

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

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

Purdue University

\*Currently at AMD Research

# FAQs, Concerns and Pitfalls

I have presented this work multiple times, and this slides contain most of the questions and concerns I received along with my answers. If you do not find your question here, feel free to contact us.

Mahmoud Khairy – [abdallm@purdue.edu](mailto:abdallm@purdue.edu)

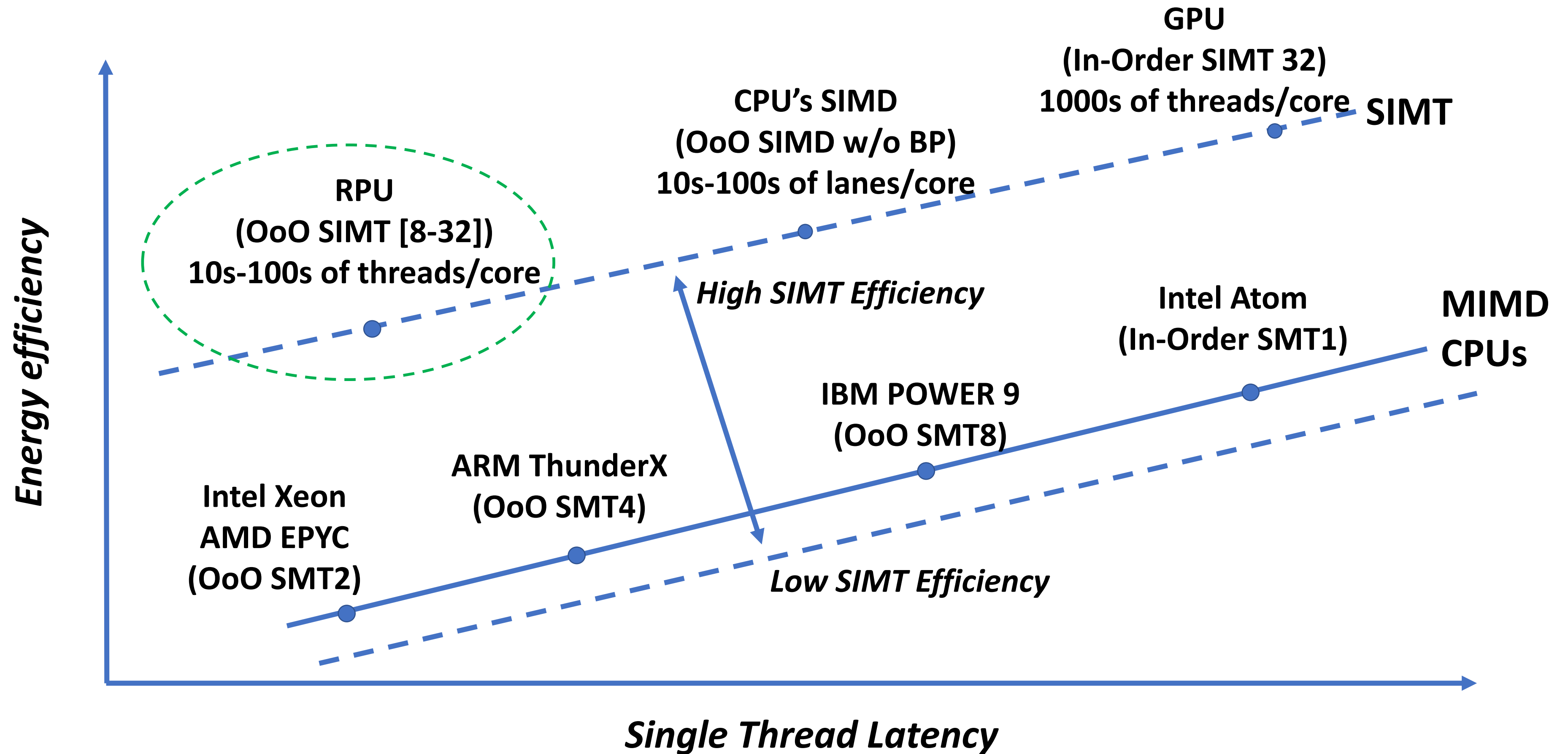
Tim Rogers – [timrogers@purdue.edu](mailto:timrogers@purdue.edu)

**Question: There have been many CPU solutions that increase throughput and improve energy efficiency, e.g. Sun's Niagara, Intel Atom, ARM Cortex, and others. How much is your RPU solution different from those?**

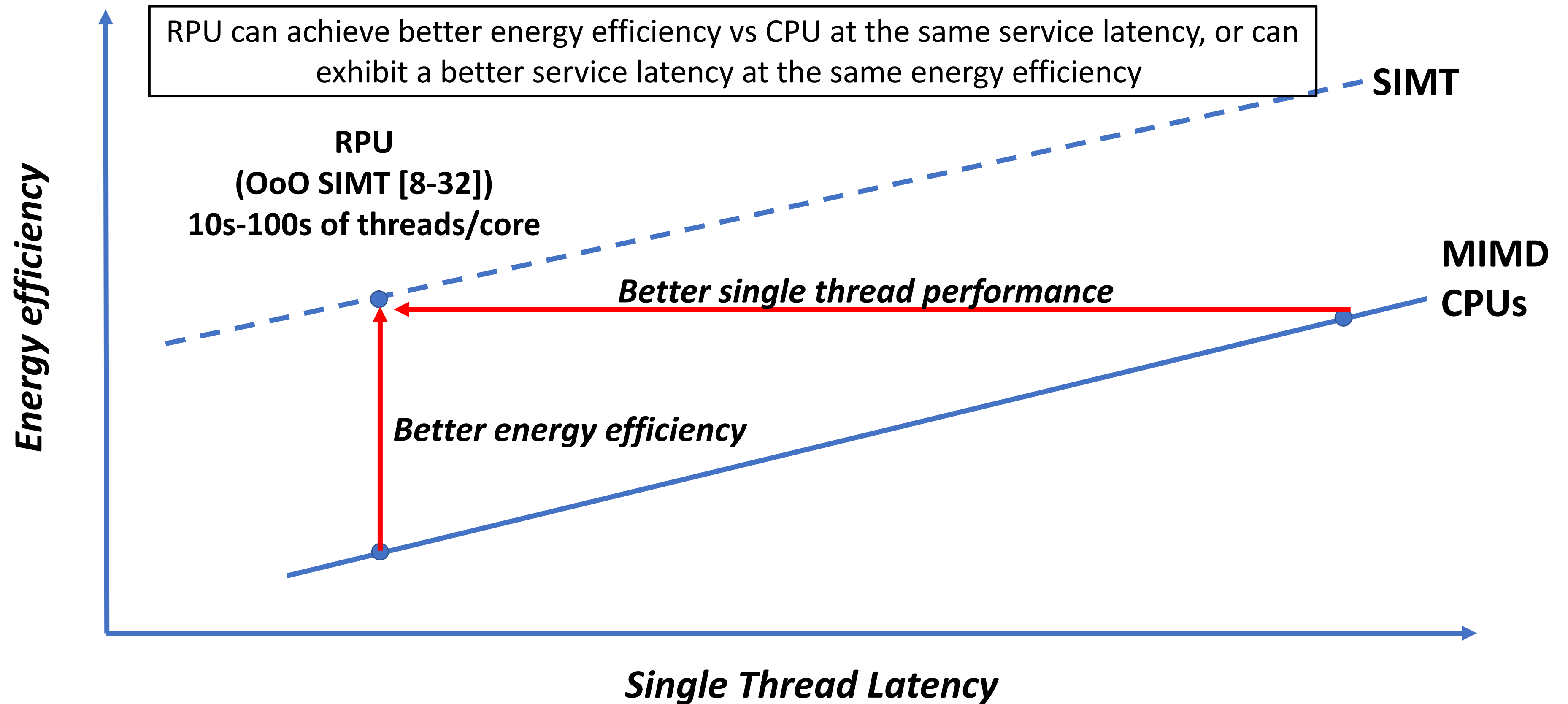
Answer: RPU is better in single-thread latency. Compared to wimpy CPUs, RPU will be better in either energy efficiency or single-thread latency or both, thanks to SIMT efficiency.

See next slides.

# Latency & Energy-Efficiency Tradeoff



# Latency & Energy-Efficiency Tradeoff



**Question: In RPU, you increase single thread latency by 1.44x, how much CPU's energy efficiency you can get from 1.44x higher latency? This should be your baseline**

Answer: Good Question! You can relax CPU's freq to 0.7x to get 1.44x higher latency. Typically, CPU power decreases by approximately  $O(k^2)$  when CPU frequency decreases by  $k$ . Thus, this will lead to 2x power efficiency.

But, we care about energy efficiency not power, energy = power x time. So, energy =  $0.5 * 1.44 = 0.72$ . Then, energy efficiency =  $1/0.72 = 1.38x$  only. However, RPU achieves 5.7x energy efficiency.

➔ Then, RPU's energy efficiency =  $5.77/1.38 = 4.1x$  better than CPU at the same service latency.

This comparison is missing in the original paper, but we plan to add it in our extended version of our paper.

**Question: Similarly, how is wimpy CPU's latency you can get at the same RPU's energy efficiency?**

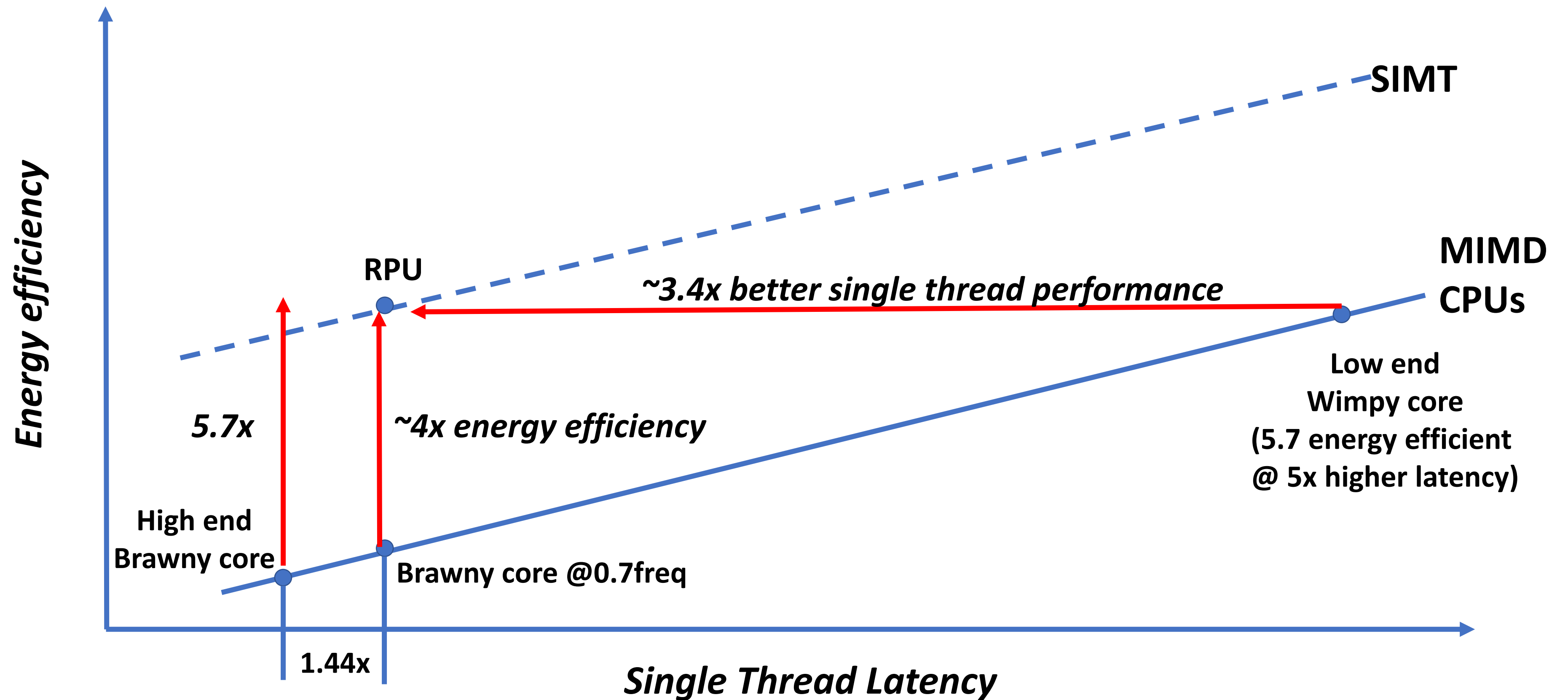
Answer: Intel Atom can get 10x less power at 3x higher latency vs high end brawny core. This means 3.3x energy efficiency (energy = power \* time). But, to achieve 5.7x energy as RPU, this means we need to scale latency to 5x down.

➔ Then, RPU is  $= 5/1.44 = 3.4x$  better single thread latency than CPU at the same energy efficiency.

Recall: power efficiency is easy to get, but energy efficiency is hard.

Note: this is a very high level analysis, further simulation-based analysis is needed to prove our claim.

# Latency & Energy-Efficiency Tradeoff



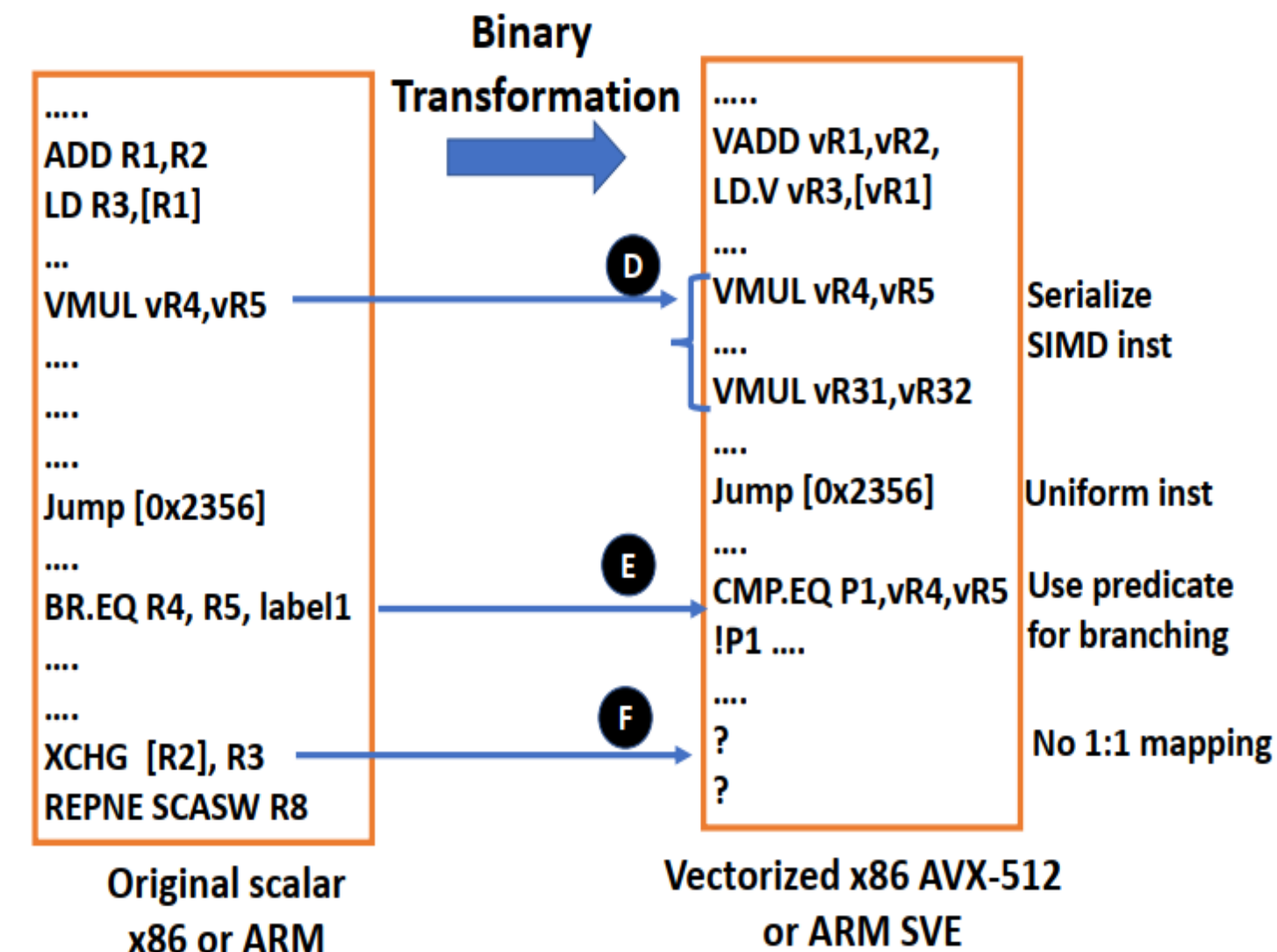


**Question: What about CPU's SIMD? Can we use SIMD instead of RPU?**

Answer: Let's recall what was SIMD designed for: CPU's SIMD was designed to accelerate data-parallel portion of the code (i.e., loops). It was not intended to execute the entire application (including system calls, initialization, heavily branching and function calls). Therefore, there are many hurdles that SIMD is facing to execute microservices. See next slide.

# What about CPU's SIMD?

- Recall: CPU's SIMD was designed to accelerate data-parallel portion (i.e., loops), not to execute the serial portion
- Therefore, many existing scalar instructions lack a 1:1 mapping with any vector instruction
  - For example, x86 AVX instructions only cover 27% of the scalar ISA
- Even if the ISA barriers is resolved, two more issues still exist:
  - Non transparency: recompilation/transformation for the entire SW stack (user code, libs & OS system calls)
  - Limited service latency:
    - Running coarse-grain threads in fine grain lane context. What if the scalar code already contains SIMD insts?
    - SIMD relies heavily on predicates for conditional branches, limited support for per-lane branch prediction



## Concern: Batching Overhead?

Answer:

- 1- At a very high traffic (with billion-scale services), batching overhead is amortized
- 2- Batching is already applied in today's data centers widely (see Google TPUv4\*) with batch sizes of range 8-200 requests. So, the data centers are able to deal with batching overhead already.
- 3- The user can set a time-out per service to ensure the request meets the QoS

See next slides.

In general, if the service cannot tolerate batching of any size (we support as small as 8 request batch) or shows low SMT efficiency, then they can be executed on the CPUs at lower throughput

\* Jouppi, Norman P., et al. "Ten lessons from three generations shaped Google's TPUv4i: Industrial product." *ISCA 2021*

# Batching Optimization

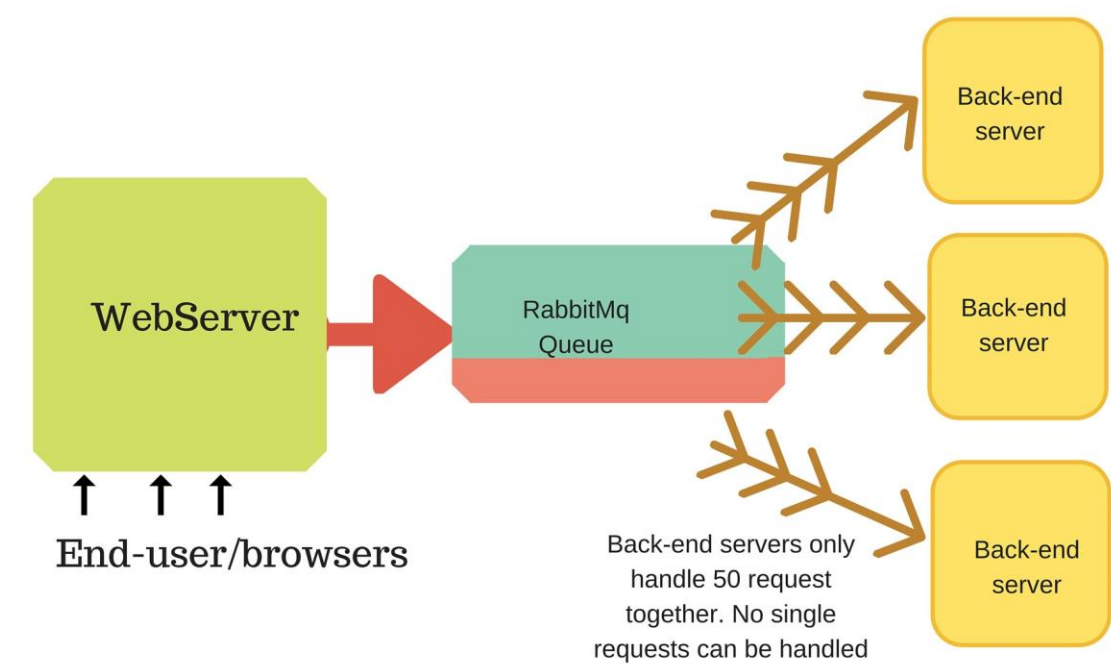
From Google’s Production DL Inference

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.

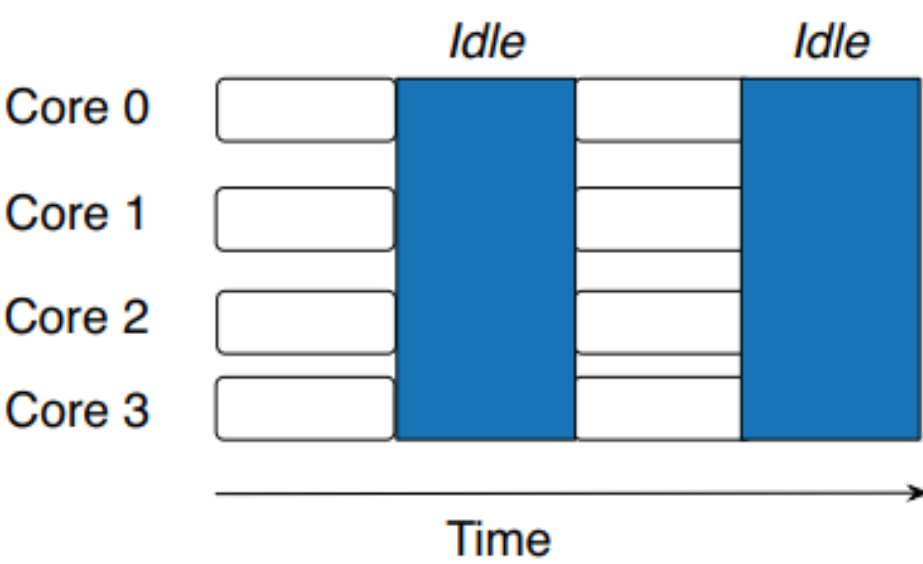
DL Inference Batching

Memcached servers



Network Batching

Power management



Batching for deep sleep

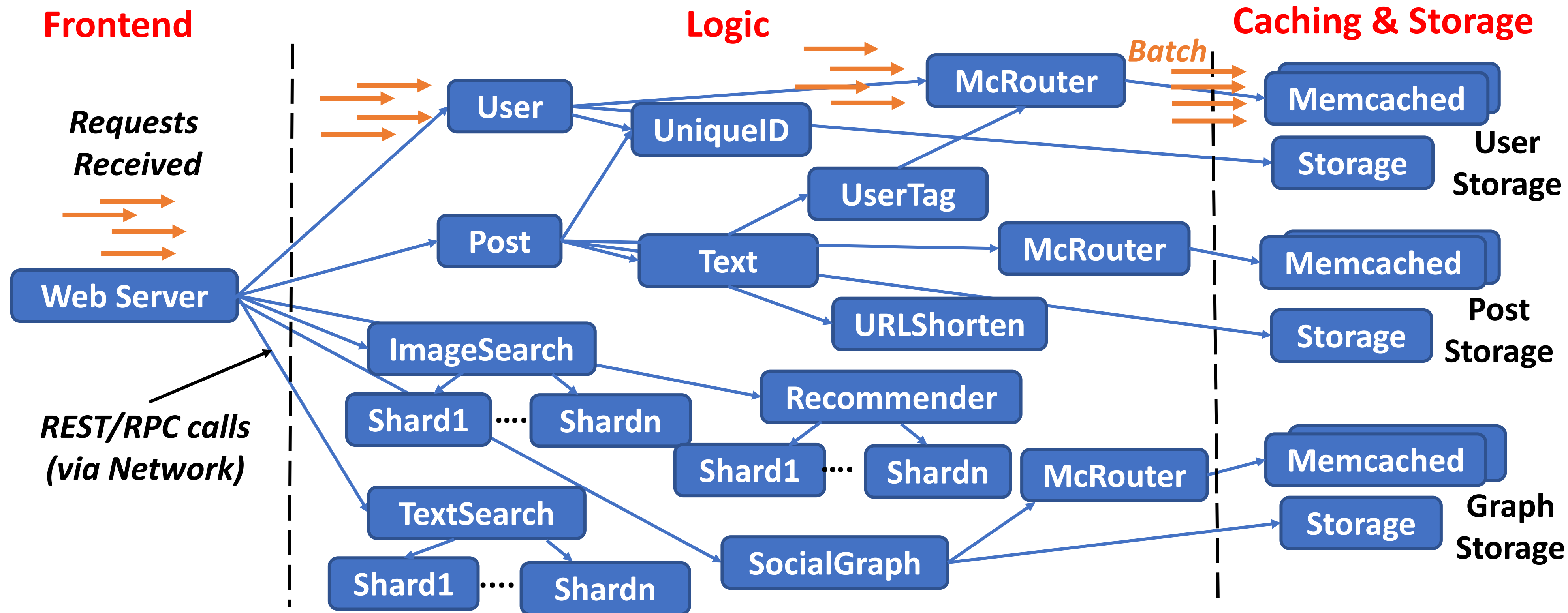
**Key Observation:** Modern data centers already rely on request batching heavily

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

<https://memcached.org/blog/nvm-multidisk/>

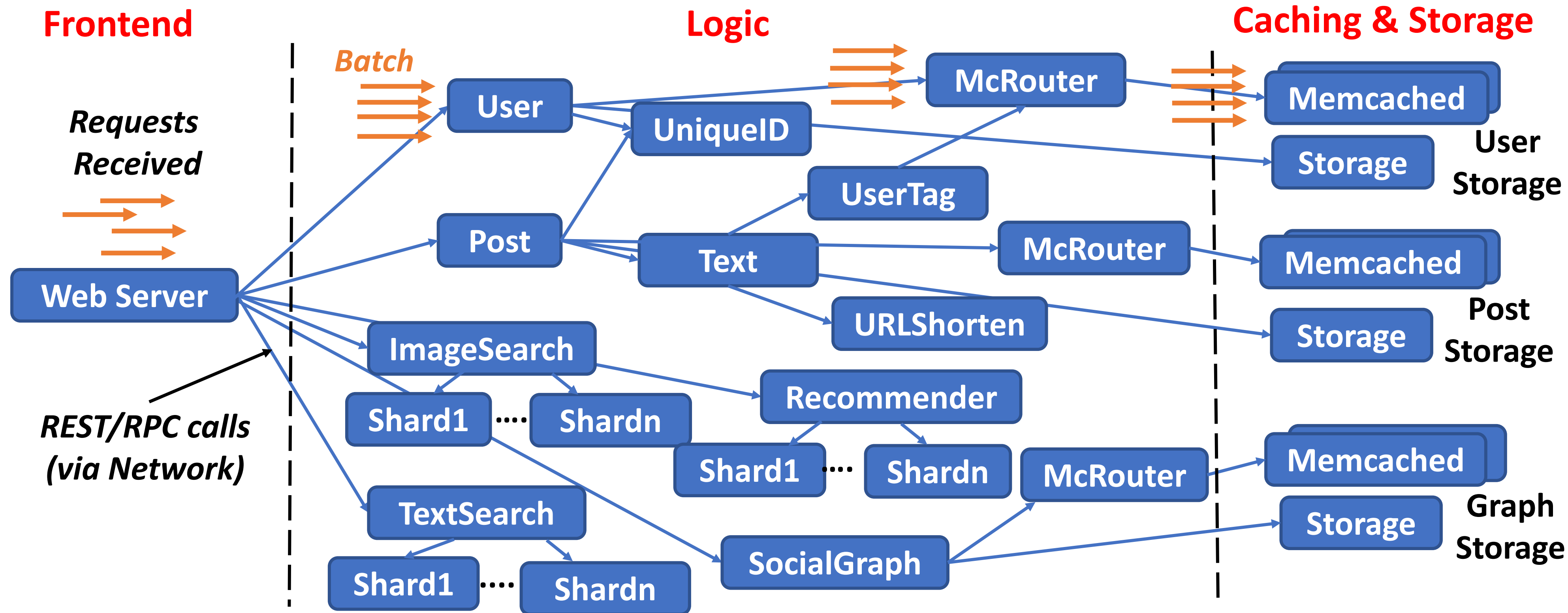
Meisner, David, and Thomas F. Wenisch. "Dreamweaver: architectural support for deep sleep." ASPLOS 2012

# Current System: Selective Batching



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

# Our Proposed System-Level Batching





# 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:

$\mu$ service	Throughput (QPS)	Req. latency	Insn./query	
Web	O (100)	O (ms)	O ( $10^6$ )	} <b>Low traffic but high latency</b>
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 ( $\mu$ s)	O ( $10^3$ )	} <b>Low latency but high traffic</b>
Cache2	O (100K)	O ( $\mu$ s)	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.

# Batching Opportunity for Google Services

- (1) from Google in-production ML inference services:
  - Batching is widely used for DL inference with size = 8-200 reqs based on traffic and latency

<i>Production</i>						<i>MLPerf 0.7</i>		
<i>DNN</i>	<i>ms</i>	<i>batch</i>	<i>DNN</i>	<i>ms</i>	<i>batch</i>	<i>DNN</i>	<i>ms</i>	<i>batch</i>
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



**Concern: Batching already has an overhead and batching per-argument size as you proposed makes the overhead even worse?**

Answer: In real world scenarios, most of the arguments/queries length is within a very small standard deviation.

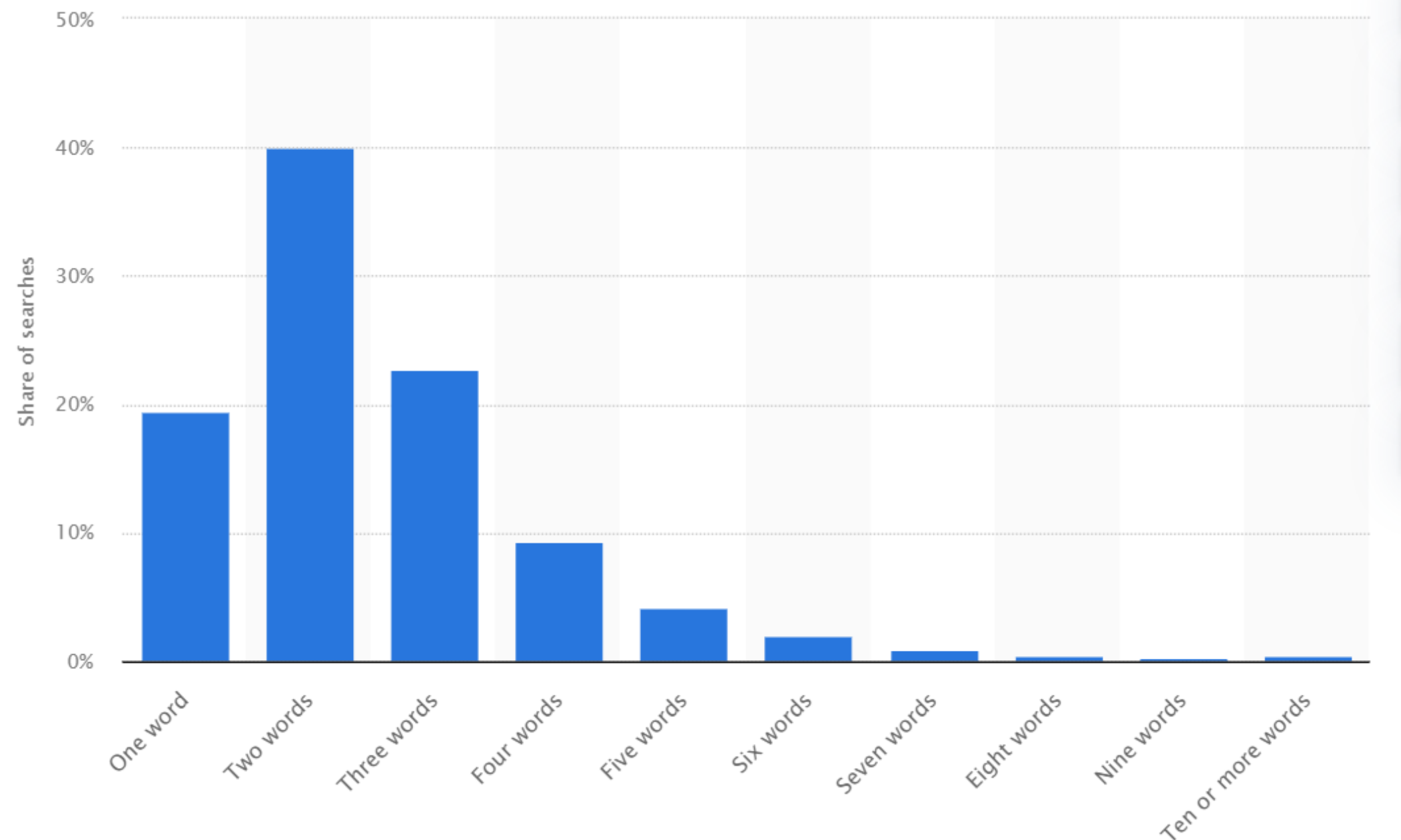
For example, 90% of Google search queries are 1-5 words.

Small standard deviation query length + high traffic will make the chance of formulating a batch of similar size very high.

Here, we make an assumption the query length is an indication for the query service time, If not (in the rare cases), apply batch split technique when needed.

# Online search queries statistics

google queries length



Small standard deviation

google queries frequency (2018)

- Searches per second: 63,000
- Searches per minute: 3.8 million
- Searches per hour: 228 million
- Searches per day: 5.5 billion
- Searches per month: 167 billion
- Searches per year: 2 trillion

Very High Traffic

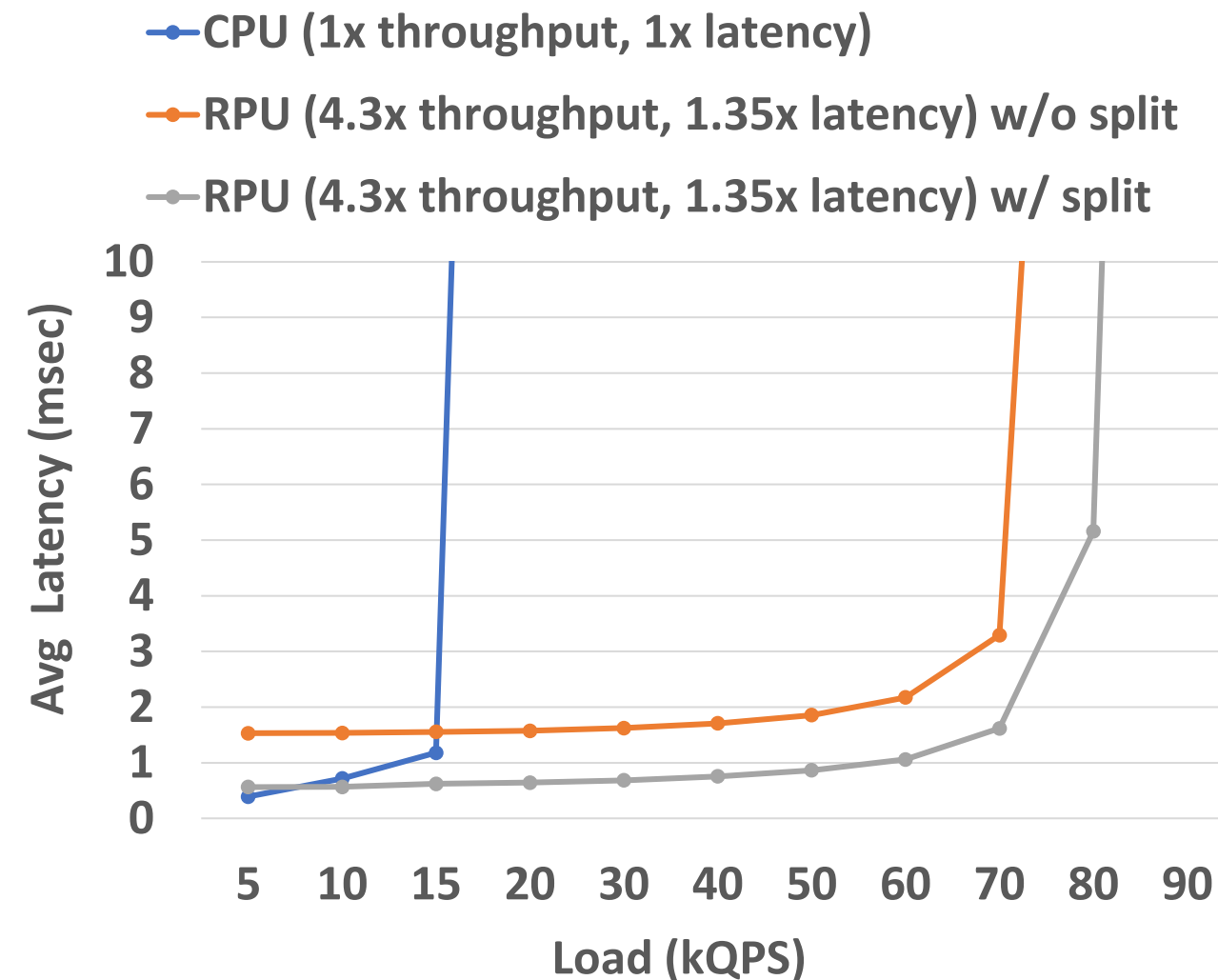
**Concern: How can you get high SIMT efficiency on data dependent code?  
Batching per length will help on some cases, but what if there is IF condition based on the query value itself?**

Answer: In fact, this is what the 8% control divergence is about. We do not claim 100% efficiency, we achieve 92%. We do have some divergence occurring. For example, B+ tree traversal, data compression in zlib library, and others, all of these are dependent on data values and report high control divergence in our traces.

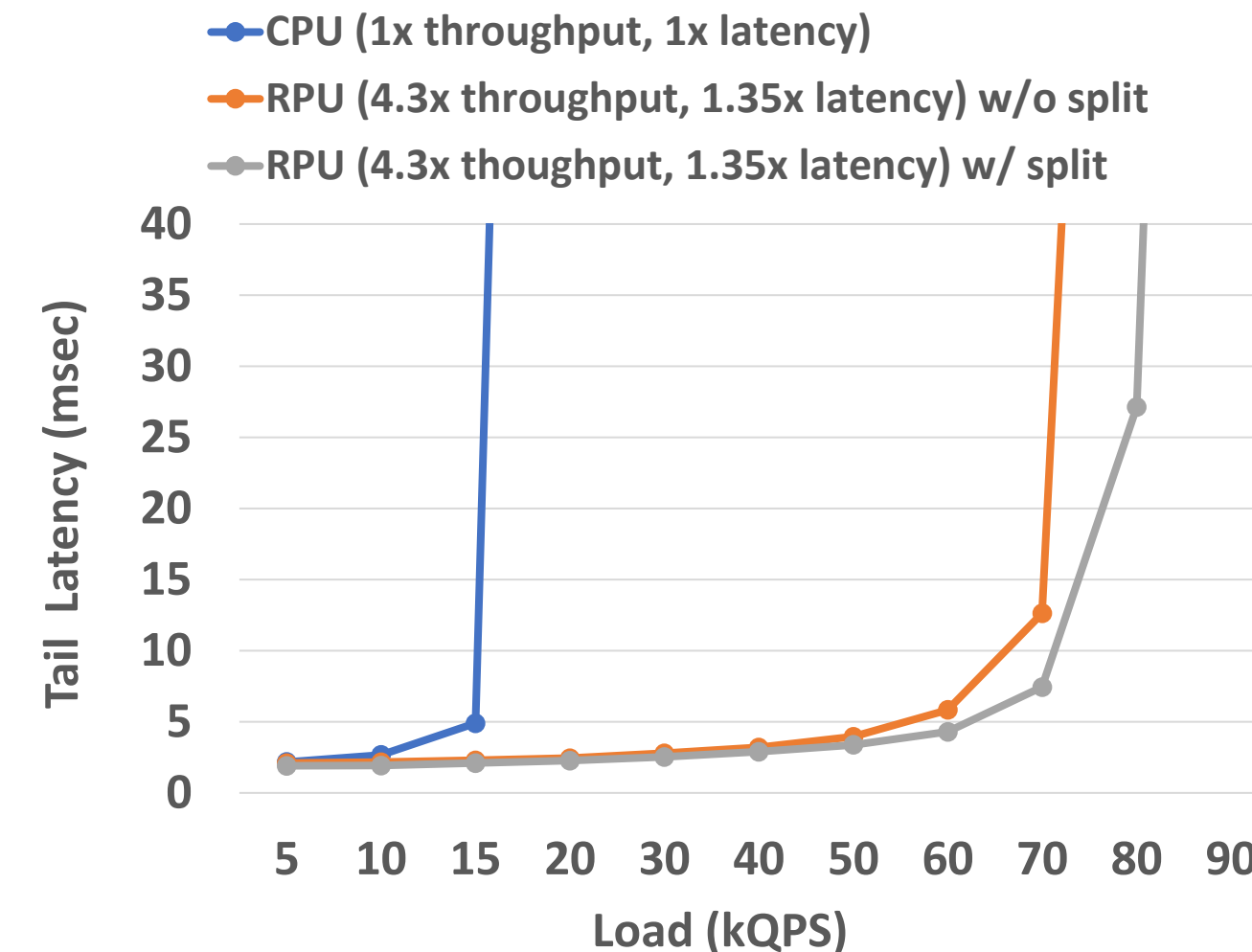
**Question: How much does the 1.44x higher latency affect your end-to-end average and tail latency?**

Answer: We run our experiments to study the service latency increase effect on system scale. Requests typically spend 50% of time in networking and 50% in processing. So, If the processing time (i.e., RPU service latency) increases by 44%, then this will lead to end-to-end latency increase by 22% as networking time remains the same. With further tuning and throttling, we can maintain the almost same latency as CPU systems on system level.

# 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 the same average and tail latency as CPU system at 4x higher throughput
- Without the batch split technique, we are still able to get a good tail latency

**Question: What would the threshold be for latency increase that can be tolerated by data centers?**

Answer: Up to 2x slower latency should be tolerated by data center providers. So, for example, see the high-end CPU in the market (e.g. Intel Xeon or AMD EPYC) and ensure your proposed energy-efficiency CPU design is not behind by more than 2x of single thread performance. See Hölzle [MICRO'10].

# Maximum Tolerated Latency

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

***Quoted:***

So, although we're enthusiastic users of multicore systems, and believe that throughput-oriented designs generally beat peak-performance-oriented designs, smaller isn't always better. **Once a chip's single-core performance lags by more than a factor to two or so behind the higher end of current-generation commodity processors, making a business case for switching to the wimpy system becomes increasingly difficult** because application programmers will see it as a significant performance regression: their single-threaded request handlers are no longer fast enough to meet latency targets. So go forth and multiply your cores, but do it in moderation, or the sea of wimpy cores will stick to your programmers' boots like clay.

**Urs Hölzle**  
*Google*

**Question: Scaling the threads will be limited by lock contention & critical section?**

Answer: Real-world data center workloads are highly optimized to reduce lock contention by employing lock-free data structure, highly-concurrent memory allocator and fine-grain locking. See next slides.

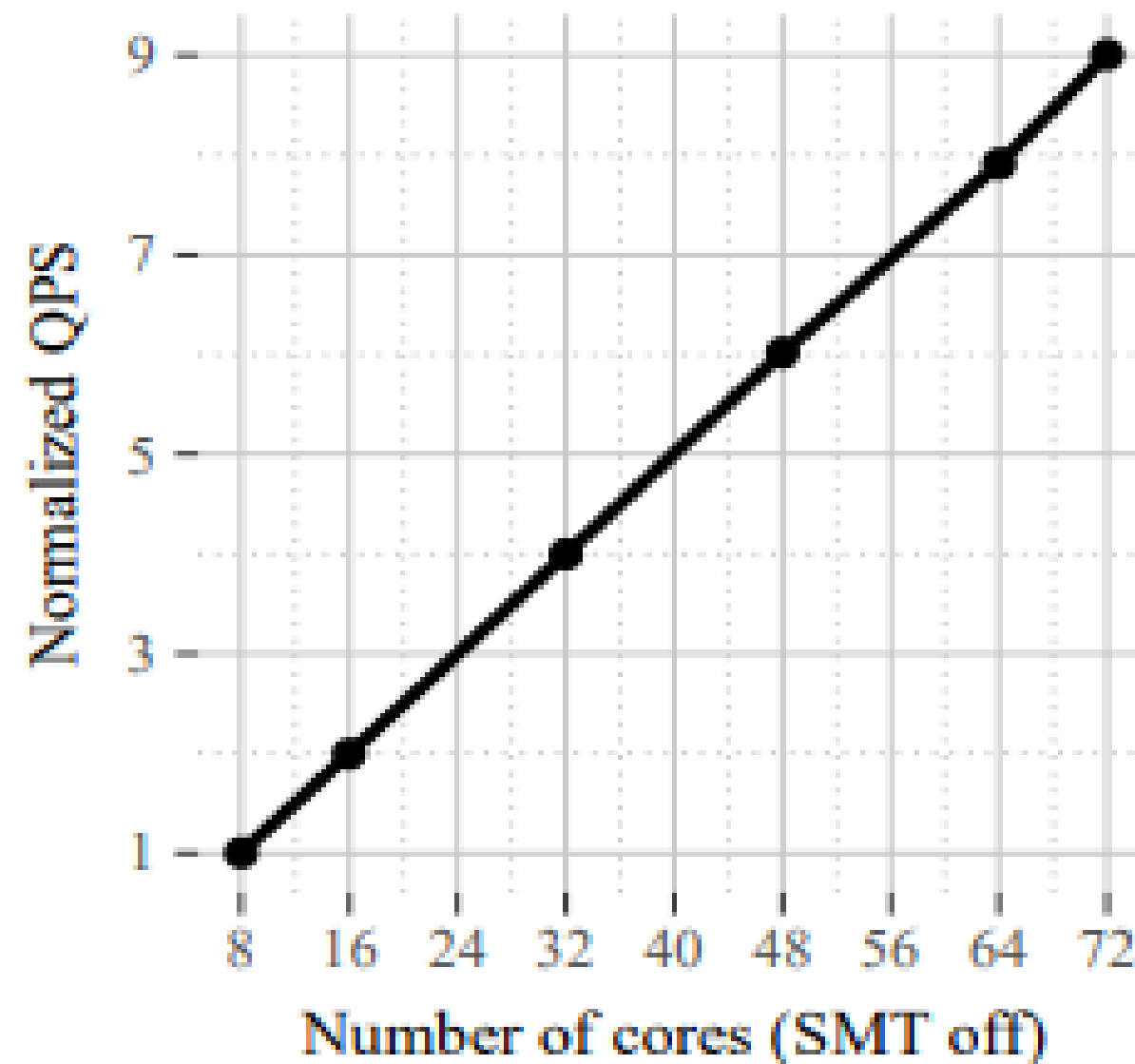
Note that: this is not just a scalability issue for RPU, it is also an issue for multi-core CPU designs too.



# Perfect Scaling in Real-world Server Workloads (I)

- From Google in-production Search service

**9x more cores = 9x more QPS!**



Multi Threaded Servers

Quoted: “The near-perfect scaling implies that search has a limited amount of read/write sharing or locking in the memory system”

# Perfect Scaling in Real-world Server Workloads (II)

Another example: Multi-threaded Memcached

**4x more cores = 4x more RPS**

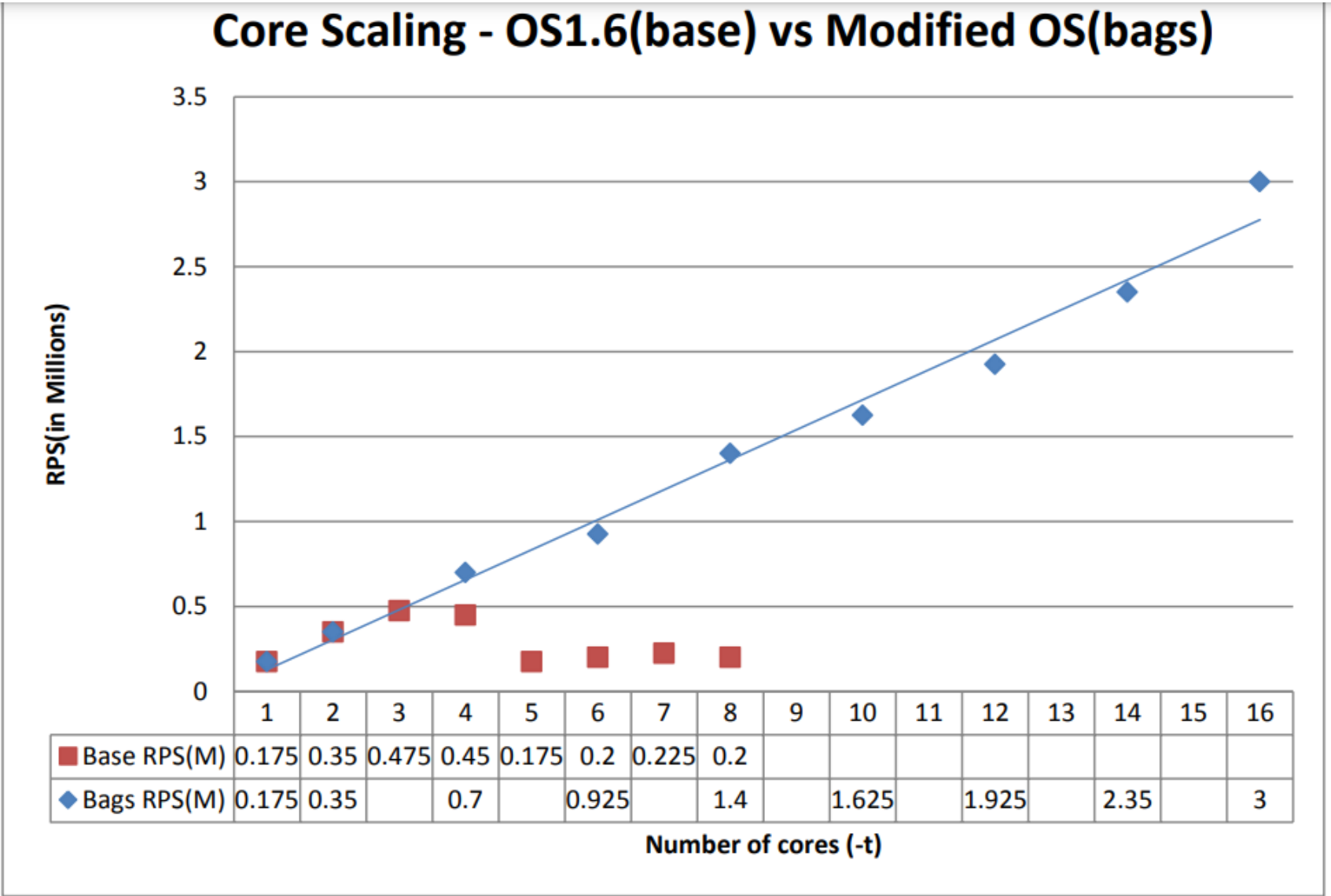


Figure 14 - Maximum throughput with a median RTT < 1ms SLA as core counts increase

Quoted: “The approach employs Concurrent data structures and a modified cache replacement strategy to improve scalability. These data structures enable concurrent lockless item retrieval and provide striped lock capability for hash table updates”

**Question: Why do you call it RPU?**

Answer: An Accelerator to exploit thread similarity and Request level parallelism  
→ Requ<sup>st</sup> Process<sup>ing</sup> Unit (RPU)

In Literature, RLP is different than TLP

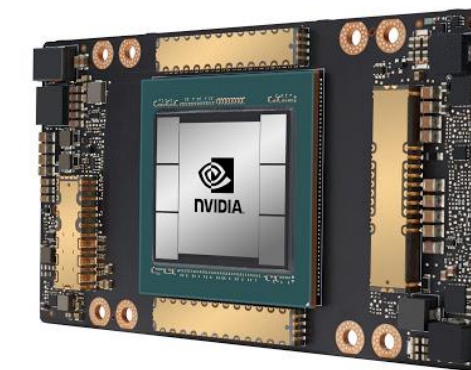
Chapter 3	
<b>Instruction-Level Parallelism and Its Exploitation</b>	
3.1	Instruction-Level Parallelism: Concepts and Challenges
3.2	Basic Compiler Techniques for Exposing ILP
3.3	Reducing Branch Costs with Advanced Branch Prediction
3.4	Overcoming Data Hazards with Dynamic Scheduling
3.5	Dynamic Scheduling: Examples and the Algorithm
3.6	Hardware-Based Speculation
3.7	Exploiting ILP Using Multiple Issue and Static Scheduling
3.8	Exploiting ILP Using Dynamic Scheduling, Multiple Issue, and Speculation
3.9	Advanced Techniques for Instruction Delivery and Speculation
3.10	Studies of the Limitations of ILP
3.11	Cross-Cutting Issues: ILP Approaches and the Memory System
3.12	Multithreading: Exploiting Thread-Level Parallelism to Improve Uniprocessor Throughput
3.13	Putting It All Together: The Intel Core i7 and ARM Cortex-A8
3.14	Fallacies and Pitfalls
3.15	Concluding Remarks: What's Ahead?
3.16	Historical Perspective and References
	Case Studies and Exercises by Jason D. Bakos and Robert P. Colwell
Chapter 4	
<b>Data-Level Parallelism in Vector, SIMD, and GPU Architectures</b>	
4.1	Introduction
4.2	Vector Architecture
Chapter 5	
<b>Thread-Level Parallelism</b>	
5.1	Introduction
5.2	Centralized Shared-Memory Architectures
5.3	Performance of Symmetric Shared-Memory Multiprocessors
Chapter 6	
<b>Warehouse-Scale Computers to Exploit Request-Level and Data-Level Parallelism</b>	
6.1	Introduction
6.2	Programming Models and Workloads for Warehouse-Scale Computers
6.3	Computer Architecture of Warehouse-Scale Computers



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



Data level parallelism (DLP)



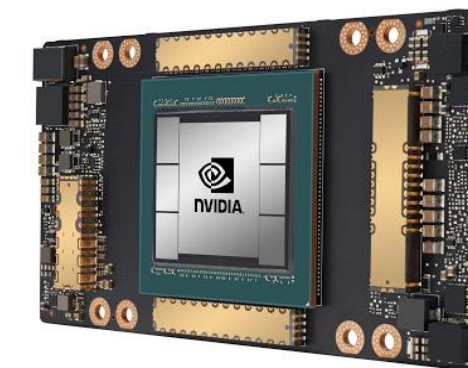
Request level parallelism (RLP)

??

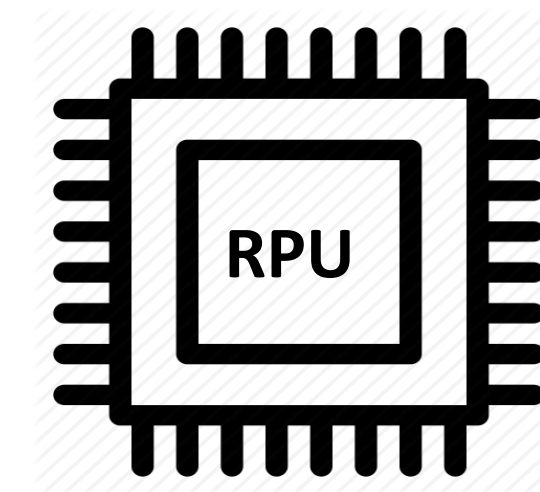
Instruction level parallelism (ILP) &  
Thread level parallelism (TLP)



Data level parallelism (DLP)



Request level parallelism (RLP)



**Question: How latency is only increased by 44%?**

**4x higher instruction latency, 2x higher L1 access latency and sub batch interleaving should lead to more than that**

Answer: As shown in previous studies, data center workloads exhibit a limited IPC and retire rate as they are bounded by memory latency. See next slides

# Low IPC in Data Center

Facebook (SMT is On)

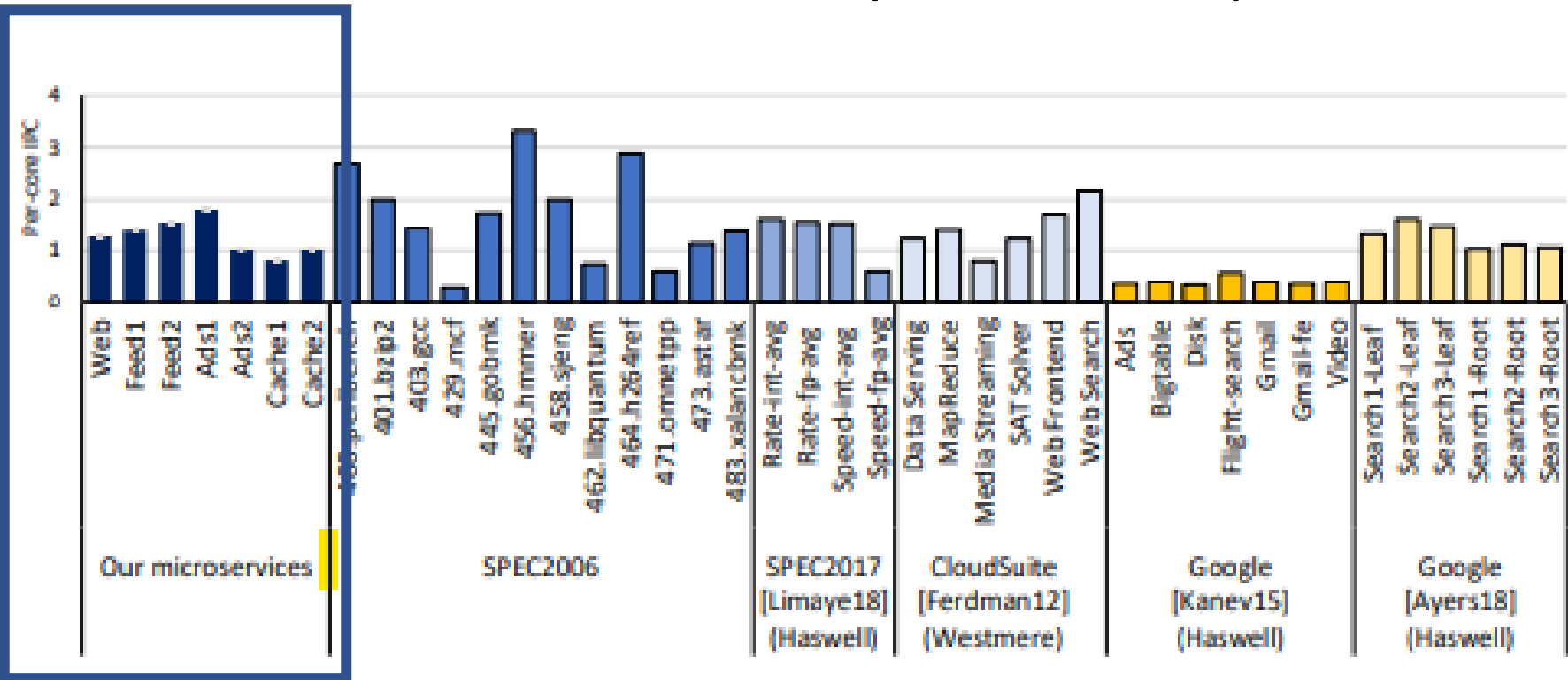


Figure 6: Per-core IPC across our  $\mu$ services & prior work (IPC measured on other platforms): our  $\mu$ services have a high IPC diversity.

For FB, SMT is on, so divide the IPC per 2 to get IPC per thread approximately

IPC per thread = 0.5-1

Google (SMT is off)

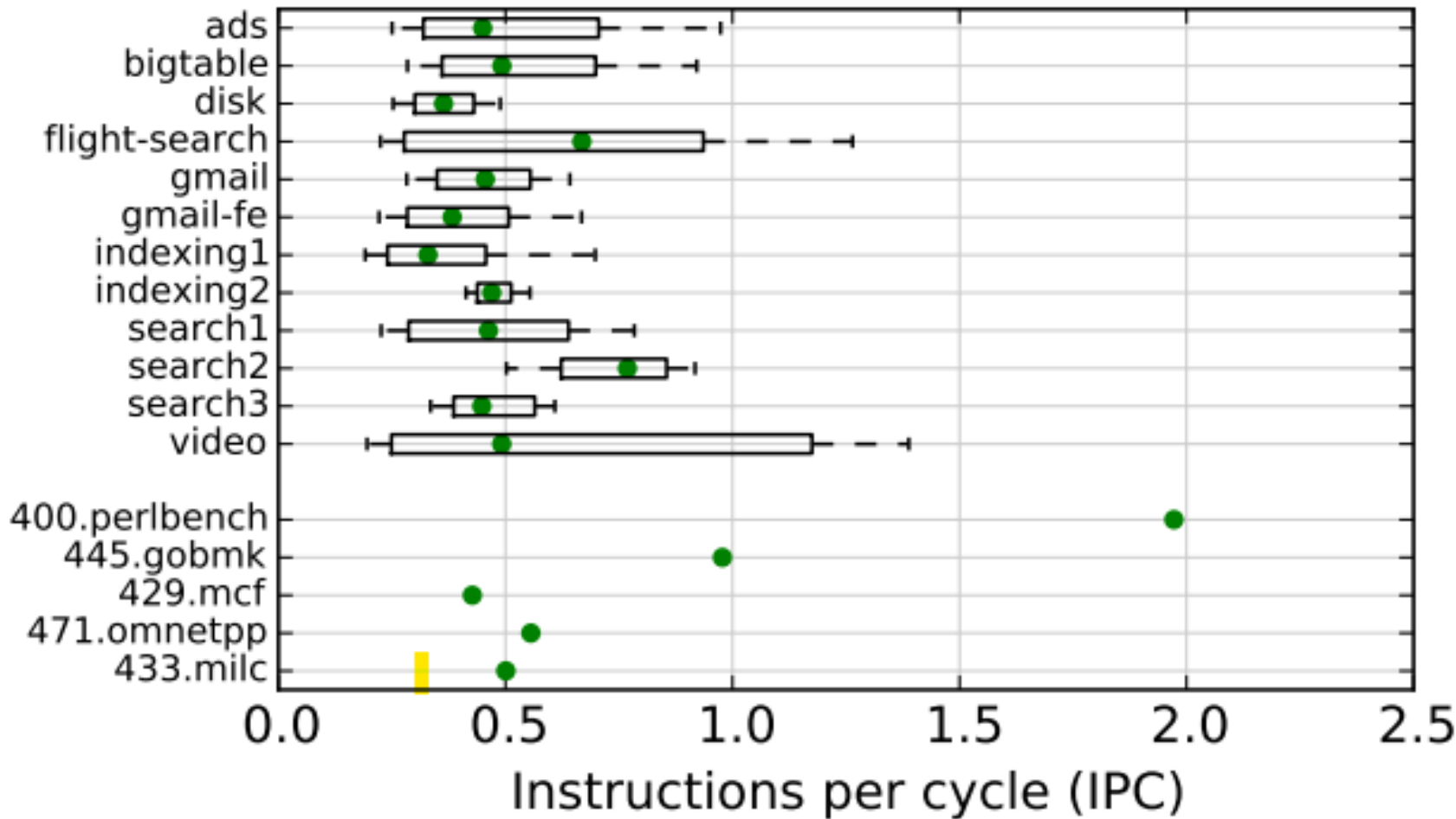
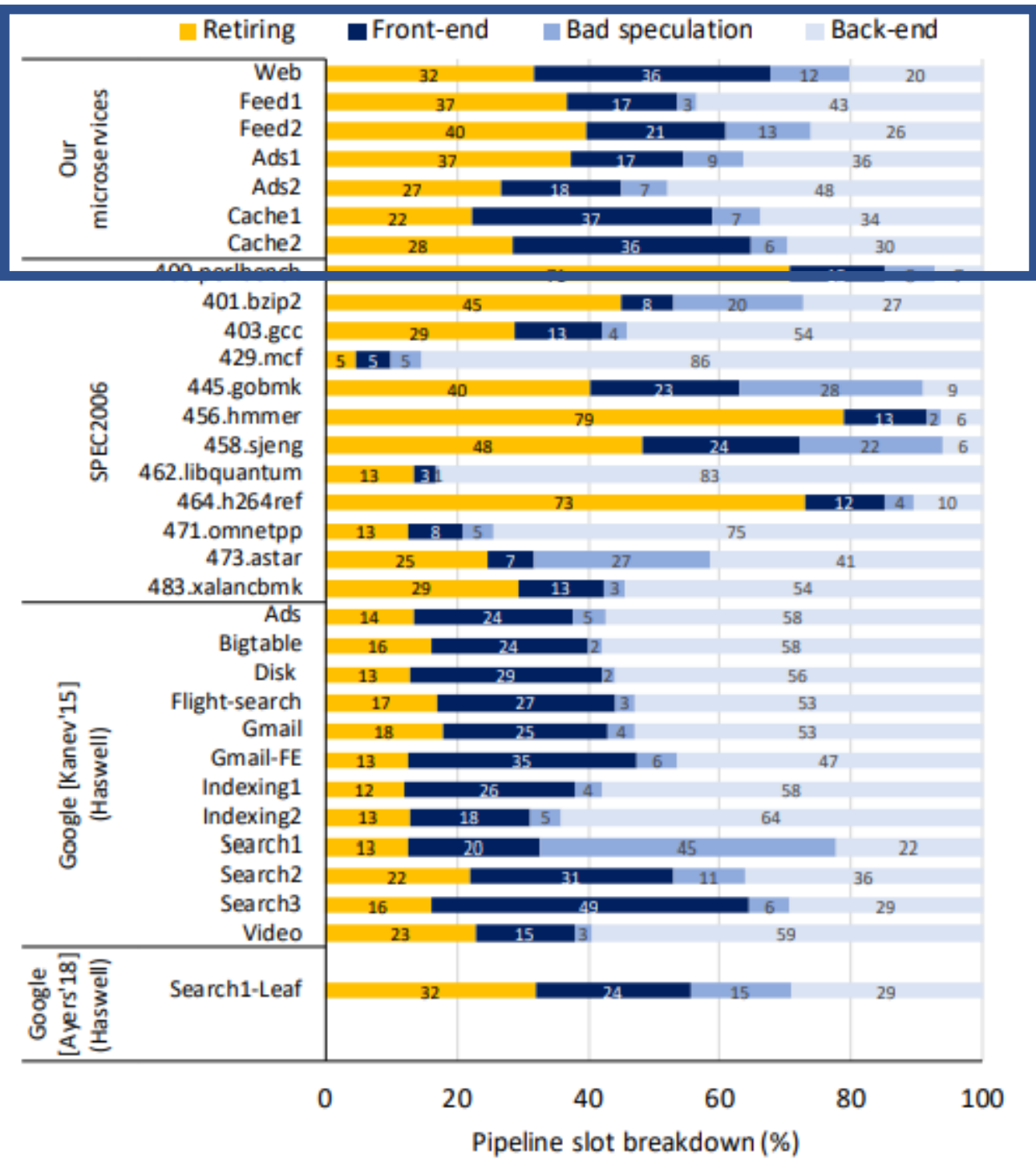


Figure 10: IPC is universally low.

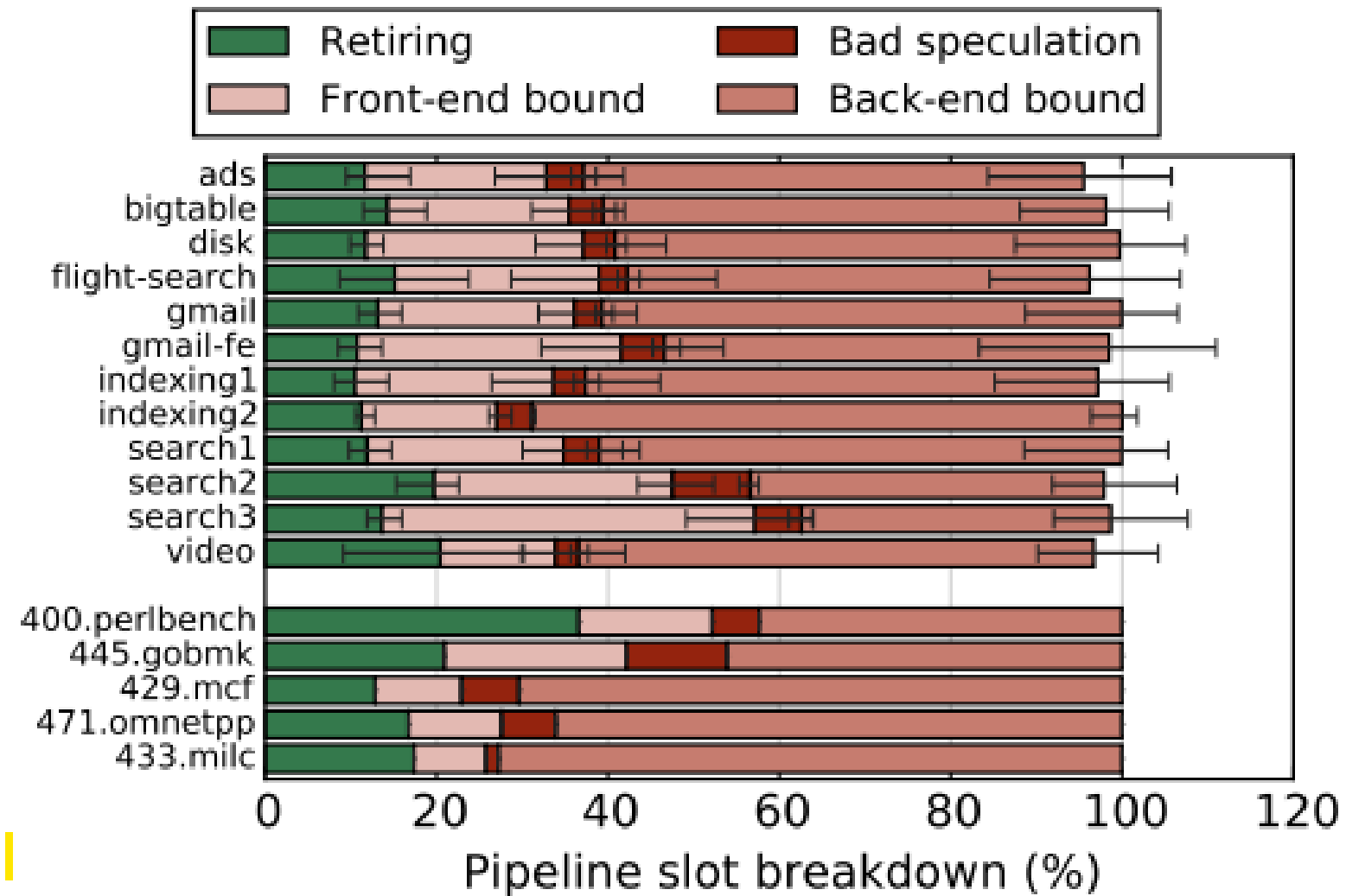


# Low Retirement Rate in Data Center

## Facebook (SMT is On)



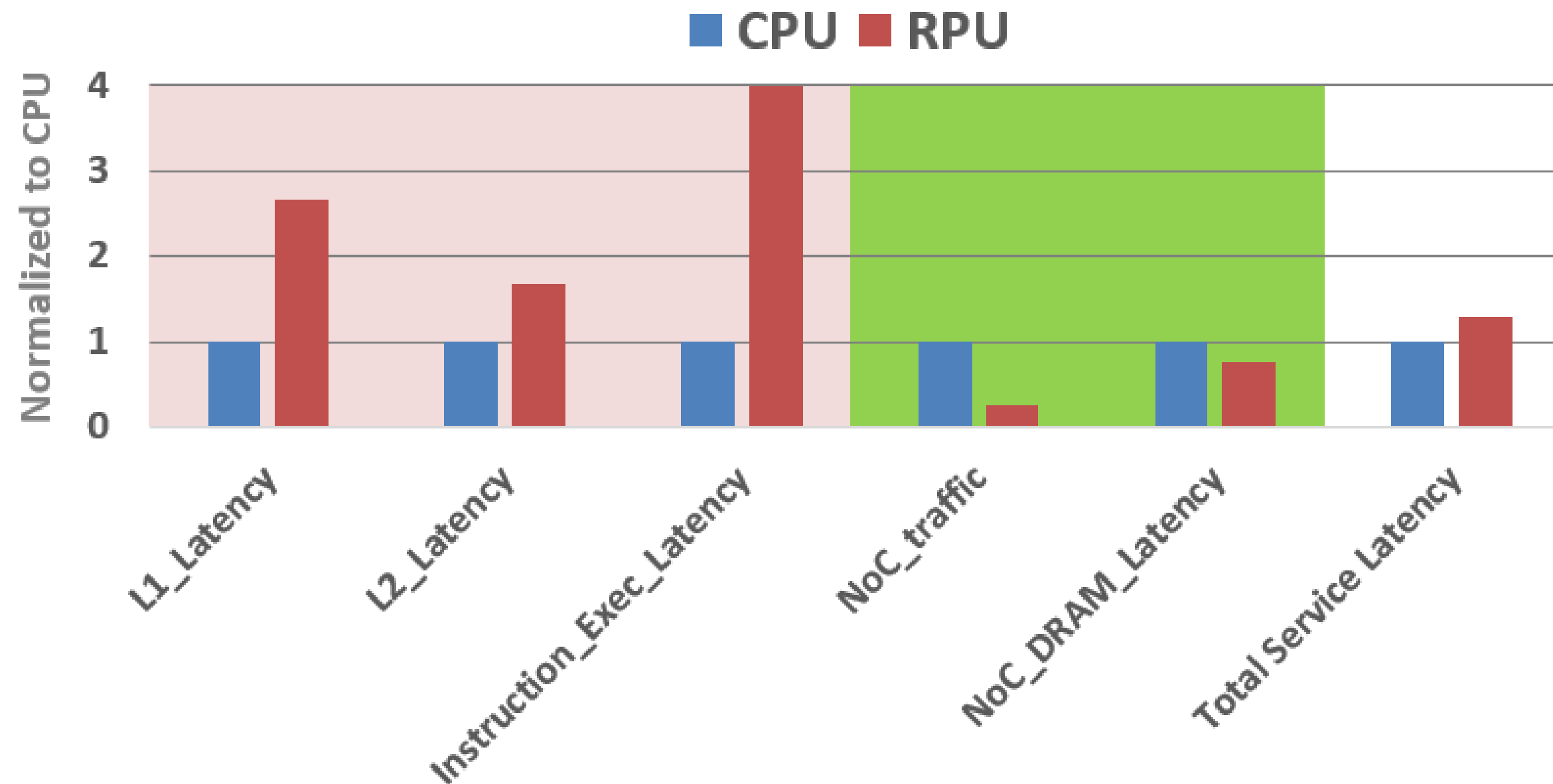
## Google (SMT is off)



**Figure 6: Top-level bottleneck breakdown. SPEC CPU2006 benchmarks do not exhibit the combination of low retirement rates and high front-end boundedness of WSC ones.**

**Retire rate = 10-25% per thread**

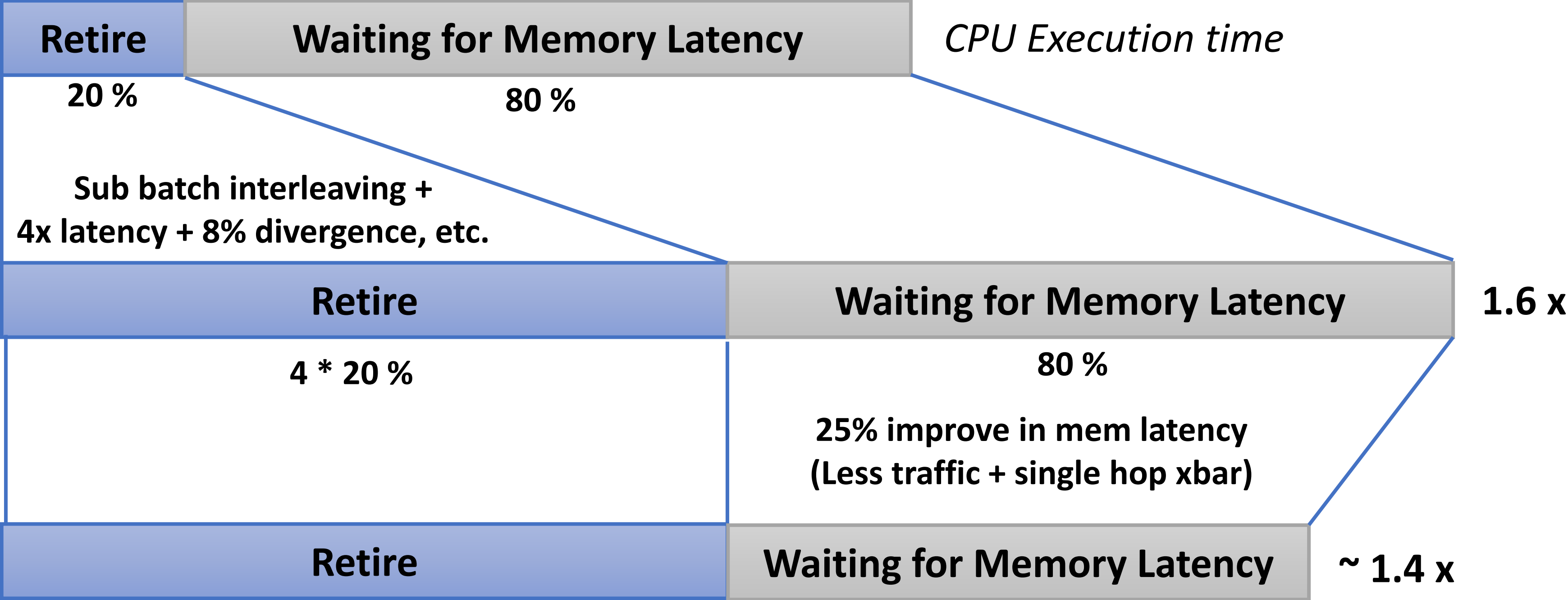
# Memory Latency Improvement



**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

# High-Level Service Latency Analysis



**Concern: Weak consistency + NMCA model is hard to program in multi-threaded applications?**

Answer: But, the data center workloads by nature does not need strong consistency model nor MCA

# 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)

# 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

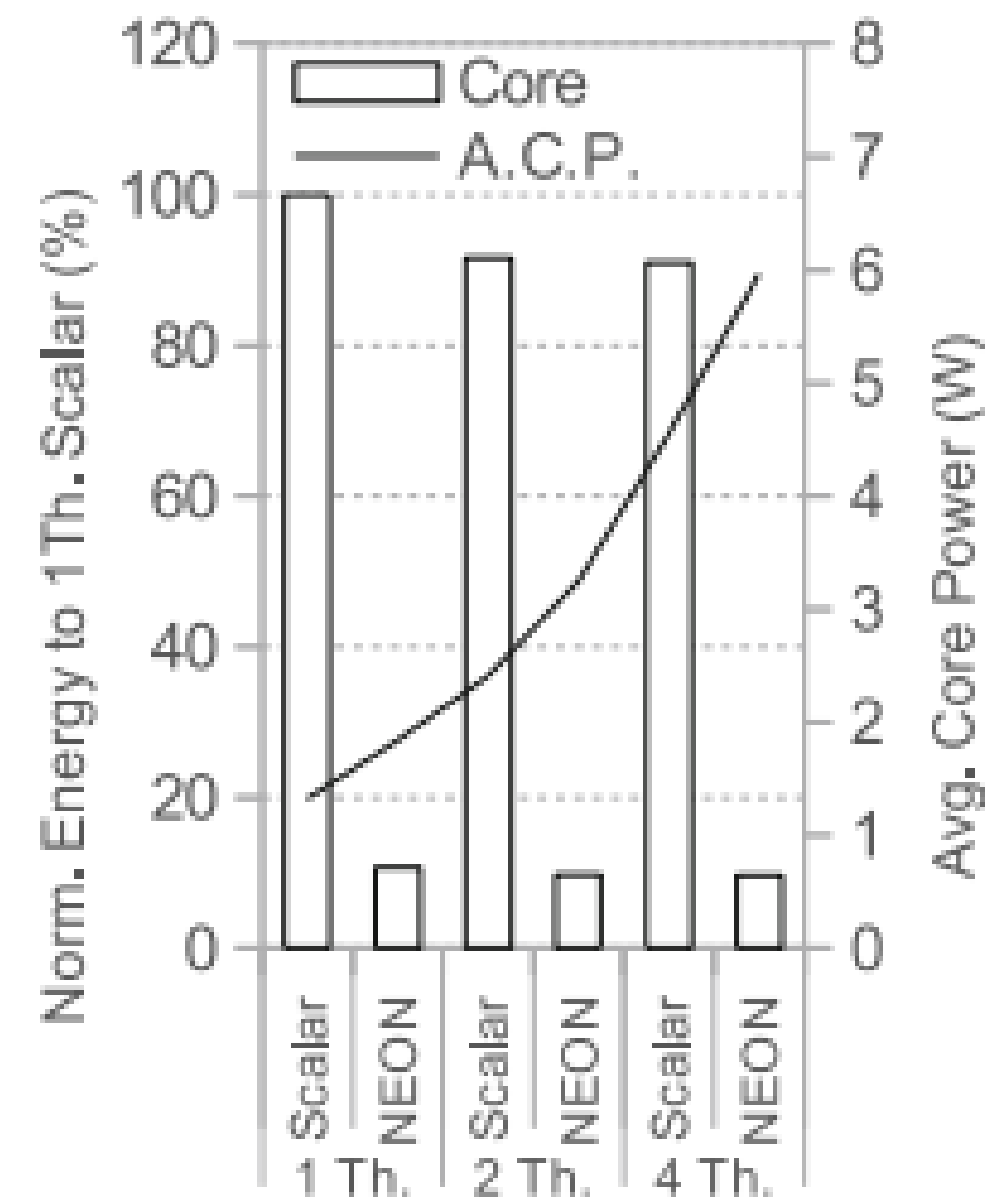
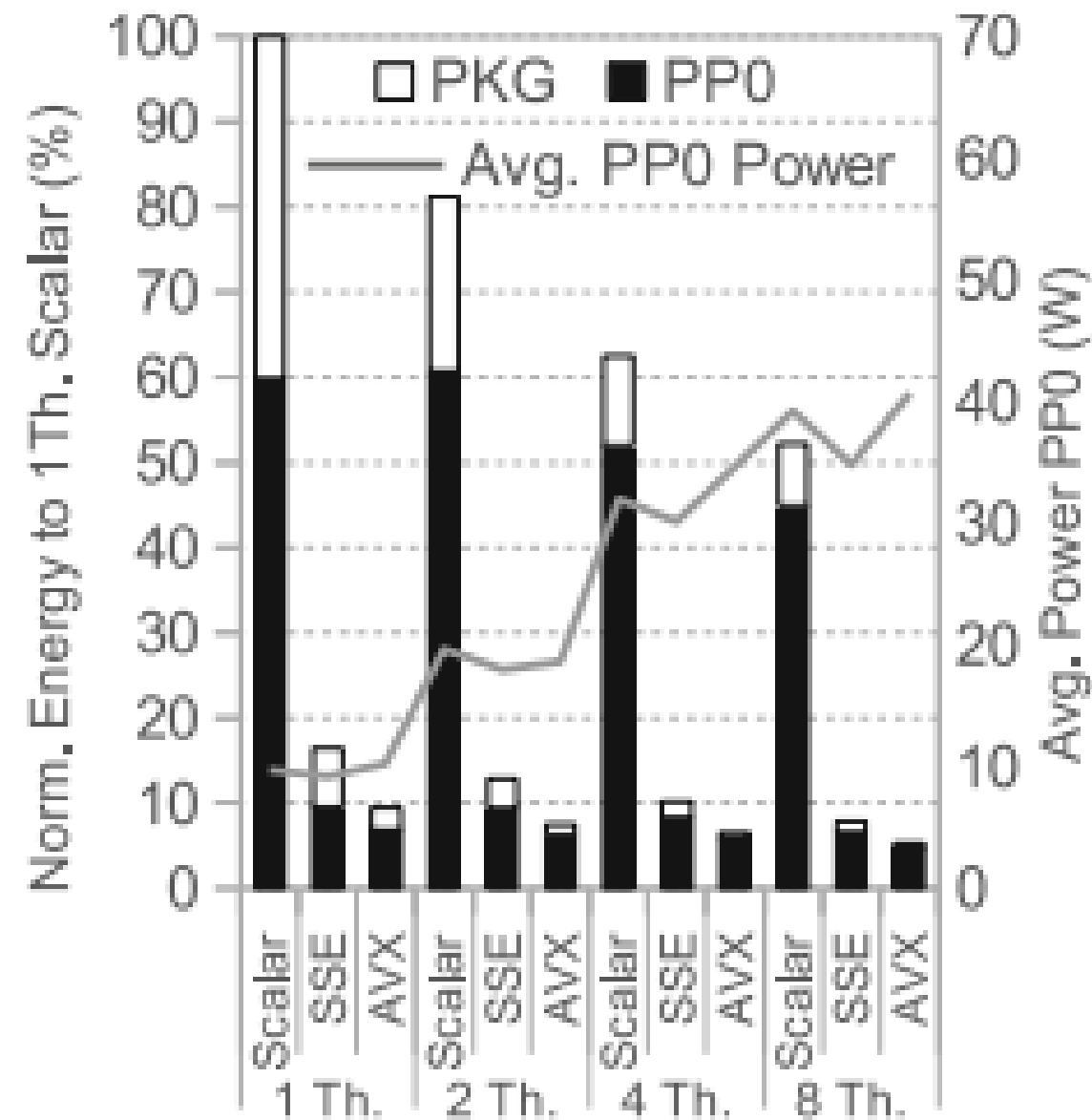
## **Concern: RPU's 5.7x energy efficiency is a big number? Be realistic!**

Answer: There has been previous work showing that they can get 8x energy efficiency from vectorizing data parallel workloads, like PARSEC and Rodinia, on real HW CPUs (if the workload is SIMD-friendly). See next slide.

So, my question is: If we can get 8x energy efficiency from vectorizing data parallel workloads, why we cannot get similar energy efficiency from vectorizing microservices?! Since both have the same concept: amortize the frontend+OoO.

I showed that frontend+OoO in serial workloads are 75% and we can also amortize up to 15% of the memory and static energy, this increases our amortization factor to 90%, apply Amdahl's law, then up to 10x energy efficiency is achievable.

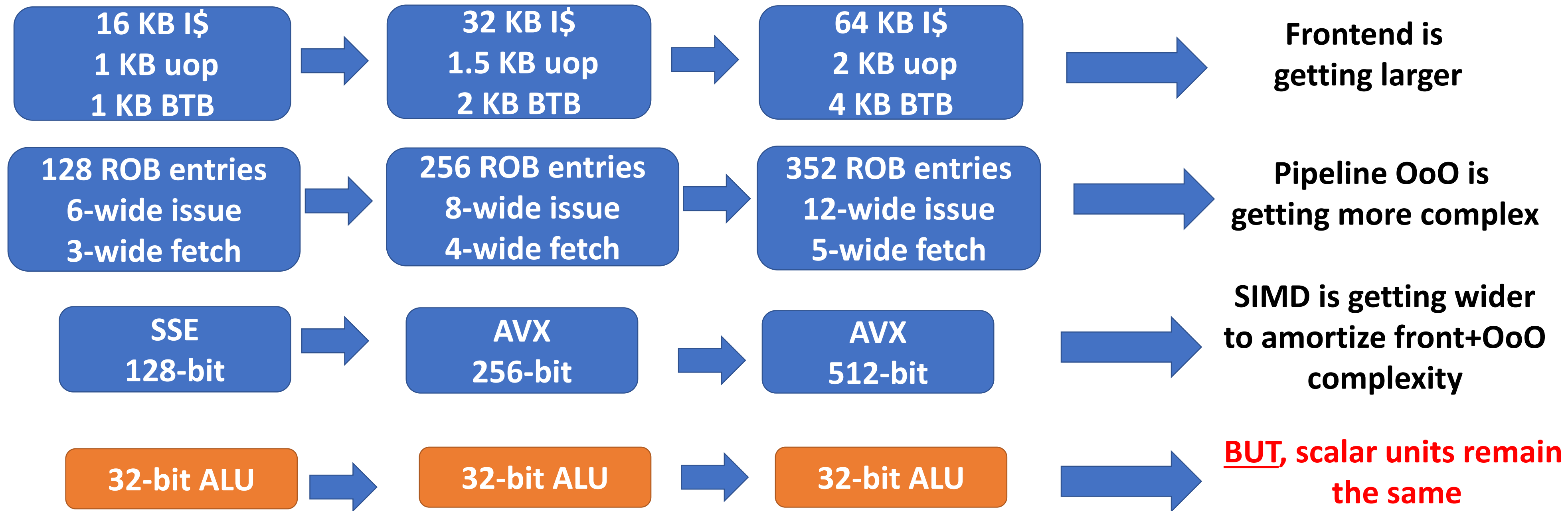
# Example: Vectorizing Blackscholes



Up to 10x energy efficiency from vectorizing on real HW Intel and ARM CPUs



# Frontend+OoO Overhead is Increasing



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

**Question: As you increase the core and caches size, the energy of cache access and data wiring will increase, the bigger the core the large energy you consume, how do you still get 5.7x energy after that?**

Answer: We show in our paper that the dynamic energy per L1 access and L2 access in RPU is higher by a factor of 1.72x and 1.82x respectively than in CPU, due to the larger cache size. However, the generated traffic reduction and other energy savings in the frontend will outweigh this energy increase as detailed in our experimental results.

# Energy Efficiency of CPU vs RPU

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

batch size ( $n$ ) = 8-32

SIMT Efficiency=92%

Amortized factors = 50-90%

Larger L1/L2 Caches, etc.

→ At high SIMT efficiency, the energy savings from the amortized metrics greatly outweigh the SIMT management overhead

## **Concern: Do you batch on TCP/IP processing? Batching on TCP/IP will cause a negative interaction with TCP congestion avoidance protocol, hurting network RTT and throughput?**

To mitigate this issue, we can bypass batching on the web server (or at least its TCP/IP processing) to send acknowledgments to the end users as soon as possible. However, we assume TCP/IP batching after that as the data center provider has full control of the network stack within the data center so they can tune/adapt the TCP congestion protocol parameters (timeout and congestion window) along with the batching timeout/size to ensure there is no negative effect on the network throughput. We did not study this in detail as it is beyond the scope of the paper. We leave this study for future work.

Other solution: we can apply batching on network traffic too. So, the RPU driver SW (with or without HW support) can detect if all the requests are sent to the same microservice in the graph (i.e., same server IP/socket). If so, send only one network request. See the below patent:

<https://patentimages.storage.googleapis.com/69/ee/89/f8fc1838fefeef/US20220050707A1.pdf>

**Concern: Large paper scope, superficial discussion and evaluation? Many details are missing, especially at system-scale?**

Answer: We agree that the scope of the paper is large. We believe that the paper opens up a wide number of interesting research directions in a relatively unexplored space. However, to prove that the space is worth exploring, an initial paper must detail a feasible full-system solution, where each technical contribution is afforded less space for discussion.

**Concern: Why SMT\_8 shows poor performance and exhibit high latency in your results (5x higher than single thread performance)?**

Answer: Recall, SMT works very well when the threads exhibit dissimilarity during execution. For example, one thread is compute bound and the other is memory-bound. In our evaluation, all the threads are running the same program/microservice (SPMD), so dissimilarity is very rare. Even more, we launch the threads at the same time making dissimilarity is even harder to exist.

In recent study from Google about in-production search service (ref is below), they show that enabling SMT\_2 on Intel Haswell CPU has led to barely 1.37x higher throughput (QPS), which indicates service latency has increased by 1.45x. Scale this number to SMT\_8 (8 threads), then service latency is expected to increase to 4.4x for SMT\_8, close to the number that I have in the paper. In fact, IBM POWER SMT\_8 scales better but I do not have enough details about IBM architecture.

**Question: In figure 11, why did select 75 batches (2400 requests) for per-argument batching? This number looks weird.**

Answer: We select this number based on the following assumptions:

Assume an online search service with 120 ms latency and receive about 100K QPS (see refs below)

Now, assume a batch overhead is 20% (close to previous work assuming batching for power management)

So, batching window =  $0.2 * 120 \text{ ms} = 24 \text{ ms}$

100K QPS means 100 reqs per 1 ms

So reqs received per batching window =  $24 * 100 = 2400 \text{ reqs}$

**Question: Do you plan to open source your infrastructure?**

Answer: We plan to open source our workloads and the PIN-based tool (SIMTec) we used to calculate the SIMT efficiency and generate the traces.

For Accel-sim, uQsim and McPAT, they re already open sourced.



**Question: Why did you use Accel-Sim not gem5? Where did you get your CPU configuration from. Your CPU configuration looks impractical?**

Answer: We select the simulation tools that we are familiar and more productive with. In academia, the researcher is free to select his own simulation infrastructure. After all, the industry does not trust open-source simulators.

We configure our CPU to be similar to previous work and close enough to industrial designs.

**Question: The rise of serverless computing has made multi-process and containerized-based microservices more commo. How do you handle multi-process services in this case?**

Answer: In multi-process services, the separate virtual address spaces can cause both control flow and memory divergence. We believe that with user-orchestrated inter-process data sharing and some modifications to the RPU's virtual memory these effects can be mitigated. However, since the contemporary services we study are all multi-threaded, we leave such a study as future work.

## **Concern: Security Implications? The grouping of concurrent requests for SIMT execution may enable new vulnerabilities**

Answer: Two security breaches may exist:

(1) a malicious user may generate a very long query that could affect the QoS of other short requests. Such attacks can be mitigated in our input size-aware batching software by detecting and isolating maliciously long requests. If this does not work, our run-time batch split technique can be used as needed

(2) Another security vulnerability is the potential for parallel threads to access each other's stack data (exploiting the fact that threads' stack data are adjacent in the physical space). However, as described in Section III-B2, the RPU's address generation unit is able to identify inter-thread stack accesses and throw an exception if such sharing is not permitted.

## **Pitfall: The RPU can be bottlenecked by I/O throughput**

Answer: we demonstrate that the off-chip memory bandwidth will dramatically scale in future years with the introduction of DRR5, DDR6, and HBM in the data center. We observe similar trends for I/O standards like PCIe5 and PCIe6 [137]. 128x PCIe6 lanes per single socket can provide 2 TB/sec of bidirectional I/O bandwidth. Ethernet 400/800 Gb/sec and recent NVMe interface advances will enable substantial network and storage throughput improvements.

# PCIe I/O Scaling

Interconnect Standard	Width	Transfer Rate	Effective Bandwidth	Year Introduced
ISA	16-bit	8MHz	8.33MB/s	1984
PCI	32-bit	33MHz	133MB/s	1993
PCIe 1.0	up to 16 lanes (consumer)	2.5GHz	8GB/s	2004
PCIe 2.0	up to 16 lanes (consumer)	5GHz	16GB/s	2008
PCIe 3.0	up to 16 lanes (consumer)	8GHz	32GB/s	2011
PCIe 4.0	up to 16 lanes (consumer)	16GHz	64GB/s	2019
PCIe 5.0	up to 16 lanes (consumer)	32GHz	128GB/s	2020 - 2021 (Est)
PCIe 6.0	up to 16 lanes (consumer)	64GHz	256GB/s	2022 or later (Est)

## **Pitfall: SIMR requires a lot of changes**

Answer: We design the SIMR system so that the SW stack changes are as minimal as possible. Only the HTTP server, HW and some OS system calls are required to change in the software stack while we keep the programming interface, compiler, runtime, and ISA unaltered. In fact, data center providers adopted DL accelerators [9], [46] and changed the entire software stack for similar outcomes and efficiency.

## Question: SIMT vs SMT vs SIMD

Answer: In SMT, the entire CPU pipeline is partitioned among the simultaneous threads. Threads on the same core are executed independently, which fails to exploit thread similarity and increases single thread latency. SIMT avoids all of these issues exploiting the fact that threads are running the same instruction stream and allocate the entire pipeline resources for the same task.

In SIMD, It is up to the programmer/compiler to express the data parallelism in SIMD vector ISA. However, in SIMT, the programmer can still write the programs in scalar ISA and the HW will detect convergence opportunities and handle control and memory divergence transparently.

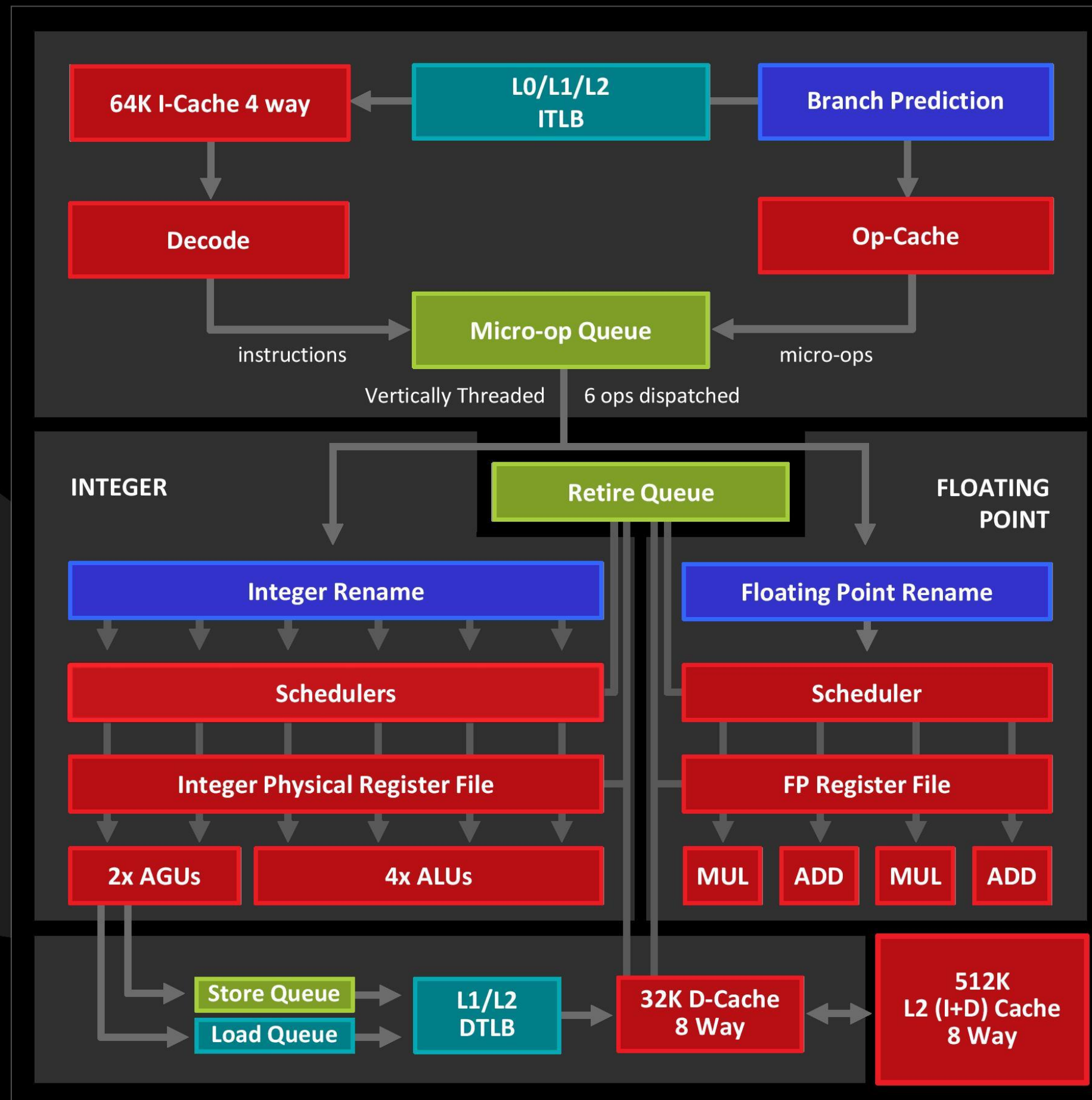
See next slides.



## SMT OVERVIEW

- ▲ All structures fully available in 1T mode
- ▲ Front End Queues are round robin with priority overrides
- ▲ Increased throughput from SMT

- Competitively shared structures
- Competitively shared and SMT Tagged
- Competitively shared with Algorithmic Priority
- Statically Partitioned





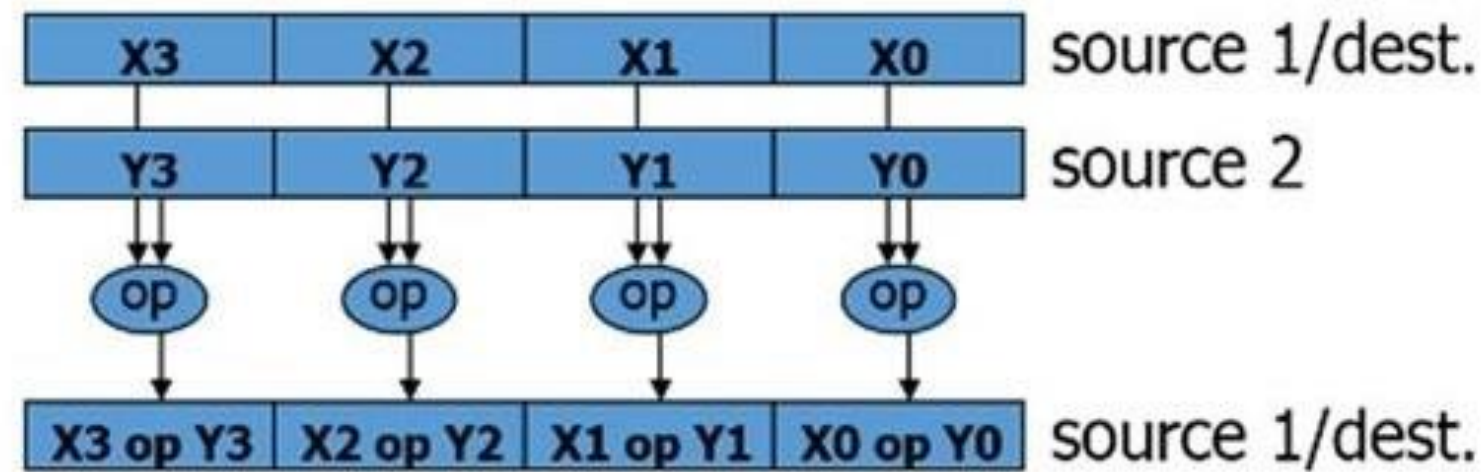
# SIMD vs SIMT

## CPU

### SIMD

1 instruction – multiple data

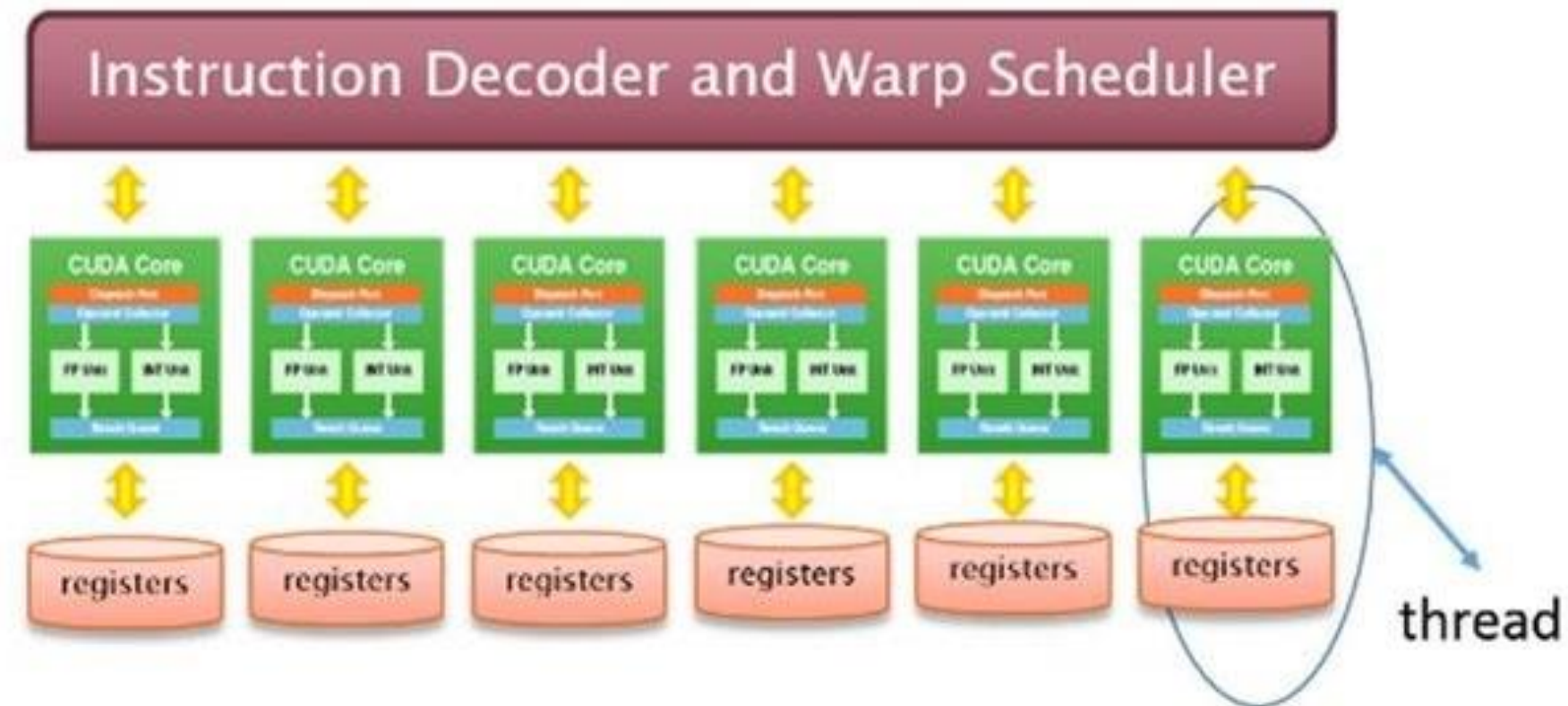
SSE2/3/4 – Neon – AltiVec  
AVX – AVX2...



## GPU

### SIMT

1 instruction – multiple threads



*Thank You!*

If you do not find your question here, feel free to  
contact us