MAP55611 - High-Performance Computing

Mike Peardon — mjp@maths.tcd.ie
School of Mathematics

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Michaelmas Term 2024/25

MAP55611 — Part I

Introduction to parallel software for HPC

What is HPC?

- Solve large-scale challenging numerical and data-driven problems

> HPC methods effectively harness significant computing resources to solve technical numerical problems at the largest possible scales or finest level of detail.

- Where are the limits of memory and computational resources?
- How can these resources be coordinated and harnessed effectively?
- What level of accuracy and efficiency can be achieved?

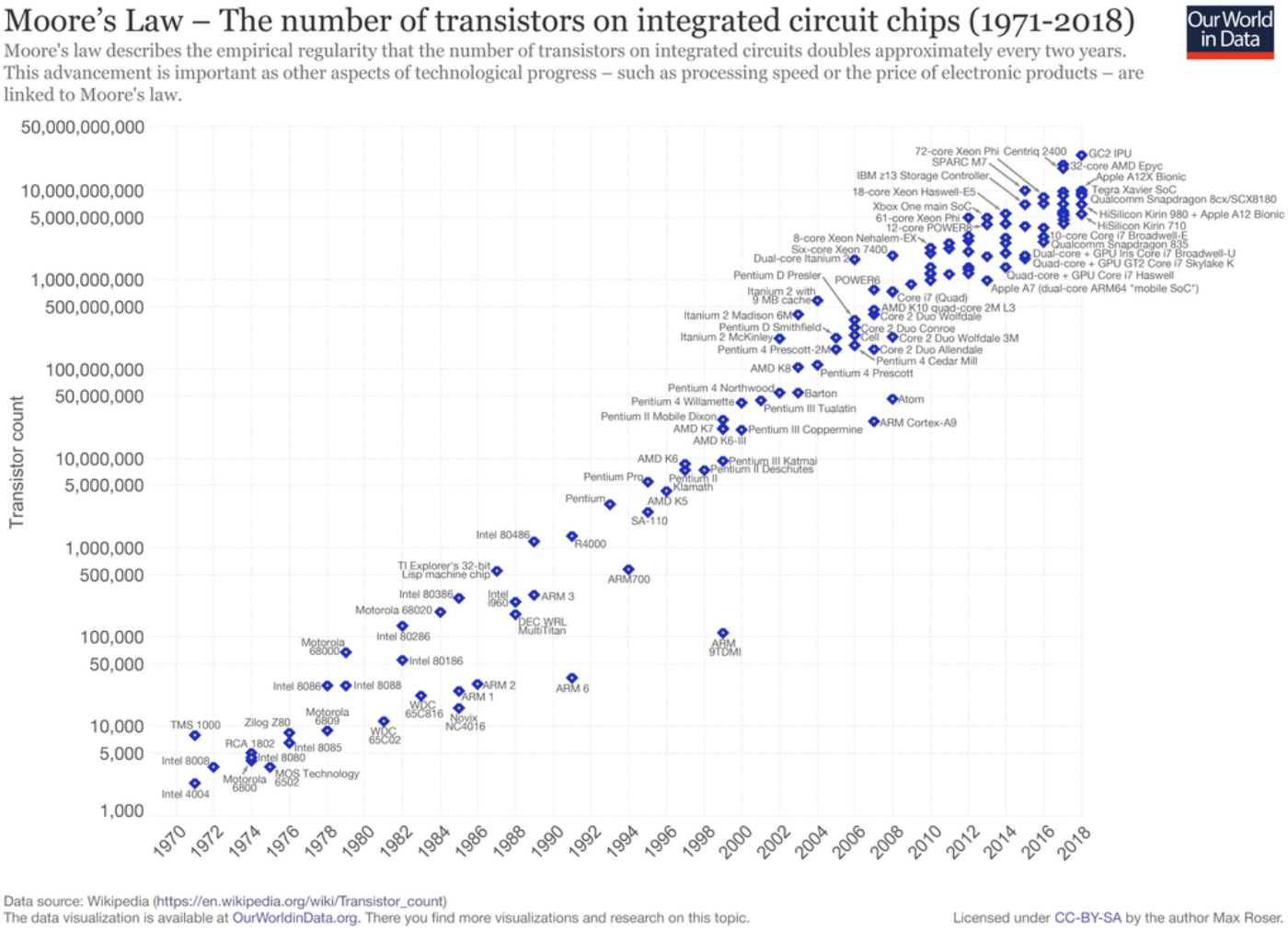
To be an HPC guru means understanding the connections between:

- **Mathematical methods** and algorithms (numerical and data-driven)
- Techniques for **effective software** to exploit multiple compute cores
- Essentials of the **hardware** of parallel and hybrid computers.

This module: parallel software on CPUs

Moore's law (1)

Number of transistors on an integrated circuit doubles ≈ every 2 years.

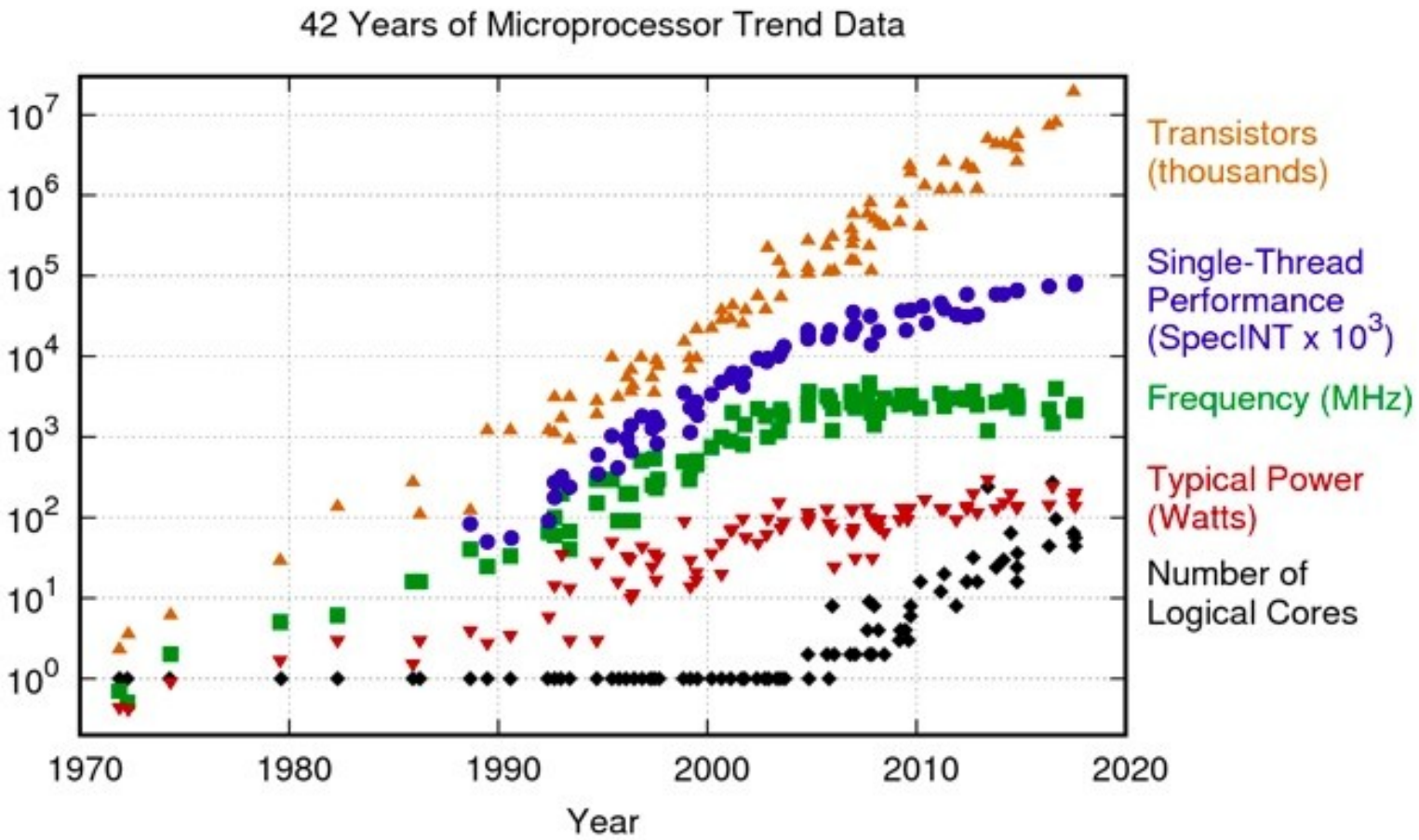


Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Licensed under CC-BY-SA by the author Max Roser.

Credit: https://ourworldindata.org/uploads/2019/05/Transistor-Count-over-time-to-2018.png

Moore's law (2)



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

https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Moore's law (3)

- Number of transistors on an integrated circuit doubles approximately every two years.
- Clock speed has not increased much over the past two decades.
- Number of cores on each integrated circuit has grown significantly in that same time.

Need for algorithms and software that can exploit multiple compute cores is becoming more important to solve bigger problems.

Amdahl's law (1)

Suppose a composite **fixed** task has some parts that can be parallelised and other parts that can not:

- p: fraction of whole task that can be perfectly parallelised.
- n: number of compute cores available.
- t_n: execution time on n cores (so t_1 is the serial run-time).

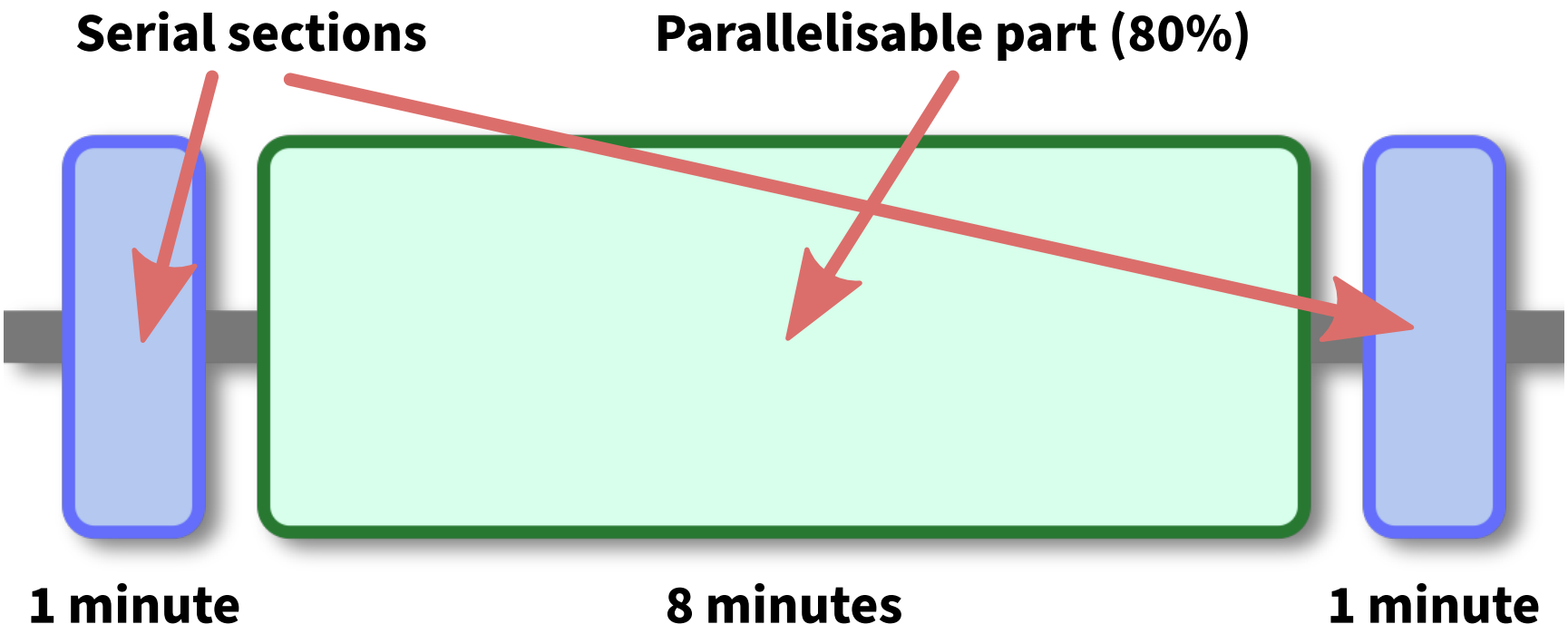
Amdahl's law

$$t_n = \left(\frac{p}{n} + (1-p)\right) t_1$$

Consequence: The limit to how much speed-up to expect is:

$$\lim_{n\to\infty} \quad t_n = (1-p)t_1 \quad \text{or} \quad \lim_{n\to\infty} \quad t_1/t_n = \frac{1}{1-p}$$

Amdahl's law (2)



Total execution time: 10 minutes

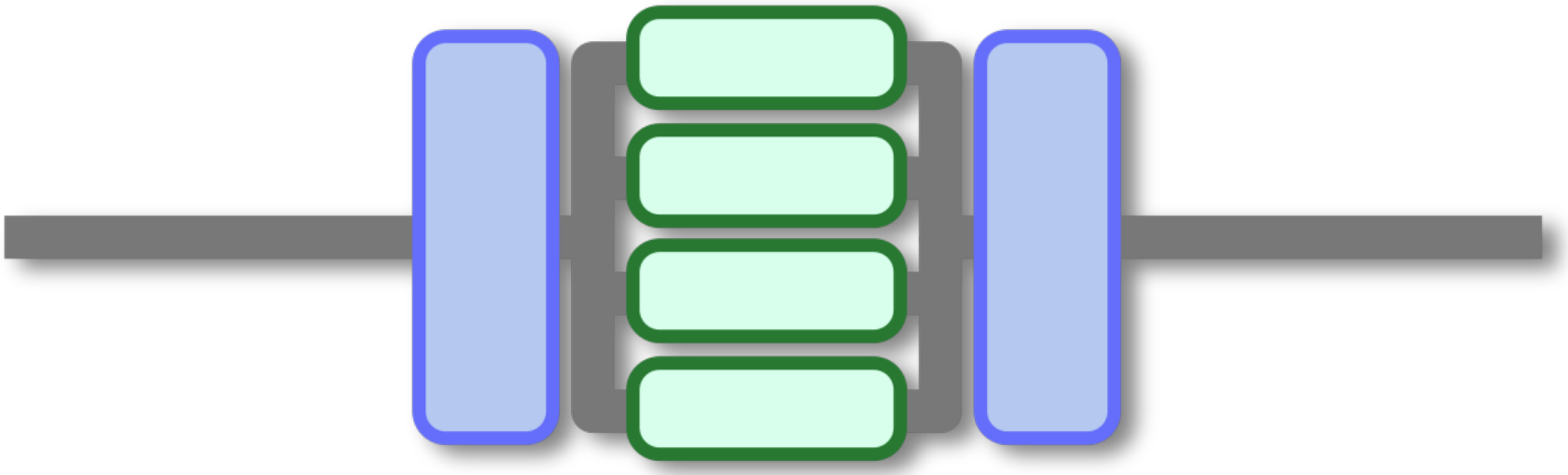
Amdahl's law (2)

1 minute 4 minutes 1 minute

Total execution time: 6 minutes

Amdahl's law (2)

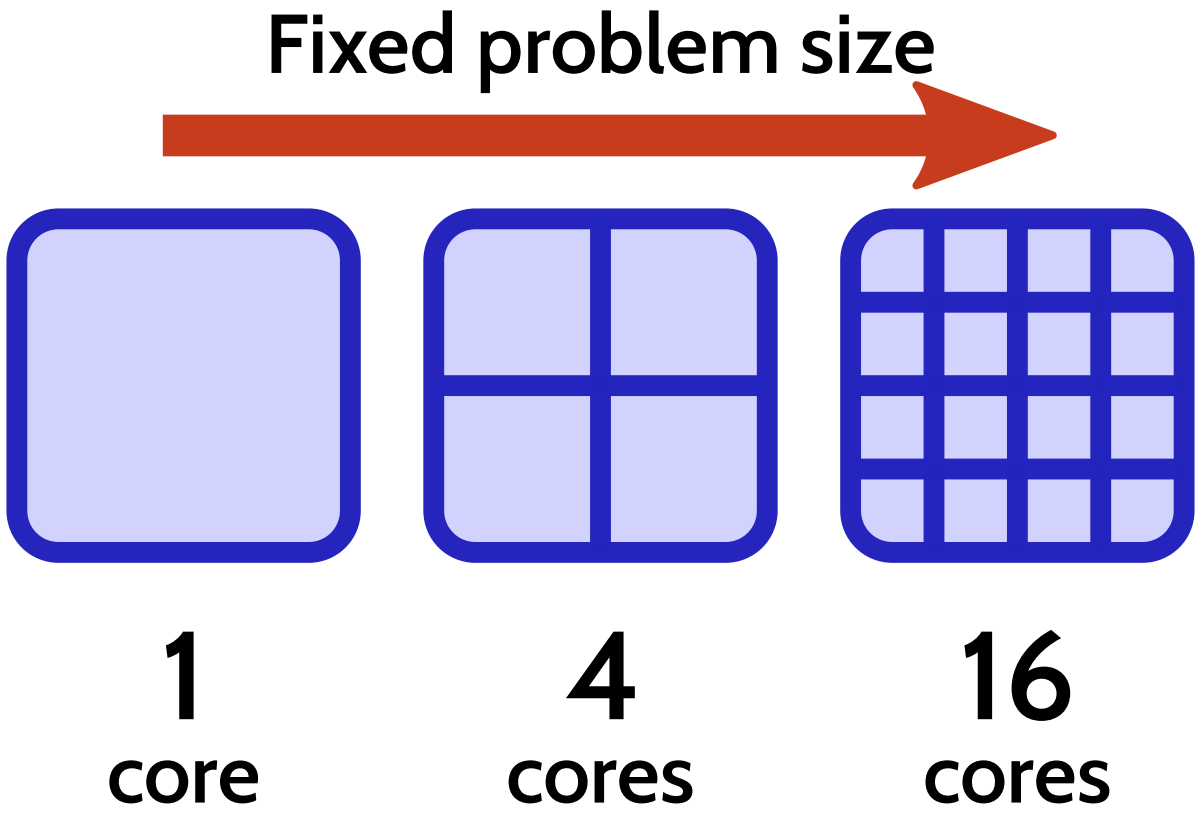
$n = 4$ **four-way parallelism**



1 minute **2 minutes** **1 minute**

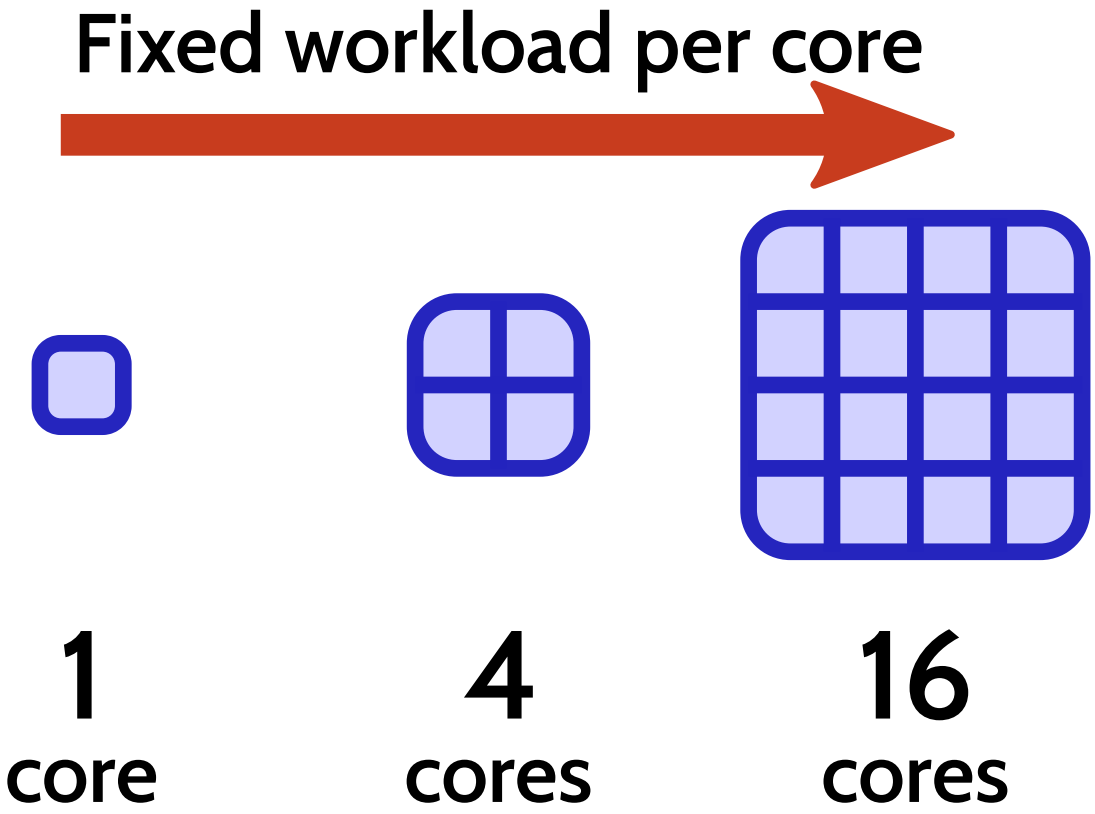
Total execution time: 4 minutes

Strong scaling



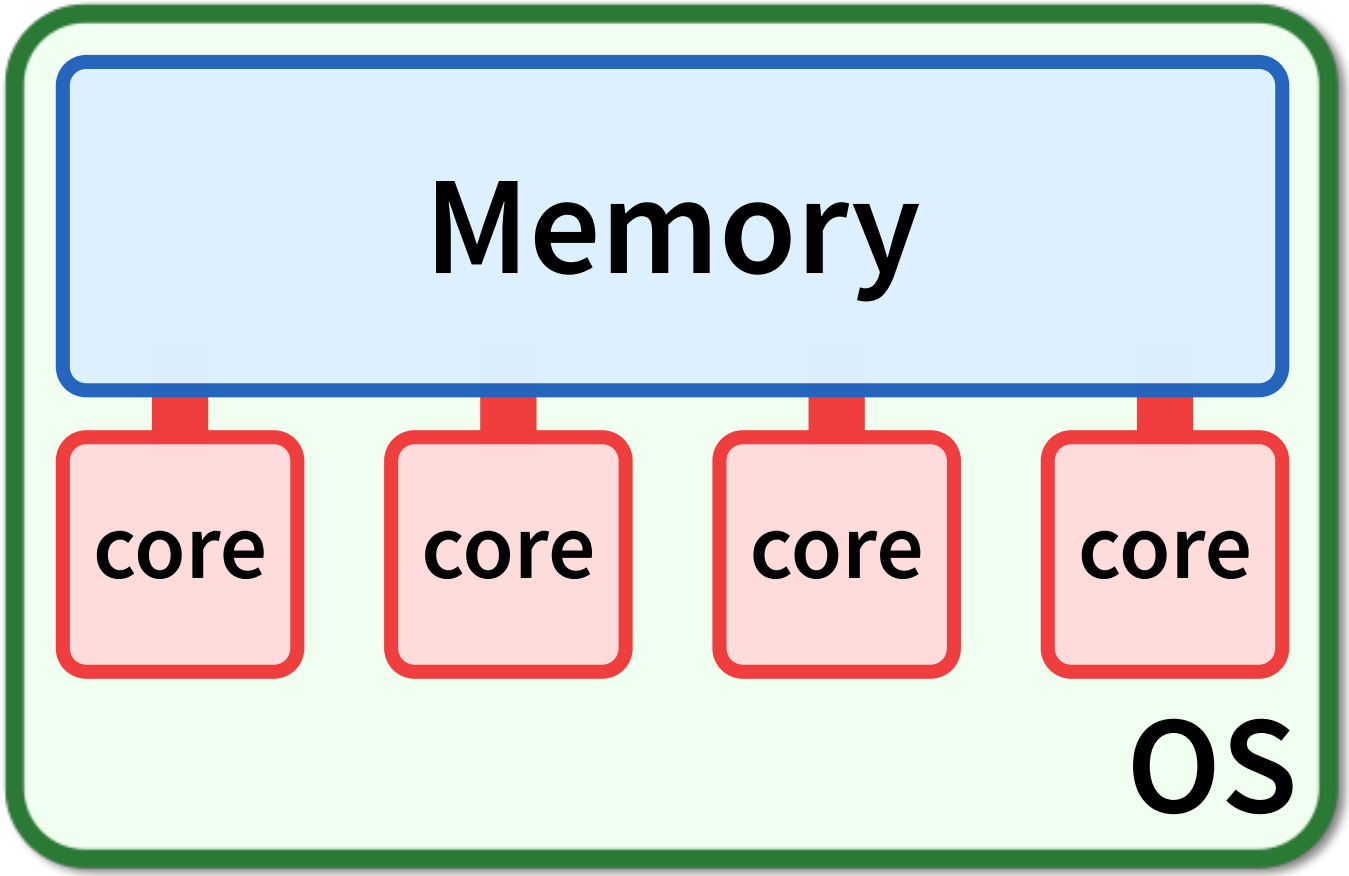
- Perfect strong scaling means run-time **falls** in inverse proportion to the number of cores used.
- Amdahl's law limits strong scaling if parts of the task are not parallelisable.

Weak scaling



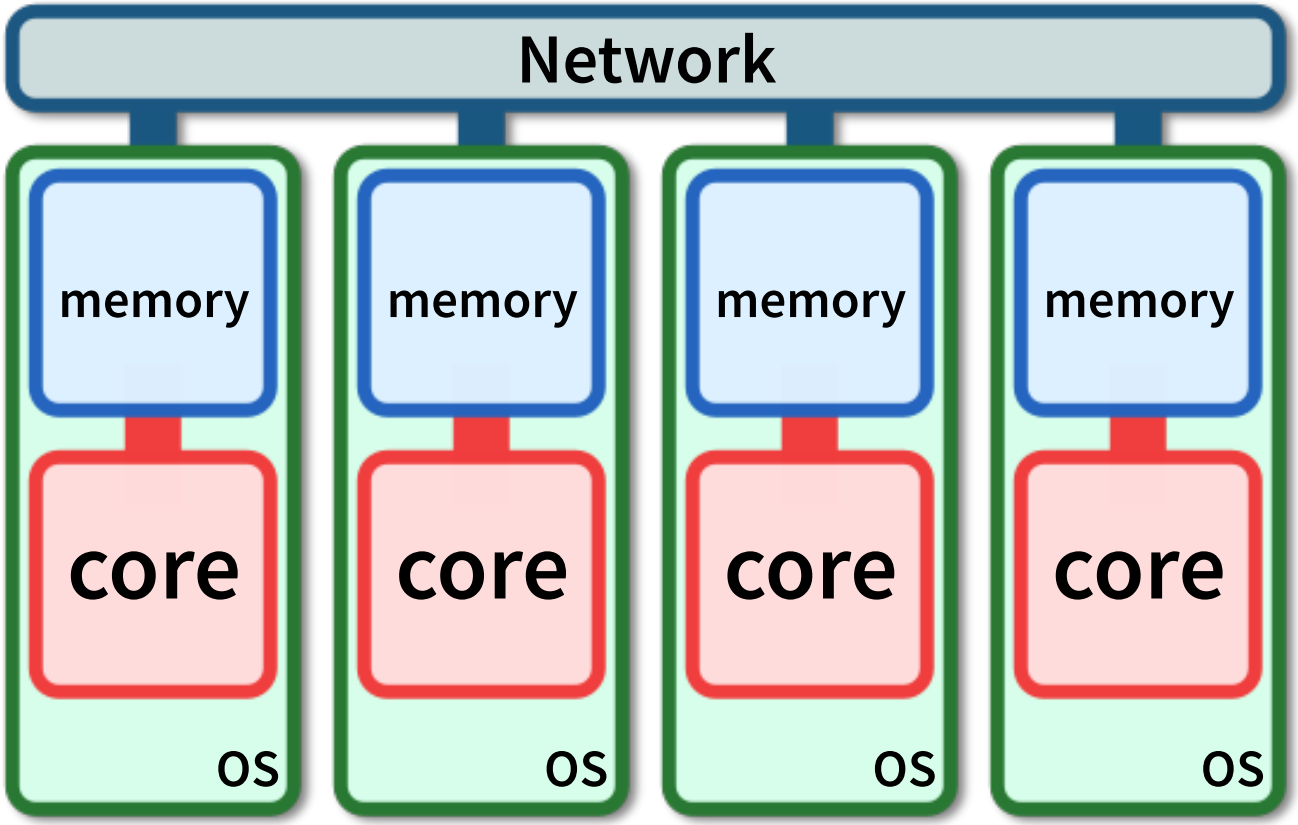
- Perfect weak scaling means run-time **unchanged** as more cores used.
- Usually easier to get good weak scaling vs strong scaling.

The shared memory model



- Single operating-system manages whole system.
- Can use eg. OpenMP programming model
- Most computers look like this today

The distributed memory model



- Multiple autonomous systems (OS) working simultaneously
- Connected via a (fast) network
- `MPI` programming model

Summary

- **Moore's Law**
  - Number of transistors on an integrated circuit doubles (about) every two years.
  - Most CPUs now host multiple independent cores.
- **Amdahl's Law**
  - How much faster will my code run when I use more cores?
- **Strong** vs **weak** scaling.
  - Strong scaling - fixed problem size - does run-time fall when more cores are used?
  - Weak scaling - problem size proportional to number of cores - does run-time stay the same?
- **Shared** vs **distributed** memory model.
  - Determines programming model - (eg) `OpenMP` vs `MPI`