Programming on Parallel Architectures

Hung-Wei Tseng

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

Demo: ILP within a program

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

Wider-issue processors won't give you much more

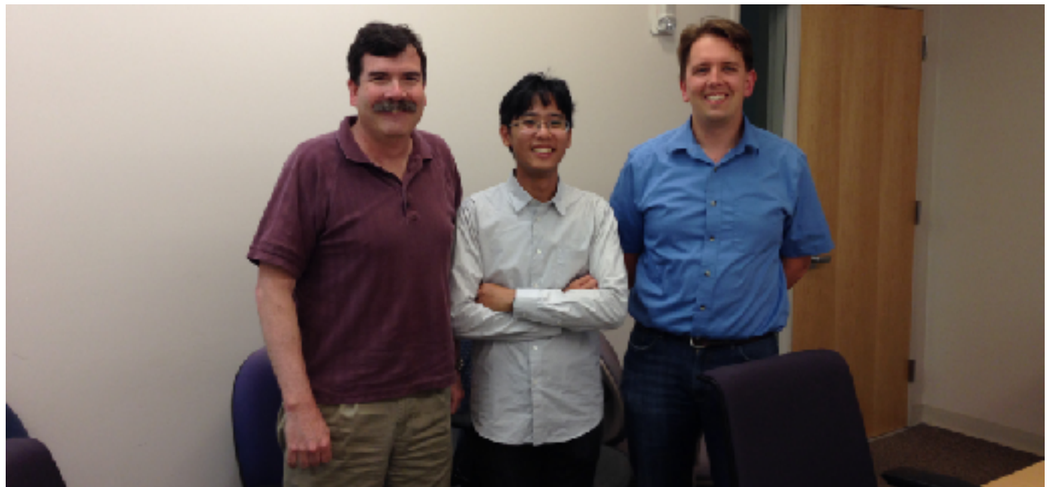
Program	IPC	BP Rate %	I cache %MPCI	D cache %MPCI	L2 cache %MPCI
compress	0.9	85.9	0.0	3.5	1.0
eqntott	1.3	79.8	0.0	0.8	0.7
m88ksim	1.4	91.7	2.2	0.4	0.0
MPsim	0.8	78.7	5.1	2.3	2.3
applu	0.9	79.2	0.0	2.0	1.7
apsi	0.6	95.1	1.0	4.1	2.1
swim	0.9	99.7	0.0	1.2	1.2
tomcatv	0.8	99.6	0.0	7.7	2.2
pmake	1.0	86.2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

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

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

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

Simultaneous multithreading

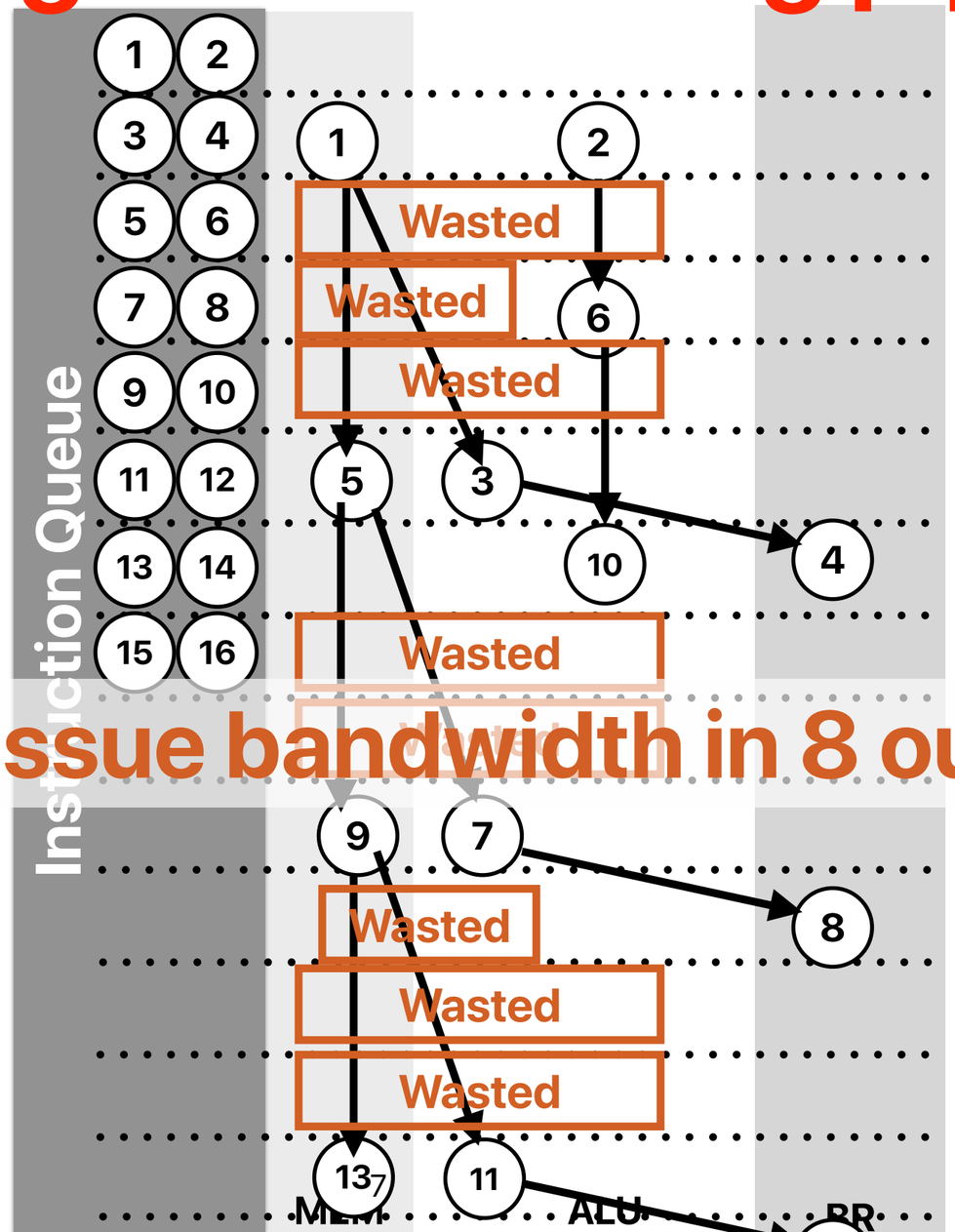


Simultaneous multithreading

- 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

2-issue register renaming pipeline

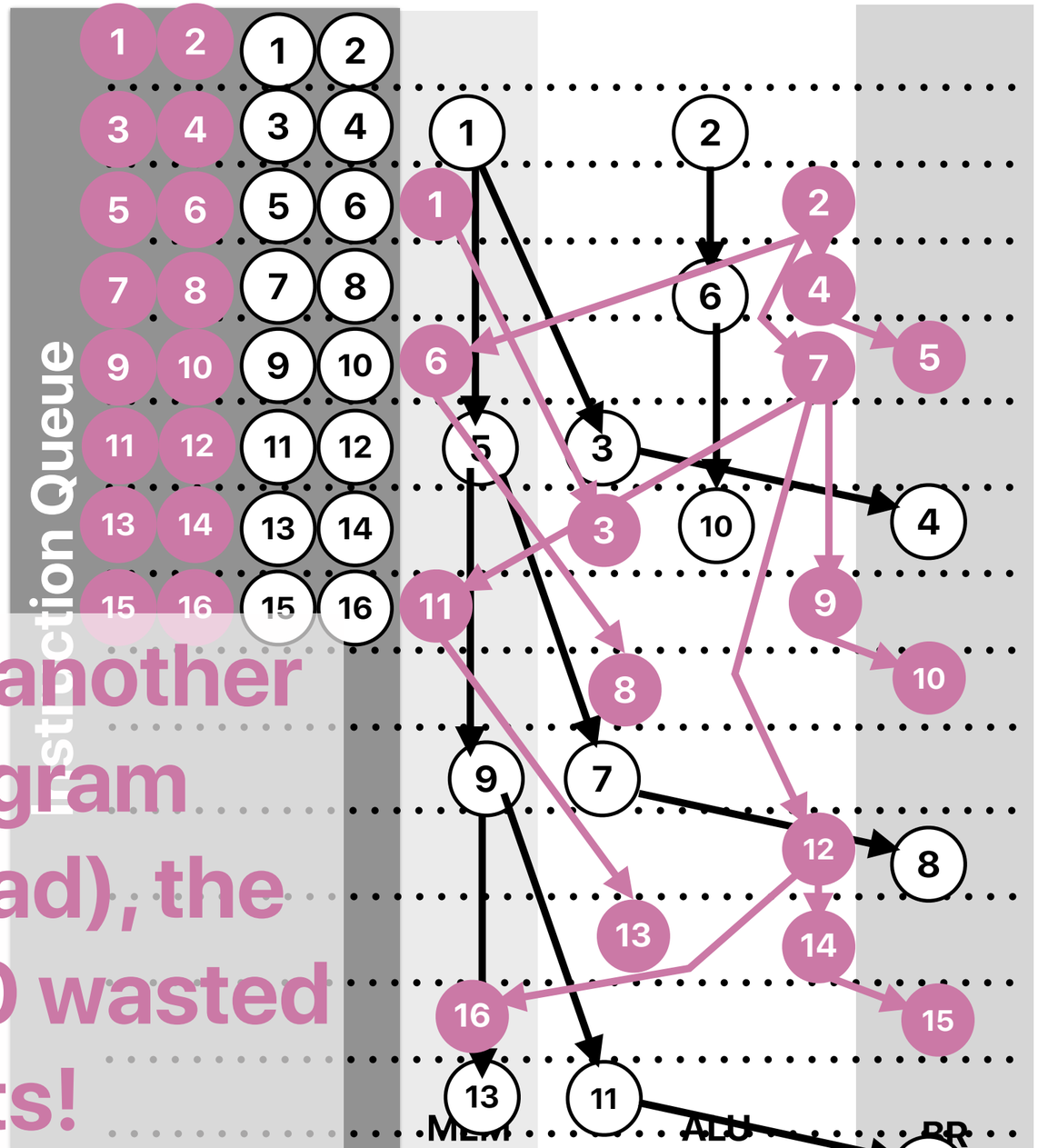
```
① movq    8(%rdi), %rdi
② addl    $1, %eax
③ testq   %rdi, %rdi
④ jne     .L3
⑤ movq    8(%rdi), %rdi
⑥ addl    $1, %eax
⑦ testq   %rdi, %rdi
⑧ jne     .L3
⑨ movq    8(%rdi), %rdi
⑩ addl    $1, %eax
⑪ testq   %rdi, %rdi
⑫ jne     .L3
```



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

Concept: Simultaneous Multithreading

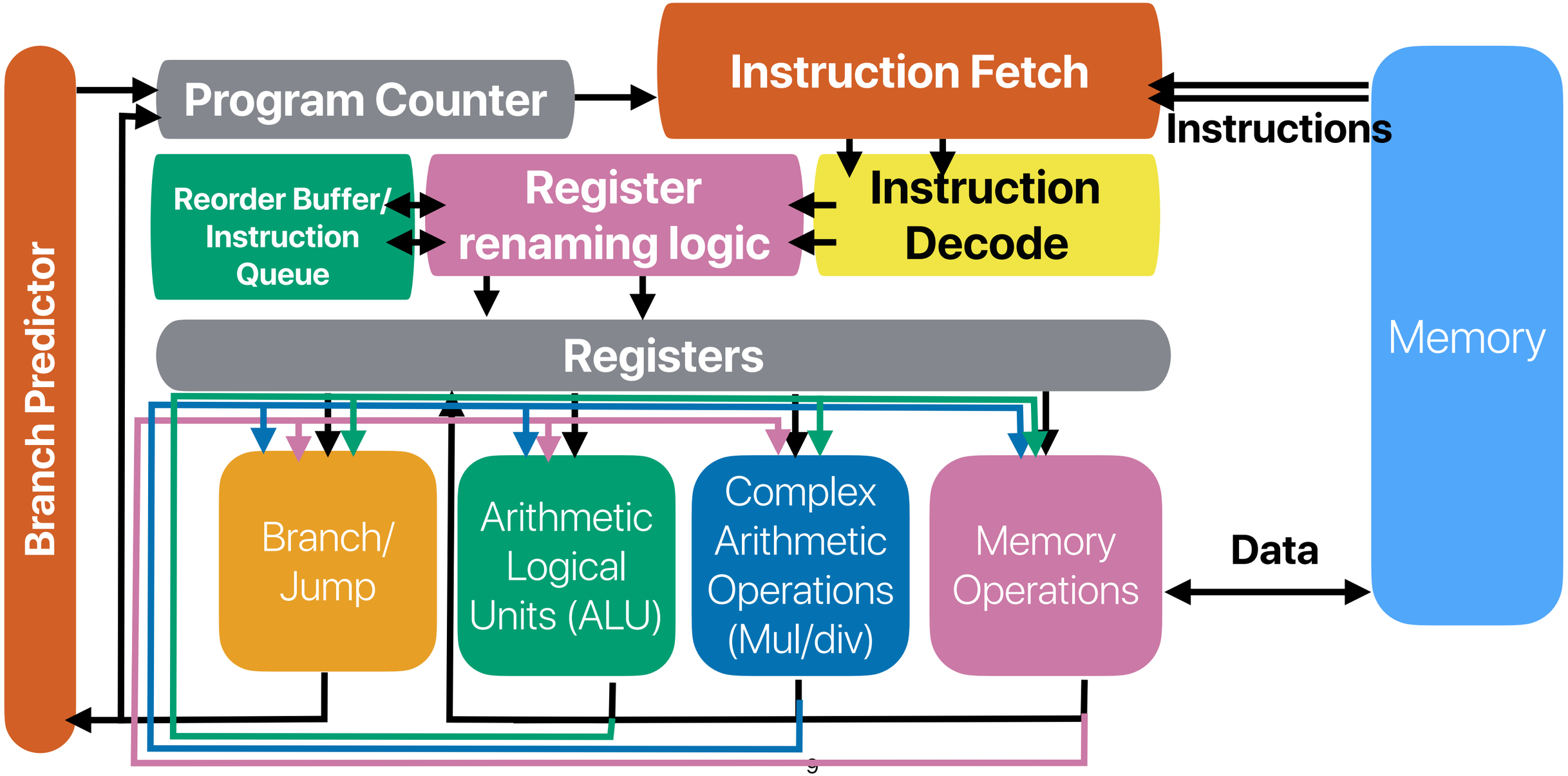
```
① movq    8(%rdi), %rdi
② addl    $1, %eax
③ testq   %rdi, %rdi
④ jne     .L3
⑤ movq    8(%rdi), %rdi
⑥ addl    $1, %eax
⑦ testq   %rdi, %rdi
⑧ jne     .L3
⑨ movq    8(%rdi), %rdi
⑩ addl    $1, %eax
⑪ testq   %rdi, %rdi
⑫ jne     .L3
```



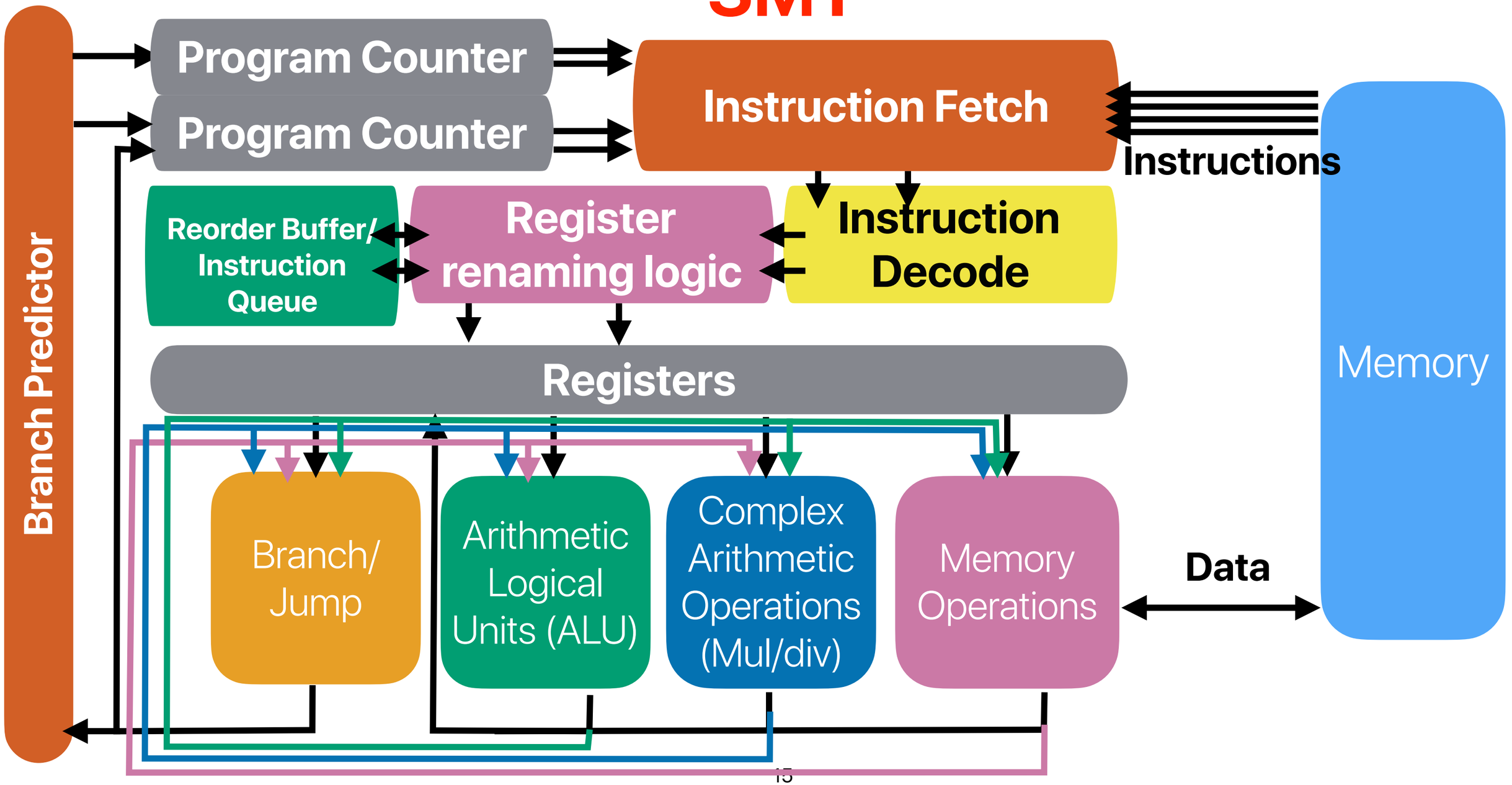

```
① movl (%rdi), %ecx
② addq $4, %rdi
③ addl %ecx, %eax
④ cmpq %rdx, %rdi
⑤ jne  .L3
⑥ movl (%rdi), %ecx
⑦ addq $4, %rdi
⑧ addl %ecx, %eax
⑨ cmpq %rdx, %rdi
⑩ jne  .L3
⑪ movl (%rdi), %ecx
⑫ addq $4, %rdi
⑬ addl %ecx, %eax
⑭ cmpq %rdx, %rdi
⑮ jne  .L3
⑯ movl (%rdi), %ecx
```

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

Register renaming



SMT


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

SMT

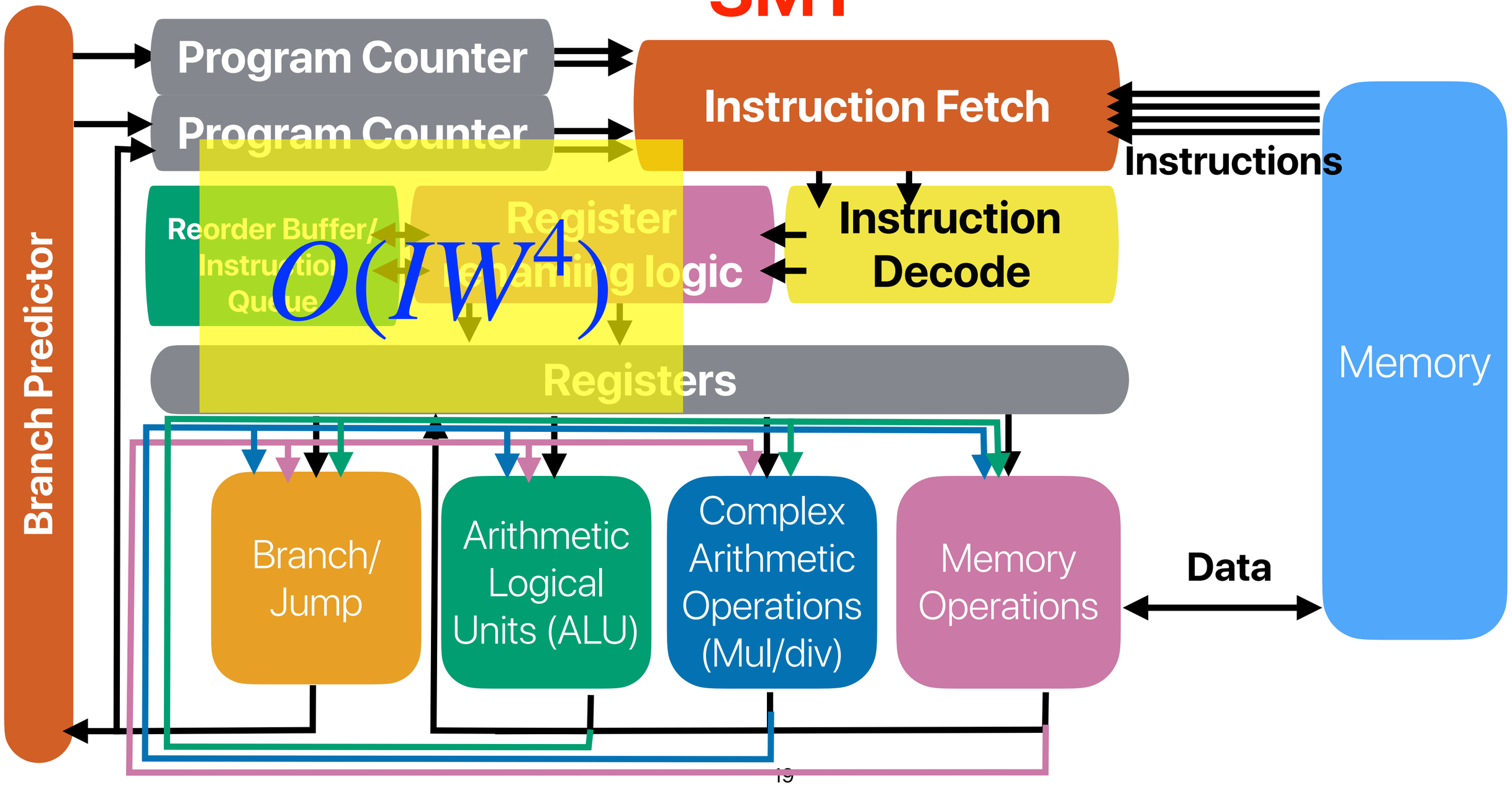
- Improve the throughput of execution
  - May increase the latency of a single thread
- Less branch penalty per thread
- Increase hardware utilization
- Simple hardware design: Only need to duplicate PC/Register Files
- Real Case:
  - Intel HyperThreading (supports up to two threads per core)
    - Intel 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:                        48 bits physical, 48 bits virtual
CPU(s):                               16
On-line CPU(s) list:                  0-15
Thread(s) per core:                   2
Core(s) per socket:                   8
Socket(s):                            1
NUMA node(s):                         1
Vendor ID:                            AuthenticAMD
CPU family:                           25
Model:                                80
Model name:                           AMD Ryzen 7 5700G with Radeon Graphics
Stepping:                             0
```

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!

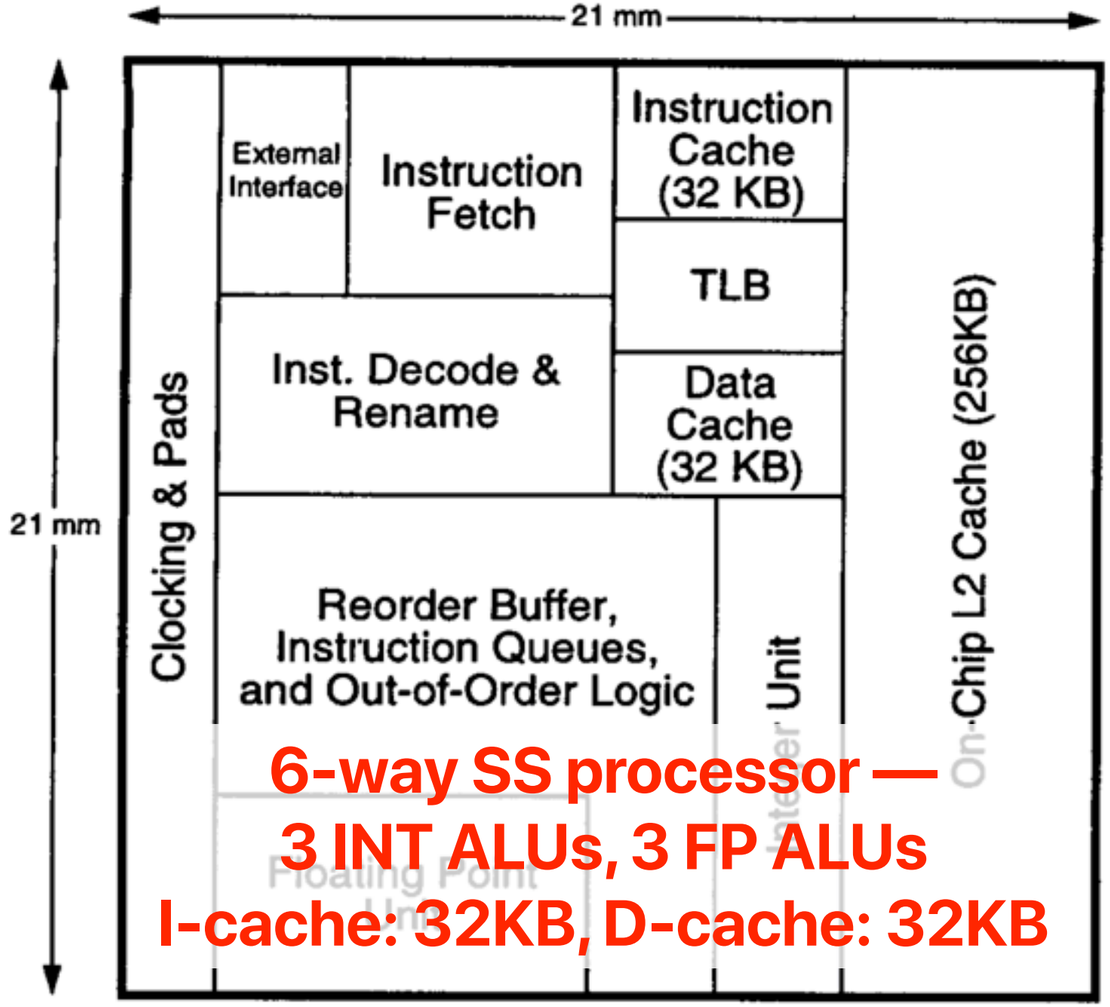
SMT


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

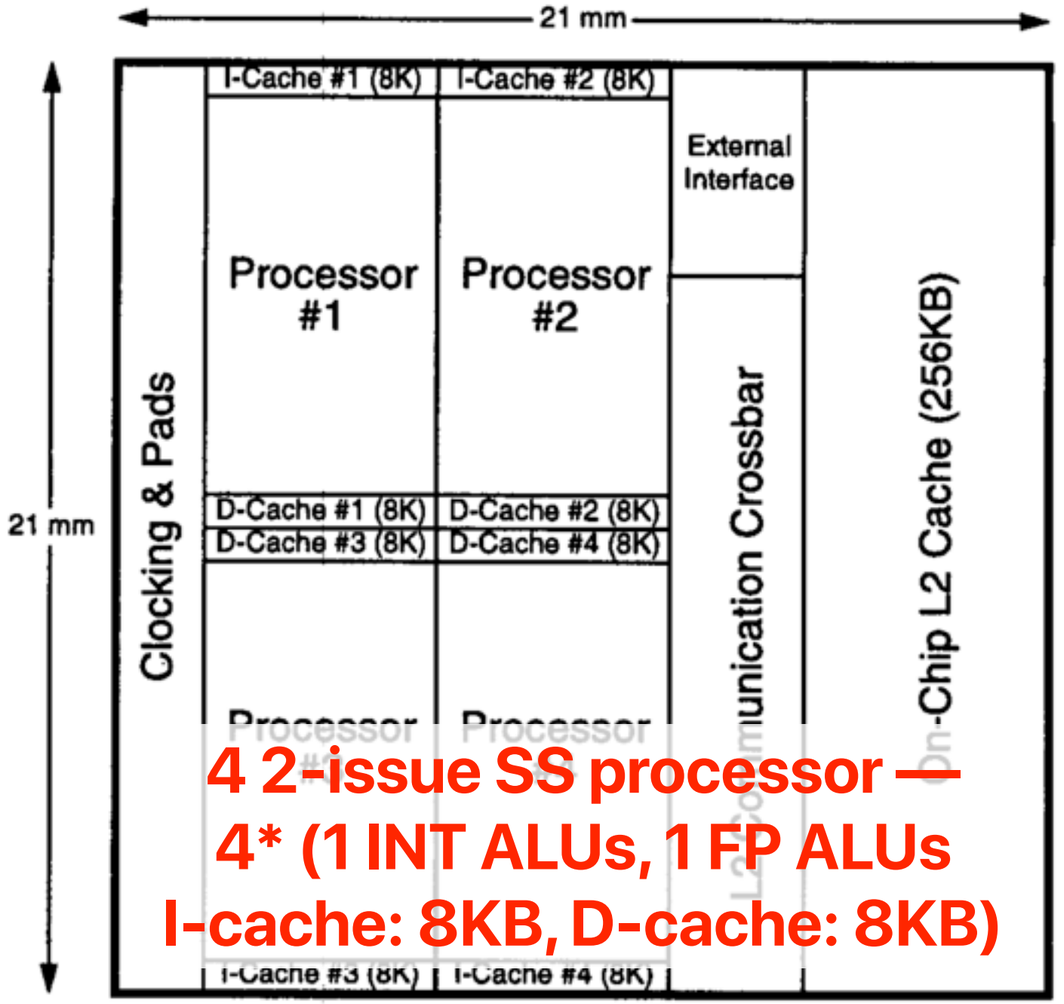
The case for a Single-Chip Multiprocessor

Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang
Stanford University

Wide-issue SS processor v.s. multiple narrower-issue SS processors



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



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

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

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

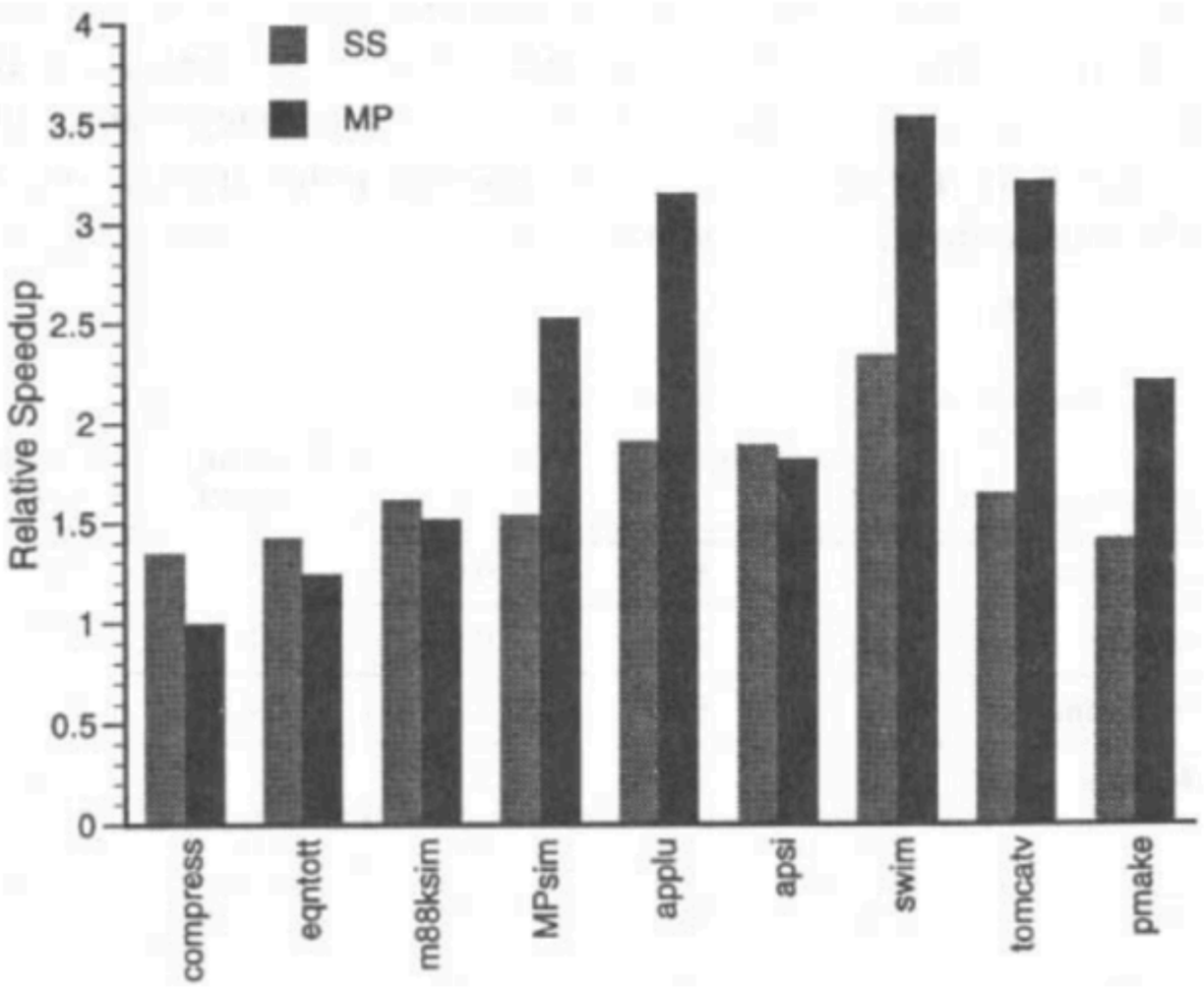
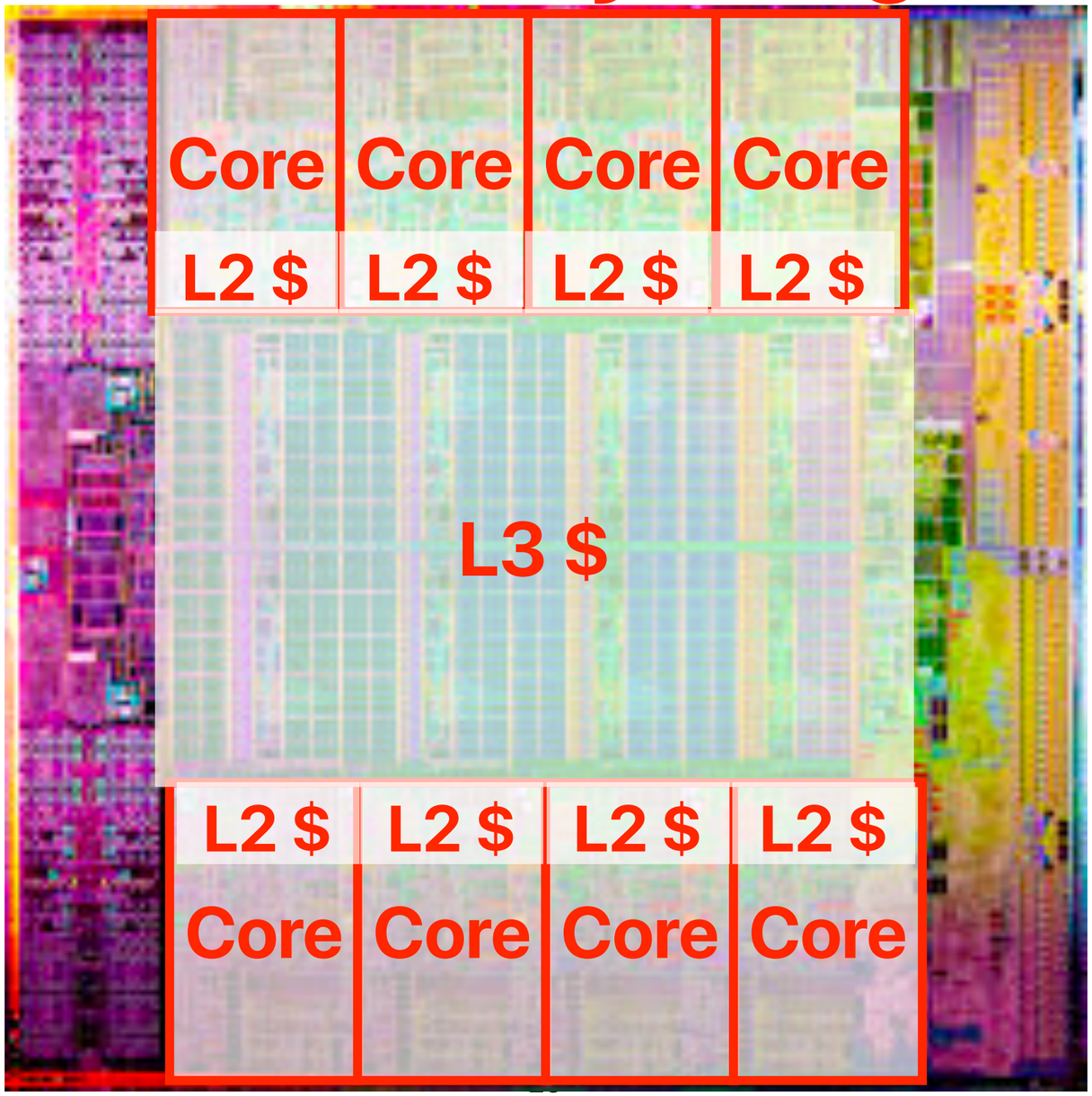
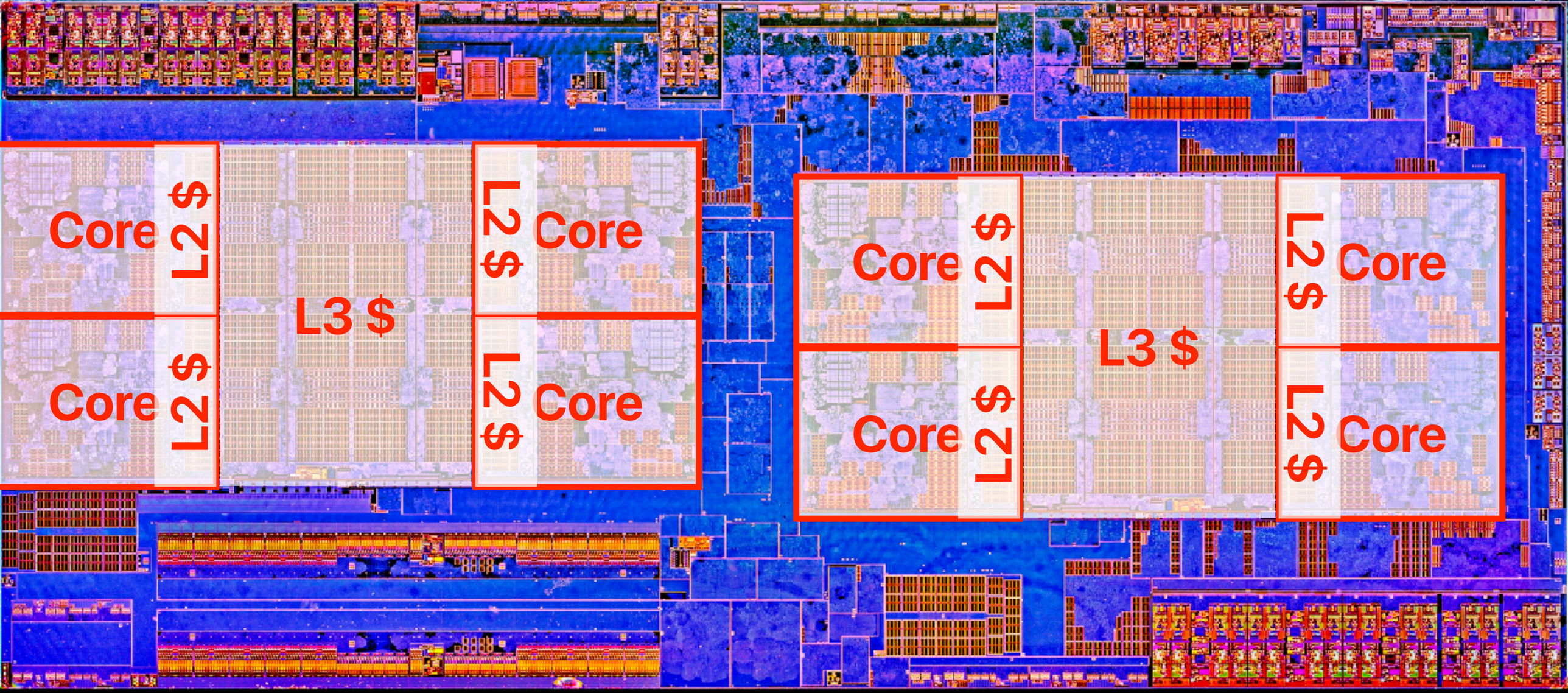


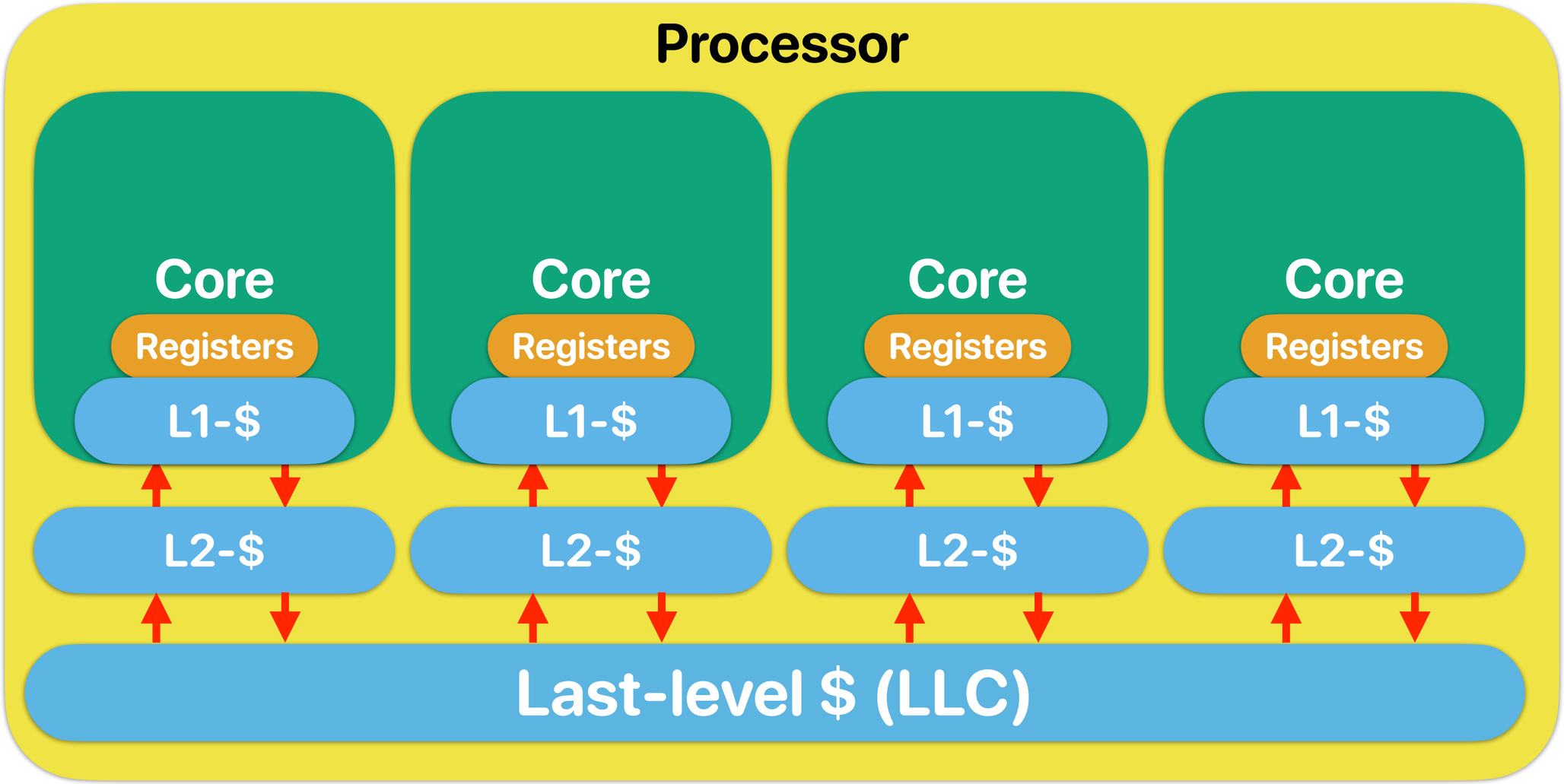
Figure 6. Performance comparison of SS and MP.

Intel Sandy Bridge




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

Concept of CMP

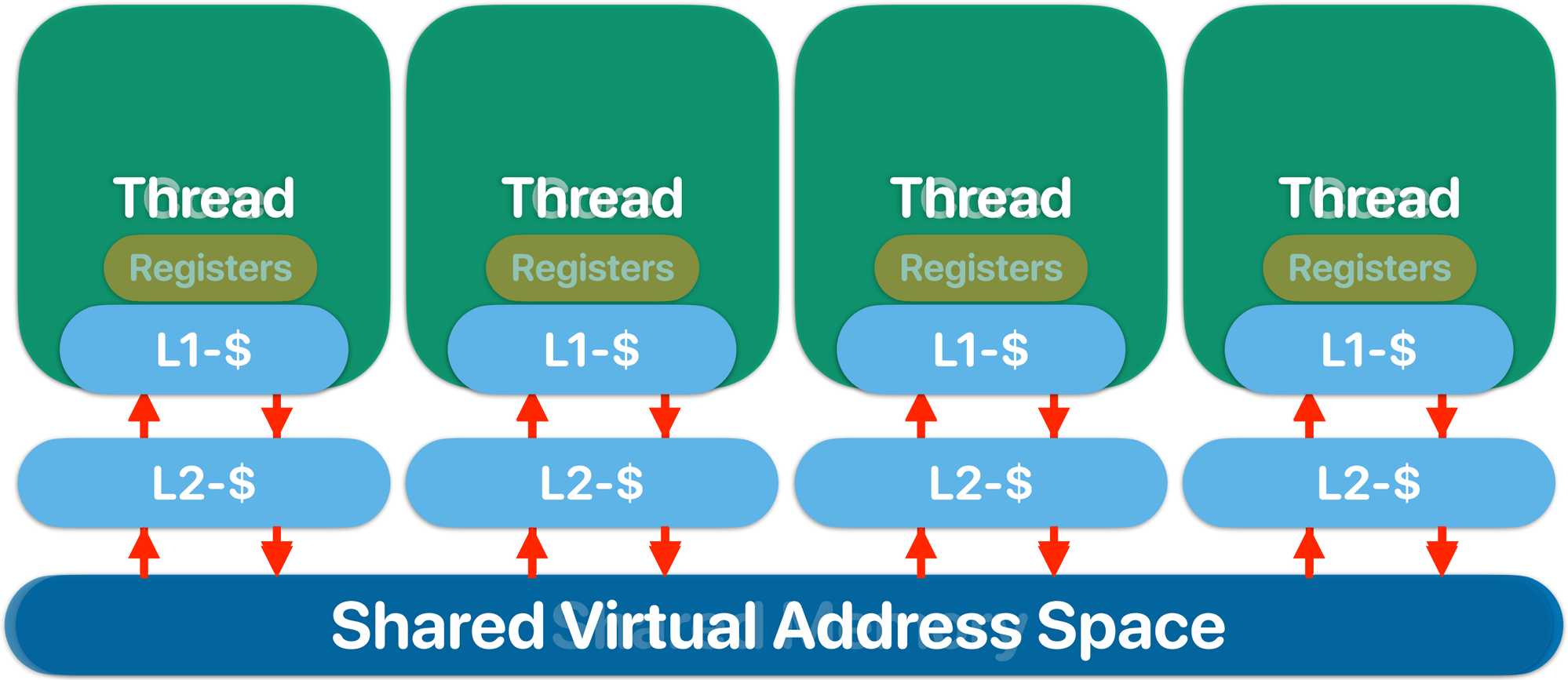


Architectural Support for Parallel Programming

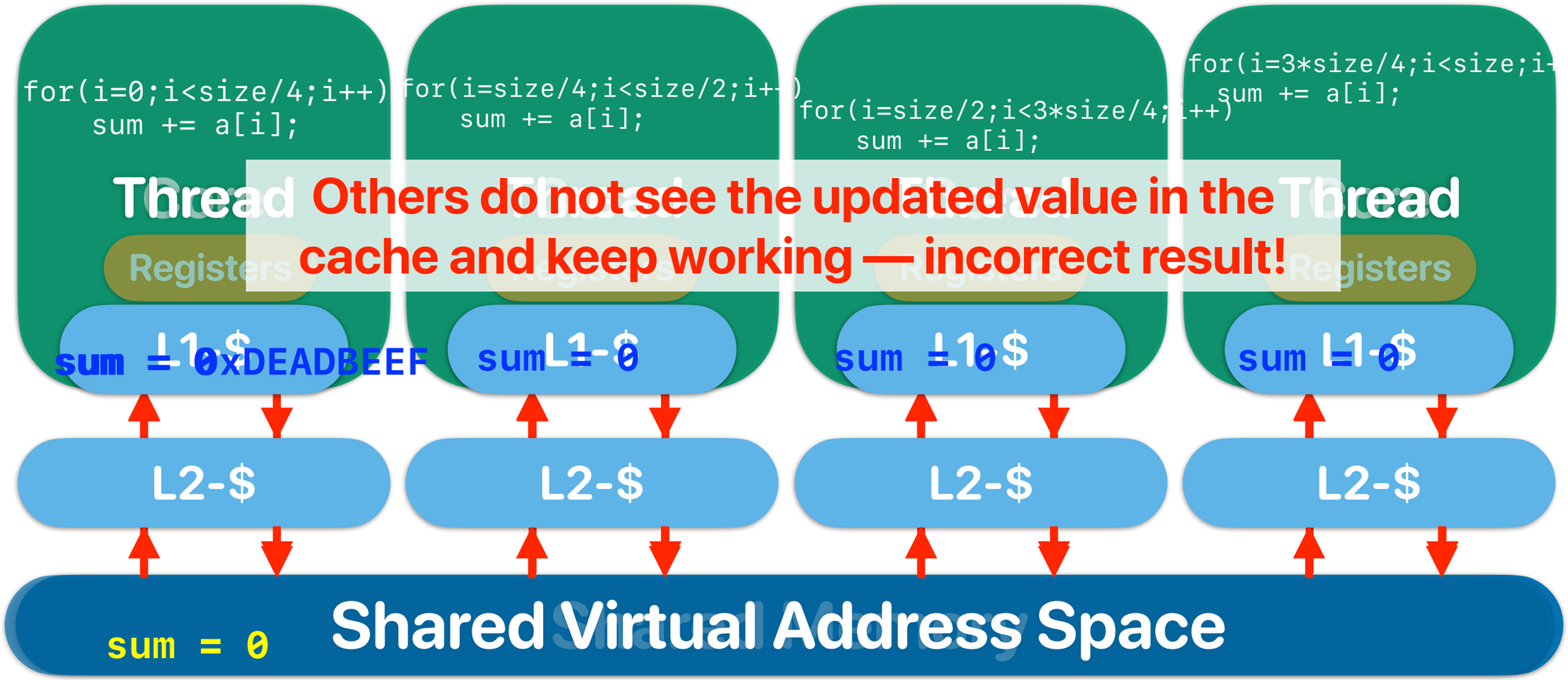
Parallel programming

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

What software thinks about “multiprogramming” hardware



What software thinks about “multiprogramming” hardware



Others do not see the updated value in the cache and keep working — incorrect result!

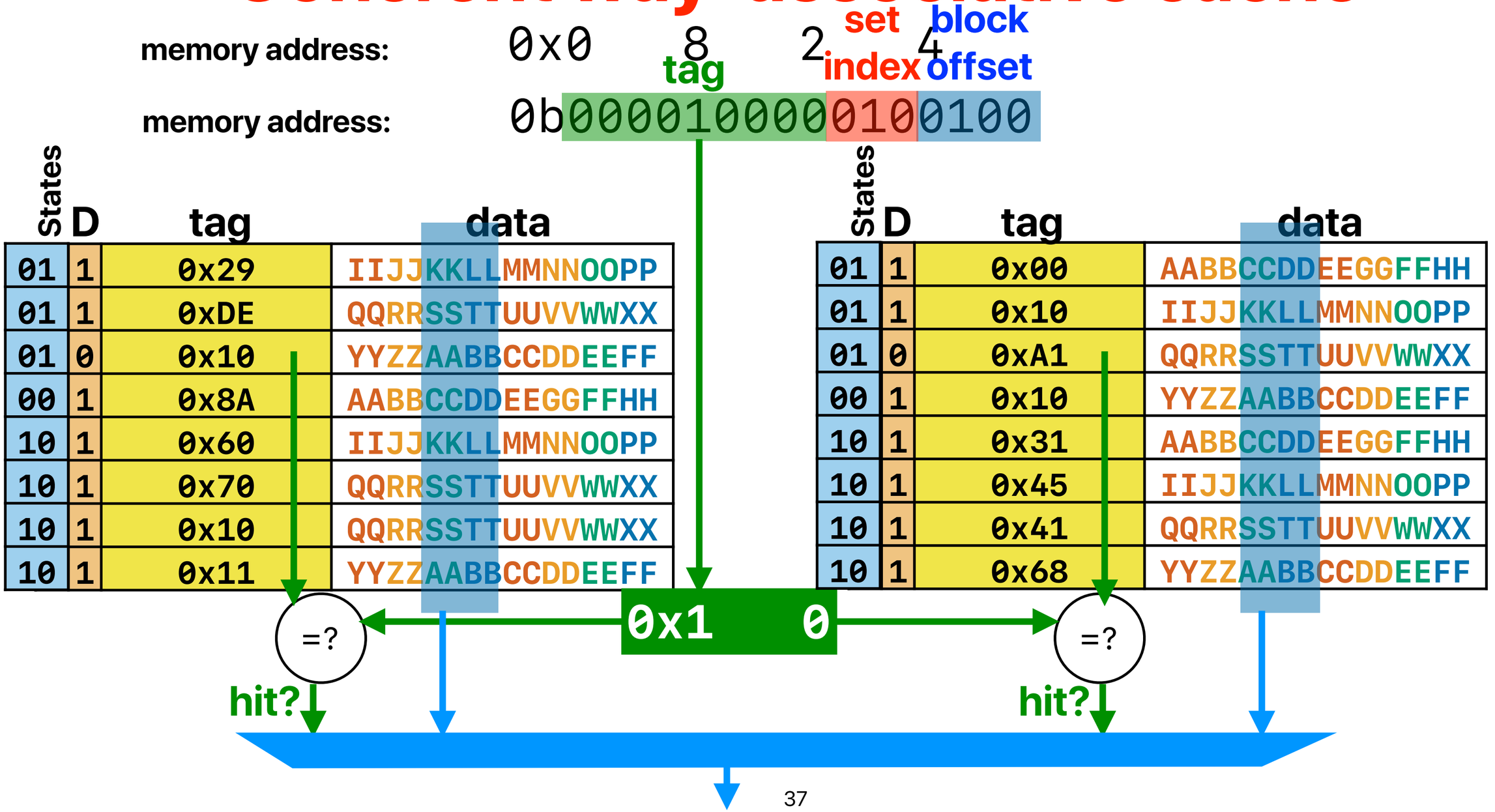
Coherency & Consistency

- Coherency — Guarantees all processors see the same value for a variable/memory address in the system when the processors need the value at the same time
  - What value should be seen
- Consistency — All threads see the change of data in the same order
  - When the memory operation should be done

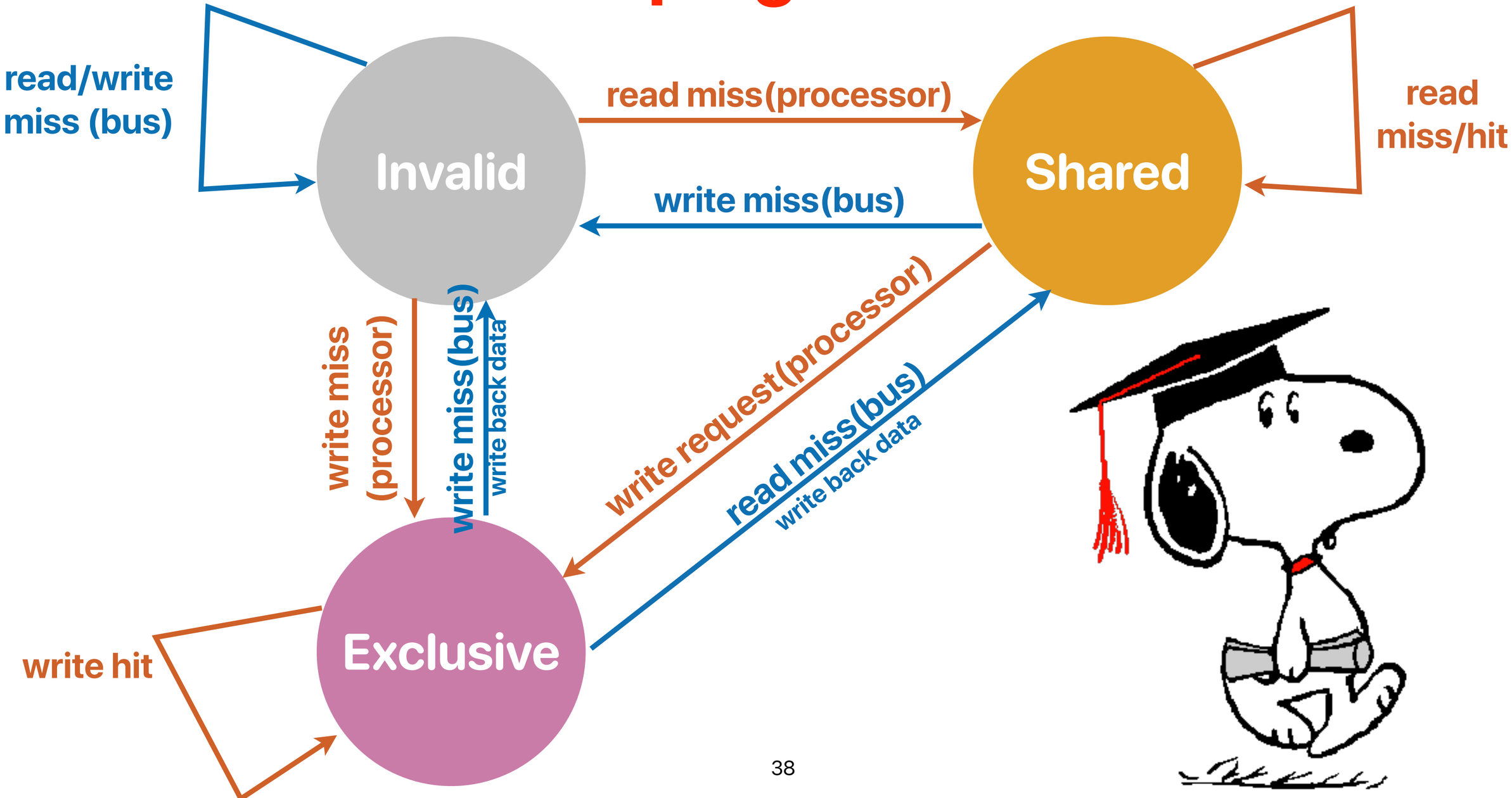
Simple cache coherency protocol

- Snooping protocol
  - Each processor broadcasts / listens to cache misses
- State associate with each block (cacheline)
  - Invalid
    - The data in the current block is invalid
  - Shared
    - The processor can read the data
    - The data may also exist on other processors
  - Exclusive
    - The processor has full permission on the data
    - The processor is the only one that has up-to-date data

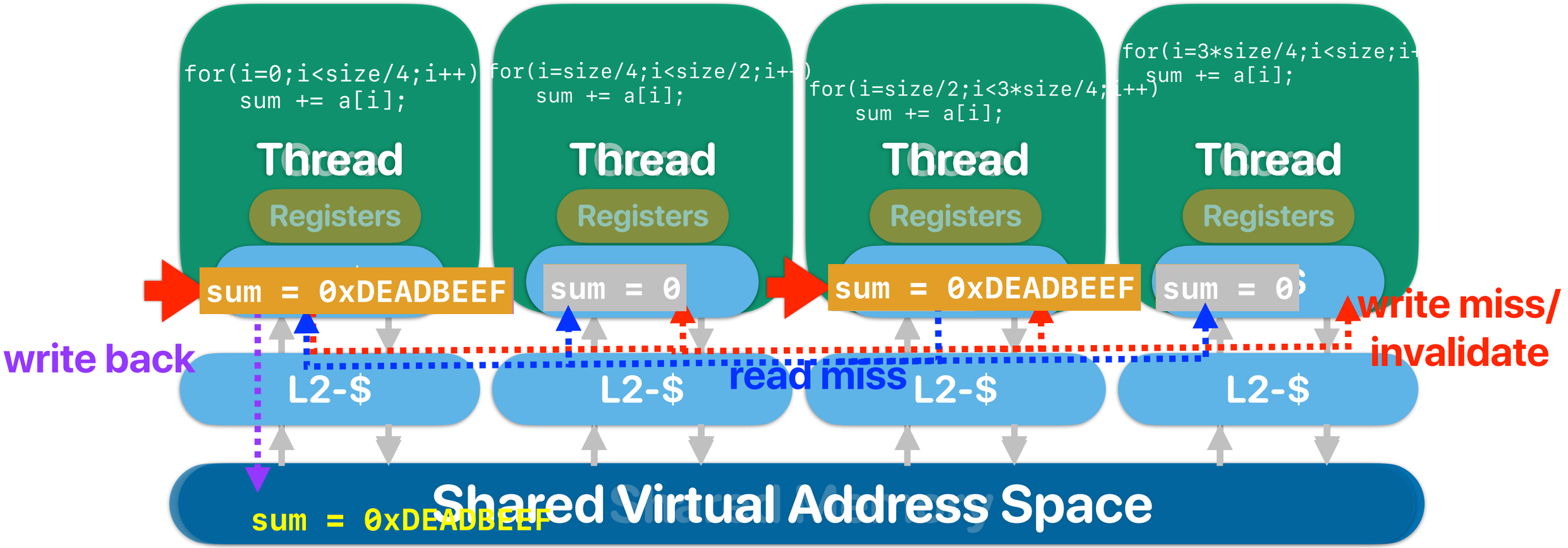
Coherent way-associative cache



Snooping Protocol



What happens when we write in coherent caches?



Observer

thread 1

```
int loop;

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

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

thread 2

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

Observer

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

thread 1	thread 2

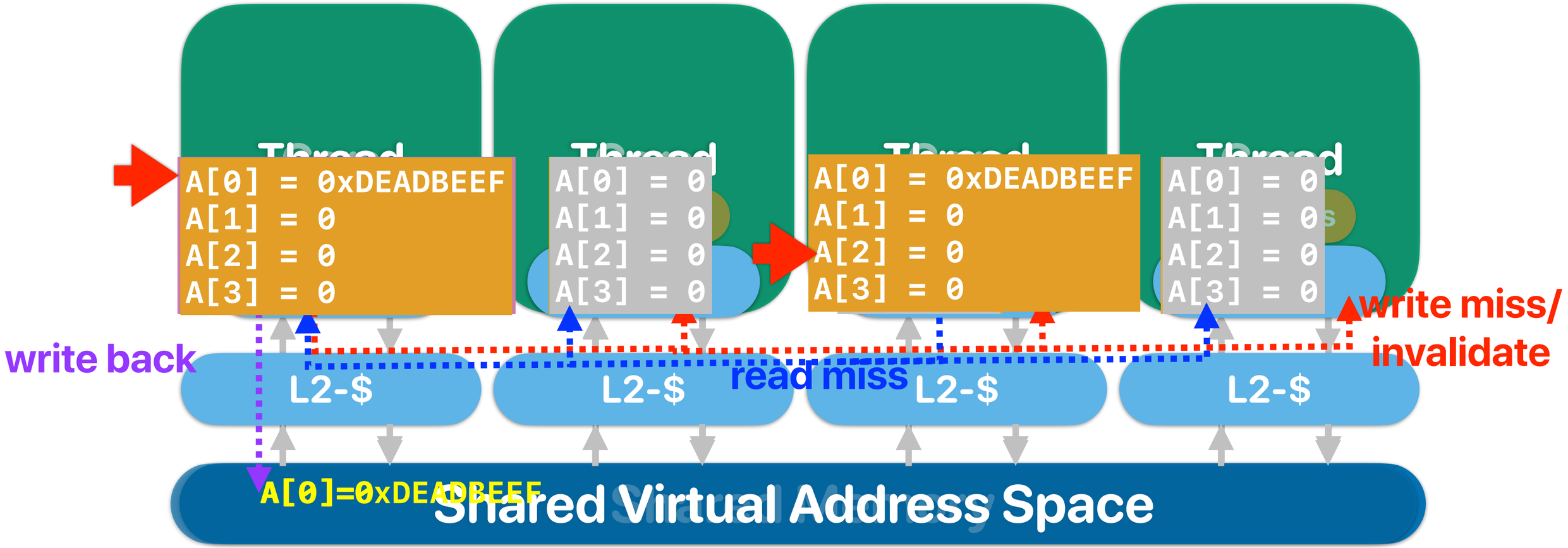
```
volatile int loop;

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

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

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

Cache coherency



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!

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

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 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!

Metric	Sum	Min	Max	Avg
Runtime (RDTSC) [s] STAT	1.1772	0.1962	0.1962	0.1962
Runtime unhalted [s] STAT	38080.0461	0	38080.0460	6346.6743
Clock [MHz] STAT	9.629741e+08	1697.5067	962966500	1.604957e+08
CPI STAT	17.7088	1	5.4991	2.9515
Temperature [C] STAT	236	36	49	39.3333
Energy [J] STAT	2.5281	0	2.5281	0.4213
Power [W] STAT	12.8846	0	12.8846	2.1474
Energy PP0 [J] STAT	2.3954	0	2.3954	0.3992
Power PP0 [W] STAT	12.2080	0	12.2080	2.0347
Energy PP1 [J] STAT	0	0	0	0
Power PP1 [W] STAT	0	0	0	0
Energy DRAM [J] STAT	0.2024	0	0.2024	0.0337
Power DRAM [W] STAT	1.0315	0	1.0315	0.1719

Metric	Sum	Min	Max	Avg
Runtime (RDTSC) [s] STAT	4.0692	0.6782	0.6782	0.6782
Runtime unhalted [s] STAT	38080.0031	0	38080.0030	6346.6672
Clock [MHz] STAT	2.211432e+08	797.0617	221140000	3.685720e+07
CPI STAT	12.0339	1	4.4400	2.0057
Temperature [C] STAT	213	35	36	35.5000
Energy [J] STAT	1.4547	0	1.4547	0.2425
Power [W] STAT	2.1450	0	2.1450	0.3575
Energy PP0 [J] STAT	1.0040	0	1.0040	0.1673
Power PP0 [W] STAT	1.4804	0	1.4804	0.2467
Energy PP1 [J] STAT	0	0	0	0
Power PP1 [W] STAT	0	0	0	0
Energy DRAM [J] STAT	0.6870	0	0.6870	0.1145
Power DRAM [W] STAT	1.0130	0	1.0130	0.1688

Metric	Sum	Min	Max	Avg
Runtime (RDTSC) [s] STAT	28.3404	4.7234	4.7234	4.7234
Runtime unhalted [s] STAT	5.8087	1.914906e-05	5.8083	0.9681
Clock [MHz] STAT	20478.2138	2237.6941	4560.8206	3413.0356
CPI STAT	13.7354	0.2683	4.8856	2.2892
Temperature [C] STAT	264	40	59	44
Energy [J] STAT	106.6913	0	106.6913	17.7819
Power [W] STAT	22.5877	0	22.5877	3.7646
Energy PP0 [J] STAT	103.5564	0	103.5564	17.2594
Power PP0 [W] STAT	21.9240	0	21.9240	3.6540
Energy PP1 [J] STAT	0	0	0	0
Power PP1 [W] STAT	0	0	0	0
Energy DRAM [J] STAT	4.7322	0	4.7322	0.7887
Power DRAM [W] STAT	1.0019	0	1.0019	0.1670

Metric	Sum	Min	Max	Avg
Runtime (RDTSC) [s] STAT	161.6694	26.9449	26.9449	26.9449
Runtime unhalted [s] STAT	5.8108	0.0002	5.8088	0.9685
Clock [MHz] STAT	4788.5470	797.9943	798.2168	798.0912
CPI STAT	4.6770	0.2683	1.1603	0.7795
Temperature [C] STAT	211	34	36	35.1667
Energy [J] STAT	47.8532	0	47.8532	7.9755
Power [W] STAT	1.7760	0	1.7760	0.2960
Energy PP0 [J] STAT	29.9814	0	29.9814	4.9969
Power PP0 [W] STAT	1.1127	0	1.1127	0.1855
Energy PP1 [J] STAT	0	0	0	0
Power PP1 [W] STAT	0	0	0	0
Energy DRAM [J] STAT	26.9831	0	26.9831	4.4972
Power DRAM [W] STAT	1.0014	0	1.0014	0.1669

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)

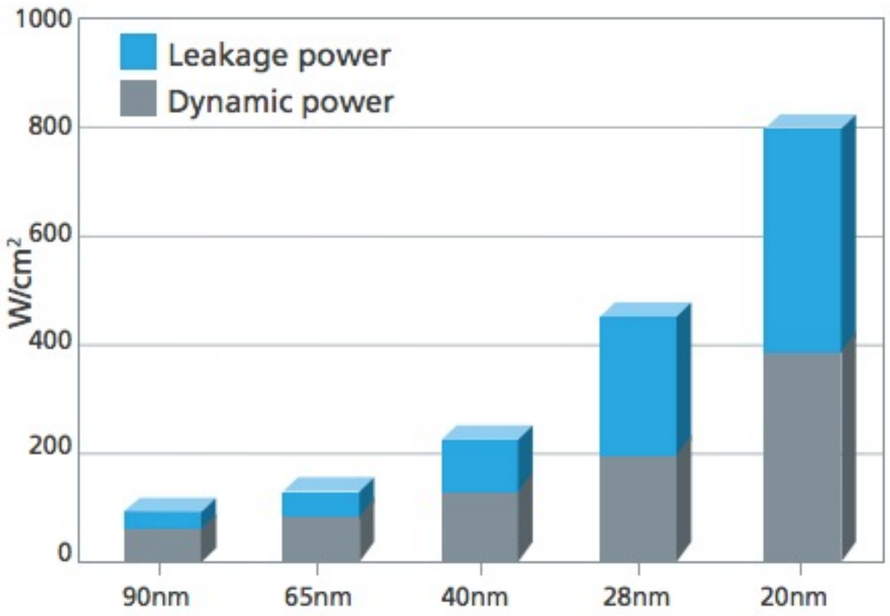


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

Dennardian Broken

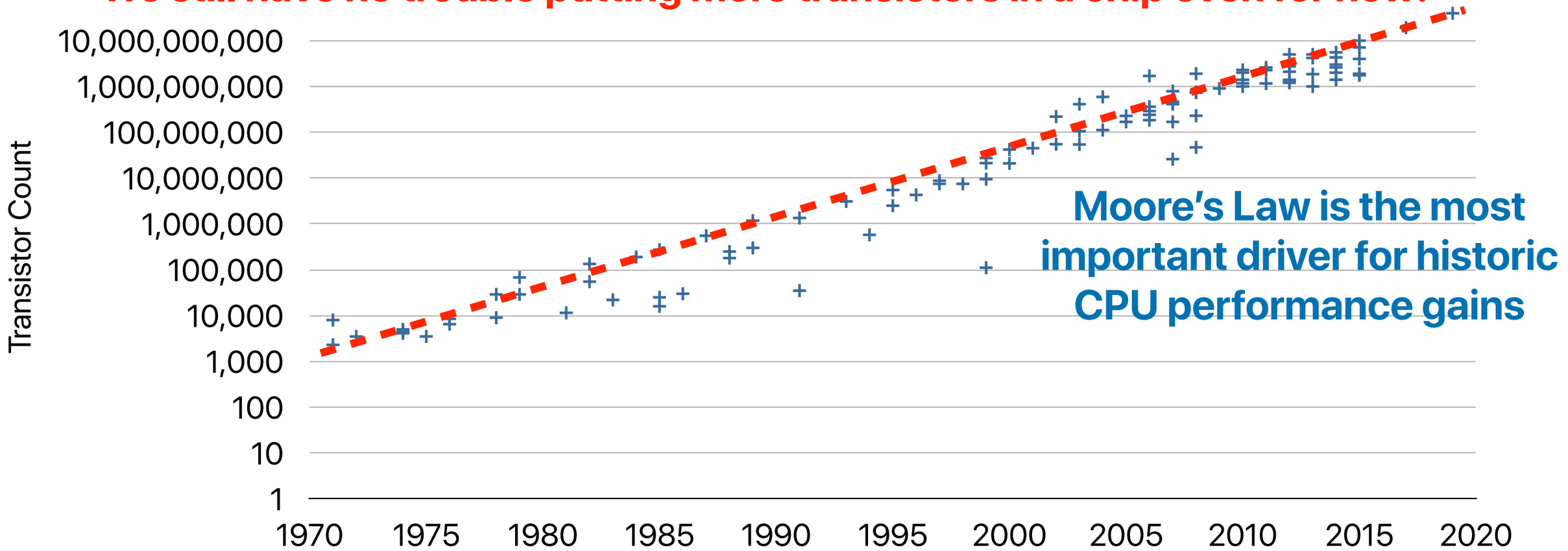
- Given a scaling factor S

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!



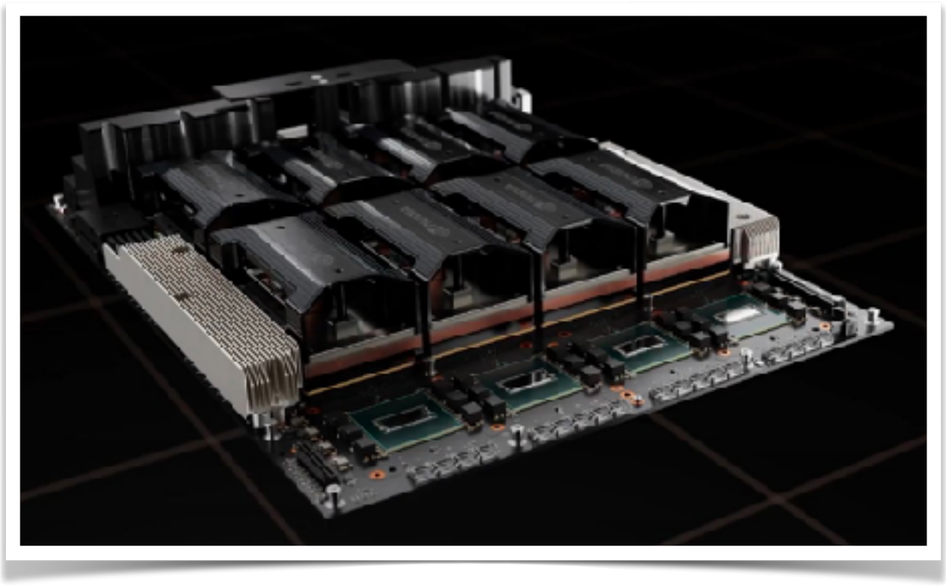
(1) Moore, G. E. (1965), 'Cramming more components onto integrated circuits', Electronics 38 (8) .

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

If you can add power budget…

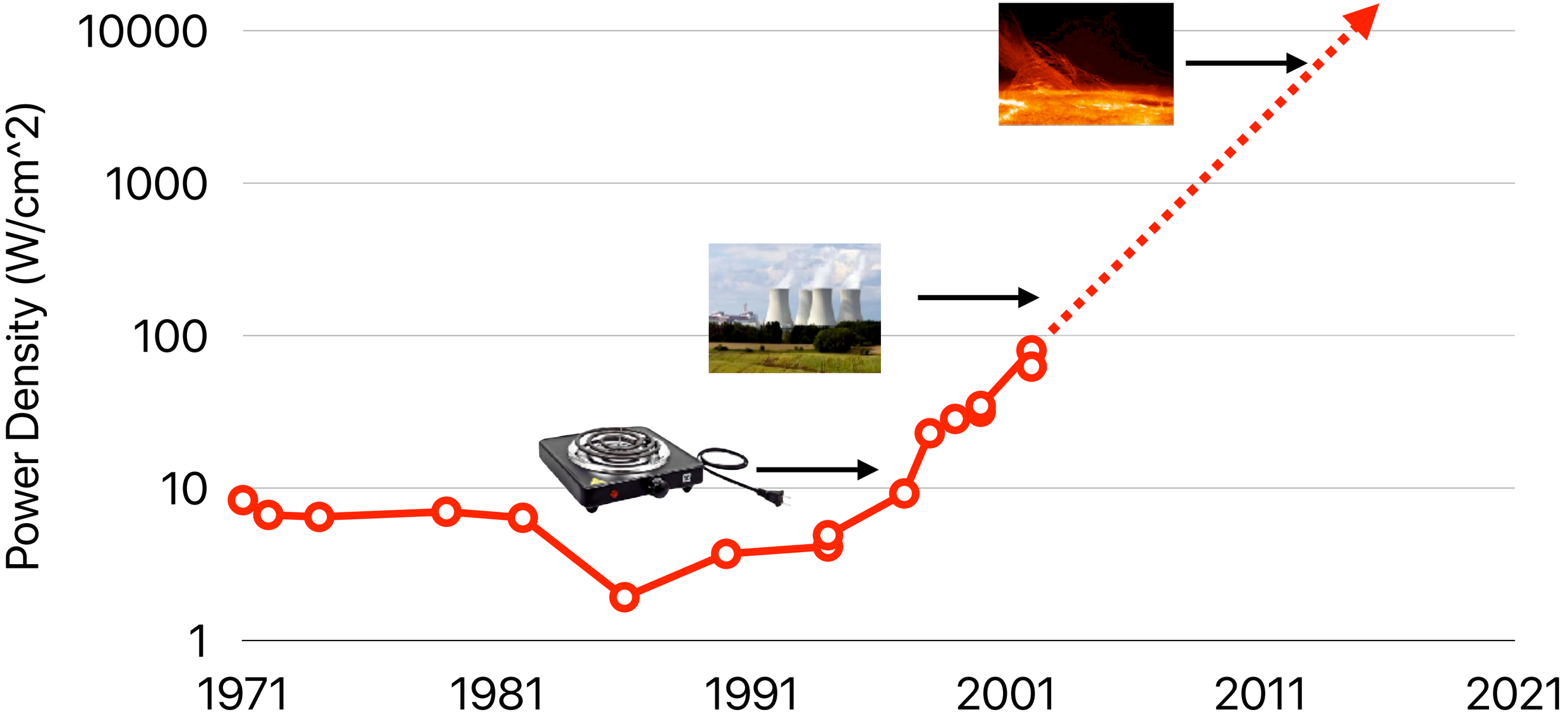
NVIDIA Accelerator Specification Comparison			
	H100	A100 (80GB)	V100
FP32 CUDA Cores	16896	6912	5120
Tensor Cores	528	432	640
Boost Clock	~1.78GHz (Not Finalized)	1.41GHz	1.53GHz
Memory Clock	4.8Gbps HBM3	3.2Gbps HBM2e	1.75Gbps HBM2
Memory Bus Width	5120-bit	5120-bit	4096-bit
Memory Bandwidth	3TB/sec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

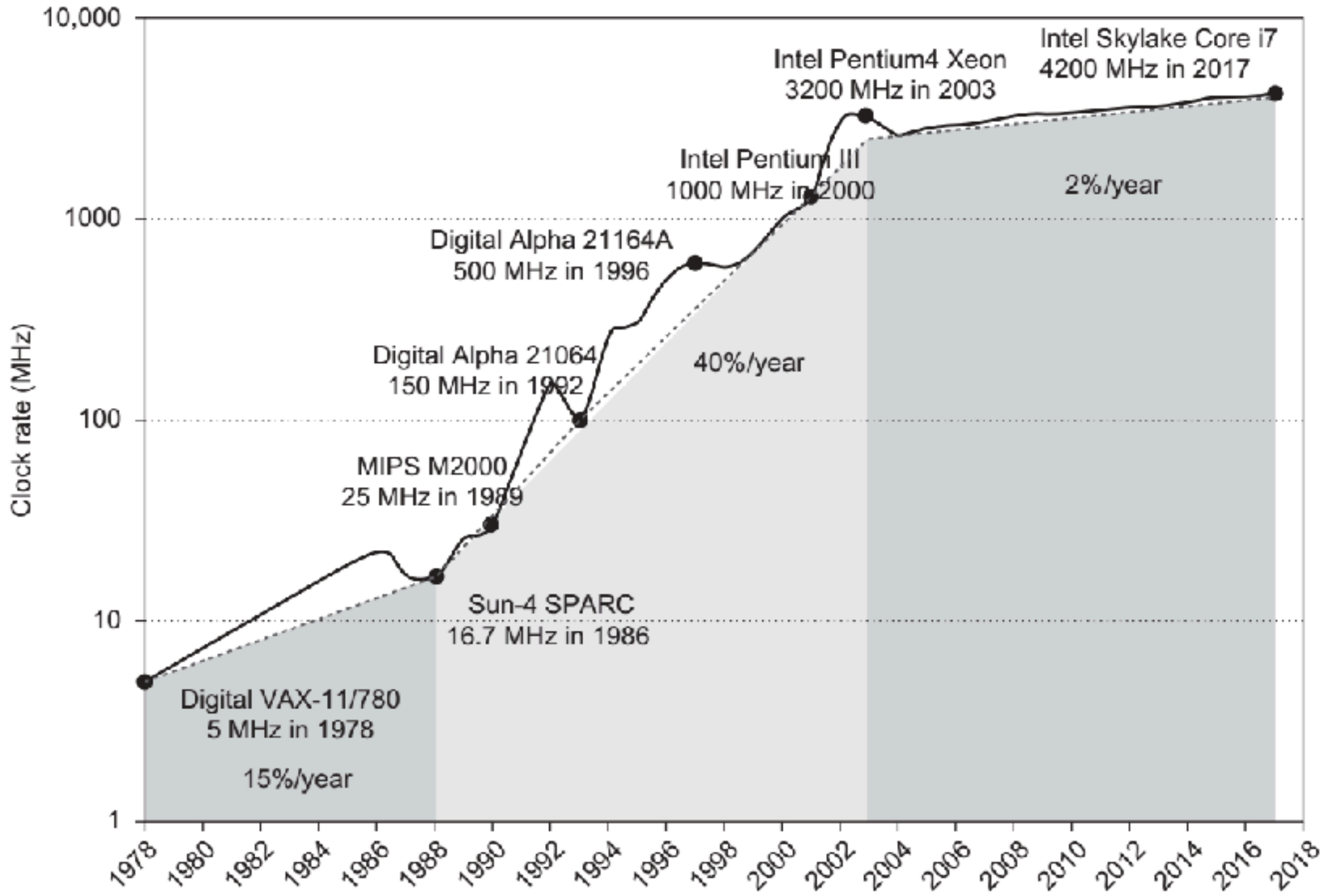
On ~ 50W

Off ~ 0W

Dark!

=100W!

Clock rate improvement is limited nowadays



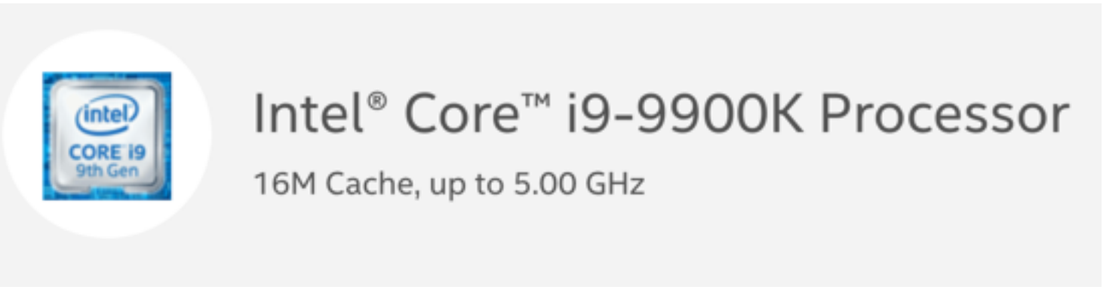
Solutions/trends in dark silicon era

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.

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

How about static power?

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

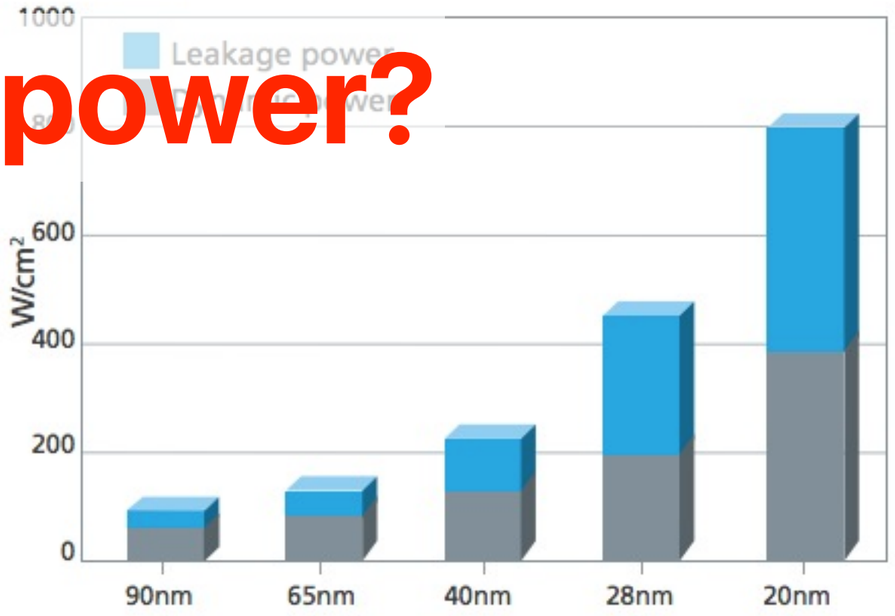


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

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

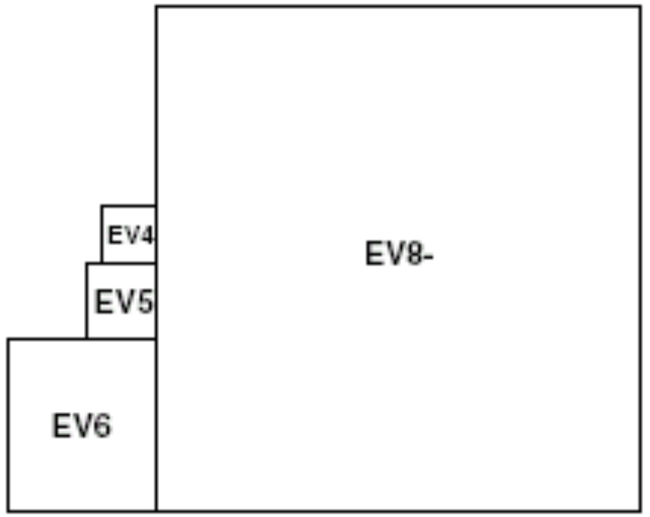
UCSD and HP Labs*

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

Areas of different processor generations

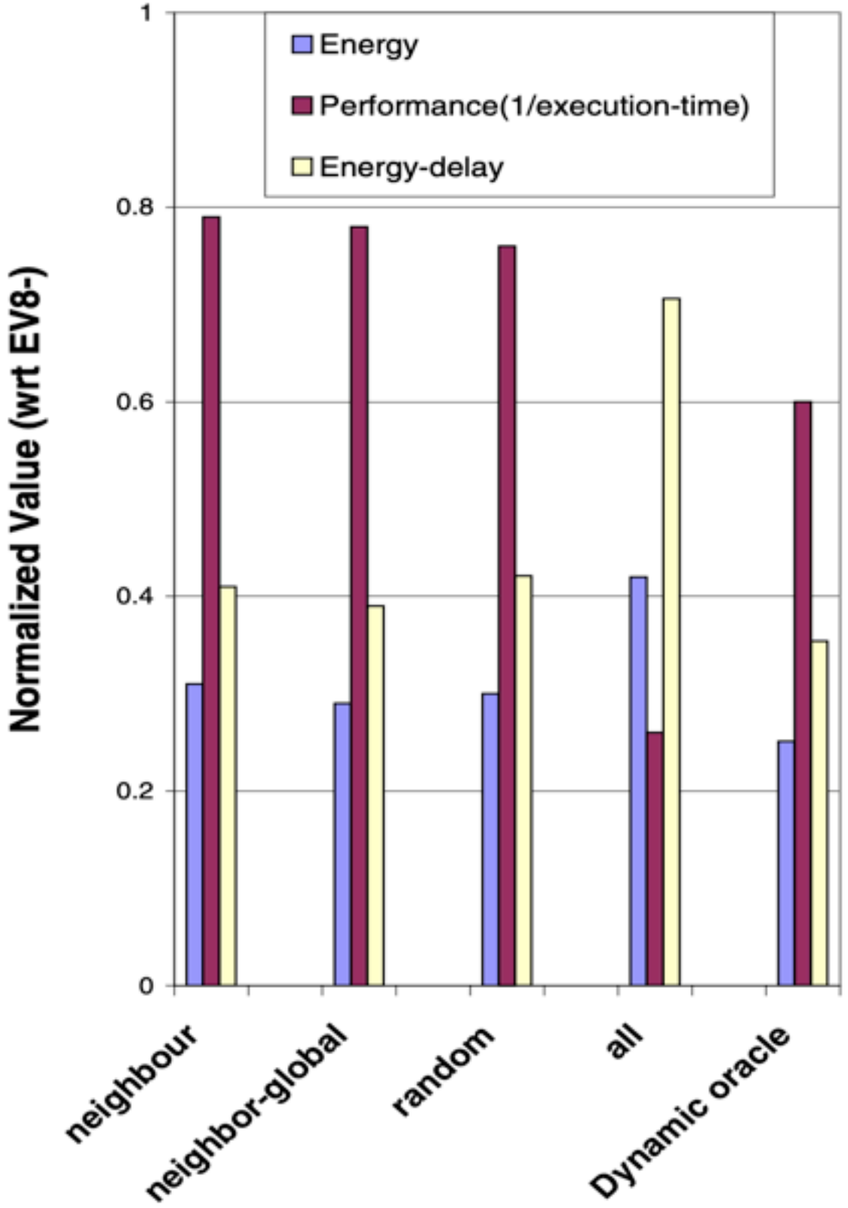
- You fit about 5 EV5 cores within the same area of an EV6
- If you build a quad-core EV6, you can use the same area to
  - build 20-core EV5
  - 3EV6+5EV5

Processor	EV5	EV6	EV6+
Issue-width	4	6 (OOO)	6 (OOO)
I-Cache	8KB, DM	64KB, 2-way	64KB, 2-way
D-Cache	8KB, DM	64KB, 2-way	64KB, 2-way
Branch Pred.	2K-gshare	hybrid 2-level	hybrid 2-level
Number of MSHRs	4	8	16
Number of threads	1	1	4
Area (in mm^2)	5.06	24.5	29.9



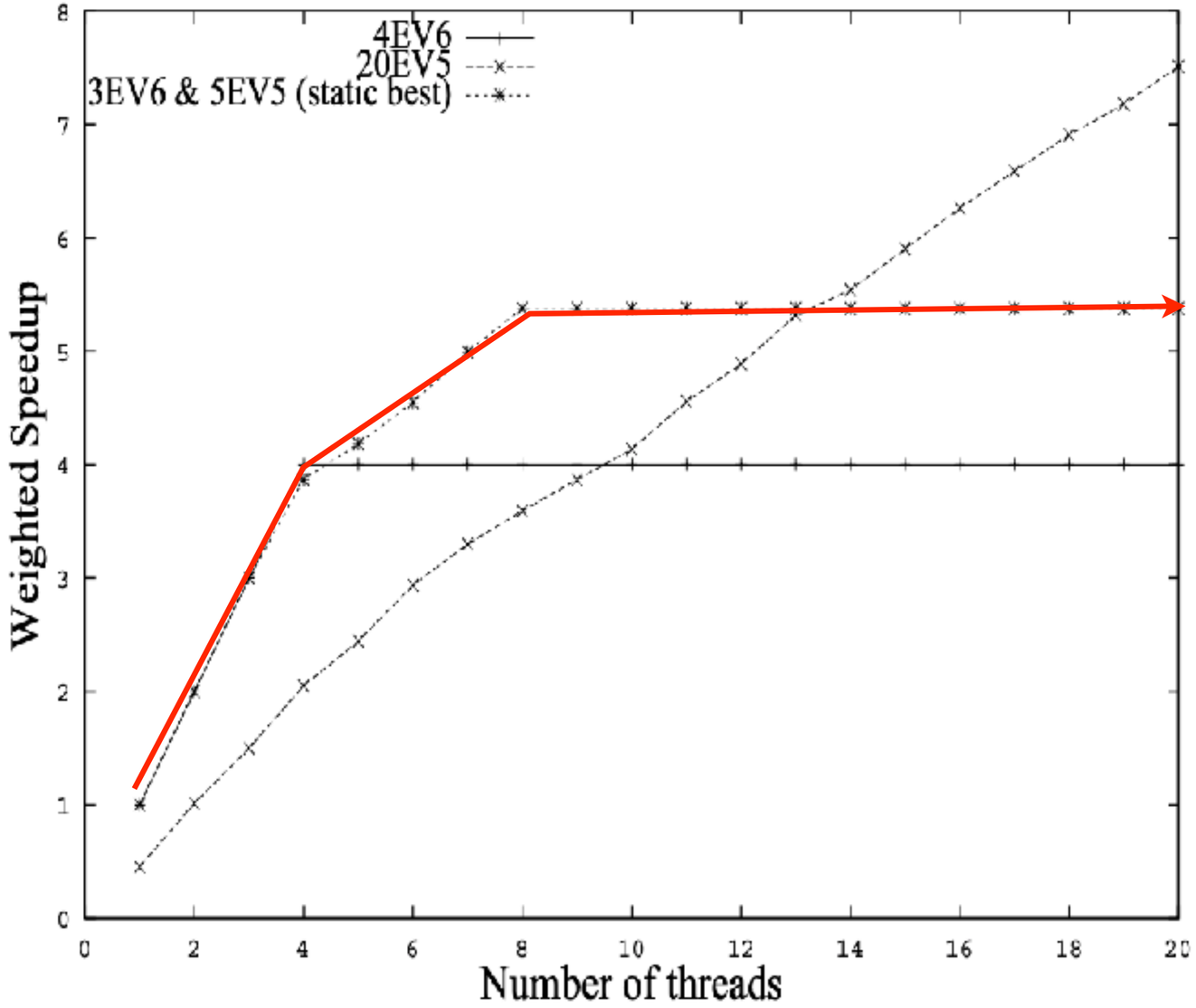
Energy-delay

- Energy * delay = Power * ET * ET = Power * ET^2

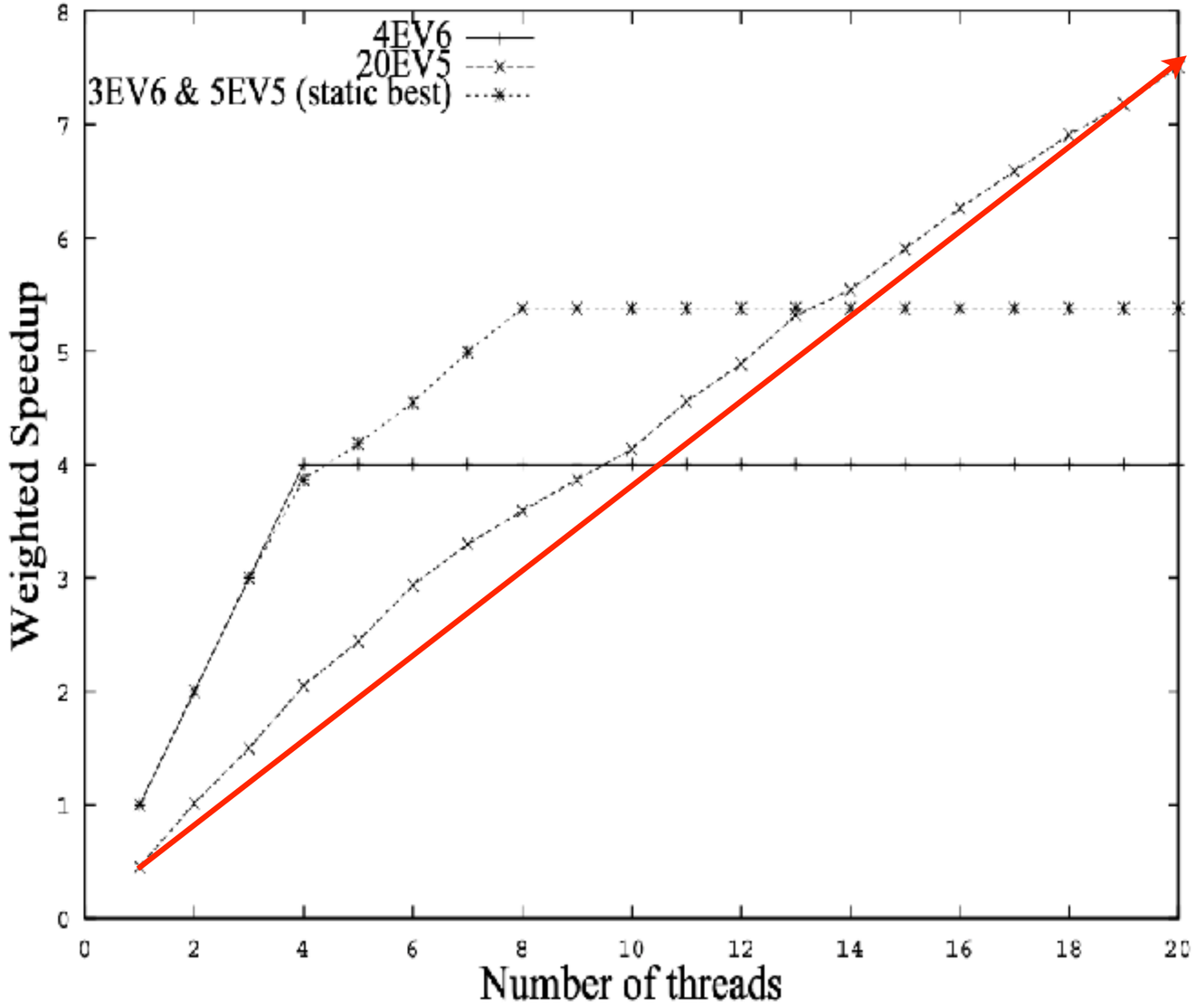


Benchmark	Total switches	% of instructions per core				Energy Savings(%)	ED Savings(%)	ED^2 Savings(%)	Perf. Loss (%)
		EV4	EV5	EV6	EV8-				
ammp	0	0	0	0	100	0	0	0	0
applu	27	2.2	0.1	54.5	43.2	42.7	38.6	33.6	7.1
apsi	2	0	0	62.2	37.8	27.6	25.3	22.9	3.1
art	0	0	0	100	0	74.4	73.5	72.6	3.3
equake	20	0	0	97.9	2.1	72.4	71.3	70.1	3.9
fma3d	0	0	0	0	100	0	0	0	0
wupwise	16	0	0	99	1	72.6	69.9	66.2	10.0
bzip	13	0	0.1	84.0	15.9	40.1	38.7	37.2	2.3
crafty	0	0	0	0	100	0	0	0	0
eon	0	0	0	100	0	77.3	76.3	75.3	4.2
gzip	82	0	0	95.9	4.1	74.0	73.0	71.8	3.9
mcf	0	0	0	0	100	0	0	0	0
twolf	0	0	0	0	100	0	0	0	0
vortex	364	0	0	73.8	26.2	56.2	51.9	46.2	9.8
Average	1(median)	0.2%	0%	54.8%	45.0%	38.5%	37.0%	35.4%	3.4%

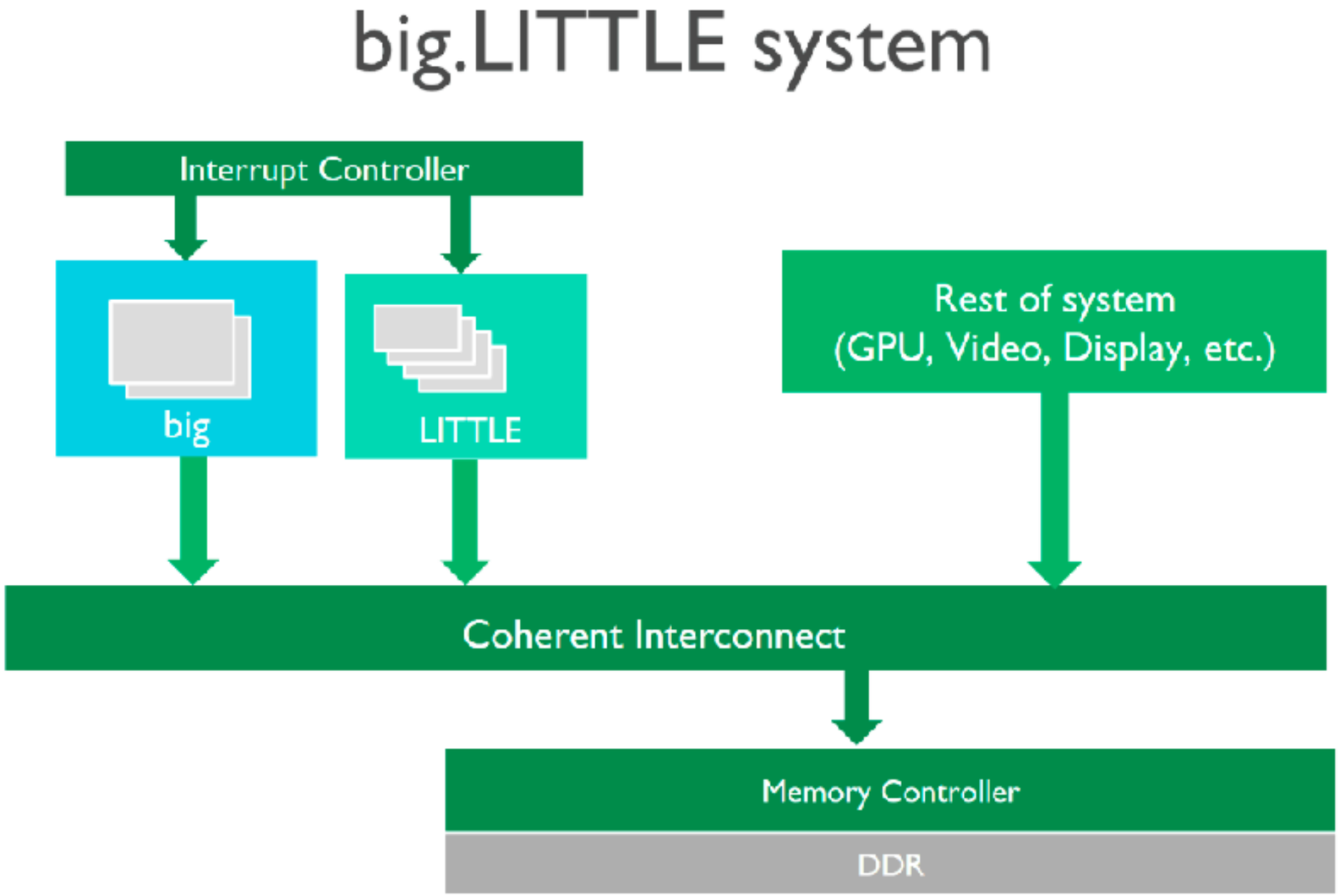
4EV6 v.s. 20 EV5 v.s. 3EV6+5EV5



4EV6 v.s. 20 EV5 v.s. 3EV6+5EV5



ARM's big.LITTLE architecture



More cores per chip, slower per core

Products Solutions Support

intel

	Intel® Xeon® Processor E7-8890 v4	Intel® Xeon® Processor E7-8893 v4	Intel® Xeon® Processor E7-8880 v4
Status	Launched	Launched	Launched
Launch Date	Q2'16	Q2'16	Q2'16
Lithography	14 nm	14 nm	14 nm
Performance			
# of Cores	24	4	22
# of Threads	48	8	44
Processor Base Frequency	2.20 GHz	3.20 GHz	2.20 GHz
Max Turbo Frequency	3.40 GHz	3.50 GHz	3.30 GHz
Cache	60 MB	60 MB	55 MB
Bus Speed	9.6 GT/s	9.6 GT/s	9.6 GT/s
# of QPI Links	3	3	3
TDP	165 W	140 W	150 W

Xeon Phi

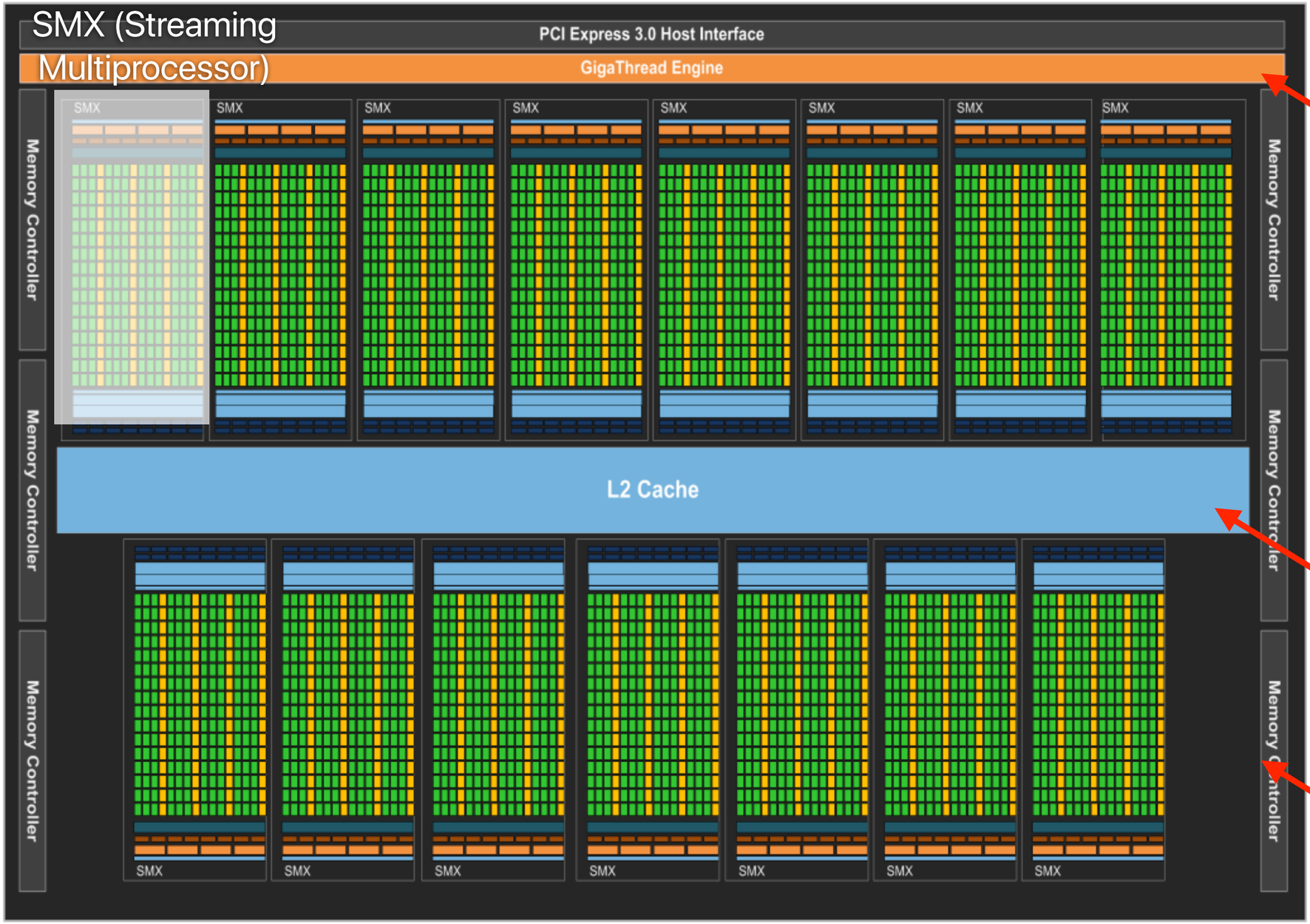
Essentials

Product Collection	Intel® Xeon Phi™ 72x5 Processor Family
Code Name	Products formerly Knights Mill
Vertical Segment	Server
Processor Number	7295
Off Roadmap	No
Status	Launched
Launch Date	Q4'17
Lithography	14 nm

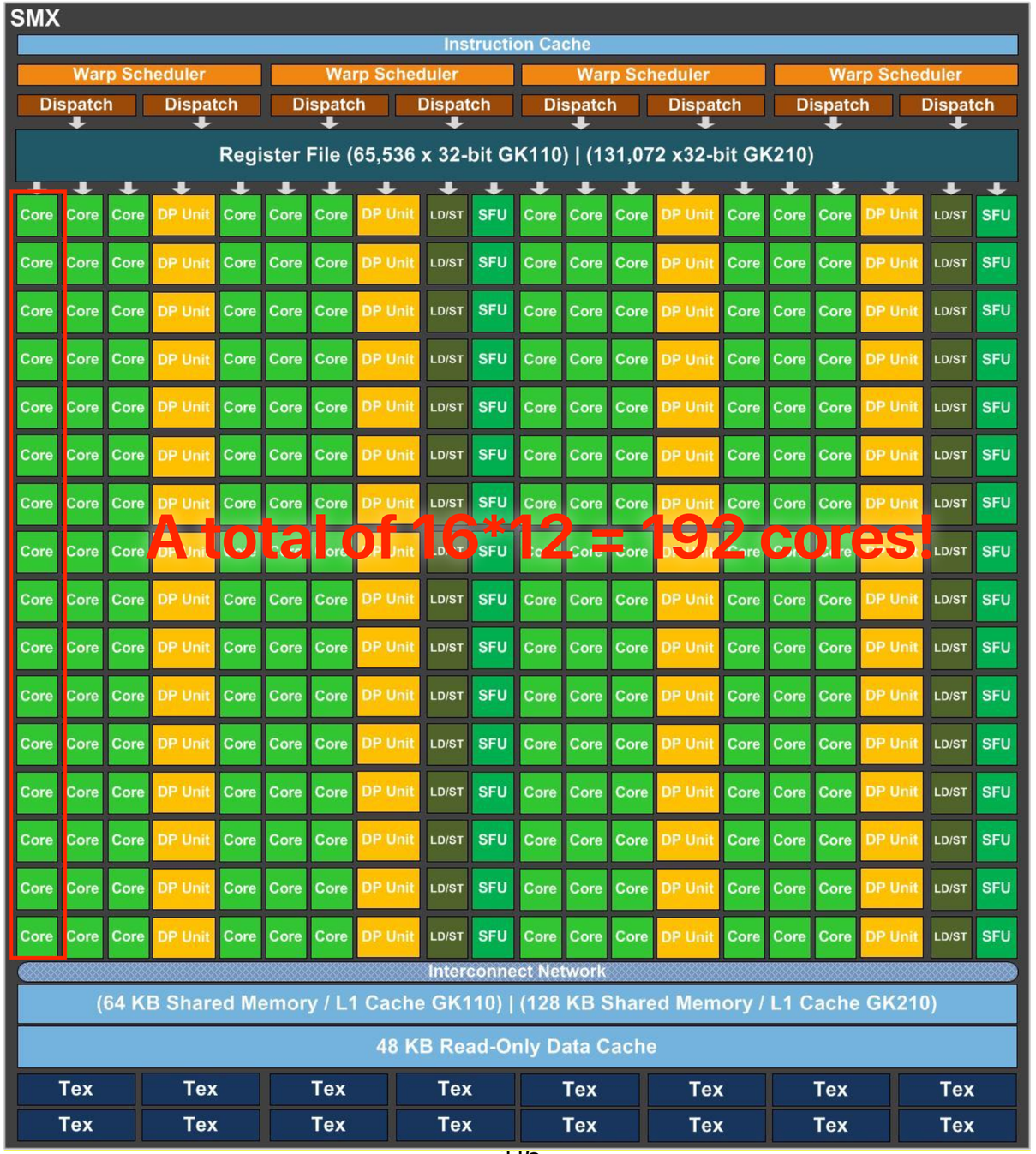
Performance

# of Cores	72
# of Threads	72
Processor Base Frequency	1.50 GHz
Max Turbo Frequency	1.50 GHz
Cache	36 MB L2 Cache
TDP	320 W

The rise of GPU

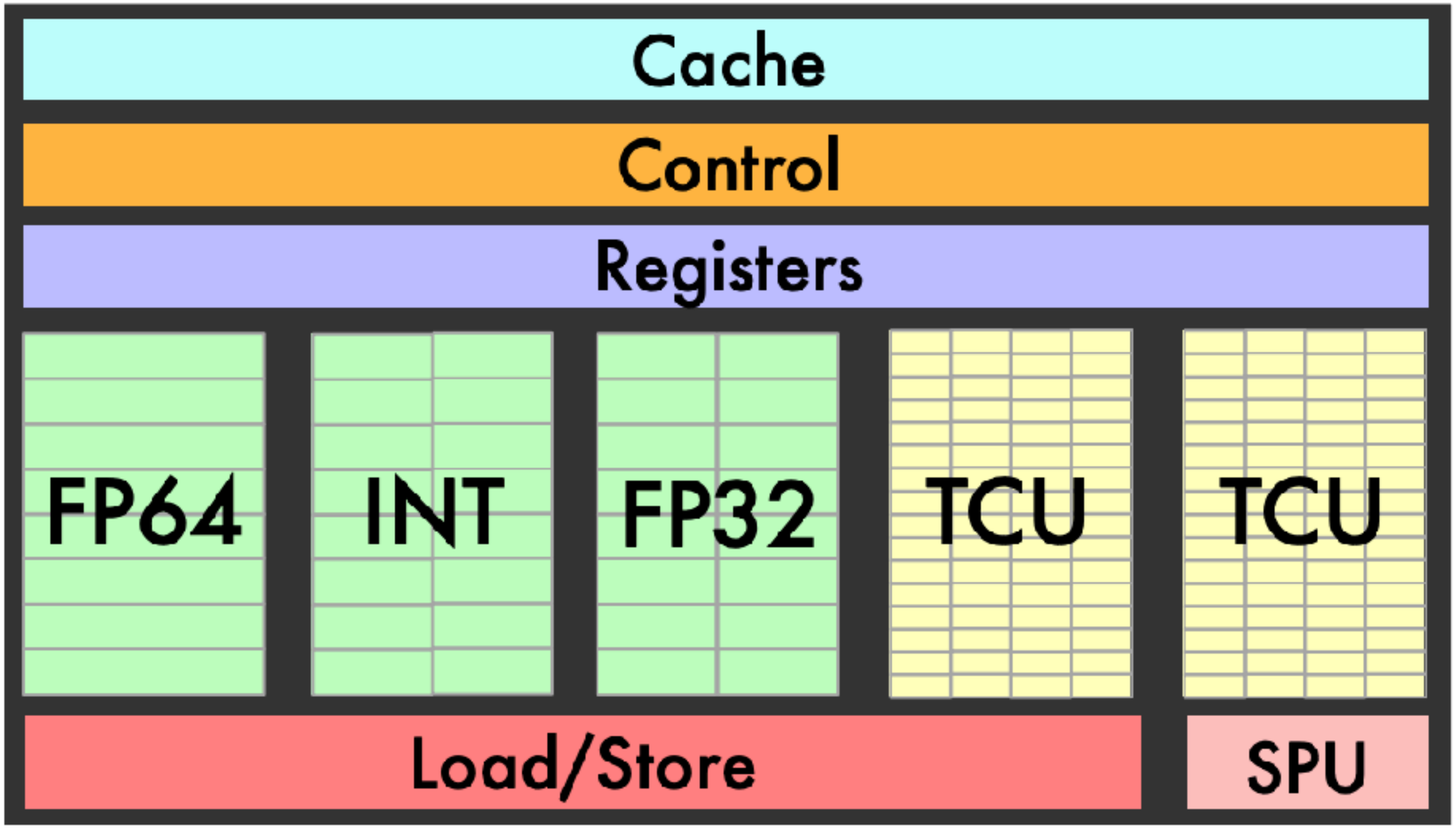


Each of these performs the same operation, but each of these is also a "thread"



Just let it dark

NVIDIA's Turing Architecture



Programming in Turing Architecture

Use tensor cores

```
cublasErrCheck(cublasSetMathMode(cublasHandle, CUBLAS_TENSOR_OP_MATH));
```

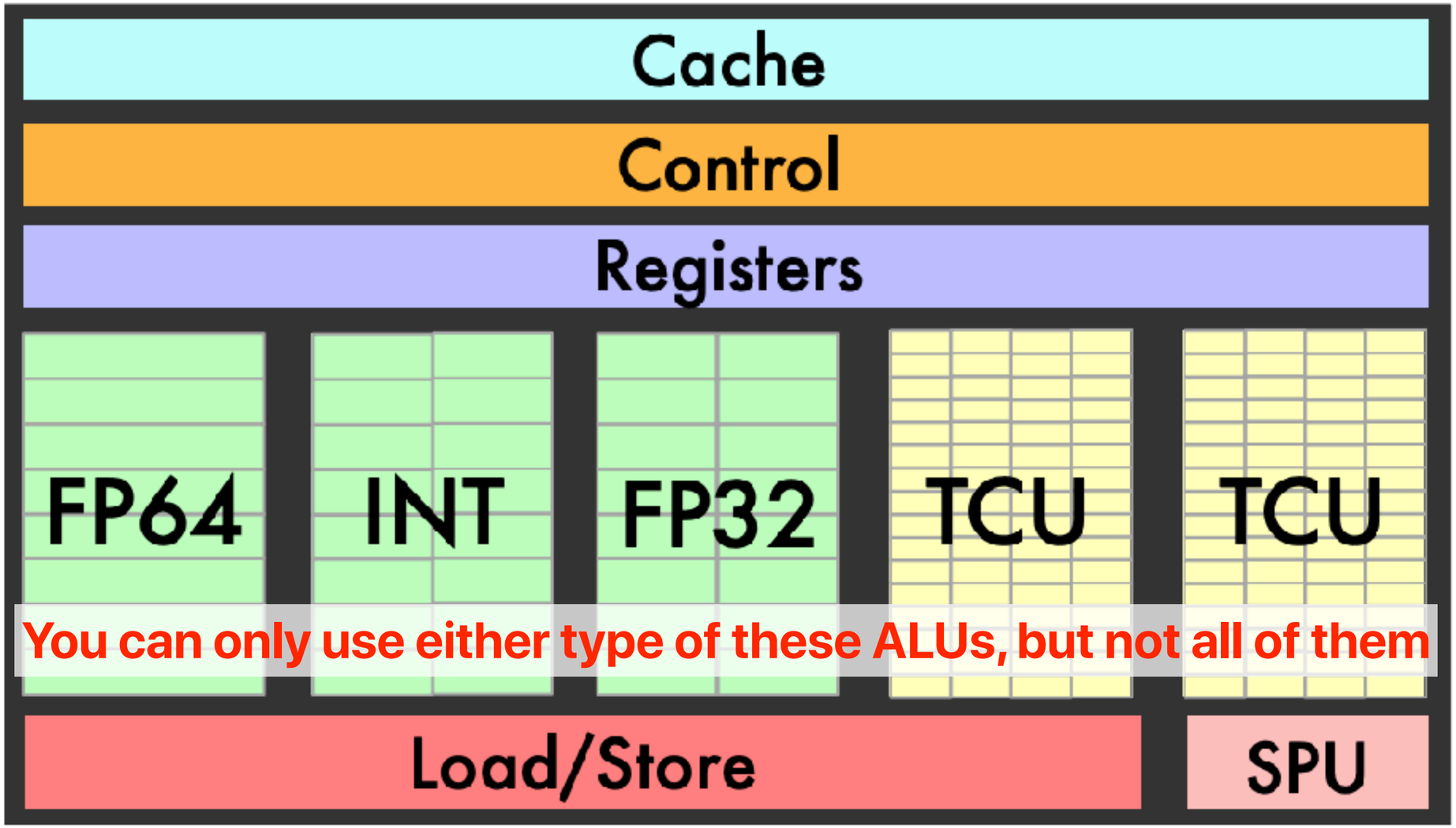
Make them 16-bit

```
convertFp32ToFp16 <<< (MATRIX_M * MATRIX_K + 255) / 256, 256 >>> (a_fp16, a_fp32,
MATRIX_M * MATRIX_K);
   convertFp32ToFp16 <<< (MATRIX_K * MATRIX_N + 255) / 256, 256 >>> (b_fp16, b_fp32,
MATRIX_K * MATRIX_N);
```

```
cublasErrCheck(cublasGemmEx(cublasHandle, CUBLAS_OP_N, CUBLAS_OP_N,
                 MATRIX_M, MATRIX_N, MATRIX_K,
                 &alpha,
                 a_fp16, CUDA_R_16F, MATRIX_M,
                 b_fp16, CUDA_R_16F, MATRIX_K,
                 &beta,
                 c_cublas, CUDA_R_32F, MATRIX_M,
                 CUDA_R_32F, CUBLAS_GEMM_DFALT_TENSOR_OP));
```

call Gemm

NVIDIA's Turing Architecture

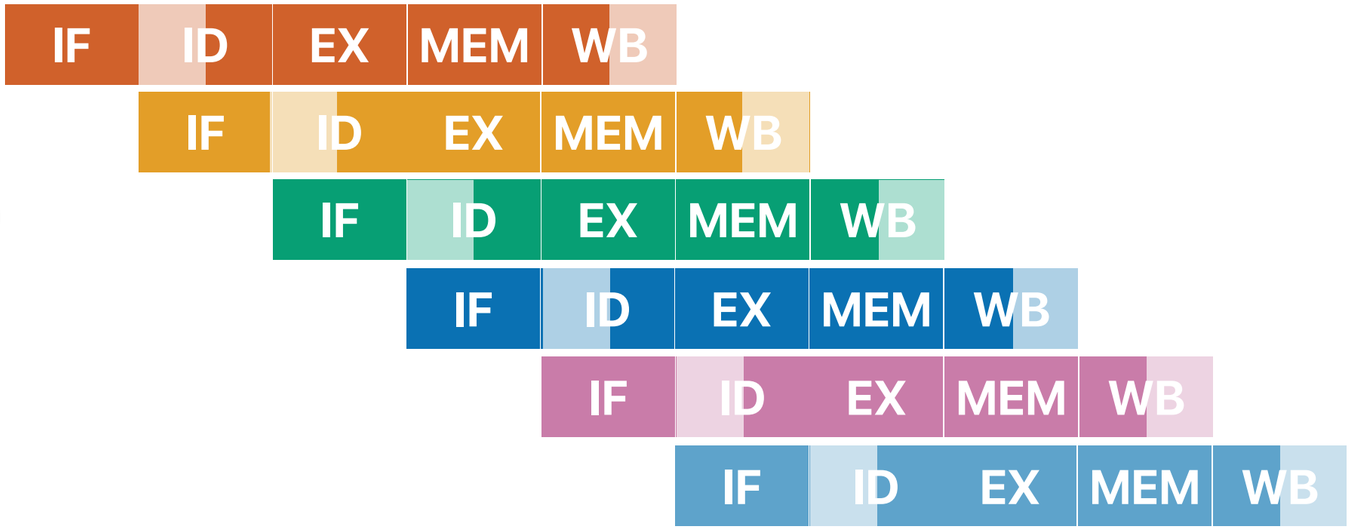


The Rise of ASICs

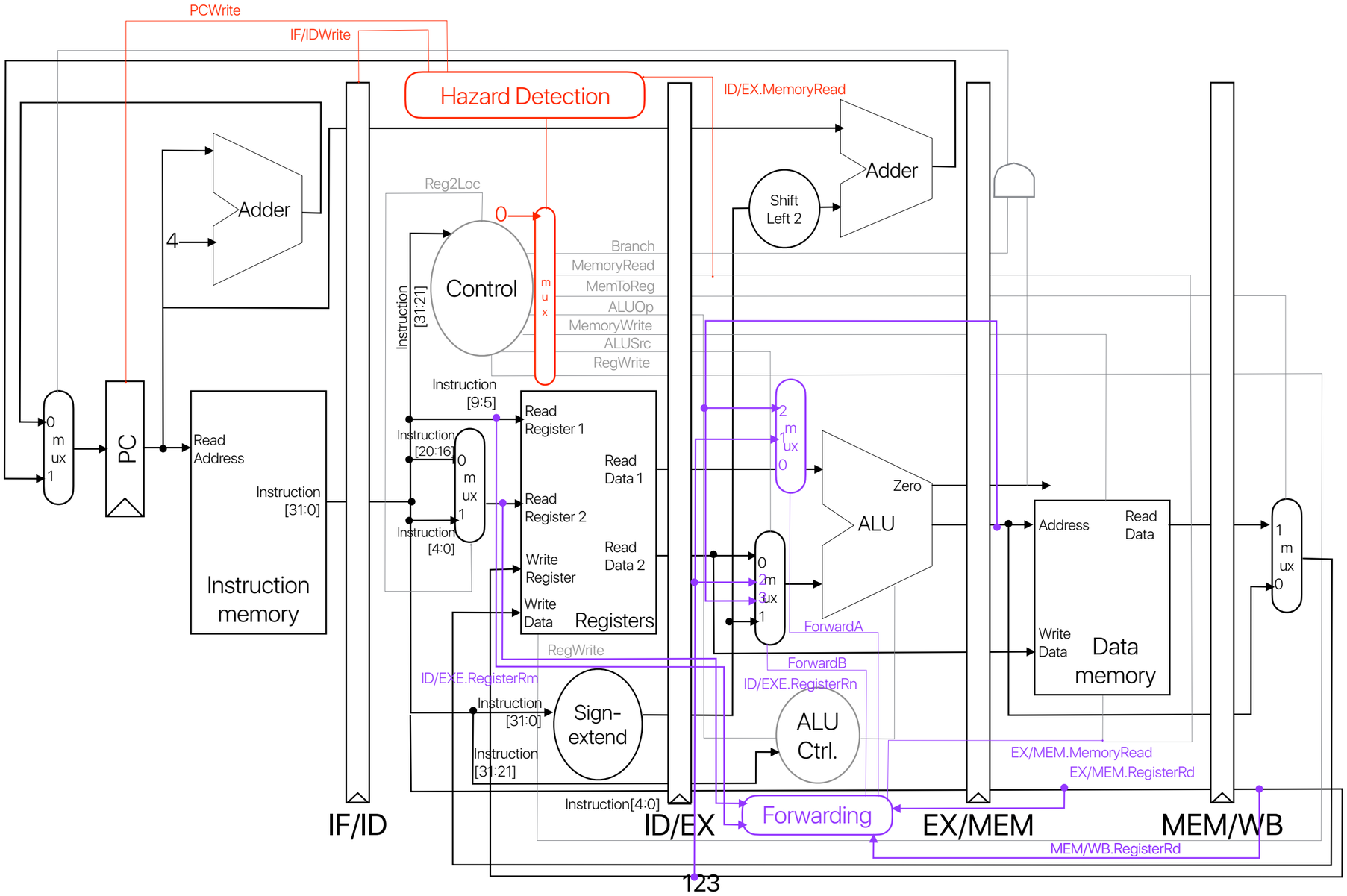
Say, we want to implement a[i] += a[i+1]*20

- This is what we need in RISC-V in each iteration

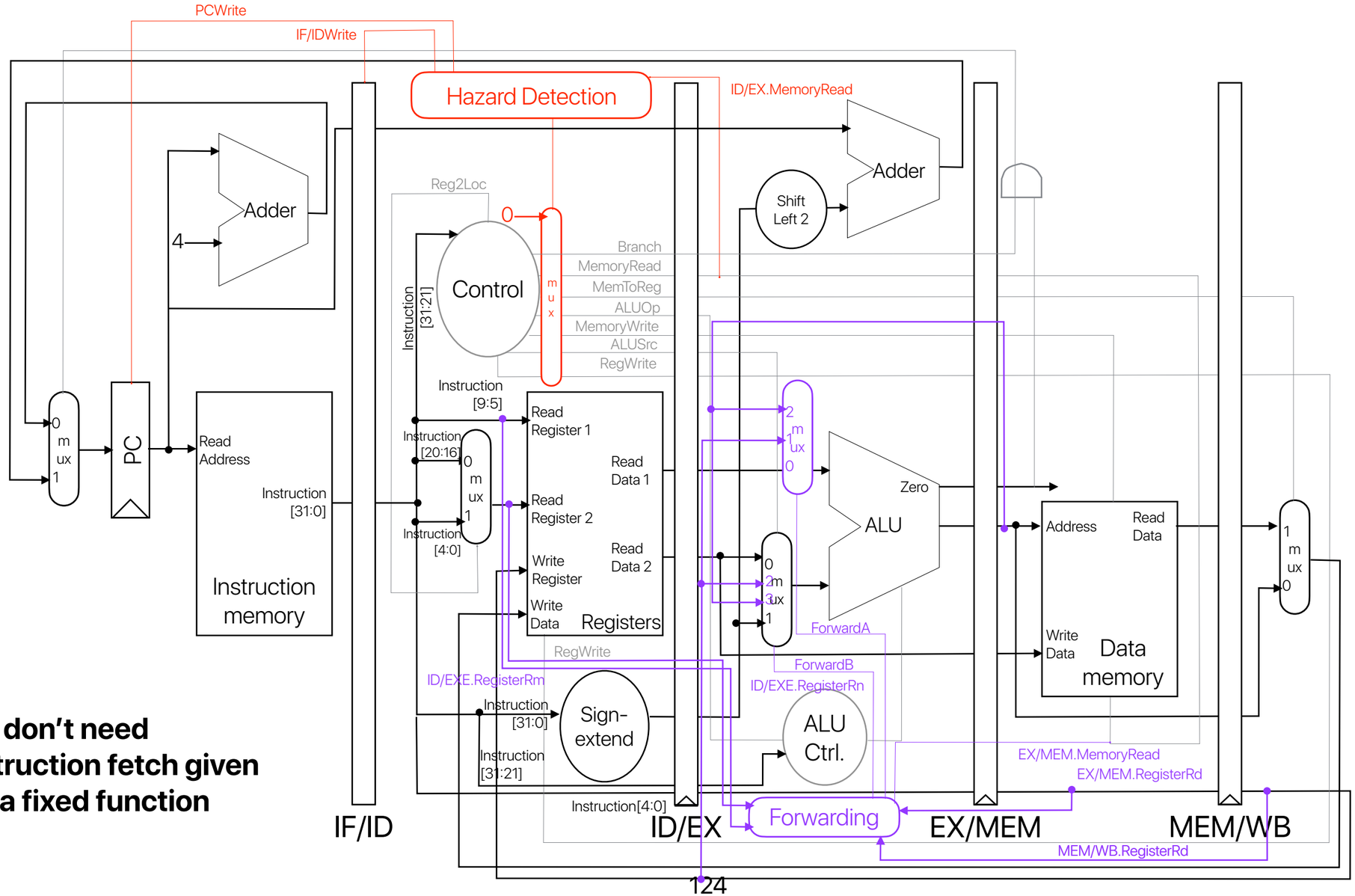
```
ld    X1, 0(X0)
ld    X2, 8(X0)
add   X3, X31, #20
mul   X2, X2, X3
add   X1, X1, X2
sd    X1, 0(X0)
```



This is what you need for these instructions



Specialize the circuit

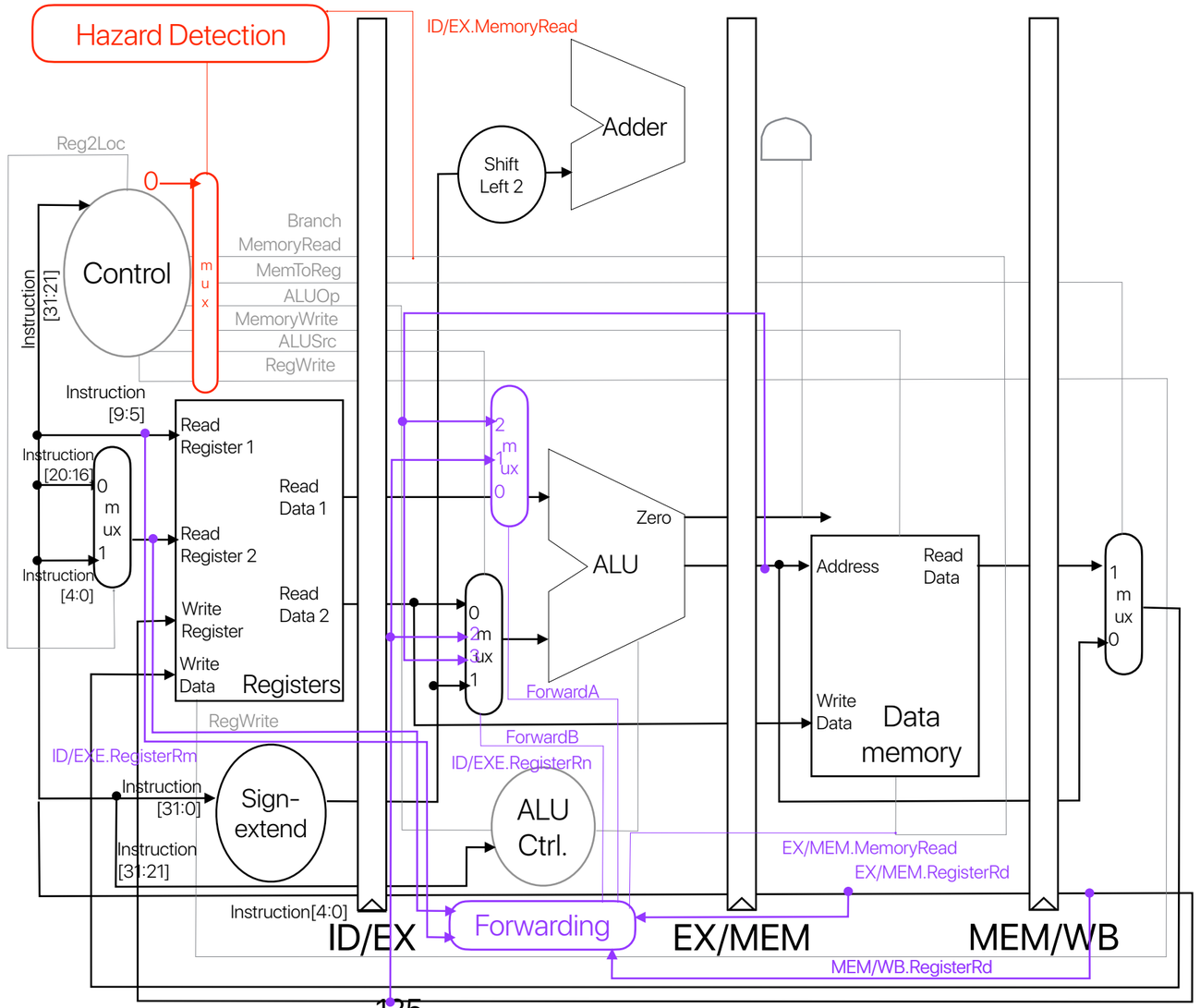


We don't need instruction fetch given it's a fixed function

Specialize the circuit

We don’t need these many registers, complex control, decode

We don’t need instruction fetch given it’s a fixed function

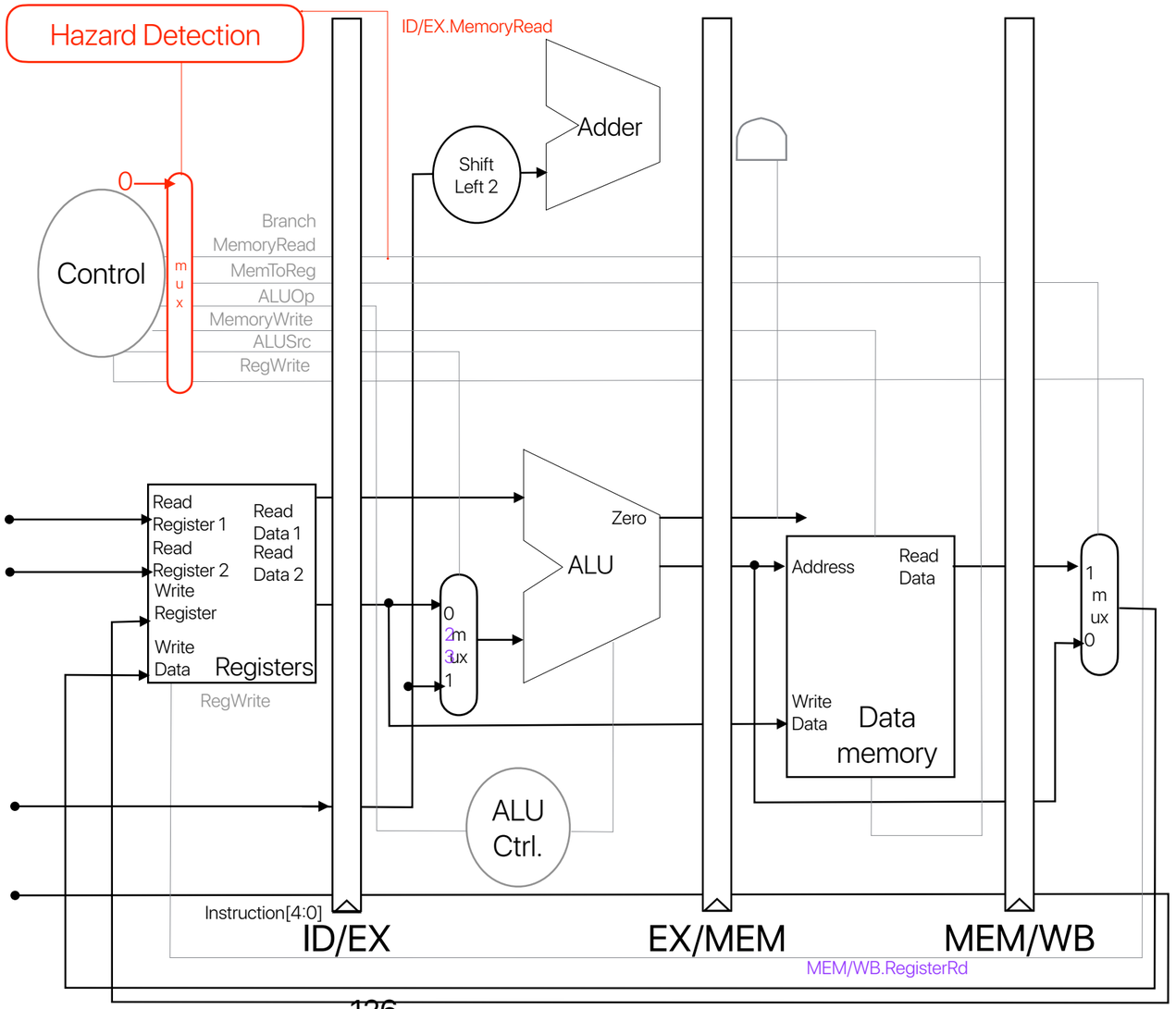


Specialize the circuit

We don't need ALUs, branches, hazard detections…

We don't need these many registers, complex control, decode

We don't need instruction fetch given it's a fixed function

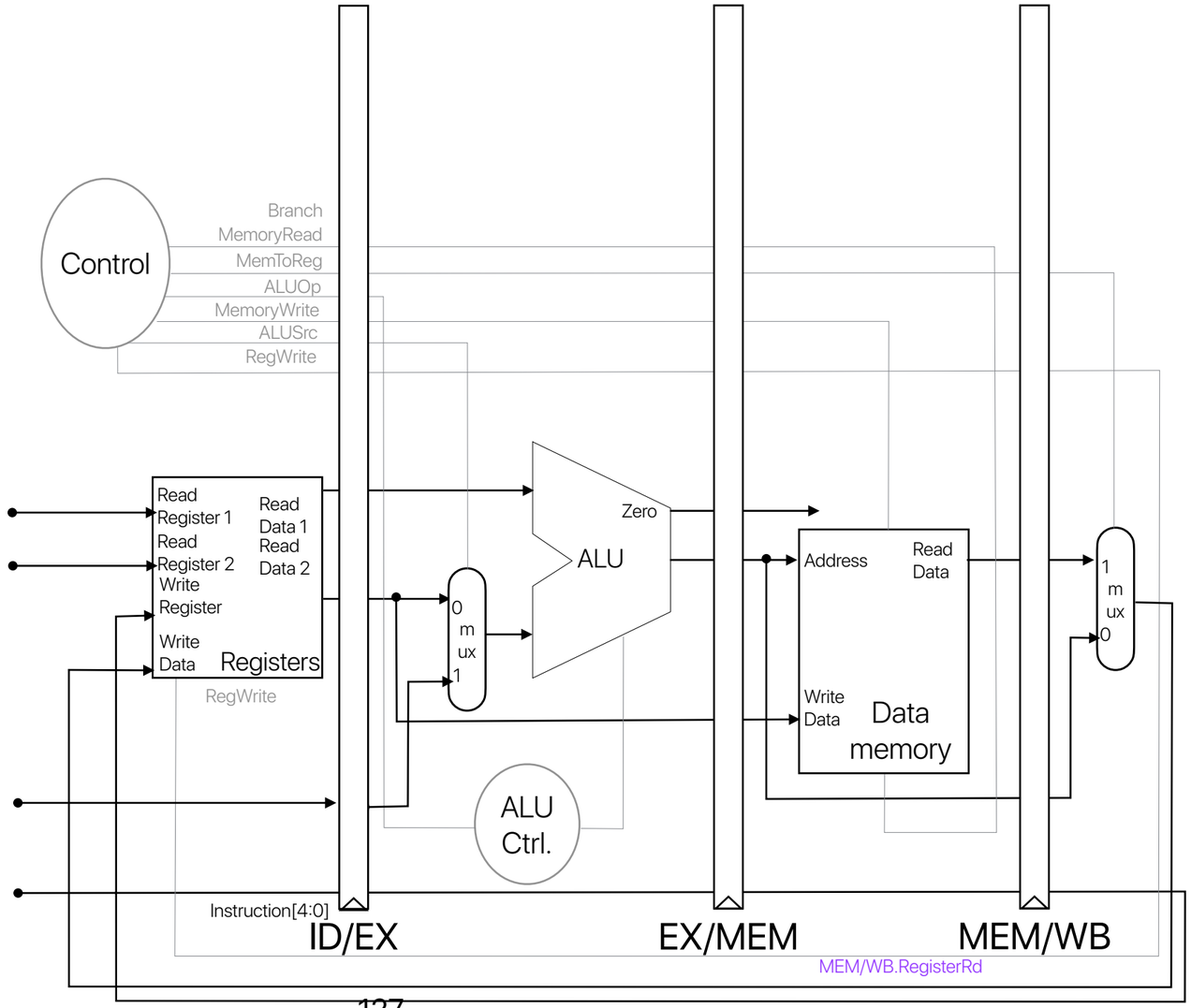


Specialize the circuit

We don’t need big ALUs, branches, hazard detections…

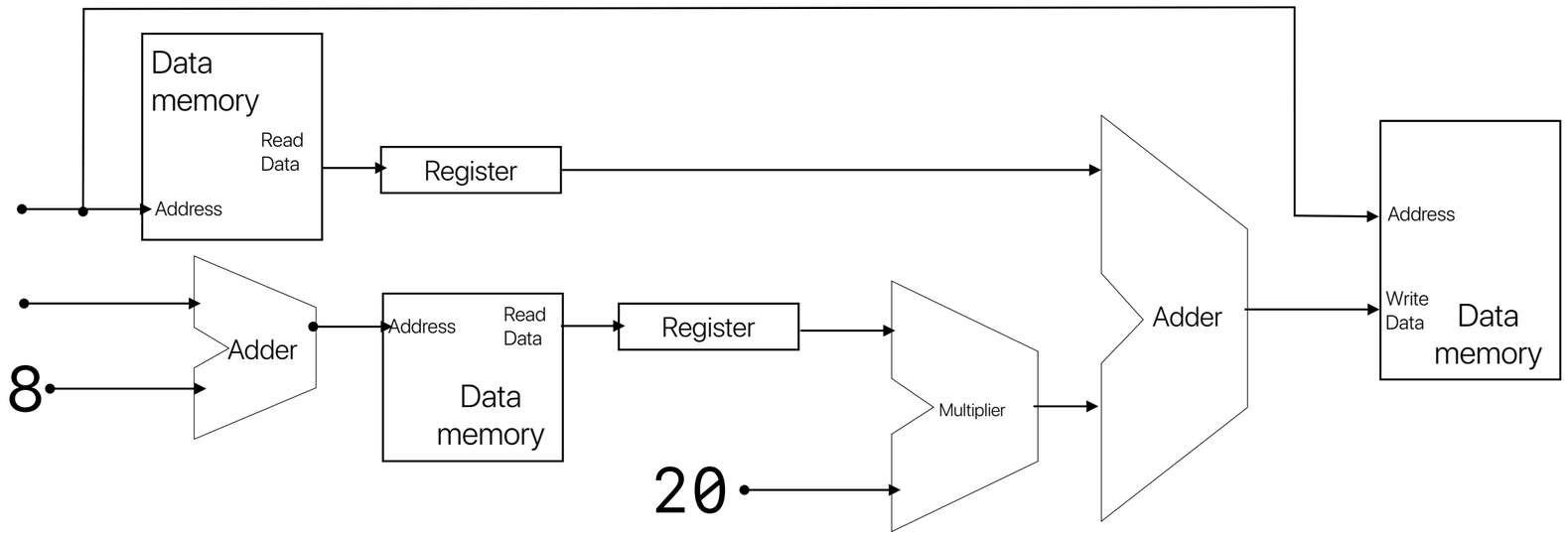
We don’t need these many registers, complex control, decode

We don’t need instruction fetch given it’s a fixed function



Rearranging the datapath

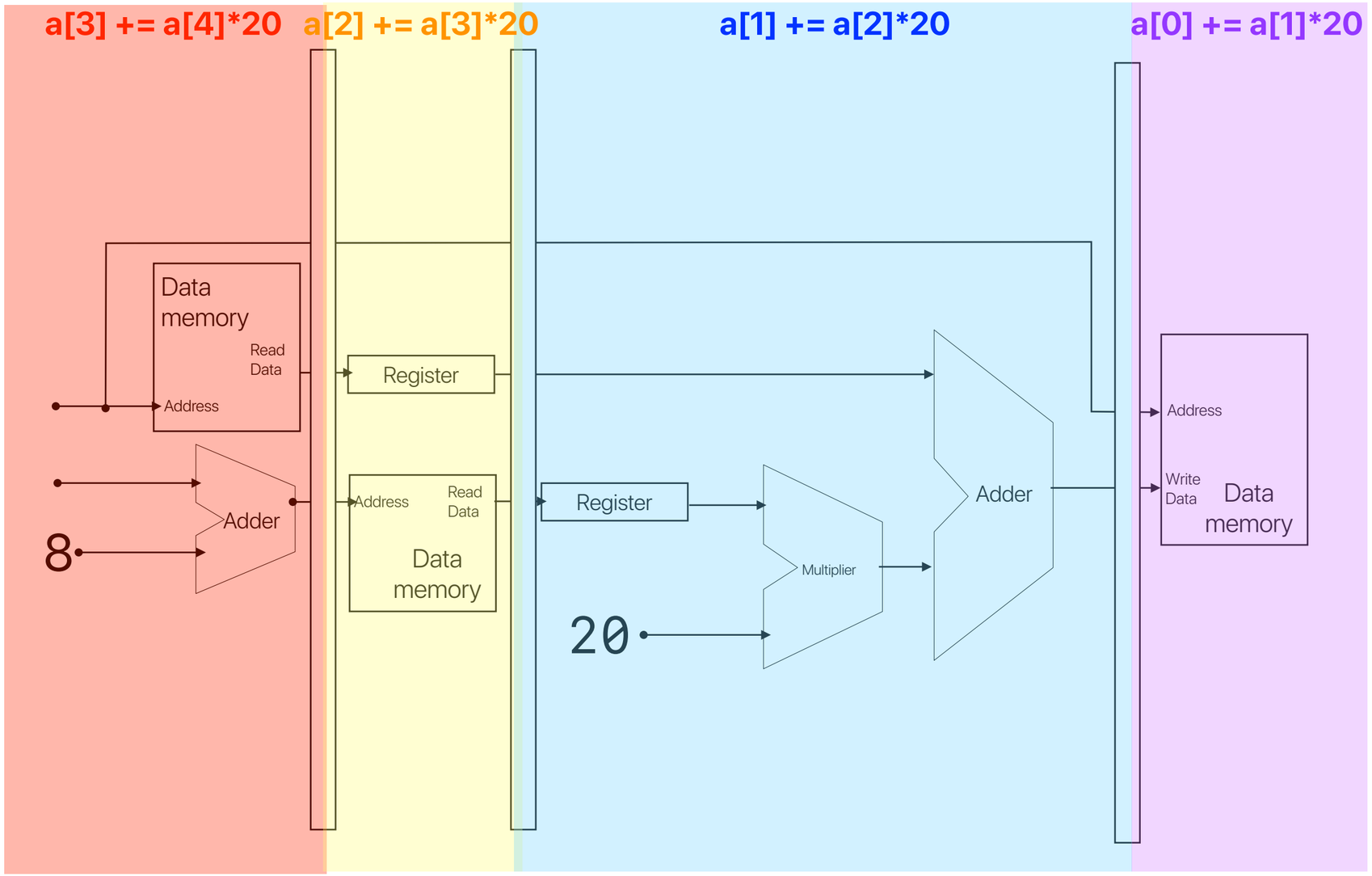
```
ld    X1, 0(X0)
ld    X2, 8(X0)
add   X3, X31, #20
mul   X2, X2, X3
add   X1, X1, X2
sd    X1, 0(X0)
```



The pipeline for a[i] += a[i+1]*20

Each stage can still be as fast as the pipelined processor

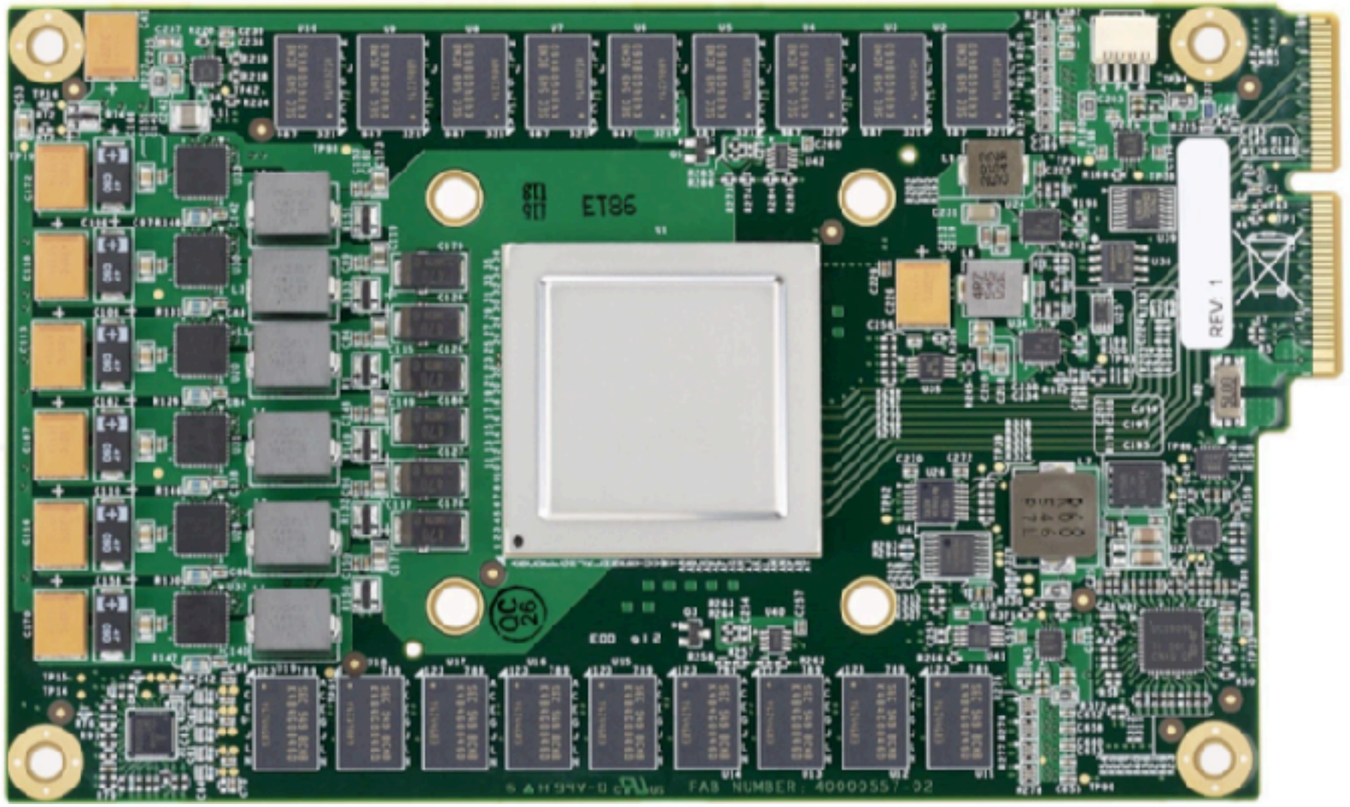
But each stage is now working on what the original 6 instructions would do



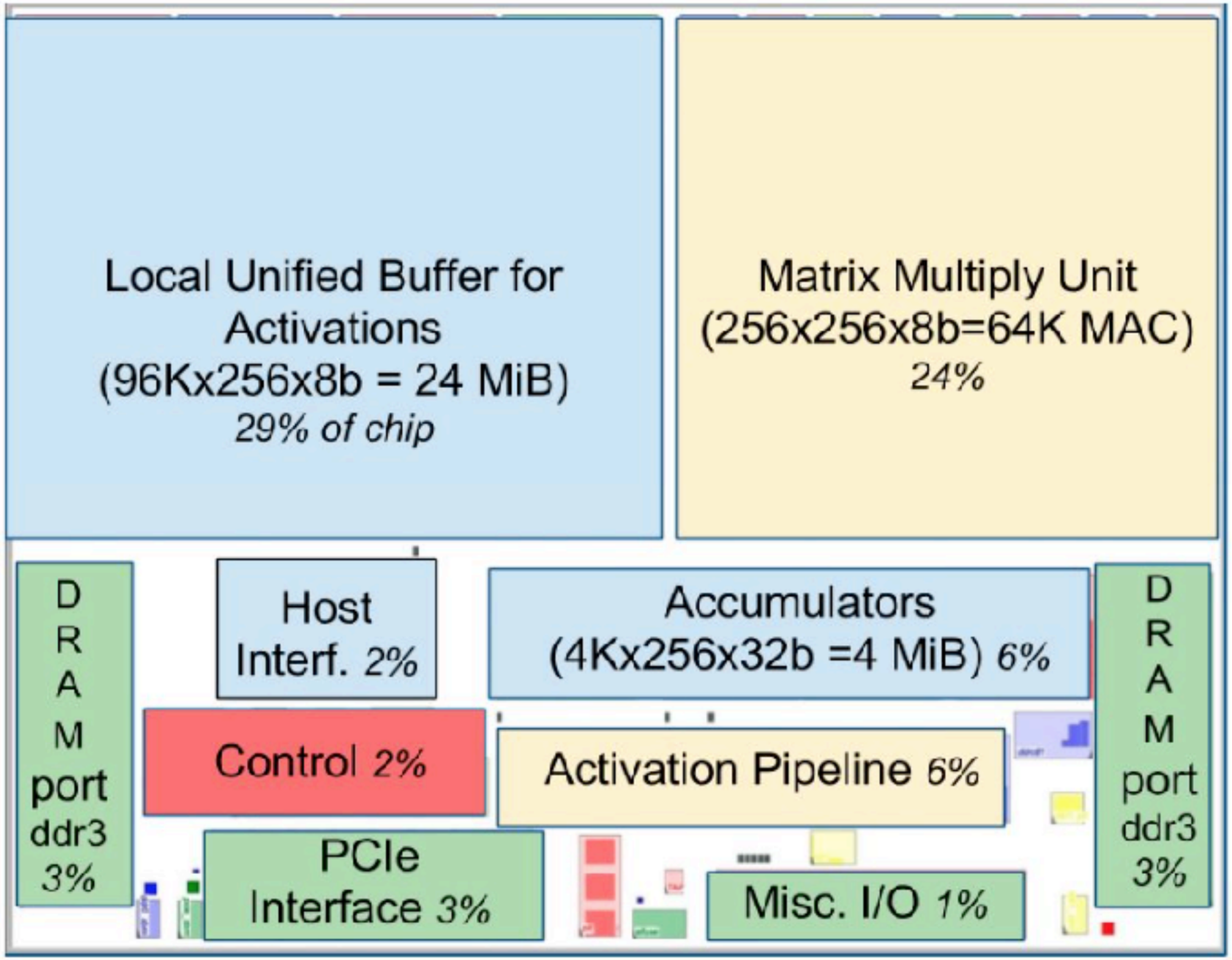
In-Datacenter Performance Analysis of a Tensor Processing Unit

N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Na- garajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Wal- ter, W. Wang, E. Wilcox, and D. H. Yoon
Google Inc.

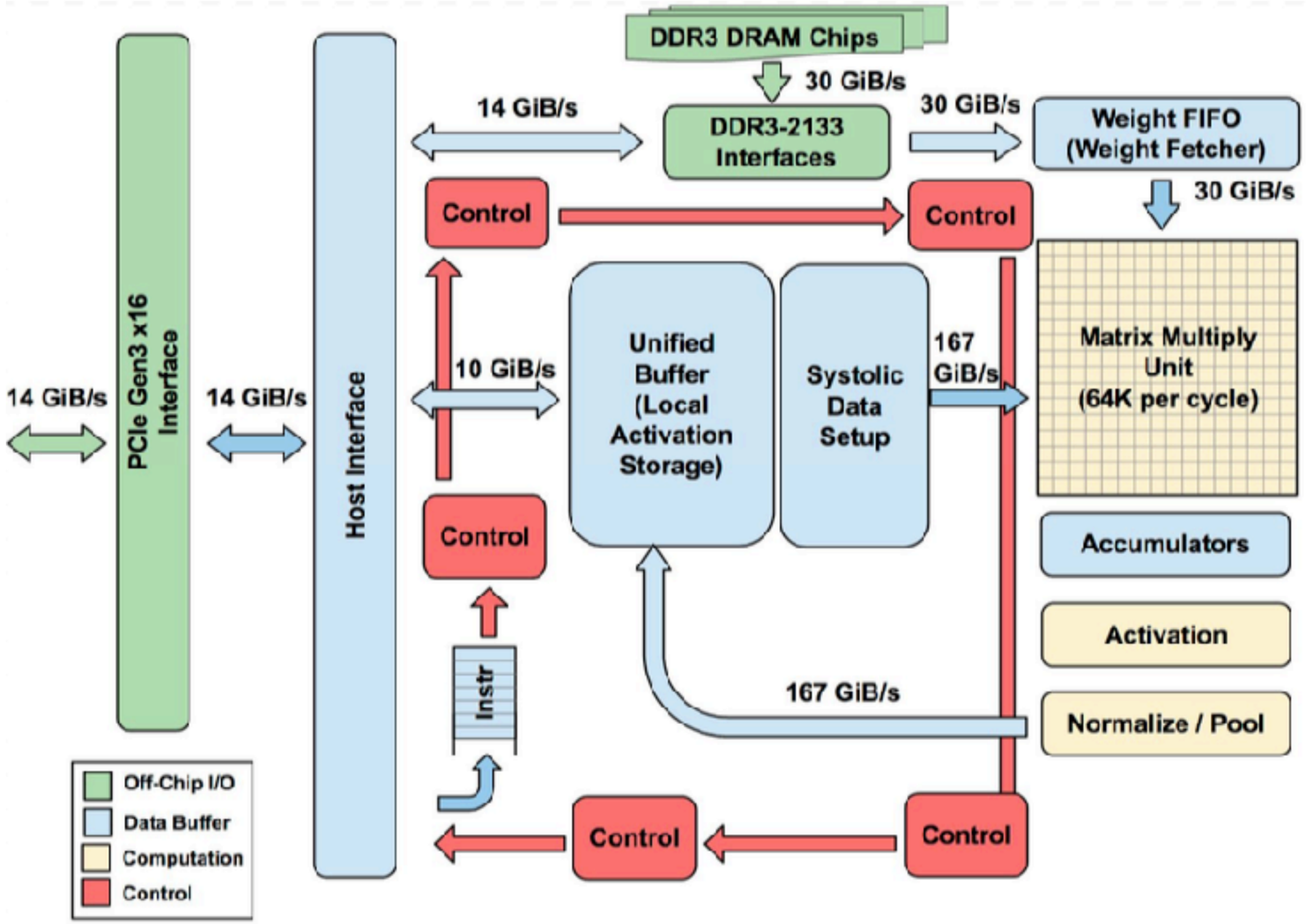
What TPU looks like



TPU Floorplan



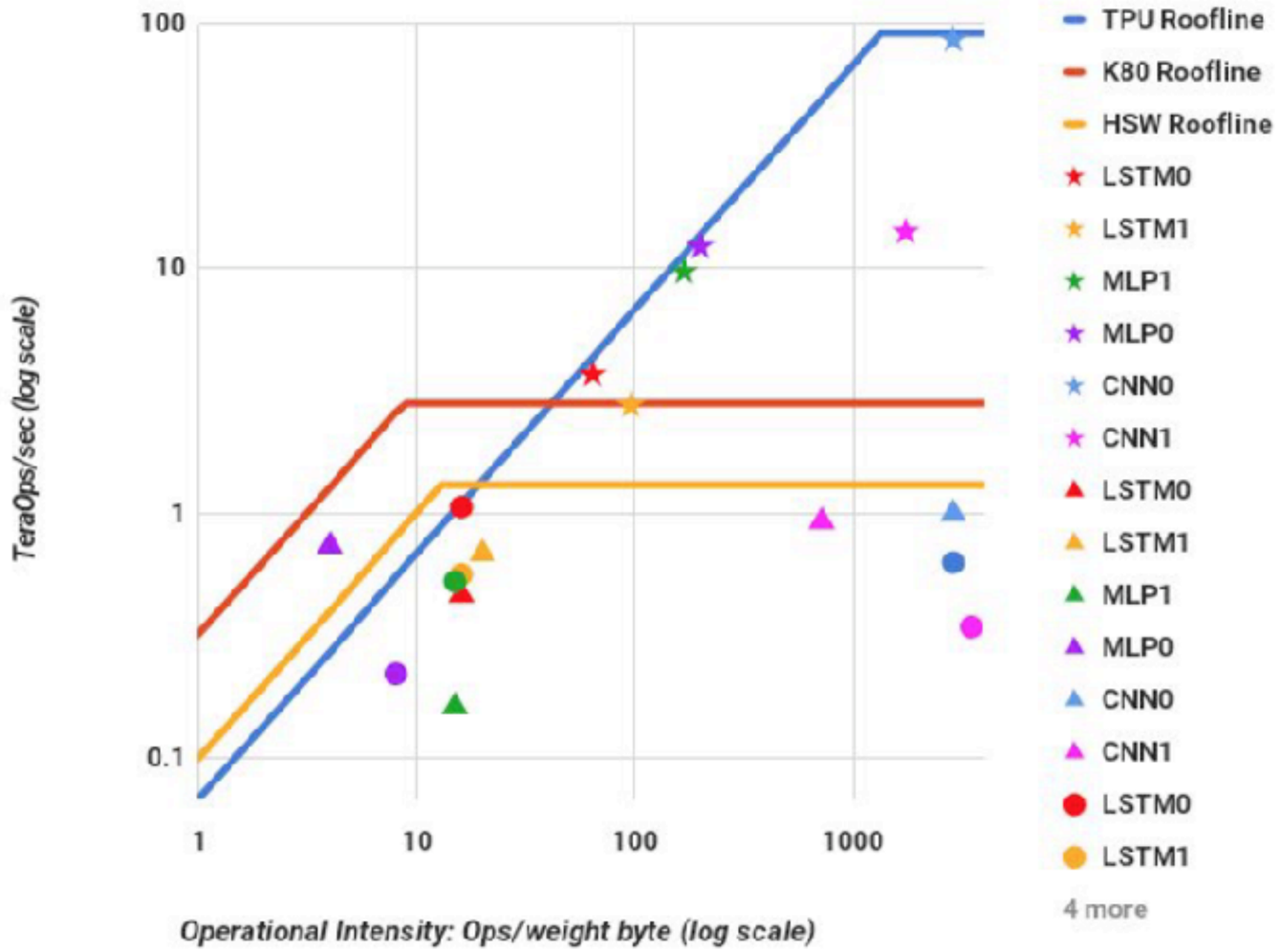
TPU Block diagram



Experimental setup

Model	Die										Benchmarked Servers				
	mm^2	nm	MHz	TDP	Measured		TOPS/s		GB/s	On-Chip Memory	Dies	DRAM Size	TDP	Measured	
					Idle	Busy	8b	FP						Idle	Busy
Haswell E5-2699 v3	662	22	2300	145W	41W	145W	2.6	1.3	51	51 MiB	2	256 GiB	504W	159W	455W
NVIDIA K80 (2 dies/card)	561	28	560	150W	25W	98W	--	2.8	160	8 MiB	8	256 GiB (host) + 12 GiB x 8	1838W	357W	991W
TPU	NA*	28	700	75W	28W	40W	92	--	34	28 MiB	4	256 GiB (host) + 8 GiB x 4	861W	290W	384W

Performance/Rooflines

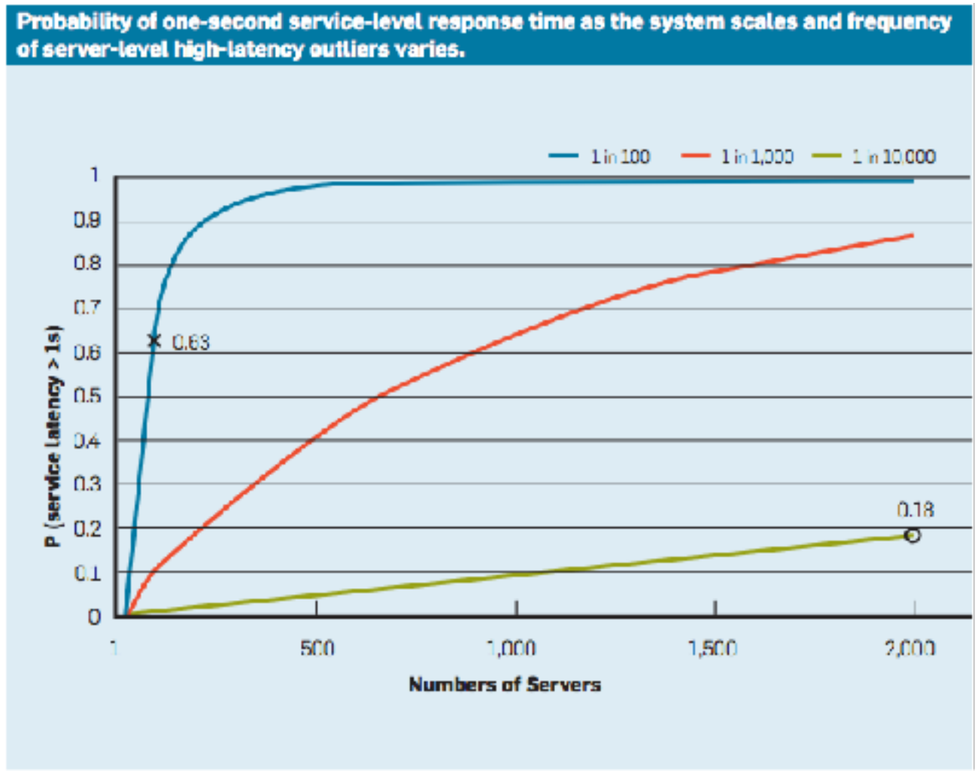


Tail latencies

Type	*Batch*	*99th% Response*	*Inf/s (IPS)*	*% Max IPS*
CPU	16	7.2 ms	5,482	42%
CPU	64	21.3 ms	13,194	100%
GPU	16	6.7 ms	13,461	37%
GPU	64	8.3 ms	36,465	100%
TPU	200	7.0 ms	225,000	80%
TPU	250	10.0 ms	280,000	100%

Table 4. 99-th% response time and per die throughput (IPS) for MLP0 as batch size varies for MLP0. The longest allowable latency is 7 ms. For the GPU and TPU, the maximum MLP0 throughput is limited by the host server overhead. Larger batch sizes increase throughput, but as the text explains, their longer response times exceed the limit, so CPUs and GPUs must use less-efficient, smaller batch sizes (16 vs. 200).

Tail latencies



- Tail Latency == 1 in X servers being slow
- Why is this bad? – Each user request now needs several servers – Changes of experience tail is much higher
- If 99% of the server's response time is 10ms, but 1% of them take 1 second to response
  - If the user only needs one, the mean is OK
  - If the user needs 100 partitions from 100 servers, 63% of the requests takes more than 1 seconds.

Tail latency

Type	*Batch*	*99th% Response*	*Inf/s (IPS)*	*% Max IPS*
CPU	16	7.2 ms	5,482	42%
CPU	64	21.3 ms	13,194	100%
GPU	16	6.7 ms	13,461	37%
GPU	64	8.3 ms	36,465	100%
TPU	200	7.0 ms	225,000	80%
TPU	250	10.0 ms	280,000	100%

Table 4. 99-th% response time and per die throughput (IPS) for MLP0 as batch size varies for MLP0. The longest allowable latency is 7 ms. For the GPU and TPU, the maximum MLP0 throughput is limited by the host server overhead. Larger batch sizes increase throughput, but as the text explains, their longer response times exceed the limit, so CPUs and GPUs must use less-efficient, smaller batch sizes (16 vs. 200).

What NVIDIA says

https://blogs.nvidia.com/blog/2017/04/10/ai-drives-rise-accelerated-computing-datacenter/

	K80 2012	TPU 2015	P40 2016
Inferences/Sec <10ms latency	1/13X	1X	2X
Training TOPS	6 FP32	NA	12 FP32
Inference TOPS	6 FP32	90 INT8	48 INT8
On-chip Memory	16 MB	24 MB	11 MB
Power	300W	75W	250W
Bandwidth	320 GB/S		

While Google and NVIDIA chose different development paths, there were several themes common to both our approaches. Specifically:

- AI requires accelerated computing. Accelerators provide the significant data processing necessary to keep up with the growing demands of deep learning in an era when Moore's law is slowing.
- Tensor processing is at the core of delivering performance for deep learning training and inference.
- Tensor processing is a major new workload enterprises must consider when building modern data centers.
- Accelerating tensor processing can dramatically reduce the cost of building modern data centers.

7.10 Fallacies and Pitfalls

In these early days of both DSAs and DNNs, fallacies abound.

Fallacy *It costs $100 million to design a custom chip.*

Figure 7.51 shows a graph from an article that debunks the widely quoted $100-million myth that it was "only" $50 million, with most of the cost being salaries (Olofsson, 2011). Note that the author's estimate is for sophisticated processors that include features that DSAs by definition omit, so even if there were no improvement to the development process, you would expect the cost of a DSA design to be less.

Why are we more optimistic six years later, when, if anything, mask costs are even higher for the smaller process technologies?

First, software is the largest category, at almost a third of the cost. The availability of applications written in domain-specific languages allows the compilers to do most of the work of porting the applications to your DSA, as we saw for the TPU and Pixel Visual Core. The open RISC-V instruction set will also help reduce the cost of getting system software as well as cut the large IP costs.

Mask and fabrication costs can be saved by having multiple projects share a single reticle. As long as you have a small chip, amazingly enough, for $30,000 anyone can get 100 untested parts in 28-nm TSMC technology (Patterson and Nikolić, 2015).

Fallacies & Pitfalls

- Fallacy: NN inference applications in data centers value throughput as much as response time.
- Fallacy: The K80 GPU architecture is a good match to NN inference — GPU is throughput oriented
- Pitfall: For NN hardware, Inferences Per Second (IPS) is an inaccurate summary performance metric — it's simply the inverse of the complexity of the typical inference in the application (e.g., the number, size, and type of NN layers)
- Fallacy: The K80 GPU results would be much better if Boost mode were enabled — Boost mode increased the clock rate by a factor of up to 1.6—from 560 to 875 MHz—which increased performance by 1.4X, but it also raised power by 1.3X. The net gain in performance/Watt is 1.1X, and thus Boost mode would have a minor impact on LSTM1
- Fallacy: CPU and GPU results would be comparable to the TPU if we used them more efficiently or compared to newer versions.

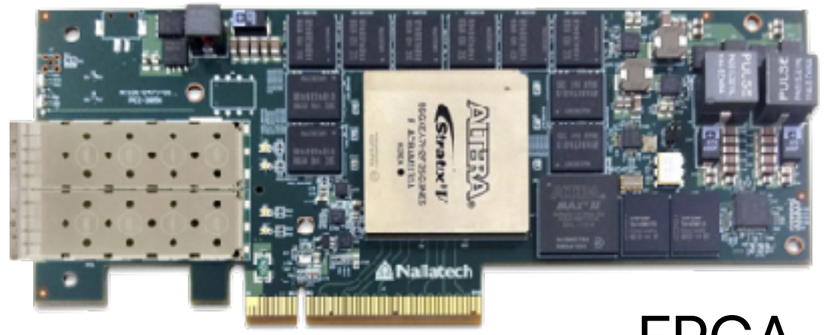
Fallacies & Pitfalls

- Pitfall: Architects have neglected important NN tasks.
  - CNNs constitute only about 5% of the representative NN workload for Google. More attention should be paid to MLPs and LSTMs. Repeating history, it's similar to when many architects concentrated on floating- point performance when most mainstream workloads turned out to be dominated by integer operations.
- Pitfall: Performance counters added as an afterthought for NN hardware.
- Fallacy: After two years of software tuning, the only path left to increase TPU performance is hardware upgrades.
- Pitfall: Being ignorant of architecture history when designing a domain-specific architecture
  - Systolic arrays
  - Decoupled-access/execute
  - CISC instructions

A Cloud-Scale Acceleration Architecture

Adrian Caulfield, Eric Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, Doug Burger
Microsoft

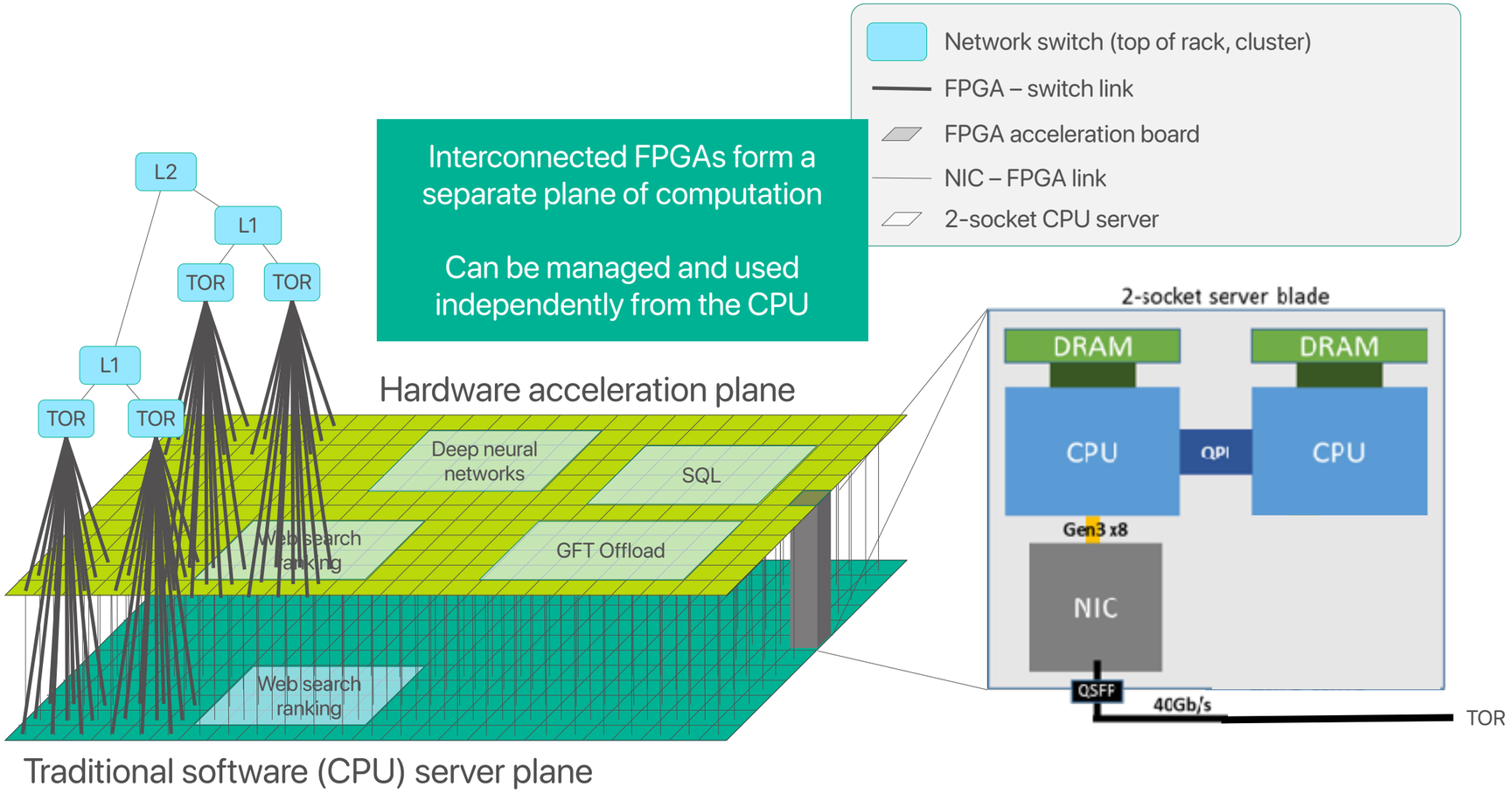
FPGA



FPGA

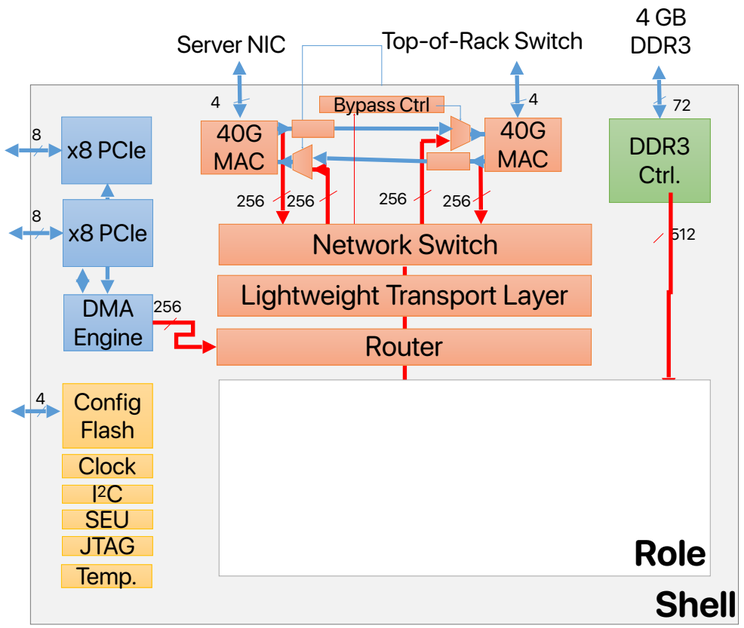
- Field Programmable Gate Array
  - An array of “Lookup tables (LUTs)”
  - Reconfigurable wires or say interconnects of LUTs
  - Registers
- An LUT
  - Accepts a few inputs
  - Has SRAM memory cells that store all possible outputs
  - Generates outputs according to the given inputs
- As a result, you may use FPGAs to emulate any kind of gates or logic combinations, and create an ASIC-like processor

Configurable cloud



Gen2 shell

- Foundation for all accelerators
  - Includes PCIe, Networking and DDR IP
  - Common, well tested platform for development
- Lightweight Transport Layer
  - Reliable FPGA-to-FPGA Networking
  - Ack/Nack protocol, retransmit buffers
  - Optimized for lossless network
  - Minimized resource usage

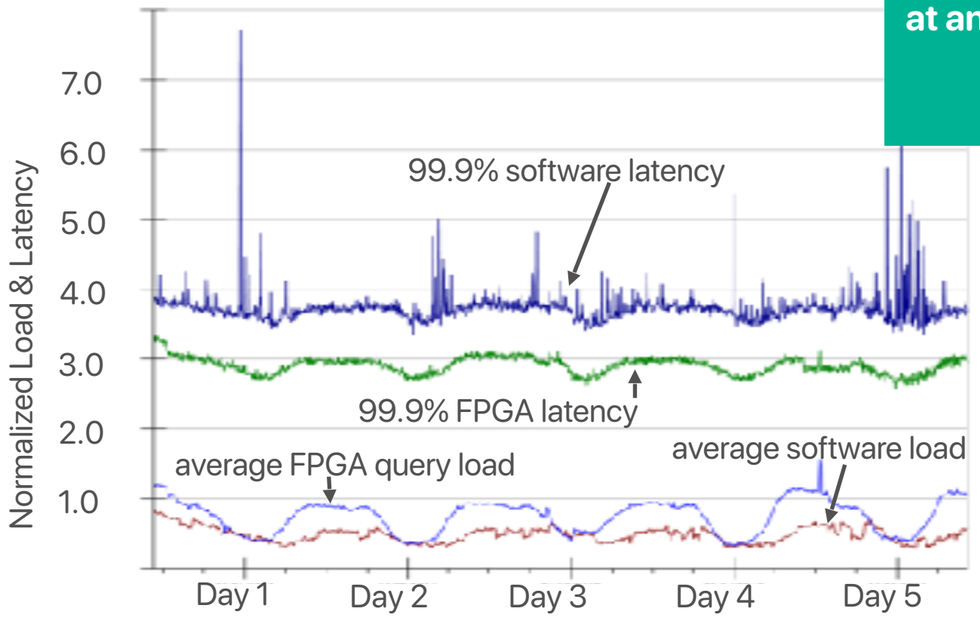


Use cases

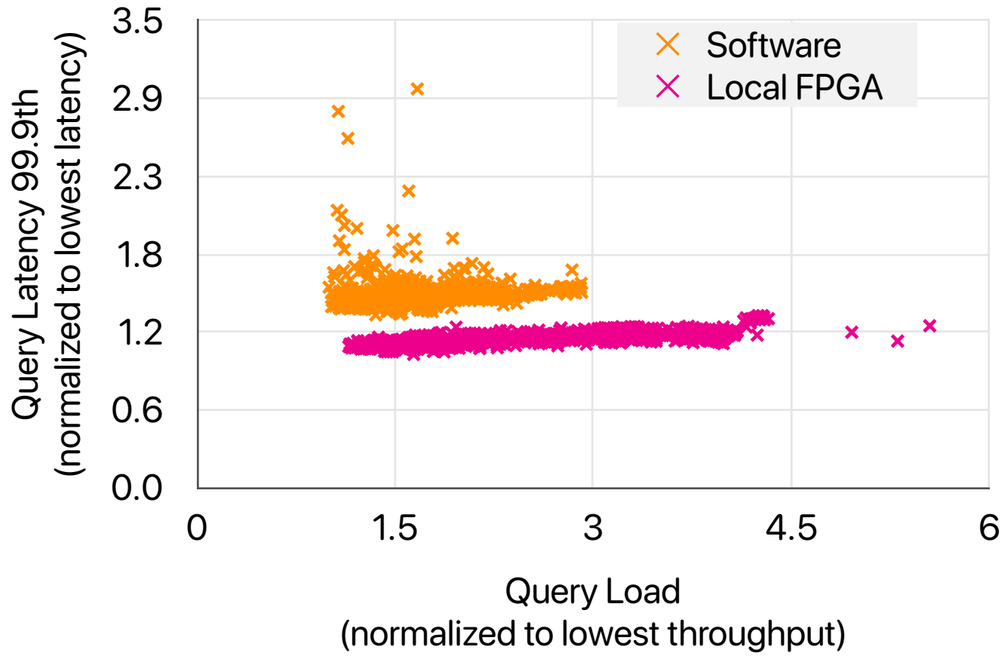
- Local: Great service acceleration
- Infrastructure: Fastest cloud network
- Remote: Reconfigurable app fabric (DNNs)

5 day bed-level latency

- Lower & more consistent 99.9th tail latency
- In production for years



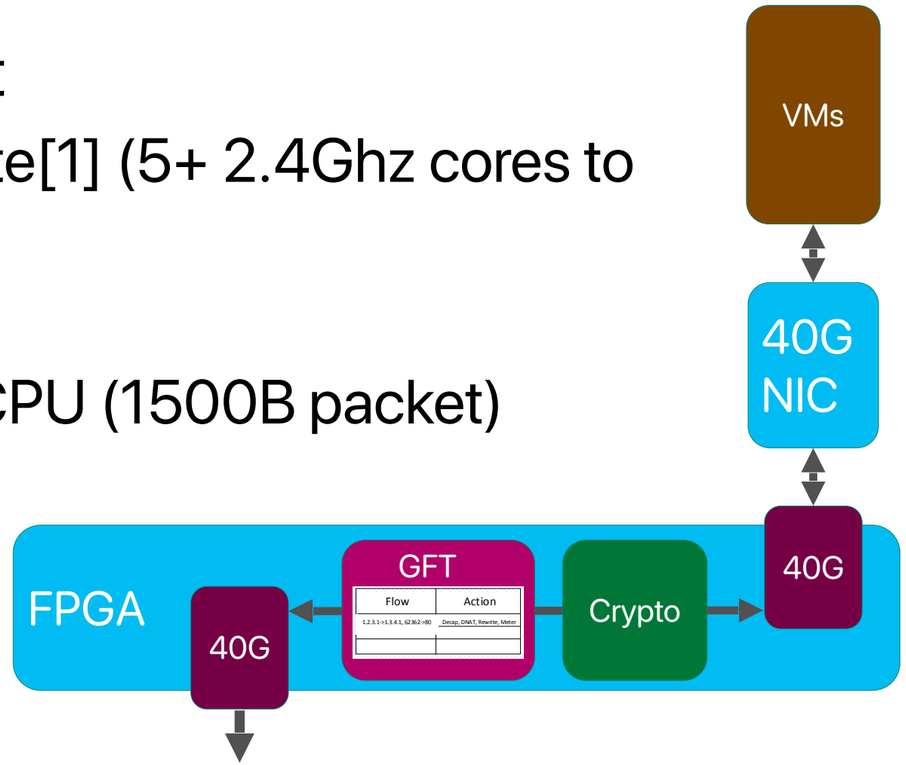
Even at 2× query load, accelerated ranking has lower latency than software **at any load**



Accelerated networking

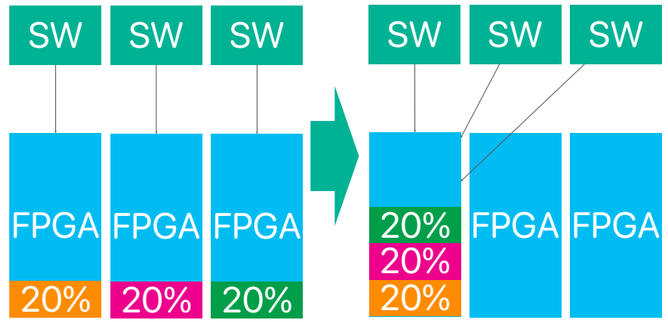
- Software defined networking
  - Generic Flow Table (GFT) rule based packet rewriting
  - 10x latency reduction vs software, CPU load now <1 core
  - 25Gb/s throughput at 25µs latency – the fastest cloud network

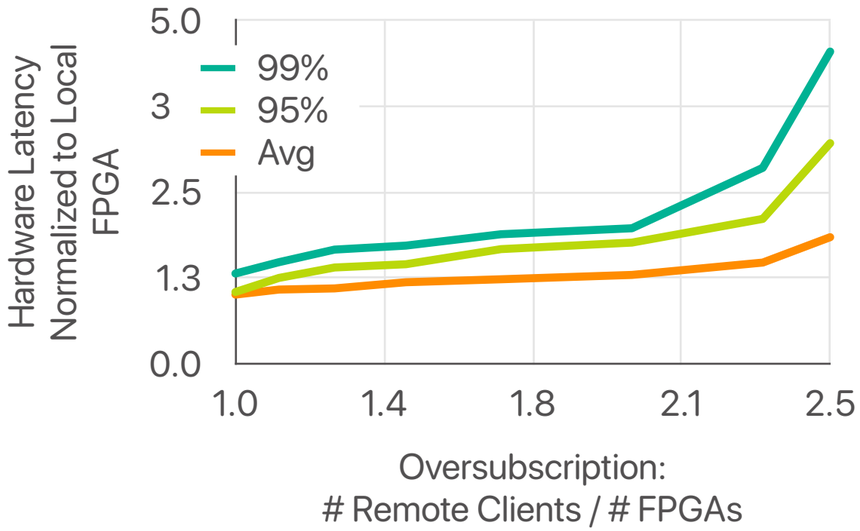
- Capable of 40 Gb line rate encrypt and decrypt
  - On Haswell, AES GCM-128 costs 1.26 cycles/byte[1] (5+ 2.4Ghz cores to sustain 40Gb/s)
  - CBC and other algorithms are more expensive
  - AES CBC-128-SHA1 is 11µs in FPGA vs 4µs on CPU (1500B packet)
  - Higher latency, but significant CPU savings



Shared DNN

- Economics: consolidation
  - Most accelerators have more throughput than a single host requires
  - Share excess capacity, use fewer instances
  - Frees up FPGAs for other use services
- DNN accelerator
  - Sustains 2.5x busy clients in microbenchmark, before queuing delay drives latency up





Why FPGAs?

- Which of the following is the **main** reason why Microsoft adopts FPGAs instead of the alternatives chosen by their rivals?
  A. Cost
  B. Performance
  C. Scalability
  D. Flexibility
  E. Easier to program

Why FPGA?

This model offers significant flexibility. From the local perspective, the FPGA is used as a compute or a network accelerator. From the global perspective, the FPGAs can be managed as a large-scale pool of resources, with acceleration

hyperscale infrastructure. The acceleration system we describe is sufficiently flexible to cover three scenarios: local compute acceleration (through PCIe), network acceleration, and global application acceleration, through configuration as pools of remotely accessible FPGAs. Local acceleration handles high-

These programmable architectures allow for hardware homogeneity while allowing fungibility via softwa[illegible]ifferent services. They must be highly *flexible* at the [illegible]stem [illegible]

Flexible

This paper described Configurable Clouds, a datacenter-scale acceleration architecture, based on FPGAs, that is both scalable and flexible. By putting in FPGA cards both in I/O

In addition to architectural requirements that provide sufficient flexibility to justify scale production deployment, there are also physical restrictions in current infrastructures that

Summary: What makes a configurable cloud?

- Local, infrastructure and remote acceleration
  - Gen1 showed significant gains even for complex services (~2x for Bing)
  - Needs to have clear benefit for majority of servers: infrastructure
- Economics must work
  - What works at small scale doesn’t always work at hyperscale and vice versa
  - Little tolerance for superfluous costs
  - Minimized complexity and risk in deployment and maintenance
- Must be flexible
  - Support simple, local accelerators and complex, shared accelerators at the same time
  - Rapid deployment of new protocols, algorithms and services across the cloud

Final words

Conclusion

- Computer architecture is now more important than you could ever imagine
- Being a “programmer” is easy. You need to know architecture a lot to be a “performance programmer”
  - Branch prediction
  - Cache
- Multicore era — to get your multithreaded program correct and perform well, you need to take care of coherence and consistency
- We’re now in the “dark silicon era”
  - Single-core isn’t getting any faster
  - Multi-core doesn’t scale anymore
  - We will see more and more ASICs
  - You need to write more “system-level” programs to use these new ASICs.