上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

UM-PIM: DRAM-based PIM with Uniform & Shared Memory Space

Yilong Zhao, Mingyu Gao, Fangxin Liu*, Yiwei Hu, Zongwu Wang, Han Lin, Ji Li, He Xian, Hanlin Dong, Tao Yang, Naifeng Jing, Xiaoyao Liang, Li Jiang*

Shanghai Jiao Tong University

2024/7/2

饮 水 思 源 • 爱 国 荣 校

Outline

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Outline

Background: the "Memory Wall"

- According to Computation / Memory
  - Compute bound: Matrix Multiplication
  - Memory bound: Element-wise (dropout， masking) Reduce (Layer nom, Sum)

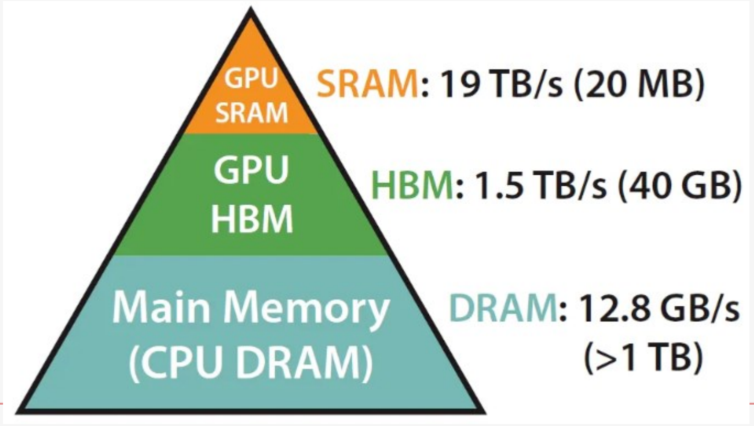
- Memory bottleneck becomes severe with the emergence of large models

Memory increases with Token linearly

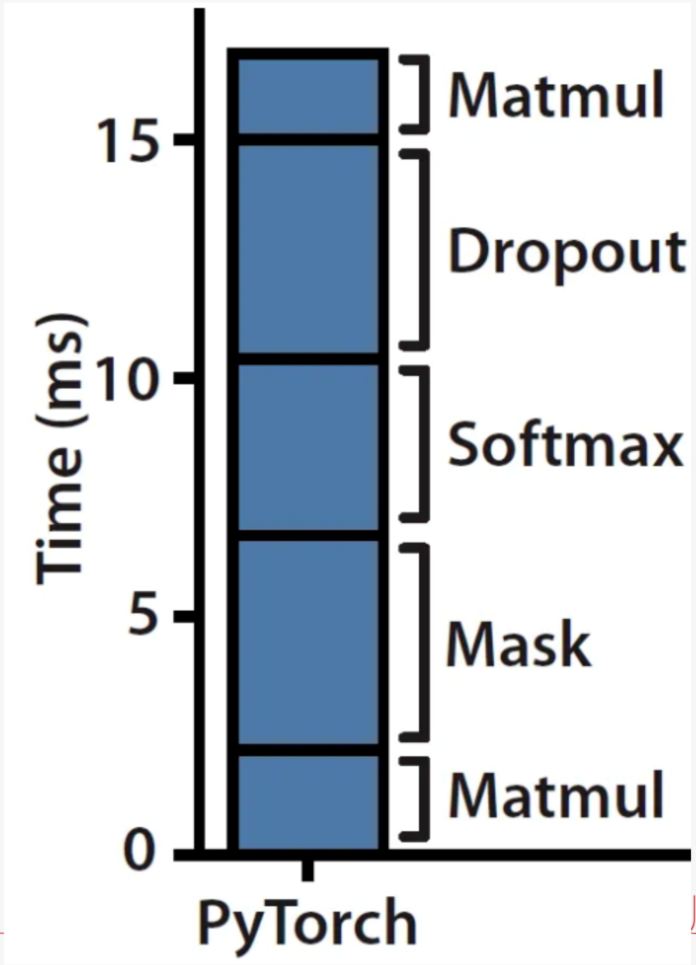
Llama2-13B Model

Token len	4K	16K	128K
FP Memory	3G	12G	100G
BP Memory	10G	39G	320G

Accelerate with DRAM?

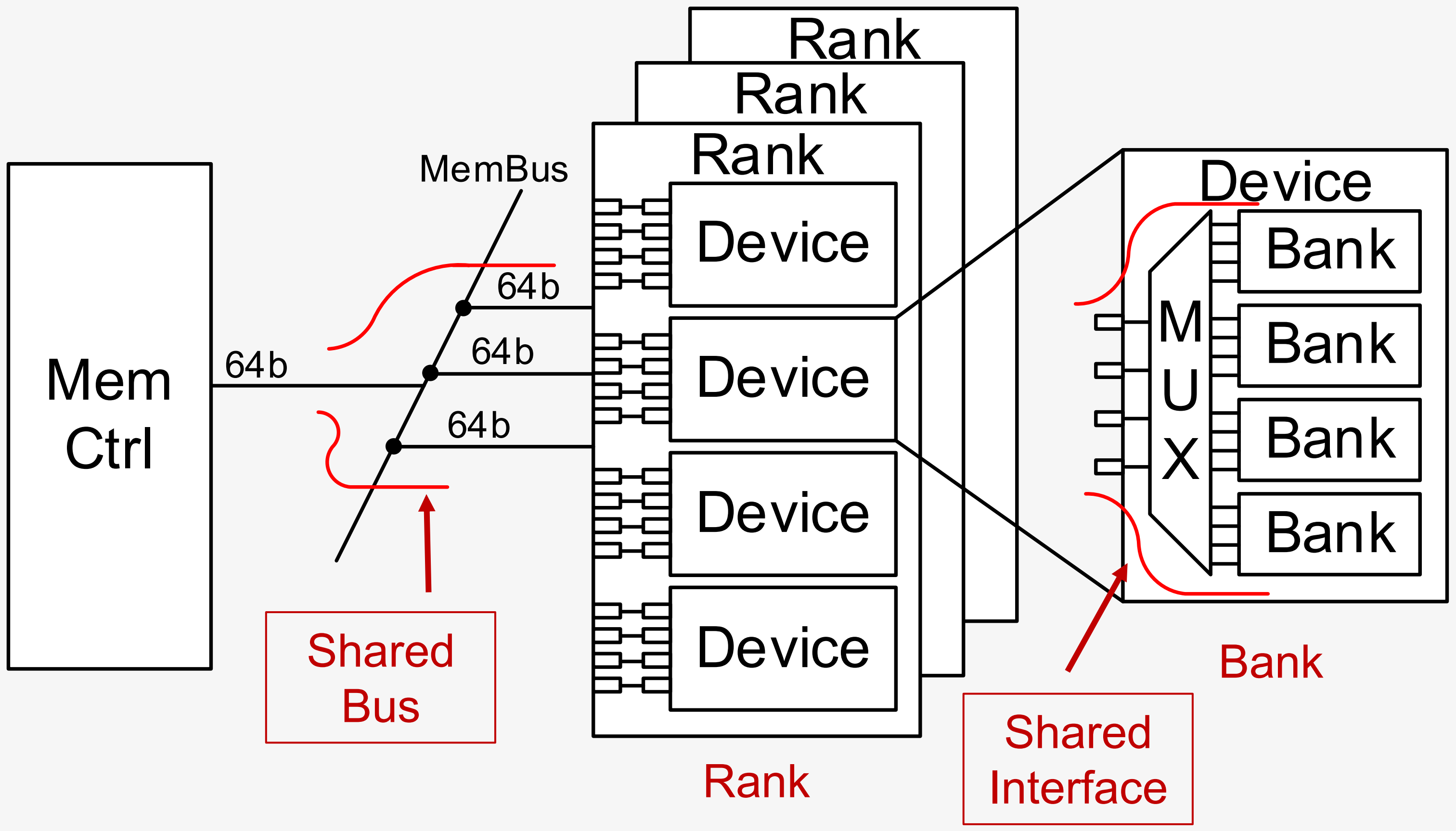


Attention on GPT2



Background: the "Memory Wall" in DRAM

- DRAM ranks/banks can only be accessed sequentially, and shares data line!

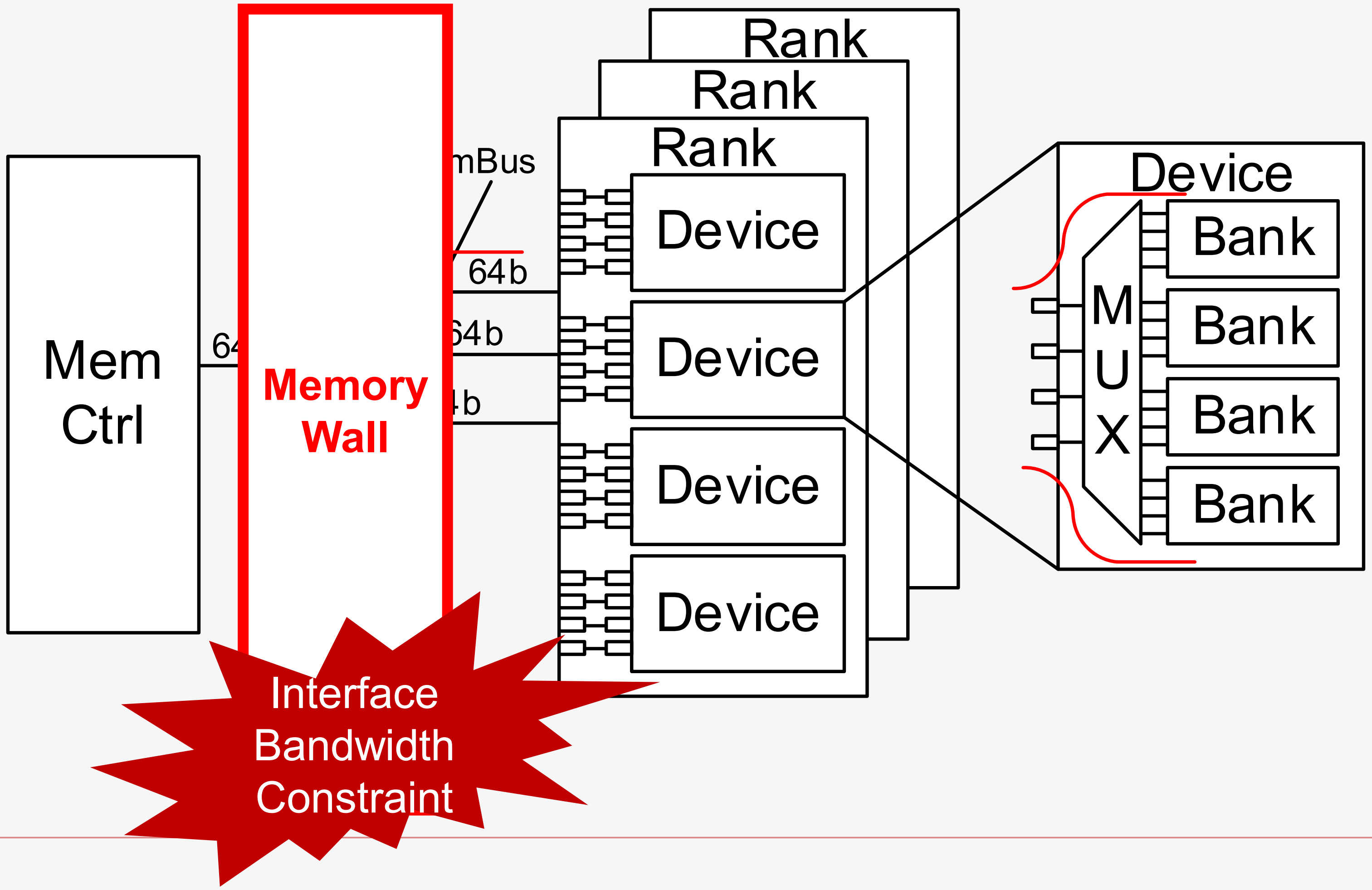


Ranks (Banks) share memory interface

Only one Rank (Bank) can be activated at the same time!

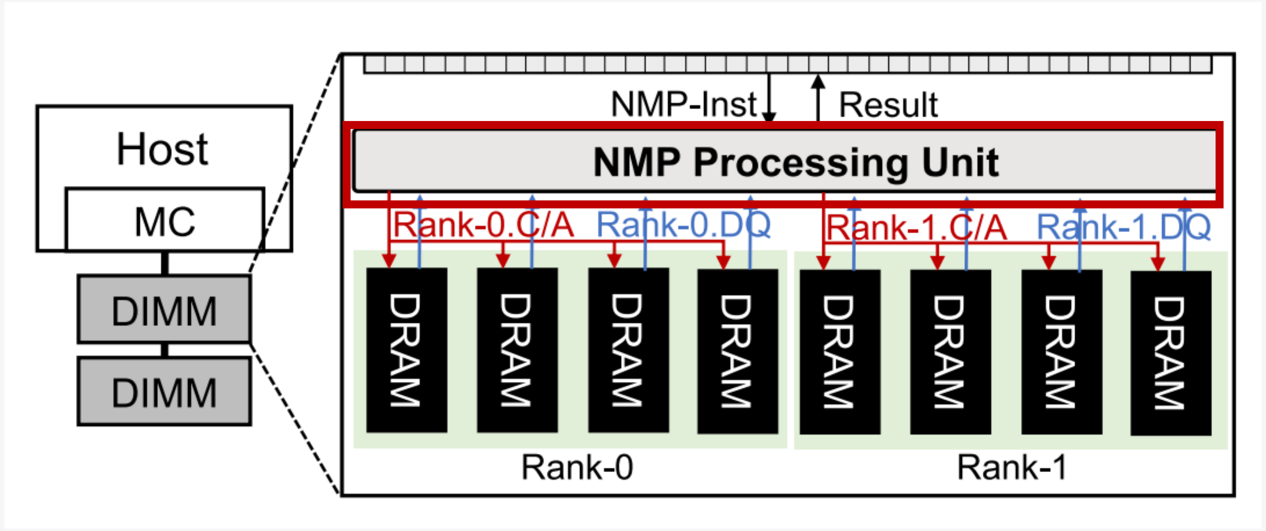
Background: the Memory Wall

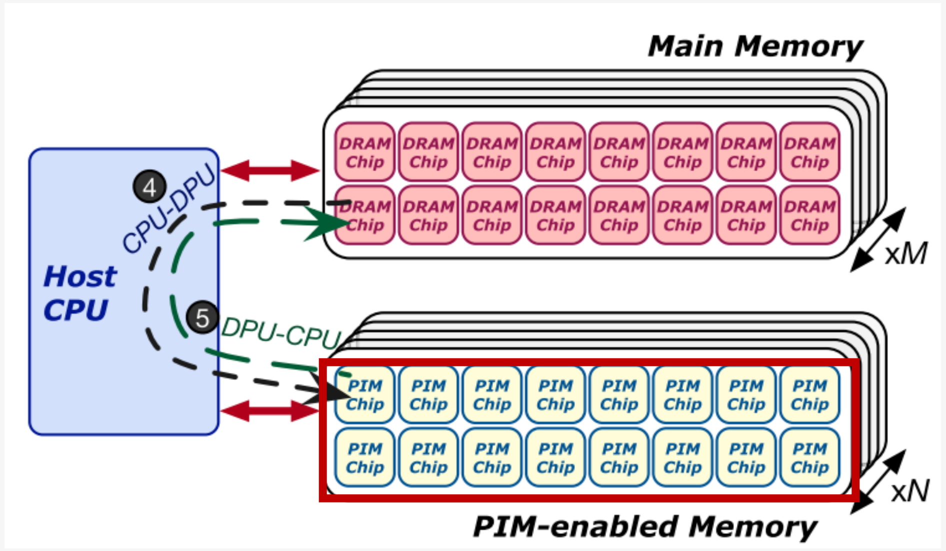
- The INTERFACE BANDWIDTH CONSTRAINT makes a **Memory Wall** Between CPU and DRAM!



Background: Process in Memory (PIM)

- Integrate computing units (**PIM units**) within DRAM to better utilize internal bandwidth



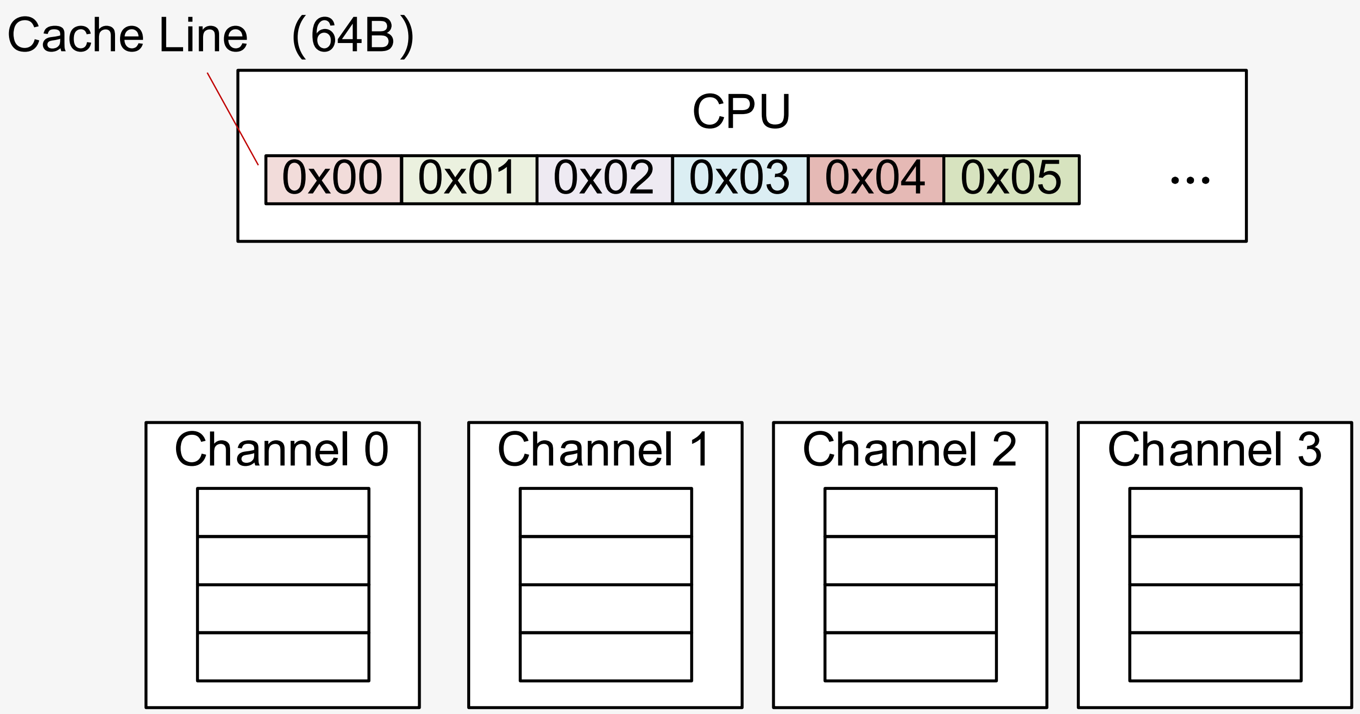


AxDIMM: in DIMM

UPMEM, AiM: In Bank

Memory Interleaving

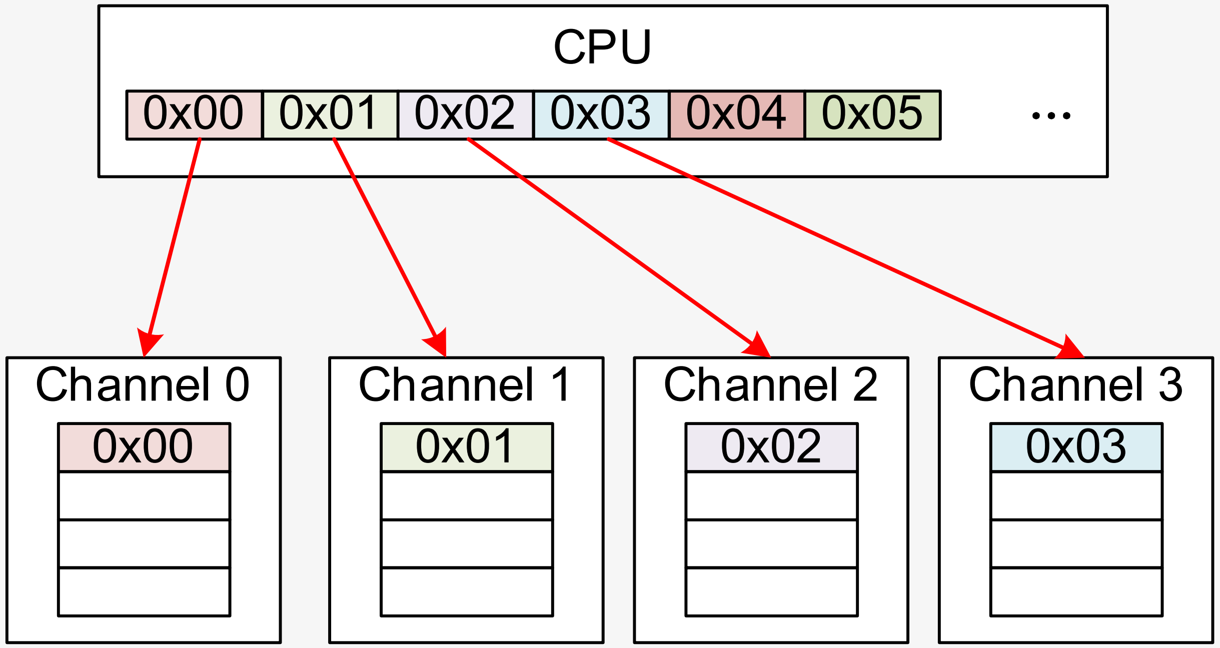
- To increase CPU bandwidth:
  - Adjacent data block stored in different channels
  - And Access in Parallel/pipeline



Memory Interleaving

To increase CPU bandwidth:

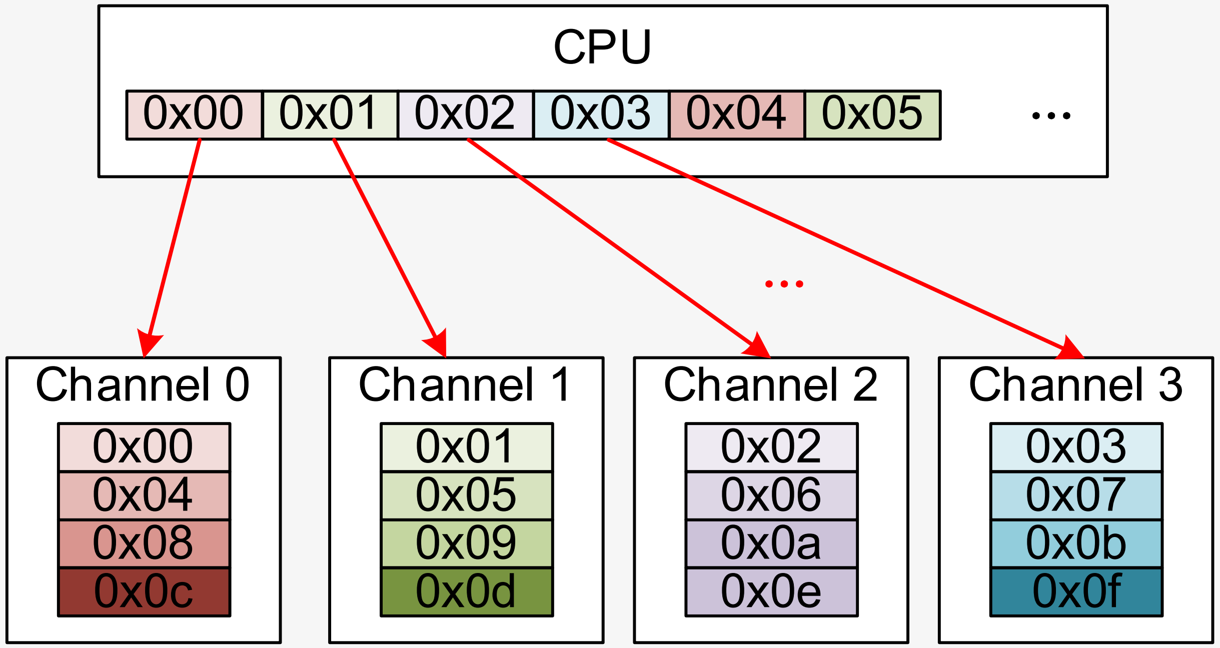
- Adjacent data block stored in different channels
- And Access in Parallel/pipeline



Memory Interleaving

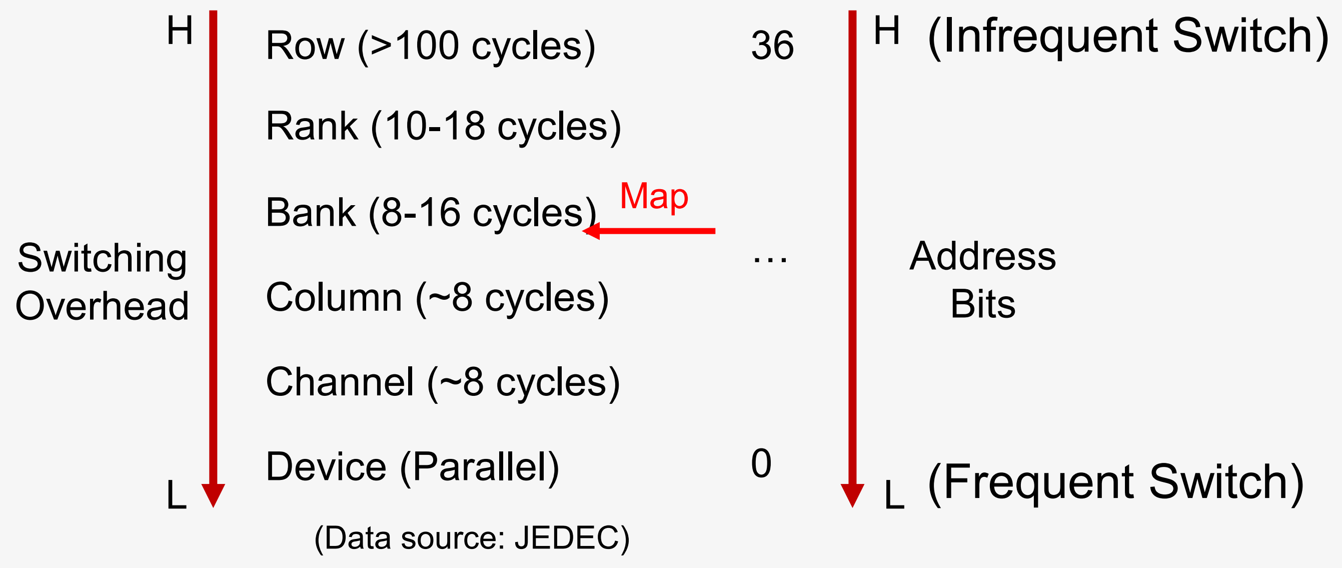
To increase CPU bandwidth:

- Adjacent data block stored in different channels
- And Access in Parallel/pipeline

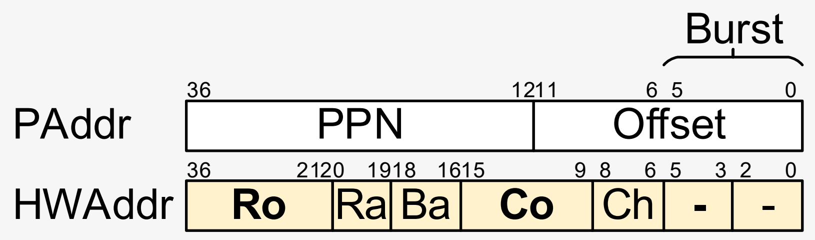


Memory Interleaving with Address Mapping

- According to **Switch Overhead**
- Map **lower**-order bits to **low**-switch-overhead levels, **higher**-order bits to **high**-switch-overhead levels

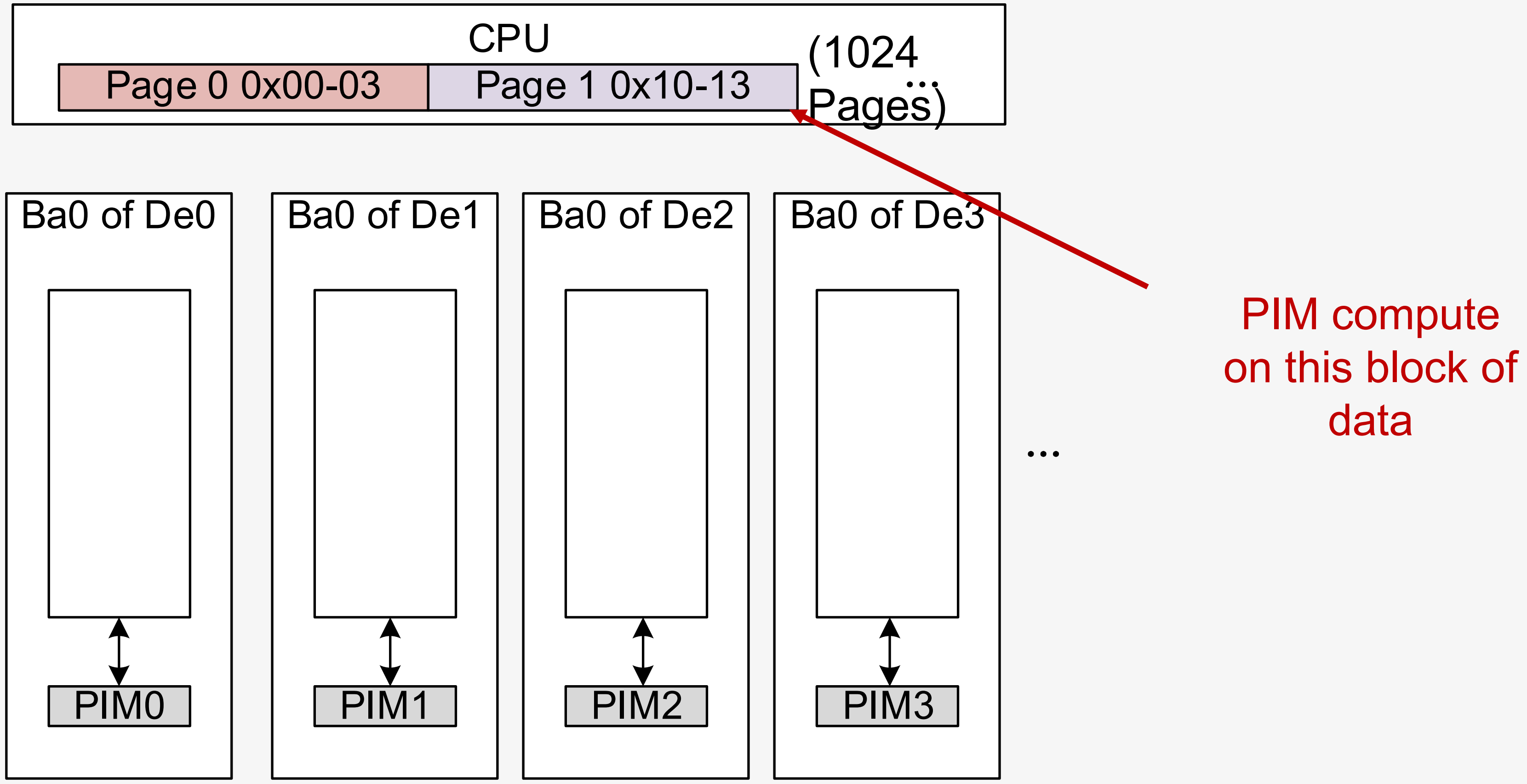


(Data source: JEDEC)



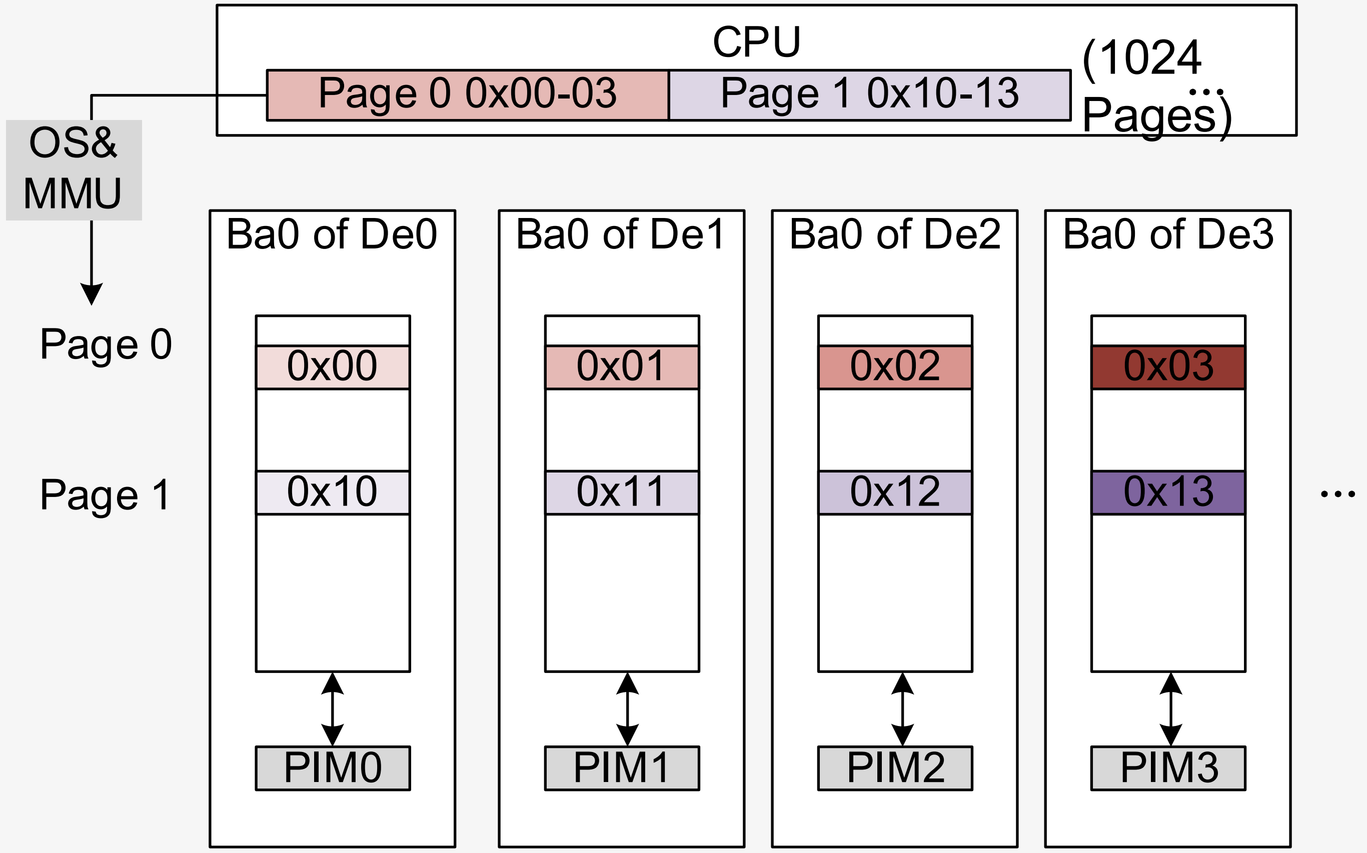
PIM vs Memory Management: Incompatibility

- Memory interleaving, virtual memory limit the length of contiguous data block visible to PIM, limit offload granularity



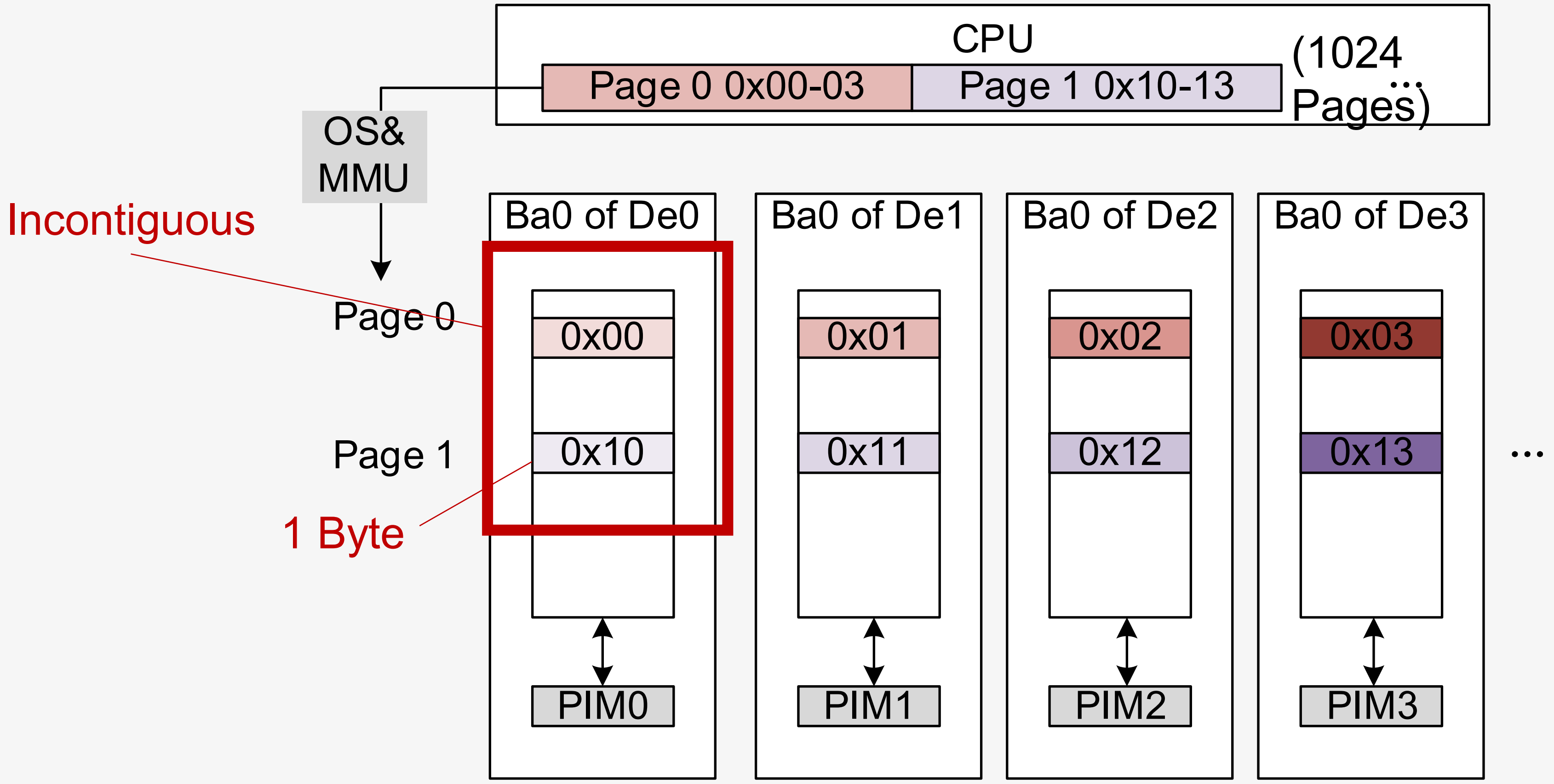
PIM vs Memory Management: Incompatibility

- Memory interleaving, virtual memory limit the length of contiguous data block visible to PIM, limit offload granularity



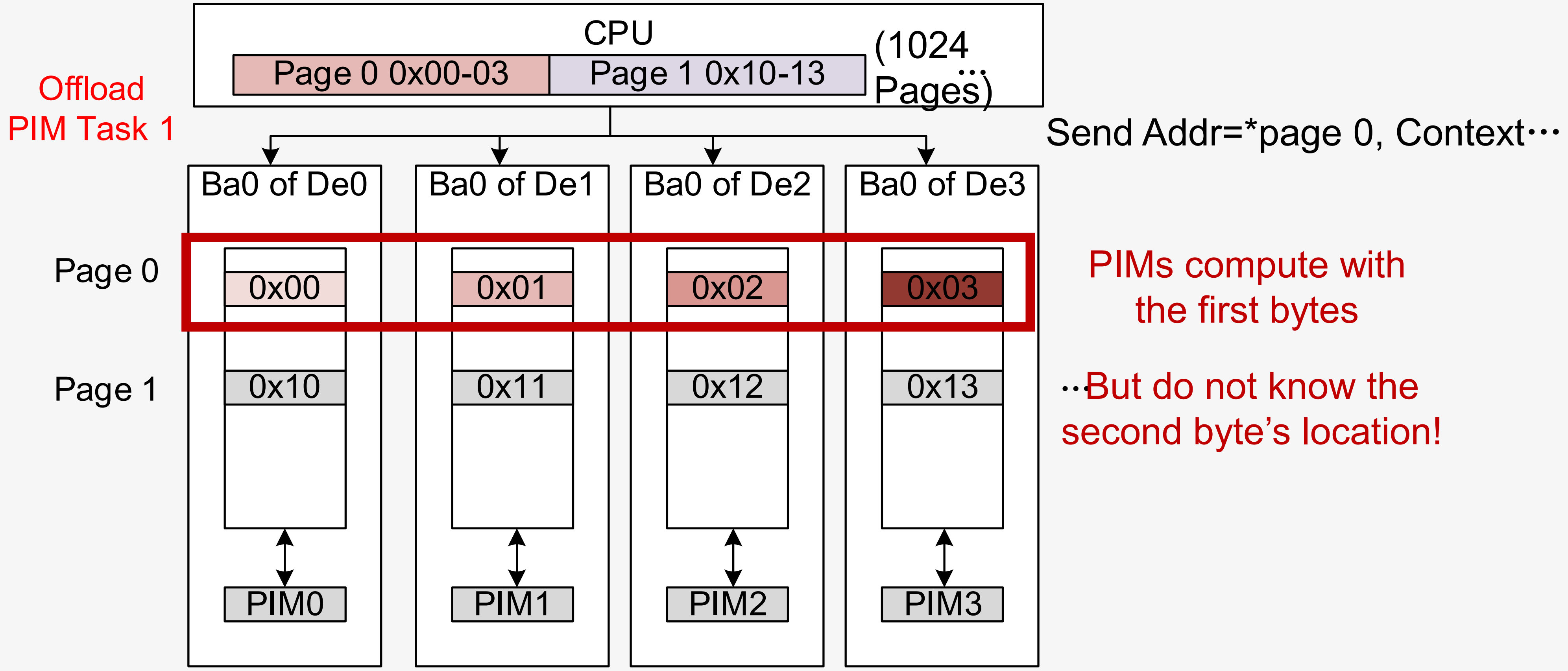
PIM vs Memory Management: Incompatibility

- Memory interleaving, virtual memory limit the length of contiguous data block visible to PIM, limit offload granularity



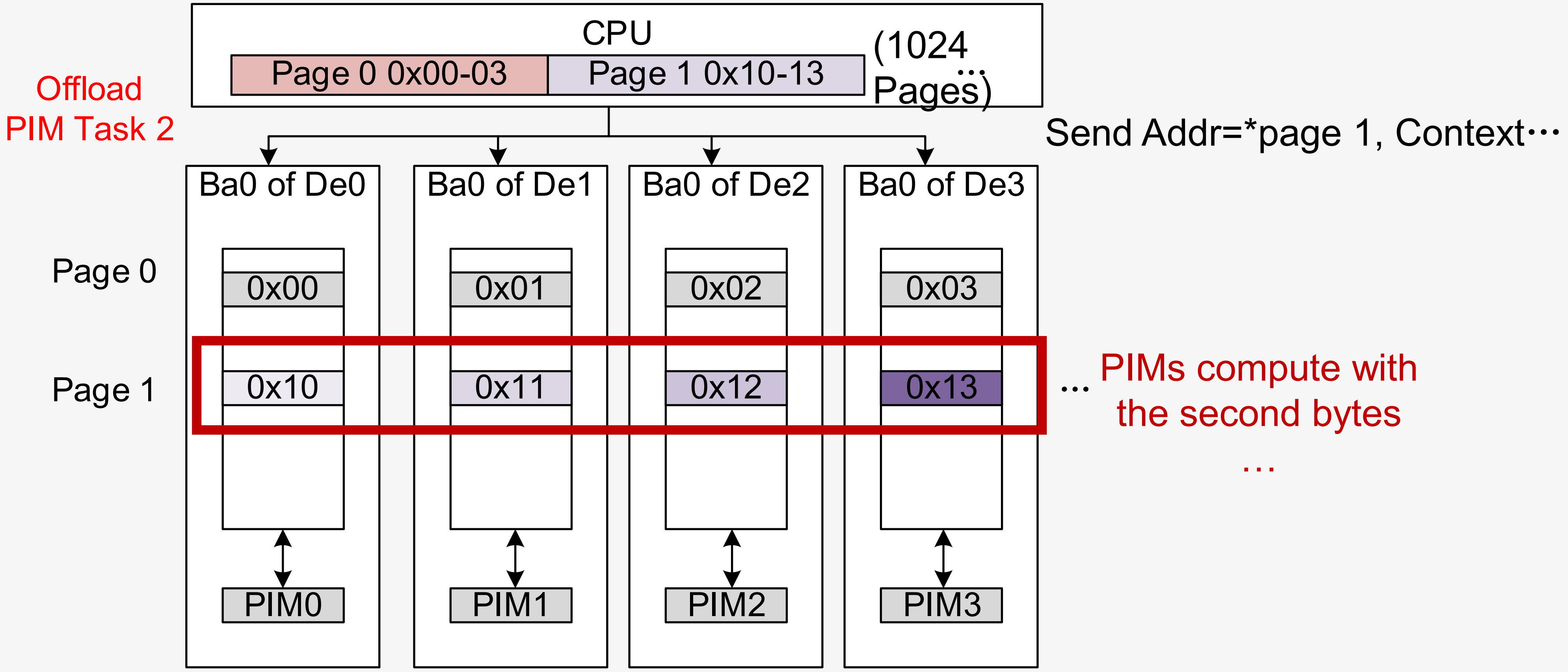
PIM vs Memory Management: Incompatibility

- Memory interleaving, virtual memory limit the length of contiguous data block visible to PIM, limit offload granularity



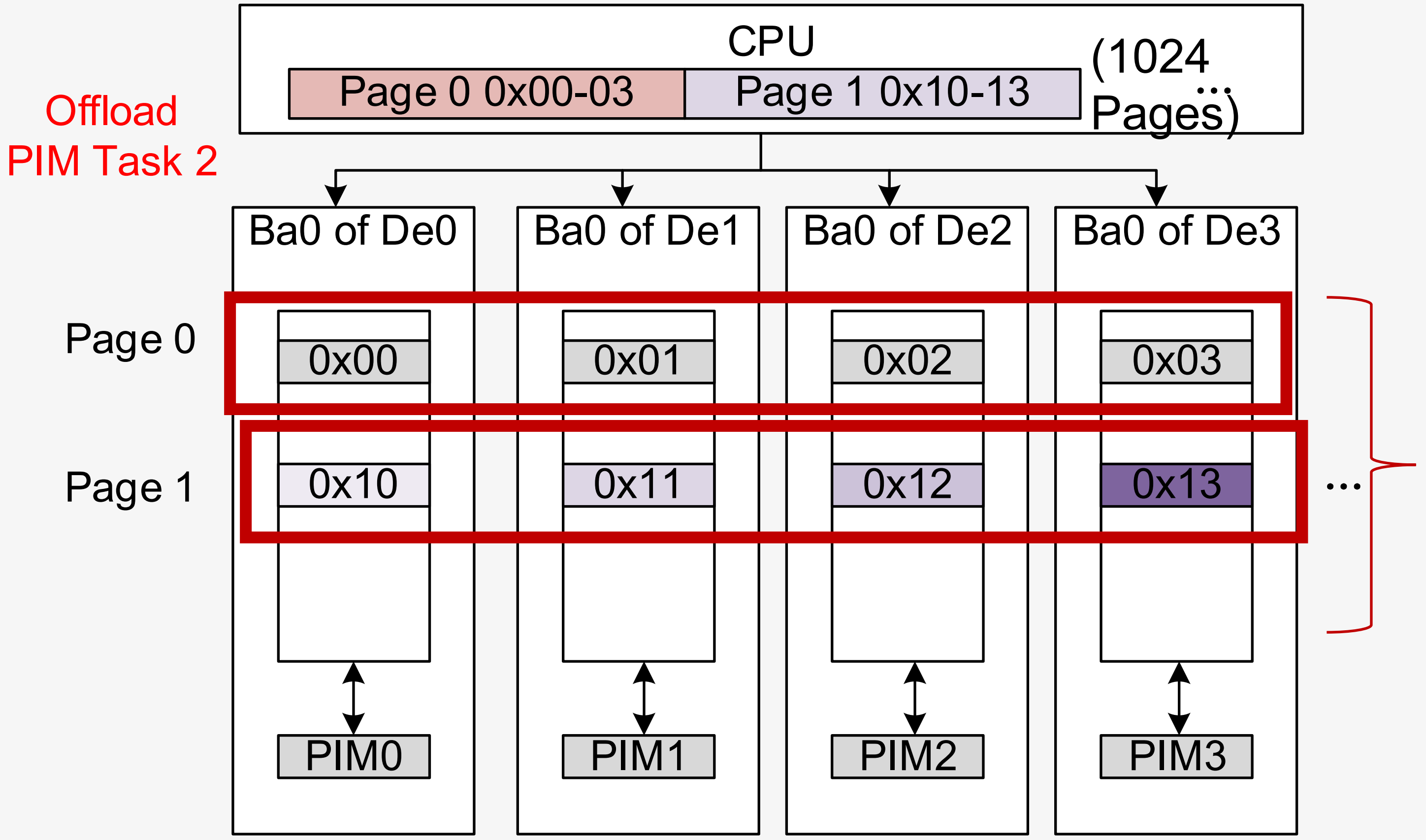
PIM vs Memory Management: Incompatibility

- Memory interleaving, virtual memory limit the length of contiguous data block visible to PIM, limit offload granularity



PIM vs Memory Management: Incompatibility

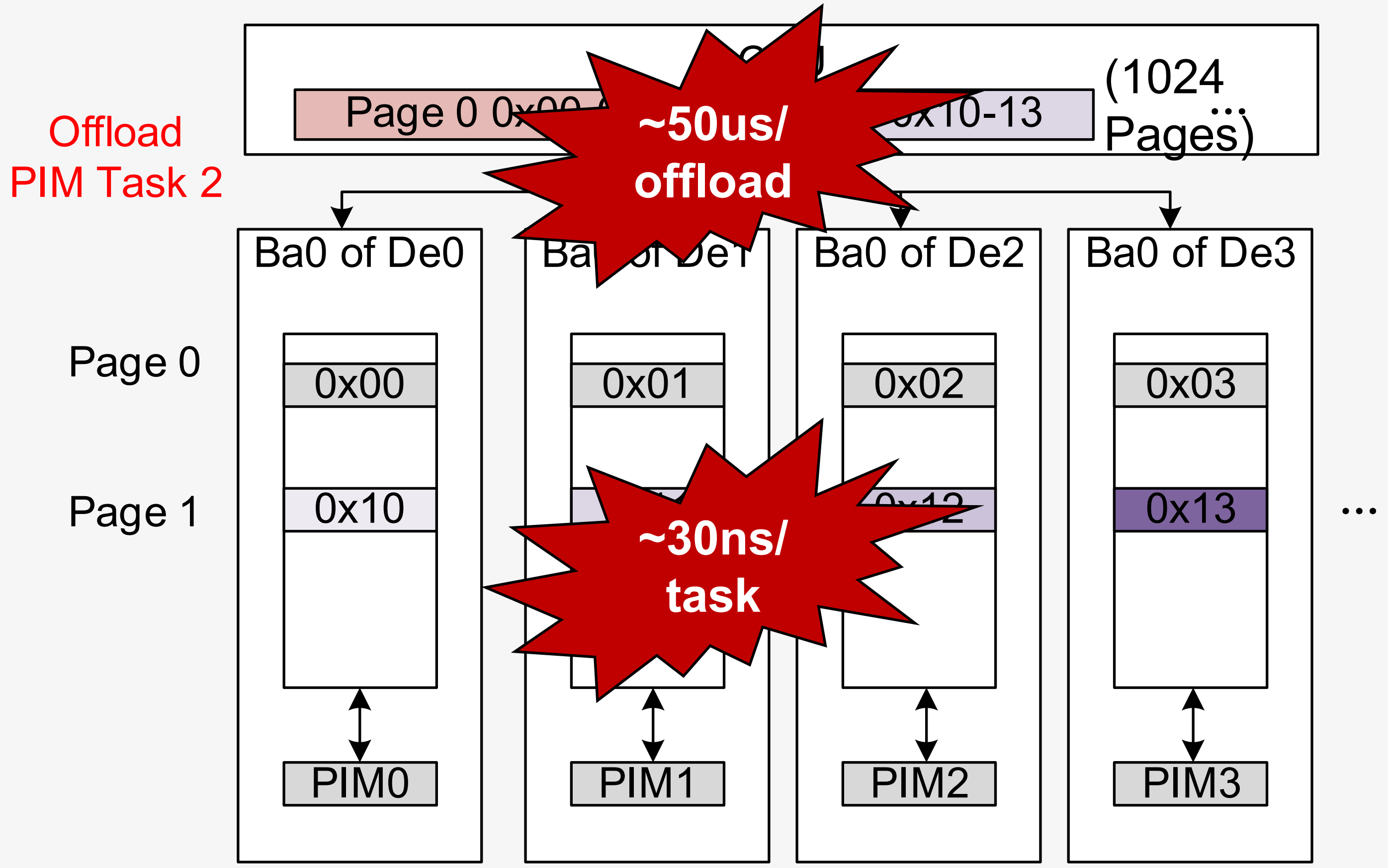
- Memory interleaving, virtual memory limit the length of contiguous data block visible to PIM, limit offload granularity



Need **Many Offloads** for the Task!

PIM vs Memory Management: Incompatibility

- Memory interleaving, virtual memory limit the length of contiguous data block visible to PIM, limit offload granularity



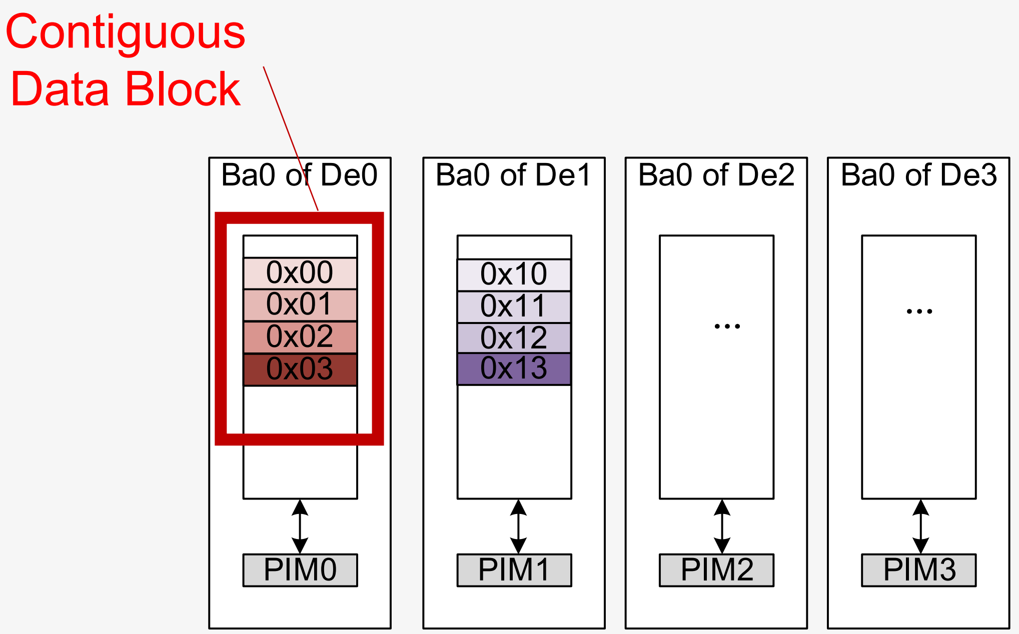
Need 1024 Offloads for the Task!

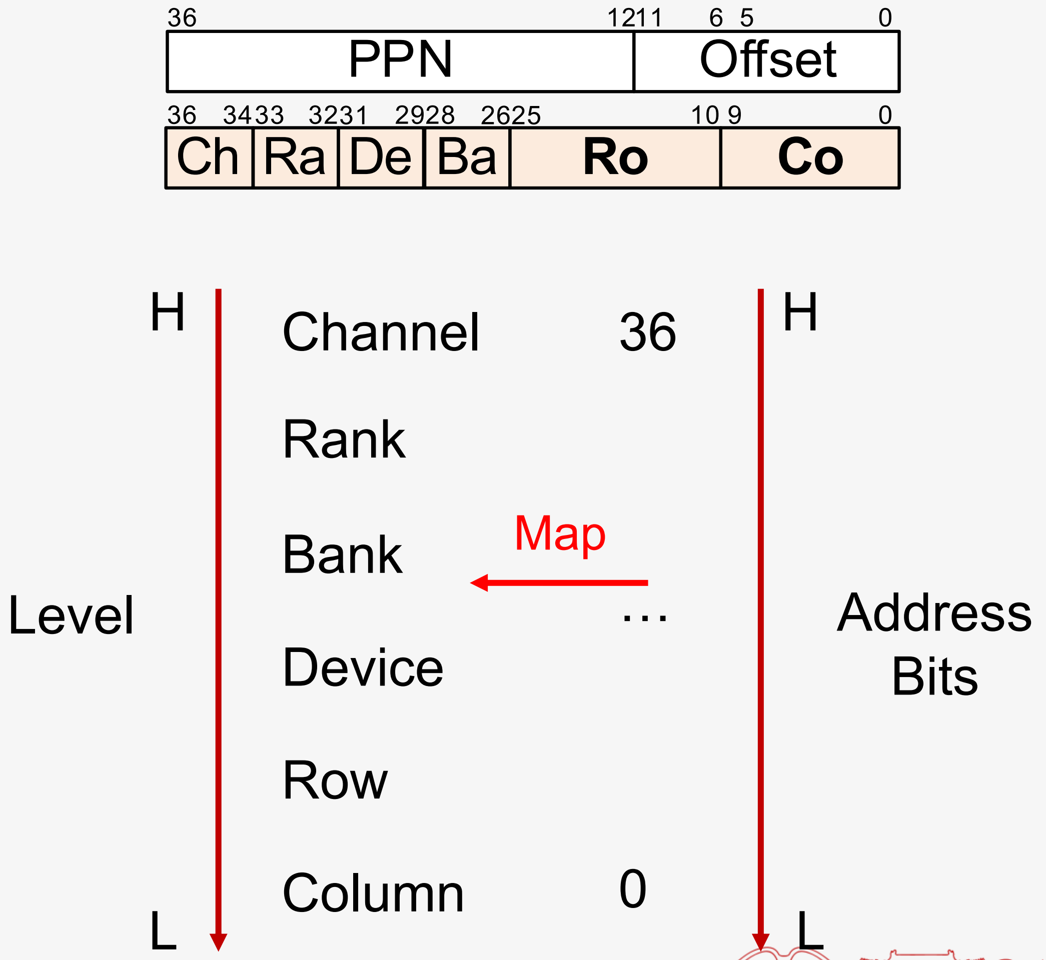
50us offload
30ns task
Offload >> Task
In-efficient Offload

PIM's ideal address mapping

- PIM prefers contiguous data block: address mapping according to level

Only Need One Offload: Less offload overhead

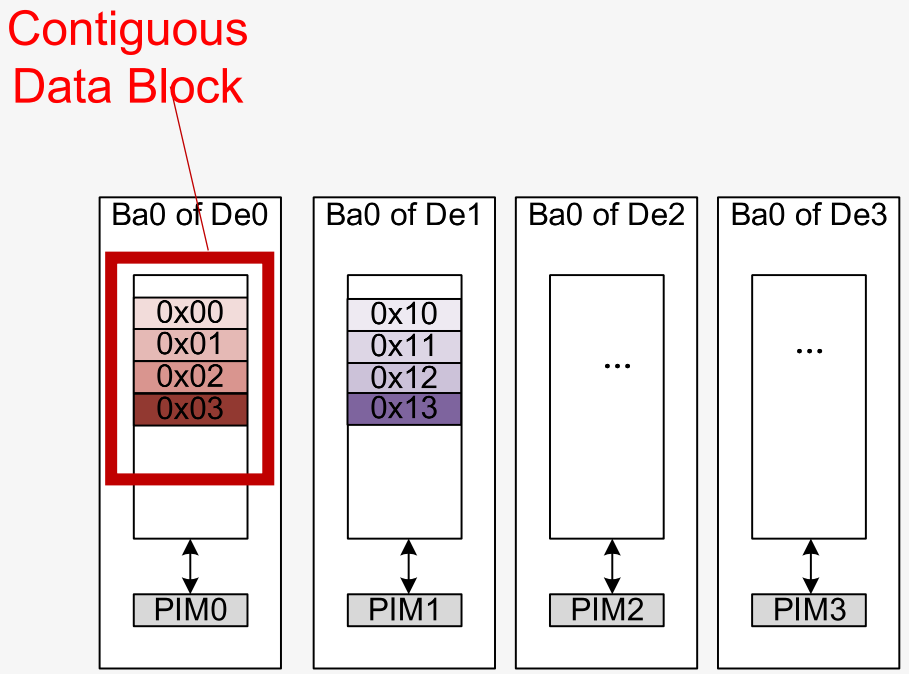


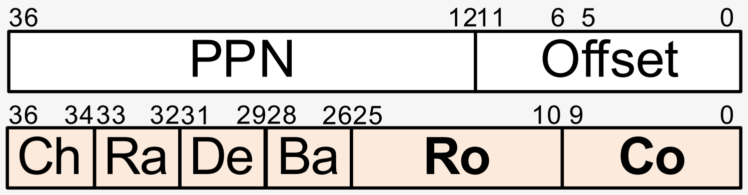


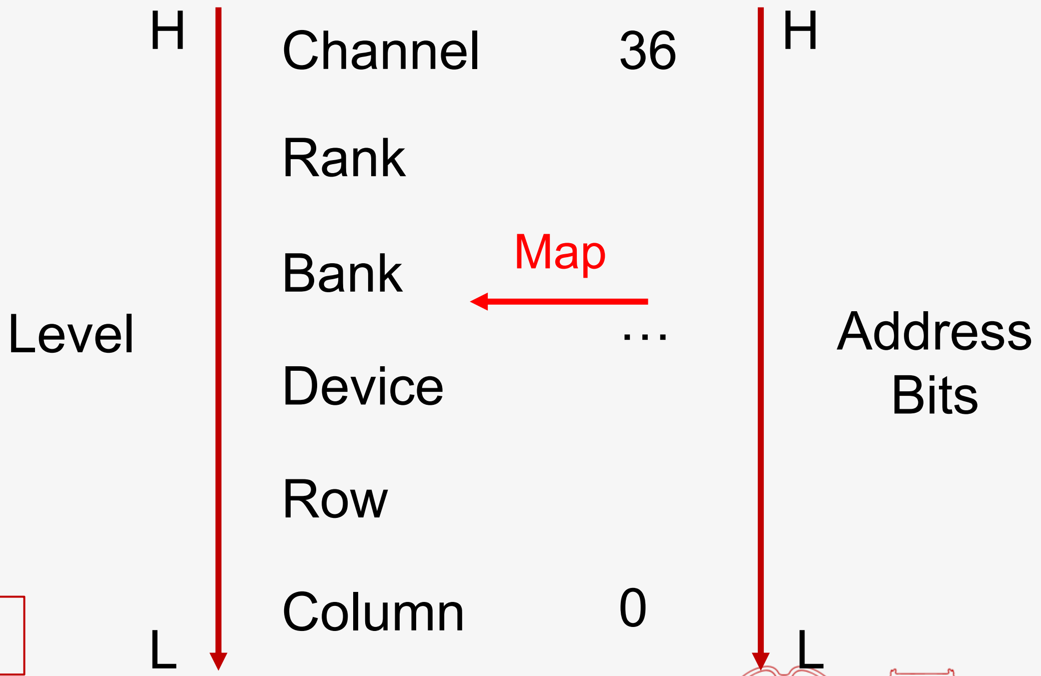
PIM's ideal address mapping

- PIM prefers contiguous data block: address mapping according to level

Only Need One Offload: Less offload overhead



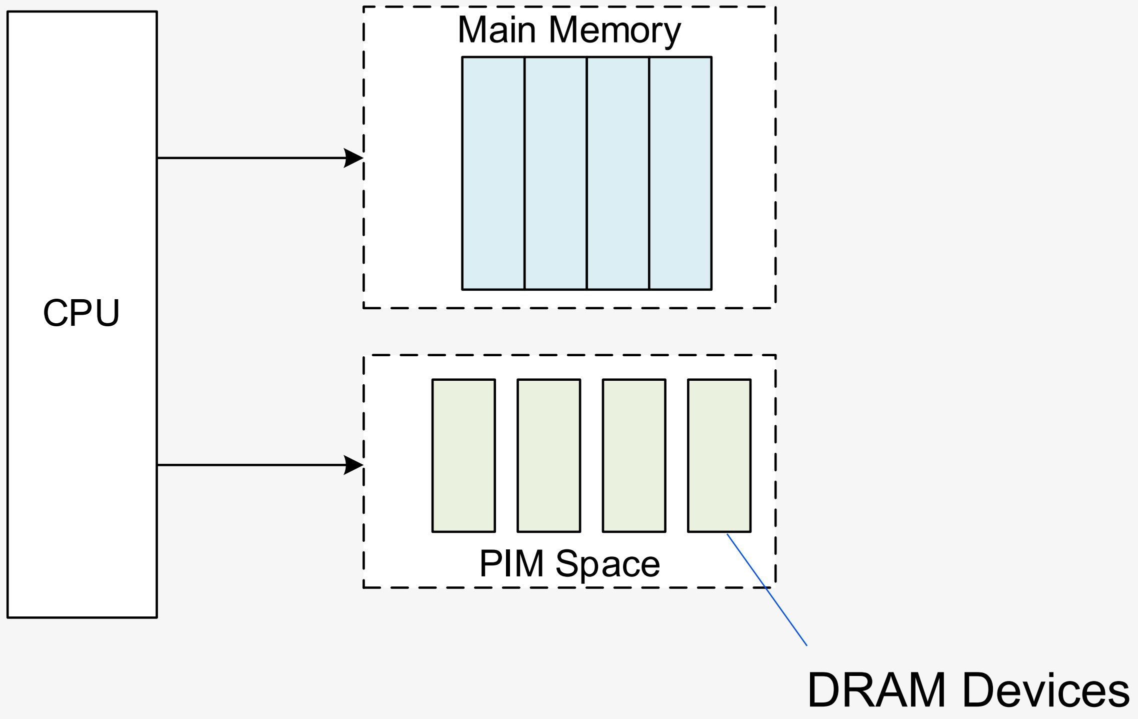




CPU & PIM require different Data Layouts!

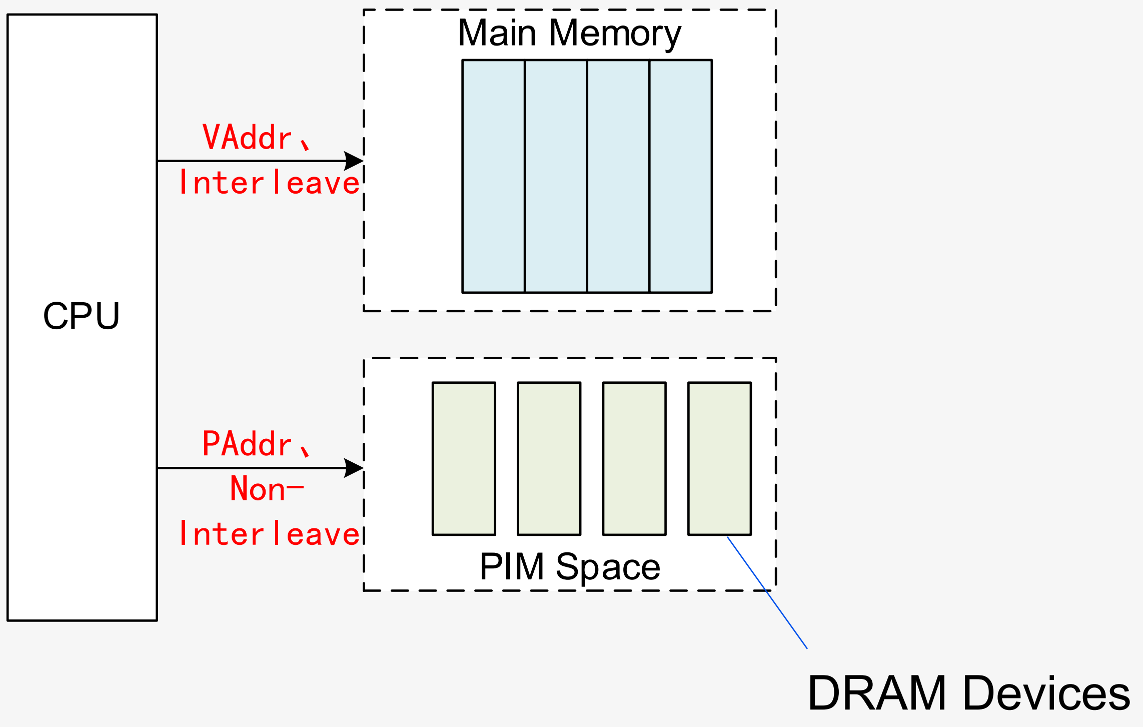
Measures of current PIM systems

Measure 1: Isolated Memory Space



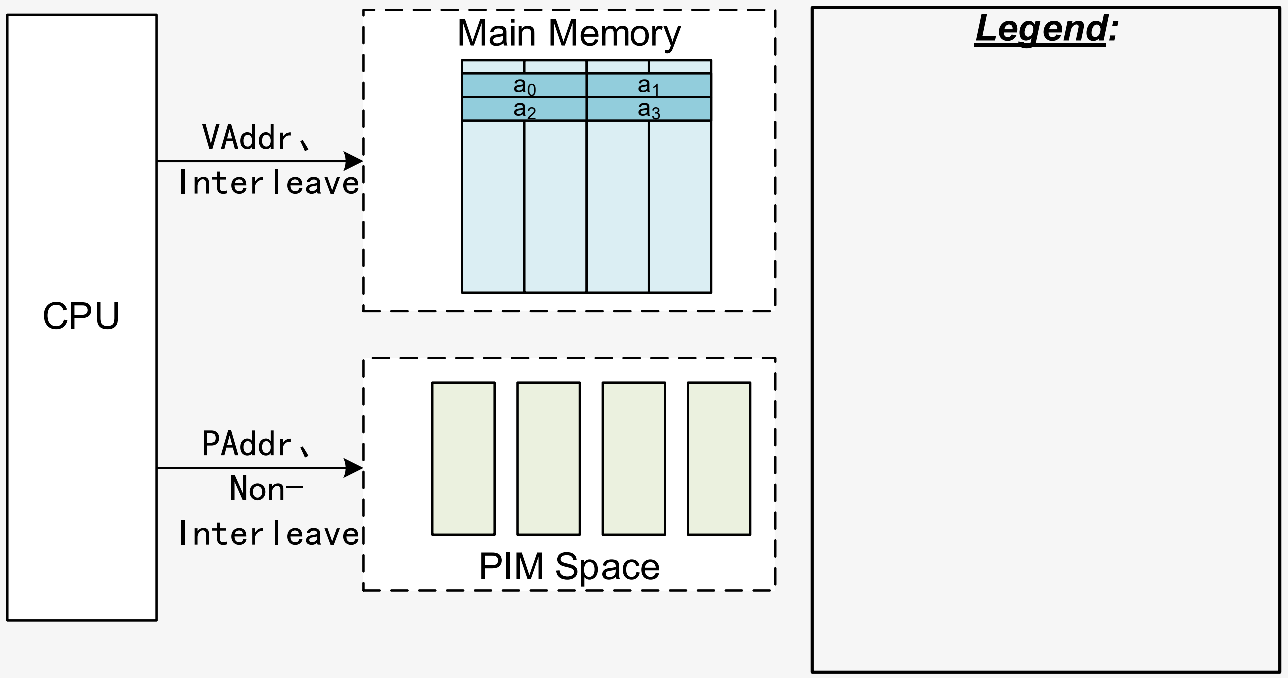
Measures of current PIM systems

Measure 1: Isolated Memory Space



Measures of current PIM systems

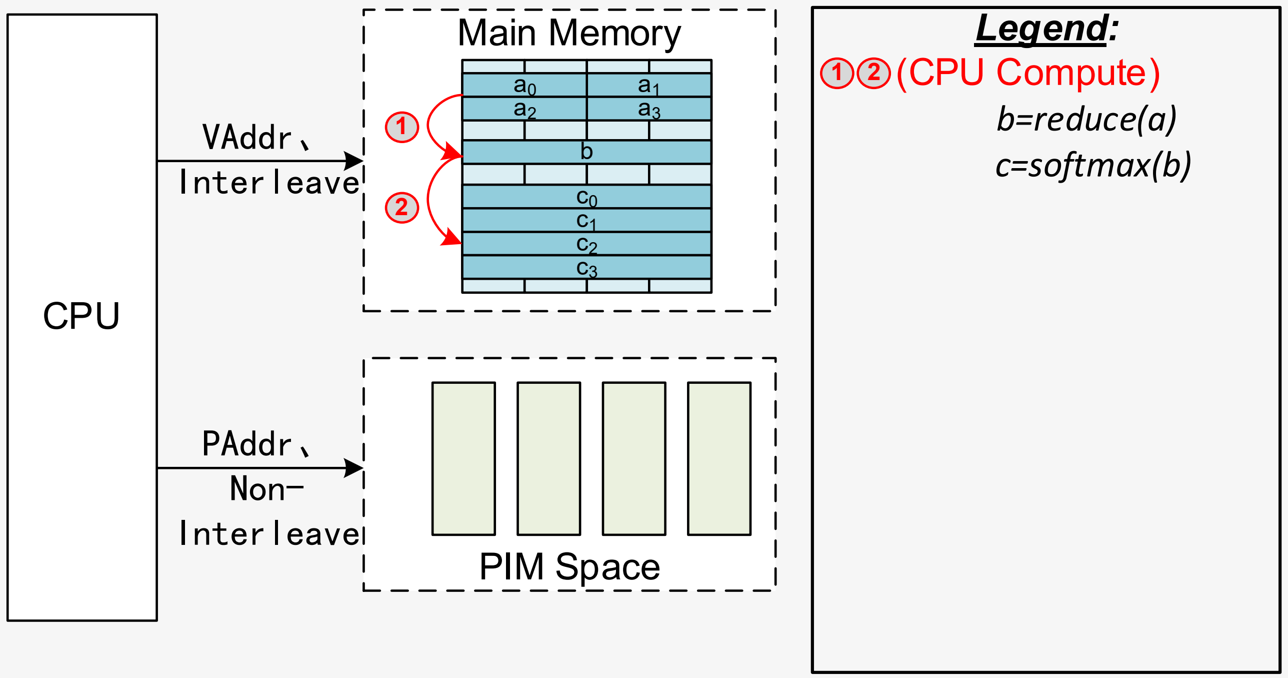
Measure 1: Isolated Memory Space



Example: MLP

Measures of current PIM systems

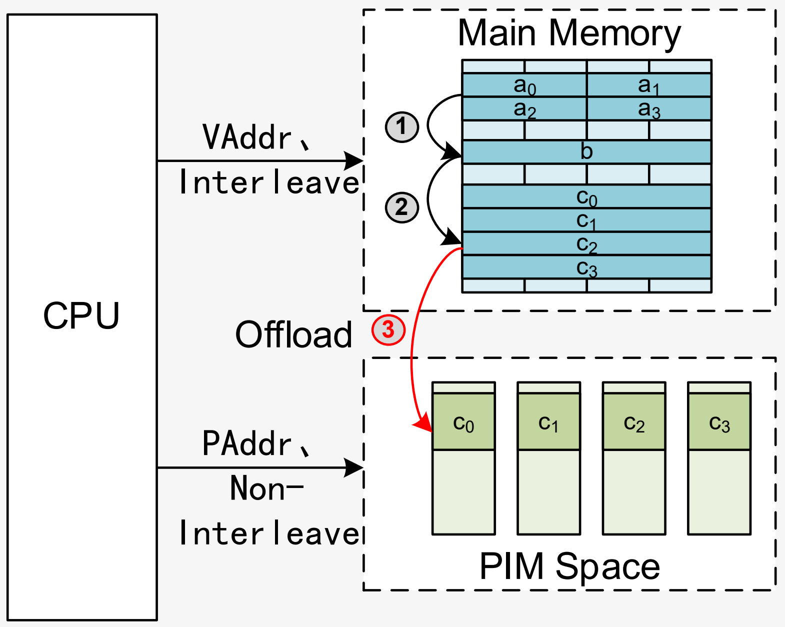
Measure 1: Isolated Memory Space



Example: MLP

Measures of current PIM systems

- Measure 1: Isolated Memory Space **(Resulting in Data Transfer)**



***Legend*:**

①② (CPU Compute)

b=reduce(a)

c=softmax(b)

③ (CPU-PIM Move)

buf_0=reshape(c_0)

memcpy(a_0, buf_0)

buf_1=reshape(c_1)

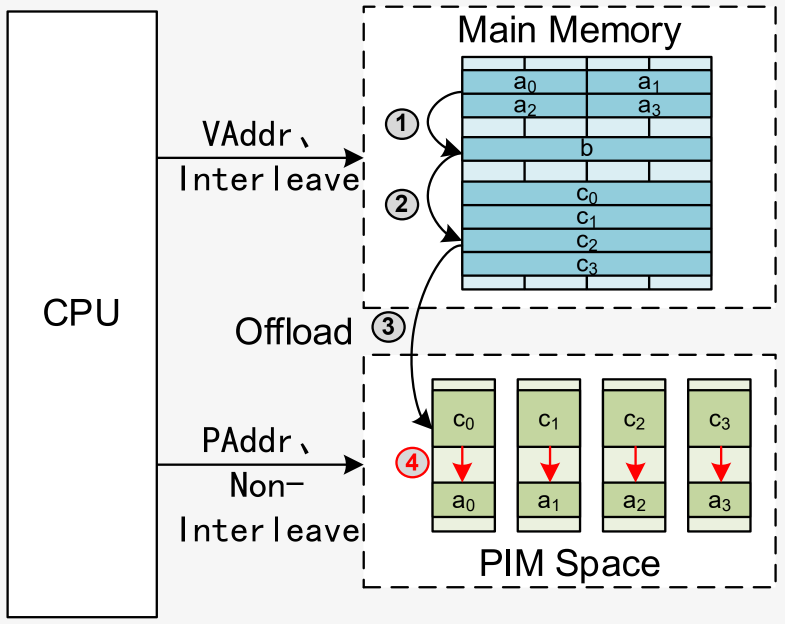
...

Offload

Example: MLP

Measures of current PIM systems

Measure 1: Isolated Memory Space **(Resulting in Data Transfer)**



***Legend*:**

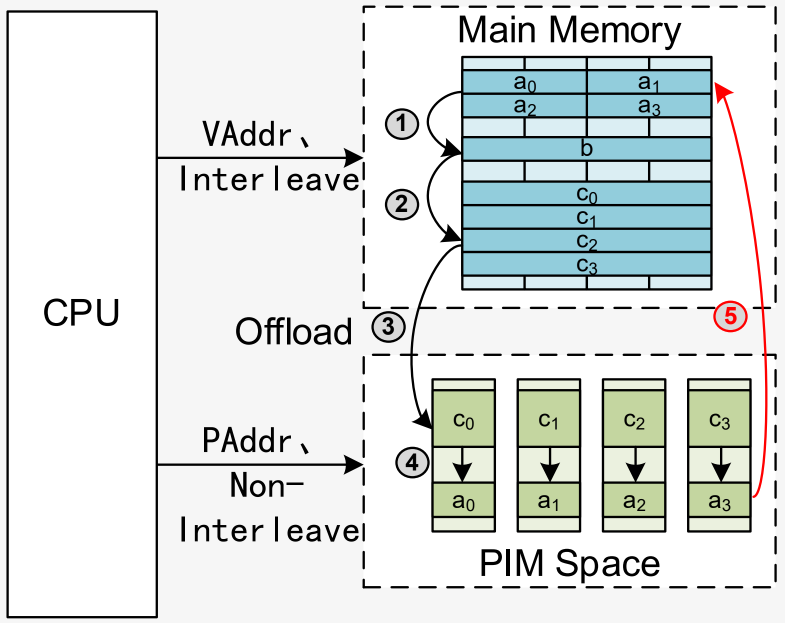
①② (CPU Compute)
  b=reduce(a)
  c=softmax(b)

Offload
③ (CPU-PIM Move)
  buf_0=reshape(c_0)
  memcpy(a_0, buf_0)
  buf_1=reshape(c_1)
  …
④ (PIM Compute)
  a=gemm(c)

Example: MLP

Measures of current PIM systems

- Measure 1: Isolated Memory Space **(Resulting in Data Transfer)**



***Legend*:**

①②(CPU Compute)

b=reduce(a)

c=softmax(b)

Offload

③(CPU-PIM Move)

buf_0=reshape(c_0)

memcpy(a_0, buf_0)

buf_1=reshape(c_1)

...

④(PIM Compute)

a=gemm(c)

⑤(PIM-CPU Move)

buf_0=reshape(c_0)

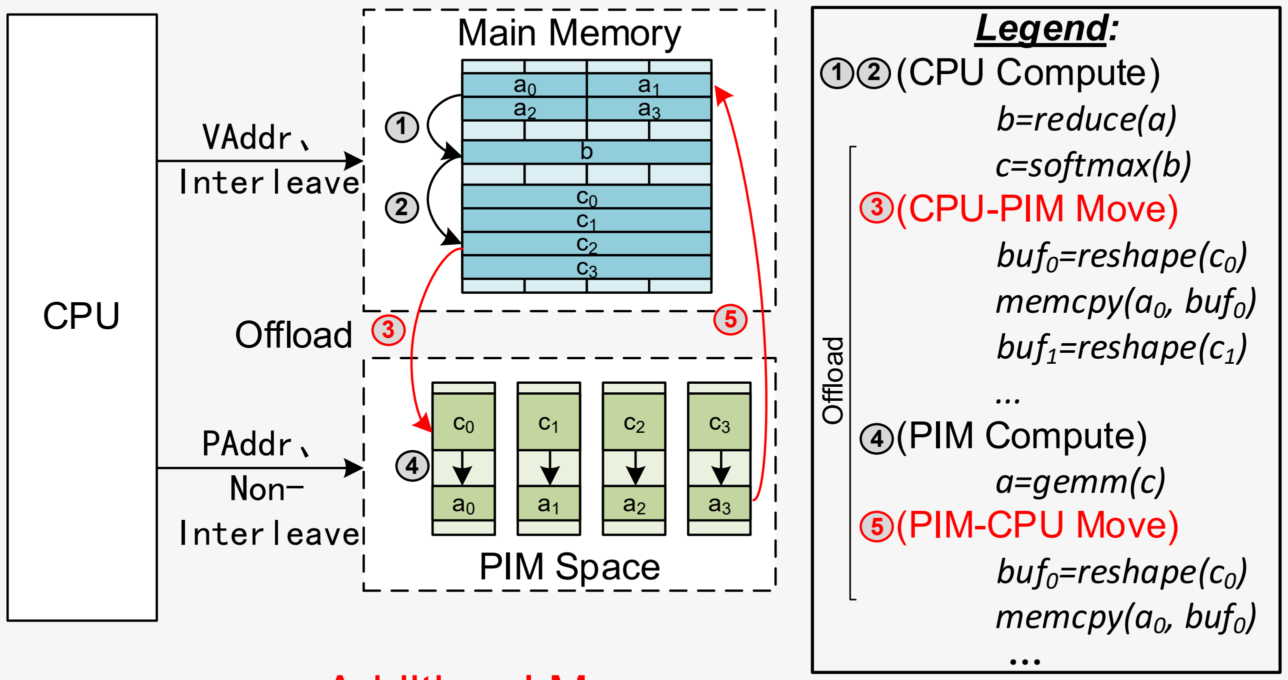
memcpy(a_0, buf_0)

...

Example: MLP

Measures of current PIM systems

Measure 1: Isolated Memory Space **(Resulting in Data Transfer)**



Legend:

①②(CPU Compute)

b=reduce(a)

c=softmax(b)

③(CPU-PIM Move)

buf_0=reshape(c_0)

memcpy(a_0, buf_0)

buf_1=reshape(c_1)

...

④(PIM Compute)

a=gemm(c)

⑤(PIM-CPU Move)

buf_0=reshape(c_0)

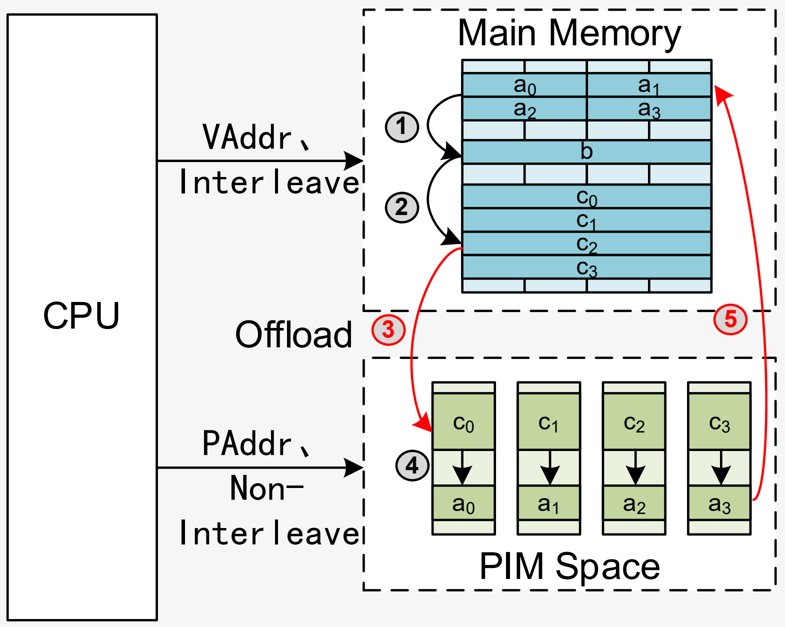
memcpy(a_0, buf_0)

...

Additional Memory copy

Measures of current PIM systems

Measure 1: Isolated Memory Space (Resulting in Data Transfer)



Legend:

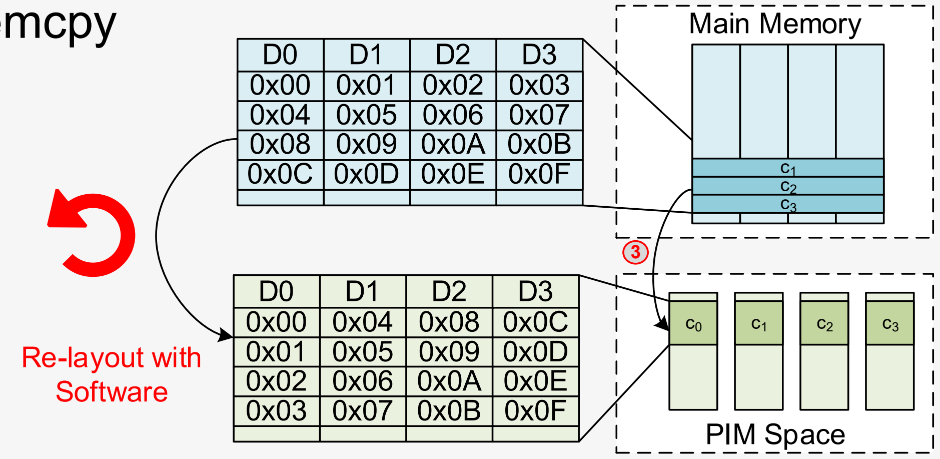
①② (CPU Compute)

```
    b=reduce(a)
    c=softmax(b)
③(CPU-PIM Move)
    buf0=reshape(c0)
    memcpy(a0, buf0)
    buf1=reshape(c1)
    ...
④(PIM Compute)
    a=gemm(c)
⑤(PIM-CPU Move)
    buf0=reshape(c0)
    memcpy(a0, buf0)
    ...
```

Offload

Data Re-layout
(Because of different data layout)

Memcpy



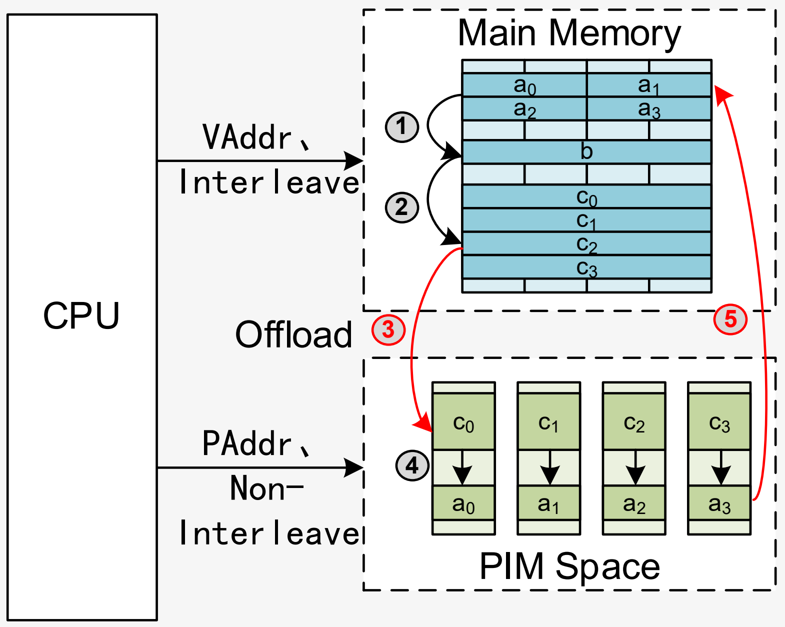
D0	D1	D2	D3
0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

D0	D1	D2	D3
0x00	0x04	0x08	0x0C
0x01	0x05	0x09	0x0D
0x02	0x06	0x0A	0x0E
0x03	0x07	0x0B	0x0F

Additional Memory copy and data re-layout

Measures of current PIM systems

- Measure 1: Isolated Memory Space (Resulting in Data Transfer)



Legend:

①②(CPU Compute)

b=reduce(a)

c=softmax(b)

③(CPU-PIM Move)

buf_0=reshape(c_0)

memcpy(a_0, buf_0)

buf_1=reshape(c_1)

...

④(PIM Compute)

a=gemm(c)

⑤(PIM-CPU Move)

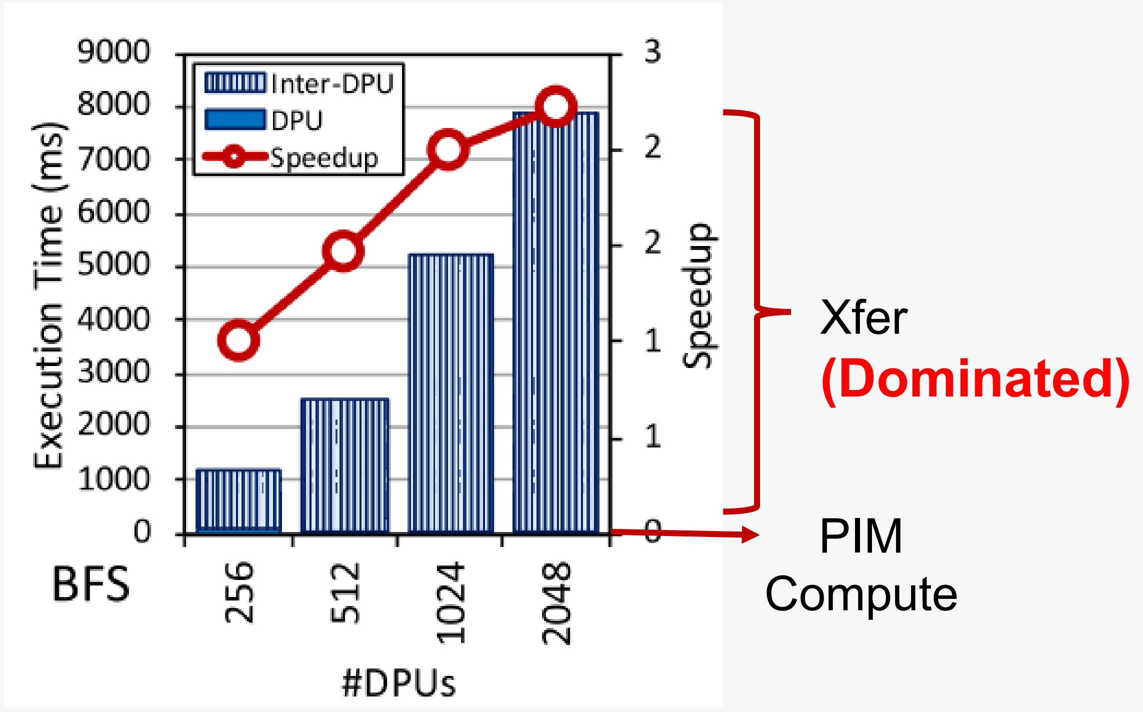
buf_0=reshape(c_0)

memcpy(a_0, buf_0)

...

Offload

Additional Memory copy and data re-layout



Gómez-Luna et.al. 2022

Measures of current PIM systems

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Measure 2： Switch off channel & rank interleaving

- Reduce re-layout overhead, but damage CPU bandwidth
- The data re-layout overhead still accounts for 70% of data transfer time

Save Computations

Ch on, Ba on $\rightarrow$ Ch off, Ba on

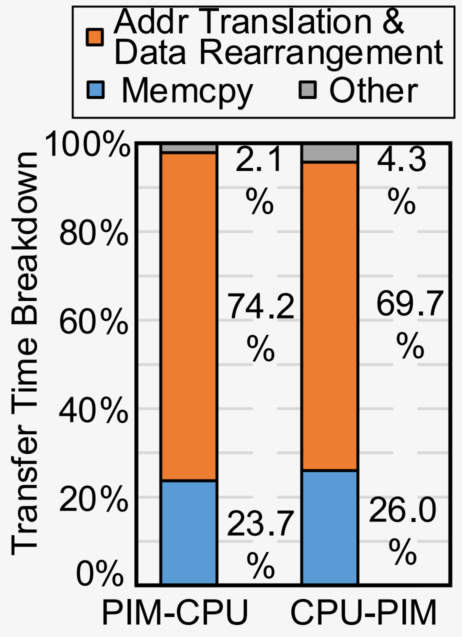
$PAddr| = Ch \ll m \mid Ba \ll n$ $PAddr| = Ba \ll n$

C0B0	C0B1	C1B0	C1B1
0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

C0B0	C0B1	C1B0	C1B1
0x00	0x01	0x08	0x09
0x02	0x03	0x0A	0x0B
0x04	0x05	0x0C	0x0D
0x06	0x07	0x0E	0x0F

Re-layout with Software

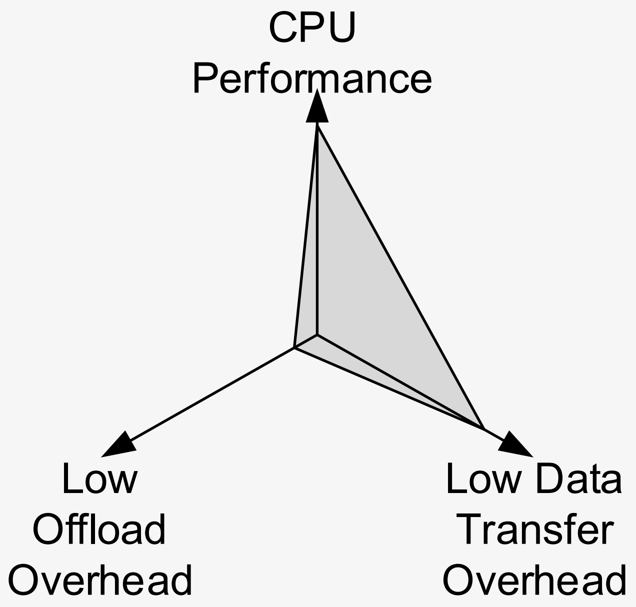
C0B0	C0B1	C1B0	C1B1
0x00	0x04	0x08	0x0C
0x01	0x05	0x09	0x0D
0x02	0x06	0x0A	0x0E
0x03	0x07	0x0B	0x0F

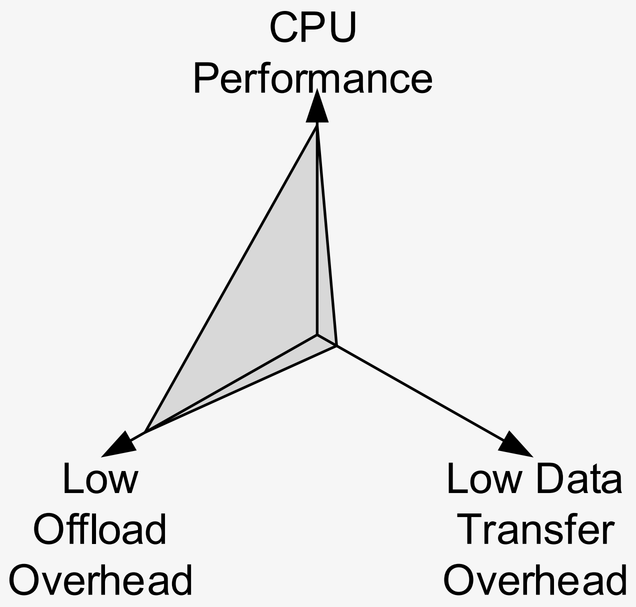


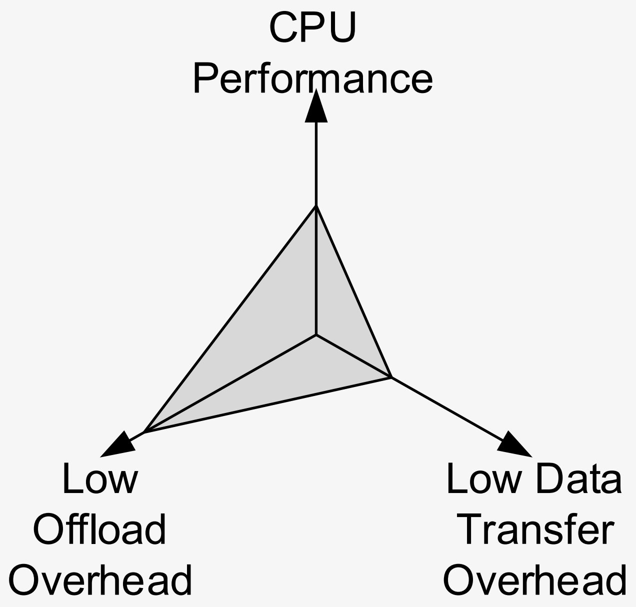
UPMEM Data Transfer Breakdown

Summary: An Impossible Triangle

- Memory spaces with single data layout.







(Naïve)
Uniform Layout
Fine-grained Offload

Isolated Memory Space
Software Re-layout

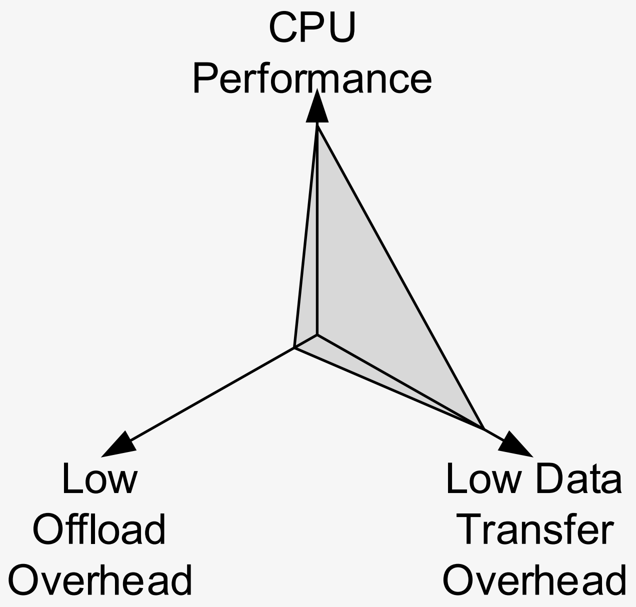
MetaPNM,PIM-HBM

Isolated Memory Space
Switch off Interleaving

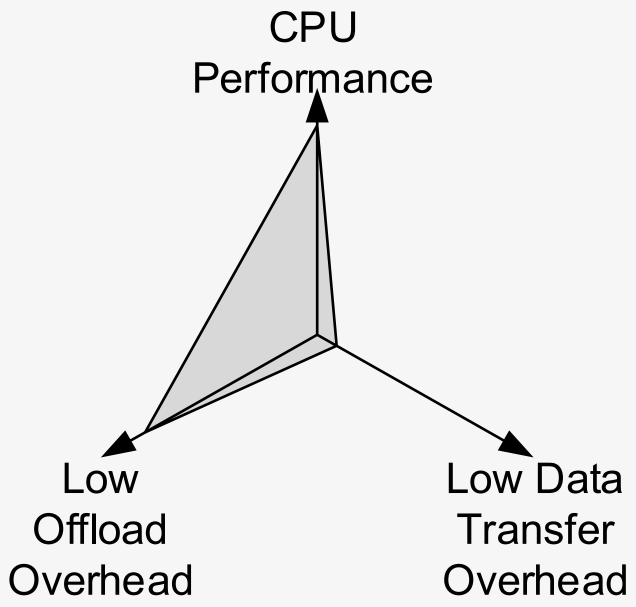
AxDIMM, UPMEM

Summary: An Impossible Triangle

- Memory spaces ~~with single data layout.~~

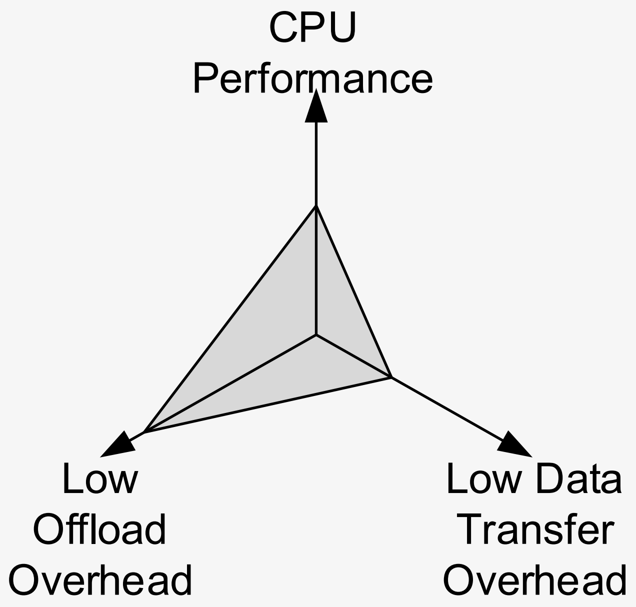


(Naïve)
Uniform Layout
Fine-grained Offload



Isolated Memory Space
Software Re-layout

MetaPNM,PIM-HBM

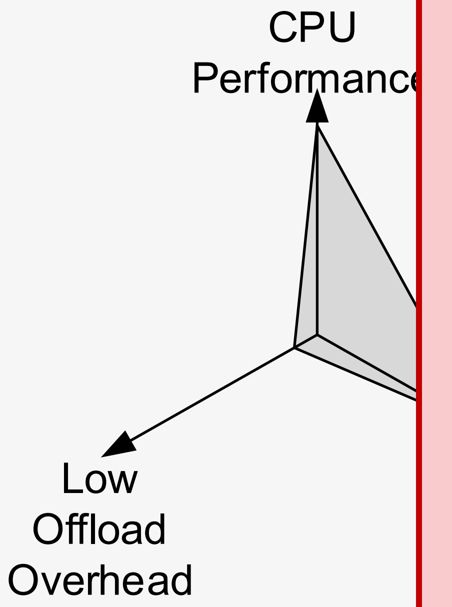


Isolated Memory Space
Switch off Interleaving

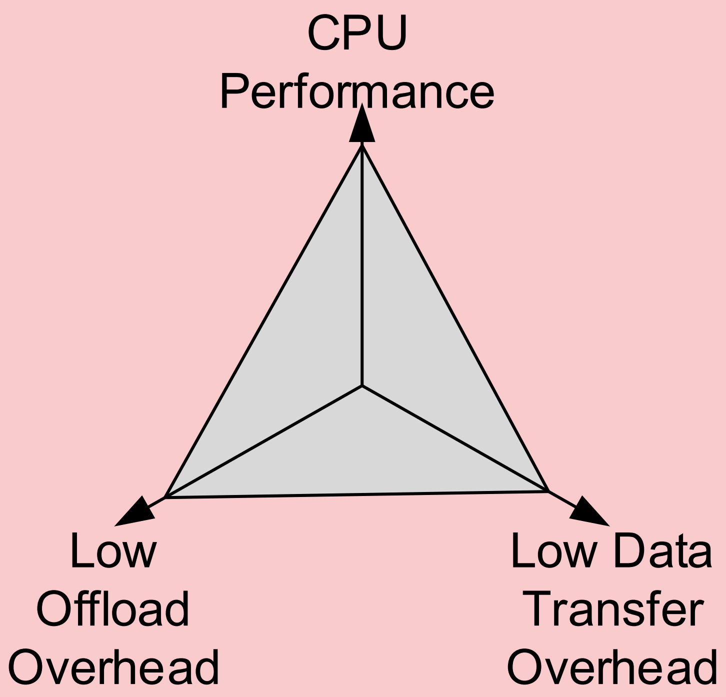
AxDIMM, UPMEM

Summary: An Impossible Triangle

- Memory spaces ~~with single data layout.~~

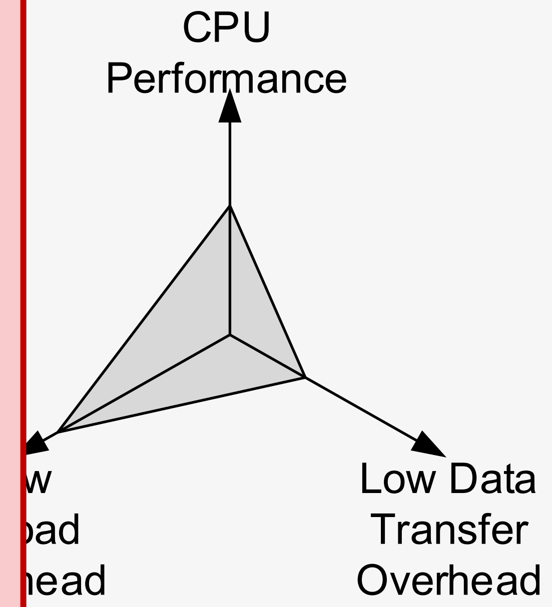


(Naïve)
Uniform Layout
Fine-grained Offload



Our goal

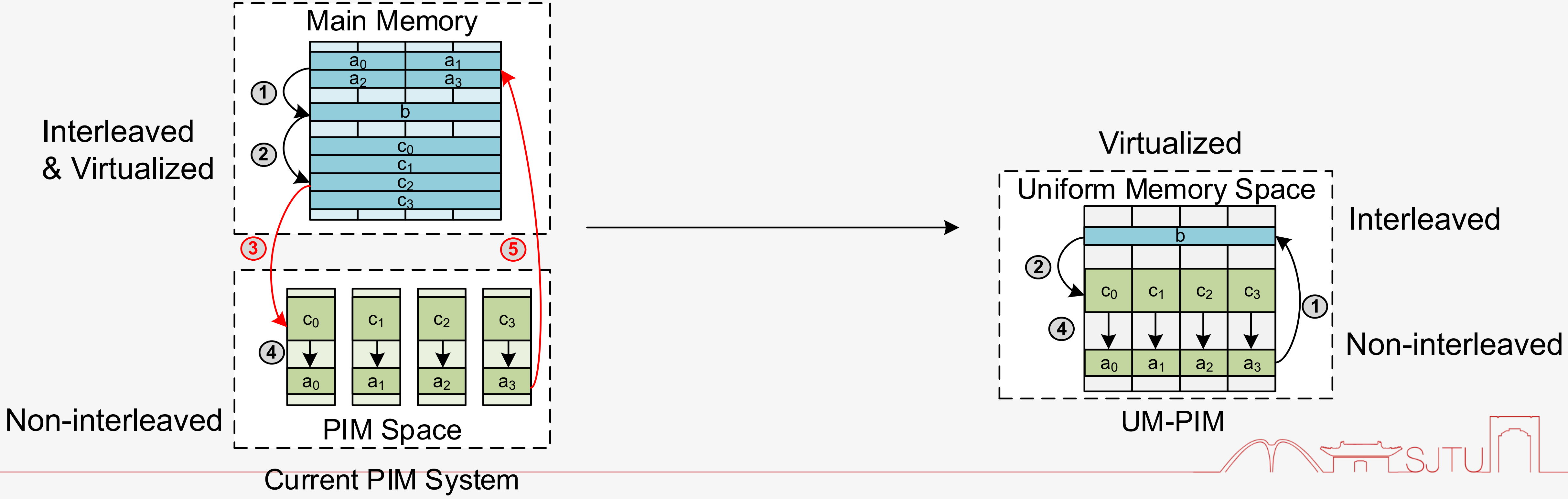
MetaPNM,PIM-HBM



Isolated Memory Space
Switch off Interleaving
AxDIMM, UPMEM

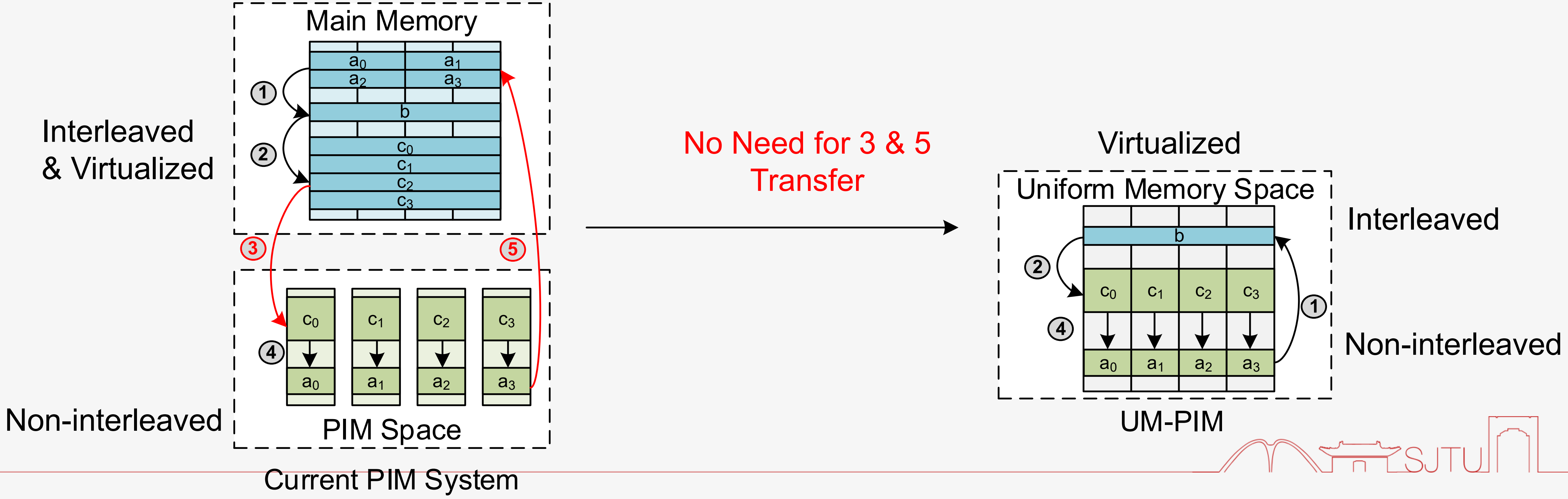
UM-PIM Motivation

- Key insight: A **Uniform Memory Space** with CPU & PIM Pages (Different layout) co-existing in the space
  - No Data Transfer Overhead
  - Satisfy CPU and PIM's data layout



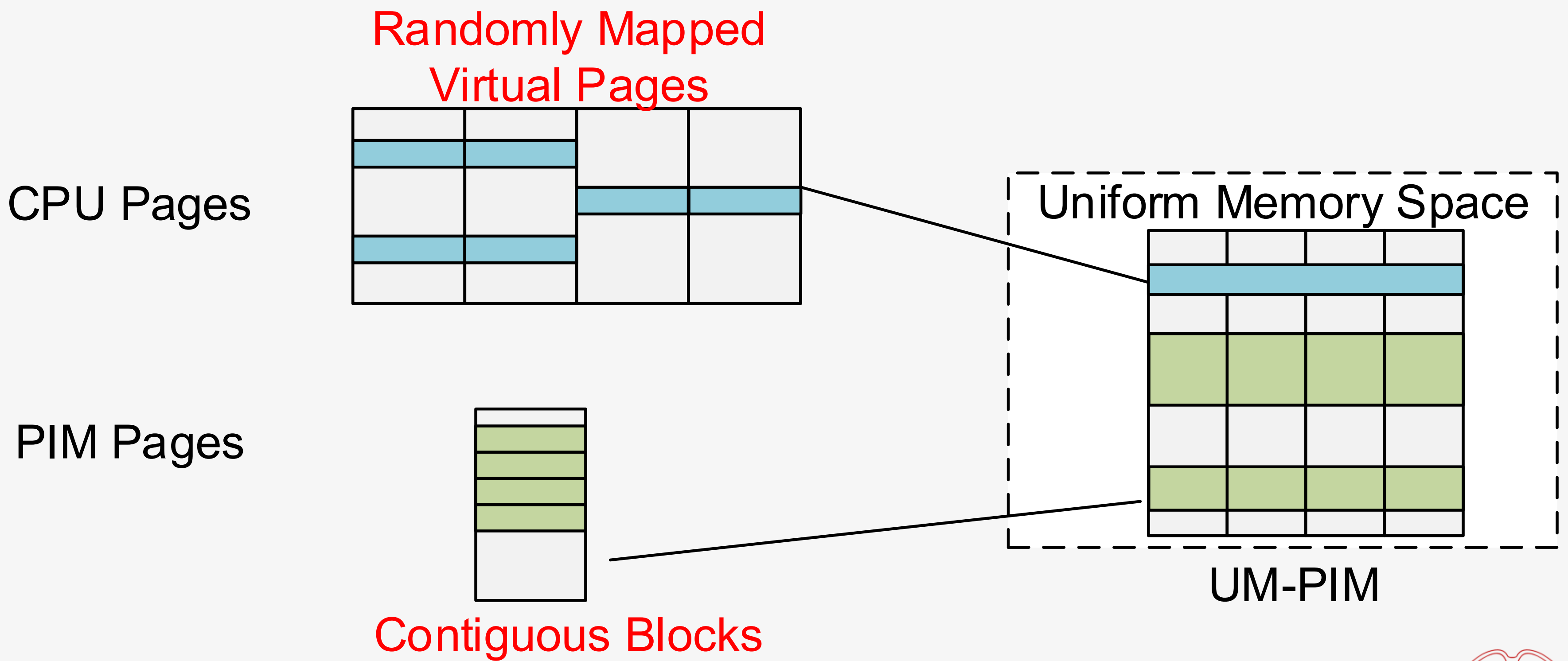
UM-PIM Motivation

- Key insight: A **Uniform Memory Space** with CPU & PIM Pages (Different layout) co-existing in the space
  - No Data Transfer Overhead
  - Satisfy CPU and PIM’s data layout



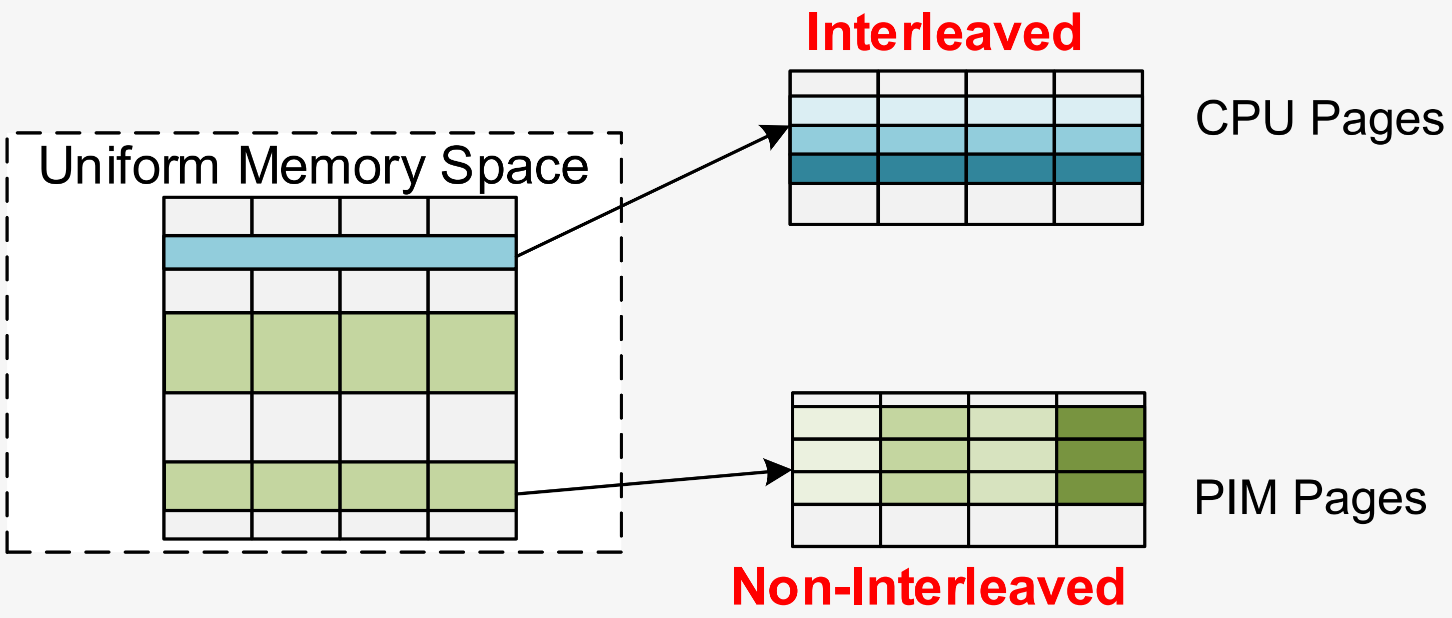
UM-PIM Motivation

- Key insight: A **Uniform Memory Space**
- Challenges:
  - **Co-existence (PIM's contiguous data block, CPU's virtual memory)**



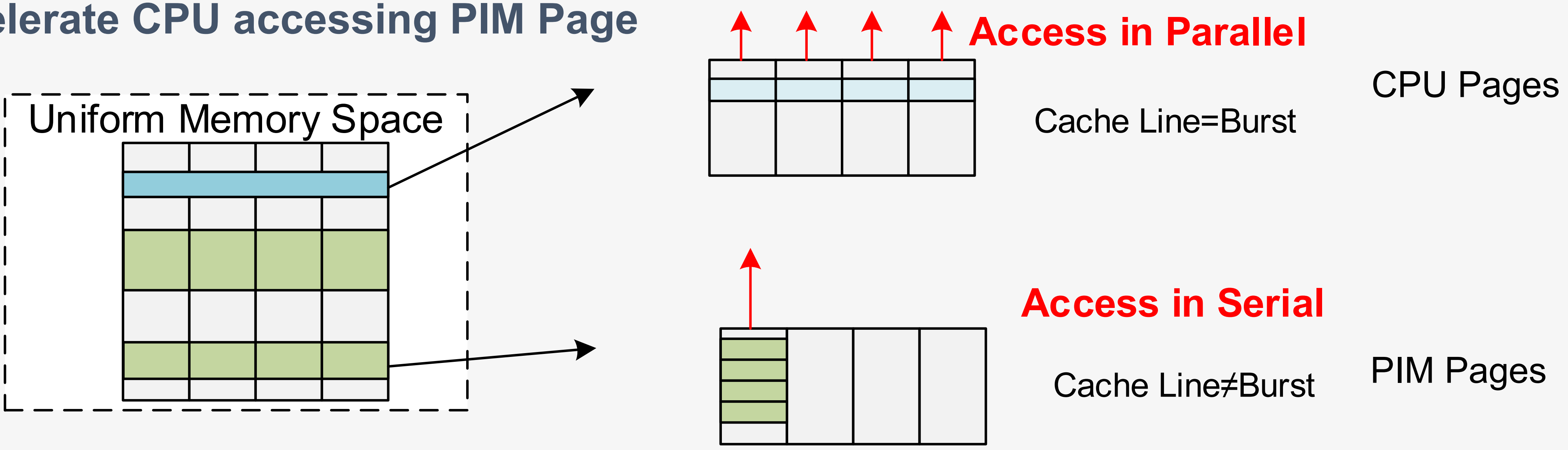
UM-PIM Motivation

- Key insight: A **Uniform Memory Space**
- Challenges:
  - Co-existence (PIM’s contiguous data block, CPU’s virtual memory)
  - **Two different data layout**



UM-PIM Motivation

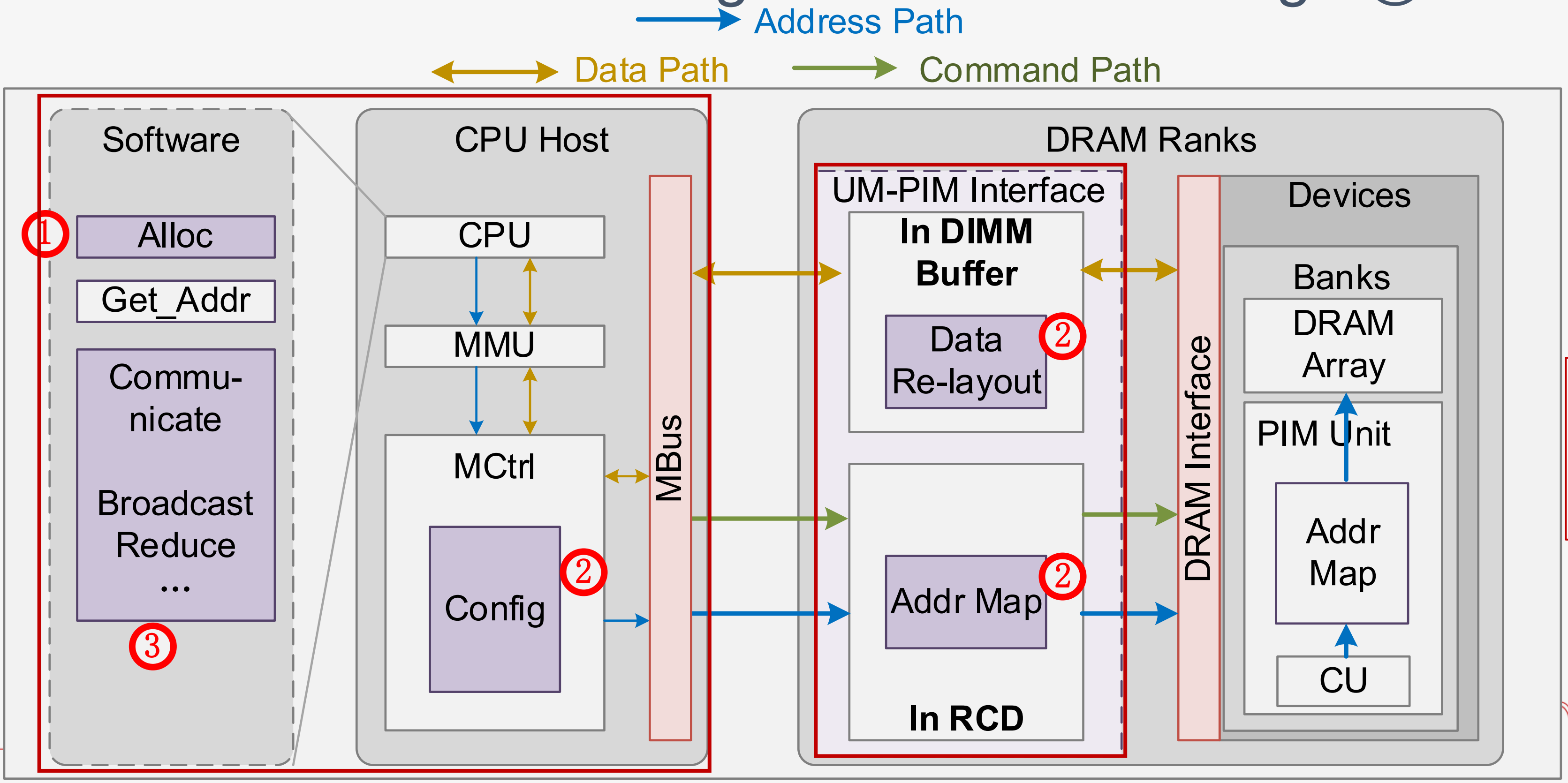
- Key insight: A **Uniform Memory Space**
- Challenges:
  - Co-existence (PIM's contiguous data block, CPU's virtual memory)
  - Two different data layout
  - **Accelerate CPU accessing PIM Page**



Outline

UM-PIM Overview

- A. Memory Management: Chunk-based management of PIM Pages ①
- B. Data Layout: Address Mapping and Data Re-layout ②
- C. Accelerate CPU Access PIM Page: HW-SW Co-design ③



Software modifications on **CPU side**

Hardware modules between MemBus & **DRAM** interface

A. Memory Management – CPU Malloc ①

Challenge 1: Co-existence of virtualized CPU Page & contiguous PIM Page

- CPU Pages: keep current management strategy
- PIM Pages: Chunk-based Management

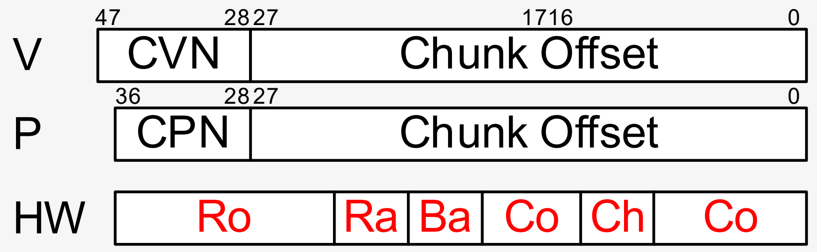
A. Memory Management – CPU Malloc ①

Challenge 1: Co-existence of virtualized CPU Page & contiguous PIM Page

PIM Pages: Chunk-based Management

- Let high-order bits mapped to rows and Allocate a **Huge Page** (Chunk)

A **huge page** 0x2..00000~0x3..00000



Switch on interleaving

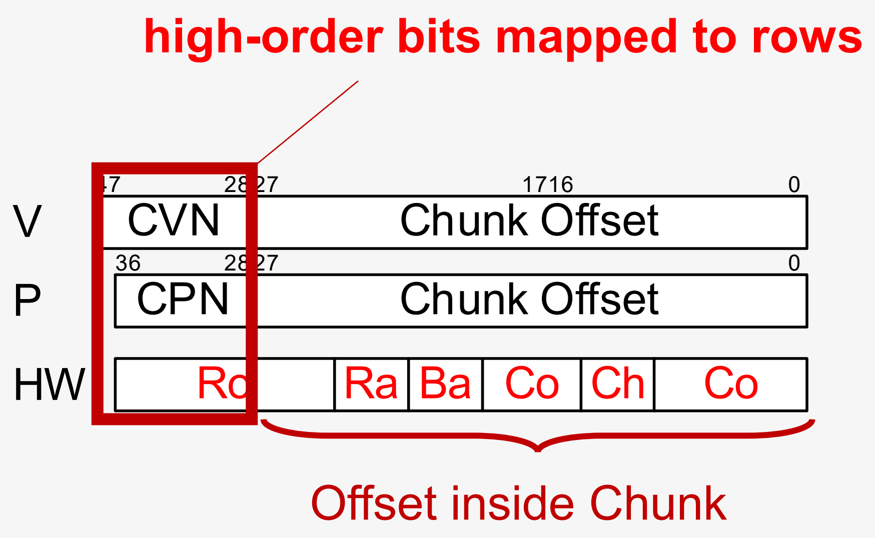
A. Memory Management – CPU Malloc ①

Challenge 1: Co-existence of virtualized CPU Page & contiguous PIM Page

PIM Pages: Chunk-based Management

- Let high-order bits mapped to rows and Allocate a **Huge Page** (Chunk)

A **huge page** 0x2..00000~0x3..00000



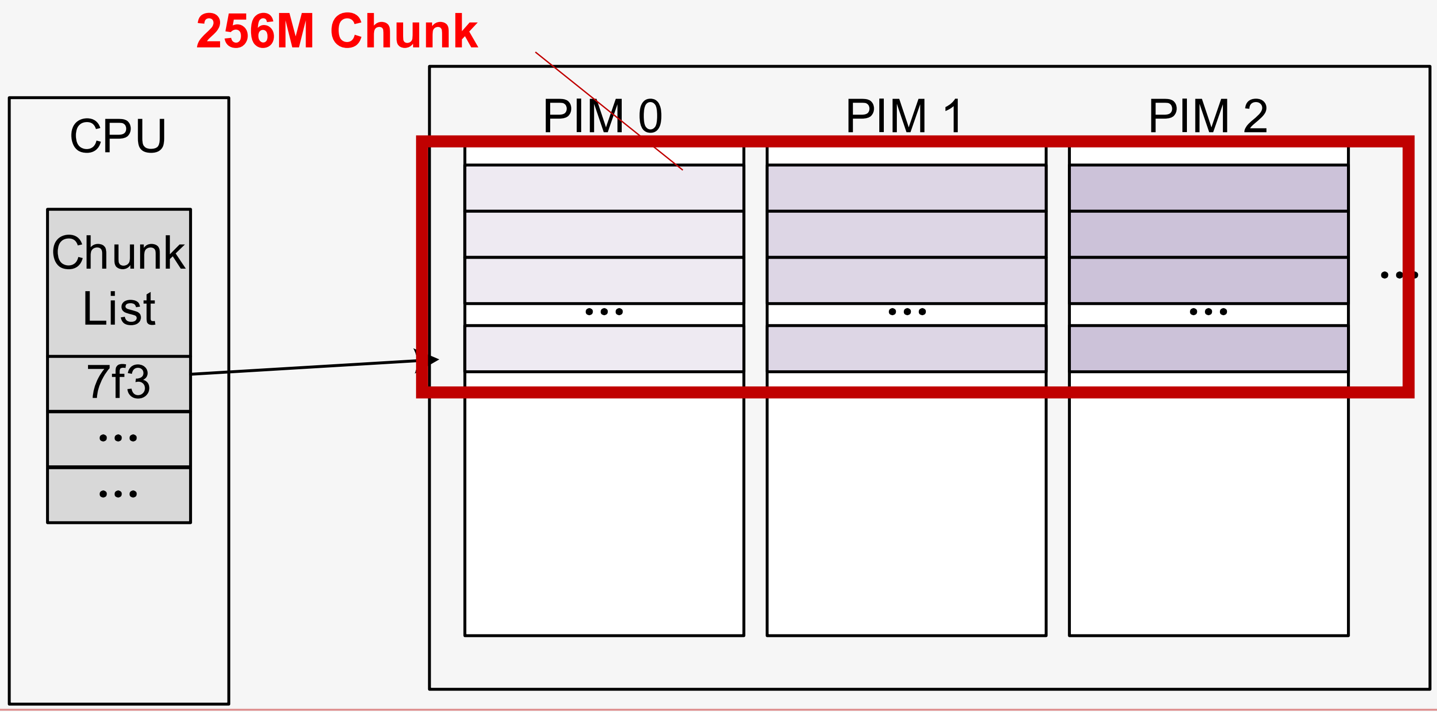
A. Memory Management – CPU Malloc ①

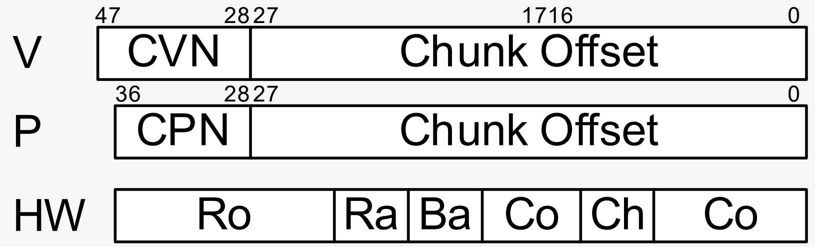
Challenge 1: Co-existence of virtualized CPU Page & contiguous PIM Page

PIM Pages: Chunk-based Management

- Let high-order bits mapped to rows and Allocate a Huge Page (Chunk):
  - The Chunk Evenly distributed on every PIMs

A huge page 0x2..00000~0x3..00000





A. Memory Management – CPU Malloc ①

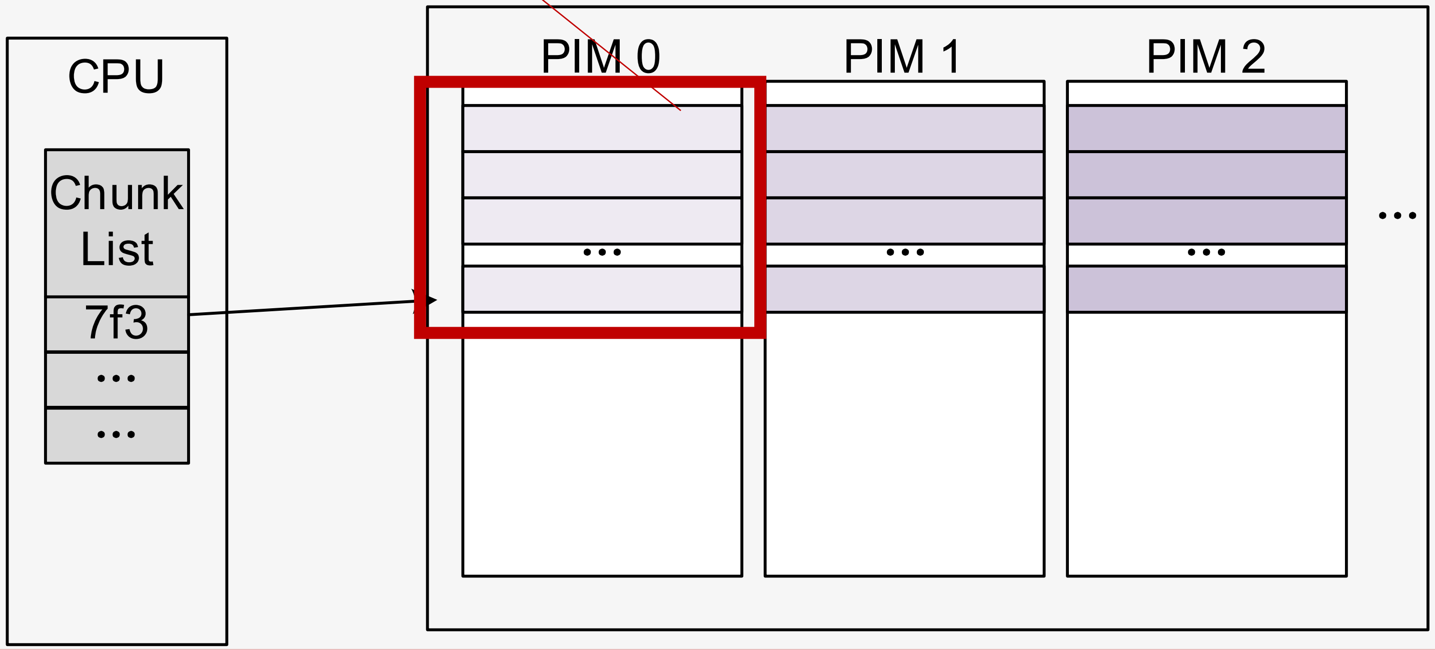
Challenge 1: Co-existence of virtualized CPU Page & contiguous PIM Page

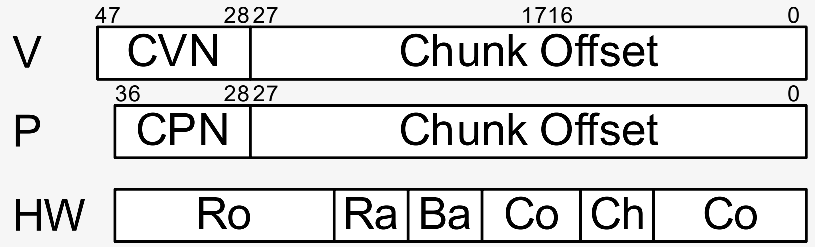
PIM Pages: Chunk-based Management

- Let high-order bits mapped to rows and Allocate a Huge Page (Chunk):
  - The Chunk Evenly distributed on every PIMs

A huge page 0x2..00000~0x3..00000

256M Chunk = 128 k PIM Page on each PIM





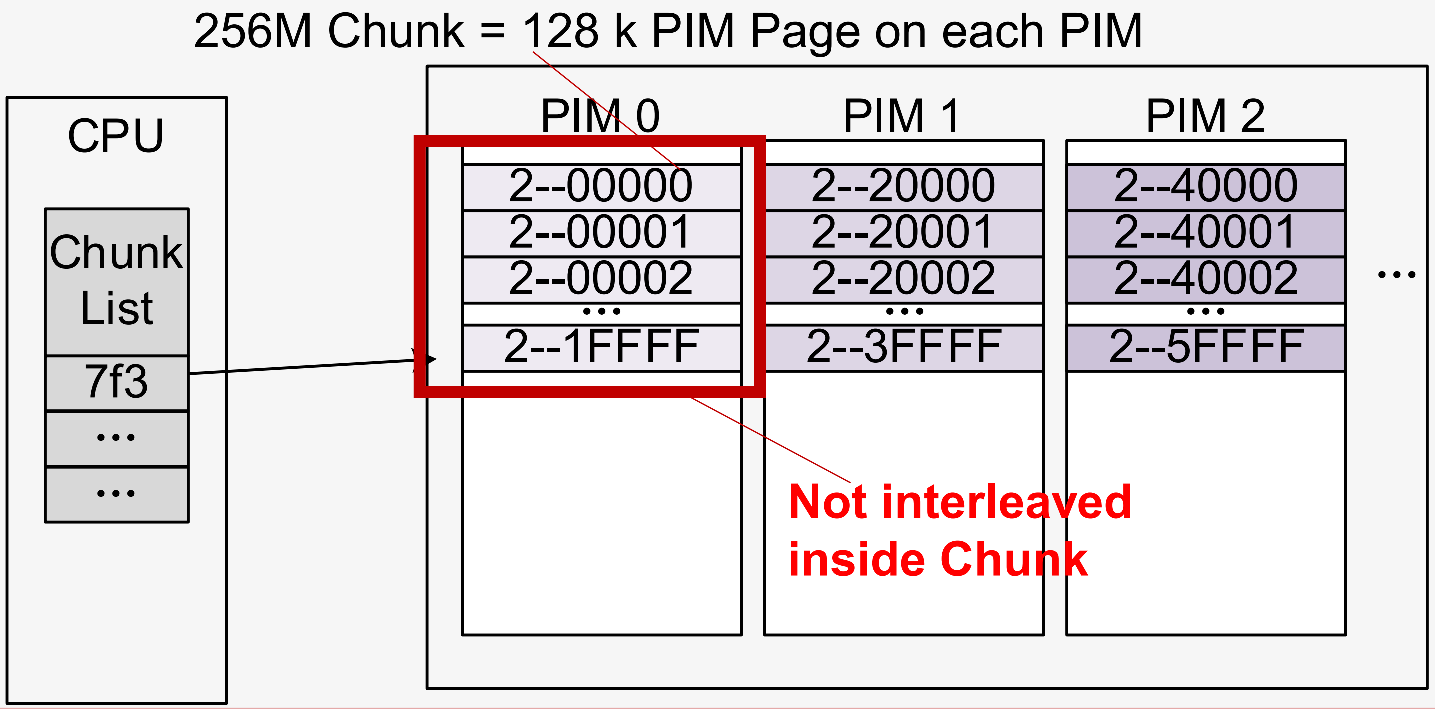
A. Memory Management – CPU Malloc ①

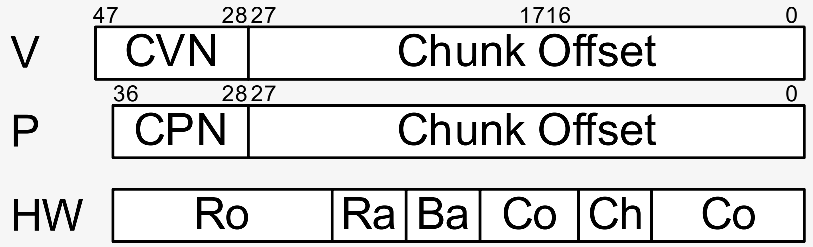
Challenge 1: Co-existence of virtualized CPU Page & contiguous PIM Page

PIM Pages: Chunk-based Management

- Let high-order bits mapped to rows and Allocate a Huge Page (Chunk):
  - The Chunk Evenly distributed on every PIMs

A huge page 0x2..00000~0x3..00000

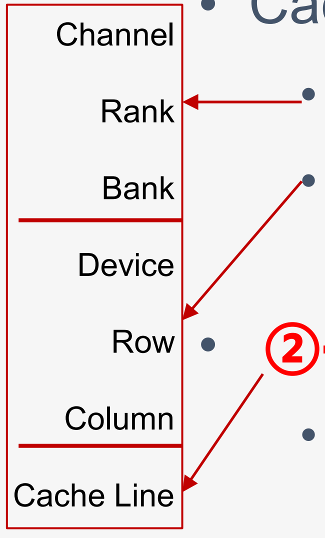




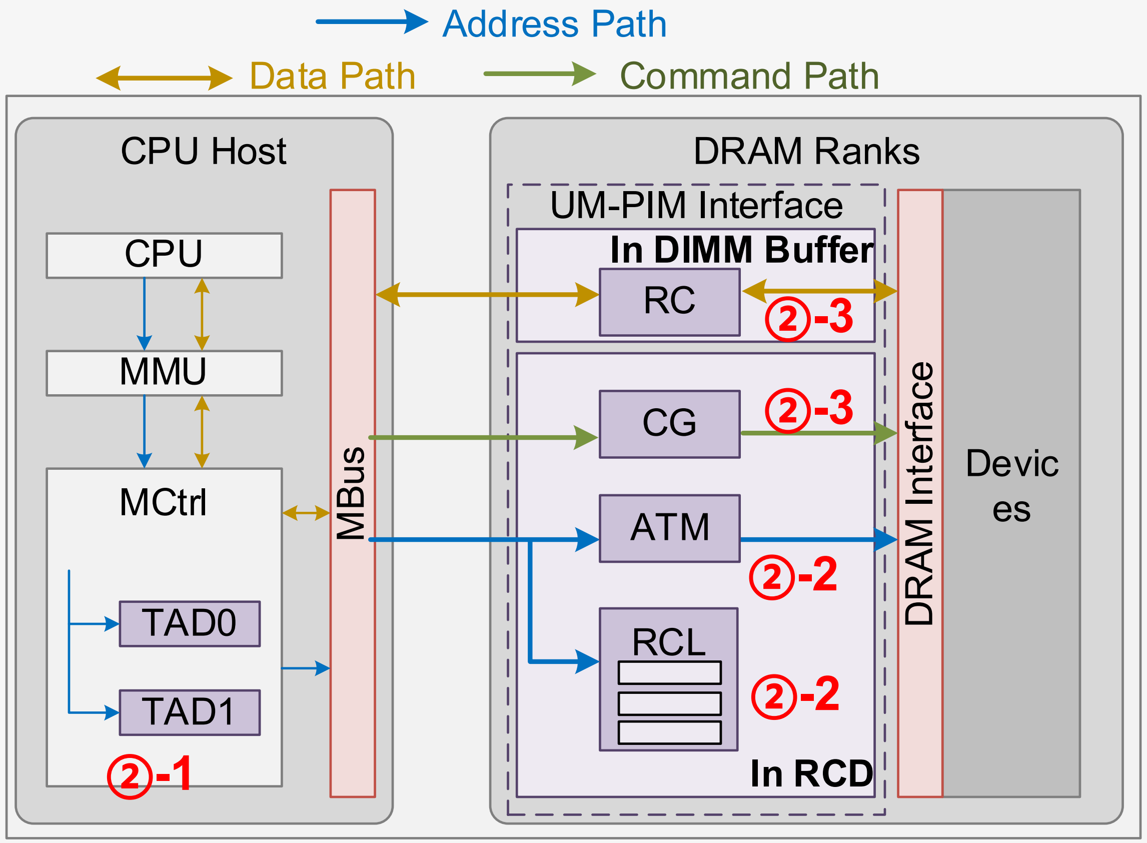
B Data Layout

Challenge 2: Different layout of CPU&PIM Pages

- Integrate Hardware Modules between DRAM Interface & Memory Bus, on DRAM side
- 3 Levels:
  - Cache Line: Dynamic Address Mapping
    - ②-1 Above Bank: Config Mem Ctrl
    - ②-2 Below Bank: DRAM-side HW (ATM &RCL)
  - ②-3 Inside Cache Line: Data re-layout
    - DRAM-side HW (RC&CG)



DRAM Hierarchy



B Data Layout — Address Mapping ②-1

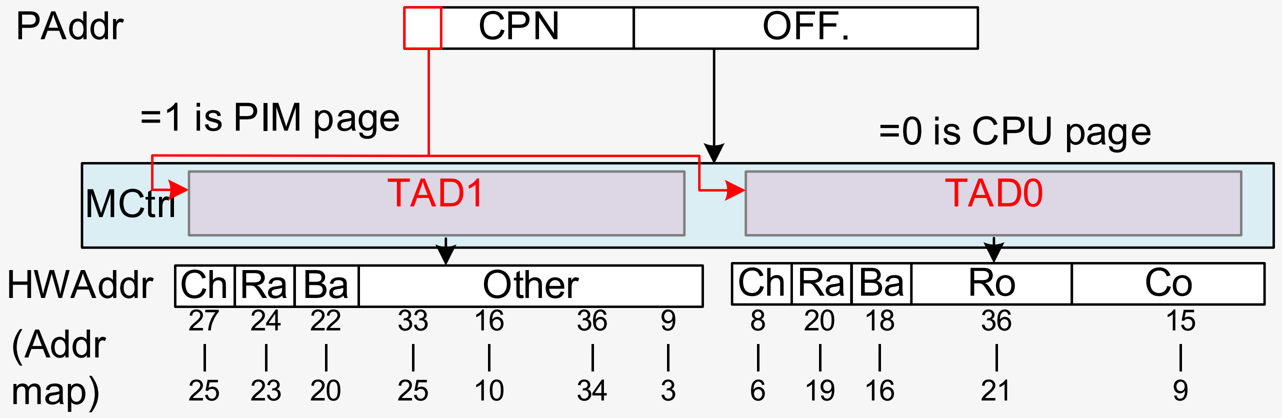
- **(Above Bank)** Add a Address Alias (TAD 1) in MemCtrl
  - TAD1 have different address mapping from the origin TAD0

Channel
Rank
Bank

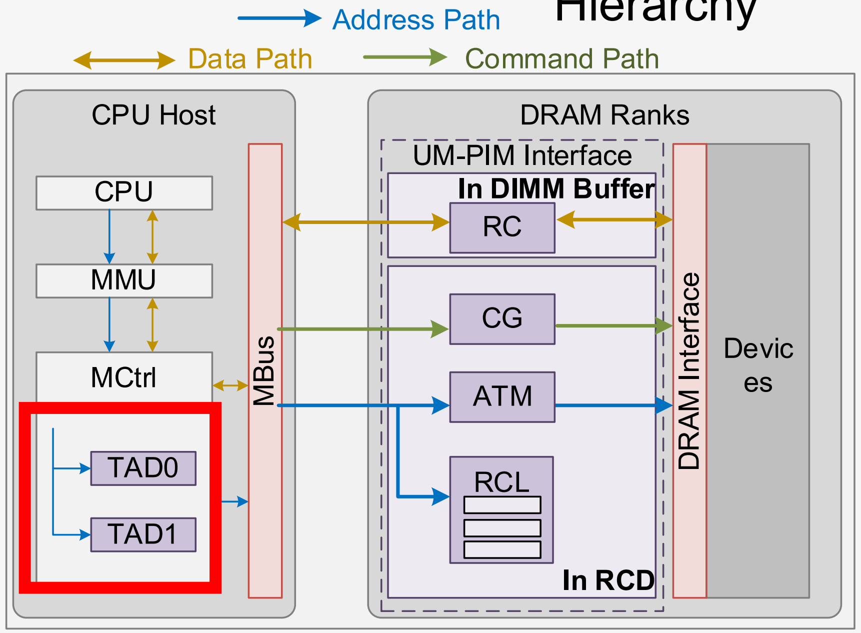
Device
Row
Column

Cache Line

DRAM Hierarchy



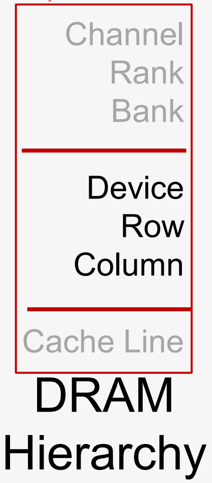
TAD0 & TAD1 have different address mapping

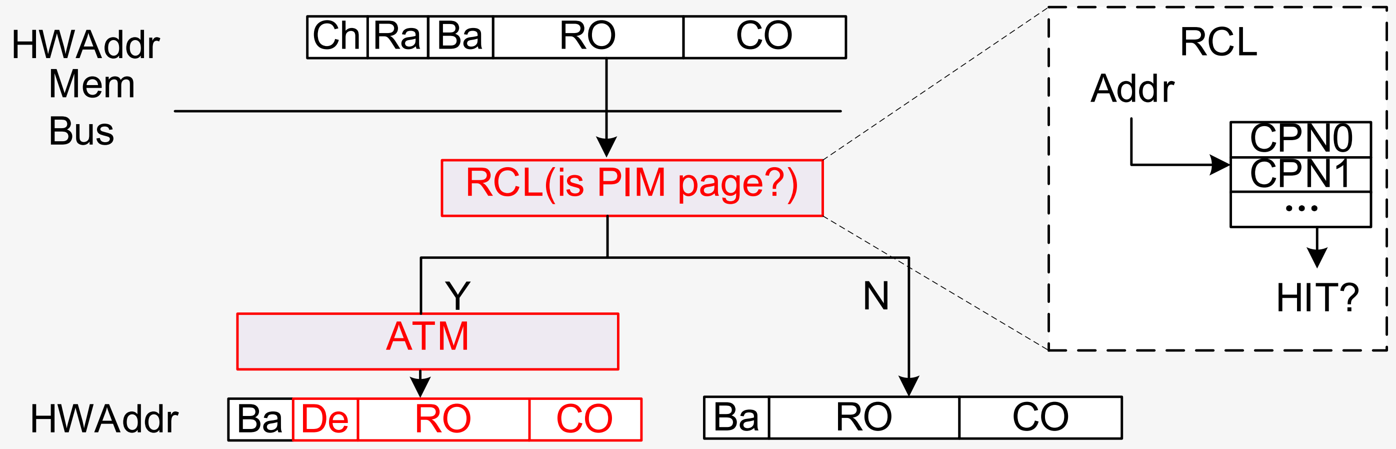


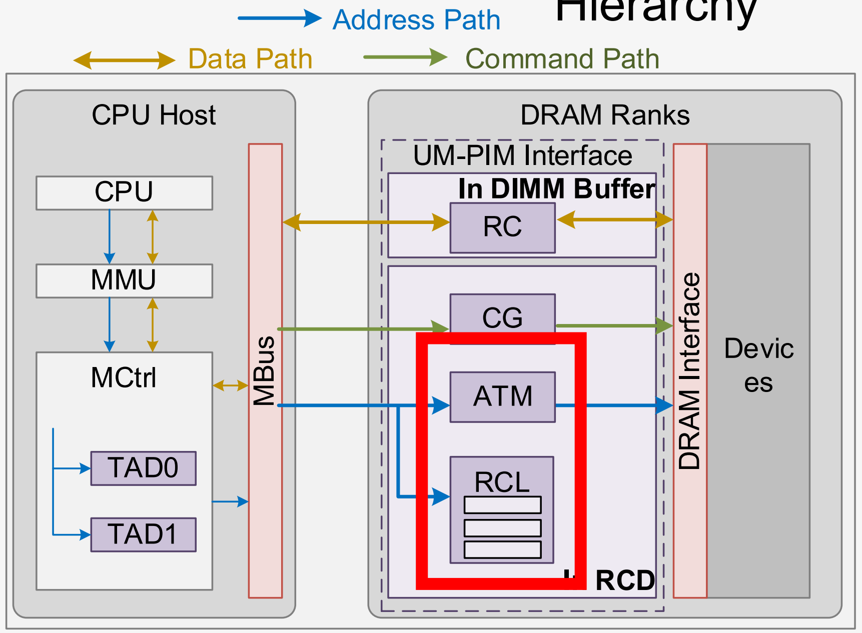
B Data Layout — Address Mapping ②-2

(Below Bank) Hardware module on DRAM side

- RCL: Check whether the Address belongs to PIM Page
- ATM: If is PIM address, map the address bits to Device Column and Row



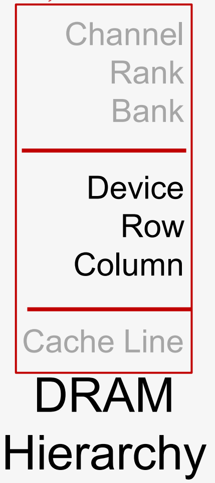




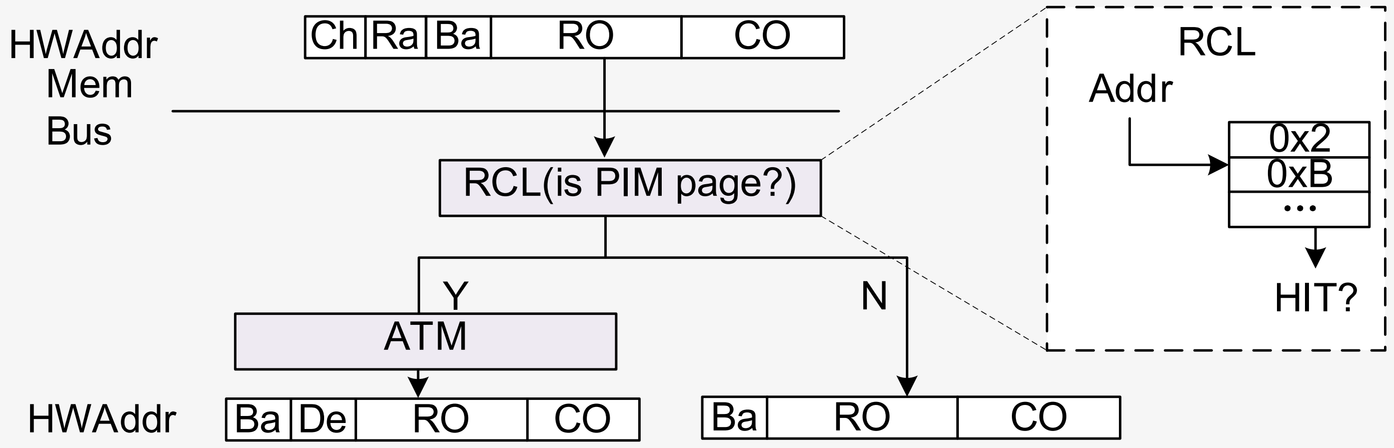
B Data Layout — Address Mapping ②-2

(Below Bank) Hardware module on DRAM side

- RCL: Check whether the Address belongs to PIM Page
- ATM: If is PIM address, map the address bits to Device Column and Row



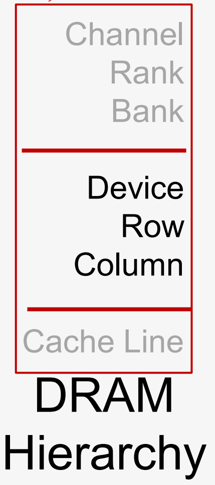
Address: 0xB0020001

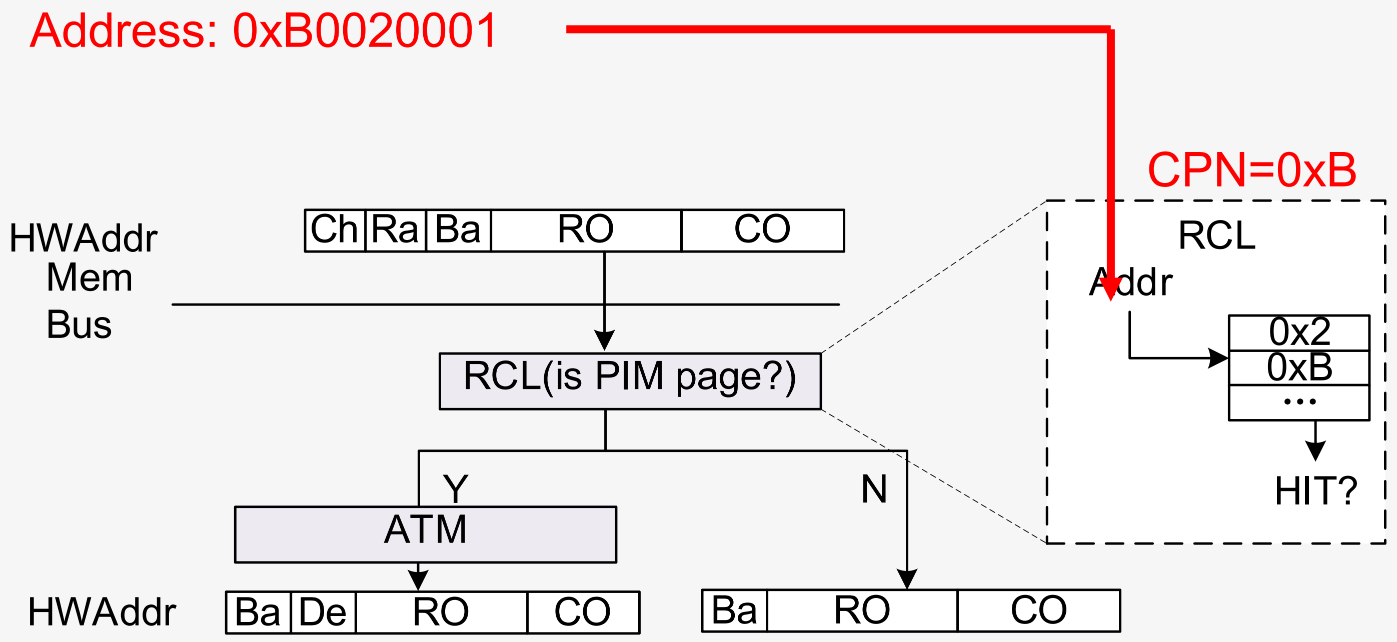


B Data Layout — Address Mapping ②-2

(Below Bank) Hardware module on DRAM side

- RCL: Check whether the Address belongs to PIM Page
- ATM: If is PIM address, map the address bits to Device Column and Row

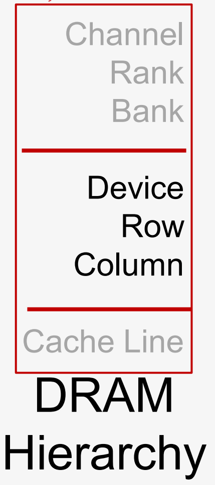


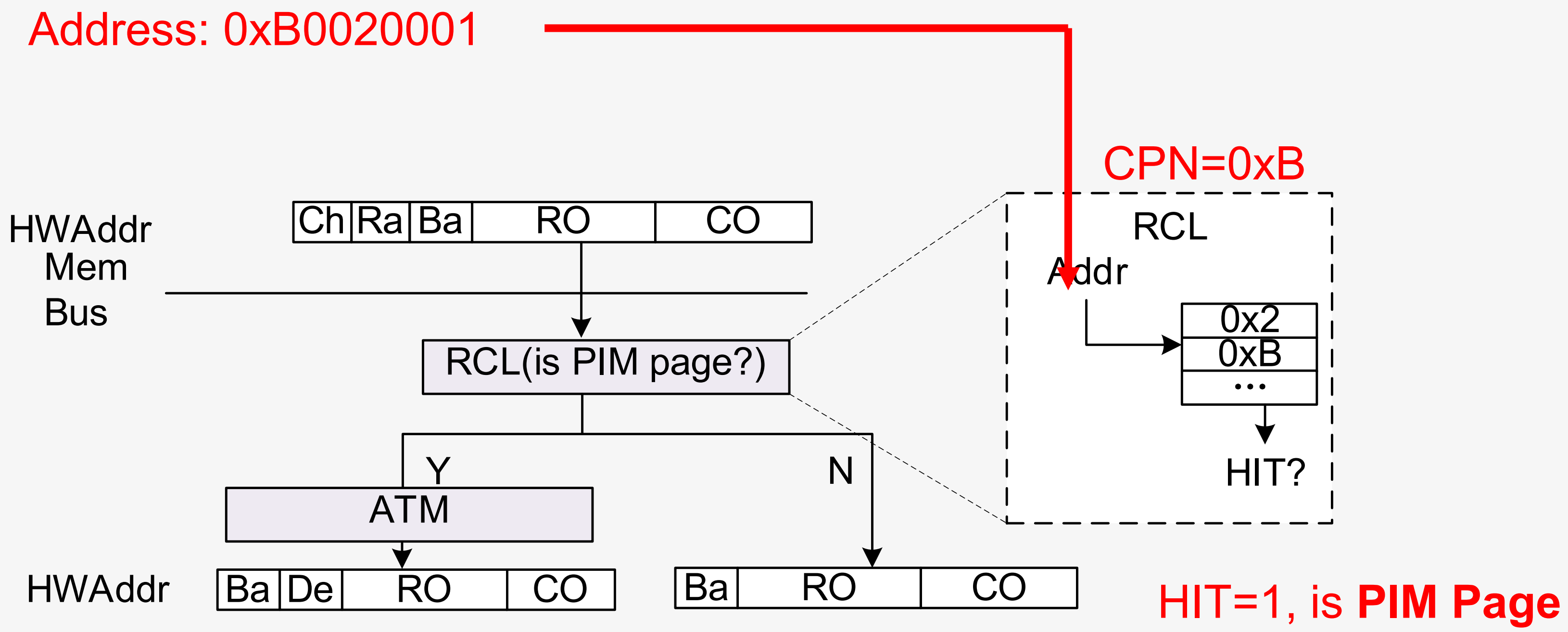


B Data Layout — Address Mapping ②-2

(Below Bank) Hardware module on DRAM side

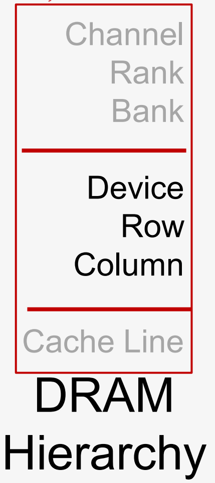
- RCL: Check whether the Address belongs to PIM Page
- ATM: If is PIM address, map the address bits to Device Column and Row



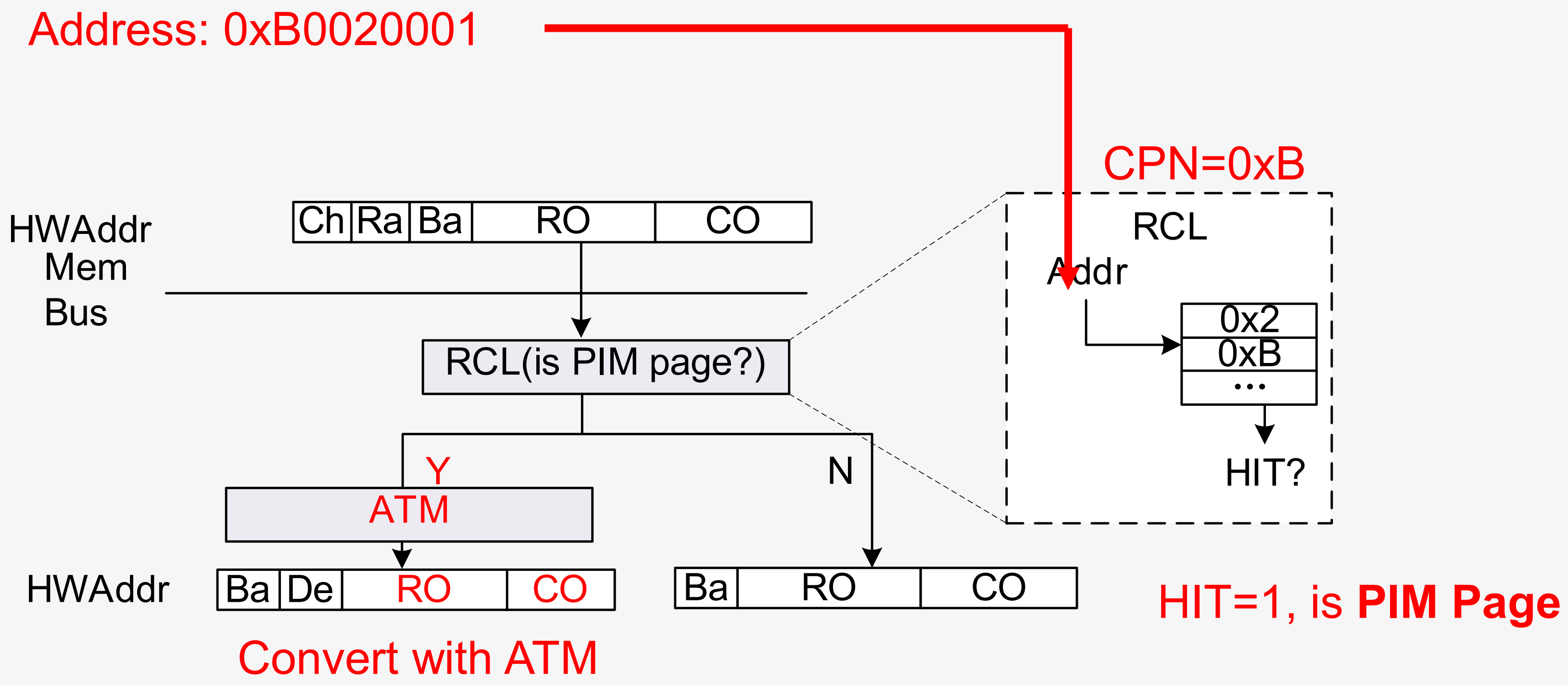


B Data Layout — Address Mapping ②-2

- **(Below Bank)** Hardware module on DRAM side
  - RCL: Check whether the Address belongs to PIM Page
  - ATM: If is PIM address, map the address bits to Device Column and Row



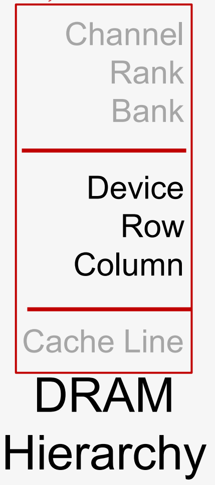
DRAM Hierarchy

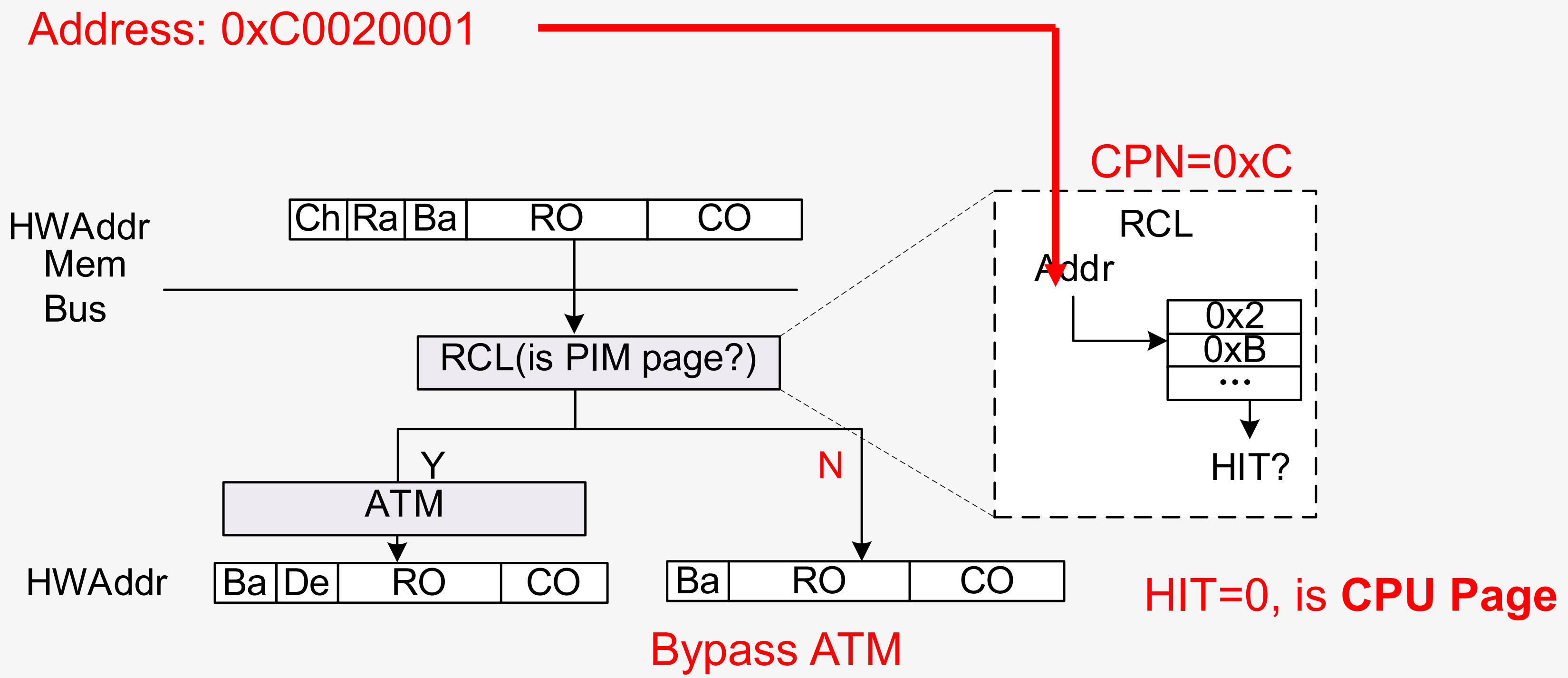


B Data Layout — Address Mapping ②-2

(Below Bank) Hardware module on DRAM side

- RCL: Check whether the Address belongs to PIM Page
- ATM: If is PIM address, map the address bits to Device Column and Row





B Data Layout — Re-layout ②-3

Channel
Rank
Bank

Device
Row
Column

Cache Line

DRAM
Hierarchy

(Inside Cache Line) Cache Line ≠ Burst problem of PIM Page

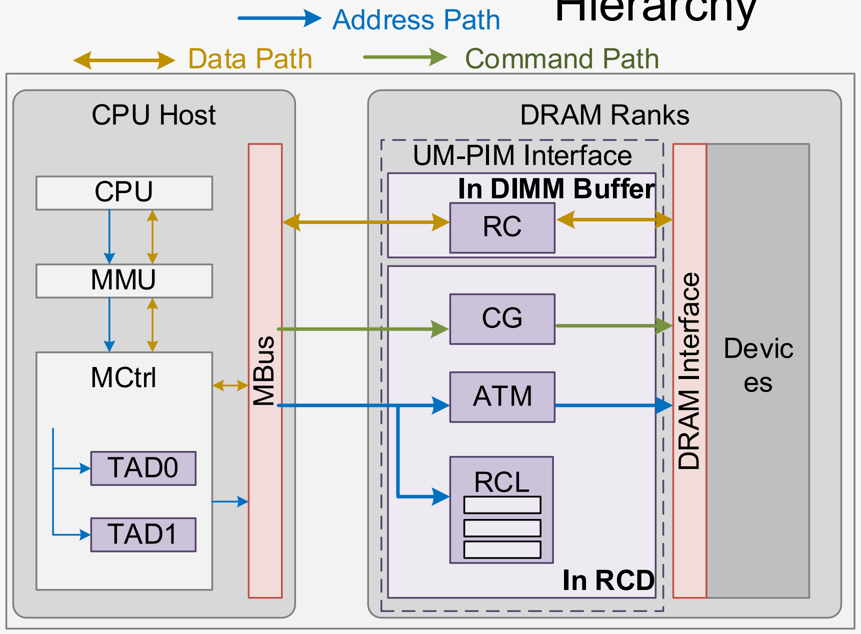
PIM Friendly
Data Localization
A Cache Line of PIM Page

CPU Friendly
High Bandwidth
Decided by Hardware
A DRAM Burst

D0	D1	D2	D3
0x00	0x04	0x08	0x0C
0x01	0x05	0x09	0x0D
0x02	0x06	0x0A	0x0E
0x03	0x07	0x0B	0x0F

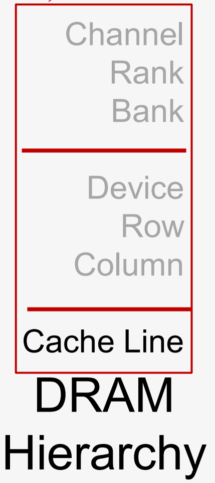
…

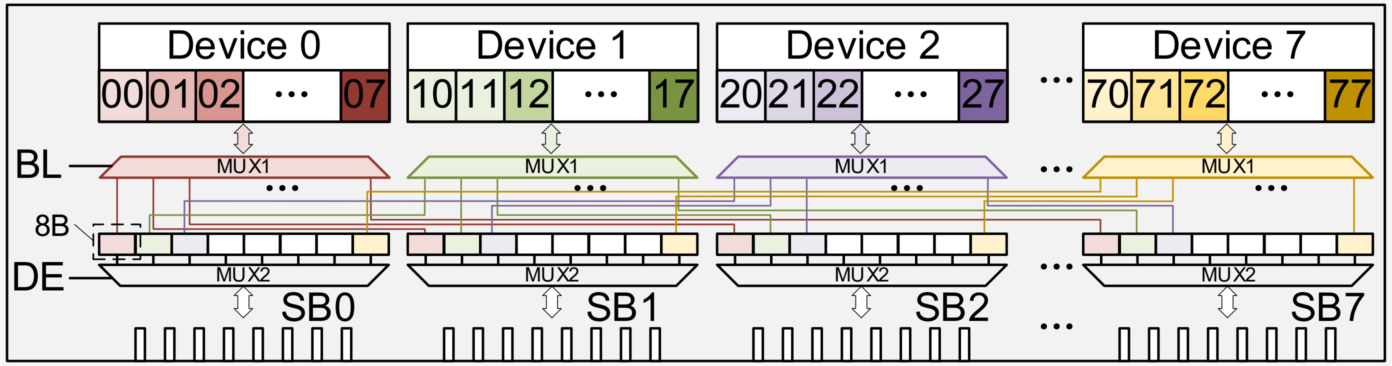
D： Device



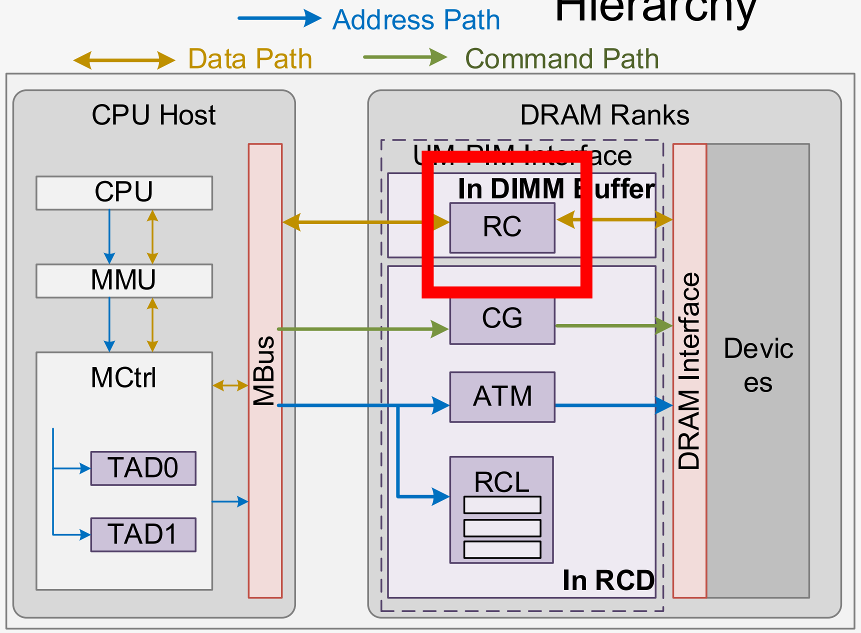
B Data Layout — Re-layout ②-3

- **(Inside Cache Line)** Cache Line ≠ Burst problem of PIM Page
- A Hardware module **RC** for data re-layout inside Cache Line



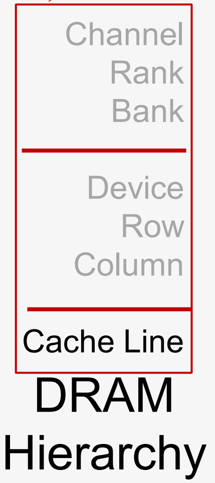


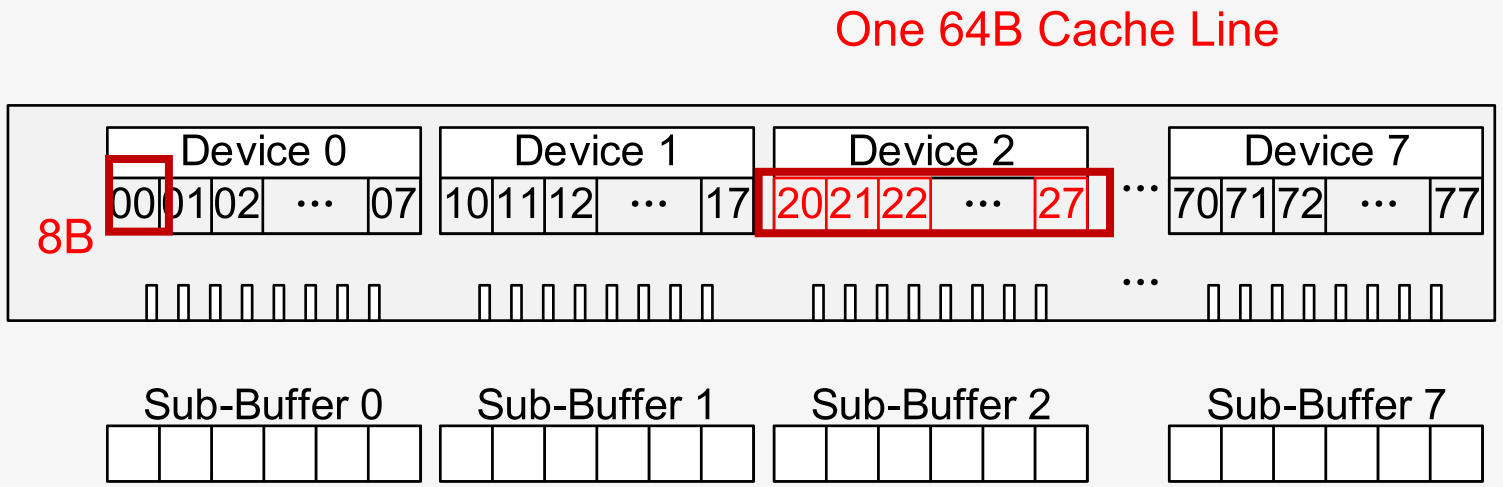
Re-layout cache (RC)



B Data Layout — Re-layout ②-3

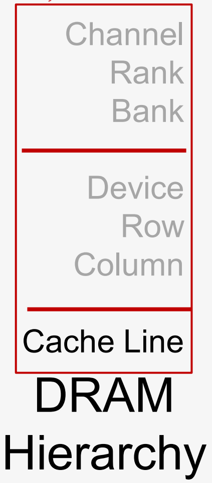
- **(Inside Cache Line)**
- Read a 64B Cache Line of PIM Page in Device 2

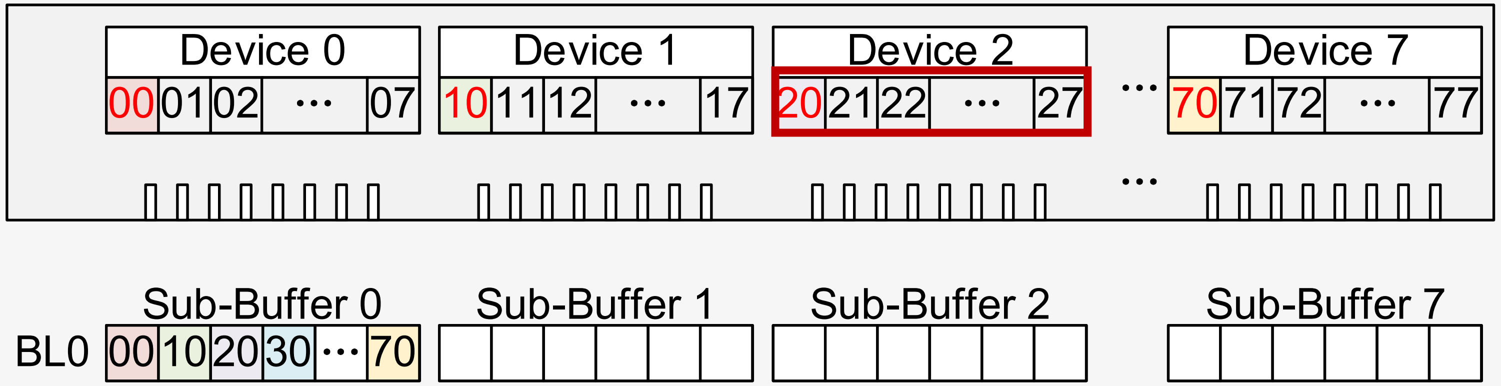




B Data Layout — Re-layout ②-3

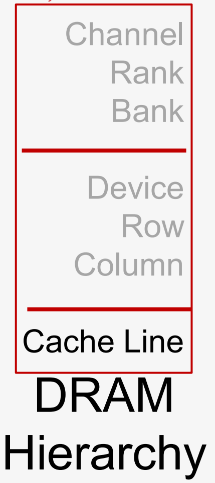
- **(Inside Cache Line)**
- Read a 64B Cache Line of PIM Page in Device 2
  - Cycle 0-7: Read 8 Burst form every device, cache the Bursts in RC

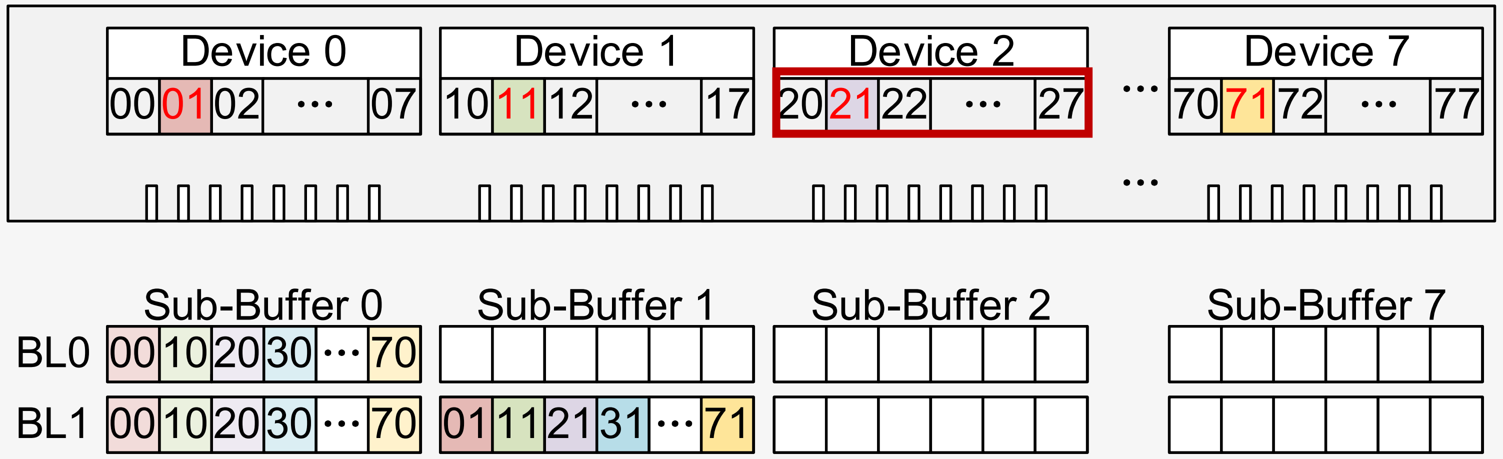




B Data Layout — Re-layout ②-3

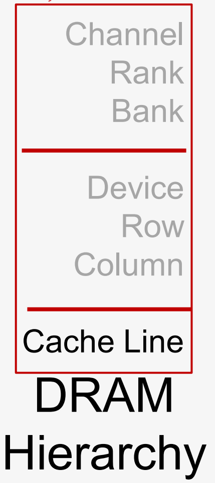
- **(Inside Cache Line)**
- Read a 64B Cache Line of PIM Page in Device 2
  - Cycle 0-7: Read 8 Burst form every device, cache the Bursts in RC

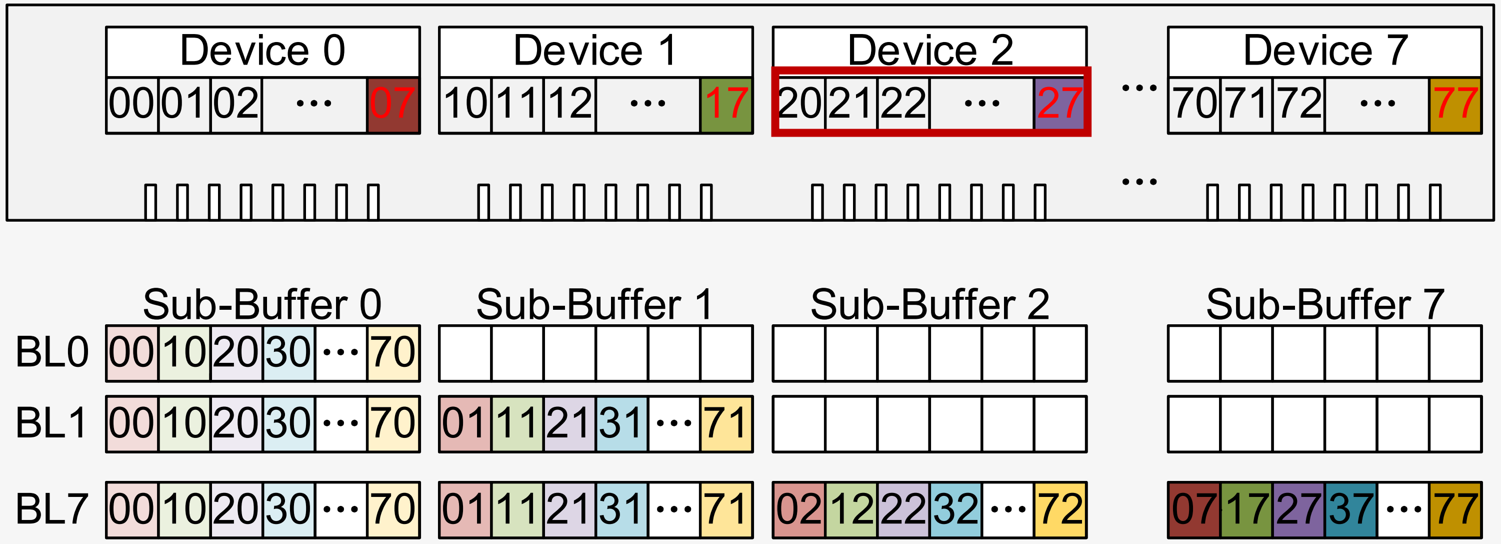




B Data Layout — Re-layout ②-3

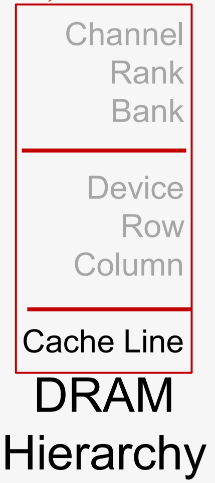
- (Inside Cache Line)
- Read a 64B Cache Line of PIM Page in Device 2
  - Cycle 0-7: Read 8 Burst form every device, cache the Bursts in RC

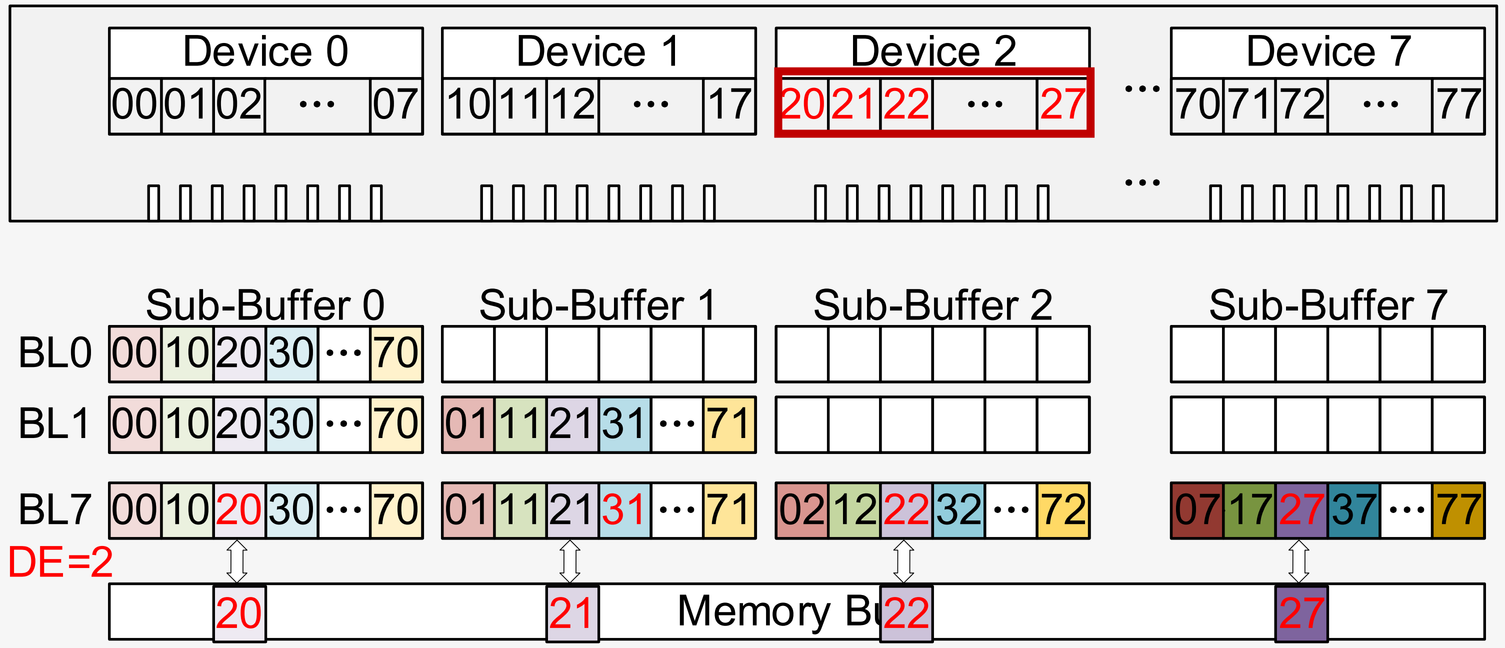




B Data Layout — Re-layout ②-3

- **(Inside Cache Line)**
- Read a 64B Cache Line of PIM Page in Device 2
  - Cycle 0-7: Read 8 Burst form every device, cache the Bursts in RC
  - Cycle 8: Get the data from Device 2

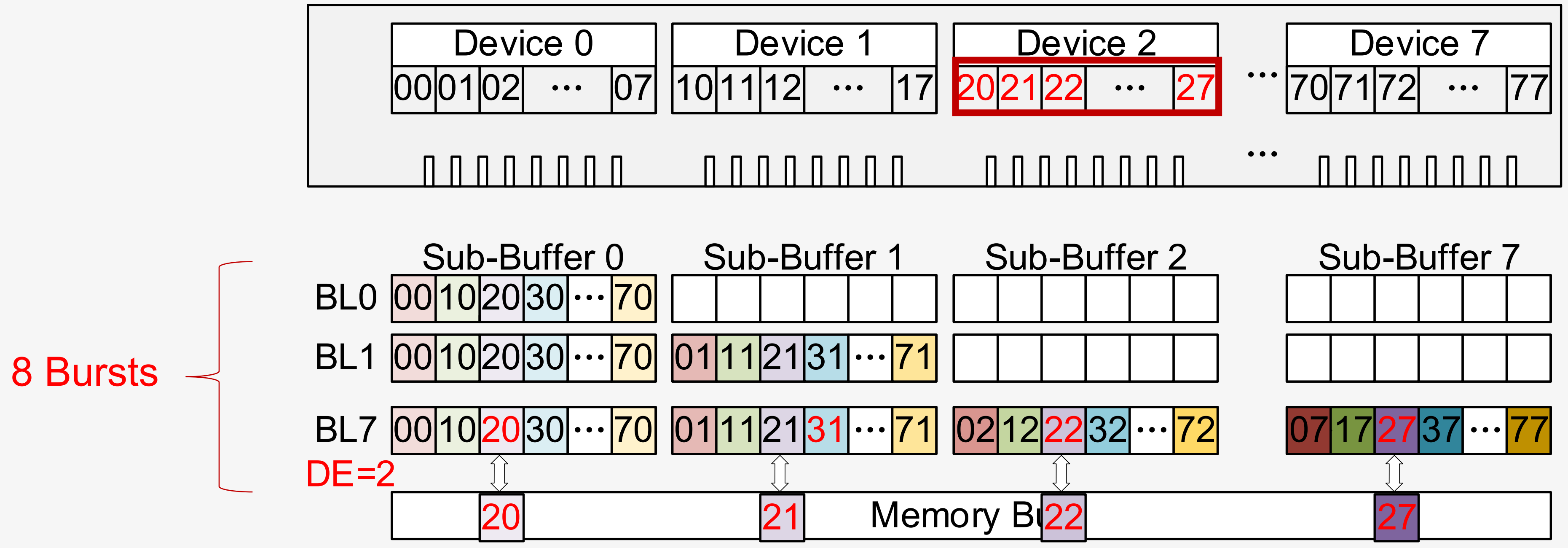




C Accelerate CPU Access PIM Page ③

- Require 8 bursts to read one cache line of PIM Page

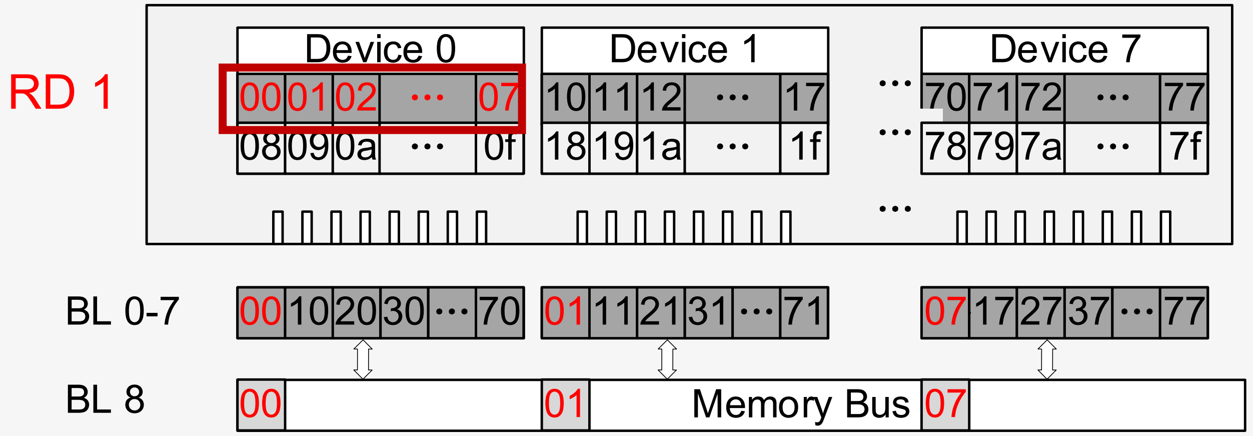
Challenge 3: Improve CPU bandwidth when accessing PIM page



C Accelerate CPU Access PIM Page ③

- When CPU iterate over results of PIM units:

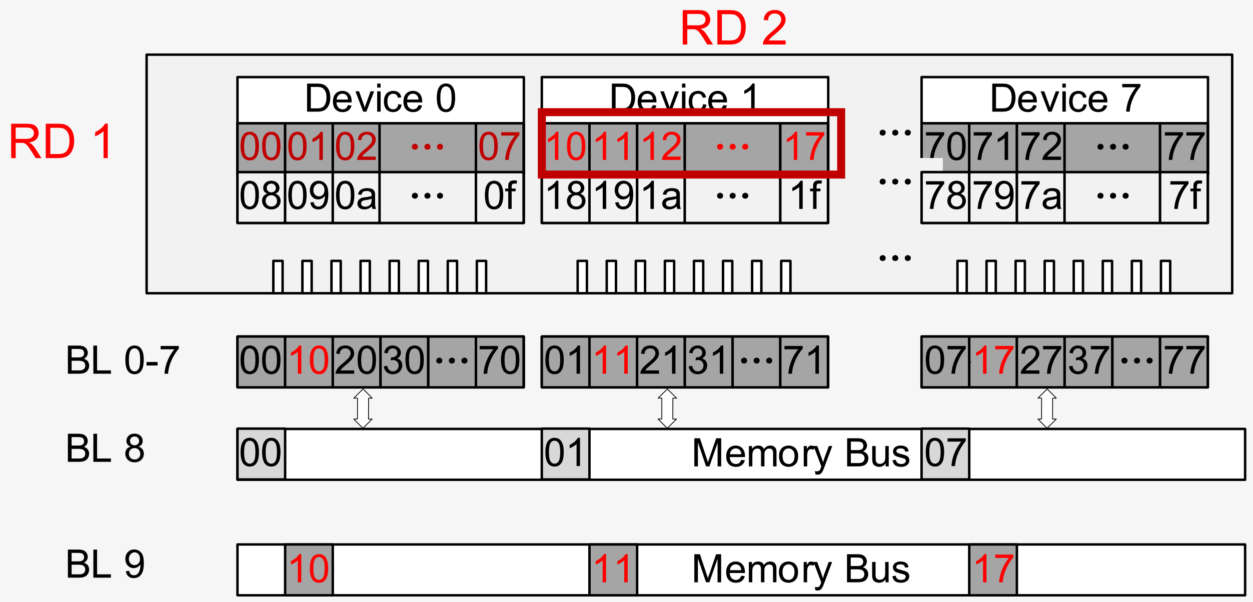
```
for de in range (#device):
    for l in range (len):
        access(de, l)
```



C Accelerate CPU Access PIM Page ③

- When CPU iterate over results of PIM units:

```
for de in range (#device):
    for l in range (len):
        access(de, l)
```

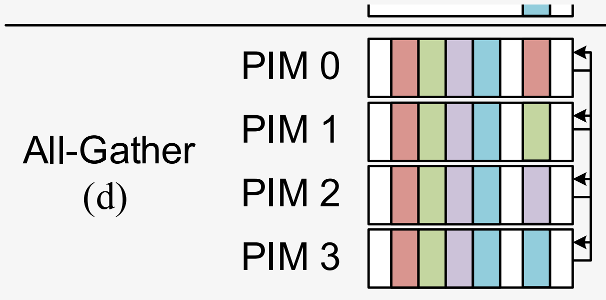


C Accelerate CPU Access PIM Page ③

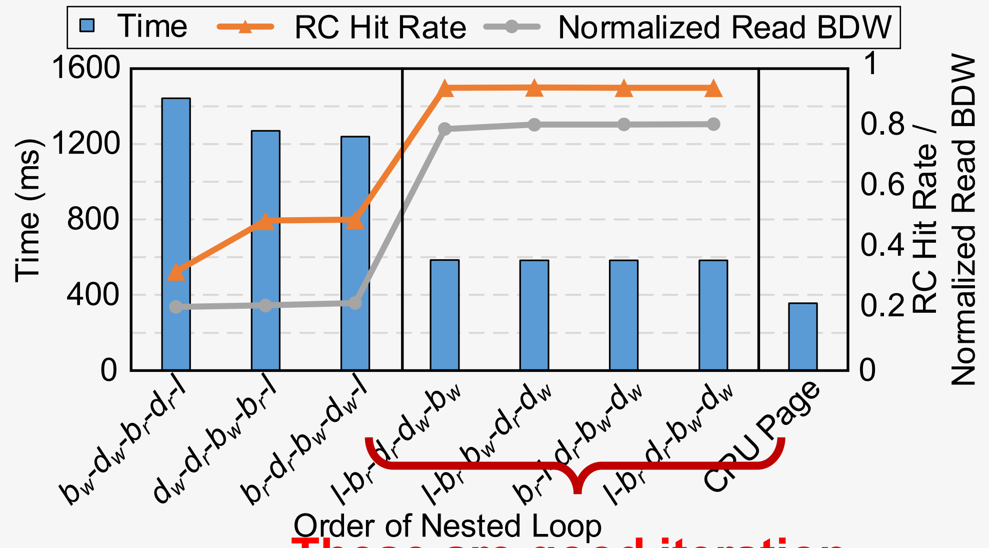
SHANGHAI JIAO TONG UNIVERSITY

When CPU iterate over results of PIM units:

- Let iteration on devices be the inner loop
  - *dw*: device address for writing



```
for l in range(len/64):
 for br in range(#bank):
  for dr in range(8):
   for bw in range(#bank):
    for dw in range(8):
      memcpy(dest[bw, dw, (br*8+dr)*s+l],
             src[br, dr, l], 64)
```



These are good iteration orders

SJTU

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Outline

Evaluation Methodology

Simulator & Configuration

- CPU: GEM5 + Ramulator2; PIM units: UPMEM DPU @ 500MHz
- DDR4-2400, 8(Channel)x 8(Ranks)

Benchmarks:

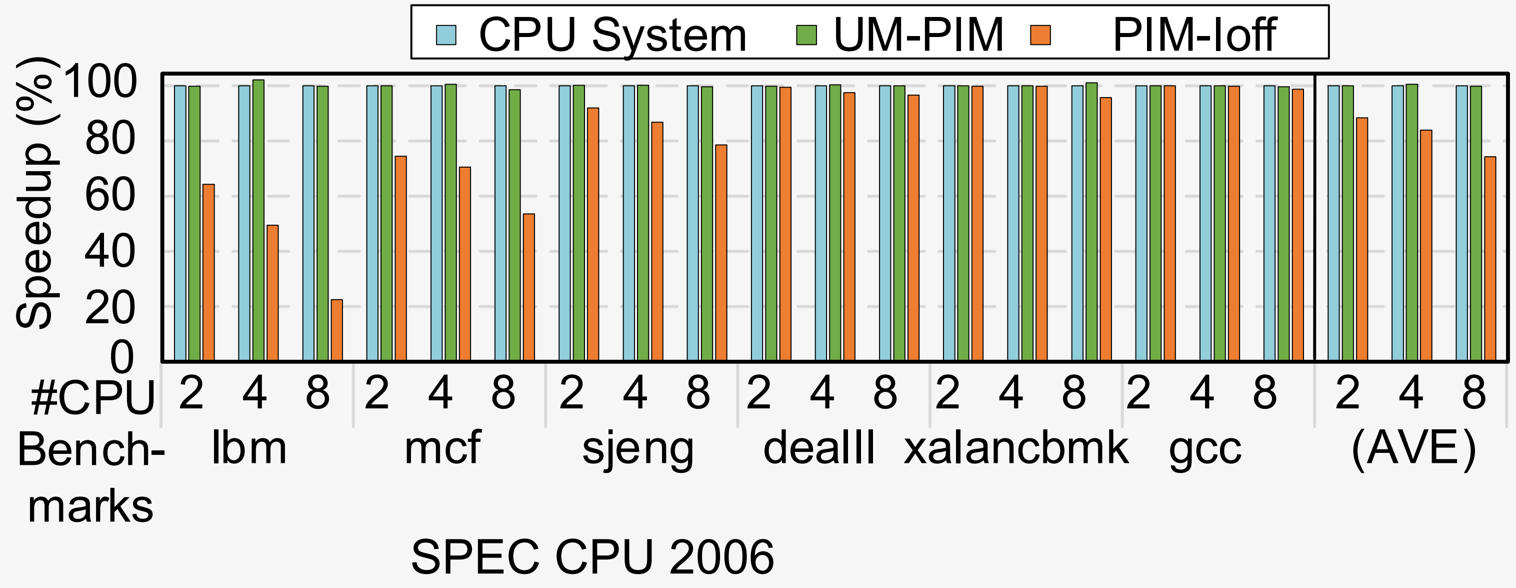
- CPU Workloads: from SPEC CPU 2006
  - lbm, mcf, sjeng, dealII, xalancbmk, gcc
- PIM Workloads (requires CPU & PIM): from PRIM
  - BFS, PR, MLP, UNI, TC, SCAN-RSS/SSA, SEL, HST, NW, WFA, RL

Baseline:

- PIM-Ion: PIM System with Memory interleaving switched on
- PIM-Ioff: PIM System with Rank & Channel interleaving switched off, (e.g. UPMEM)

Results on CPU Workloads

- Because of CPU Page's memory interleaving is switched on

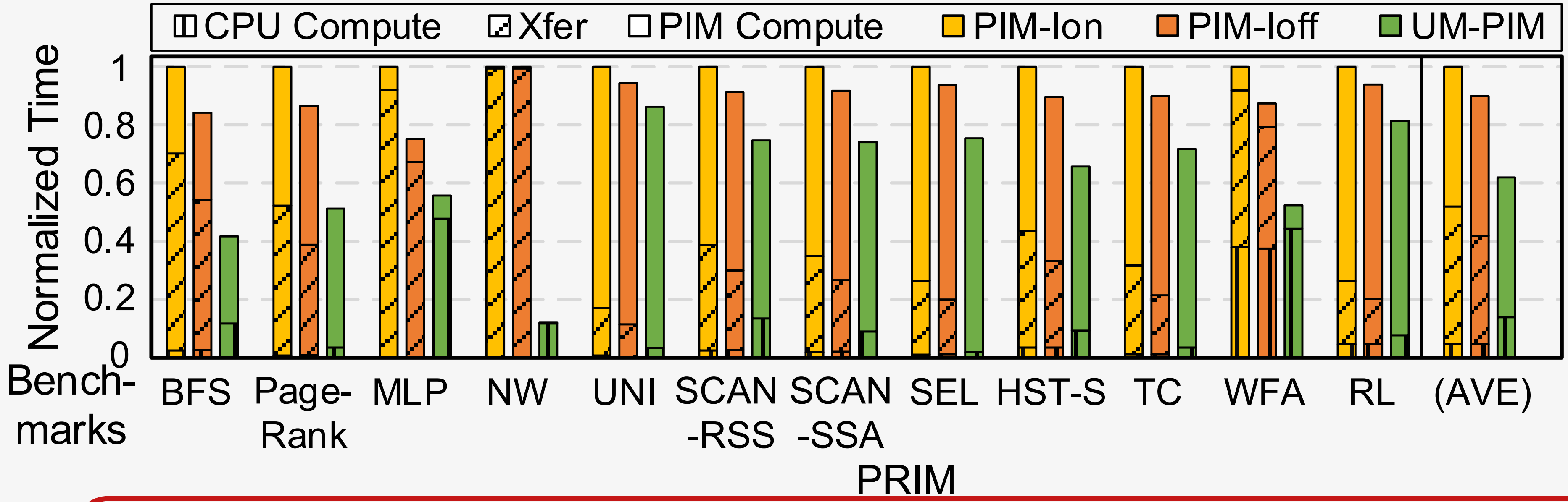


PIM-Ioff: 22% performance degradation because of memory interleaving switched off

UM-PIM: **<0.1% performance loss** because CPU page is interleaved.

Results on PIM Workloads

- UM-PIM do not need data transfer between isolated memory spaces

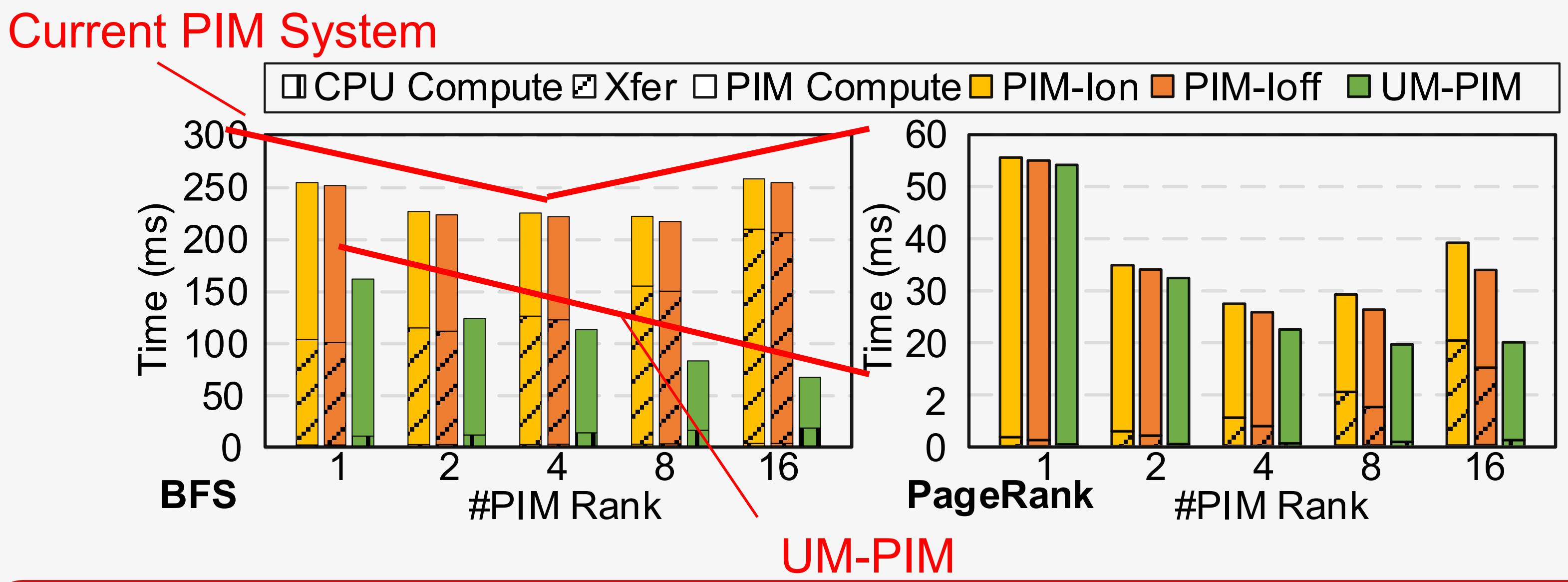


Compared to PIM-Ioff (Rank & Channel interleaving switched off, UPMEM) UM-PIM has **4.93× reduction on CPU time** (including Computing and Data Transfer), resulting in **1.96× speed up** on the whole workloads.

NW have intensive inter-PIM communication, therefore, it has the best speedup.

Results on PIM Workloads

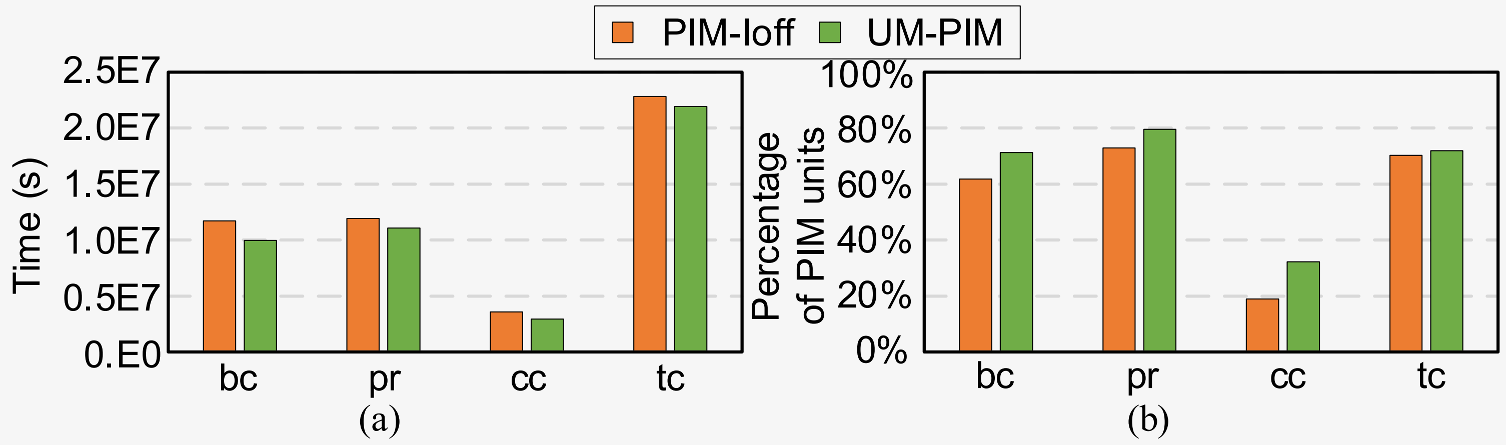
- UM-PIM do not need data transfer between isolated memory spaces



For workloads with intensive inter-PIM-units data transfer:
UM-PIM can **still benefit from computing using more PIM units**
In contrast, time on current PIM systems increases when using more PIM units

Analysis – Enable More PIM Tasks

- PIM compilers decide program segment offload to PIM units according to offload overhead & compute speedup. UM-PIM reduce offload overhead because of eliminating data transfer.



UM-PIM can offload **8% more program segments** to PIM units, resulting in 1.13x speedup

Discussion

Extend to different Device number of DRAM:

- Change number of SB in RC module. E.g. for x4 DRAM, RC circuit have 4 SBs.

Extend to other DRAM type:

- HBM and LPDDR do not have Device level in DRAM hierarchy. ATM and RC is not needed. Other APIs and memory management are still effective.

More in the paper

- Details of CPU and PIM access PIM pages
- Extended DRAM Instruction support for UM-PIM
- Inter-PIM-units Communication APIs
  - Four inter-PIM units communications APIs like NCCL

Conclusion

Key advantages of UM-PIM:

- **Uniform & shared memory space:** CPU and PIM pages co-exist in a uniform memory space, and data transfer is eliminated compared to isolated space design
- **Fast & Transparent data layout:** Dynamic address mapping and data re-layout for two kinds of pages are processed fast and transparently by DRAM-side hardware module
- **Compatibility with current CPU systems:** no modifications on CPU-side hardware and CPU page's memory management.

上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

Thanks!