CSE 6220/CX 4220
Introduction to HPC

Lecture 2: Design and Analysis of Parallel Algorithms

Helen Xu

Georgia Tech College of Computing
School of Computational Science and Engineering

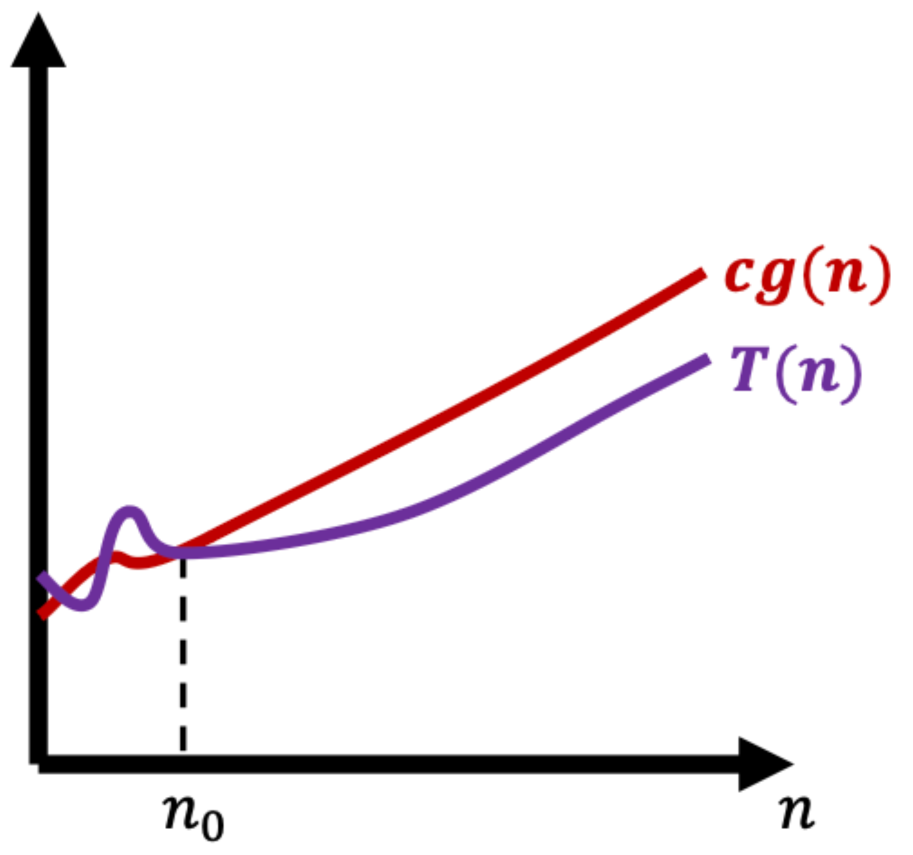
(Some slides from
Srinivas Aluru's/Ümit V. Çatalyürek's CSE 6220/CS4220,
MIT's OCW 6.172)

Parallel algorithm analysis measures

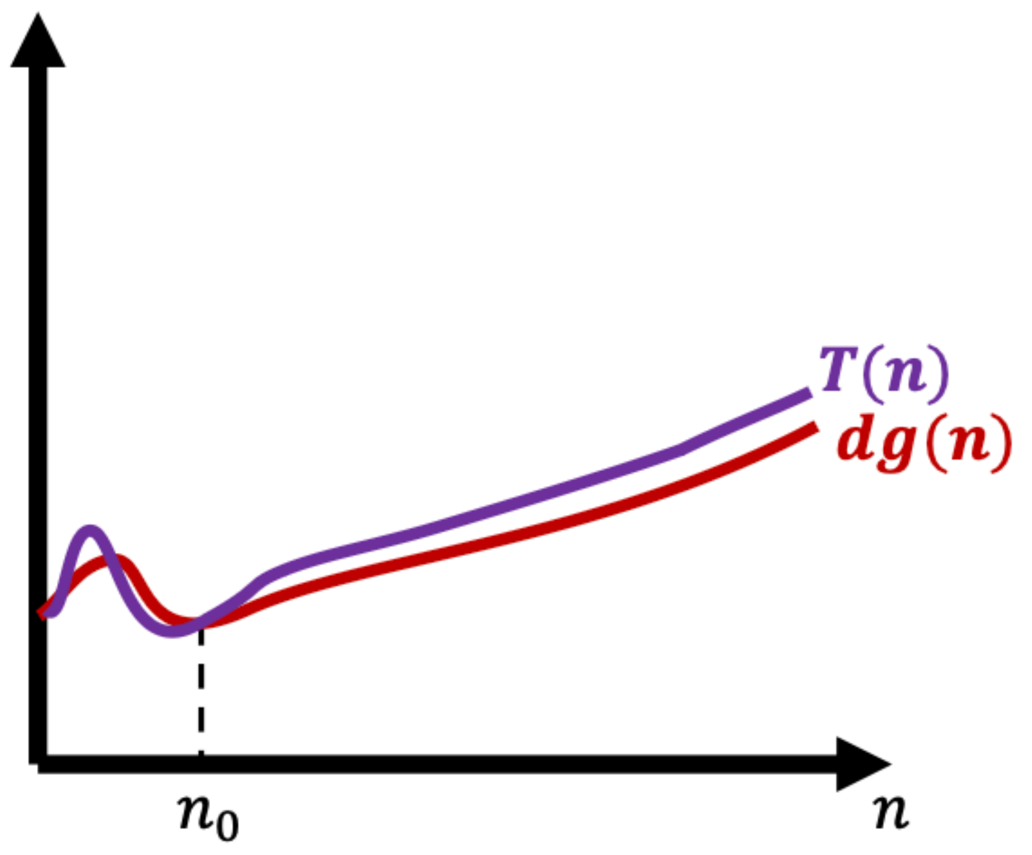
Review: Algorithm Performance Measures

Suppose we have a sequential algorithm that runs in time $T(n)$ where n is the problem size.

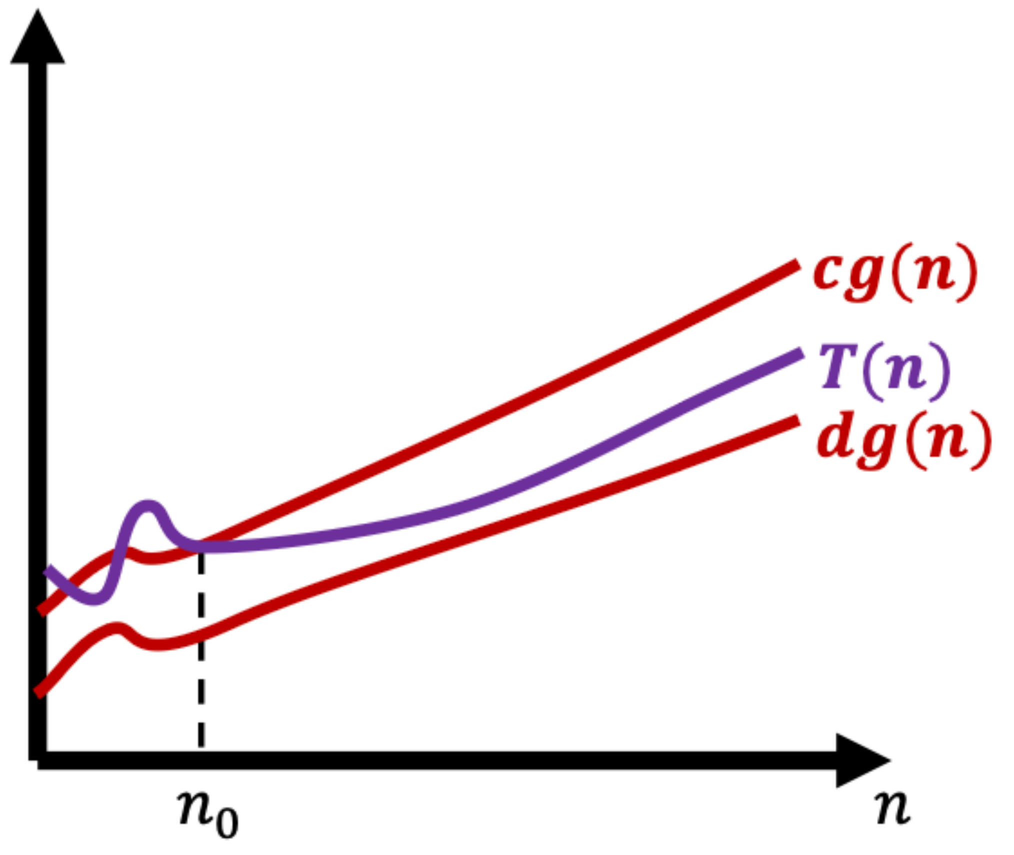
$T(n) = O(g(n))$



$T(n) = \Omega(g(n))$



$T(n) = \Theta(g(n))$

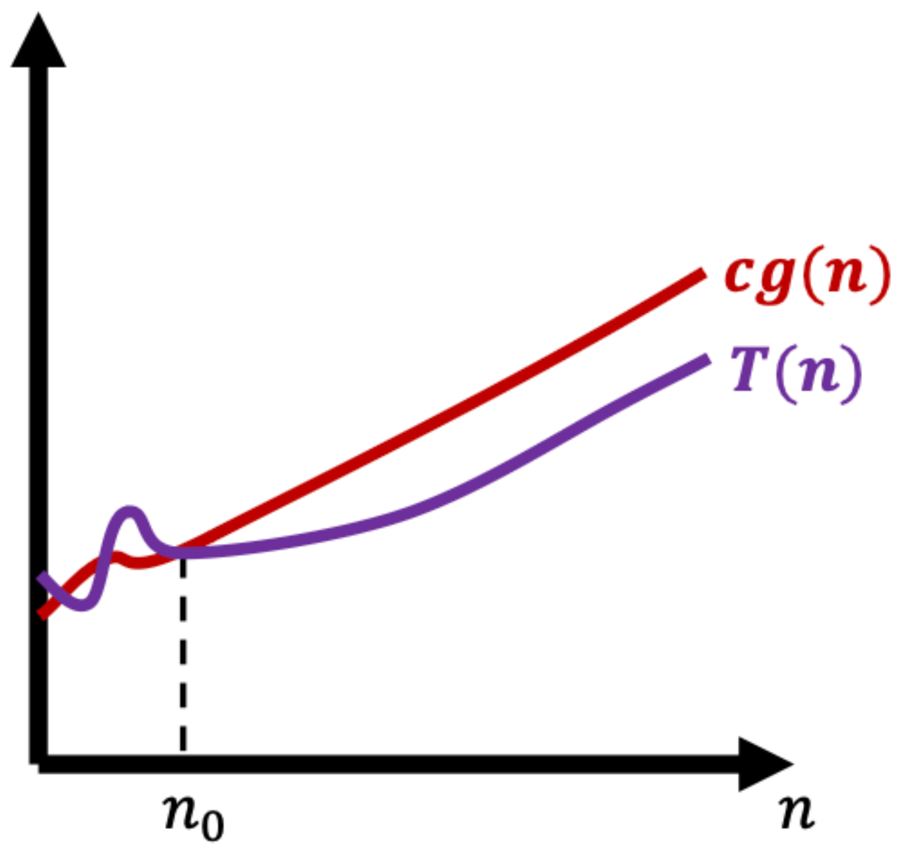


Review: Algorithm Performance Measures

Suppose we have a sequential algorithm that runs in time $T(n)$ where n is the problem size.

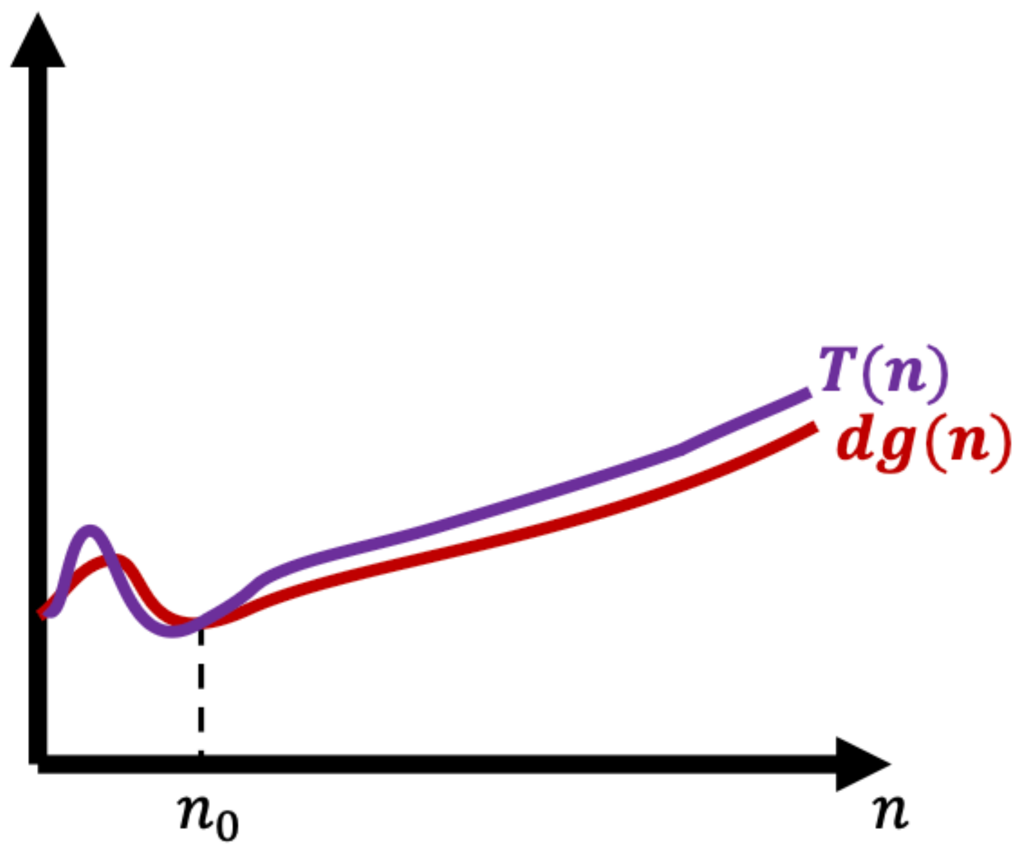
$\leq c \times g(n)$

$$T(n) = O(g(n))$$



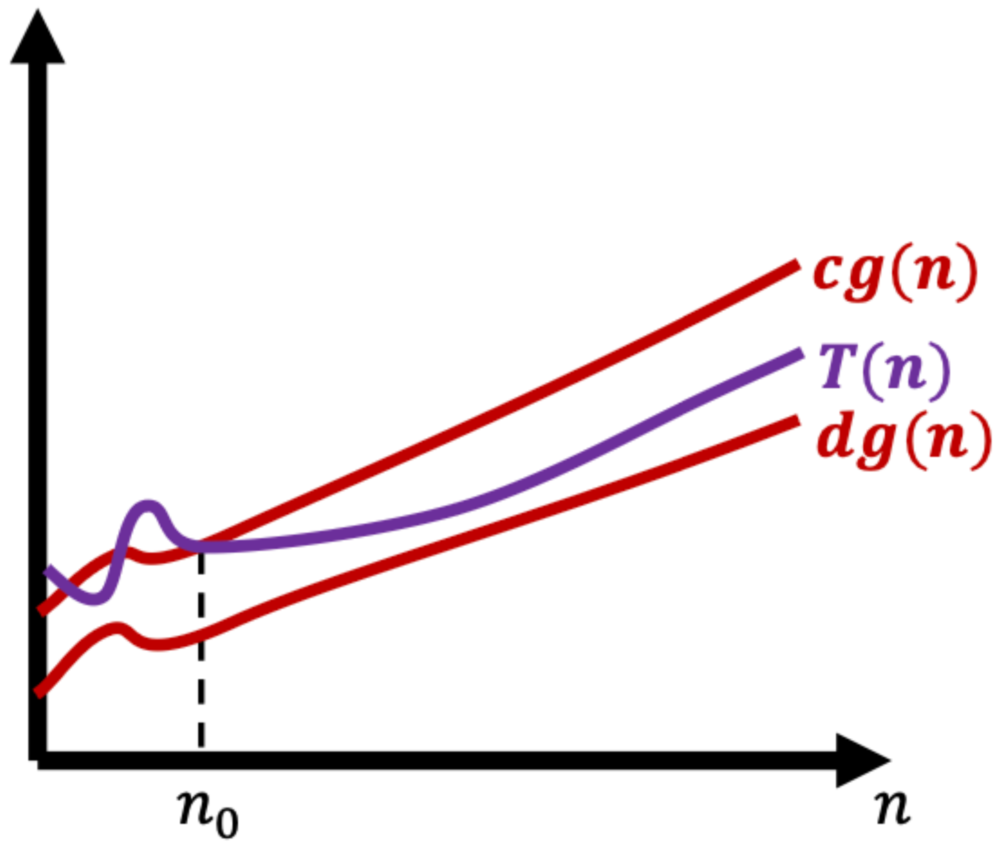
$\geq d \times g(n)$

$$T(n) = \Omega(g(n))$$

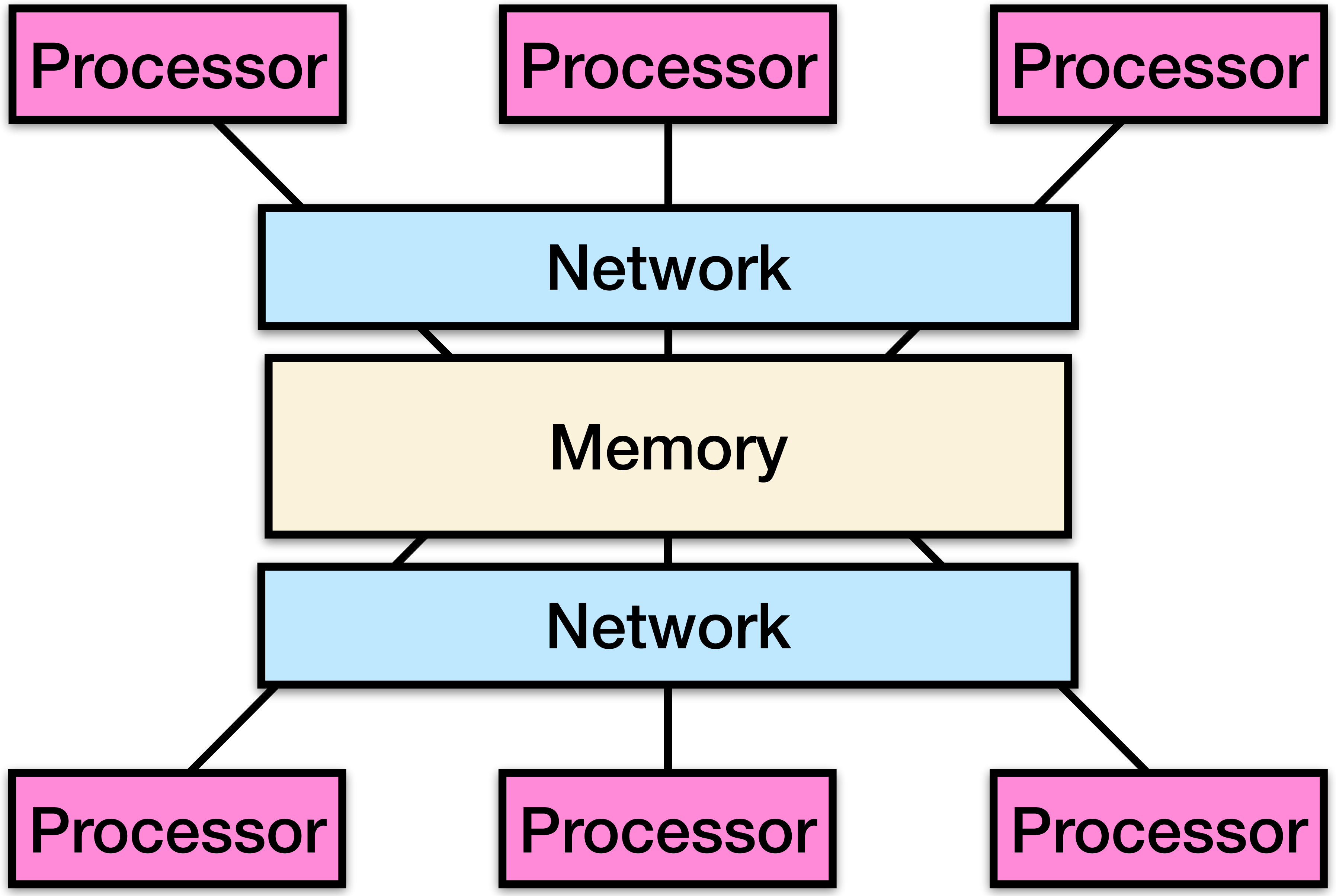


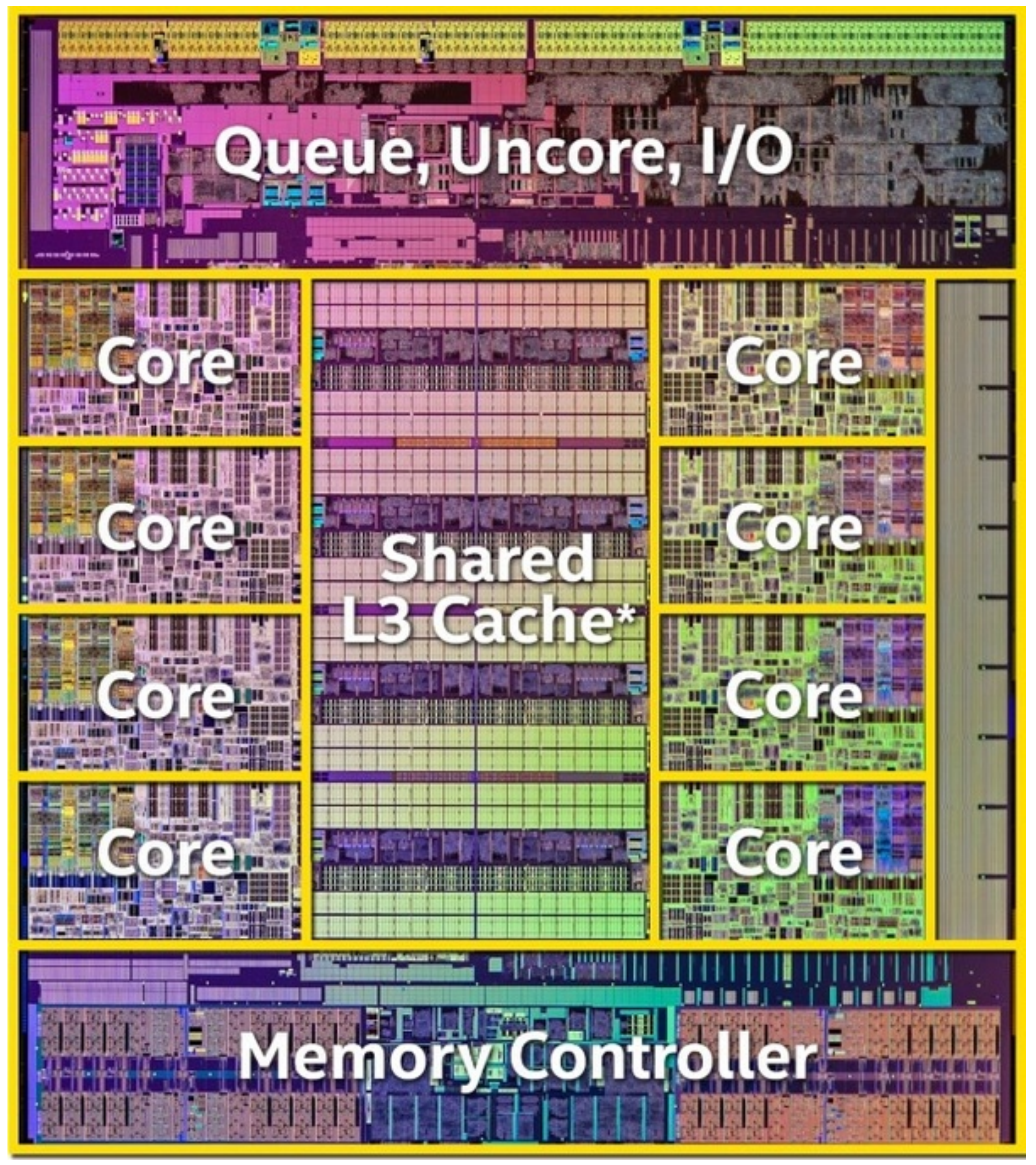
$O(g(n))$ and $\Omega(g(n))$

$$T(n) = \Theta(g(n))$$



Recall: Shared-memory parallel computers



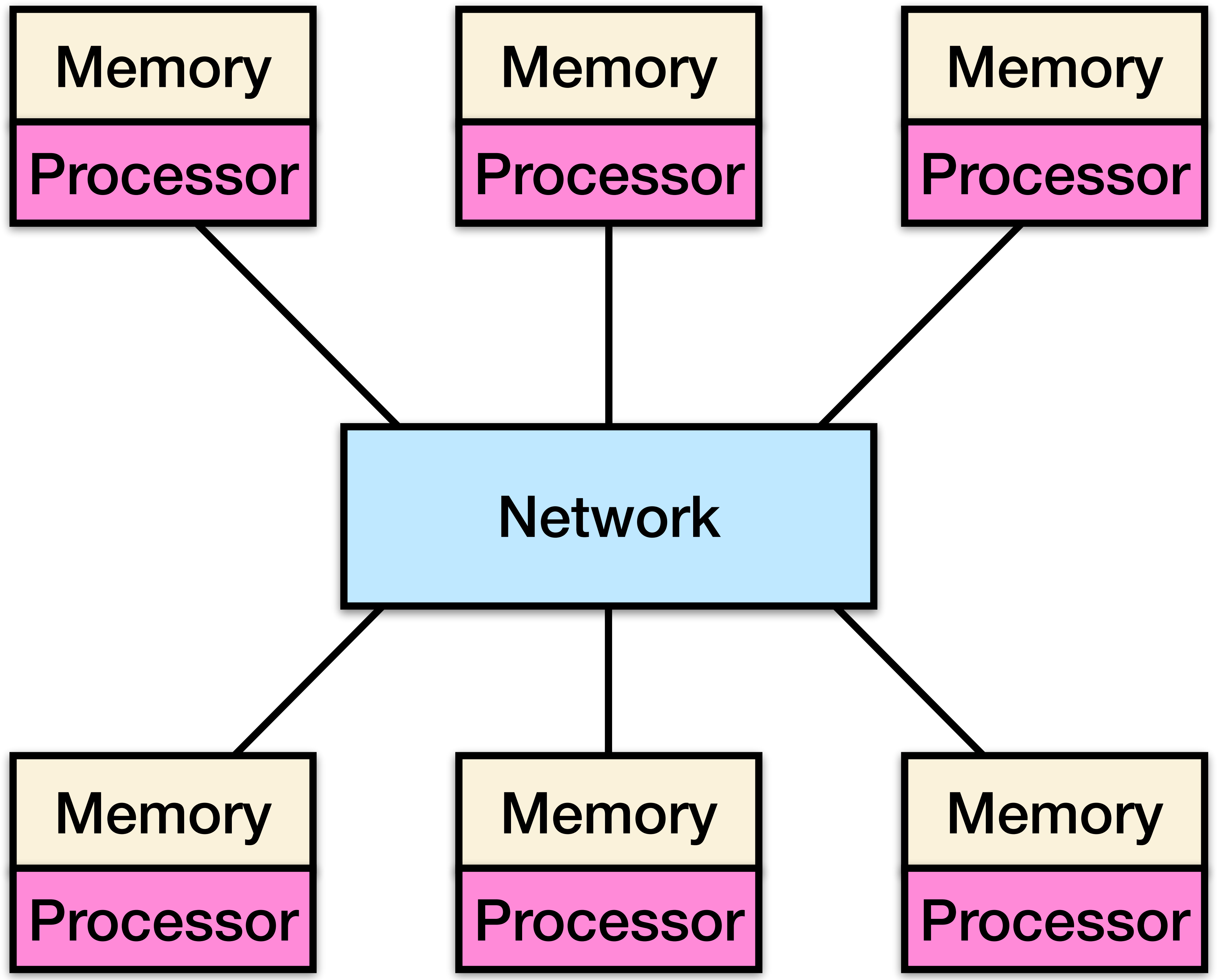


E.g. Intel Haswell

A **shared-memory multiprocessor** (SMP) connects multiple processors to a single memory system.

A multicore processor contains multiple processors (cores) on a single chip.

Recall: Distributed-memory parallel computers



A **distributed-memory multiprocessor** has processors with their own memories connected by a high-speed network.

Also called a **cluster**.

A **high-performance computing** (HPC) system contains 100s or 1000s of such processors (nodes).

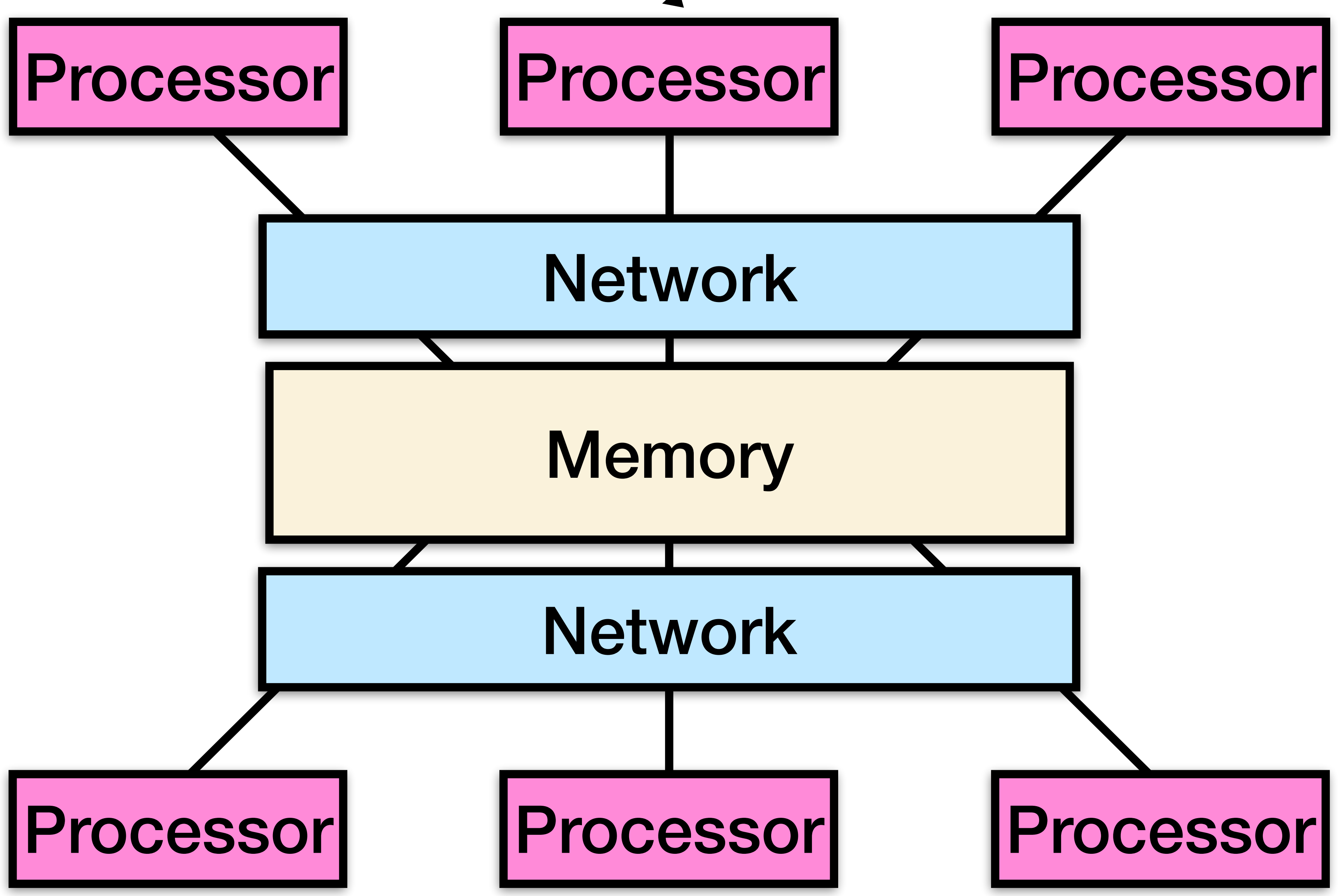
From UC Berkeley CS267

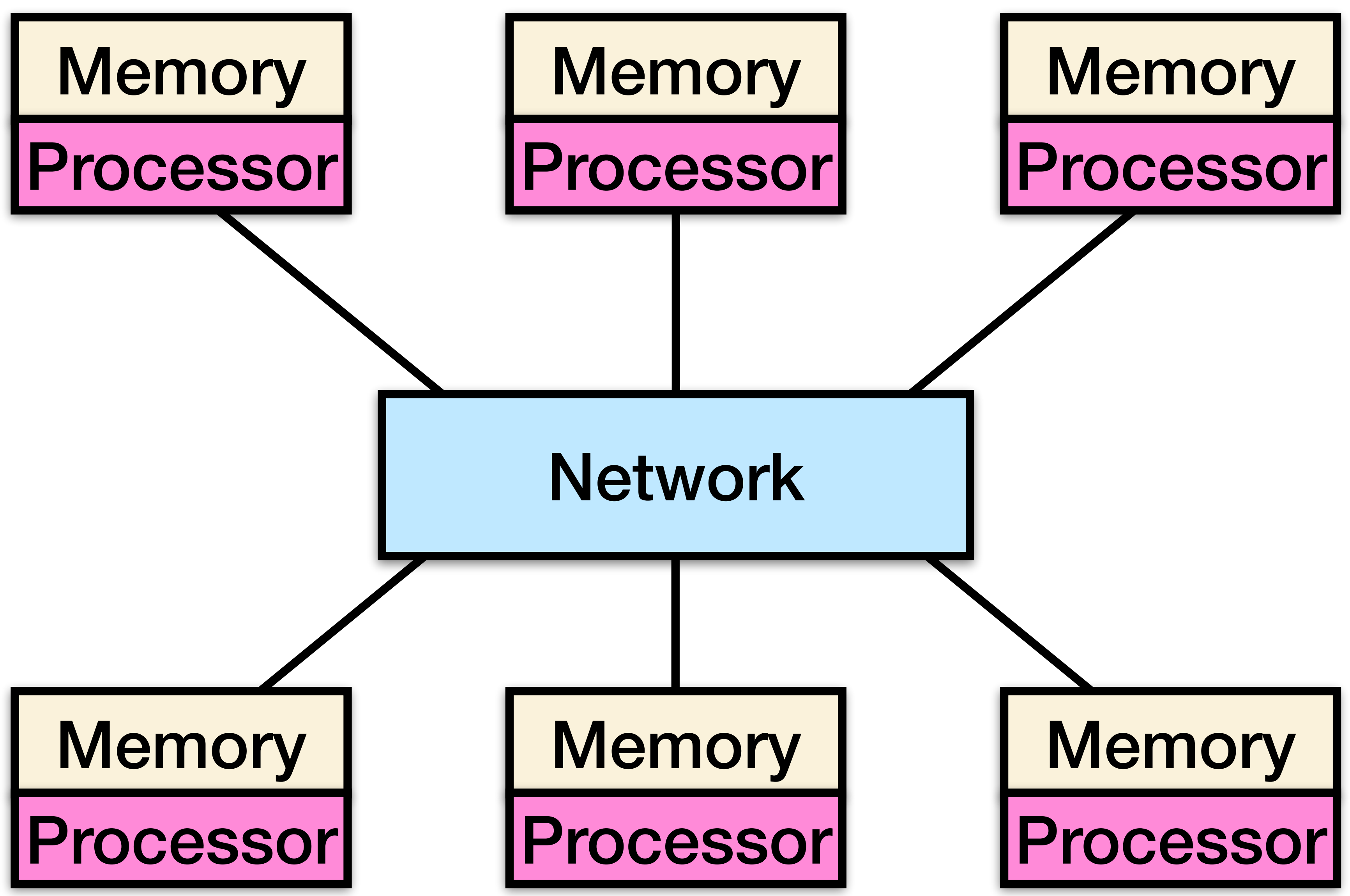
Parallel Computing Measures

Parallel runtime can be expressed with $T(n, p)$, where n is the problem size and $p > 1$ is the number of processors.

Can different algorithms perform better for different p?

$p = 6$



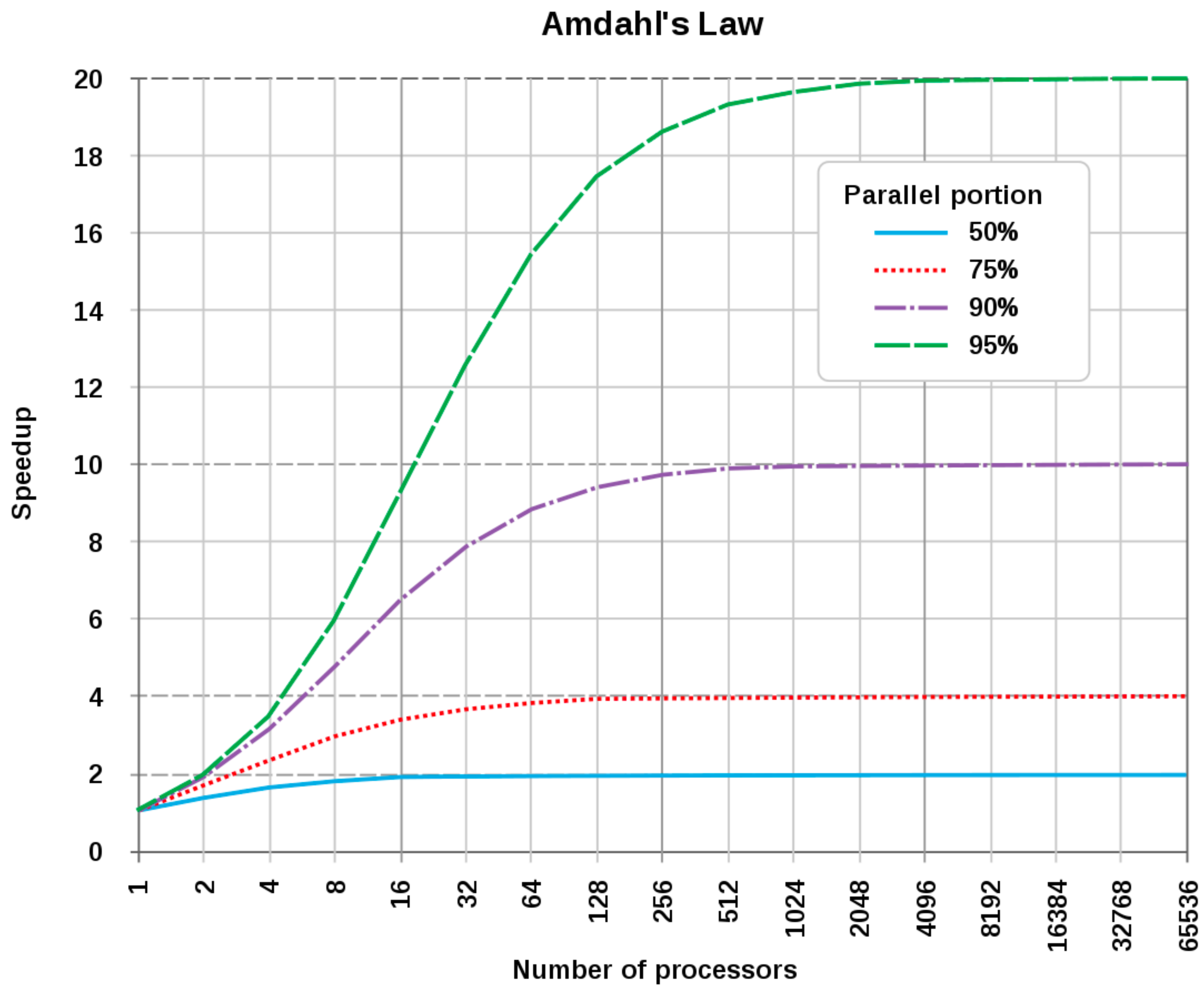


Parallel speedup

$$\text{Speedup} = \frac{\textbf{Runtime} \text{ of the best sequential algorithm}}{\textbf{Runtime} \text{ of the parallel algorithm}}$$

Speedup on p processors

$$S(p) = \frac{T(n,1)}{T(n,p)}$$



Lemma 1: $S(p) \leq p$

Speedup on p processors is bounded above by p

Proof: By contradiction

Suppose $S(p) > p$

so $\dfrac{T(n,1)}{T(n,p)} > p \implies T(n,1) > p \cdot T(n,p)$

Lemma 1: $S(p) \leq p$

Speedup on p processors is bounded above by p

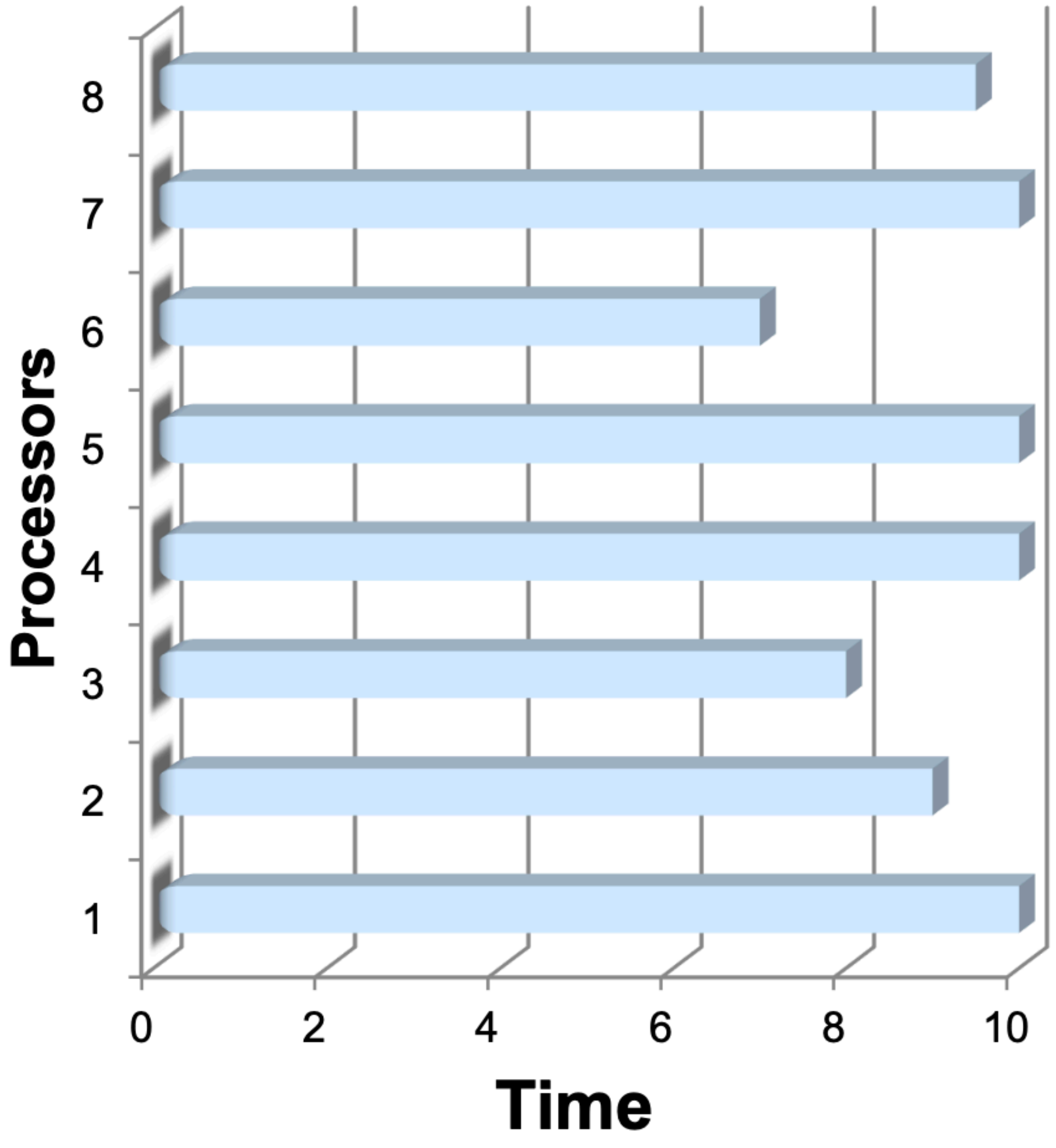
Using this very fast parallel algorithm, design a new sequential alg. by **simulating** the actions of processors in the parallel alg. using a single processor.

Runtime of the new sequential algorithm:

$$T(n,1) \leq p \cdot T(n,p)$$

But this contradicts with:

$$T(n,1) > p \cdot T(n,p)$$



Theoretical vs practical speedup

We just proved that the maximum theoretical speedup we can get on P processors is P.

In reality, there cases when we might achieve **superlinear speedup**, or more than P speedup.

One possible reason is that **more cache memory** is available when running on multiple processors.

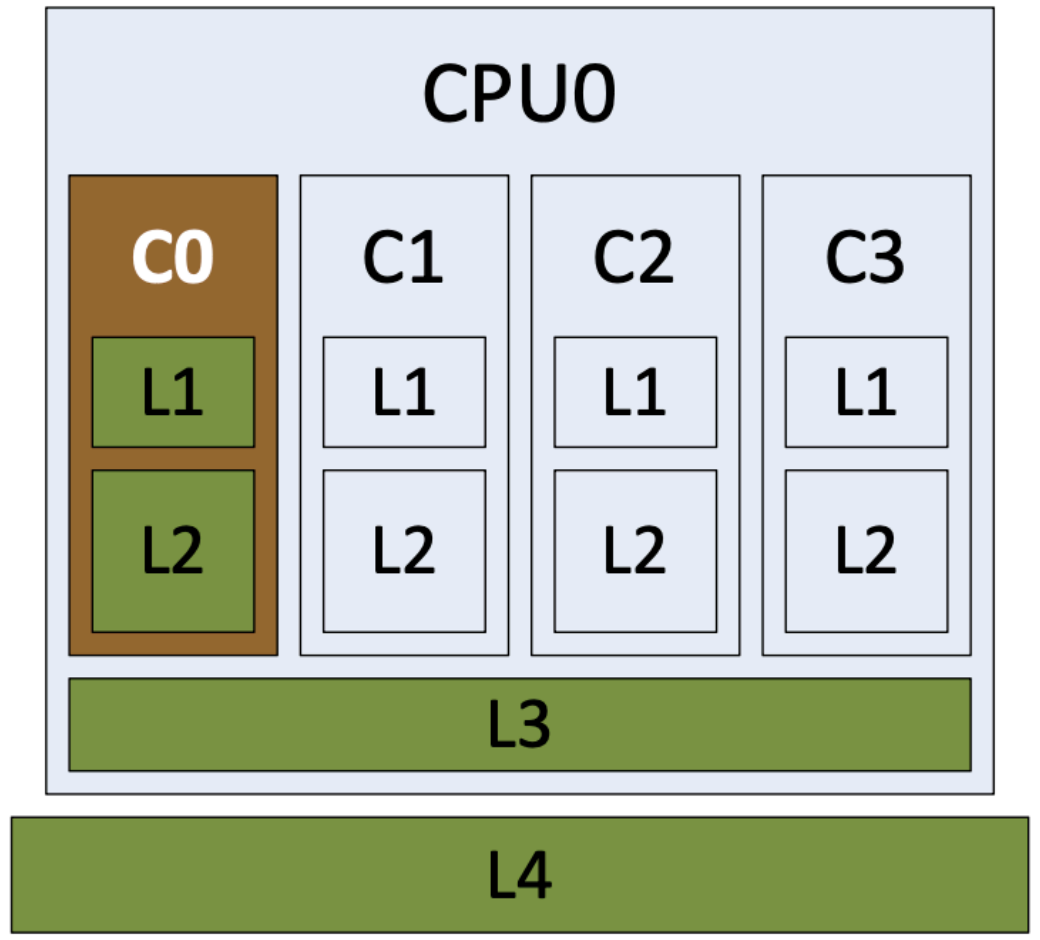


Fig. 3. Utilized memory for sequential execution

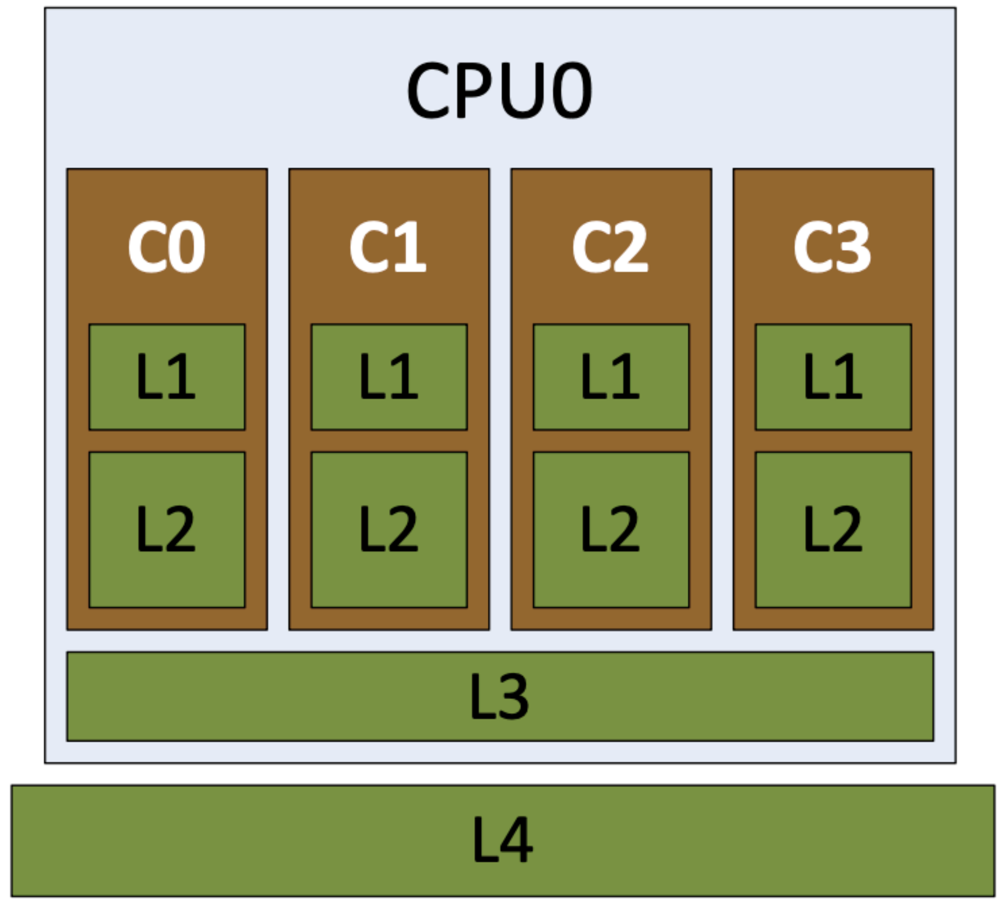


Fig. 4. Utilized multi tiered memory for loosely coupled processors for parallel execution

From “Superlinear Speedup in HPC Systems: why and when?” - Ristov et al. 2016

Parallel efficiency

Efficiency is a measure of how **effectively** resources of a parallel computer are utilized.

A **work-efficient parallel algorithm** performs no more than a constant factor of extra work compared to the best serial algorithm for the problem.

Theoretically measured in ops

$$\text{Efficiency} = \frac{\textbf{Work} \text{ done by the best sequential algorithm}}{\textbf{Work} \text{ done by the parallel algorithm}}$$

Speedup on p processors

$$E(p) = \frac{T(n,1)}{p \cdot T(n,p)} = \frac{S(p)}{p} \leq 1$$

Example: Matrix-matrix multiplication

For simplicity, let's assume the matrices are square:

```
void Mult(double *C, double *A, double *B, int64_t n) {
  for (int64_t i=0; i < n; i++)
    for (int64_t j=0; j < n; j++)
      for (int64_t k=0; k < n; k++)
        C[i*n+j] += A[i*n+k] * B[k*n+j];
}
```

What is the serial work of this algorithm?

Example: Matrix-matrix multiplication

For simplicity, let's assume the matrices are square:

```
void Mult(double *C, double *A, double *B, int64_t n) {
  for (int64_t i=0; i < n; i++)
    for (int64_t j=0; j < n; j++)
      for (int64_t k=0; k < n; k++)
        C[i*n+j] += A[i*n+k] * B[k*n+j];
}
```

What is the serial work of this algorithm?

$$T(n,1) = \Theta(n^3)$$

Example: Speedup vs efficiency

Suppose we have two parallel algorithms for matrix multiplication -

Parallel alg 1: $T(n, n^3) = \Theta(\log n)$

Parallel alg 2: $T(n, n^2) = \Theta(n)$

Number of processors

Speedup vs efficiency

Reminder:
$T(n,1) = \Theta(n^3)$

Speedup

Efficiency

What is the speedup?
Speedup = serial time / parallel time

Alg 1:
$T(n, n^3) = \Theta(\log n)$

Why is it important that this is Θ?

Alg 2:
$T(n, n^2) = \Theta(n)$

Reminder:
$T(n,1) = \Theta(n^3)$

Speedup vs efficiency

Speedup

Efficiency

What is the speedup?
Speedup = serial time / parallel time

Alg 1:
$T(n, n^3) = \Theta(\log n)$

Why is it important that this is Θ?

Alg 2:
$T(n, n^2) = \Theta(n)$

$$f(n) = O(s(n)) \land g(n) = O(r(n)) \not\Rightarrow \frac{f(n)}{g(n)} = O\left(\frac{s(n)}{r(n)}\right)$$

Speedup vs efficiency

Reminder:
$T(n,1) = \Theta(n^3)$

Speedup

Efficiency

What is the efficiency?
Efficiency = speedup / p

Alg 1:
$T(n, n^3) = \Theta(\log n)$

$$\Theta\left(\frac{n^3}{\log n}\right)$$

Alg 2:
$T(n, n^2) = \Theta(n)$

Speedup vs efficiency

Reminder:
$T(n,1) = \Theta(n^3)$

	Speedup	Efficiency
Alg 1: $T(n, n^3) = \Theta(\log n)$	$\Theta\left(\dfrac{n^3}{\log n}\right)$	$\Theta\left(\dfrac{n^3}{n^3 \log n}\right) = \Theta\left(\dfrac{1}{\log n}\right)$
Alg 2: $T(n, n^2) = \Theta(n)$		

What about for alg 2?

Reminder:
$T(n,1) = \Theta(n^3)$

Speedup vs efficiency

	Speedup	Efficiency
Alg 1: $T(n, n^3) = \Theta(\log n)$	$\Theta\left(\frac{n^3}{\log n}\right)$	$\Theta\left(\frac{n^3}{n^3 \log n}\right) = \Theta\left(\frac{1}{\log n}\right)$
Alg 2: $T(n, n^2) = \Theta(n)$	$\Theta(n^2)$	$\Theta(1)$

Speedup vs efficiency

Reminder:
$T(n,1) = \Theta(n^3)$

	Speedup	Efficiency
Alg 1: $T(n, n^3) = \Theta(\log n)$	💚 $\Theta\left(\frac{n^3}{\log n}\right)$	$\Theta\left(\frac{n^3}{n^3 \log n}\right) = \Theta\left(\frac{1}{\log n}\right)$
Alg 2: $T(n, n^2) = \Theta(n)$	$\Theta(n^2)$	💚 $\Theta(1)$

Which of speedup or efficiency should we aim for?

Lemma 2: Brent's Lemma

Let $T(n, p_1)$ be the runtime of a parallel algorithm designed to run on p_1 processors.

The same algorithm can be run on $p_2 < p_1$ processors **without loss of efficiency**: i.e., $E(p_2) = E(p_1)$.

Sometimes called efficiency scaling

Lemma 2: Brent's Lemma

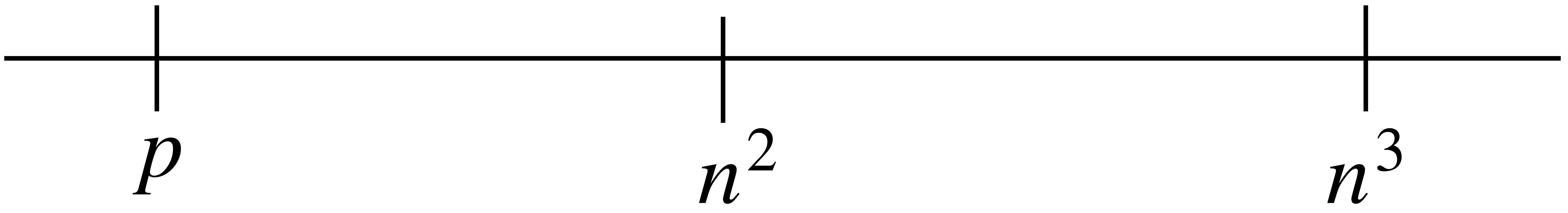
Proof: To run on p_2 processors, let each processor simulate $\dfrac{p_1}{p_2}$ processors.

If they don't perfectly divide, take the ceiling. You lose at most a factor of 2 (proof omitted).

$$T(n,p_2) = \frac{p_1}{p_2} T(n,p_1)$$

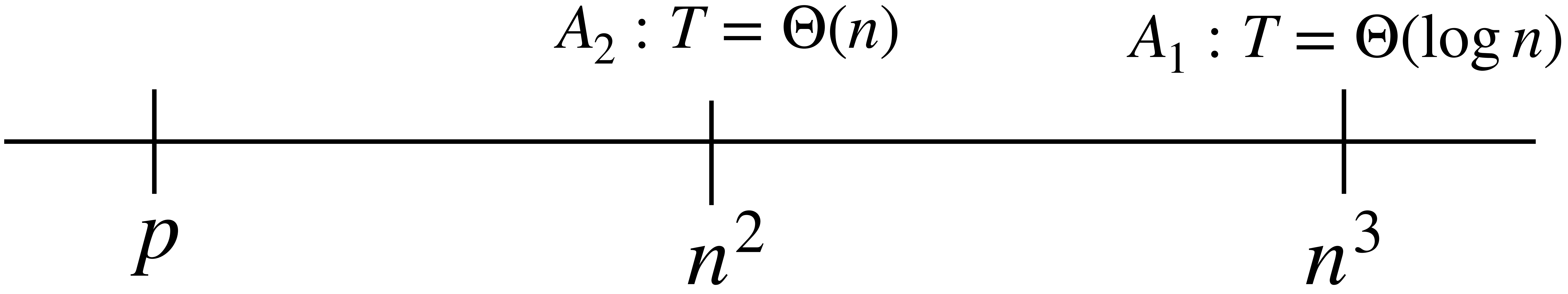
$$E(p_2) = \frac{T(n,1)}{p_2 \cdot T(n,p_2)} = \frac{T(n,1)}{p_2 \cdot \frac{p_1}{p_2} T(n,p_1)} = E(p_1)$$

Speedup vs efficiency



Number of processors is usually much less than the problem size

Speedup vs efficiency



Number of processors is usually much less than the problem size

Speedup vs efficiency

Ratio of processors

$$\Theta\left(\frac{n^3}{p} \cdot \log n\right)$$

$A_2 : T = \Theta(n)$

$A_1 : T = \Theta(\log n)$

Total work is processor-time product

p $\quad n^2 \quad n^3$

Number of processors is usually much less than the problem size

Speedup vs efficiency

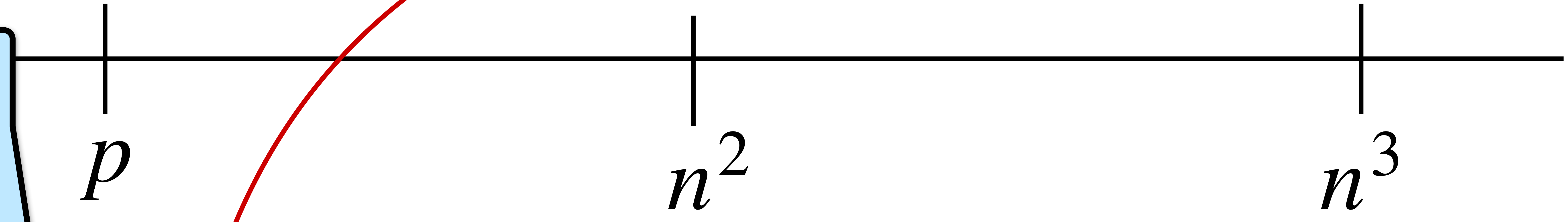
Ratio of processors

$$\Theta\left(\frac{n^3}{p} \cdot \log n\right)$$

Total work is processor-time product

$A_2 : T = \Theta(n)$

$A_1 : T = \Theta(\log n)$



Ratio of processors

$$\Theta\left(\frac{n^2}{p} \cdot n\right)$$ 💚

Number of processors is usually much less than the problem size

Goals of parallel algorithm design

Speed: Design alg. to **minimize** $T(n, p)$ using the smallest possible value p.

Efficiency: Design alg. to **maximize** $E(p)$ s.t. p is the largest possible.

To design fast parallel algorithms, target **efficiency**.

Efficiency $\Rightarrow$ Speedup

*on the range of processors that the efficient algorithm is guaranteed on

Quick check for efficiency

The work of an algorithm is the **processor-time product**.

Parallel alg 1: $T(n, n^3) = \Theta(\log n)$ 😡

Parallel alg 2: $T(n, n^2) = \Theta(n)$ 💚

Parallel alg 3: $T(n, n) = \Theta(n^2)$ 💚

How do you choose between two efficient algorithms?

Scalability: Matrix Multiplication Example

- Fixed-time scalability is one notion of scalability - when faced with more resources and a bigger problem, does the problem have the same runtime?

- are there things that more money / resources cannot fix?

Scalability: Matrix Multiplication Example

Ideal efficient algorithm:

$$T(n, n^3) = O(1), \leq n^3$$

$$T(n, p) = O\left(\frac{n^3}{p}\right)$$

If we increase the problem size by 2x, work increases 8x, so we try 8x processors

$$T(2n, 8p) = O\left(\frac{8n^3}{8p}\right) = O\left(\frac{n^3}{p}\right)$$

Scalability: Matrix Multiplication Example

Ideal efficient algorithm:

$$T(n, n^3) = O(1), \leq n^3$$

$$T(n, p) = O\left(\frac{n^3}{p}\right)$$

If we increase the problem size by 2x,
work increases 8x, so we try 8x processors

$$T(2n, 8p) = O\left(\frac{8n^3}{8p}\right) = O\left(\frac{n^3}{p}\right)$$

Inefficient algorithm:

$$T(n, n^3) = O(\log n)$$

$$T(n, p) = O\left(\frac{n^3 \log n}{p}\right)$$

If we increase the problem size by 2x,
work increases 8x, so we try 8x processors

$$T(2n, 8p) = O\left(\frac{8n^3 \log 2n}{8p}\right) = O\left(\frac{n^3 \log 2n}{p}\right)$$

Scalability example

What if we have two efficient algorithms?

A_2: $T(n, n^2) = \Theta(n)$ $\quad\quad p \leq n^2$

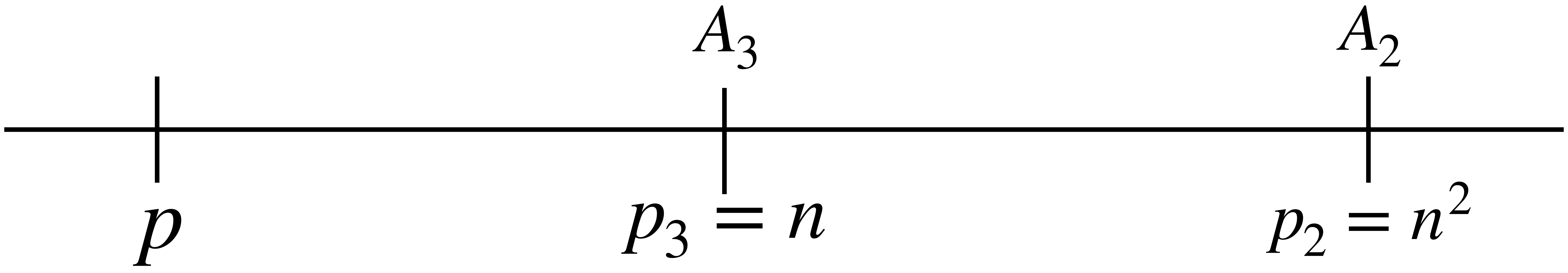
4x number of processors
-> 2x runtime

A_3: $T(n, n) = \Theta(n^2)$ $\quad\quad p \leq n$

2x number of processors
-> 4x runtime

In general, prefer the one that can use more processors

Scalability example

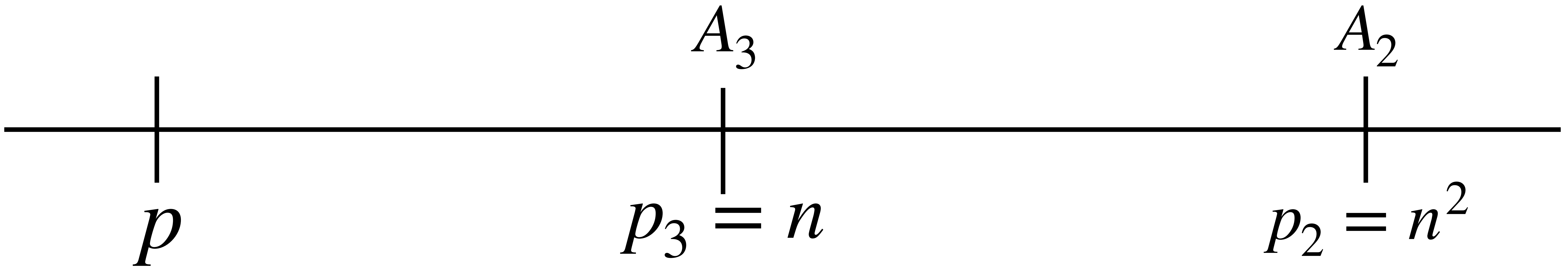


Suppose we are multiplying two 1000 x 1000 matrices.
Problem size is n = 1000, and suppose p = 100,

What are p_2 and p_3?

p_3= 1000, p_2 = 1000000

Scalability example

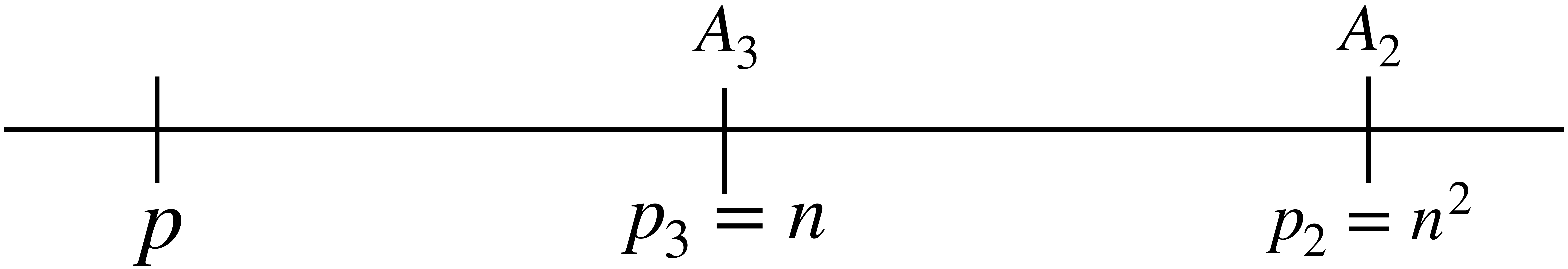


Now suppose the problem size doubles ($n = 2000$).

The work is increased 8x (matrix multiplication), let's use 8x processors to try to preserve the runtime.

$p = 800$, $p_3 = 2000$, and $p_2 = 4{,}000{,}000$, since $p < p_3$ and $p < p_2$, you can use either algorithm

Scalability example



If we double again?

$p = 6400$, $p_3 = 4000$, p_2 = 16000000, now $p > p_3$ so we cannot use A_3, so A_2 better!

Choose the algorithm that has a higher maximum number of processors that it can use

Summary of goals

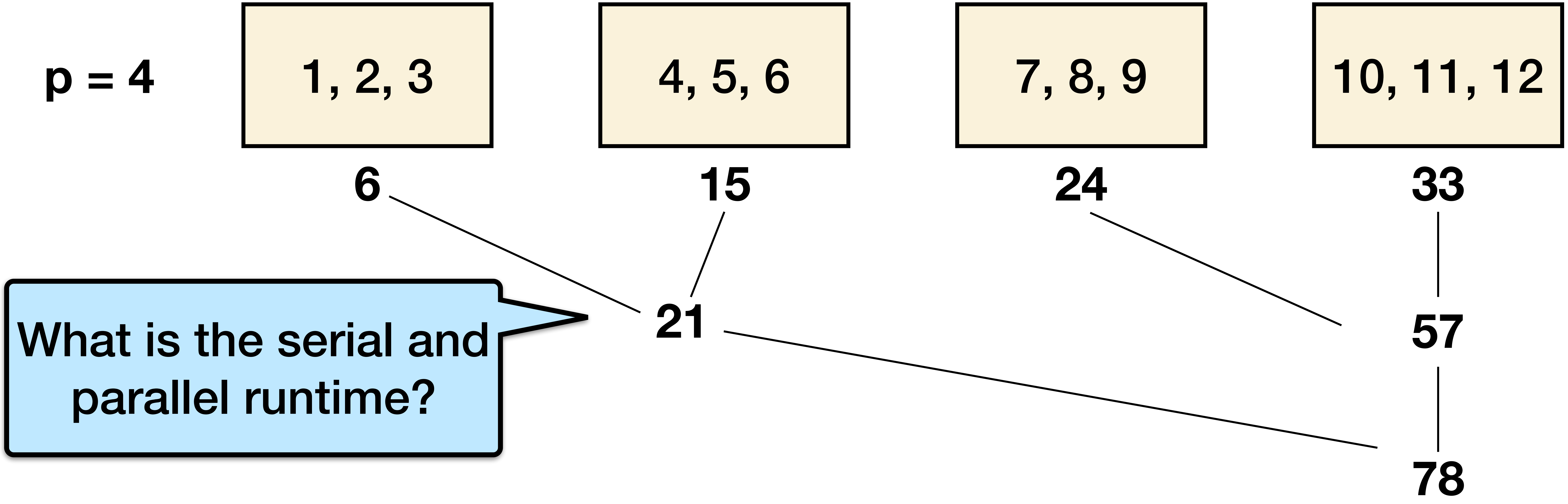
- Step 1 - Maximize efficiency
- Step 2 - Pick algorithm that uses max number of processors

Example: Parallel sum

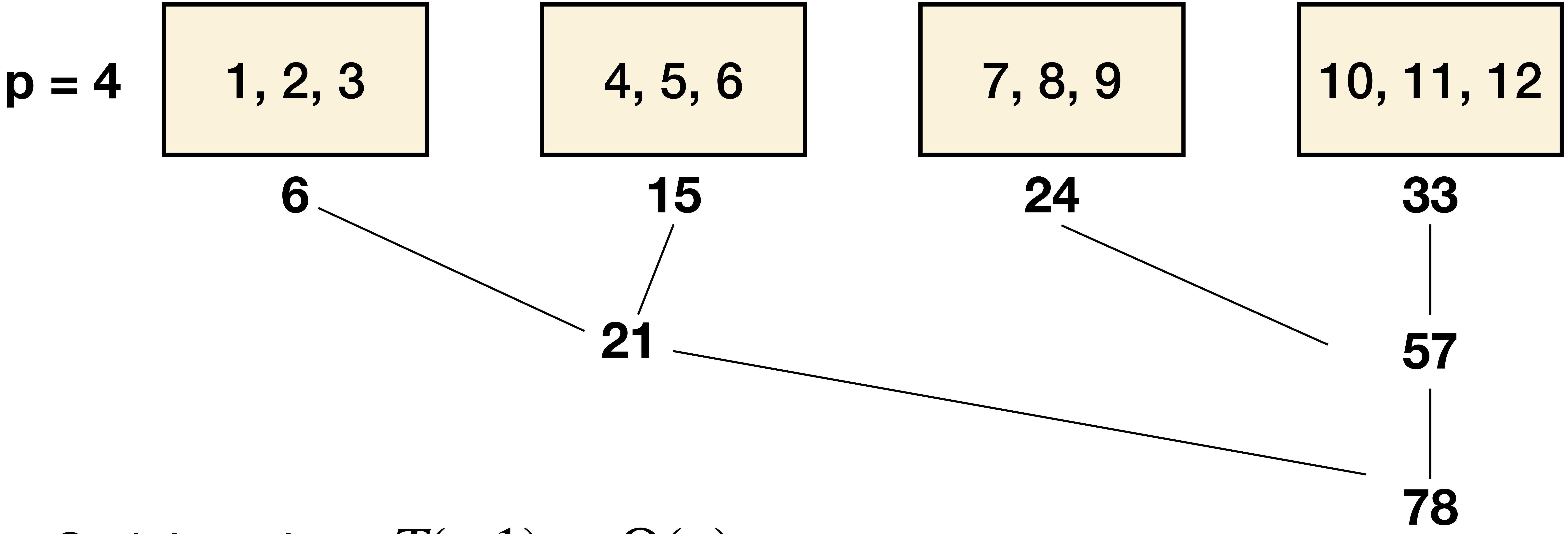
Given n numbers, compute their sum using p processors.

Step 1: Serial - Sum local n/p numbers

Step 2: Parallel - Add p numbers using p processors



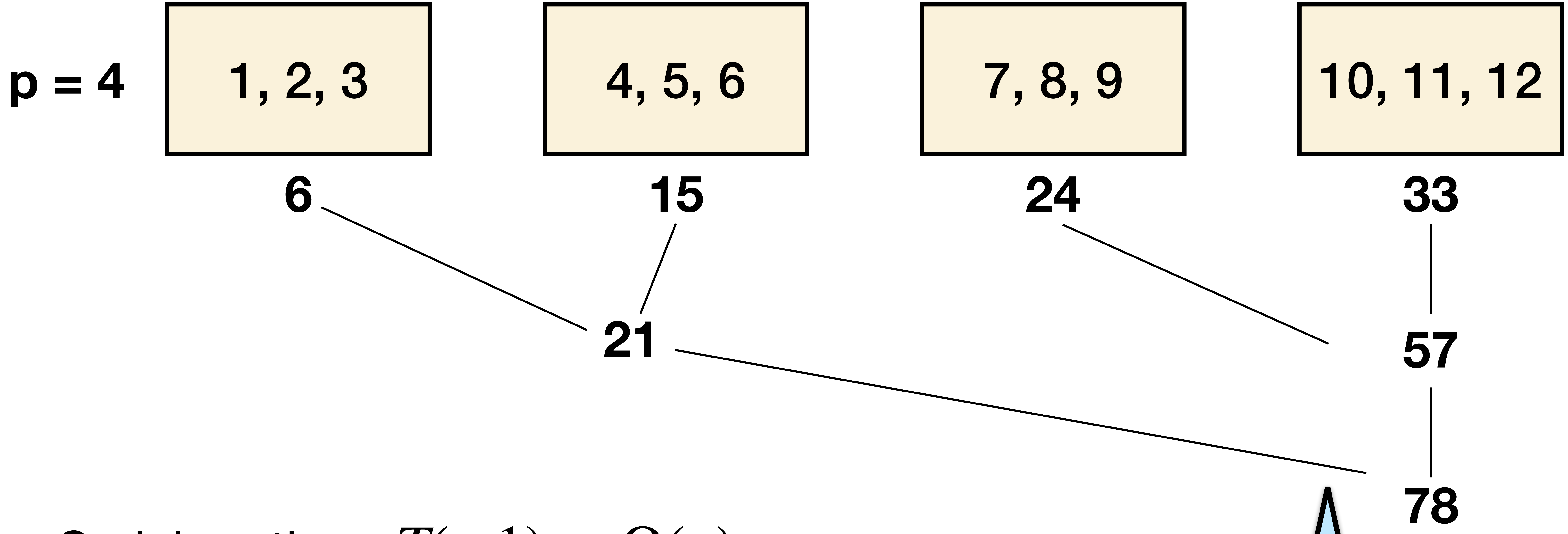
Example: Parallel sum



Serial runtime: $T(n,1) = \Theta(n)$

Example: Parallel sum

How long does it take to add up locally?

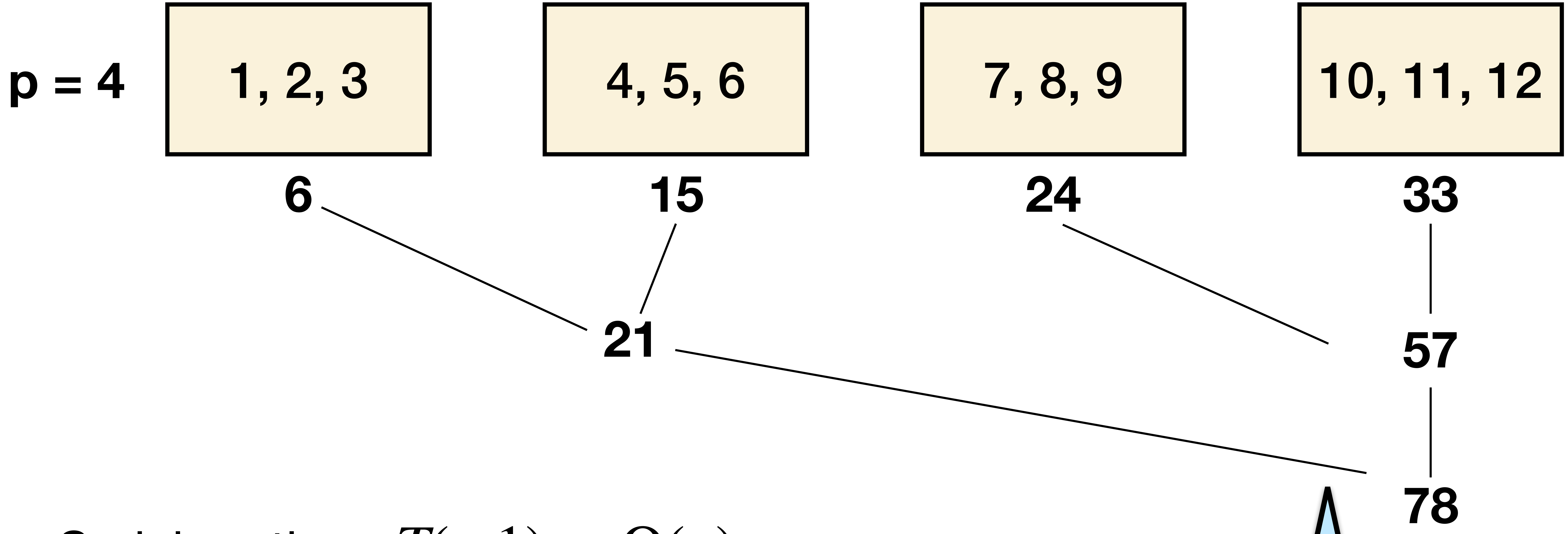


What is the height of this tree?

Serial runtime: $T(n,1) = \Theta(n)$

How long does it take to add up locally?

Example: Parallel sum



What is the height of this tree?

Serial runtime: $T(n,1) = \Theta(n)$

Parallel runtime: $T(n,p) = \Theta\left(\frac{n}{p} + \log p\right)$

Maximize speed in parallel sum

Find p such that $T(n,p)$ is minimized:

$$\frac{d}{dp}\left(\frac{n}{p} + \log p\right) = 0$$

$$-\frac{n}{p^2} + \frac{1}{p} = 0$$

$$\Rightarrow p = n$$

Maximize efficiency in parallel sum

What range of p can support $E(p) = \Theta(1)$?

$$E(p) = \frac{T(n,1)}{pT(n,p)} = \Theta(1)$$

$$E(p) = \frac{n}{p\left(\frac{n}{p} + \log p\right)} = \Theta(1)$$

$$\Rightarrow \frac{n}{n + p\log p} = \Theta(1) \Rightarrow p\log p = O(n)$$

Maximize efficiency in parallel sum

What range of p can support $E(p) = \Theta(1)$?

Prove by substitution

$$p \log p = O(n)$$

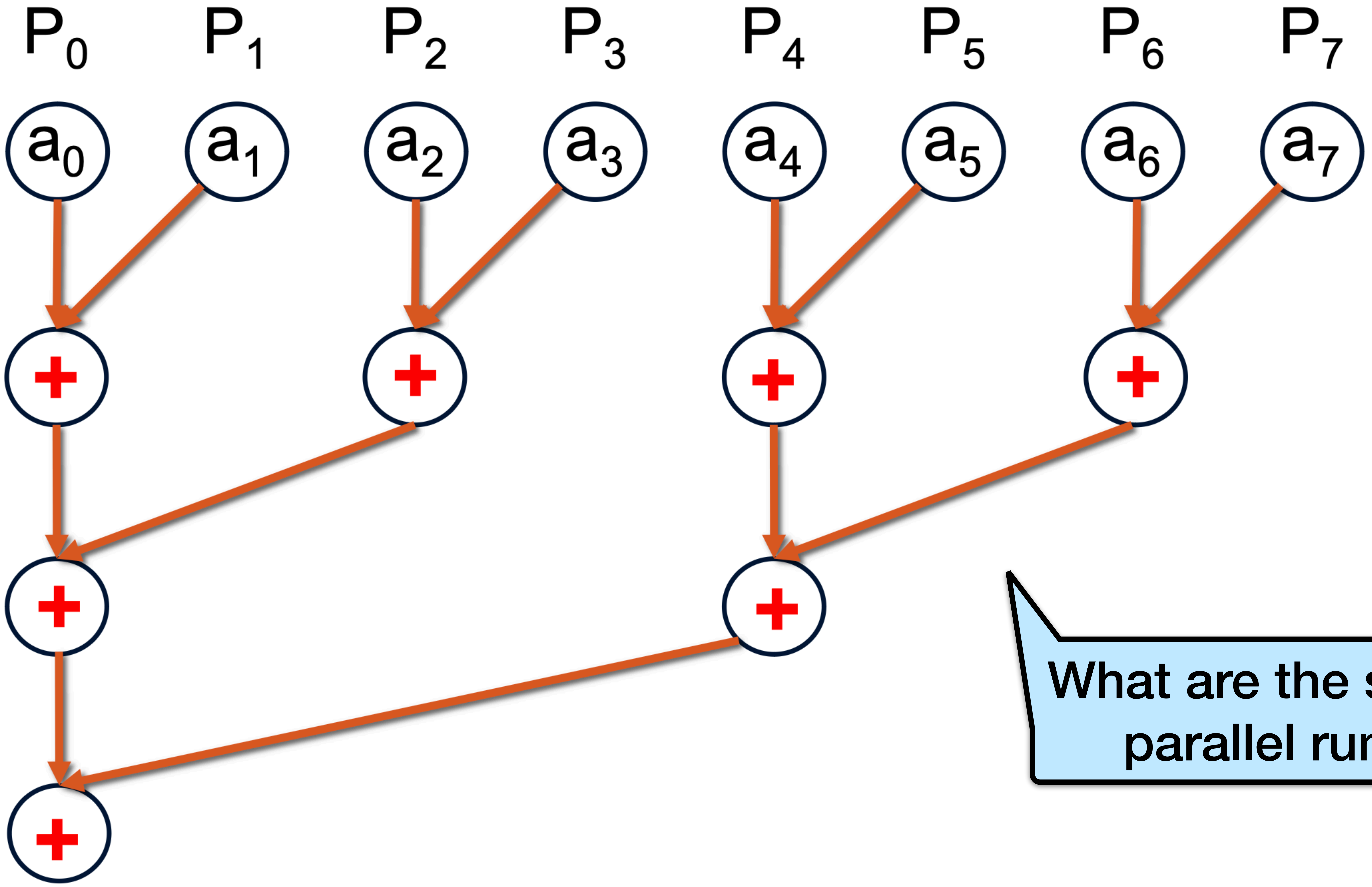
Guess: $p = O\left(\frac{n}{\log n}\right)$

$$O\left(\frac{n}{\log n} \cdot \log \frac{n}{\log n}\right) = O\left(\frac{n}{\log n} \cdot (\log n - \log \log n)\right)$$

$$O\left(\frac{n}{\log n} \cdot \log n\right) = O(n)$$

Communication and parallel algorithms

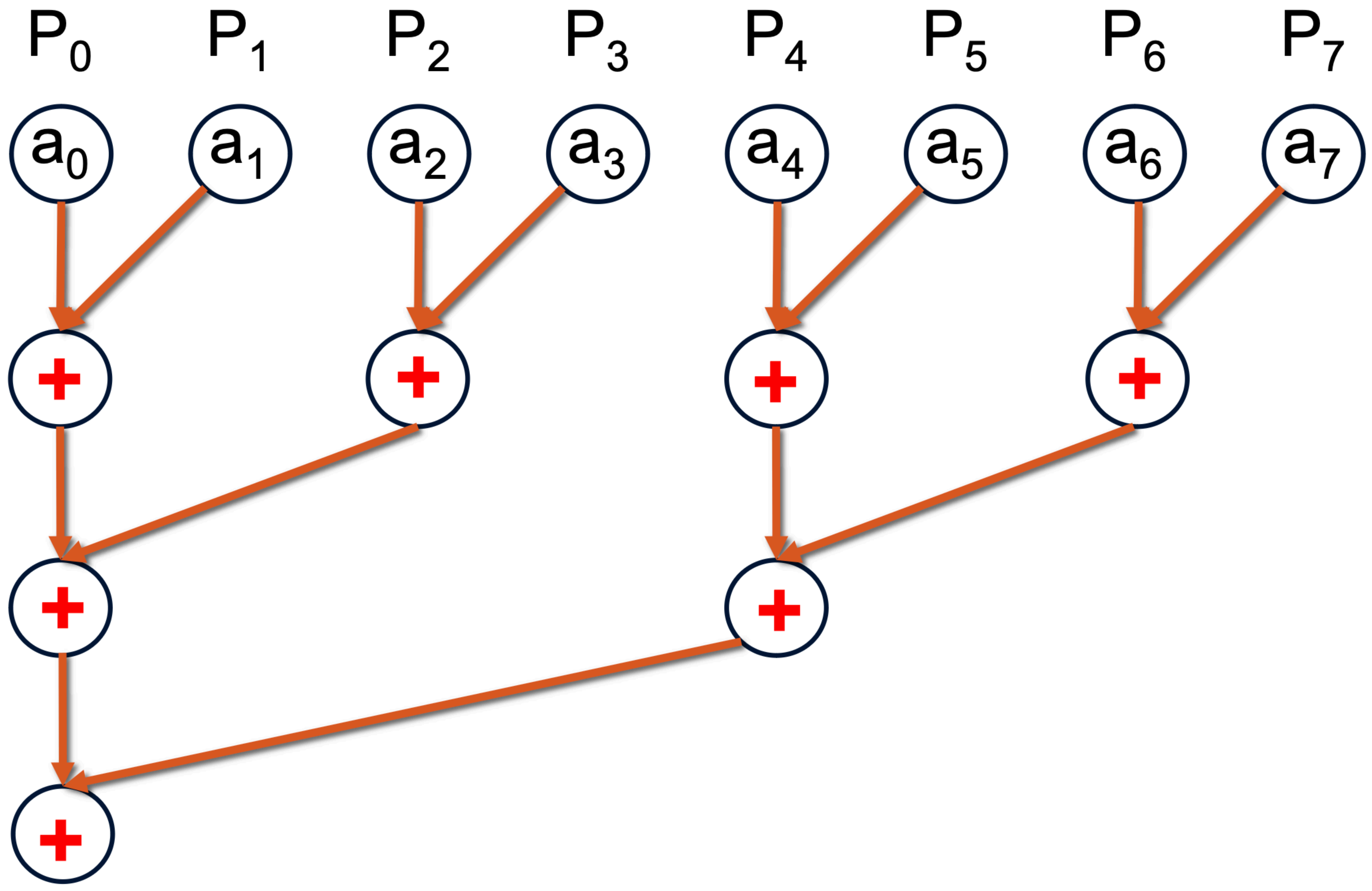
Example: Parallel sum



What are the serial and parallel runtime?

Analysis of parallel sum

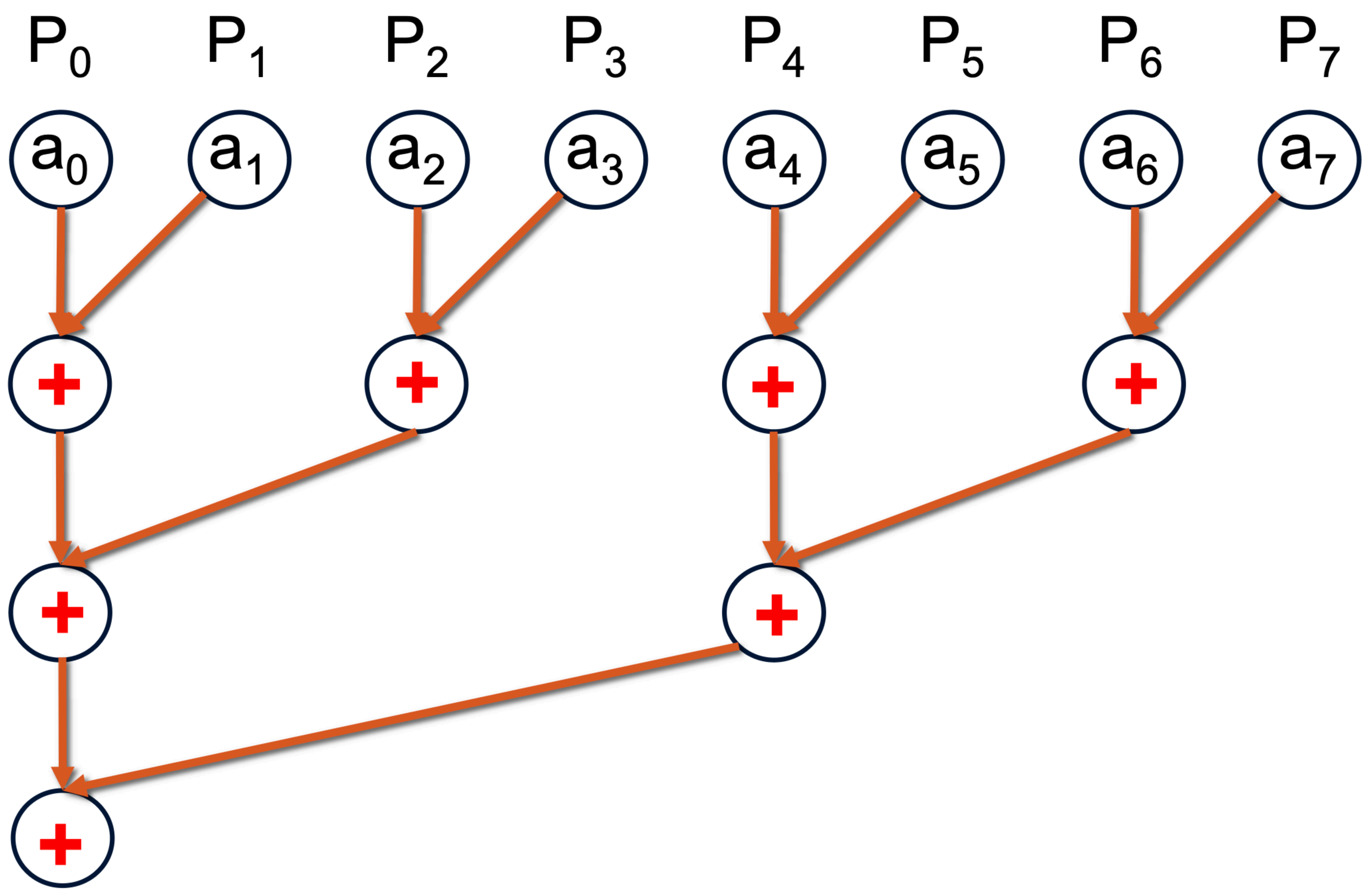
Serial runtime?



Analysis of parallel sum

Serial runtime: $T(n,1) = \Theta(n)$

Parallel runtime?

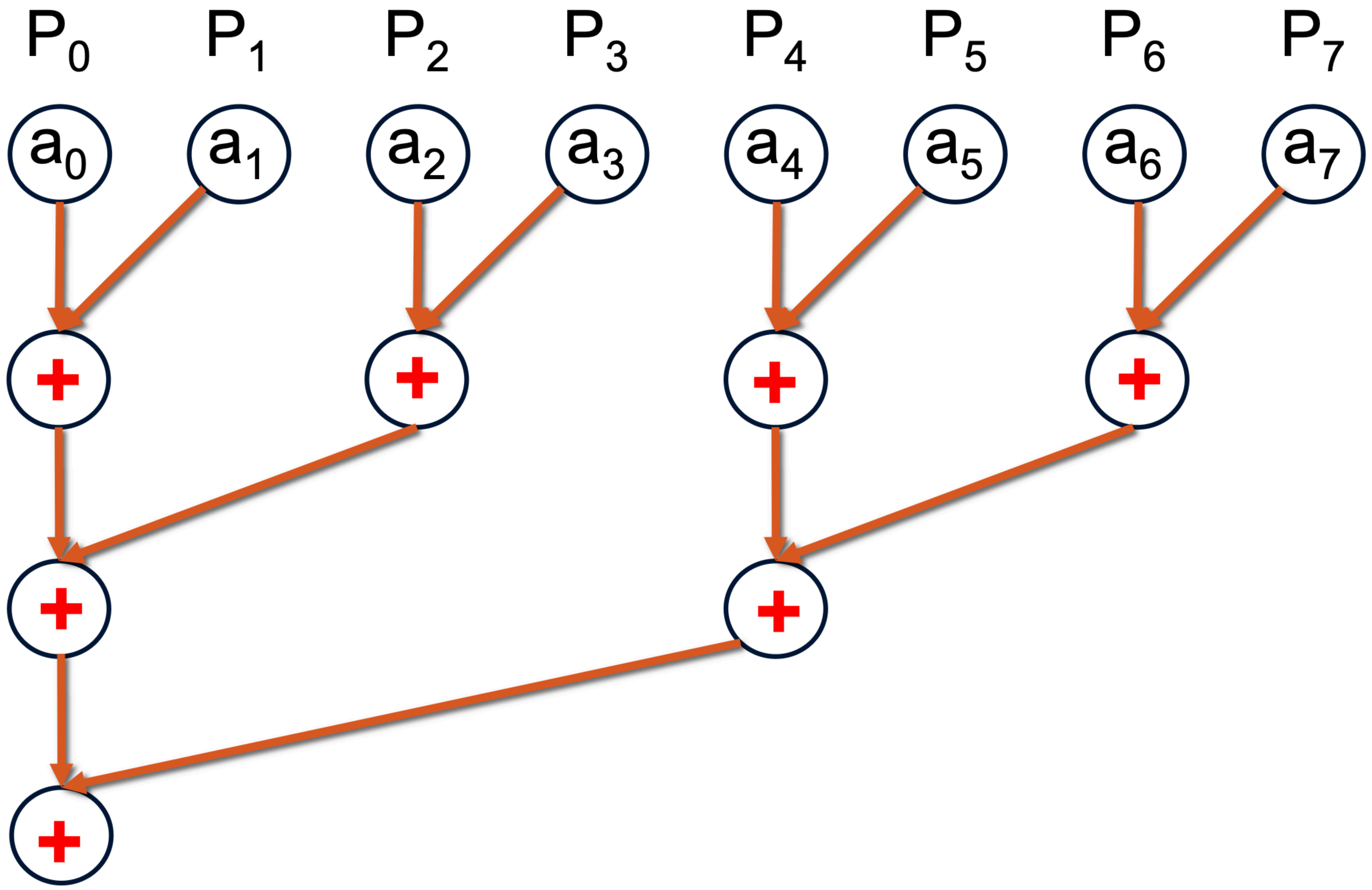


Analysis of parallel sum

Serial runtime: $T(n,1) = \Theta(n)$

Parallel runtime: $T(n, n) = \Theta(\log n)$

Speedup?

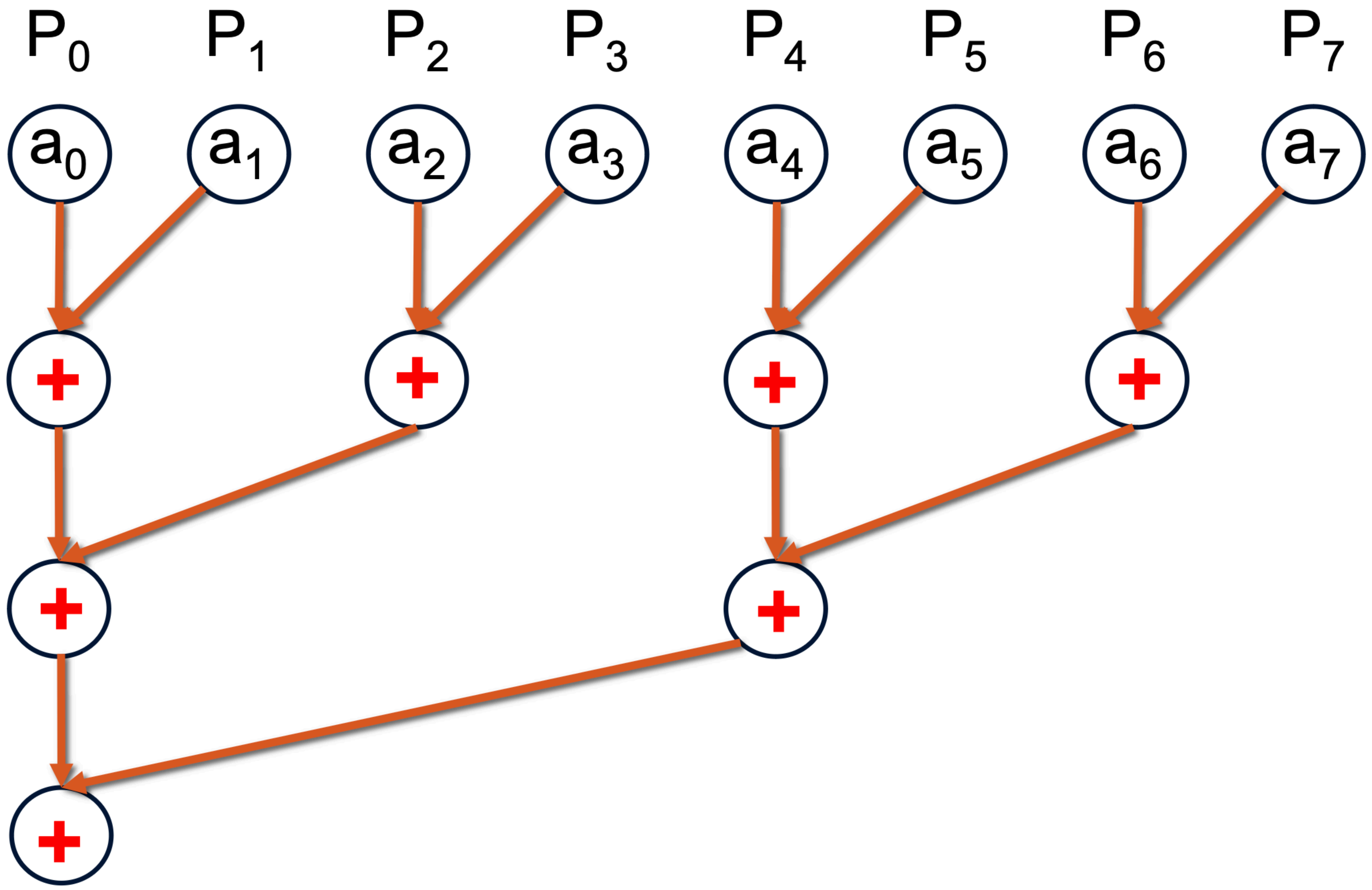


Analysis of parallel sum

Serial runtime: $T(n,1) = \Theta(n)$

Parallel runtime: $T(n,n) = \Theta(\log n)$

Speedup: $S(n) \leq \dfrac{n}{\log n}$



Concrete example of parallel sum

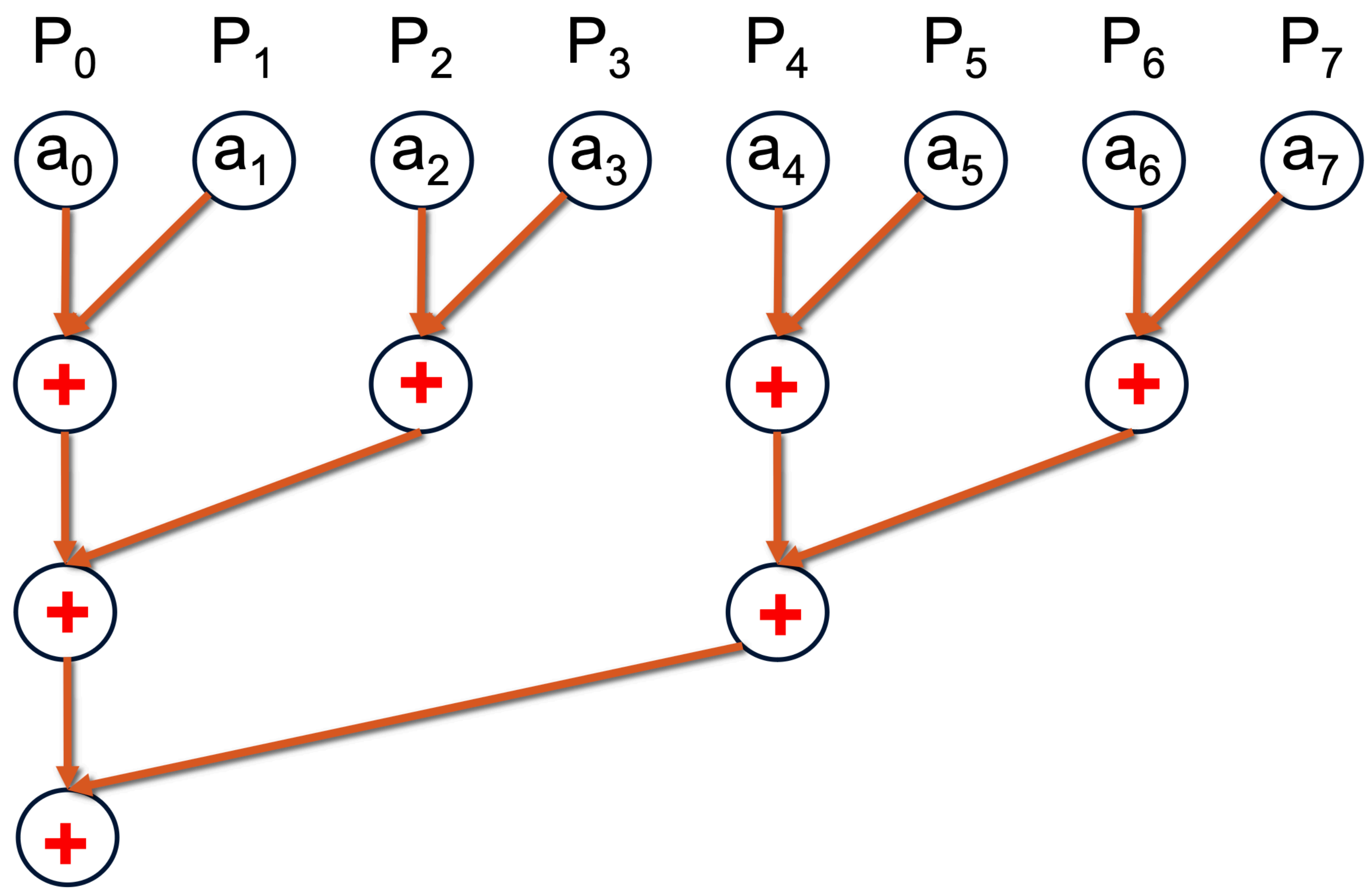
Suppose $n = 1{,}000{,}000$.

$$S(n) \leq \frac{n}{\lg n}$$

$$S(n) \leq \frac{1{,}000{,}000}{20}$$

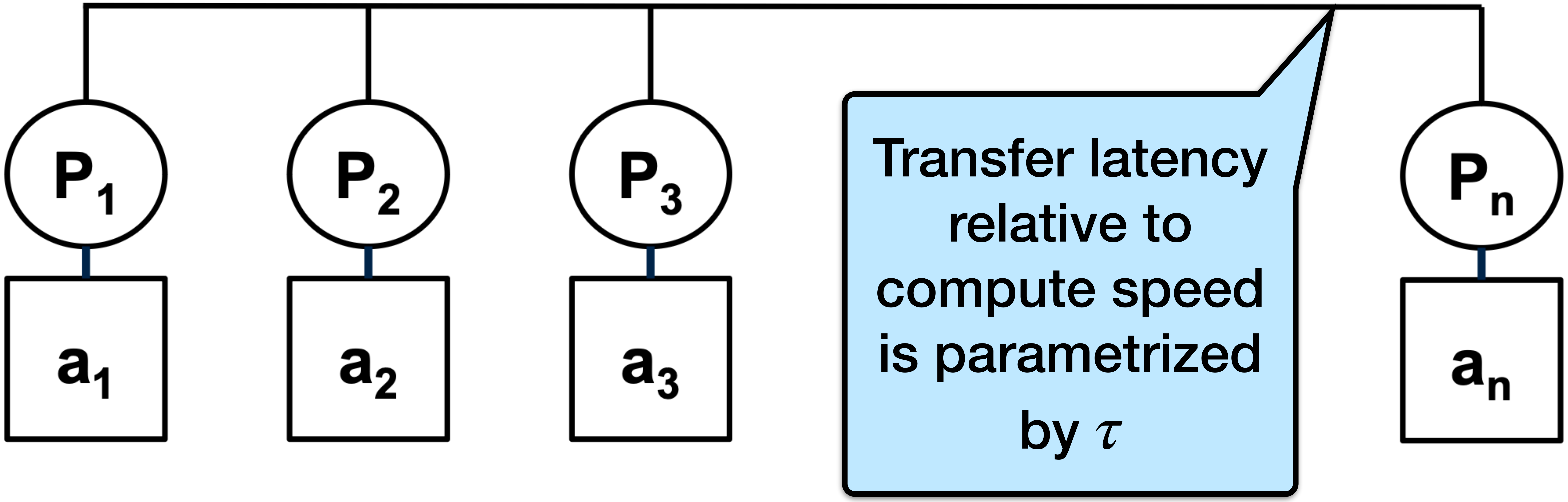
$$S(n) \leq 50{,}000$$

5% of n



Interconnection Networks

In real systems, processors are connected by a bus:



Transfer latency τ is usually on the order of 10,000 slower than local computation.

Interconnect Network and Parallel Speedup

Serial runtime: $T(n,1) \approx 1 \cdot n$

Parallel runtime: $T(n,n) \approx \tau \cdot \log n$

With the previous example numbers of $n = 1{,}000{,}000$ and $\tau = 10{,}000$

$$\text{Speedup: } S(n) \approx \frac{1{,}000{,}000}{10{,}000 \cdot 20} = 5$$

Any ideas about how to improve the speedup?

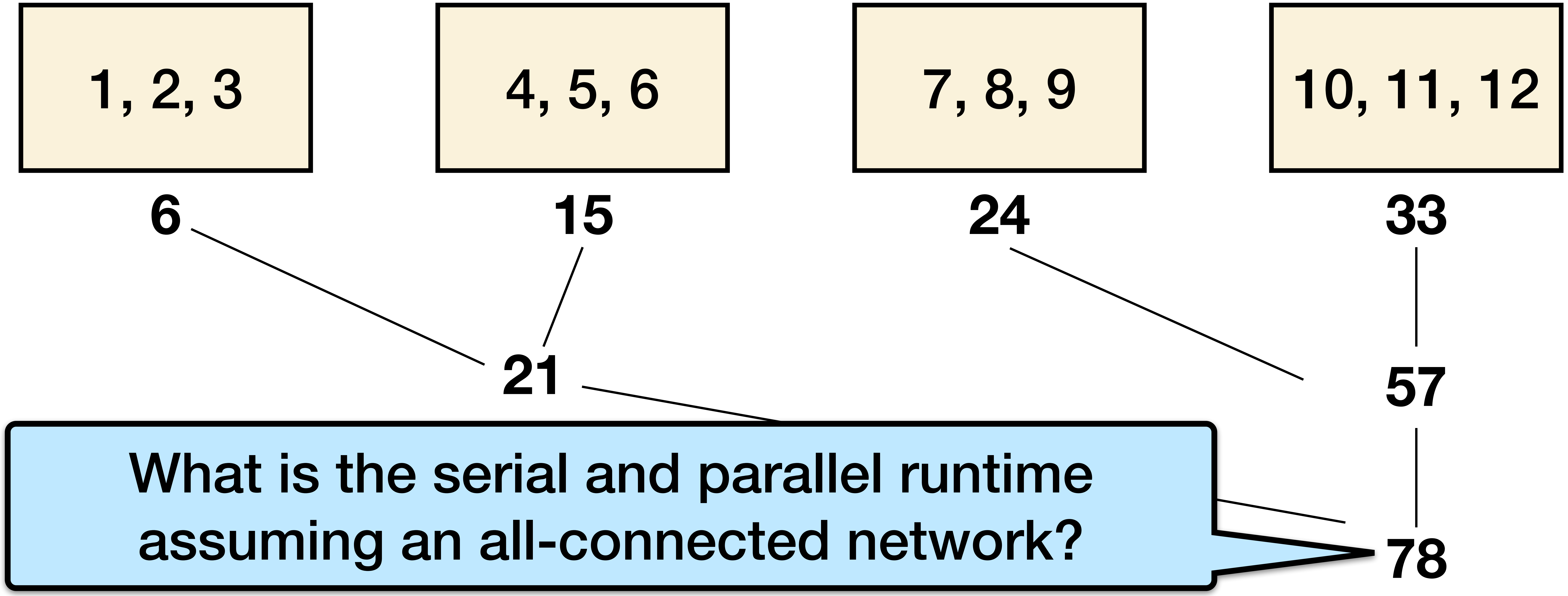
Another parallel sum algorithm

Given n numbers, compute their sum using $p \ll n$ processors.

Step 1: Serial - Sum local n/p numbers

Step 2: Parallel - Add p numbers using p processors

p = 4



What is the serial and parallel runtime assuming an all-connected network?

Analysis of partially local parallel sum

Serial runtime: ?

Analysis of partially local parallel sum

Serial runtime: $T(n,1) \approx n$

Parallel runtime?

Hint: should include τ

Analysis of partially local parallel sum

Serial runtime: $T(n,1) \approx n$

Parallel runtime: $T(n,n) \approx \frac{n}{p} + \tau \cdot \log p$

Speedup?

Analysis of partially local parallel sum

Serial runtime: $T(n,1) \approx n$

Parallel runtime: $T(n,n) \approx \dfrac{n}{p} + \tau \cdot \log p$

Speedup: $S(p) \approx \dfrac{n}{\frac{n}{p} + \tau \cdot \log p}$

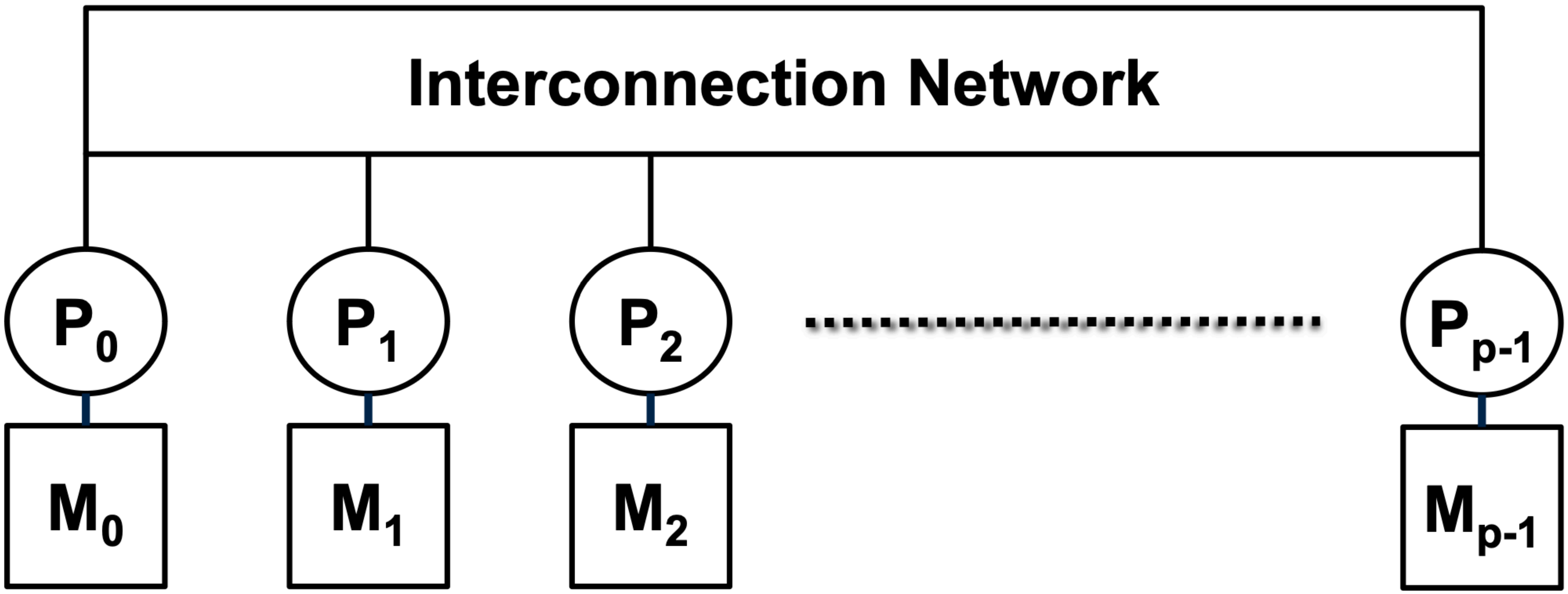
$$S(p) \geq \frac{p}{2} \quad \text{if} \quad \tau p \log p \leq n$$

n = 1,000,000

$$p \log p \leq \frac{n}{\tau} = 100$$

$$p \leq 22$$

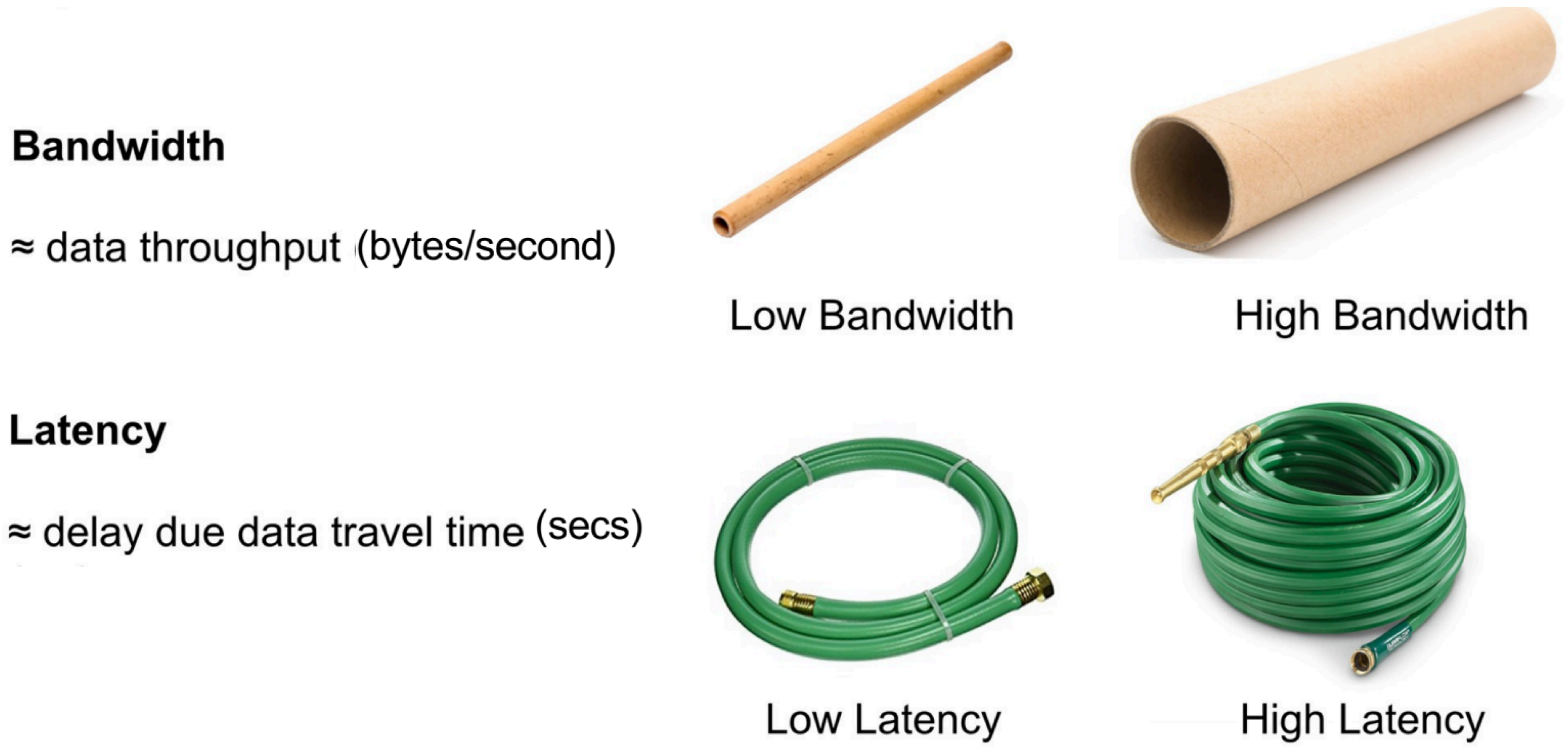
Modeling Parallel Interconnects



Interconnect Network Metrics

Memory accesses (load/store) have two costs:

- **Latency** - the startup cost to transfer (τ)
- **Bandwidth** - the average rate (bytes / sec) to send/receive a large chunk of data (μ)



Modeling Communication

Suppose processor p_i is sending message of size m to processor p_j

Transfer time: $t_c = \tau + \mu \cdot m$

Example: Bandwidth is 100 Gbps = 12.5 GBytes/s

μ (per byte) = $\dfrac{1}{12.5 \times 10^9} = 0.08$ns

μ (per word) = 0.32ns

$$\tau : \mu : 1 \Rightarrow 10^3 - 10^5 : 1 - 10 : 1$$

3GHz ⇒ 1 clock cycle = $\dfrac{1}{3}$ns

Summary

- The goal of parallel algorithm design is to achieve **work efficiency**, or asymptotically equal work to the best serial algorithm.

- Parallel speedup may be limited by the **interconnect network**.

  - Speedup may be improved by **coarsening** the parallelism.