Multithreaded Architectures and Programming on Multithreaded Architectures

Hung-Wei Tseng

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

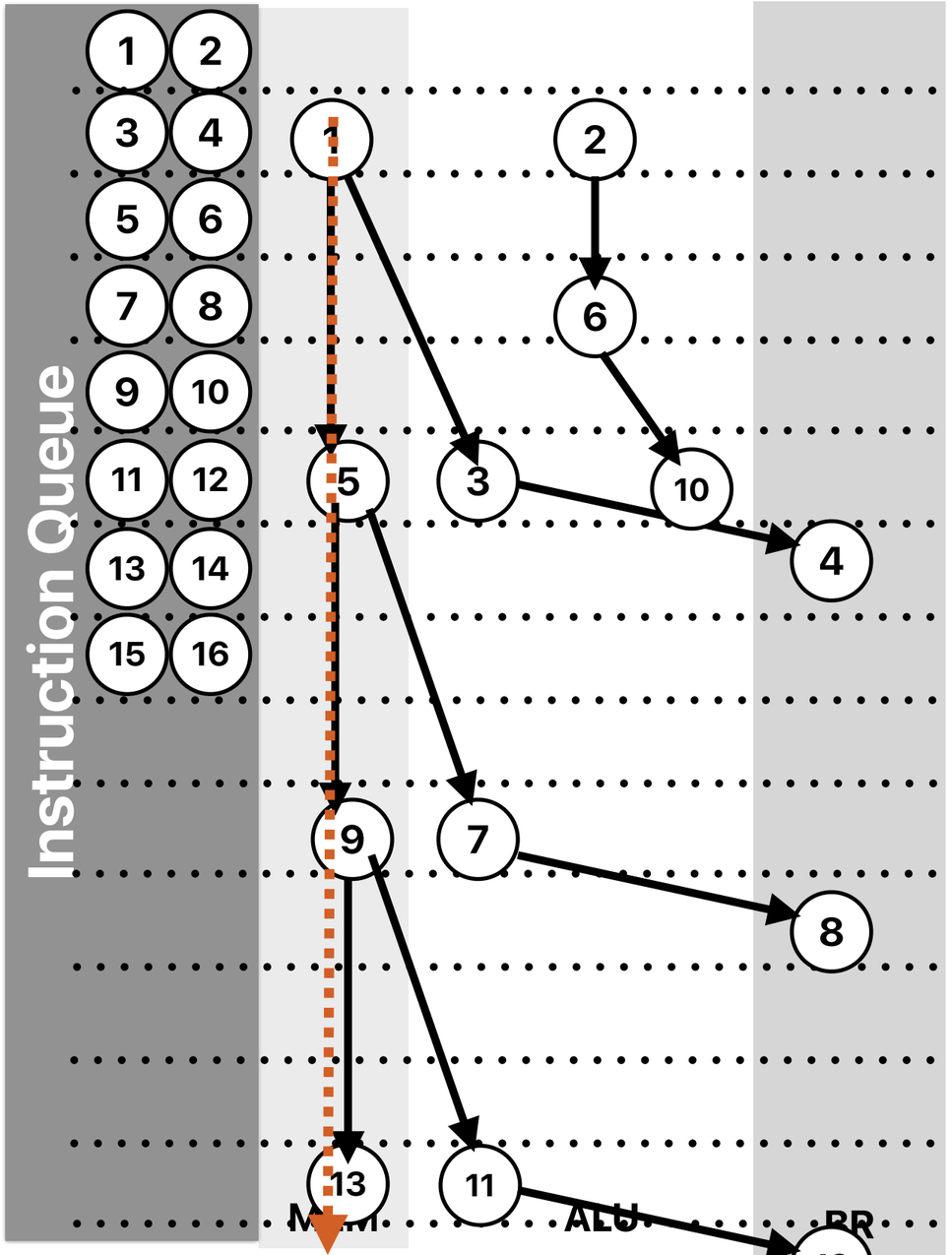
- Minimize the critical path operations
  - Don’t forget about optimizing cache/memory locality first!
    - Memory latencies are still way longer than any arithmetic instruction
    - Can we use arrays/hash tables instead of lists?
  - Branch can be expensive as pipeline get deeper
    - Sorting
    - 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

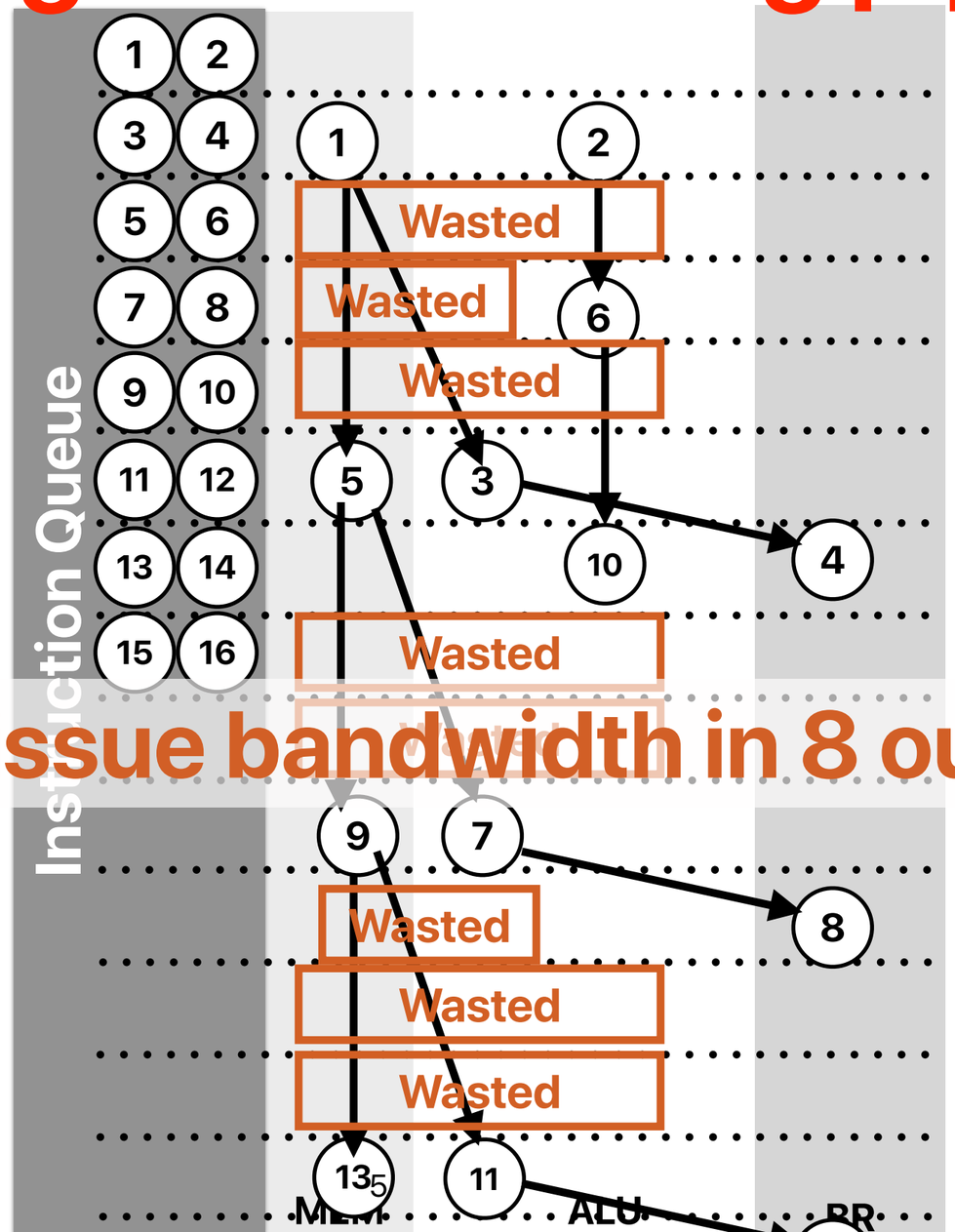
```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL );
```

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



2-issue register renaming pipeline

```
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
```



We're wasting the issue bandwidth in 8 out of 12 cycles

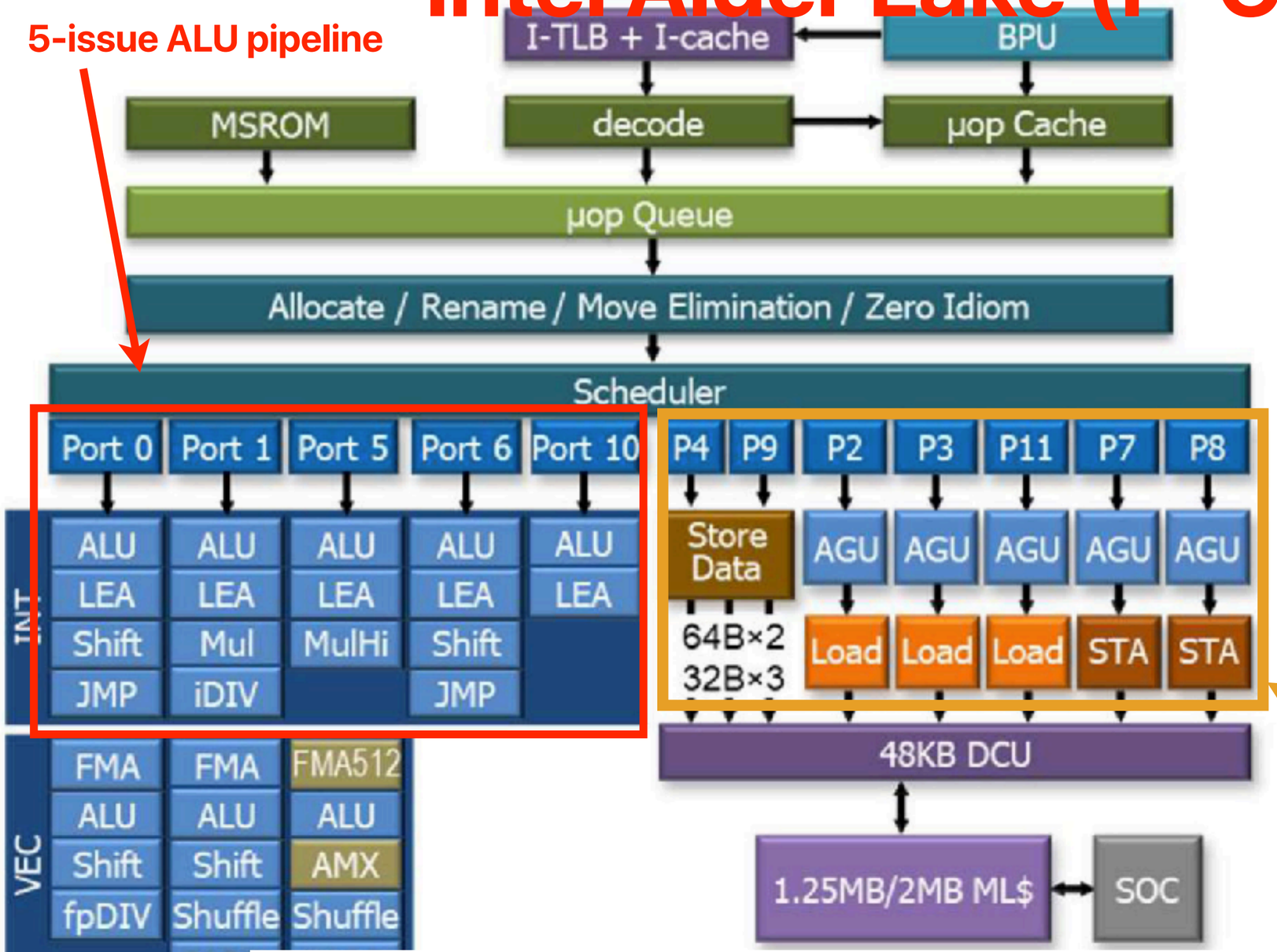
Intel Alder Lake (P-Core)

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

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

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

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



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

SMT

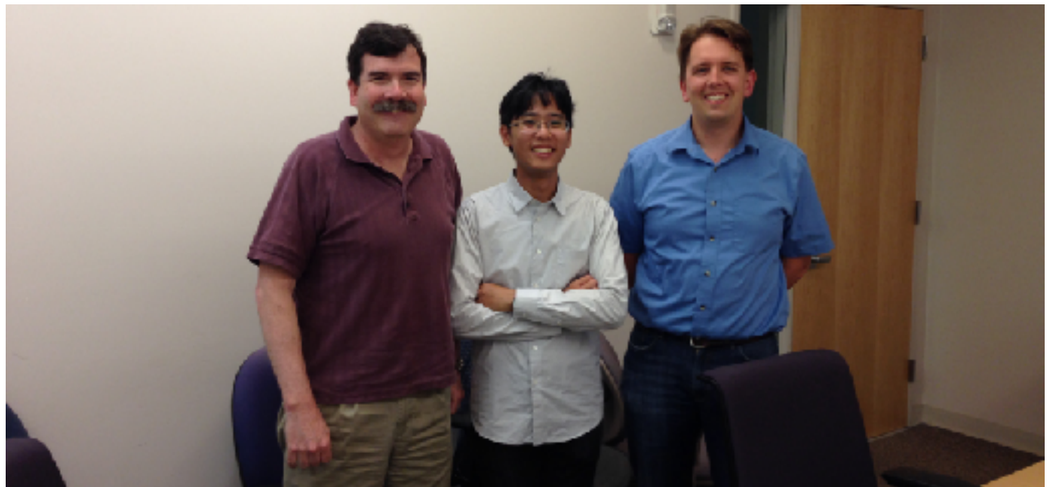
- Improve the throughput of execution
  - May increase the latency of a single thread
- Less branch penalty per thread
- Increase hardware utilization
- Simple hardware design: Only need to duplicate PC/Register Files
- Real Case:
  - Intel HyperThreading (supports up to two threads per core)
    - Intel 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!

```
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
```

Outline

- Multithreaded processors
- Programming multithreaded processors & necessary architectural support

Simultaneous multithreading

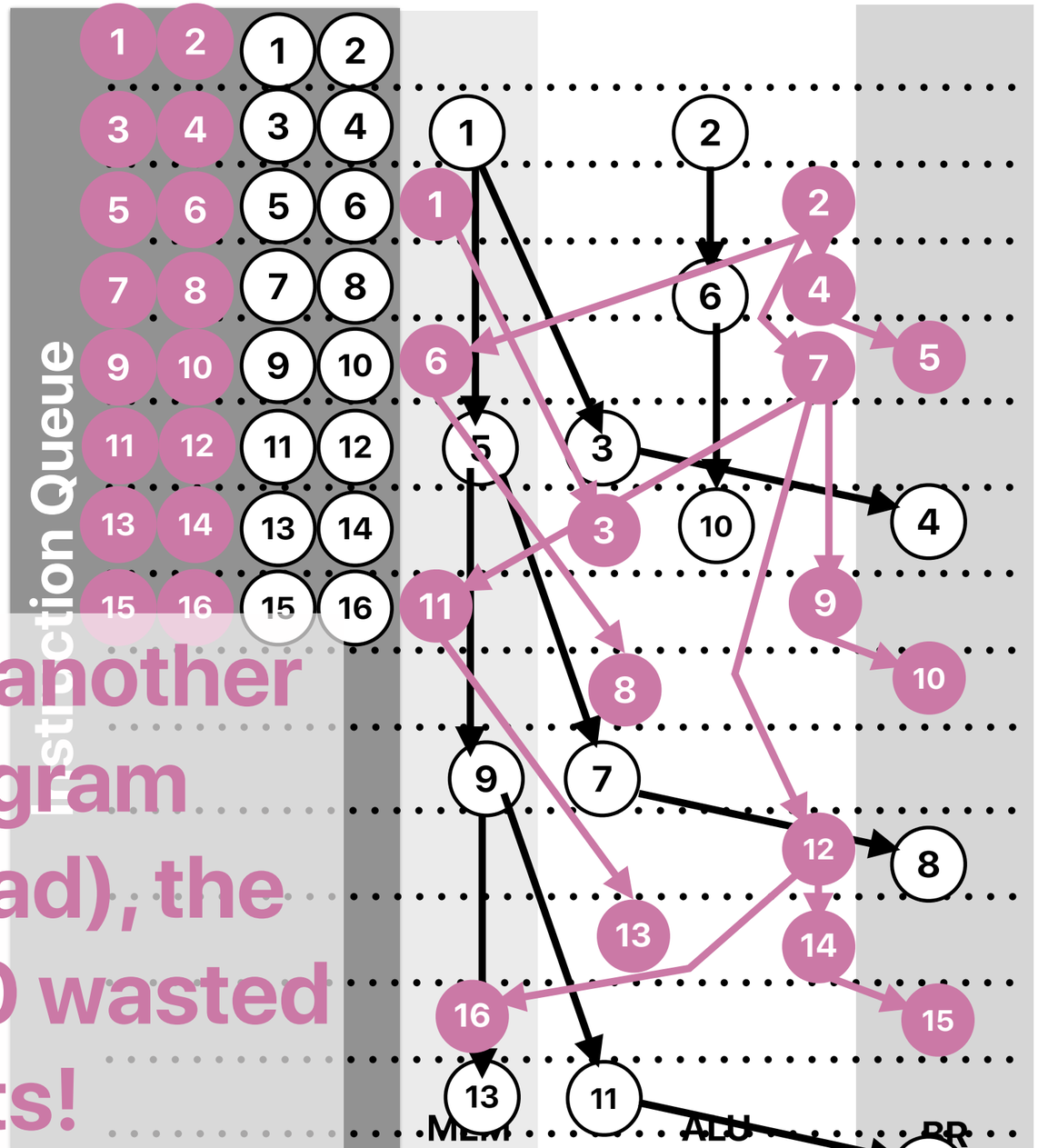


Simultaneous multithreading

- Invented by Dean Tullsen (Now a professor at **UCSD CSE**)
- 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

Concept: Simultaneous Multithreading (SMT)

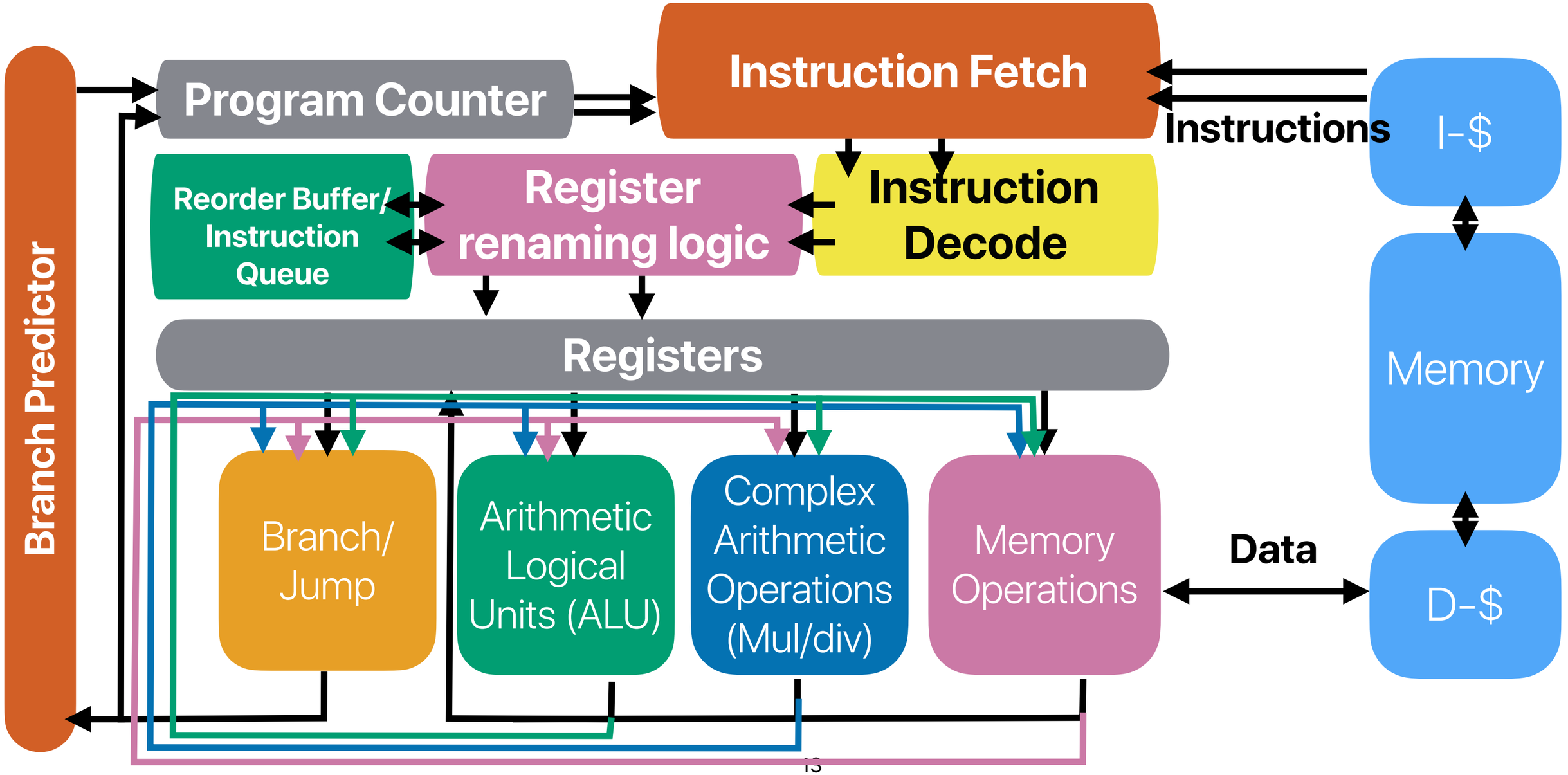
```
①  movq    8(%rdi), %rdi
②  addl    $1, %eax
③  testq   %rdi, %rdi
④  jne     .L3
⑤  movq    8(%rdi), %rdi
⑥  addl    $1, %eax
⑦  testq   %rdi, %rdi
⑧  jne     .L3
⑨  movq    8(%rdi), %rdi
⑩  addl    $1, %eax
⑪  testq   %rdi, %rdi
⑫  jne     .L3
```



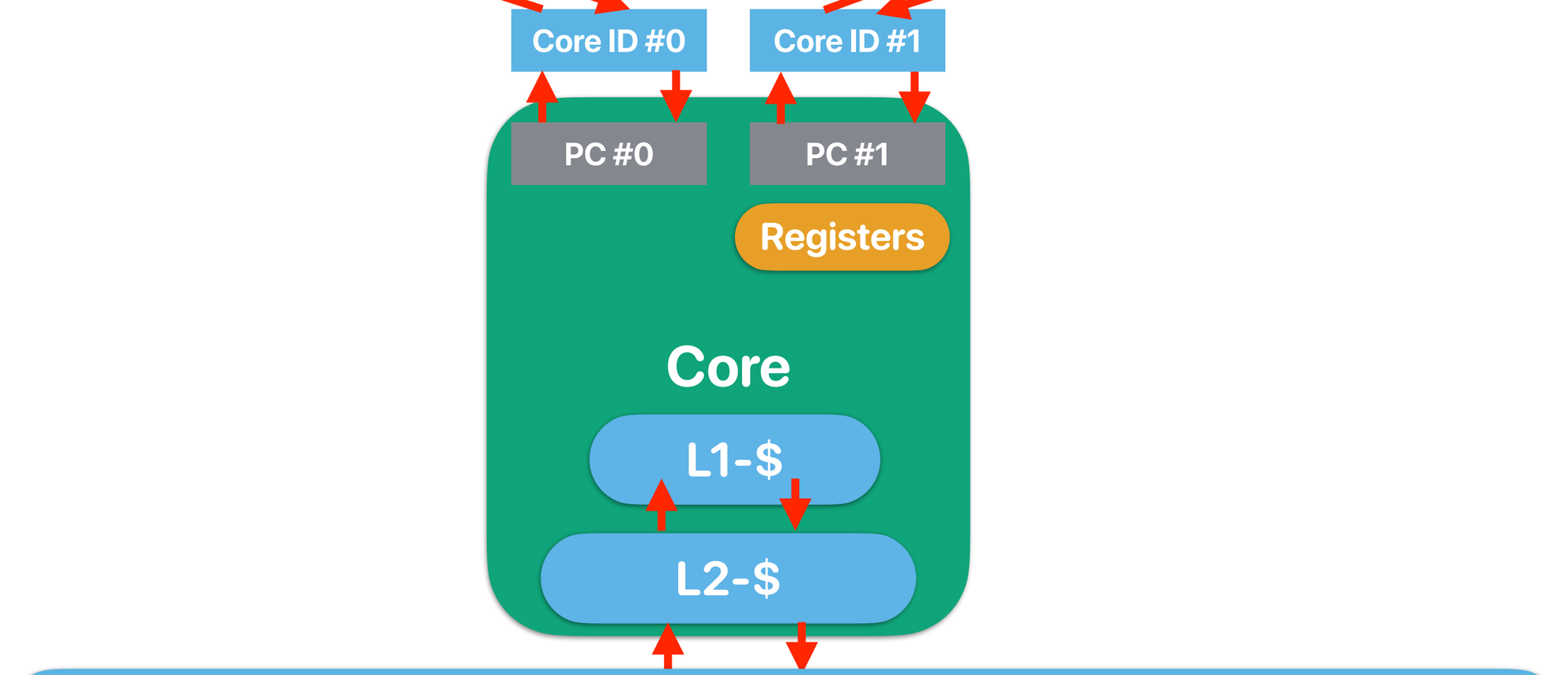
By scheduling another running program instance (thread), the processor has 0 wasted issue slots!

```
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: Register renaming



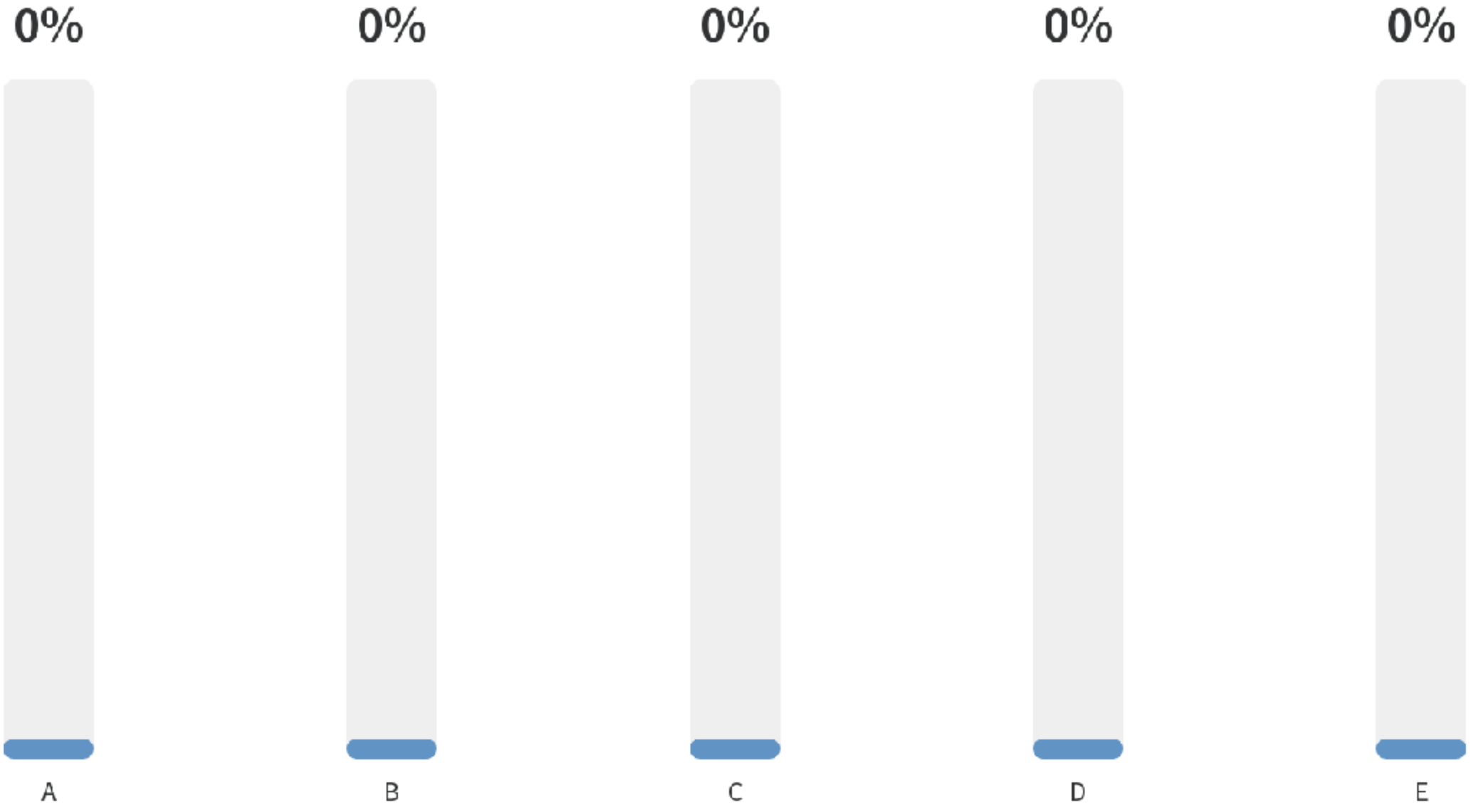
SMT from the user/OS' perspective



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

0



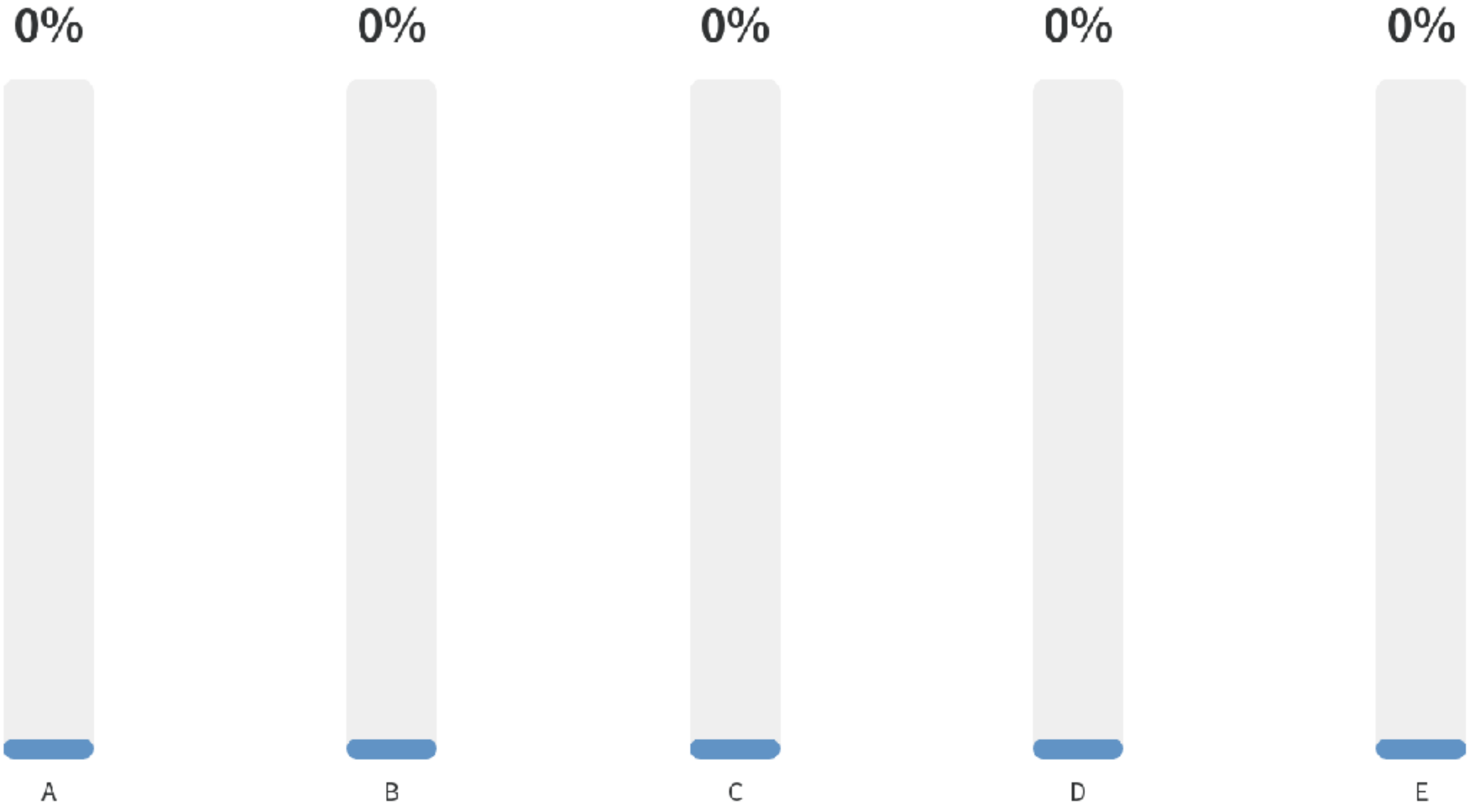
https://www.pollev.com/hungweitseng close in 1:30

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

0



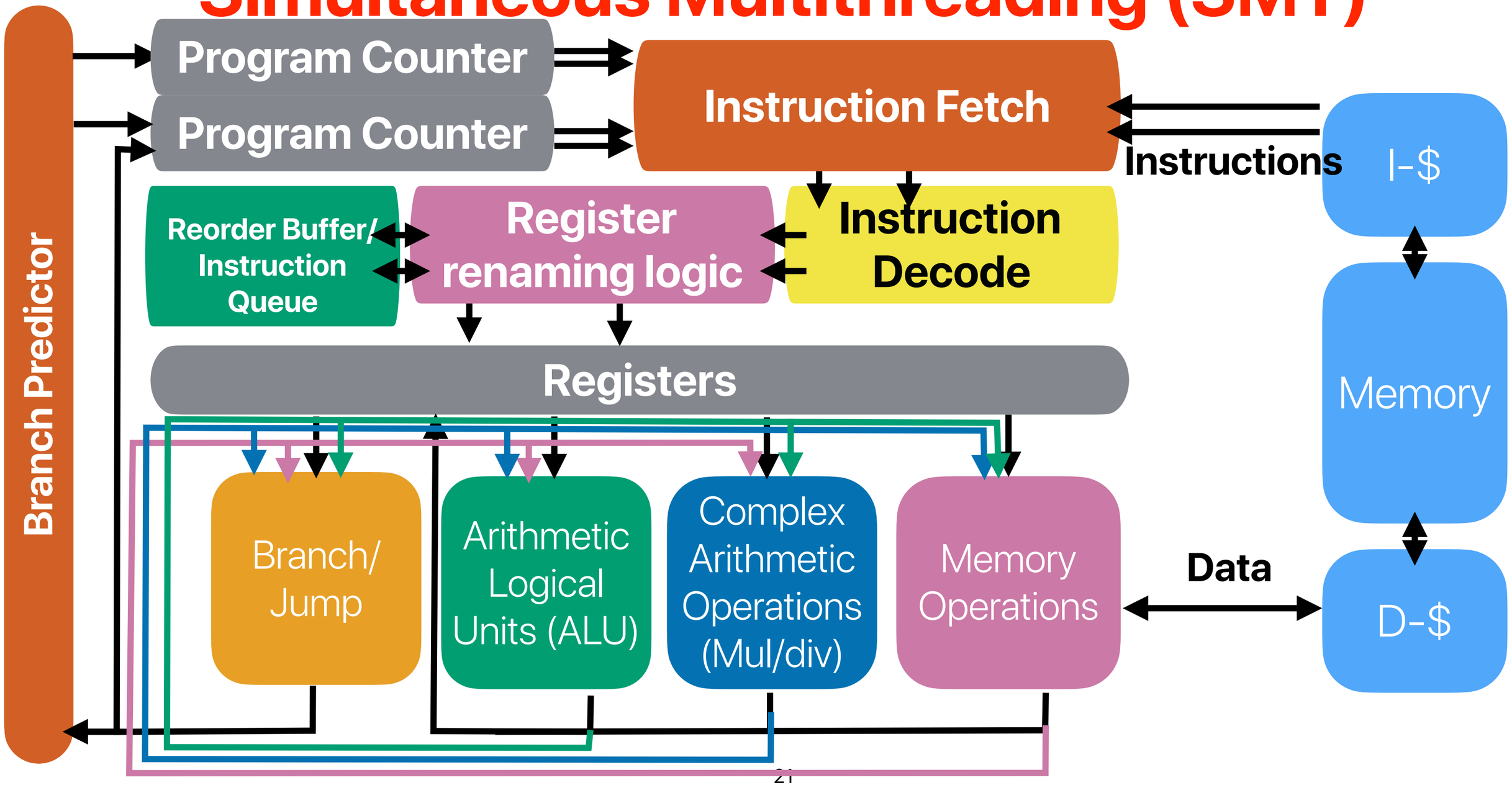
Architectural support for simultaneous multithreading

- 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

How do we support two running programs in one pipeline?

- We need two program counters
- We need two sets of architectural to physical register mappings
- We do not need
  - Duplicated cache — virtually indexed, physically tagged cache already addressed that
  - Duplicated pipeline functional units — isn’t sharing the whole purpose?
  - Duplicated reorder buffer — you simply need to tag which process the instruction belongs to

Simultaneous Multithreading (SMT)



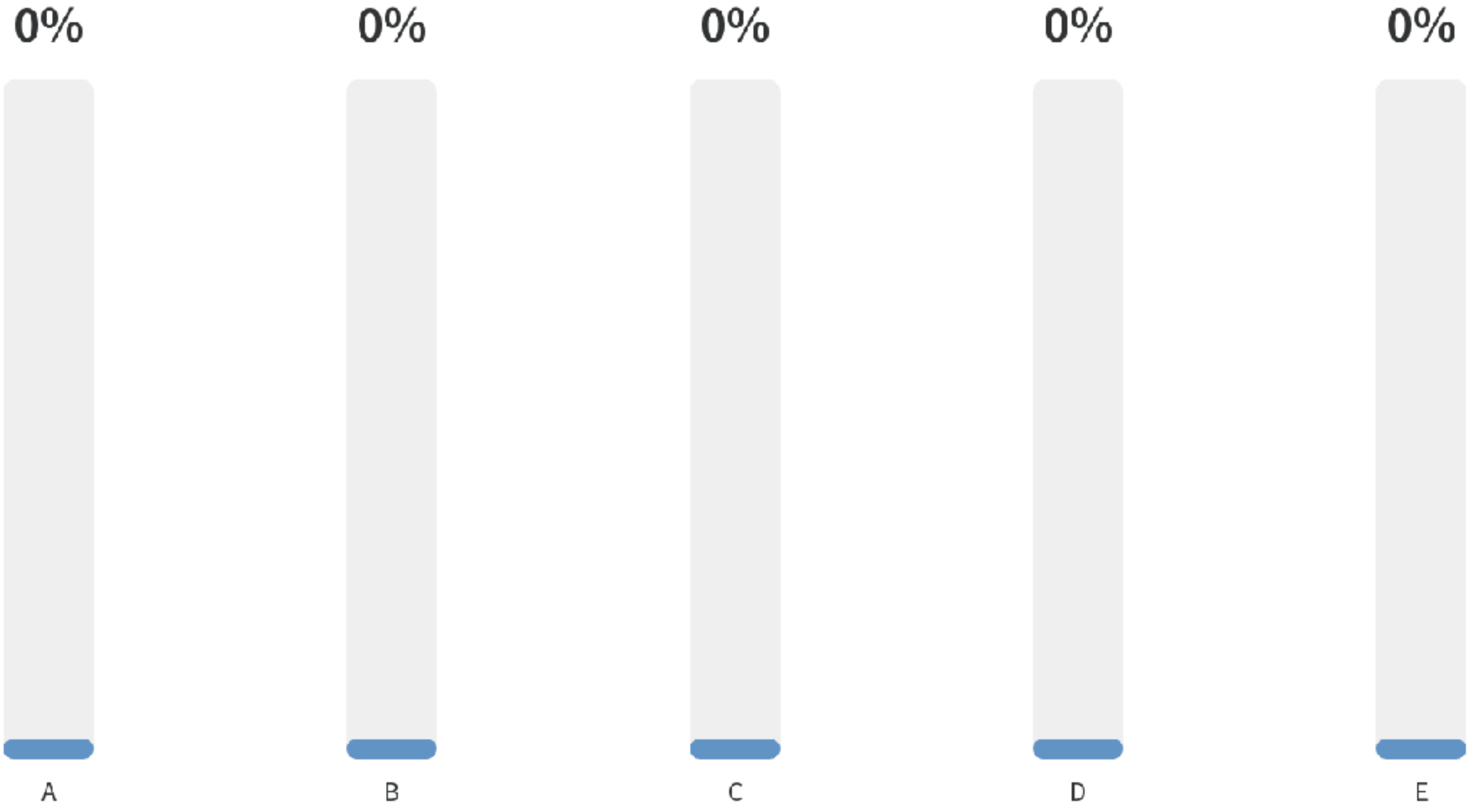
https://www.pollev.com/hungweitseng close in 1:30

Pros/Cons of SMT

- How many of the following statements about SMT is/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

0



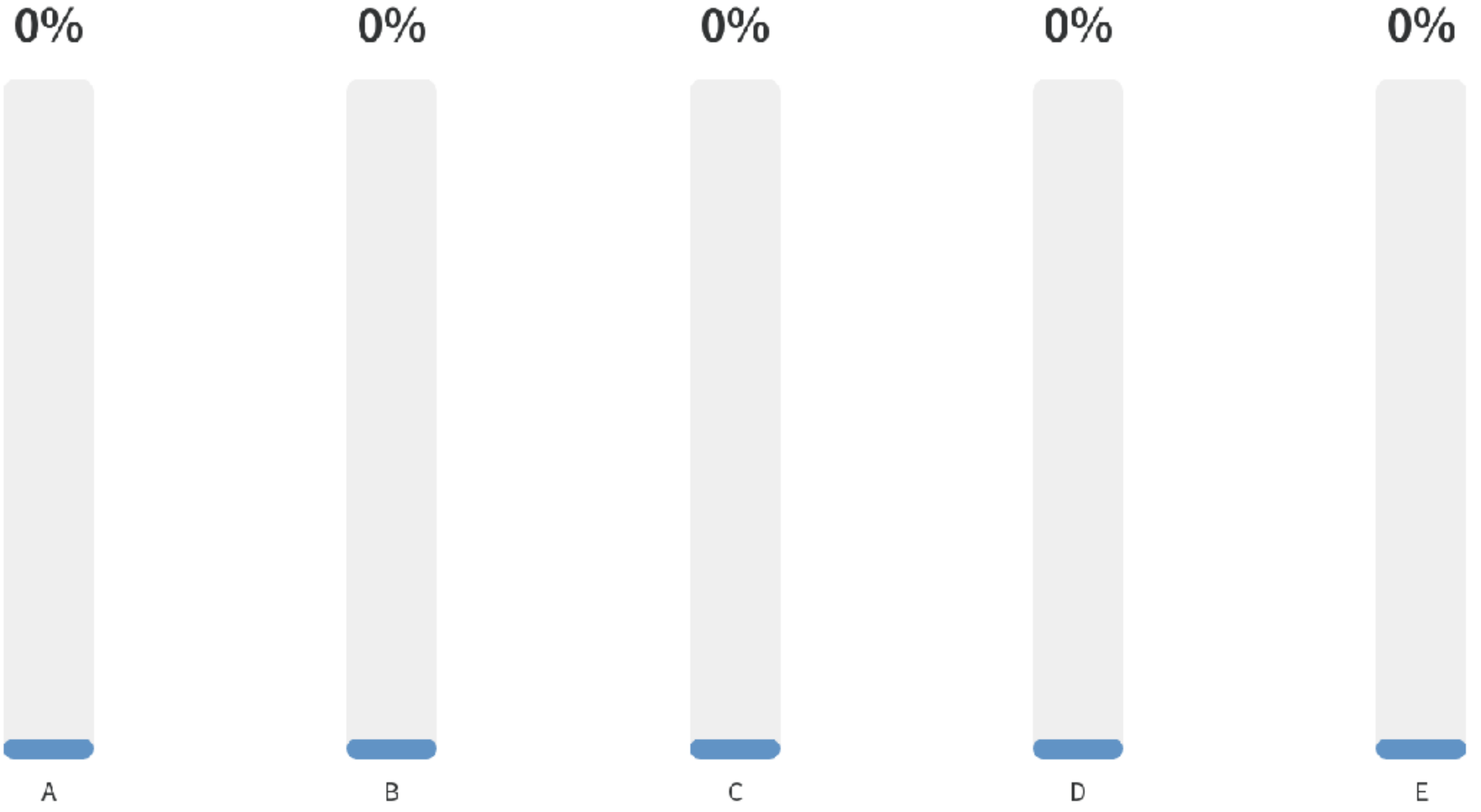
https://www.pollev.com/hungweitseng close in 1:30

Pros/Cons of SMT

- How many of the following statements about SMT is/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

0



Pros/Cons of SMT

- How many of the following statements about SMT is/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

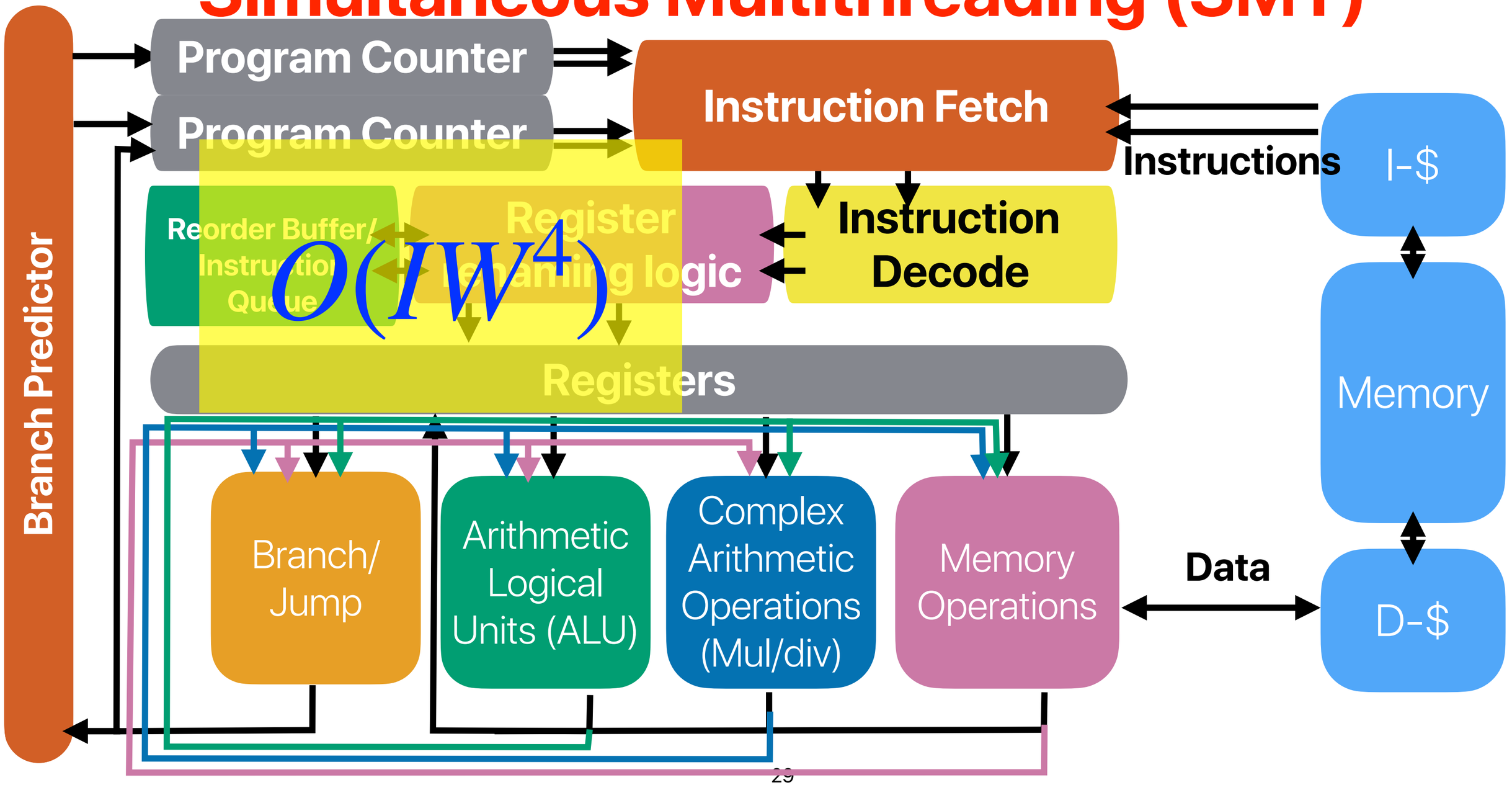
b/c we're sharing the cache

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)
  - If you see a processor with “threads” more than “cores”, that must because of SMT!

```
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
```

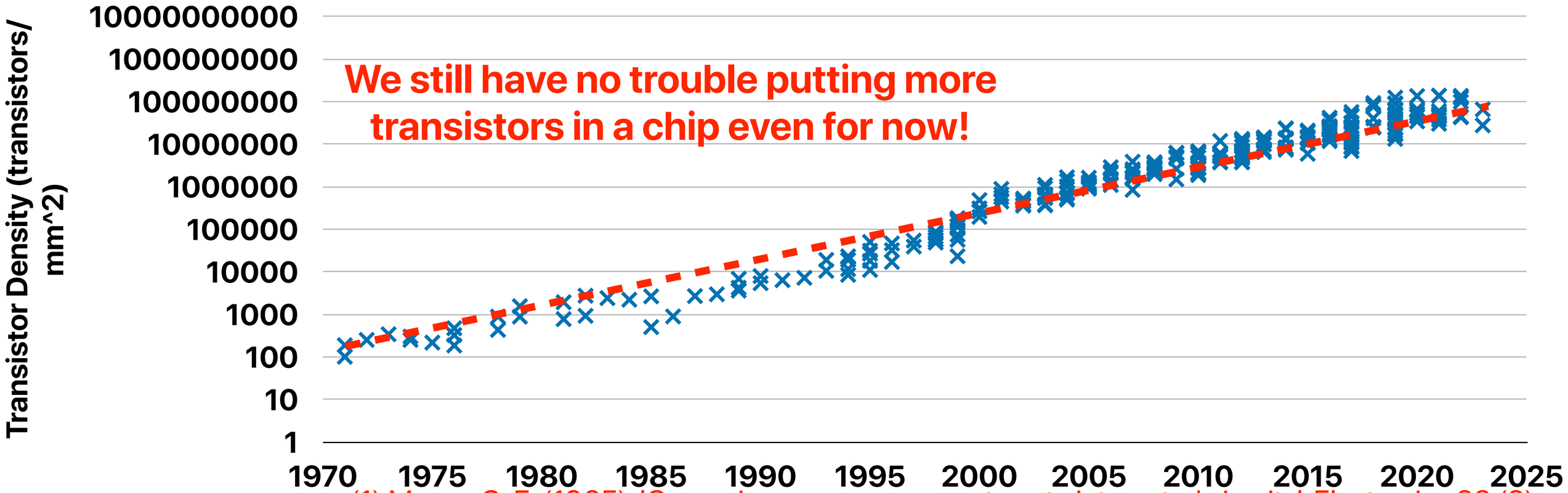
Simultaneous Multithreading (SMT)



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

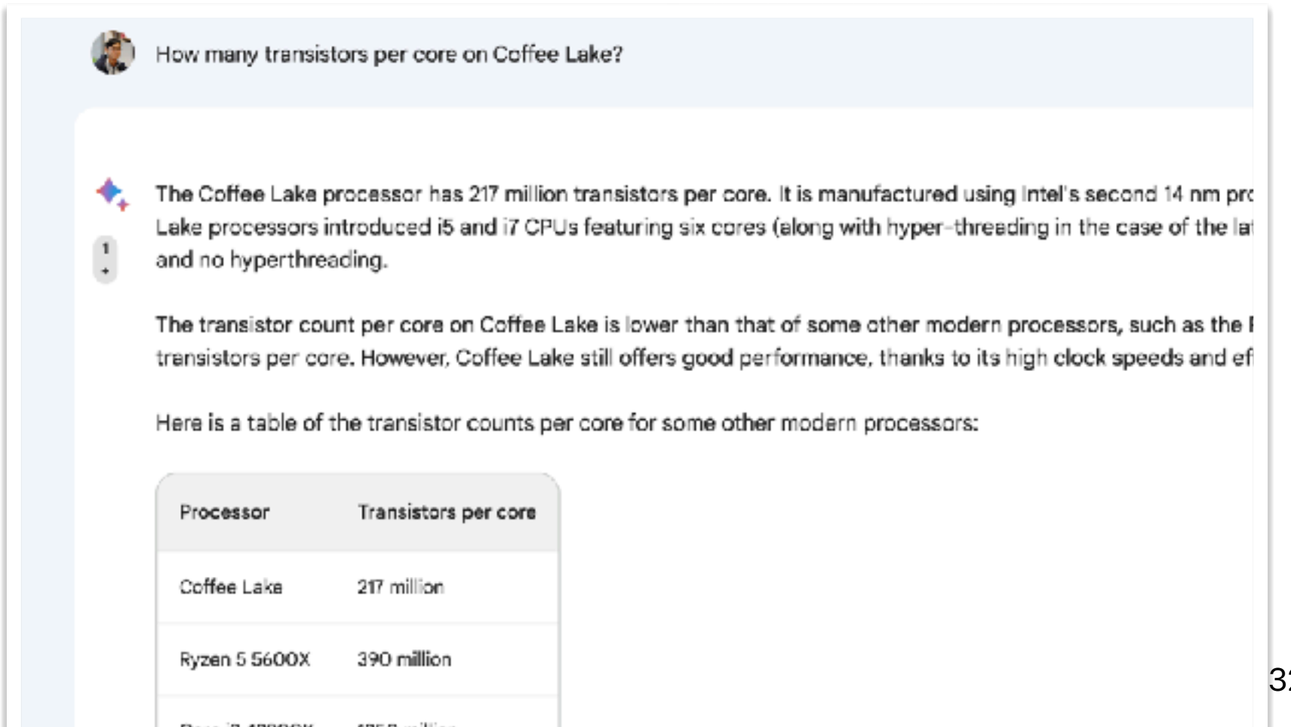


(1) Moore, G. E. (1965), 'Cramming more components onto integrated circuits', Electronics 38 (8) .

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

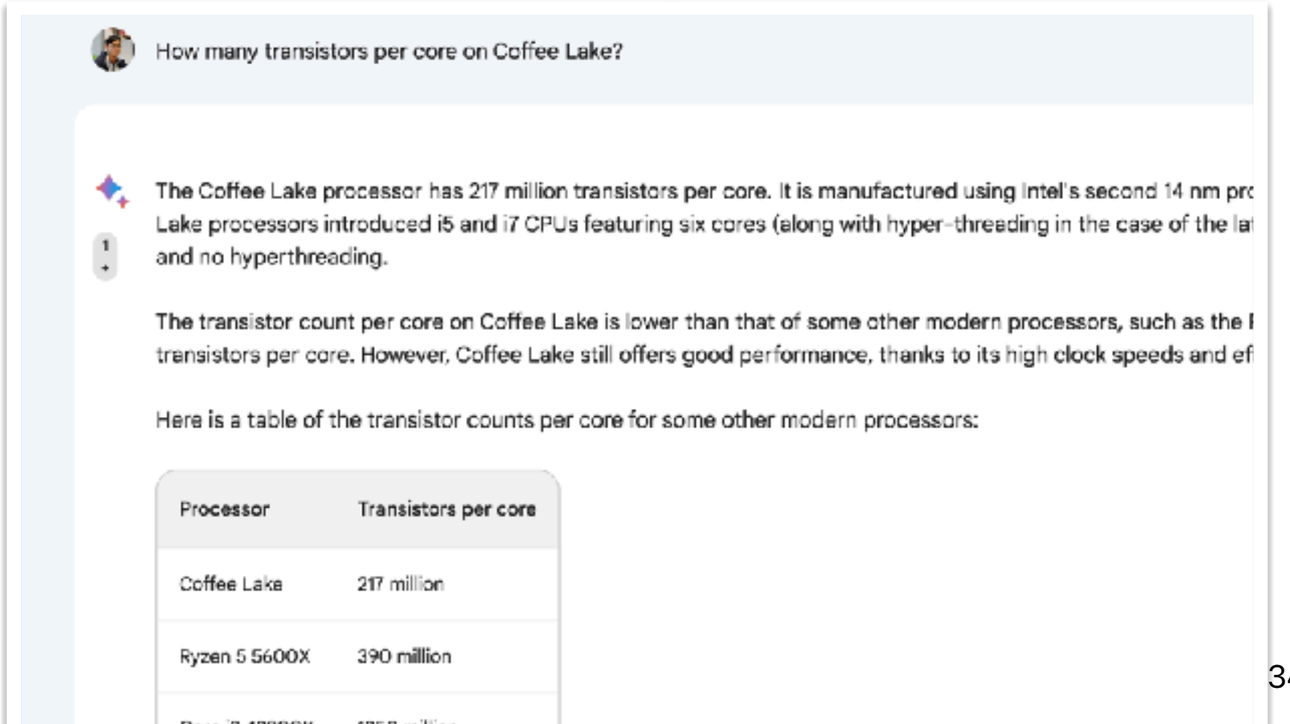
Based on https://en.wikipedia.org/wiki/Transistor_count

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



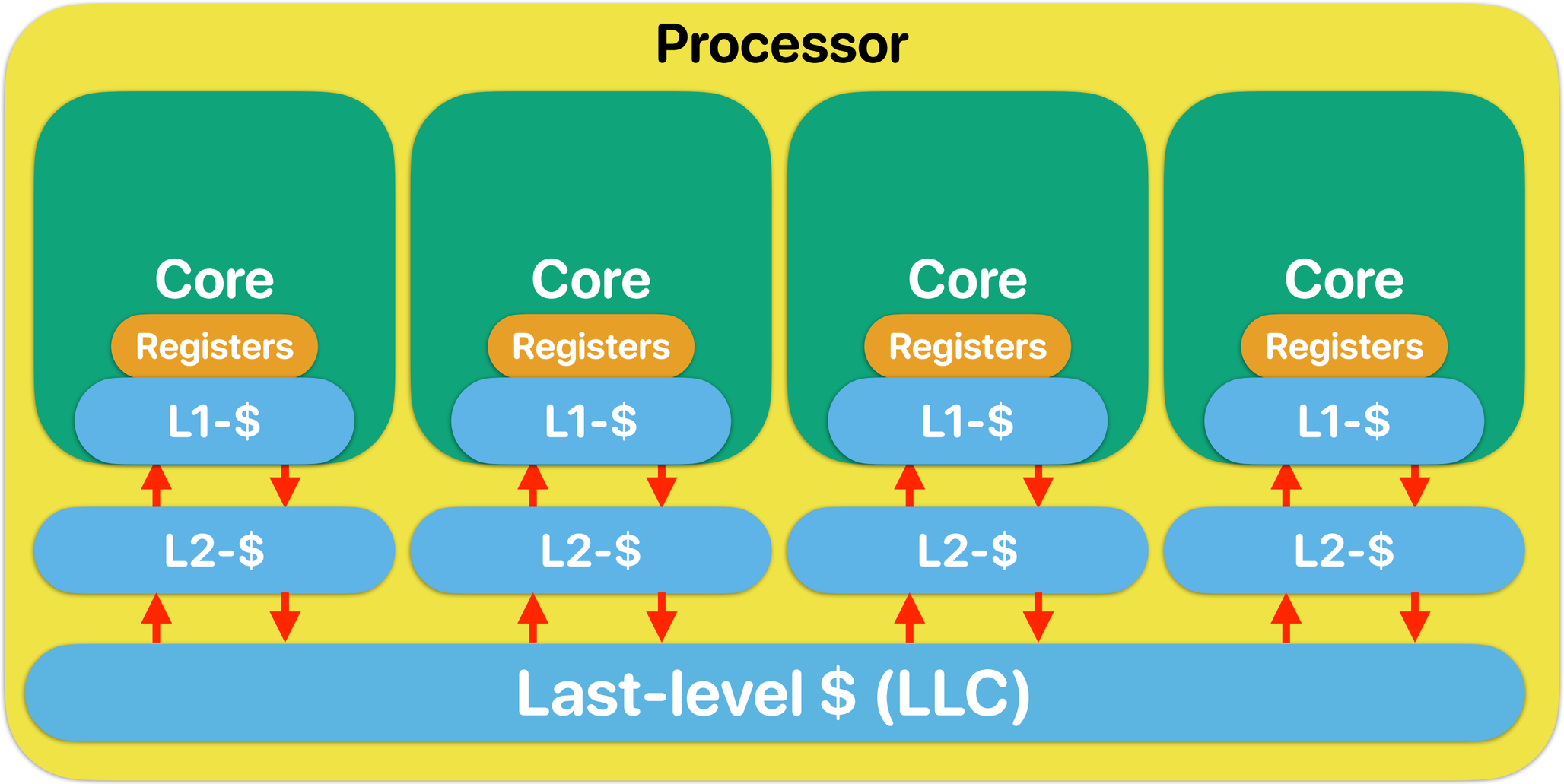
2x 3-issue ALUs Nehalem

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

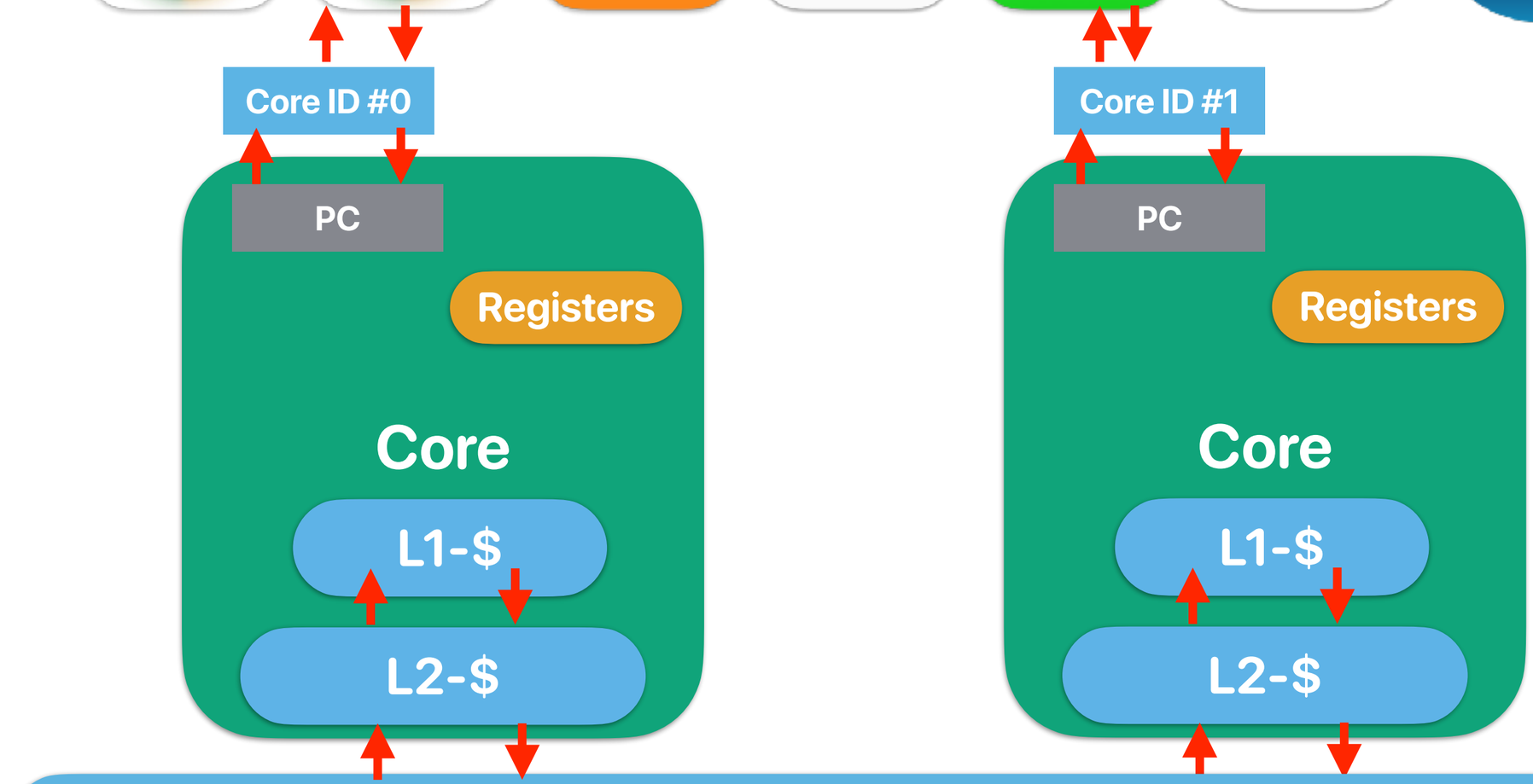
1x 5-issue ALUs Alder Lake

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

Concept of CMP



CMP from the user/OS' perspective



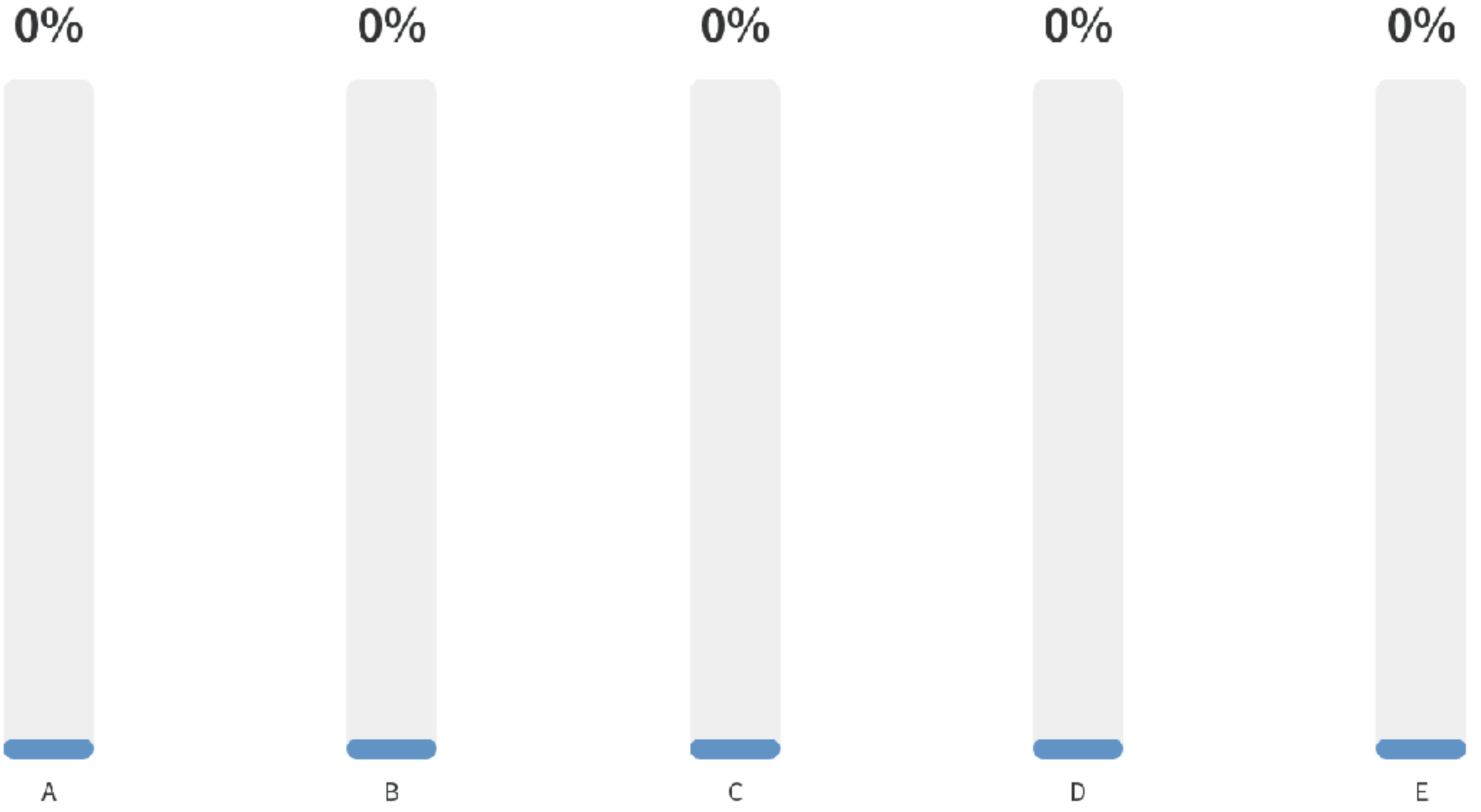
https://www.pollev.com/hungweitseng close in 1:30

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

0



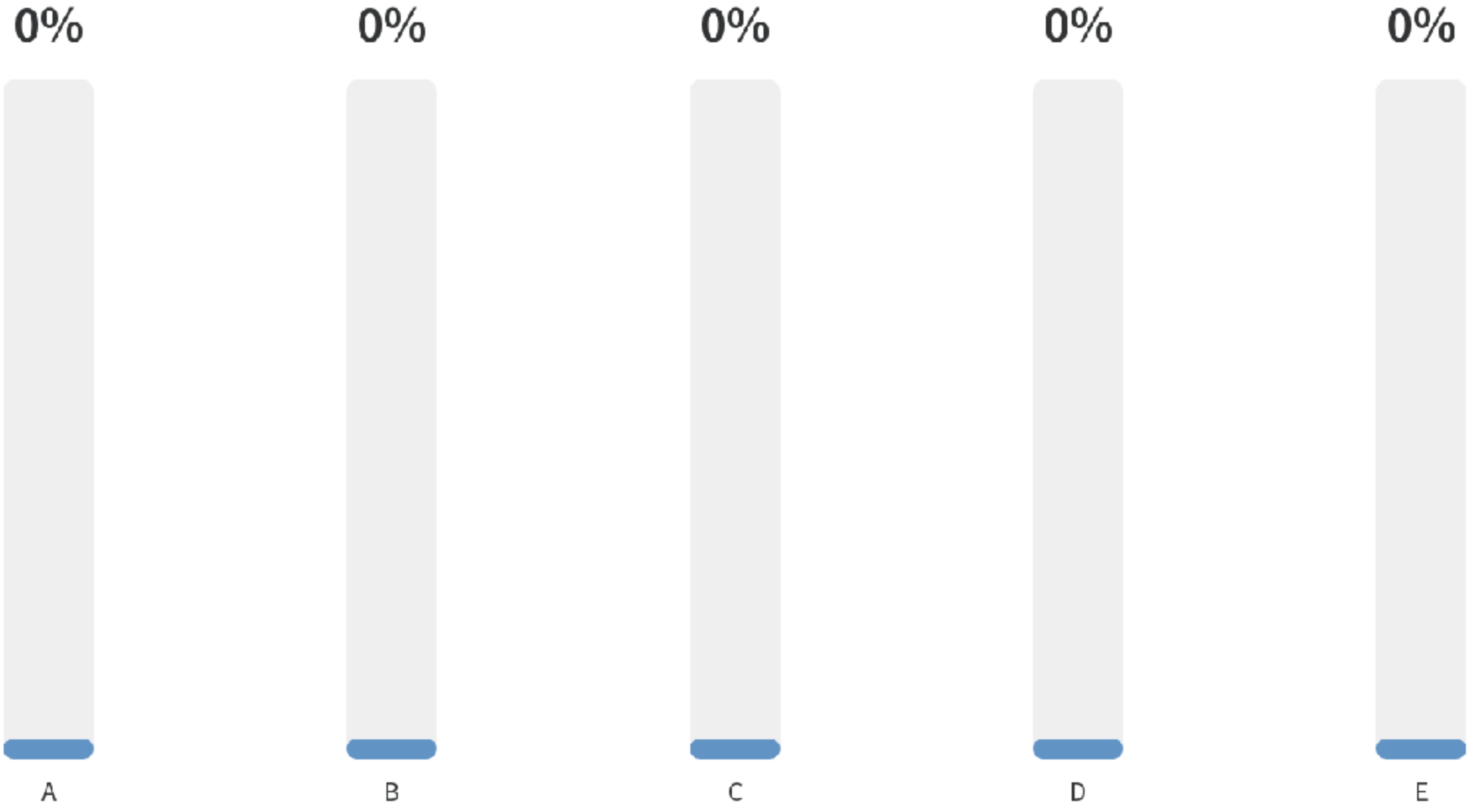
https://www.pollev.com/hungweitseng close in 1:30

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

0



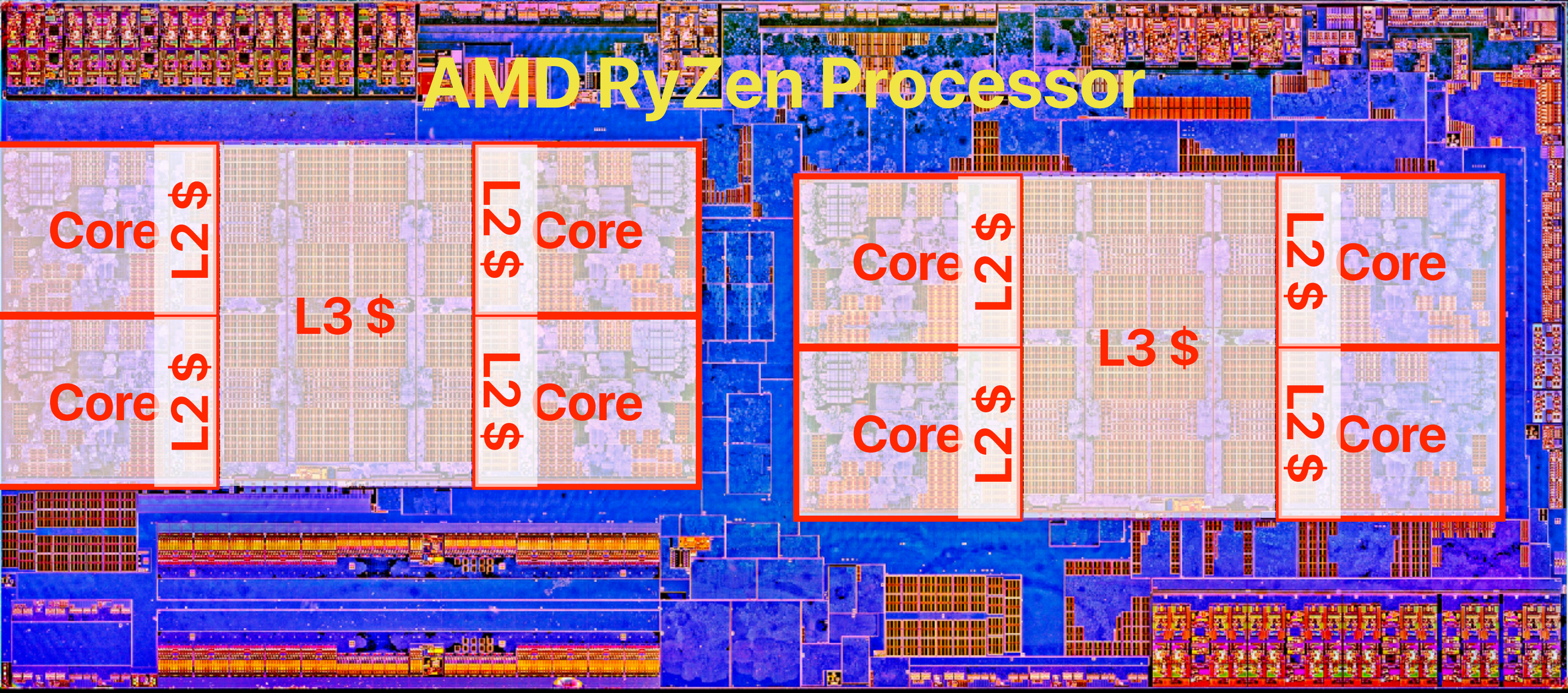
SMT v.s. CMP

- 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
  - 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! — 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!

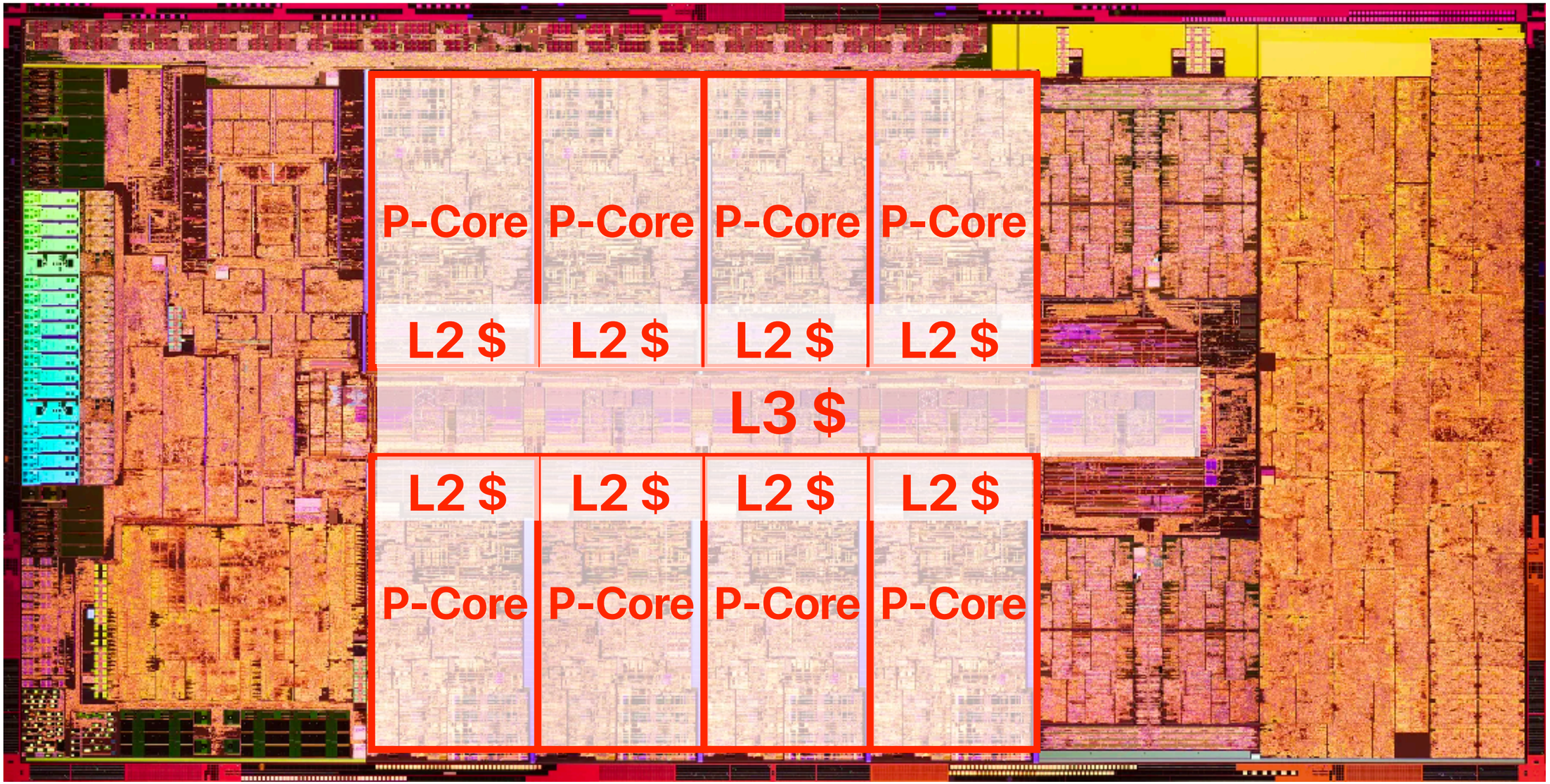
  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

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

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?

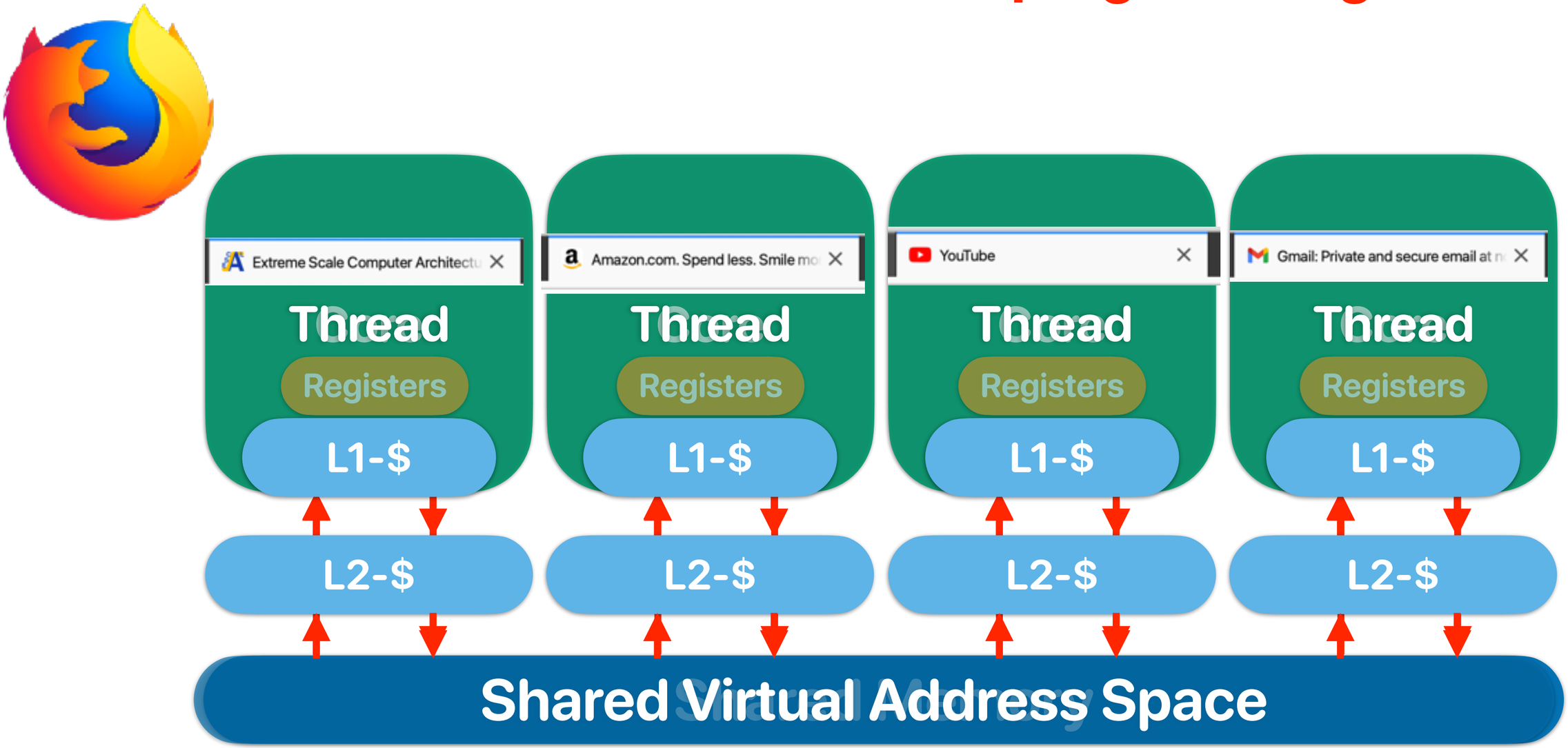
The only thing we know for sure — if we don't parallel the program, it won't get any faster on CMP

Parallel Programming & Architectural Supports for Parallel Programming

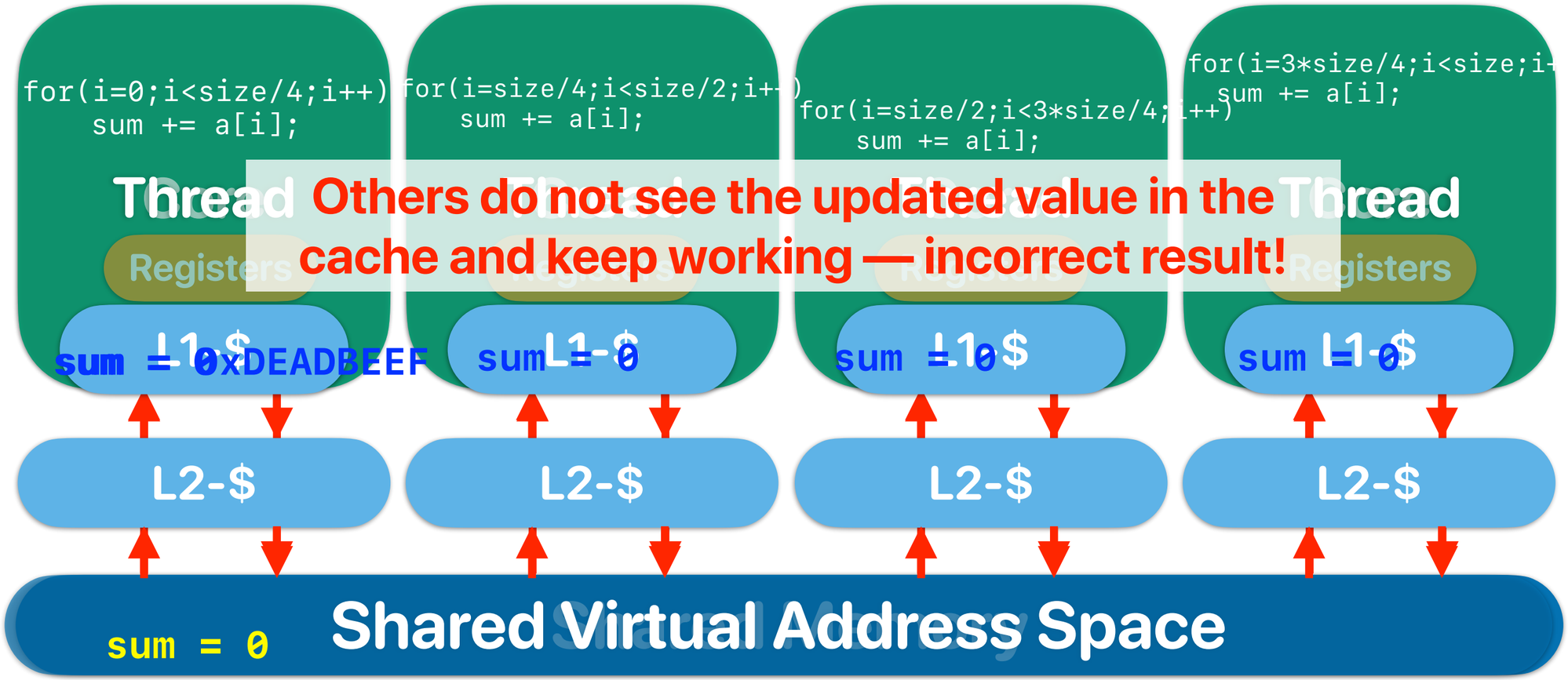
Parallel programming

- 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
- 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!

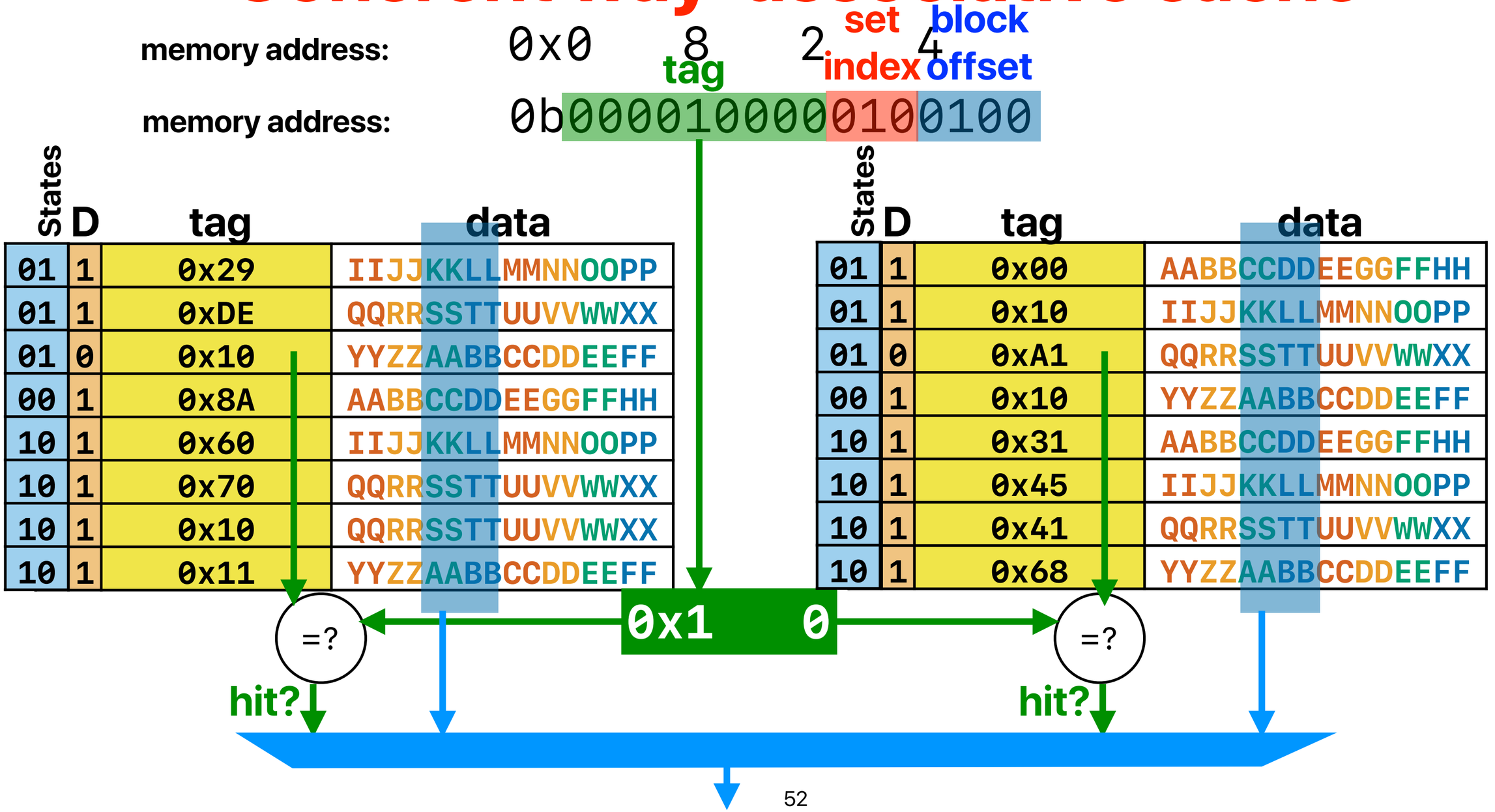
Coherency & Consistency

- 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

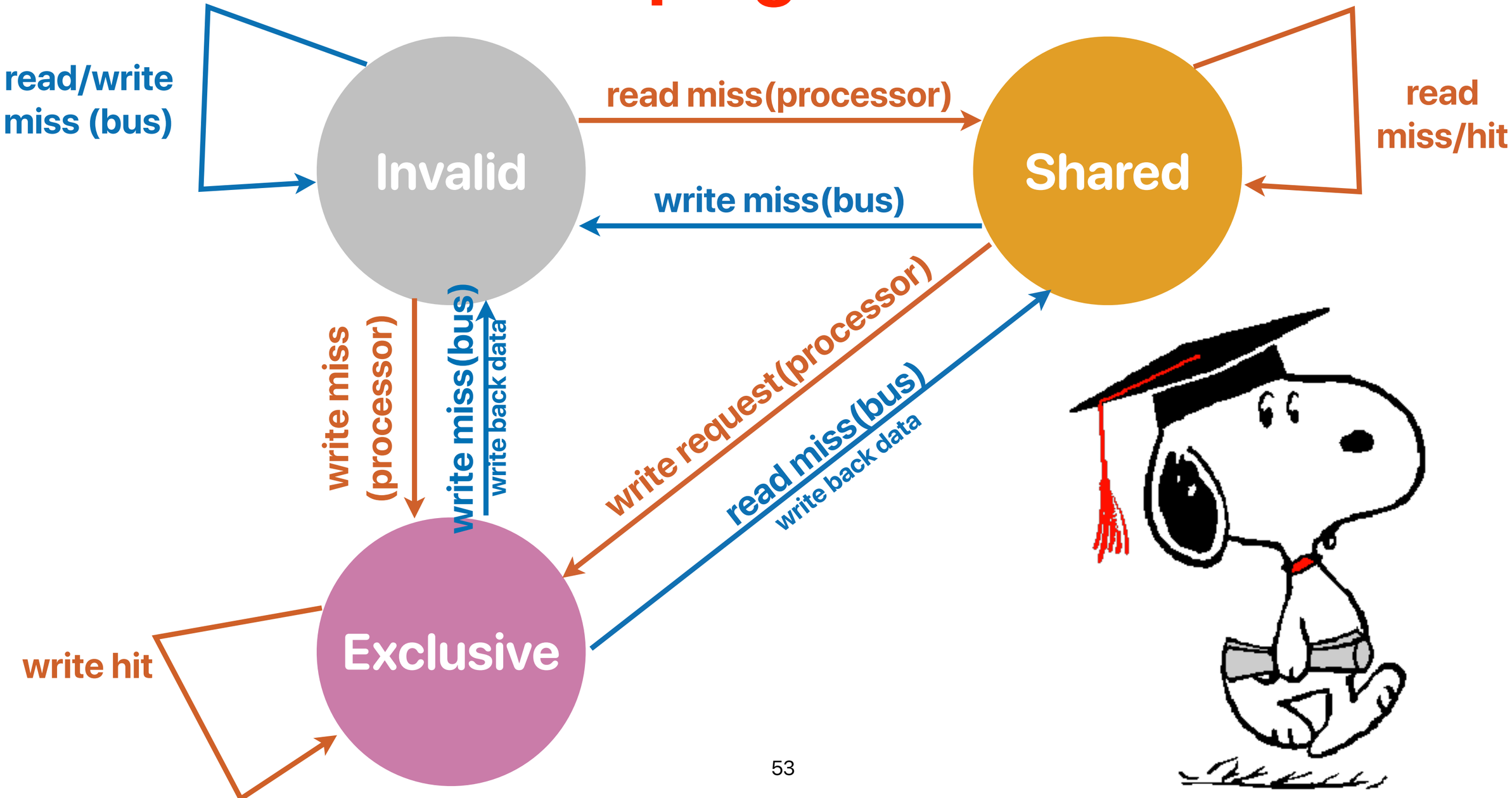
Simple cache coherency protocol

- 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

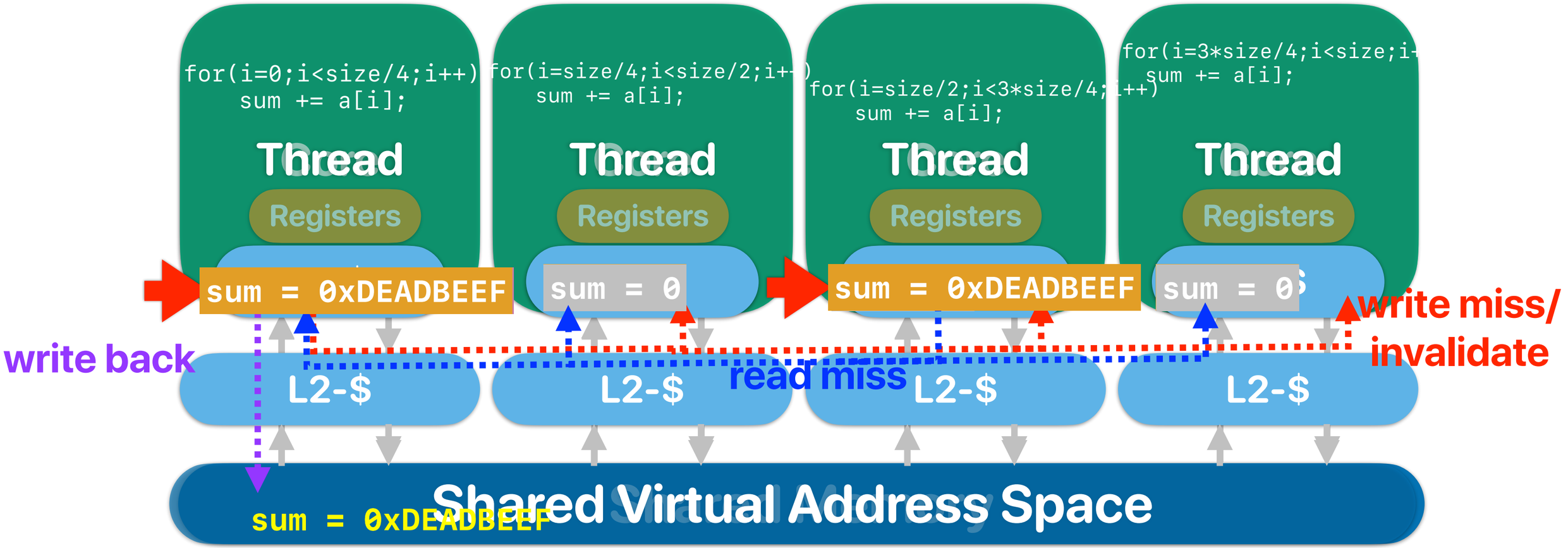
Coherent way-associative cache



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",
```

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;
}
```

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

Cache coherency

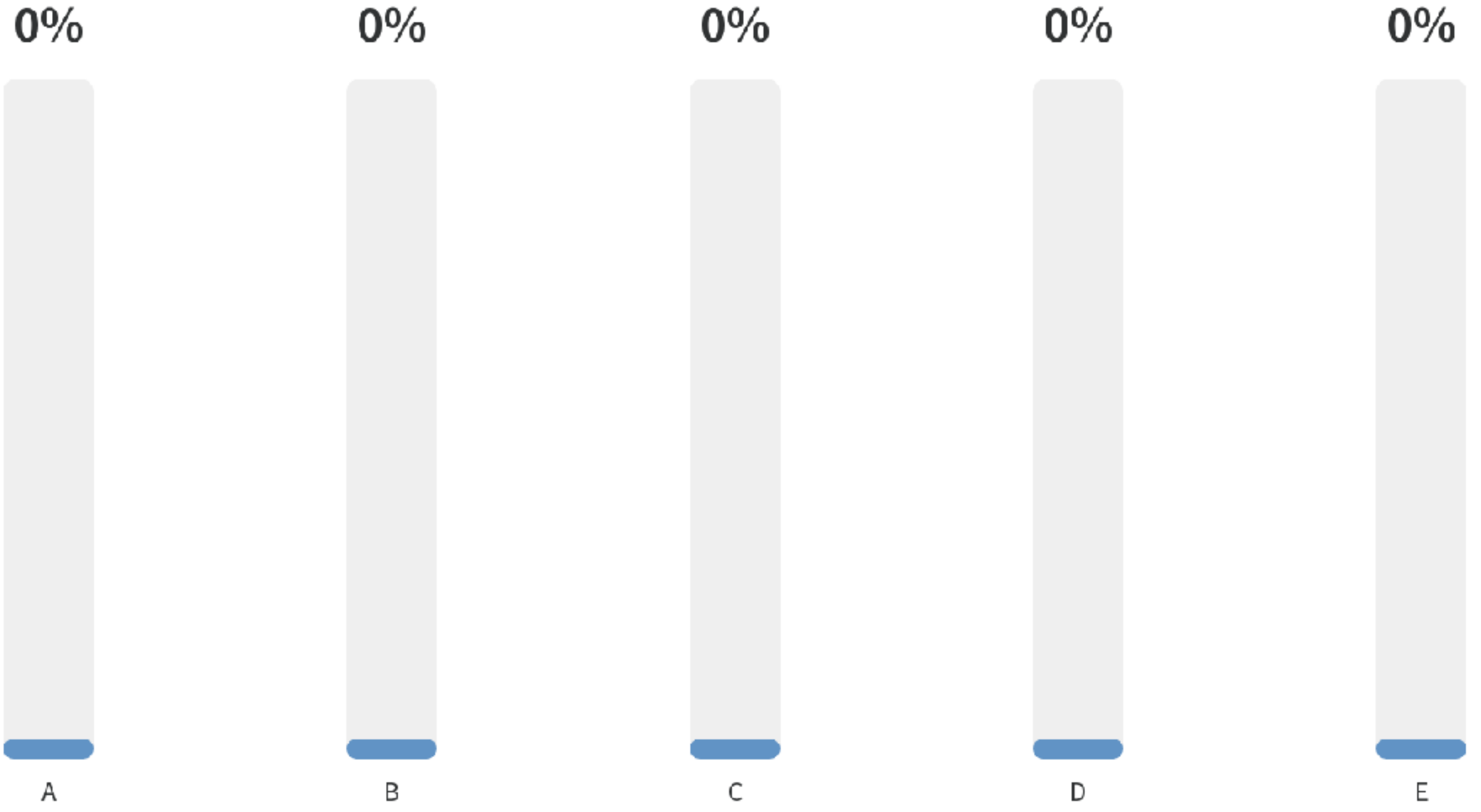
- Assuming that we are running the following code on a CMP with a cache coherency protocol, how many of the following outputs are possible? (a is initialized to 0 as assume we will output more than 10 numbers)

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

0



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

Cache coherency

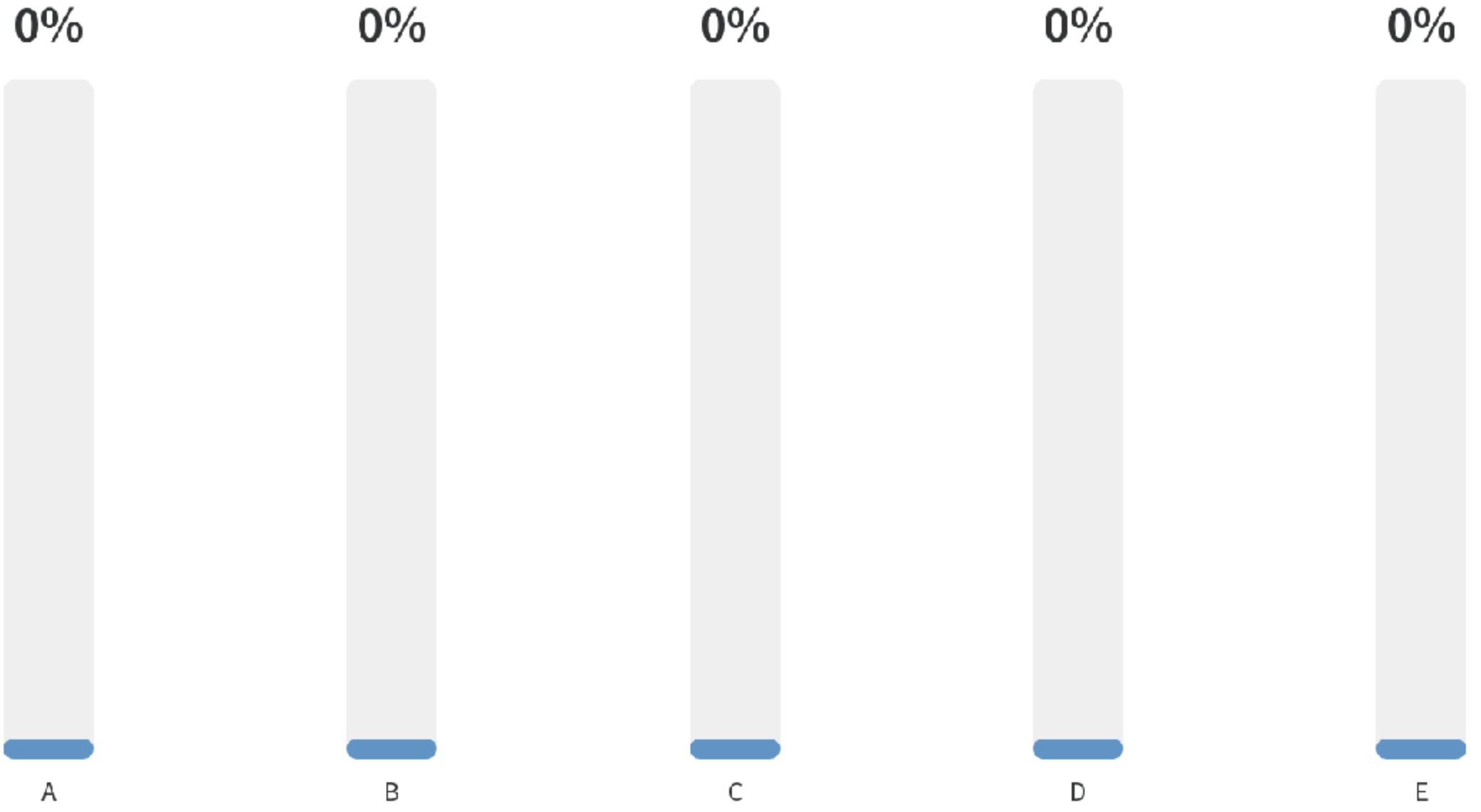
- Assuming that we are running the following code on a CMP with a cache coherency protocol, how many of the following outputs are possible? (a is initialized to 0 as assume we will output more than 10 numbers)

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

0



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

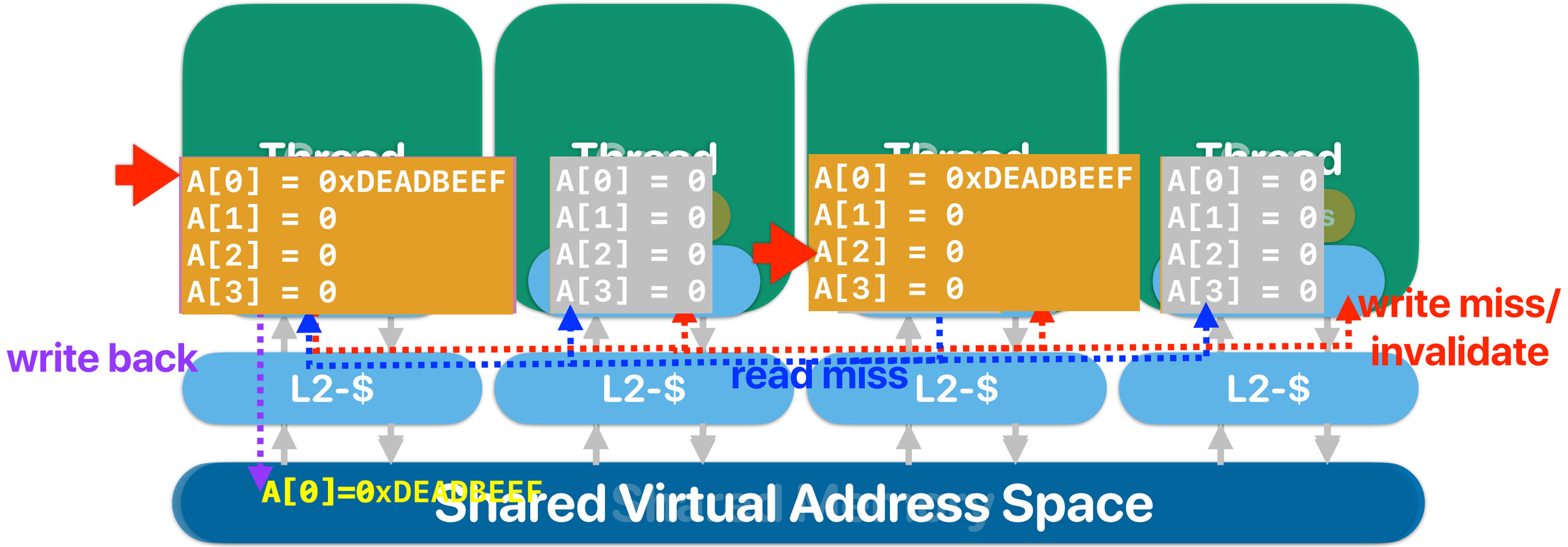
B. 1

C. 2

D. 3

E. 4

Cache coherency



Performance comparison

- Comparing implementations of thread_vadd — L and R, please identify which one will be performing better and why

Version L

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid;i<ARRAY_SIZE;i+=NUM_OF_THREADS)
  {
        c[i] = a[i] + b[i];
  }
  return NULL;
}
```

Version R

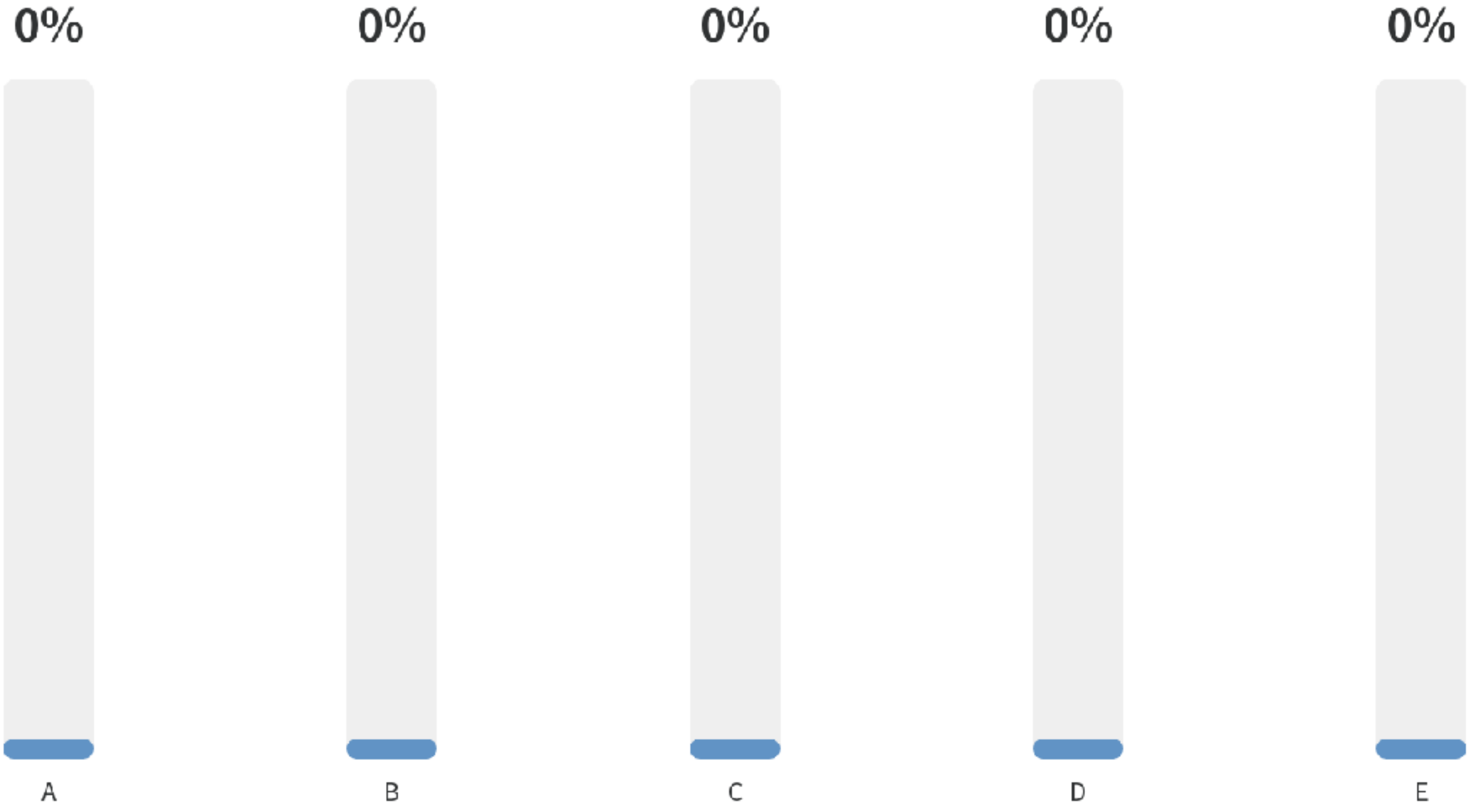
```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid*(ARRAY_SIZE/NUM_OF_THREADS);i<(tid+1)*(ARRAY_SIZE/NUM_OF_THREADS);i++)
  {
      c[i] = a[i] + b[i];
  }
  return NULL;
}
```

A. L is better, because the cache miss rate is lower
B. R is better, because the cache miss rate is lower
C. L is better, because the instruction count is lower
D. R is better, because the instruction count is lower
E. Both are about the same

Main thread

```
for(i = 0 ; i < NUM_OF_THREADS ; i++)
{
  tids[i] = i;
  pthread_create(&thread[i], NULL, threaded_vadd, &tids
}
for(i = 0 ; i < NUM_OF_THREADS ; i++)
  pthread_join(thread[i], NULL);
```

0



Performance comparison

- Comparing implementations of thread_vadd — L and R, please identify which one will be performing better and why

Version L

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid;i<ARRAY_SIZE;i+=NUM_OF_THREADS)
  {
        c[i] = a[i] + b[i];
  }
  return NULL;
}
```

Version R

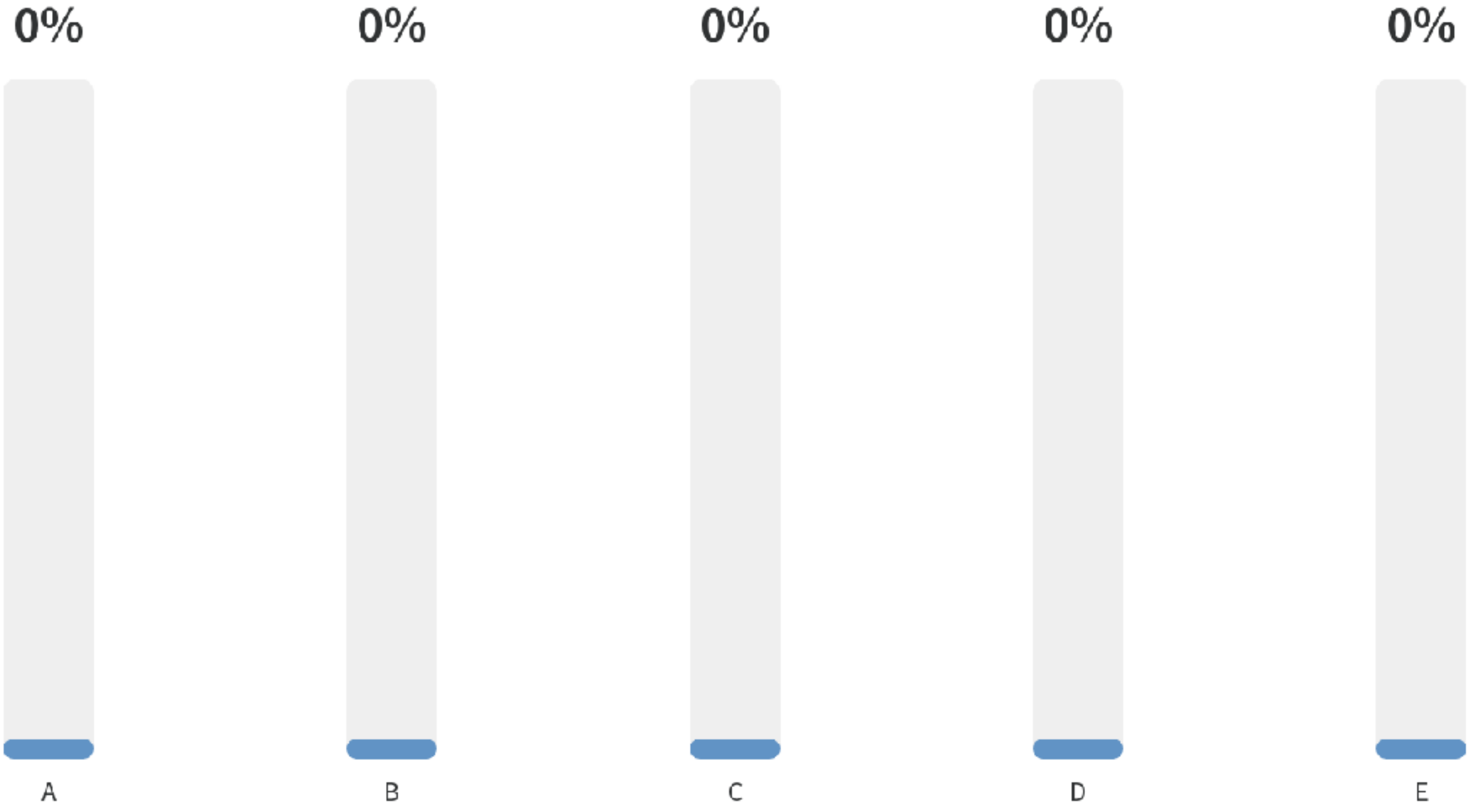
```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid*(ARRAY_SIZE/NUM_OF_THREADS);i<(tid+1)*(ARRAY_SIZE/NUM_OF_THREADS);i++)
  {
      c[i] = a[i] + b[i];
  }
  return NULL;
}
```

A. L is better, because the cache miss rate is lower
B. R is better, because the cache miss rate is lower
C. L is better, because the instruction count is lower
D. R is better, because the instruction count is lower
E. Both are about the same

Main thread

```
for(i = 0 ; i < NUM_OF_THREADS ; i++)
{
  tids[i] = i;
  pthread_create(&thread[i], NULL, threaded_vadd, &tids
}
for(i = 0 ; i < NUM_OF_THREADS ; i++)
  pthread_join(thread[i], NULL);
```

0



Performance comparison

- Comparing implementations of thread_vadd — L and R, please identify which one will be performing better and why

Version L

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid;i<ARRAY_SIZE;i+=NUM_OF_THREADS)
  {
        c[i] = a[i] + b[i];
  }
  return NULL;
}
```

Version R

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid*(ARRAY_SIZE/NUM_OF_THREADS);i<(tid+1)*(ARRAY_SIZE/NUM_OF_THREADS);i++)
  {
      c[i] = a[i] + b[i];
  }
  return NULL;
}
```

A. L is better, because the cache miss rate is lower
B. R is better, because the cache miss rate is lower
C. L is better, because the instruction count is lower
D. R is better, because the instruction count is lower
E. Both are about the same

Main thread

```
for(i = 0 ; i < NUM_OF_THREADS ; i++)
{
  tids[i] = i;
  pthread_create(&thread[i], NULL, threaded_vadd, &tids
}
for(i = 0 ; i < NUM_OF_THREADS ; i++)
  pthread_join(thread[i], NULL);
```

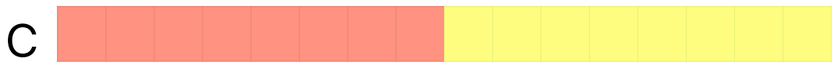
L v.s. R

Version L

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid;i<ARRAY_SIZE;i+=NUM_OF_THREADS)
  {
        c[i] = a[i] + b[i];
  }
  return NULL;
}
```

Version R

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid*(ARRAY_SIZE/NUM_OF_THREADS);i<(tid+1)*(ARRAY_SIZE/NUM_OF_THREADS);i++)
  {
      c[i] = a[i] + b[i];
  }
  return NULL;
}
```



4Cs of cache misses

- 3Cs:
  - Compulsory, Conflict, Capacity
- Coherency miss:
  - A “block” invalidated because of the sharing among processors.

False sharing

- True sharing
  - Processor A modifies X, processor B also want to access X.
- False sharing
  - Processor A modifies X, processor B also want to access Y. However, Y is invalidated because X and Y are in the same block!

Performance comparison

- Comparing implementations of thread_vadd — L and R, please identify which one will be performing better and why

Version L

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid;i<ARRAY_SIZE;i+=NUM_OF_THREADS)
  {
        c[i] = a[i] + b[i];
  }
  return NULL;
}
```

Version R

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid*(ARRAY_SIZE/NUM_OF_THREADS);i<(tid+1)*(ARRAY_SIZE/NUM_OF_THREADS);i++)
  {
      c[i] = a[i] + b[i];
  }
  return NULL;
}
```

A. L is better, because the cache miss rate is lower
B. R is better, because the cache miss rate is lower
C. L is better, because the instruction count is lower
D. R is better, because the instruction count is lower
E. Both are about the same

Main thread

```
for(i = 0 ; i < NUM_OF_THREADS ; i++)
{
  tids[i] = i;
  pthread_create(&thread[i], NULL, threaded_vadd, &tids
}
for(i = 0 ; i < NUM_OF_THREADS ; i++)
  pthread_join(thread[i], NULL);
```

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

Again — how many values are possible?

- Consider the given program. You can safely assume the caches are coherent. How many of the following outputs will you see?

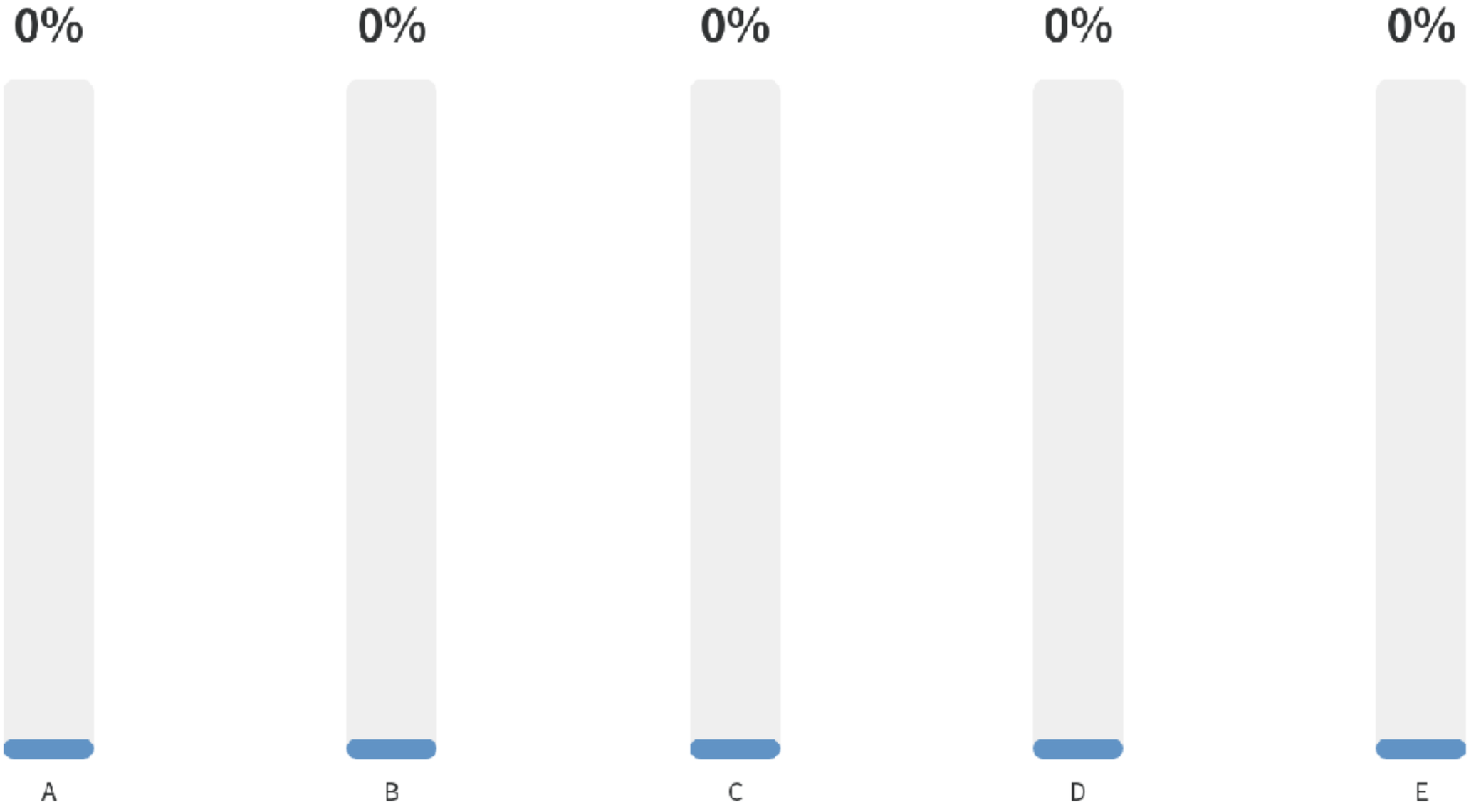
① (0, 0)
② (0, 1)
③ (1, 0)
④ (1, 1)

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

volatile int a,b;
volatile int x,y;
volatile int f;
void* modifya(void *z) {
  a=1;
  x=b;
  return NULL;
}
void* modifyb(void *z) {
  b=1;
  y=a;
  return NULL;
}
```

```c
int main() {
  int i;
  pthread_t thread[2];
  pthread_create(&thread[0], NULL, modifya, NULL);
  pthread_create(&thread[1], NULL, modifyb, NULL);
  pthread_join(thread[0], NULL);
  pthread_join(thread[1], NULL);
  fprintf(stderr,"(%d, %d)\n",x,y);
  return 0;
}
```

0



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

Again — how many values are possible?

- Consider the given program. You can safely assume the caches are coherent. How many of the following outputs will you see?

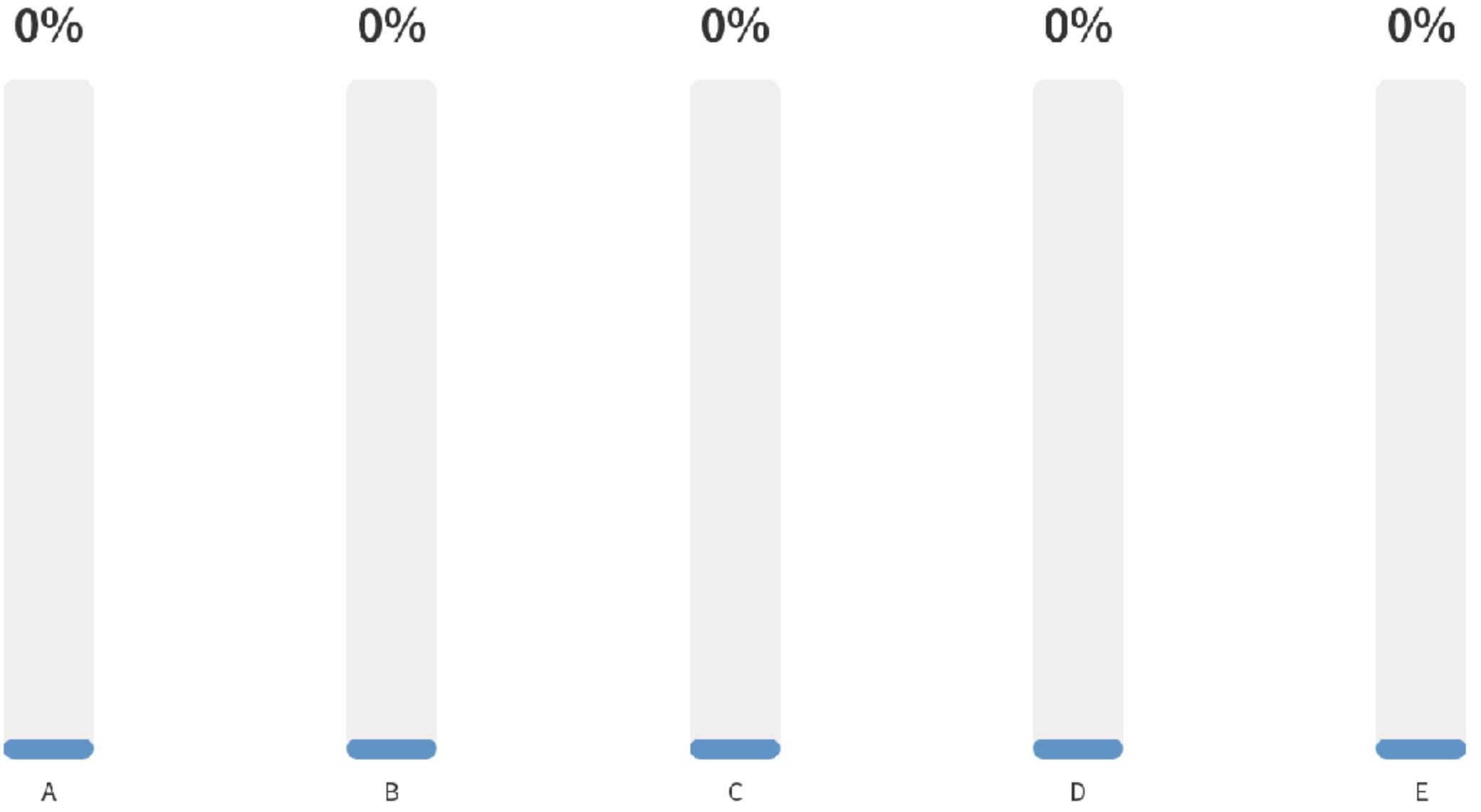
① (0, 0)
② (0, 1)
③ (1, 0)
④ (1, 1)

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

volatile int a,b;
volatile int x,y;
volatile int f;
void* modifya(void *z) {
  a=1;
  x=b;
  return NULL;
}
void* modifyb(void *z) {
  b=1;
  y=a;
  return NULL;
}
```

```c
int main() {
  int i;
  pthread_t thread[2];
  pthread_create(&thread[0], NULL, modifya, NULL);
  pthread_create(&thread[1], NULL, modifyb, NULL);
  pthread_join(thread[0], NULL);
  pthread_join(thread[1], NULL);
  fprintf(stderr,"(%d, %d)\n",x,y);
  return 0;
}
```

0



Again — how many values are possible?

- Consider the given program. You can safely assume the caches are coherent. How many of the following outputs will you see?

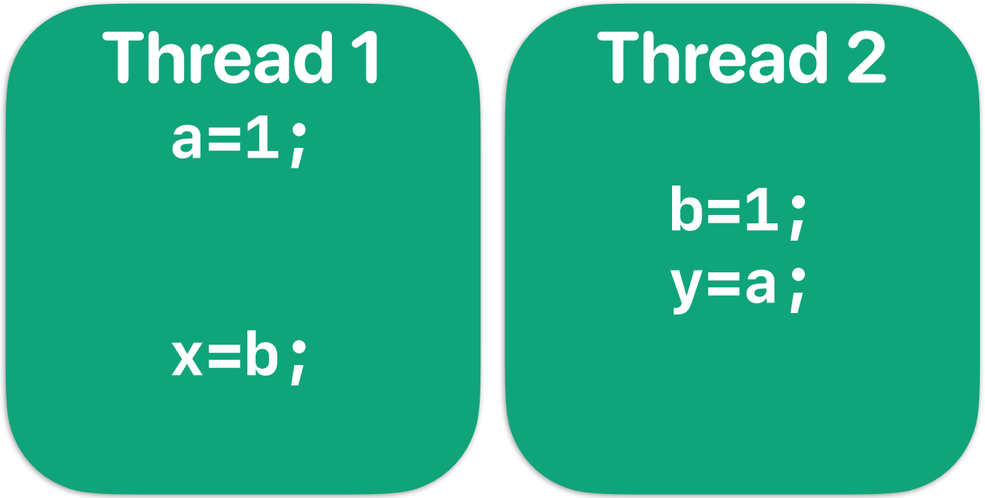
① (0, 0)
② (0, 1)
③ (1, 0)
④ (1, 1)

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

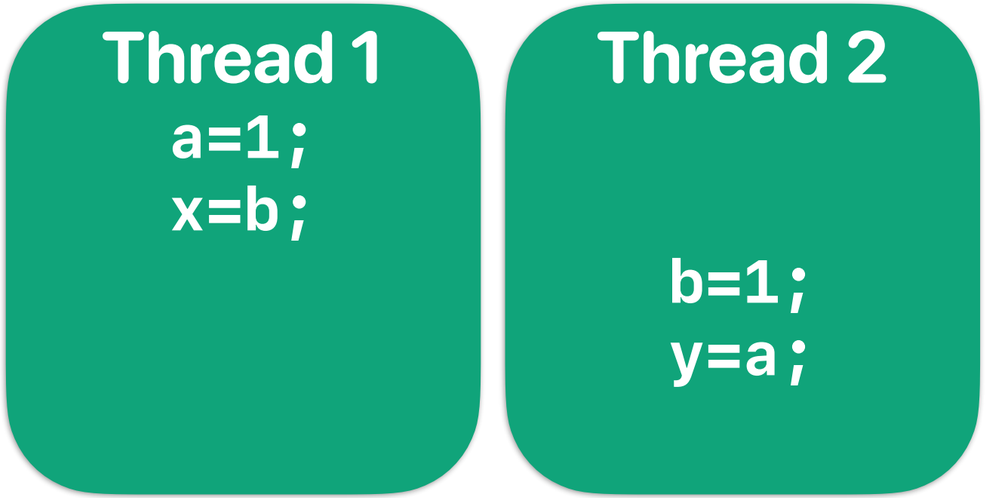
```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

volatile int a,b;
volatile int x,y;
volatile int f;
void* modifya(void *z) {
  a=1;
  x=b;
  return NULL;
}
void* modifyb(void *z) {
  b=1;
  y=a;
  return NULL;
}
```

```c
int main() {
  int i;
  pthread_t thread[2];
  pthread_create(&thread[0], NULL, modifya, NULL);
  pthread_create(&thread[1], NULL, modifyb, NULL);
  pthread_join(thread[0], NULL);
  pthread_join(thread[1], NULL);
  fprintf(stderr,"(%d, %d)\n",x,y);
  return 0;
}
```

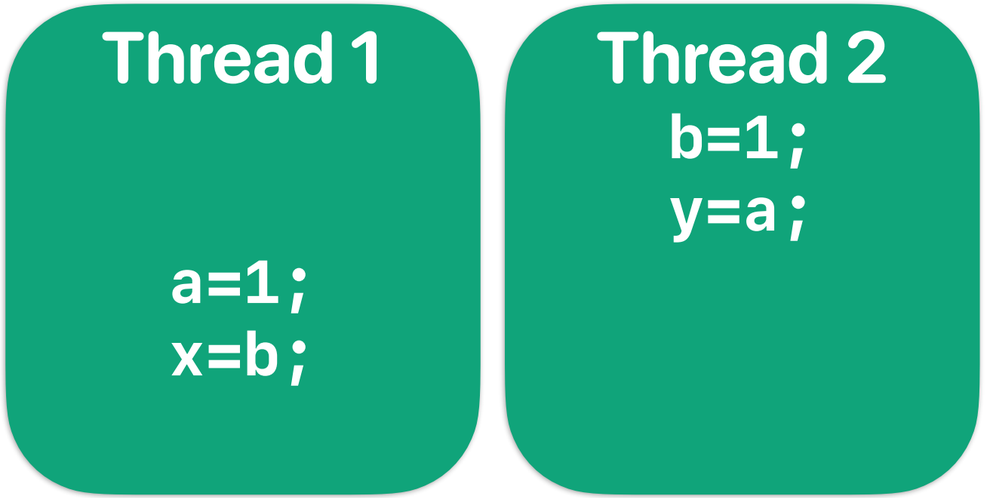
Possible scenarios



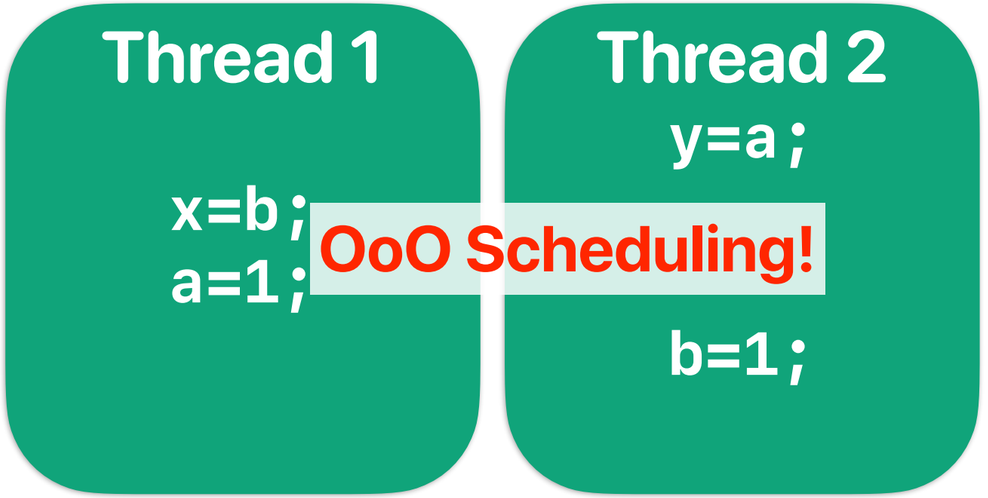
(1,1)



(0,1)



(1,0)



(0,0)

Why (0,0)?

- Processor/compiler may reorder your memory operations/ instructions
  - Coherence protocol can only guarantee the update of the same memory address
  - Processor can serve memory requests without cache miss first
  - Compiler may store values in registers and perform memory operations later
- Each processor core may not run at the same speed (cache misses, branch mis-prediction, I/O, voltage scaling and etc..)
- Threads may not be executed/scheduled right after it's spawned

Again — how many values are possible?

- Consider the given program. You can safely assume the caches are coherent. How many of the following outputs will you see?

① (0, 0)
② (0, 1)
③ (1, 0)
④ (1, 1)

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

volatile int a,b;
volatile int x,y;
volatile int f;
void* modifya(void *z) {
  a=1;
  x=b;
  return NULL;
}
void* modifyb(void *z) {
  b=1;
  y=a;
  return NULL;
}
```

```c
int main() {
  int i;
  pthread_t thread[2];
  pthread_create(&thread[0], NULL, modifya, NULL);
  pthread_create(&thread[1], NULL, modifyb, NULL);
  pthread_join(thread[0], NULL);
  pthread_join(thread[1], NULL);
  fprintf(stderr,"(%d, %d)\n",x,y);
  return 0;
}
```

fence instructions

- x86 provides an “mfence” instruction to prevent reordering across the fence instruction
  - All updates prior to mfence must finish before the instruction can proceed
- x86 only supports this kind of “relaxed consistency” model. You still have to be careful enough to make sure that your code behaves as you expected

thread 1	thread 2

`a=1;`
`mfence` **a=1 must occur/update before mfence**
`x=b;`
	`b=1;`
	`mfence` **b=1 must occur/update before mfence**
	`y=a;`

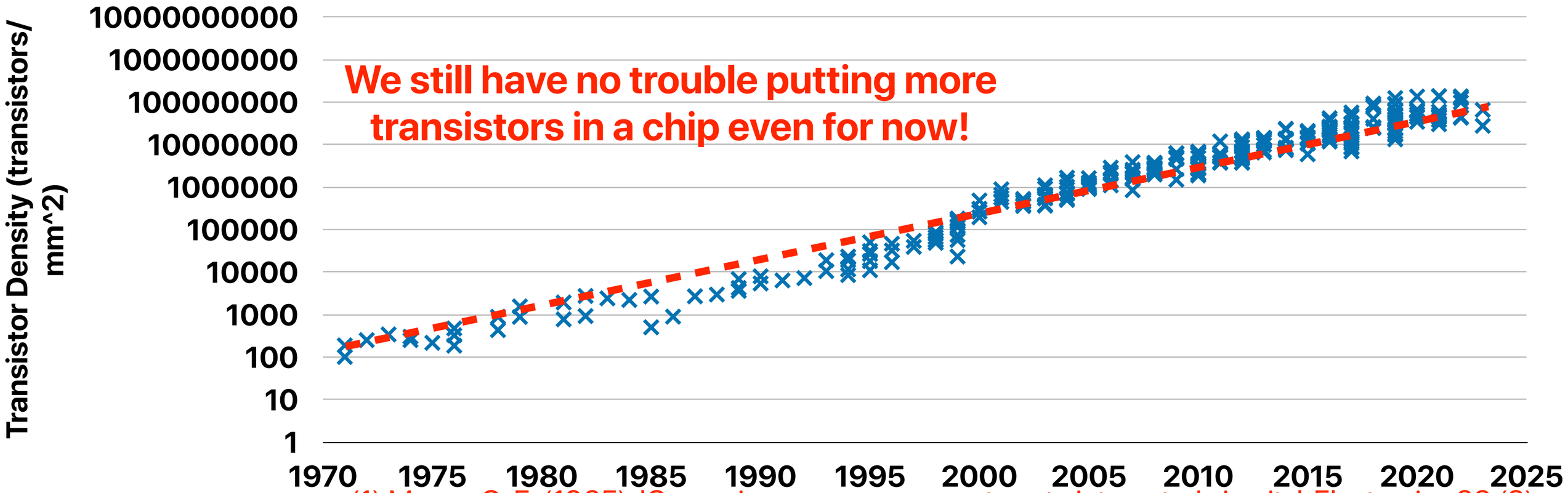
Take-aways of parallel programming

- Processor behaviors are non-deterministic
  - You cannot predict which processor is going faster
  - You cannot predict when OS is going to schedule your thread
- Cache coherency only guarantees that everyone would eventually have a coherent view of data, but not when
- Cache consistency is hard to support

The limiting factor of modern processor performance

Moore's Law[1]

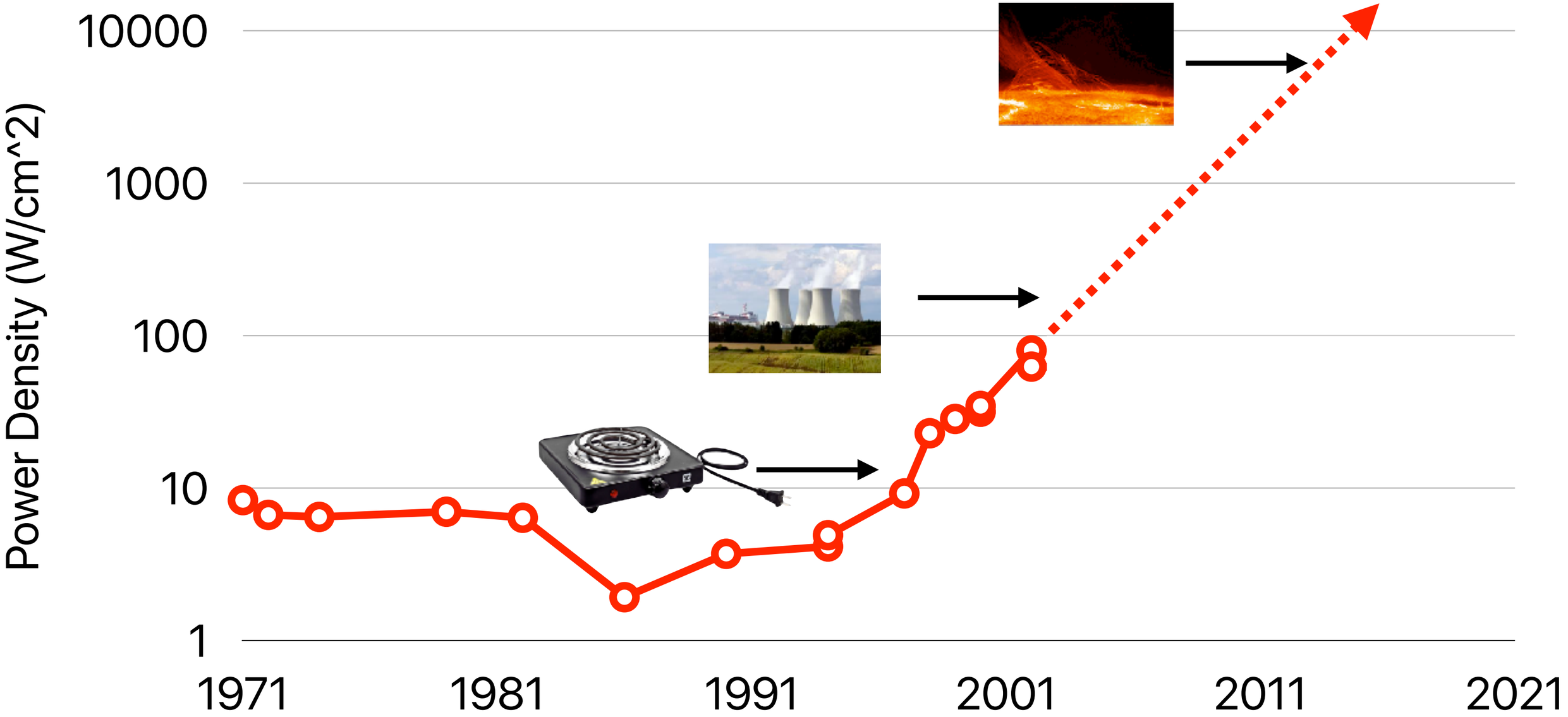
- The number of transistors we can build in a fixed area of silicon doubles every 12 ~ 24 months.
- Moore's Law "was" the most important driver for historic CPU performance gains



(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

Power Density of Processors

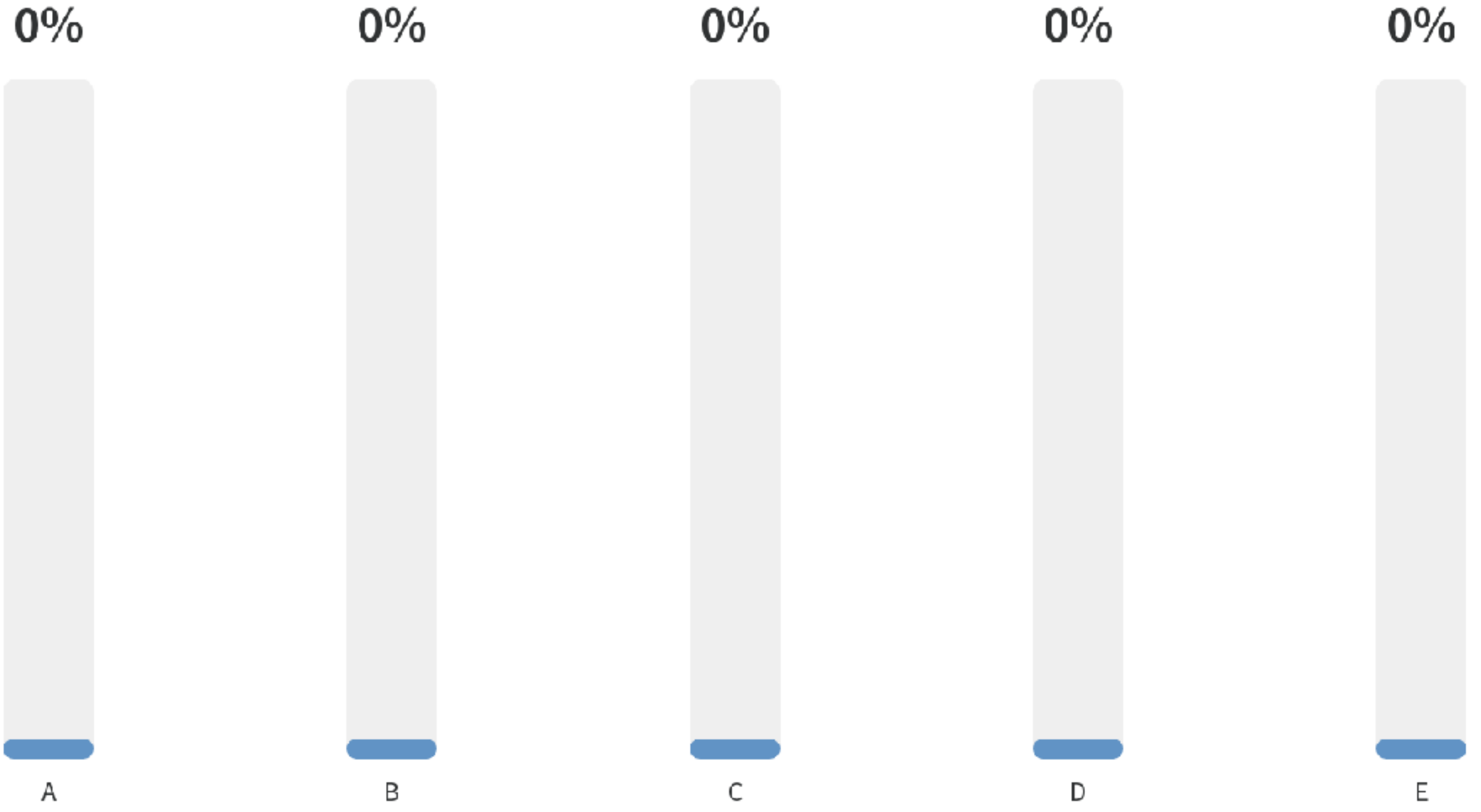


https://www.pollev.com/hungweitseng close in 1:30

Power & Energy

- Regarding power and energy, how many of the following statements are correct?
  - ① Lowering the power consumption helps extending the battery life
  - ② Lowering the power consumption helps reducing the heat generation
  - ③ Lowering the energy consumption helps reducing the electricity bill
  - ④ A CPU with 10% utilization can still consume 33% of the peak power
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

0

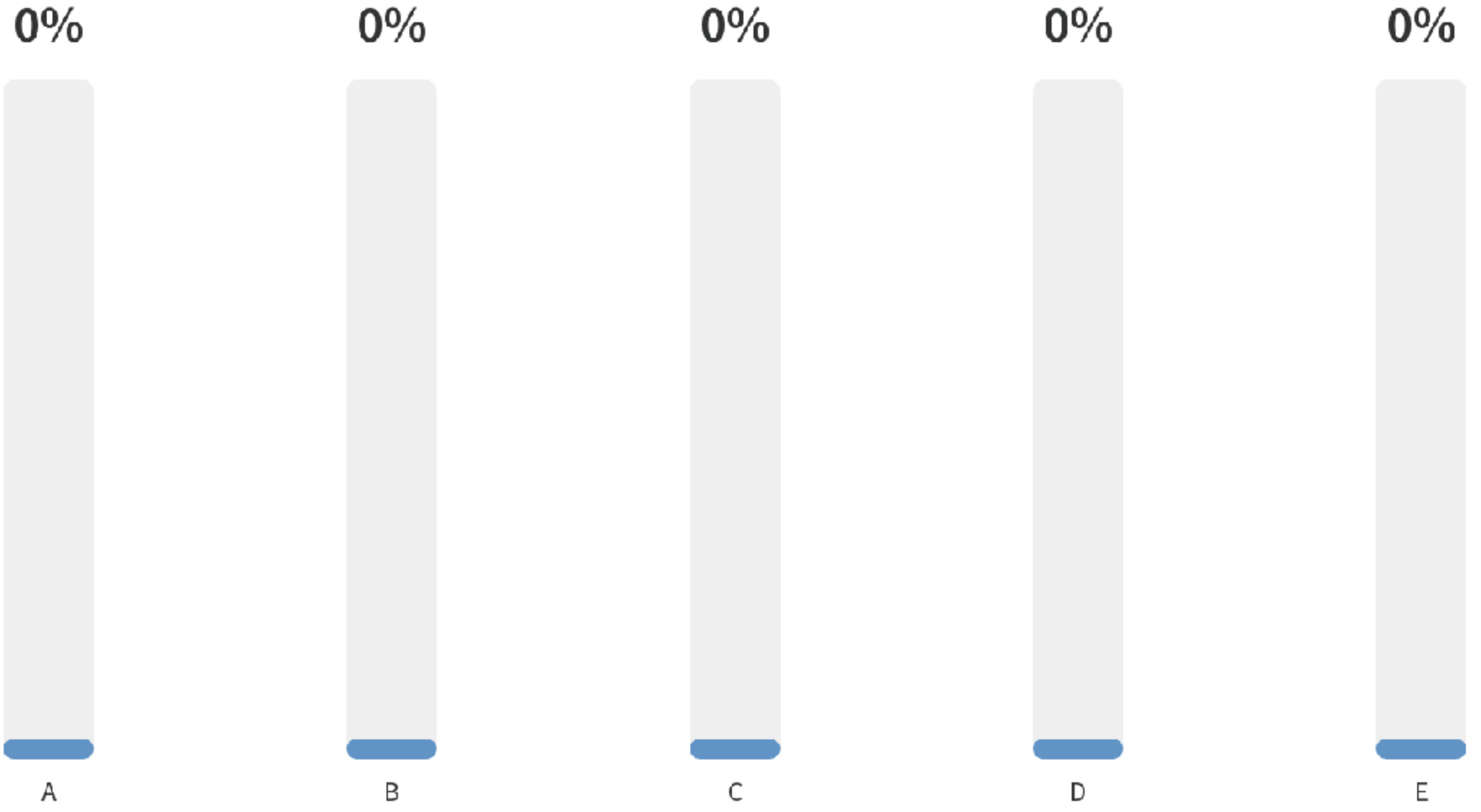


https://www.pollev.com/hungweitseng close in 1:30

Power & Energy

- Regarding power and energy, how many of the following statements are correct?
  - ① Lowering the power consumption helps extending the battery life
  - ② Lowering the power consumption helps reducing the heat generation
  - ③ Lowering the energy consumption helps reducing the electricity bill
  - ④ A CPU with 10% utilization can still consume 33% of the peak power
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

0



Power & Energy

- Regarding power and energy, how many of the following statements are correct?
  - ① Lowering the power consumption helps extending the battery life
  - ② Lowering the power consumption helps reducing the heat generation
  - ③ Lowering the energy consumption helps reducing the electricity bill
  - ④ A CPU with 10% utilization can still consume 33% of the peak power
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Power consumption

Power v.s. Energy

- Power is the direct contributor of “heat”
  - Packaging of the chip
  - Heat dissipation cost
- Energy = P * ET
  - The electricity bill and battery life is related to energy!
  - Lower power does not necessary means better battery life if the processor slow down the application too much

Dynamic/Active Power

- 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

Static/Leakage Power

- The power consumption due to leakage — transistors do not turn all the way off during no operation
- Becomes the **dominant** factor in the most advanced process technologies.

$$P_{leakage} \sim N \times V \times e^{-V_t}$$

  - N: number of transistors
  - V: voltage
  - V_t: threshold voltage where transistor conducts (begins to switch)

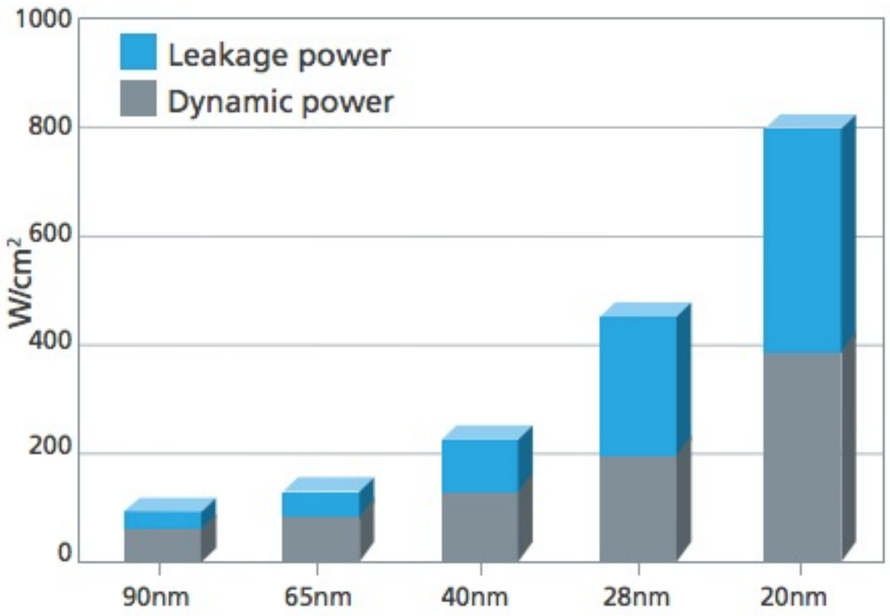


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

Dynamic/Active Power

- 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

Demo — changing the max frequency and performance

- Change the maximum frequency of the intel processor — you learned how to do this when we discuss programmer's impact on performance
- LIKWID a profiling tool providing power/energy information
  - likwid-perfctr -g ENERGY [command_line]
  - Let's try blockmm and popcount and see what's happening!

matmul

```
CPU name:       Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz
CPU type:       Intel Coffeelake processor
CPU clock:      3.70 GHz
```

Metric	Sum	Min	Max	Avg
Runtime (RDTSC) [s] STAT	56.7402	9.4567	9.4567	9.4567
Runtime unhalted [s] STAT	38089.9565	2.764721e-05	38089.9547	6348.3261
Clock [MHz] STAT	1.504735e+07	2530.3125	15027380	2.507892e+06
CPI STAT	29.6923	0.7684	11.1499	4.9487
Temperature [C] STAT	275	42	57	45.8333
Energy [J] STAT	247.1262	0	247.1262	41.1877
Power [W] STAT	26.1325	0	26.1325	4.3554
Energy PP0 [J] STAT	240.7318	0	240.7318	40.1220
Power PP0 [W] STAT	25.4563	0	25.4563	4.2427
Energy PP1 [J] STAT	0	0	0	0
Power PP1 [W] STAT	0	0	0	0
Energy DRAM [J] STAT	17.2582	0	17.2582	2.8764
Power DRAM [W] STAT	1.8250	0	1.8250	0.3042

Metric	Sum	Min	Max	Avg
Runtime (RDTSC) [s] STAT	202.3806	33.7301	33.7301	33.7301
Runtime unhalted [s] STAT	38089.3146	4.738974e-05	38089.3131	6348.2191
Clock [MHz] STAT	4.202013e+06	1135.3997	4196110	700335.5505
CPI STAT	14.9386	0.9997	6.2225	2.4898
Temperature [C] STAT	223	37	38	37.1667
Energy [J] STAT	88.4080	0	88.4080	14.7347
Power [W] STAT	2.6210	0	2.6210	0.4368
Energy PP0 [J] STAT	67.5248	0	67.5248	11.2541
Power PP0 [W] STAT	2.0019	0	2.0019	0.3337
Energy PP1 [J] STAT	0	0	0	0
Power PP1 [W] STAT	0	0	0	0
Energy DRAM [J] STAT	44.9120	0	44.9120	7.4853
Power DRAM [W] STAT	1.3315	0	1.3315	0.2219

pop count

Metric	Sum	Min	Max	Avg
Runtime (RDTSC) [s] STAT	11.8086	1.9681	1.9681	1.9681
Runtime unhalted [s] STAT	2.3622	0	2.3620	0.3937
Clock [MHz] STAT	17807.5257	2106.5416	4465.2964	2967.9209
CPI STAT	36.5243	0.3968	24.1941	6.0874
Temperature [C] STAT	249	39	49	41.5000
Energy [J] STAT	40.4858	0	40.4858	6.7476
Power [W] STAT	20.5715	0	20.5715	3.4286
Energy PP0 [J] STAT	39.1853	0	39.1853	6.5309
Power PP0 [W] STAT	19.9106	0	19.9106	3.3184
Energy PP1 [J] STAT	0	0	0	0
Power PP1 [W] STAT	0	0	0	0
Energy DRAM [J] STAT	1.9736	0	1.9736	0.3289
Power DRAM [W] STAT	1.0028	0	1.0028	0.1671

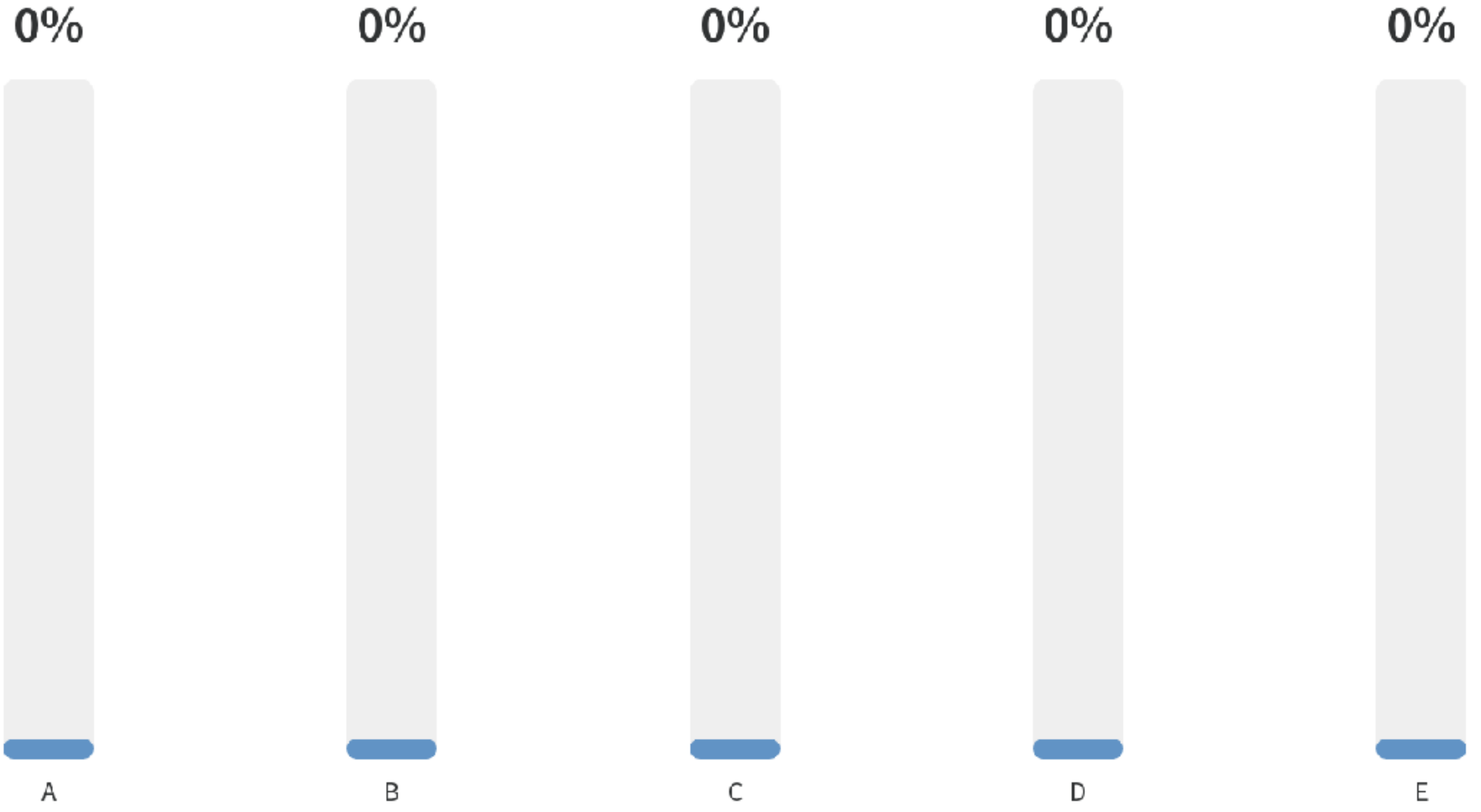
Metric	Sum	Min	Max	Avg
Runtime (RDTSC) [s] STAT	43.8276	7.3046	7.3046	7.3046
Runtime unhalted [s] STAT	2.3623	3.218655e-06	2.3620	0.3937
Clock [MHz] STAT	6822.7683	903.2673	1197.0151	1137.1281
CPI STAT	20.0735	0.3968	12.4175	3.3456
Temperature [C] STAT	218	36	37	36.3333
Energy [J] STAT	16.6005	0	16.6005	2.7668
Power [W] STAT	2.2726	0	2.2726	0.3788
Energy PP0 [J] STAT	11.7680	0	11.7680	1.9613
Power PP0 [W] STAT	1.6110	0	1.6110	0.2685
Energy PP1 [J] STAT	0	0	0	0
Power PP1 [W] STAT	0	0	0	0
Energy DRAM [J] STAT	7.3173	0	7.3173	1.2196
Power DRAM [W] STAT	1.0017	0	1.0017	0.1670

What happens if power doesn't scale with process technologies?

- If we are able to cram more transistors within the same chip area (Moore's law continues), but the power consumption per transistor remains the same. Right now, if put more transistors in the same area because the technology allows us to. How many of the following statements are true?
  - ① The power consumption per chip will increase
  - ② The power density of the chip will increase
  - ③ Given the same power budget, we may not able to power on all chip area if we maintain the same clock rate
  - ④ Given the same power budget, we may have to lower the clock rate of circuits to power on all chip area

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

0



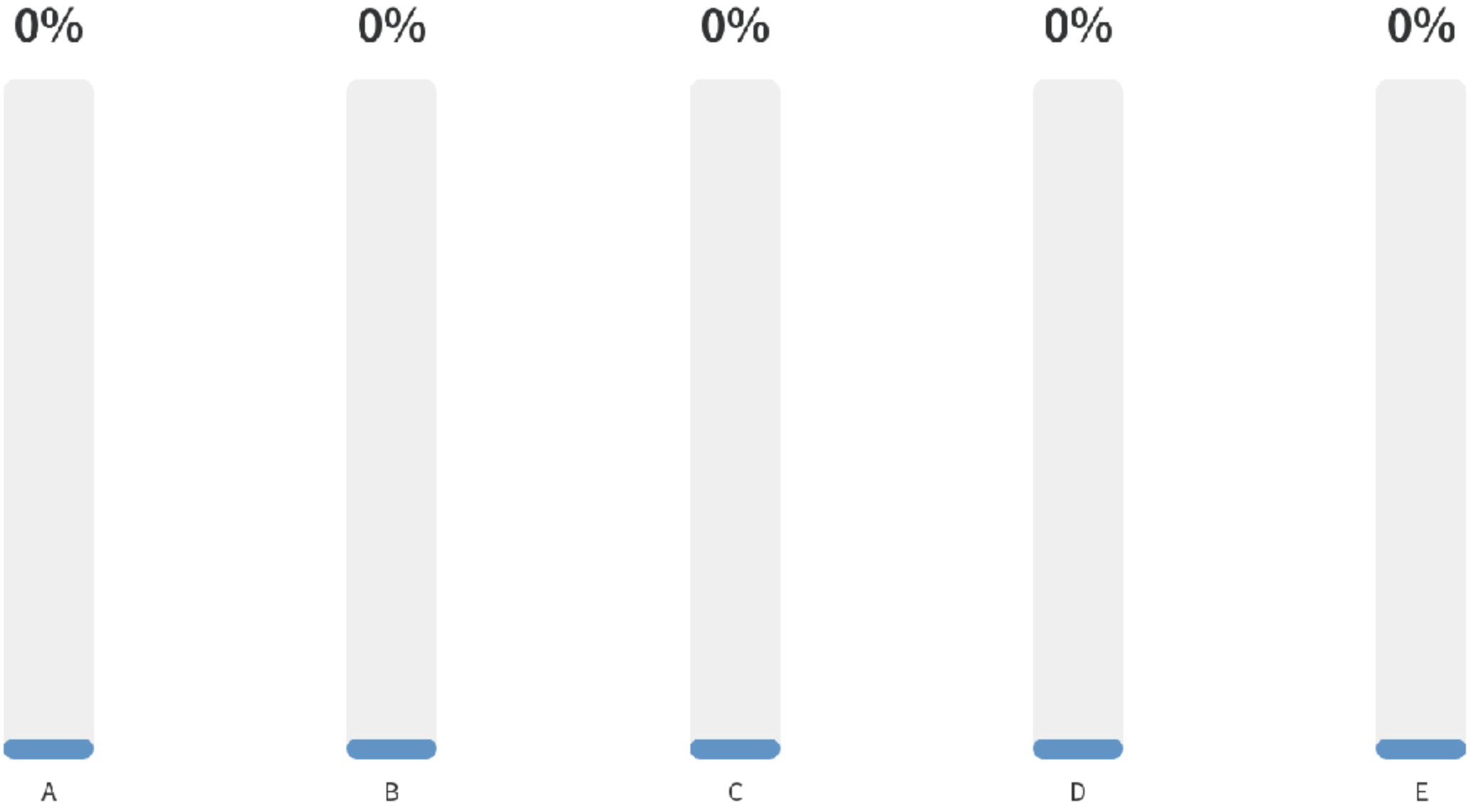
https://www.pollev.com/hungweitseng close in 1:30

What happens if power doesn't scale with process technologies?

- If we are able to cram more transistors within the same chip area (Moore's law continues), but the power consumption per transistor remains the same. Right now, if put more transistors in the same area because the technology allows us to. How many of the following statements are true?
  - ① The power consumption per chip will increase
  - ② The power density of the chip will increase
  - ③ Given the same power budget, we may not able to power on all chip area if we maintain the same clock rate
  - ④ Given the same power budget, we may have to lower the clock rate of circuits to power on all chip area

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

0



What happens if power doesn't scale with process technologies?

- If we are able to cram more transistors within the same chip area (Moore's law continues), but the power consumption per transistor remains the same. Right now, if put more transistors in the same area because the technology allows us to. How many of the following statements are true?
  - ① The power consumption per chip will increase
  - ② The power density of the chip will increase
  - ③ Given the same power budget, we may not able to power on all chip area if we maintain the same clock rate
  - ④ Given the same power budget, we may have to lower the clock rate of circuits to power on all chip area

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

Power consumption to light on all transistors

Chip

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

=49W

Dennardian Scaling

Chip

0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

=50W

Dennardian Broken

Chip

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

On ~ 50W

Off ~ 0W

Dark!

=100W!

Recap: Speedup

- Assume that we have an application composed with a total of 500000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest instructions are integer instructions with average CPI of 1 cycle when using a 2GHz processor.
  - If we double the CPU clock rate to 4GHz that helps to accelerate all instructions by 2x except that load/store instruction cannot be improved — their CPI will become 12 cycles. What's the performance improvement after this change?

A. No change

B. 1.25

C. 1.5

D. 2

E. None of the above

$$ET = IC \times CPI \times CT$$

$$ET_{baseline} = (5 \times 10^5) \times (20\% \times 6 + 80\% \times 1) \times \frac{1}{2 \times 10^{-9}} sec = 5^{-3}$$

$$ET_{enhanced} = (5 \times 10^5) \times (20\% \times 12 + 80\% \times 1) \times \frac{1}{4 \times 10^{-9}} sec = 4^{-3}$$

$$Speedup = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}}$$

$$= \frac{5}{4} = 1.25$$

The "power/energy cost" of doubling the clocking rate

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

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

$$Speedup = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}} = \frac{5}{4} = 1.25$$

$$Energy_{new} = Power_{new} \times ET_{new}$$

$$= Power_{new} \times \frac{ET_{old}}{Speedup}$$

$$= Power_{old} \times 8 \times \frac{ET_{old}}{1.25} = 6.4 \times Power_{old} \times ET_{old}$$

$$= 6.4 \times Energy_{old}$$

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

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

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:

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

Recap: Amdahl's Law on Multicore Architectures

- Symmetric multicore processor with n cores (if we assume the processor performance scales perfectly)

$$Speedup_{parallel}(f_{parallelizable}, n) = \frac{1}{(1 - f_{parallelizable}) + \frac{f\_parallelizable}{n}}$$

What if we double the number of cores?

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

Assume 40% of execution time can be parallelized

$$Speedup_{parallel}(f_{parallelizable}, n) = \frac{1}{(1 - f_{parallelizable}) + \frac{f\_parallelizable}{n}}$$

$$= \frac{1}{(1 - 40\%) + \frac{40\,\%}{2}} = 1.25$$

Same performance gain!

$$Energy_{new} = Power_{new} \times ET_{new}$$

$$= Power_{new} \times \frac{ET_{old}}{Speedup}$$

$$= Power_{old} \times 2 \times \frac{ET_{old}}{1.25} = 1.6 \times Power_{old} \times ET_{old}$$

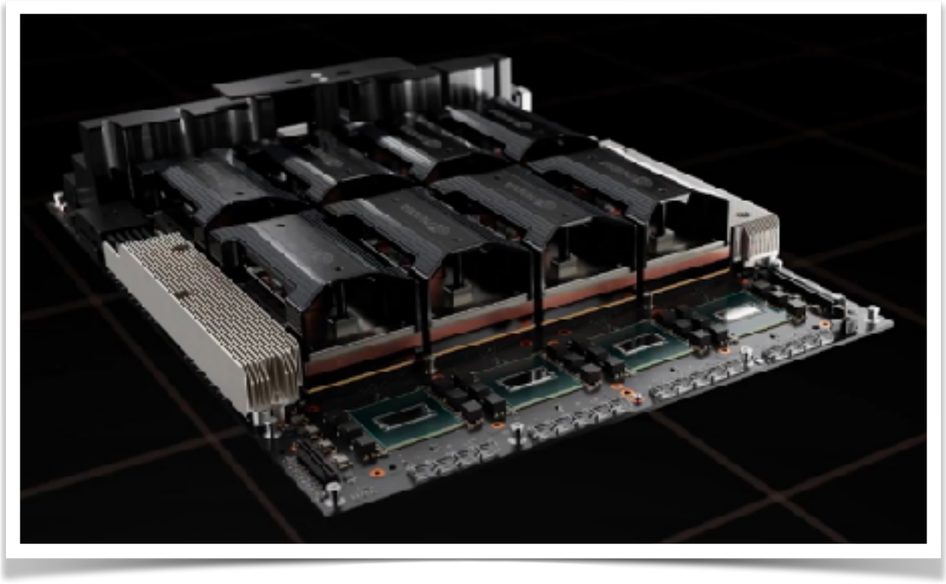
A better deal in terms of energy!

$$= 1.6 \times Energy_{old}$$

If you can add power budget…

NVIDIA Accelerator Specification Comparison			
	H100	A100 (80GB)	V100
FP32 CUDA Cores	16896	6912	5120
Tensor Cores	528	432	640
Boost Clock	~1.78GHz (Not Finalized)	1.41GHz	1.53GHz
Memory Clock	4.8Gbps HBM3	3.2Gbps HBM2e	1.75Gbps HBM2
Memory Bus Width	5120-bit	5120-bit	4096-bit
Memory Bandwidth	3TB/sec	2TB/sec	900GB/sec
VRAM	80GB	80GB	16GB/32GB
FP32 Vector	60 TFLOPS	19.5 TFLOPS	15.7 TFLOPS
FP64 Vector	30 TFLOPS	9.7 TFLOPS (1/2 FP32 rate)	7.8 TFLOPS (1/2 FP32 rate)
INT8 Tensor	2000 TOPS	624 TOPS	N/A
FP16 Tensor	1000 TFLOPS	312 TFLOPS	125 TFLOPS
TF32 Tensor	500 TFLOPS	156 TFLOPS	N/A
FP64 Tensor	60 TFLOPS	19.5 TFLOPS	N/A
Interconnect	NVLink 4 18 Links (900GB/sec)	NVLink 3 12 Links (600GB/sec)	NVLink 2 6 Links (300GB/sec)
GPU	GH100 (814mm2)	GA100 (826mm2)	GV100 (815mm2)
Transistor Count	80B	54.2B	21.1B
TDP	700W	400W	300W/350W
Manufacturing Process	TSMC 4N	TSMC 7N	TSMC 12nm FFN
Interface	SXM5	SXM4	SXM2/SXM3
Architecture	Hopper	Ampere	Volta

https://www.workstationspecialist.com/product/nvidia-tesla-a100/



https://www.servethehome.com/wp-content/uploads/2022/03/NVIDIA-GTC-2022-H100-in-HGX-H100.jpg

Announcements

- Last reading quiz due this Thursday
- iEVAL — if you fill out iEVAL, please submit a screenshot
  - You will receive a full credit reading quiz if you submit the screenshot before the in-person final examine
  - We will drop your **lowest two** reading quizzes — so that if you don't want to fill the form, you still have the lowest one dropped
- Final Exam — will release some sample questions next Tuesday at the end of the lecture
  - 12/7 3:30p—4:50p
    - Including CSMS comprehensive examine questions
    - In-person, no outside materials
  - 12/11-12/14 6pm—6pm — any consecutive three-hour slot you pick
    - Online examine
    - Including coding assessments (similar to company interviews)
    - Essay-style questions that require thorough understandings of course materials

Computer
Science &
Engineering

203

つづく