Introduction to Parallel Processing

Lecture 0 : What is Parallel Processing?

Professor Amanda Bienz

What is Parallel Processing?

- Main idea : Execute multiple instructions at the same time

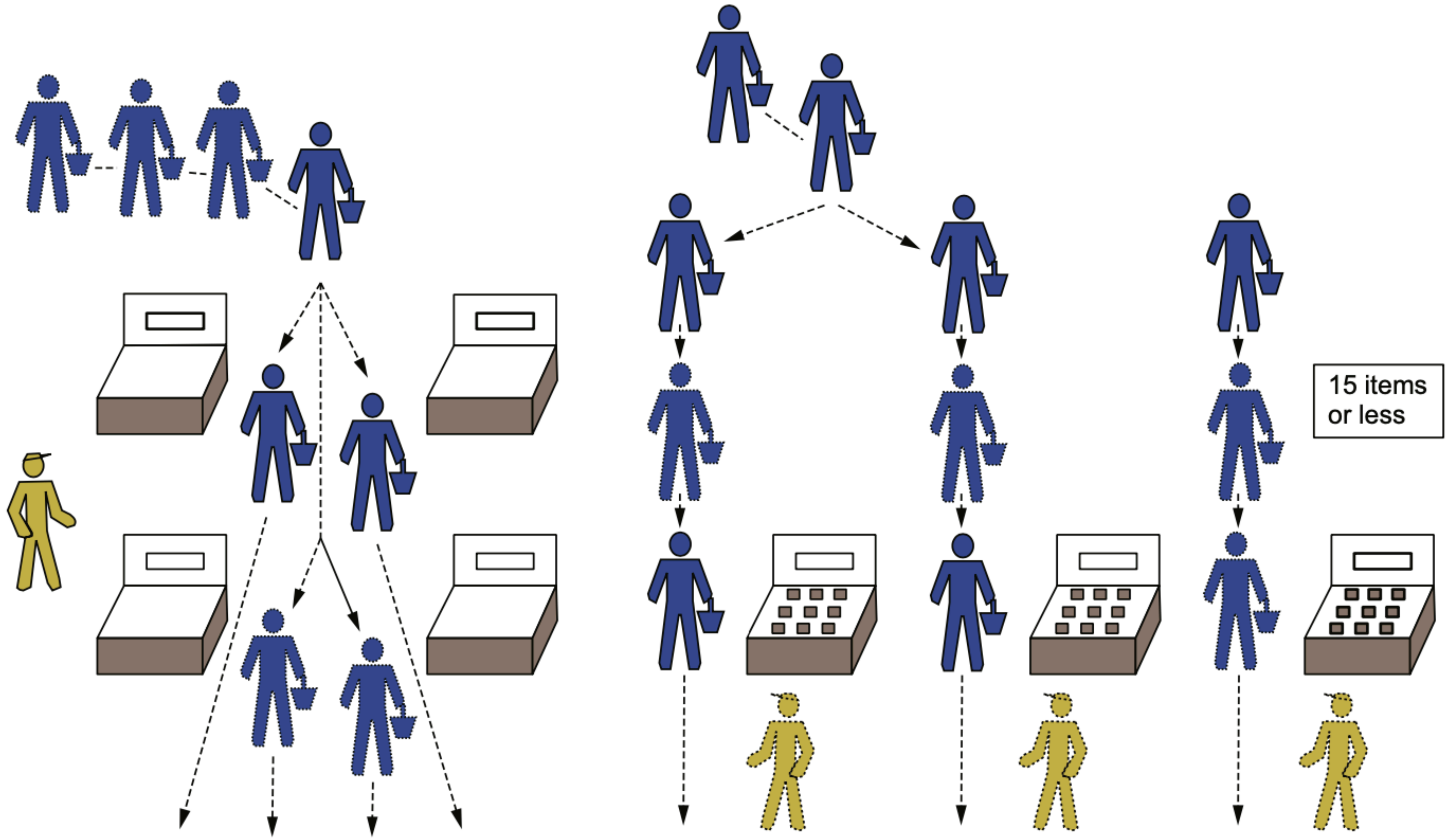
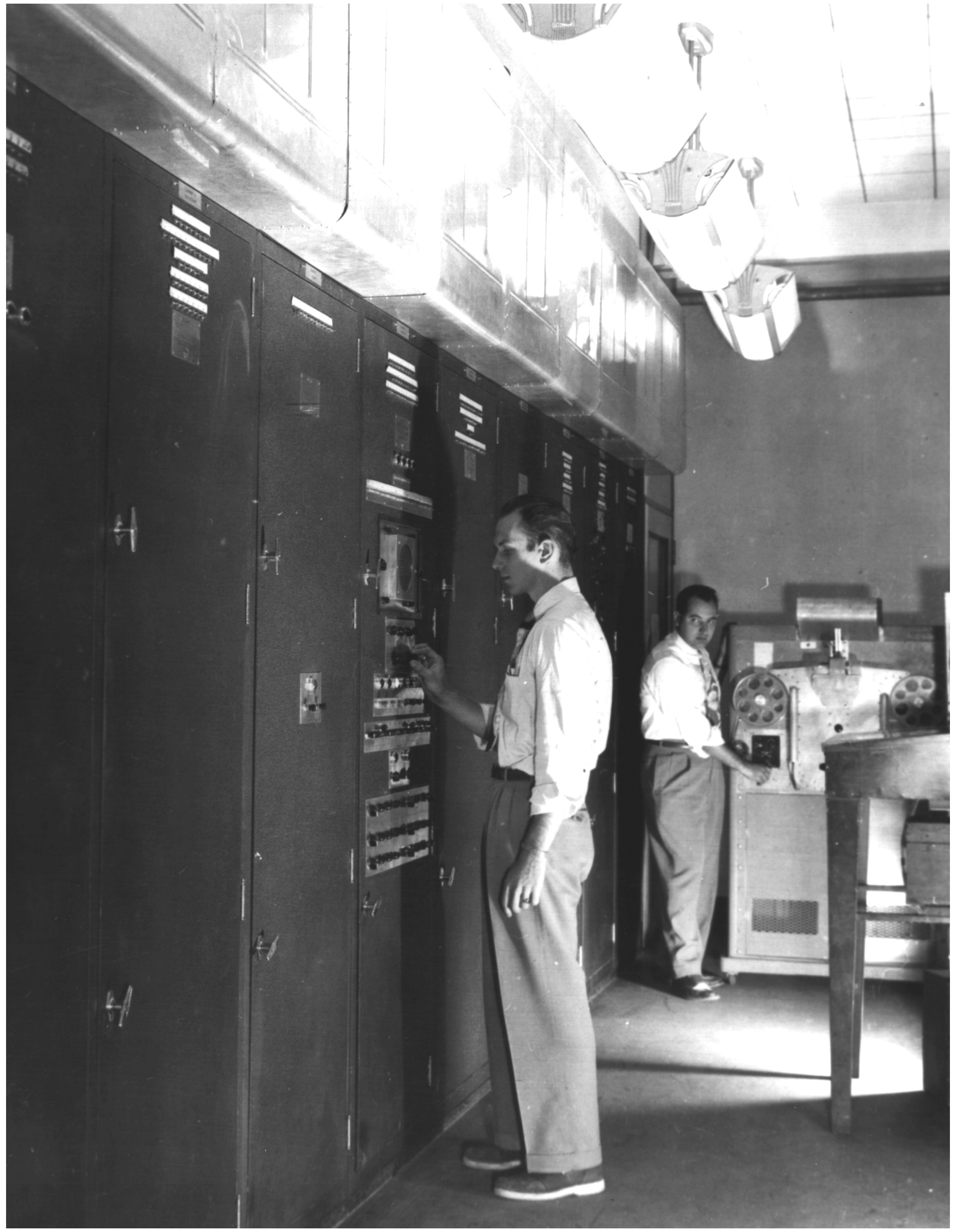


Figure 1.1 Everyday parallelism in supermarket checkout queues. The checkout cashiers (with caps) process their queue of customers (with baskets). On the left, one cashier processes four self-checkout lanes simultaneously. On the right, one cashier is required for each checkout lane. Each option impacts the supermarket's costs and checkout rates.

First Binary Computer



- EDVAC : 1949 at US Army Ballistic Research Lab
- First binary computer
- Memory capacity : 1,000 34-bit words
- Addition took 864 microseconds
- Multiplication took 2,900 microseconds
- Cost nearly $500,000

Could multiply 345 sets of numbers in 1 second

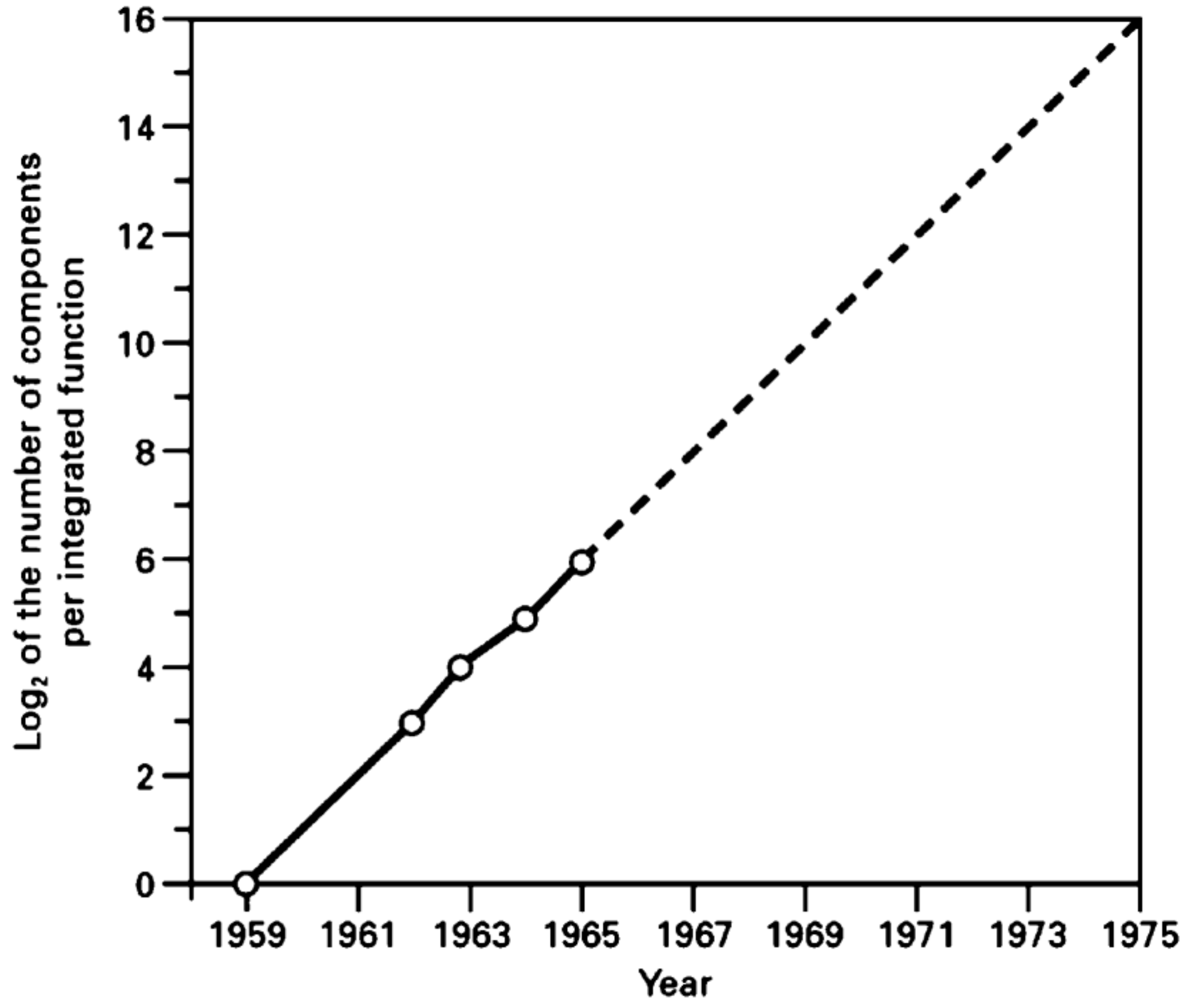
https://apps.dtic.mil/dtic/tr/fulltext/u2/694600.pdf

Advances in Computers

- Processors have obviously gotten faster over the years
- Can now perform trillions of calculations per second
- Which brings us to Moore's Law…

Moore's Law

- 1965 : Gorden Moore, co-founder of Intel, looked at recent advances in complexity of computer chips
- Predicted number of components (transistors) in a chip would double every two years for 10 years
- Was skeptical that so many components could be used
- **This was a business prediction : Number of transistors per chip that yields minimum cost per transistor**

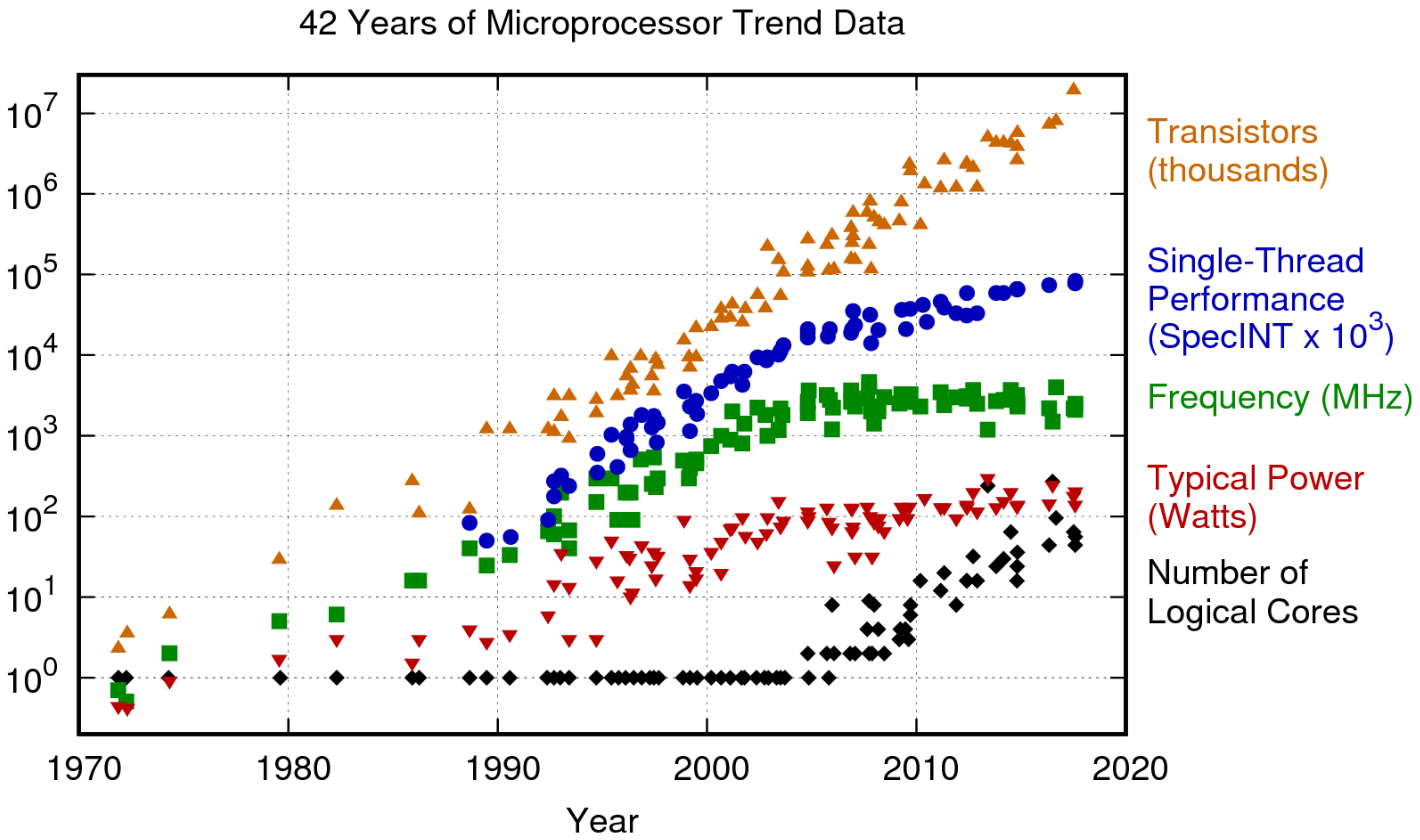


Golio, M. "Fifty Years of Moore's Law [Scanning Our Past]"

Extra Transistor Uses

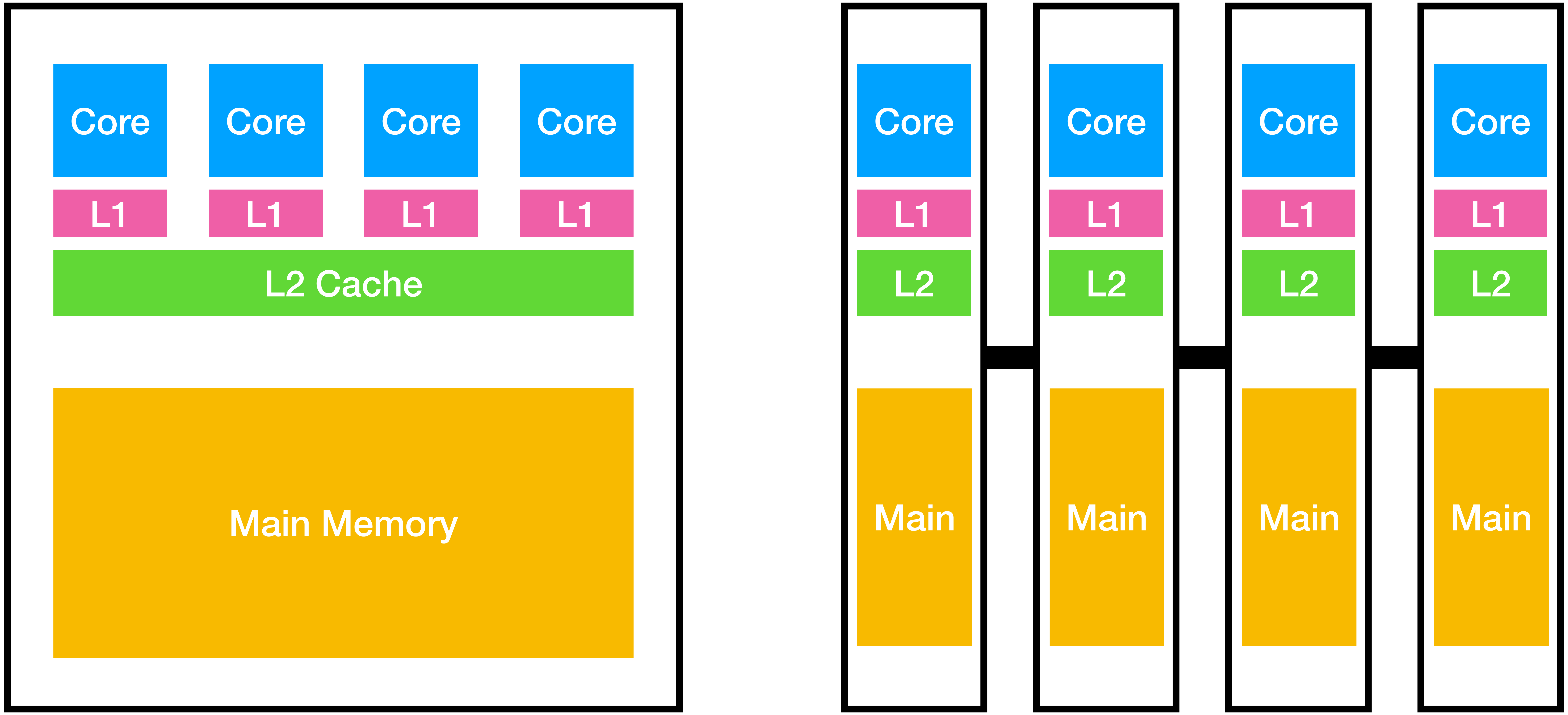
- Maintain same power, increase clock speed
- Maintain same power and clock speed, increase functionality (parallelism)
- Maintain same clock speed, use less power

Moore's Law



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Distributed vs Shared Memory



Shared **Distributed**

Today's Supercomputers

Must communicate data between processes or nodes

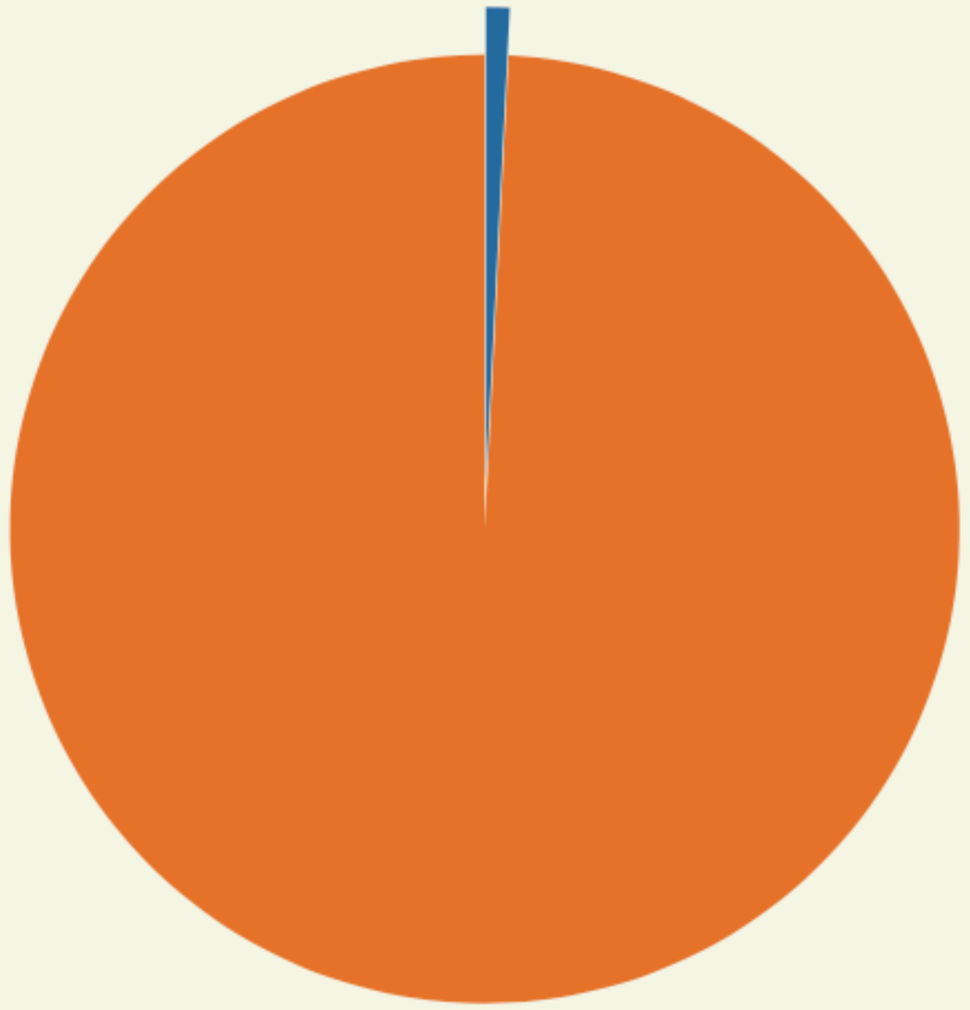
Why Parallelism?

Example

Let's take a 16-core CPU with hyperthreading and a 256 bit-wide vector unit, commonly found in home desktops. A serial program using a single core and no vectorization only uses 0.8% of the theoretical processing capability of this processor! The calculation is

$$16 \text{ cores} \times 2 \text{ hyperthreads} \times (256 \text{ bit-wide vector unit})/(64\text{-bit double}) = 128\text{-way parallelism}$$

where 1 serial path/128 parallel paths = .008 or 0.8%. The following figure shows that this is a small fraction of the total CPU processing power.



A serial application only accesses 0.8% of the processing power of a 16-core CPU.

Why Parallelism?

- Let's look at dense matrix multiplication

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 5 \\ 2 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 2 & 3 \\ 5 & 7 & 1 \\ 2 & 0 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 18 & 16 & 14 \\ 21 & 13 & 25 \\ 11 & 11 & 10 \end{bmatrix}$$

Why Parallelism?

- Let's look at dense matrix multiplication

$$\begin{bmatrix}1 & 2 & 3\\ 3 & 1 & 5\\ 2 & 1 & 1\end{bmatrix} \times \begin{bmatrix}2 & 2 & 3\\ 5 & 7 & 1\\ 2 & 0 & 3\end{bmatrix} \rightarrow \begin{bmatrix}\boxed{18} & 16 & 14\\ 21 & 13 & 25\\ 11 & 11 & 10\end{bmatrix}$$

- 1x2 + 2x5 + 3x2 = 18

Why Parallelism?

- Let's look at dense matrix multiplication

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 5 \\ 2 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 2 & 3 \\ 5 & 7 & 1 \\ 2 & 0 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 18 & \boxed{16} & 14 \\ 21 & 13 & 25 \\ 11 & 11 & 10 \end{bmatrix}$$

- 1x2 + 2x7 + 3x0 = 16

Why Parallelism?

Let’s look at dense matrix-matrix multiplication

$$\underset{n \times n}{\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 5 \\ 2 & 1 & 1 \end{bmatrix}} \times \underset{n \times n}{\begin{bmatrix} 2 & 2 & 3 \\ 5 & 7 & 1 \\ 2 & 0 & 3 \end{bmatrix}} \rightarrow \underset{n \times n}{\begin{bmatrix} 18 & 16 & 14 \\ 21 & 13 & 25 \\ 11 & 11 & 10 \end{bmatrix}}$$

$\mathcal{O}(n^3)$ operation

In this example, n is 3, requires 27 calculations.
But what about for more realistic, larger n?

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let’s look at dense matrix-matrix multiplication

$$\underset{n \text{ x } n}{\begin{bmatrix} \quad & \quad \end{bmatrix}} \times \underset{n \text{ x } n}{\begin{bmatrix} \quad & \quad \end{bmatrix}} \rightarrow \underset{n \text{ x } n}{\begin{bmatrix} \quad & \quad \end{bmatrix}}$$

Assume a single core can compute 20 billion floating-point operations per second (20 GFlops/sec).

Let n be 100.
How long does matrix-matrix multiplication take?
What about for n = 1000? 10,000? 100,000?

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let's look at dense matrix-matrix multiplication

$$\underbrace{[\quad\quad]}_{n \times n} \times \underbrace{[\quad\quad]}_{n \times n} \rightarrow \underbrace{[\quad\quad]}_{n \times n}$$

n	n^3	Time
100	1 million	.00005s
1,000	1 billion	.05s
10,000	1E+12	50s
100,000	1E+15	50,000s

13.9 hours

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let's look at dense matrix-matrix multiplication

n x

x n

Use parallelism, because otherwise the application will take too long!

10,000	1E+12	50s
100,000	1E+15	50,000s

13.9 hours

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let's look at dense matrix-matrix multiplication

$$\underbrace{[\quad\quad]}_{n \times n} \times \underbrace{[\quad\quad]}_{n \times n} \rightarrow \underbrace{[\quad\quad]}_{n \times n}$$

Assume this computer has 4 gigabytes of memory
and matrix is full of double precision numbers

For n = 100; 1,000; 10,000; 100,000
How much memory does matrix-matrix multiplication require?

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let’s look at dense matrix-matrix multiplication

n x n $\times$ n x n $\rightarrow$ n x n

n	$3 \times n^2$	Bytes
100	30,000	240,000
1,000	3 million	24 million
10,000	300 million	2.4 billion
100,000	30 billion	240 billion

240 Gigabytes

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let's look at dense matrix-matrix multiplication

n x n

x n

Use parallelism, because otherwise you will run out of memory!

10,000	300 million	2.4 billion
100,000	30 billion	240 billion

240 Gigabytes

Why Parallelism?

- Energy Consumption
- P = (N Processors) x (R W/Processor) x (T hours)
- Gives P in watt-hours, likely want it in kWhrs
  - Divide by 1000

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let's look at dense matrix-matrix multiplication

$$[\quad] \times [\quad] \rightarrow [\quad]$$

n x n   n x n   n x n

- Assume 120W processes
- 1 process: matrix multiplication takes 8 hours

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let's look at dense matrix-matrix multiplication

$$\underset{n \times n}{\left[\quad\right]} \times \underset{n \times n}{\left[\quad\right]} \rightarrow \underset{n \times n}{\left[\quad\right]}$$

- Assume 120W processes
- 8 processes: matrix multiplication takes 2 hours

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let's look at dense matrix-matrix multiplication

$$\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}_{n \times n} \times \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}_{n \times n} \rightarrow \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}_{n \times n}$$

- Assume now on GPUs : 300W GPUs
- 1 GPU: matrix multiplication takes 2 hours

Why Parallelism?

$\mathcal{O}(n^3)$ operation

Let's look at dense matrix-matrix multiplication

n x n

n x n

Use energy-efficient parallelism, such as GPUs, to solve problems with less energy!

- Assume now or

- 1 GPU: matrix multiplication takes 2 hours

Amdahl's Law

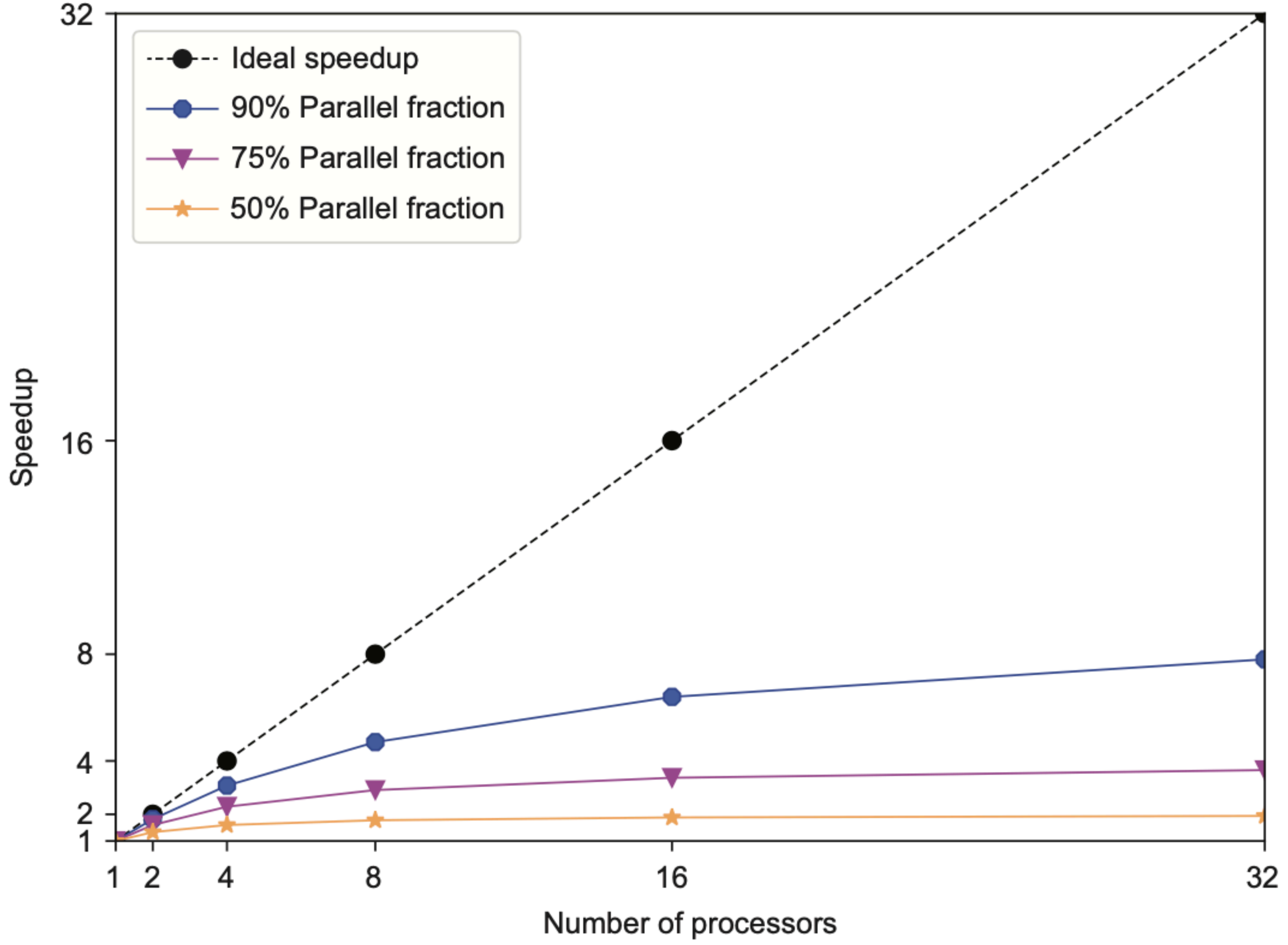
- Speedup(N) = $\dfrac{1}{S + \frac{P}{N}}$
  - S : serial fraction of code
  - P : parallel fraction of code
  - N : number of processes

Amdahl's Law

- Speedup(N) = $\dfrac{1}{S + \frac{P}{N}}$
  - S : serial fraction of code
  - P : parallel fraction of code
  - N : number of processes
- **No matter how fast we make the parallel portion, we will always be limited by the serial portion!**

Amdahl's Law

- Fixed problem size:



Amdahl's Law

- Scaled problem size: