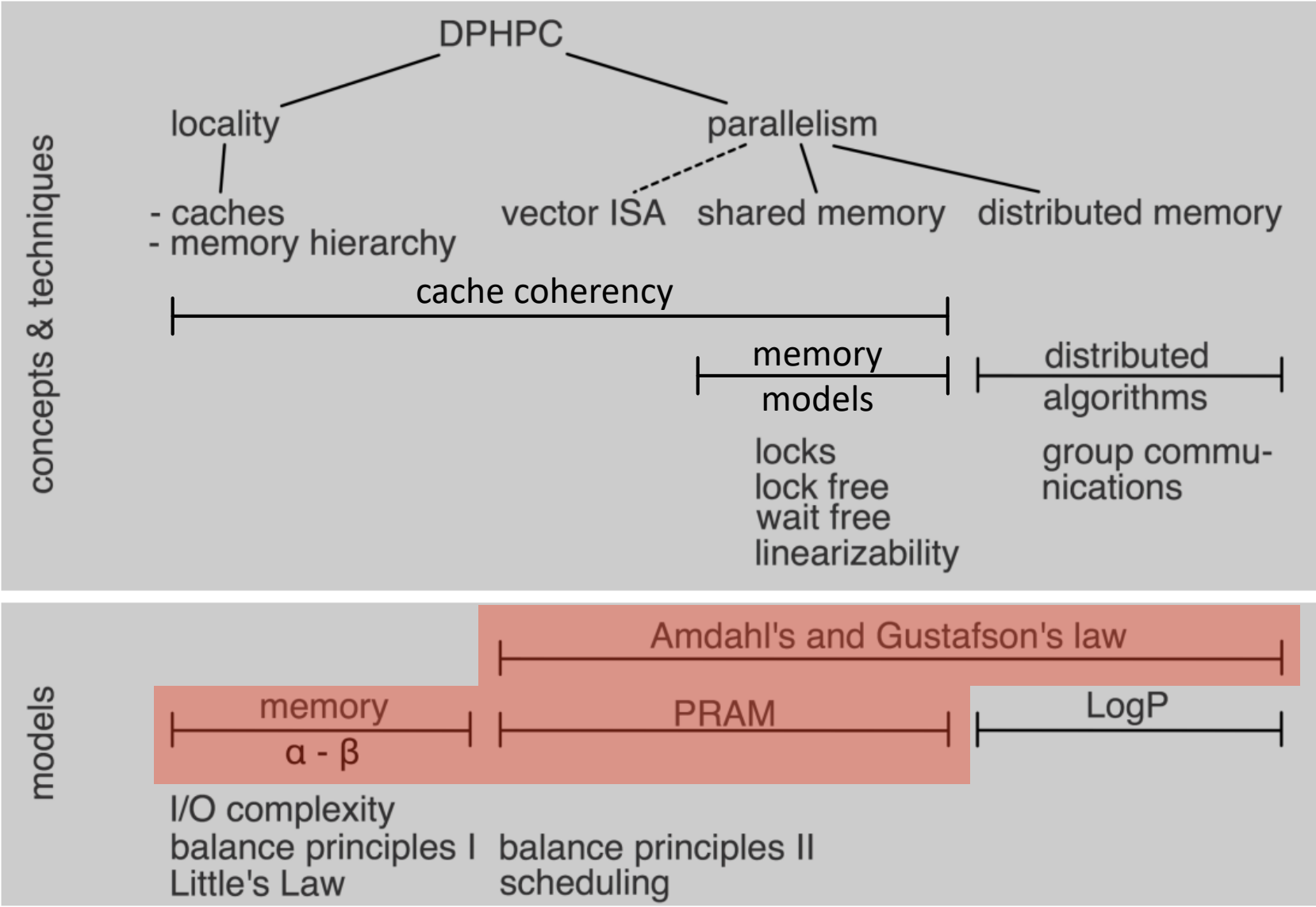
Models: Amdahl's Law, PRAM, α-β

Tal Ben-Nun

Design of Parallel and High-Performance Computing

Fall 2017

DPHPC Overview



Speedup

- An application runs in time T_1 on one processor, T_p for p processors.

- Measured speedup on p processors: $\frac{T_1}{T_p}$

- Can we bound/estimate speedup without running the application?

Scaling Models – Intuition

Consider the following application:

```
auto A = InitializeArray(random_seed,
                         important_info);
```

Sequential

```
for (int i = 0; i < 10000; ++i)
    for (int j = 0; j < 10000; ++j)
        B[i][j] = Compute(A[i][j]);
```

Parallel

```
SequentialDecisionProcess(B);
```

Sequential

```
for (int i = 0; i < 10000; ++i)
    for (int j = 0; j < 10000; ++j)
        C[i][j] = Compute(B[i][j]);
```

Parallel

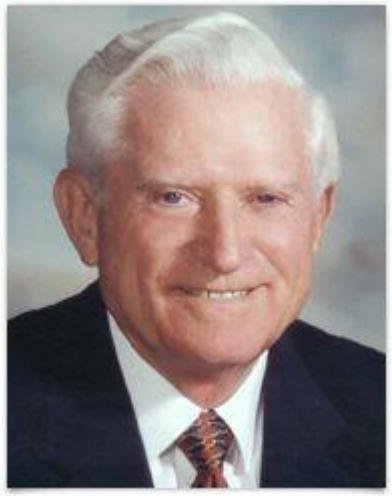
```
WriteResults(C);
```

Sequential

Amdahl's Law

- **Given $f \in [0, 1]$ the sequential fraction of an application:**

$$T_p \geq \frac{(1-f)T_1}{p} + f \cdot T_1$$



Gene Amdahl
1922-2015

Scaling Models – Intuition (revisited)

Consider the following application:

```
auto A = InitializeArray(random_seed,
                         important_info);
for (int i = 0; i < 10000; ++i)
    for (int j = 0; j < 10000; ++j)
        B[i][j] = Compute(A[i][j]);
SequentialDecisionProcess(B);
for (int i = 0; i < 10000; ++i)
    for (int j = 0; j < 10000; ++j)
        C[i][j] = Compute(B[i][j]);
WriteResults(C);
```

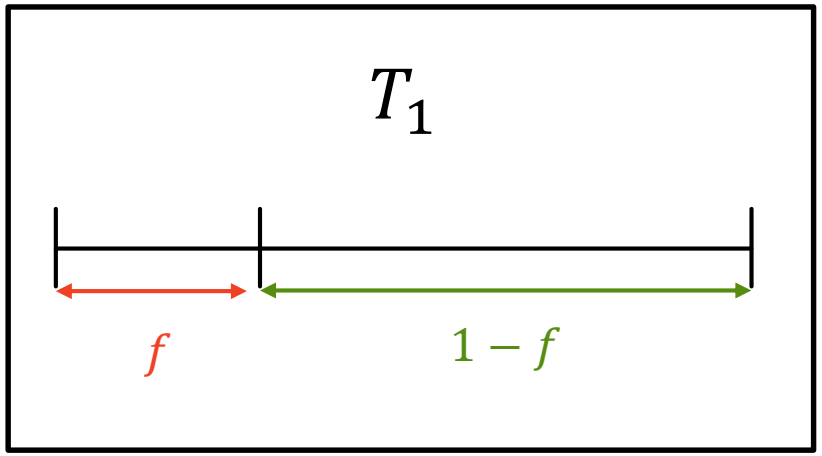
Sequential

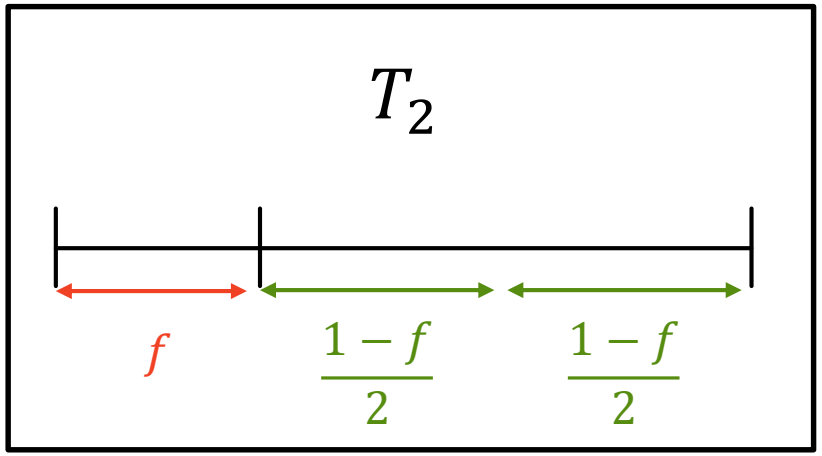
Parallel

Sequential

Parallel

Sequential





Amdahl's Law

- **Given $f \in [0, 1]$ the sequential fraction of an application:**

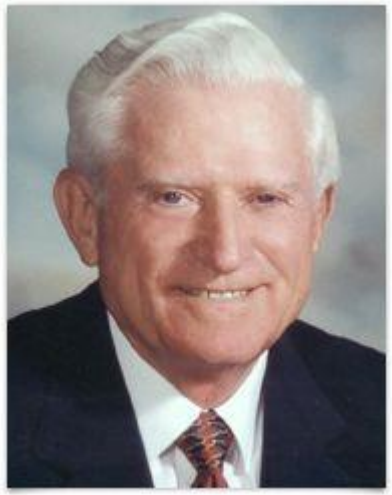
$$T_p \geq \frac{(1-f)T_1}{p} + f \cdot T_1$$

- **Speedup can be modeled as:**

$$S_p \leq \frac{T_1}{T_p} \leq \frac{1}{\frac{1-f}{p} + f}$$

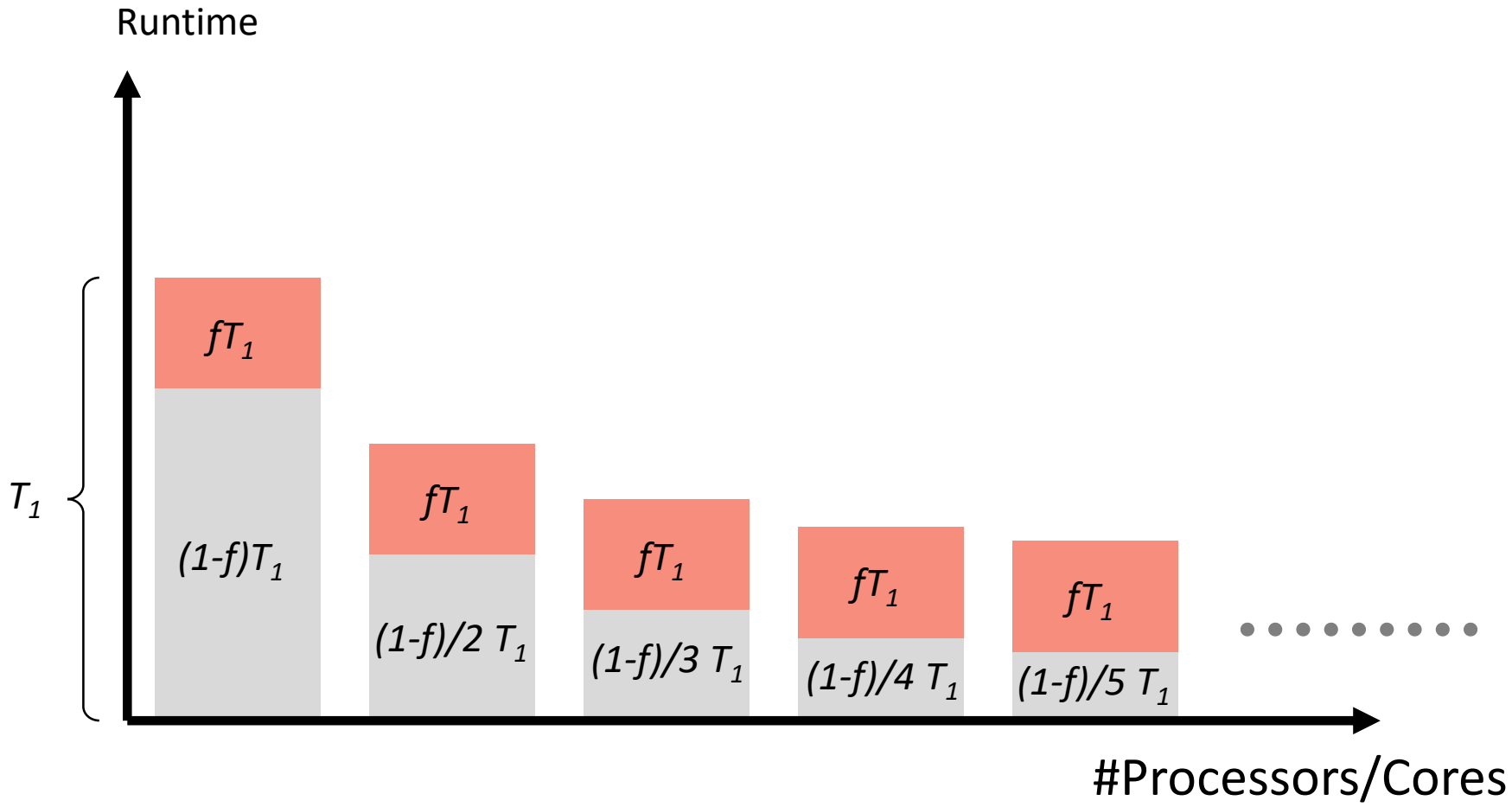
- **And efficiency:**

$$E_p = \frac{S_p}{p} \leq \frac{1}{1 - f + fp}$$

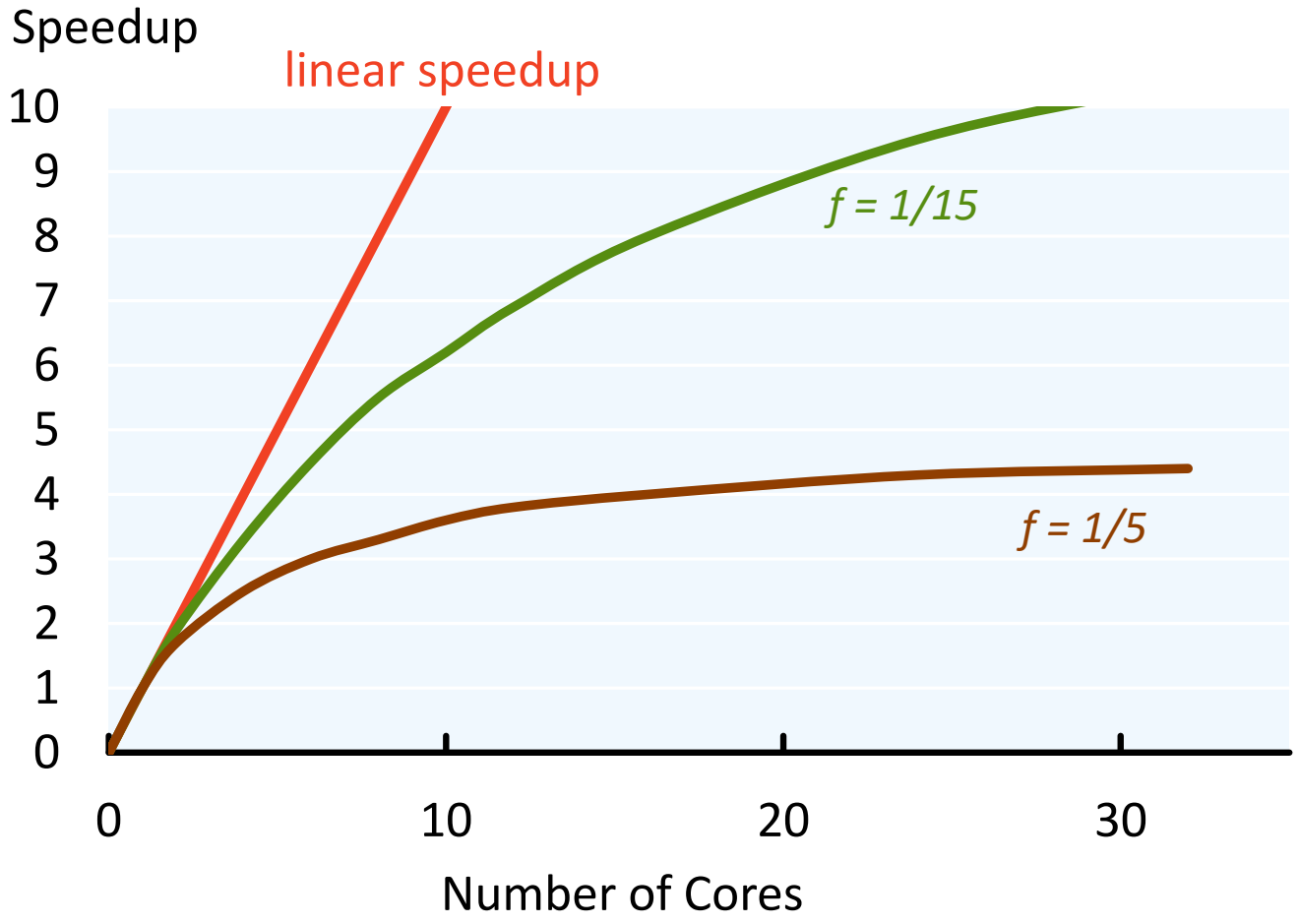


Gene Amdahl
1922-2015

Amdahl's Law: Runtime



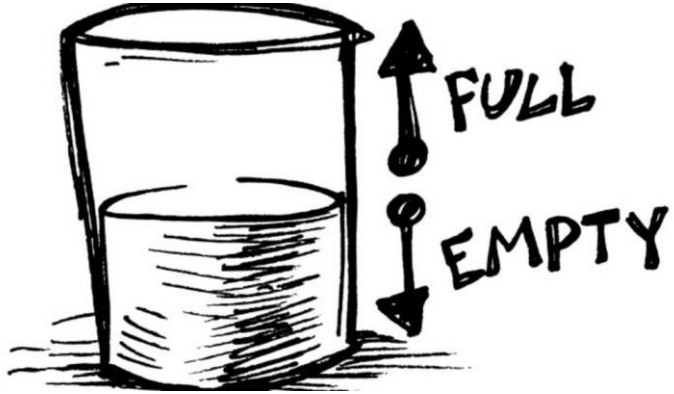
Amdahl's Law: Speedup



Amdahl's Law

- **As** $p \to \infty$:
  - $T_\infty \geq f \cdot T_1$
  - $S_\infty \leq \frac{1}{f}$
  - $E_\infty = 0$ (if $f \neq 0$)

Question



Is Amdahl’s law pessimistic or optimistic?

Amdahl's Law is Pessimistic

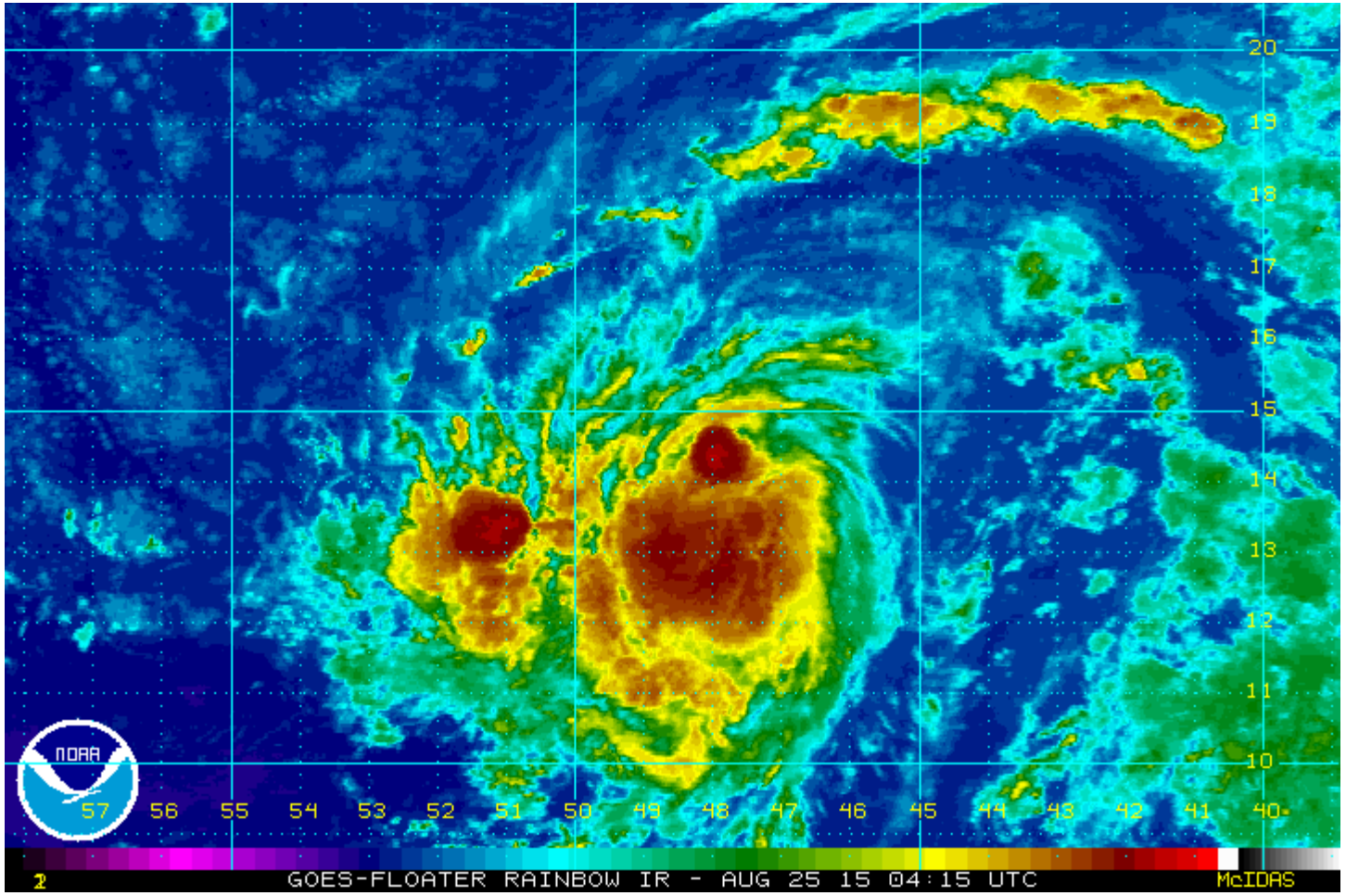
- **Fixes the problem size**

- **More processors → Larger problems**

- **$\boldsymbol{f}$ is not a constant fraction, but $\boldsymbol{f(n)}$**
  - The same applies for T_1 and $T_1(n)$

Case Study: Weather Forecasting

- **Grid point climate/atmospheric models**
  - GFS
  - GME
  - COSMO
- **The denser the grid, the more accurate prediction becomes**
- **Important for anticipating natural disasters!**



Gustafson's Law

- **States:**

$$\boldsymbol{S_\infty \leq \frac{1}{f(n)} \xrightarrow[n \to \infty]{} \infty} \quad \text{(if } f(n) \to 0\text{)}$$

John Gustafson

→ Larger problems can be solved (with more resources) at the same time.

Strong and Weak Scaling

- **<u>Strong scaling:</u>**

  - Behavior of $S_p(n)$ for a fixed n, and $p \to \infty$

    Fixed problem size as p increases

- **<u>Weak scaling:</u>**

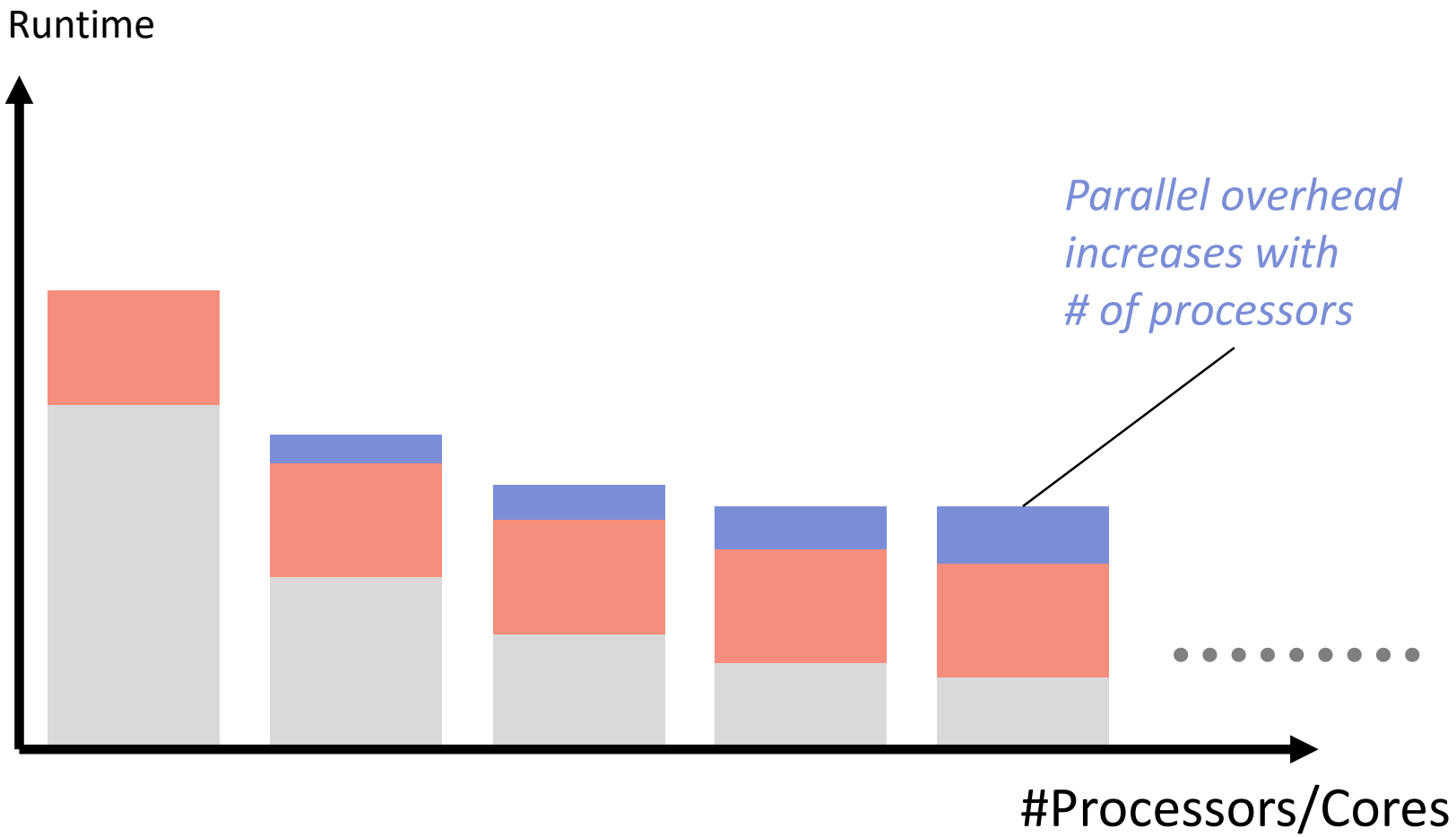
  - Behavior of $S_p(n)$ for $n, p \to \infty$

    Increasing problem size as p increases

Pessimism/Optimism Revisited

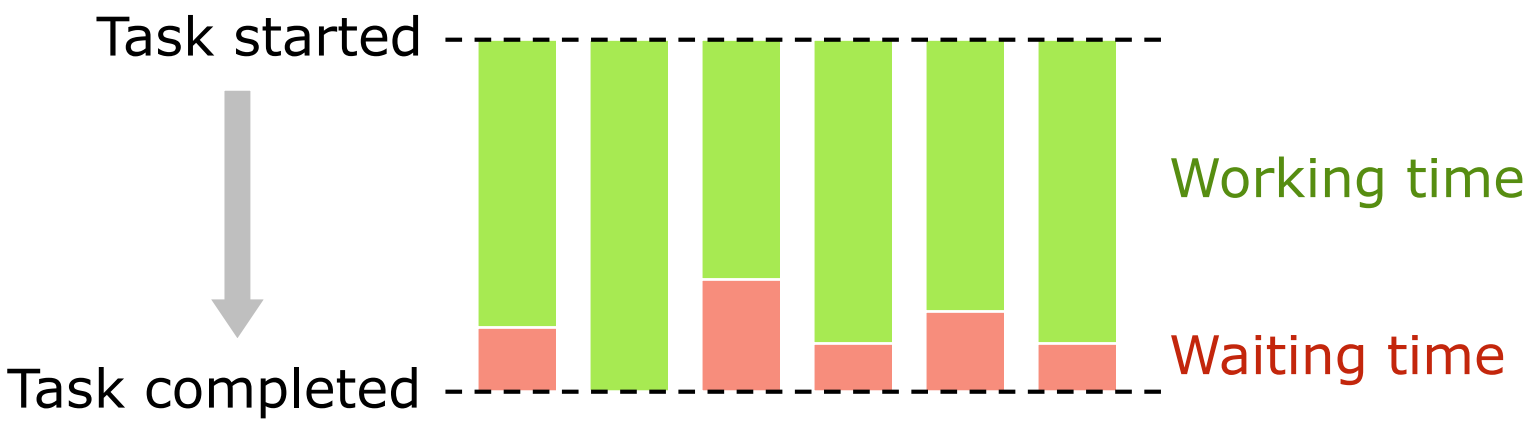
Could both models be optimistic?

Runtime With Parallel Overhead

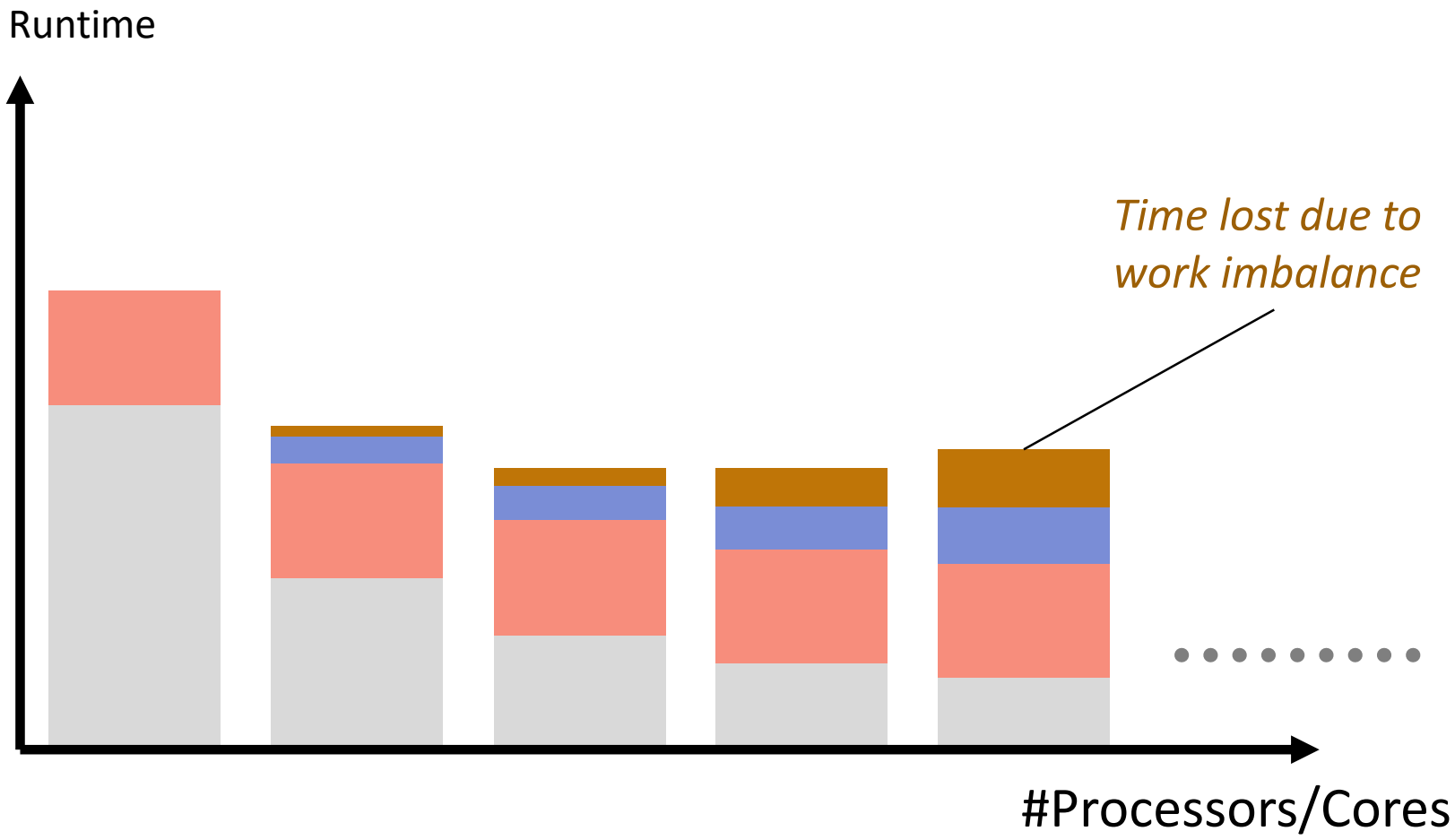


Other Issue: Load Balancing

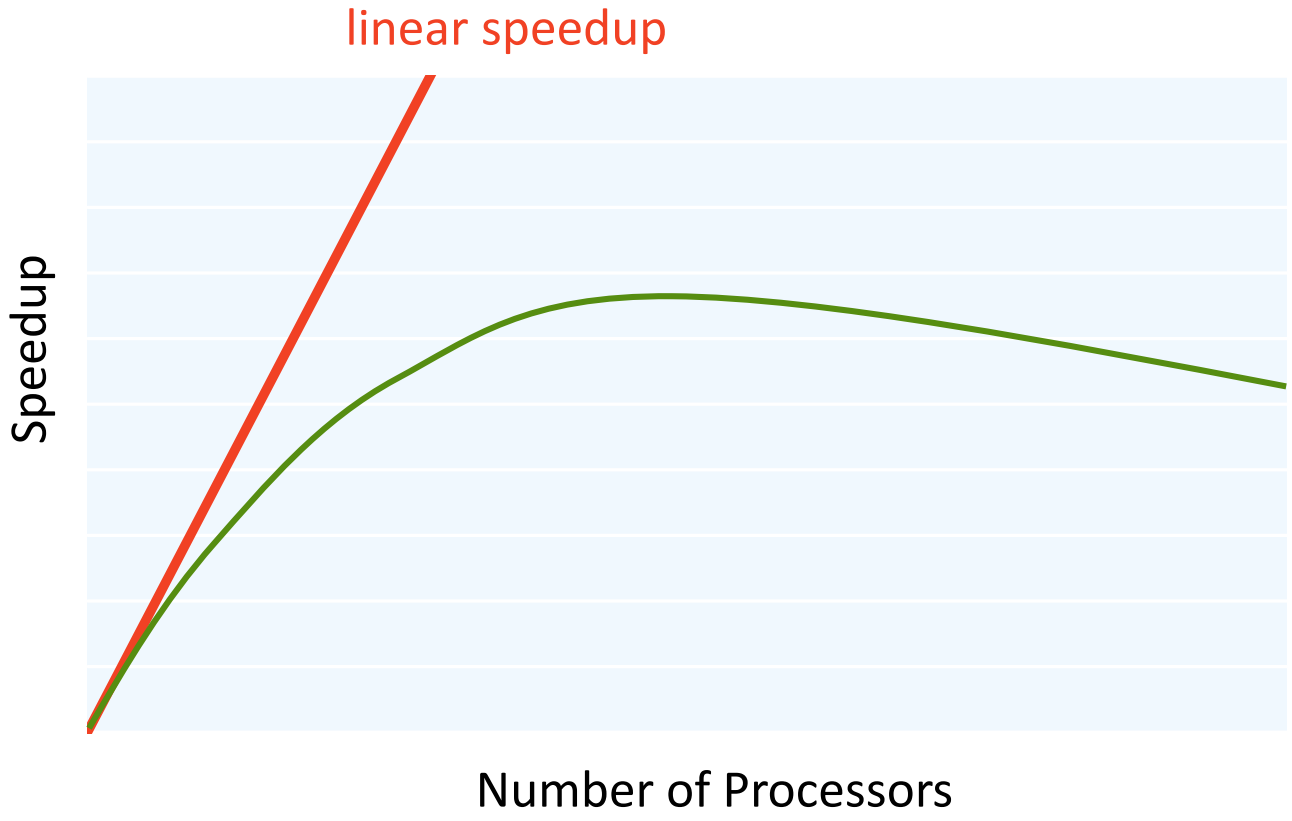
- **Amdahl's Law assumes that computation divides evenly**
- **Processor waiting time is another form of overhead**



Runtime with Parallel Overhead and Work Imbalance



Real Speedup Graphs often Look like This:



Additional Resources

- **Assumption in both laws: increasing p does not change other resources (e.g. cache)**

- **It is thus possible to achieve super-linear speedup:**
  - Example 1: Cache size increase leads to larger working set
  - Example 2: Probabilistic convergent algorithms

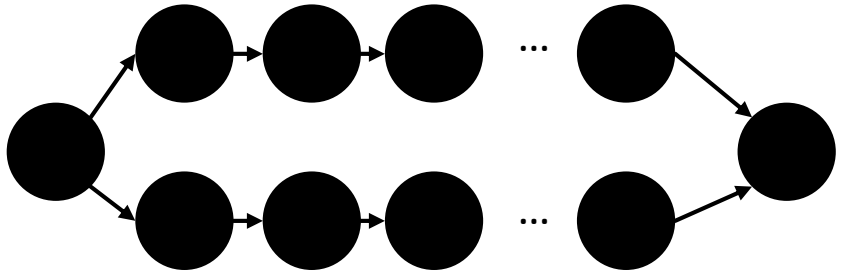
Amdahl's Law: Summary

- **Strong, simple models for scaling**

- **Ignore overhead incurred by parallelization and load imbalance**

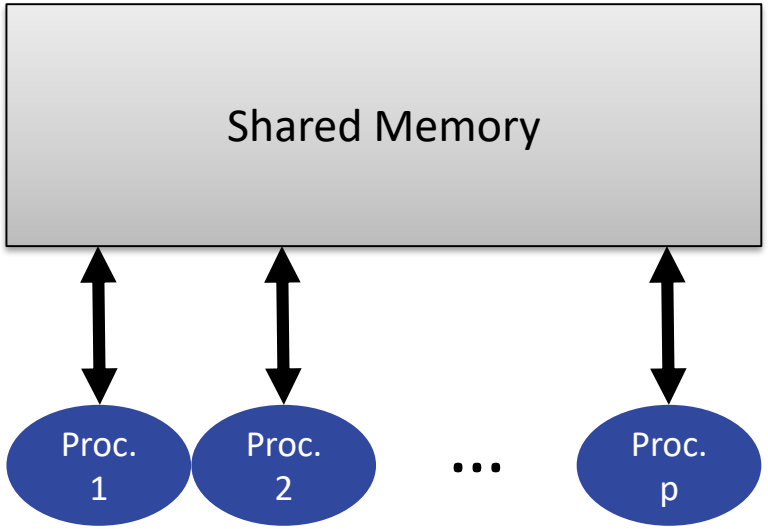
- **In reality:** $\boldsymbol{S_p(n) = \frac{T_1(n)}{T_p(n)+A_p(n)}}$

- **Programs are not always that simple**
  - $\boldsymbol{f}$ does not always exist



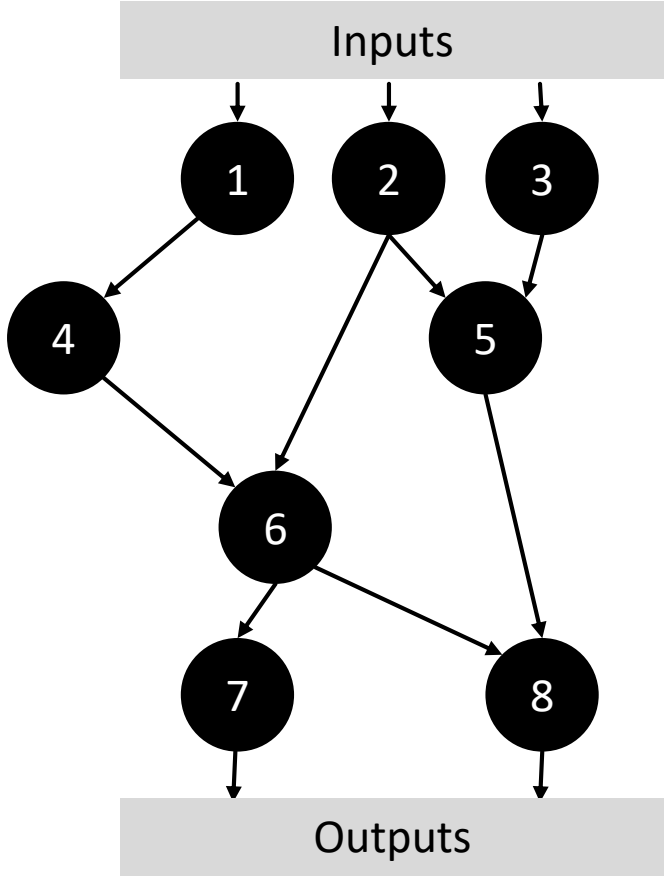
PRAM Model

- **Parallel Random Access Machine**
- **Abstract model for shared memory parallelism**
- **Any processor can access the memory at unit time**
- **What about access conflicts?**
  - Exclusive read exclusive write (EREW)
  - Concurrent read exclusive write (CREW)
  - **Concurrent read concurrent write (CRCW)**



PRAM Model: Program Representation

- **Programs represented as DAGs (Directed Acyclic Graphs)**
  - Nodes: Unit-time operations
  - Edges: Dependencies



- **We can now define:**
  - $W(n) = \#nodes$
  - $D(n) =$longest path from input to output
- **Average parallelism:** $W(n)/D(n)$

PRAM Model: Advantages

- **Powerful tool for analyzing parallel algorithms**

- **No need to manage communication and synchronization**
  - Focus on algorithmic aspects

- **Time-unit and processor abstraction is robust w.r.t. architectures**

PRAM Model: Examples

- **Consider reduction: $(x_1 + x_2 + \cdots + x_n)$**

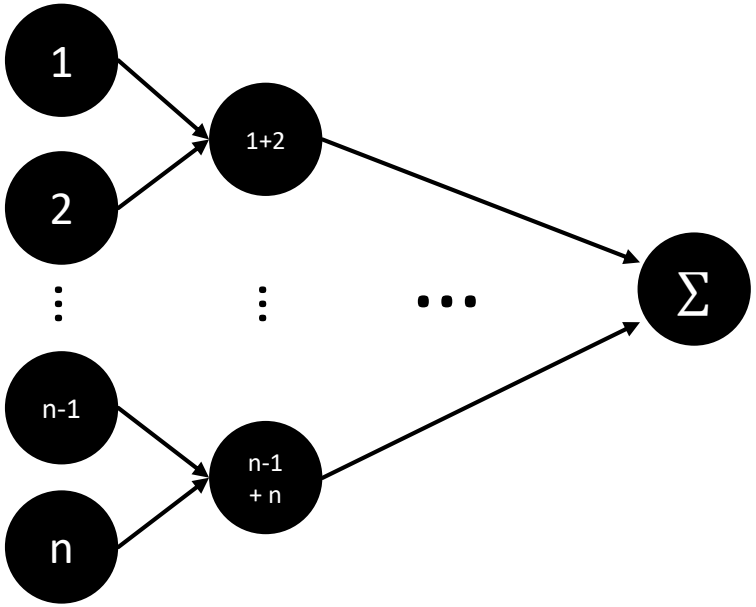
- **Sequential:**

$W(n) = \Theta(n)$

$D(n) = \Theta(n)$

Parallelism = $\mathcal{O}(1)$

- **Binary tree:**



$W(n) = \Theta(n)$

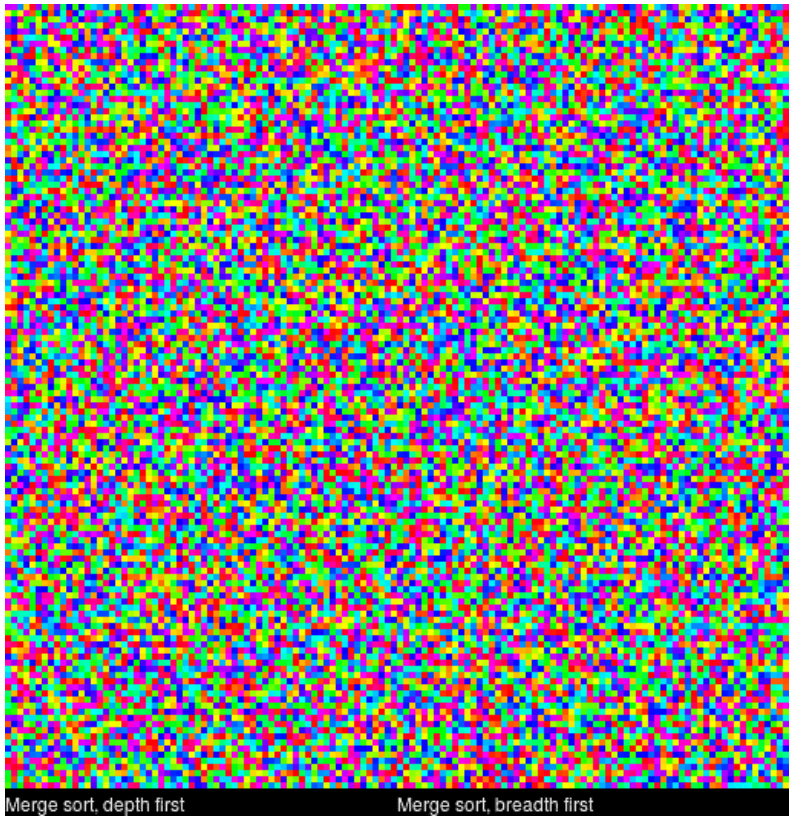
$D(n) = \Theta(\log n)$

Parallelism = $\mathcal{O}(n/\log n)$

PRAM Model: Examples

- **Merge-sort:**

```
List MergeSort(List L) {
    if (L.length() == 1)
        return L;
    auto L1 = MergeSort(L.sublist(0, n / 2));
    auto L2 = MergeSort(L.sublist(n / 2 + 1, n));
    return merge(L1, L2);
}
```



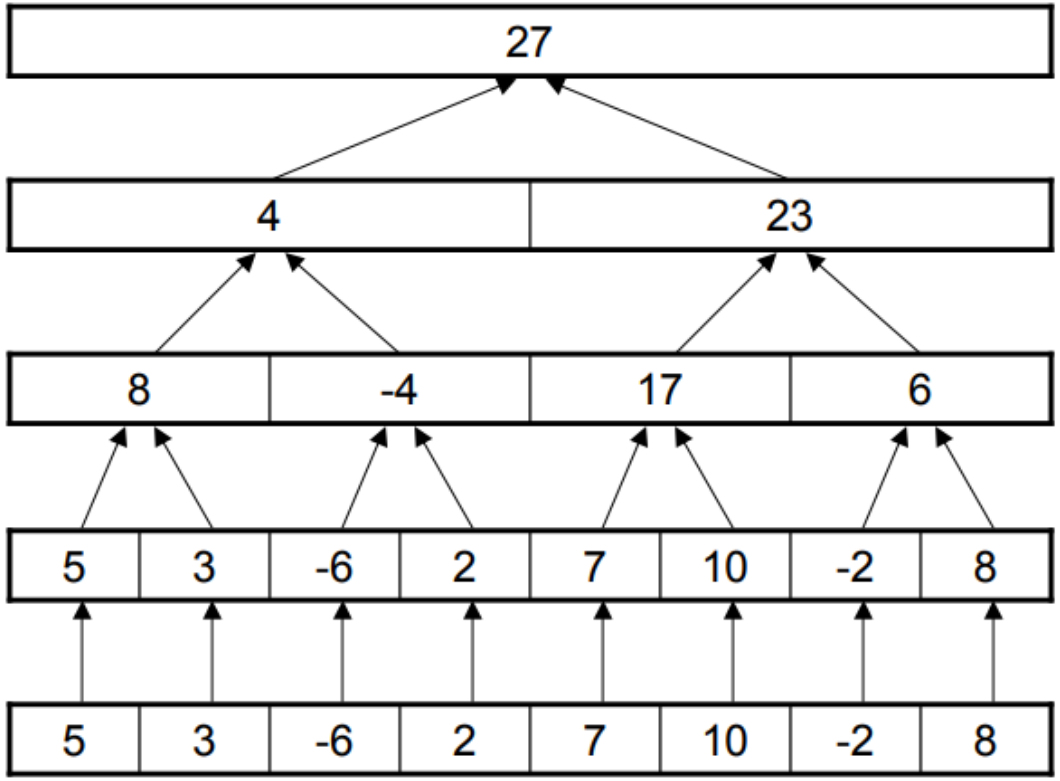
Animation: u/morolin

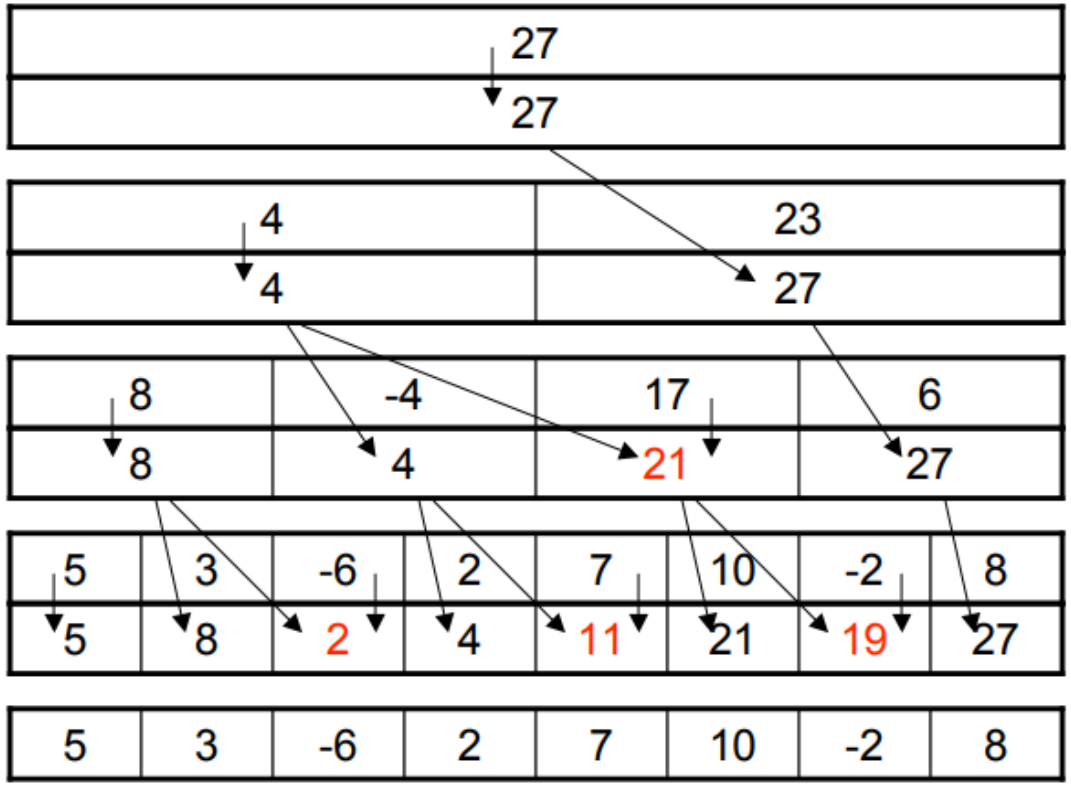
- $W(n) = \Theta(n \log n)$
- $D(n) = D\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n)$
- **Average parallelism:** $\Theta(\log n)$

PRAM Model: Examples

Images: The PRAM Model and Algorithms
Prof. Robert van Engelen

- **Scan (prefix sum):**
  - Input: L = $(x_1, \dots, x_n)$
  - Output: $(0, x_1, x_1 + x_2, \dots, \sum x_i)$





PRAM Model: Examples

- **Scan (prefix sum):**
  - Input: L = $(x_1, \ldots, x_n)$
  - Output: $(0, x_1, x_1 + x_2, \ldots, \sum x_i)$

```
List Scan(List L) {
    if (L.length() == 1)
        return List { 0 };
    List sums (L.length()/2);
    for (int i = 0; i < L.length()/2; ++i)
        sums[i] = L[2 * i - 1] + L[2 * i];

    List evens = Scan(sums);

    List odds(ceil(L.length() / 2.0));
    for (int i = 0; i < odds.length(); ++i)
        odds[i] = evens[i] + L[2 * i];

    return Interleave(evens, odds);
}
```

$$W(n) = W\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n)$$

$$D(n) = D\left(\frac{n}{2}\right) + \Theta(1) = \Theta(\log n)$$

Parallelism: $\mathcal{O}(n/\log n)$

Reasoning in PRAM

- **Given a DAG with $\boldsymbol{W(n)}$ nodes and $\boldsymbol{D(n)}$ depth:**
  - Sequential runtime: $T_1(n) = W(n)$
  - Time on ∞ processors: $T_\infty(n) = D(n)$
  - Time on p processors: $T_p(n) = ?$

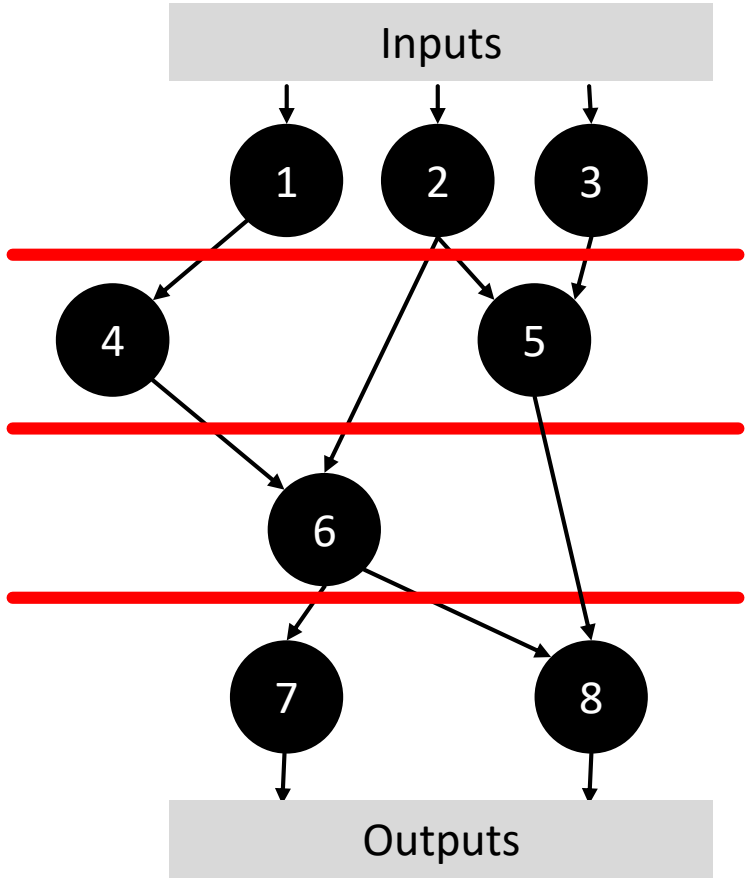
- $T_p(n) \geq D(n), \quad T_p(n) \geq W(n)/p$

- Can we bound T_p from above?

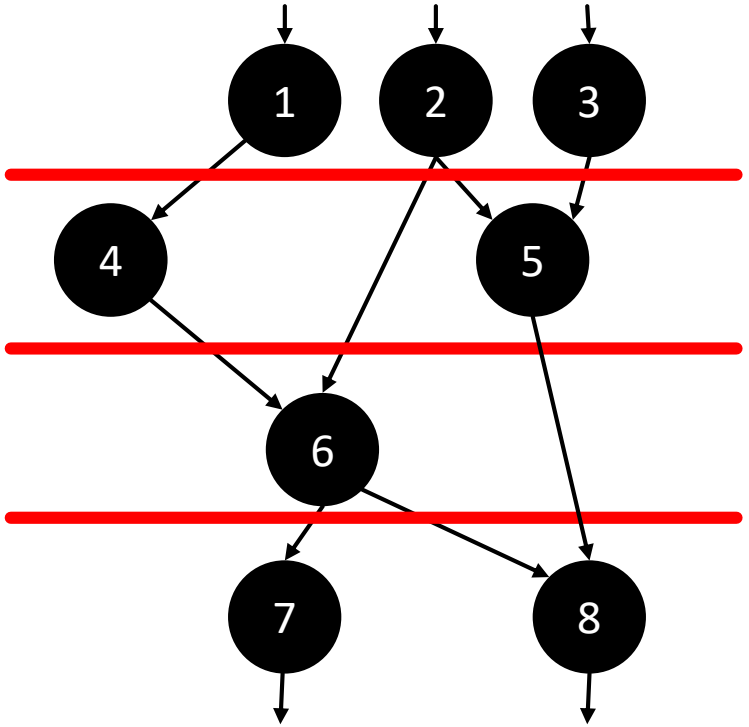
Brent's Theorem

$$T_p(n) \leq D(n) + \frac{W(n) - D(n)}{p}$$

- **Intuition: Bound parallelism at each level of the DAG**



Brent's Theorem



	T_p breakdown
s_1 operations	$\left\lceil \frac{s_1}{p} \right\rceil$
s_2 operations	$\left\lceil \frac{s_2}{p} \right\rceil$
$\vdots$	
s_D operations	$\left\lceil \frac{s_D}{p} \right\rceil$

$$\sum_{i=1}^{D} s_i = W$$

Brent's Theorem: Proof

- Observe: a DAG of depth D can be partitioned to D levels, s.t. there are no interdependencies between nodes at the same level

- At each DAG level i, we define T_p^i as the time for layer i and p processors, s_i $(= T_1^i)$ denotes the number of nodes in level i, thus:

$$T_p^i = \left\lceil \frac{s_i}{p} \right\rceil \leq \frac{s_i + p - 1}{p}$$

- It follows that:

$$T_p(n) \leq \sum_{i=1}^{D} T_p^i \leq \sum_{i=1}^{D} \frac{s_i + p - 1}{p}$$

Brent's Theorem: Proof

$$T_p(n) \leq \sum_{i=1}^{D} \frac{s_i + p - 1}{p} =$$

$$= \frac{1}{p} W(n) + \frac{p-1}{p} D(n) =$$

$$= D(n) + \frac{W(n) - D(n)}{p} \leq D(n) + \frac{W(n)}{p}$$

- To conclude:

$$\boldsymbol{\frac{T_1}{p} = \frac{W(n)}{p} \leq T_p(n) \leq \frac{W(n)}{p} + D(n) = \frac{T_1}{p} + T_\infty}$$

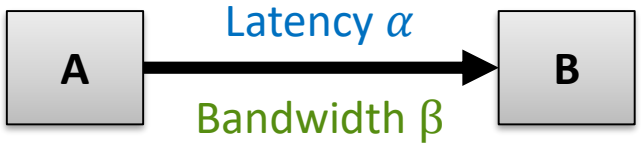
Speedups

- $S_p(n) = \frac{T_1(n)}{T_p(n)} \leq \frac{W(n)}{D(n)}$
- $S_p(n) \leq p$
- $S_p(n) \geq \frac{p}{\frac{D(n)}{W(n)}p+1} \xrightarrow[p \to \infty]{} \frac{W(n)}{D(n)}$

$$\Rightarrow S_\infty(n) = \frac{W(n)}{D(n)}$$

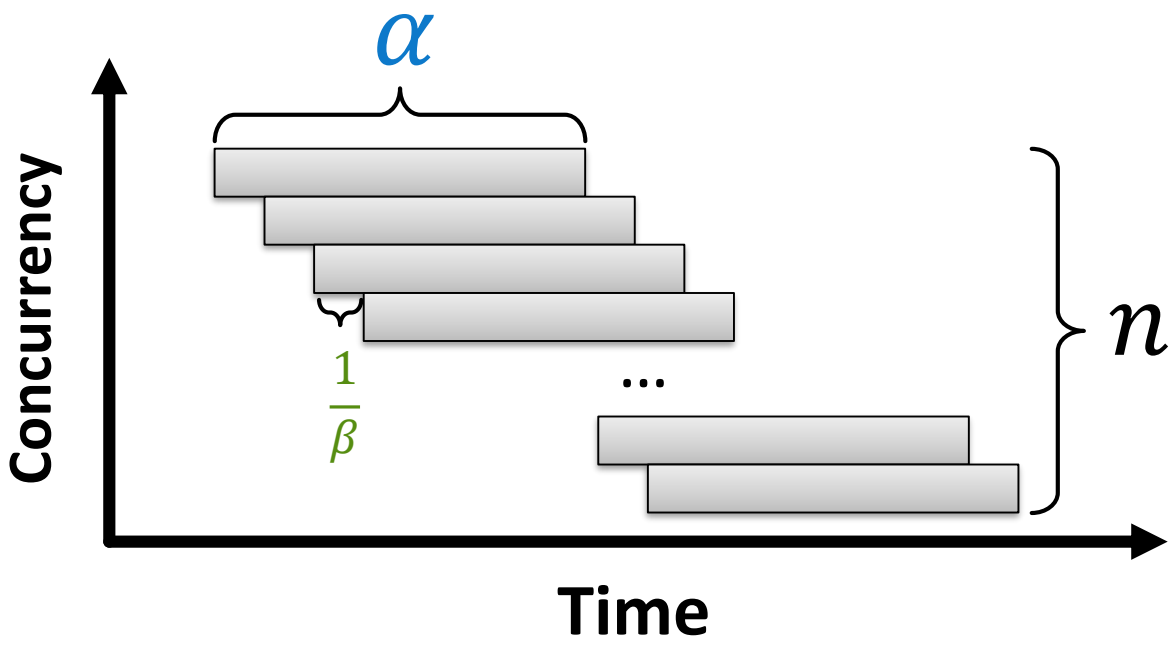
- **For fixed n, speedup is limited. For $n \to \infty$, speedup can be unbounded**
  - Example: Tree reduction

Communication Models: α-β Model

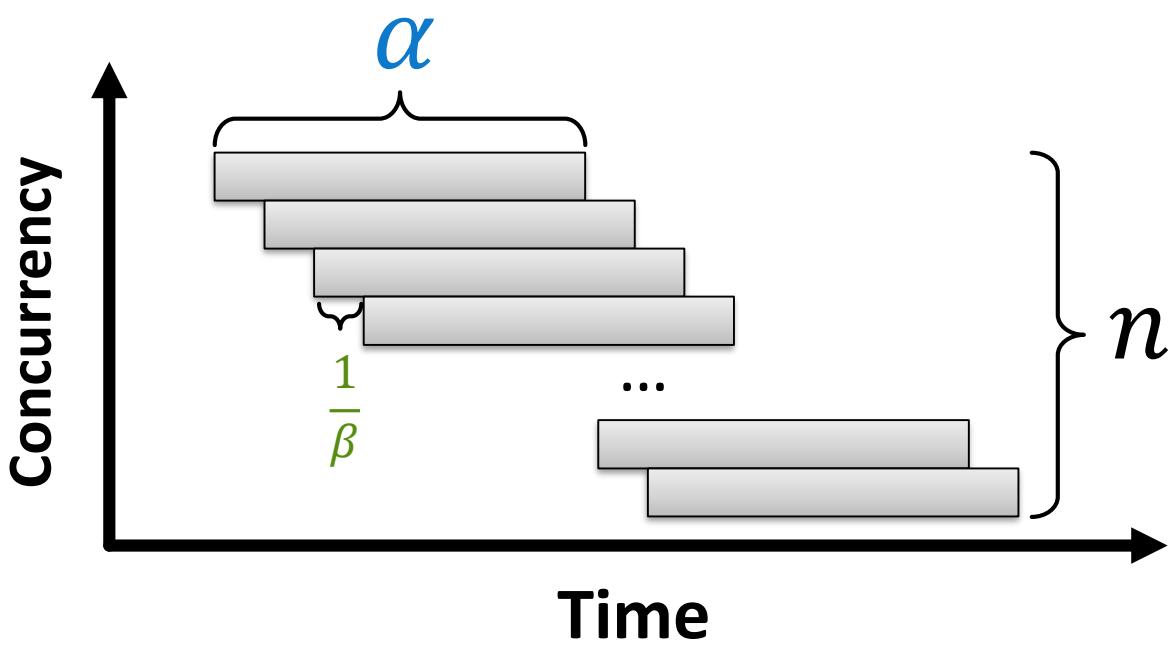


- **Gist:** α = Latency (cycles),
  β = Bandwidth (units/cycle)

- **How long does it take to send a message of size n?**



α-β Model



$$T(n) = \alpha + \frac{n-1}{\beta}$$

Little's Law: Primer

- **Problem 1:**
  - In Tannenbar, every minute <u>on average</u> 2 students come and leave
  - Each student spends 8 minutes inside
  - Q: How many students are in Tannenbar at a given time?

    A: 16

- **Problem 2:**
  - In your fridge, you have 150 bottles of Vivi Kola (on average)
  - You drink and buy on average 25 bottles per week
  - Q: How many weeks does every bottle last?

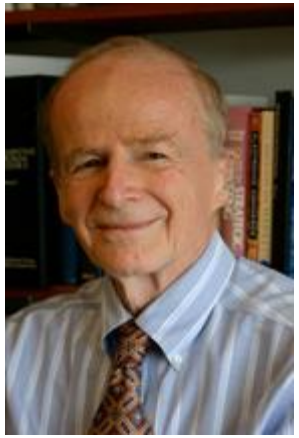
    A: 6

Little's Law

- **Queueing theory**
- **In a stationary system (input rate = output rate), the number of customers N in a queue is given by:**

$$\boldsymbol{N = \lambda W}$$

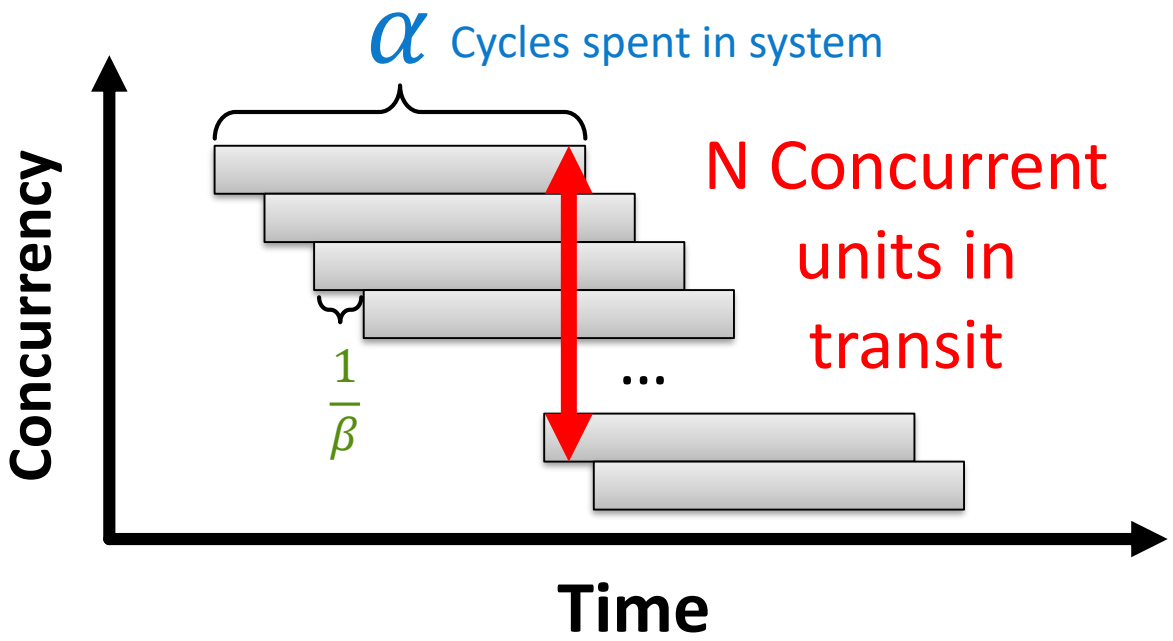
  - Where λ = arrival rate and W = average time spent in the system
  - Simple rule, yet powerful: **N is independent of I/O distribution**



John Little

- **Connection: in the α-β model, arrival rate is β and time spent in system is α cycles**

Little's Law and the α-β Model

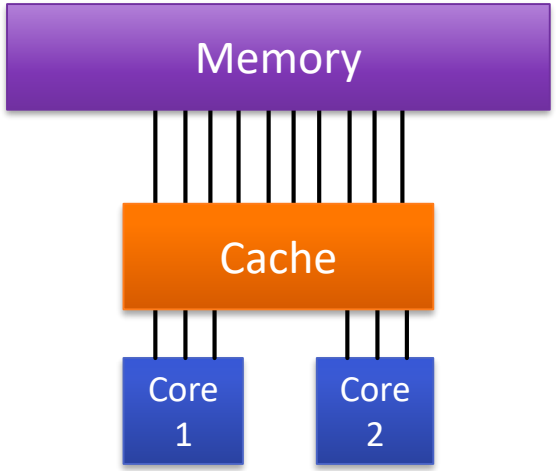


$$N = \alpha \cdot \beta$$

Example: Memory Systems

- **Latency $\times$ Throughput = Concurrency**
  - Latency is reduced by half every 9 years
  - Throughput doubles every 3 years

    ➔ *Parallel processing beneficial!*



- **Real-world statistics:**
  - Intel Core 2 (2006): $\alpha = 100$ cycles, $\beta = 2$ bytes/cycle → N = 200
  - Intel Haswell (2014): $\alpha = 63$ cycles, $\beta = 23$ bytes/cycle → N = 1449