Scheduling

…from CPUs to Clusters to Grids…

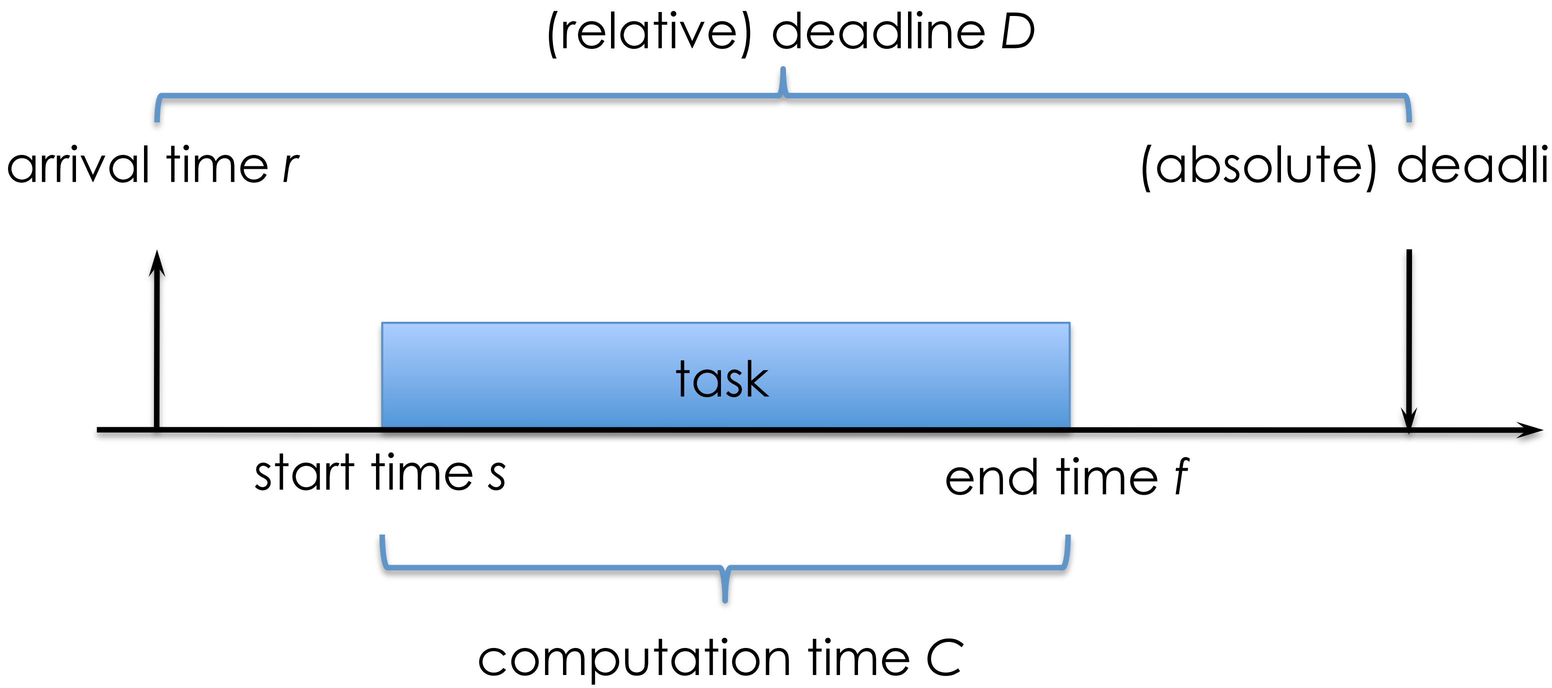
Outline

- Terminology
- CPU Scheduling
- Real-time Scheduling
- Cluster Scheduling
- Grid Scheduling
- Cloud Scheduling

General

- *Scheduling* refers to allocate limited resources to activities over time
  - assigning a resource and a start time to a task
  - A related term is *mapping* that assigns a resource to a task but not the start time
- Activities:
  - executables
  - steps of a project
  - operations
  - lectures
- Resource:
  - processors
  - workers
  - machines
  - rooms

Terminology



- Lateness *L* = *f* – *d* (can be negative)
- Tardiness *E* = *max(0, L)*
- Laxity *Lx* = *D* – *C*
- Completion time Rt = *f* – *r* (a.k.a. response time)

General Problem

Assign a set of tasks to a limited set of resources and find starting times for each task in such a way that some constraints are satisfied and some objective function is minimized.

- Constraints
  - Temporal (deadlines)
  - Precedence (DAGs)
  - Resource (sharing)
- Objective functions:
  - Maximum lateness
  - Total tardiness
  - Average response time
  - Average weighted response time
  - Total computation time
  - Number of late tasks
  - Schedulability

Taxonomies

- Scheduling taxonomy:
  - Online/Offline
  - Local/Global
  - Optimal/Suboptimal
  - Approximate/Heuristic
- System taxonomy:
  - Real-time
  - General purpose
  - Parallel
  - Distributed
  - Shared
  - Heterogeneous

Basic CPU Scheduling

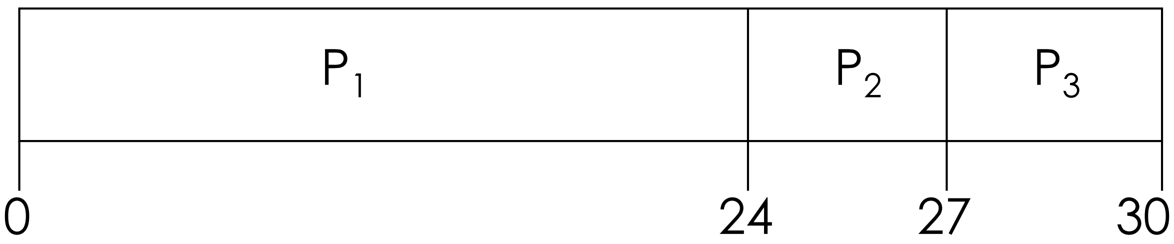
Basic CPU Scheduling

- First Come First Served (FCFS)
- Round Robin (RR)
- Shortest Job First (SJF)
- Multilevel Queue (MLQ)

FCFS

- Simple "first in first out" queue
- Assign the resource to the first task in queue
- Long average waiting time
- Non-preemptive

Process	C
P_1	24
P_2	3
P_3	3



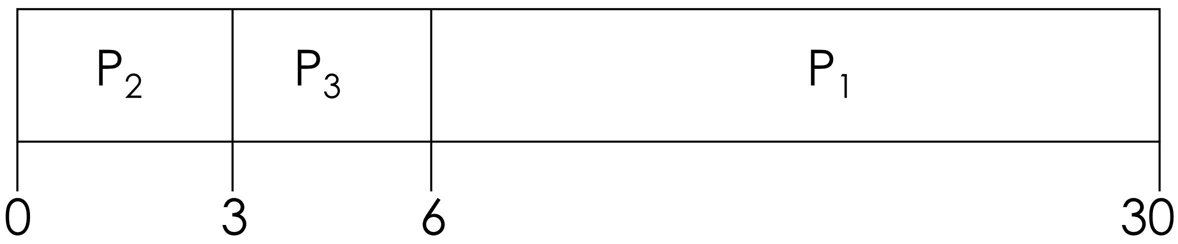
- Average waiting time: $(0 + 24 + 27)/3 = 17$

Example

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:



- Average waiting time: (6 + 0 + 3)/3 = 3

RR

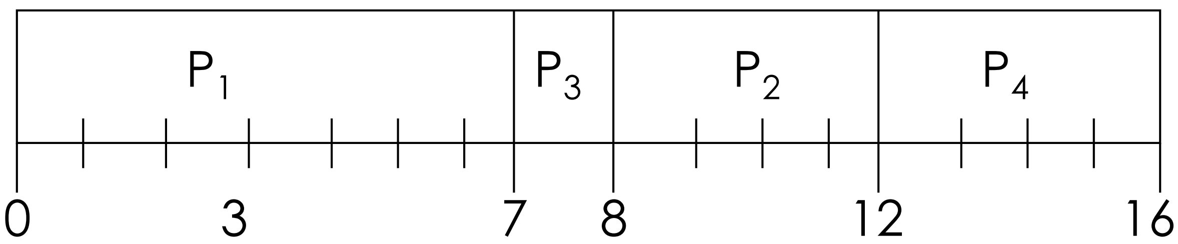
- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1) q$ time units.
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high.

SJF

- Order the tasks in increasing order of computation time
- Assign the CPU to the first task in queue
- Can be preemptive
- SJF gives minimum average waiting time

Example (Non-Preemptive)

Process	r	C
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



Average waiting time = (0 + 6 + 3 + 7)/4 = 4

MLQ

- A process can move between the various queues.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Real Time Scheduling

Real-Time Scheduling

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time.
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones.

Rate-Monotonic (RM)

- A set of independent periodic tasks
- Relative deadline is period
- Static priority scheduling: the shorter the period of a task, the higher is its priority
- The tasks can be scheduled by the rate monotonic policy if

  $C_1/P_1 + C_2/P_2 + \ldots + C_n/P_n \leq n\ (2^{1/n} - 1)$

  The upper bound on utilization is ln2 = 0.69 as *n* approaches infinity.
- If RM can not find a schedule for a set of independent periodic tasks, no other static priority assignment strategy can find a feasible schedule

Earliest Deadline First (EDF)

- Dynamic Priority Scheduling
- The first and the most effectively widely used dynamic priority-driven scheduling algorithm.
- Effective for both preemptive and scheduling periodic and aperiodic tasks.
- For a set of preemptive periodic, aperiodic, tasks, EDF is optimal in the sense that EDF will find a schedule if a schedule is possible for other algorithms.
- Scheduling periodic and aperiodic non-preemptive tasks is NP-hard.

Cluster Scheduling

Execution Alternatives

Time sharing:

- The local scheduler starts multiple processes per physical CPU with the goal of increasing resource utilization.
  - multi-tasking
- The scheduler may also suspend jobs to keep the system load under control
  - preemption

Space sharing:

- The job uses the requested resources exclusively; no other job is allocated to the same set of CPUs.
  - The job has to be queued until sufficient resources are free.

Job Classifications

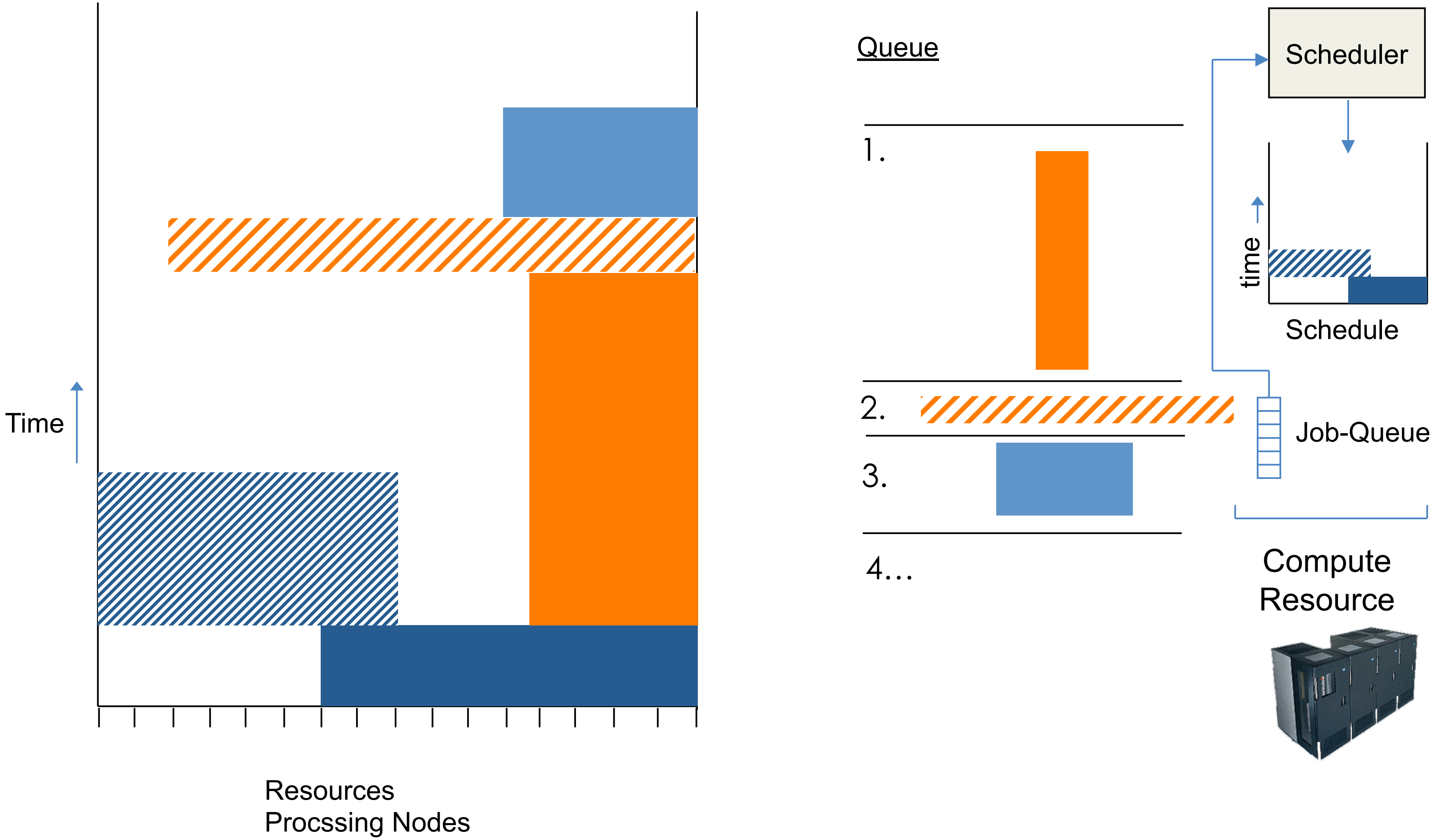
- Batch Jobs vs interactive jobs
  - batch jobs are queued until execution
  - interactive jobs need immediate resource allocation
- Parallel vs. sequential jobs
  - a job requires several processing nodes in parallel

- the majority of HPC installations are used to run batch jobs in space-sharing mode!
  - a job is not influenced by other co-allocated jobs
  - the assigned processors, node memory, caches etc. are exclusively available for a single job.
  - overhead for context switches is minimized
  - important aspects for parallel applications

FCFS

- Well known and very simple: First-Come First-Serve
- Jobs are started in order of submission
- Ad-hoc scheduling when resources become free again
  - no advance scheduling
- Advantage:
  - simple to implement
  - easy to understand and fair for the users (job queue represents execution order)
  - does not require a priori knowledge about job lengths
- Problems:
  - performance can extremely degrade; overall utilization of a machine can suffer if highly parallel jobs occur, that is, if a significant share of nodes is requested for a single job.

FCFS Schedule



Backfilling

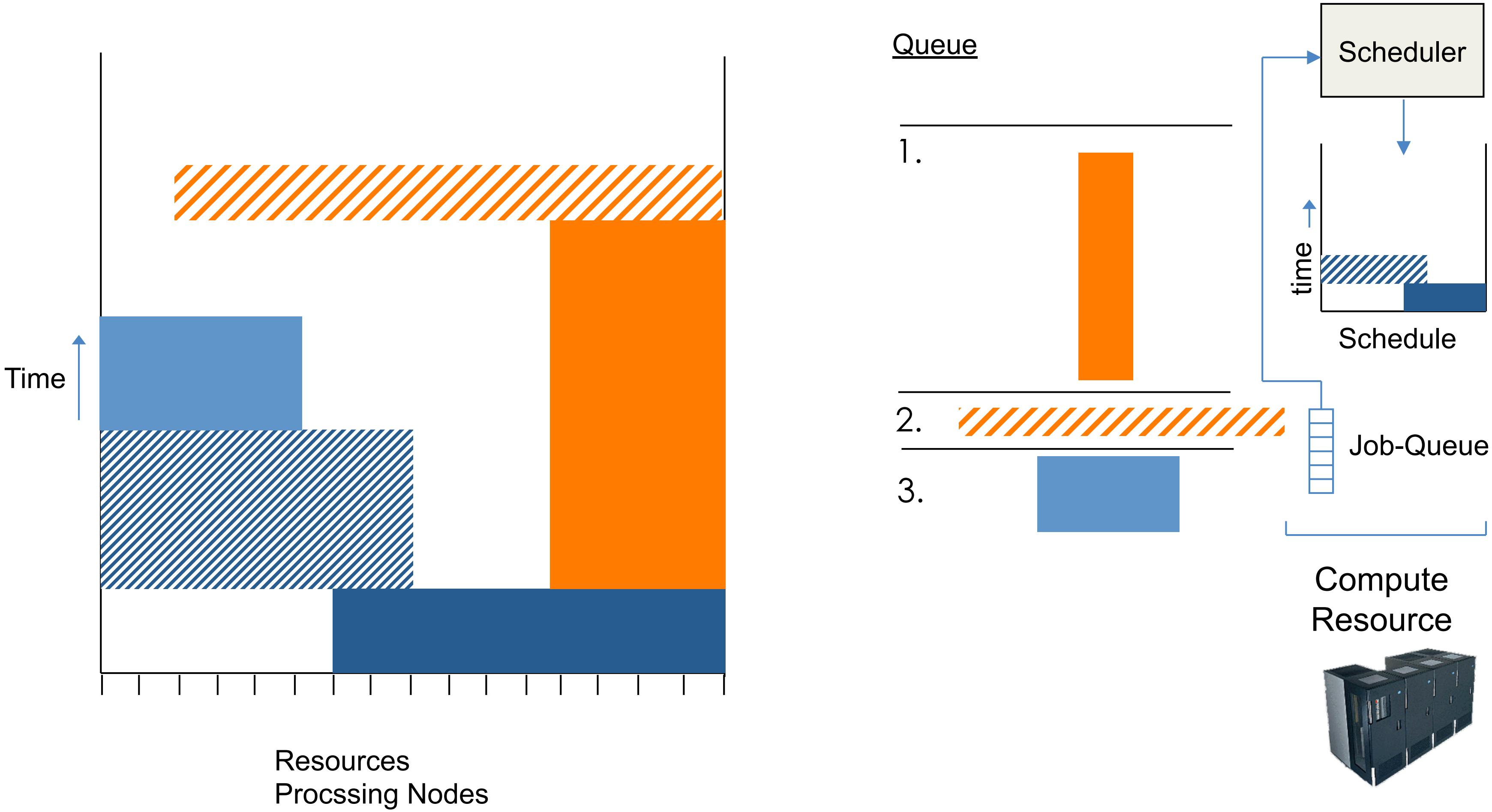
- Improvement over FCFS
- A job can be started before an earlier submitted job if it does not delay the first job in the queue
  - may still cause delay of other jobs further down the queue

- Some fairness is still maintained
- Advantage:
  - utilization is improved

- Information about the job execution length is needed
  - sometimes difficult to provide
  - user estimation not necessarily accurate
  - Jobs are usually terminated after exceeding its allocated execution time;
  - otherwise users may deliberately underestimate the job length to get an earlier job start time

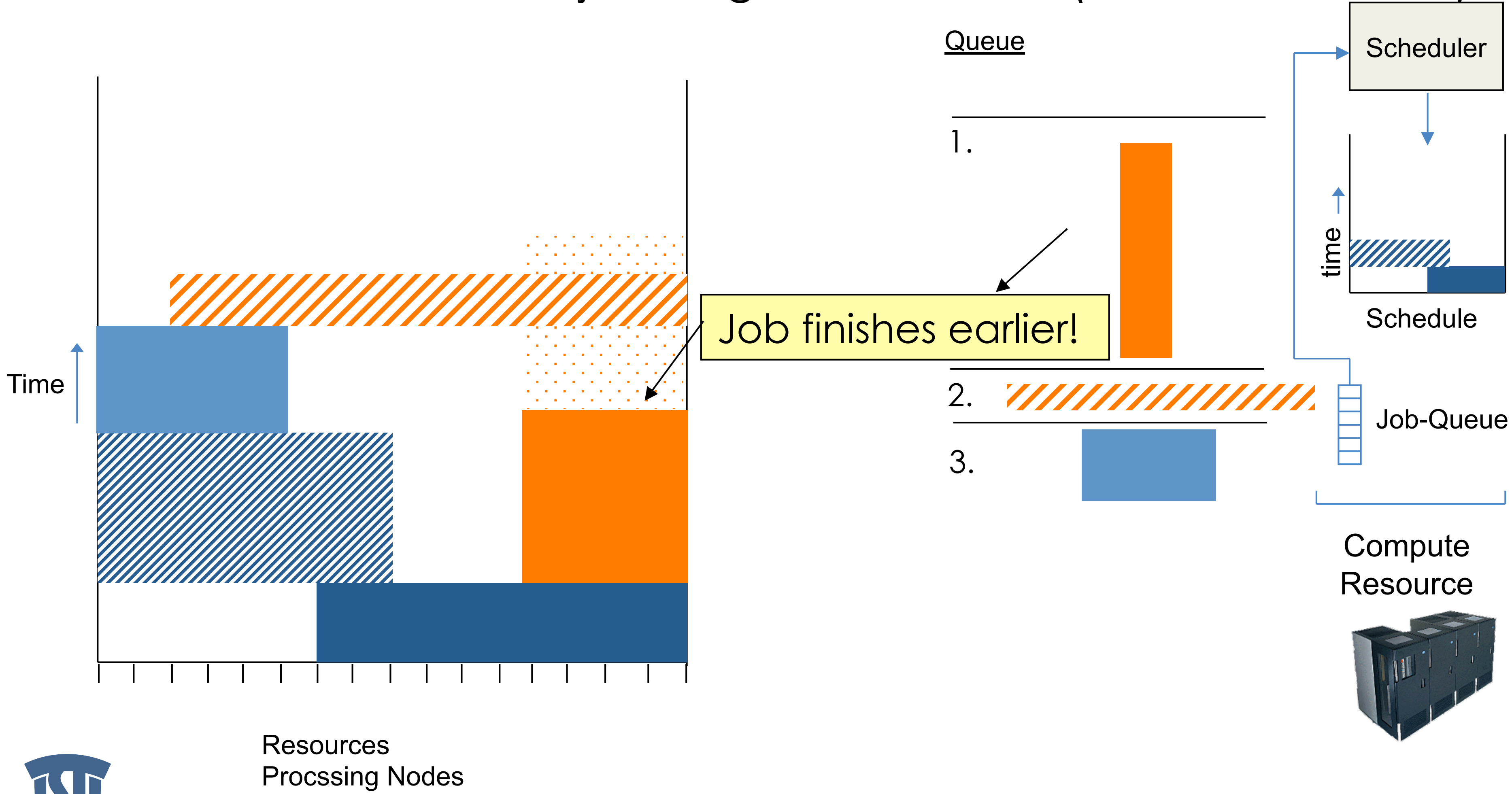
Backfilling Schedule

- Job 3 is started before Job 2 as it does not delay it



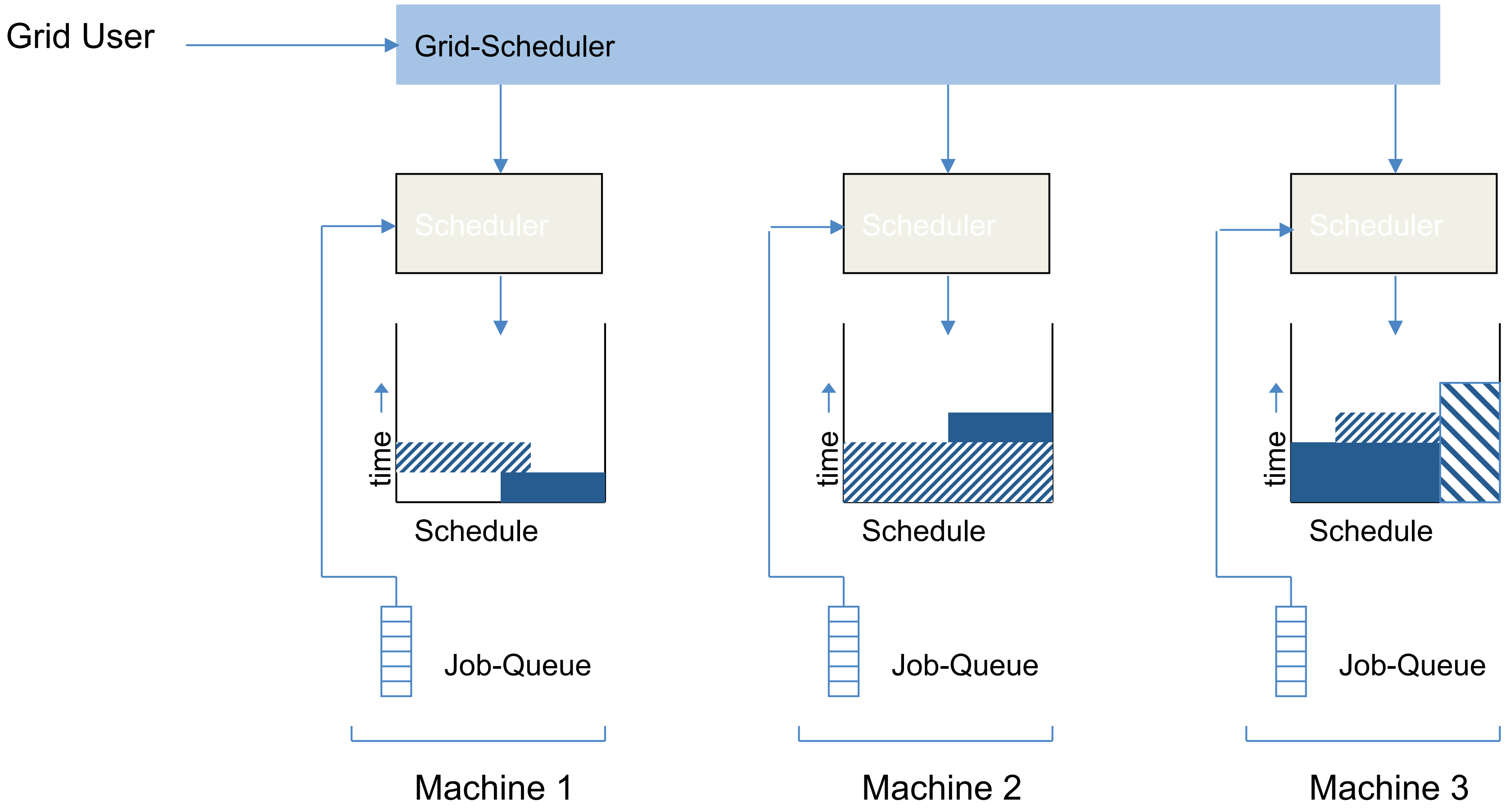
However, if a job finishes earlier than expected, the backfilling causes delays that otherwise would not occur

- need for accurate job length information (difficult to obtain)



Grid Scheduling

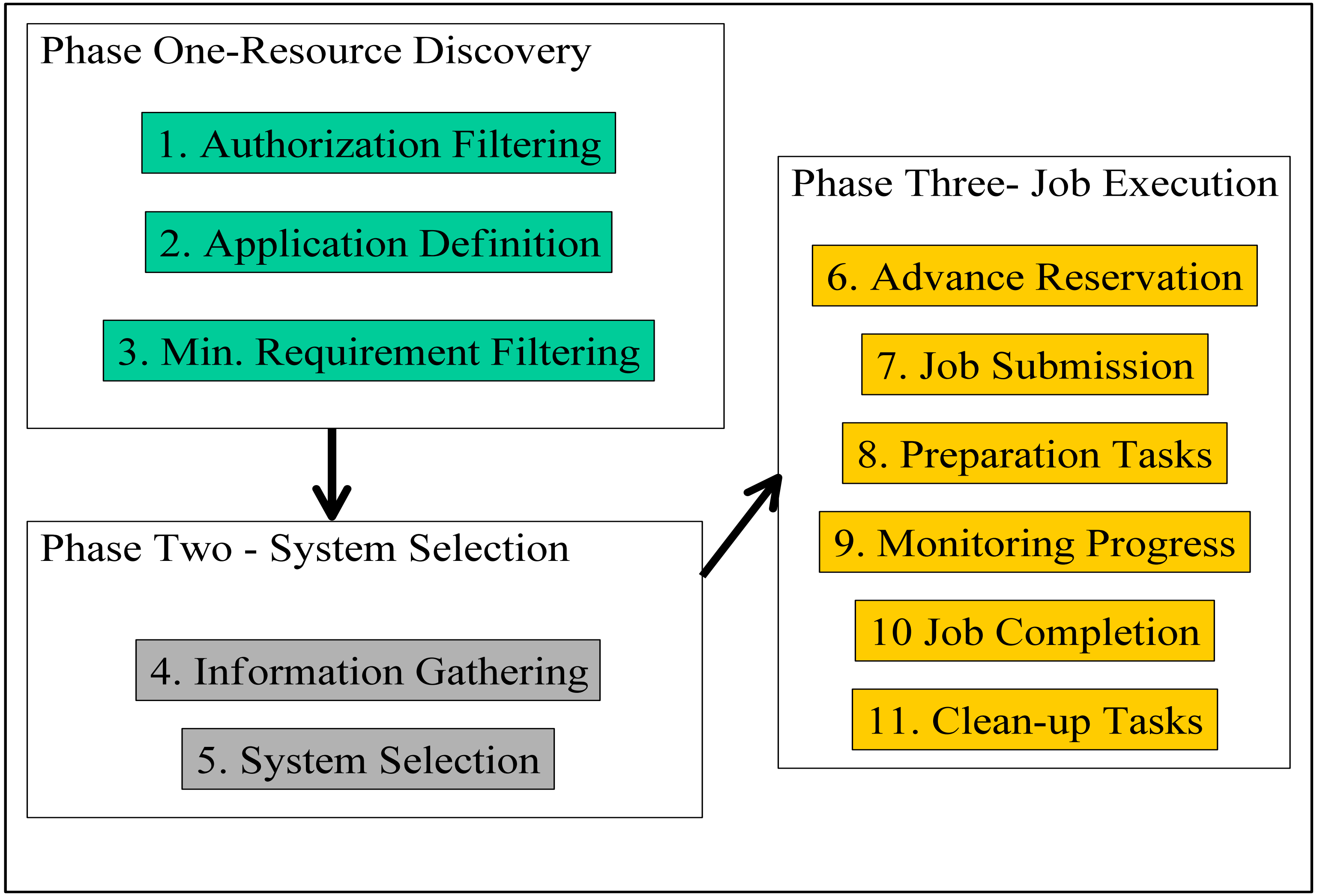
Grid Scheduling



Different Level of Scheduling

- **Resource-level scheduler**
  - low-level scheduler, local scheduler, local resource manager
  - scheduler close to the resource, controlling a supercomputer, cluster, or network of workstations, on the same local area network
  - Examples: Open PBS, PBS Pro, LSF, SGE
- **Enterprise-level scheduler**
  - Scheduling across multiple local schedulers belonging to the same organization
  - Examples: PBS Pro peer scheduling, LSF Multicluster
- **Grid-level scheduler**
  - also known as super-scheduler, broker, community scheduler
  - Discovers resources that can meet a job's requirements
  - Schedules across lower level schedulers

Activities of a Grid Scheduler



Grid Scheduling

- A Grid scheduler allows the user to specify the required resources and environment of the job without having to indicate the exact location of the resources
  - A Grid scheduler answers the question: to which local resource manger(s) should this job be submitted?

- Answering this question is hard:
  - resources may dynamically join and leave a computational grid
  - not all currently unused resources are available to grid jobs:
    - resource owner policies such as "maximum number of grid jobs allowed"
  - it is hard to predict how long jobs will wait in a queue

Select a Resource for Execution

- Most systems do not provide advance information about future job execution
  - user information not accurate as mentioned before
  - new jobs arrive that may surpass current queue entries due to higher priority

- Grid scheduler might consider current queue situation, however this does not give reliable information for future executions:
  - A job may wait long in a short queue while it would have been executed earlier on another system.

- Available information:
  - Grid information service gives the state of the resources and possibly authorization information
  - Prediction heuristics: estimate job's wait time for a given resource, based on the current state and the job's requirements.

Selection Criteria

- Distribute jobs in order to balance load across resources
  - not suitable for large scale grids with different providers
- Data affinity: run job on the resource where data is located
- Use heuristics to estimate job execution time.
- Best-fit: select the set of resources with the smallest capabilities and capacities that can meet job's requirements

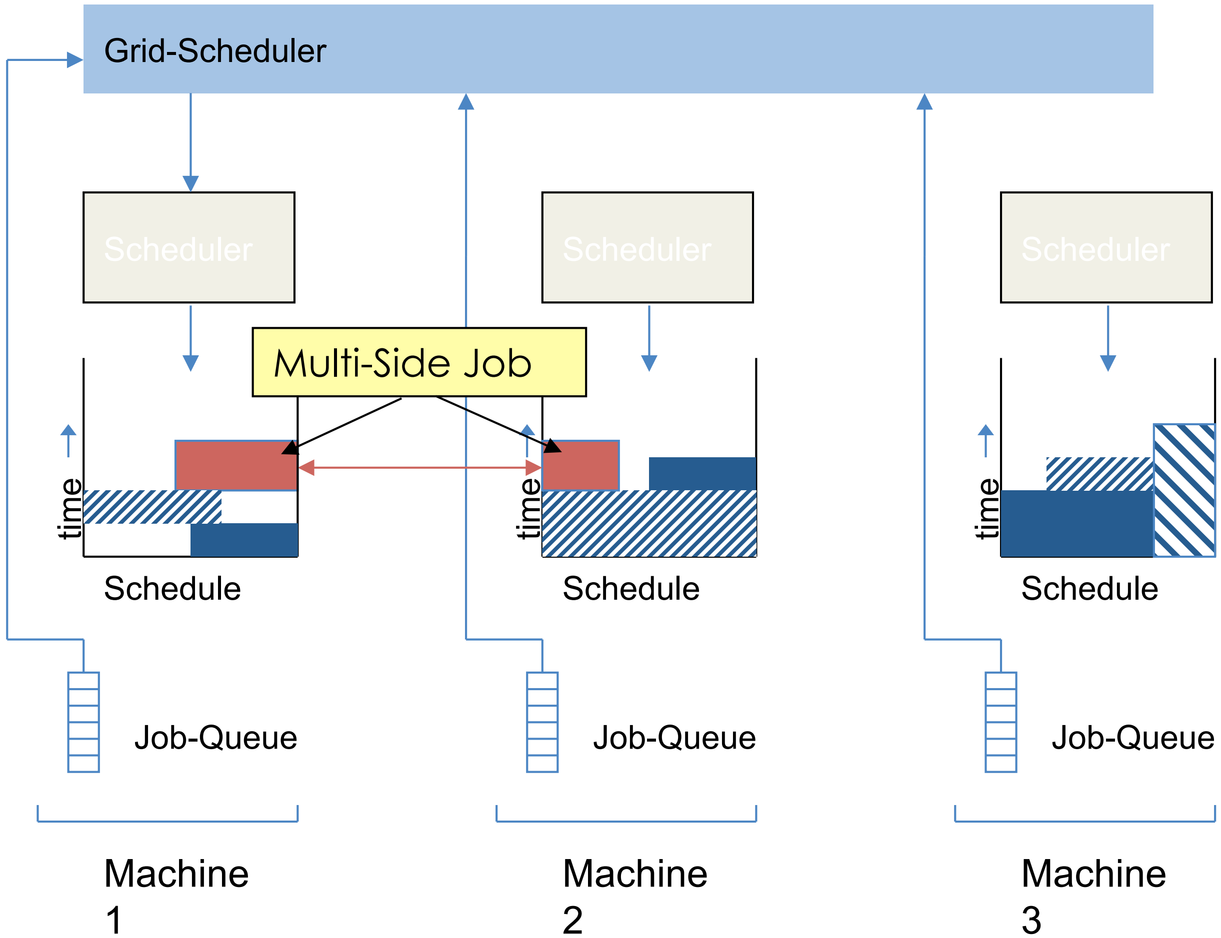
Co-allocation

- It is often requested that several resources are used for a single job.
  - that is, a scheduler has to assure that all resources are available when needed.
    - in parallel (e.g. visualization and processing)
    - with time dependencies (e.g. a workflow)

- The task is especially difficult if the resources belong to different administrative domains.
  - The actual allocation time must be known for co-allocation
  - or the different local resource management systems must synchronize each other (wait for availability of all resources)

- Co-allocation and other applications require a priori information about the precise resource availability
- With the concept of advanced reservation, the resource provider guarantees a specified resource allocation.
  - includes a two- or three-phase commit for agreeing on the reservation

Example Multi-Site Job Execution



- A job uses several resources at different sites in parallel.
- ➡ Network communication is an issue.

Available Information from the Local Schedulers

- Decision making is difficult for the Grid scheduler
  - limited information about local schedulers is available
  - available information may not be reliable

- Possible information:
  - queue length, running jobs
  - detailed information about the queued jobs
    - execution length, process requirements,…
  - tentative schedule about future job executions

- These information are often technically not provided by the local scheduler
- In addition, these information may be subject to privacy concerns!

Applications taxonomy

- Bag of tasks – Independent tasks

- Workflows – dependent tasks
  - Generally Directed Acyclic Graphs (DAGs)

Min-Min Heuristic

- For each task determine its minimum completion time over all machines
- Over all tasks find the minimum completion time
- Assign the task to the machine that gives this completion time
- Iterate till all the tasks are scheduled

Example of Min-Min

	T1	T2	T3
M1	140	20	60
M2	100	100	70

Stage 1:
T1-M2 = 100
T2-M1 = 20
T3-M1 = 60
Assign T2 to M1

	T1	T3
M1	160	80
M2	100	70

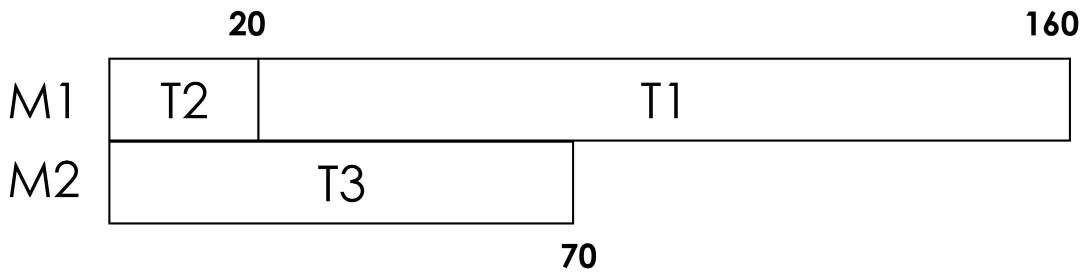
Stage 2:
T1-M2 = 100
T3-M2 = 70

Assign T3 to M2

	T1
M1	160
M2	170

Stage 3:
T1-M1 = 160

Assign T1 to M1



Max-Min Heuristic

- For each task determine its minimum completion time over all machines
- Over all tasks find the maximum completion time
- Assign the task to the machine that gives this completion time
- Iterate till all the tasks are scheduled

Example of Max-Min

	T1	T2	T3
M1	140	20	60
M2	100	100	70

Stage 1:
T1-M2 = 100
T2-M1 = 20
T3-M1 = 60
Assign T1 to M2

	T2	T3
M1	20	60
M2	200	170

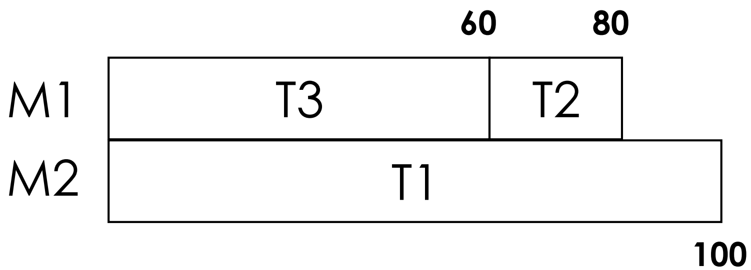
Stage 2:
T2-M1 = 20
T3-M1 = 60

Assign T3 to M1

	T2
M1	80
M2	200

Stage 3:
T2-M1 = 80

Assign T2 to M1



Sufferage Heuristic

- For each task determine the difference between its minimum and second minimum completion time over all machines (sufferage)
- Over all tasks find the maximum sufferage
- Assign the task to the machine that gives this sufferage
- Iterate till all the tasks are scheduled

Example of Sufferage

	T1	T2	T3
M1	140	20	60
M2	100	100	70

Stage 1:
T1 = 40
T2 = 80
T3 = 10
Assign T2 to M1

	T1	T3
M1	160	80
M2	100	70

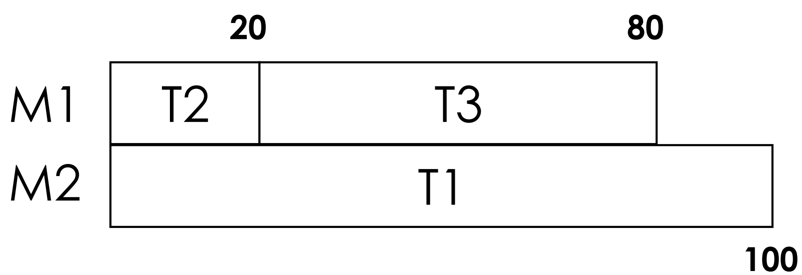
Stage 2:
T1 = 60
T3 = 10

Assign T1 to M2

	T3
M1	80
M2	170

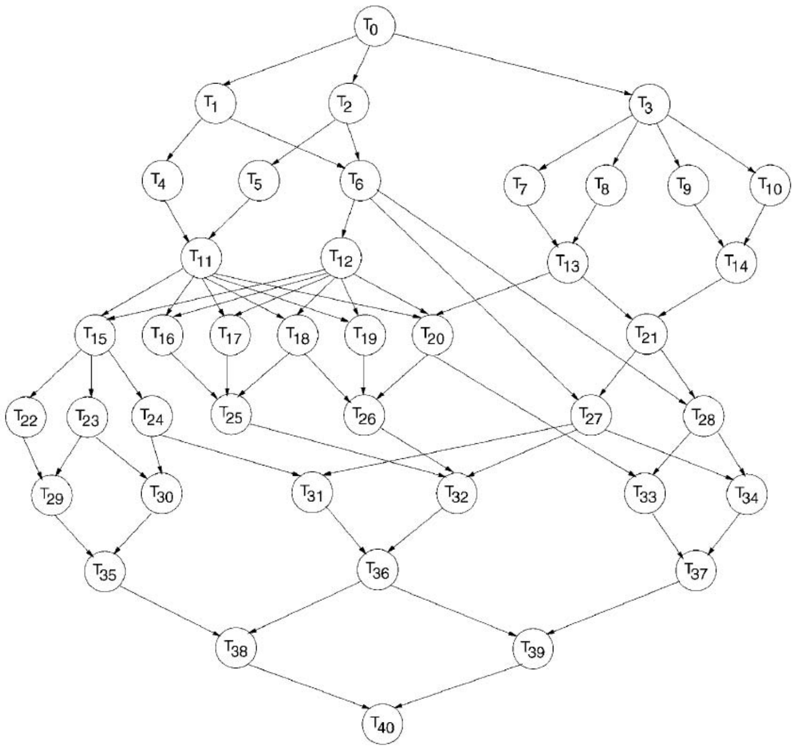
Stage 3:
T3 = 90

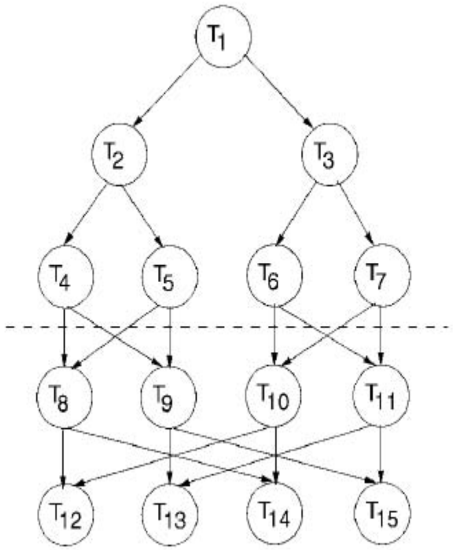
Assign T3 to M1

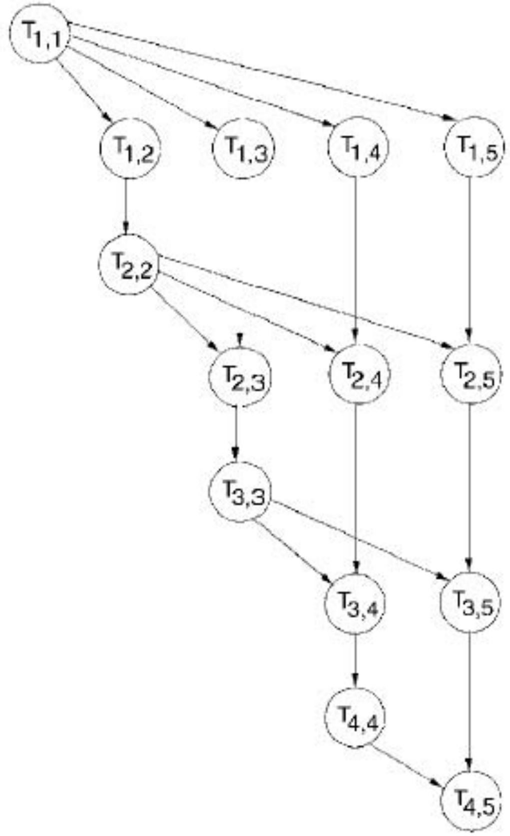


Scheduling Task Graphs

- Task Graphs have dependencies between the tasks in the Application
- Scheduling methods for bag of task applications cannot be directly applied







Guided Random Search Based

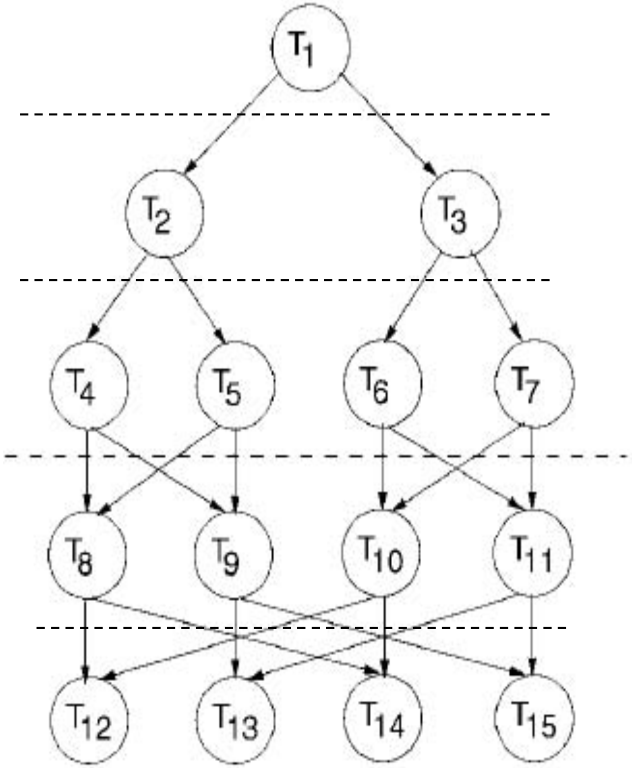
- Genetic Algorithms
  - A chromosome is an ordering of tasks
  - A rule is required to convert it to a schedule
- Simulated Annealing
- Local Search Techniques, taboo, etc…

List Scheduling Heuristics

- An ordered list of tasks is constructed by assigning priority to each task
- Tasks are selected on priority order and scheduled in order to minimize a predefined cost function
- Tasks have to be in a topologically sorted order

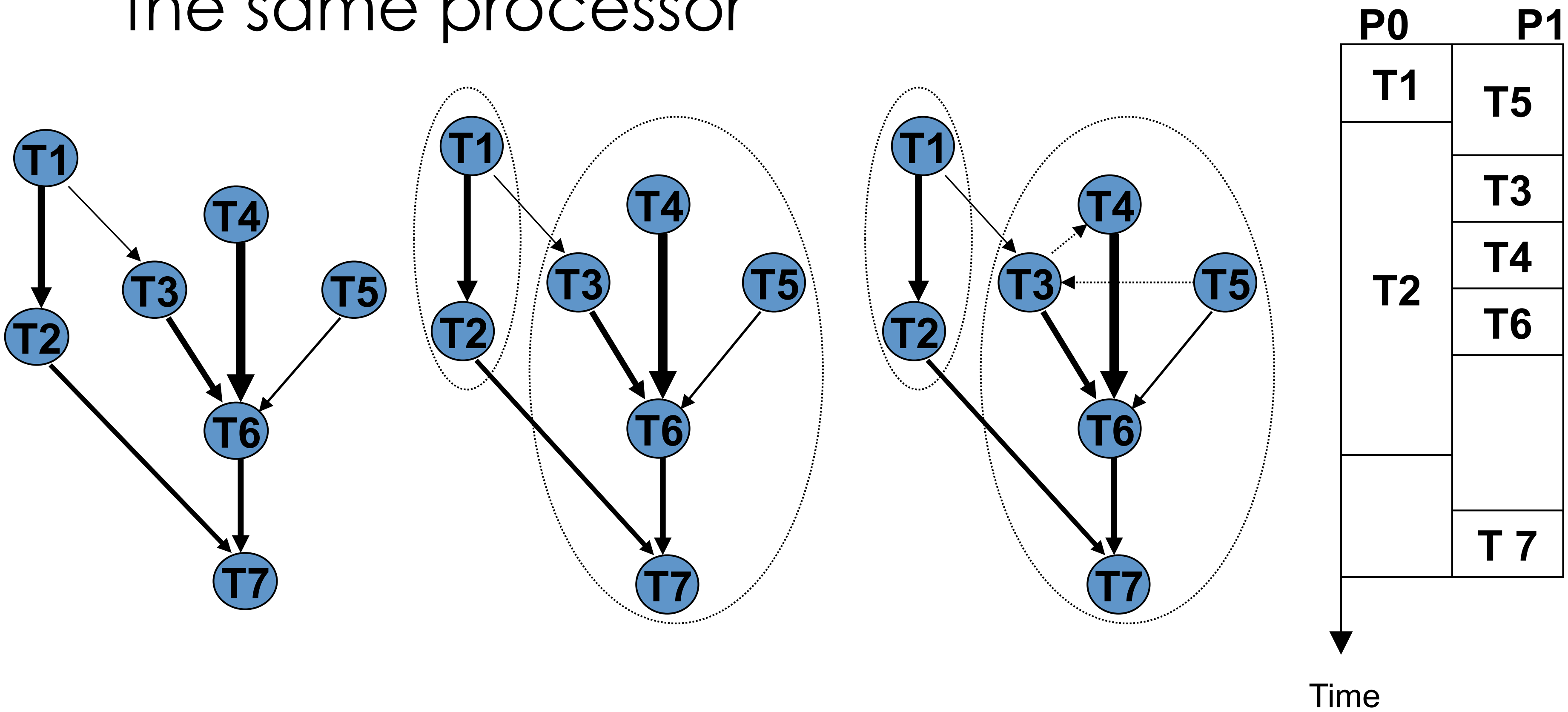
Level by Level Scheduling

- Partition a DAG into multiple levels such that task in each level are independent.
- Apply Min-Min, Max-Min or other heuristics to tasks at each level.



Clustering Heuristics

- Clustering heuristics cluster tasks together
- Tasks in the same cluster are scheduled on the same processor



Scheduling Objectives in the Grid

- In contrast to local computing, there is no general scheduling objective anymore
  - minimizing response time
  - minimizing cost
  - tradeoff between quality, cost, response-time etc.
- Cost and different service quality come into play
  - the user will introduce individual objectives
  - the Grid can be seen as a market where resource are concurring alternatives
- Similarly, the resource provider has individual scheduling policies
- Problem:
  - the different policies and objectives must be integrated in the scheduling process
  - different objectives require different scheduling strategies
  - part of the policies may not be suitable for public exposition (e.g. different pricing or quality for certain user groups)

User Objective

Local computing typically has:

- A given scheduling objective as minimization of response time
- Use of batch queuing strategies
- Simple scheduling algorithms: FCFS, Backfilling

Grid Computing requires:

- Individual scheduling objective
  - better resources
  - faster execution
  - cheaper execution

- More complex objective functions apply for individual Grid jobs!

Provider/Owner Objective

Local computing typically has:

- Single scheduling objective for the whole system:
- e.g. minimization of average weighted response time or high utilization/job throughput

In Grid Computing:

- Individual policies must be considered:
  - access policy,
  - priority policy,
  - accounting policy, and other
- More complex objective functions apply for individual resource allocations!
- User and owner policies/objectives may be subject to privacy considerations!

Economic Scheduling

- Market-oriented approaches are a suitable way to implement the interaction of different scheduling layers
  - agents in the Grid market can implement different policies and strategies
  - negotiations and agreements link the different strategies together
  - participating sites stay autonomous

- Needs for suitable scheduling algorithms and strategies for creating and selecting offers
  - need for creating the Pareto-Optimal scheduling solutions
- Performance relies highly on the available information
  - negotiation can be hard task if many potential providers are available.

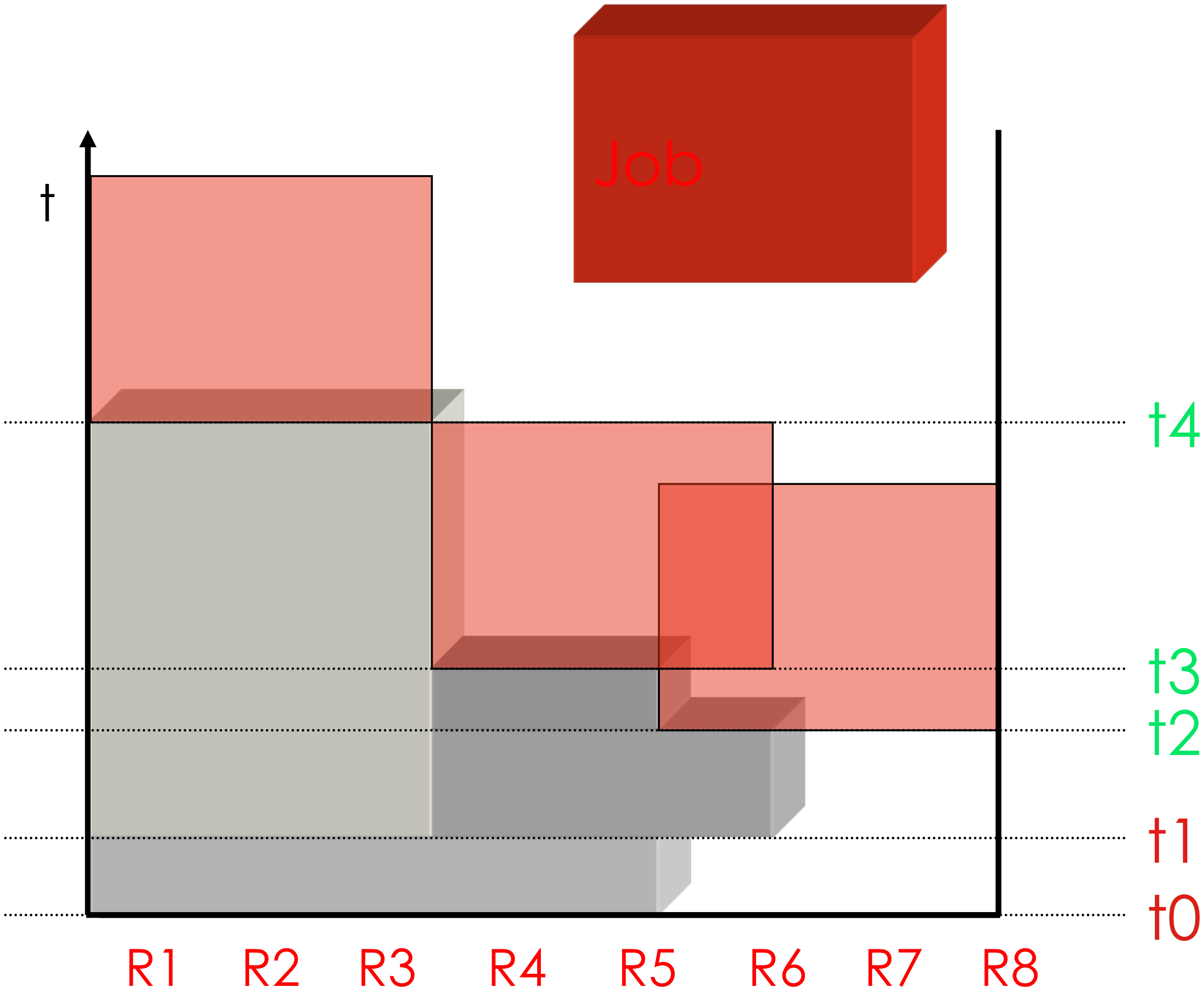
Economic Scheduling (2)

- ➔ Several possibilities for market models:
  - ➔ auctions of resources/services
  - ➔ auctions of jobs

- Offer-request mechanisms support:
  - ➔ inclusion of different cost models, price determination
  - ➔ individual objective/utility functions for optimization goals

- Market-oriented algorithms are considered:
  - ➔ robust
  - ➔ flexible in case of errors
  - ➔ simple to adapt
  - ➔ markets can have unforeseeable dynamics

Offer Creation



Evaluate Offers

- Evaluation with utility functions
  - A utility function is a mathematical representation of a user's preference
  - The utility function may be complex and
  - contain several different criteria

  - Example using response time (or delay time) and price:

$$util = U_{max} - (a_1 \cdot latency + a_2 \cdot price)$$