# Lecture 8 - Batch Scheduling

## Parallel Algorithms – 2022W

Assoc.Prof. Dr. Sascha Hunold

TU Wien

2022-12-14



Informatics

# Last Lecture

- Scheduling with Parallel Tasks
  - Parallel Task Types
  - Scheduling Independent Moldable Tasks

# Batch Scheduling

# Practical Batch Scheduling

- cluster
  - head/login node
  - worker/compute nodes
- batch systems
  - jobs
    - wall-clock time
    - number of nodes, cores
    - desired nodes (if necessary and permitted)
  - queues
    - often several queues
    - used for prioritization

# EASY[1]

- Extensible Argonne Scheduling sYstem
- developed for the first large IBM SP1 installation at Argonne National Lab
- EASY backfill
  - only checks that jobs that move ahead in the queue do not delay the first queued job
  - this aggressive approach can lead to unbounded queuing delays

[1] D. A. Lifka. "An Extensible Job Scheduling System for Massively Parallel Processor Architectures". AAI9833045. PhD thesis. Chicago, IL, USA, 1998. ISBN: 0-591-86149-6.

# Backfill Algorithm in EASY

- first, look for number of available nodes in the system
- if there are not enough nodes to start the first job in queue, calculate
  - shadow-time
    - amount of time that currently available nodes can be used by other jobs (in the queue) without delaying the start of the first job in the queue
  - extra-nodes
    - number of nodes that are available if the first job in the queue were started at shadow-time
- now backfill: search rest of queue to find job that either
  1. can be completed before shadow-time, or
  2. uses extra-nodes many nodes

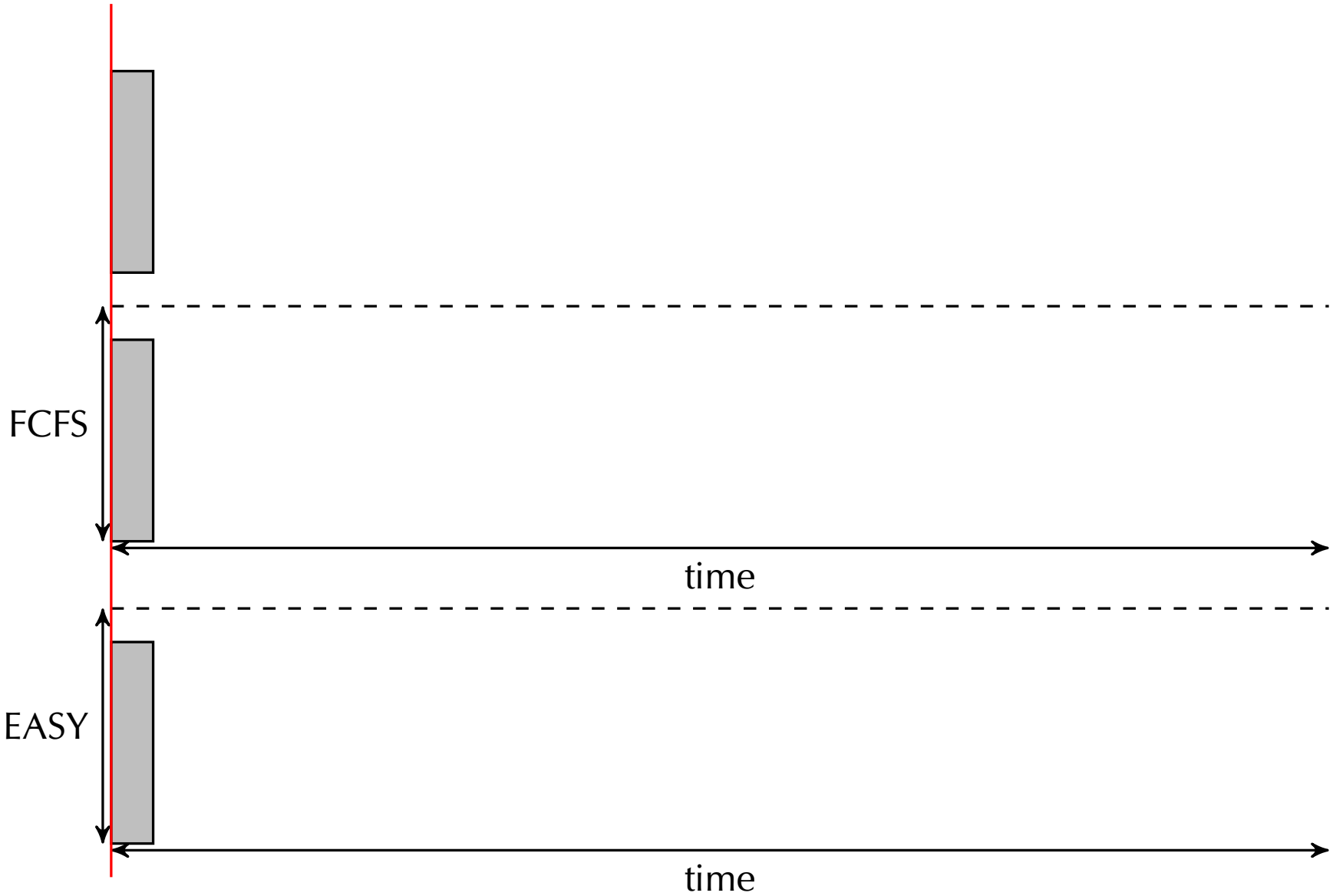
# Backfill Algorithm in EASY - Pseudocode

```
Determine number of nodes in use and available;
If (first Job in the queue can run) {  Start job; }
else
 {calculate shadow-time and number of extra-nodes based on first job;
  /* attempt to backfill around it */
  for (each Job in the ordered list of all other jobs)
   {
    calculate run-time for this job;
    if (required-nodes <= available nodes)
     { if (run-time < shadow-time) || (required nodes <= extra-nodes))
        { Start job;  return; } /* This job can backfill */
     }
   }
 }
```

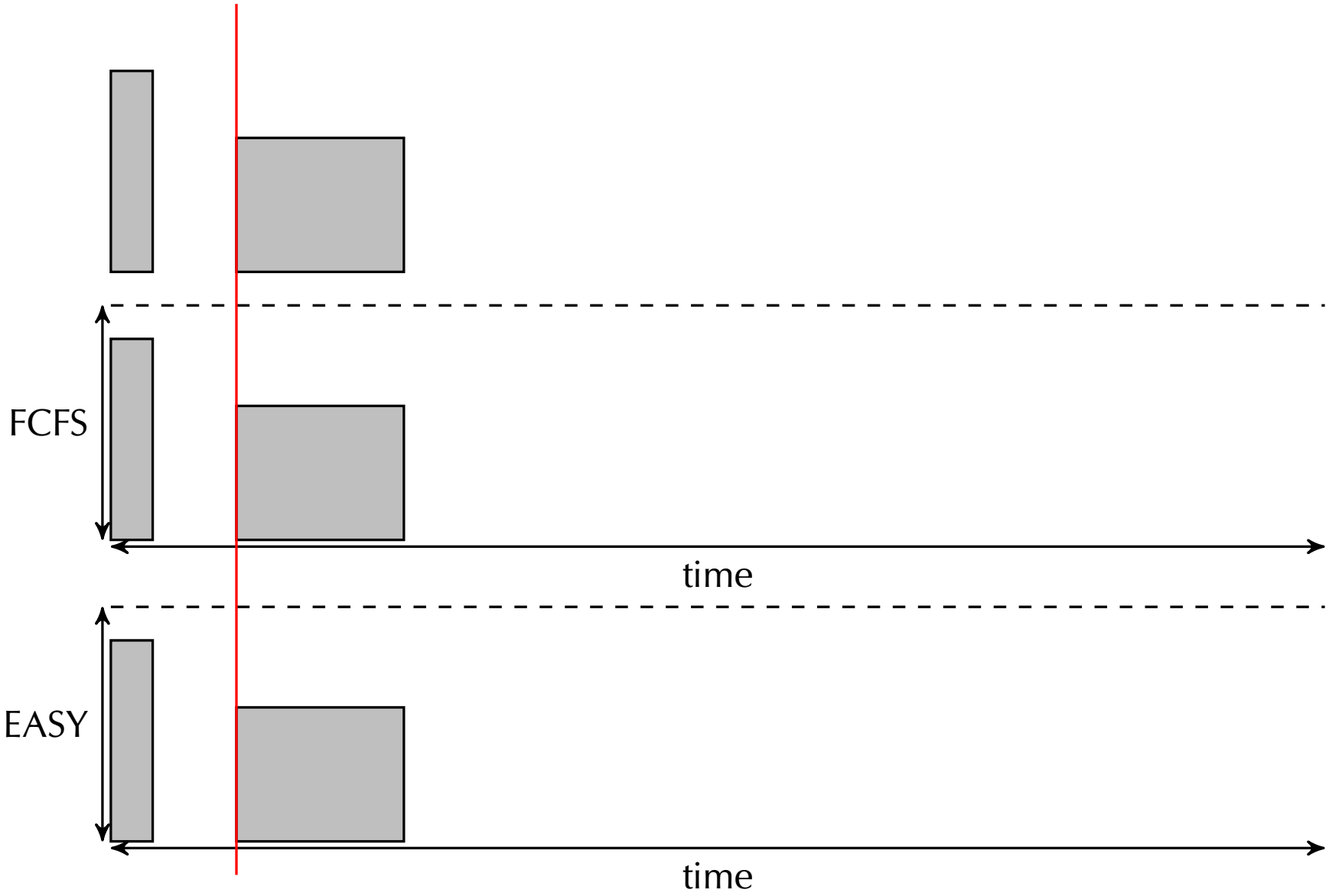
Figure 2.1 Backfill Algorithm Pseudo-code

image source: D. A. Lifka. "An Extensible Job Scheduling System for Massively Parallel Processor Architectures". AAI9833045. PhD thesis. Chicago, IL, USA, 1998. ISBN: 0-591-86149-6

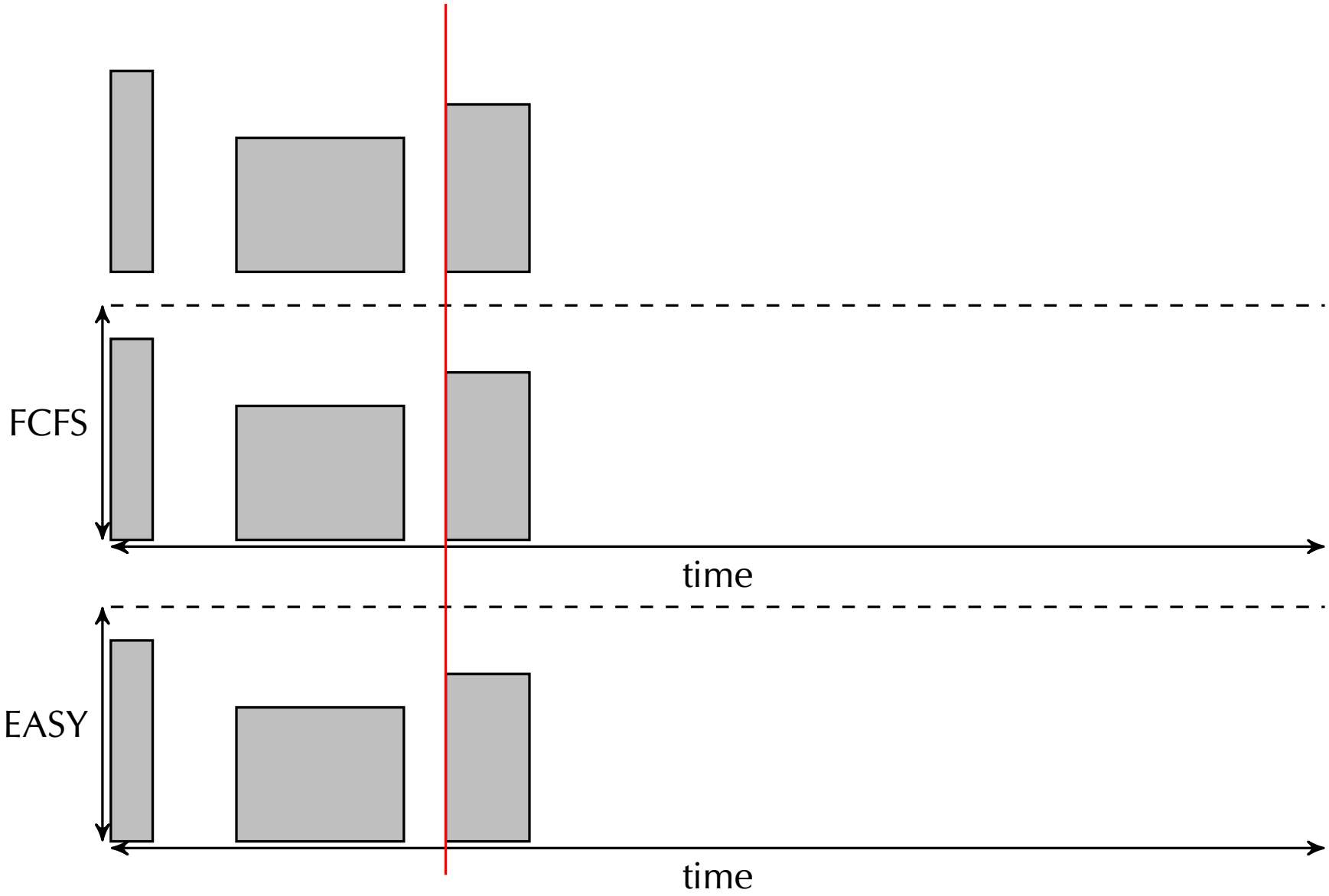
# EASY vs. FCFS (First Come First Served)



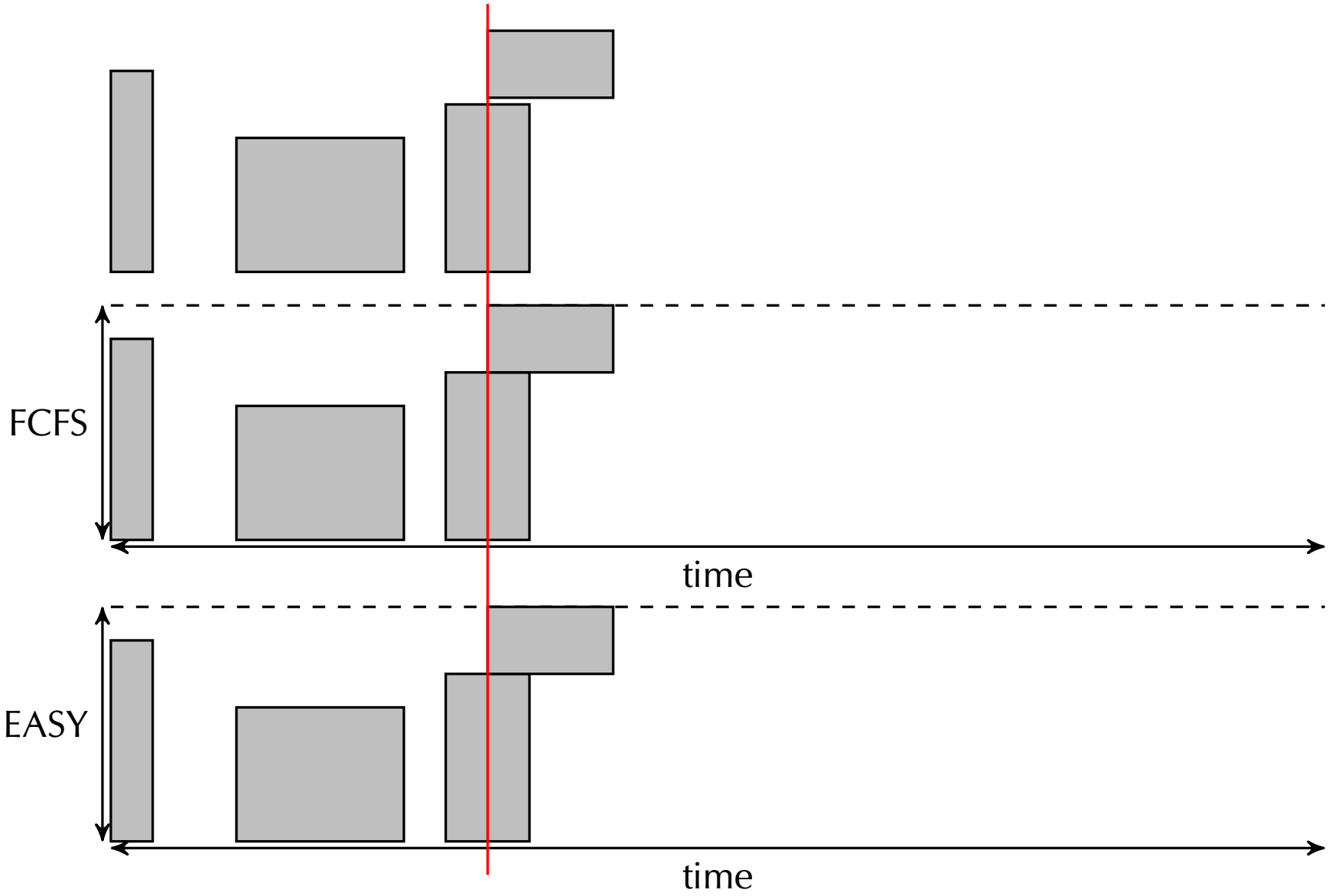
# EASY vs. FCFS (First Come First Served)



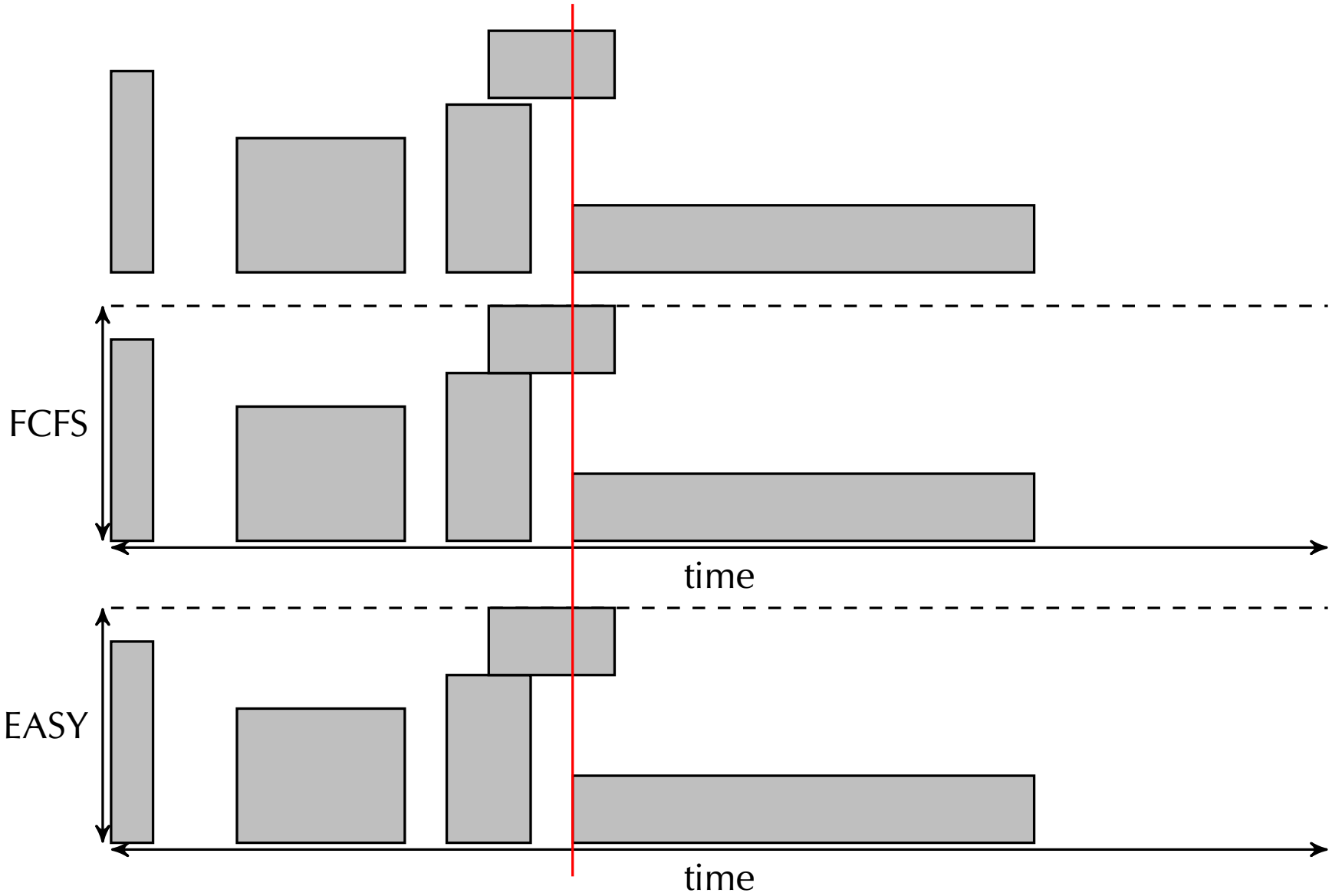
# EASY vs. FCFS (First Come First Served)



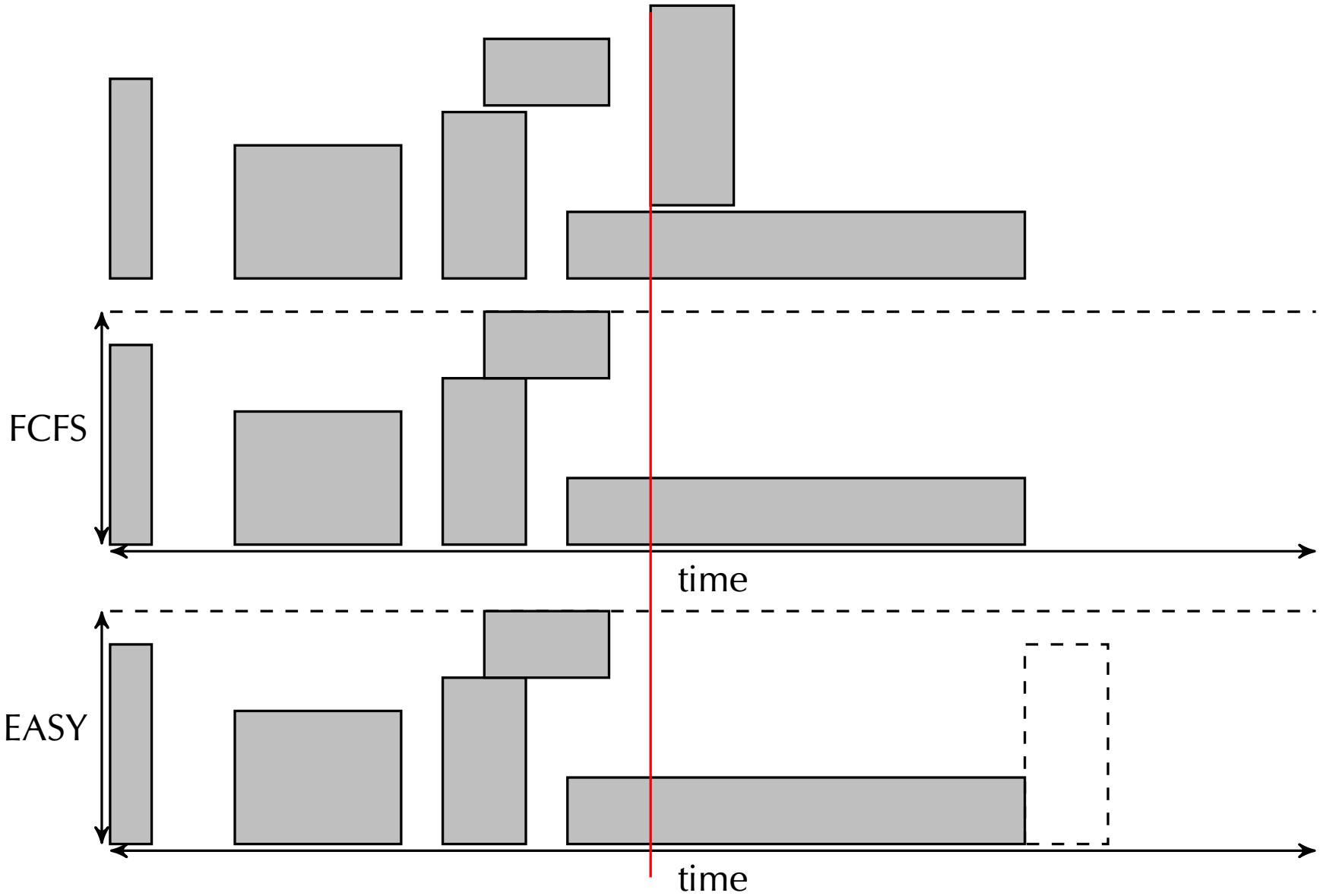
# EASY vs. FCFS (First Come First Served)



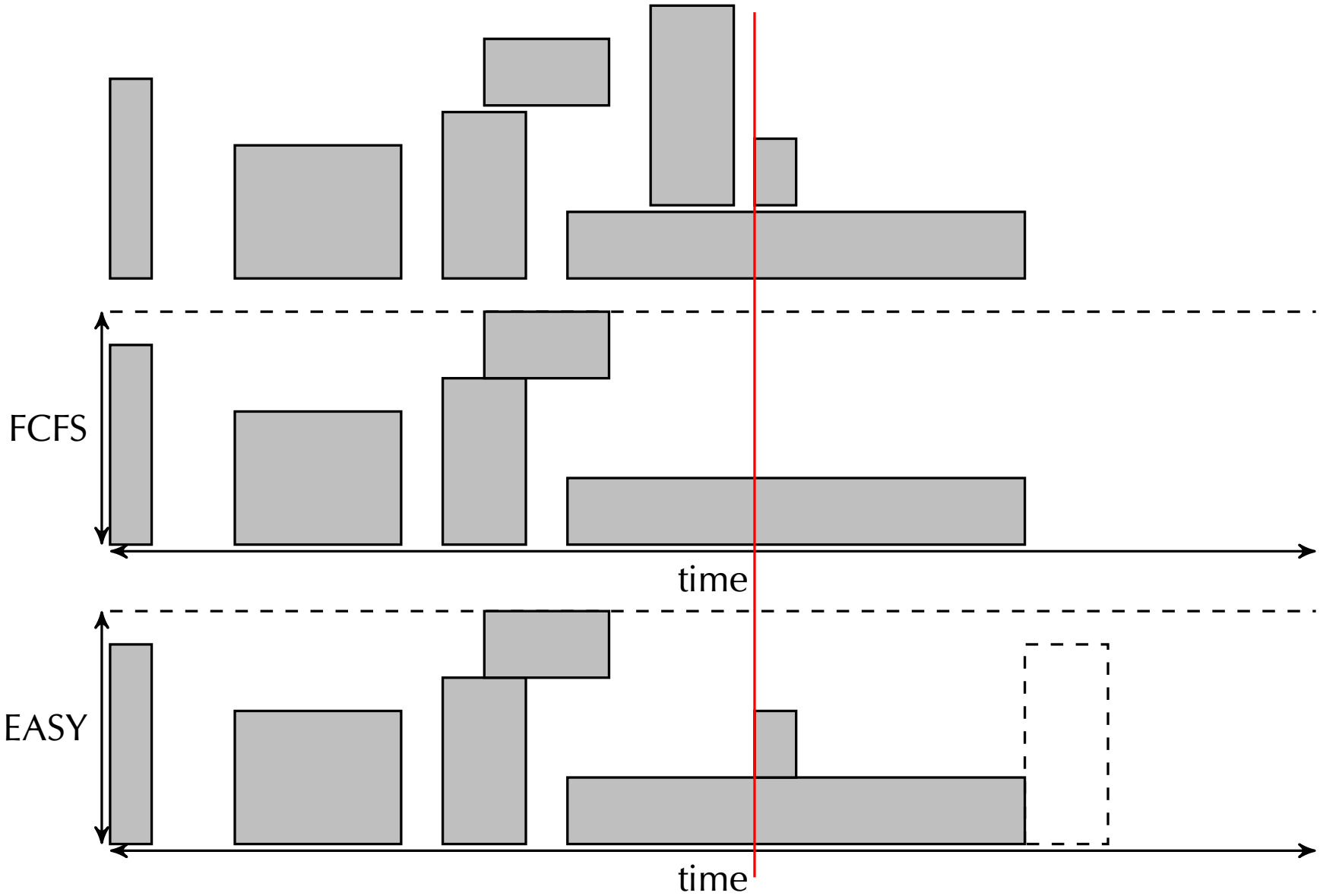
# EASY vs. FCFS (First Come First Served)



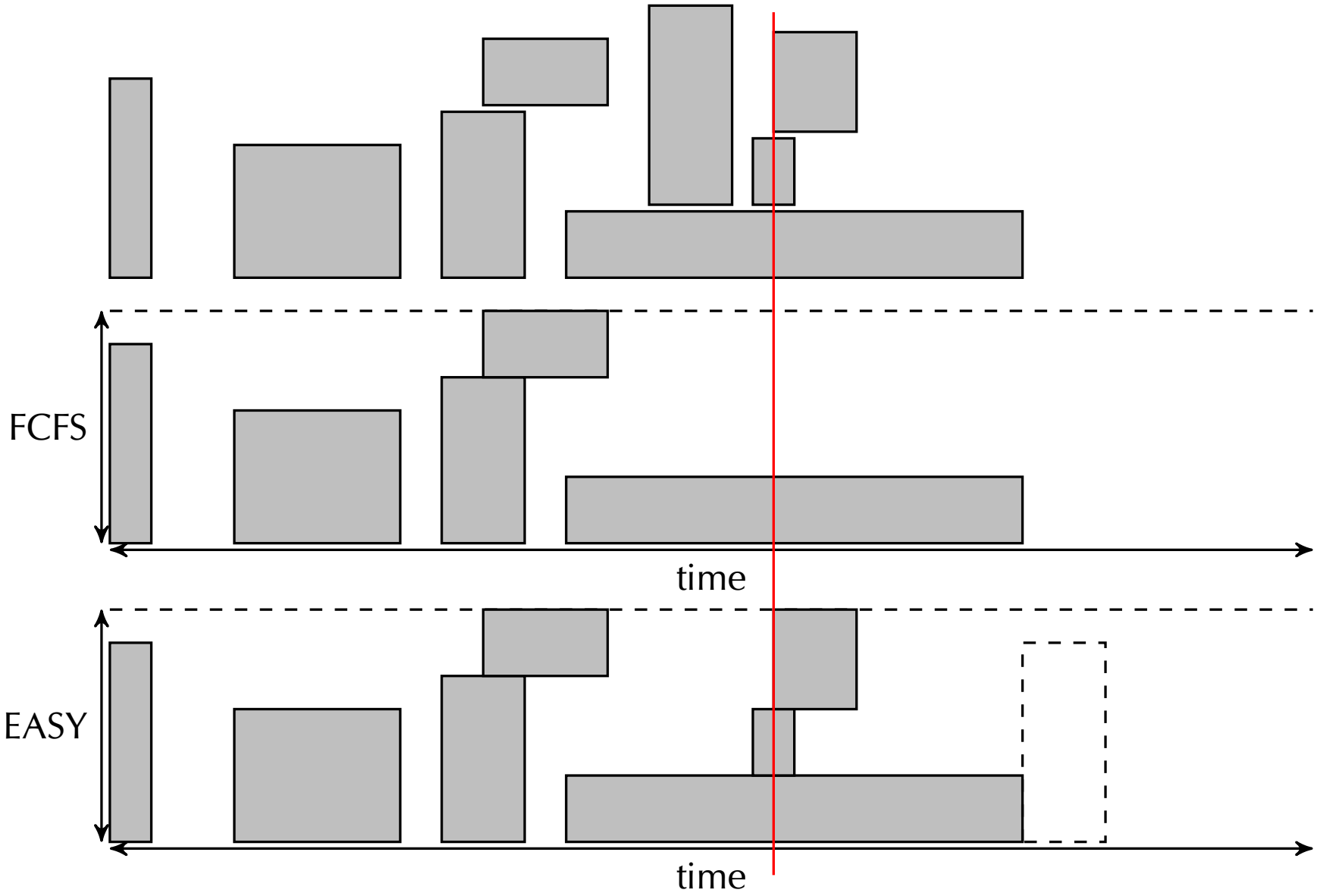
# EASY vs. FCFS (First Come First Served)



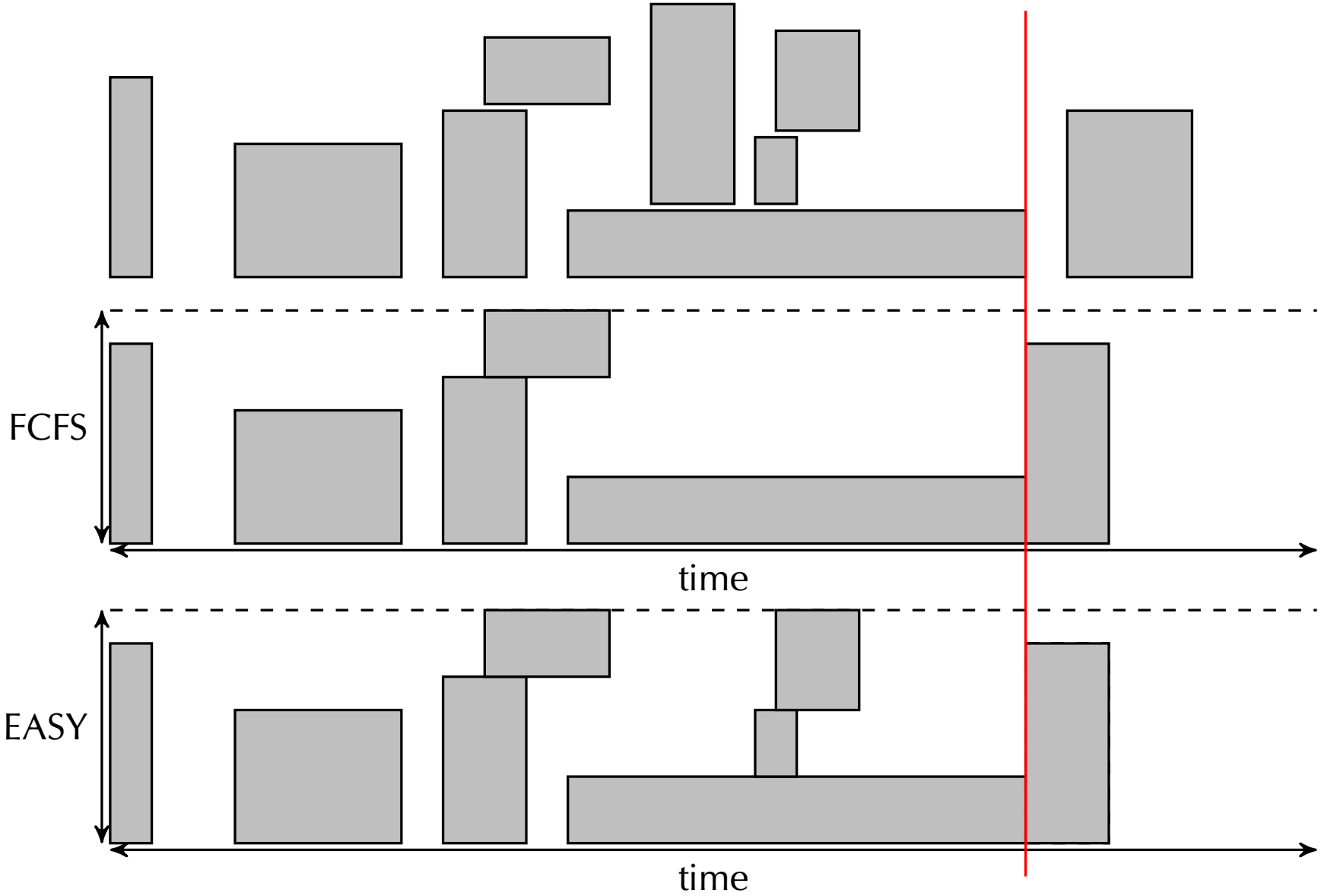
# EASY vs. FCFS (First Come First Served)



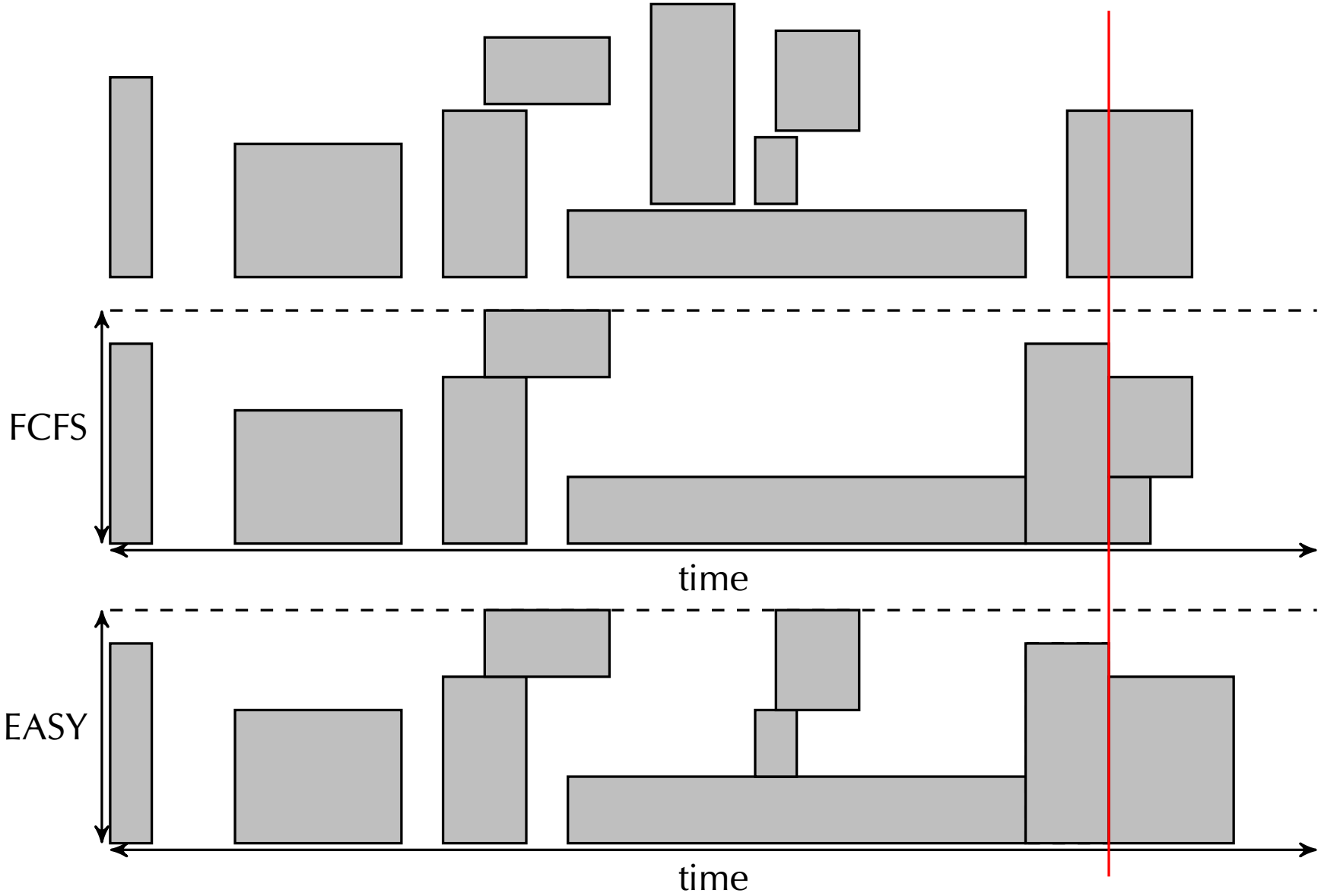
# EASY vs. FCFS (First Come First Served)



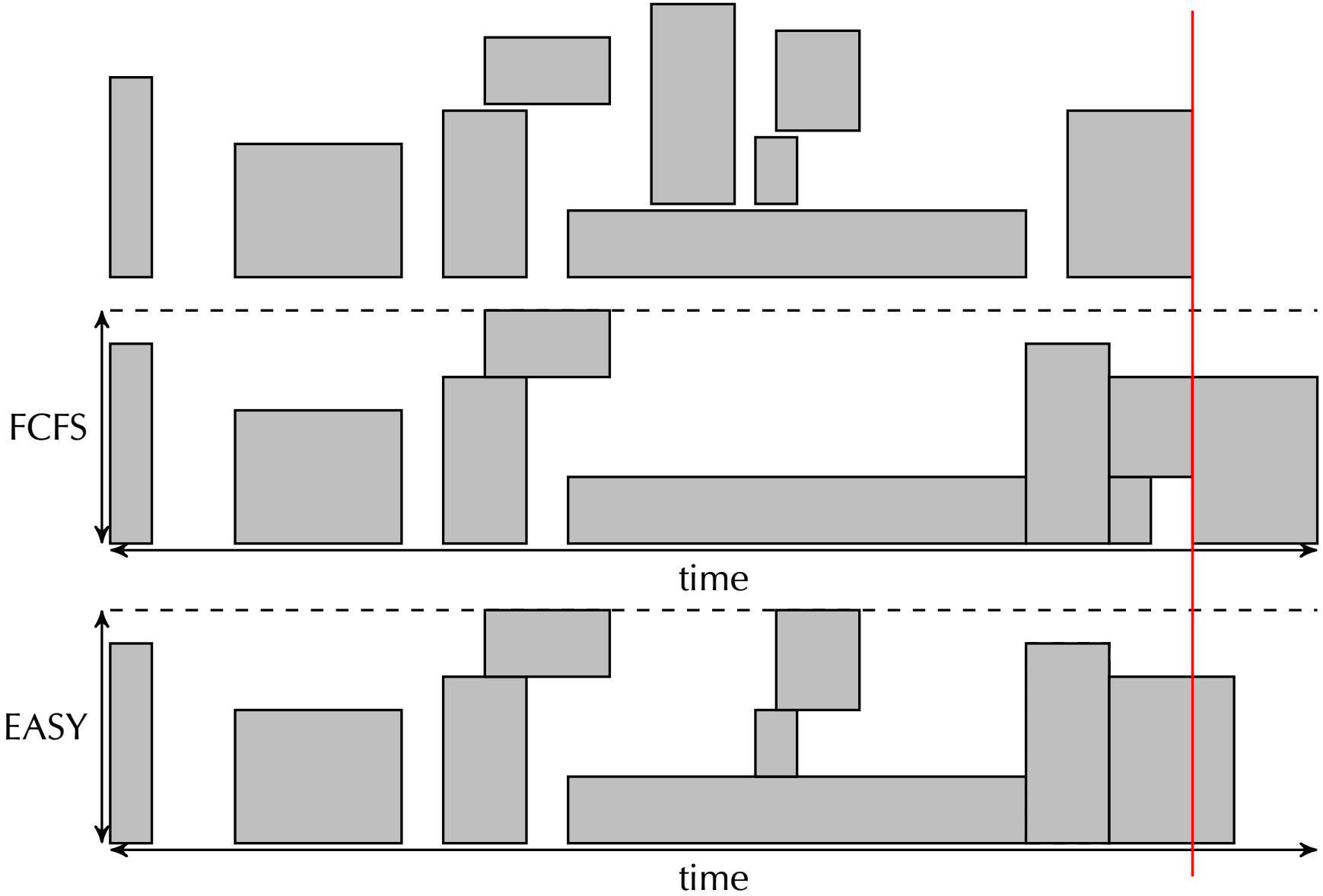
# EASY vs. FCFS (First Come First Served)



# EASY vs. FCFS (First Come First Served)

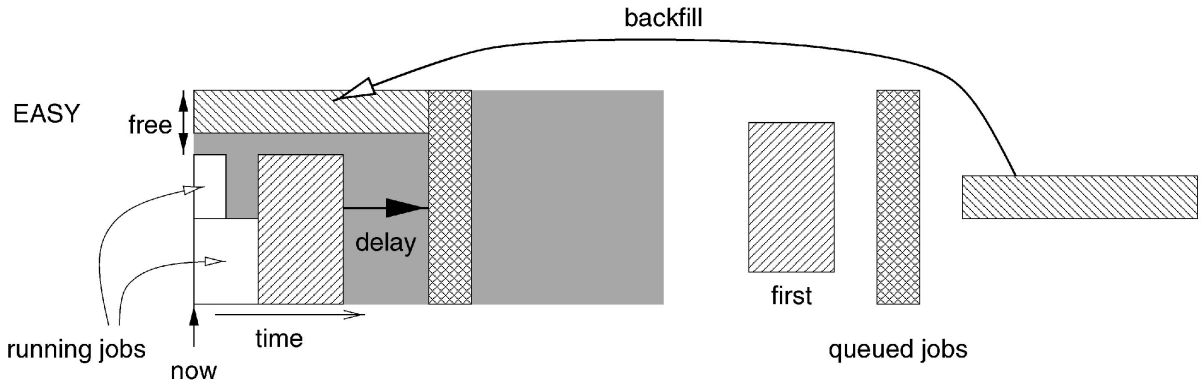


# EASY vs. FCFS (First Come First Served)



# EASY Properties[2]

- Queued jobs may suffer an unbounded delay.
  - second job in the queue may be delayed, for example, by length of backfilled job (unbounded in theory)




- There is no starvation.
  - queuing delay for the job at the head of the queue depends only on jobs that are already running
  - backfilled jobs will not delay first job in queue
  - guaranteed that first jobs is eventually run

[2] A. W. Mu'alem and D. G. Feitelson. "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling". In: *IEEE Trans. Parallel Distrib. Syst.* 12.6 (2001), pp. 529–543. DOI: 10.1109/71.932708.

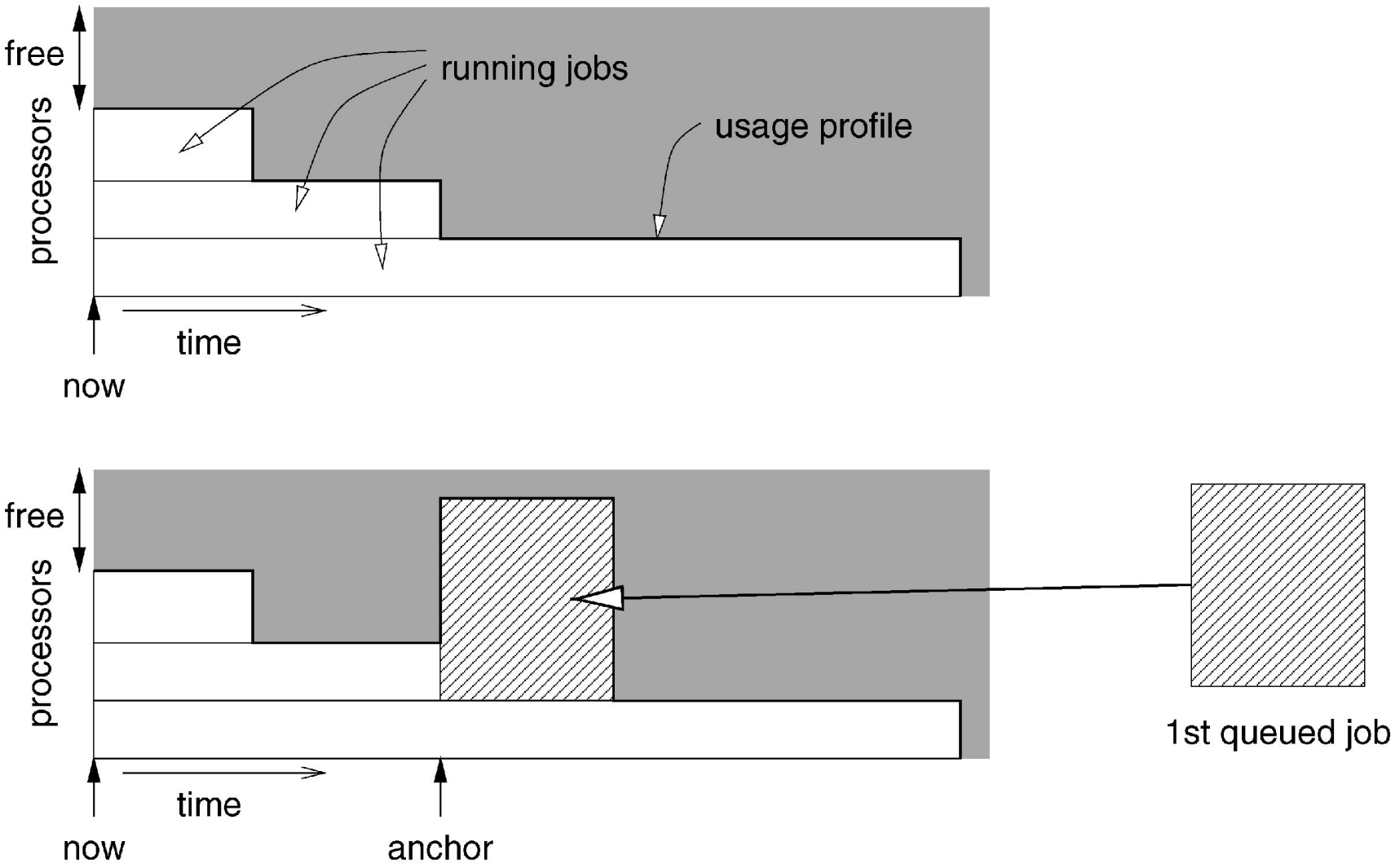
# Conservative Backfill[3]

## Algorithm: Conservative Backfill

1. Find anchor point
   a) Scan the profile and find the first point where enough processors are available to run this job. This is called the anchor point.
   b) Starting from this point, continue scanning the profile to ascertain that the processors remain available until the job's expected termination.
   c) If not, return to (a) and continue the scan to find the next possible anchor point.
2. Update the profile to reflect the allocation of processors to this job, starting from its anchor point.
3. If the job's anchor is the current time, start job immediately.

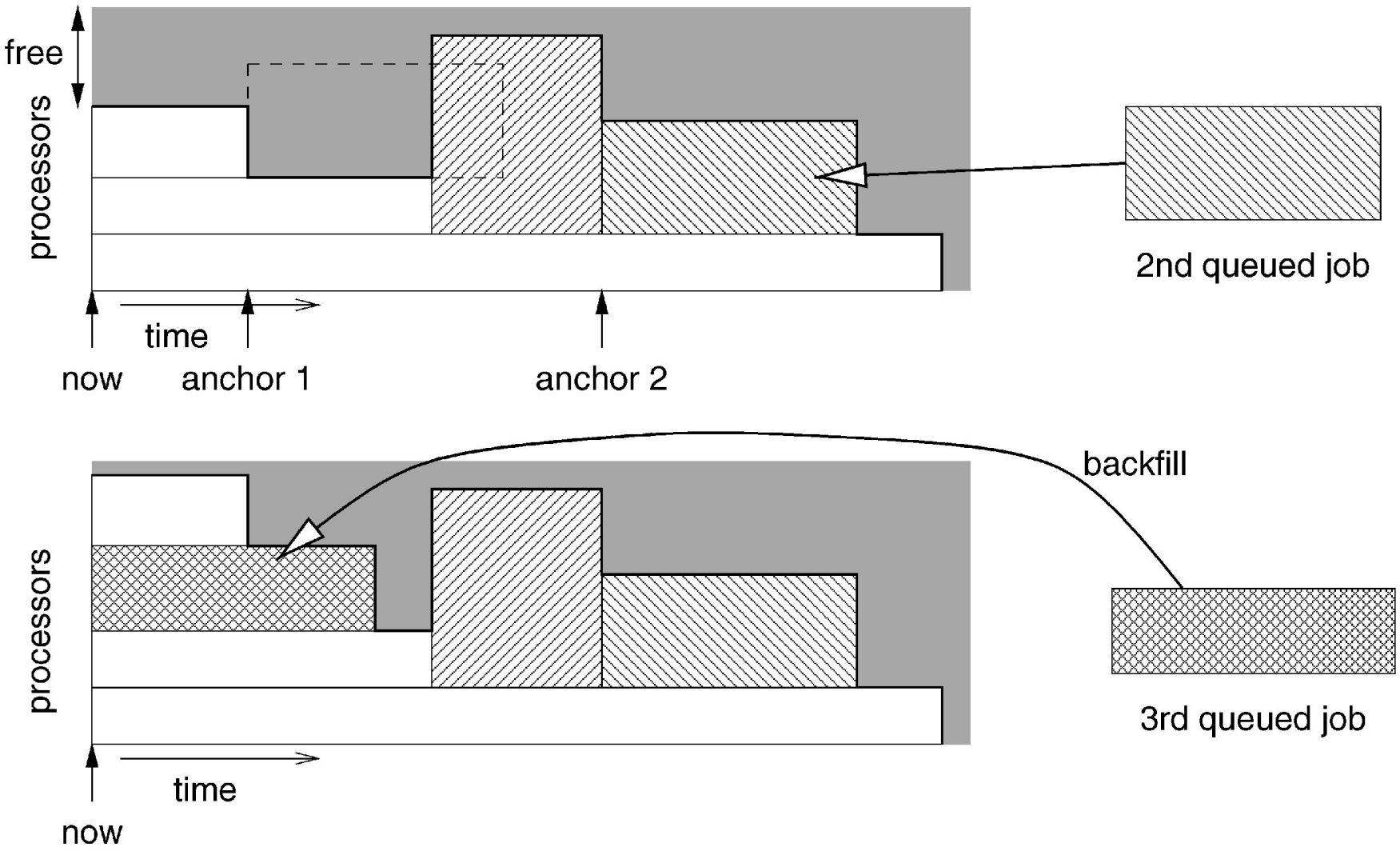
[3] A. W. Mu'alem and D. G. Feitelson. "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling". In: *IEEE Trans. Parallel Distrib. Syst.* 12.6 (2001), pp. 529–543. DOI: 10.1109/71.932708.

# Conservative Backfill Example I




source: [4]

# Conservative Backfill Example II

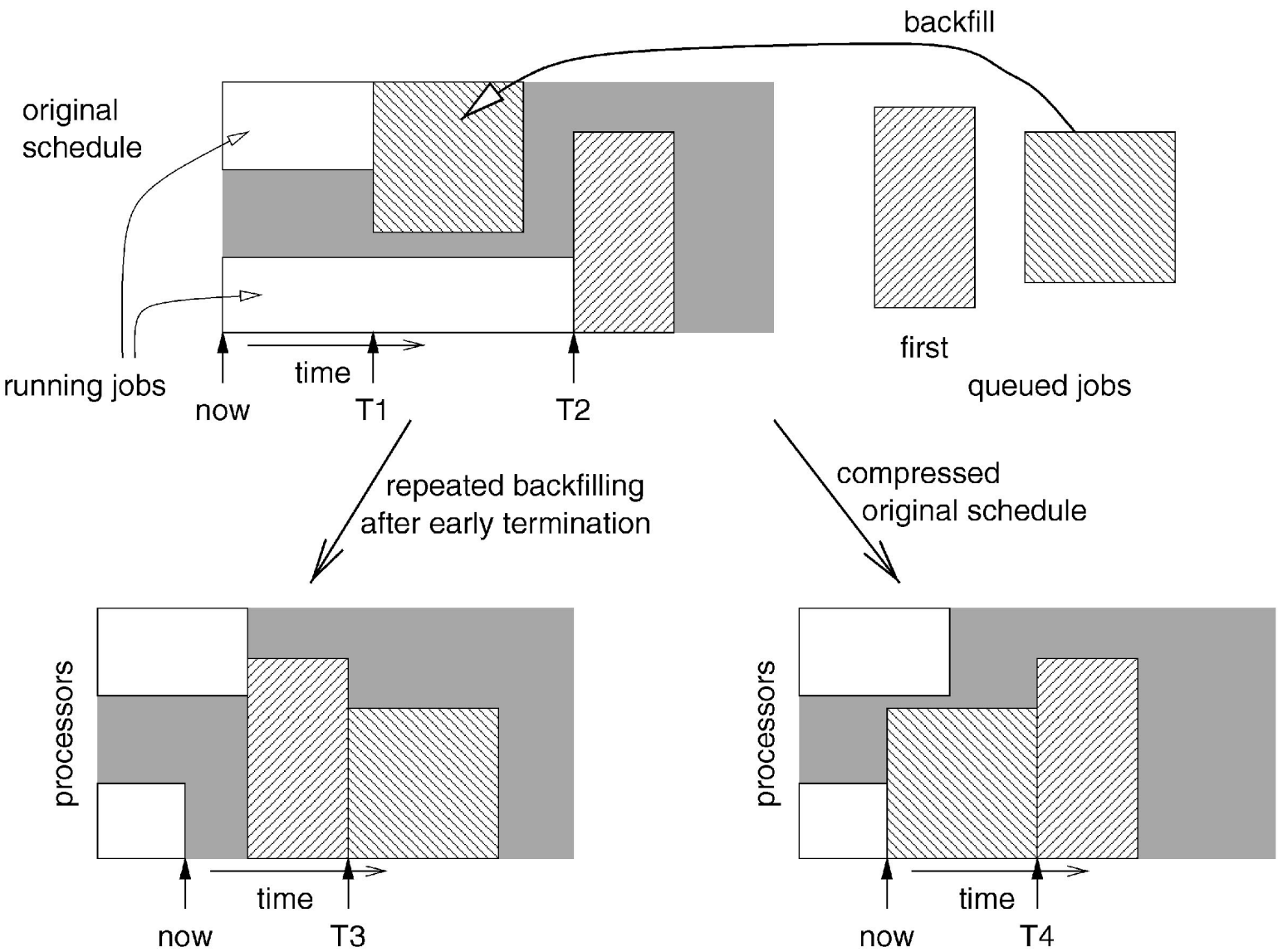



source: [4]

# Conservative Backfill

- algorithm moves jobs forward only if they do not delay any previously queued job
- What happens when jobs finish earlier than expected?
  1. Option 1: new round of backfilling with different data about job runtimes
     - BUT: the same job may not be backfilled and will therefore run much later than the guaranteed time
  2. Option 2: compress the existing schedule (taken)
     - remove all jobs from profile
     - reinsert them at earliest possible time
     - cannot delay jobs, as in the worst case, they get inserted where they were

# Repeated Conservative Backfilling



# Performance of Backfilling Methods I

- flow time: $f_j = c_j - r_j$
  - also called response time, stretch, turn-around time
  - $c_j$ completion time, $s_j$ start time, $r_j$ release time
- bounded slowdown for
  - running time $T_{r_j} = c_j - s_j$ and
  - waiting time $T_{w_j} = s_j - r_j$
  - $T_{r_j} + T_{w_j} = c_j - s_j + s_j - r_j = c_j - r_j$

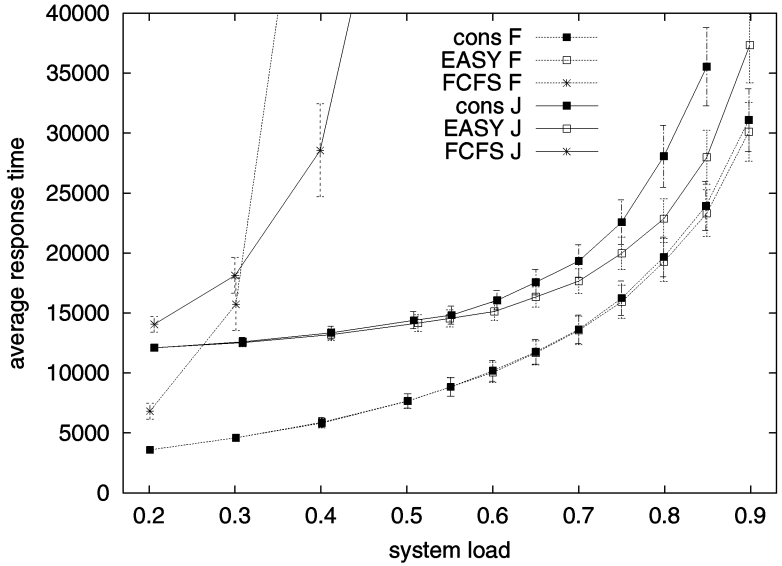
$$b\_sld = \begin{cases} \frac{T_w + T_r}{T_r} & \text{if } T_r > 10s, \\ \frac{T_w + T_r}{10s} & \text{otherwise} \end{cases}$$

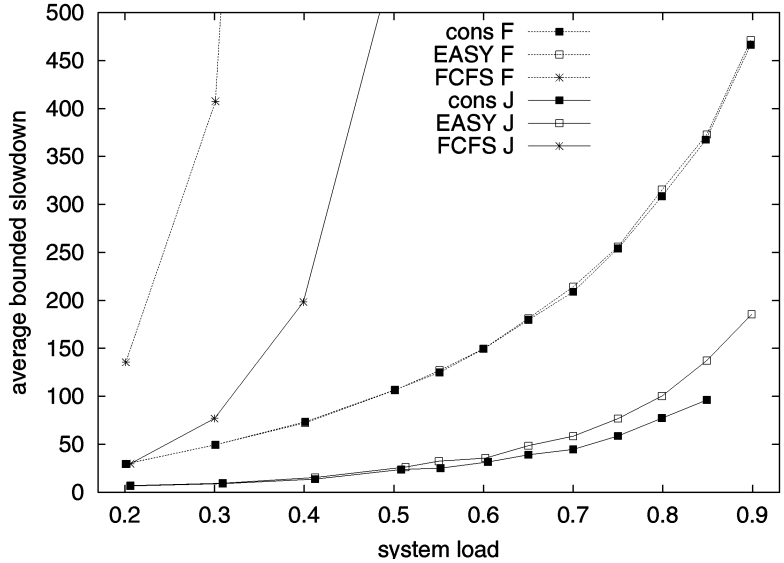
# Performance of Backfilling Methods II

Workload models

- Feitelson (F): a general model based on data from six different traces, including CTC (Cornell Theory Center, IBM SP2) and Par (San Diego Supercomputing Center, Intel Paragon)
  - 350,000 jobs
- Jann (J): a model developed specifically for the CTC trace
  - 100,000 jobs

# Performance of Backfilling Methods III



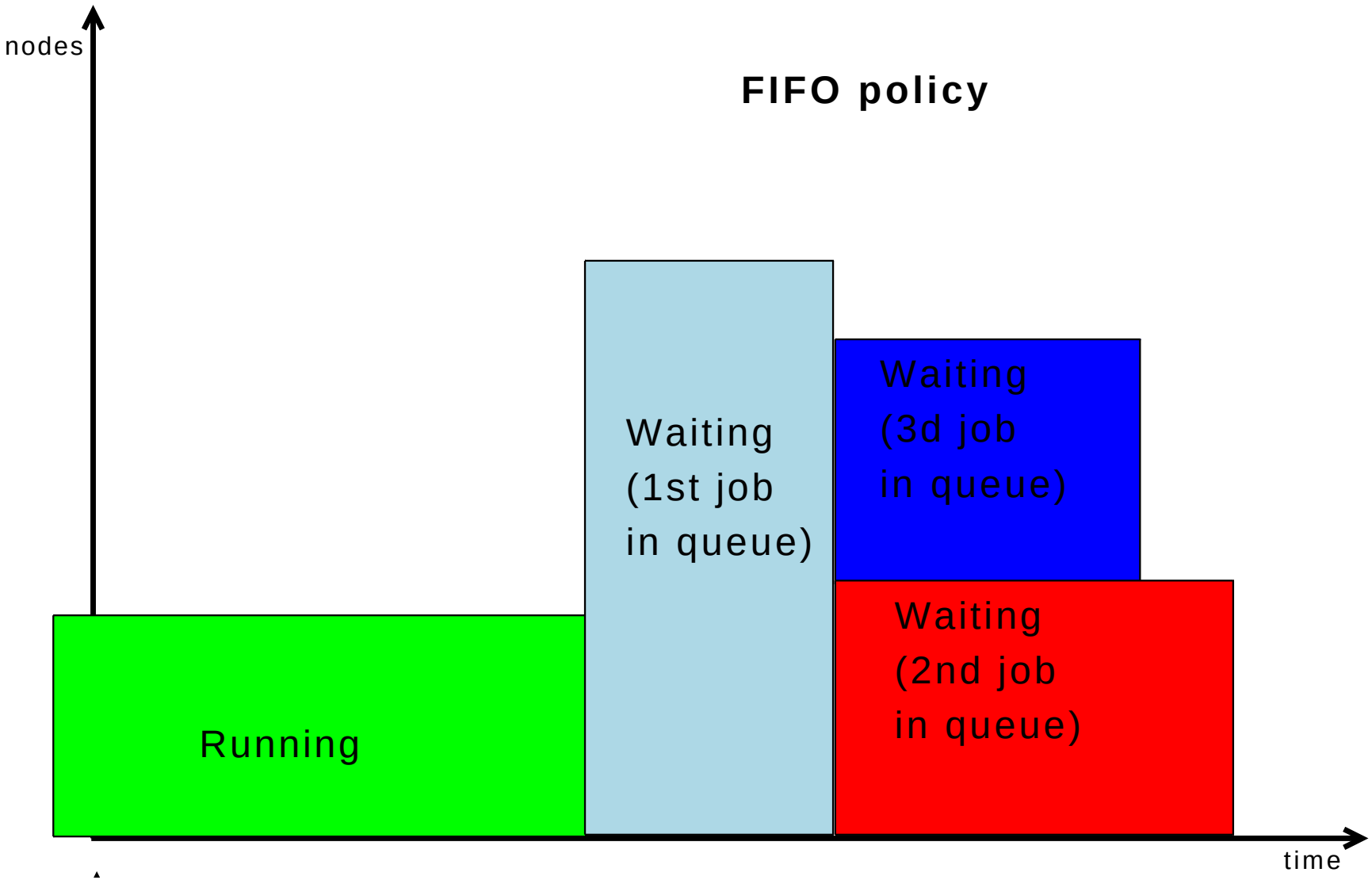

- results for different workload models
  - why not using a logarithmic y axis?
  - what happened with cons J for system load of 0.9?

# Common Policies for Batch Scheduling

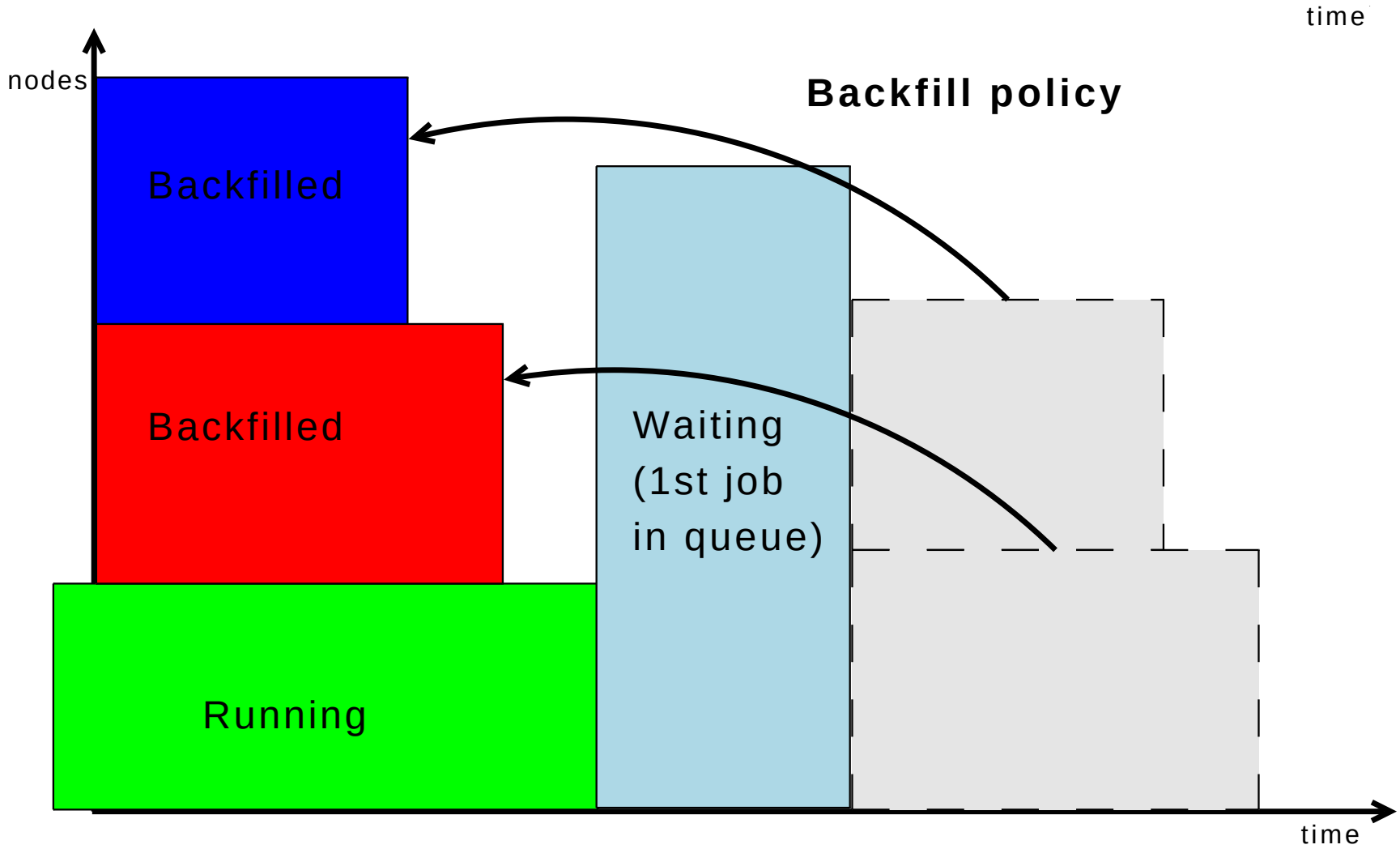
# Scheduling Policies

- FIFO / FCFS
- Backfill
- Gang Scheduling
  - "Slurm supports timesliced gang scheduling in which two or more jobs are allocated to the same resources and these jobs are alternately suspended to let one job at a time have dedicated access to the resources for a configured period of time."
    from `http://slurm.schedmd.com/gang_scheduling.html`
- TimeSharing
  - concurrent execution of jobs on the same resources
- Fairshare
  - mechanism which allows historical resource utilization information to be incorporated into job feasibility and priority decisions
- Preemption

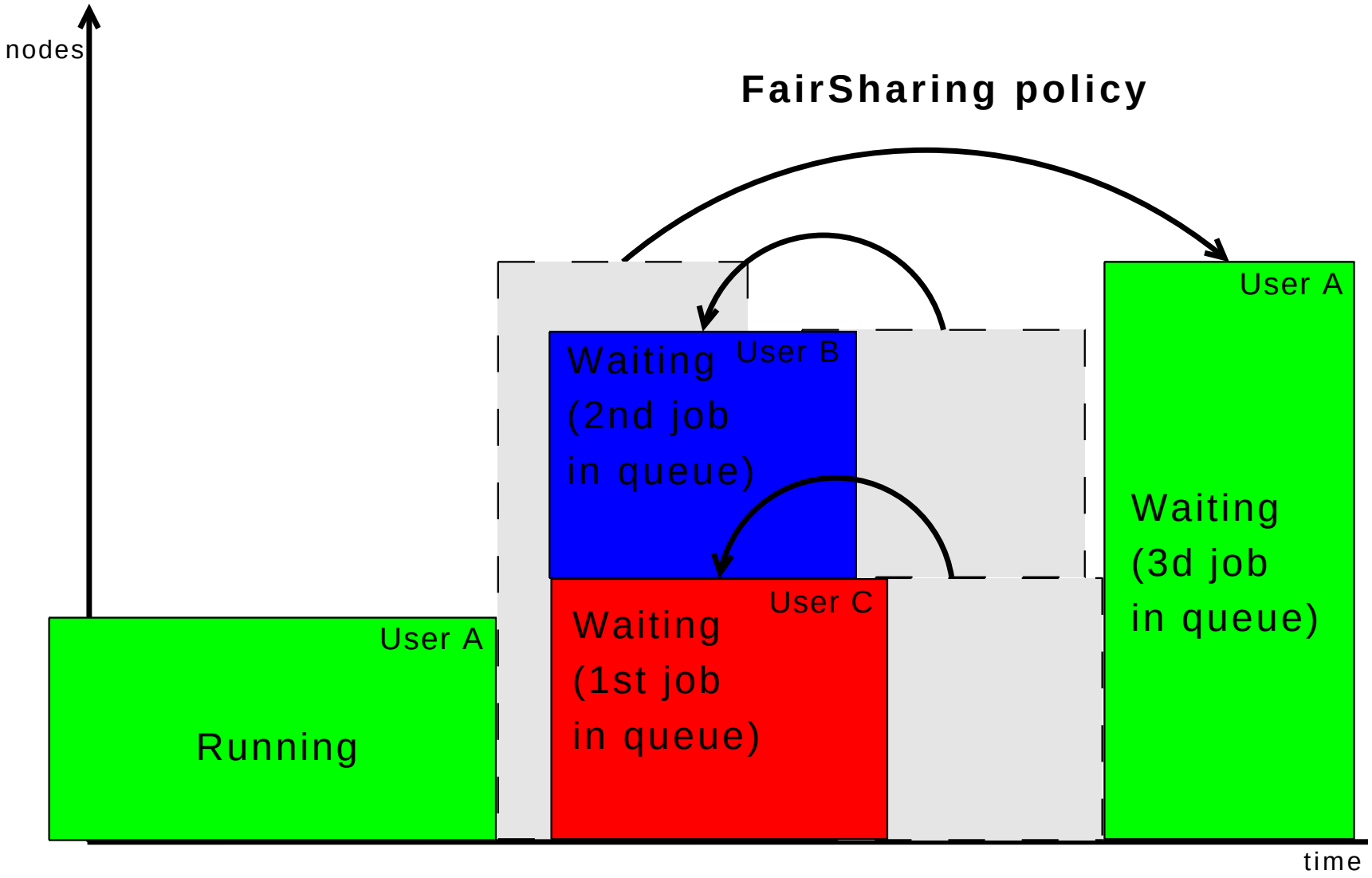
# Example Scheduling Policies [2] I



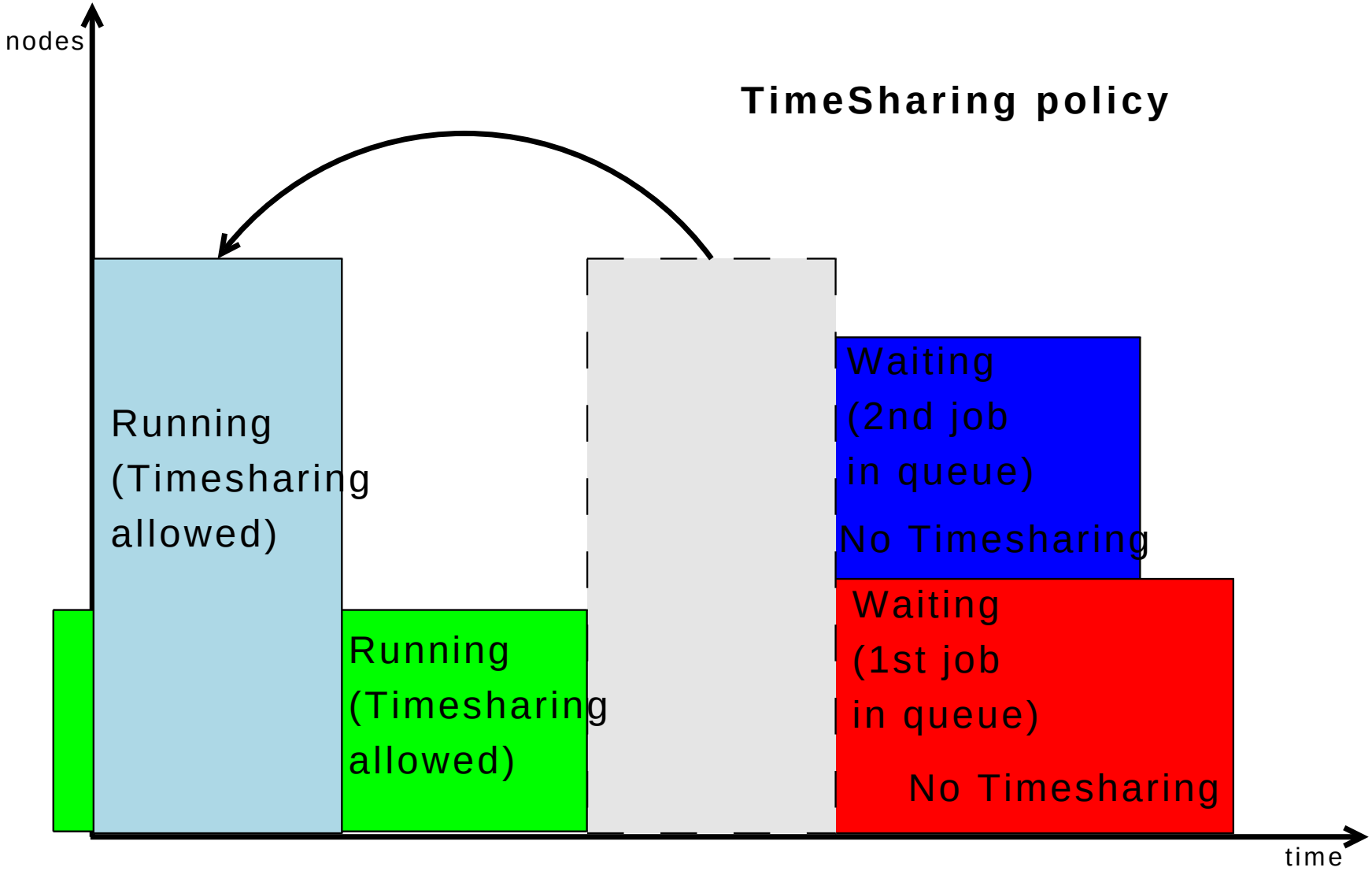
# Example Scheduling Policies [2] II



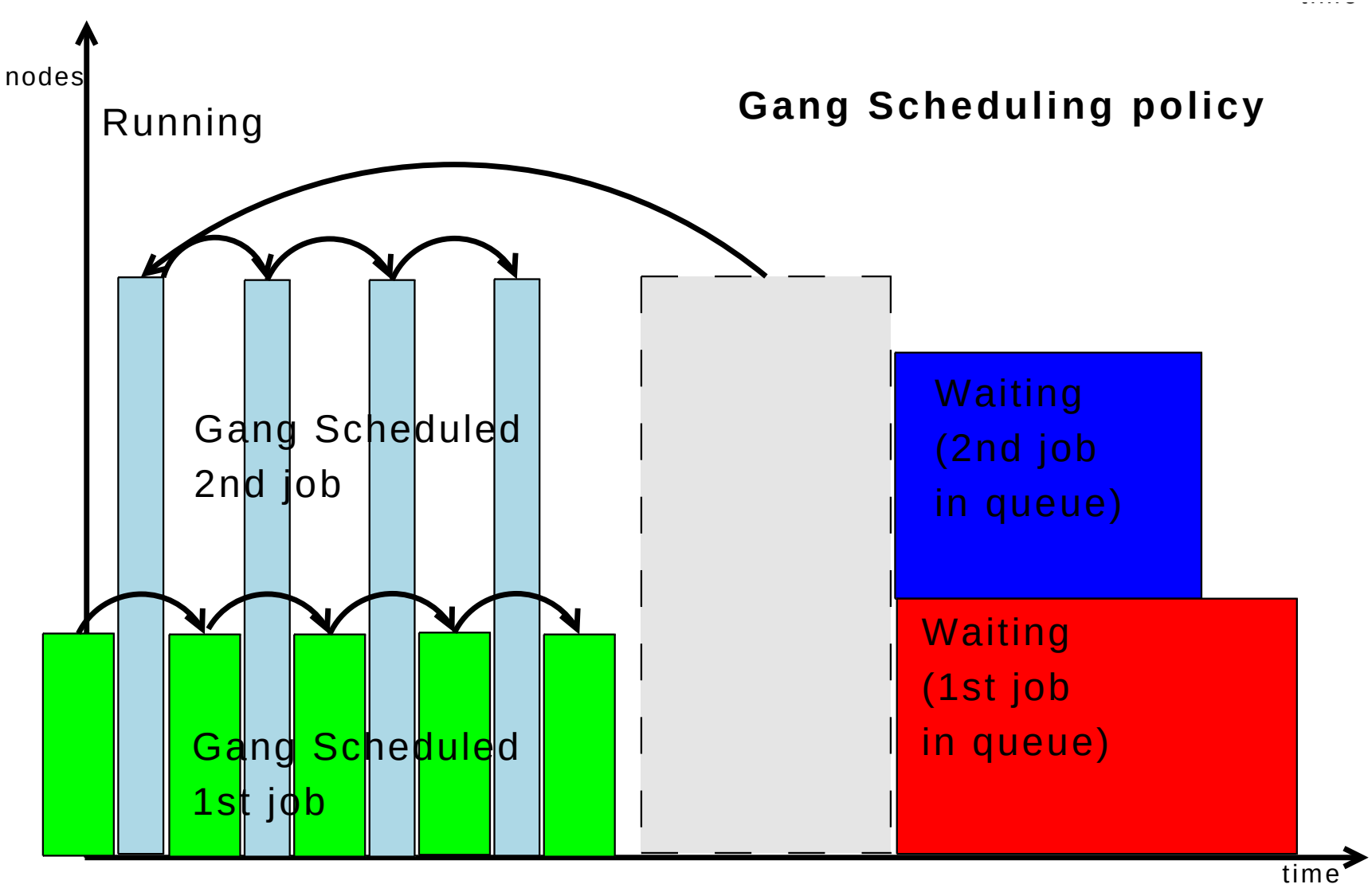
# Example Scheduling Policies [2] III



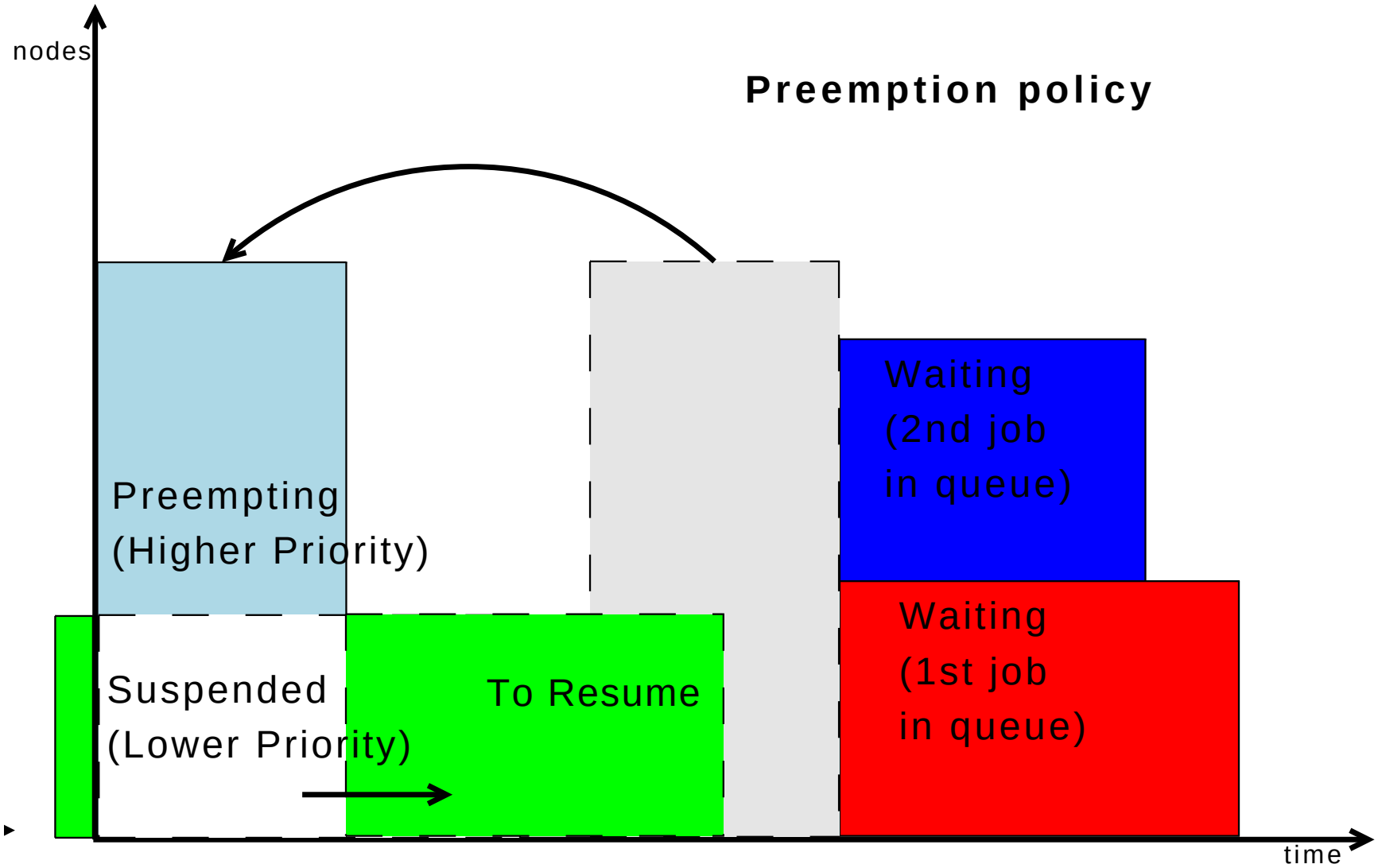
# Example Scheduling Policies [2] IV



# Example Scheduling Policies [2] V



# Example Scheduling Policies [2] VI



# Production Schedulers

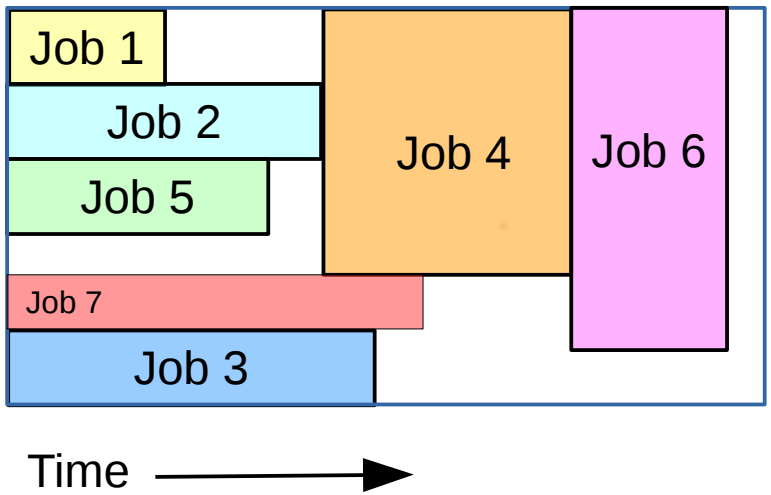
- PBS Professional - Altair
  - workload manager and scheduler
- Torque - Adaptive Computing
  - Torque is the resource manager
  - Maui: cluster scheduler (can also be used with LoadLeveler, LSF, RMS)
- Moab: commercial scheduler from Adaptive Computing
- SLURM
  - Simple Linux Utility for Resource Management
  - free and open source (GPL)
- IBM LSF
- IBM LoadLeveler
- Univa Grid Engine
  - previously Oracle Grid Engine, previously Sun Grid Engine (SGE)

# SLURM

```
#!/bin/bash
#SBATCH -n 4                # Number of cores
#SBATCH -t 0-00:05          # Runtime in D-HH:MM
#SBATCH -p student          # Partition to submit to
#SBATCH -o job_%j.out # File to which STDOUT will be written
#SBATCH -e job_%j.err # File to which STDERR will be written
#SBATCH --mail-type=END     # Type of email notification- BEGIN,END,FAIL,ALL
#SBATCH --mail-user=foo@tuwien.ac.at # Email to which notifications will be sent

hostname
```

# SLURM




1. Jobs 1, 2 and 3 running now
2. Job 4 can not start now, waiting for job 2
3. Job 5 can start now on available resources, but only if it completes no later than job 2 (when job 4 can start)
4. Backfill scheduler goes down queue a configurable depth, determining where and when each pending job will run

source: `http://slurm.schedmd.com/SUG14/sched_tutorial.pdf`

# SLURM

https://github.com/SchedMD/slurm/blob/master/src/plugins/sched/backfill/backfill.c

```
/*****************************************************************************\
 *  backfill.c - simple backfill scheduler plugin.
 *
 *  If a partition does not have root only access and nodes are not shared
 *  then raise the priority of pending jobs if doing so does not adversely
 *  effect the expected initiation of any higher priority job. We do not alter
 *  a job's required or excluded node list, so this is a conservative
 *  algorithm.
 *
 *  For example, consider a cluster "lx[01-08]" with one job executing on
 *  nodes "lx[01-04]". The highest priority pending job requires five nodes
 *  including "lx05". The next highest priority pending job requires any
 *  three nodes. Without explicitly forcing the second job to use nodes
 *  "lx[06-08]", we can't start it without possibly delaying the higher
 *  priority job.
 *****************************************************************************
```

- function `_attempt_backfill`
  - starts at line 1557
  - ends at line 2786

# Moab

from `http://docs.adaptivecomputing.com/mwm/archive/6-0/8.2backfill.php`

By default, Moab reserves only the highest priority job resulting in a liberal and aggressive backfill. This reservation guarantees that backfilled jobs will not delay the highest priority job, although they may delay other jobs.

The parameter RESERVATIONDEPTH controls how conservative or liberal the backfill policy is. This parameter controls how deep down the queue priority reservations will be made. While increasing this parameter improves guarantees that priority jobs will not be bypassed, it reduces the freedom of the scheduler to backfill resulting in somewhat lower system utilization. The significance of the trade-offs should be evaluated on a site by site basis.

# Examples

- Fugaku / RIKEN Japan
  - Fujitsu specific
- SuperMUC-NG, LRZ Munich
  - SLURM
- Tianhe-2A, China
  - SLURM
- Summit, DOE/SC/Oak Ridge National Laboratory
  - IBM Spectrum Load Sharing Facility (LSF)
- Piz Daint, CSCS, CH
  - SLURM
- JUWELS, Juelich
  - SLURM
- VSC4
  - SLURM
- Hydra (that's us)
  - SLURM

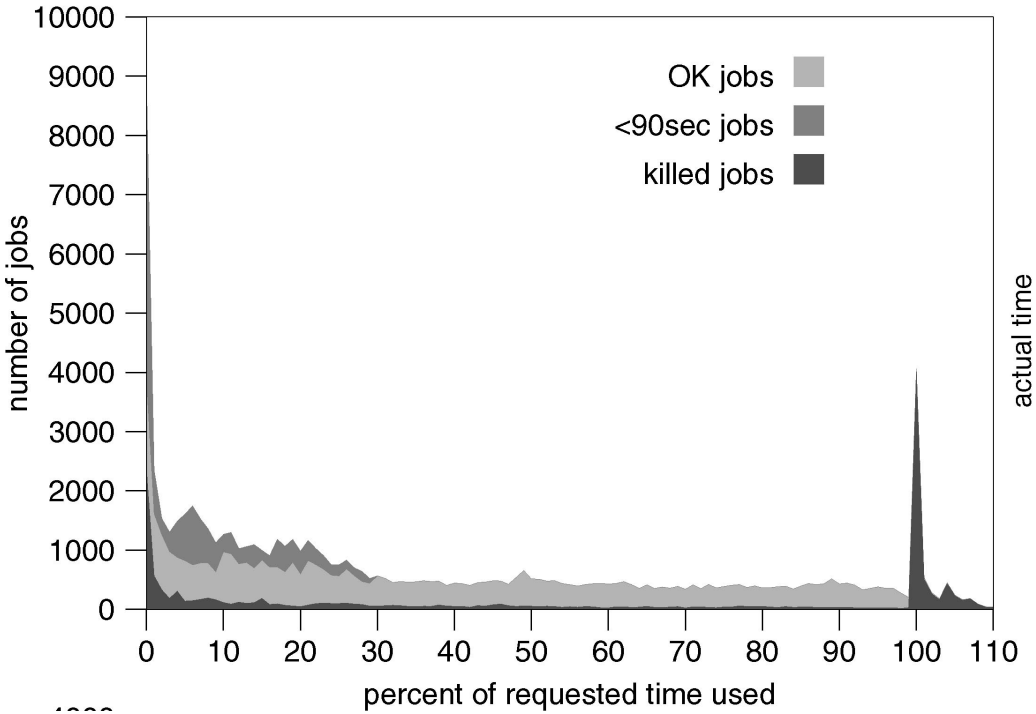
# Workload Traces

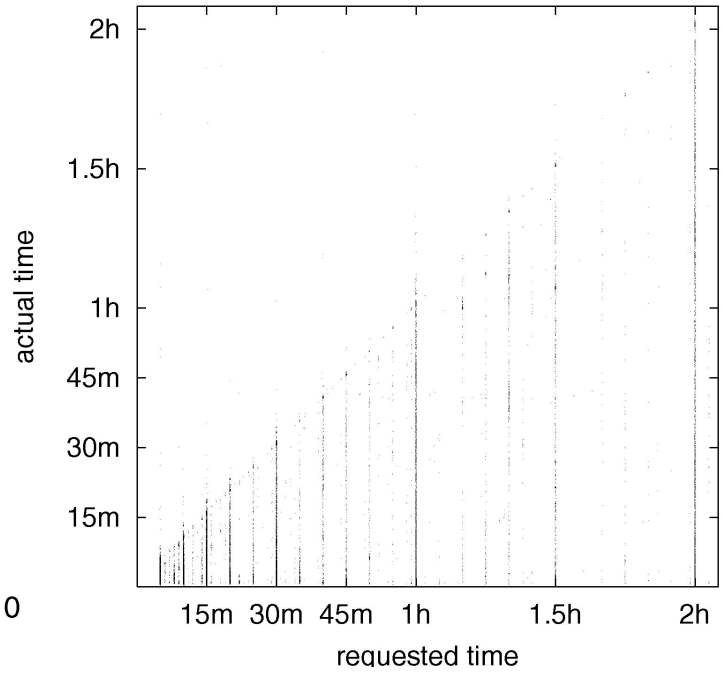
# Analyzing Workload Traces

- workload traces of supercomputers have been published
- several are available in the Parallel Workload Archive (PWA)
  - Dror Feitelson
  - `http://www.cs.huji.ac.il/labs/parallel/workload/`

# Run-time Estimates I

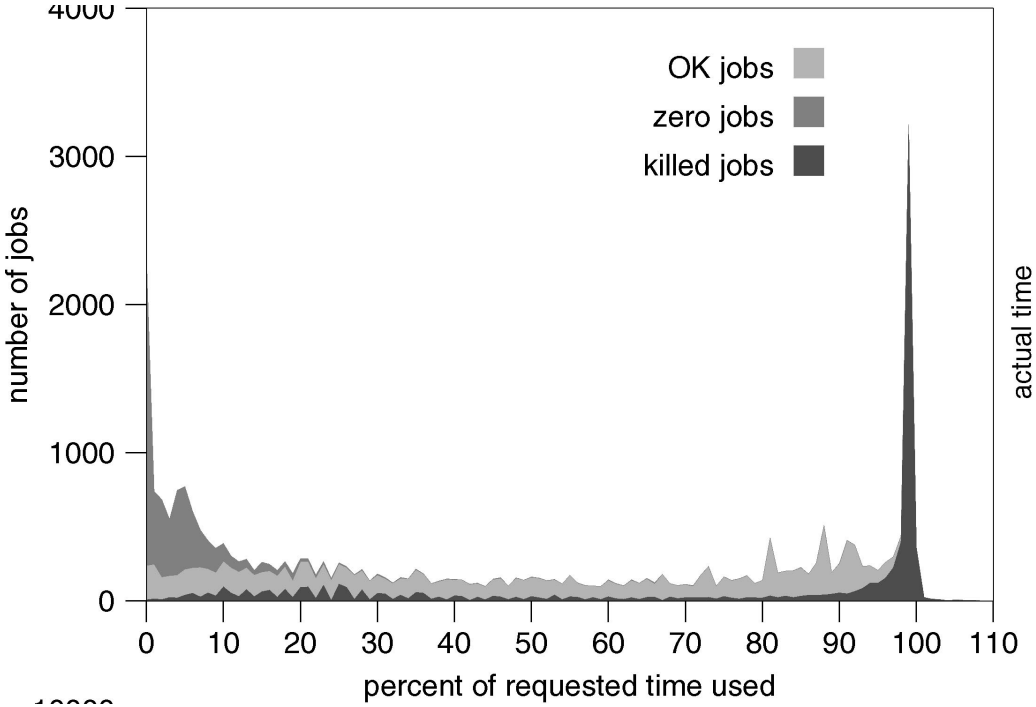
- CTC: the Cornell Theory Center 512-node IBM SP2 (79,296 jobs from July 1996 to May 1997),

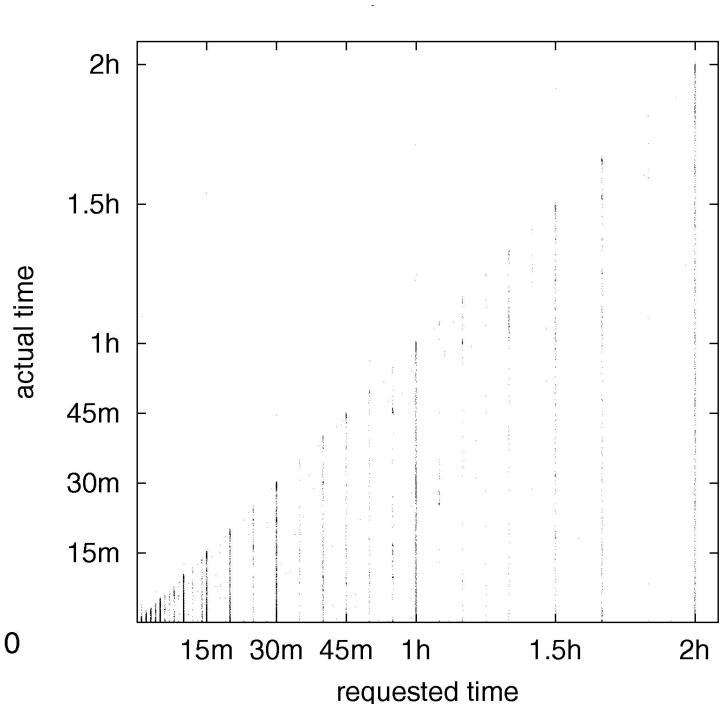


# Run-time Estimates II

- KTH: the Swedish Royal Institute of Technology 100-node IBM SP2 (28,490 jobs from October 1996 to August 1997)



# Further Trace Analyses

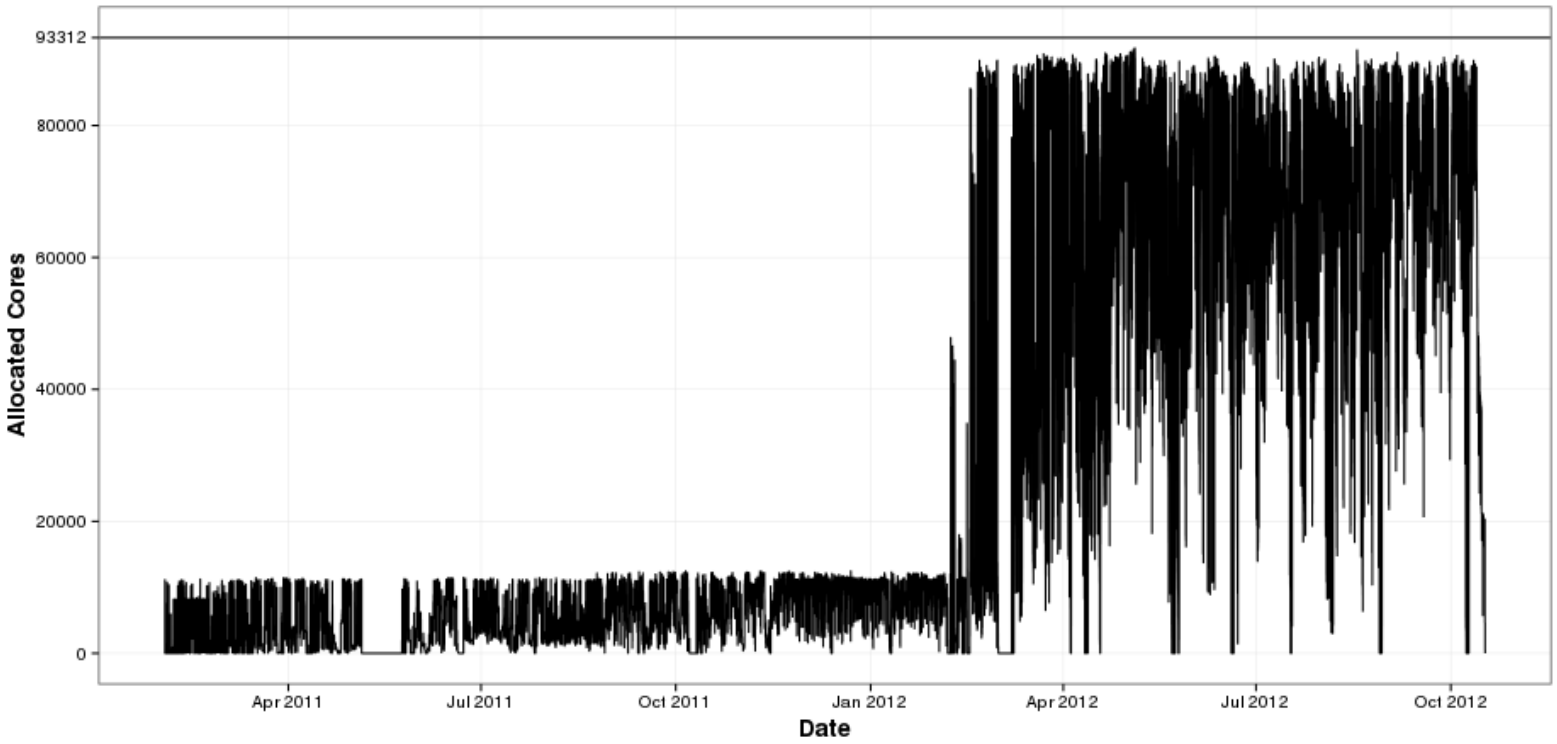
- Further Trace Analyses by Emeras
  - J. Emeras. “Workload Traces Analysis and Replay in Large Scale Distributed Systems”. PhD thesis. University of Grenoble, France, 2013

# Specific Workloads - Curie Trace I

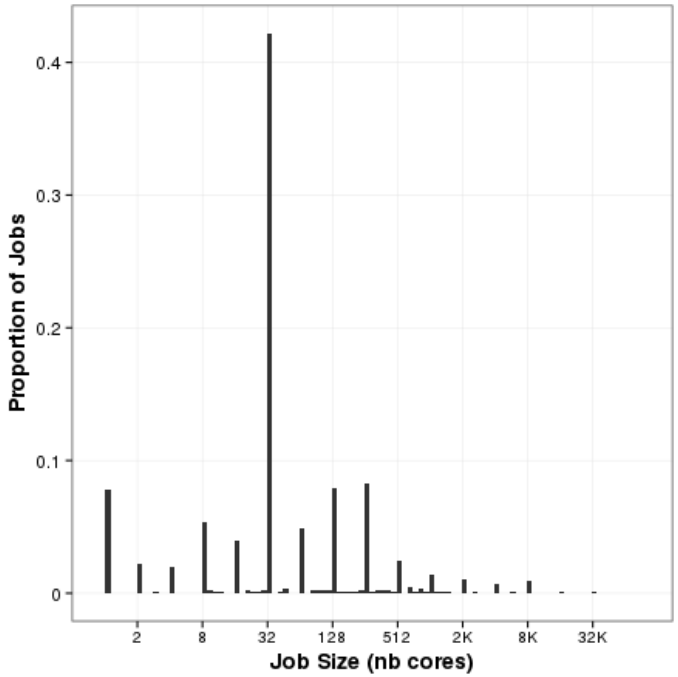
## Curie

- CEA (a French government-funded technological research organization)
- upgraded in February 2012
- fat nodes: 360 S6010 bullx nodes, 4 eight-core Intel Nehalem-EX, total: 11520 cores
- thin nodes: 5,040 bullx B510 nodes, two 8-core Intel Sandy Bridge EP processors, total of 80640 cores
- hybrid nodes: combine Intel processors and Nvidia GPUs
  - 144 blades, each blade has 2 Intel Westmere 2.66 GHz processors and 2 Nvidia M2090 T20A GPUs
- interconnect: InfiniBand QDR full fat tree network
- also see cleaned trace: `http://www.cs.huji.ac.il/labs/parallel/workload/l_cea_curie/index.html`

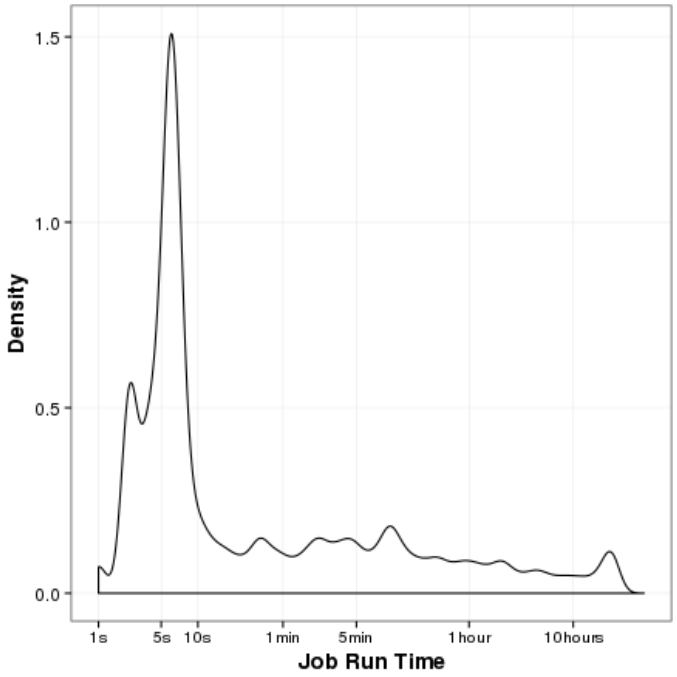
# Specific Workloads - Curie Trace II



# Specific Workloads - Curie Trace III




(a) Distribution of Job Core Allocations for Curie Trace.



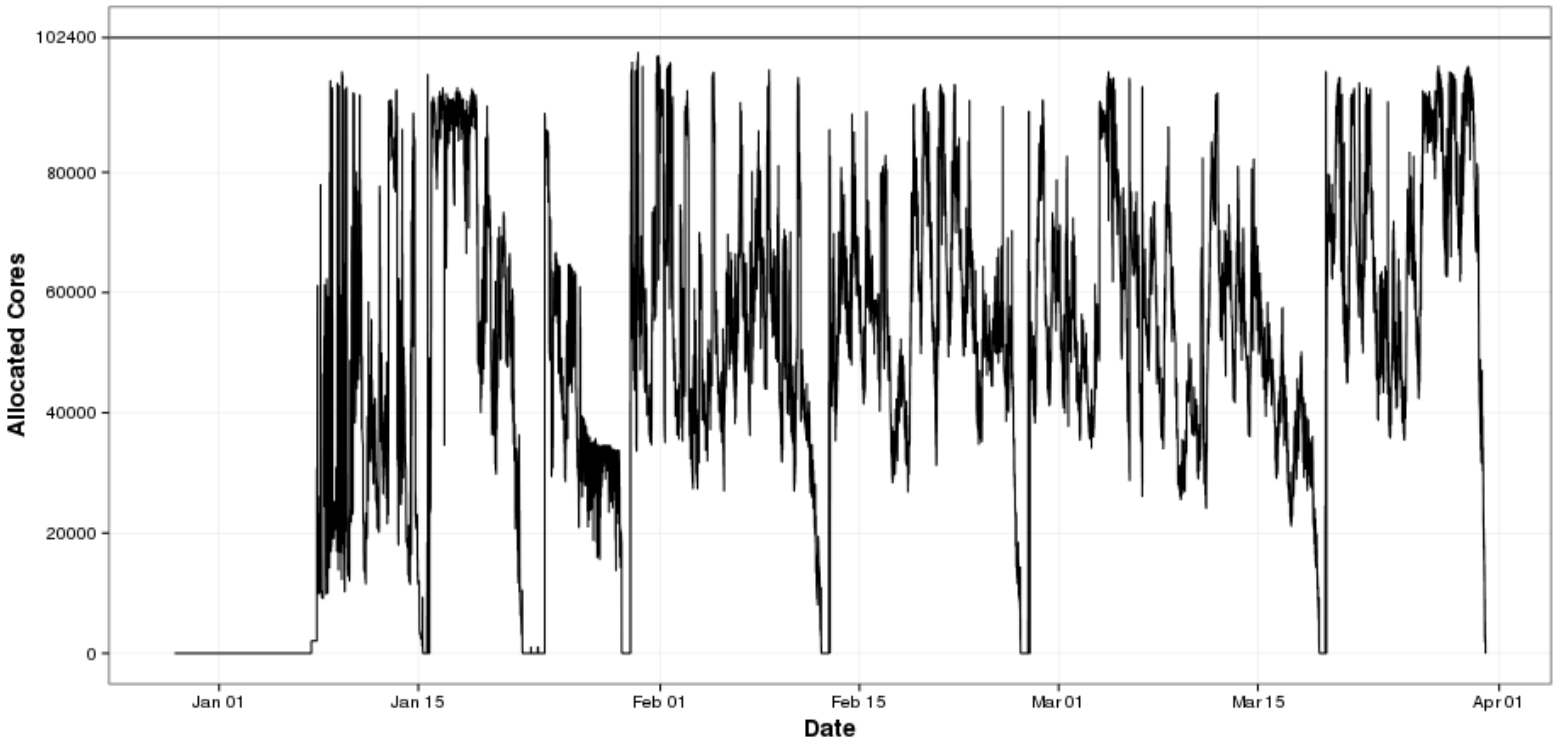

(b) Density Function of Job Run Times for Curie Trace.

# Specific Workloads - Stampede Trace I

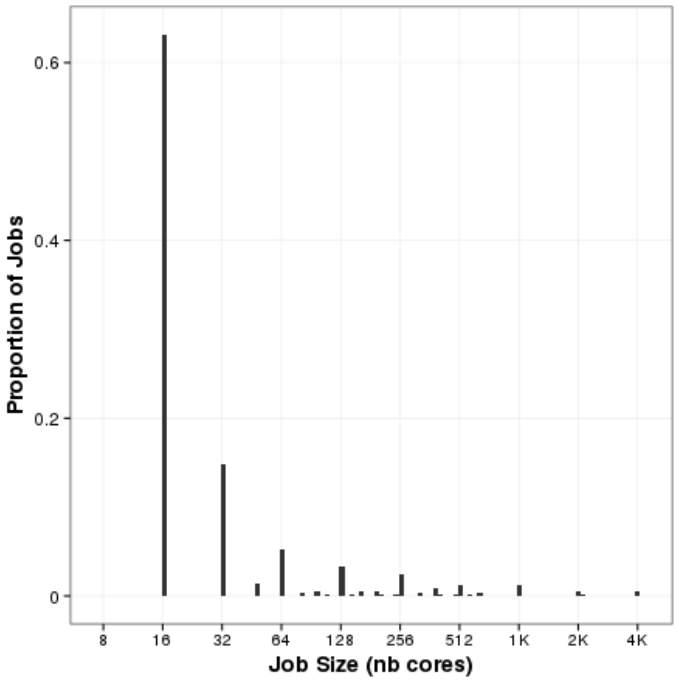
Stampede (as of 2013, time of trace)

- Texas Advanced Computing Center (TACC) at UT Austin
- Dell Linux Cluster based on 6,400 Dell PowerEdge server nodes
- interconnect: Intel Omni Path
- each node
  - 2 8-core Intel Xeon E5 (Sandy Bridge) processors and
  - 1 Intel Xeon Phi SE10P Coprocessor (MIC Architecture)
- roughly: 100k cores
- total of 457,540 jobs in trace

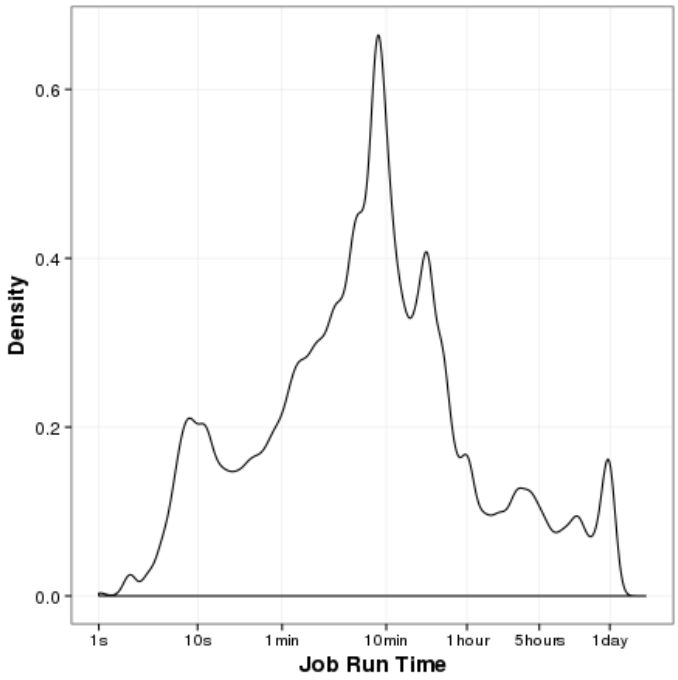
# Specific Workloads - Stampede Trace II



# Specific Workloads - Stampede Trace III




(a) Distribution of Job Core Allocations for Stampede Trace.




(b) Density Function of Job Run Times for Stampede Trace.

# References - Lecture 8 I


[1] J. Emeras. “Workload Traces Analysis and Replay in Large Scale Distributed Systems”. PhD thesis. University of Grenoble, France, 2013.

[2] Y. Georgiou. “Contributions for Resource and Job Management in High Performance Computing”. PhD thesis. Universite Joseph Fourier, Grenoble, 2010.

[3] D. A. Lifka. “An Extensible Job Scheduling System for Massively Parallel Processor Architectures”. AAI9833045. PhD thesis. Chicago, IL, USA, 1998. ISBN: 0-591-86149-6.

[4] A. W. Mu’alem and D. G. Feitelson. “Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling”. In: *IEEE Trans. Parallel Distrib. Syst.* 12.6 (2001), pp. 529–543. DOI: 10.1109/71.932708.