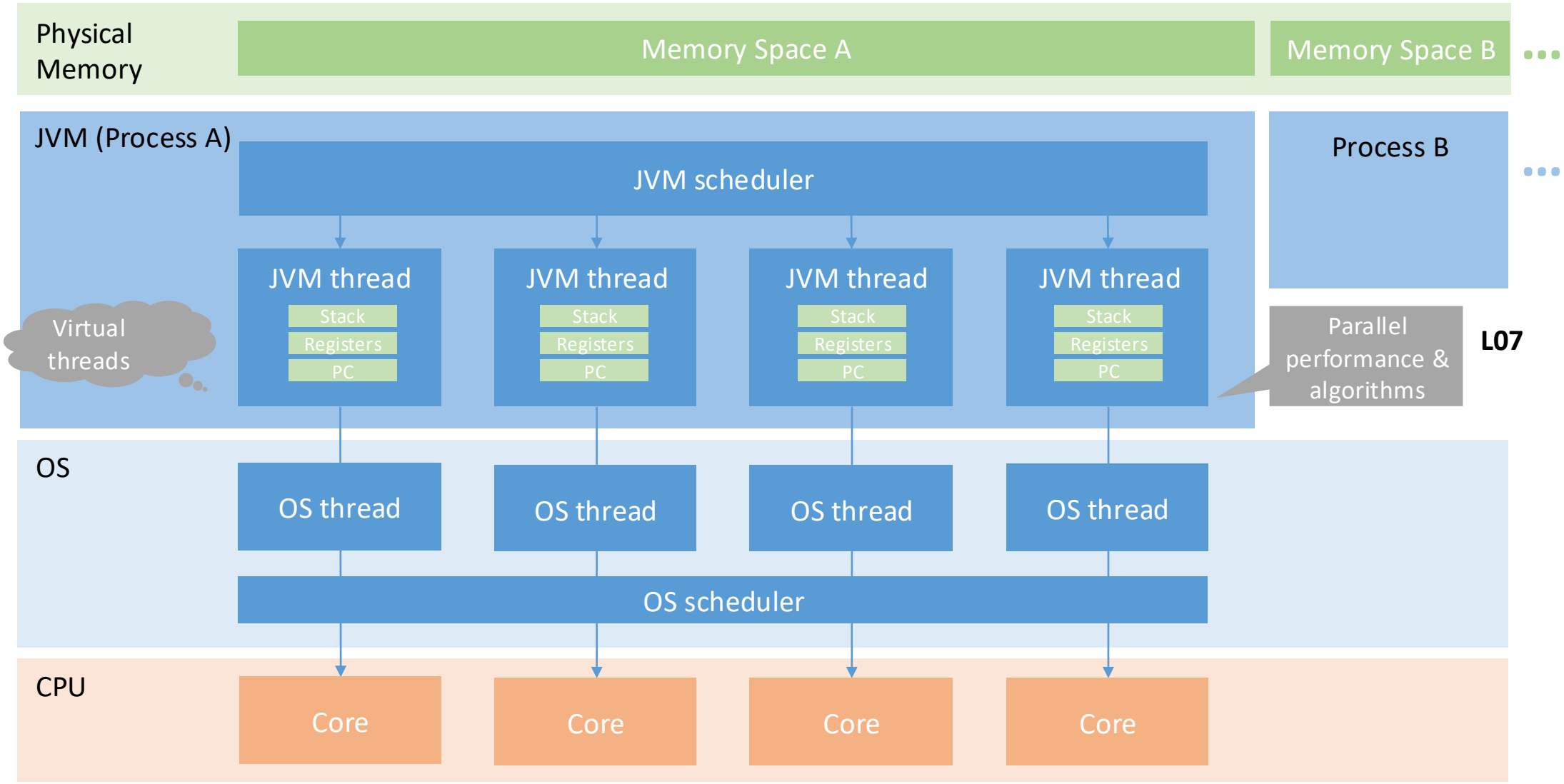
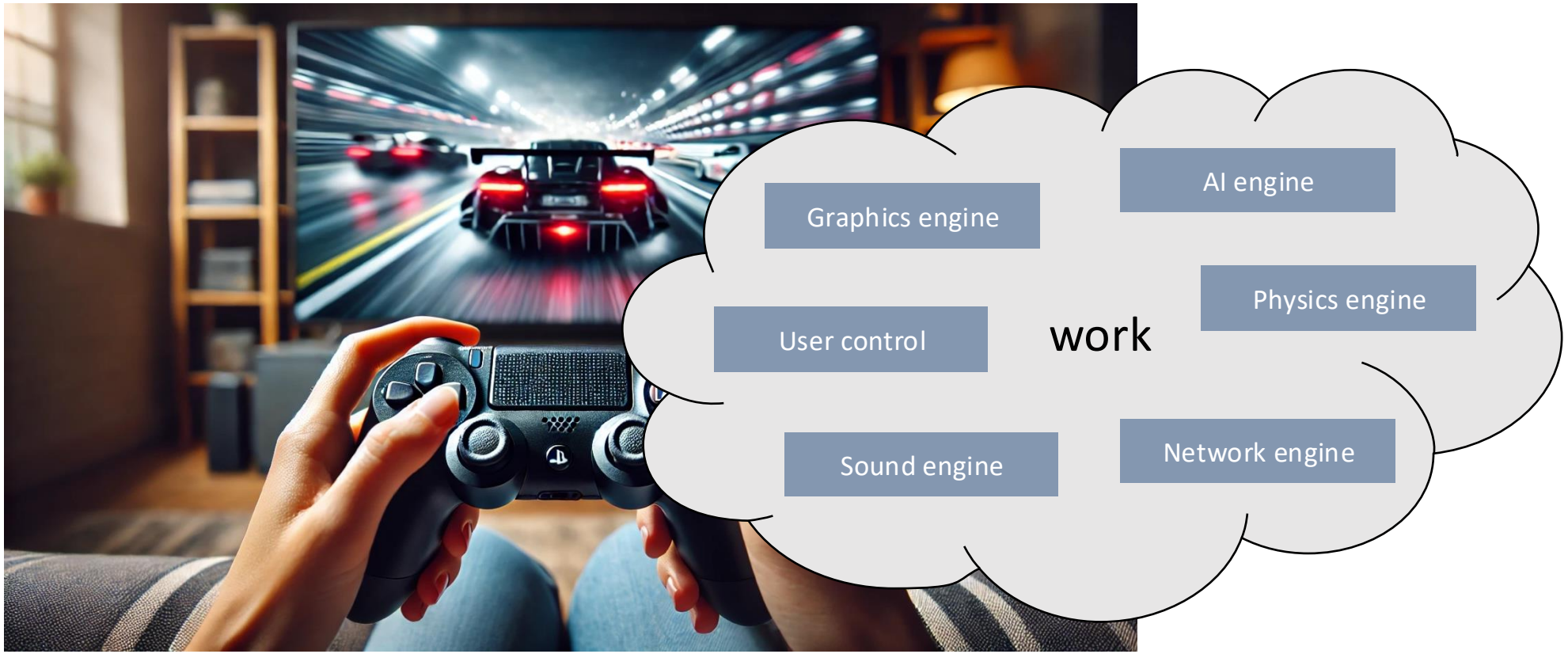
Parallel Programming

Basic Concepts in Parallelism: Speed-up, Efficiency, Amdahl, Gustafson

Big Picture (Part I)

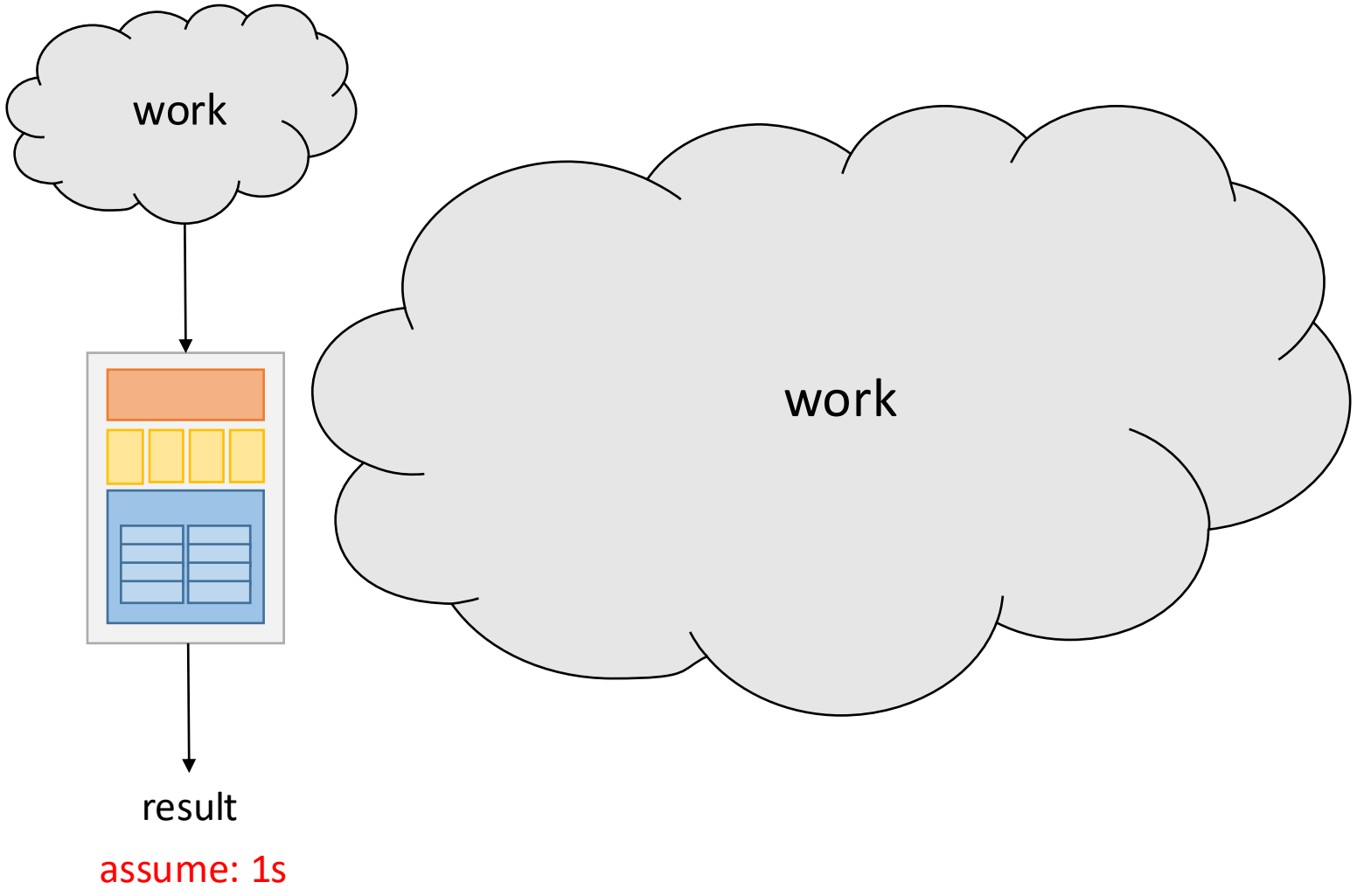


Example: Video games require real-time computation

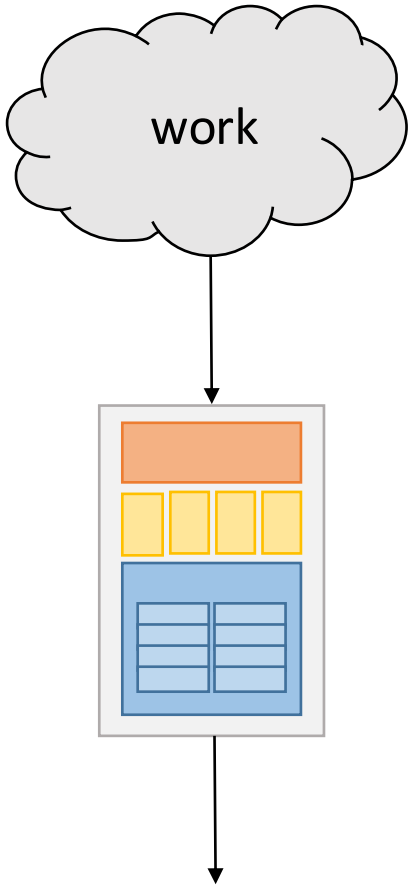


Video game must run at 30fps → 0.033s to compute a single frame

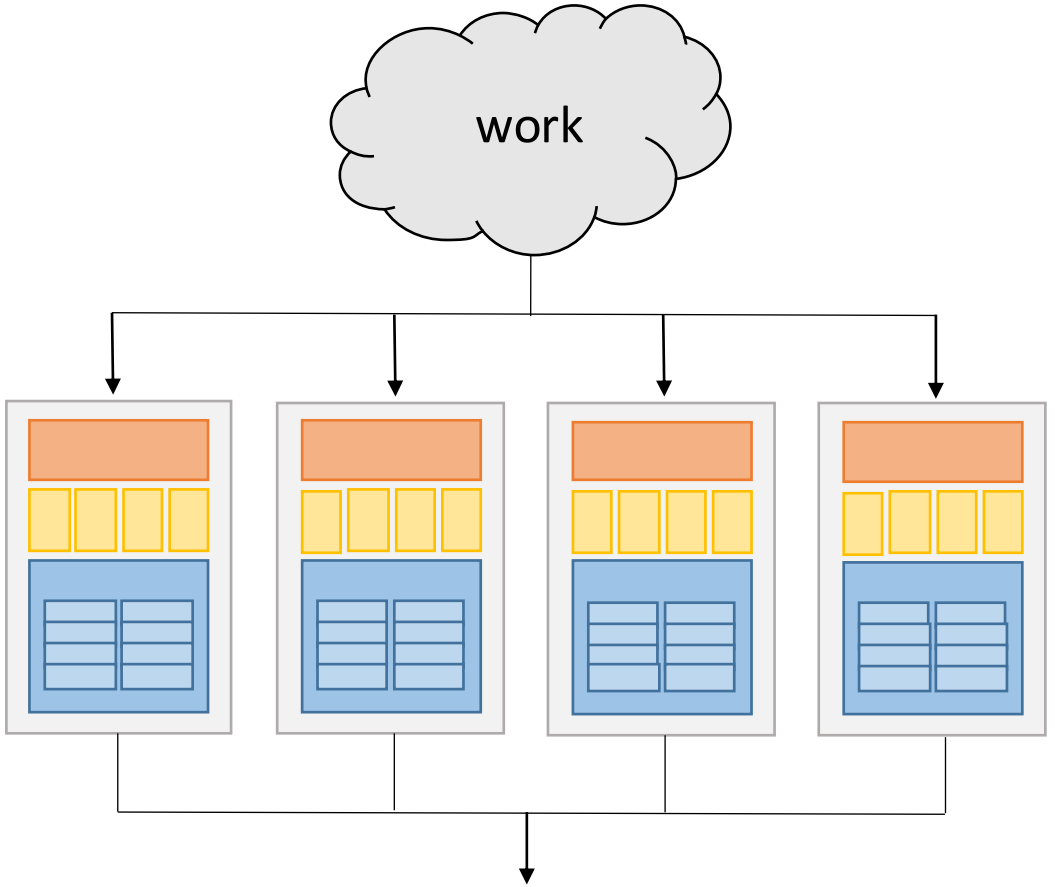
What if this video game ran on a single-core machine?



Throwing more cores at the problem – is it worth it?

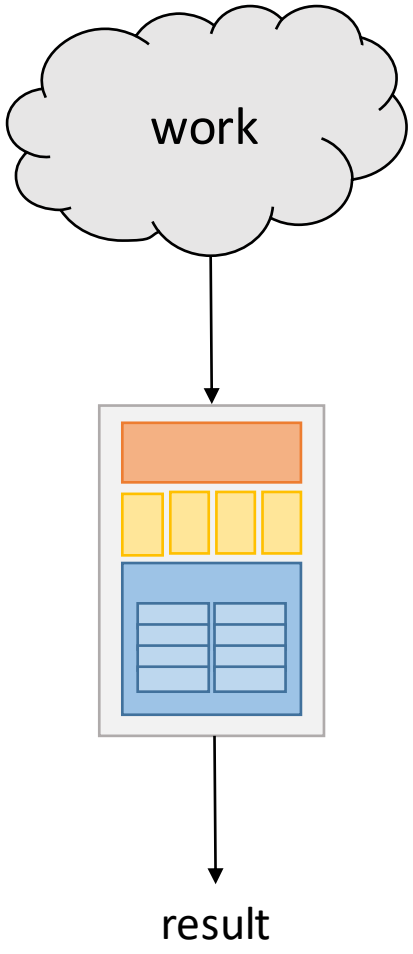


assume: 1s

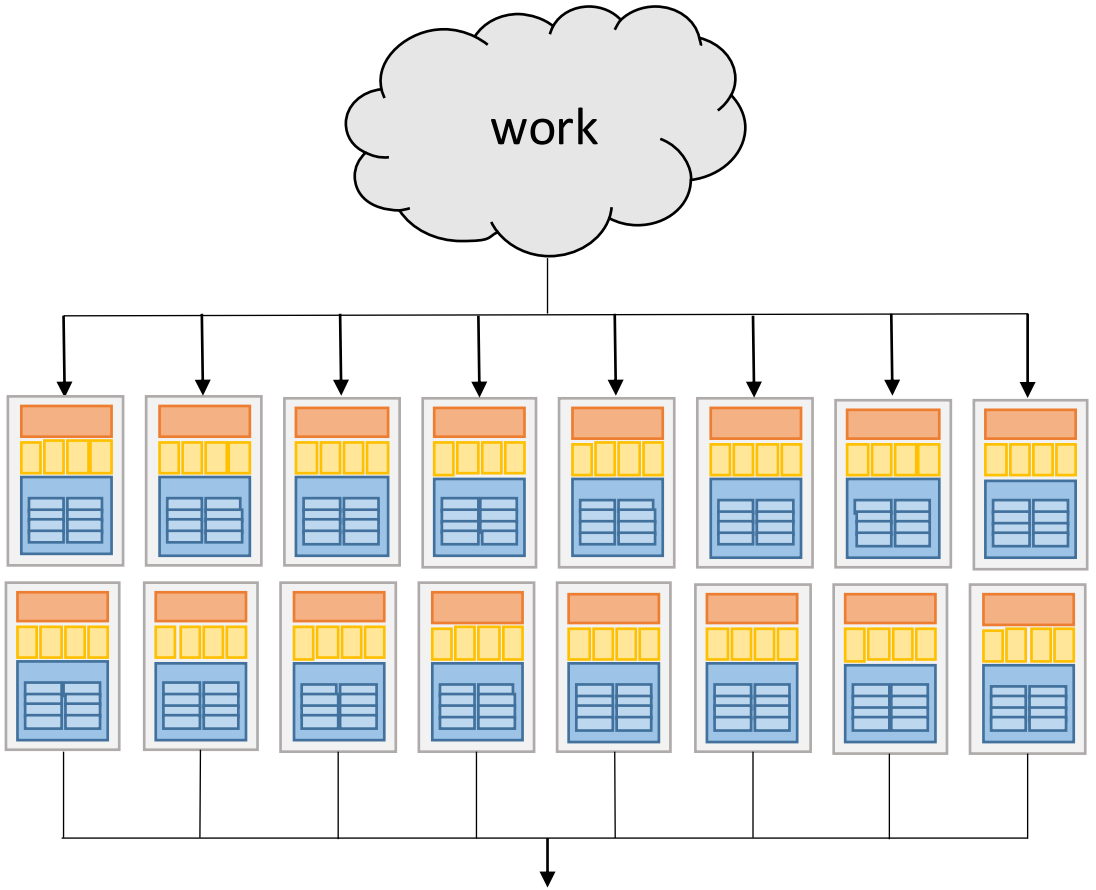


0.25s?

Throwing more cores at the problem – is it worth it?

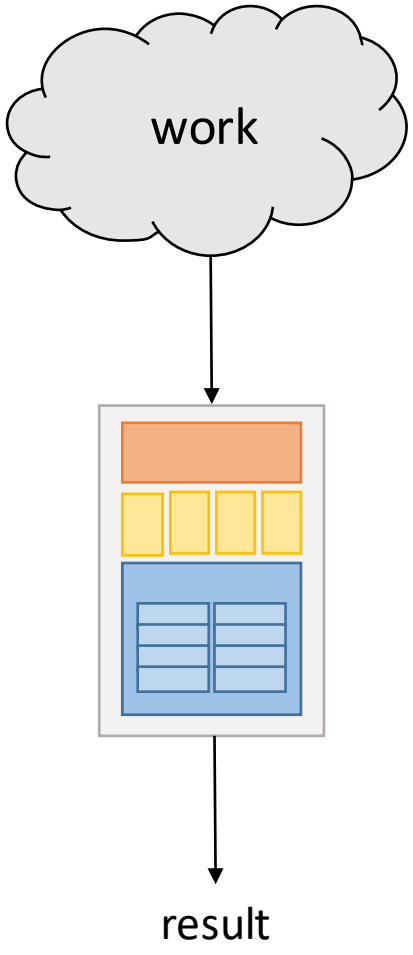


assume: 1s

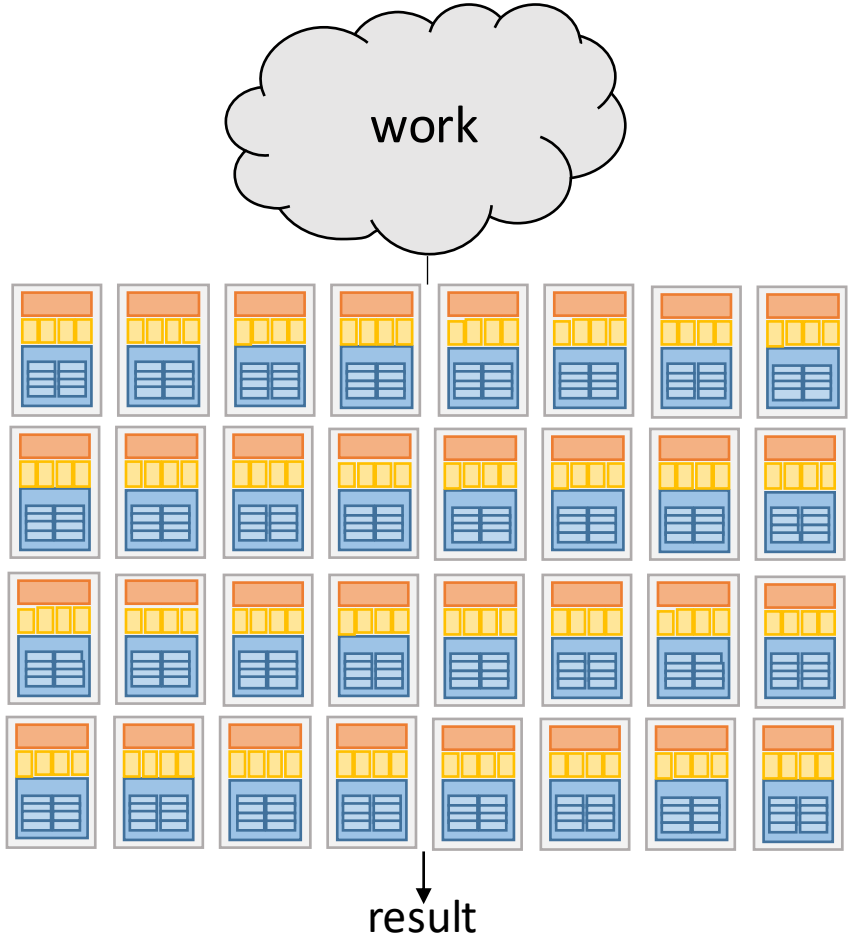


0.0625s?

Throwing more cores at the problem – is it worth it?



assume: 1s



0.03135?

Scalability

How does performance improve with additional cores?

More cores do not always result in a linear speedup
(or any speedup at all!)

Fast != efficient

Just because your program runs faster on a parallel computer, it does not mean it is using the hardware efficiently

- Is 2x speedup on computer with 10 processors a good result?
- Is increase in power consumption justified by performance gains?

What factors limit scalability?

Sequential part of the program (Amdahl's law)

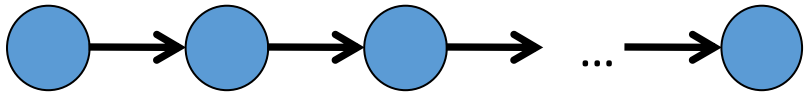
Can tasks be executed independently or do they rely on one another?

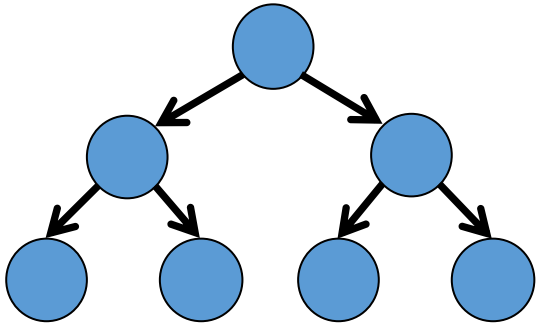
Even with ideal parallelization, some problems have inherent dependencies that limit speedup

Even if 99% of the program is parallelized, the remaining 1% sequential part dominates at high core counts

Data structures and algorithms

Some data structures require sequential access





Some algorithms naturally expose more parallelism than others

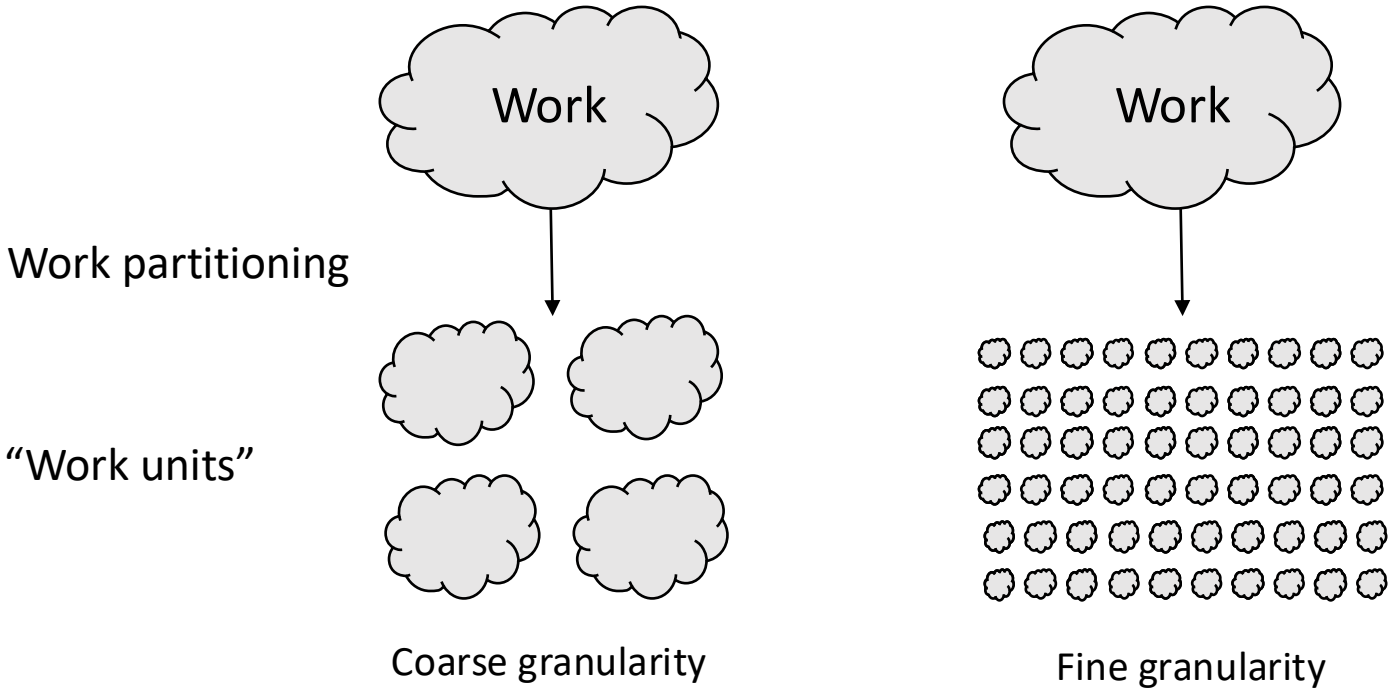
Algorithms can be restructured to improve parallelism

Parallel efficiency starts with the right data structures and algorithms (L10 / 11)

Work distribution strategy

How evenly can work be distributed among cores?

Some workloads create imbalances, causing some cores to remain idle while others are overloaded



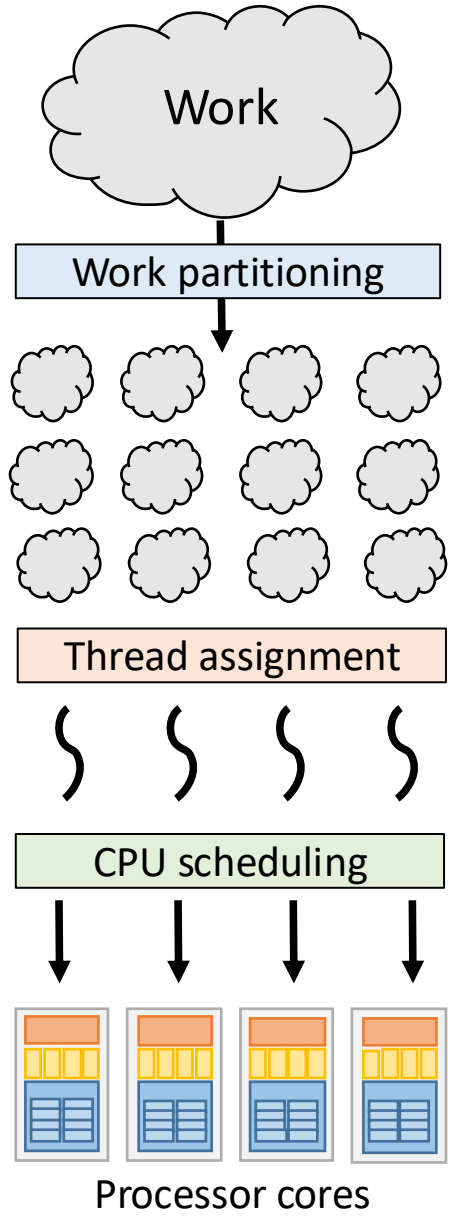
Work scheduling strategy

Goal: **full utilization** (no processor is idle)

How should work units be assigned to threads (L08/09)?

Efficiency of user-level scheduling: How quickly can tasks be assigned and reassigned to threads?

Overhead from task creation and management can offset gains

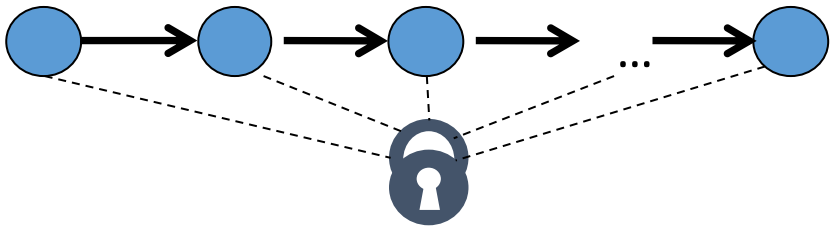


Overhead and synchronization barriers

How much time is spent on synchronization, locking, context switching?

Frequent context switches introduce delays that degrade parallel performance

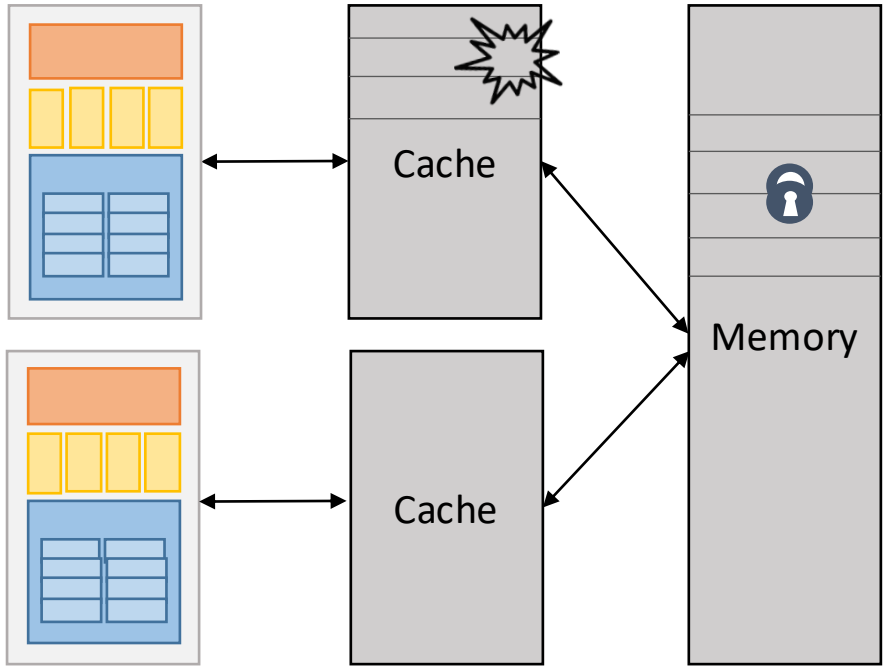
High contention for shared resources or excessive synchronization barriers create bottlenecks that limit parallel efficiency



Memory access and caches

Memory access contention: If all cores access shared memory, memory bandwidth becomes a bottleneck

More cores may lead to increased cache misses if data is not well-partitioned



Parallel performance

Scalability

How does performance improve with additional cores?

- Speedup when we increase processors
- A program scales linearly → linear speedup
- What will happen if processors → ∞

Parallel performance

Sequential execution time: $\mathbf{T_1}$

Execution time $\mathbf{T_p}$ on $\mathbf{p}$ CPUs

- $\mathbf{T_p} = \mathbf{T_1} / \mathbf{p}$ (perfection)
- $\mathbf{T_p} > \mathbf{T_1} / \mathbf{p}$ (performance loss, what normally happens)
- $\mathbf{T_p} < \mathbf{T_1} / \mathbf{p}$ (sorcery!)

(parallel) speedup

(parallel) speedup S_p on **p** CPUs:

$$S_p = T_1 / T_p$$

- $S_p = p$ → linear speedup (perfection)
- $S_p < p$ → sub-linear speedup (performance loss)
- $S_p > p$ → super-linear speedup (sorcery!)

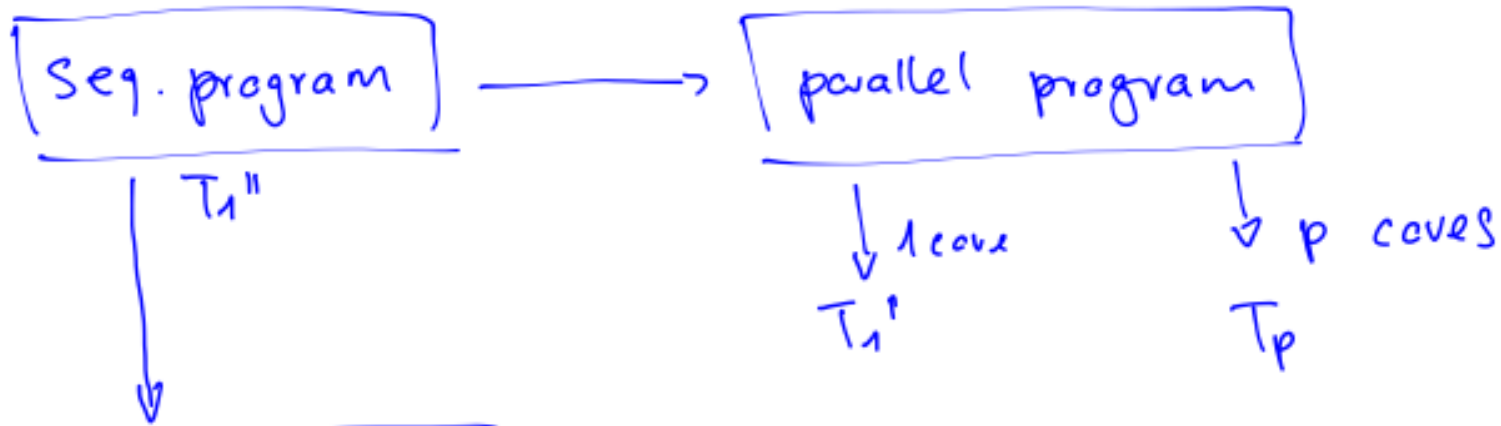
Efficiency: S_p / p

What is T1?

Isolate effect of parallelization: Baseline = serialization of the parallel algorithm

Fair comparison between sequential and parallel: Baseline = optimized sequential program (which might not parallelize well)

Using an unnecessarily poor baseline (unoptimized sequential program) artificially inflates speedup and efficiency

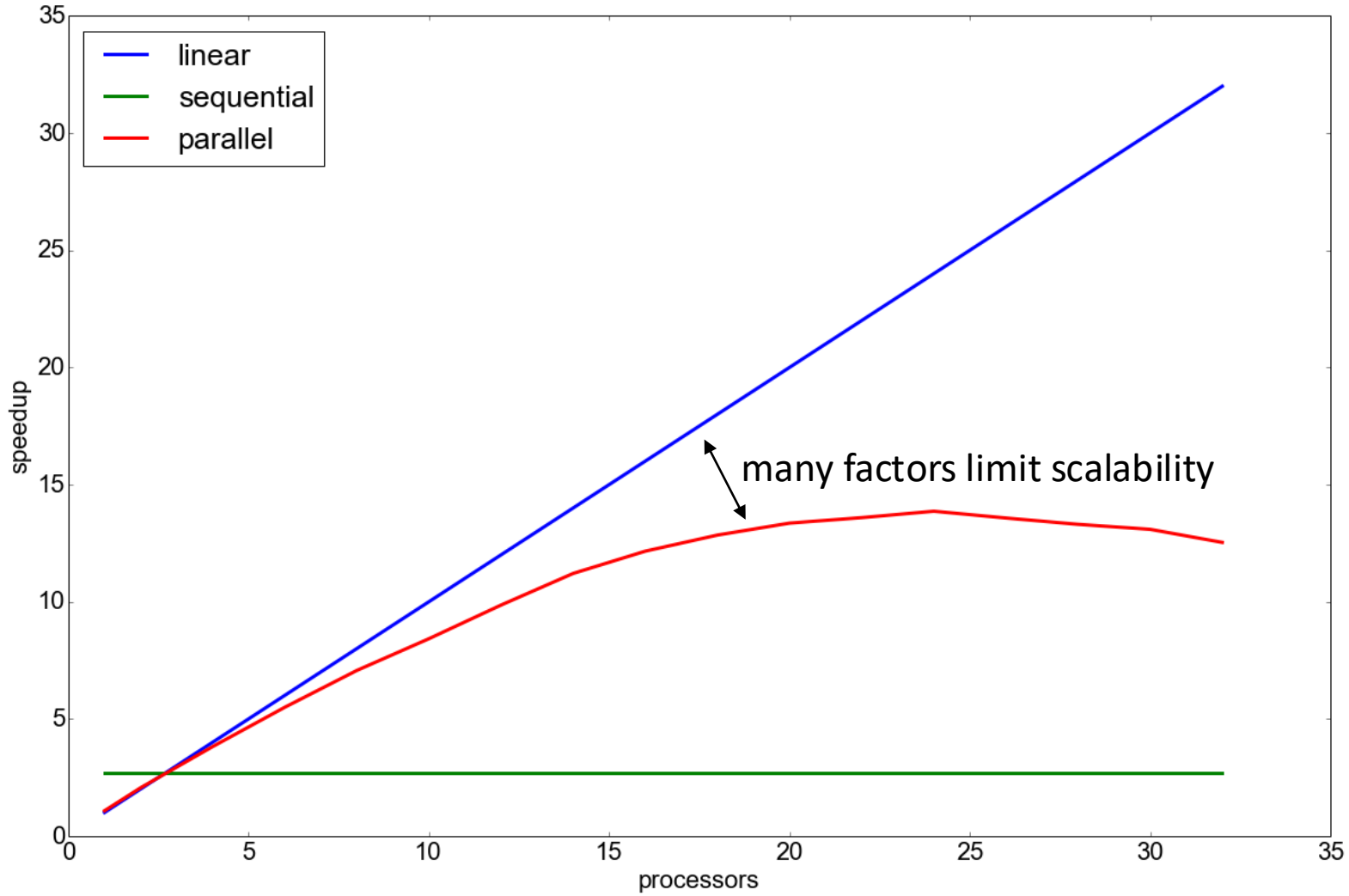

Seq. program
parallel program
T_1''
1 core
p cores
T_1'
T_p
algorithmically optimized seq. program
T_1'''
$T_1''' \ll T_p'' < T_1'$
fair: $T_1 = T_1'''$
scalability: $T_1 = T_1'$

$$S_p = 100$$

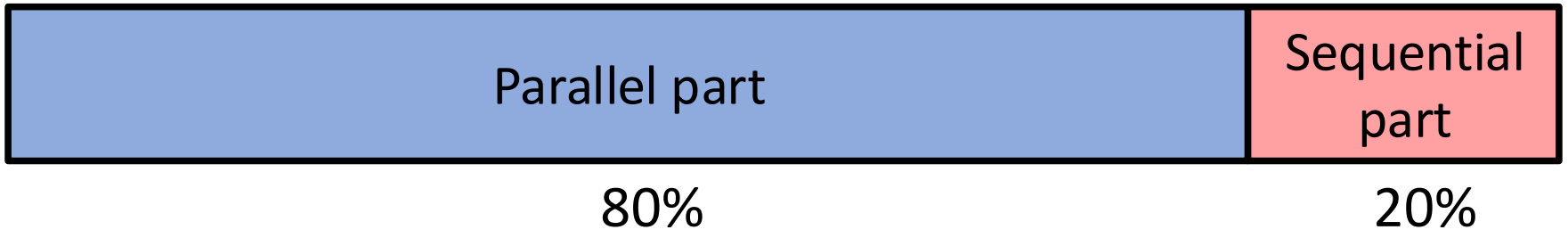
$$P = 1000$$

$$E = \frac{S_p}{P} = 0.1 \quad \Rightarrow 10\%$$

(Parallel) speedup graph example



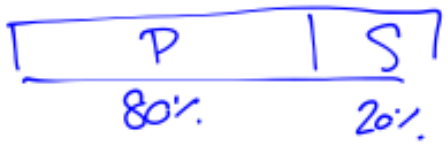
Question:



Parallel program:

- sequential part: 20%
- parallel part: 80% (assume it scales linearly)
- $T_1 = 10$

What is T_8 ? What is the speedup S_8 ?



$T_1 = 10$

$T_{seq} = 10 \cdot 0.2$

$T_p = 10 \cdot 0.8$

$p = 8$

$T_8 = ?$

$$T_8 = T_{seq} + T_{par}$$

$$= 10 \cdot 0.2 + \frac{10 \cdot 0.8}{8}$$

$$= 2 + 1$$

$$= 3$$

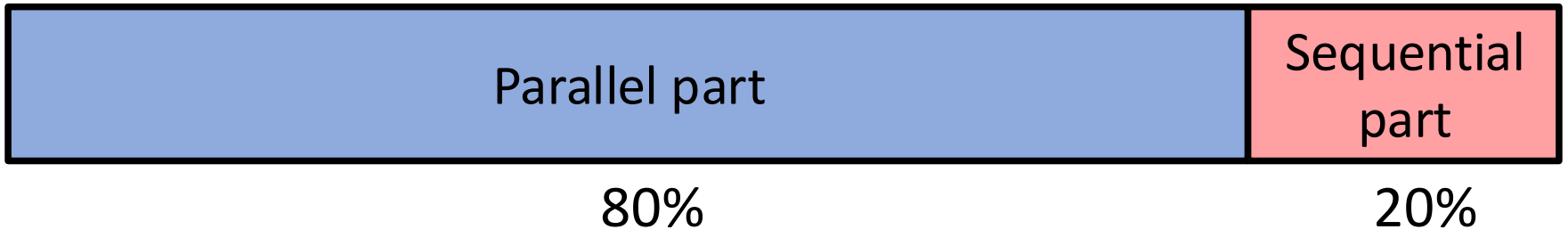
$$S_8 = \frac{T_1}{T_8} = \frac{10}{3} = 3.33$$

$$E = \frac{3.33}{8} \approx 0.4 \Rightarrow 40\%$$

80% // 20% se

P	Tn	Tp	Sp	E
1	10	10	1	1
2	10	$2+\frac{8}{2}=6$	$\frac{10}{6}\simeq 1.6$	$\frac{1.6}{2}=83\%$
3	10	$2+\frac{8}{3}=4.6$	$\frac{10}{4.6}\simeq 2.1$	$\frac{2.1}{3}=71\%$
8	10	$2+\frac{8}{8}=3$	$\frac{10}{3}=3.3$	$\frac{3.3}{8}\simeq 40\%$

Answer:



- $T_1 = 10$
- $T_8 = 3$
- $S_8 = T_1/T_8 = 10/3 = 3.33$

Amdahl's Law

The maximum speedup from parallelization is limited by the fraction of a program that must run sequentially.

Amdahl's Law – Ingredients

Execution time T_1 of a program falls into two categories:

- Time spent doing non-parallelizable serial work
- Time spent doing parallelizable work

Call these W_{ser} and W_{par} respectively

Amdahl's Law – Ingredients

Given P workers available to do parallelizable work, the times for sequential execution and parallel execution are:

$$T_1 = W_{ser} + W_{par}$$

And this gives a bound on speed-up:

$$T_p \geq W_{ser} + \frac{W_{par}}{P}$$

Amdahl’s Law

Plugging these relations into the definition of speedup yields Amdahl's Law:

$$S_p \leq \frac{W_{ser} + W_{par}}{W_{ser} + \frac{W_{par}}{P}}$$

Amdahl's Law - Corollary

$$S_p \leq \frac{W_{ser} + W_{par}}{W_{ser} + \frac{W_{par}}{P}}$$

If $\boldsymbol{f}$ is the non-parallelizable serial fractions of the total work, then the following equalities hold:

$$W_{ser} = \boldsymbol{f} T_1,$$
$$W_{par} = (1 - \boldsymbol{f}) T_1$$

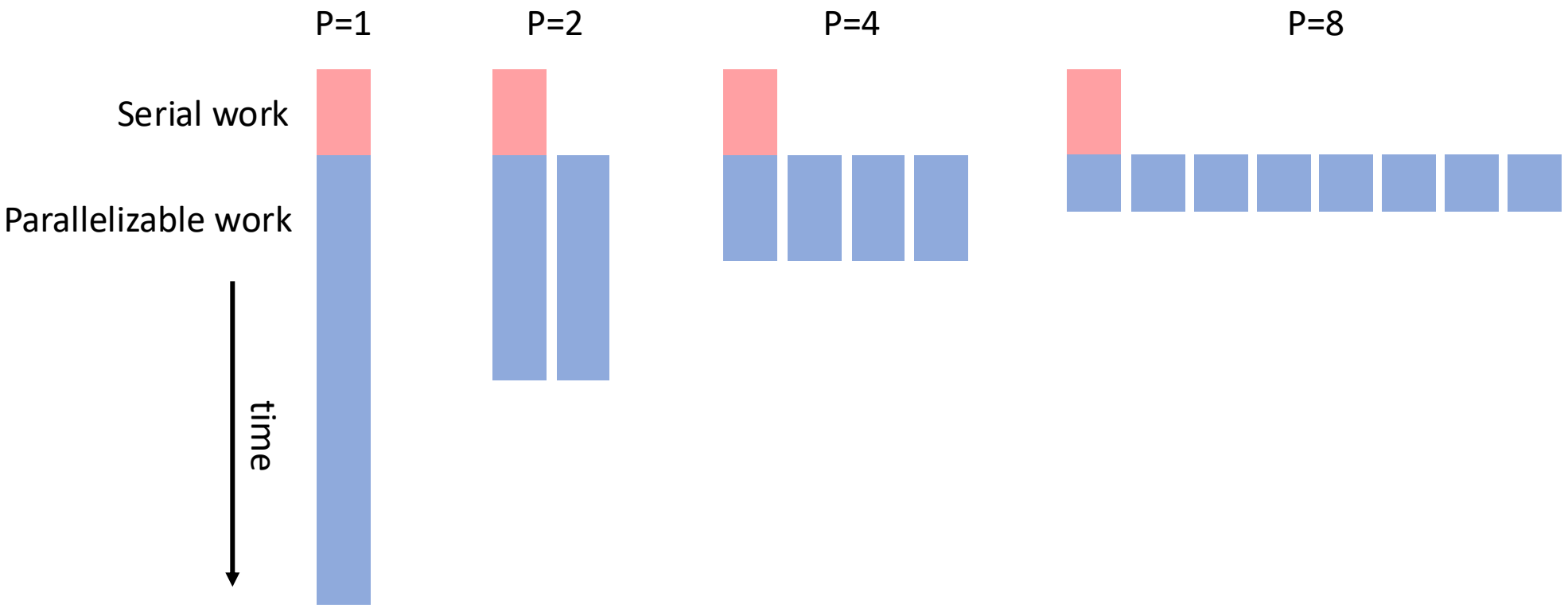
which gives:

$$S_p \leq \frac{1}{\boldsymbol{f} + \frac{1 - \boldsymbol{f}}{P}}$$

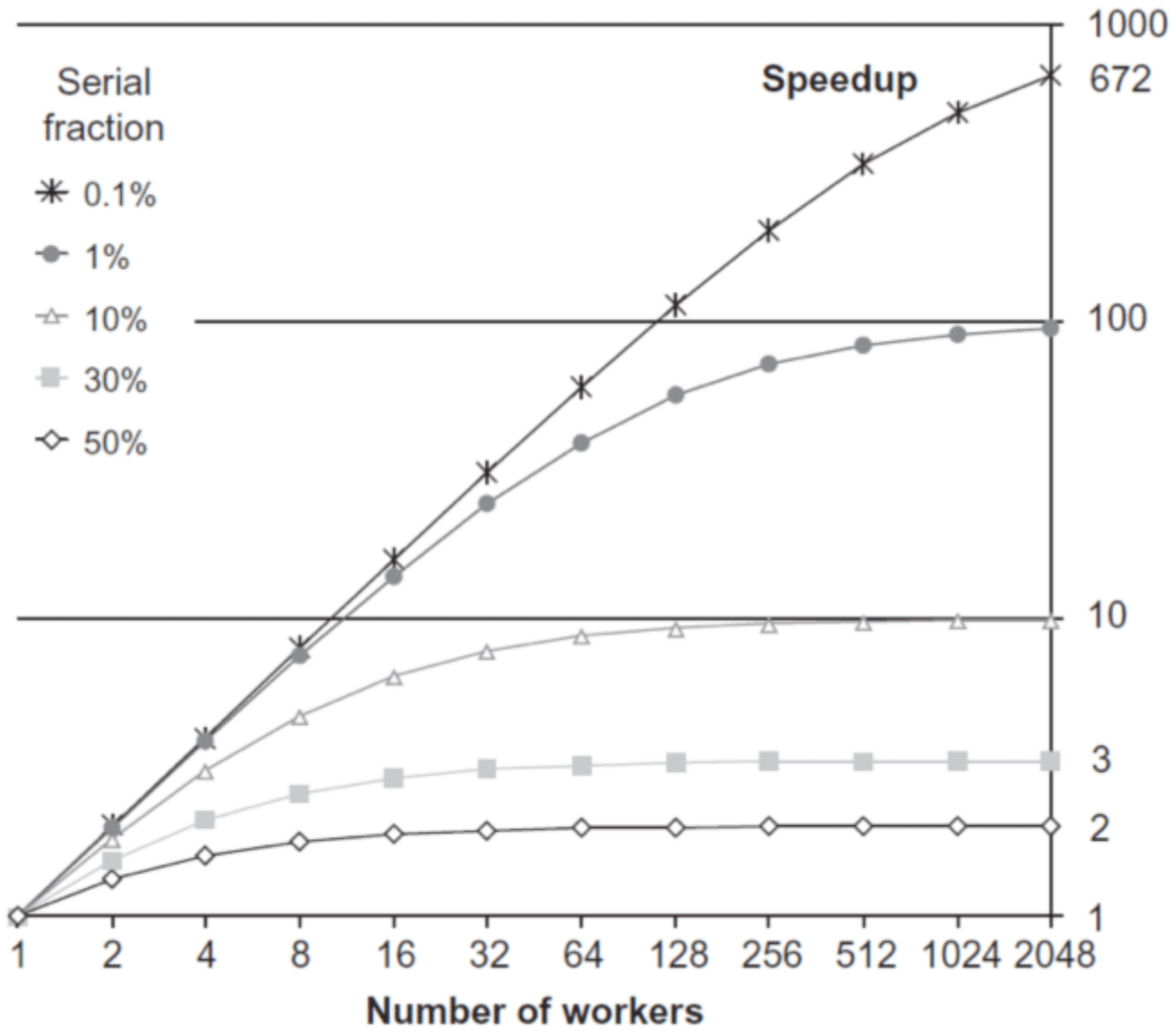
What happens if we have infinite workers?

$$S_\infty \leq \frac{1}{f}$$

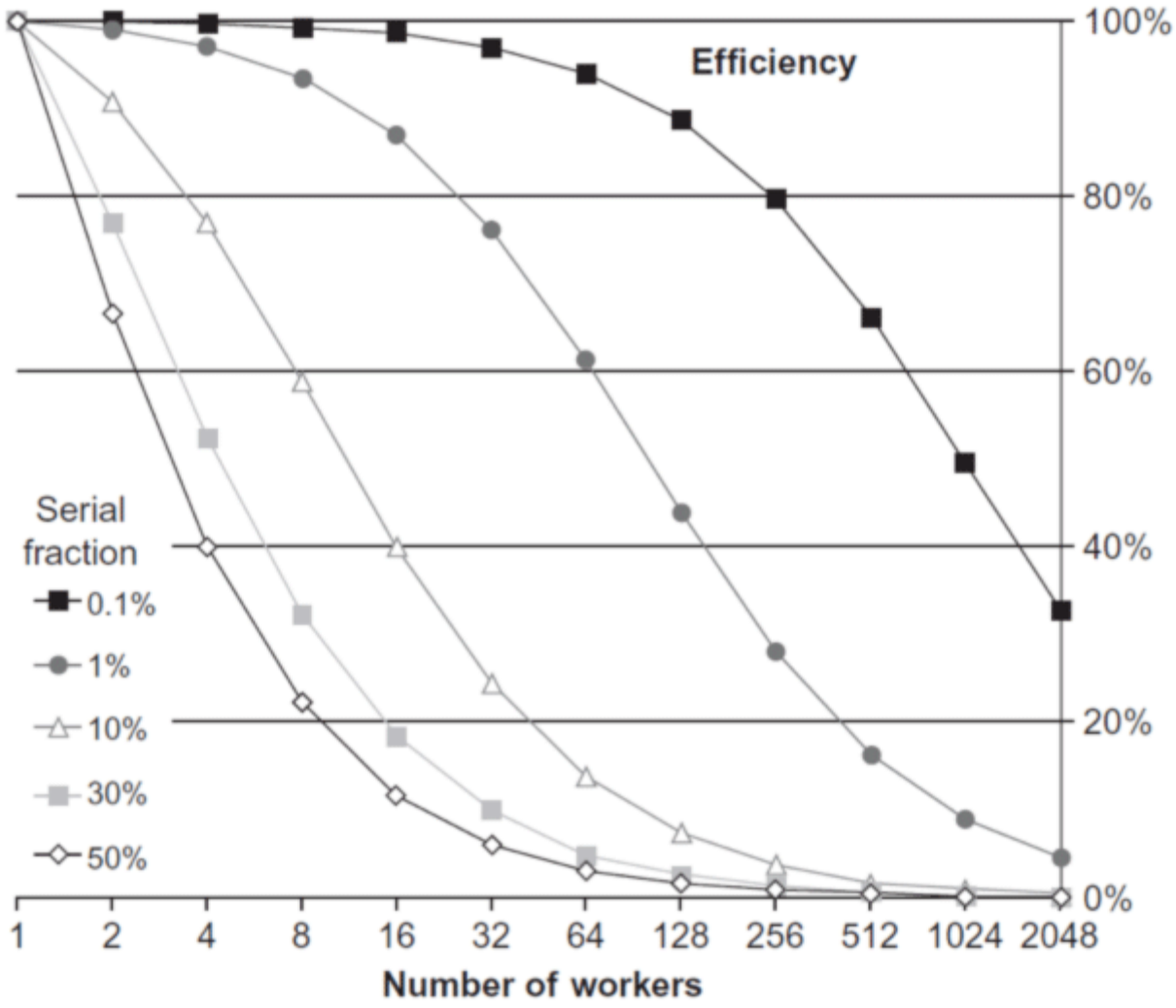
Amdahl's Law illustrated



Speedup


Speedup
Serial fraction
0.1%
1%
10%
30%
50%
1000
672
100
10
3
2
1
1
2
4
8
16
32
64
128
256
512
1024
2048
Number of workers

Efficiency


Efficiency
100%
80%
60%
40%
20%
0%
Serial fraction
0.1%
1%
10%
30%
50%
1
2
4
8
16
32
64
128
256
512
1024
2048
Number of workers

Remarks about Amdahl's Law

- It concerns **maximum speedup** (Amdahl was an optimist or pessimist?)
  -> mostly bad news (as it puts a limit on scalability)

- All non-parallel parts of a program (no matter how small) can cause problems

- Amdahl's law shows that efforts required to further reduce the fraction of the code that is sequential may pay off in large performance gains (-> parallel algorithms)

- One has to carefully weigh the trade-off between increasing parallel resources and the actual performance gain

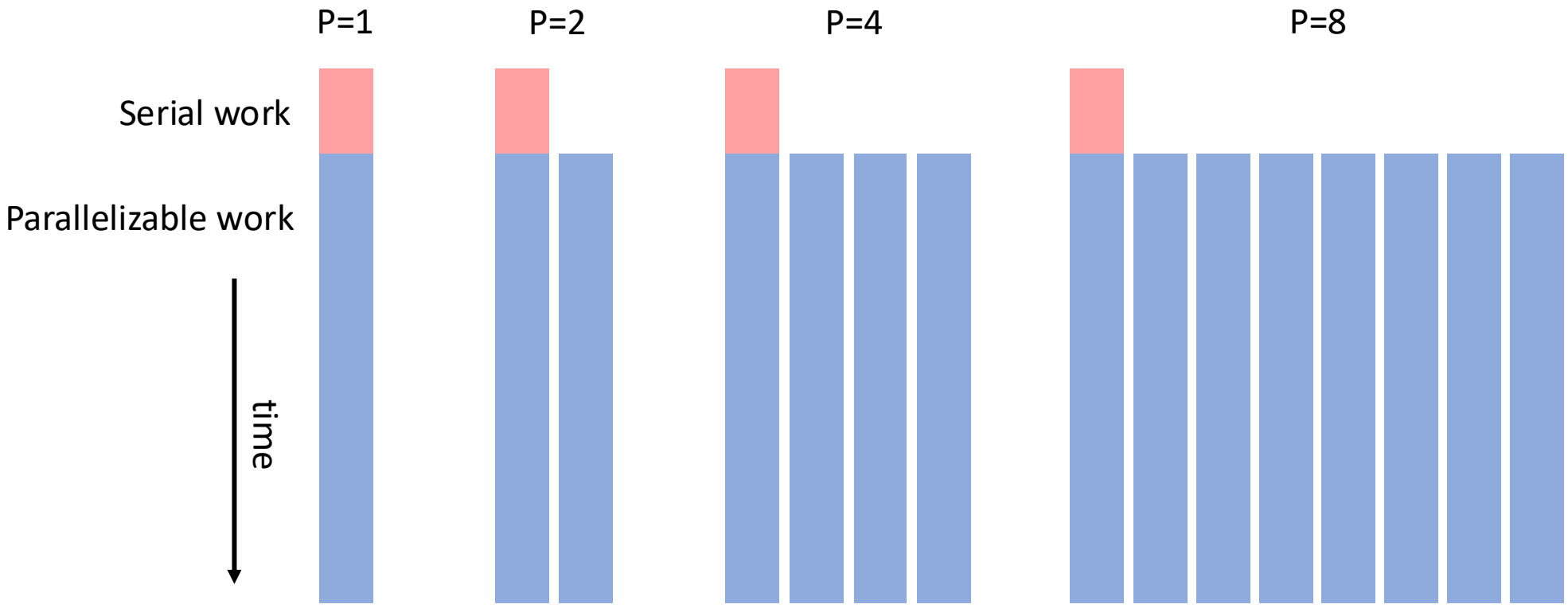
Gustafson’s Law

An alternative (optimistic) view to Amdahl's Law

Observations:

- Consider problem size
- Run-time, not problem size, is constant
- More processors allow to solve larger problems in the same time
- Parallel part of a program scales with the problem size

Gustafson's Law illustrated



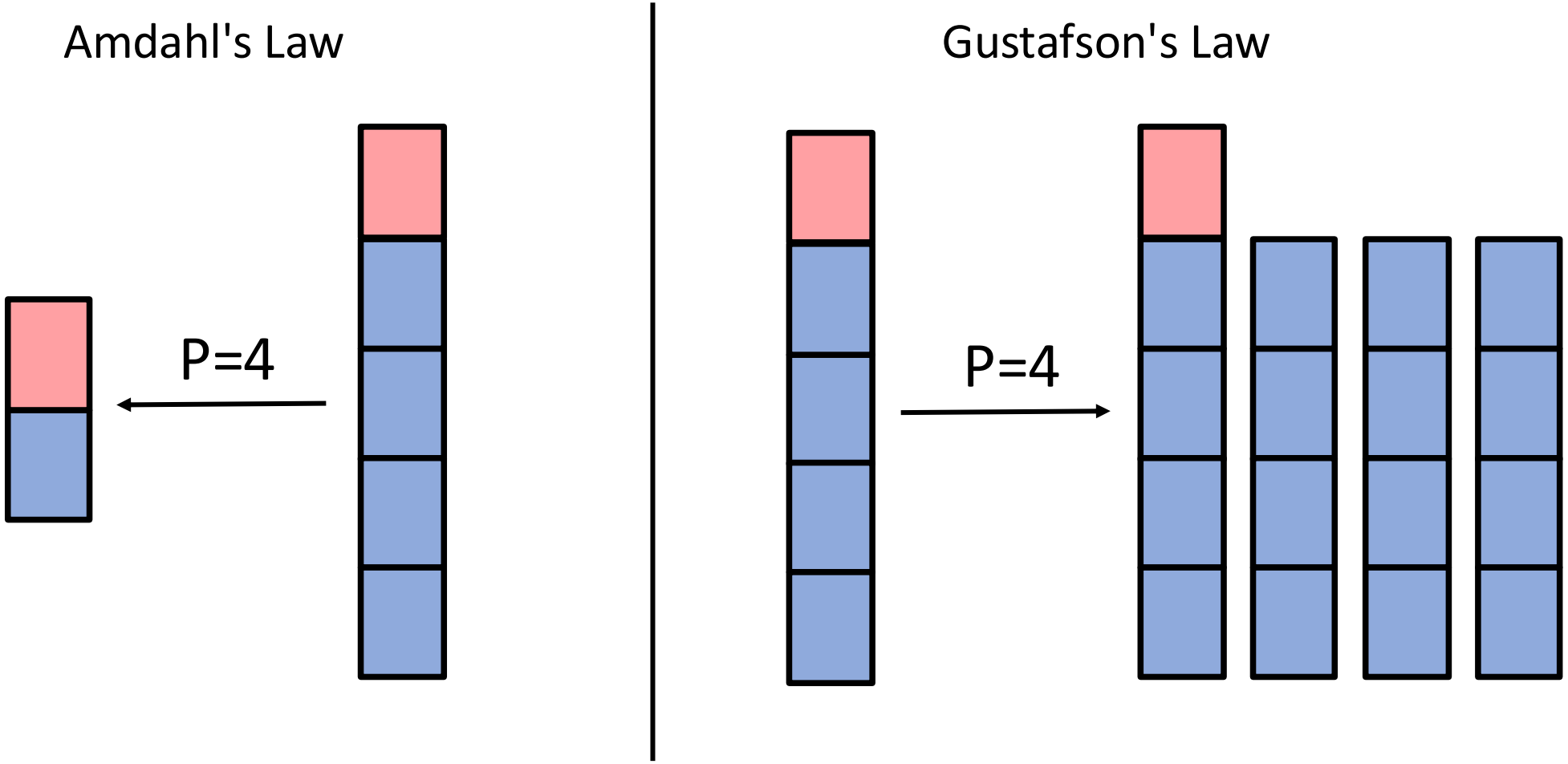
Gustafson's Law

f: sequential part (no speedup)

$$W = P(1-f)T_{wall} + fT_{wall}$$

$$S_p = f + P(1-f)$$
$$= P - f(P-1)$$

Amdahl's vs Gustafson's Law



Amdahl's vs Gustafson's Law

Amdahl's Law

Video game must run in realtime, make it faster!
(= same work but faster)

Gustafson's Law

It runs in realtime, but now we can add cool effects!
(= more work in same time)