Parallel Architectures & Programming on Parallel Architectures (1)

Hung-Wei Tseng

Recap: Characteristics of modern processor architectures

- SuperScalar
  - Multiple-issue pipelines with multiple functional units available
  - Multiple instructions can execute in the same pipeline
- Register renaming + Reorder Buffer (ROB) — enable dynamic OoO scheduling to issue/execute instructions whenever possible
- Cache
- Branch 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

Recap: ILP within a program

- perf is a tool that captures performance counters of your processors and can generate results like branch mis-prediction rate, cache miss rates and ILP.

Wider-issue processors won't give you much more

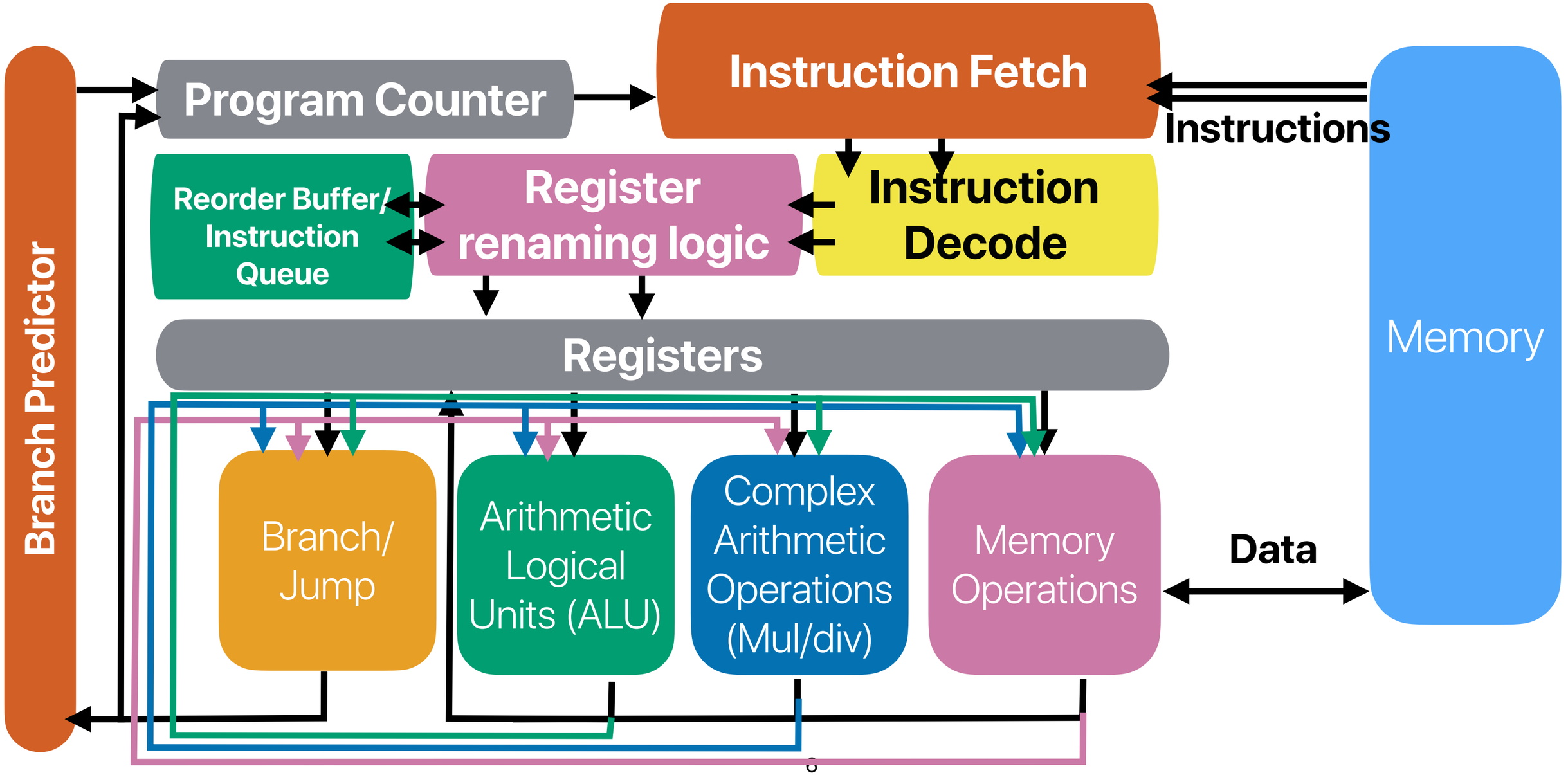
Program	IPC	BP Rate %	I cache %MPCI	D cache %MPCI	L2 cache %MPCI
compress	0.9	85.9	0.0	3.5	1.0
eqntott	1.3	79.8	0.0	0.8	0.7
m88ksim	1.4	91.7	2.2	0.4	0.0
MPsim	0.8	78.7	5.1	2.3	2.3
applu	0.9	79.2	0.0	2.0	1.7
apsi	0.6	95.1	1.0	4.1	2.1
swim	0.9	99.7	0.0	1.2	1.2
tomcatv	0.8	99.6	0.0	7.7	2.2
pmake	1.0	86.2	2.3	2.1	0.4

Table 5. Performance of a single 2-issue superscalar processor.

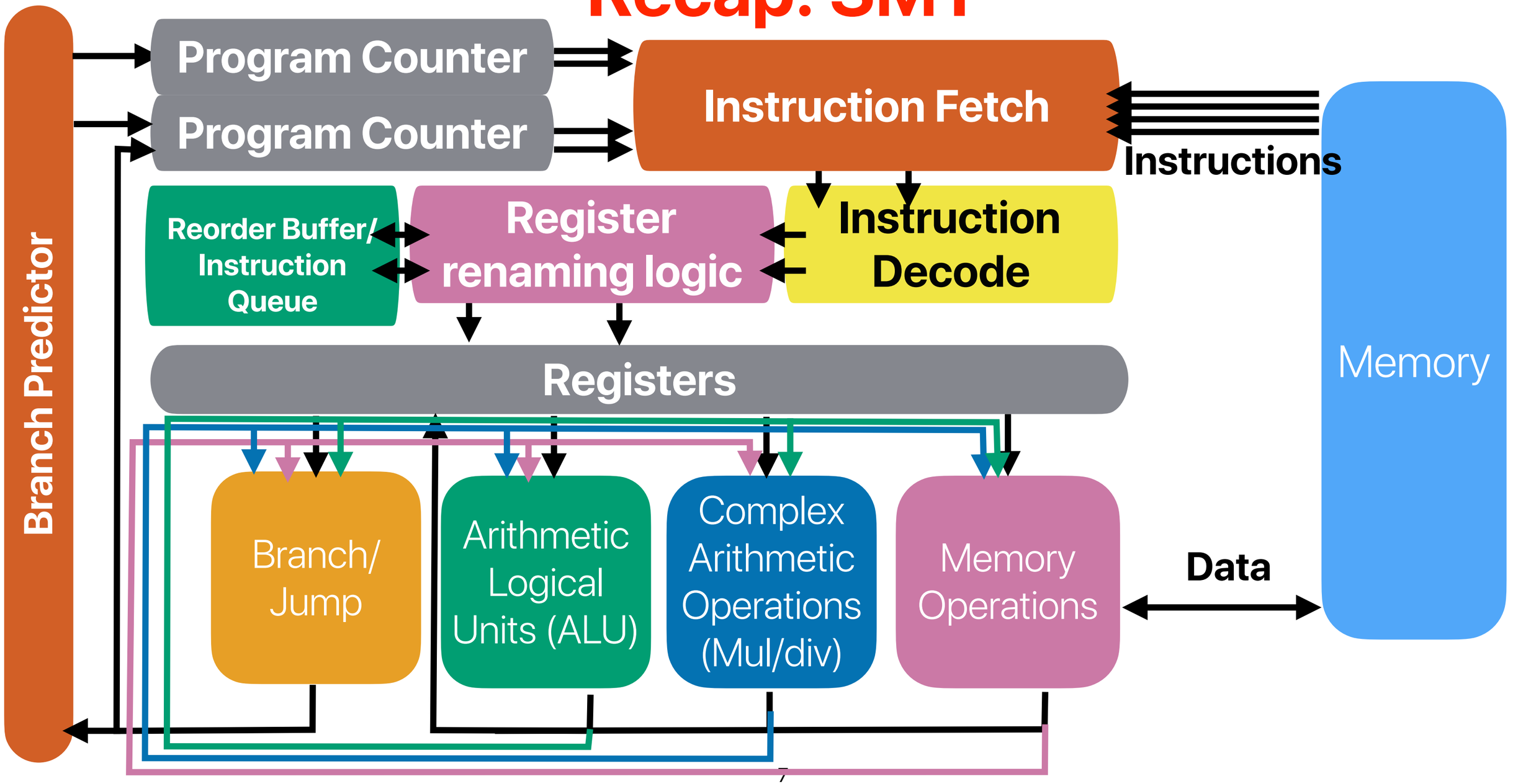
Program	IPC	BP Rate %	I cache %MPCI	D cache %MPCI	L2 cache %MPCI
compress	1.2	86.4	0.0	3.9	1.1
eqntott	1.8	80.0	0.0	1.1	1.1
m88ksim	2.3	92.6	0.1	0.0	0.0
MPsim	1.2	81.6	3.4	1.7	2.3
applu	1.7	79.7	0.0	2.8	2.8
apsi	1.2	95.6	0.2	3.1	2.6
swim	2.2	99.8	0.0	2.3	2.5
tomcatv	1.3	99.7	0.0	4.2	4.3
pmake	1.4	82.7	0.7	1.0	0.6

Table 6. Performance of the 6-issue superscalar processor.

Register renaming



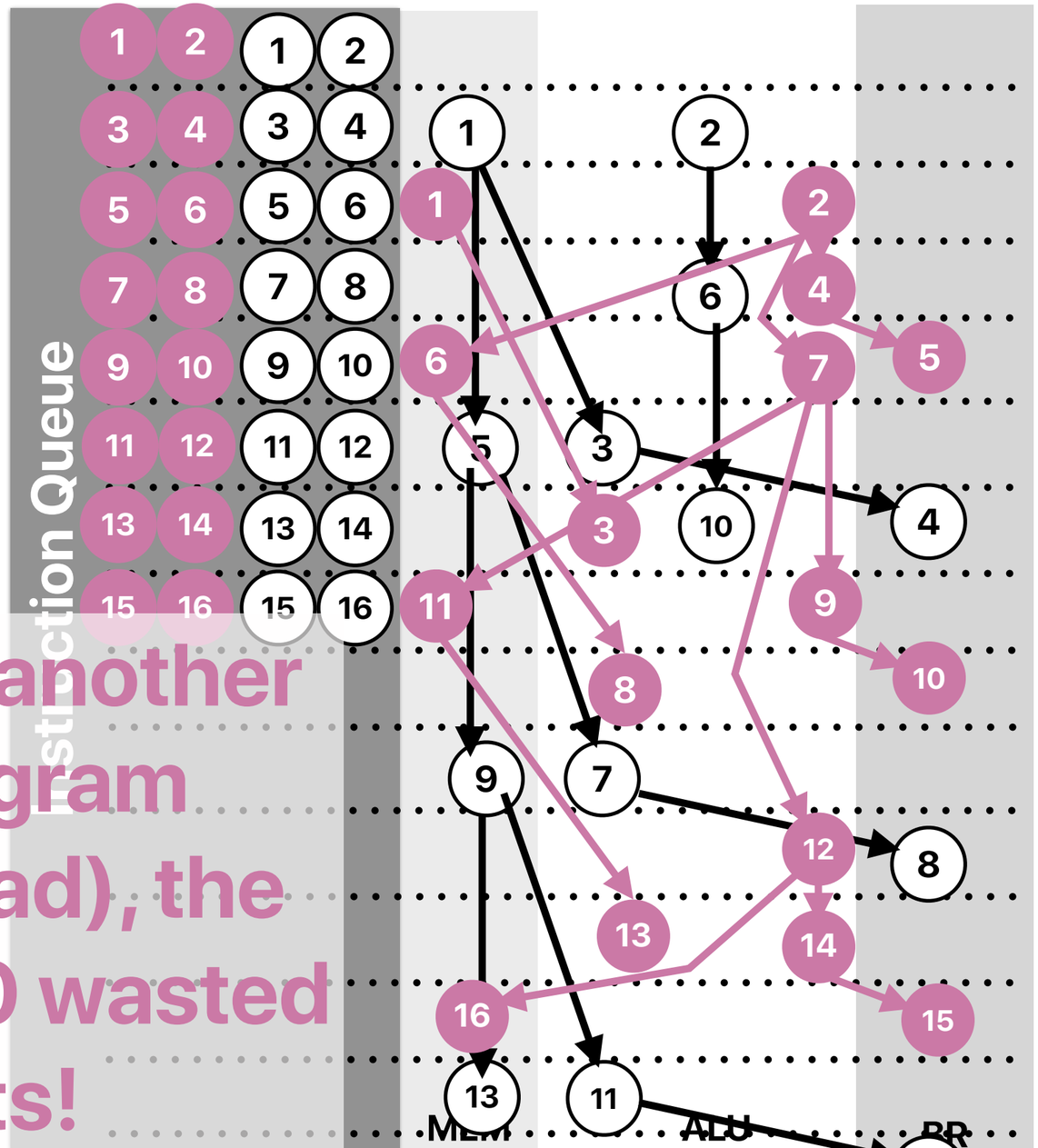
Recap: SMT


Program Counter
Program Counter
Instruction Fetch
Instructions
Branch Predictor
Reorder Buffer/ Instruction Queue
Register renaming logic
Instruction Decode
Registers
Memory
Branch/ Jump
Arithmetic Logical Units (ALU)
Complex Arithmetic Operations (Mul/div)
Memory Operations
Data

```
Architecture:                         x86_64
CPU op-mode(s):                       32-bit, 64-bit
Byte Order:                           Little Endian
Address sizes:                        48 bits physical, 48 bits virtual
CPU(s):                               16
On-line CPU(s) list:                  0-15
Thread(s) per core:                   2
Core(s) per socket:                   8
Socket(s):                            1
NUMA node(s):                         1
Vendor ID:                            AuthenticAMD
CPU family:                           25
Model:                                80
Model name:                           AMD Ryzen 7 5700G with Radeon Graphics
Stepping:                             0
```

Recap: Simultaneous Multithreading

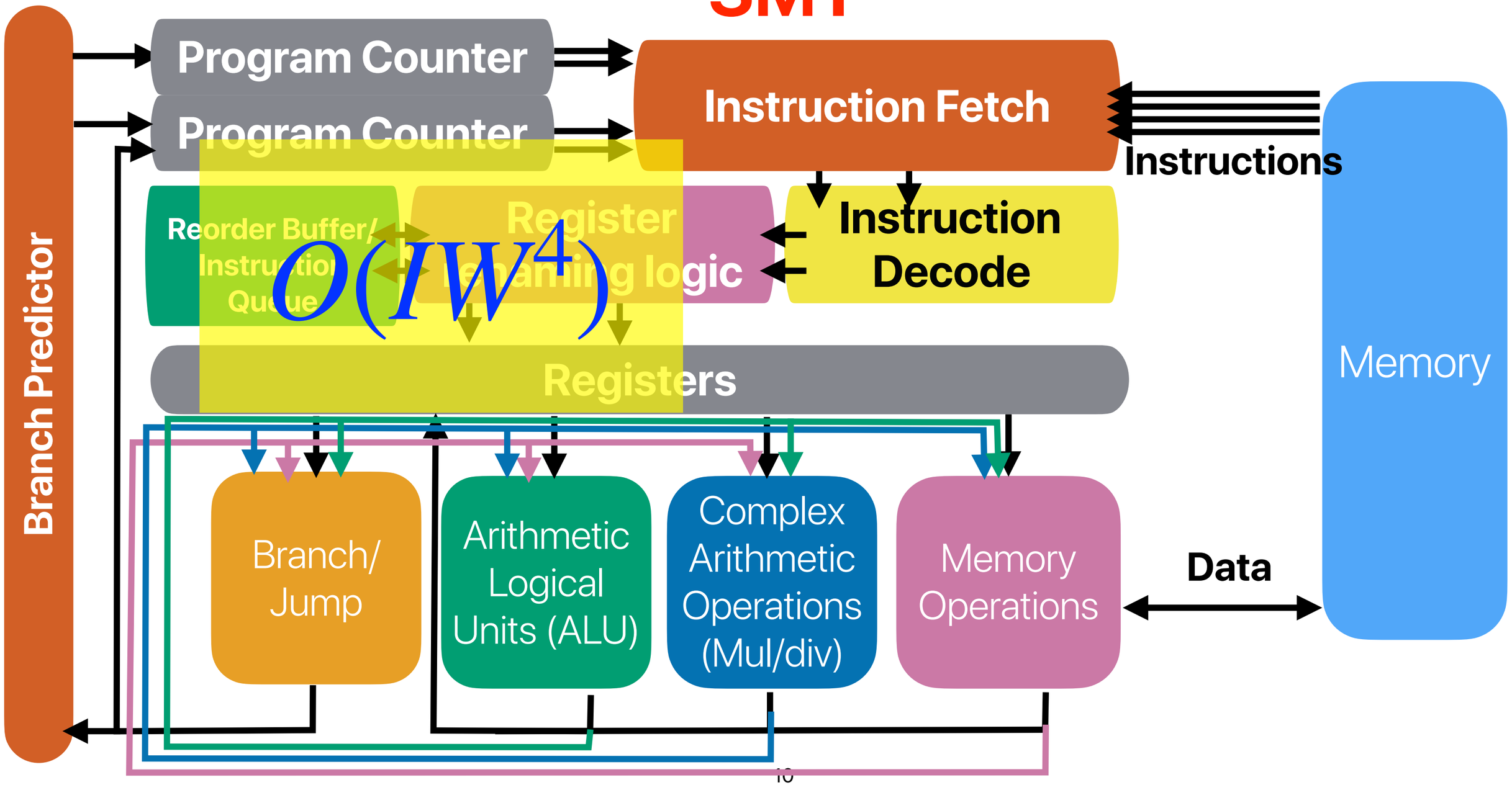
```
① movq    8(%rdi), %rdi
② addl    $1, %eax
③ testq   %rdi, %rdi
④ jne     .L3
⑤ movq    8(%rdi), %rdi
⑥ addl    $1, %eax
⑦ testq   %rdi, %rdi
⑧ jne     .L3
⑨ movq    8(%rdi), %rdi
⑩ addl    $1, %eax
⑪ testq   %rdi, %rdi
⑫ jne     .L3
```



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
```

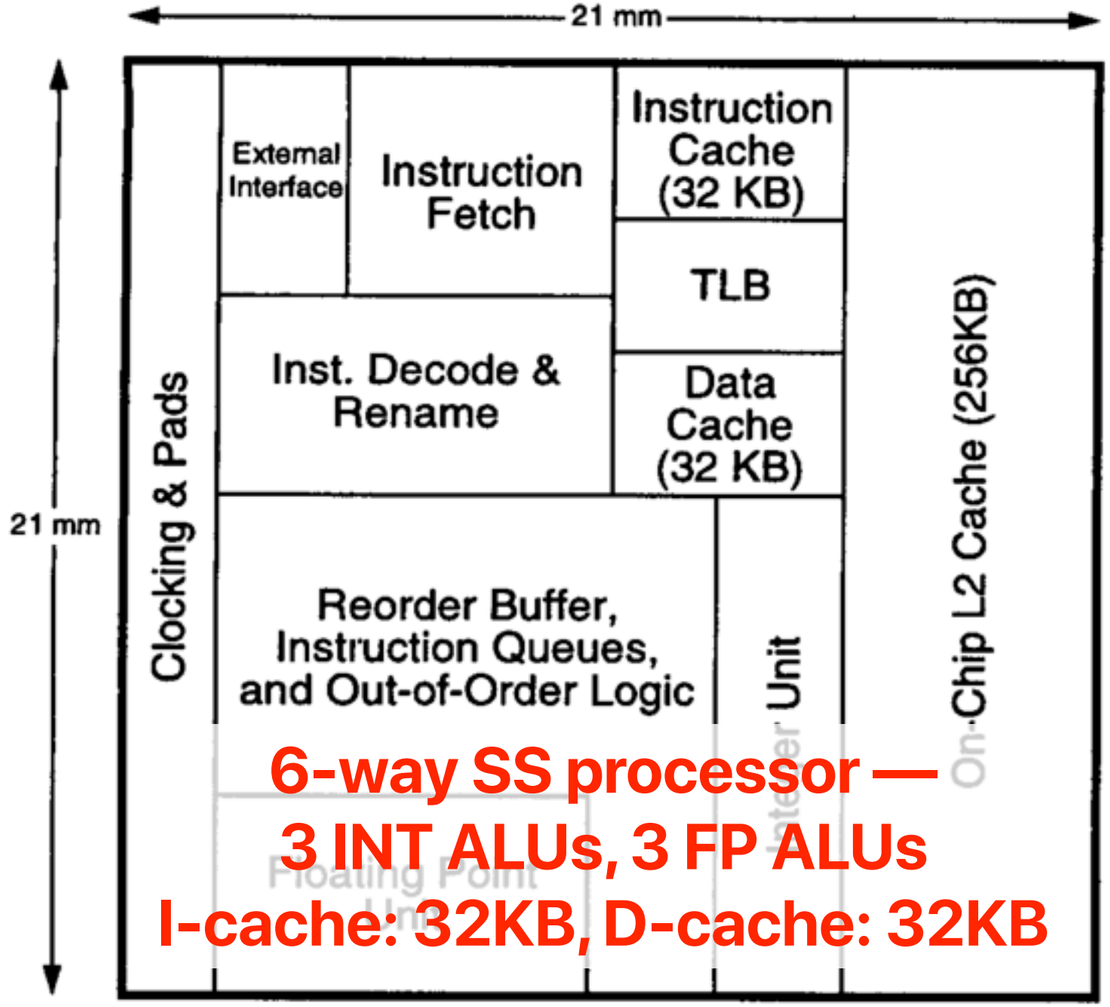
SMT


Program Counter
Program Counter
Instruction Fetch
Instructions
Branch Predictor
Reorder Buffer/ Instruction Queue
Register renaming logic
Instruction Decode
$O(IW^4)$
Registers
Memory
Branch/ Jump
Arithmetic Logical Units (ALU)
Complex Arithmetic Operations (Mul/div)
Memory Operations
Data

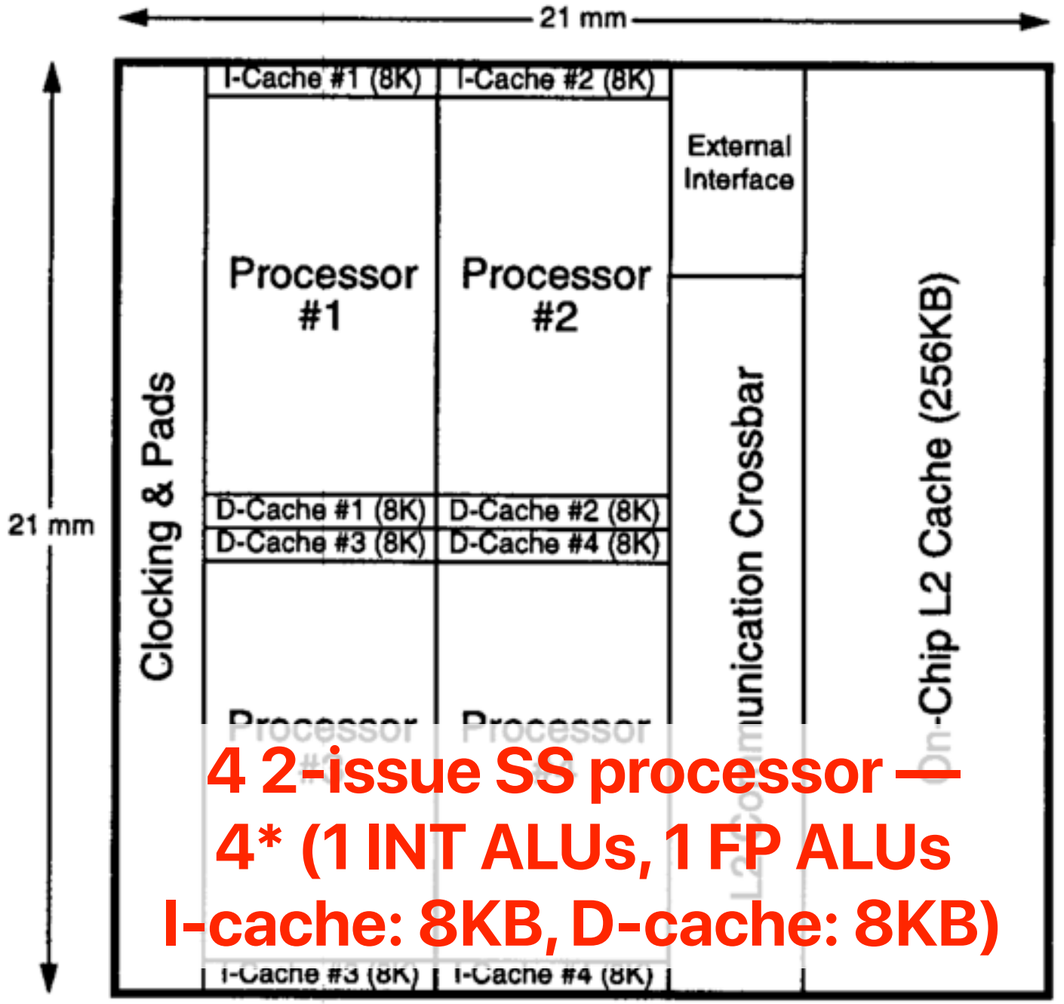
The case for a Single-Chip Multiprocessor

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

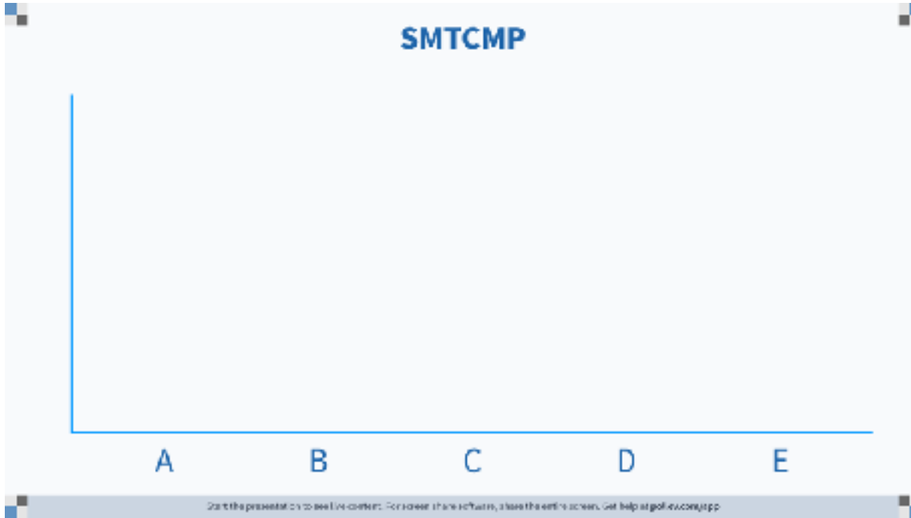


4 2-issue SS processor —
4* (1 INT ALUs, 1 FP ALUs
I-cache: 8KB, D-cache: 8KB)

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 **The only thing we know for sure — if we don't parallel the program, it won't get any faster on CMP**
  D. 4
  E. 5

6-way SuperScalar v.s. quad-core CMP

The applications are parallelized in different ways to run on the MP microarchitecture. Compress is run unmodified on both the SS and MP microarchitectures; using only one processor of the MP architecture. Eqntott is parallelized manually by modifying a single bit vector comparison routine that is responsible for 90% of the execution time of the application [16]. The CPU simulator m88ksim is also parallelized manually into three threads using the SUIF compiler runtime system. Each of the three threads is allowed to be in a different phase of simulating a different instruction at the same time. This style of parallelization is very similar to the overlap of instruction execution that occurs in hardware pipelining. The

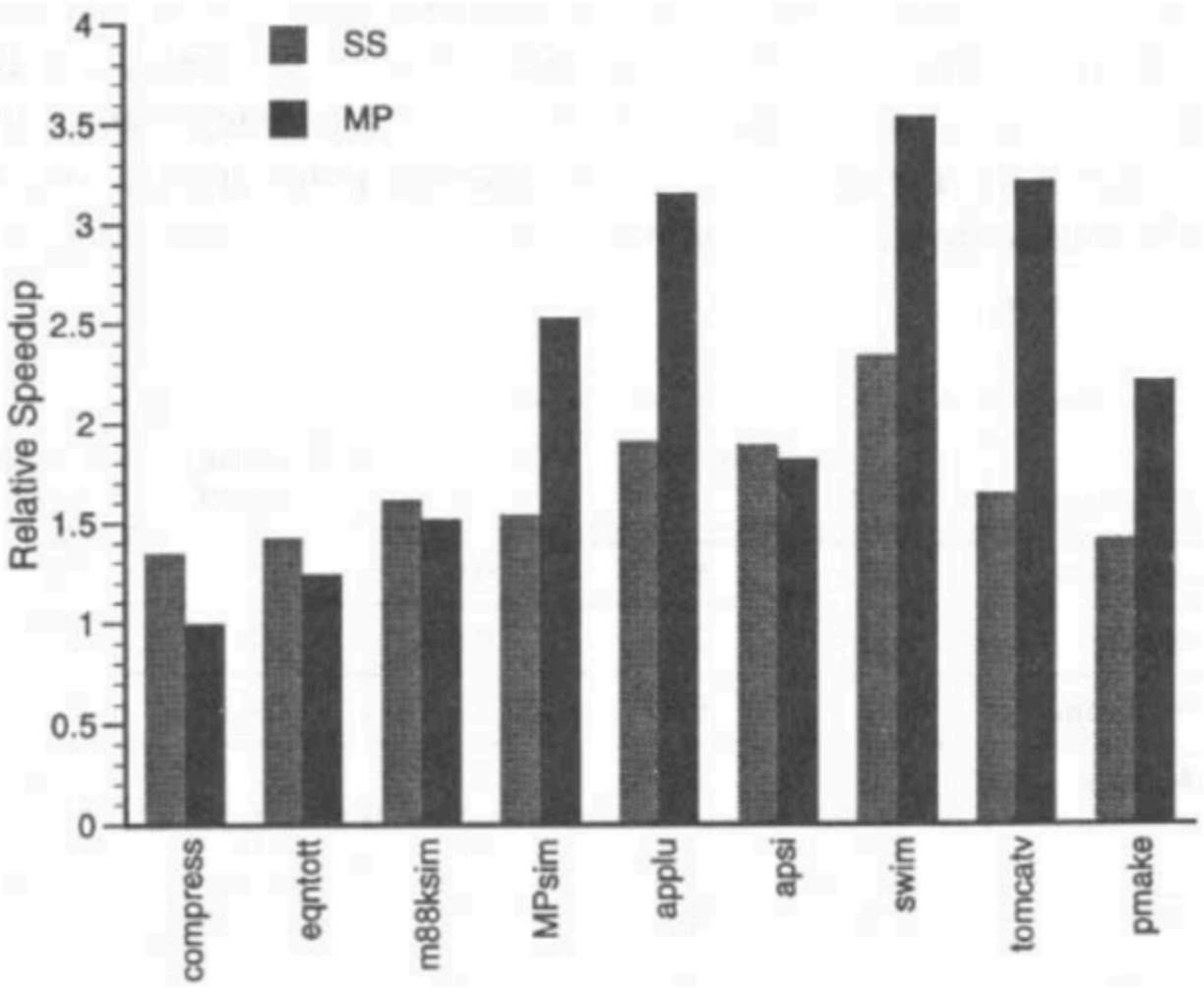
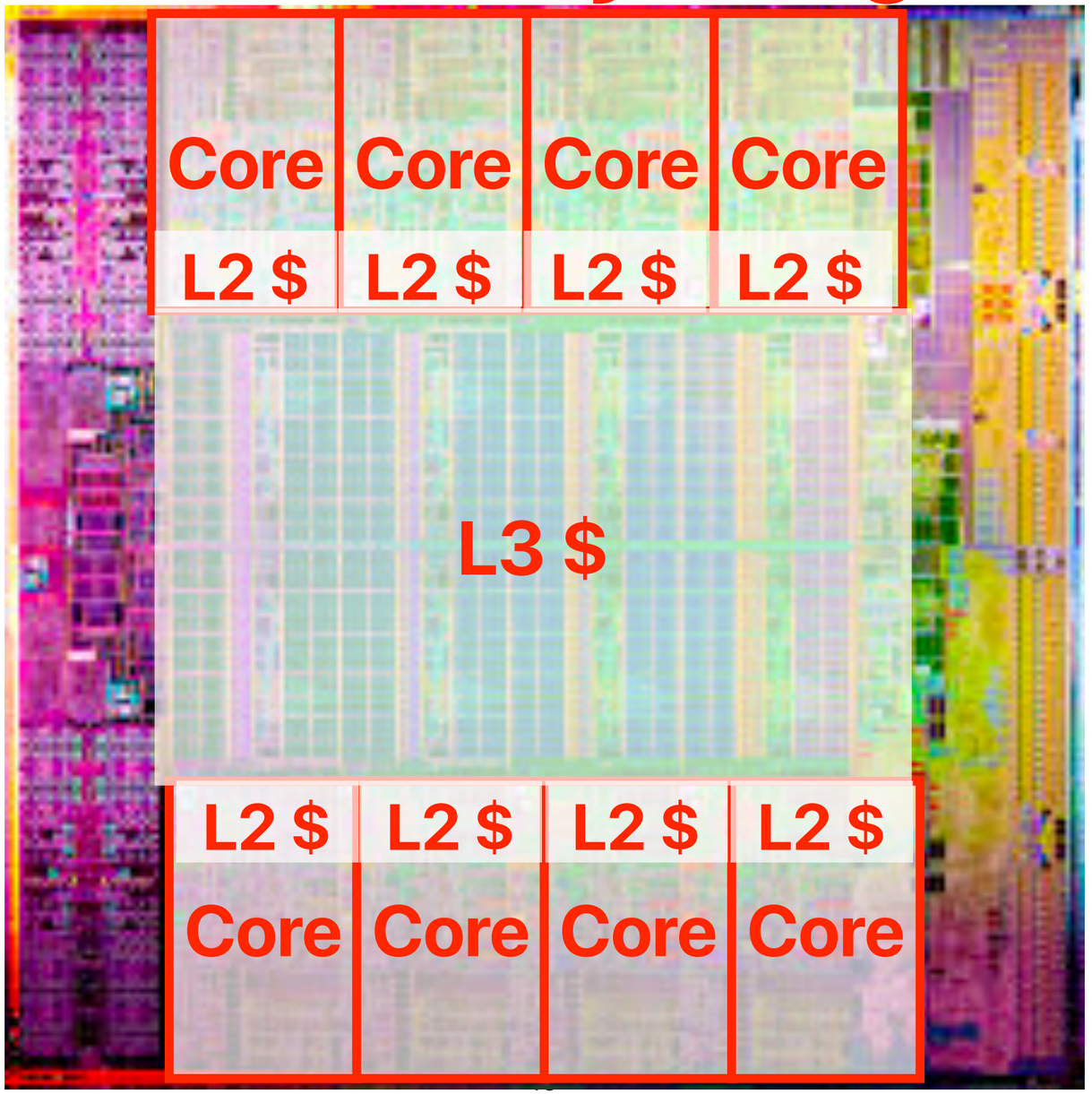
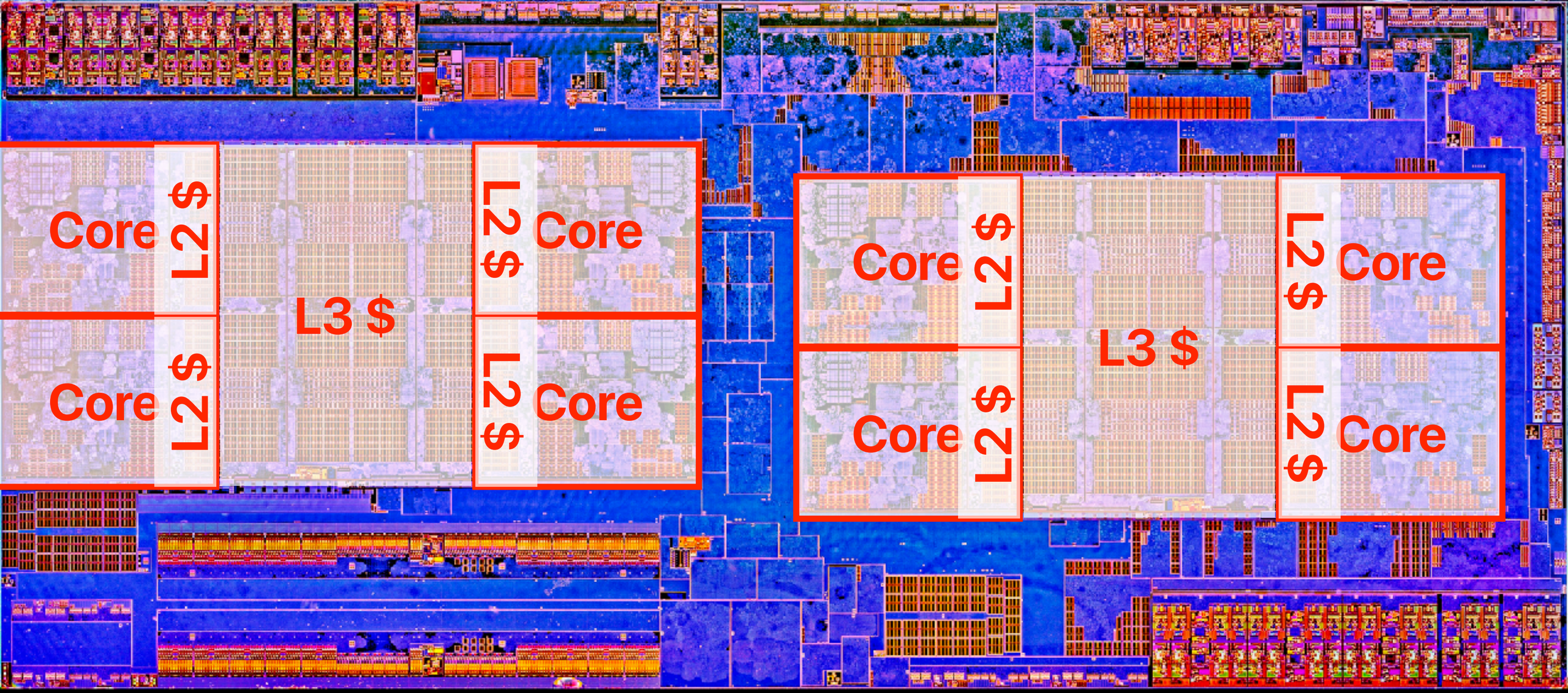


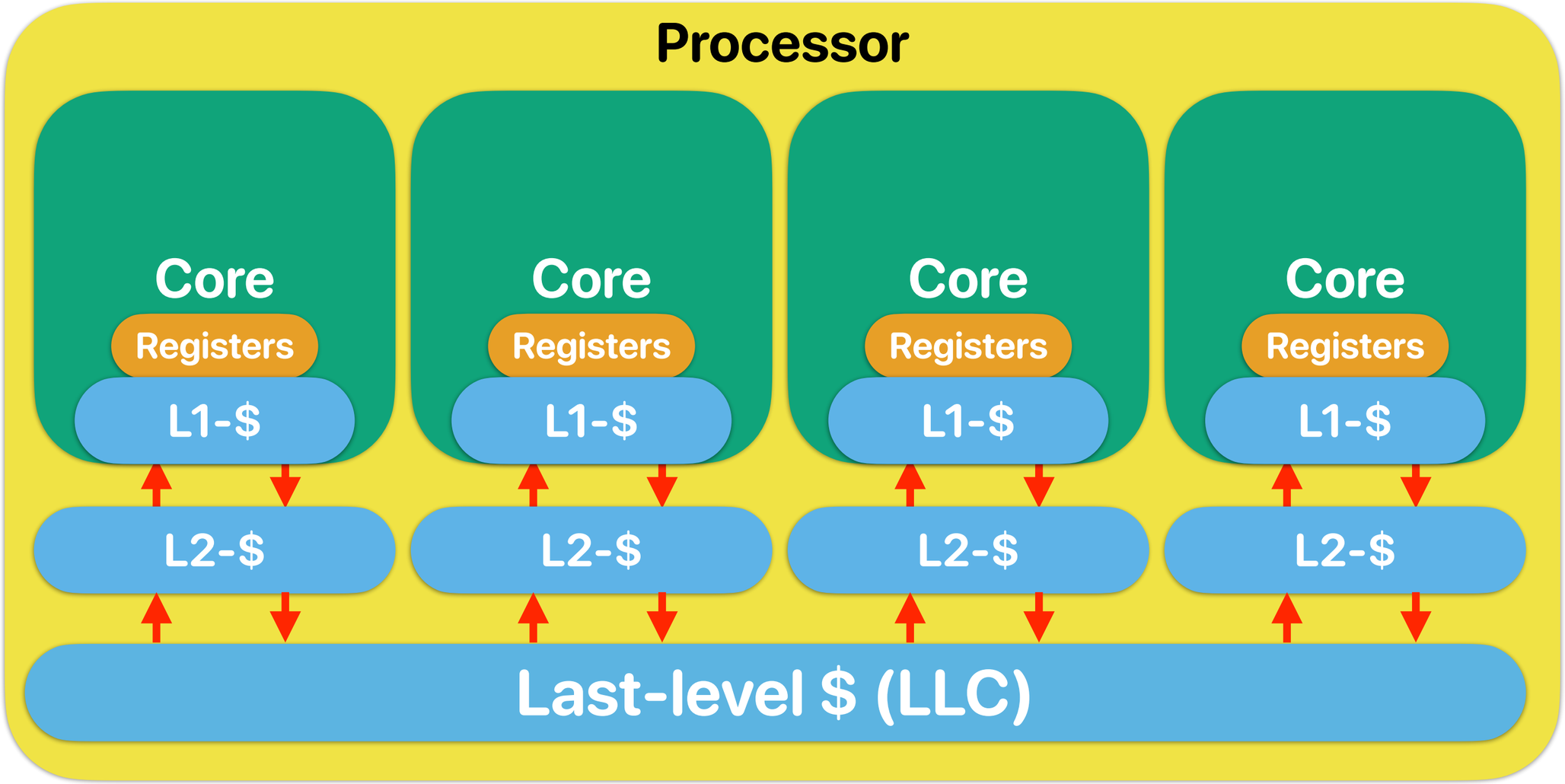
Figure 6. Performance comparison of SS and MP.

Intel Sandy Bridge




Core
L2 $
L3 $
L2 $
Core
Core
L2 $
L2 $
Core
Core
L2 $
L3 $
L2 $
Core
Core
L2 $
L2 $
Core
AMD
RYZEN

Concept of CMP

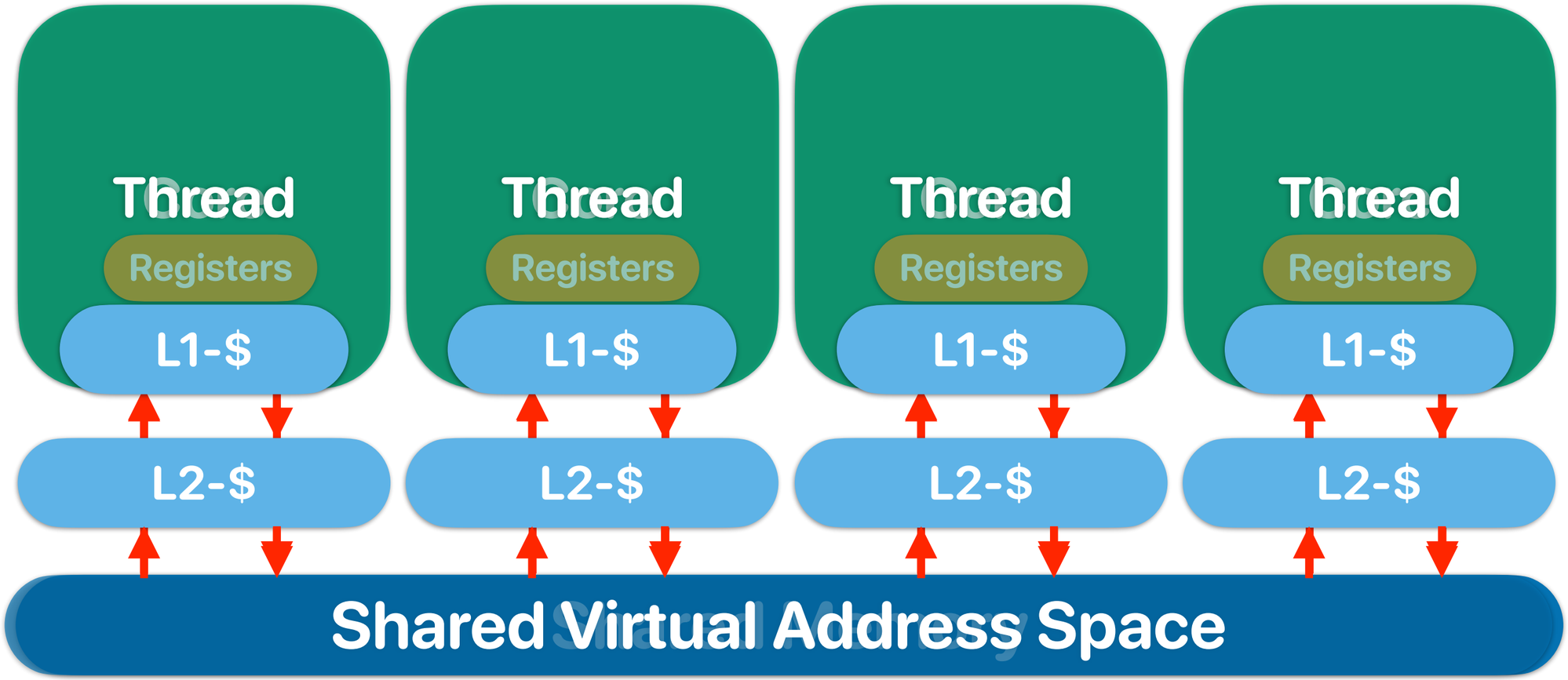


Architectural Support for Parallel Programming

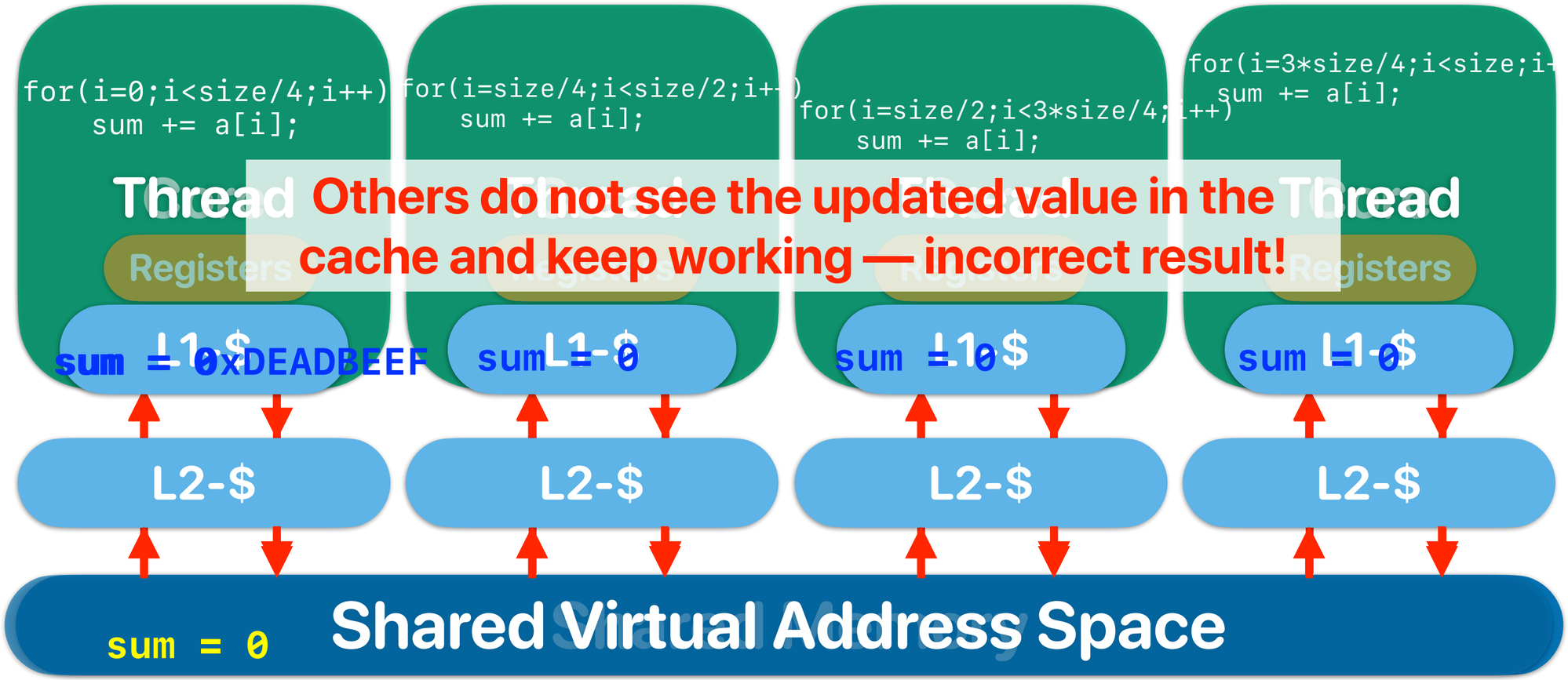
Parallel programming

- 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
- Threads (in OS/software systems)
  - Independent portions of your program that can run in parallel
  - All threads share the same virtual memory space
- 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!

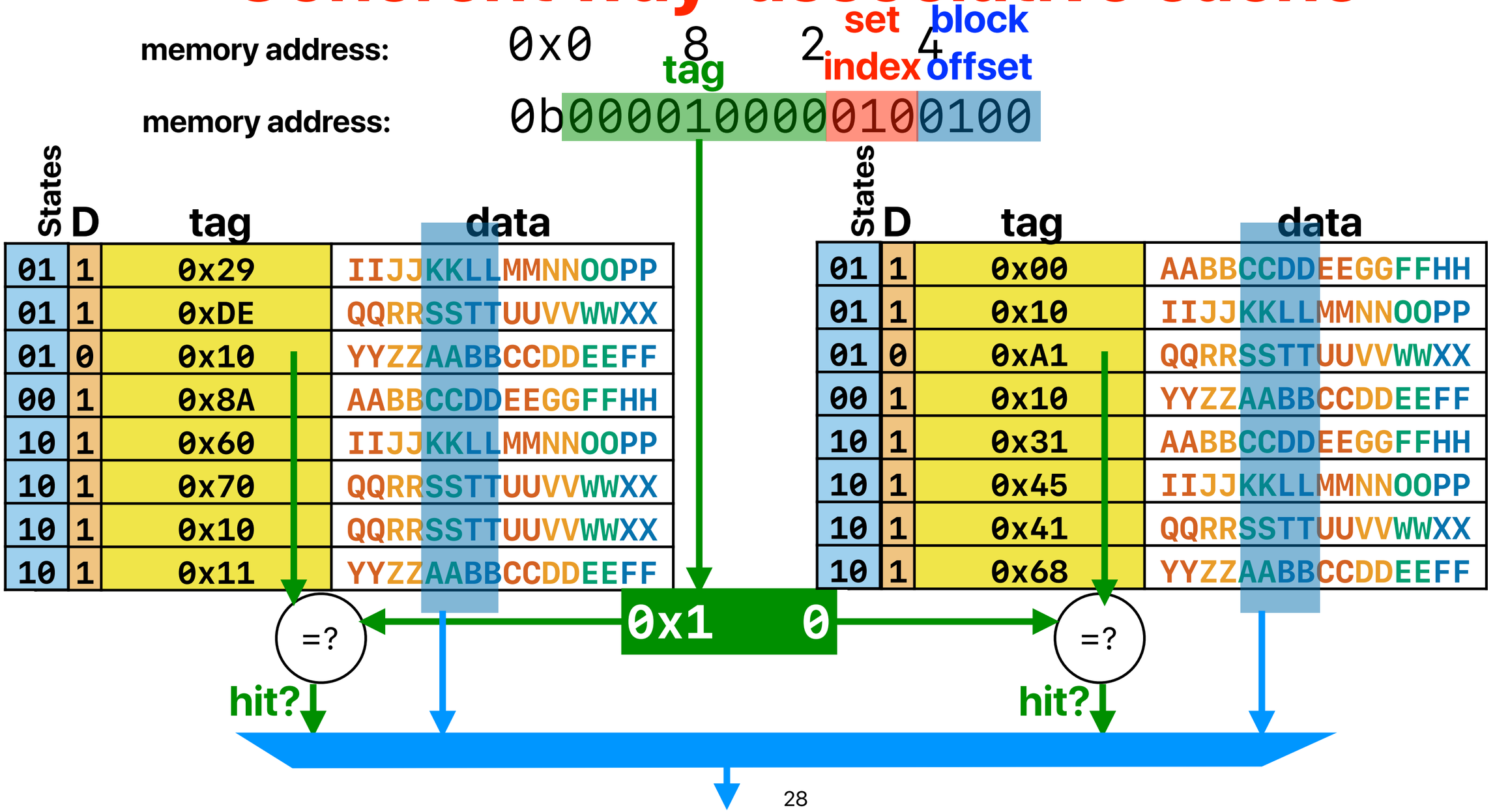
Coherency & Consistency

- 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

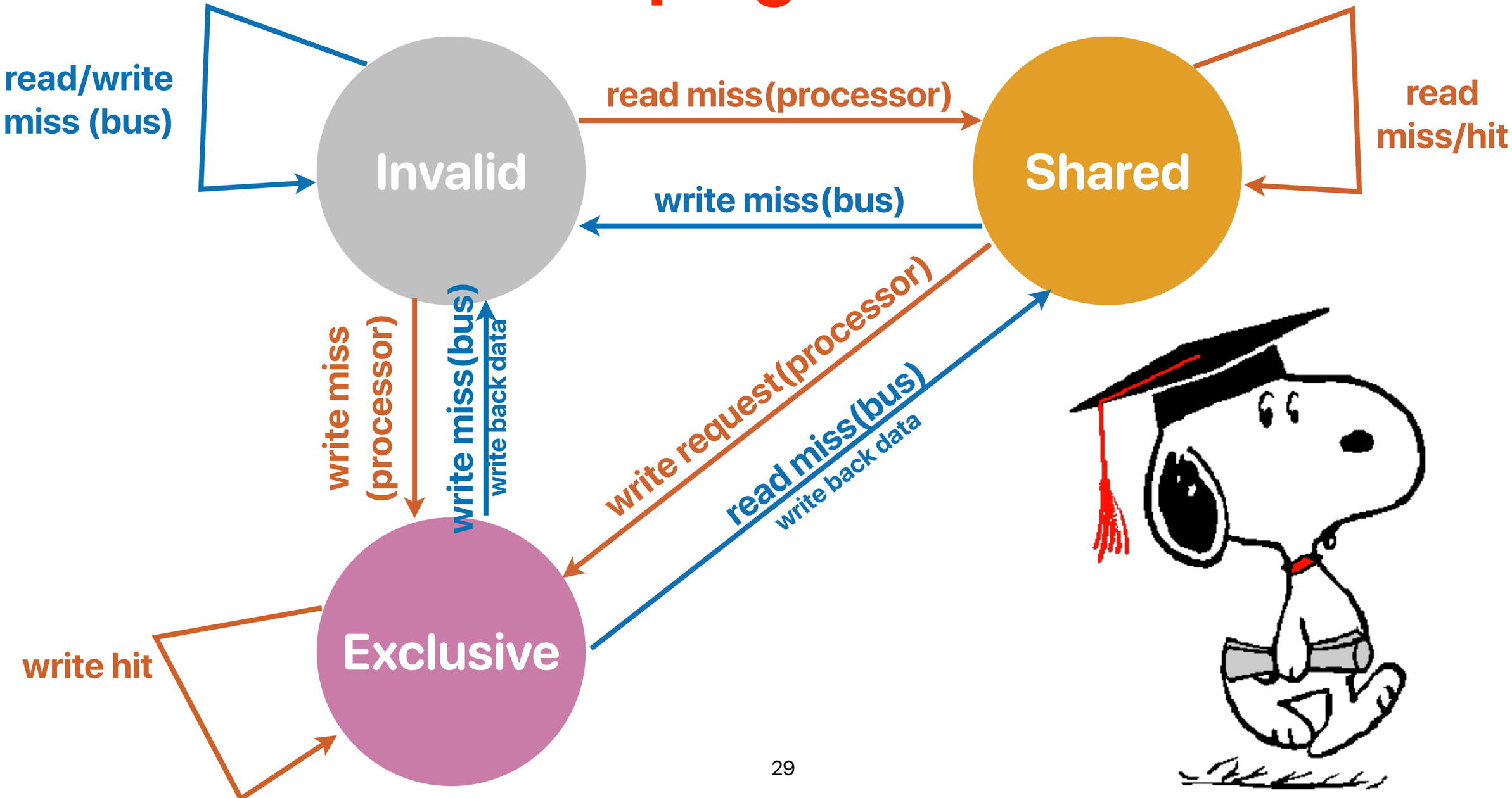
Simple cache coherency protocol

- 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

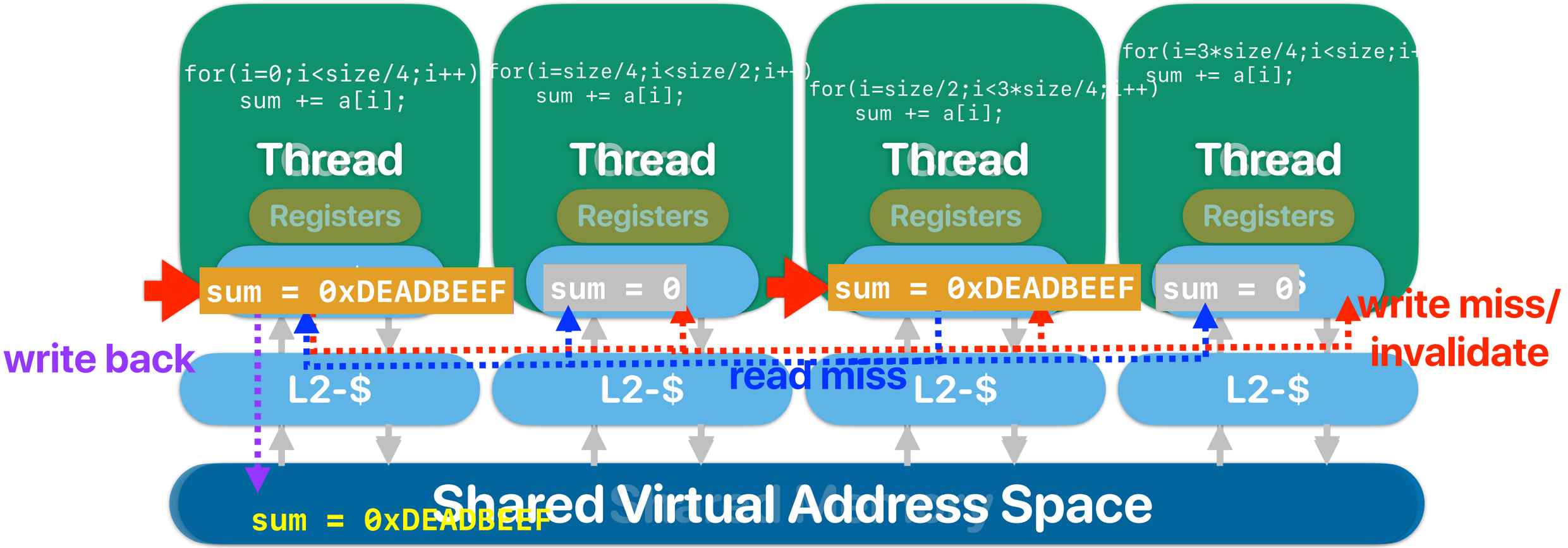
Coherent way-associative cache



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", loop);
  return 0;
}
```

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	thread 2

```
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", loop);
  return 0;
}
```

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

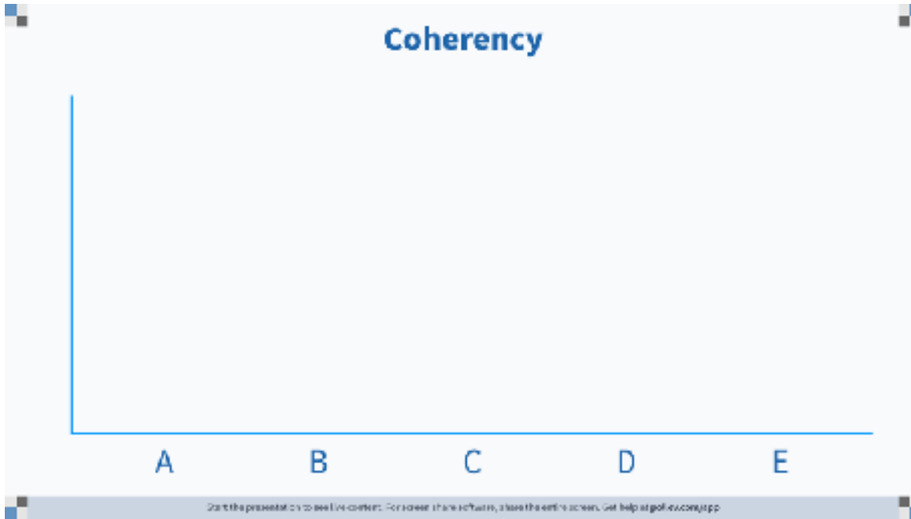
Cache coherency

- 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

Announcements

- Last Reading Quiz due next Tuesday
  - We drop two of your lowest ones
- Assignment #4 due 6/8
- If you submit iEVAL and submit the screenshot through eLearn, it counts as a "full-credit" notebook assignment
  - We drop two notebook assignments with this one included
  - In other words, if you submit iEVAL and the screenshot, you got two lowest assignments dropped.

Computer
Science &
Engineering

203

つづく