Dusk and dawn: dark silicon and the new golden age of computer architecture

Hung-Wei Tseng

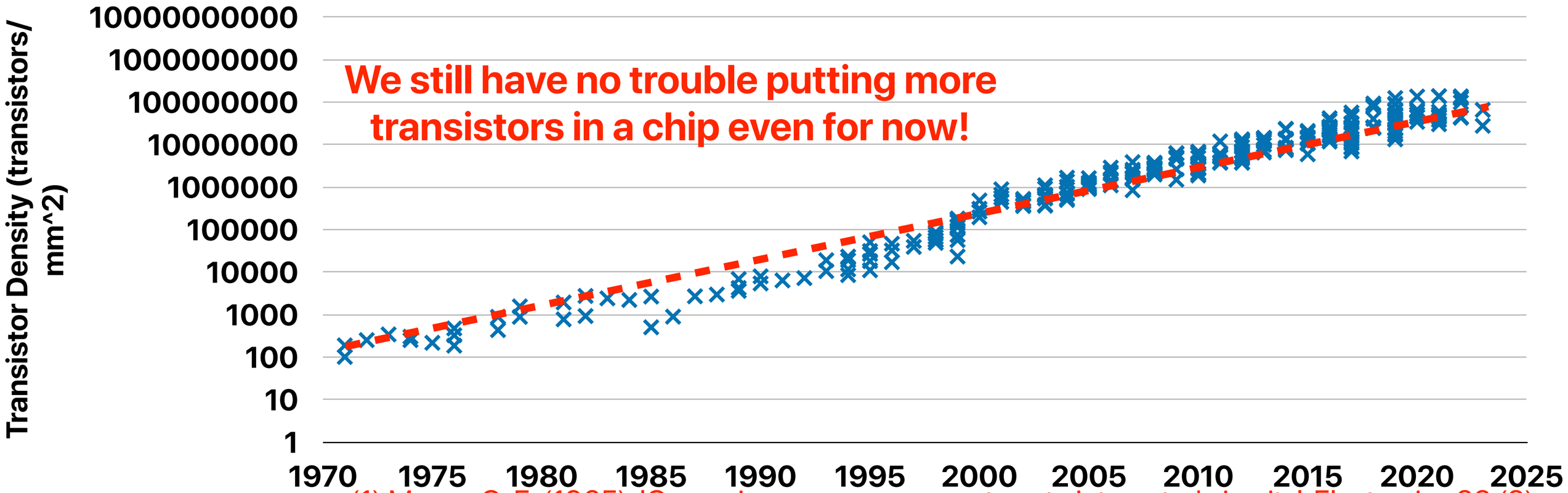
Outline

- The dark silicon problem
- Challenges and state-of-the-art solutions in the dark silicon era
- The new golden age of Computer Architecture

What is and why is dark silicon?

Moore's Law[1]

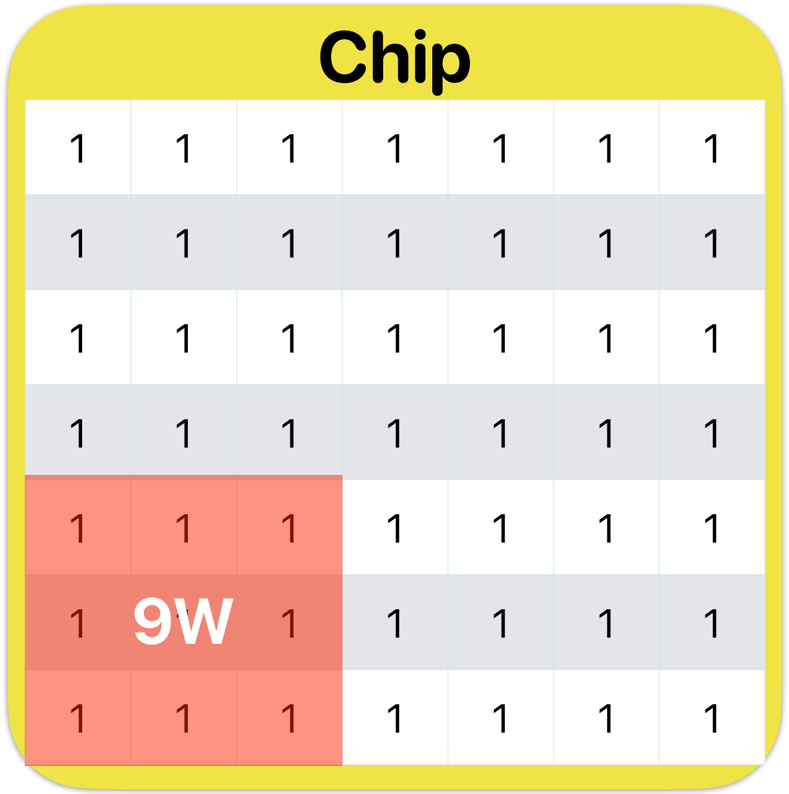
- The number of transistors we can build in a fixed area of silicon doubles every 12 ~ 24 months.
- Moore's Law "was" the most important driver for historic CPU performance gains



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

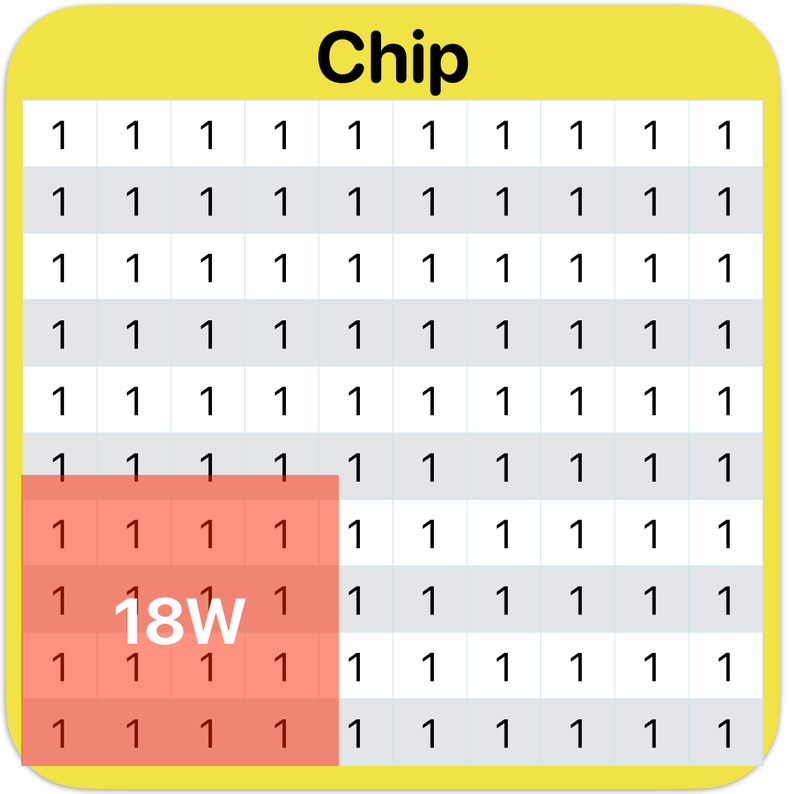
Plot based on https://en.wikipedia.org/wiki/Transistor_count by Hung-Wei Tseng

Power consumption to light on all transistors



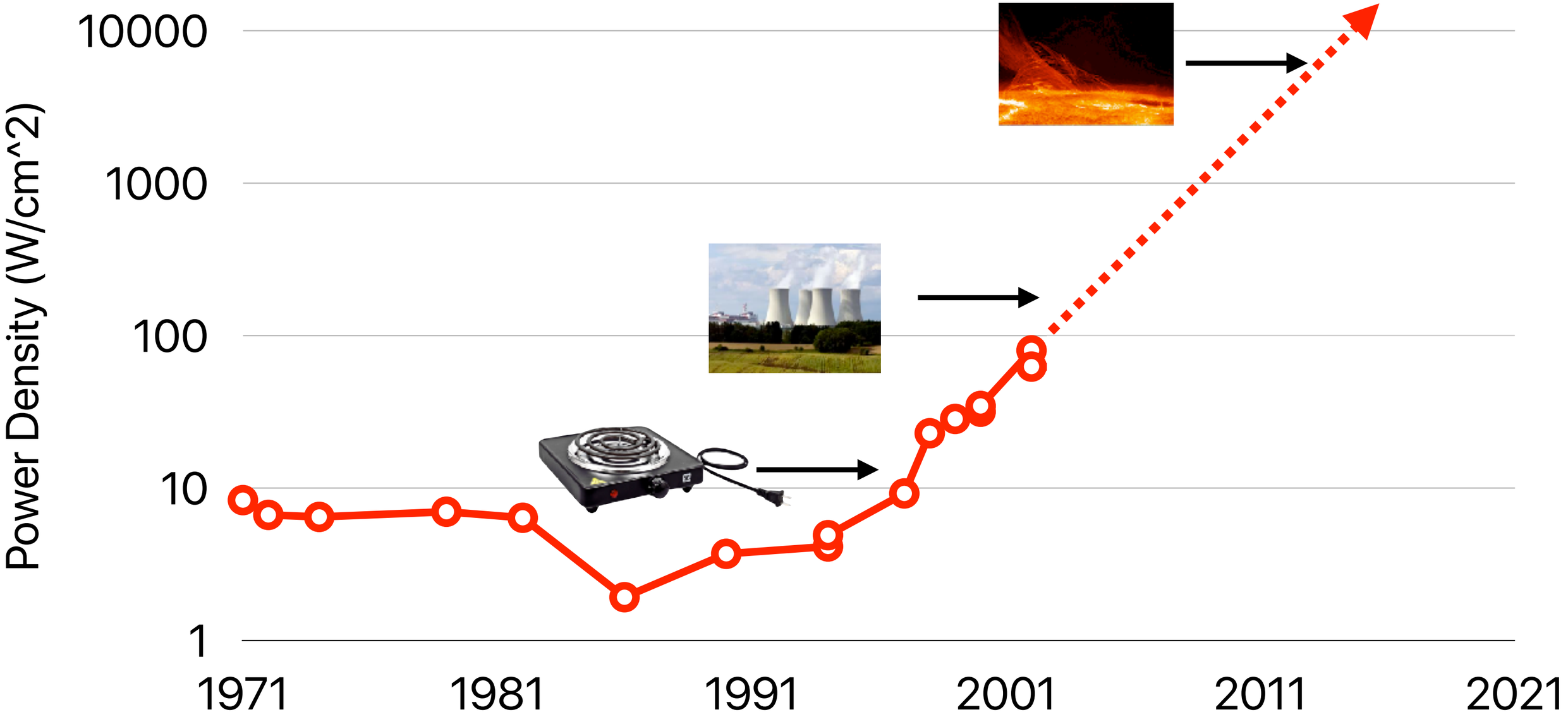
=49W

Dennardian Broken

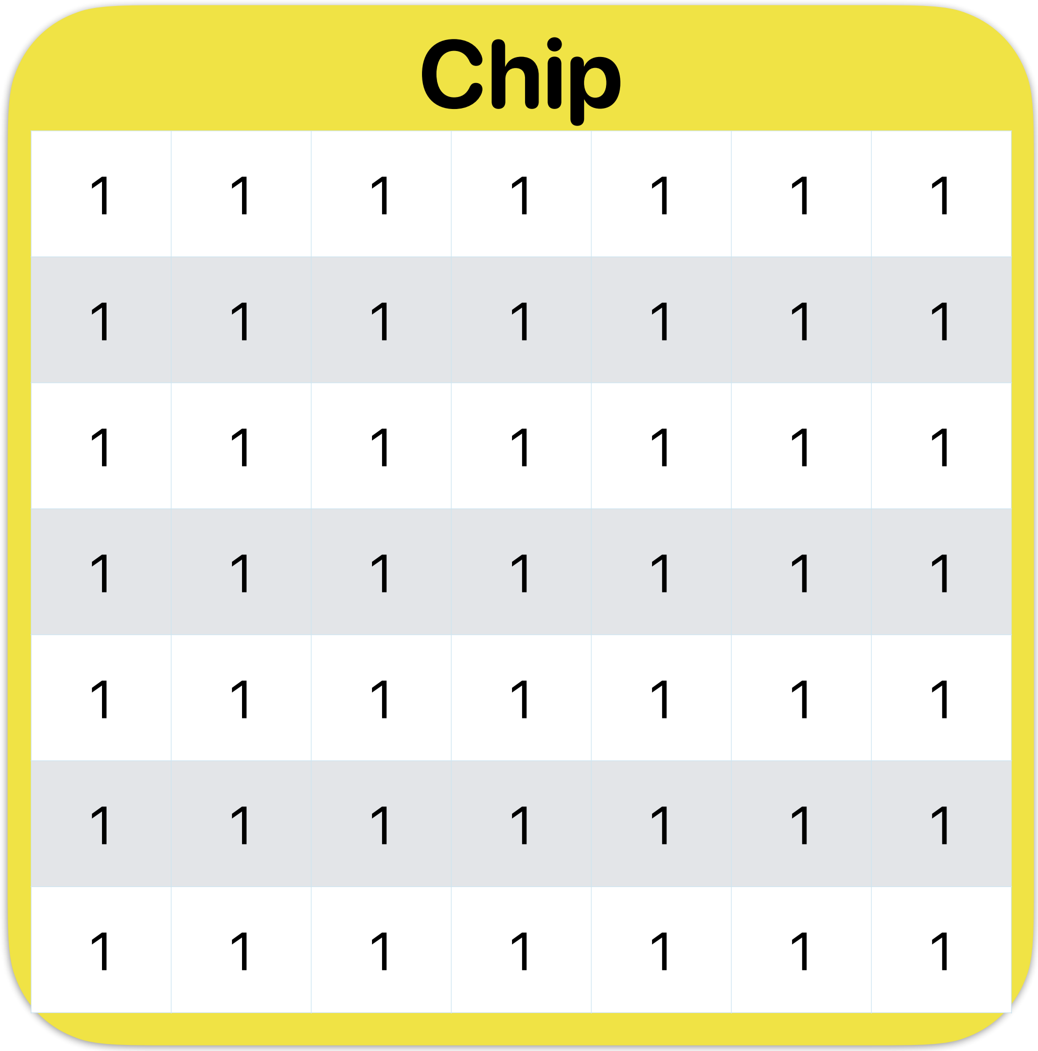


=100W!

Power Density of Processors

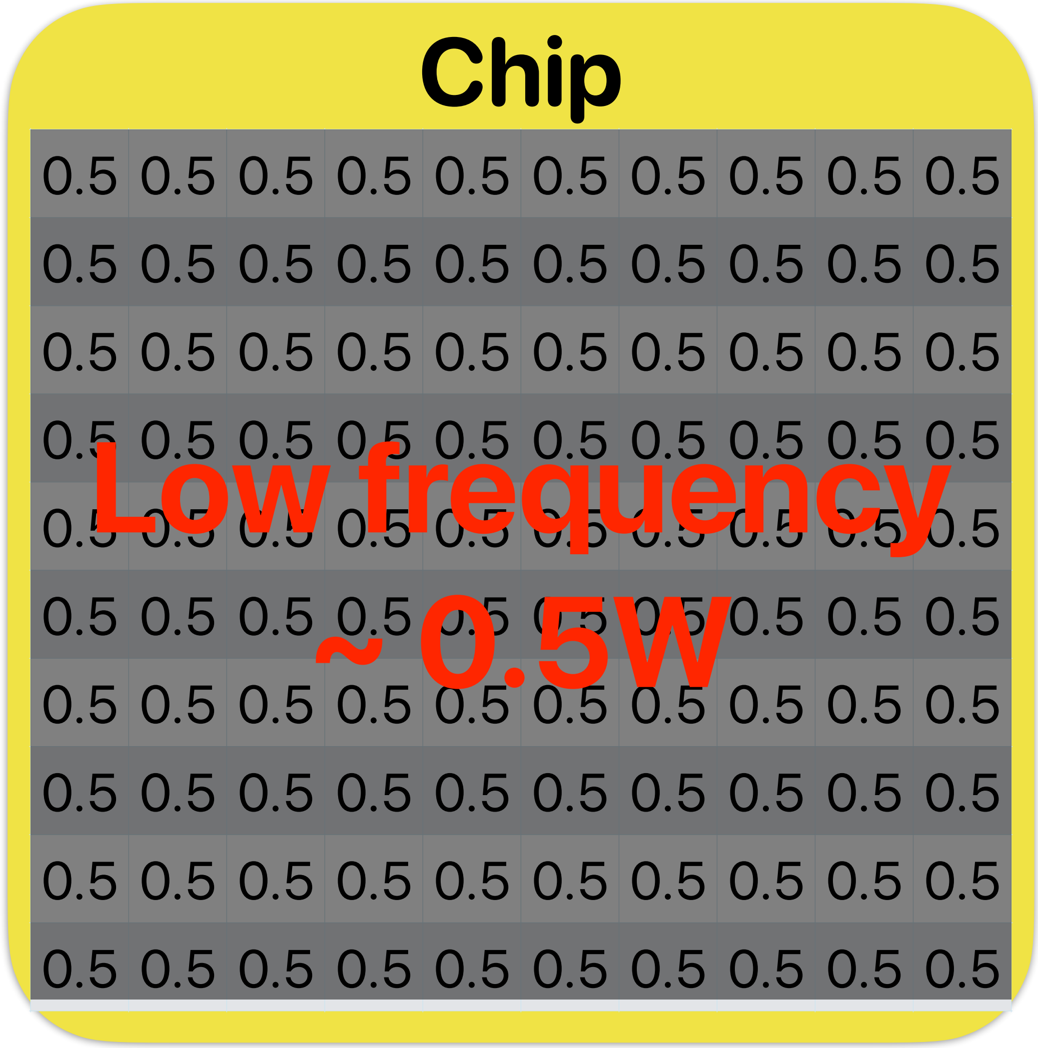


If we have a fixed power budget



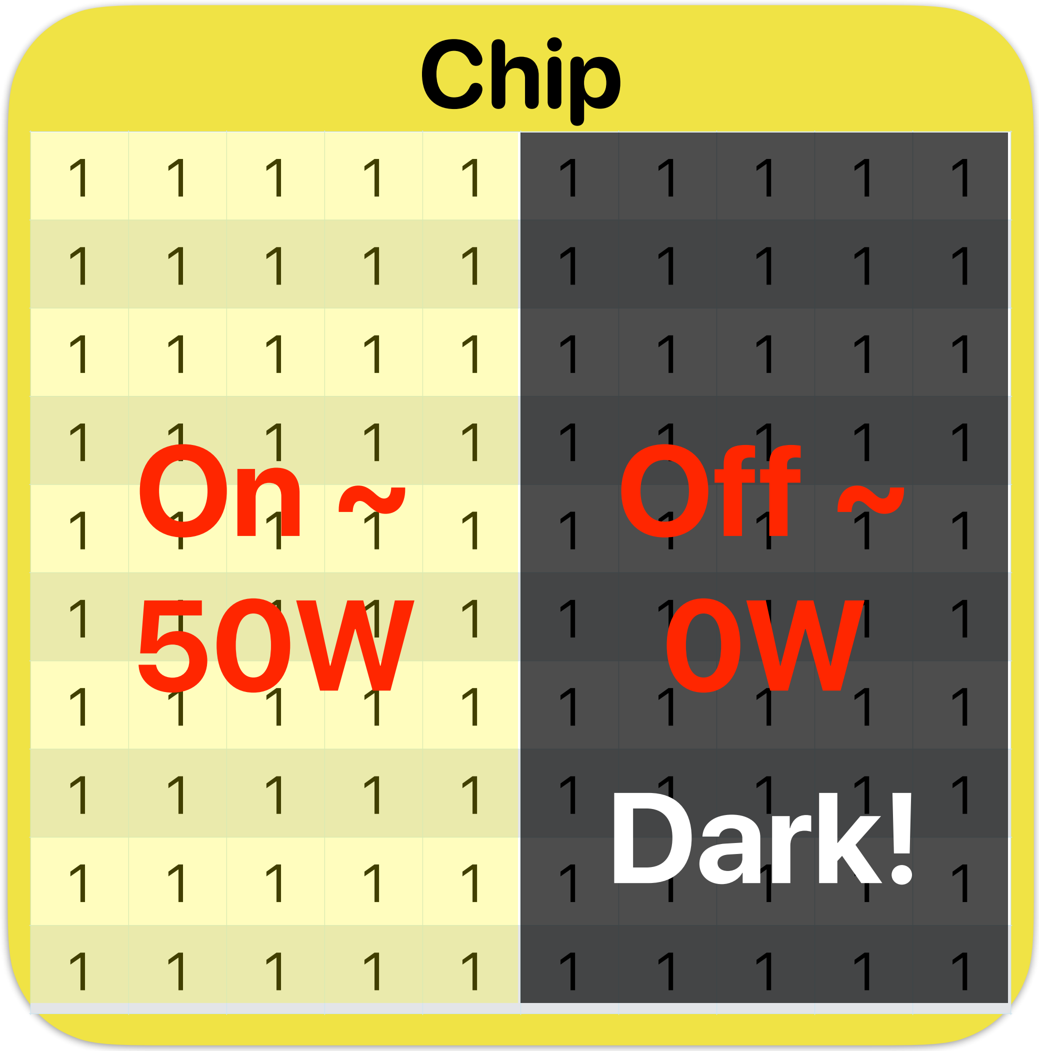
=49W

Slowdown all of them



=50W!

Dark silicon!



=50W!

Power consumption & power density

- $P_{dynamic} \sim \alpha \times C \times V^2 \times f \times N$
  - α: average switches per cycle
  - C: capacitance
  - V: voltage
  - f: frequency, usually linear with V
  - N: the number of transistors
- $P_{leakage} \sim N \times V \times e^{-V}$
  - N: number of transistors
  - V: voltage
  - V_t: threshold voltage where transistor conducts (begins to switch)
- Power density:

$$P_{density} = \frac{P}{area}$$

Dennard scaling discontinued — we cannot make voltage lower

Moore's Law allows higher frequencies as transistors are smaller

Moore's Law makes this smaller

Recap — Take-aways: parallel programming

- Cache coherency only guarantees that everyone would eventually have a coherent view of data, but not when
- Cache coherency may create unexpected cache invalidations/misses if you do it wrong
- Processor behaviors are non-deterministic
  - You cannot predict which processor is going faster
  - You cannot predict when OS is going to schedule your thread
  - You cannot predict when the processor is going to schedule an instruction
- Cache consistency is hard to support
- **Even if we can address all programming challenges, multi-core performance has stopped to scale due to the Dark Silicon problem**

Dark silicon problem

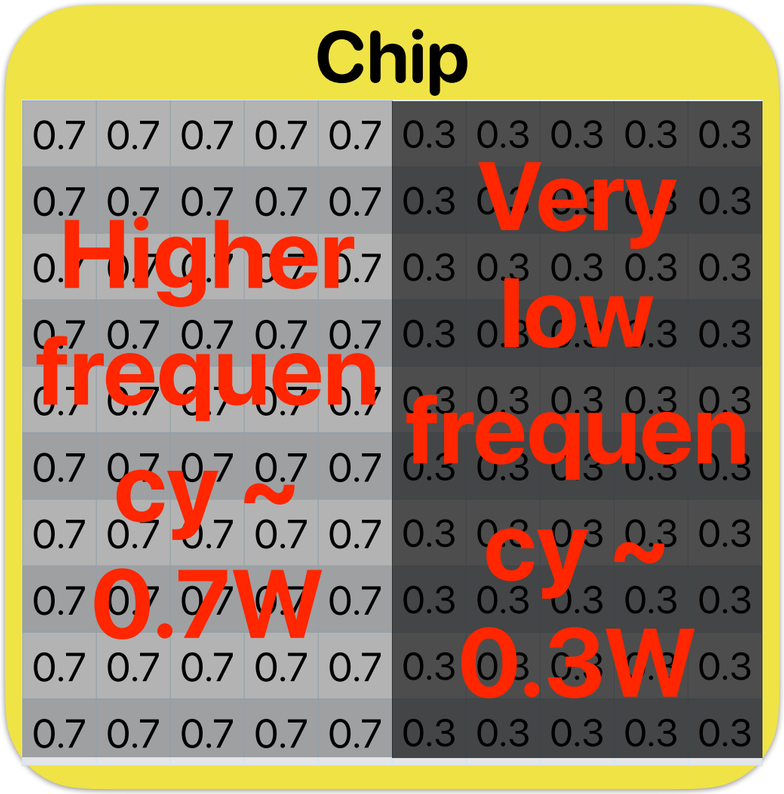
Dark silicon problem

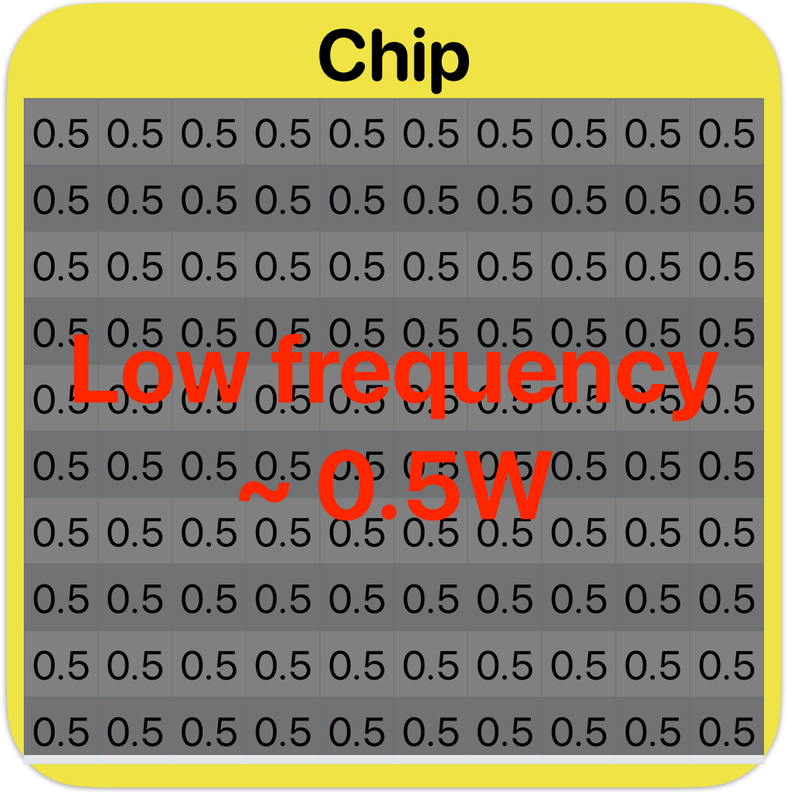
- We are given the same area, twice amount of transistors
- We are given the same power budget
- We are given a certain applications
- How can we maximize the performance for these applications within the given constraints?

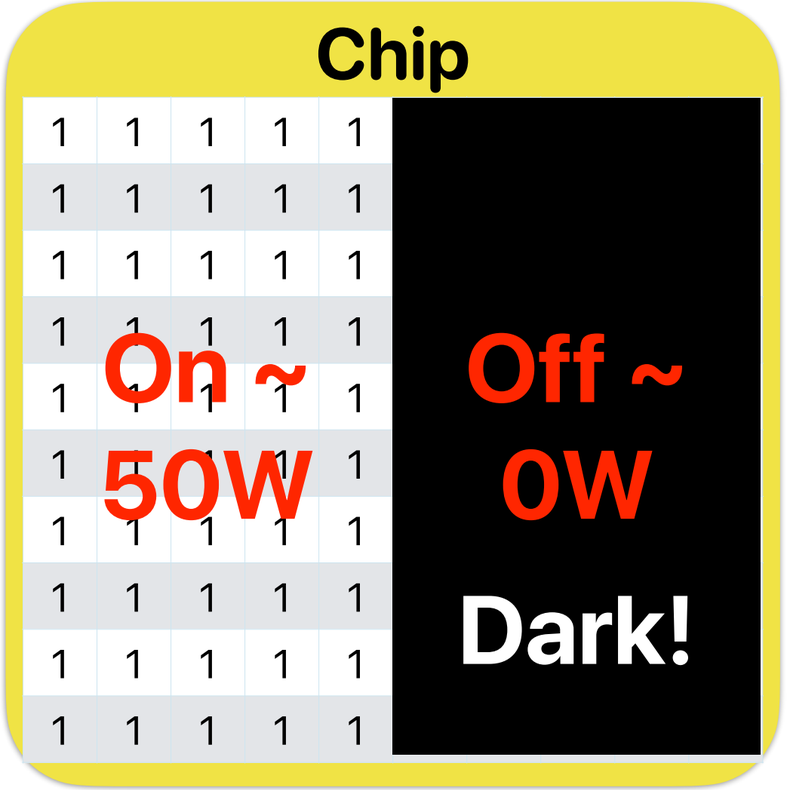
Given the same power budget, maximize the efficiency per chip

Some faster,
some slower

Some at top speed,
Slowdown all of themsome are not functioning







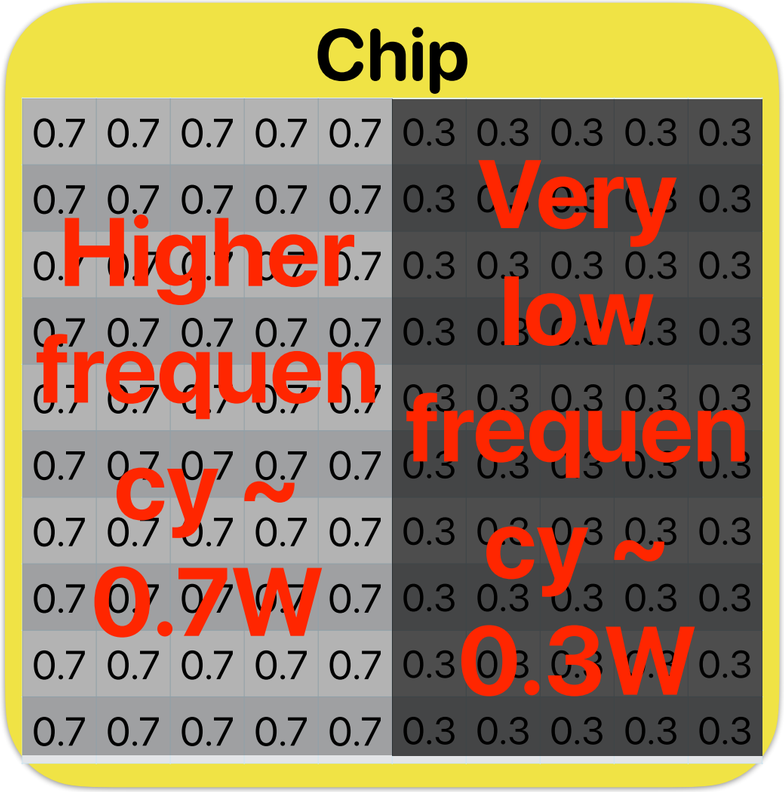
=50W!

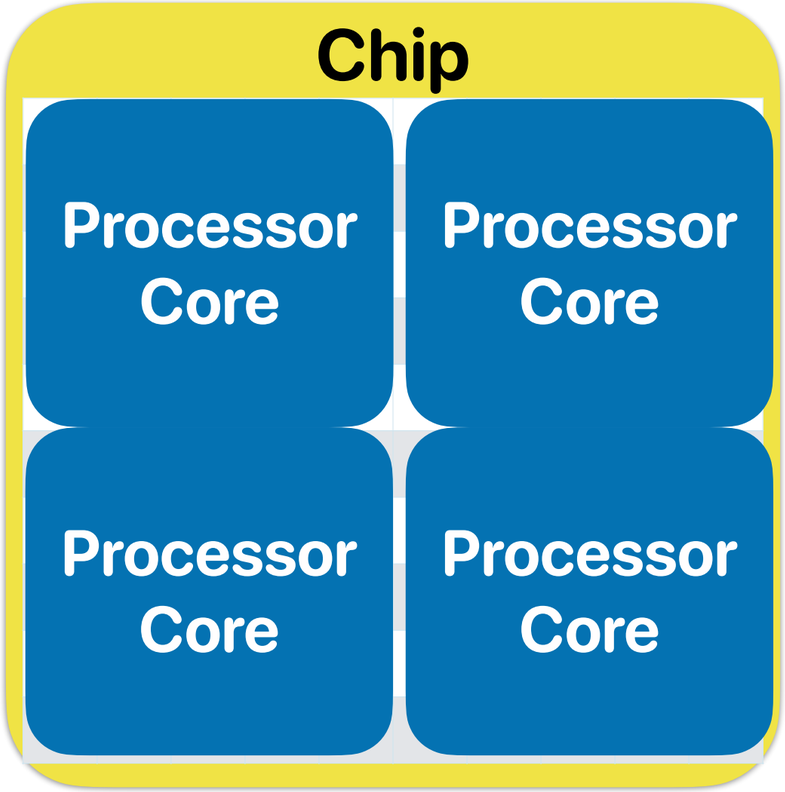
=50W!

=50W!

Given the same power budget, maximize the efficiency per chip

Some faster, some slower





Aggressively adjust the frequency of processor cores — If only one program is compute-intensive, having it running at full speed, others at lower frequency or turning off

Modern processor's frequency

```
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           151
Model name:                      12th Gen Intel(R) Core(TM) i7-12700KF
Stepping:                        2
CPU MHz:                         1226.409
CPU max MHz:                     5000.0000
CPU min MHz:                     800.0000
BogoMIPS:                        7219.20
```

	i7-12700K
	Intel 7
	$450.00 - $460.00
	PC/Client/Tablet, Workstation

CPU Specifications

Total Cores	12
# of Performance-cores	8
# of Efficient-cores	4
Total Threads	20
Max Turbo Frequency	5.00 GHz
Intel® Turbo Boost Max Technology 3.0 Frequency ‡	5.00 GHz
Performance-core Max Turbo Frequency	4.90 GHz
Efficient-core Max Turbo Frequency	3.80 GHz
Performance-core Base Frequency	3.60 GHz
Efficient-core Base Frequency	2.70 GHz
Cache	25 MB Intel® Smart Cache

https://www.intel.com/content/www/us/en/products/sku/134594/intel-core-i712700k-processor-25m-cache-up-to-5-00-ghz/specifications.html

Modern processor's frequency

```
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):                          12
On-line CPU(s) list:             0-11
Thread(s) per core:              2
Core(s) per socket:              6
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
CPU family:                      25
Model:                           80
Model name:                      AMD Ryzen 5 5500
Stepping:                        0
Frequency boost:                 enabled
CPU MHz:                         3600.000
CPU max MHz:                     3600.0000
CPU min MHz:                     1400.0000
```

Take-aways: Challenges and SOTA solutions in the dark silicon era

- **Even if we can address all programming challenges, multi-core performance has stopped to scale due to the Dark Silicon problem**
- Aggressive dynamic frequency/voltage scaling on CMP to accommodate the demand of **latency**-sensitive, parallelism-limited applications, but the area-efficiency of the slower cores is not great

More cores per chip, slower per core

Export comparison	Intel® Xeon® Platinum 849...	Intel® Xeon® Platinum 846...	Intel® Xeon® Gold 6448H ...	Intel® Xeon® Platinum 844...	Intel® Xeon® Gold 6434H ...
Total Cores	60	48	32	16	8
Total Threads	120	96	64	32	16
Max Turbo Frequency	3.50 GHz	3.80 GHz	4.10 GHz	4.00 GHz	4.10 GHz
Processor Base Frequency	1.90 GHz	2.10 GHz	2.40 GHz	2.90 GHz	3.70 GHz
Cache	112.5 MB	105 MB	60 MB	45 MB	22.5 MB
Intel® UPI Speed	16 GT/s	16 GT/s	16 GT/s	16 GT/s	16 GT/s
Max # of UPI Links	4	4	3	4	3
TDP	350 W	330 W	250 W	270 W	195 W

https://ark.intel.com/content/www/us/en/ark/products/series/228622/4th-generation-intel-xeon-scalable-processors.html

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

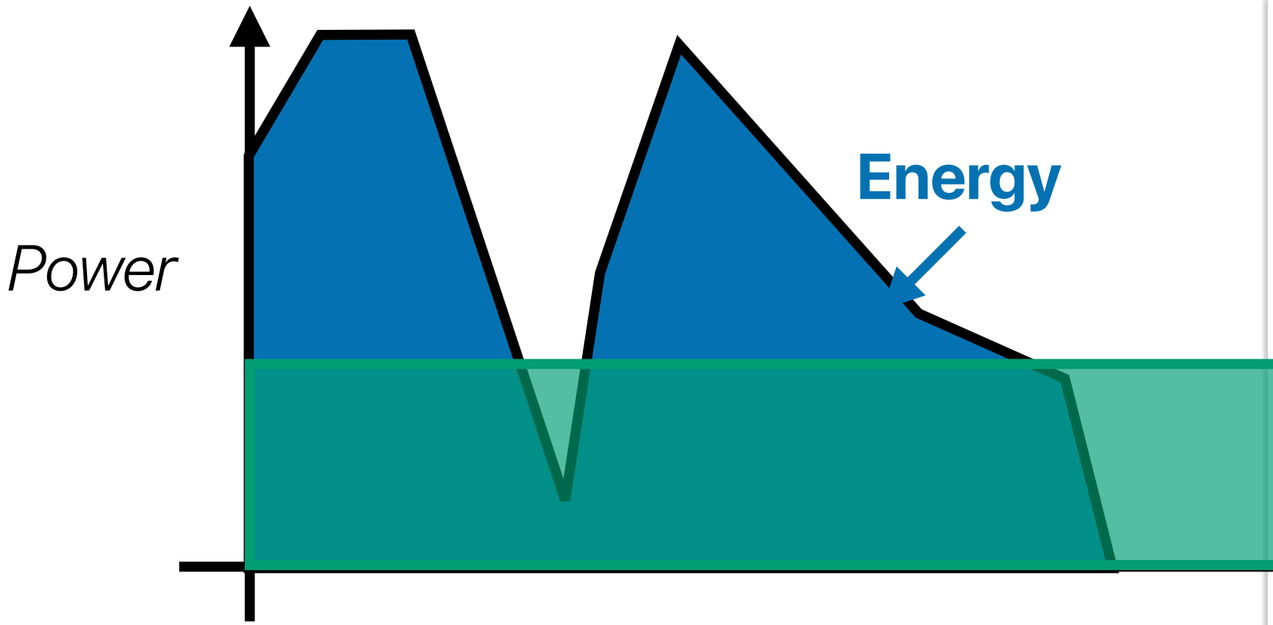
# of Cores	72
# of Threads	72
Processor Base Frequency	1.50 GHz
Max Turbo Frequency	1.50 GHz
Cache	36 MB L2 Cache
TDP	320 W

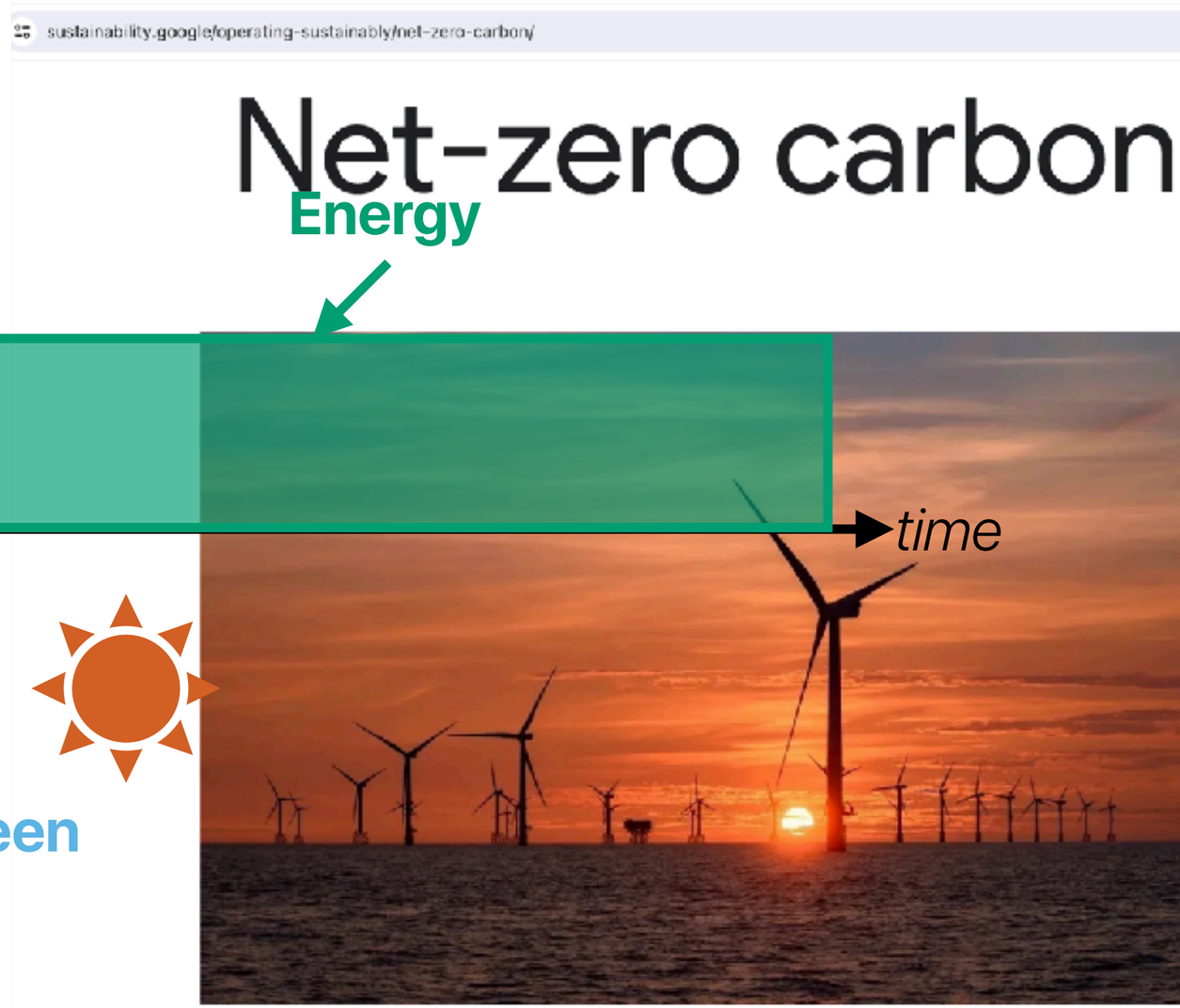
Power v.s. Energy

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

Power/Energy/Carbon footprint

The Green can be more if power is not low enough





If we run the task when there is no green energy— more carbon footprint!

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!

Power v.s. Energy

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

Power consumption & power density

- $P_{dynamic} \sim \alpha \times C \times V^2 \times f \times N$
  - α: average switches per cycle
  - C: capacitance
  - V: voltage
  - f: frequency, usually linear with V
  - N: the number of transistors
- $P_{leakage} \sim N \times V \times e^{-V_t}$
  - N: number of transistors
  - V: voltage
  - V_t: threshold voltage where transistor conducts (begins to switch)
- Power density:

$$P_{density} = \frac{P}{area}$$

Voltage is a key factor of power consumption

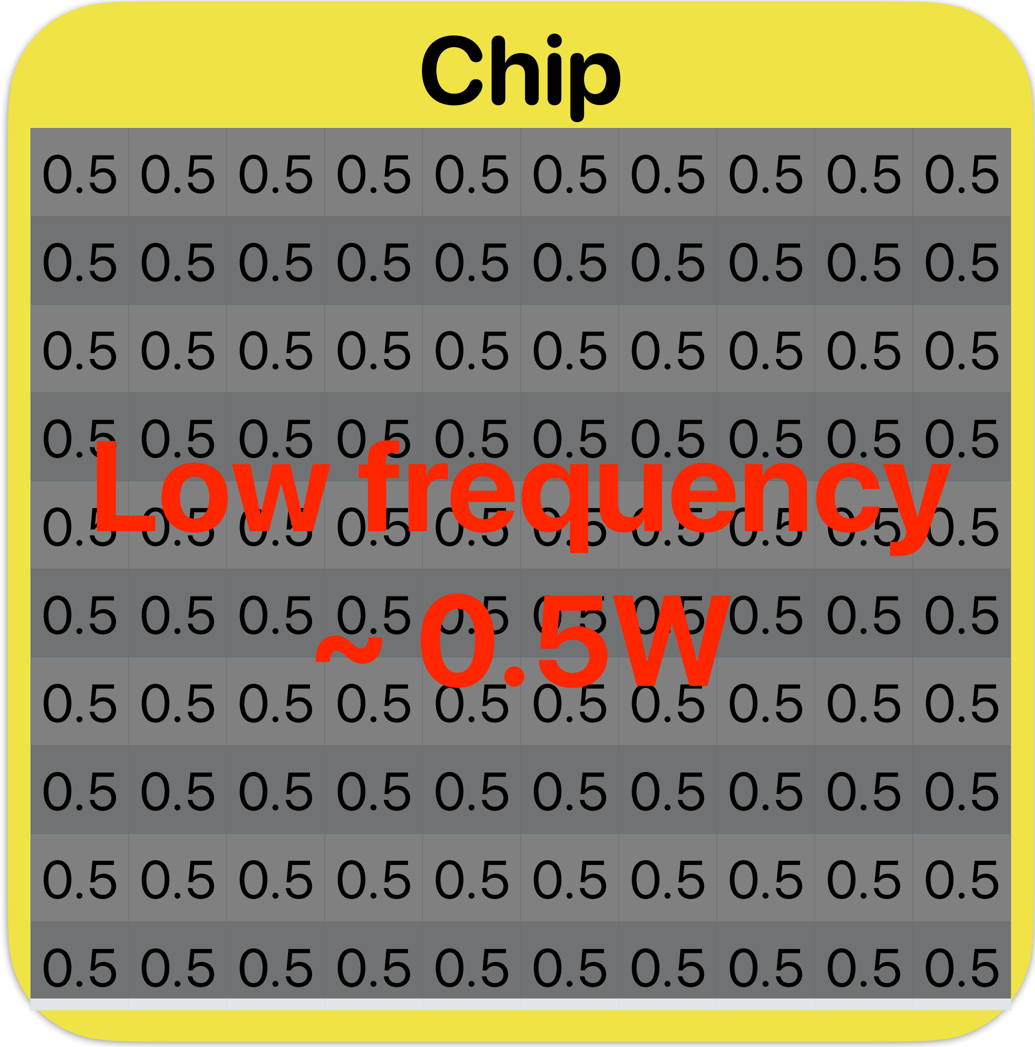
Frequency is depending on the supply voltage

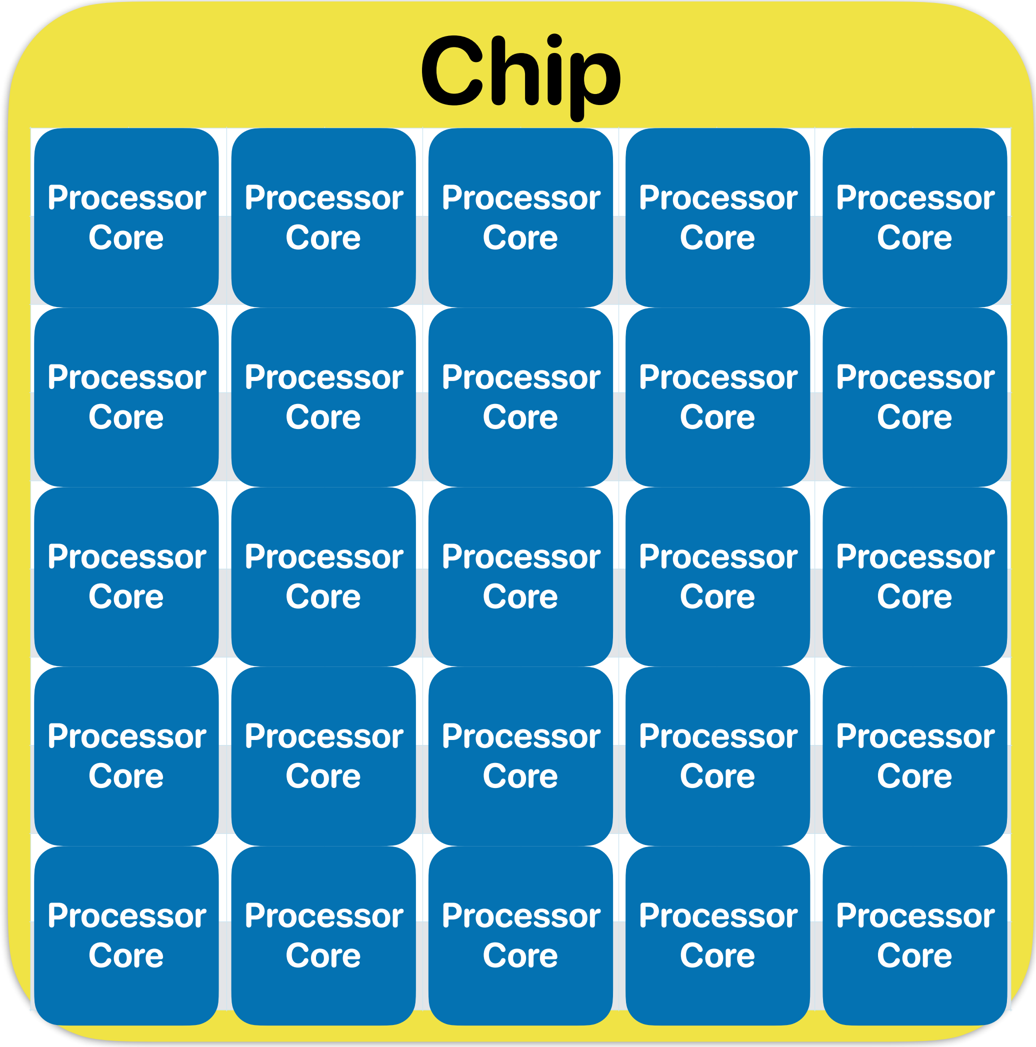
Demo — changing the max frequency and performance

- 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!

Given the same power budget, maximize the efficiency per chip

Slowdown all of them





All of them are smaller, simpler, lower-frequency, lower-power cores

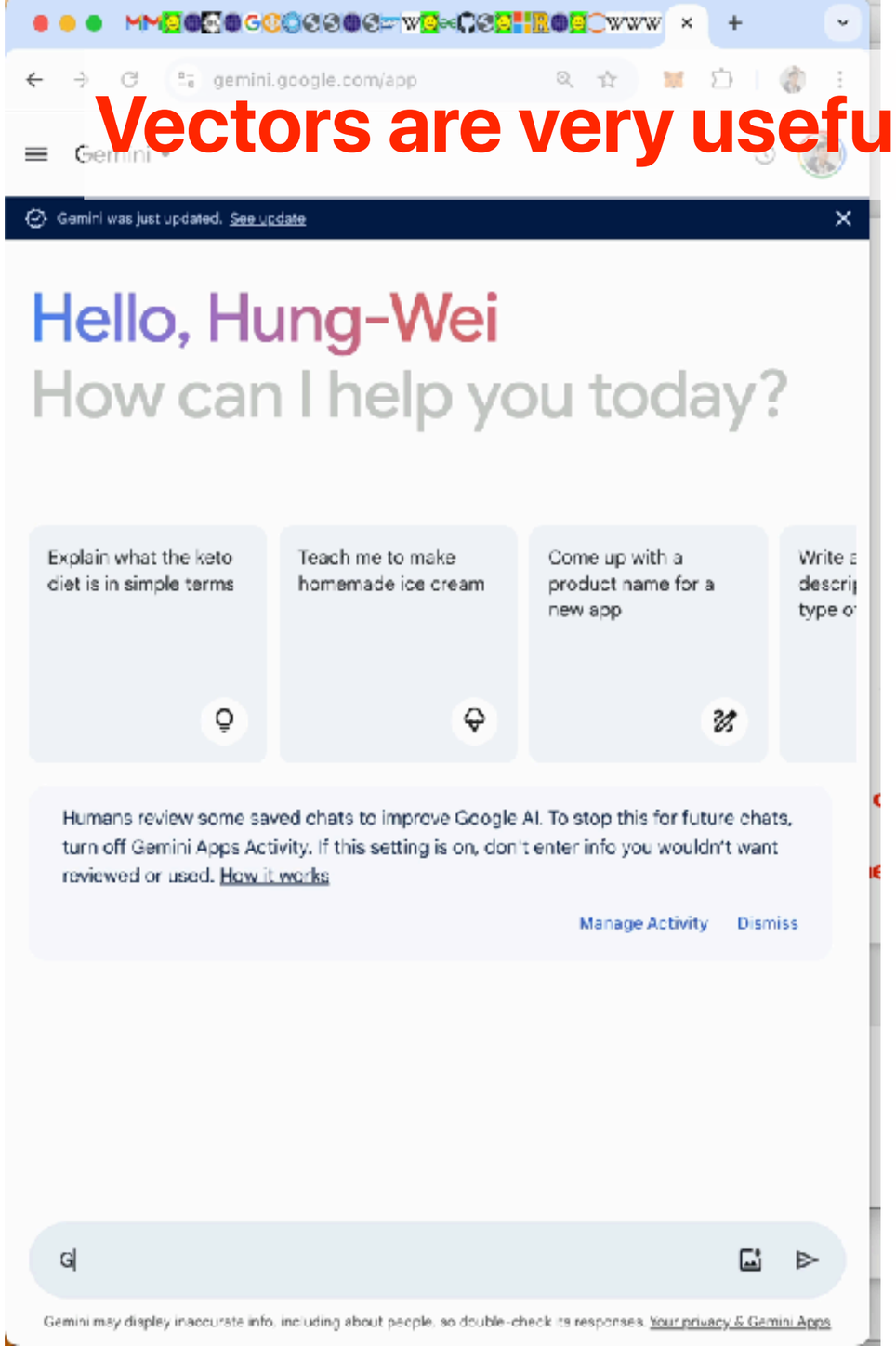
=50W!

New type of parallelism

Parallelism in modern computers

- Instruction-level parallelism — perform various, independent instructions simultaneously
  - Pipeline
  - OoO/Superscalar
- Thread-level parallelism — perform independent computation streams (composed of many instructions or SIMD instructions)
  - Multicore/SMT processors

Vectors are very useful data/mathematical representations

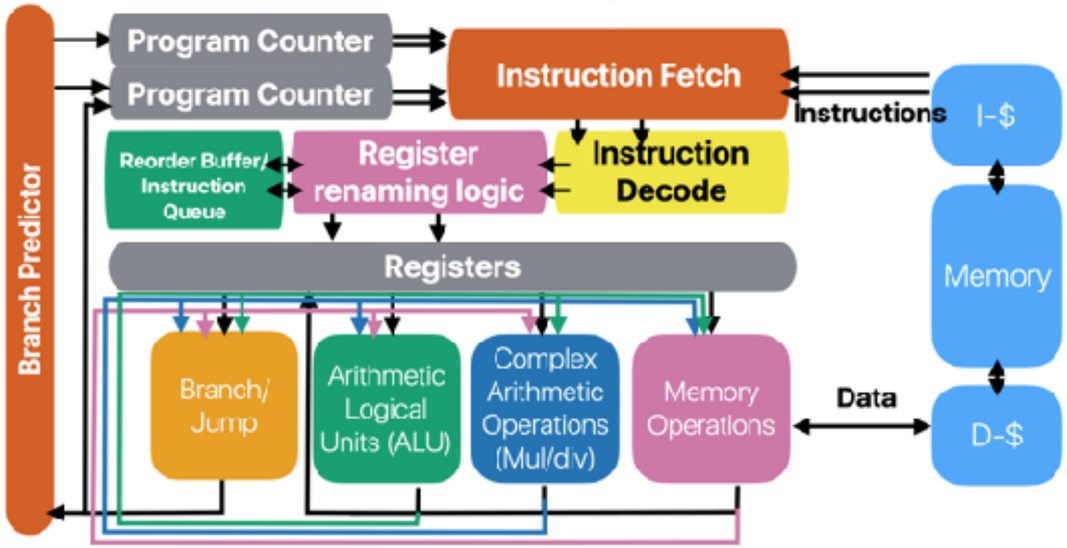


- But how do we process them using "superscalar" processors?

```
for(uint64_t i = 0; i < m; i++) {
    result = 0;
    for(uint64_t j = 0; j < n; j++) {
        result += matrix[i][j]*vector[j];
    }
     output[i] = result;
}
```

How well does vector processing map to super "scalar" processors

```
for(uint64_t i = 0; i < m; i++) {
    result = 0;
    for(uint64_t j = 0; j < n; j++) {
        result += matrix[i][j]*vector[j];
    }
    output[i] = result;
}
```



Assume we have a 5-issue INT, 3-load, 4-store pipeline like Alder Lake

Performance is very limited by both the memory bandwidth and available functional units

```
load vector[0]      load matrix[0][1]          load vector[3]          load matrix[0][4]
load matrix[0][0]   load vector[2]             load matrix[0][3]       load vector[5]
load vector[1]      load matrix[0][2]          load vector[4]          load matrix[0][5]

                    mul matrix[0][0], vector[0] mul matrix[0][1], vector[1] mul matrix[0][3], vector[3]
                                                mul matrix[0][2], vector[2]
```

Characteristics of vector processing

- Their operations are uniform across all pairs of elements

— Can we have just one instruction to control all pair-wise operations?

- There are very limited vector operations with mathematical meanings

— Can we simplify the processing elements design and make space for more PEs?

- They have very good spatial locality

— Can we have fetch them once & put in wide registers?

```
for(uint64_t i = 0; i < m; i++) {
    result = 0;
    for(uint64_t j = 0; j < n; j++) {
          result += matrix[i][j]*vector[j];
    }
     output[i] = result;
}
```

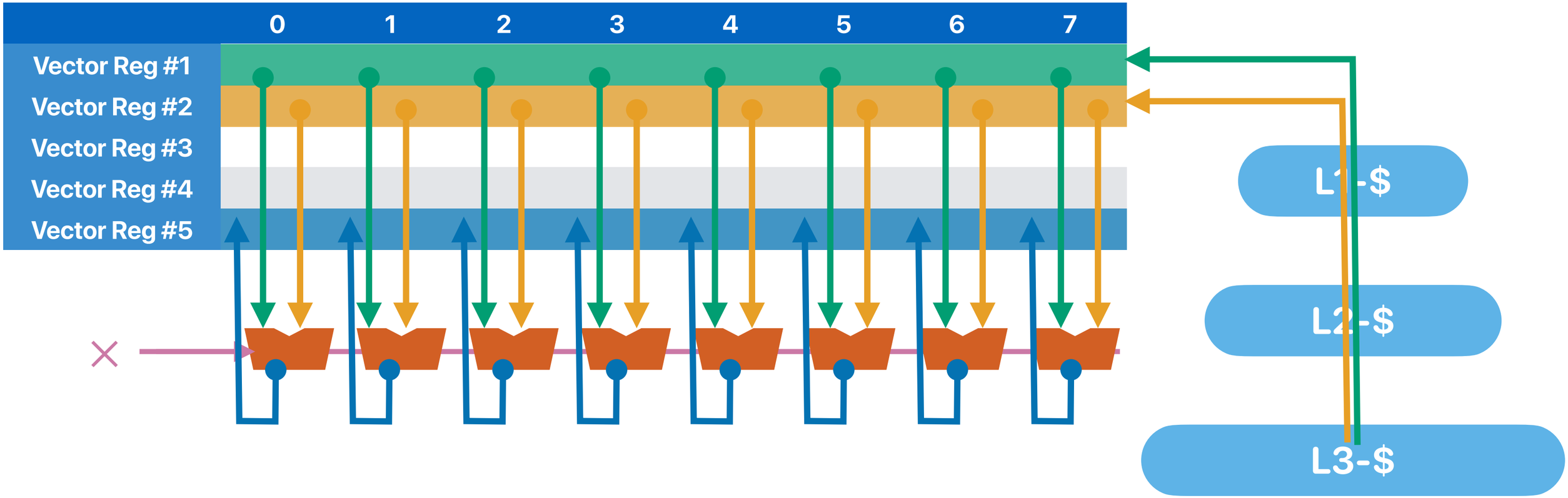
```
load vector[0]
load matrix[0][0]
load vector[1]
```

```
load matrix[0][1]
load vector[2]
load matrix[0][2]

mul matrix[0][0], vector[0]
```

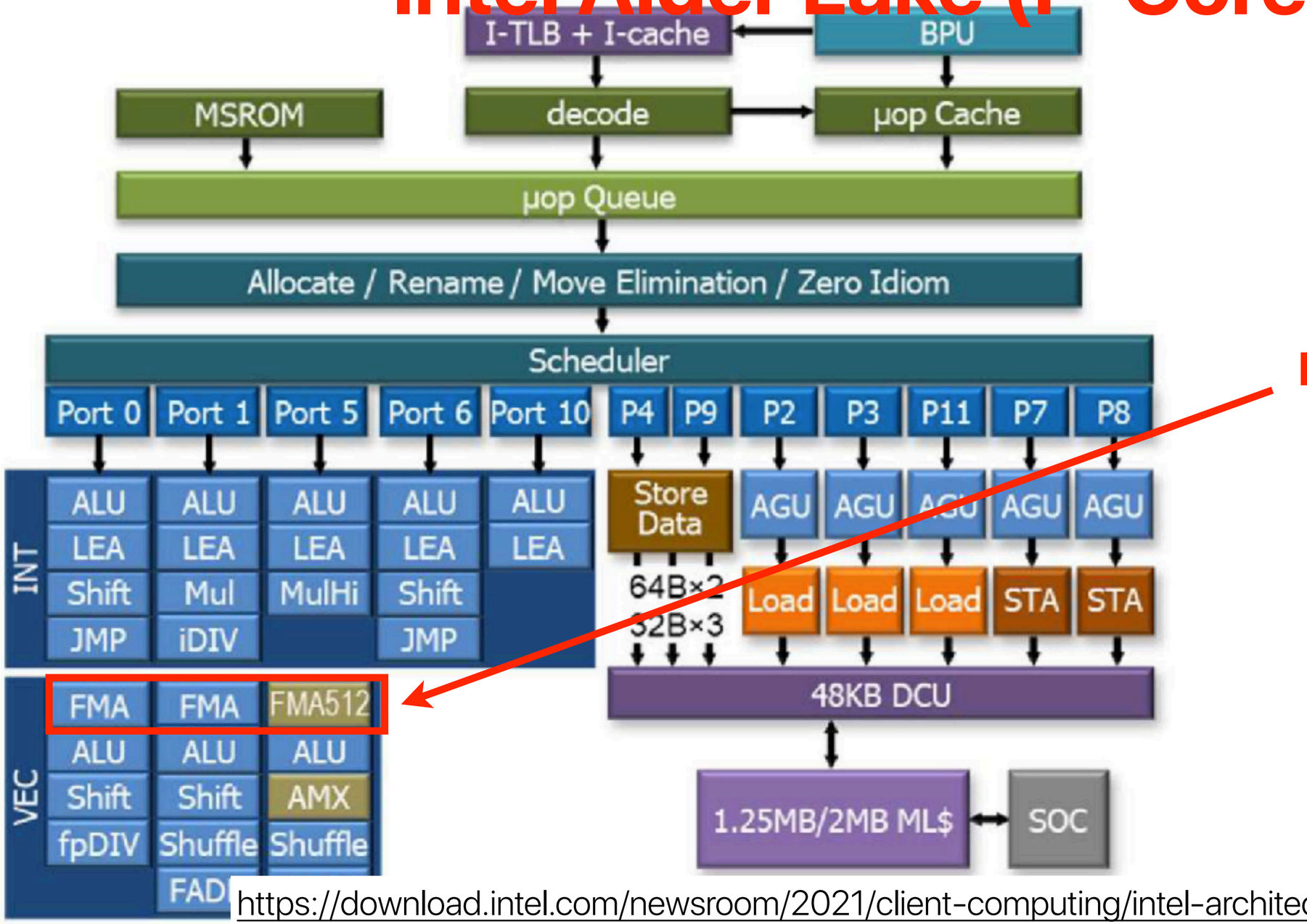
```
load vector[3]
load matrix[0][3]
load vector[4]

mul matrix[0][1], vector[1]
mul matrix[0][2], vector[2]
```

```
load matrix[0][4]
load vector[5]
load matrix[0][5]

mul matrix[0][3], vector[3]
```

Vector processing architecture



Intel Alder Lake (P-Core)



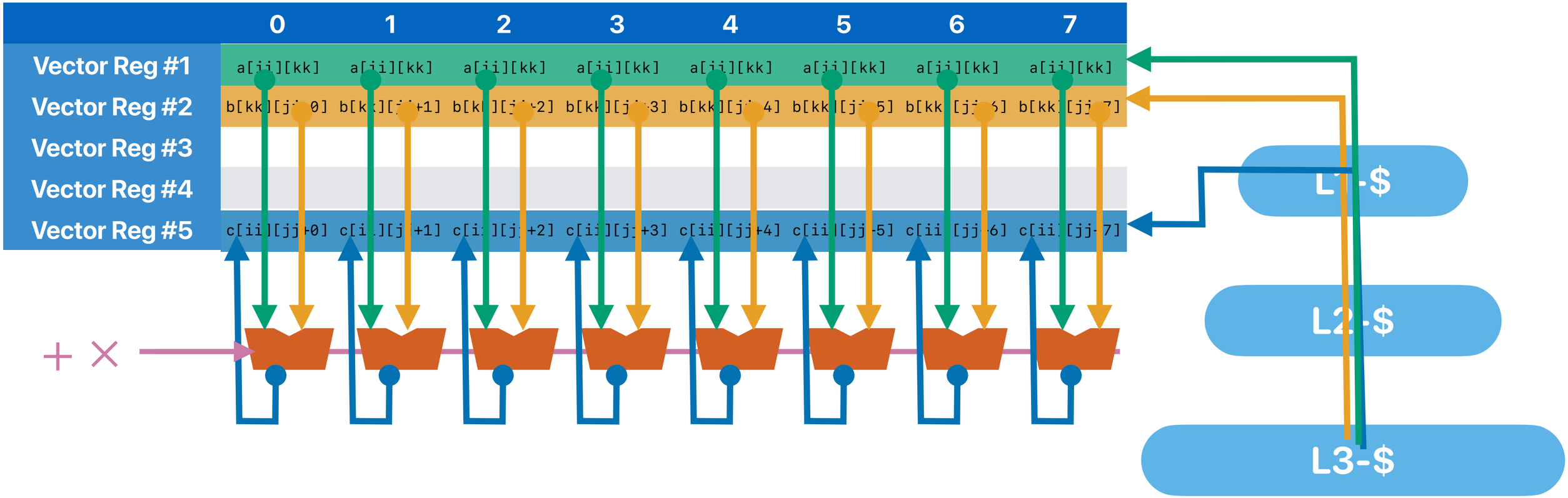
AVX Vector Processing Units

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

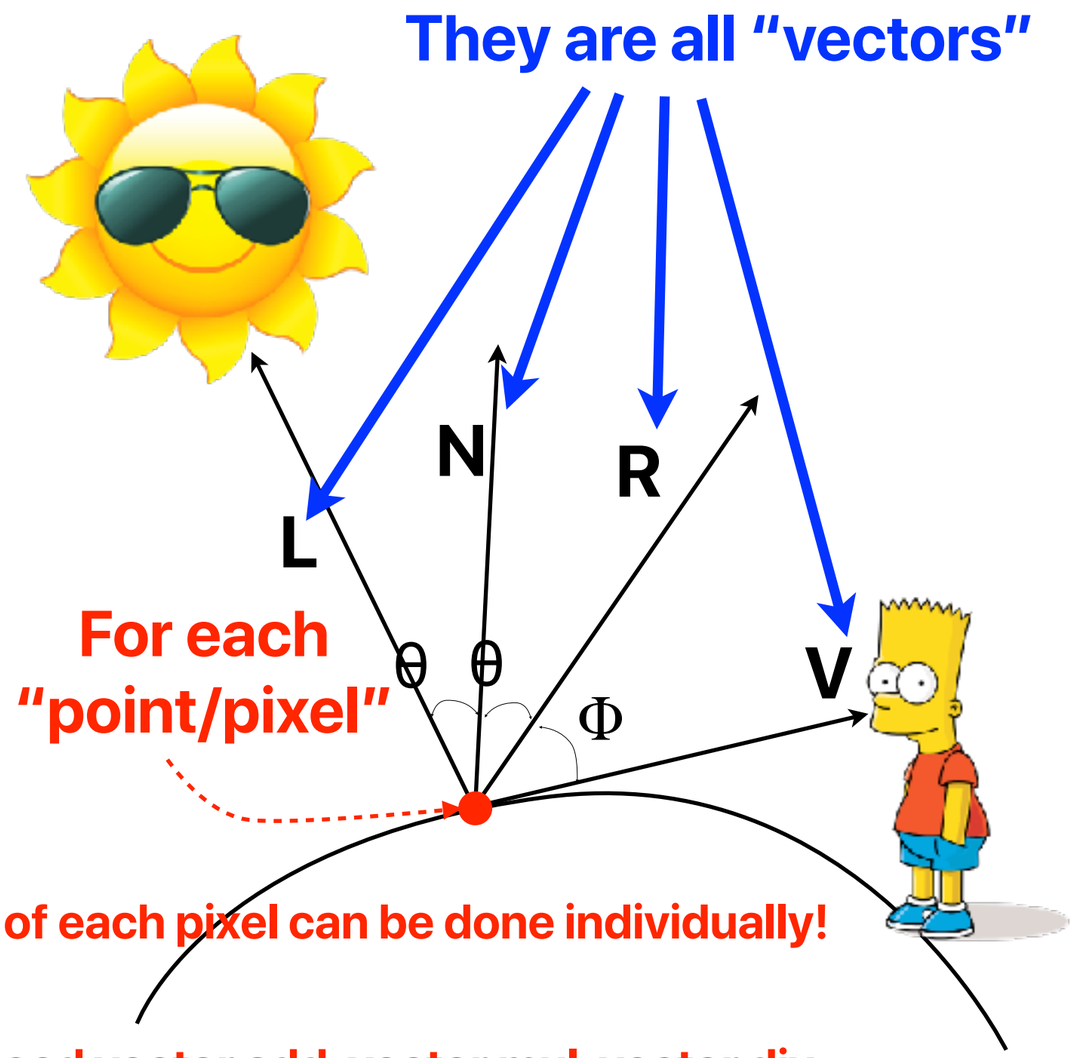
Vectorized matrix multiplications

```
#define VECTOR_WIDTH 8 // VECTOR_WIDTH is 8 as we're using AVX-512
void vector_blockmm(double **a, double **b, double **c, \
uint64_t tile_size) {
  int i,j,k, ii, jj, kk, x;
  __m512d va, vb, vc;
  for(i = 0; i < ARRAY_SIZE; i+=tile_size) {
    for(j = 0; j < ARRAY_SIZE; j+=tile_size) {
      for(k = 0; k < ARRAY_SIZE; k+=tile_size) {
        for(ii = i; ii < i+tile_size; ii++) {
          for(jj = j; jj < j+tile_size; jj+=VECTOR_WIDTH) {
             vc = _mm512_load_pd(&c[ii][jj]); // load c[ii][jj] to c[ii][jj+7] to vc[0-7]
             for(kk = k; kk < k+tile_size; kk++) {
                va = _mm512_broadcastsd_pd(&a[ii][kk]);// load a[ii][kk] to va[0-7]
                vb = _mm512_load_pd(&b[kk][jj]);  // load b[kk][jj] to b[kk][jj+7] to vb[0-7]
                vc = _mm512_add_pd(vc,_mm512_mul_pd(va,vb));    // vc += va * vb
             }
             _mm512_store_pd(&c[ii][jj],vc);      // store vc to c[ii][jj] to c[ii][jj+4]
          }
        }
      }
    }
  }
}
```

Vector processing architecture



Basic concept of shading



$$I_{amb} = K_{amb} \cdot M_{amb}$$

$$I_{diff} = K_{diff} \cdot M_{diff} \cdot (N \cdot L)$$

$$I_{spec} = K_{spec} \cdot M_{spec} \cdot (R \cdot V)^n$$

$$I_{total} = I_{amb} + I_{diff} + I_{spec}$$

```
void main(void)
{
    // normalize vectors after interpolation
    vec3 L = normalize(o_toLight);
    vec3 V = normalize(o_toCamera);
    vec3 N = normalize(o_normal);

    // get Blinn-Phong reflectance components
    float Iamb = ambientLighting();
    float Idif = diffuseLighting(N, L);
    float Ispe = specularLighting(N, L, V);

    // diffuse color of the object from texture
    vec3 diffuseColor = texture(u_diffuseTexture, o_texcoords)

    // combination of all components and diffuse color of the
    resultingColor.xyz = diffuseColor * (Iamb + Idif + Ispe);
    resultingColor.a = 1;
}
```

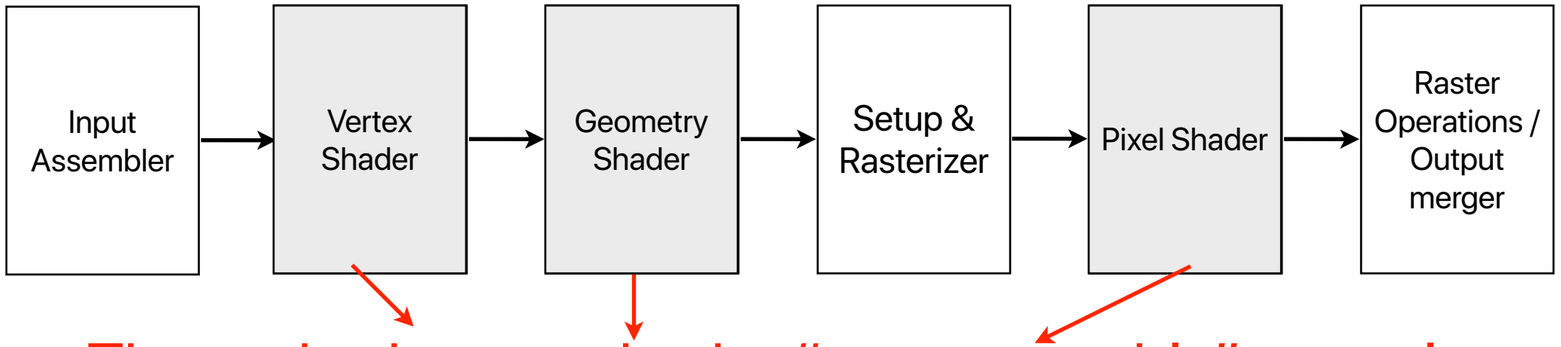
The work of each pixel can be done individually!

We only need vector add, vector mul, vector div.

GPU (Graphics Processing Unit)

- Originally for displaying images
- HD video: 1920*1080 pixels * 60 frames per second
  - Therefore, GPU is not latency-oriented by design!
  - Even for 120 frames, you still have 8ms latency to get everything done!
- Graphics processing pipeline

1 GHz can give you 8000000 cycles!!!



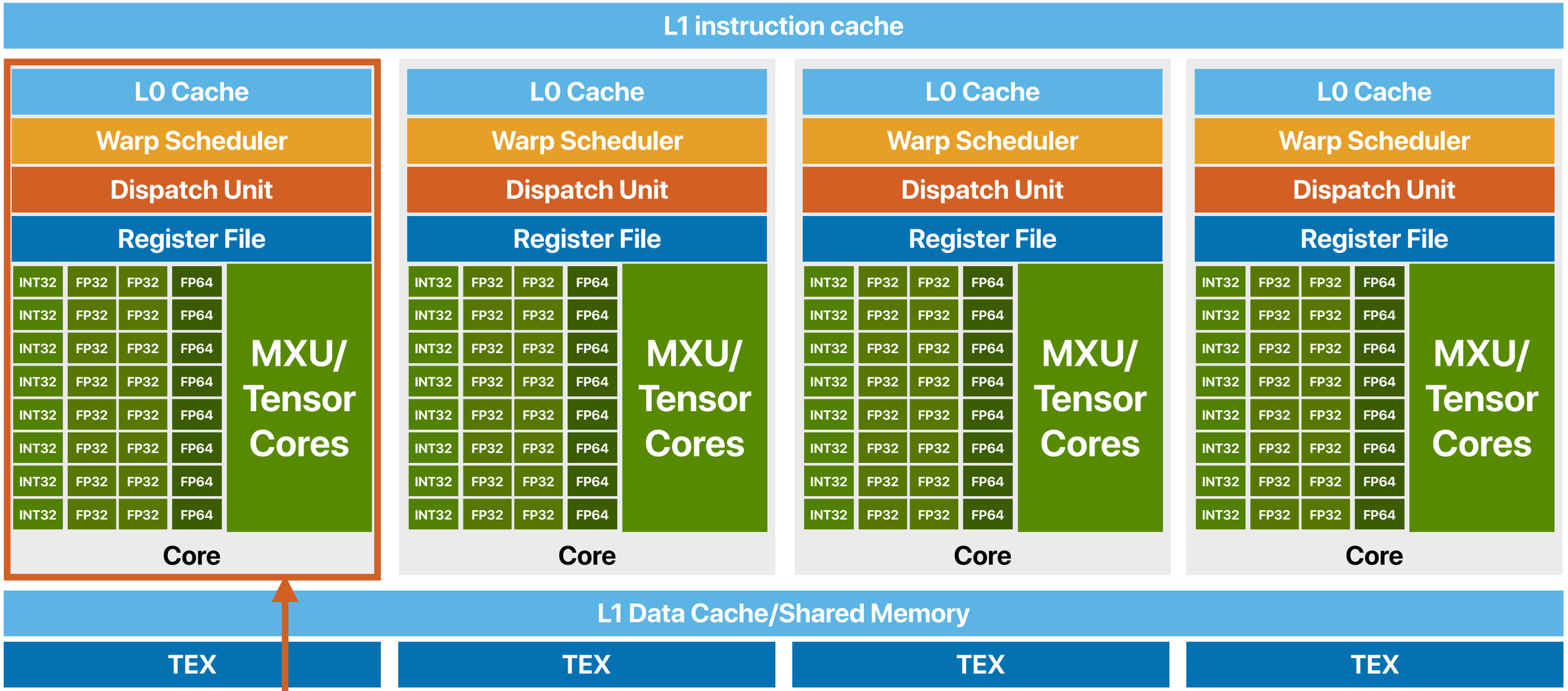
These shaders need to be "programmable" to apply different rendering effects/algorithms (Phong shading, Gouraud shading, and etc...)

What's the "appropriate" GPU architecture

- Lots of ALUs to process pixels in parallel — 2M pixels in HD resolution, very regular workloads
  - Vector processing model
- Simple operations
  - The ALUs only supports very few instructions
  - Almost no branches
- Deadline driven and throughput-oriented rather than latency oriented
  - High-bandwidth but also "higher-latency" memory
  - ALUs can be slower

GPU also follows the idea of slower, but more!

GPU Architecture



NVIDIA CUDA GPU function

```
__global__ void gpu_matrix_mult(D_TYPE *a,D_TYPE *b,
D_TYPE *c, int size)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    D_TYPE sum = 0;
    for(int i = 0; i < size; i++) {
            sum += a[row * size + i] * b[i * size + col];
    }
    c[row * size + col] = sum;
}
```

Parallelism in modern computers

- Instruction-level parallelism — perform various, independent instructions simultaneously
  - Pipeline
  - OoO/Superscalar
- Data-level parallelism — perform the same operation on multiple data elements in parallel
  - SIMD instructions
  - Compute within an SM in GPUs
- Thread-level parallelism — perform independent computation streams (composed of many instructions or SIMD instructions)
  - Multicore/SMT processors
  - Compute using multiple SMs in GPUs

GPUs still suffer from broken Dennard Scaling

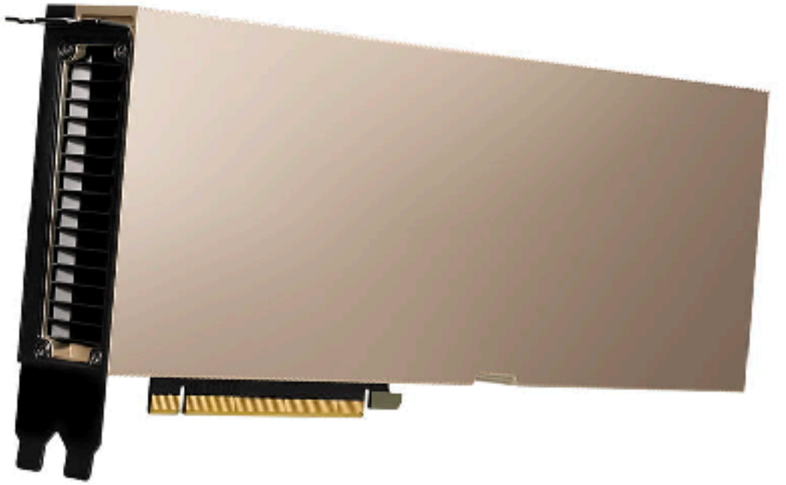
NVIDIA Accelerator Specification Comparison			
	H100	A100 (80GB)	V100
FP32 CUDA Cores	16896	6912	5120
Tensor Cores	528	432	640
GPU	GH100 (814mm2)	GA100 (826mm2)	GV100 (815mm2)
Transistor Count	80B	54.2B	21.1B
TDP	700W	400W	300W/350W
Manufacturing Process	TSMC 4N	TSMC 7N	TSMC 12nm FFN
Interface	SXM5	SXM4	SXM2/SXM3
Architecture	Hopper	Ampere	Volta

1.46x

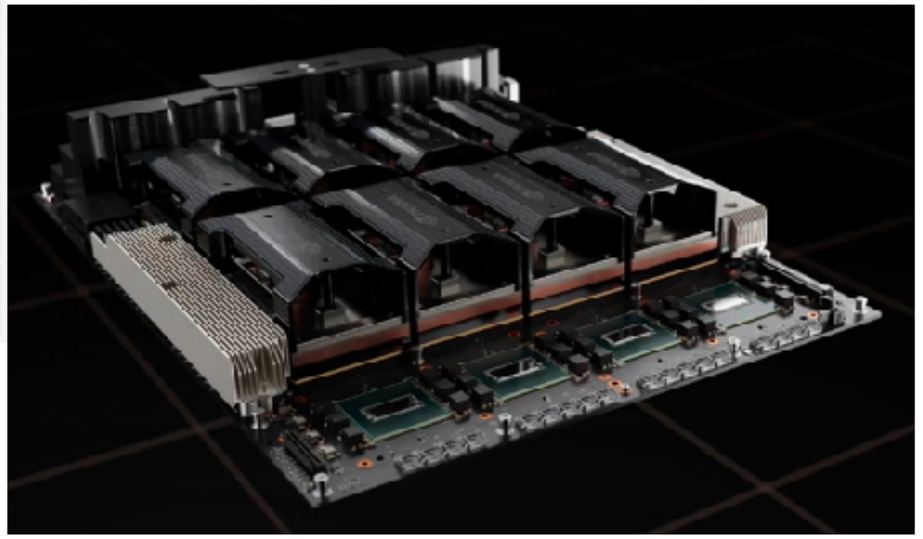
1.75x

8.75 W/ B Transistors

7.38 W/ B Transistors



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



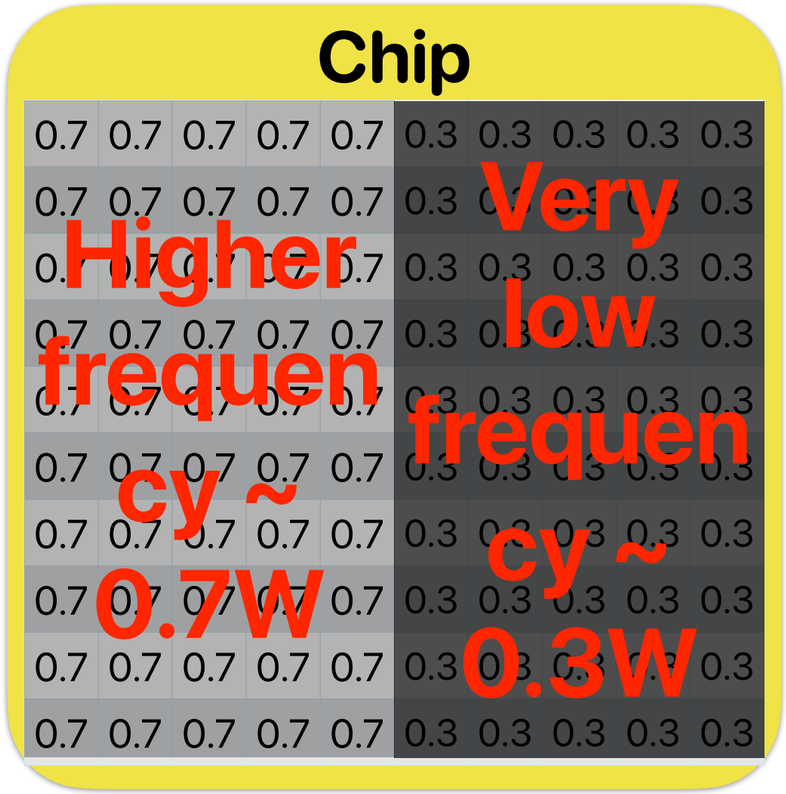
https://www.servethehome.com/wp-content/uploads/2022/03/NVIDIA-GTC-2022-H100-in-HGX-H100.jpg

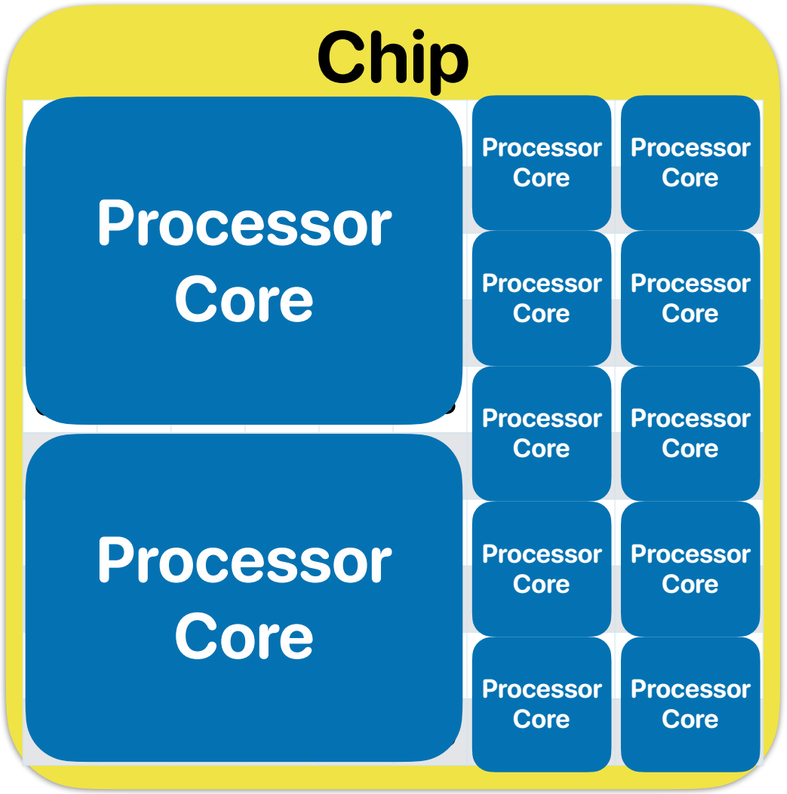
Take-aways: Challenges and SOTA solutions in the dark silicon era

- **Even if we can address all programming challenges, multi-core performance has stopped to scale due to the Dark Silicon problem**
- Aggressive dynamic frequency/voltage scaling on CMP to accommodate the demand of **latency**-sensitive, parallelism-limited applications, but the area-efficiency of the slower cores is not great
- GPUs/many-core processors improve the **throughput** per-chip through providing massive parallelism where each processing element operates at a lower speed, but not-ideal for latency-sensitive workloads

Given the same power budget, maximize the efficiency per chip

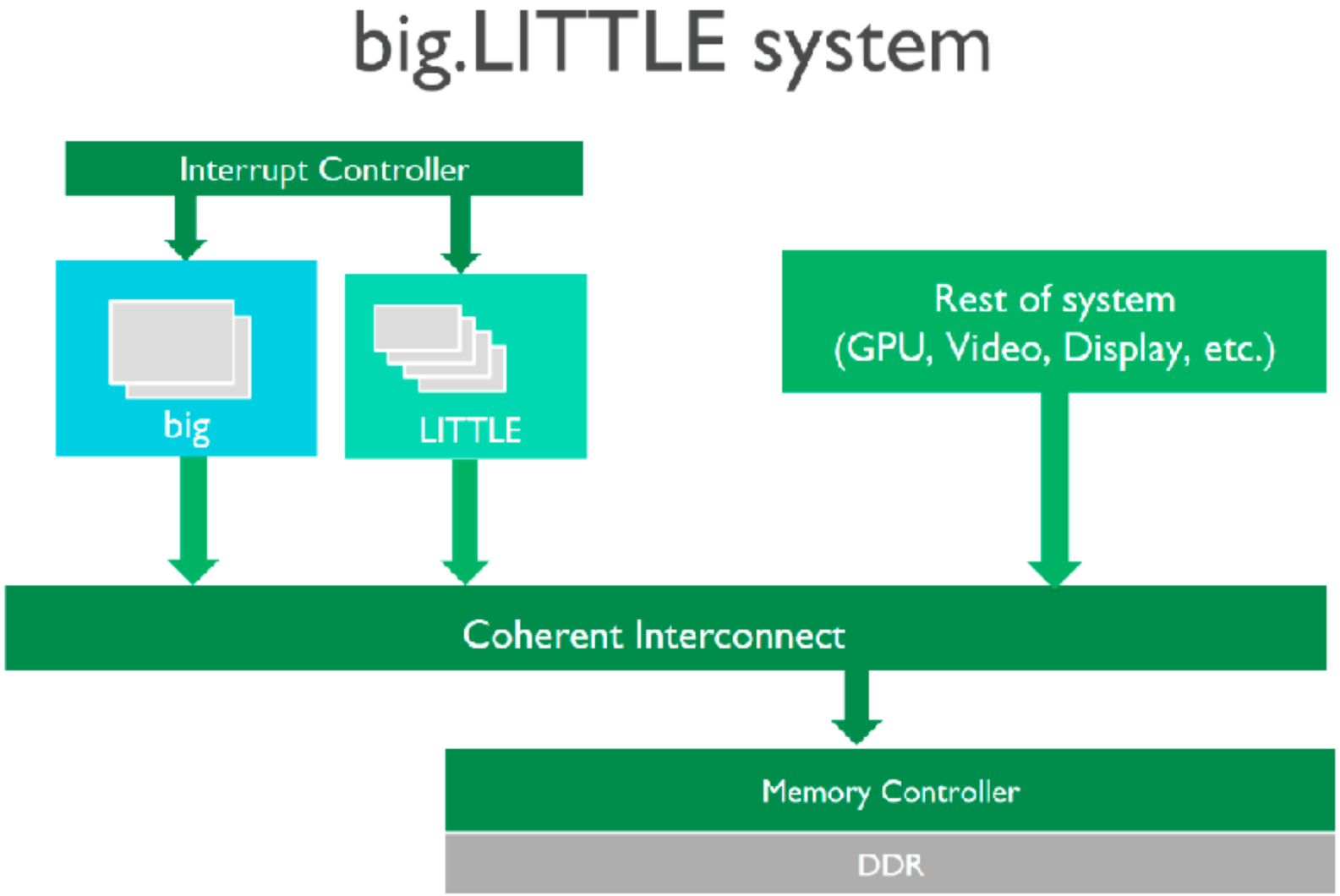
Some faster, some slower



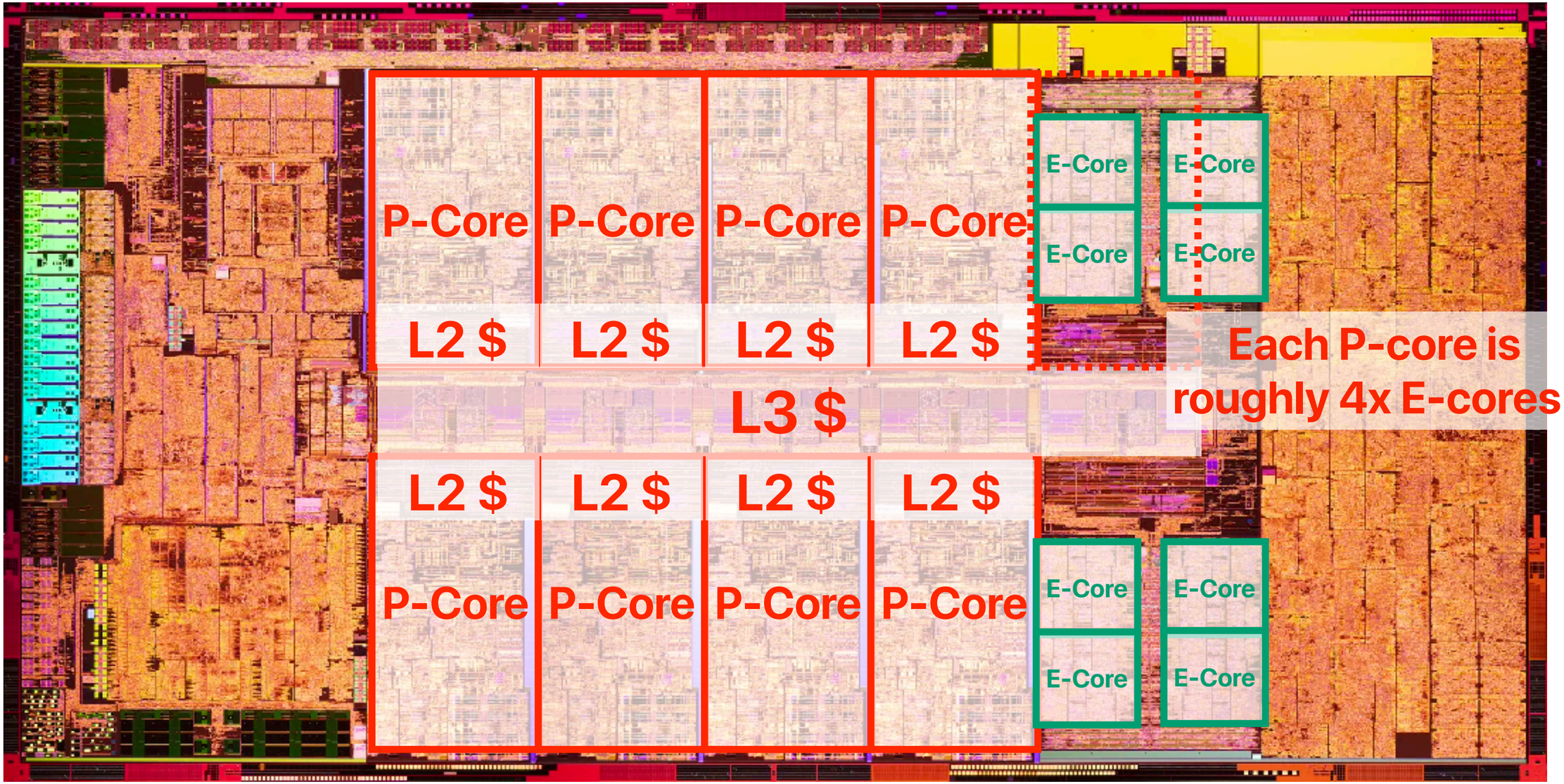


Some powerful cores for latency sensitive workloads, many small cores for highly parallelizable workloads

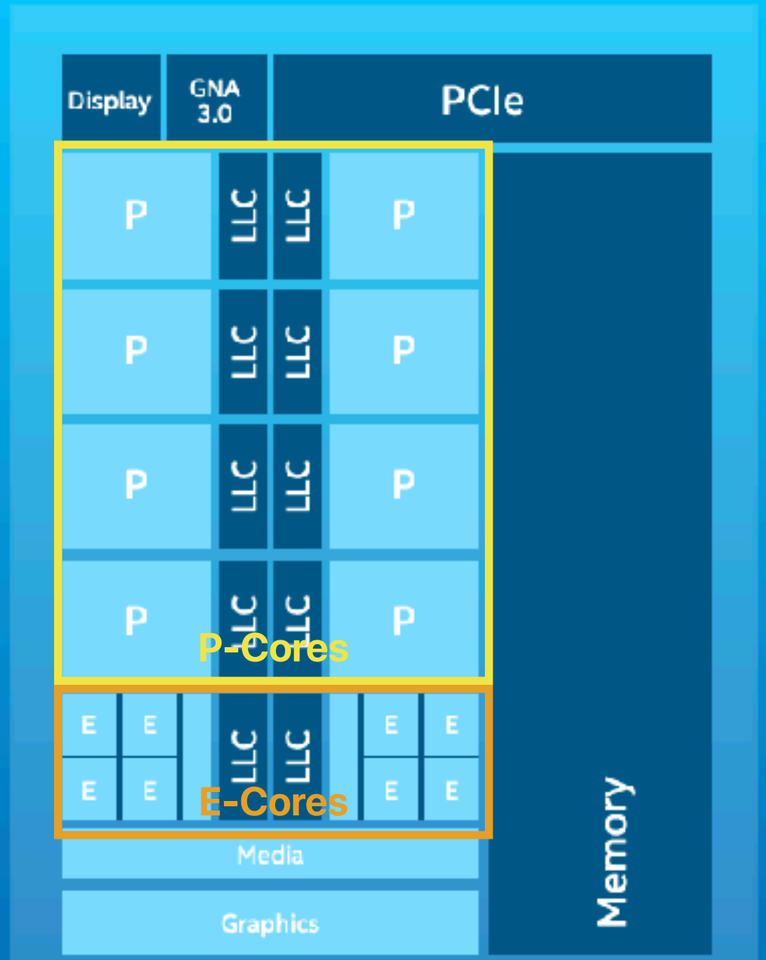
Single ISA heterogeneous CMP in ARM's big.LITTLE architecture



Intel Alder Lake



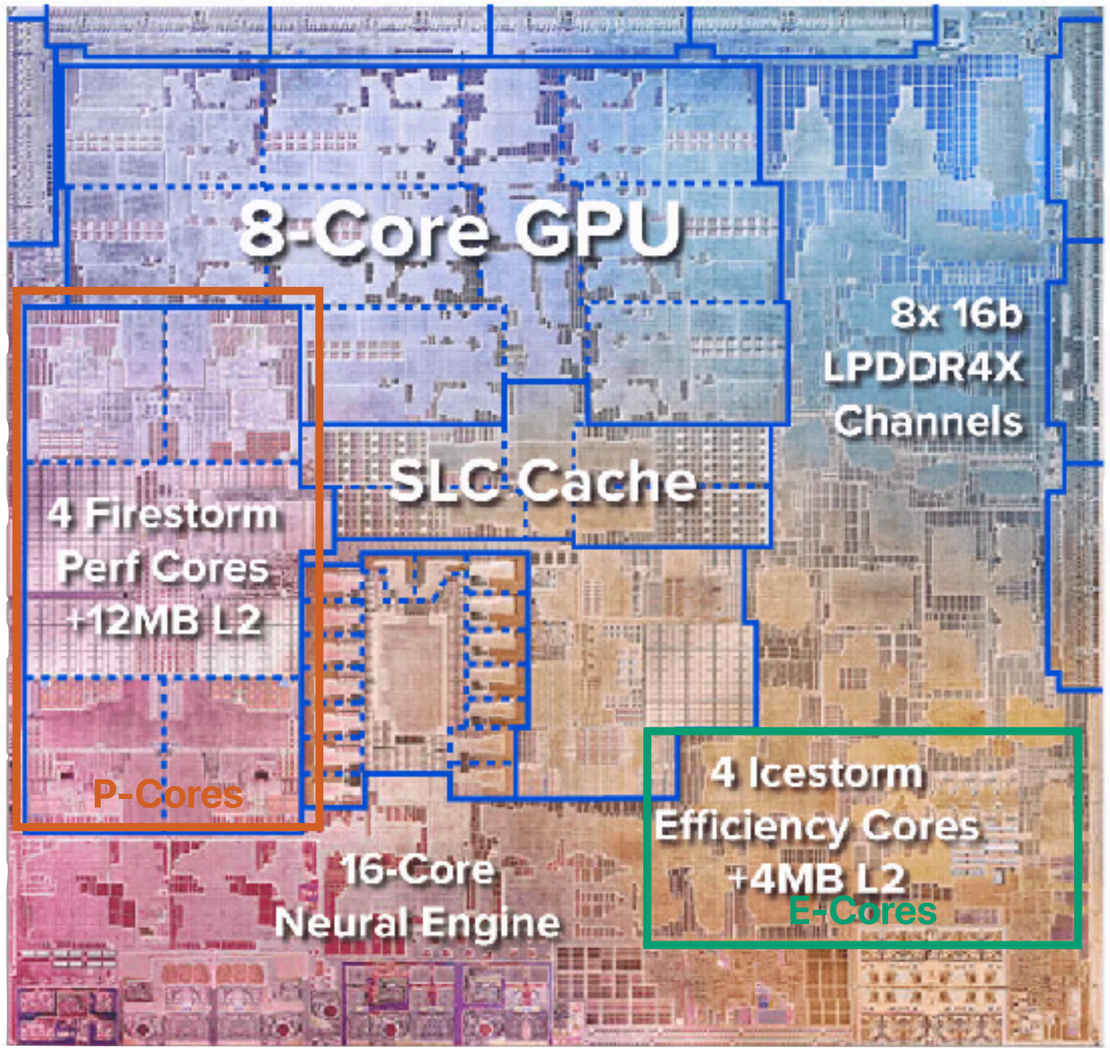
Single ISA heterogeneous CMP in Intel Processors



In a game-changing advance for core performance, 12th Gen Intel® Core™ desktop processors power a revolutionary approach to the x86 architecture. Its Performance-cores—or "P-cores"—are optimized for single & lightly-threaded performance, while its Efficient-cores—or "E-cores"—are optimized for scaling highly-threaded workloads. Intel® Thread Director helps to monitor and analyze performance data in real-time to seamlessly place the right application thread on the right core and optimize performance per watt.[1] That means gamers, creators, and professionals can harness both intelligence and power to enhance the experiences that matter most.

https://download.intel.com/newsroom/2021/client-computing/12th-gen-processor-product-brief.pdf

Single ISA heterogeneous CMP in Apple's M1



https://www.toptal.com/apple/apple-m1-processor-compatibility-overview

Single ISA heterogeneous CMP (big.Little)

- Assume we have two processor core architectures:
  (1) P-core: higher performance, but also higher power consumption and larger in size and
  (2) E-core: lower performance, but also lower power consumption and smaller in size.
  If we can build three types of multi-processors, all has the same area/size and power budget:
  (**big Only**) 4x P-cores only.
  (**Little Only**) 24x E-cores only and
  (**big.Little**) 2x P-cores and 12 E-cores. Please identify the correct expectations of their performance
  - ① For a program with limited thread-level parallelism, **big.Little** would deliver better than or at least the same level of performance as **Little Only**
  - ② For a program with limited thread-level parallelism, **big.Little** would deliver better than or at least the same level of performance as **Big Only**
  - ③ For a program with rich thread-level parallelism, **big.Little** would deliver better or at least the same level of performance than **Little Only**
  - ④ For a program with rich thread-level parallelism, s **big.Little** would deliver better or at least the same level of performance than **big Only**

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

Take-aways: Challenges and SOTA solutions in the dark silicon era

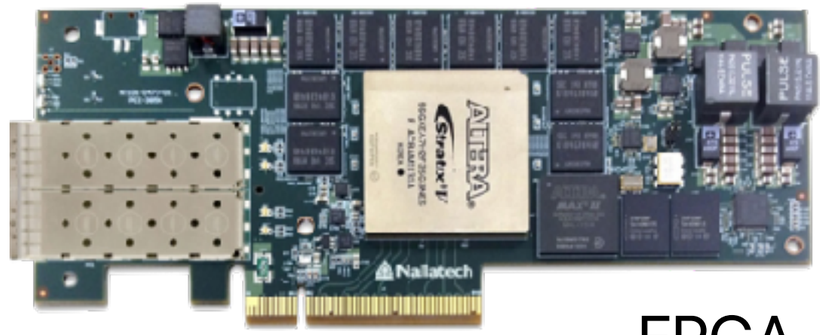
- **Even if we can address all programming challenges, multi-core performance has stopped to scale due to the Dark Silicon problem**
- Aggressive dynamic frequency/voltage scaling on CMP to accommodate the demand of **latency**-sensitive, parallelism-limited applications, but the area-efficiency of the slower cores is not great
- GPUs/many-core processors improve the **throughput** per-chip through providing massive parallelism where each processing element operates at a lower speed, but not-ideal for latency-sensitive workloads
- Single ISA, heterogeneous CMPs (e.g., big.Little cores, Intel/Apple's P-cores/E-cores) find a balance the trade-offs of general-purpose workloads, but won't be ideal if your applications go to either extreme of throughput (massive parallelism) or latency

A New “Golden” Age of Architects

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

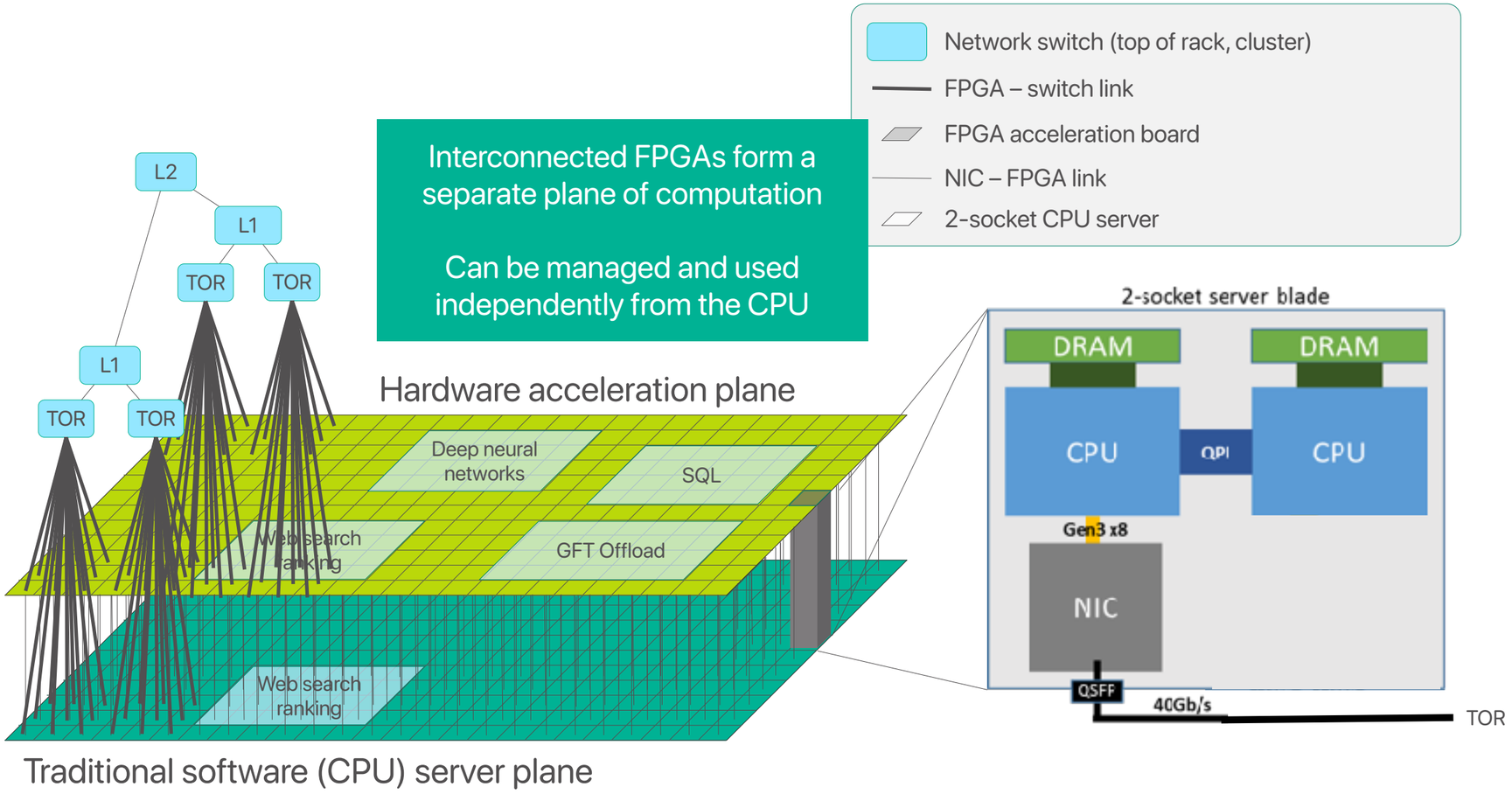
FPGA



FPGA

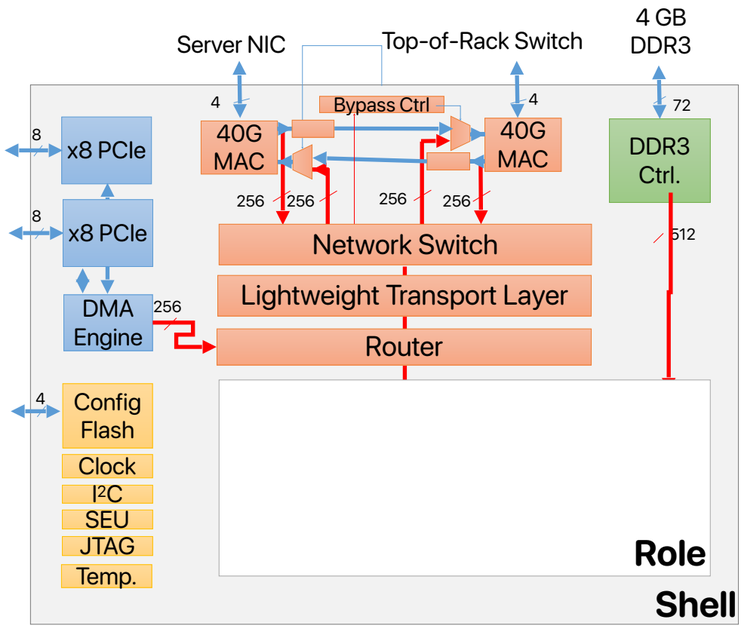
- Field Programmable Gate Array
  - An array of “Lookup tables (LUTs)”
  - Reconfigurable wires or say interconnects of LUTs
  - Registers
- An LUT
  - Accepts a few inputs
  - Has SRAM memory cells that store all possible outputs
  - Generates outputs according to the given inputs
- As a result, you 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

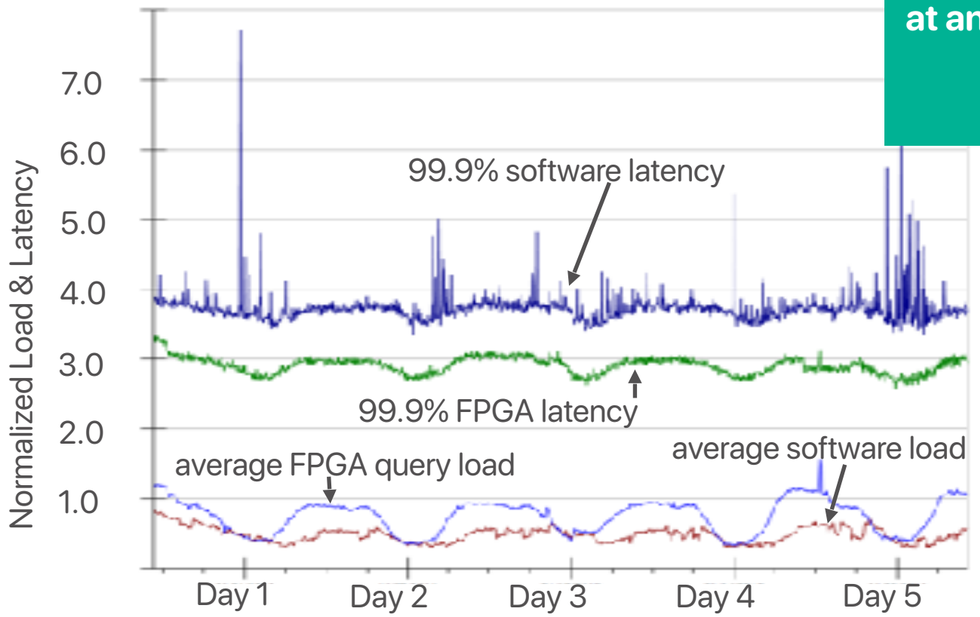


Use cases

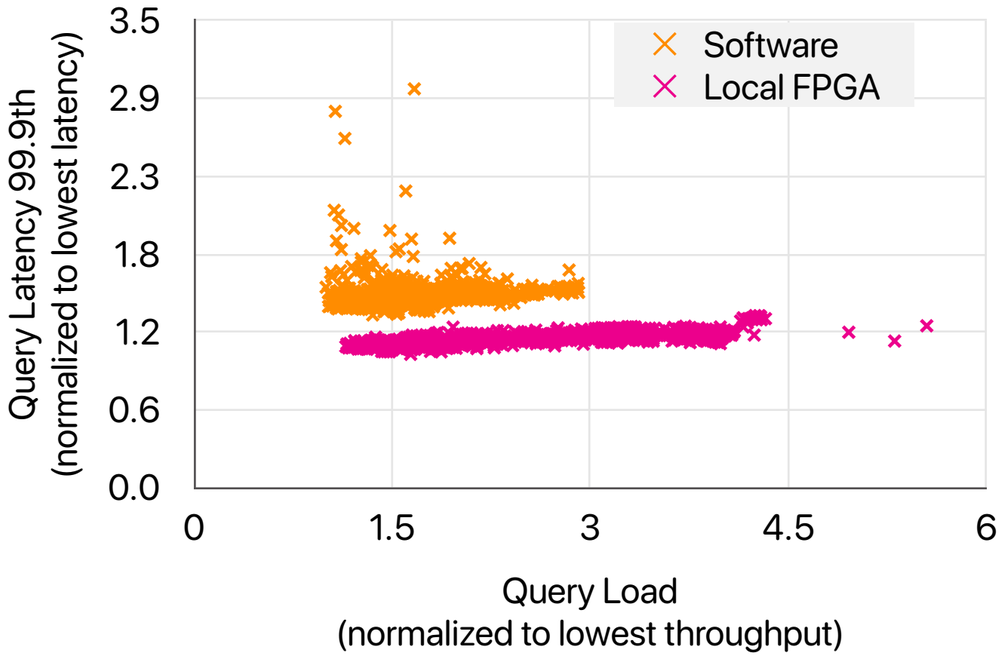
- Local: Great service acceleration
- Infrastructure: Fastest cloud network
- Remote: Reconfigurable app fabric (DNNs)

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**

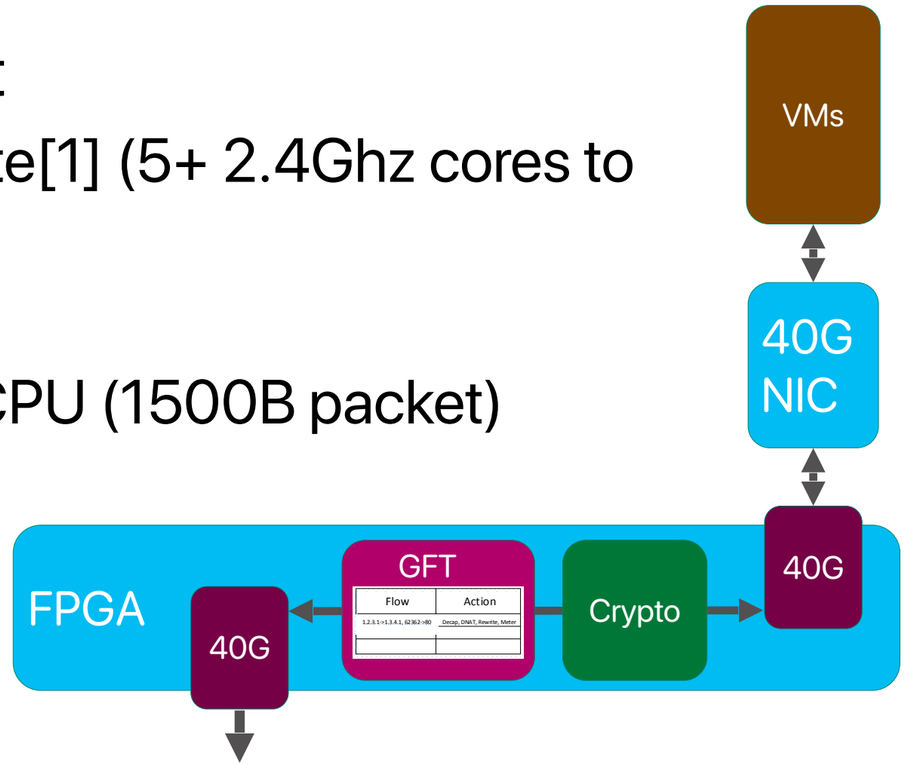


Accelerated networking

- Software defined networking
  - Generic Flow Table (GFT) rule based pack
  - 10x latency reduction vs software, CPU lo
  - 25Gb/s throughput at 25µs latency – the fastest cloud network

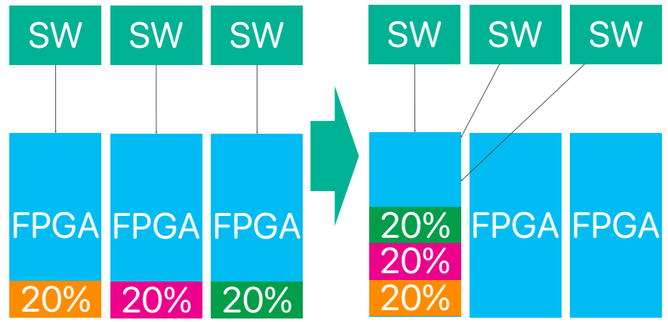
> Our FPGA implementation supports full 40 Gb/s encryption and decryption. The worst case half-duplex FPGA crypto latency for AES-CBC-128-SHA1 is 11 μs for a 1500B packet, from first flit to first flit. In software, based on the Intel numbers, it is approximately 4 μs. AES-CBC-SHA1 is, however,

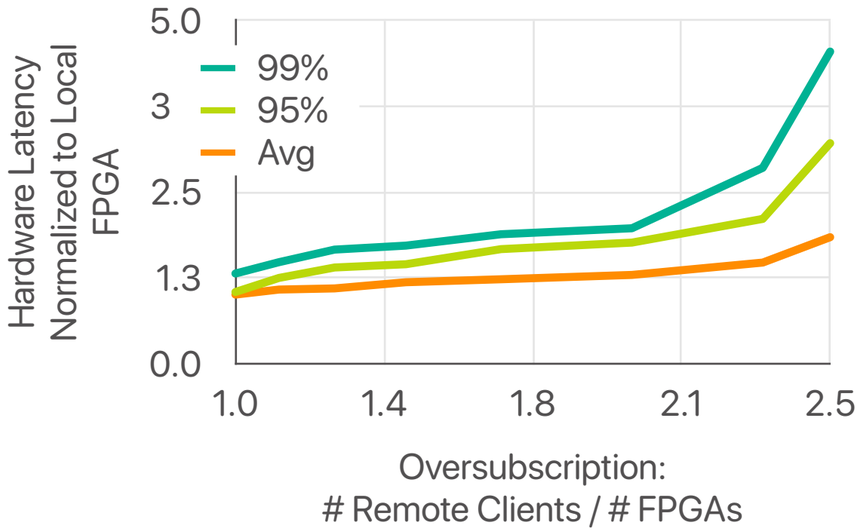
- Capable of 40 Gb line rate encrypt and decrypt
  - On Haswell, AES GCM-128 costs 1.26 cycles/byte[1] (5+ 2.4Ghz cores to sustain 40Gb/s)
  - CBC and other algorithms are more expensive
  - AES CBC-128-SHA1 is 11µs in FPGA vs 4µs on CPU (1500B packet)
  - Higher latency, but significant CPU savings



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 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

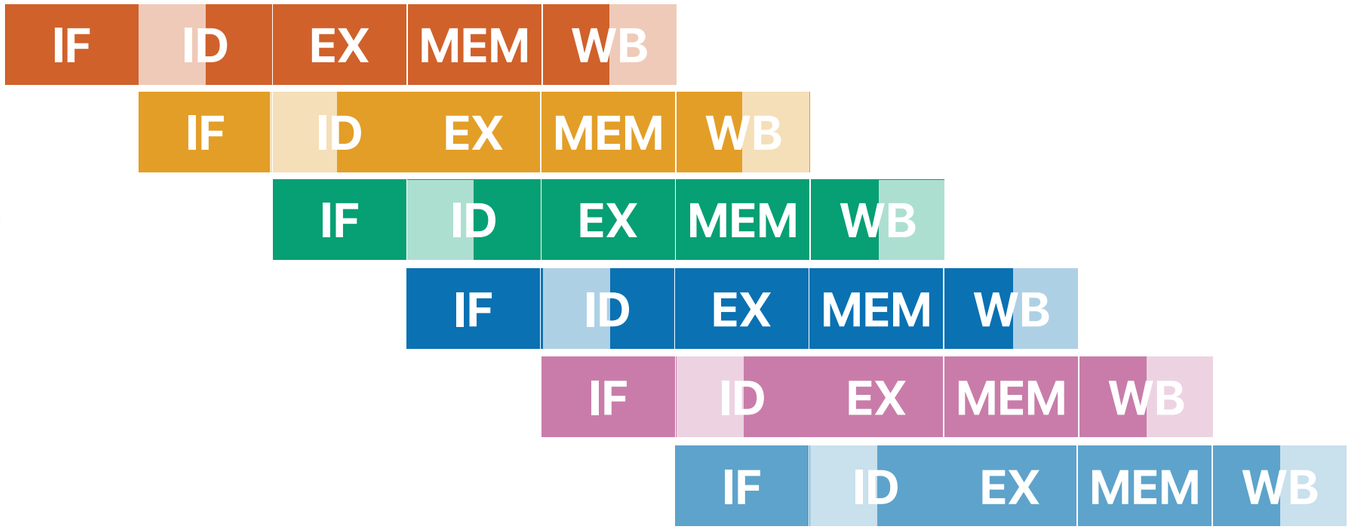
Domain Specific Hardware Accelerators

William J. Dally, Yatish Turakhia, and Song Han.
NVIDIA, UC Santa Cruz, and MIT

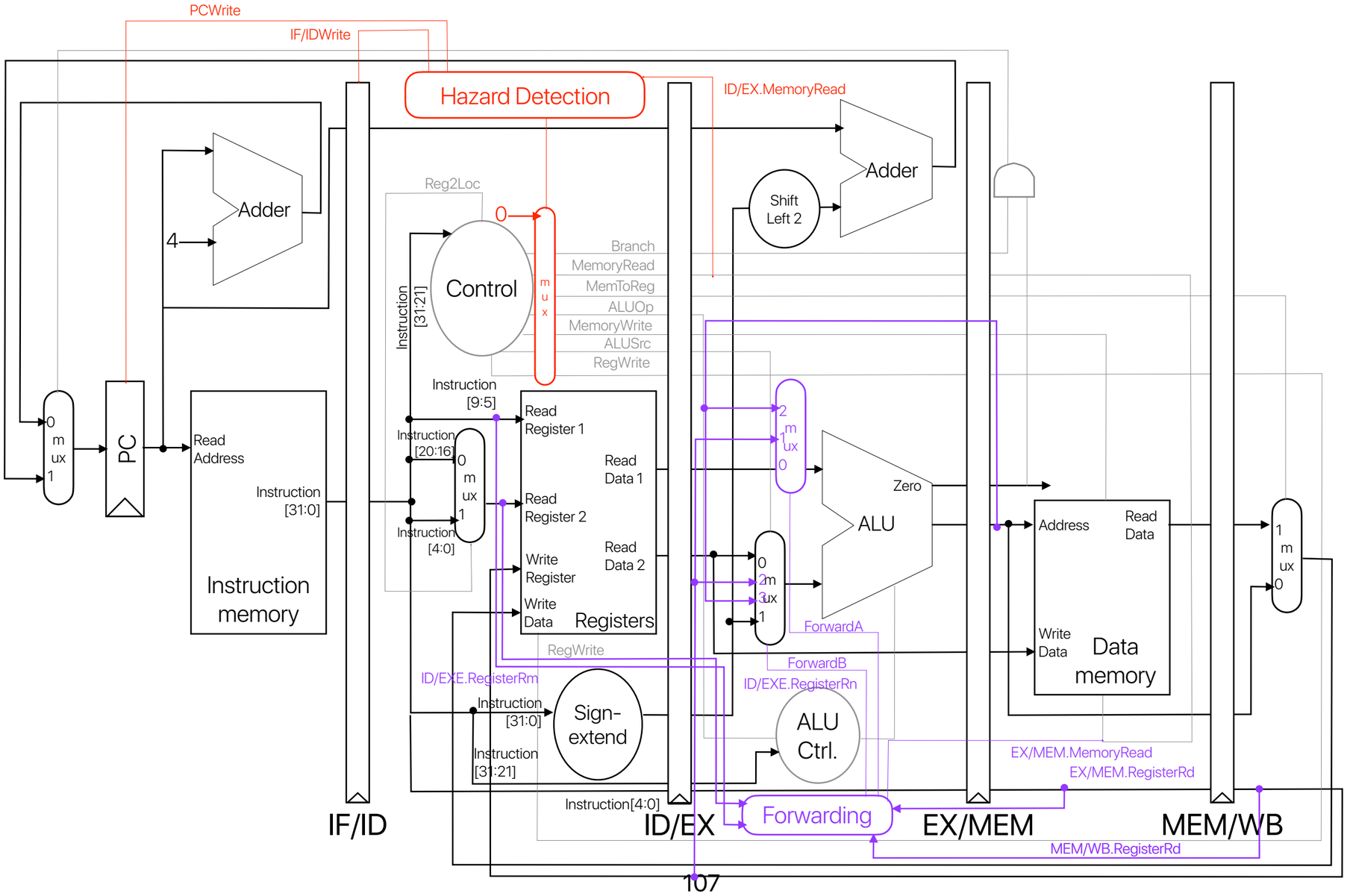
Say, we want to implement a[i] += a[i+1]*20

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

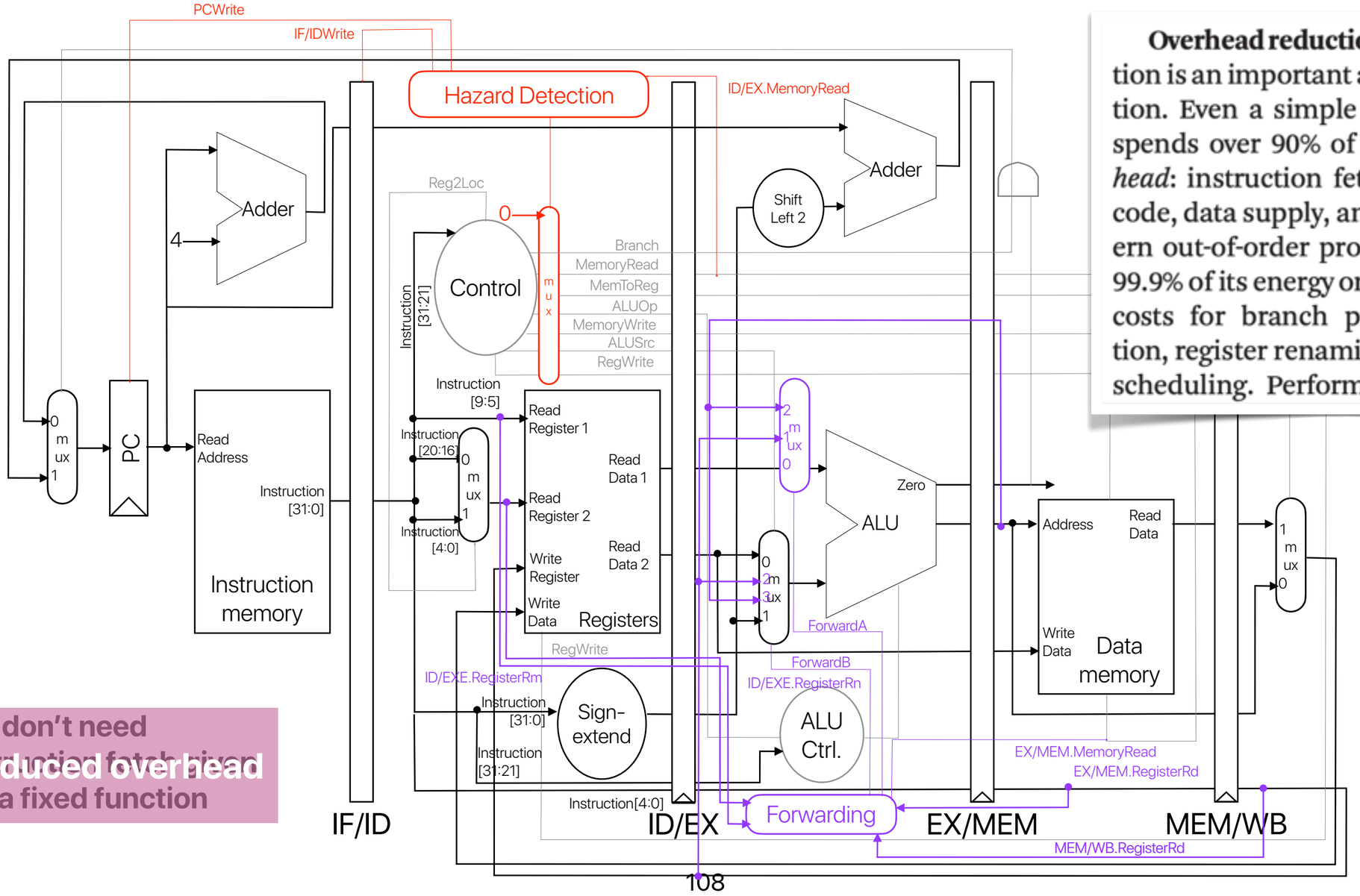
```
ld    X1, 0(X0)
ld    X2, 8(X0)
add   X3, X31, #20
mul   X2, X2, X3
add   X1, X1, X2
sd    X1, 0(X0)
```



This is what you need for these instructions



Specialize the circuit/datapath



Overhead reduction. Overhead reduction is an important aspect of specialization. Even a simple in-order processor spends over 90% of its energy on *overhead*: instruction fetch, instruction decode, data supply, and control.[10] A modern out-of-order processor spends over 99.9% of its energy on overhead[51] adding costs for branch prediction, speculation, register renaming, and instruction scheduling. Performing a 32-b integer

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

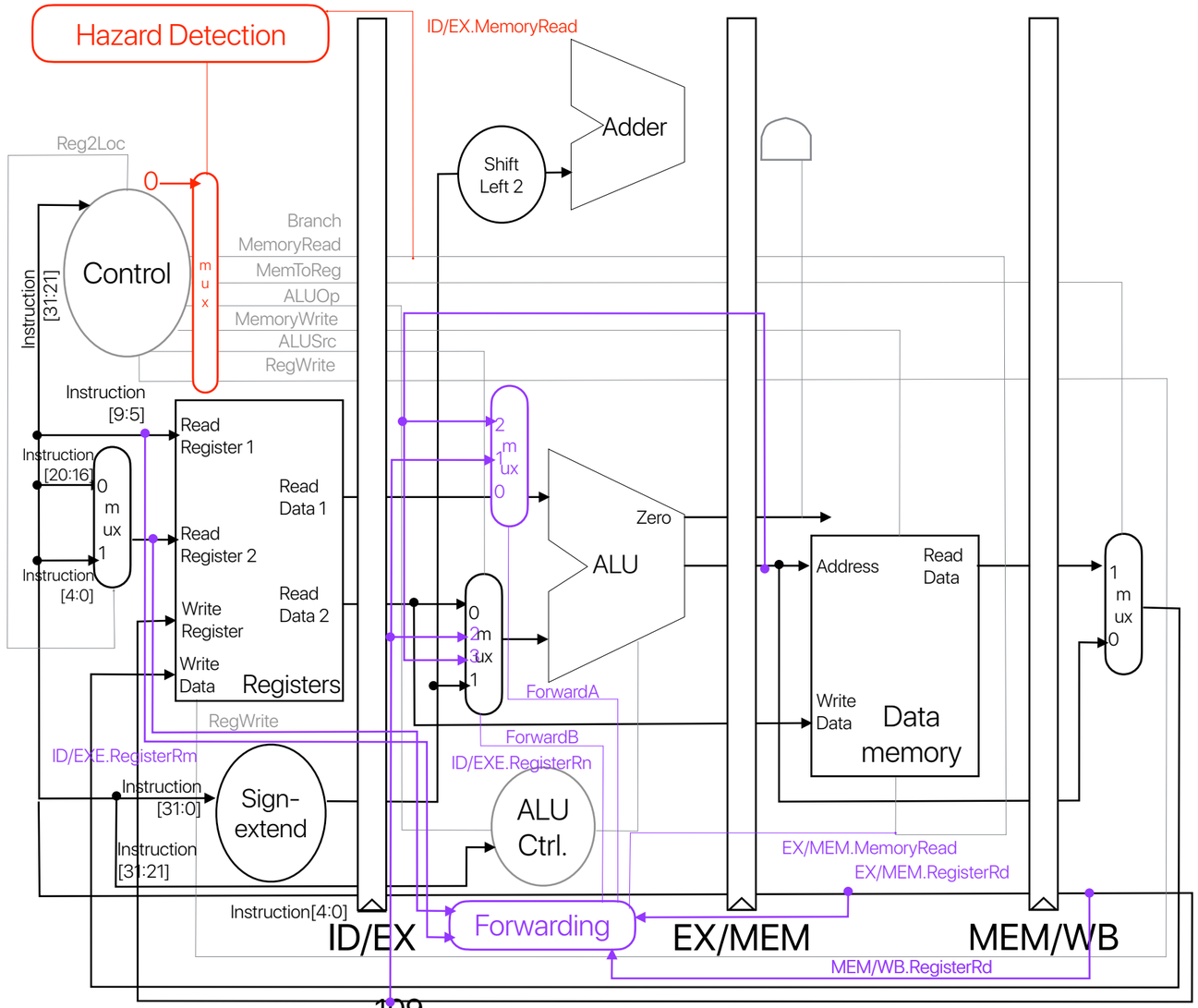
Reduced overhead

Specialize the circuit/datapath

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

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

Reduced overhead



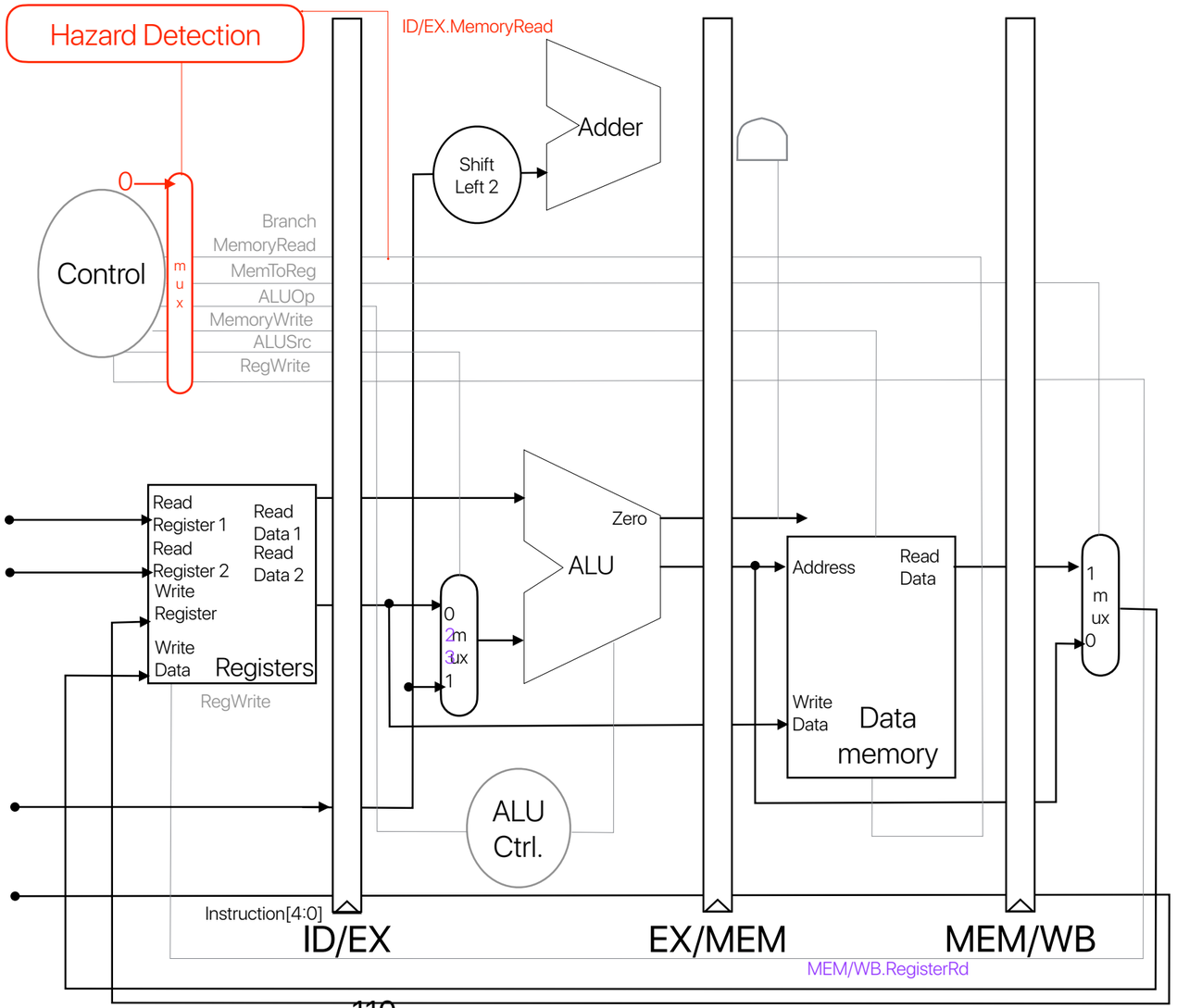
Specialize the circuit/datapath

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

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

We don’t need instruction fetch/issue, it’s a fixed function

Reduced overhead



Specialize the circuit/datapath

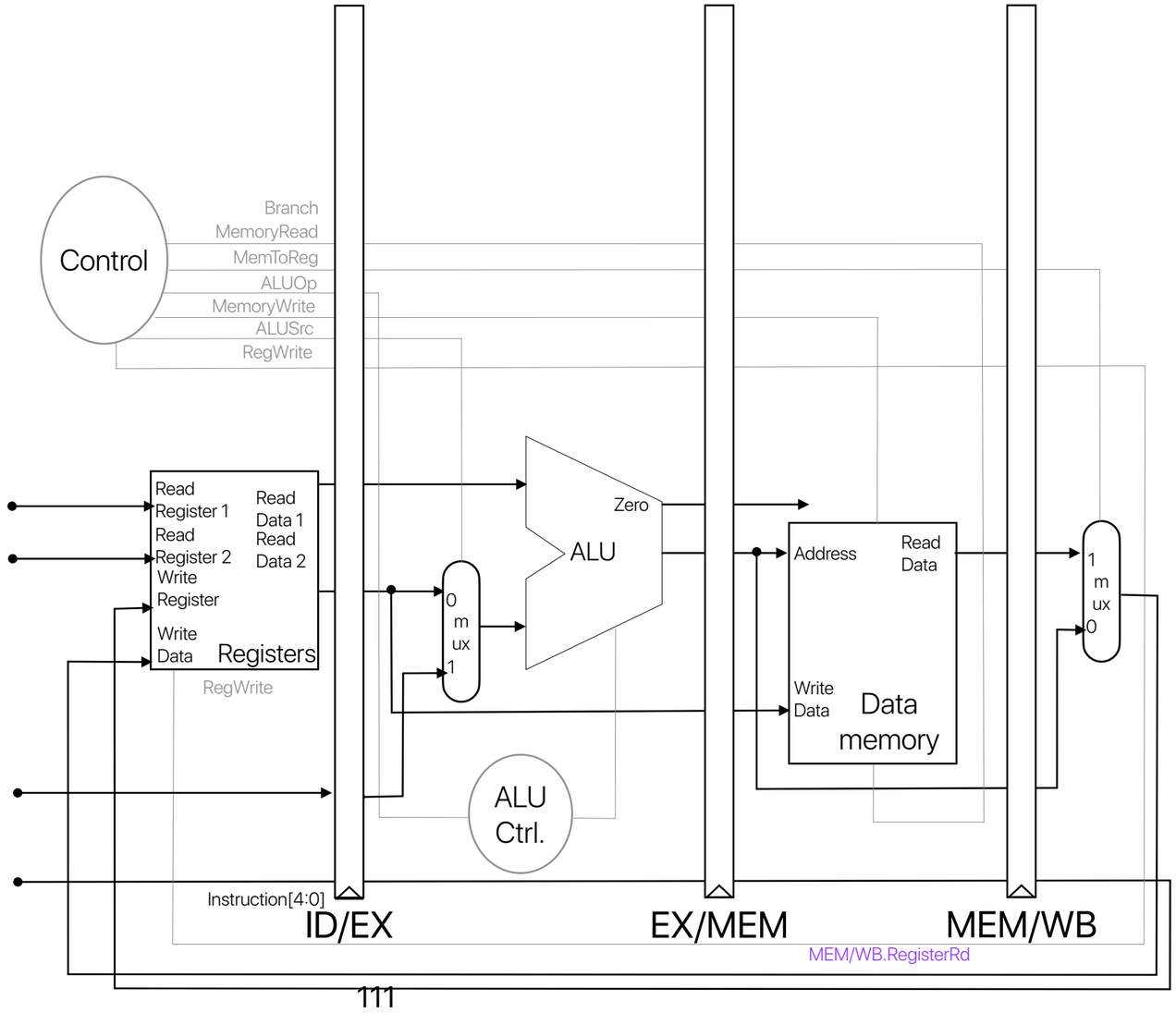
We don't need big ALUs, branches, hazard detections...

Datapath specialization

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

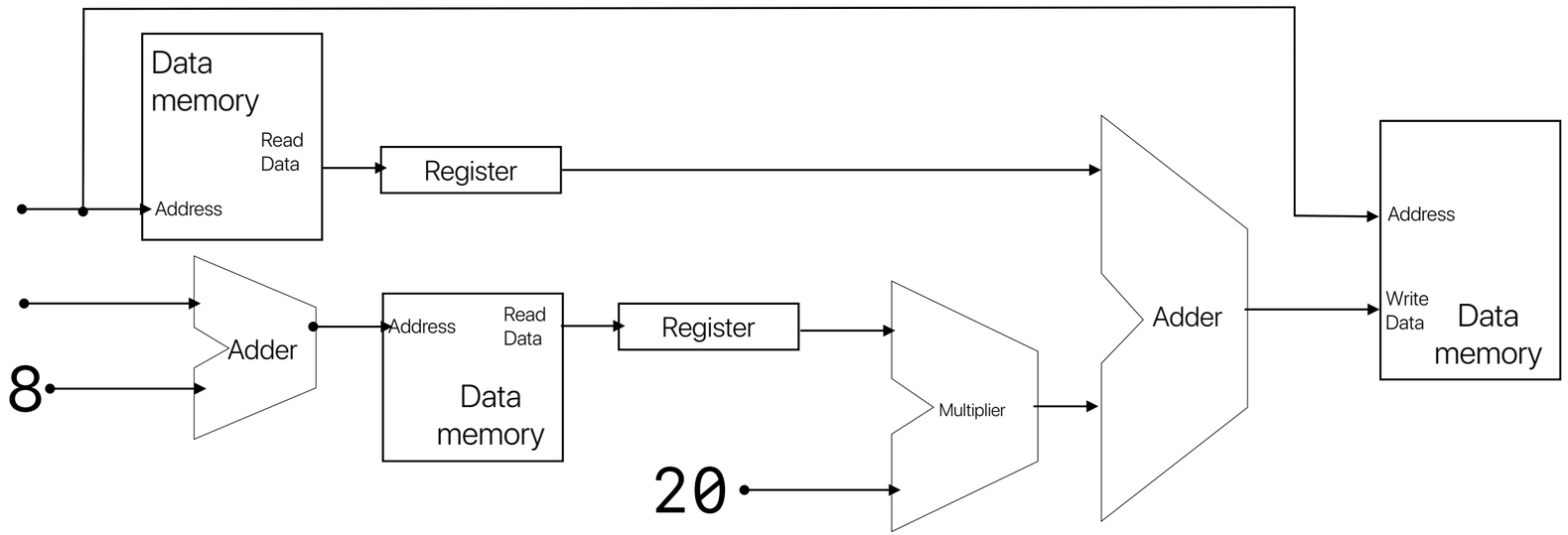
We don't need instruction fetch, it's a fixed function

Reduced overhead



Rearranging the datapath

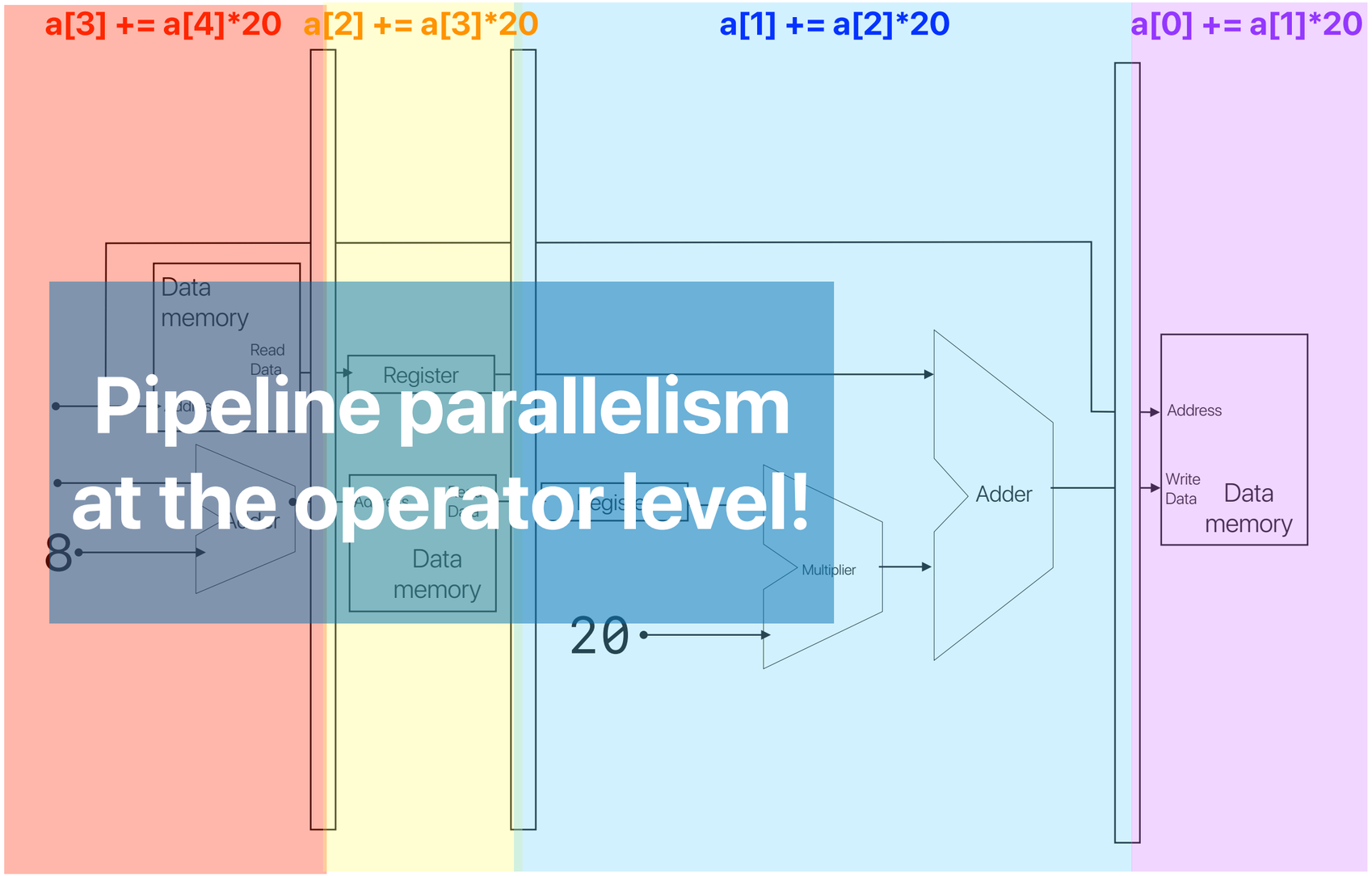
```
ld    X1, 0(X0)
ld    X2, 8(X0)
add   X3, X31, #20
mul   X2, X2, X3
add   X1, X1, X2
sd    X1, 0(X0)
```



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



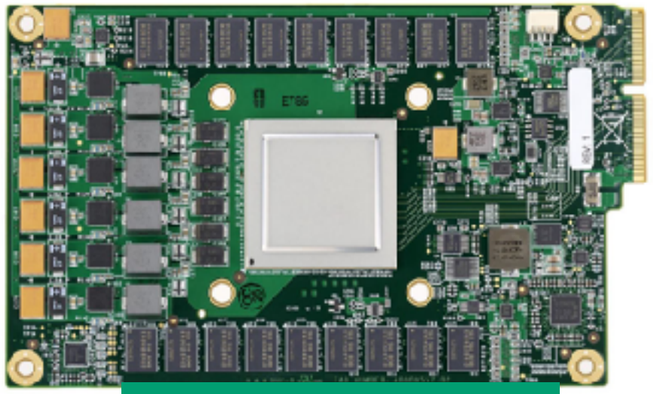
Why DSAs are more efficient?

- Datapath specialization
- Parallelism
- Local and optimized memory
- Reduced overhead

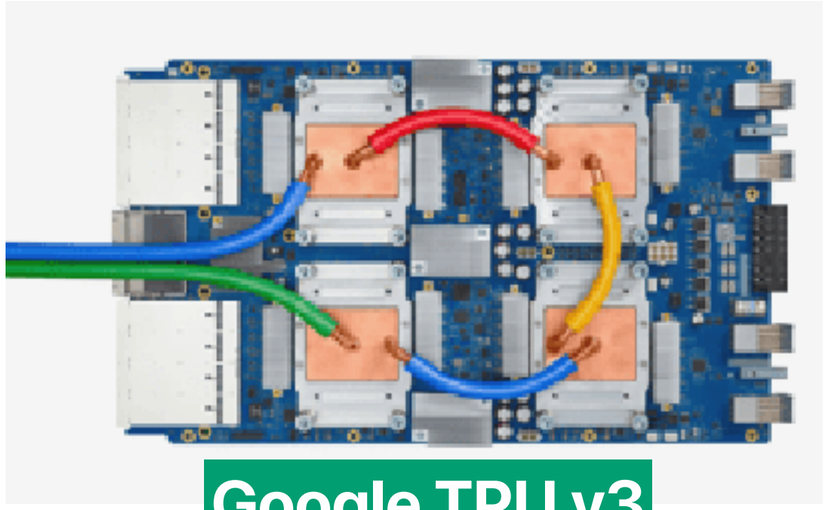
In-Datacenter Performance Analysis of a Tensor Processing Unit

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.

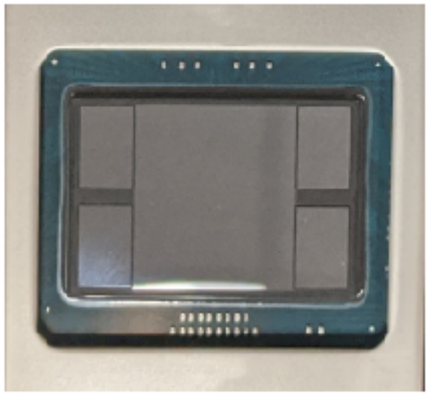
Tensor Processing Units

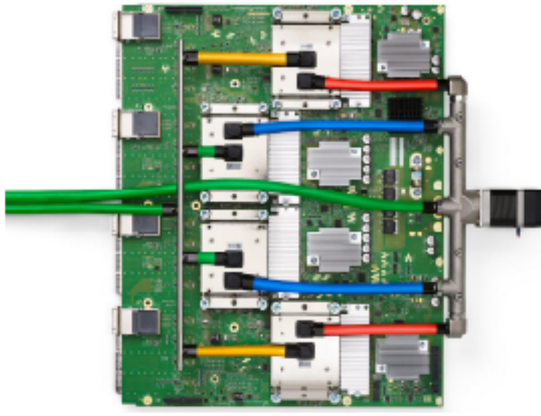


Google TPU v1

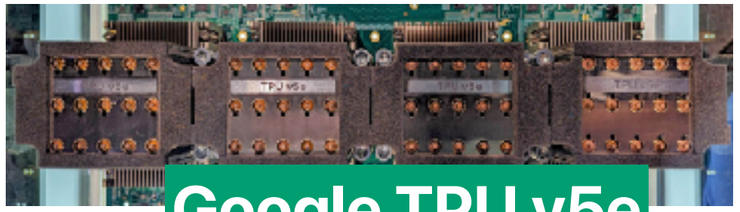


Google TPU v3





Google TPU v4



Google TPU v5e

Edge TPU

Google TPU v2

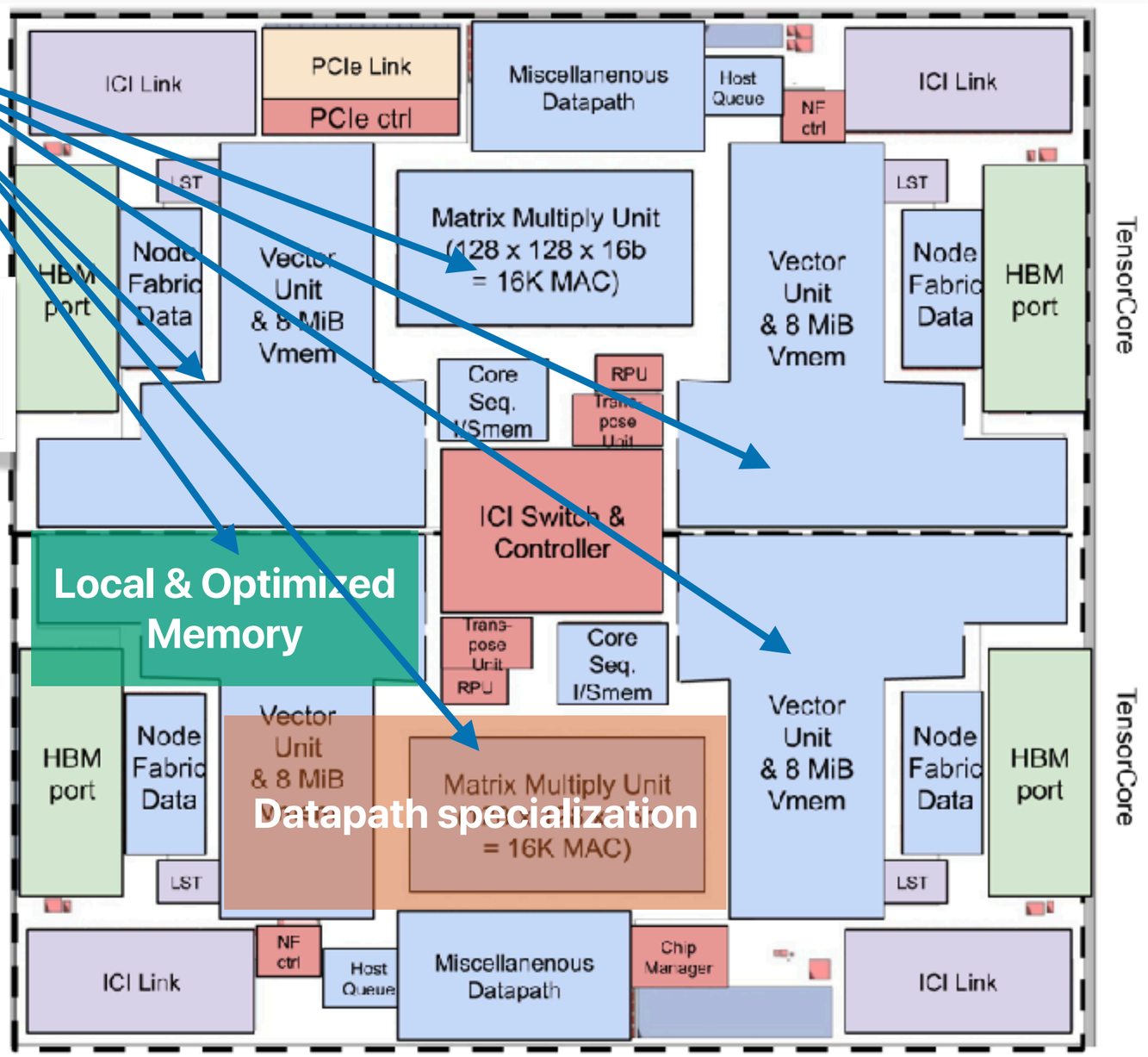
Google Tensor G1/G2/G3

TPUv2 floor plan

Parallelism

As instructions are sent over the relatively slow PCIe bus, TPU instructions follow the CISC tradition, including a repeat field. The average clock cycles per instruction (CPI) of these CISC instructions is typically 10 to 20. It has about a dozen

Reduced overhead



T. Norrie et al., "The Design Process for Google's Training Chips: TPUv2 and TPUv3," in IEEE Micro, vol. 41, no. 2, pp. 56-63

TensorFlow — the standard TPU programming interface

```
resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu='')
tf.config.experimental_connect_to_cluster(resolver)
# This is the TPU initialization code that has to be at the beginning.
tf.tpu.experimental.initialize_tpu_system(resolver)
print("All devices: ", tf.config.list_logical_devices('TPU'))

a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])

with tf.device('/TPU:0'):
  c = tf.matmul(a, b)

print("c device: ", c.device)
print(c)
```

Only very high-level mathematical/domain-specific functions are exposed to the user!

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 TPU can reduce the "IC" in the performance equation
  - ④ TPU features an instruction set architecture that facilitates pipelining

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

- *Pitfall: Being ignorant of architecture history when designing a domain-specific architecture.*

Ideas that didn't fly for general-purpose computing may be ideal for domain-specific architectures. For the TPU, three important architectural features date back to the early 1980s: systolic arrays [31], decoupled-access/execute [54], and CISC instructions [41]. The first reduced the area and power of the large matrix multiply unit, the second fetches weights concurrently during operation of the matrix multiply unit, and the third better utilizes the limited bandwidth of the PCIe bus for delivering instructions. History-aware architects could have a competitive edge.

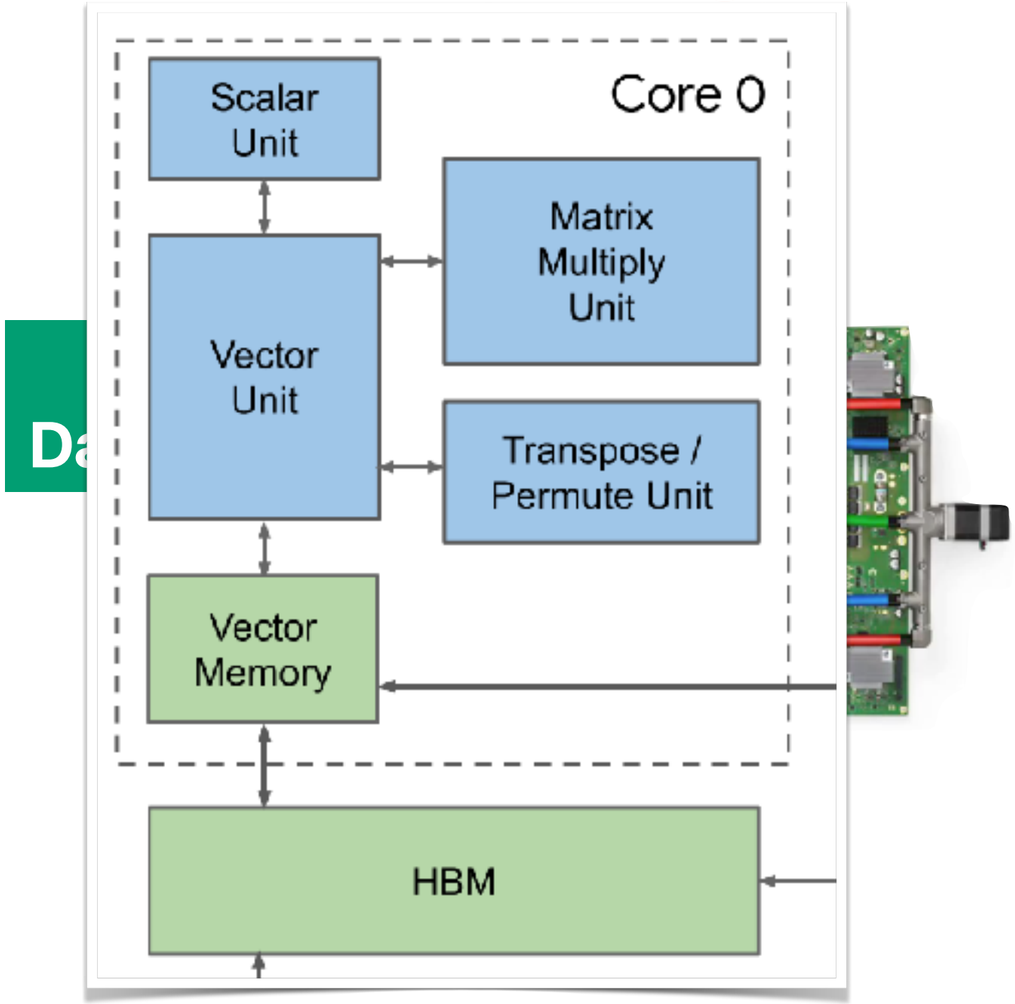
peak performance.

When using a mix of 8-bit weights and 16-bit activations (or vice versa), the Matrix Unit computes at half-speed, and it computes at a quarter-speed when both are 16 bits. It reads and

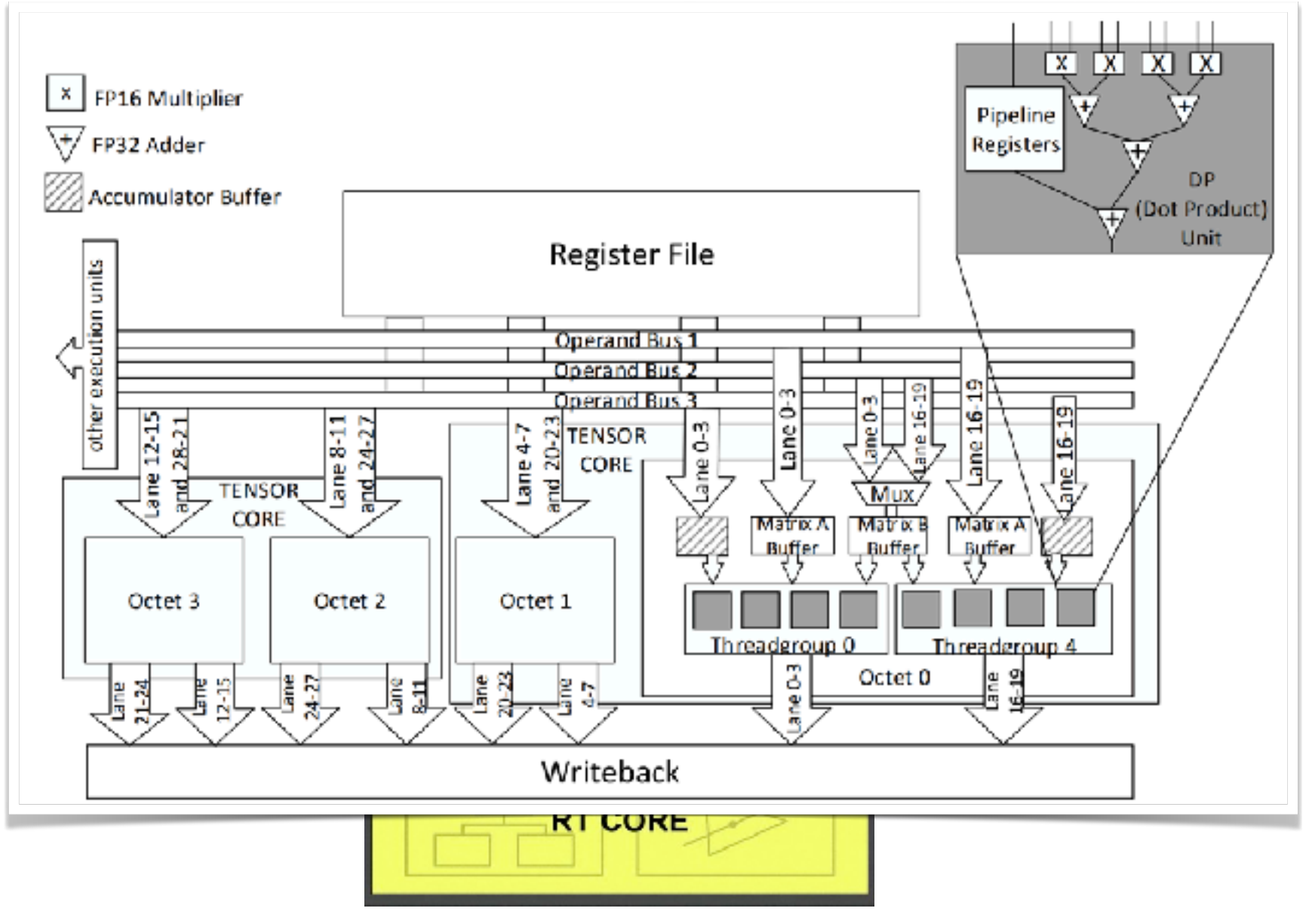
takes to shift a tile in). This unit is designed for dense matrices. Sparse architectural support was omitted for time-to-deployment reasons. The weights for the matrix unit are staged through an on-

As instructions are sent over the relatively slow PCIe bus, TPU instructions follow the CISC tradition, including a repeat field. The average clock cycles per instruction (CPI) of these CISC instructions is typically 10 to 20. It has about a dozen

Matrix processing — the core of AI/ML accelerators



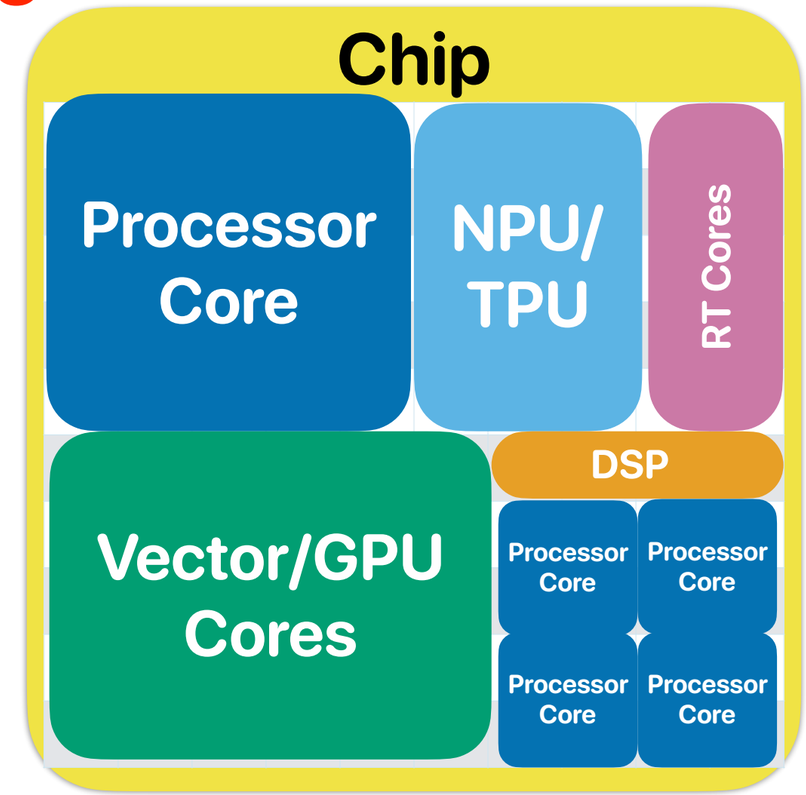
T. Norrie et al., "The Design Process for Google's Training Chips: TPUv2 and TPUv3," in IEEE Micro, vol. 41, no. 2, pp. 56-63



M. Raihan, N. Goli and T. Aamodt, "Modeling Deep Learning Accelerator Enabled GPUs, ISPASS 2019

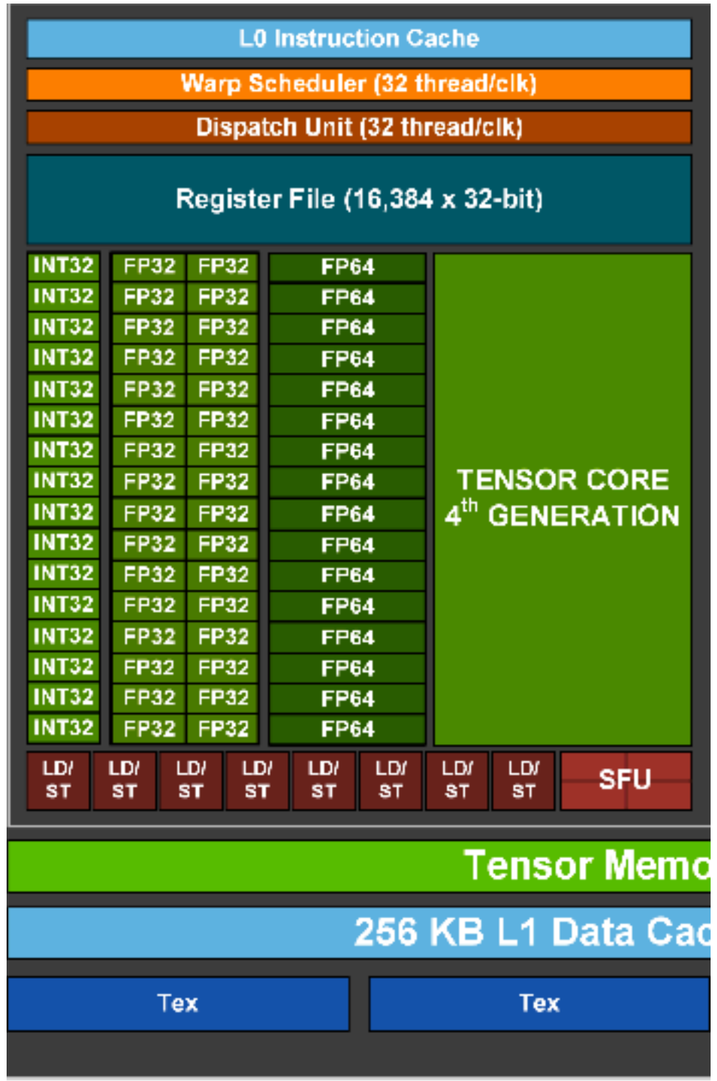
Given the same power budget, maximize the efficiency per chip

Some at top speed, some are not functioning.



Turn off unnecessary ones when we don't need specialized functions!

NVIDIA's Tensor Cores



What are Tensor Cores?

Tesla V100's Tensor Cores are programmable matrix-multiply-and-accumulate units that can deliver up to 125 Tensor TFLOPS for training and inference applications. The Tesla V100 GPU contains 640 Tensor Cores: 8 per SM. Tensor Cores and their associated data paths are custom-crafted to dramatically increase floating-point compute throughput at only modest area and power costs. Clock gating is used extensively to maximize power savings.

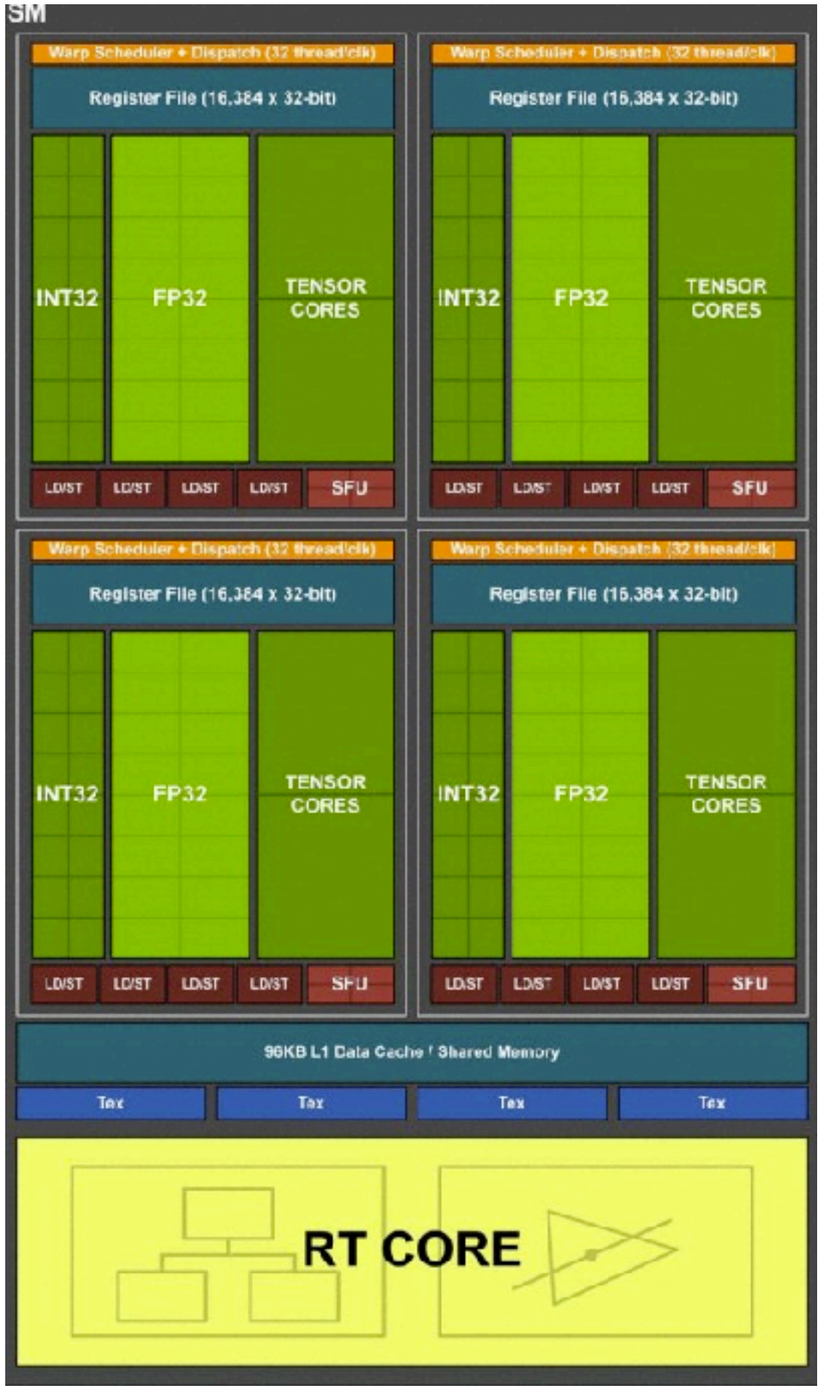
Each Tensor Core provides a 4x4x4 matrix processing array which performs the operation **D** = **A** * **B** + **C**, where **A**, **B**, **C** and **D** are 4×4 matrices as Figure 1 shows. The matrix multiply inputs **A** and **B** are FP16 matrices, while the accumulation matrices **C** and **D** may be FP16 or FP32 matrices.

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 FP16 FP16 FP16 or FP32

Figure 1: Tensor Core 4x4x4 matrix multiply and accumulate.

https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/



NVIDIA GPU =
CUDA Cores (vector)+
Tensor Cores (matrix) +
RT Cores (tree intersection)+
DPX Units (dynamic programming)

Programming in Turing Architecture

Use tensor cores

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

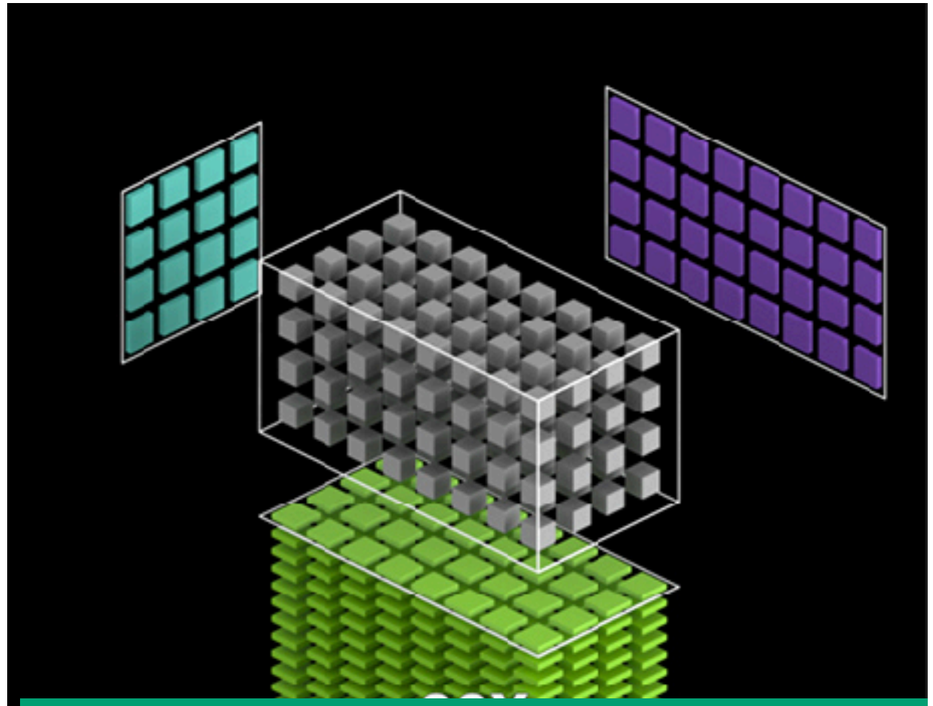
Make them 16-bit

```
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

How can Tensor Cores faster despite higher algorithm complexity?



Each tensor core operation performs 8x4x4 MMA in one cycle
~ 256 FP operations in conventional scalar models

	RTX 3090
CUDA Cores	10496
Tensor Cores	328

FLOPS of 8K by 8K matrix multiplications =

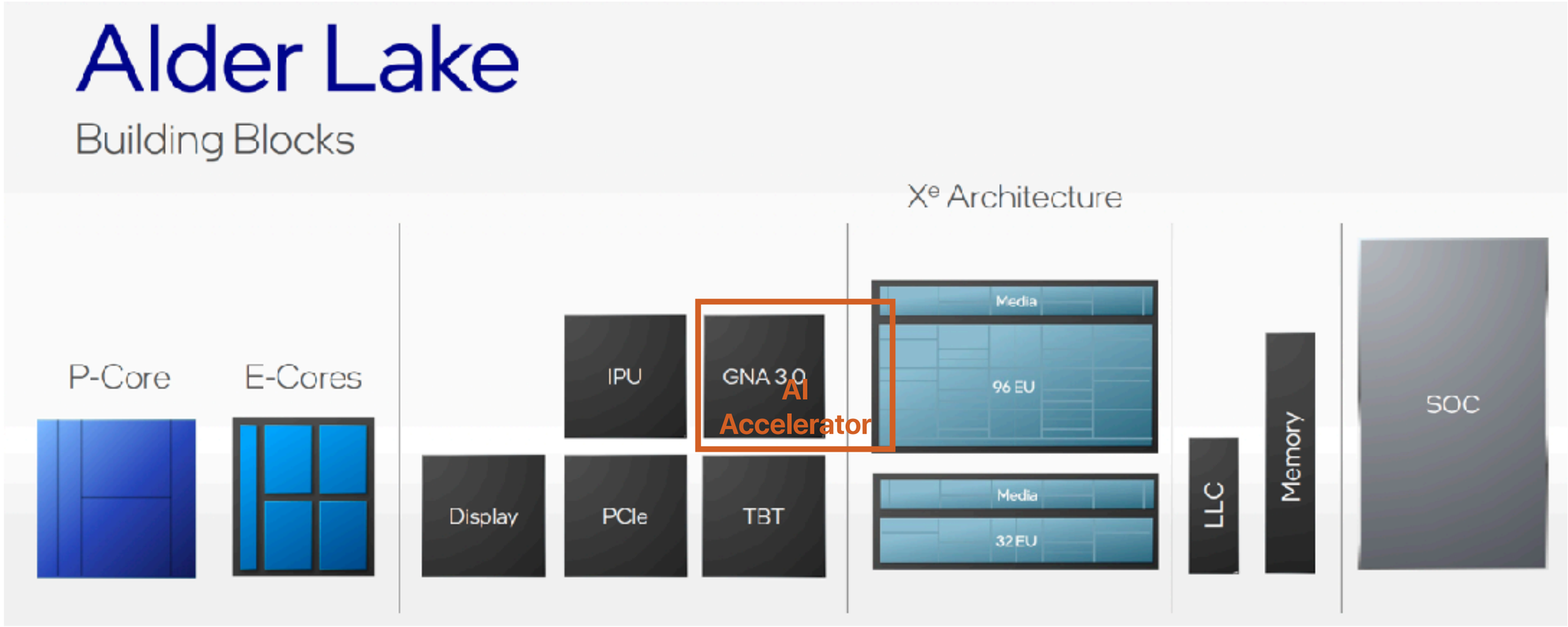
$$8192 \times 8192 \times 8192 \times 2$$

$$\frac{8192 \times 8192 \times 8192 \times 2}{10496} = 104{,}755{,}300 \ CUDA \ core \ cycles$$

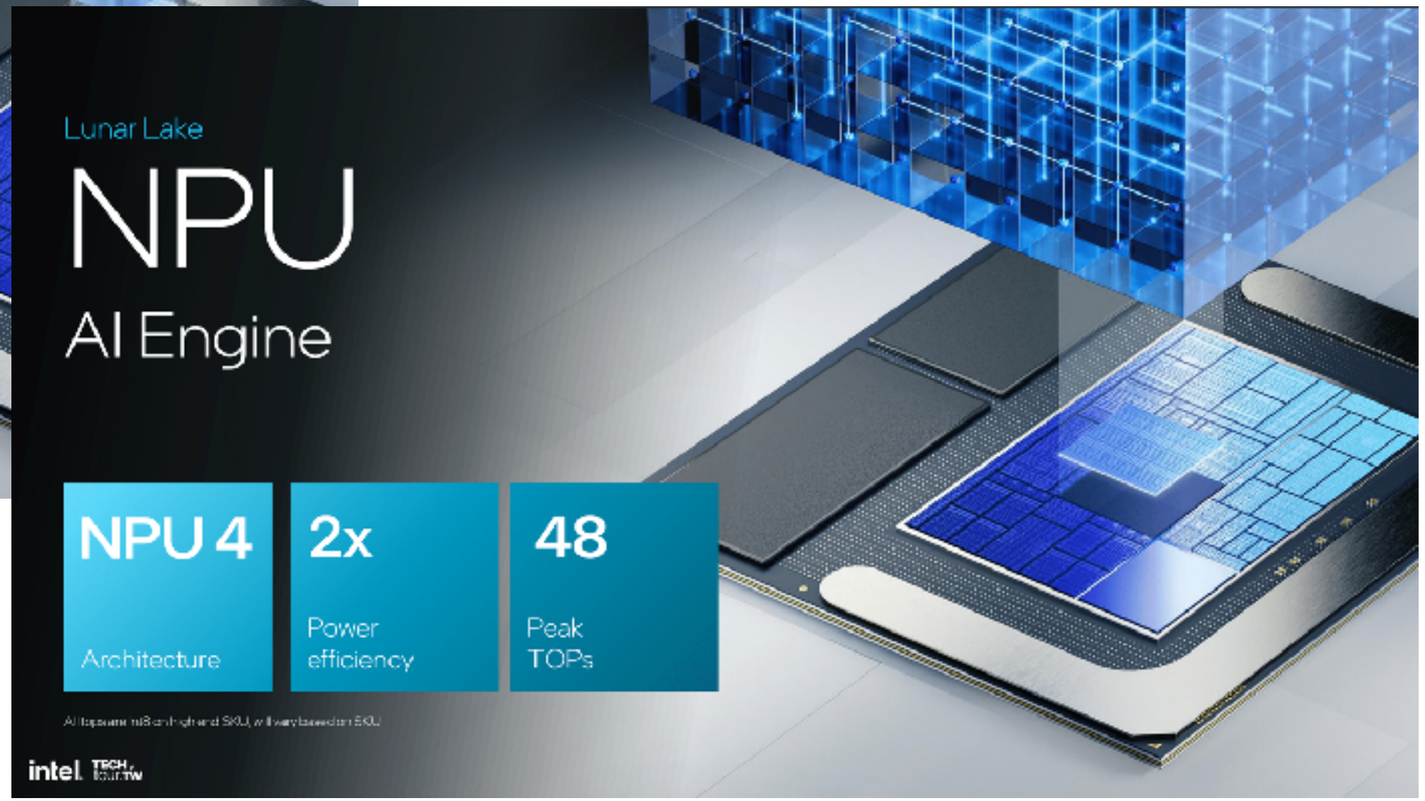
$$\frac{8192 \times 8192 \times 8192 \times 2}{328 \times 256} = 13{,}094{,}413 \ Tensor \ core \ cycles$$

Tensor cores are 8x faster

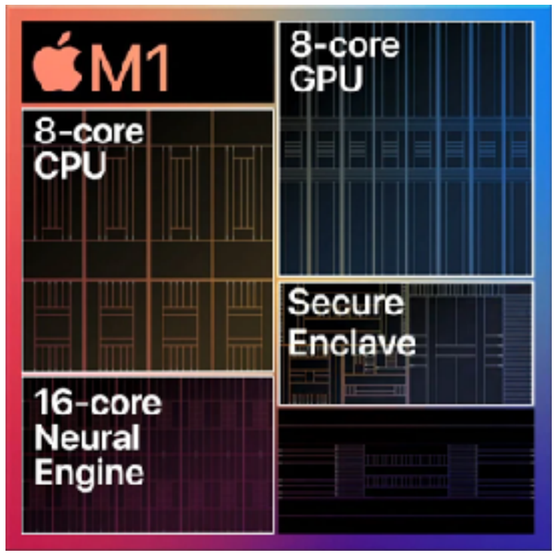
Modern processors also contain “accelerators”

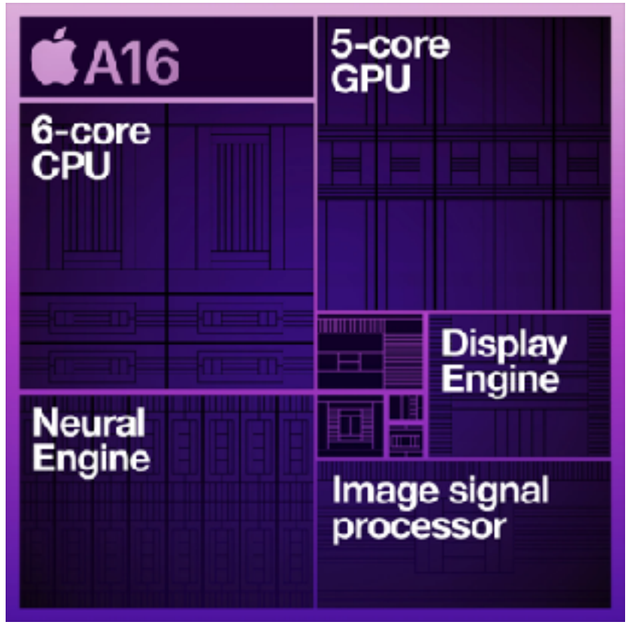


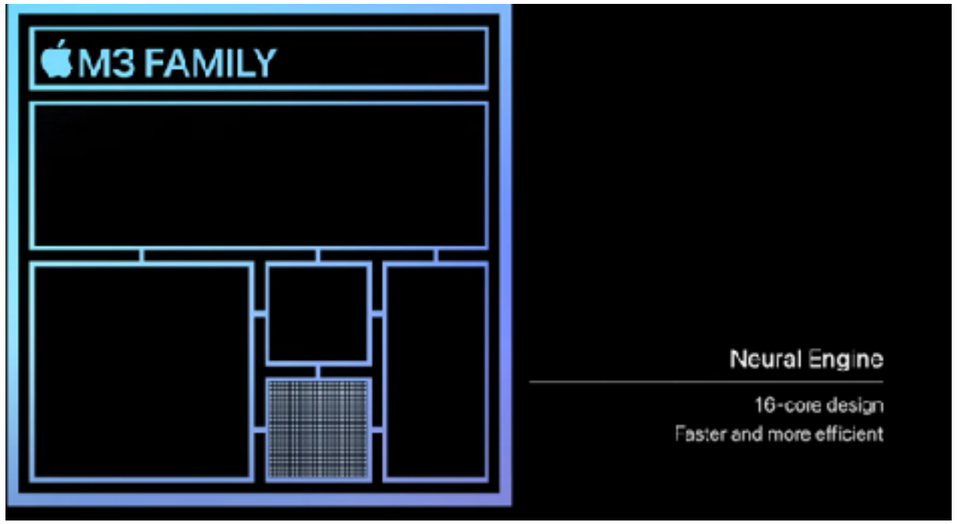
Intel's Neural Processing Unit

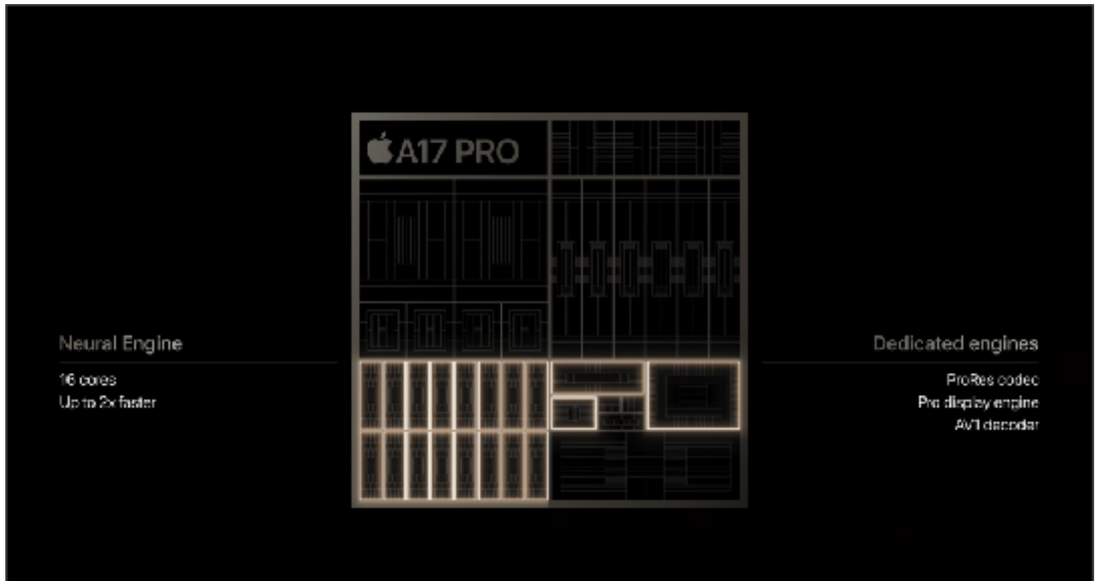


Apple's Neural Engine



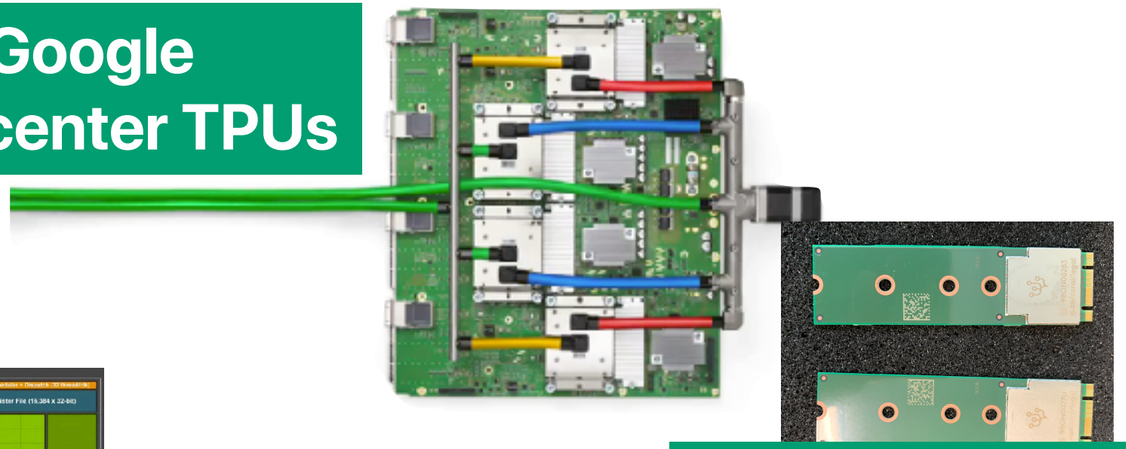






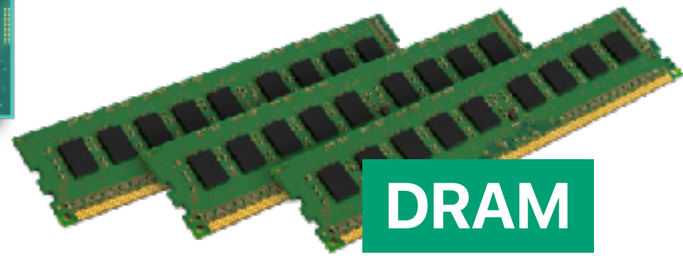
The “landscape” of modern computers

Google Datacenter TPUs

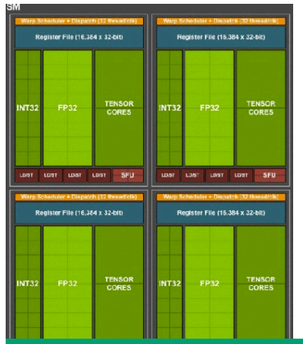


Google Edge TPUs

CPU

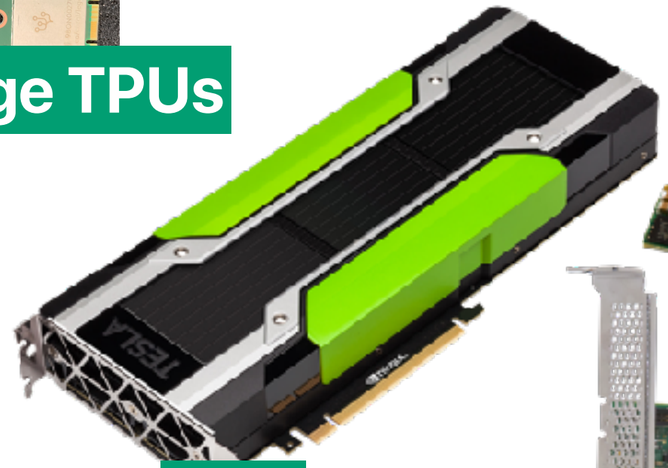


DRAM



NVIDIA Tensor Core Units

AI/ML Accelerators



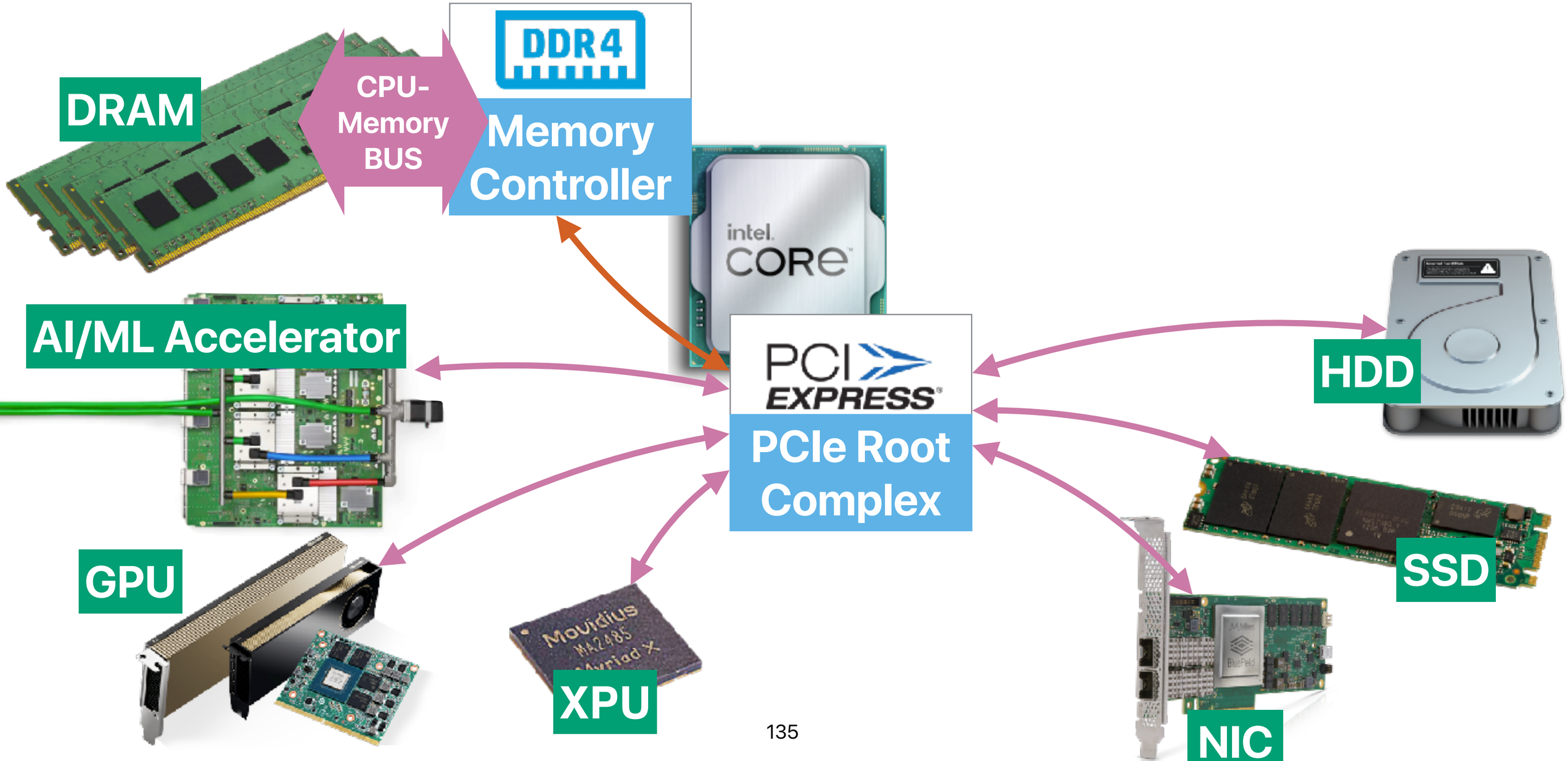
GPU

SSD

Apple Neural Engines

NIC

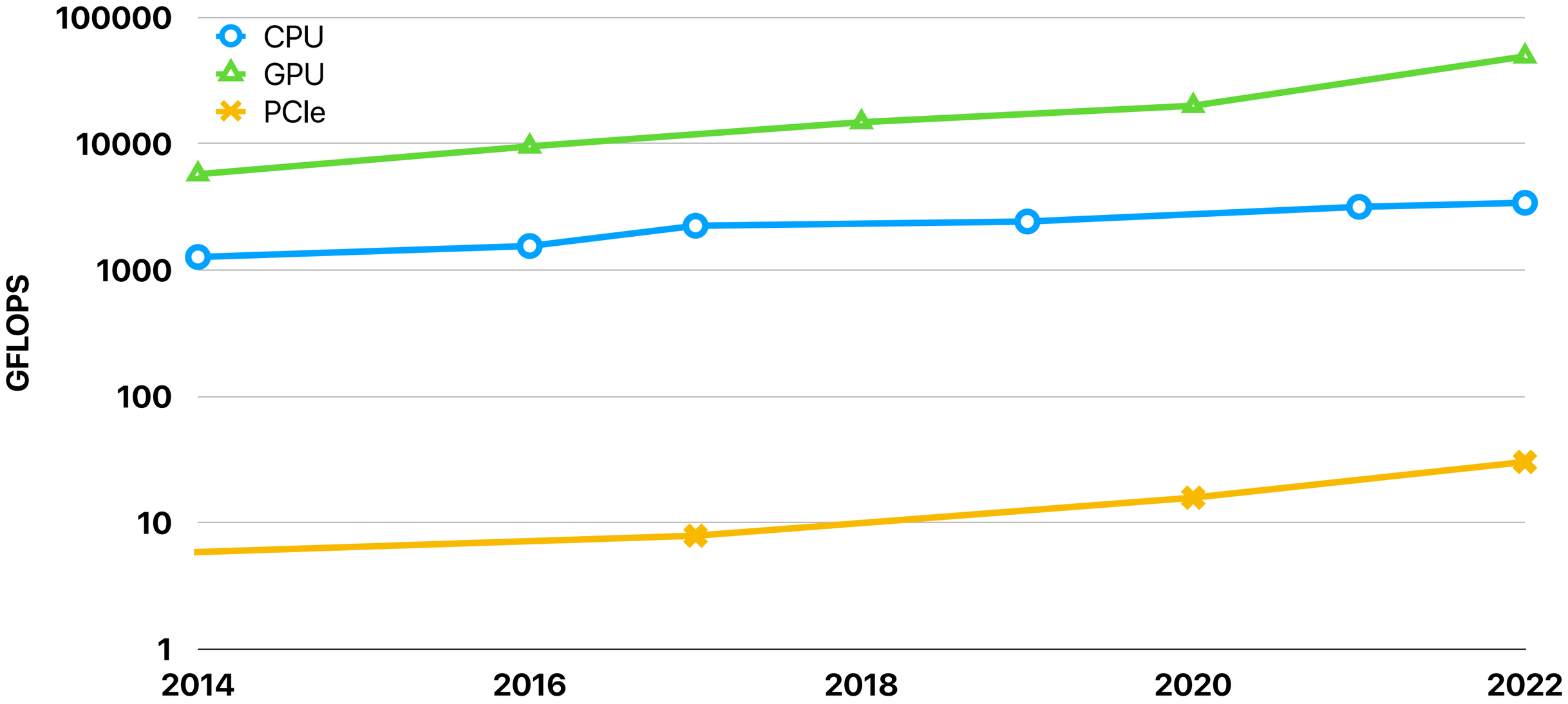
No free lunch! — Data movement as an overhead



Data movement overhead before calling GPUs

```
// Allocate memory space on the device
D_TYPE *d_a, *d_b, *d_c;
cudaMalloc((void **) &d_a, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE);
cudaMalloc((void **) &d_b, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE);
cudaMalloc((void **) &d_c, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE);

// copy matrix A and B from host to device memory
cudaMemcpy(d_a, a, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE, cudaMemcpyHostToDevice);

unsigned int grid_rows = (ARRAY_SIZE + BLOCK_SIZE - 1) / BLOCK_SIZE;
unsigned int grid_cols = (ARRAY_SIZE + BLOCK_SIZE - 1) / BLOCK_SIZE;
dim3 dimGrid(grid_cols, grid_rows);
dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);

// Launch kernel
gpu_block_matrix_mult<<<dimGrid, dimBlock>>>(d_a, d_b, d_c, ARRAY_SIZE);

// Transefr results from device to host
cudaMemcpy(c, d_c, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE, cudaMemcpyDeviceToHost);
cudaThreadSynchronize();
// time counting terminate
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);

// compute time elapse on GPU computing
cudaEventElapsedTime(&gpu_elapsed_time_ms, start, stop);
```

The “speed” of PCIe compared to computing



Take-aways: the new golden age of computer architectures

- Challenges and SOTA solutions in the dark silicon era
  - GPUs/many-core processors improve the **throughput** per-chip through providing massive parallelism where each processing element operates at a lower speed, but not-ideal for latency-sensitive workloads
  - Aggressive dynamic frequency/voltage scaling on CMP to accommodate the demand of **latency**-sensitive, parallelism-limited applications, but the area-efficiency of the slower cores is not great
  - Single ISA, heterogeneous CMPs (e.g., big.Little cores, Intel/Apple's P-cores/E-cores) find a balance the trade-offs of general-purpose workloads, but won't be ideal if your applications go to either extreme of throughput or latency
  - Domain-specific accelerators or application-specific ICs make more efficient use of chip areas to fulfill the latency or throughput demand of applications, but sacrifices flexibility and application designers are now also hardware designers

Conclusion: A New Golden Age