Principles of Parallel Algorithm Design: Concurrency and Decomposition

Milind Chabbi

Nikhil Hegde

Department of Computer Science
IIT Dharwad

Parallel Algorithm

Recipe to solve a problem using multiple processors

Typical steps for constructing a parallel algorithm

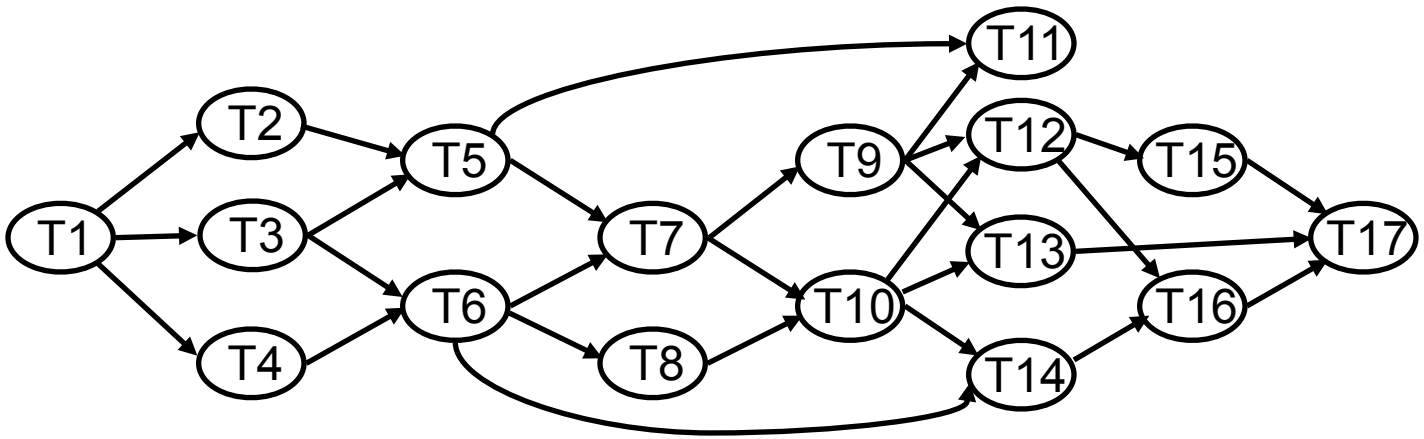
- identify what pieces of work can be performed concurrently
- partition and map work onto independent processors
- distribute a program's input, output, and intermediate data
- coordinate accesses to shared data: avoid conflicts
- ensure proper order of work using synchronization

Why "typical"? Some of the steps may be omitted.

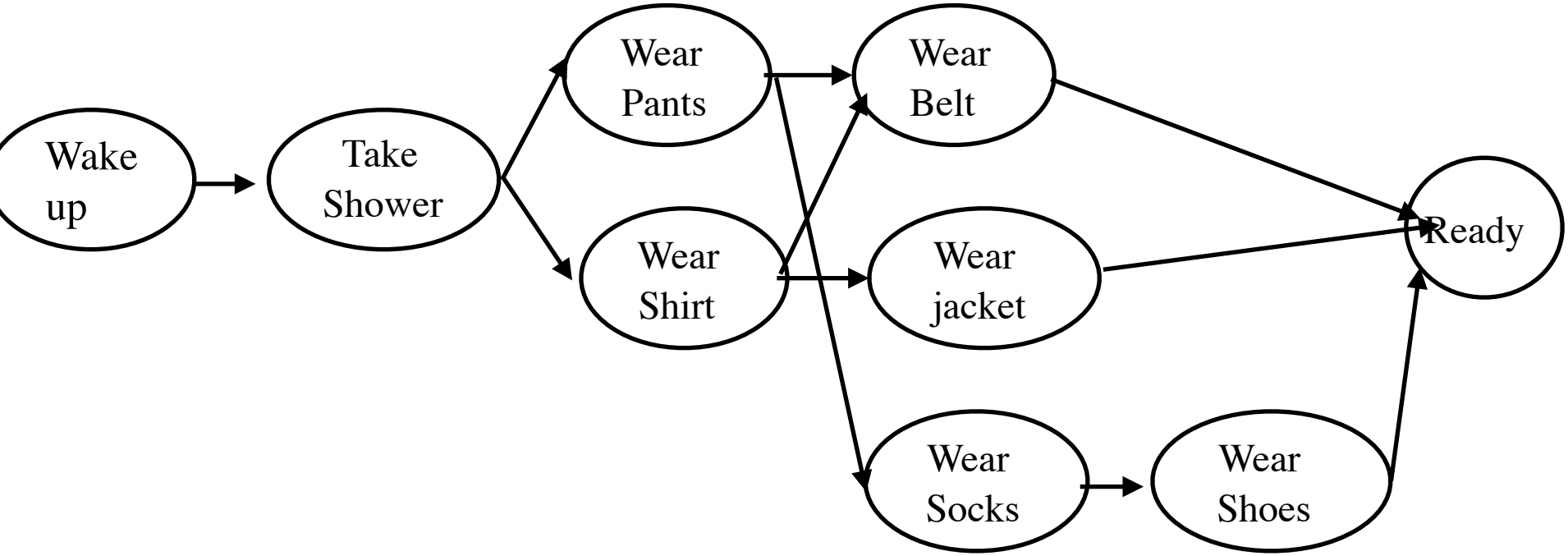
- if data is in shared memory, distributing it may be unnecessary
- if using message passing, there may not be shared data
- the mapping of work to processors can be done statically by the programmer or dynamically by the runtime

Decomposing Work for Parallel Execution

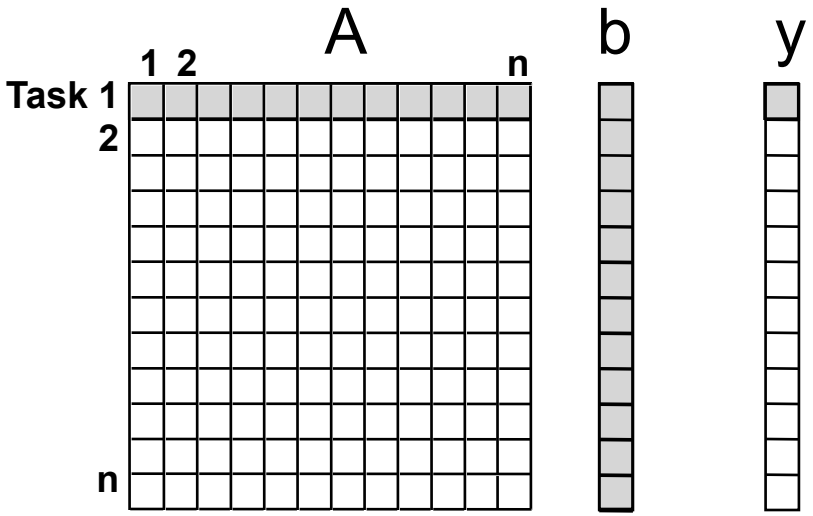
- Divide work into tasks that can be executed concurrently
- Many different decompositions possible for any computation
- Tasks may be same, different, or even indeterminate sizes
- Tasks may be independent or have non-trivial order
- Conceptualize tasks and ordering as task dependency DAG
  - —node = task
  - —edge = control dependence



Task Dependency Graph



Example: Dense Matrix-Vector Product



- Computing each element of output vector y is independent
- Easy to decompose dense matrix-vector product into tasks
  - —one per element in y
- Observations
  - —task size is uniform
  - —no control dependences between tasks
  - —tasks share b

Example: Database Query Processing

Consider the execution of the query:

MODEL = "CIVIC" AND YEAR = 2001 AND

(COLOR = "GREEN" OR COLOR = "WHITE")

on the following database:

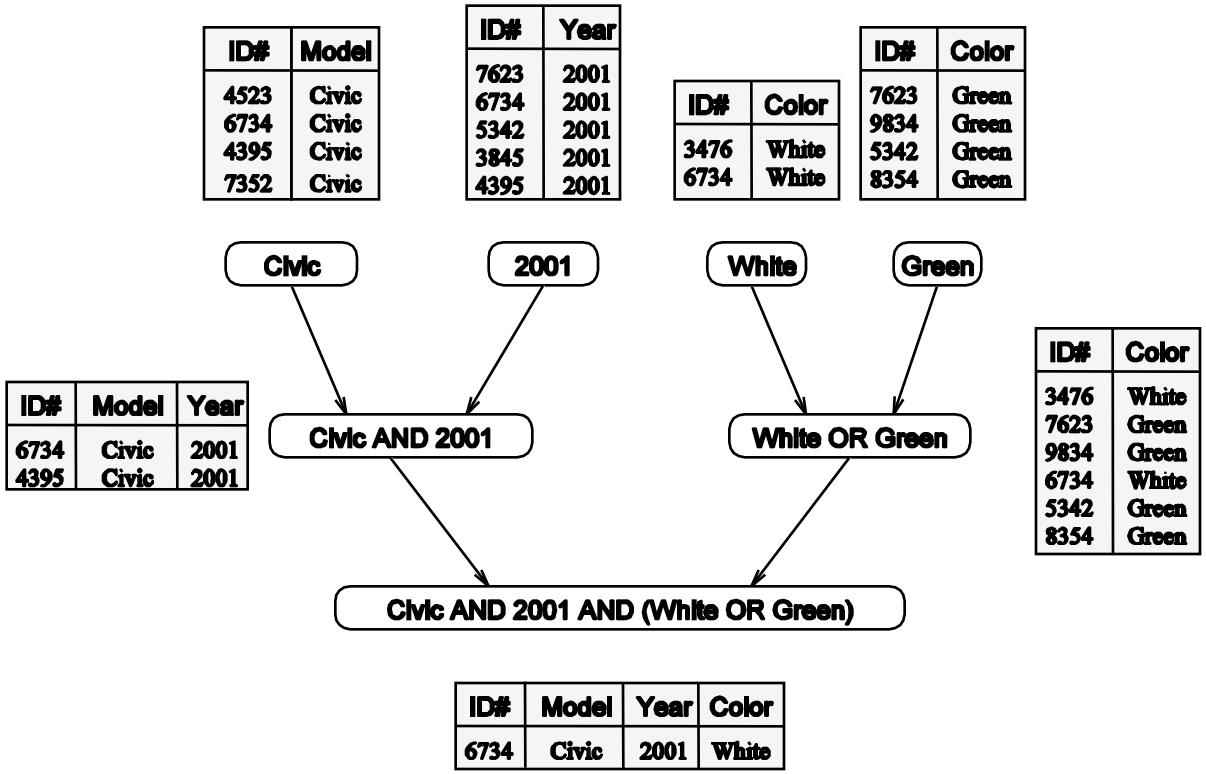
ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Example: Database Query Processing

- Task: compute set of elements that satisfy a predicate
  - —task result = table of entries that satisfy the predicate
- Edge: output of one task serves as input to the next

MODEL = "CIVIC" AND YEAR = 2001 AND

(COLOR = "GREEN" OR COLOR = "WHITE")

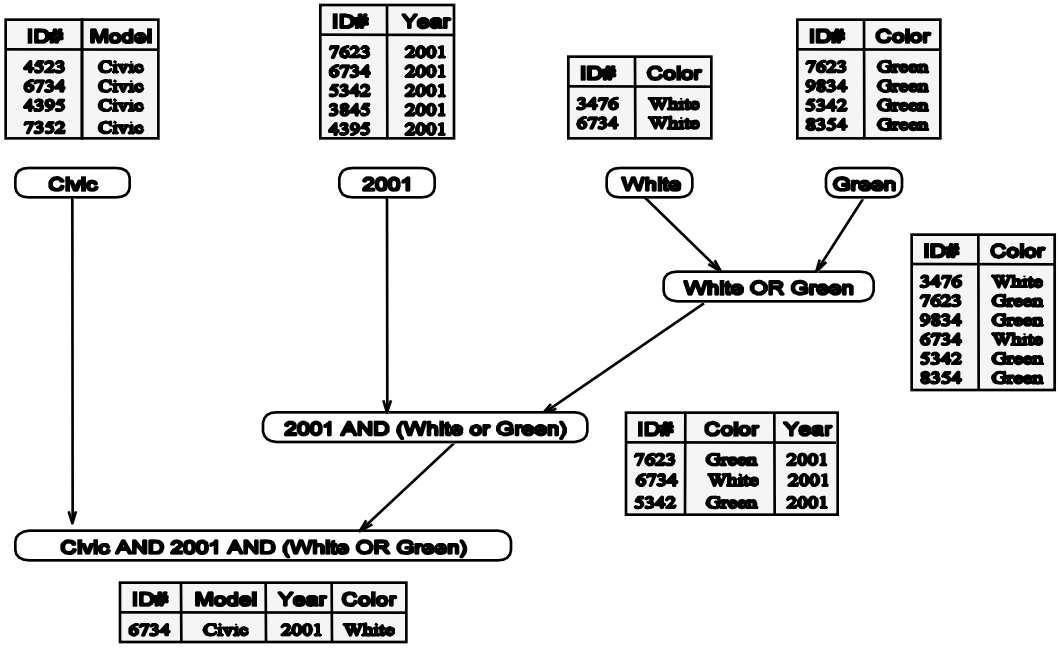


Example: Database Query Processing

- **Alternate task decomposition for query**

MODEL = "CIVIC" AND YEAR = 2001 AND

(COLOR = "GREEN" OR COLOR = "WHITE")

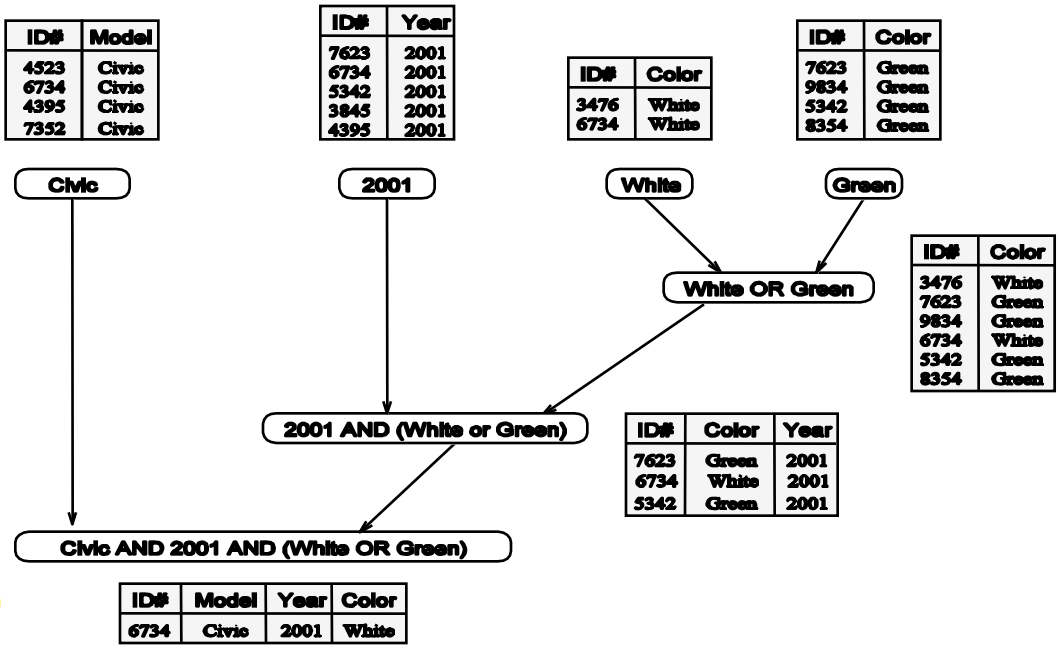


Example: Database Query Processing

- Alternate task decomposition for query

MODEL = "CIVIC" AND YEAR = 2001 AND

(COLOR = "GREEN" OR COLOR = "WHITE")



Note: Different decompositions may yield different parallelism and different amounts of work

Granularity of Task Decompositions

Granularity of Task Decompositions

- **Granularity = task size**
  - **—depends on the number of tasks**

Granularity of Task Decompositions

- **Granularity = task size**
  - **—depends on the number of tasks**
- **Fine-grain = large number of tasks**

Granularity of Task Decompositions

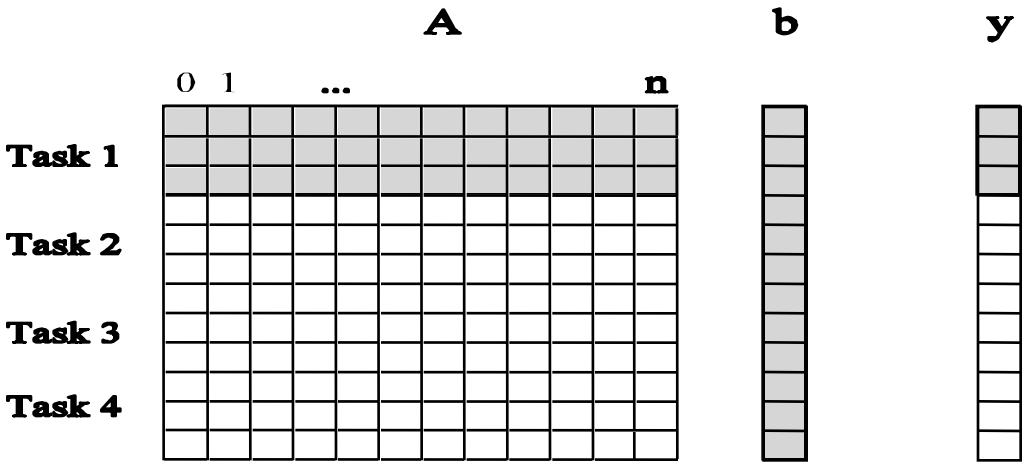
- **Granularity = task size**
  - **—depends on the number of tasks**
- **Fine-grain = large number of tasks**
- **Coarse-grain = small number of tasks**

Granularity of Task Decompositions

- Granularity = task size
  - —depends on the number of tasks
- Fine-grain = large number of tasks
- Coarse-grain = small number of tasks
- Granularity examples for dense matrix-vector multiply
  - —fine-grain: each task represents an individual element in y

Granularity of Task Decompositions

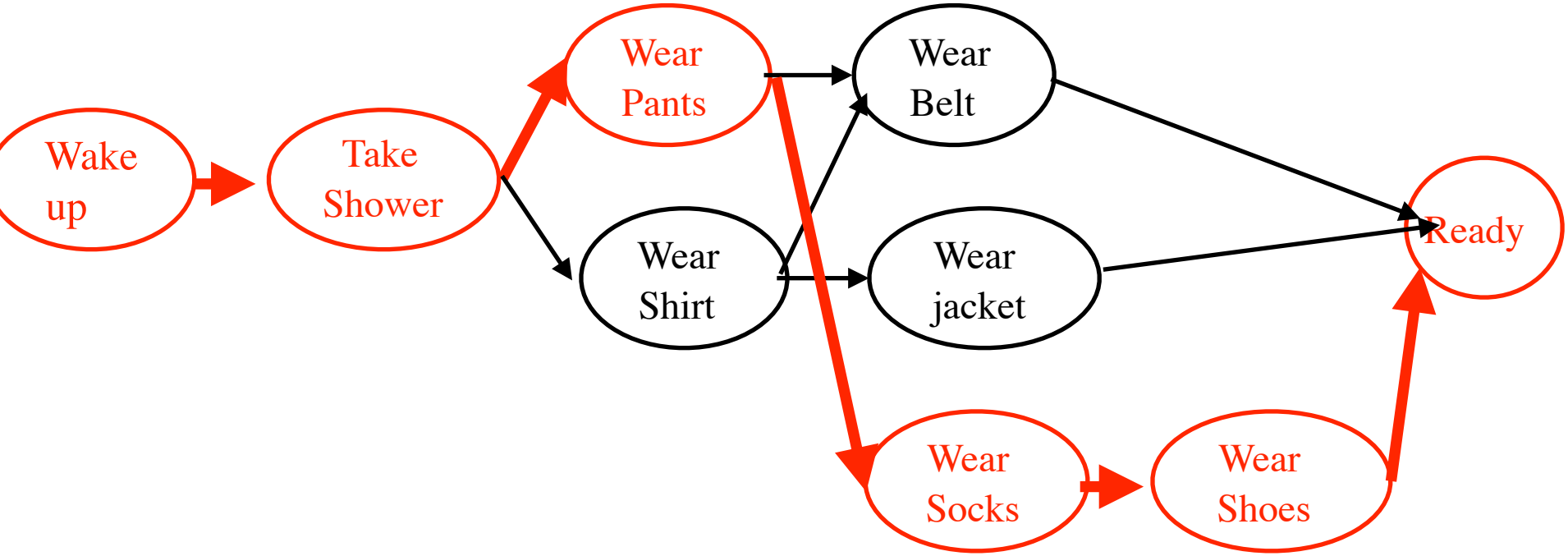
- Granularity = task size
  - —depends on the number of tasks
- Fine-grain = large number of tasks
- Coarse-grain = small number of tasks
- Granularity examples for dense matrix-vector multiply
  - —fine-grain: each task represents an individual element in y
  - —coarser-grain: each task computes 3 elements in y



Critical Path

- Edge in task dependency graph represents task serialization
- Critical path = longest weighted path through graph
- Critical path length = lower bound on parallel execution time

Task Dependency Graph



Degree of Concurrency

- Definition: number of tasks that can execute in parallel
- May change during program execution
- Metrics
  - —maximum degree of concurrency
    - largest # concurrent tasks at any point in the execution
  - —average parallelism
    - total amount of work divided by length of the critical path
- Degree of concurrency vs. task granularity
  - —inverse relationship

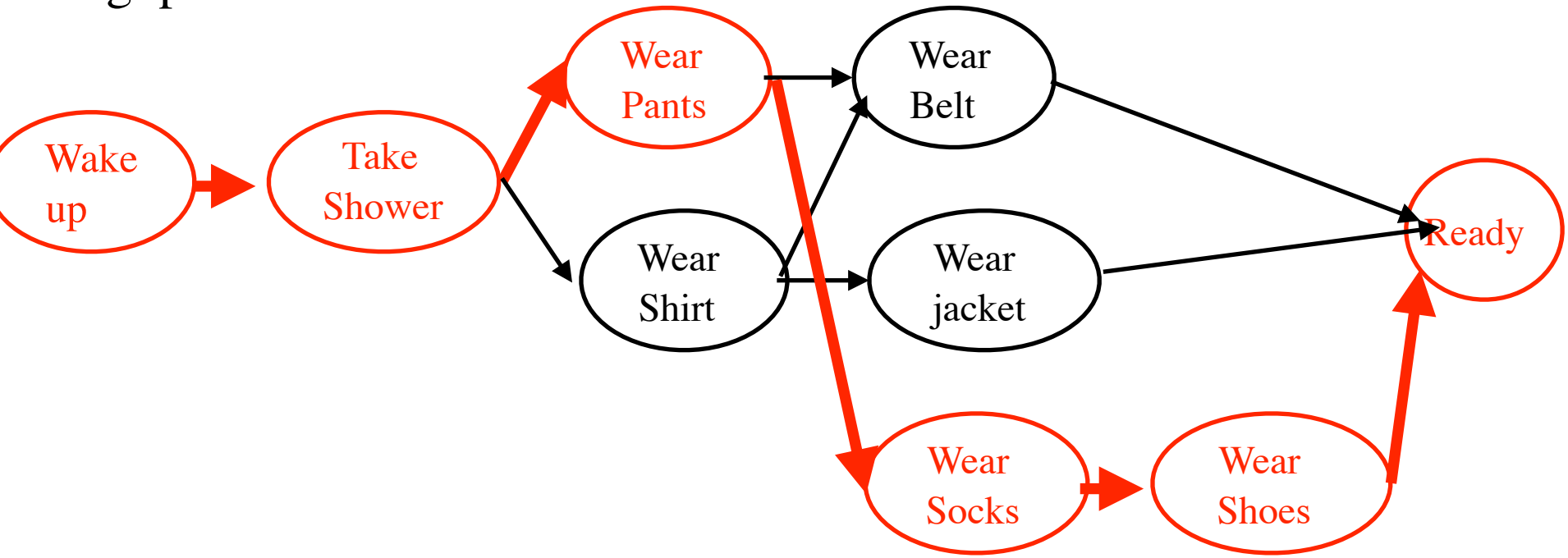
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



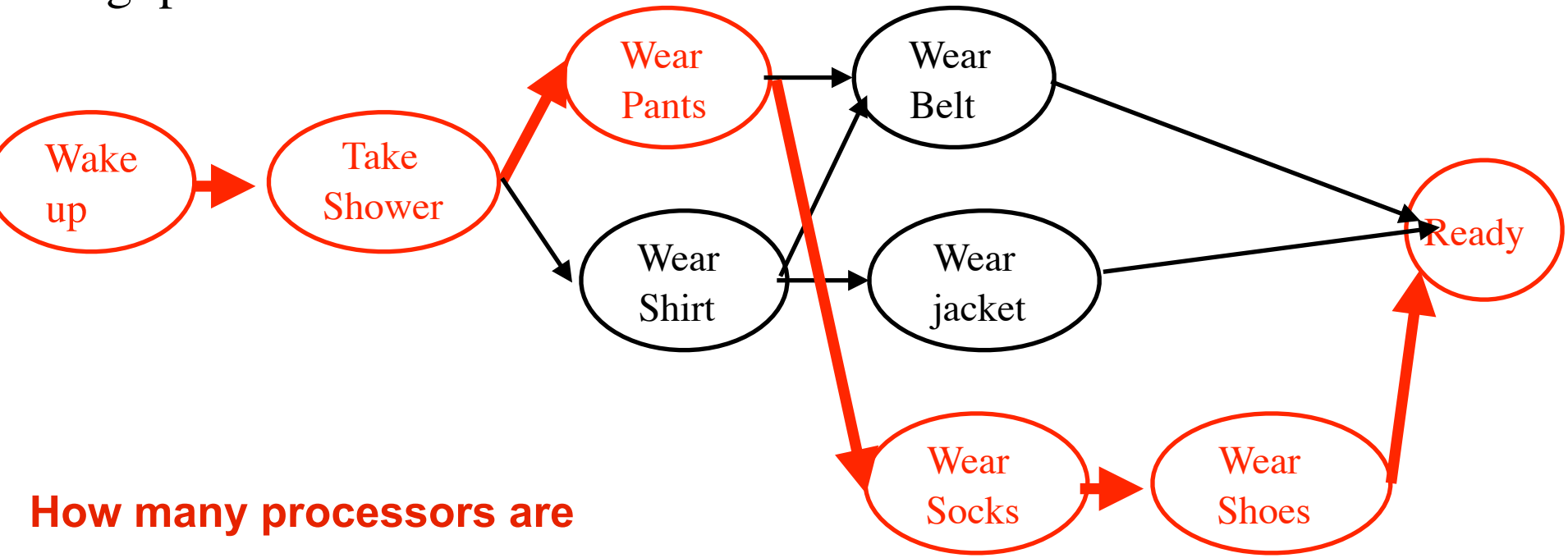
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 1.5 CPUs

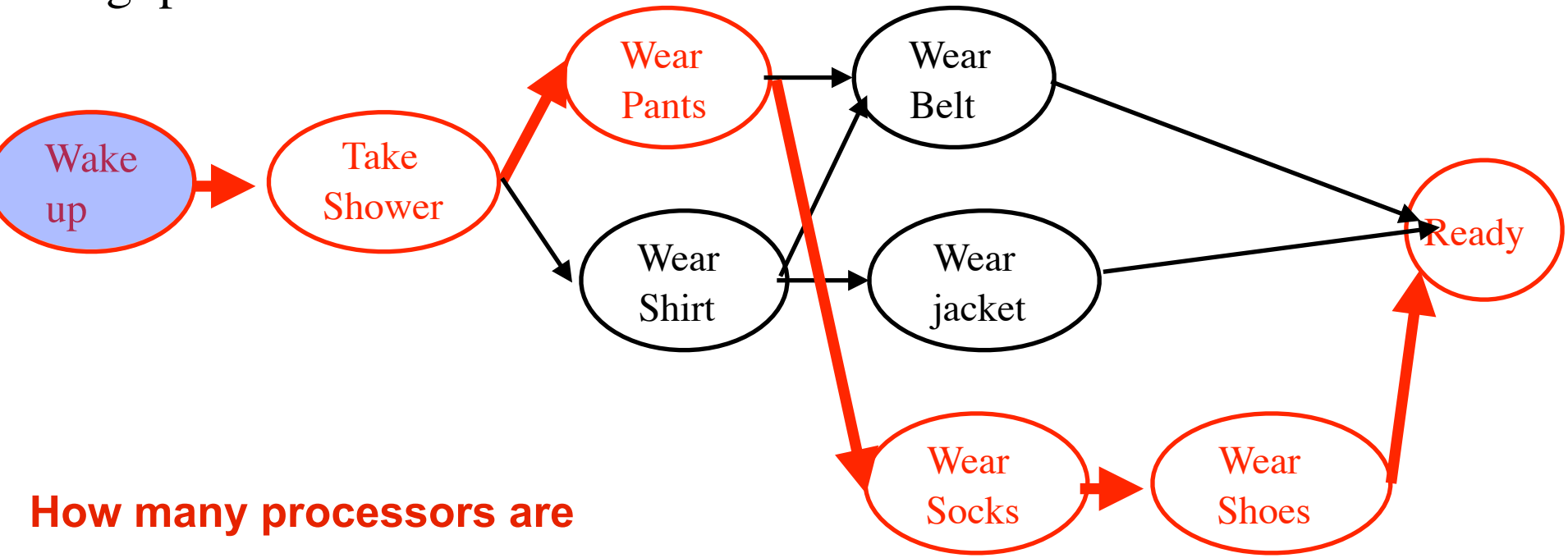
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 1.5 CPUs

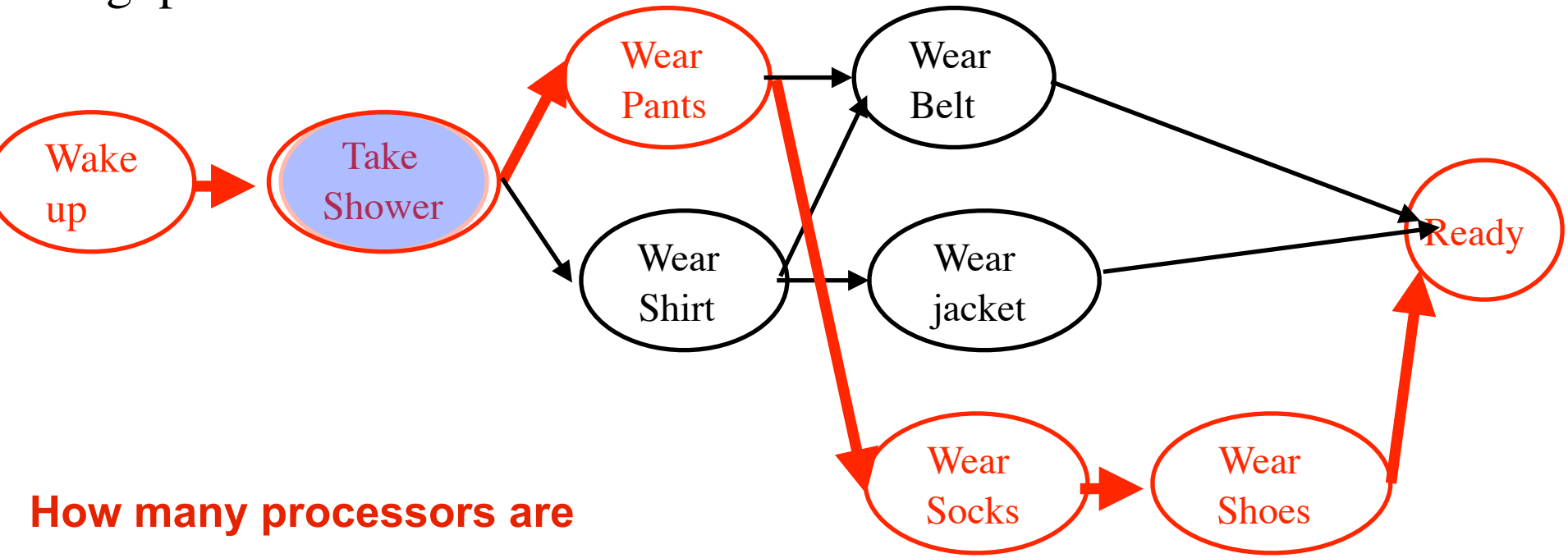
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 1.5 CPUs

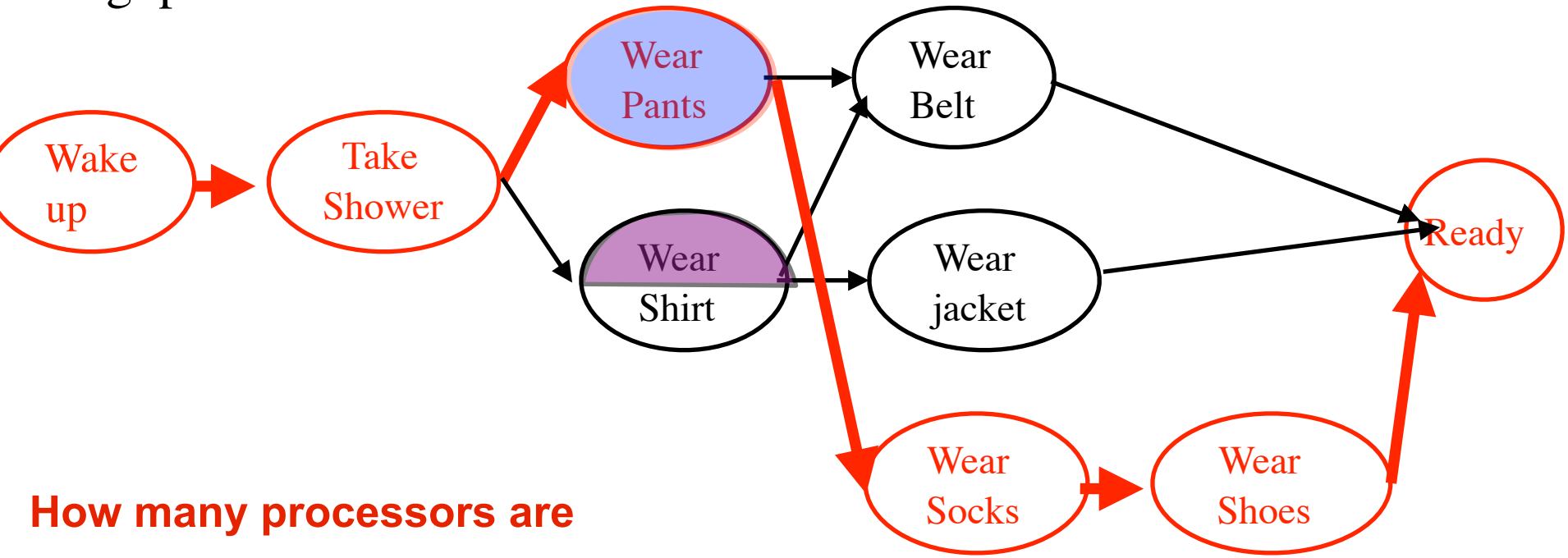
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 1.5 CPUs

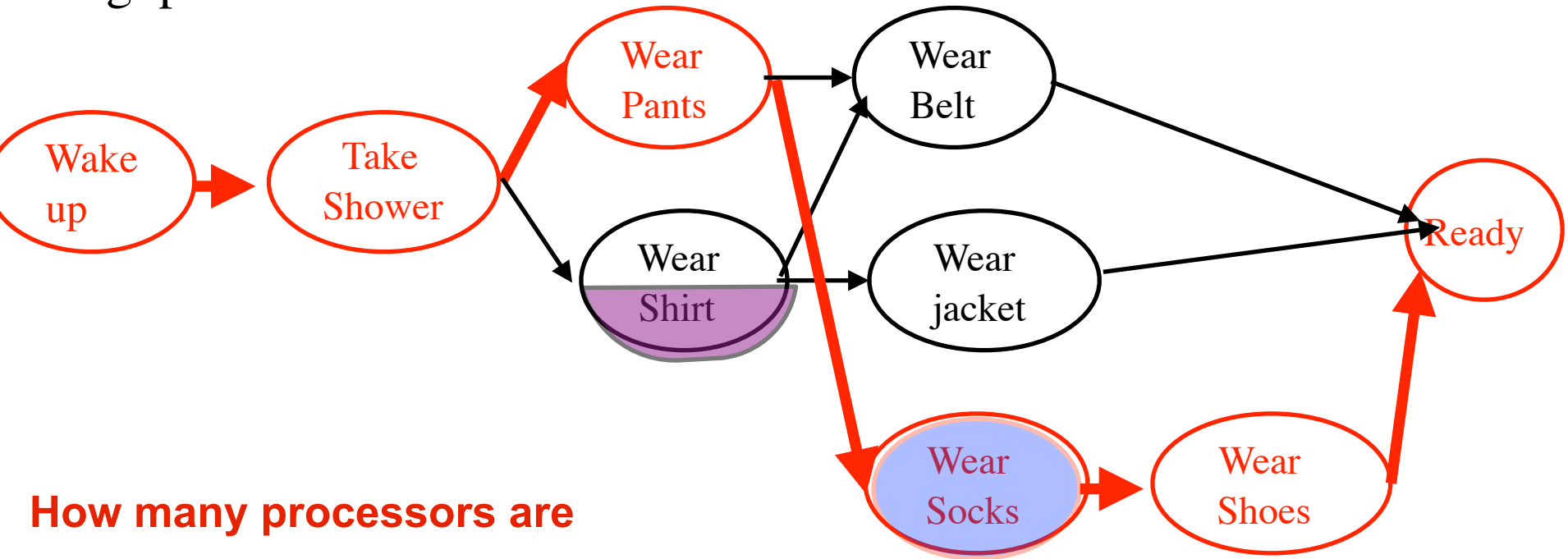
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 1.5 CPUs

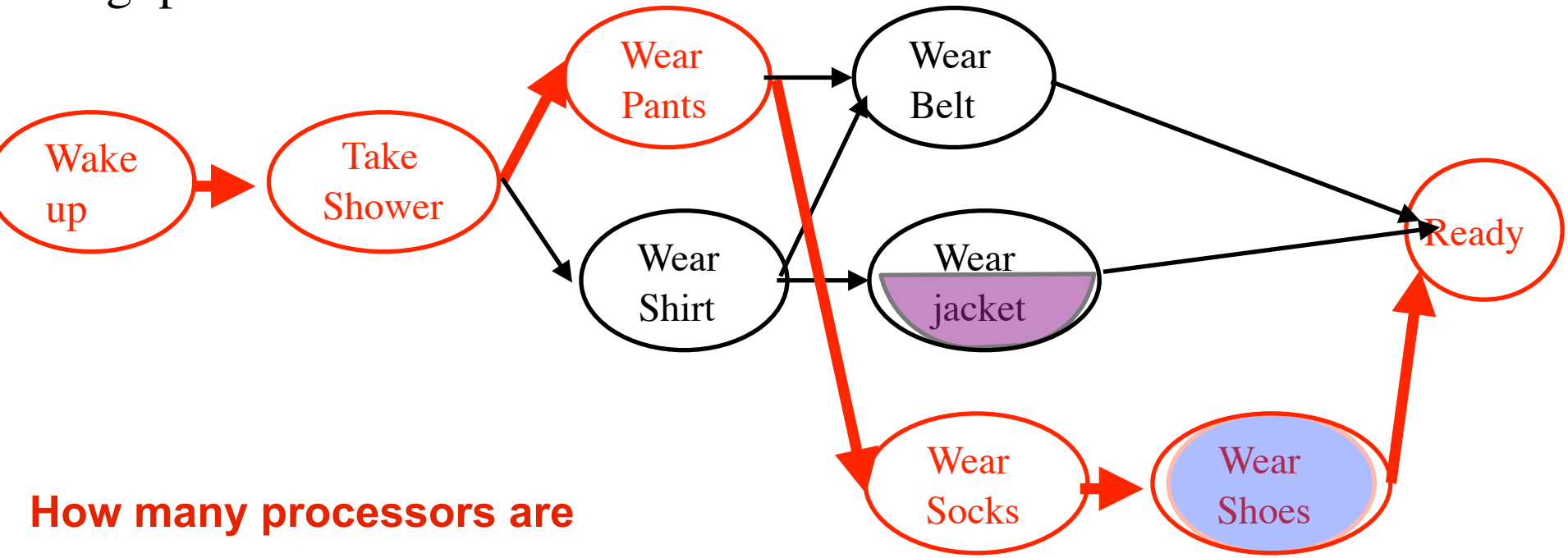
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 1.5 CPUs

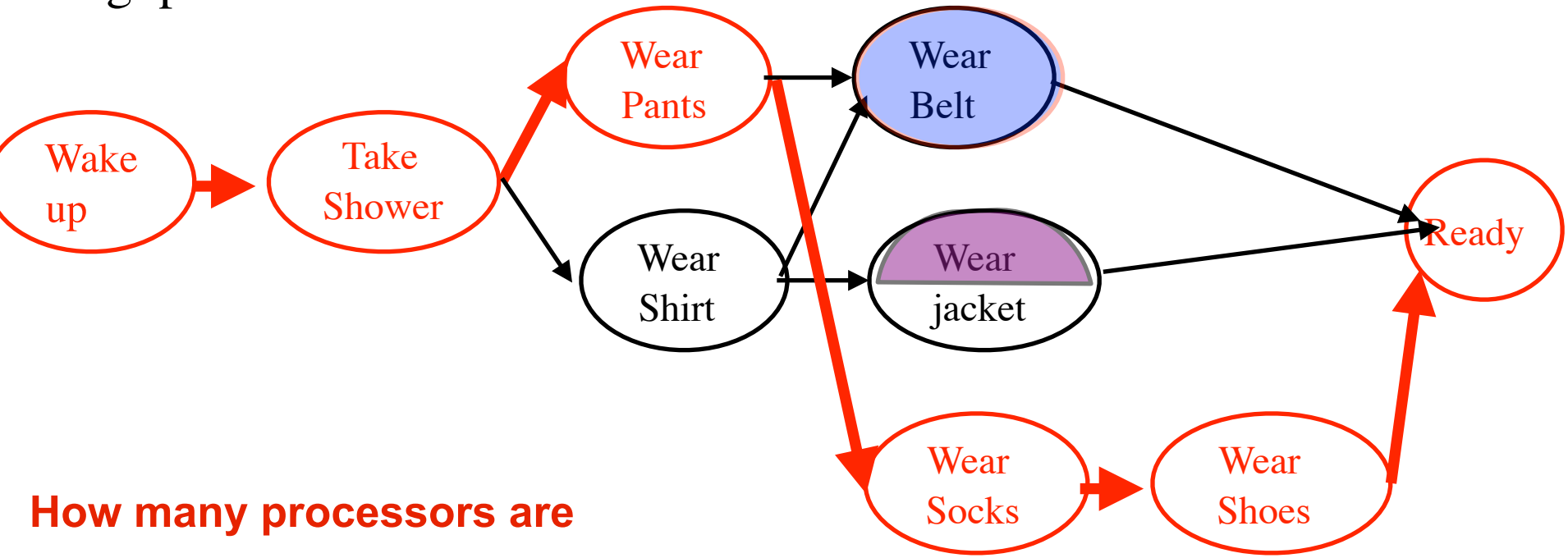
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 1.5 CPUs

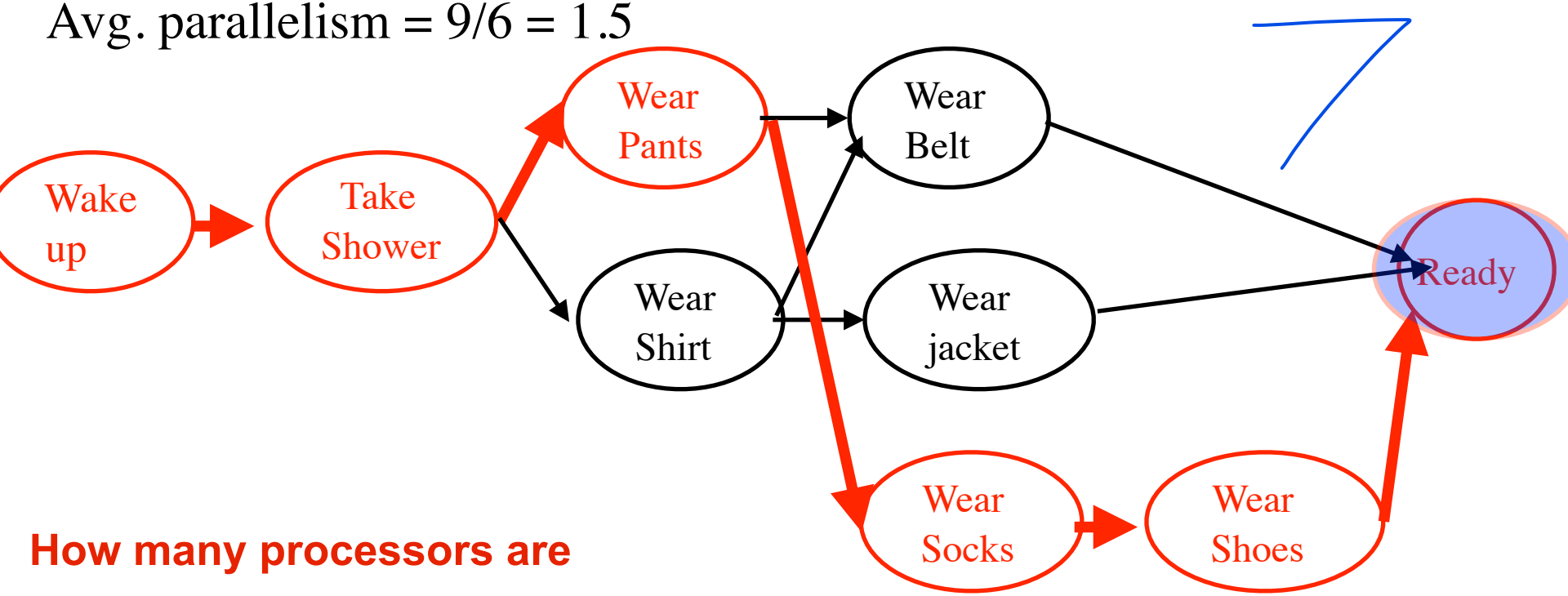
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 1.5 CPUs

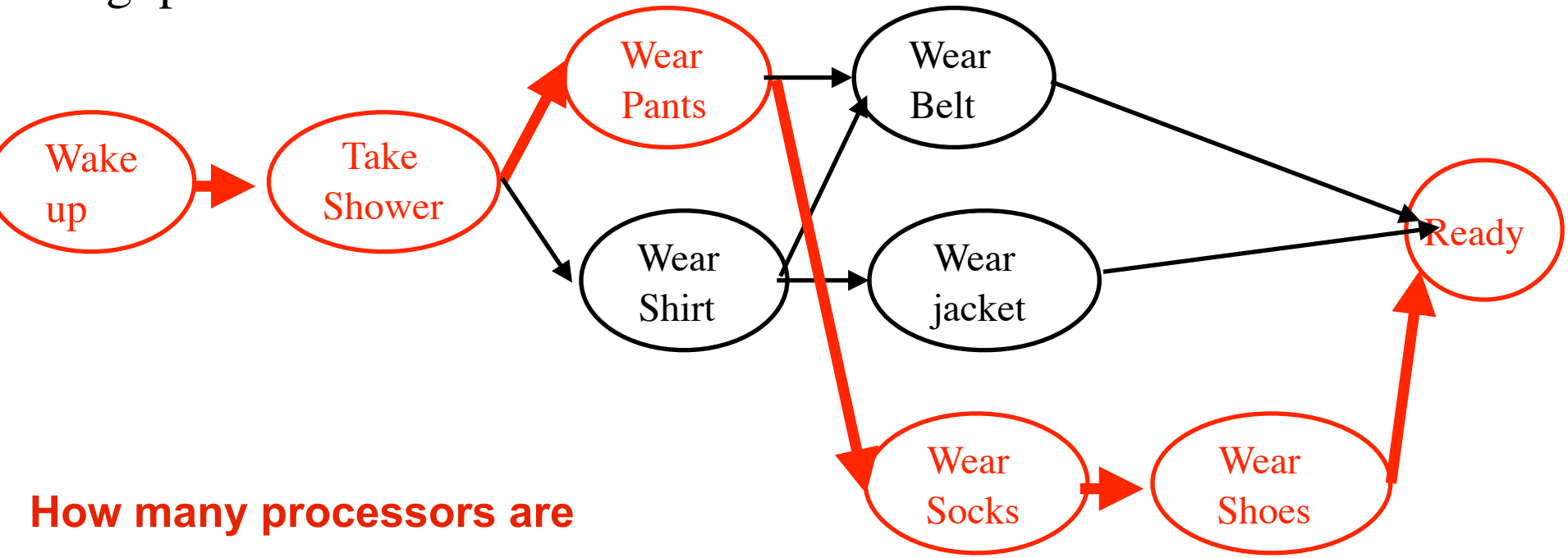
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 2 CPUs

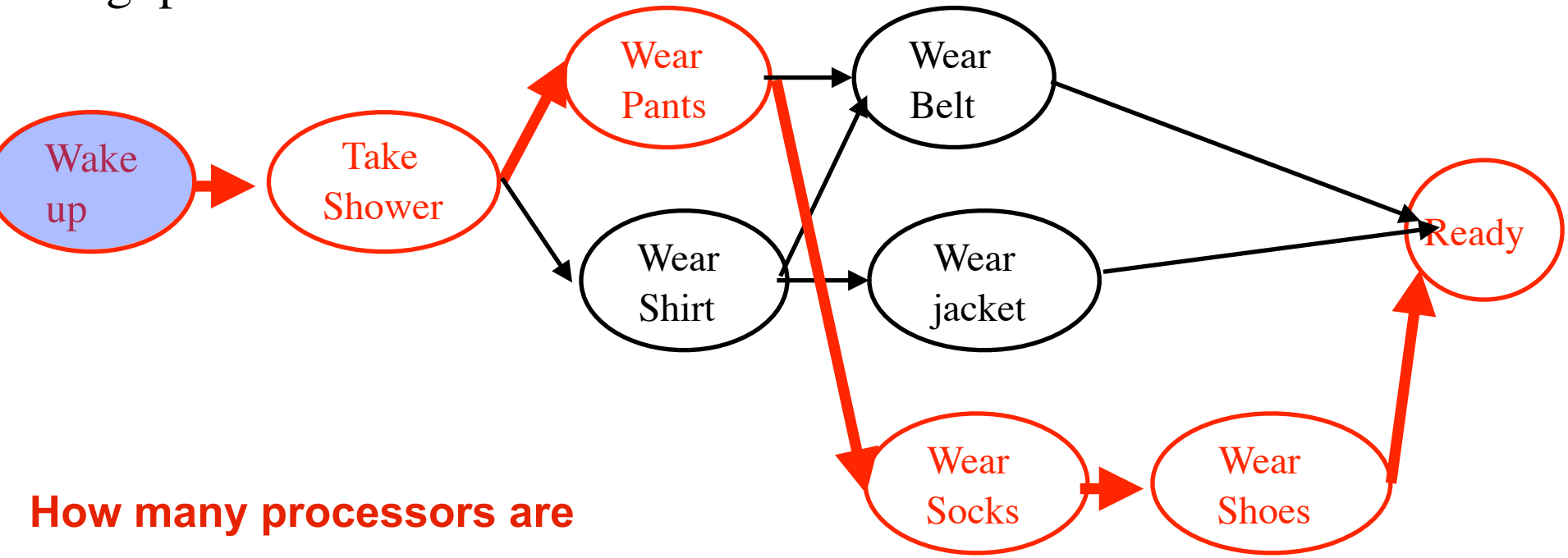
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 2 CPUs

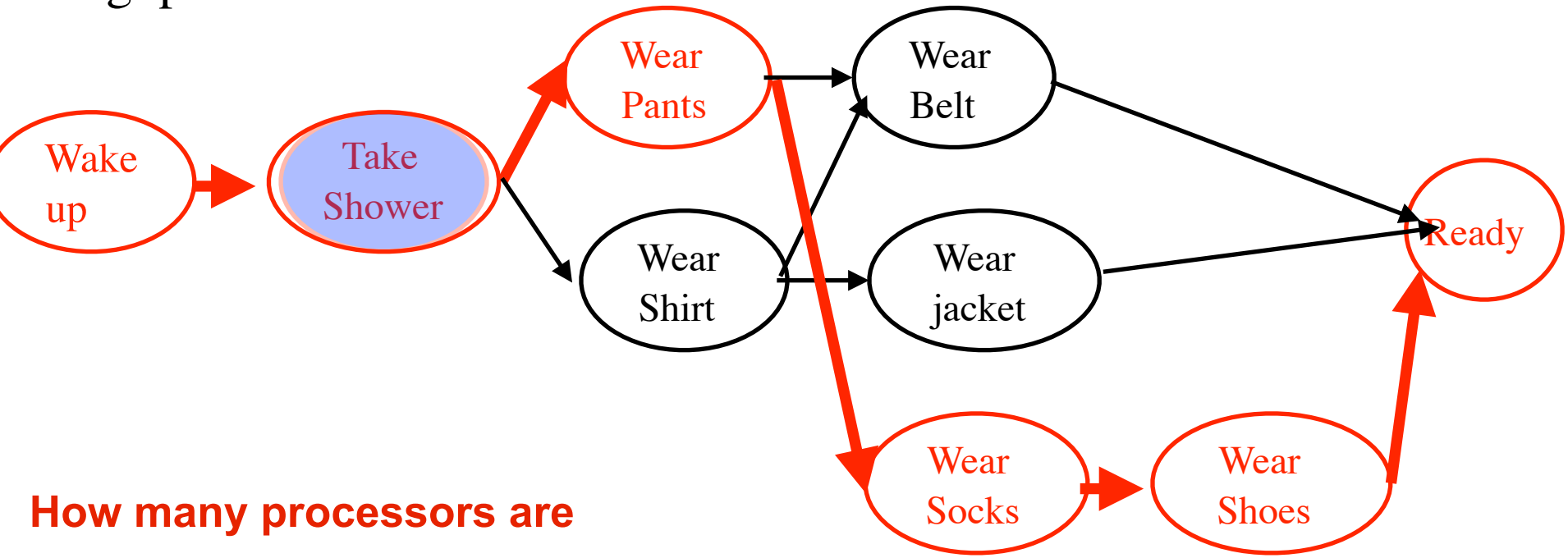
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 2 CPUs

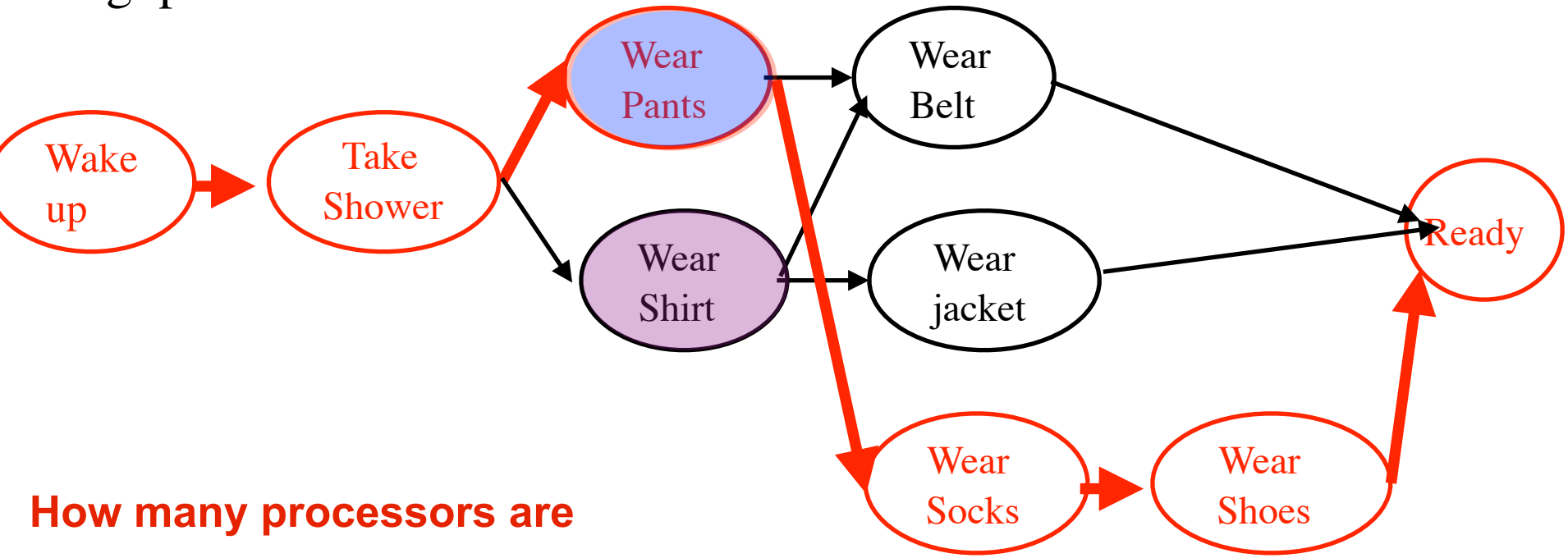
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 2 CPUs

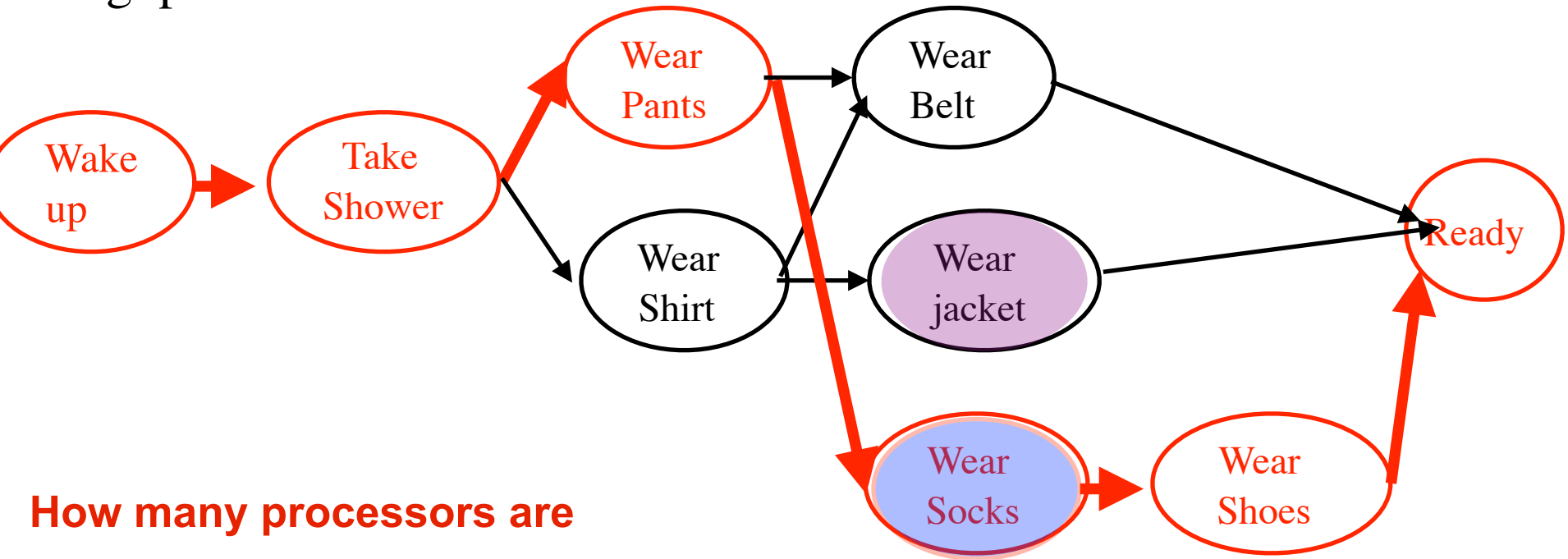
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 2 CPUs

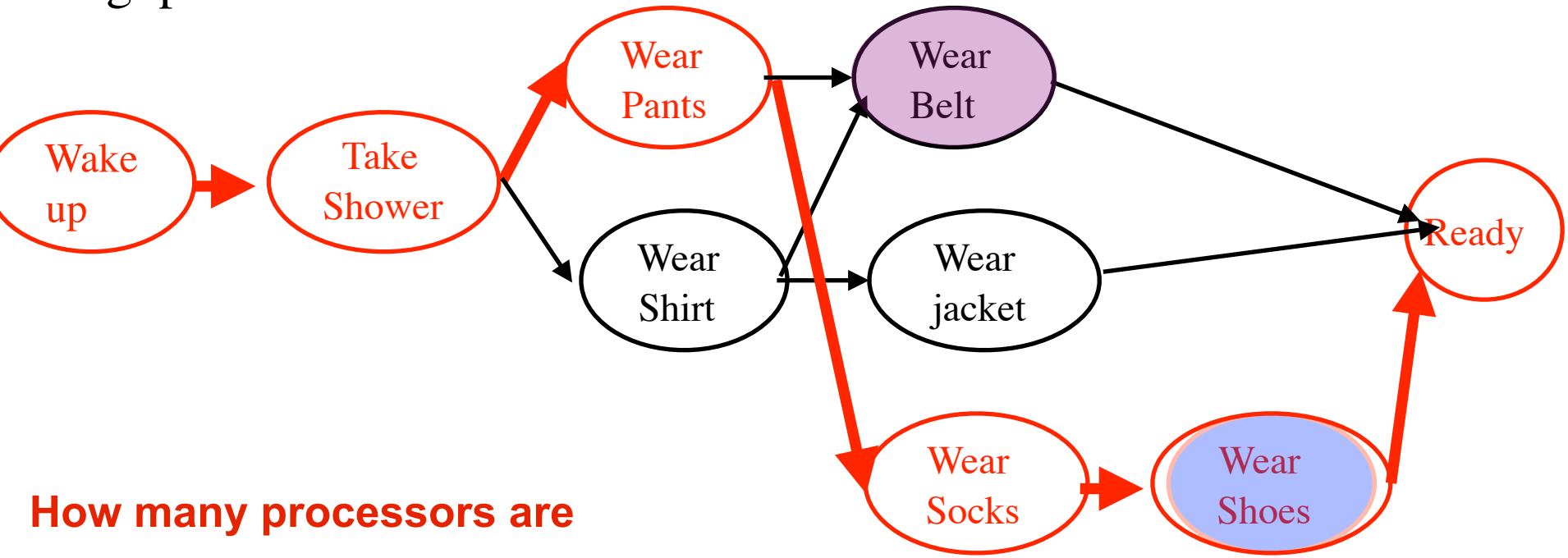
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 2 CPUs

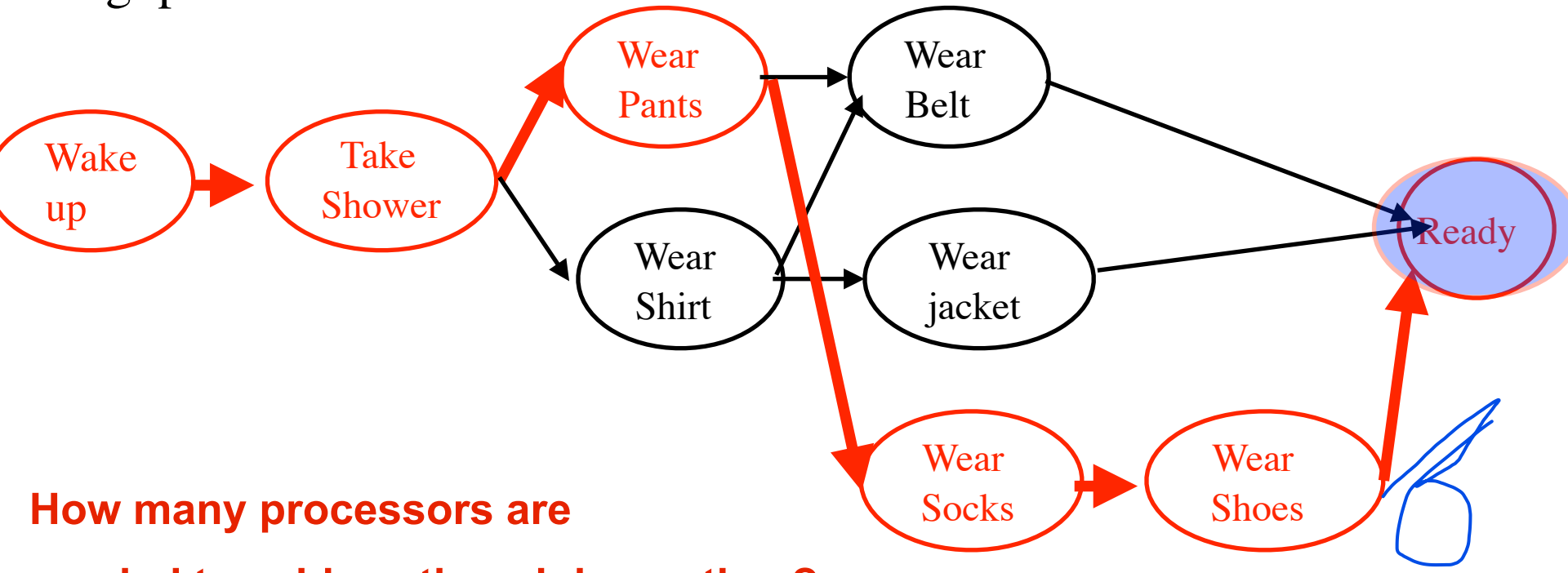
Metrics

Degree of concurrency

1	1	2	3	1	1

Max

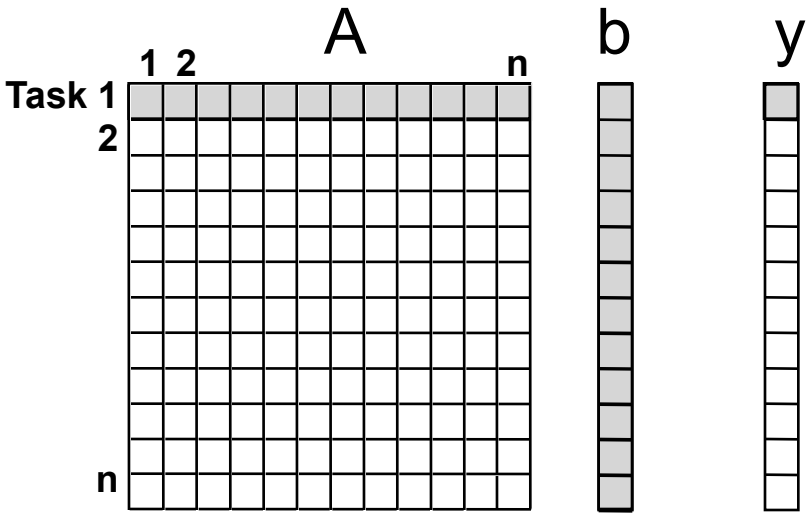
Work = 9 (assume unit work / node)
Critical Path Length = 6
Avg. parallelism = 9/6 = 1.5



How many processors are needed to achieve the minimum time?

Try with 2 CPUs

Example: Dense Matrix-Vector

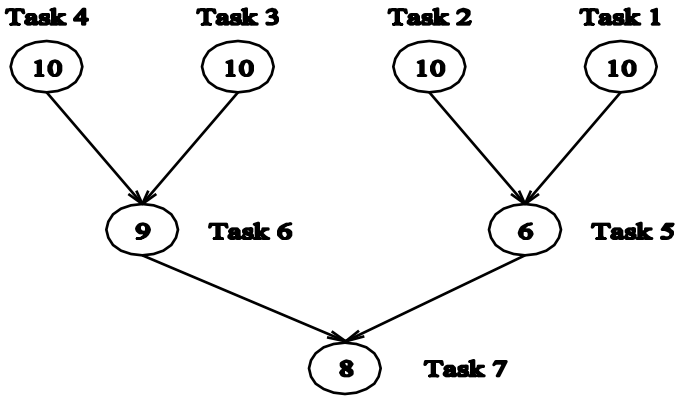


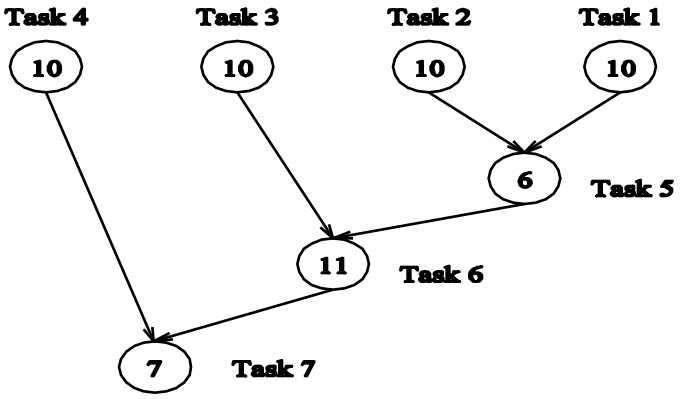
- Computing each element of output vector y is independent
- Easy to decompose dense matrix-vector product into tasks
  - —one per element in y
- Observations
  - —task size is uniform
  - —no control dependences between tasks
  - —tasks share b

Question: Is *n* the maximum number of tasks possible?

Critical Path Length

Examples: database query task dependency graphs





Note: number in each vertex represents the cost of its task

Questions:

What tasks are on the critical path for each dependency graph?

What is the shortest parallel execution time for each decomposition?

What is the maximum degree of concurrency?

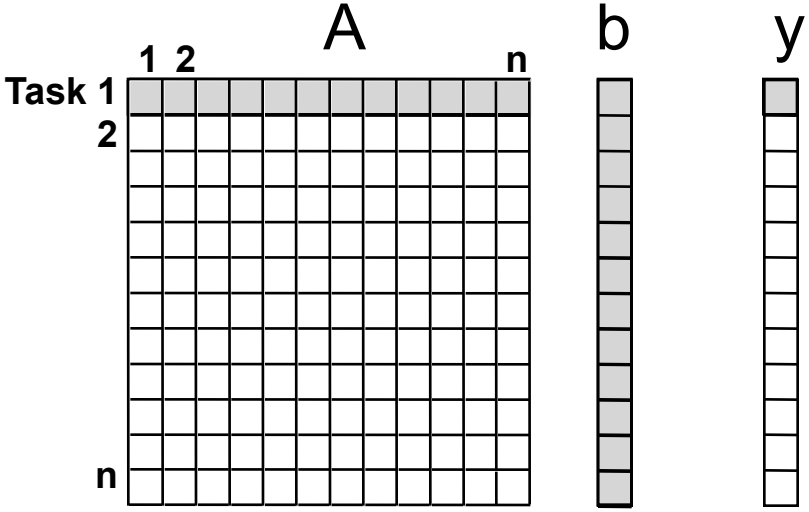
What is the average parallelism?

How many processors are needed to achieve the minimum time?

For simplicity: assume each node takes unit CPU time.

Critical Path Length

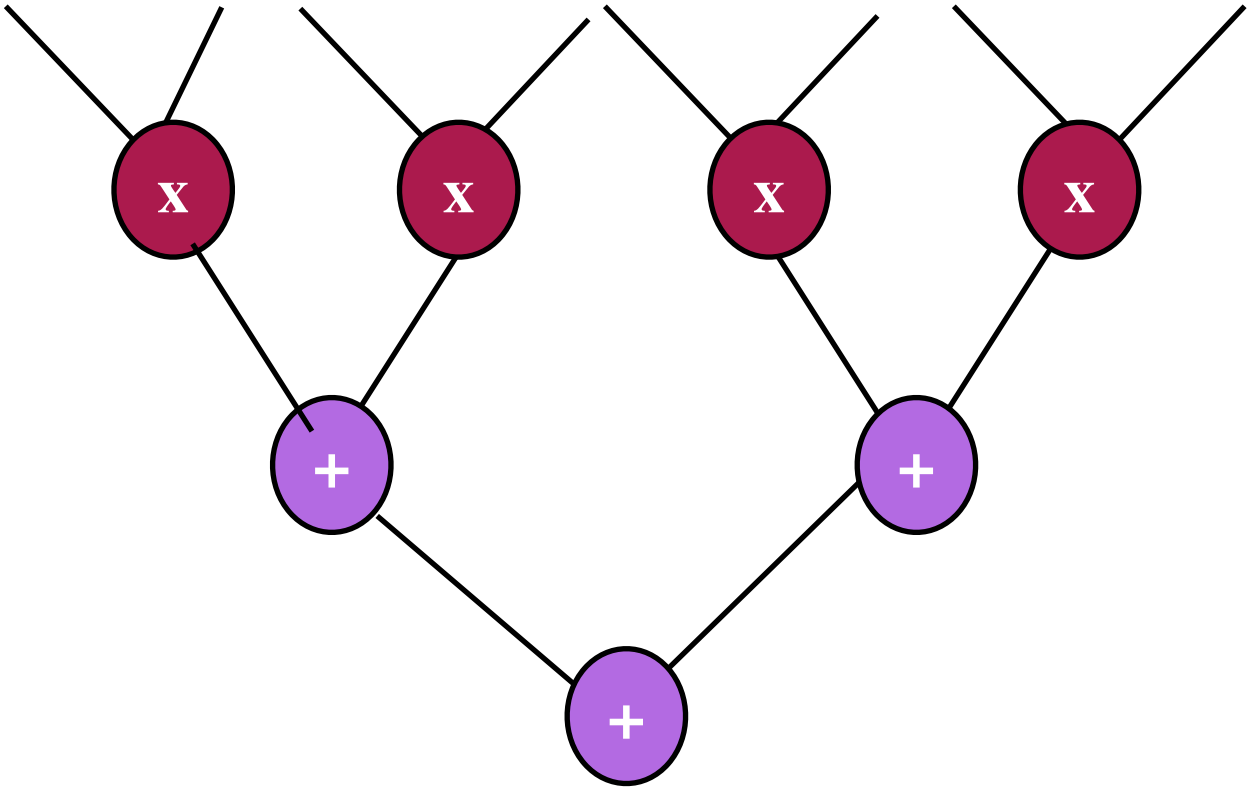
Example: dependency graph for dense-matrix vector product



Questions:

- What is the maximum number of tasks possible?
- What does a task dependency graph look like for this case?
- What is the shortest parallel execution time for the graph?
- What is the maximum degree of concurrency?
- How many processors are needed to achieve the minimum time?
- What is the average parallelism?

Computation Graph:
Dot Product for One Row



Limits on Parallel Performance

- What bounds parallel execution time?
  - —minimum task granularity
    - e.g. dense matrix-vector multiplication ≤ $O(n^2)$ concurrent tasks
  - —dependencies between tasks
  - —parallelization overheads
    - e.g., cost of communication between tasks
  - —fraction of application work that can't be parallelized
    - Amdahl's law
- Measures of parallel performance
  - —speedup = T_1/T_p
  - —parallel efficiency = $T_1/(pT_p)$

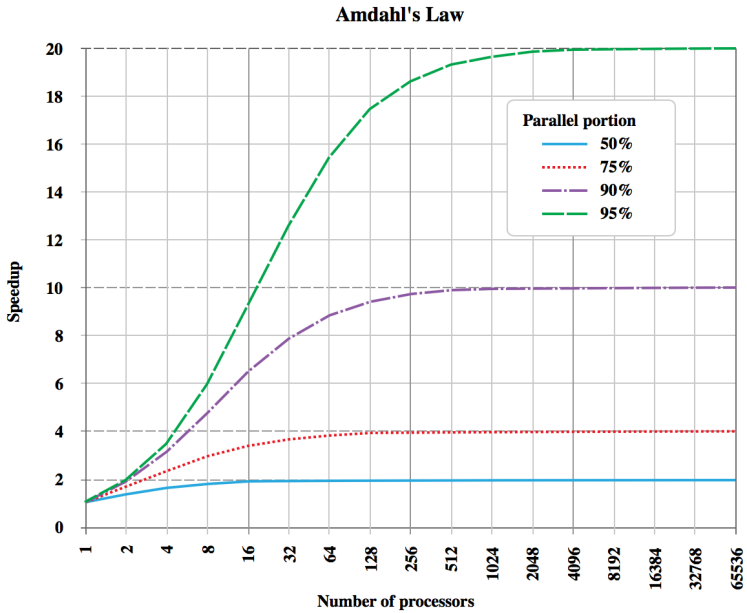
Amdahl's Law

- **A hard limit on the speedup that can be obtained using multiple CPUs**
- **Two expressions of Amdahl's law**
  - **—execution time on N CPUs**

$$t_N = (f_p/N + f_s)t_1$$

  - **—speedup on N CPUs**

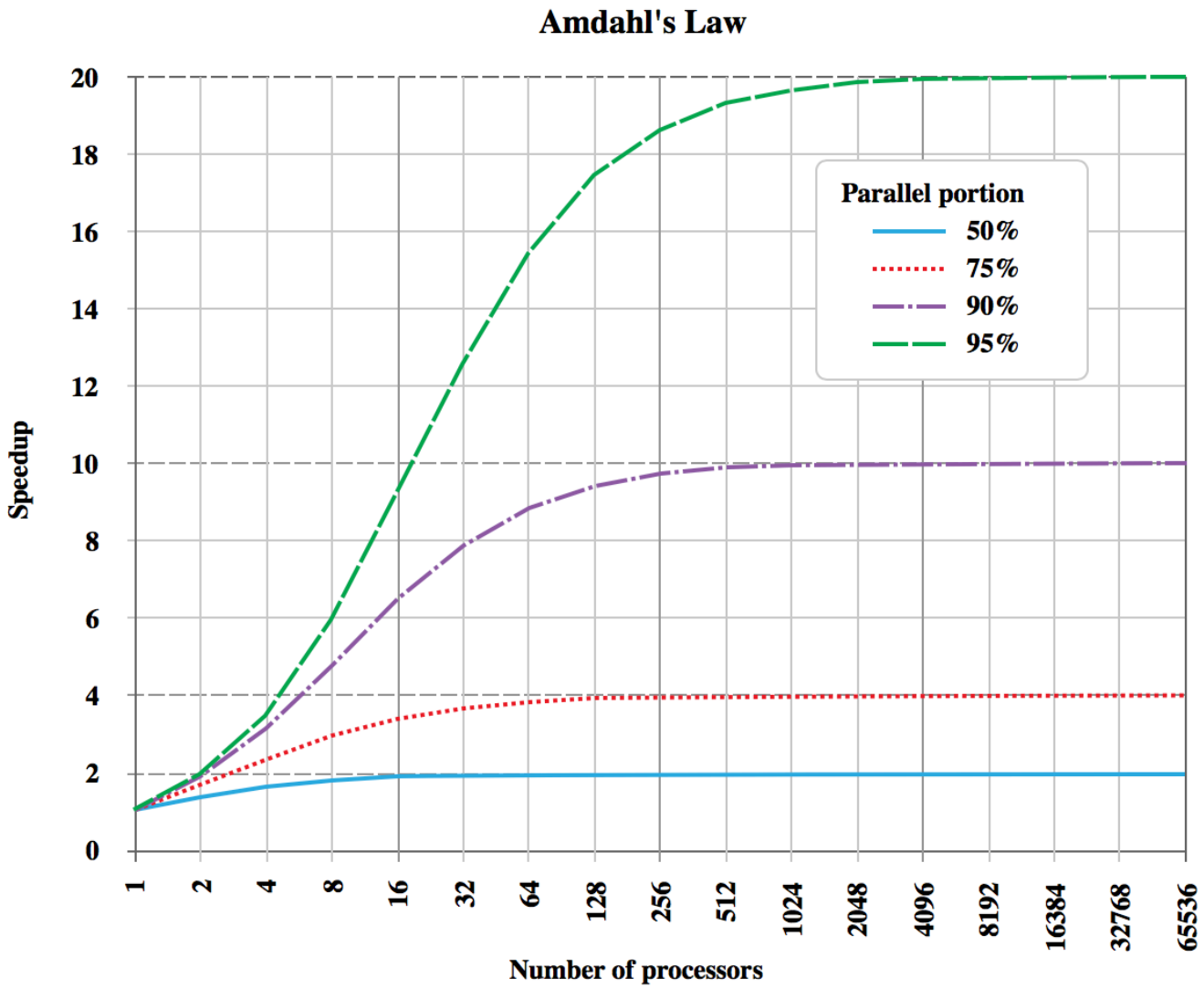
$$S = 1/(f_s + f_p/N)$$



https://en.wikipedia.org/wiki/Amdahl's_law

$$t_1 : time\ on\ 1\ CPU$$
$$t_N : time\ on\ N\ CPUs$$
$$f_p : parallel\ fraction$$
$$f_s : serial\ fraction = 1 - f_p$$

Amdahl's Law



https://en.wikipedia.org/wiki/Amdahl's_law

Task Interaction Graphs

- Tasks generally exchange data with others
  - example: dense matrix-vector multiply
    - if vector b is not replicated in all tasks, tasks will have to communicate elements of b
- Task interaction graph
  - node = task
  - edge = interaction or data exchange
- Task interaction graphs vs. task dependency graphs
  - *task interaction graphs* represent data dependences
  - *task dependency graphs* represent control dependences

Task Interaction Graph Example

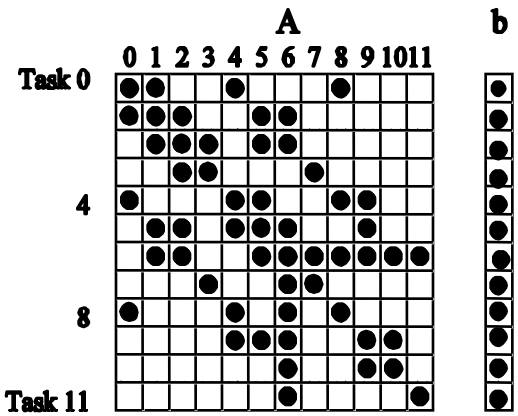
Sparse matrix-vector multiplication

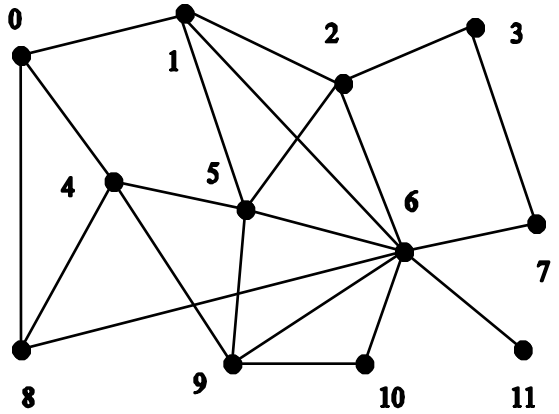
- Computation of each result element = independent task
- Only non-zero elements of sparse matrix *A* participate
- If, b is partitioned among tasks …
  - — structure of the task interaction graph = graph of the matrix *A*

  (i.e. the graph for which *A* represents the adjacency structure)

b[0] is with task0
b[1] is with task1

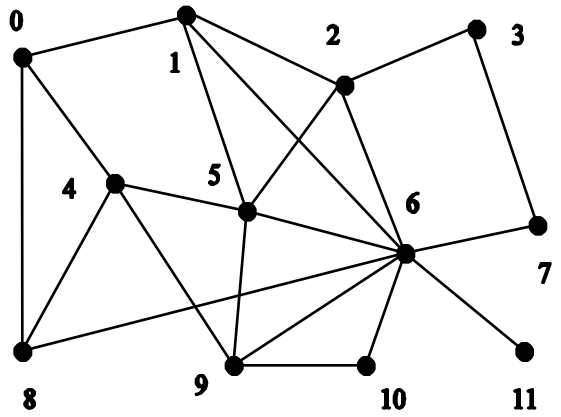
b[11] with task11





Interaction Graphs, Granularity, & Communication

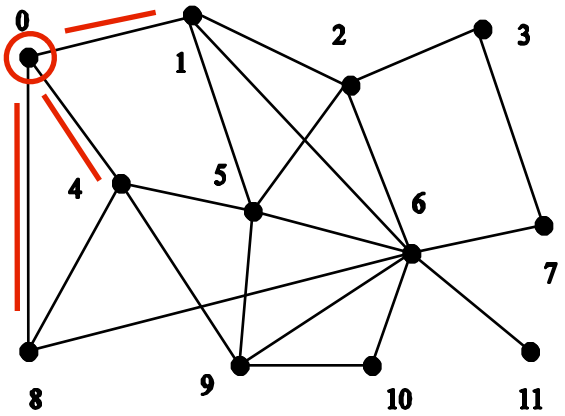
- **Finer task granularity increases communication overhead**
- **Example: sparse matrix-vector product interaction graph**



- **Assumptions:**
  - **each node takes unit time to process**
  - **processing each incident interaction (edge) costs unit time**
- **If node 0 is a task: communication = 3; computation = 4**
- **If nodes 0, 4, and 5 are a task: communication = 5; computation = 15**
  - **coarser-grain decomposition → smaller communication/computation**

Interaction Graphs, Granularity, & Communication

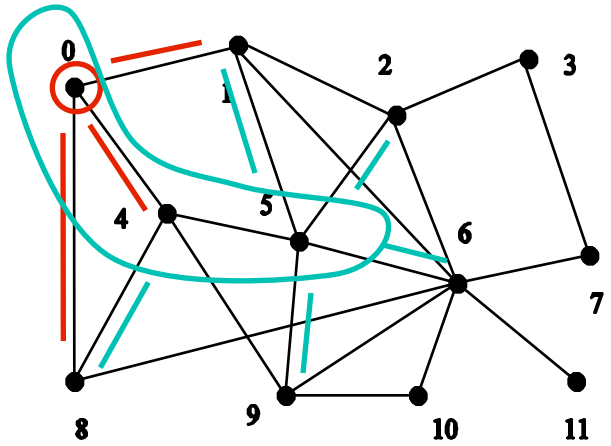
- Finer task granularity increases communication overhead
- Example: sparse matrix-vector product interaction graph



- Assumptions:
  - each node takes unit time to process
  - processing each incident interaction (edge) costs unit time
- If node 0 is a task: communication = 3; computation = 4
- If nodes 0, 4, and 5 are a task: communication = 5; computation = 15
  - coarser-grain decomposition → smaller communication/computation

Interaction Graphs, Granularity, & Communication

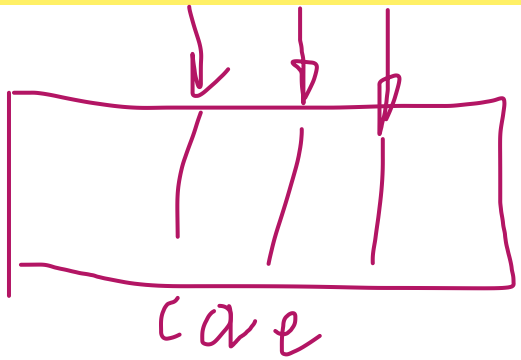
- Finer task granularity increases communication overhead
- Example: sparse matrix-vector product interaction graph



- Assumptions:
  - each node takes unit time to process
  - processing each incident interaction (edge) costs unit time
- If node 0 is a task: communication = 3; computation = 4
- If nodes 0, 4, and 5 are a task: communication = 5; computation = 15
  - coarser-grain decomposition → smaller communication/computation

Tasks, Threads, and Mapping

- **Generally**
  - —**# of tasks > # threads available**
  - —**parallel algorithm must map tasks to threads**
- **Why threads rather than CPU cores?**
  - —**aggregate tasks into threads**
    - **thread = processing or computing agent that performs work**
    - **assign collection of tasks and associated data to a thread**
  - —**operating system maps threads to physical cores**
    - **operating systems often enable one to bind a thread to a core**
    - **for multithreaded cores, the OS can bind multiple software threads to distinct hardware threads associated with a core**



Tasks, Threads, and Mapping

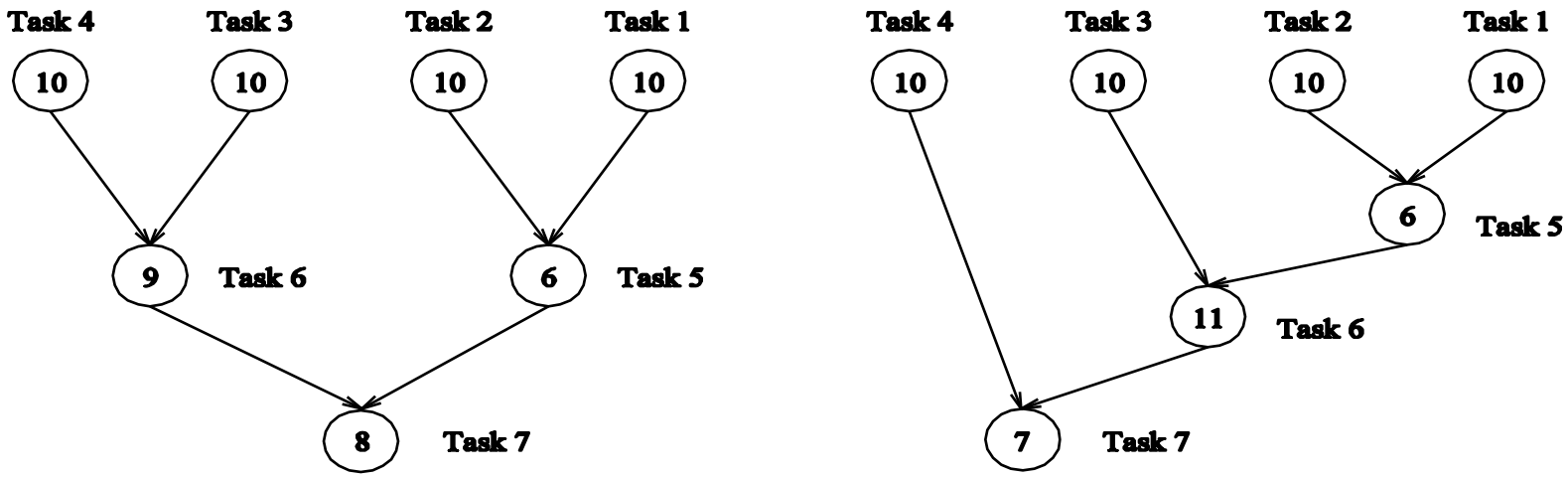
- Mapping tasks to threads is critical for parallel performance
- On what basis should one choose mappings?
  - —using task dependency graphs
    - schedule concurrent tasks on separate threads
      - minimum execution time
      - load balance
  - —using task interaction graphs
    - want threads to have minimum interaction with one another
      - minimum communication

Tasks, Threads, and Mapping

A good mapping must minimize parallel execution time by

- **Mapping independent tasks to different threads**
- **Assigning tasks on critical path to threads ASAP**
- **Minimizing interactions between threads**
  - **—map tasks with dense interactions to the same thread**
- **Difficulty: criteria often conflict with one another**
  - **—e.g. no decomposition minimizes interactions but no speedup!**

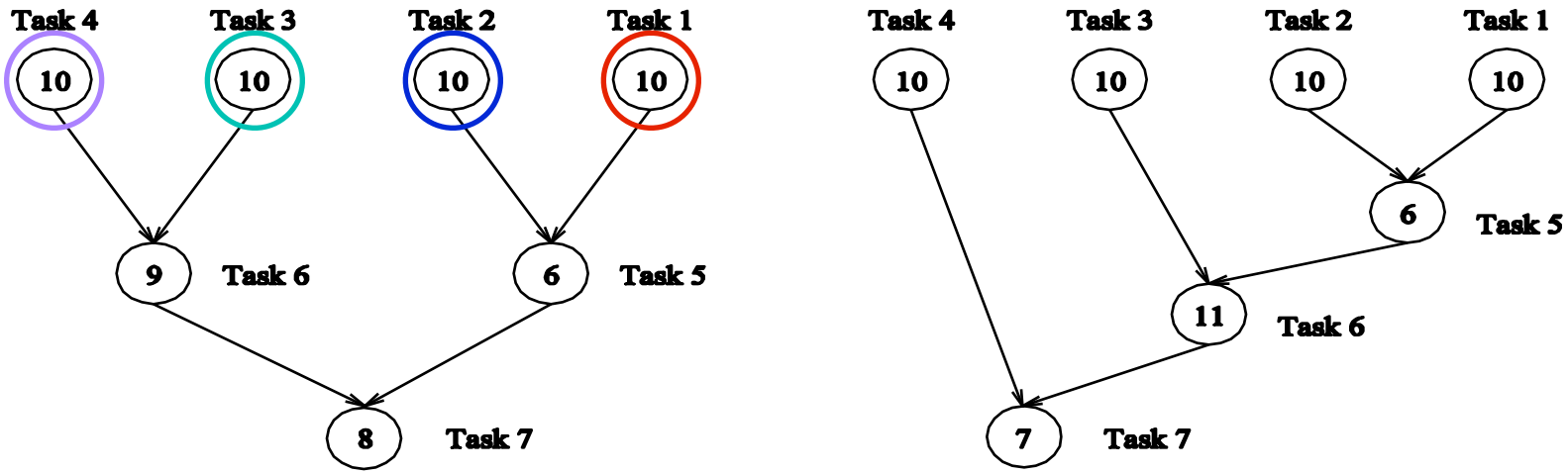
Tasks, Threads, and Mapping Example



Example: mapping database queries to threads

- **Consider the dependency graphs in levels**
  - **no nodes in a level depend upon one another**
  - **compute levels using topological sort**
- **Assign all tasks within a level to different threads**

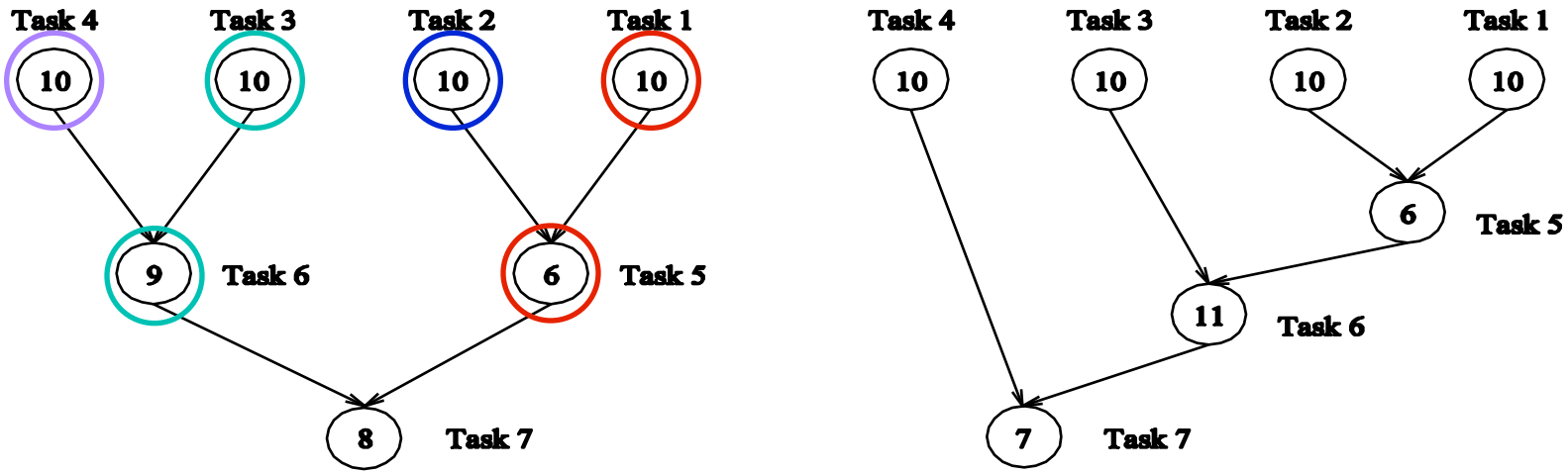
Tasks, Threads, and Mapping Example



Example: mapping database queries to threads

- **Consider the dependency graphs in levels**
  - **no nodes in a level depend upon one another**
  - **compute levels using topological sort**
- **Assign all tasks within a level to different threads**

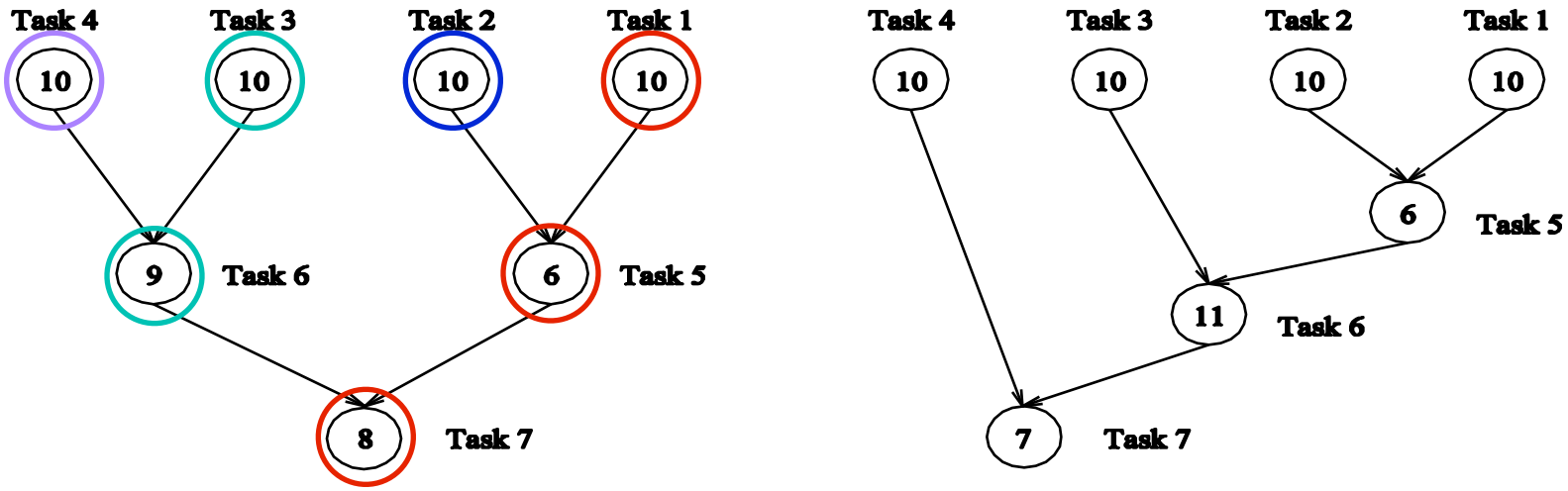
Tasks, Threads, and Mapping Example



Example: mapping database queries to threads

- **Consider the dependency graphs in levels**
  - **no nodes in a level depend upon one another**
  - **compute levels using topological sort**
- **Assign all tasks within a level to different threads**

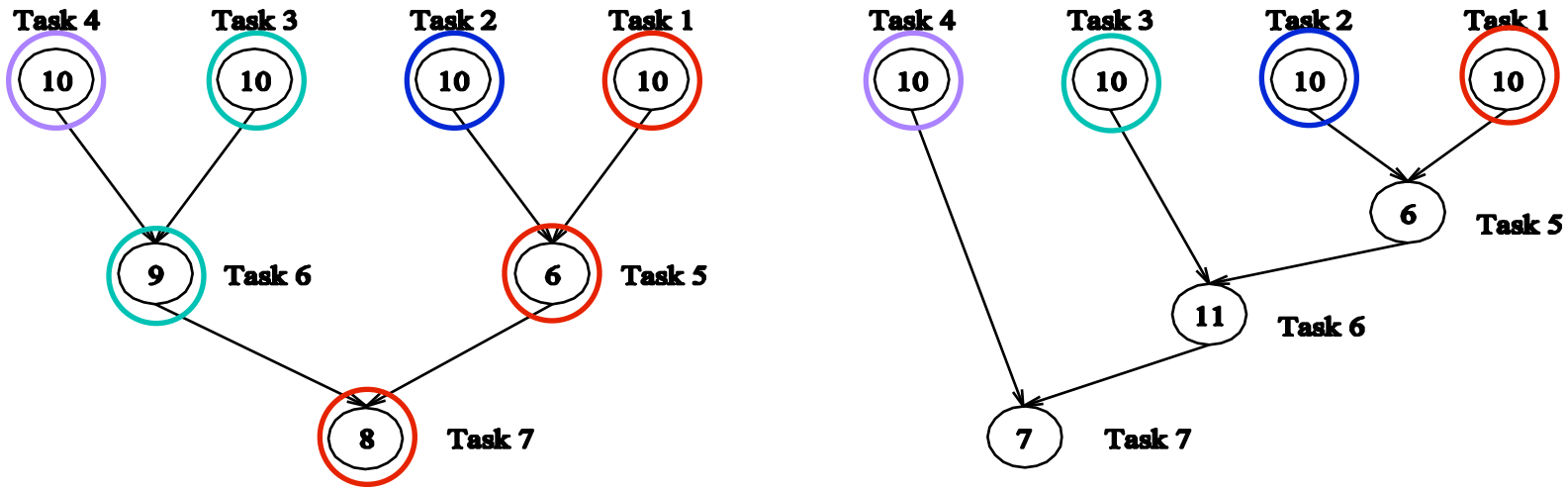
Tasks, Threads, and Mapping Example



Example: mapping database queries to threads

- **Consider the dependency graphs in levels**
  - **no nodes in a level depend upon one another**
  - **compute levels using topological sort**
- **Assign all tasks within a level to different threads**

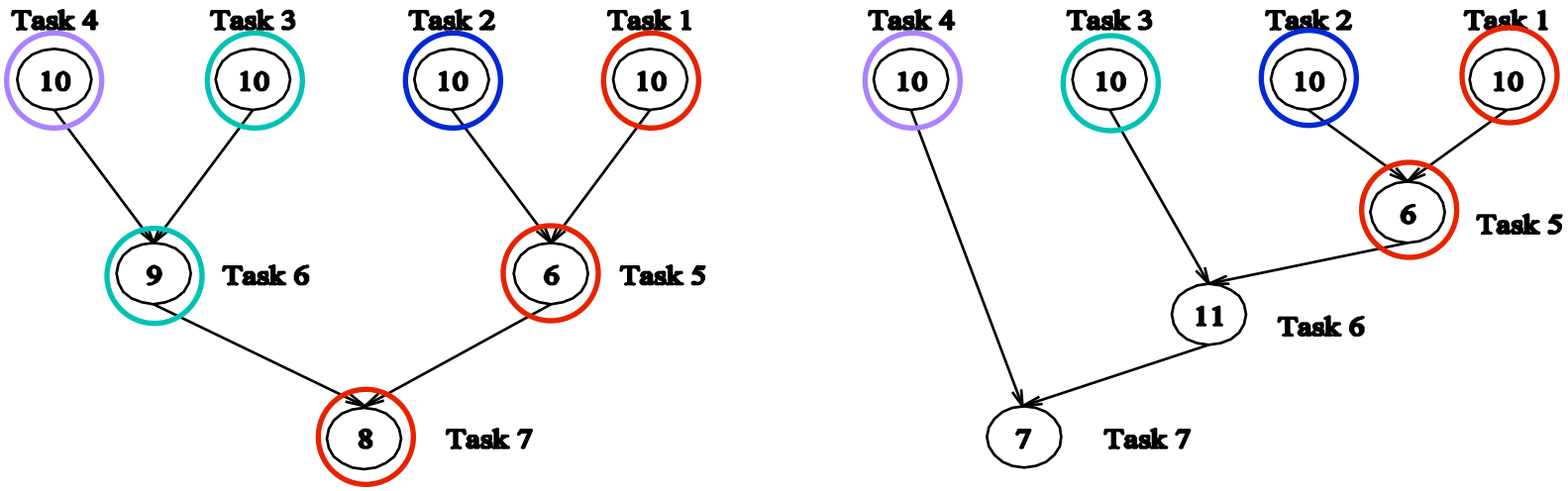
Tasks, Threads, and Mapping Example



Example: mapping database queries to threads

- **Consider the dependency graphs in levels**
  - **no nodes in a level depend upon one another**
  - **compute levels using topological sort**
- **Assign all tasks within a level to different threads**

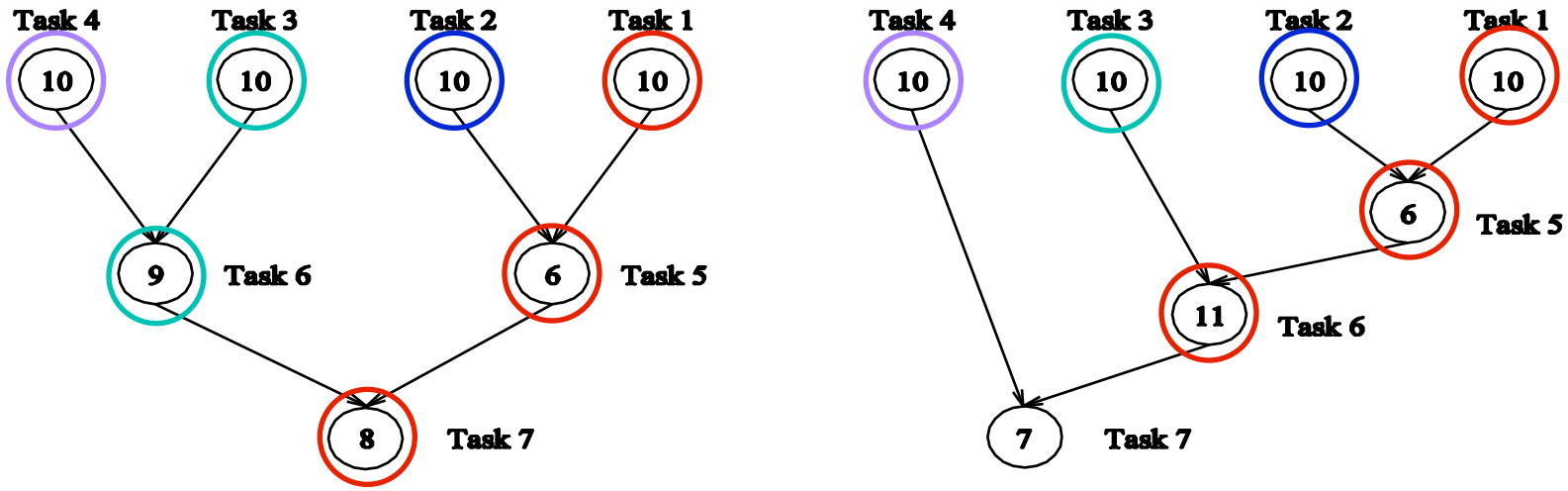
Tasks, Threads, and Mapping Example



Example: mapping database queries to threads

- **Consider the dependency graphs in levels**
  - **no nodes in a level depend upon one another**
  - **compute levels using topological sort**
- **Assign all tasks within a level to different threads**

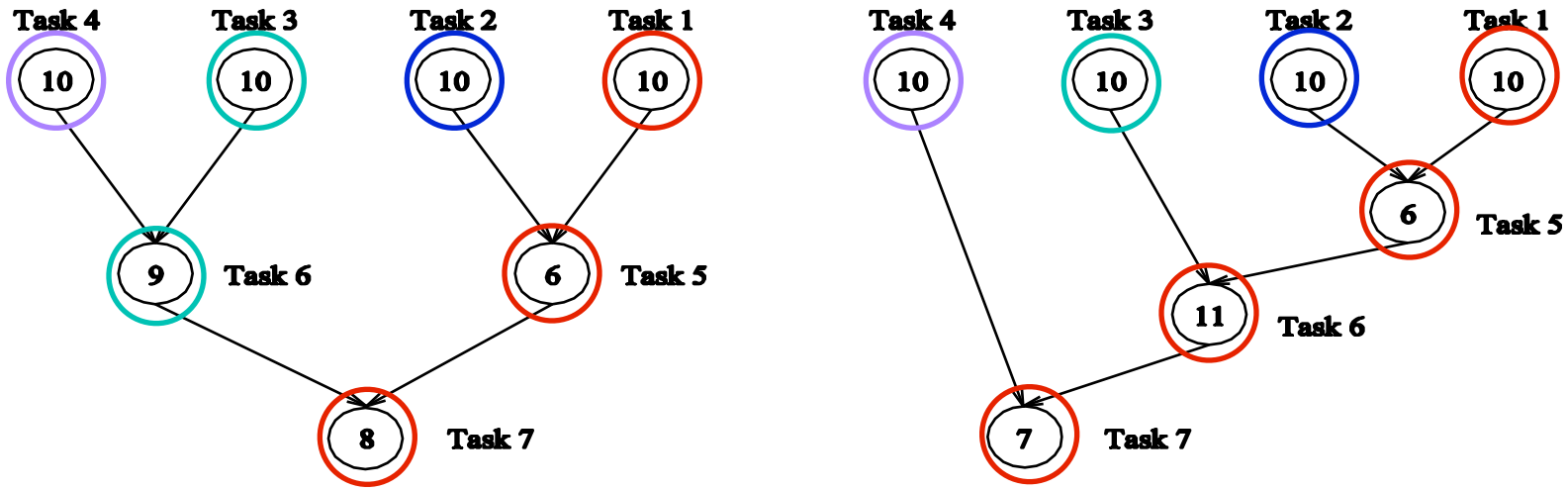
Tasks, Threads, and Mapping Example



Example: mapping database queries to threads

- **Consider the dependency graphs in levels**
  - **no nodes in a level depend upon one another**
  - **compute levels using topological sort**
- **Assign all tasks within a level to different threads**

Tasks, Threads, and Mapping Example



Example: mapping database queries to threads

- **Consider the dependency graphs in levels**
  - **no nodes in a level depend upon one another**
  - **compute levels using topological sort**
- **Assign all tasks within a level to different threads**