# Processing-in-Memory in the Wild

Sasha Fedorova

University of British Columbia

MongoDB

# Acknowledgments


## Students

- Andrada Zoltan
- Craig Mustard
- Jacob Grossbard
- Joel Nider
- John Ramsden
- Larry Liu
- Mohammad Dashti
- Niloo Zarif


## Collaborators (UPMEM)

- Alexandre Ghiti
- Jordi Chauzi
- Rémy Cimadomo
- Fabrice Devaux
- Romaric Jodin
- Julien Legriel
- Vincent Palatin

# PIM: Now commercially available

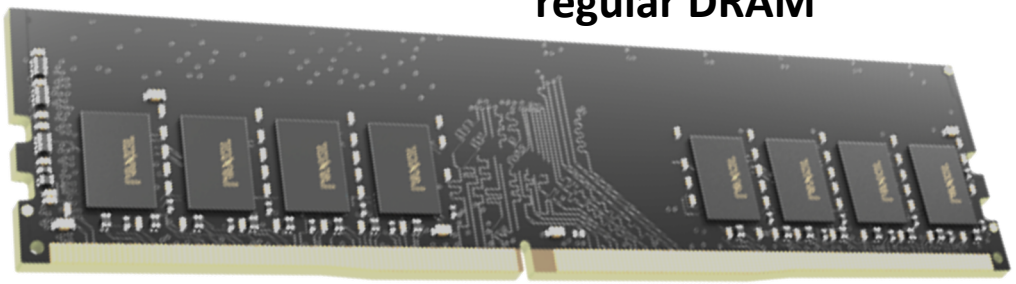
- UPMEM PIM-capable DRAM commercially available today
  - DDR4-compatible DRAM with small general-purpose processors
  - Integrated into standard off-the-shelf servers

- Samsung Function in Memory Architecture (FIMA) – coming up
  - Processors near HBM memory
  - Execute 32-instruction snippets, total of nine instructions available

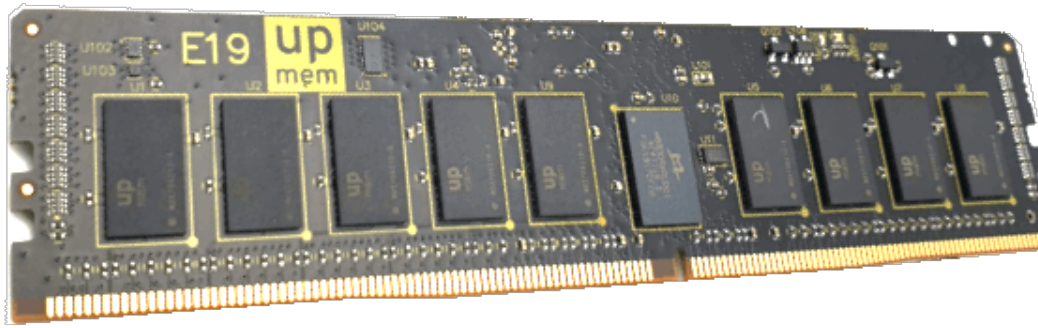
# UPMEM Overview

- Looks and can be used like regular DRAM
- Drop-in replacement for DDR4 DRAM
- Each chip is equipped with small processors
- Can be used in off-the-shelf servers

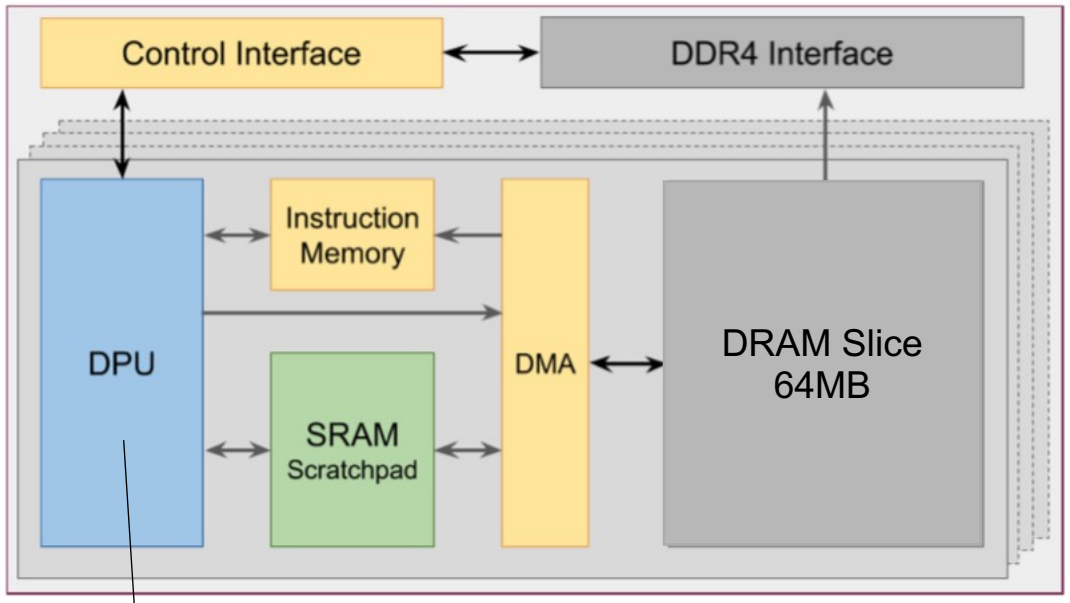
**regular DRAM**



**UPMEM DRAM**



# UPMEM architecture



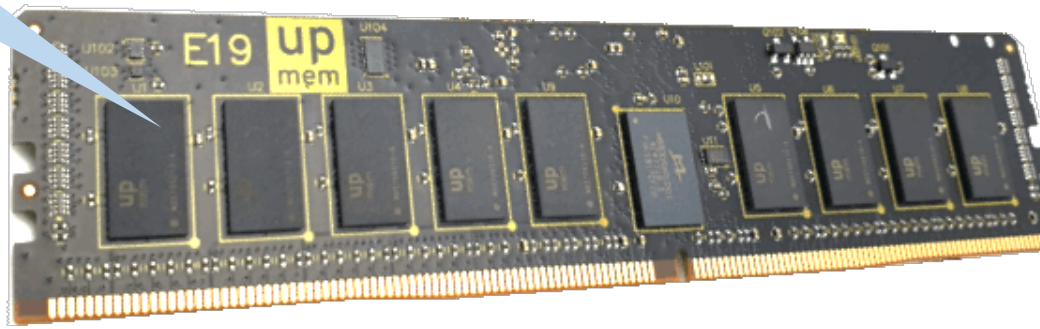

**DRAM Processing UNIT**

- General-purpose processor
- Simple, in-order, 267-500MHz
- No cache, only scratchpad memory

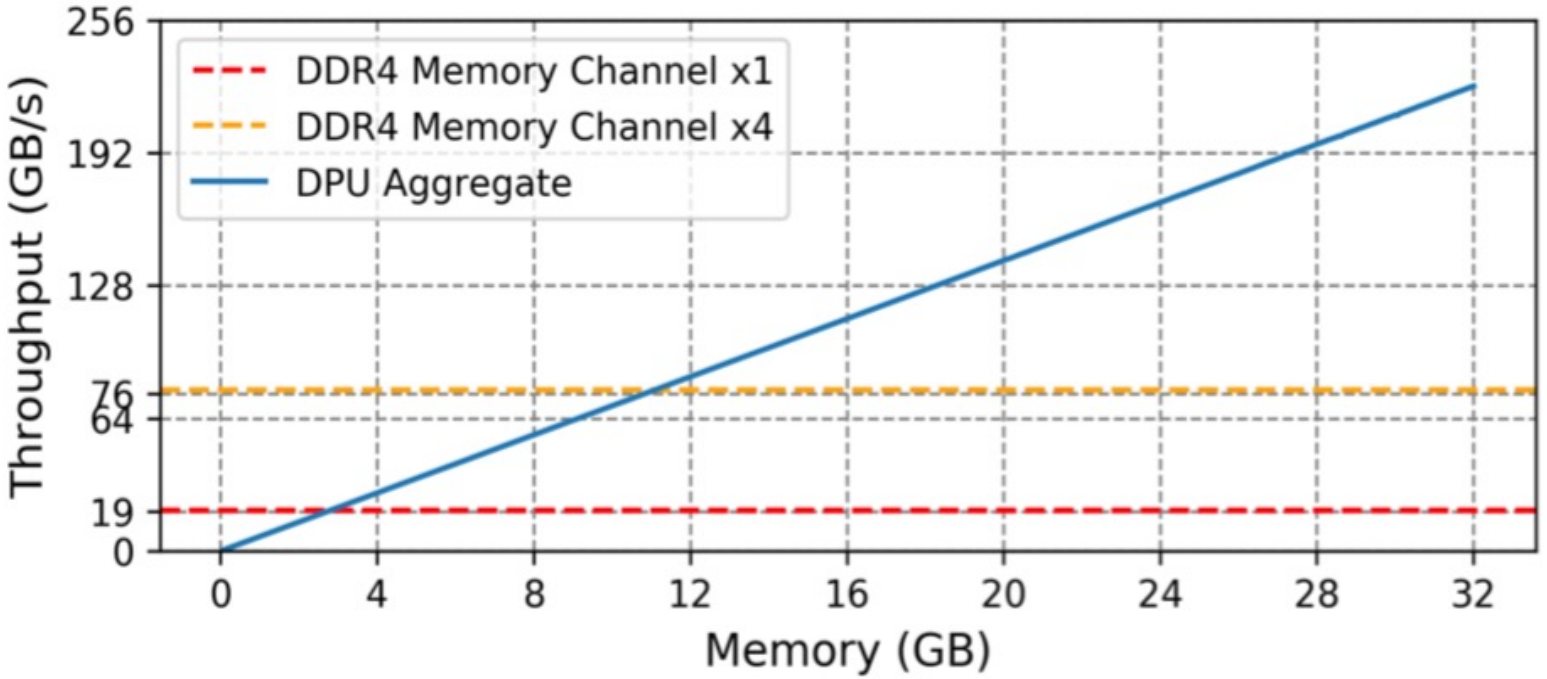
- One DPU per 64MB slice of DRAM
- 8GB DIMM: 128 DPUs
- Each DPU has its own DMA engine

***Many DPUs can achieve very high DRAM bandwidth together***

**UPMEM DRAM**



# PIM superpower: High DRAM bandwidth




- DPUs running at 267 MHz
- Only 32GB PIM DRAM
- Sequentially read every byte of memory

- 128GB DRAM, 500MHz DPUs: **2TB/sec bandwidth**

J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi, A. Fedorova
**A Case Study of Processing-in-Memory in off-the-Shelf Systems**, USENIX ATC 2021

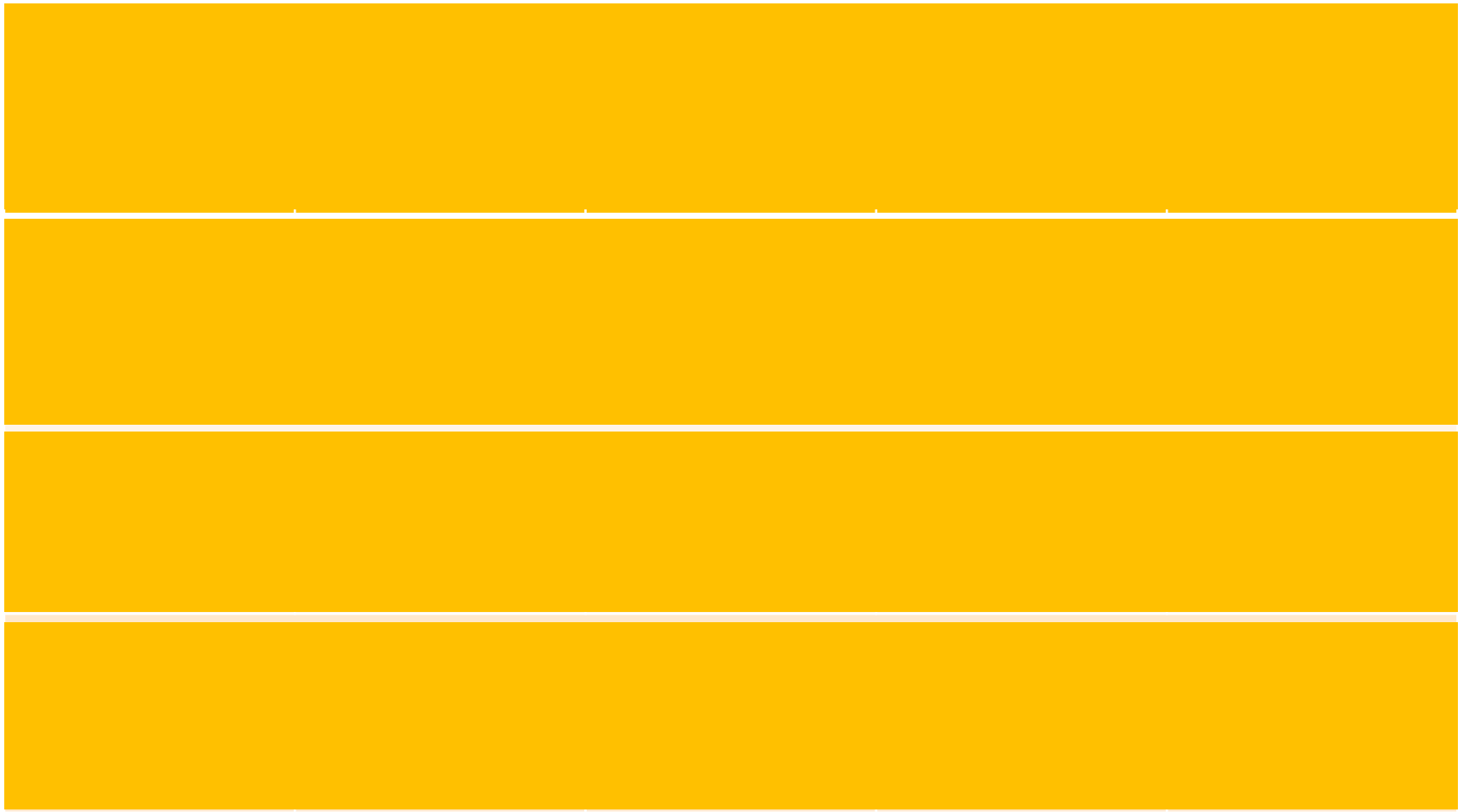
# Total cost of ownership

- You can also get high memory bandwidth
  - On high-end CPUs
  - HBM (high-bandwidth memory)
  - GPUs

*PIM can deliver high bandwidth at a lower cost ($$ and Watts)*

*…if you are able to use it efficiently*

# PIM superpower: Low TCO



**Price:**

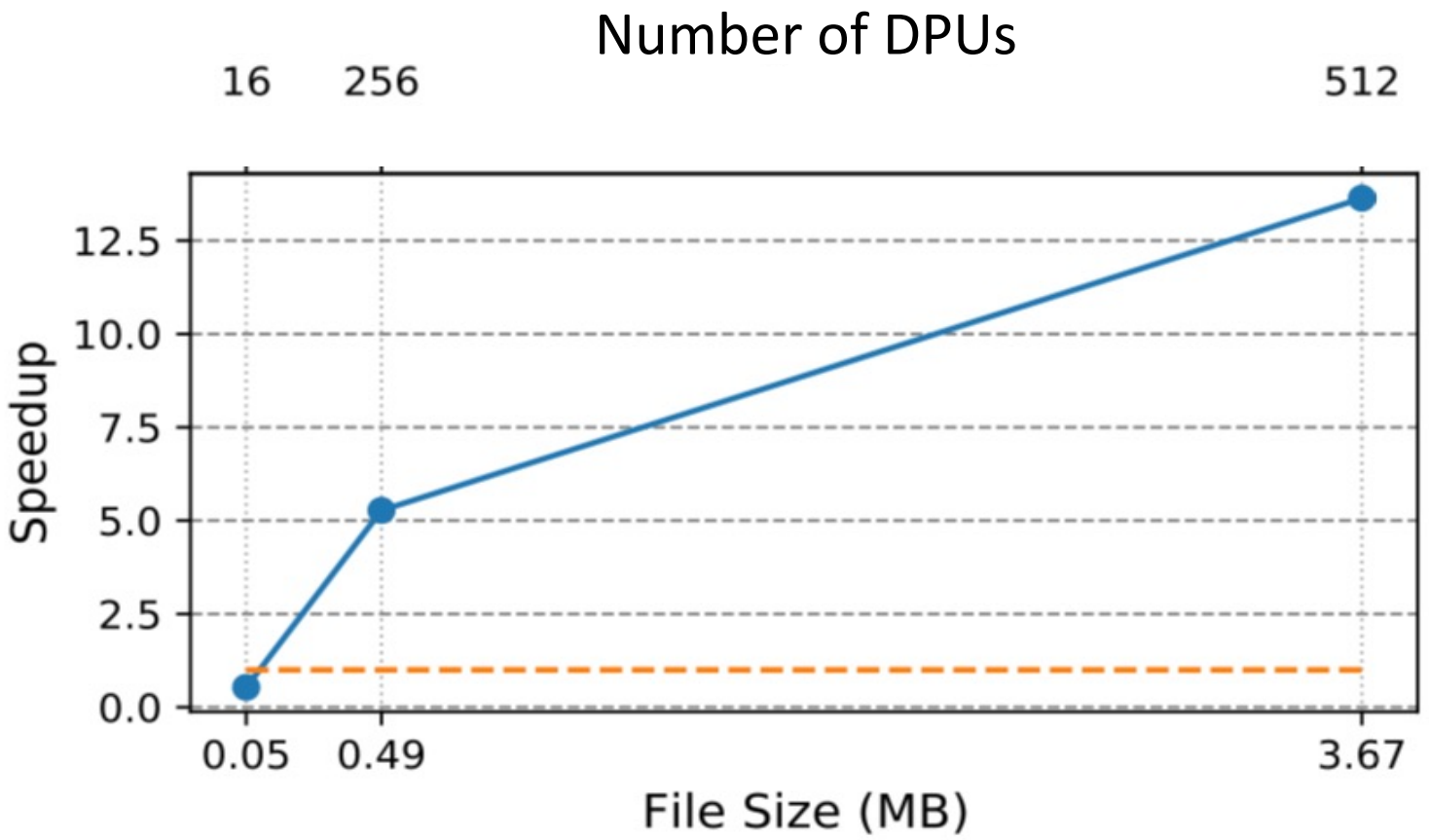
- 8GB UPMEM DIMM: ~$300
- More CPUs adds memory channels, but requires expensive high-end CPUs

**Power:**

- Typical server with DRAM: about 400W
- A server with 128GB PIM DRAM: 700W

*Prices include just CPU and memory

# PIM performance: hyper-dimensional computing

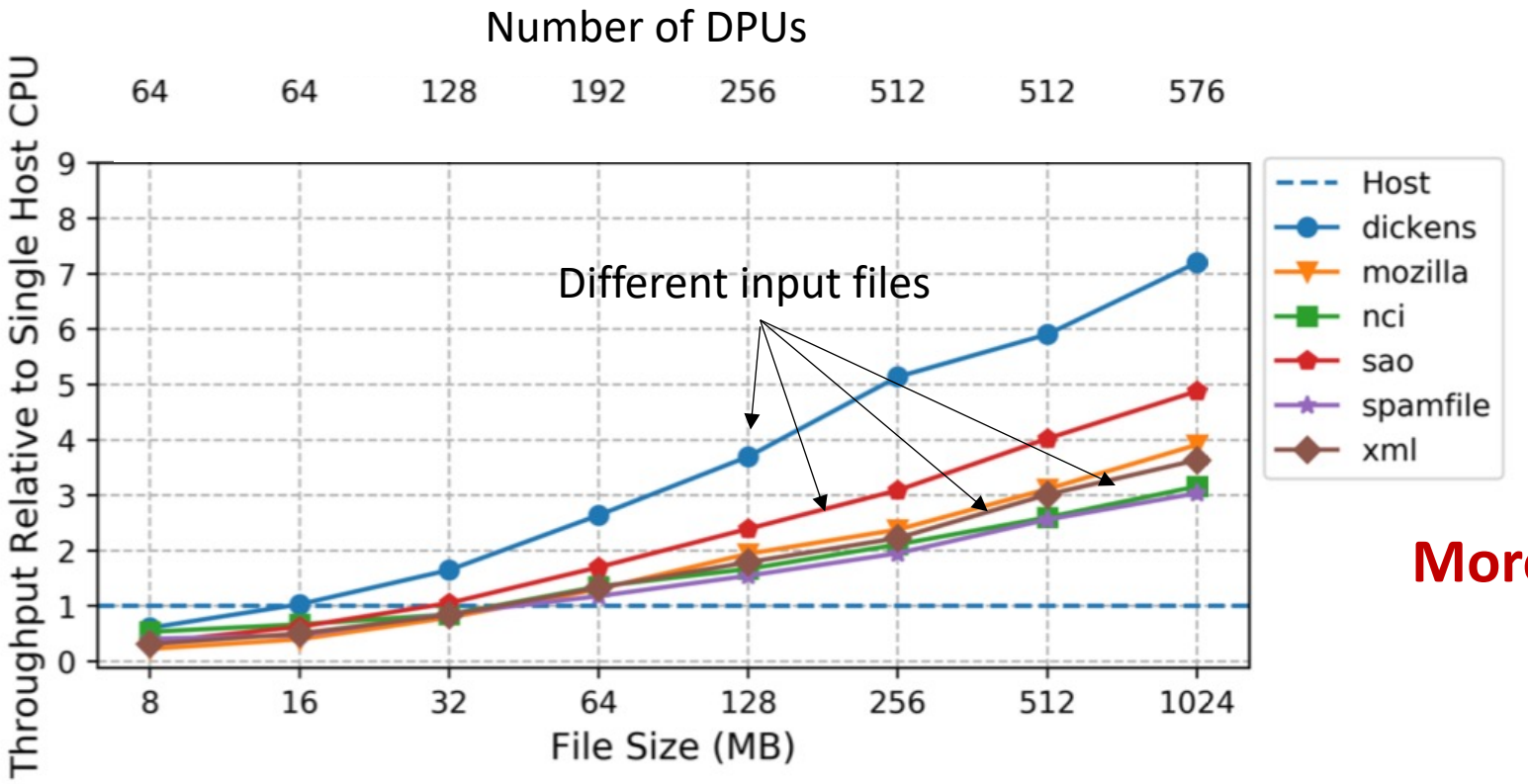



**More data → more DPUs → more throughput**

- Speedup over a single host CPU
- End-to-end measurements

J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi, A. Fedorova
**A Case Study of Processing-in-Memory in off-the-Shelf Systems**, to appear in USENIX ATC 2021

# PIM performance: compression



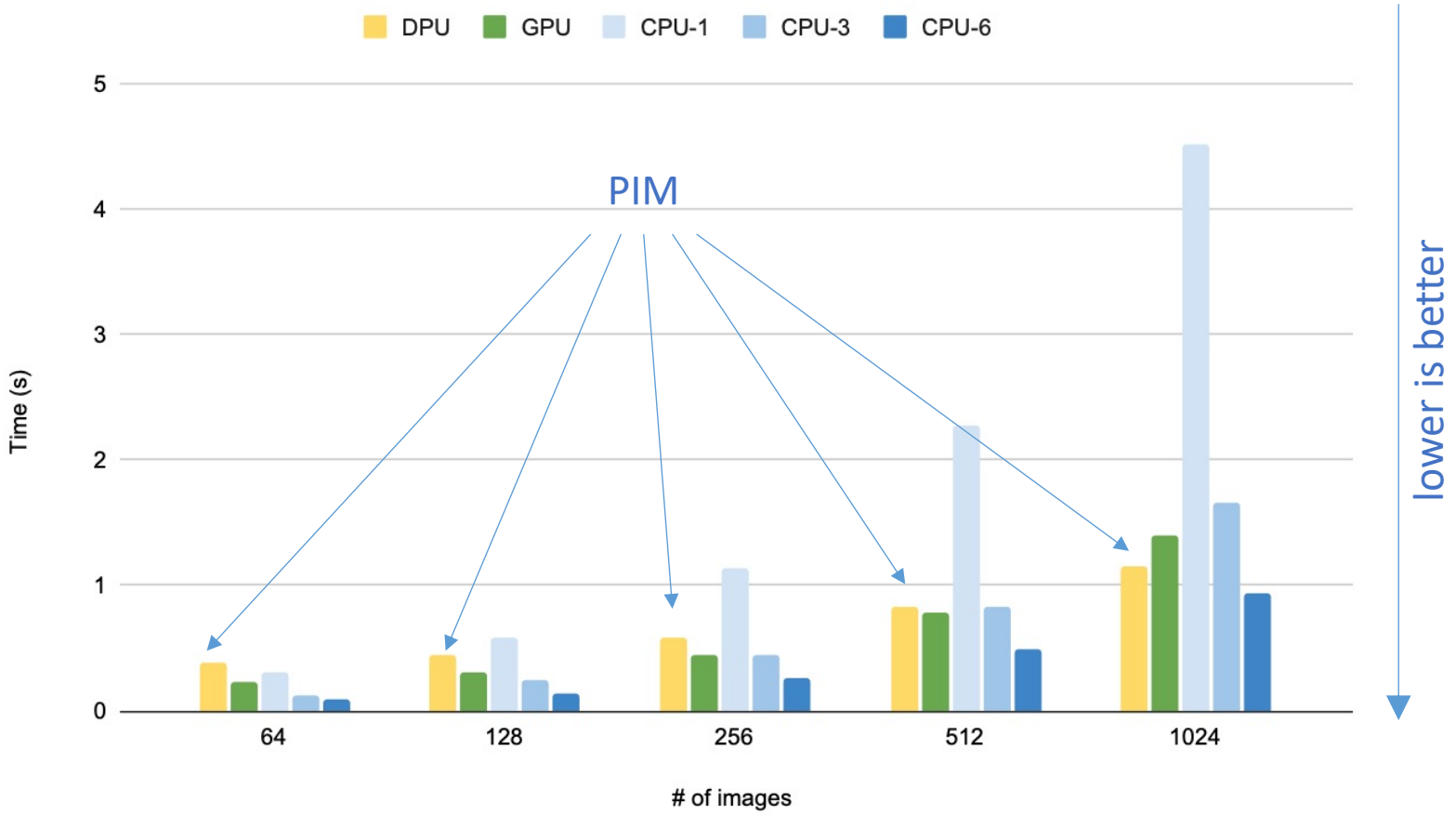

- Snappy compression
- Speedup over a single CPU
- **End-to-end measurements**

**More data → more DPUs → more throughput**

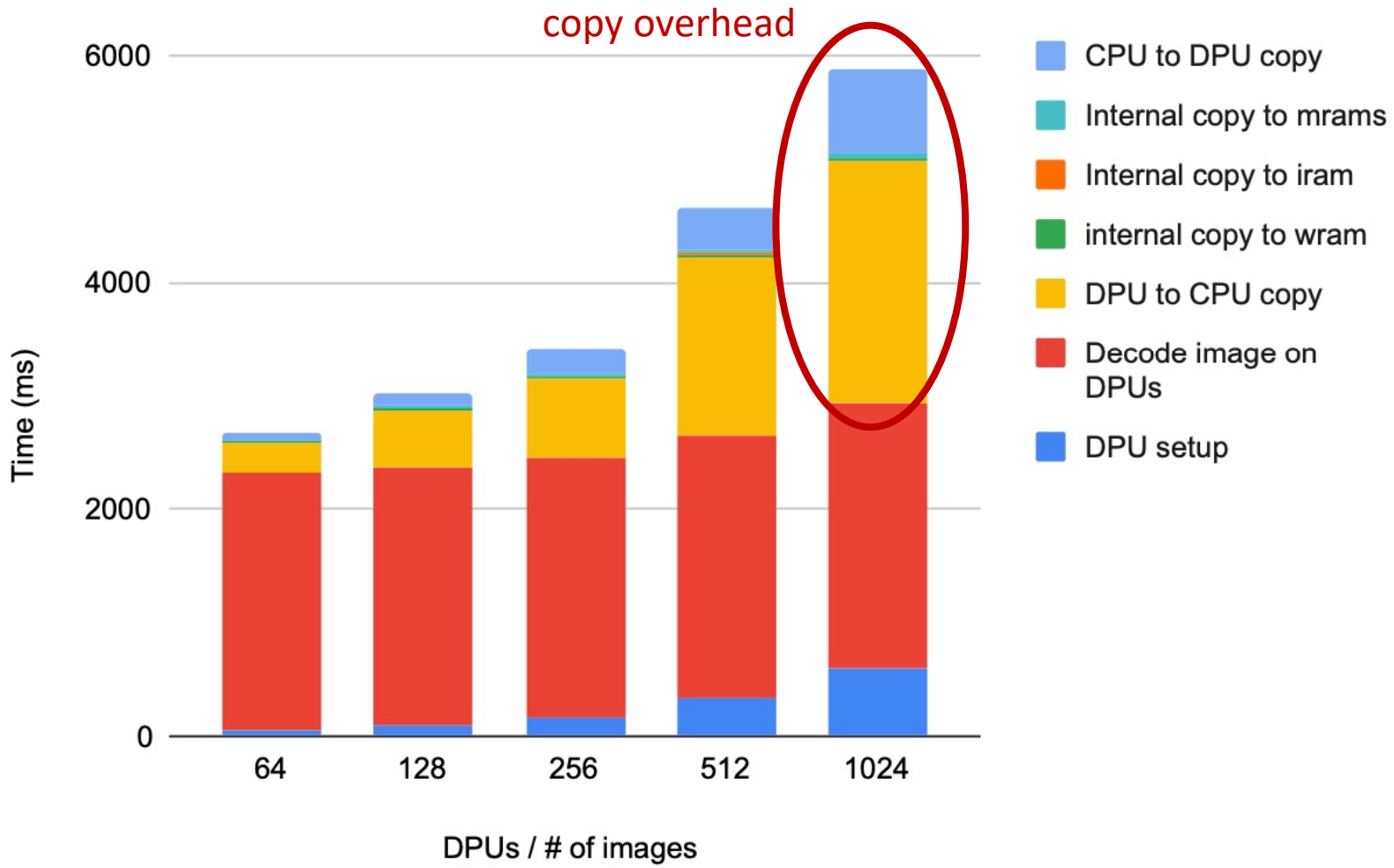
J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi, A. Fedorova
**A Case Study of Processing-in-Memory in off-the-Shelf Systems**, to appear in USENIX ATC 2021

# Image Processing on PIM

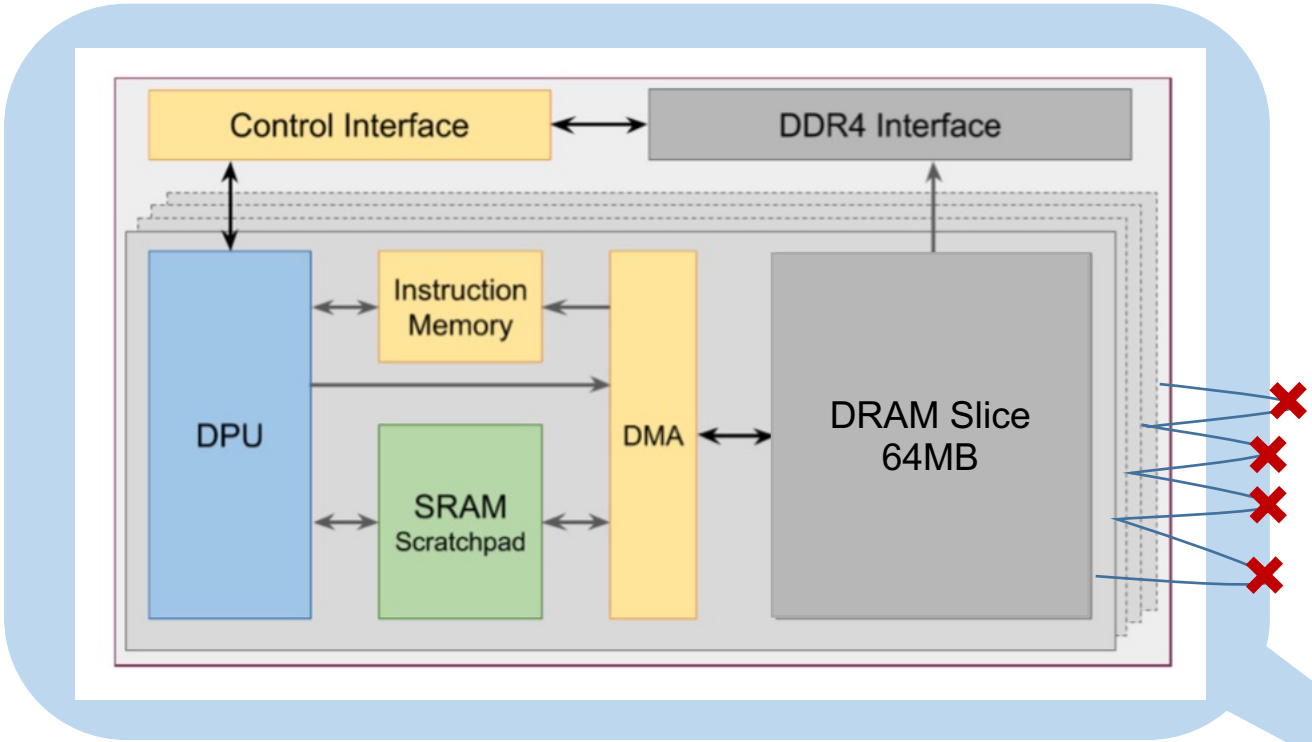
Motivation: pre-process images for ML training



# Moving data back-and-forth



# PIM limitations



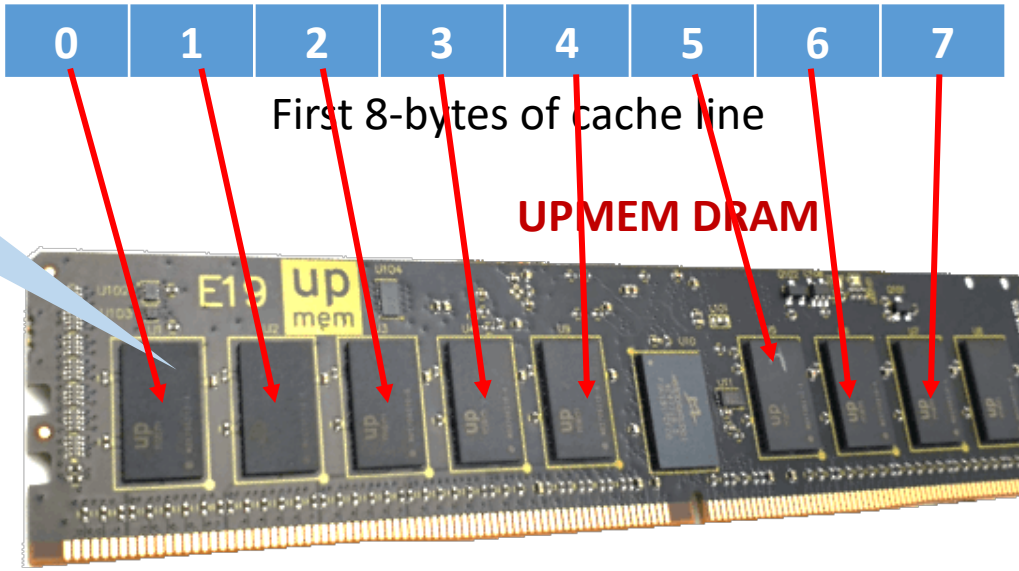

**DPUs store data in different format than the CPU expects:**

- Has to do with interleaving
- CPU has to de-interleave the data used by DPU

**DPUs can't talk to each other**

- Each DPU sees data only in its slice

**CPU splits a cache line across slices for better performance**



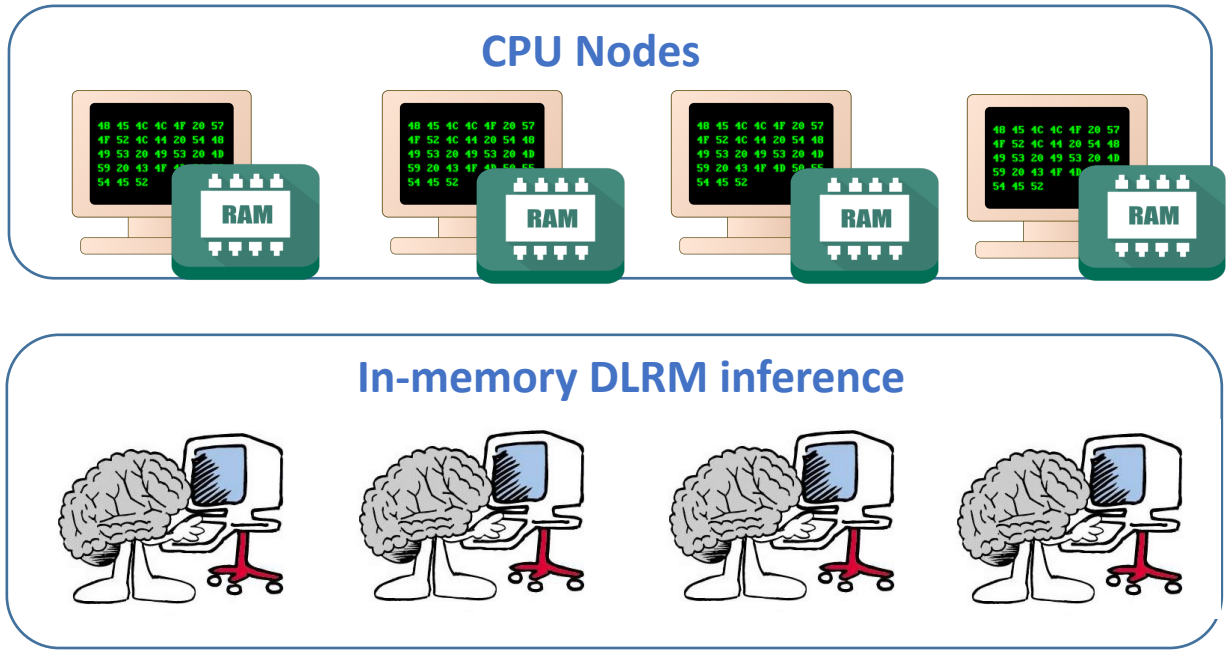
# Deep Learning Recommendation Models in PIM

## Problem

- DNN-trained models
- Inference done on CPUs
- Models can be very large (even terabytes[1])
- Inference is bandwidth intensive

## Can this be done?

- Parts of DLRM inference were done in-memory[2] and in-storage[1]

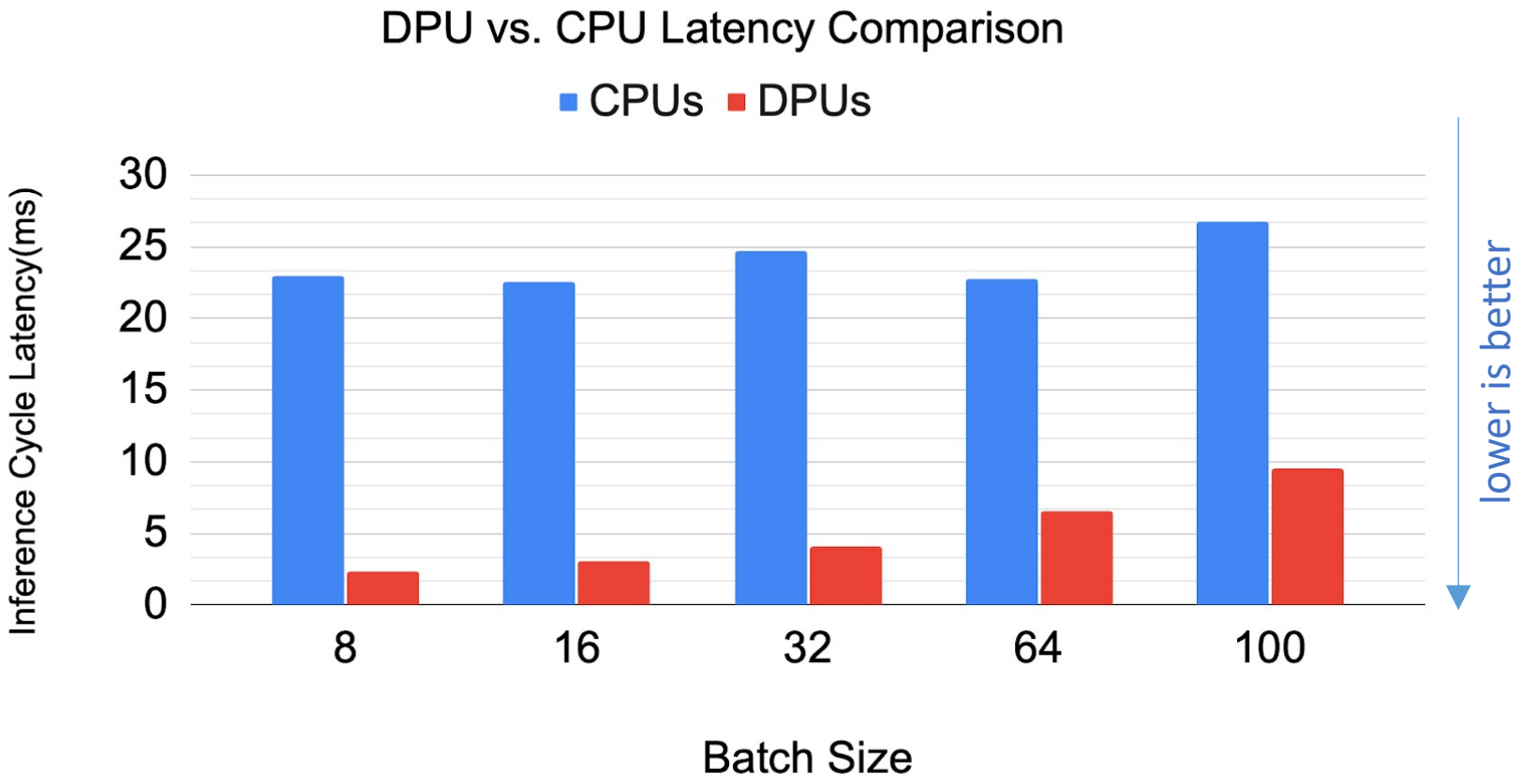



## Challenges

- Some DLRM inference phases are compute-intensive
- Skewed workload: balanced parallelism may be difficult to achieve

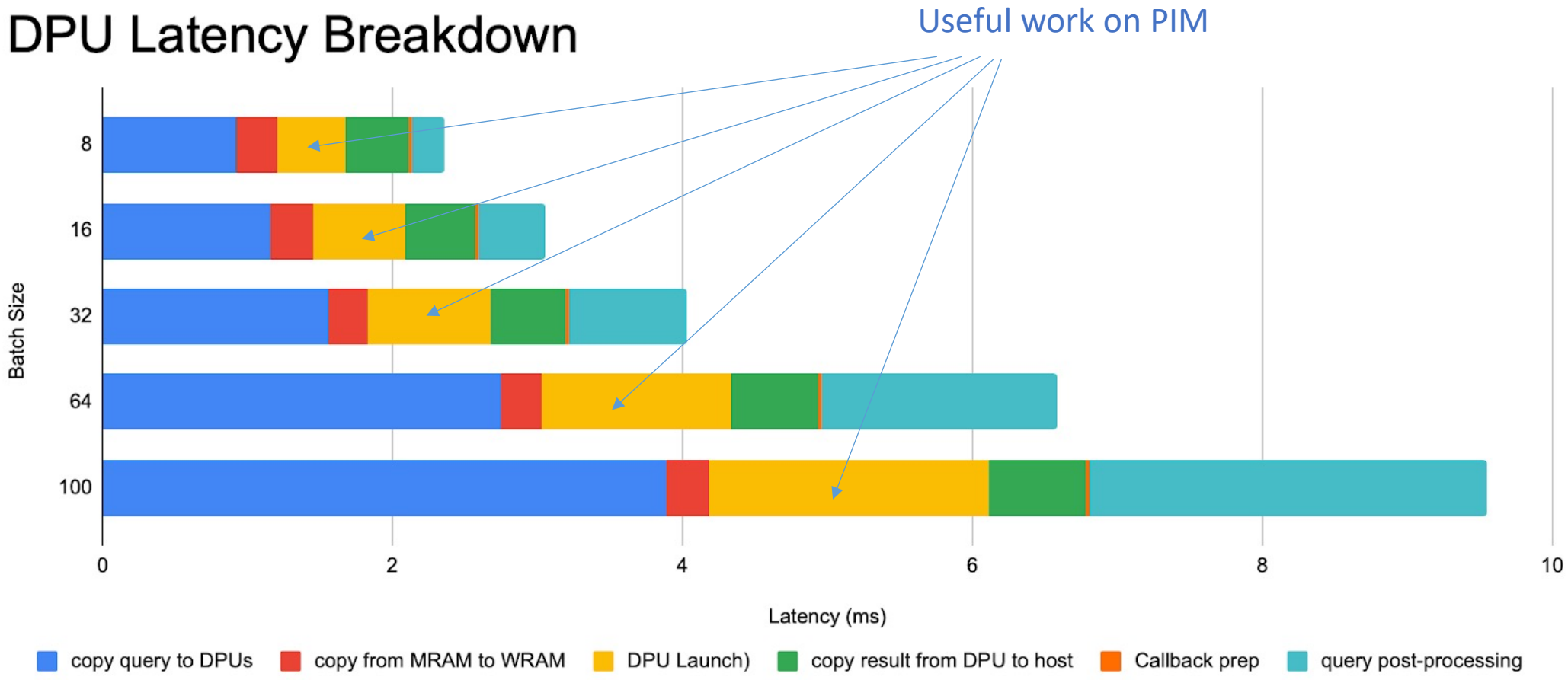
[1] Wilkening et al. RecSSD…, ASPLOS 2021
[2] Ke et al. RecNMP…, ISCA 2020

# DLRM Inference in PIM

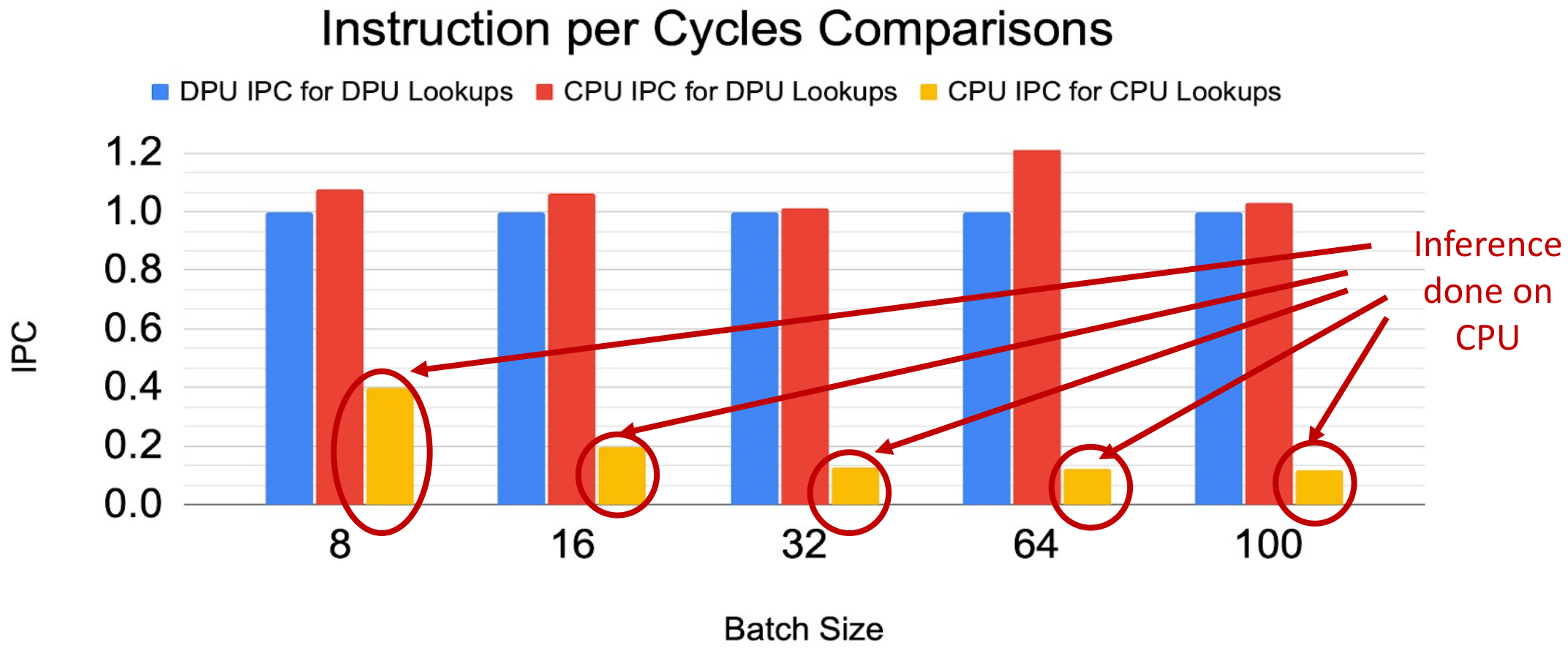



- RMC2 model
- Latency of a single reference cycle. Larger batch size – more work.

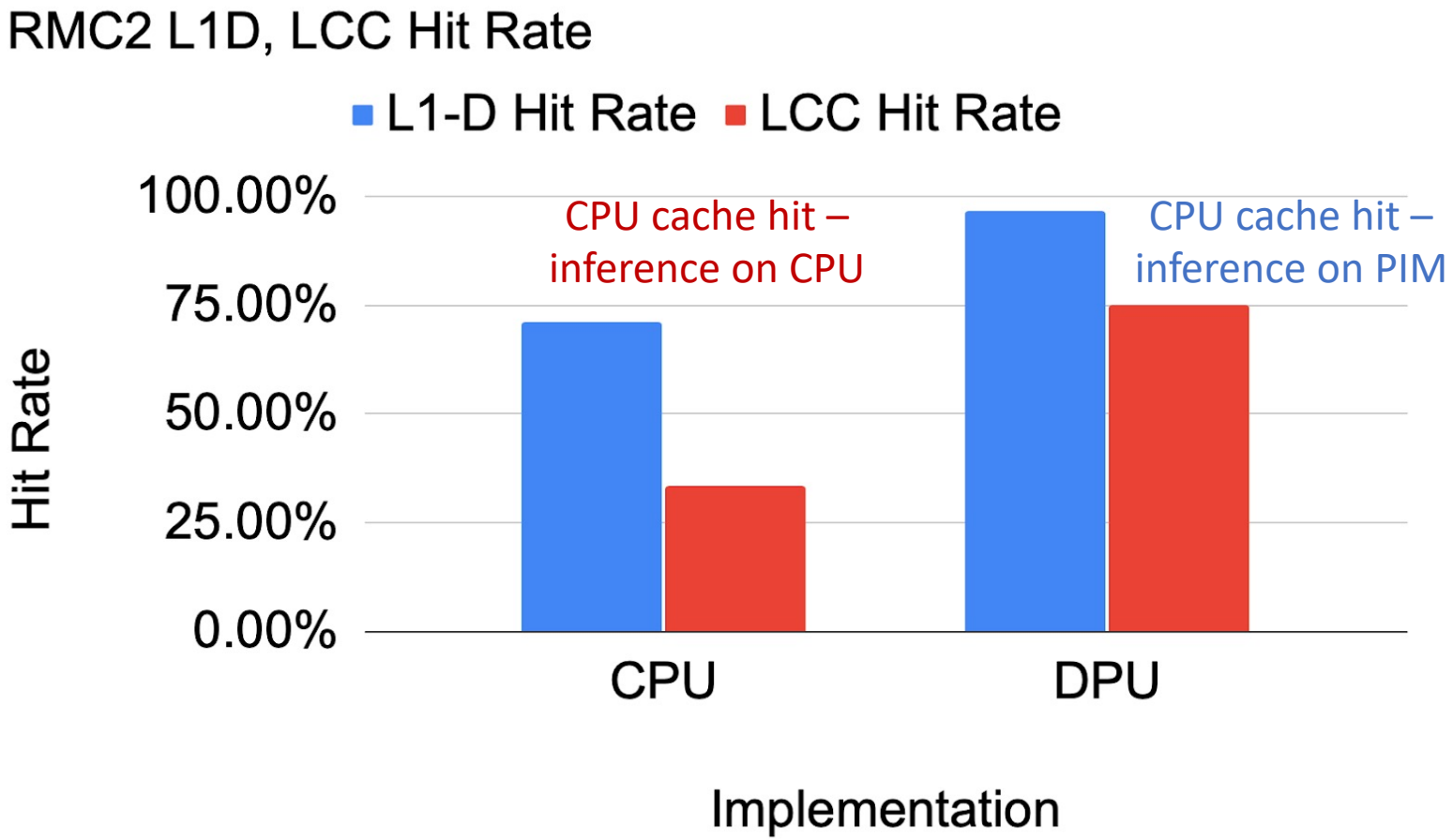
# Where does the time go?



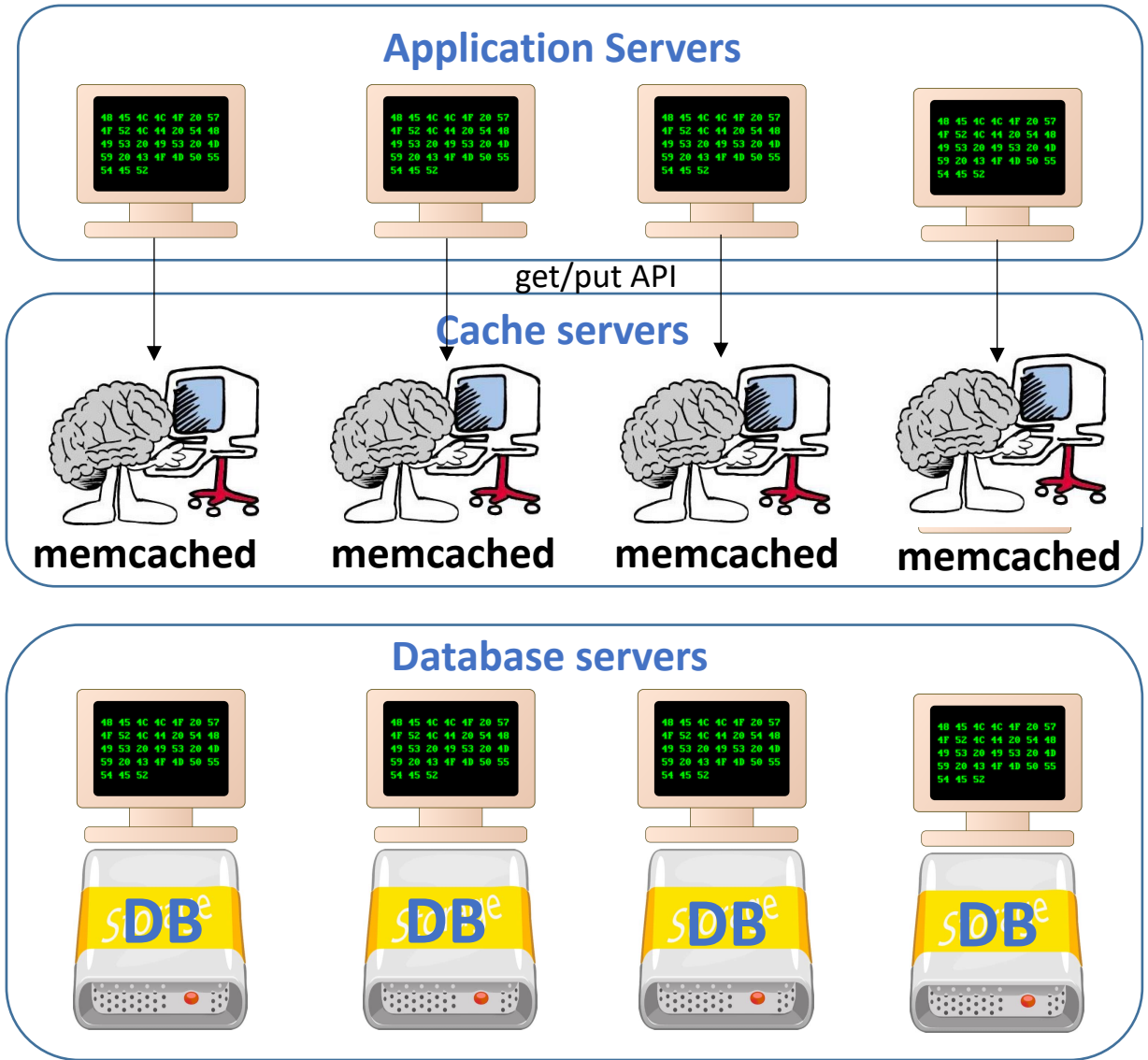
# Low IPC when inference done on CPU



# CPU implementation hits the memory wall



# Key-value store (in-memory cache) in PIM




**Benefits:**

- Cache servers use general-purpose CPUs
- Get rid of expensive CPUs
- Save money and power

**Can this be done?**

- Get/put interface to memory
- Hashtable and a memory allocator inside PIM
- Data already sharded across cache servers – no communication needed!

**Challenges**

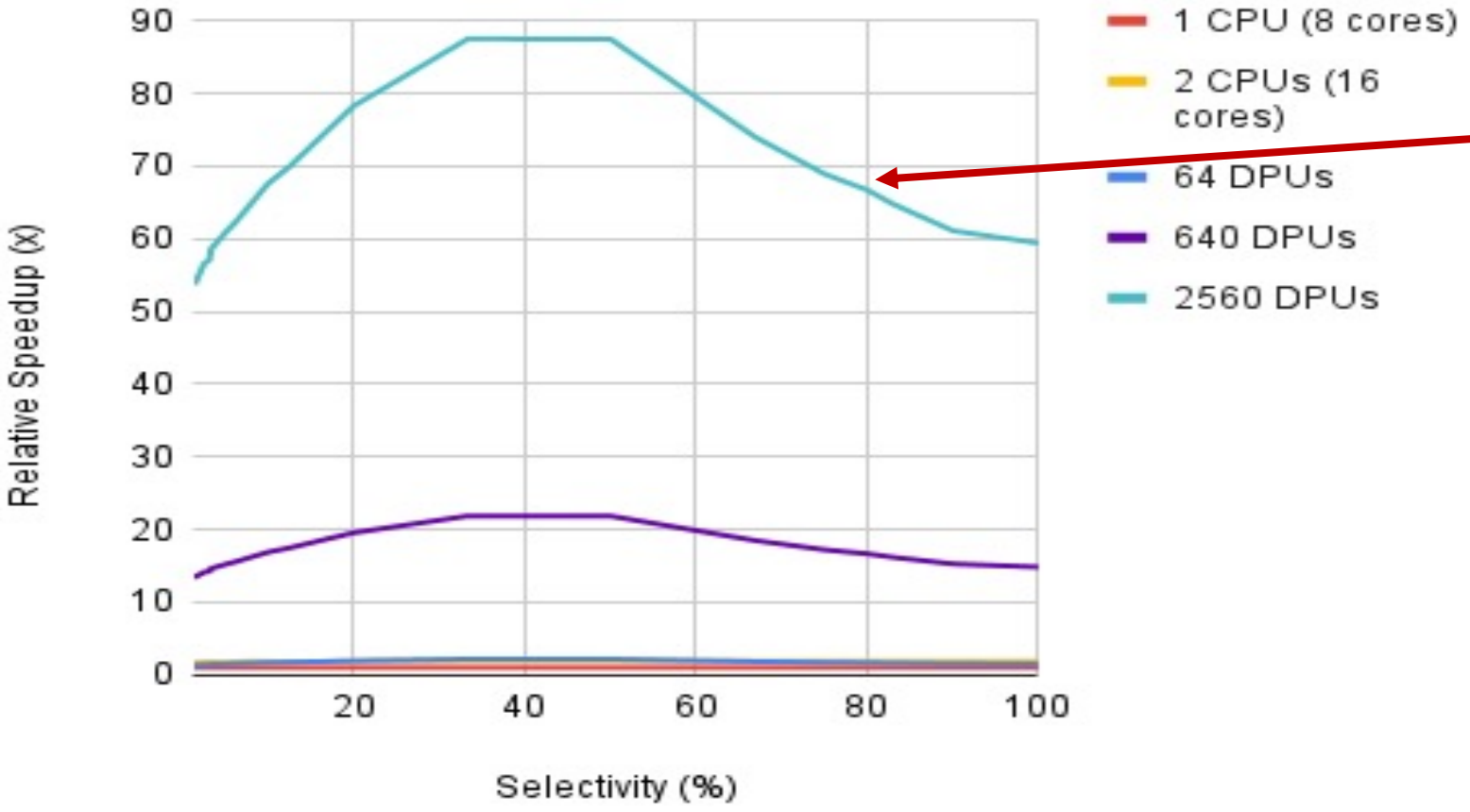
- Difficult to buffer get requests; executing one at a time may be inefficient

[1] LightStore (ASPLOS'19) – a key value store in SSD

# Filtering-in-memory

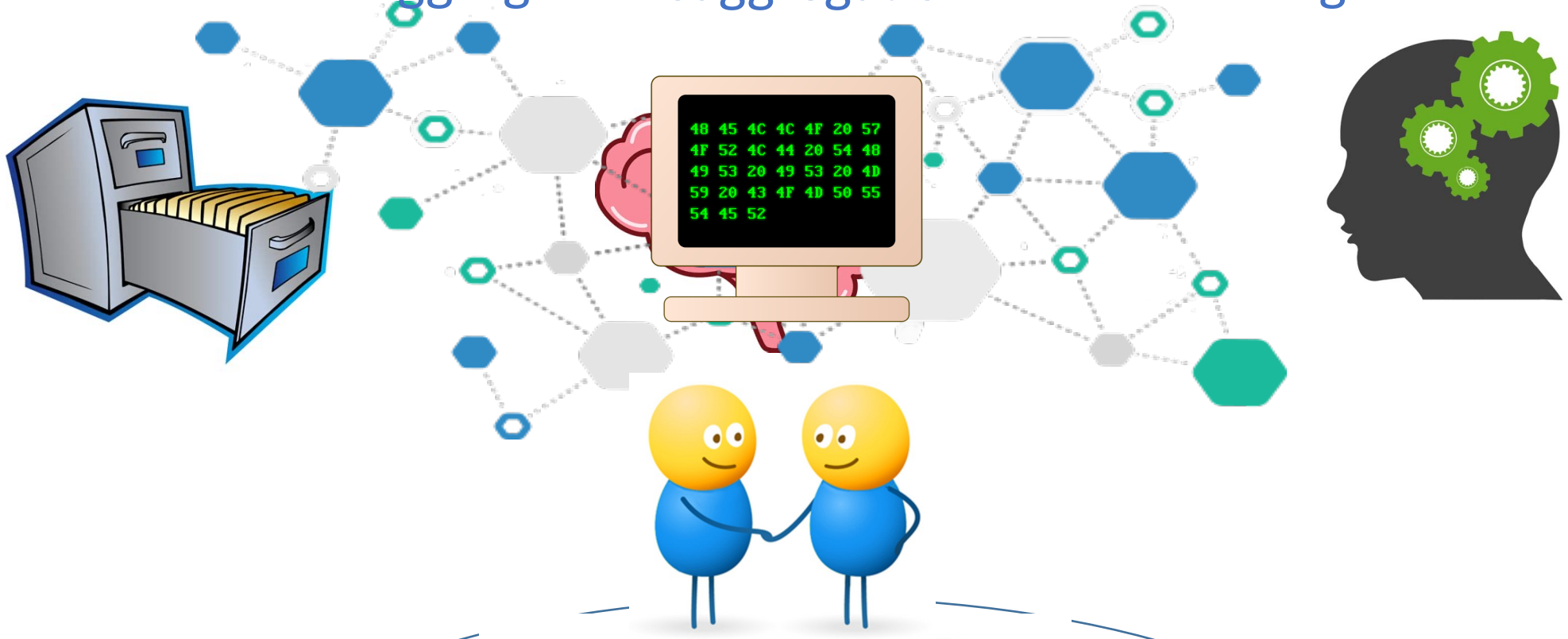
- Key-value store with a column-store architecture
- 30MB of data per DPU
- 20 tasklets

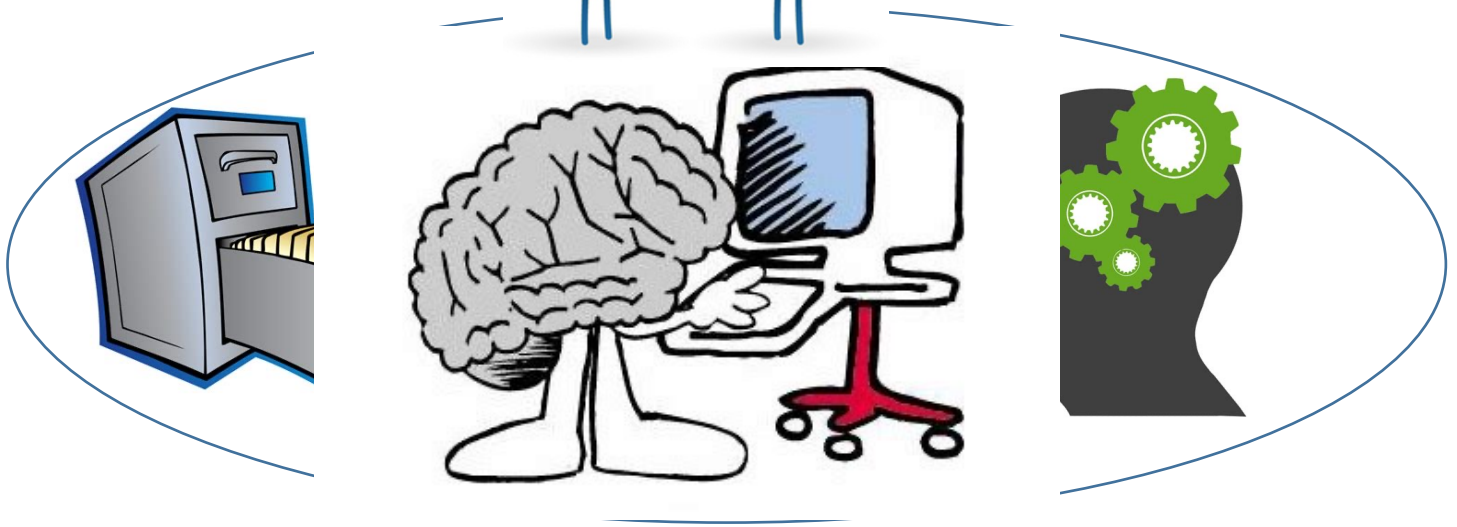
# Filtering on PIM vs on CPU



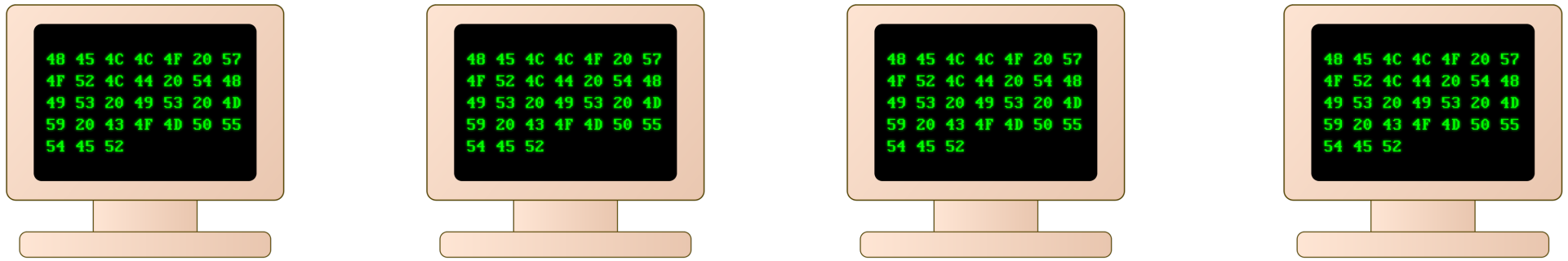

As selectivity increases, we need to copy more data to back to the CPU

# Disaggregation Meets Data Processing

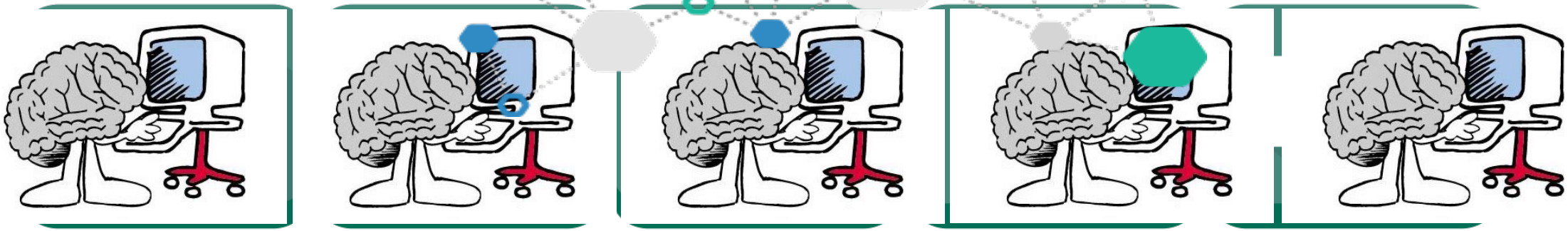


# Disaggregated memory meets PIM



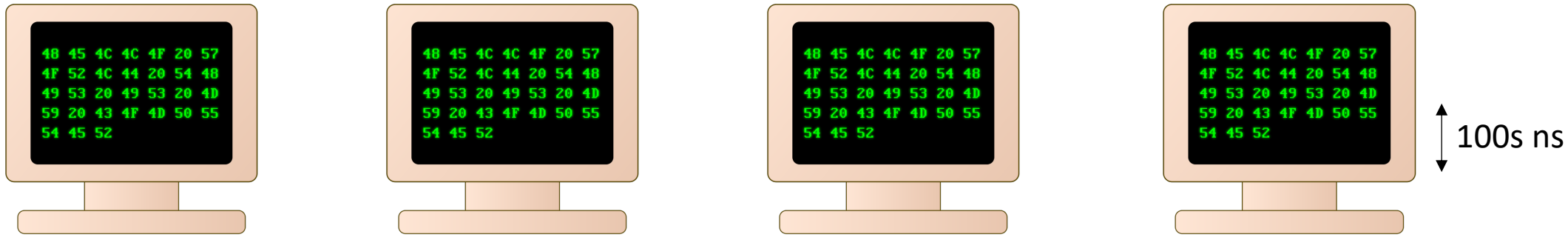

**PIM-enabled memory blades**



# Memory management

- Each DPU can run its own memory allocator
- Keep a local page table
- Can even compress pages on the fly

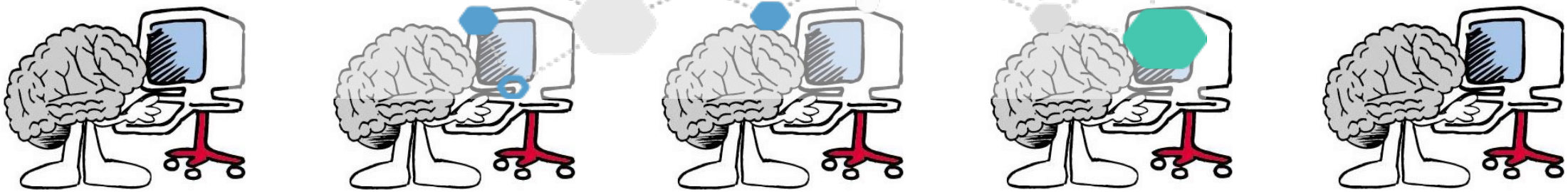
# Disaggregated memory meets PIM




**CPUs use load/store interface to access memory**

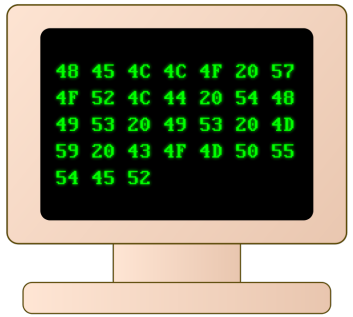
**~10s of µs**

**Let memory run applications!**



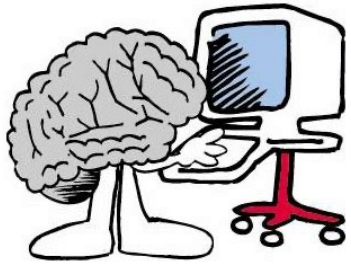
# Disaggregated Memory Runs Applications

- Disaggregated memory consists of CPU-less PIM blades
- PIM array runs an entire application
- CPU blades access memory via a higher-level API




load/store

Higher-level interface



Similar ideas:

- AIFM (OSDI 2020): Application-integrated far memory
- KV-Direct (SOSP 2017): key-value API to memory via a smart NIC and RDMA
- StRoM (EuroSys 2020): Smart Remote Memory