Multithreaded Architectures and Programming on Multithreaded Architectures: Cannot do this alone

Hung-Wei Tseng

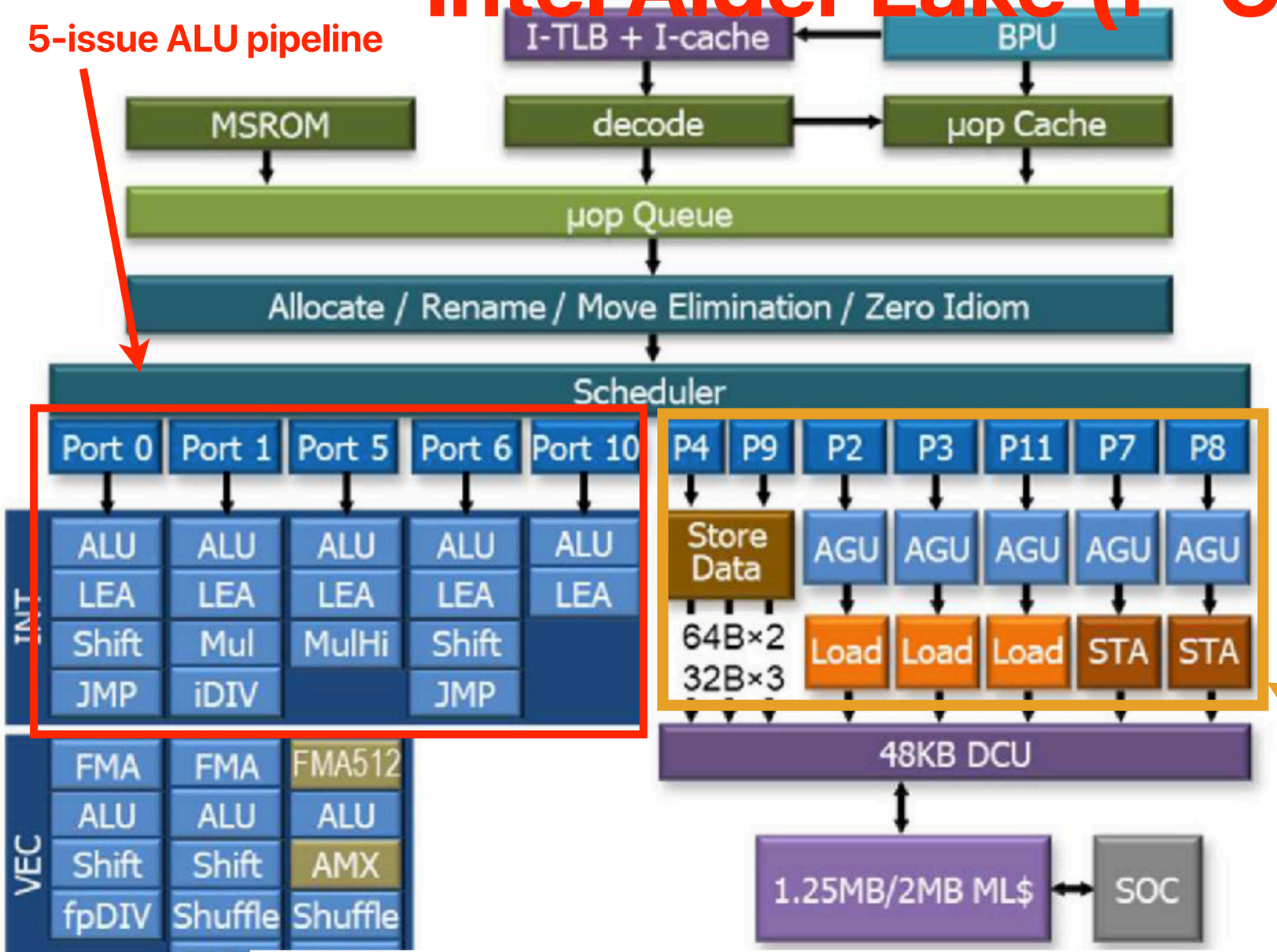
Intel Alder Lake (P-Core)

$$MinCPI = \frac{1}{12}$$

$$MinINTInst.CPI = \frac{1}{5}$$

$$MinMEMInst.CPI = \frac{1}{7}$$

$$MinBRInst.CPI = \frac{1}{2}$$



https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf

Summary: Characteristics of modern processor architectures

- Multiple-issue pipelines with multiple functional units available
  - Multiple ALUs
  - Multiple Load/store units
  - Dynamic OoO scheduling to reorder instructions whenever possible
- Cache — very high hit rate if your code has good locality
  - Very matured data/instruction prefetcher
- Branch predictors — very high accuracy if your code is predictable
  - Perceptron
  - Variable history predictors

Recap: Tips of programming on modern processors

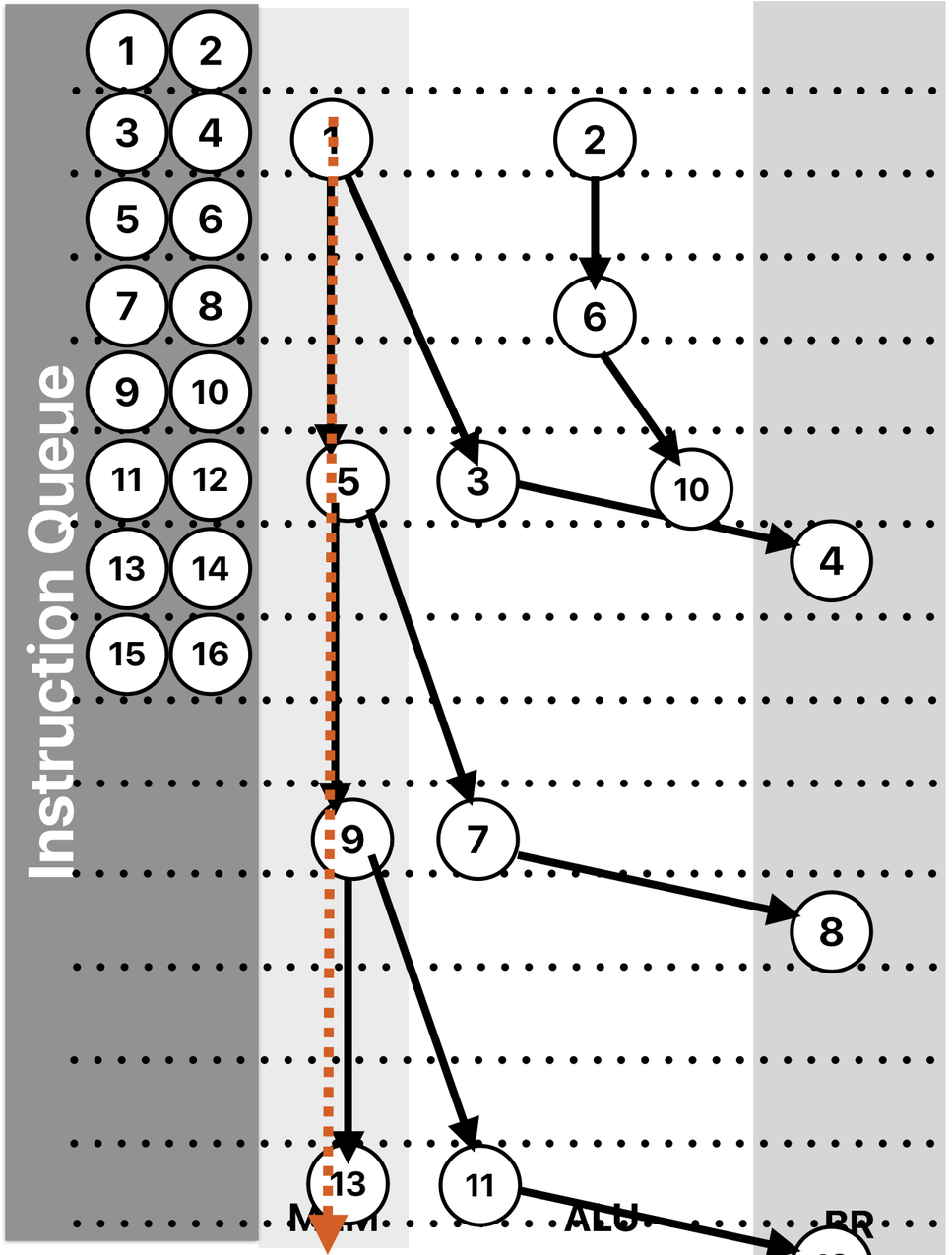
- Minimize the critical path operations
  - Don’t forget about optimizing cache/memory locality first!
    - Memory latencies are still way longer than any arithmetic instruction
    - Can we use arrays/hash tables instead of lists?
  - Branch can be expensive as pipeline get deeper
    - Sorting
    - Loop unrolling
  - Still need to carefully avoid long latency operations (e.g., mod)
- Since processors have multiple functional units — code must be able to exploit instruction-level parallelism
  - Hide as many instructions as possible under the “critical path”
  - Try to use as many different functional units simultaneously as possible
- Modern processors also have accelerated instructions
- Compiler can do fairly go optimizations, but with limitations

What if we have “unlimited” fetch/issue width — “linked list”

Doesn’t help that much!
— It’s important that the programmer should write code that can exploit “ILP”
— But — there’re always cases we cannot do further in ILP

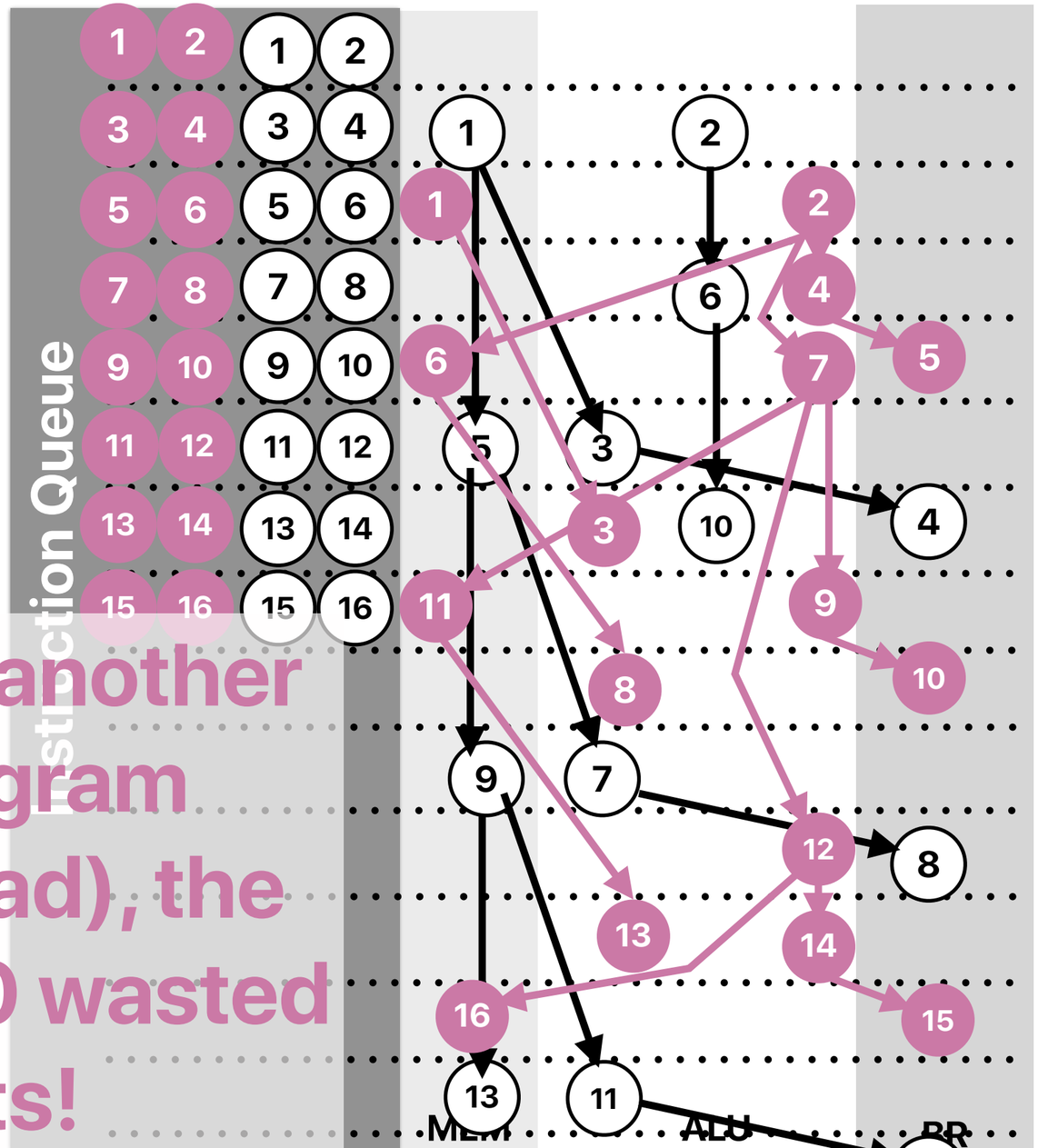
```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL );
```

```
① .L3:    movq    8(%rdi), %rdi
②         addl    $1, %eax
③         testq   %rdi, %rdi
④         jne     .L3
```



Recap: Simultaneous Multithreading (SMT)

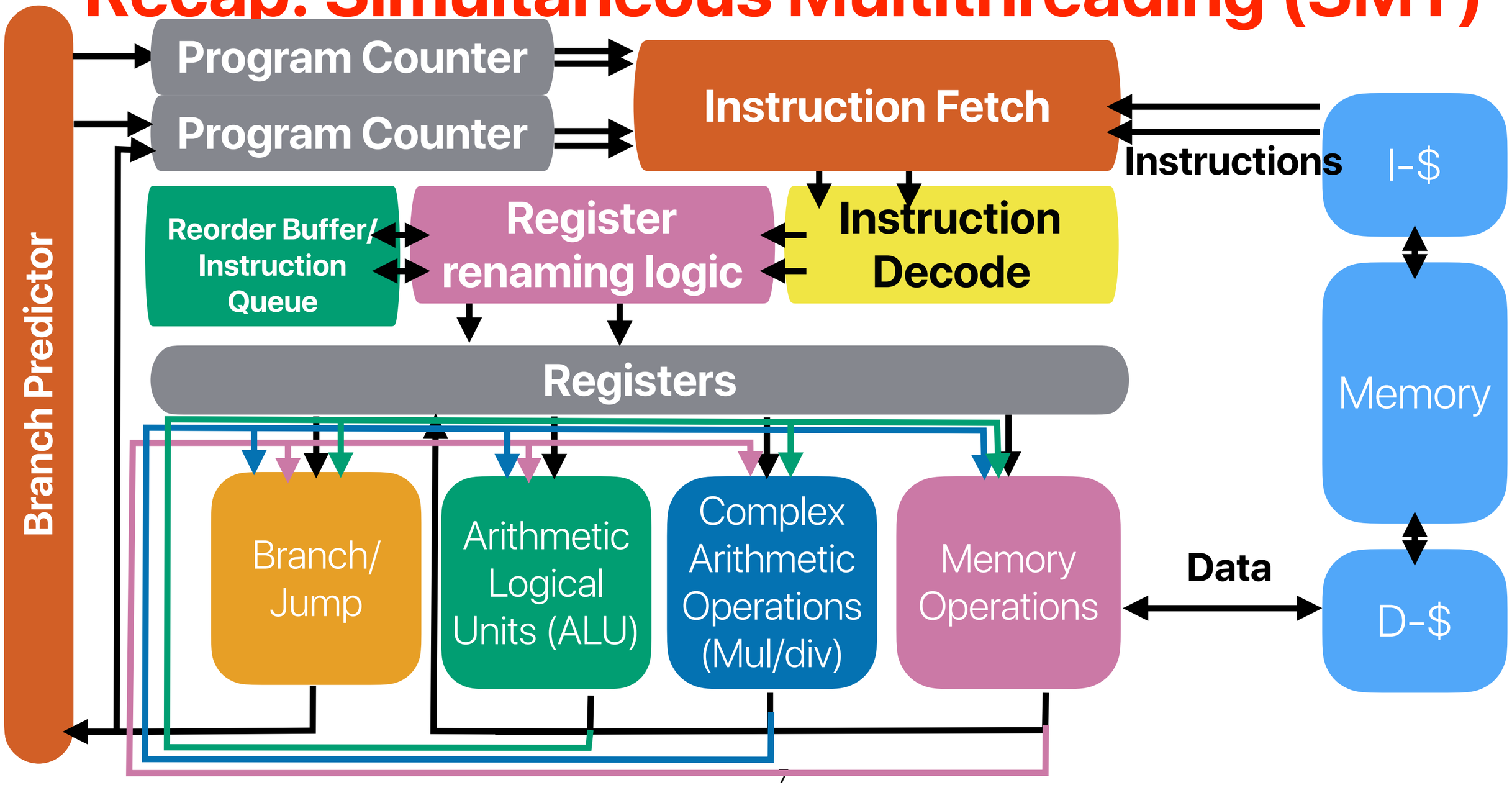
```
① movq     8(%rdi), %rdi
② addl     $1, %eax
③ testq    %rdi, %rdi
④ jne      .L3
⑤ movq     8(%rdi), %rdi
⑥ addl     $1, %eax
⑦ testq    %rdi, %rdi
⑧ jne      .L3
⑨ movq     8(%rdi), %rdi
⑩ addl     $1, %eax
⑪ testq    %rdi, %rdi
⑫ jne      .L3
```



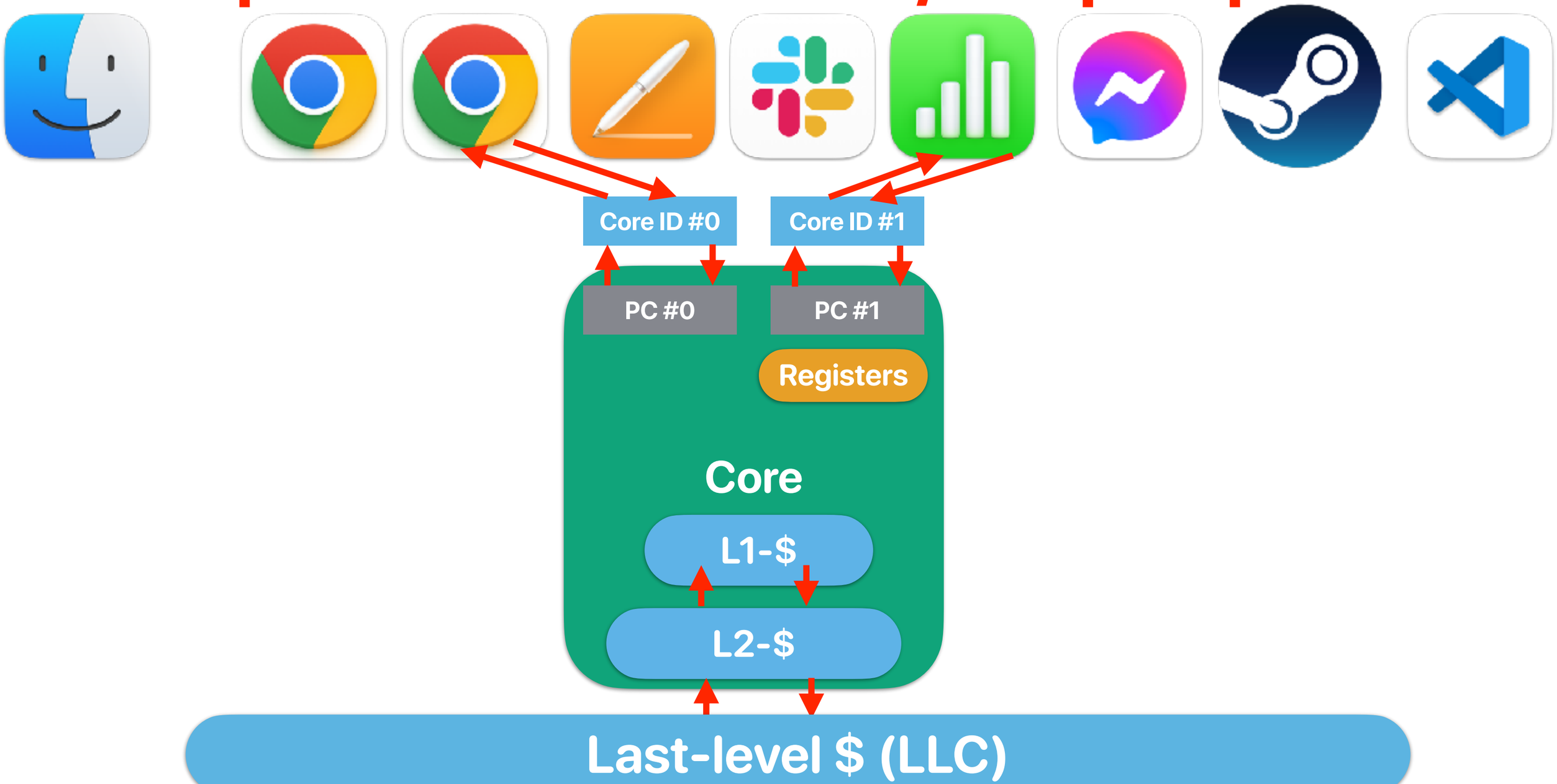
By scheduling another running program instance (thread), the processor has 0 wasted issue slots!

```
① movl (%rdi), %ecx
② addq $4, %rdi
③ addl %ecx, %eax
④ cmpq %rdx, %rdi
⑤ jne  .L3
⑥ movl (%rdi), %ecx
⑦ addq $4, %rdi
⑧ addl %ecx, %eax
⑨ cmpq %rdx, %rdi
⑩ jne  .L3
⑪ movl (%rdi), %ecx
⑫ addq $4, %rdi
⑬ addl %ecx, %eax
⑭ cmpq %rdx, %rdi
⑮ jne  .L3
⑯ movl (%rdi), %ecx
```

Recap: Simultaneous Multithreading (SMT)



Recap: SMT from the user/OS' perspective



```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           151
Model name:                      12th Gen Intel(R) Core(TM) i3-12100F
Stepping:                        5
CPU MHz:                         3300.000
CPU max MHz:                     5500.0000
CPU min MHz:                     800.0000
BogoMIPS:                        6604.80
Virtualization:                  VT-x
L1d cache:                       192 KiB
L1i cache:                       128 KiB
```

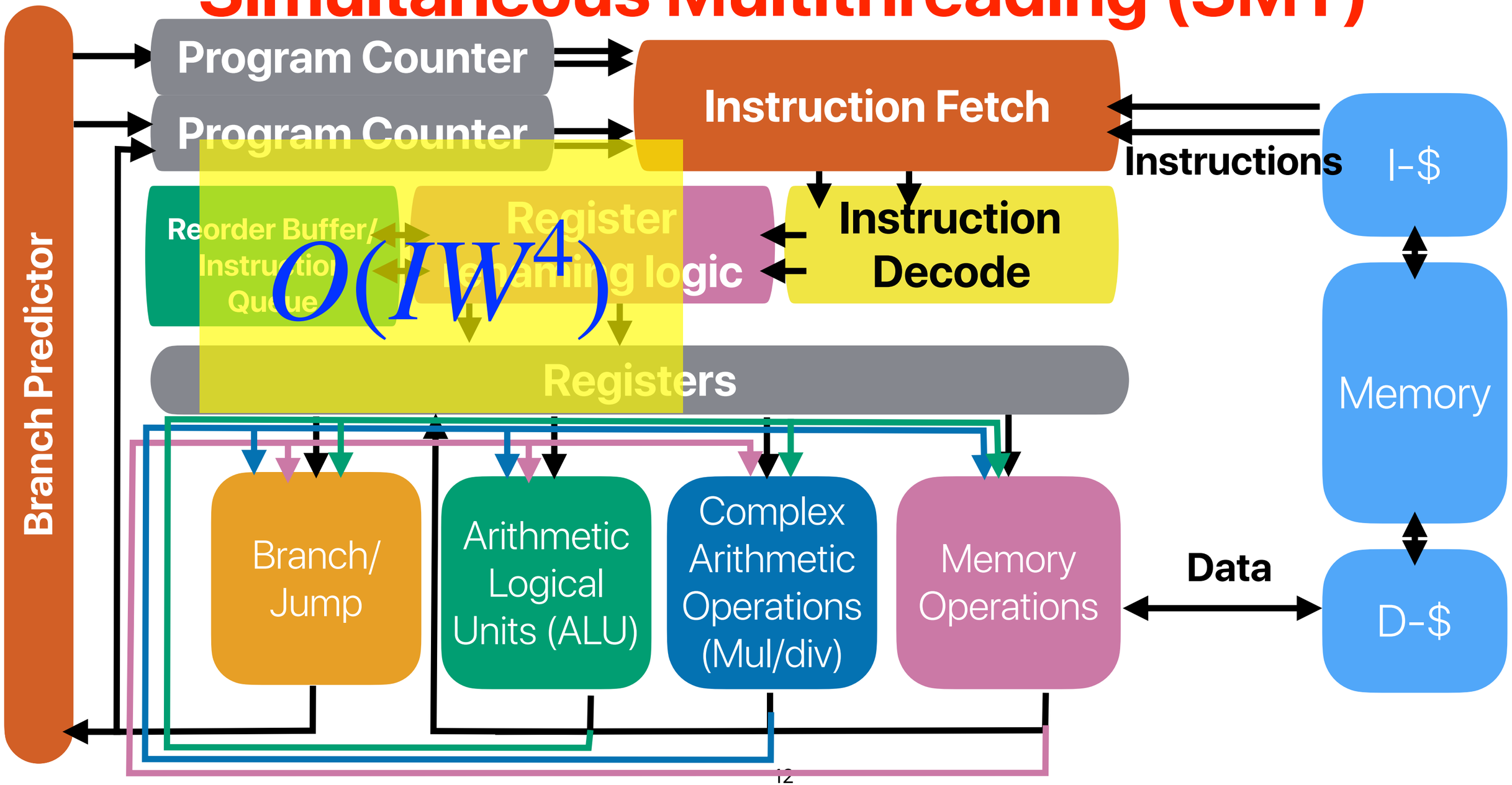
SMT

- Improve the throughput of execution
  - May increase the latency of a single thread
- Less branch penalty per thread
- Increase hardware utilization
- Simple hardware design: Only need to duplicate PC/Register Files
- Real Case:
  - Intel HyperThreading (supports up to two threads per core)
    - Intel Pentium 4, Intel Atom, Intel Core i7
  - AMD RyZen (Zen microarchitecture)
  - If you see a processor with “threads” more than “cores”, that must because of SMT!

Outline

- Multi-core processors (Chip multiprocessors)
- Programming on parallel, multithreaded architectures

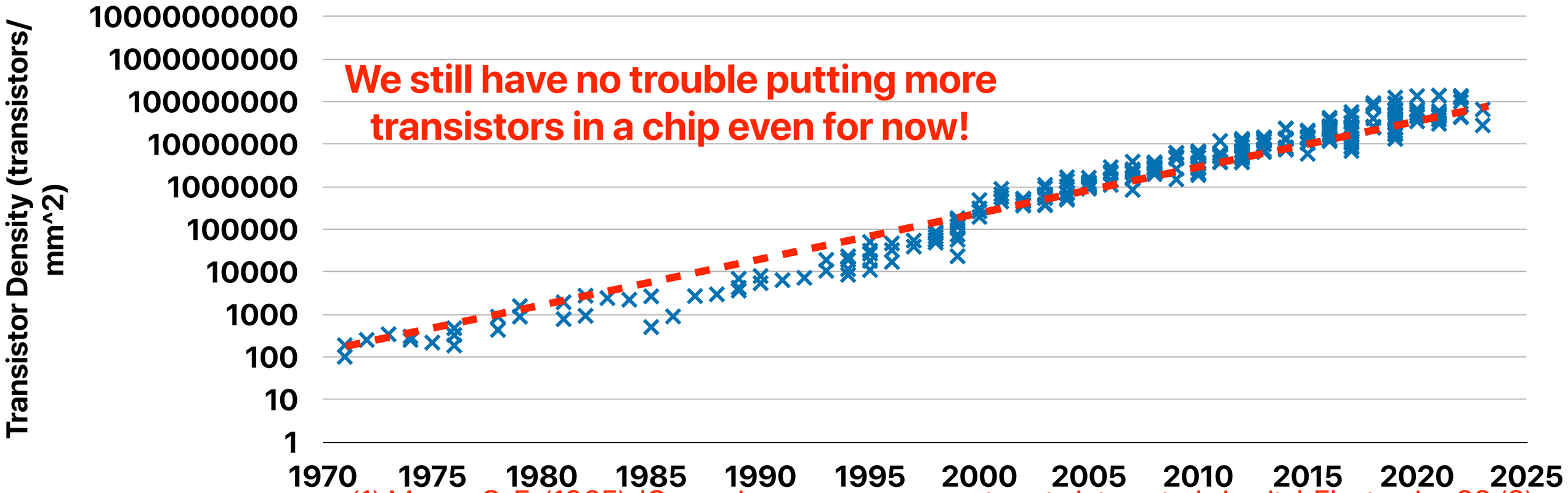
Simultaneous Multithreading (SMT)



Chip-Multiprocessors (CMP) or Multi-core processors

Recap: Moore's Law(1)

- The number of transistors we can build in a fixed area of silicon doubles every 12 ~ 24 months.
- Moore's Law "was" the most important driver for historic CPU performance gains

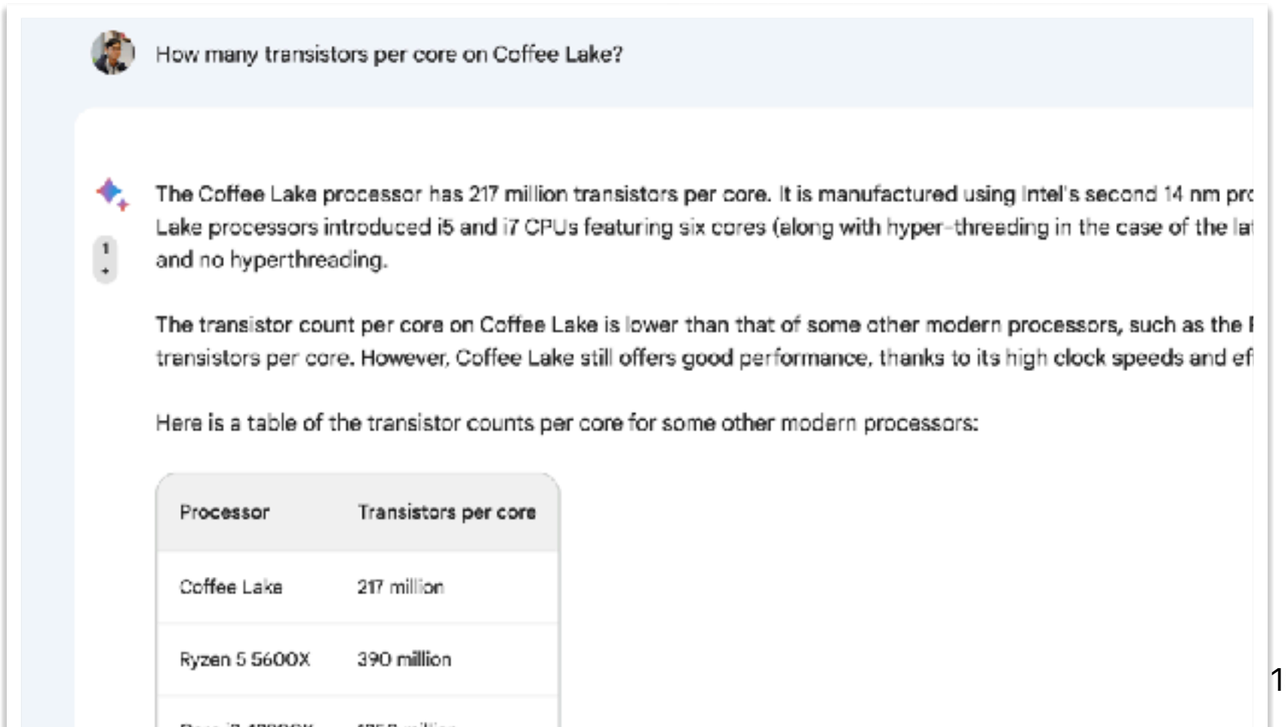


(1) Moore, G. E. (1965), 'Cramming more components onto integrated circuits', Electronics 38 (8) .

Plot based on https://en.wikipedia.org/wiki/Transistor_count by Hung-Wei Tseng

Transistor Counts

Microarchitecture	Transistor Count	Issue-width	Year
Alder Lake	325 M	5x ALU, 7x Memory	2021
Coffee Lake	217 M	4x ALU, 4x Memory	2017
Sandy Bridge	290 M	3x ALU, 3x Memory	2011
Nehalem	182.75 M	3x ALU, 3x Memory	2008



Nehalem Alder Lake
6-issue 12-issue

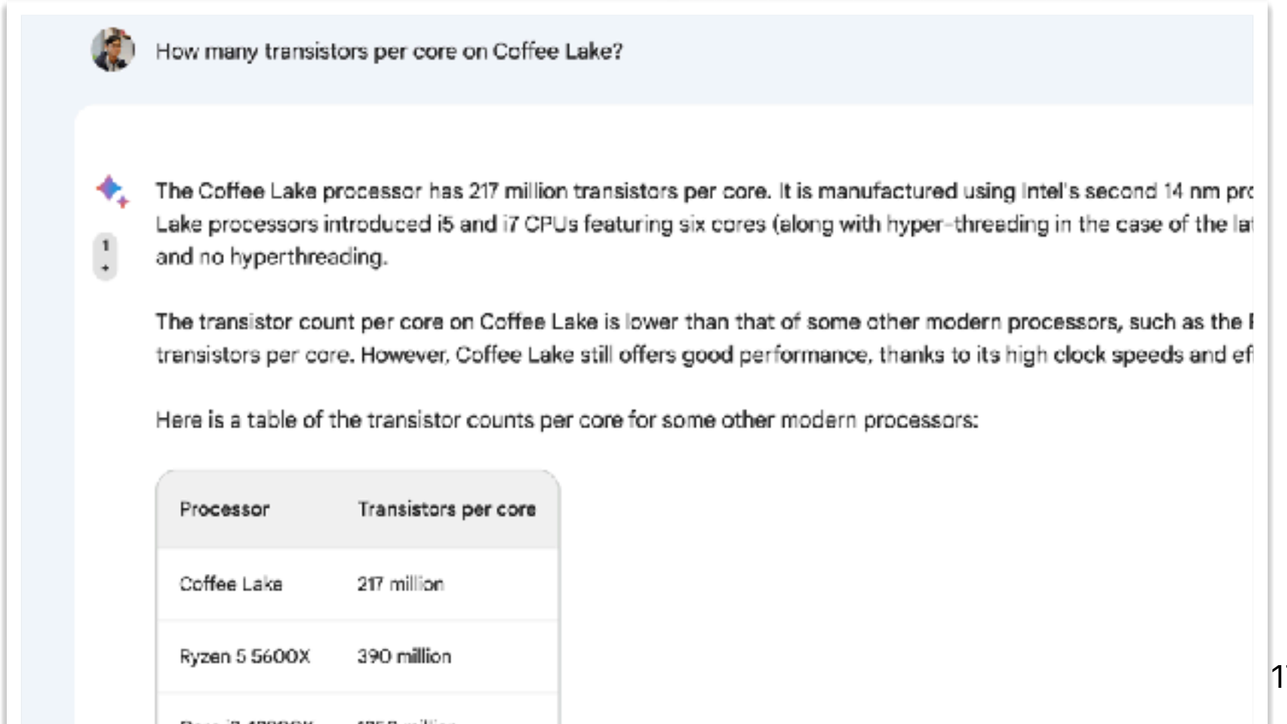
Based on https://en.wikipedia.org/wiki/Transistor_count

Recap: do we really need very wide issue processors?

	ET	IC	IPC/ILP	# of branches	Branch mis-prediction rate
A	22.21	332 Trillions	2.88	65 Trillions	1.13%
B	12.29	287 Trillions	4.52	17 Trillions	0.04%
C	5.01	102 Trillions	3.95	17 Trillions	0.04%
D	3.73	80 Trillions	4.13	1 Trillions	~0%
E	54.4	173 Trillions	0.61	44 Trillions	18.6%
SSE4.2	1.57	22 Trillions	2.7	1 Trillions	~0%

We don’t we build processors with more cores?

Microarchitecture	Transistor Count	Issue-width	Year
Alder Lake	325 M	5x ALU, 7x Memory	2021
Coffee Lake	217 M	4x ALU, 4x Memory	2017
Sandy Bridge	290 M	3x ALU, 3x Memory	2011
Nehalem	182.75 M	3x ALU, 3x Memory	2008



2x 3-issue ALUs Nehalem

Nehalem 6-issue | Alder Lake 12-issue | Nehalem 6-issue

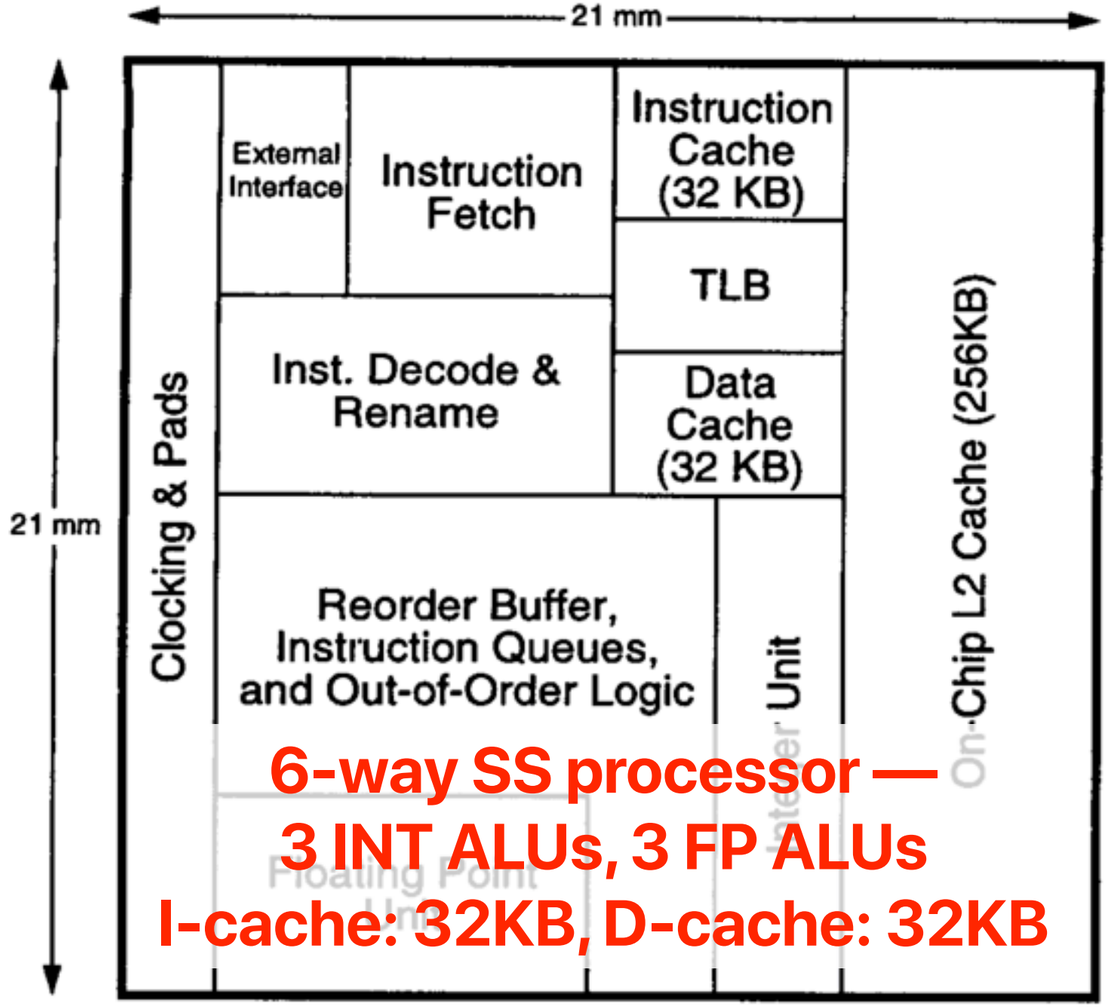
1x 5-issue ALUs Alder Lake

Based on https://en.wikipedia.org/wiki/Transistor_count

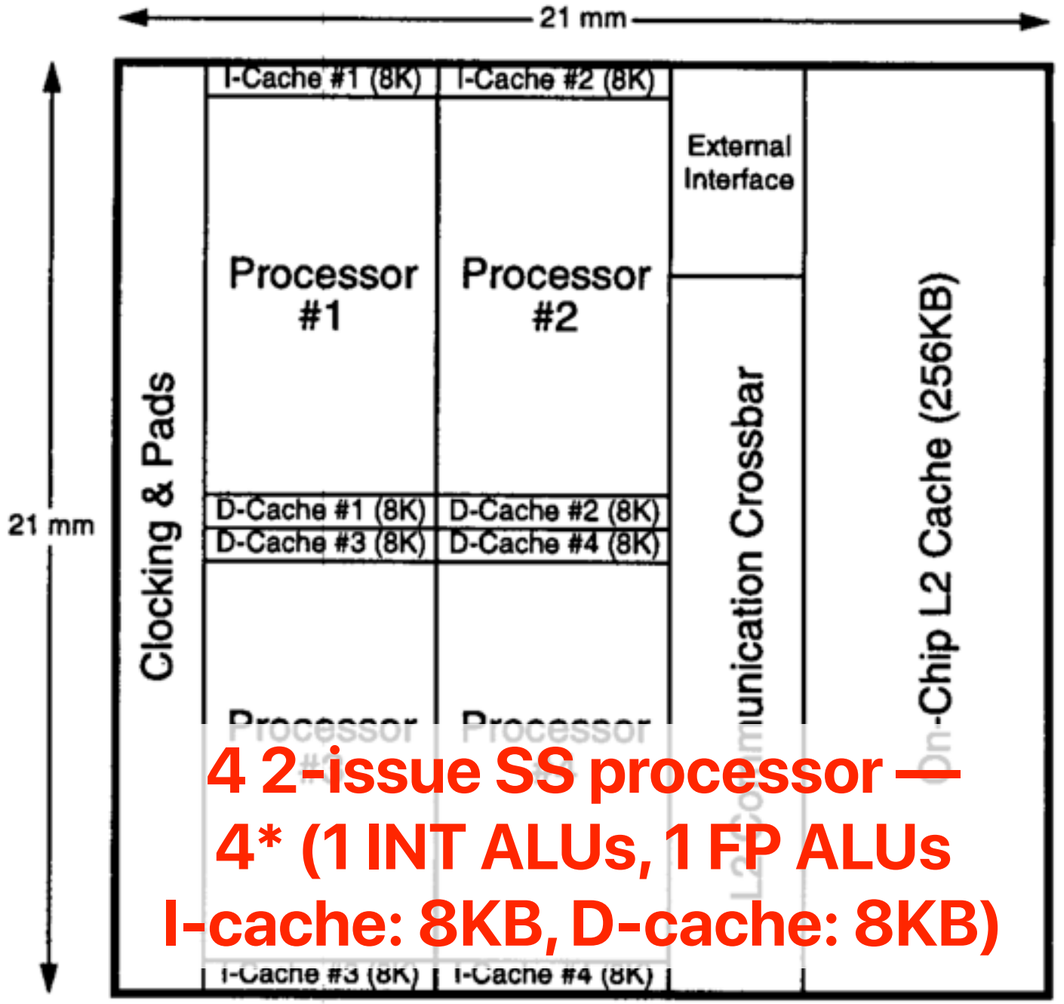
The case for a Single-Chip Multiprocessor

Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang
Stanford University

Wide-issue SS processor v.s. multiple narrower-issue SS processors

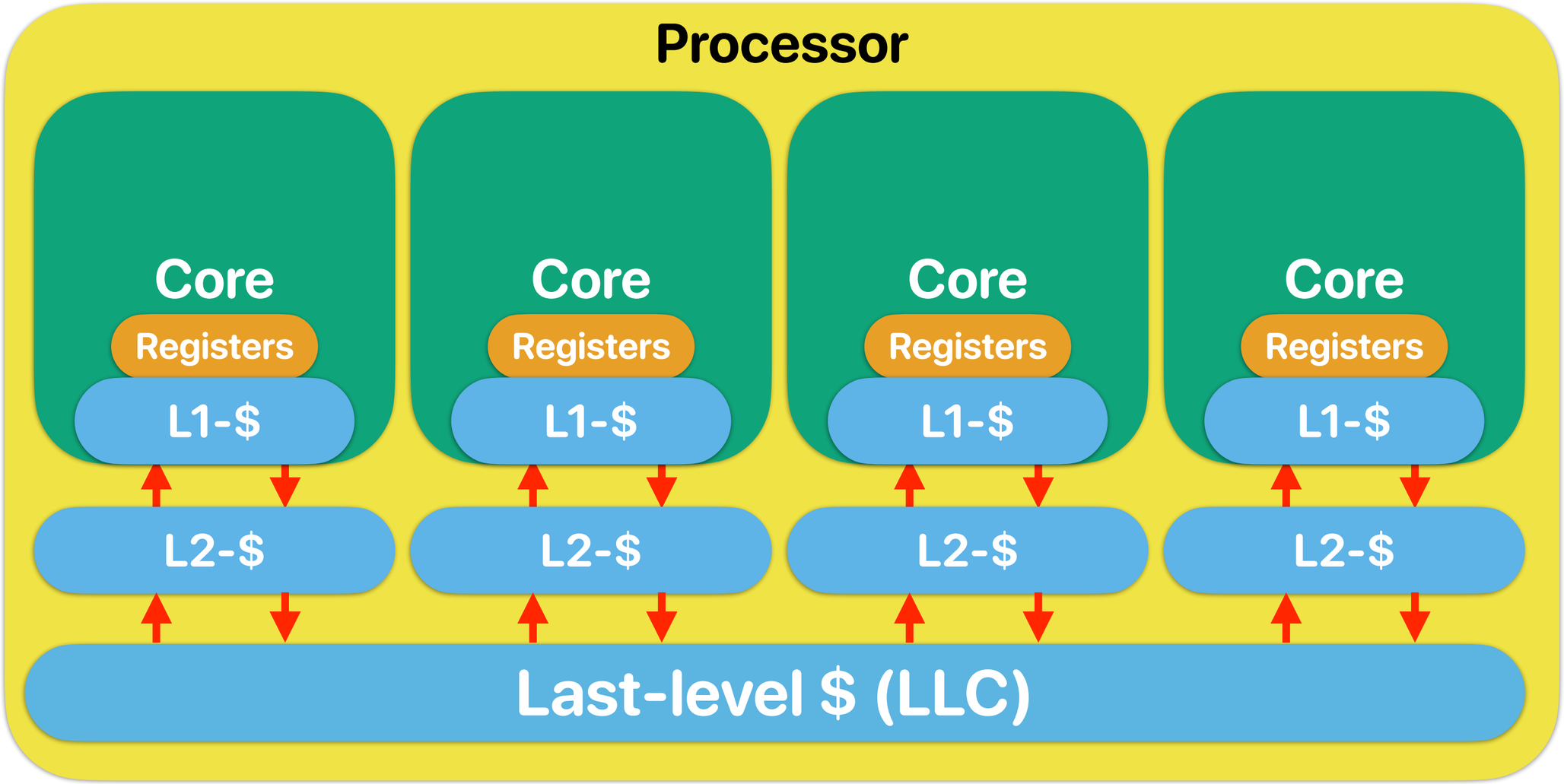


6-way SS processor —
3 INT ALUs, 3 FP ALUs
I-cache: 32KB, D-cache: 32KB

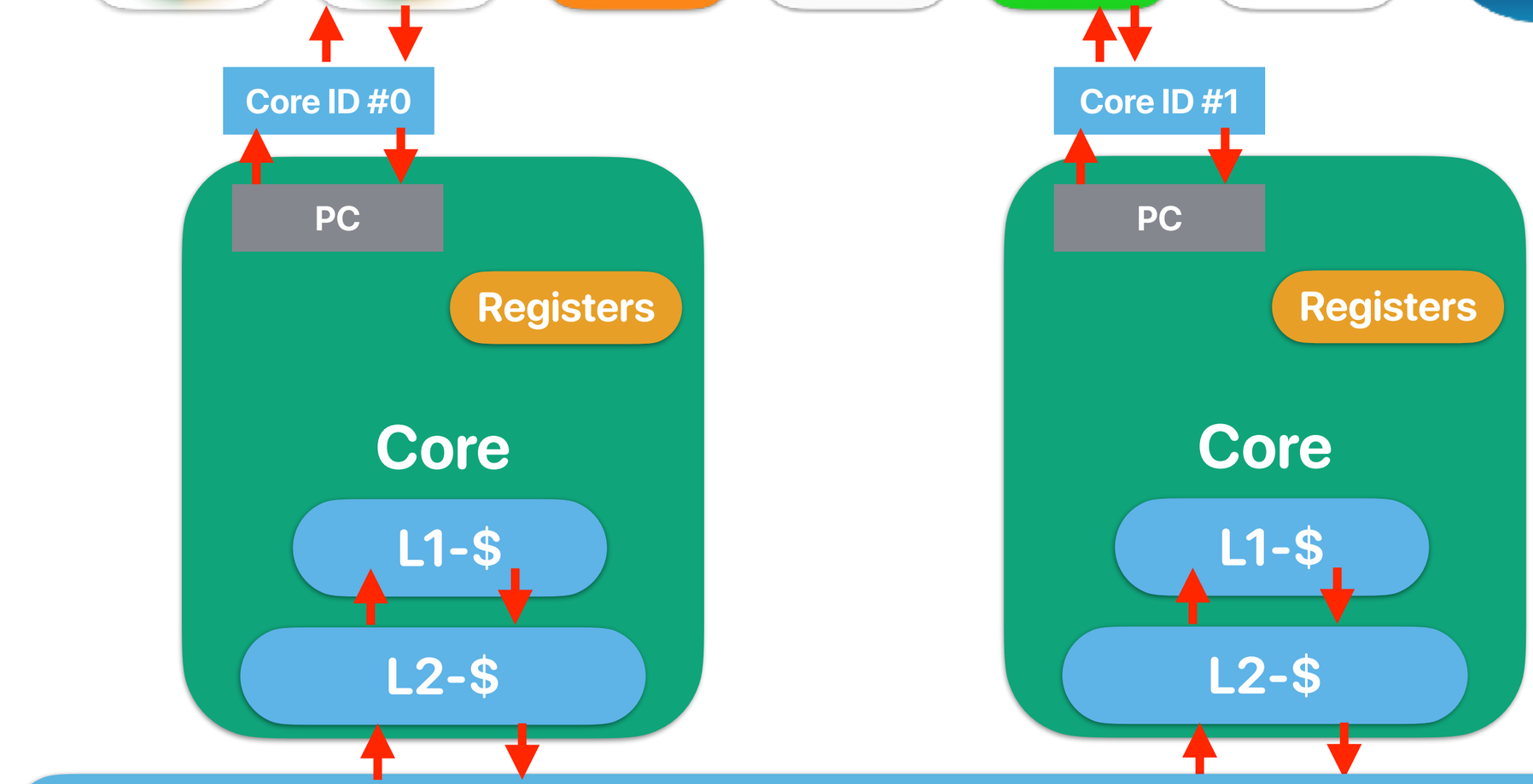


4 2-issue SS processor —
4* (1 INT ALUs, 1 FP ALUs
I-cache: 8KB, D-cache: 8KB)

Concept of CMP



CMP from the user/OS' perspective



https://www.pollev.com/hungweitseng close in 1:30

SMT v.s. CMP

- An SMT processor is basically a SuperScalar processor with multiple instruction front-end. Assume within the same chip area, we can build an SMT processor supporting 4 threads, with 6-issue pipeline, 64KB cache or — a CMP with 4x 2-issue pipeline & 16KB cache in each core. Please identify how many of the following statements are/is correct when running programs on these processors.
  - ① If we are just running one program in the system, the program will perform better on an SMT processor
  - ② If we are running 4 applications simultaneously, the cache miss rates will be higher in the SMT processor
  - ③ If we are running 4 applications simultaneously, the branch mis-prediction will be higher in the SMT processor
  - ④ If we are running one program with 4 parallel threads, the cache miss rates will be higher in the SMT processor
  - ⑤ If we are running one program with 4 parallel threads simultaneously, the branch mis-prediction will be longer in the SMT processor

  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

SMT v.s. CMP

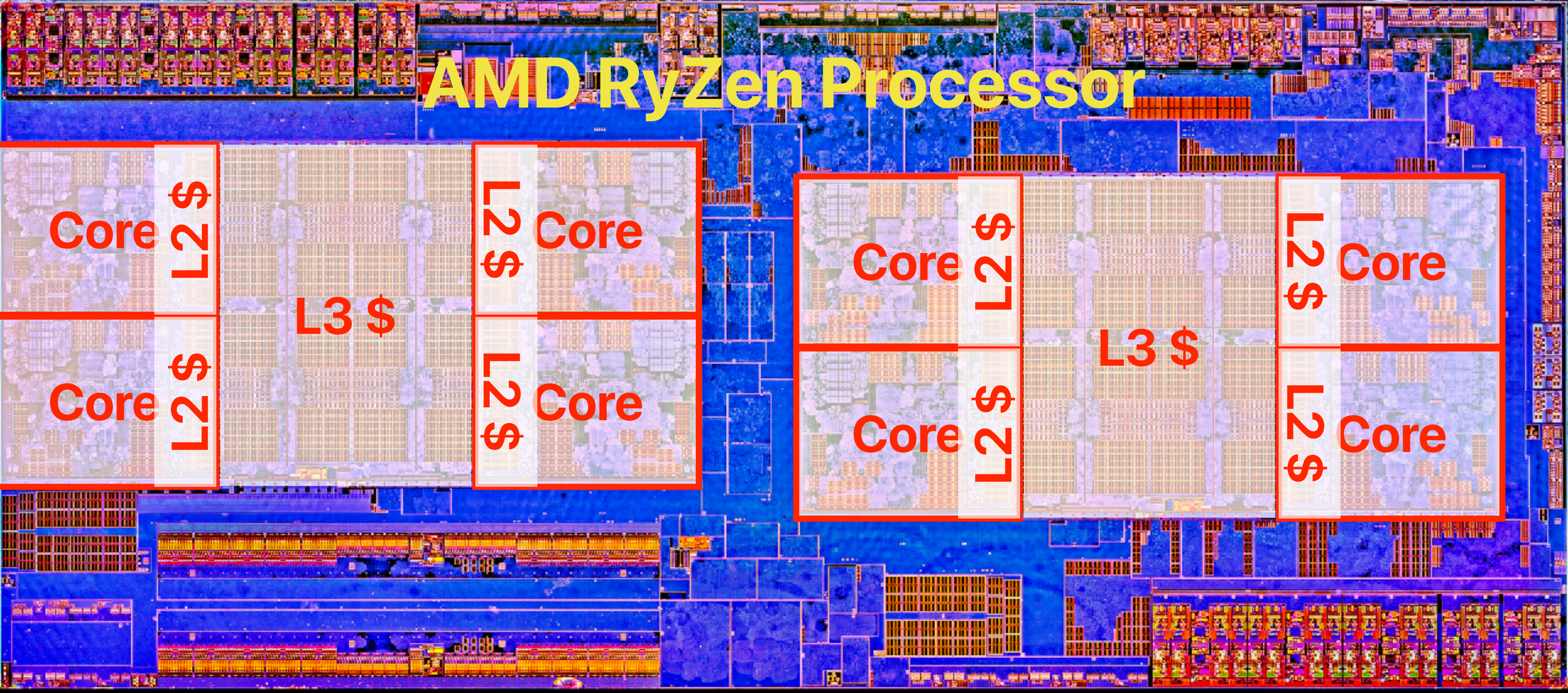
- An SMT processor is basically a SuperScalar processor with multiple instruction front-end. Assume within the same chip area, we can build an SMT processor supporting 4 threads, with 6-issue pipeline, 64KB cache or — a CMP with 4x 2-issue pipeline & 16KB cache in each core. Please identify how many of the following statements are/is correct when running programs on these processors.
  - ① If we are just running one program in the system, the program will perform better on an SMT processor — you have more resources for the program
  - ② If we are running 4 applications simultaneously, the cache miss rates will be higher in the SMT processor — it depends!
  - ③ If we are running 4 applications simultaneously, the branch mis-prediction will be higher in the SMT processor — it depends!
  - ④ If we are running one program with 4 parallel threads, the cache miss rates will be higher in the SMT processor — it depends!
  - ⑤ If we are running one program with 4 parallel threads simultaneously, the branch mis-prediction will be longer in the SMT processor — it depends!

  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

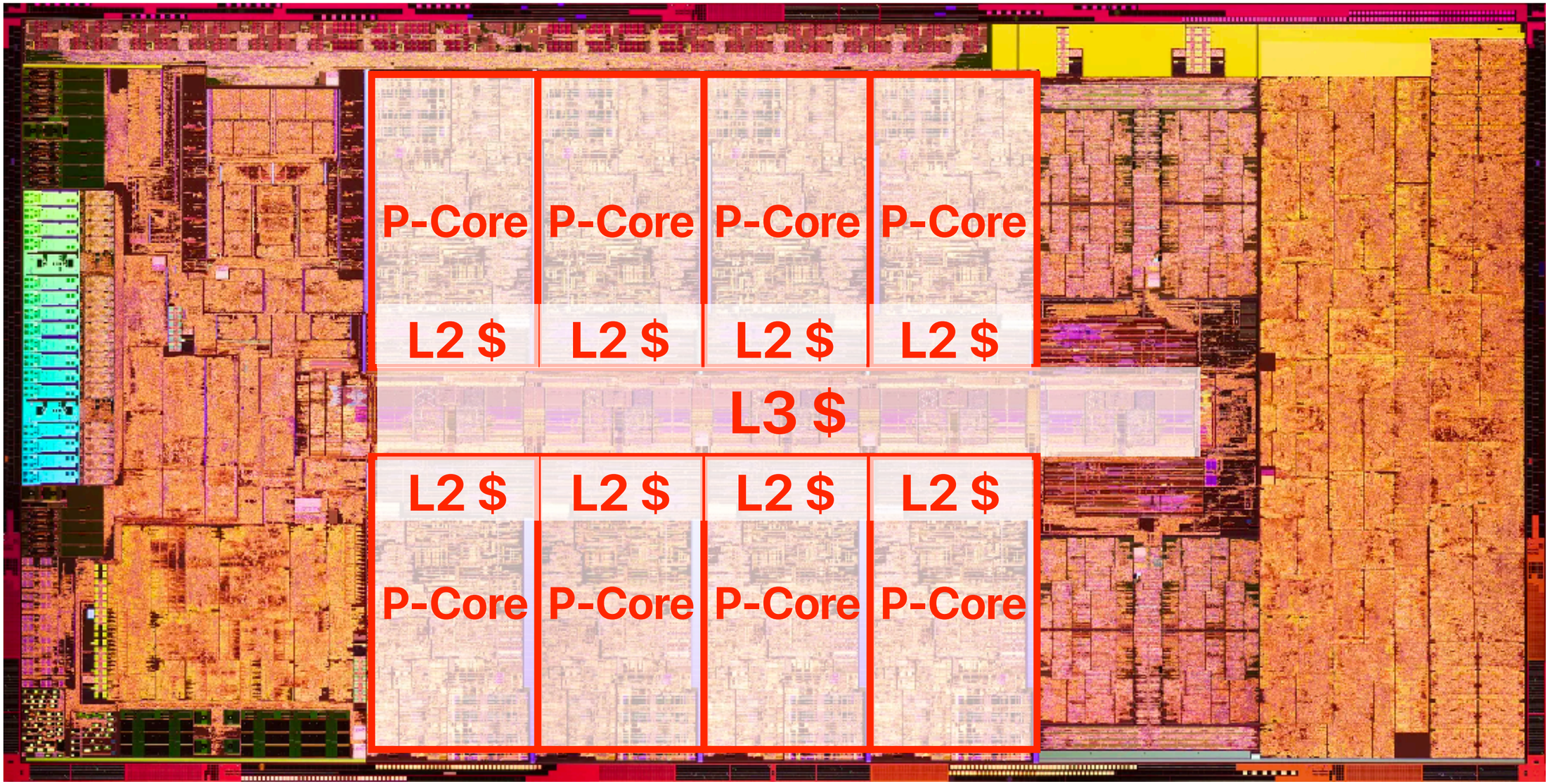
There is no clear win on each — why not having both?

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           151
Model name:                      12th Gen Intel(R) Core(TM) i3-12100F
Stepping:                        5
CPU MHz:                         3300.000
CPU max MHz:                     5500.0000
CPU min MHz:                     800.0000
BogoMIPS:                        6604.80
Virtualization:                  VT-x
L1d cache:                       192 KiB
L1i cache:                       128 KiB
```

Modern processors have both CMP/SMT


AMD RyZen Processor
Core
L2 $
L3 $
Core
L2 $
Core
L2 $
Core
L2 $
Core
L2 $
L3 $
Core
L2 $
Core
L2 $
AMD
RYZEN

Intel Alder Lake



SMT v.s. CMP

- An SMT processor is basically a SuperScalar processor with multiple instruction front-end. Assume within the same chip area, we can build an SMT processor supporting 4 threads, with 6-issue pipeline, 64KB cache or — a CMP with 4x 2-issue pipeline & 16KB cache in each core. Please identify how many of the following statements are/is correct when running programs on these processors.
  - ① If we are just running one program in the system, the program will perform better on an SMT processor — you have more resources for the program
  - ② If we are running 4 applications simultaneously, the cache miss rates will be higher in the SMT processor — it depends!
  - ③ If we are running 4 applications simultaneously, the branch mis-prediction will be higher in the SMT processor — it depends!
  - ④ If we are running one program with 4 parallel threads, the cache miss rates will be higher in the SMT processor — it depends!
  - ⑤ If we are running one program with 4 parallel threads simultaneously, the branch mis-prediction will be longer in the SMT processor — it depends!

  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

There is no clear win on each — why not having both?

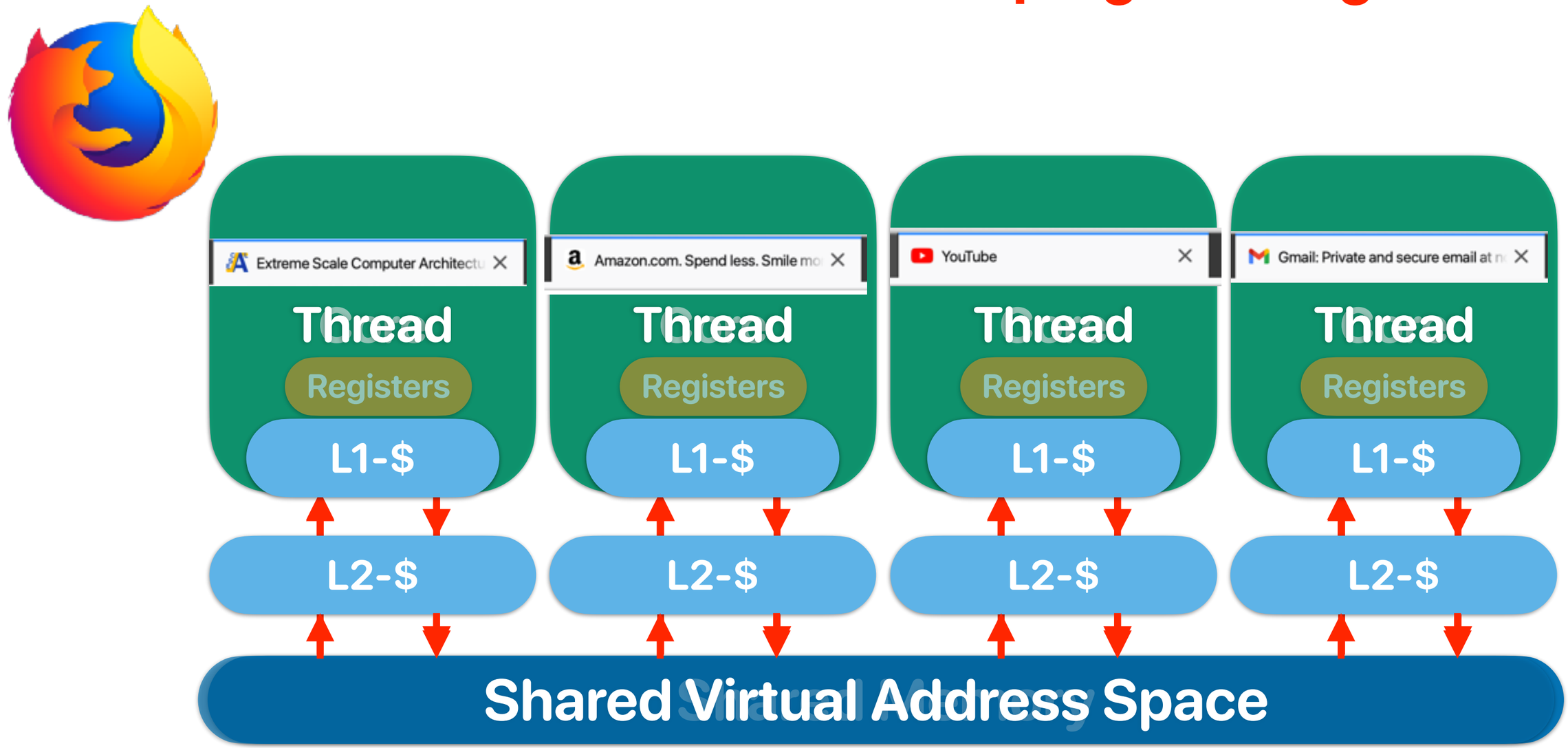
The only thing we know for sure — if we don't parallel the program, it won't get any faster on SMT/CMP

Parallel Programming & Architectural Supports for Parallel Programming

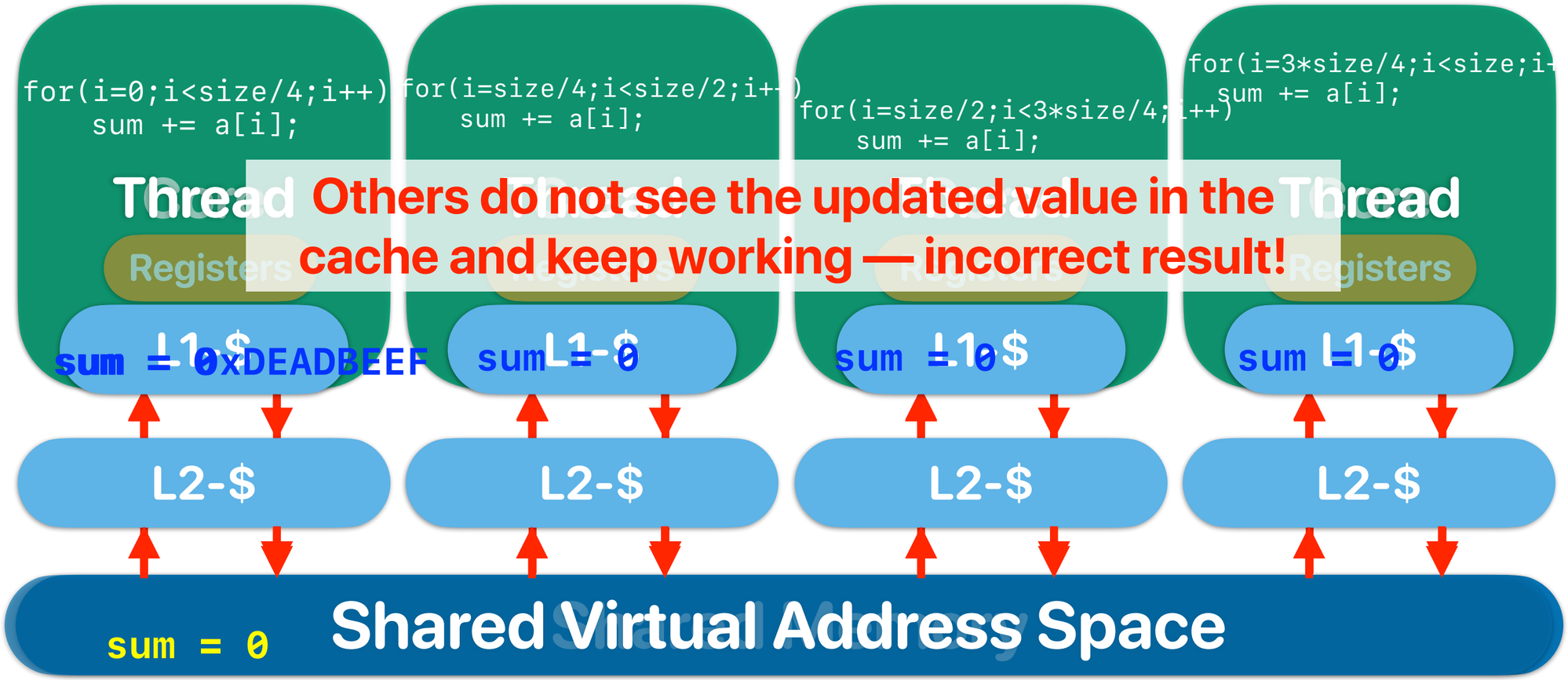
Parallel programming

- To exploit parallelism you need to break your computation into multiple “processes” or multiple “threads”
- Processes (in OS/software systems)
  - Separate programs actually running (not sitting idle) on your computer at the same time.
  - Each process will have its own virtual memory space and you need explicitly exchange data using inter-process communication APIs
  - Programming model: MPI, Spark
- Threads (in OS/software systems)
  - Independent portions of your program that can run in parallel
  - All threads share the same virtual memory space
  - Programming model: pthread or openmp
- We will refer to these collectively as “threads”
  - A typical user system might have 1-8 actively running threads.
  - Servers can have more if needed (the sysadmins will hopefully configure it that way)

What software thinks about “multiprogramming” hardware



What software thinks about “multiprogramming” hardware



Others do not see the updated value in the cache and keep working — incorrect result!

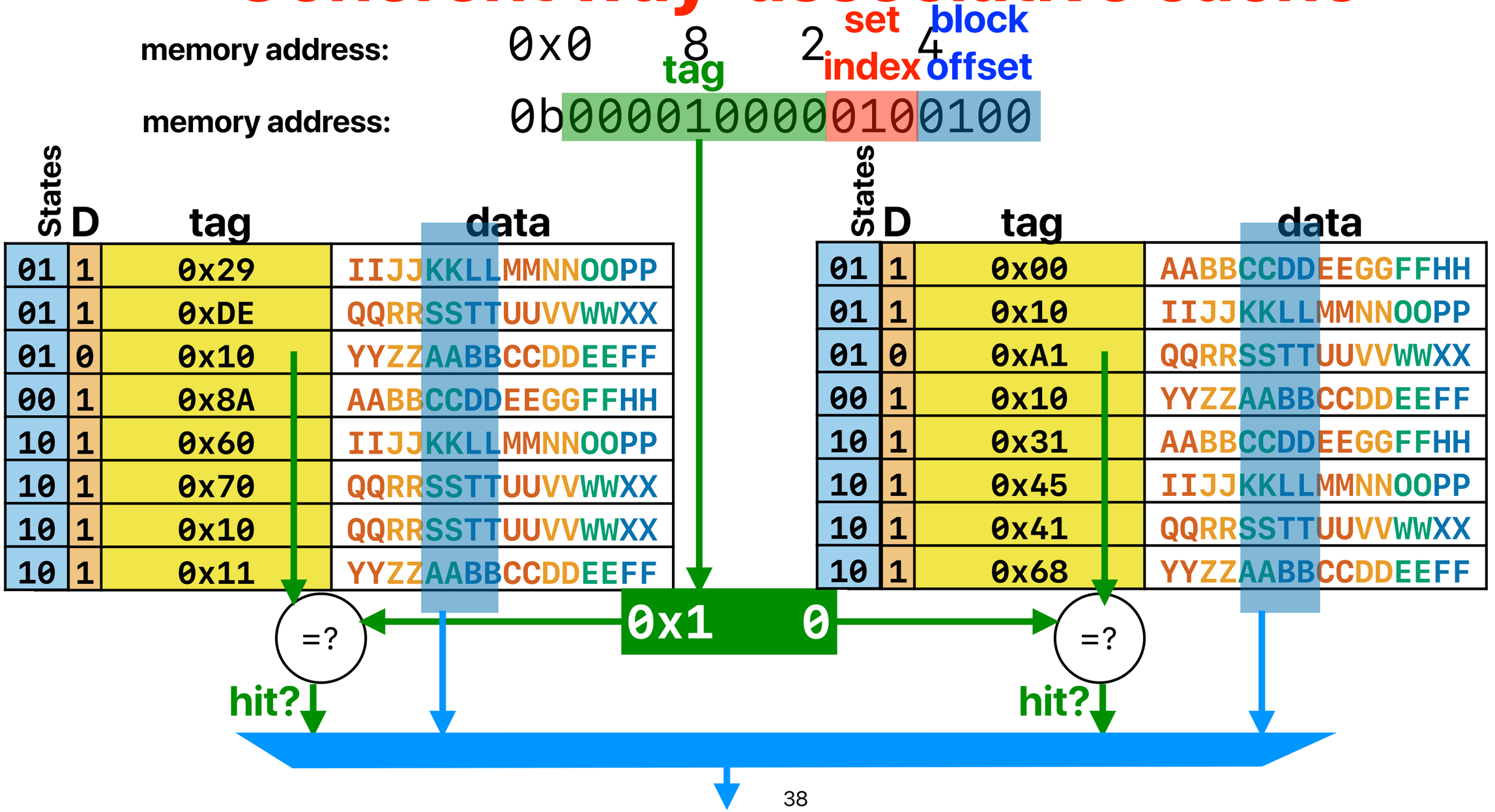
Coherency & Consistency

- Coherency — Guarantees all processors see the same value for a variable/memory address in the system when the processors need the value at the same time
  - What value should be seen
- Consistency — All threads see the change of data in the same order
  - When the memory operation should be done

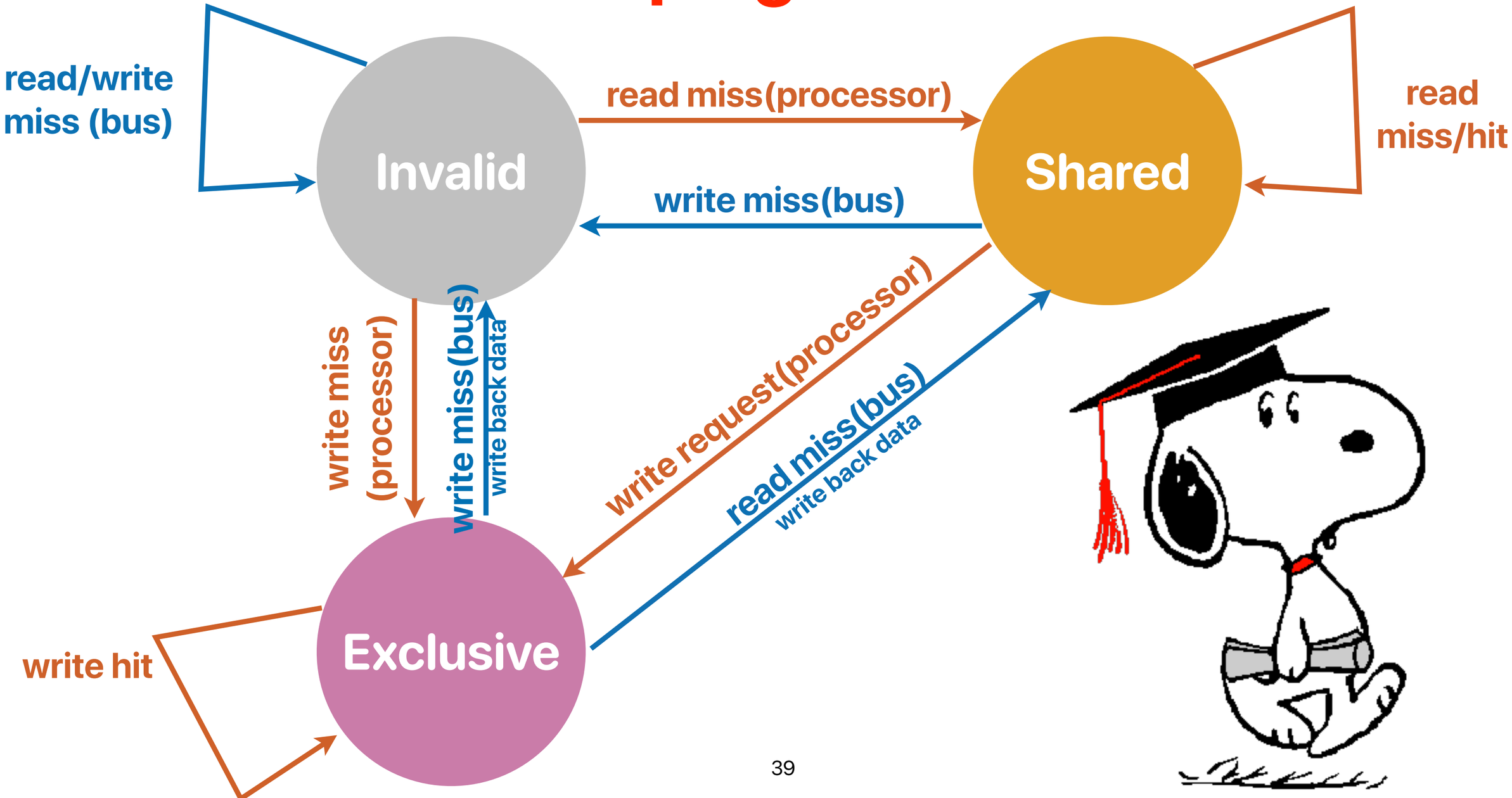
Simple cache coherency protocol

- Snooping protocol
  - Each processor broadcasts / listens to cache misses
- State associate with each block (cacheline)
  - Invalid
    - The data in the current block is invalid
  - Shared
    - The processor can read the data
    - The data may also exist on other processors
  - Exclusive
    - The processor has full permission on the data
    - The processor is the only one that has up-to-date data

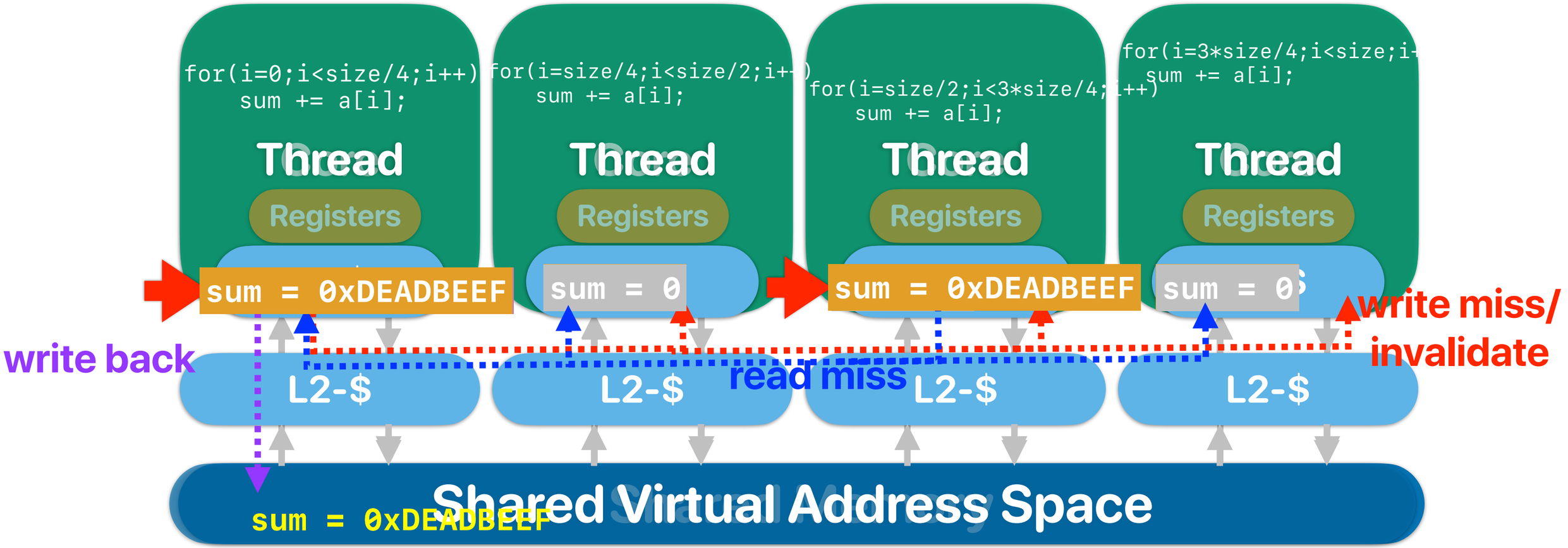
Coherent way-associative cache



Snooping Protocol



What happens when we write in coherent caches?



Observer

thread 1

```
int loop;

int main()
{
  pthread_t thread;
  loop = 1;

  pthread_create(&thread, NULL,
modifyloop, NULL);
  while(loop == 1)
  {
    continue;
  }
  pthread_join(thread, NULL);
  fprintf(stderr,"User input: %d\n",
```

thread 2

```
void* modifyloop(void *x)
{
  sleep(1);
  printf("Please input a number:\n");
  scanf("%d",&loop);
  return NULL;
}
```

Observer

prevents the compiler from putting the variable “loop" in the “register”

thread 1

```
volatile int loop;

int main()
{
  pthread_t thread;
  loop = 1;

  pthread_create(&thread, NULL,
modifyloop, NULL);
  while(loop == 1)
  {
    continue;
  }
  pthread_join(thread, NULL);
  fprintf(stderr,"User input: %d\n",
```

thread 2

```
void* modifyloop(void *x)
{
  sleep(1);
  printf("Please input a number:\n");
  scanf("%d",&loop);
  return NULL;
}
```

Cache coherency

- Assuming that we are running the following code on a CMP with a cache coherency protocol, how many of the following outputs are possible? (a is initialized to 0 as assume we will output more than 10 numbers)

thread 1	thread 2
`while(1)` `  printf("%d ",a);`	`while(1)` `  a++;`

① 0 1 2 3 4 5 6 7 8 9
② 1 2 5 9 3 6 8 10 12 13
③ 1 1 1 1 1 1 1 64 100
④ 1 1 1 1 1 1 1 1 1 100

A. 0
B. 1
C. 2
D. 3
E. 4

Cache coherency

- Assuming that we are running the following code on a CMP with a cache coherency protocol, how many of the following outputs are possible? (a is initialized to 0 as assume we will output more than 10 numbers)

thread 1	thread 2
`while(1)` `  printf("%d ",a);`	`while(1)` `  a++;`

① 0 1 2 3 4 5 6 7 8 9

② 1 2 5 9 3 6 8 10 12 13

③ 1 1 1 1 1 1 1 64 100

④ 1 1 1 1 1 1 1 1 1 100

A. 0

B. 1

C. 2

D. 3

E. 4

Cache coherency

- Assuming that we are running the following code on a CMP with a cache coherency protocol, how many of the following outputs are possible? (a is initialized to 0 as assume we will output more than 10 numbers)

thread 1	thread 2
`while(1)` `  printf("%d ",a);`	`while(1)` `  a++;`

① 0 1 2 3 4 5 6 7 8 9

② 1 2 5 9 3 6 8 10 12 13

③ 1 1 1 1 1 1 1 64 100

④ 1 1 1 1 1 1 1 1 1 100

A. 0

B. 1

C. 2

D. 3

E. 4

Take-aways: parallel programming

- Cache coherency only guarantees that everyone would eventually have a coherent view of data, but not when

Announcements

- **Reading Quiz 8 (last reading quiz)** due **next Tuesday** before the lecture
- **iEVAL** started and ends on 6/06/2024
  - Submit the prove of your participation in iEVAL through Gradescope
  - It can become a full credit reading quiz (it helps to amortize the penalty of another least performing one)
- **Assignment 5** is **released** due 6/09/2024
  - The same programming assignment as Assignment 3 but you need to speedup by 4x on Gradescope this time
- **Final exam**
  - 6/14 8a-11a @ **BOYHL 1471**
  - Closed book, no cheatsheet — the same rules as the midterm
  - Two questions can be used as CSMS comprehensive examine questions — one is memory-hierarchy related, and the other is OoO scheduling and code optimization

Computer Science & Engineering

203

つづく