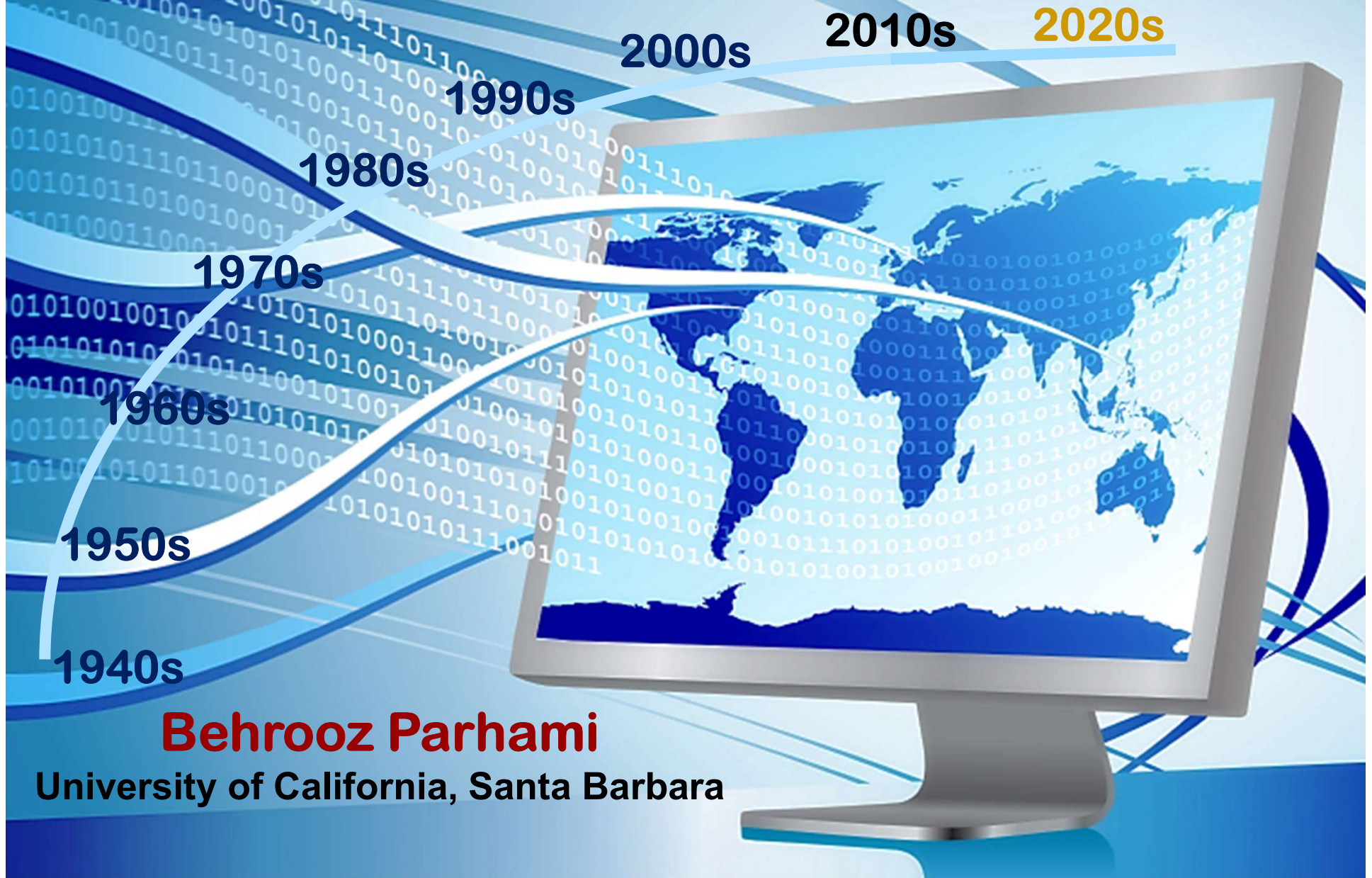


Eight Key Ideas in Computer Architecture, from Eight Decades of Innovation



Behrooz Parhami

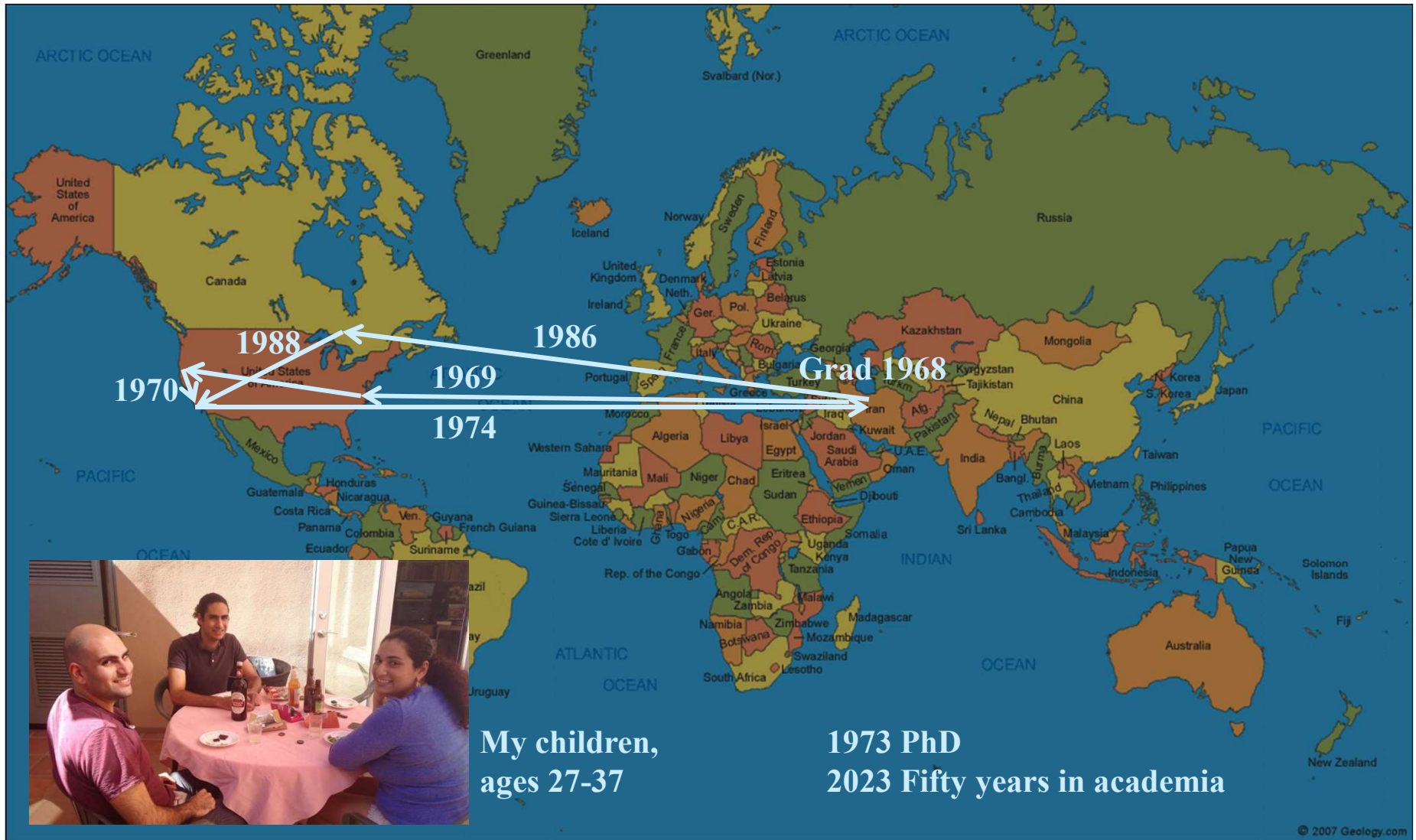
University of California, Santa Barbara

About This Presentation

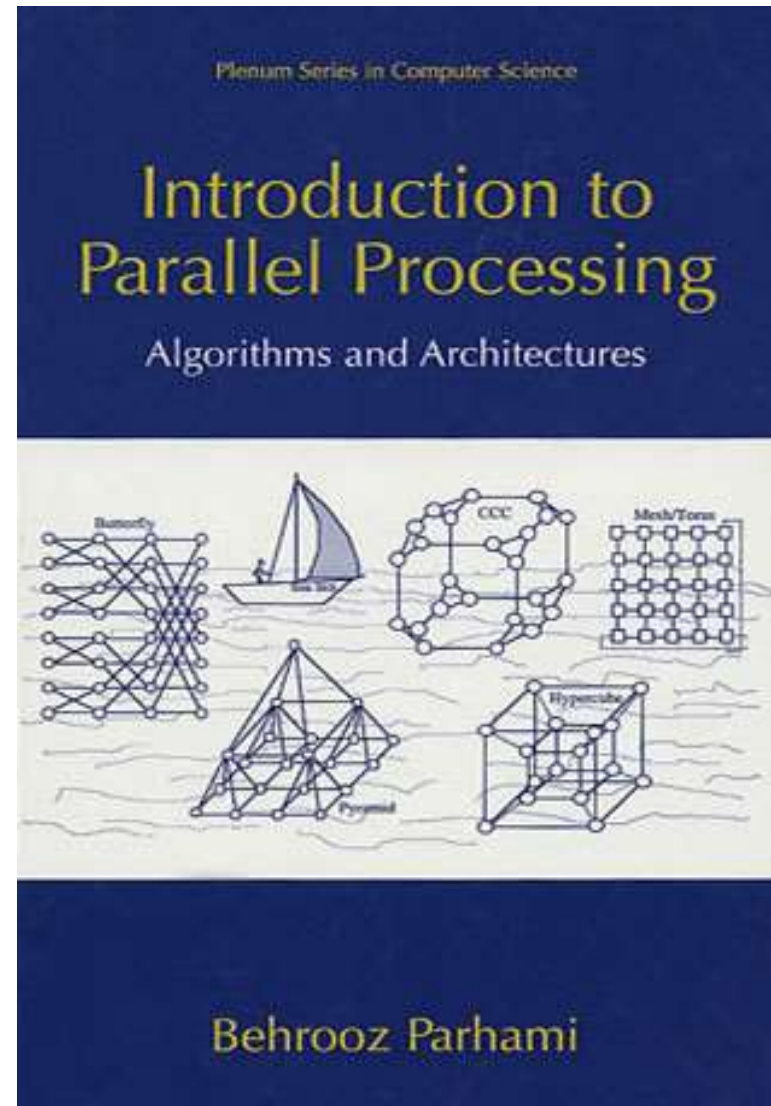
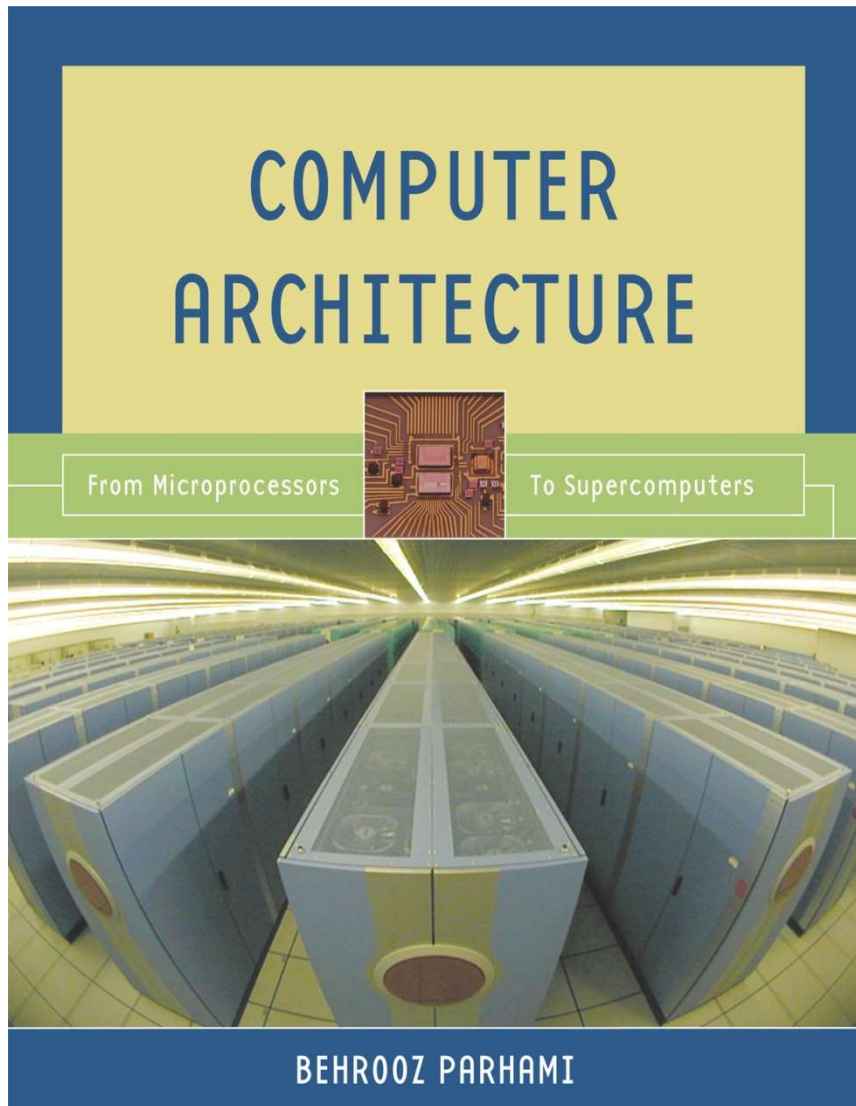
This slide show was first developed as a keynote talk for remote delivery at CSICC-2016, Computer Society of Iran Computer Conference, held in Tehran on March 8-10. The talk was presented at a special session on March 9, 11:30 AM to 12:30 PM Tehran time (12:00-1:00 AM PST). An updated version of the talk was prepared and used within B. Parhami's suite of lectures for IEEE Computer Society's Distinguished Visitors Program, 2021-2023. All rights reserved for the author. ©2021 Behrooz Parhami

Edition	Released	Revised	Revised	Revised
First	Mar. 2016	July 2017		
Second	Mar. 2021			

My Personal Academic Journey



Some of the material in this talk come from, or will appear in updated versions of, my two computer architecture textbooks



Mar. 2021



Eight Key Ideas in Computer Architecture

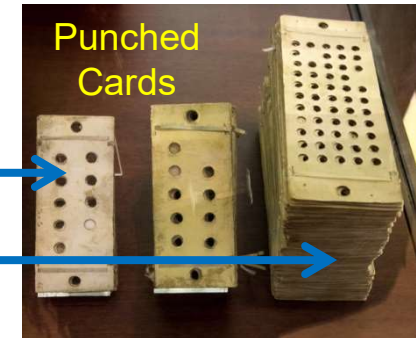
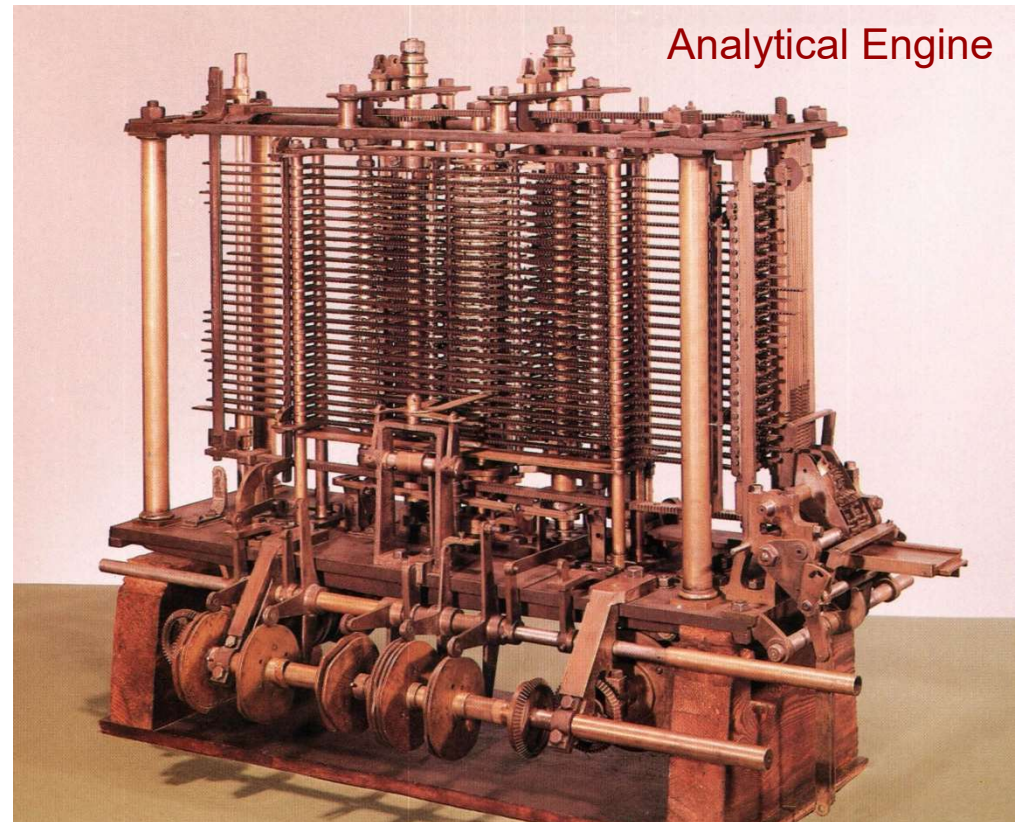
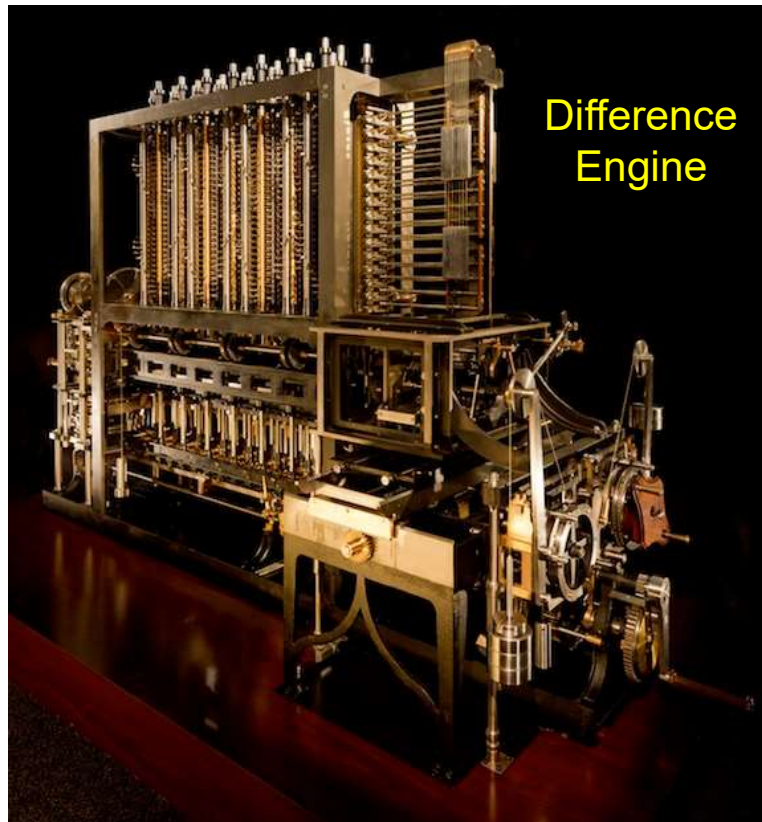


Slide 4

Eight Key Ideas in Computer Architecture, from Eight Decades of Innovation

Computer architecture became an established discipline when the stored-program concept was incorporated into bare-bones computers of the 1940s. Since then, the field has seen multiple minor and major innovations in each decade. I will present my pick of the most-important innovation in each of the eight decades, from the 1940s to the 2010s, and show how these ideas, when connected to each other and allowed to interact and cross-fertilize, produced the phenomenal growth of computer performance, now approaching exa-op/s (billion billion operations per second) level, as well as to ultra-low-energy and single-chip systems. I will also offer predictions for what to expect in the 2020s and beyond.

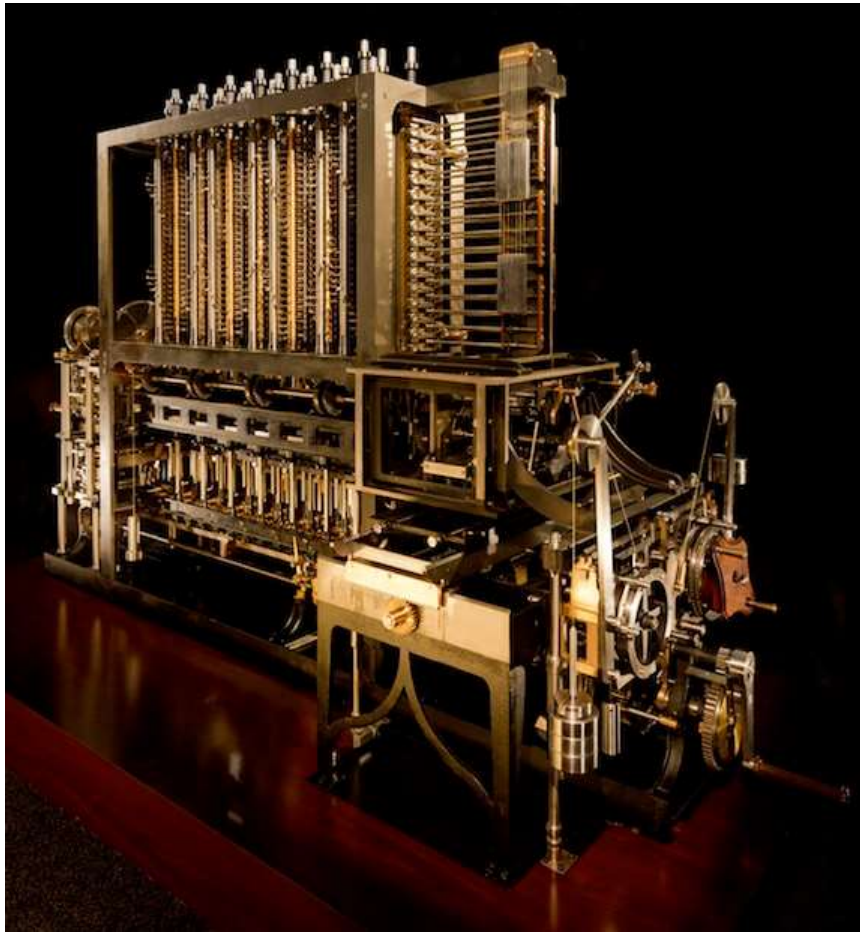
Background: 1820s-1930s



Program (Instructions)

Data (Variable values)

Difference Engine: Fixed Program

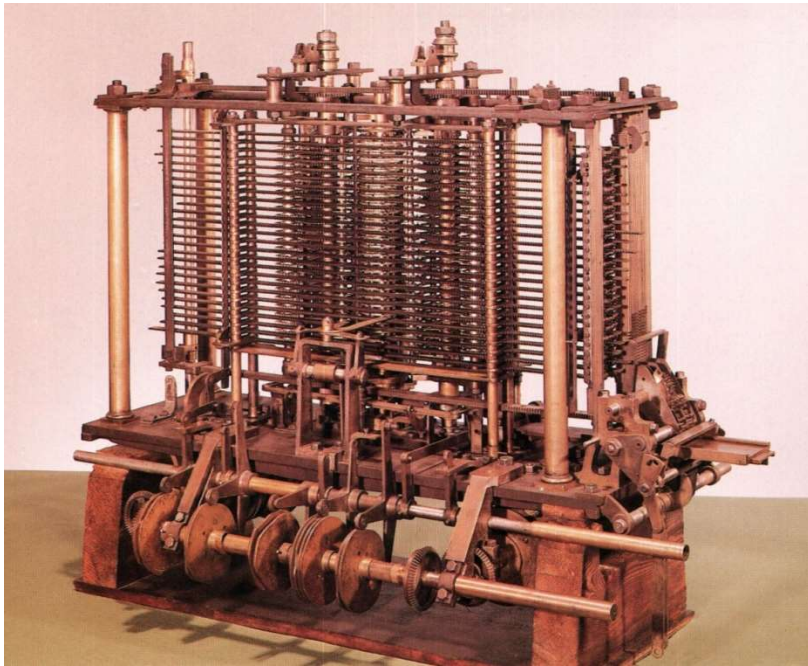


Babbage's Difference Engine 2

$D^{(2)}$	$D^{(1)}$	$f(x)$ x^2+x+41	x
		41	0
2	2	43	1
2	4	47	2
2	6	53	3
2	8	61	4
2	10	71	5

2nd-degree polynomial evaluation
Babbage used 7th-degree $f(x)$

Analytical Engine: Programmable

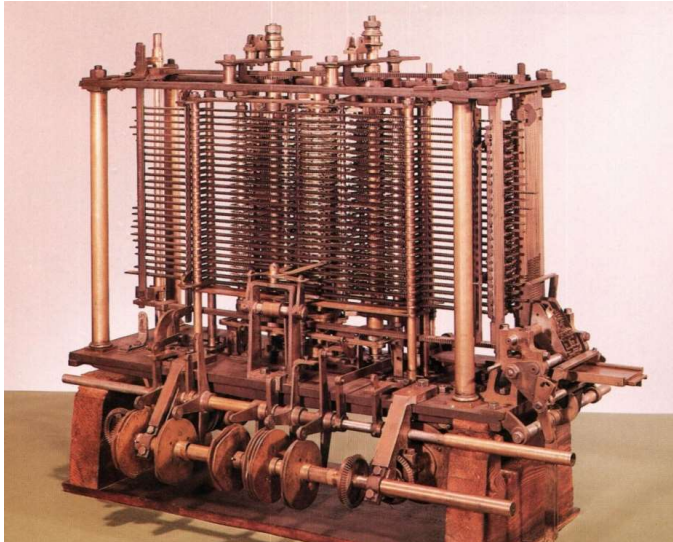


Ada Lovelace,
world's first programmer

Sample program >

Number of Operation	Nature of Operation	Variables acted upon	Variables receiving results	Indication of change in the value on any Variable	Statement of Results	Data				
						1V_1 0 0 0 1	1V_2 0 0 0 2	1V_3 0 0 0 4	0V_4 0 0 0 0	0V_5 0 0 0 0
1	\times	${}^1V_2 \times {}^1V_3$	${}^1V_4, {}^1V_5, {}^1V_6$	$\left\{ \begin{array}{l} {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \\ {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \end{array} \right\}$	$= 2n \dots \dots \dots$...	2	n	$2n$	$2n$
2	$-$	${}^1V_4 - {}^1V_1$	${}^2V_4 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \end{array} \right\}$	$= 2n-1 \dots \dots \dots$	1	$2n-1$...
3	$+$	${}^1V_5 + {}^1V_1$	${}^2V_5 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \end{array} \right\}$	$= 2n+1 \dots \dots \dots$	1	$2n+1$
4	$+$	${}^1V_5 + {}^1V_4$	${}^1V_{12} \dots \dots$	$\left\{ \begin{array}{l} {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \\ {}^1V_{12} = {}^1V_{12} \end{array} \right\}$	$= \frac{2n-1}{2n+1} \dots \dots \dots$	0	0
5	$+$	${}^1V_{11} + {}^1V_2$	${}^2V_{11} \dots \dots$	$\left\{ \begin{array}{l} {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \end{array} \right\}$	$= \frac{1}{2} \frac{2n-1}{2n+1} \dots \dots \dots$...	2
6	$-$	${}^0V_{13} - {}^1V_{11}$	${}^1V_{18} \dots \dots$	$\left\{ \begin{array}{l} {}^0V_{13} = {}^0V_{13} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \end{array} \right\}$	$= -\frac{1}{2} \frac{2n-1}{2n+1} = A_0 \dots \dots \dots$
7	$-$	${}^1V_2 - {}^1V_1$	${}^1V_{10} \dots \dots$	$\left\{ \begin{array}{l} {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \\ {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \\ {}^1V_7 = {}^1V_7 \\ {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \end{array} \right\}$	$= n-1 (=3) \dots \dots \dots$	1	...	n
8	$+$	${}^1V_2 + {}^0V_7$	${}^1V_7 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_2 = {}^1V_2 \\ {}^0V_7 = {}^0V_7 \\ {}^1V_3 = {}^1V_3 \\ {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \end{array} \right\}$	$= 2+0 = 2 \dots \dots \dots$...	2
9	$+$	${}^1V_8 + {}^1V_7$	${}^2V_{11} \dots \dots$	$\left\{ \begin{array}{l} {}^0V_7 = {}^0V_7 \\ {}^1V_7 = {}^1V_7 \\ {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \end{array} \right\}$	$= \frac{2n}{2} = A_1 \dots \dots \dots$
10	\times	${}^1V_{11} \times {}^1V_{11}$	${}^1V_{18} \dots \dots$	$\left\{ \begin{array}{l} {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1 \dots \dots \dots$
11	$+$	${}^1V_{12} + {}^1V_{15}$	${}^1V_{18} \dots \dots$	$\left\{ \begin{array}{l} {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= -\frac{1}{2} \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2} \dots \dots \dots$
12	$-$	${}^1V_{16} - {}^1V_1$	${}^2V_{10} \dots \dots$	$\left\{ \begin{array}{l} {}^1V_{16} = {}^1V_{16} \\ {}^1V_1 = {}^1V_1 \\ {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \\ {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \\ {}^1V_7 = {}^1V_7 \\ {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= n-2 (=2) \dots \dots \dots$	1
13	$-$	${}^1V_4 - {}^1V_1$	${}^2V_6 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \\ {}^1V_7 = {}^1V_7 \\ {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= 2n-1 \dots \dots \dots$	1
14		$+$	${}^1V_1 + {}^1V_7$	${}^2V_7 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_1 = {}^1V_1 \\ {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \\ {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \\ {}^1V_7 = {}^1V_7 \\ {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= 2+1 = 3 \dots \dots \dots$	1
15		$+$	${}^1V_6 + {}^1V_7$	${}^2V_4 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_6 = {}^1V_6 \\ {}^1V_7 = {}^1V_7 \\ {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= \frac{2n-1}{3} \dots \dots \dots$
16	\times	${}^1V_6 \times {}^1V_{11}$	${}^2V_{11} \dots \dots$	$\left\{ \begin{array}{l} {}^1V_6 = {}^1V_6 \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= \frac{2n}{2} \frac{2n-1}{3} \dots \dots \dots$
17	$-$	${}^1V_8 - {}^1V_1$	${}^2V_8 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= 2n-2 \dots \dots \dots$	1
18		$+$	${}^1V_1 + {}^1V_7$	${}^2V_7 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_1 = {}^1V_1 \\ {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \\ {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \\ {}^1V_7 = {}^1V_7 \\ {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= 3+1 = 4 \dots \dots \dots$	1
19		$+$	${}^1V_6 + {}^1V_7$	${}^1V_8 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_6 = {}^1V_6 \\ {}^1V_7 = {}^1V_7 \\ {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= \frac{2n-2}{4} \dots \dots \dots$
20	\times	${}^1V_8 \times {}^1V_{11}$	${}^2V_{11} \dots \dots$	$\left\{ \begin{array}{l} {}^1V_8 = {}^1V_8 \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= \frac{2n}{2} \frac{2n-1}{3} \frac{2n-2}{4} = A_2 \dots \dots \dots$
21	\times	${}^1V_{11} \times {}^1V_{11}$	${}^0V_{13} \dots \dots$	$\left\{ \begin{array}{l} {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= B_2 \cdot \frac{2n}{2} \frac{2n-1}{3} \frac{2n-2}{4} = B_2 A_2 \dots \dots \dots$
22	$+$	${}^2V_{12} + {}^2V_{18}$	${}^3V_{13} \dots \dots$	$\left\{ \begin{array}{l} {}^2V_{12} = {}^2V_{12} \\ {}^2V_{13} = {}^2V_{13} \\ {}^2V_{14} = {}^2V_{14} \\ {}^2V_{15} = {}^2V_{15} \\ {}^2V_{16} = {}^2V_{16} \\ {}^2V_{17} = {}^2V_{17} \\ {}^2V_{18} = {}^2V_{18} \end{array} \right\}$	$= A_0 + B_1 A_1 + B_2 A_2 \dots \dots \dots$
23	$-$	${}^1V_{10} - {}^1V_1$	${}^3V_{10} \dots \dots$	$\left\{ \begin{array}{l} {}^2V_{10} = {}^2V_{10} \\ {}^2V_{11} = {}^2V_{11} \\ {}^2V_{12} = {}^2V_{12} \\ {}^2V_{13} = {}^2V_{13} \\ {}^2V_{14} = {}^2V_{14} \\ {}^2V_{15} = {}^2V_{15} \\ {}^2V_{16} = {}^2V_{16} \\ {}^2V_{17} = {}^2V_{17} \\ {}^2V_{18} = {}^2V_{18} \\ {}^1V_1 = {}^1V_1 \\ {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \\ {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \\ {}^1V_7 = {}^1V_7 \\ {}^1V_8 = {}^1V_8 \\ {}^1V_9 = {}^1V_9 \\ {}^1V_{10} = {}^1V_{10} \\ {}^1V_{11} = {}^1V_{11} \\ {}^1V_{12} = {}^1V_{12} \\ {}^1V_{13} = {}^1V_{13} \\ {}^1V_{14} = {}^1V_{14} \\ {}^1V_{15} = {}^1V_{15} \\ {}^1V_{16} = {}^1V_{16} \\ {}^1V_{17} = {}^1V_{17} \\ {}^1V_{18} = {}^1V_{18} \end{array} \right\}$	$= n-3 (=1) \dots \dots \dots$	1
						Here follows a repetition				
24	$+$	${}^4V_{12} + {}^4V_{14}$	${}^1V_{24} \dots \dots$	$\left\{ \begin{array}{l} {}^4V_{12} = {}^4V_{12} \\ {}^4V_{13} = {}^4V_{13} \\ {}^4V_{14} = {}^4V_{14} \\ {}^4V_{15} = {}^4V_{15} \\ {}^4V_{16} = {}^4V_{16} \\ {}^4V_{17} = {}^4V_{17} \\ {}^4V_{18} = {}^4V_{18} \end{array} \right\}$	$= B_7 \dots \dots \dots$
25	$+$	${}^1V_2 + {}^1V_3$	${}^1V_3 \dots \dots$	$\left\{ \begin{array}{l} {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \\ {}^1V_4 = {}^1V_4 \\ {}^1V_5 = {}^1V_5 \\ {}^1V_6 = {}^1V_6 \\ {}^1V_7 = {}^1V_7 \end{array} \right\}$	$= n+1 = 4+1 = 5 \dots \dots \dots$ by a Variable-card. by a Variable-card.	1	...	$n+1$

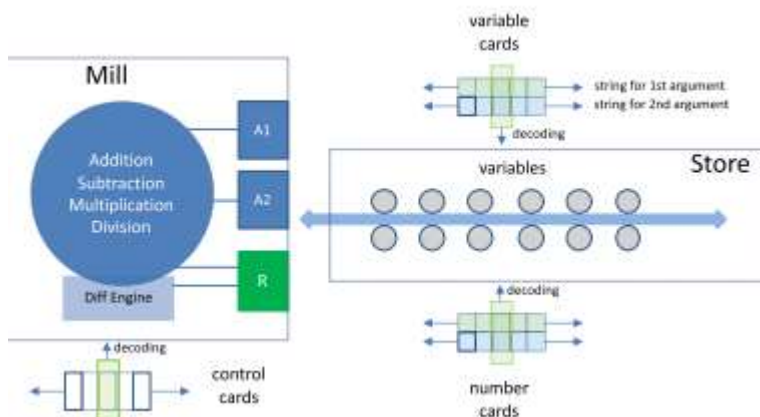
The Programs of Charles Babbage



THE LIST OF PROGRAMS

The 26 Babbage's tables in London's Museum can be arranged in the following groups. There are (with overlaps):

- Eleven programs for linear algebra (mostly solutions of simultaneous equations).
- Four for the evaluation of polynomial multiplication and division.
- Two for computing recursions (using loop-unrolling, i.e., sequential code obtained by writing one loop execution right after the other).
- Four for the simulation of a Difference Engine using the so-called carriages.
- Three for the evaluation of astronomical formulas.
- A program showing how to use "combinatorial cards."
- An example program for the computation of a simple formula.

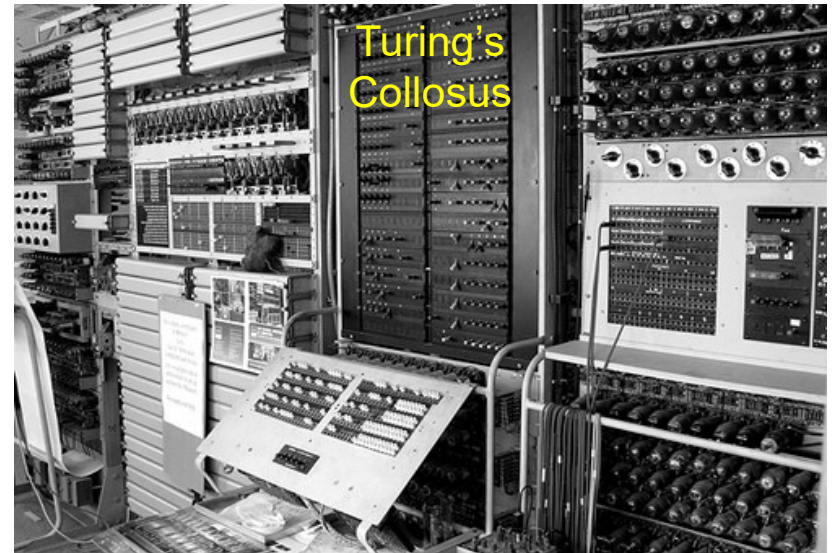
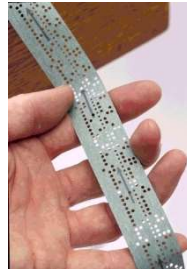


IEEE Annals of the History of Computing,
January-March 2021 (article by Raul Rojas)

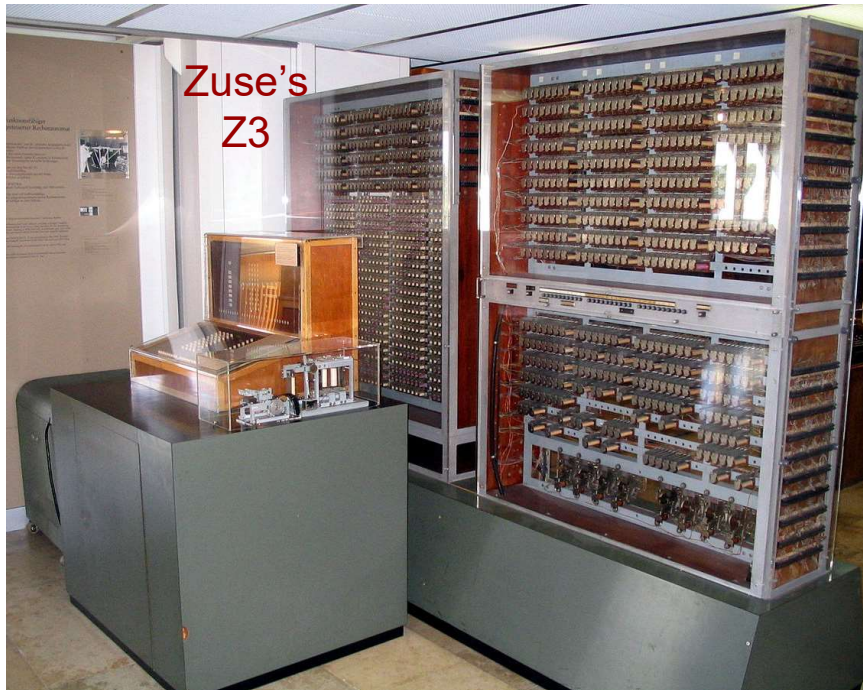
Electromechanical and Plug-Programmable Computing Machines



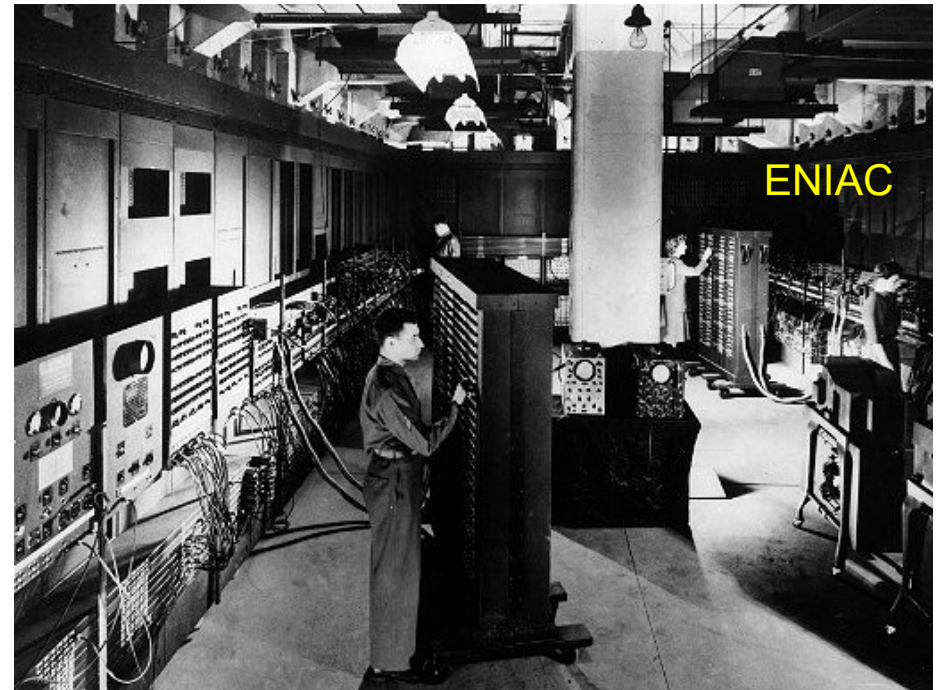
Punched-card device



Turing's Colossus




Zuse's Z3



ENIAC

The Eight Key Ideas

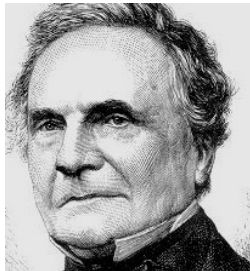
- 
- 1940s **Stored program**
 - 1950s **Microprogramming**
 - 1960s **Parallel processing**
 - 1970s **Cache memory**
 - 1980s **Pipelining**
 - 1990s **FPGAs**
 - 2000s **GPUs**
 - 2010s **Specialization**

**Come back in 2040,
for my top-10 list!**

1940s: Stored Program

Exactly who came up with the stored-program concept is unclear

Legally, John Vincent Atanasoff is designated as inventor, but many others deserve to share the credit



Babbage



Turing



Atanasoff



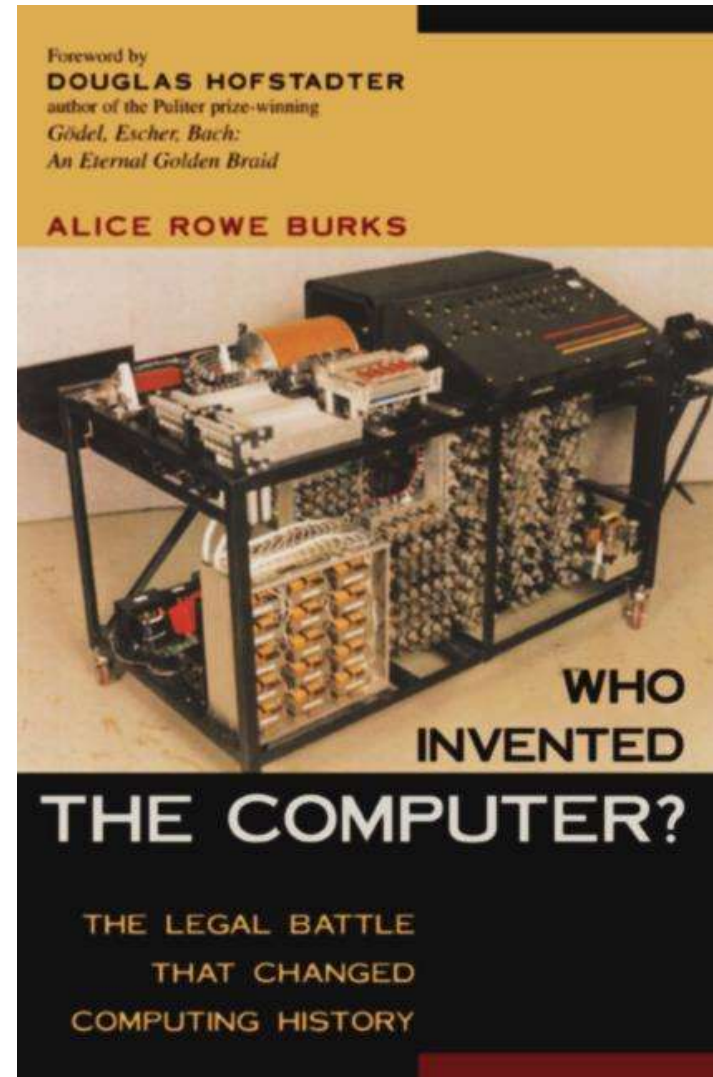
Eckert



Mauchly



von Neumann



First Stored-Program Computer

Manchester Small-Scale Experimental Machine

Ran a stored program on June 21, 1948

(Its successor, Manchester Mark 1, operational in April 1949)

EDSAC (Cambridge University; Wilkes et al.)

Fully operational on May 6, 1949

EDVAC (IAS, Princeton University; von Neumann et al.)

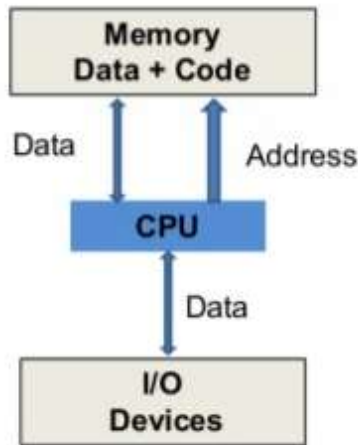
Conceived in 1945 but not delivered until August 1949

BINAC (Binary Automatic Computer, Eckert & Mauchly)

Delivered on August 22, 1949, but did not function correctly

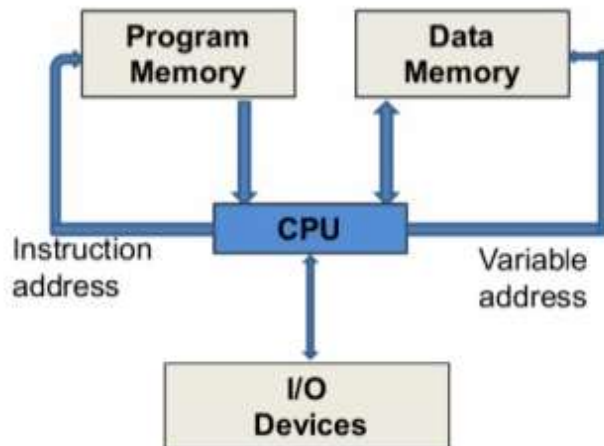
Source: Wikipedia

von Neumann vs. Harvard Architecture



von Neumann architecture
(unified memory for code & data)

Programs can be modified like data
More efficient use of memory space

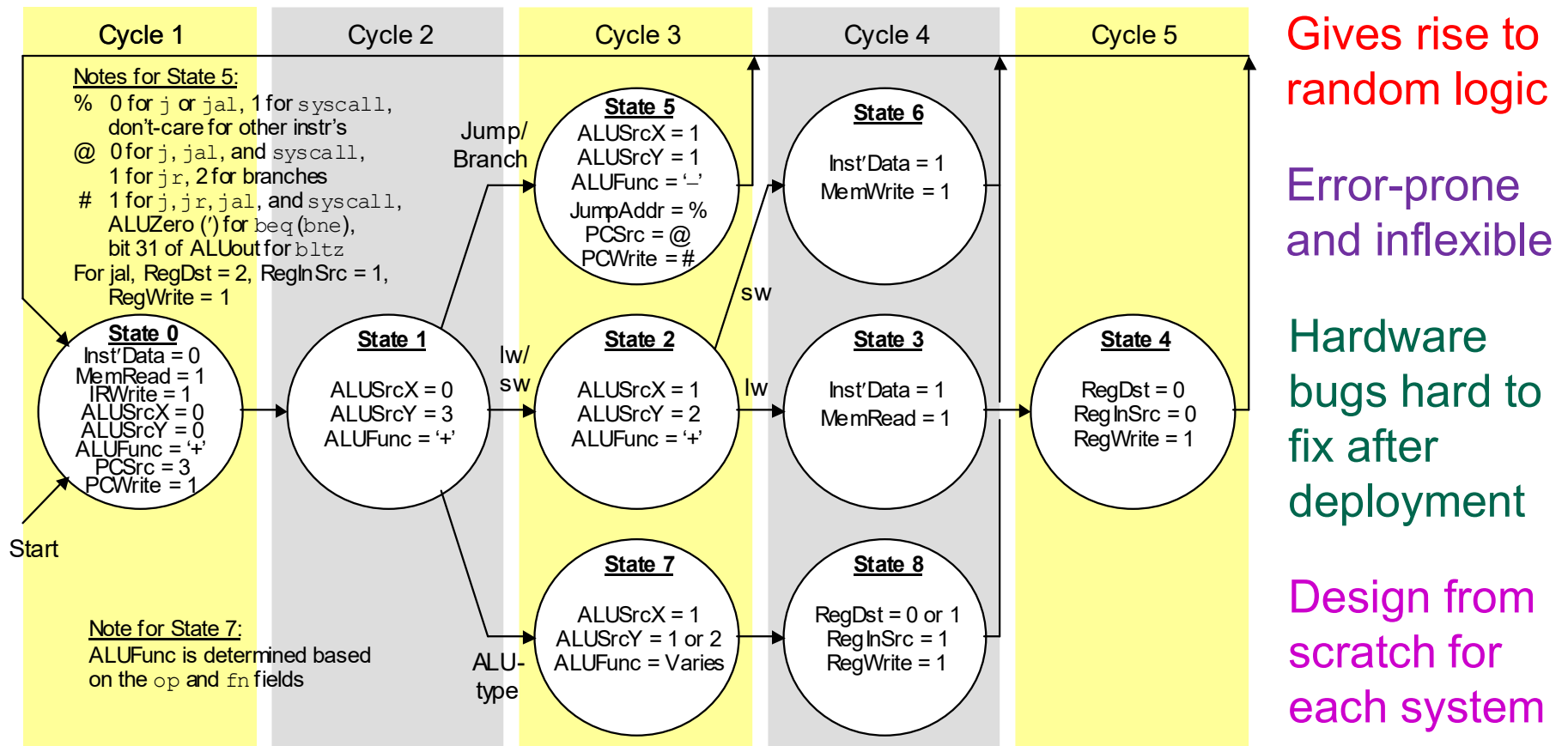


Harvard architecture
(separate memories for code & data)

Better protection of programs
Higher aggregate memory bandwidth
Memory optimization for access type

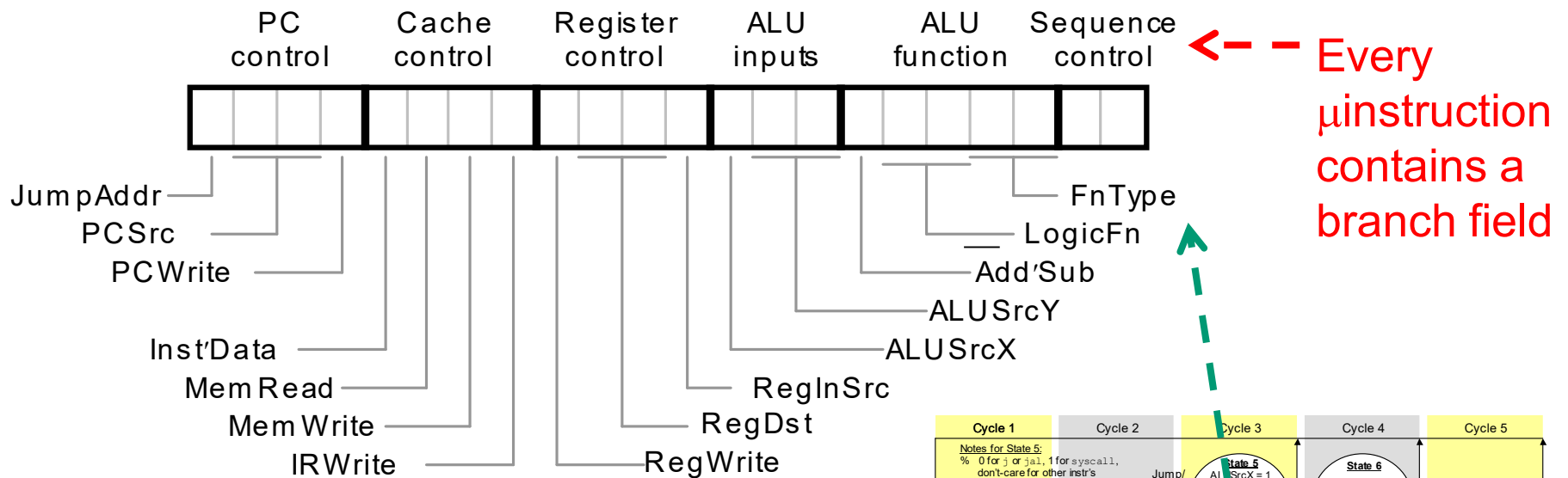
1950s: Microprogramming

Traditional control unit design (multicycle): Specify which control signals are to be asserted in each cycle and synthesize



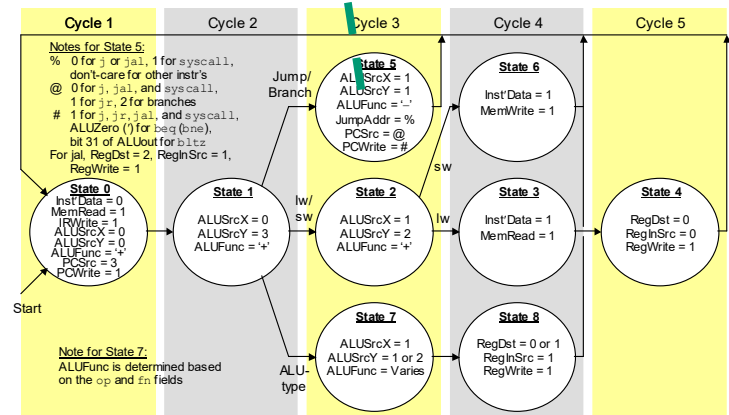
The Birth of Microprogramming

The control state machine resembles a program (microprogram) comprised of instructions (microinstructions) and sequencing



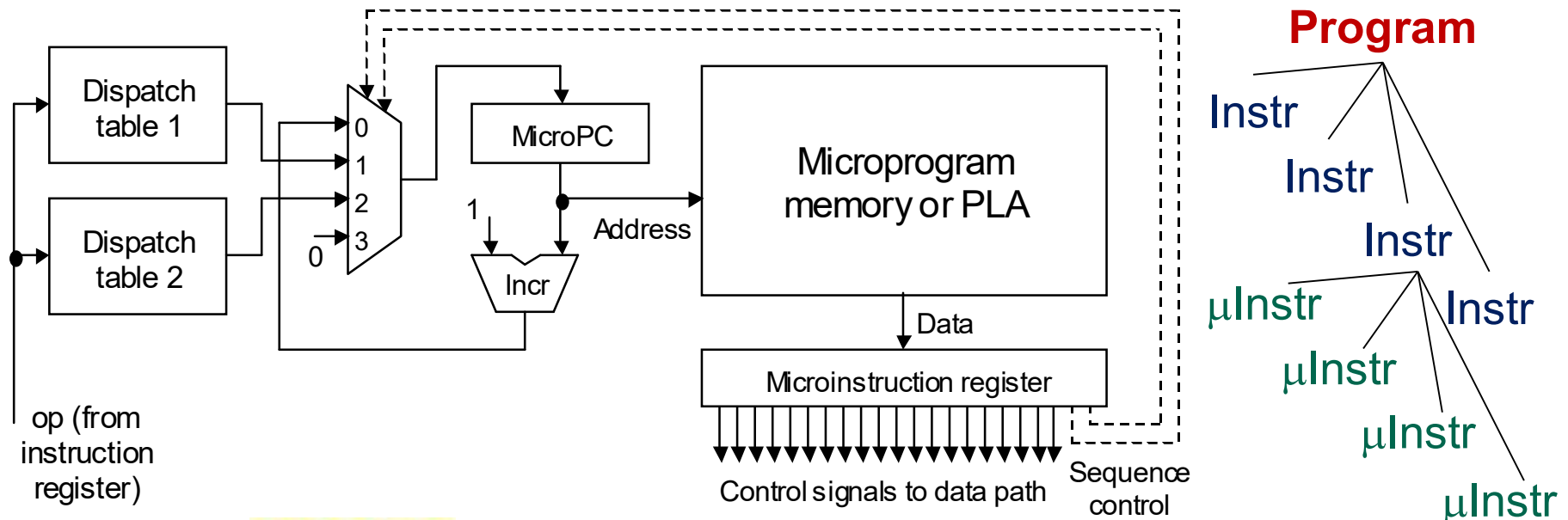
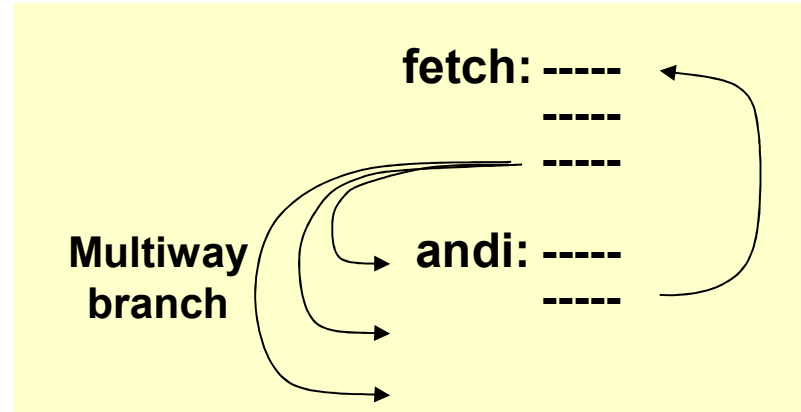
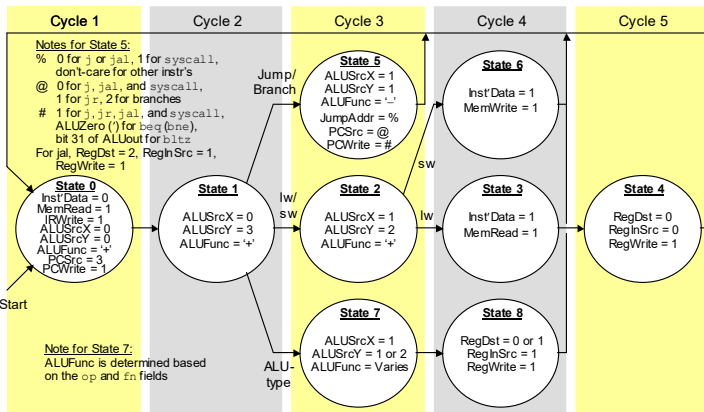
Every μ instruction contains a branch field

Maurice V. Wilkes
(1913-2010)



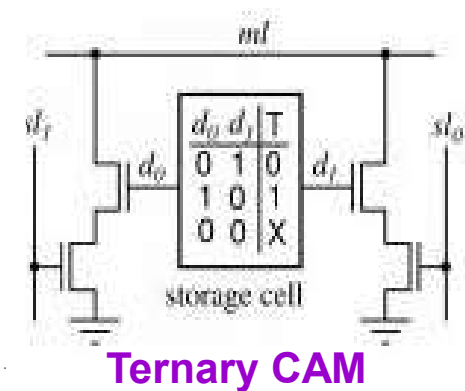
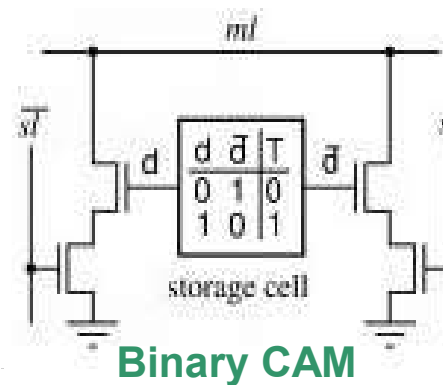
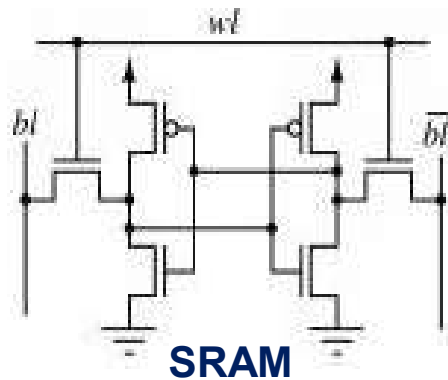
Microprogramming Implementation

Each microinstruction controls the data path for one clock cycle



1960s: Parallel Processing

Associative (content-addressed) memories and other forms of parallelism (compute-I/O overlap, functional parallelism) had been in existence since the 1940s



Highly parallel machine, proposed by Daniel Slotnick in 1964, later morphed into ILLIAC IV in 1968 (operational in 1975)

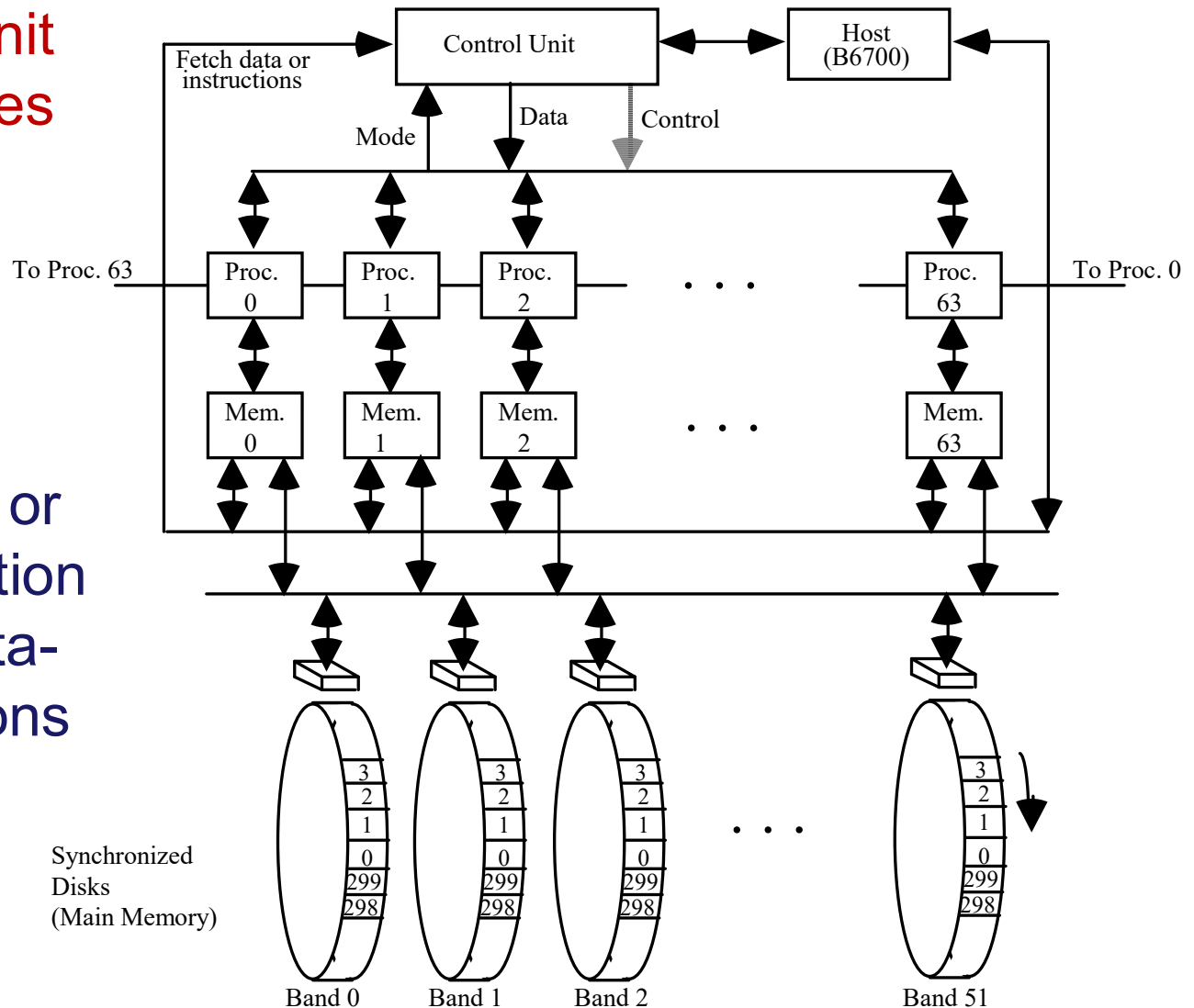
Michael J. Flynn devised his now-famous 4-way taxonomy (SISD, SIMD, MISD, MIMD) in 1966 and Amdahl formulated his speed-up law and rules for system balance in 1967

The ILLIAC IV Concept: SIMD Parallelism

Common control unit fetches and decodes instructions, broadcasting the control signals to all PEs

Each PE executes or ignores the instruction based on local, data-dependent conditions

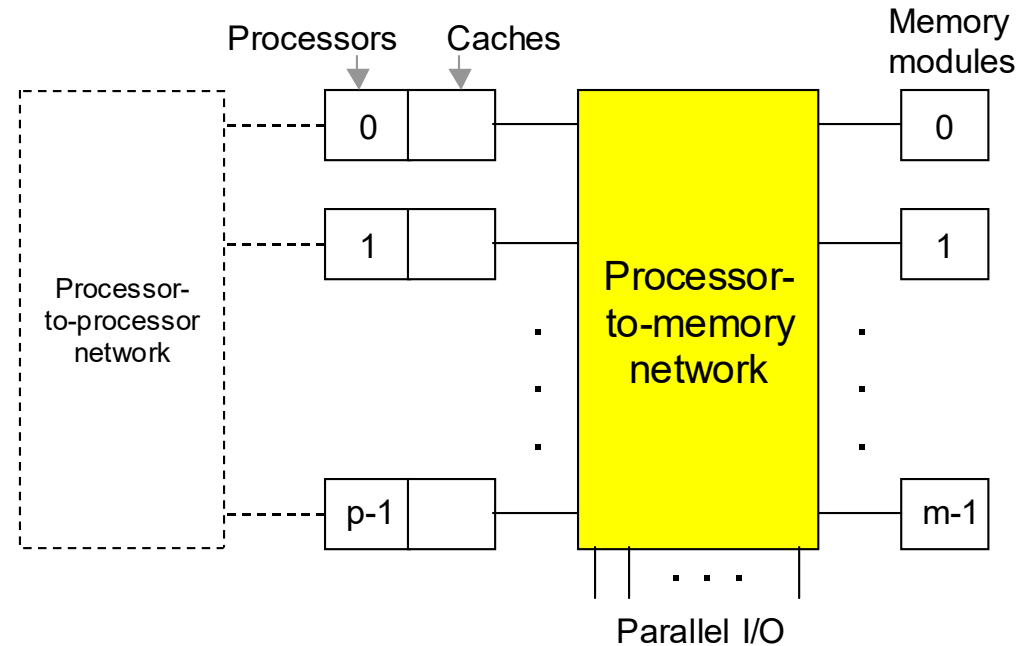
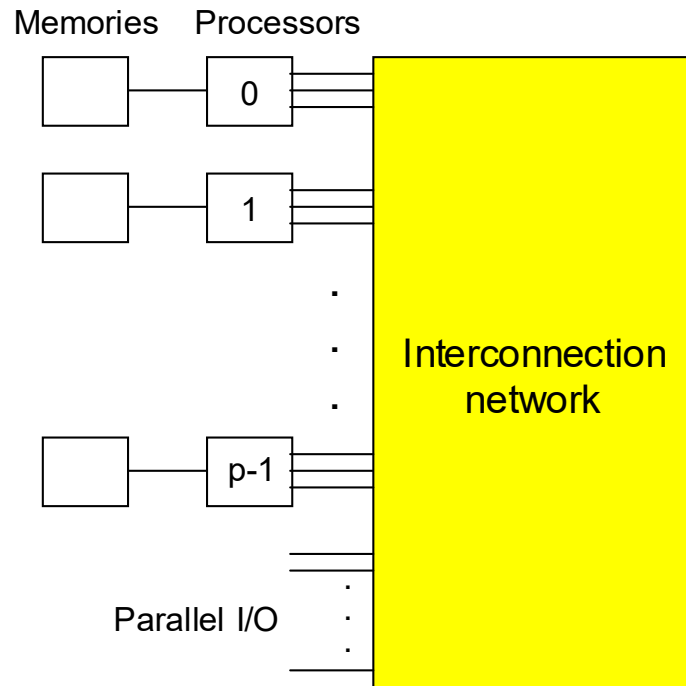
The interprocessor routing network is only partially shown



Various Forms of MIMD Parallelism

Global shared memory

- Memory latency
 - Memory bandwidth
 - Cache coherence
- Wall**



Distributed shared memory or message-passing architecture

- Scalable network performance
- Flexible and more robust
- Memory consistency model

Warehouse-Sized Data Centers

COOLING: High-efficiency water-based cooling systems—less energy-intensive than traditional chillers—circulate cold water through the containers to remove heat, eliminating the need for air-conditioned rooms.

STRUCTURE: A 24 000-square-meter facility houses 400 containers. Delivered by trucks, the containers attach to a spine infrastructure that feeds network connectivity, power, and water. The data center has no conventional raised floors.

POWER: Two power substations feed a total of 300 megawatts to the data center, with 200 MW used for computing equipment and 100 MW for cooling and electrical losses. Batteries and generators provide backup power.

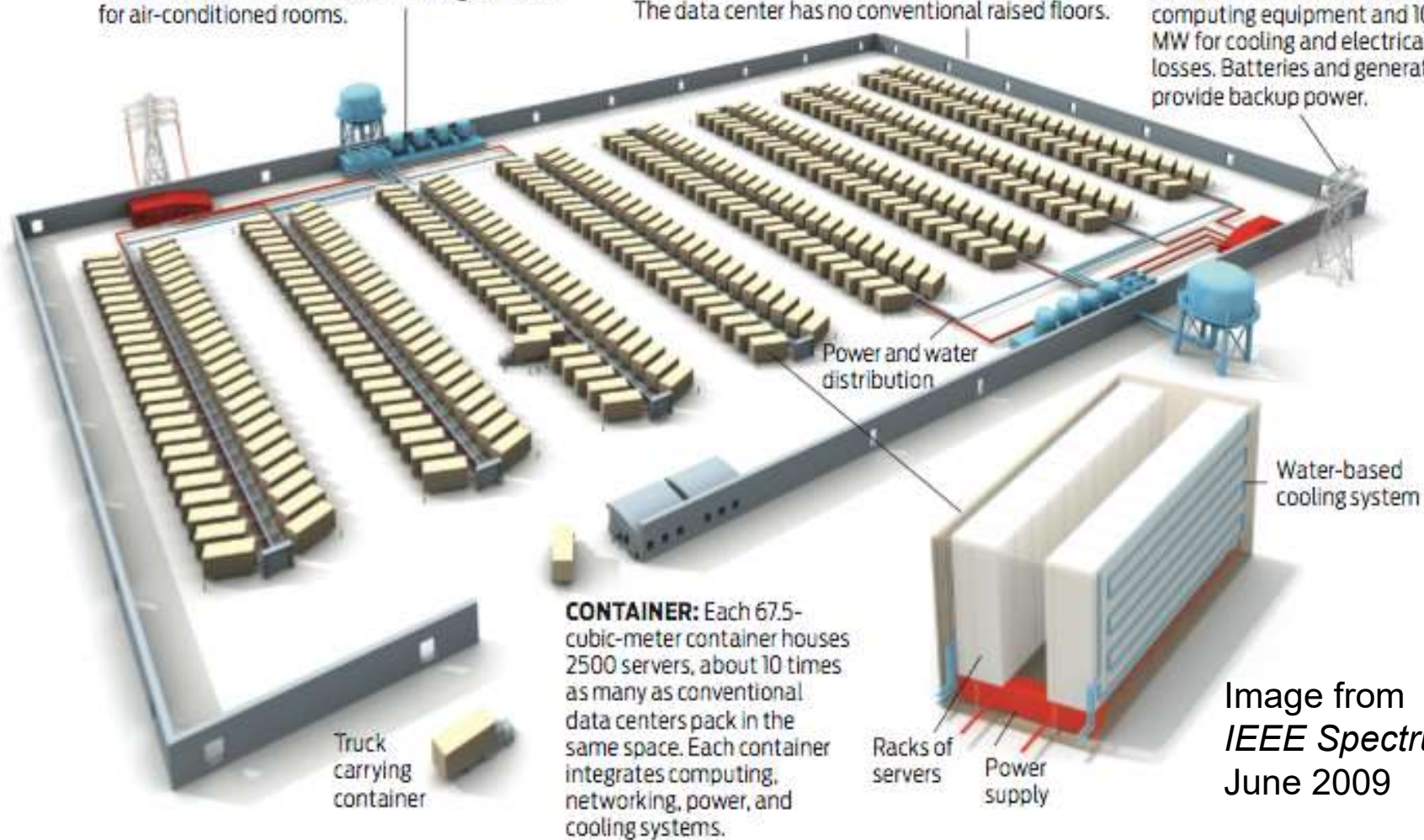
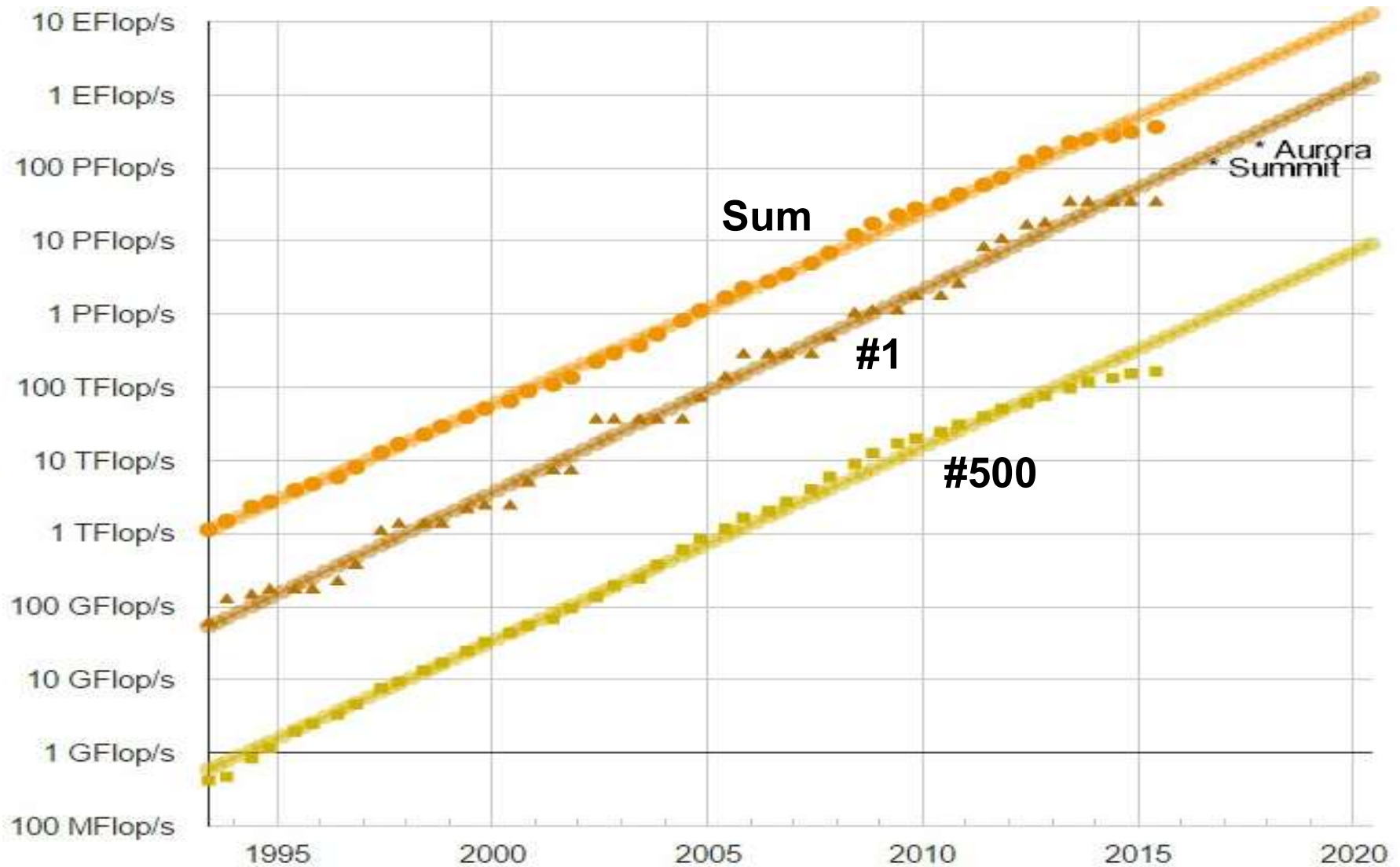
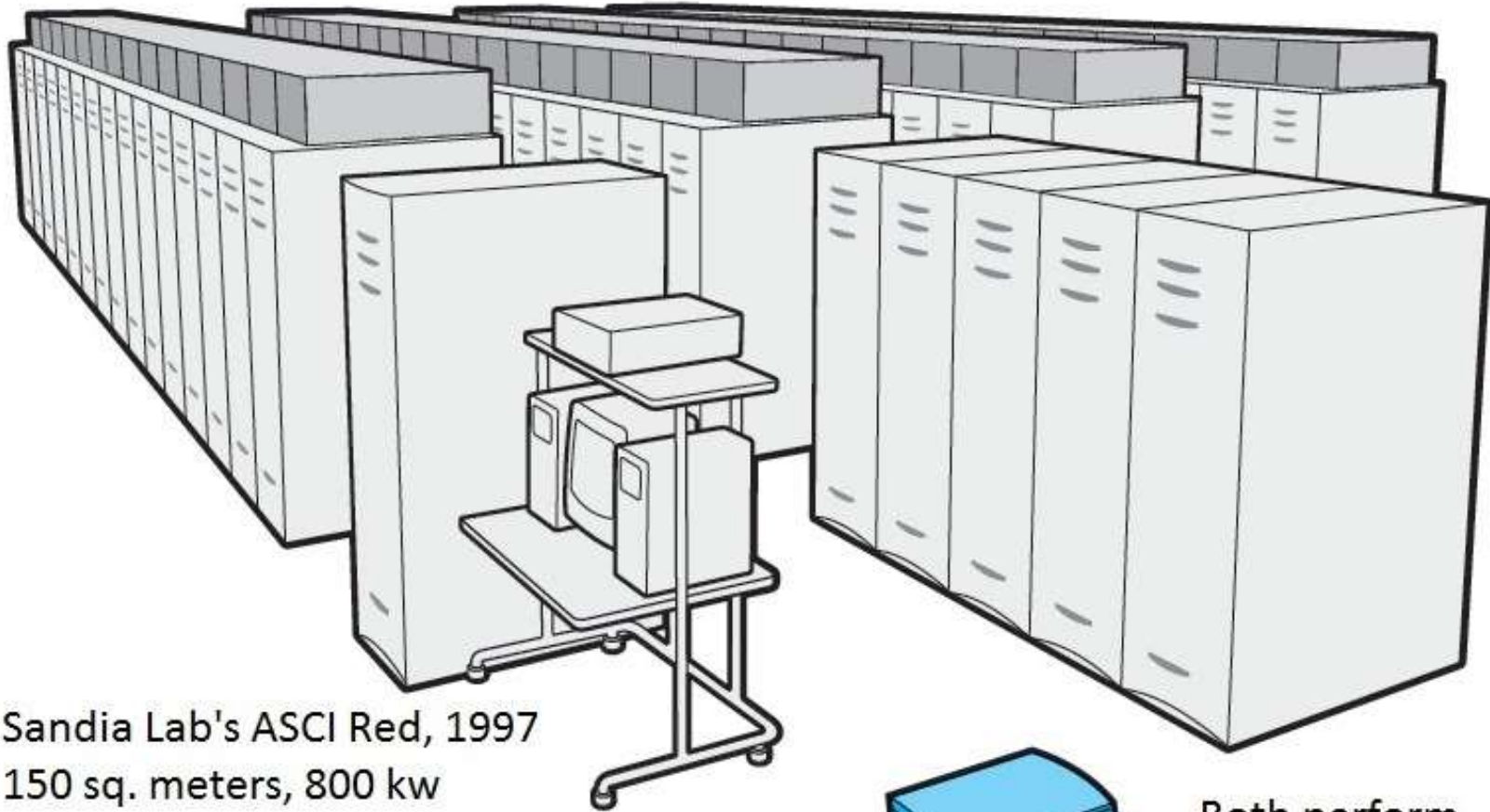


Image from
IEEE Spectrum,
June 2009

Top 500 Supercomputers in the World



The Shrinking Supercomputer



Sandia Lab's ASCI Red, 1997
150 sq. meters, 800 kw

Sony Playstation, 2006
0.08 sq. meter, < 0.2 kw

Both perform
at ~2 TFLOPS

Status of Computing Power circa 2000 (2010 2020)

Giga = 10^9

GFLOPS on desktop:

Apple Macintosh, with G4 processor

TFLOPS

PFLOPS
NVIDIA AI chip

Tera = 10^{12}

TFLOPS in supercomputer center:

1152-proc IBM RS/6000 SP; Cray T3E

PFLOPS

EFLOPS
folding@home

Peta = 10^{15}

PFLOPS on the drawing board:

1M-processor IBM Blue Gene (2005?)

32 proc/chip, 64 chips/board, 8 boards/tower, 64 towers

Processor: 8 threads, on-chip memory, no data cache

Chip: defect-tolerant, row/column rings in a 6×6 array

Board: 8×8 chip grid organized as $4 \times 4 \times 4$ cube

Tower: Boards linked to 4 neighbors in adjacent towers

System: $32 \times 32 \times 32$ cube of chips, 1.5 MW (water-cooled)

Exa = 10^{18}

EFLOPS

Zeta = 10^{21}

ZFLOPS

1970s: Cache Memory

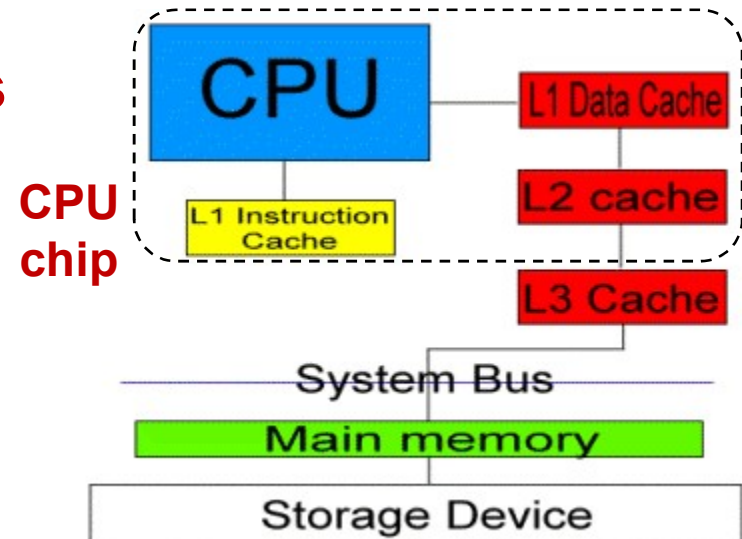
First paper on “buffer” memory: Maurice Wilkes, 1965

First implementation of a general cache memory:
IBM 360 Model 85 (J. S. Liptay; *IBM Systems J.*, 1968)

Broad understanding, varied implementations, and studies of optimization and performance issues in the 1970s

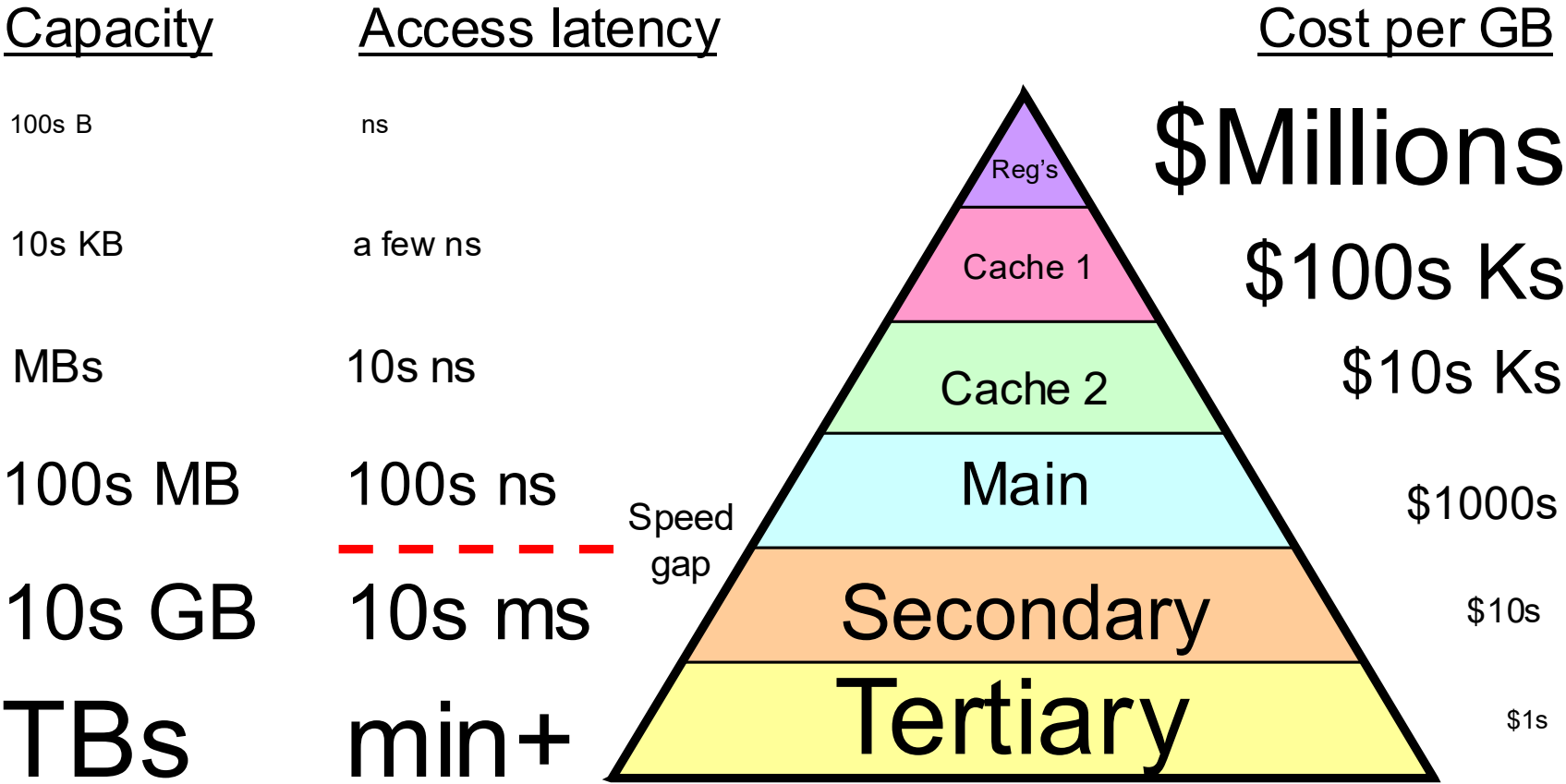
Modern cache implementations

- Harvard arch for L1 caches
- von Neumann arch higher up
- Many other caches in system besides processor caches



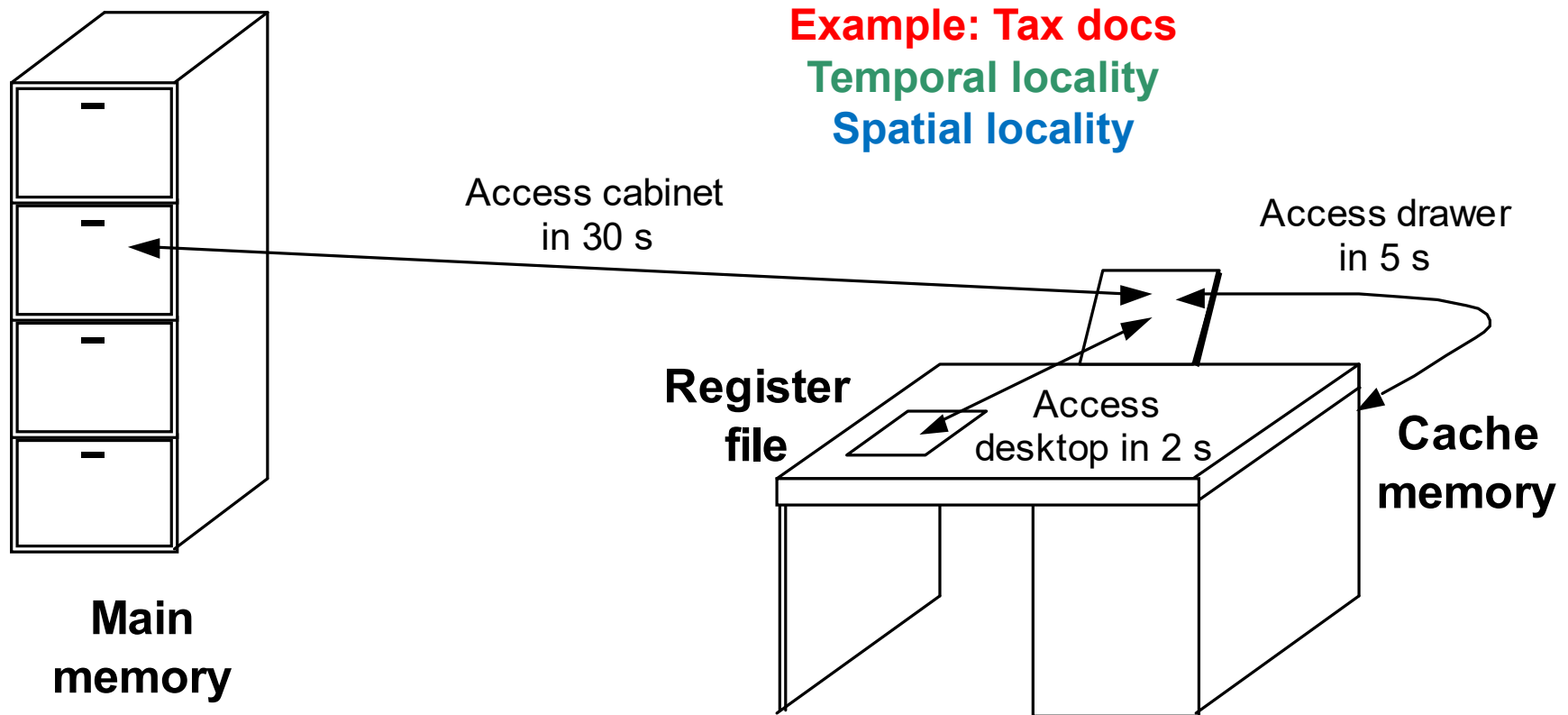
Memory Hierarchy

Hierarchical memory provides the illusion that high speed and large size are achieved simultaneously



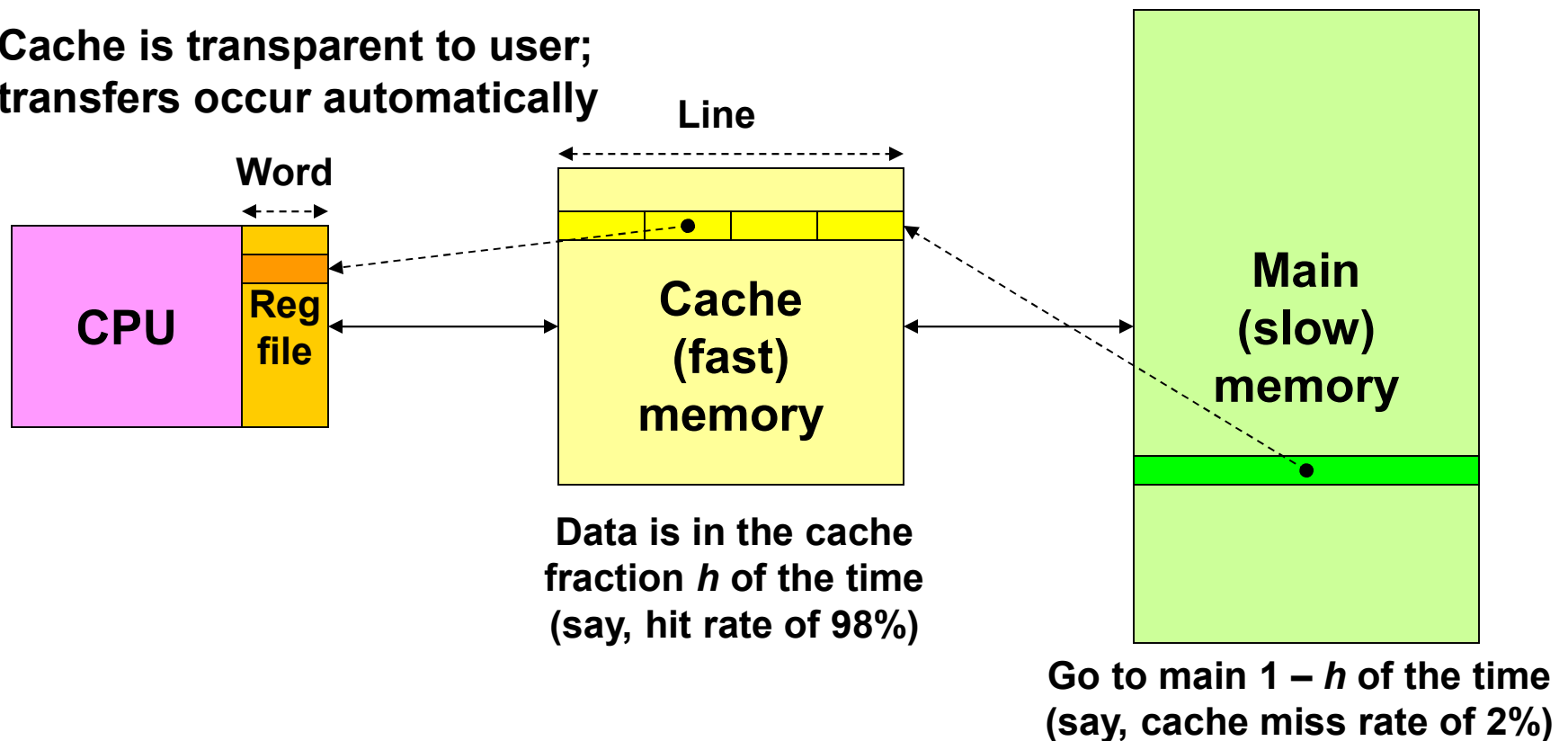
Cache Memory Analogy

Hierarchical memory provides the illusion that high speed and large size are achieved simultaneously



Hit/Miss Rate, and Effective Cycle Time

Cache is transparent to user;
transfers occur automatically



One level of cache with hit rate h

$$C_{\text{eff}} = hC_{\text{fast}} + (1 - h)(C_{\text{slow}} + C_{\text{fast}}) = C_{\text{fast}} + (1 - h)C_{\text{slow}}$$

The Locality Principle

From Peter Denning's *CACM* paper,
July 2005 (Vol. 48, No. 7, pp. 19-24)

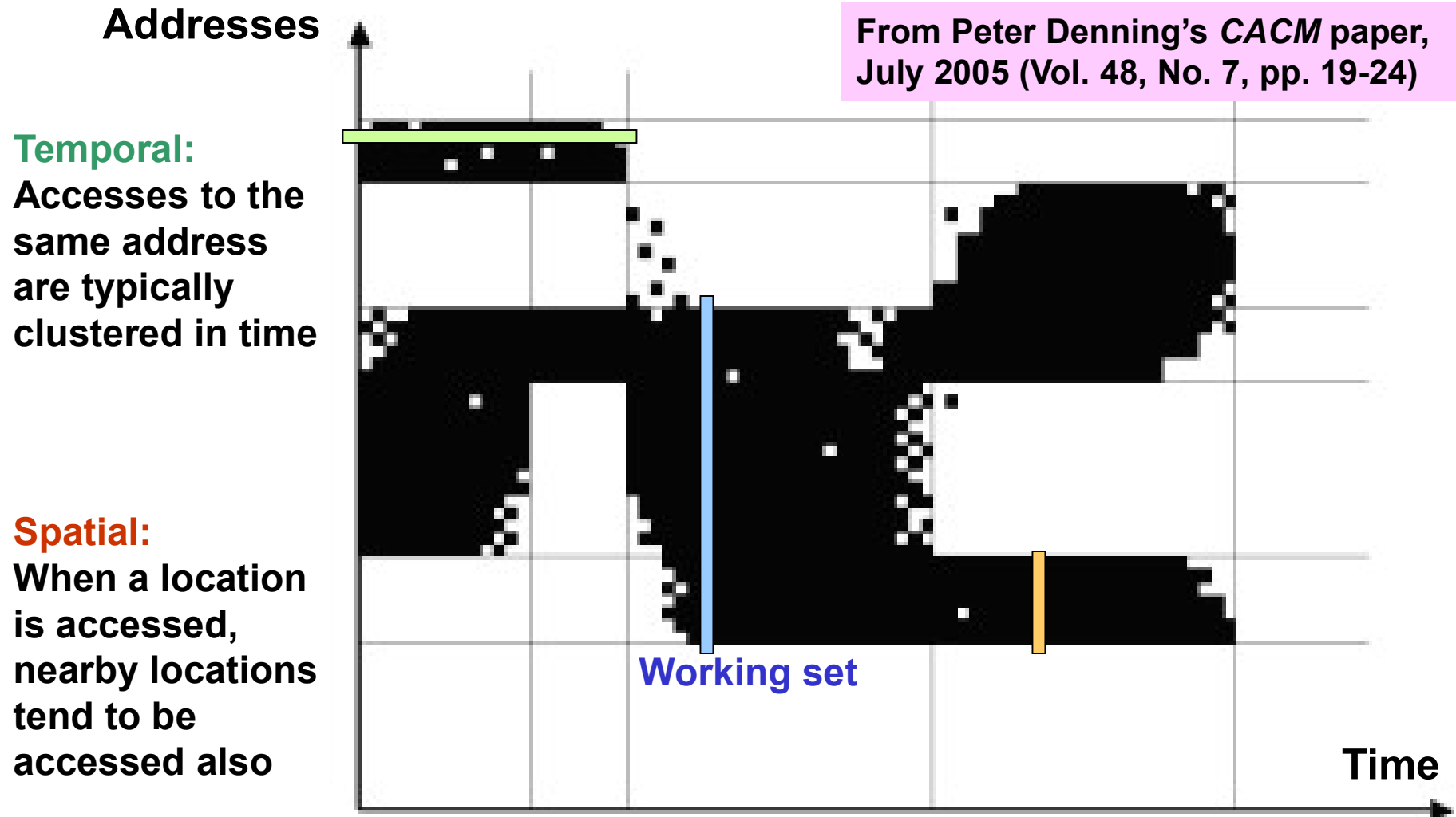


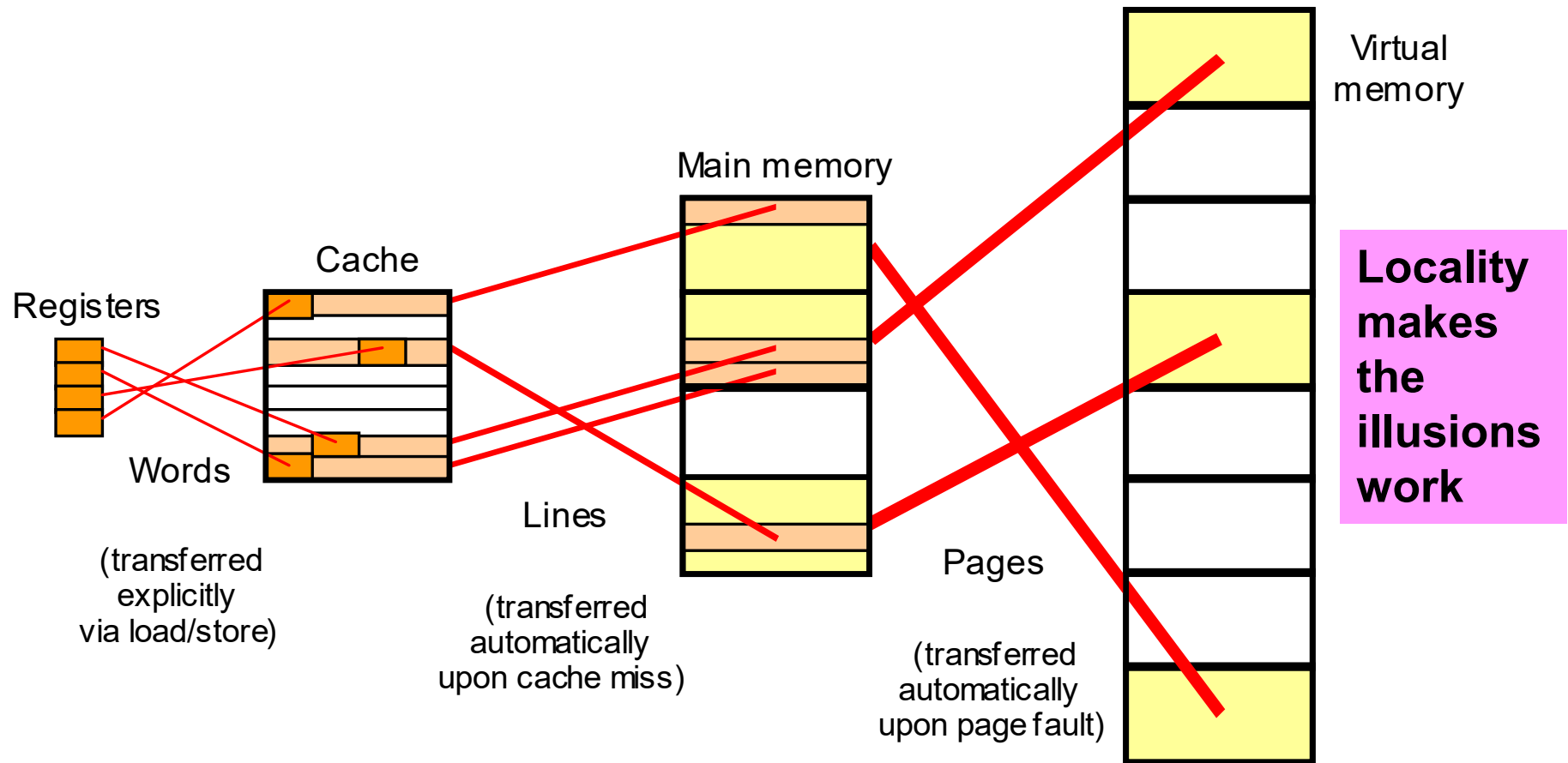
Illustration of temporal and spatial localities

Summary of Memory Hierarchy

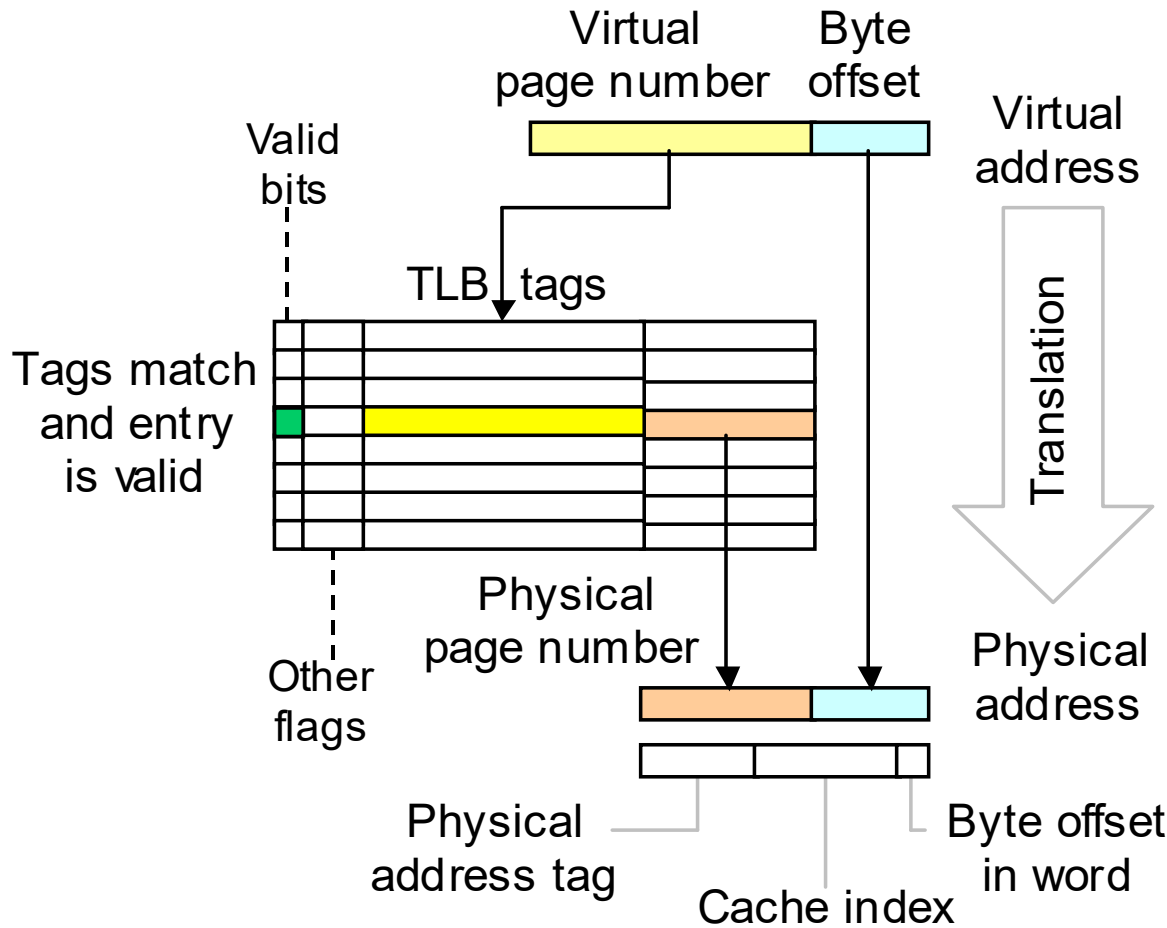
Cache memory:
provides illusion
of very high speed

Main memory:
reasonable cost,
but slow & small

Virtual memory:
provides illusion of
very large size



Translation Lookaside Buffer



Program page in virtual memory

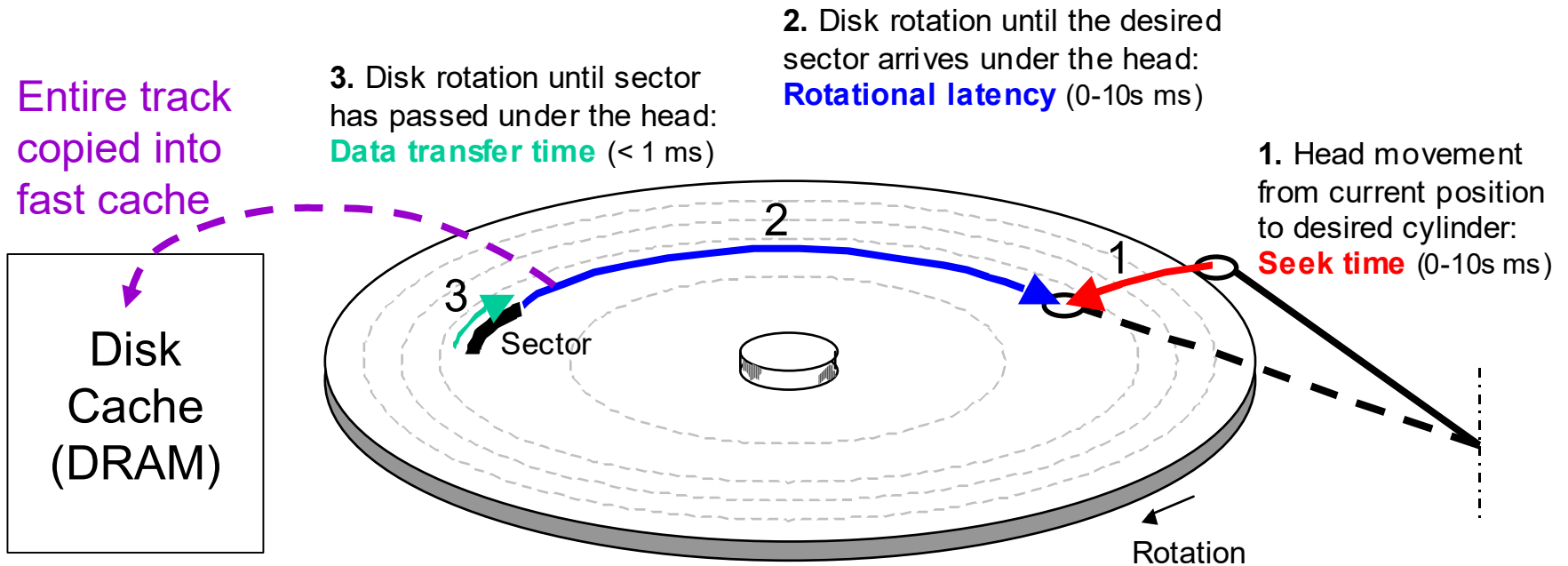
```

...
lw    $t0, 0($s1)
addi  $t1, $zero, 0
L:    add  $t1, $t1, 1
      beq  $t1, $s2, D
      add  $t2, $t1, $t1
      add  $t2, $t2, $t2
      add  $t2, $t2, $s1
      lw  $t3, 0($t2)
      slt  $t4, $t0, $t3
      beq  $t4, $zero, L
      addi $t0, $t3, 0
      j   L
D:    ...
    
```

All instructions on this page have the same virtual page address and thus entail the same translation

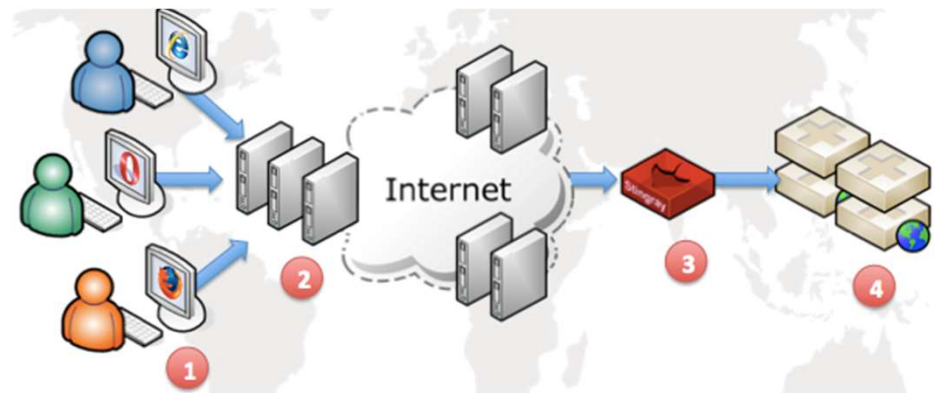
Virtual-to-physical address translation by a TLB and how the resulting physical address is used to access the cache memory.

Disk Caching and Other Applications



Web caching

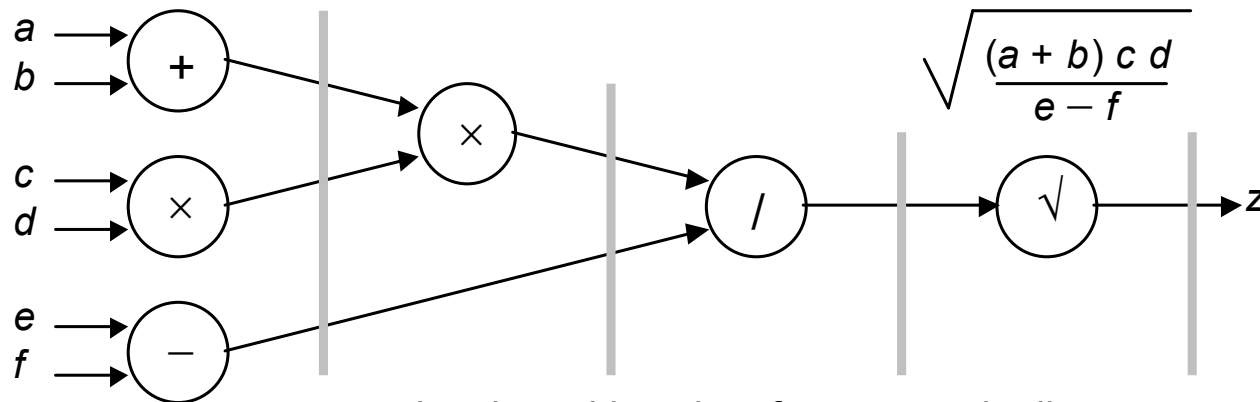
- Client-side caching
- Caching within the cloud
- Server-side caching



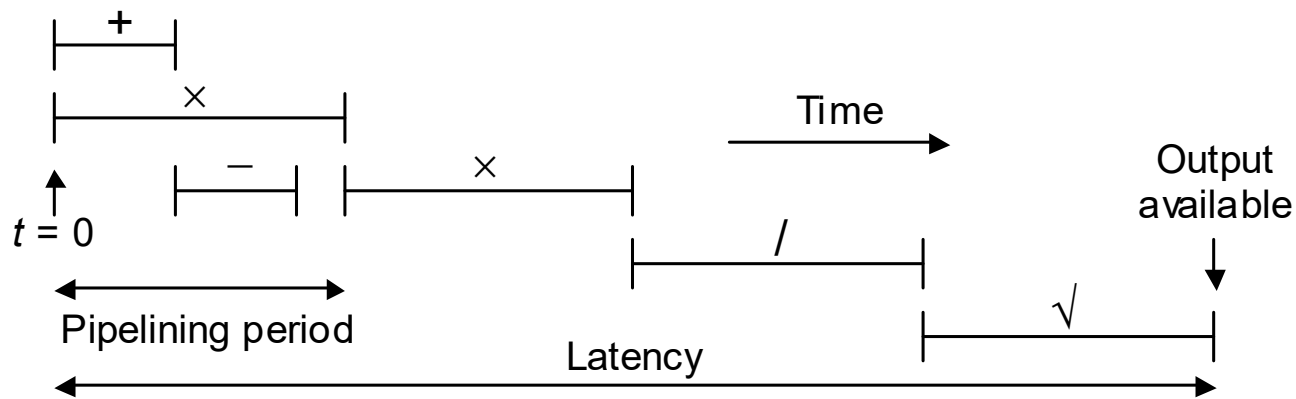
1980s: Pipelining

An important form of parallelism that is given its own name

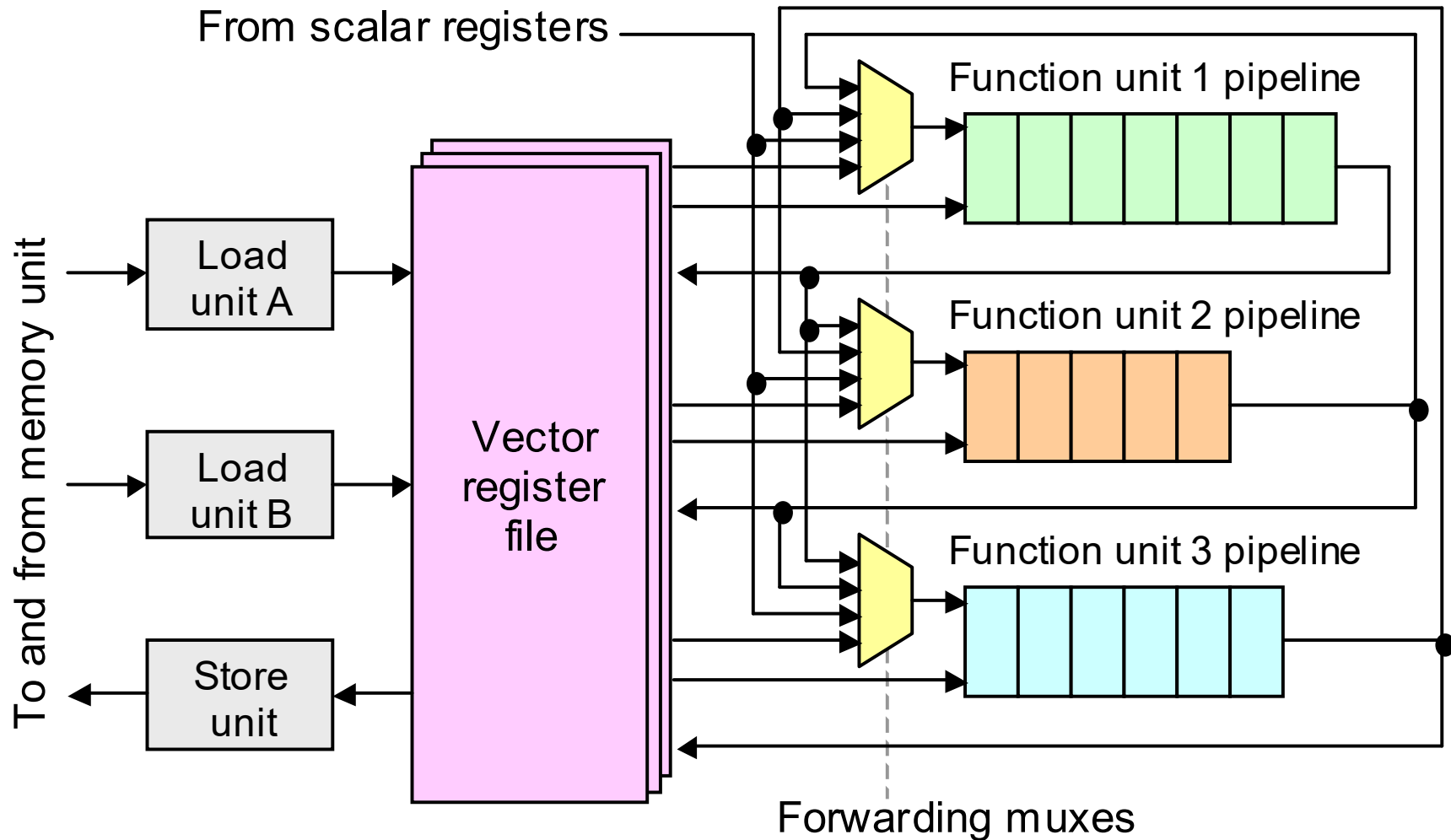
Used from early days of digital circuits in various forms



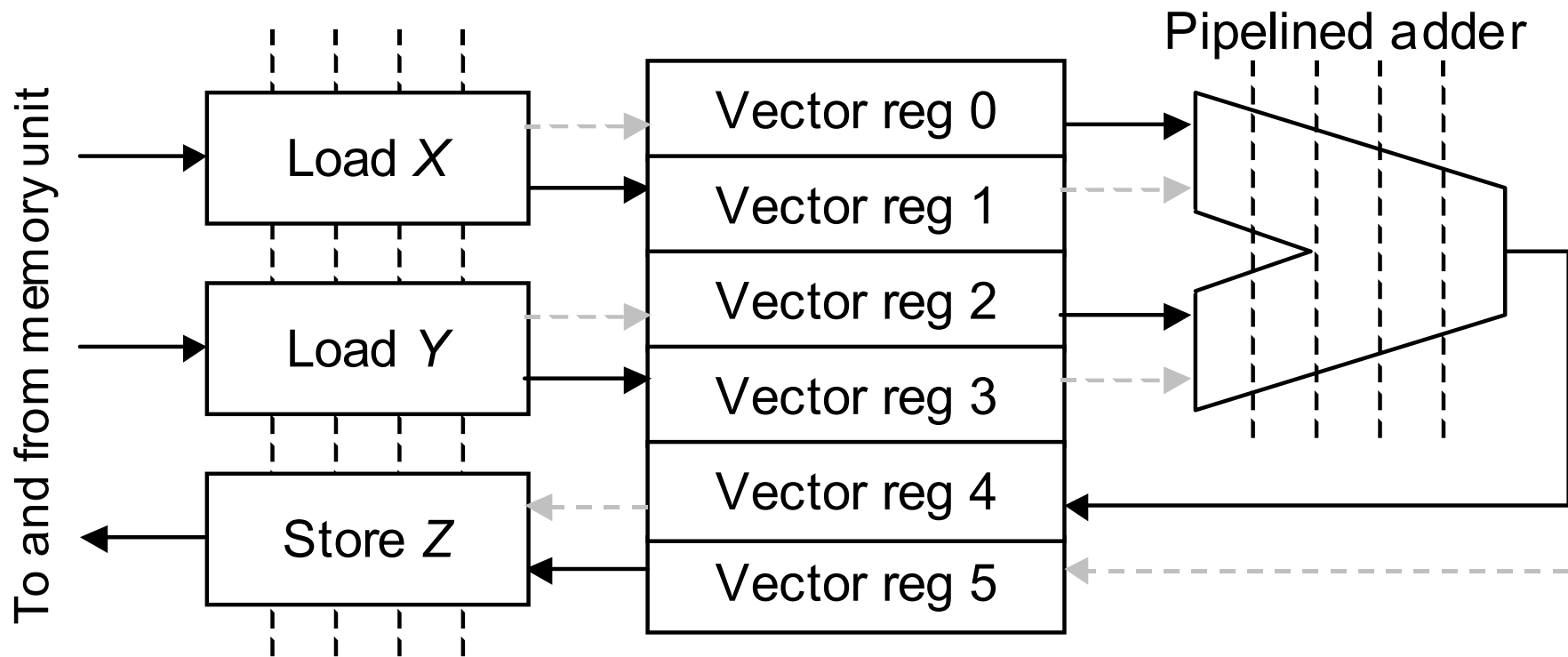
Latch positions in a four-stage pipeline



Vector Processor Implementation

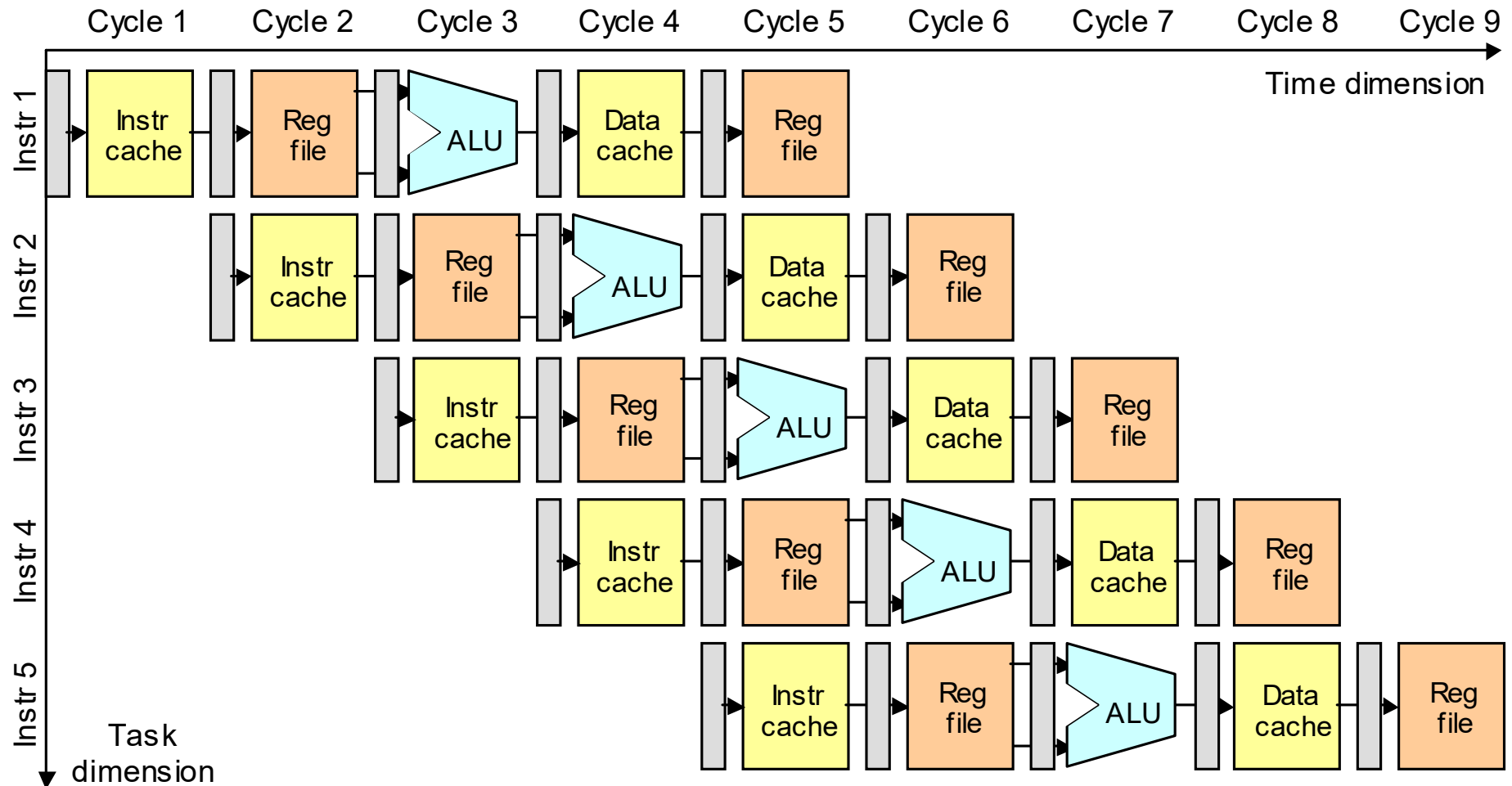


Overlapped Load/Store and Computation



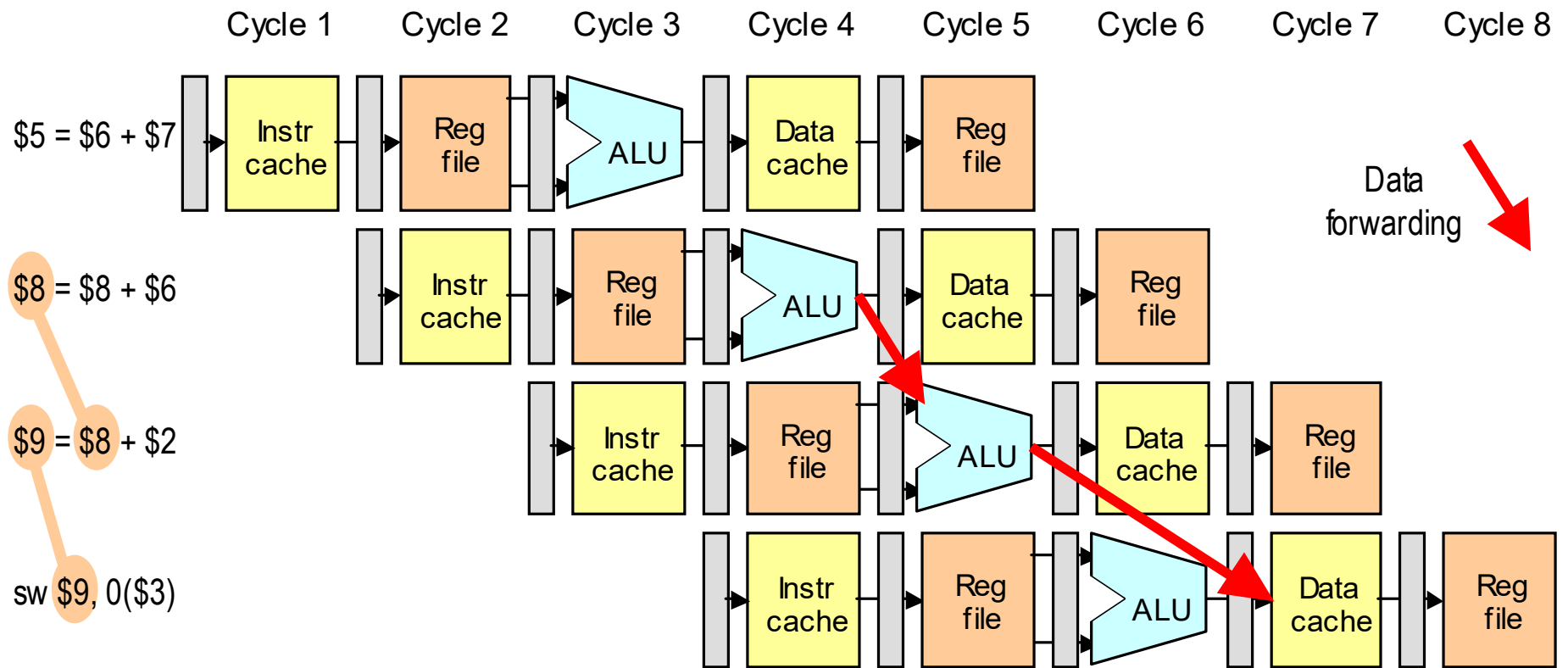
Vector processing via segmented load/store of vectors in registers in a double-buffering scheme. Solid (dashed) lines show data flow in the current (next) segment.

Simple Instruction-Execution Pipeline

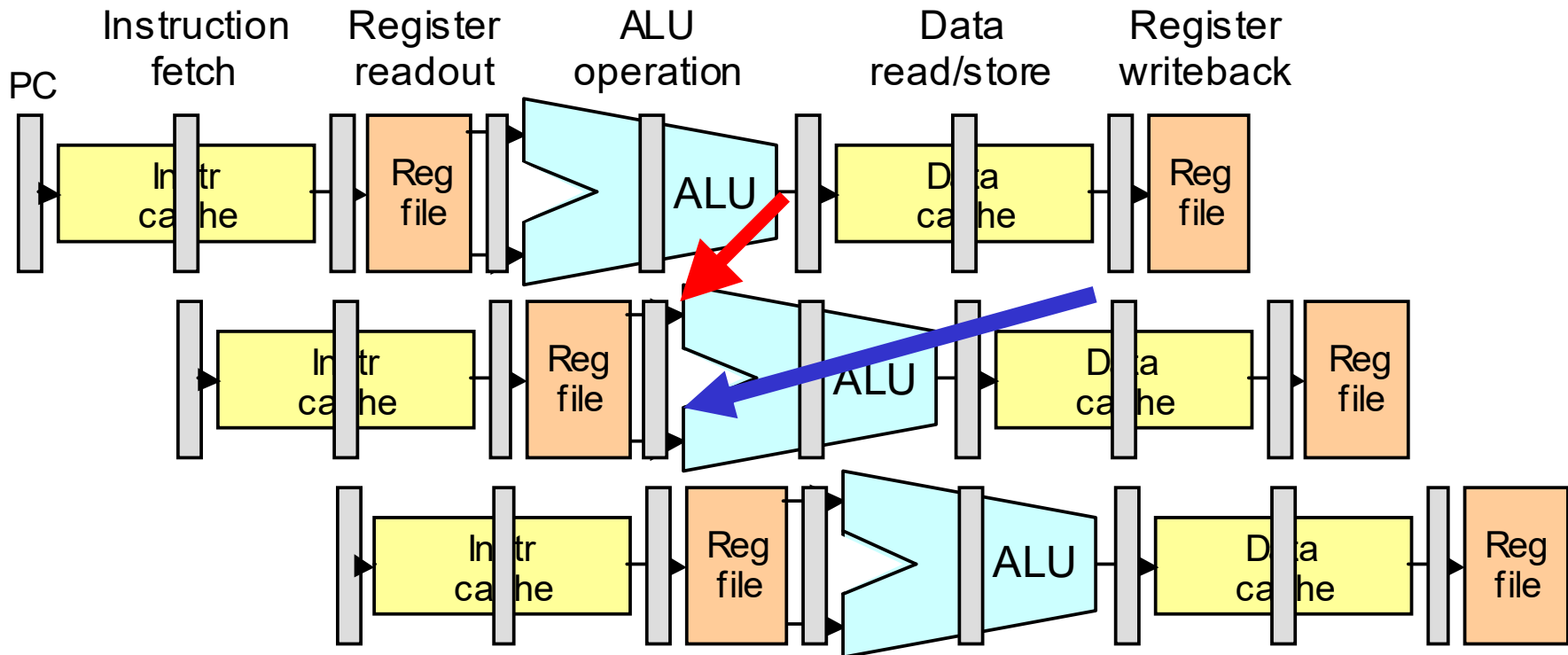


Pipeline Stalls or Bubbles

Data dependency and its possible resolution via forwarding

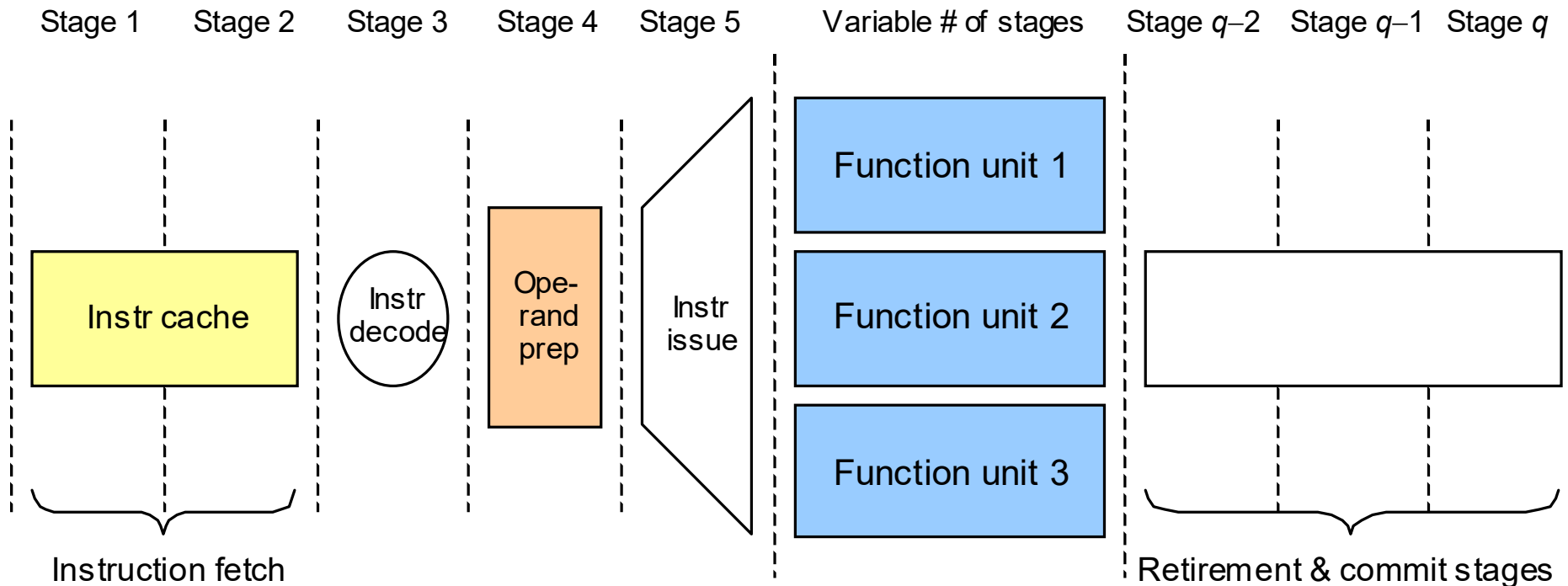


Problems Arising from Deeper Pipelines



Forwarding more complex and not always workable
Interlocking/stalling mechanisms needed to prevent errors

Branching and Other Complex Pipelines



Front end:	In-order or out-of-order	}
Instr. issue:	In-order or out-of-order	
Write-back:	In-order or out-of-order	
Commit:	In-order or out-of-order	

**The more OoO stages,
the higher the complexity**

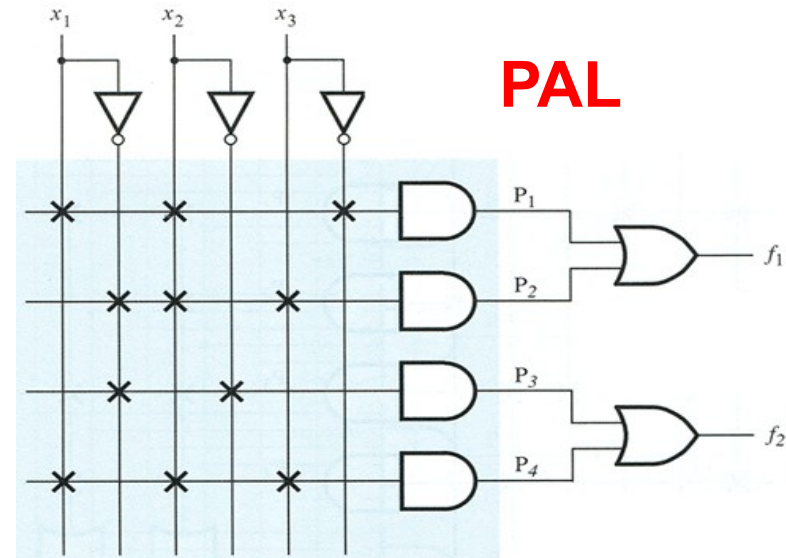
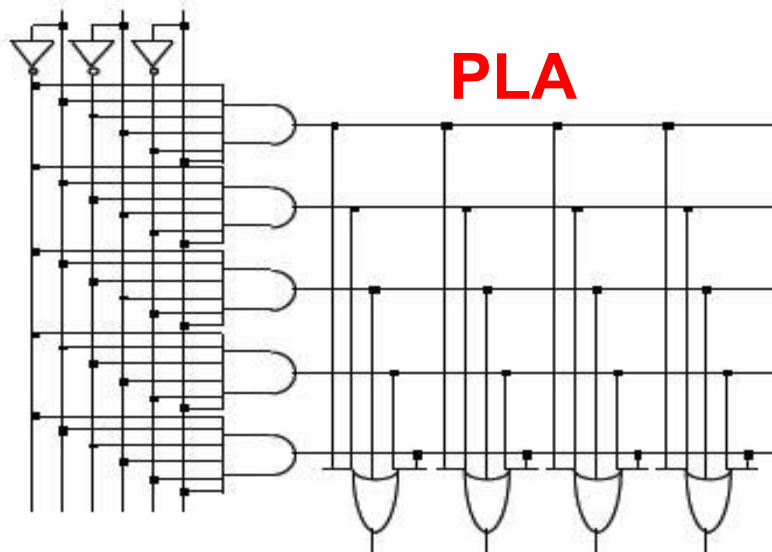
1990s: FPGAs

Programmable logic arrays were developed in the 1970s

PLAs provided cost-effective and flexible replacements for random logic or ROM/PROM

The related programmable array logic devices came later

PALs were less flexible than PLAs, but more cost-effective



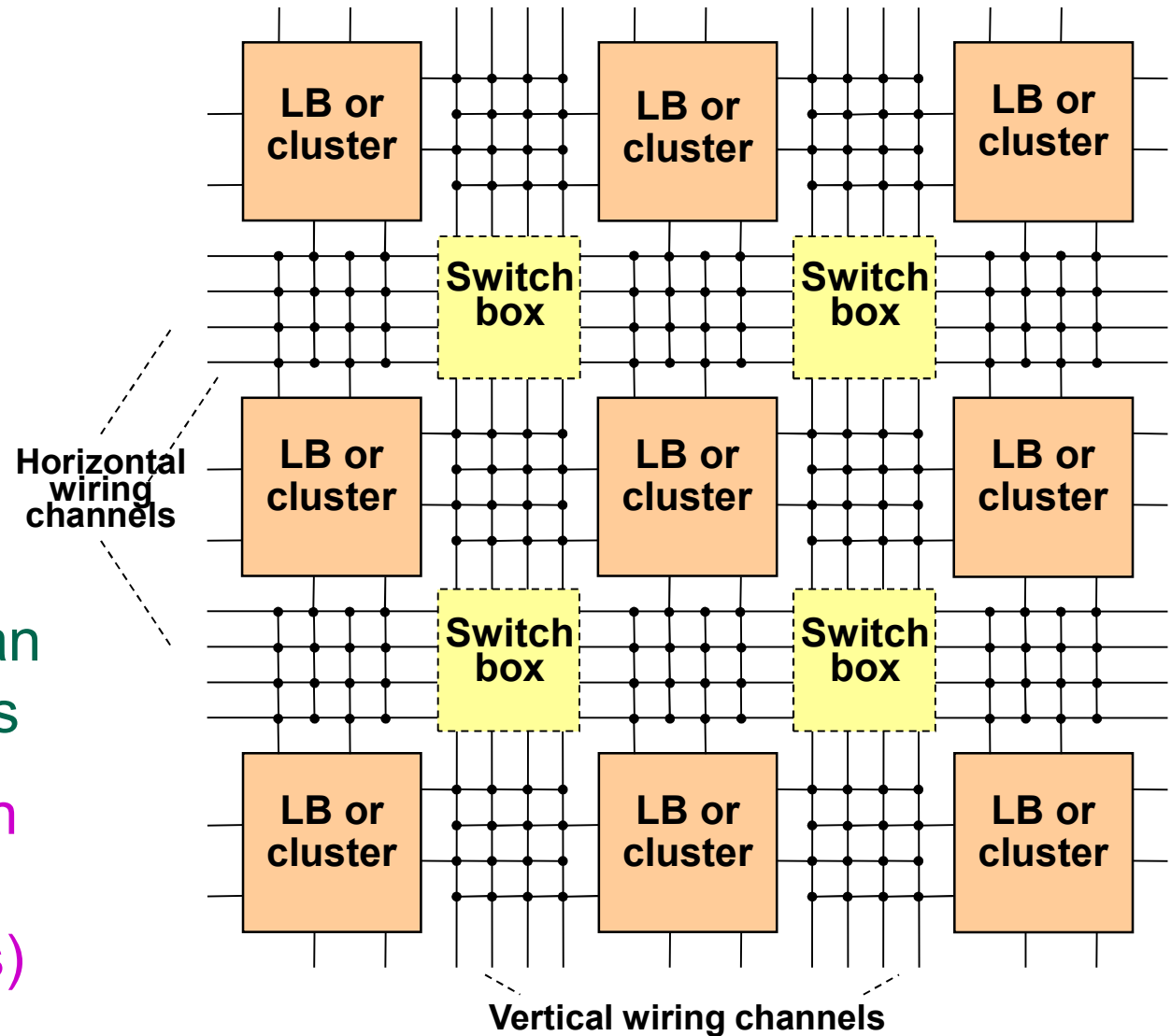
Why FPGA Represents a Paradigm Shift

Modern FPGAs can implement any functionality

Initially used only for prototyping

Even a complete CPU needs a small fraction of an FPGA's resources

FPGAs come with multipliers and IP cores (CPUs/SPs)



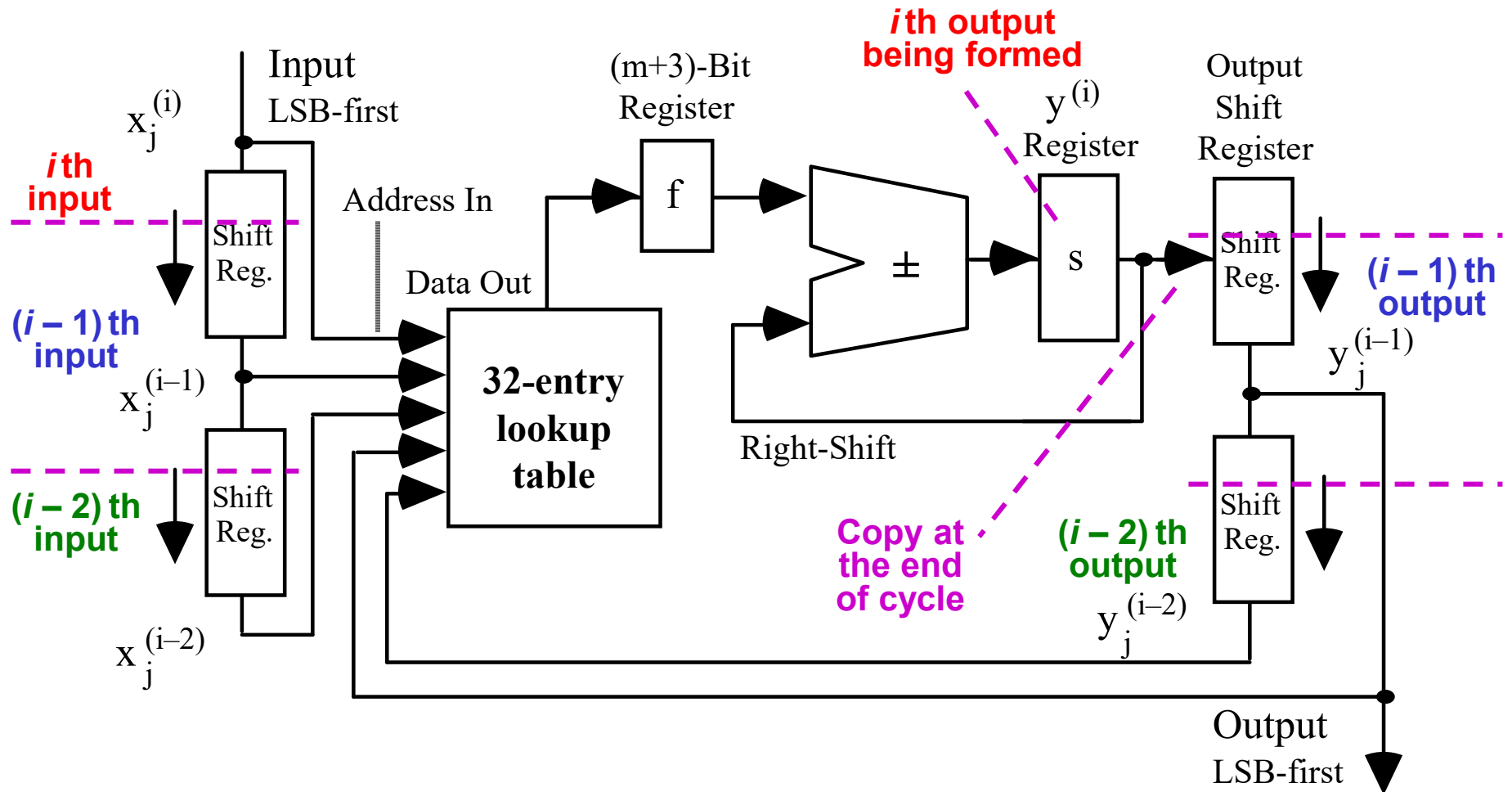
FPGAs Are Everywhere

Applications are found in virtually all industry segments:

- Aerospace and defense
- Medical electronics
- Automotive control
- Software-defined radio
- Encoding and decoding



Example: Bit-Serial 2nd-Order Digital Filter



LUTs, registers, and an adder are all we need for linear expression evaluation: $y^{(i)} = ax^{(i)} + bx^{(i-1)} + cx^{(i-2)} + dy^{(i-1)} + ey^{(i-2)}$

2000s: GPUs

Simple graphics and signal processing units were used since the 1970s

In the early 2000s, the two major players, ATI and Nvidia, produced powerful chips to improve the speed of shading

In the late 2000s, GPGPUs (extended stream processors) emerged and were used in lieu of, or in conjunction with, CPUs in high-performance supercomputers

GPUs are faster and more power-efficient than CPUs.

GPUs use a mixture of parallel processing and functional specialization to achieve super-high performance

CPU vs. GPU Organization

Small number of powerful cores
versus

Very large number of simple stream processors

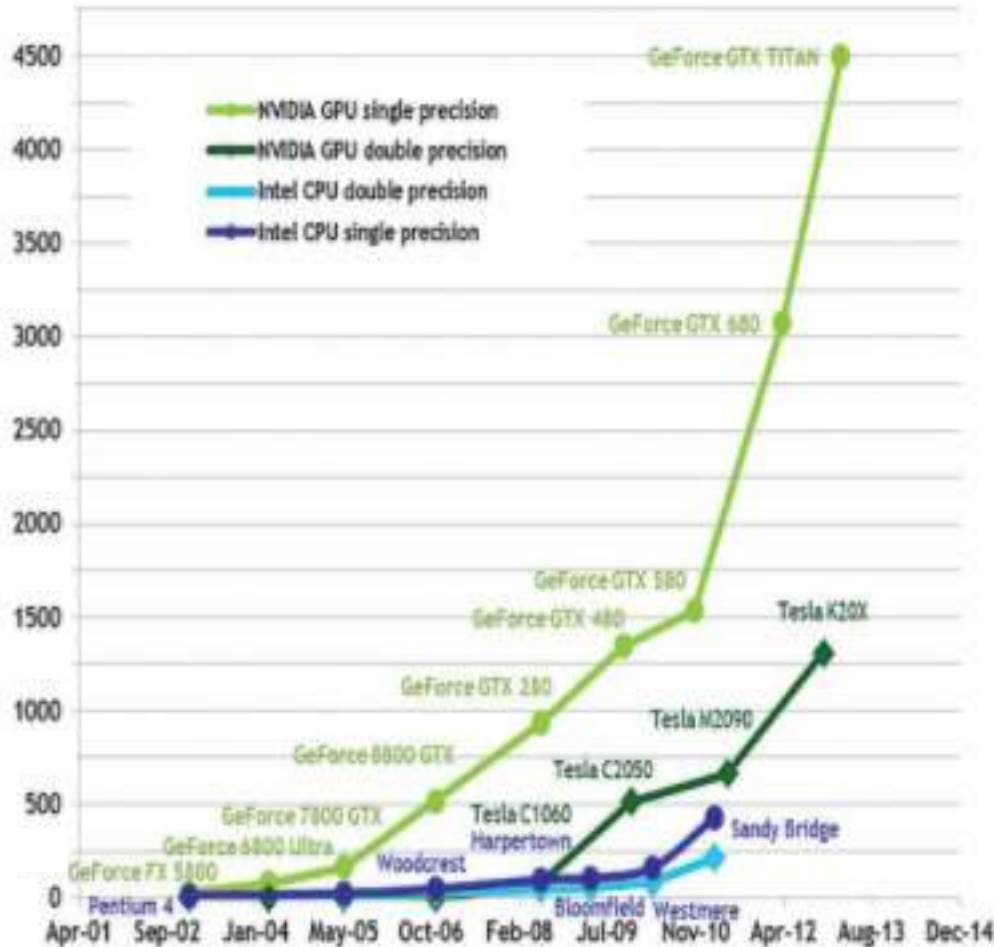
Demo (analogy for MPP): <https://www.youtube.com/watch?v=fKK933KK6Gg>



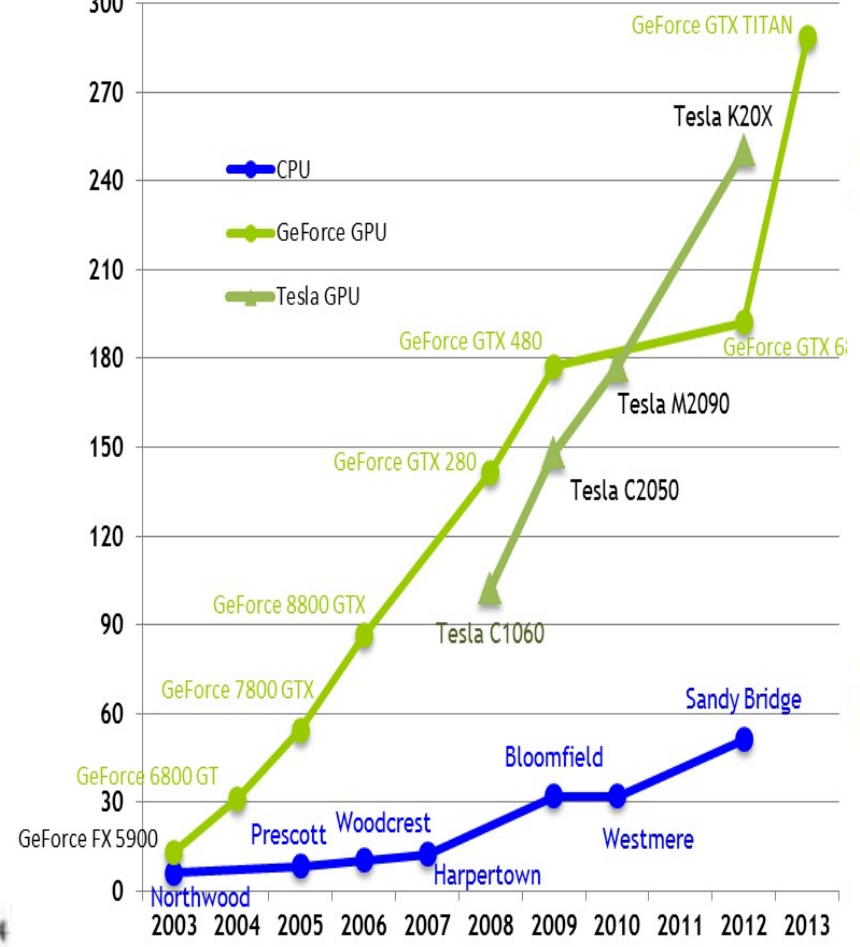
CPU vs. GPU Performance

Peak performance (GFLOPS) and peak data rate (GB/s)

GFLOP/s



GB/s



General-Purpose Computing on GPUs

Suitable for numerically intensive matrix computations

First application to run faster on a GPU was LU factorization

Users can ignore GPU features and focus on problem solving

- Nvidia CUDA Programming System
- Matlab Parallel Computing Toolbox
- C++ Accelerated Massive Parallelism

Many vendors now give users direct access to GPU features



Example system (Titan):
Cray XK7 at DOE's Oak Ridge Nat'l Lab used more than 1/4 M Nvidia K20x cores to accelerate computations (energy-efficient: 2+ gigaflops/W)

The Eight Key Ideas

2010s Specialization

2000s GPUs

1990s FPGAs

1980s Pipelining

1970s Cache memory

1960s Parallel processing

1950s Microprogramming

1940s Stored program

Methodological advances
Performance improvements



Innovations for Improved Performance

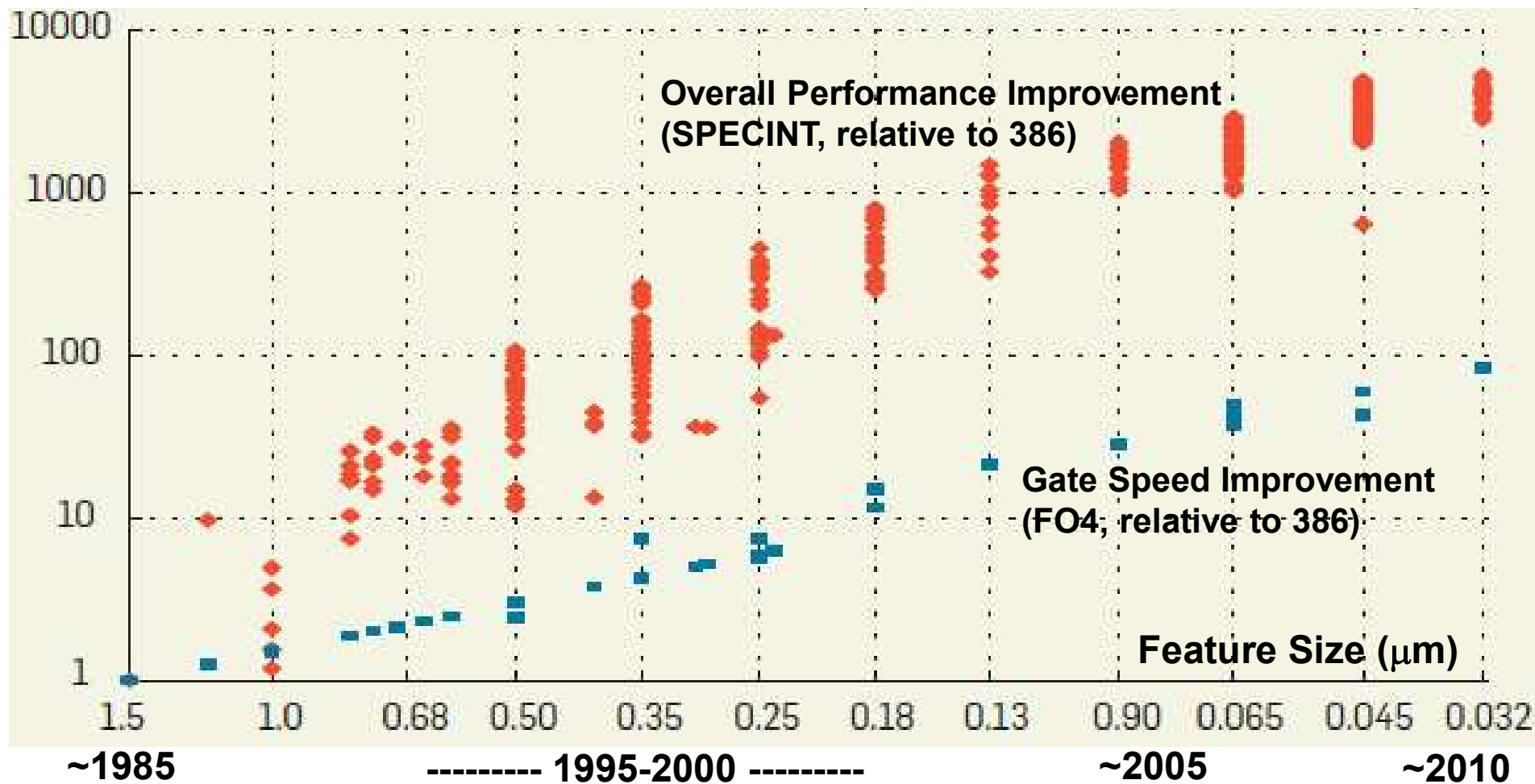
(Parhami: *Computer Architecture*, 2005)

Computer performance grew by a factor of about 10000 between 1980 and 2000
 100 due to faster technology
 100 due to better architecture

Available computing power ca. 2000:
 GFLOPS on desktop
 TFLOPS in supercomputer center
 PFLOPS on drawing board

	<u>Architectural method</u>	<u>Improvement factor</u>	
Established methods	1. Pipelining (and superpipelining)	3-8 ✓	} Previously discussed
	2. Cache memory, 2-3 levels	2-5 ✓	
	3. RISC and related ideas	2-3 ✓	
	4. Multiple instruction issue (superscalar)	2-3 ✓	
	5. ISA extensions (e.g., for multimedia)	1-3 ✓	
Newer methods	6. Multithreading (super-, hyper-)	2-5 ?	} Covered in Part VII
	7. Speculation and value prediction	2-3 ?	
	8. Hardware acceleration [e.g., GPU]	2-10 ?	
	9. Vector and array processing	2-10 ?	
	10. Parallel/distributed computing	2-1000s ?	

Shares of Technology and Architecture in Processor Performance Improvement

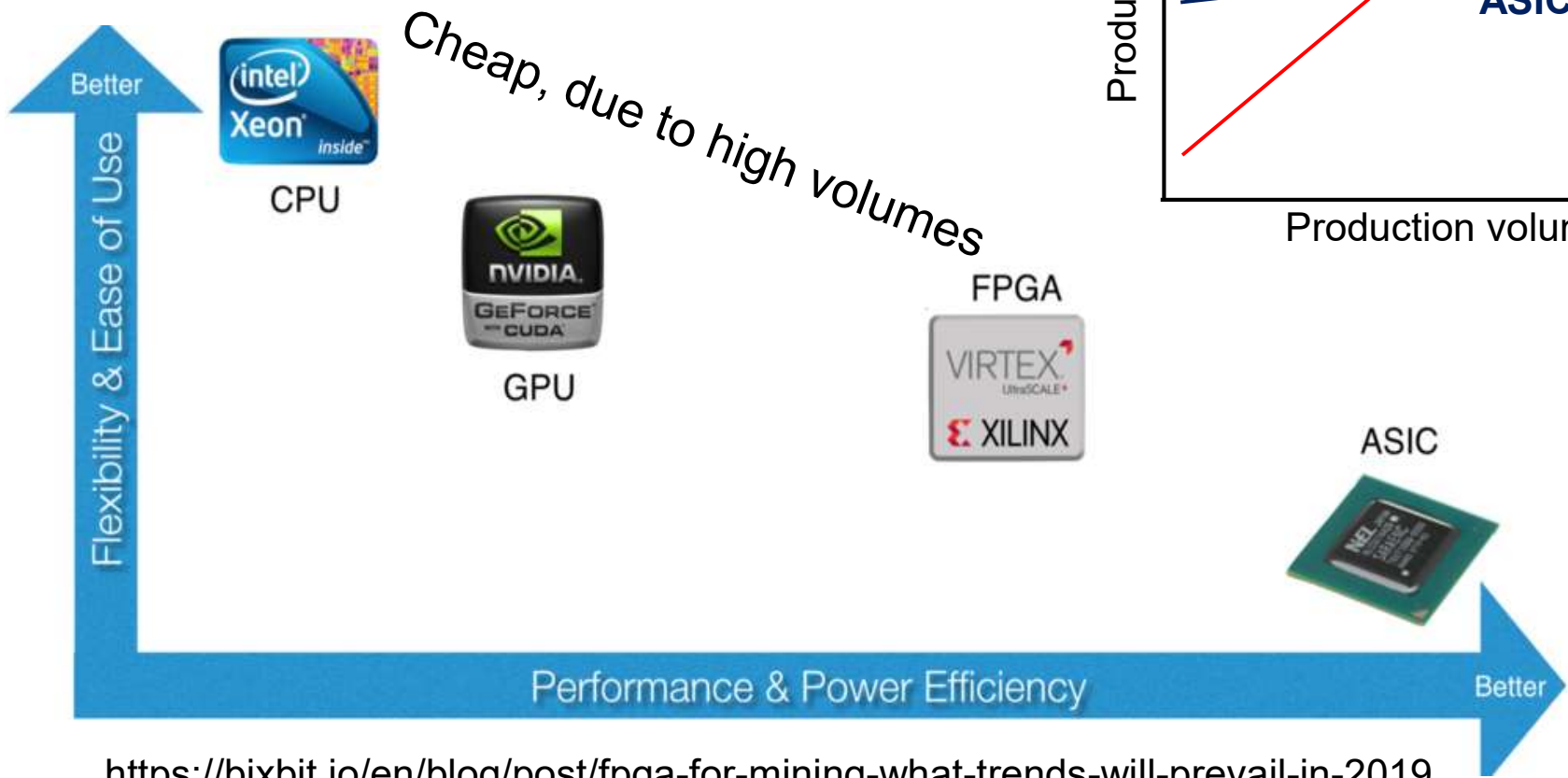
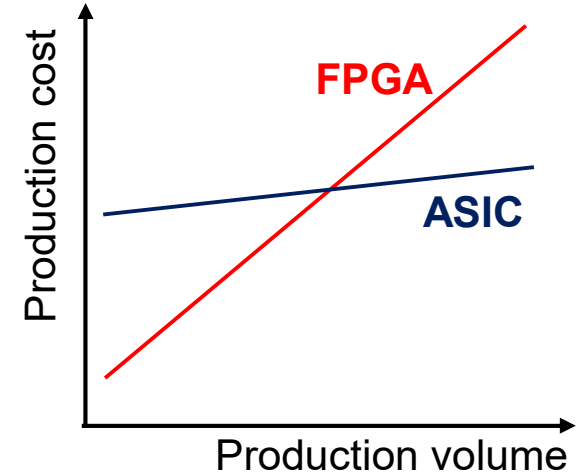


Much of arch. improvements already achieved

Source: "CPU DB: Recording Microprocessor History," CACM, April 2012.

2010s: Specialization

Specialization entails high initial cost and creates a danger of obsolescence



<https://bixbit.io/en/blog/post/fpga-for-mining-what-trends-will-prevail-in-2019>

Hennessy/Patterson's Turing Lecture

Recipients of 2017 ACM Turing Award heralded in their 2018 joint lecture **“A New Golden Age of Computer Architecture”** in which domain-specific hardware dominates

Further progress in “conventional” architectures impeded by slowing Moore scaling, end of Dennard scaling (power wall), diminishing return at high energy cost for ILP, multi-core, etc., sorry state of security (software-only solutions not working)

- Domain-specific languages along with domain-specific arch's
- Agile hardware-development methodologies (for, otherwise, developing many domain-specific systems would be costly)
- Free open architectures and open-source implementations

Shades of Domain-Specificity

Domain-specificity is achieved via ASIC design

ASIC types {
Full-custom
Semi-custom
Programmable

Nothing new: ASICs have been around since the 1970s

**Controller chips for DVD players,
automotive electronics, phones**

Generality lowers per-unit cost but hurts performance

ASIC's popularity: Standards & high-volume products
Higher volume → We can afford to make it more specific

Google's Tensor-Processing Unit

AI-accelerator ASIC aimed at machine-learning applications
 Developed in 2015 and released for third-party use in 2018
 Pixel Neural Core announced in 2019 for Pixel 4 phone

Integer FLP →

Table from Wikipedia

Google TPU	TPUv1	TPUv2	TPUv3	TPUv4 ^[11]	Edge v1
Date Introduced	2016	2017	2018	2020	2018
Process Node	28 nm	20 nm?	12 nm?	?	
Die Size (mm ²)	331	?	?	?	
On chip memory (MiB)	28	?	?	?	
Clock Speed (MHz)	700	?	?	?	
Memory (GB)	8GB DDR3	16GB HBM	32GB HBM	?	
TDP(W)	40	200	250	?	2
TOPS	23	45	90	?	4

NVIDIA: Tensor Core

→ 2x

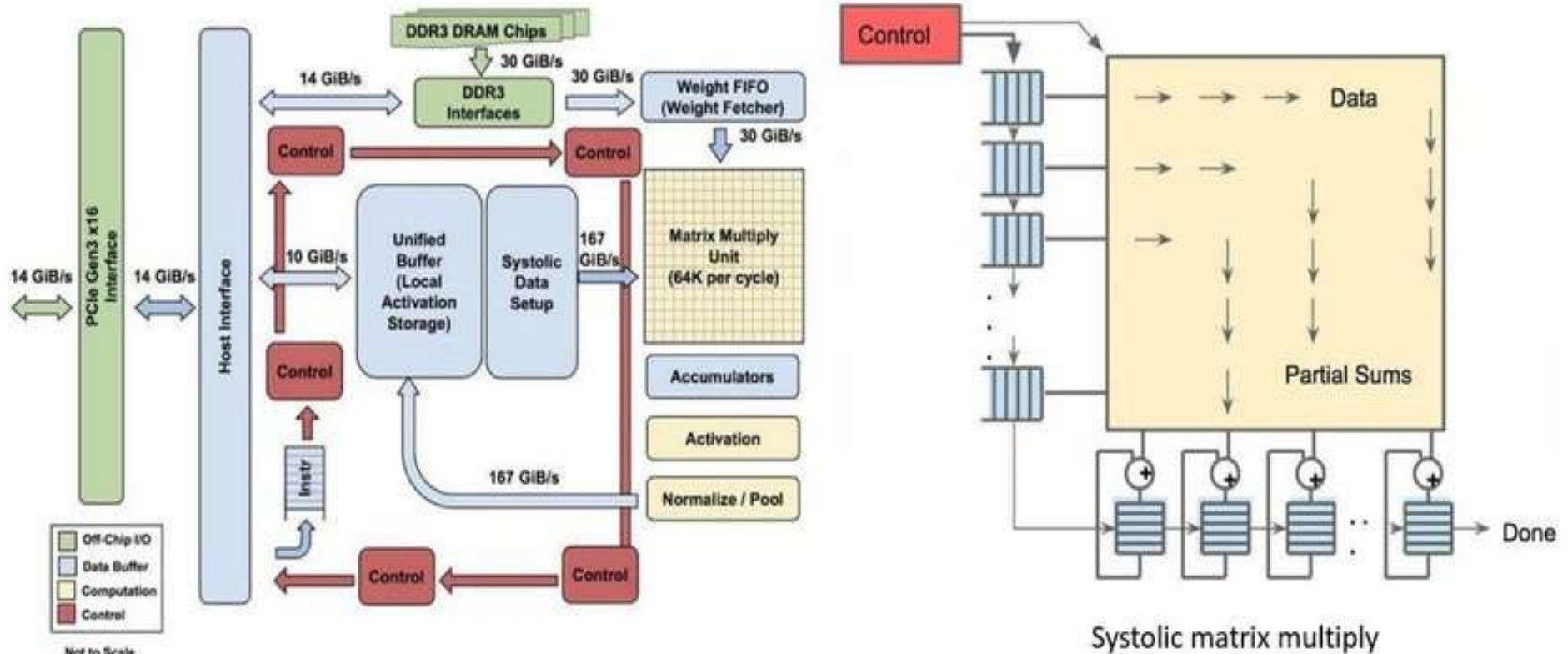
→ 2x

→ 2x?

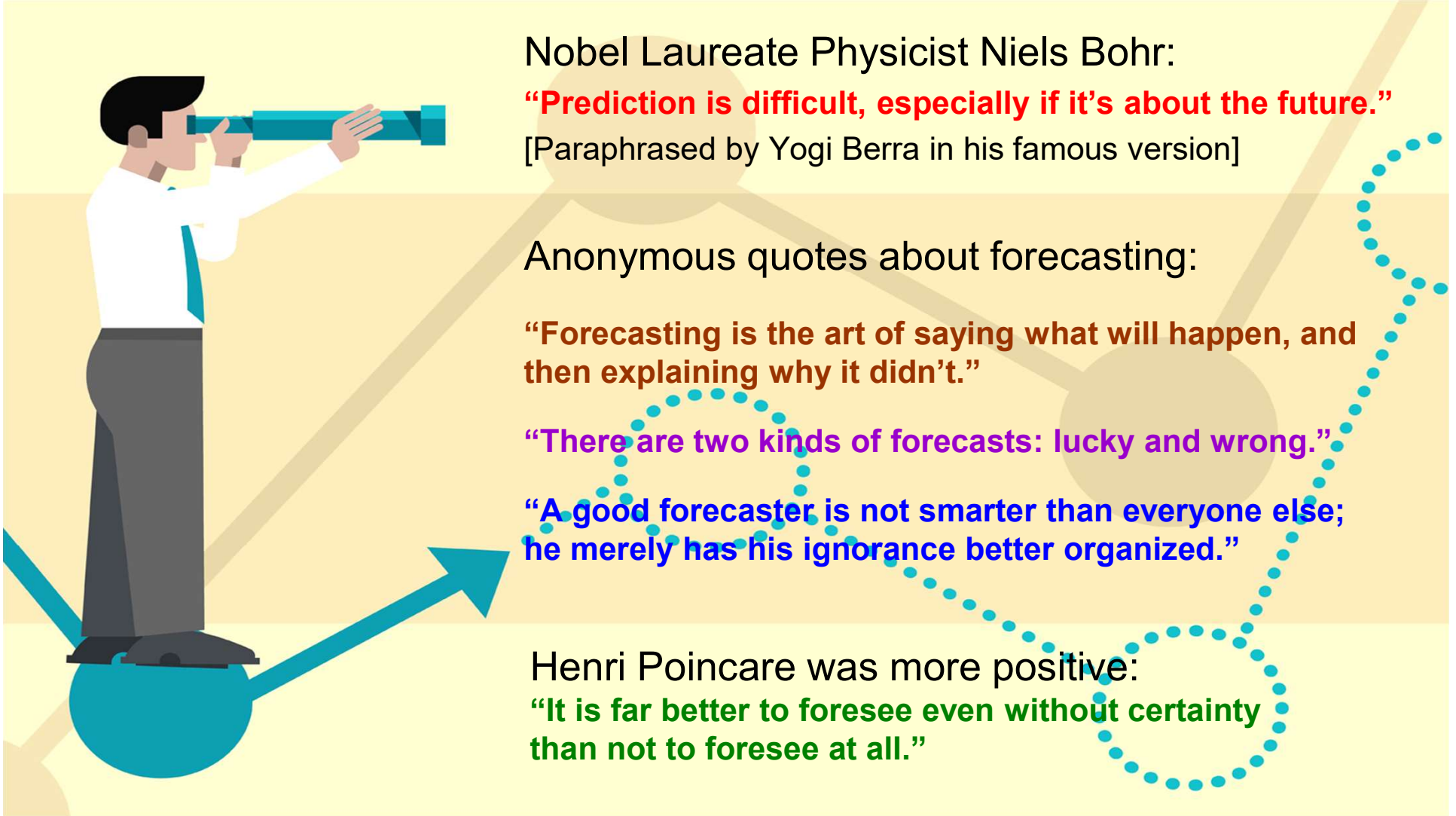
Tensor-Processing Unit Architecture

Optimized for matrix multiplication, a key computation in ML
Faster than CPU/GPU, 15-30x; More energy-frugal, 30-80x

Figures from: <https://research.google/pubs/pub46078/>



Perils of Technology Forecasting



Nobel Laureate Physicist Niels Bohr:
“Prediction is difficult, especially if it’s about the future.”
[Paraphrased by Yogi Berra in his famous version]

Anonymous quotes about forecasting:

“Forecasting is the art of saying what will happen, and then explaining why it didn’t.”

“There are two kinds of forecasts: lucky and wrong.”

“A good forecaster is not smarter than everyone else; he merely has his ignorance better organized.”

Henri Poincare was more positive:
“It is far better to foresee even without certainty than not to foresee at all.”

2020s and Beyond: Looking Ahead

Design improvements

- Adaptation and self-optimization (learning)
- Security (hardware-implemented)
- Reliability via redundancy and self-repair
- Logic-in-memory designs (memory wall)
- Mixed analog/digital design style

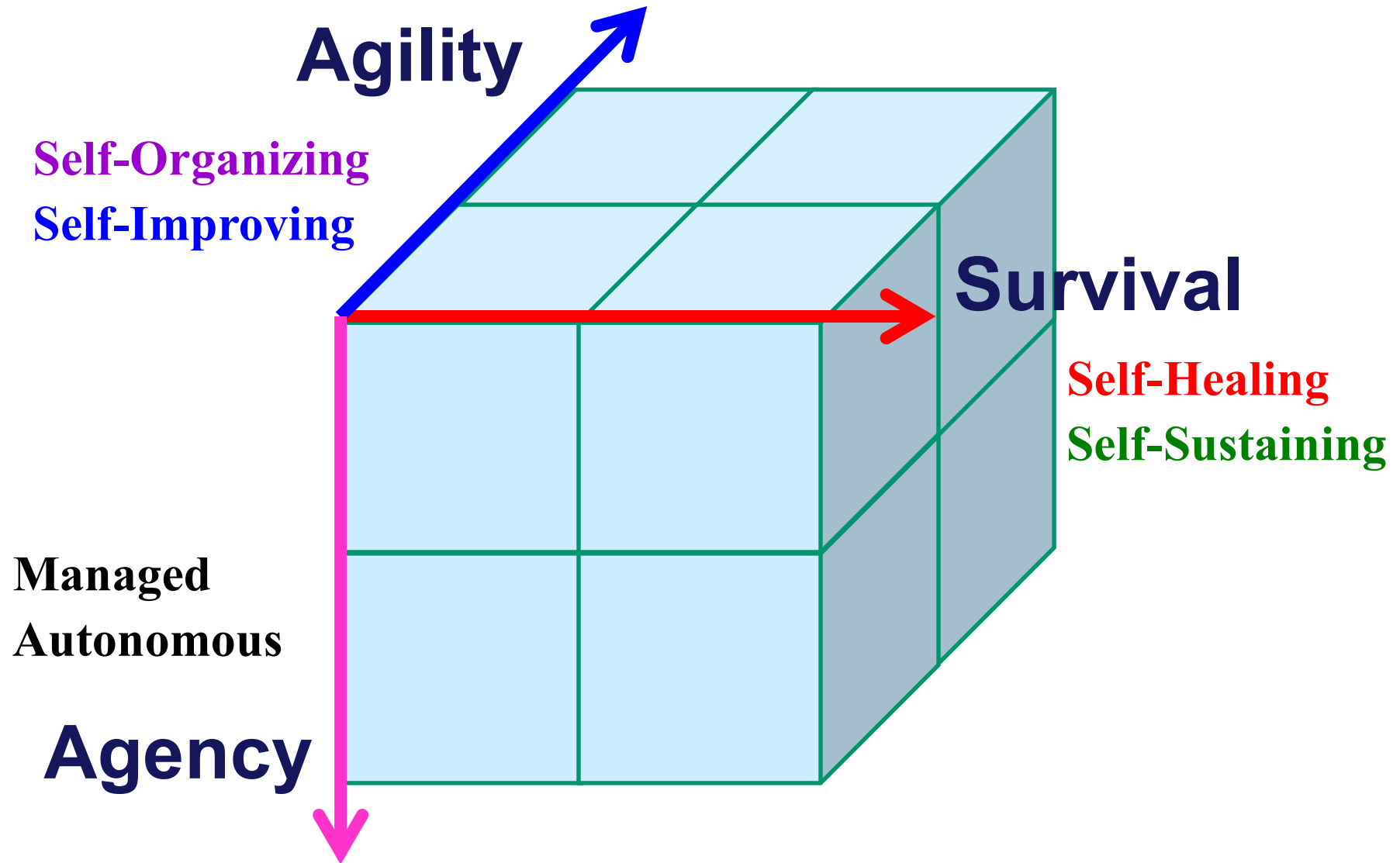


Performance improvements

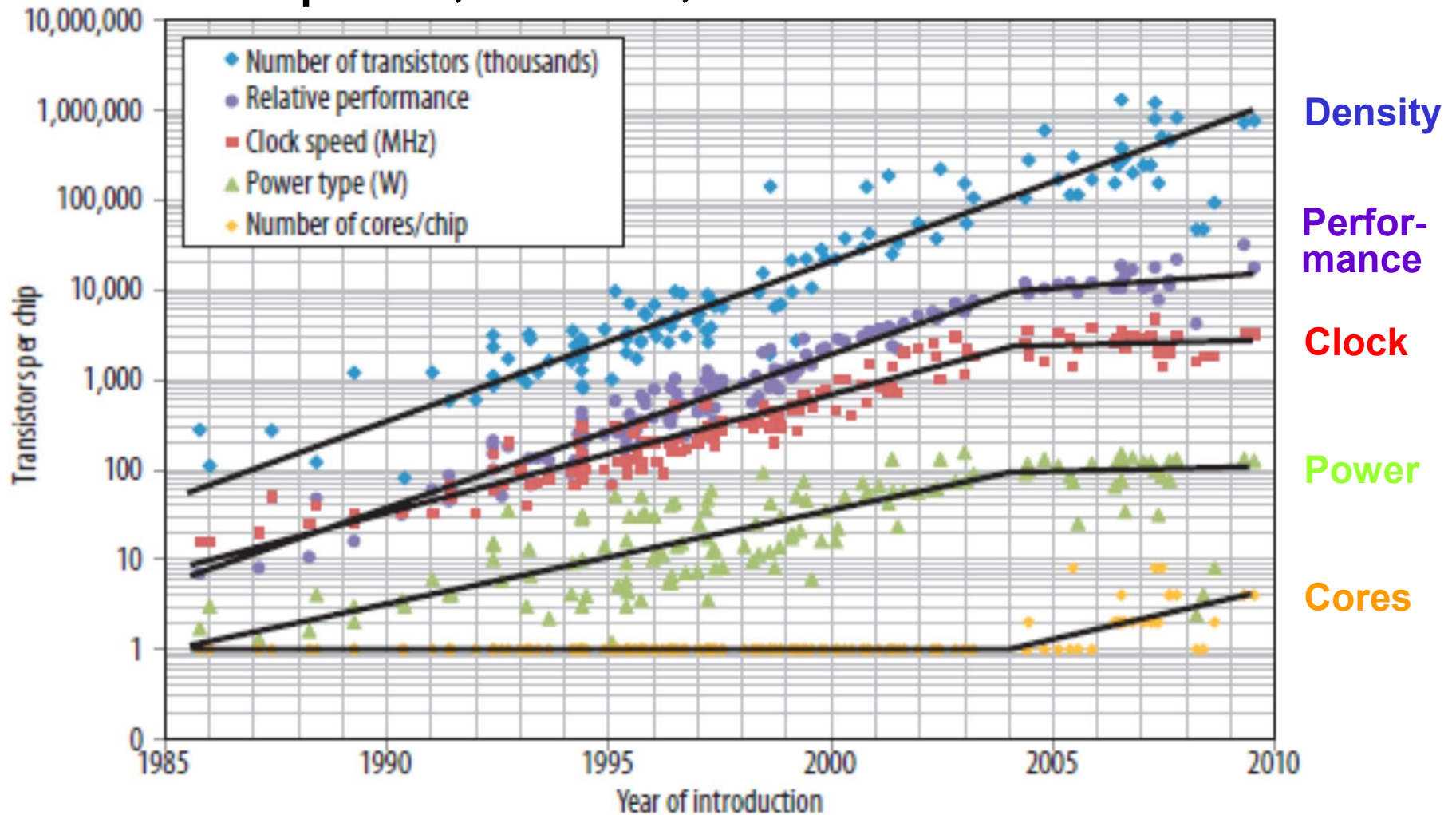
- Revolutionary new technologies
- New computational paradigms
- Brain-inspired and biological computing
- Speculation and value prediction
- Better performance per watt (power wall)



Three Directions for Future Systems



Trends in Processor Chip Density, Performance, Clock Speed, Power, and Number of Cores



NRC Report (2011): The Future of Computing Performance: Game Over or Next Level?

The Quest for Higher Performance

Top-Five Supercomputers in November 2020 (<http://www.top500.org>)

Rank (previous) ↕	Rmax Rpeak (PFLOPS) ↕	Name ↕	Model ↕	CPU cores ↕	Accelerator (e.g. GPU) cores ↕	Interconnect ↕	Manufacturer ↕
1	442.010 537.212	Fugaku	Supercomputer Fugaku	158,976 × 48 A64FX @2.2 GHz	0	Tofu interconnect D	Fujitsu
2 ▼ (1)	148.600 200.795	Summit	IBM Power System AC922	9,216 × 22 POWER9 @3.07 GHz	27,648 × 80 Tesla V100	InfiniBand EDR	IBM
3 ▼ (2)	94.640 125.712	Sierra	IBM Power System S922LC	8,640 × 22 POWER9 @3.1 GHz	17,280 × 80 Tesla V100	InfiniBand EDR	IBM
4 ▼ (3)	93.015 125.436	Sunway TaihuLight	Sunway MPP	40,960 × 260 SW26010 @1.45 GHz	0	Sunway ^[26]	NRCPC
5 ▲ (7)	63.460 79.215	Selene	Nvidia	1,120 × 64 Epyc 7742 @2.25 GHz	4,480 × 108 Ampere A100	Mellanox HDR Infiniband	Nvidia

We Need More than Sheer Performance

Environmentally responsible design

Reusable designs, parts, and material

Power efficiency

Starting publication in 2016: *IEEE Transactions on Sustainable Computing*



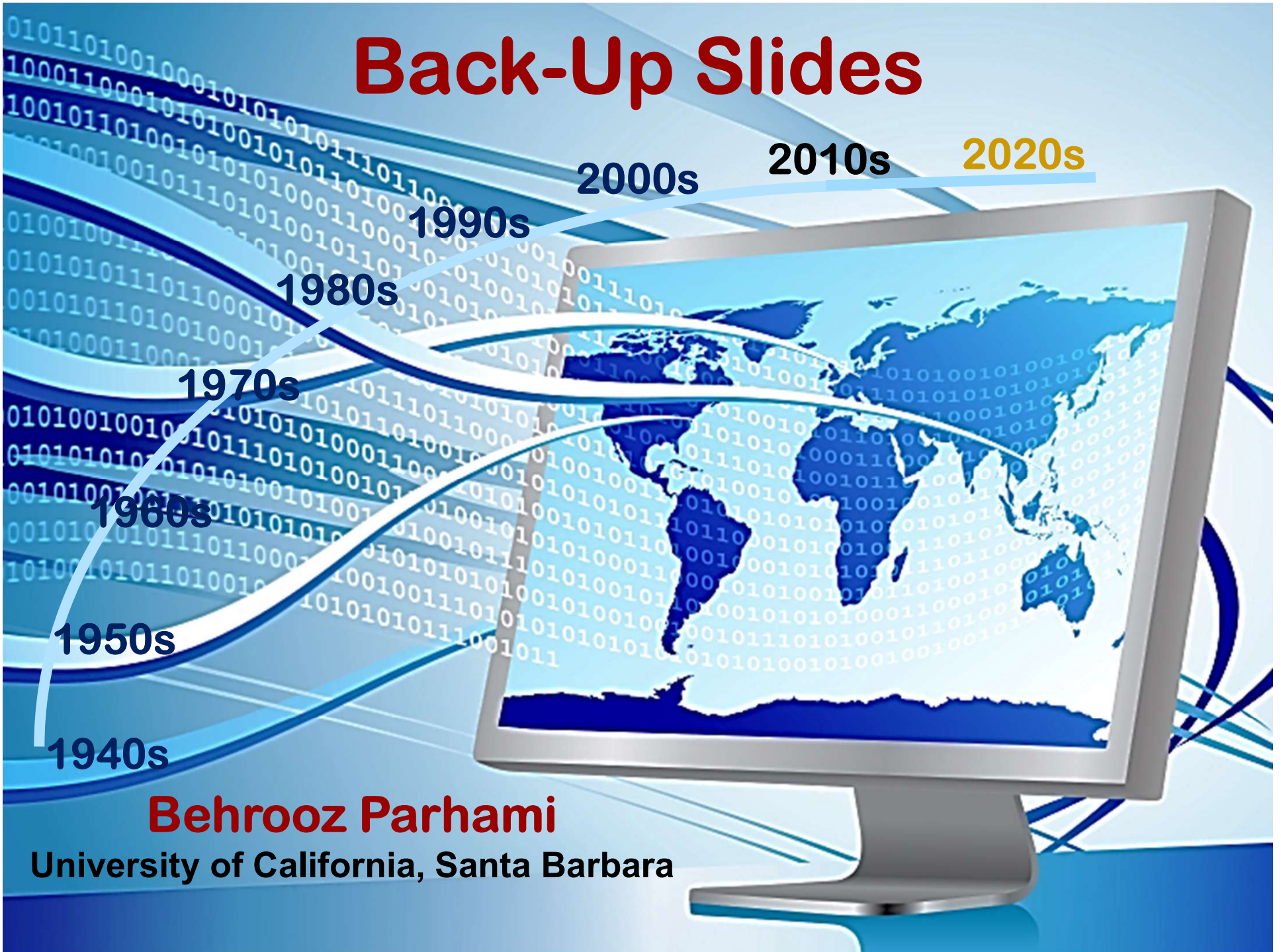
Questions or Comments?

parhami@ece.ucsb.edu

<http://www.ece.ucsb.edu/~parhami/>



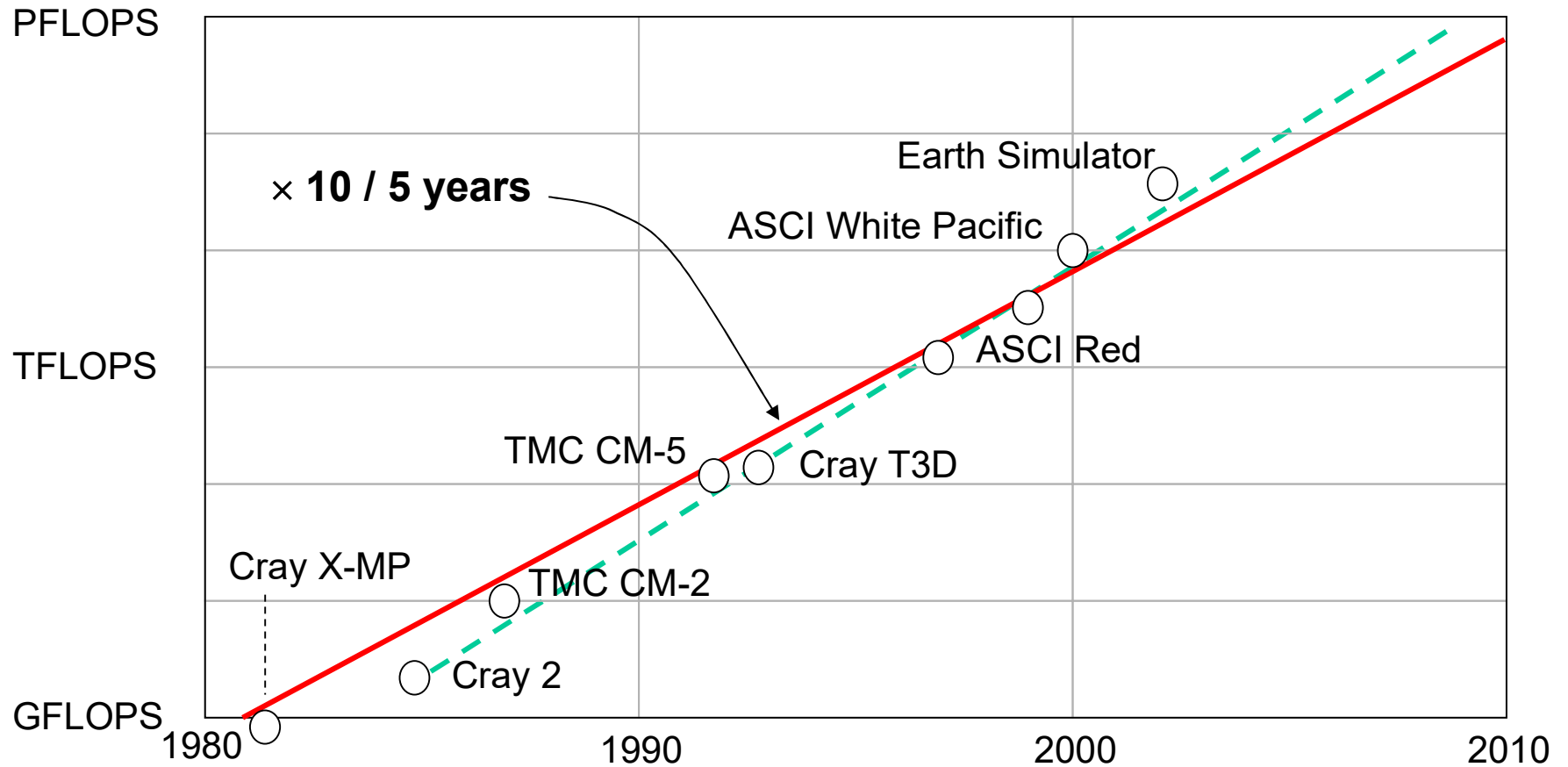
Back-Up Slides



Behrooz Parhami

University of California, Santa Barbara

Peak Performance of Supercomputers

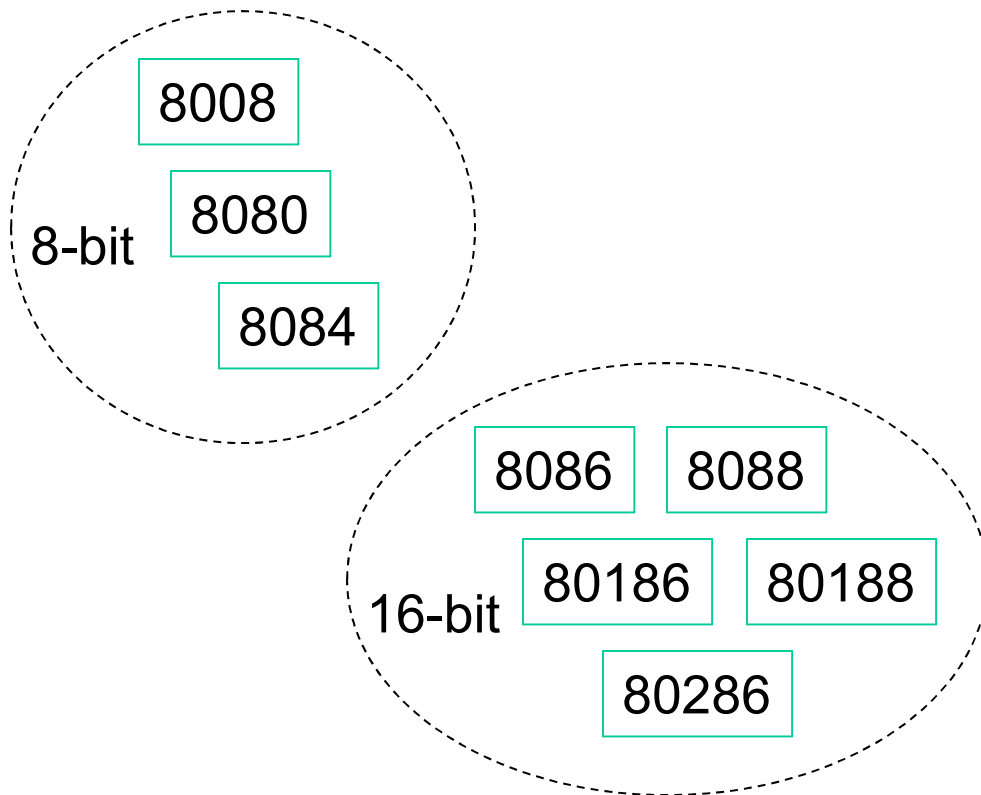


Dongarra, J., "Trends in High Performance Computing,"
Computer J., Vol. 47, No. 4, pp. 399-403, 2004. [Dong04]

Past and Current Performance Trends

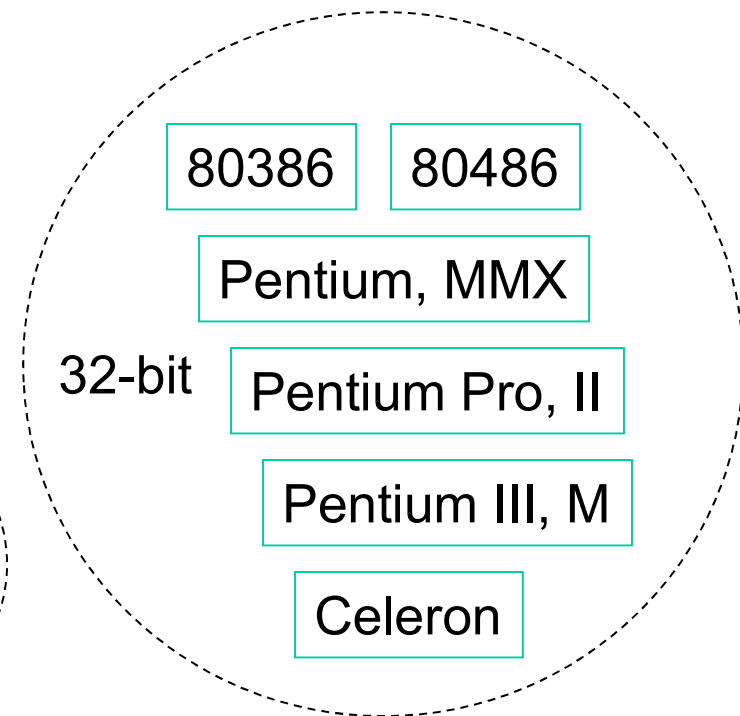
Intel 4004: The first μ p (1971)

0.06 MIPS (4-bit processor)

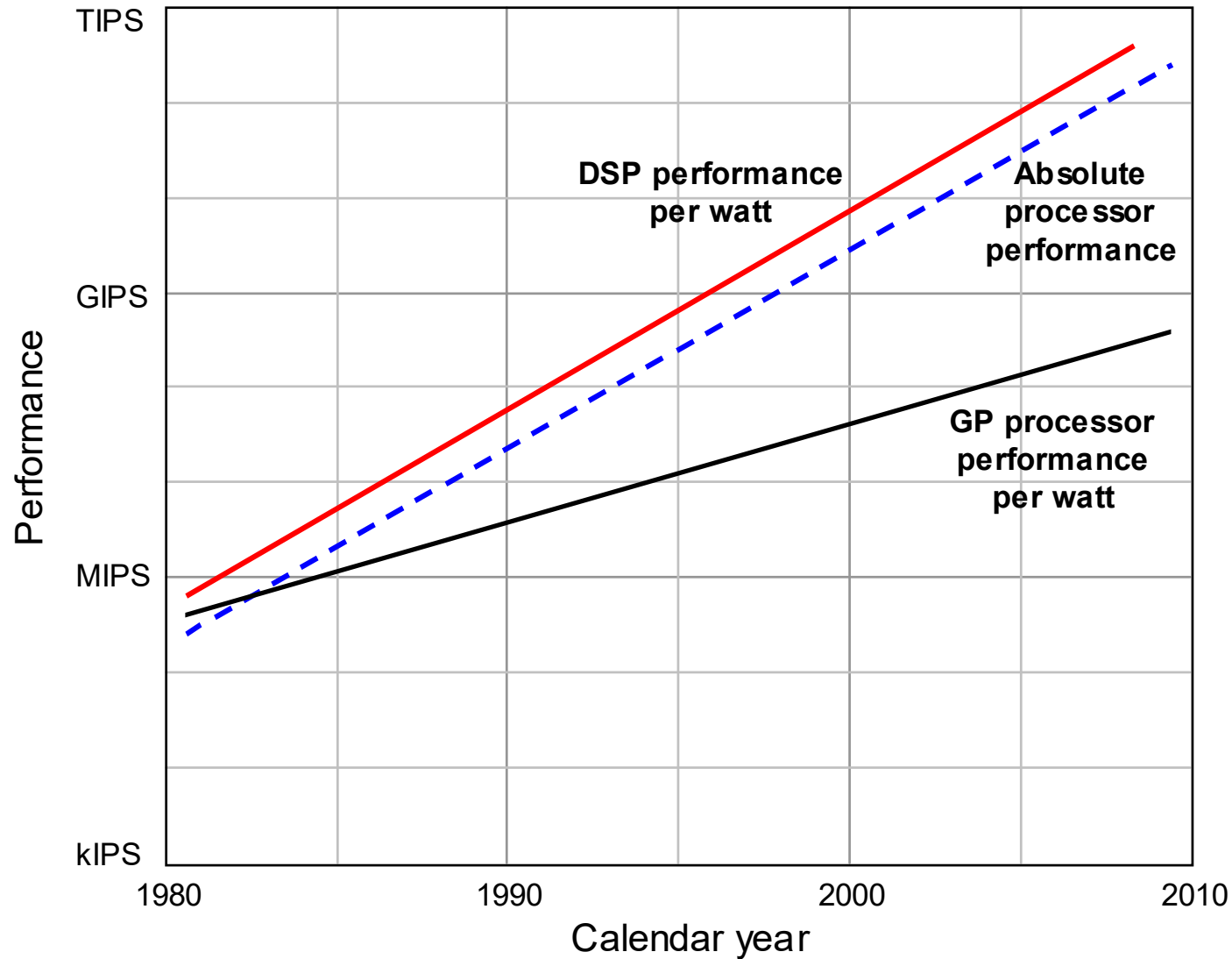


Intel Pentium 4, circa 2005

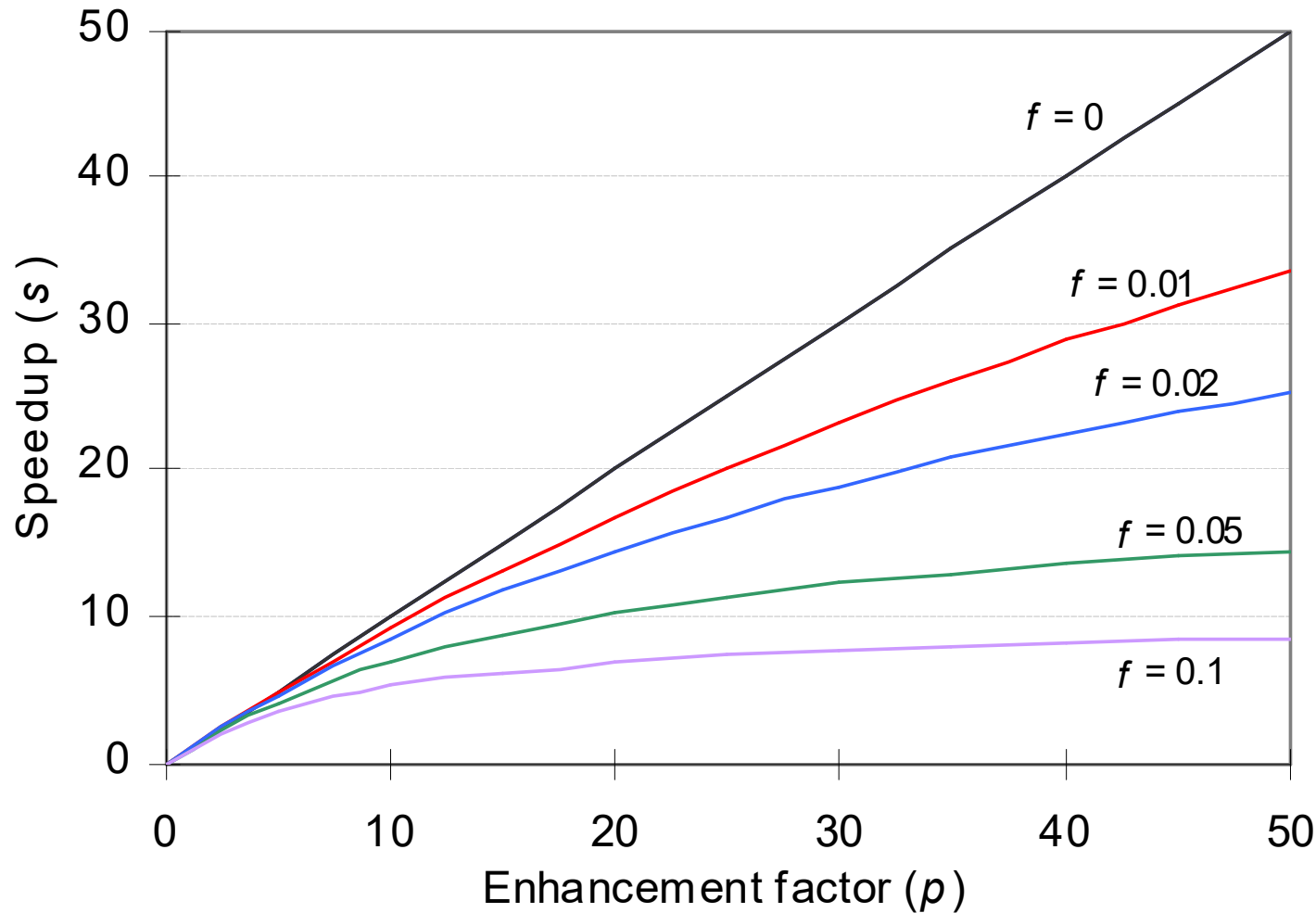
10,000 MIPS (32-bit processor)



Energy Consumption is Getting out of Hand



Amdahl's Law



f = fraction
unaffected

p = speedup
of the rest

$$s = \frac{1}{f + (1-f)/p}$$
$$\leq \min(p, 1/f)$$

Amdahl's System Balance Rules of Thumb

The need for high-capacity, high-throughput secondary (disk) memory

Processor speed	RAM size	Disk I/O rate	Number of disks	Disk capacity	Number of disks
1 GIPS	1 GB	100 MB/s	1	100 GB	1
1 TIPS	1 TB	100 GB/s	1000	100 TB	100
1 PIPS	1 PB	100 TB/s	1 Million	100 PB	100 000
1 EIPS	1 EB	100 PB/s	1 Billion	100 EB	100 Million



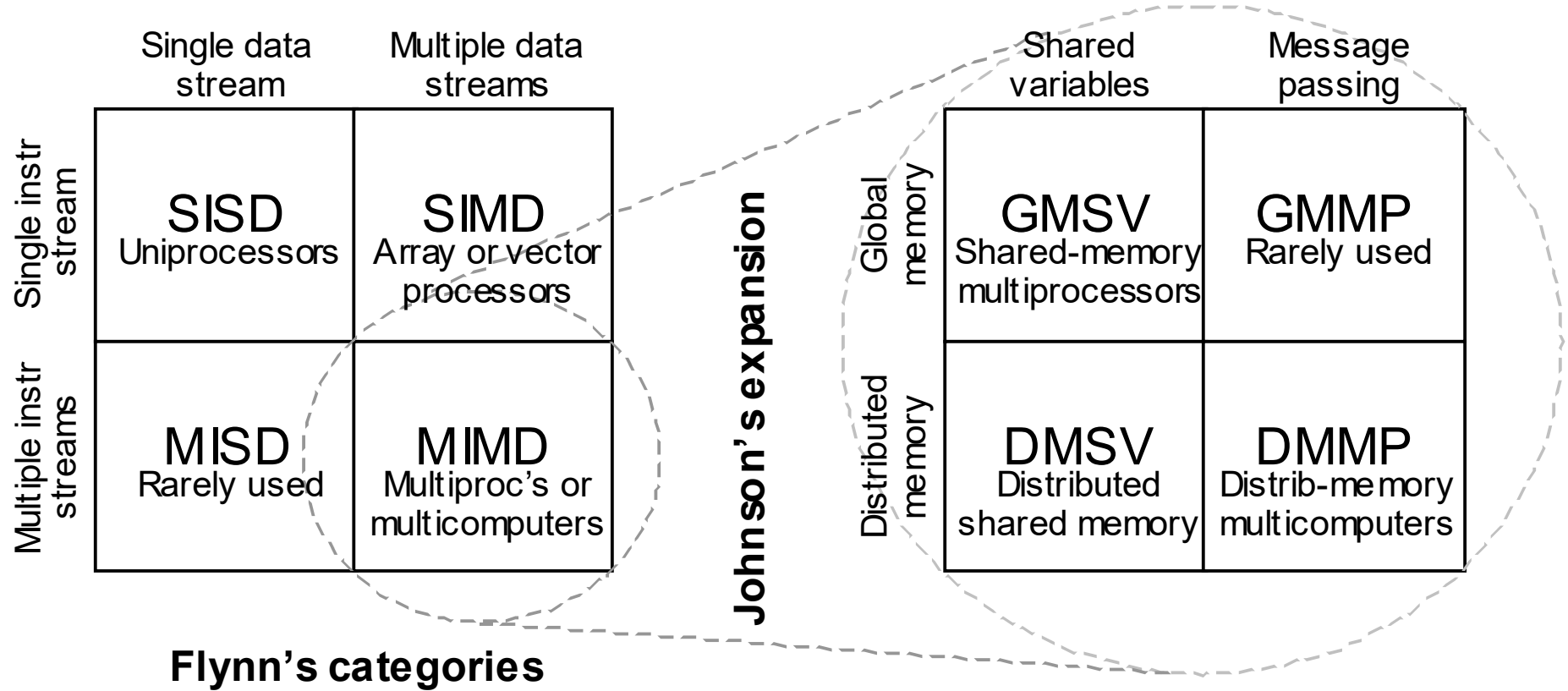
1 RAM byte
for each IPS

1 I/O bit per sec
for each IPS

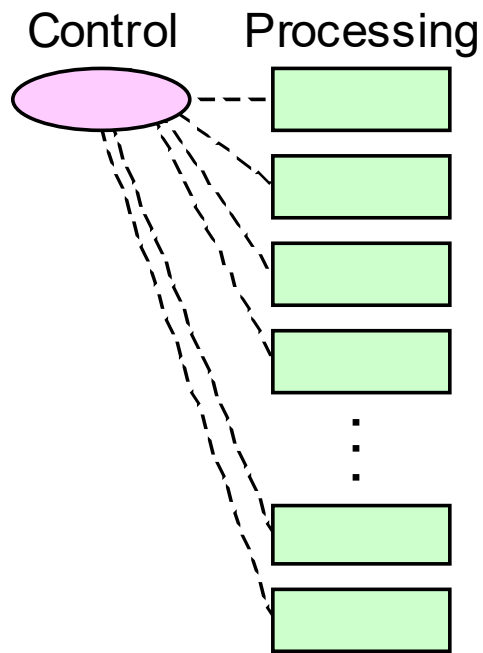
100 disk bytes
for each RAM byte

G Giga
T Tera
P Peta
E Exa

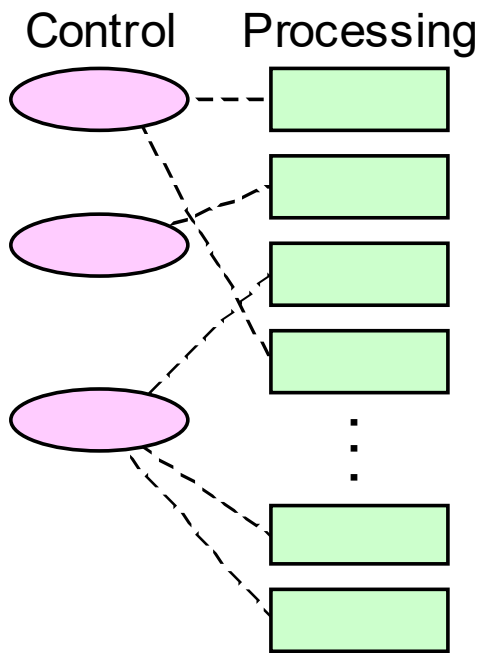
The Flynn/Johnson Classification



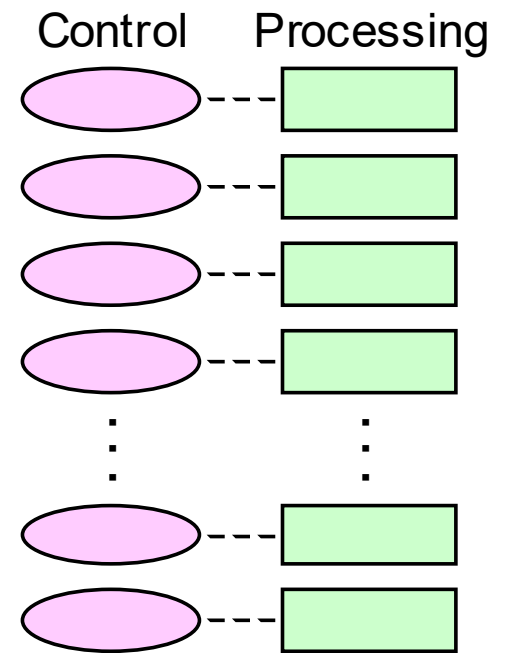
Shared-Control Systems



(a) Shared-control array processor, SIMD



(b) Multiple shared controls, MSIMD



(c) Separate controls, MIMD

From completely shared control to totally separate controls.

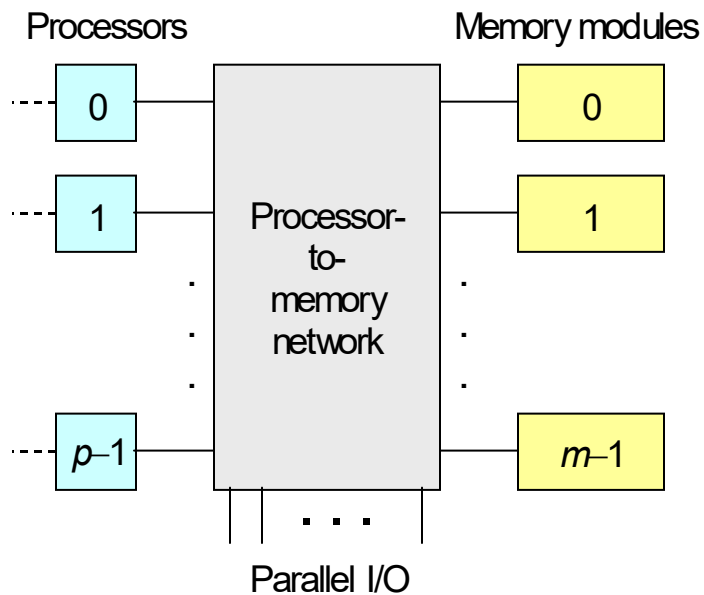
MIMD Architectures

Control parallelism: executing several instruction streams in parallel

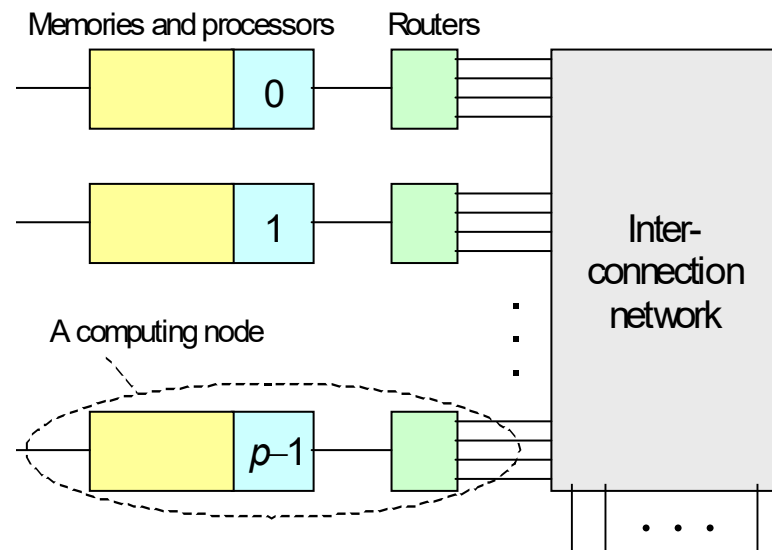
GMSV: Shared global memory – symmetric multiprocessors

DMSV: Shared distributed memory – asymmetric multiprocessors

DMMP: Message passing – multicomputers

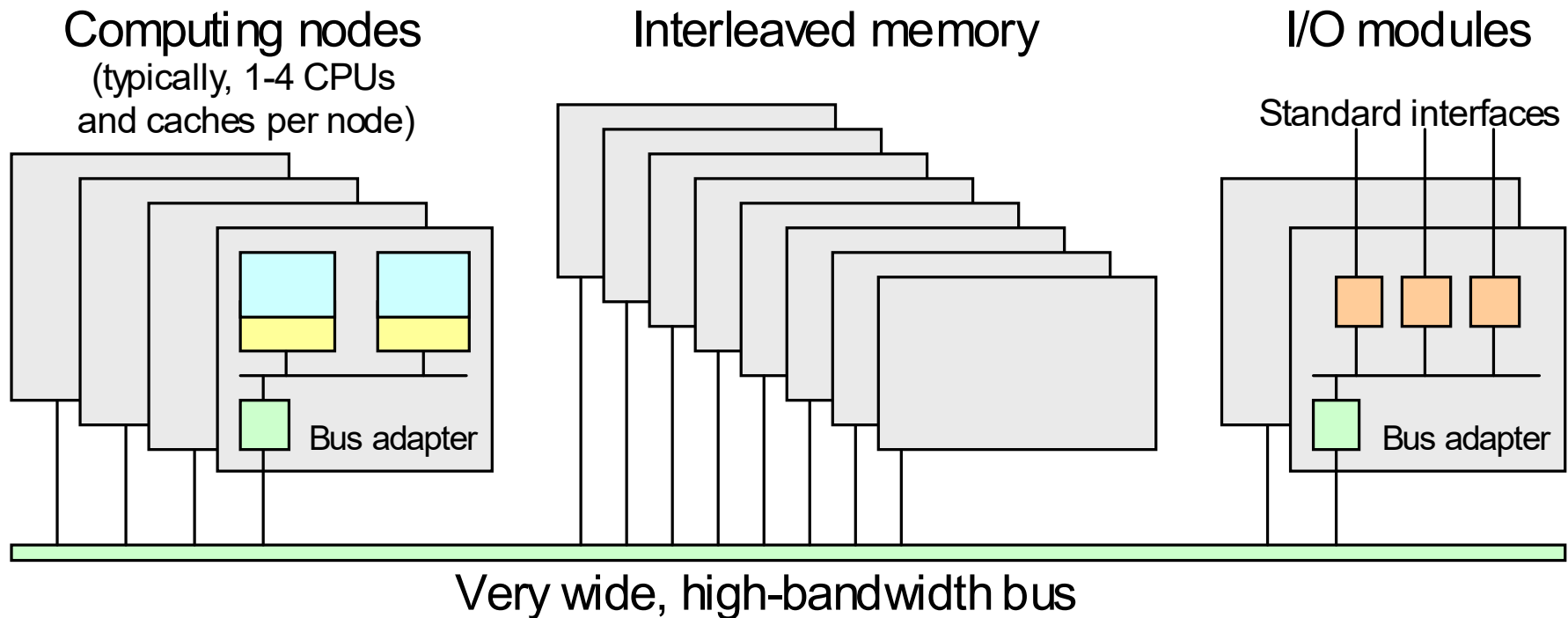


Centralized shared memory



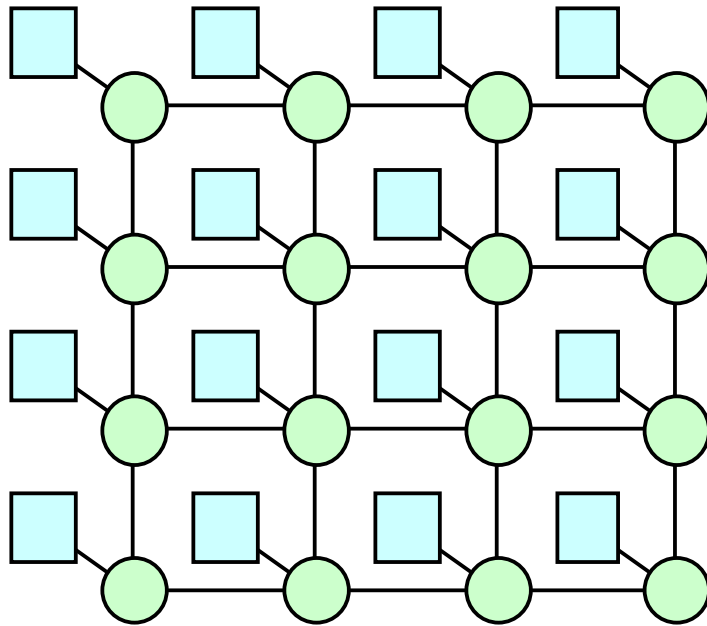
Distributed memory

Implementing Symmetric Multiprocessors

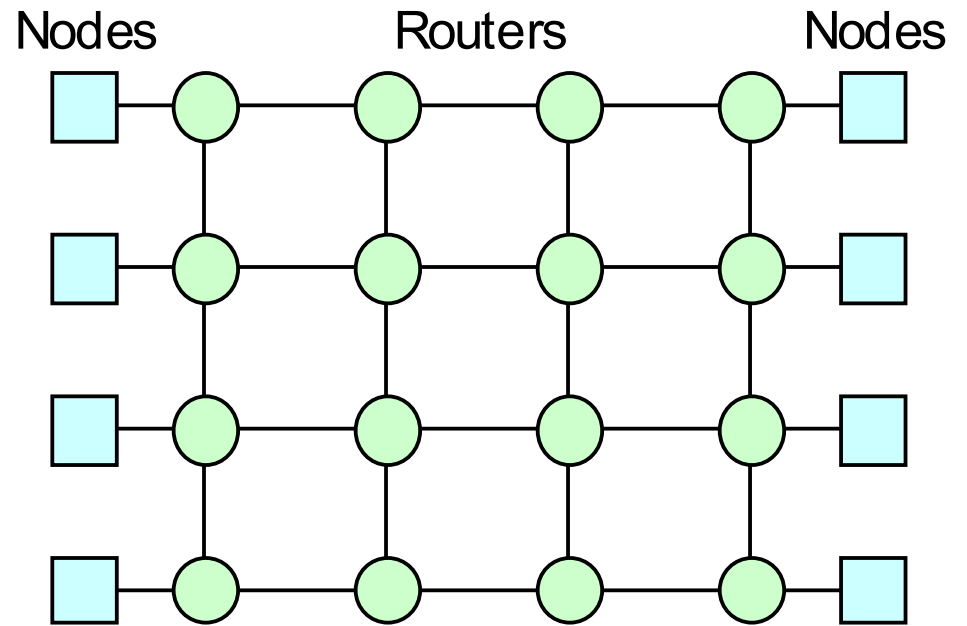


Structure of a generic bus-based symmetric multiprocessor.

Interconnection Networks



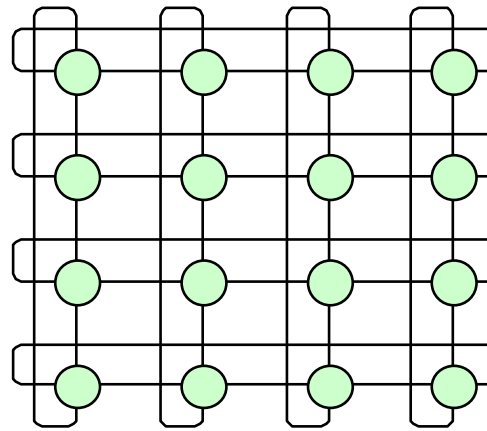
(a) Direct network



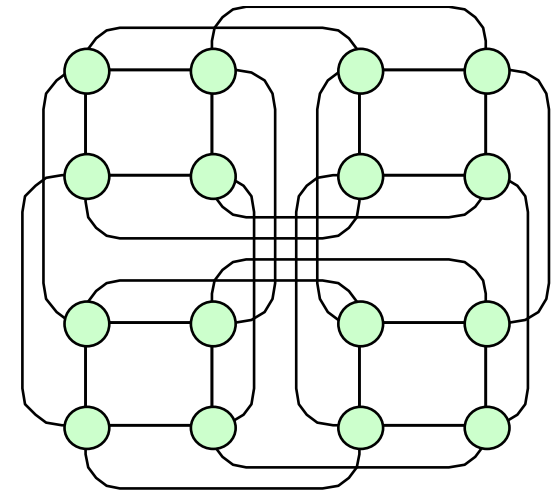
(b) Indirect network

Examples of direct and indirect interconnection networks.

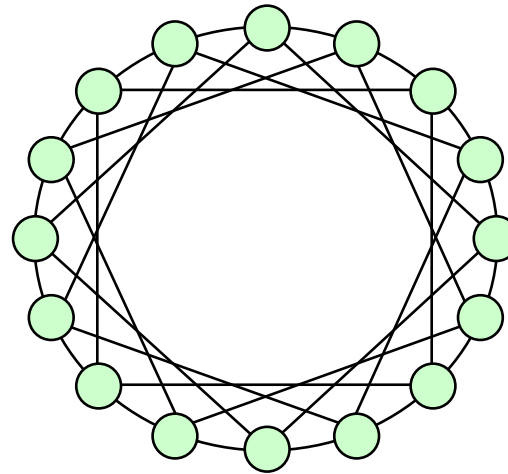
Direct Interconnection Networks



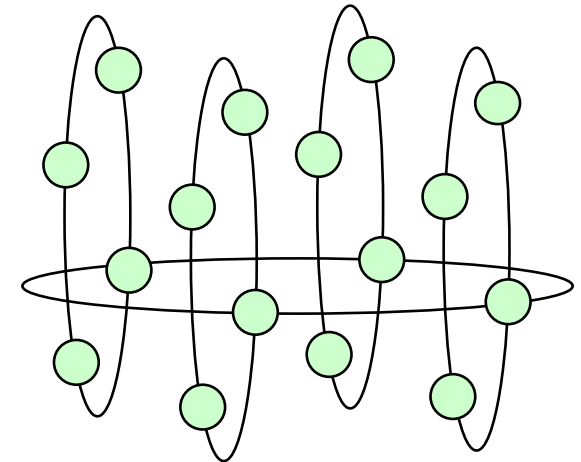
(a) 2D torus



(b) 4D hypercube



(c) Chordal ring



(d) Ring of rings

A sampling of common direct interconnection networks.
Only routers are shown; a computing node is implicit for each router.

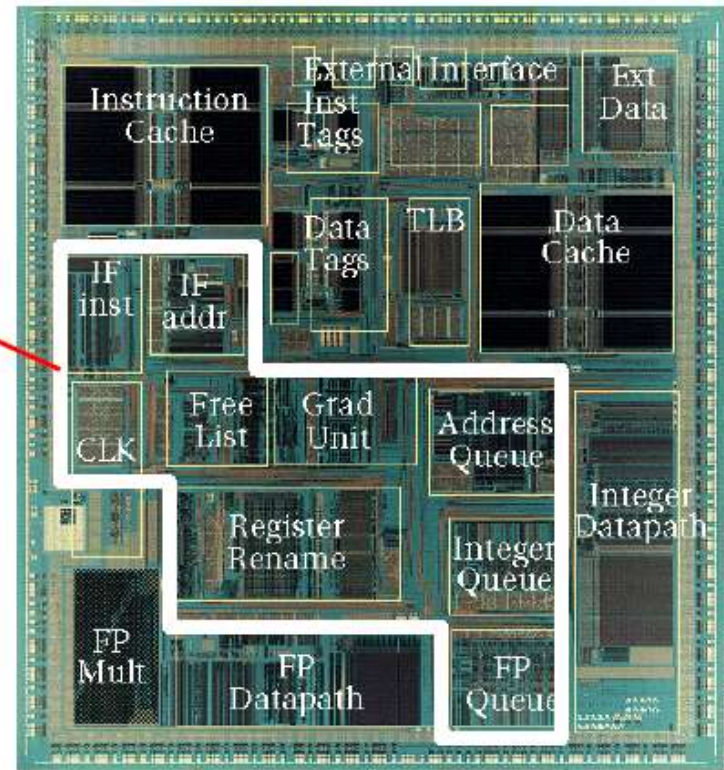
Design Space for Superscalar Pipelines

Front end: In-order or out-of-order
Instr. issue: In-order or out-of-order
Writeback: In-order or out-of-order
Commit: In-order or out-of-order

The more OoO stages,
the higher the complexity

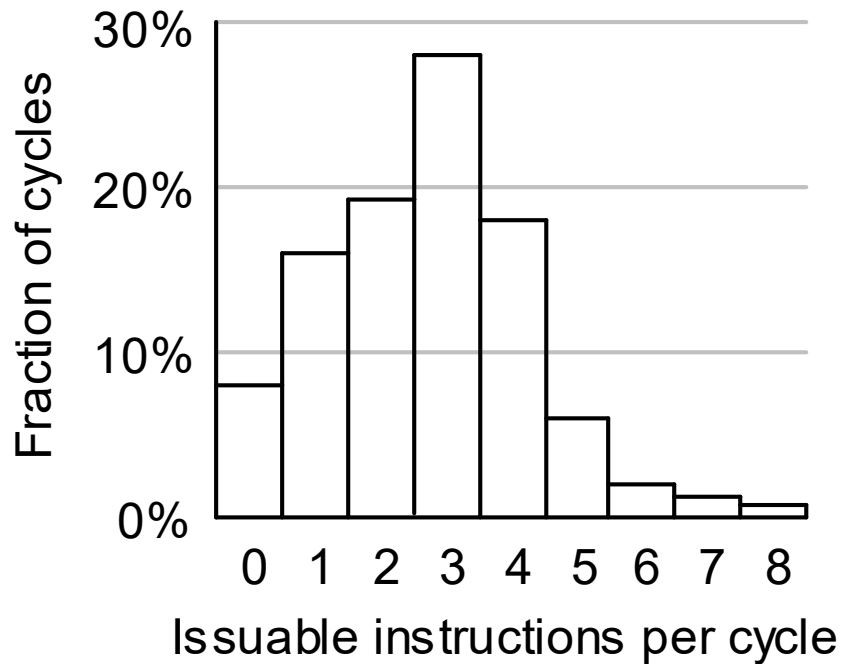
Example of complexity due to
out-of-order processing:
MIPS R10000

Control
Logic

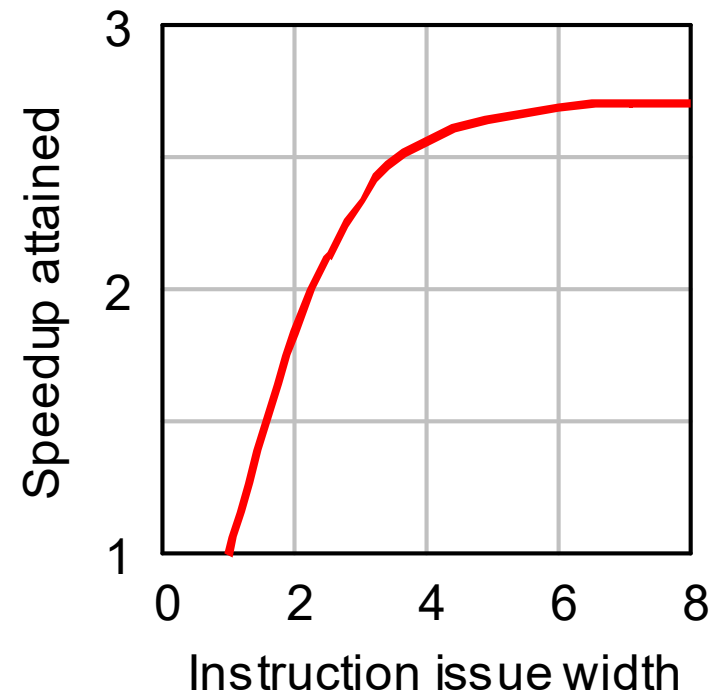


Source: Ahi, A. et al., "MIPS R10000
Superscalar Microprocessor,"
Proc. Hot Chips Conf., 1995.

Instruction-Level Parallelism



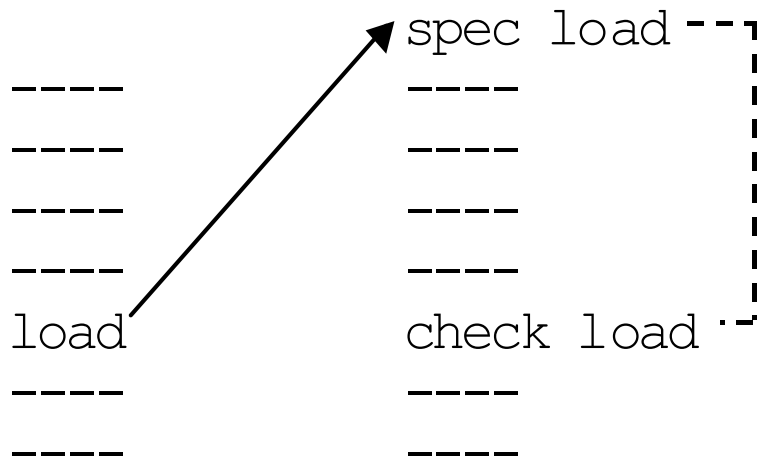
(a)



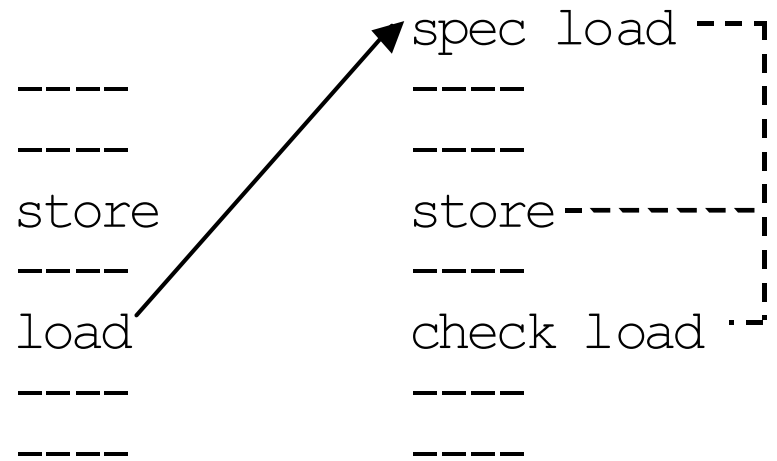
(b)

Available instruction-level parallelism and the speedup due to multiple instruction issue in superscalar processors [John91].

Speculative Loads



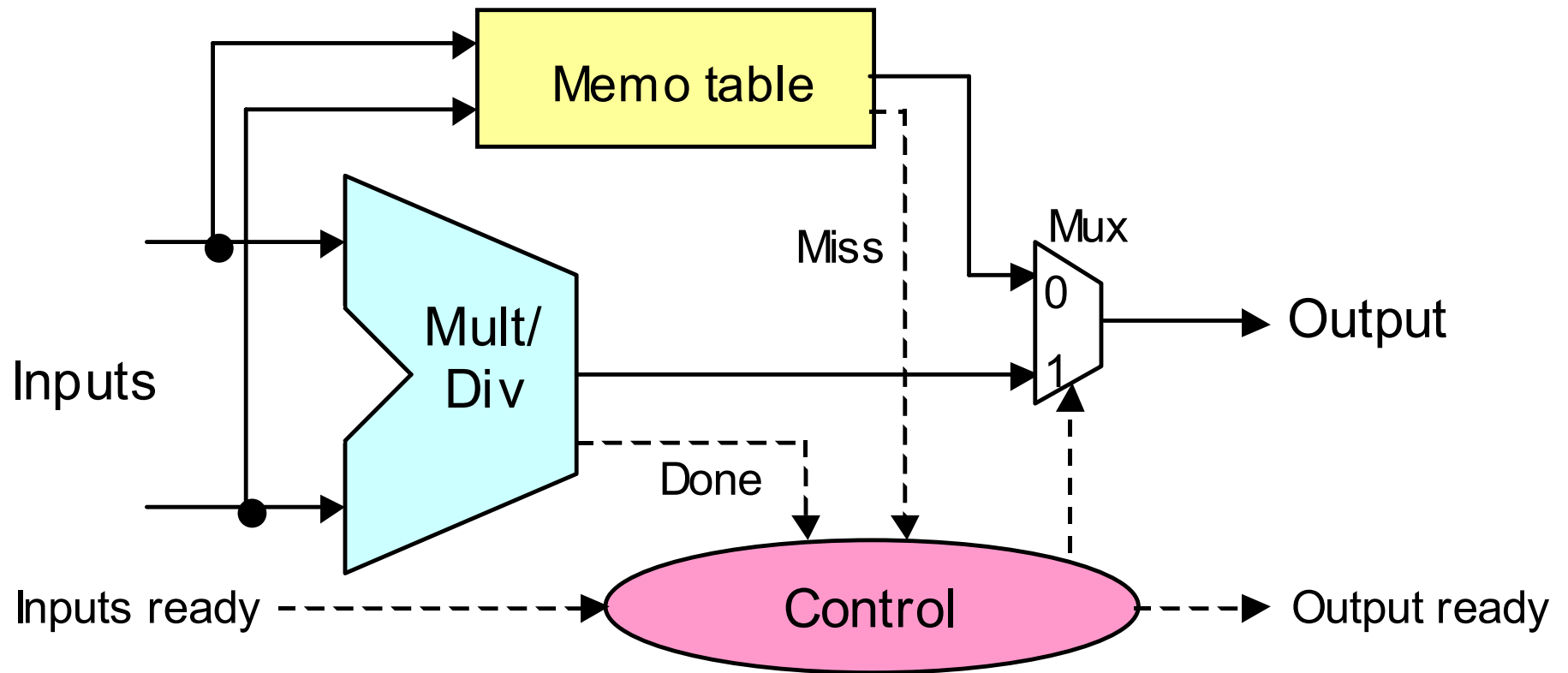
(a) Control speculation



(b) Data speculation

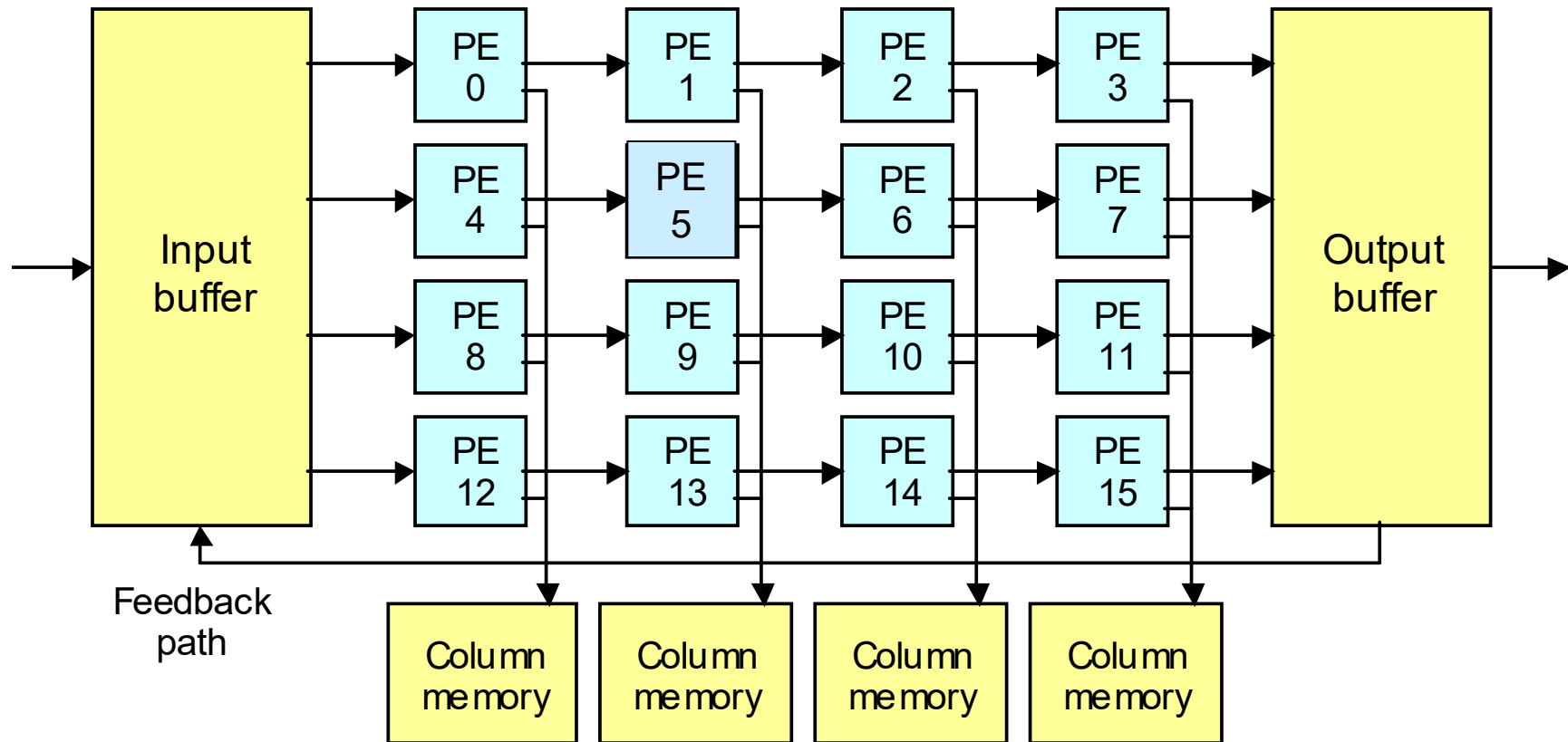
Examples of software speculation in IA-64.

Value Prediction



Value prediction for multiplication or division via a memo table.

Graphic Processors, Network Processors, ...



Simplified block diagram of Toaster2,
Cisco Systems' network processor.

Computing in the Cloud

Computational resources, both hardware and software, are provided by, and managed within, the cloud

Users pay a fee for access

Managing / upgrading is much more efficient in large, centralized facilities (warehouse-sized data centers or server farms)



Image from Wikipedia

This is a natural continuation of the outsourcing trend for special services, so that companies can focus their energies on their main business