



departamento de informática  
FACULDADE DE CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE NOVA DE LISBOA

# Parallel Performance

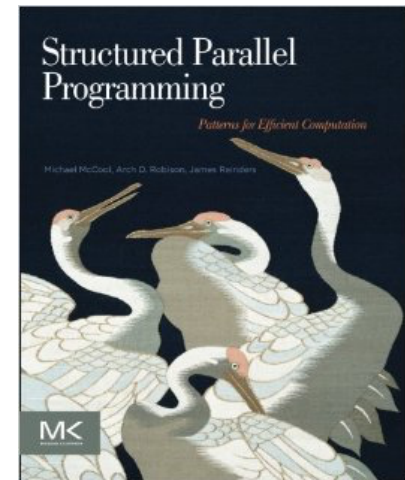
Concurrency and Parallelism — 2016-17  
Master in Computer Science  
(Mestrado Integrado em Eng. Informática)

Joao Lourenço <[joao.lourenco@fct.unl.pt](mailto:joao.lourenco@fct.unl.pt)>

Source: Parallel Computing, CIS 410/510, Department of Computer and Information Science

# Outline

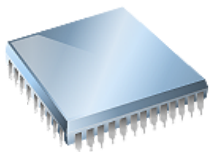
- Performance scalability
  - Analytical performance measures
  - Amdahl's law
  - Gustafson-Barsis' law
  - Work-span and Brent's lemma
- Bibliography:
  - **Chapter 2** of book  
McCool M., Arch M., Reinders J.;  
Structured Parallel Programming: Patterns for  
Efficient Computation;  
Morgan Kaufmann (2012);  
ISBN: 978-0-12-415993-8



# What is Performance?

- In computing, performance is defined by 2 factors
  - Computational requirements (what needs to be done?)
  - Computing resources (how much will it cost?)
- Computational problems translate to requirements
- Computing resources interplay and tradeoff

$$\textit{Performance} \sim \frac{1}{\textit{Resources for solution}}$$



Hardware



Time



Energy

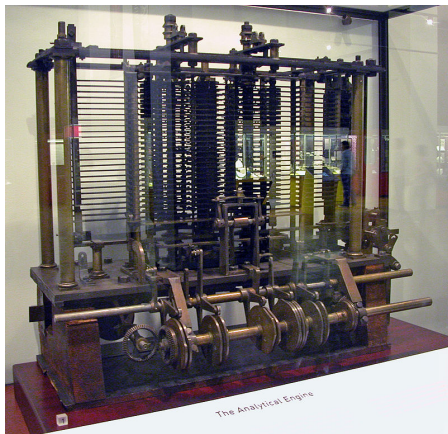
... and ultimately



Money

# Why do we care about Performance?

- Performance itself is a measure of how well the computational requirements can be satisfied
- We evaluate performance to understand the relationships between requirements and resources
  - Decide how to change “solutions” to target objectives
- Performance measures reflect decisions about how and how well “solutions” are able to satisfy the computational requirements



*“The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible.”*

*Charles Babbage, 1791 – 1871*

# What is Parallel Performance?

- Here we are concerned with performance issues when using a parallel computing environment
  - Performance with respect to parallel computation
- Performance is the *raison d'être* for parallelism
  - Parallel performance versus sequential performance
  - If the “performance” is not better, parallelism is not necessary
- *Parallel processing* includes techniques and technologies necessary to compute in parallel
  - Hardware, networks, operating systems, parallel libraries, languages, compilers, algorithms, tools, ...
- Parallelism must deliver performance
  - How? How well?

# Performance Expectation (Loss)

- If each processor is rated at  $x$  MFLOPS and there are  $p$  processors, should we see  $x \cdot p$  MFLOPS performance?
  - If it takes 100 seconds on 1 processor, shouldn't it take 10 seconds on 10 processors?
- Several causes affect performance
  - Each must be understood separately
  - But they interact with each other in complex ways
    - Solution to one problem may create another
    - One problem may mask another
- Scaling (system, problem size) can change conditions
- Need to understand *performance space*

# Embarrassingly Parallel Computations

- An embarrassingly parallel computation is one that can be obviously divided into completely independent parts that can be executed simultaneously
  - In a truly embarrassingly parallel computation there is no interaction between separate processes
  - In a nearly embarrassingly parallel computation results must be distributed and collected/combined in some way
- Embarrassingly parallel computations have potential to achieve maximal speedup on parallel platforms
  - If it takes  $T$  time sequentially, there is the potential to achieve  $T/P$  time running in parallel with  $P$  processors
  - Why is this not always the case?

# Scalability

---

- A program can scale up to use many processors
  - What does that mean?
- How do you evaluate scalability?
- How do you evaluate scalability goodness?
- Comparative evaluation
  - If double the number of processors, what to expect?
  - Is scalability linear?
- Use parallel efficiency measure
  - Is efficiency retained as problem size increases?
- Apply performance metrics



# Performance and Scalability

- Evaluation

- *Sequential* runtime ( $T_{seq}$  or  $T_1$ ) is a function of
  - problem size and architecture
- *Parallel* runtime ( $T_{par}$ ) is a function of
  - problem size and parallel architecture
  - # processors used in the execution
- Parallel performance is affected by
  - algorithm + architecture

- Scalability

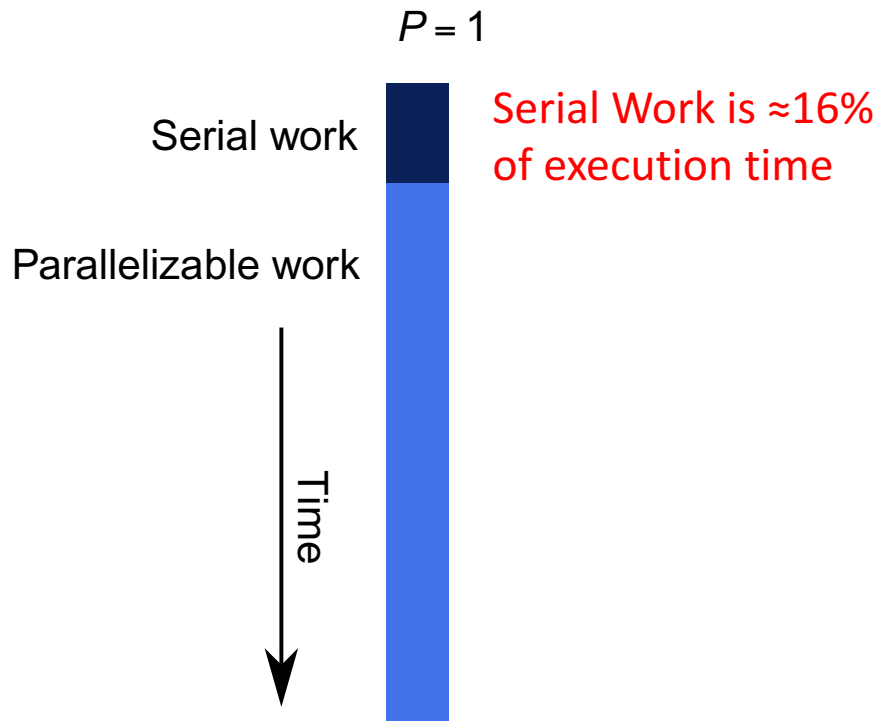
- Ability of parallel algorithm to achieve performance gains proportional to the number of processors and the size of the problem

# Performance Metrics and Formulas

- $T_1$  is the execution time on a single processor
  - $T_p$  is the execution time on a  $p$  processor system
  - $S_p$  is the speedup  $S(p) = \frac{T_1}{T_p}$
  - $E_p$  is the efficiency  $E(p) = \frac{S_p}{p}$
  - $C_p$  is the cost  $Cost(p) = p \times T_p$
- A parallel algorithm is *cost-optimal* if
    - $\sum$  Parallel time = sequential time ( $C_p = T_1$  ,  $E_p = 100\%$ )

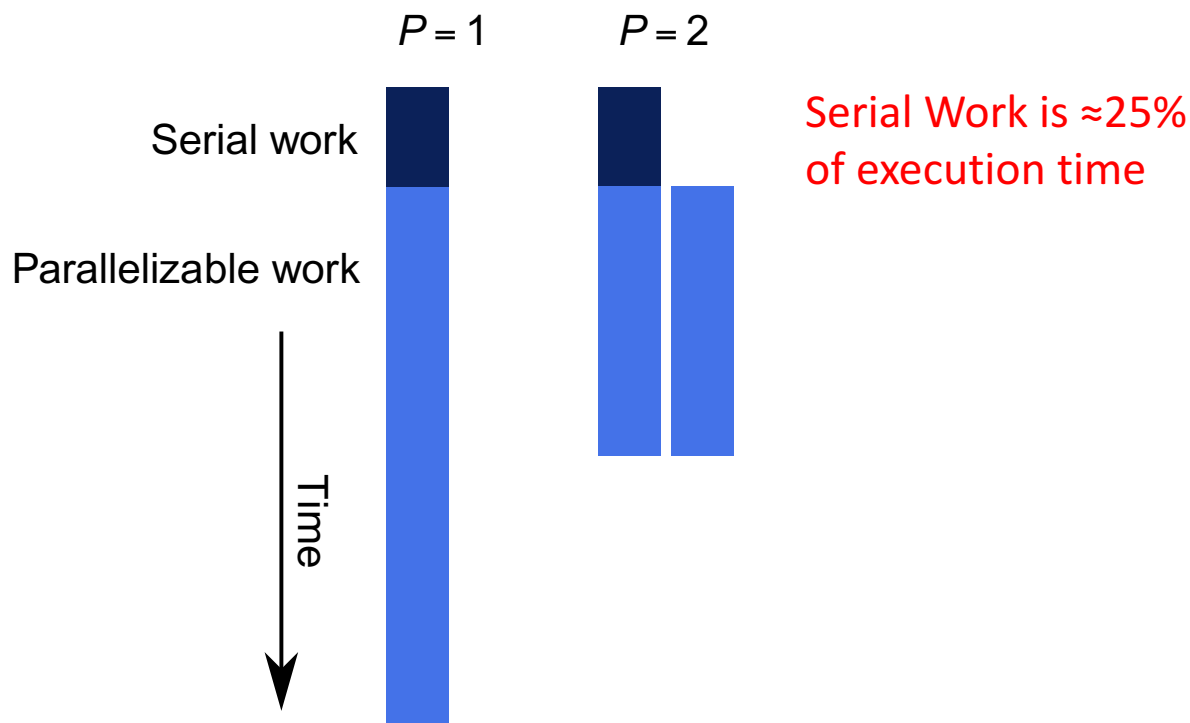
# Amdahl's Law (Fixed Size Speedup)

- Interested in solving the problem faster
- *Reduce execution time*



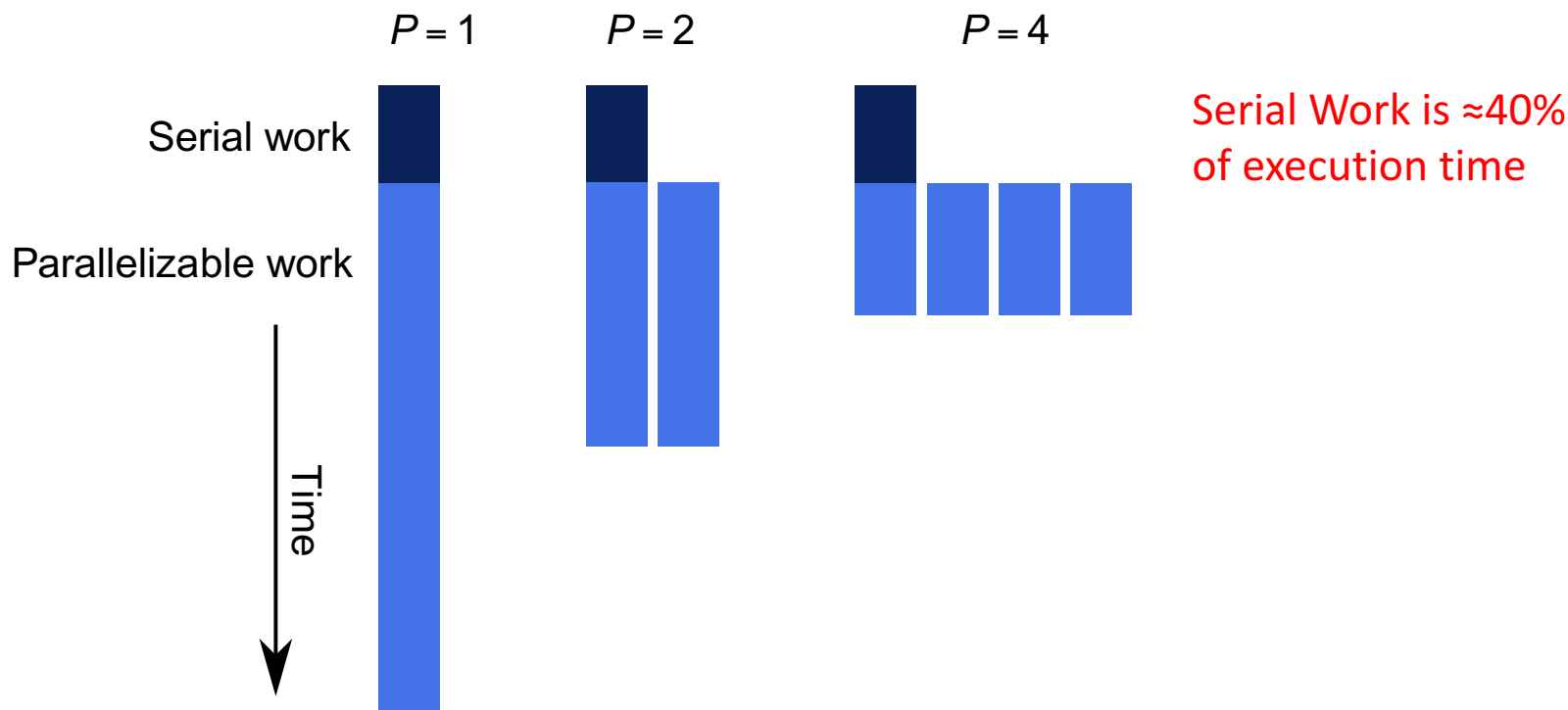
# Amdahl's Law (Fixed Size Speedup)

- Interested in solving the problem faster
- *Reduce execution time*



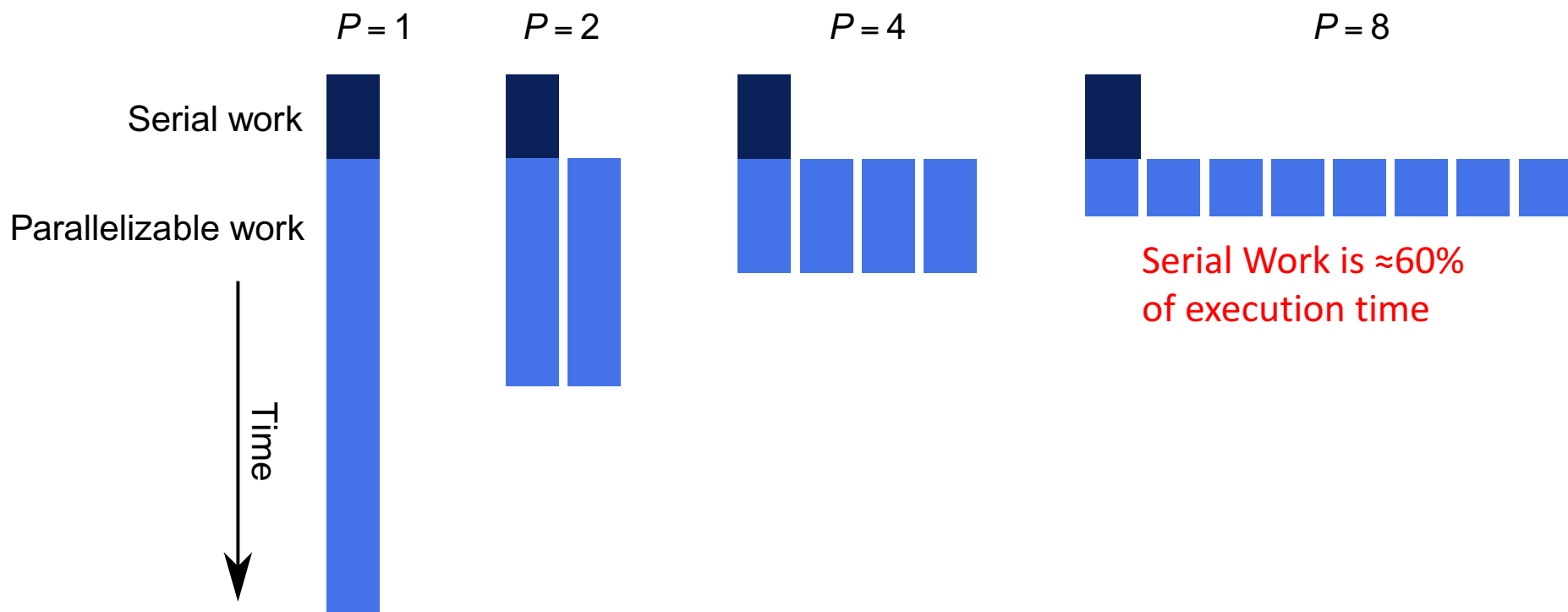
# Amdahl's Law (Fixed Size Speedup)

- Interested in solving the problem faster
- *Reduce execution time*



# Amdahl's Law (Fixed Size Speedup)

- Interested in solving the problem faster
- *Reduce execution time*



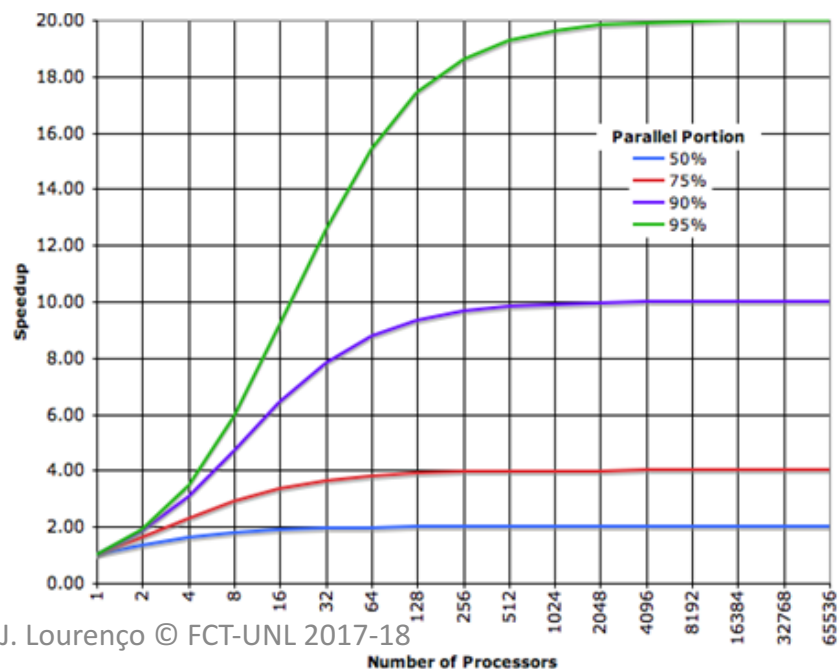
# Amdahl's Law (Fixed Size Speedup)

- Let  $f$  be the fraction of a program that is sequential
  - $1-f$  is the fraction that can be parallelized
- Let  $T_1$  be the execution time on 1 processor
- Let  $T_p$  be the execution time on  $p$  processors
- $S_p$  is the *speedup*

$$S_p \leq \frac{T_1}{T_p} = \frac{T_1}{fT_1 + \frac{(1-f)T_1}{p}}$$

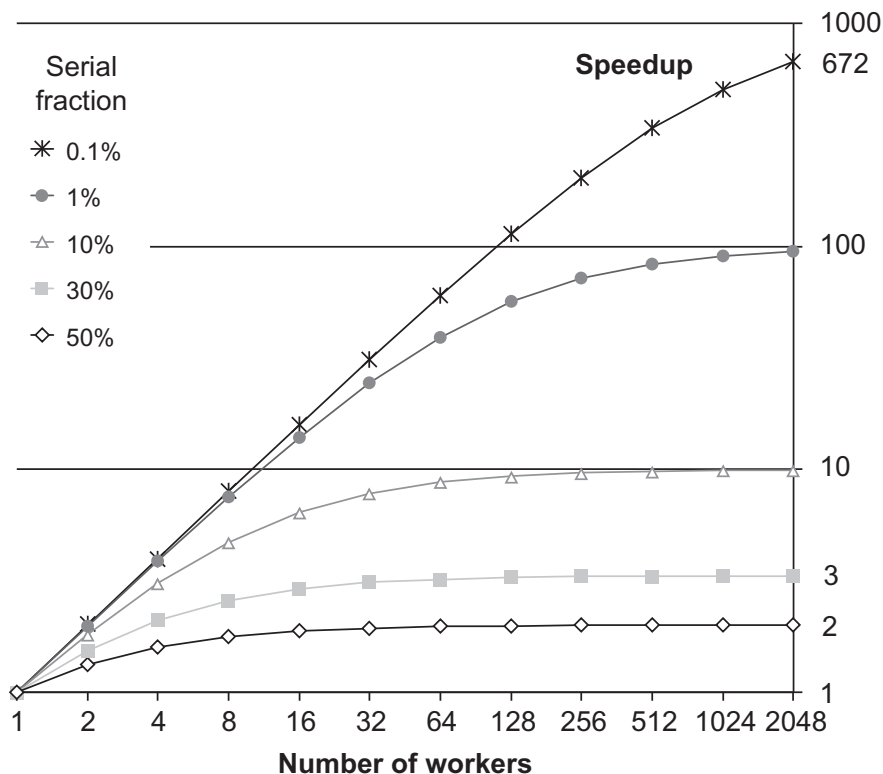
$$S_p \leq \frac{1}{f + \frac{(1-f)}{p}}$$

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



# Amdahl's Law (Fixed Size Speedup)

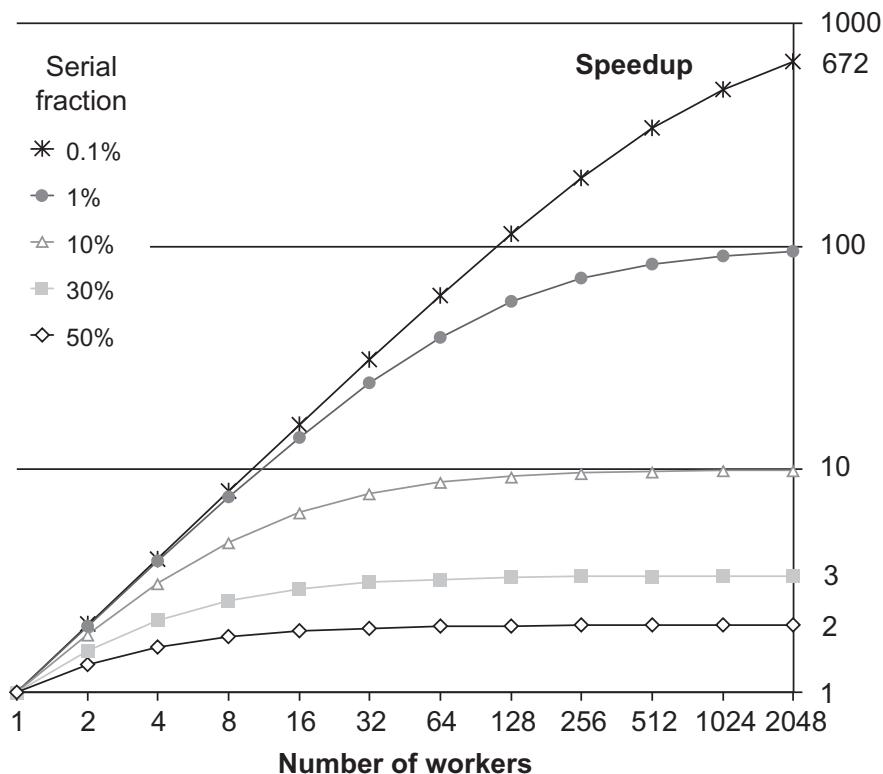
- Amdahl's Law:  
Maximal Speedup



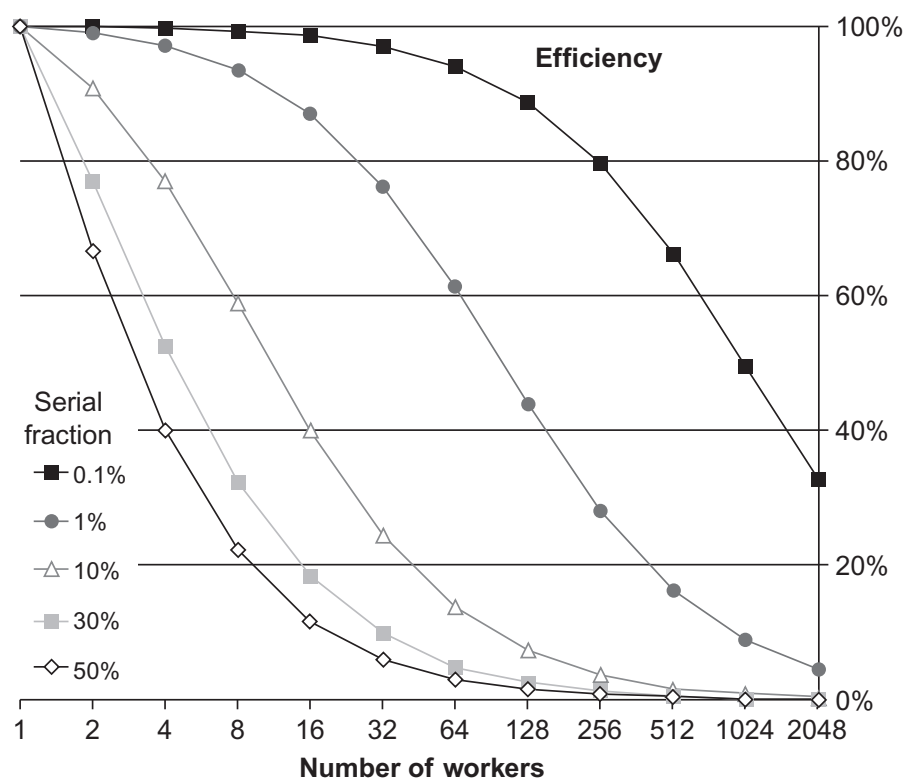


# Amdahl's Law (Fixed Size Speedup)

- Amdahl's Law:  
Maximal Speedup



- Amdahl's Law:  
Efficiency



# Amdahl's Law (Exmple)

- If 90% of the computation can be parallelized, what is the max. speedup achievable using 8 processors?
- Solution:

$$f = 10\% = 0.1$$

$$S(8) \leq \frac{1}{0.1 + \frac{1-0.1}{8}} \approx 4.7$$

# Amdahl's Law and Scalability

- Scalability
  - Ability of parallel algorithm to achieve performance gains proportional to the number of processors and the size of the problem
- When does Amdahl's Law apply?
  - When the problem size is fixed
  - *Strong scaling* ( $p \rightarrow \infty, S_p = S_\infty \rightarrow 1 / f$ )
  - Speedup bound is determined by the degree of sequential execution time in the computation, not # processors!!!
  - Uhh, this is not good ... Why?
  - Perfect efficiency is hard to achieve
- See original paper by Amdahl at
  - <http://inst.eecs.berkeley.edu/~n252/sp07/Papers/Amdahl.pdf>

# Gustafson-Barsis' Law (Scaled Speedup)

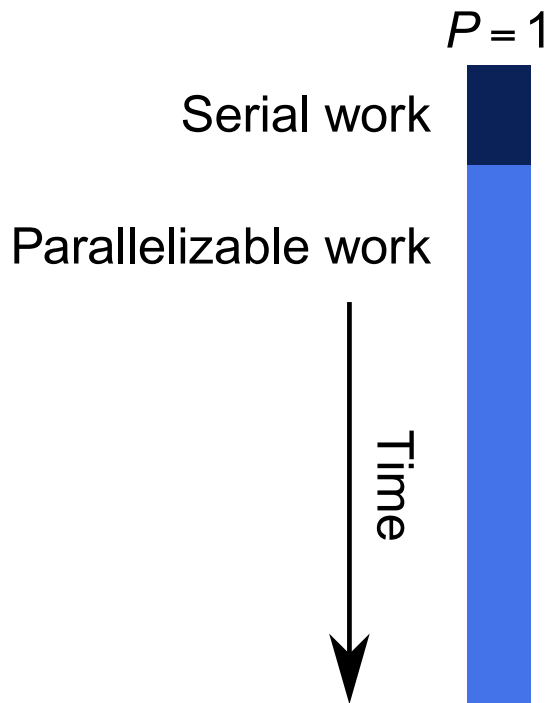
---

*...speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.*

— John Gustafson

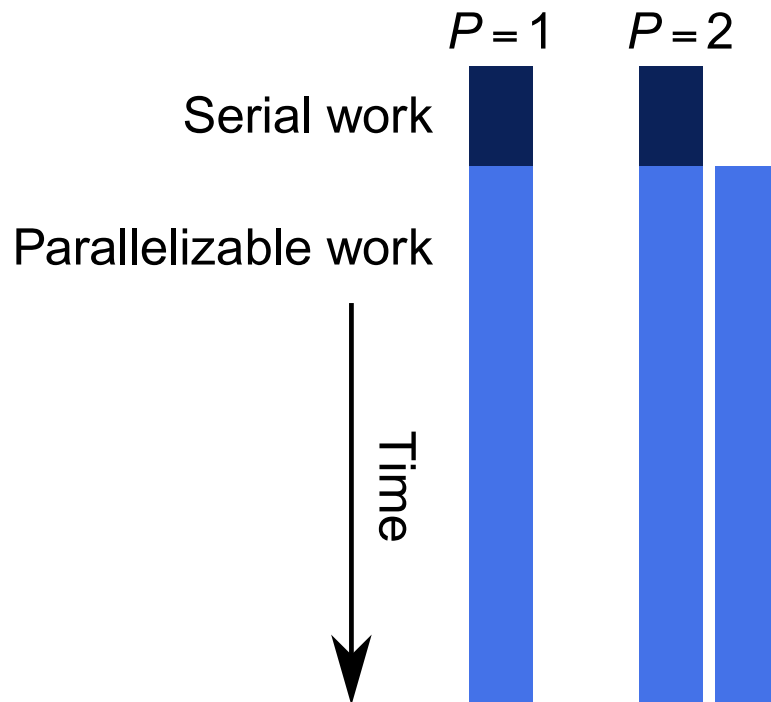
# Gustafson-Barsis' Law (Scaled Speedup)

- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time



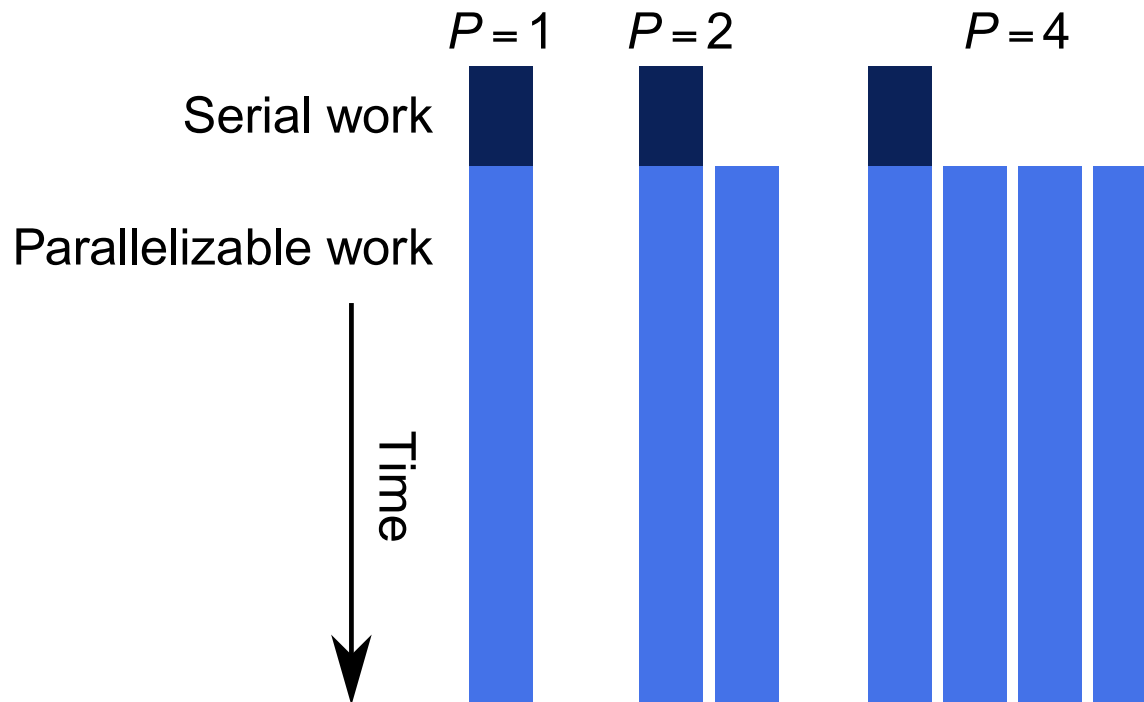
# Gustafson-Barsis' Law (Scaled Speedup)

- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time



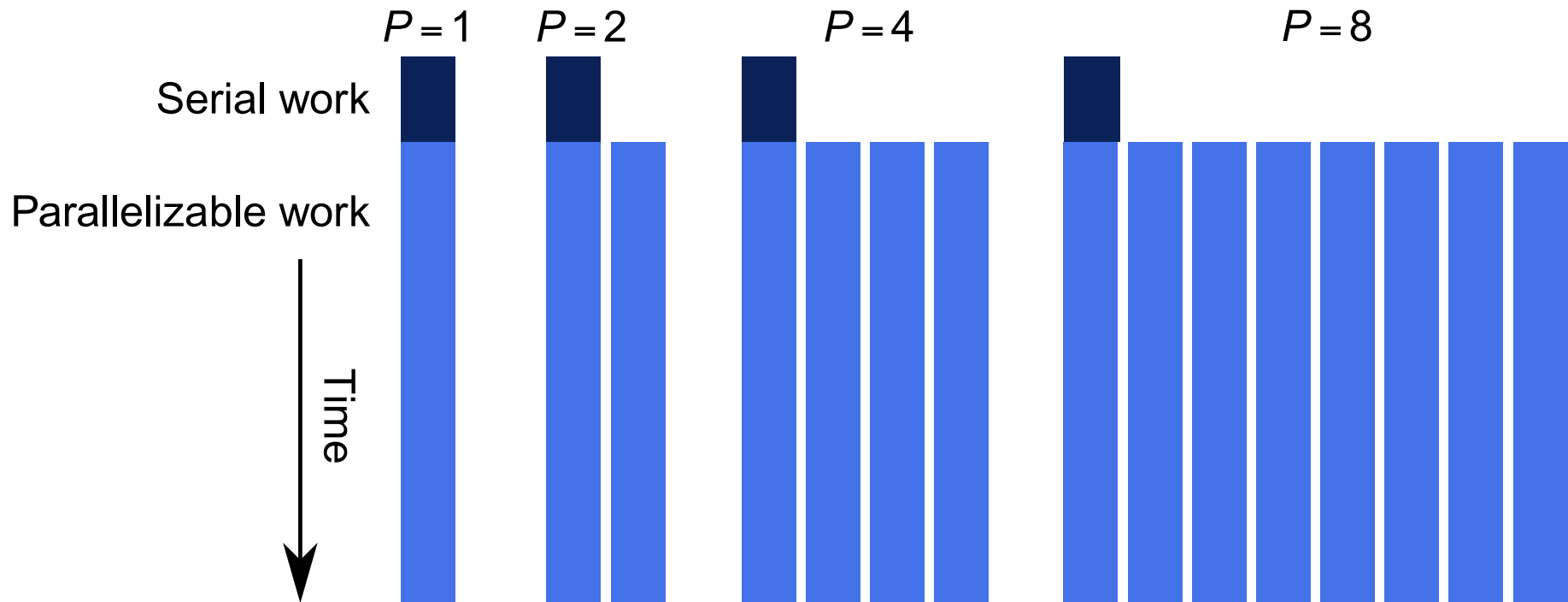
# Gustafson-Barsis' Law (Scaled Speedup)

- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time



# Gustafson-Barsis' Law (Scaled Speedup)

- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time





# Gustafson-Barsis' Law (Scaled Speedup)

- Execution time of a parallel program:  $T_1 = a + b$ 
  - $a \Rightarrow$  part not parallelizable
  - $b \Rightarrow$  part parallelizable
- Because we are scaling the problem (data being processed), with  $P$  processors we have:  
 $T_P = a + P \cdot b$
- The wall clock execution time is always the same, so scaled speedup is calculated on the volume of data processed (which is proportional to the total/accumulated execution time):  
$$S = T_P / T_1 = (a + P \cdot b) / (a + b)$$

# Gustafson-Barsis' Law (Scaled Speedup)

- Scaled speedup  $S = T_p / T_1 = (a + P \cdot b) / (a + b)$
- Let  $\alpha = a / (a+b)$  be the sequential fraction of the parallel execution time
- Then the **scaled speedup** is

$$S(P) \leq \alpha + P \cdot (1 - \alpha) = P - \alpha \cdot (P-1)$$

- If  $\alpha \rightarrow 0$  then  $S(P) \rightarrow P$

# Gustafson-Barsis' Law (Example)

- An application executing on 64 processors uses 5% of the total time on non-parallelizable computations. What is the scaled speedup?
- Solution:

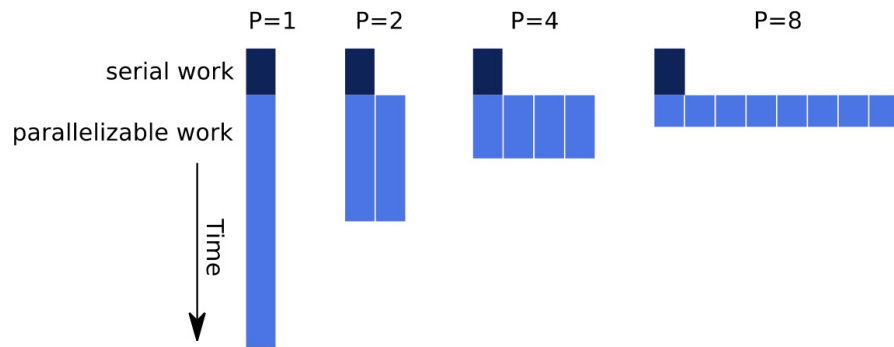
$$\begin{aligned} S(64) &\leq P - \alpha \cdot (P-1) \\ &\leq 64 - 0.05 (64-1) \\ &\leq 60.85 \end{aligned}$$

# Gustafson-Barsis' Law and Scalability

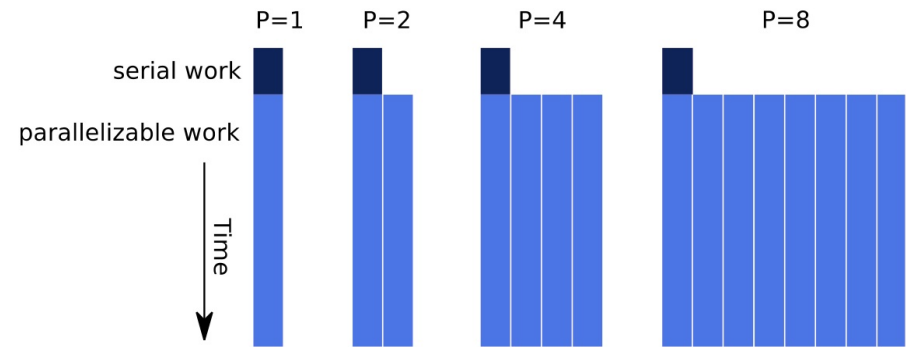
- Scalability
  - Ability of parallel algorithm to achieve performance gains proportional to the number of processors and the size of the problem
- When does Gustafson's Law apply?
  - When the problem size can increase when the number of processors increases
  - Speedup function includes the number of processors!!!
  - Can maintain or increase parallel efficiency as the problem scales

# Amdahl versus Gustafson-Baris

Amdahl

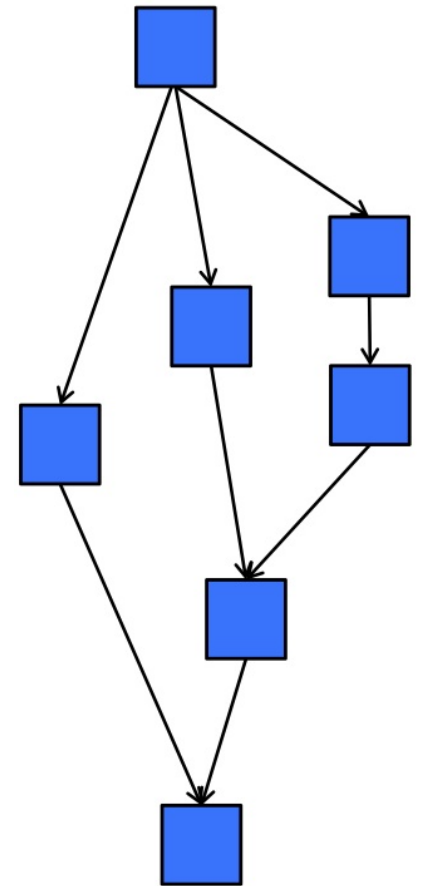


Gustafson-Baris



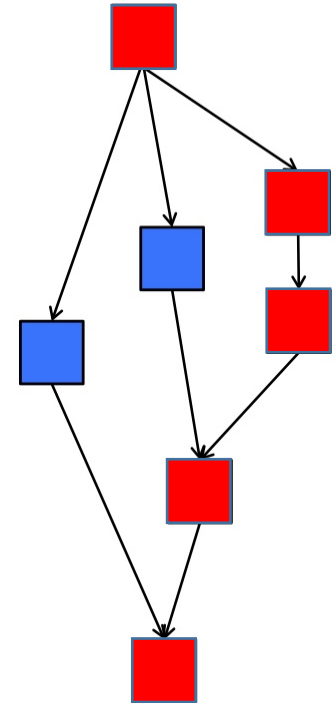
# DAG Model of Computation

- Think of a program as a directed acyclic graph (DAG) of tasks
  - A task can not execute until all the inputs to the tasks are available
  - These come from outputs of earlier executing tasks
  - DAG shows explicitly the task dependencies
- Think of the hardware as consisting of workers (processors)
- Consider a *greedy* scheduler of the DAG tasks to workers
  - No worker is idle while there are tasks still to execute



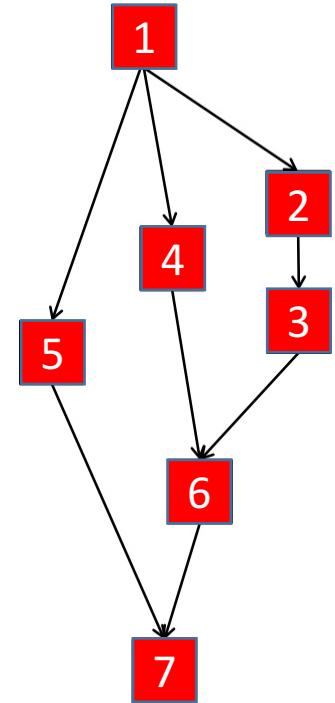
# Work-Span Model

- $T_P$  = time to run with  $P$  workers
- $T_1$  = *work*
  - Time for serial execution
    - execution of all tasks by 1 worker
  - Sum of all work
- $T_\infty$  = *span*
  - Time along the *critical path*
- Critical path
  - Sequence of task execution (path) through DAG that takes the longest time to execute
  - Assumes an infinite # workers available



# Work-Span Example

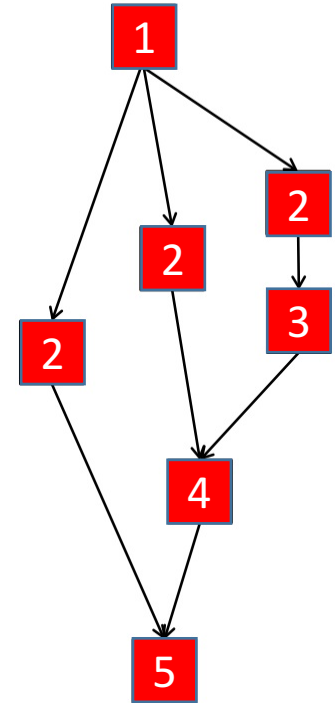
- DAG at the right has 7 tasks
- Let each task take 1 unit of time
- $T_1 = 7$ 
  - All tasks have to be executed
  - Tasks are executed in a serial order
  - Can them execute in any order?





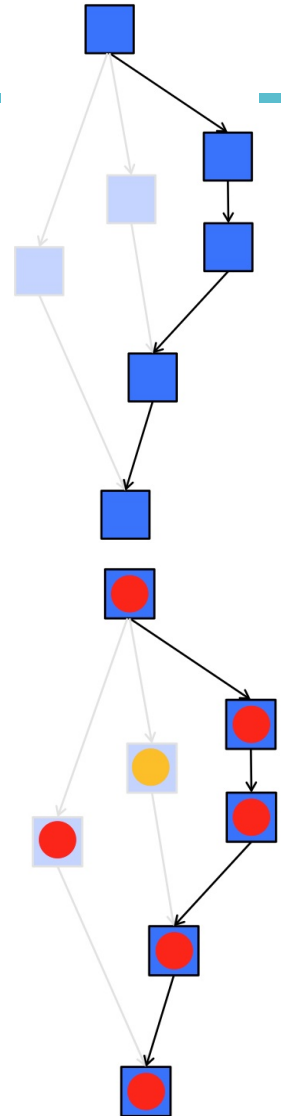
# Work-Span Example

- DAG at the right has 7 tasks
- Let each task take 1 unit of time
- $T_1 = 7$ 
  - All tasks have to be executed
  - Tasks are executed in a serial order
  - Can them execute in any order?
- $T_\infty = 5$ 
  - Time along the *critical path*
  - In this case, it is the longest pathlength of any task order that maintains necessary dependencies



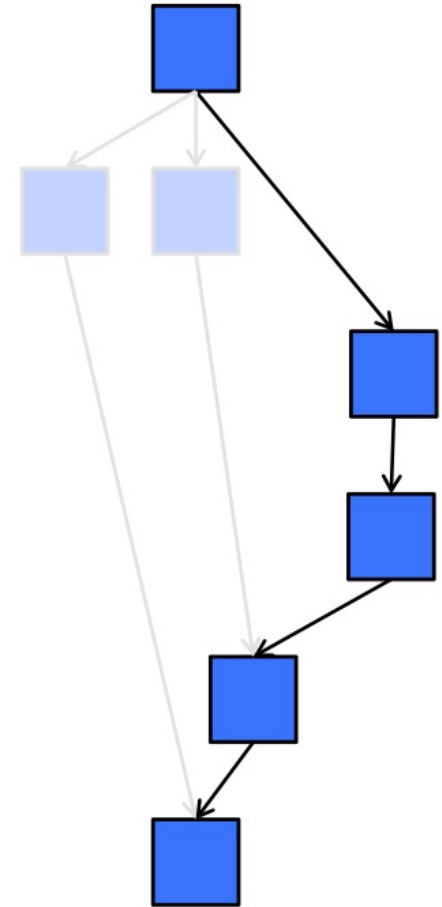
# Lower/Upper Bound on Greedy Scheduling

- Suppose we only have  $P$  workers
- We can write a work-span formula to derive a lower bound on  $T_P$ 
  - $\text{Max}(T_1 / P, T_\infty) \leq T_P$
- $T_\infty$  is the best possible execution time
- Brent's Lemma derives an upper bound
  - Capture the additional cost executing the other tasks not on the critical path
  - Assume can do so without overhead
  - $T_P \leq (T_1 - T_\infty) / P + T_\infty$



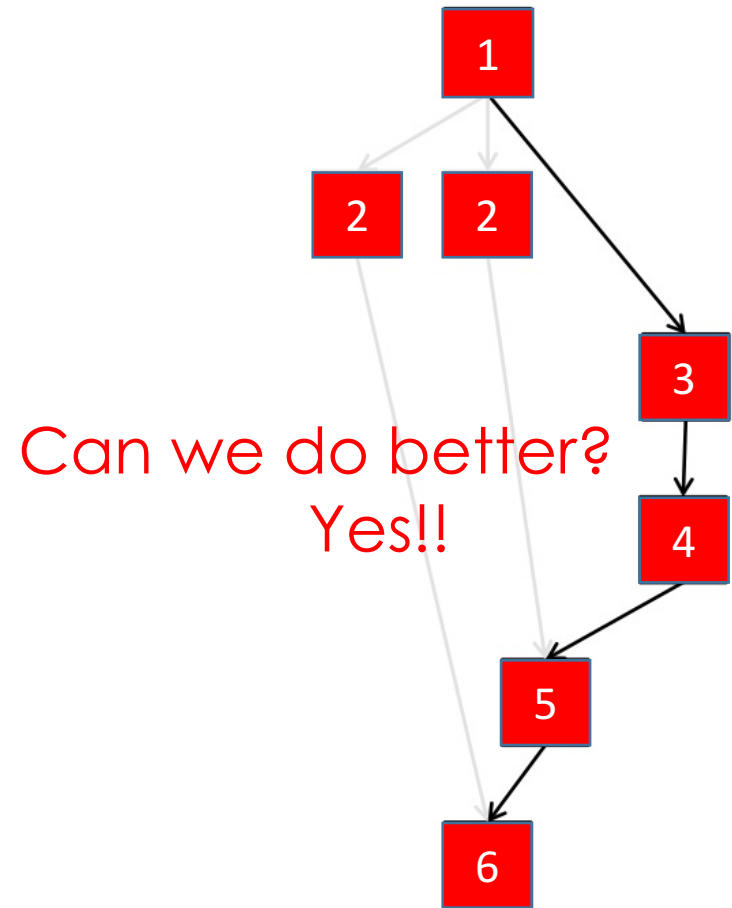
# Consider Brent's Lemma for 2 Processors

- $T_1 = 7$
- $T_\infty = 5$
- $T_2 \leq (T_1 - T_\infty) / P + T_\infty$   
 $\leq (7 - 5) / 2 + 5$   
 $\leq 6$



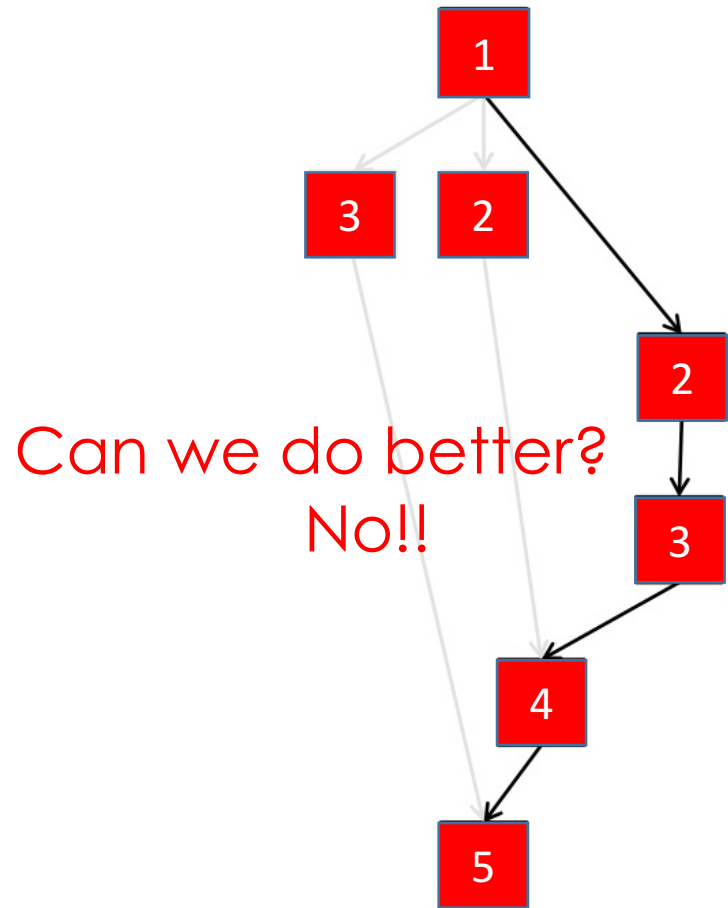
# Consider Brent's Lemma for 2 Processors

- $T_1 = 7$
- $T_\infty = 5$
- $T_2 \leq (T_1 - T_\infty) / P + T_\infty$   
 $\leq (7 - 5) / 2 + 5$   
 $\leq 6$

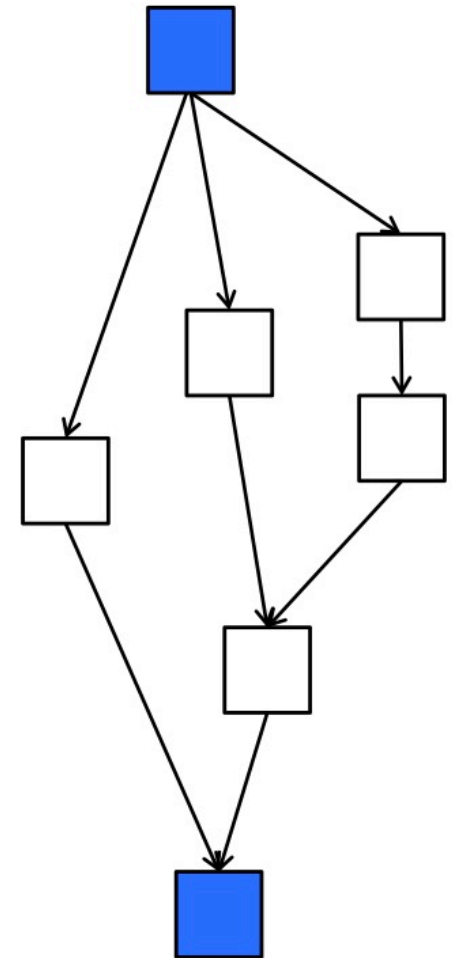
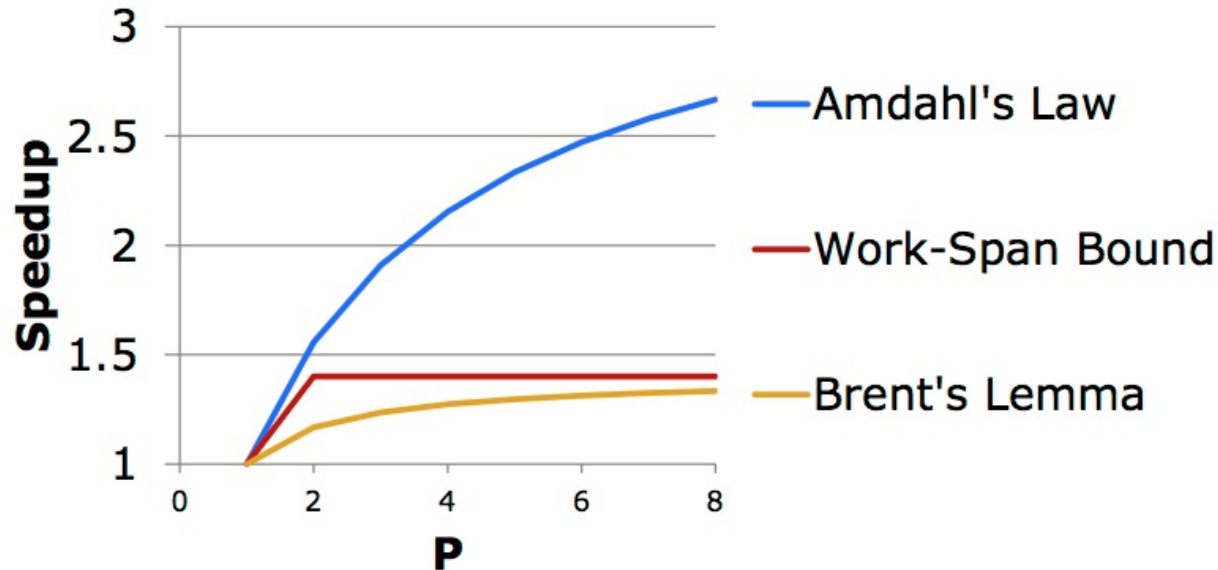


# Consider Brent's Lemma for 2 Processors

- $T_1 = 7$
- $T_\infty = 5$
- $T_2 \leq (T_1 - T_\infty) / P + T_\infty$   
 $\leq (7 - 5) / 2 + 5$   
 $\leq 6$



# Amdahl was an optimist!



# Estimating Running Time

- Scalability requires that  $T_\infty$  be dominated by  $T_1$   
$$T_P \leq (T_1 - T_\infty) / P + T_\infty$$


$$T_P \approx T_1 / P + T_\infty \quad \text{if} \quad T_\infty \ll T_1$$


- Increasing work hurts parallel execution proportionately
- The span impacts scalability, even for finite  $P$

# Parallel Slack

- Sufficient parallelism implies linear speedup

$$T_p \approx T_1/P \quad \text{if} \quad T_1/T_\infty \gg P$$

  
Linear speedup

  
Parallel slack



# The END

---