

Introduction to Computer Architecture

Chapter 1

Performance

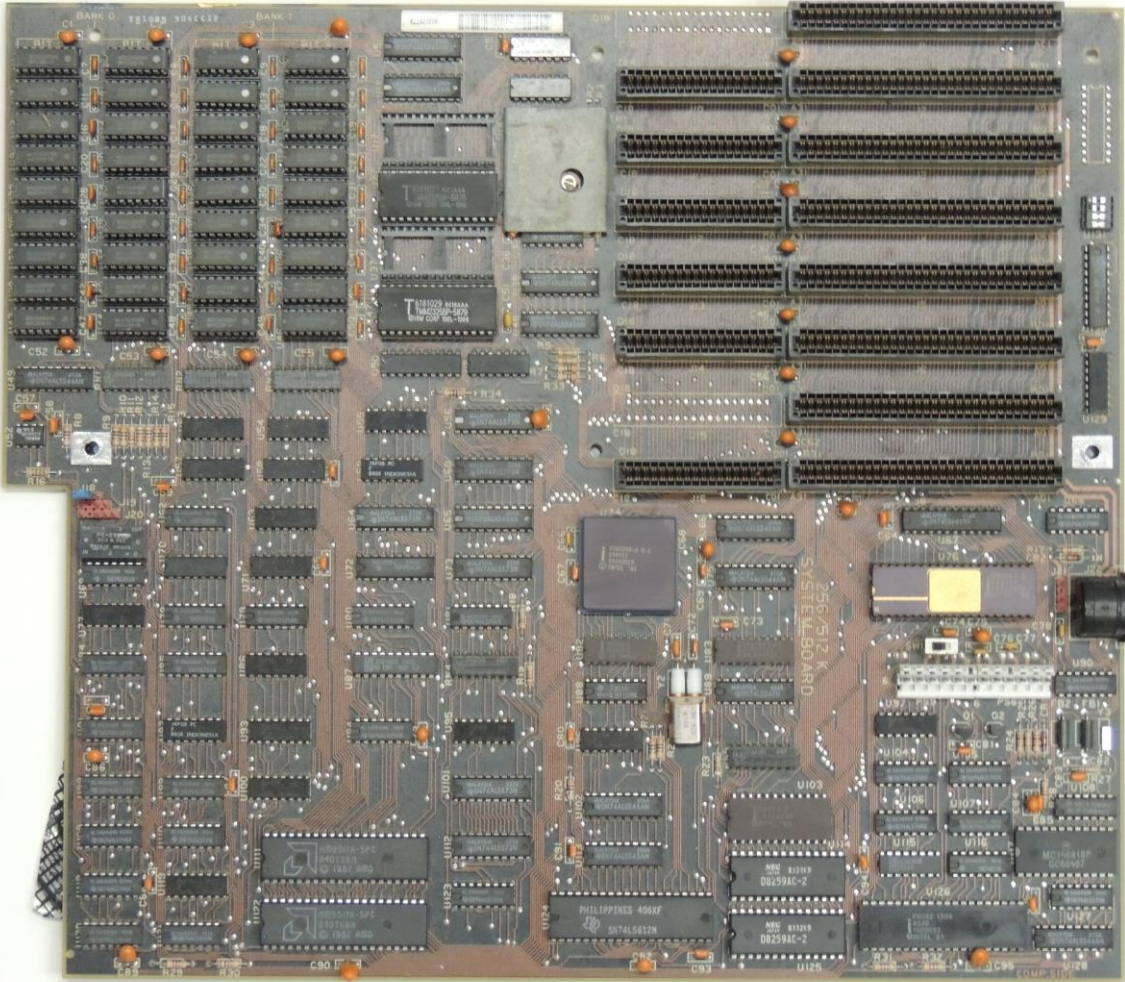
Hyungmin Cho

Department of Computer Science and Engineering
Sungkyunkwan University

(SEMICONDUCTOR) TECHNOLOGIES

Old Computer Systems

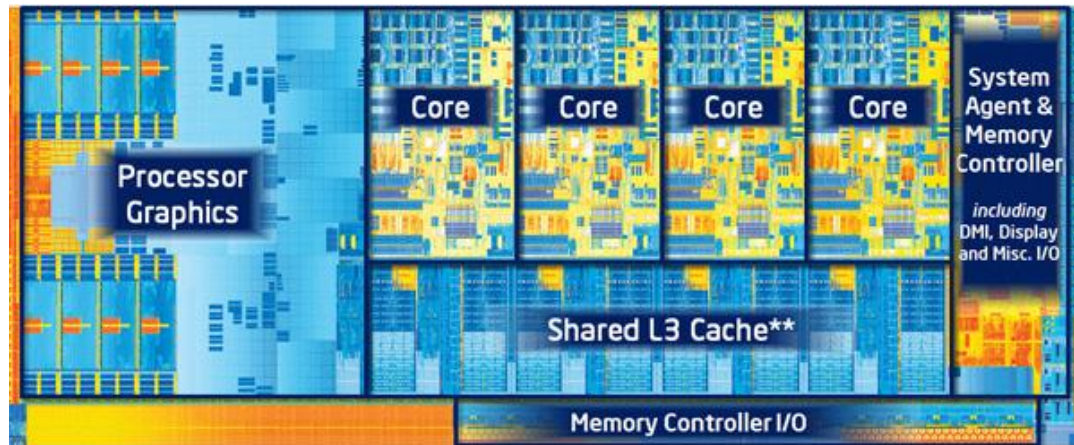
■ 1980s IBM AT (“286” computer) Personal Computer Board



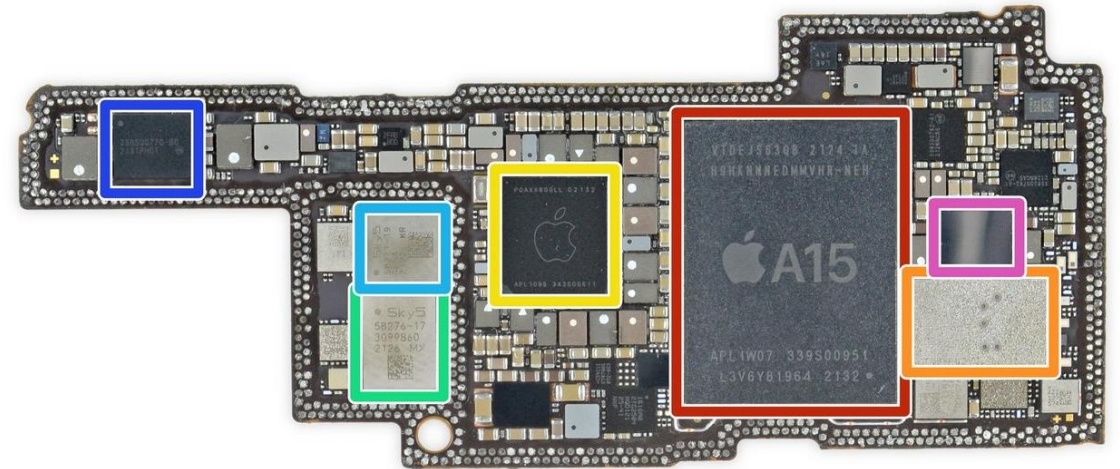
- Numerous individual chips per function
 - ❖ Can't fit many transistors inside a single IC
- Drawbacks
 - ❖ Chip-to-chip data communication : Slow
 - ❖ High power consumption
 - ❖ Expensive \$\$
 - ❖ Low reliability

System-on-Chip

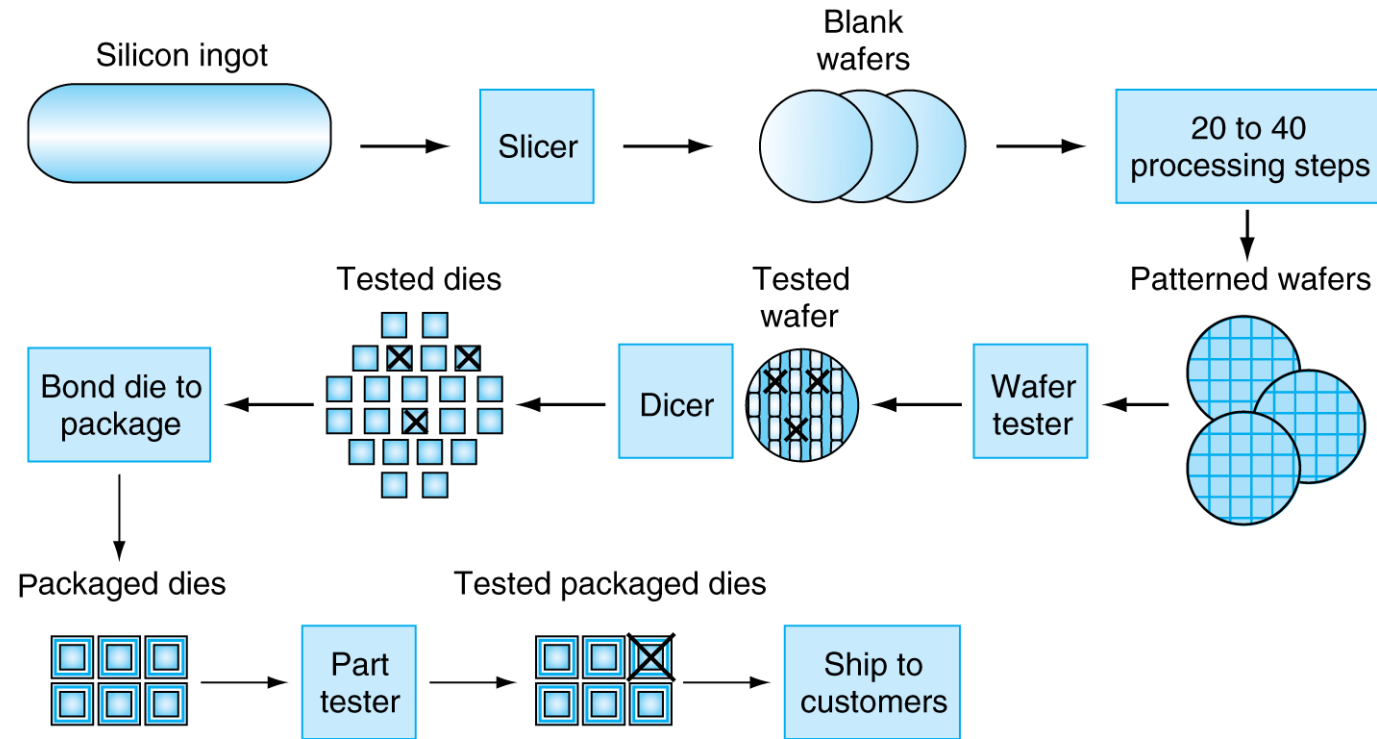
- Intel i7 “Ivy bridge”



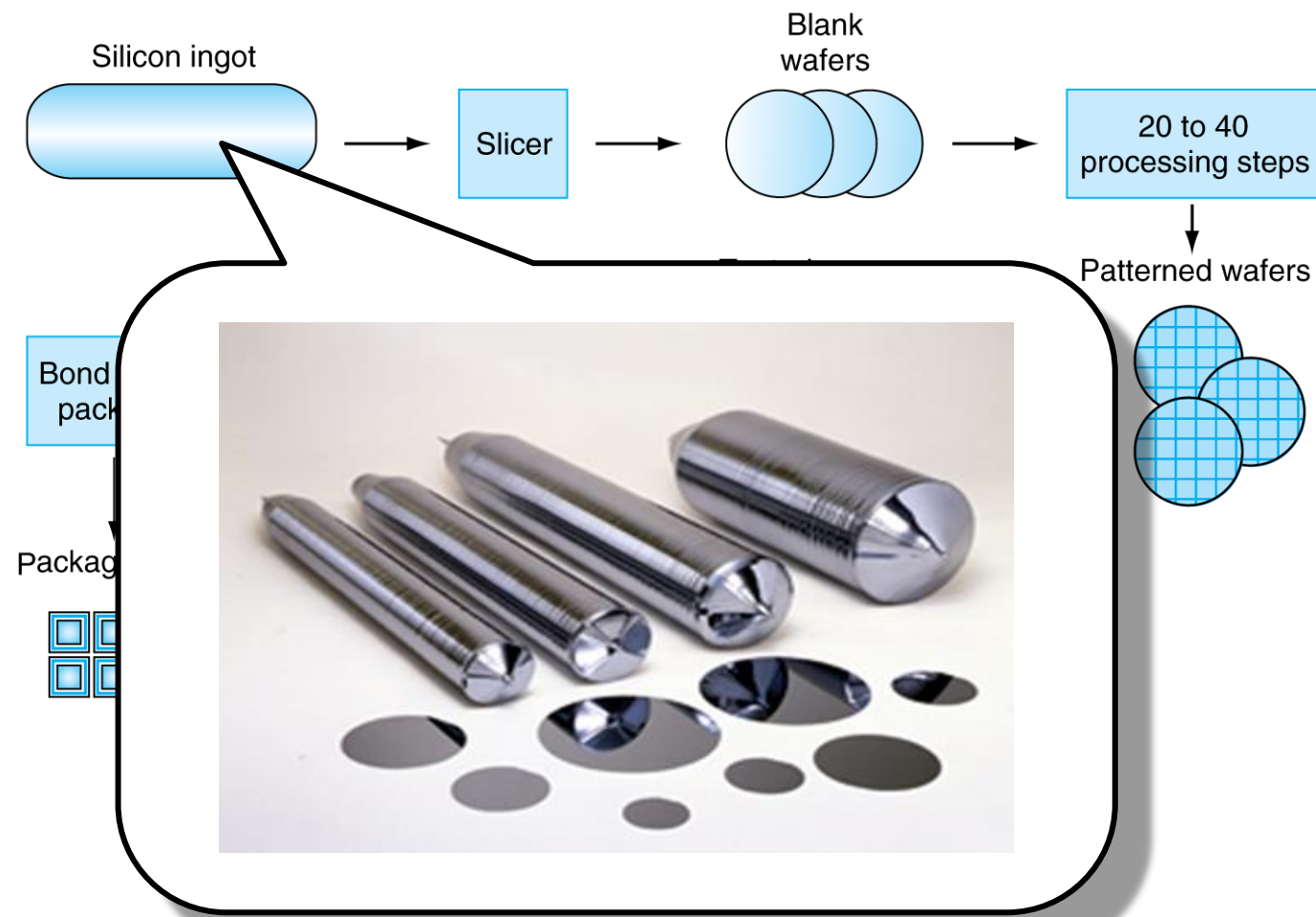
- iPhone 13 Pro



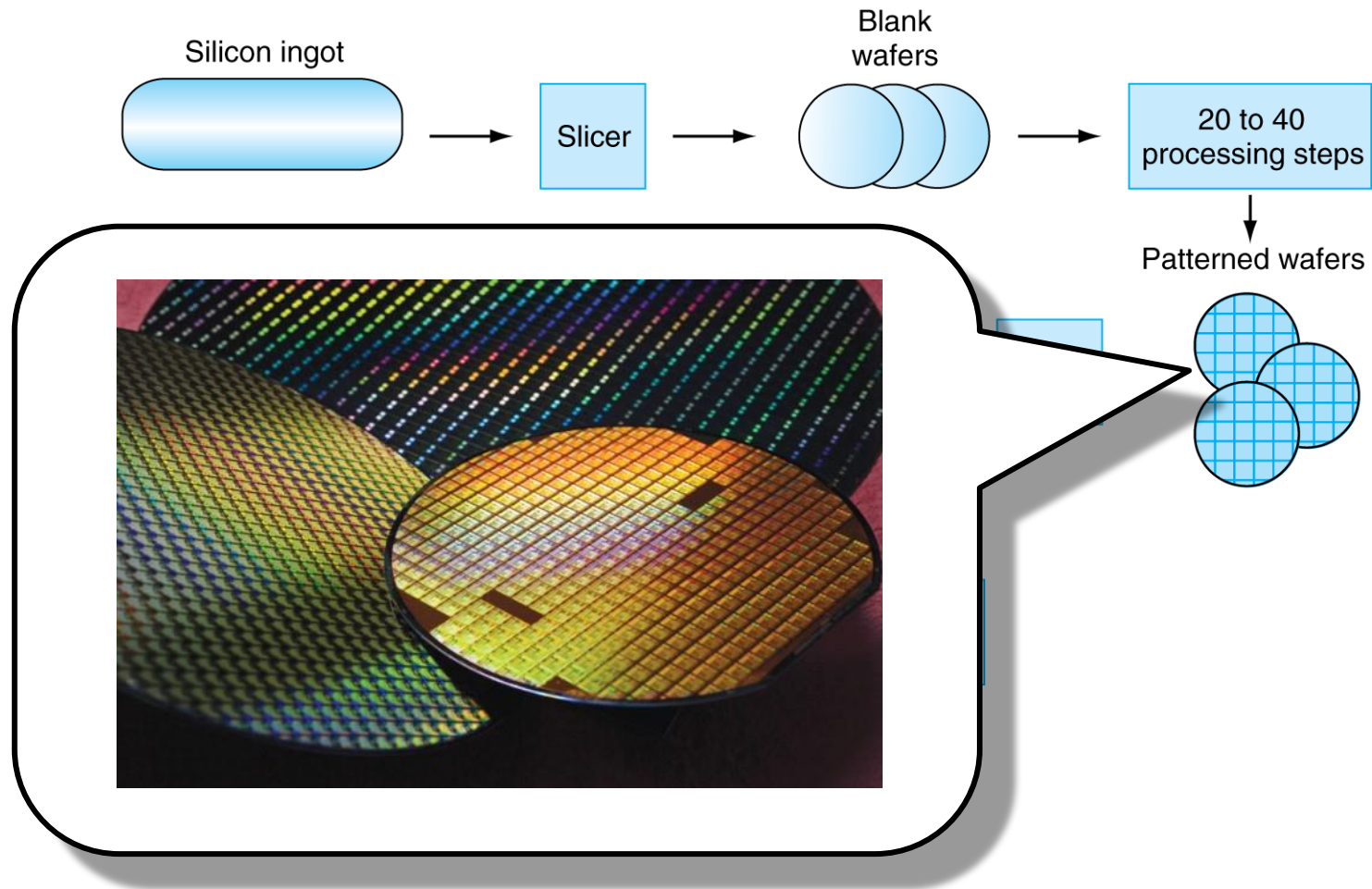
Manufacturing Integrated Circuits (ICs)



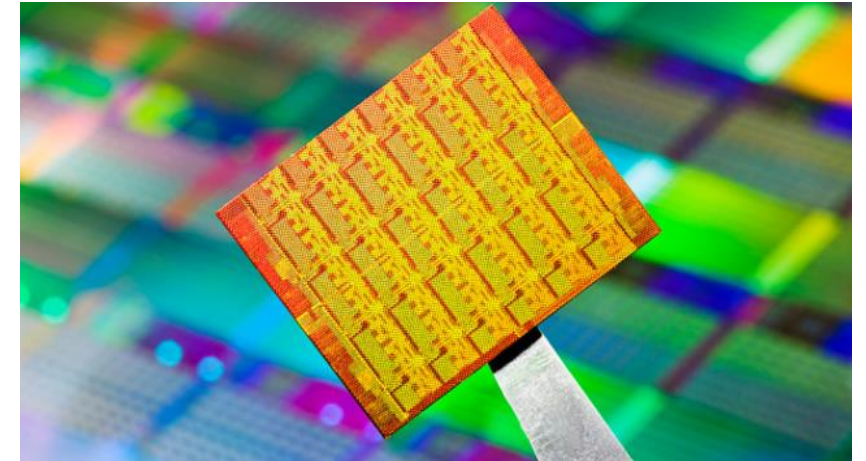
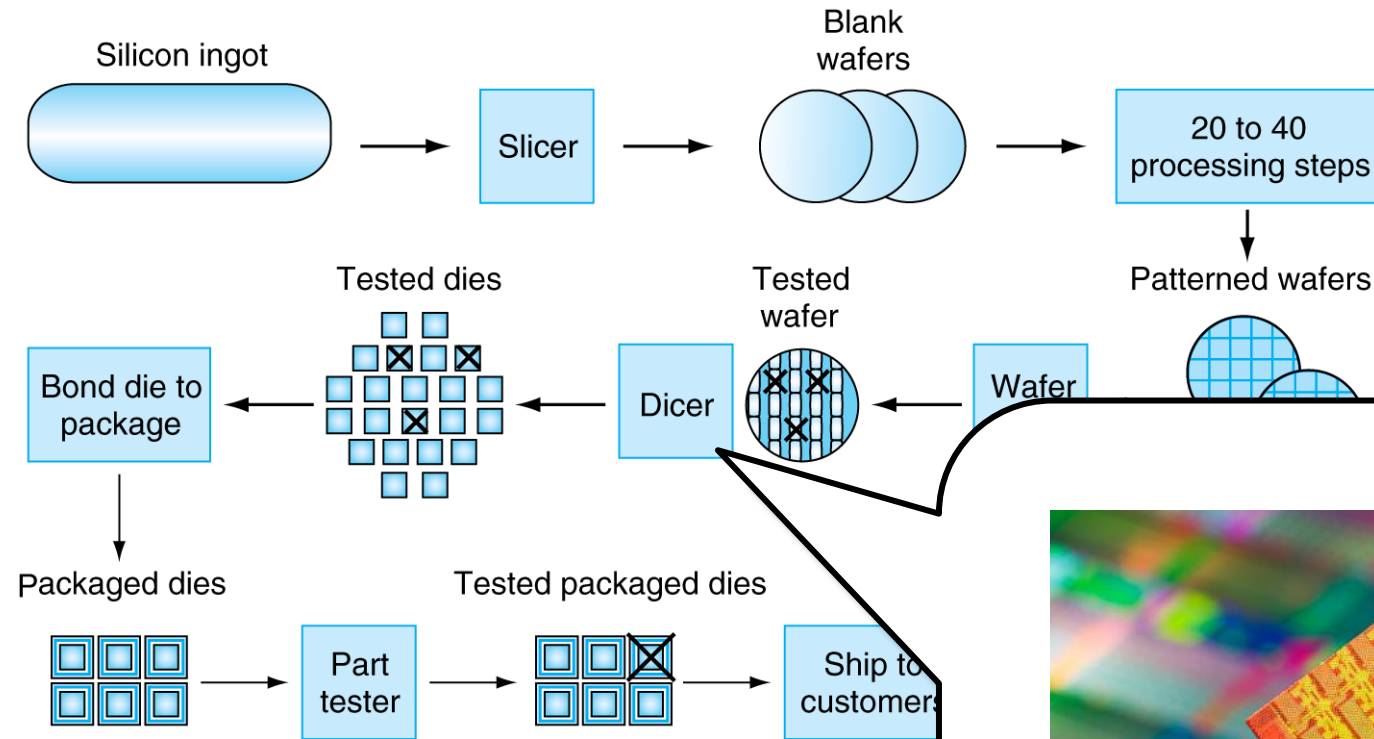
Manufacturing ICs



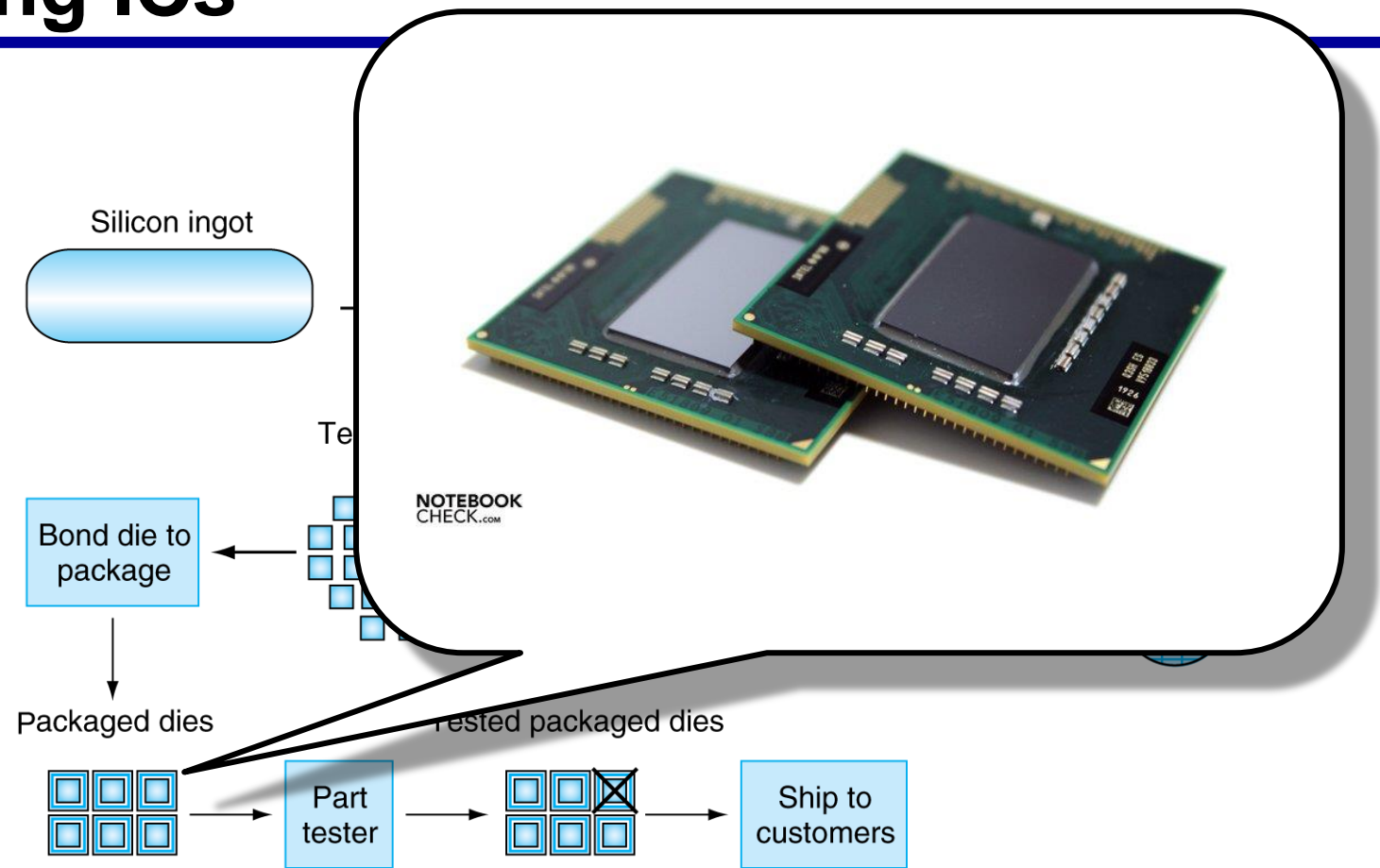
Manufacturing ICs



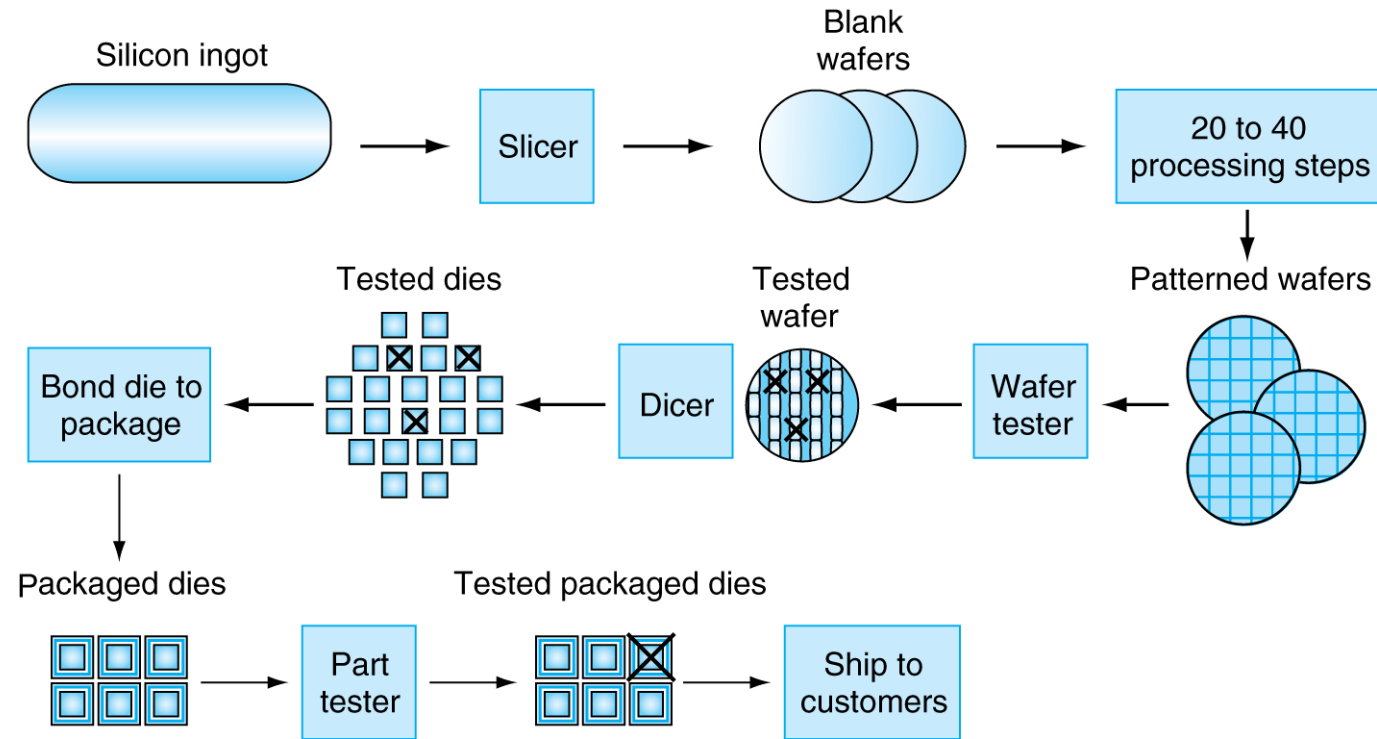
Manufacturing ICs



Manufacturing ICs



Manufacturing ICs



- Yield: Percentage of correctly working dies

Integrated Circuit Cost

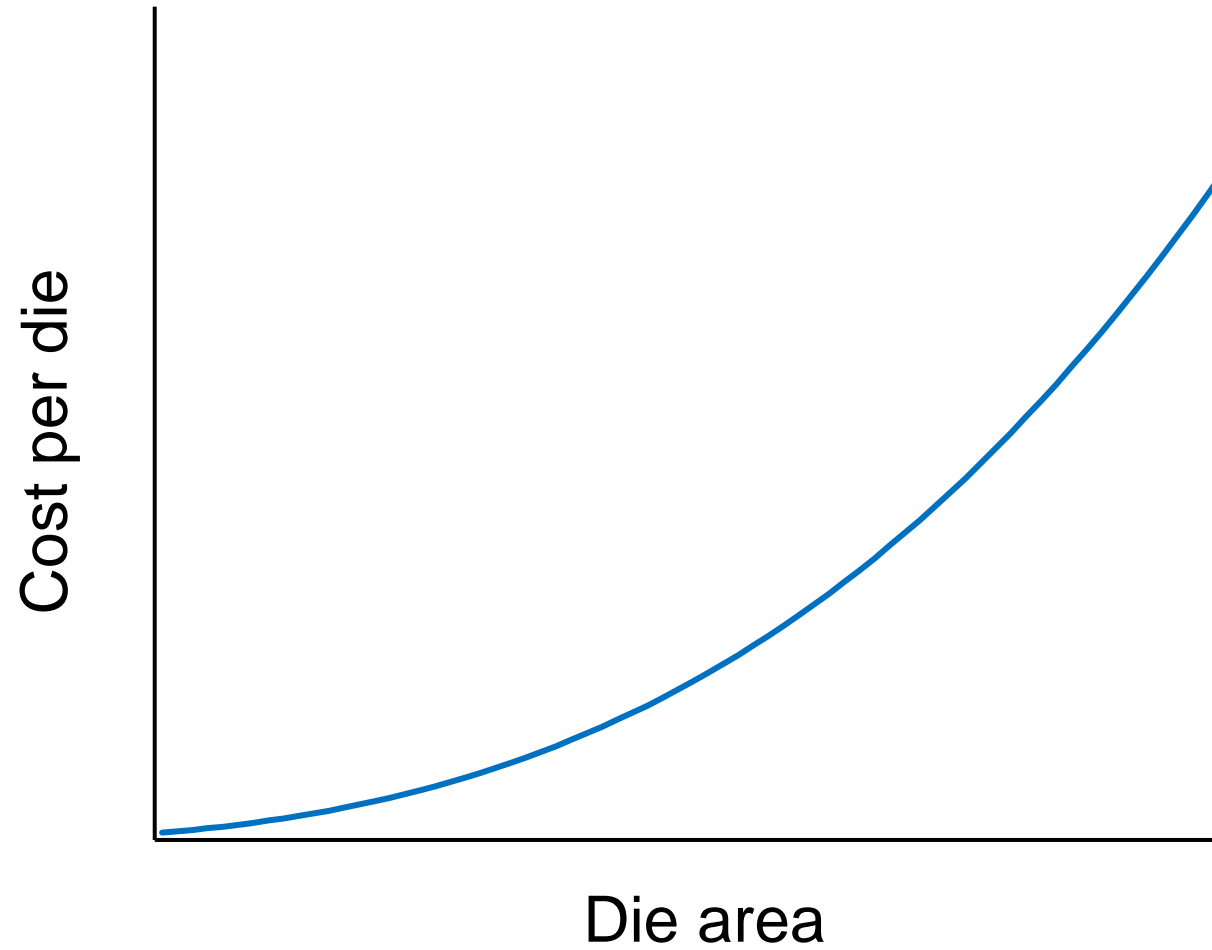
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

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

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

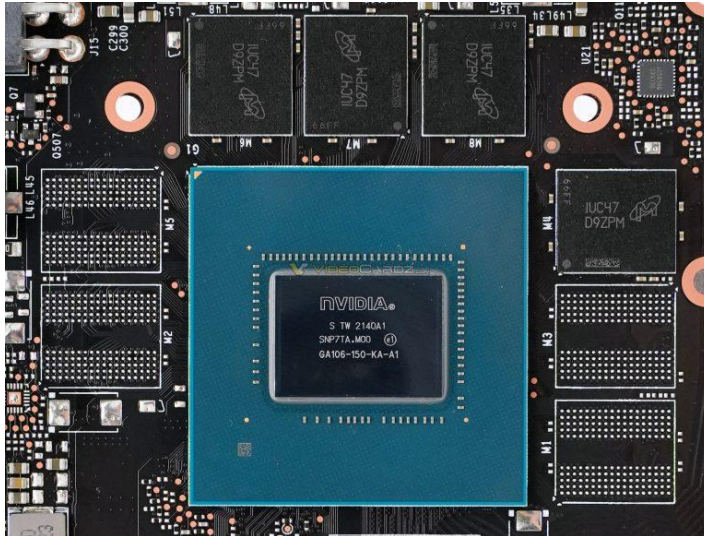
- Non-linear relation to area and defect rate
 - ❖ Wafer cost and area remain fixed
 - ❖ Defect rate determined by the manufacturing process (N)
 - ❖ Die area determined by architecture and circuit design

Cost per Die vs. Die Area



Die Size

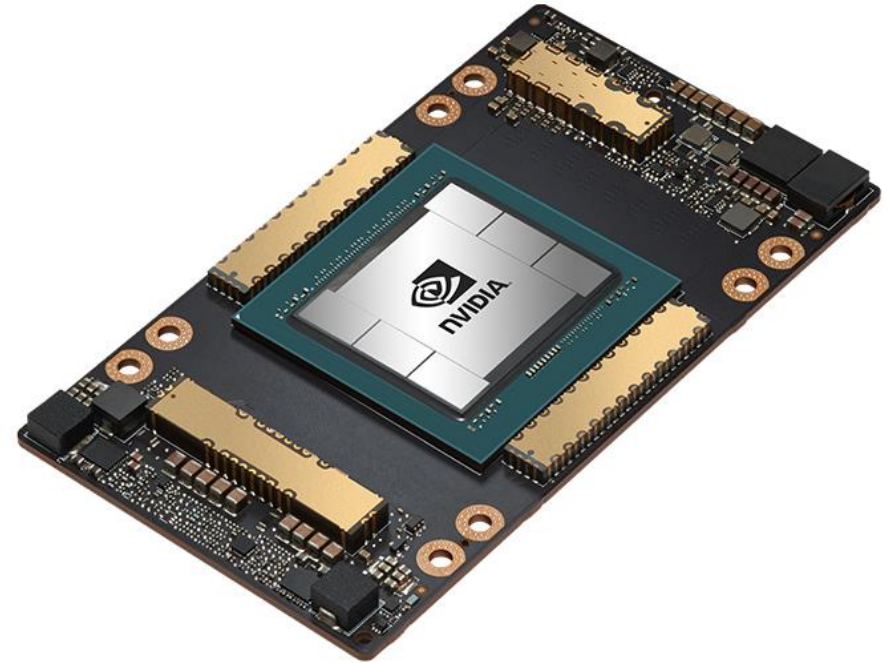
NVIDIA GeForce RTX 3050



276 mm² @ Samsung 8nm

~ \$250

NVIDIA Tesla A100

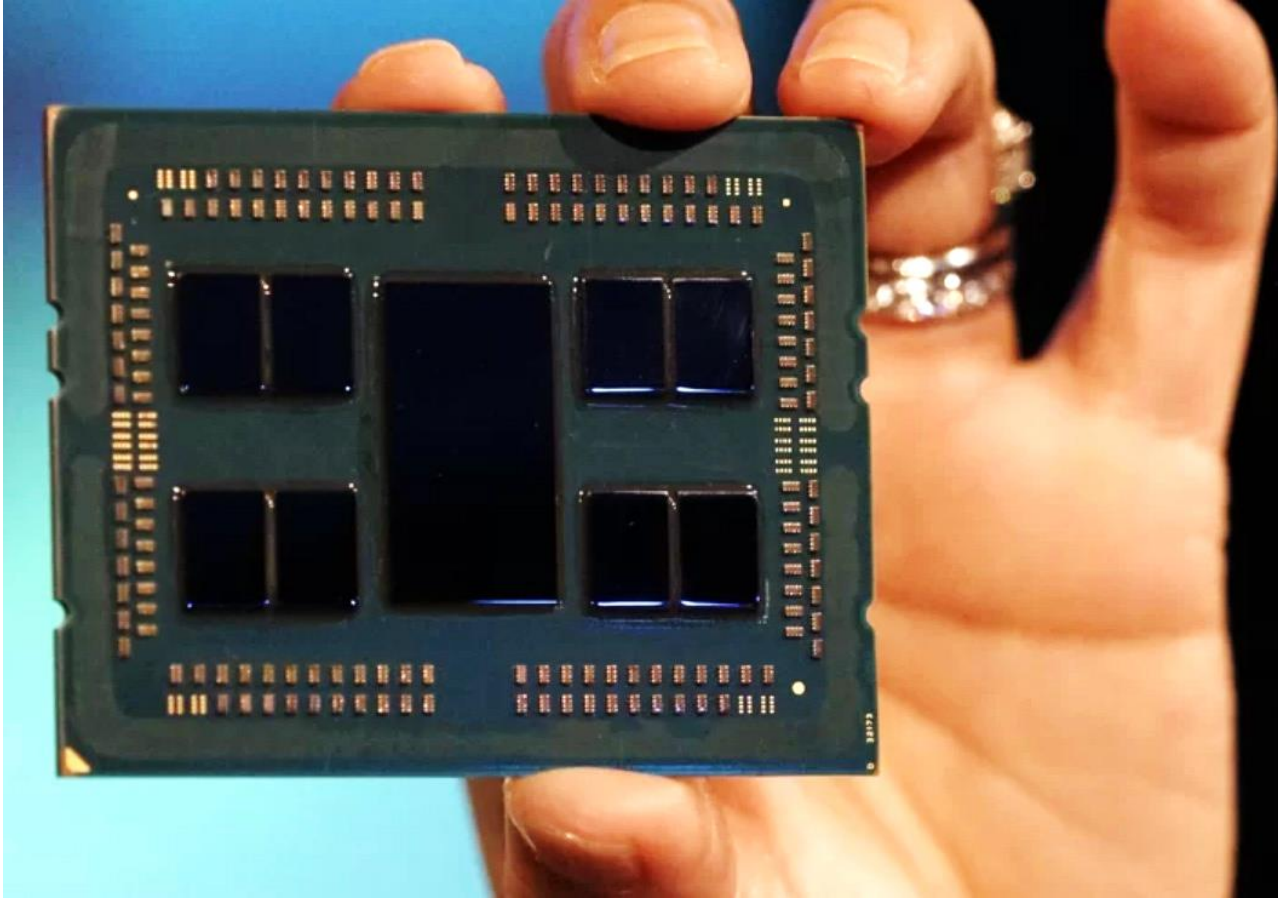


826 mm² @ TSMC 7nm

~ \$10,000

“Chiplet” Trend

AMD Ryzen Threadripper 64-core CPU



- Made with multiple small chips known as “chiplets”
- Connections between the chiplets also use semiconductor technologies

PERFORMANCE

Airline Example

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h.)	Passenger throughput (passengers × m.p.h.)
Boeing 737	240	3000	564	135,360
BAC/Sud Concorde	132	4000	1350	178,200
Boeing 777-200LR	301	9395	554	166,761
Airbus A380-800	853	8477	587	500,711



Response Time and Throughput

- Response time (latency)
 - ❖ **Duration** between the initiation and the completion of a task
 - ❖ Critical for individual users
 - ❖ Embedded computers and PCs are more focused on response time
- Throughput
 - ❖ Overall **number of tasks** completed within a specific timeframe
 - ❖ Critical factors for servers

Relative Performance

- (For now,) Define Performance = 1 / Execution Time
- “X is *n* times faster than Y”

$$\begin{aligned} & \text{Performance}_X / \text{Performance}_Y \\ &= \text{Execution time}_Y / \text{Execution time}_X = n \end{aligned}$$

- Example: time taken to run a program
 - 10s on A, 15s on B
 - $\text{Execution Time}_B / \text{Execution Time}_A = 15\text{s} / 10\text{s} = 1.5$
 - A is 1.5 times faster than B

What Affects Performance of a Computer?

- Application (Algorithm)
 - ❖ Determines the volume (amount) of computation
- Programming language, compiler, processor ISA
 - ❖ Determines the **number of instructions** to execute the computation
- Processor and memory system
 - ❖ Determines **the speed** at which **each instruction** is executed
- Input/Output (I/O) system
 - ❖ Determines **the speed** at which **I/O operations** are performed (not the main focus of this class)

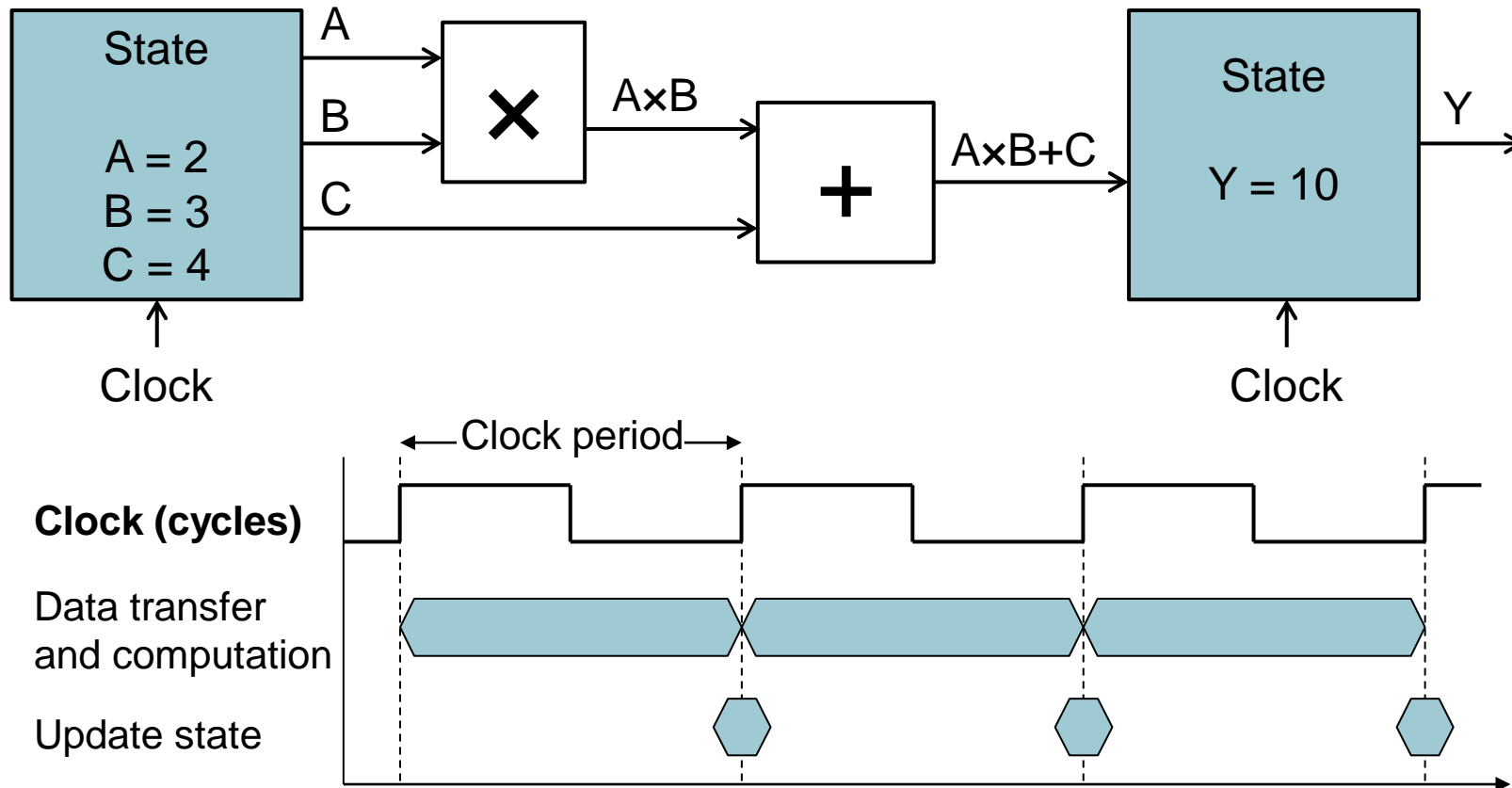
CPU Time

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

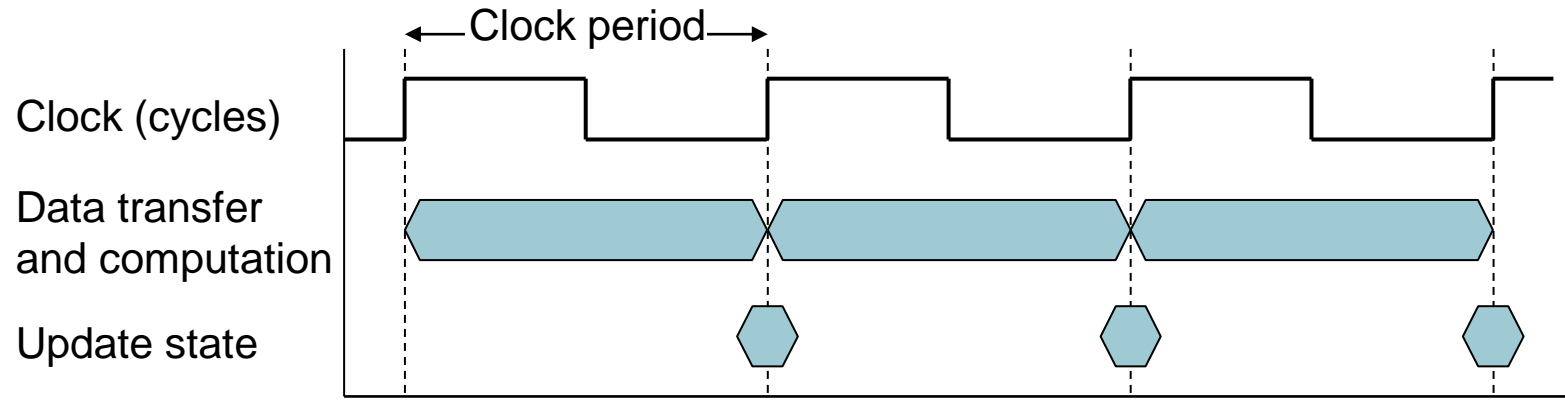
- Performance can be improved by
 - ❖ Reducing the number of clock cycles
 - ❖ Increasing the clock rate
 - ❖ Trade off: clock rate vs. cycle count

CPU Clocking

- Digital circuits are driven by a clock signal



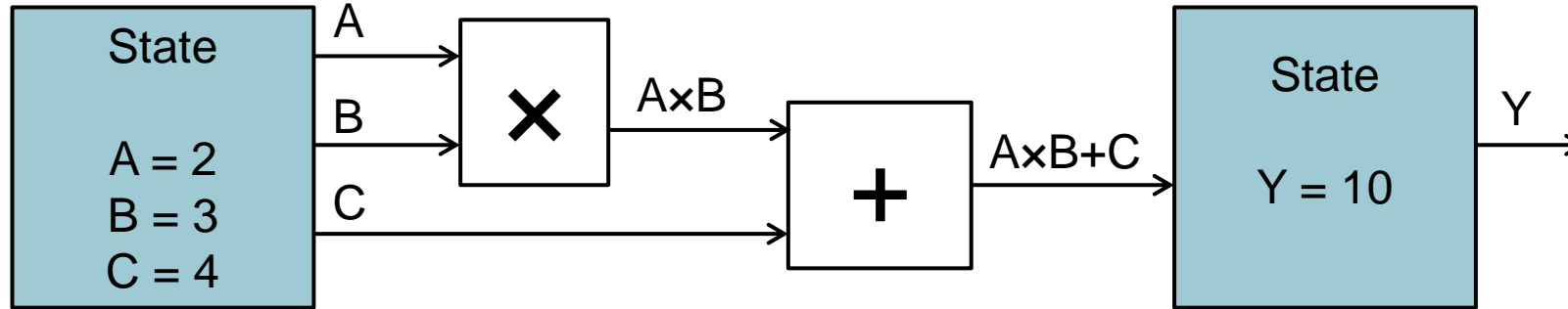
CPU Clocking



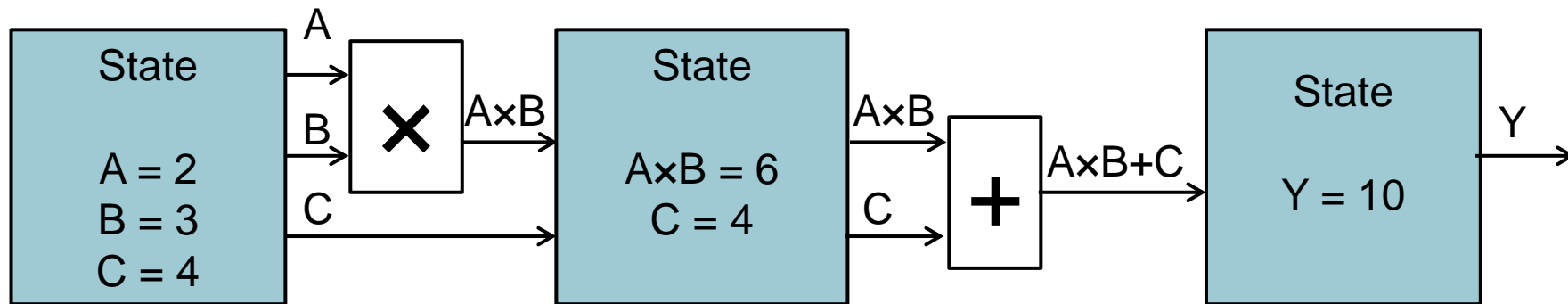
- Clock period: The duration for the clock signal toggles $1 \rightarrow 0 \rightarrow 1$
 - e.g., $250 \text{ ps} = 0.25 \text{ ns} = 250 \times 10^{-12} \text{ s}$
- Clock frequency (rate): The number of clock cycles per second
 - e.g., $4.0 \text{ GHz} = 4000 \text{ MHz} = 4.0 \times 10^9 \text{ Hz}$

Different Clock Cycles

1 cycle for $A \times B + C$



2 cycles for $A \times B + C$



CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
 - ❖ Faster clock frequency, but clock cycle is increased by 1.2x
- Goal: 6s CPU time
 - ❖ How fast Computer B' clock frequency (rate) should be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}$$

$$\begin{aligned}\text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\ &= 10s \times 2\text{GHz} = 20 \times 10^9\end{aligned}$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4\text{GHz}$$

Instruction Count and CPI (Without Pipelining)

Clock Cycles = Instruction Count \times Cycles Per Instruction

CPU Time = Instruction Count \times CPI \times Clock Cycle Time

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count (IC)
 - ❖ Number of instructions executed by a program
- Cycles Per Instruction (CPI)
 - ❖ Determined by the processor design
 - ❖ Can be different for different instruction type

CPI Example

- Computer A: Cycle Time = 250ps, Average CPI = 2.0
- Computer B: Cycle Time = 500ps, Average CPI = 1.2
- Same ISA, Same Program
- Which is faster, and by how much?

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps}\end{aligned}$$

A is faster...

$$\begin{aligned}\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2$$

...by this much

CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^n (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^n \left(\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

Relative frequency

CPI Example

Instruction Type	A	B	C
CPI for type	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
 - Clock Cycles
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$
 $= 10$
 - Avg. CPI = $10/5 = 2.0$

- Sequence 2: IC = 6
 - Clock Cycles
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$
 $= 9$
 - Avg. CPI = $9/6 = 1.5$

Performance Summary

The BIG Picture

$$\text{CPU Time} = \text{Seconds/Program} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

SPEC CPU Benchmark

- Programs used to measure performance
 - ❖ Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
 - ❖ Develops benchmarks for CPU, I/O, Web, ...
- SPEC CPU2006
 - ❖ Elapsed time to execute a selection of programs
 - ❖ SPECratio: Ratio of elapsed time compared to the **reference machine**

Other Benchmark Suites

- EEMBC - Embedded Systems



- TPC – Database transactions



- LINPACK - Supercomputers



- 3DMark – 3D graphics



- Geekbench – Cross-platform



Aggregating Performance Ratios

- Use the geometric mean to aggregate the ratios

$$\text{Geometric mean} = \sqrt[n]{\prod_{i=1}^n \text{SPECratio}_i}$$

	Computer 1 execution time	Computer 2 execution time	Computer 2 is x times faster than Computer 1	Computer 1 is x times faster than Computer 2
Program 1	3	1	3	$1/3$
Program 2	3	2	$3/2$	$2/3$
Arithmetic mean			$9/4$	$1/2$
Geometric mean			$\sqrt{9/2}$	$\sqrt{2/9}$

SPEC CPU Benchmark


SPEC application	Reference	Computer 1	SPECratio
A	10s	10s	10/10 = 1
B	40s	20s	40/20 = 2
C	60s	15s	60/15 = 4

$$\text{Geometric mean} = \sqrt[3]{1 \times 2 \times 4} = \sqrt[3]{8} = 2$$

SPEC application	Reference	Computer 2	SPECratio
A	10s	5s	10/5 = 2
B	40s	5s	40/5 = 8
C	60s	15s	60/15 = 4

$$\text{Geometric mean} = \sqrt[3]{2 \times 8 \times 4} = \sqrt[3]{64} = 4$$

Computer 2 is
2x faster than
Computer 1



SPEC CPU Benchmark


SPEC application	Reference	Computer 1	SPECratio
A	20s	10s	20/10 = 2
B	100s	20s	100/20 = 5
C	5s	15s	5/15 = 1/3

$$\text{Geometric mean} = \sqrt[3]{2 \times 5 \times 1/3} = \sqrt[3]{10/3} \cong 1.49$$

SPEC application	Reference	Computer 2	SPECratio
A	20s	5s	20/5 = 4
B	100s	5s	100/5 = 20
C	5s	15s	5/15 = 1/3

$$\text{Geometric mean} = \sqrt[3]{4 \times 20 \times 1/3} = \sqrt[3]{80/3} \cong 2.98$$

Computer 2 is
still 2x faster
than
Computer 1



CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 ⁹	CPI	Clock cycle time (seconds x 10 ⁻⁹)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

SPEC CPU 2017

Benchmarks

- ☒ Cloud
- ☒ CPU
- ☒ Graphics/Workstations
- ☒ ACCEL/MPI/OMP
- ☒ Java Client/Server
- ☒ Mail Servers
- ☒ Storage
- ☒ Power
- ☒ Virtualization
- ☒ Web Servers

CPU

- **SPEC CPU 2017**
[\[benchmark info\]](#) [\[published results\]](#) [\[support\]](#) [\[order benchmark\]](#)
Designed to provide performance measurements that can be used to compare compute-intensive workloads on different computer systems, SPEC CPU 2017 contains 43 benchmarks organized into four suites: SPECspeed 2017 Integer, SPECspeed 2017 Floating Point, SPECrate 2017 Integer, and SPECrate 2017 Floating Point.
- **SPEC CPU 2006**
[\[Retired\]](#)
- **SPEC CPU 2000**
[\[Retired\]](#)

[\[www.spec.org\]](http://www.spec.org)

Two Metrics in 2017

- <https://www.spec.org/cpu2017/Docs/overview.html#metrics>

There are many ways to measure computer performance. Among the most common are:

- Time - For example, seconds to complete a workload.
- Throughput - Work completed per unit of time, for example, jobs per hour.

SPECspeed is a time-based metric; SPECrate is a throughput metric.

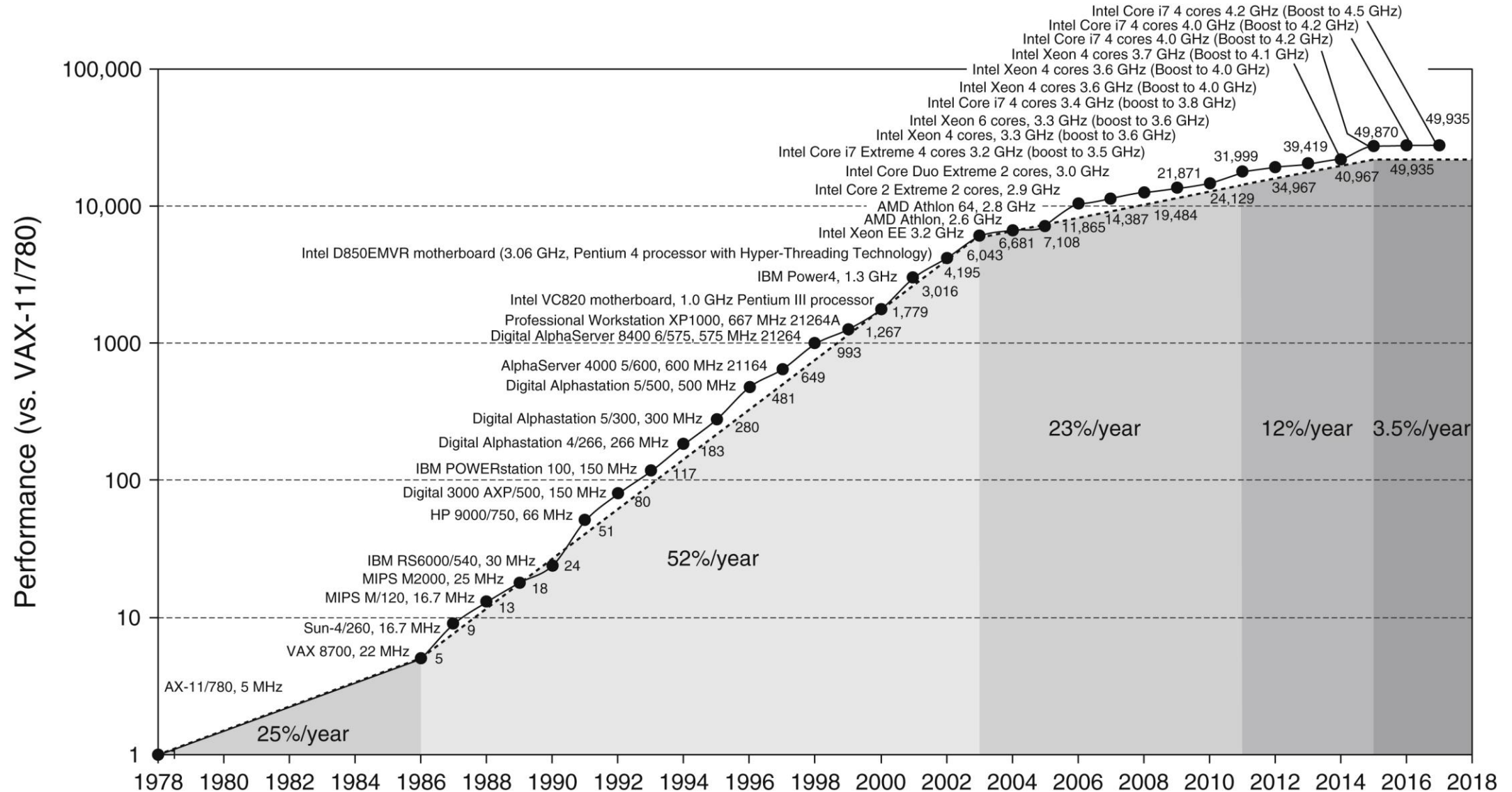
Calculating SPECspeed Metrics	Calculating SPECrate Metrics
1 copy of each benchmark in a suite is run.	The tester chooses how many concurrent copies to run
The tester may choose how many OpenMP threads to use.	OpenMP is disabled.
For each benchmark, a performance ratio is calculated as: $\text{time on a reference machine} / \text{time on the SUT}$	For each benchmark, a performance ratio is calculated as: $\text{number of copies} * (\text{time on a reference machine} / \text{time on the SUT})$
Higher scores mean that less time is needed.	Higher scores mean that more work is done per unit of time.
Example: <ul style="list-style-type: none">• The reference machine ran 600.perlbench_s in 1775 seconds.• A particular SUT took about 1/5 the time, scoring about 5.• More precisely: $1775/354.329738 = 5.009458$	Example: <ul style="list-style-type: none">• The reference machine ran 1 copy of 500.perlbench_r in 1592 seconds.• A particular SUT ran 8 copies in about 1/3 the time, scoring about 24.• More precisely: $8 * (1592/541.52471) = 23.518776$
For both SPECspeed and SPECrate, in order to provide some assurance that results are repeatable, the entire process is repeated. The tester may choose: <ol style="list-style-type: none">To run the suite of benchmarks three times, in which case the tools select the medians.Or to run twice, in which case the tools select the lower ratios (i.e. slower). <i>New with CPU2017</i>	
For both SPECspeed and SPECrate, the selected ratios are averaged using the Geometric Mean, which is reported as the overall metric.	

Benchmark Applications

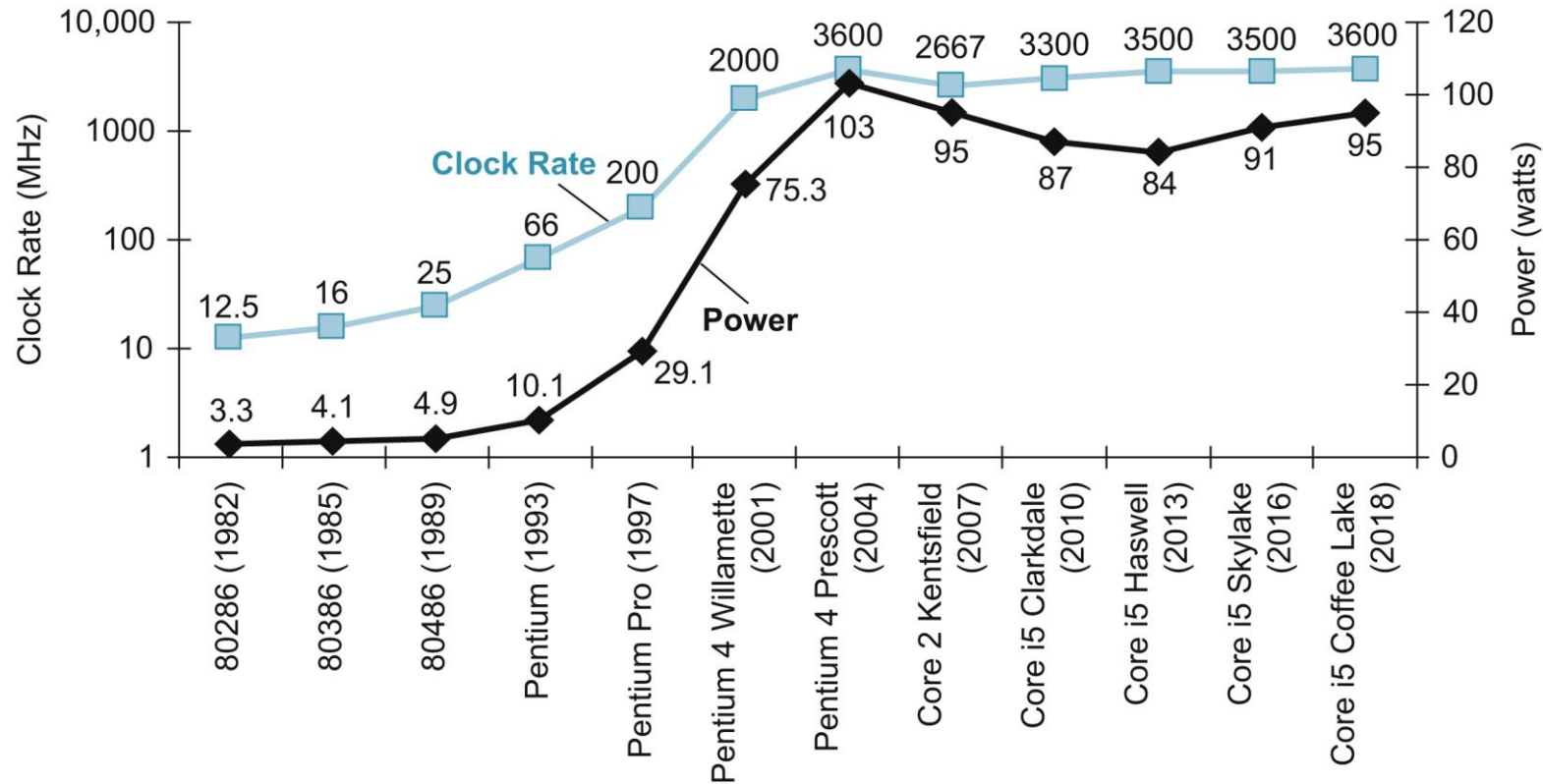
SPECrate 2017 Integer	SPECspeed 2017 Integer	Language [1]	KLOC [2]	Application Area
500.perlbench_r	600.perlbench_s	C	362	Perl interpreter
502.gcc_r	602.gcc_s	C	1,304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	21	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression

SPECrate 2017 Floating Point	SPECspeed 2017 Floating Point	Language [1]	KLOC [2]	Application Area
503.bwaves_r	603.bwaves_s	Fortran	1	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	Physics: relativity

Uniprocessor Performance



Power Consumption Trends



- In CMOS IC technology

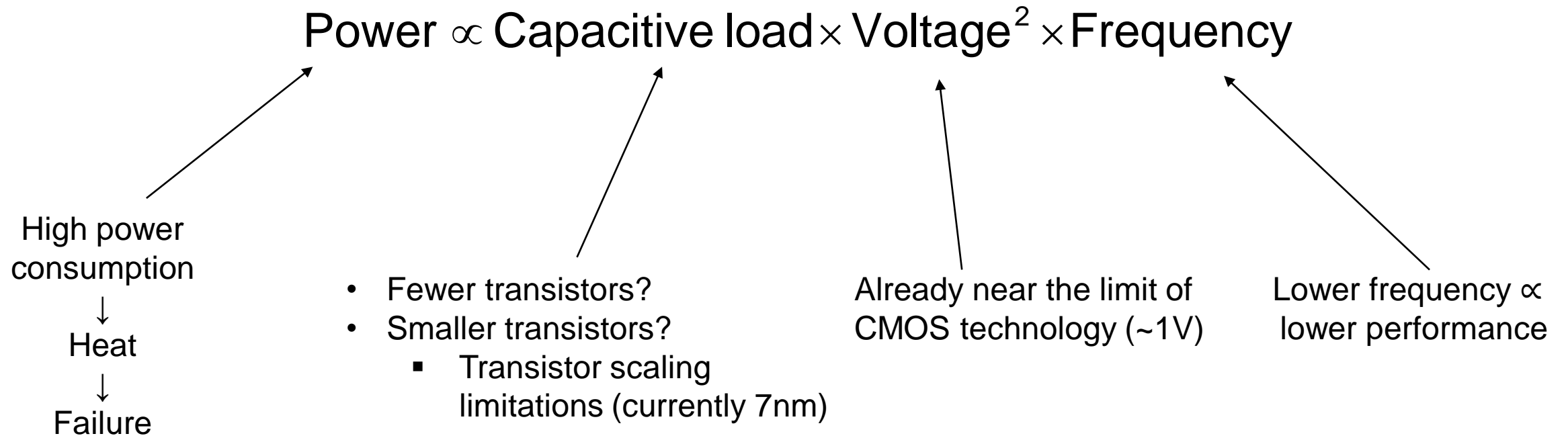
$$\text{Power} \propto \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

↑
x30

↑
5V → 1V

↑
x300

The Power Wall

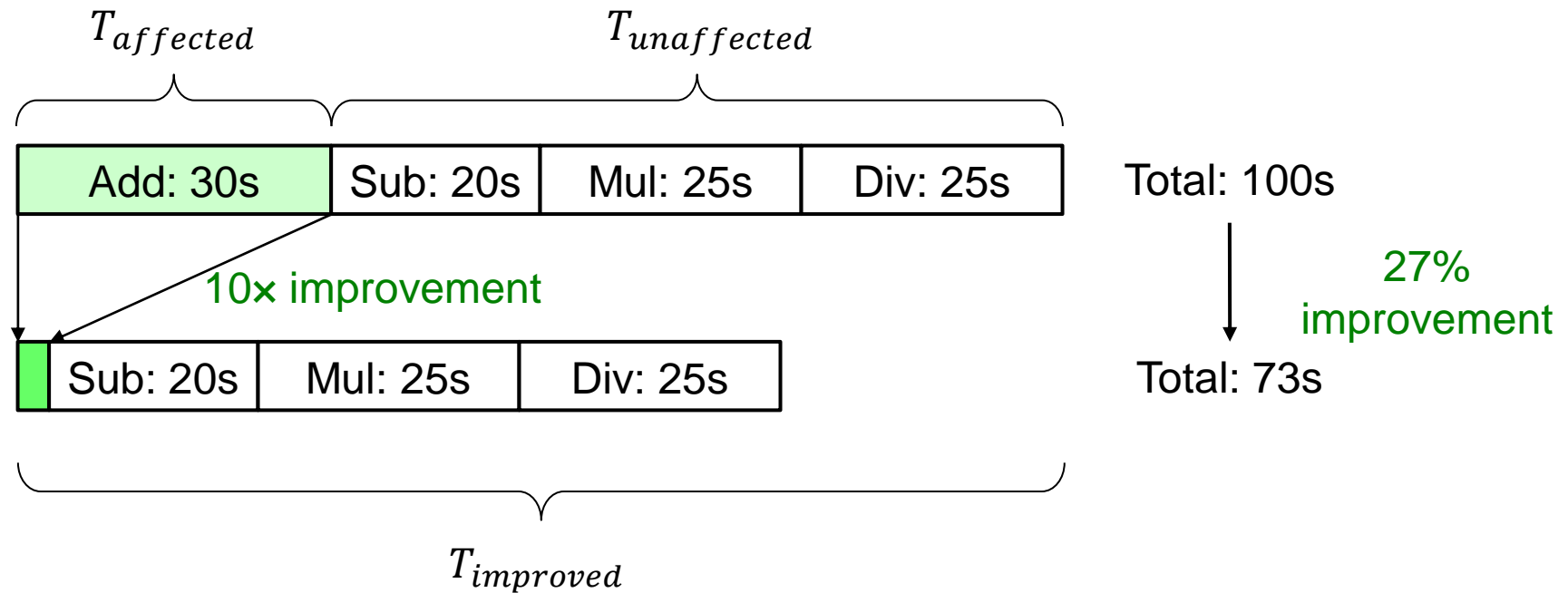


- How else can we improve performance?

Multiprocessors

- Multicore microprocessors
 - ❖ More than one processor per chip
 - ❖ Requires parallel programming
 - Divide the computation to multiple independent tasks
 - Load balancing
 - Optimizing communication and synchronization
- GPUs
 - ❖ Optimized for parallel computations
- Domain-Specific Accelerators
 - ❖ Neural Processing Units

Amdahl's Law



$$T_{improved} = \frac{T_{affected}}{\text{improvement factor}} + T_{unaffected}$$

- Corollary: make the **common case fast**

Amdahl's Law Example

- Multiply operations accounts for 80% of the application
 - ❖ Simply, total application: 100s, multiply: 80s
- You can accelerate multiply operations as much as you want
- How fast the multiply operation should be to make the application to run 5x faster?

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

$$20s = \frac{80s}{n} + 20s$$

Impossible unless multiply operation takes zero second ($n=\infty$)