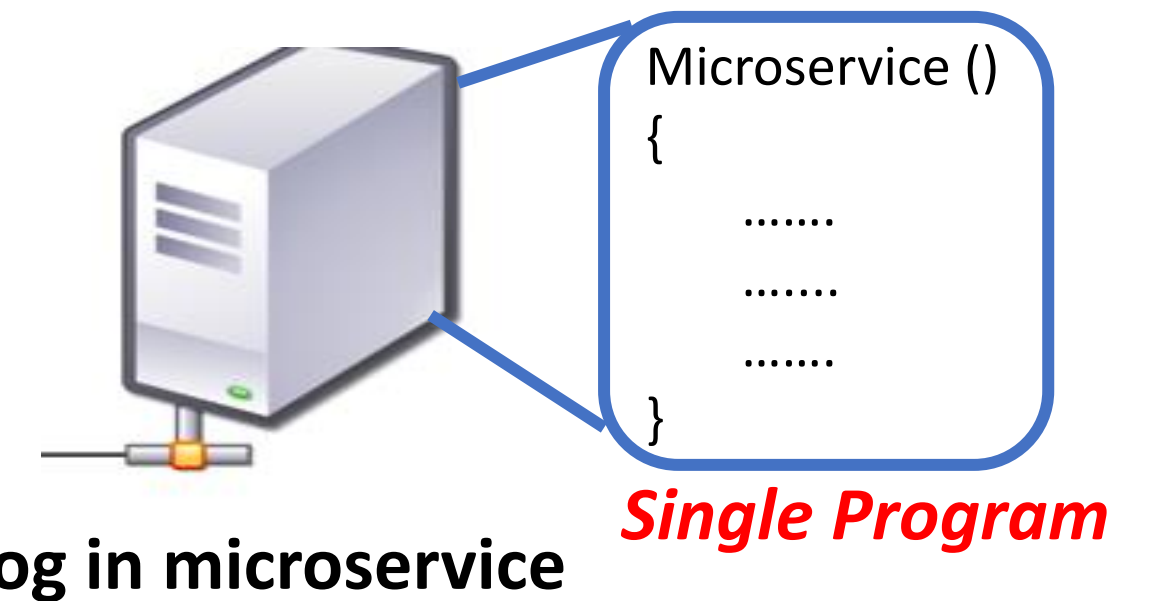**Private Datacenter**

Uber

AWS

Public Datacenter

NETFLIX

"Similar" Request-Level Parallelism
1000s of independent requests are all running the same code

Log-in reqs

("xyz", "1234")
("john", "5678")
("ma98", "4444")
("mah", "ko56")

Multiple Data

log in microservice

Microservice ()
{
   ....
   ....
   ....
}

Single Program

search reqs

("purdue univ")
("arsenal fc")
("elections 2024")
("stock today")

search microservice

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

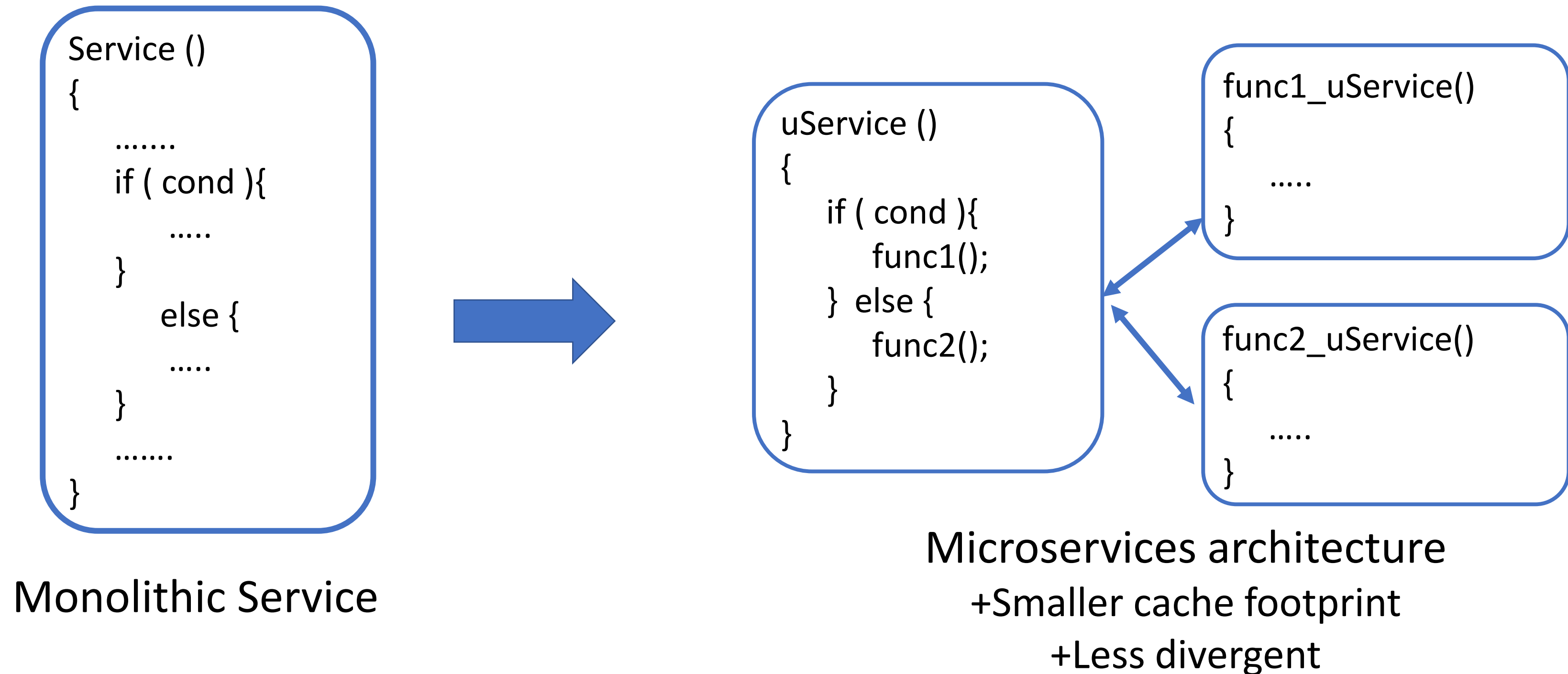Recall: GPUs and SIMDs were designed to execute data parallel portion (i.e., loops) not the entire application

"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

```
Service ()
{
    .......
    if ( cond ){
        .....
    }
        else {
        .....
    }
    .......
}
```

Monolithic Service

```
uService ()
{
    if ( cond ){
        func1();
    } else {
        func2();
    }
}
```

```
func1_uService()
{
    .....
}
```

```
func2_uService()
{
    .....
}
```

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
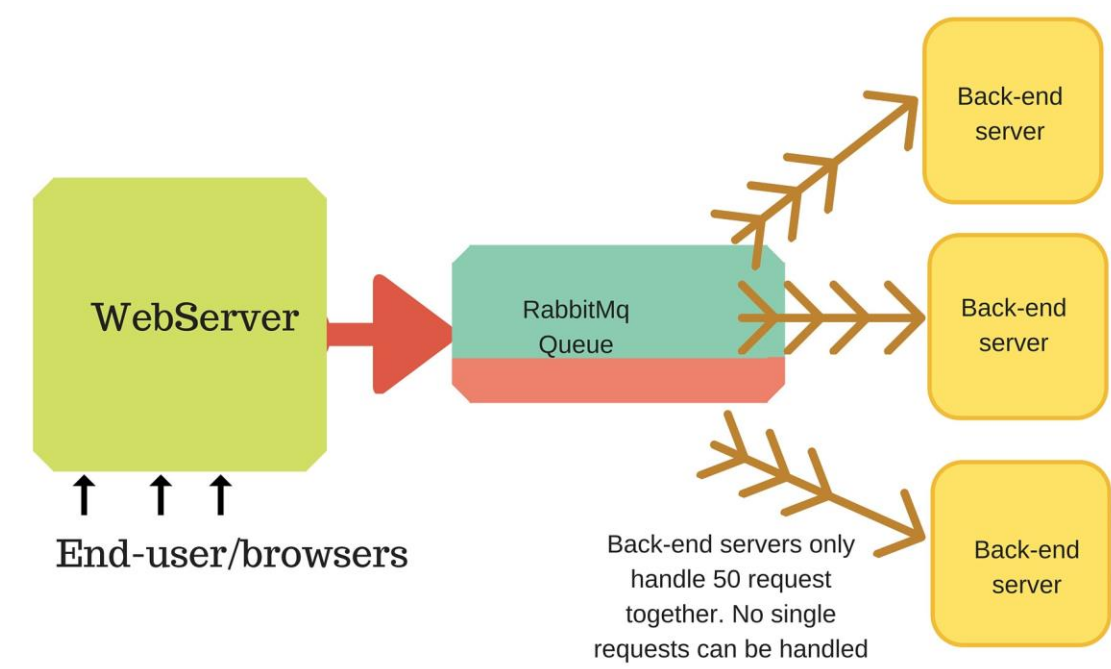
Network Batching

Power management
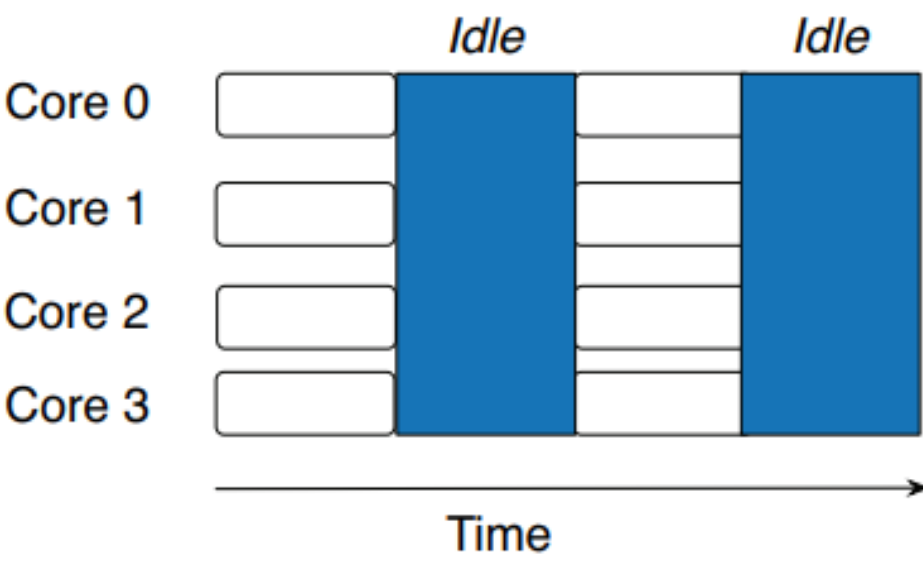
Batching for deep sleep

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*

8

Off-Chip BW Scaling
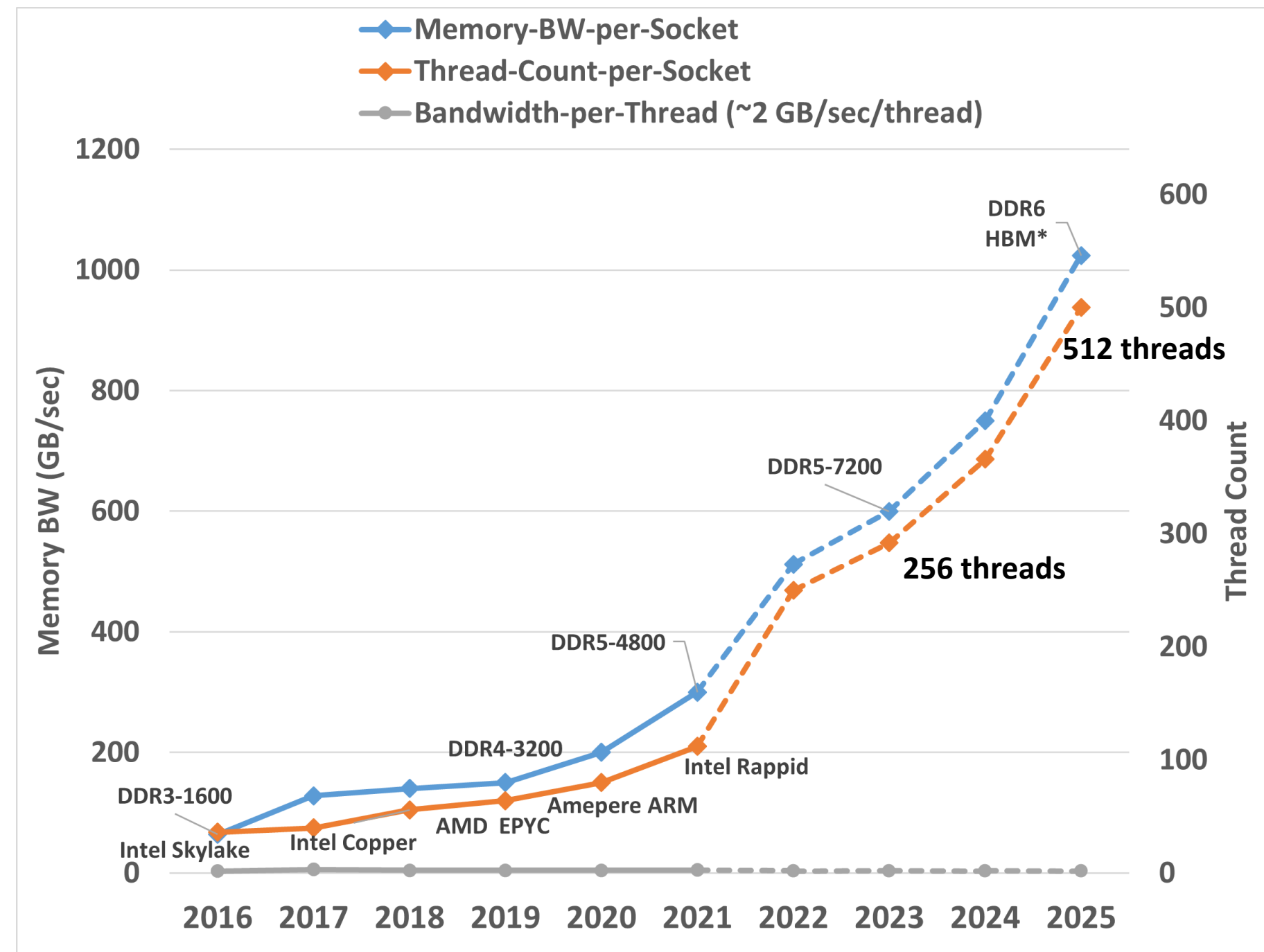
Key Observation #4: There is available headroom to increase on-chip throughput (thread count) in the foreseeable future.
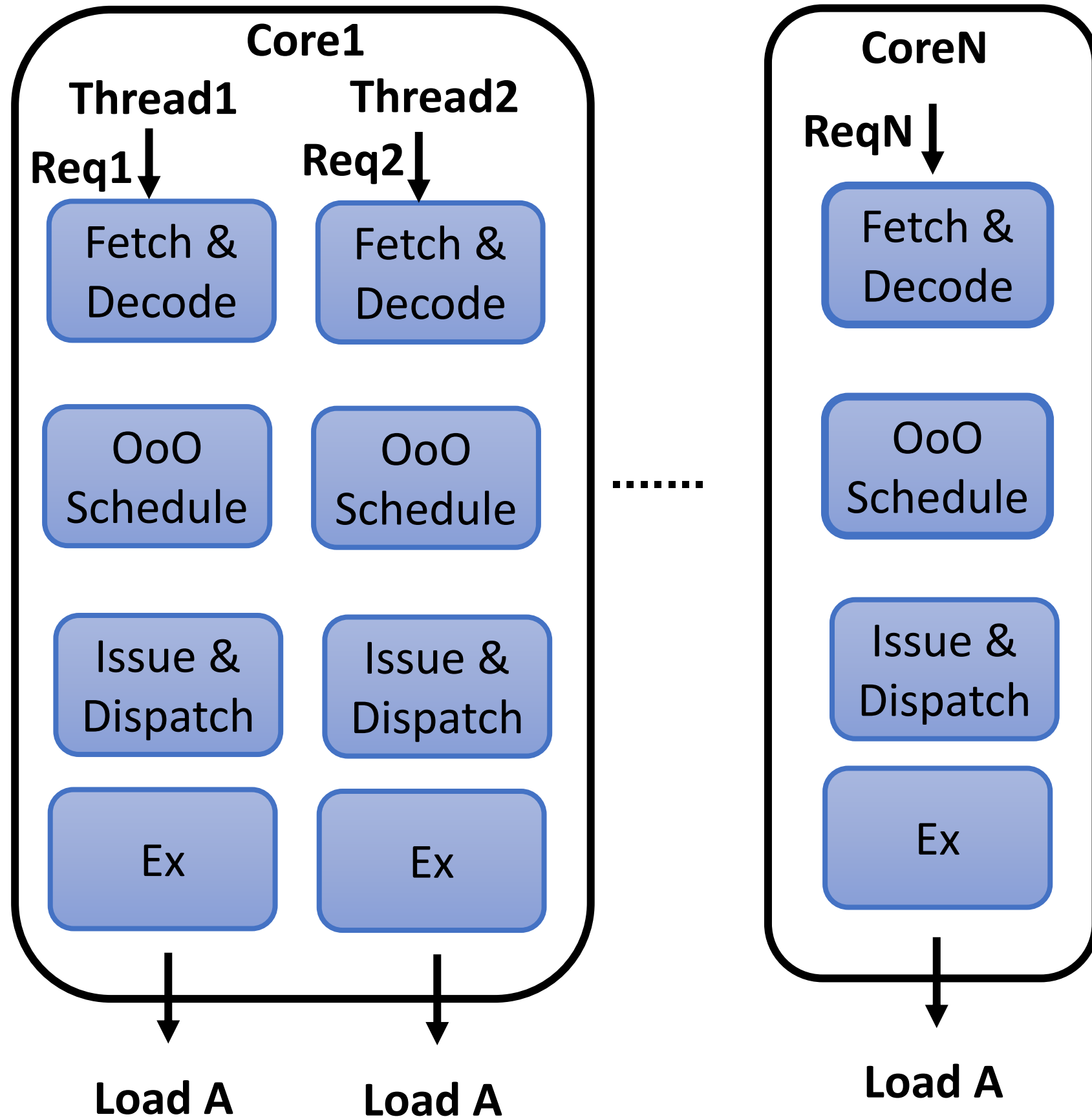
How to increase on-chip throughput of CPU?

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

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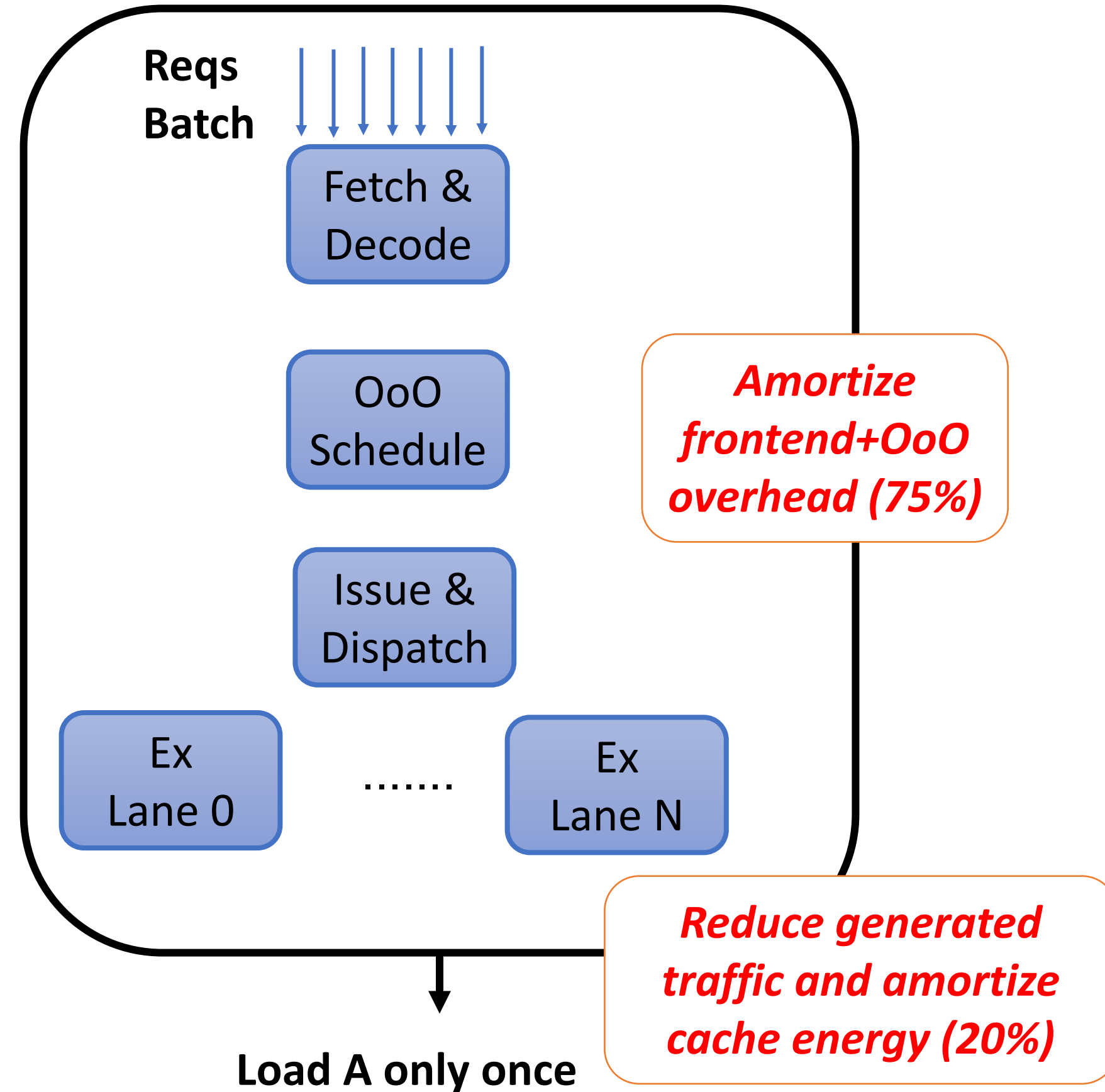 - Cost-effective (throughput/area)
  - Better scalability

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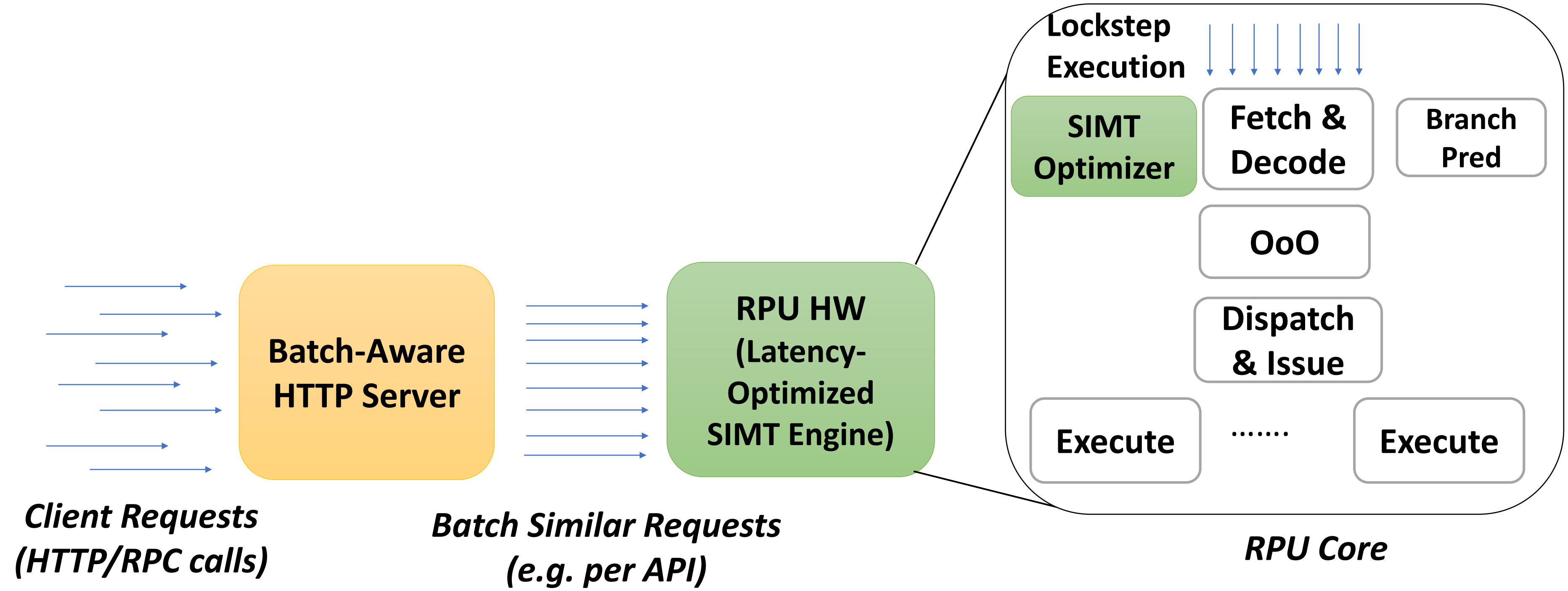
SIMT Efficiency

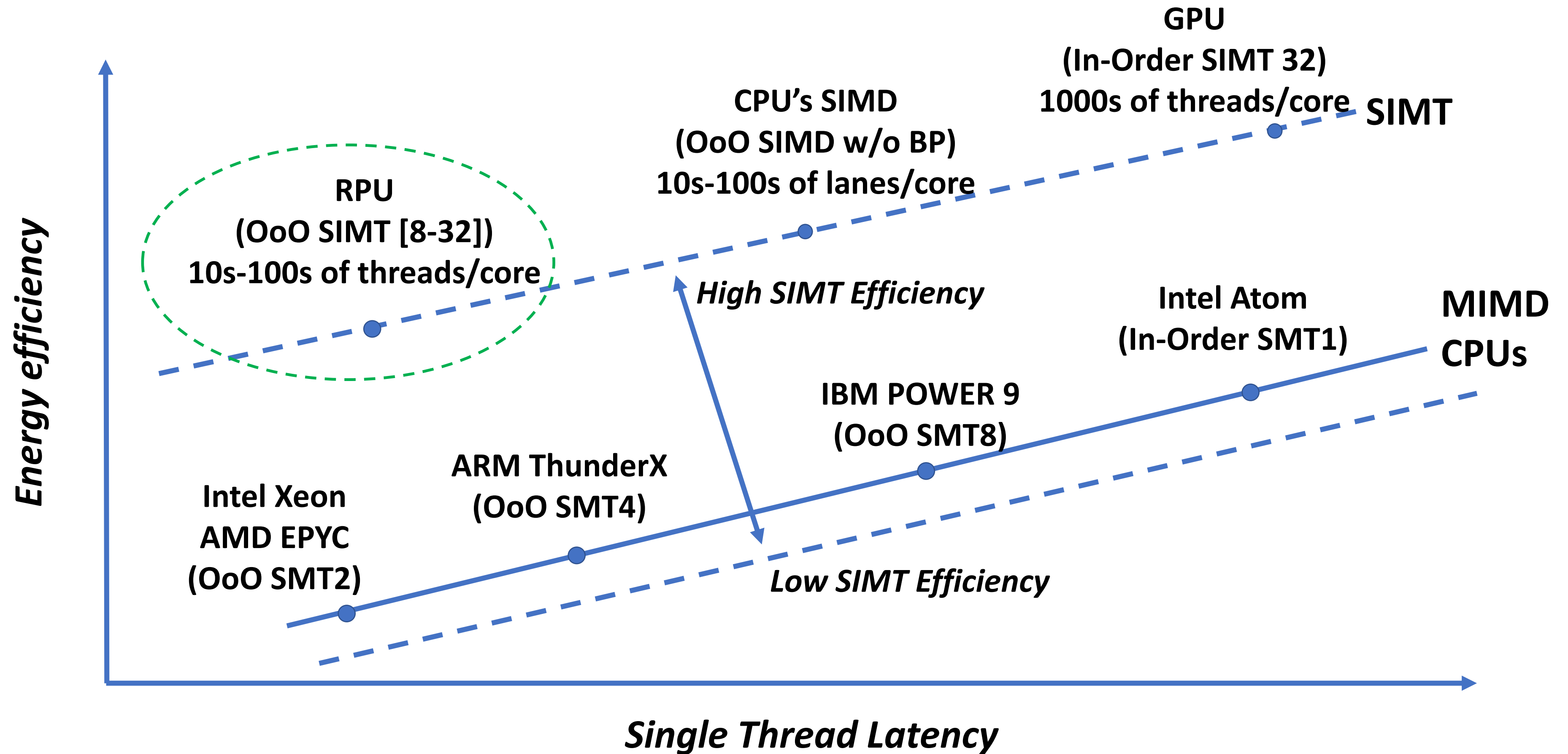**CPU Multi-Core with Simultaneous Multi-Threading**

Request Processing Unit (RPU)
SIMT+OoO Architecture

Amortize frontend+OoO overhead (75%)

Reduce generated traffic and amortize cache energy (20%)

SIMR System Overview

Client Requests
(HTTP/RPC calls)

Batch Similar Requests
(e.g. per API)

**Batch-Aware
HTTP Server**

**RPU HW
(Latency-
Optimized
SIMT Engine)**

Lockstep
Execution

**SIMT
Optimizer**

**Fetch &
Decode**

**Branch
Pred**

OoO

**Dispatch
& Issue**

Execute

......

Execute

RPU Core

Latency & Energy-Efficiency Tradeoff

GPU
(In-Order SIMT 32)
1000s of threads/core

CPU's SIMD
(OoO SIMD w/o BP)
10s-100s of lanes/core

SIMT

RPU
(OoO SIMT [8-32])
10s-100s of threads/core

High SIMT Efficiency

Intel Atom
(In-Order SMT1)

MIMD
CPUs

IBM POWER 9
(OoO SMT8)

ARM ThunderX
(OoO SMT4)

Intel Xeon
AMD EPYC
(OoO SMT2)

Low SIMT Efficiency

Energy efficiency

Single Thread Latency

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

Deep Dive into RPU's Challenges

Deep Dive into RPU's Challenges

- ## Control Divergence
  - Control divergence wit high latency branch

A (1111)

10 ns   B (1100)   C (0011)   **System call (10 ms)**

D (1111)   **Reconvergence?**

- ## Memory Divergence
  - Cache Contention & Bank Conflicts

Thrashing   **L1 cache**

Many threads

- ## Higher instruction execution & L1 hit latency
  - Due to larger execution units & cache resources at the backend

Deep Dive into RPU's Challenges

- ## Control Divergence
  - Control divergence wit high latency branch
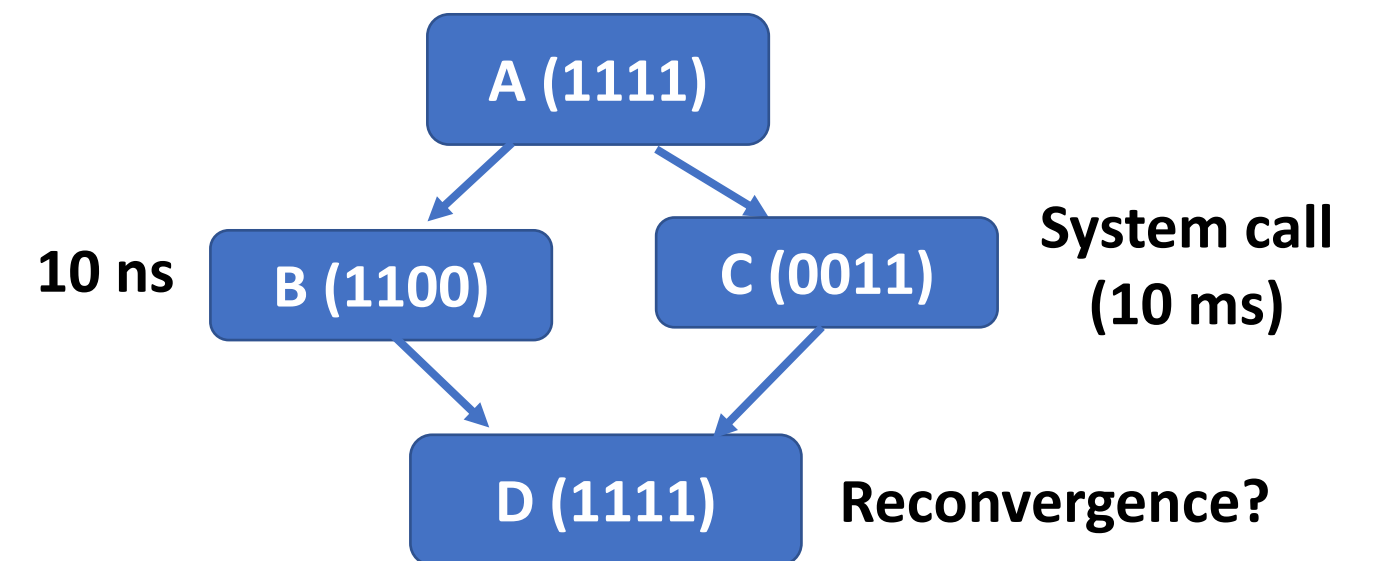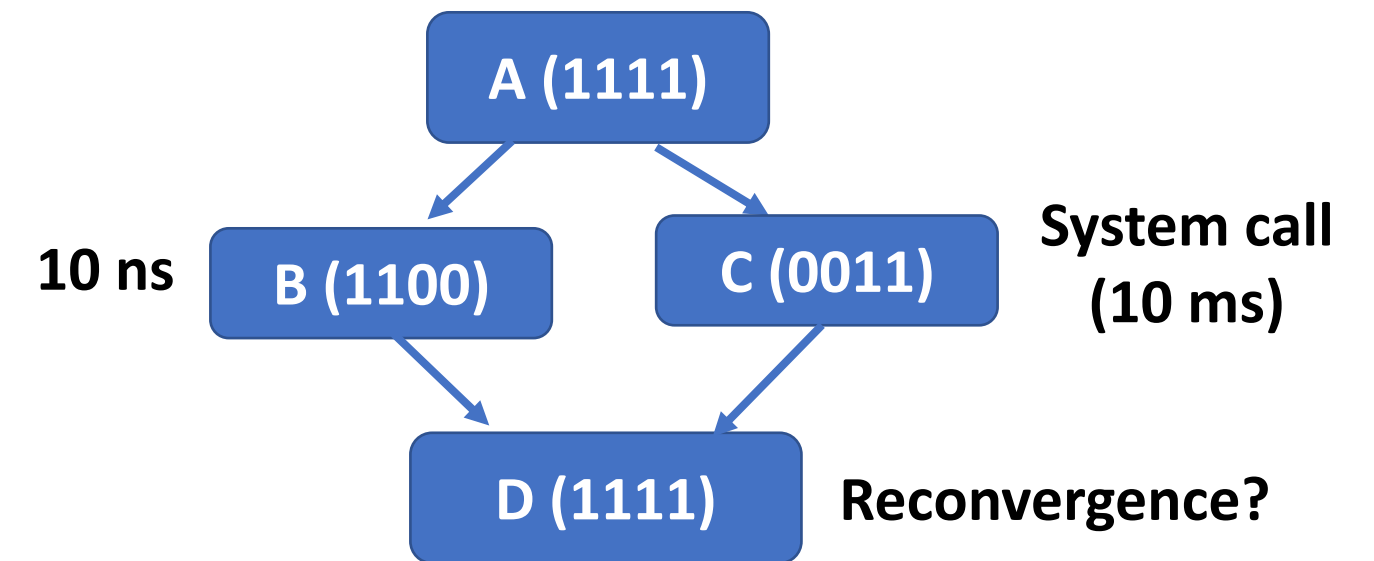
- ## Memory Divergence
  - Cache Contention & Bank Conflicts

- ## Higher instruction execution & L1 hit latency
  - Due to larger execution units & cache resources at the backend

HW/SW Stack

CPU 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

GPU 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

RPU 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

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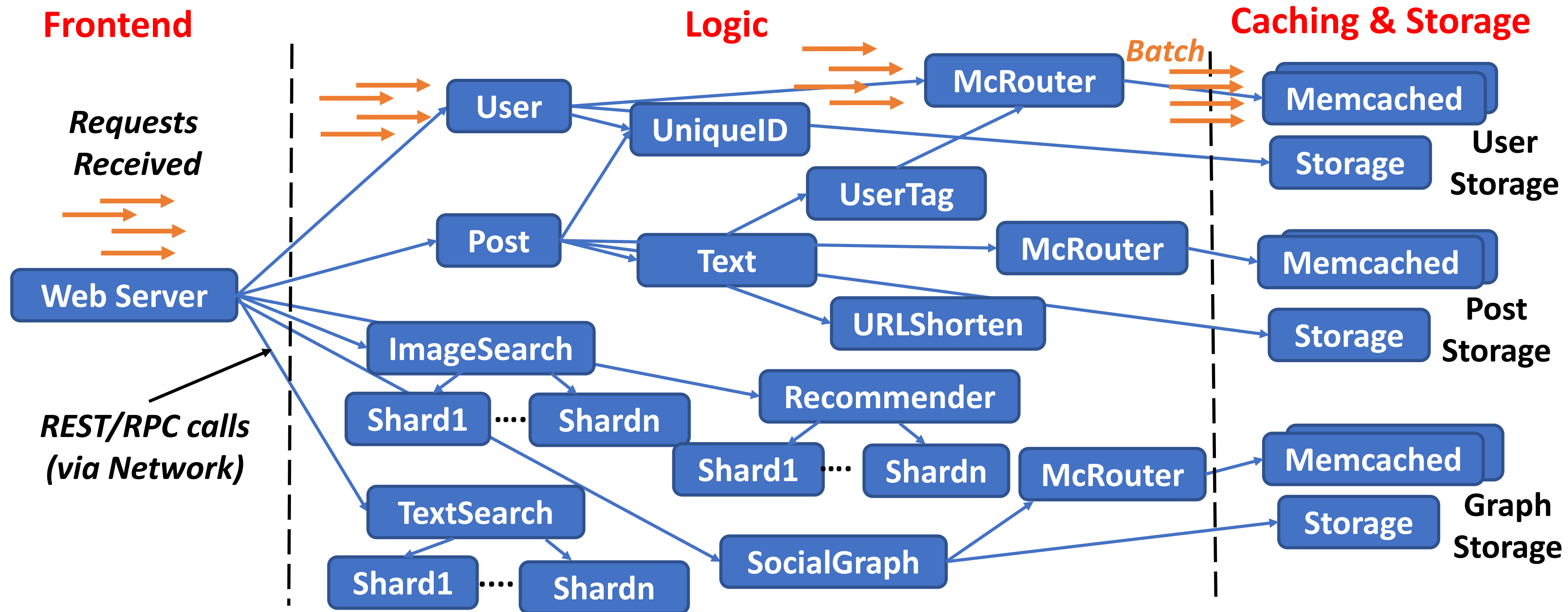
SIMT Control Efficiency
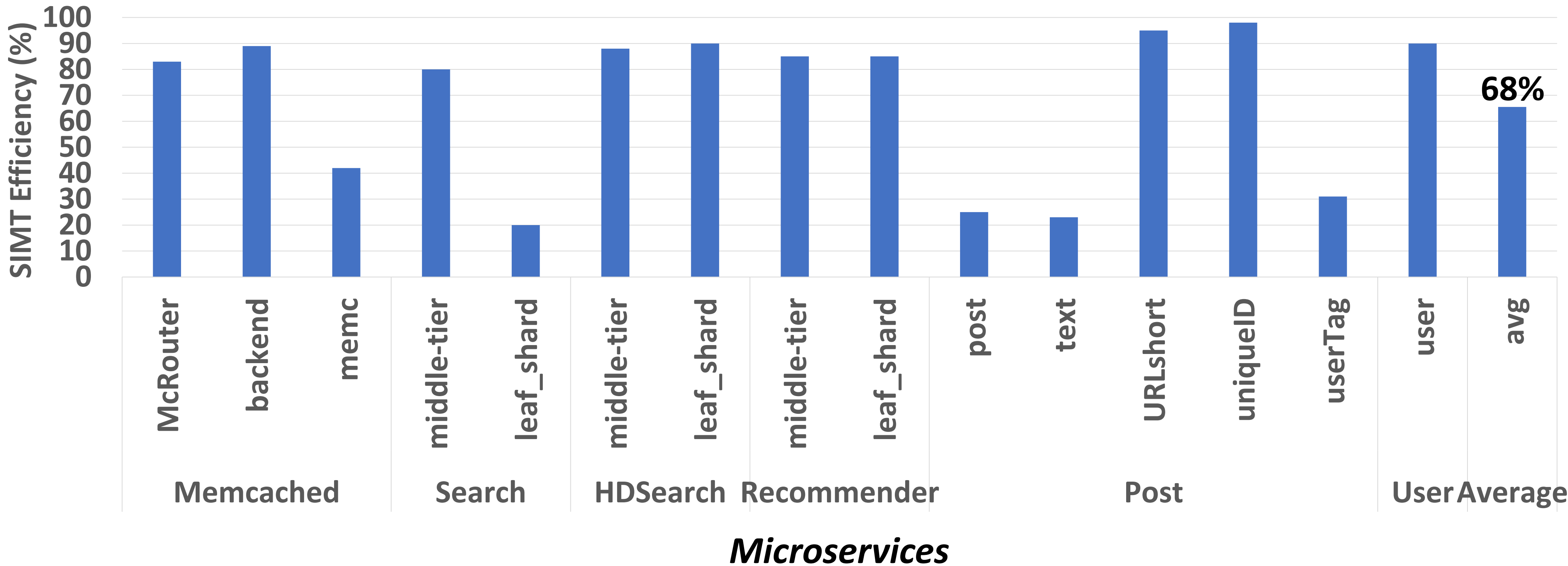
Notes: (1) Batch Size = 32, (2) System Calls are not included, (3) SIMT Eff = scalar-instructions / (batch-instructions * batch-size), (4) fine-grain locking are assumed

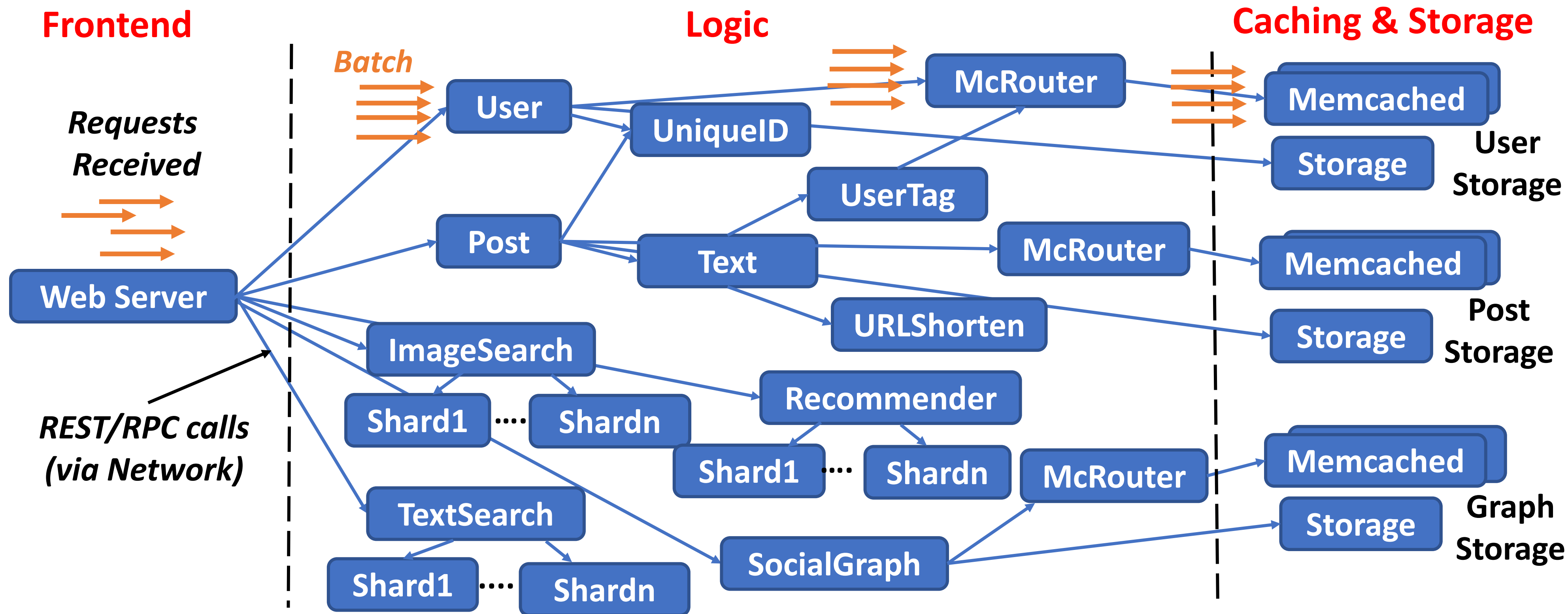# SIMT Control Efficiency (Optimized)
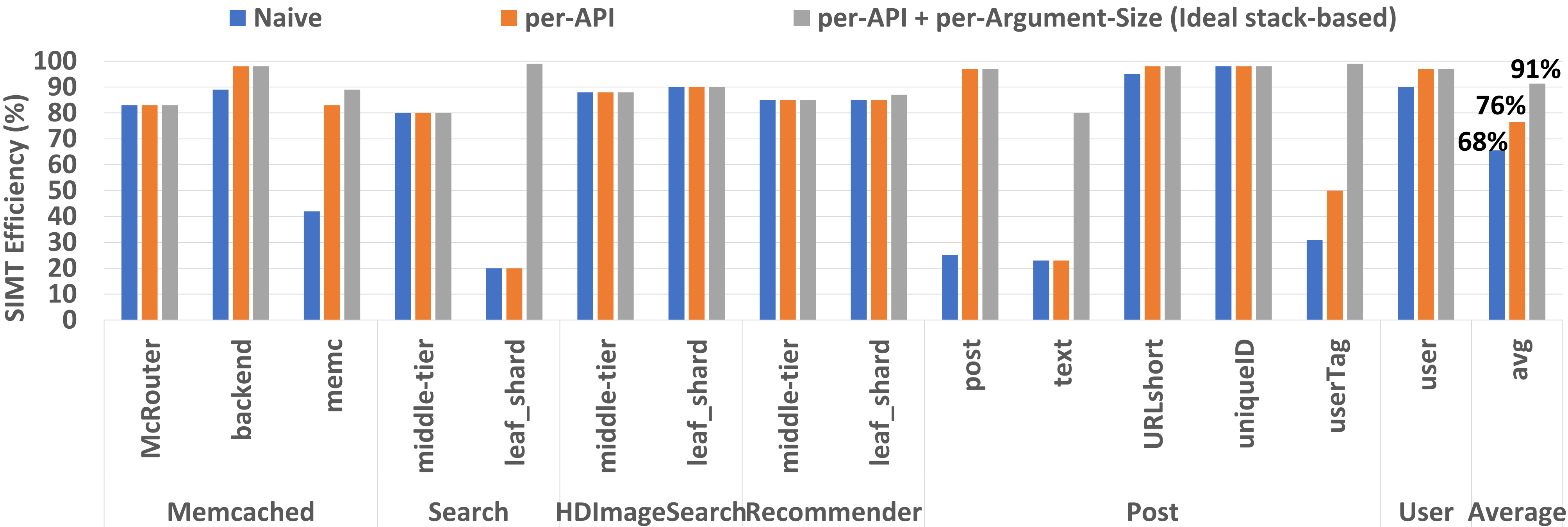
Current System: Selective Batching

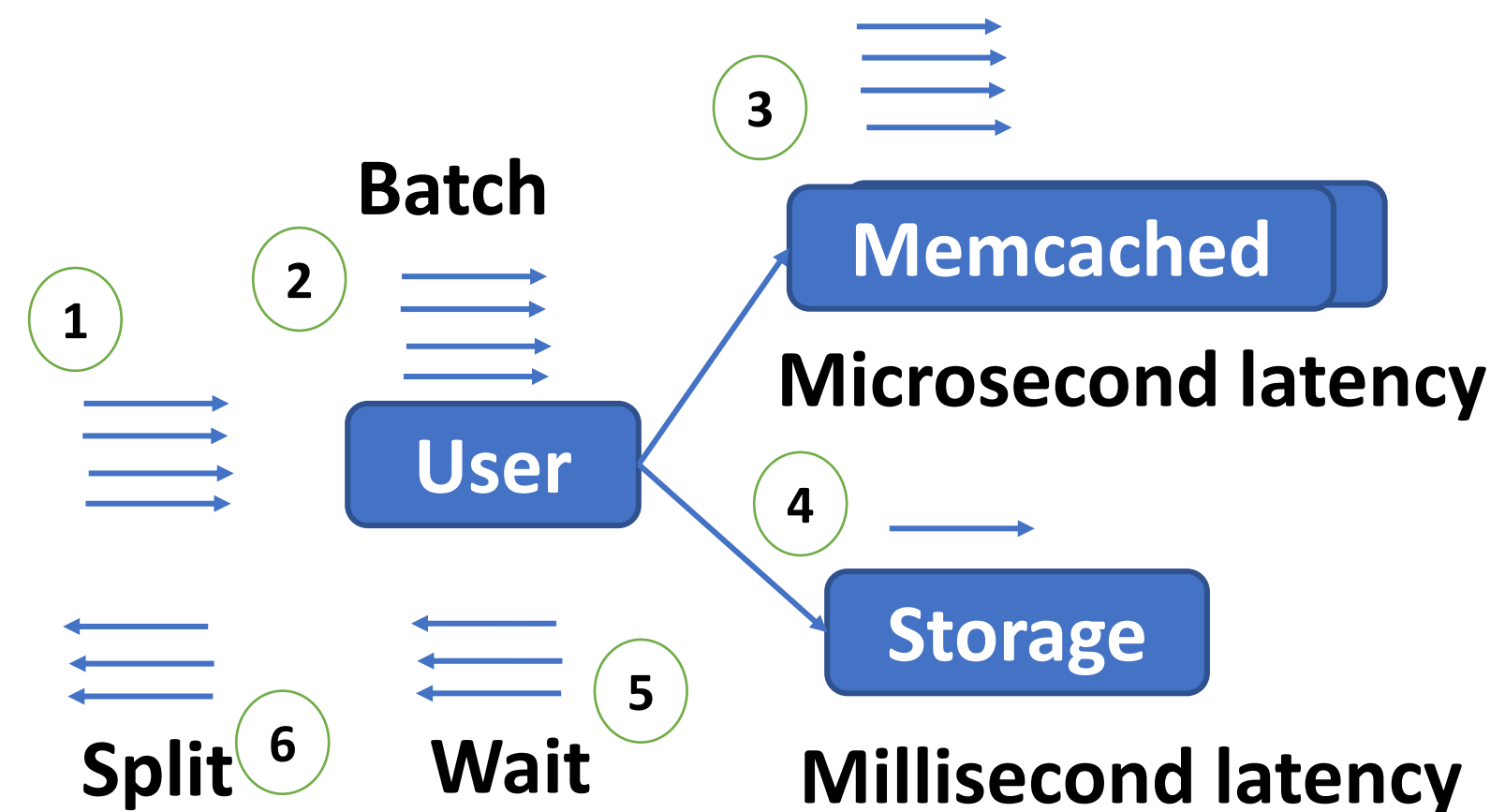Key Observation: Batching is heavily employed in the data center (DL inference, Memcached, ..)
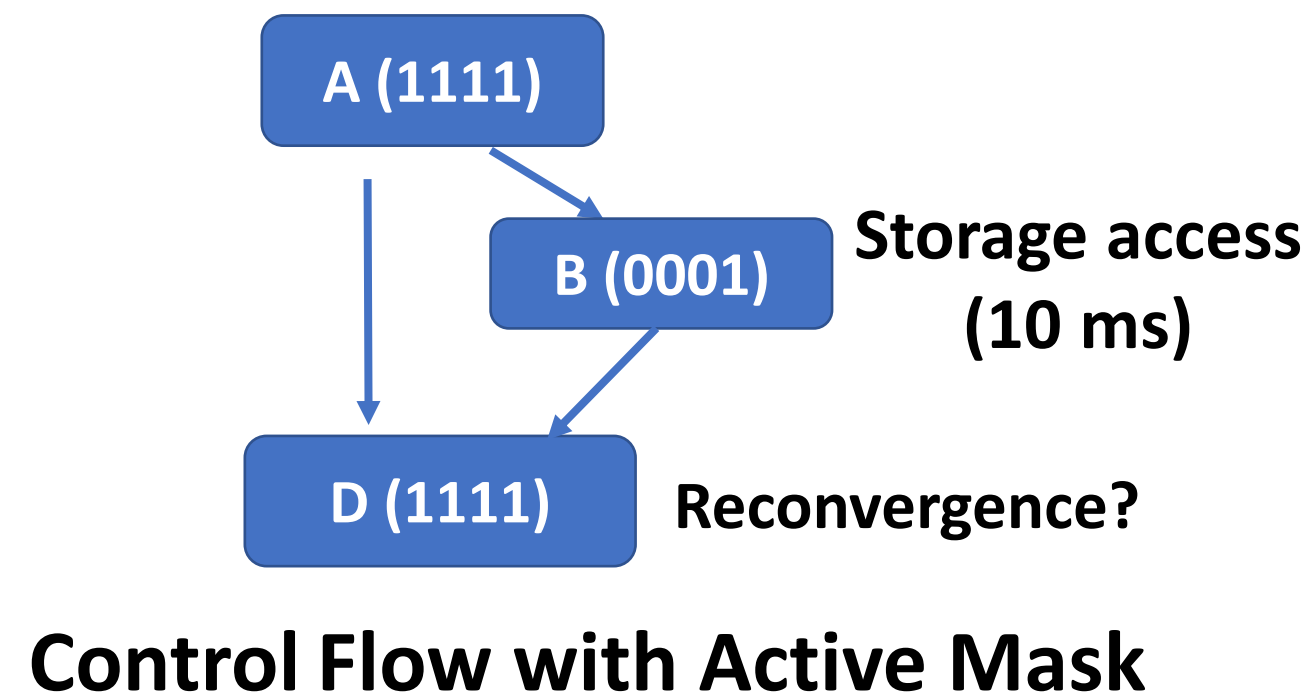
SIMR: System-Level RPU Batching

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

1. Procedure get_user(**int** userid)

2.   */* first try the cache */*

3.   data = memcached_fetch(**"userrow:"** + userid)

4.   **if** not data   */* SIMT Divergence*/*

5.   */* not found : request database */*

6.   data = db_select(**"SELECT * FROM users WHERE userid = ?"**, userid)

7.   */* then store in cache until next get */*

8.   memcached_add(**"userrow:"** + userid, data)

9.   end   */* SIMT Reconvergence Point*/*

10.   **return** data

A (1111)

B (0001) → **Storage access (10 ms)**

D (1111) → Reconvergence?

Control Flow with Active Mask

Batch

① ② ③ Memcached — **Microsecond latency**

User

④ Storage — **Millisecond latency**

Split ⑥ Wait ⑤

HW/SW Stack

CPU 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

GPU 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

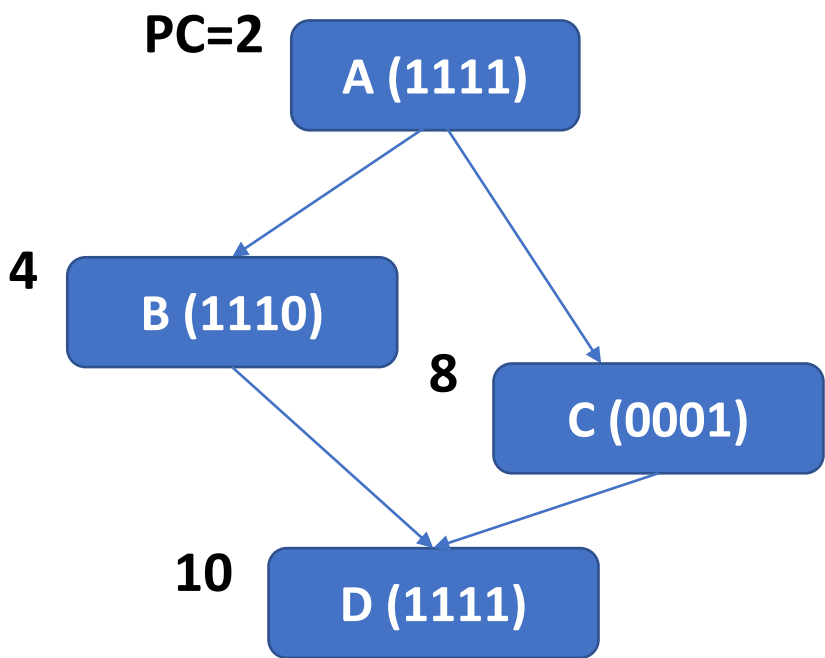| RPU SW Stack |
| --- |
| **Webservice (C++, PHP, …)** |
| **ARM/x86 compiler** |
| **Batch-aware HTTP server** |
| **Runtime/libs (pthread, cstdlib, ..)** |
| **OS (I/Os management)** |
| **RPU driver (VM/thread management)** |
| **RPU Hardware** |

RPU SW Stack

RPU HW

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

PC=2 A (1111)

4 B (1110)

8 C (0001)

10 D (1111)

Control Flow with Active Mask

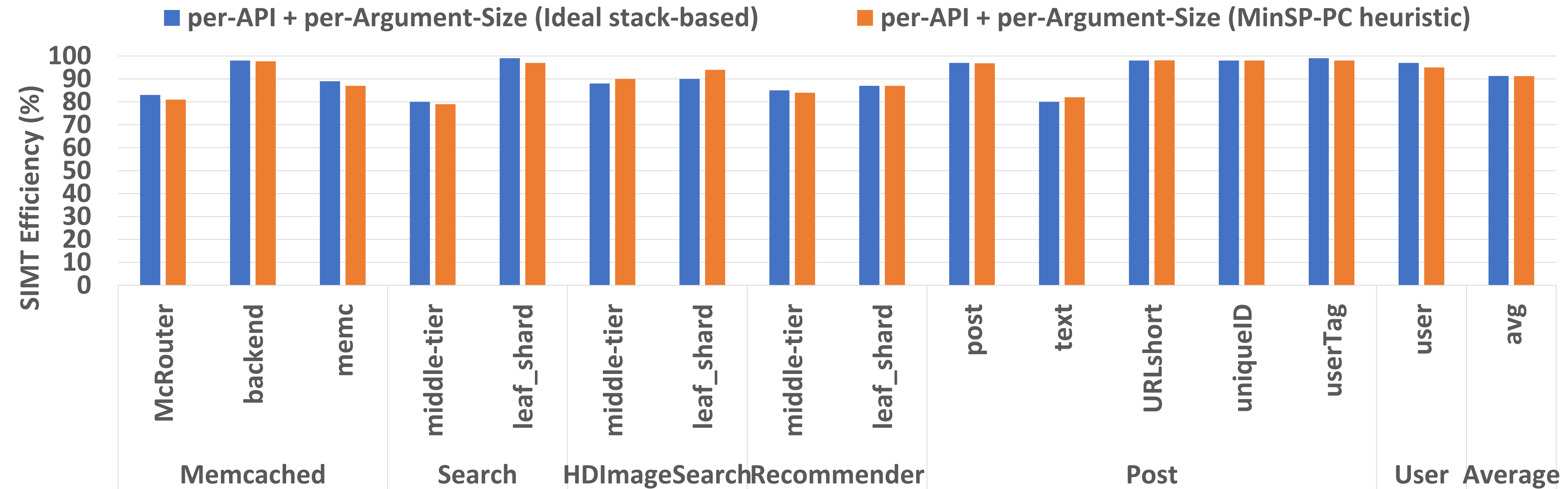| PC1	PC2	PC3	PC4	Current PC (min)	Active mask	Next PC (BP)
2	2	2	2	2	1111	4
4	4	4	4(F)	4	1111	6
6	6	6	8	6	1110	10
10	10	10	8	8	0001	10
10	10	10	10	10	1111	12

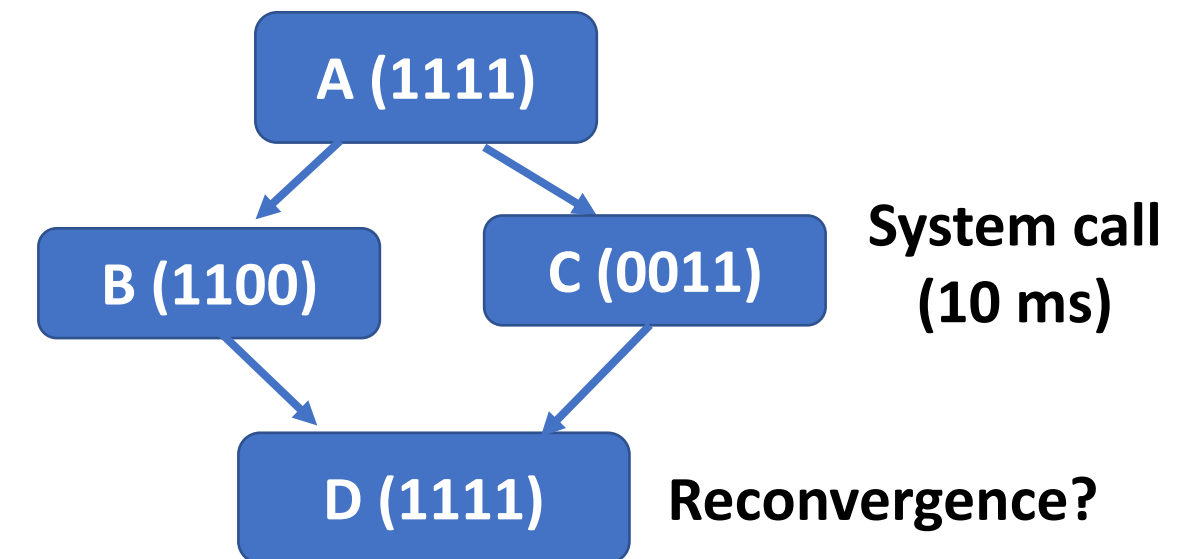MinPC selection policy

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
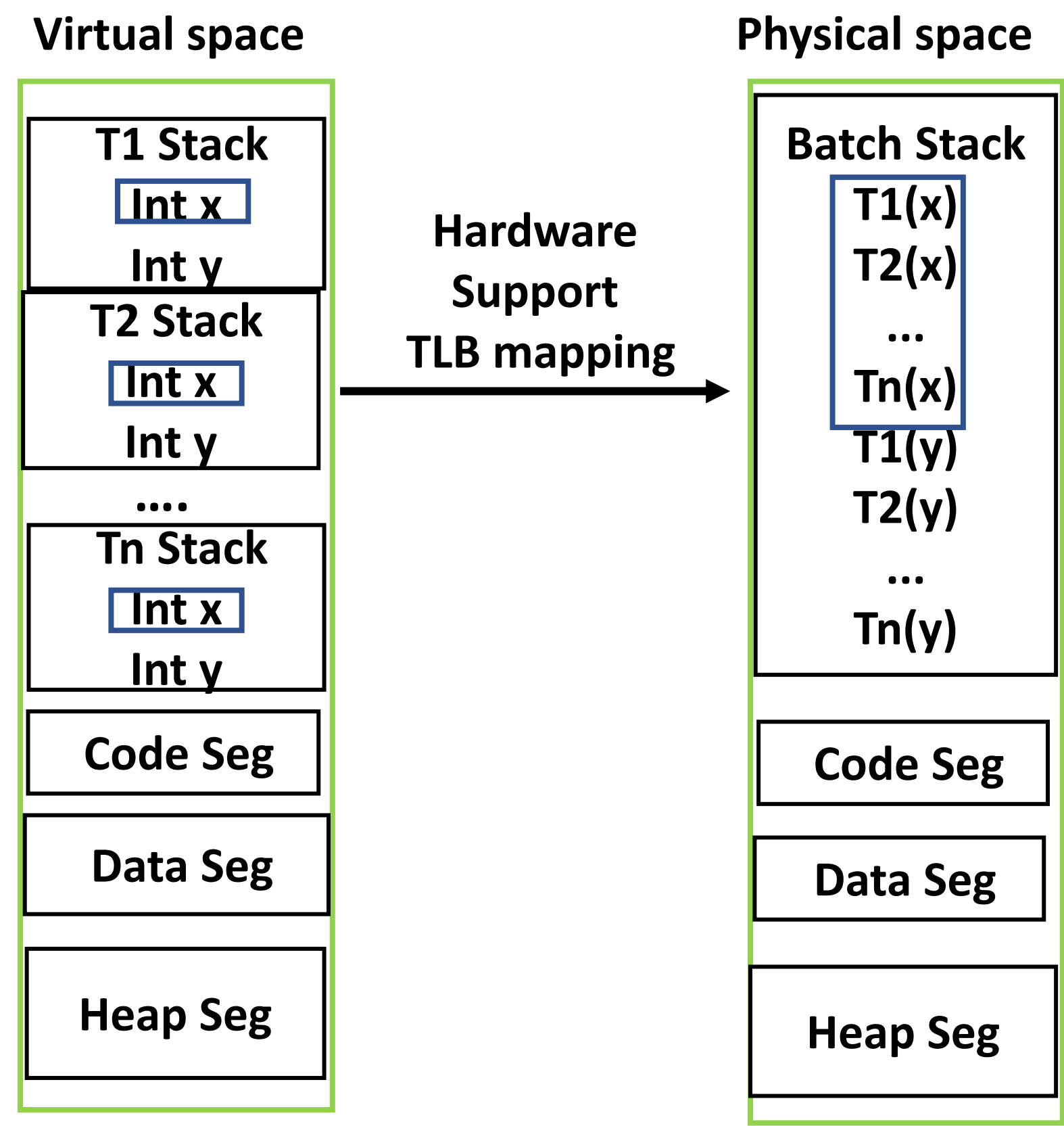 - Control divergence wit high latency branch

A (1111)

B (1100)   C (0011)   **System call
(10 ms)**

D (1111)   **Reconvergence?**

- Memory Divergence
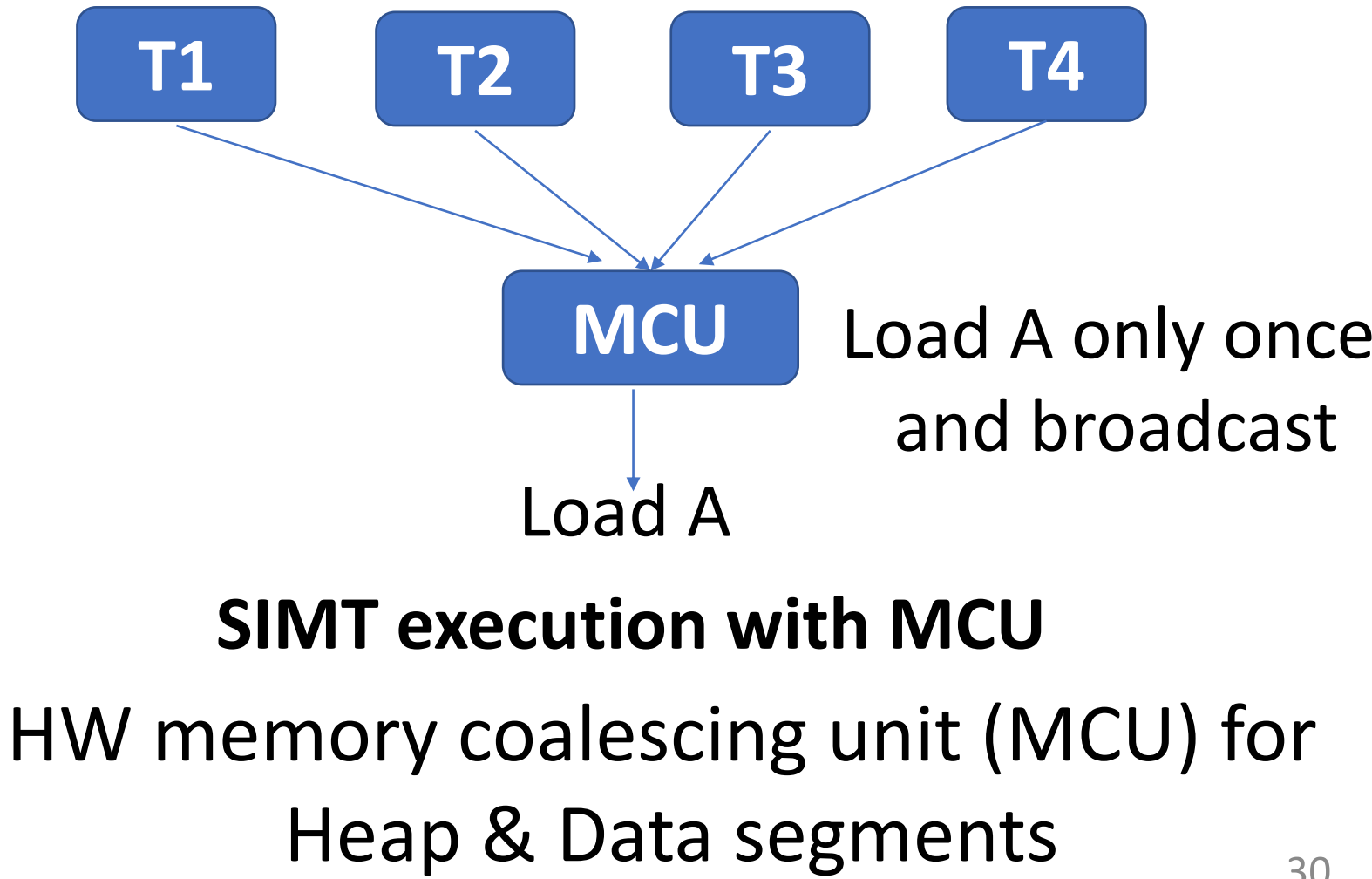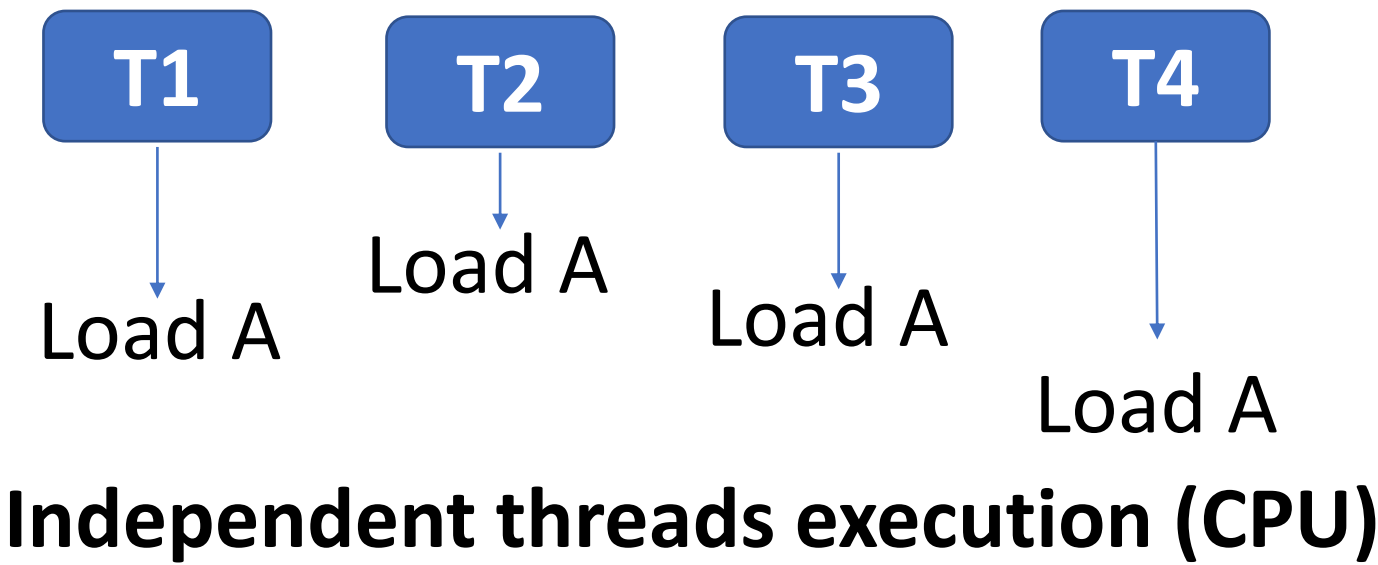 - Cache Contention & Bank Conflicts

Thrashing **L1 cache**

Many threads

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

Memory Coalescing Optimizations

Virtual space

T1 Stack
Int x
Int y

T2 Stack
Int x
Int y

....

Tn Stack
Int x
Int y

Code Seg

Data Seg

Heap Seg

Hardware Support TLB mapping →

Physical space

Batch Stack
T1(x)
T2(x)
...
Tn(x)
T1(y)
T2(y)
...
Tn(y)

Code Seg

Data Seg

Heap Seg

Stack segment coalescing with data interleaving

T1   **T2**   **T3**   **T4**

Load A   Load A   Load A

Load A

Independent threads execution (CPU)

T1   **T2**   **T3**   **T4**

MCU   Load A only once and broadcast

Load A

SIMT execution with MCU

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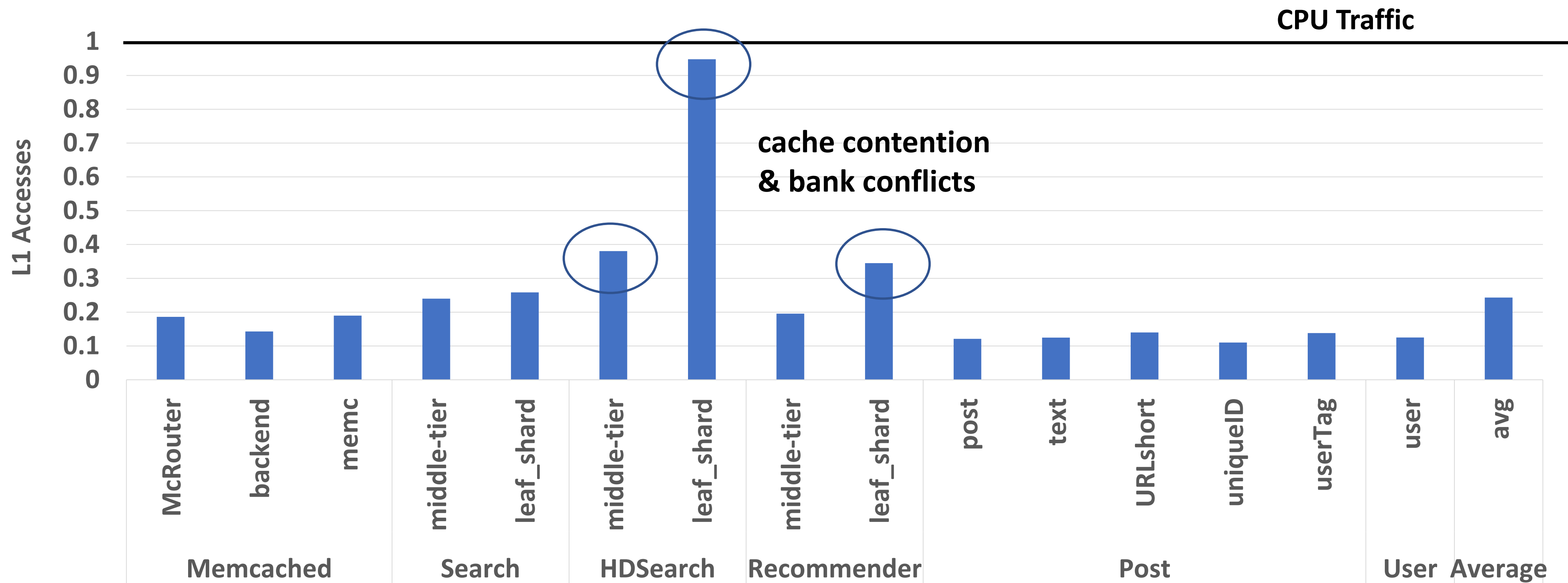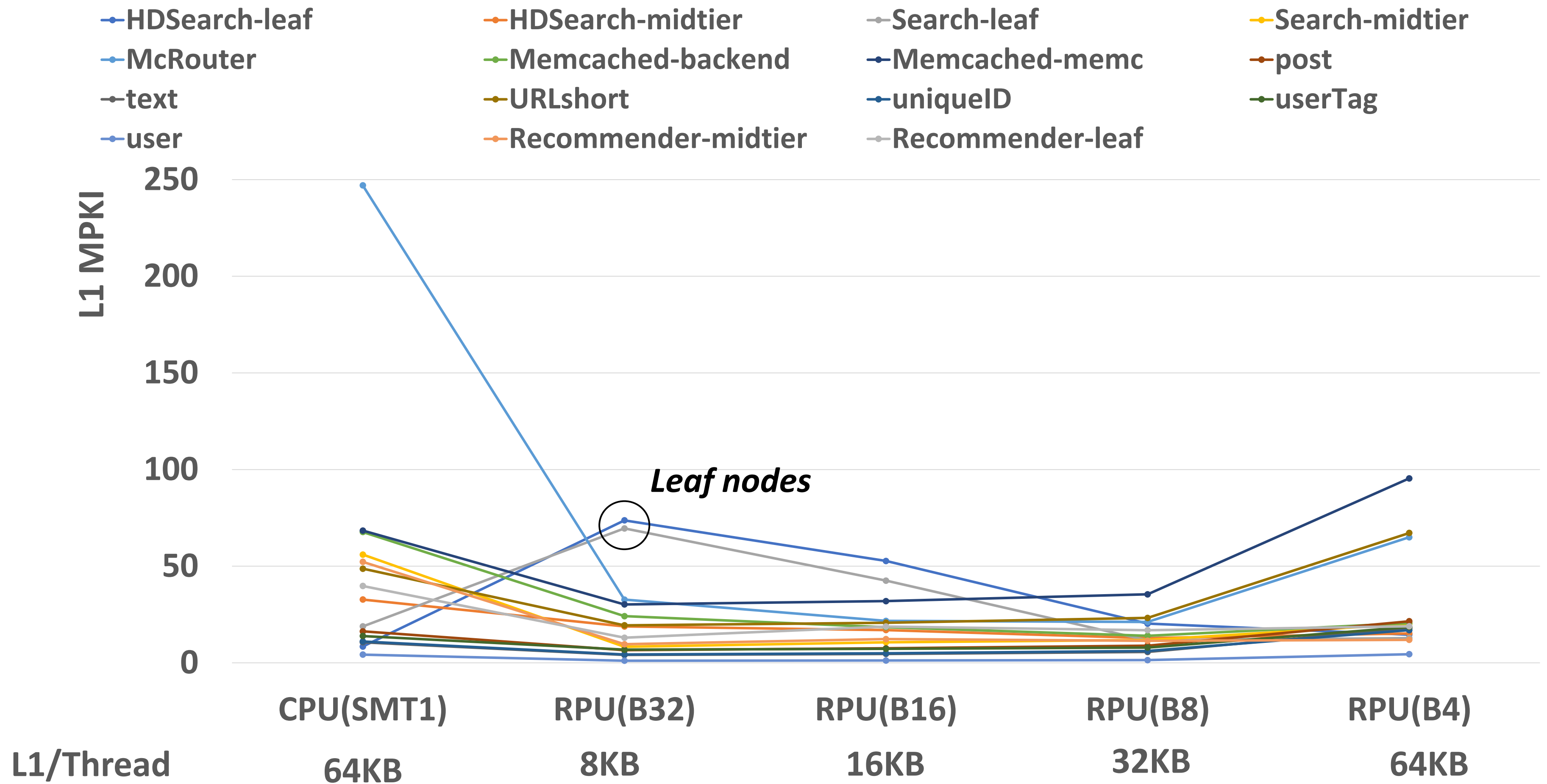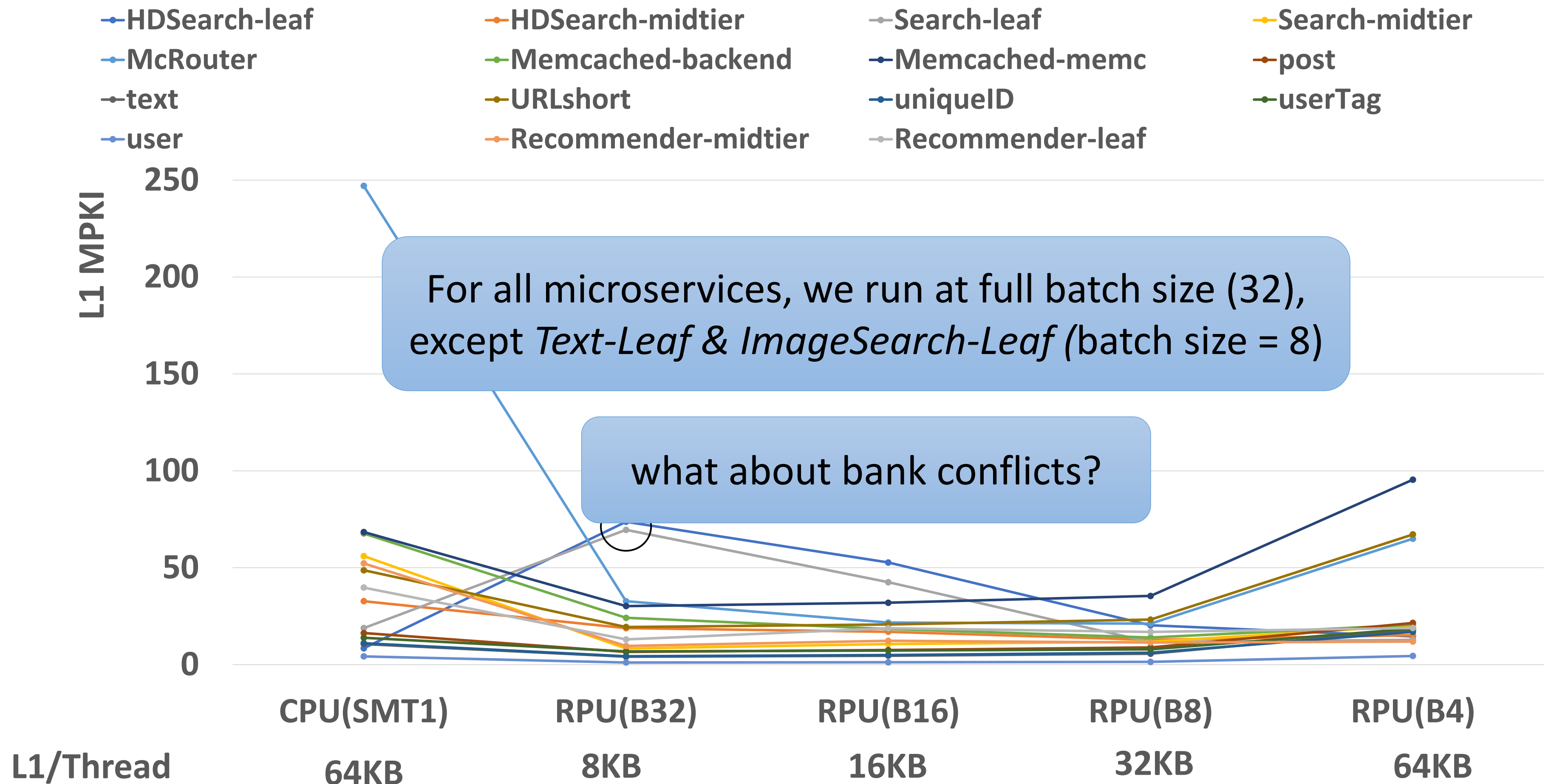
Traffic Reduction

→ **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)
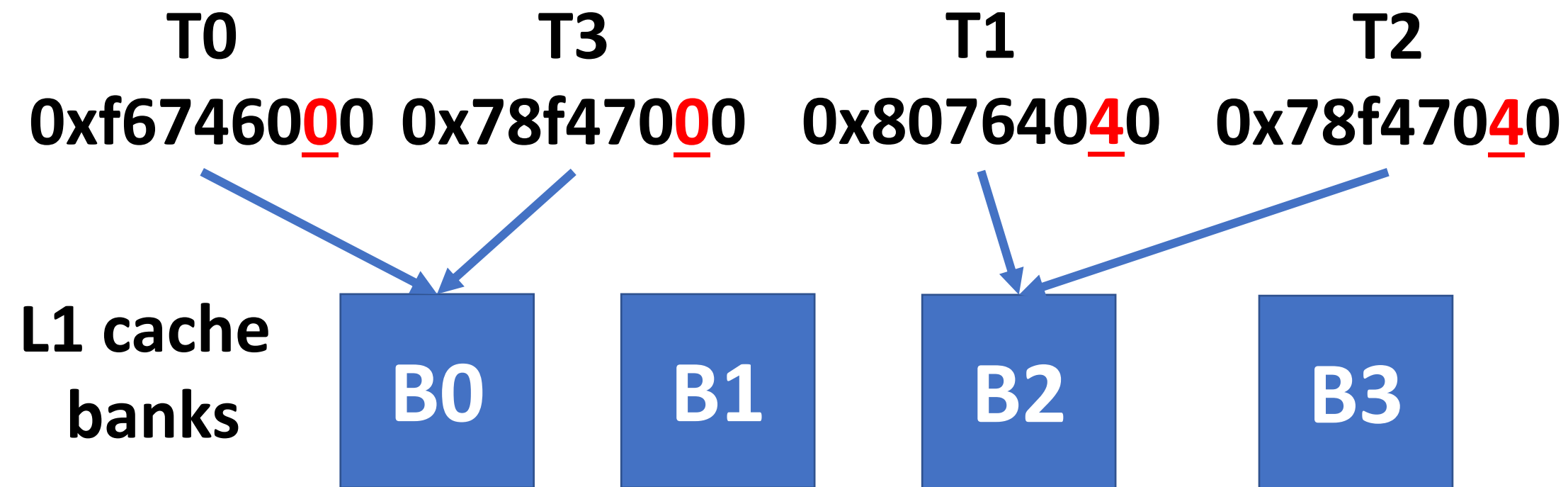
what about bank conflicts?

SIMT-Agnostic Memory Allocator

1. **Microservice ()**

2. *//Create a private temporary array in the*

3. *// heap segment*

4. int* temp = new int[n];

5. ....

6. for(int i=0; i<n; i++)

7. temp[i] = i;   *//Write to the temp*

8. ....

9. for(int i=0; i<n; i++)

10. sum += temp[i]; *//Read from the temp*

11. ....

Assume data are interleaved every 32B

temp **array address**

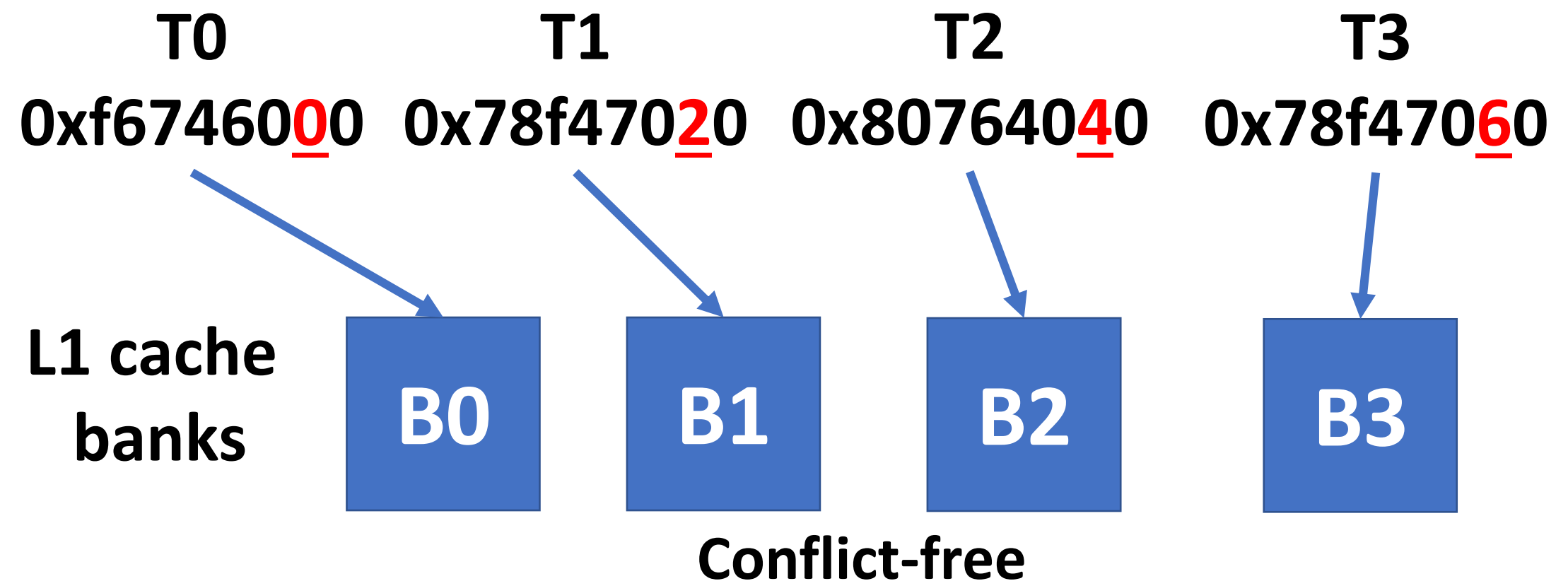| T0	T3	T1	T2
0xf6746000	0x78f47000	0x80764040	0x78f47040

L1 cache banks

| B0 | B1 | B2 | B3 |

Severe Bank Conflicts

C++ SIMT-Agnostic Memory Allocator

SIMT-Aware Memory Allocator

1. **Microservice ()**

2. *//Create a private temporary array in the*

3. *// heap segment*

4. int* temp = new int[n];

5.   ………..

6.   for(int i=0; i<n; i++)

7.   temp[i] = i;   *//Write to the temp*

8.   ………..

9. for(int i=0; i<n; i++)

10.   sum += temp[i]; *//Read from the temp*
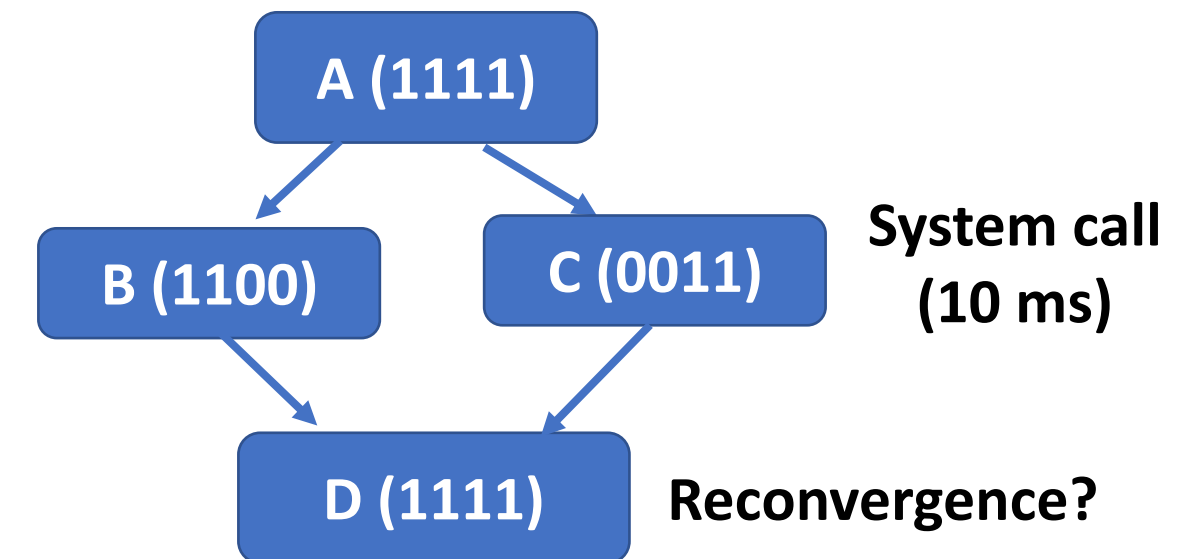
11. ………..

Assume data are interleaved every 32B

T0	T1	T2	T3
0xf6746000	0x78f47020	0x80764040	0x78f47060

L1 cache banks

B0   B1   B2   B3

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

A (1111)

B (1100)   C (0011)   **System call (10 ms)**

D (1111)   **Reconvergence?**

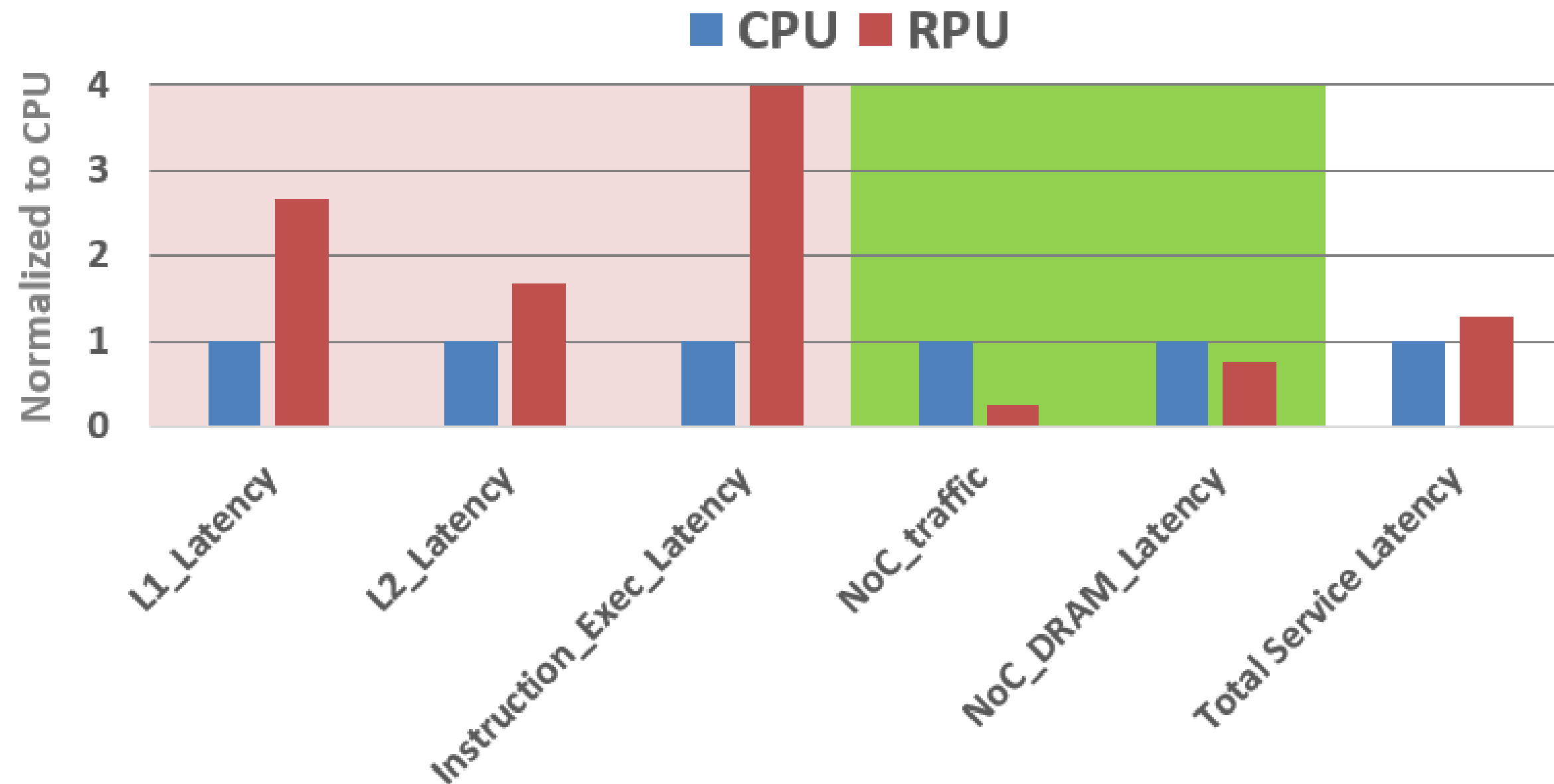- Memory Divergence
  - Cache Contention & Bank Conflicts

Thrashing   **L1 cache**

threads

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

Memory Latency Improvement

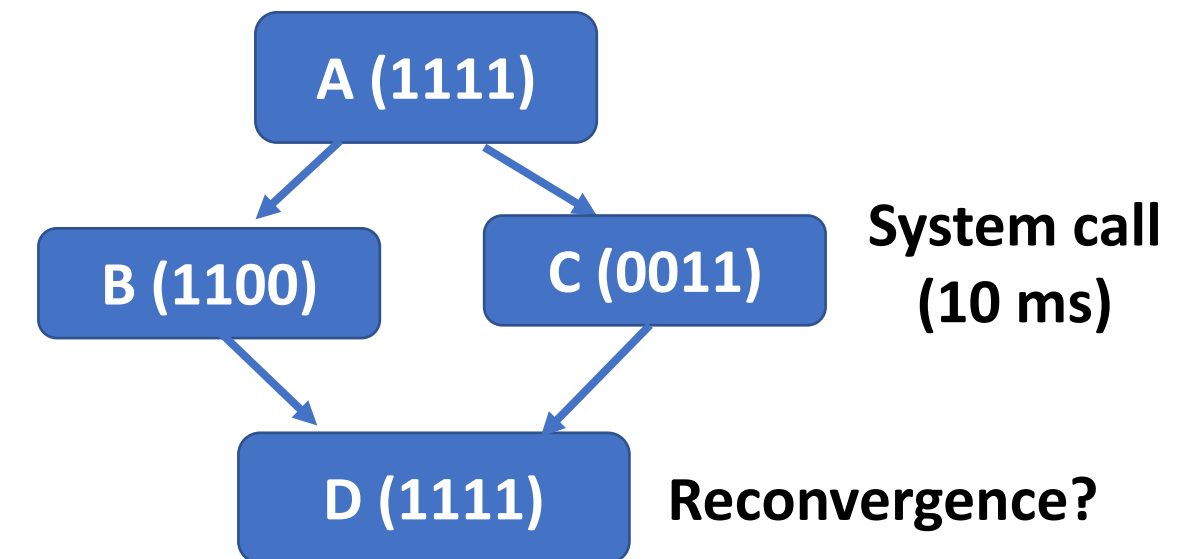**Metrics that contribute to total service latency**

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

Recall: 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

A (1111)

B (1100)   C (0011)   System call (10 ms)

D (1111)   Reconvergence?

- Memory Divergence
  - Cache Contention & Bank Conflicts

Thrashing   L1 cache

threads

- 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)

$$\frac{CPU\ Energy}{RPU\ Energy} = \frac{Execution\ Energy\ +\ Memory\ system\ Energy + Front\_OoO\ Energy +\ Static\ Energy}{Execution\ Energy +\ (1-r)\ (Memory\ system\ Energy) + \frac{1}{n\ *\ eff}\ [\ Front\_OoO\ Energy +\ r * Memory\ system\ Energy + Static\ Energy] + SIMT\_Overhead}$$

Amortized factors = 50-90%

data locality ratio =75%

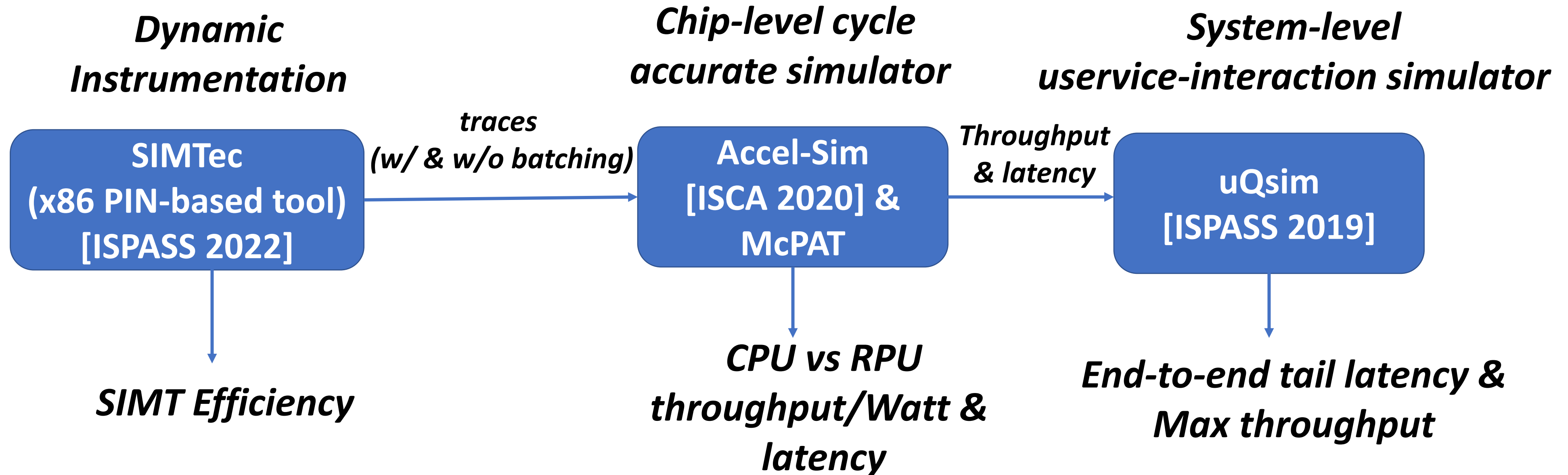batch size (n) = 8-32

SIMT Efficiency=92%

Larger L1/L2 MCUs Active mask etc.

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

40

CPU Dynamic Energy Breakdown

Experimental Setup

Dynamic Instrumentation

Chip-level cycle accurate simulator

System-level uservice-interaction simulator

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

traces (w/ & w/o batching) →

Accel-Sim [ISCA 2020] & McPAT

Throughput & latency →

uQsim [ISPASS 2019]

↓

SIMT Efficiency

↓

CPU vs RPU throughput/Watt & latency

↓

End-to-end tail latency & Max throughput
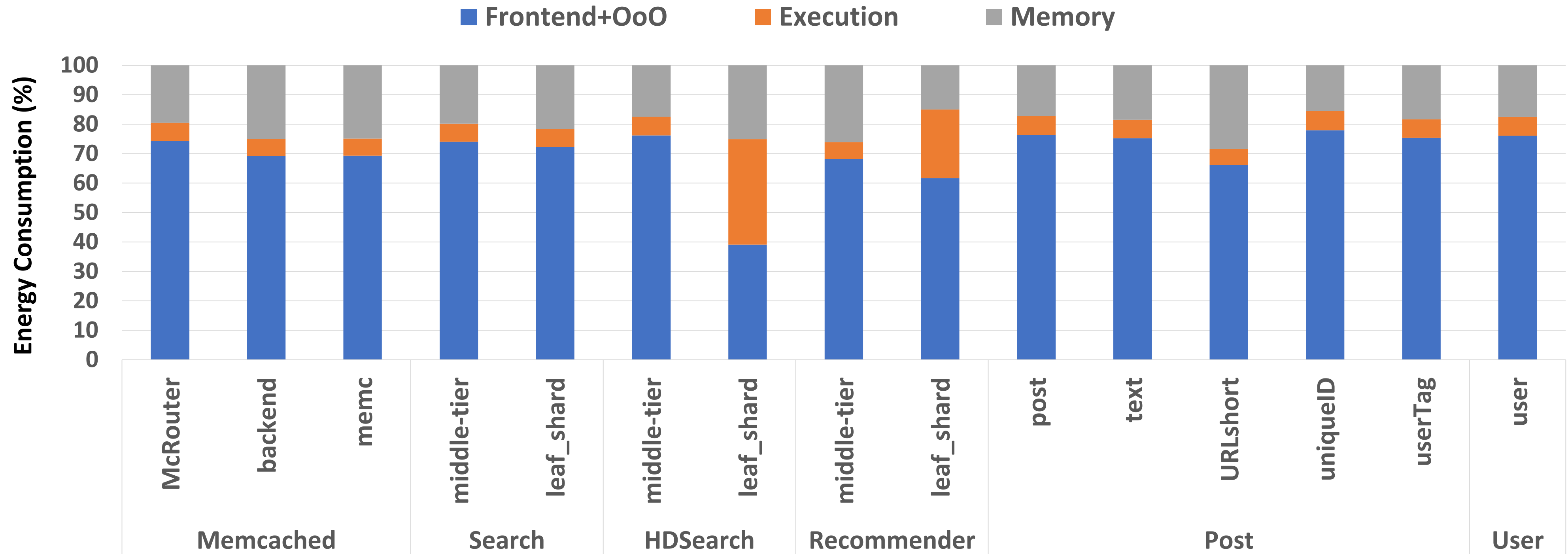
Workloads: Social Network Microservices

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

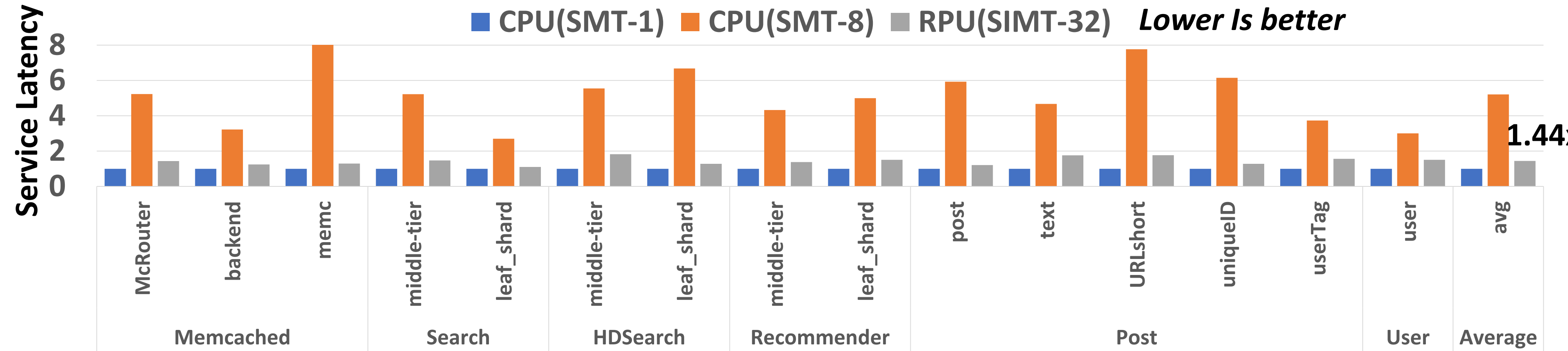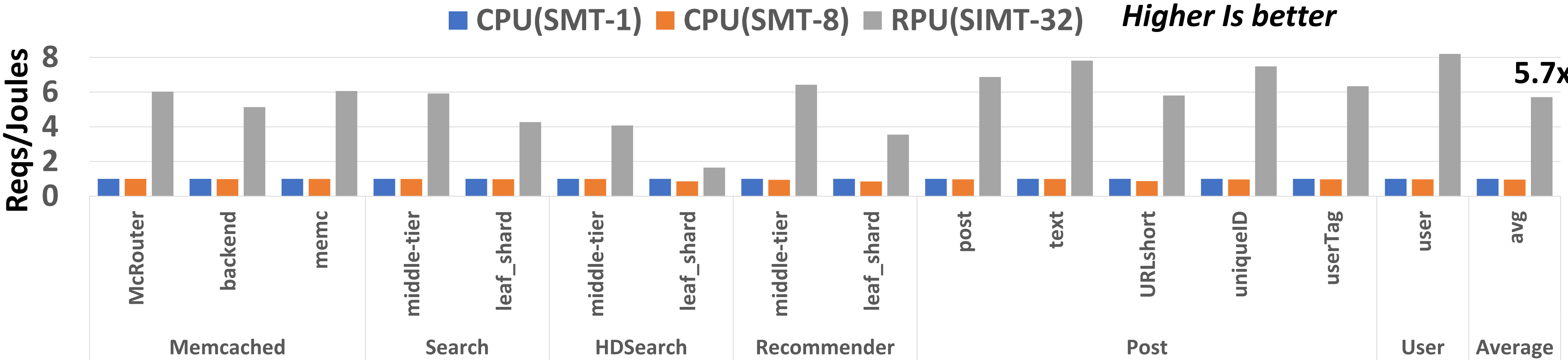| Metric	CPU	CPU SMT	RPU
Core Pipeline	8-wide 128-entry OoO	8-wide 128-entry OoO	8-wide 128-entry OoO
Freq	2.5 GHZ	2.5 GHZ	2.5 GHZ
#Cores	98	80	20
Threads/core	1	SMT-8	SIMT-32 (1 batch)
Total Threads	98	640	640
#Lanes	1	1	8
Max IPC/core	8	8	64 (issue x lanes)
ALU/Bra Exec Lat	1-cycle	1-cycle	4-cycle
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.

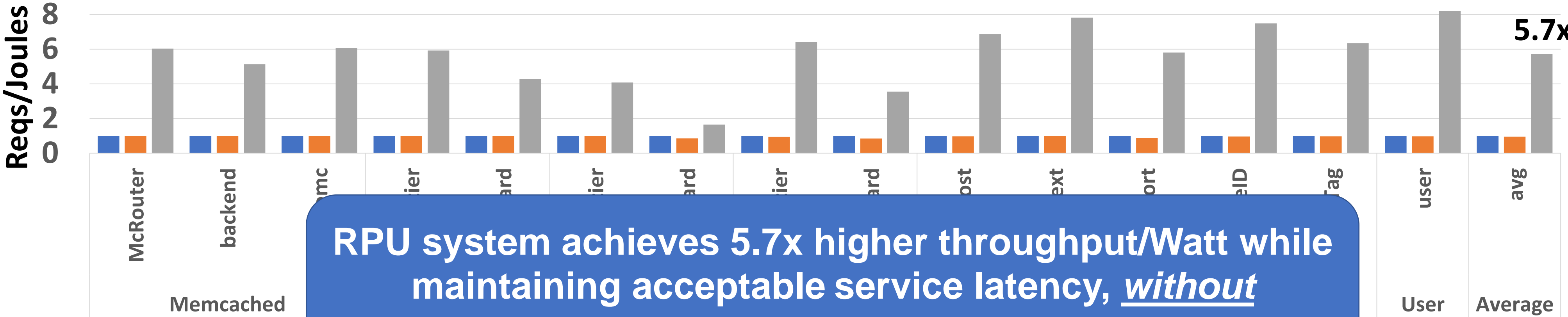Table V: Per-component area and peak power estimates

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)

Legend: CPU(SMT-1), CPU(SMT-8), RPU(SIMT-32) — *Higher Is better*

Reqs/Joules chart with categories: McRouter, backend, memc (Memcached); middle-tier, leaf_shard (Search); middle-tier, leaf_shard (HDSearch); middle-tier, leaf_shard (Recommender); post, text, URLshort, uniqueID, userTag (Post); user (User); avg (Average). Average RPU efficiency: 5.7x

Legend: CPU(SMT-1), CPU(SMT-8), RPU(SIMT-32) — *Lower Is better*

Service Latency chart with same categories. Average: 1.44x

45

Efficiency and Service Latency Results (Simulation)

Legend: ■ CPU(SMT-1) ■ CPU(SMT-8) ■ RPU(SIMT-32)   *Higher Is better*

Reqs/Joules (top chart, y-axis: 0–8)

Categories: McRouter, backend, memc (Memcached); middle-tier, leaf_shard (Search); middle-tier, leaf_shard (HDSearch); middle-tier, leaf_shard (Recommender); post, text, URLshort, uniqueID, userTag (Post); user (User); avg (Average)

5.7x

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

Service Latency (bottom chart, y-axis: 0–8)

Categories: McRouter, backend, memc (Memcached); middle-tier, leaf_shard (Search); middle-tier, leaf_shard (HDSearch); middle-tier, leaf_shard (Recommender); post, text, URLshort, uniqueID, userTag (Post); user (User); avg (Average)

1.44x

46
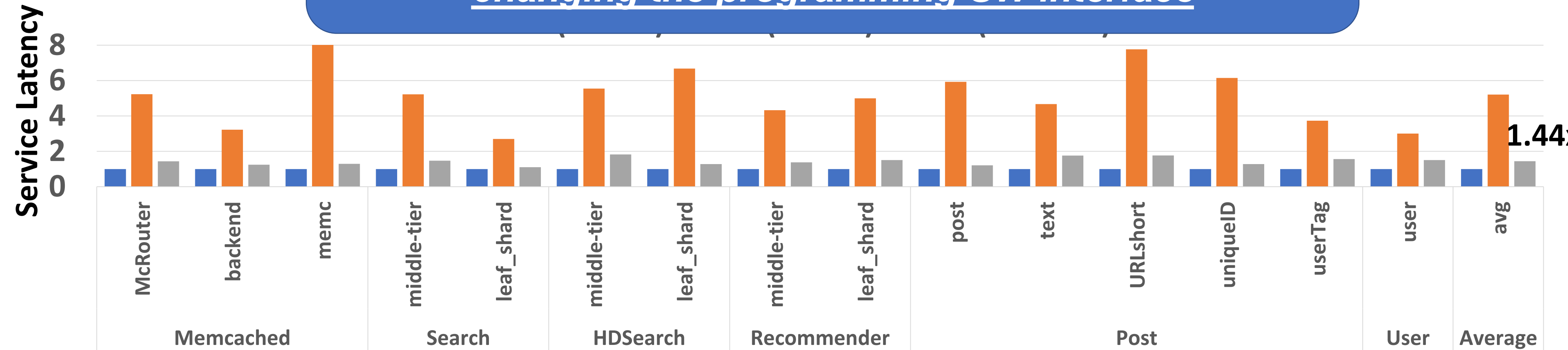
System-Level Results (uQsim Simulator)
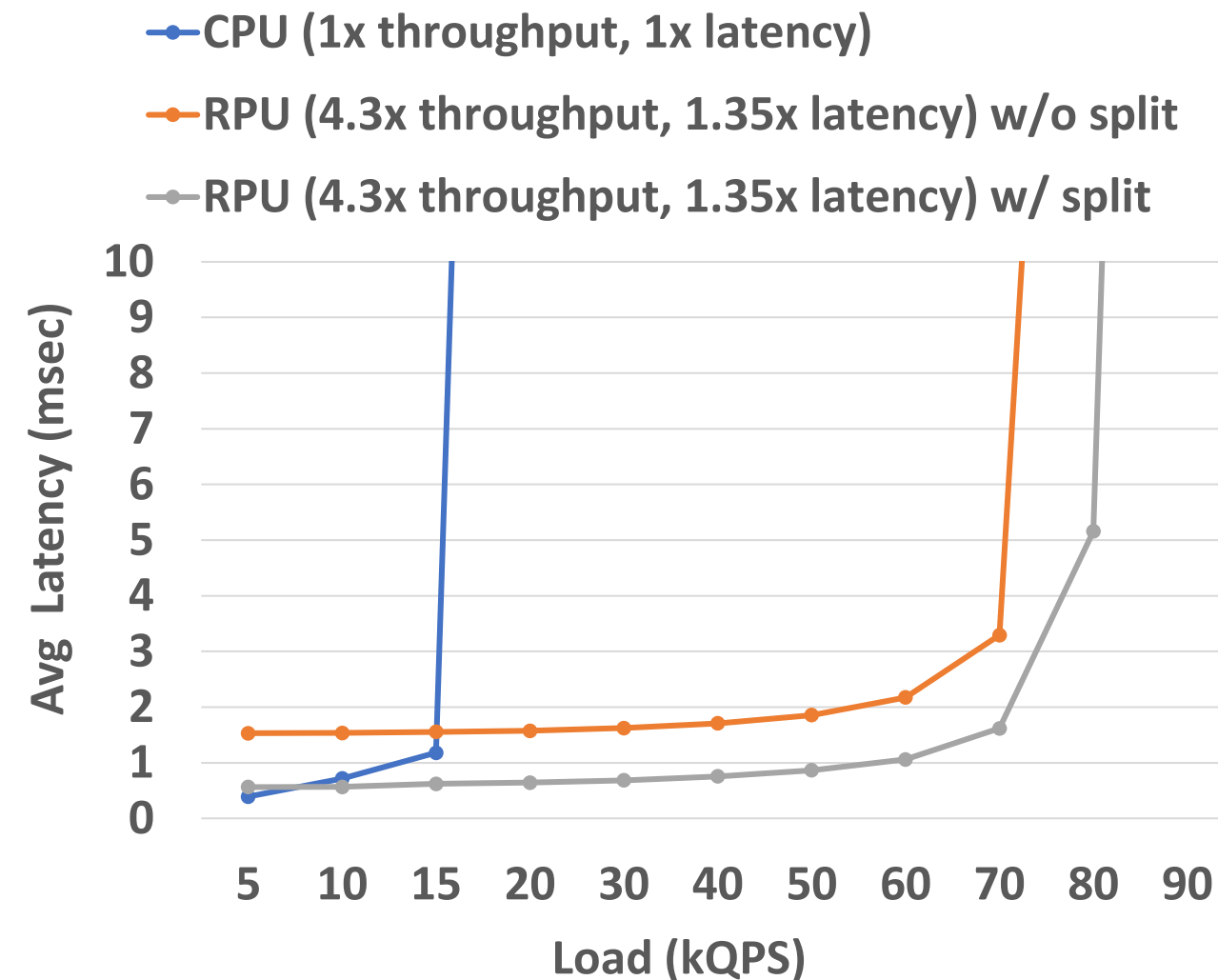
Average latency
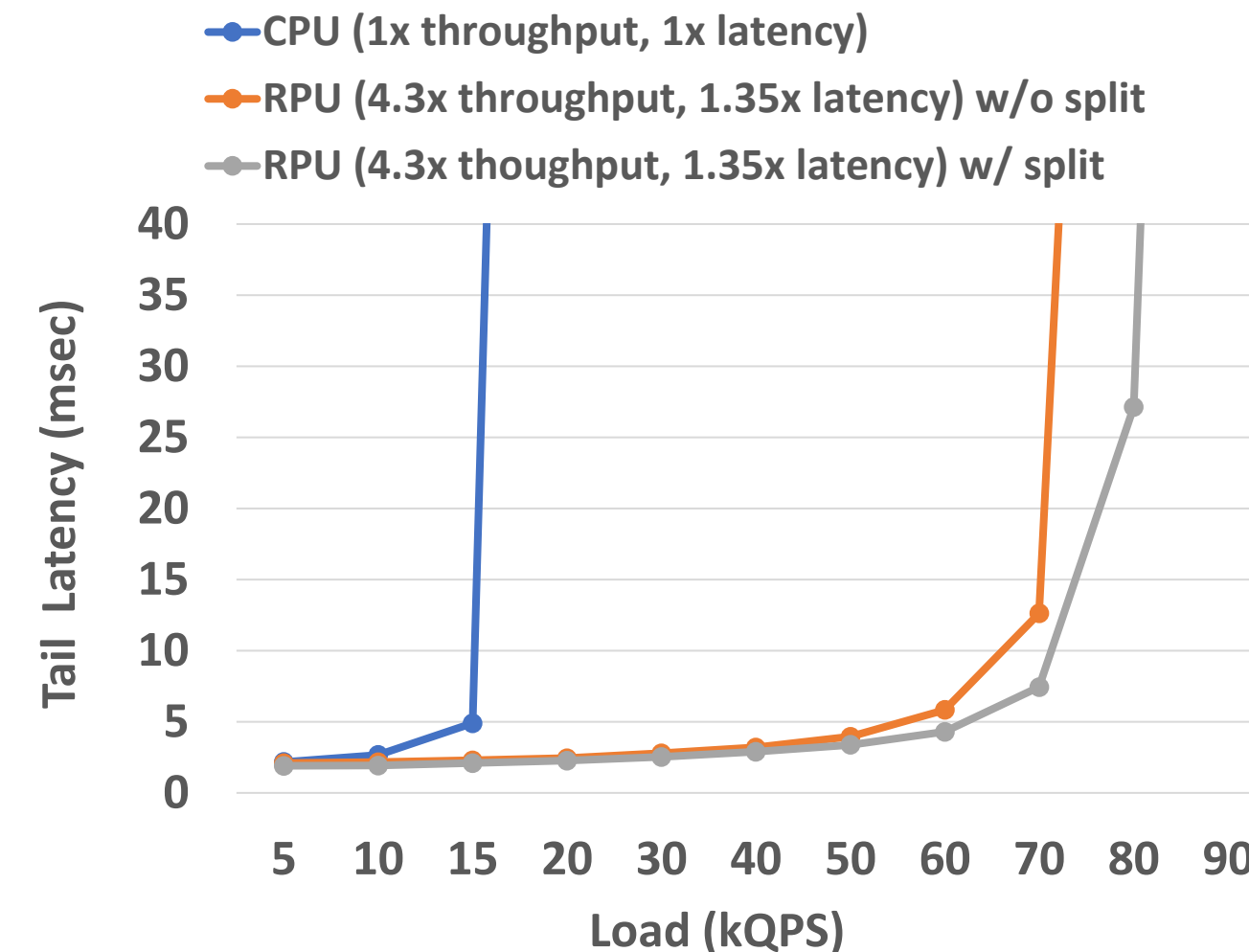
99% tail latency

→ 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

47

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)
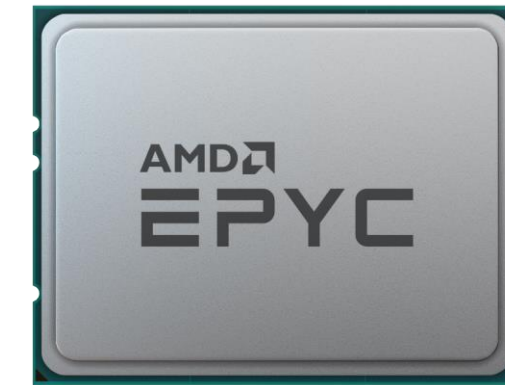
Data level parallelism (DLP)
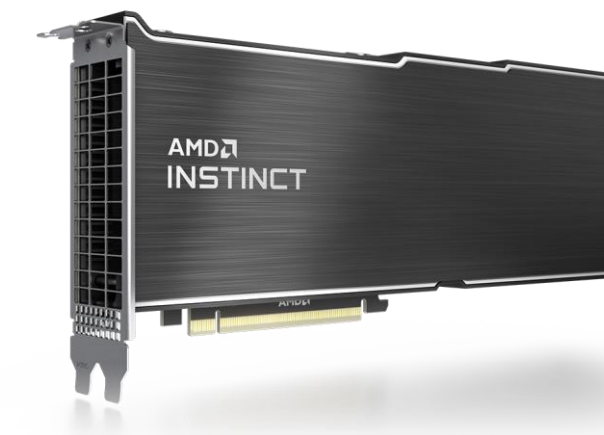
Request level parallelism (RLP)
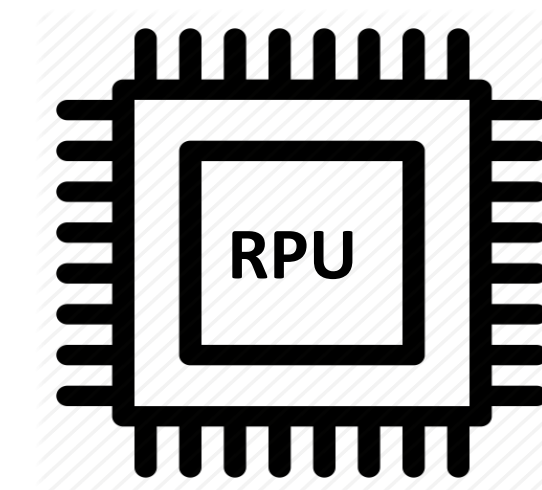
Back-Up Slides

Motivation & Background Slides

Energy Efficiency Crisis

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

9,000 terawatt hours (TWh)

ENERGY FORECAST
Widely cited forecasts suggest that the total electricity demand of information and communications technology (ICT) will accelerate in the 2020s, and that data centres will take a larger slice.

20.9% of projected electricity demand

- Networks (wireless and wired)
- Production of ICT
- Consumer devices (televisions, computers, mobile phones)
- Data centres

Google's Electricity Use, 2011 to 2019

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

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

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)

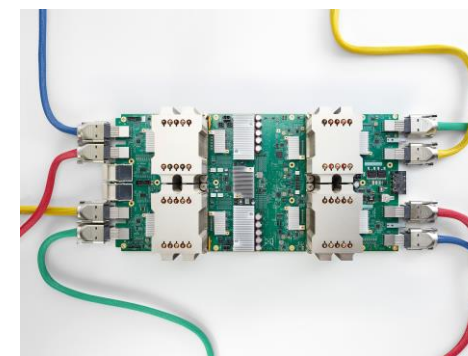Potential Server Spending Over The Next Ten Years

Solution: Hardware/Software Co-Design (Accelerators)

Software
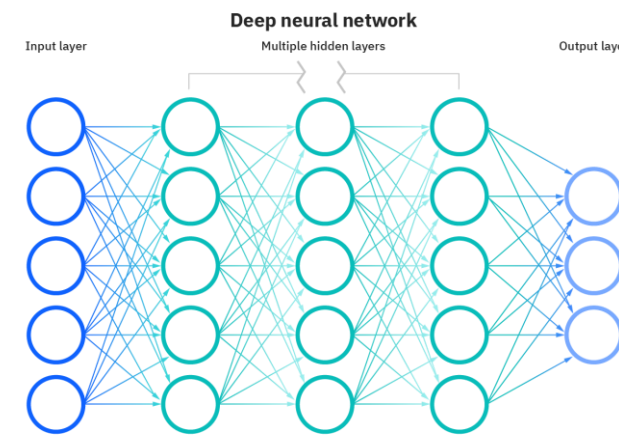
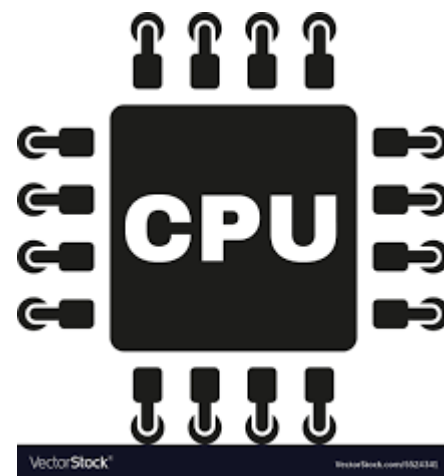*Deep Learning*
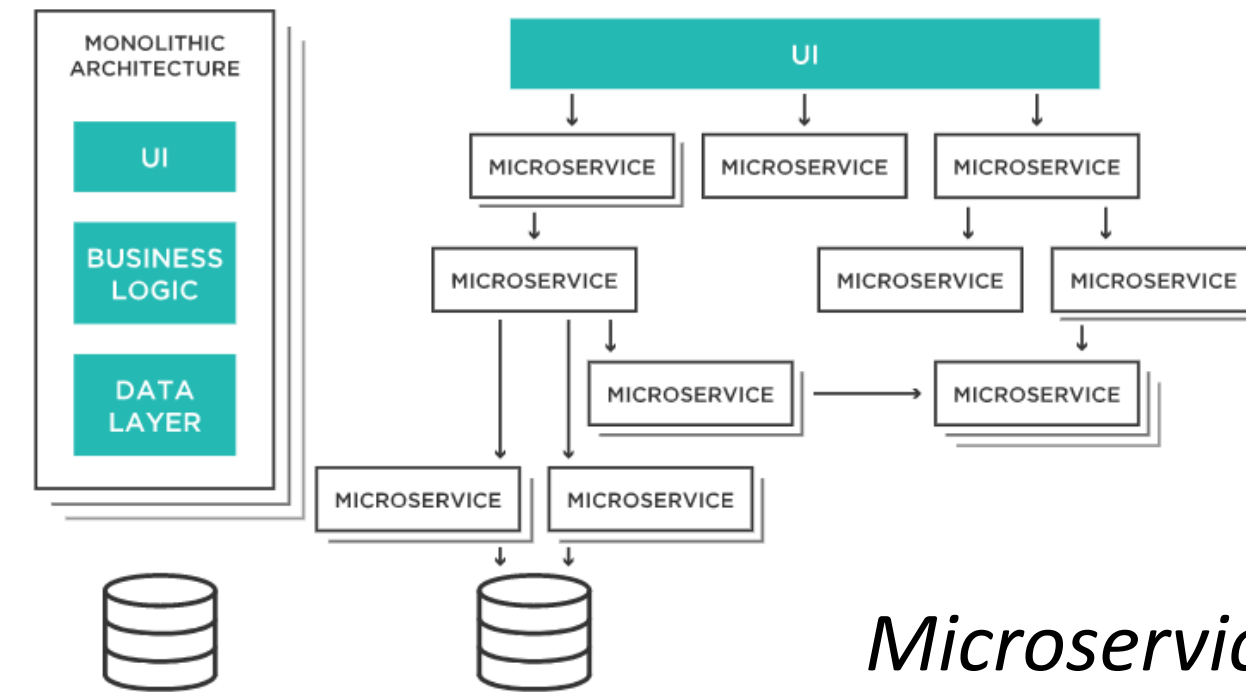
Microservices

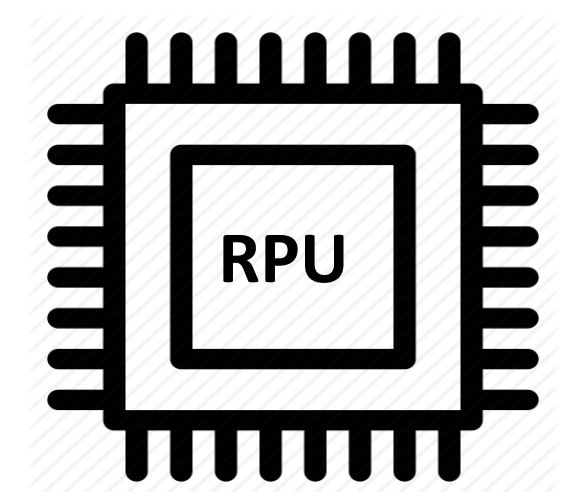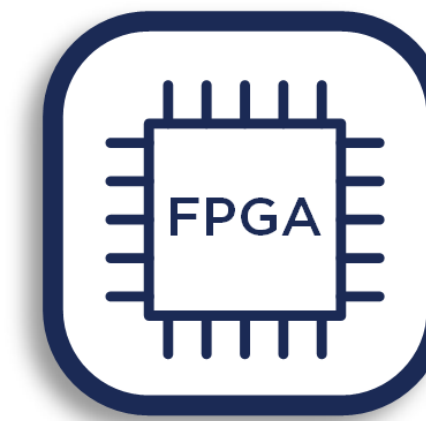
Hardware
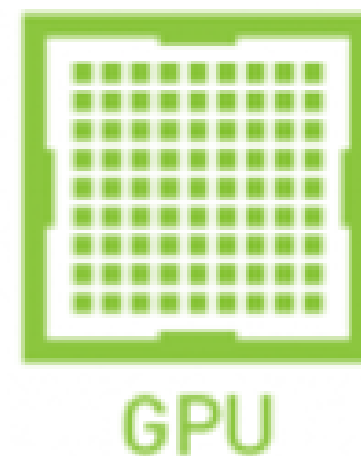
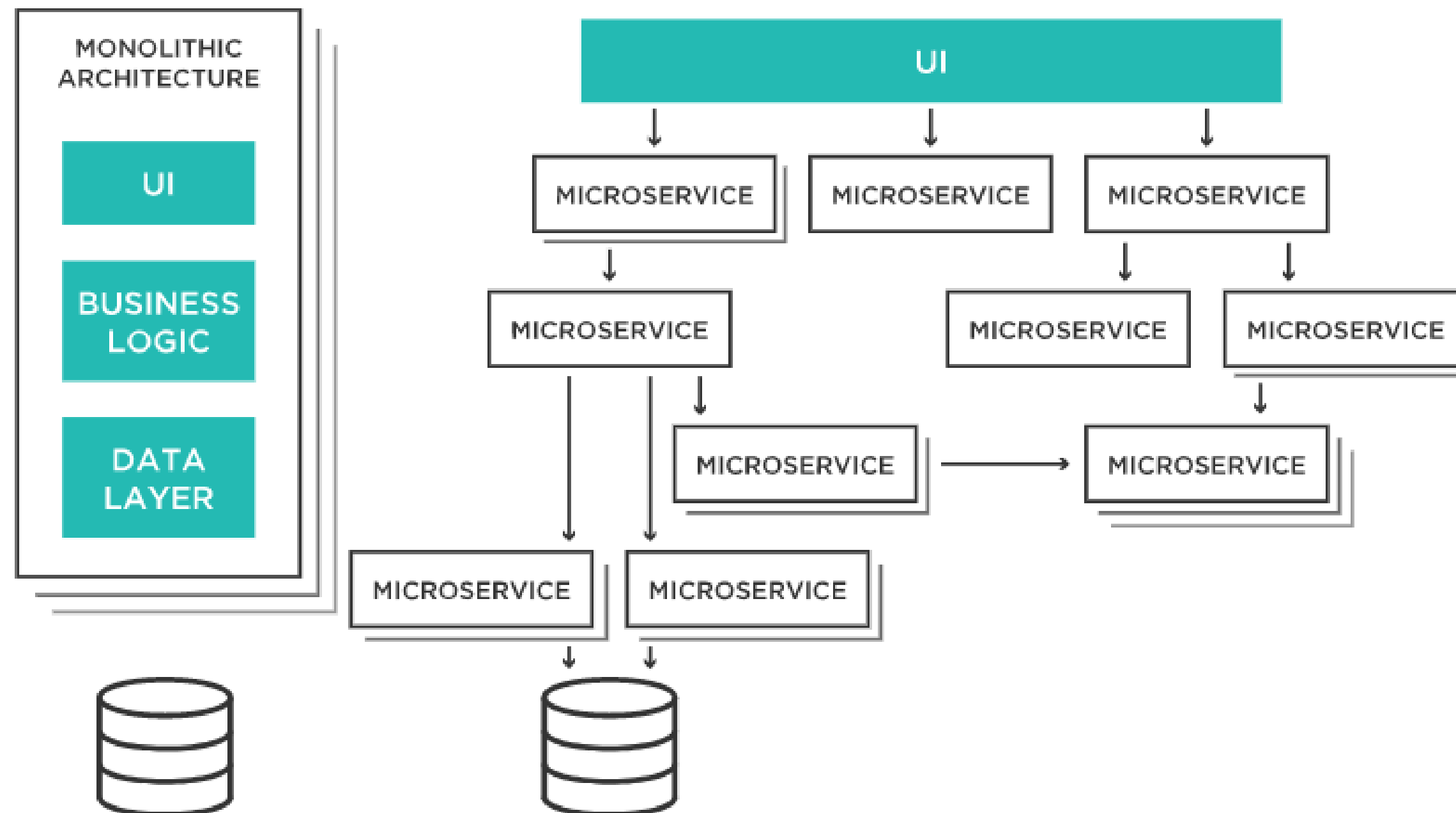

CPU **GPU** **TPU** **VCU** **FPGA** **RPU**

Accelerators

Microservices Architecture

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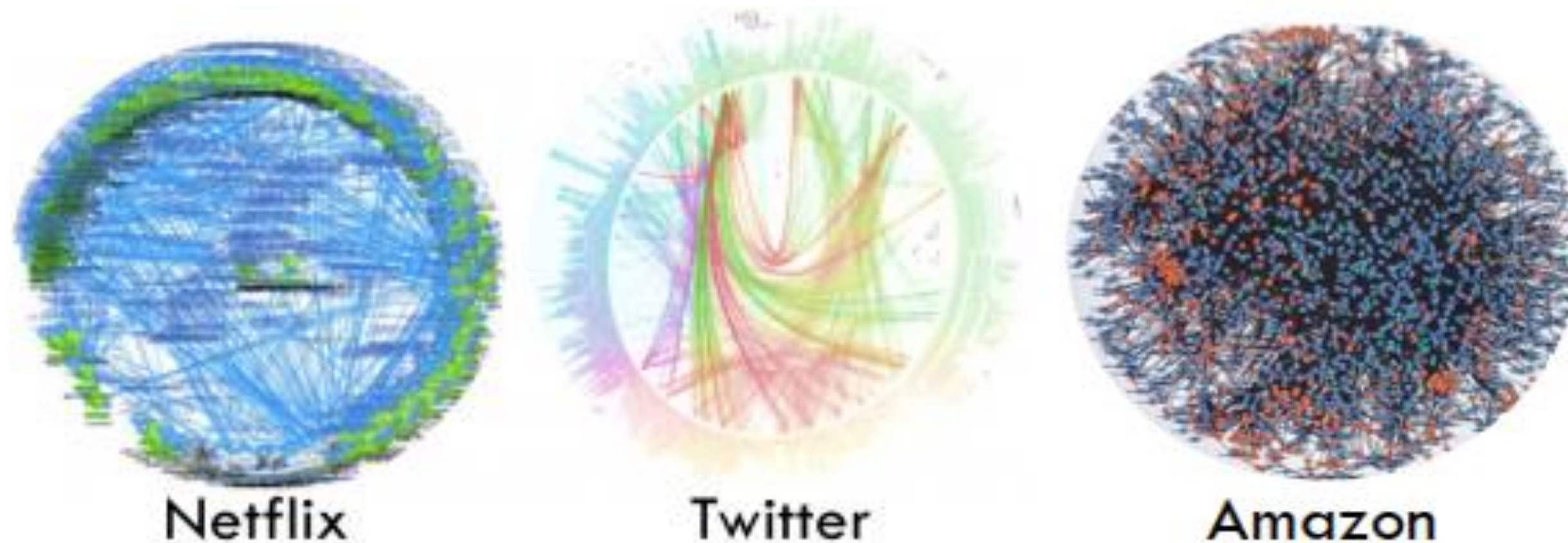

Benefits:
- Scalability
- Modularity
- Easy to maintain/debugging
- Different programming languages
- Loose-coupling, reliability
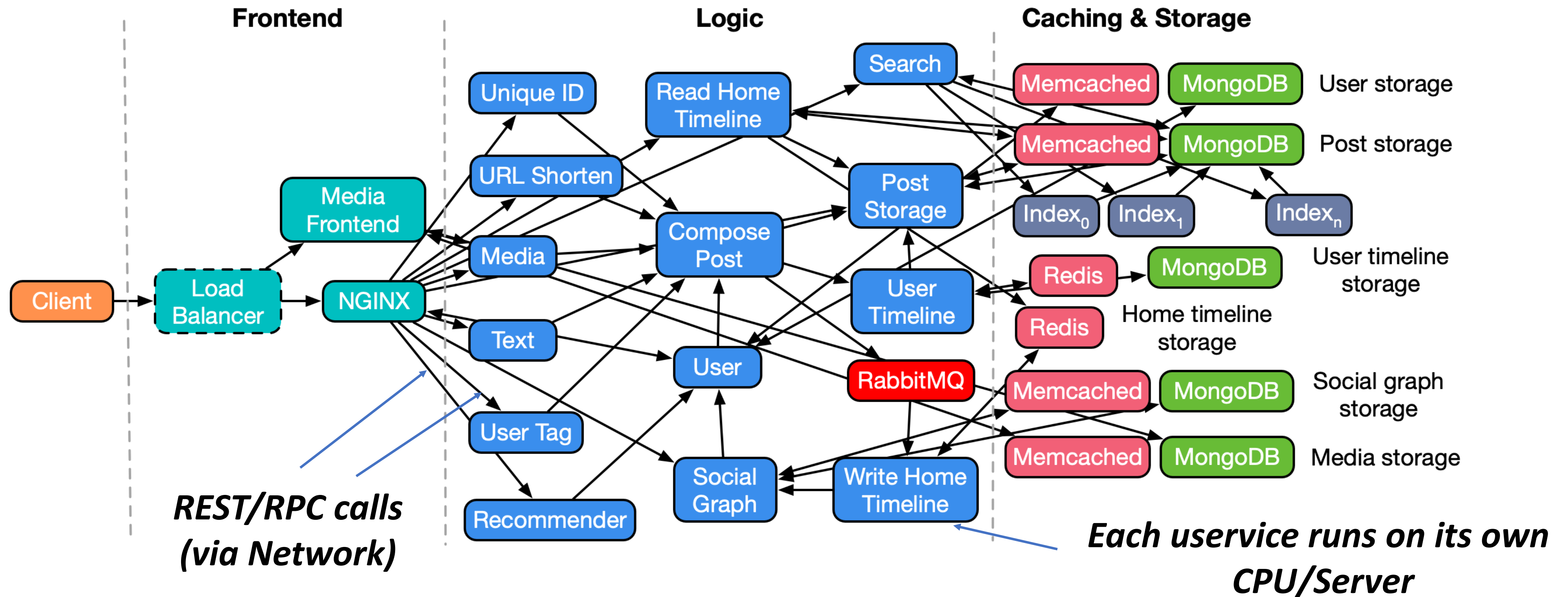- Owned by a small team

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

Reality is Much Complex

Netflix   Twitter   Amazon

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

58

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"

$\rightarrow$ 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

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

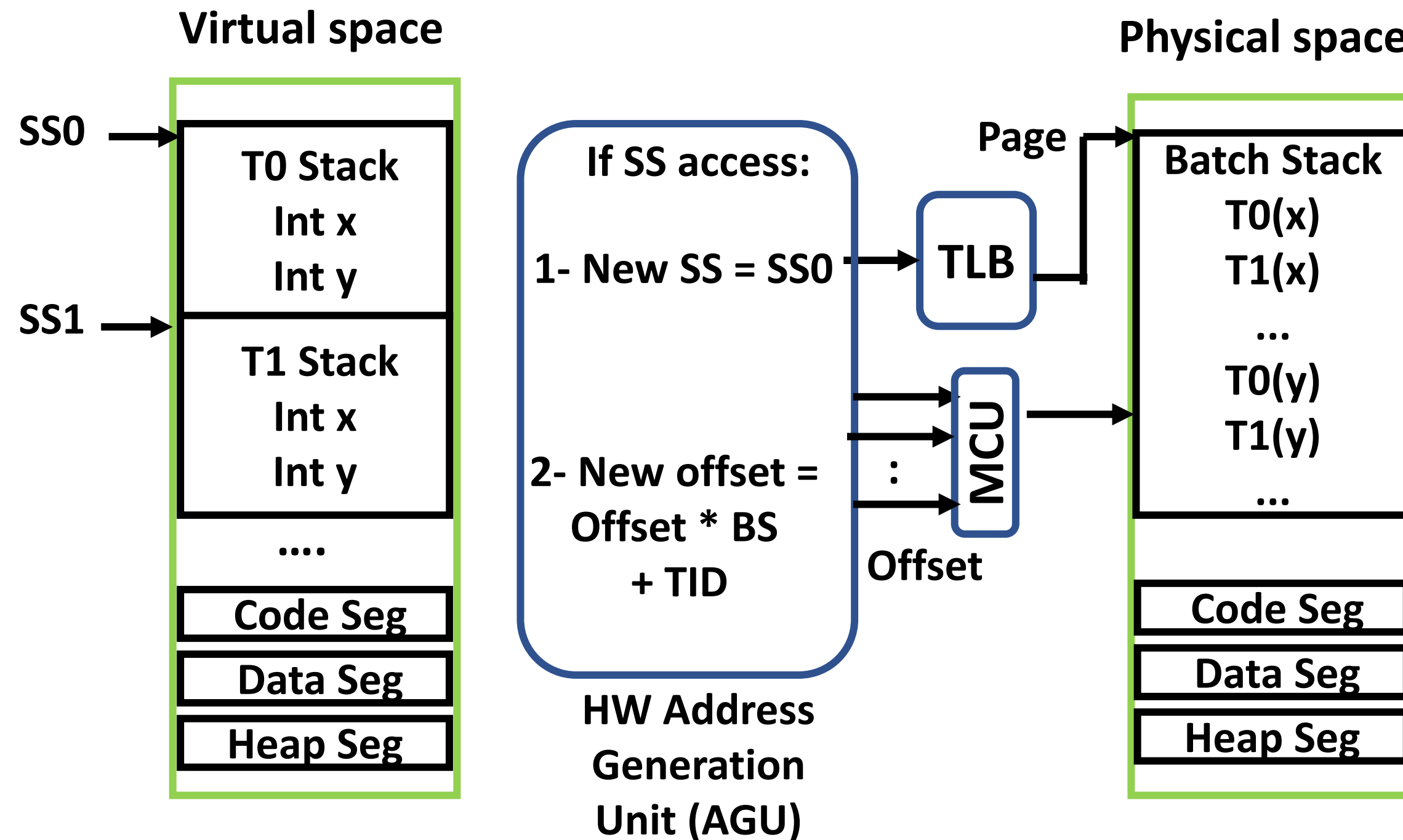- We need energy-efficient high-throughput system

What does this look like?

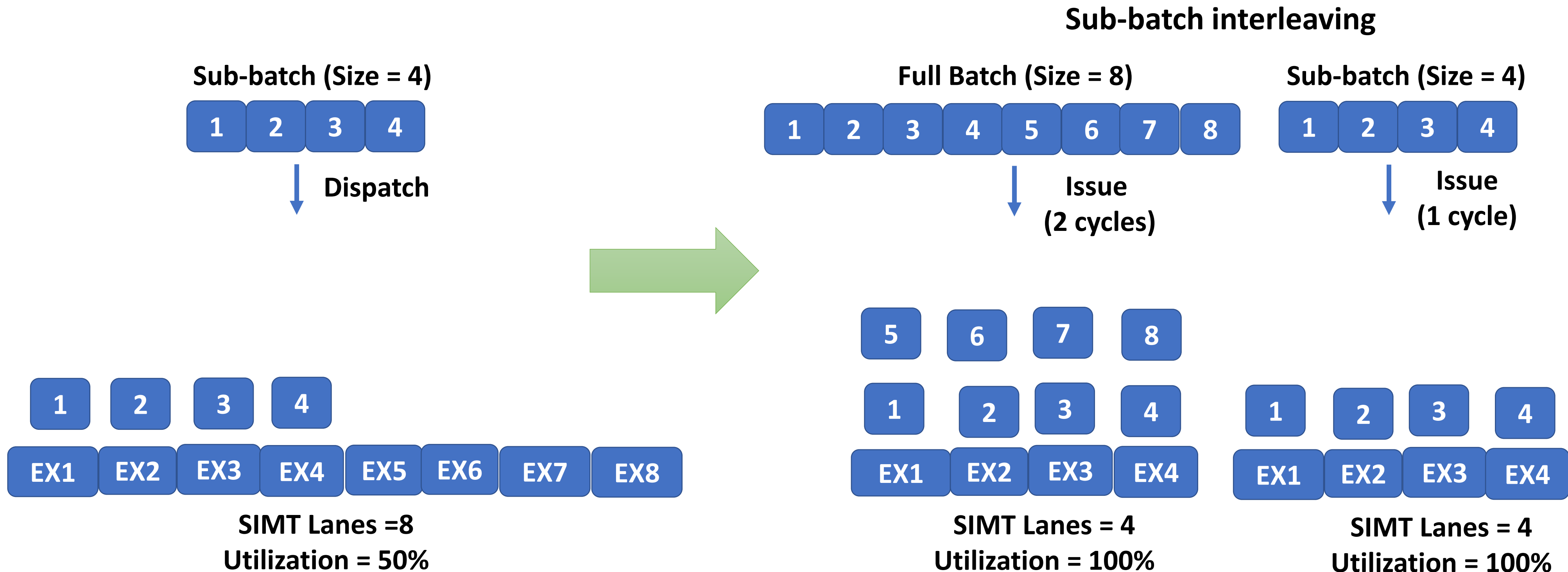Single Instruction Multiple Threads (SIMT) Or SIMD

But, wait, what about service latency?

More RPU Hardware Details

Transparent Stack Segment Coalescing

Sub-batch Interleaving

Sub-batch interleaving

Sub-batch (Size = 4)

| 1 | 2 | 3 | 4 |

↓ **Dispatch**

Full Batch (Size = 8)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

↓ **Issue (2 cycles)**

Sub-batch (Size = 4)

| 1 | 2 | 3 | 4 |

↓ **Issue (1 cycle)**

| 1 | | 2 | | 3 | | 4 |

| EX1 | EX2 | EX3 | EX4 | EX5 | EX6 | EX7 | EX8 |

SIMT Lanes =8
Utilization = 50%

| 5 | 6 | 7 | 8 |

| 1 | 2 | 3 | 4 |

| EX1 | EX2 | EX3 | EX4 |

SIMT Lanes = 4
Utilization = 100%

| 1 | 2 | 3 | 4 |

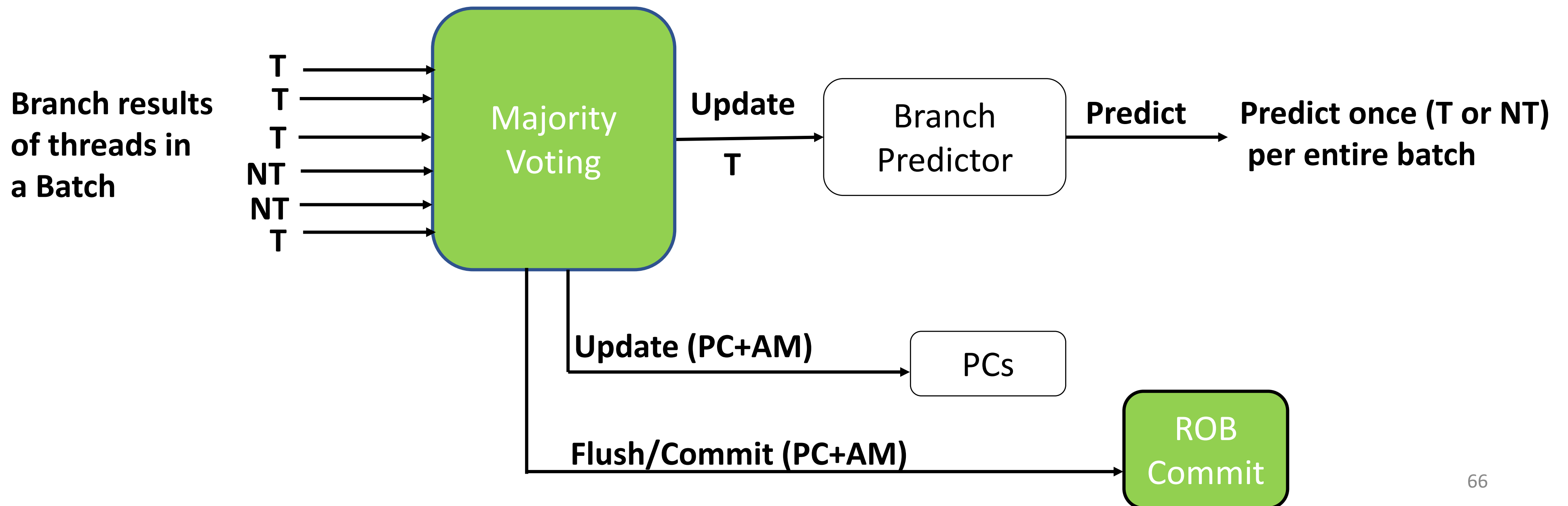| EX1 | EX2 | EX3 | EX4 |

SIMT Lanes = 4
Utilization = 100%

→ Alleviate divergence, exploit deeper pipeline & fully utilize your IPC utilization
  → In our final RPU configuration, SIMT lanes = 8 & max batch size = 32
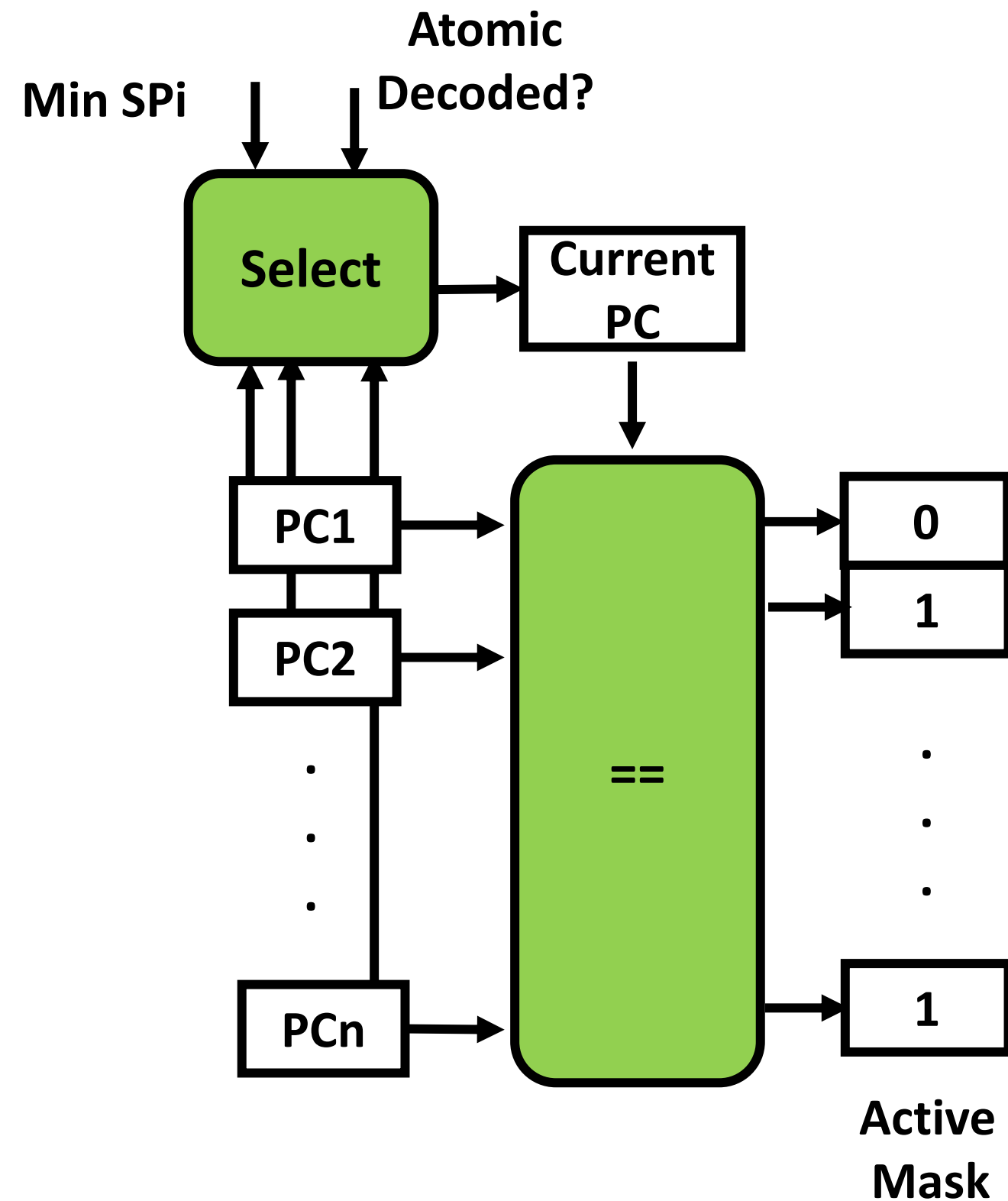
RPU's LD/ST Unit
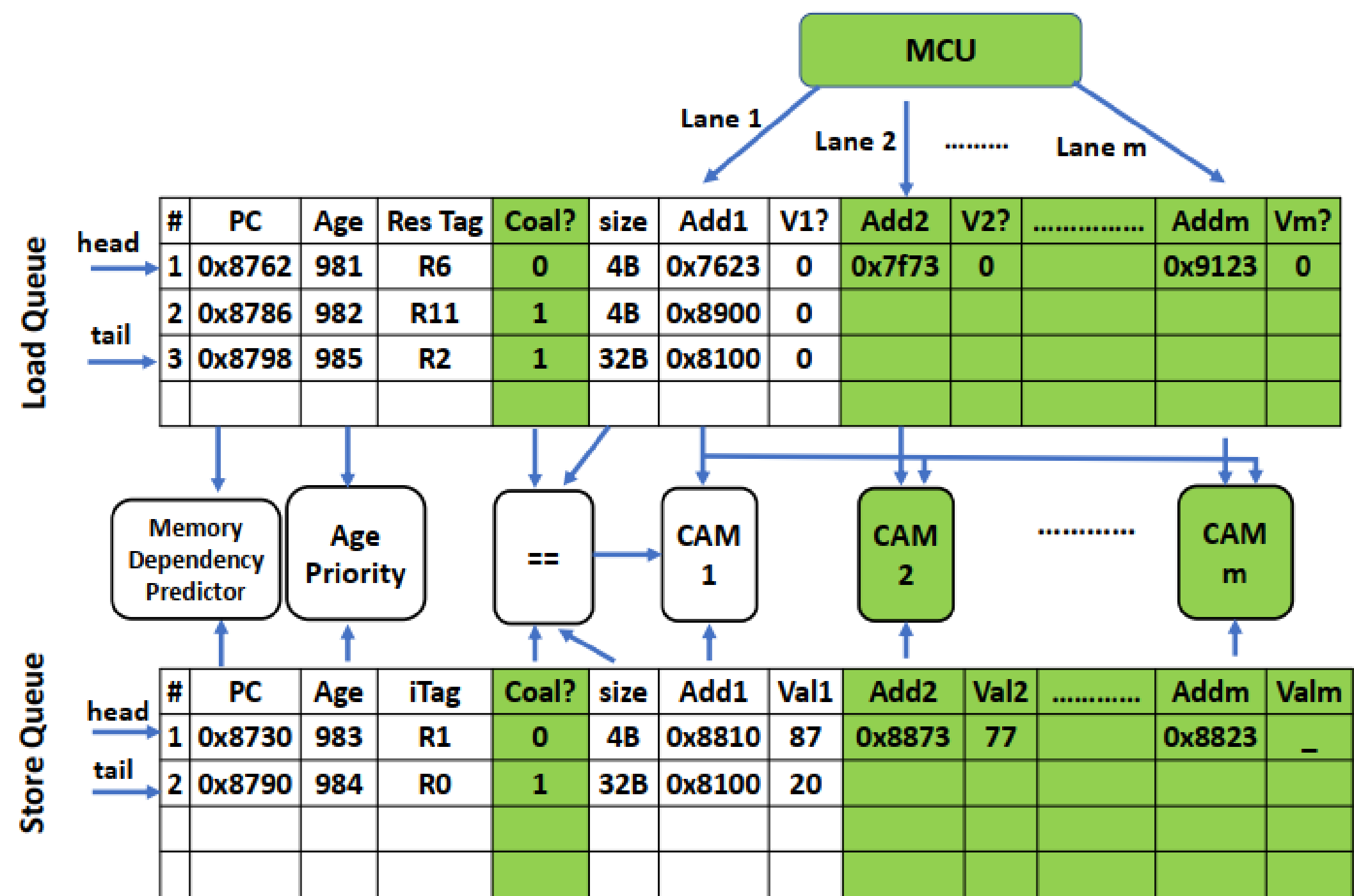
SIMT + Branch Predictor

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

Branch results of threads in a Batch

T
T
T
NT
NT
T

Majority Voting

Update T → Branch Predictor → **Predict** → **Predict once (T or NT) per entire batch**

Update (PC+AM) → PCs

Flush/Commit (PC+AM) → ROB Commit

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

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 in-efficiency	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:

μ service	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)	Low traffic but high latency
Ads1	O (10)	O (ms)	O (10^9)	
Ads2	O (100)	O (ms)	O (10^9)	
Cache1	O (100K)	O (μs)	O (10^3)	Low latency but high traffic
Cache2	O (100K)	O (μs)	O (10^3)	

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

Frontend+OoO Overhead is Increasing

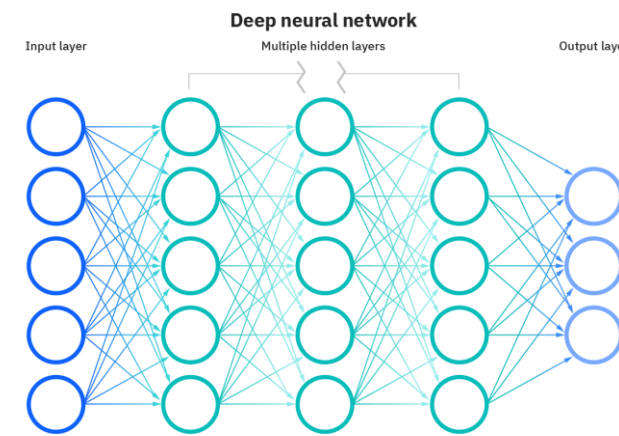| 128 ROB entries
6-wide issue
3-wide fetch | 256 ROB entries
8-wide issue
4-wide fetch | 352 ROB entries
12-wide issue
5 wide fetch | → Pipeline OoO is getting complex |
| SSE
128-bit | AVX
256-bit | AVX
512-bit | → SIMD is getting wider to amortize OoO complexity |
| 32-bit ALU | 32-bit ALU | 32-bit ALU | → BUT, scalar units remain the same |

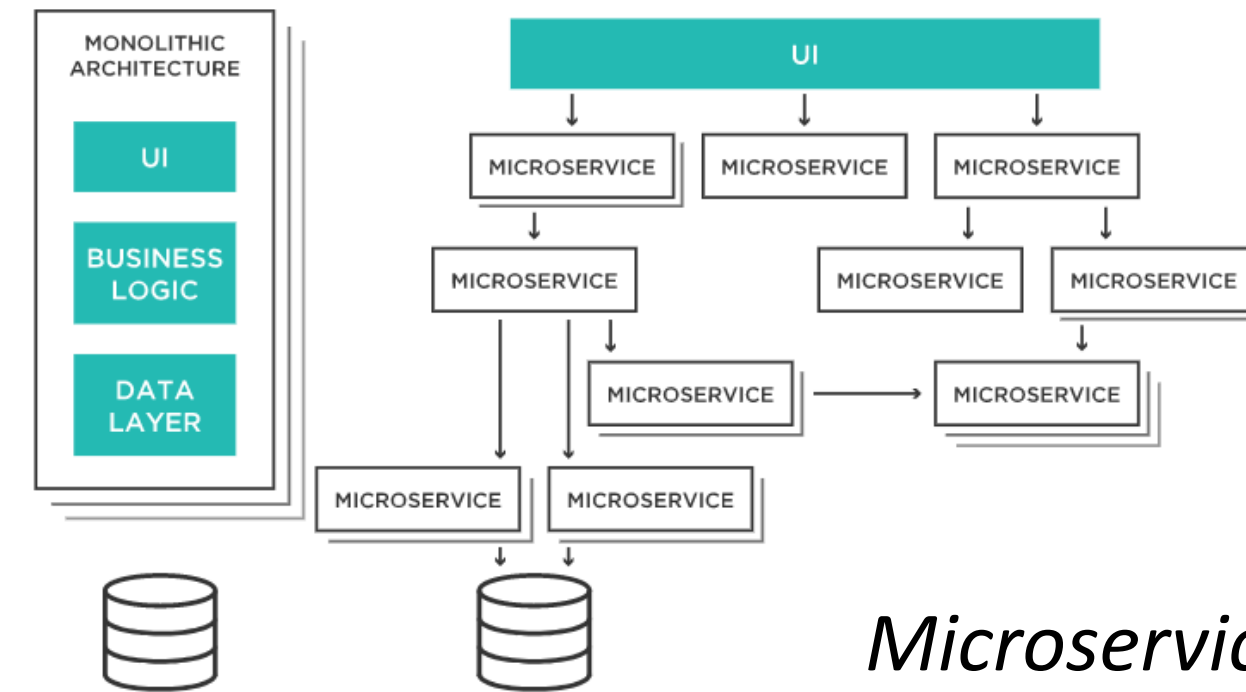As we move forward, the frontend+OoO overhead is getting larger compared to the scalar units
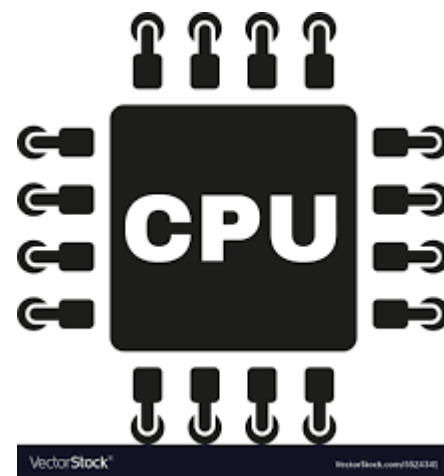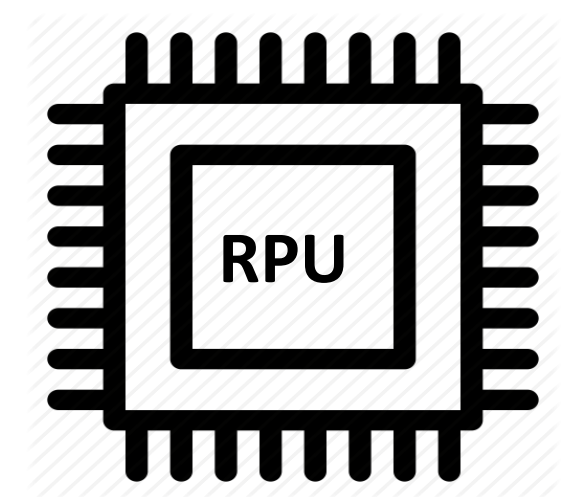
Thank You!
Q&A?

Software

Deep
Learning

Microservices

Hardware

| CPU | GPU | TPU | VCU | FPGA | RPU |

Accelerators