Topics in Computer Architecture Research

Best Paper Nominees at HPCA 2024

Outline

- Tori Koizumi (Nagoya Institute of Technology), Ryota Shioya, She Sugita, Taichi, Amano, Yuya Degawa, Junichiro Kadomoto, Hidetsugu Irie, Shuichi Saki (University of Tokyo) “**Clockhands**: Rename-free Instruction Set Architecture for Out-of-order Processors”
- Bongjoon Hyun, Taehun Kim, Dongjae Lee, Minsoo Rhu (KAIST) “Pathfinding Future PIM Architectures by **Demystifying a Commercial PIM Technology**”
- Johannes Wikner, Daniel Trujillo, Kaveh Razavi (ETH Zurich) “**Phantom**: Exploiting Decoder-detectable Mispredictions”

Clockhands (Best Paper Nominee MICRO23)

Remember Re-ordering?

Minimize WAR and WAW hazards by **register renaming**!

```
div t0, t1, t2
add t3, t0, t4
sw t3, 0(s0)
sub t4, t5, t6
mul t3, t5, t4
```

What if we could use temporary registers X and Y?

```
div t0, t1, t2
add X, t0, t4
sw X, 0(s0)
sub Y, t5, t6
mul t3, t5, Y
```

We can do work between when we wait fro the result of the division by reordering!

```
div t0, t1, t2
sub Y, t5, t6
mul t3, t5, Y
add X, t0, t4
sw X, 0(s0)
```

Anyone ever write code like this?

```
// step 1. access some data
int *x = arr[1024];
*x += 2;
// step 2. access some other data
x = arr[2048];
*x += 4;
```

in assembly…

```
// step 1. access some data
lw a0, 1024(s0)
addi a0, a0,
sw a0, 1024
// step 2. acc                    a
lw a0, 2048(s0)
addi a0, a0, 4
sw a0, 2048(s0)
```

Is this a dependence?

Problem

- O3 CPUs use a lot of power to perform tasks like register renaming, but register renaming is limited by false dependences
- False dependences are somewhat inevitable because developers (and compilers) are bad at assigning registers!
- But, this isn’t their fault, it’s the ISA’s fault!

Chat with your neighbors!

- What questions do you have?
- Brainstorm: How could we fix the problem?
    - What leads to a false dependence?
    - What about the ISA causes this?

Proposed Solution

- No destination registers in ISA!
- All instructions operands are in terms of “how many instructions ago” was the operand produced

Proposed Solution (v1)

```
// step 1. access some data
lw a0, 1024(s0)
addi a0, a0, 2
sw a0, 1024(s0)
// step 2. access some other data
lw a0, 2048(s0)
addi a0, a0, 4
sw a0, 2048(s0)
```

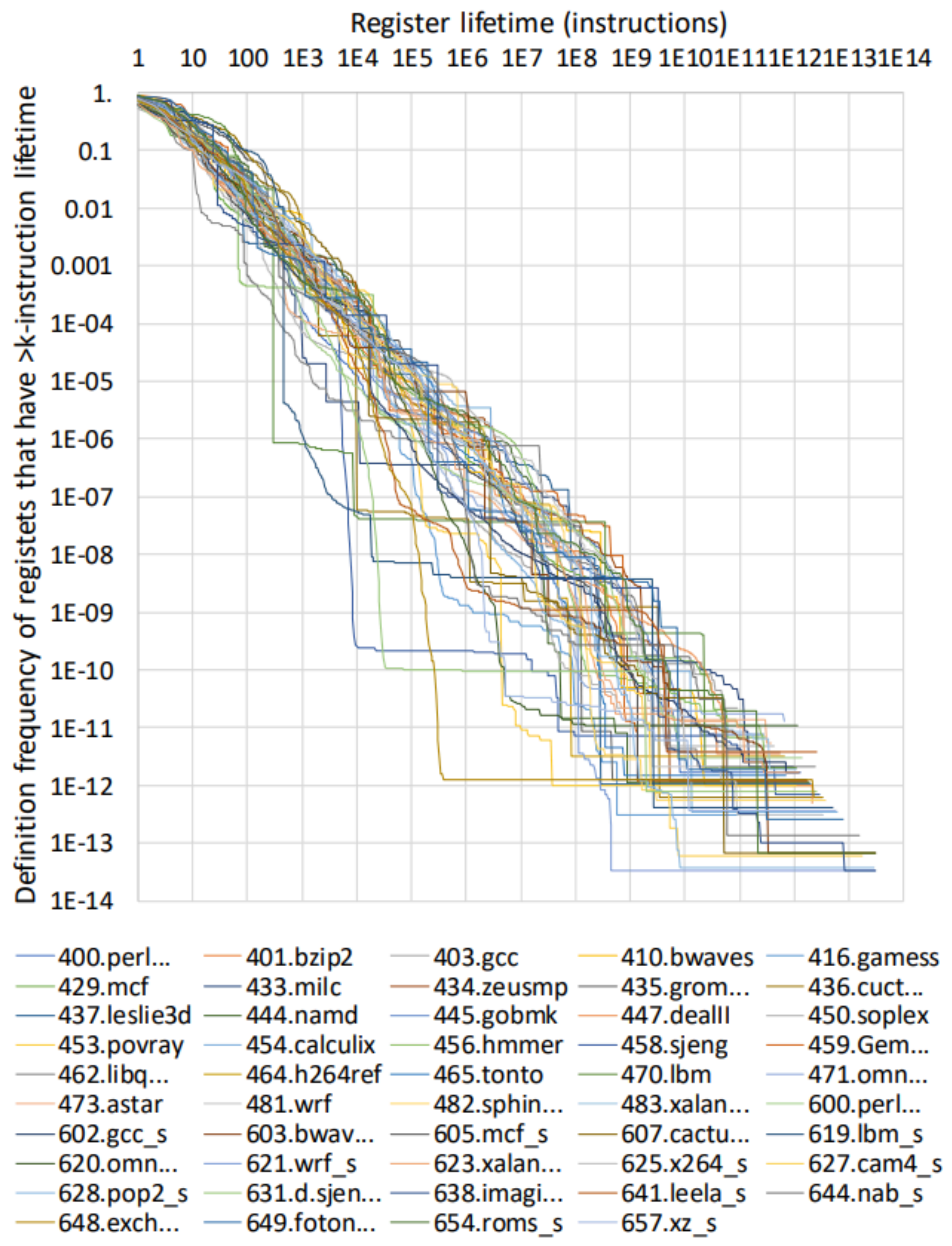
Becomes...

```
// step 1. access some data
lw 1024([n])
addi [0], 2
sw [0], 1024([n + 2])
// step 2. access some other data
lw 2048([n + 3])
addi [0], 4
sw [0], 2048([n + 5])
```

Next register is allocated from ring buffer of available physical registers!

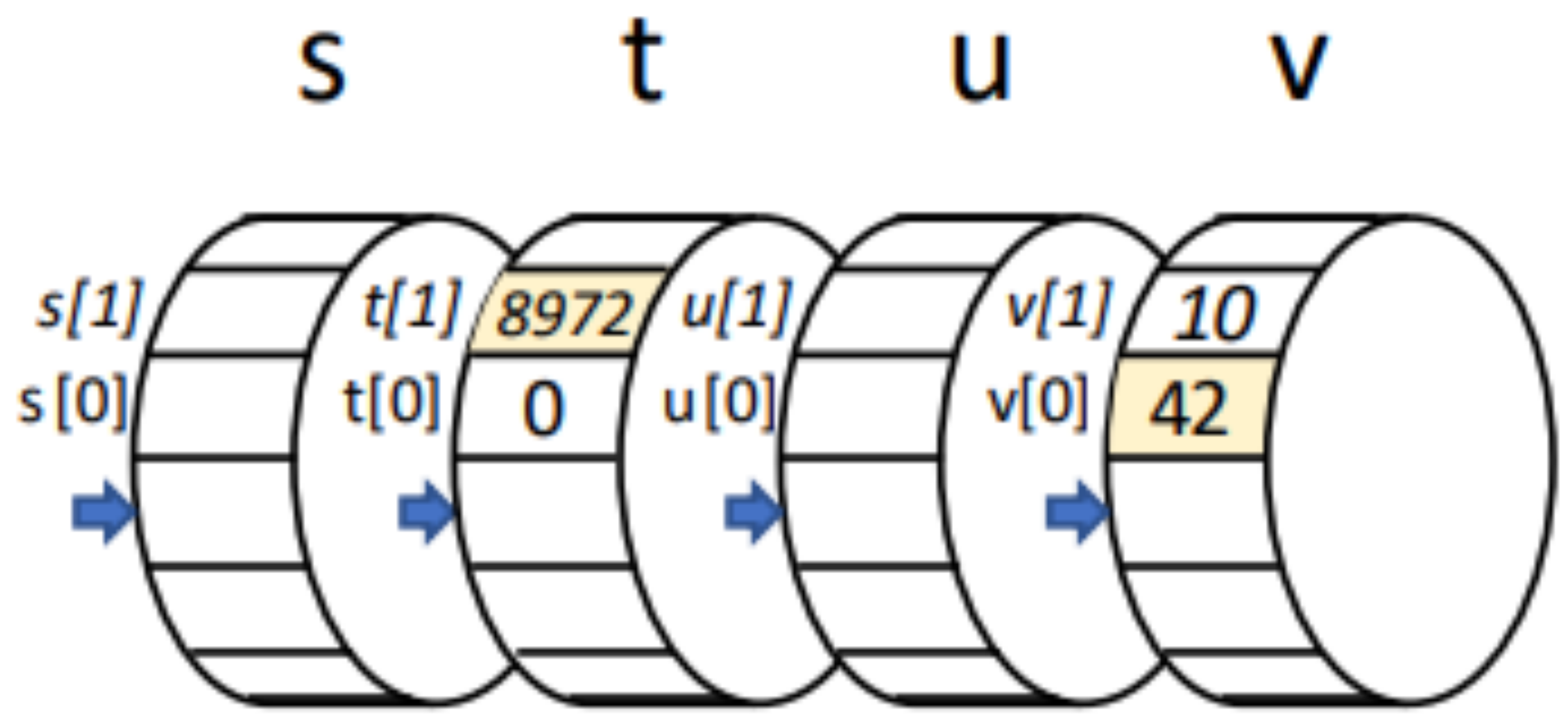
Proposed Solution (v1)

- No false dependences,
- Uses way more register r almost every instruction
- How do we handle loop
    - *for (int i = 0; i < 100;*
    - *i* keeps getting furthe
    - Handle this by calling nce away

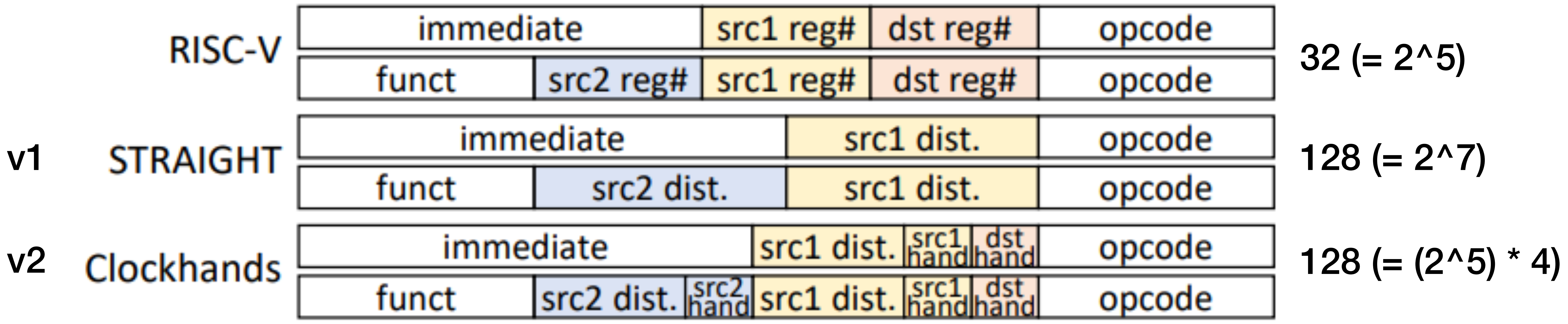


Proposed Solution (v2)

- Don't just naively allocate from a ring buffer…
- Different variables have different purposes
  1. s —> stack pointer and function args
  2. t —> temporary variables
  3. u —> variables with long lifetimes
  4. v —> loop constants
- Let's allocate registers according to their usage as defined by the developer!



Proposed Solution (v2)



What is the maximum allowable number of usable registers in each ISA?

Proposed Solution (v2)

- Implemented compiler as LLVM extension
- Implemented ISA on an FPGA

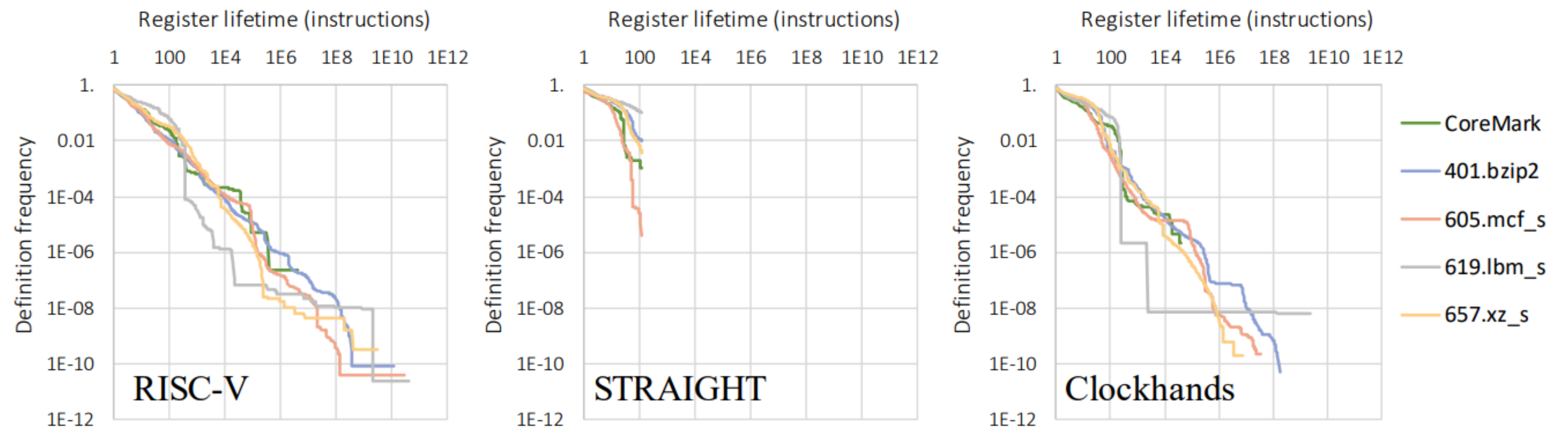


Figure 17: Frequency at which a destination register is defined with a lifetime greater than a certain number of instructions (same as Fig. 4).

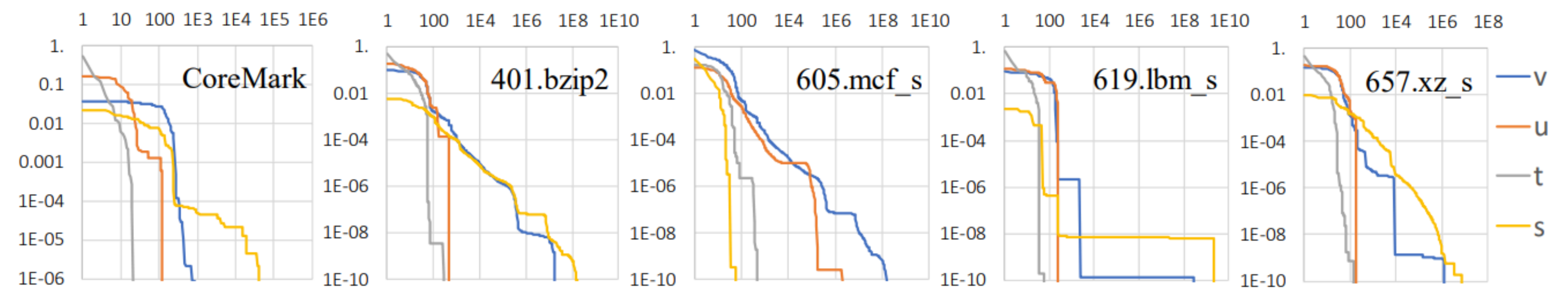
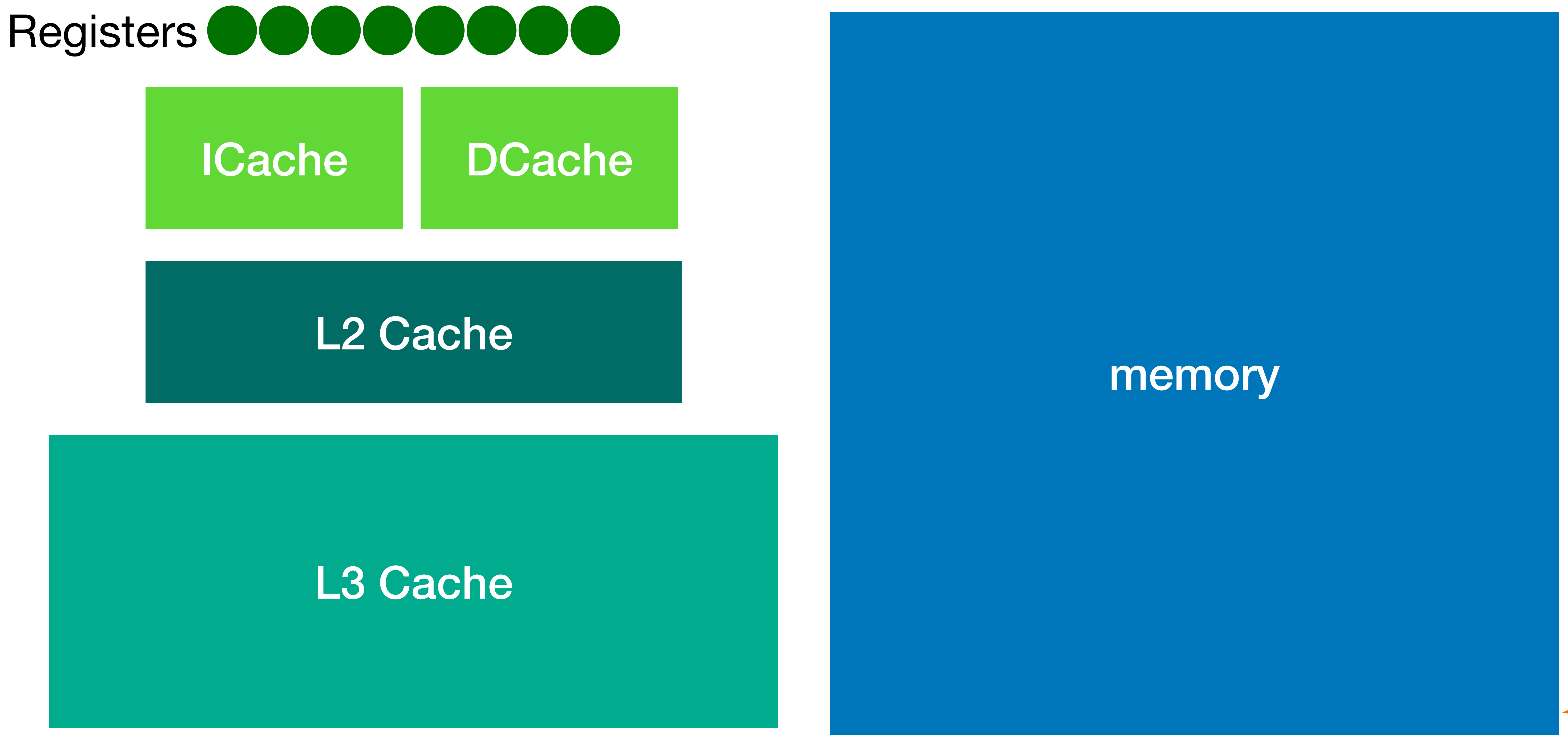


Figure 18: Frequency at which a destination register is defined with a lifetime greater than a certain number of instructions (same as Fig. 4). The vertical axes indicate definition frequency and the horizontal axes indicate register lifetime.

What do you think?

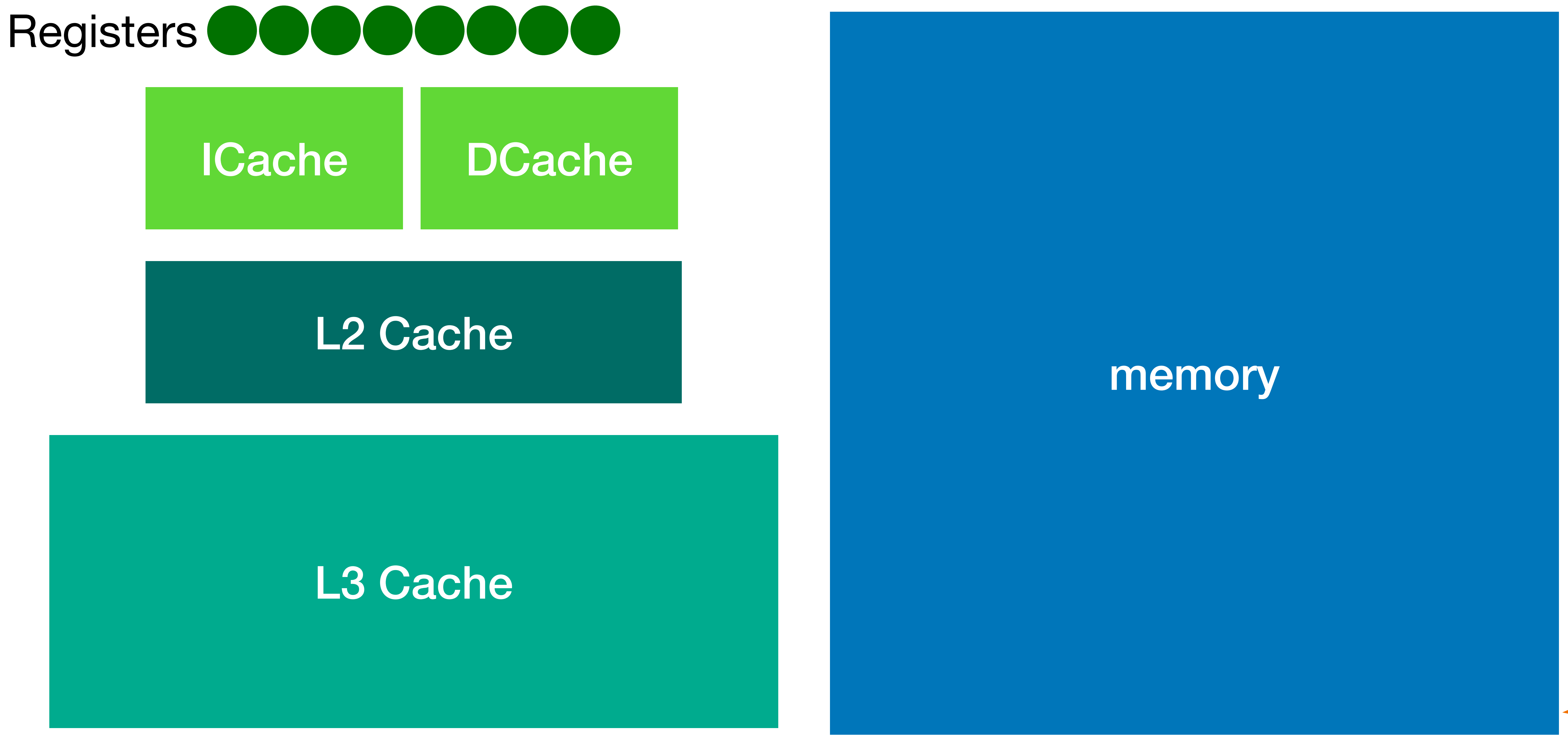
Demystifying a Commercial PIM Technology (Best Paper Winner HPCA24)

Remember the Memory Hierarchy?



Thinking about architecture as boxes hides details!

Let’s Dig into Memory!



“According to our study, more than 88% of the total training time is consumed by transferring data” — SmartInfinity (HPCA 2024)

Let’s Dig Into Memory!

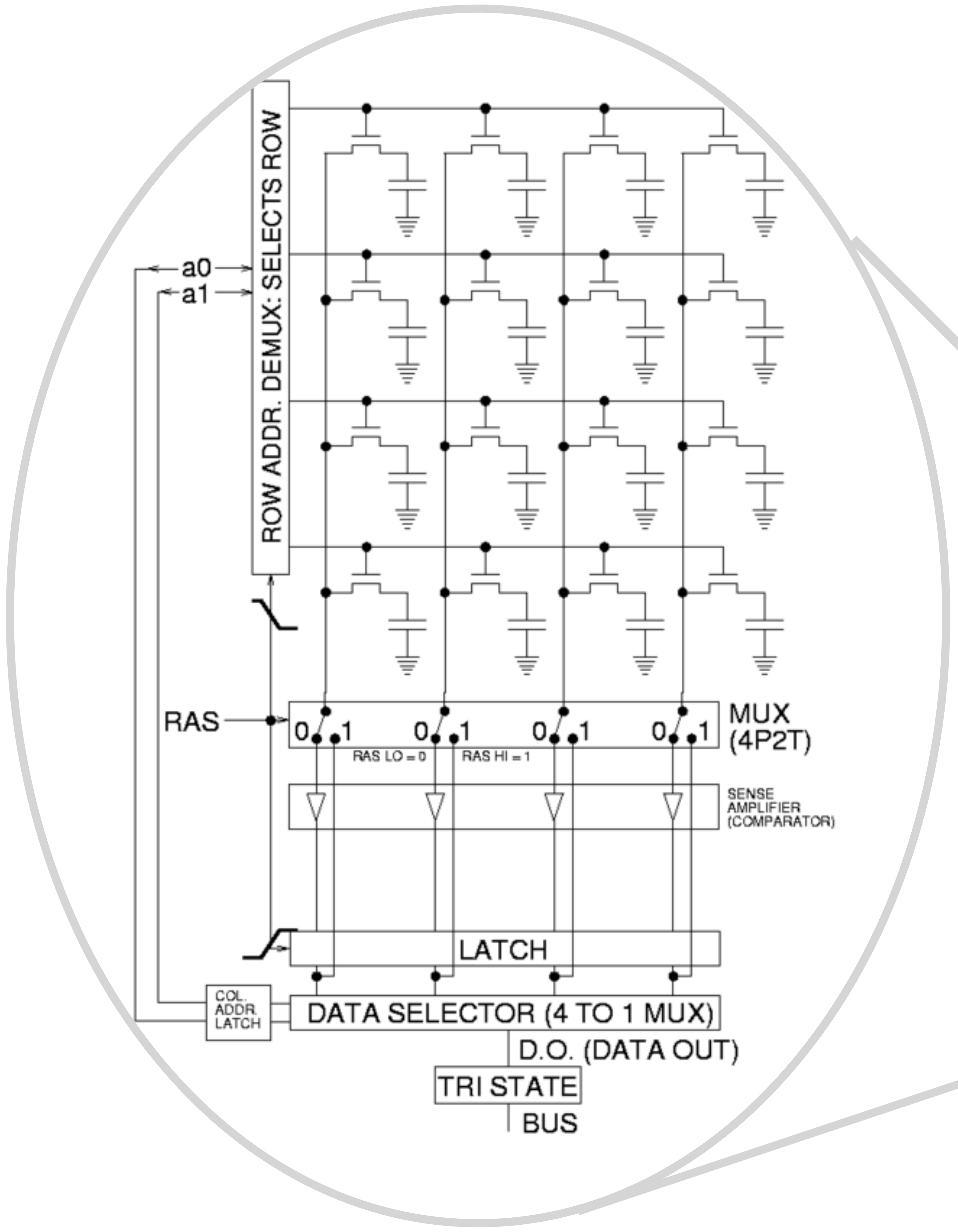


Image credit: https://en.wikipedia.org/wiki/Dynamic_random-access_memory

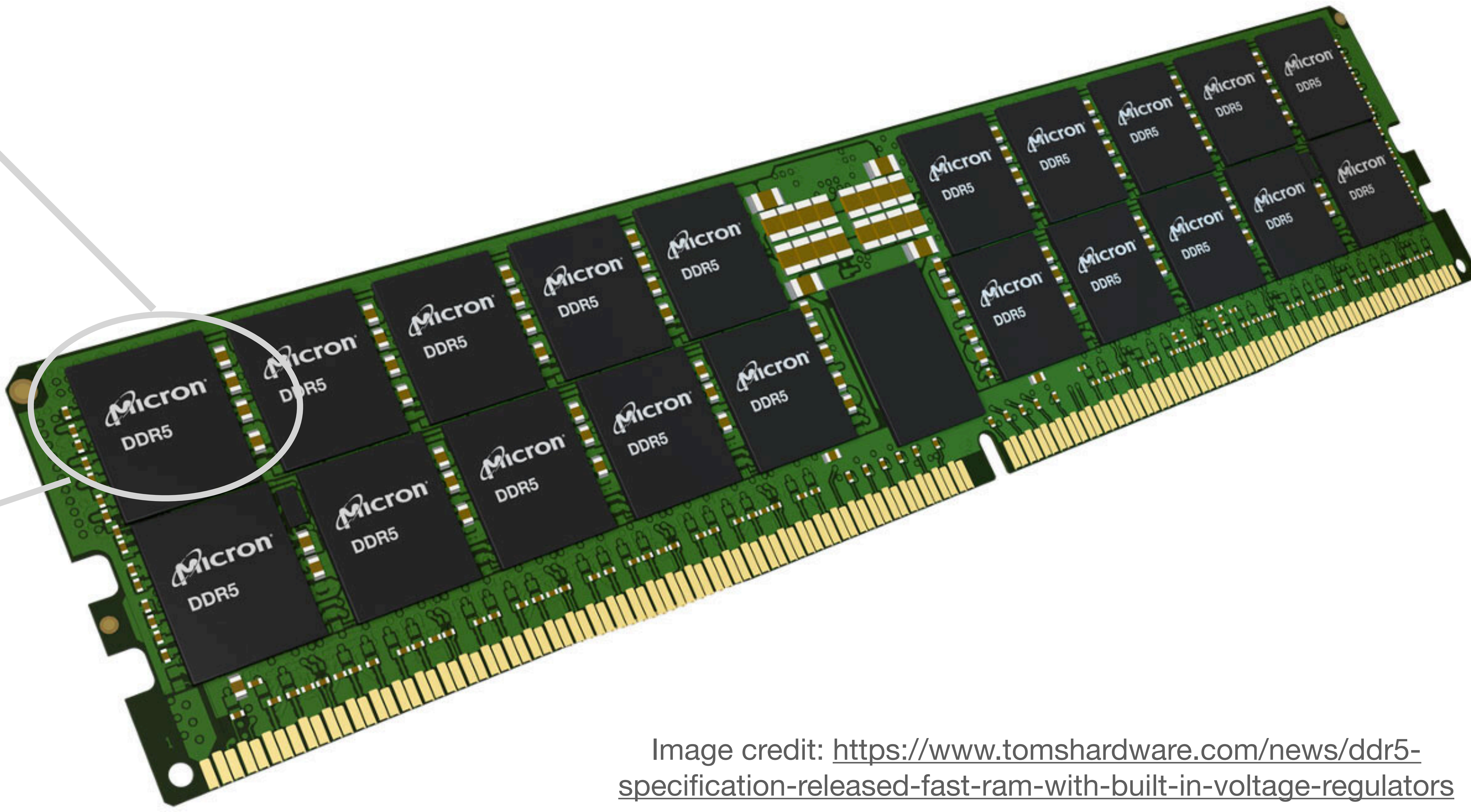


Image credit: https://www.tomshardware.com/news/ddr5-specification-released-fast-ram-with-built-in-voltage-regulators

Let’s Dig Into Memory!

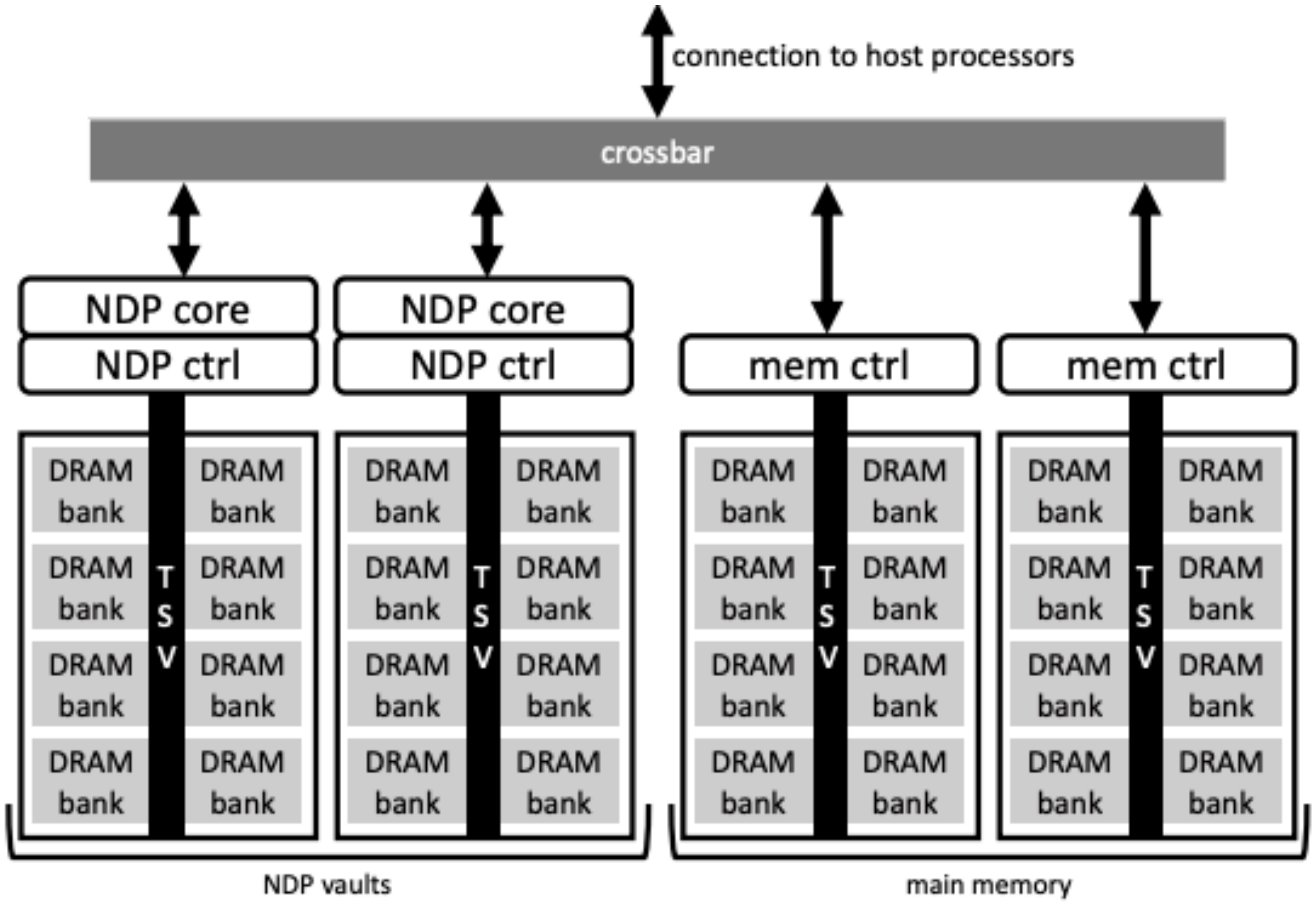


Image credit: my former lab mate Jiwon Choe :-)

Processing In-Memory (PIM)

- What if we didn't have to transfer data into the memory hierarchy?
- What if there was a data processing unit on the memory die?

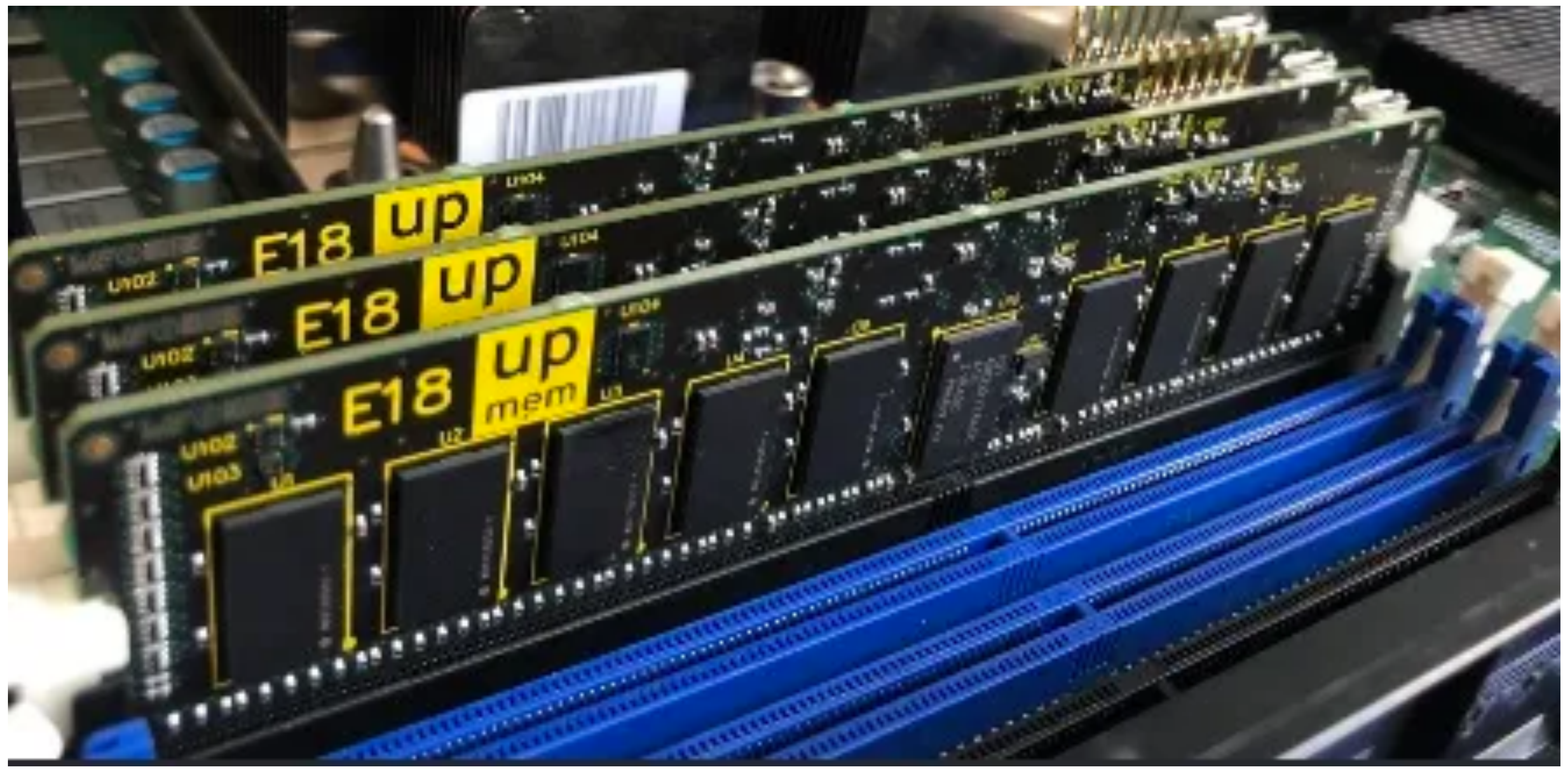


Image credit: https://thememoryguy.com/upmem-releases-processor-in-memory-benchmark-results/

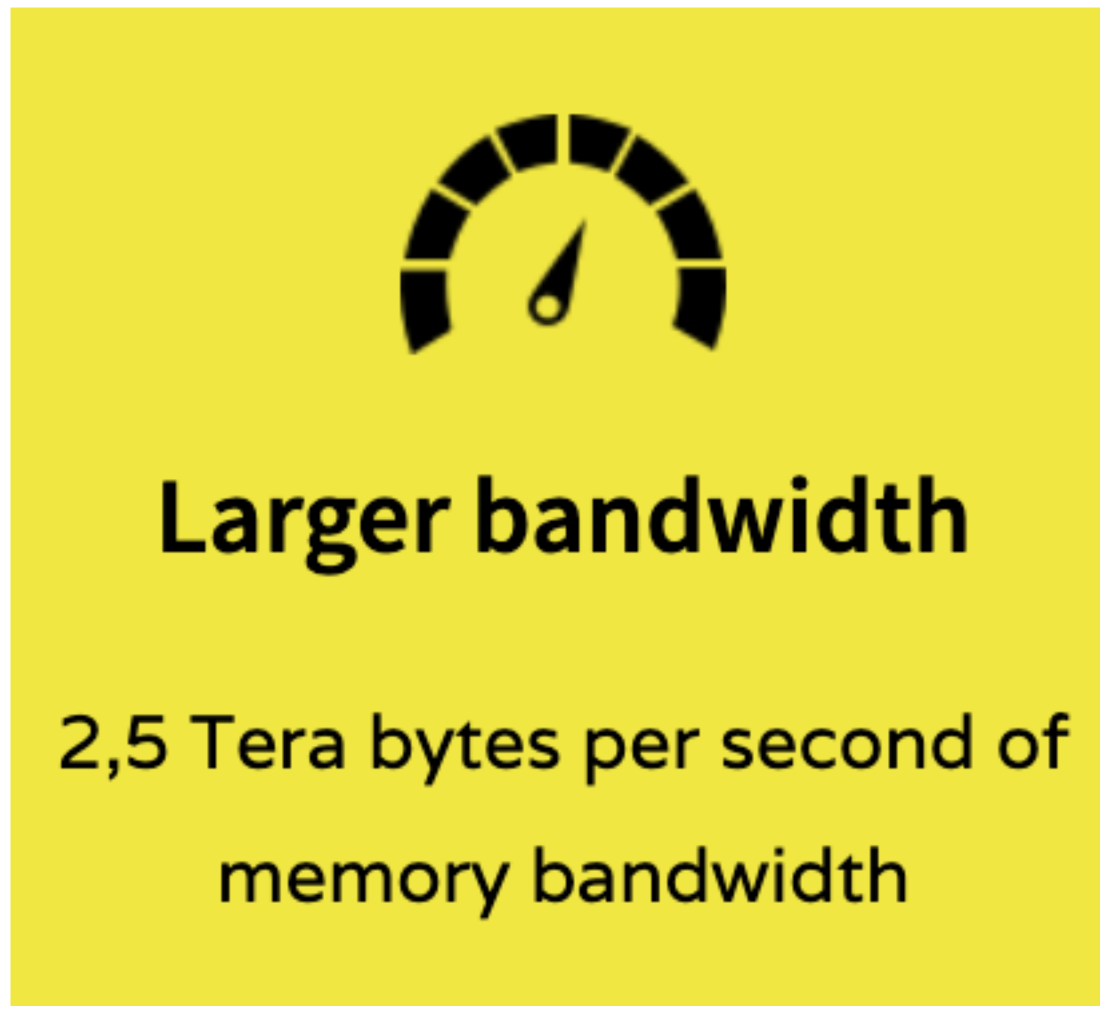


Image credit: https://www.upmem.com/

Processing In-Memory (PIM)

- Great idea, right?
- “We’ve investigated applying PIM to our workloads and determined there are several challenges to using these approaches. Perhaps the biggest challenge of PIM is its programmability. **It is hard to anticipate future model compression methods, so programmability is required to adapt to these**. PIM must also support flexible parallelization since it is hard to predict how much each dimension (of embedding tables) will scale in the future.”
  — Facebook, 2021

UPMEM-PIM

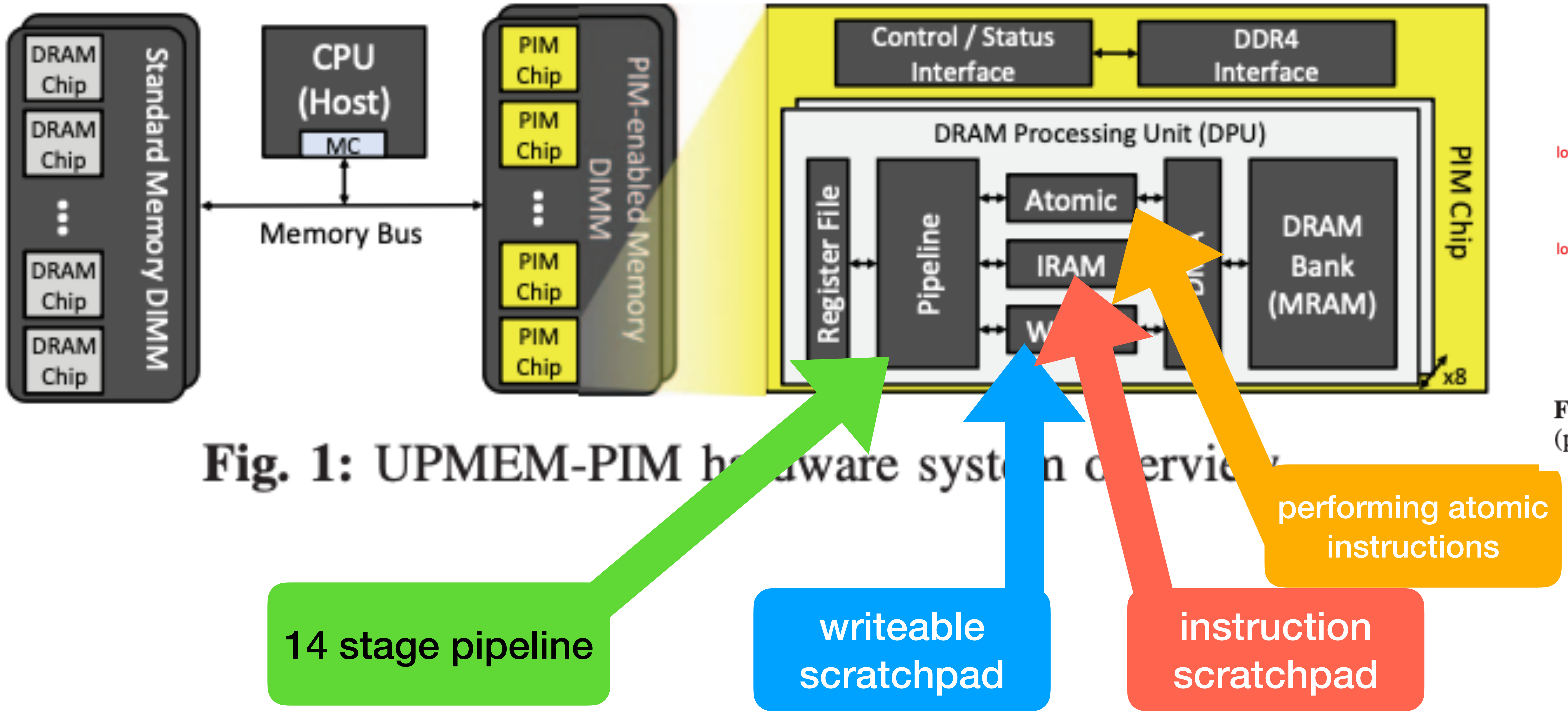


Fig. 1: UPMEM-PIM h aware syst n o ervie

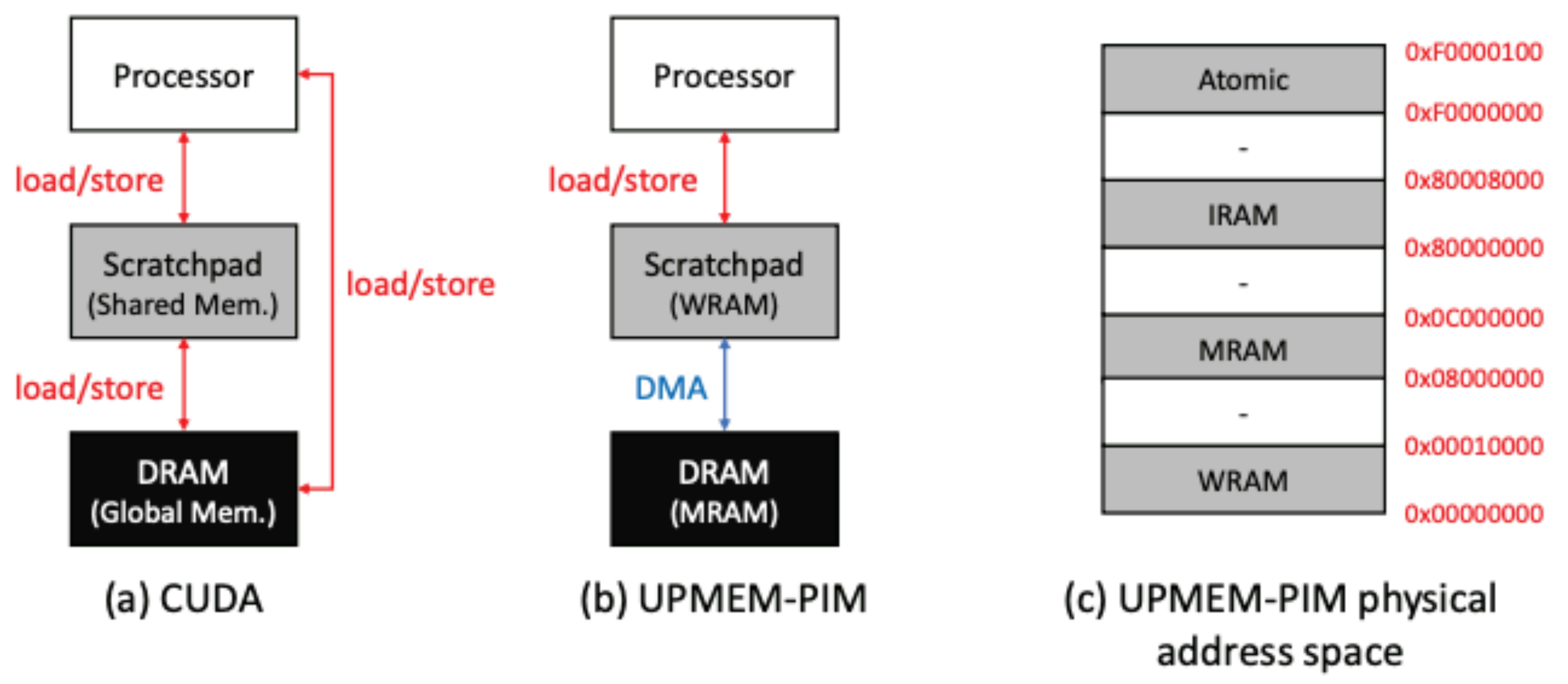
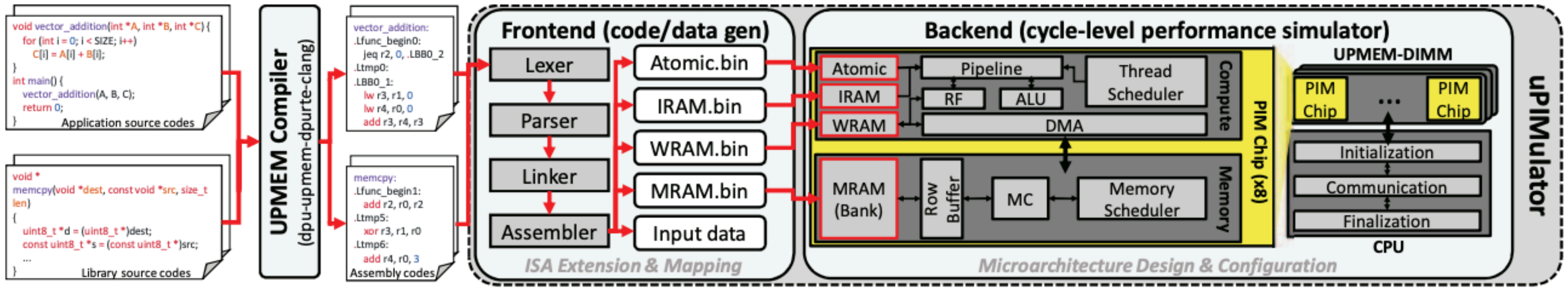


Fig. 3: Memory model of (a) CUDA and (b) UPMEM-PIM. (c) The (physical) address map of UPMEM-PIM.

UPMEM-PIM

- 20 double-ranked UPMEM-PIM DIMMs
- 8 DPUs per DRAM bank (where each has 64MB of private memory + scratchpads)
- 8 DPU chips per memory rank
- 20 x 2 x 8 x 8 = 2560 DPUs!

uPIMulator


```
void vector_addition(int *A, int *B, int *C) {
  for (int i = 0; i < SIZE; i++)
    C[i] = A[i] + B[i];
}
int main() {
  vector_addition(A, B, C);
  return 0;
}
```
Application source codes

```
void *
memcpy(void *dest, const void *src, size_t
len)
{
  uint8_t *d = (uint8_t *)dest;
  const uint8_t *s = (const uint8_t *)src;
  ...
}
```
Library source codes

```
vector_addition:
.Lfunc_begin0:
  jeq r2, 0, .LBB0_2
.Ltmp0:
.LBB0_1:
  lw r3, r1, 0
  lw r4, r0, 0
  add r3, r4, r3
```

```
memcpy:
.Lfunc_begin1:
  add r2, r0, r2
.Ltmp5:
  xor r3, r1, r0
.Ltmp6:
  add r4, r0, 3
```
Assembly codes

Fig. 4: uPIMulator simulation framework overview.

Chat with your neighbors!

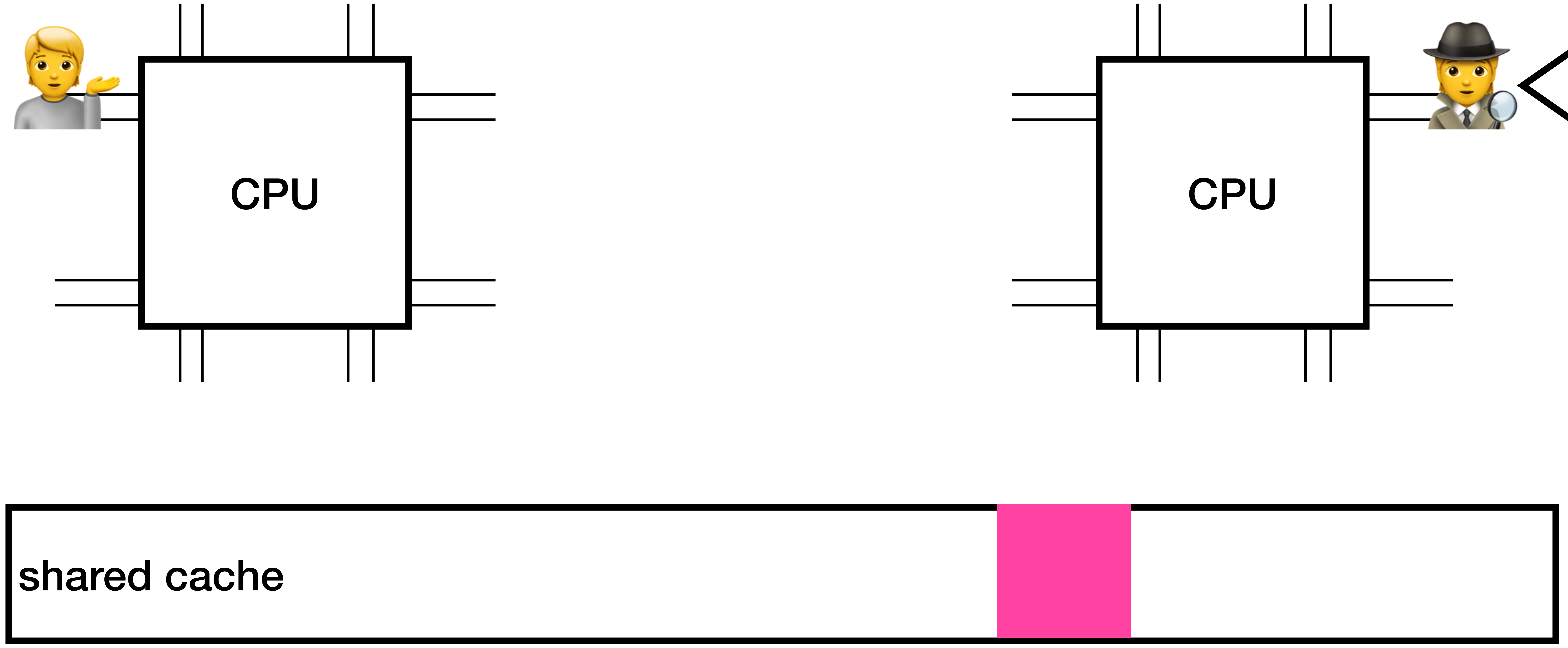
- What do you think about PIM?
  - Do you agree with Facebook’s claim about usability?
  - What do you think are the implications of specialization versus generalizability?
  - Will UPMEM fail???

Limitations of PIM

- Really hard to do coherence between processor memory and PIM memory
- Is it useful to have programmable logic on a memory die?
  - Why not just make a fixed-logic accelerator?
- What if my offloaded program could benefit from a bigger cache?
- What's the right granularity of computation? PIM versus NDP versus NMP versus NSP versus PUM

Phantom (Best Paper Nominee MICRO23)

Remember Flush + Reload Attack?

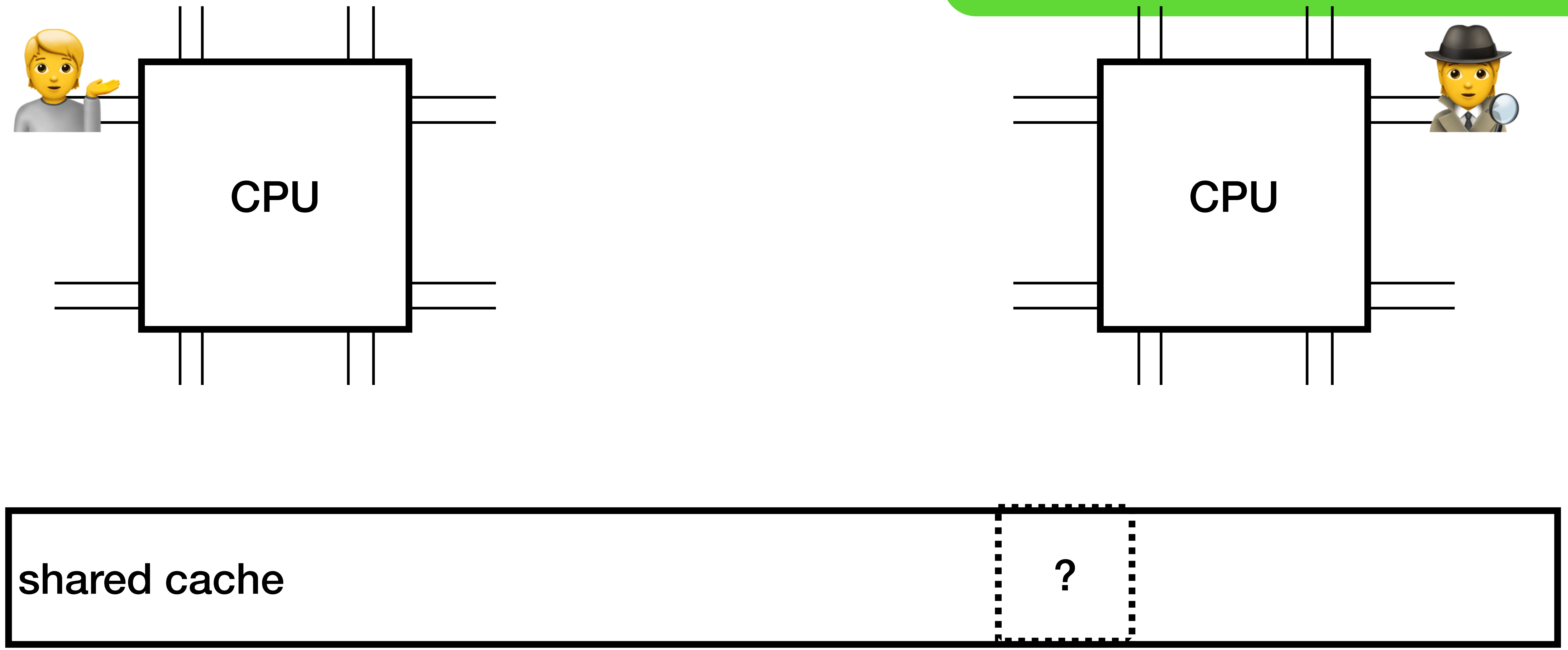


```
// flush the line
clflush 0xLIBCADDR;

// wait some time
t1 = time.now();
while (time.now() - t1 < 100ns) {};

// access line
t2 = time.now()
x = *(0xLIBCADDR);
access_time = time.now() - t2;

// if slow access, unused
// else, used!
```

Remember Flush + Reload Attack?

```
if (strcmp(argv[1], "supersecretdata") == 0) {
  x = data[addr1];
} else {
  x = data[addr2];
}
```

We know this is going to execute speculatively!



Remember Speculative Execution?

- We use the BPB to predict outcomes and the BTB to predict targets
    - This allows the processor to race ahead of the actually evaluated truth most of the time!
- We use the ROB to execute instructions without committing them
- Window of time between when a prediction is made and is evaluated is called *transient execution* or the *speculative window*
- Code snippets (gadgets) that leak information speculatively are called *spectre gadgets*

Types of Spectre Attacks

- Spectre v1 (Bounds Check Bypass): If an array is accessed inside a speculative window at a parameterized index, pass an illegal index to read memory from elsewhere — the branch will be squashed and the data will be in the cache

- Spectre v2 (Branch Target Injection): An attacker may poison all of the potential branch targets for particular addresses to point to some malicious code. Only when the branch target is resolved will the malicious region be squashed, but the data from this malicious region will be in the cache

- …

- Phantom: you will help unpack!

Chat with your neighbors!

“We hypothesize that the asymmetric combinations of branch types will likely lead to short mispredictions that the CPU can detect during decode due to mismatching instruction types. Consequently, our analysis could benefit from observation channels that allow us to infer how far in the pipeline a mispredicted control flow advances. For example, if we observe transient memory operations from the mispredicted target, we can infer that the mispredicted control flow reached Execute (EX) and advanced through the preceding stages, namely IF and ID”

Phantom

- Create a target that maps from instruction at address A to target instruction C
- Flush instruction B from the ICache so that fetching it and decoding it will be slow
- So long as B also maps to C, C will execute transiently

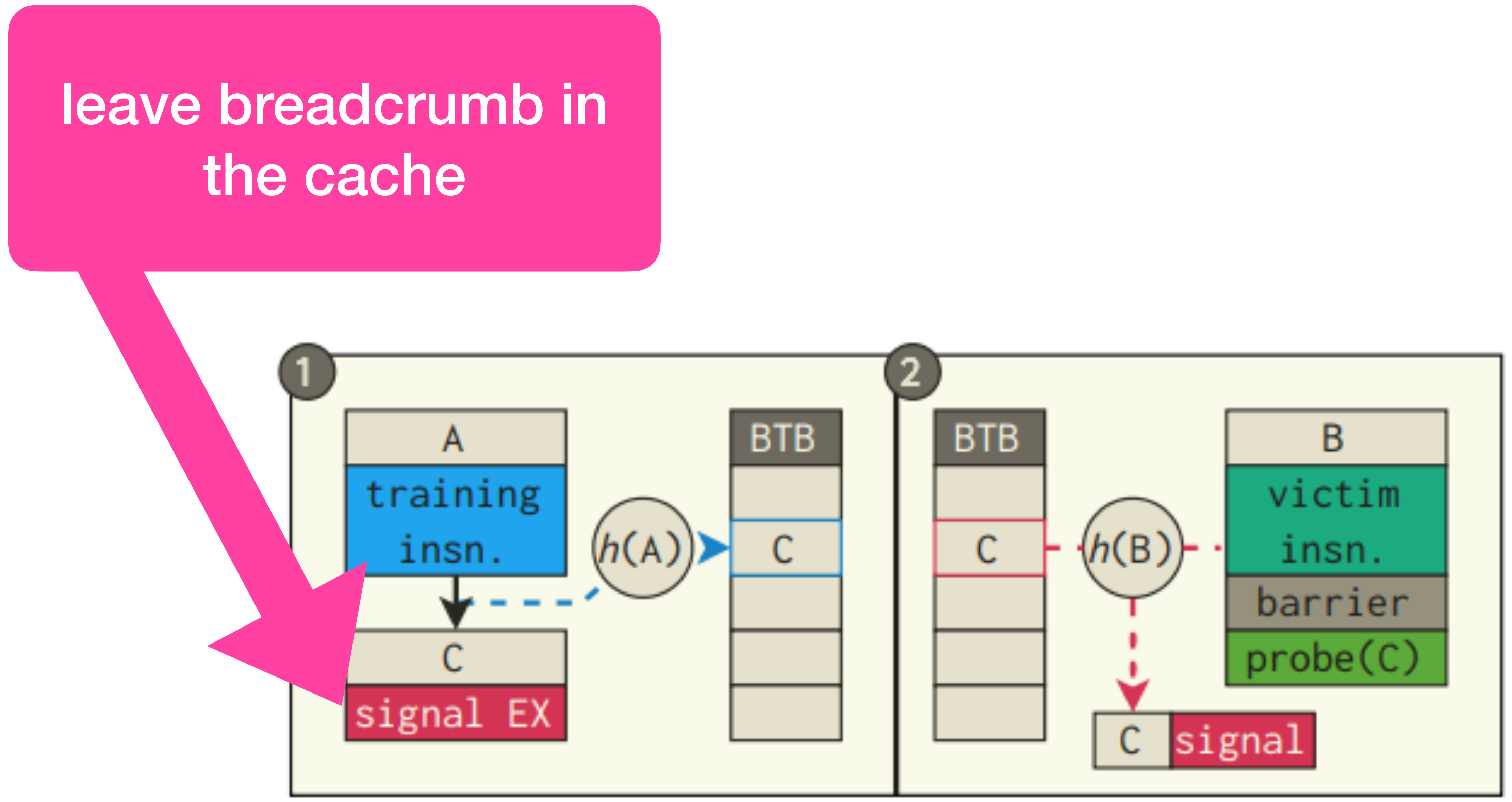
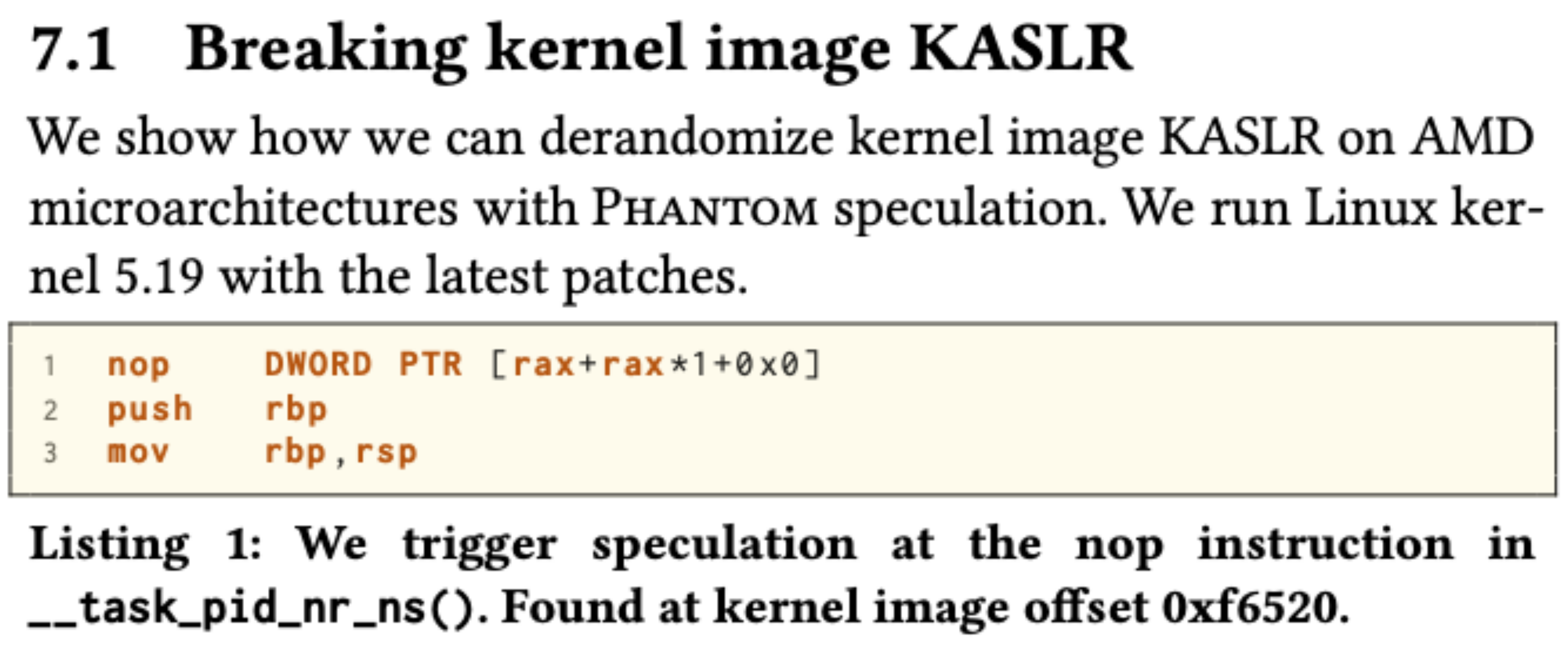


Figure 4: In ❶, *A* creates a BTB entry to *C*, so that in ❷, the victim instruction of *B* may reuse that BTB entry. The instructions in *C* emit a transient execution signal. By fetching and decoding *C*, transient fetch and transient decode signals are already emitted.

Phantom

- How bad is it?
- Take Vasilis' class to see just how bad this is!



7.1 Breaking kernel image KASLR

We show how we can derandomize kernel image KASLR on AMD microarchitectures with PHANTOM speculation. We run Linux kernel 5.19 with the latest patches.

```
nop     DWORD PTR [rax+rax*1+0x0]
push    rbp
mov     rbp,rsp
```

Listing 1: We trigger speculation at the nop instruction in __task_pid_nr_ns(). Found at kernel image offset 0xf6520.

Phantom: Follow Up Thoughts

- Is this *actually* a new Spectre variant?
- Is this dangerous?
- Is this cool?

Concluding Thoughts

- Research in architecture is super diverse!
- General themes are:
    1. Software uses hardware, so hardware should be better!
    2. Software uses hardware, so software should be better!
    3. Architecture is heterogeneous, can we offload?
    4. Can we make components in architecture more efficient?
    5. Architecture is broken because it's insecure
- You are now hopefully equipped with the vocabulary to see why these problems are interesting and/or hard!