
Performance Analysis of Local Synchronization

Julia Lipman

Synchronization

- A general problem that appears in many contexts
 - Any time distributed agents can communicate information about their individual states to each other
 - Waiting for this information can cause system slowdowns
-

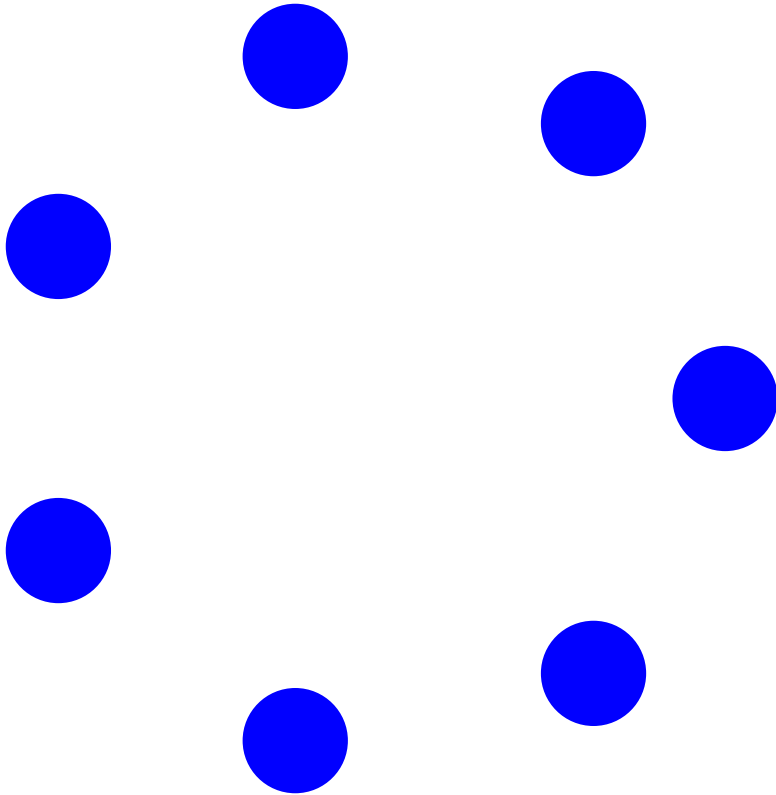
Synchronization

- A general problem that appears in many contexts
- Any time distributed agents can communicate information about their individual states to each other
- Waiting for this information can cause system slowdowns

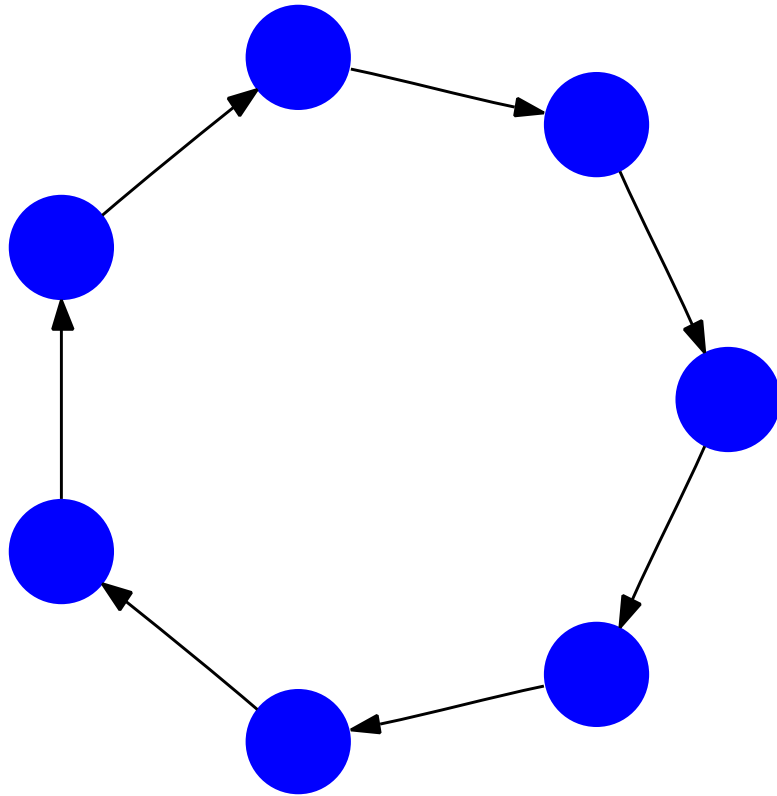
How bad are these slowdowns?

Types of synchronization

- no synchronization

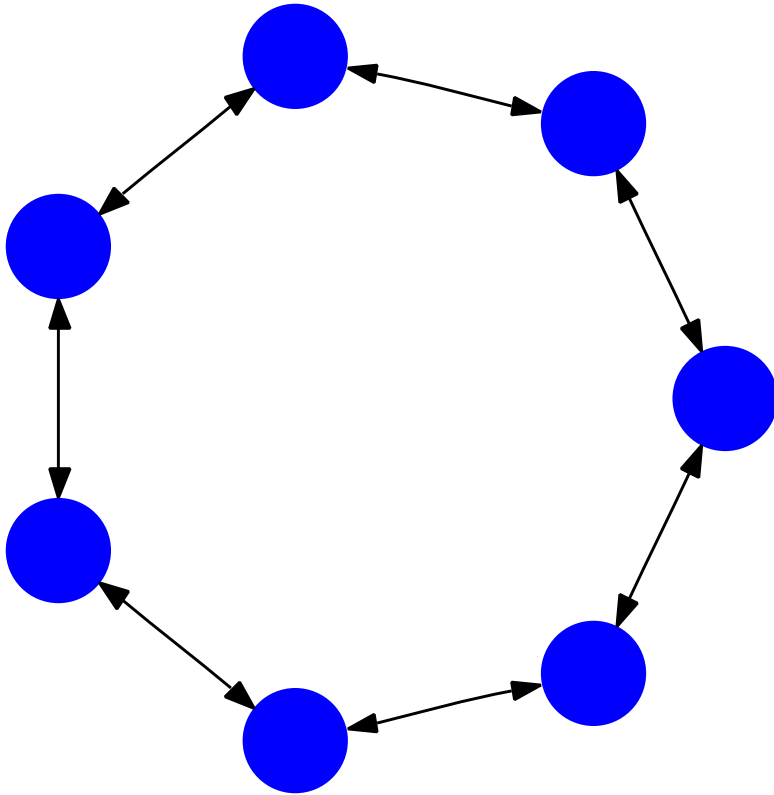


Types of synchronization



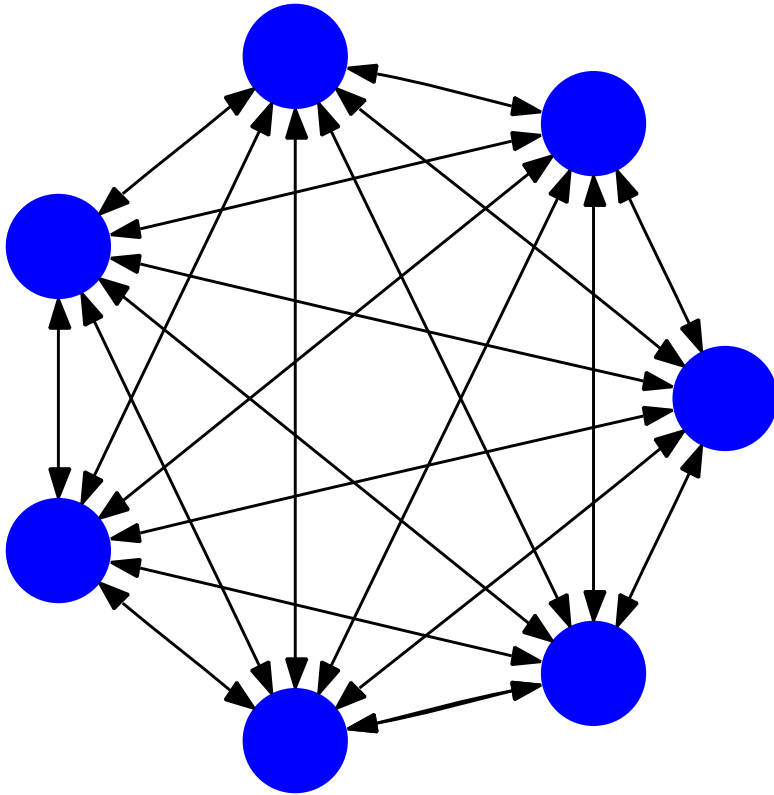
- no synchronization
- directed cycle

Types of synchronization



- no synchronization
- directed cycle
- undirected cycle

Types of synchronization



- no synchronization
- directed cycle
- undirected cycle
- complete graph

Formulation of the problem

- There are n processors completing successive tasks that take identically distributed amounts of time
- Processors are connected in a graph
- A processor must wait to start on its $(i + 1)$ st task until every processor from which it has an incoming edge has finished its i th

Definition

- Let G be the synchronization graph, and F the distribution of task times
- $\mathcal{T}_{G,F}(i)$ denotes a processor's expected time per task when all processors have completed the i th task.
- We define the (asymptotic) *expected time per task* as follows:

$$\mathcal{T}_{G,F}^* = \lim_{i \rightarrow \infty} \mathcal{T}_{G,F}(i)$$

Global vs. local synchronization

- Global (or barrier) refers to a complete graph; local to any other connected graph
 - Sometimes global is used when complete information about the state of a system is needed
 - Local (or partial) must clearly be at least as fast
 - However, global is often used because it is easier to implement
-

Previous work

- The n th moment of a distribution F is $E(X^n)$, where X is a random variable with distribution F
 - Chang and Nelson (1995) showed that if all moments of F are finite (i.e., it's not a heavy-tailed distribution) and G is of bounded degree, then $\mathcal{T}_{G,F}^*$ is bounded above by a constant as the number n of processors goes to infinity
 - However, their bound does not have a closed form in general
-

Previous work

"The difficulty of the analysis lies in the fact that the times processors start their i th iteration differ, which is not the case for barrier synchronization. This makes an exact analysis intractable..."

– Chang and Nelson

Overview

- Geometric and power-law distribution
- Exact formal analysis of the combinatorial properties of the geometric distribution on a directed cycle
- A lower bound for the power-law distribution

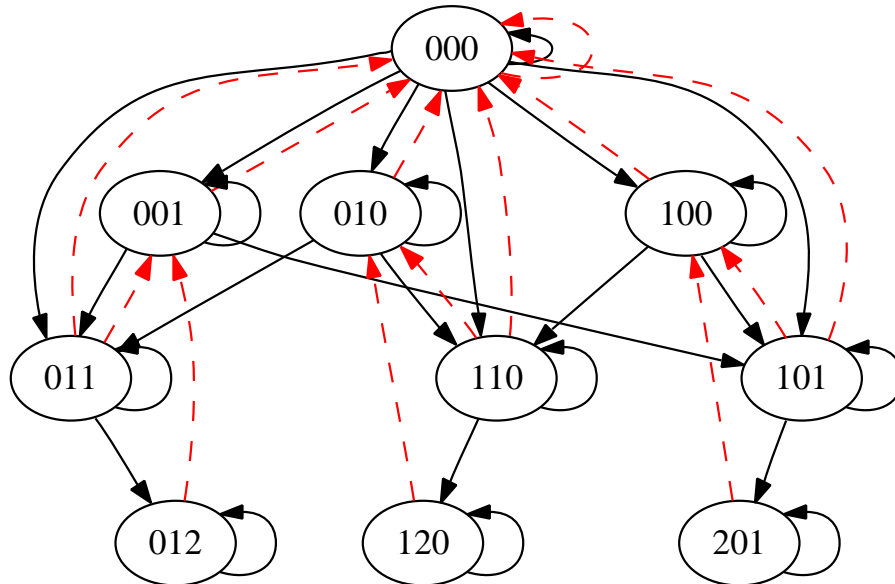
The geometric distribution on a directed cycle

- Each task involves flipping a fair coin until it comes up heads
 - Processors are connected in a directed cycle
 - Can model the system as a Markov chain where, for example, 001 refers to a state where the first two processors have not completed any tasks and the third one has completed one
-

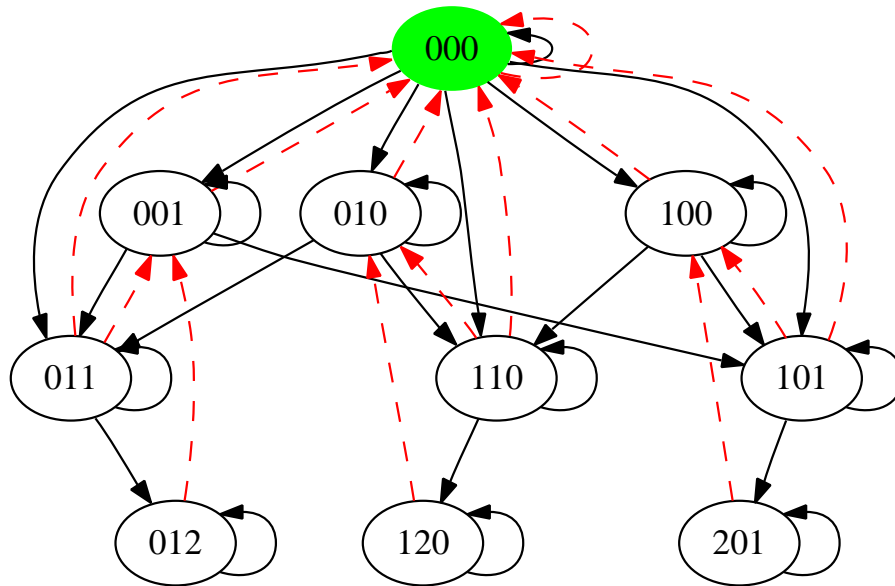
Markov chain for the geometric distribution

- In order to keep the chain from going on forever, we introduce a *rezeroing* operation that decrements every processor's number when every processor has completed a task
- For example, 112 would rezero to 001

Markov chain for the geometric distribution

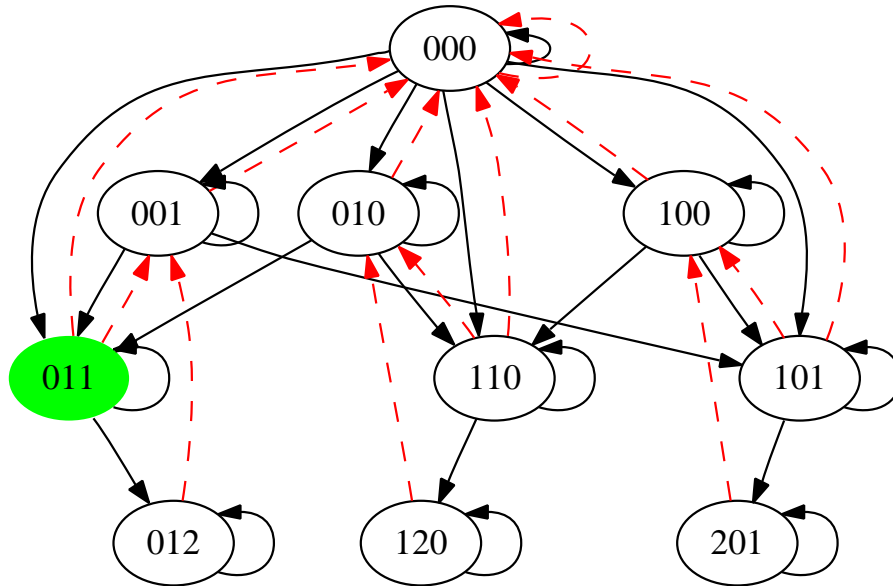


Markov chain for the geometric distribution



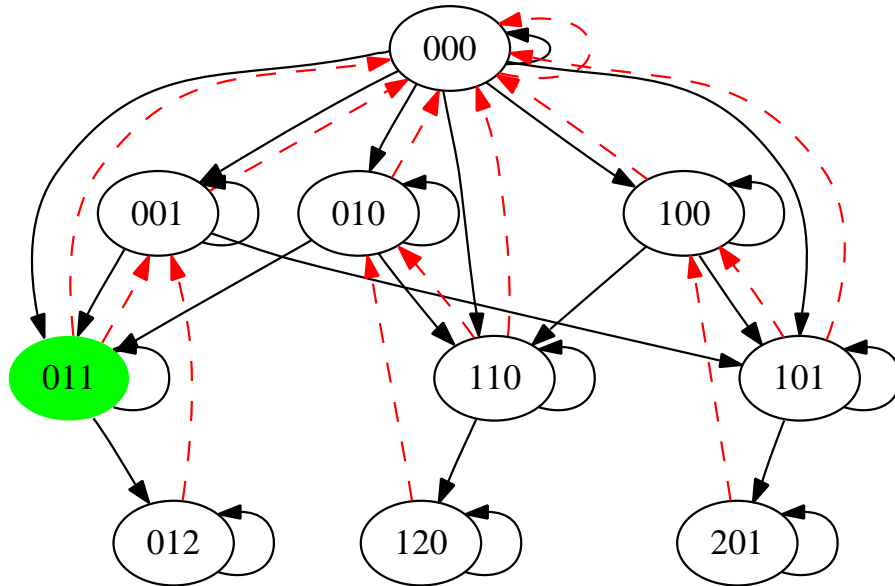
p_0 p_1 p_2
processor

Markov chain for the geometric distribution



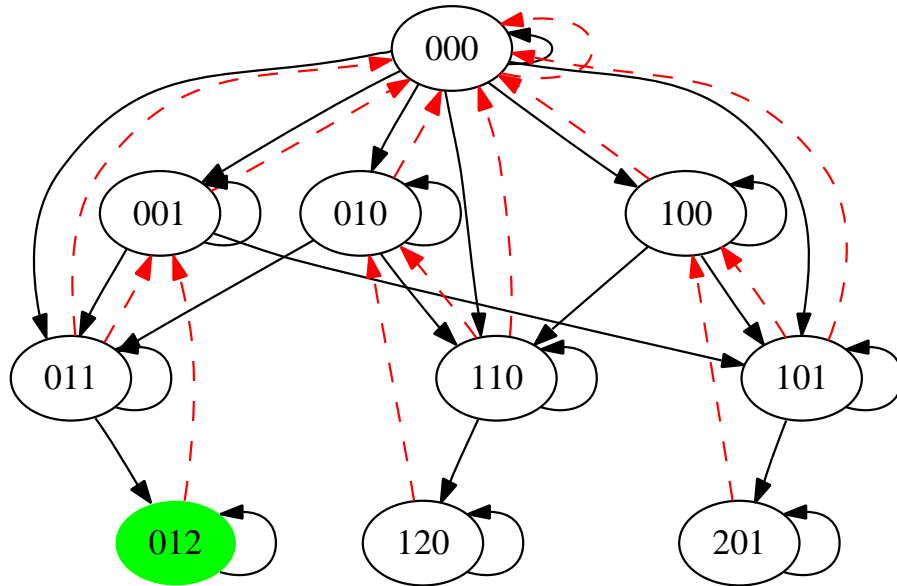
T	H	H
p_0	p_1	p_2
processor		

Markov chain for the geometric distribution



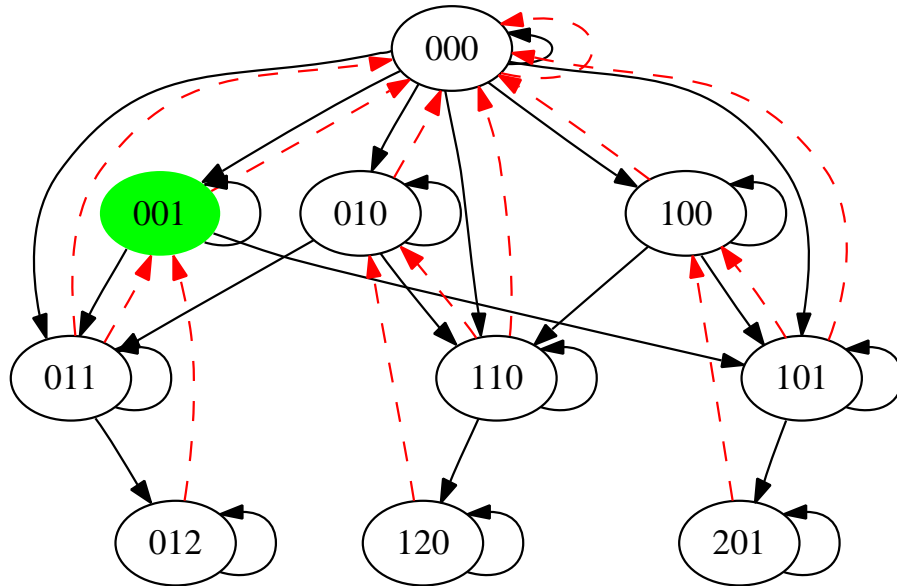
T	-	T
T	H	H
p_0	p_1	p_2
processor		

Markov chain for the geometric distribution



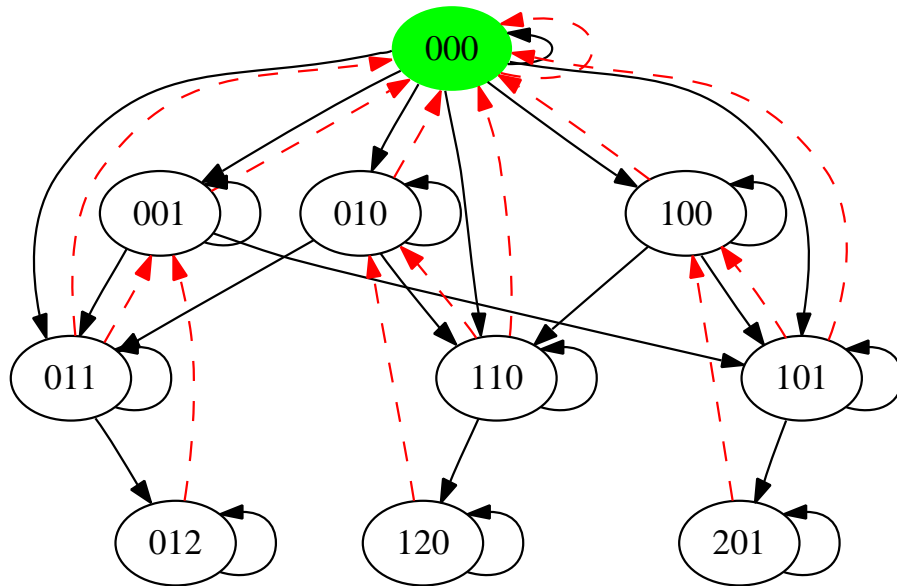
T	-	H
T	-	T
T	H	H
p_0	p_1	p_2
processor		

Markov chain for the geometric distribution



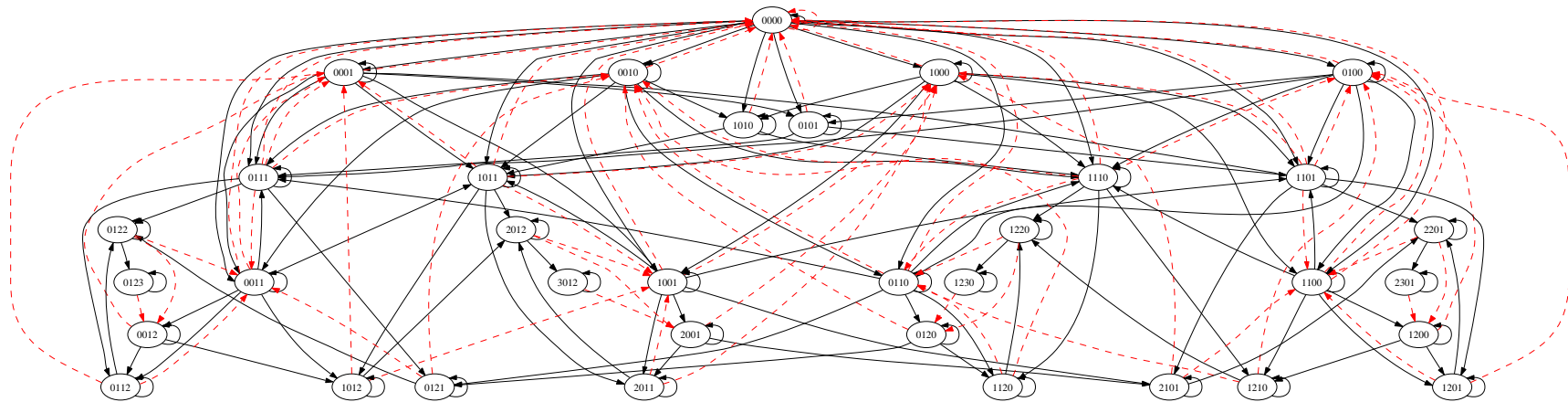
H	-	-
T	-	H
T	-	T
T	H	H
p_0	p_1	p_2
processor		

Markov chain for the geometric distribution



H	H	-
H	-	-
T	-	H
T	-	T
T	H	H
<hr/>		
p_0	p_1	p_2
processor		

Markov chain for n=4



- The chain for n processors has $\frac{\binom{2n}{n}}{2}$ states!
- Need to find some way to analyze the chain without looking at it directly

Lemma 1

The Markov chain for the directed cycle with geometrically distributed tasks is a balanced digraph; that is, each state has the same indegree as its outdegree.

Explanation of Lemma 1

- The incoming edges correspond to the number of processors that could have just finished a task (*last-minute* processors)
 - The outgoing edges correspond to the number of processors that are not waiting for a neighbor (*free* processors)
 - There is a bijection between free and last-minute processors
-

Implication of Lemma 1

Since the Markov chain is balanced, the steady-state probability of any state is proportional to its degree

Time between levels

- We want to find the expected time per task, $\mathcal{T}_{G,F}^*$
- By finding the expected proportion of free processors, we can calculate by symmetry the proportion of time that each processor is expected to be free

Theorem 1

There are $\binom{n}{k} \binom{n-1}{k-1}$ states with exactly k free processors in the n -processor Markov chain.

Proof of Theorem 1

Too complicated to explain here, but it has something to do with

The Narayana numbers

- Definition: $N(n, k)$ is the number of legal parenthesizations of length $2n$ containing k pairs of adjacent left parentheses.
- For example, $N(4, 2) = 6$:

$((()))()$ $()((()))$
 $(())()$ $((())())$
 $((()()))$ $((())())$

Calculating the expected number of free processors

- There are $\binom{n}{k} \binom{n-1}{k-1}$ states with exactly k free processors in the n -processor Markov chain
- A state with k free processors has 2^k outgoing edges and, since the chain is balanced, a relative steady-state probability proportional to 2^k
- So we can calculate the expected number of free processors as
$$\frac{\sum_{k=1}^n k 2^k \binom{n}{k} \binom{n-1}{k-1}}{\sum_{k=1}^n 2^k \binom{n}{k} \binom{n-1}{k-1}}$$

Theorem 2

As n goes to infinity, the expected number of free processors approaches $(2 - \sqrt{2})n \approx .58n$.

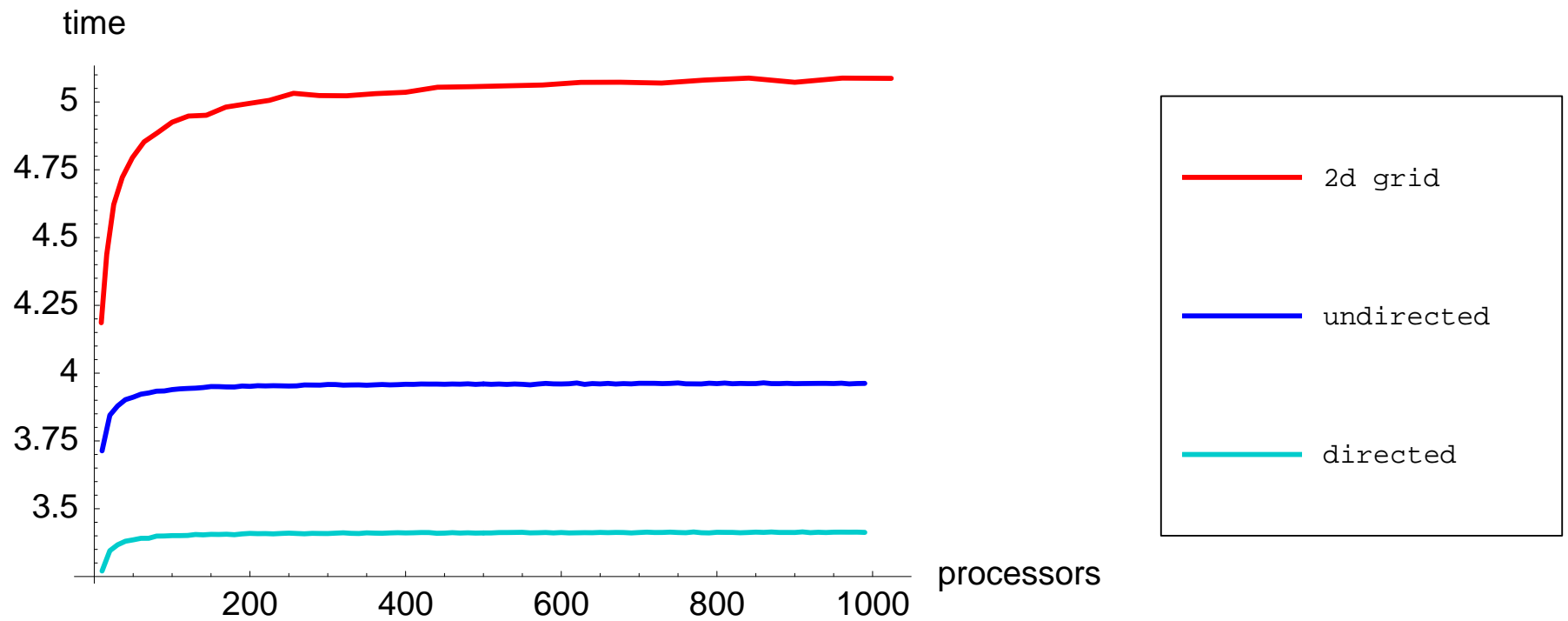
Corollary

- As n goes to infinity, each processor will be free for a fraction of the time that approaches $(2 - \sqrt{2})$.
- Since a free processor takes an expected time of 2 to complete a task, the total time between levels, including synchronization time, will approach $\frac{2}{2 - \sqrt{2}} = 2 + \sqrt{2} \approx 3.42$ time units.

Comparing different types of synchronization

synchronization type	expected time between levels
none	2
directed cycle	≤ 3.42
global	$\log_2(n)$

Simulation results for the geometric distribution



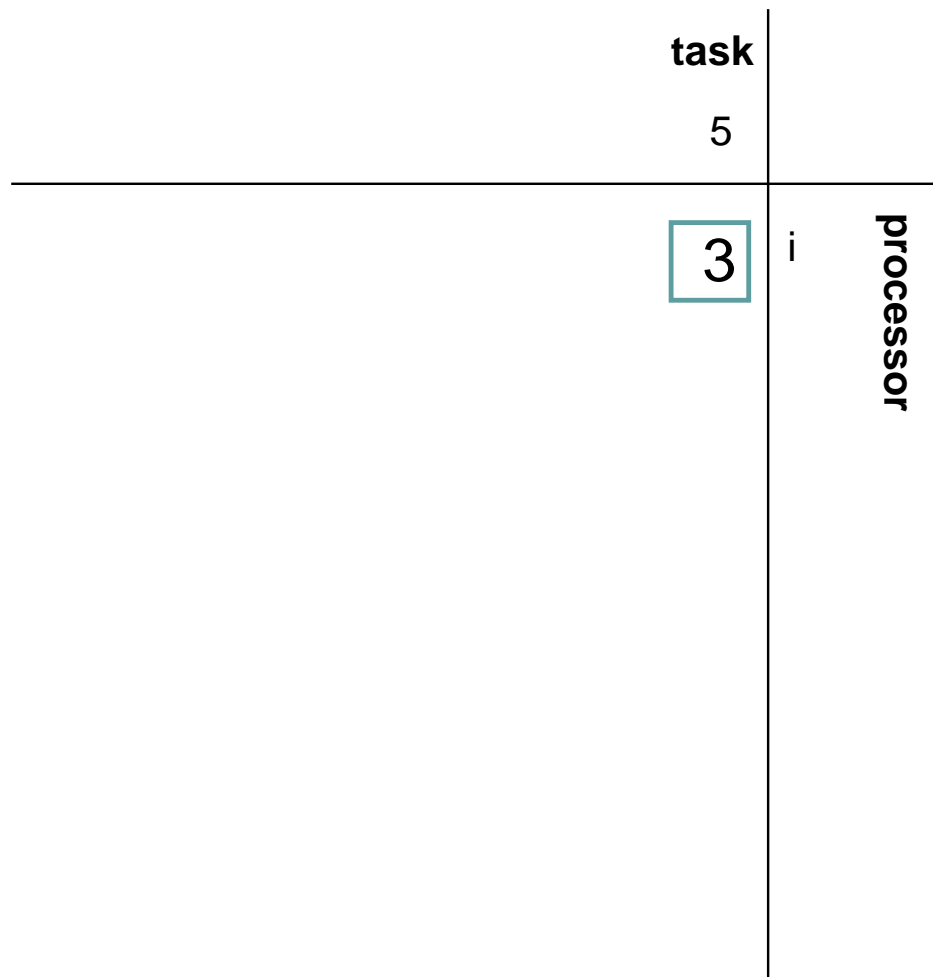
The exponential distribution

- The exponential distribution, $F(x) = 1 - e^{-\lambda x}$, is the continuous analogue to the geometric distribution, also memoryless
 - However, a directed cycle with exponential tasks X_i with $E(X_i) = 2$ behaves differently than one with geometric tasks Y_i with $E(Y_i) = 2$
 - Recasting the problem as a cyclic queue problem, results from queueing theory show that $\mathcal{T}_{G,f}^* = 4$, not 3.42
-

Heavy-tailed distributions

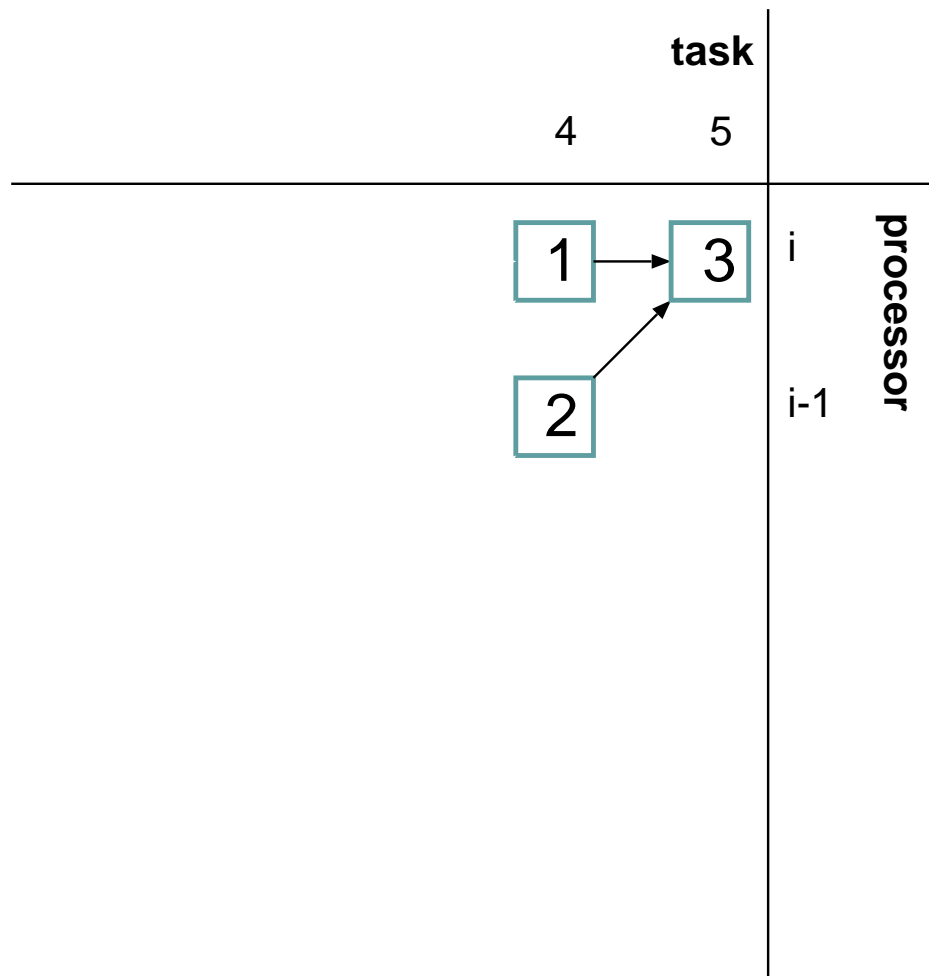
- Heavy-tailed distributions, like the Pareto (power law) distribution, often arise in analyses of network traffic
- Let the task time distribution $F(x) = 1 - x^{-\lambda}$
- We can show that there are some graphs for which $\mathcal{T}_{G,F}^*$ has no constant upper bound as the number of processors goes to infinity

The dependency graph



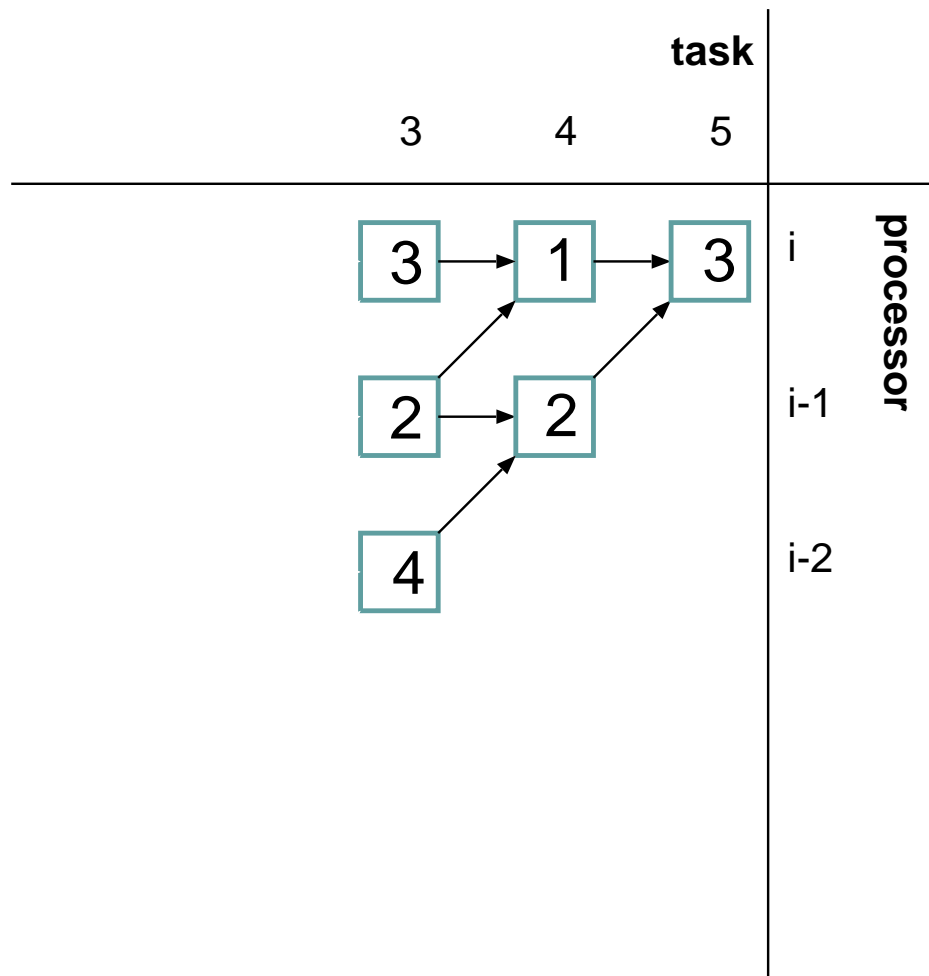
- Processor i depends immediately on itself and $i - 1$

The dependency graph



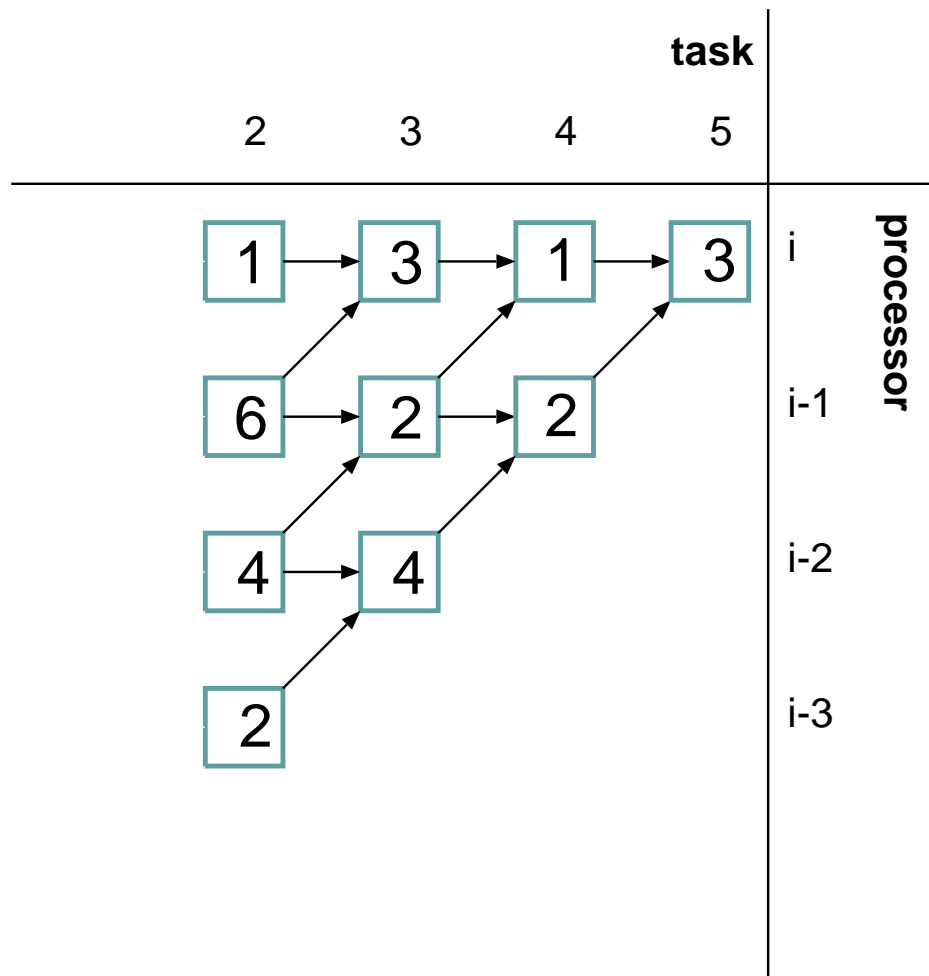
- Processor i depends immediately on itself and $i - 1$

The dependency graph



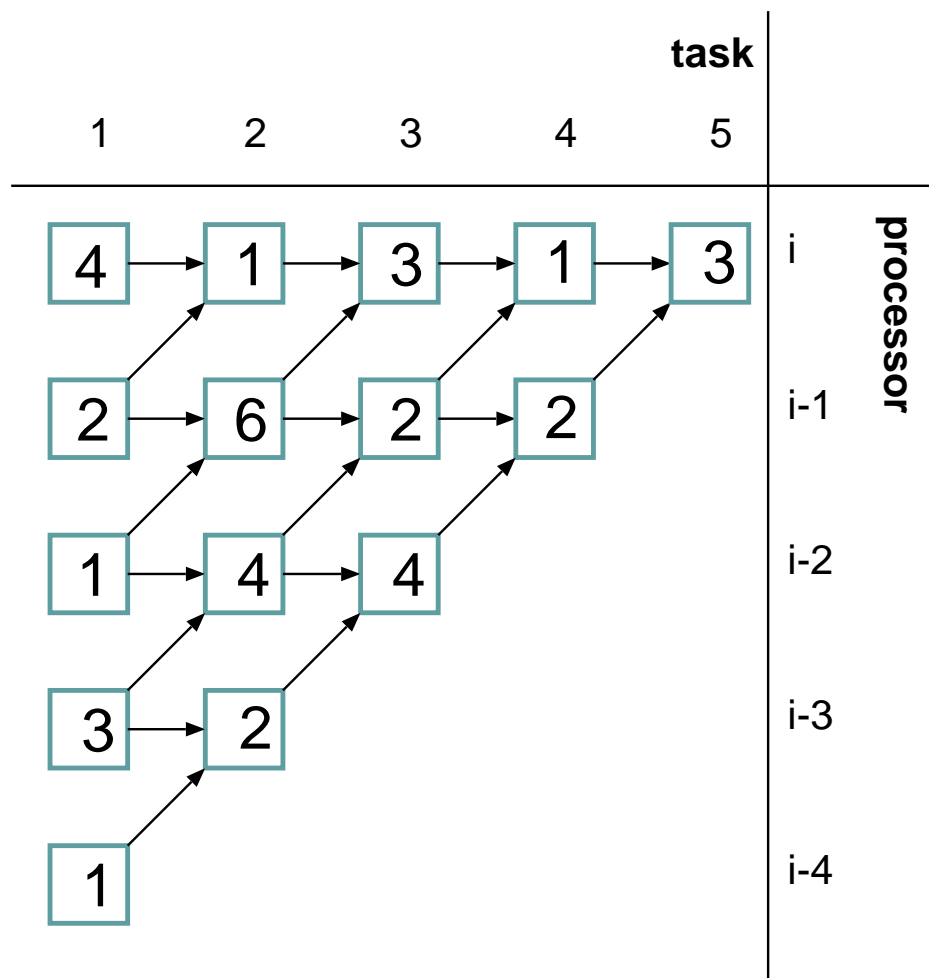
- Processor i depends immediately on itself and $i - 1$

The dependency graph



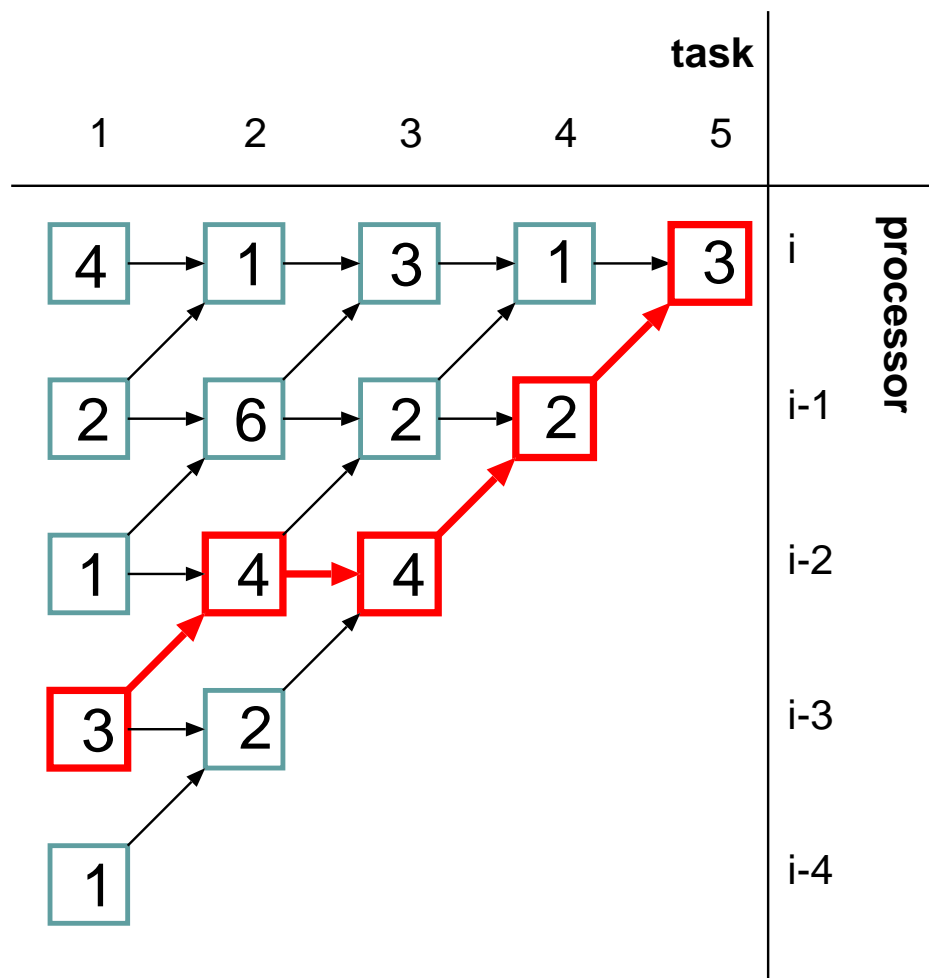
- Processor i depends immediately on itself and $i - 1$

The dependency graph



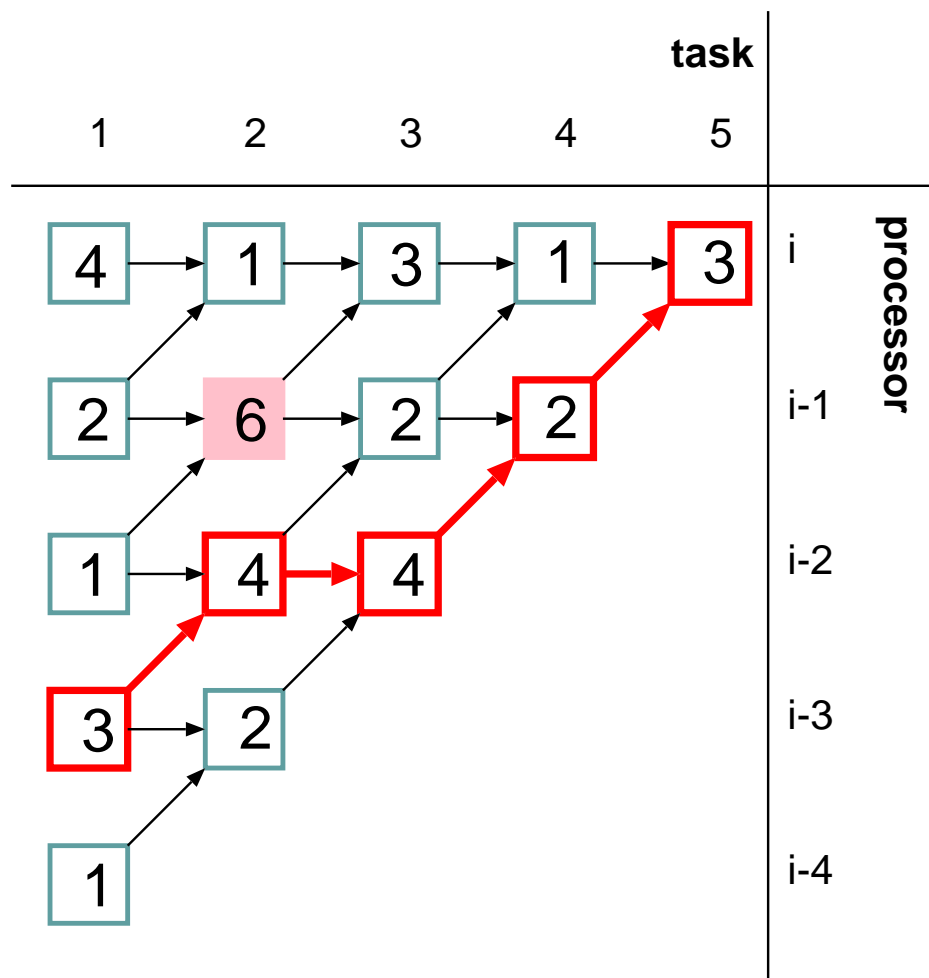
- Processor i depends immediately on itself and $i - 1$

The dependency graph



- Processor i depends immediately on itself and $i - 1$
- The time when task 5 is finished is the maximum sum along paths in the graph

The dependency graph



- Processor i depends immediately on itself and $i - 1$
- The time when task 5 is finished is the maximum sum along paths in the graph
- This is bounded below by the single worst task time

A lower bound

- Let $t(j, k)$ be the time it takes for the j th processor to do the k th task
 - For a processor i , $\mathcal{T}_{G,F}^*$ is bounded below by $\lim_{r \rightarrow \infty} \frac{1}{r} E(\max(t(j, k)))$ for all $t(j, k)$ that could affect the time at which the i th processor finishes the r th task
 - For the directed cycle, there are $1 + 2 + 3 + \dots + r = \Theta(r^2)$ such task times
 - In general, for a d -dimensional grid, there are $\Theta(r^{d+1})$ such task times
-

A lower bound

- The expected maximum of m iid Pareto random variables with parameter λ grows as $m^{\frac{1}{\lambda}}$
- This is fast!
- If $n > r$, then the lower bound for $\mathcal{T}_{G,F}^*$ is growing as $\frac{1}{r} (r^{d+1})^{\frac{1}{\lambda}}$
- So if we pick $d > \lambda - 1$, then this is a strictly increasing function of r , showing that there is no constant upper bound on $\mathcal{T}_{G,F}^*$
- Example: G is a directed cycle, $\lambda = 1.5$

Conclusion

- Local synchronization is much faster than global synchronization on a directed cycle with geometrically distributed tasks
 - Geometric distributions are among the most heavy-tailed of distributions with a moment generating function, so that implies that similar results apply to other distributions with moment generating functions
 - However, there is no similar upper bound for certain task graphs with power-law distributions
-