Lecture 1:

Course Introduction + Review of Throughput HW Architecture

Visual Computing Systems
Stanford CS348K, Spring 2026

Hello from the course staff

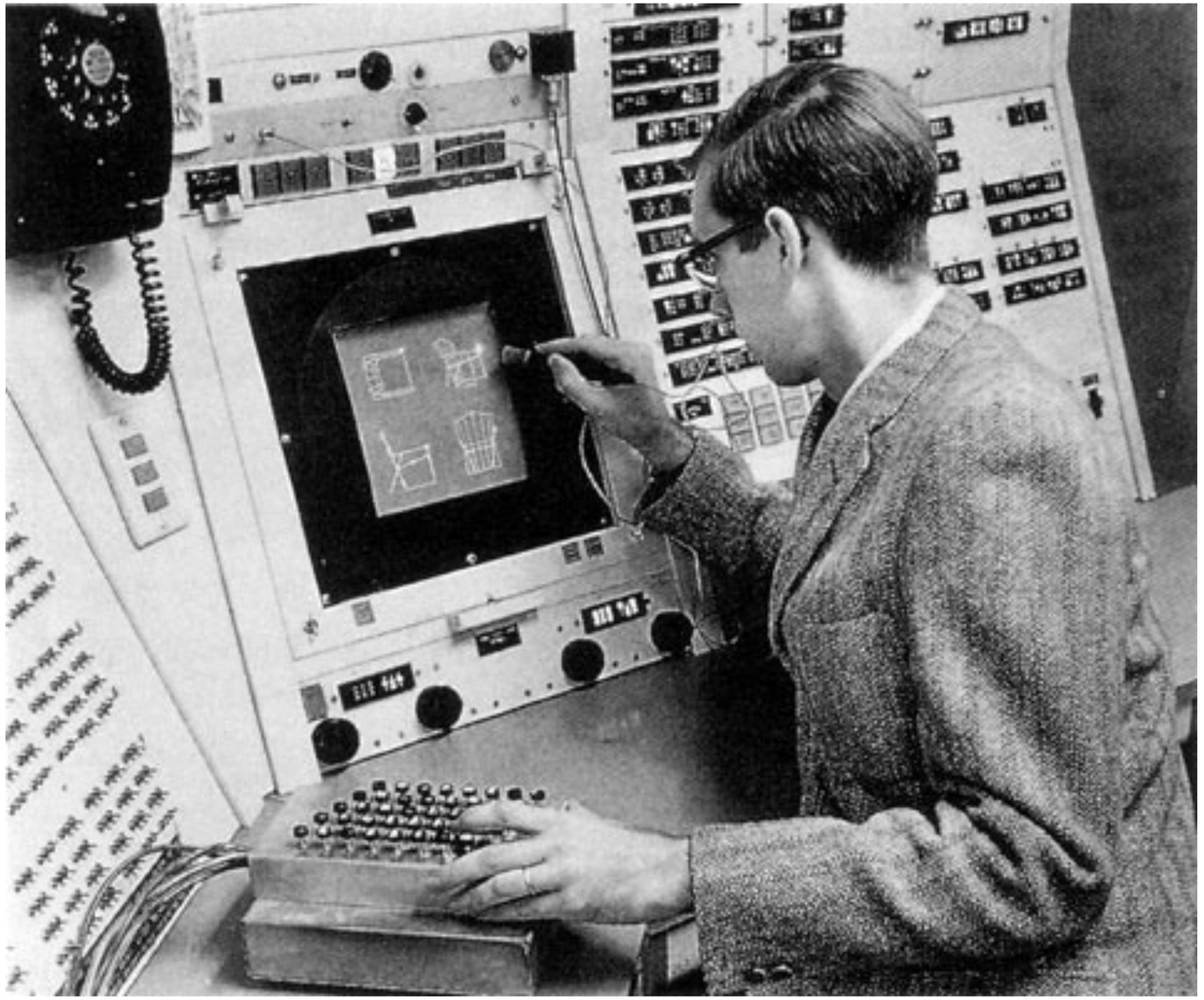
Your instructor (me)

Prof. Kayvon

Your CA

Sharon

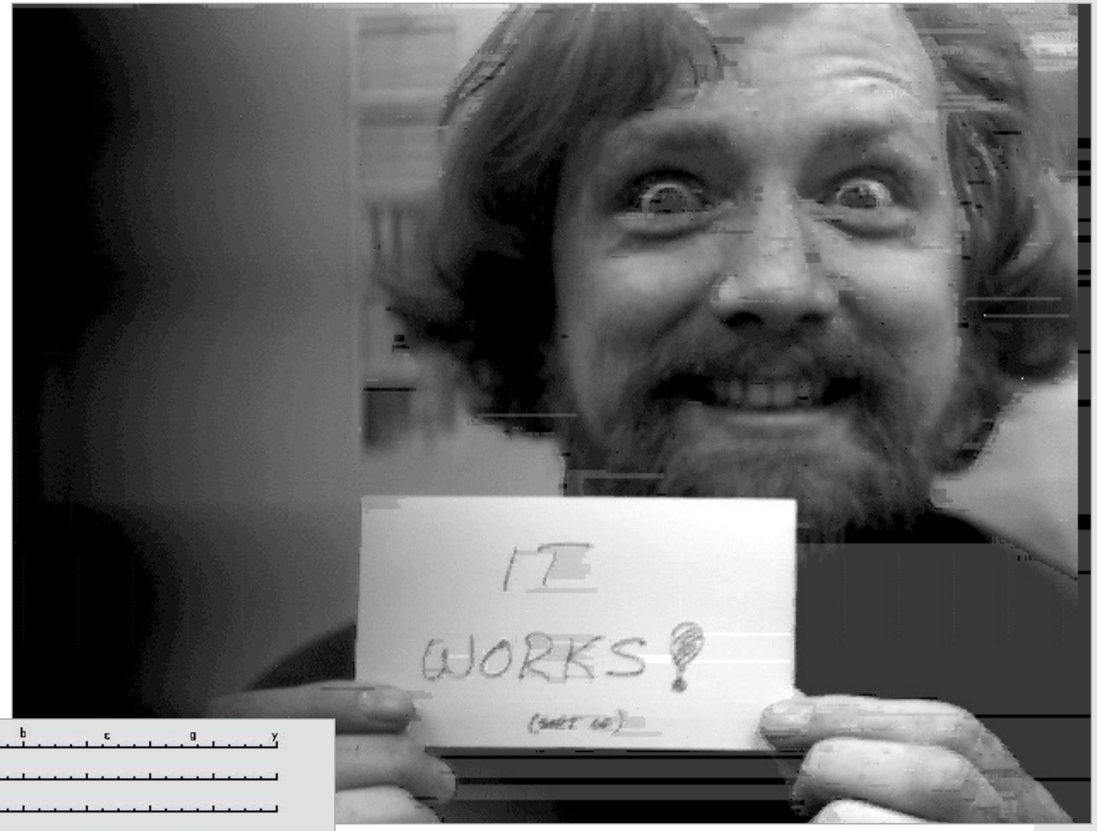
Visual computing applications have always demanded some of the world's most advanced computing systems

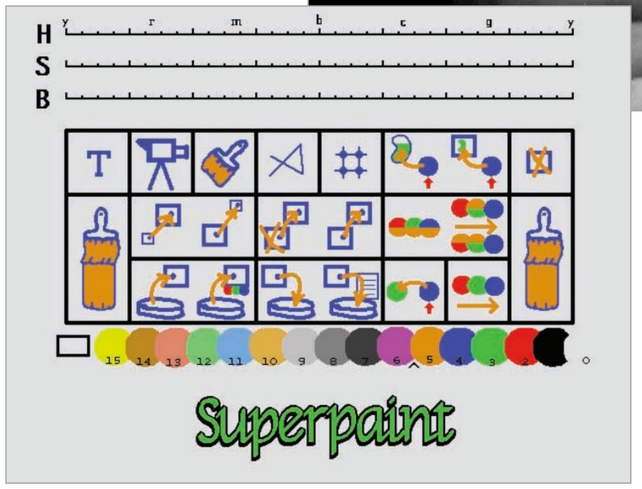


Ivan Sutherland's Sketchpad on MIT TX-2 (1962)

The frame buffer

Shoup's SuperPaint (PARC 1972-73)

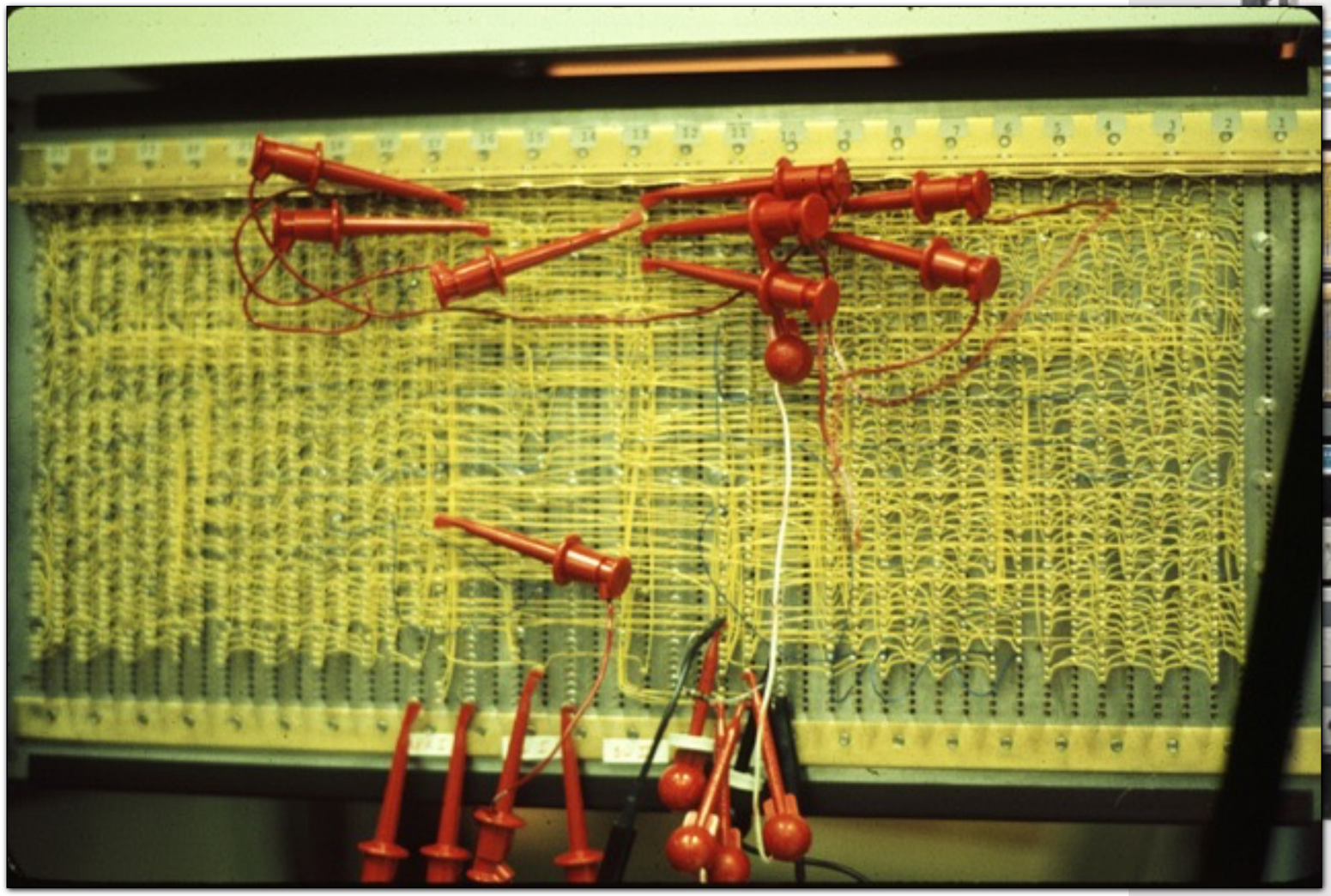




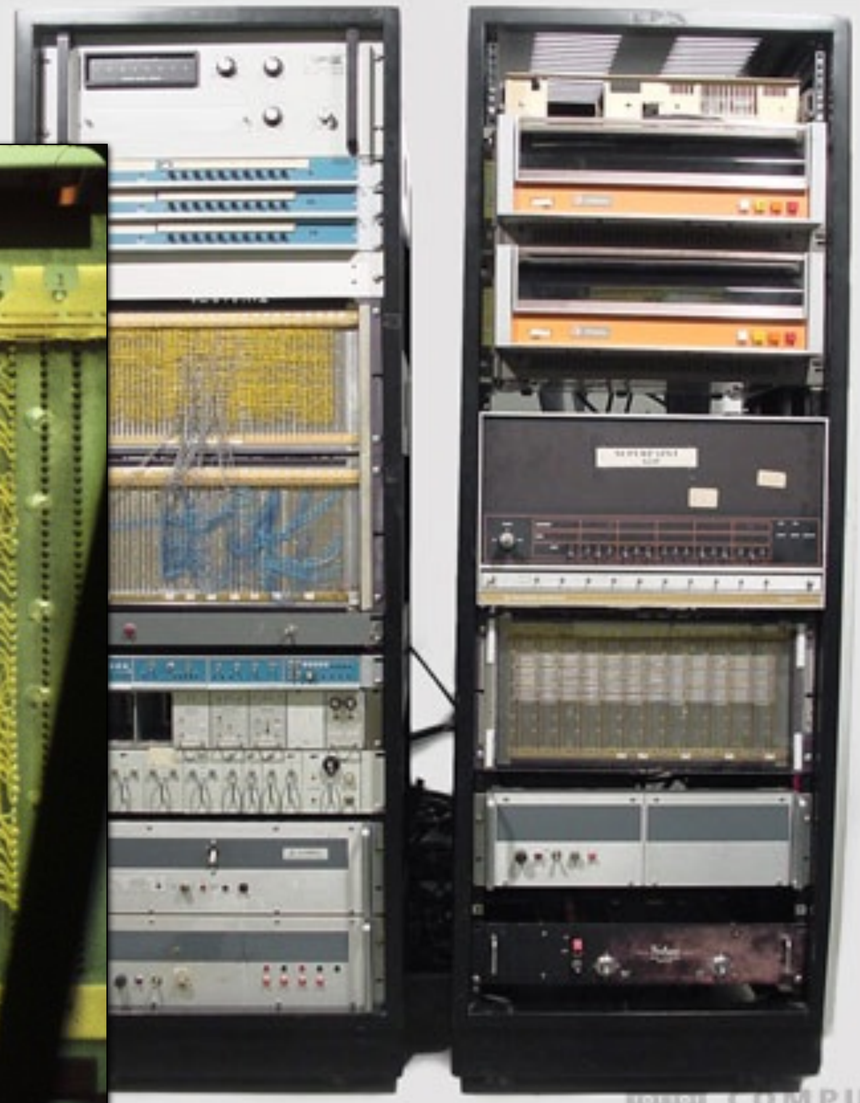
16 2K shift registers (640 x 486 x 8 bits)

The frame buffer

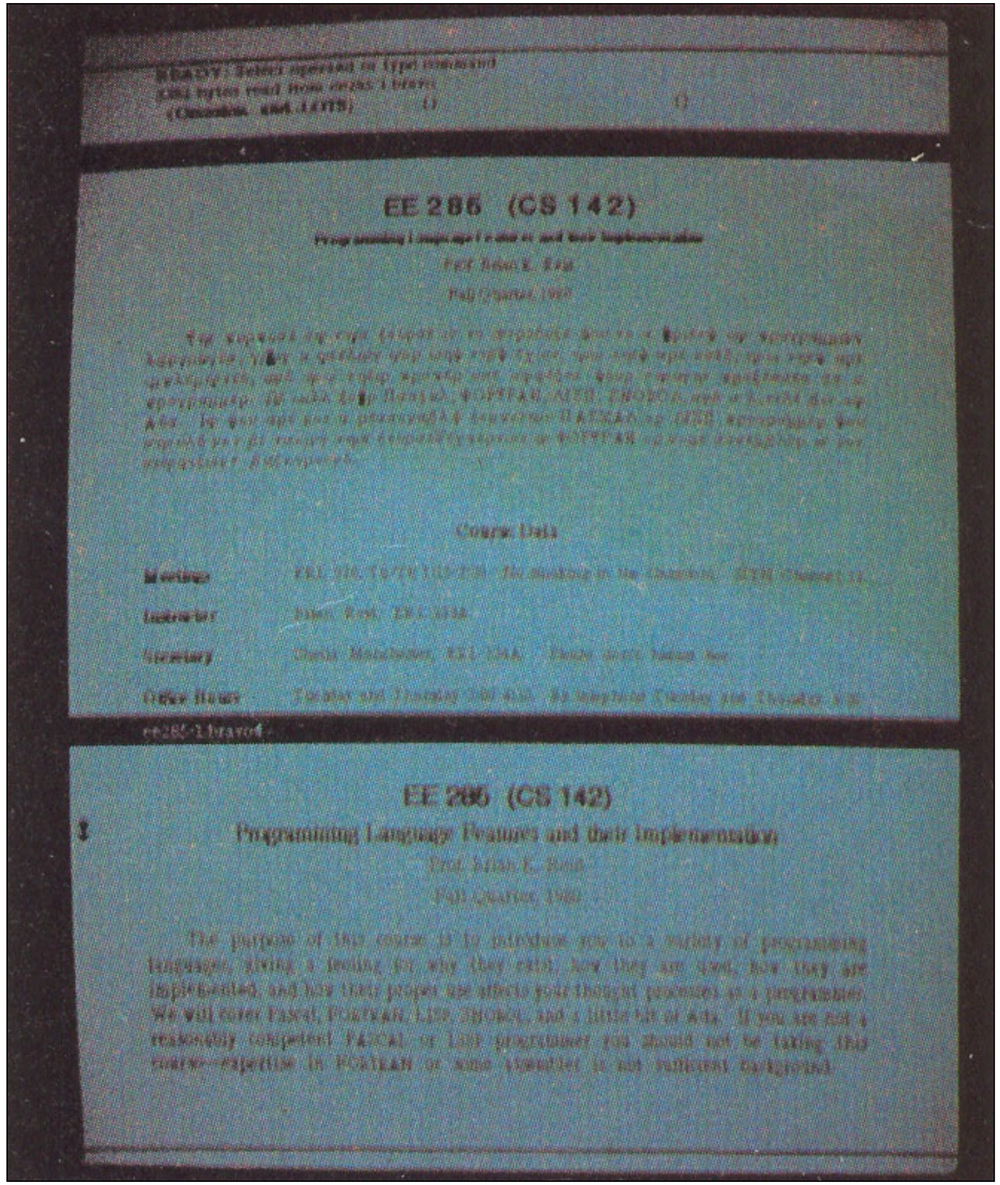
Shoup's SuperPaint (PARC 1972-73)



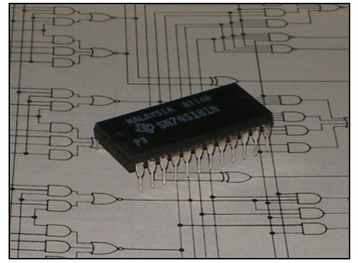
16 2K shift registers (640 x 486 x 8 bits)



Xerox Alto (1973)



Bravo (WYSIWYG)



TI 74181 ALU

Goal: render everything you've ever seen

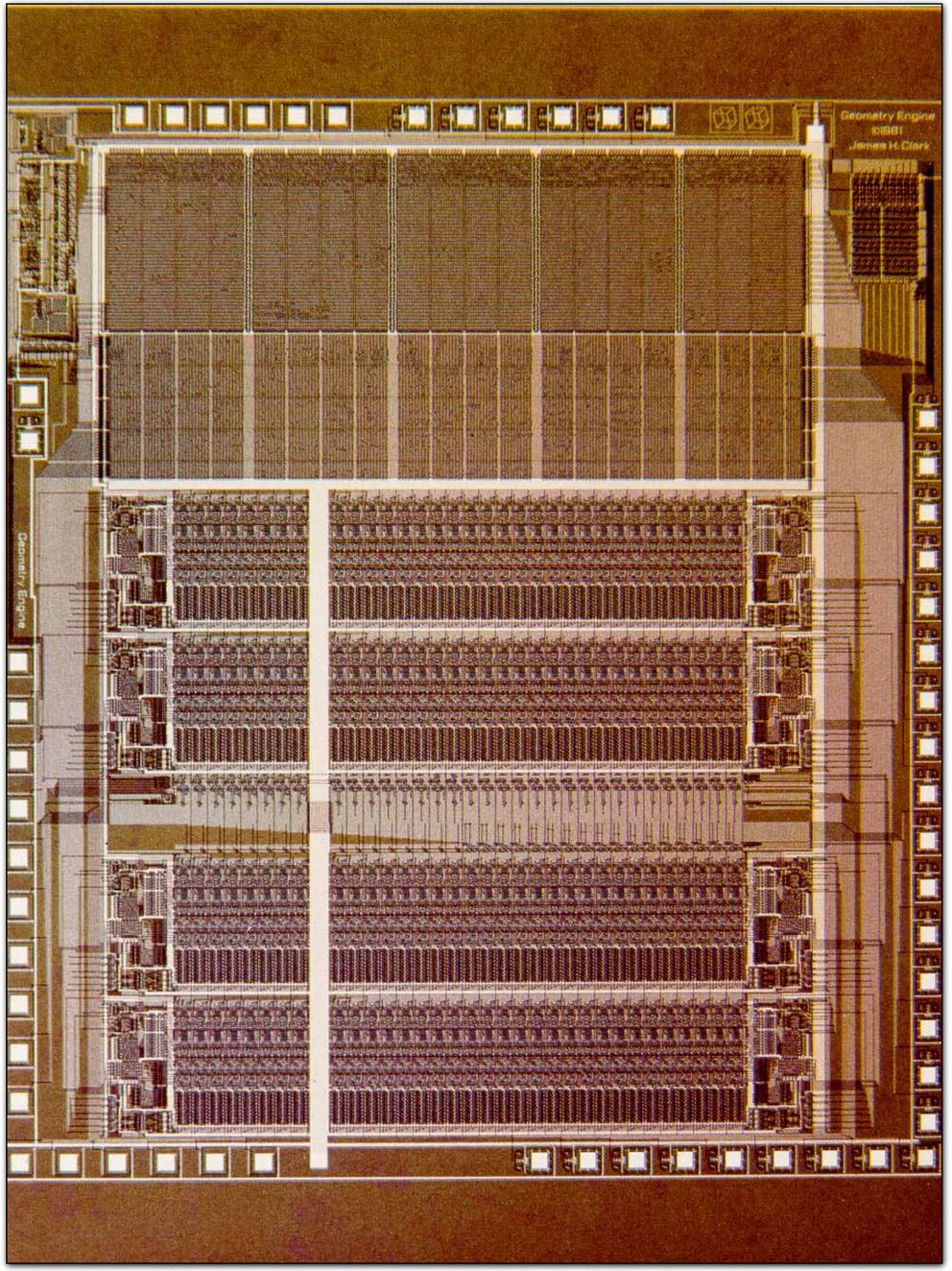
"Road to Pt. Reyes"
LucasFilm (1983)

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 4090 GPU

~ 80 TFLOPs fp32 *

About the performance of the world's top supercomputer in 2004 **

* doesn't count texture filtering ops, ray tracing ops, and 1300 TFLOPS of DNN compute

** not apples-to-apples since BlueGene/L is double precision flops

Cyberpunk 2077

Modern smartphones utilize multiple processing units

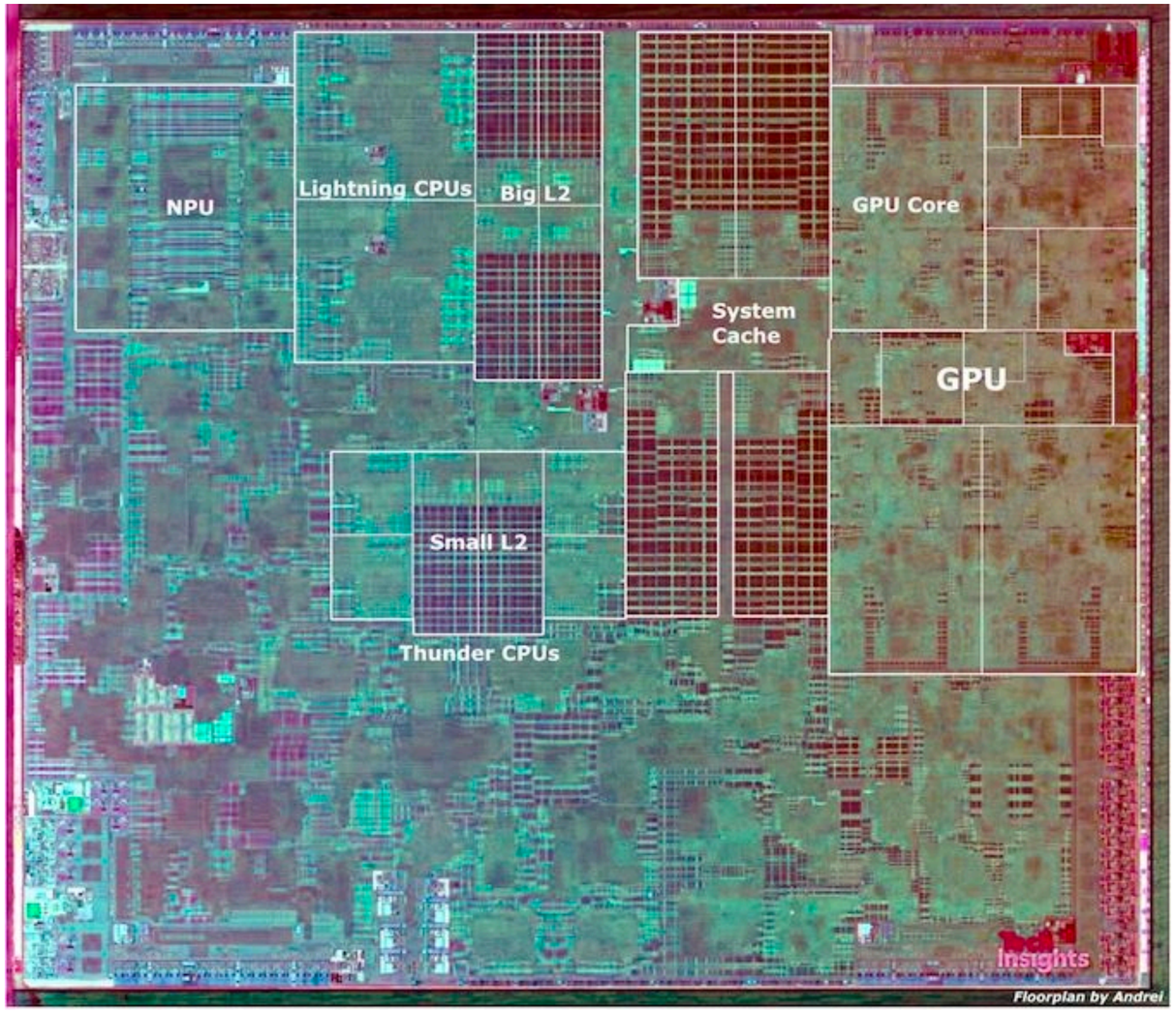


Image Credit: Anandtech / TechInsights Inc.

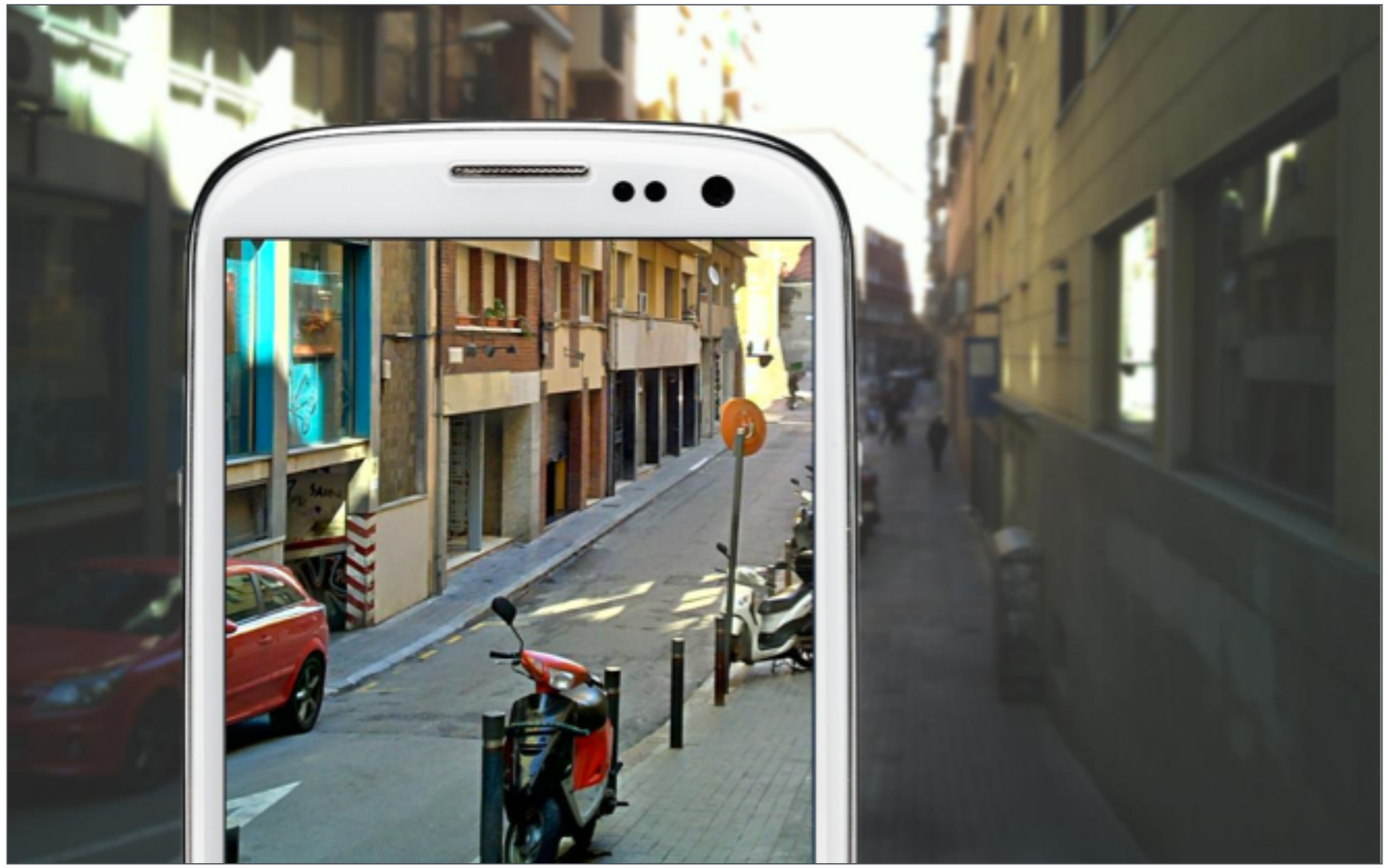
Apple A13 Bionic

Multi-core CPU (heterogeneous cores)
Multi-core GPU
Neural accelerator
Sensor processing accelerator
Video compression/decompression HW
Etc…

Digital photography: major driver of compute capability of modern smartphones

Portrait mode
(simulate effects of large aperture DSLR lens)

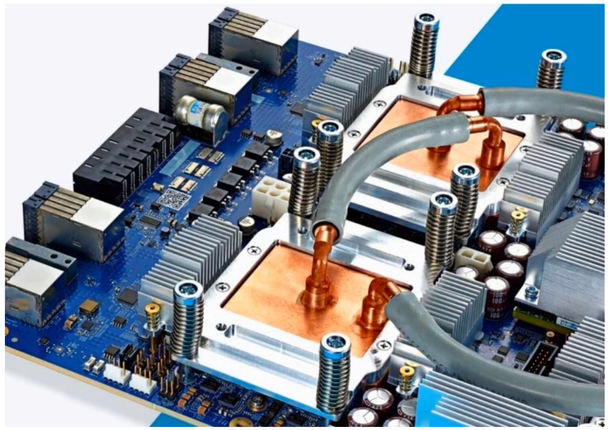
High dynamic range (HDR) photography



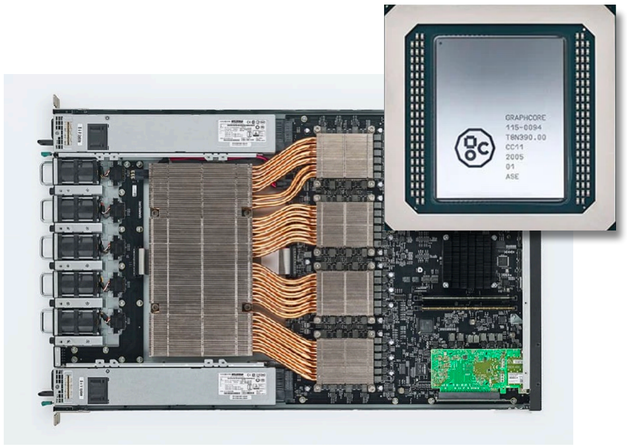
Apple Vision Pro (2024)

~11.4M visible pixels per panel
(28 Mpixel display)

Hardware acceleration of DNN inference/training



Google TPU3

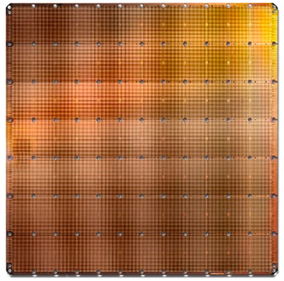


GraphCore IPU

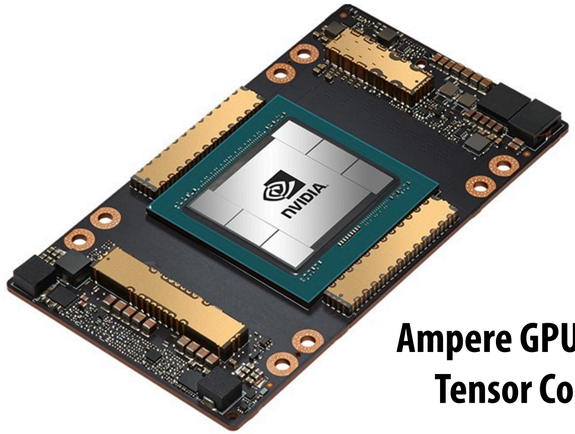
Apple Neural Engine

Intel Deep Learning
Inference Accelerator

SambaNova
Cardinal SN10

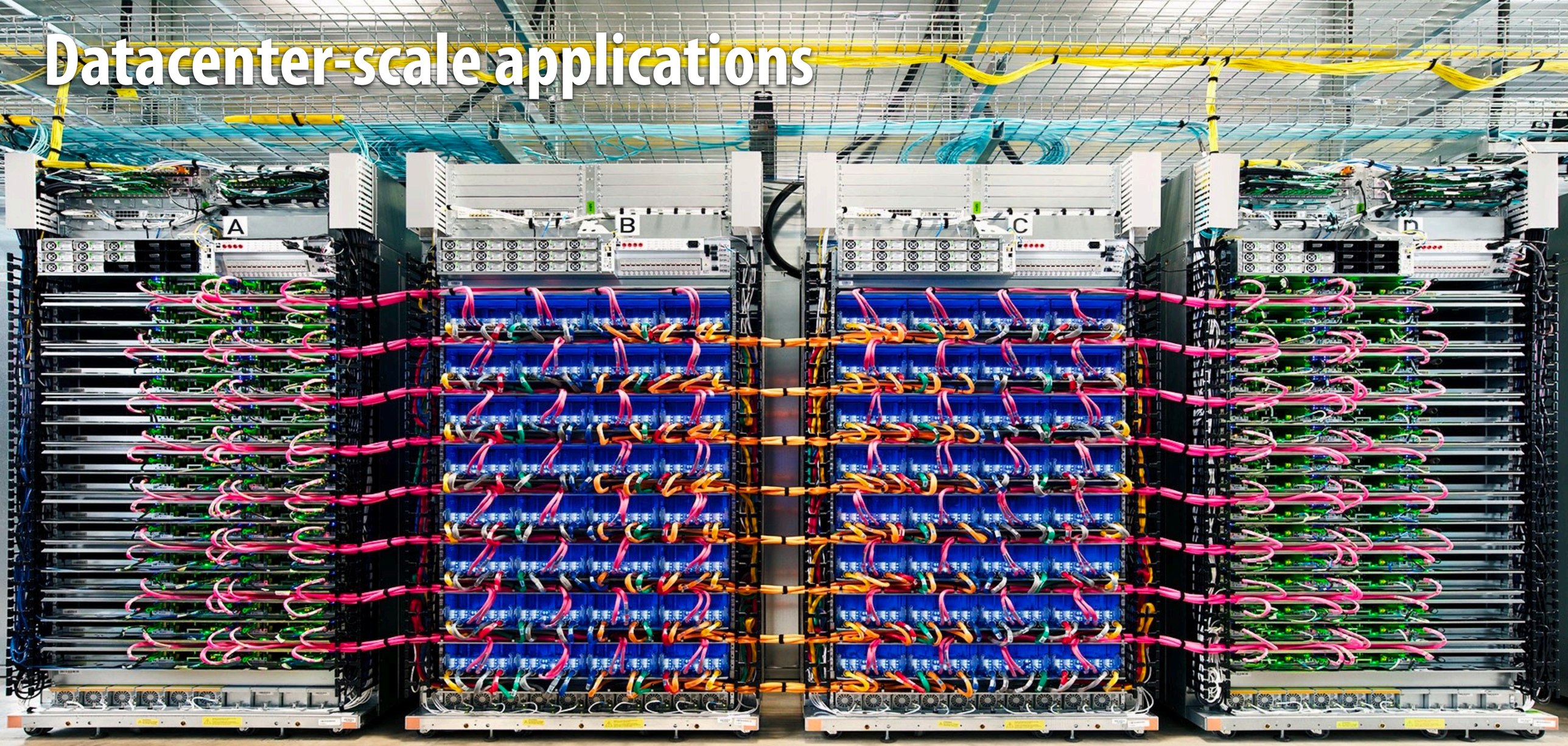


Cerebras Wafer Scale Engine



Ampere GPU with
Tensor Cores

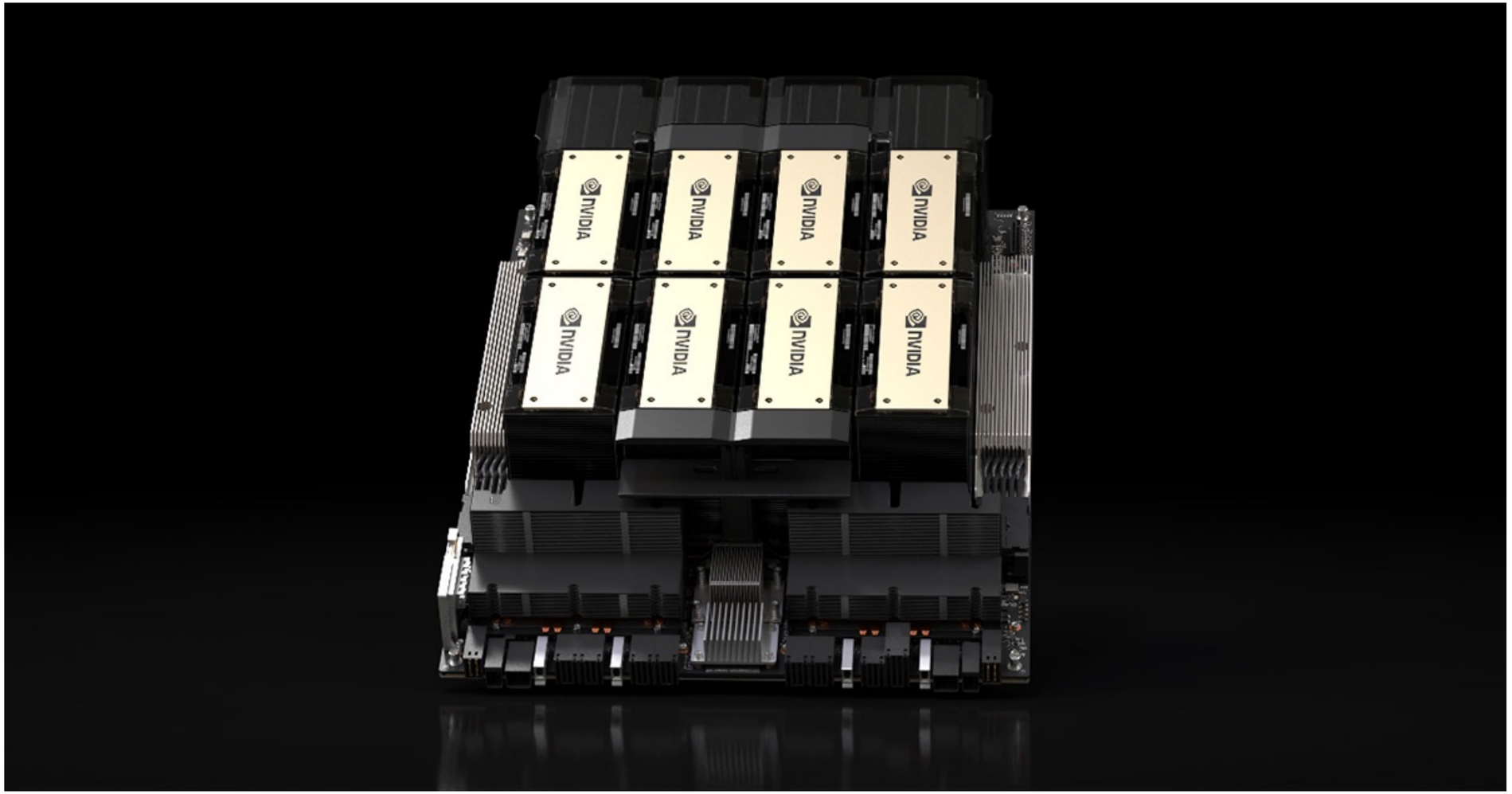
Datacenter-scale applications



Google TPU pods

Image Credit: TechInsights Inc.

NVIDIA B200 DGX

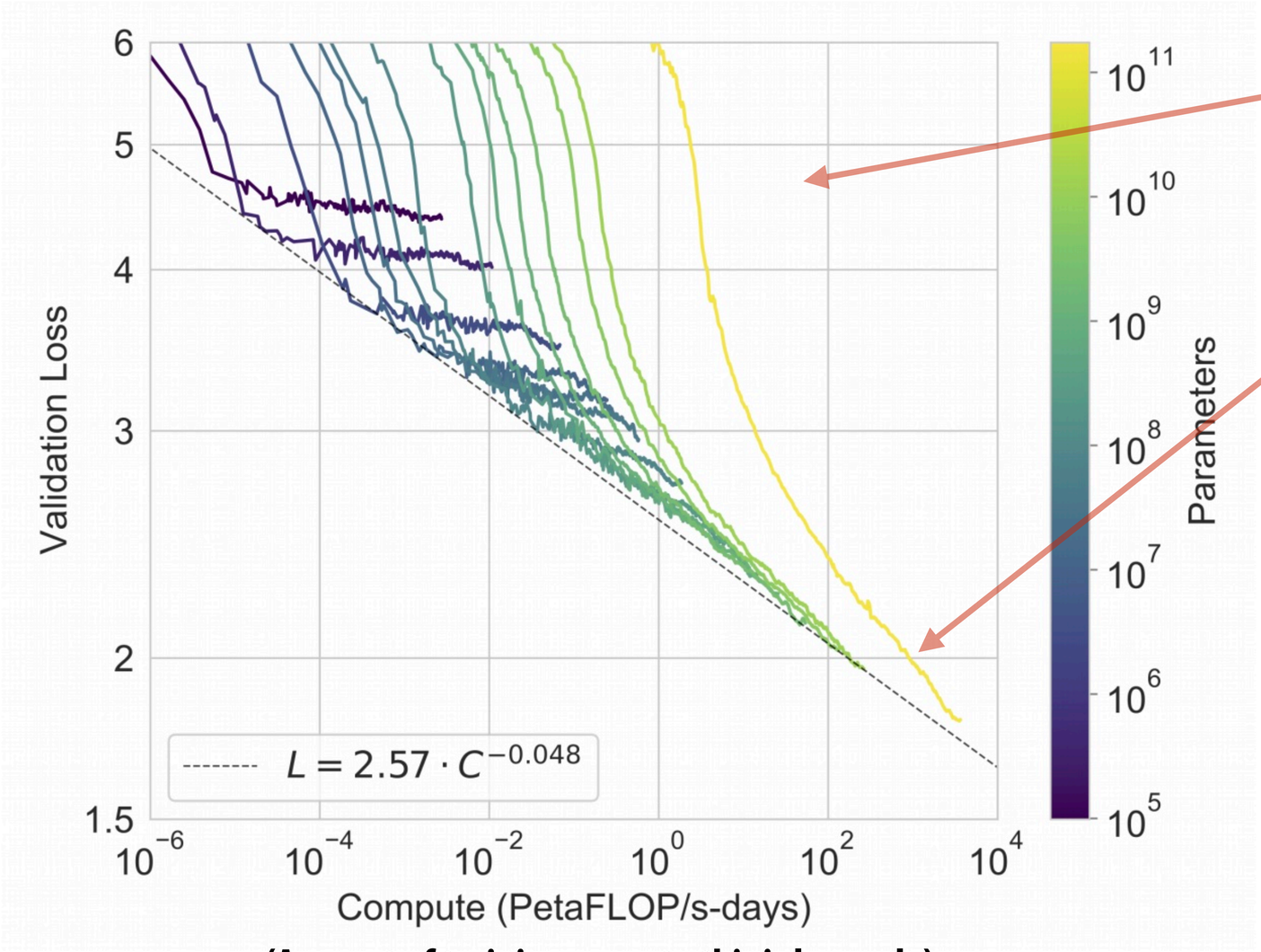


36 PFLOPs (Tensor Core BF16 precision)

600 TFLOPS (FP32 math)

Scaling up (for training big models)

Example: GPT-3 language model

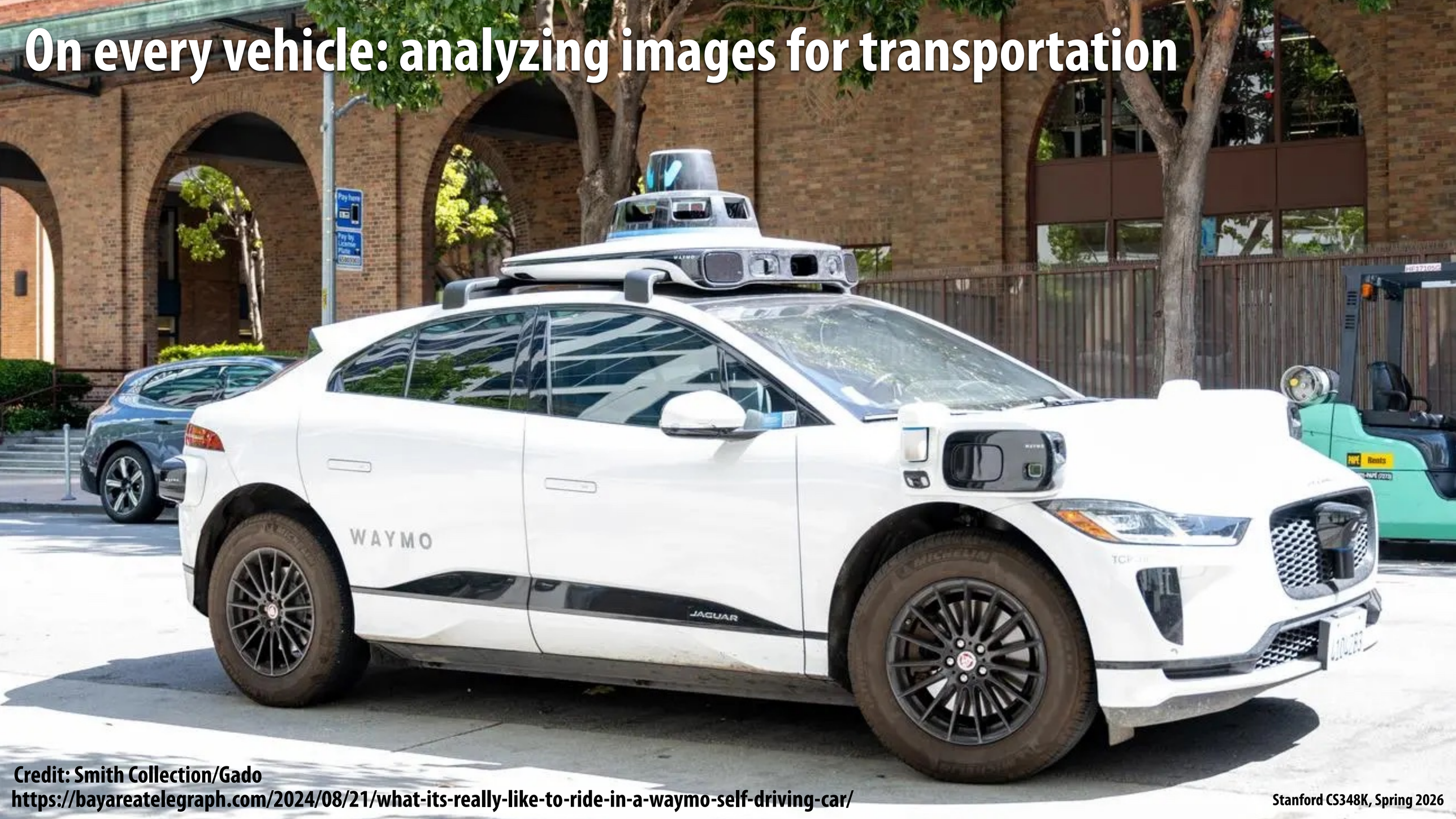


(Amount of training — note this is log scale)

Very big models +
More training
=
Better accuracy

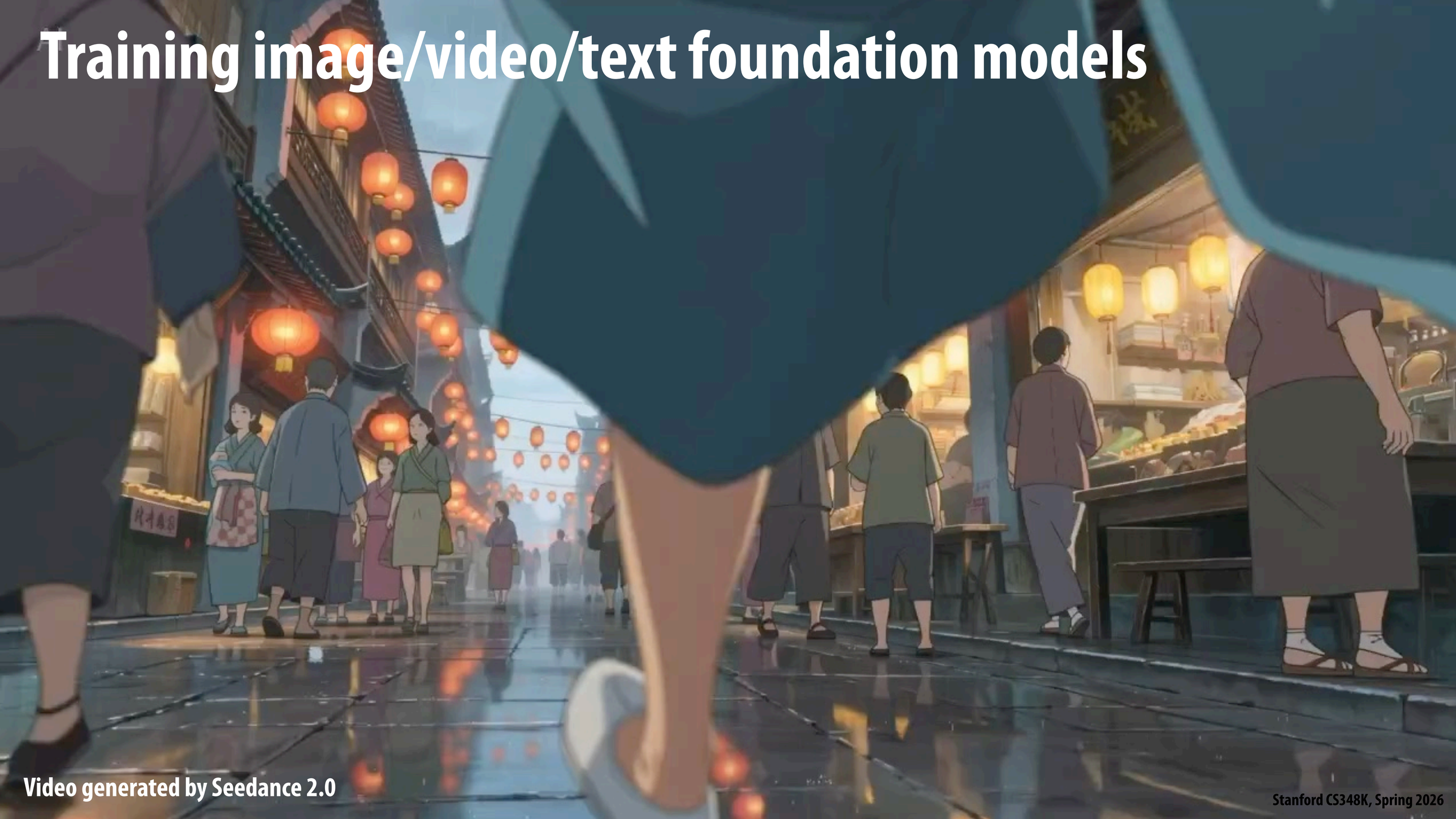
Power law effect:
exponentially more compute to take constant step in accuracy

On every vehicle: analyzing images for transportation



Credit: Smith Collection/Gado
https://bayareatelegraph.com/2024/08/21/what-its-really-like-to-ride-in-a-waymo-self-driving-car/

Training image/video/text foundation models



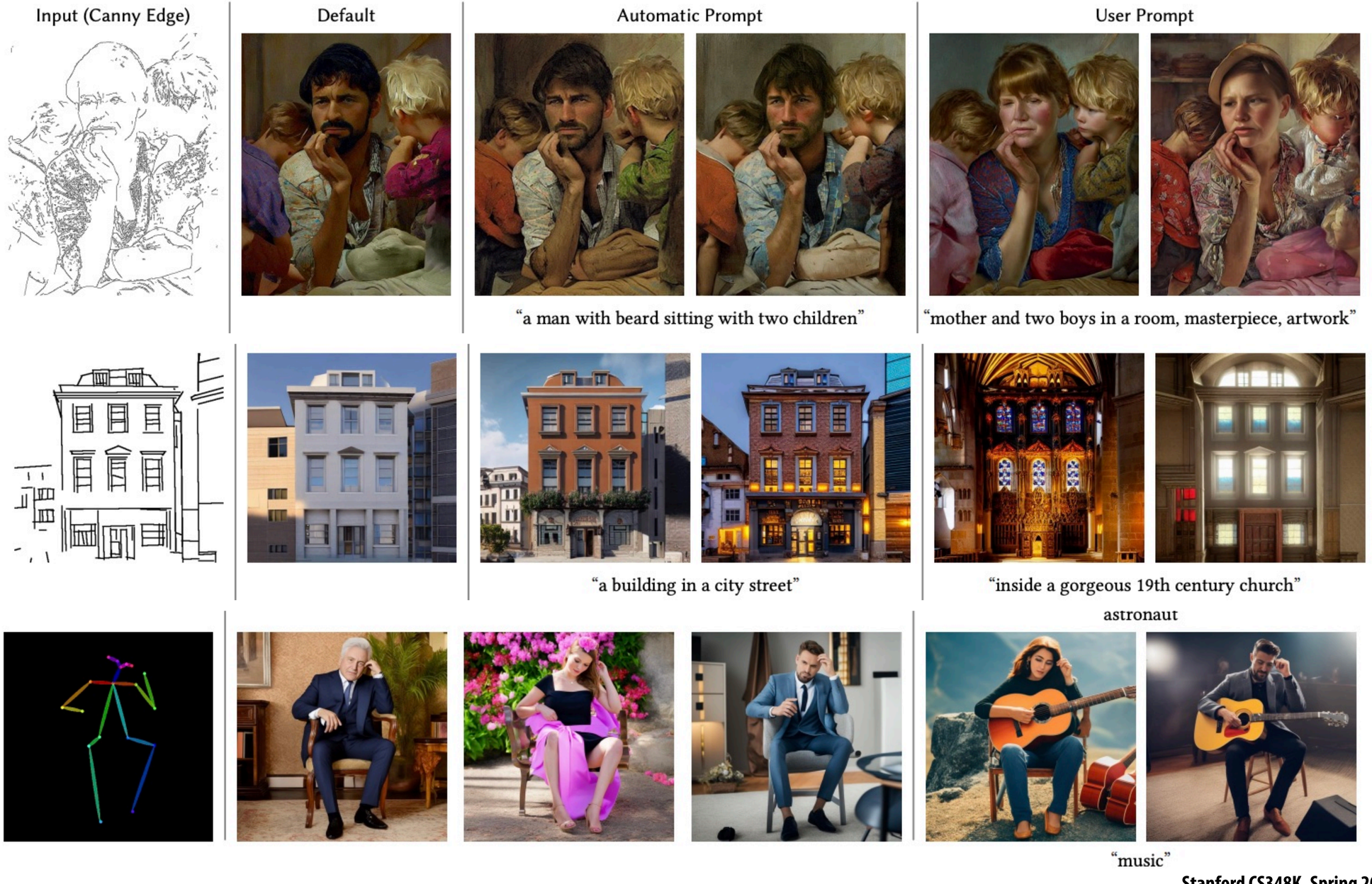
Video generated by Seedance 2.0

Interactive World Models

Interactive World Models

Controlling AI generated visual content

[ControlNet 2023]



What is this course about?

Accelerator hardware architecture?

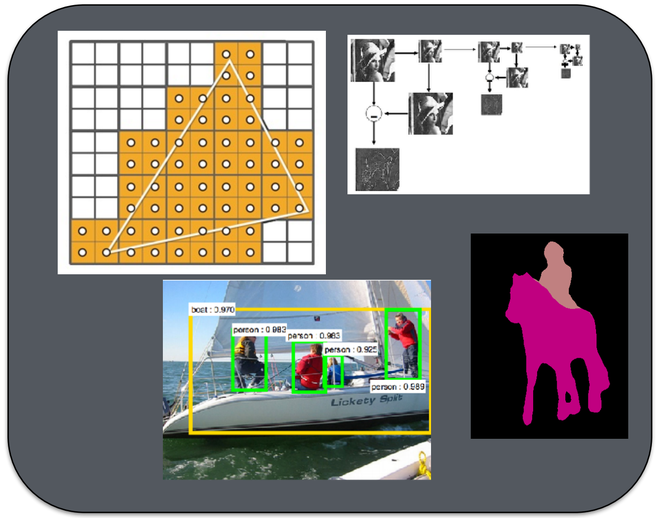
Graphics/vision/digital photography algorithms?

Programming systems?

What we will be learning about

Visual Computing Workloads

Algorithms for image/video processing, DNN evaluation, generative AI, differentiable X, agentic processing, 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

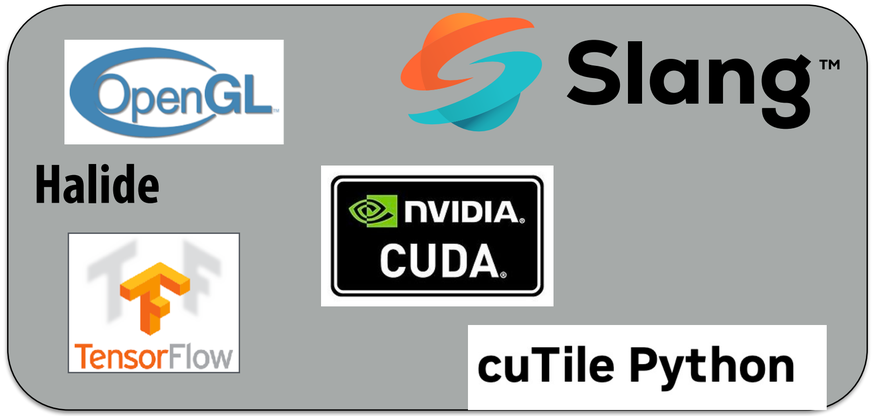
High-throughput hardware designs
(parallel, heterogeneous, and specialized)
fundamental constraints like area and power

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**

2026 course topics

The digital camera photo processing pipeline in modern smartphones

- What a modern smartphone camera does
- Programming abstractions for scheduling image processing code onto parallel hardware

Techniques (and programming systems) for automatically generating ML kernels for GPUs/TensorCores

- LLM based optimization techniques, vs. more traditional forms of search
- Domain Specific Languages for writing kernels

Use of differential programming and differentiable rendering for reconstruction/capture of 3D content

- The design of a modern differentiable programming language for the GPU
- Applications of differentiable programming

Making generative AI (for images, videos, animation, and more) usable

- The problem of controlling the output of these models
- How we can automatically verify the outputs meet user's intent

Generative AI for video and interactive worlds ("world models")

- Will future world simulators be more like game engines? Or more like learned models?
- Neural upsampling: ways to "lift" traditionally rendered images into much more realistic images
- How will address challenges of control and efficiency?

Developing Embodied AI Agents for 3D environments (games, robotics)

- RL and LLM-based agents and computer game bots
- Training agents in simulation, and the high-performance simulation systems needed to do this

ALWAYS SUBJECT TO CHANGE!

Activity: meet your classmates

Logistics and Expectations of Students

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 (25% in class + 25% written responses)
  - There will be ~1 assigned technical paper reading per class
  - You will submit a response to each reading by 9am on class days via Gradescope
  - We will do an impromptu in class question during most class days as well
  - We will start most classes with a 30-45 minute discussion of the reading

Implications:
Attendance is required
Auditing is not permitted

- 50% self-selected term project
  - I suggest you start thinking about projects now
  - We can meet 1-on-1 now about ideas, even if they are covered in later lecture topics
  - Teams of up to 3

Reading response template

Reminder: We will concatenate all responses and give everyone in the class a PDF of all responses. If you wish your answers to be anonymous to the class, please leave your name off your PDF.

Part 1: Top N (N<3) takeaways from discussions in the last class. Note: this part of the response is unrelated to the current reading, but should pertain to the discussion of the prior reading in class (or just discussion in the class in general):

- What was the most surprising/interesting thing you learned?
- Is there anything you feel passionate about (agreed with, disagreed with?) that you want to react to?
- What was your big takeaway in general?

Part 2: Answers/reactions to instructor's specific prompts for this reading. (Please see course website for prompts).

Part 3: [Optional] Do you have unanswered questions you would like to have specifically addressed. (We also encourage you to just post these questions on Ed immediately so anyone can answer!)

Thinking about goals and constraints

Systems research

A systems research contribution is typically about organizing resources to efficiently or elegantly solve a complex (set of) tasks in the face of conflicting goals and constraints

Examples:

Enable high-performance rendering of complex scenes that include a wide variety of types of materials, lights, and surfaces

(maximal performance vs. flexibility/versatility)

Design a VR rendering system that provides retina resolution output across the human field of view, without causing the user to throw up, for $300.

(achieving minimum viable requirements vs. cost)

Allow a scientist to express algorithms naturally in terms of differential geometry operators yet still obtain performance of hand-tuned code on a GPU

(productivity vs. good performance)

Flavors of organization:

- **Machine organization (hardware architecture)**
- **Software organization (software architecture)**
- **Human-thought organization (programming frameworks/ languages, interfaces)**

Metrics of efficiency:

- **Performance (how fast does a task complete, how big of a task can be done in a fixed amount of time)**
- **Material cost**
- **Energy**
- **Human productivity / expressibility**

Can do something != should do something this way

- Algorithms papers can get by with a contribution that says "here's a way to do solve a problem".

- Systems papers typically argue that a particular way of structuring a solution is a good one
  - This is a "good way" to think about things…

- How do we define good?

> As is the case for almost all system designs, most features of the Cg language and system are not novel when considered individually. However, when considered as a whole, we believe that the system and its design goals are substantially different from any previously-implemented system for programming graphics hardware.

Describing an implementation alone, regardless of how large or complex an effort, is not a good systems paper

A.K.A. let me tell you what the system does.

This is the common failure mode of authors… they describe the features or capabilities of an implementation but no more generalizable insights… and the reviewers (rightfully) reject it.

Why is clear articulation of goals and constraints is so important…

I've thought about this problem deeply, and so I'll distill things down to what really matters for the reader.

Systems architects begin with explicitly stating goals, non-goals, and assumptions

- "Given these inputs, we wish to generate these outputs…"
- "We are working under the following constraints"
  - Example: the output images should have these properties
  - Example: the algorithm should have certain performance…
    - Should run in real time
    - Should be widely parallelizable, so it can run efficiently on a multi-core GPU
  - Example: the user experience must have these properties
    - Should not require user intervention to get "good" output
- "The following things are NOT goals of the system (and why)"

Why is establishing goals and constraints so important?

- It defines the requirements of a "good" solution
- It provides a framework for assessing/evaluating the quality of the designer’s decisions
- And for communicating with others: It provides context that leads to more generalizable knowledge. Since a paper’s readers likely do not have the same goals and constraints as a paper's authors, understanding the author's goals and constraints helps readers understand which design decisions are applicable to their own problems

Clear articulation of goals and constraints

- What are the salient aspects of the design problem to be solved? (that someone that hasn't spent years solving the problem wouldn't obviously know)
- How do they compliment or conflict with each other?
- What are priorities on these goals?

Example from Reyes Image Rendering Architecture, [Cook 84]

Many of our design decisions are based on some specific assumptions about the types of complex scenes that we want to render and what makes those scenes complex. Since this architecture is optimized for these types of scenes, we begin by examining our assumptions and goals.

- **Model complexity.** We are interested in making images that are visually rich, far more complex than any pictures rendered to date. This goal comes from noticing that even the most complex rendered images look simple when compared to real scenes and that most of the complexity in real scenes comes from rich shapes and textures. We expect that reaching this level of richness will require scenes with hundreds of thousands of geometric primitives, each one of which can be complex.
- **Model diversity.** We want to support a large variety of geometric primitives, especially data amplification primitives such as procedural models, fractals [18], graftals [35], and particle systems [30, 31].
- **Shading complexity.** Because surface reflection characteristics are extremely varied and complex, we consider a programmable shader a necessity. Our experience with such a shader [11] is that realistic surfaces frequently require complex shading and a large number of textures. Textures can store many different types of data, including surface color [8], reflections (environment maps) [3], normal perturbation (bump maps) [4], geometry perturbation (displacement maps) [11], shadows [32], and refraction [25].
- **Minimal ray tracing.** Many non-local lighting effects can be approximated with texture maps. Few objects in natural scenes would seem to require ray tracing. Accordingly, we consider it more important to optimize the architecture for complex geometries and large models than for the non-local lighting effects accounted for by ray tracing or radiosity.
- **Speed.** We are interested in making animated images, and animation introduces severe demands on rendering speed. Assuming 24 frames per second, rendering a 2 hour movie in a year would require a rendering speed of about 3 minutes per frame. Achieving this speed is especially challenging for complex images.
- **Image Quality.** We eschew aliasing and faceting artifacts, such as jagged edges, Moiré patterns in textures, temporal strobing, and highlight aliasing.
- **Flexibility.** Many new image rendering techniques will undoubtedly be discovered in the coming years. The architecture should be flexible enough to incorporate many of these new techniques.

Clear articulation of goals and constraints

The language and system design was guided by a handful of high-level goals:

- **Ease of programming.**
  Programming in assembly language is slow and painful, and discourages the rapid experimentation with ideas and the easy reuse of code that the off-line rendering community has already shown to be crucial for shader design.
- **Portability.**
  We wanted programs to be portable across hardware from different companies, across hardware generations (for DX8-class hardware or better), across operating systems (Windows, Linux, and MacOS X), and across major 3D APIs (OpenGL [Segal and Akeley 2002] and DirectX [Microsoft Corp. 2002a]). Our goal of portability across APIs was largely motivated by the fact that GPU programs, and especially "shader" programs, are often best thought of as art assets – they are associated more closely with the 3D scene model than they are with the actual application code. As a result, a particular GPU program is often used by multiple applications (e.g. content-creation tools), and on different platforms (e.g. PCs and entertainment consoles).
- **Complete support for hardware functionality.**
  We believed that developers would be reluctant to use a high-level language if it blocked access to functionality that was available in assembly language.
- **Performance.**
  End users and developers pay close attention to the performance of graphics systems. Our goal was to design a language and system architecture that could provide performance equal to, or better than, typical hand-written GPU assembly code. We focused primarily on interactive applications.
- **Minimal interference with application data.**
  When designing any system layered between applications and the graphics hardware, it is tempting to have the system manage the scene data because doing so facilitates resource virtualization and certain global optimizations. Toolkits such as SGI's Performer [Rohlf and Helman 1994] and Electronic Arts's EAGL [Lalonde and Schenk 2002] are examples of software layers that successfully manage scene data, but their success depends on both their domain-specificity and on the willingness of application developers to organize their code in conforming ways. We wanted Cg to be usable in existing applications, without the need for substantial reorganization. And we wanted Cg to be applicable to a wide variety of interactive and non-interactive application categories. Past experience suggests that these goals are best achieved by avoiding management of scene data.

Goals

- **Support for non-shading uses of the GPU.**
  Graphics processors are rapidly becoming sufficiently flexible that they can be used for tasks other than programmable transformation and shading (e.g. [Boltz et al. 2003]). We wanted to design a language that could support these new uses of GPUs.

Some of these goals are in partial conflict with each other. In cases of conflict, the goals of high performance and support for hardware functionality took precedence, as long as doing so did not fundamentally compromise the ease-of-use advantage of programming in a high-level language.

Philosophy of addressing conflicts

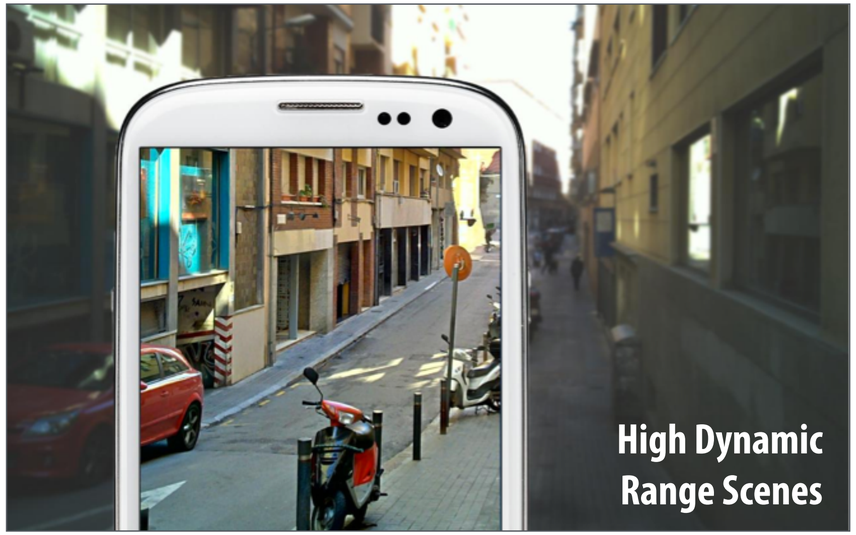
Often system designers must preserve substantial compatibility with old system interfaces (e.g. OpenGL is similar to IRIS GL). In our case, that was a non-goal because most pre-existing high level shader code (e.g. RenderMan shaders) must be modified anyway to achieve real-time performance on today's graphics architectures.

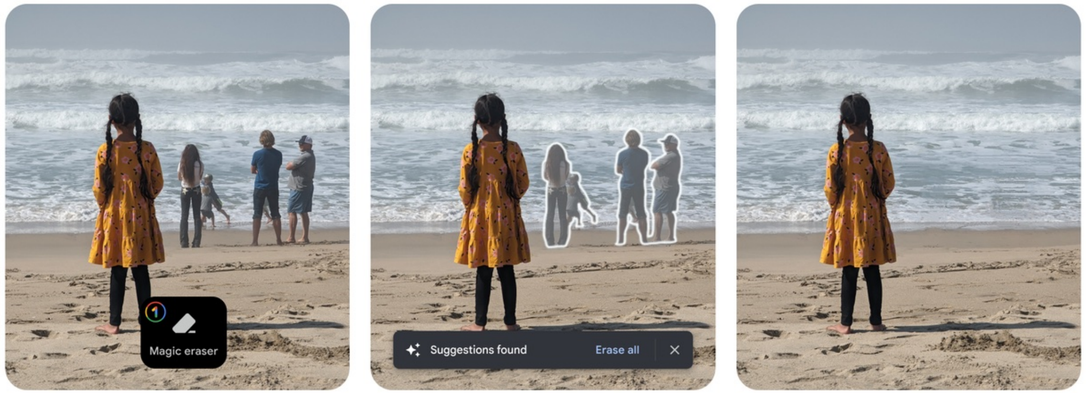
← Non-goals

Cg, [Mark 04]

Activity: let's design two systems

System 1: Anthropic is getting into the smartphone camera business. You were just hired as the lead architect for ClaudeCam





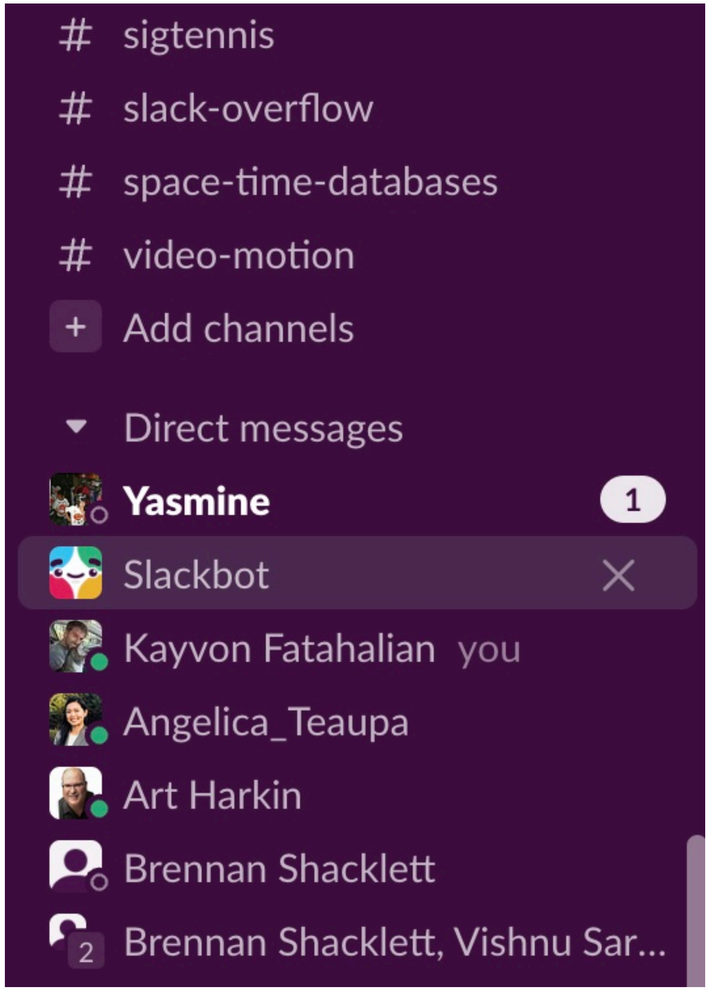
Magic Eraser Feature

Image credit: Google

Discussion

- What are your image quality / feature list goals?
- What are your performance goals? Why?
- What are your user experience goals?

System 2: Kayvon wants to have an office accessible to the world



Can we solve the case of a remote person interrupting me in my office for a quick conversation (in a socially acceptable way)?


EXIT
What's your UIST halloween costume?
Clippy
nsb group dresses (i.e. michael
FALL
Y'ALL


Keyvon Fatahalian
366


EXIT
What's your UIST halloween costume?
msb group dresses (a)
CLIPPY


EXIT
What's your UIST halloween costume?
clippy
msb group dresses as michael
Please let

Tonight's reading

- "What Makes a Graphics System Beautiful," (2019), a blog post by me about thinking about goals and constraints.
  - The ideas in this post are how I want to you think about the systems we discuss in this course

- "Burst Photography for High Dynamic Range and Low-light Imaging on Mobile Cameras" (2016)
  - How a key feature in the Google Pixel phone camera works
  - Tonight read the front part of the paper for goals/constraints/assumptions.
  - We'll finish up the technical details of the paper after next lecture

Welcome to CS348K!

- See website for tonight's reading