Processing-in-Memory: Theory & Practice

Phillip B. Gibbons

Carnegie Mellon University

EMERALD Workshop

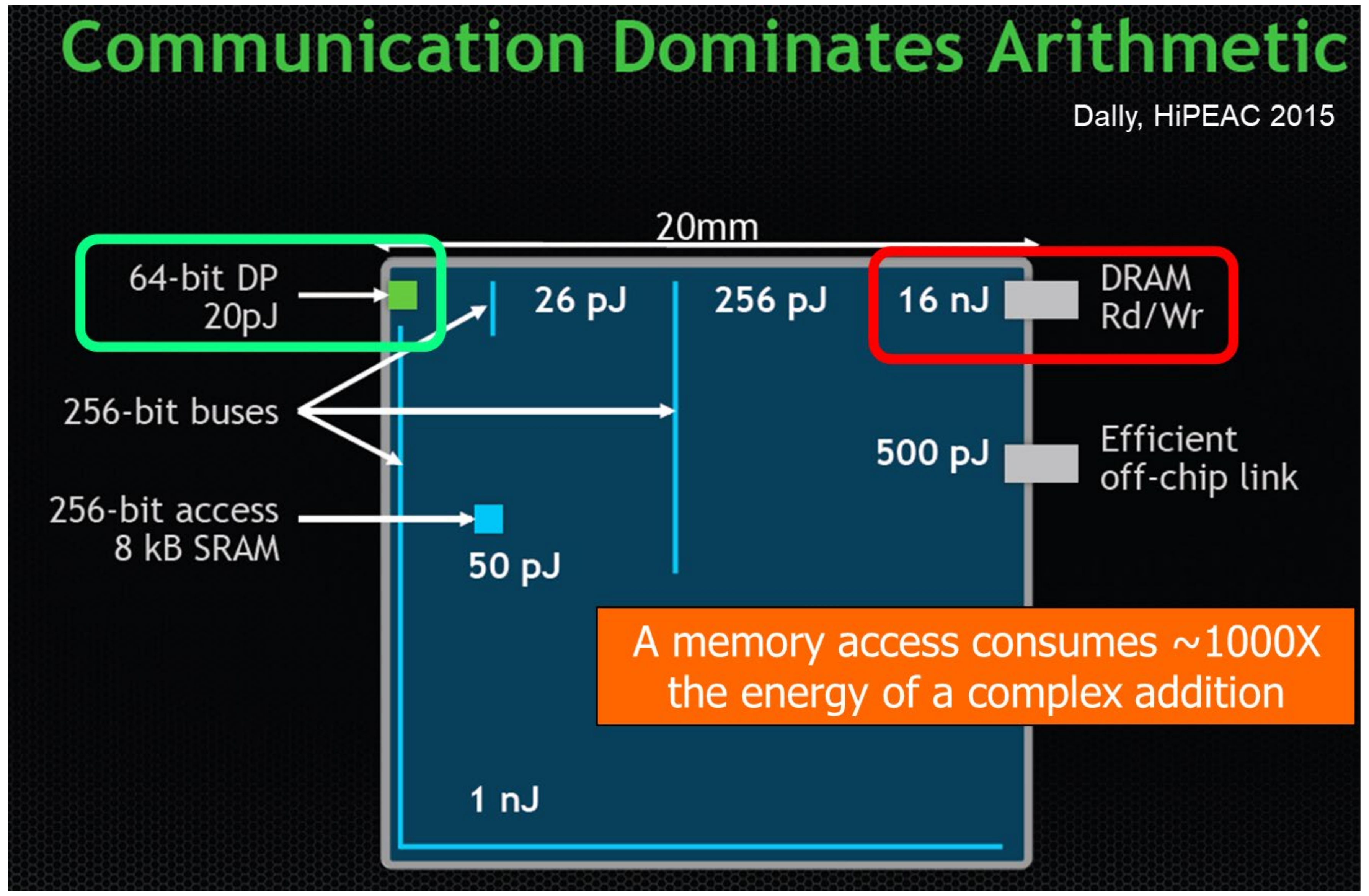
21 June 2024

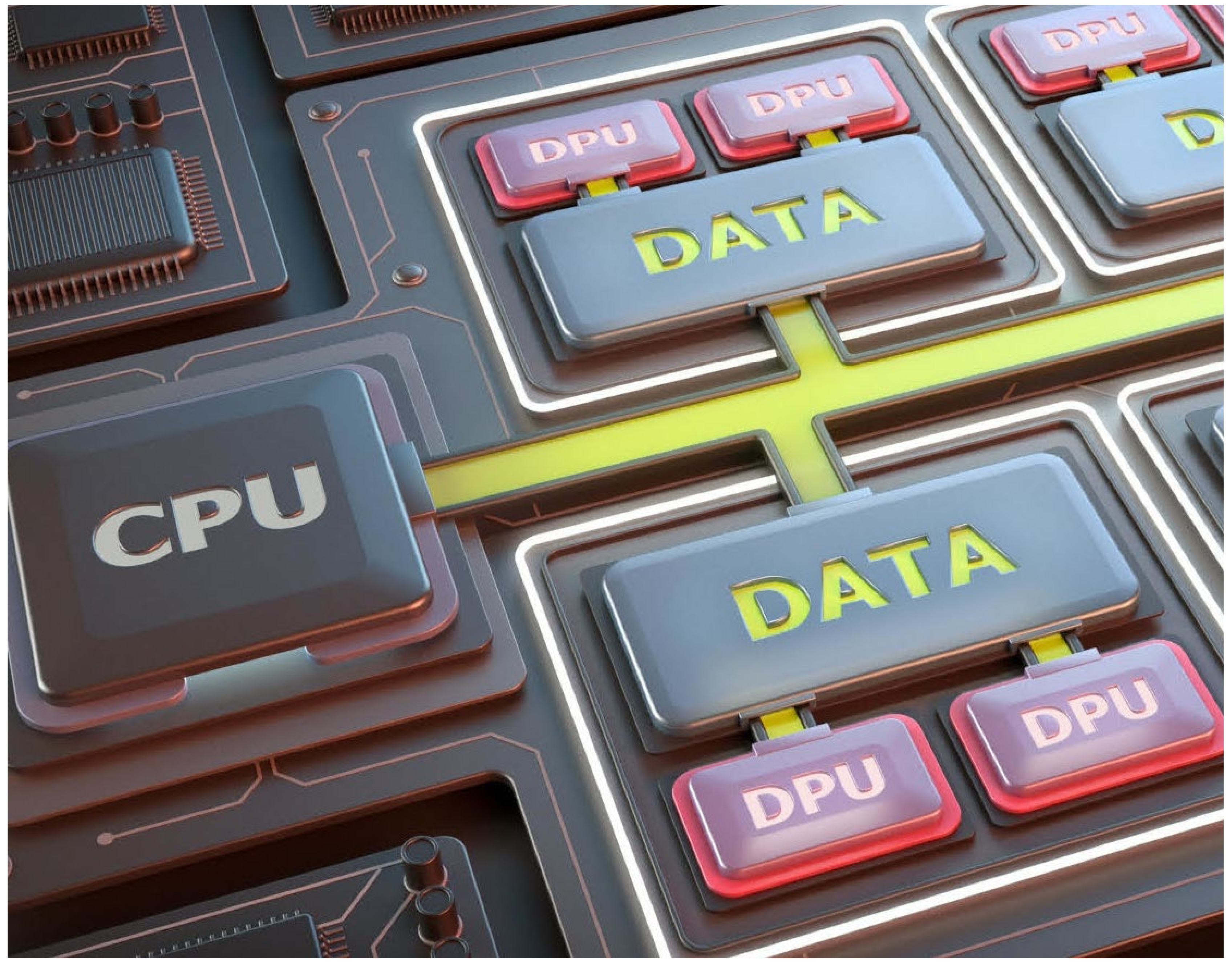
Why Processing-in-Memory

- Data movement is often the dominant cost.

- Processing-in-memory
  (a.k.a., near-data-processing)
  enables compute to be pushed to memory.

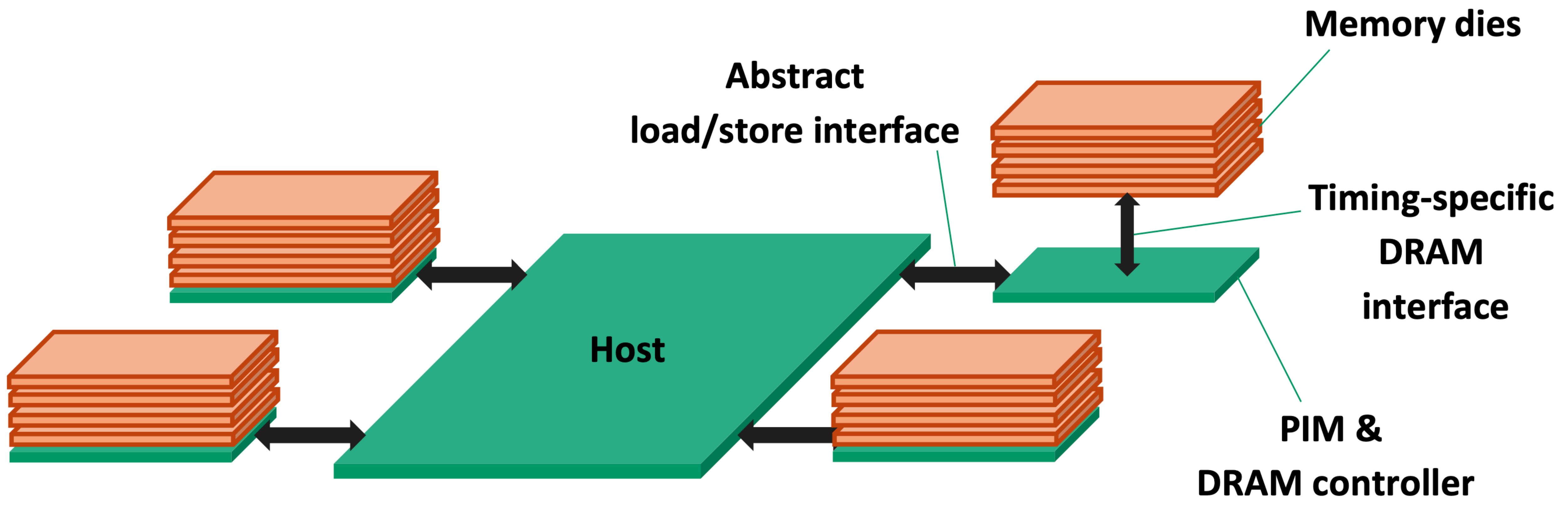
  => Saves data movement.

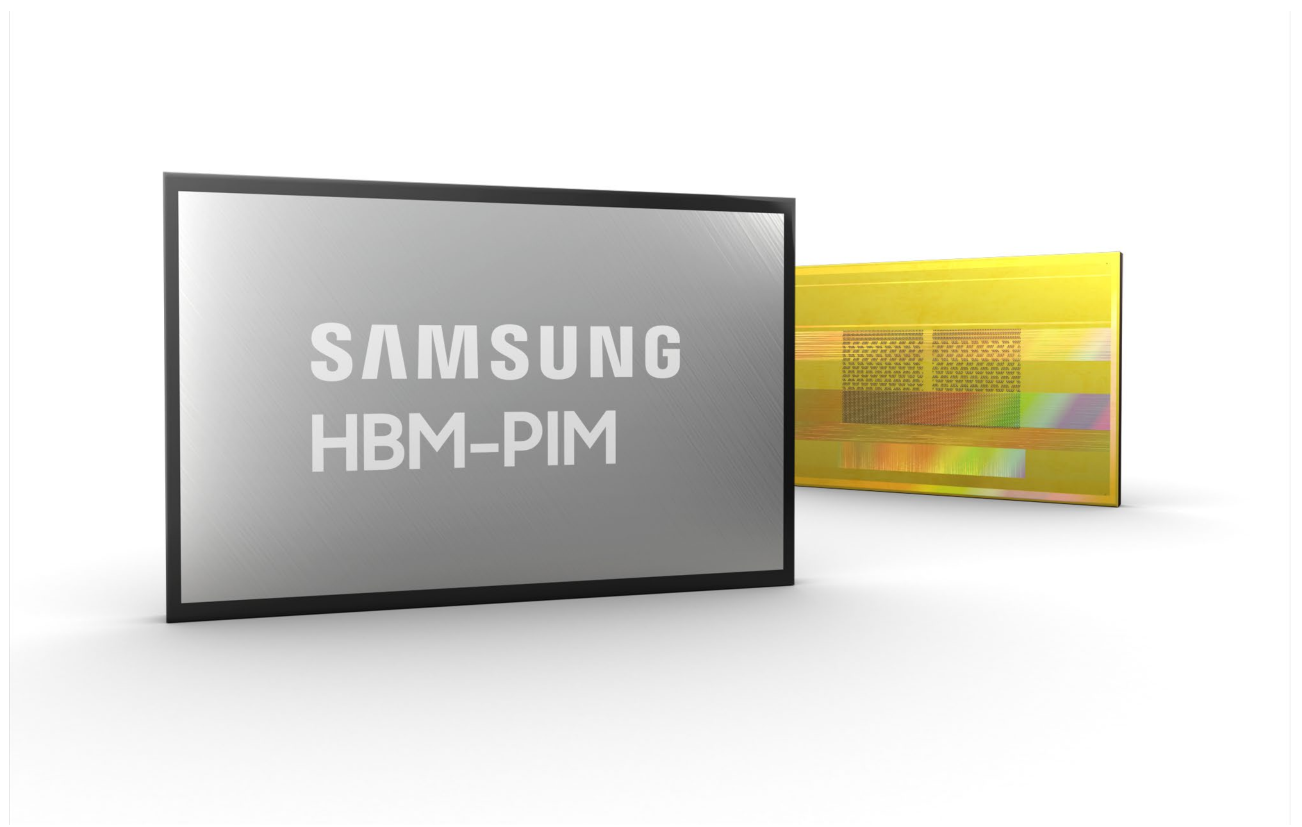


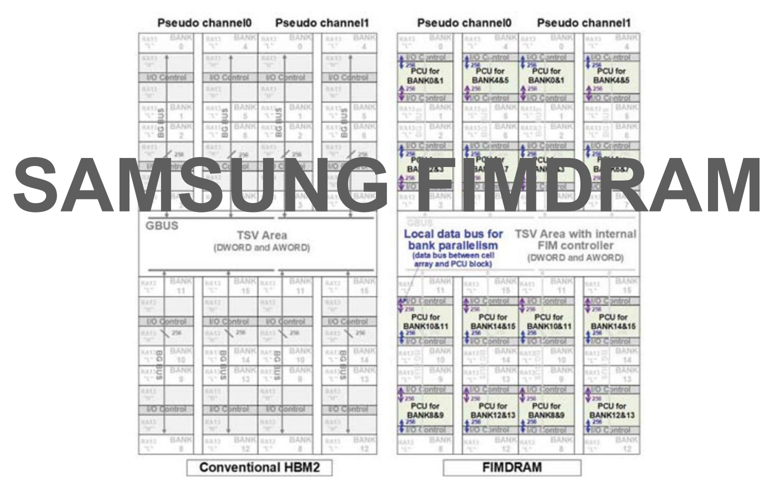


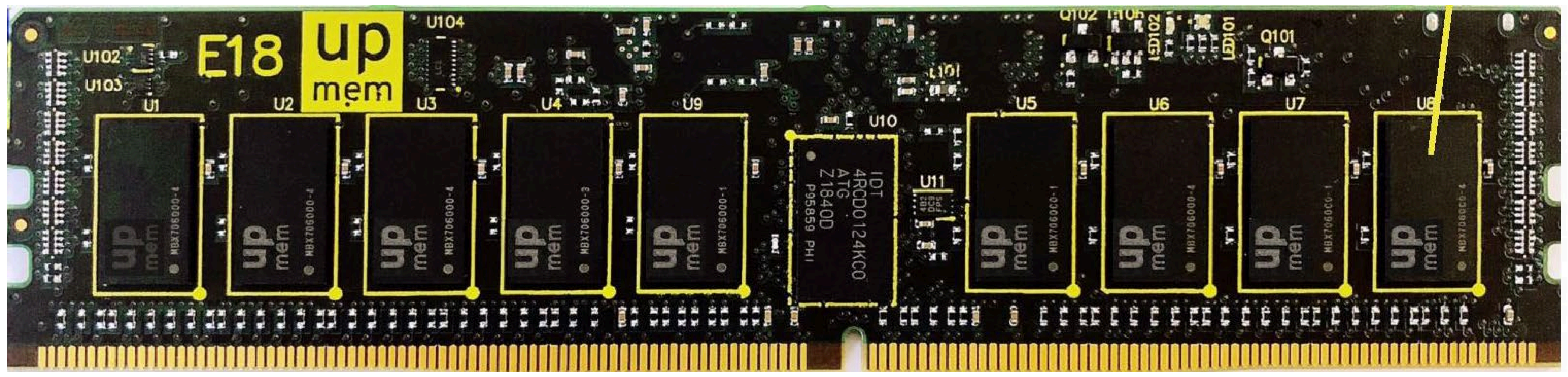
Why Processing-in-Memory now

- Idea back to 1970
- New 3D die-stacked memory cubes
- Low-latency, high-bandwidth local memory access









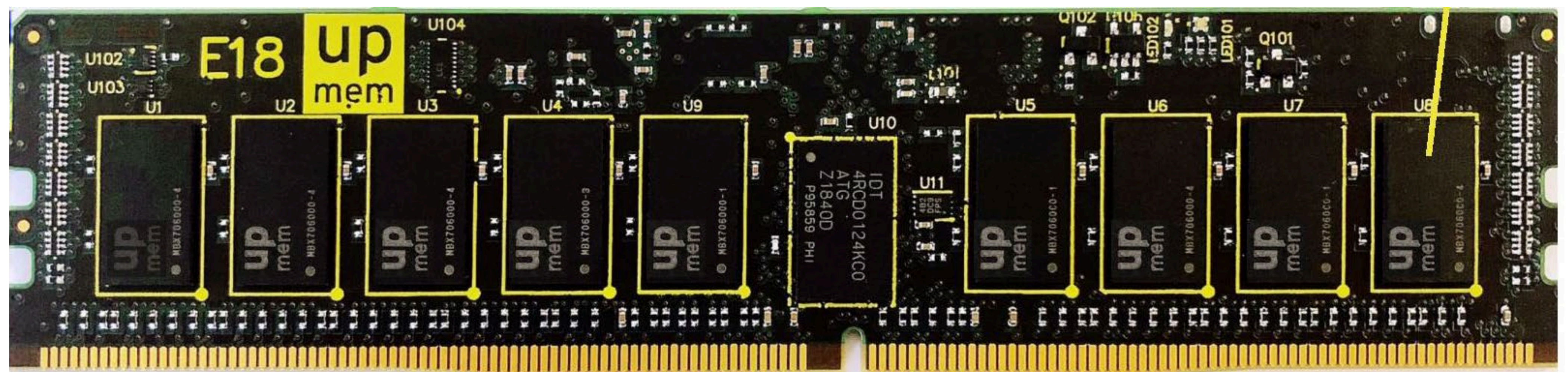
Foundations of PIM (?)

Amount of recent (2010-2020) PIM research on:

- Systems / architecture / technology side? 100+ papers
- Theory / algorithms side? ~3 papers (as of 2020)

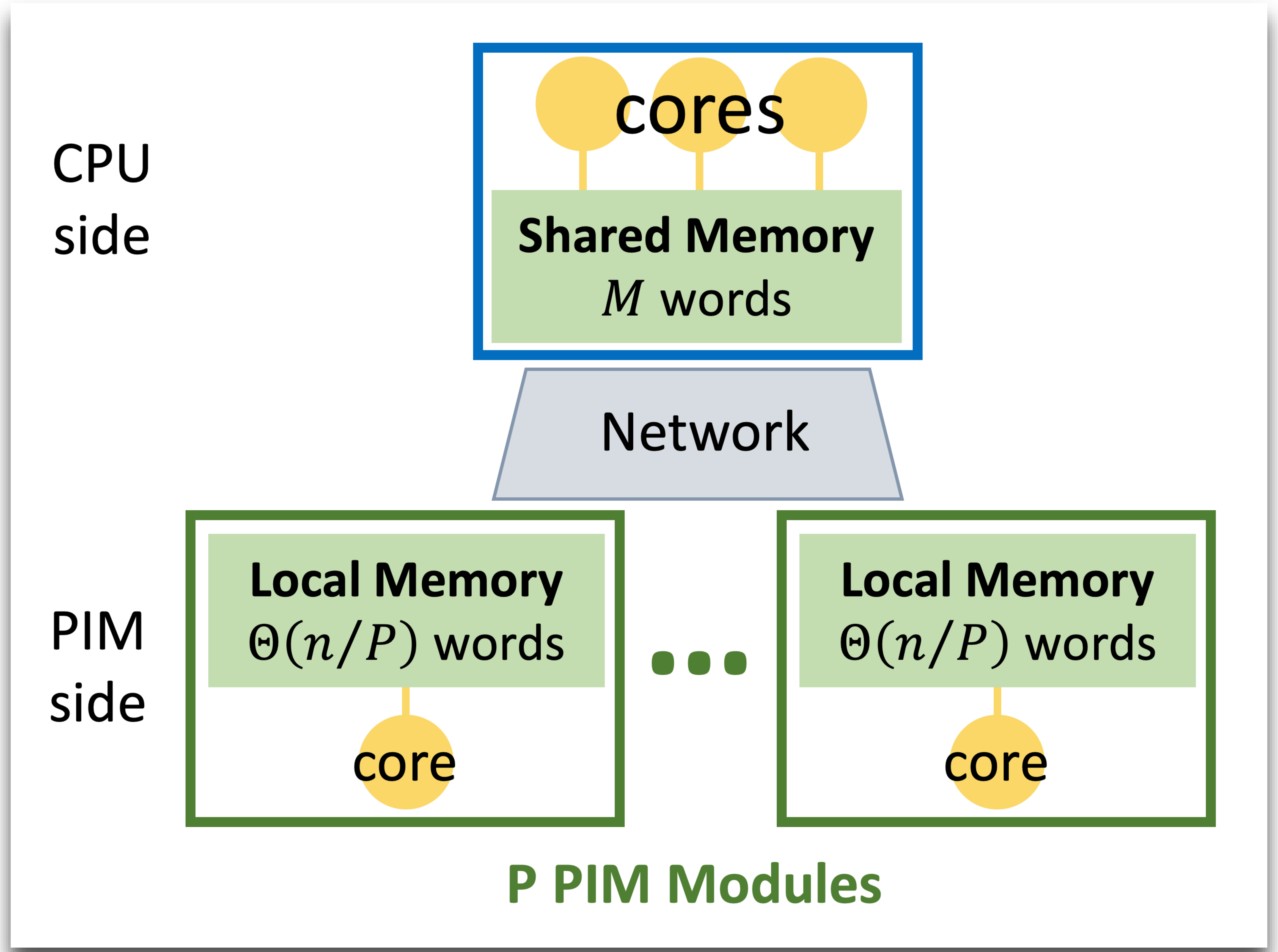
Generic PIM System

- Host CPU side:
  - Parallel cores with shared cache
- PIM side:
  - Many PIM modules, each with a core & a memory
- All communication between components go through the network.



The Processing-in-Memory Model

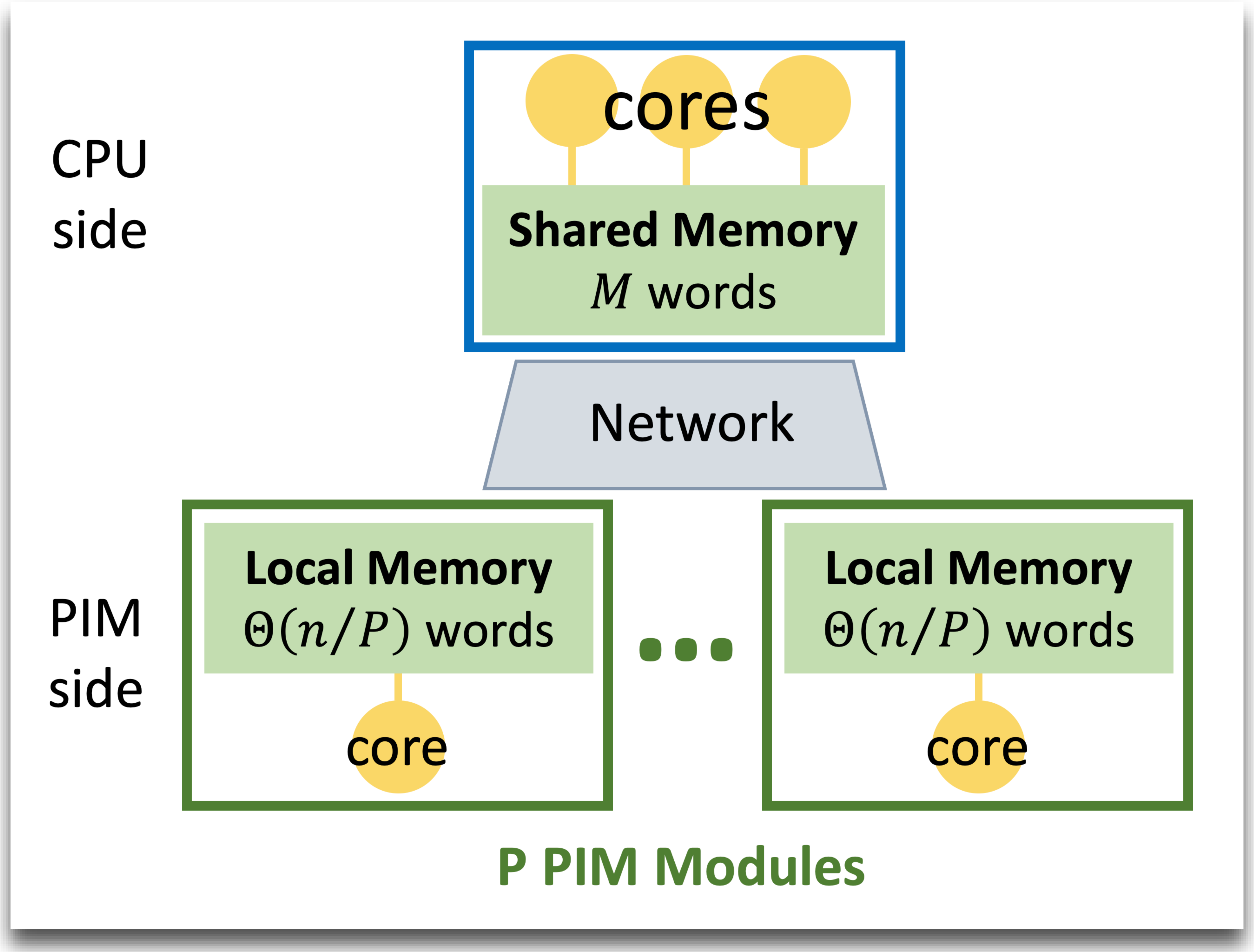
- CPU side:
  - Parallel cores with shared cache of size $M \ll n$ words
- PIM side:
  - P PIM modules, each with a core & a memory of size $\Theta(n/P)$ words
- All communication between components go through the network.



n = total memory size

Computationally, what is distinctive about PIM?

- Strawman 1: Treat entirely as a shared-memory model
  - Emulate shared memory on PIM modules, using hashing
  - But would make all memory accesses non-local!
- Strawman 2: Treat entirely as a distributed-memory model
  - Ignore the available shared memory
  - But would lose the potential benefits (we show) of using the shared memory



Distributed Memory with a powerful Shared Memory front-end
(Bulk-synchronous offload of compute)

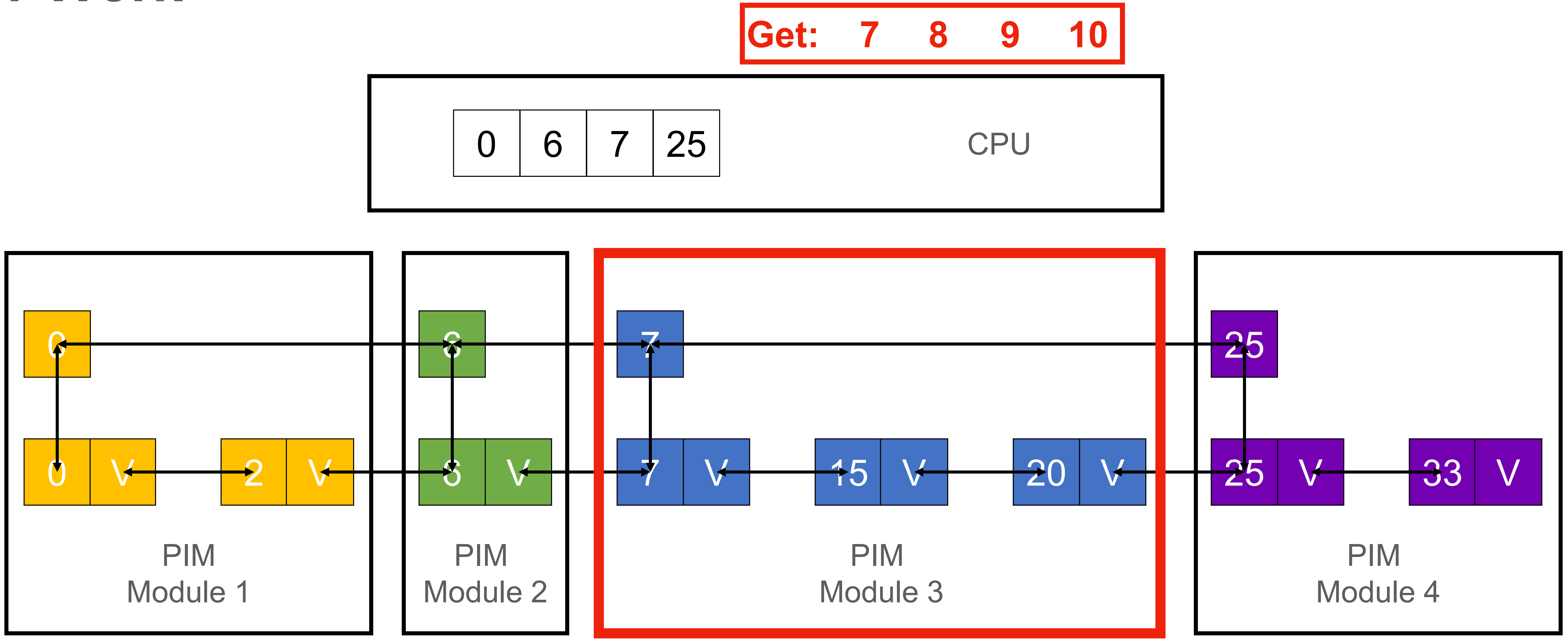
Pushing Compute to Memory

Inherent tension between

- Minimizing communication
- Achieving load balance
- (Without replication causing a blow up in space and/or update costs)

Example: Range Partitioned Index

Prior Work

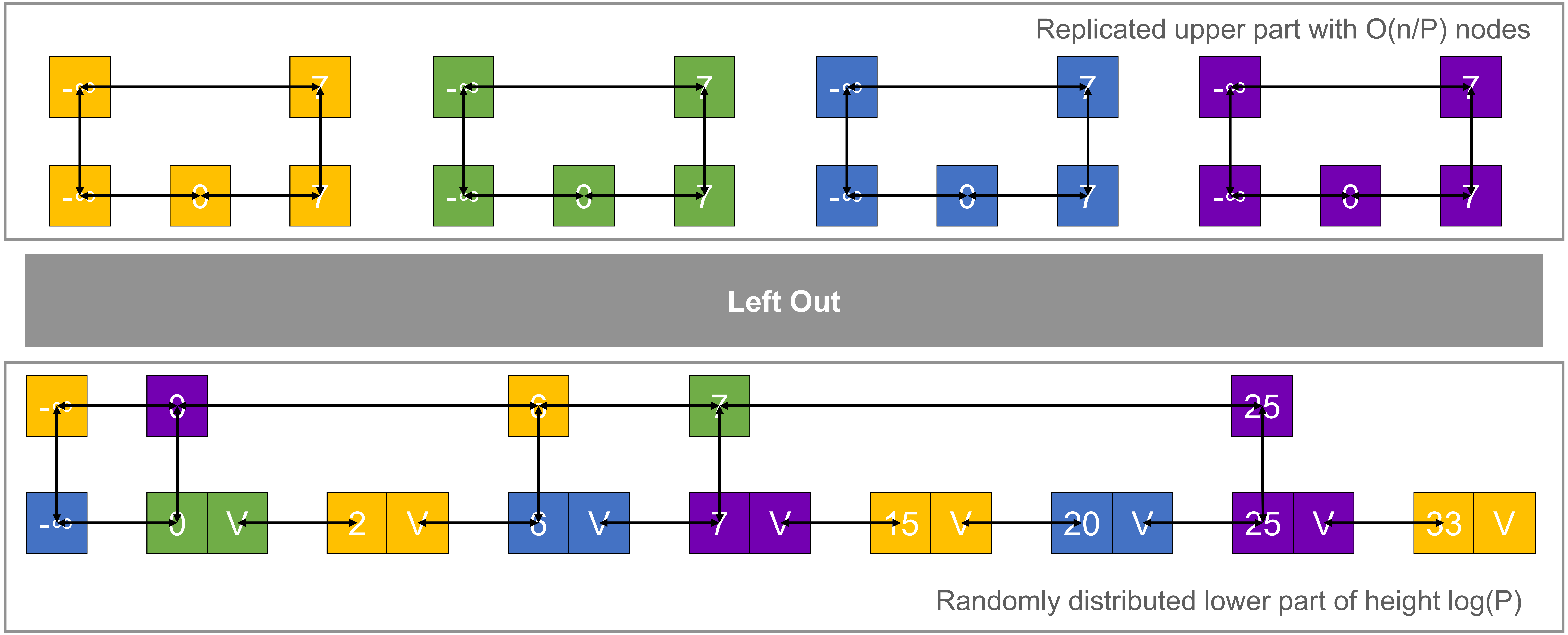


Bottleneck: Fully serializes the batch of Get operations

Overcoming bottleneck: Need smart mix of randomization & replication

Our Approach: PIM-Tree (simplified view)

Assuming 4 PIM modules (P = 4)



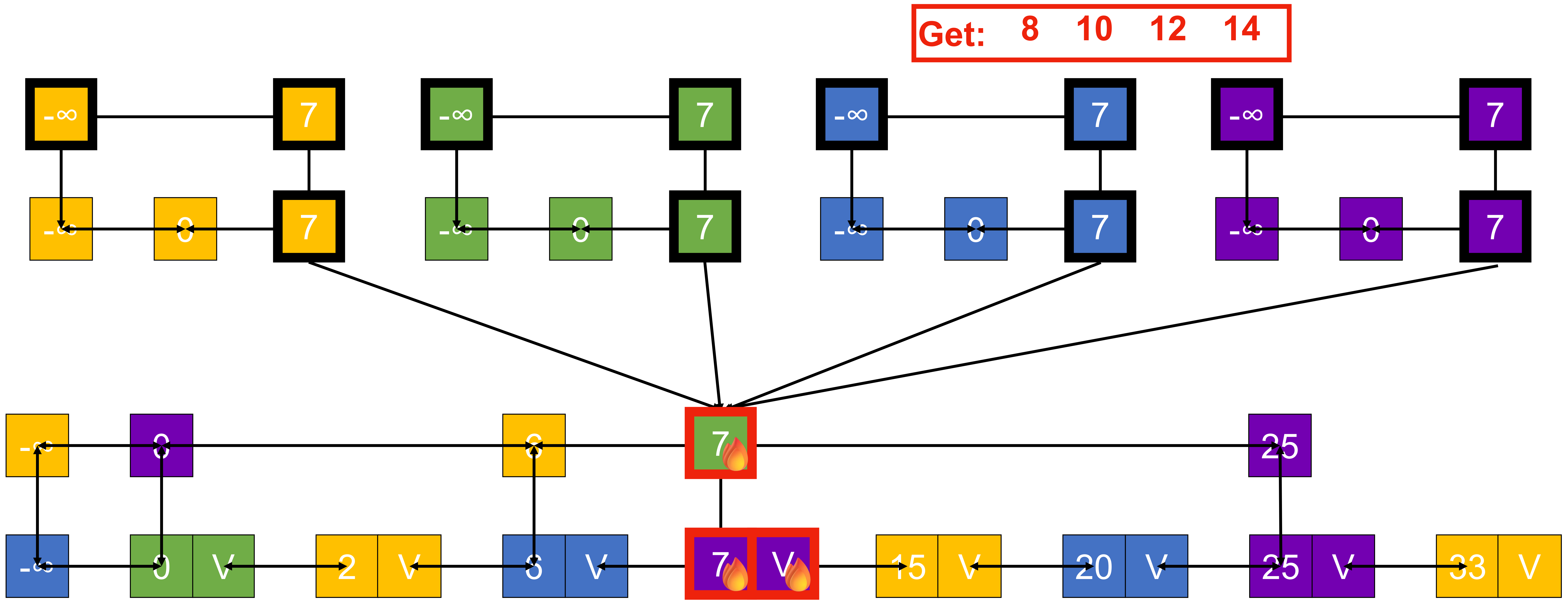
Replicated upper part with O(n/P) nodes
-∞
7
-∞
0
7
Left Out
-∞
0
6
7
25
-∞
0
V
2
V
6
V
7
V
15
V
20
V
25
V
33
V
Randomly distributed lower part of height log(P)

[Kang, Zhao, Blelloch, Dhulipala, Gu, McGuffey, Gibbons VLDB'23 Best Paper Runner-up]

Batch-parallel execution of adversarial batches

But Load Imbalance Persists

Predecessor Search

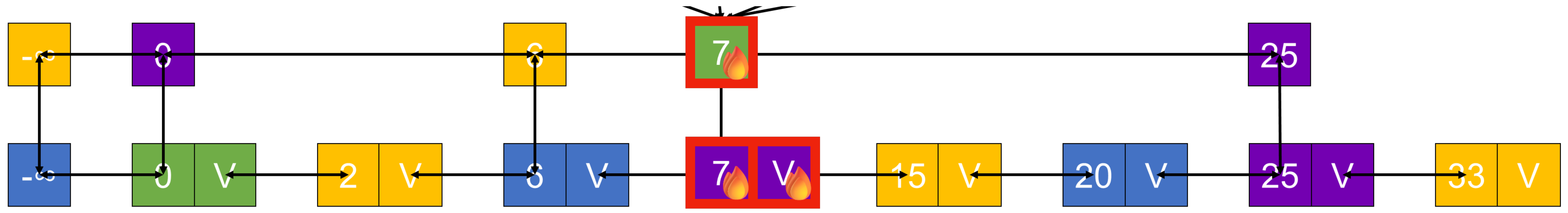


Key Insight: Push-Pull Search

Independently for each PIM module:
CPU side dynamically decides whether to

- push queries to the PIM-tree node or
- pull the node's keys back to the CPU

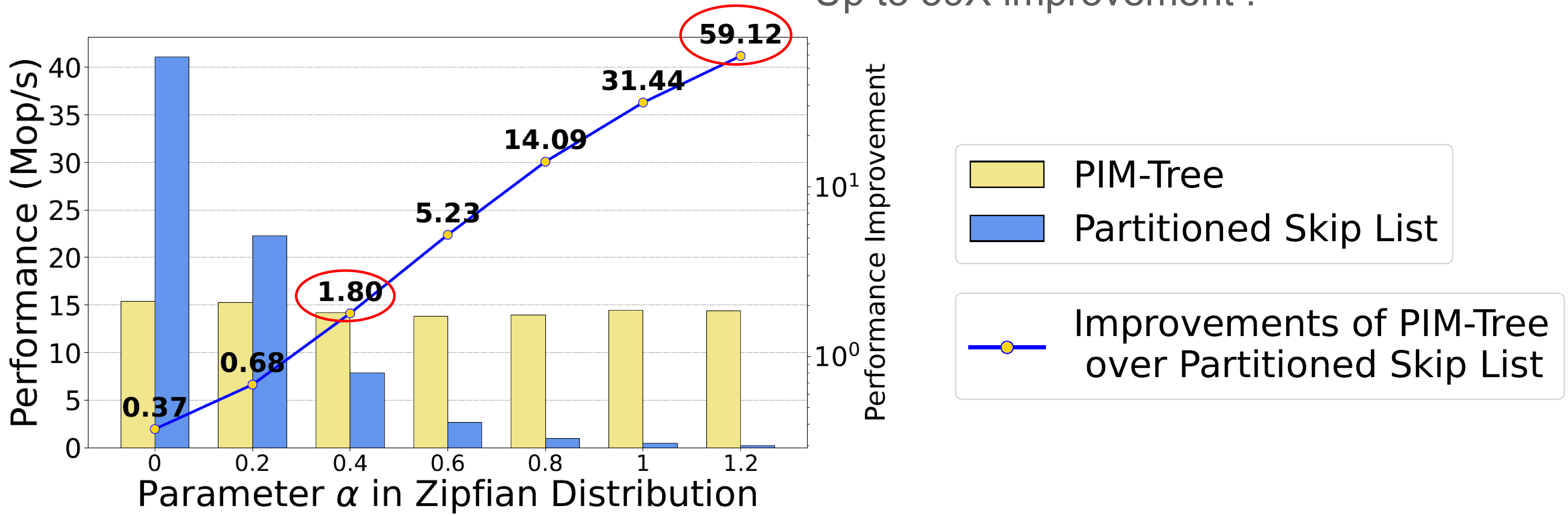
…based on which moves less data



Host CPU is great for hot spots over limited memory: So use it!

Results Against Prior PIM-based Indexes

Up to 59X improvement !

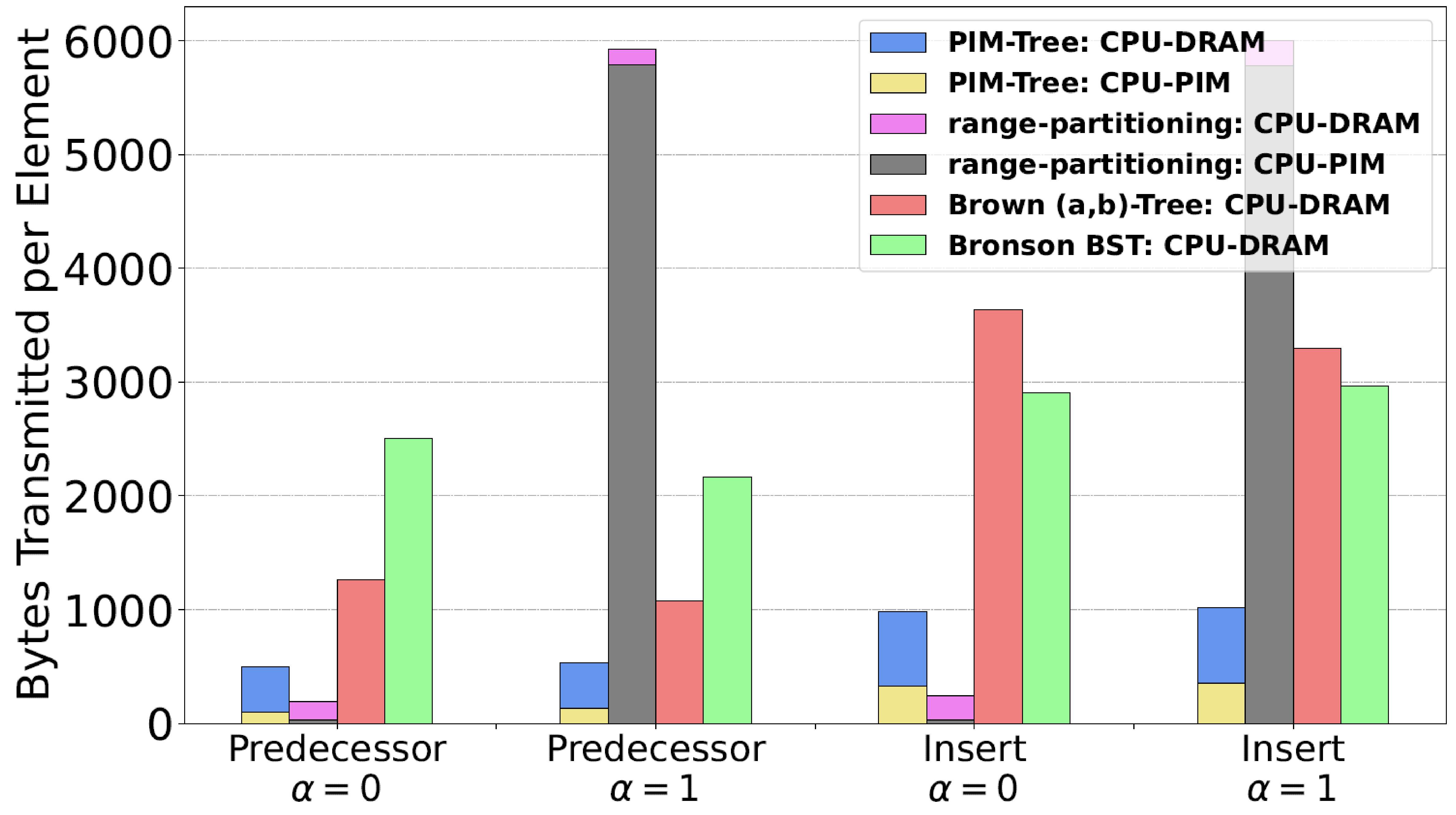


Pred() Throughput on Zipf workloads on UPMEM

Plus: Good Asymptotic Bounds on all Metrics

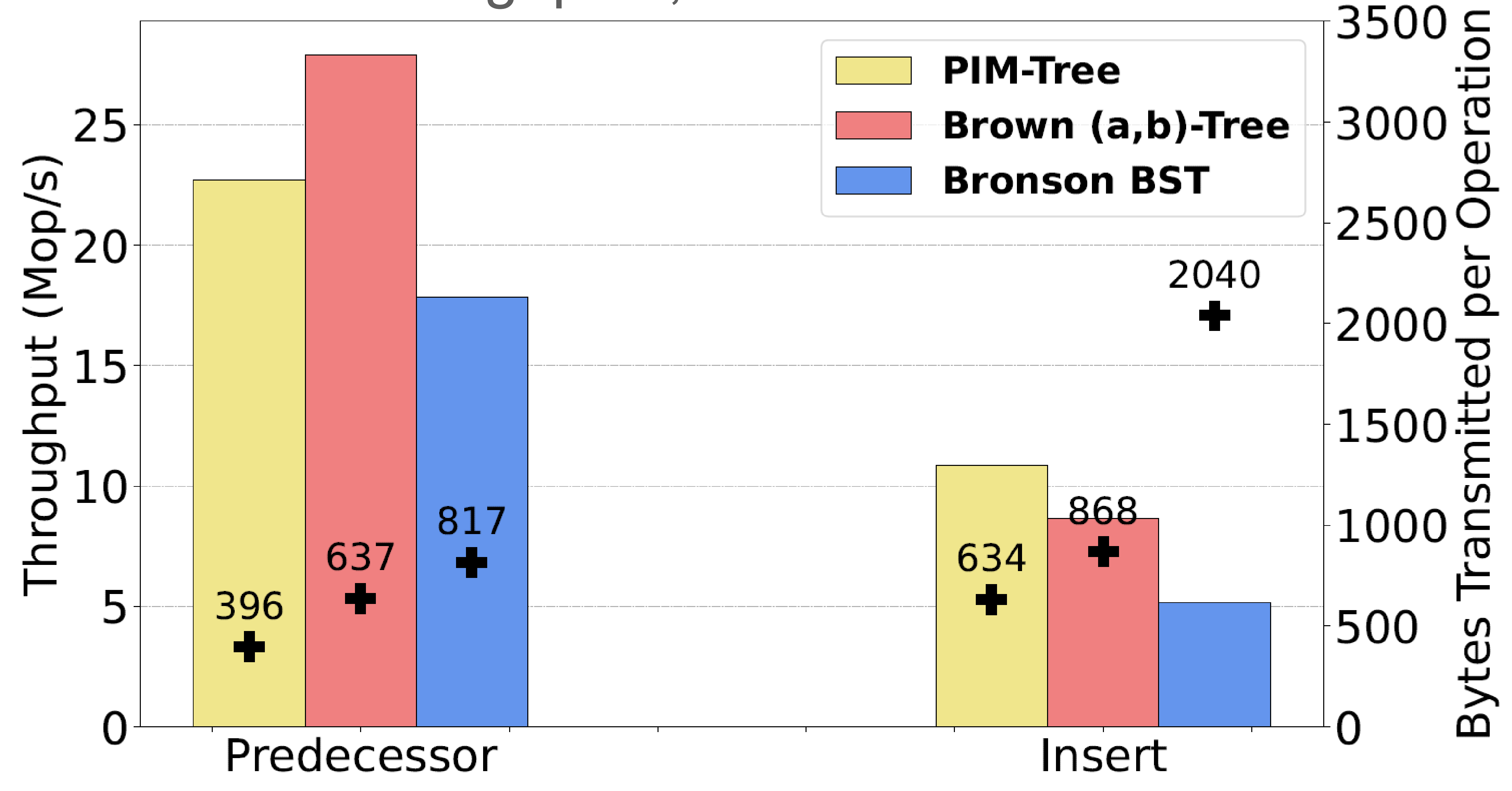
Results Against CPU-based Indexes

$\leq \mathbf{0.3}$X Communication !



Communication on Zipf workloads
vs. prior indexes

* Bars are throughputs; '+' are communications



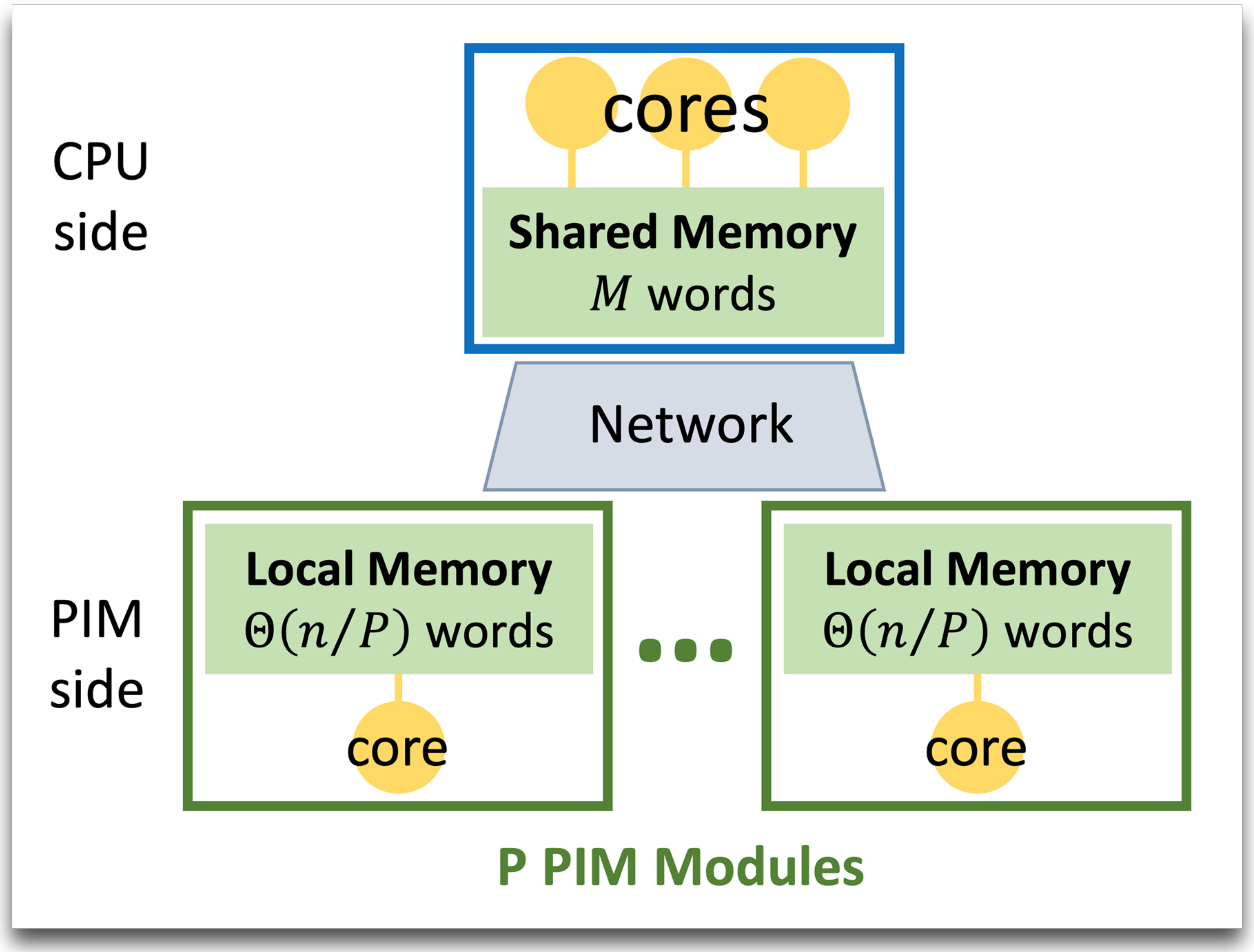
Experiments on Wikipedia dataset
vs. SOTA shared-memory Indexes

Results for First Generation PIM

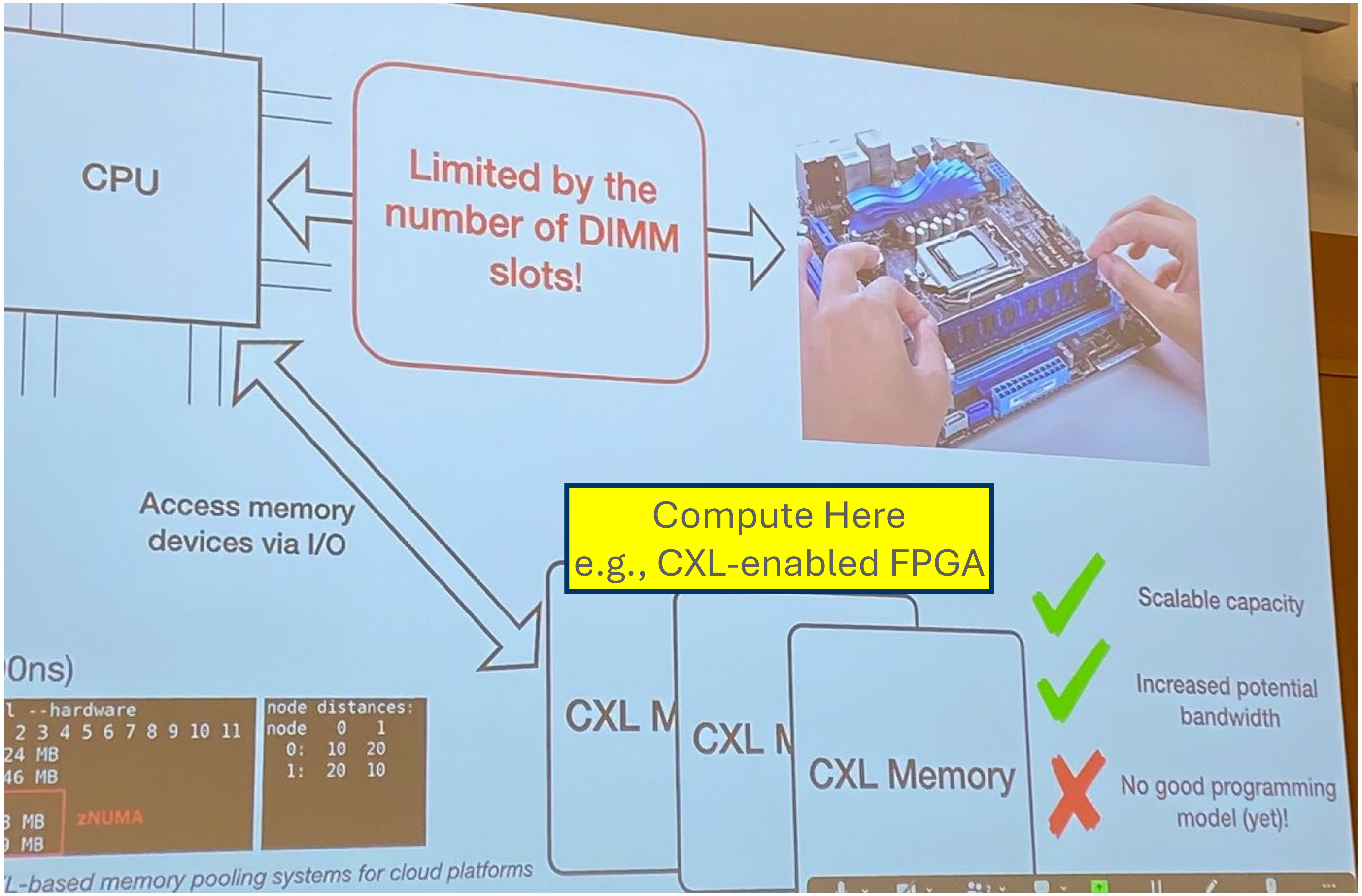
Recent: Left blue bars for PIM-tree removed by optimizing UPMEM communication library

Other Completed and Ongoing Results

- PIM-trie: PIM-optimized radix tree [Kang, Zhao, Blelloch, Dhulipala, Gu, McGuffey, Gibbons, SPAA'23]
- PIM optimized spatial index (zd-tree)
- Designing entire PIM-optimized DBMS (skew-resistant)
- Host CPU is beneficial due to its asymmetrically-high bandwidth



Future: PIM-equipped CXL


CPU
Limited by the number of DIMM slots!
Access memory devices via I/O
0ns)
zNUMA
L-based memory pooling systems for cloud platforms
Compute Here
e.g., CXL-enabled FPGA
CXL M
CXL N
CXL Memory
Scalable capacity
Increased potential bandwidth
No good programming model (yet)!

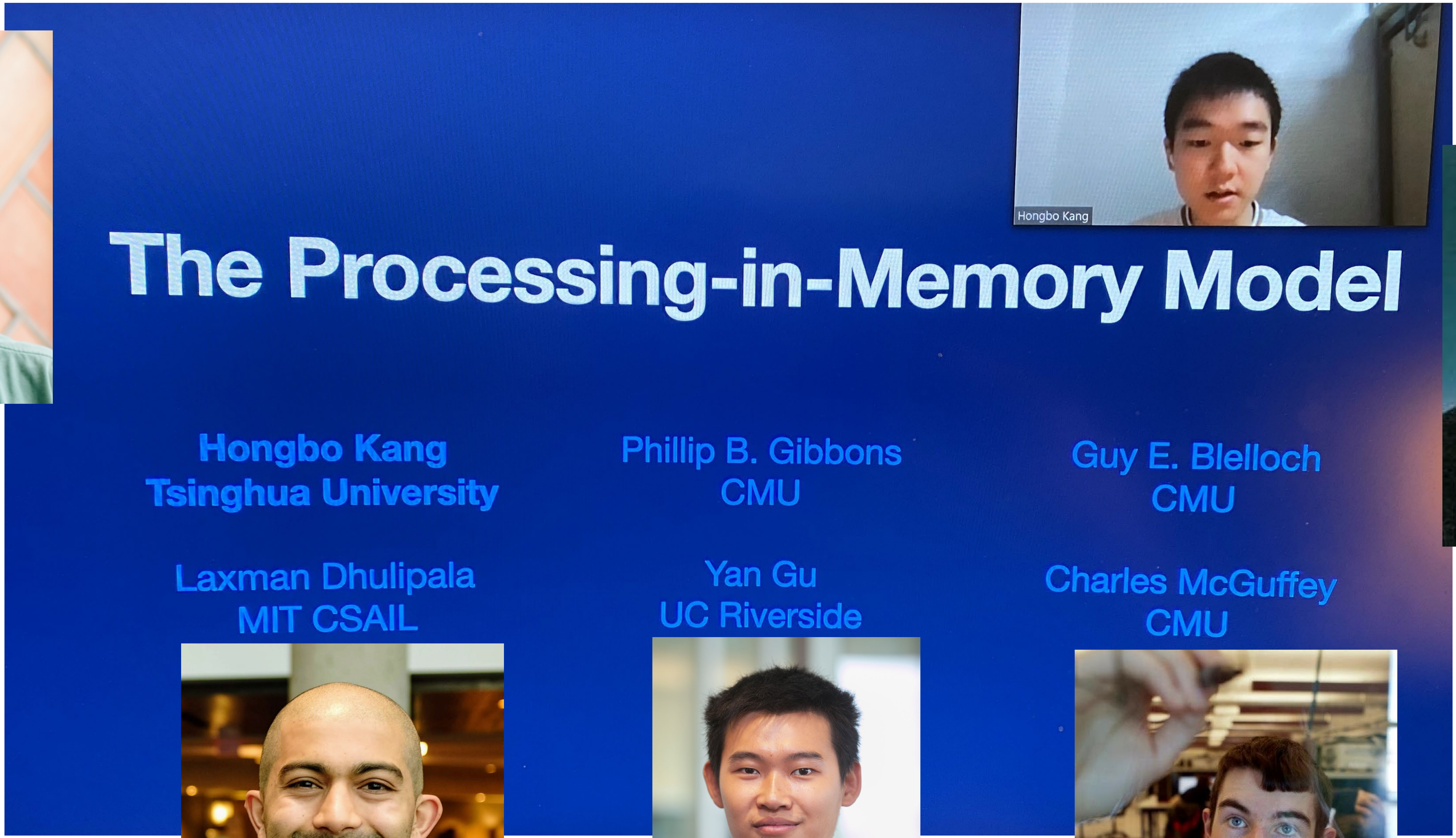
```
l --hardware
 2 3 4 5 6 7 8 9 10 11
24 MB
46 MB

3 MB
9 MB
```

```
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

Slide from Samuel Thomas' EMERALD presentation on CXL, with "Compute Here" added

Foundations of PIM: Project Team



Hongbo Kang

The Processing-in-Memory Model

Hongbo Kang
Tsinghua University

Phillip B. Gibbons
CMU

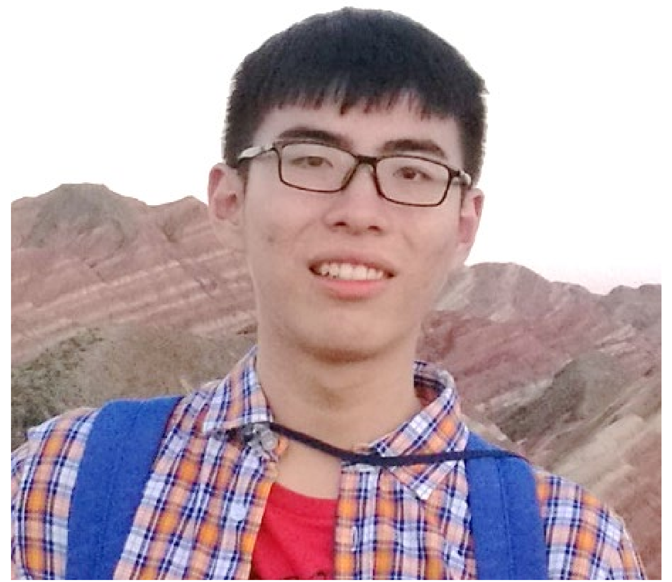
Guy E. Blelloch
CMU

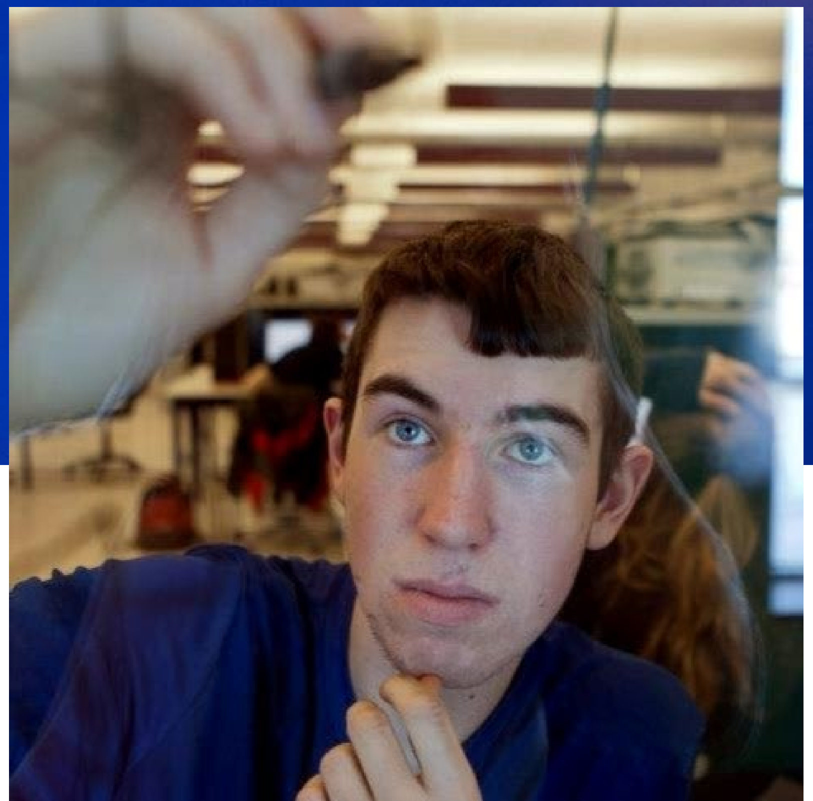
Laxman Dhulipala
MIT CSAIL

Yan Gu
UC Riverside

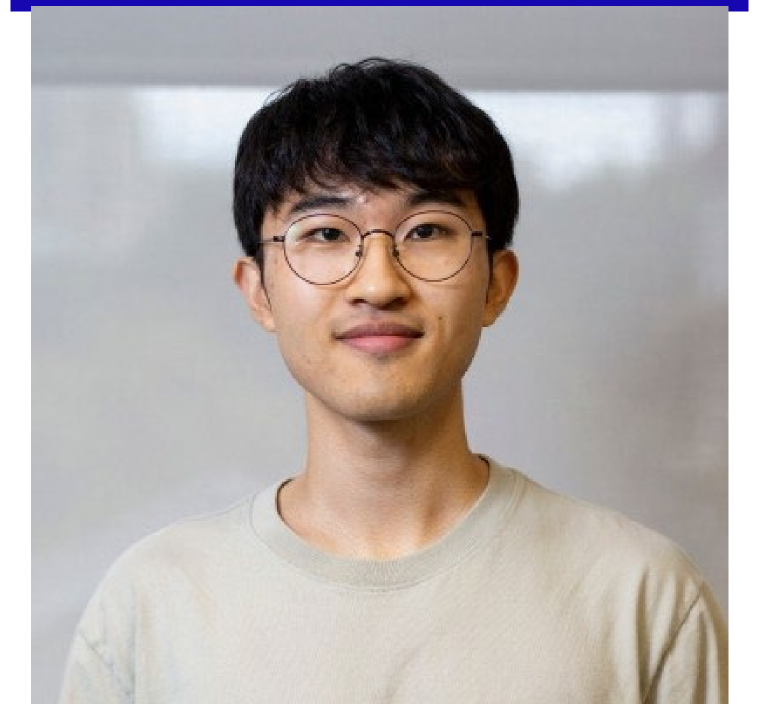
Charles McGuffey
CMU

Yiwei Zhao
CMU



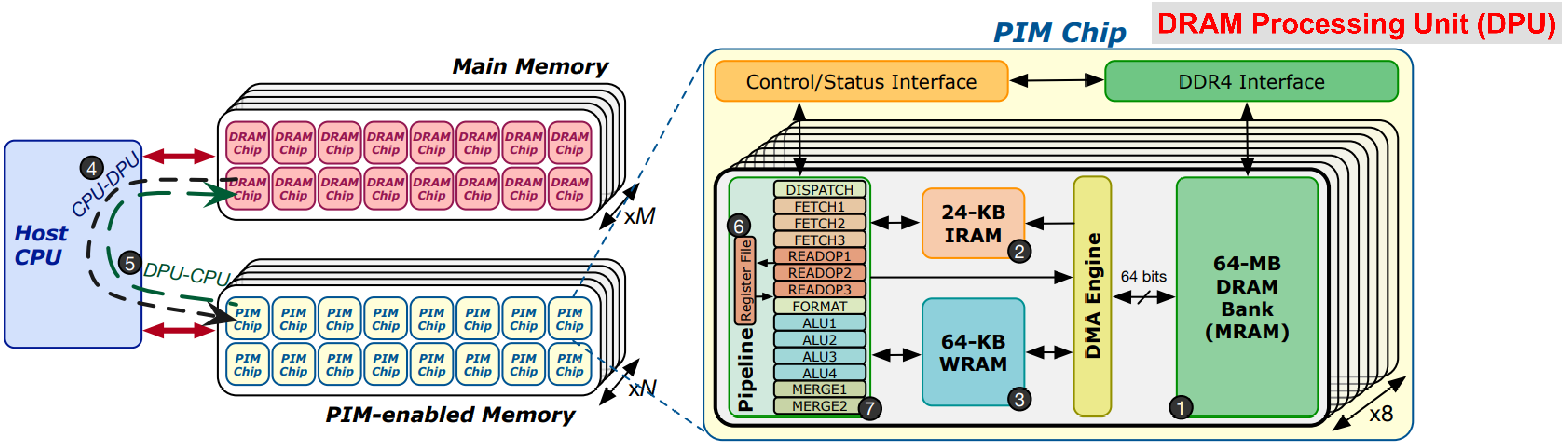


Hyoungjoo Kim
CMU



Backup Slides

UPMEM's PIM System



System	PIM-enabled Memory							DRAM Memory	
	DIMM Codename	Number of DIMMs	Ranks/ DIMM	DPUs/ DIMM	Total DPUs	DPU Frequency	Total Memory	Number of DIMMs	Total Memory
2,556-DPU System	P21	20	2	128	2,556[9]	350 MHz	159.75 GB	4	256 GB

System	CPU				
	CPU Processor	CPU Frequency	Sockets	Mem. Controllers/ Socket	Channels/ Mem. Controller
2,556-DPU System	Intel Xeon Silver 4215 [209]	2.50 GHz	2	2	3

Figure is from Gomez-Luna et al., "Benchmarking a New Paradigm…", arXiv, July 2021

Pretty good match for our PIM model