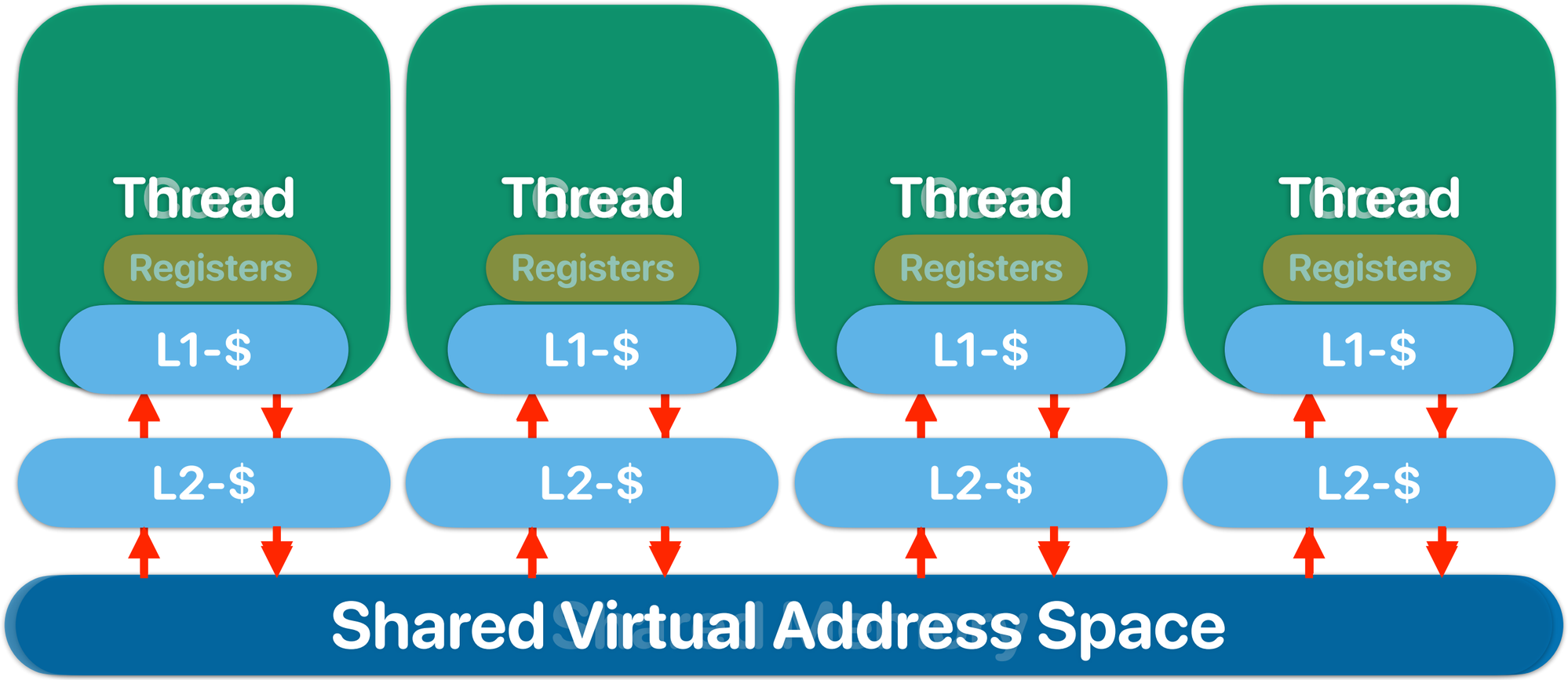
Dark Silicon & Modern Computer Architecture

Hung-Wei Tseng

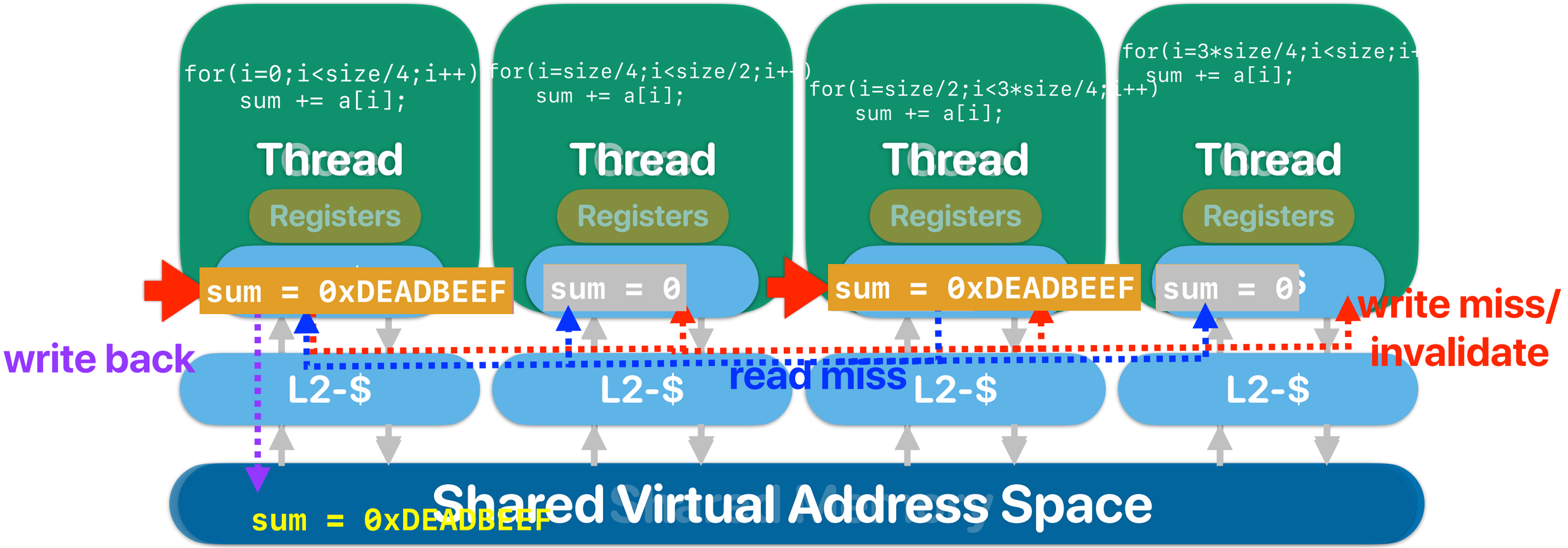
Recap: What software thinks about “multiprogramming” hardware



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

What happens when we write in coherent caches?



Observer

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

thread 1	thread 2

```
volatile int loop;

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

  pthread_create(&thread, NULL, modifyloop,
NULL);
  while(loop == 1)
  {
    continue;
  }
  pthread_join(thread, NULL);
  fprintf(stderr,"User input: %d\n", loop);
  return 0;
}
```

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

Cache coherency

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

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

① 0 1 2 3 4 5 6 7 8 9

② 1 2 5 9 3 6 8 10 12 13

③ 1 1 1 1 1 1 1 64 100

④ 1 1 1 1 1 1 1 1 1 100

A. 0

B. 1

C. 2

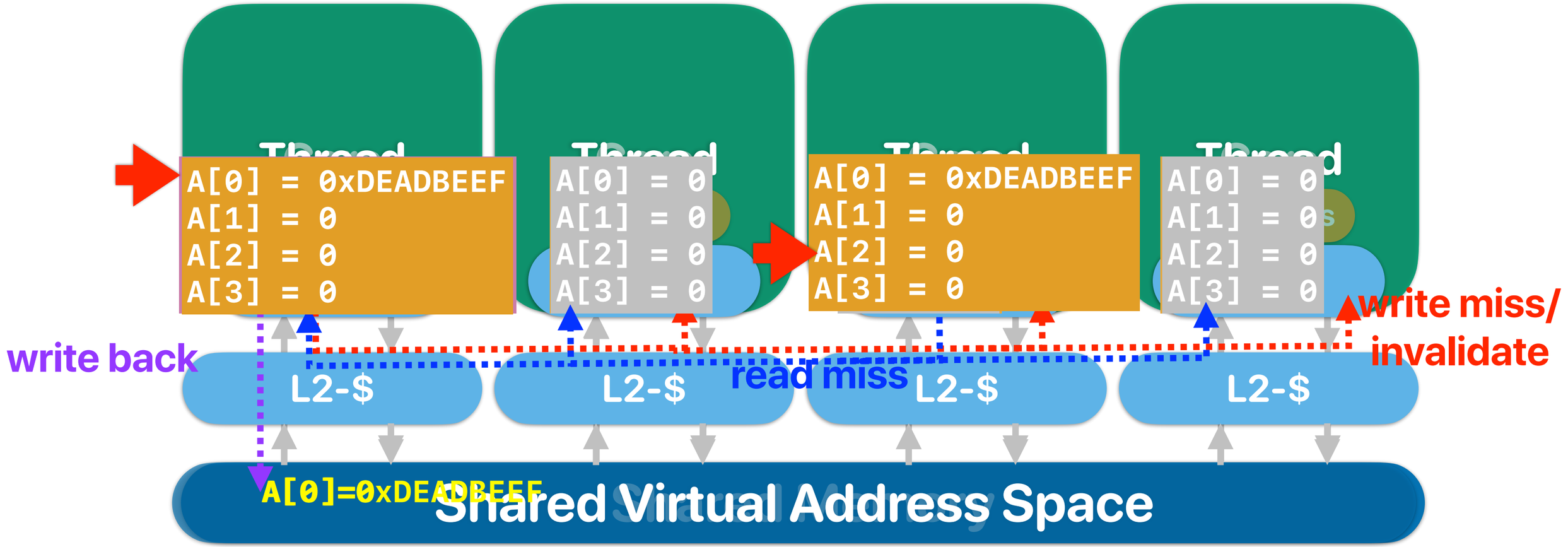
D. 3

E. 4

Outline

- Parallel programming
- Dark Silicon and its impact on computer architecture

Cache coherency



Performance comparison

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

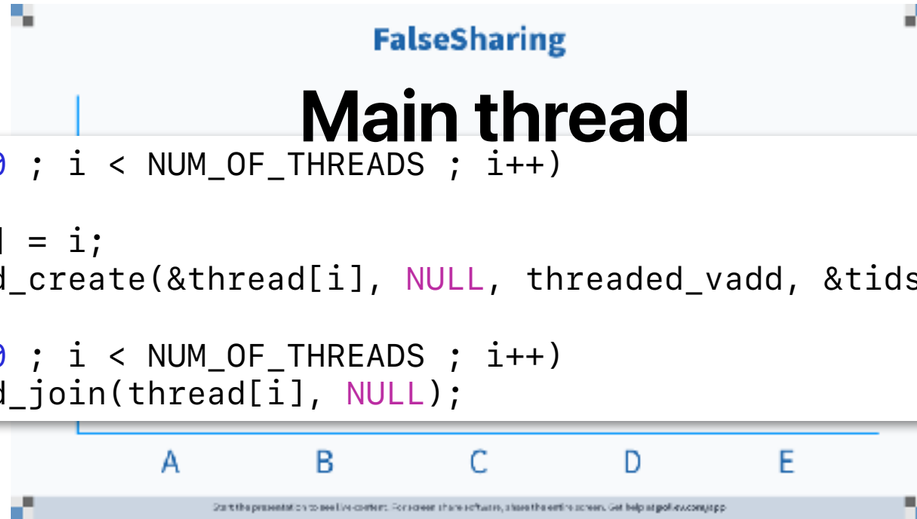
Version L

```
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



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;
}
```

C 

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;
}
```

C 

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);
```

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?

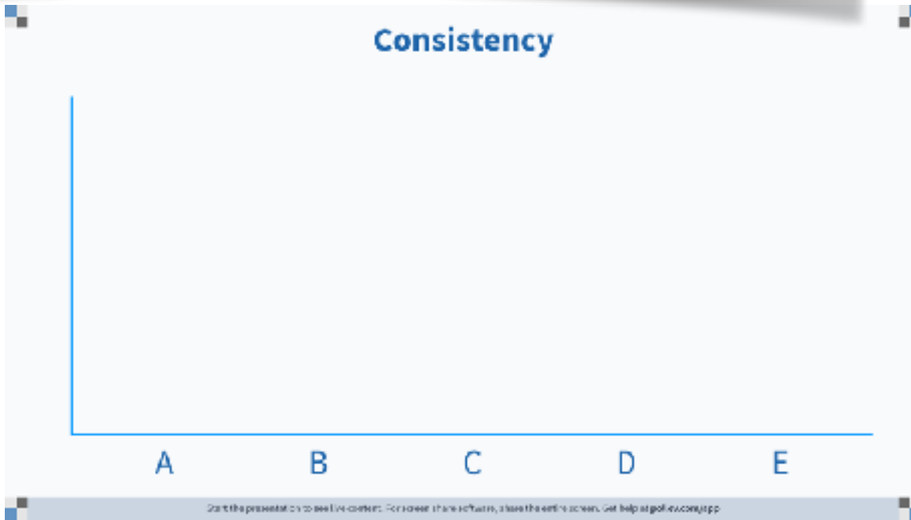
① (0, 0)
② (0, 1)
③ (1, 0)
④ (1, 1)

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

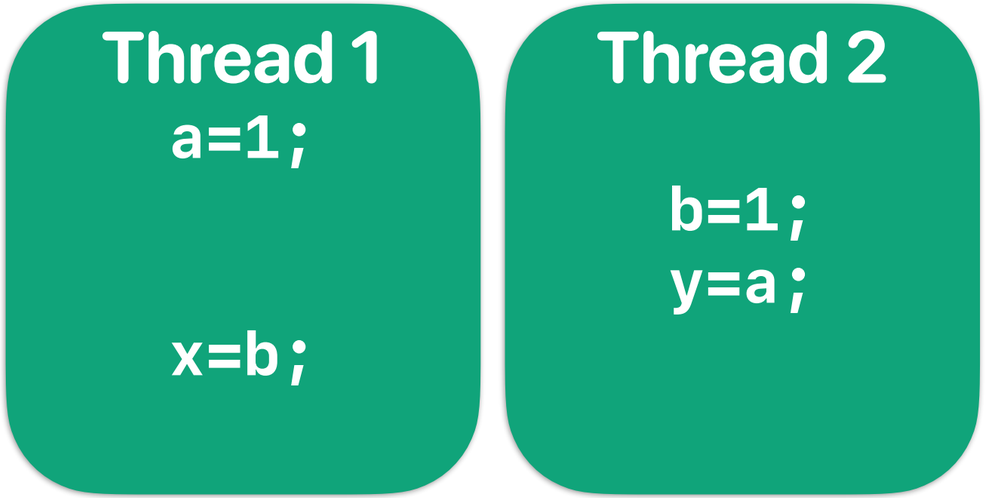
```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

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;
}
```

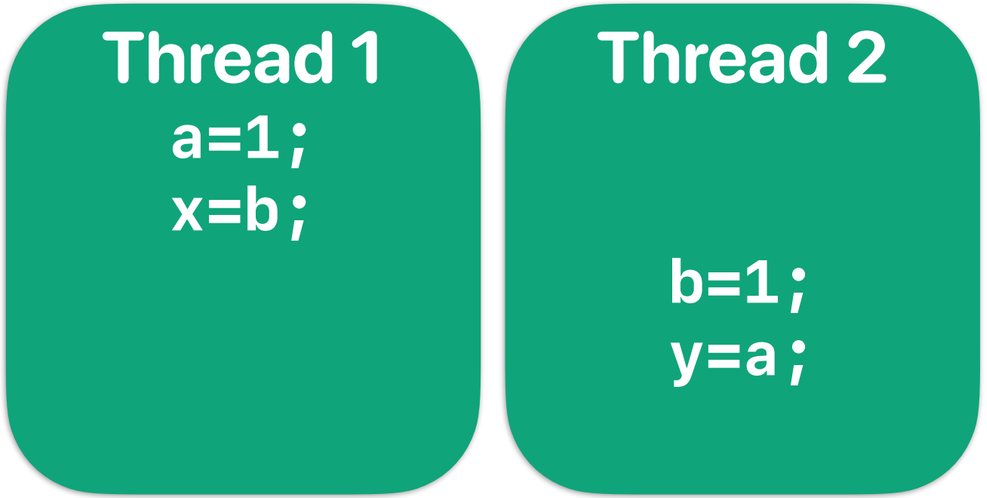
```c
int main() {
  int i;
  pthread_t thread[2];
  pthread_create(&thread[0], NULL, modifya, NULL);
  pthread_create(&thread[1], NULL, modifyb, NULL);
  pthread_join(thread[0], NULL);
  pthread_join(thread[1], NULL);
  fprintf(stderr,"(%d, %d)\n",x,y);
  return 0;
}
```



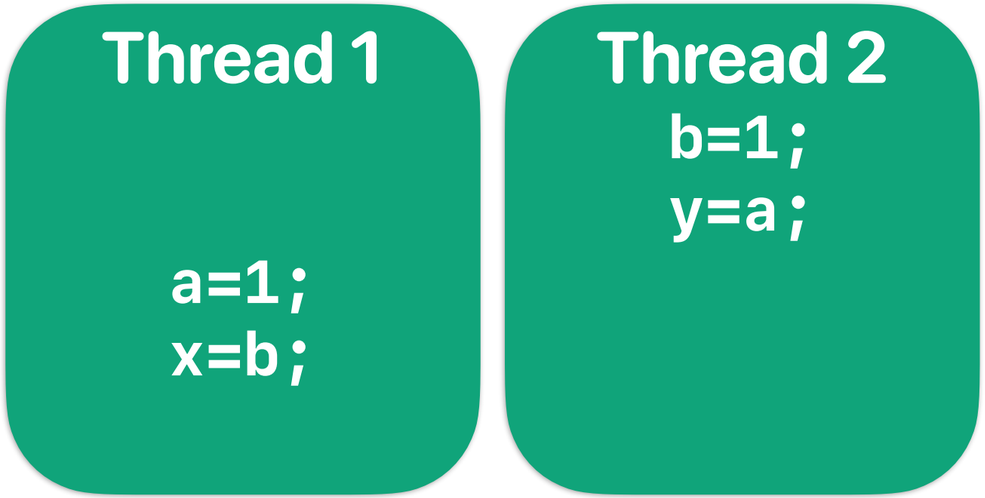
Possible scenarios



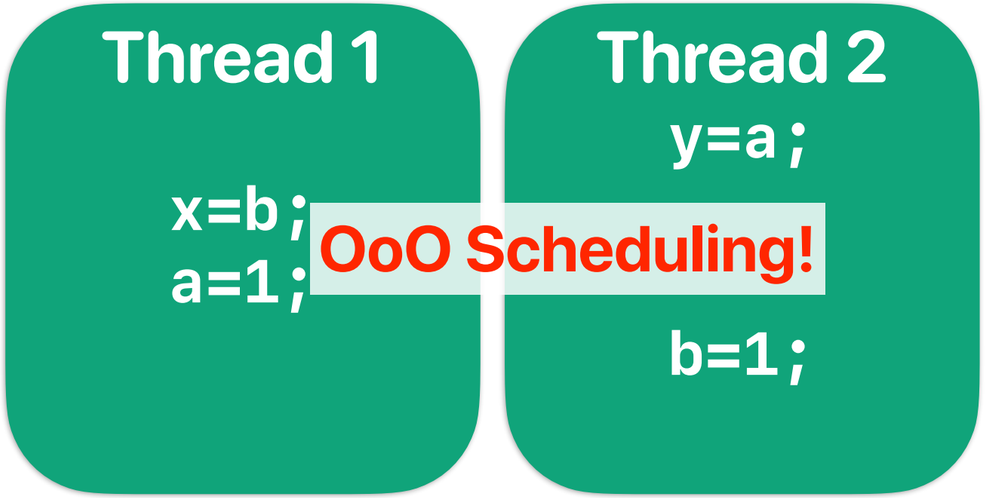
(1,1)



(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;`

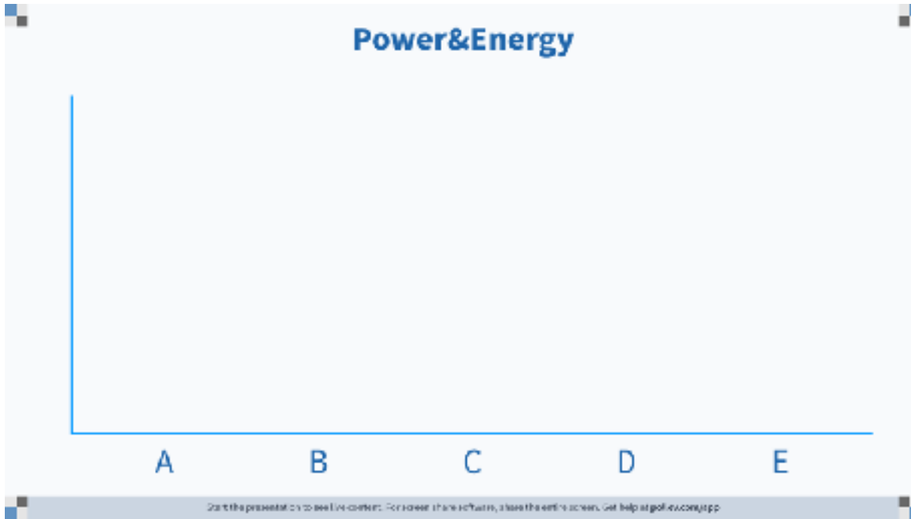
Take-aways of parallel programming

- 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

Power and Energy

Power & Energy

- Regarding power and energy, how many of the following statements are correct?
  - ① Lowering the power consumption helps reducing the heat generation
  - ② Lowering the energy consumption helps reducing the electricity bill
  - ③ Lowering the power consumption helps extending the battery life
  - 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 v.s. Energy

- Power is the direct contributor of “heat”
  - Packaging of the chip
  - Heat dissipation cost
  - Power = $P_{Dynamic} + P_{static}$
- 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 Power

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

Double Clock Rate or Double the # of Processors?

- Assume 60% of the application can be fully parallelized with 2-core or speedup linearly with clock rate. Should we **double the clock rate** or **duplicate a core**?

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

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

$$Speedup_{parallel}(60\,\%, 2) = \frac{1}{(1 - 60\%) + \frac{60\,\%}{2}} = 1.43$$

$$Power_{2-core} = 2 \times P_{baseline}$$

$$Energy_{2-core} = 2 \times P_{baseline} \times ET_{baseline} \times \frac{1}{1.43} = 1.39 \times Energy_{baseline}$$

$$Speedup_{2\times clock} = 2$$

$$Power_{2\times clock} = 2^3 \times P_{baseline} = 8 \times P_{baseline}$$

$$Energy_{2\times clock} = 2^3 \times P_{baseline} \times ET_{baseline} \times \frac{1}{2} = 4 \times P_{baseline} \times ET_{baseline}$$

Dynamic voltage/frequency scaling

- Dynamically lower power for performance
  - Change the voltage and frequency at runtime
  - Under control of operating system — that's why updating iOS may slow down an old iPhone
- Recall: $P_{dynamic} \sim \alpha \times C \times V^2 \times f \times N$
  - Because frequency ~ to V…
  - $P_{dynamic}$ ~ to V^3
- Reduce both V and f linearly
  - Cubic decrease in dynamic power
  - Linear decrease in performance (actually sub-linear)
    - Thus, only about quadratic in energy
  - Linear decrease in static power
    - Thus, only modest static energy improvement
  - Newer chips can do this on a per-core basis
    - cat /proc/cpuinfo in linux

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!

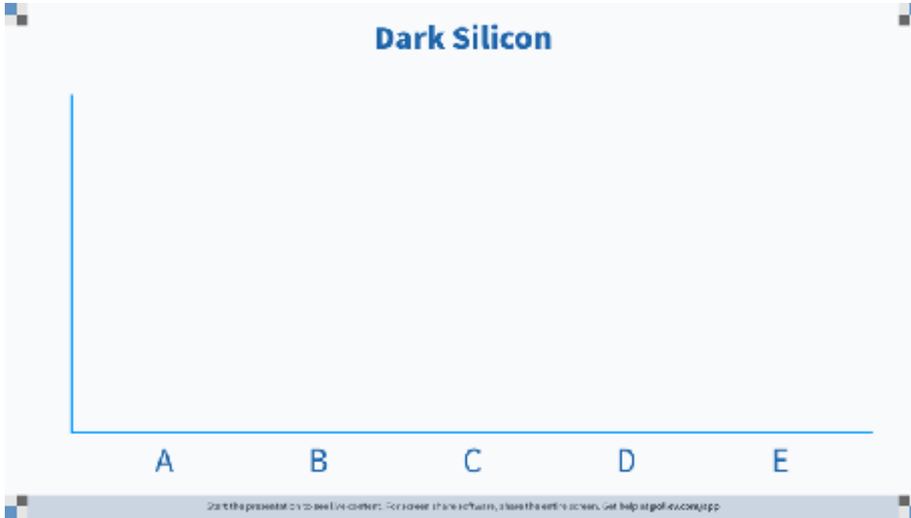
Power & Energy

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



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

Dark Silicon and the End of Multicore Scaling

H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam and D. Burger
University of Washington, University of Wisconsin—Madison, University of Texas at Austin, Microsoft Research

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)

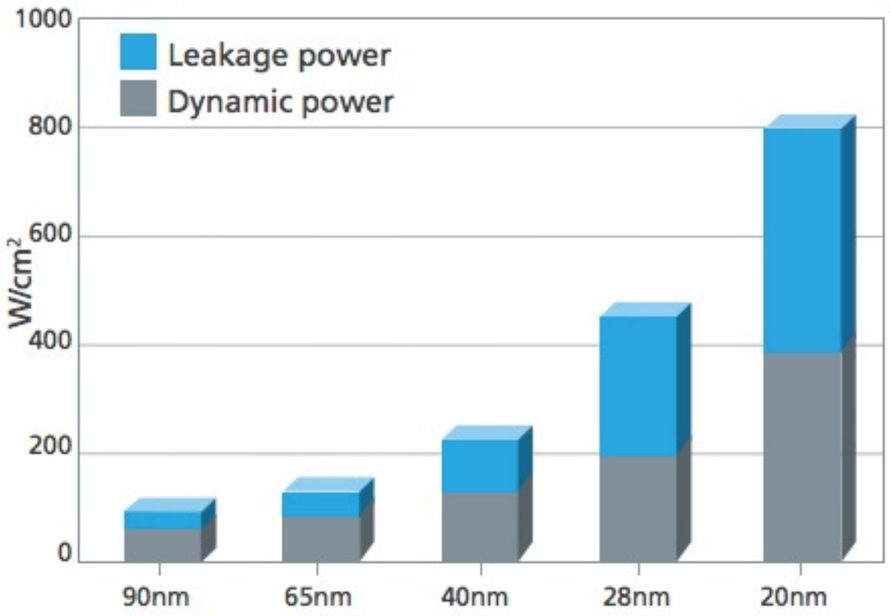


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

Dennardian Broken

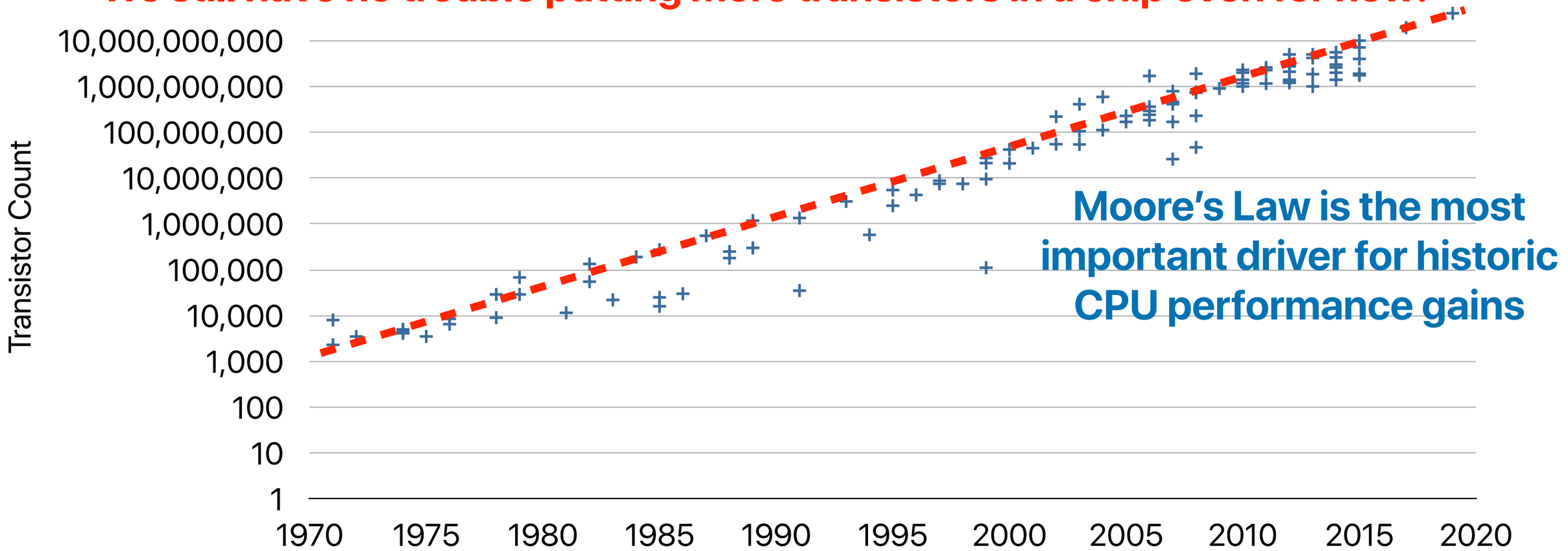
- Given a scaling factor S

Parameter	Relation	Classical Scaling	Leakage Limited
Power Budget		1	1
Chip Size		1	1
Vdd (Supply Voltage)		1/S	1
Vt (Threshold Voltage)	1/S	1/S	1
tex (oxide thickness)		1/S	1/S
W, L (transistor dimensions)		1/S	1/S
Cgate (gate capacitance)	WL/tox	1/S	1/S
Isat (saturation current)	WVdd/tox	1/S	1
F (device frequency)	Isat/(CgateVdd)	S	S
D (Device/Area)	1/(WL)	S^2	S^2
p (device power)	IsatVdd	$1/S^2$	1
P (chip power)	Dp	1	S^2
U (utilization)	1/P	1	$1/S^2$

Moore's Law[1]

- The number of transistors we can build in a fixed area of silicon doubles every 12 ~ 24 months.

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



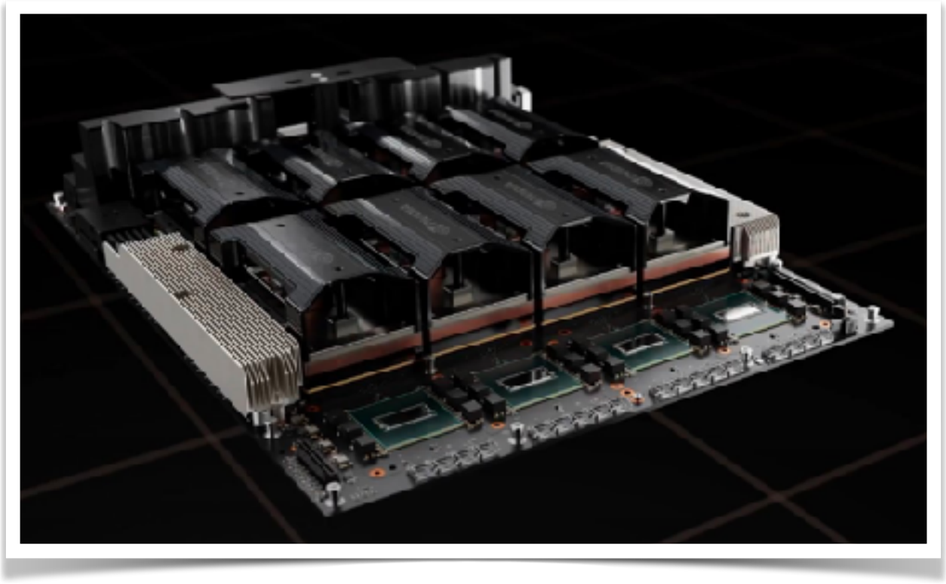
(1) Moore, G. E. (1965), 'Cramming more components onto integrated circuits', Electronics 38 (8) .

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

If you can add power budget…

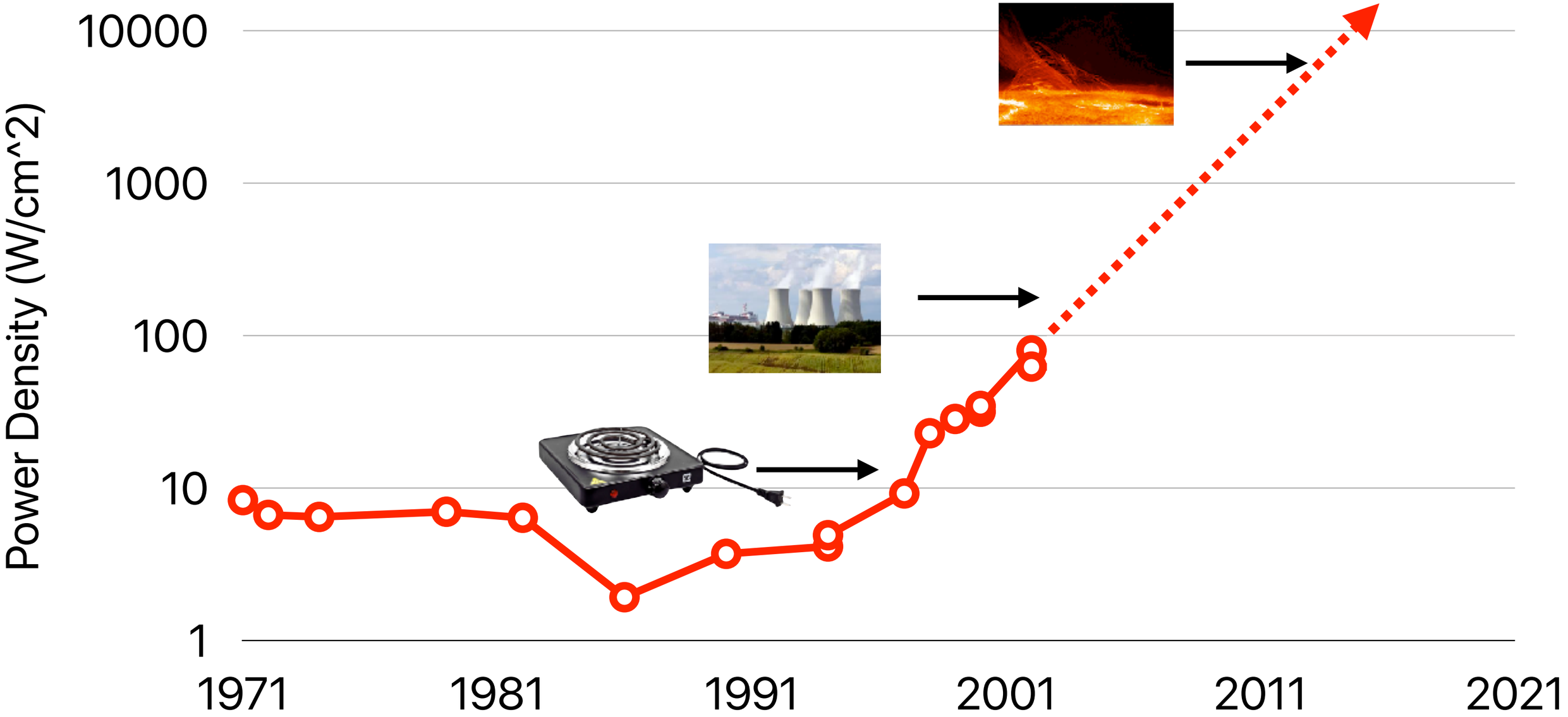
NVIDIA Accelerator Specification Comparison			
	H100	A100 (80GB)	V100
FP32 CUDA Cores	16896	6912	5120
Tensor Cores	528	432	640
Boost Clock	~1.78GHz (Not Finalized)	1.41GHz	1.53GHz
Memory Clock	4.8Gbps HBM3	3.2Gbps HBM2e	1.75Gbps HBM2
Memory Bus Width	5120-bit	5120-bit	4096-bit
Memory Bandwidth	3TB/sec	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 Density of Processors



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

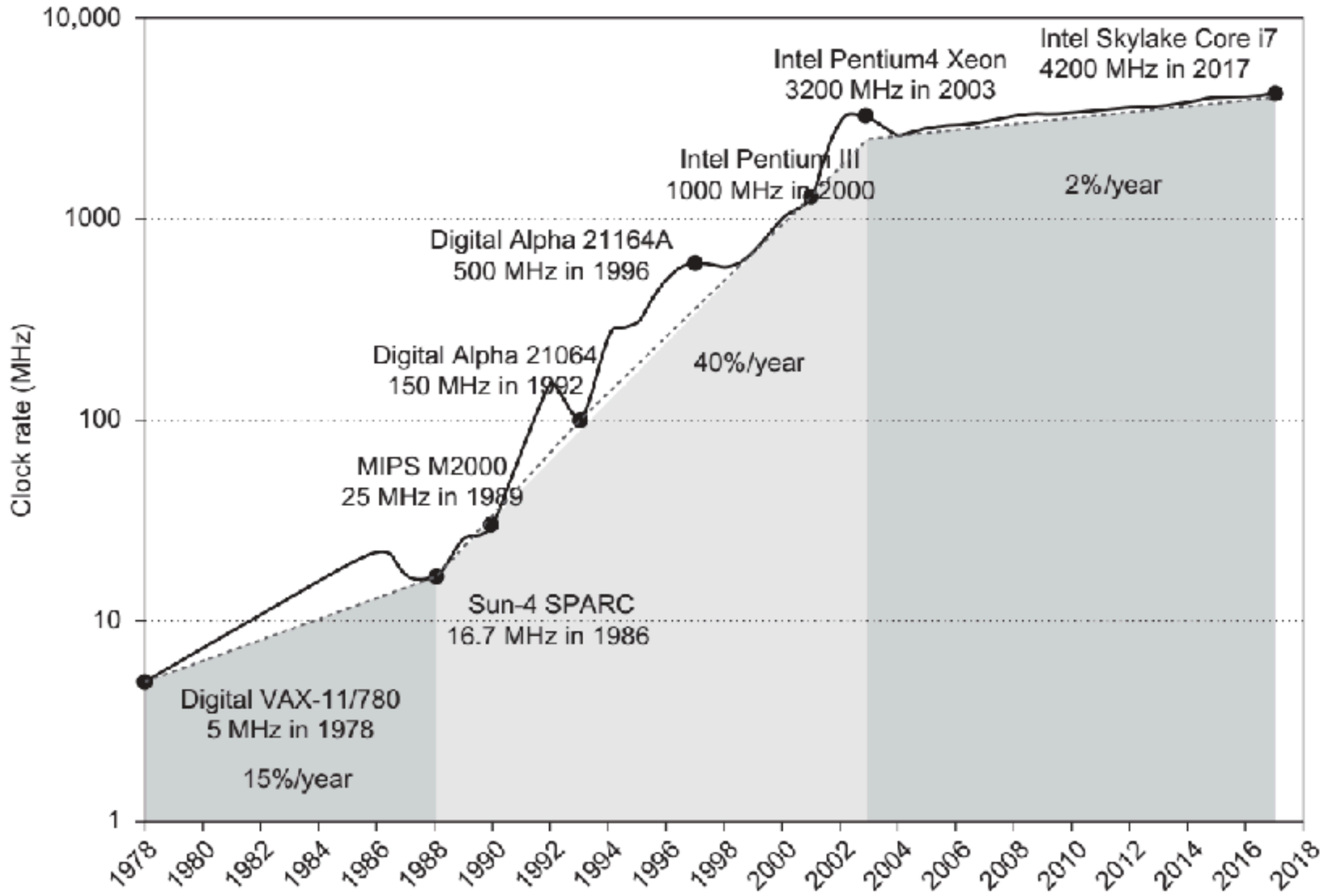
On ~ 50W

Off ~ 0W

Dark!

=100W!

Clock rate improvement is limited nowadays



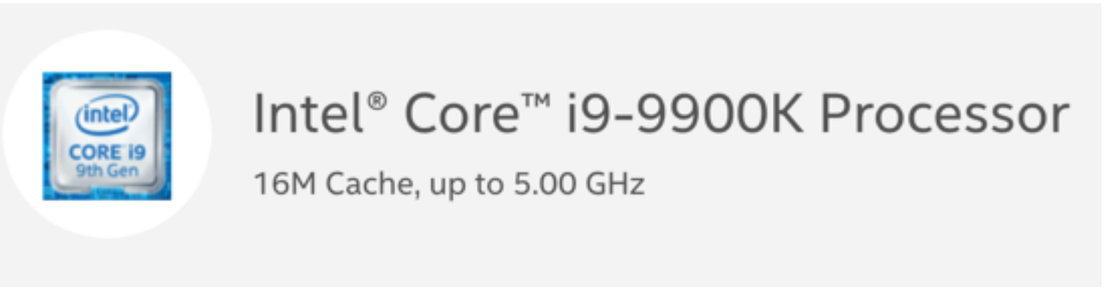
Solutions/trends in dark silicon era

Trends in the Dark Silicon Era

- Aggressive dynamic voltage/frequency scaling
- Throughout oriented — slower, but more
- Just let it dark — activate part of circuits, but not all
- From general-purpose to domain-specific — ASIC

Aggressive dynamic frequency scaling

Modern processor's frequency



Intel® Core™ i9-9900K Processor

16M Cache, up to 5.00 GHz

Essentials

Product Collection	
Code Name	
Vertical Segment	
Processor Number	
Status	
Launch Date	
Lithography	
Included Items	
Use Conditions	
Recommended Customer Price	

CPU Specifications

# of Cores	8
# of Threads	16
Processor Base Frequency	3.60 GHz
Max Turbo Frequency	5.00 GHz
Cache	16 MB Intel® Smart Cache
Bus Speed	8 GT/s
Intel® Turbo Boost Technology 2.0 Frequency‡	5.00 GHz
TDP	95 W

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

Recap: 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!

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.

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

How about static power?

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

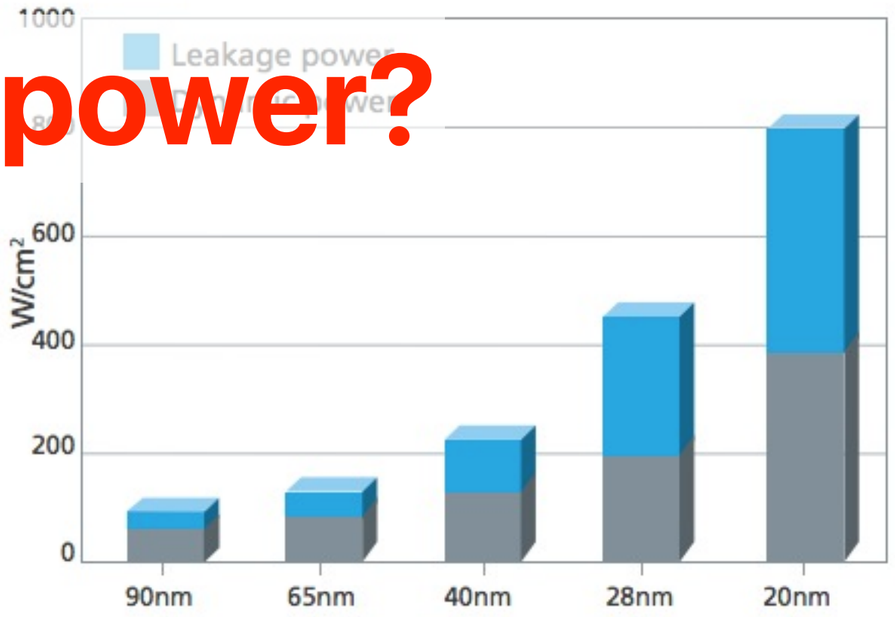


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

Slower, but more

Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction

Rakesh Kumar, Keith I. Farkas*, Norman P. Jouppi*, Parthasarathy Ranganathan*, Dean M. Tullsen

UCSD and HP Labs*

In the 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36., 2003
MICRO Test-of-time award, 2021

Announcement

- Assignment #4 due this Friday
- iEVAL, until 12/2
  - Please fill the survey to let us know your opinion!
  - Don't forget to take a screenshot of your submission and submit through iLearn — it counts as a **full credit notebook assignment**
  - **We will drop your lowest 2 notebook assignment grades**
  - **Still only one programming assignment will be dropped**
- Final Exam
  - Starting from **12/5** 12:00am to 11:59pm, any consecutive 180 minutes you pick
  - Similar to the midterm, but more time and about 1.5x longer
  - Two of the problem sets will be comprehensive exam questions
  - Will release a sample final at the end of the last lecture

Computer
Science &
Engineering

203

つづく