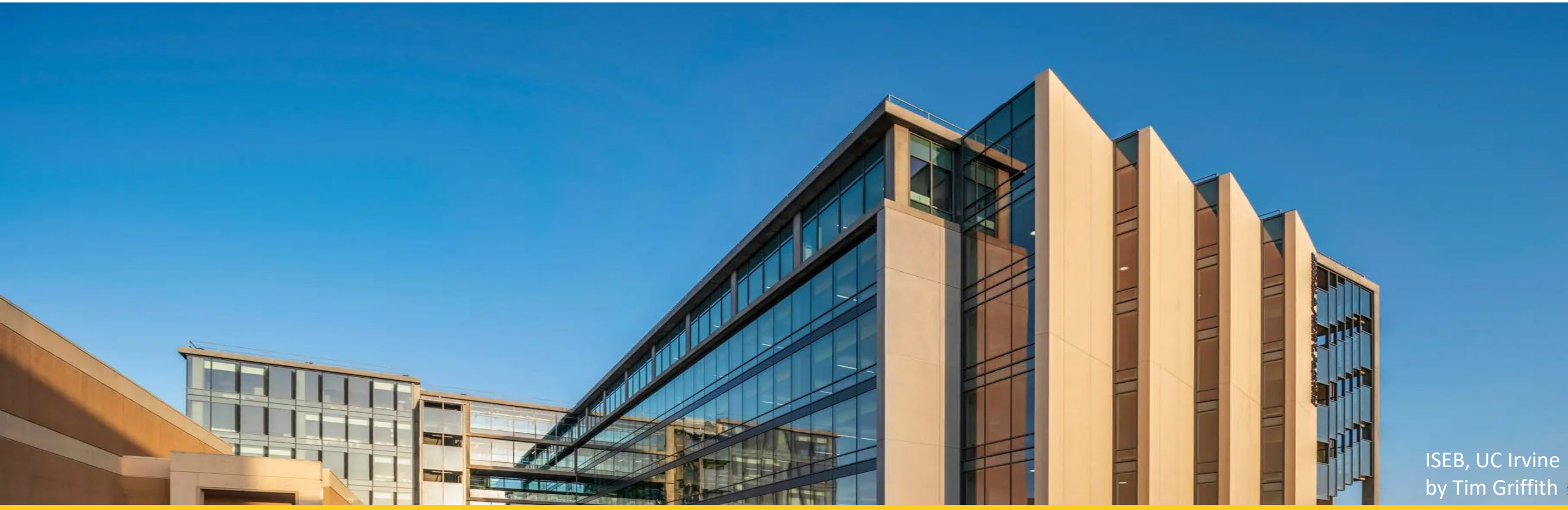
Lecture 1:

Introduction

Sitao Huang, sitaoh@uci.edu

October 2, 2023



ISEB, UC Irvine
by Tim Griffith

Course Information

Instructor

- Sitao Huang, sitaoh@uci.edu
- Office Hours: EH 3225, Mondays 11am-12pm (after class) or by appointment

Lectures

- Mondays and Wednesdays 9:30 – 10:50 AM
- Location: **SSPA 1165**

Course Website

- **Canvas**: https://canvas.eee.uci.edu/courses/58924
- Check **Canvas** and **emails** for the latest announcements
- Q & A: *Ed Discussion* (Canvas page navigation menu → Ed Discussion)

Reference Books

References (optional)

- P. Marwedel, *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems and the Internet of Things*, Fourth Edition, Springer, 2021. (website)
- D. D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner, *Embedded System Design: Modeling, Synthesis, Verification*, Springer, 2009.
- F. Vahid, T. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*, John Wiley and Sons, 2002.
- S. Pasricha, N. Dutt, *On-Chip Communication Architectures (System on Chip Interconnect)*, Morgan Kaufman, 2008. (website)
- G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

Reference Books

References (optional) – cont.

- M. Wolf, *Computers as Components: Principles of Embedded Computing System Design*, Fifth Edition, Elsevier, 2022.
- R. Zurawski (Editor), *Embedded Systems Handbook*, Second Edition, CRC Press, 2009.
- T. Noergaard, *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*, Second Edition, Newnes, 2012.

Grading Policy

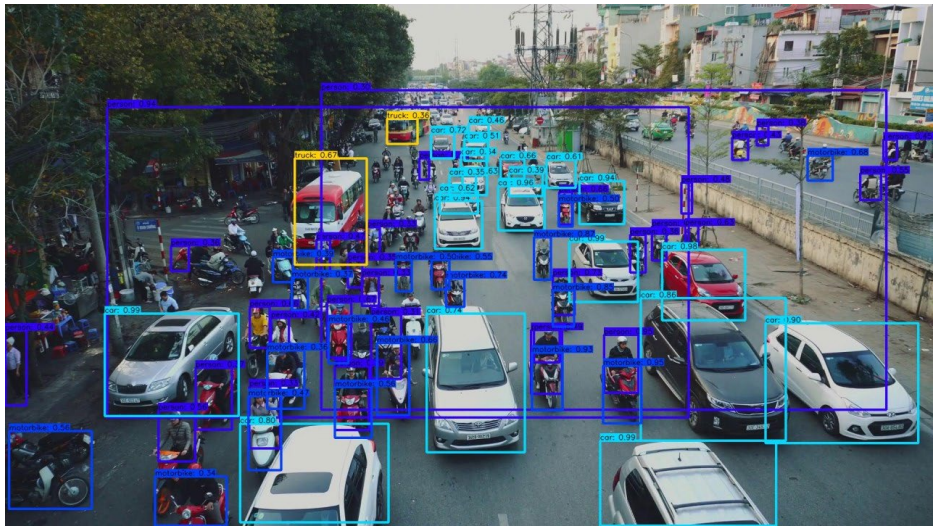
- **Homework – 30%**
  - 2~3 assignments
- **Labs – 40%**
  - FPGA-based SoC design labs
- **Final Project – 30%**

*NOTE: Please check **Canvas** for the latest announcements*

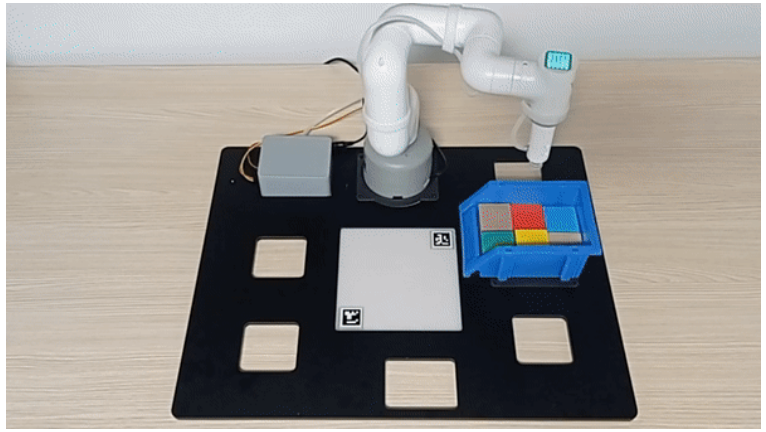
Tentative Schedule

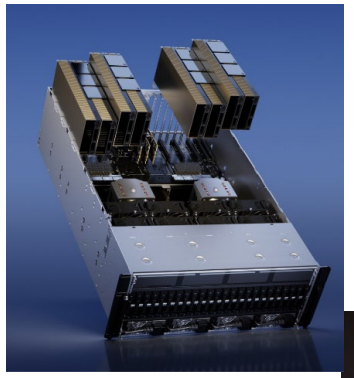
- **Week 1** (10/2): Introduction
- **Week 2** (10/9): Hardware Design
- **Week 3** (10/16): High-Level Synthesis (HLS)
- **Week 4** (10/23): High-Level Synthesis Algorithms
- **Week 5** (10/30): Hardware/Software Co-Design
- **Week 6** (11/6): SoC Performance Modeling
- **Week 7** (11/13): SoC Communication Architectures
- **Week 8** (11/20): Verification and Testing
- **Week 9** (11/27): GPUs and other Accelerators
- **Week 10** (12/4): Final Project Presentations

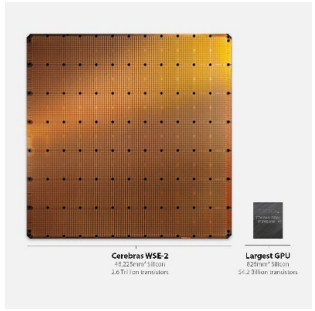
Computing *Transforms* Technology

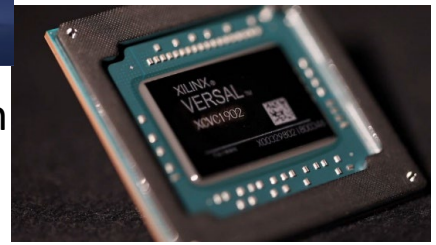


Deep Learning Demo from 1989 (Yann LeCun)

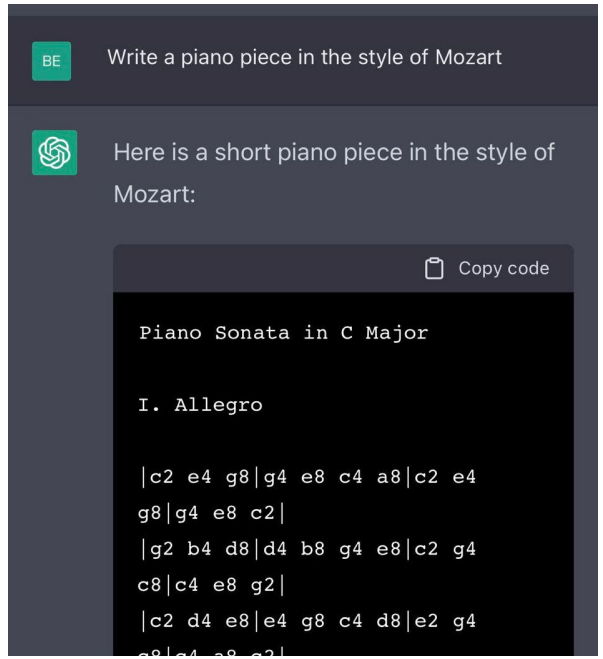


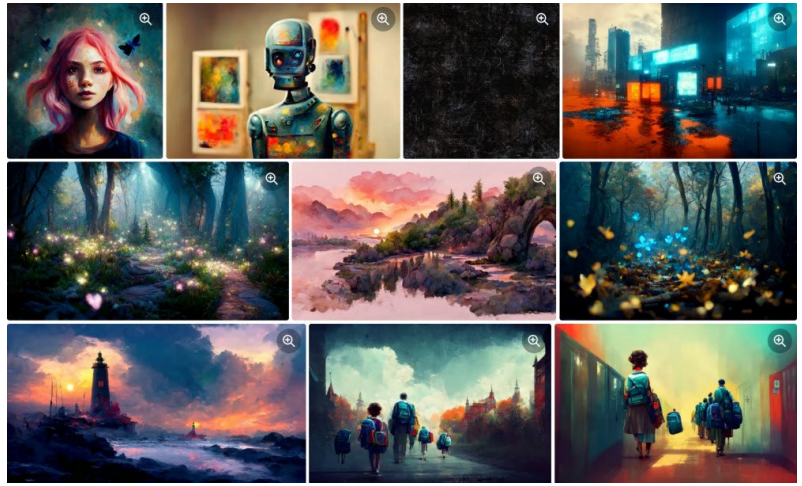




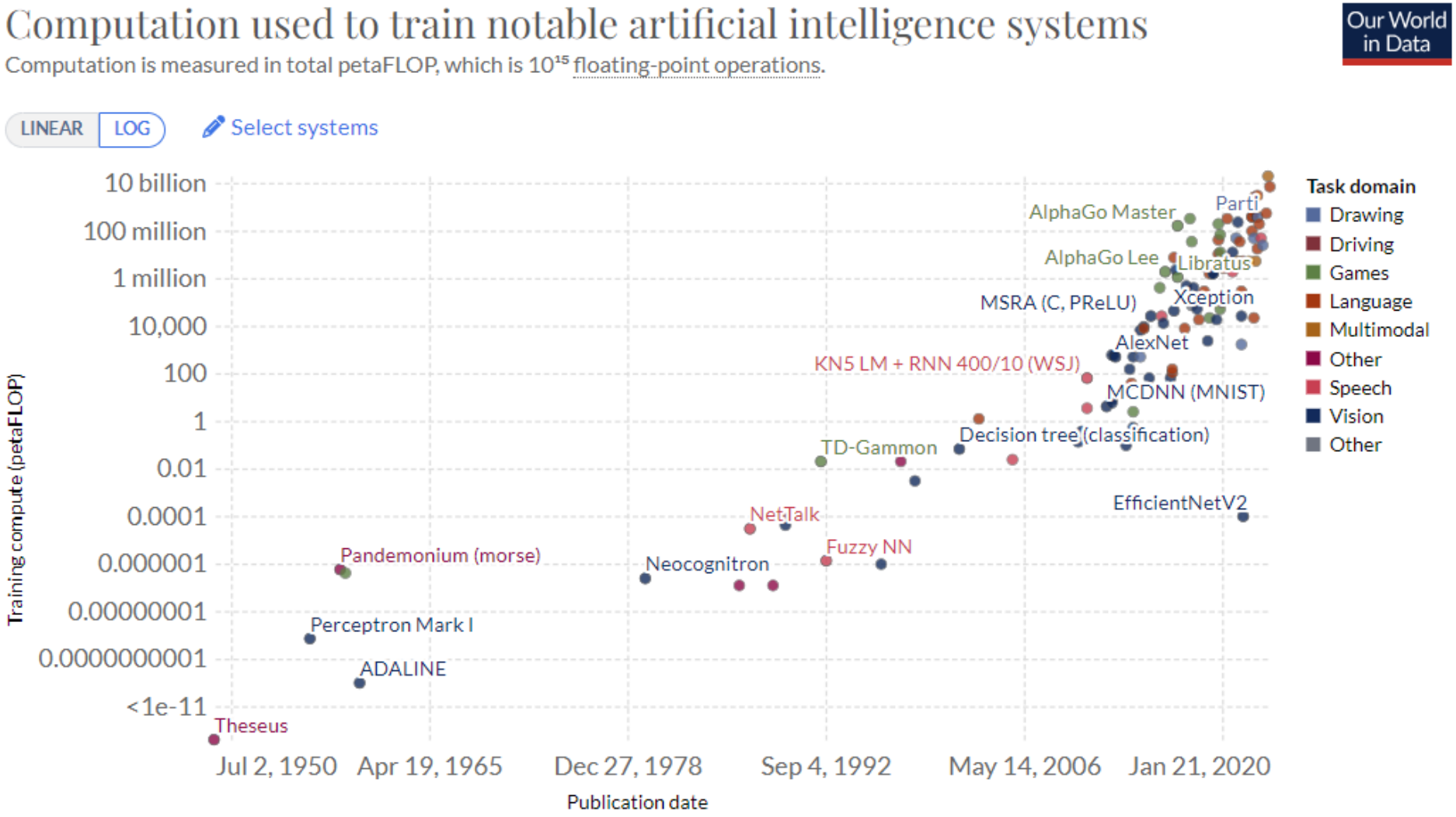


High performance, high efficiency chips and computing systems

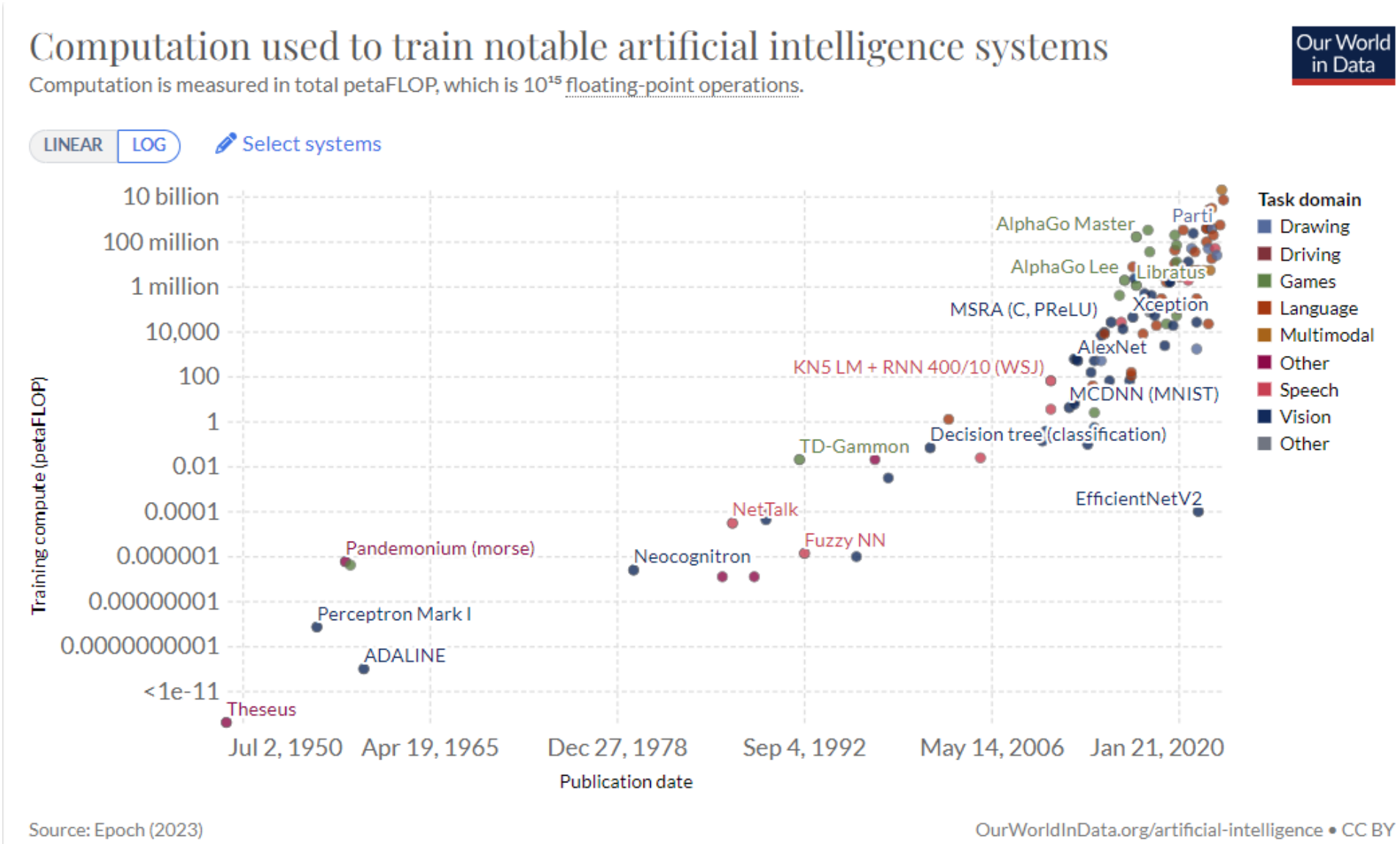




Computing *Transforms* Technology



Computing *Transforms* Technology, with Help from Design Automation



Introduction: The Computer Revolution

Three revolutions for civilization:

- Agricultural revolution
- Industrial revolution
- **Information revolution**
  - **Computer revolution is the foundation**

Computer revolution makes novel applications feasible

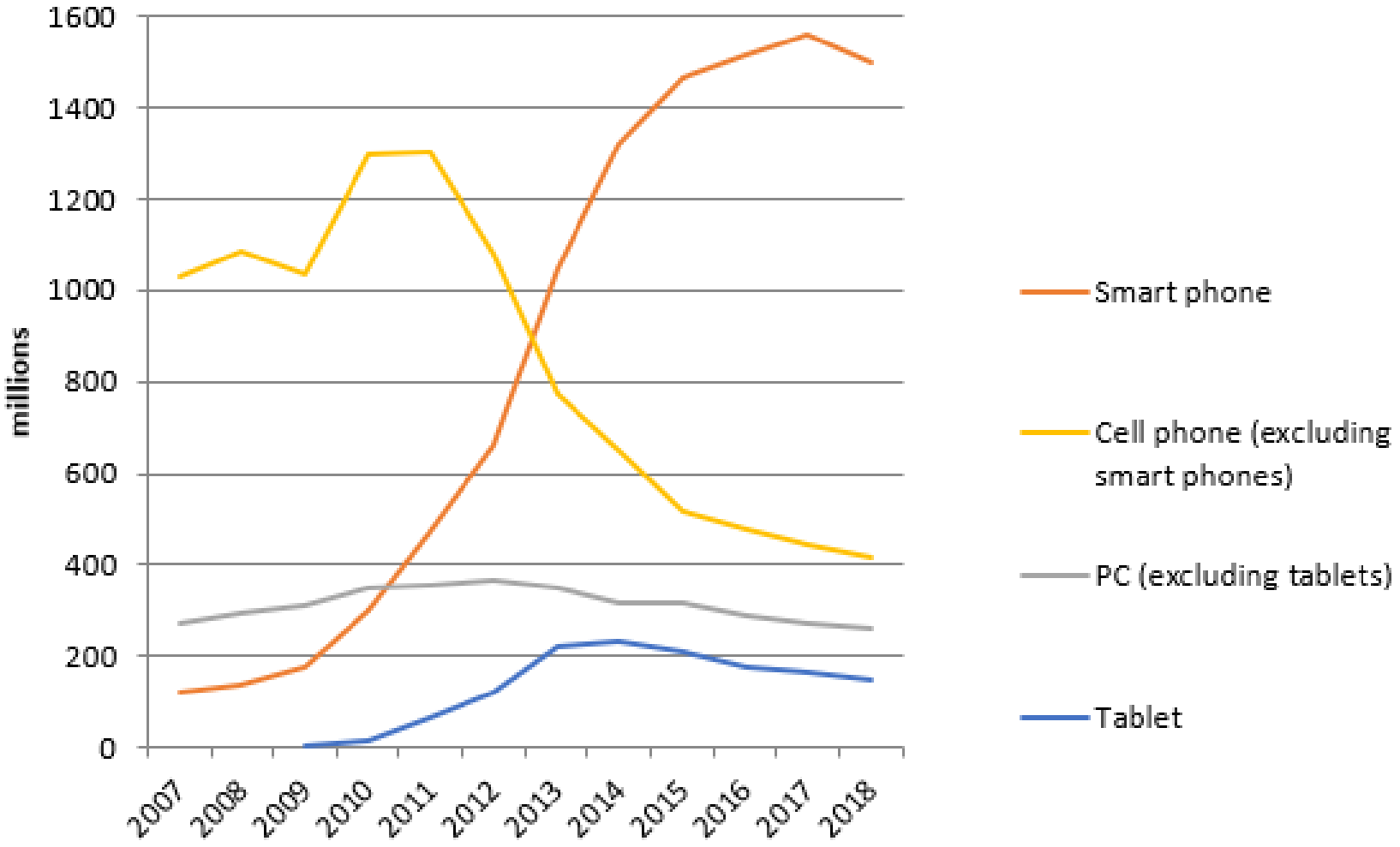
- Computers in automobiles
- Cell phones
- Human genome project
- World Wide Web
- Search Engines

Computers are pervasive

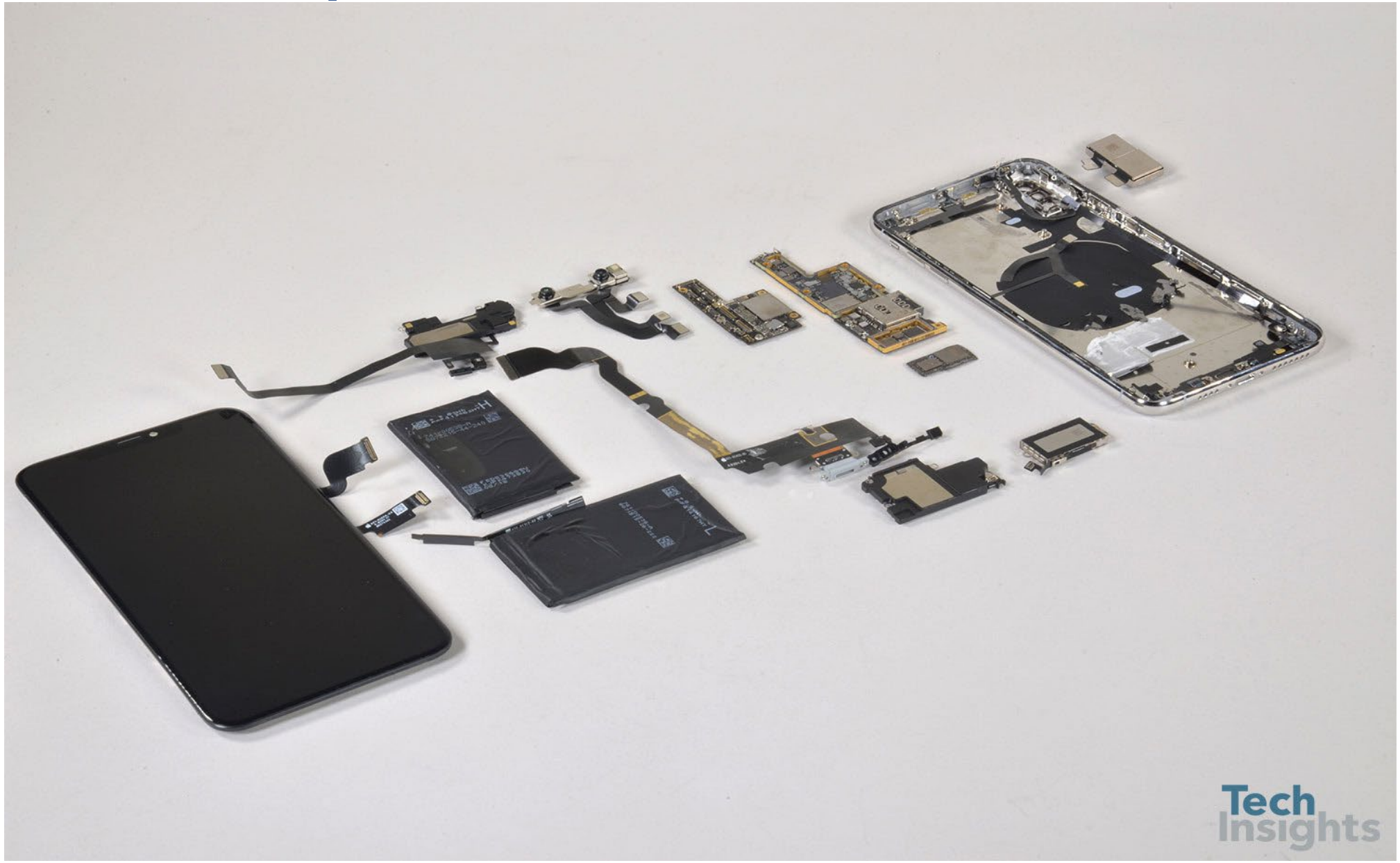
Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

The Post PC Era

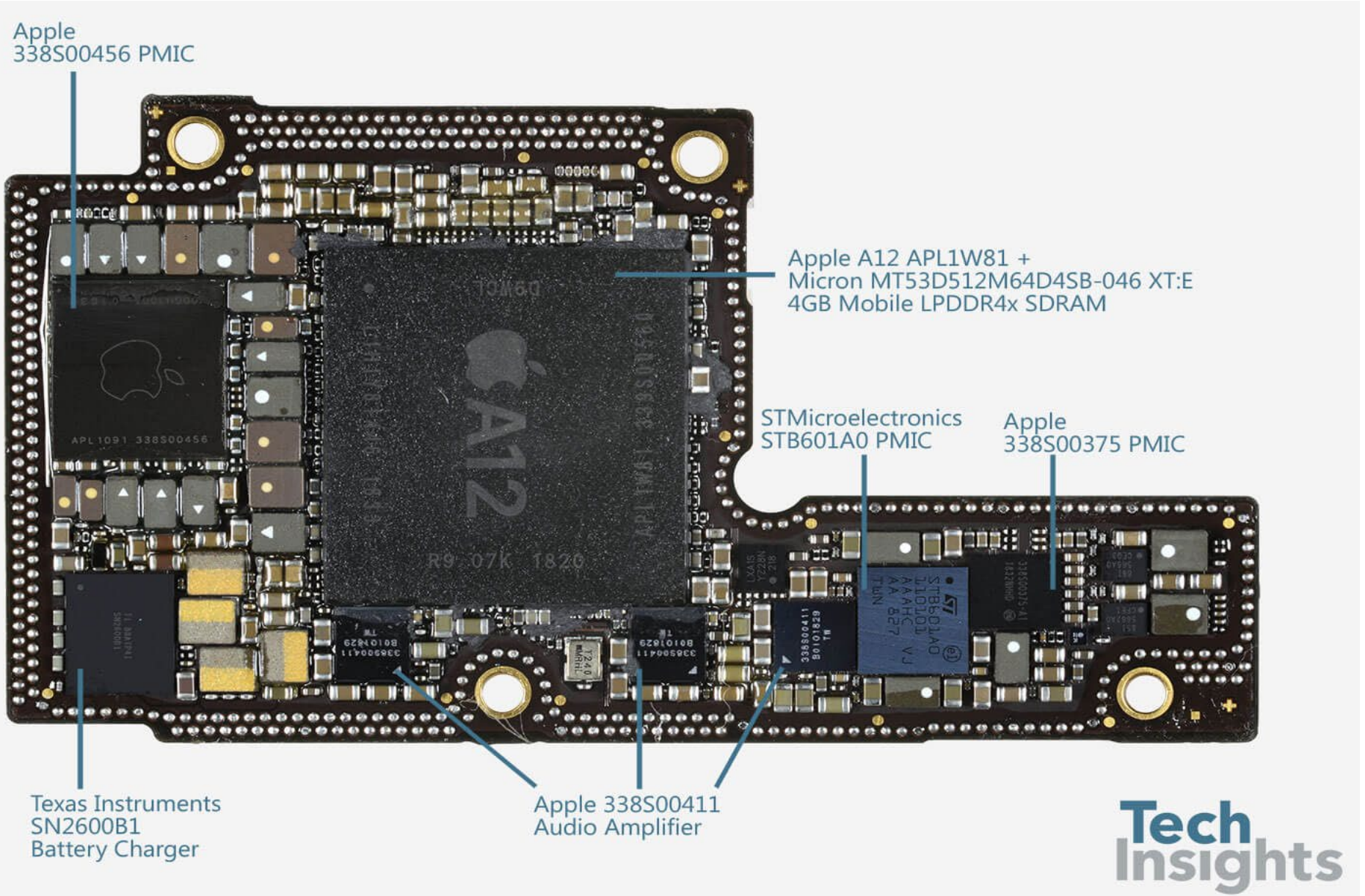


Inside Smartphones



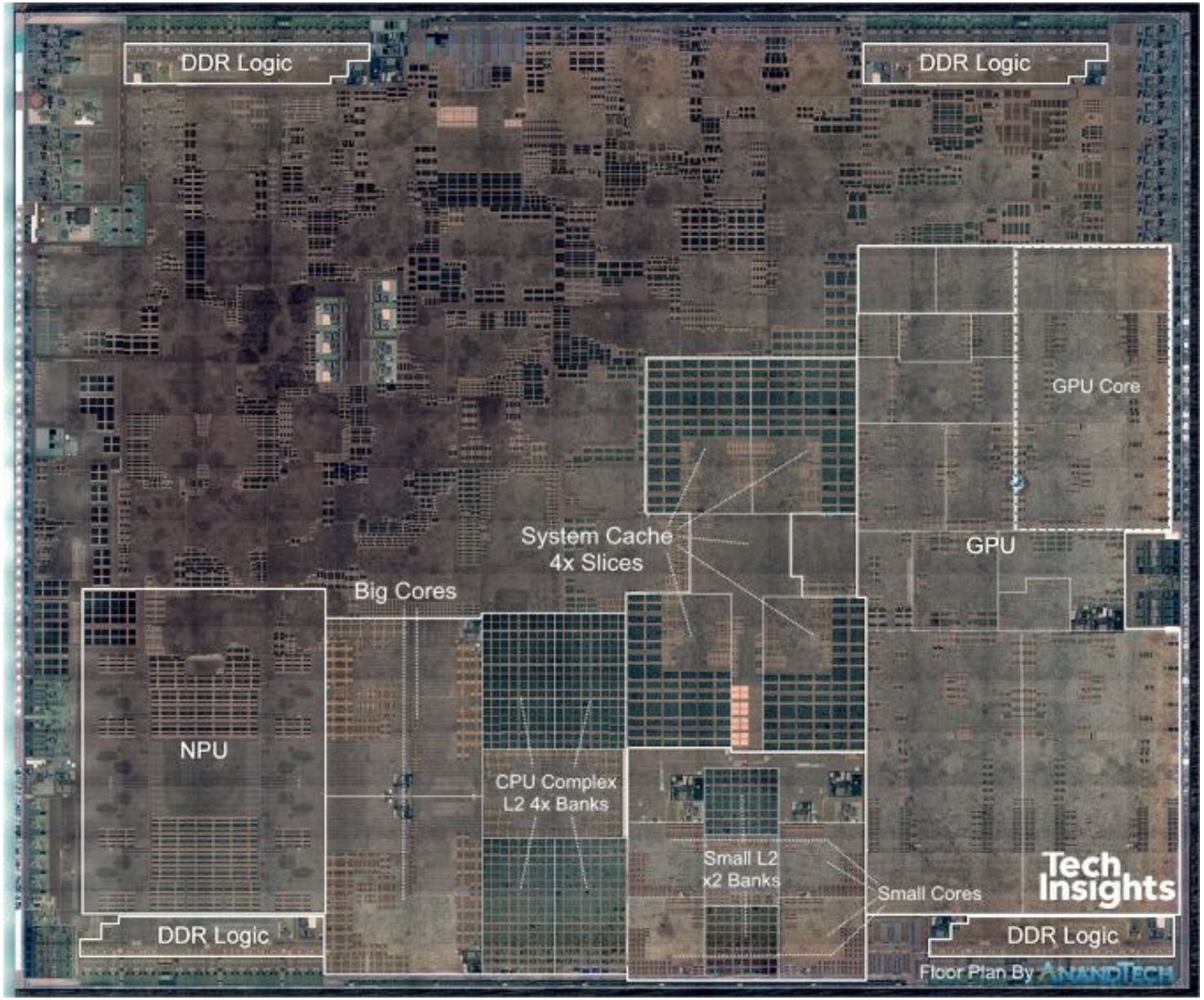
Inside iPhone
XS Max

Inside Smartphones



Inside iPhone XS Max

Inside the Processor



Apple A12 **System-on-Chip (SoC)**

What is SoC Design?

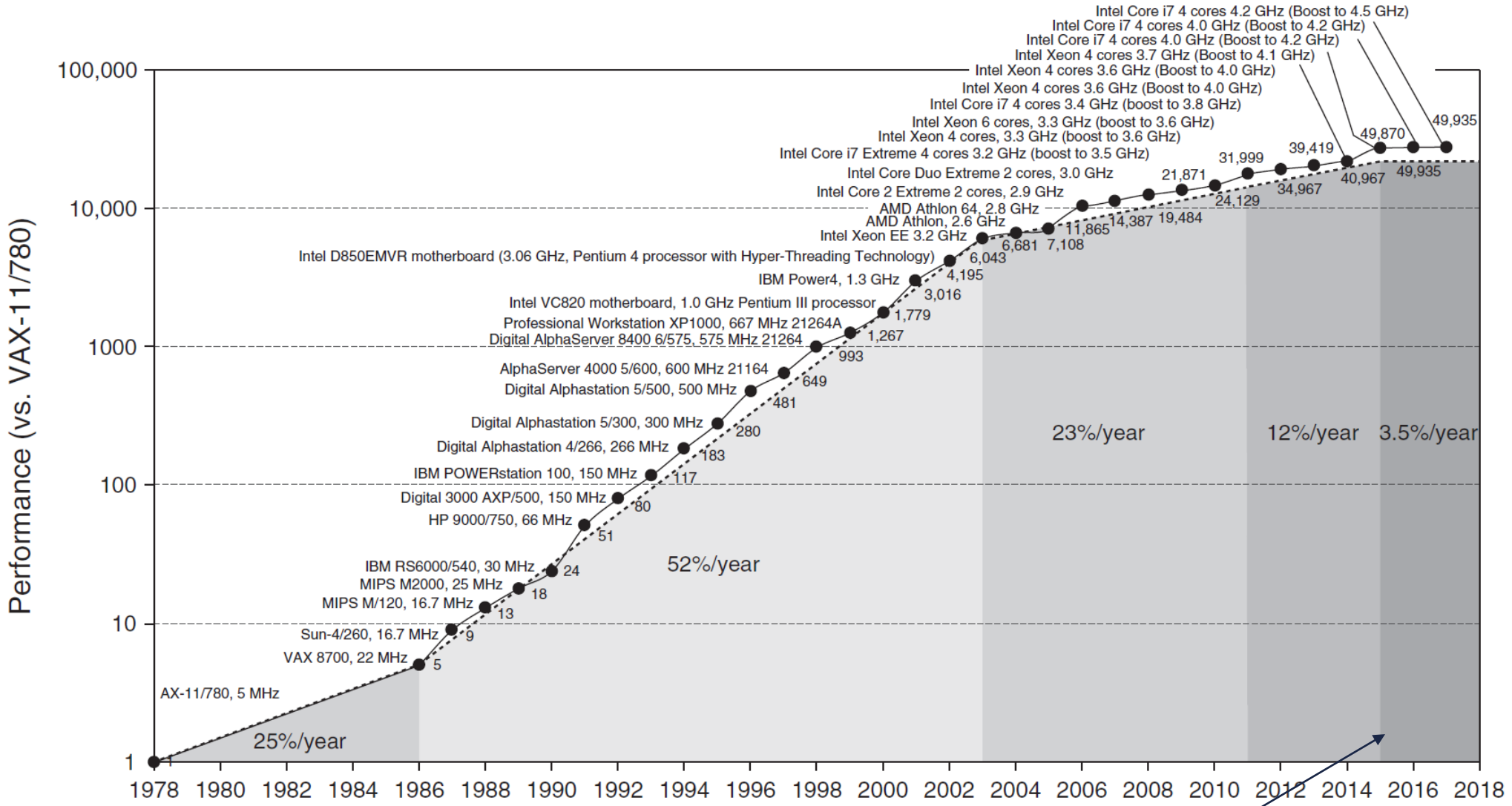
- A System-on-Chip (SoC) design is defined as a complex IC that integrates the major functional elements of a ***complete end-product (or system)*** into a ***single chip or chipset***.
- In general, SoC design incorporates at least one ***programmable processor***, on-chip memory, and accelerating function units implemented in hardware.
- It also interfaces to ***peripheral*** devices and/or the real world.
- SoC designs encompass both ***hardware*** and ***software*** components. Because SoC designs can interface to the real world, they often incorporate ***analog*** components, and can also include opto/microelectronic mechanical system (O/MEMS) components.
- Large benefits in cost, size, performance, power consumption, and time to market.

Why SoC?

- Because we can do SoC integration?

- Size
- Size/Cost
- Performance/Power

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

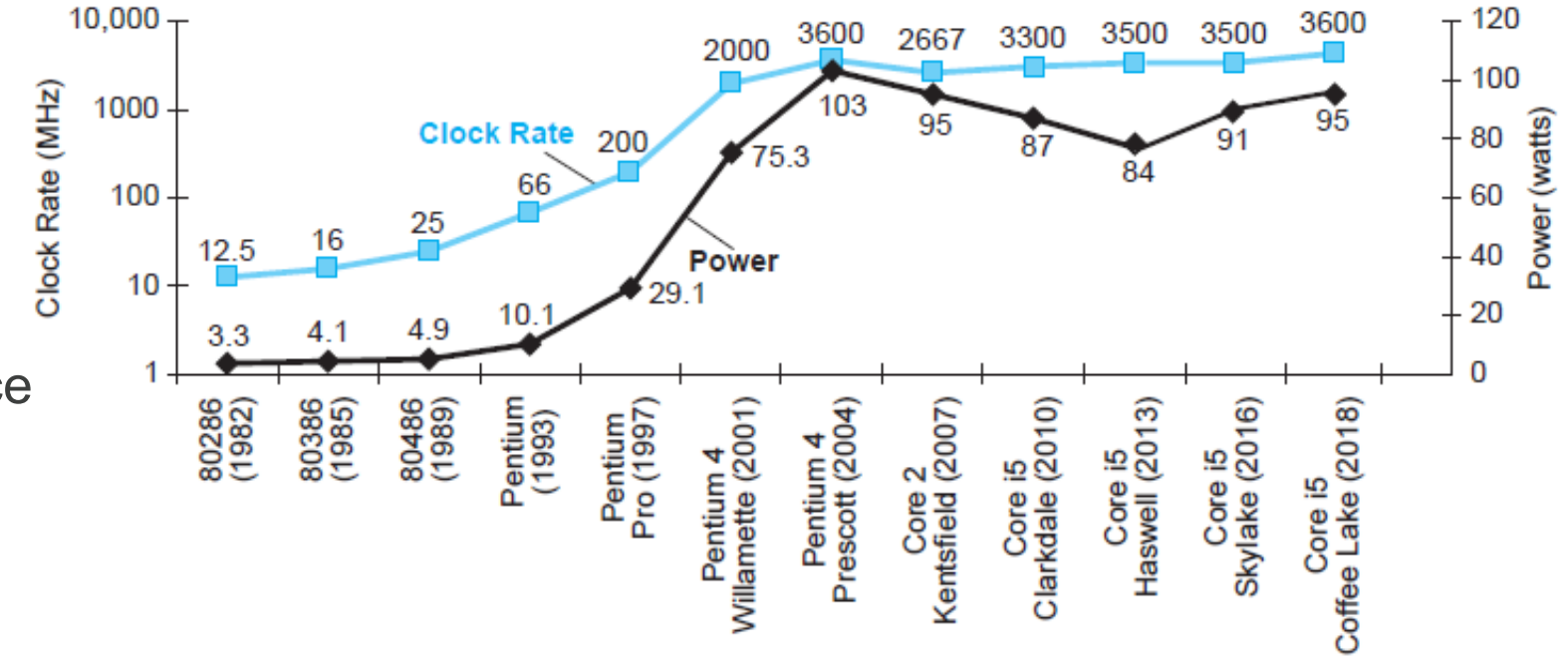
Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

The Power Wall

In CMOS IC technology

- Total Power = $P_{dynamic} + P_{static}$
- Dynamic power is the primary source of power consumption



- Dynamic energy of a pulse 0 → 1 (or 1 → 0):

$$Energy \propto \frac{1}{2} \times \text{Capacitive load} \times \text{Voltage}^2$$

Energy stored in a capacitor:

$$E = \frac{1}{2} CV^2$$

Capacitive load is a function of

- # of transistors connected to an output (*fanout*)
- Technology

- Dynamic power is the product of energy of a transition and frequency of transition:

$$Power \propto \frac{1}{2} \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Transition Frequency}$$

×30 | 5V → 1V | ×1000

Changes in the past 30 years

Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

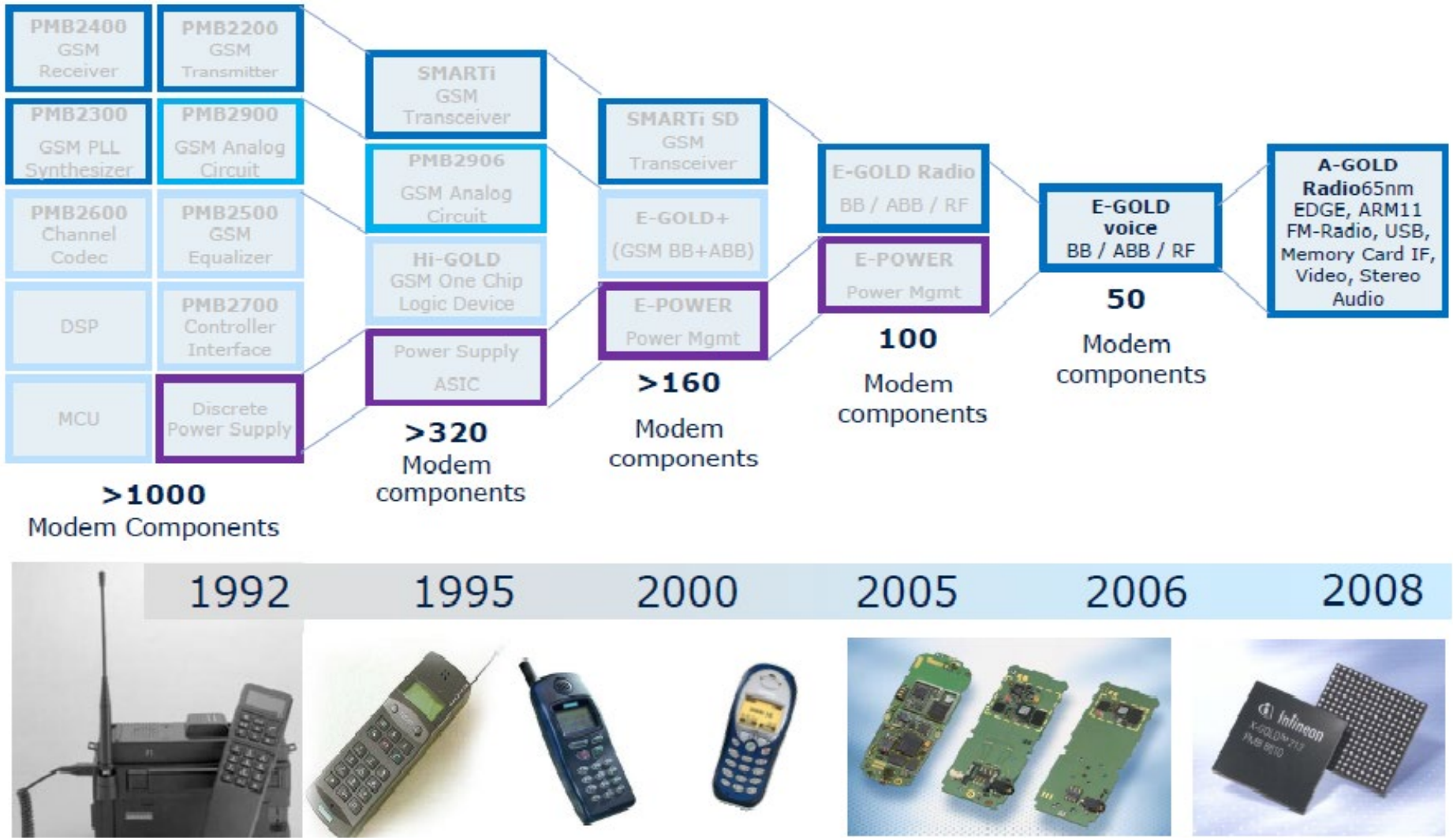
- The power wall
  - We can't reduce voltage further (further voltage lowering makes the transistors too leaky)
  - We can't remove more heat efficiently
- How else can we improve performance?

Power is a challenge for integrated circuits:

- Power must be brought in and distributed around the chip
- Power is dissipated as heat and must be removed

Benefits of Integration: Size

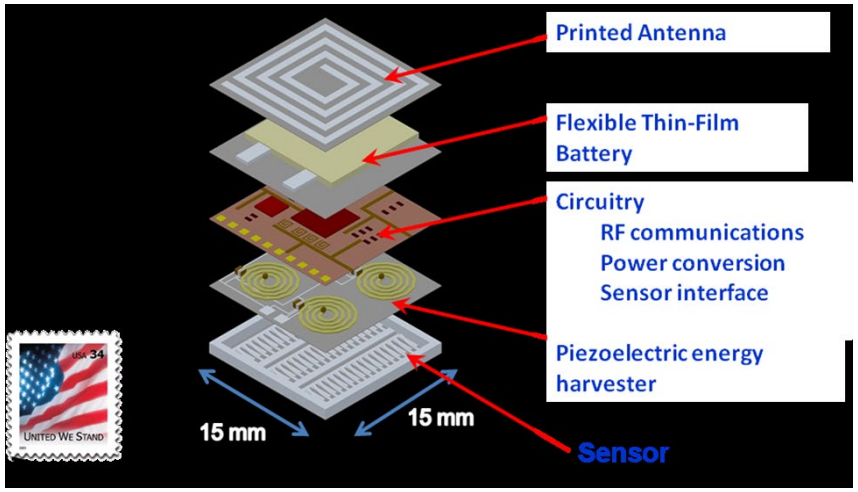
- Impact of SoCs on cell phones



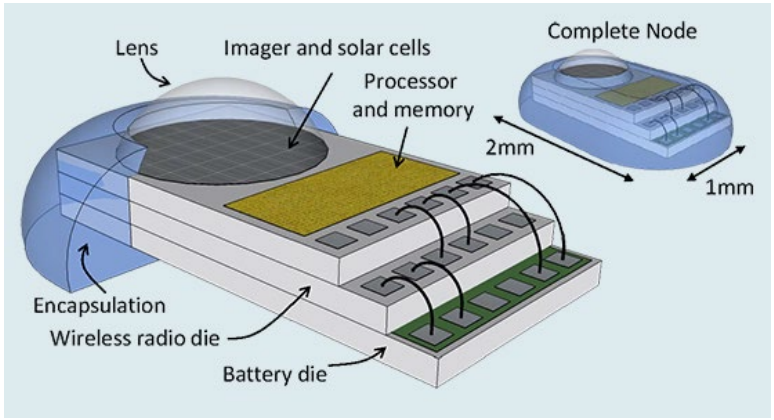
Source: Hermann Eul, Infineon

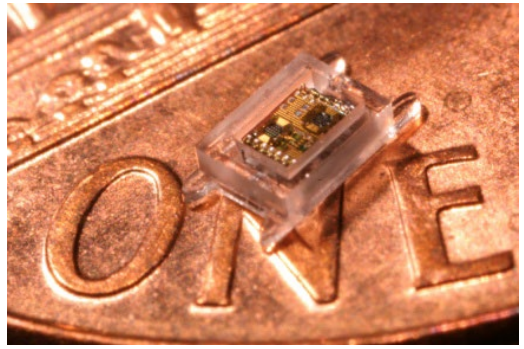
Benefits of Integration: Size/Cost

- Extreme size constraints
  - Integrated sensors for structural (building, bridge, aircraft) monitoring
  - Biomedical implants



Integrated sensor for aircraft structural monitoring - conceptual design (Courtesy Prof. Byunghoo Jung, Prof. Dimitri Peroulis, Purdue)



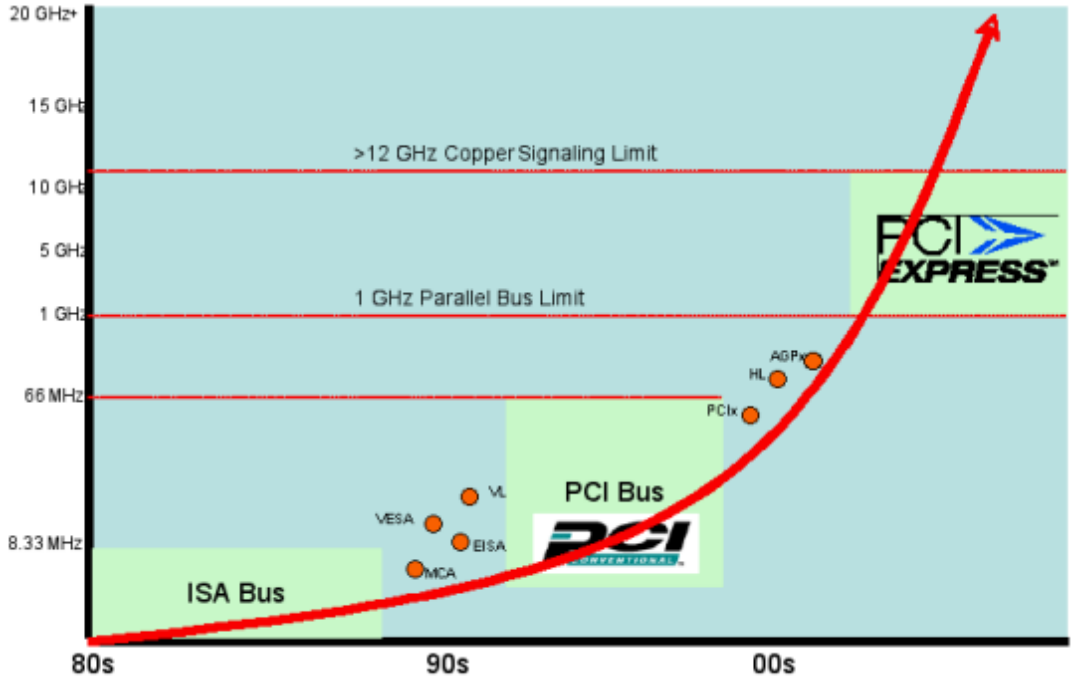


Millimeter-scale computing platforms (Courtesy Profs. David Blauuw and Dennis Sylvester, U. Michigan)

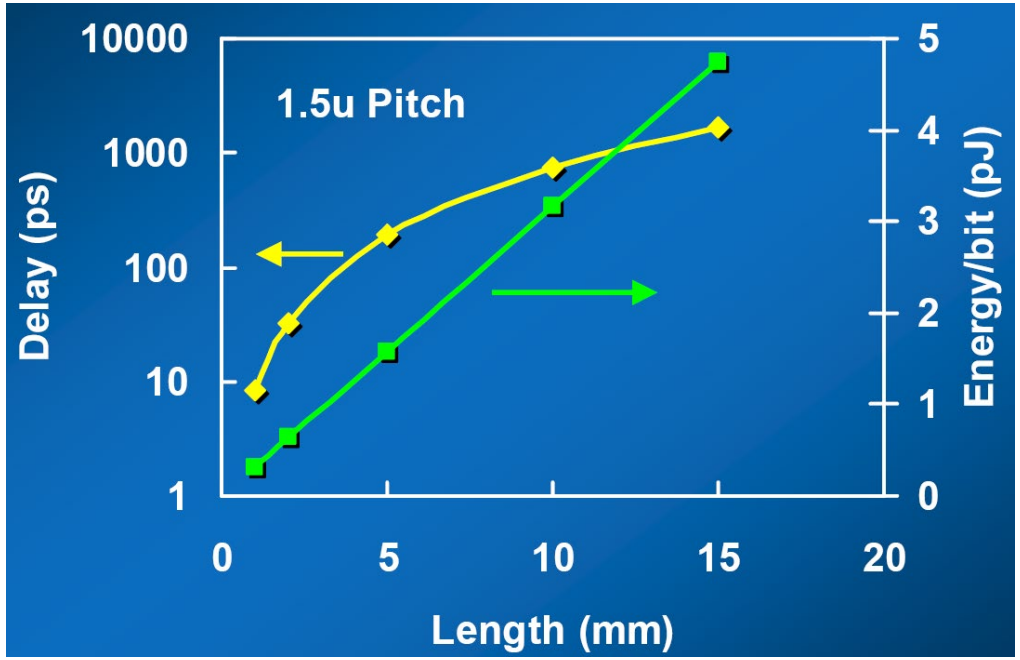
Source: Prof. Anand Raghunathan, Purdue Univ.

Benefits of Integration: Performance/Power

- Integration converts off-chip traffic into on-chip traffic
  - Off-chip communication: ~1-10GB/s, ~1nJ/bit
  - On-chip communication: ~100GB/s-10TB/s, 0.1-1 pJ/bit



PC system-level bus trends



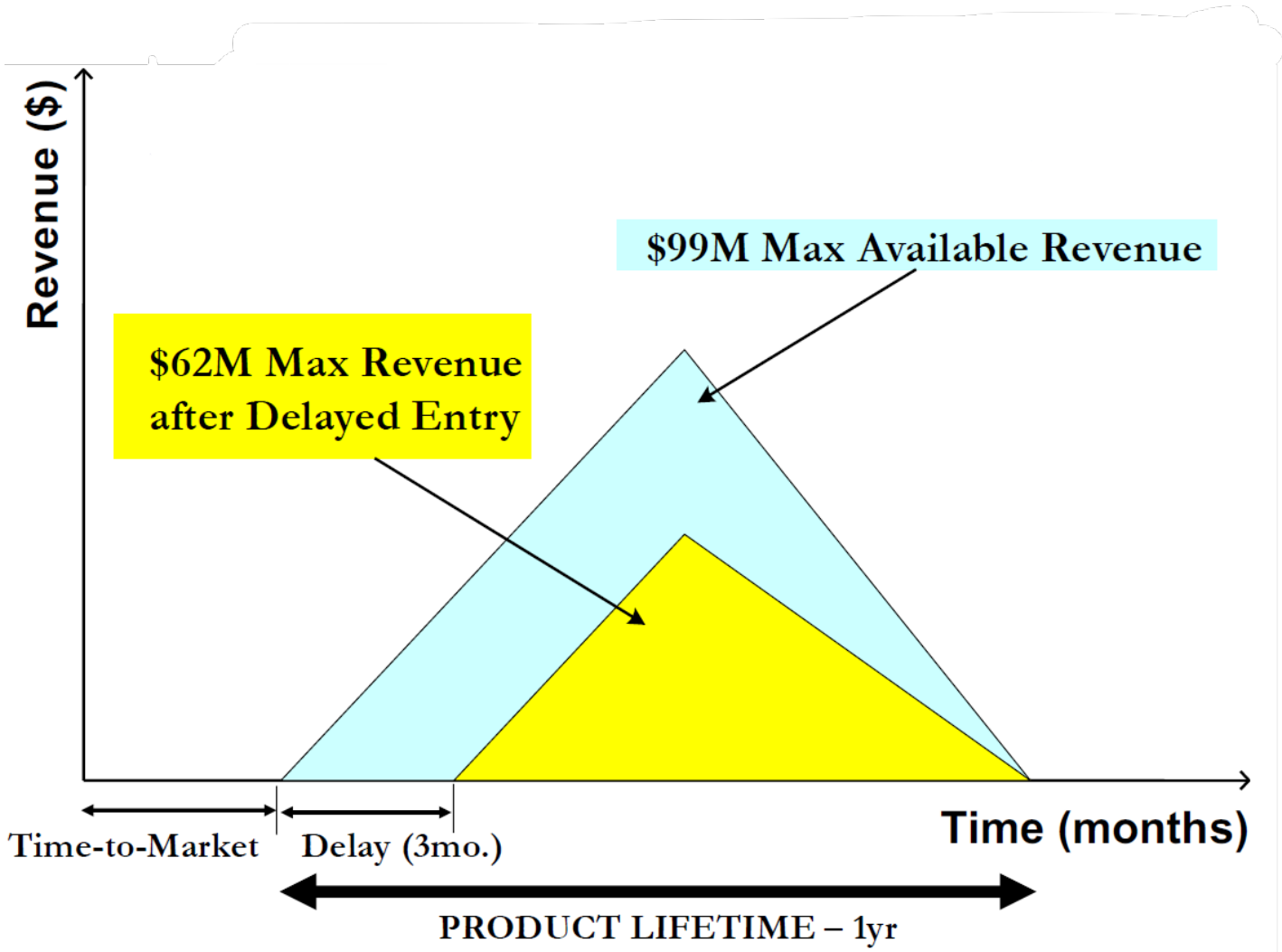
On-chip wire delay and power in 65nm technology (Source: Intel)

Prof. Deming Chen, UIUC

Key SoC Design Challenges

- Which / how many processors?
- How to partition functionality between CPUs and co-processors/accelerators?
- Which functions to accelerate?
- Which type of accelerators?
- How to interconnect the components?
- Re-use legacy / pre-designed components?

Time-to-Market Economics



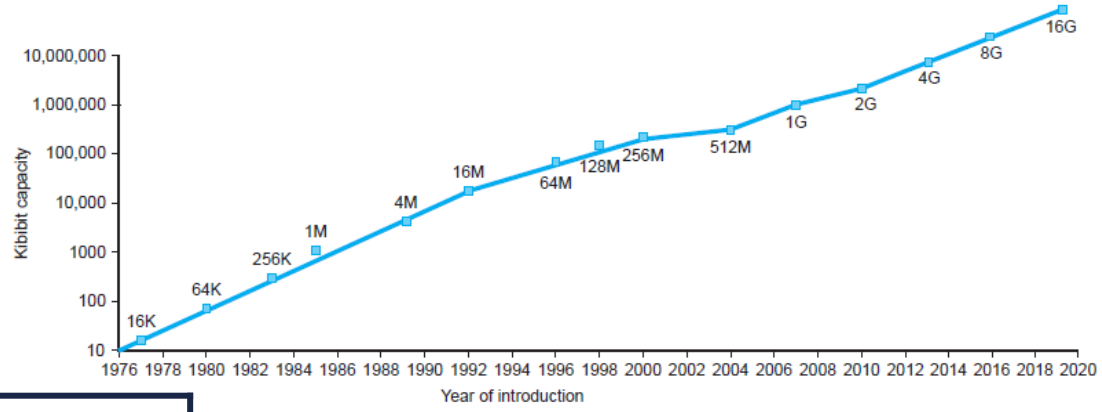
Pittsburgh Digital Greenhouse and Prof. Steve Levitan of Univ. of Pittsburgh.

Key SoC Technologies

- Hardware
  - IC Technology
- Software
  - Processor Technology
- Hardware/Software
  - Hardware/Software Co-Design Technology

Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



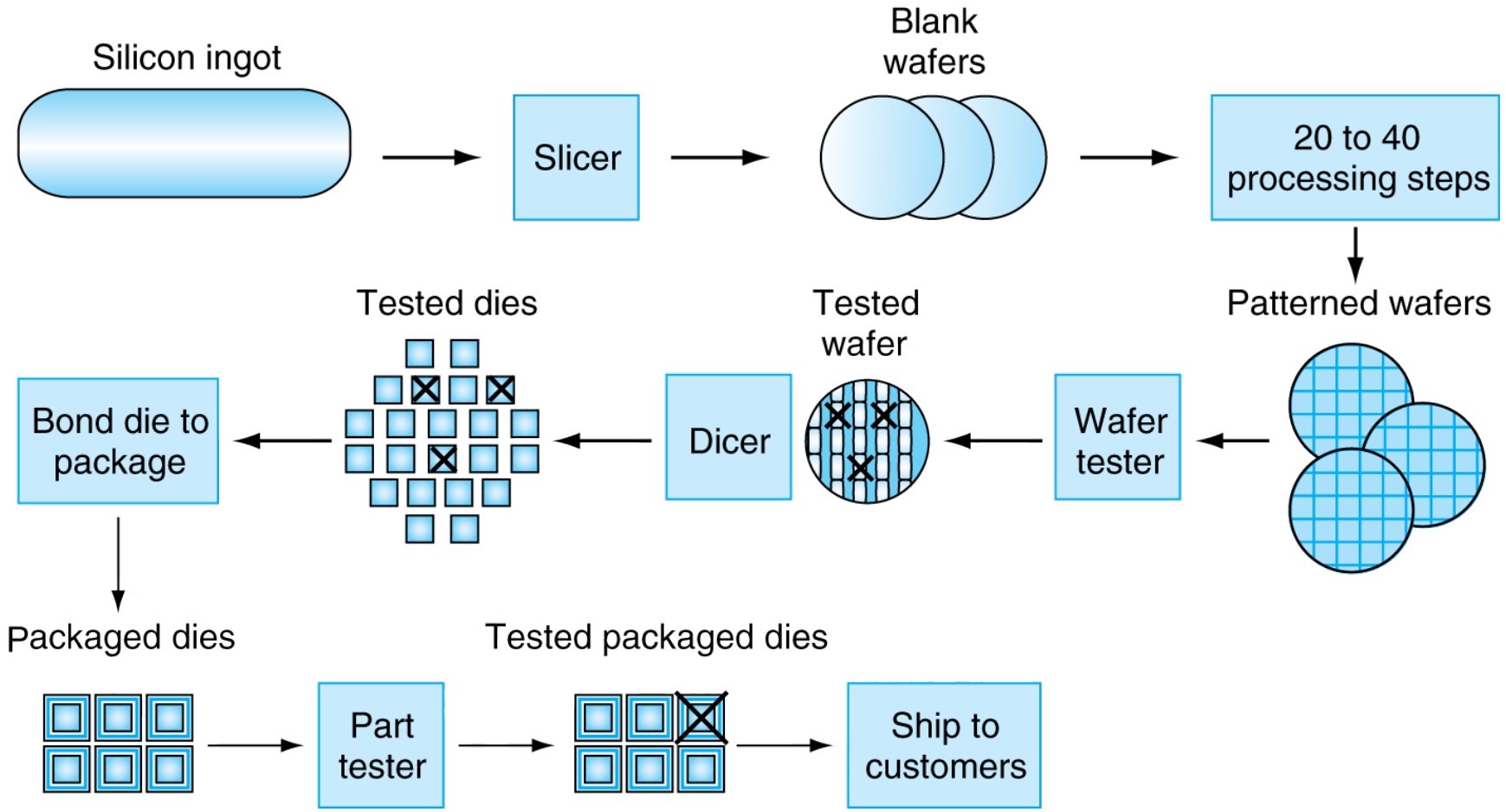
DRAM capacity

Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Semiconductor Technology

- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch (conduct or insulate under specific conditions) => transistors

Manufacturing ICs



Yield: proportion of working dies per wafer

Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

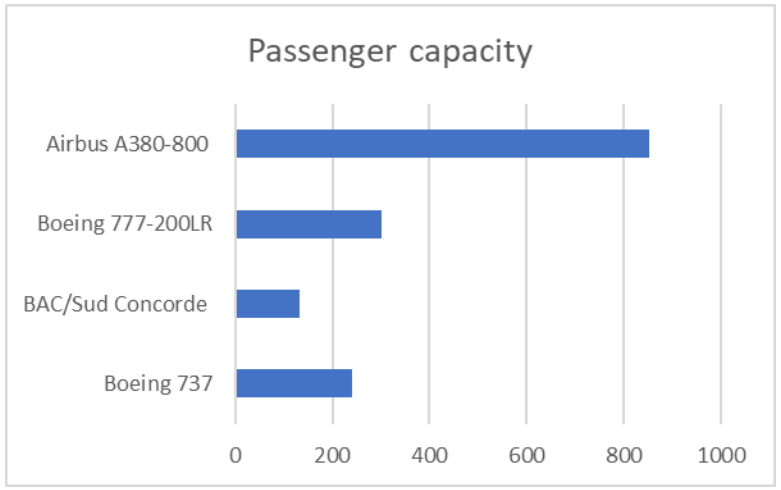
$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}))^N}$$

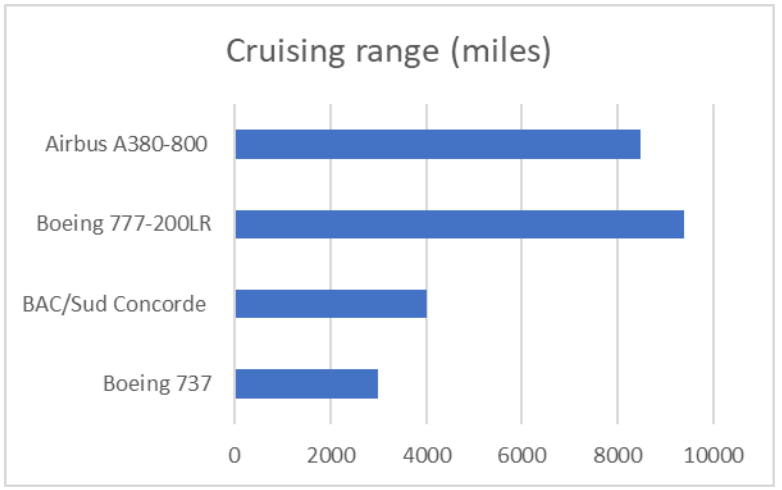
Empirical observations of yields at IC factories;
N related to the number of critical processing steps

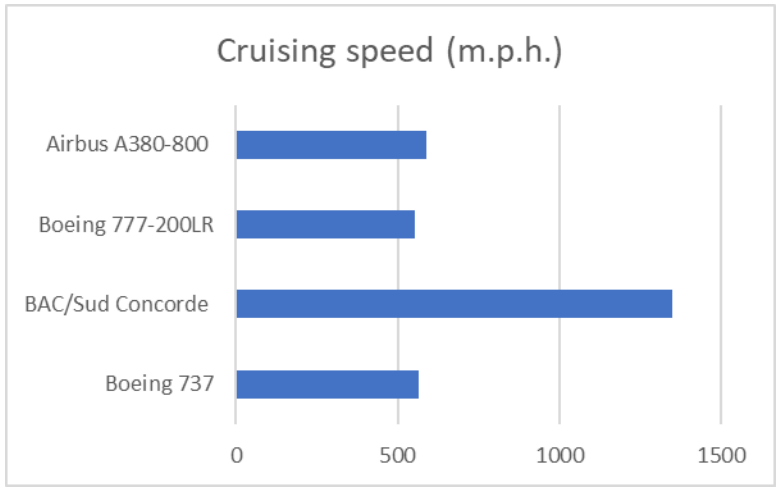
- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

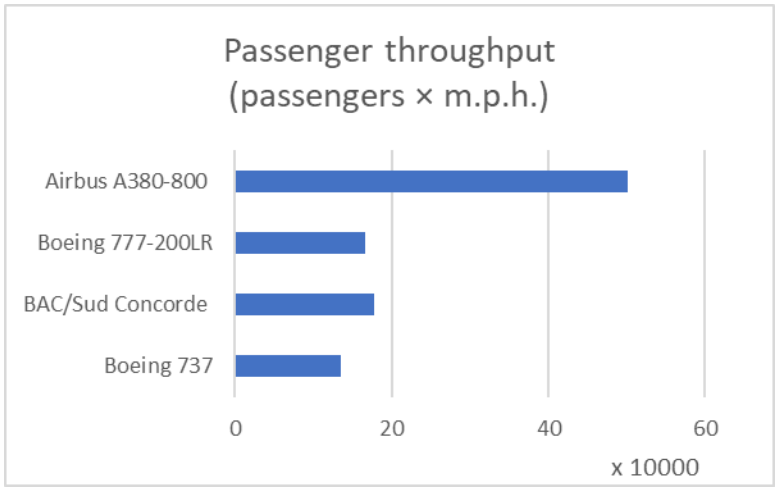
Defining Performance

- Which airplane has the best performance?



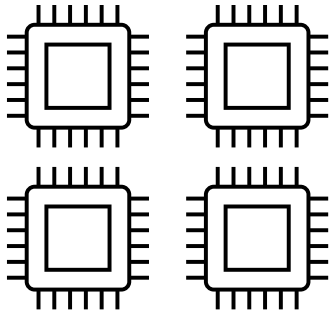






Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?



Measuring Execution Time

- Elapsed time (aka *wall clock time* or *response time*)
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance

- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises ***user*** *CPU time* and ***system*** *CPU time*
  - Different programs are affected differently by CPU and system performance

Processors and Accelerators

- **Hardware accelerator**: “computer hardware designed to perform *specific functions more efficiently* compared to software running on a CPU” (Wikipedia)
- Tradeoff between flexibility and efficiency
- Invest time and money for better performance and efficiency

More flexible
More general

More specialized
More efficient

CPU

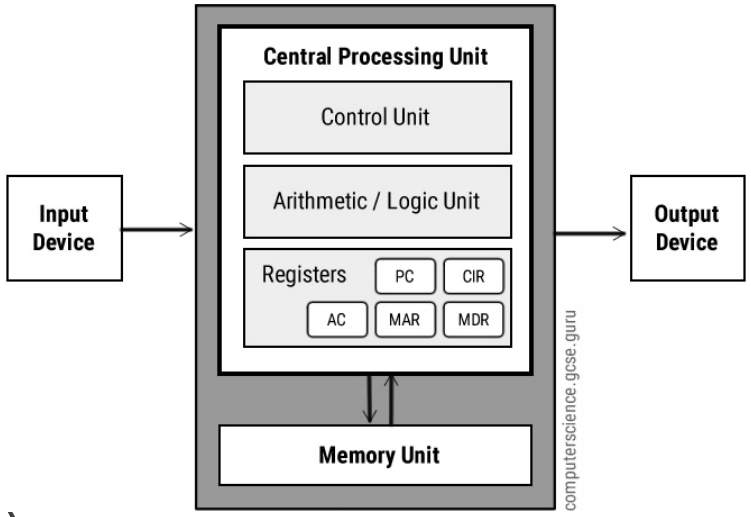
GPU

FPGA
(Field-Programmable Gate Array)

ASIC
(Application-Specific Integrated Circuit)

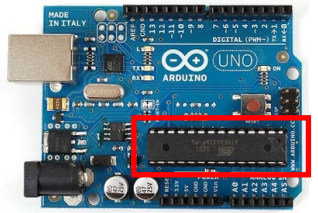
General Purpose Processor

- Programmable processors used in a variety of applications
- Not designed for any specialized purposes
- Central Processing Unit (CPU)
- Architecture (von Neumann)
  - Arithmetic Logic Unit (ALU)
  - Control Unit
  - Registers
  - Memory management unit (MMU)
  - Cache
- Implemented on Integrated Circuit (IC)
  - one or more CPU cores in a single IC chip
- An IC that contains a CPU may also contain memory, peripheral interfaces
  - Microcontrollers and System-on-Chip (SoC)

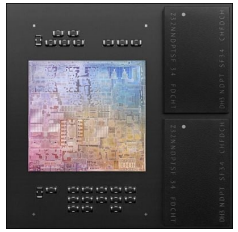


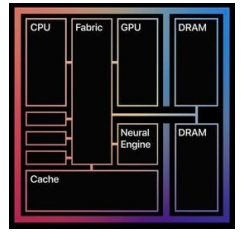
Intel i9-12900K CPU

Qualcomm Snapdragon SoC



Microcontroller Unit (MCU) on an Arduino board





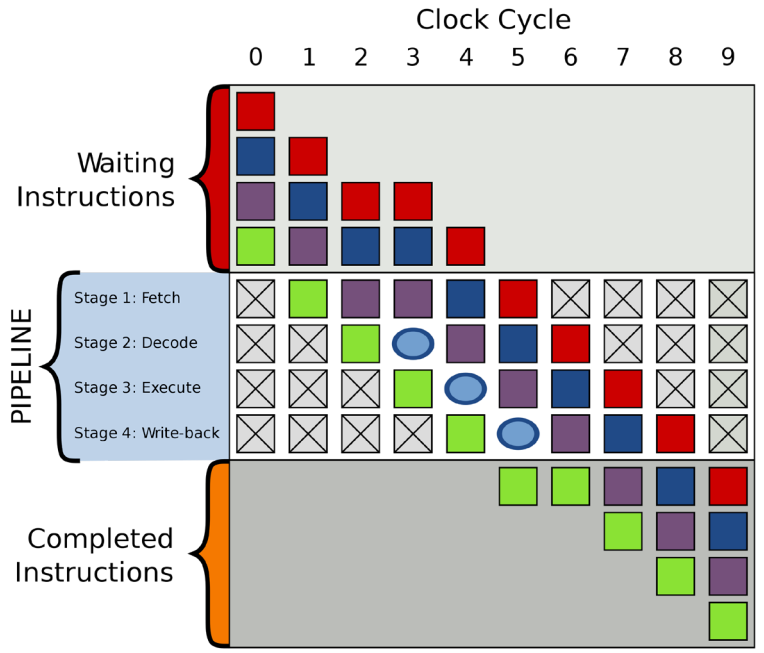
Apple M1 SoC

General Purpose Processor

- Instruction Pipelining
  - A technique for implementing ***instruction-level parallelism (ILP)*** within a processor
  - Pipeline stages (five-stage pipeline case)
    - Instruction Fetch (IF)
    - Instruction Decode (ID)
    - Execute (EX)
    - Memory access (MEM)
    - Register write back (WB)
- Hazards
  - *Data hazard*: values produced from one instruction are not available when needed by a subsequent instruction
  - *Control hazard*: a branch in the control flow makes ambiguous what is the next instruction to fetch
- Pipeline Stall
  - Delay in execution of an instruction to resolve a hazard

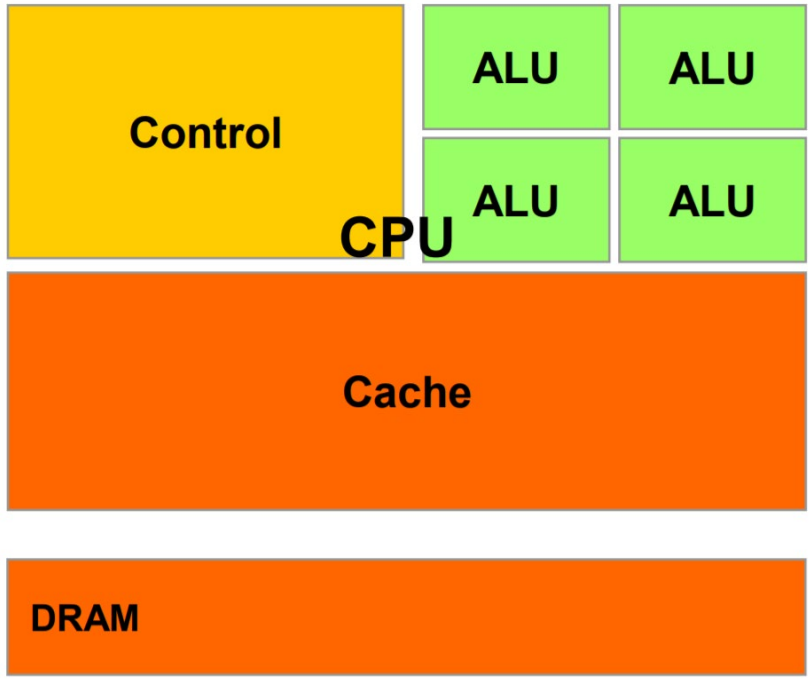
Basic five-stage pipeline

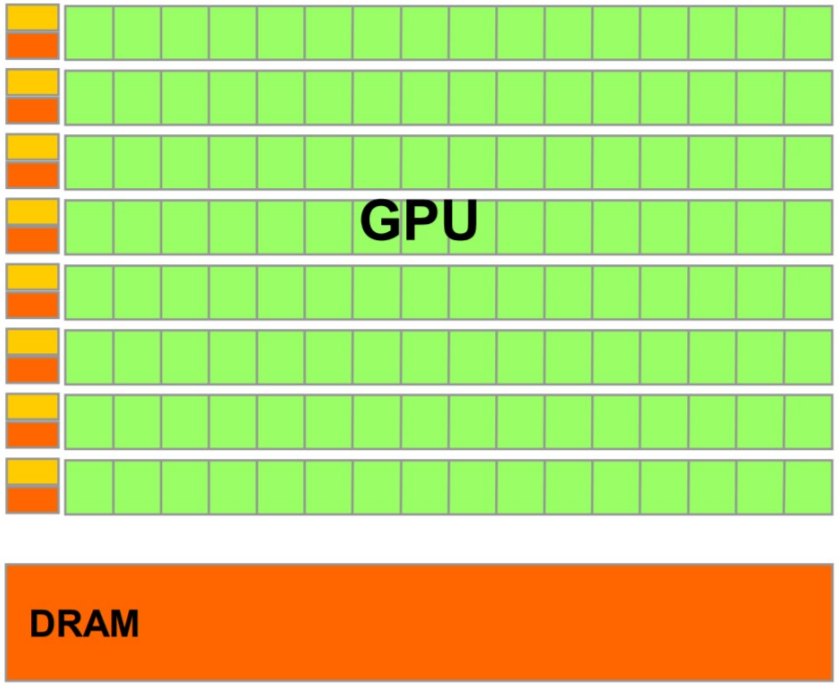
Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX



Graphics Processing Unit (GPU)

- CPUs and GPUs have fundamentally different design philosophies
  - CPUs: ***low***-latency, ***low***-throughput
    - high clock freq., large caches, sophisticated control, powerful ALUs
  - GPUs: ***high***-latency, ***high***-throughput
    - moderate clock freq., small caches, simple control, (many) energy efficient ALUs
    - *Require **massive** number of threads to tolerate latencies* ⇨ ***More specialized in specific applications***

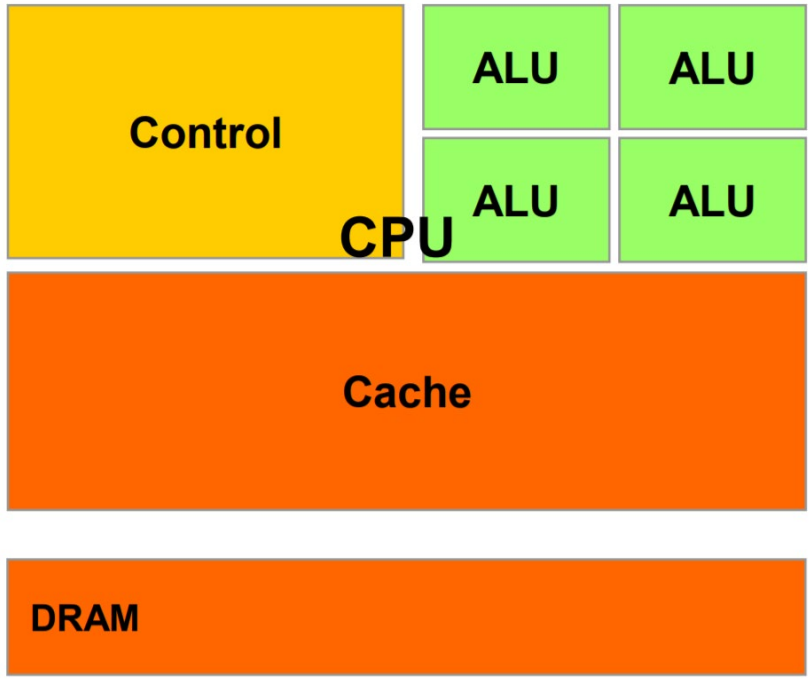


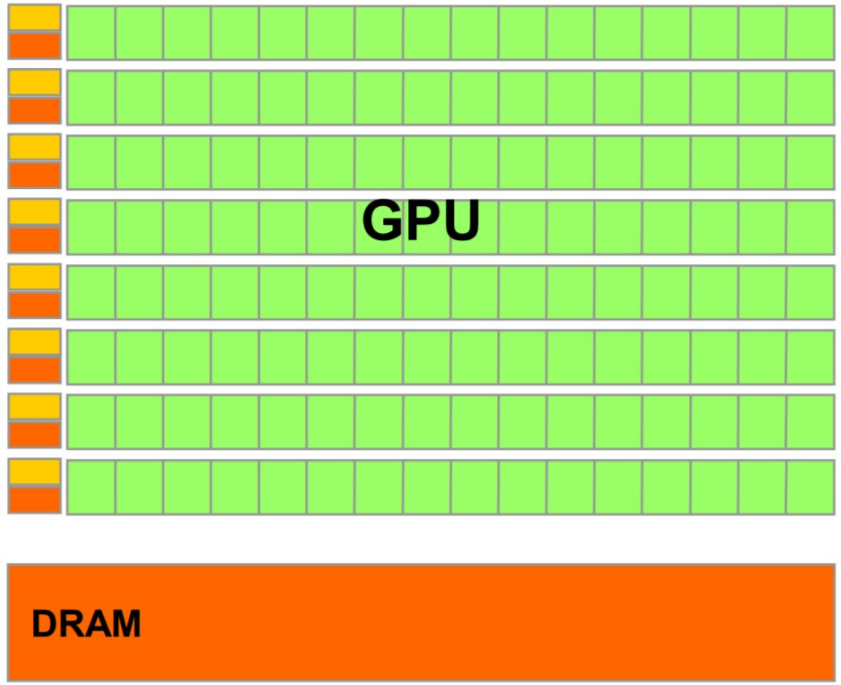


Source: Prof. Wen-mei Hwu, UIUC ECE 408: "Applied Parallel Programming"

Graphics Processing Unit (GPU)

- CPUs and GPUs have fundamentally different design philosophies
  - CPUs: Good for sequential parts where latency matters
    - CPUs can be > 10x faster than GPUs for sequential code
  - GPUs: Good for parallel parts where throughput wins
    - GPUs can be > 10x faster than CPUs for parallel code

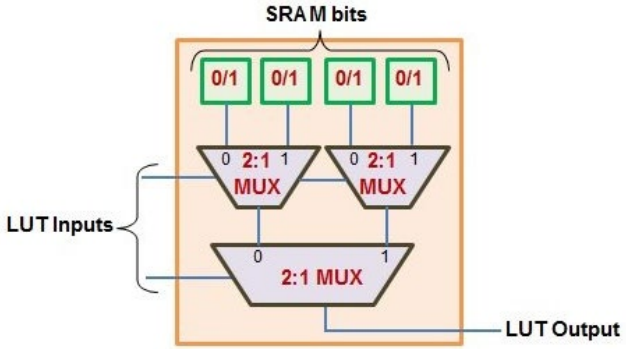


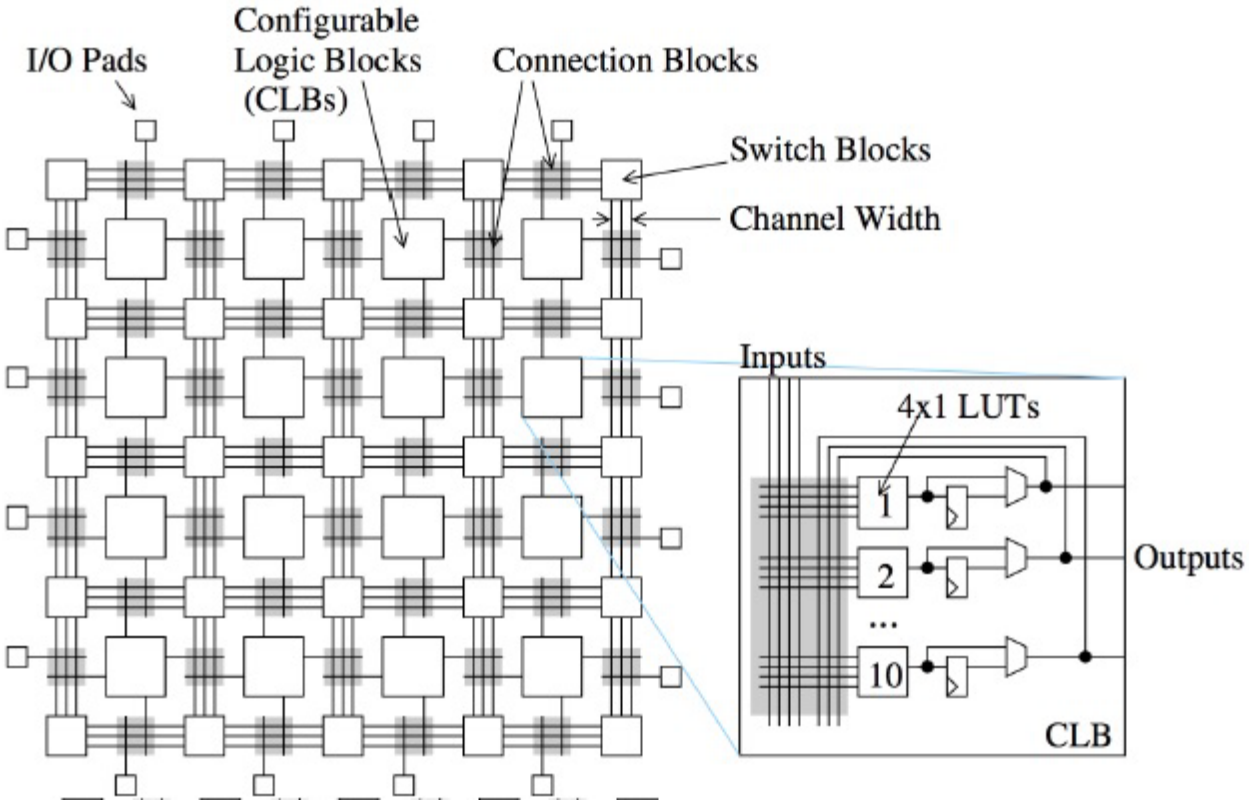


Source: Prof. Wen-mei Hwu, UIUC ECE 408: "Applied Parallel Programming"

Field-Programmable Gate Array (FPGA)

- High-density programmable gate array
- Fully programmable through bit-stream configuration file
  - Programmable Logic
  - Programmable I/O
  - Programmable Interconnects (routing)
- Look-up table (LUT) based combinational logic
  - Can be implemented with SRAM (volatile)

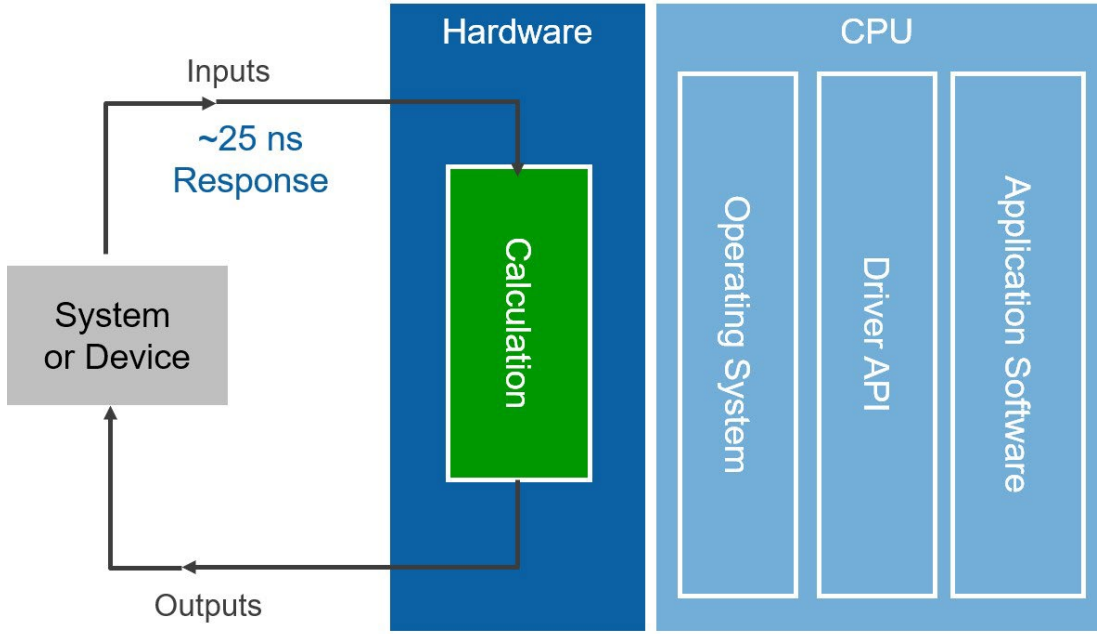




Field-Programmable: An electronic device or embedded system is said to be field-programmable or in-place programmable if its firmware can be modified "in the field", without disassembling the device or returning it to its manufacturer. (Wikipedia)

Field-Programmable Gate Array (FPGA)

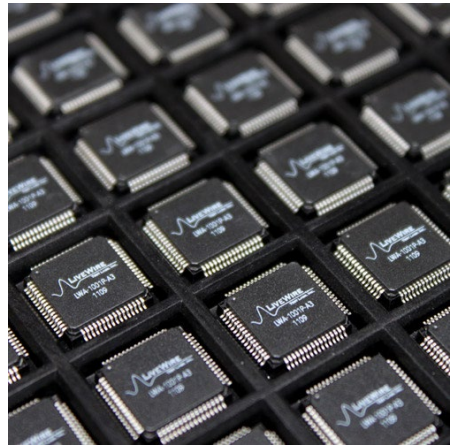
- Massive fine-grained parallelism
- Clock cycle accurate control & compute
- Very low latency, very short response time (benefits from specialization and customization)
  - Applications: real-time video processing, signal processing, high-frequency trading, etc.
- Lower clock frequency (typically ~200 MHz) compared to CPU/GPU/ASIC

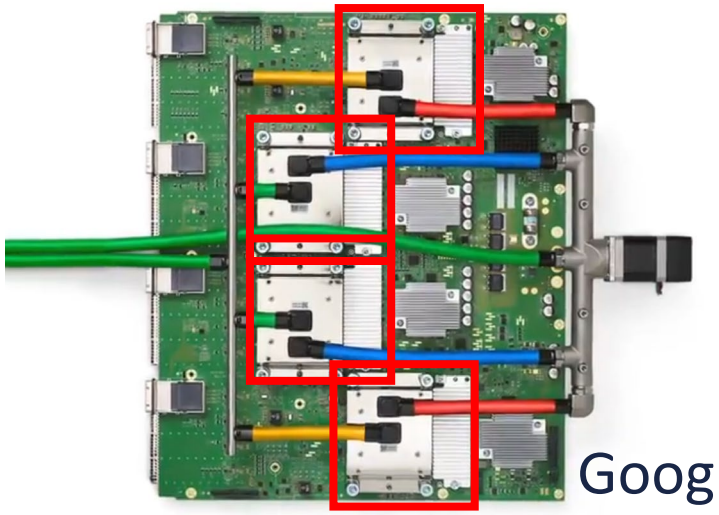


Field-Programmable: An electronic device or embedded system is said to be field-programmable or in-place programmable if its firmware can be modified “in the field”, without disassembling the device or returning it to its manufacturer. (Wikipedia)

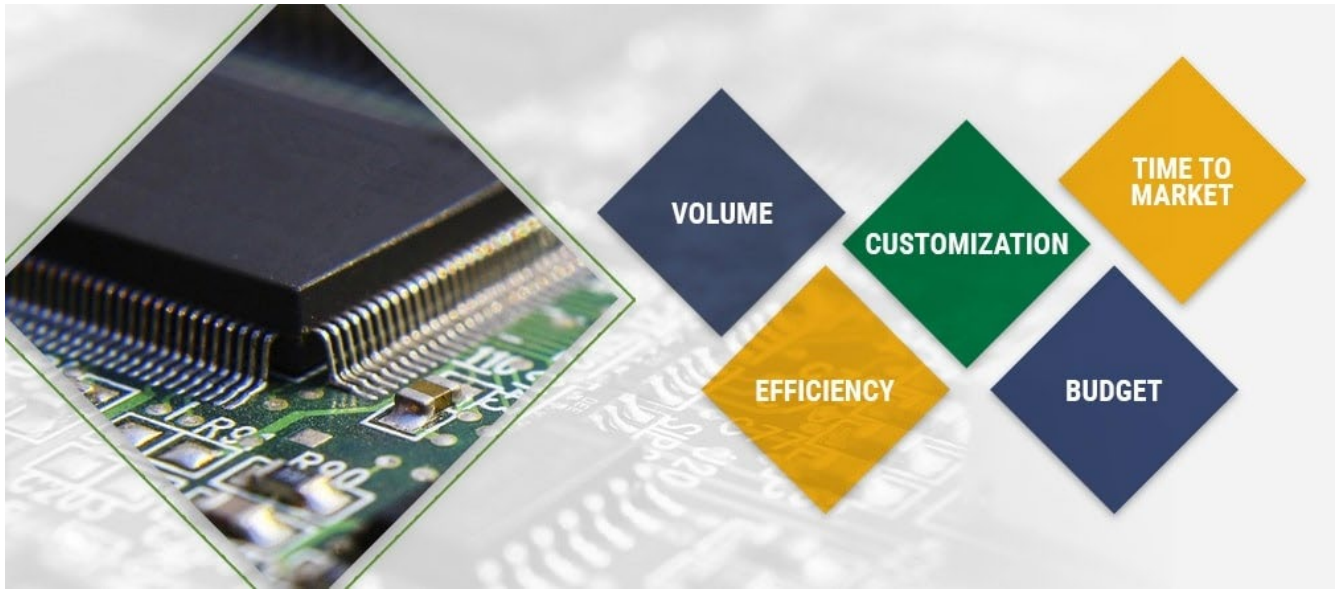
Application-Specific Integrated Circuit (ASIC)

- An IC chip customized for a particular uses
- Cannot be reprogrammed for multiple applications
- Notably more efficiently than FPGAs
- Require a higher initial cost and longer design time compared to FPGA, more cost-effective if produced in large quantities
- Use: permanent applications in electronic devices
- NOTE: ASICs may be controlled with instructions





Google TPU v4



Reference: https://www.asicnorth.com/blog/asic-vs-fpga-difference/

Hardware Accelerators

- **Hardware accelerator**: “computer hardware designed to perform *specific functions more efficiently* compared to software running on a CPU” (Wikipedia)
- Hardware accelerators have been there since early days
- Intel 8087: the first x87 floating-point coprocessor for the 8086 line of microprocessors (announced in 1980)
  - Speed up computations for floating-point arithmetic
- Digital Signal Processor (DSP)
  - Accelerates digital signal processing
- Graphics Processing Unit (GPU)
  - Specialized for graphics processing, now also used for general computing
- FPGA: reconfigurable domain-specific accelerators
- ASIC: customized accelerators

A Taxonomy of Accelerators* *(from host processor's perspective)*

- Granularity: what kinds of computation are offloaded to accelerator?
  - Instruction level: primitives like arithmetic operators, e.g., sqrt, sin/cos, etc.
  - Kernel level: functional units, modules, e.g., matrix multiply, FFT, etc.
  - Application level: accelerates entire applications, e.g., DNN inference, video decoding, etc.

- Coupling (with host): where accelerators are deployed in the system?

  (assumed system hierarchy: pipelined processor core with multiple levels of caches attached to the memory bus and then connected to I/O devices through I/O bus)
  - Part of the processor pipeline
  - Attached to cache
  - Attached to memory bus
  - Attached to the I/O bus

Tightly coupled with host processor
- more design constraints, lower invocation overhead

Loosely coupled with host processor
- less design constraints, higher invocation overhead

*Source: Yakun Sophia Shao and David Brooks, Research Infrastructures for Hardware Accelerators, 2015

A Taxonomy of Accelerators* *(from host processor's perspective)*

	Part of the Pipeline	Attached to Cache	Attached to the Memory Bus	Attached to the I/O Bus
Instruction-Level	**FPU, SIMD,** DySER [58],	Hwacha [76, 95, 121], CHARM [43, 44],		
Kernel-Level	NPU [52], 10x10 [40], Convolution Engine [98], H.264 [61],	SNNAP [91], C-Cores [119],	Database [35], Q100 [126], LINQits [41], AccStore [86],	
Application-Level	**x86 AES [18], Oracle/Cavium Crypto Acc [11, 69],**	Key-Value Stores [87], Memcached [80],	Sonic3D [103], DianNao [37, 38], HARP [125], **TI OMAP5 [16], IBM PowerEN [71], IBM POWER7+ [30],**	**GPU, Catapult [97], IBM Power8 CAPI Acc [4],**

*Source: Yakun Sophia Shao and David Brooks, Research Infrastructures for Hardware Accelerators, 2015

Accelerator Design

- What to accelerate?
  - Decide the operational specifications of the hardware accelerator
  - Profile software applications
  - Determine the critical path/bottleneck, and frequently used kernels or functions
- How to accelerate?
  - Architecture of the accelerator
  - Memory hierarchy and I/O interfaces
  - CPU-accelerator interfaces
  - Programming interfaces
- Acceleration goals/requirements/constraints?
  - Maximum latency
  - Minimum throughput
  - Maximum power consumption
  - Cost, time to market, etc.

Accelerator Design

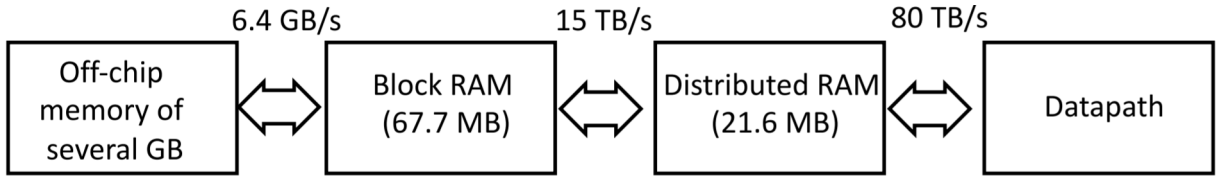
A few examples of choices in hardware accelerator design

- Types of parallelism exploited
  - Fine-grained vs coarse-grained
  - Data parallel vs task parallel
- Optimized for high throughput vs low latency
  - E.g., optimizing number of tasks completed per unit of time, OR, execution time of a single task
- Memory organization
- External interfaces
- On-chip memory usage, data buffering schemes

Basic Concepts

- Latency
  - Time required to perform certain task or to produce certain result.
  - Measured in units of time, e.g., hours, minutes, seconds, nanoseconds, or clock cycles
  - Applications that need low latency: object detection/tracking, DNN inference, etc.

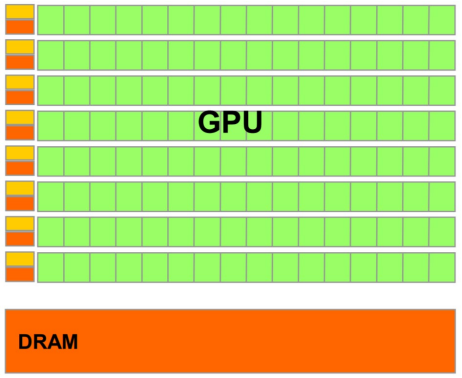
- Throughput
  - The number of tasks or results produced per unit of time
  - Memory bandwidth: how much data is moved through memory interface per unit time. MB/s or GB/s
  - Computational throughput: how much computation is done per unit time. FLOP/s, OP/s, IOP/s
  - Applications that need high throughput: image/video post-processing, DNN training, etc.

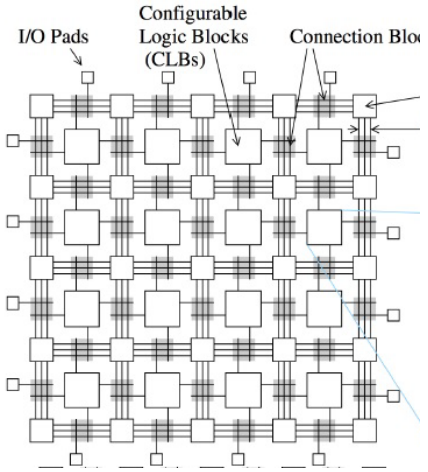


Bandwidth/memory distribution in Xilinx Virtex-7 FPGA
(source: F. Siddiqui et al, "FPGA-Based Processor Acceleration for Image Processing Applications")

Parallelism

- Why are accelerators faster?
  - Exploit the parallelism in kernels/applications





- Types of parallelism
  - Fine-grained (low level) vs coarse-grained (high level)
    - Instruction level
    - Thread level
    - Task level
    - Data level
    - … (name your levels)
  - Data parallel vs task parallel

Parallel Hardware Design

- Consider vector addition:

```
for (i = 0; i < N; i++)
    C[i] = A[i] + B[i];
```

- No data dependences between loop iterations
- Explicit data parallelism in this example
- We could instantiate K parallel adders
  - Speedup = N/K
  - *Can we really achieve N/K speedup?*

Parallel Hardware Design

- Parallel processing units come with a cost
  - More area
  - More power consumption
  - Higher complexity in place & route (could lead to worse timing)
- In our vector addition example
  - In each loop iteration: 2 reads and 1 write for 1 add
  - Assume all values are 32-bit floating-point numbers, that requires reading 8 bytes of data per add

```
for (i = 0; i < N; i++)
    C[i] = A[i] + B[i];
```

- *How much memory bandwidth we need for supplying input data to K-wide adder? (not considering writes; assume adds are single cycle)*
  - *8*K bytes per clock cycle*

Parallel Hardware Design

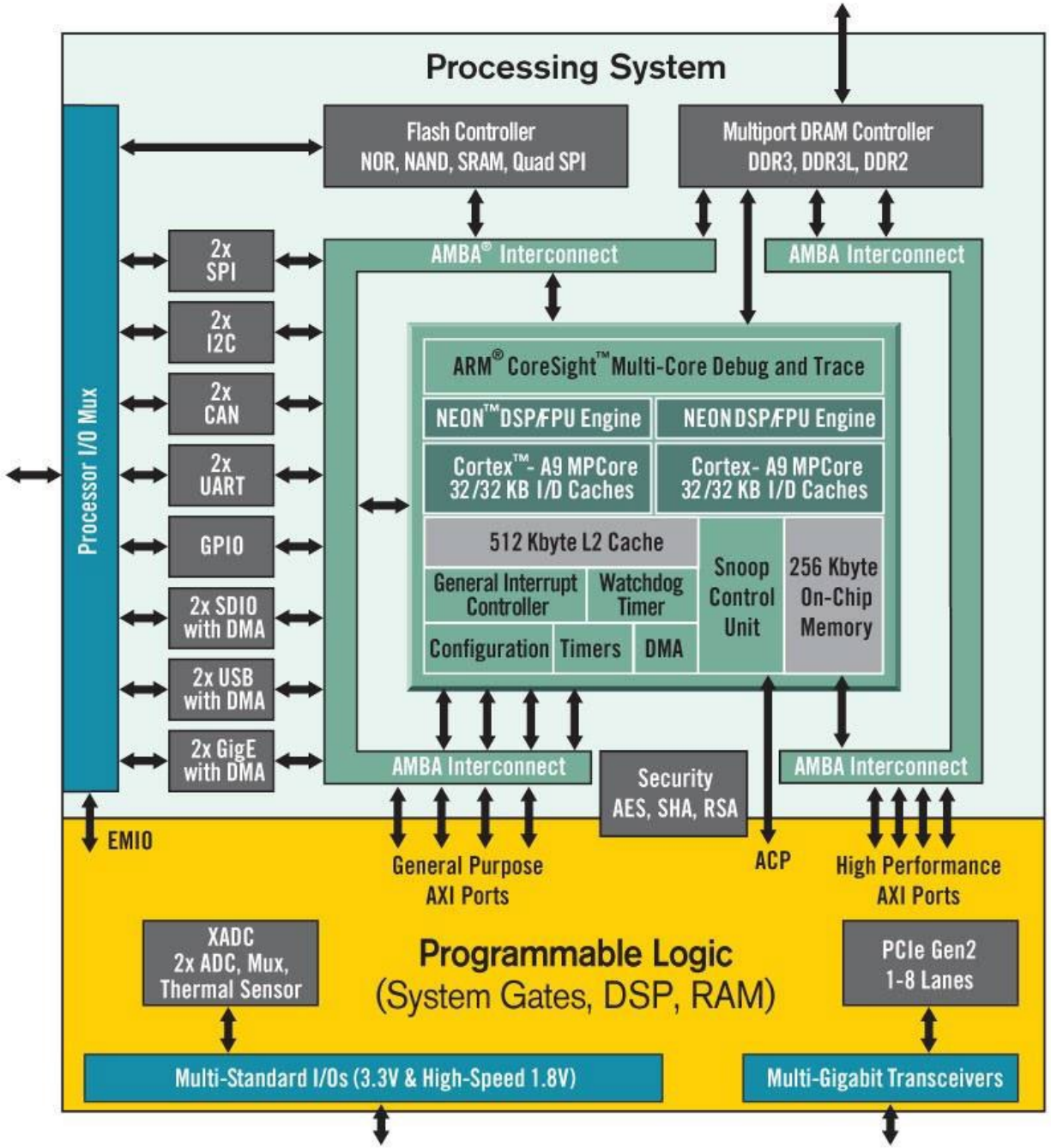
- For a specific platform, an application can be *Compute Bound* or *Memory Bound*
  - "Compute" to "Memory" ratio:

$$\frac{\text{Number of operations (FLOPs)}}{\text{Data transfered through memory for the operations (Bytes)}}$$

- Understand the nature of the application and optimize the design accordingly
- Some design techniques can be used to change the "Compute" to "Memory" ratio
  - Example 1: increase data reuse rate using on-chip memory (increase the ratio)
  - Example 2: re-compute intermediate results without write backs (increase the ratio)
  - Example 3: Write back intermediate results immediately (save on-chip memory, decrease the ratio)

Interface Choices

- How do data move in and out of the accelerator?
- What are the bandwidths needed for the interfaces?



Courses I teach

UCI EECS 112: Organization of Digital Computers

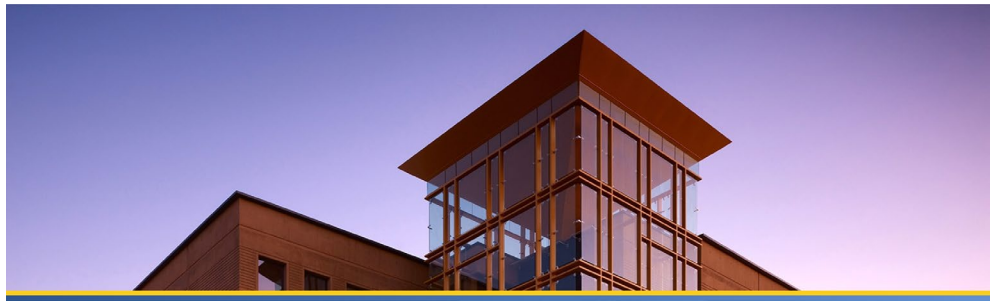
EECS 112: Organization of Digital Computers

Sitao Huang
sitaoh@uci.edu

UCI EECS 221: *Languages and Compilers for Hardware Accelerators*

EECS 221:
Languages and Compilers for Hardware Accelerators

Sitao Huang
sitaoh@uci.edu



UCI ECPS 209: CPS Case Studies

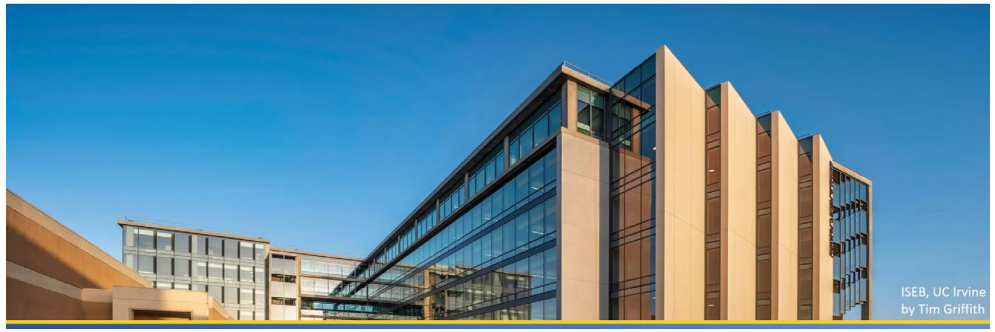
ECPS 209:
CPS Case Studies – GPU Parallel Programming

Sitao Huang
sitaoh@uci.edu

UCI EECS 298: System-on-Chip Design

EECS 298: System-on-Chip Design

Sitao Huang
sitaoh@uci.edu



AMD Xilinx FPGA Design Tools

Vitis Unified Software Platform

- **Option 1**: UCI Engineering Remote Labs:
  - Remote Windows machines
  - https://laptops.eng.uci.edu/computer-labs/remote-labs
- **Option 2**: UCI EECS Instructional Servers
  - Servers: laguna.eecs.uci.edu, bondi.eecs.uci.edu, crystalcove.eecs.uci.edu
  - Setup X11 display server (Xming for Windows and XQuartz for macOS)
  - SSH to any of these servers with your UCI credentials
    - `ssh -X yourUCInetID@laguna.uci.edu`
  - Xilinx tools installed under: `/ecelib/eceware/xilinx_2022.2/`
    - Initialization: `source /ecelib/eceware/xilinx_2022.2/Vitis/2022.2/settings64.csh`
    - Start Vitis HLS: `vitis_hls &`
- **Option 3**: Install Vitis tools on your own computer (>100 GB)
  - https://www.xilinx.com/support/download.html

EECS 298:
System-on-Chip Design

Sitao Huang, sitaoh@uci.edu