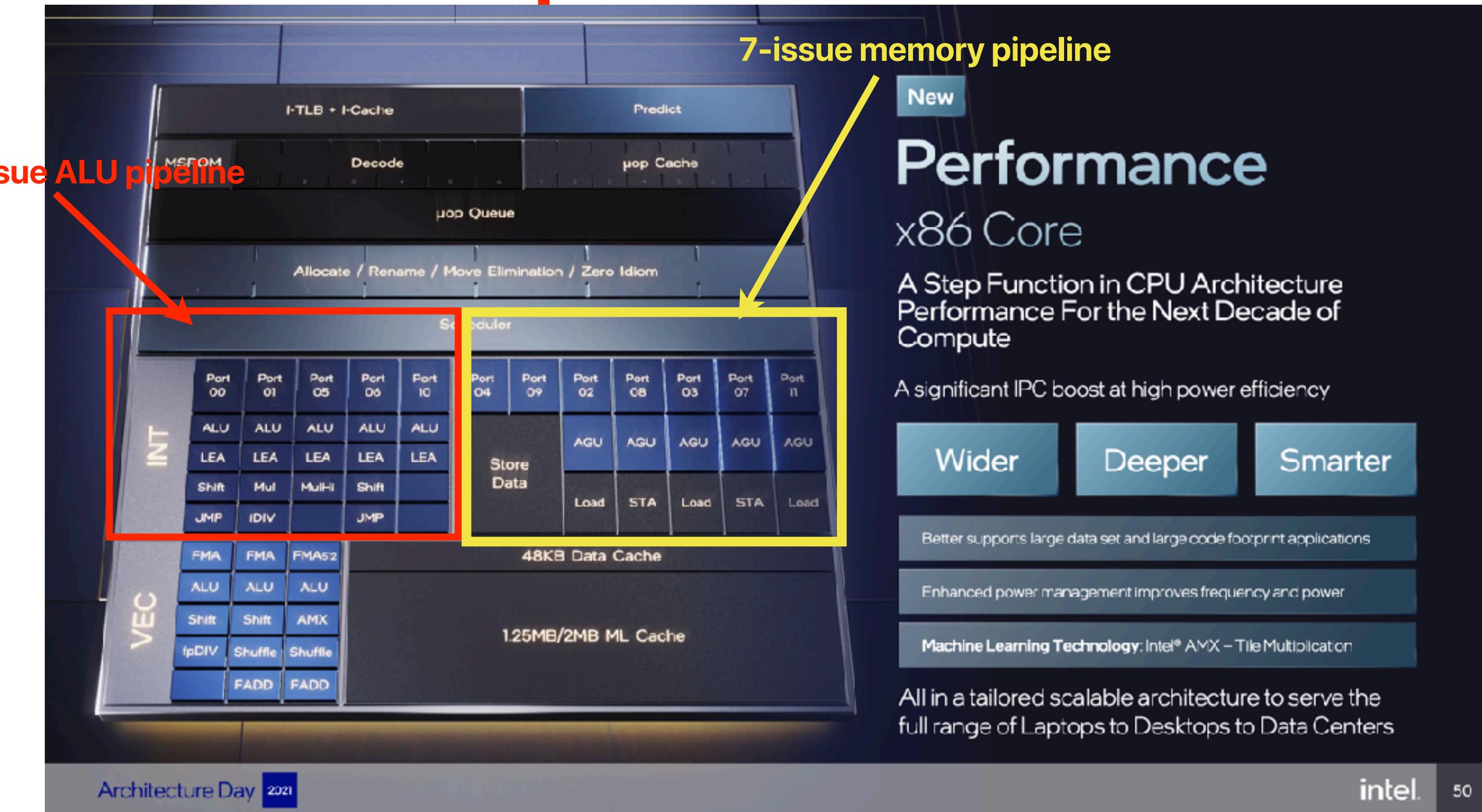


# **The “Golden Age” of Computer Architectures**

Hung-Wei Tseng

# Recap: Intel Alder Lake



# 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**?

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

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

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

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

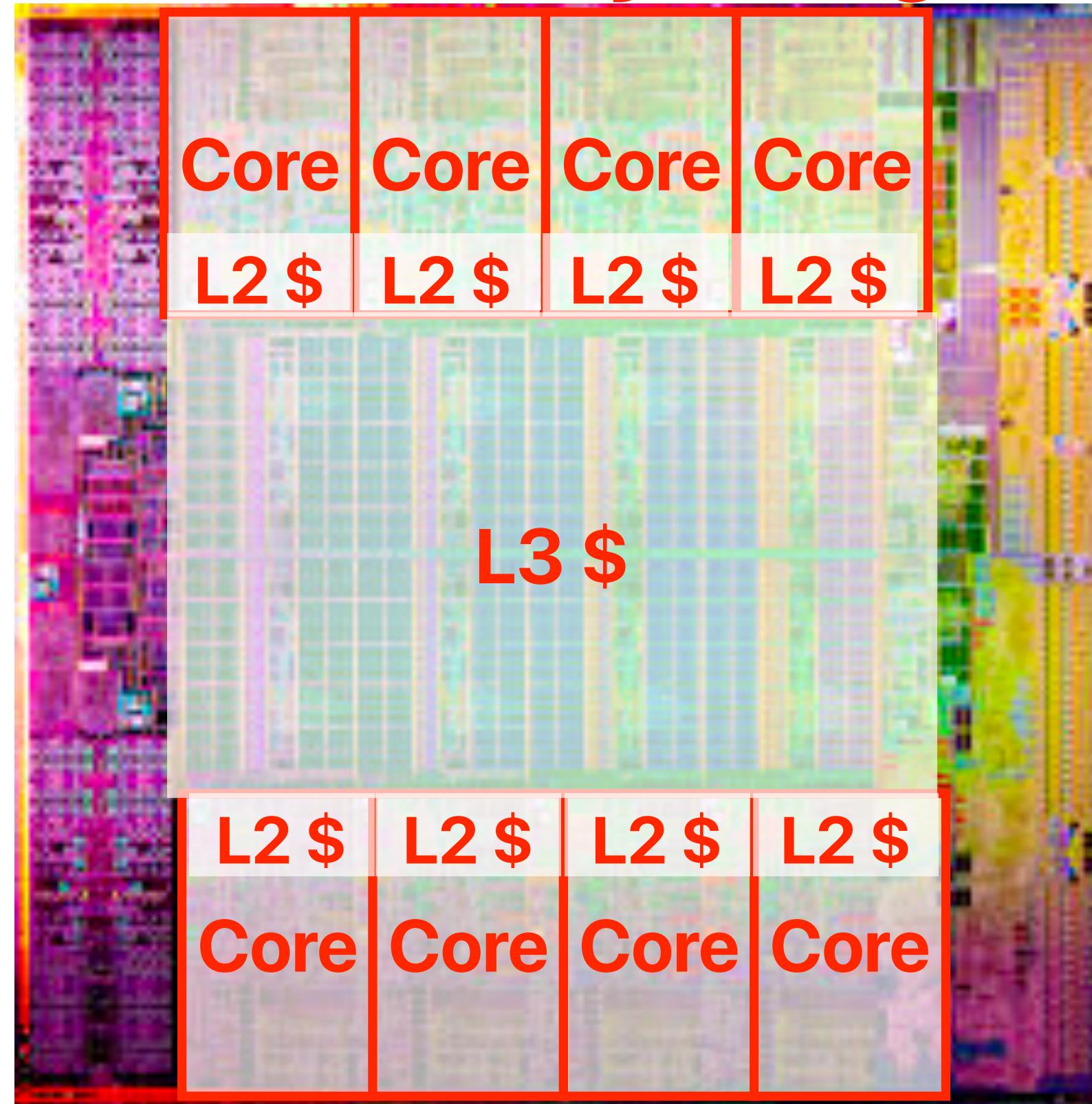
$$Energy_{2-core} = 2 \times P_{baseline} \times ET_{baseline} \times \frac{1}{1.43} = 1.39 \times Energy_{baseline}$$

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

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

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

# Intel Sandy Bridge



# Again — how many values are possible?

Thread 1

a=1;

x=b;

④ (1, 1) (1,1)

Thread 2

b=1;  
y=a;

```
#include <cs.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/conf.h>
#include <sys/thread.h>
#include <sys/thread.h>
#include <sys/thread.h>
#include <sys/thread.h>
```

```
volatile int x,y;
```

```
volatile int f;
```

```
void* modifya(void *z) {
```

```
    x=1;
```

```
    f=1;
```

```
    if(z!=NULL) {
```

```
        *(int *)z=x;
```

```
        *(int *)z=f;
```

```
    }
```

```
    return z;
```

```
}
```

Thread 1

a=1;

x=b;

(1,0)

Thread 1

a=1;  
x=b;

Thread 1

x=b;  
a=1;

OoO Scheduling!

(0,0)

Thread 2

b=1;  
y=a;

L);

L);

# Recap: Dark Silicon and the End of Multicore Scaling

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

=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

=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

=100W!

# 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

# Modern processor's frequency



Intel® Core™ i9-9900K Processor  
16M Cache, up to 5.00 GHz

Essentials	CPU Specifications
Product Collection	# of Cores <span style="color: blue;">?</span> 8
Code Name	# of Threads <span style="color: blue;">?</span> 16
Vertical Segment	Processor Base Frequency <span style="color: blue;">?</span> 3.60 GHz
Processor Number <span style="color: blue;">?</span>	Max Turbo Frequency <span style="color: blue;">?</span> 5.00 GHz
Status	Cache <span style="color: blue;">?</span> 16 MB Intel® Smart Cache
Launch Date <span style="color: blue;">?</span>	Bus Speed <span style="color: blue;">?</span> 8 GT/s
Lithography <span style="color: blue;">?</span>	Intel® Turbo Boost Technology 2.0 Frequency† <span style="color: blue;">?</span> 5.00 GHz
Included Items	TDP <span style="color: blue;">?</span> 95 W
Use Conditions <span style="color: blue;">?</span>	
Recommended Customer Price <span style="color: blue;">?</span>	

**Slower, but more**

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

**Rakesh Kumar, Keith I. Farkas\*, Norman P. Jouppi\*,  
Partha Sarathy Ranganathan\*, Dean M. Tullsen**

**UCSD and HP Labs\***

In the 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003.

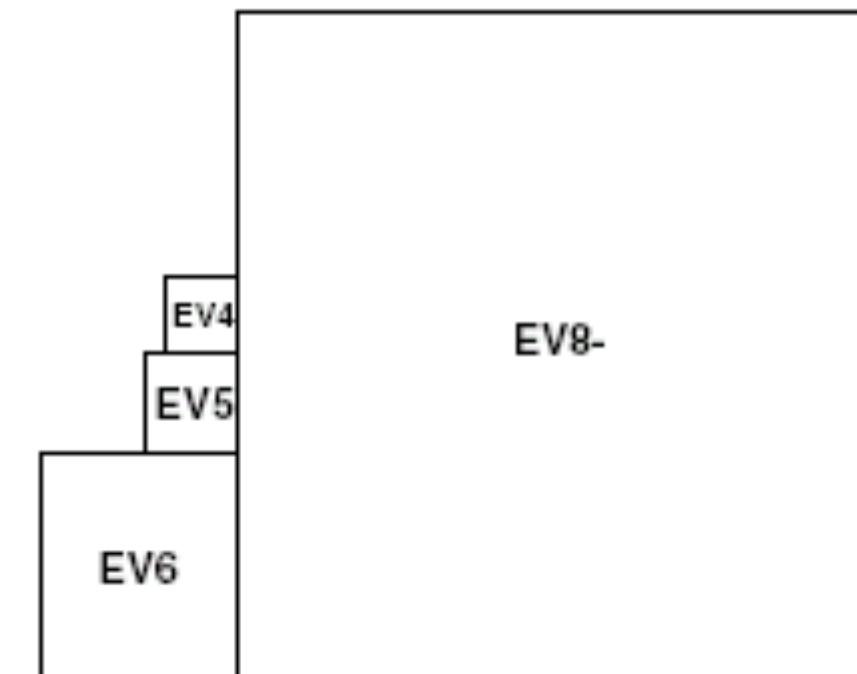
MICRO-36., 2003

MICRO Test-of-time award, 2021

# Areas of different processor generations

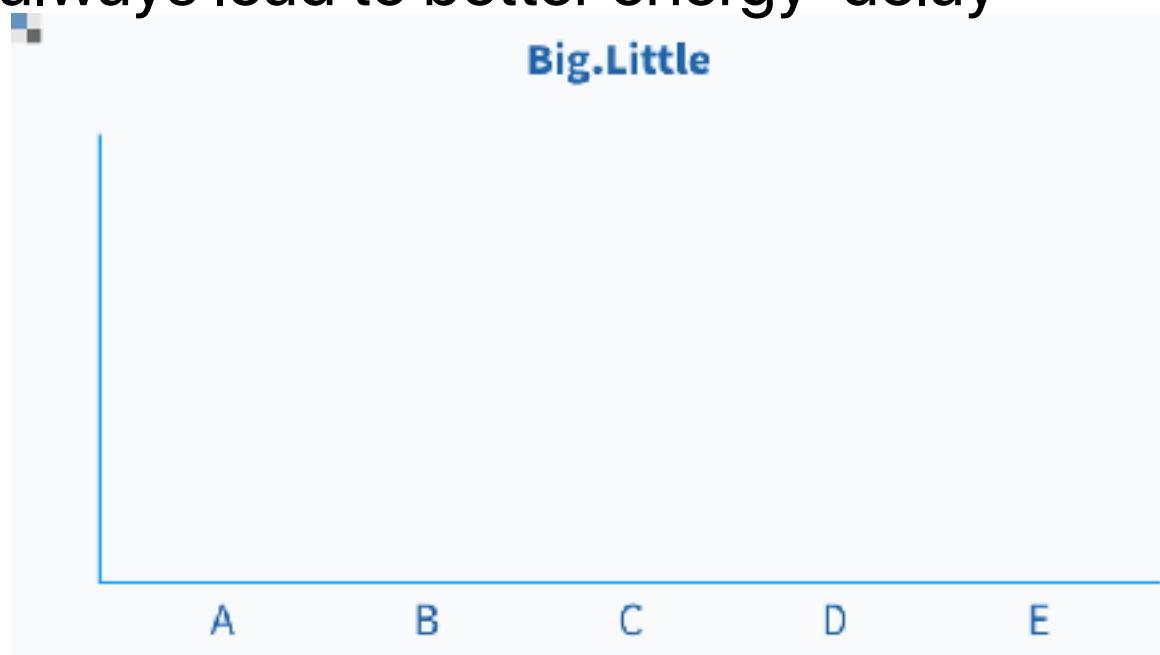
- You fit about 5 EV5 cores within the same area of an EV6
- If you build a quad-core EV6, you can use the same area to
  - build 20-core EV5
  - 3EV6+5EV5

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



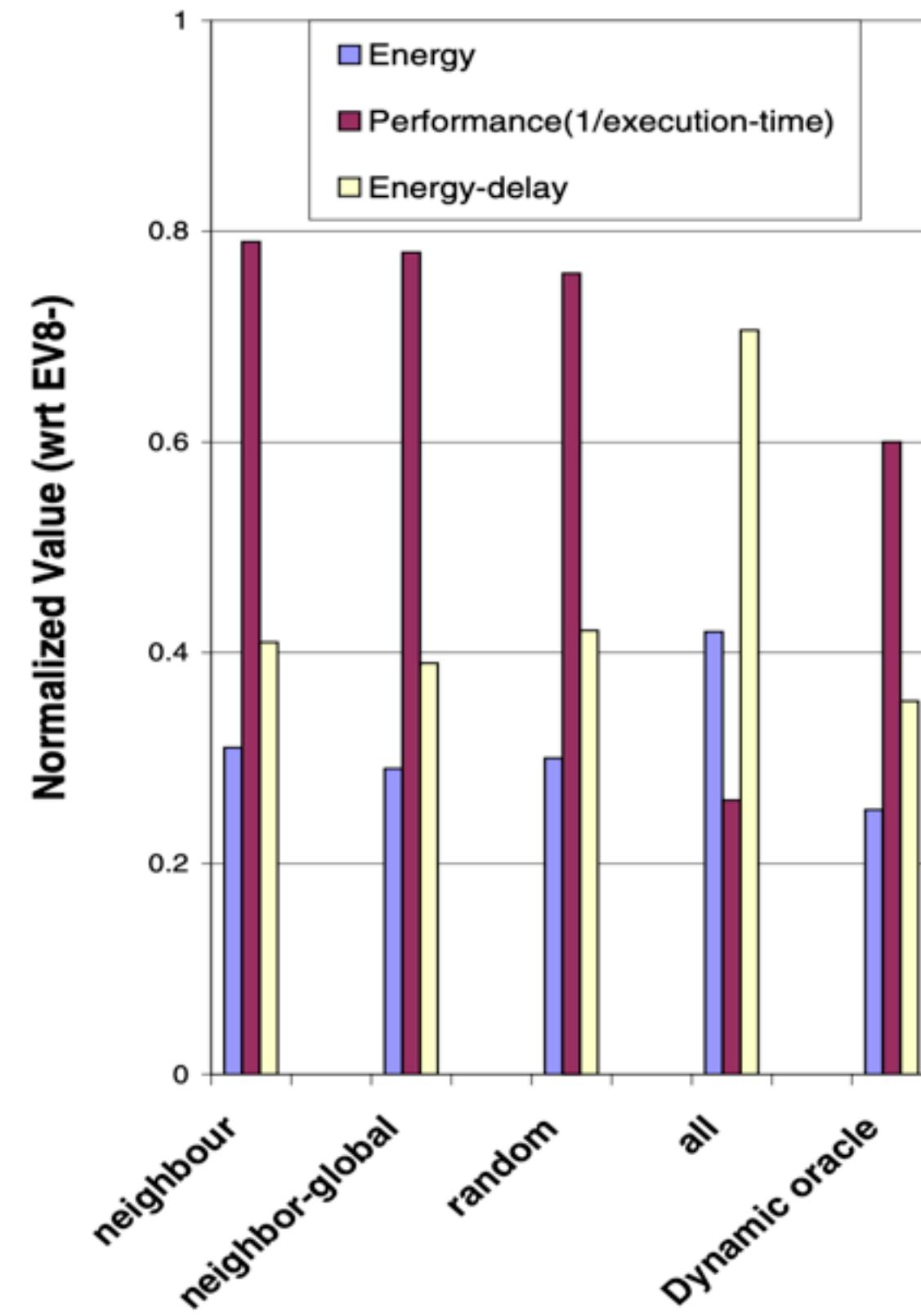
# Single ISA heterogeneous CMP

- Regarding "Single-ISA Heterogeneous Multi-Core Architectures", how many of the following statements is/are correct?
    - ① You need to recompile and optimize the binary for each core architecture to exploit the thread-level parallelism in this architecture
    - ② For a program with limited thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level of performance than homogeneous CMP
    - ③ For a program with rich thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level of performance than homogeneous CMP built with older-generation cores
    - ④ Spending more instructions on older-generation cores would always lead to better energy-delay
- A. 0  
B. 1  
C. 2  
D. 3  
E. 4



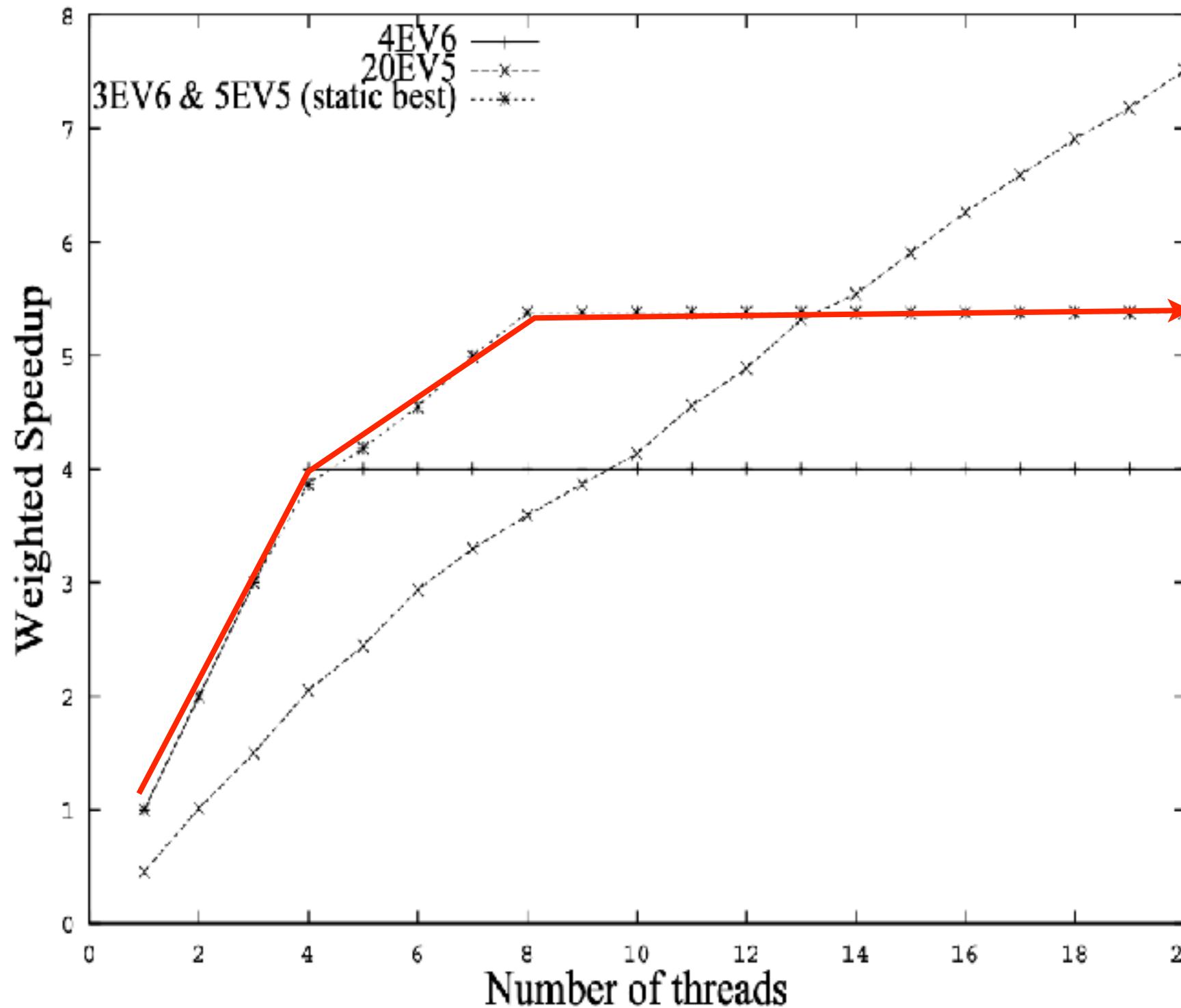
# Energy-delay

- Energy \* delay = Power \* ET \*  
ET = Power \* ET<sup>2</sup>

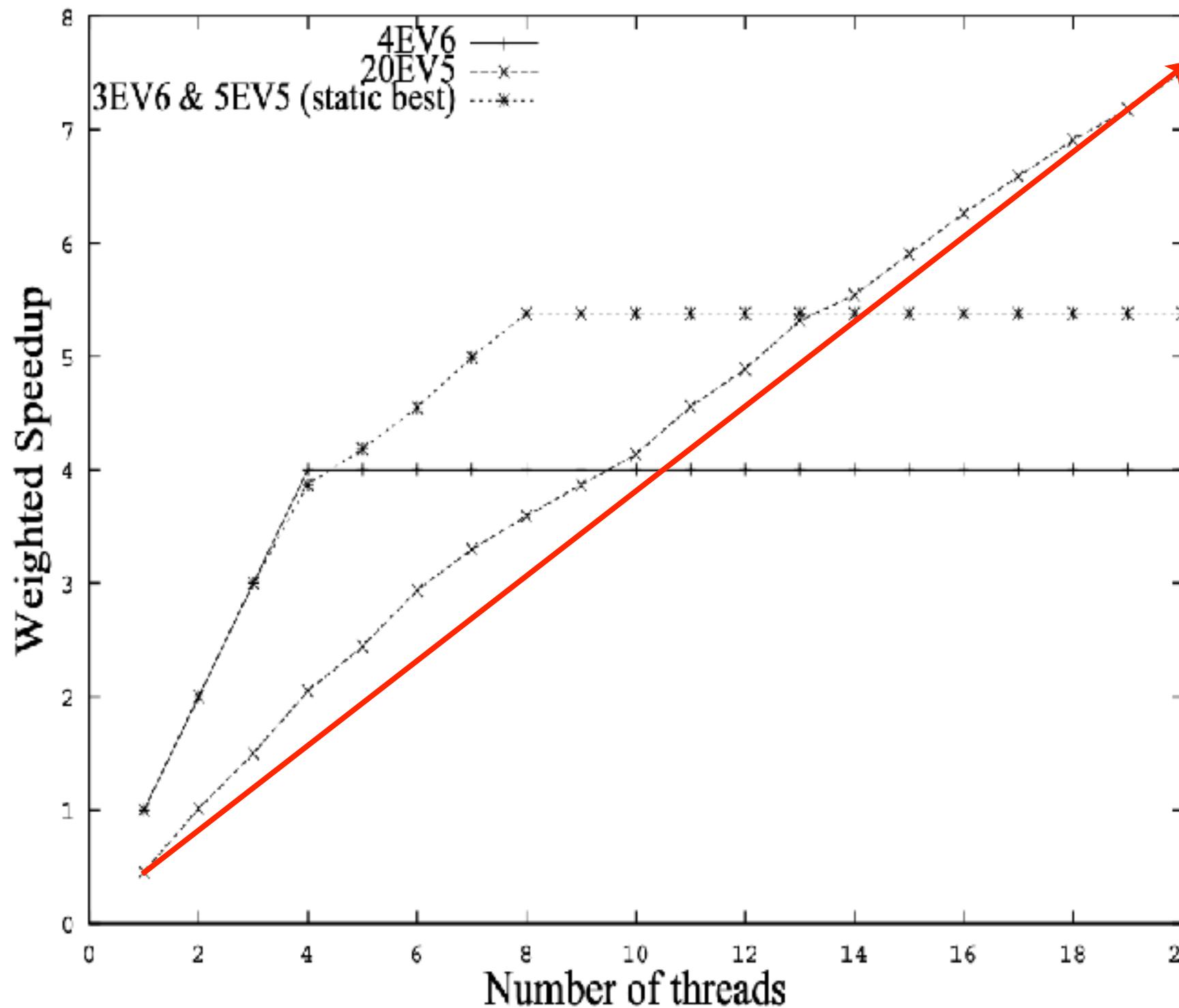


Benchmark	Total switches	% of instructions per core				Energy Savings(%)	ED Savings(%)	$ED^2$ Savings(%)	Perf. Loss (%)
		EV4	EV5	EV6	EV8-				
ammp	0	0	0	0	100	0	0	0	0
applu	27	2.2	0.1	54.5	43.2	42.7	38.6	33.6	7.1
apsi	2	0	0	62.2	37.8	27.6	25.3	22.9	3.1
art	0	0	0	100	0	74.4	73.5	72.6	3.3
equake	20	0	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
<i>Average</i>	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



# Demo

- We can use taskset command in linux to control the task allocation.
- Core i7 12700K is a processor with big.Little architecture

Processor Number	i7-12700K
Status	Launched
Launch Date	Q4'21
Lithography	Intel 7
Recommended Customer Price	\$450.00 - \$460.00
Use Conditions	PC/Client/Tablet, Workstation

## CPU Specifications

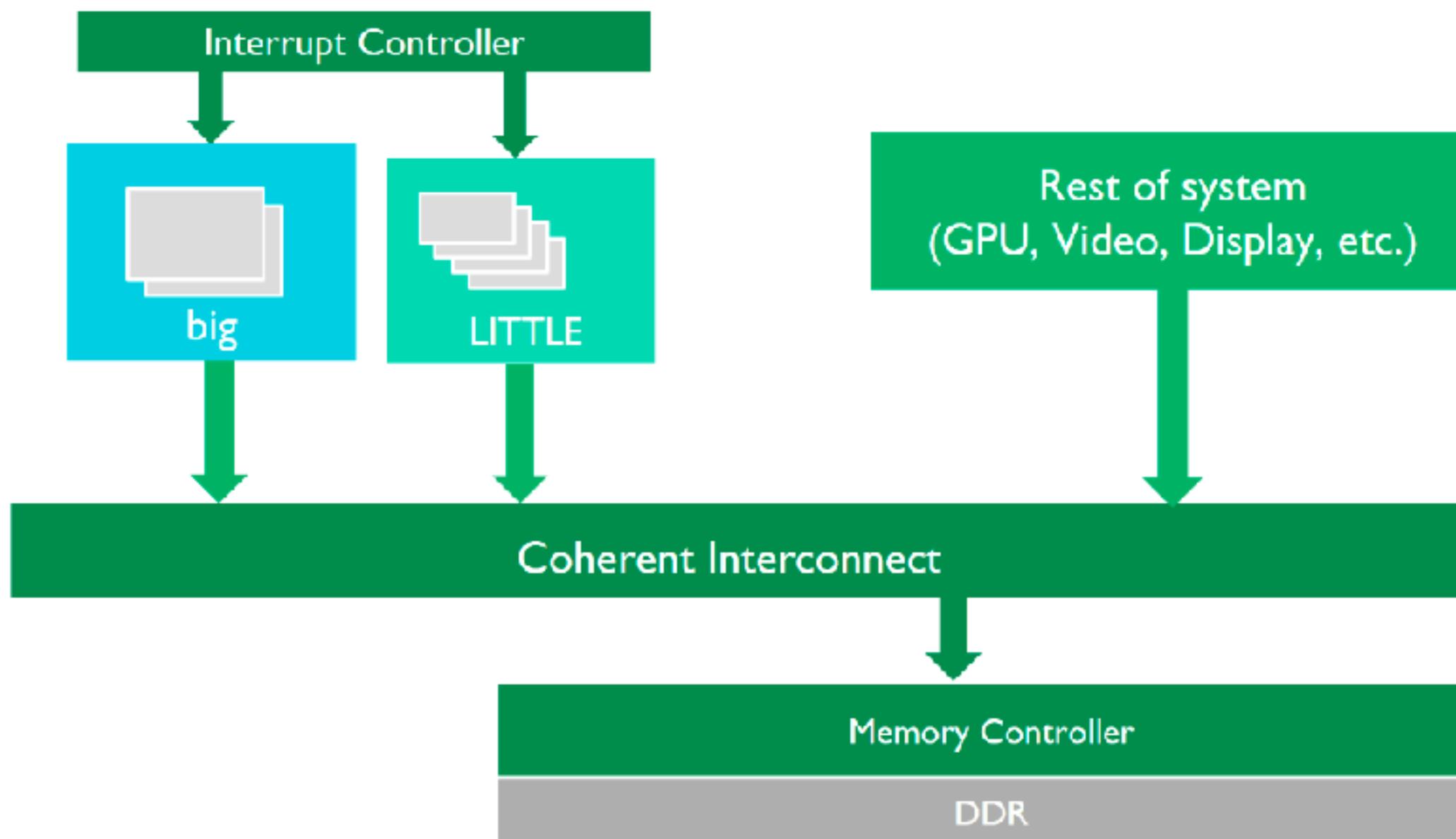
Total Cores	12
# of Performance-cores	8
# of Efficient-cores	4

# Single ISA heterogeneous CMP

- Regarding “Single-ISA Heterogeneous Multi-Core Architectures”, how many of the following statements is/are correct?
  - ① You need to recompile and optimize the binary for each core architecture to exploit the thread-level parallelism in this architecture
  - ② For a program with limited thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level (less than 10% difference) of performance than homogeneous CMP
  - ③ For a program with rich thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level of performance than homogeneous CMP built with older-generation cores
  - ④ Spending more instructions on older-generation cores would always lead to better energy-delay
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

# ARM's big.LITTLE architecture

## big.LITTLE system



# More cores per chip, slower per core

Products	Solutions	Support	intel
Intel® Xeon® Processor E7-8890 v4	X	Intel® Xeon® Processor E7-8893 v4	X
Intel® Xeon® Processor E7-8880 v4	X		
Status	Launched	Launched	Launched
Launch Date <span style="color: blue;">i</span>	Q2'16	Q2'16	Q2'16
Lithography <span style="color: blue;">i</span>	14 nm	14 nm	14 nm
Performance			
# of Cores <span style="color: blue;">i</span>	24	4	22
# of Threads <span style="color: blue;">i</span>	48	8	44
Processor Base Frequency <span style="color: blue;">i</span>	2.20 GHz	3.20 GHz	2.20 GHz
Max Turbo Frequency <span style="color: blue;">i</span>	3.40 GHz	3.50 GHz	3.30 GHz
Cache <span style="color: blue;">i</span>	60 MB	60 MB	55 MB
Bus Speed <span style="color: blue;">i</span>	9.6 GT/s	9.6 GT/s	9.6 GT/s
# of QPI Links <span style="color: blue;">i</span>	3	3	3
TDP <span style="color: blue;">i</span>	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 <span style="color: blue;">?</span>	Q4'17
Lithography <span style="color: blue;">?</span>	14 nm

## Performance

# of Cores <span style="color: blue;">?</span>	72
# of Threads <span style="color: blue;">?</span>	72
Processor Base Frequency <span style="color: blue;">?</span>	1.50 GHz
Max Turbo Frequency <span style="color: blue;">?</span>	1.60 GHz
Cache <span style="color: blue;">?</span>	36 MB L2 Cache
TDP <span style="color: blue;">?</span>	320 W

# The rise of GPU



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

A total of  $16 \times 12 = 192$  cores!



# 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.

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

**How about static power?**

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

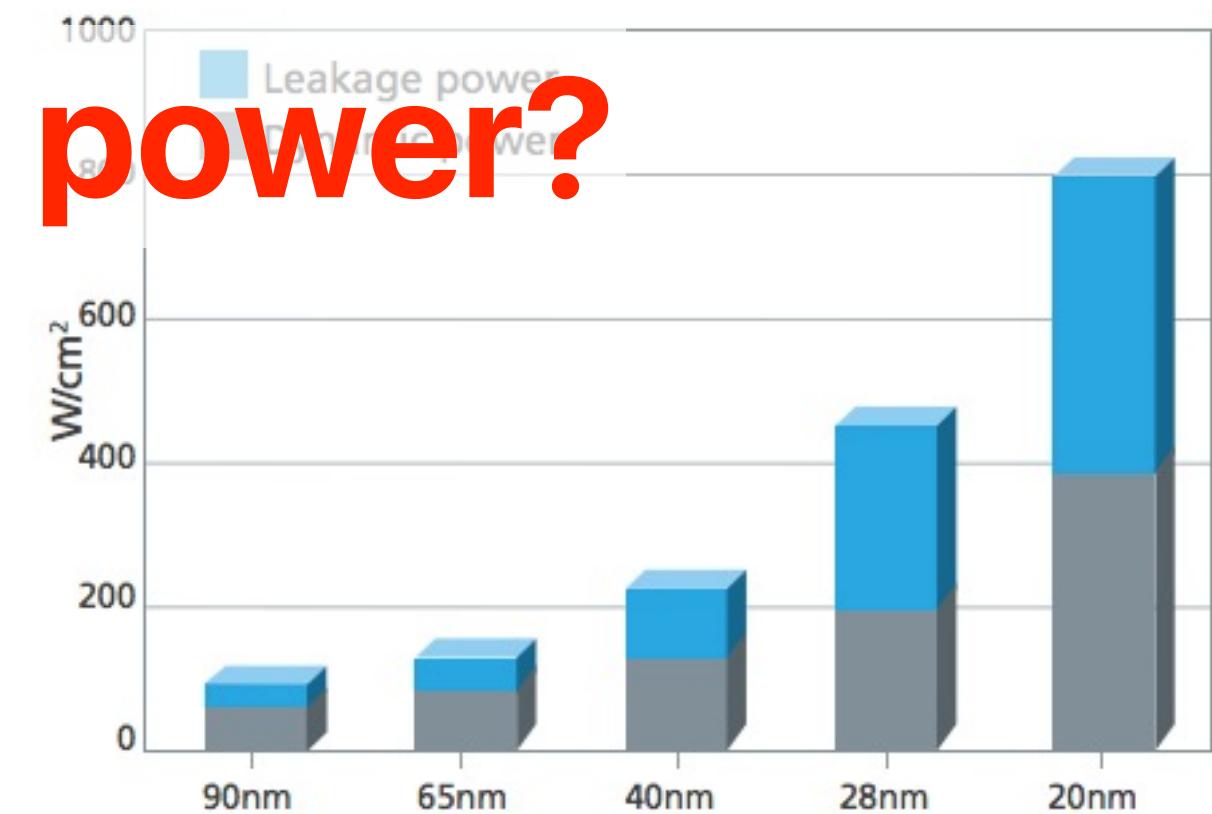
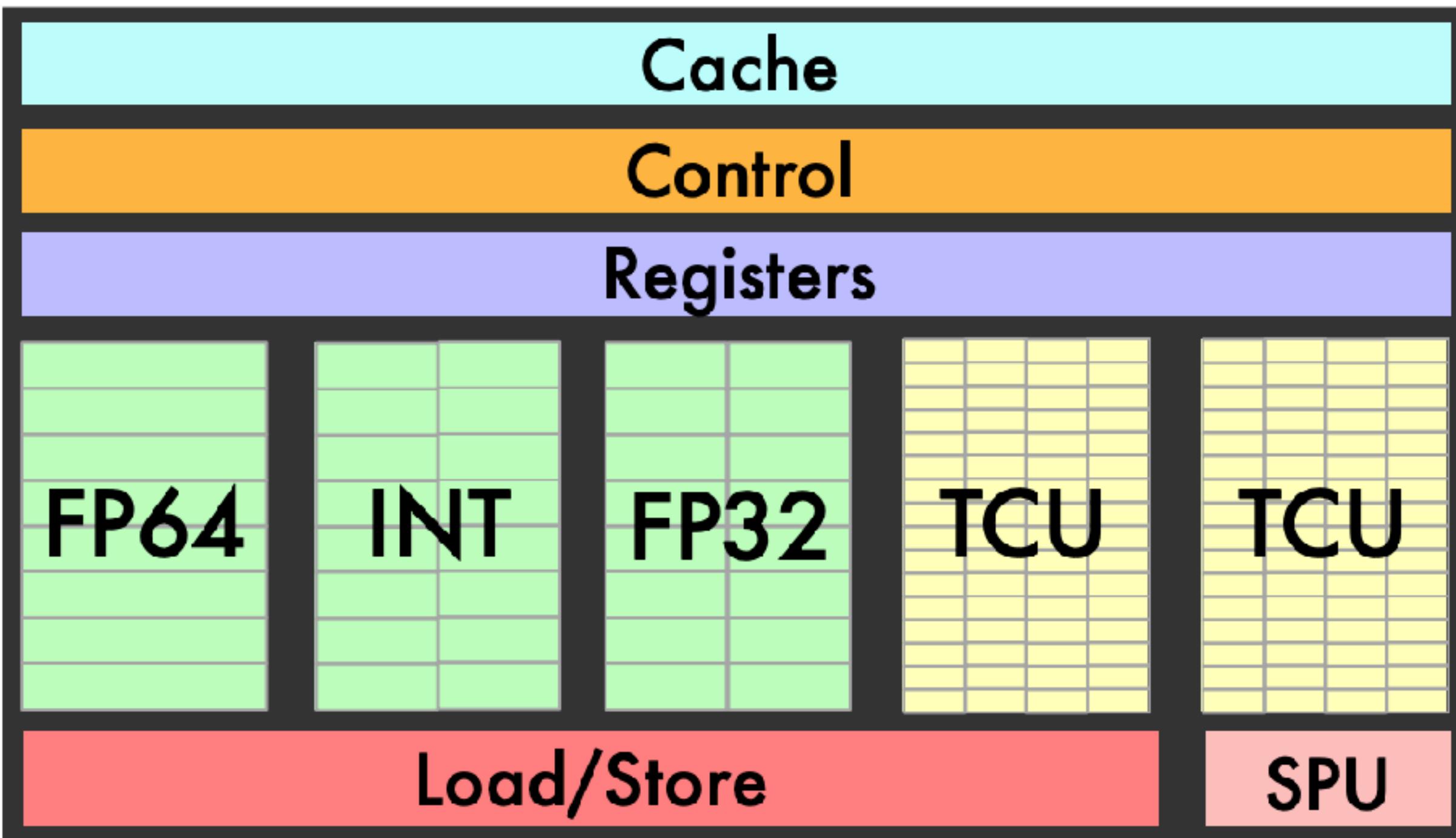


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

**Just let it dark**

# NVIDIA's Turing Architecture



# Programming in Turing Architecture

Use tensor cores

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

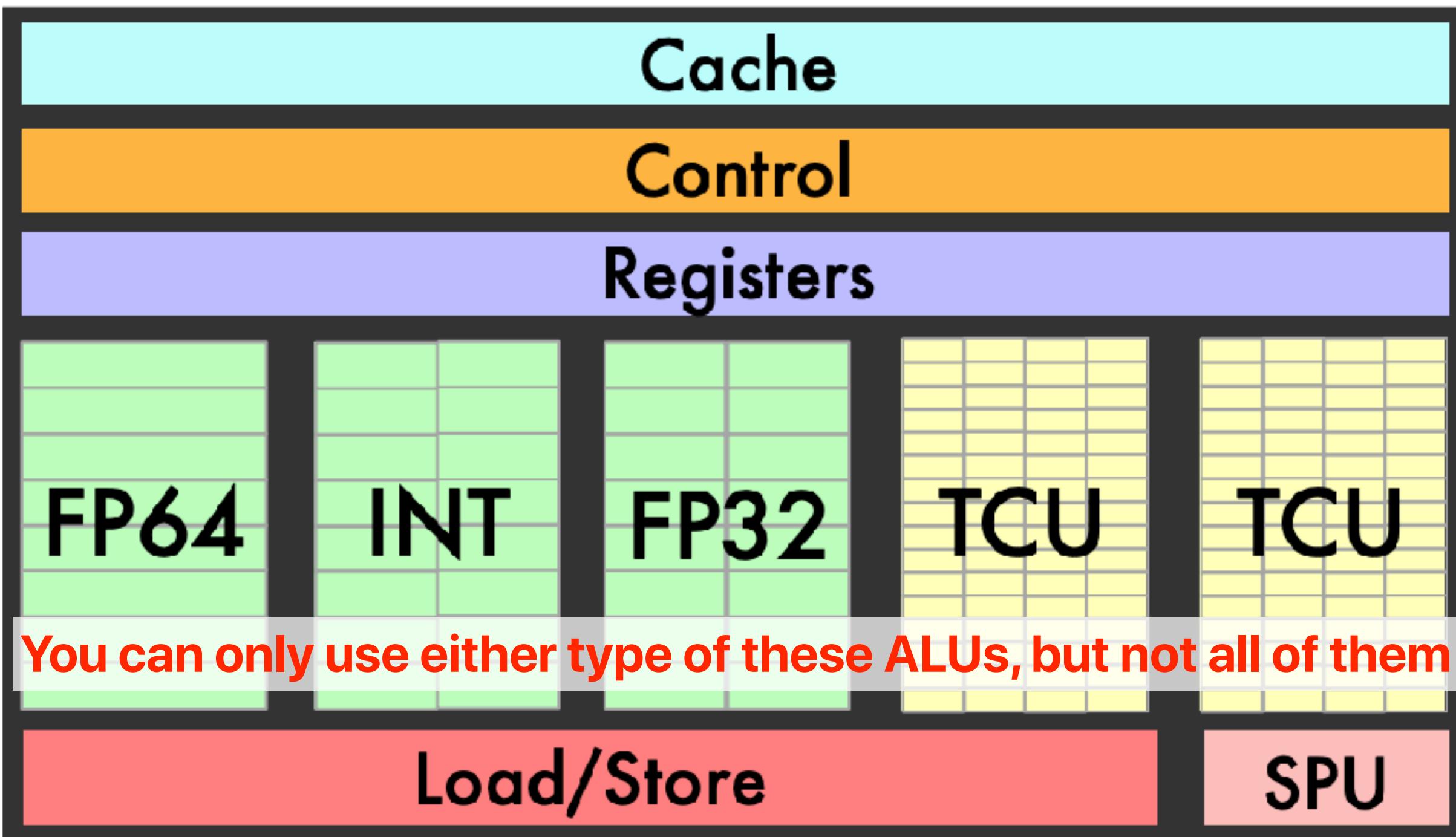
Make them 16-bit

```
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

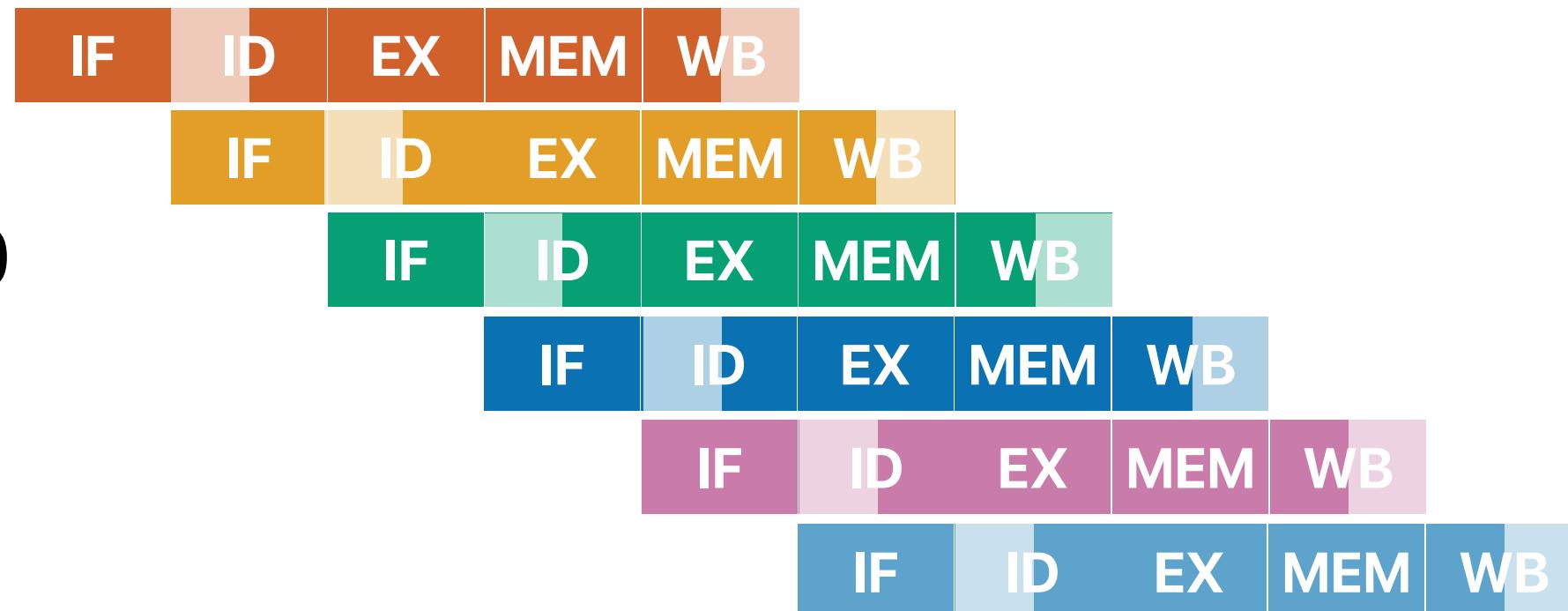


# The Rise of ASICS

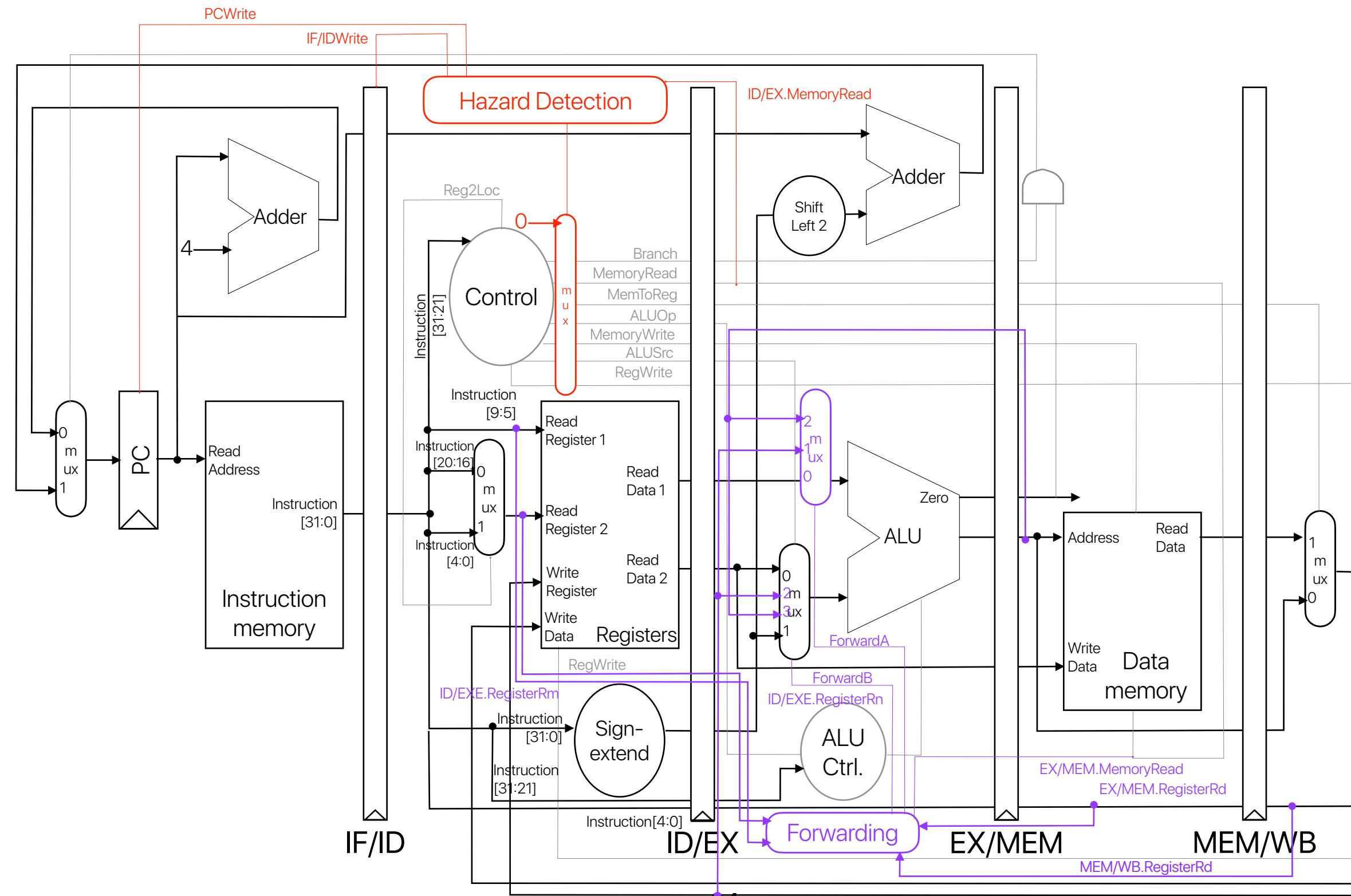
# Say, we want to implement $a[i] += a[i+1]*20$

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

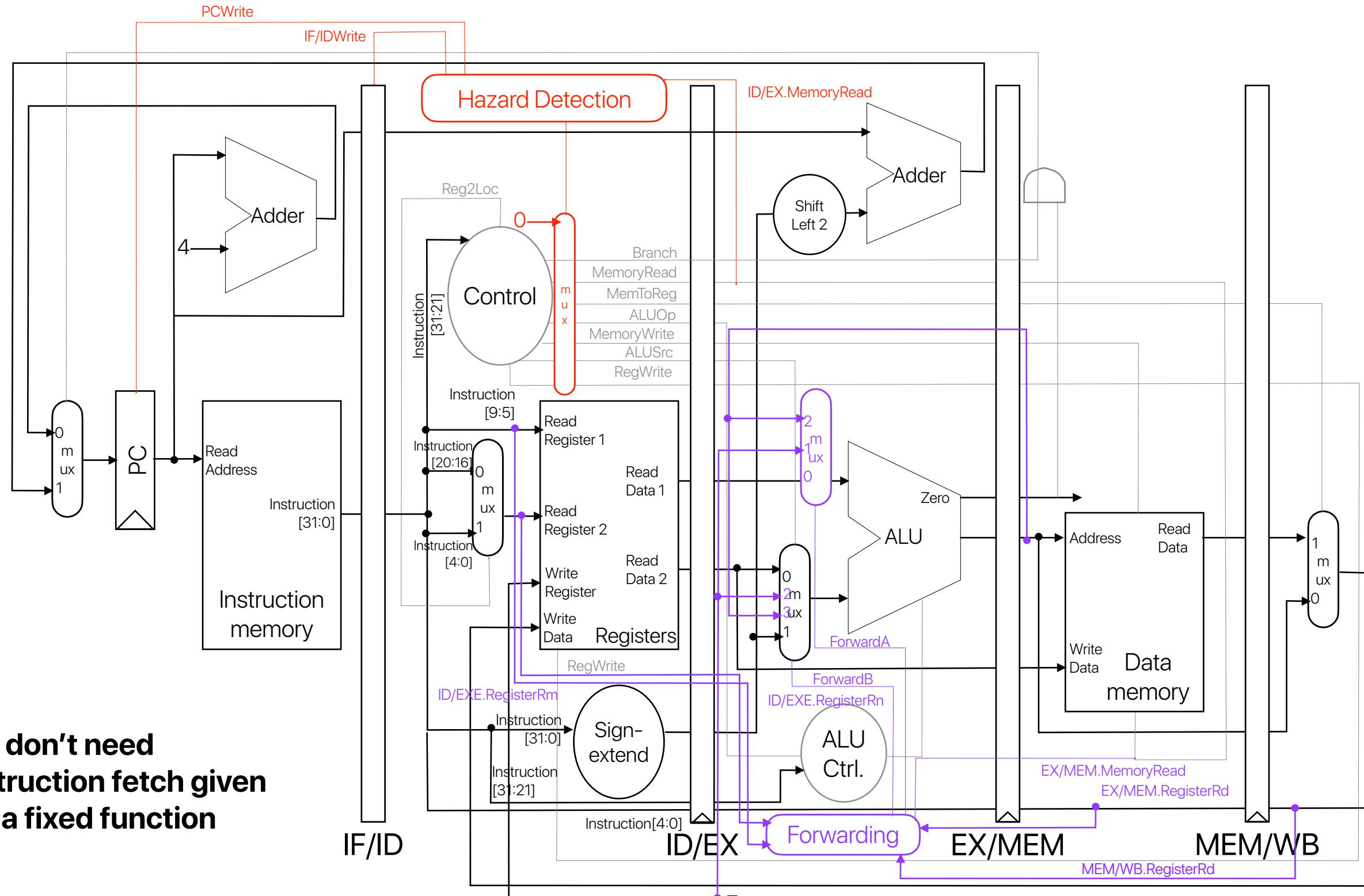
ld	X1,	0(X0)
ld	X2,	8(X0)
add	X3,	X31, #20
mul	X2,	X2, X3
add	X1,	X1, X2
sd	X1,	0(X0)



# This is what you need for these instructions



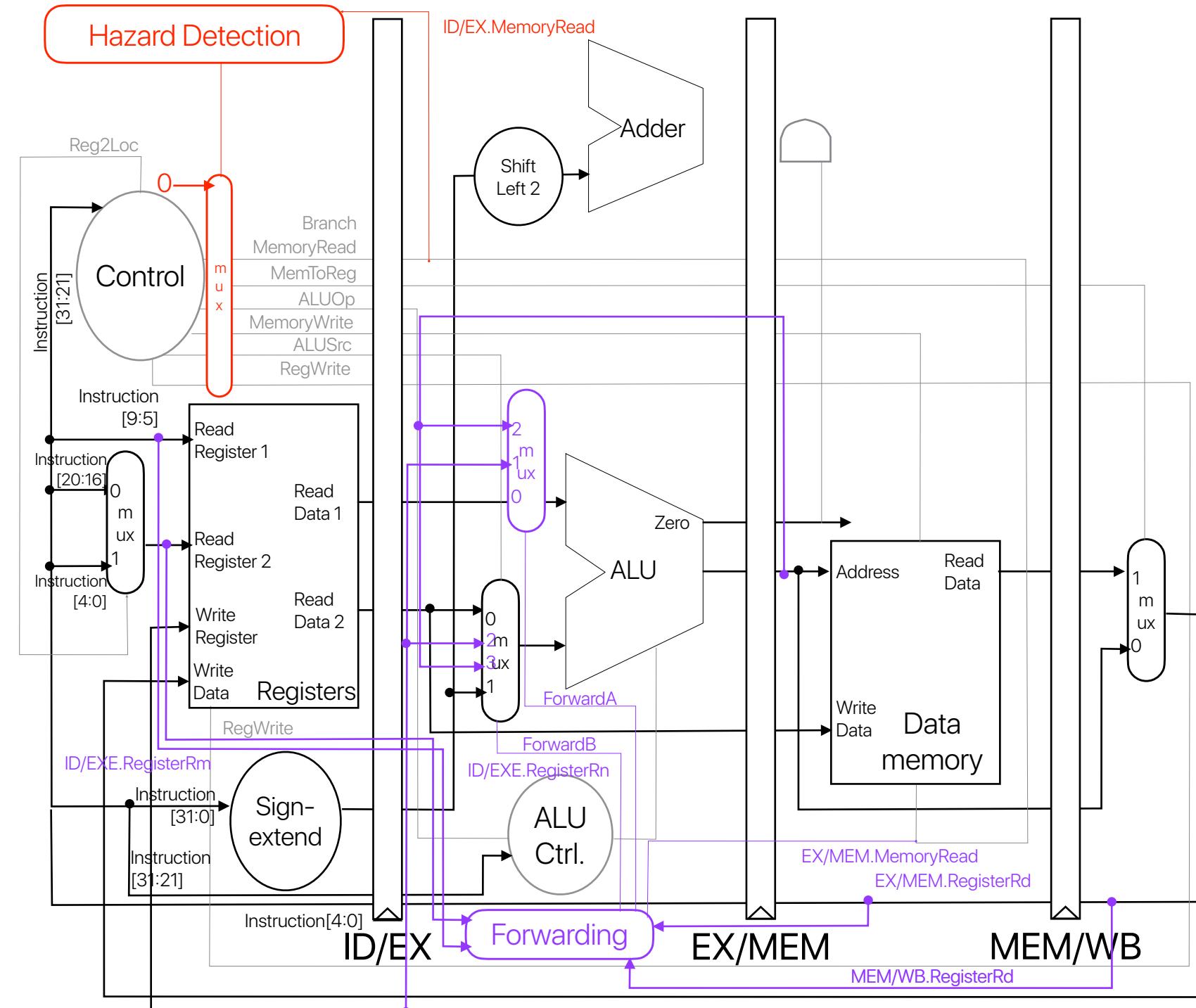
# Specialize the circuit



# 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

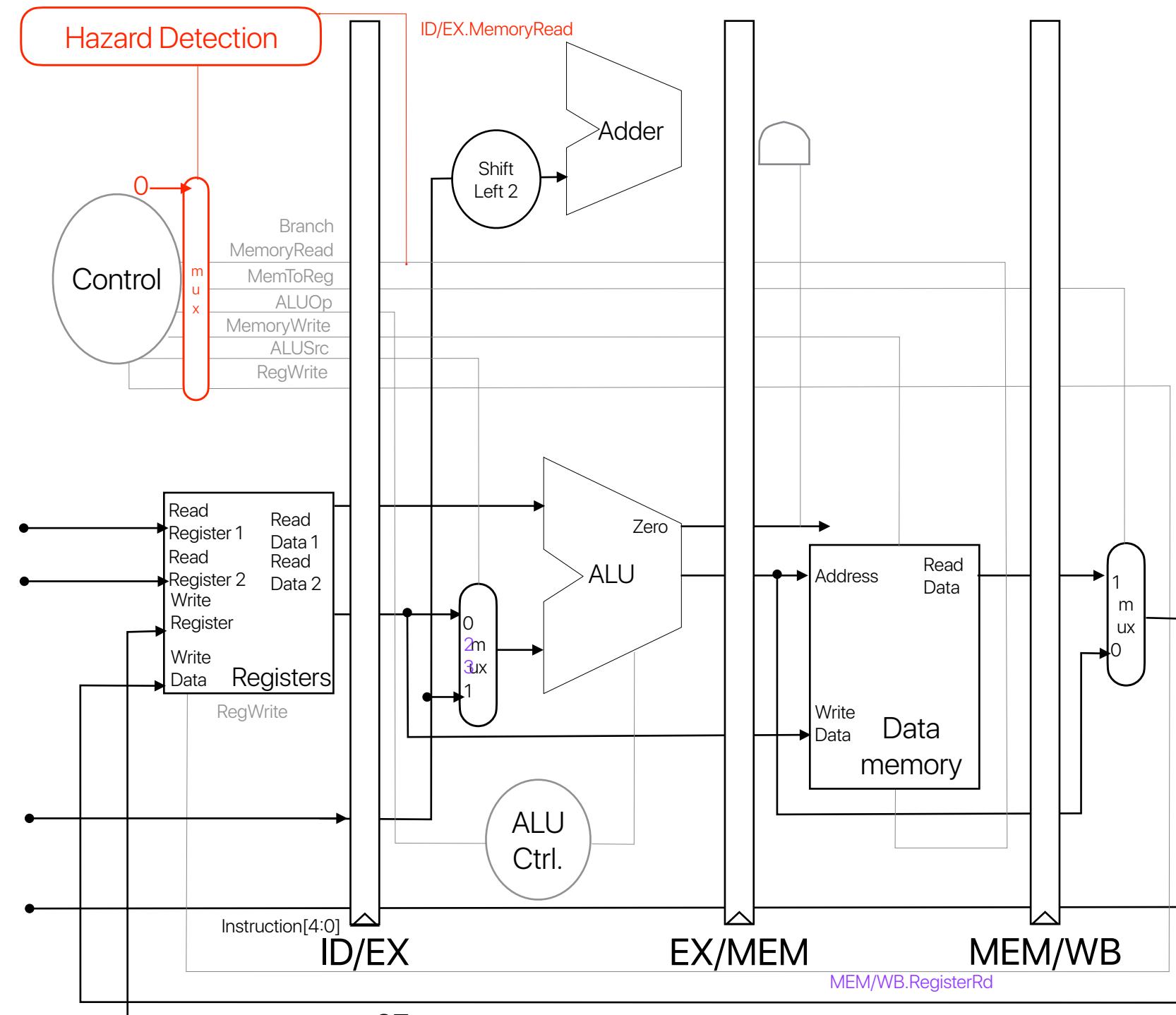


# Specialize the circuit

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

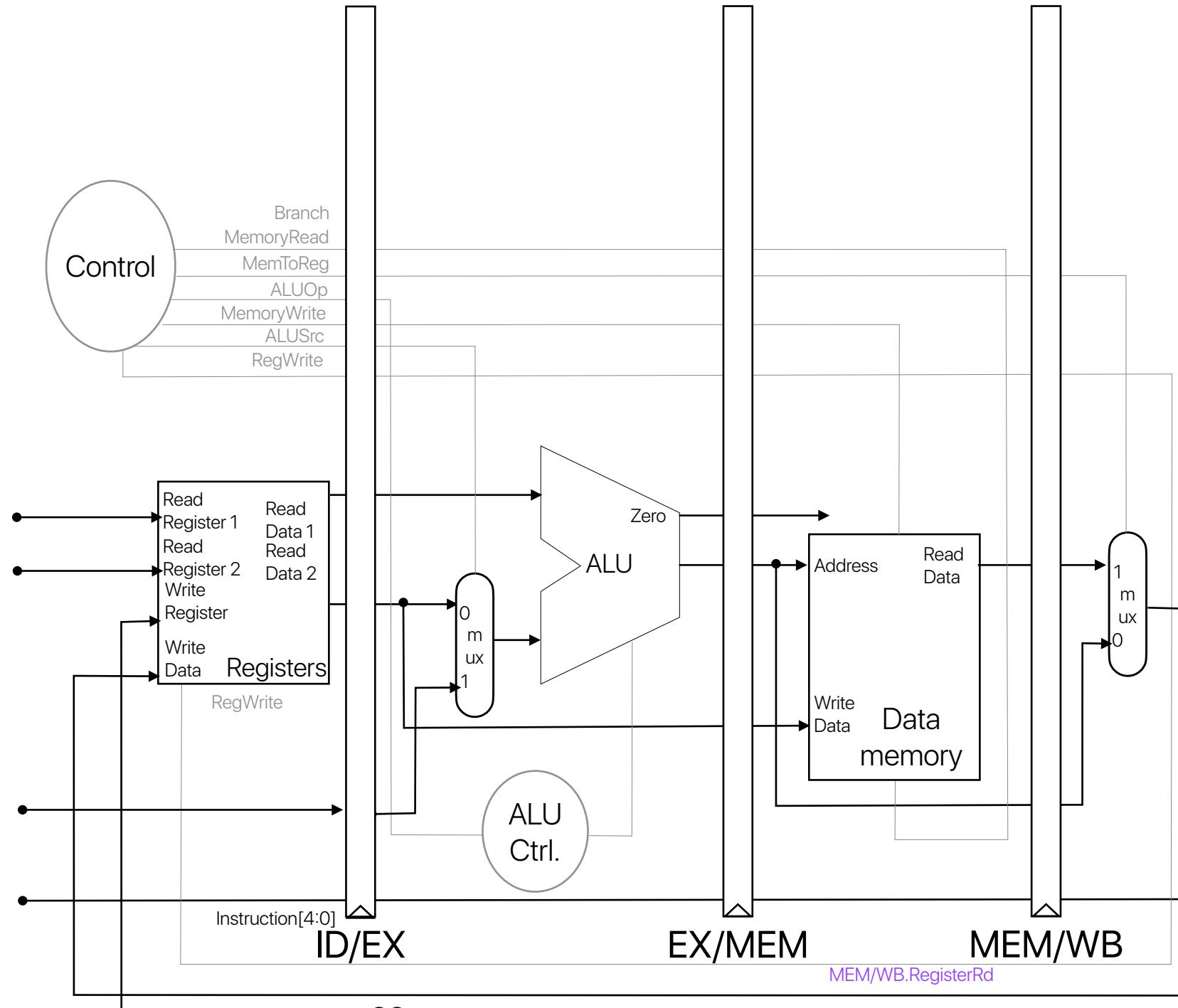


# Specialize the circuit

We don't need big ALUs,  
branches, hazard  
detections...

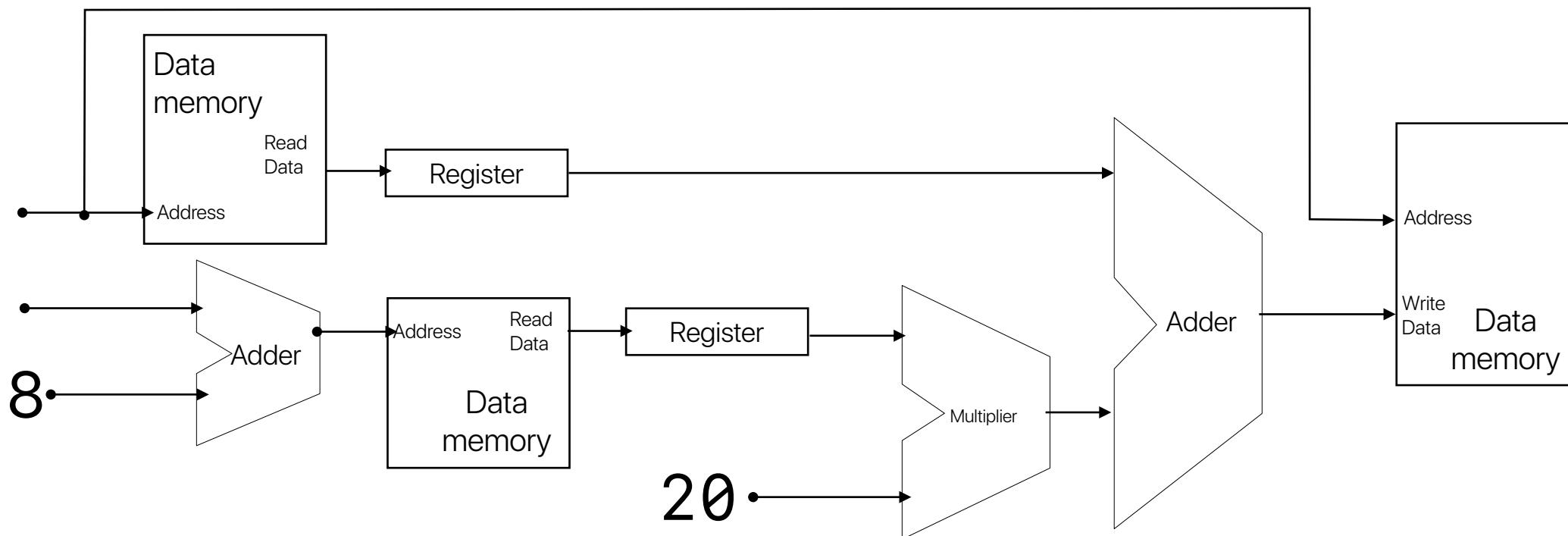
We don't need these  
many registers, complex  
control, decode

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



# Rearranging the datapath

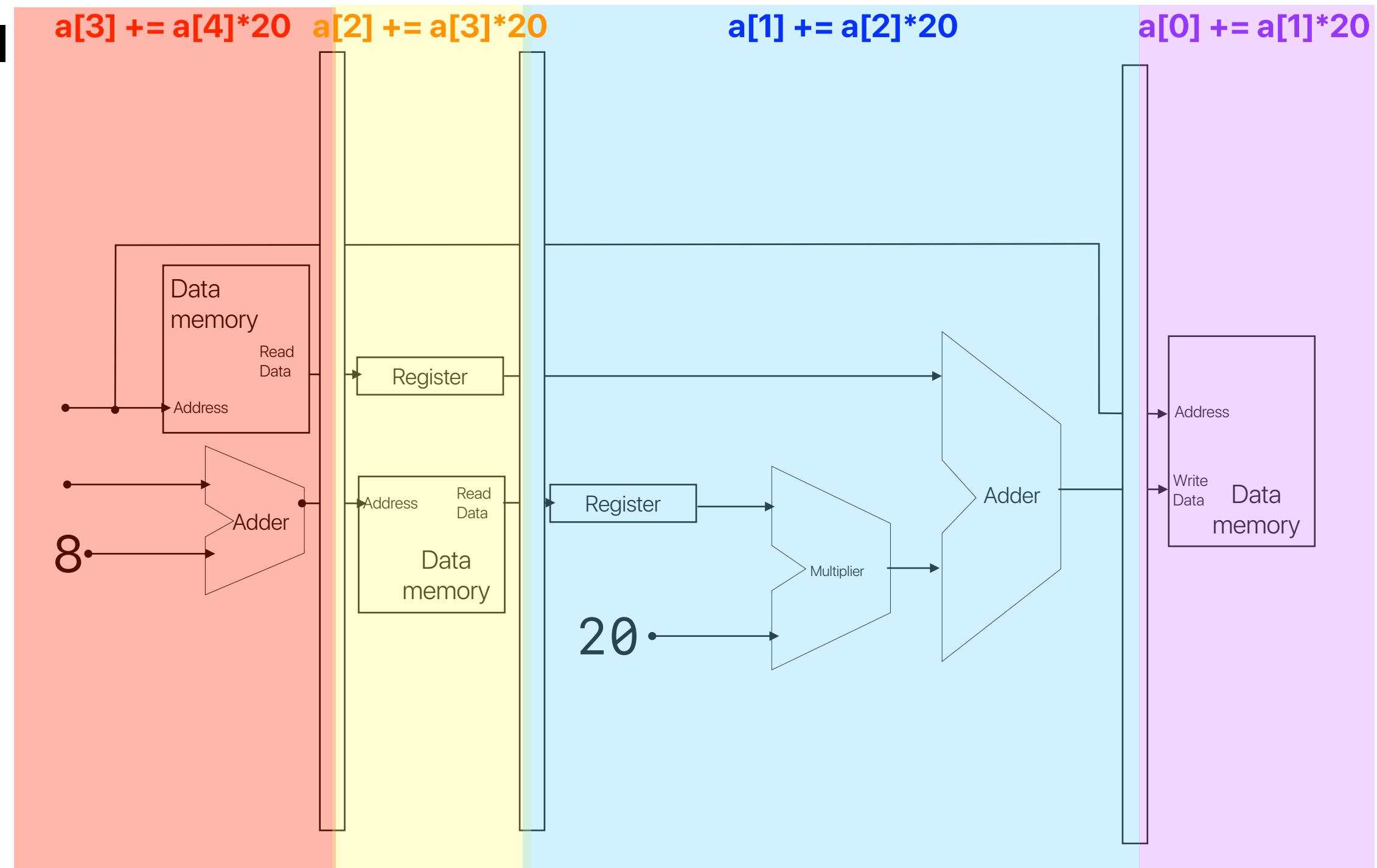
```
ld    X1, 0(X0)
ld    X2, 8(X0)
add   X3, X31, #20
mul   X2, X2, X3
add   X1, X1, X2
sd    X1, 0(X0)
```



# 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

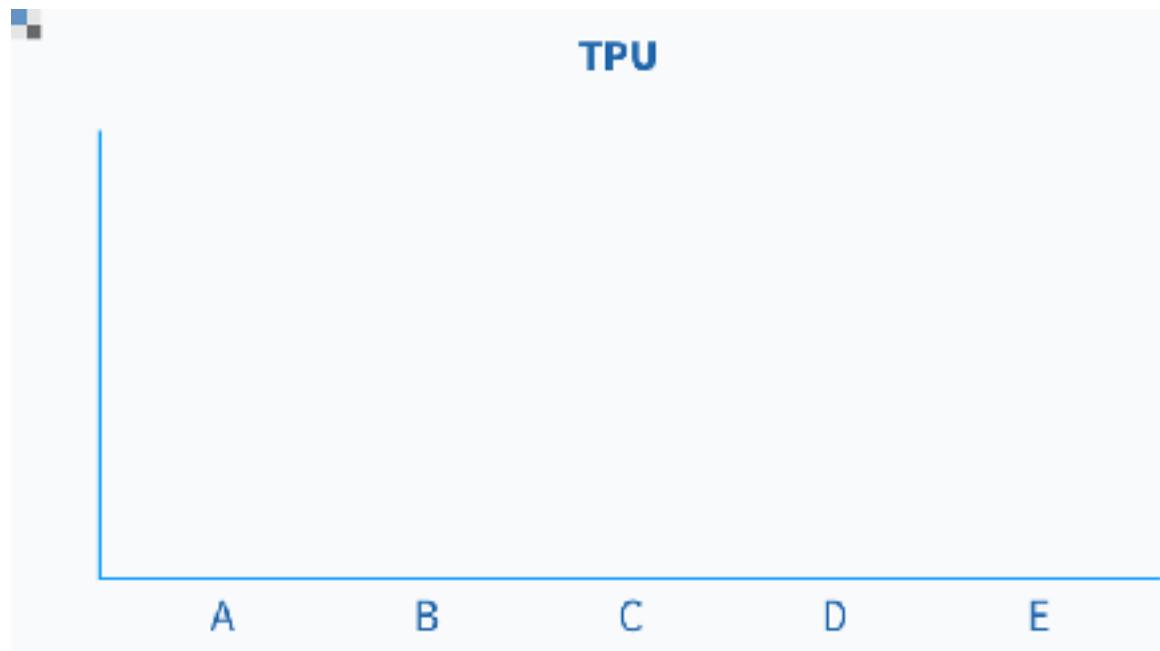
N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-I. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, 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.

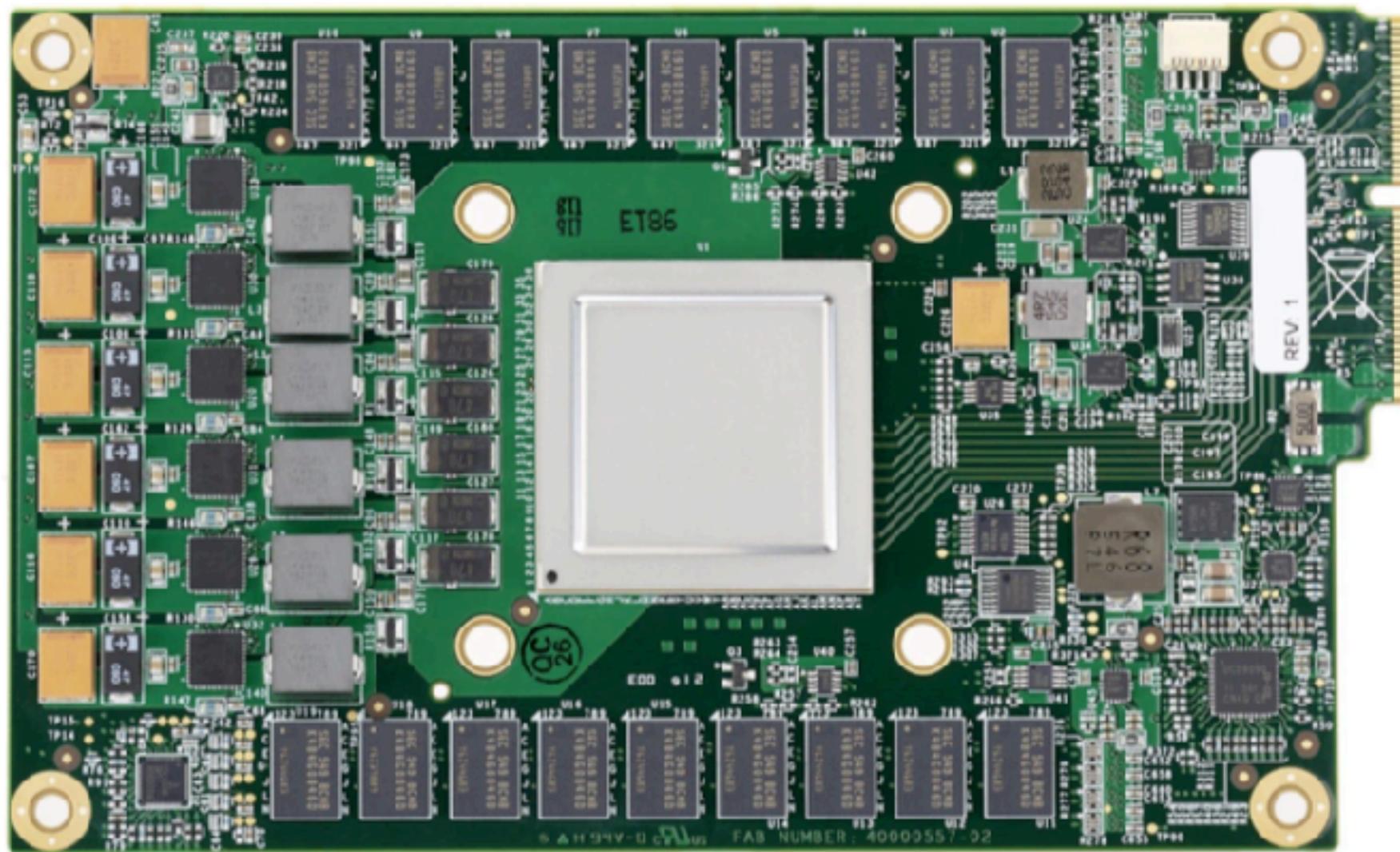
# TPU (Tensor Processing Unit)

- Regarding TPUs, please identify how many of the following statements are correct.
  - ① TPU is optimized for highly accurate matrix multiplications
  - ② TPU is designed for dense matrices, not for sparse matrices
  - ③ A majority of TPU's area is used by memory buffers
  - ④ All TPU instructions are equally long

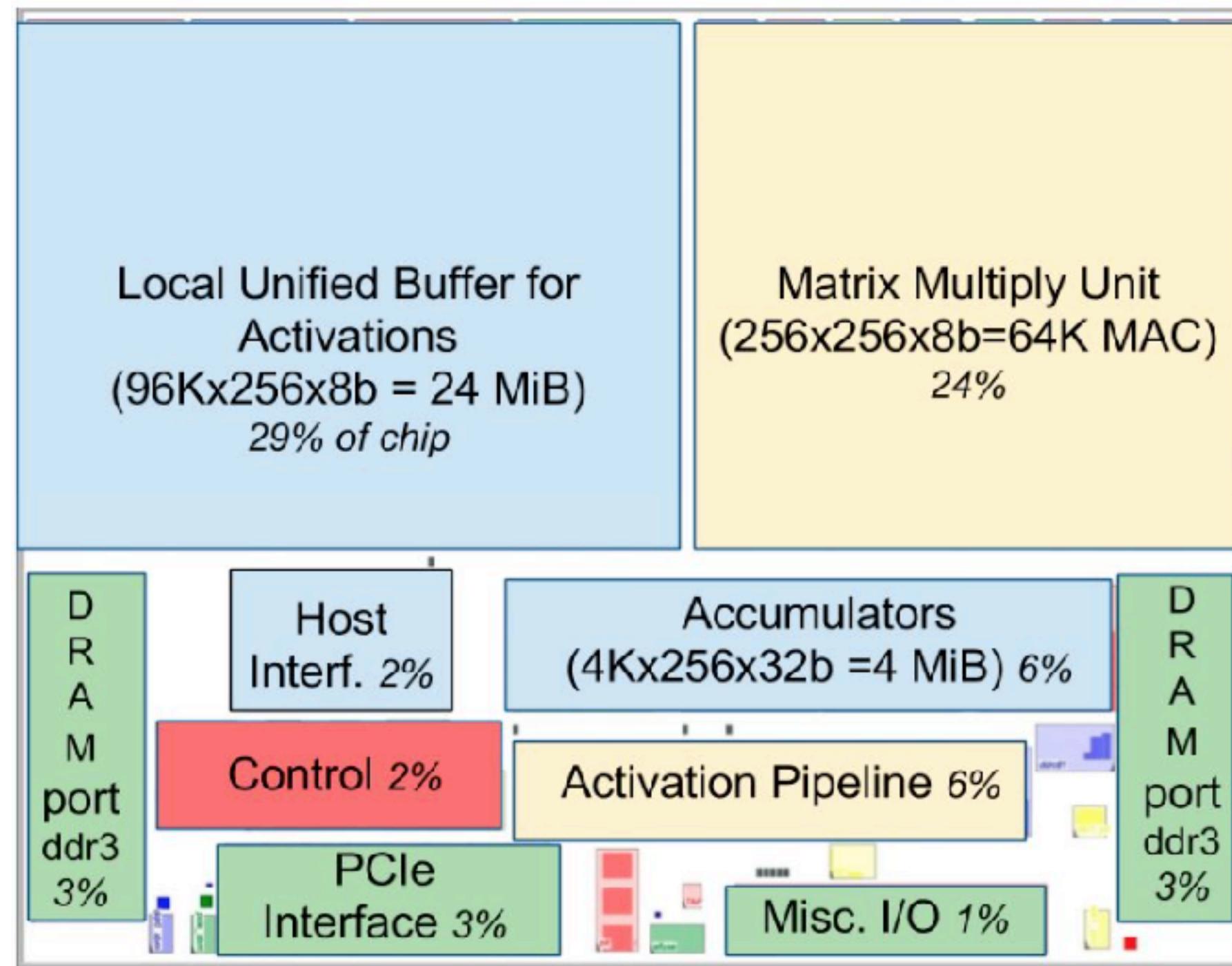
A. 0  
B. 1  
C. 2  
D. 3  
E. 4



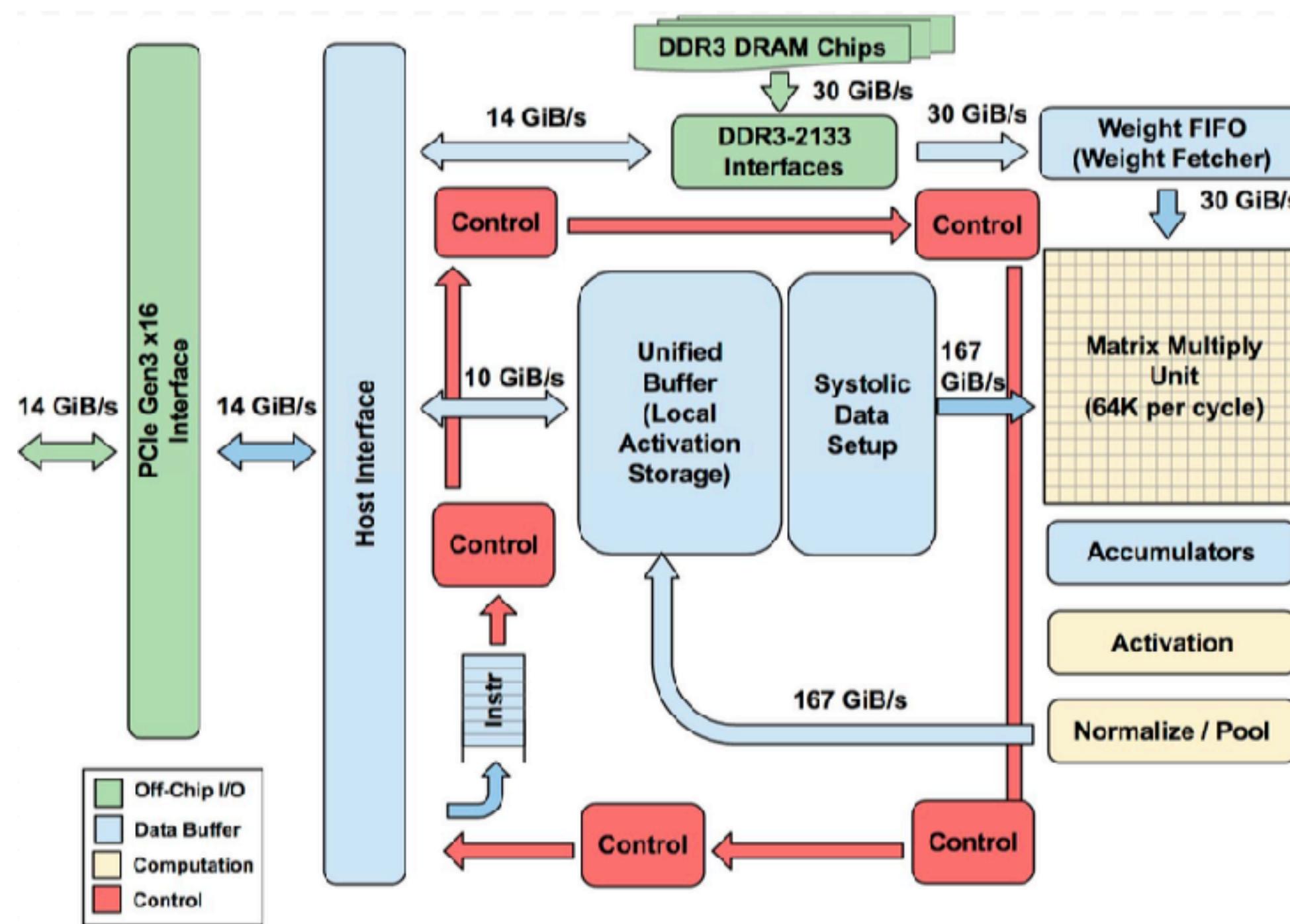
# What TPU looks like



# TPU Floorplan



# TPU Block diagram



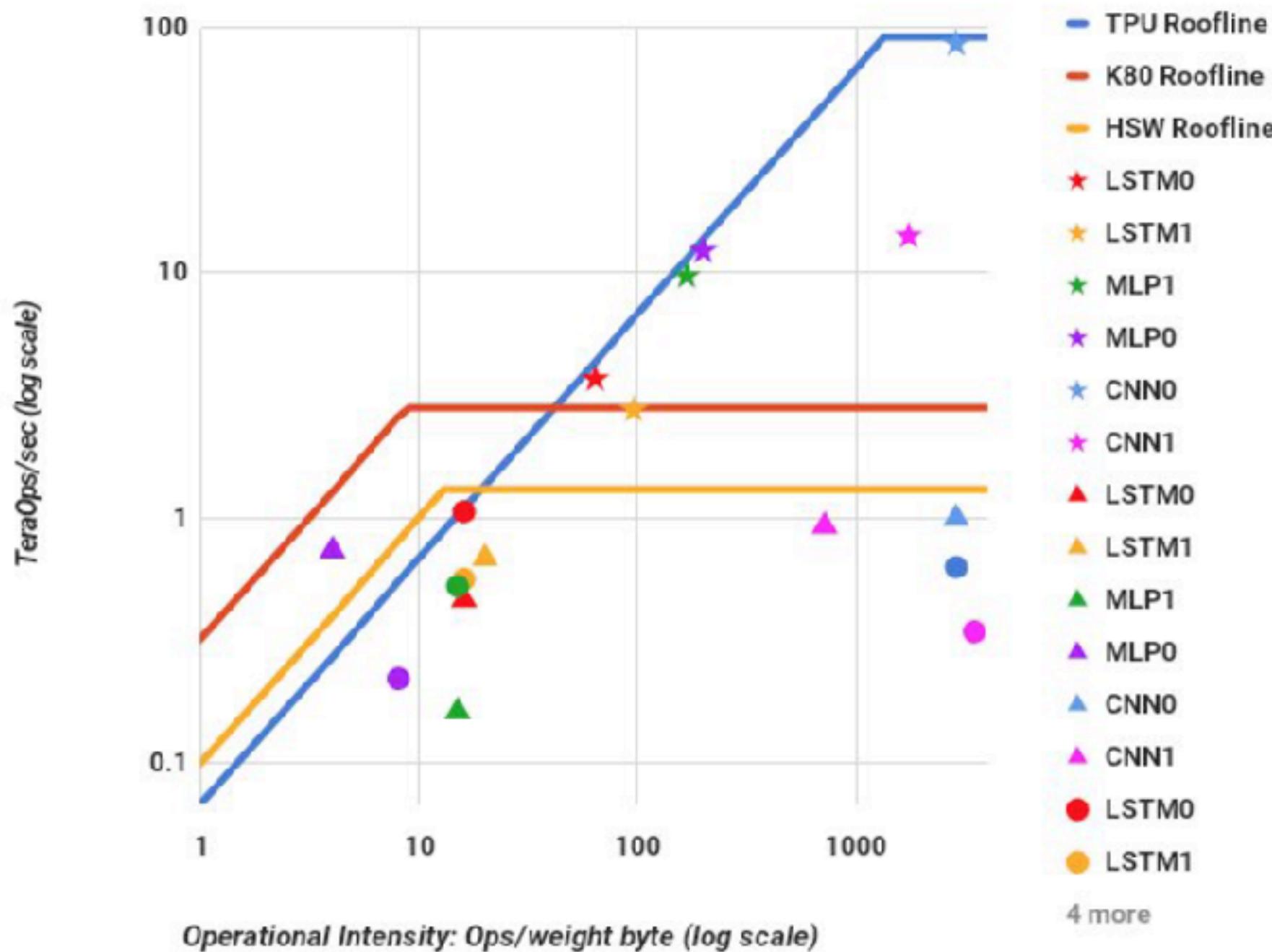
# TPU (Tensor Processing Unit)

- Regarding TPUs, please identify how many of the following statements are correct.
    - ① TPU is optimized for highly accurate matrix multiplications
    - ② TPU is designed for dense matrices, not for sparse matrices
    - ③ A majority of TPU's area is used by memory buffers
    - ④ All TPU instructions are equally long
- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

# Experimental setup

Model	Die									Benchmarked Servers						
	<i>mm</i> <sup>2</sup>	nm	MHz	TDP	Measured		TOPS/s		GB/s	On-Chip Memory	Dies	DRAM Size		TDP	Measured	
					Idle	Busy	8b	FP				Idle	Busy		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

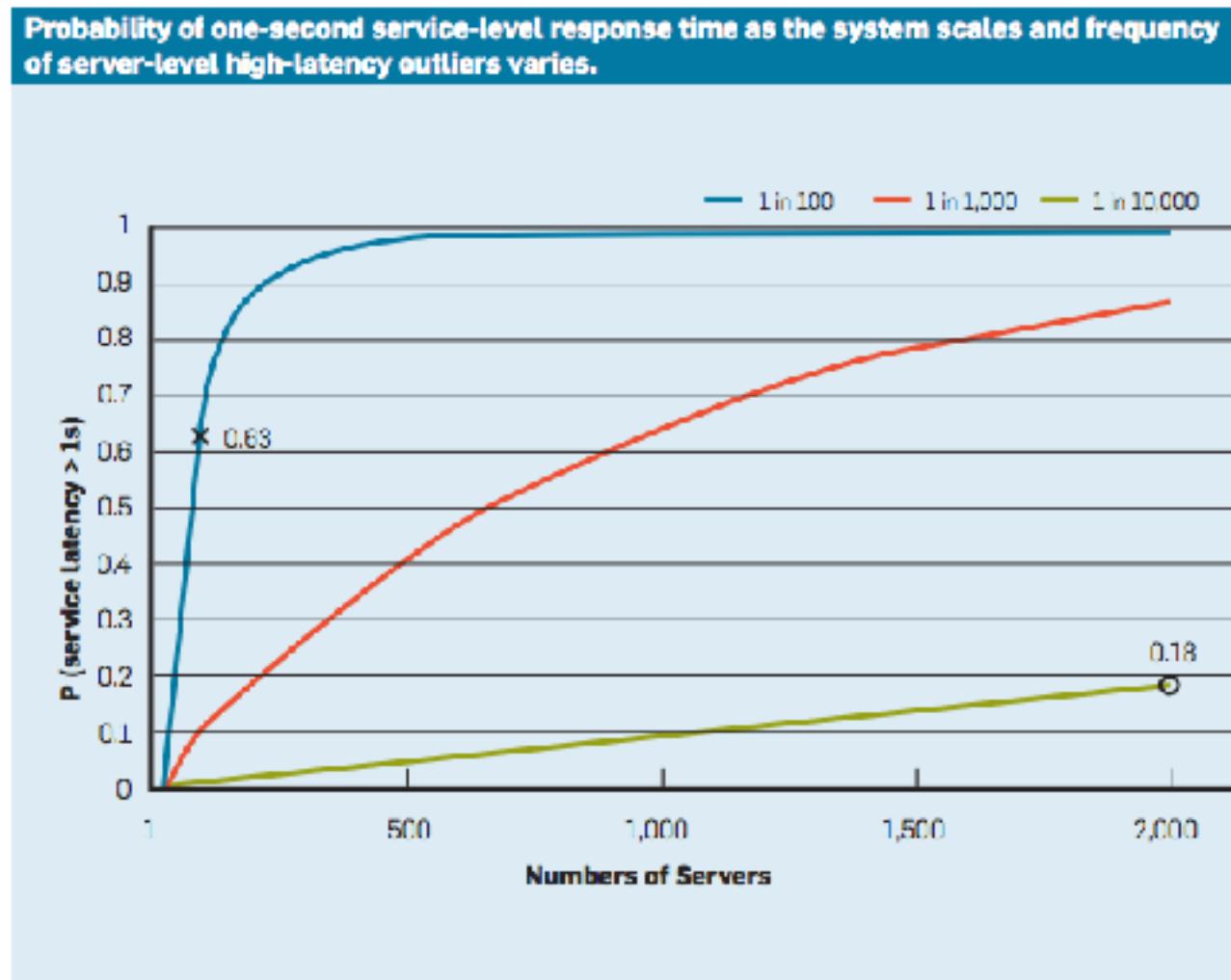


# Tail latencies

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
<b>Inferences/Sec &lt;10ms latency</b>	$\frac{1}{13}X$	1X	2X
<b>Training TOPS</b>	6 FP32	NA	12 FP32
<b>Inference TOPS</b>	6 FP32	90 INT8	48 INT8
<b>On-chip Memory</b>	16 MB	24 MB	11 MB
<b>Power</b>	300W	75W	250W
<b>Bandwidth</b>	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.

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

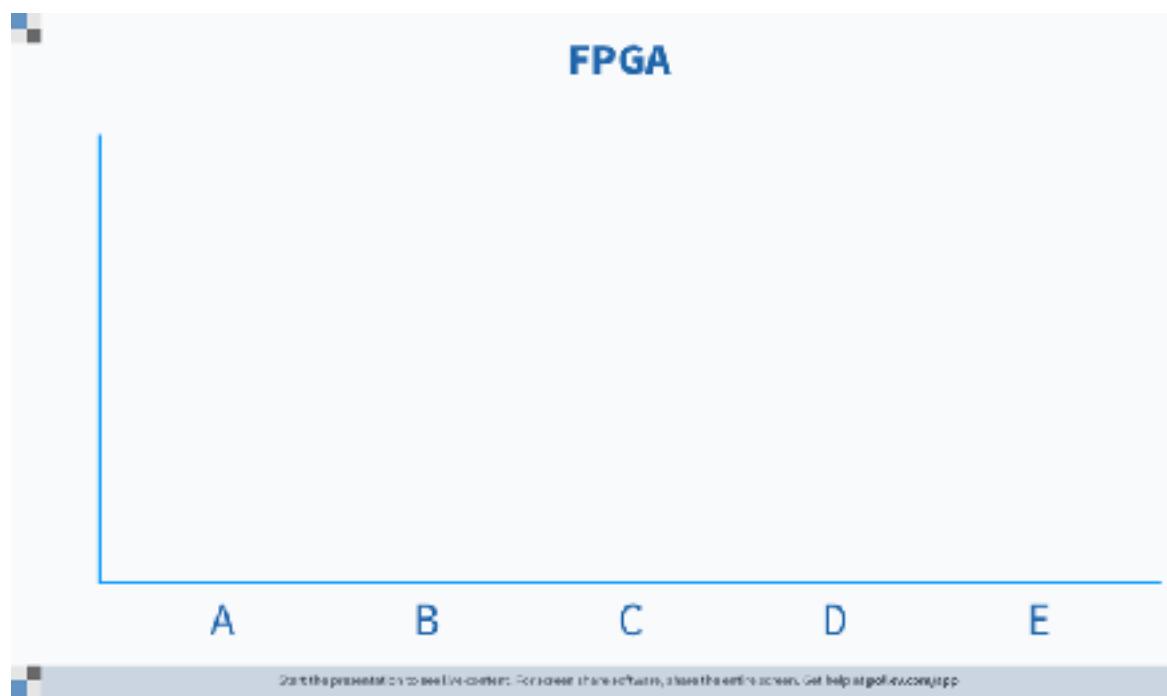
- Pitfall: Architects have neglected important NN tasks.
  - CNNs constitute only about 5% of the representative NN workload for Google. More attention should be paid to MLPs and LSTMs. Repeating history, it's similar to when many architects concentrated on floating- point performance when most mainstream 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**

# 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



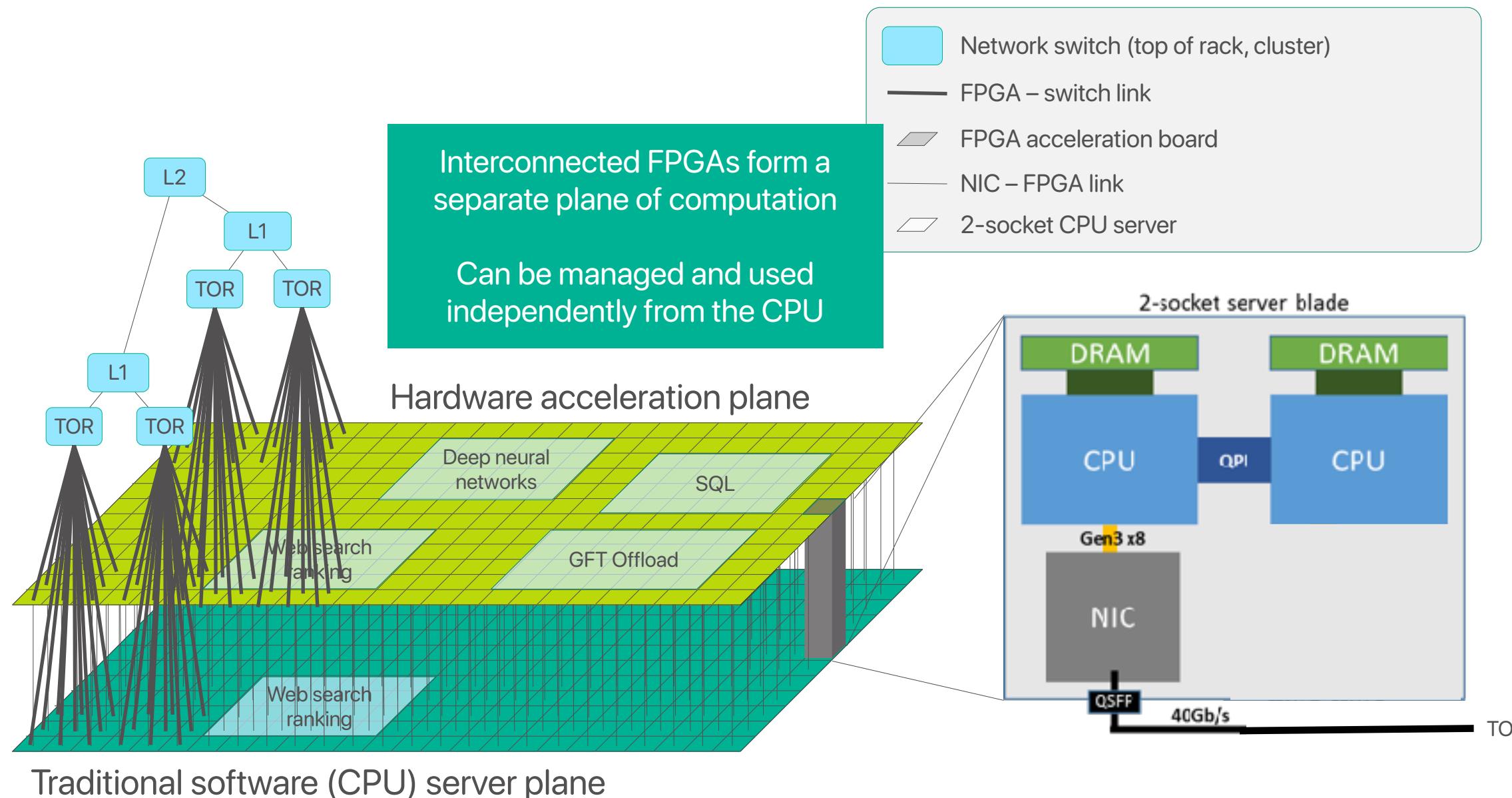
# FPGA

- 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



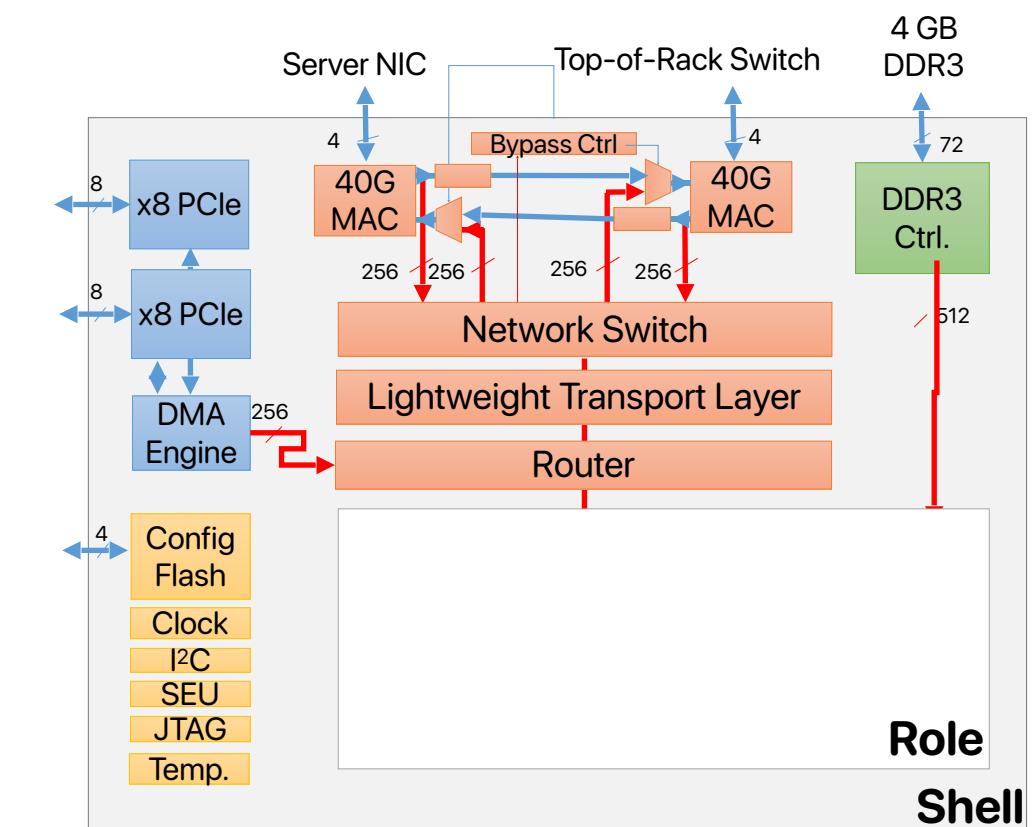
FPGA

# 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

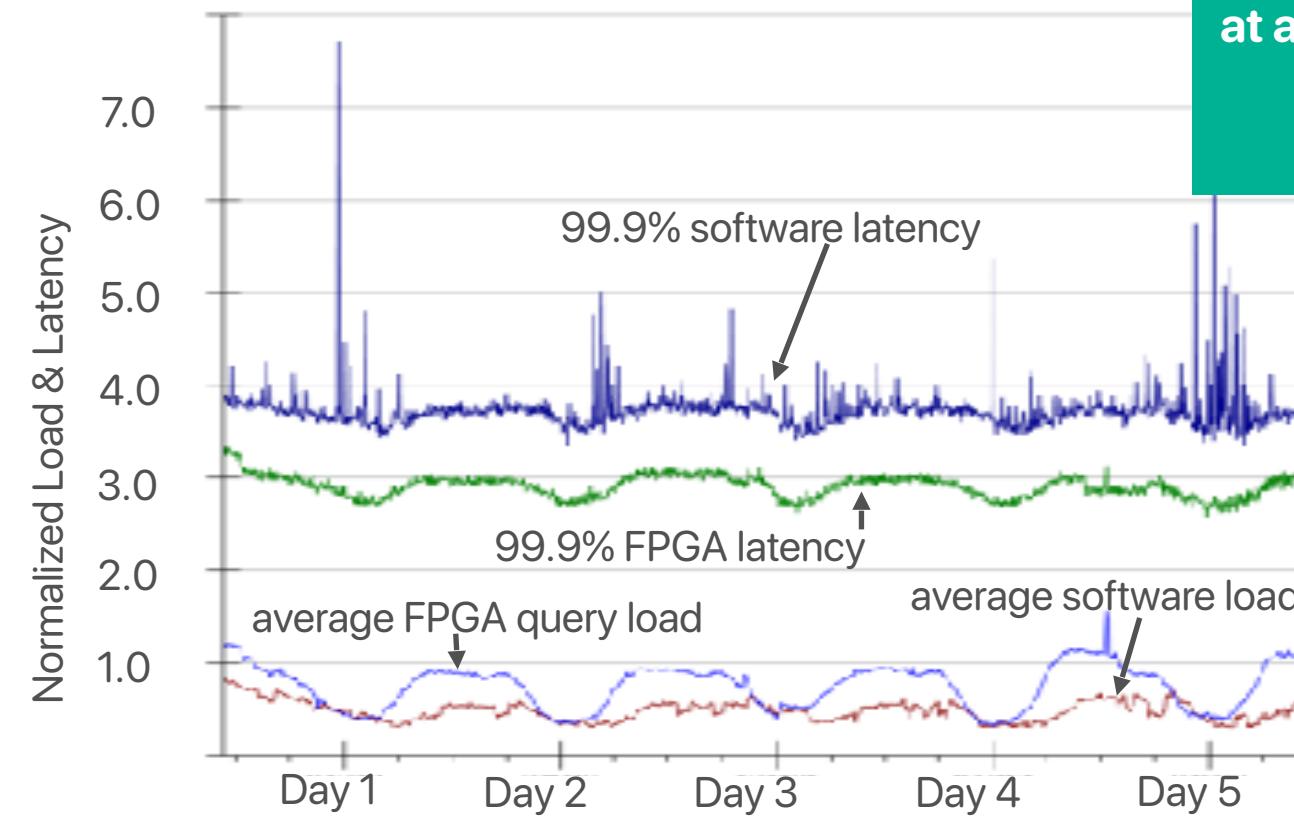


# Use cases

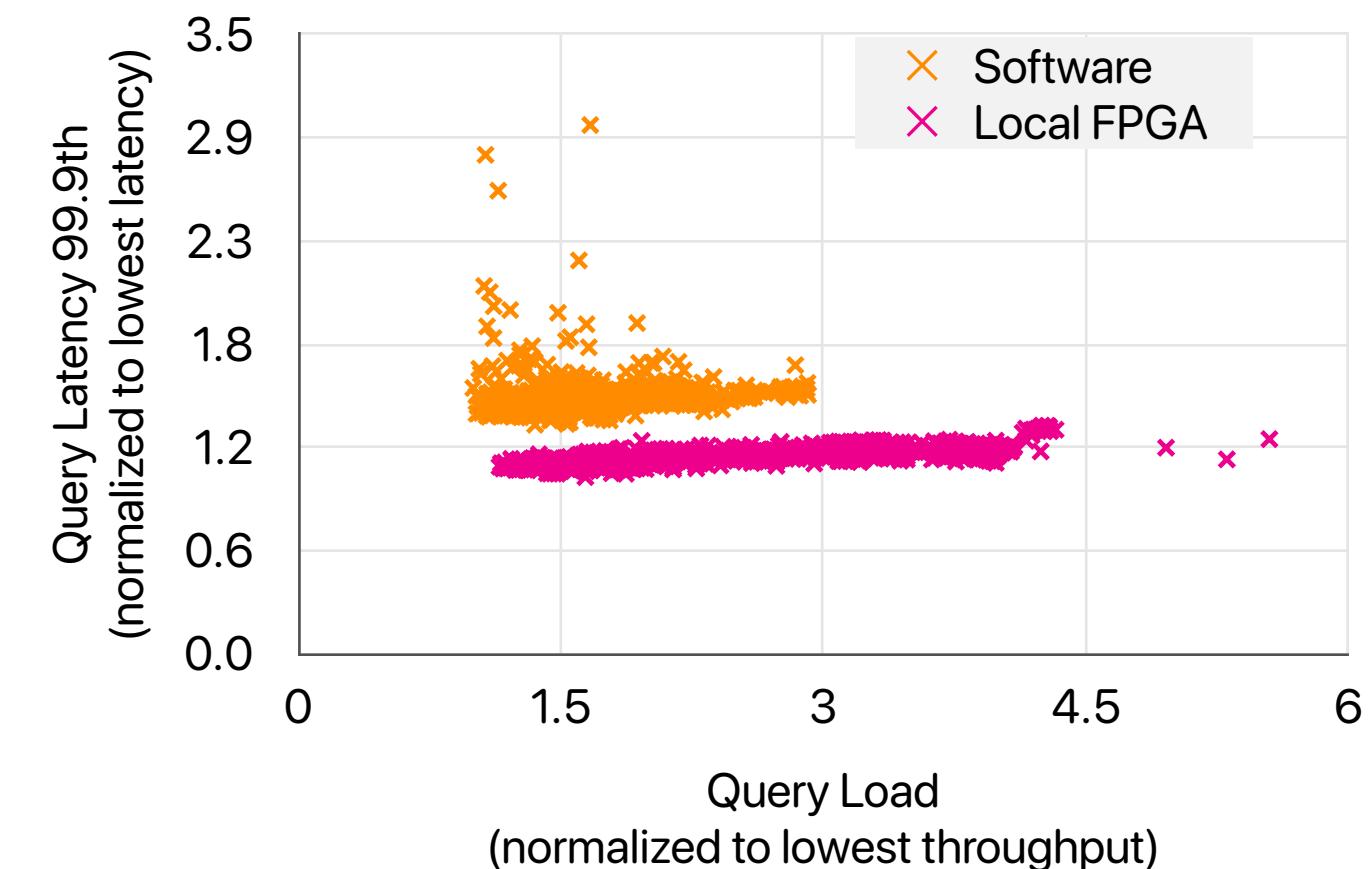
- Local: Great service acceleration
- Infrastructure: Fastest cloud network
- Remote: Reconfigurable app fabric (DNNs)

# 5 day bed-level latency

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

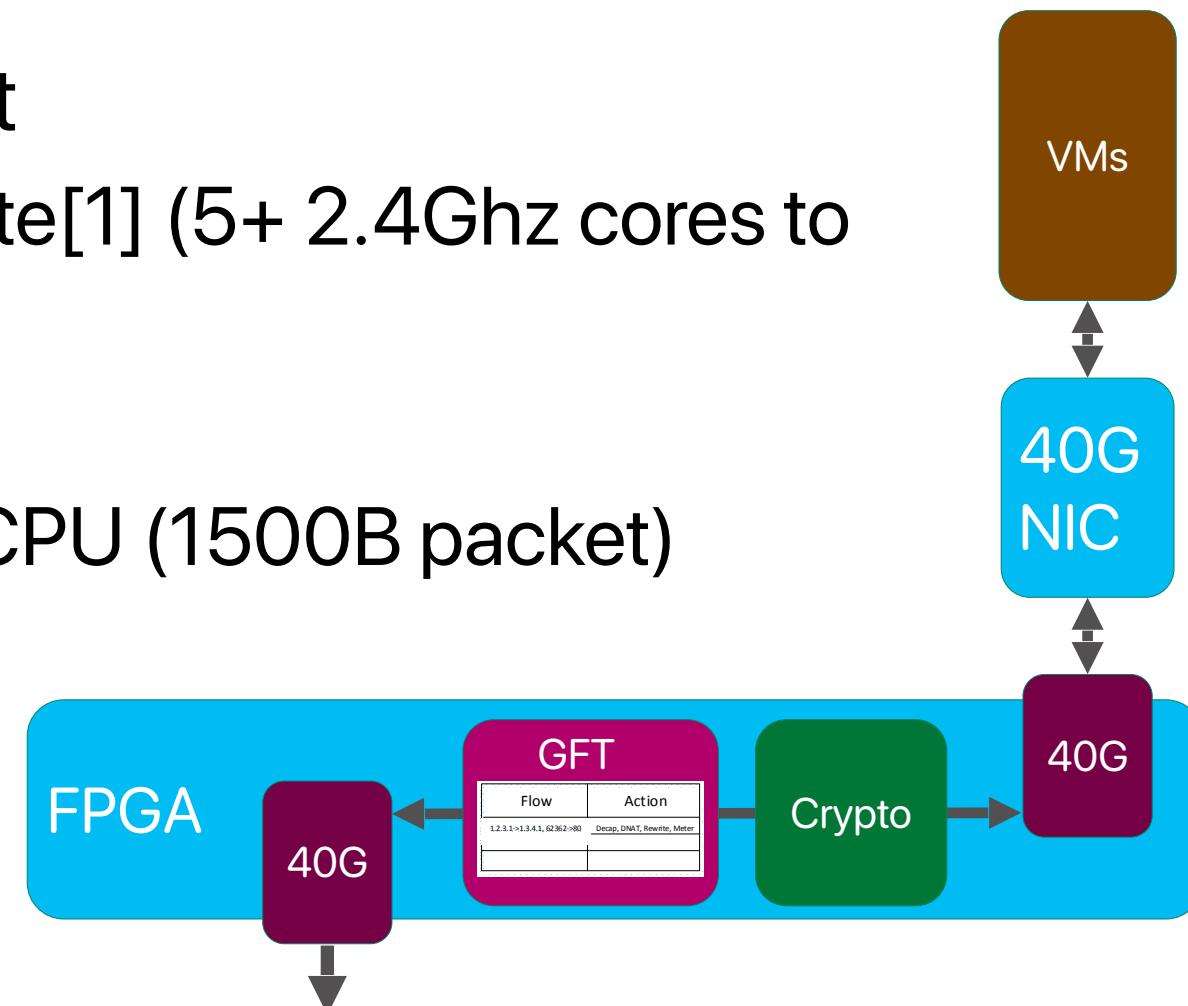


Even at 2x query load,  
accelerated ranking has  
lower latency than software  
**at any load**



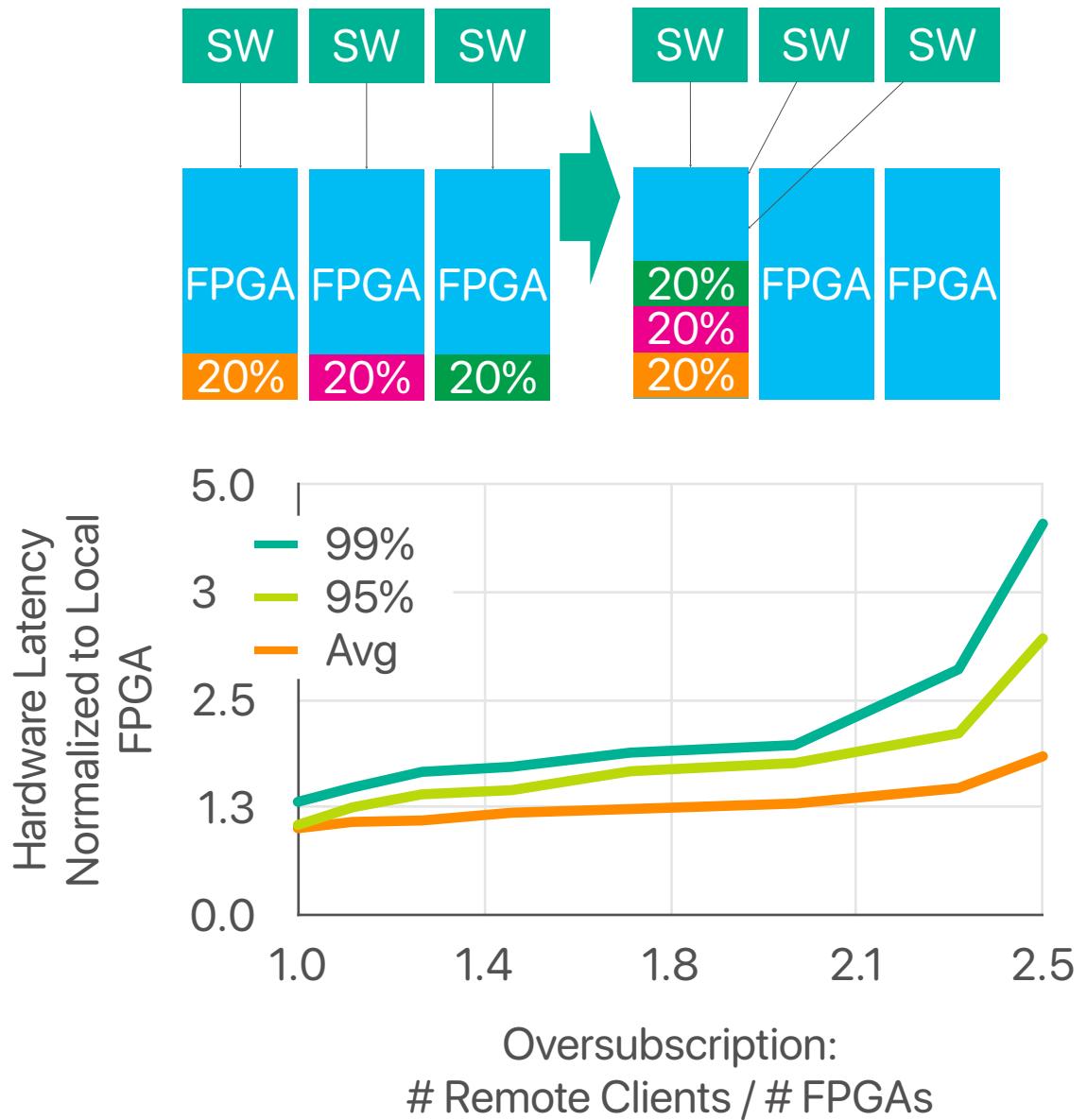
# Accelerated networking

- 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 25 $\mu$ s 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 11 $\mu$ s in FPGA vs 4 $\mu$ s 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 FPPGA 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

These programmable architectures allow for hardware homogeneity while allowing fungibility via software for different services. They must be highly **flexible** at the system level, to

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-

# 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.

**Thank you all for this great quarter!  
Let's take a group photo now!**

**One more thing...**

# **Sample Final**

# Format of the final

- Multiple choices (18 questions)
  - They're like your clicker/midterm multiple choices questions
  - Cumulative, don't forget your midterm and midterm review
- Free answer and open-ended questions \* 2 problem sets, 8 questions in total
  - Two of them are also MSCS comprehensive exam questions
  - Explain your answer clearly and "correctly". If you include "incorrect" information in your answer, it's going to hurt your grade
  - May not have a standard answer. You need to understand the concepts to provide a good answer
  - You should review your assignments carefully

# **Multiple choices**

# How many dependencies do we have?

- How many pairs of data dependences are there in the following x86 instructions?

```
movl    (%rdi), %eax  
xorl    (%rsi), %eax  
movl    %eax, (%rdi)  
xorl    (%rsi), %eax  
movl    %eax, (%rsi)  
xorl    %eax, (%rdi)
```

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

# The effect of code optimization

- By reordering which pair of the following instruction stream can we eliminate all stalls without affecting the correctness of the code?
  - ① `movl (%rdi), %ecx`
  - ② `addl %ecx, %eax`
  - ③ `addq $4, %rdi`
  - ④ `cmpq %rdx, %rdi`
  - ⑤ `jne .L3`
  - ⑥ `ret`
  - A. (1) & (2)
  - B. (2) & (3)
  - C. (3) & (4)
  - D. (4) & (5)
  - E. None of the pairs can be reordered

# CMP advantages

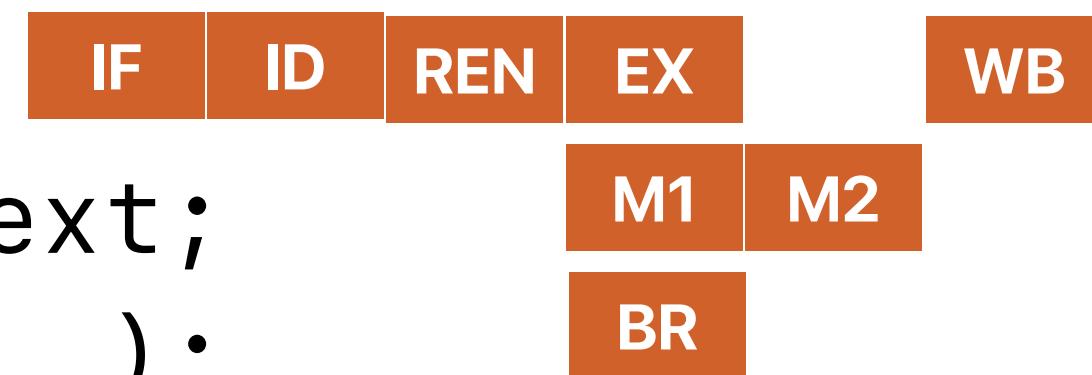
- How many of the following are advantages of CMP over traditional superscalar processor
    - ① CMP can provide better energy-efficiency within the same area
    - ② CMP can deliver better instruction throughput within the same die area (chip size)
    - ③ CMP can achieve better ILP for each running thread
    - ④ CMP can improve the performance of a single-threaded application without modifying code
- A. 0  
B. 1  
C. 2  
D. 3  
E. 4

# What about “linked list”

- Assume the current PC is already at instruction (1) and this linked list has only three nodes. This processor can fetch and issue 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

Which of the following C state of the code snippet determines the performance?

- A. do {
- B.     number\_of\_nodes++;
- C.     current = current->next;
- D. } while ( current != NULL );



# Amdahl's Law on Multicore Architectures

- Regarding Amdahl's Law on multicore architectures, how many of the following statements is/are correct?
  - ① If we have unlimited parallelism, the performance of each parallel piece does not matter as long as the performance slowdown in each piece is bounded
  - ② With unlimited amount of parallel hardware units, single-core performance does not matter anymore
  - ③ With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts
  - ④ With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor

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

## Virtual indexed, physical tagged cache limits the cache size

- If you want to build a virtual indexed, physical tagged cache with 32KB capacity, which of the following configuration is possible? Assume the system use 4K pages.
  - A. 32B blocks, 2-way
  - B. 32B blocks, 4-way
  - C. 64B blocks, 4-way
  - D. 64B blocks, 8-way

# Power & Energy

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

A. 0

B. 1

C. 2

D. 3

E. 4

# Why is D better than C?

- How many of the following statements explains the main reason why B outperforms C with compiler optimizations
  - D has lower dynamic instruction count than C
  - D has significantly lower branch mis-prediction rate than C
  - D has significantly fewer branch instructions than C
  - D can incur fewer memory accesses than C

A. 0

```
inline int __popcount(uint64_t x) {  
    int c = 0;  
    int table[16] = {0, 1, 1, 2, 1,  
                    2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};  
    while(x) {  
        c += table[(x & 0xF)];  
        x = x >> 4;  
    }  
    return c;  
}
```

B. 1

C. 2

D. 3

E. 4



```
inline int __popcount(uint64_t x) {  
    int c = 0;  
    int table[16] = {0, 1, 1, 2, 1,  
                    2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};  
    for (uint64_t i = 0; i < 16; i++) {  
        c += table[(x & 0xF)];  
        x = x >> 4;  
    }  
    return c;  
}
```

# Demo revisited

- Why the performance is better when option is not “0”
    - ① The amount of dynamic instructions needs to execute is a lot smaller
    - ② The amount of branch instructions to execute is smaller
    - ③ The amount of branch mis-predictions is smaller
    - ④ The amount of data accesses is smaller
- A. 0   **if(option)**  
          std::sort(data, data + arraySize);
- B. 1   **for (unsigned i = 0; i < 100000; ++i) {**
- C. 2    int **threshold** = std::rand();
- D. 3    **for (unsigned i = 0; i < arraySize; ++i) {**  
          **if (data[i] >= threshold)**  
            sum ++;
- E. 4    **}**  
        **}**

# Why can't we proceed without stalls/no-ops?

- How many of the following statements are true regarding why we have to stall for each branch in the current pipeline processor
    - ① The target address when branch is taken is not available for instruction fetch stage of the next cycle
    - ② The target address when branch is not-taken is not available for instruction fetch stage of the next cycle
    - ③ The branch outcome cannot be decided until the comparison result of ALU is not out
    - ④ The next instruction needs the branch instruction to write back its result
- A. 0  
B. 1  
C. 2  
D. 3  
E. 4

# What if the code look like this?

- D-L1 Cache configuration of NVIDIA Tegra X1
  - Size 64KB, 4-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    c[i] = a[i]; //load a and then store to c
for(i = 0; i < 512; i++)
    c[i] += b[i]; //load b, load c, add, and then store to c
```

What's the data cache miss rate for this code?

- A. 6.25%
- B. 56.25%
- C. 66.67%
- D. 68.75%
- E. 100%

# What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {  
    for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {  
        for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {  
            for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)  
                for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)  
                    for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)  
                        c[ii][jj] += a[ii][kk]*b[kk][jj];  
        }  
    }  
}
```

- A. Compulsory miss
- B. Capacity miss
- C. Conflict miss
- D. Capacity & conflict miss
- E. Compulsory & conflict miss

Block + Transpose

```
// Transpose matrix b into b_t  
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {  
    for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {  
        b_t[i][j] += b[j][i];  
    }  
}  
  
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {  
    for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {  
        for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {  
            for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)  
                for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)  
                    for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)  
                        // Compute on b_t  
                        c[ii][jj] += a[ii][kk]*b_t[jj][kk];  
        }  
    }  
}
```

# MS' "Configurable Clouds"

- Regarding MS' configurable clouds that are powered by FPGAs, please identify how many of the following are correct
    - ① Each FPGA is dedicated to one machine
    - ② Each FPGA is connected through a network that is separated from the data center network
    - ③ FPGA can deliver shorter average latency for AES-CBC-128-SHA1 encryption and decryption than Intel's high-end processors
    - ④ FPGA-accelerated search queries are always faster than a pure software-based datacenter
- A. 0  
B. 1  
C. 2  
D. 3  
E. 4

# Summary of Optimizations

- Regarding the following cache optimizations, how many of them would help improve miss rate?
  - ① Non-blocking/pipelined/multibanked cache
  - ② Critical word first and early restart
  - ③ Prefetching
  - ④ Write buffer

A. 0

B. 1

C. 2

D. 3

E. 4

# What data structure is performing better

	Array of objects	object of arrays
	<pre>struct grades {     int id;     double *homework;     double average; };</pre>	<pre>struct grades {     int *id;     double **homework;     double *average; };</pre>
average of each homework	<pre>for(i=0;i&lt;homework_items; i++) { gradesheet[total_number_students].homework[i] = 0.0;     for(j=0;j&lt;total_number_students;j++) gradesheet[total_number_students].homework[i] +=gradesheet[j].homework[i];     gradesheet[total_number_students].homework[i] /= (double)total_number_students; }</pre>	<pre>for(i = 0;i &lt; homework_items; i++) {     gradesheet.homework[i][total_number_students] = 0.0;     for(j = 0; j &lt;total_number_students;j++)     {         gradesheet.homework[i][total_number_students] += gradesheet.homework[i][j];     }     gradesheet.homework[i][total_number_students] /= total_number_students; }</pre>

- Considering your workload would like to calculate the average score of **one of the homework for all students**, which data structure would deliver better performance?
  - Array of objects
  - Object of arrays

# 3Cs and A, B, C

- Regarding 3Cs: compulsory, conflict and capacity misses and  
A, B, C: associativity, block size, capacity

How many of the following are correct?

- ① Increasing associativity can reduce conflict misses
- ② Increasing associativity can reduce hit time
- ③ Increasing block size can increase the miss penalty
- ④ Increasing block size can reduce compulsory misses

A. 0

B. 1

C. 2

D. 3

E. 4

# Cache coherency

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

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

- ① 0123456789
- ② 1259368101213
- ③ 1111111164100
- ④ 111111111100
- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

# 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;
}
```

- L is better, because the cache miss rate is lower
- R is better, because the cache miss rate is lower
- L is better, because the instruction count is lower
- R is better, because the instruction count is lower
- Both are about the same

### FalseSharing

## 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);
```

# **Free-answer questions**

# Register renaming

- For the following x86 instructions:

①	.L3:	addq	\$8, %rdi
②		movq	(%rdi), %rdi
③		addl	\$1, %eax
④		testq	%rdi, %rdi
⑤		jne	.L3

- Assume we have a dual-fetch, dual-issue, out-of-order pipeline where
  - INT ALU takes 1 cycle
  - MEM pipeline: AR-AQ-MEM — 3 cycles in total
  - BR takes 1 cycle to resolve
- If the loop is taken twice, how many cycles it takes to issue all instructions?
- If the loop is taken 100 times, what's the average CPI?

# Best cache configuration

- Consider the following code. Integers and pointers are both 4 bytes.

```
struct List {  
    List * next;  
    int data;  
}  
  
void foo(List *head) {  
    List * cur = head;  
    while(cur->next) {  
        cur = cur->next;  
    }  
}
```

- For a given total cache size, what cache line size will provide the best performance for this code?  
(hint: Your answer should not depend on the number of lines or the associativity of the cache.)

# Reverse caching

- Below, we have given you four different sequences of addresses generated by a program running on a processor with a data cache. Cache hit ratio for each sequence is also shown below. Assuming that the cache is initially empty at the beginning of each sequence, find out the following parameters of the processor's data cache (ensure that you sufficiently explain your answer)
    - Associativity (1, 2, or 4 ways)
    - Block size (1, 2, 4, 8, 16, or 32 bytes)
    - Total cache size (256B, or 512B)
    - Replacement policy (LRU or FIFO)
- Address Sequence 1: [0, 2, 4, 8, 16, 32] Hit Ratio: 0.33
  - Address Sequence 2: [0, 512, 1024, 1536, 2048, 1536, 1024, 512, 0] Hit Ratio: 0.33
  - Address Sequence 3: [0, 64, 128, 256, 512, 256, 128, 64, 0] Hit Ratio: 0.33
  - Address Sequence 4: [0, 512, 1024, 0, 1536, 0, 2048, 512] Hit Ratio: 0.25

# **Open-ended questions**

# Code and cache miss rate

- Assume my cache has 16KB capacity, 16 byte block size and is 2-way set associative. Integers are 4 bytes. Give the C code for a loop that has a very poor hit rate in this cache but whose hit rate raises to almost 100% if we double the capacity to 32KB.

# Branch predictions

- Increasing the size of a branch predictor typically reduces the chances of "aliasing" -- two branches sharing the same predictor. Usually, sharing results in negative interference (decreased prediction accuracy), but sometimes it can result in positive interference.  
Assuming a PC-indexed table of 2-bit predictors
  - Give an example of two branches (eg, show the T, N patterns for each, and how they are interleaved) that would result in positive interference (increased overall prediction accuracy).
  - Give an example of two branches that would result in negative interference.
  - Explain why most of the time you would expect to see negative interference with real code.

# SMT v.s. CMP

- Both CMP & SMT exploit thread-level or task-level parallelism. Assuming both application X and application Y have similar instruction combination, say 60% ALU, 20% load/store, and 20% branches. Consider two processors:

P1: CMP with a 2-issue pipeline on each core. Each core has a private L1 32KB D-cache

P2: SMT with a 4-issue pipeline. 64KB L1 D-cache

Which one do you think is better?

- A. P1
- B. P2

# Other open-ended questions

- Given the instruction front-end is decoupled from the backend of the pipeline ALUs, do you think ISA still affect performance?
  - Emily Blehm, Jaikrishnan Menon, and Karthikeyan Sankaralingam. 2013. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA) (HPCA '13). <https://minds.wisconsin.edu/handle/1793/64923>
  - Ashish Venkat and Dean M. Tullsen. 2014. Harnessing ISA diversity: design of a heterogeneous-ISA chip multiprocessor. In Proceeding of the 41st annual international symposium on Computer architecuture (ISCA '14). <http://www.cs.virginia.edu/venkat/papers/isca2014.pdf>
- What features in modern processor architecture enable the potential of "Meltdown and Spectre" attacks? Should we live without those features? How to solve these security issues?
  - M. D. Hill, J. Masters, P. Ranganathan, P. Turner and J. L. Hennessy, "On the Spectre and Meltdown Processor Security Vulnerabilities," in IEEE Micro, vol. 39, no. 2. [http://pages.cs.wisc.edu/~markhill/papers/ieeemicro19\\_spectre\\_meltdown\\_2019\\_01\\_30](http://pages.cs.wisc.edu/~markhill/papers/ieeemicro19_spectre_meltdown_2019_01_30)
- What compiler optimizations would not be effective given OoO execution hardware?

# Other open-ended questions

- Can you name and briefly describe a few “trends” in the dark silicon era?
- If you’re asked to design a machine learning hardware, what will you do?
- If you’re asked to build an Xeon Phi type processor where each core also has many-way SMT, are you going to give the processor more cache or better branch predictor?
- Can we focus on improving the throughput of computing instead of latency? Can you give an example on what type of applications will not work well in this way
- Pros and cons for branch prediction using perceptrons?

# Announcement — final exam

- Final Exam
  - The final can be opened only once -- if you accidentally close the browser or the browser crashes or your lose Internet connection, you cannot re-initiate it and we WILL NOT help you for these cases. Browsers crash and accidental closing of tabs occur a lot when you have many opened tabs. Please be careful.
  - Q2 and Q3 are comprehensive exam questions -- You must receive at least 60% from Q2 AND 60% from Q3 to be considered as PASS
  - This final covers EVERYTHING mentioned/assigned this quarter.
  - This is an open-book, open-note test, but again, the more you open, the higher chance your computer will have issues.
  - We have MANY questions for you, but you only have a total of 180 minutes to finish. Heavily rely on your notes/book/cheatsheets is not a good idea.
  - Please show your work if appropriate -- we don't give credits to answers only have the final result
  - There is no partial credits for multiple choice questions. Please think thoroughly.
  - Reference online solution, discuss with ANY other human being or digital assistant (e.g, Siri, Google Home, Alexa or whatever you name it) is considered as cheating.

# Announcement

- According to the log, only 20 unique ones have started.
- 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
- Last office hours
  - Hung-Wei Tseng: 12/1 11a-12p (additional one)
  - Jinyoung Choi: 12/1 2p-3p

# Computer Science & Engineering

203

# Finale

