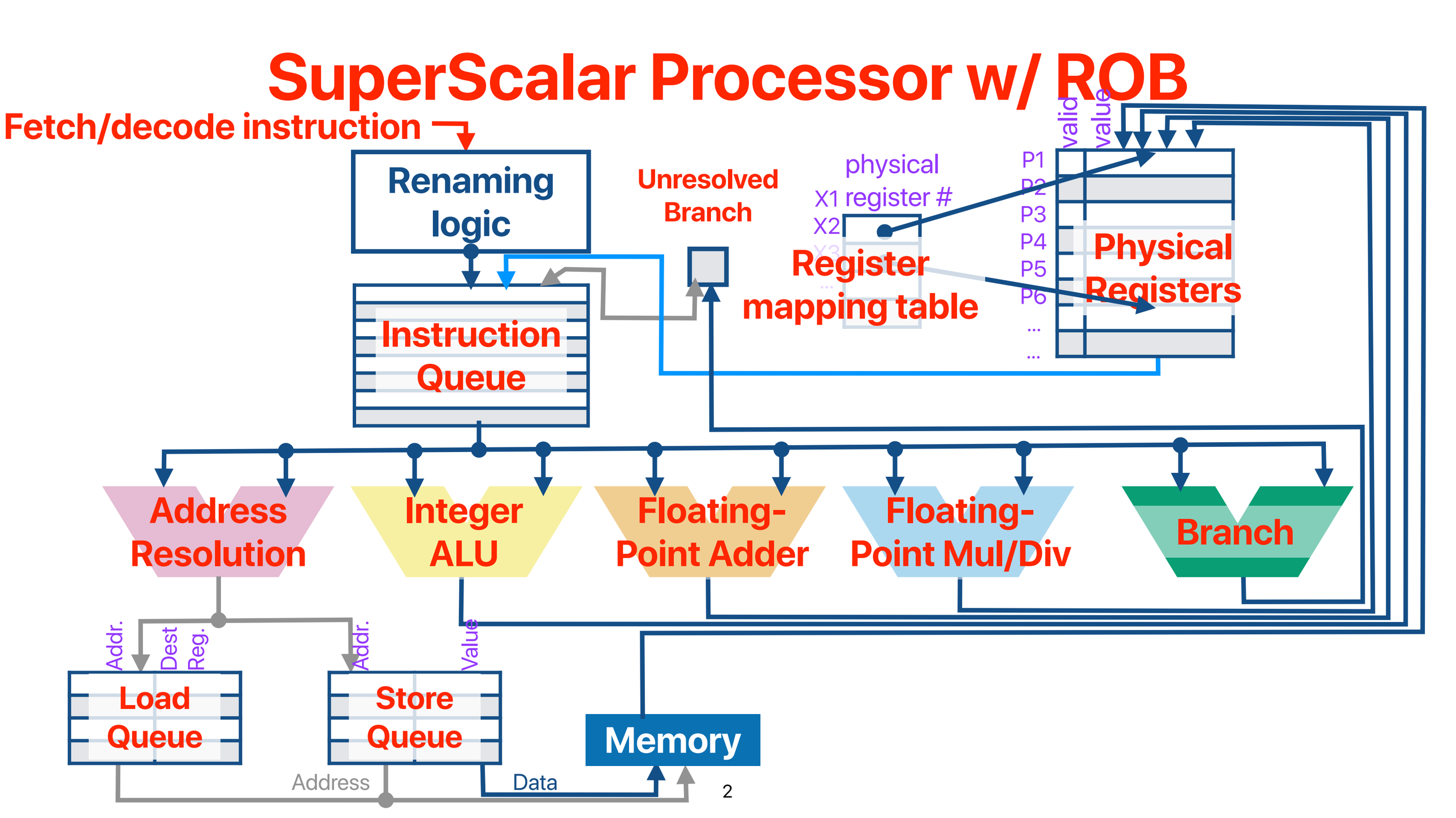
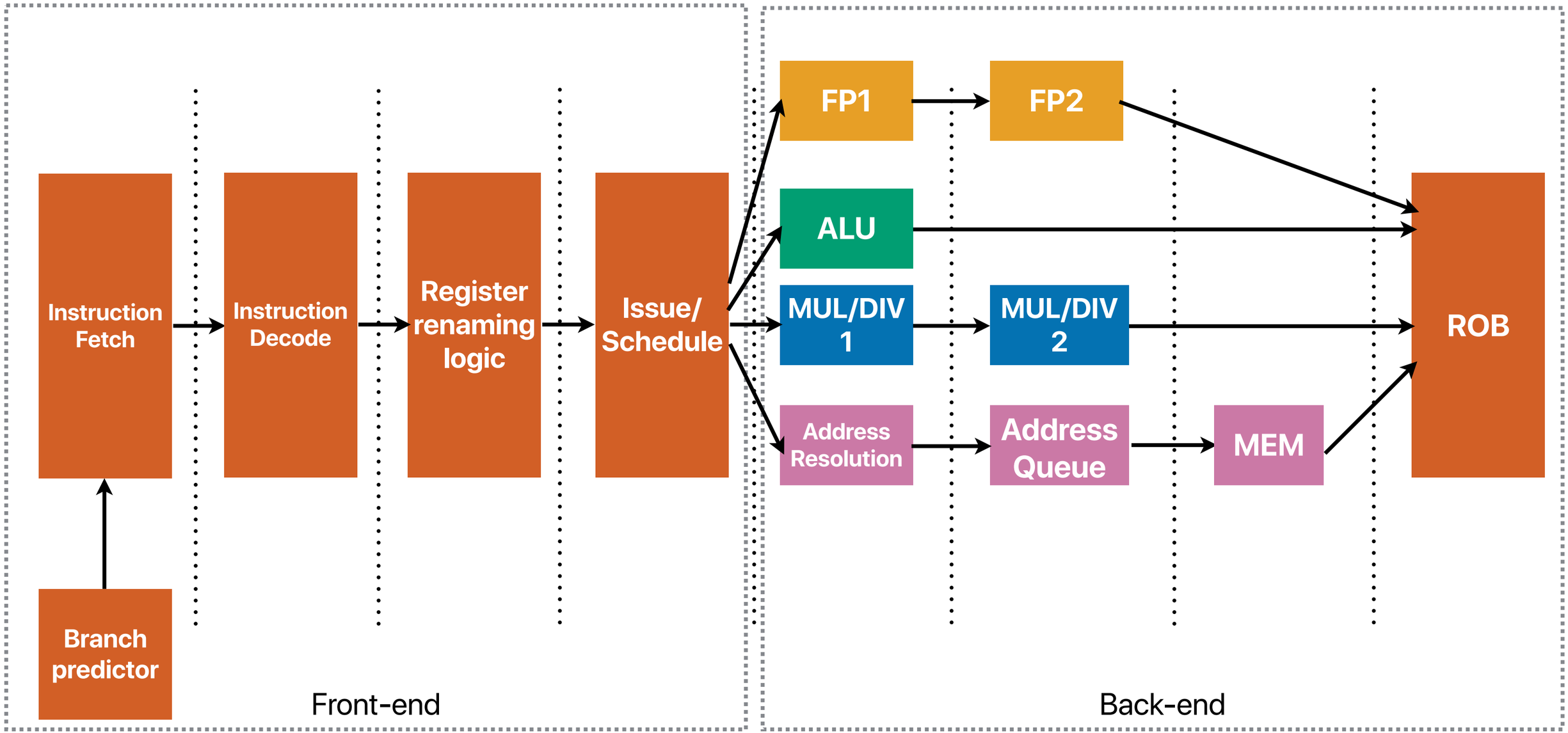
Multithreaded Architectures and Programming on Multithreaded Architectures

Hung-Wei Tseng

SuperScalar Processor w/ ROB



Recap: Pipeline SuperScalar/OoO/ROB



Recap: Register renaming

- Provide a set of “physical registers” and a mapping table mapping “architectural registers” to “physical registers”
- Allocate a physical register for a new output
- Eliminate all false dependencies
- Stages
  - Dispatch (D) — allocate a “physical” for the output of a decoded instruction
  - Issue (I) — collect pending values/branch outcome from common data bus
  - Execute (INT, AQ/AQ/MEM, M1/M2/M3, BR) — send the instruction to its corresponding pipeline if no structural hazards
  - Write Back (WB) — broadcast the result through CDB

Recap: What about “linked list”

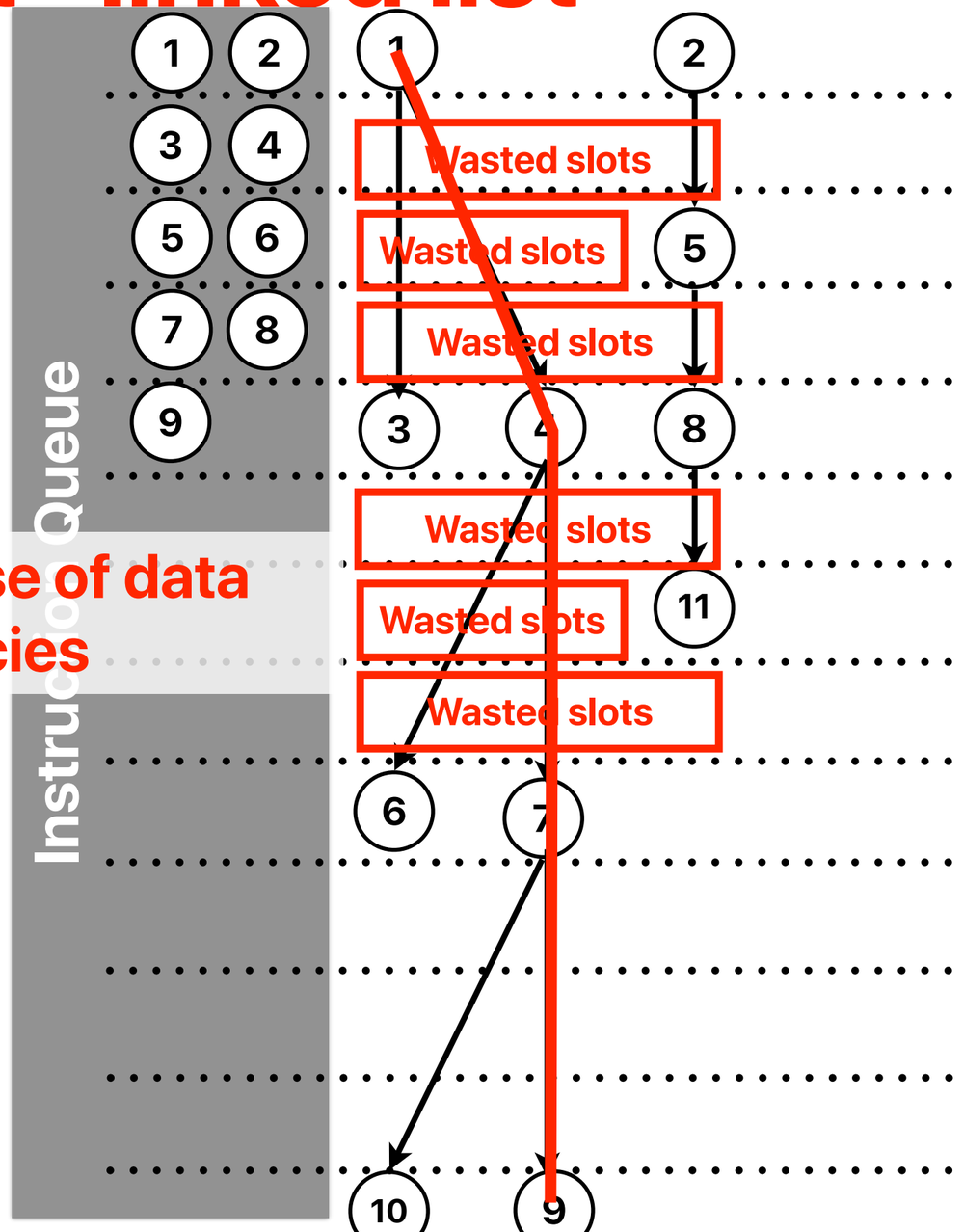
Static instructions

```
LOOP: ld   X10, 8(X10)
      addi  X7, X7, 1
      bne  X10, X0, LOOP
```

Dynamic instructions

```
① ld   X10, 8(X10)
② addi  X7, X7, 1
③ bne  X10, X0, LOOP
④ ld   X10, 8(X10)
⑤ addi  X7, X7, 1
⑥ bne  X10, X0, LOOP
⑦ ld   X10, 8(X10)
⑧ addi  X7, X7, 1
⑨ bne  X10, X0, LOOP
```

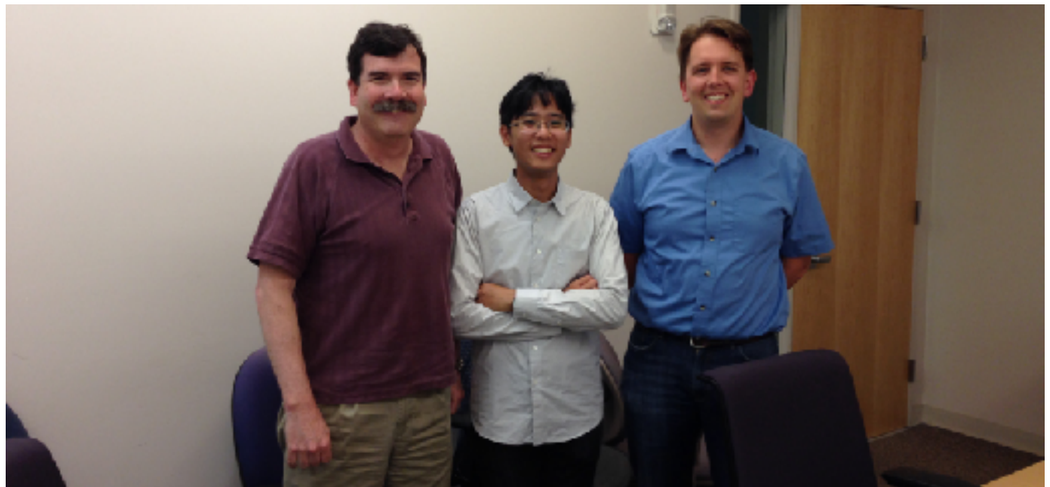
ILP is low because of data dependencies



Demo: 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.

Simultaneous multithreading

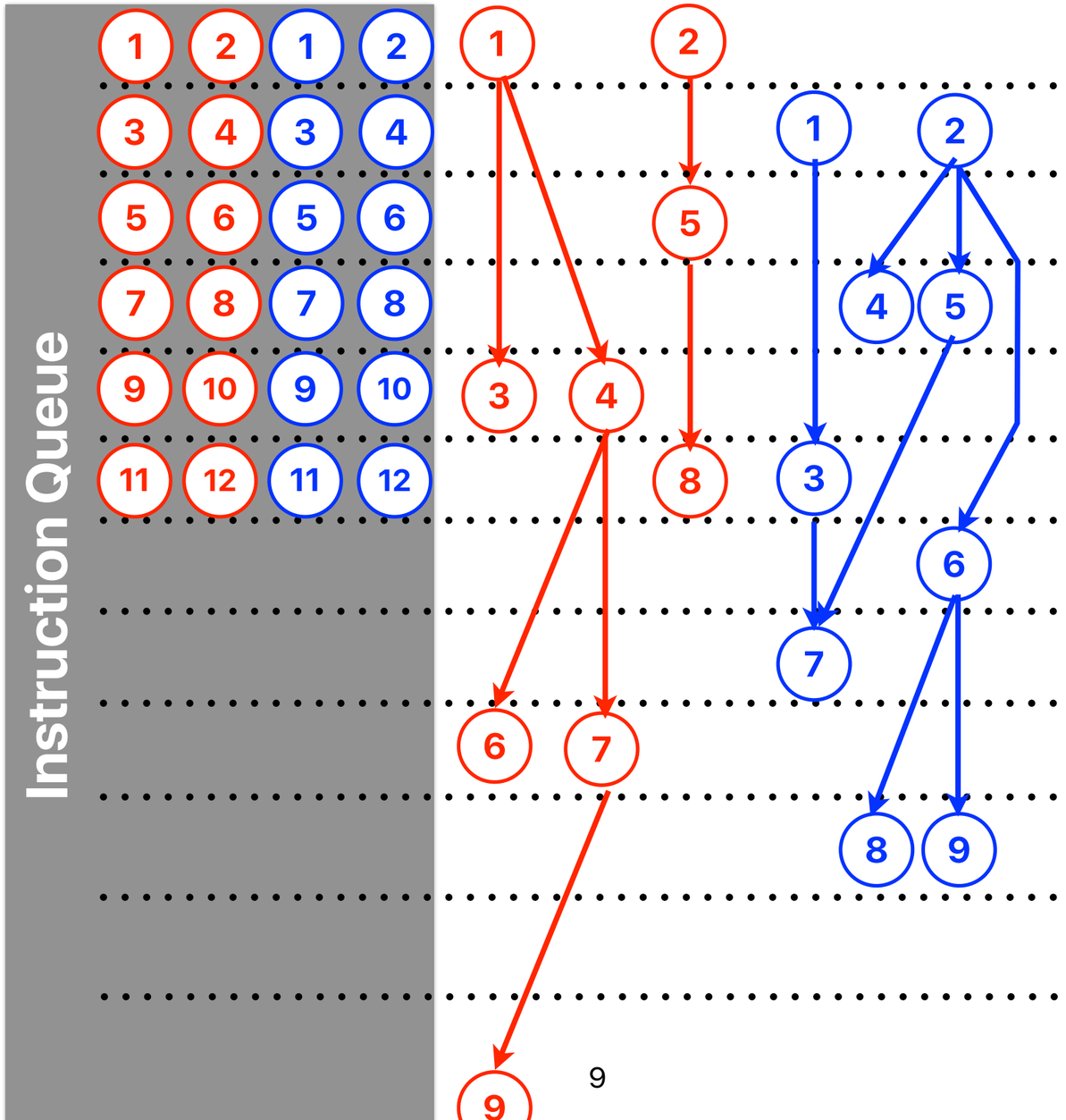


Simultaneous multithreading

- The processor can schedule instructions from different threads/processes/programs
- Fetch instructions from different threads/processes to fill the not utilized part of pipeline
  - Exploit “thread level parallelism” (TLP) to solve the problem of insufficient ILP in a single thread
  - You need to create an illusion of multiple processors for OSs

Simultaneous multithreading

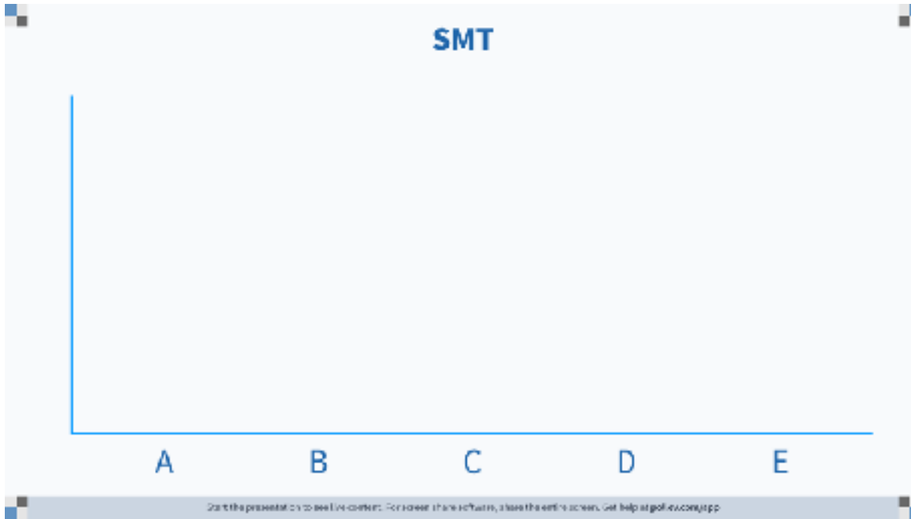
```
① ld   X10, 8(X10)
② addi  X7, X7, 1
③ bne  X10, X0, LOOP
④ ld   X10, 8(X10)
⑤ addi  X7, X7, 1
⑥ bne  X10, X0, LOOP
⑦ ld   X10, 8(X10)
⑧ addi  X7, X7, 1
⑨ bne  X10, X0, LOOP
```




```
① ld   X1, 0(X10)
② addi X10, X10, 8
③ add  X20, X20, X1
④ bne  X10, X2, LOOP
⑤ ld   X1, 0(X10)
⑥ addi X10, X10, 8
⑦ add  X20, X20, X1
⑧ bne  X10, X2, LOOP
⑨ ld   X1, 0(X10)
⑩ addi X10, X10, 8
⑪ add  X20, X20, X1
⑫ bne  X10, X2, LOOP
```

Architectural support for simultaneous multithreading

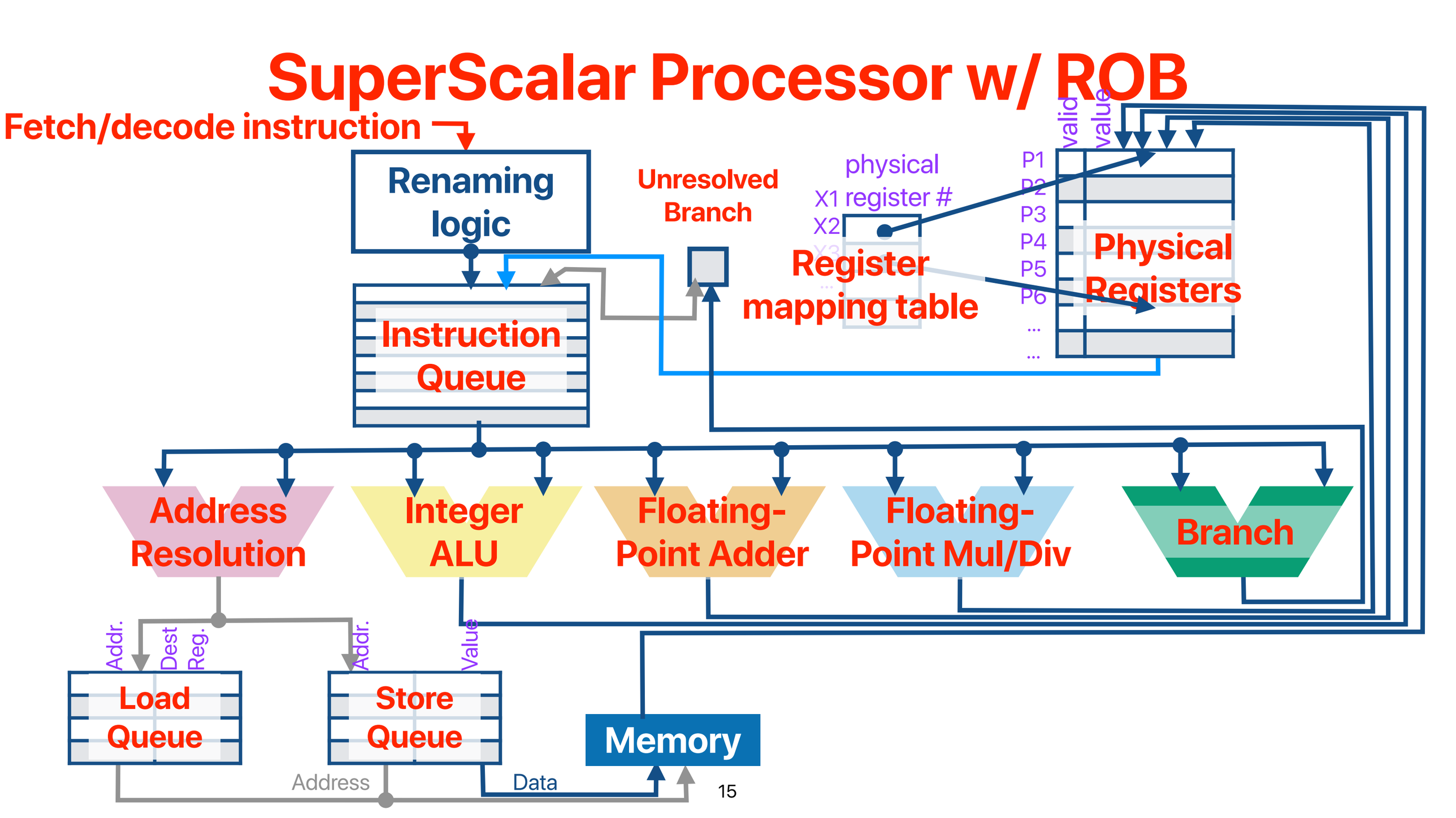
- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  - ① Program counter
  - ② Register mapping tables
  - ③ Physical registers
  - ④ ALUs
  - ⑤ Data cache
  - ⑥ Reorder buffer/Instruction Queue
  - A. 2
  - B. 3
  - C. 4
  - D. 5
  - E. 6



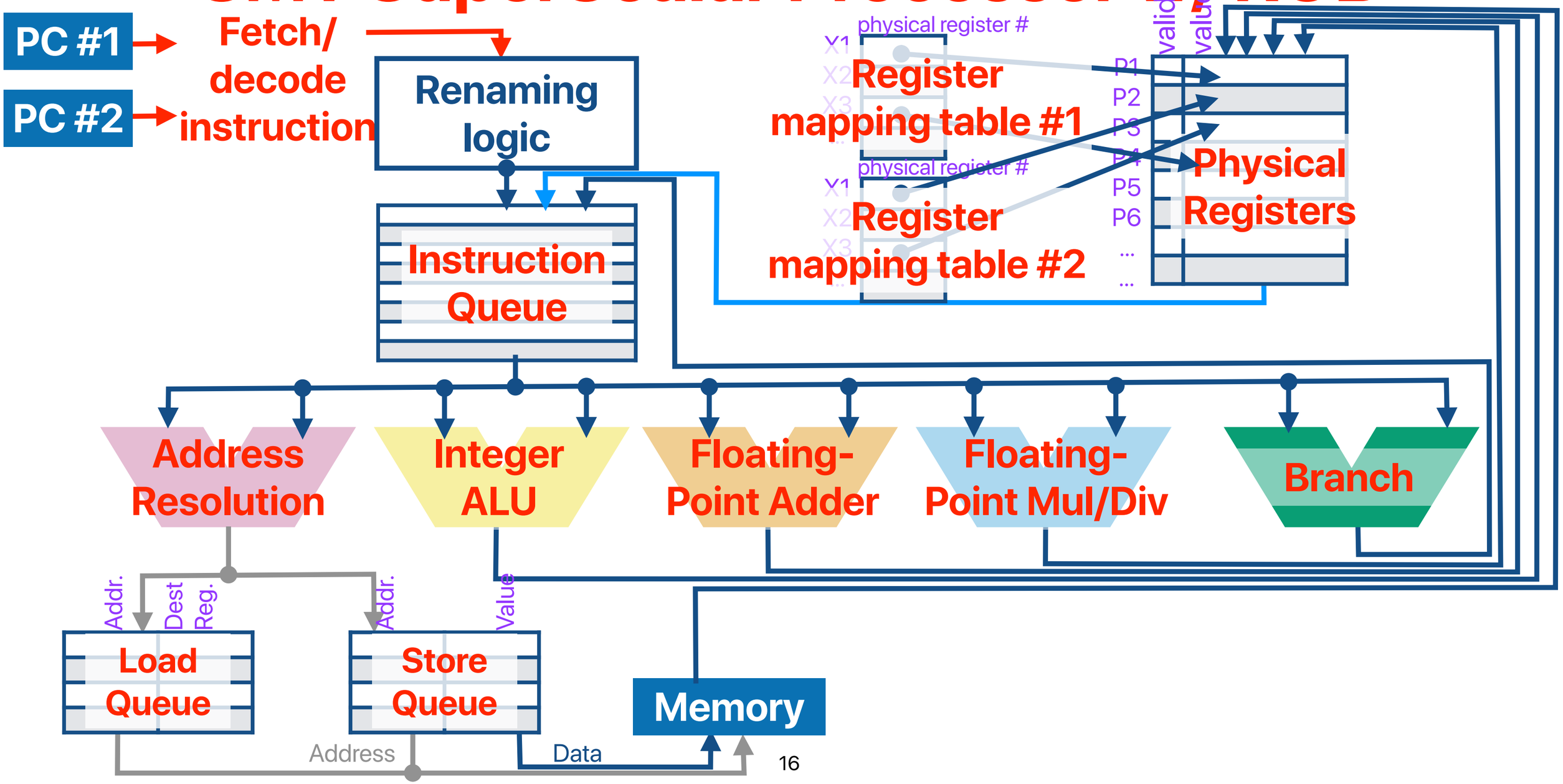
Architectural support for simultaneous multithreading

- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  - ① Program counter — **you need to have one for each context**
  - ② Register mapping tables — **you need to have one for each context**
  - ③ Physical registers — **you can share**
  - ④ ALUs — **you can share**
  - ⑤ Data cache — **you can share**
  - ⑥ Reorder buffer/Instruction Queue — **you need to indicate which context the instruction is from**
  - A. 2
  - B. 3
  - C. 4
  - D. 5
  - E. 6

SuperScalar Processor w/ ROB



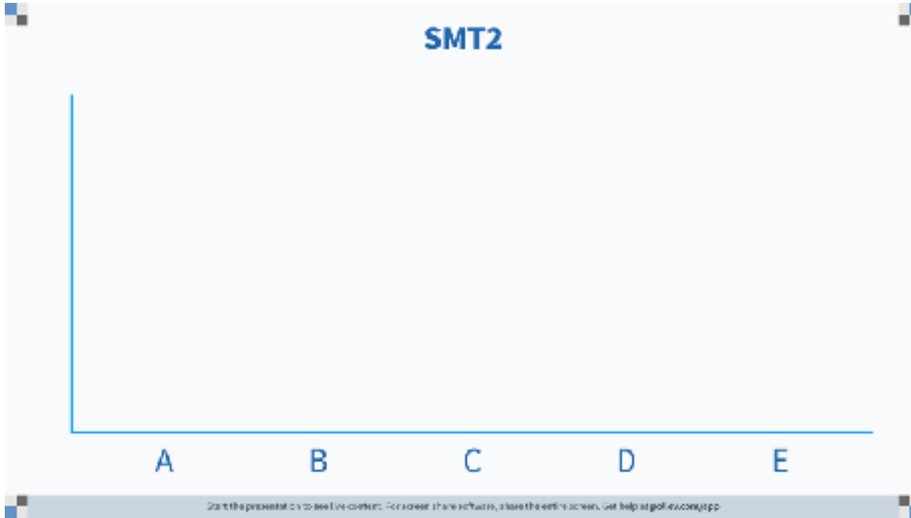
SMT SuperScalar Processor w/ ROB



SMT

- How many of the following about SMT are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can improve the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

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



SMT

- How many of the following about SMT are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can ~~improve~~ the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

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

We can execute from other threads/contexts instead of the current one

hurt, b/c you are sharing resource with other threads.

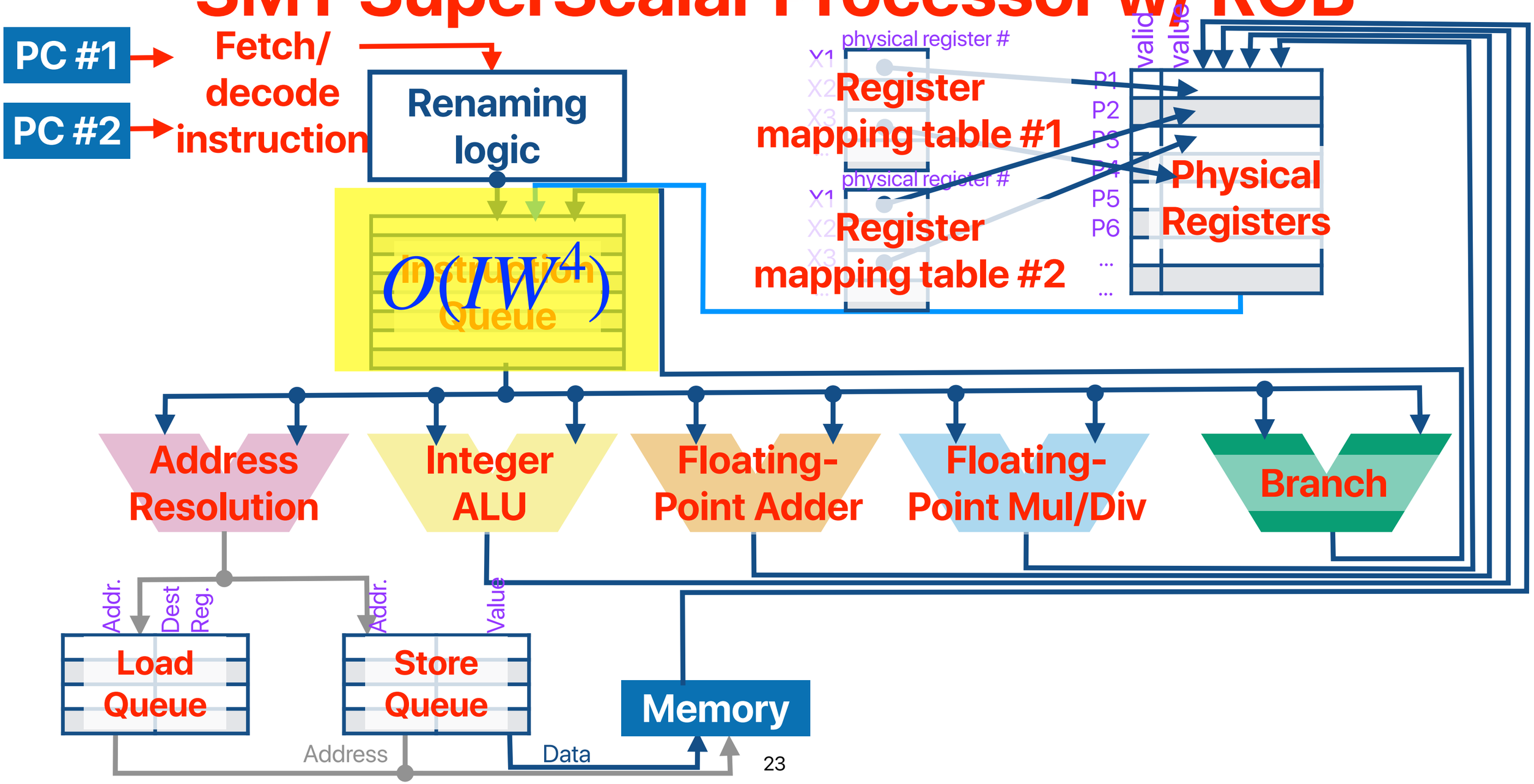
We can execute from other threads/ contexts instead of the current one

b/c we're sharing the cache

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)

SMT SuperScalar Processor w/ ROB



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.

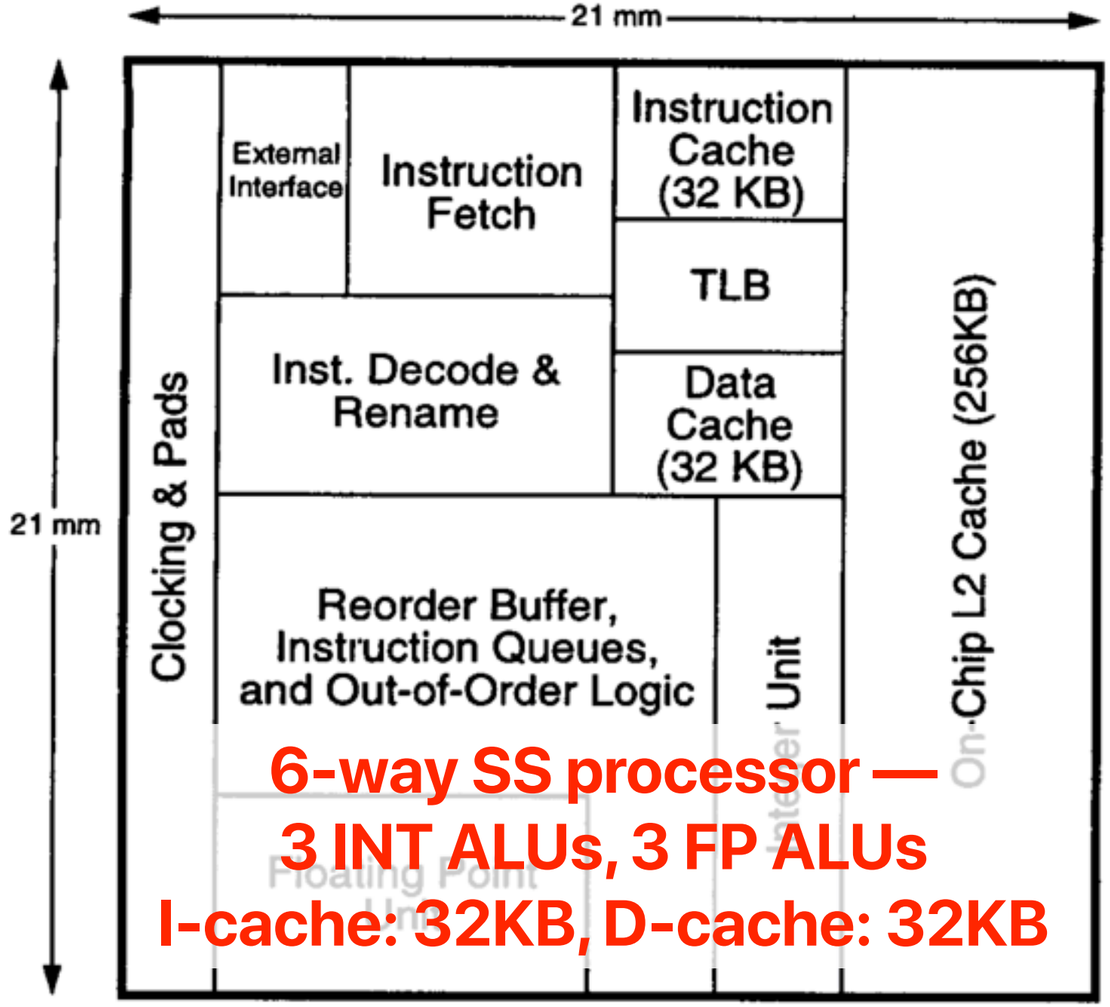
Program	IPC	BP Rate %	I cache %MPCI	D cache %MPCI	L2 cache %MPCI
compress	1.2	86.4	0.0	3.9	1.1
eqntott	1.8	80.0	0.0	1.1	1.1
m88ksim	2.3	92.6	0.1	0.0	0.0
MPsim	1.2	81.6	3.4	1.7	2.3
applu	1.7	79.7	0.0	2.8	2.8
apsi	1.2	95.6	0.2	3.1	2.6
swim	2.2	99.8	0.0	2.3	2.5
tomcatv	1.3	99.7	0.0	4.2	4.3
pmake	1.4	82.7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

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

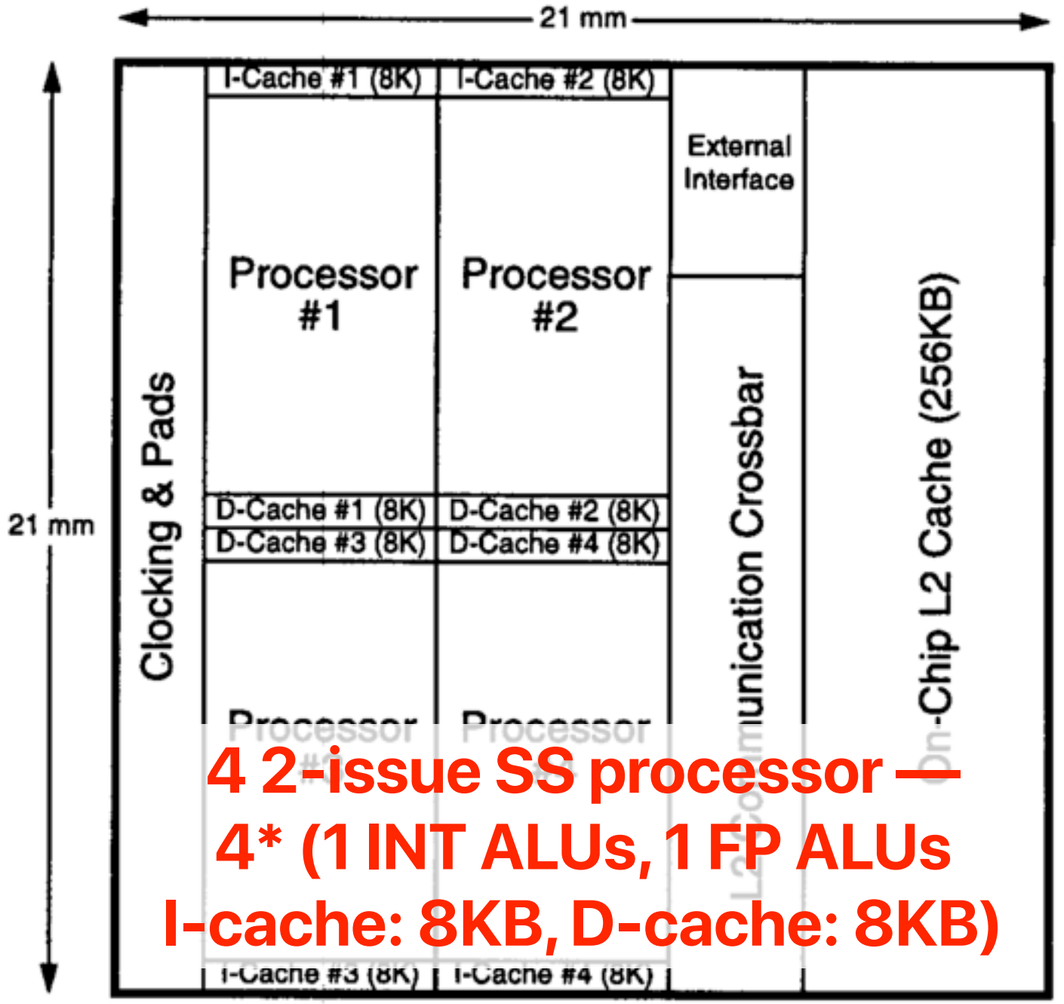
The case for a Single-Chip Multiprocessor

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

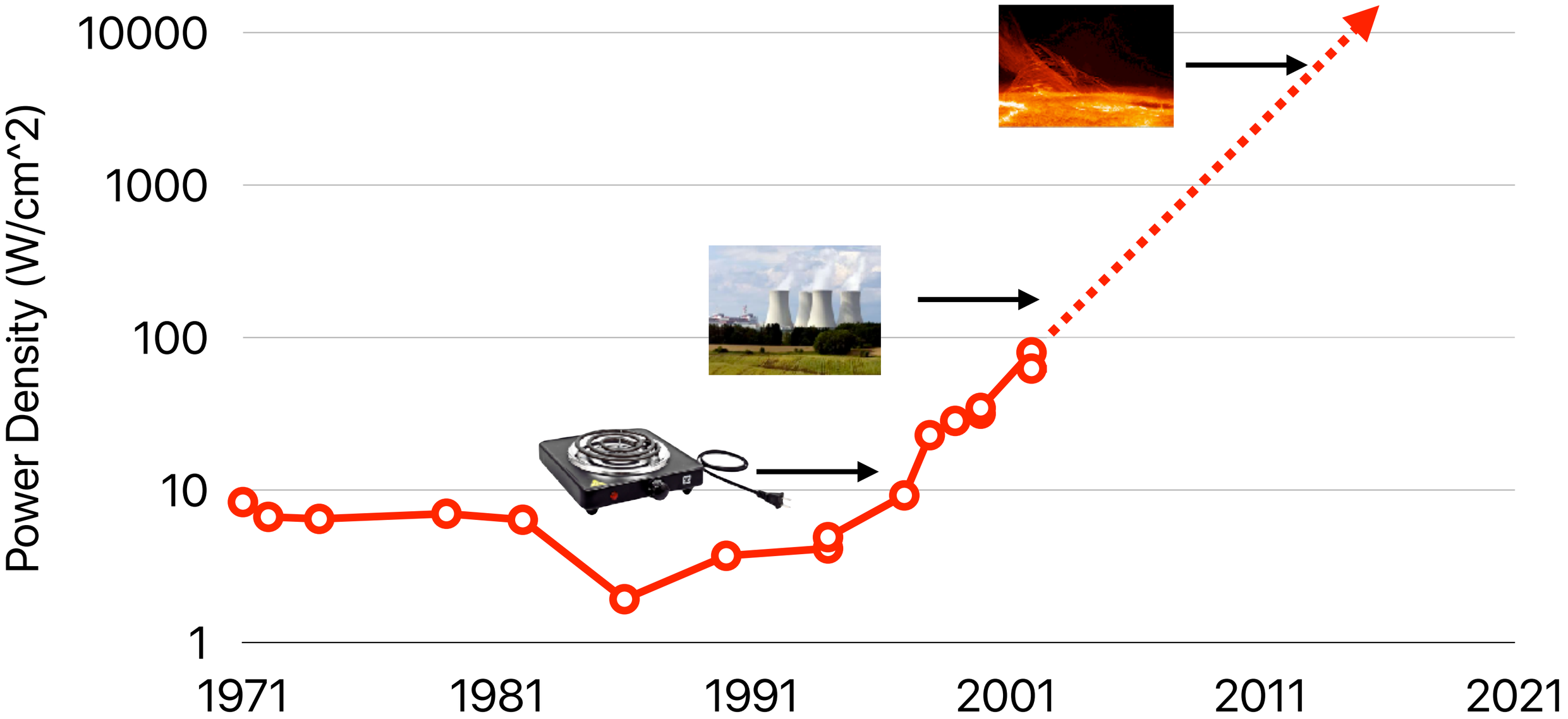


6-way SS processor —
3 INT ALUs, 3 FP ALUs
I-cache: 32KB, D-cache: 32KB



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

Recap: Power Density of Processors



Recap: Power consumption & power density

- The power consumption due to the switching of transistor states
- Dynamic power per transistor:

$$P_{dynamic} \sim \alpha \times C \times V^2 \times f \times N$$

  - α: average switches per cycle
  - C: capacitance
  - V: voltage
  - f: frequency, usually linear with V
  - N: the number of transistors

We cannot make chips always operating at very high frequencies

- Power density:

$$P_{density} = \frac{P}{area}$$

Moore's Law allows higher frequencies as transistors are smaller

Moore's Law makes this smaller

The "power" of doubling the clocking rate v.s. doubling the number of cores

$$P_{dynamic} \sim \alpha \times C \times \boxed{V^2 \times f} \times \boxed{N}$$

Doubling the clocking rate:

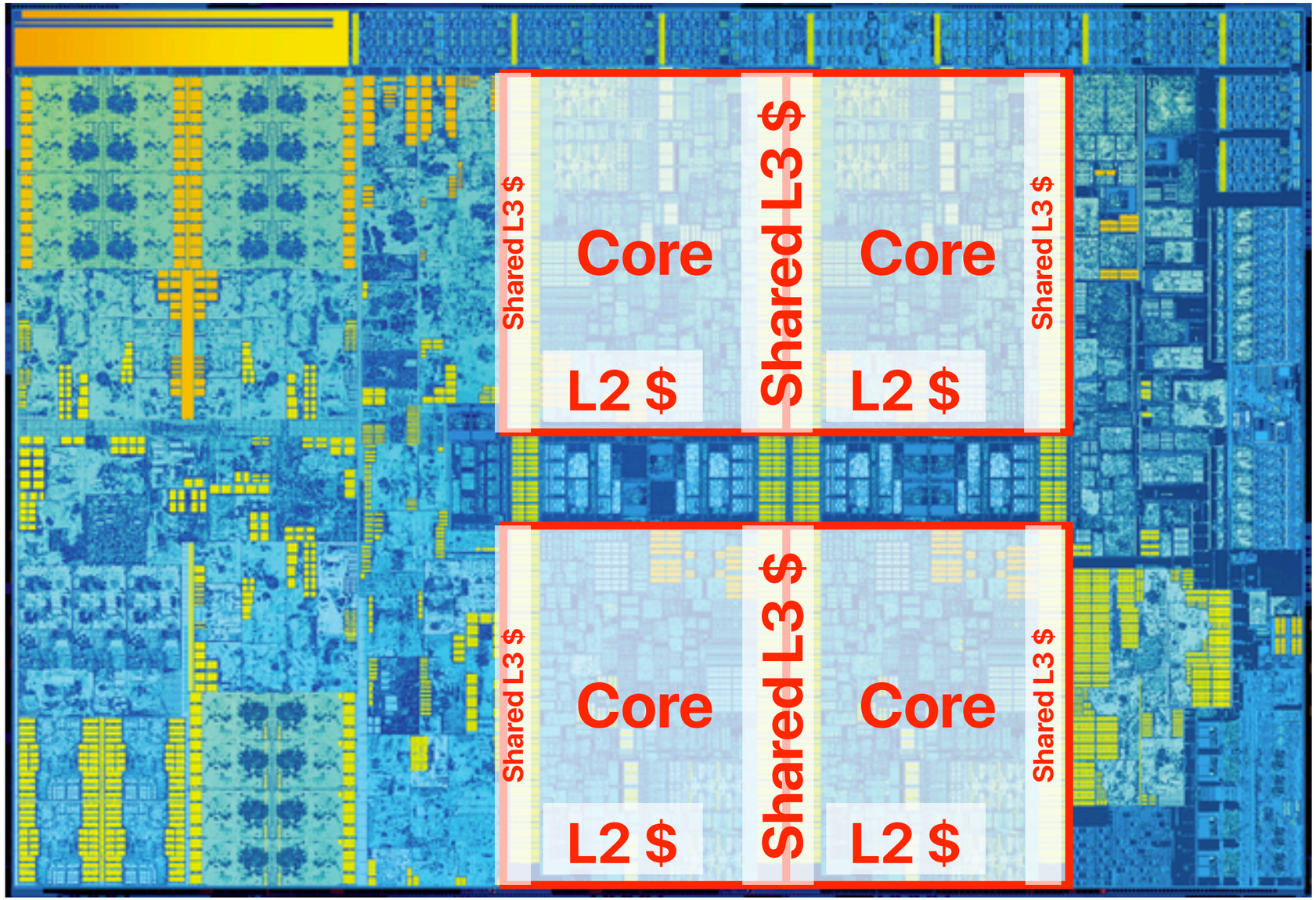
$$Power_{new} = Power_{old} \times (\frac{f_{new}}{f_{old}})^3$$

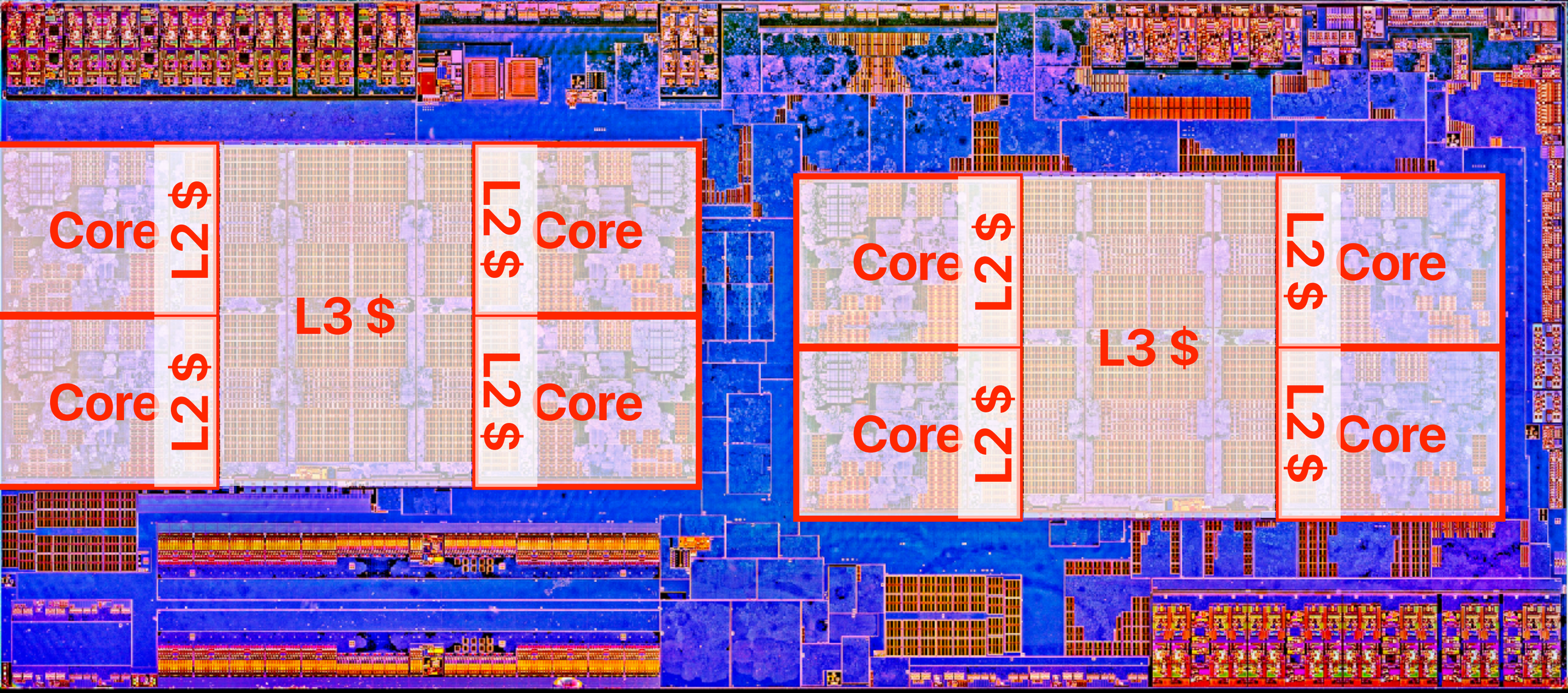
$$Power_{new} = Power_{old} \times (2)^3 = Power_{old} \times 8$$

Doubling the number of cores:

$$Power_{new} = Power_{old} \times number\_of\_cores = Power_{old} \times 2$$

Intel SkyLake

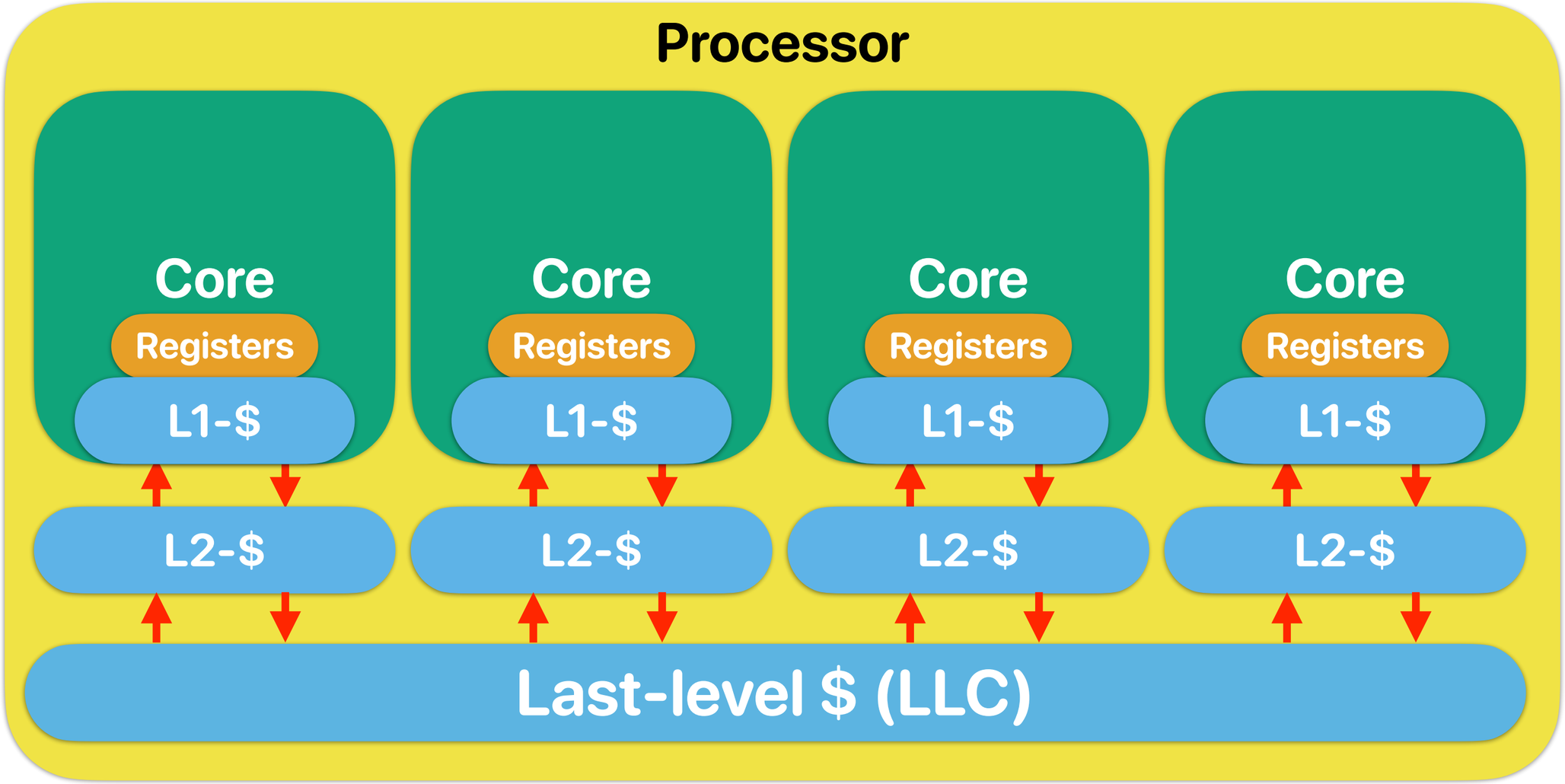



Core
L2 $
L3 $
L2 $
Core
Core
L2 $
L2 $
Core
Core
L2 $
L3 $
L2 $
Core
Core
L2 $
L2 $
Core

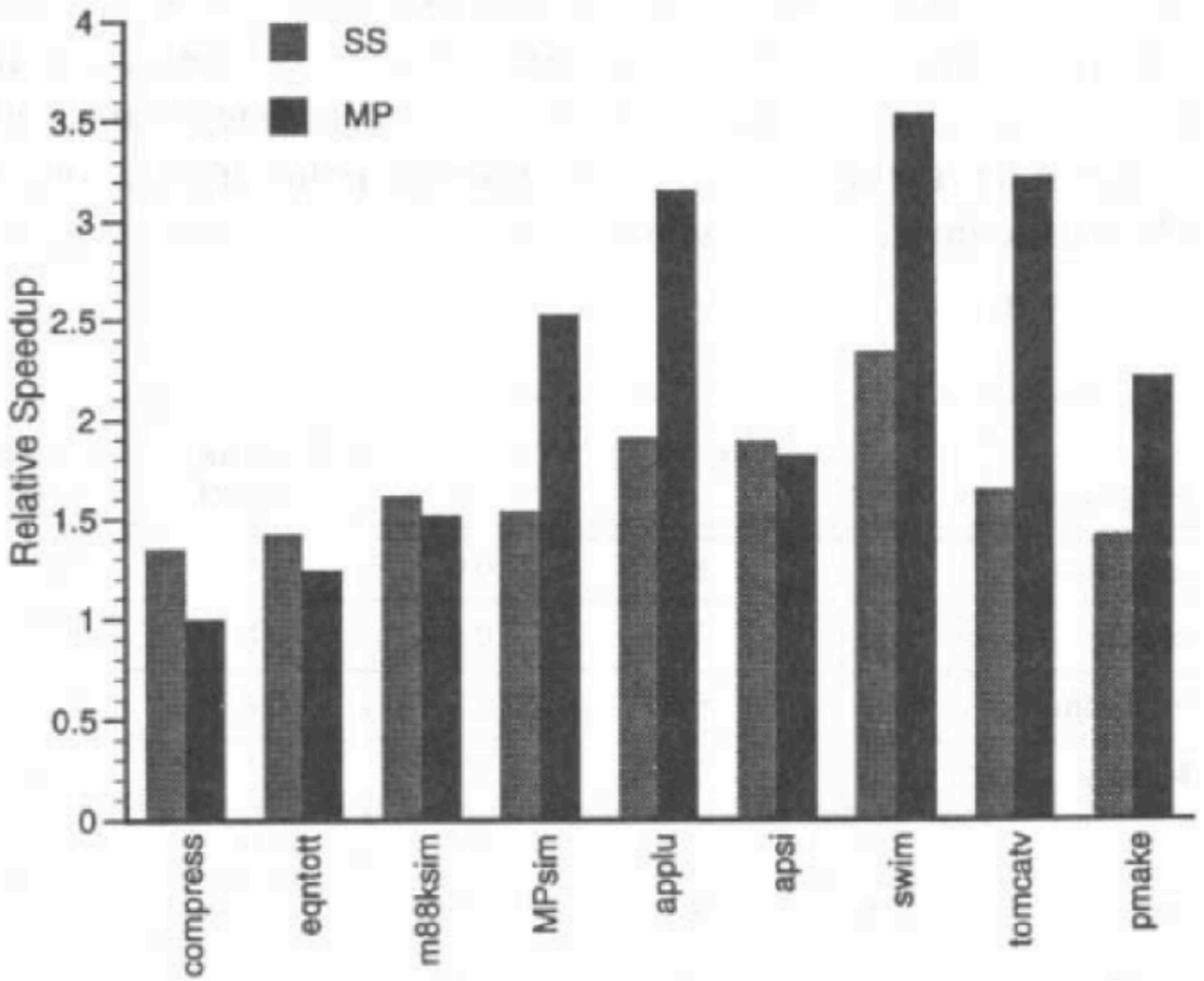
AMD

RYZEN

Concept of CMP



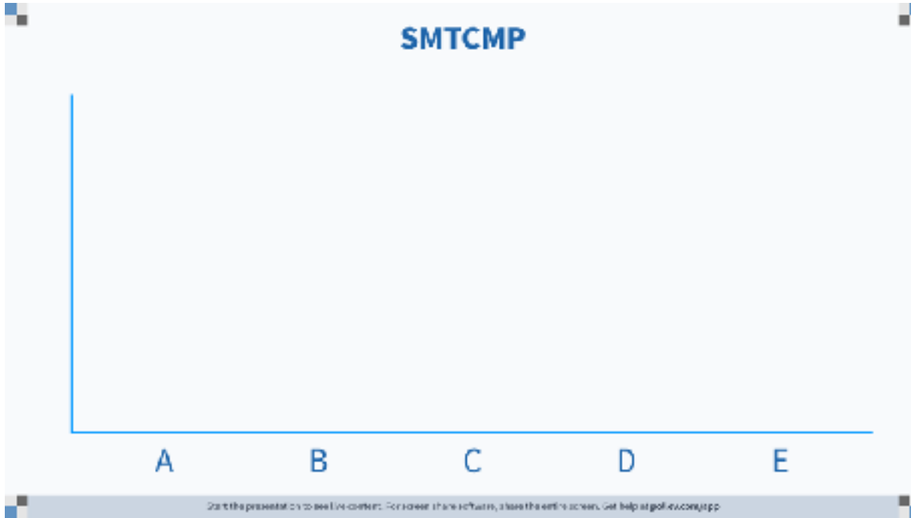
Performance of CMP



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

Helper thread

Helper Thread Prefetching for Loosely-Coupled Multiprocessor Systems*

Changhee Jung[1‡], Daeseob Lim[2‡], Jaejin Lee[3], and Yan Solihin[4†]

```
00 long refresh_potential
01    (network_t * net) {
02  node_t * node, * tmp;
03  ... // some computation
04  while (node != root) {
05   while (node) {
06    if(node->orientation
07      == UP) {
08     node->potential
09      = node->basic_arc->cost
10      + node->pred->potential;
11    } else {
12     node->potential
13      = node->pred->potential
14      - node->basic_arc->cost;
15     checksum++;
16    }
17    tmp = node;
18    node = node->child;
19   }
20   node = tmp;
21   while (node->pred) {
22    tmp = node->sibling;
23    if(tmp) {
24     node = tmp;
25     break;
26    } else
27     node = node->pred;
28   }
29  }
30 }
```

(a)

```
long refresh_potential
   (network_t * net) {
 node_t * node, * tmp;

 while (node != root) {
  while (node) {
   pref(node);
   pref(node->pred);
   pref(node->basic_arc);
   tmp = node
   node = node->child;
  }
  node = tmp;
  while (node->pred) {
   tmp = node->sibling;
   if (tmp) {
    node = tmp;
    break;
   } else
    node = node->pred;
  }
 }
}
```

(b)

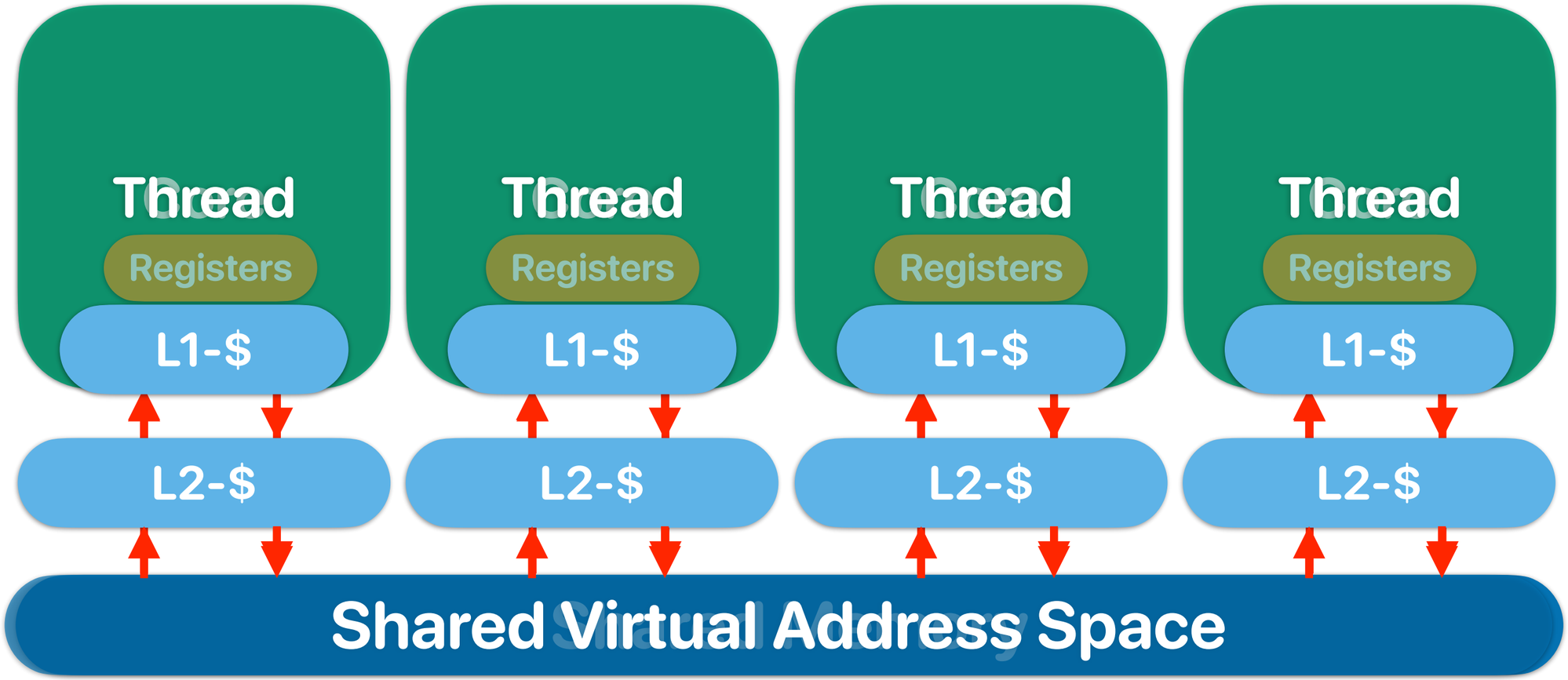
Figure 3. Constructing a prefetching helper thread: (a) the original application thread, and (b) the constructed helper thread. The prefetching section is the `refresh_potential()` subroutine in *mcf*.

Architectural Support for Parallel Programming

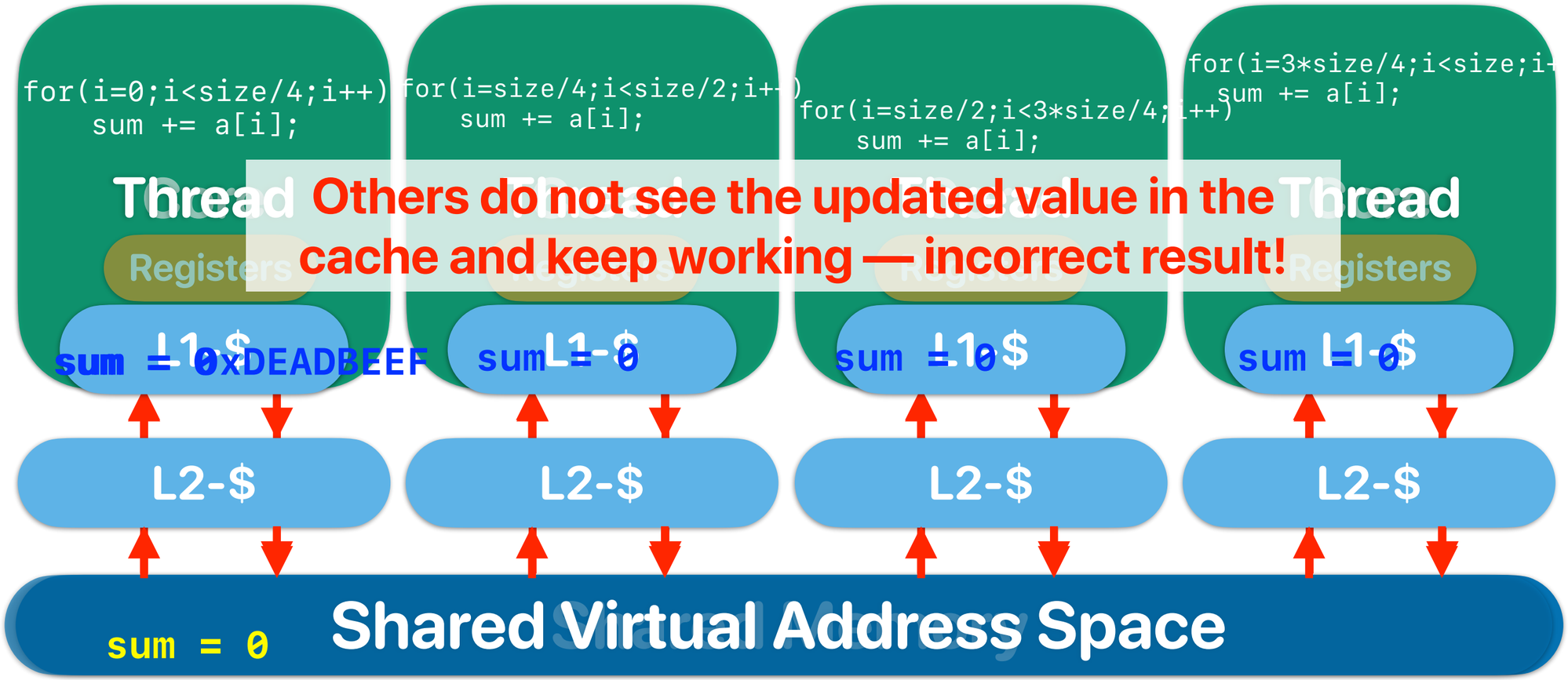
Parallel programming

- 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



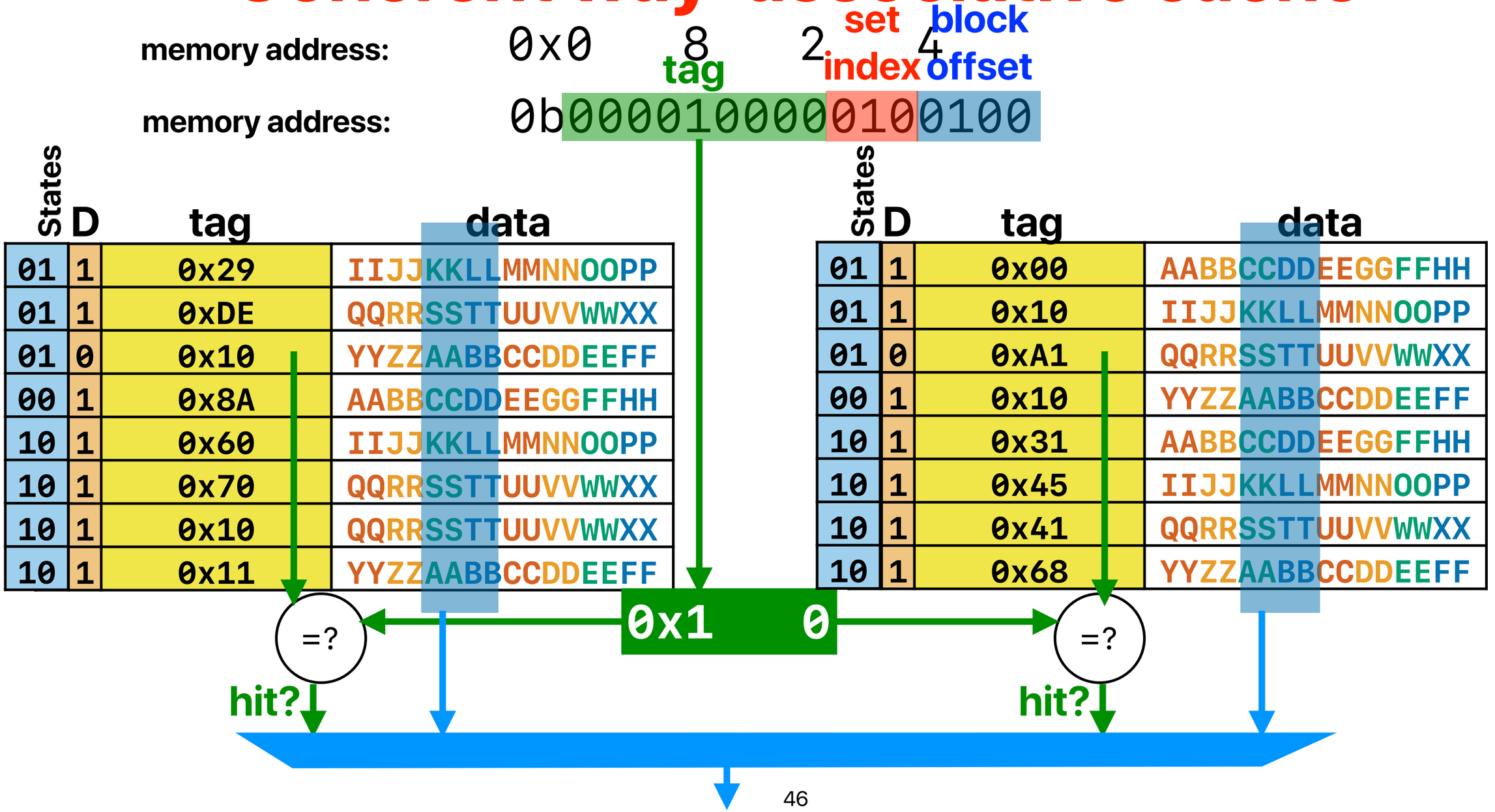
Coherency & Consistency

- 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

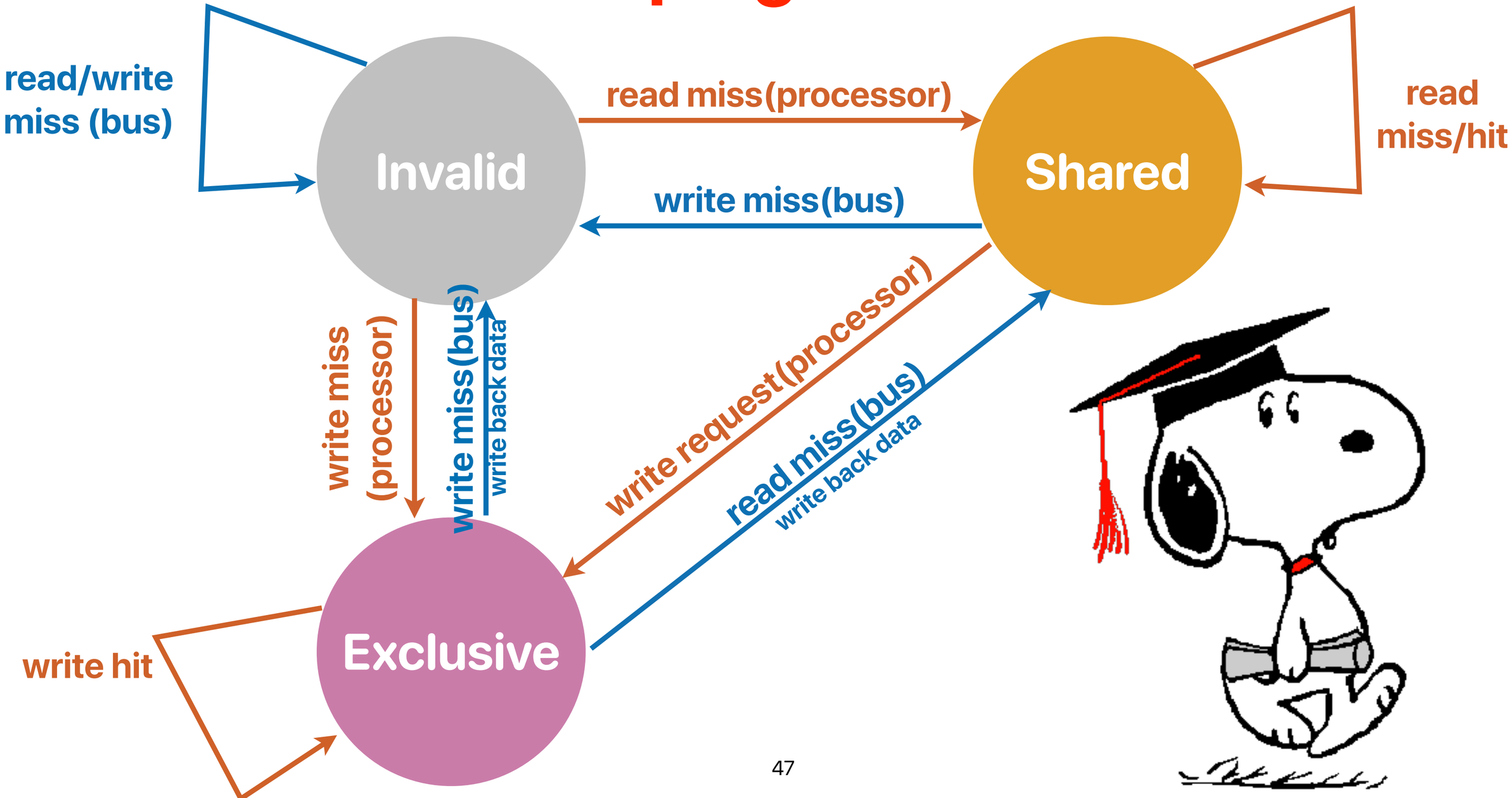
Simple cache coherency protocol

- 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

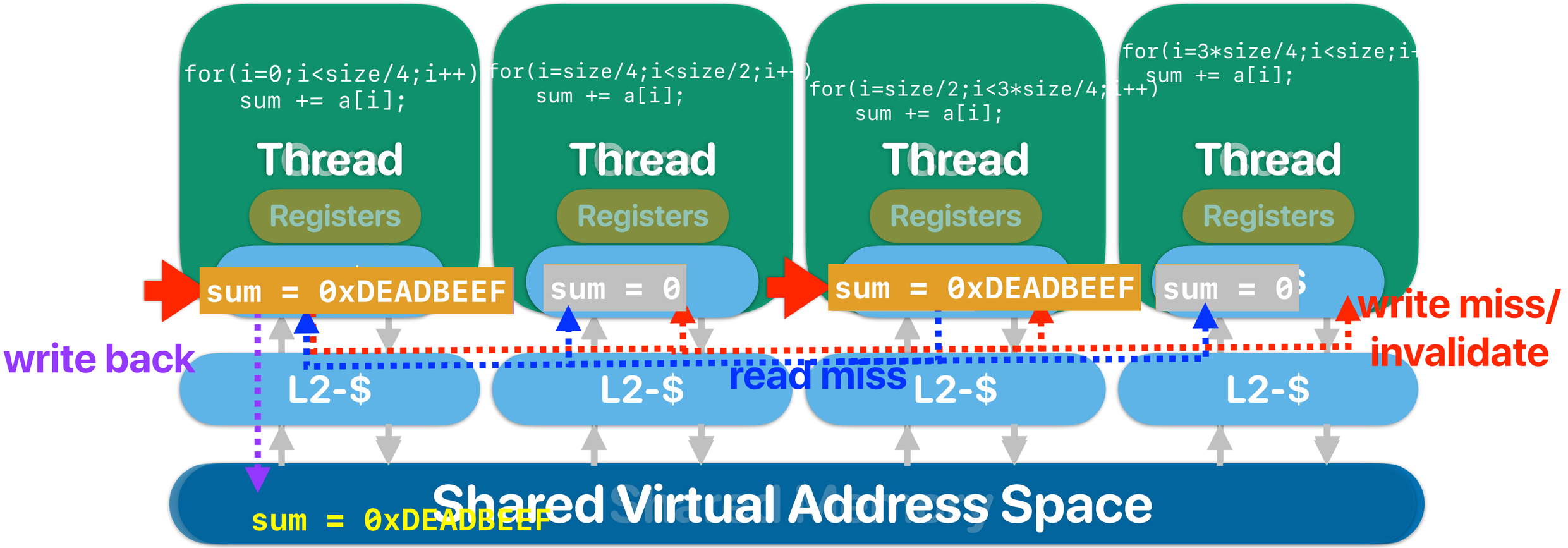
Coherent way-associative cache



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;
}
```

Announcement

- Project due next Monday
- Last reading quiz due next Monday
- Assignment #4 due next Wednesday
- iEVAL, starting tomorrow until 12/3
  - Please fill the survey to let us know your opinion!
  - Don’t forget to take a screenshot of your submission and submit through iLearn — it counts as a **full credit assignment**
  - **We will drop your lowest 2 assignment grades**
- Final Exam
  - Starting from 12/6 to 12/10 12:00pm, any consecutive 180 minutes you pick
  - Similar to the midterm, but more time and about 1.5x longer
  - Two of the questions will be comprehensive exam questions
  - Will release a sample final at the end of the last lecture

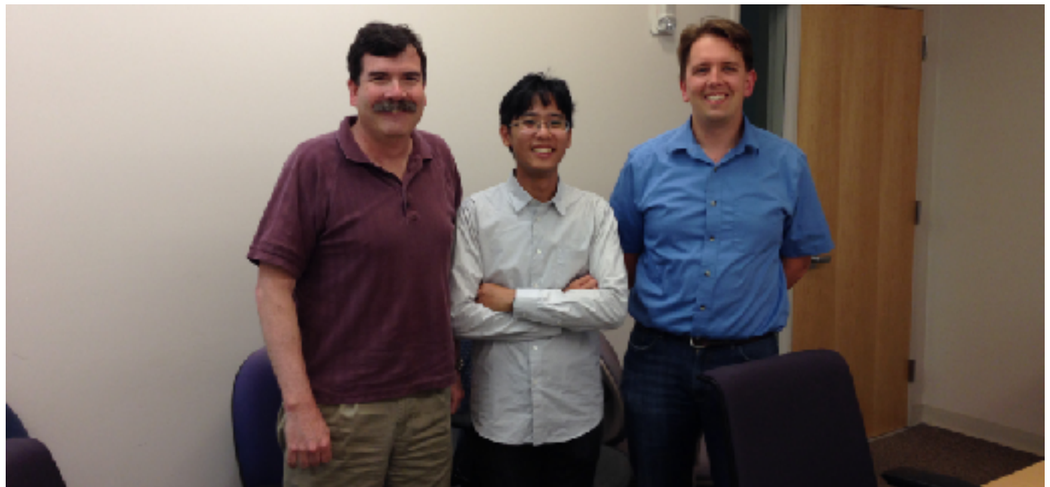
Computer
Science &
Engineering

203

つづく

Simultaneous multithreading: maximizing on-chip parallelism

Dean M. Tullsen, Susan J. Eggers, Henry M. Levy
Department of Computer Science and Engineering, University of Washington

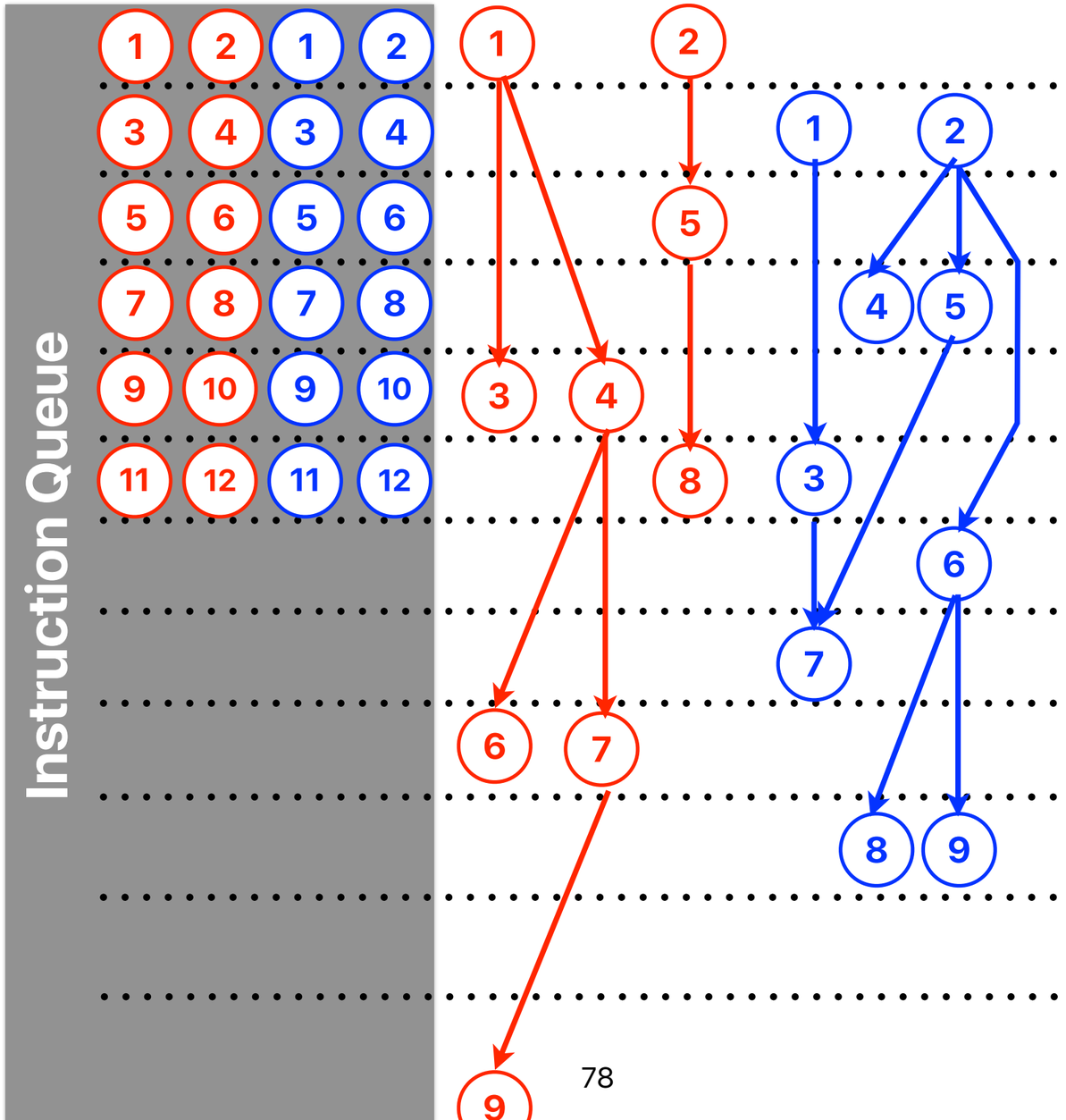


Simultaneous multithreading

- The processor can schedule instructions from different threads/processes/programs
- Fetch instructions from different threads/processes to fill the not utilized part of pipeline
  - Exploit “thread level parallelism” (TLP) to solve the problem of insufficient ILP in a single thread
  - You need to create an illusion of multiple processors for OSs

Simultaneous multithreading

```
① ld   X10, 8(X10)
② addi  X7, X7, 1
③ bne  X10, X0, LOOP
④ ld   X10, 8(X10)
⑤ addi  X7, X7, 1
⑥ bne  X10, X0, LOOP
⑦ ld   X10, 8(X10)
⑧ addi  X7, X7, 1
⑨ bne  X10, X0, LOOP
```




```
① ld   X1, 0(X10)
② addi X10, X10, 8
③ add  X20, X20, X1
④ bne  X10, X2, LOOP
⑤ ld   X1, 0(X10)
⑥ addi X10, X10, 8
⑦ add  X20, X20, X1
⑧ bne  X10, X2, LOOP
⑨ ld   X1, 0(X10)
⑩ addi X10, X10, 8
⑪ add  X20, X20, X1
⑫ bne  X10, X2, LOOP
```

Poll close in 1:30

Architectural support for simultaneous multithreading

- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  - ① Program counter
  - ② Register mapping tables
  - ③ Physical registers
  - ④ ALUs
  - ⑤ Data cache
  - ⑥ Reorder buffer/Instruction Queue

  A. 2
  B. 3
  C. 4
  D. 5
  E. 6

Poll close in 1:30

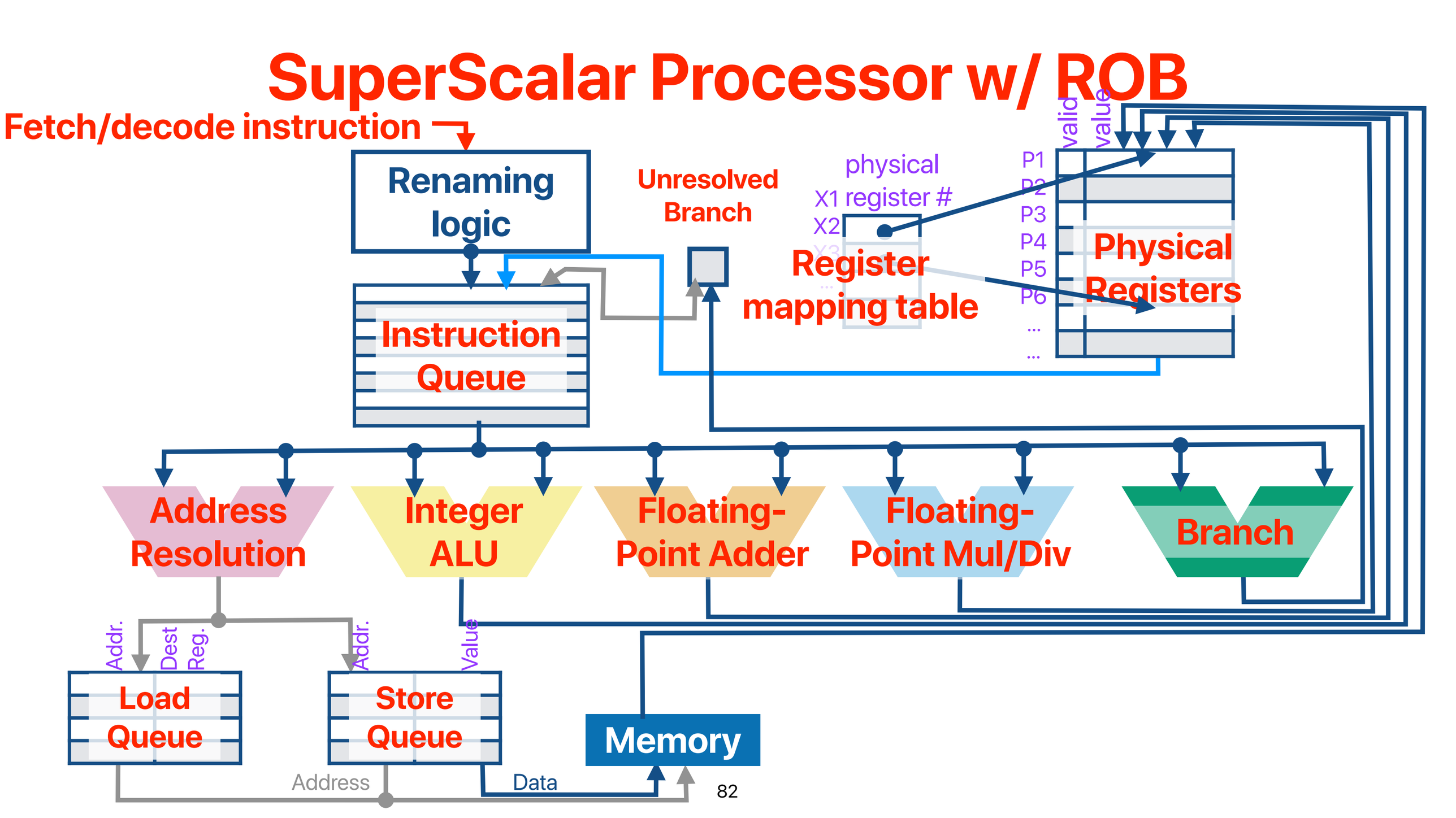
Architectural support for simultaneous multithrea

- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  - ① Program counter
  - ② Register mapping tables
  - ③ Physical registers
  - ④ ALUs
  - ⑤ Data cache
  - ⑥ Reorder buffer/Instruction Queue
  - A. 2
  - B. 3
  - C. 4
  - D. 5
  - E. 6

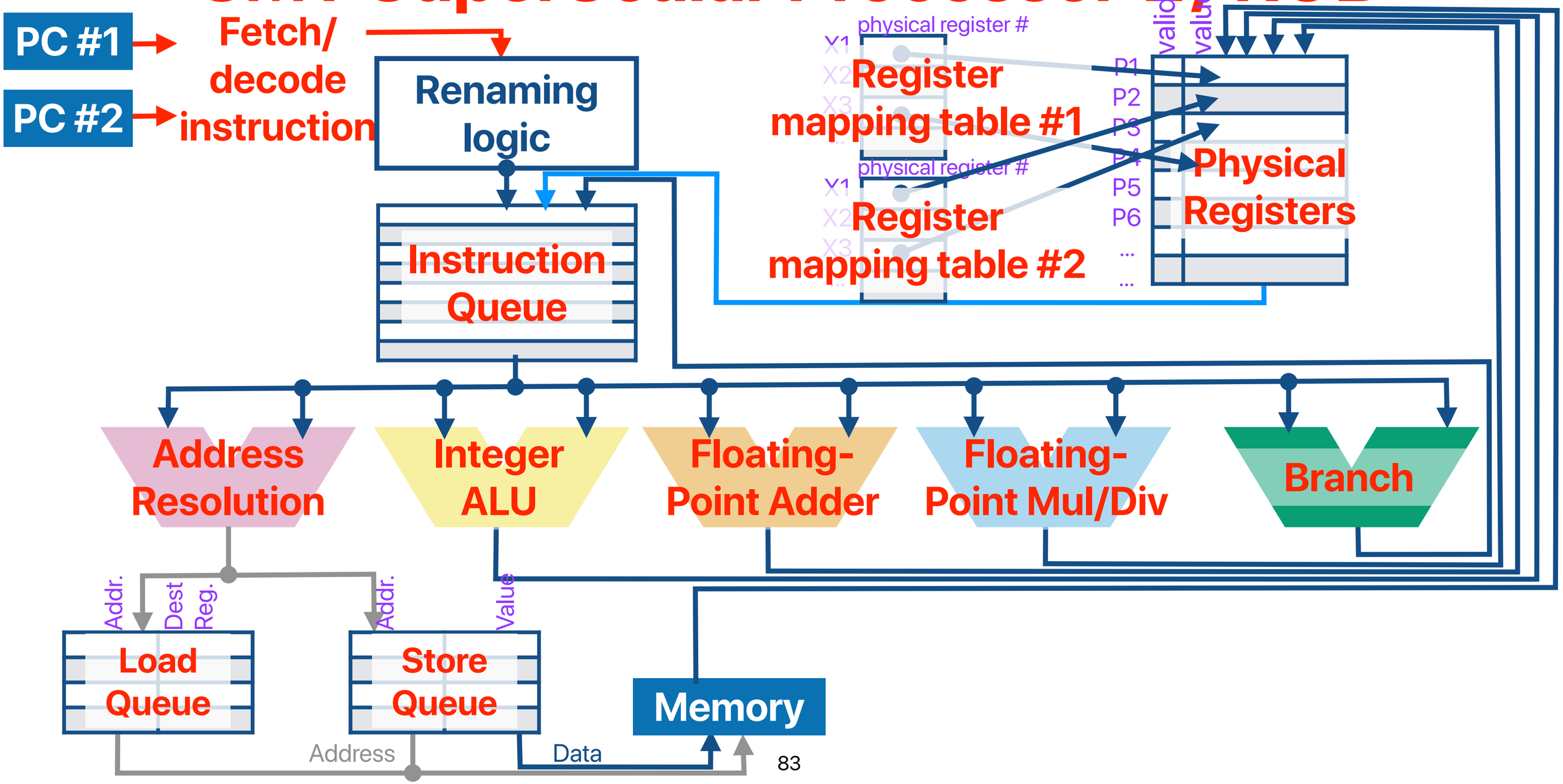
Architectural support for simultaneous multithreading

- To create an illusion of a multi-core processor and allow the core to run instructions from multiple threads concurrently, how many of the following units in the processor must be duplicated/extended?
  - ① Program counter — **you need to have one for each context**
  - ② Register mapping tables — **you need to have one for each context**
  - ③ Physical registers — **you can share**
  - ④ ALUs — **you can share**
  - ⑤ Data cache — **you can share**
  - ⑥ Reorder buffer/Instruction Queue — **you need to indicate which context the instruction is from**
  - A. 2
  - B. 3
  - C. 4
  - D. 5
  - E. 6

SuperScalar Processor w/ ROB



SMT SuperScalar Processor w/ ROB



Poll close in 1:30

SMT

- How many of the following about SMT are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can improve the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

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

Poll close in 1:30

SMT

- How many of the following about SMT are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can improve the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

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

SMT

- How many of the following about SMT are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can ~~improve~~ the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.

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

We can execute from other threads/contexts instead of the current one

hurt, b/c you are sharing resource with other threads.

We can execute from other threads/ contexts instead of the current one

b/c we're sharing the cache