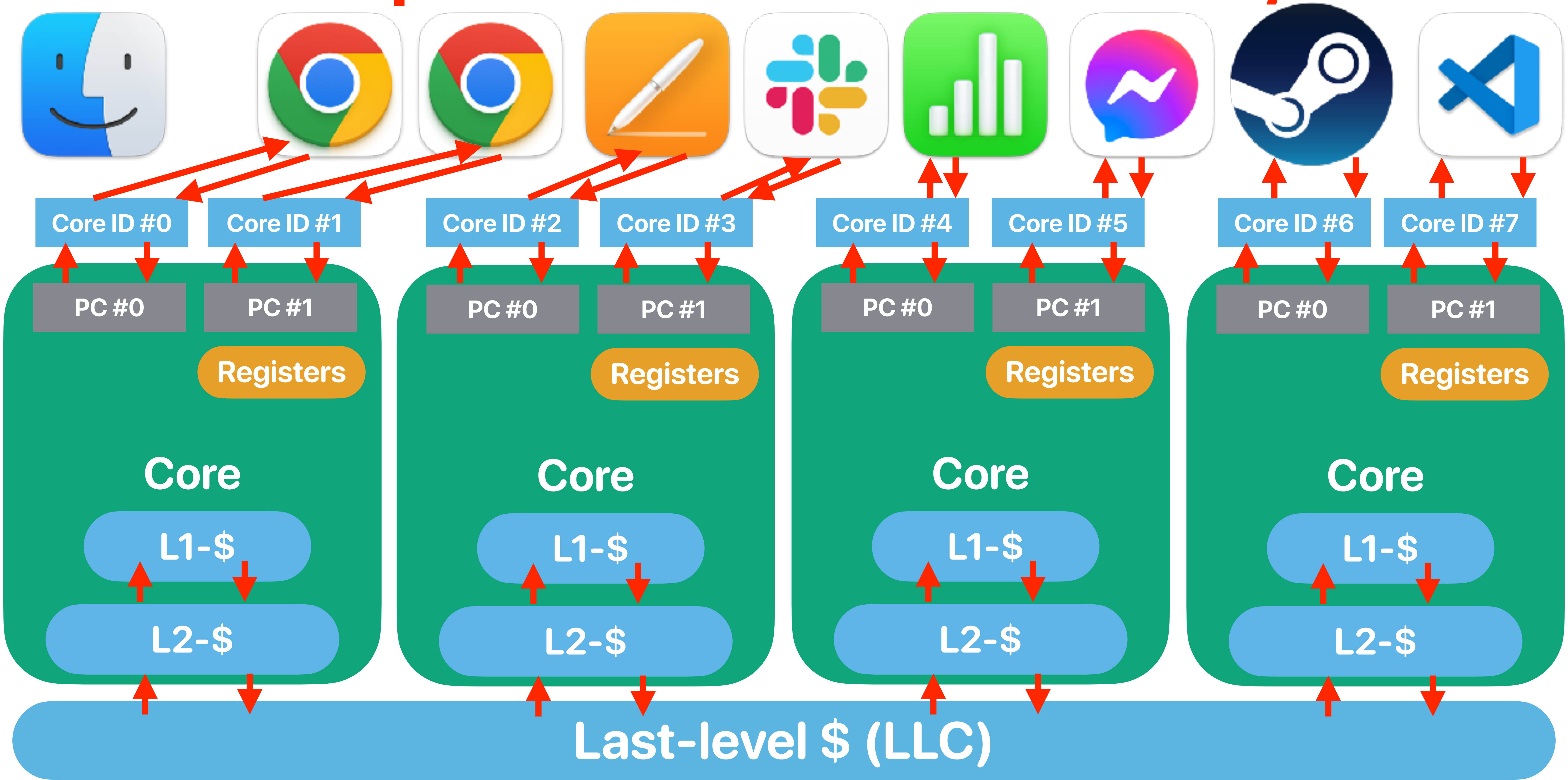
Multithreaded Architectures and Programming on Multithreaded Architectures (2): Never say never

Hung-Wei Tseng

Modern processors have both CMP/SMT



Parallel programming

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

Recap: 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

Take-aways: parallel programming

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

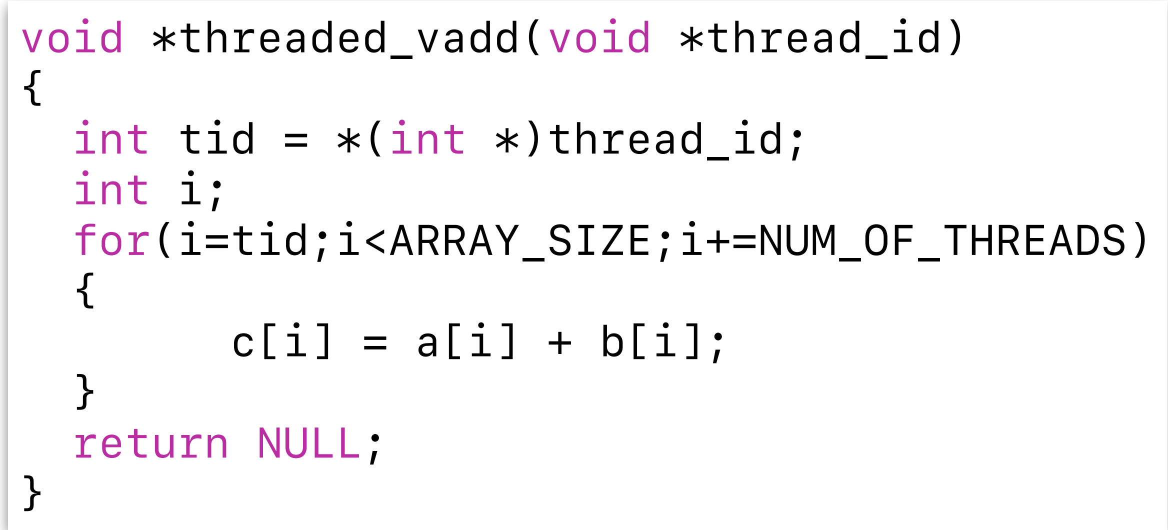
Parallel Programming & Architectural Supports for Parallel Programming (cont.)

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

Performance comparison

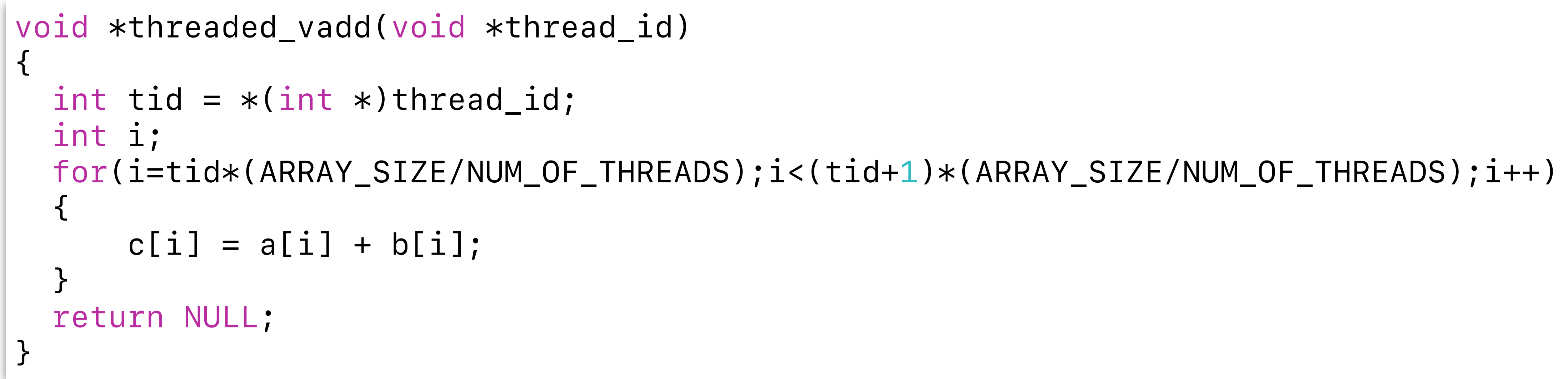
- Comparing implementations of thread_vadd — L and R, please identify which one will be performing better and why

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;
}
```

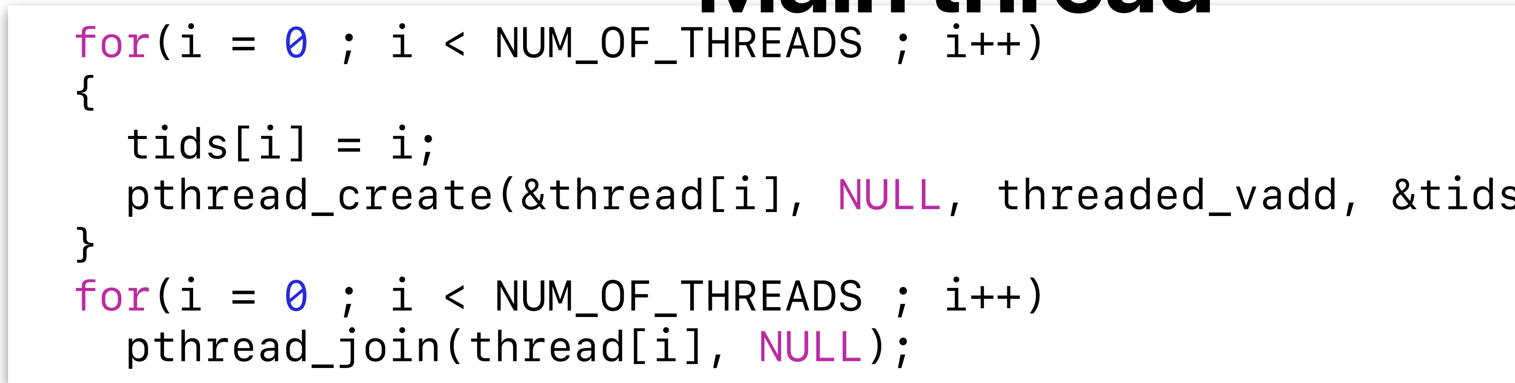
Version R



```
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

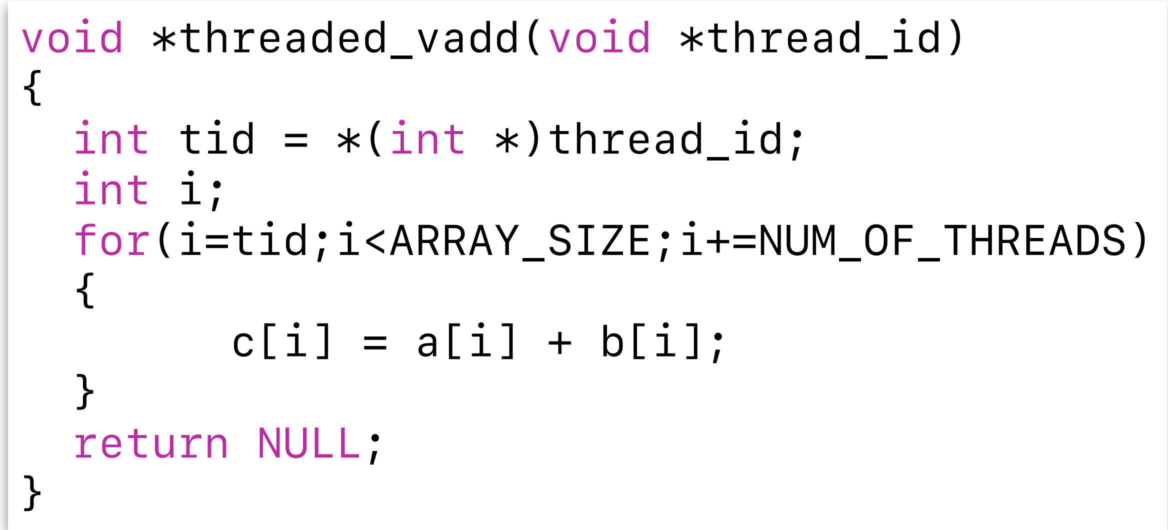


```
  for(i = 0 ; i < NUM_OF_THREADS ; i++)
  {
    tids[i] = i;
    pthread_create(&thread[i], NULL, threaded_vadd, &tids
  }
  for(i = 0 ; i < NUM_OF_THREADS ; i++)
    pthread_join(thread[i], NULL);
```

Performance comparison

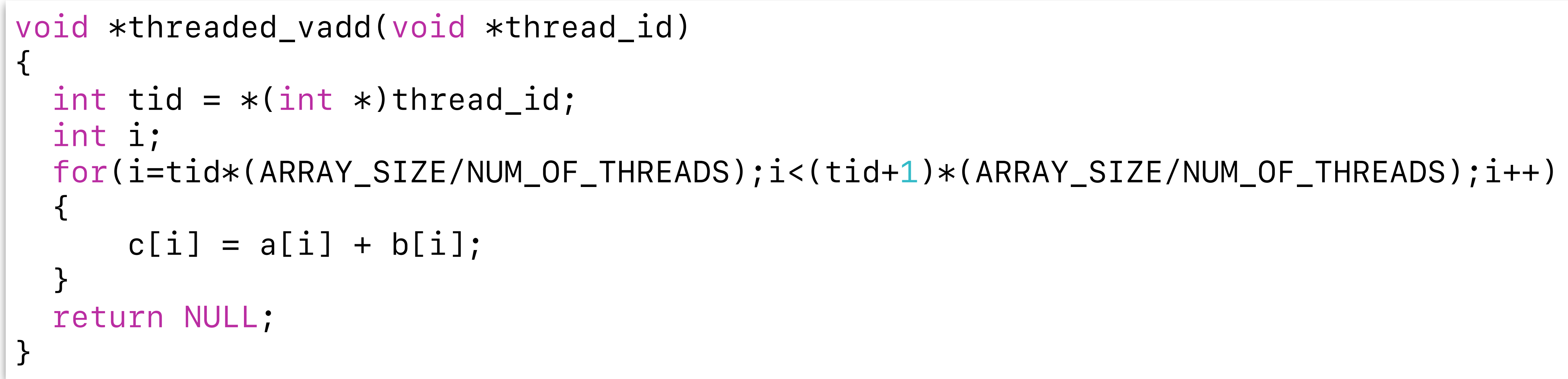
- Comparing implementations of thread_vadd — L and R, please identify which one will be performing better and why

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

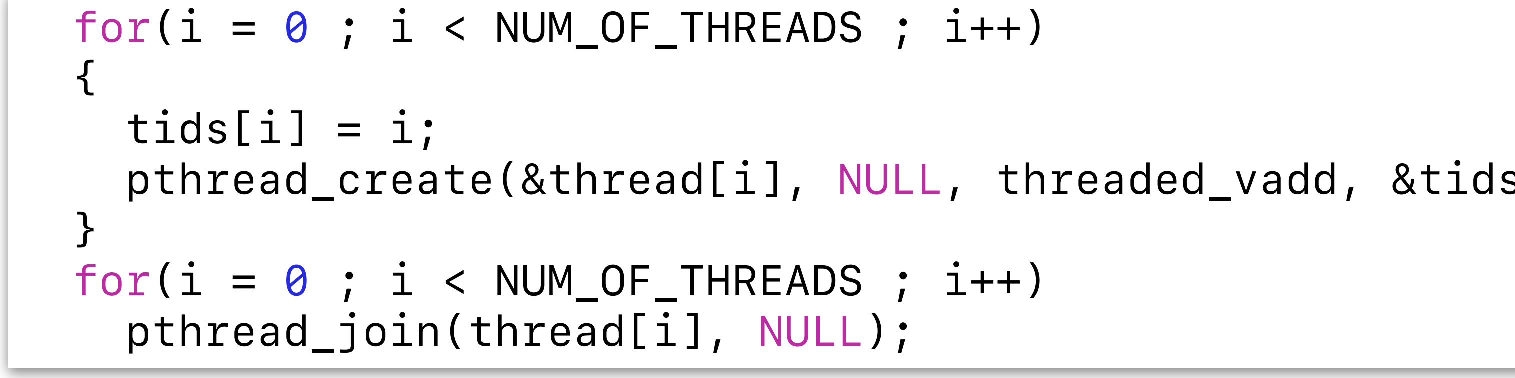
B. R is better, because the cache miss rate is lower

C. L is better, because the instruction count is lower

D. R is better, because the instruction count is lower

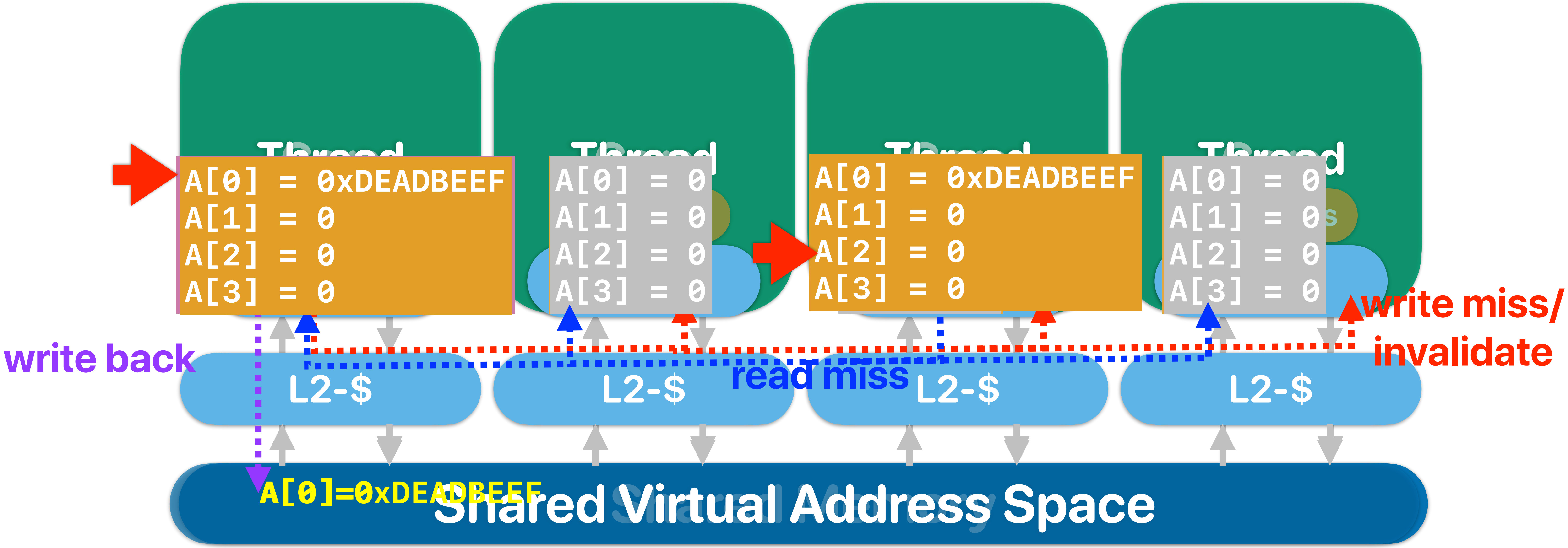
E. Both are about the same

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);
```

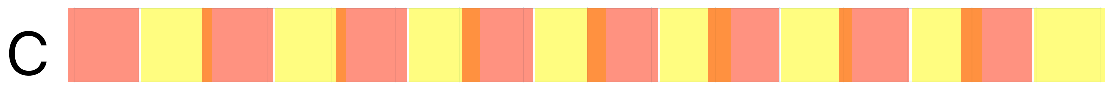
Cache coherency



L v.s. R

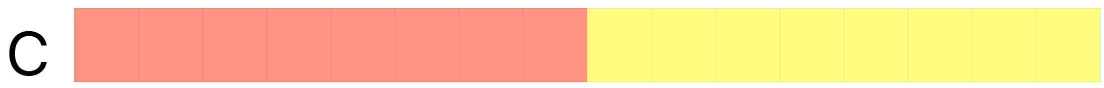
Version L

```c
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

```c
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.

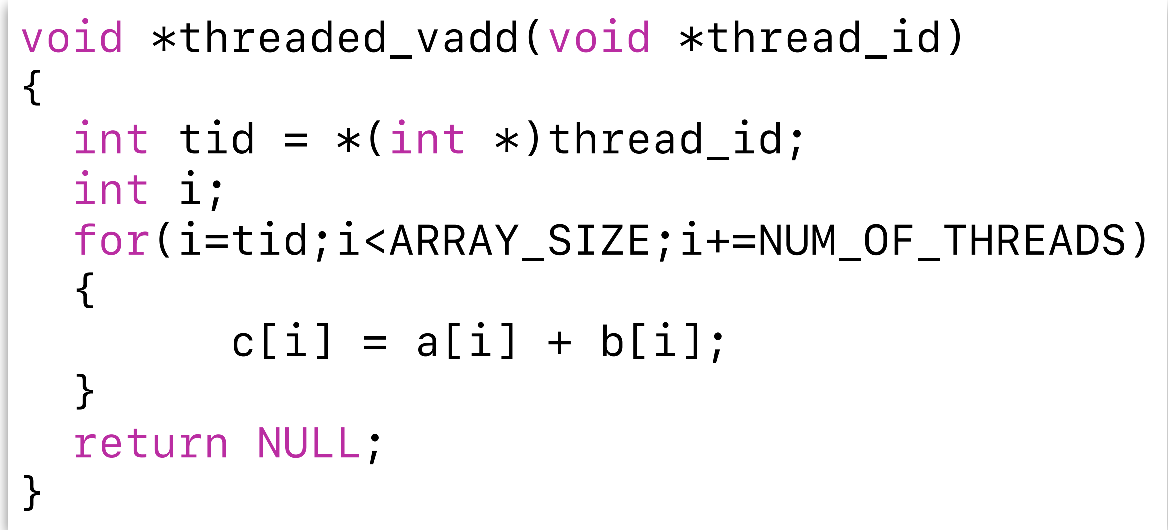
False sharing

- 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

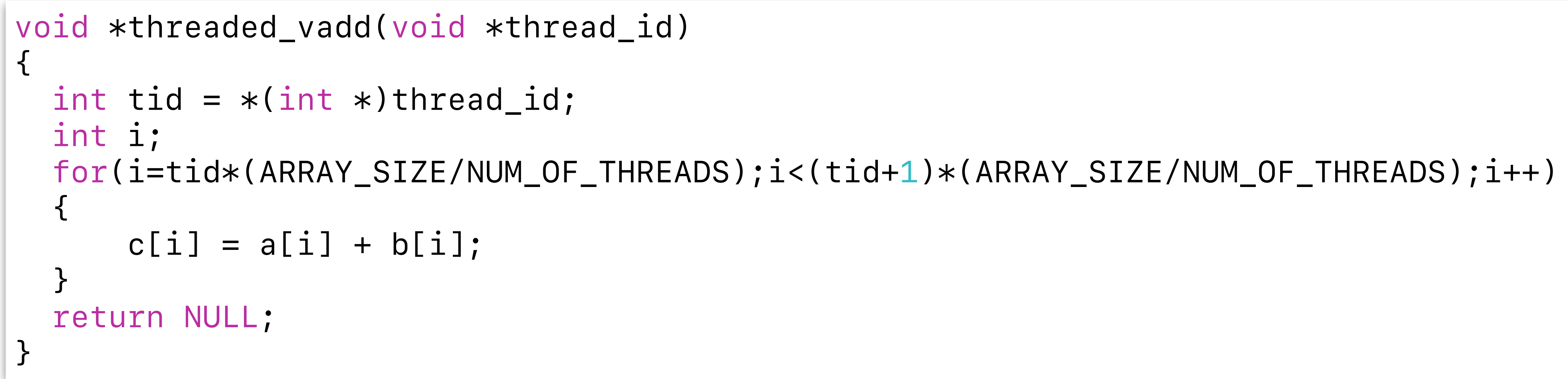
- Comparing implementations of thread_vadd — L and R, please identify which one will be performing better and why

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

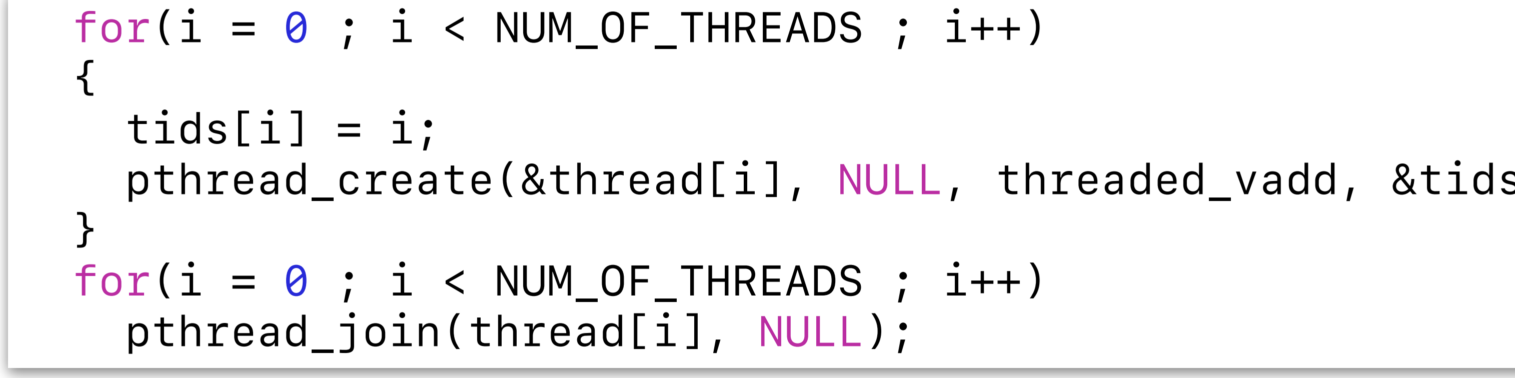
[✓ B. R is better, because the cache miss rate is lower]

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);
```

Take-aways: parallel programming

- Cache coherency only guarantees that everyone would eventually have a coherent view of data, but not when
- Cache coherency may create unexpected cache invalidations/misses if you do it wrong

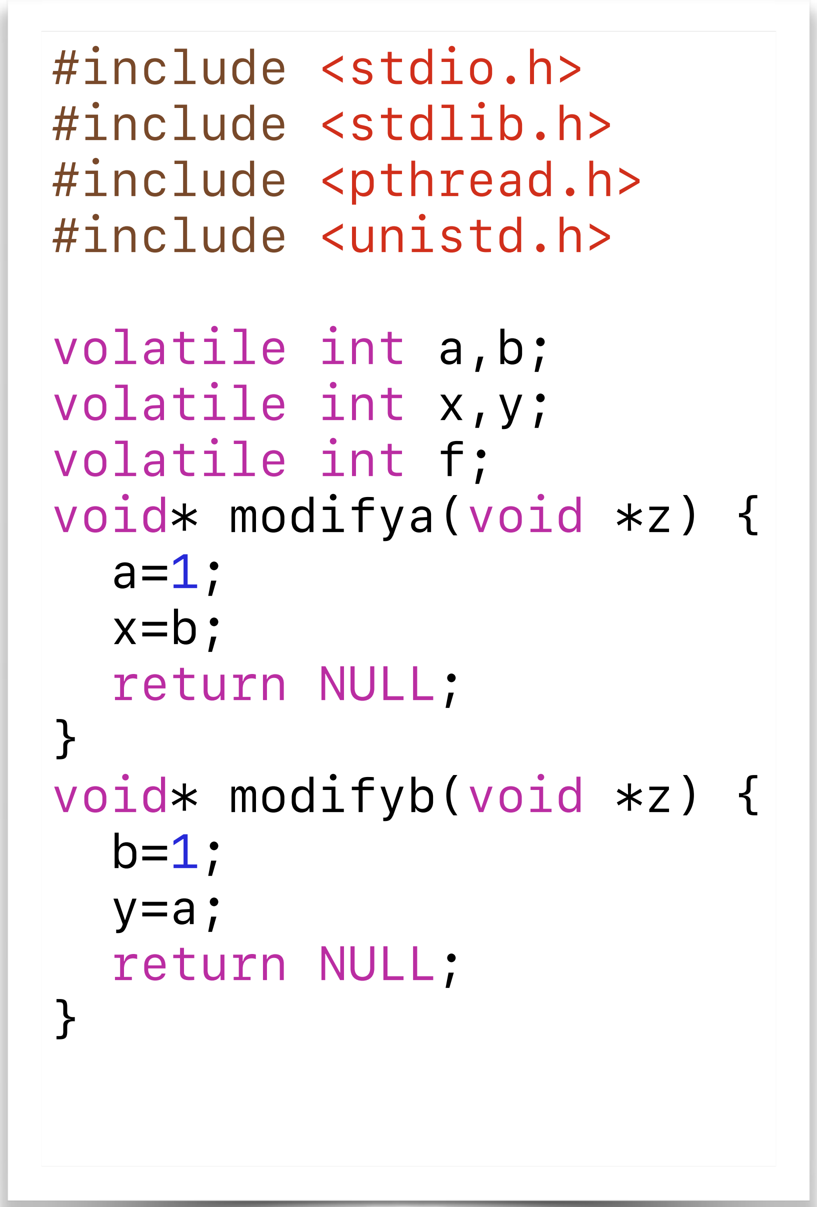
https://www.pollev.com/hungweitseng close in 1:30

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?

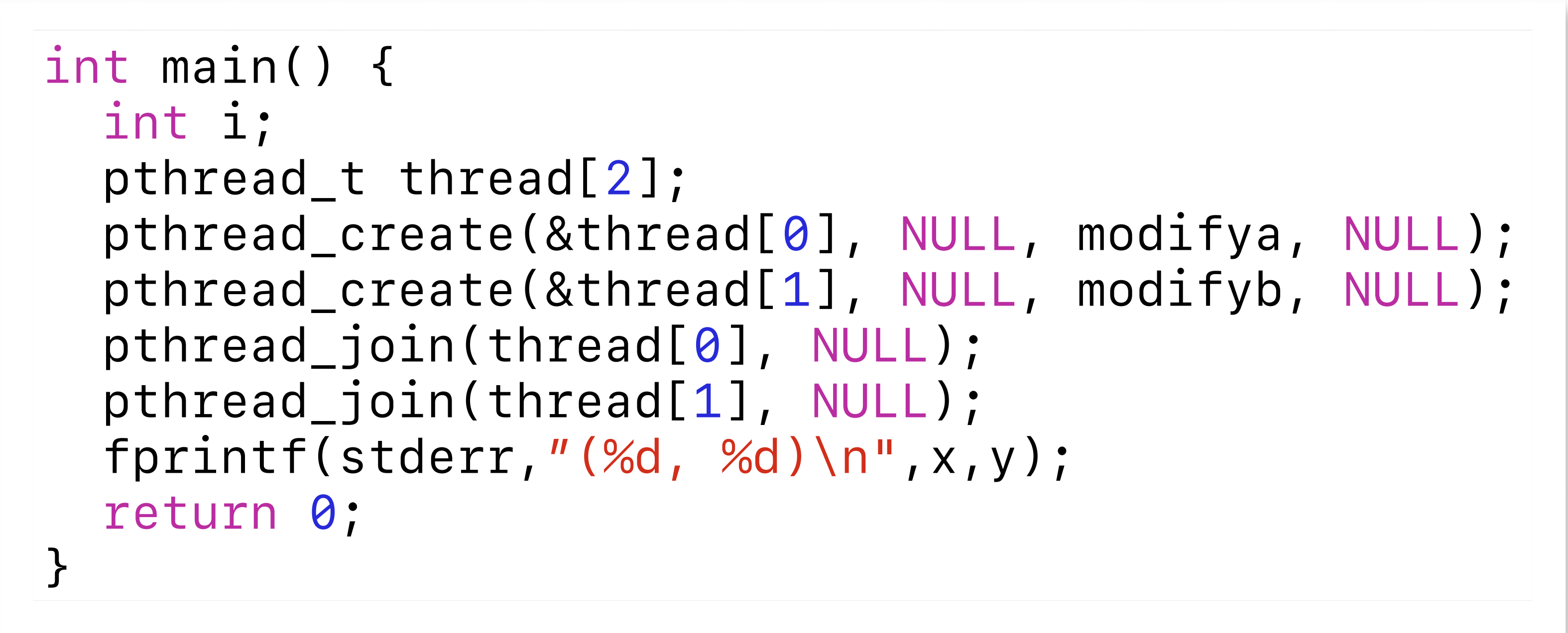
① (0, 0)
② (0, 1)
③ (1, 0)
④ (1, 1)

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



```
#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;
}
```



```
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;
}
```

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)

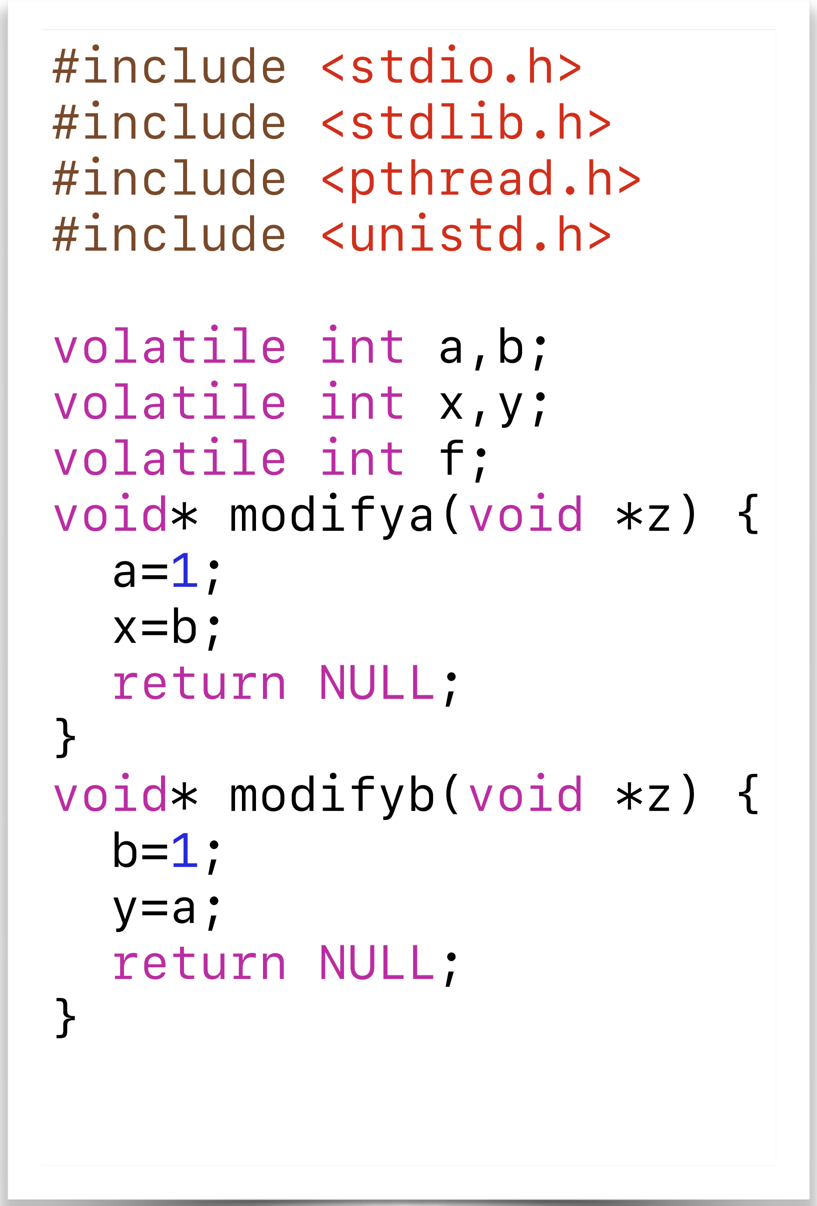
A. 0

B. 1

C. 2

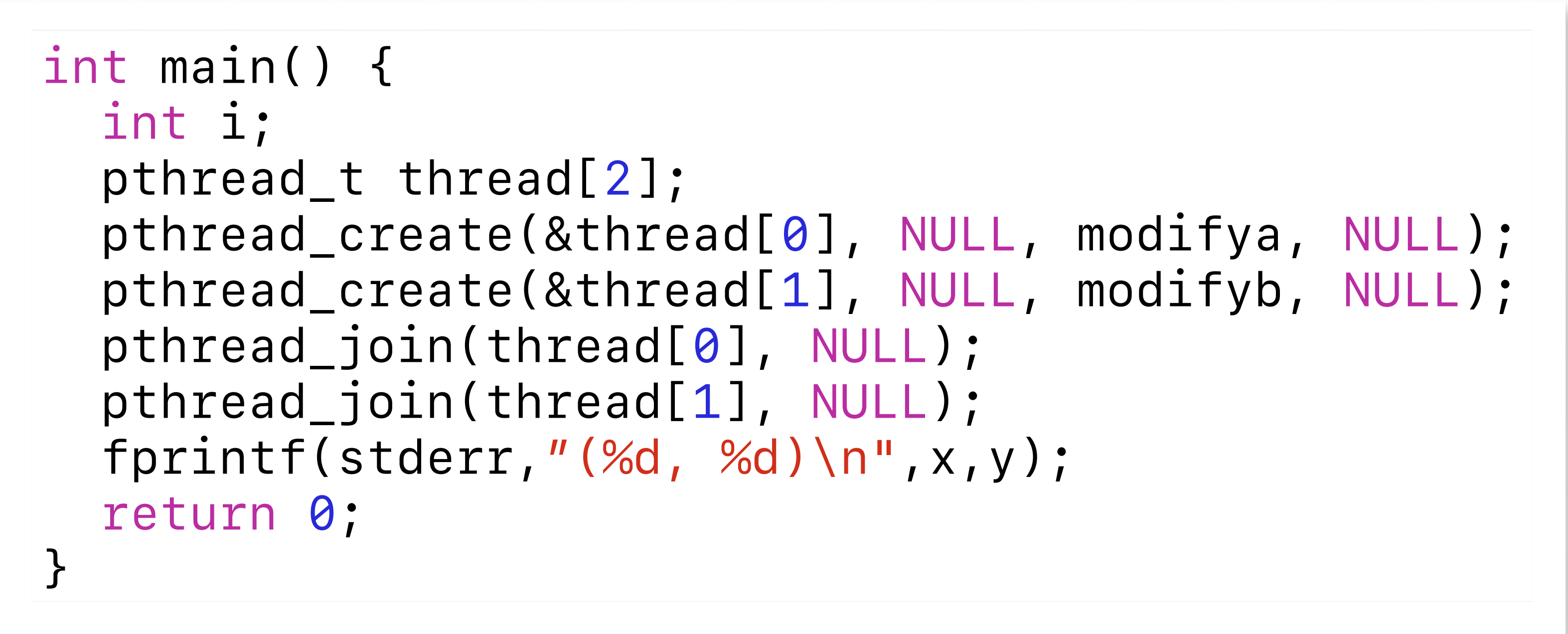
D. 3

E. 4



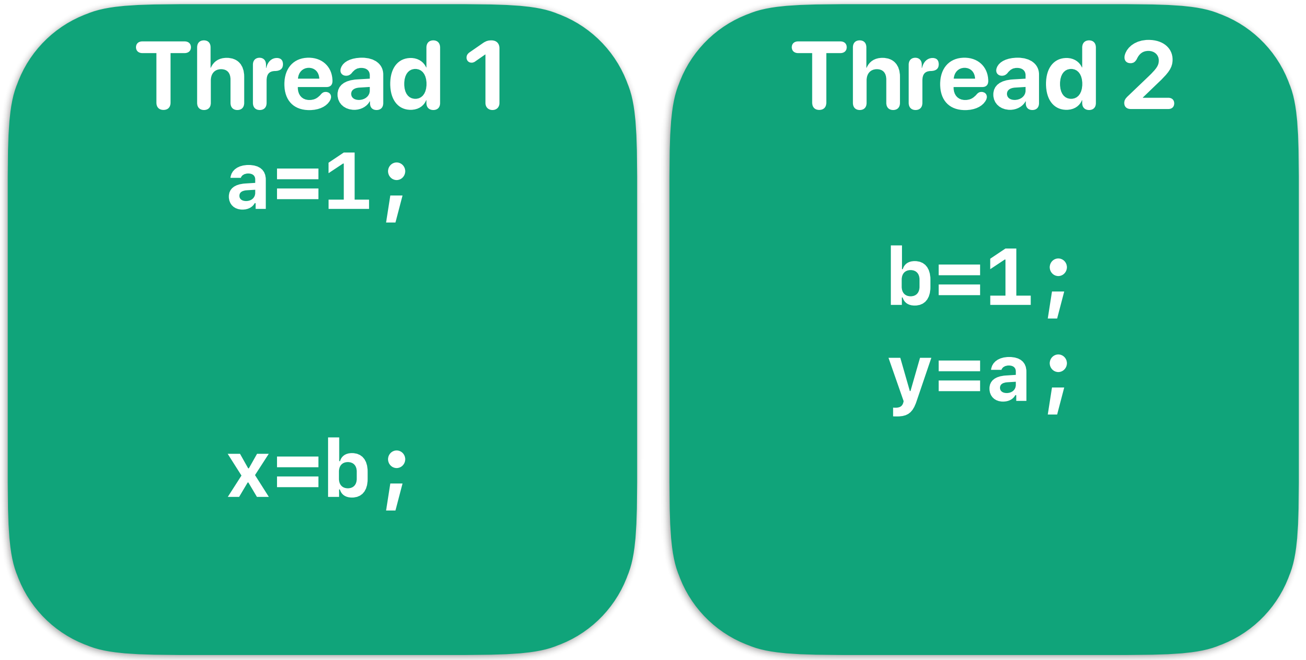
```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

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;
}
```



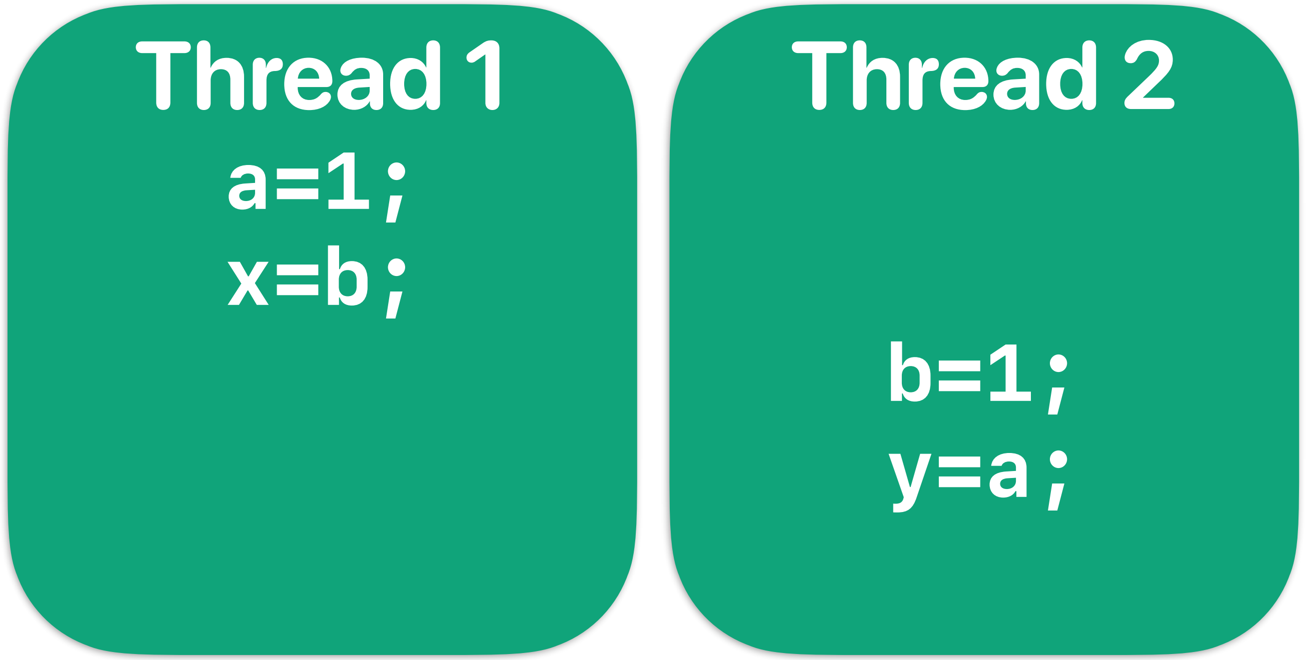
```c
int main() {
  int i;
  pthread_t thread[2];
  pthread_create(&thread[0], NULL, modifya, NULL);
  pthread_create(&thread[1], NULL, modifyb, NULL);
  pthread_join(thread[0], NULL);
  pthread_join(thread[1], NULL);
  fprintf(stderr,"(%d, %d)\n",x,y);
  return 0;
}
```

Possible scenarios



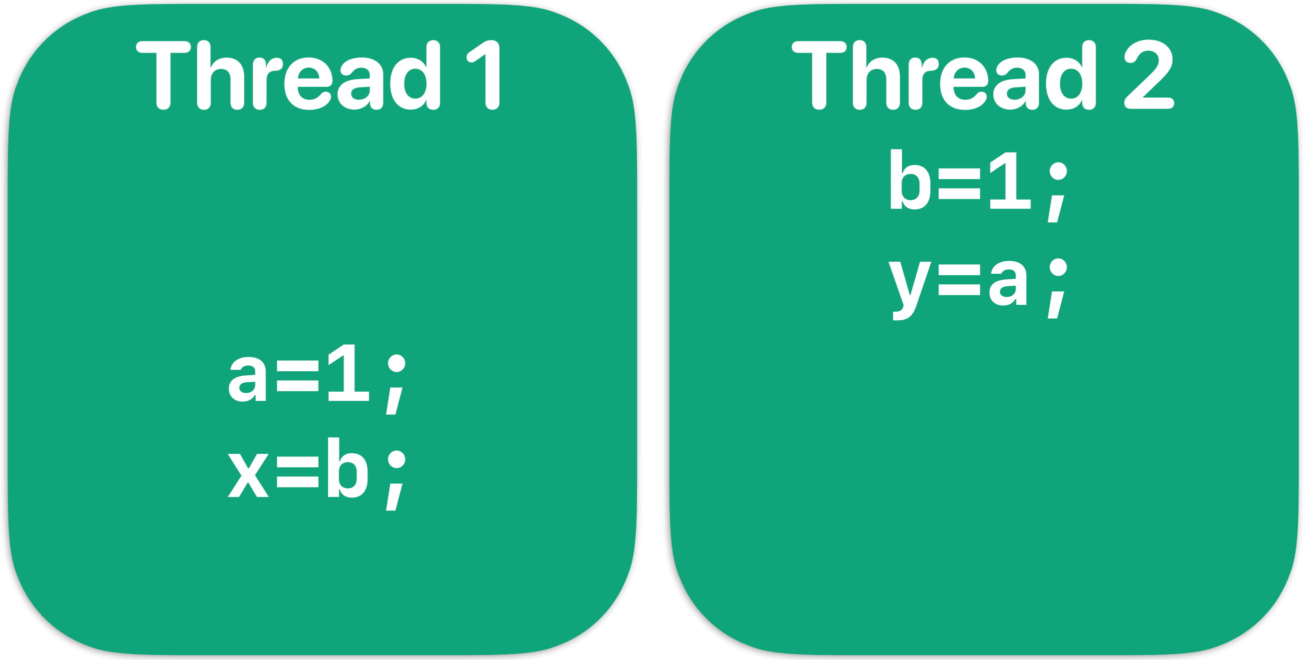
```
Thread 1    Thread 2
a=1;
            b=1;
            y=a;
x=b;
```

(1,1)



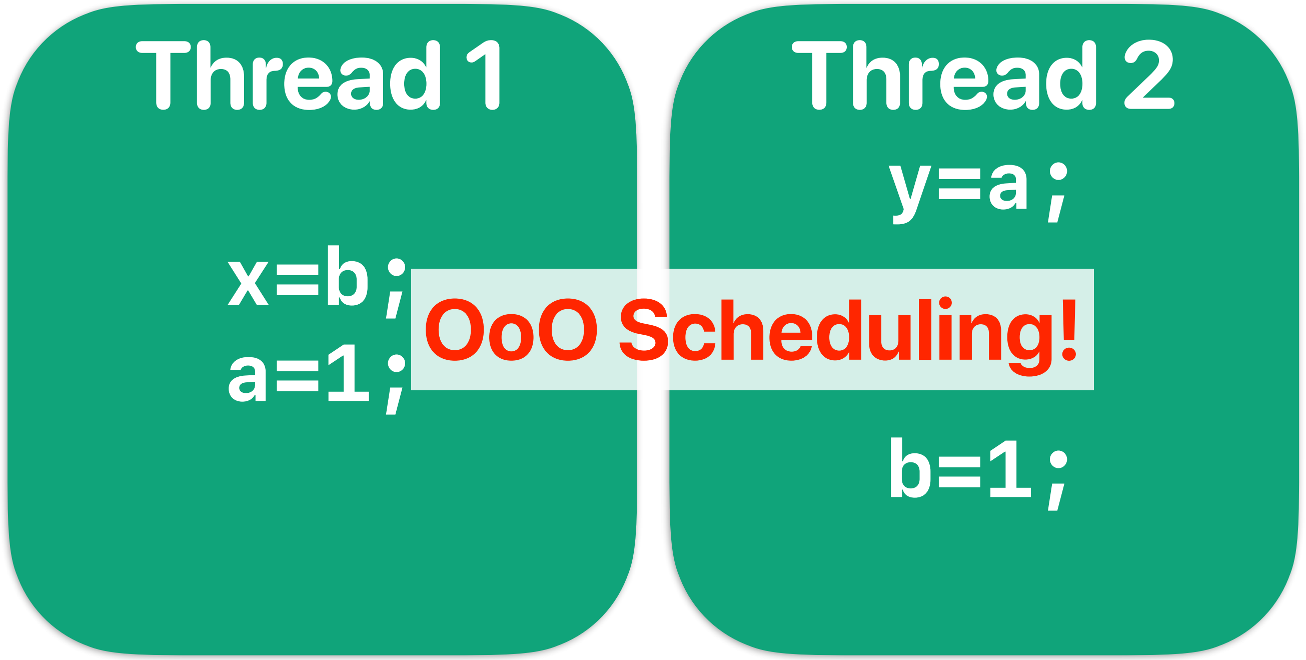
```
Thread 1    Thread 2
a=1;
x=b;
            b=1;
            y=a;
```

(0,1)



```
Thread 1    Thread 2
            b=1;
            y=a;
a=1;
x=b;
```

(1,0)



```
Thread 1    Thread 2
            y=a;
x=b;
a=1;
            b=1;
```

OoO Scheduling!

(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)

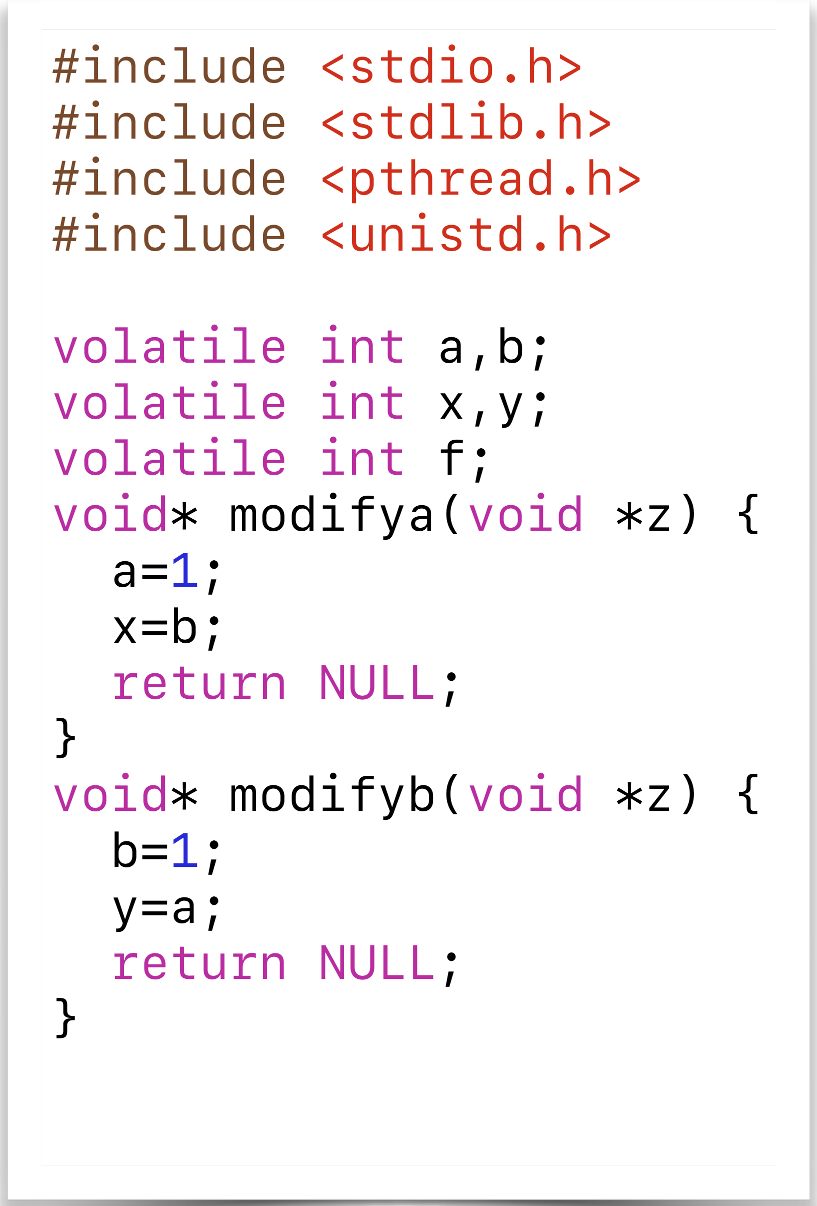
A. 0

B. 1

C. 2

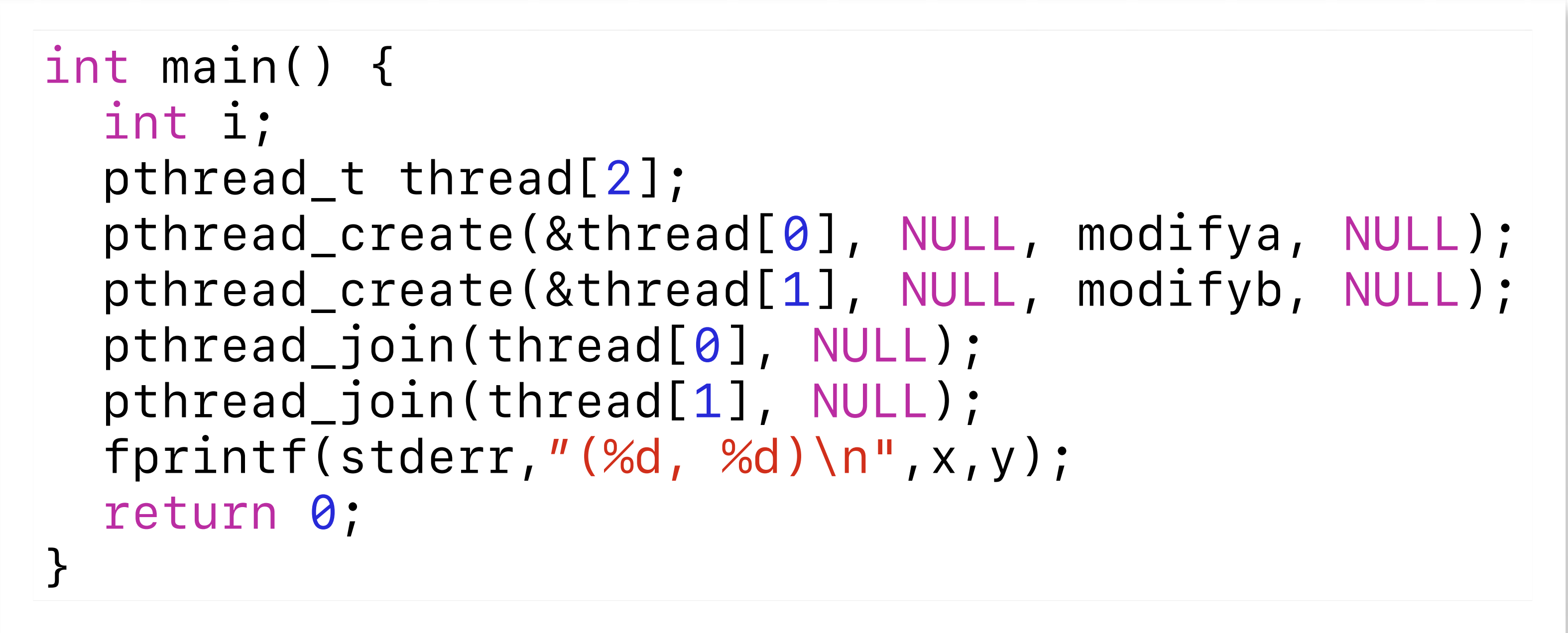
D. 3

[✓ E. 4]



```
#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;
}
```



```
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;
}
```

Take-aways: parallel programming

- Cache coherency only guarantees that everyone would eventually have a coherent view of data, but not when
- Cache coherency may create unexpected cache invalidations/misses if you do it wrong
- Processor behaviors are non-deterministic
    - You cannot predict which processor is going faster
    - You cannot predict when OS is going to schedule your thread
    - You cannot predict when the processor is going to schedule an instruction

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: parallel programming

- Cache coherency only guarantees that everyone would eventually have a coherent view of data, but not when
- Cache coherency may create unexpected cache invalidations/misses if you do it wrong
- Processor behaviors are non-deterministic
  - You cannot predict which processor is going faster
  - You cannot predict when OS is going to schedule your thread
  - You cannot predict when the processor is going to schedule an instruction
- Cache consistency is hard to support

Multithreaded Architectures in Google Search

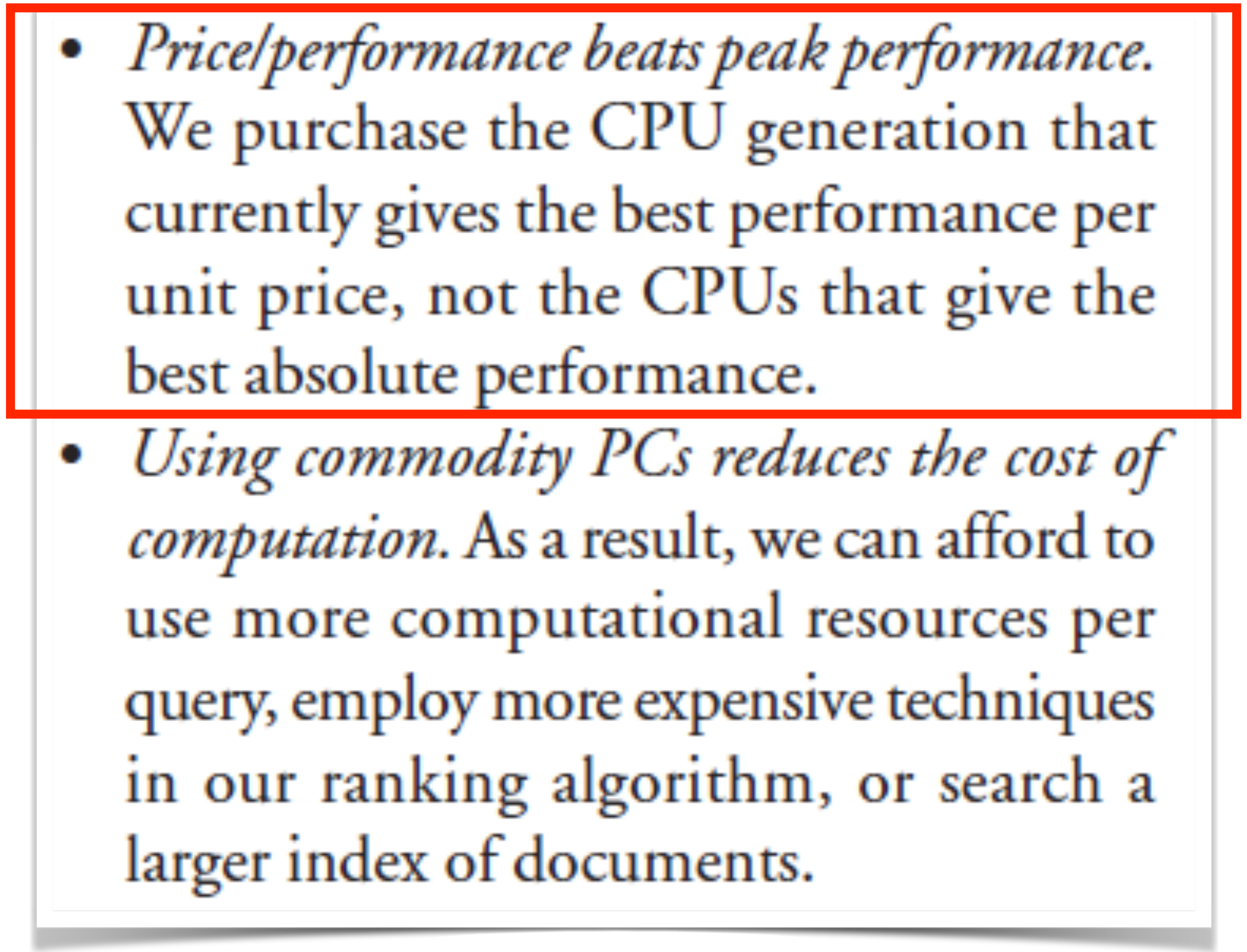
Web search for a planet: The Google cluster architecture

Luiz Andre Barroso, Jeffery Dean, Urs Holzle

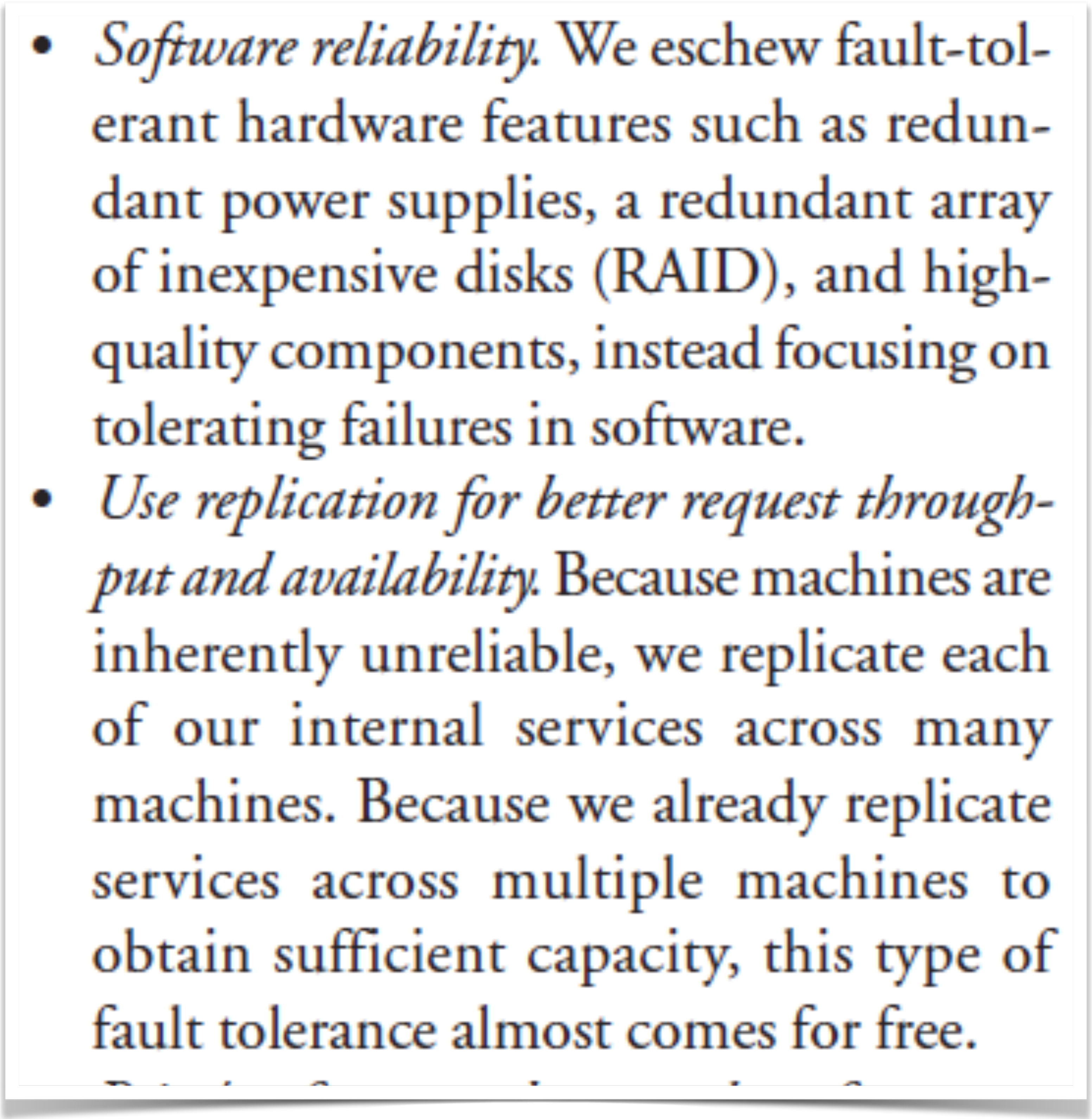
Google

IEEE Micro 2003

Google Datacenter Goals



- *Price/performance beats peak performance.* We purchase the CPU generation that currently gives the best performance per unit price, not the CPUs that give the best absolute performance.
- *Using commodity PCs reduces the cost of computation.* As a result, we can afford to use more computational resources per query, employ more expensive techniques in our ranking algorithm, or search a larger index of documents.



- *Software reliability.* We eschew fault-tolerant hardware features such as redundant power supplies, a redundant array of inexpensive disks (RAID), and high-quality components, instead focusing on tolerating failures in software.
- *Use replication for better request throughput and availability.* Because machines are inherently unreliable, we replicate each of our internal services across many machines. Because we already replicate services across multiple machines to obtain sufficient capacity, this type of fault tolerance almost comes for free.

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

What kind of processors Google search needs

- If we are designing a processor just for Google search or similar type of applications, how many of the following targets/features would fulfill the demand?

  ① Can execute many instructions from the same process/thread simultaneously

  ② Can execute many processes/threads simultaneously

  ③ Can predict branch outcome accurately

  ④ Have very large cache capacity

  A. 0

  B. 1

  C. 2

  D. 3

  E. 4

What kind of processors Google search needs

- If we are designing a processor just for Google search or similar type of applications, how many of the following targets/features would fulfill the demand?
  - ① Can execute many instructions from the same process/thread simultaneously
  - ✓ ② Can execute many processes/threads simultaneously
  - ✓ ③ Can predict branch outcome accurately
  - ④ Have very large cache capacity

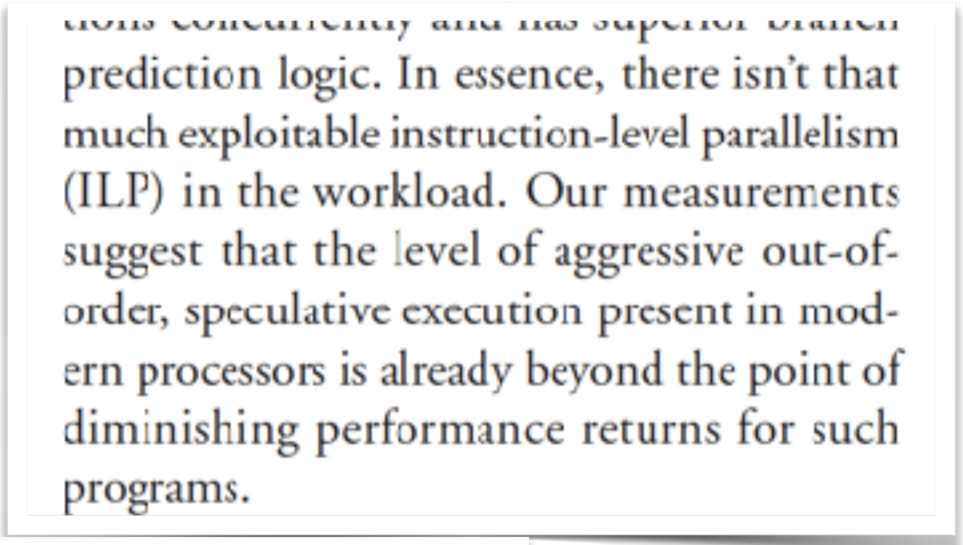
  A. 0

  B. 1

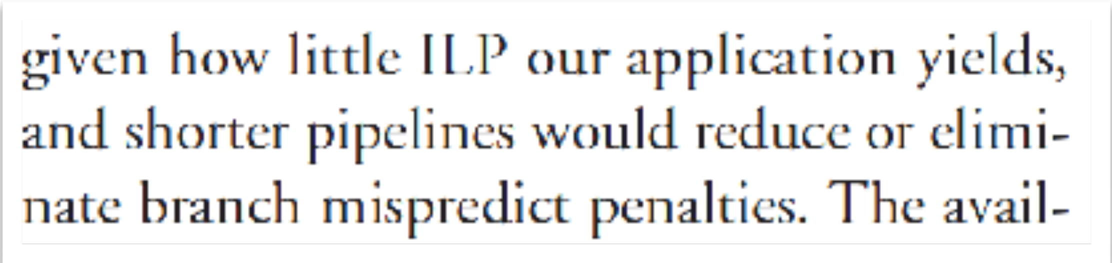
  [✓ C. 2]

  D. 3

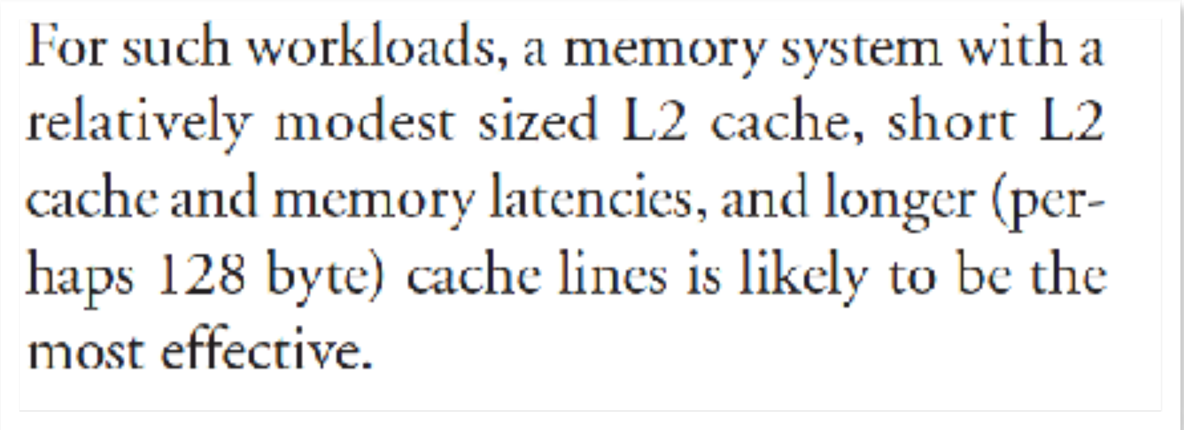
  E. 4



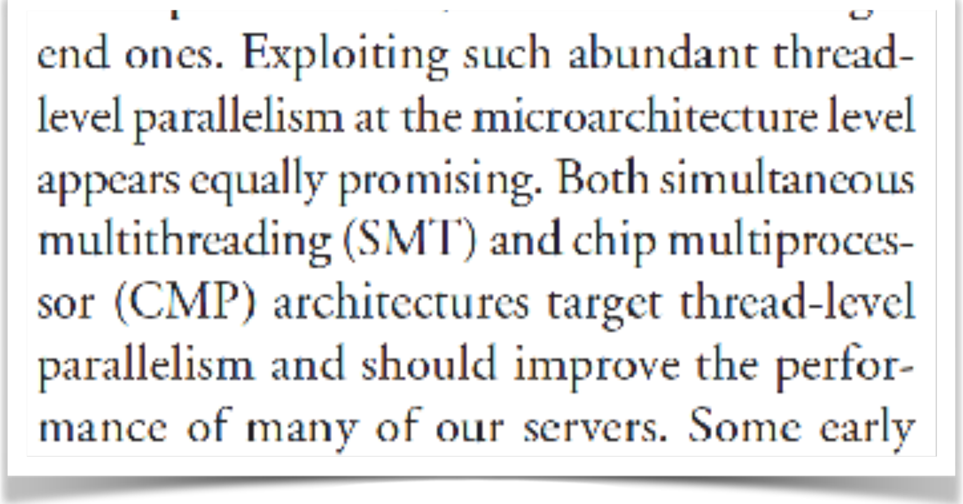
tions concurrently and has superior branch prediction logic. In essence, there isn't that much exploitable instruction-level parallelism (ILP) in the workload. Our measurements suggest that the level of aggressive out-of-order, speculative execution present in modern processors is already beyond the point of diminishing performance returns for such programs.



given how little ILP our application yields, and shorter pipelines would reduce or eliminate branch mispredict penalties. The avail-



For such workloads, a memory system with a relatively modest sized L2 cache, short L2 cache and memory latencies, and longer (perhaps 128 byte) cache lines is likely to be the most effective.

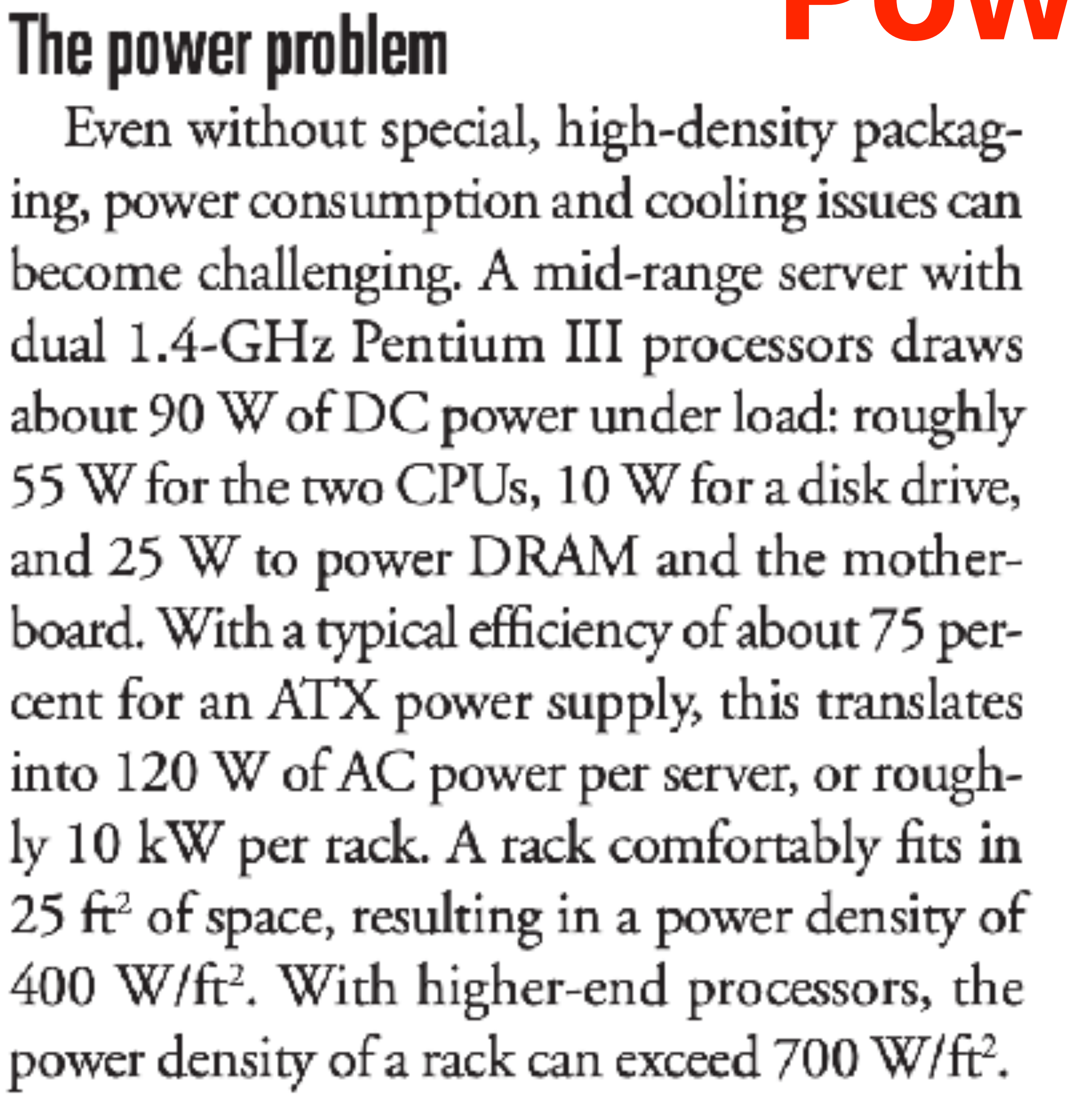


end ones. Exploiting such abundant thread-level parallelism at the microarchitecture level appears equally promising. Both simultaneous multithreading (SMT) and chip multiprocessor (CMP) architectures target thread-level parallelism and should improve the performance of many of our servers. Some early

Metrics we care about data center design

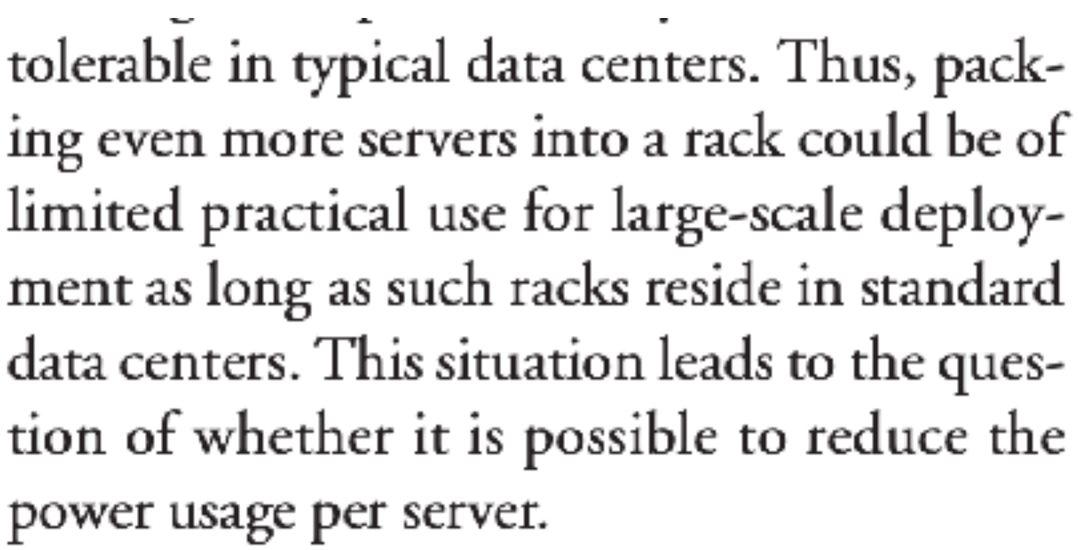
- Costs — machine architecture, distributed system architecture, replication strategies
- Power — machine architecture
- Energy — machine architecture
- Space-efficiency — erasure coding, replication, distributed
- Throughput — replication, distributed
- Reliability — replication

Power problems in Google

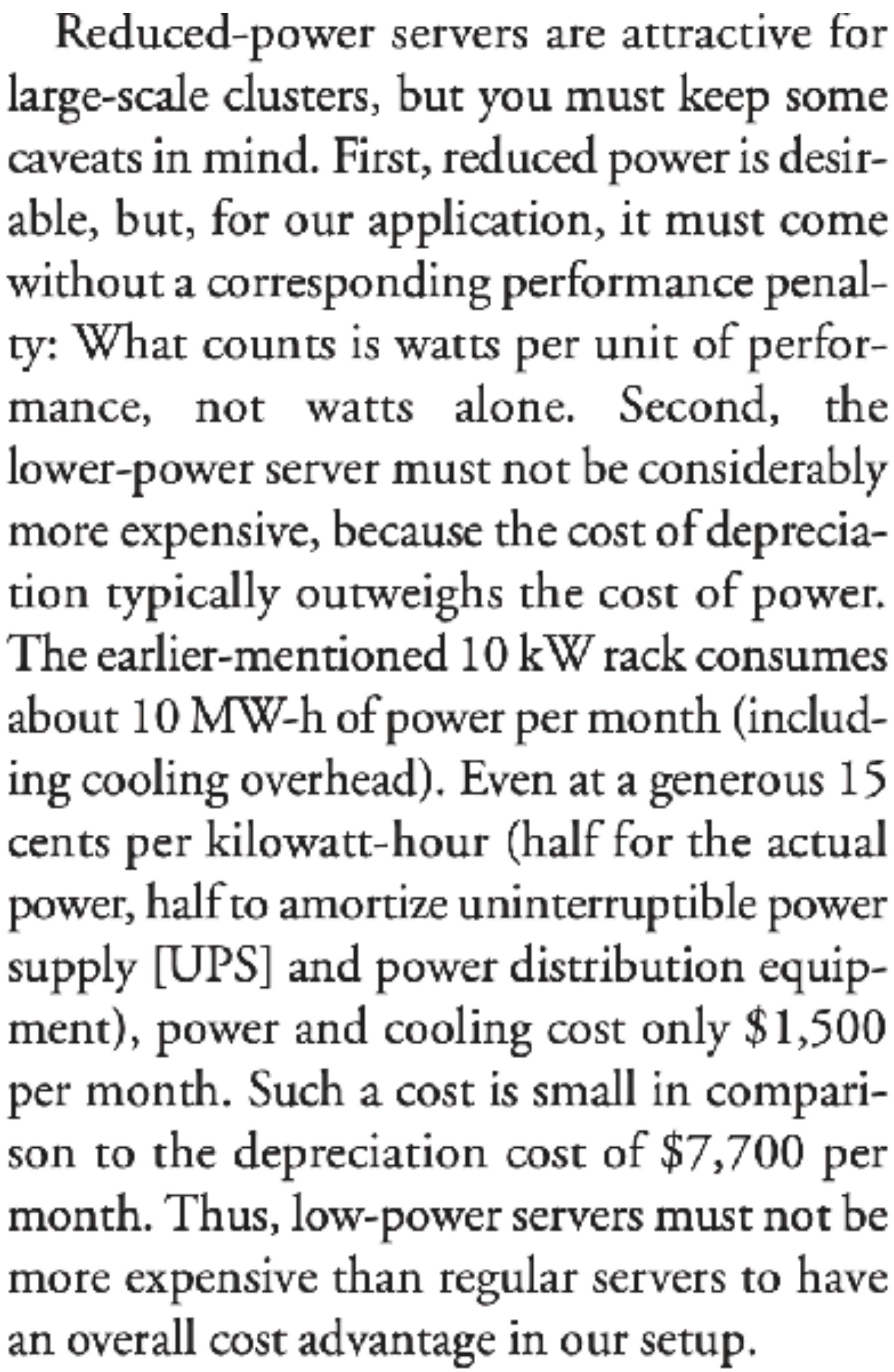


The power problem

Even without special, high-density packaging, power consumption and cooling issues can become challenging. A mid-range server with dual 1.4-GHz Pentium III processors draws about 90 W of DC power under load: roughly 55 W for the two CPUs, 10 W for a disk drive, and 25 W to power DRAM and the motherboard. With a typical efficiency of about 75 percent for an ATX power supply, this translates into 120 W of AC power per server, or roughly 10 kW per rack. A rack comfortably fits in 25 ft^2 of space, resulting in a power density of 400 W/ft^2. With higher-end processors, the power density of a rack can exceed 700 W/ft^2.

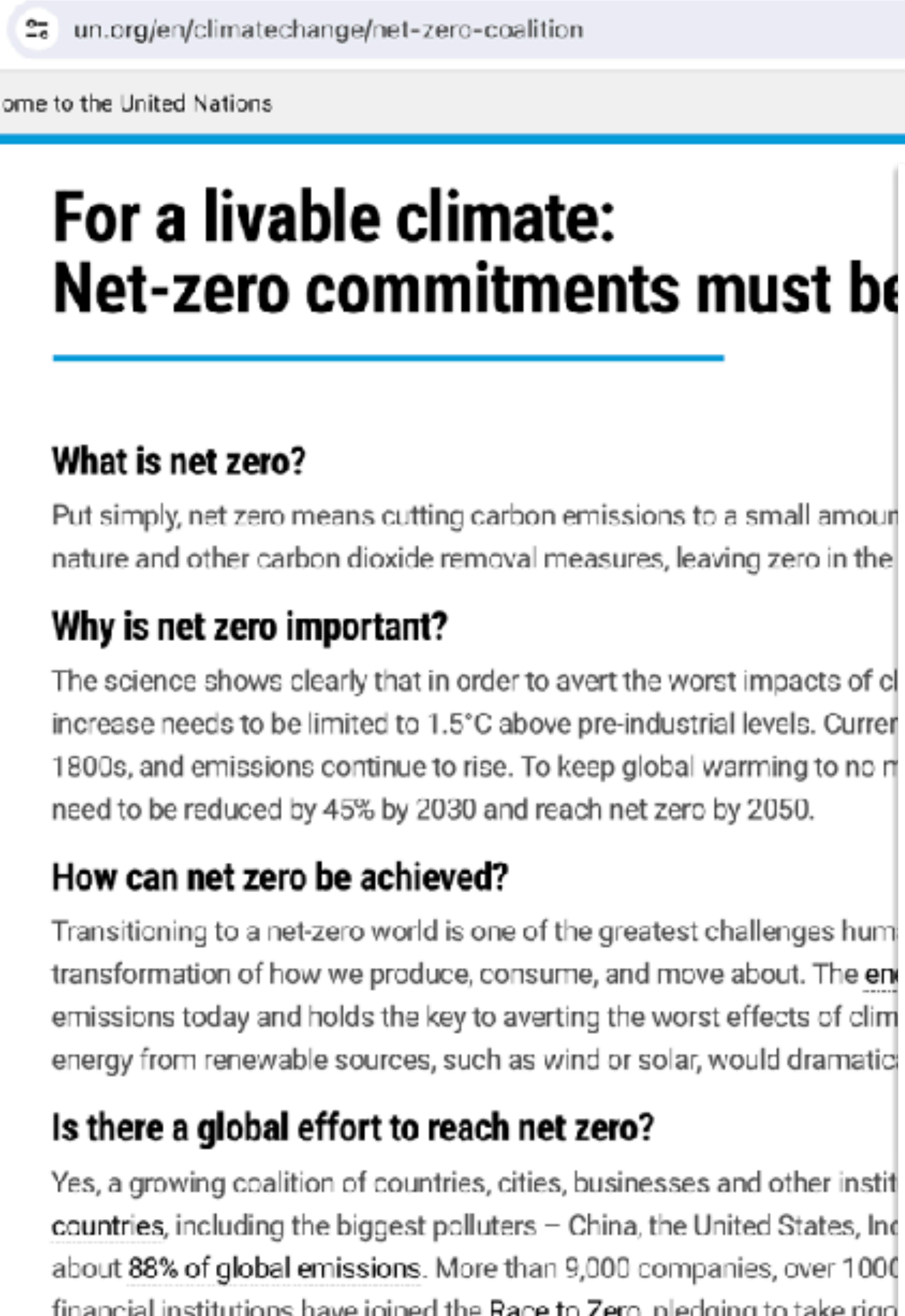


tolerable in typical data centers. Thus, packing even more servers into a rack could be of limited practical use for large-scale deployment as long as such racks reside in standard data centers. This situation leads to the question of whether it is possible to reduce the power usage per server.



Reduced-power servers are attractive for large-scale clusters, but you must keep some caveats in mind. First, reduced power is desirable, but, for our application, it must come without a corresponding performance penalty: What counts is watts per unit of performance, not watts alone. Second, the lower-power server must not be considerably more expensive, because the cost of depreciation typically outweighs the cost of power. The earlier-mentioned 10 kW rack consumes about 10 MW-h of power per month (including cooling overhead). Even at a generous 15 cents per kilowatt-hour (half for the actual power, half to amortize uninterruptible power supply [UPS] and power distribution equipment), power and cooling cost only $1,500 per month. Such a cost is small in comparison to the depreciation cost of $7,700 per month. Thus, low-power servers must not be more expensive than regular servers to have an overall cost advantage in our setup.

It's changing — Carbon footprint/emission



un.org/en/climatechange/net-zero-coalition

For a livable climate: Net-zero commitments must be

What is net zero?

Put simply, net zero means cutting carbon emissions to a small amoun
nature and other carbon dioxide removal measures, leaving zero in the

Why is net zero important?

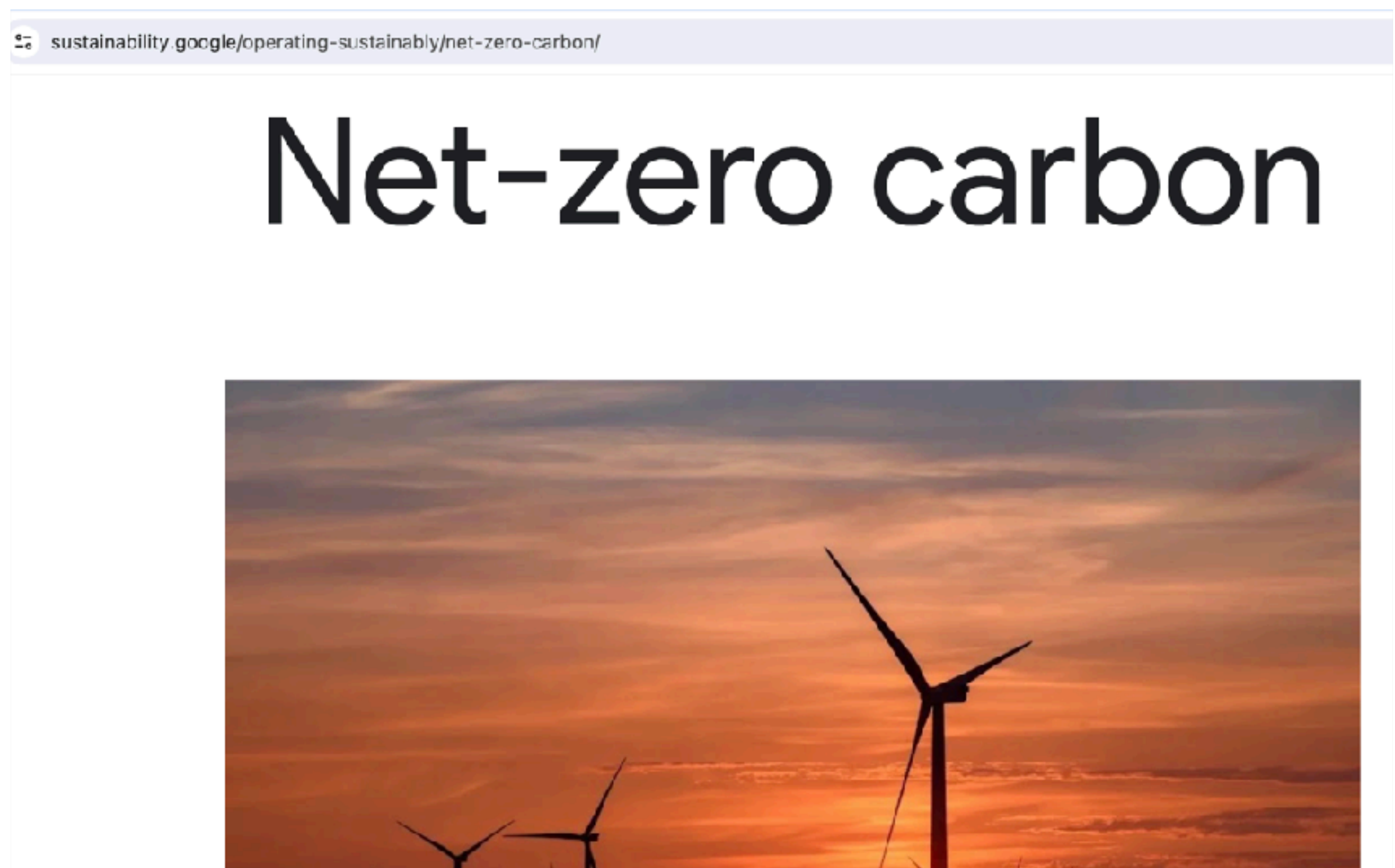
The science shows clearly that in order to avert the worst impacts of cl
increase needs to be limited to 1.5°C above pre-industrial levels. Curren
1800s, and emissions continue to rise. To keep global warming to no m
need to be reduced by 45% by 2030 and reach net zero by 2050.

How can net zero be achieved?

Transitioning to a net-zero world is one of the greatest challenges hum
transformation of how we produce, consume, and move about. The en
emissions today and holds the key to averting the worst effects of clim
energy from renewable sources, such as wind or solar, would dramatic

Is there a global effort to reach net zero?

Yes, a growing coalition of countries, cities, businesses and other instit
countries, including the biggest polluters – China, the United States, Inc
about 88% of global emissions. More than 9,000 companies, over 1000
financial institutions have joined the Race to Zero, pledging to take rigo



sustainability.google/operating-sustainably/net-zero-carbon/

Net-zero carbon

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 Lowering the frequency helps reducing the power consumption
  ⑤ A CPU operating at 20% of its top frequency can still consume 33% of the peak power
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

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 Lowering the frequency helps reducing the power consumption

  ⑤ A CPU operating at 20% of its top frequency 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
  - Dynamic power
  - Leakage power
- Energy =$Power \times Execution\_Time$
  - 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

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

  - 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)

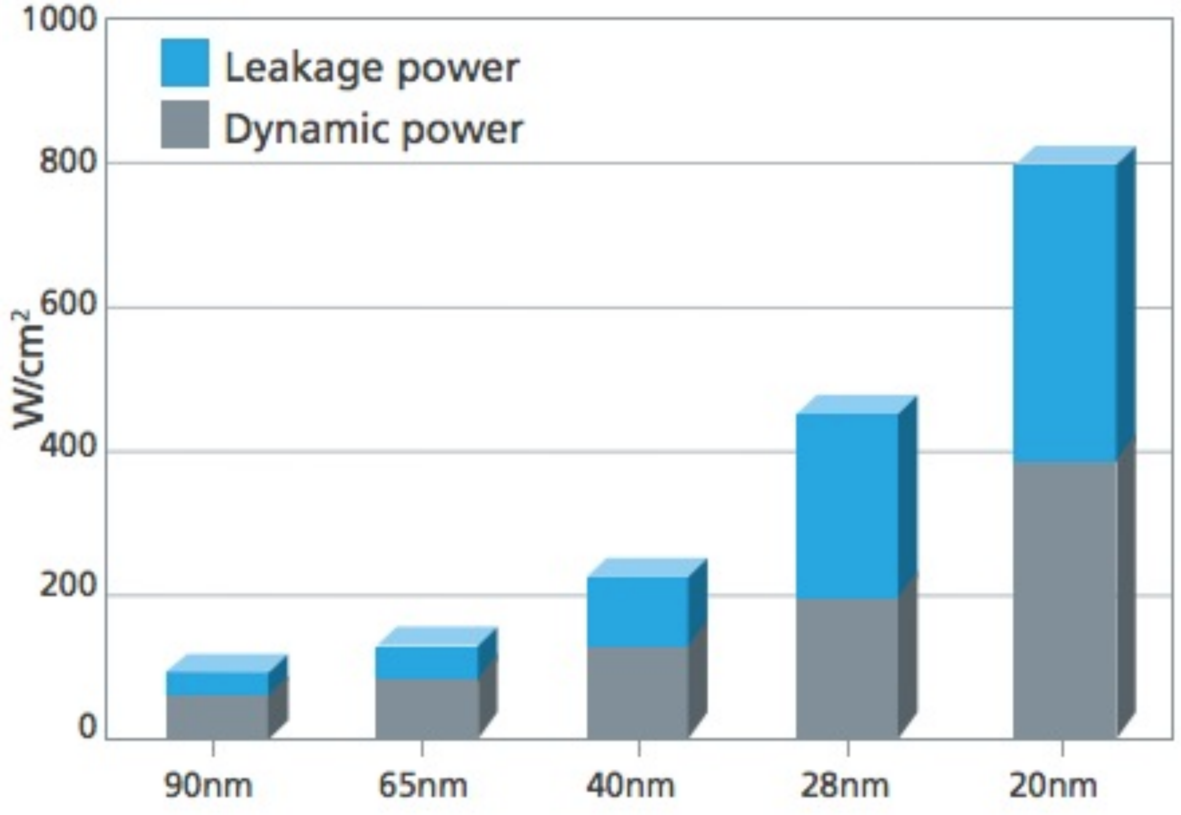


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

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!

Power & Energy

- Regarding power and energy, how many of the following statements are correct?

✗ 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 Lowering the frequency helps reducing the power consumption

⑤ A CPU operating at 20% of its top frequency can still consume 33% of the peak power

A. 0

B. 1

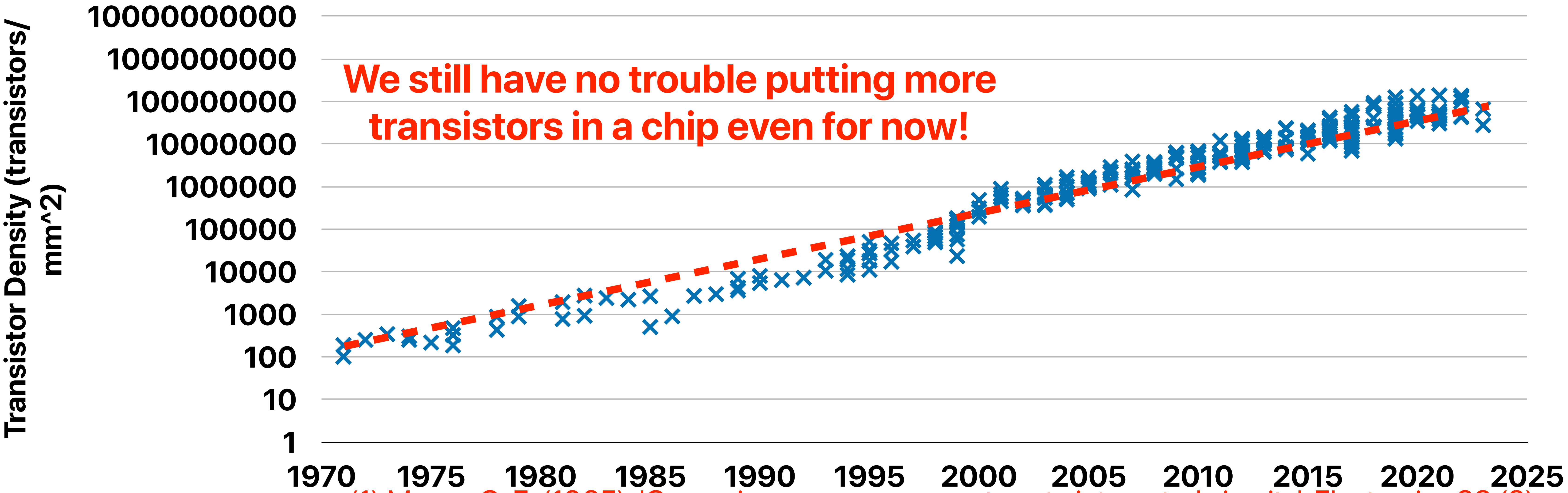
C. 2

D. 3

[✓ E. 4]

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



We still have no trouble putting more transistors in a chip even for now!

[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

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

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

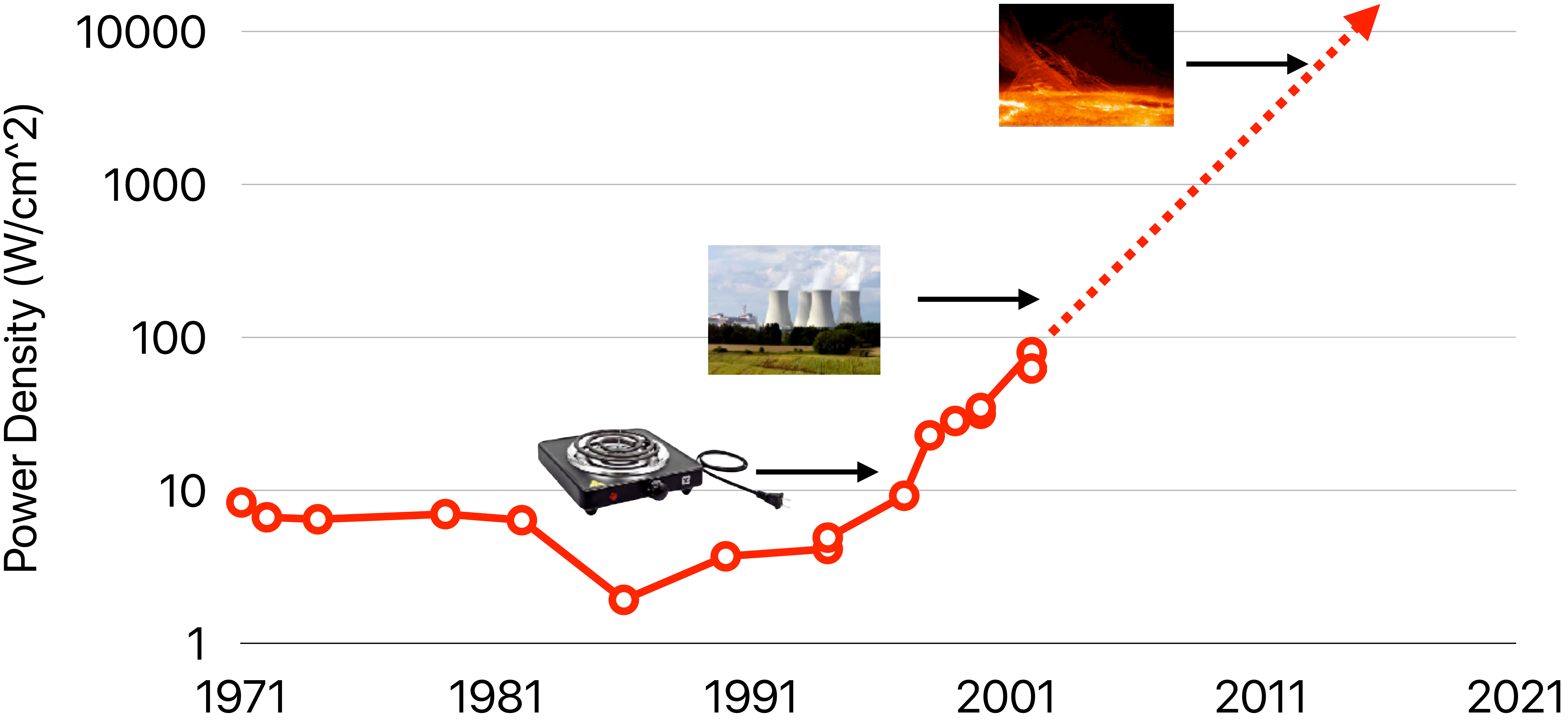
Dennardian Broken

Chip

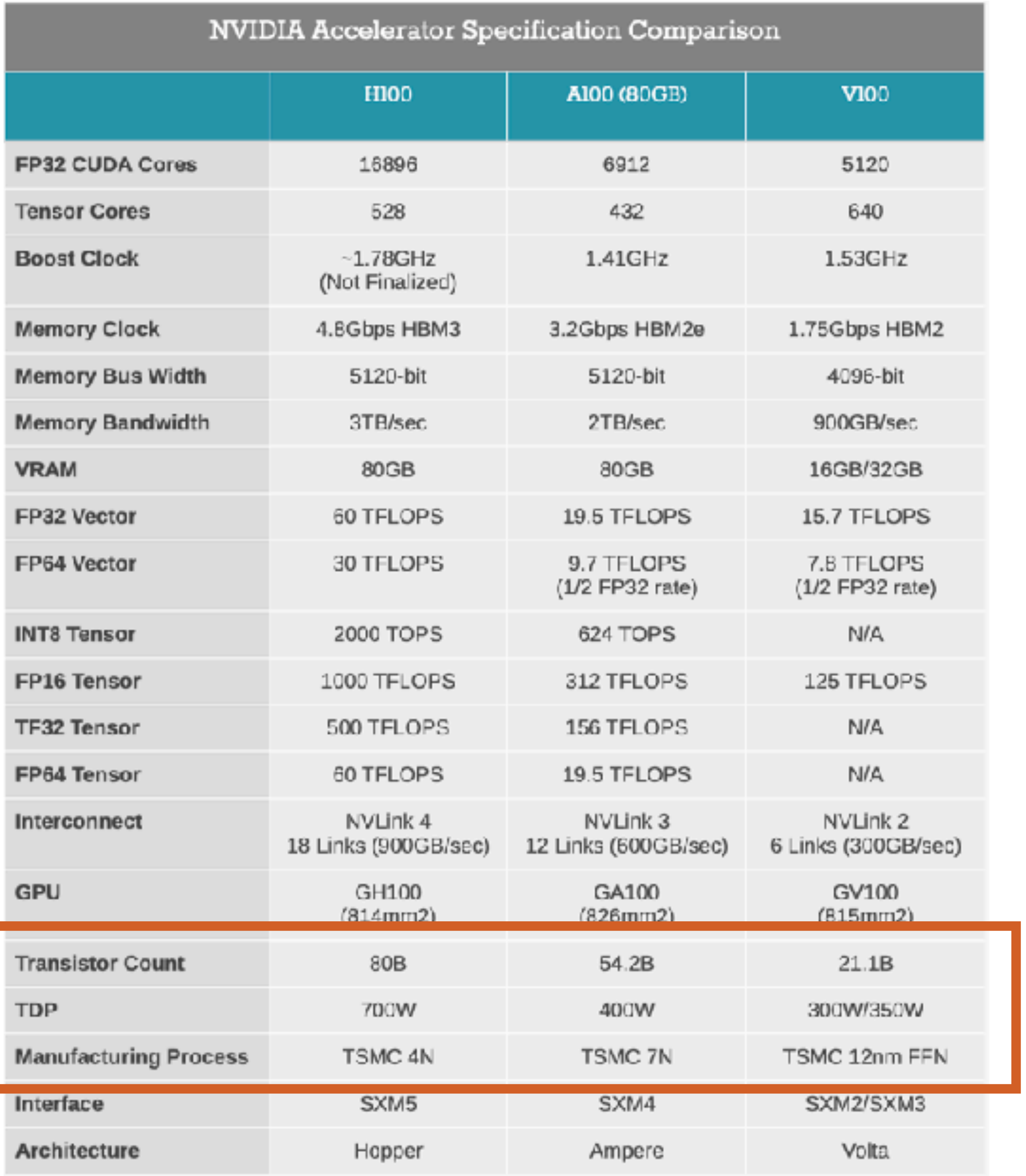
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

=100W!

Power Density of Processors



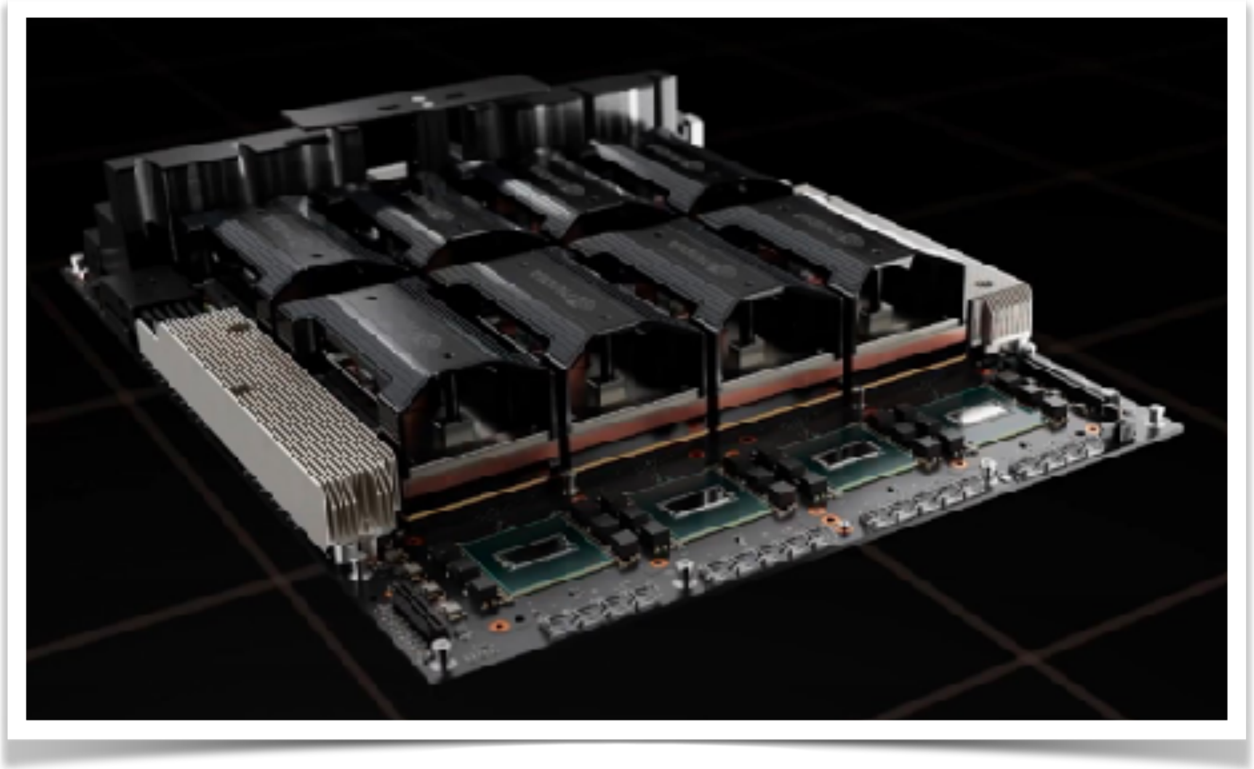
If you can add power budget...



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

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

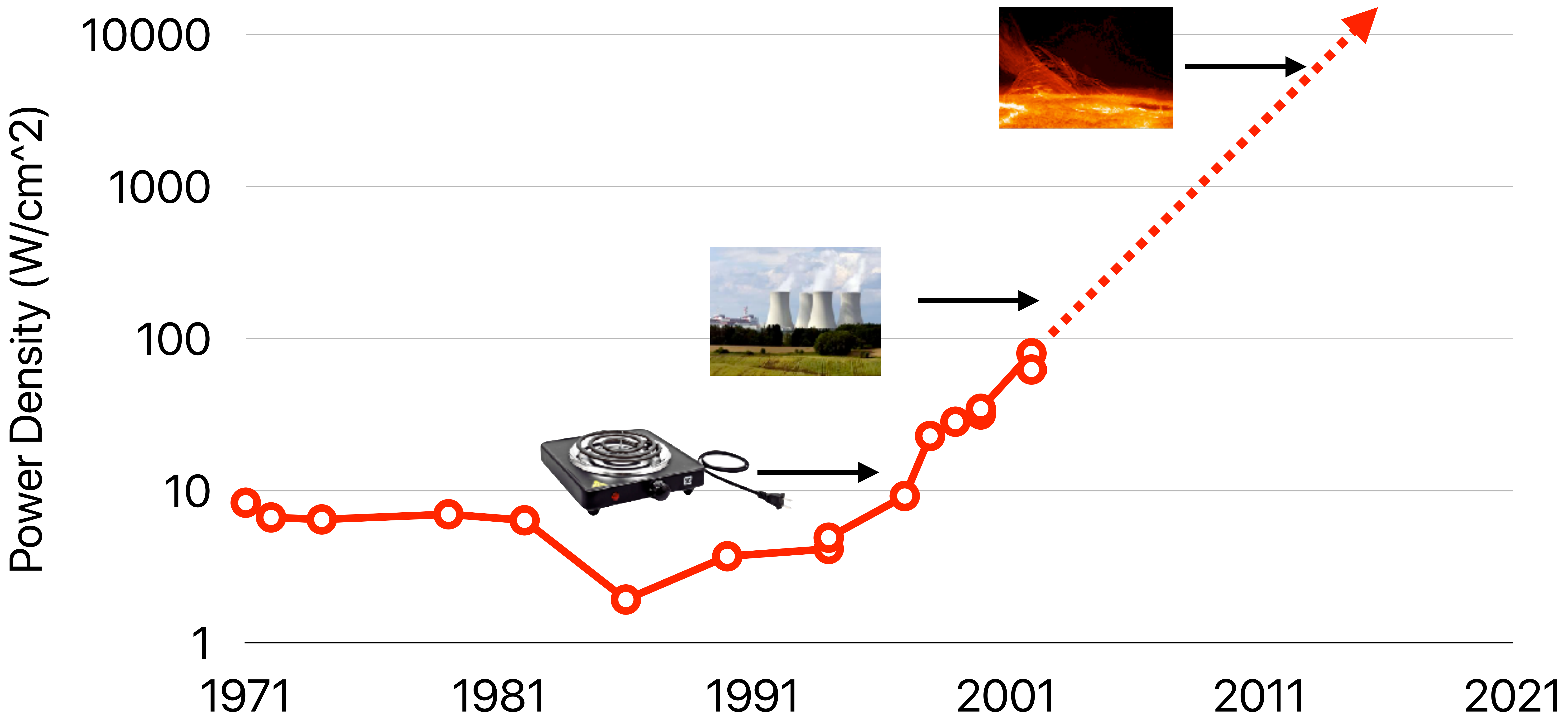
Off ~ 0W

Dark!

=100W!

Dark silicon problem

Power Density of Processors



Announcements

- **iEVAL** started and ends on 6/07/2024
  - Submit the prove of your participation in iEVAL through Gradescope
  - It can become a full credit reading quiz (it helps to amortize the penalty of another least performing one)
- **Assignment 5** is **due 6/09/2024**
  - The same programming assignment as Assignment 3 but you need to speedup by 4x on **Gradescope** this time
  - The Gradescope/AWS server gets busy and more "non-deterministic" toward the deadline
- **Final exam**
  - 6/14 8a-11a @ **BOYHL 1471**
  - Closed book, no cheatsheet — the same rules as the midterm
  - Two questions can be used as CSMS comprehensive examine questions — one is memory-hierarchy related, and the other is OoO scheduling and code optimization

Computer
Science &
Engineering

203

つづく