Machine Learning Systems

Build scalable ML systems through the vertical optimization of LM algorithms, system software, and hardware

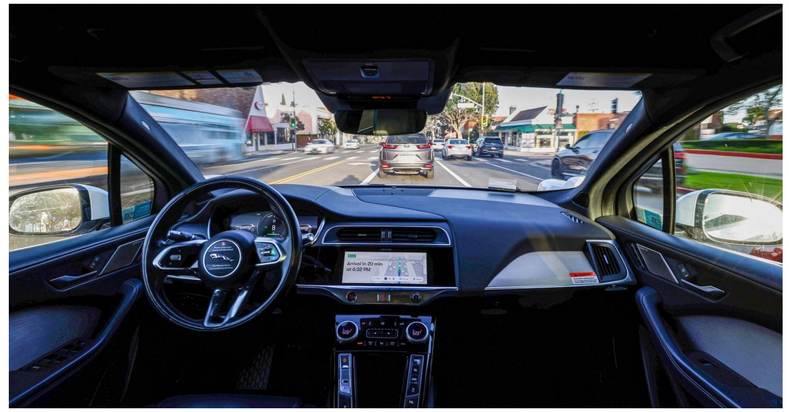
Li Shang
lishang@slai.edu.cn

AI Infra

Wearables

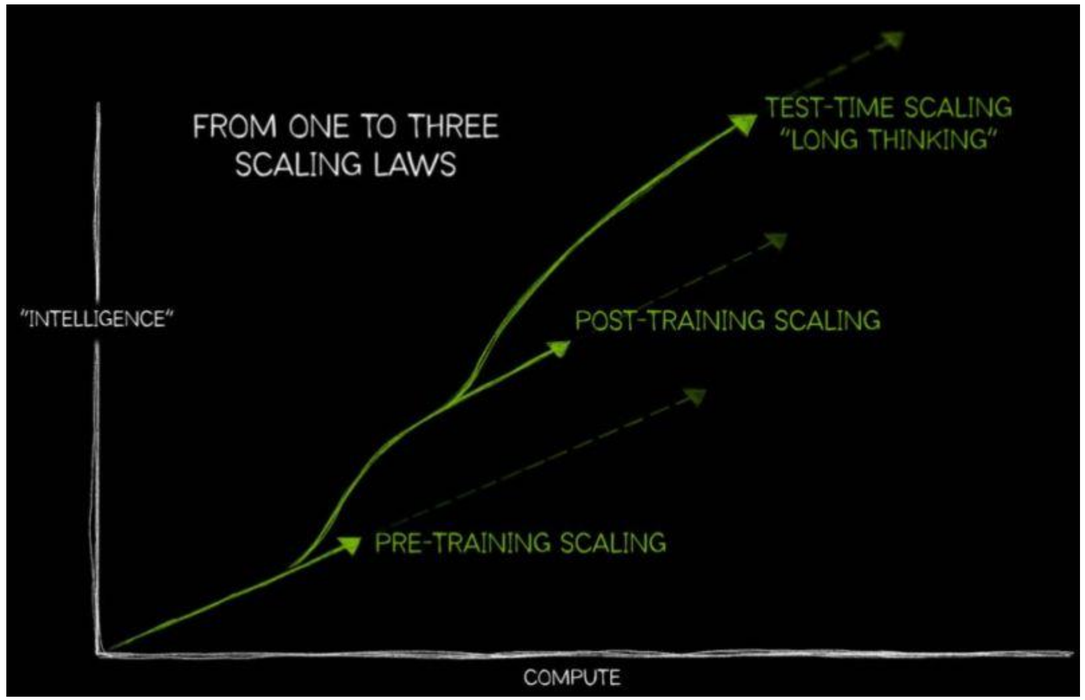
Humanoid Robots

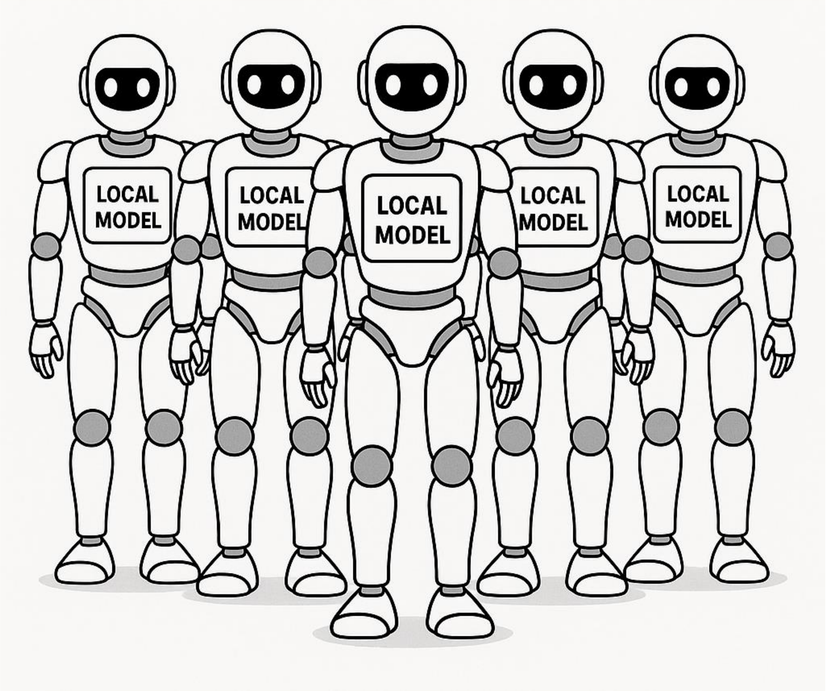
Machine Learning Systems



Robotaxi

Mission Statement





Acknowledgement

Machine learning systems is a broad and rapidly evolving field. The course material has been developed using a broad spectrum of resources, including research papers, lecture slides, blog posts, research talks, tutorial videos, and other materials shared by the research community.

Part of the course material was created by LLM itself.

Why is designing a computer system so difficult?

–Tom Jerry

Lesson 0: Good system design starts with predicting the workload of tomorrow.

–Tom Jerry

Lesson 1: Machine intelligence is about learning compact representations in high-dimensional space

–Tom Jerry

Intelligence requires mapping this raw data into structured feature spaces where meaningful patterns emerge.

1. Why High-Dimensional?

- **Raw Data is High-Dimensional**
  - Images: millions of pixels
  - Speech: long waveforms
  - Language: billions of tokens

Intelligence requires mapping this raw data into structured feature spaces where meaningful patterns emerge.

2. Representation Learning as Dimensional Geometry

- A learned representation is a **point in high-dimensional space**:

$$f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^d, \quad d \ll n$$

- Geometry encodes semantics:
  - **Closeness** = similarity (word embeddings: “king – man + woman ≈ queen”)
  - **Directions** = latent factors (sentiment, object orientation)
  - **Clusters** = categories

Intelligence requires mapping this raw data into structured feature spaces where meaningful patterns emerge.

3. Examples

- **Vision (CNNs):** filters project images into a space where dogs are close to dogs, cars to cars.
- **Language (Transformers):** contextual embeddings capture syntax/semantics in a hidden high-dimensional manifold.
- **Reinforcement Learning:** states and actions mapped into latent embeddings for generalization across environments.

Intelligence requires mapping this raw data into structured feature spaces where meaningful patterns emerge.

4. Why High-Dimensional Spaces Work

- **Concentration of Measure:** In high dimensions, distances behave differently — making separation easier for learning.
- **Expressivity:** More degrees of freedom allow encoding complex concepts.
- **Linear Separability:** High-dimensional embeddings often turn nonlinear problems in raw space into linearly separable problems.

Intuition

🔷 Step 1. Semantic subspace projection

- Suppose you have embeddings $x \in \mathbb{R}^d$.
- A **semantic category** (say *gender*, *tense*, or *sentiment*) can be modeled as a **linear subspace** $U \subset \mathbb{R}^d$.
- Mathematically, you can project a vector onto that subspace:

$$x_U = P_U x = U(U^\top U)^{-1} U^\top x$$

  where U is a basis spanning the semantic subspace.

- This isolates the "component" of the embedding that reflects the category.

Intuition

🔷 Step 2. Measuring relationships

- Once vectors are projected into the subspace, their **relative similarity** is measured (often by cosine similarity):

$$\cos(x_U, y_U) = \frac{x_U \cdot y_U}{\|x_U\| \|y_U\|}$$

- This gives a **category-specific similarity metric** — two features might be globally different, but close *with respect to* a chosen semantic category.

Lesson 2: Tensor operation is the primary workload.

–Tom Jerry

$$\begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-1,0} & b_{n-1,1} & \cdots & b_{n-1,n-1} \end{bmatrix}$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix} \quad \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

$$d = 2048 \quad \text{(hidden dimension)}, \qquad N = 24 \quad \text{(layers)}, \qquad V = 50{,}000 \quad \text{(vocab size)},$$

$$B = 8 \quad \text{(batch size)}, \qquad L = 1024 \quad \text{(sequence length)}, \qquad h = 16 \quad \text{(attention heads)}.$$

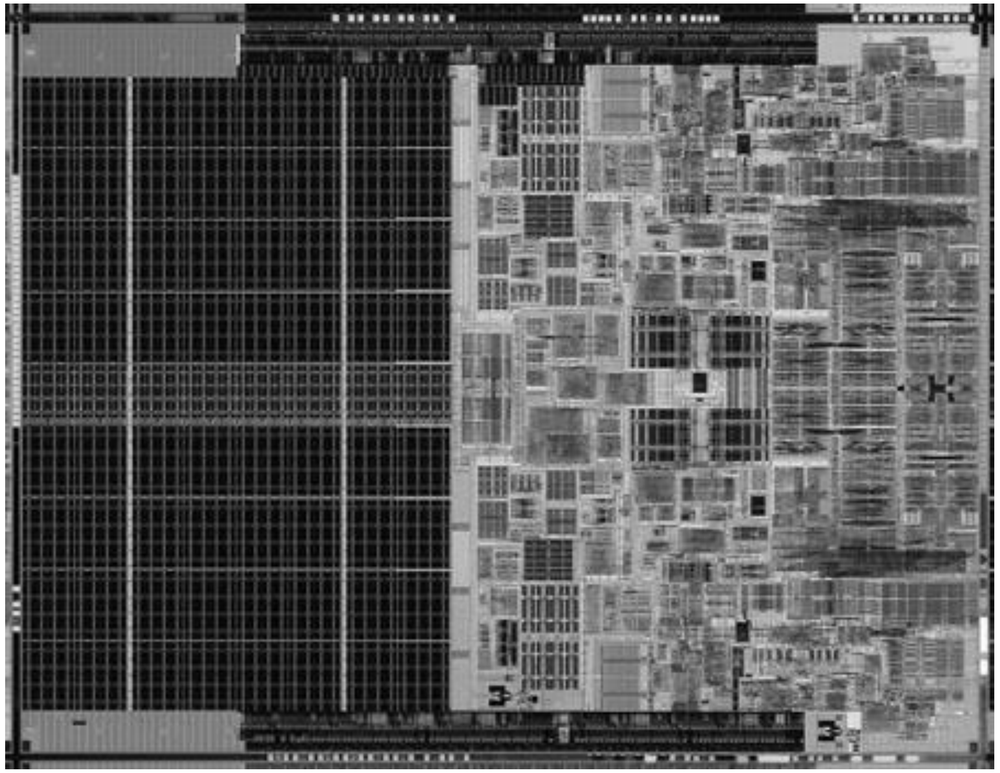
- Parameters: $12Nd^2 + Vd$
- Forward activations: $9BLd + 2BhL^2$ per layer
- Backpropagation: $\sim 2 - 3\times$ activations + gradients + optimizer states

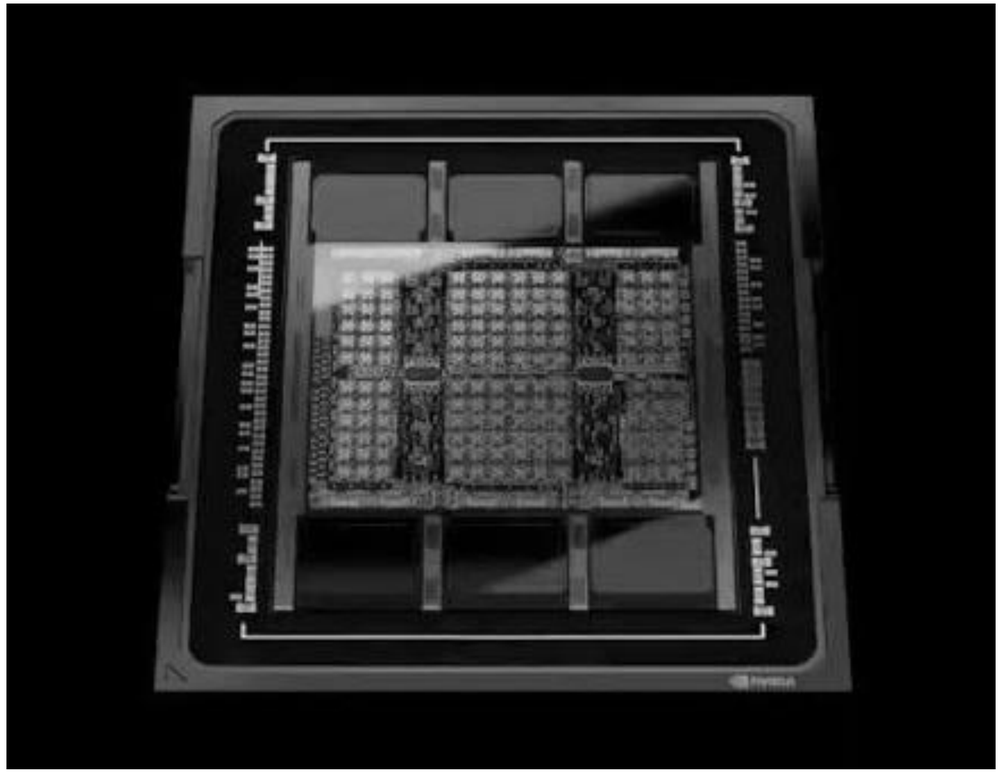
Example (d=2048, N=24, L=1024, B=8, h=16, V=50k):

- Parameters: ≈ 1.31B (≈ 2.4 GB in FP16, ≈ 21 GB with Adam states)
- Forward activations: ≈ 18.7 GB
- Backprop: ≈ 40–75 GB

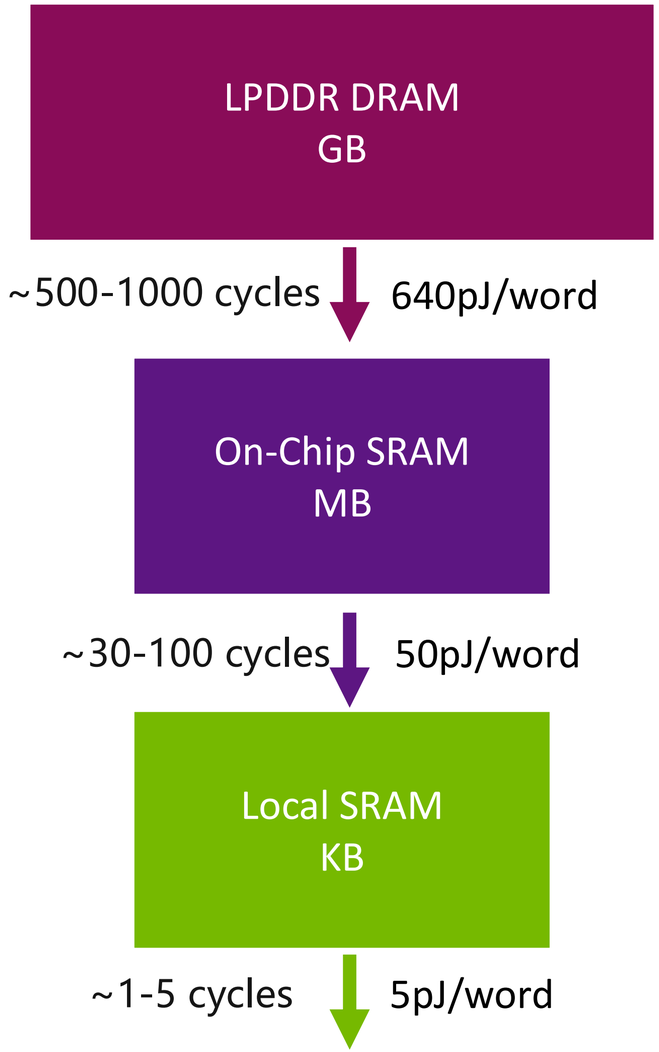
Lesson 3: Data processing is cheap, data storage is slow

–Tom Jerry



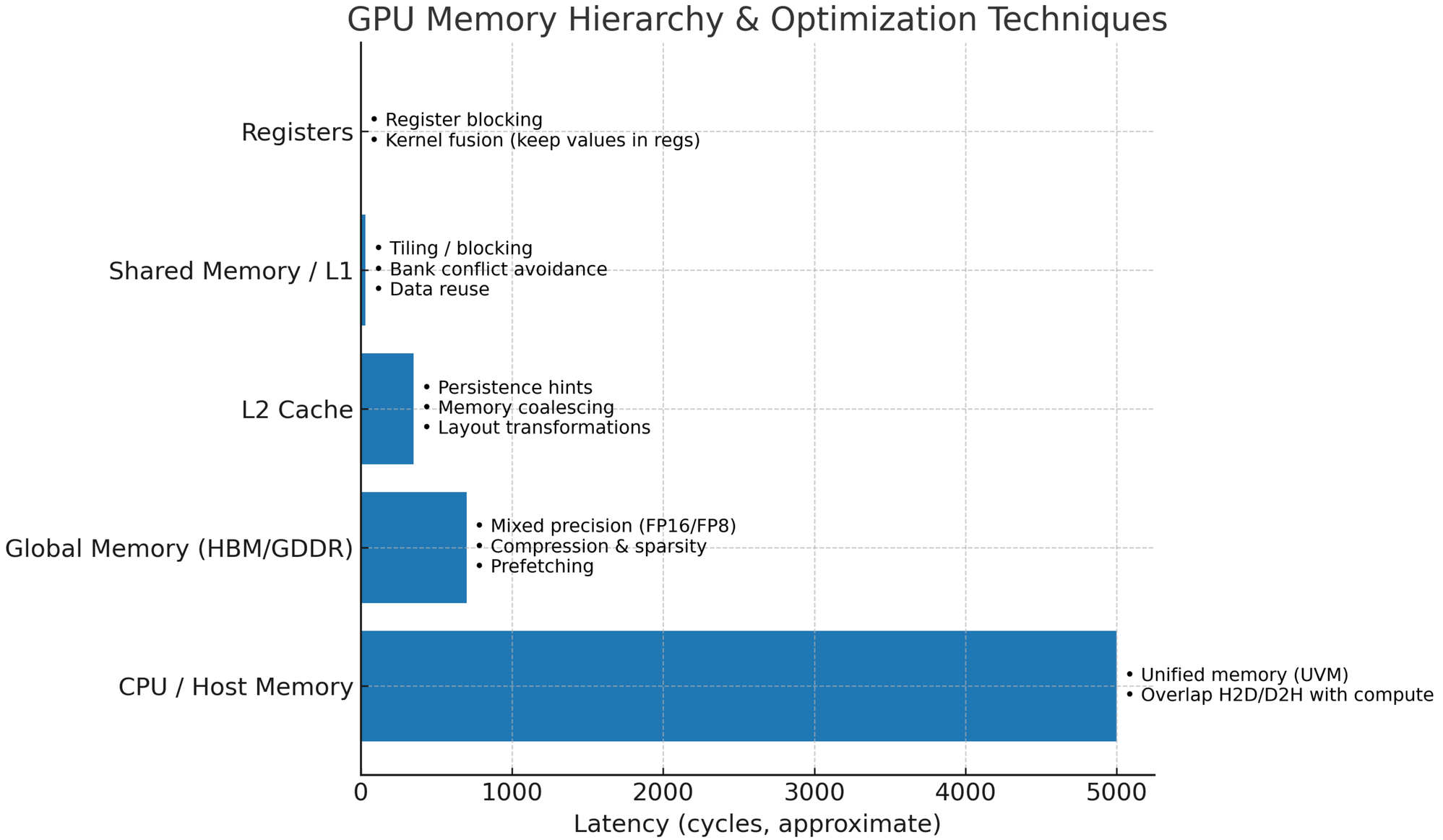


The Importance of Staying Local



Registers / shared memory (fast, small) → global DRAM (slow, huge)

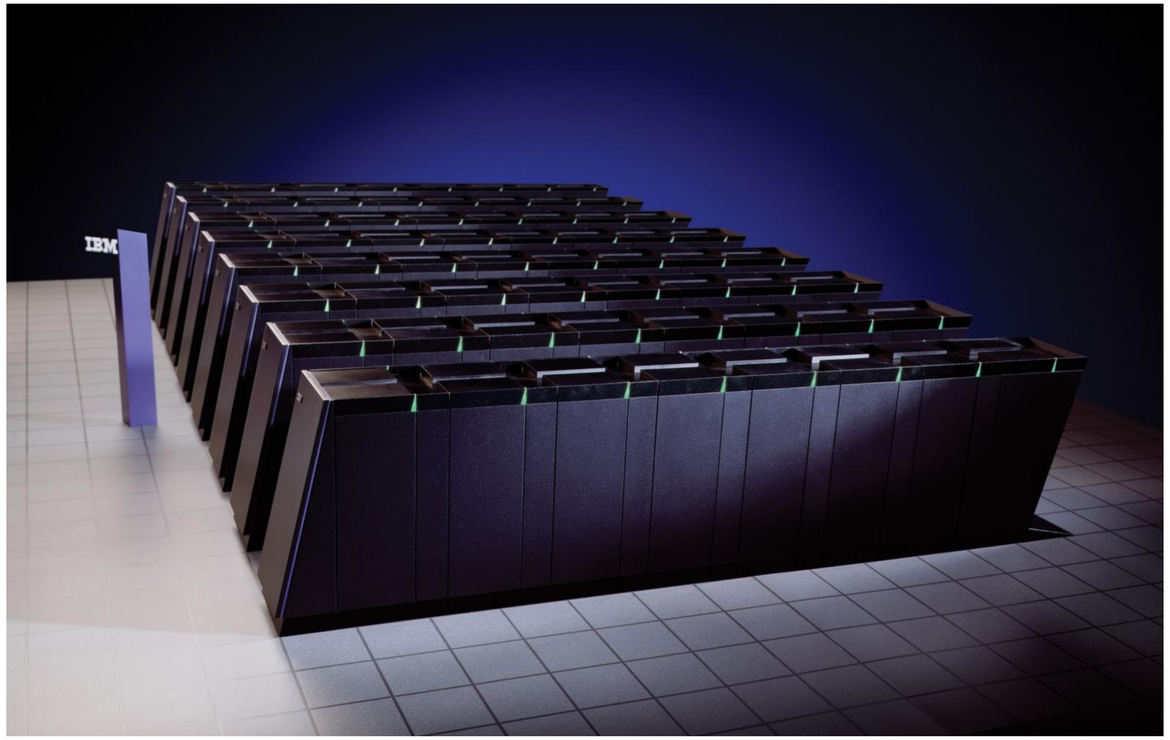
The energy to fetch a word from DRAM can equal **100–1000** MAC operations!

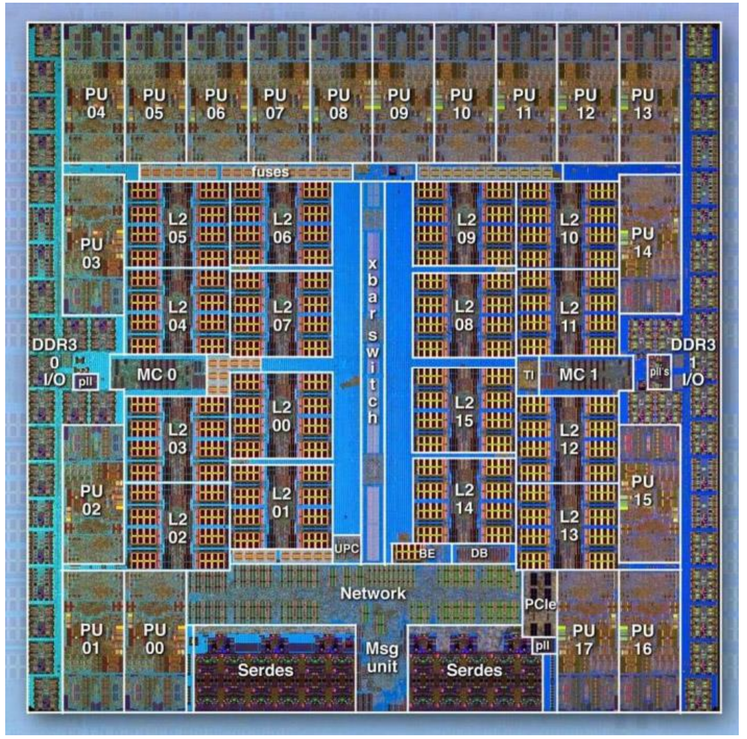


The GPU software stack aggressively tilizes locality, reuse, fusion, quantization, and overlap to minimize global memory traffic.

"*A supercomputer is a device for turning compute-bound problems into I/O-bound problems.*"

Ken Batcher*


IBM

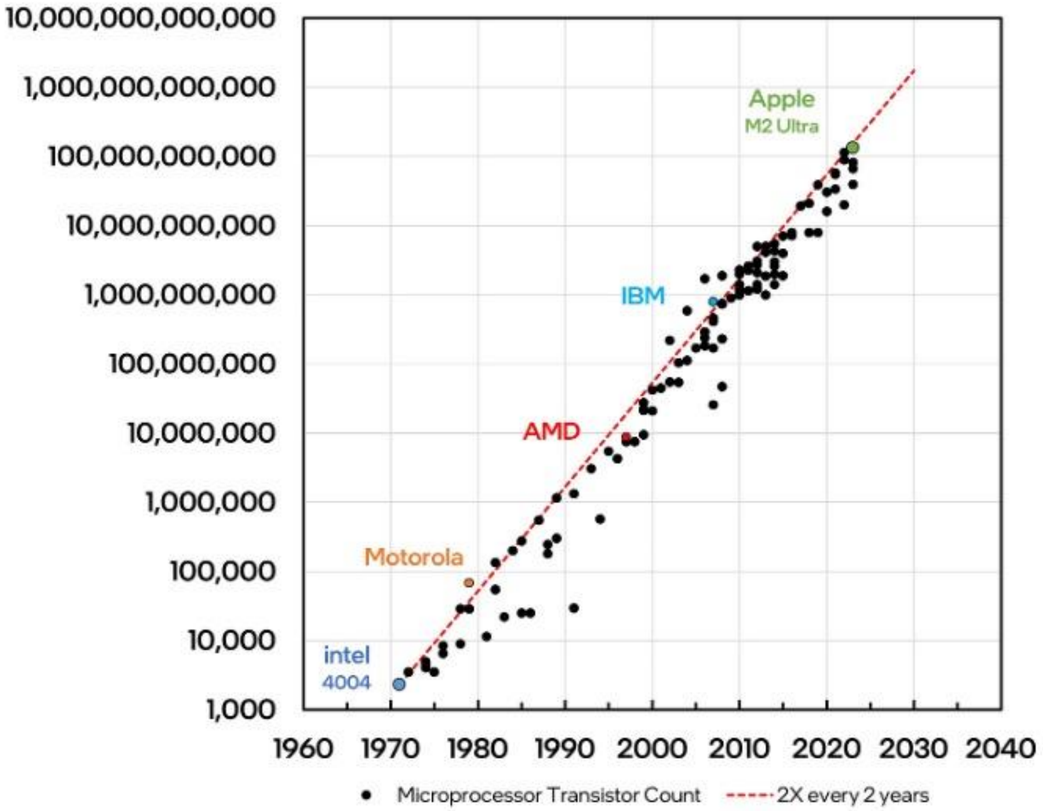

PU 04
PU 05
PU 06
PU 07
PU 08
PU 09
PU 10
PU 11
PU 12
PU 13
fuses
PU 03
L2 05
L2 06
L2 09
L2 10
PU 14
L2 04
L2 07
L2 08
L2 11
x b a r s w i t c h
DDR3 0 I/O
pll
MC 0
TI
MC 1
pll s
DDR3 1 I/O
L2 00
L2 15
L2 03
L2 12
PU 02
L2 01
L2 14
PU 15
L2 02
L2 13
UPC
BE
DB
Network
PCIe
pll
PU 01
PU 00
Serdes
Msg unit
Serdes
PU 17
PU 16

Lesson 4: Bending the laws of physics, is that possible?

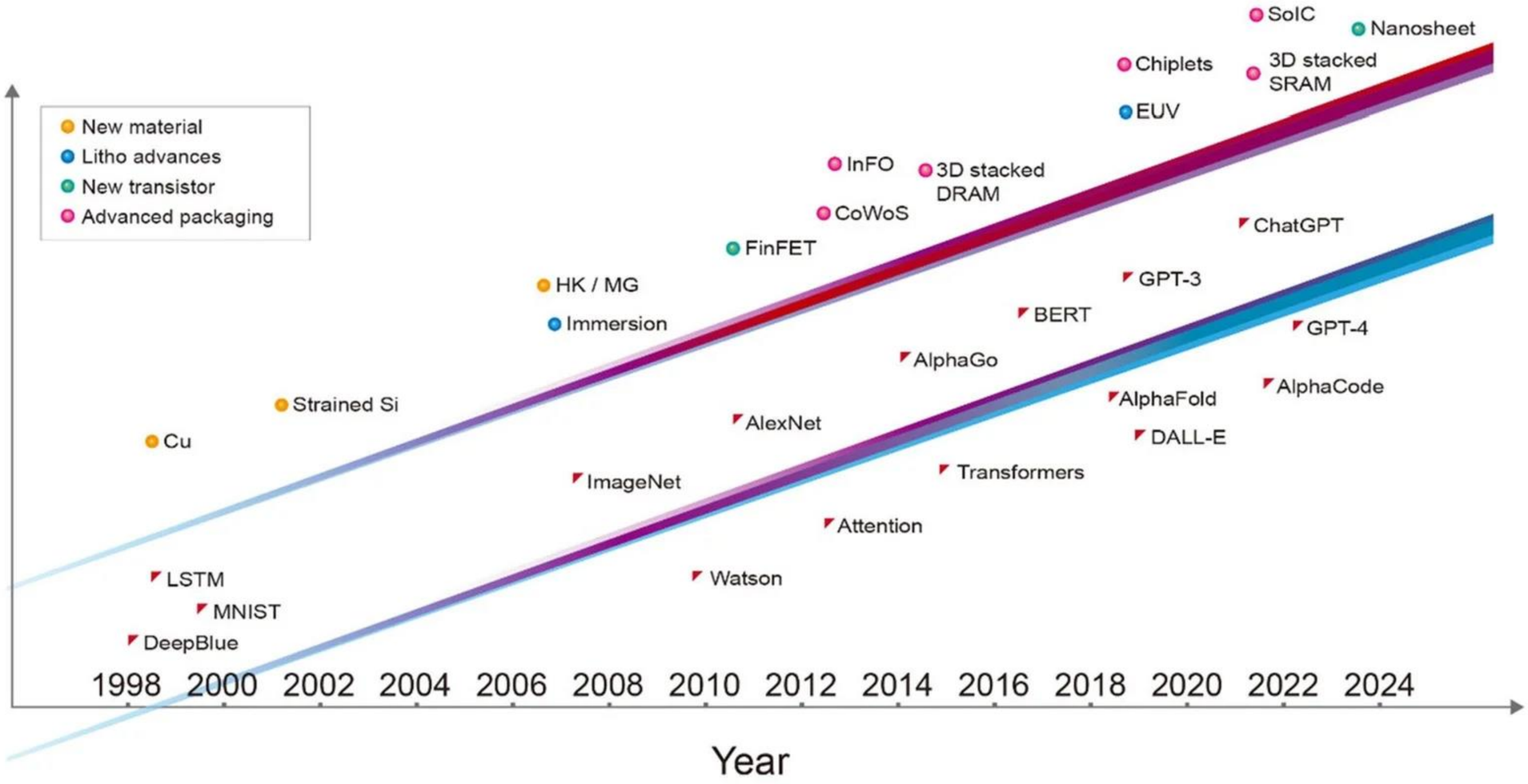
–Tom Jerry

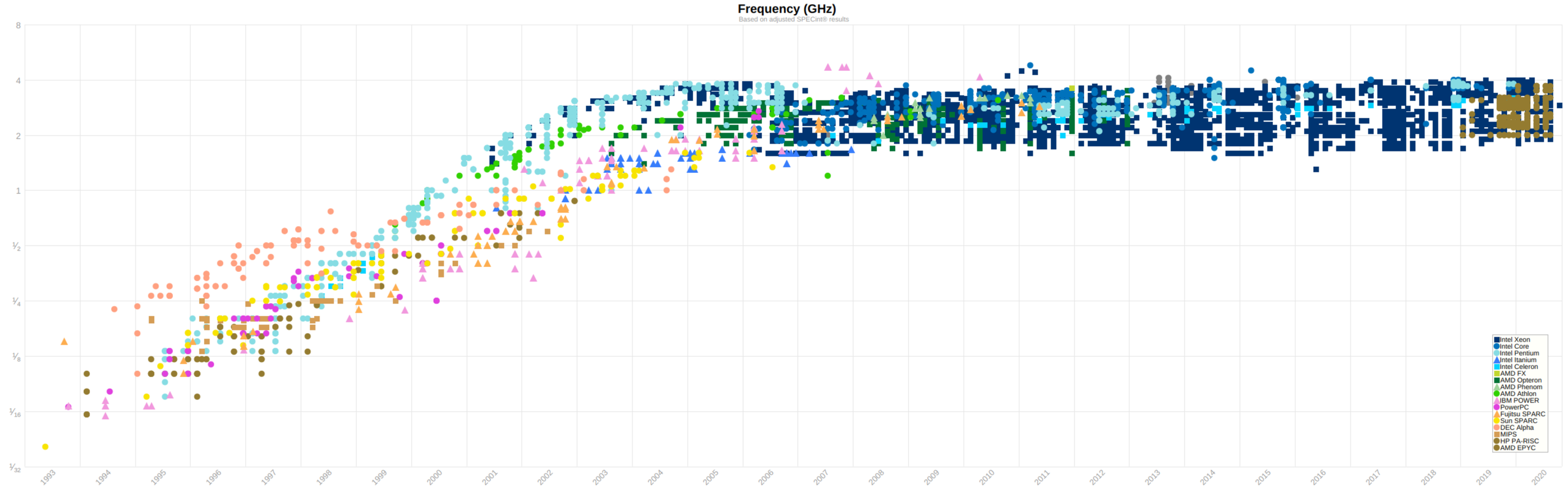
Modern Computing

- Transistors shrink, allowing more to fit on a silicon wafer, increasing computing power.
- Moore's Law predicts that microprocessor capabilities will double every 18-24 months.
- The trend has led to microprocessors with up to 100 billion transistors.



Modern Computing

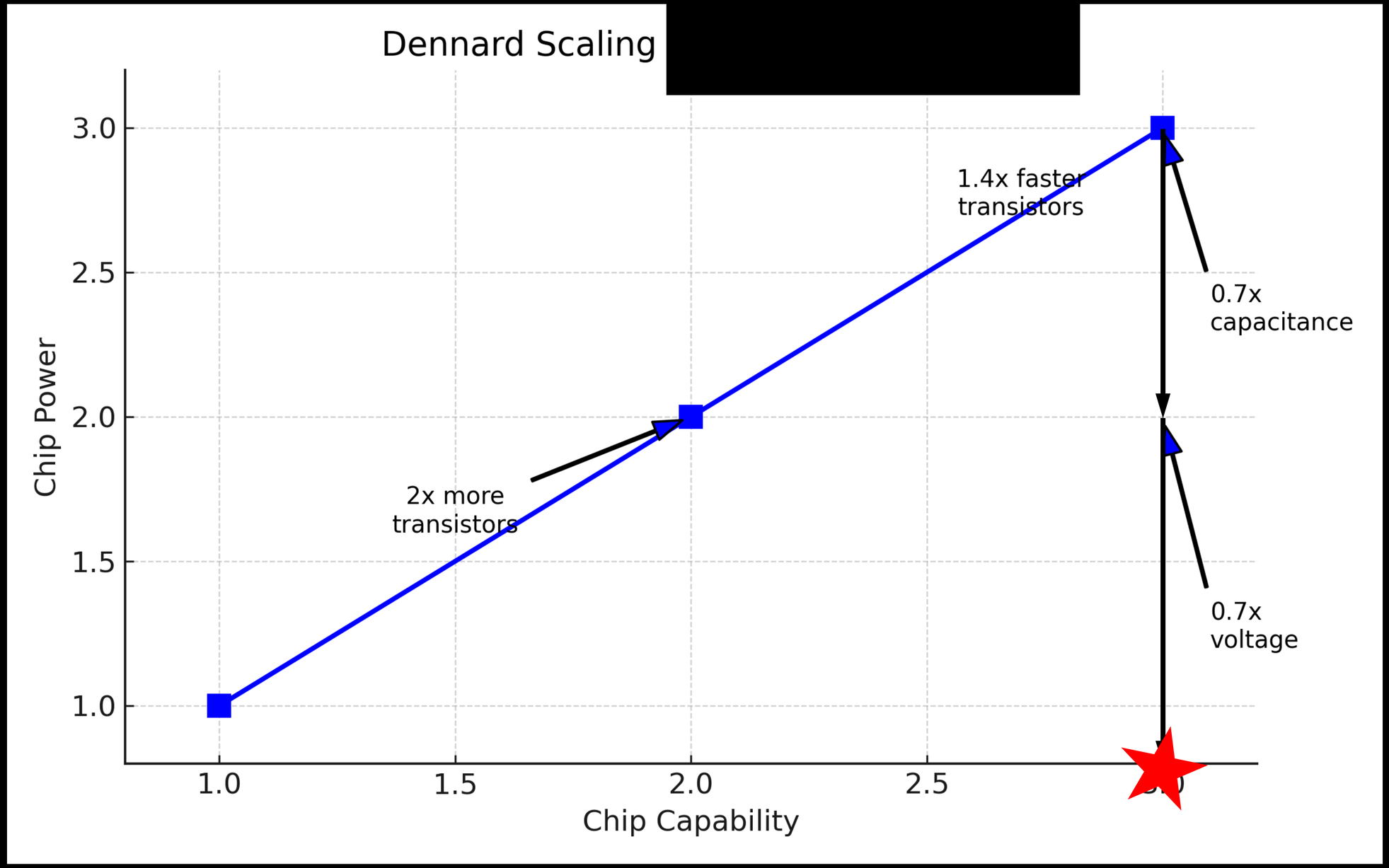




Transistors got exponentially faster until the moment they didn't.

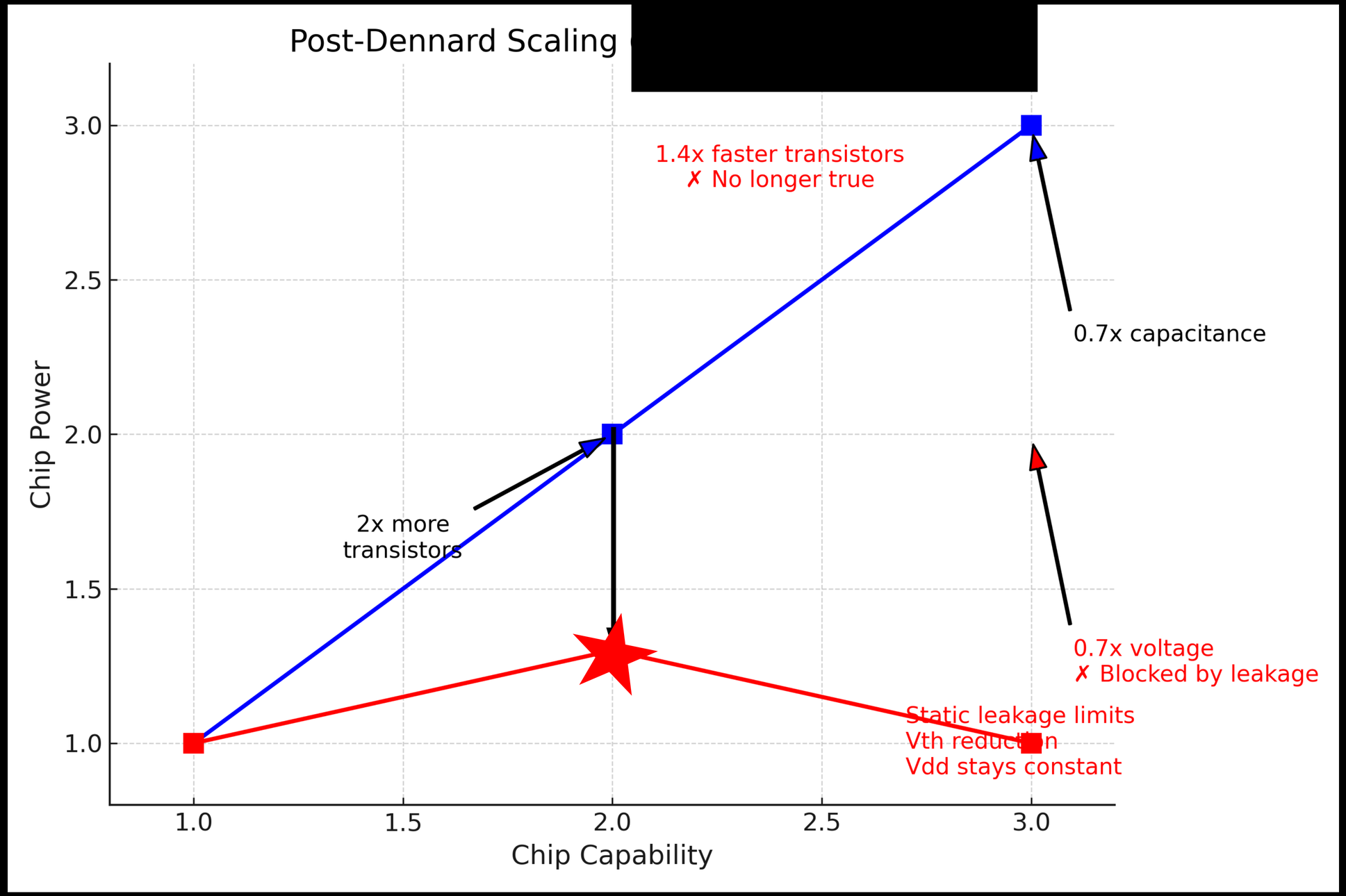
How does Dennard scaling work?

- The total power used by a chip for a given area stays the same
- The number of transistors doubles at a specific rate
- The frequency increases by a certain percentage every two years
- Both voltage and current scale down with length


Dennard Scaling
Chip Power
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
Chip Capability
1.4x faster
transistors
0.7x
capacitance
2x more
transistors
0.7x
voltage

Challenges

- However, Dennard scaling ignored "leakage current" and "threshold voltage", which establish a baseline of power per transistor.
- Leakage power became problematic around 2005.

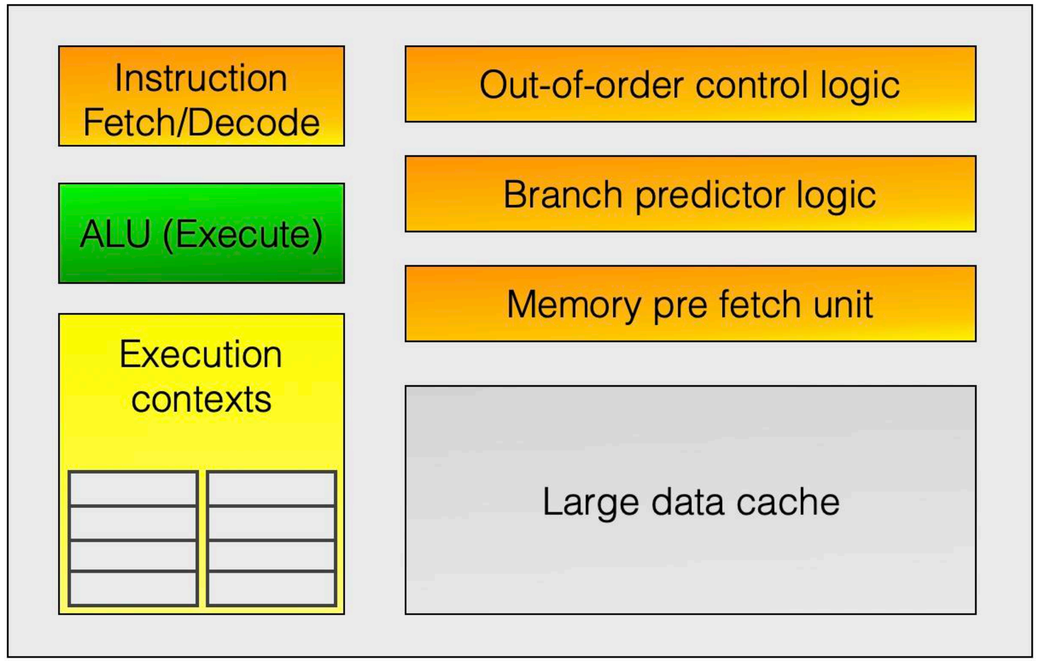

Post-Dennard Scaling
1.4x faster transistors
✗ No longer true
0.7x capacitance
2x more
transistors
0.7x voltage
✗ Blocked by leakage
Static leakage limits
Vth reduction
Vdd stays constant
Chip Power
Chip Capability
1.0
1.5
2.0
2.5
3.0

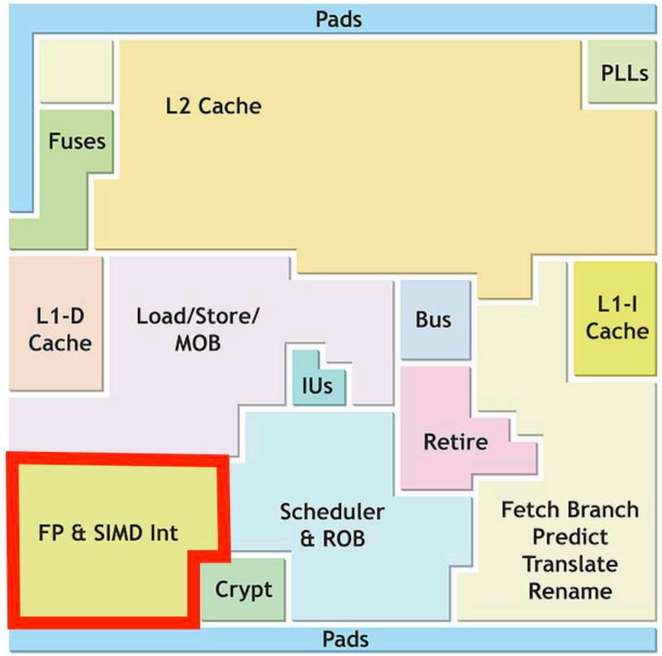
The alternative path is massive parallelism with many cores, doubles every two years, which however introduces the challenge to software development.

Lesson 5: Who sells memory better, CPU or GPU?

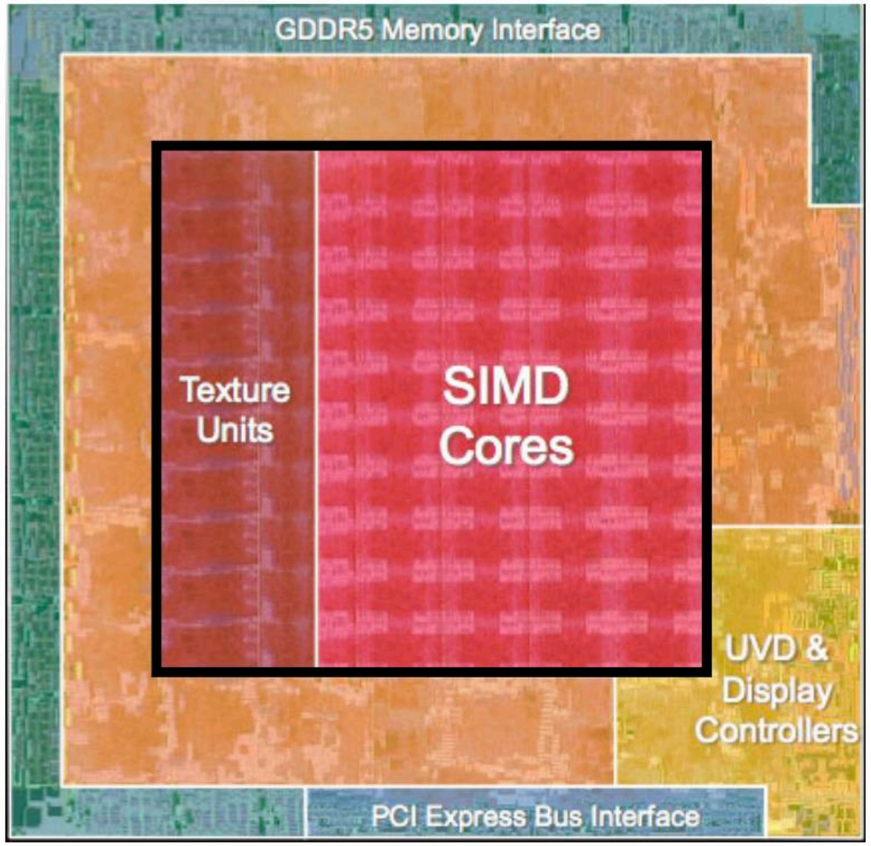
–Tom Jerry

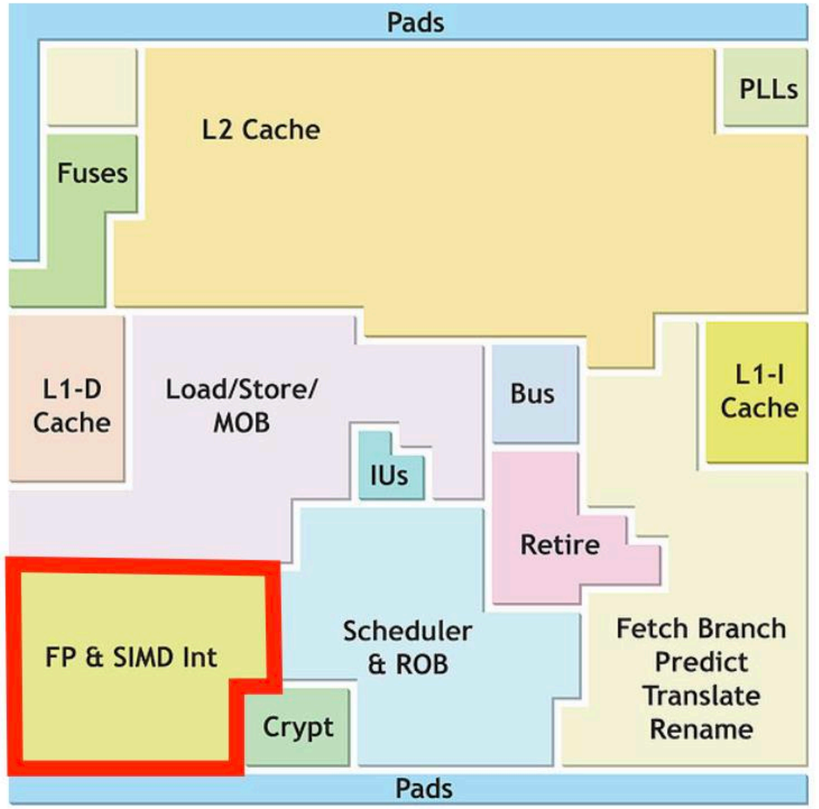
CPU: Battling for Single-Thread Performance in the Face of the Memory Wall



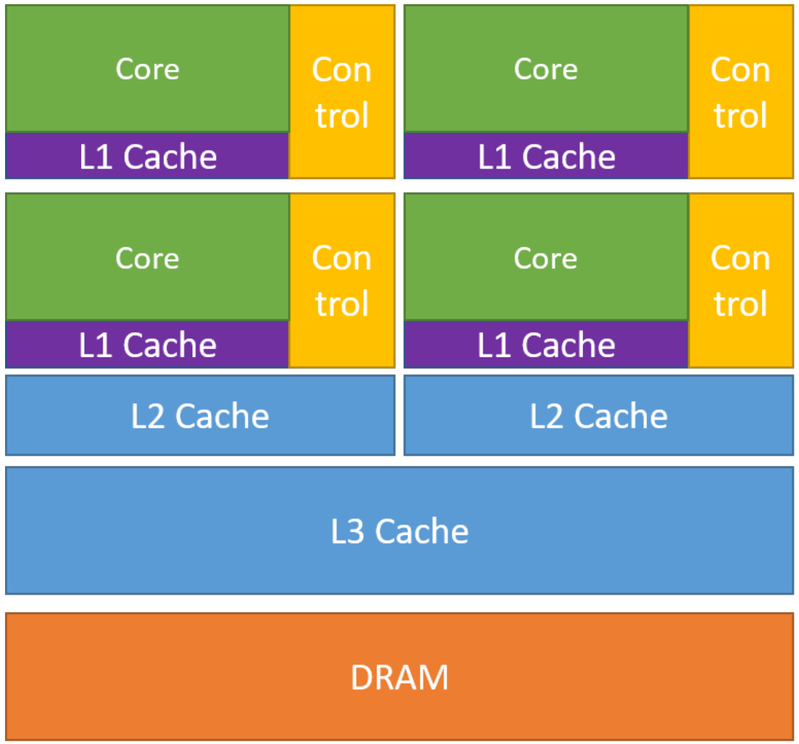


GPU: Defying Moore's Law through massive parallelism



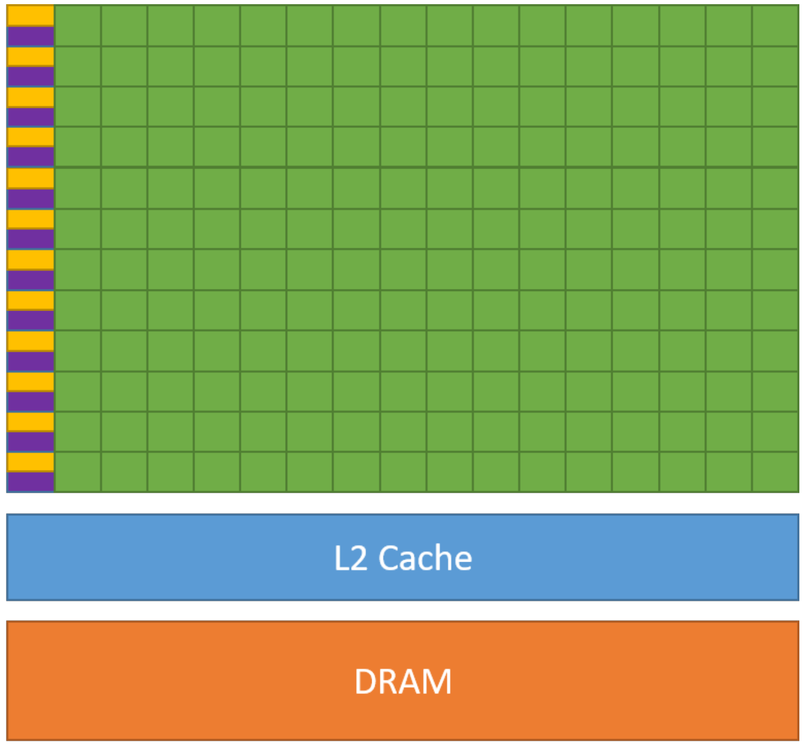


CPU vs. GPU



CPU

Less than 20 cores
1-2 threads per core
Latency is hidden by large cache



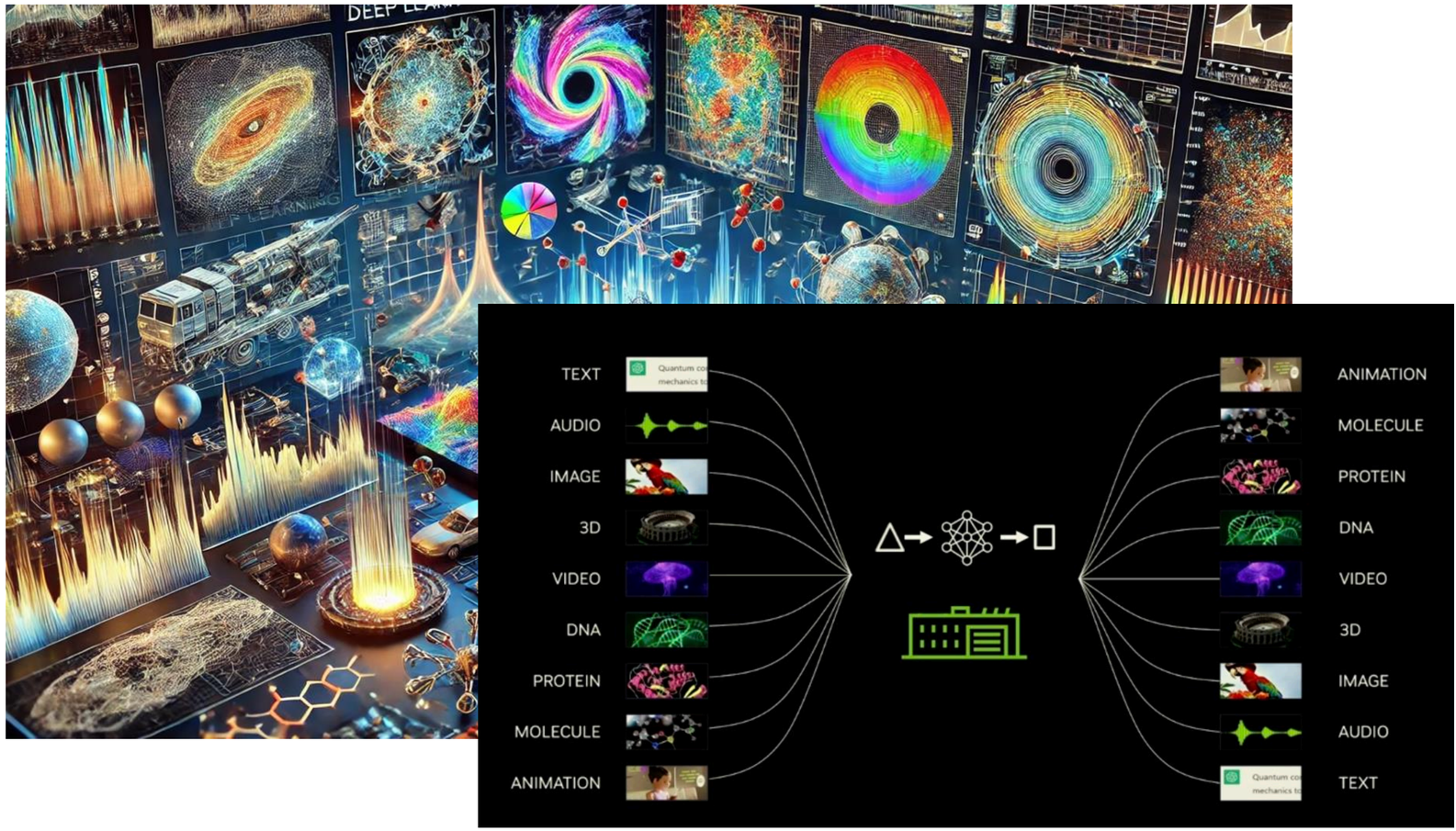
GPU

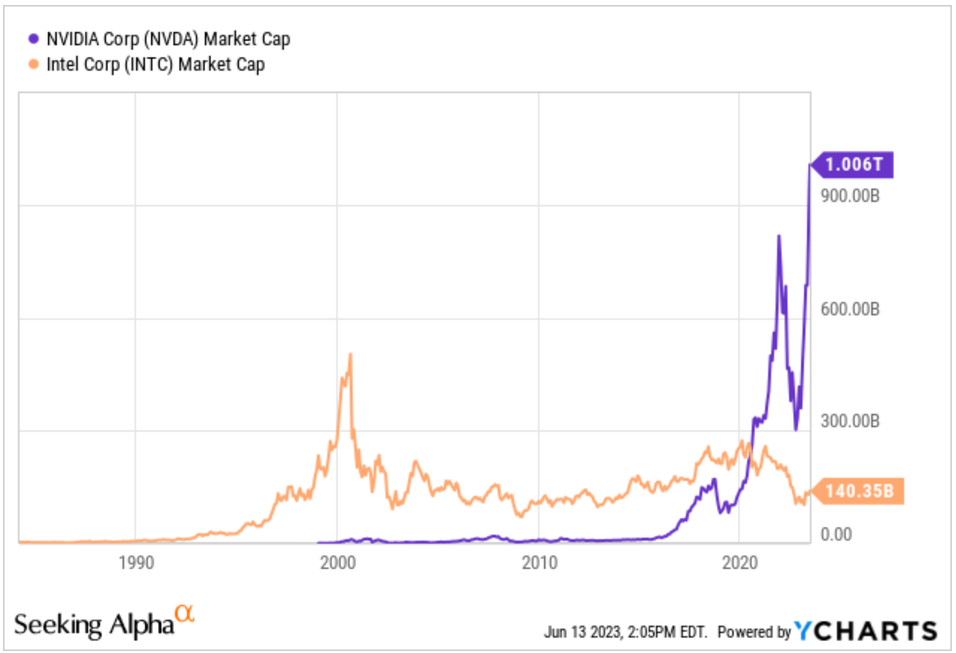
512 cores
10s to 100s of threads per core
Latency is hidden by fast context switching

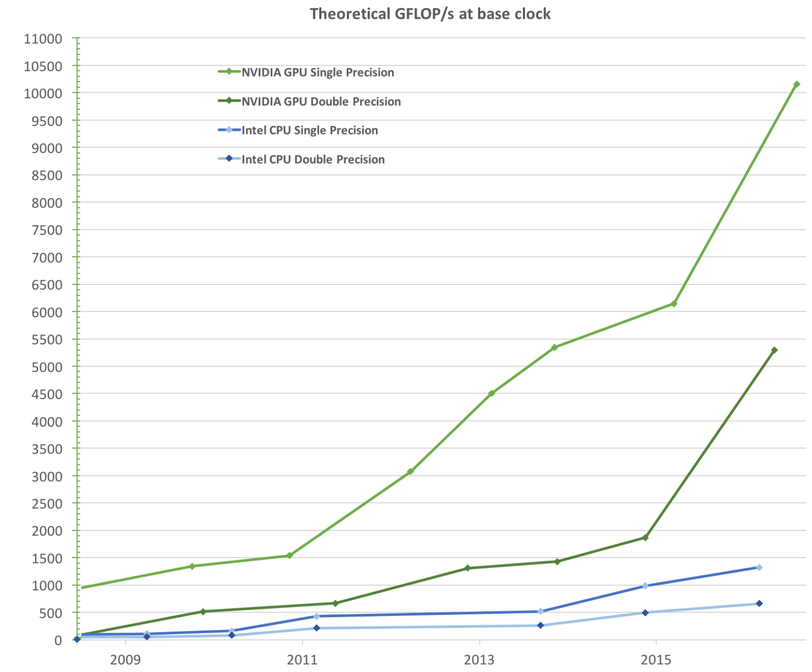
Modern GPU

- Deep Learning
- Gaming & big data
- Computational Fluid Dynamics
- Computational Structural Mechanics
- Seismic Processing
- Bioinformatics
- Materials Science
- Molecular Dynamics
- Quantum Chemistry
- Computational Physics

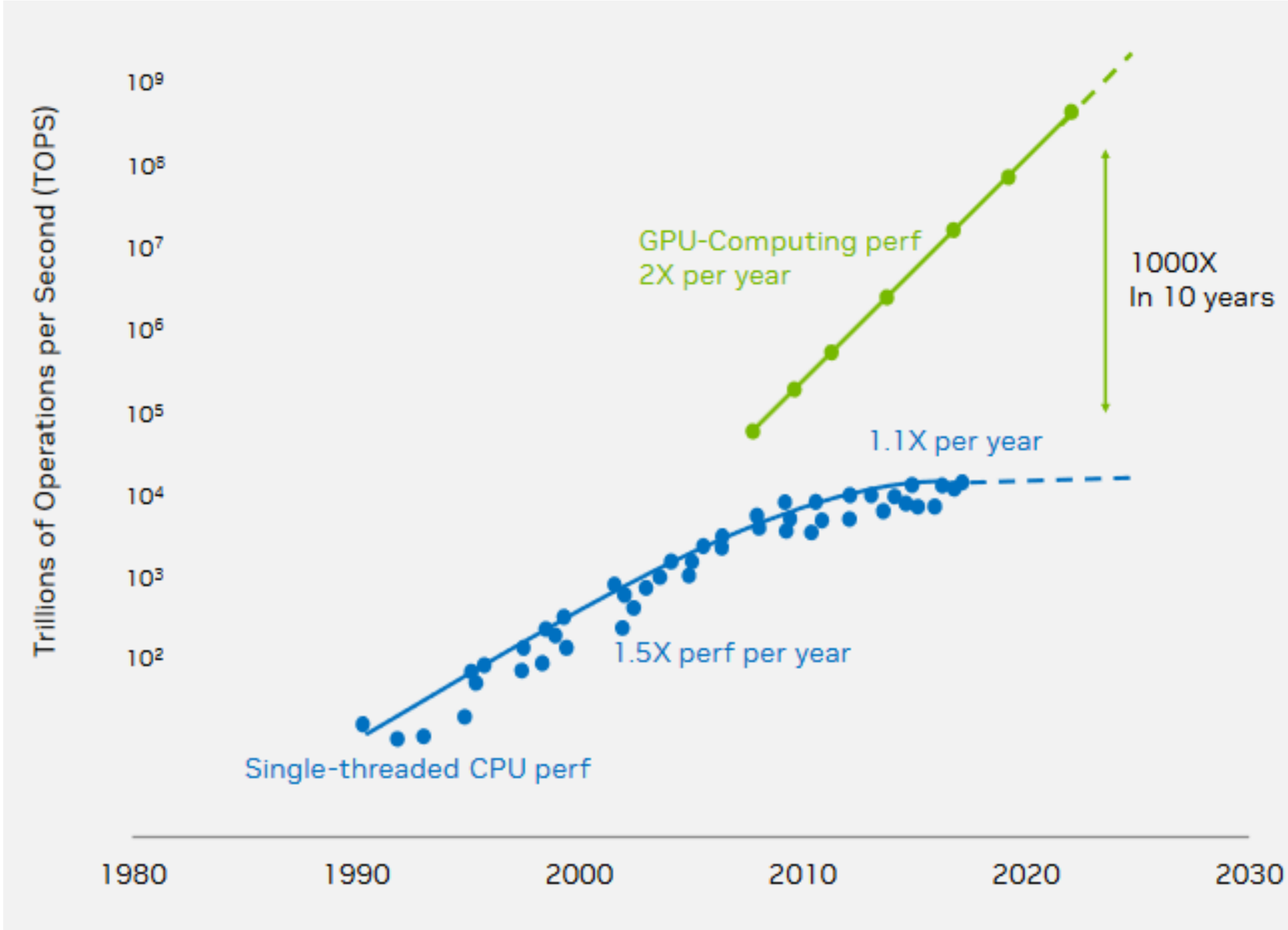
https://www.nvidia.com/en-us/accelerated-applications/







Defying Moore's Law

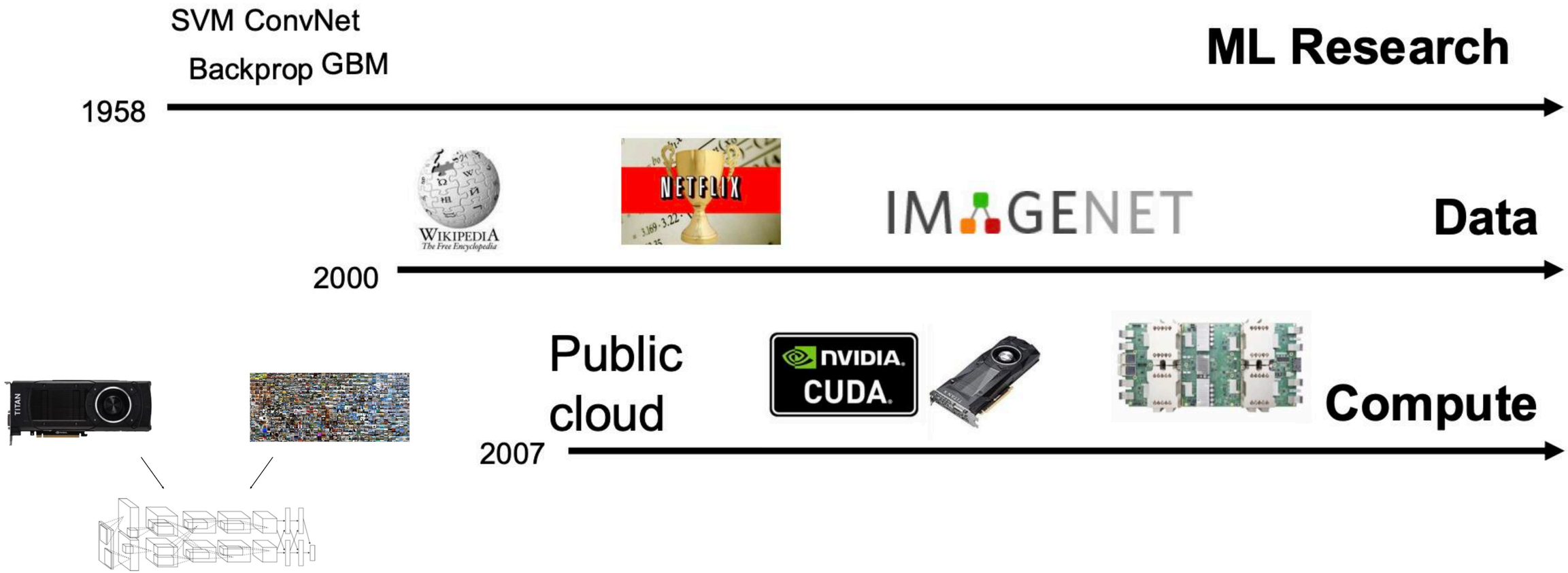


What is “Machine Learning Systems”

–Tom Jerry

Three Pillars of ML

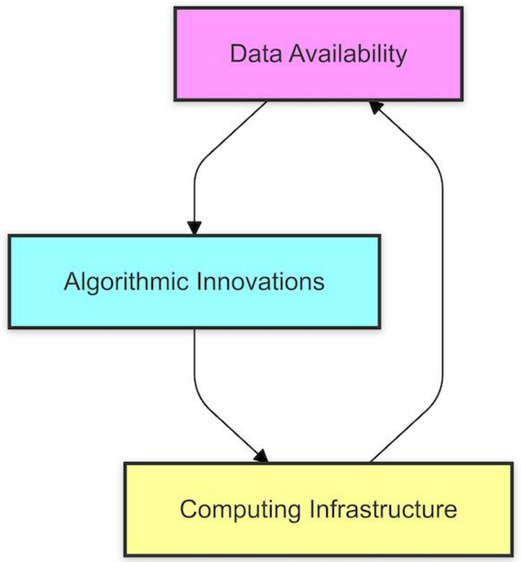
Decades long algorithmic research, big data, and scalable computing



Notable AI models

EPOCH AI

ML to DL



Training compute (FLOP)

1e26
1e24
1e22
1e20
1e18
1e16
1e14
1e12
1e10
1e8
1e6
1e4
1e2

1955 1960 1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015 2020 2025

Publication date

1.4x/year

2x/10month

1.4x/year

4.0x/year

Theseus

Perceptron Mark I

LSTM

System 11

Seq2Seq LSTM

GNMT

AlexNet

Word2Vec (large)

Swift

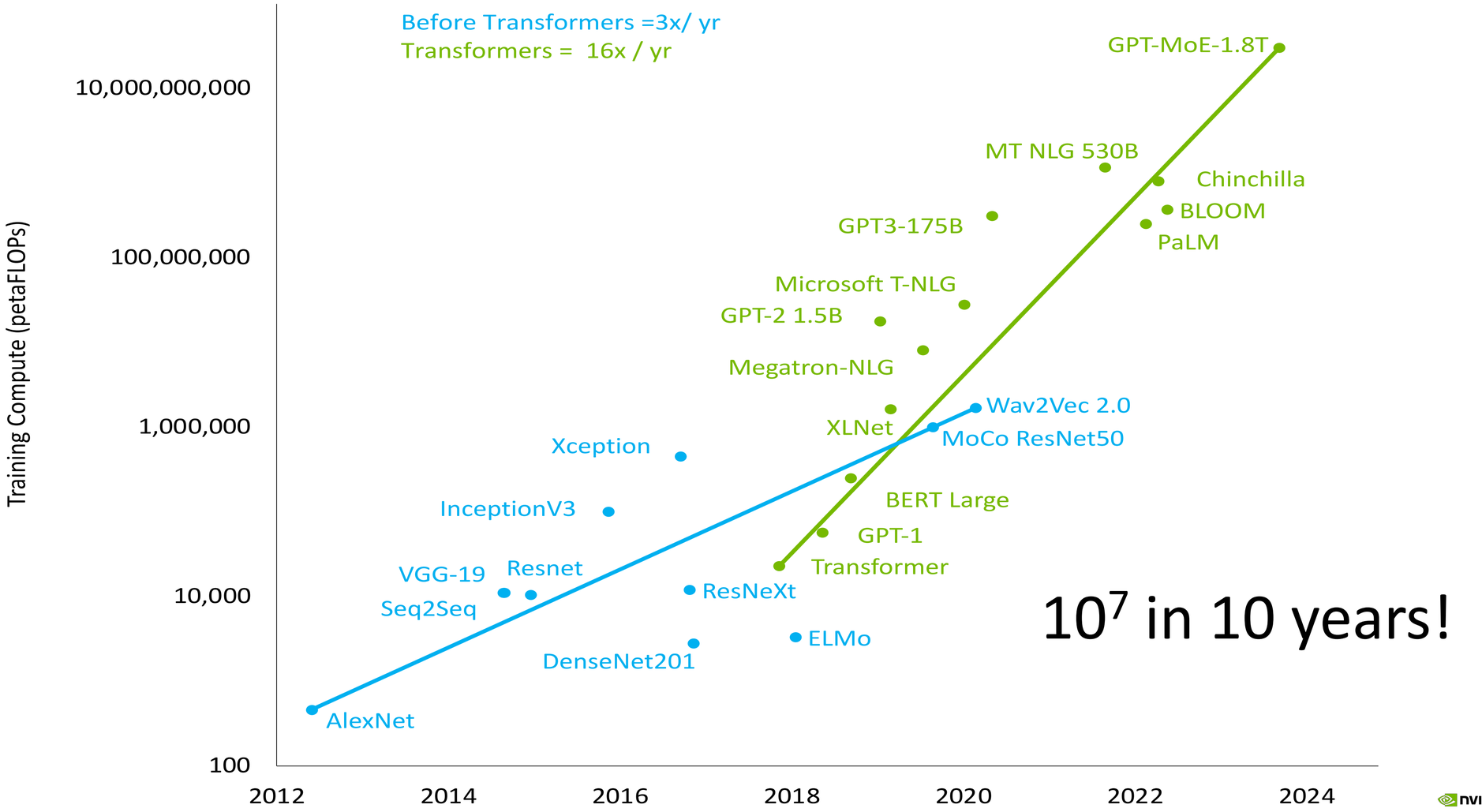
PaLM (540B)

Gemini 1.0 Ultra

Deep Learning Era

- Language
- Vision
- Biology
- Games
- Speech
- Image generatic
- Multimodal
- Robotics
- Video
- Other

Transformer: 10^7 demand



Computer Architect’s View

The computer architect’s role is to bridge the gap between hardware constraints and the computational needs of modern AI workloads by developing architectures that maximize efficiency, scalability, and performance.

- **Specialized AI accelerators:** Optimizing GPUs, TPUs, NPUs for ML computation.
- **Memory hierarchy design:** Improving HBM, cache optimizations, SRAM/DRAM bandwidth to handle the massive data movement required for ML training.
- **Interconnect innovations:** Enhancing NVLink, PCIe, InfiniBand, NVSwitch to scale large distributed training workloads across thousands of GPUs.
- **Power and efficiency trade-offs:** Designing low-power AI chips while maintaining high throughput for inference and training.

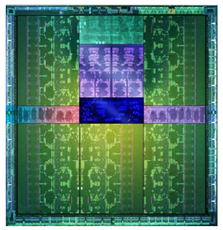
Single-Chip Inference Performance - 1000X in 10 years

B200
20,000

Int 8 TOPS

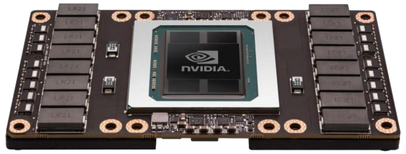
4500.00
4000.00
3500.00
3000.00
2500.00
2000.00
1500.00
1000.00
500.00
0.00

4/1/12
8/14/13
12/27/14
5/10/16
9/22/17
2/4/19
6/18/20
10/31/21
3/15/23



FP32
FMA

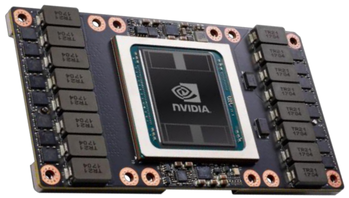
K20X
3.94



FP16
HDP4

M40
6.84

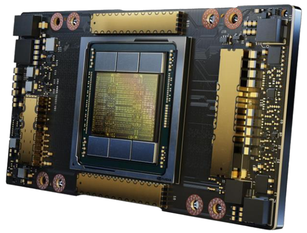
P100
21.20



FP16
HMMA

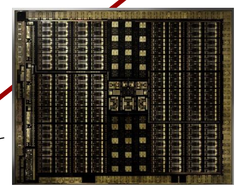
V100
125.00

Q8000
261.00

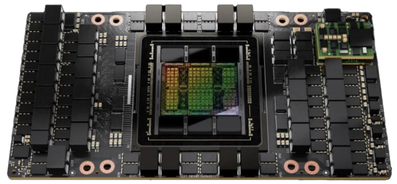


Sparsity

A100
1248.00



INT8
IMMA

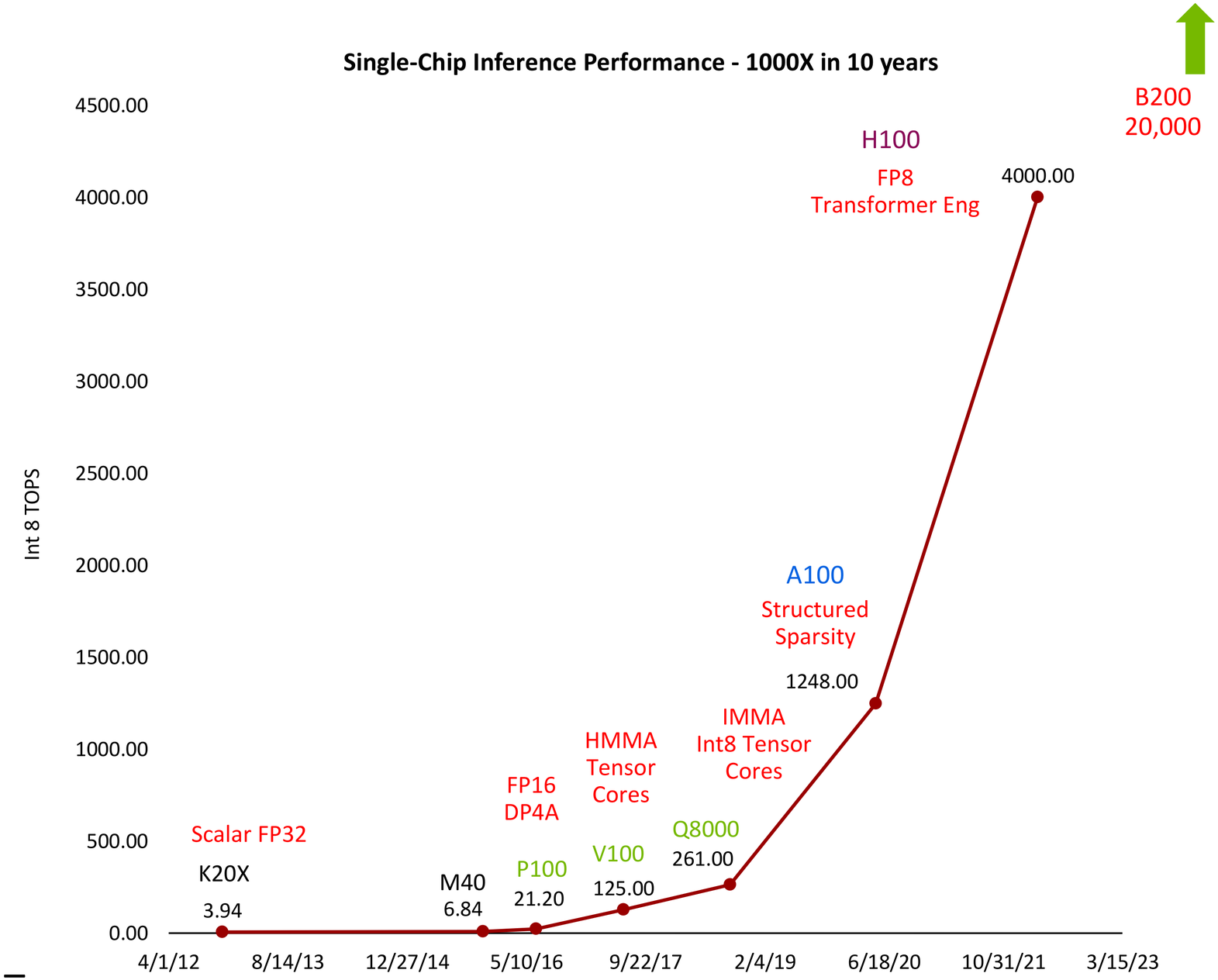


H100
4000.00

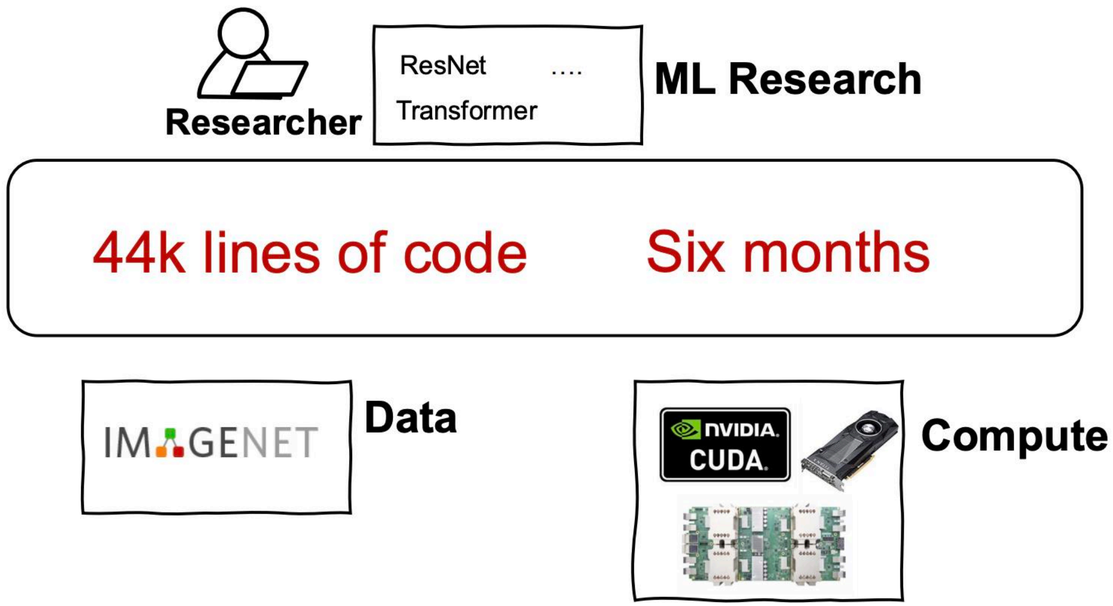
NVIDIA

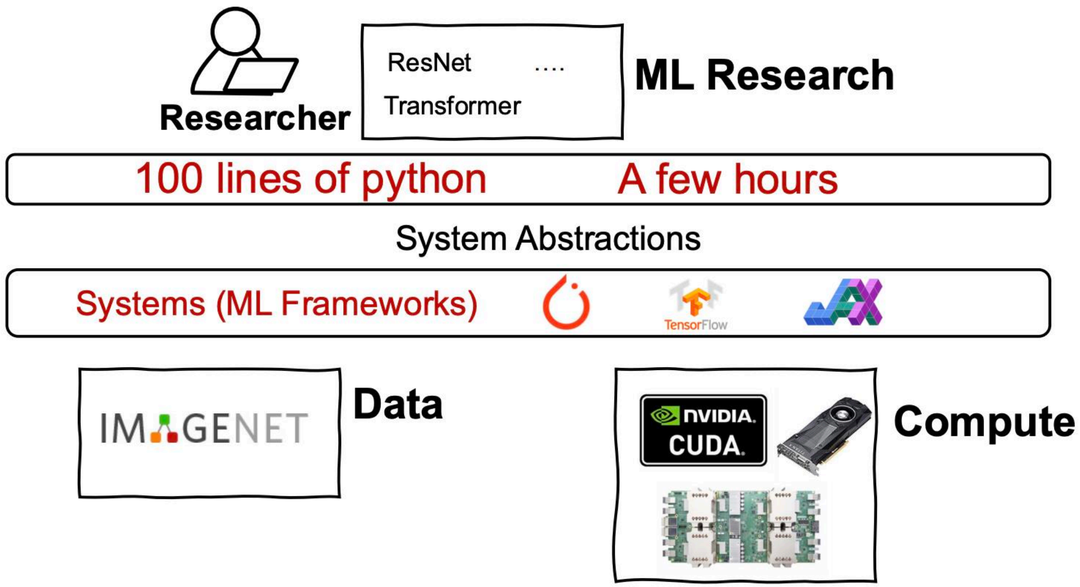
Gains from

- Number Representation
  - FP32, FP16, Int8, FP4
  - (TF32, BF16)
  - ~16x, 32x
- Complex Instructions
  - DP4, HMMA, IMMA
  - ~12.5x
- Process
  - 28nm, 16nm, 7nm, 5nm, 4nm
  - ~2.5x, 3x
- Sparsity ~2x
- Die Size 2x
- Model efficiency has also improved – overall gain > 1000x

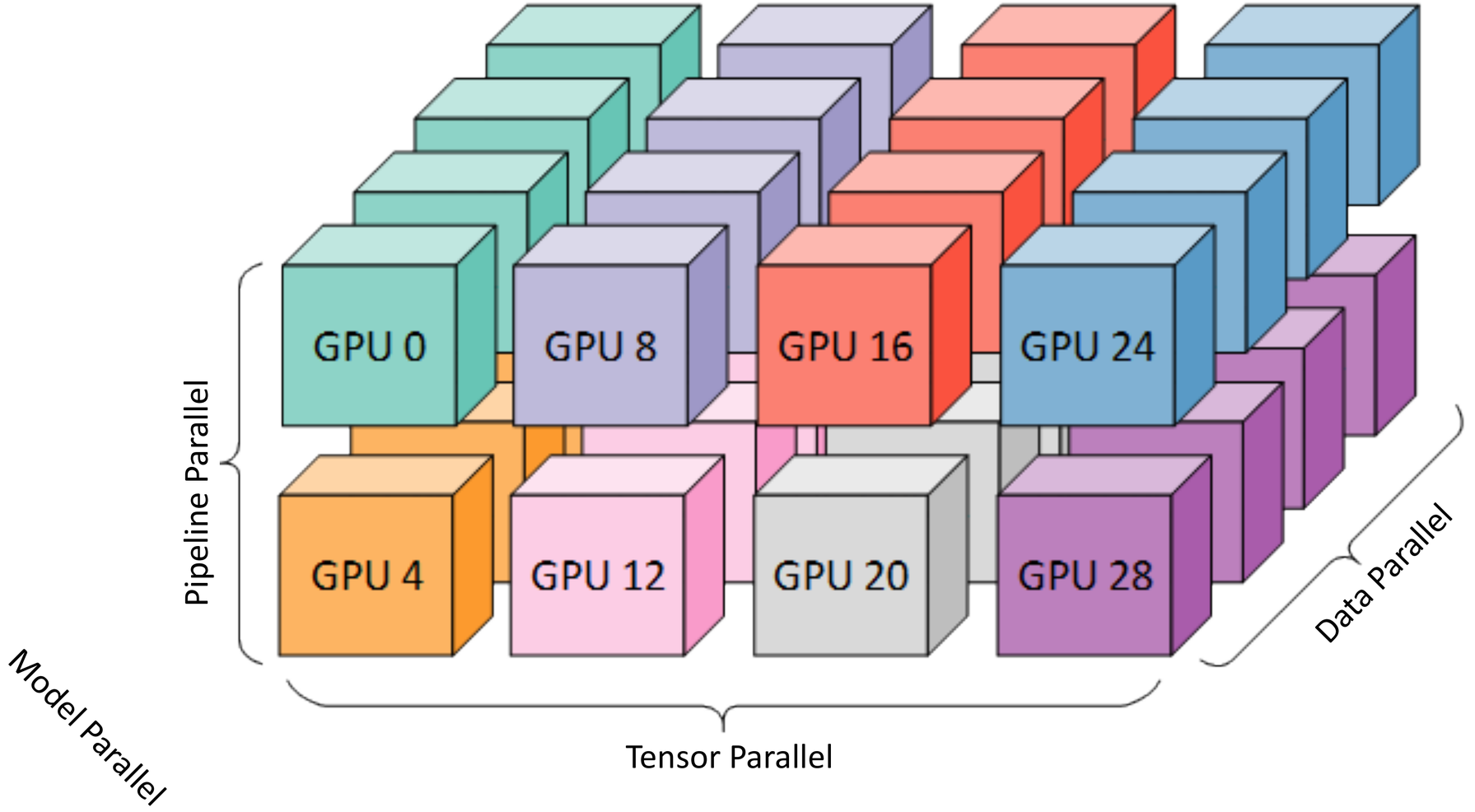


Model Developer's View





The other 1000x



It takes 20 GPUs to hold one copy of GPT4 model parameters

Algorithm Designer’s View

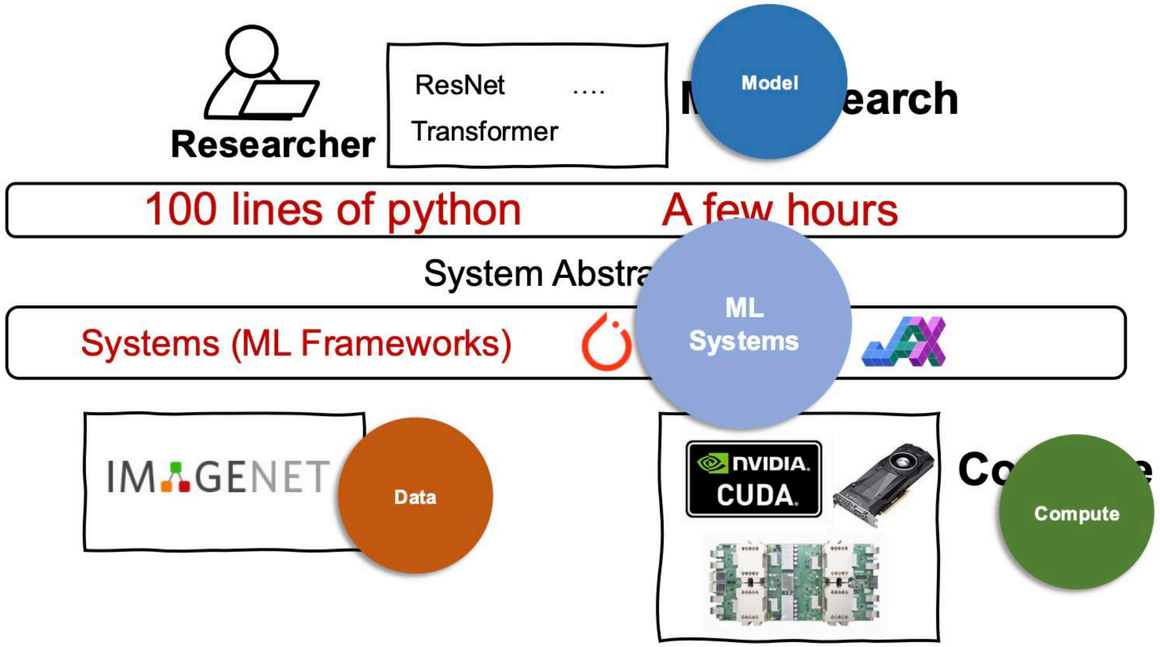
Machine Learning Algorithm Designer’s View: How to design ML algorithms that are not only accurate and intelligent but also computationally efficient.

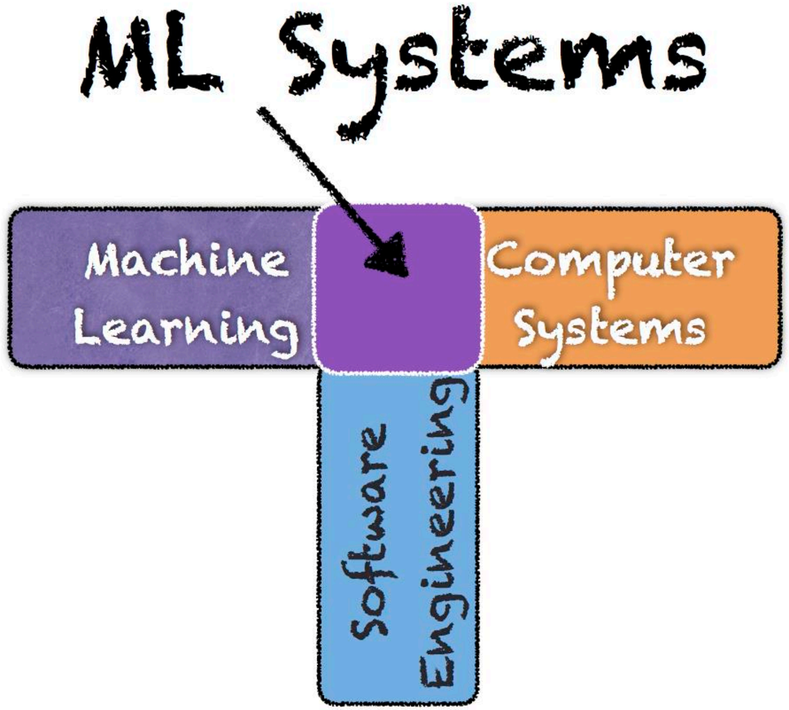
This includes:

- **Model efficiency techniques:** Quantization, pruning, knowledge distillation to reduce compute/memory footprint.
- **Data pipeline:** Techniques that cleans and tokenizes text, packs and shards it into optimized formats, streams it reliably, and monitors throughput to avoid wasted compute.
- **Training and inference optimization:** Techniques like parallel training, mixed-precision training, and batch scheduling to improve execution on real hardware.
- **Hardware-aware algorithm design:** Creating ML models that can efficiently run on modern architectures without excessive resource consumption.

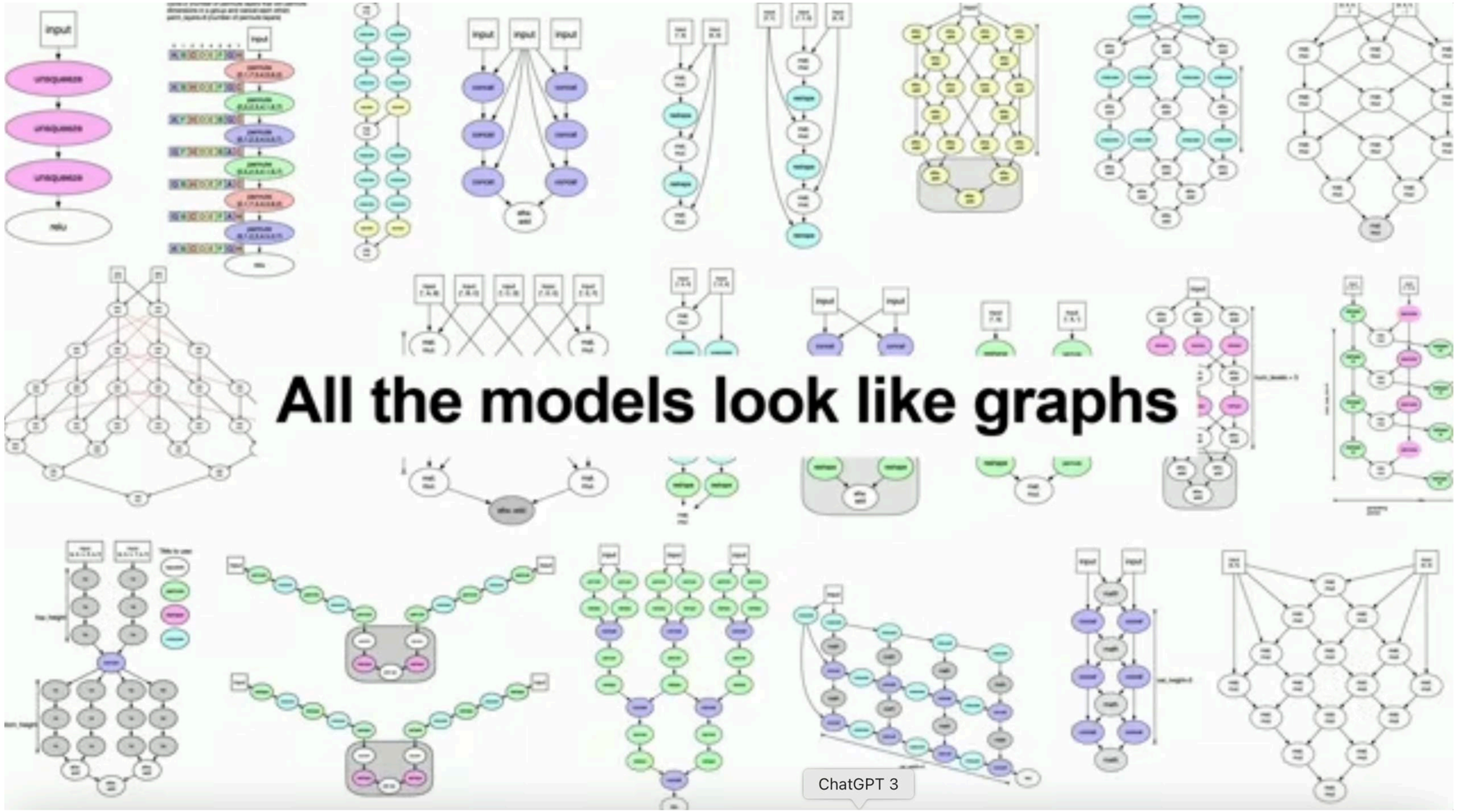
A Holistic View of Machine Learning Systems

Bridging the theoretical principles and practical implementation

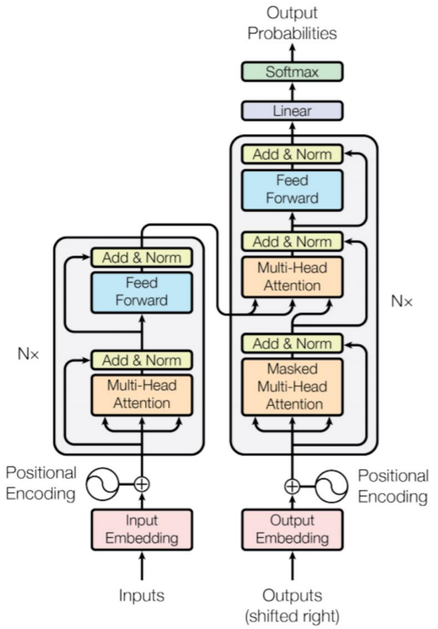


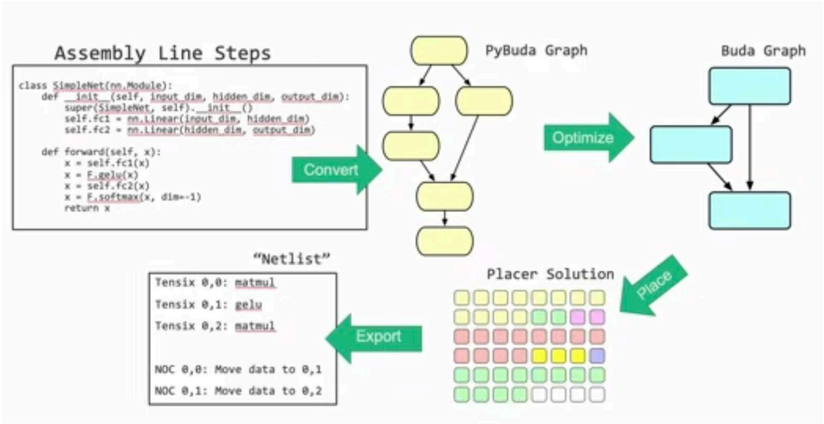


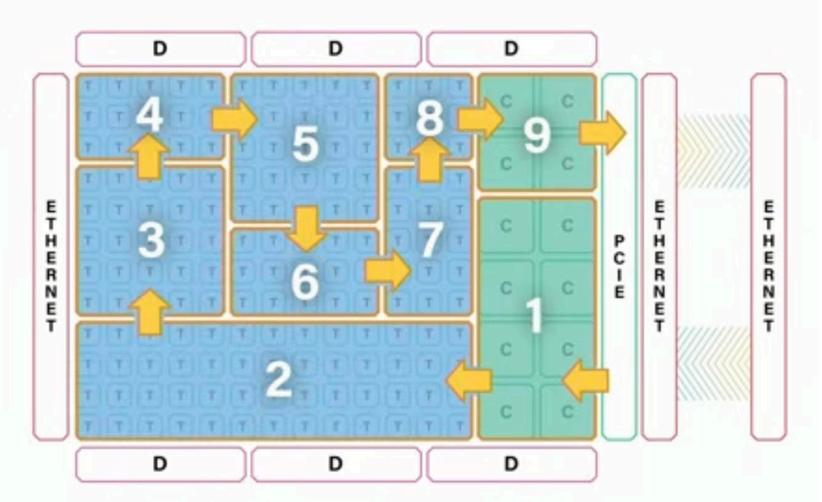
Instead of treating ML as separate layers (algorithms, software, and hardware), we must view it as a vertically integrated stack.


All the models look like graphs
ChatGPT 3

Machine Learning Systems as Graph Computing







Computation graph

Graph compiler

Graph mapping

Machine Learning Systems

ML Sys. is the discipline of designing, implementing, and operating artificially intelligent systems across computing scales—from resource-constrained embedded devices to warehouse-scale computers.

It focuses on algorithm-software-hardware co-design to create systems that are efficient, scalable, and reliable for their deployment context.

It encompasses the complete lifecycle of AI applications: from requirements engineering and data collection through model development, system integration, deployment, monitoring, and maintenance.

Course Logistics

Time: Monday 9:30-12:15

Location: B401

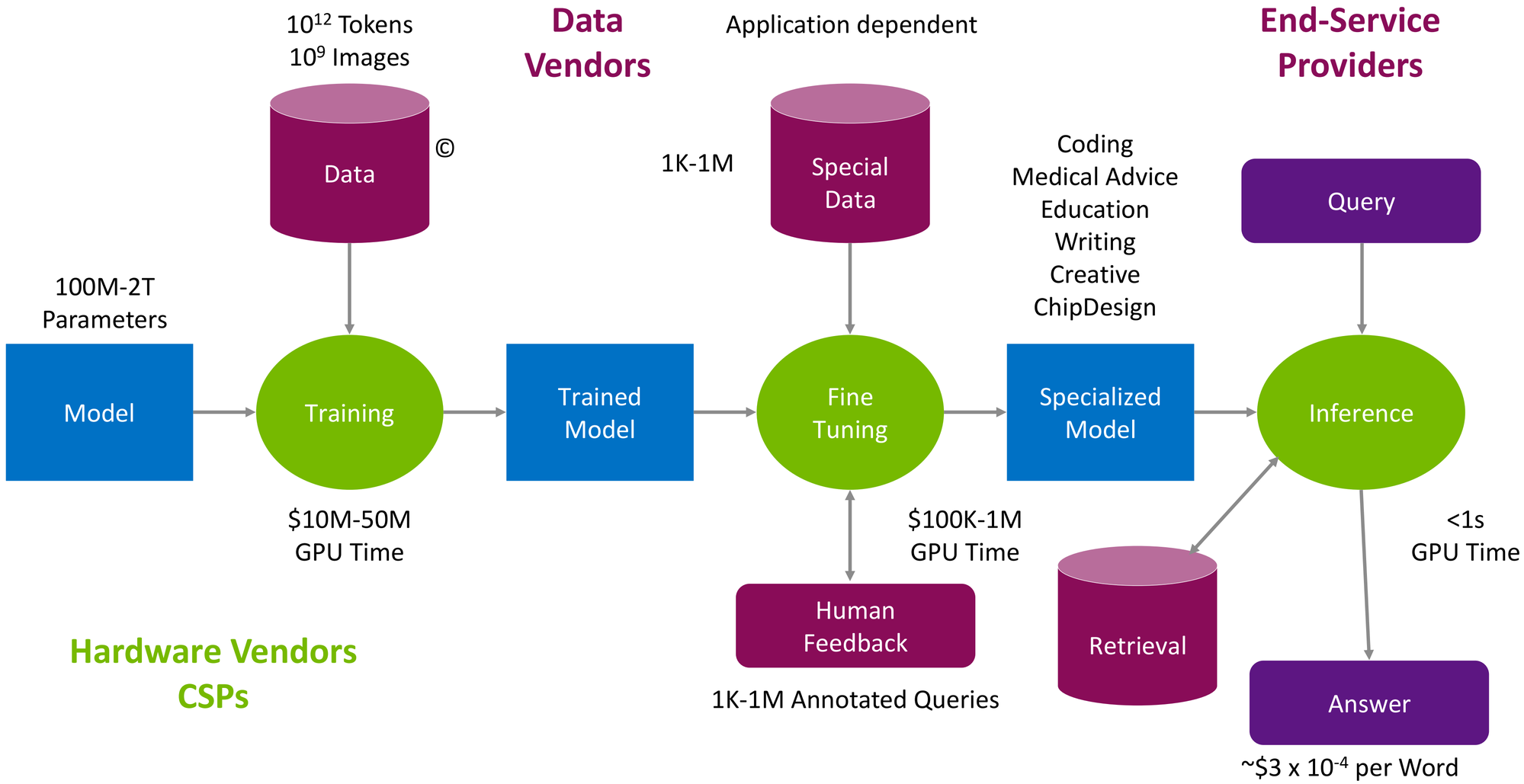
Instructor: Li Shang, Yuedong Xu

Course components: Lecture & guest lecture & research presentations

Course evaluation: class participation & discussion 25%, homework & paper reviews 25%, course project 50%

DL Lifecycle

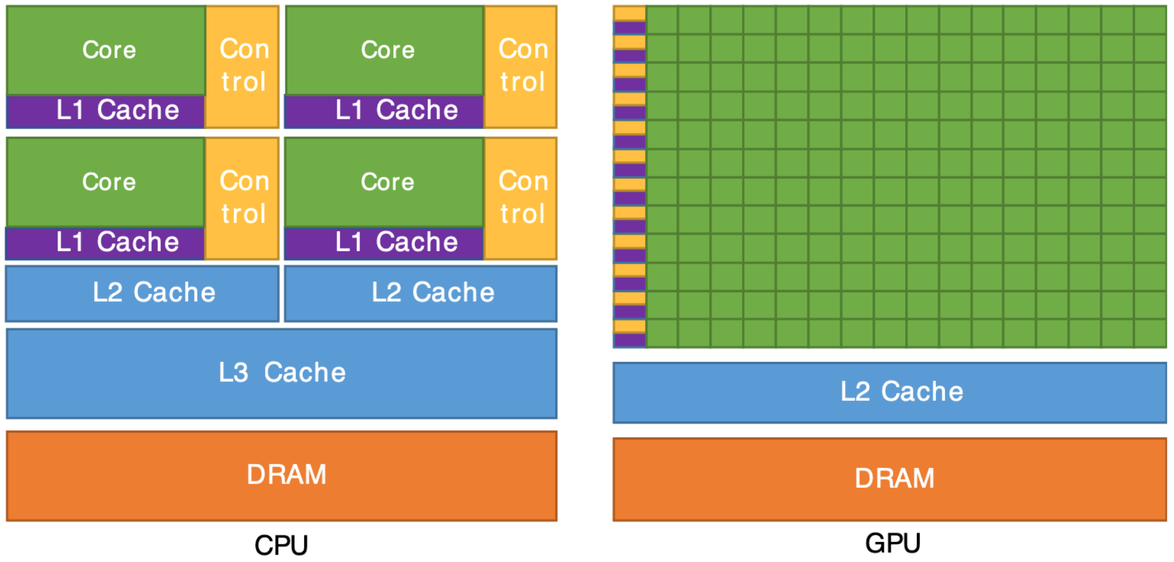
Data flow, model training and deployment



Schedule		Topics
1	Introduction	ML Sys Intro.
2-3	GPU Architecture	Historical overview, GPU architecture Advanced features
4-5	CUDA Programming	Fundamentals and architecture Management optimization and debugging Hands-on CUDA programming
6	ML Compilation	DL compiler LLM compiler
7-9	Parallel Training	LLM basics Parallelism Computation communication memory optimization
10-11	Inference	LLM inference infra
11-12	Model Optimization	Stability, Quantization Pruning & Sparsification
13	Data	Data engineering
14	AI Acceleration	AI accelerator Edge computing
15-16	Invited talks	Industry experts

GPU Architecture

- GPU: Architectural fundamentals
- Modern GPU architecture and deep learning
- Optimization Strategies in GPU Computing
- Advanced GPU Features and Future Directions



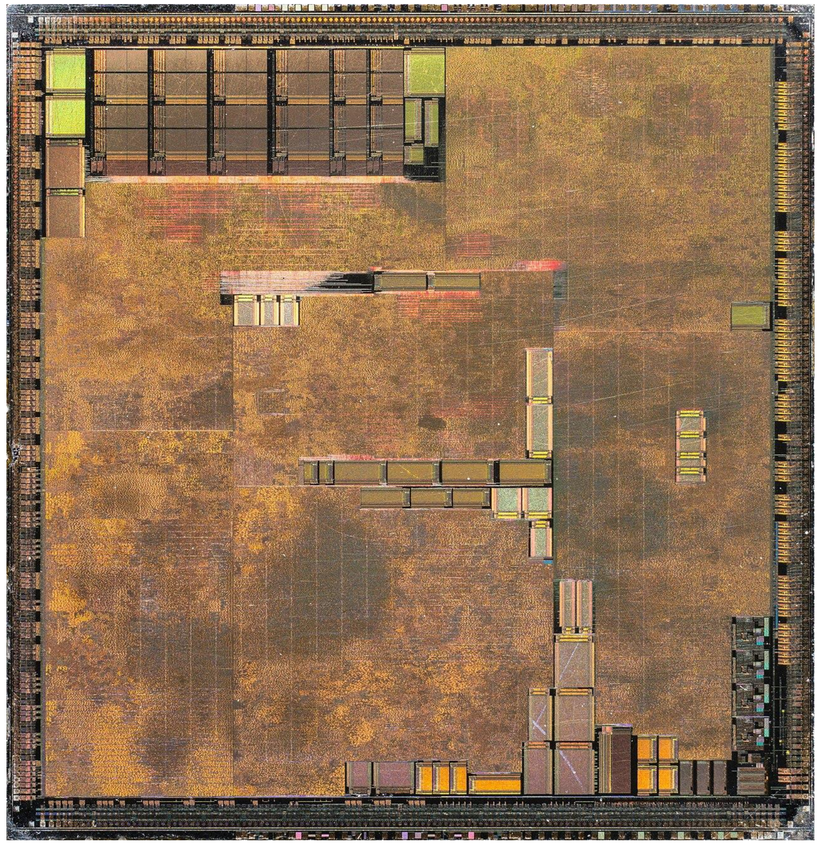
GPU: Introduction and Historical Overview

- GPU evolution: From fixed-function accelerator to programmable processor.
- Early adoption in machine learning: The massive parallel nature suitable for linear algebra.
- GPU in the deep learning : Architecture evolution driven by rapid growth of deep learning.

Phase	Era	Key Features	Examples
1️⃣ **Hardwired GPU (Fixed-Function Graphics)**	**1980s – Early 2000s**	- Specialized **fixed-function pipelines** for rendering - No programmability, only hardware-based transformations & rasterization	- **1981:** IBM CGA (First consumer graphics card) - **1999:** NVIDIA GeForce 256 (First GPU)
2️⃣ **Programmable GPU (Shader-Based Graphics Processing)**	**Early 2000s – 2010s**	- **Vertex & Pixel Shaders** introduced for **custom effects** - Unified Shader Architecture allows **software-defined rendering**	- **2001:** NVIDIA GeForce 3 (First programmable vertex shader) - **2006:** NVIDIA Tesla (First Unified Shader)
3️⃣ **GPGPU for AI/ML (General-Purpose GPU Computing)**	**2010s – Present**	- **CUDA/OpenCL** enable **parallel computing** for AI, HPC - **Tensor Cores & AI Accelerators** introduced	- **2007:** NVIDIA CUDA (GPGPU programming) - **2017:** NVIDIA Volta (First Tensor Cores for ML)

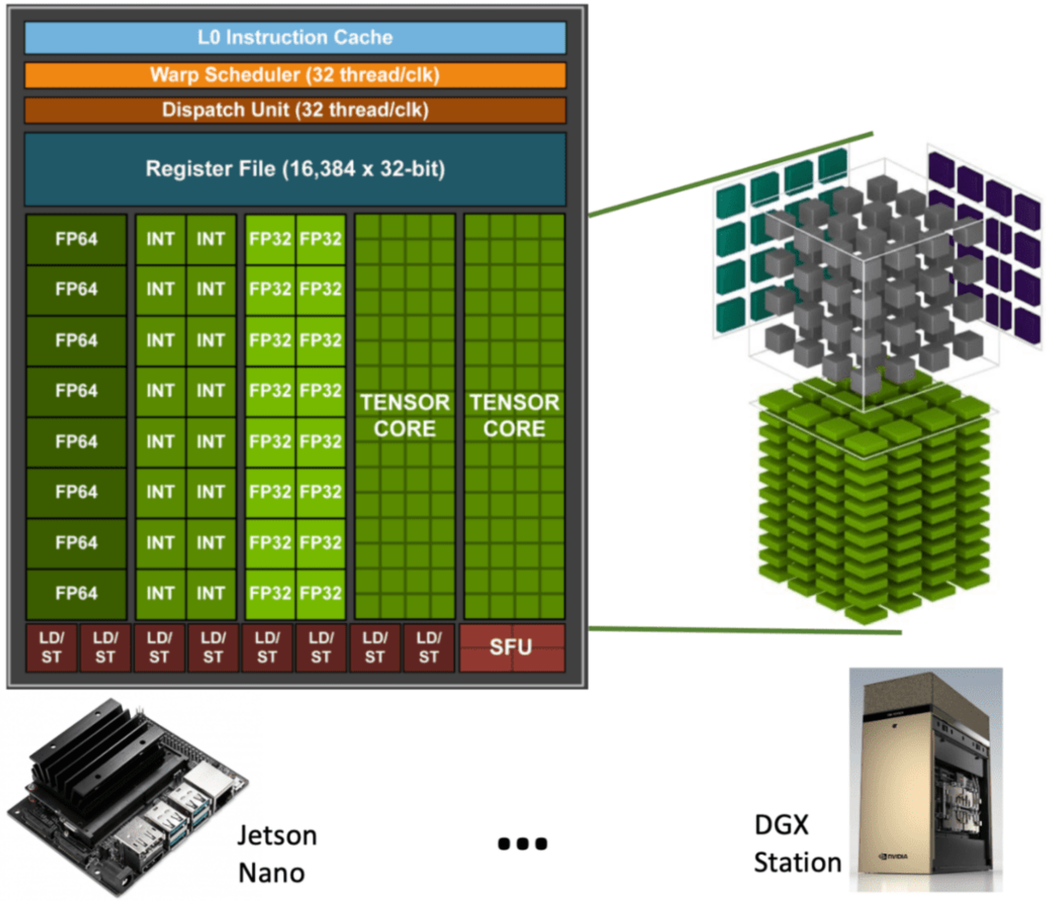
Architectural Fundamentals for Machine Learning

- GPU many-core computing
  - Unified graphics and parallel computing
  - CUDA cores, tensor cores, SFU
  - Warp Scheduler & control logic
  - Interconnect Network
- GPU memory hierarchy and bandwidth
  - Global Memory, Shared Memory, Local/Private Memory, Register File
  - L1, L2 Cache, Texture and Constant Memory
- GPU programming model basics



Modern GPU Architectures and Deep Learning

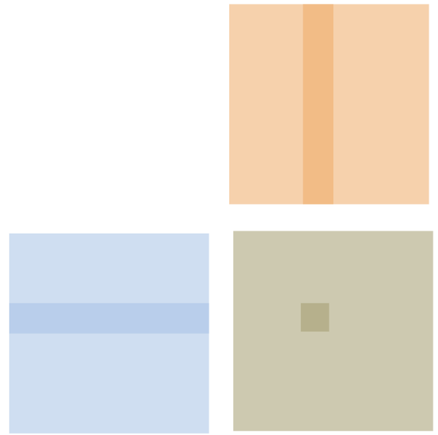
- From CUDA core to Tensor cores
- TensorFloat-32 (TF32) & Bfloat16 (BF16)
- Fine-grained structured sparsity
- Transformer engine
- Memory and interconnect advances
  - High-bandwidth memory (HBM)
  - NVLink and NVSwitch
  - Infiniband
- Multi-instance GPU (MIG)
- Specialized hardware vs GPUs

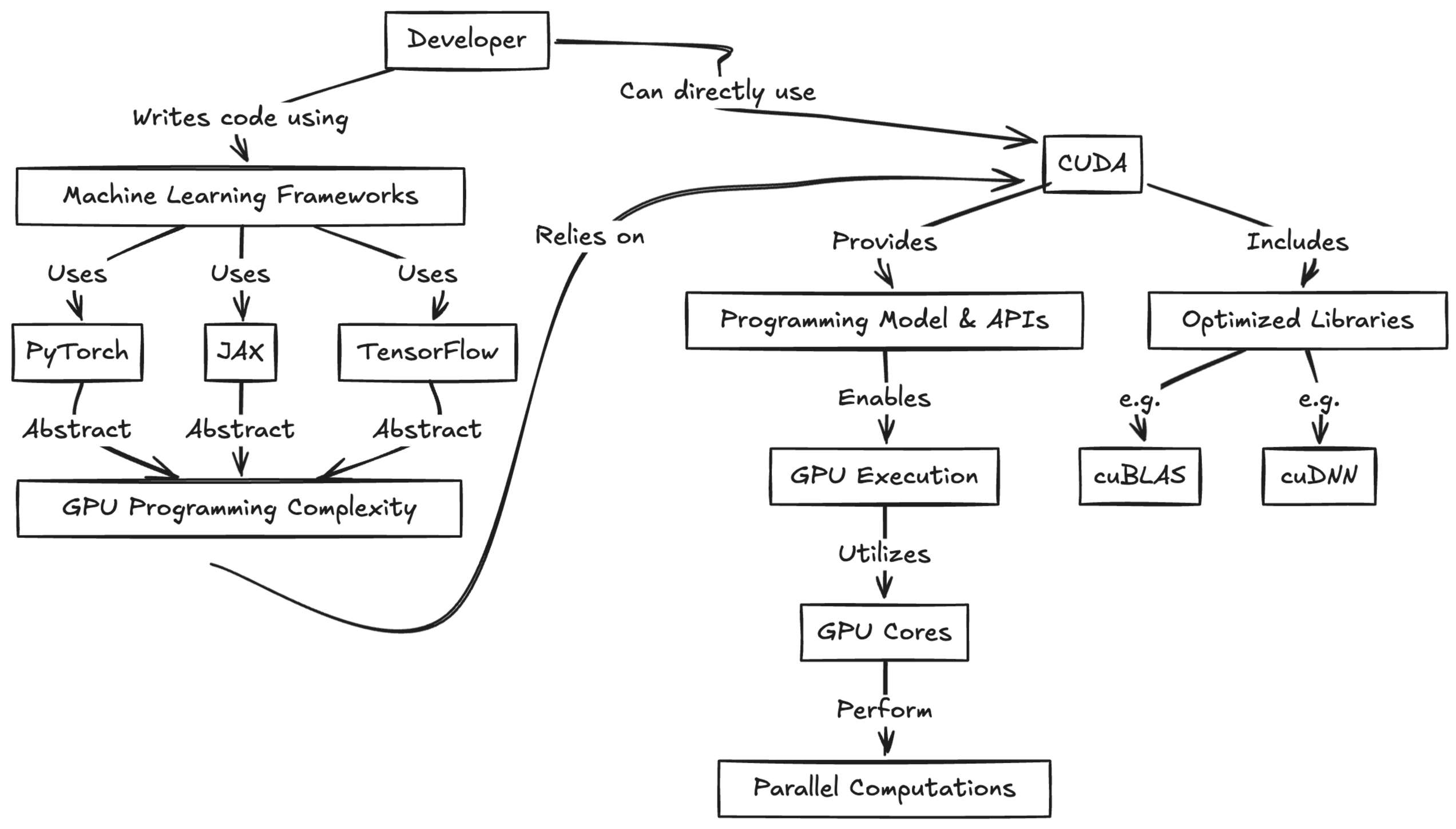


Optimization Strategies in GPU Computing

- Understand the performance bottleneck
- Efficient utilization of GPU resources
  - Maximizing parallel workload
  - Memory access patterns
  - Algorithmic optimizations: mixed precision training, overlapping computation and data transfers
- Case study: Matrix Multiplication Optimization
  - Blocking/tiling for efficient memory access
  - Arithmetic intensity considerations

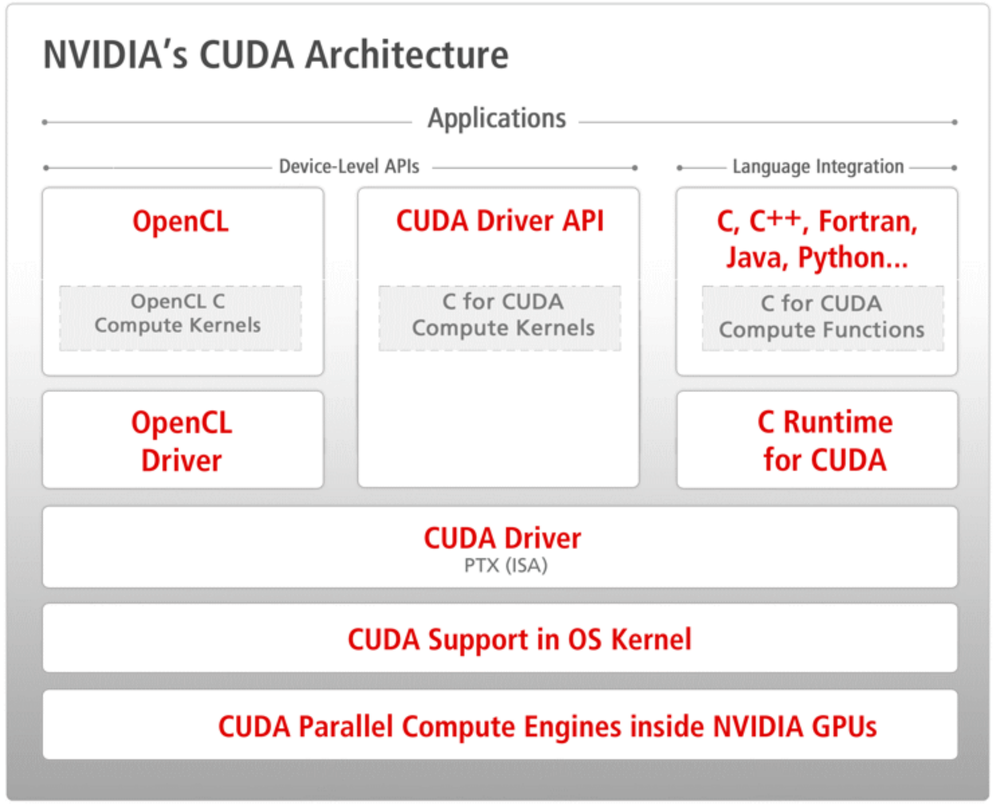
```
void matrixMultiplication ( float* A, float* B, float* C, int WIDTH)
{

    for (i → 0 : WIDHT)

        for (j → 0 : WIDTH)

            for (k → 0 : WIDTH)

                a = A_i;
                b = B_j;
                sum += a * b;

        C_ij = sum;
}
```




Developer
Writes code using
Machine Learning Frameworks
Uses
Uses
Uses
PyTorch
JAX
TensorFlow
Abstract
Abstract
Abstract
GPU Programming Complexity
Can directly use
CUDA
Relies on
Provides
Includes
Programming Model & APIs
Optimized Libraries
Enables
e.g.
e.g.
GPU Execution
cuBLAS
cuDNN
Utilizes
GPU Cores
Perform
Parallel Computations

CUDA Programming

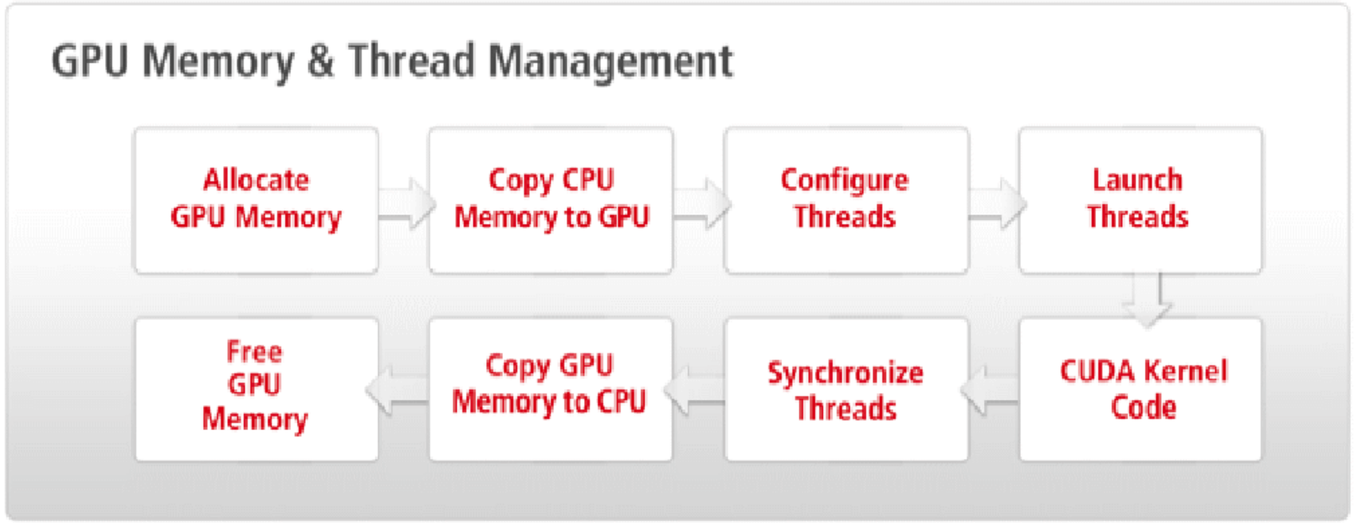
- Part 1: CUDA fundamentals and architecture
- Part 2: Hands-on CUDA program development
- Part 3: Memory management and optimization
- Part 4: Debugging tools and advanced resources
- Part 5: Recap and Q&A



CUDALink is an application of the NVIDIA architecture stack.

CUDA Programming

- Part 1: CUDA fundamentals and architecture
- Part 2: Hands-on CUDA program development
- Part 3: Memory management and optimization
- Part 4: Debugging tools and advanced resources
- Part 5: Recap and Q&A



Typical CUDA program cycle.

Deep Learning Compiler: The old days

- The MxN problems for model deployment
  - Increasing models and hardware backends
  - DNN programs struggle with memory footprints/execution efficiency
- Milestones for deep learning compilers
  - Halide: compute-schedule separation, but in graphics
  - TVM: the first demo of deep learning compilers
  - Ansor: the first demo of auto-schedulers for kernel optimization
  - Roller & Welder: tile-based operator fusion and code generation

Deep Learning Compiler: The old days

TVM: The First Demo of Deep Learning Compilers: TVM, an end-to-end deep learning compiler stack, marked its debut as a pioneering framework for optimizing deep learning models across diverse hardware (e.g., CPUs, GPUs, accelerators) with its first demonstration in 2018. It introduced graph-level and operator-level optimizations, such as high-level operator fusion, memory latency hiding, and hardware-specific code generation, showcasing its ability to outperform frameworks like TensorFlow by automating performance portability for deep learning workloads, starting with its initial release and demonstrations on server-class GPUs, embedded devices, and FPGAs.

Deep Learning Compiler for LLM

Triton, an open-source Python-like language by OpenAI, simplifies GPU programming, enabling efficient, high-performance kernel development for AI/ML workloads with minimal expertise, often outperforming PyTorch equivalents by 2x.

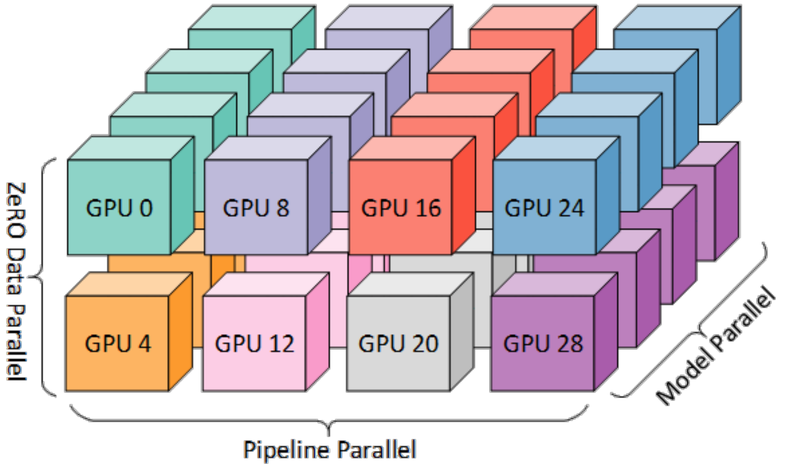
OpenAI's GPU programming framework (Python DSL)
Write NumPy-like code, compile to high-performance GPU kernels
Why Triton?

- CUDA is powerful but complex; Triton makes it simple
- Optimized for deep learning primitives (matmul, softmax, layer norm)
- Integrated into PyTorch 2.0 via TorchInductor

Deep Learning Compiler for LLM

- **High Productivity**: Python-based, no CUDA boilerplate
- **Performance**: Near cuBLAS/cuDNN with <100 lines of code
- **Customizability**: Easy to fuse ops & write custom kernels
- **Concepts**: Programs, Tiles, Program IDs (map neatly to GPU SMs)
- **Limitations**: Still young, fewer optimizations for irregular workloads
- **Big Picture**: Triton = “CUDA in Python” → core of modern ML systems stack

Parallel training



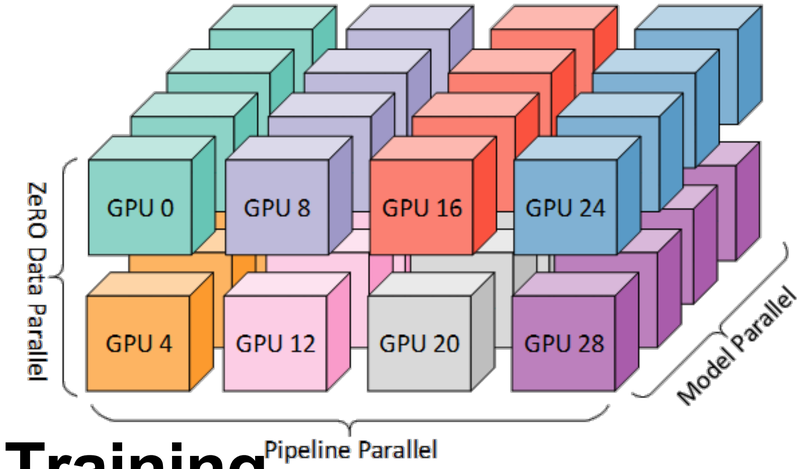
Fundamentals of Large Model Training

- Review of large-scale neural network training processes
- Introduction to Transformer model training
- Overview of software systems for large model training

Collective Communication for Large Model Training

- Basics of collective communication
- Important collective communication primitives
- Optimization of reduction (Reduce) and all-to-all collective communication

Parallel training



Multi-Dimensional Parallelization in Large Model Training

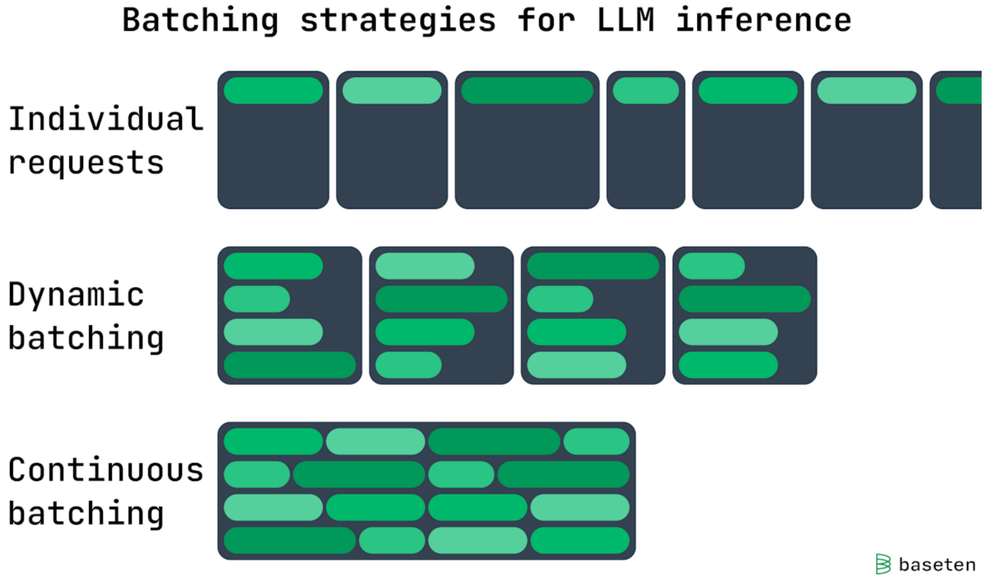
- Data parallelism, tensor parallelism, pipeline parallelism
- Expert parallelism, sequence parallelism, context parallelism

Memory Optimization and Acceleration in Large Model Training

- Re-computation methods
- Memory optimization techniques for data parallelism
- Typical methods for overlapping computation and communication

LLM Inference infra Preliminary

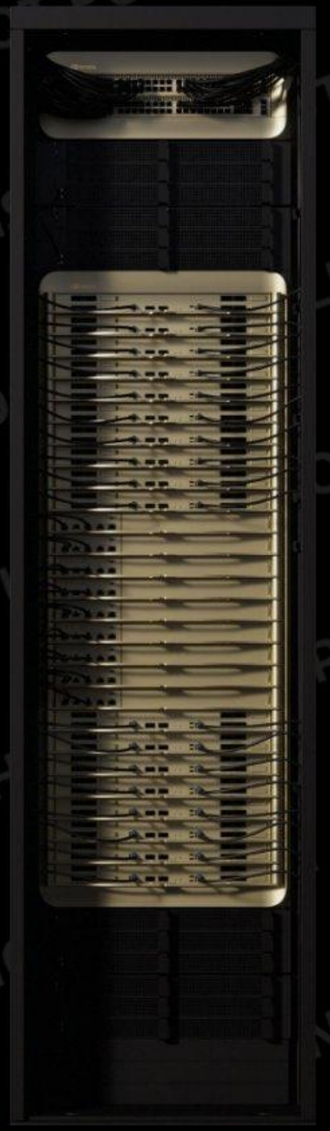
- Challenges
  - Generative Model: Prefill and Decode, KV Cache
  - Resource: Computation-bound and Memory-bound
  - Service Level Objectives
- Modeling
  - Architect: MoE (Mixture-of-experts) and its variants.
- Distributed System
  - Parallism: Tensor-Pipeline-Expert Parallelism
  - Concurrency: Batching, Caching



Announcing Vera Rubin NVL144 CPX

$5B Revenue for every $100M invested

VR NVL144 CPX



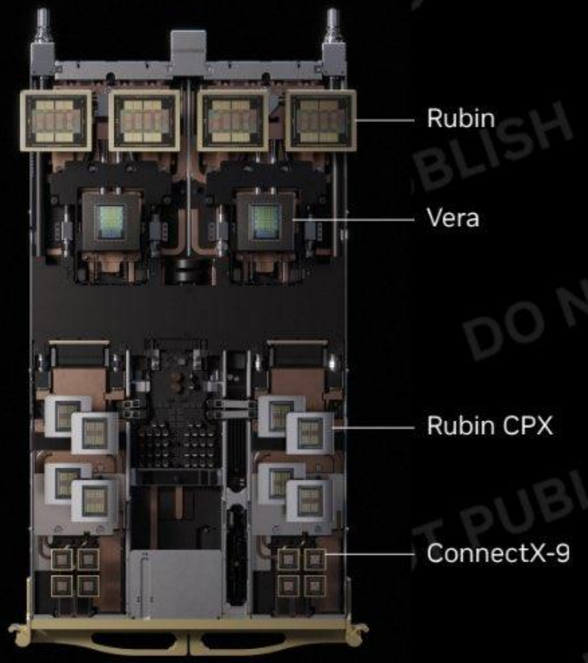
8 EF NVFP4
7.5x GB300 NVL72

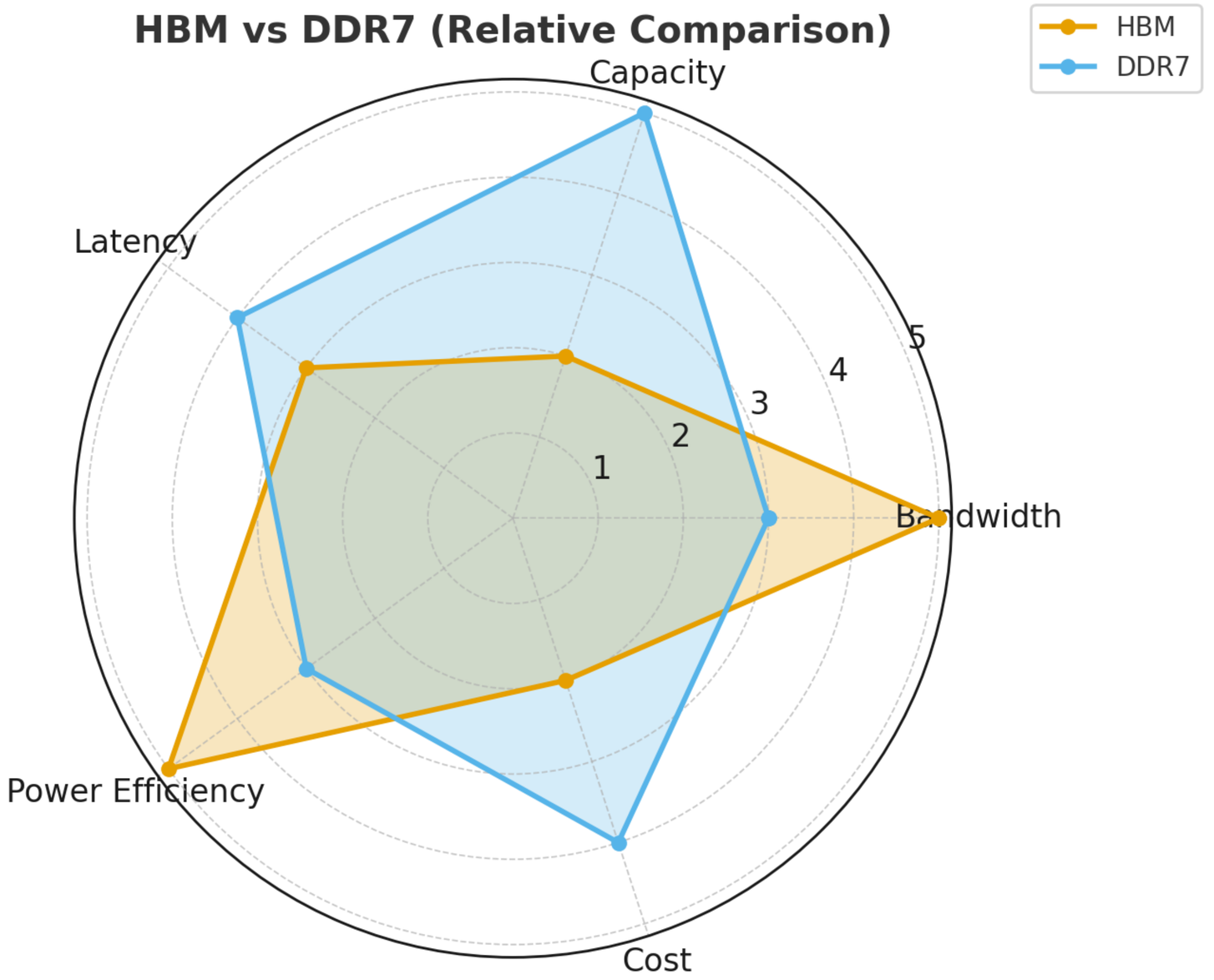
1.7 PB/s Memory
3x

100 TB Fast Memory
2.5x

Availability
End 2026

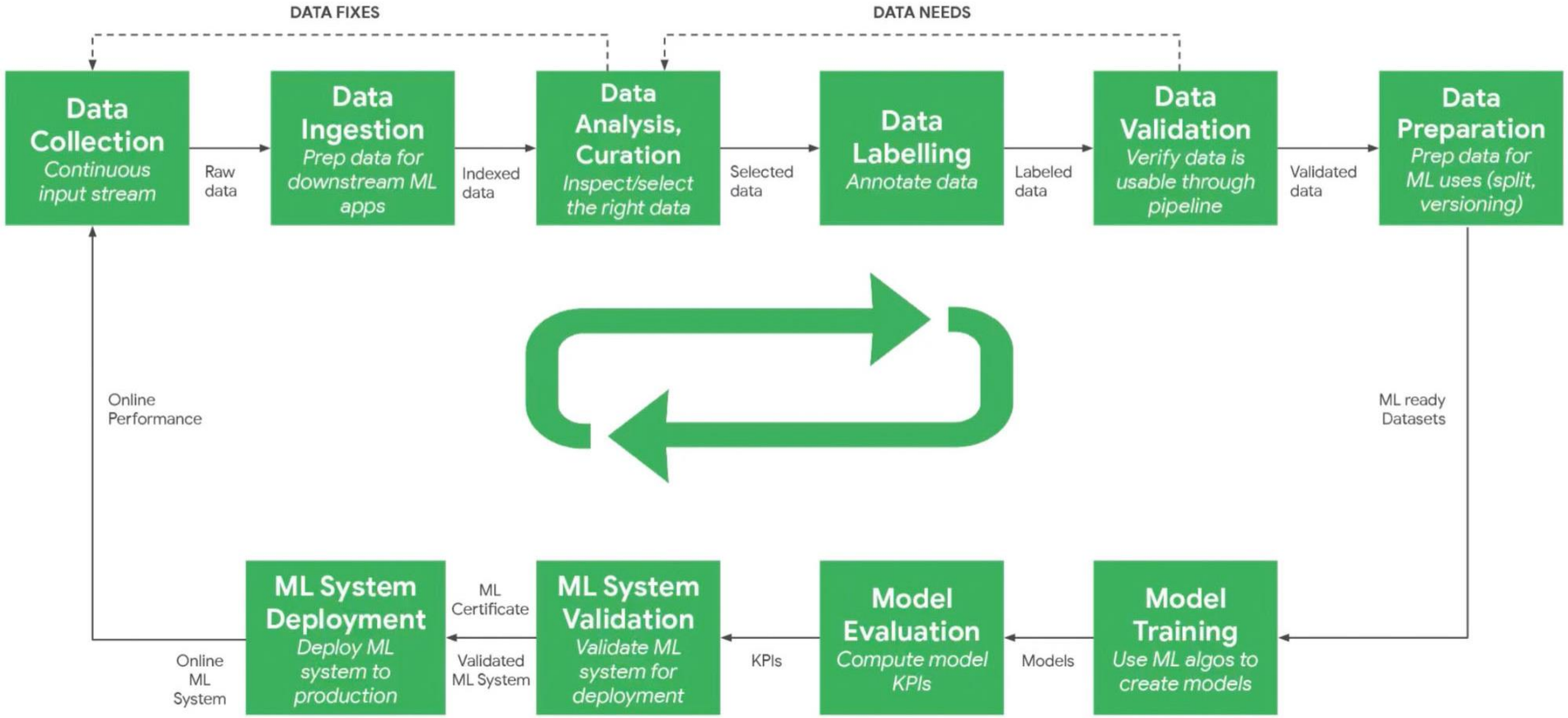
VR NVL144 CPX
Compute Tray

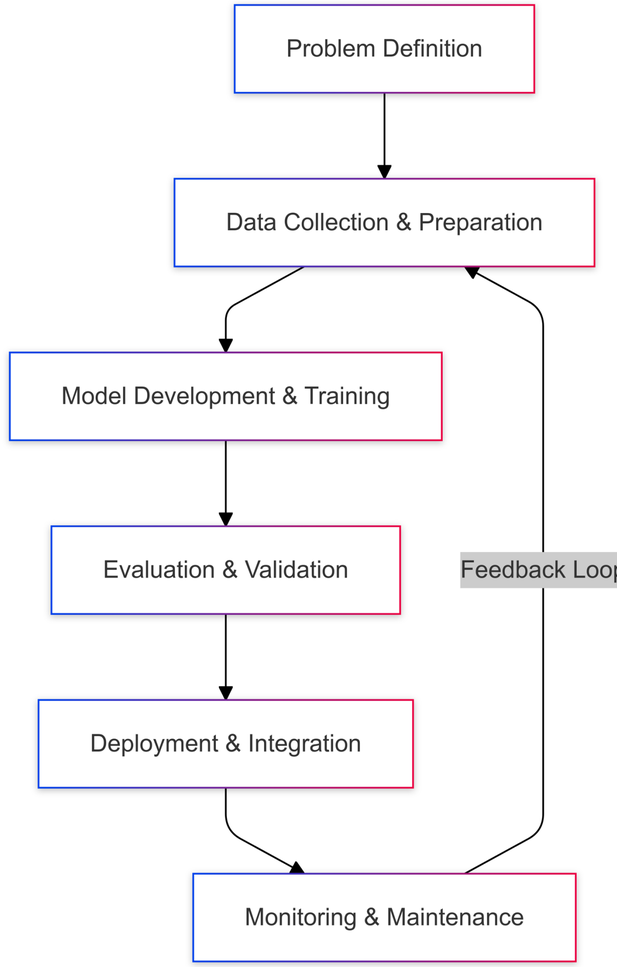



HBM vs DDR7 (Relative Comparison)
HBM
DDR7
Capacity
Latency
Bandwidth
Power Efficiency
Cost
1
2
3
4
5

DL Lifecycle

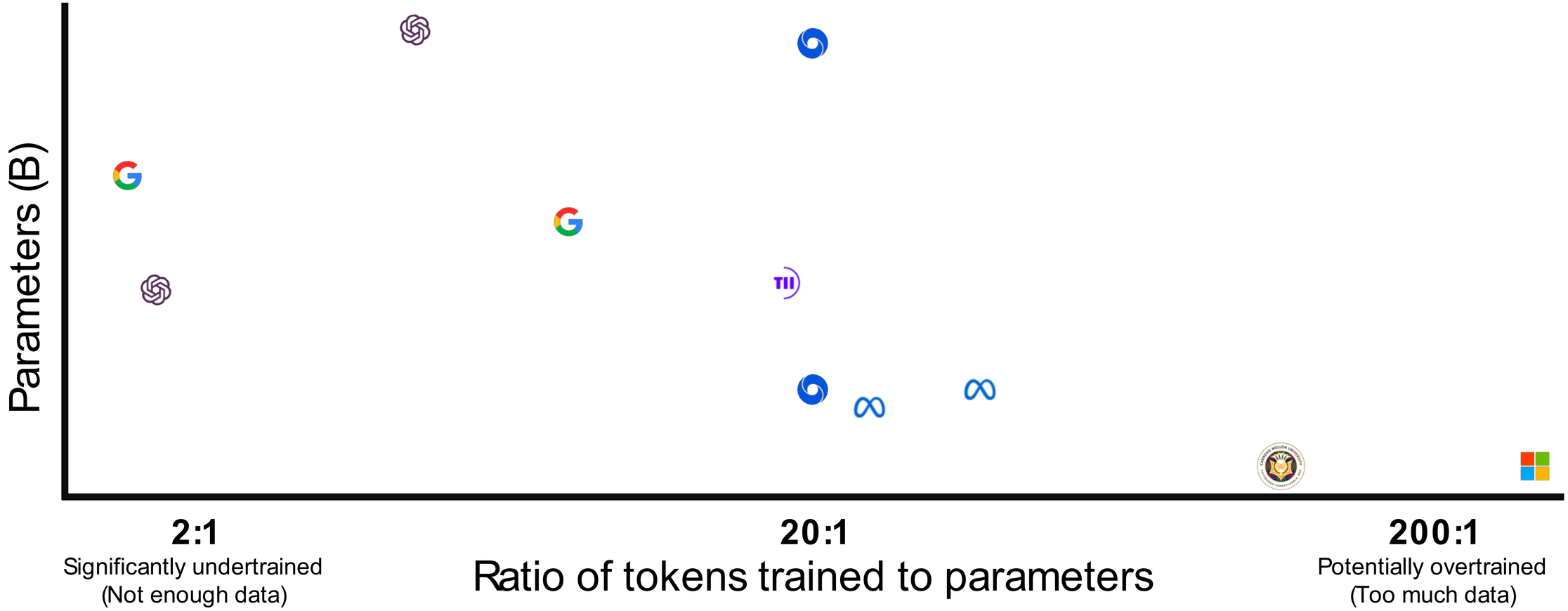
Data flow, model training and deployment





DATA-OPTIMAL (CHINCHILLA) MODEL HEATMAP

DEC/ 2023



Selected highlights only. Mostly to scale. Informed estimates for Palm 2, GPT-4, and Gemini. Alan D. Thompson. November 2022, major update December 2023. https://lifearchitect.ai/

LifeArchitect.ai/chinchilla

Data

1. **Pre-training:** Large-scale, diverse, noisy, redundant, and heterogeneous mixtures
    - Scale over quality (billions to trillions of tokens).
    - Tolerance for noise and redundancy since the goal is broad coverage and representation learning.
    - Balance across modalities and domains to avoid collapse into narrow expertise.
2. **Continued Pre-training (CPT):** Domain-specific or task-oriented corpora (e.g., biomedical papers, legal documents, scientific code).
    - Cleaner, higher-quality data compared to raw pretraining corpora.
    - Sufficient size to shift model distribution but not necessarily at web scale.
    - Designed mixtures to avoid catastrophic forgetting of general knowledge.
3. **Post-training (Fine-tuning & Alignment):** High-quality, curated, labeled, or human-annotated datasets.
    - Quality >> Quantity.
    - Human-in-the-loop or synthetic data with strict filtering.
    - Bias and safety considerations are crucial.
4. **Test-time Compute (Test-time Adaptation / Inference-time Optimization):** Input-specific or instance-specific context
    - Small but **highly relevant** context windows or retrieval sets.
    - Dynamic, real-time adaptation (e.g., chain-of-thought, self-consistency sampling, retrieval-augmented generation).
    - Not stored long-term but generated/collected on-the-fly during inference.

Beyond Performance

Stability: The fundamentals of LLM training stability, techniques to ensure training stability.

Privacy: Techniques to protect user data and prevent unauthorized data leakage during training and inference.

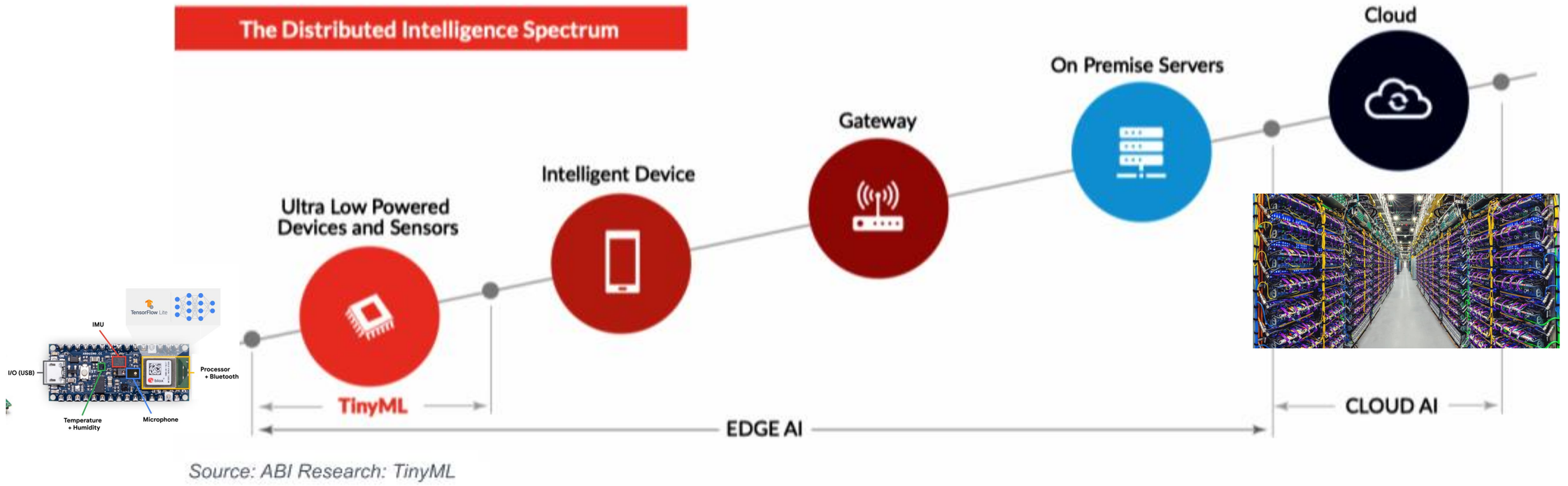
Safety: Robust content filtering and guardrails to minimize harmful outputs, such as misinformation or toxic content, ensuring safe interactions.

Security: Strengthen LLM security by using encryption, secure APIs, and adversarial testing to safeguard against attacks like model inversion or data poisoning.

Responsibility: Transparent development practices and bias mitigation strategies to ensure fair and ethical model behavior across diverse users.

Sustainability: Optimizing training efficiency, reducing energy consumption, and leveraging carbon-neutral hardware to minimize environmental impact.

Towards Edge Intelligence



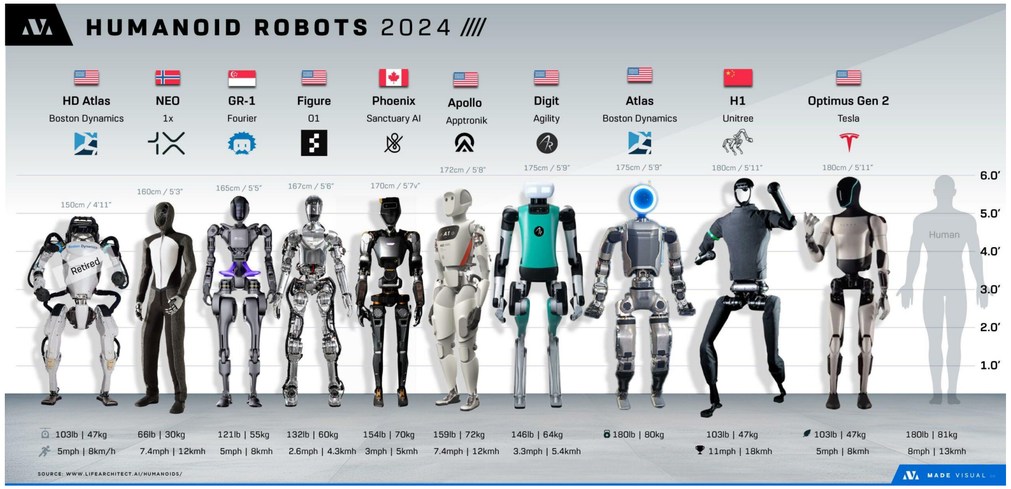
Source: ABI Research: TinyML

	Cloud AI (NVIDIA V100)		Mobile AI (iPhone 11)		Tiny AI (STM32F746)		ResNet-50	MobileNetV2	MobileNetV2 (int8)
Memory	16 GB	4× →	4 GB	3100× →	**320 kB**	← gap →	7.2 MB	6.8 MB	1.7 MB
Storage	TB~PB	1000× →	>64 GB	64000× →	**1 MB**	← gap →	102MB	13.6 MB	3.4 MB

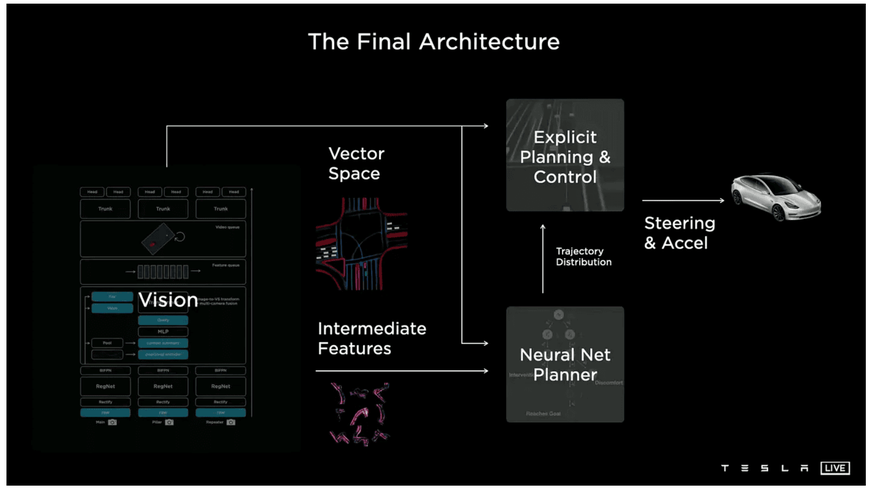
Model Optimization

- **Quantization / Data Formats:** FP16, BF16, FP8, FP4, INT8 to cut memory & compute.
- **Pruning:** Remove redundant weights/neurons to shrink model and FLOPs.
- **Low-Rank Approximation:** Factorize weights (e.g., LoRA, SVD) for fewer parameters.
- **Knowledge Distillation:** Train compact student model with teacher outputs.
- **KV Cache Compression:** Compress key–value caches for long-sequence inference.
- **Latent Space Dimension Reduction:** Reduce hidden dimensions to save memory.
- **Operator Fusion & Graph Optimization:** Fuse kernels to improve locality and throughput.

Embodied Intelligence

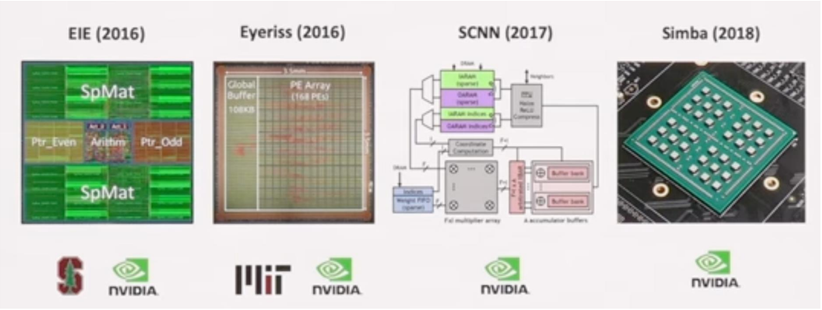


Humanoid Robotics



Automated Driving

AI Acceleration: Beyond GPU



- **Nvidia**: GPUs+CUDA
- **Cerebras**: Go big with wafer-scale engines
- **GraphCore:** In-house software stack + multi-threaded dataflow execution
- **Reconfigurable dataflow:** wave computing, SambaNova, SimpleMachines
- **Hailo:** Efficient Dataflow for edge inference
- **Systolic Arrays+VLIW:** TPUV1, Groq, and Habana
- **RISC-based AI Accelerators:** Esperanto and TensTorrent
- **LightMatter:** Photonics-based Analog Computing