

Lecture 1:

**Course Introduction +
Review of Throughput HW Architecture**

**Visual Computing Systems
Stanford CS348K, Spring 2023**

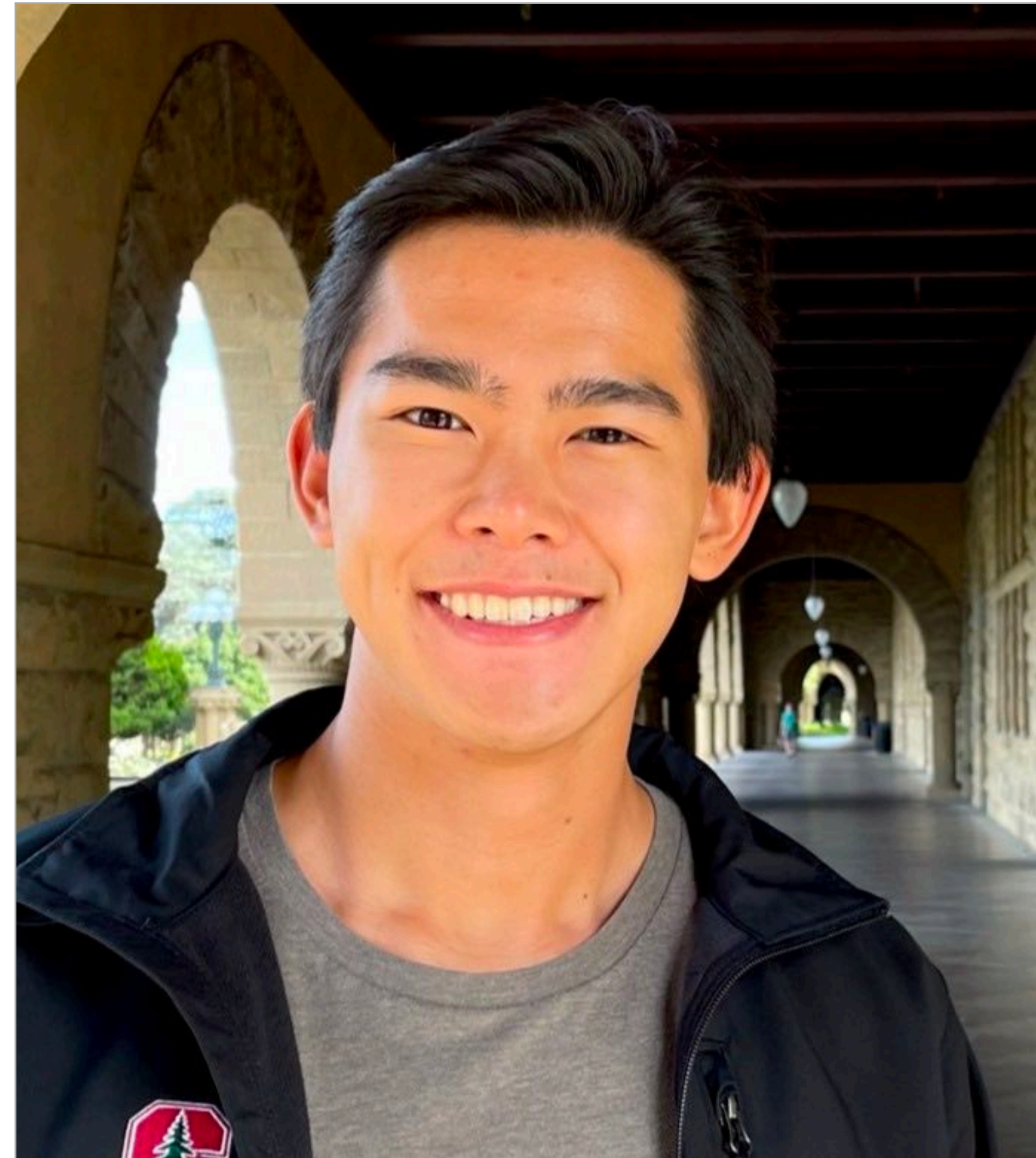
Hello from the course staff

Your instructor (me)



Prof. Kayvon

Your CA



Arden Ma

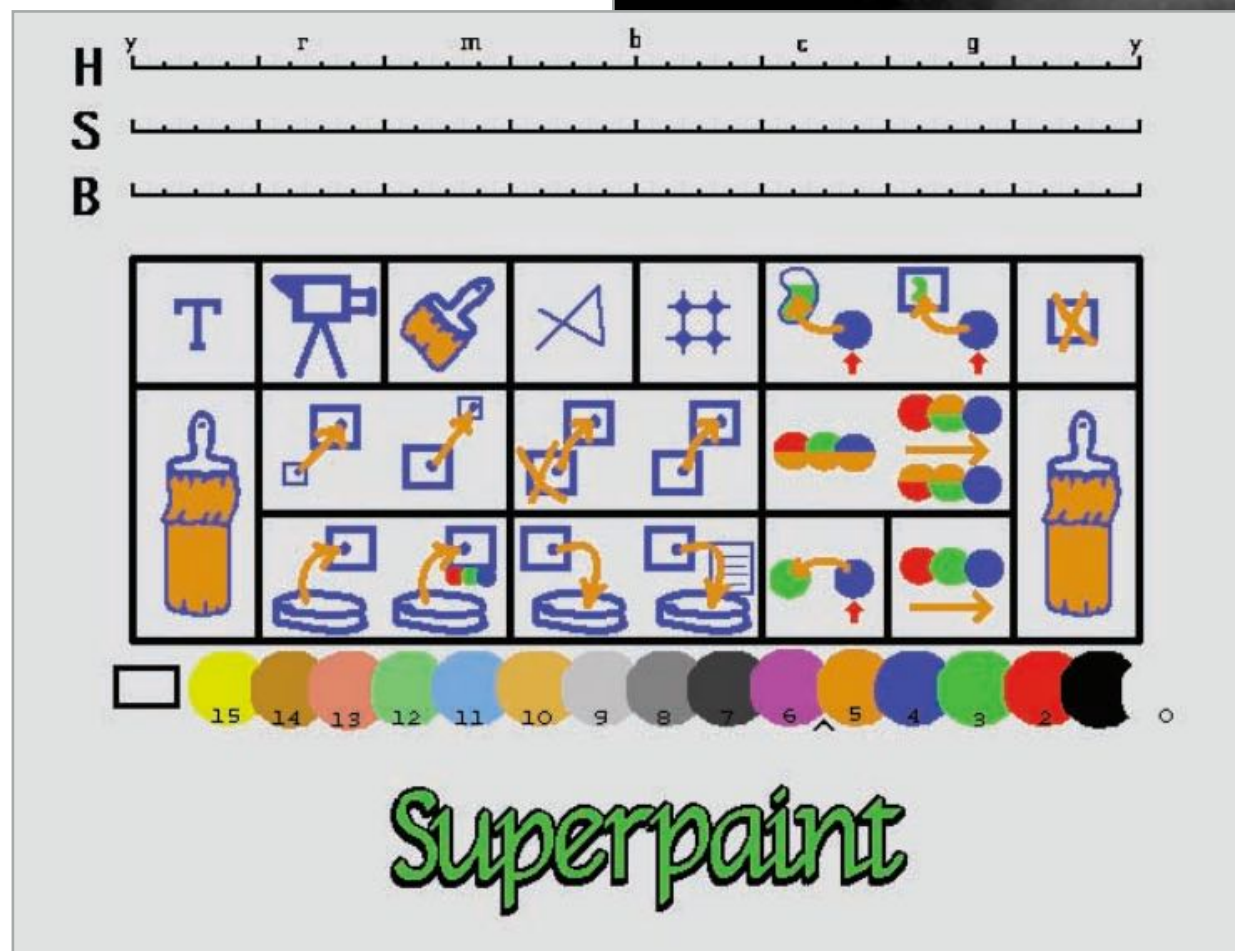
**Visual computing applications have always demanded
some of the world's most advanced parallel computing systems**



Ivan Sutherland's Sketchpad on MIT TX-2 (1962)

The frame buffer

Shoup's SuperPaint (PARC 1972-73)



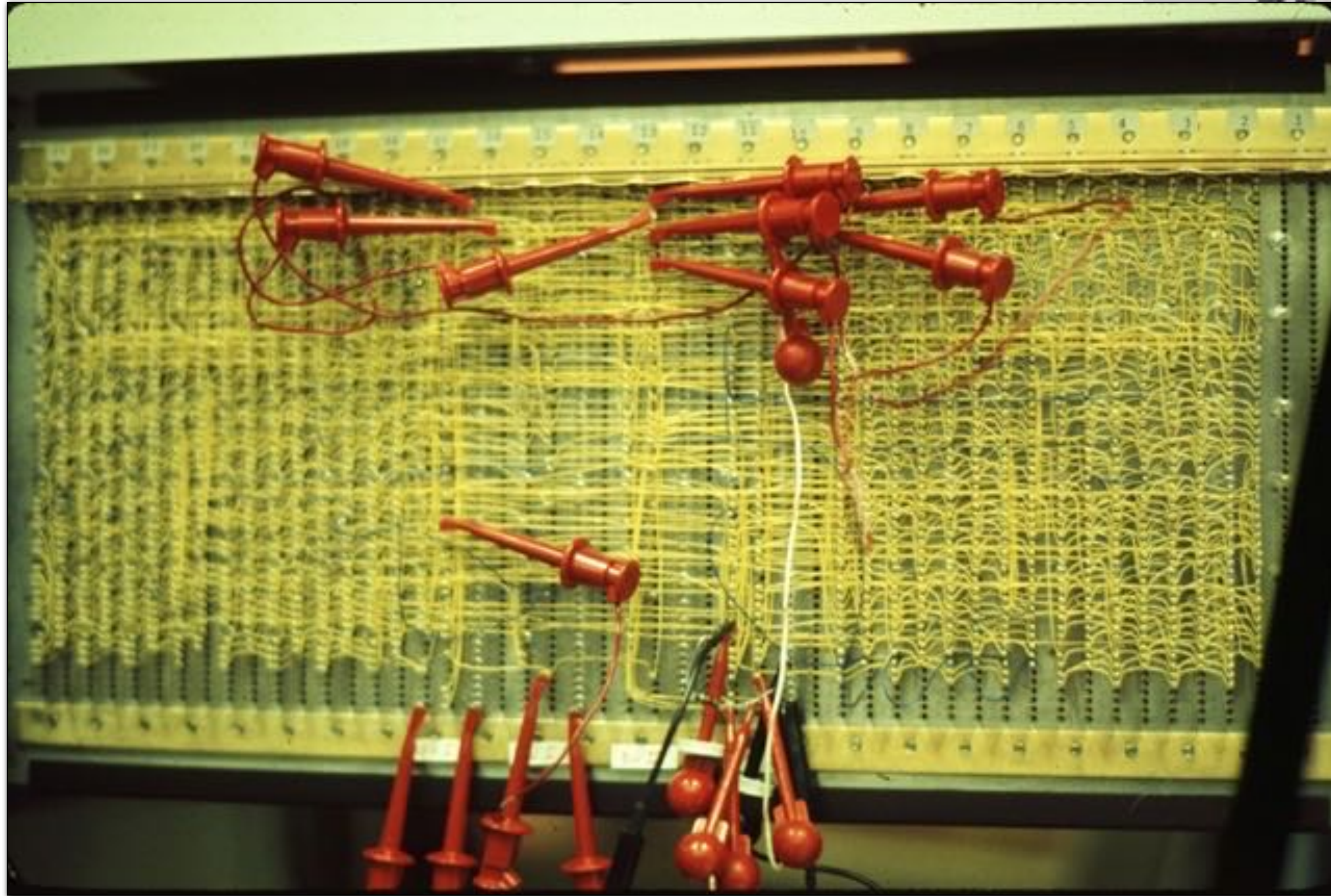
16 2K shift registers (640 x 486 x 8 bits)



COMPUTER HISTORY MUSEUM

The frame buffer

Shoup's SuperPaint (PARC 1972-73)

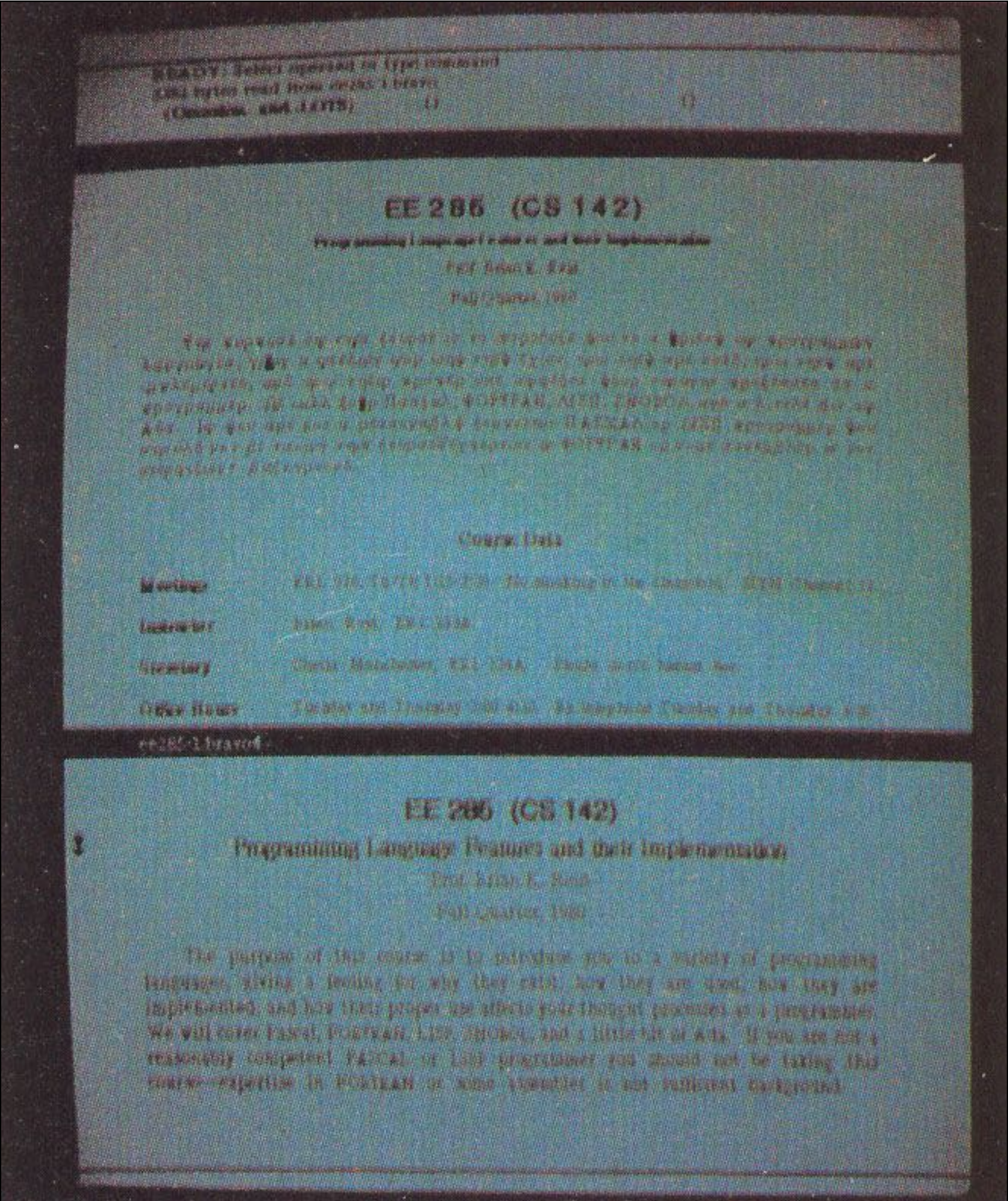


16 2K shift registers (640 x 486 x 8 bits)

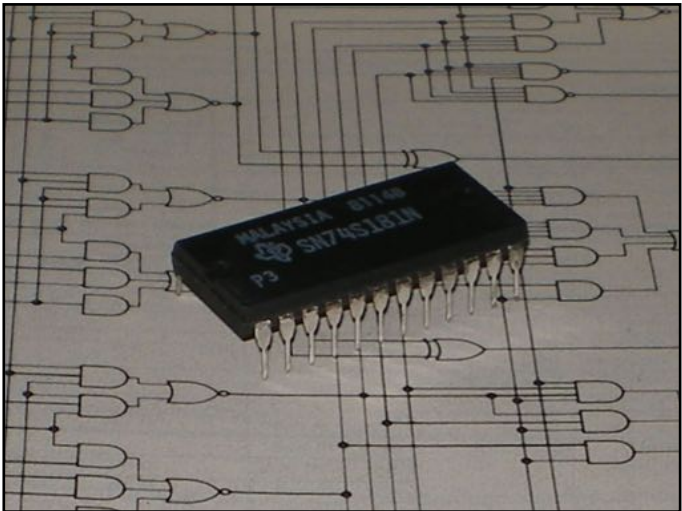


COMPUTER
HISTORY
MUSEUM

Xerox Alto (1973)



Bravo (WYSIWYG)



TI 74181 ALU

Goal: render everything you've ever seen

“Road to Pt. Reyes”
LucasFilm (1983)



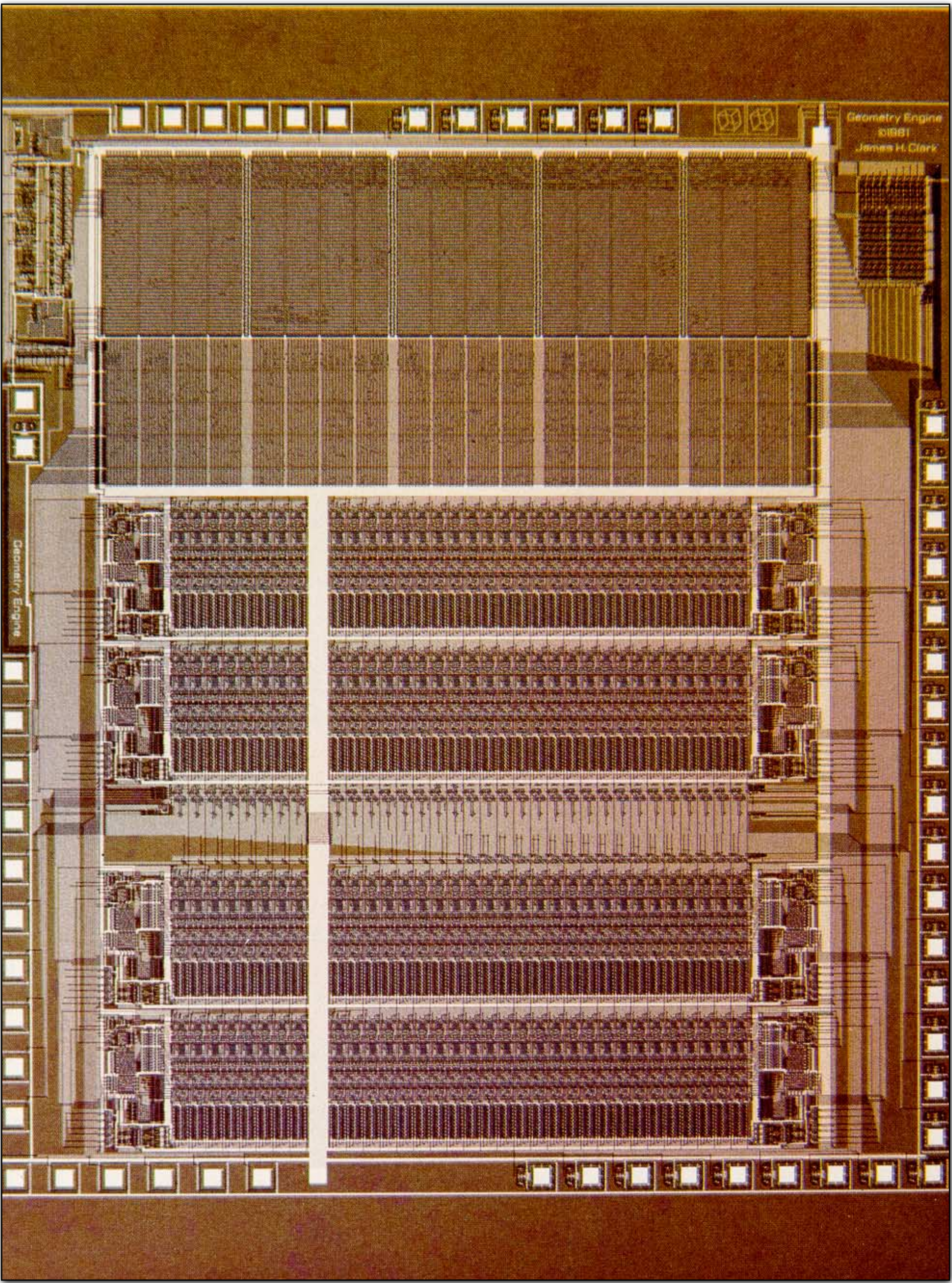
Pixar's Toy Story (1995)



**“We take an average of three hours to draw a single frame on the fastest computer money can buy.”
- Steve Jobs**

Clark's geometry engine (1982)

ASIC for geometric transforms
used in real-time graphics



NVIDIA Titan RTX 3090 GPU



~ 40 TFLOPs fp32 *

4X flops of ASCI Q (top US supercomputer circa 2002) **

* doesn't about 70 TFLOPS of ray tracing compute + 320 TFLOPS of DNN compute

** not apples-to-apples since ASCI Q is double precision flops



FORZA 7 MOTORSPORT



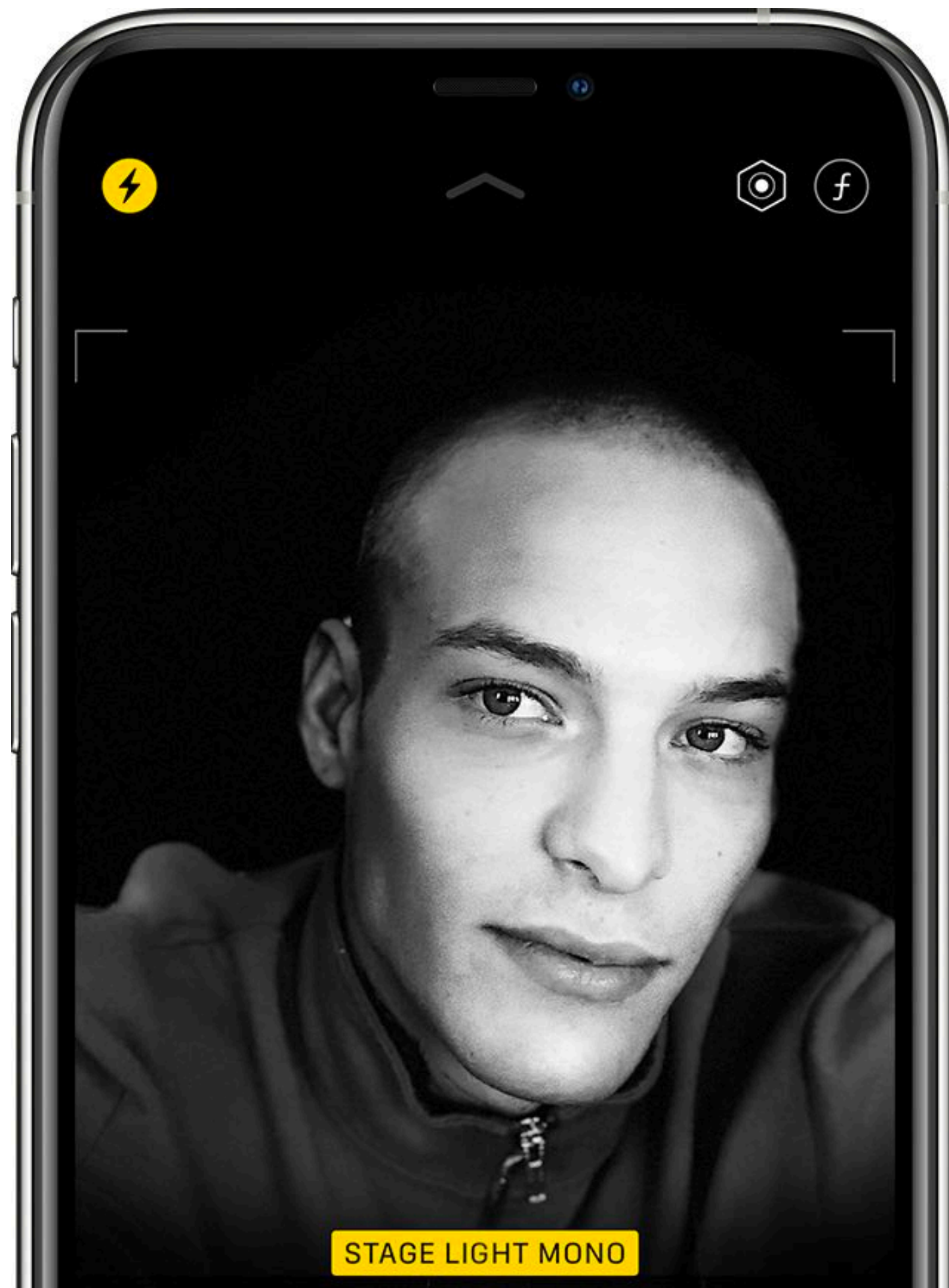
Unreal 5 Demo (Nanite renderer)



Digital photography: major driver of compute capability of modern smartphones

Portrait mode

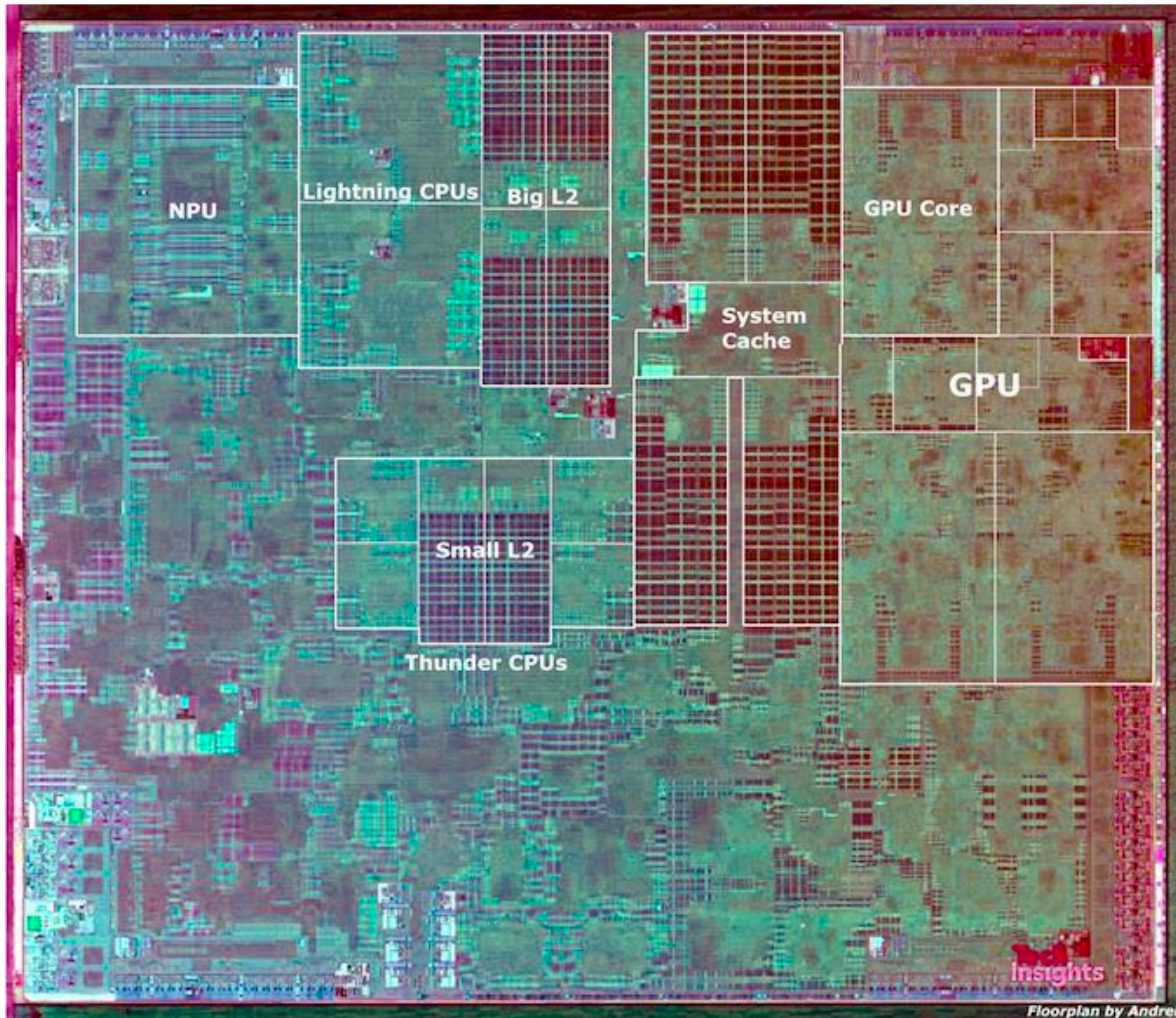
(simulate effects of large aperture DSLR lens)



High dynamic range (HDR) photography



Modern smartphones utilize multiple processing units to quickly generate high-quality images



Apple A13 Bionic

Multi-core CPU (heterogeneous cores)

Multi-core GPU

Neural accelerator

Sensor processing accelerator

Video compression/decompression HW

Etc...

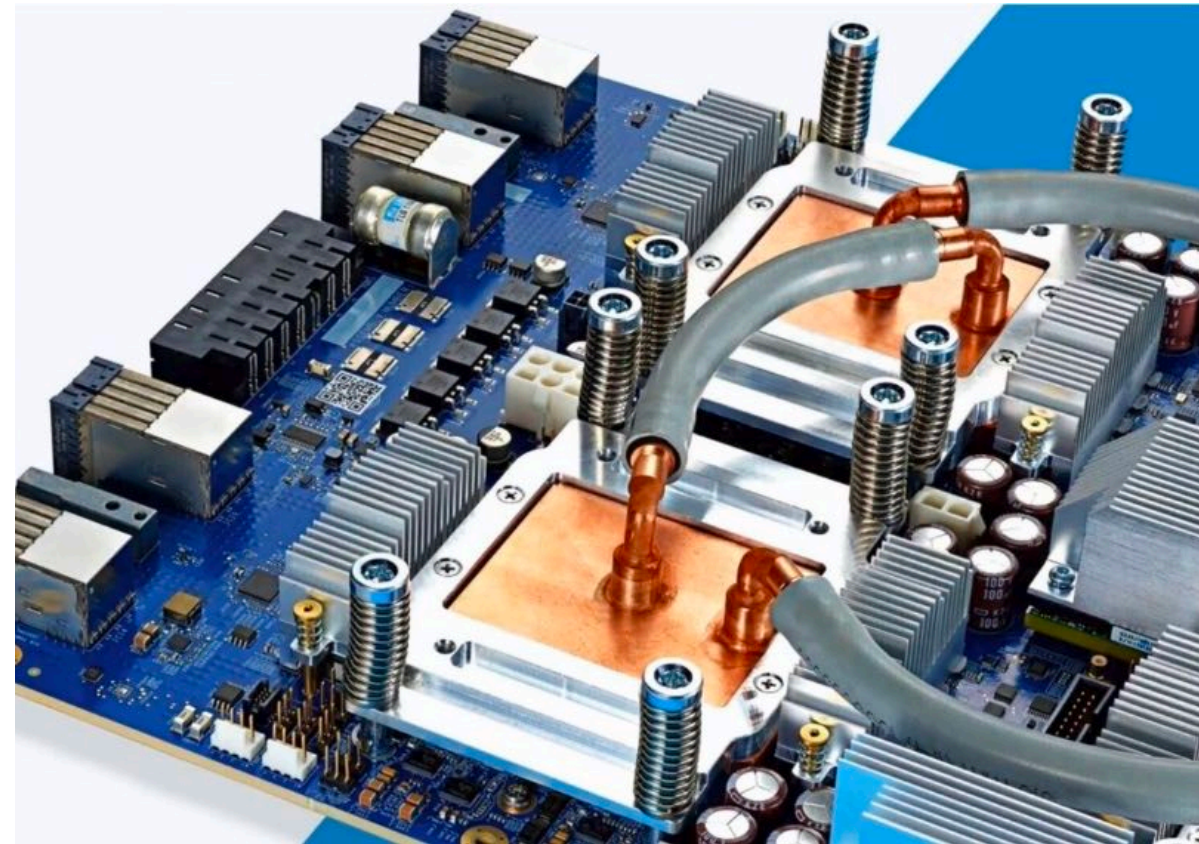
Oculus Quest 2 headset (2020)



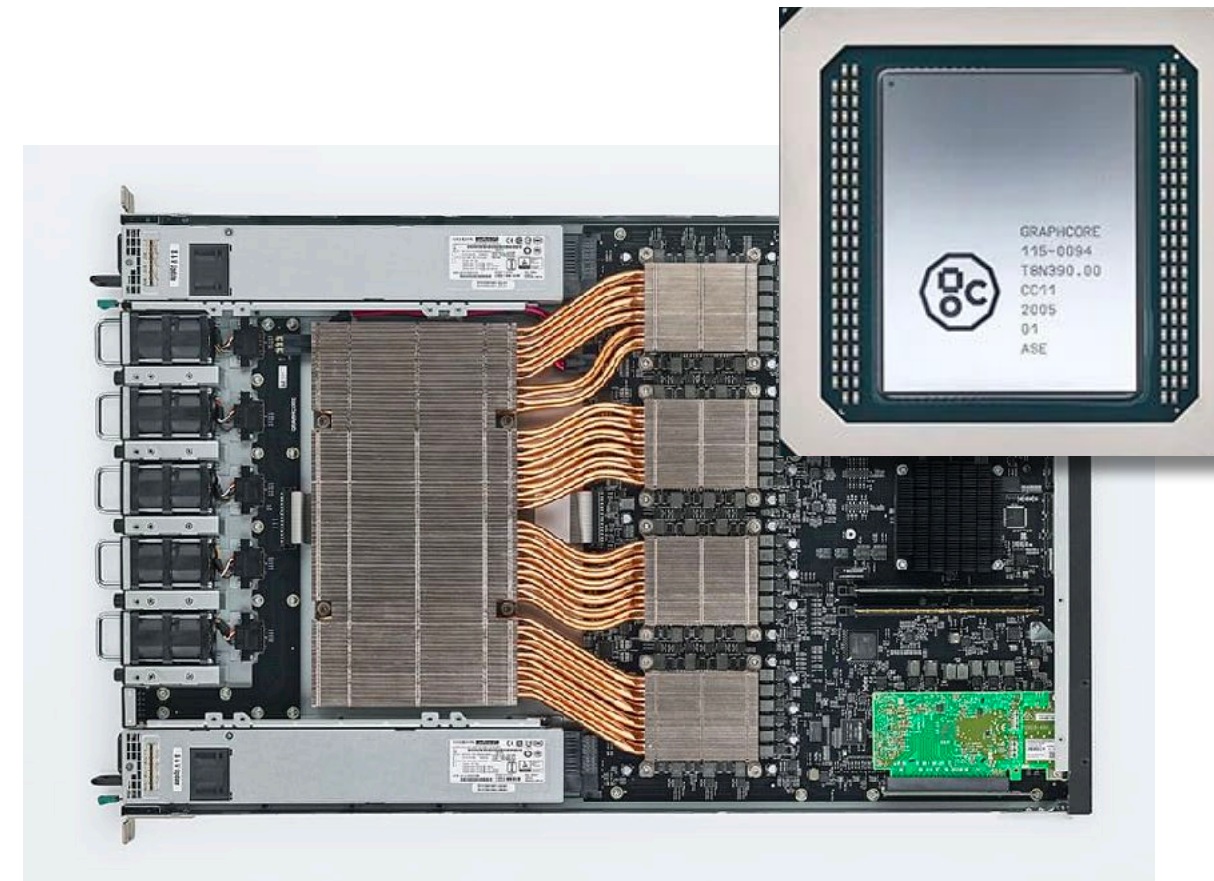
Image/video analysis via deep learning



Hardware acceleration of DNN inference/training



Google TPU3



GraphCore IPU



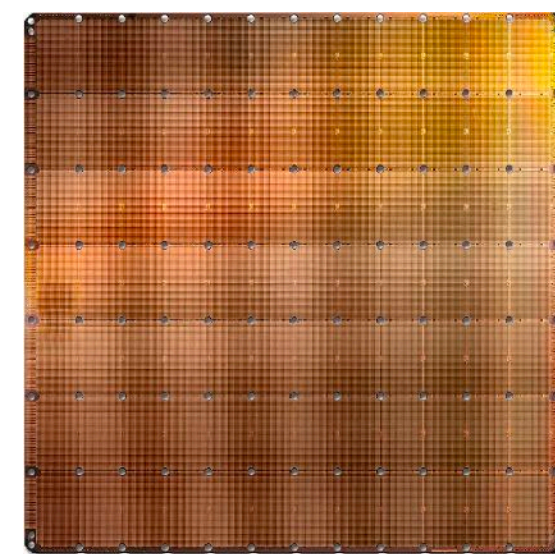
Apple Neural Engine



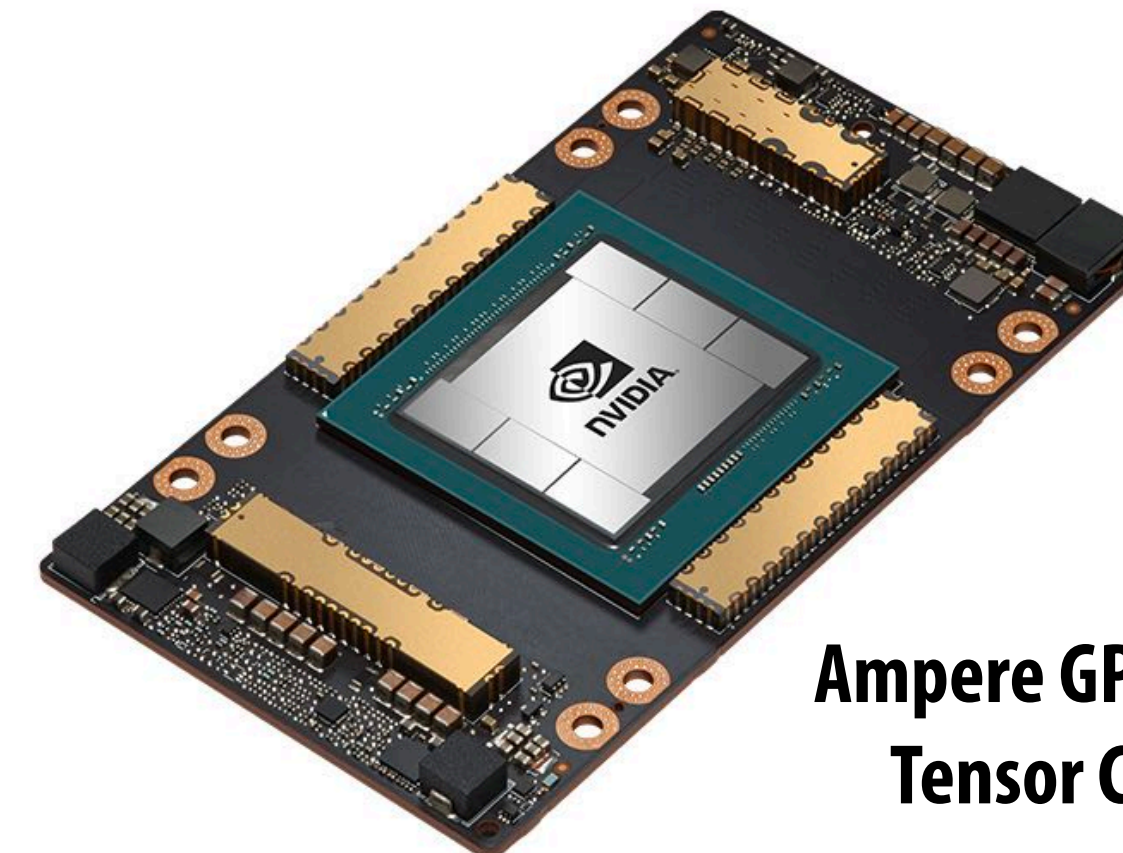
Intel Deep Learning Inference Accelerator



SambaNova Cardinal SN10



Cerebras Wafer Scale Engine



Ampere GPU with Tensor Cores

Datacenter-scale applications



Google TPU pods

Image Credit: TechInsights Inc.

AI generated visual context

[ControlNet 2023]

Input (Canny Edge)



Default



Automatic Prompt

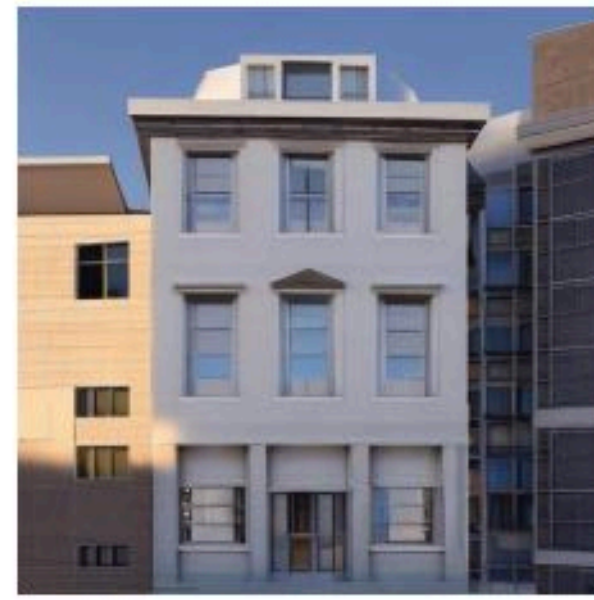


“a man with beard sitting with two children”

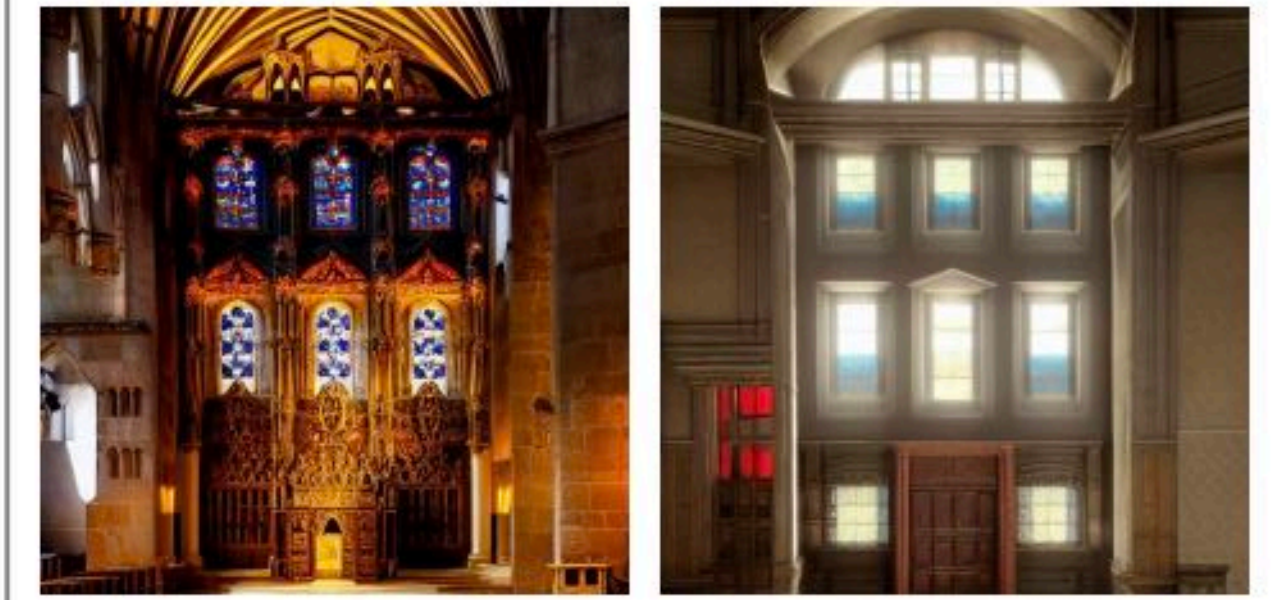
User Prompt



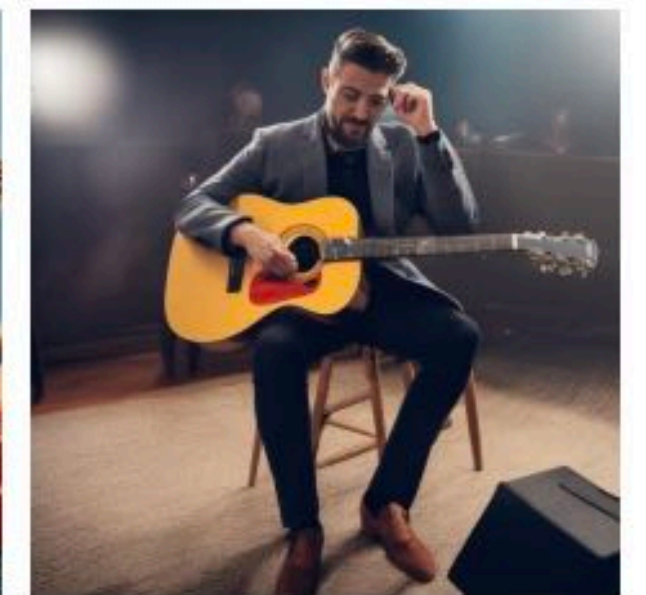
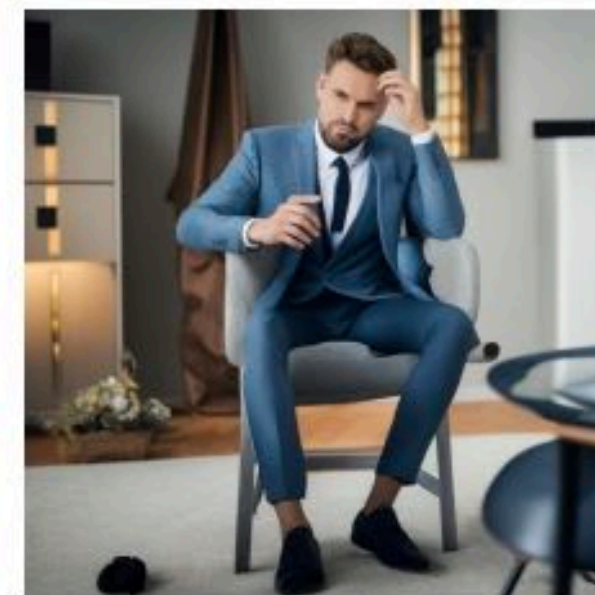
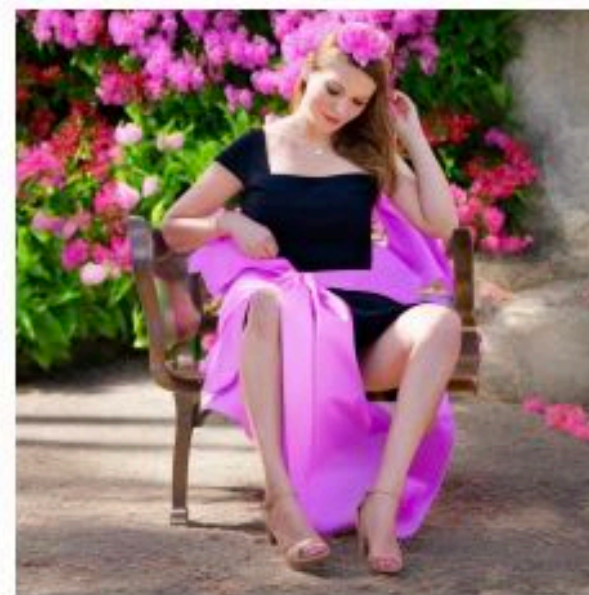
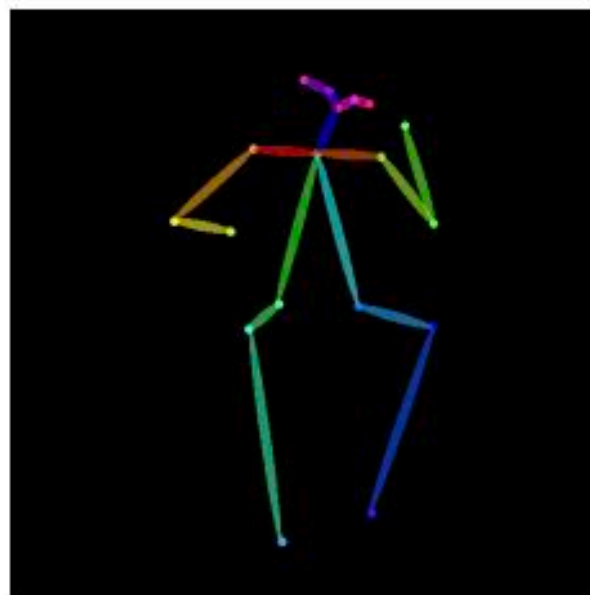
“mother and two boys in a room, masterpiece, artwork”



“a building in a city street”



“inside a gorgeous 19th century church”



astronaut

“music”



Youtube

Transcode, stream, analyze...

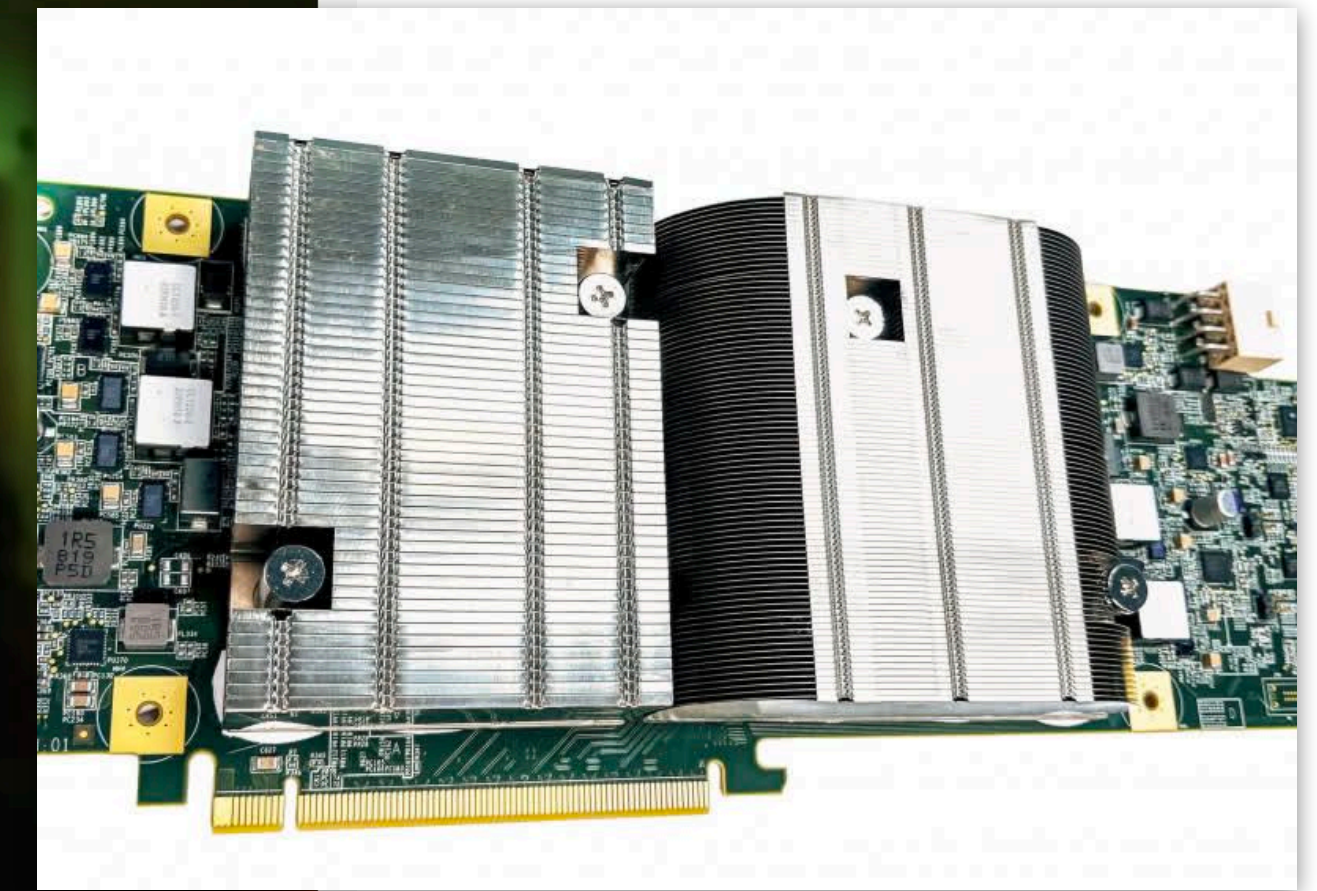


#LuisFonsi #Despacito #Imposible

Luis Fonsi - Despacito ft. Daddy Yankee

6,703,305,990 views • Jan 12, 2017

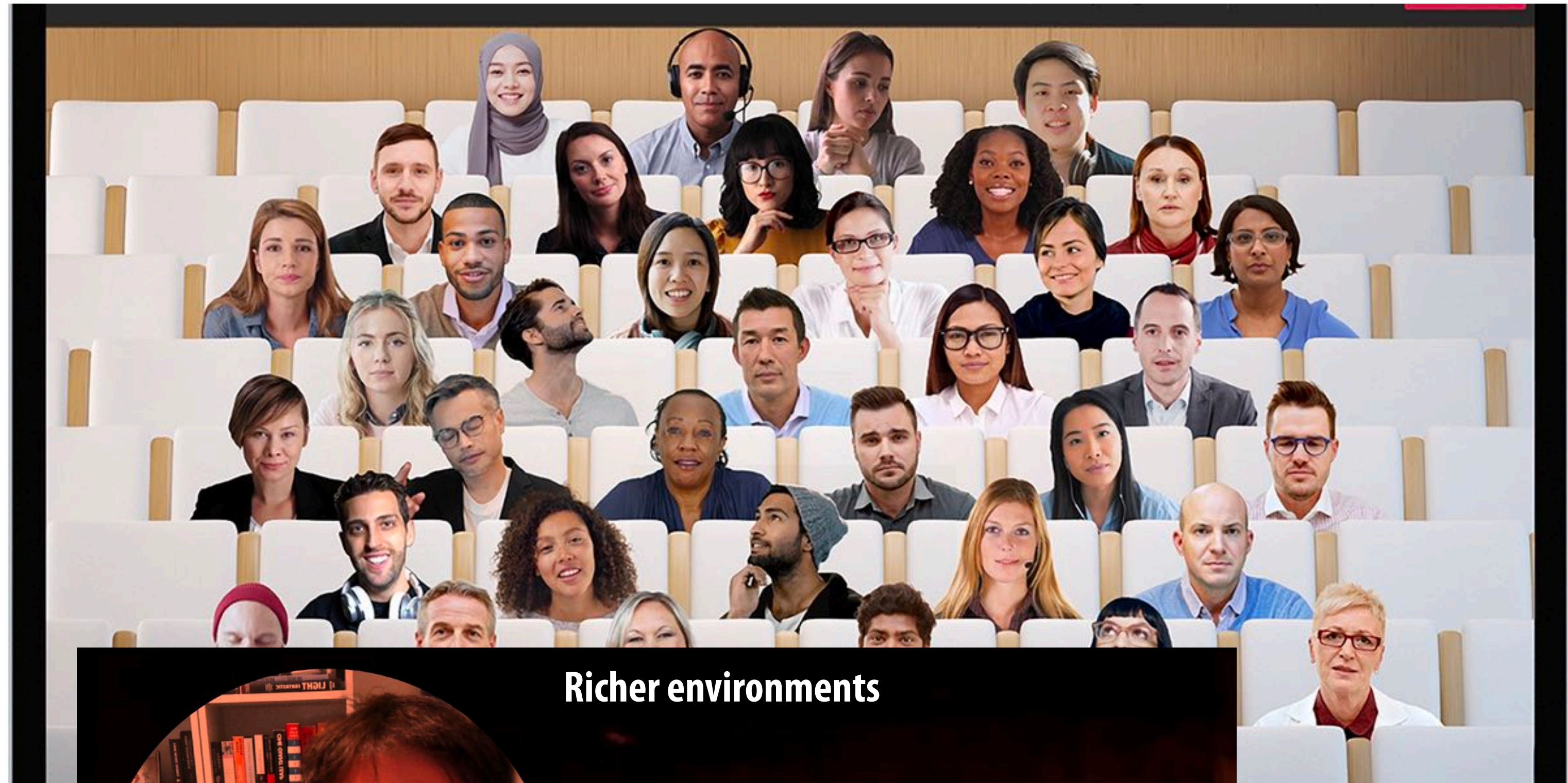
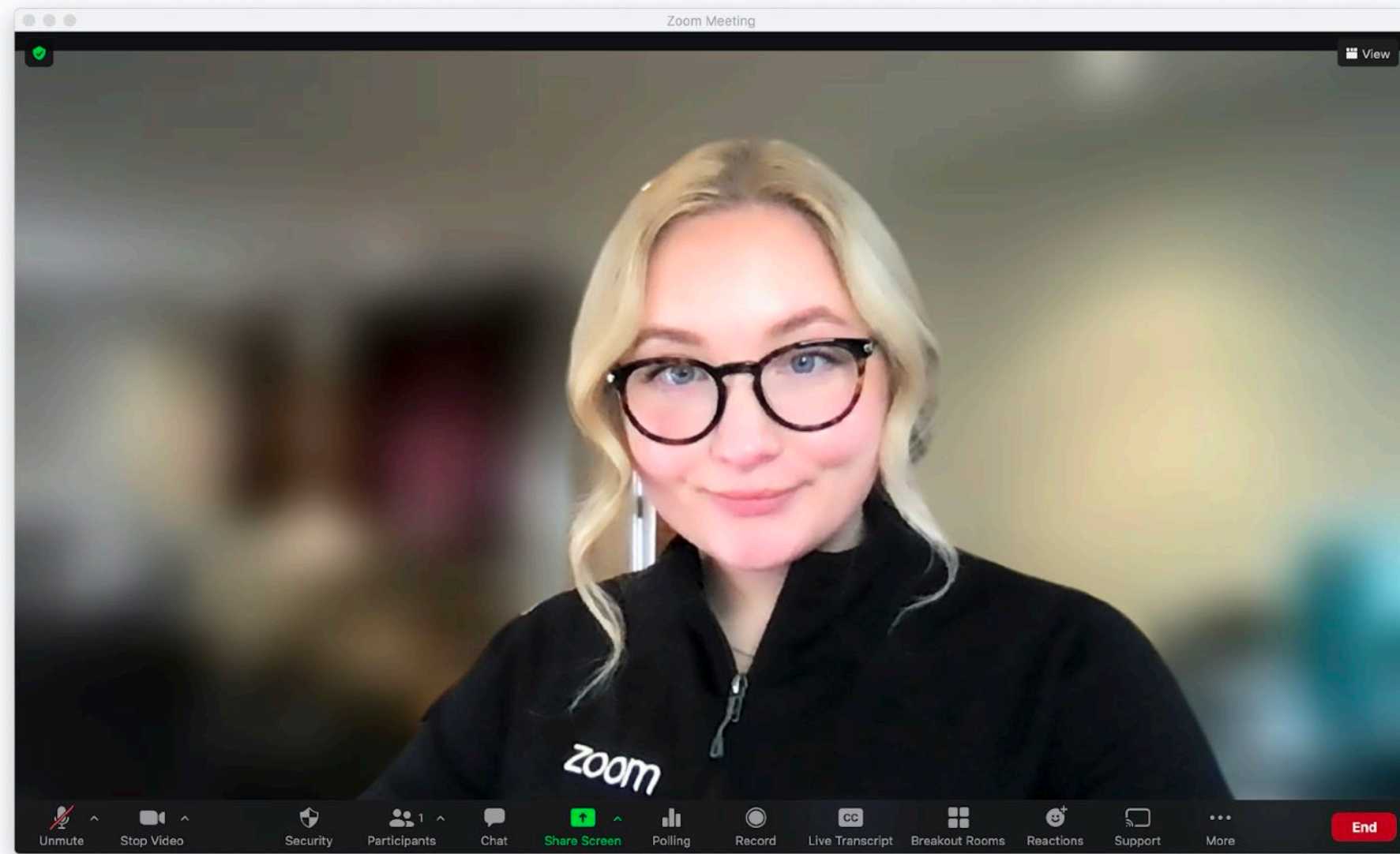
36M 4.4M SHARE SAVE ...



Google VPU transcoding HW

Video communication

Background blur



Richer environments



Add effects

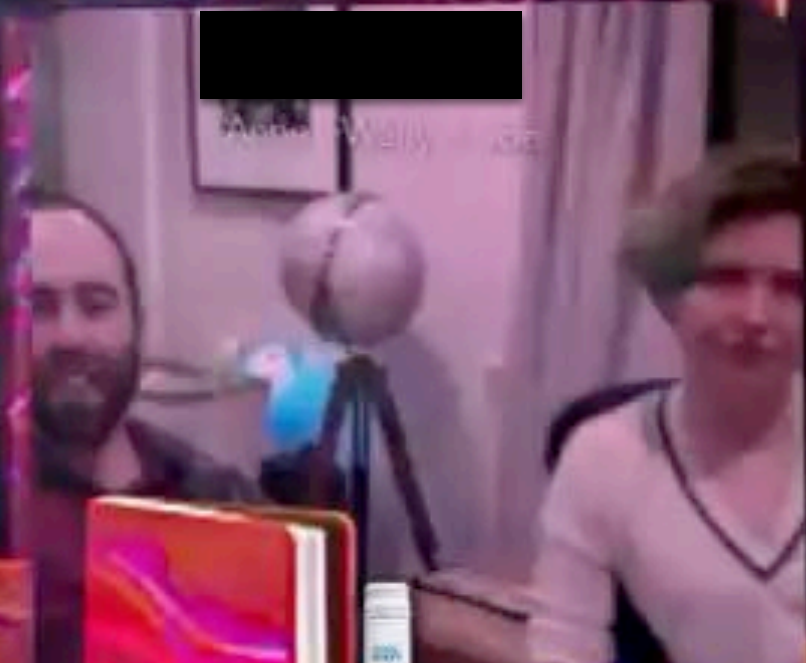




Clean up

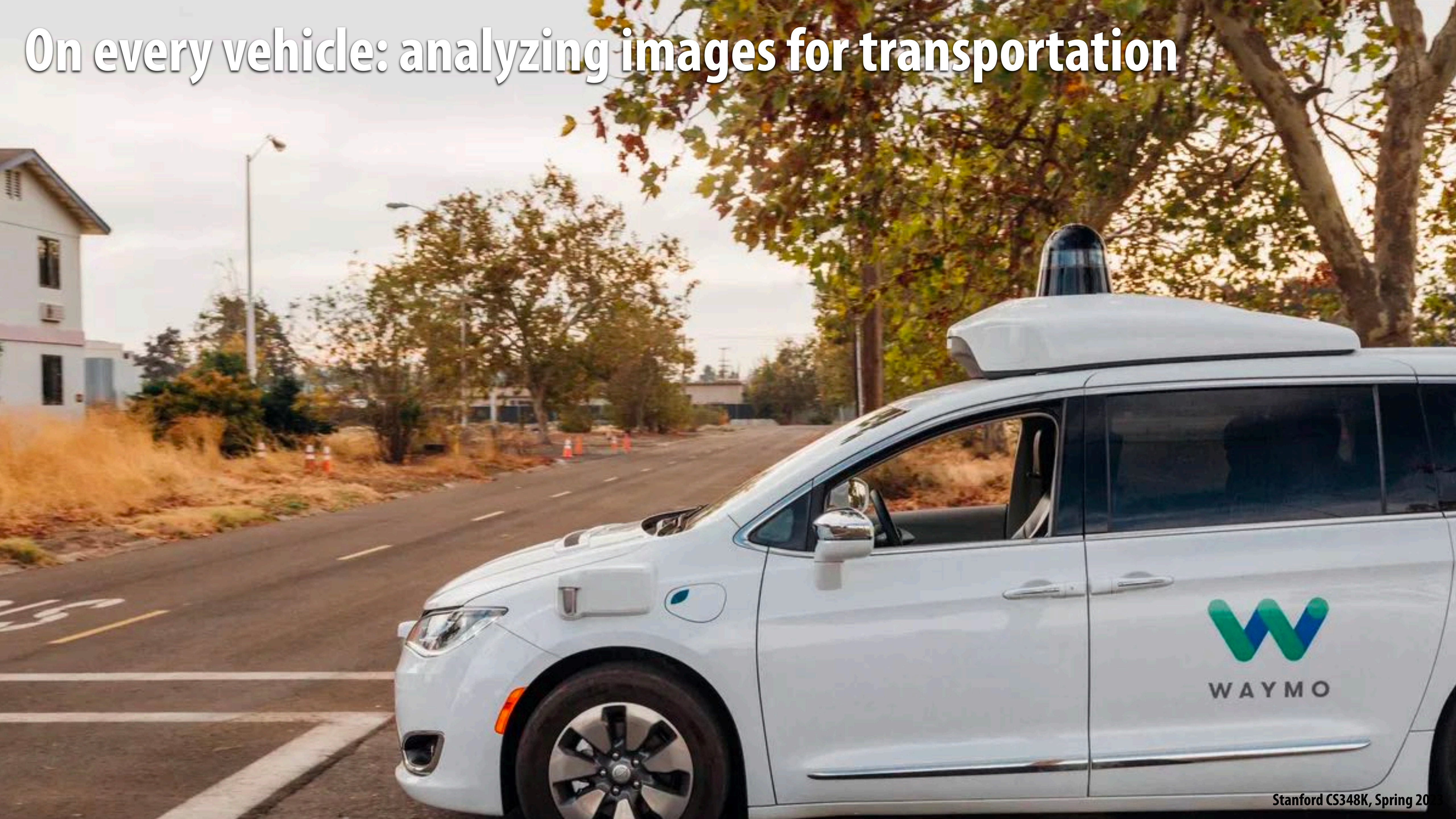


Red and black antennae waving
They all do it the same



Back to Karaoke Shower

On every vehicle: analyzing images for transportation



What is this course about?

Accelerator hardware architecture?

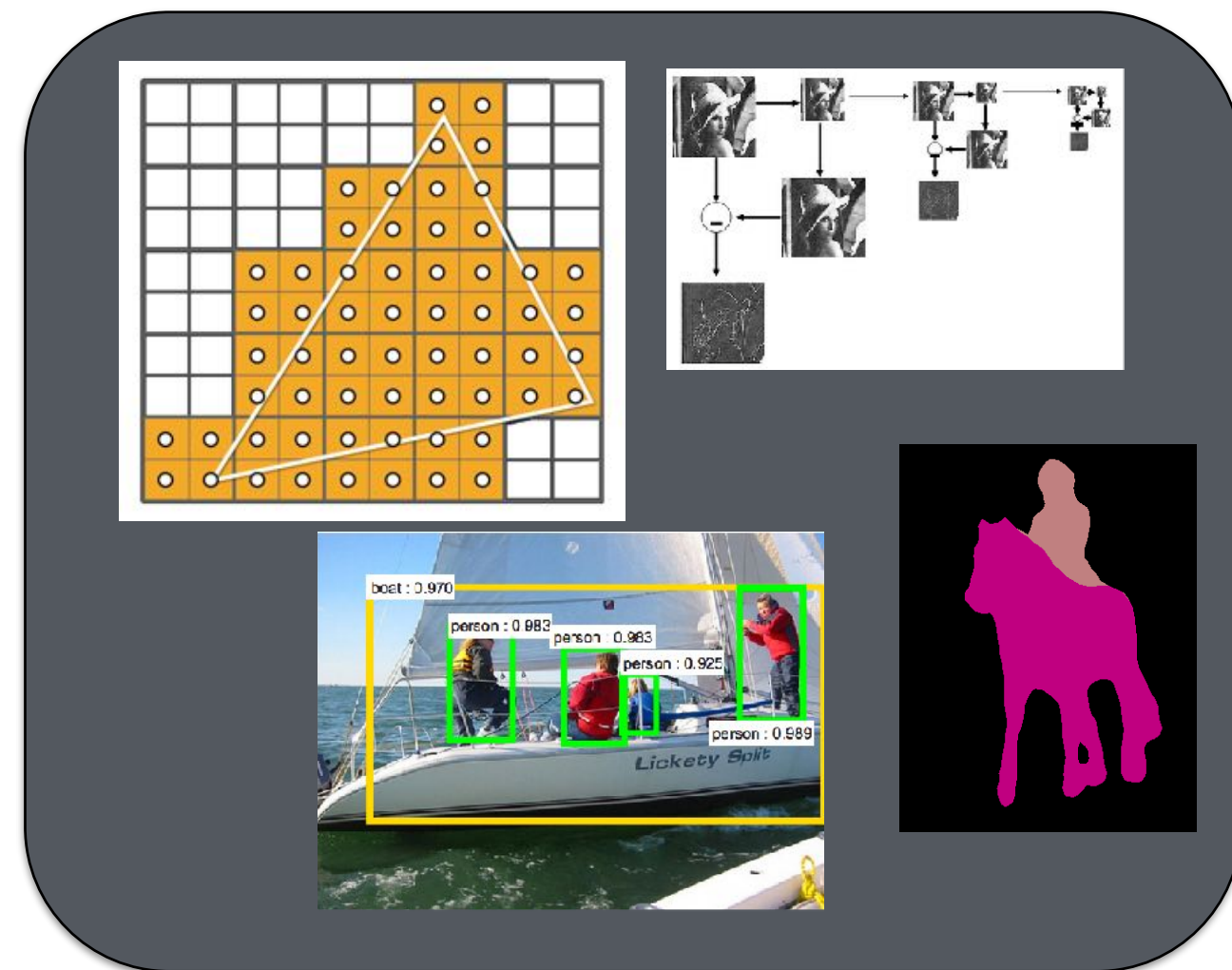
Graphics/vision/digital photography algorithms?

Programming systems?

What we will be learning about

Visual Computing Workloads

Algorithms for image/video processing,
DNN evaluation, data compression, etc.



**If you don't understand key workload characteristics,
how can you design a "good" system?**

What we will be learning about

Modern Hardware Organization

High-throughput hardware designs
(parallel, heterogeneous, and specialized)
fundamental constraints like area and power

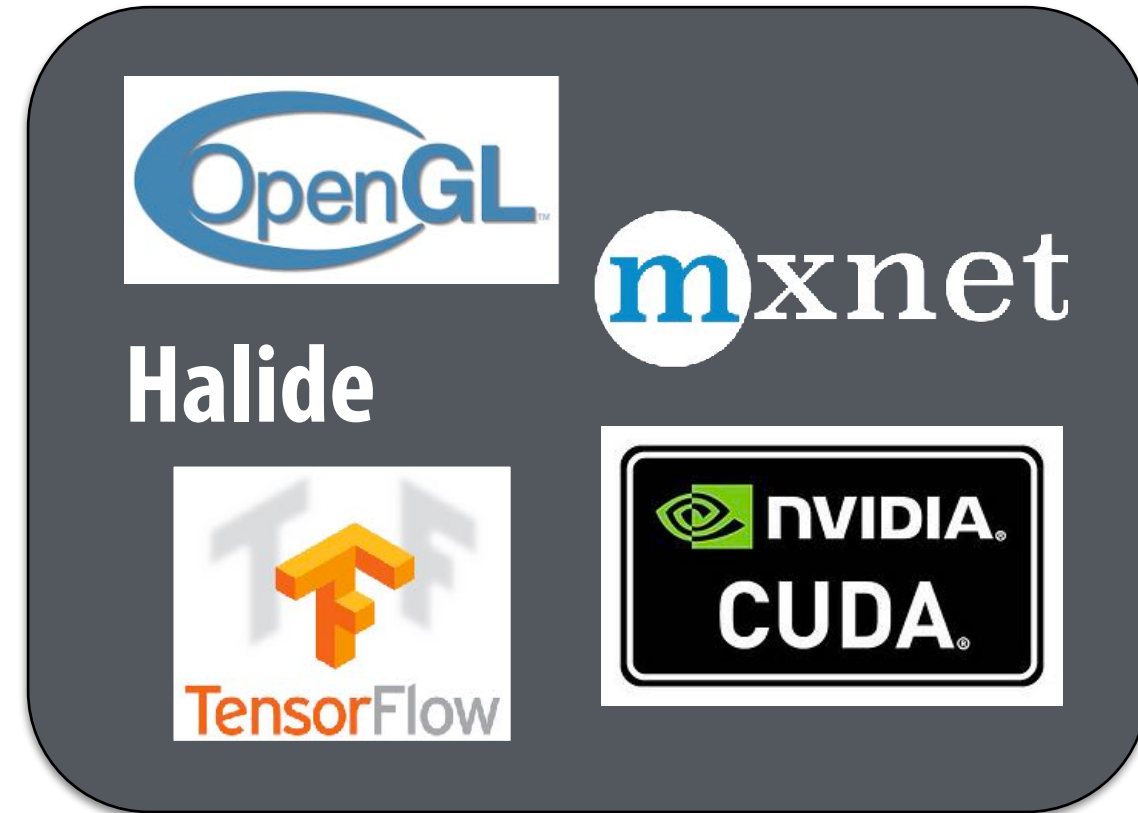


If you don't understand key constraints of modern hardware, how can you design algorithms that are well suited to run on it efficiently?

What we will be learning about

Programming Model Design

Choice of programming abstractions,
level of abstraction issues,
domain-specific vs. general purpose, etc.



Good programming abstractions enable productive development of applications, while also providing system implementors flexibility to explore highly efficient implementations

This course is about architecting efficient, scalable systems...

It is about the process of understanding the **fundamental structure of problems in the visual computing domain, and then leveraging that understanding to...**

To design more efficient and more robust algorithms

To build the most efficient hardware to run these algorithms

To design programming systems to make developing new applications simpler, more productive, and highly performant

2023 course topics

The digital camera photo processing pipeline in modern smartphones

Basic algorithms (the workload)

Programming abstractions for writing image processing apps

Mapping these algorithms to parallel hardware

Systems for creating fast and accurate deep learning models

Scheduling DNN inference efficiently onto modern GPUs (focus on convolutional models, sequence models)

Hardware for accelerating deep learning (why GPUs are not efficient enough!)

System support for the end-to-end ML process, including data labeling and validation

Processing and transmitting video

Trends in video compression (neural techniques)

How modern video conferencing systems work, and what new experiences are on the horizon

Recent neural algorithms and their implications to system design

NeRF

Generative AI for images/videos/animations

Advances in real-time (hardware accelerated) ray tracing

Recent API and hardware support for real-time ray tracing

How deep learning, combined with RT hardware, is making real time ray tracing possible

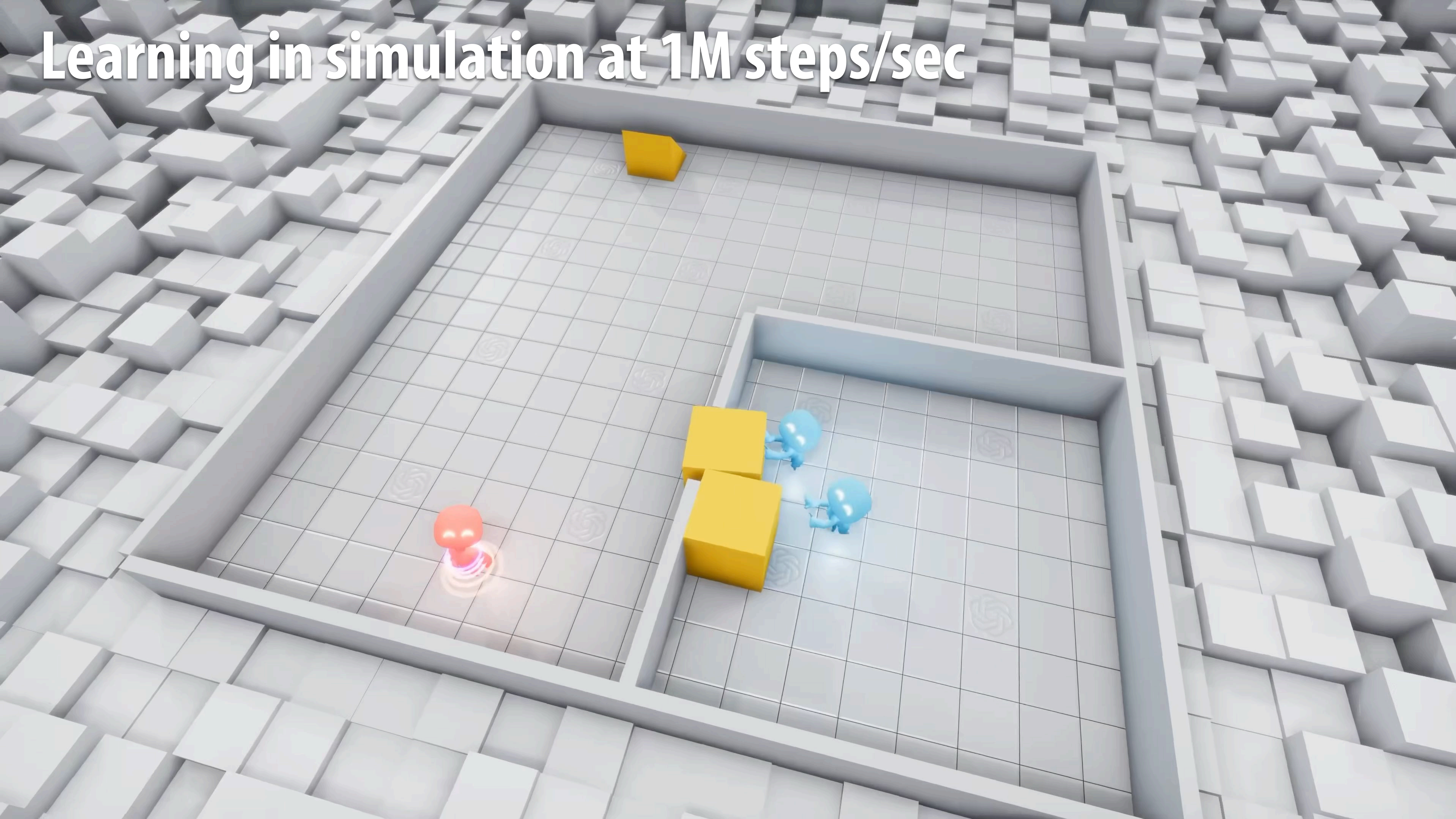
**Plus a late quarter
prompt-engineering
class hackathon!!!**

Activity: student intros / project brainstorming

Kayvon's (virtual) office



Learning in simulation at 1M steps/sec



An interactive AI-based image generation interface



a stylish computer graphics professor named Kayvon wearing a black hoodie in front of a Stanford classroom





Clear style  Photo x

Refresh

Aspect ratio

Square (1:1) ▼

Content type

 None  Photo  Graphic  Art

▼ Styles

All Popular Movements Themes
Techniques Effects Materials Concepts

Popular



Logistics and Expectations

Logistics

- **Course web site:**

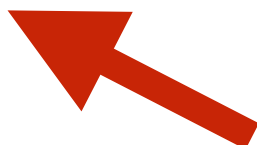
- **<http://cs348k.stanford.edu>**
- **My goal is to post lecture slides the night before class**

- **All announcements will go out via Ed Discussion**

My expectations of you

■ 50% participation

- There will be ~1 assigned technical paper reading per class
- You will submit a response to each reading by midnight prior to class days
- We will start most classes with a 30-45 minute discussion of the reading
- You can skip 2 readings to get full credit.



This is important. You've got to do the readings and come to class to make the course tick.

■ 50% self-selected term project

- I suggest you start thinking about projects now

Review (or crash course):

**key principles of modern
throughput computing hardware**

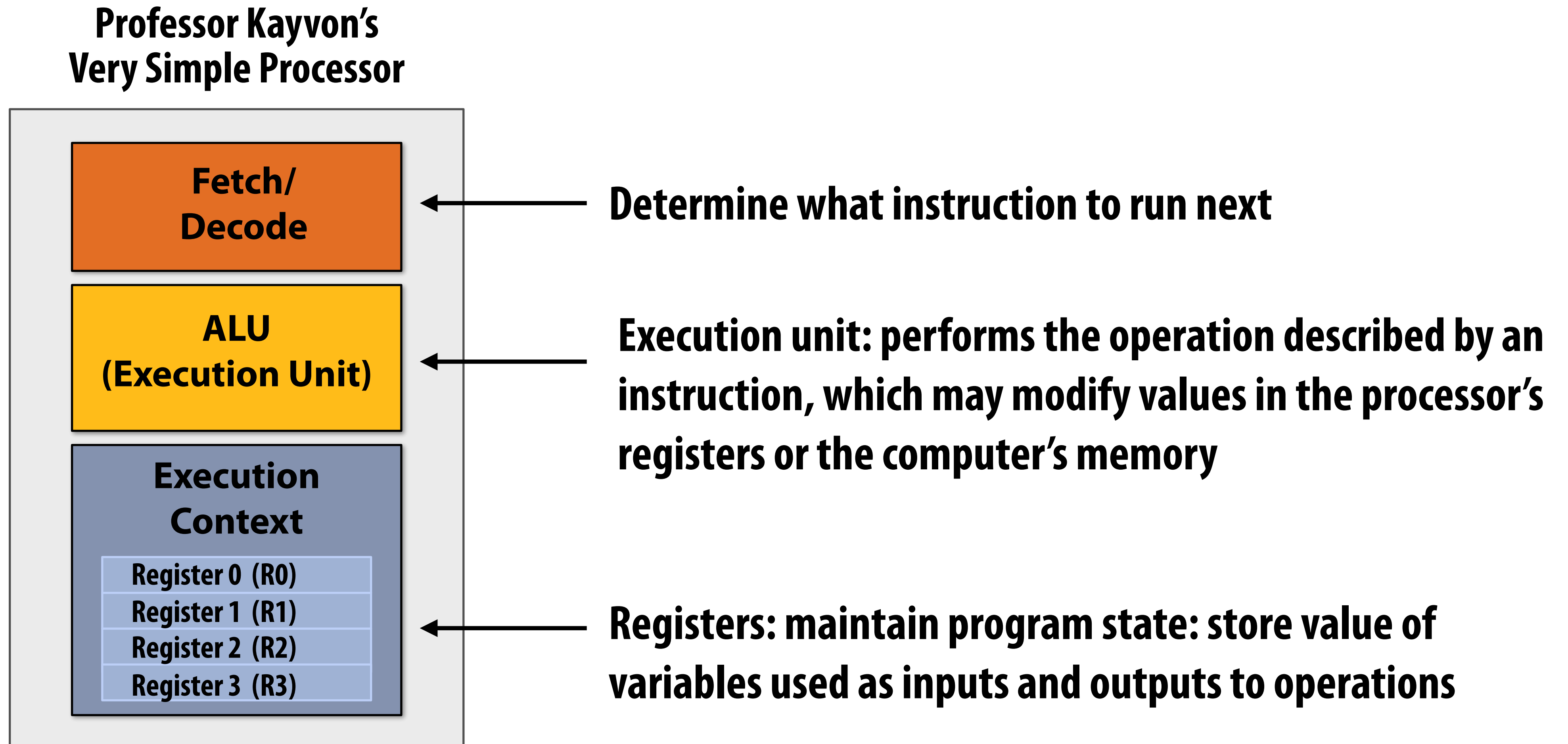
Concept #1:

The high cost of data communication

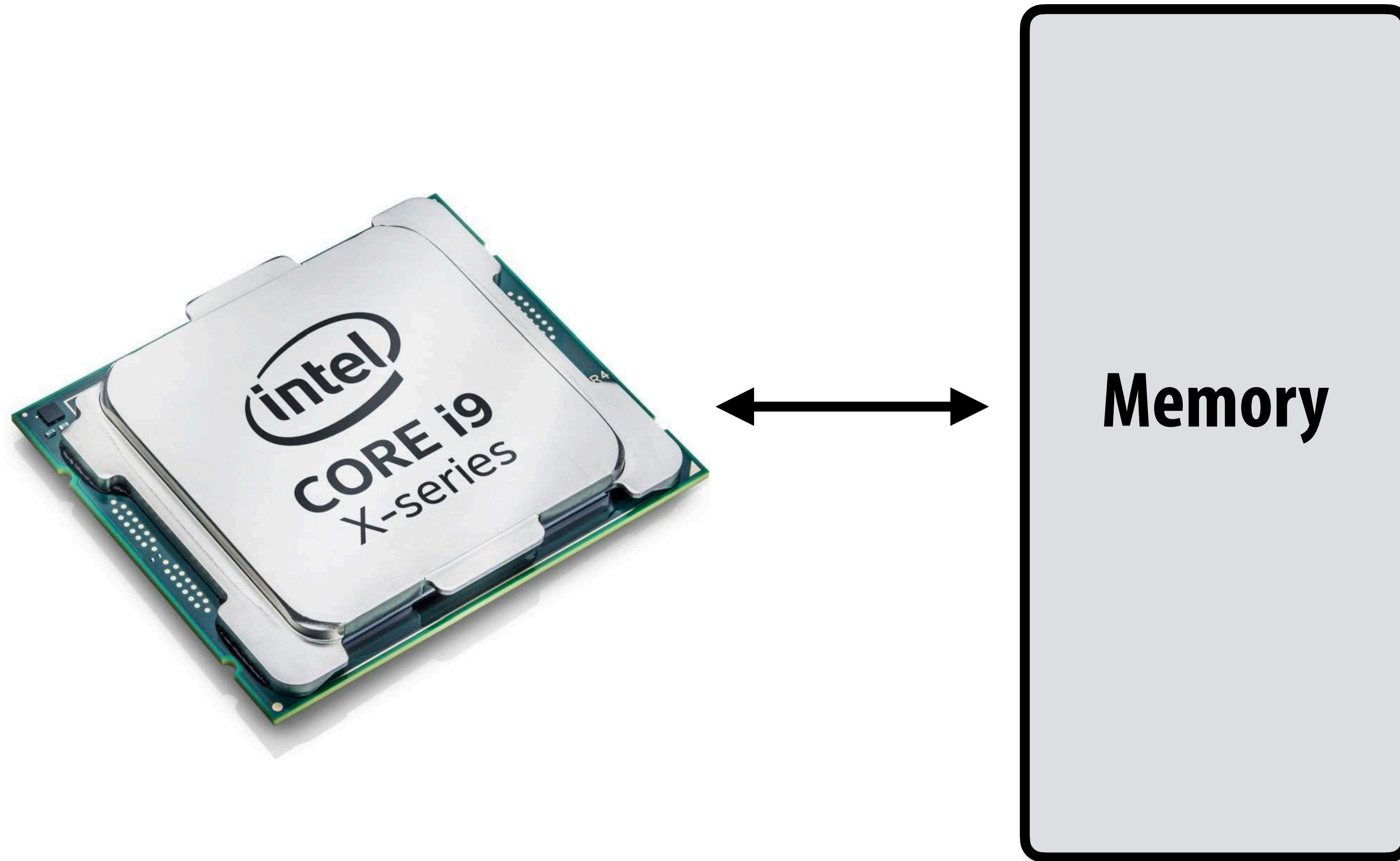
(Almost everything we talk about in this course starts from this concept)

A basic CPU that executes instructions

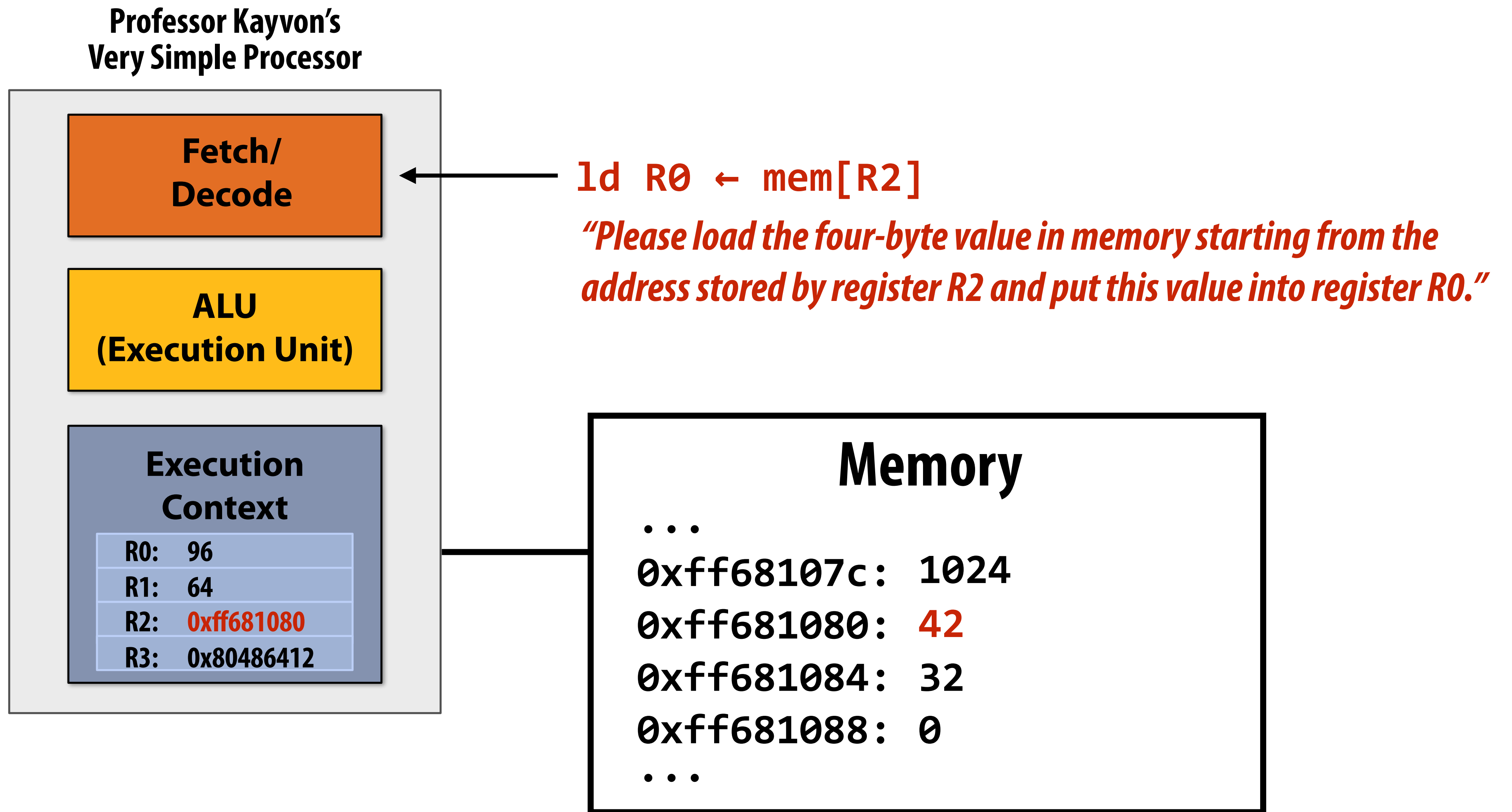
A processor executes instructions



But what is memory?



Load: an instruction for accessing the contents of memory



Terminology

■ Memory access latency

- The amount of time it takes the memory system to provide data to the processor
- Example: 100 clock cycles, 100 nsec



Data request

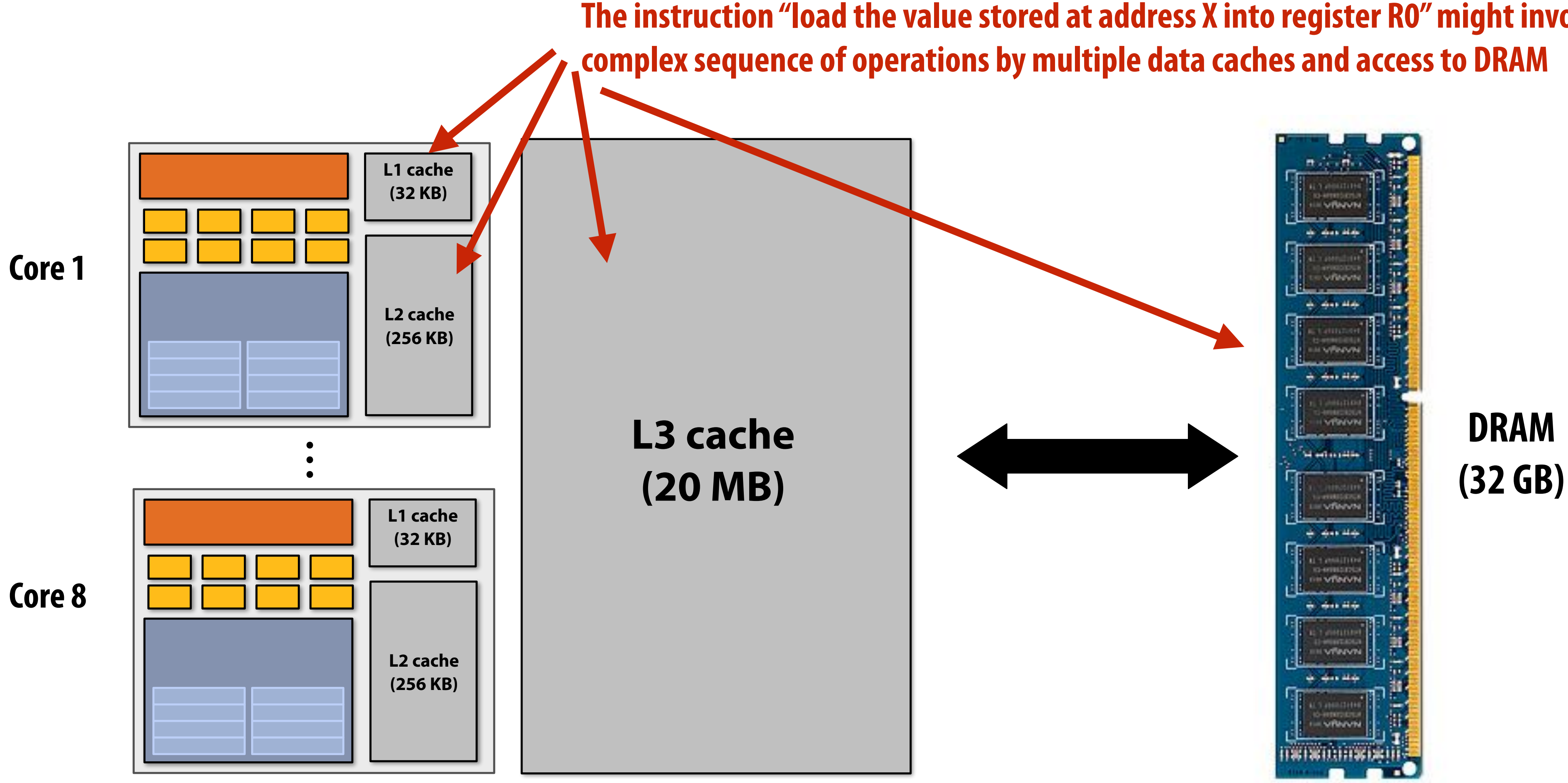


Latency ~ 2 sec

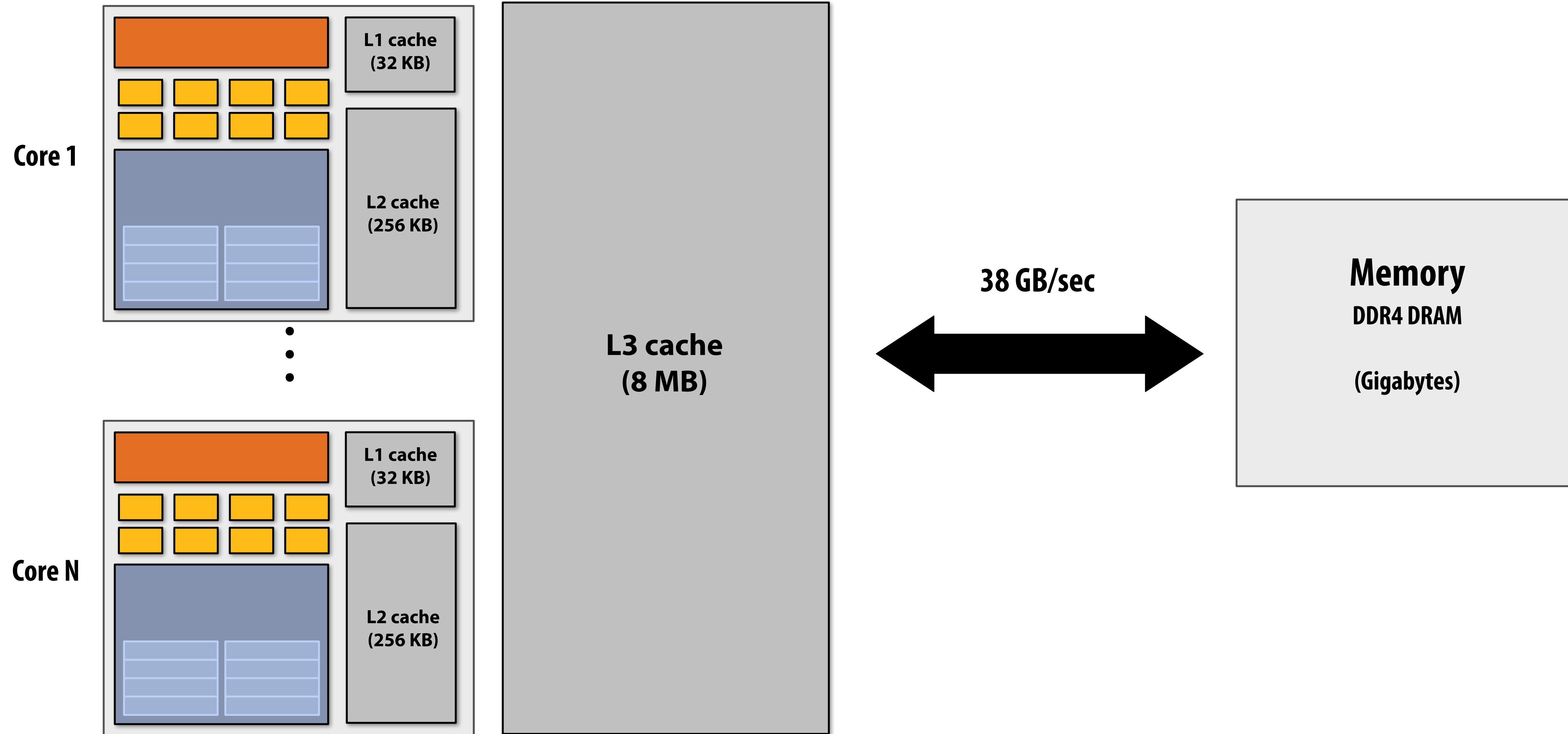
Memory

The implementation of the linear memory address space abstraction on a modern computer is complex

The instruction "load the value stored at address X into register R0" might involve a complex sequence of operations by multiple data caches and access to DRAM



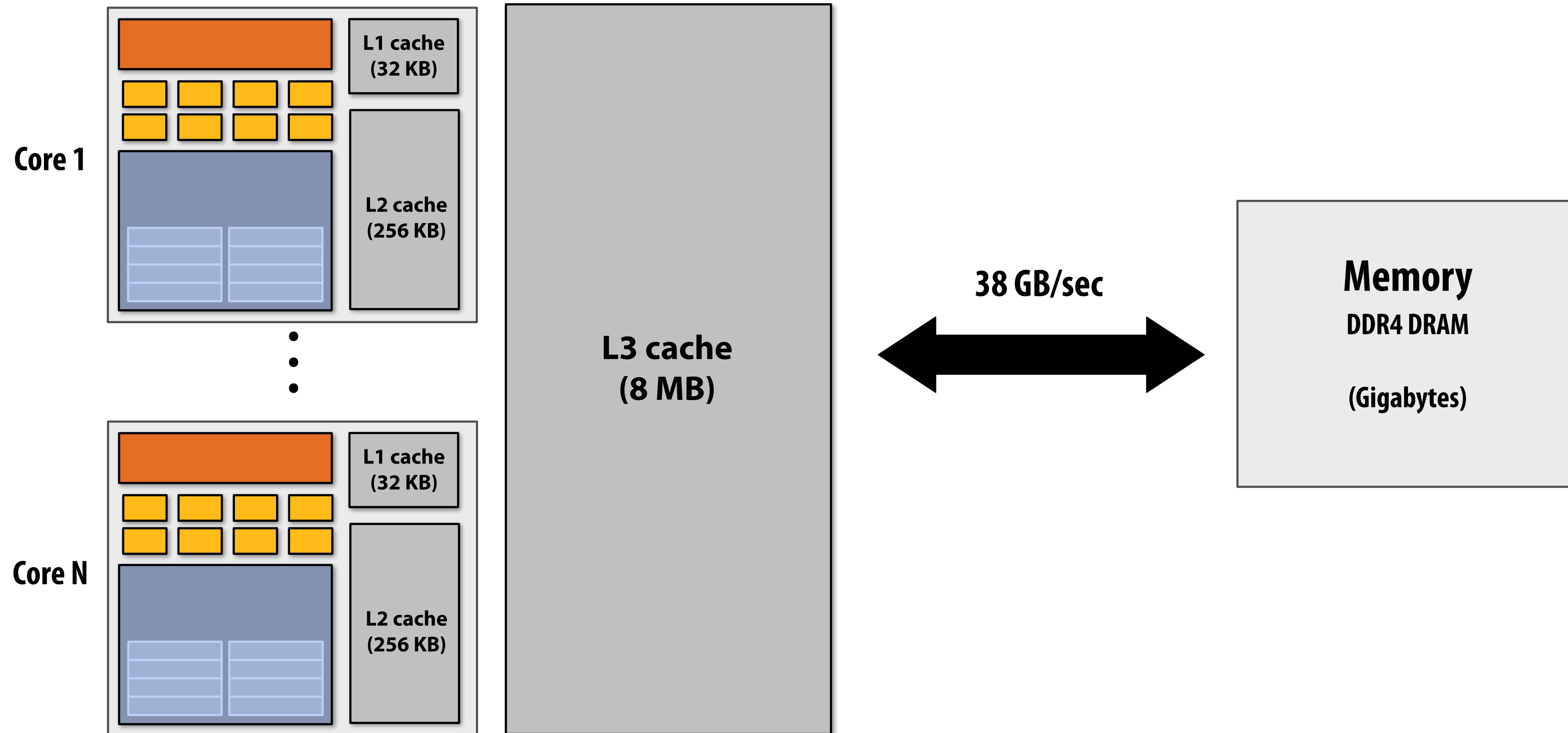
Why do modern processors have data caches?



Caches reduce length of stalls (reduce memory access latency)

Processors run efficiently when data is resident in caches

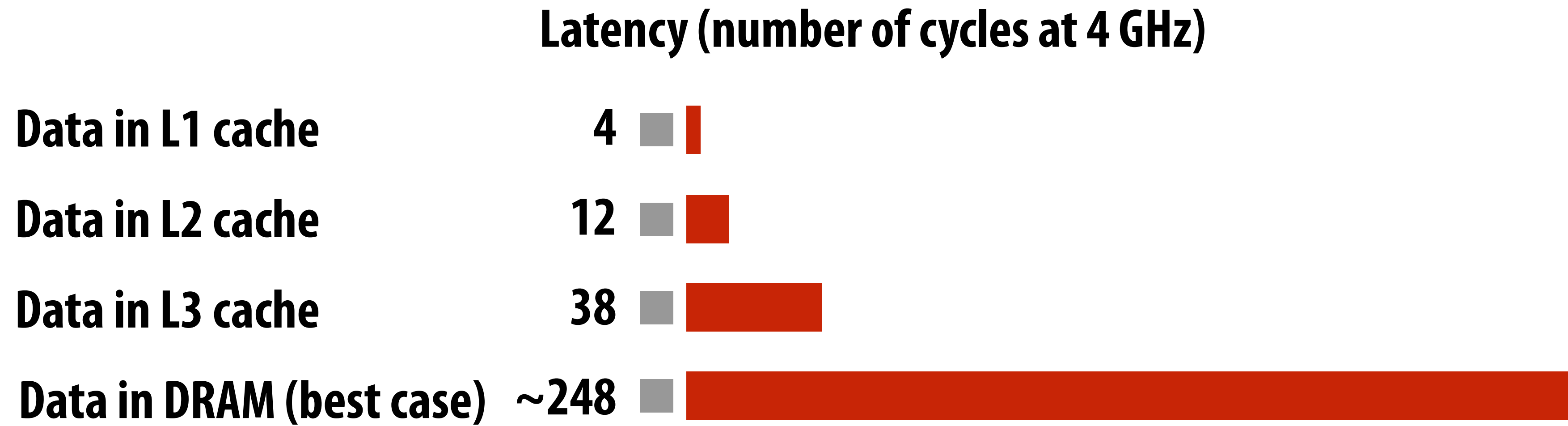
Caches reduce memory access latency *



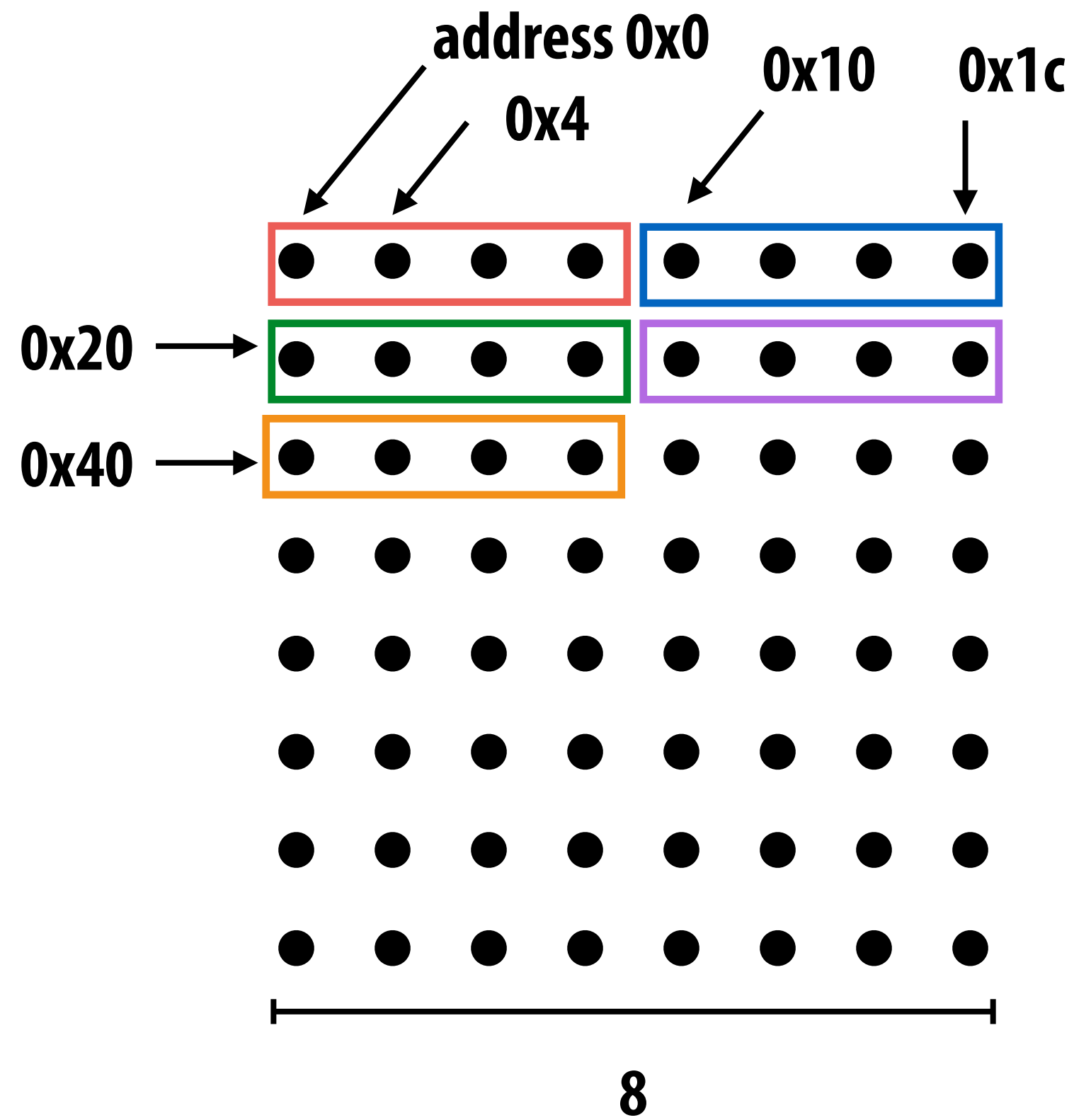
* Caches also provide high bandwidth data transfer to CPU

Data access times

(Kaby Lake CPU)



Cache review

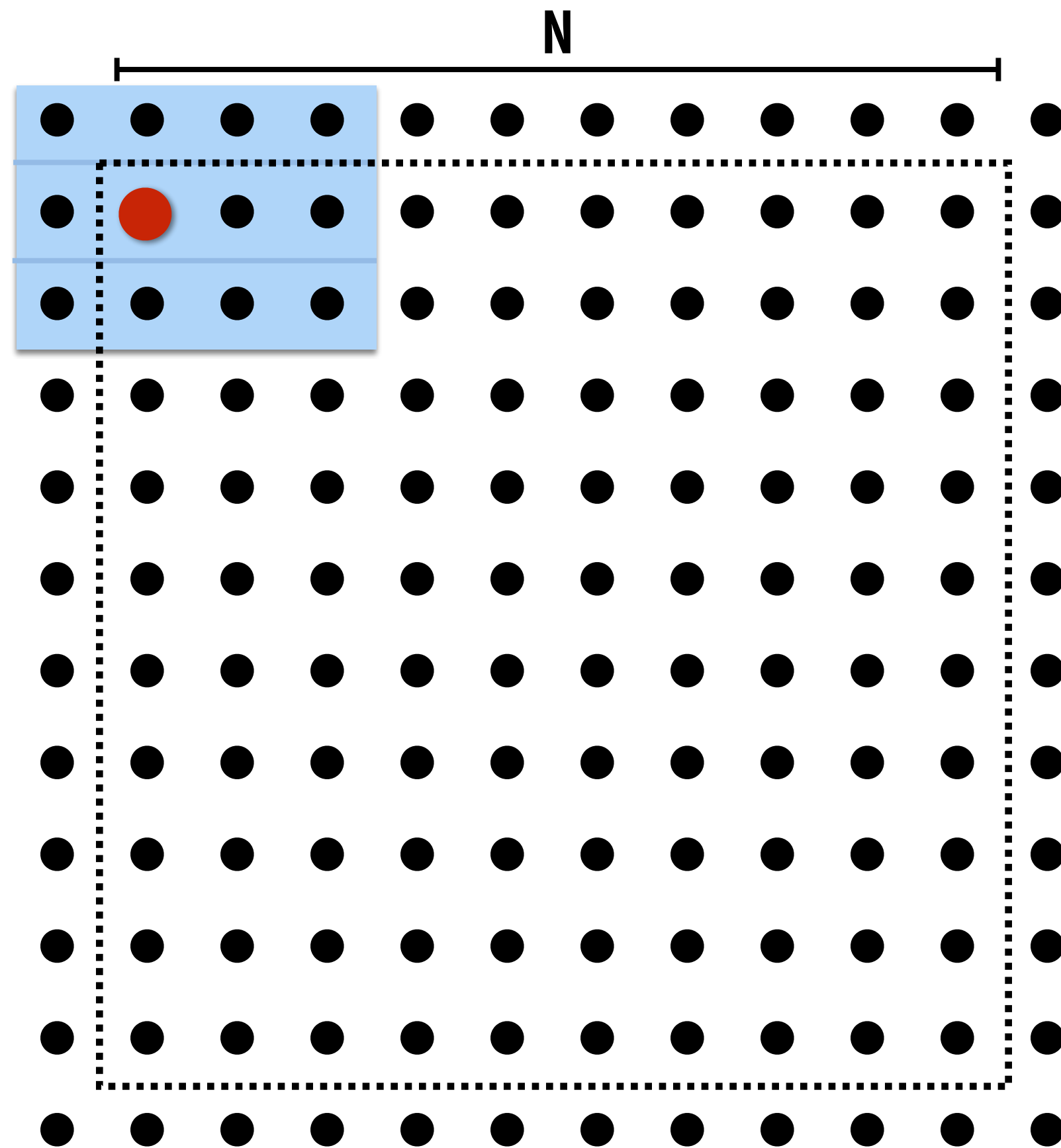


Consider 4-byte elements
 Consider a cache with 16-byte cache lines and a total capacity of 32 bytes (2 lines fit in cache)
 Least recently used (LRU) replacement policy

Address accessed	Cache state (after load is complete)		
0x0	0x0 ●●●●		"cold miss"
0x4	0x0 ●●●●		hit
0x8	0x0 ●●●●		hit
0xc	0x0 ●●●●		hit
0x10	0x0 ●●●●	0x10 ●●●●	cold miss
0x14	0x0 ●●●●	0x10 ●●●●	hit
0x18	0x0 ●●●●	0x10 ●●●●	hit
0x1c	0x0 ●●●●	0x10 ●●●●	hit
0x20	0x20 ●●●●	0x10 ●●●●	cold miss (evict 0x0)
0x24	0x20 ●●●●	0x10 ●●●●	hit
0x28	0x20 ●●●●	0x10 ●●●●	hit
0x2c	0x20 ●●●●	0x10 ●●●●	hit
0x30	0x20 ●●●●	0x30 ●●●●	cold miss (evict 0x10)
0x34	0x20 ●●●●	0x30 ●●●●	hit
0x38	0x20 ●●●●	0x30 ●●●●	hit
0x3c	0x20 ●●●●	0x30 ●●●●	hit
0x40	0x40 ●●●●	0x30 ●●●●	cold miss (evict 0x20)

Data access in grid solver: row-major traversal

“Blocking”: reorder computation to make working sets map well to system’s memory hierarchy



Assume row-major grid layout.

Assume cache line is 4 grid elements.

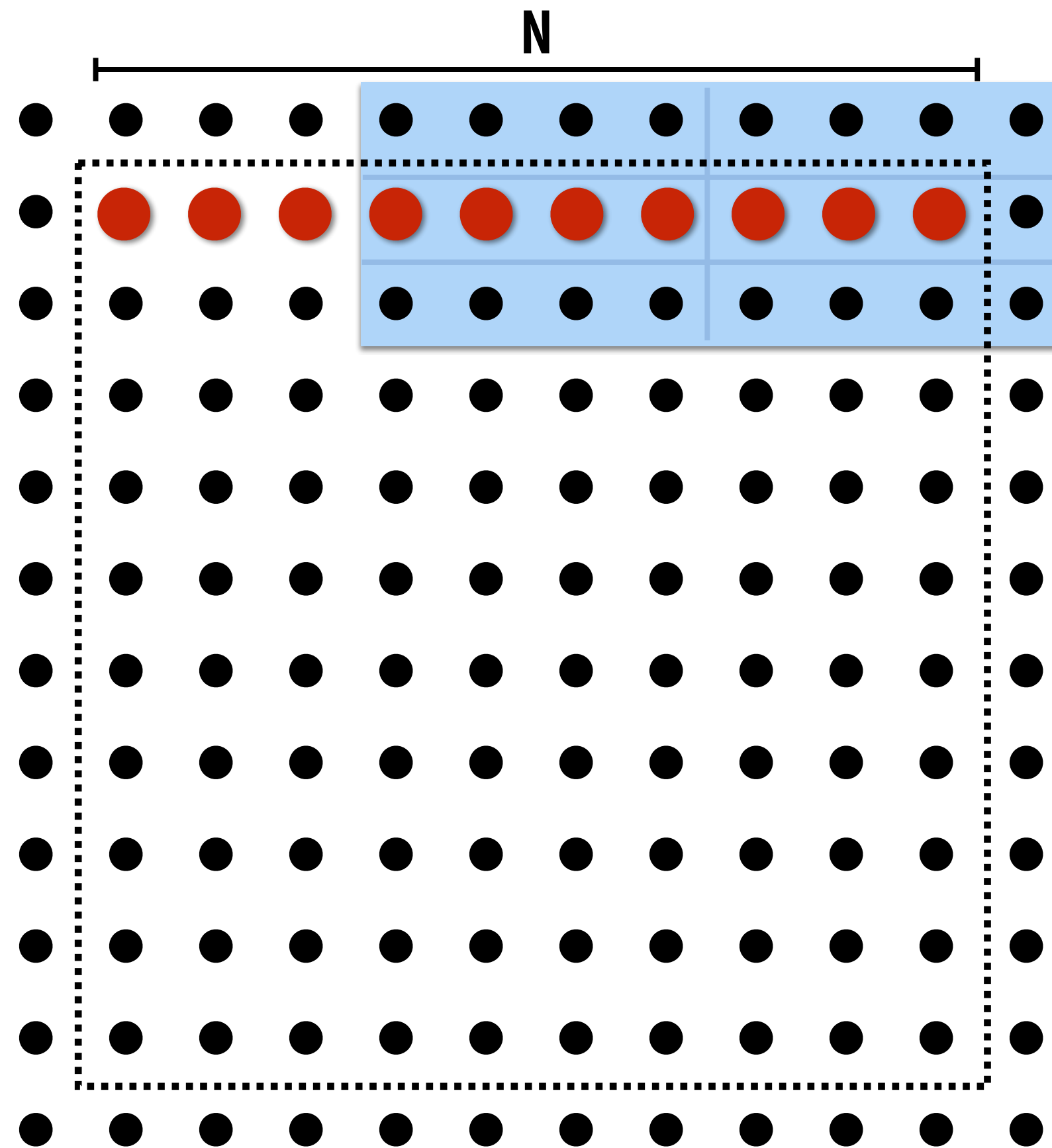
Cache capacity is 24 grid elements (6 lines)

Recall grid solver application.

Blue elements show data that is in cache after update to red element.

Data access in grid solver: row-major traversal

“Blocking”: reorder computation to make working sets map well to system’s memory hierarchy



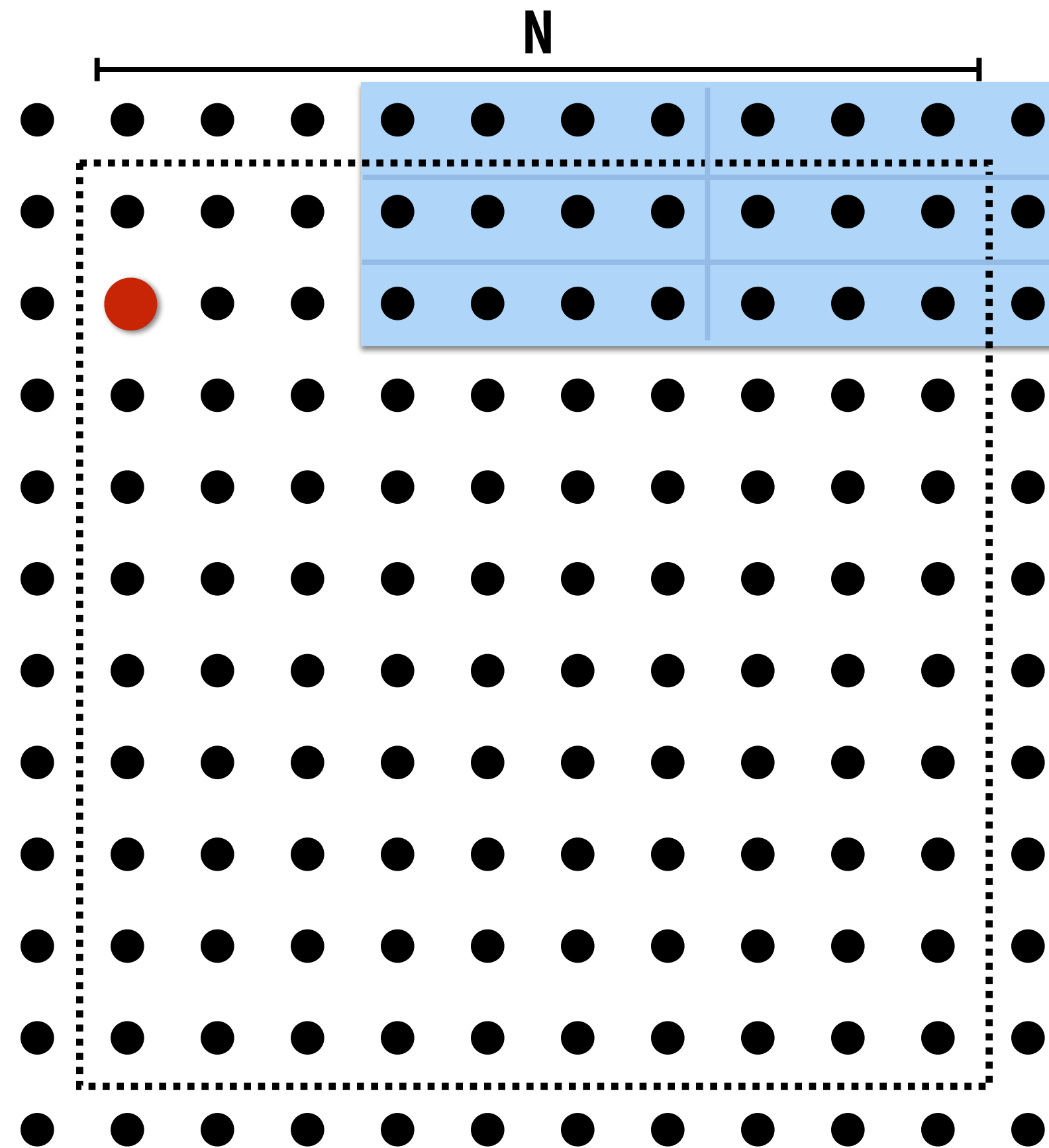
Assume row-major grid layout.

Assume cache line is 4 grid elements.

Cache capacity is 24 grid elements (6 lines)

Blue elements show data in cache at end of processing first row.

Problem with row-major traversal: long time between accesses to same data



Assume row-major grid layout.

Assume cache line is 4 grid elements.

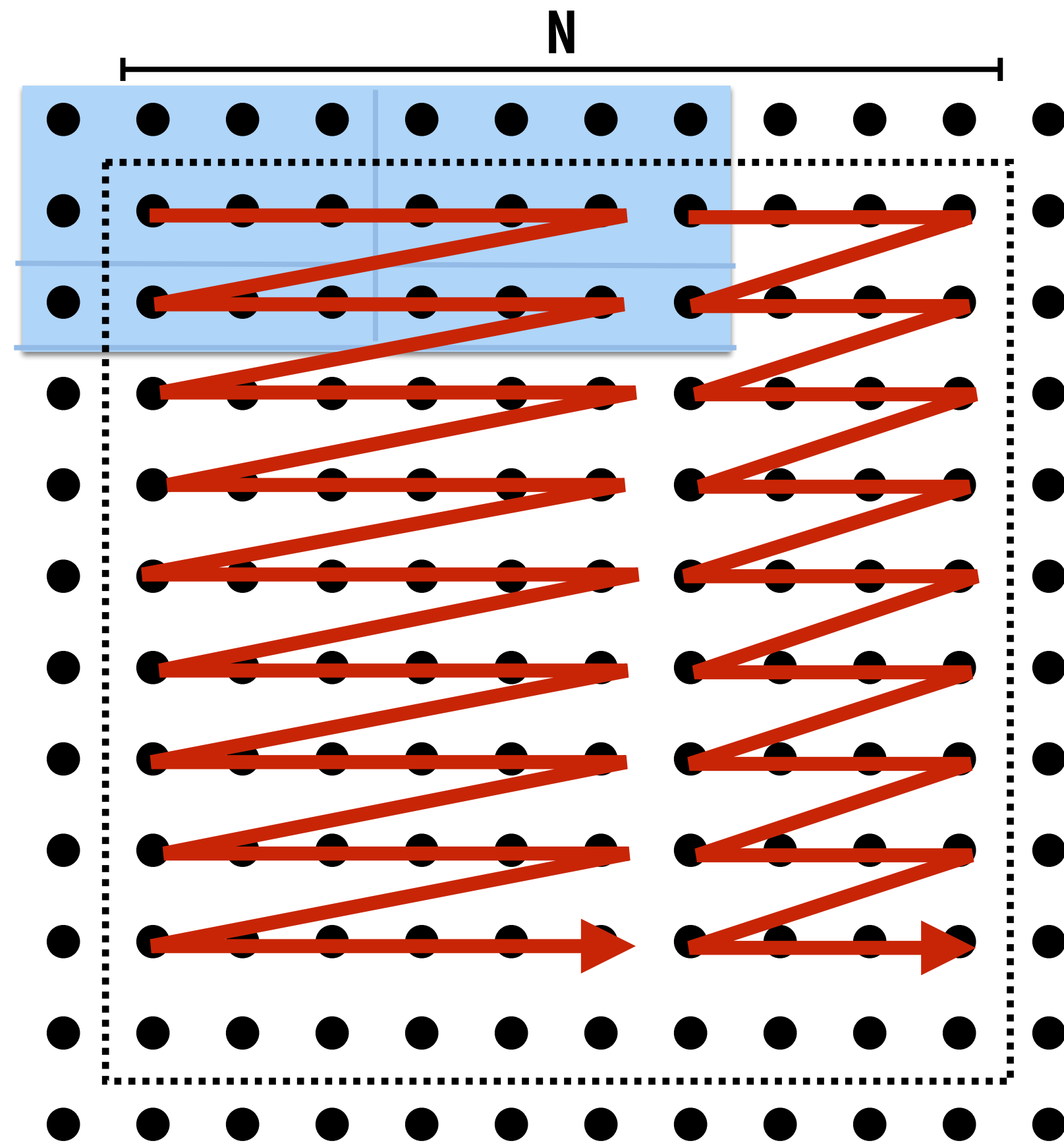
Cache capacity is 24 grid elements (6 lines)

Although elements (0,2) and (0,1) had been accessed previously, they are no longer present in cache at start of processing row 2.

This program loads three lines for every four elements of output.

Improving temporal locality by changing grid traversal order

“Blocking”: reorder computation to make working sets map well to system’s memory hierarchy



Assume row-major grid layout.

Assume cache line is 4 grid elements.

Cache capacity is 24 grid elements (6 lines)

“Blocked” iteration order

(diagram shows state of cache after finishing work from first row of first block)

Now load two cache lines for every six elements of output

Improving temporal locality by “fusing” loops

```
void add(int n, float* A, float* B, float* C) {  
    for (int i=0; i<n; i++)  
        C[i] = A[i] + B[i];  
}
```

Two loads, one store per math op
(arithmetic intensity = 1/3)

```
void mul(int n, float* A, float* B, float* C) {  
    for (int i=0; i<n; i++)  
        C[i] = A[i] * B[i];  
}
```

Two loads, one store per math op
(arithmetic intensity = 1/3)

```
float* A, *B, *C, *D, *E, *tmp1, *tmp2;
```

```
// assume arrays are allocated here
```

```
// compute E = D + ((A + B) * C)
```

```
add(n, A, B, tmp1);
```

```
mul(n, tmp1, C, tmp2);
```

```
add(n, tmp2, D, E);
```

Overall arithmetic intensity = 1/3

```
void fused(int n, float* A, float* B, float* C, float* D, float* E) {  
    for (int i=0; i<n; i++)  
        E[i] = D[i] + (A[i] + B[i]) * C[i];  
}
```

Four loads, one store per 3 math ops
(arithmetic intensity = 3/5)

```
// compute E = D + (A + B) * C
```

```
fused(n, A, B, C, D, E);
```

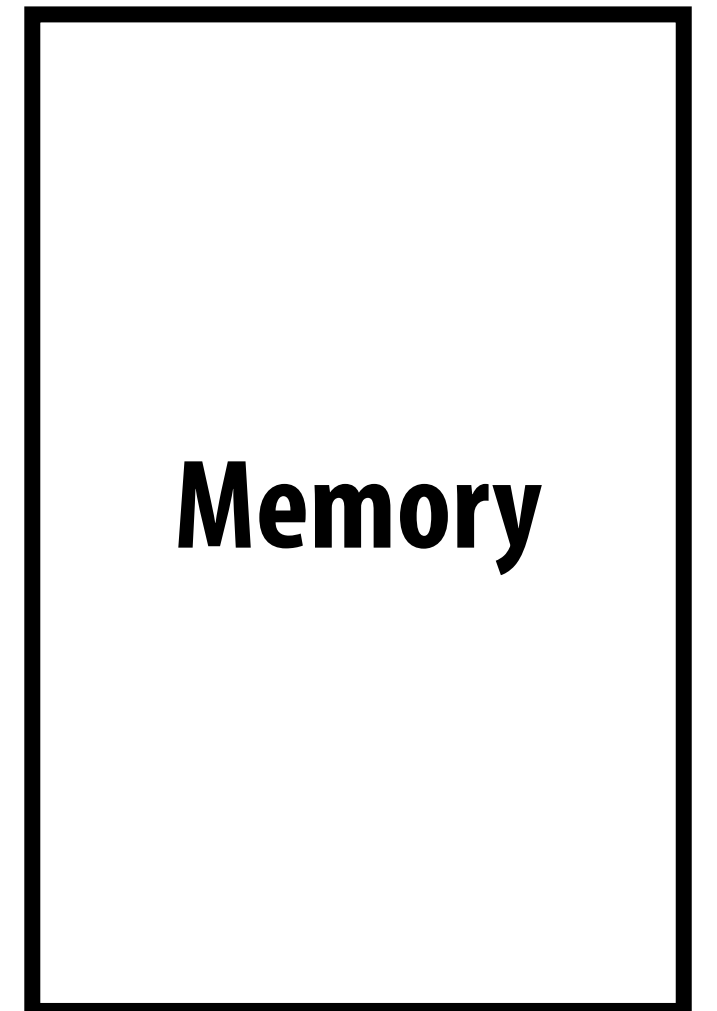
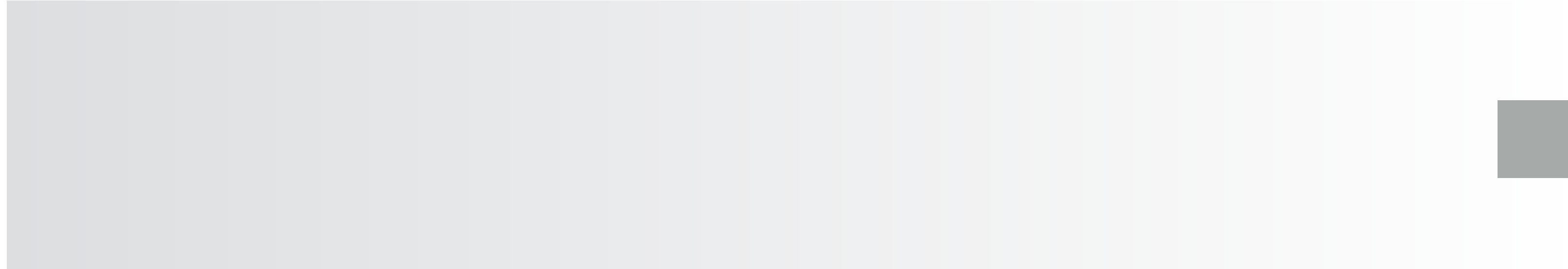
Code on top is more modular (e.g, array-based math library like numPy in Python)

Code on bottom performs much better. Why?

Terminology

■ Memory bandwidth

- The rate at which the memory system can provide data to a processor
- Example: 20 GB/s



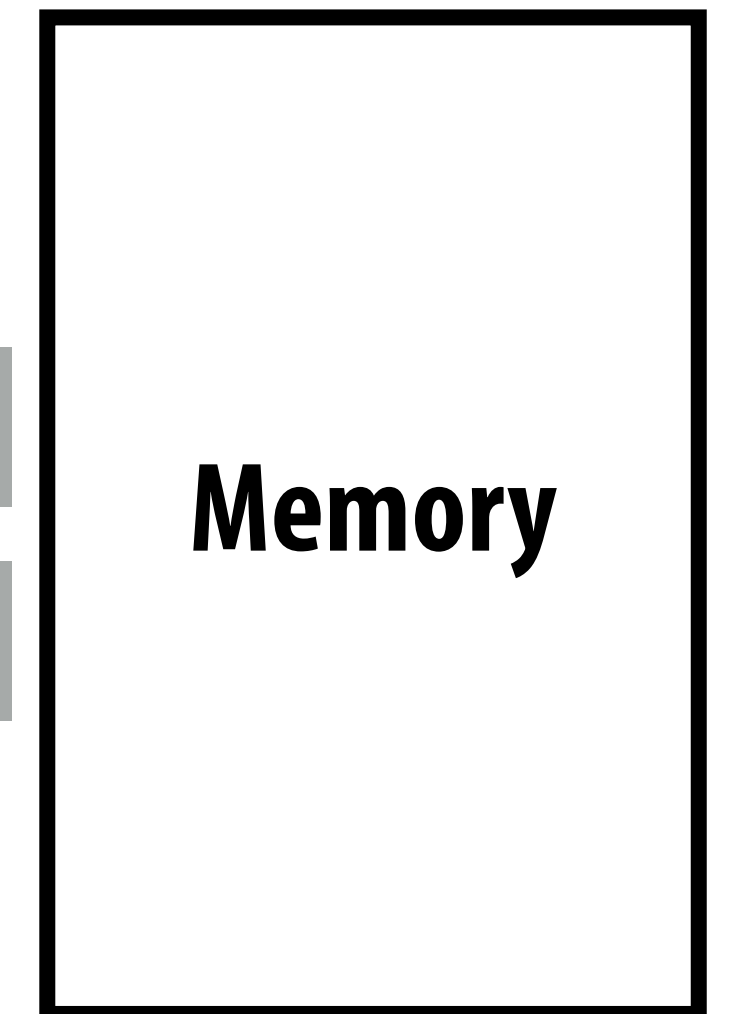
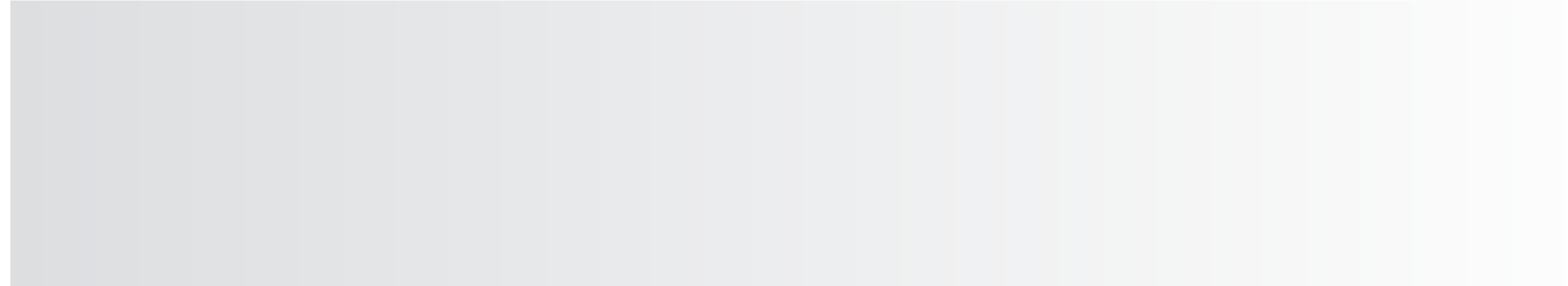
Bandwidth ~ 4 items/sec

Latency of transferring any one item: ~2 sec

Terminology

■ Memory bandwidth

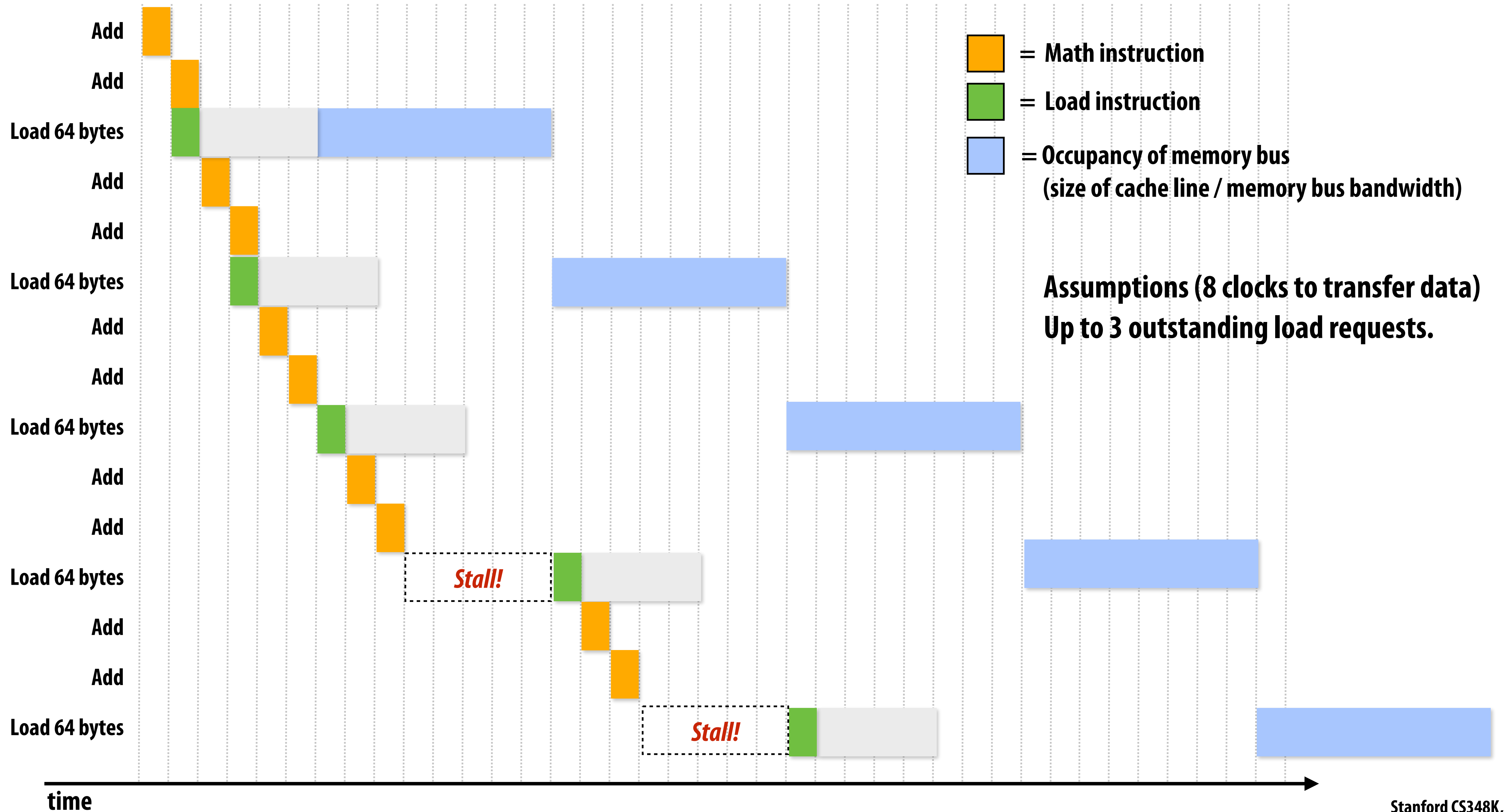
- The rate at which the memory system can provide data to a processor
- Example: 20 GB/s



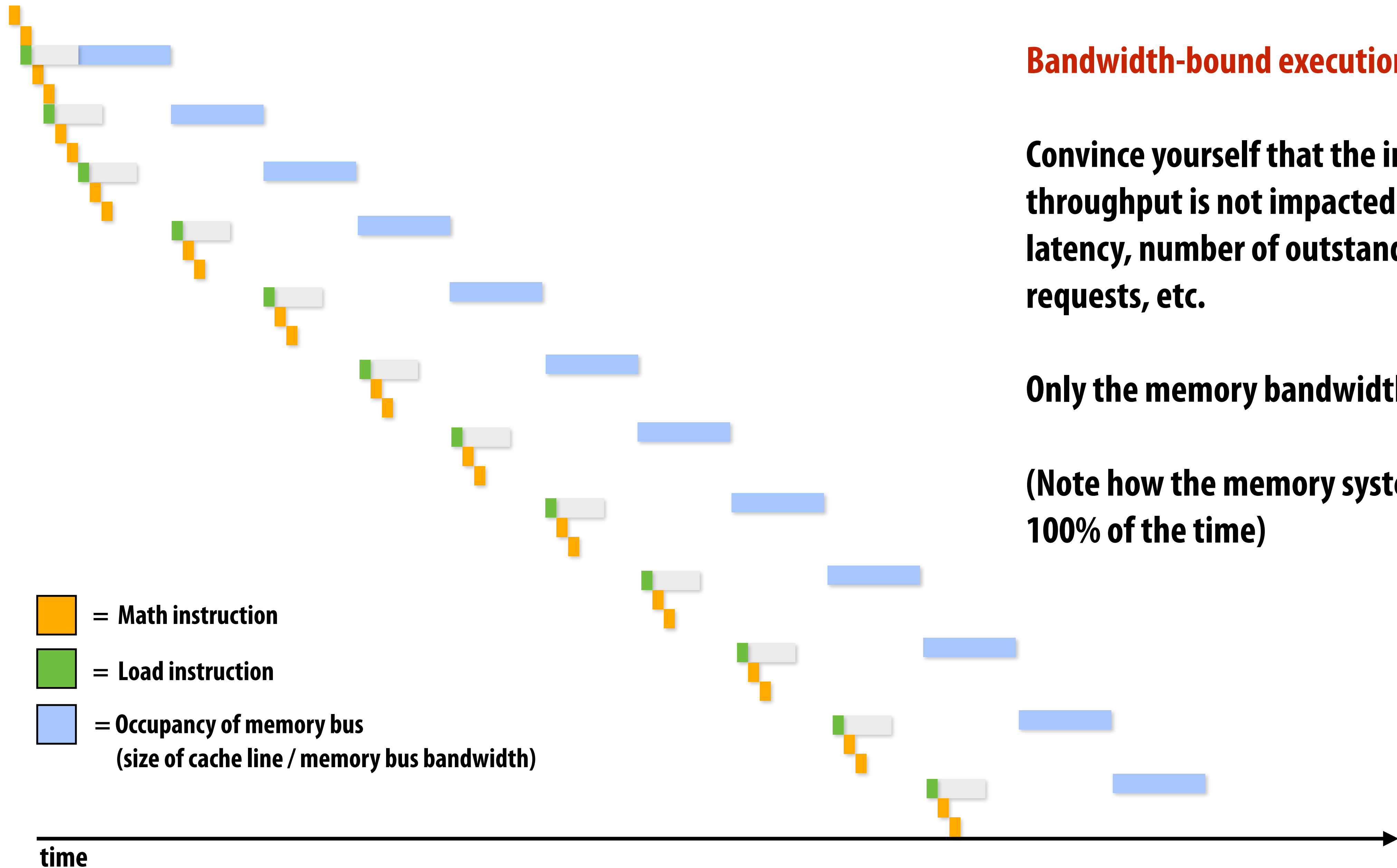
Bandwidth: ~ 8 items/sec

Latency of transferring any one item: ~2 sec

Consider a processor that can do one add per clock (+ can co-issue LD)

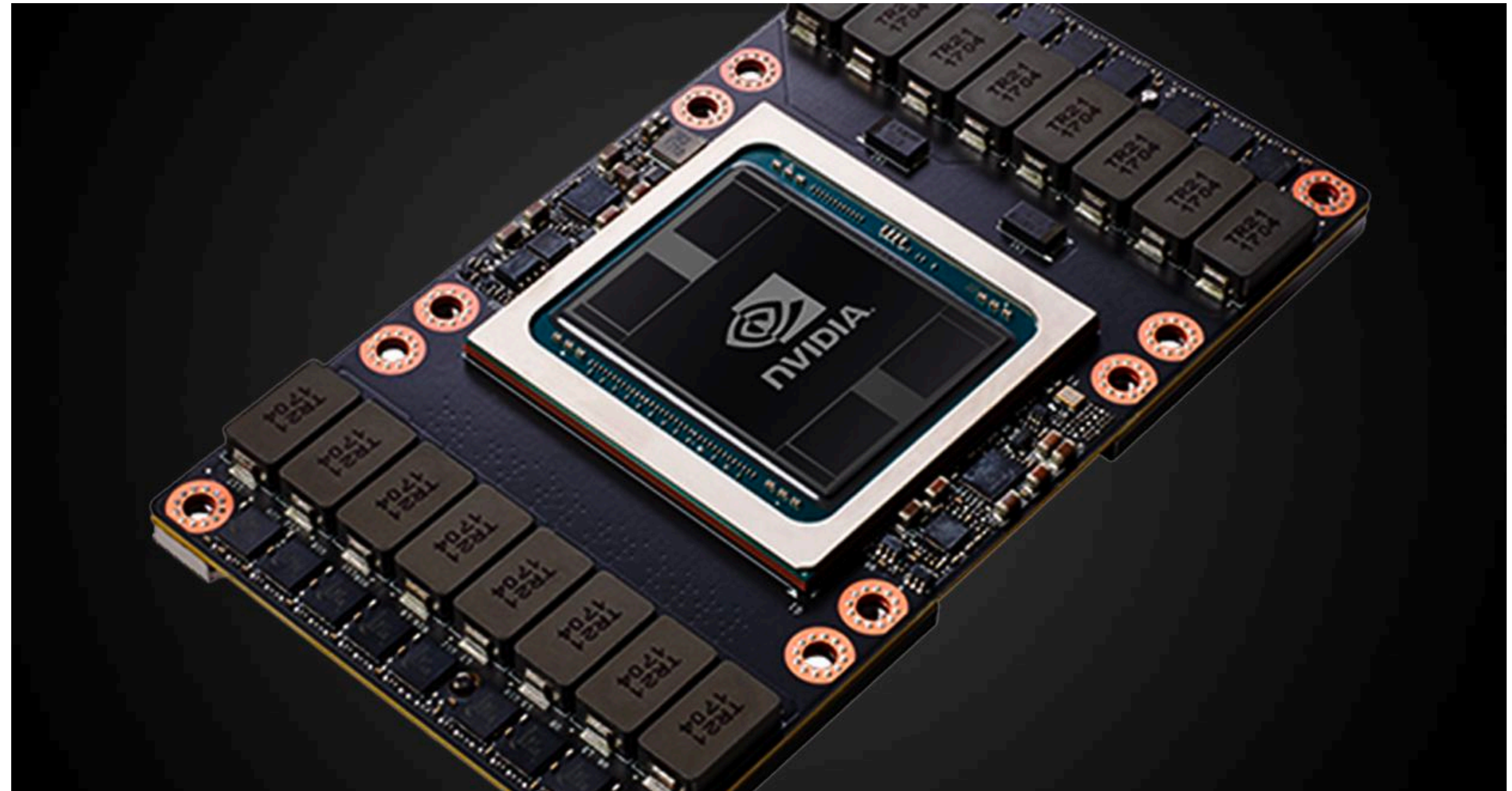


Rate of math instructions limited by available bandwidth



High bandwidth memories

- Modern GPUs leverage high bandwidth memories located near processor
- Example:
 - V100 uses HBM2
 - 900 GB/s



Thought experiment

Task: element-wise multiplication of two vectors A and B

Assume vectors contain millions of elements

- Load input A[i]
- Load input B[i]
- Compute $A[i] \times B[i]$
- Store result into C[i]

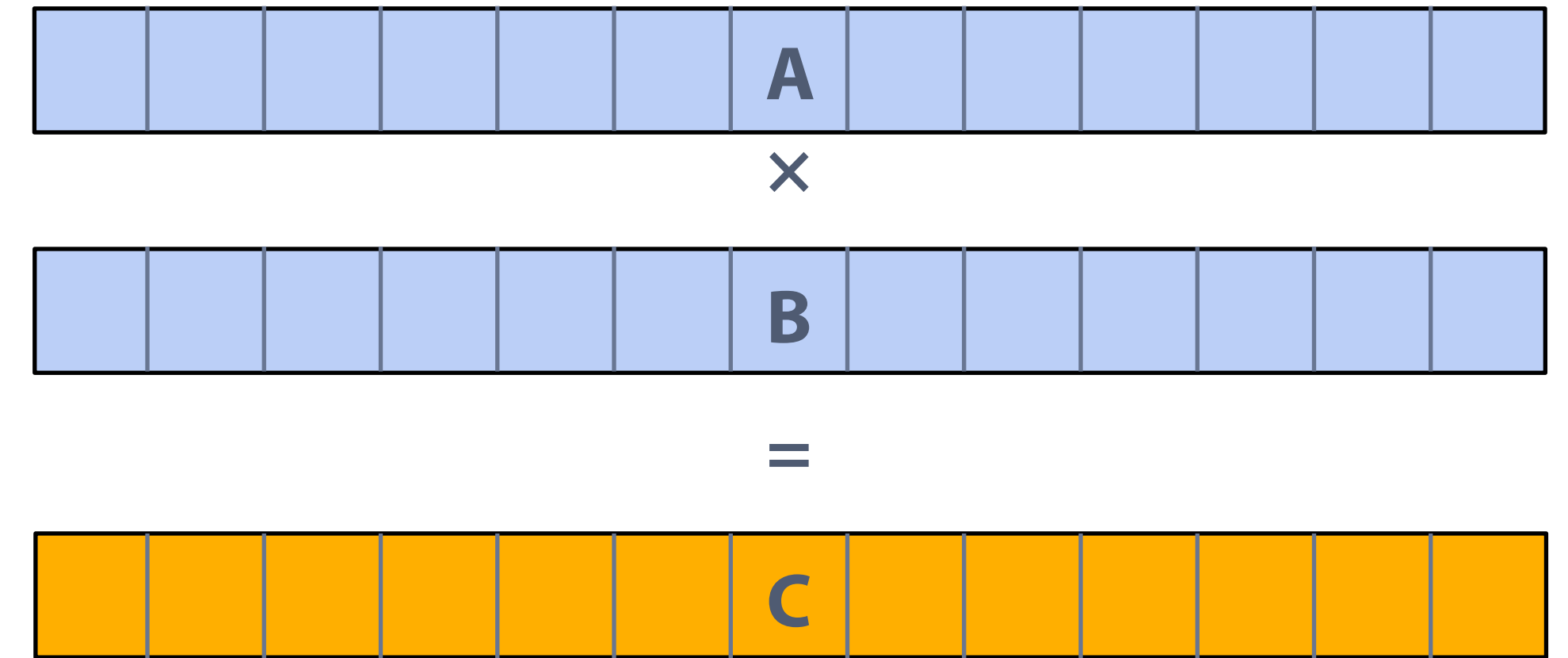
Three memory operations (12 bytes) for every MUL

NVIDIA V100 GPU can do 5120 fp32 MULs per clock (@ 1.6 GHz)

Need ~98 TB/sec of bandwidth to keep functional units busy

<1% GPU efficiency... but still 12x faster than eight-core CPU!

(3.2 GHz Xeon E5v4 eight-core CPU connected to 76 GB/sec memory bus: ~3% efficiency on this computation)



This computation is bandwidth limited!

**If processors request data at too high a rate,
the memory system cannot keep up.**

**Overcoming bandwidth limits is often the most important challenge facing
software developers targeting modern throughput-optimized systems.**

Data movement has high energy cost

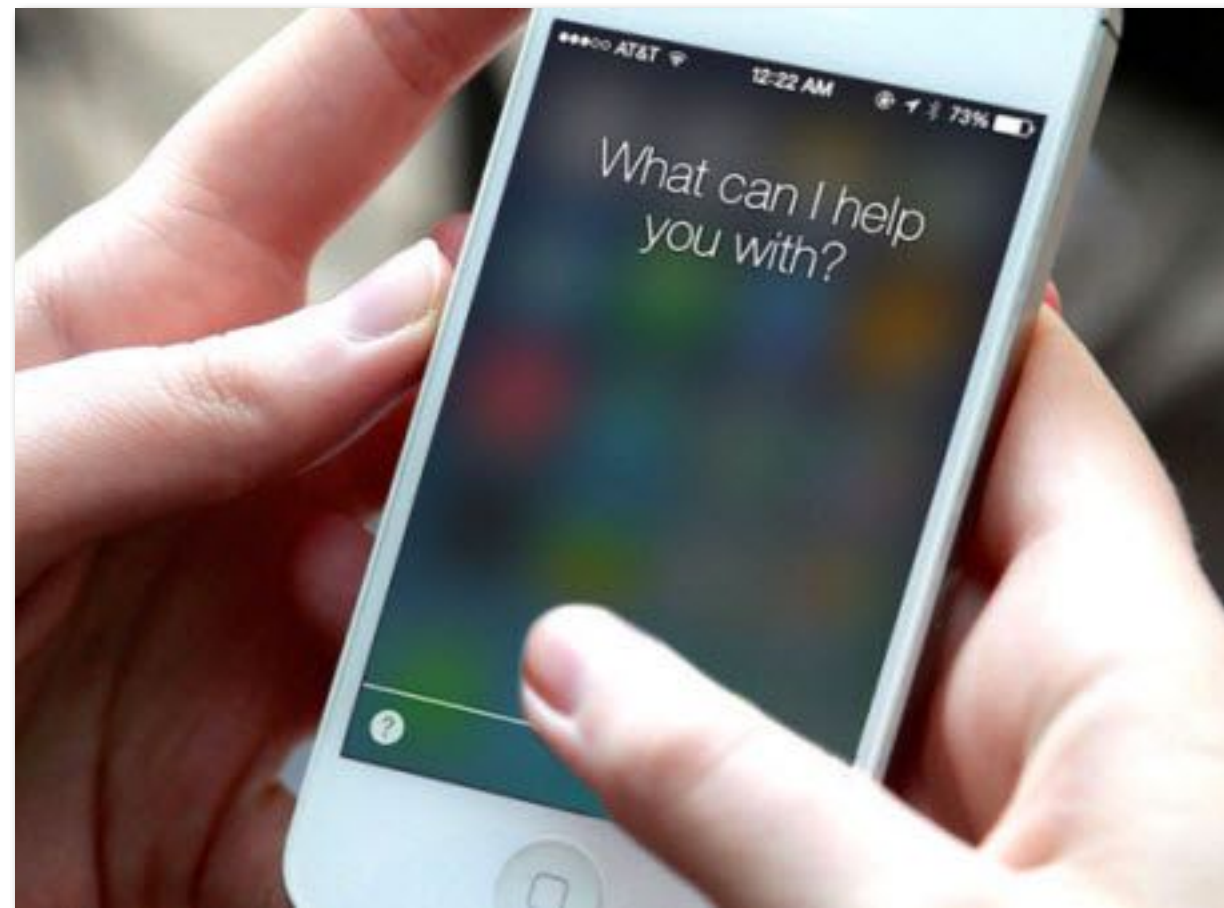
- **Rule of thumb in mobile system design: always seek to reduce amount of data transferred from memory**
 - **Earlier in class we discussed minimizing communication to reduce stalls (poor performance). Now, we wish to reduce communication to reduce energy consumption**
- **“Ballpark” numbers**
 - **Integer op: ~ 1 pJ *** [Sources: Bill Dally (NVIDIA), Tom Olson (ARM)]
 - **Floating point op: ~20 pJ ***
 - **Reading 64 bits from small local SRAM (1mm away on chip): ~ 26 pJ**
 - **Reading 64 bits from low power mobile DRAM (LPDDR): ~1200 pJ** ← **Suggests that recomputing values, rather than storing and reloading them, is a better answer when optimizing code for energy efficiency!**
- **Implications**
 - **Reading 10 GB/sec from memory: ~1.6 watts**
 - **Entire power budget for mobile GPU: ~1 watt**
(remember phone is also running CPU, display, radios, etc.)
 - **iPhone 6 battery: ~7 watt-hours (note: my Macbook Pro laptop: 99 watt-hour battery)**
 - **Exploiting locality matters!!!**

* Cost to just perform the logical operation, not counting overhead of instruction decode, load data from registers, etc.

Concept #2:
The value of specializing computation

Mobile: benefits of increasing efficiency

- **Run faster for a fixed period of time**
 - Run at higher clock, use more cores (reduce latency of critical task)
 - Do more at once
- **Run at a fixed level of performance for longer**
 - e.g., video playback, health apps
 - Achieve “always-on” functionality that was previously impossible



iPhone:
Siri activated by button press or holding phone up to ear



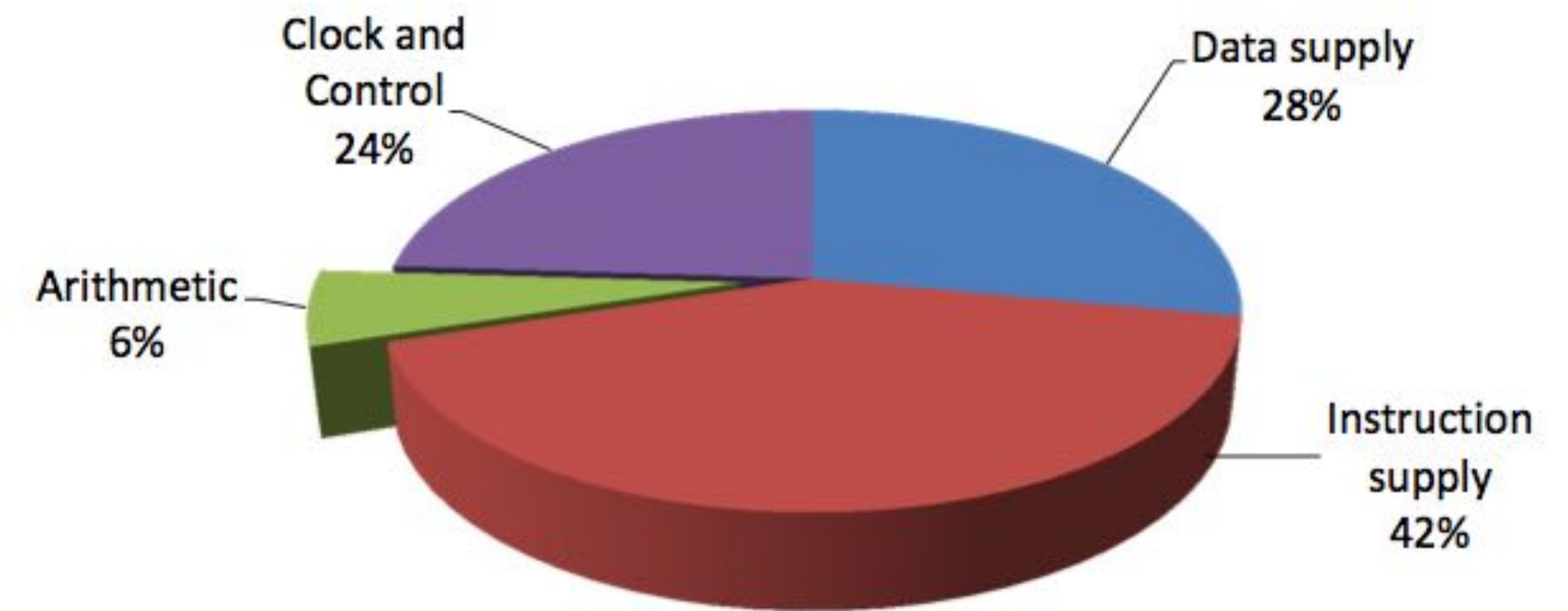
Amazon Echo / Google Home
Always listening



Google Glass: ~40 min
recording per charge
(nowhere near “always on”)

Efficiency benefits of compute specialization

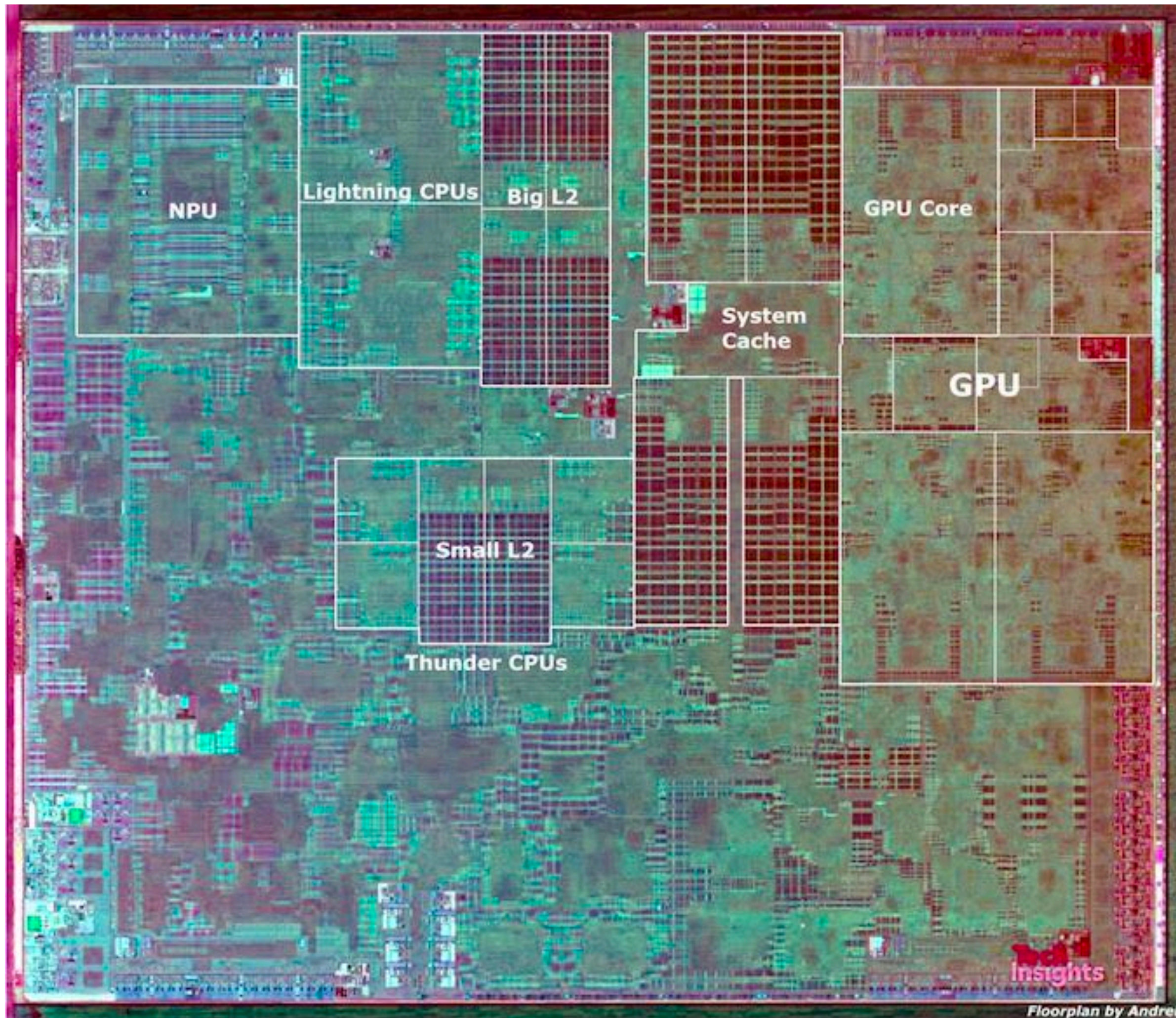
- **Rules of thumb: compared to high-quality C code on CPU...**
- **Throughput-maximized processor architectures: e.g., GPU cores**
 - **Approximately 10x improvement in perf / watt**
 - **Assuming code maps well to wide data-parallel execution and is compute bound**
- **Fixed-function ASIC (“application-specific integrated circuit”)**
 - **Can approach 100-1000x or greater improvement in perf/watt**
 - **Assuming code is compute bound and and is not floating-point math**



Efficient Embedded Computing [Dally et al. 08]

[Figure credit Eric Chung]

Modern smartphones utilize multiple processing units to quickly generate high-quality images



Apple A13 Bionic

Multi-core CPU (heterogeneous cores)

Multi-core GPU

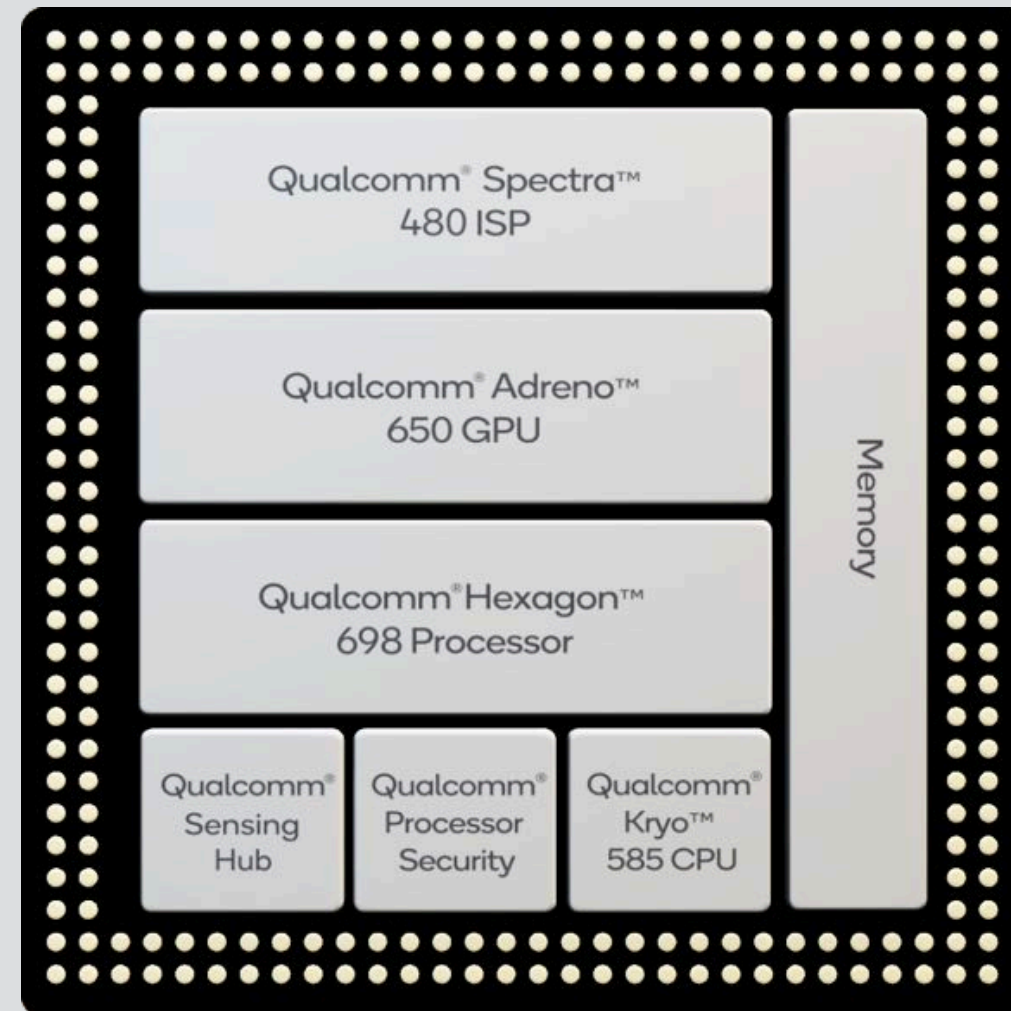
Neural accelerator

Sensor processing accelerator

Video compression/decompression HW

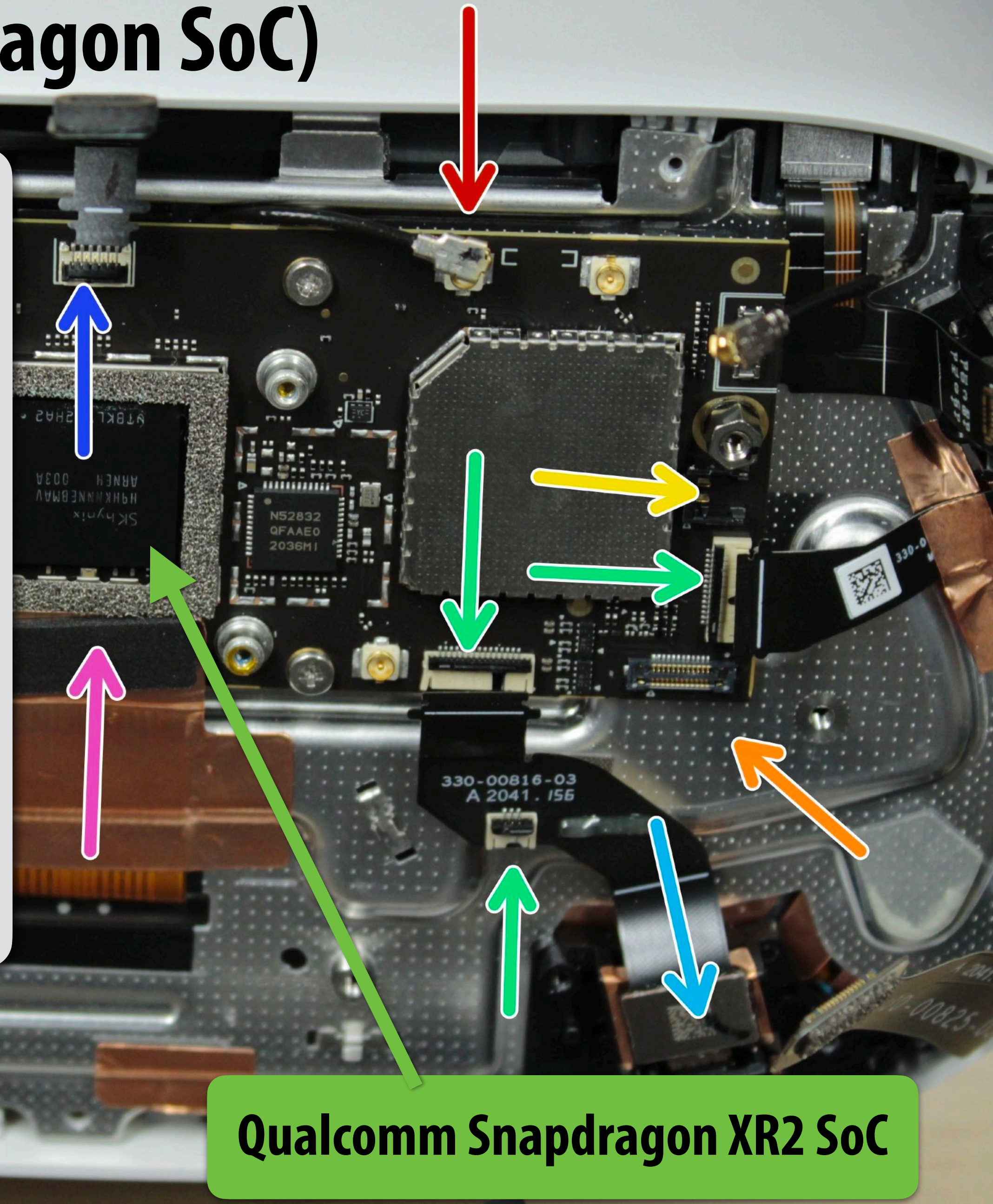
Etc...

Oculus Quest 2 headset (Snapdragon SoC)



This diagram is from Snapdragon 865

- 4 high-performance cores
- 4 low-performance (low energy) cores
- Image processor + DSP
- Multi-core graphics processor (GPU) — up to 3000 x 3000 display @ 90 Hz
- Additional processor for sensors (IMU etc)
- Can process inputs from up to seven simultaneous video camera streams



Qualcomm Snapdragon XR2 SoC

Modern systems use specialized HW for...

- **Image/video encode/decode (e.g., H.264, JPG)**
- **Audio recording/playback**
- **Voice “wake up” (e.g., Ok Google)**
- **Camera “RAW” processing: processing data acquired by image sensor into images that are pleasing to humans**
- **Many 3D graphics tasks (rasterization, texture mapping, occlusion using the Z-buffer)**
- **Continuous sensing (health, fitness, GPS, etc)**
- **Deep network evaluation (Google’s Tensor Processing Unit, Apple Neural engine, etc.)**

Welcome to CS348K!

- See website for tonight's reading