Scheduling in computational grids
with reservations

Denis Trystram

LIG-MOAIS

Grenoble University, France

AEOLUS, march 9, 2007

General Context

Recently, there was a rapid and deep evolution of high-performance execution platforms: supercomputers, clusters, computational grids, global computing, …

Need of efficient tools for resource management for dealing with these new systems.

This talk will investigate some scheduling problems and focus on reservations.

Parallel computing today.

- Different kinds of platforms
  - Clusters, collection of clusters, grid, global computing
  - Set of temporary unused resources
  - Autonomous nodes (P2P)
- Our view of grid computing (reasonable trade-off):
  - Set of computing resources under control (no hard authentication problems, no random addition of computers, etc.)

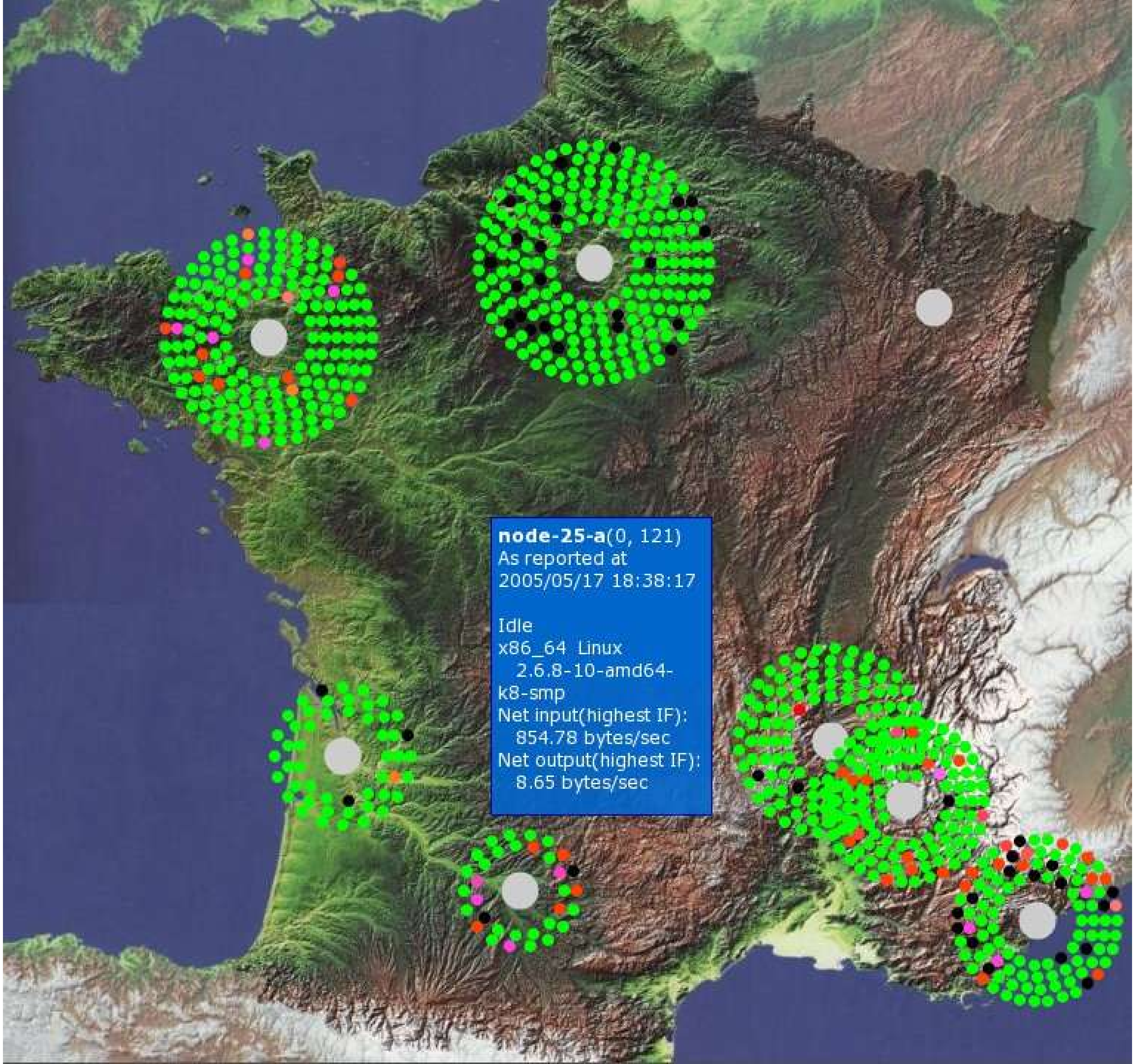
Content

- Some preliminaries (Parallel tasks model)
- Scheduling and packing problems
- On-line versus off-line: batch scheduling
- Multi-criteria
- Reservations

A national french initiative: GRID5000

Several local computational grids (like CiGri)

National project with shared resources and competences with almost 4000 processors today with local administration but centralized control.



node-25-a(0, 121)
As reported at
2005/05/17 18:38:17
Idle
x86_64 Linux
2.6.8-10-amd64-
k8-smp
Net input(highest IF):
854.78 bytes/sec
Net output(highest IF):
8.65 bytes/sec

Target Applications

New execution supports created new applications (data-mining, bio-computing, coupling of codes, interactive, virtual reality, …).

Interactive computations (human in the loop), adaptive algorithms, etc.. See MOAIS project for more details.

Scheduling problem
(informally)

Given a set of tasks, the problem is to determine when and where to execute the tasks (according to the precedence constraints - if any - and to the target architecture).

Central Scheduling Problem

The basic problem P | prec, pj | Cmax is NP-hard [Ulmann75].

Thus, we are looking for « good » heuristics.

Central Scheduling Problem

The basic problem P | prec, pj | Cmax is NP-hard [Ulmann75].

Thus, we are looking for « good » heuristics.

low cost

based on theoretical analysis:
good approximation factor

Available models

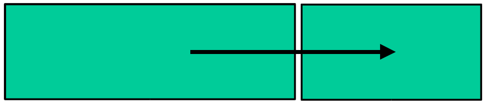
Extension of « old » existing models (delay)

Parallel Tasks

Divisible load

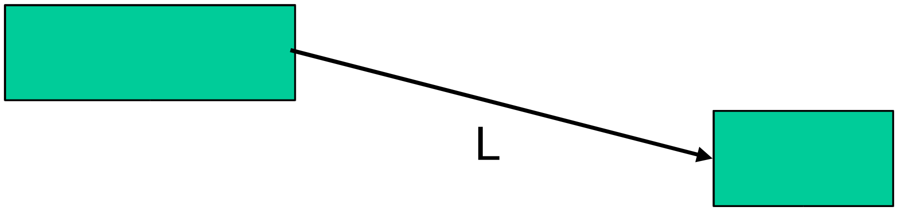
Delay:

if two consecutive tasks are allocated on different processors, we have to pay a communication delay.



Delay:
if two consecutive tasks are allocated on different processors, we have to pay a communication delay.

If L is large, the problem is very hard
(no approximation algorithm is known)



Extensions of delay

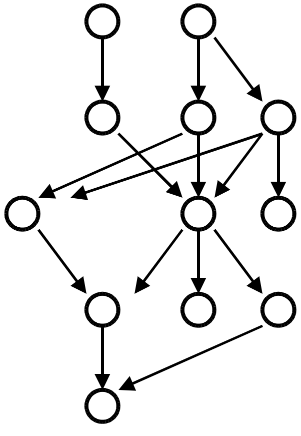
Some tentatives have been proposed (like LogP).

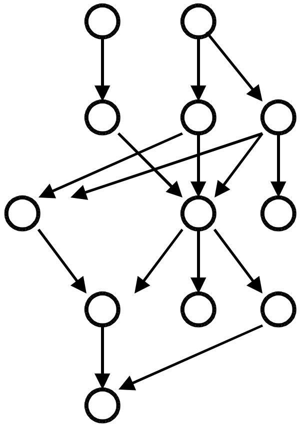
Not adequate for grids (heterogeneity, large delays, hierarchy, incertainties)…

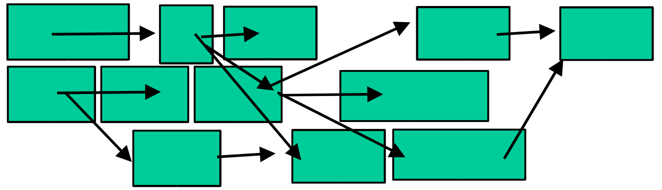
Parallel Tasks

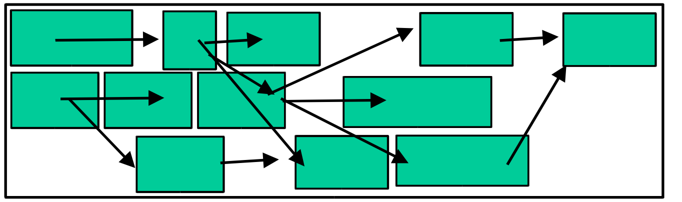
Extension of classical sequential tasks: each task may require more than one processor for its execution [Feitelson and Rudolph].

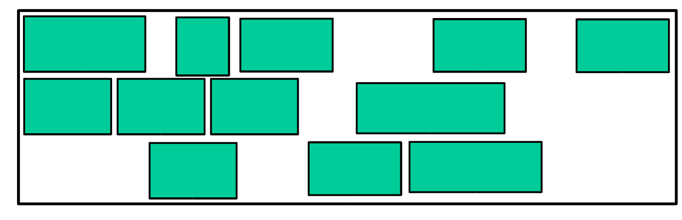
Job

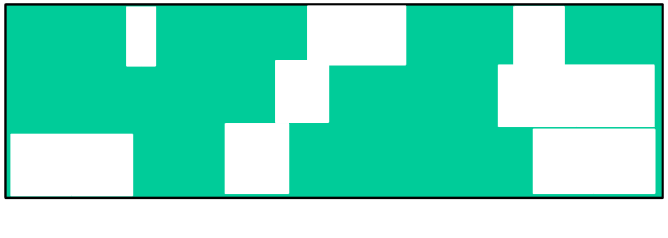


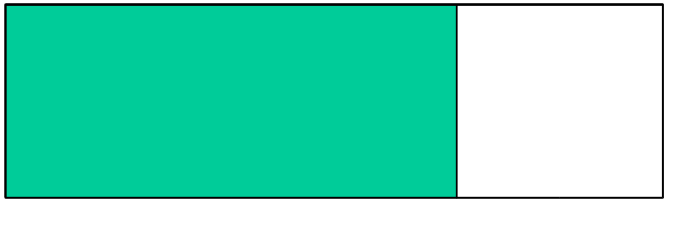


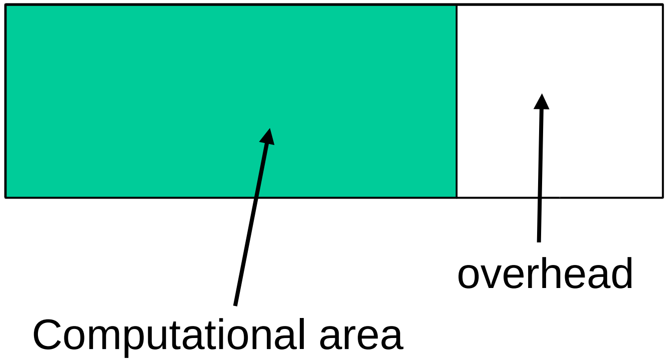










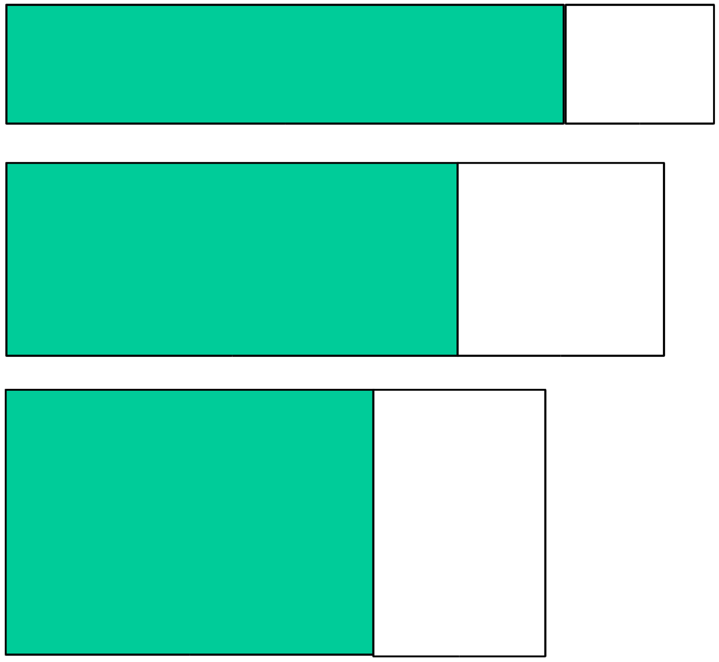


overhead
Computational area

Classification

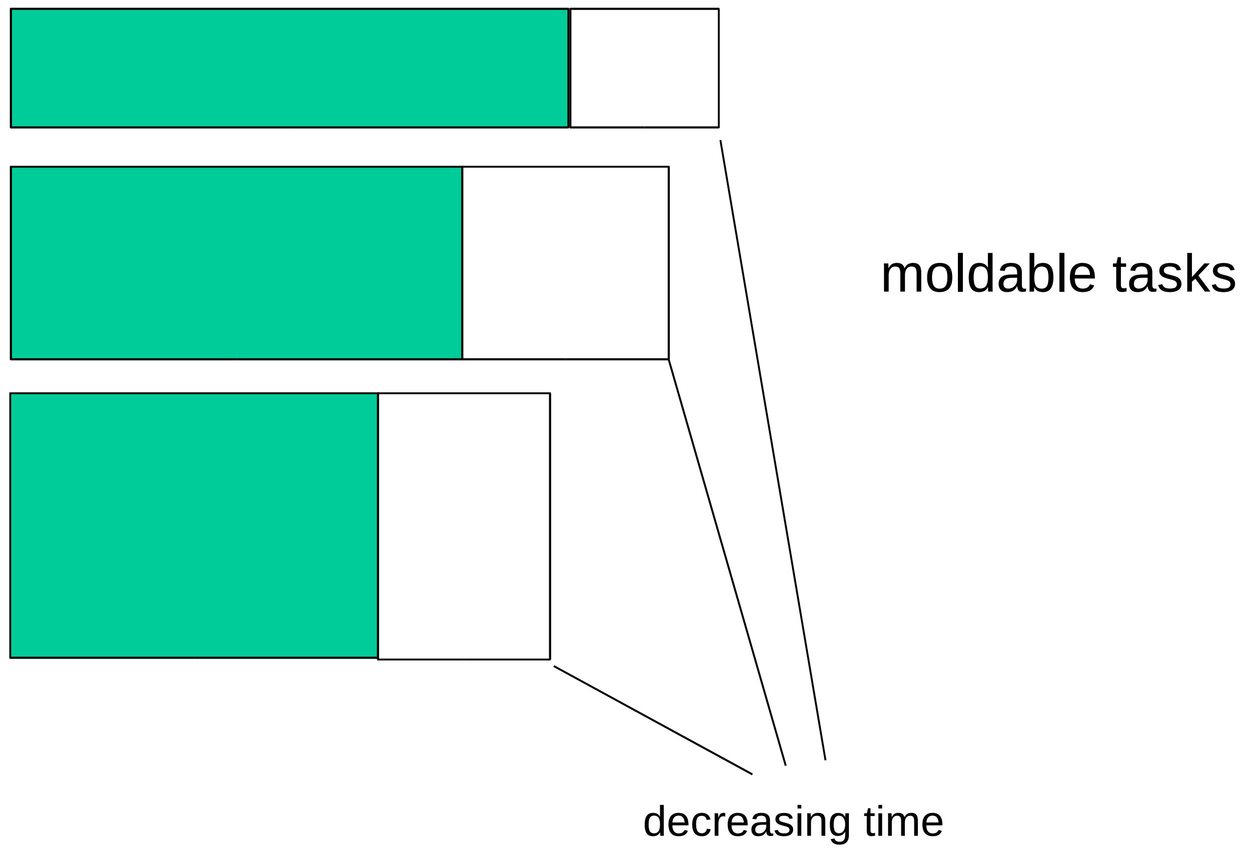
rigid tasks

Classification

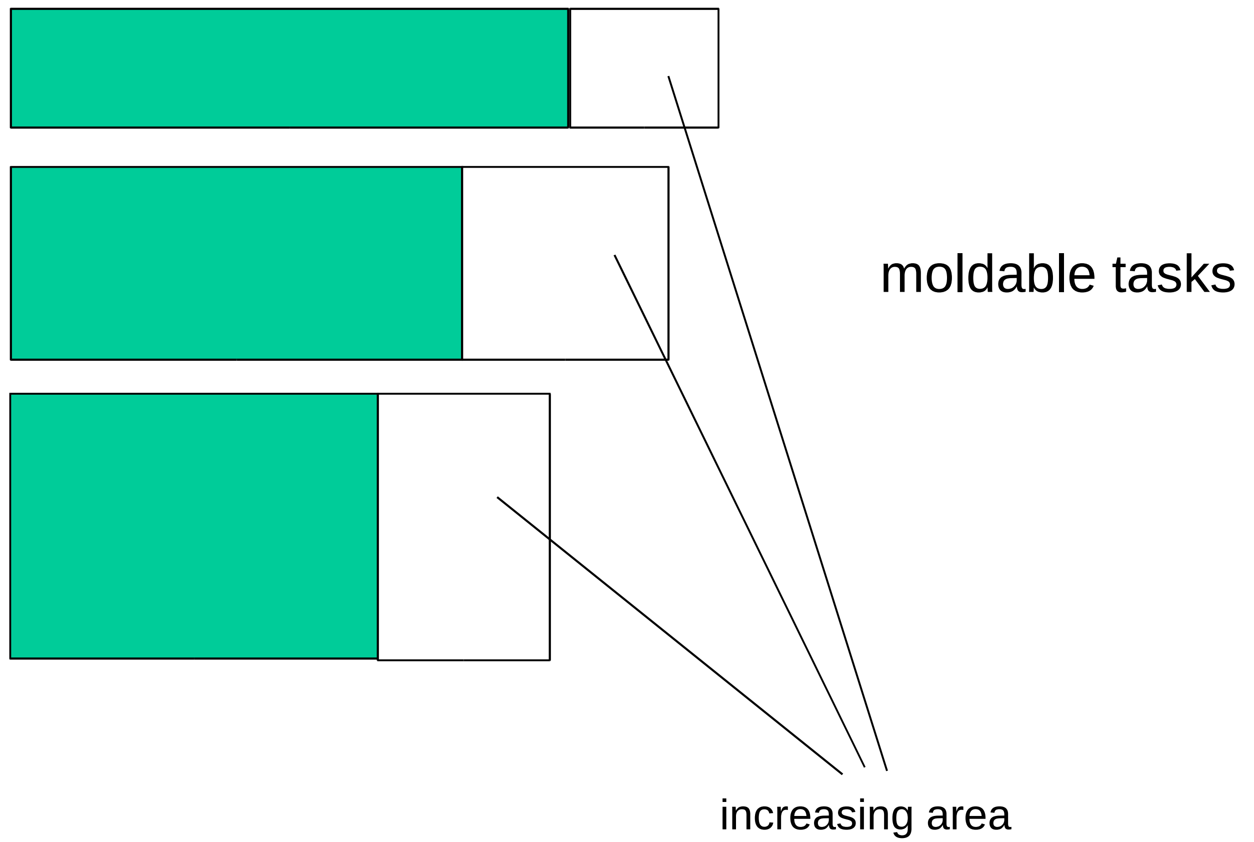


moldable tasks

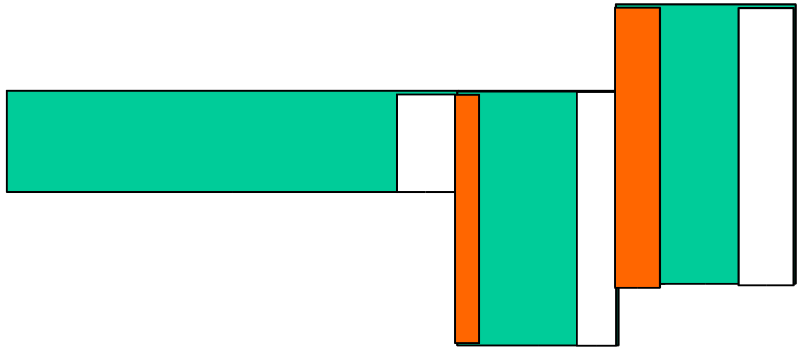
Classification



Classification

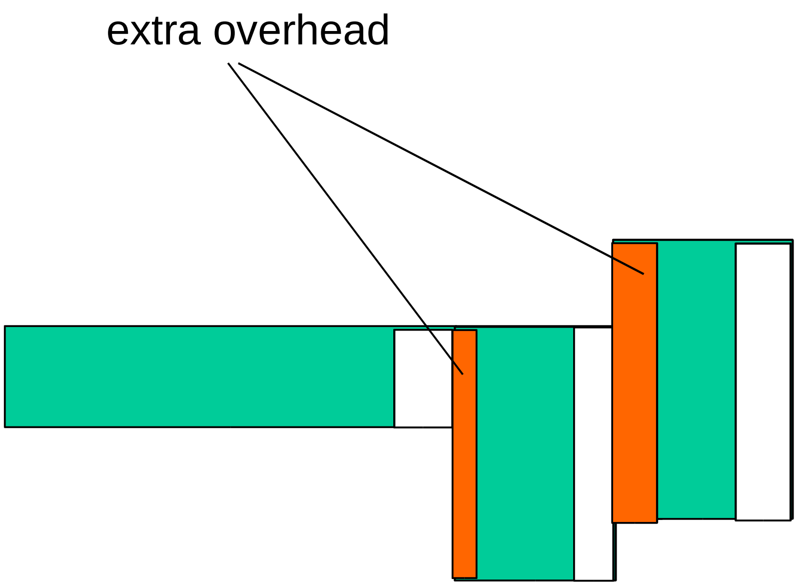


Classification



malleable tasks

Classification



malleable tasks

Divisible load

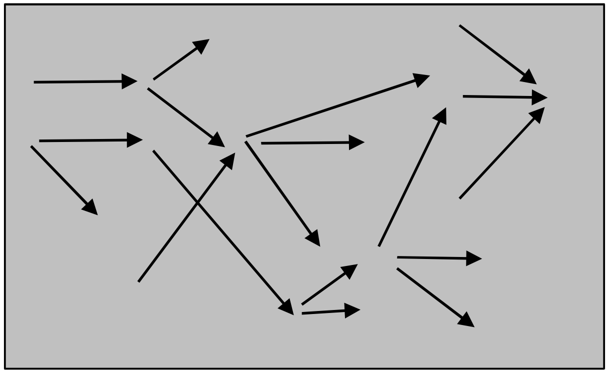
Also known as « bag of tasks »:

Big amount of arbitrary small computational units.

Divisible load

Also known as « bag of tasks »:

Big amount of arbitrary small computational units.



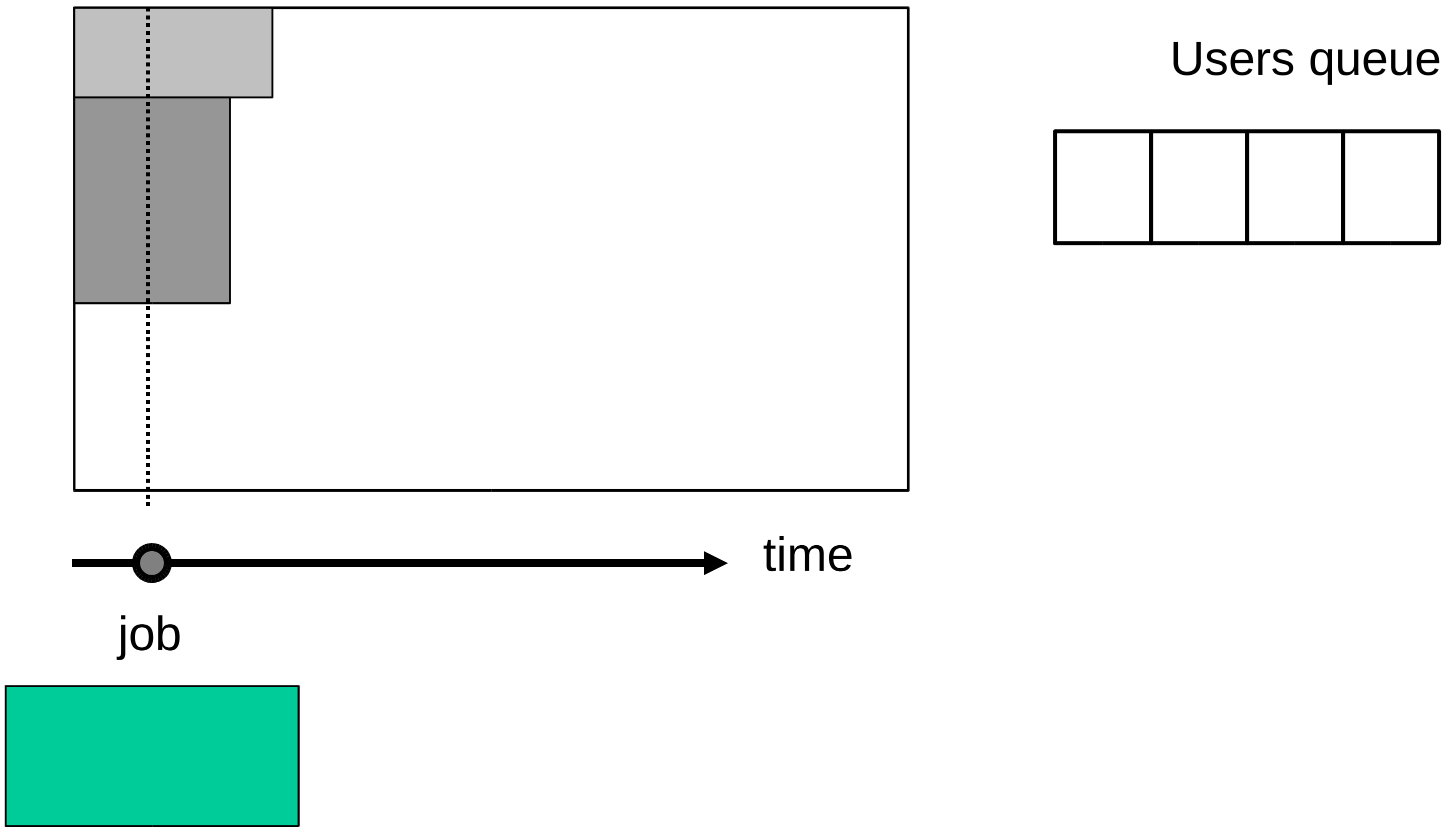
Divisible load

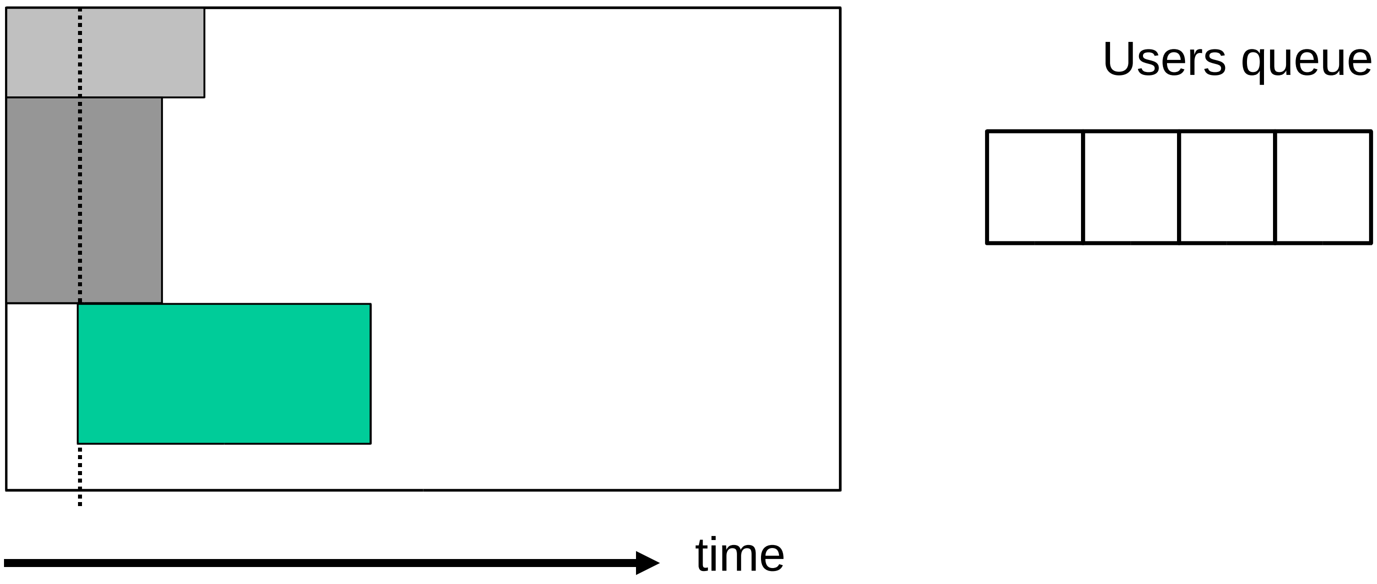
(asymptotically) optimal for some criteria (throughput).

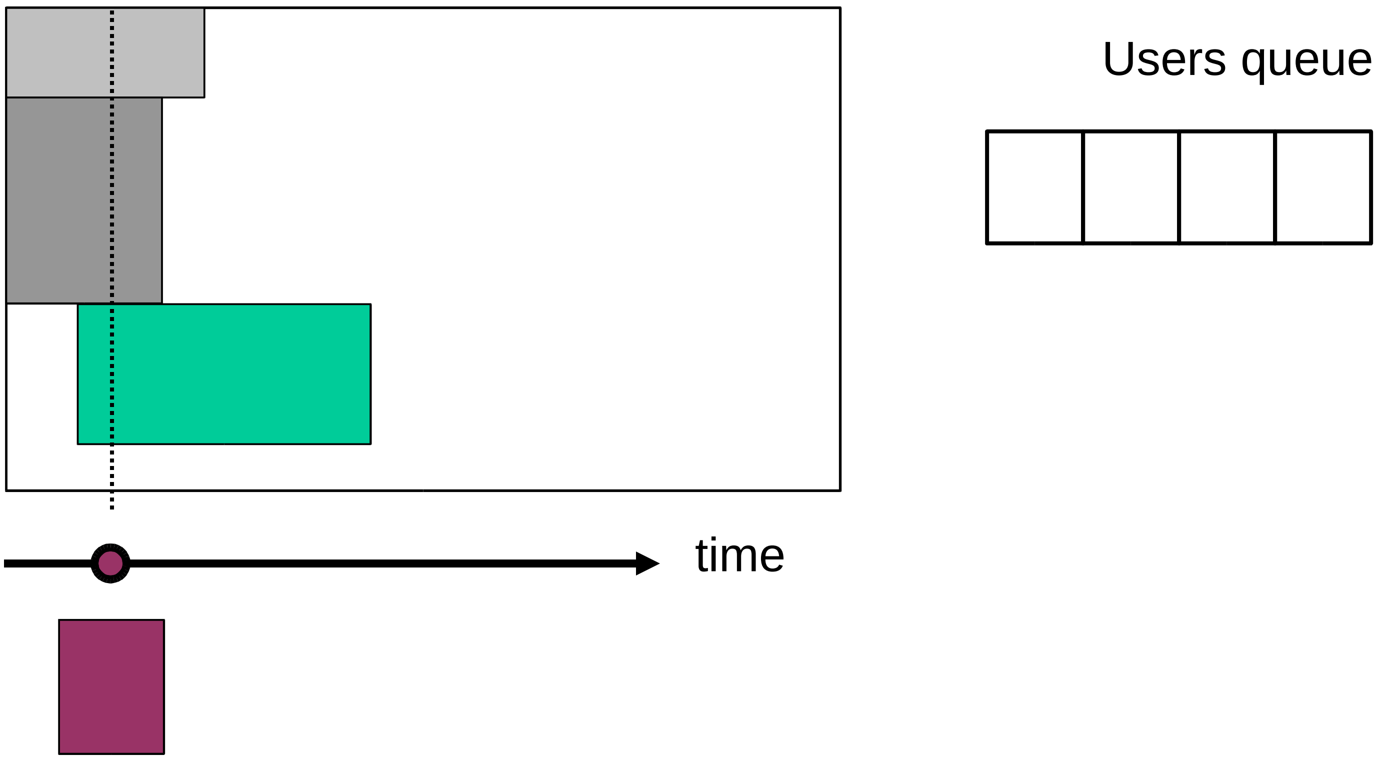
Valid only for specific applications with regular patterns.

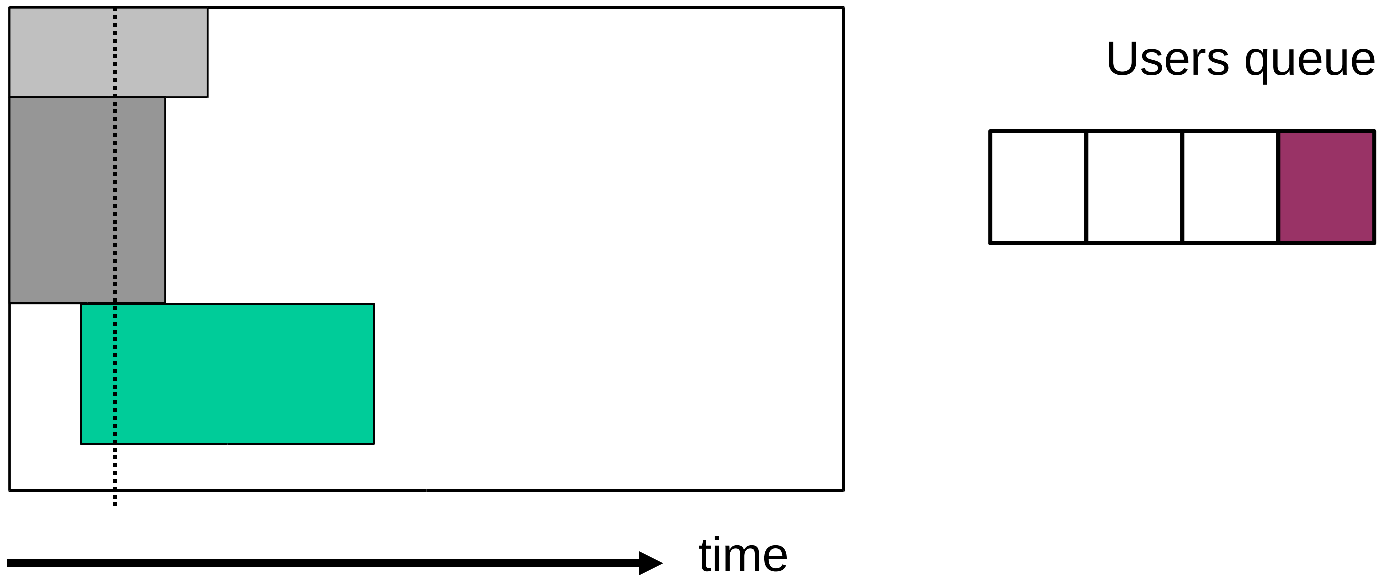
Popular for best effort jobs.

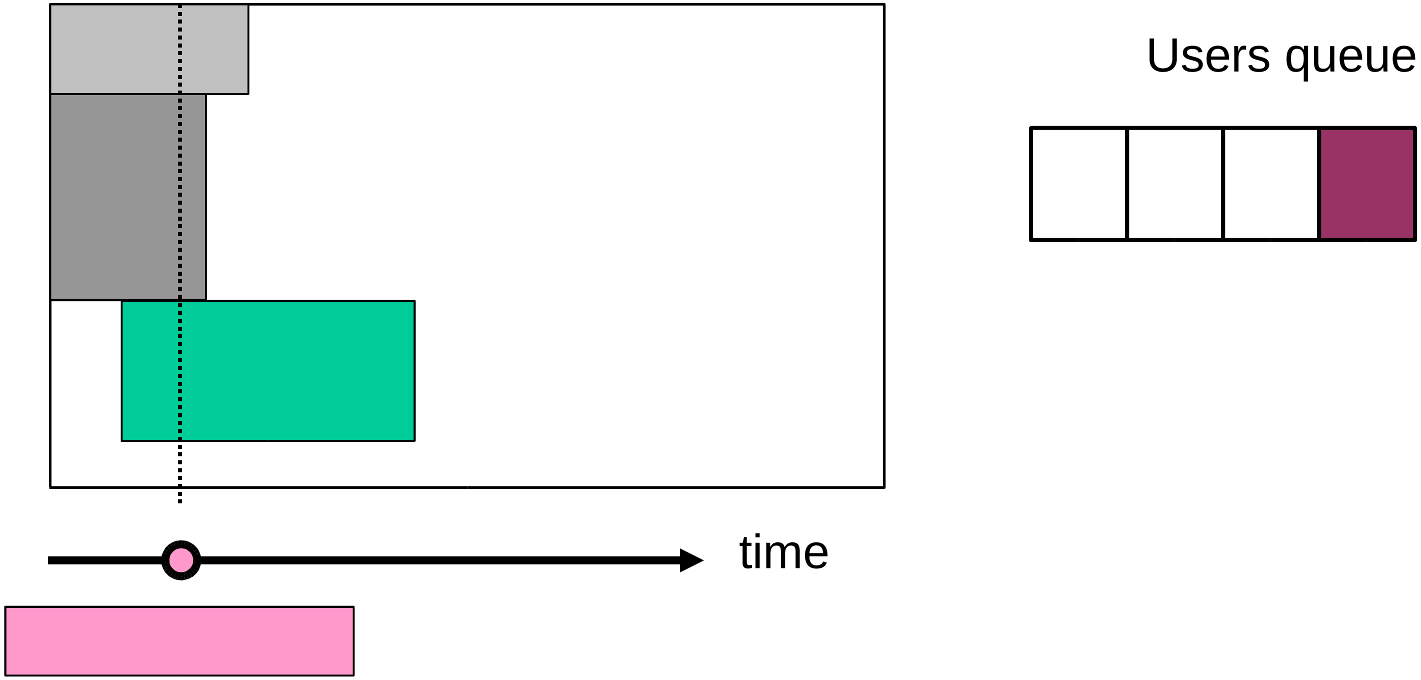
Resource management
in clusters

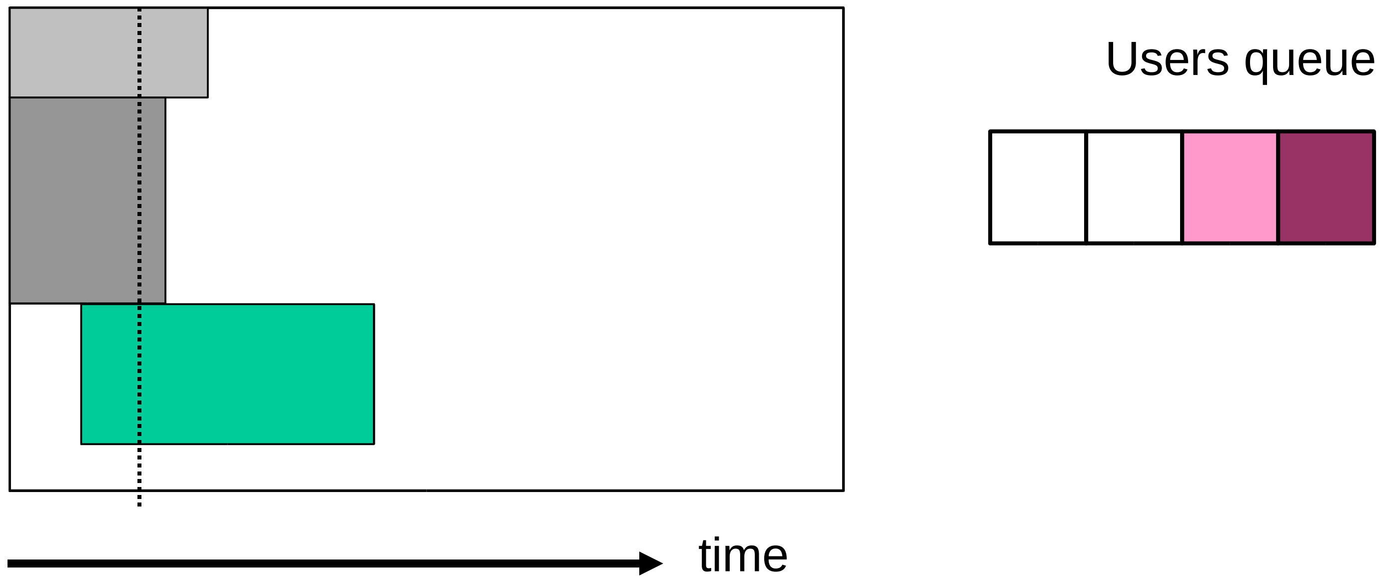

Users queue
time
job


Users queue
time

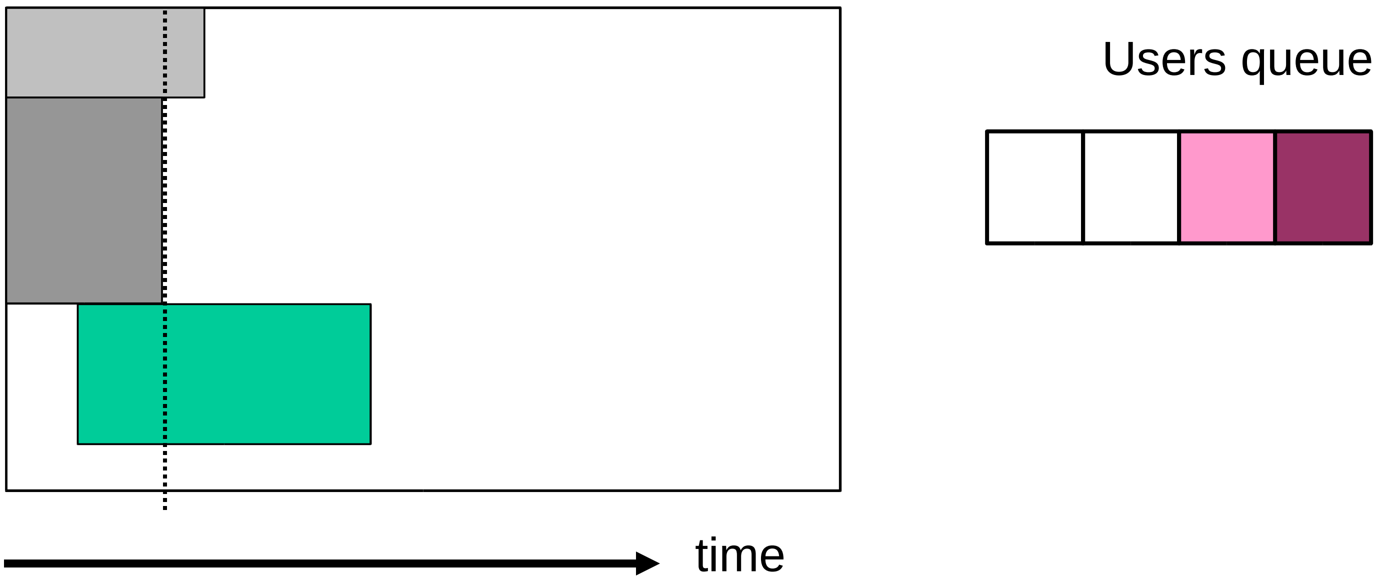

Users queue
time


Users queue
time

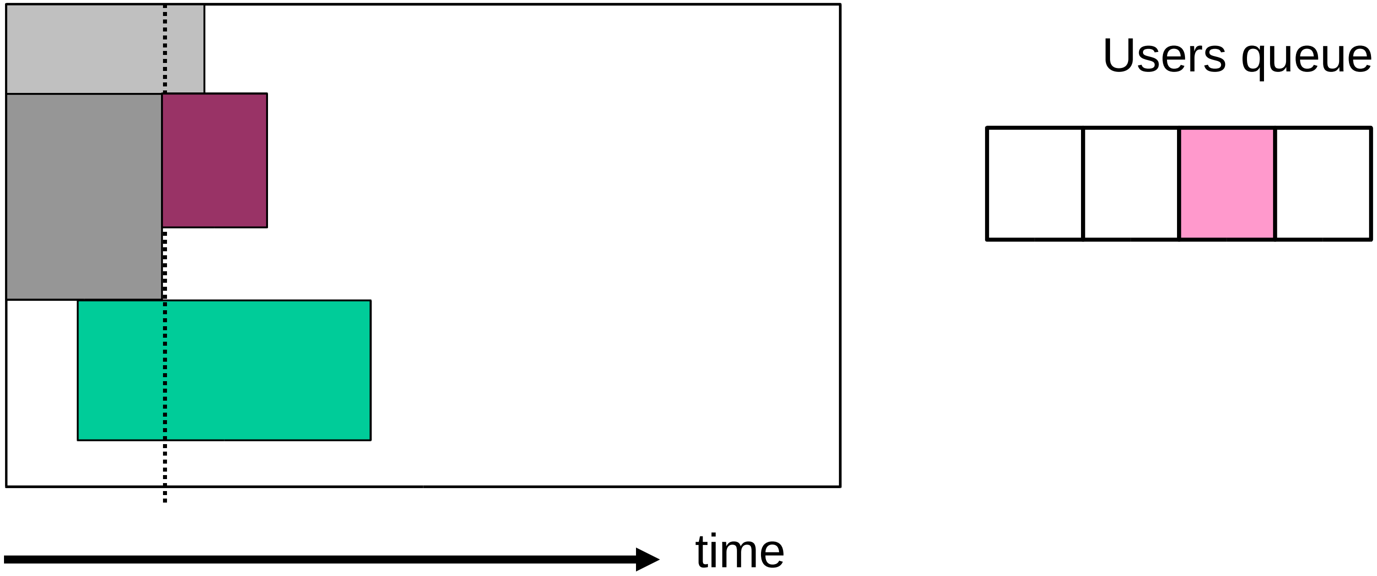

Users queue
time

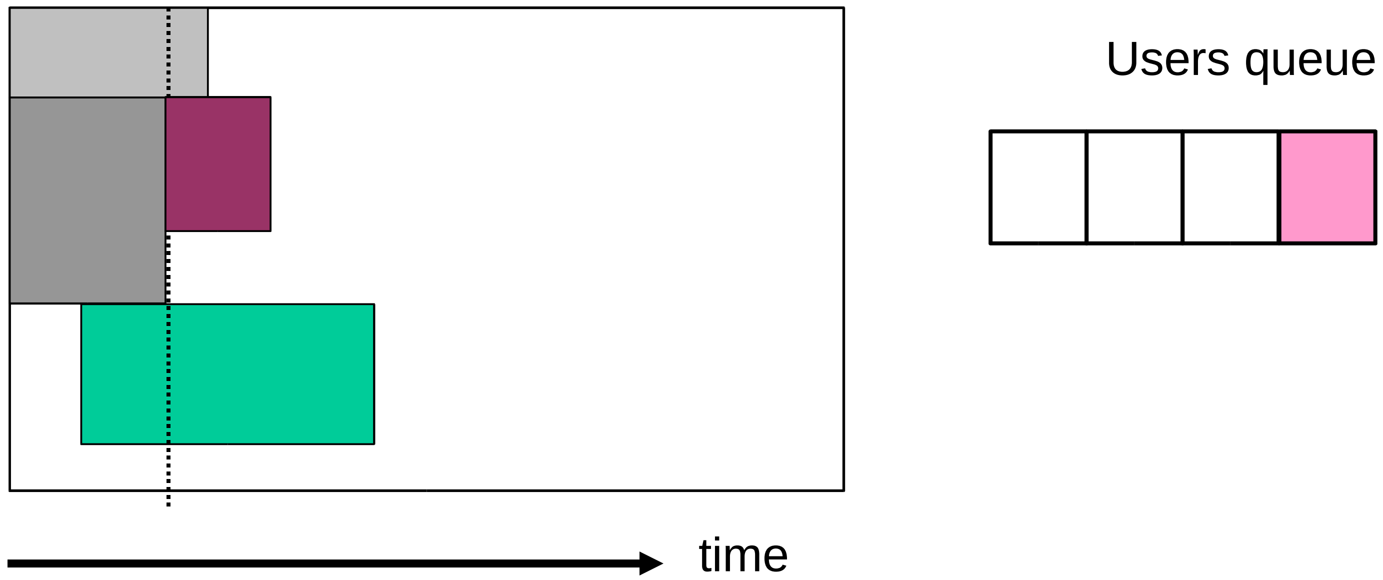


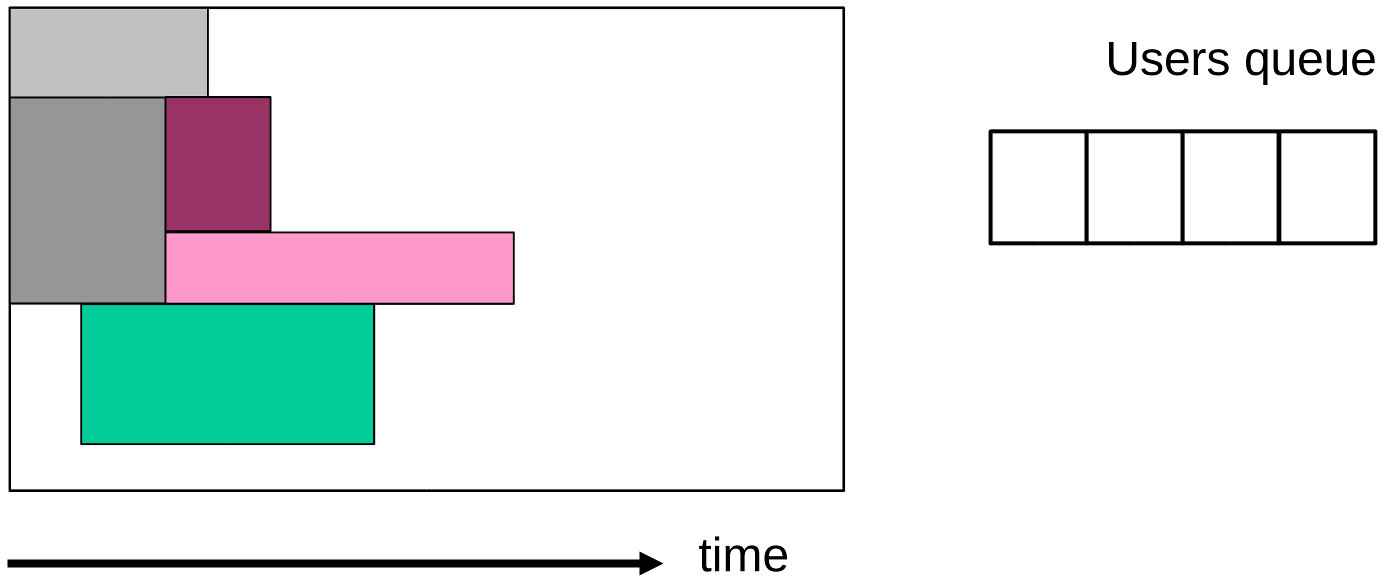
Users queue
time

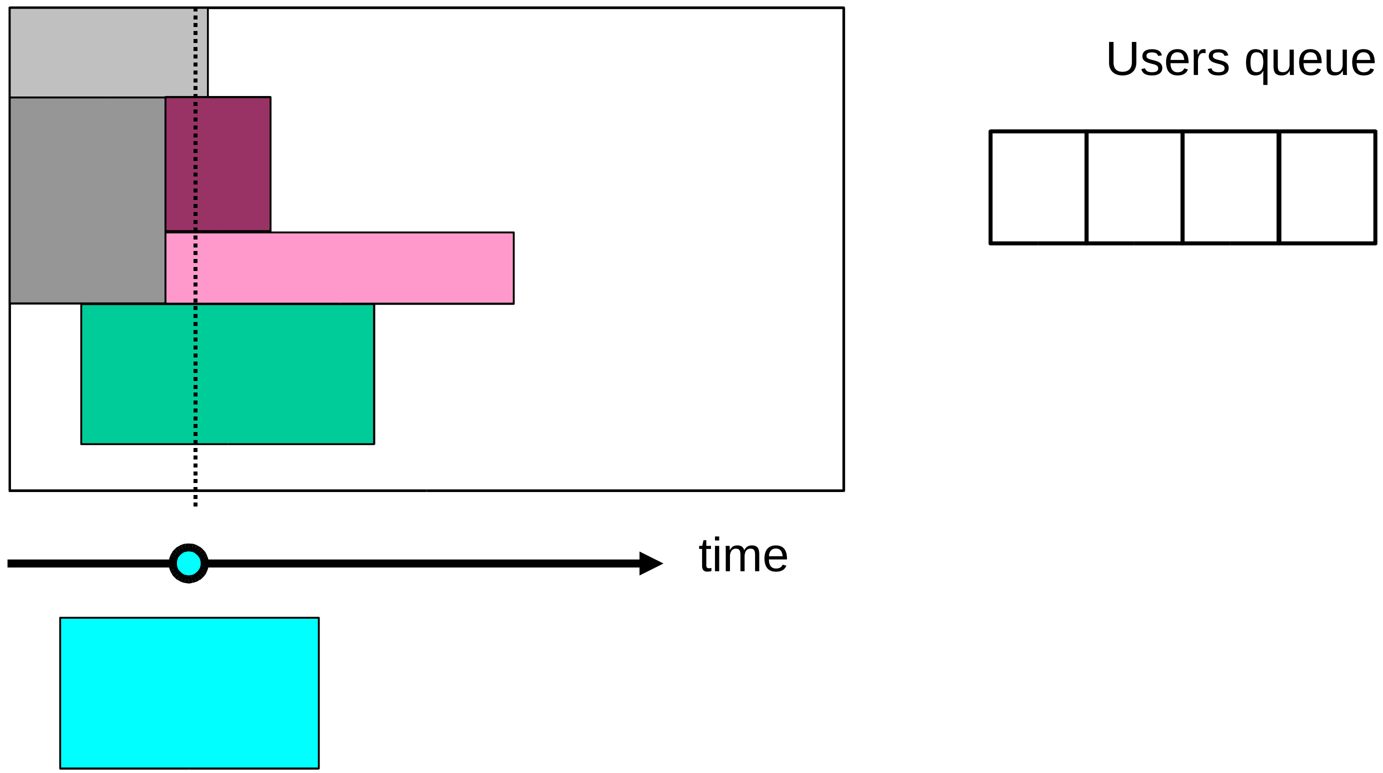


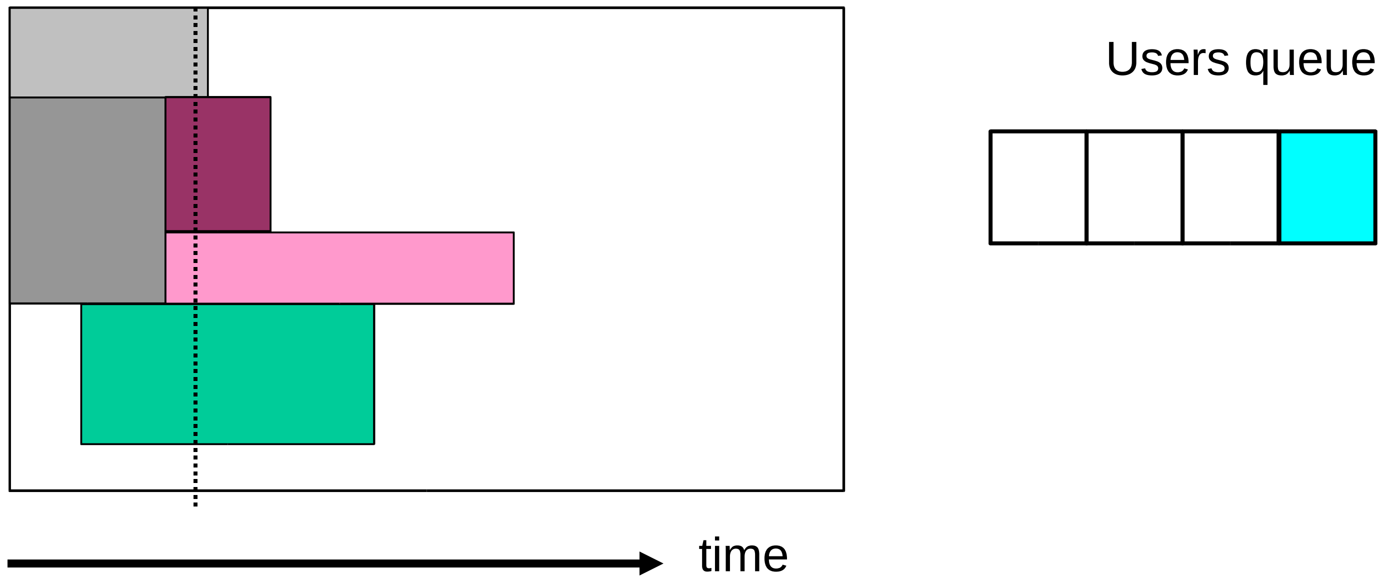
Users queue
time

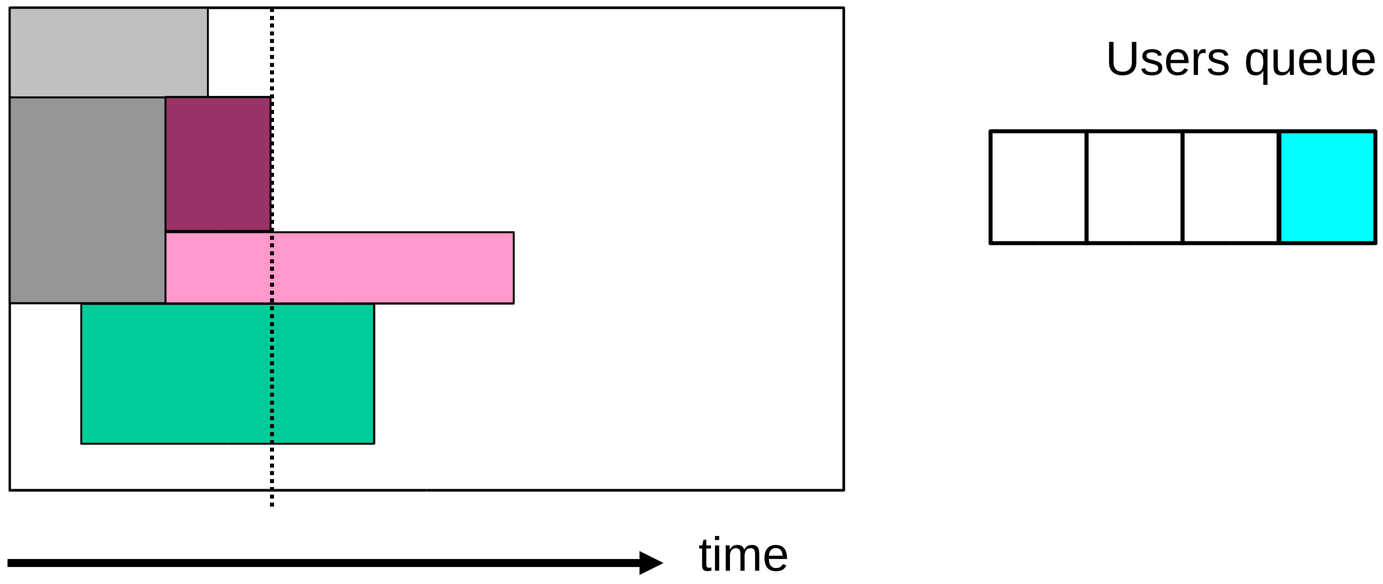

Users queue
time

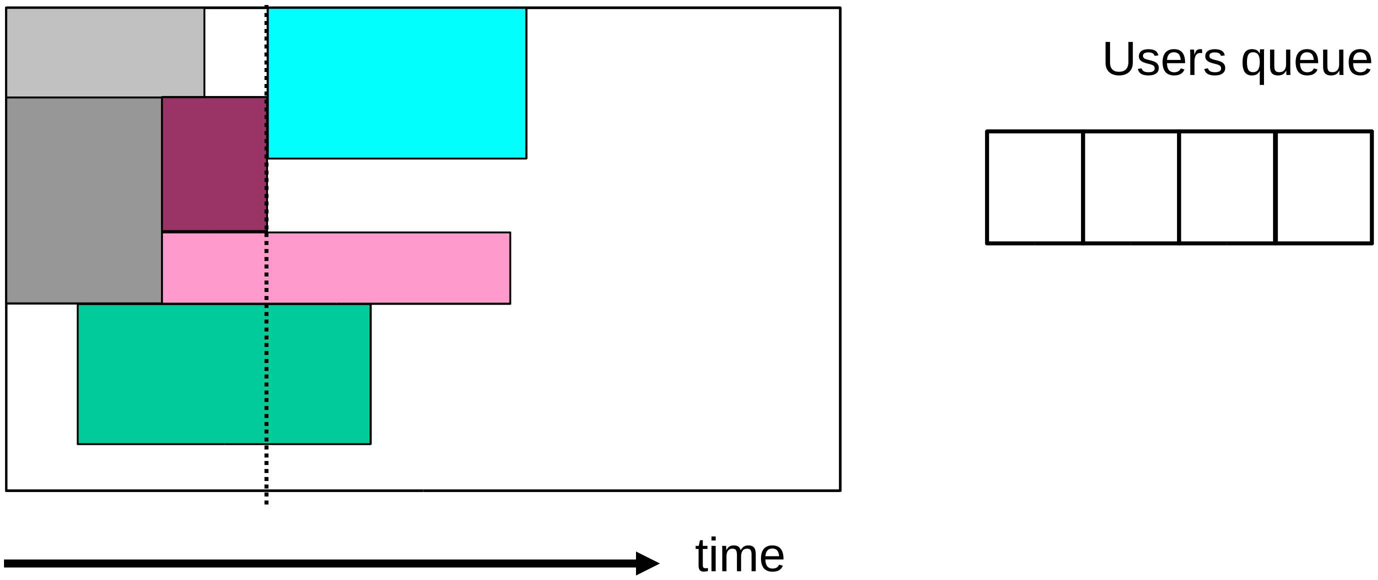

Users queue
time


Users queue
time


Users queue
time

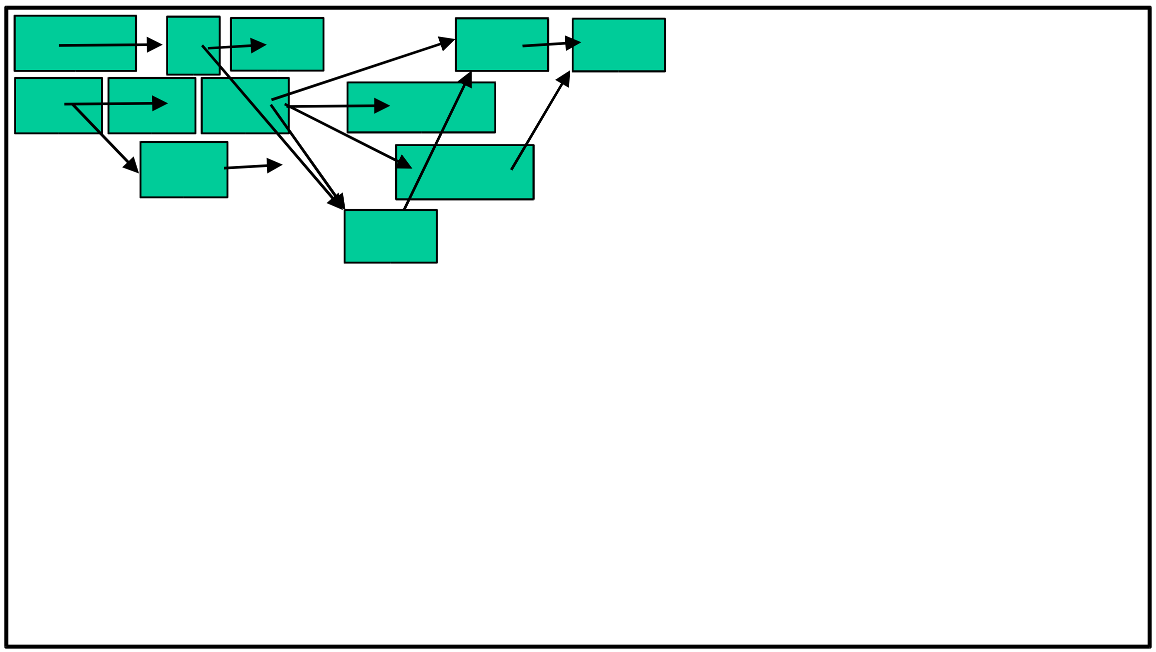

Users queue
time


Users queue
time

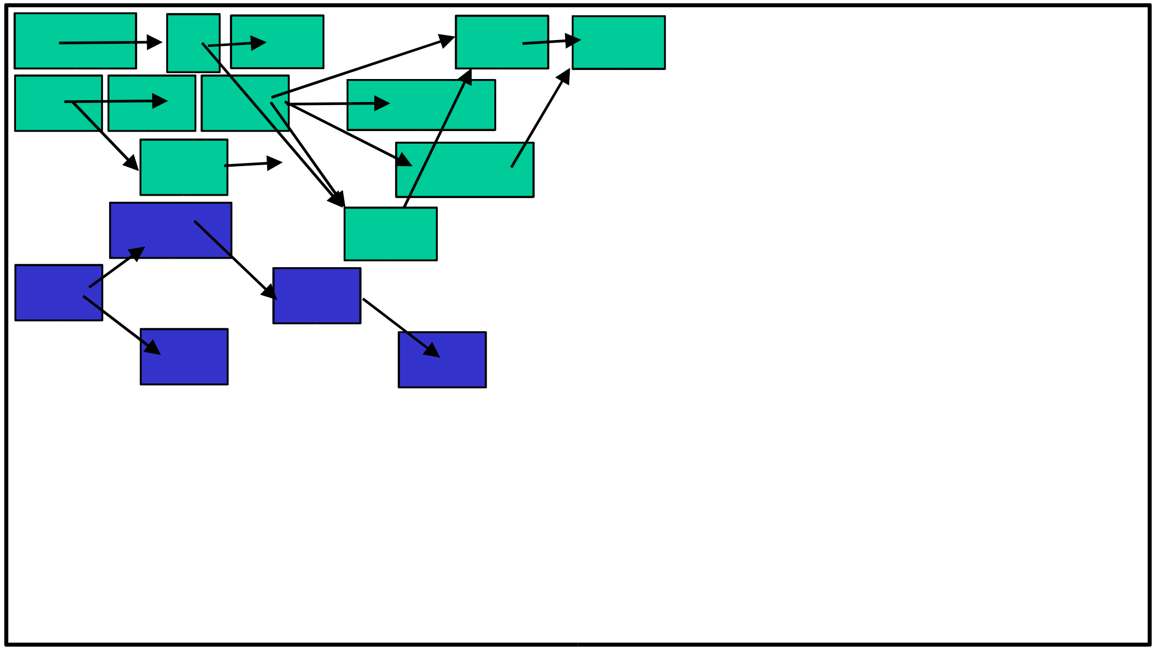

Users queue
time

Integrated approach

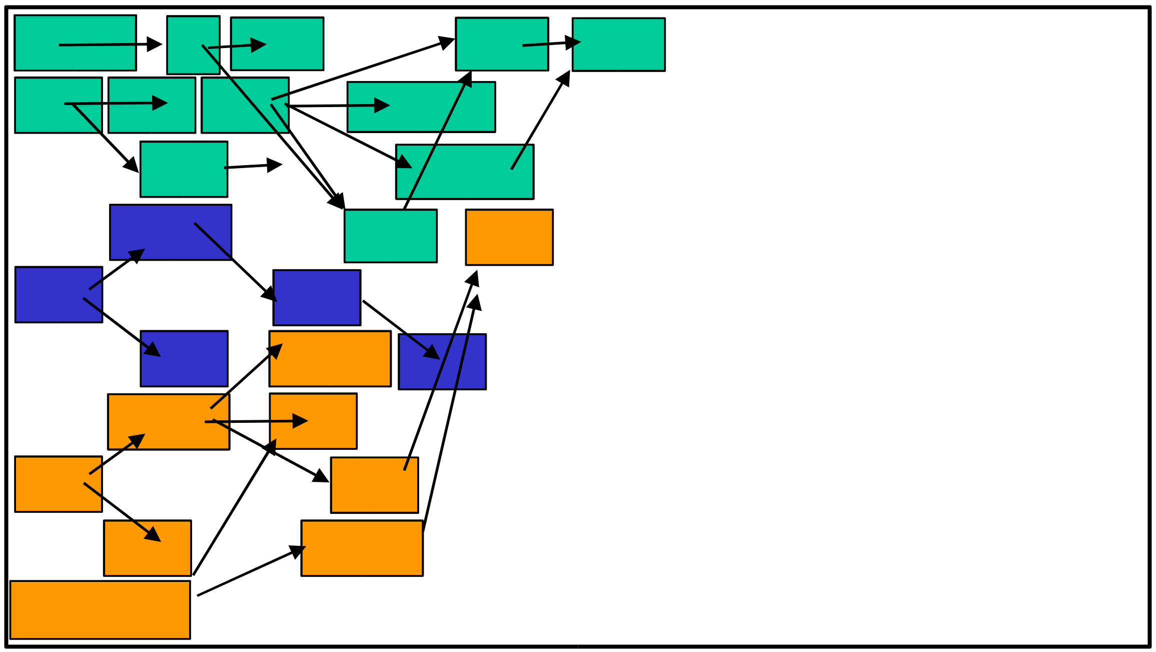
m



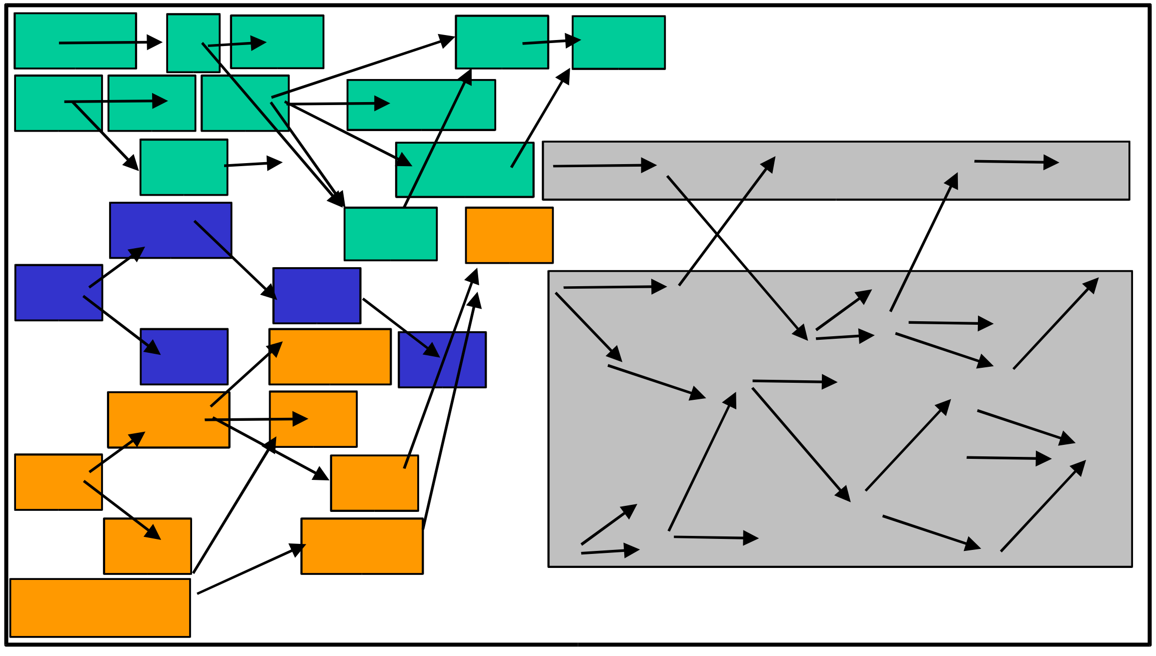
m



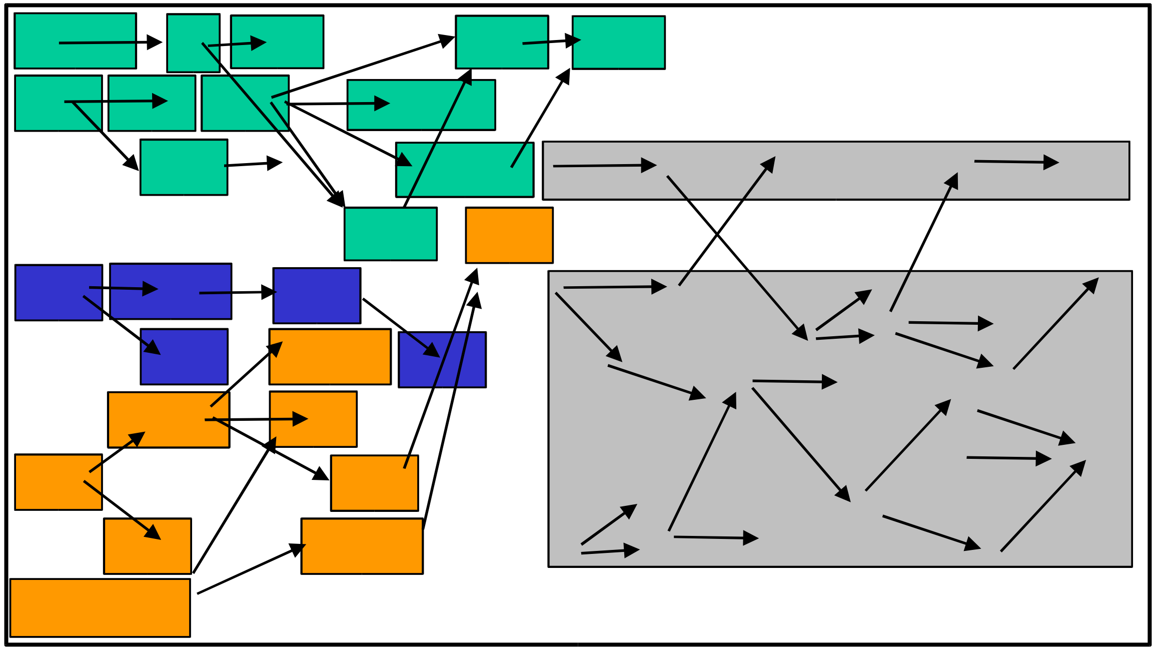
m



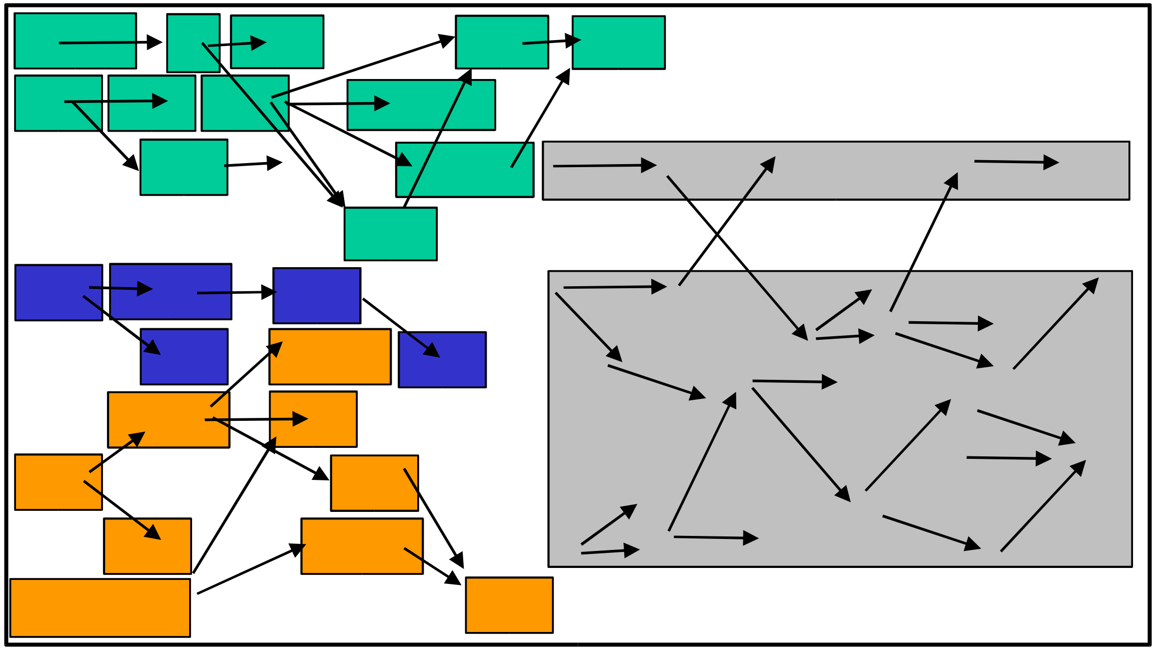
m



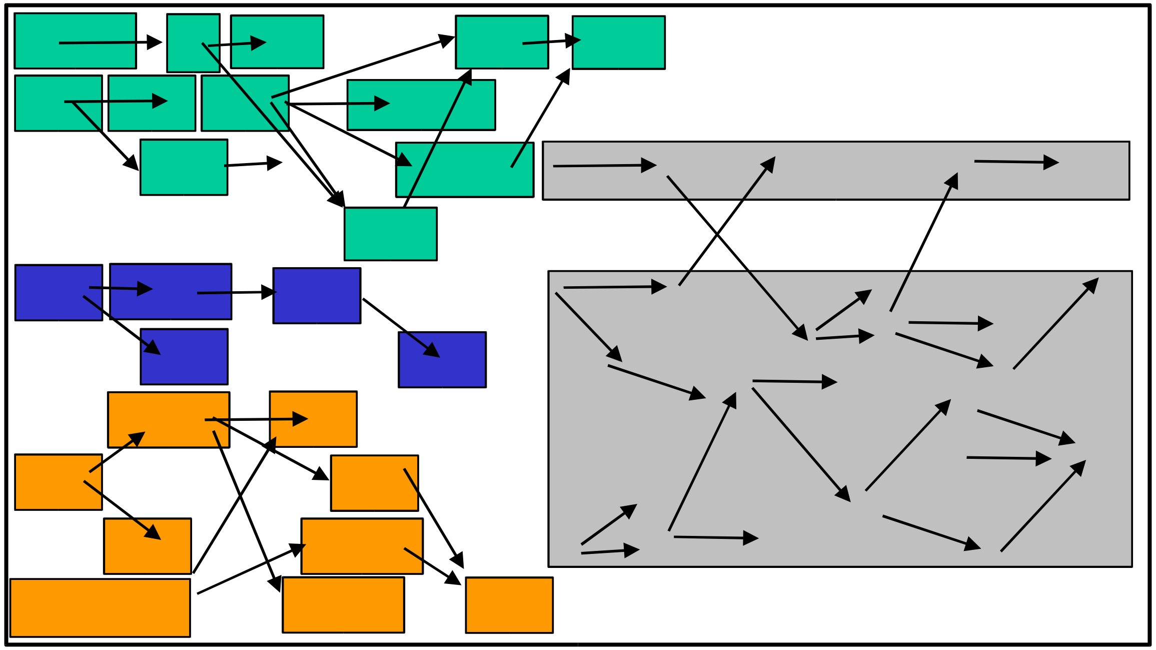
m



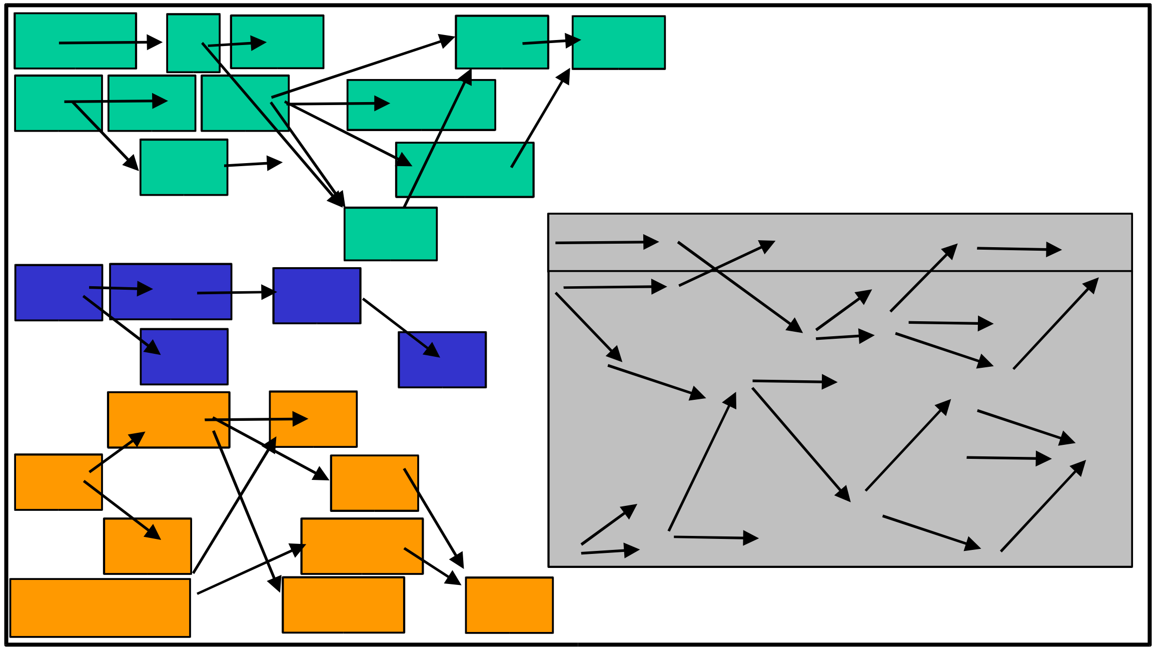
m



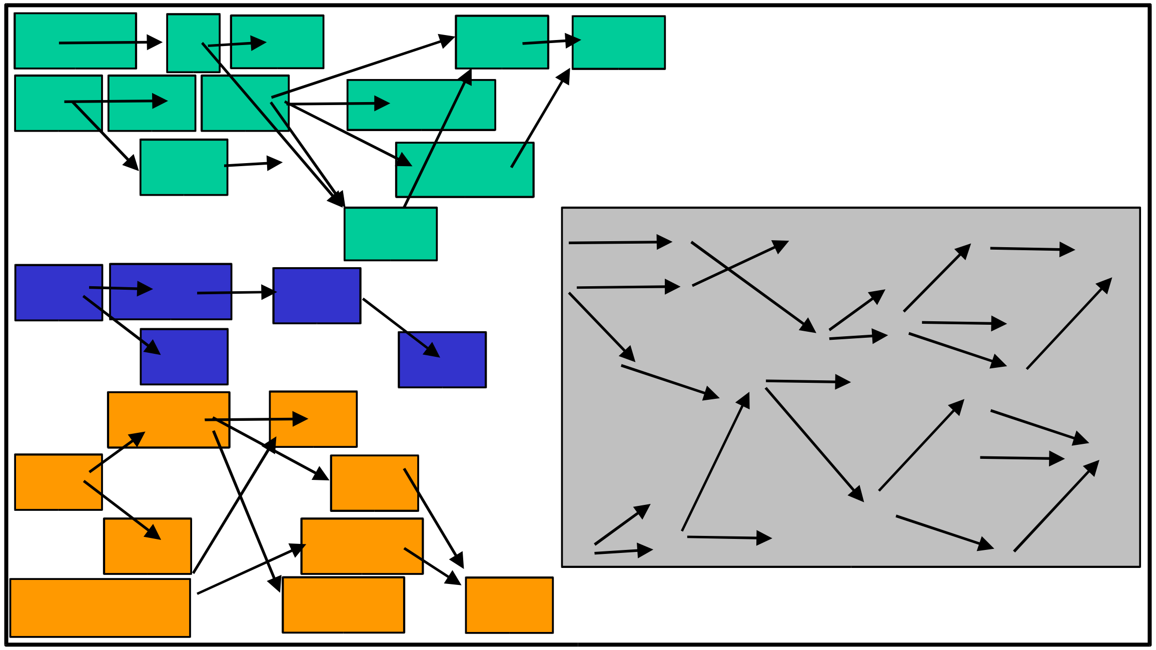
m



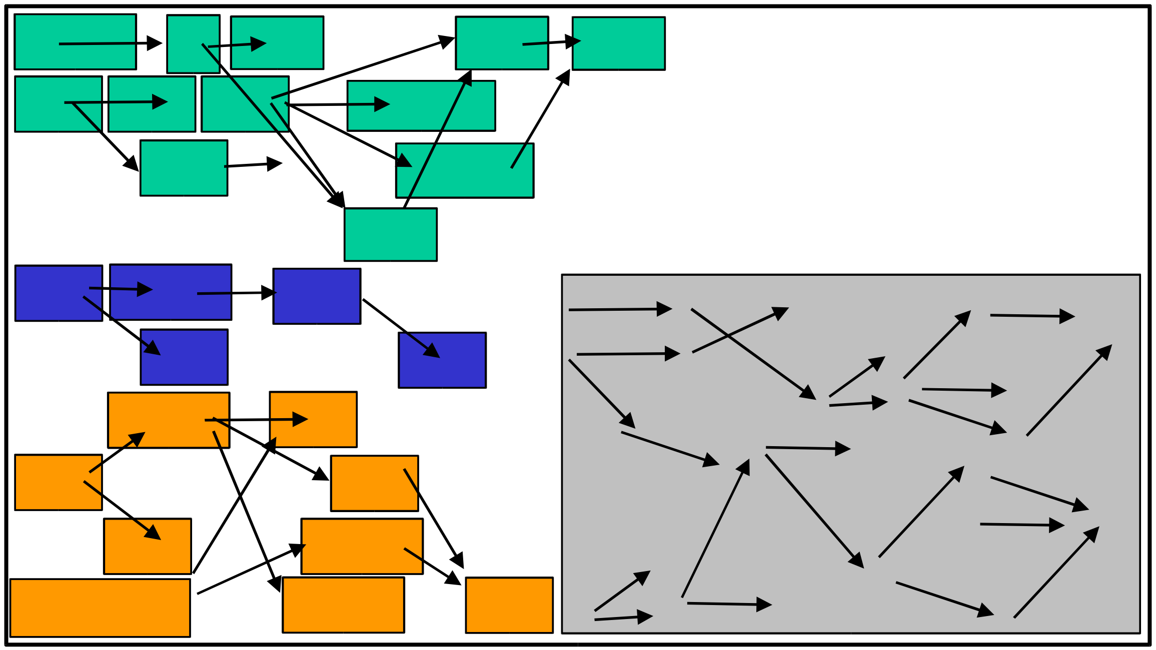
m



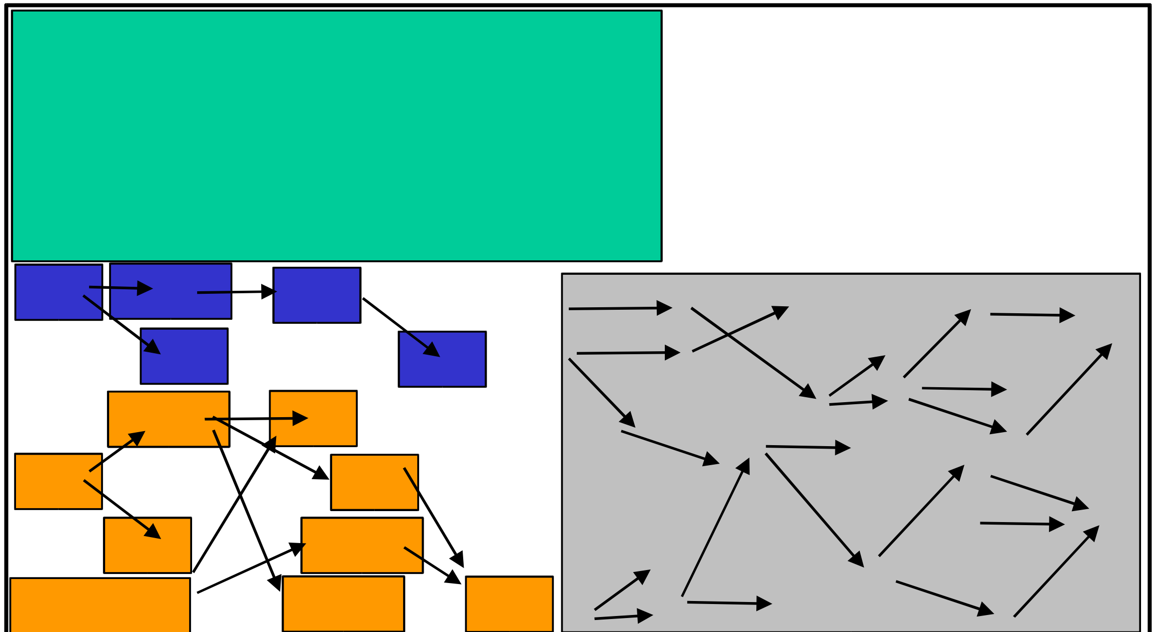
m



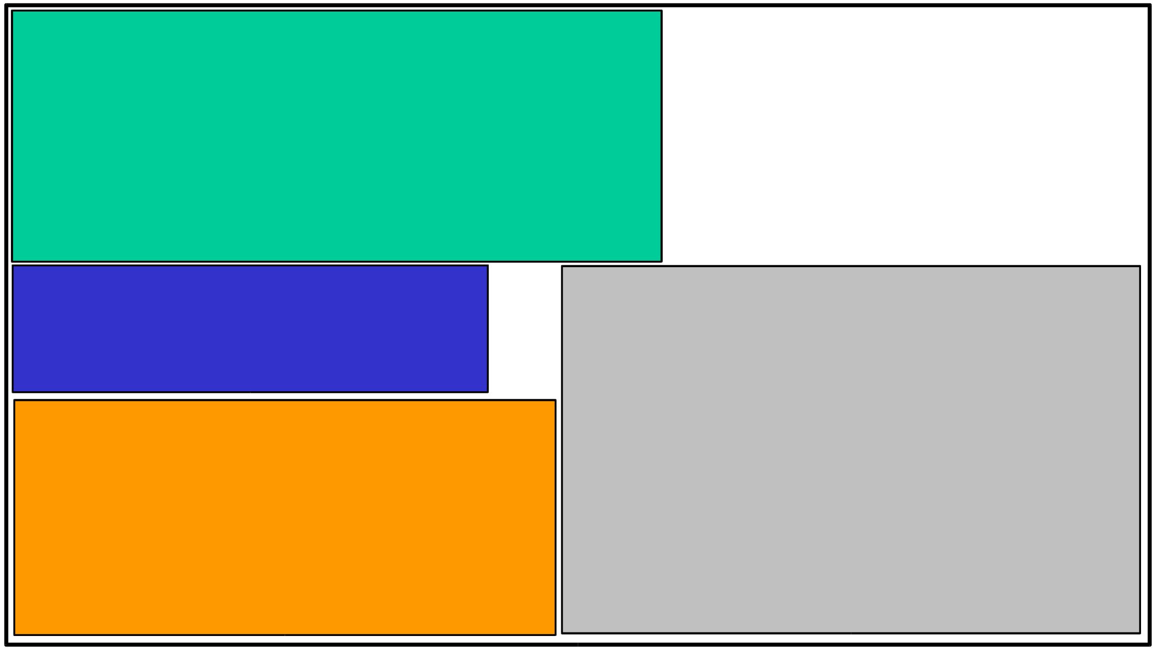
m



m



m

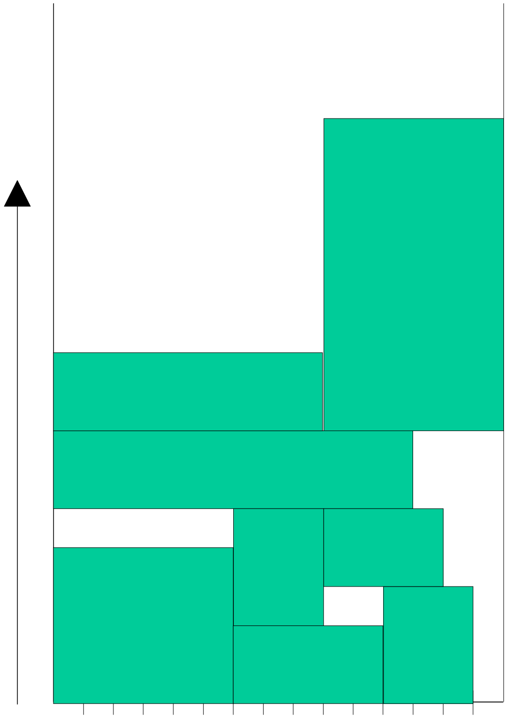


(strip) Packing problems

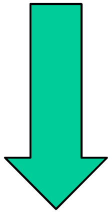
The schedule is divided into two successive steps:

2. Allocation problem
3. Scheduling with preallocation (NP-hard in general [Rayward-Smith 95]).

Scheduling: on-line vs off-line



On-line: no knowledge about the future

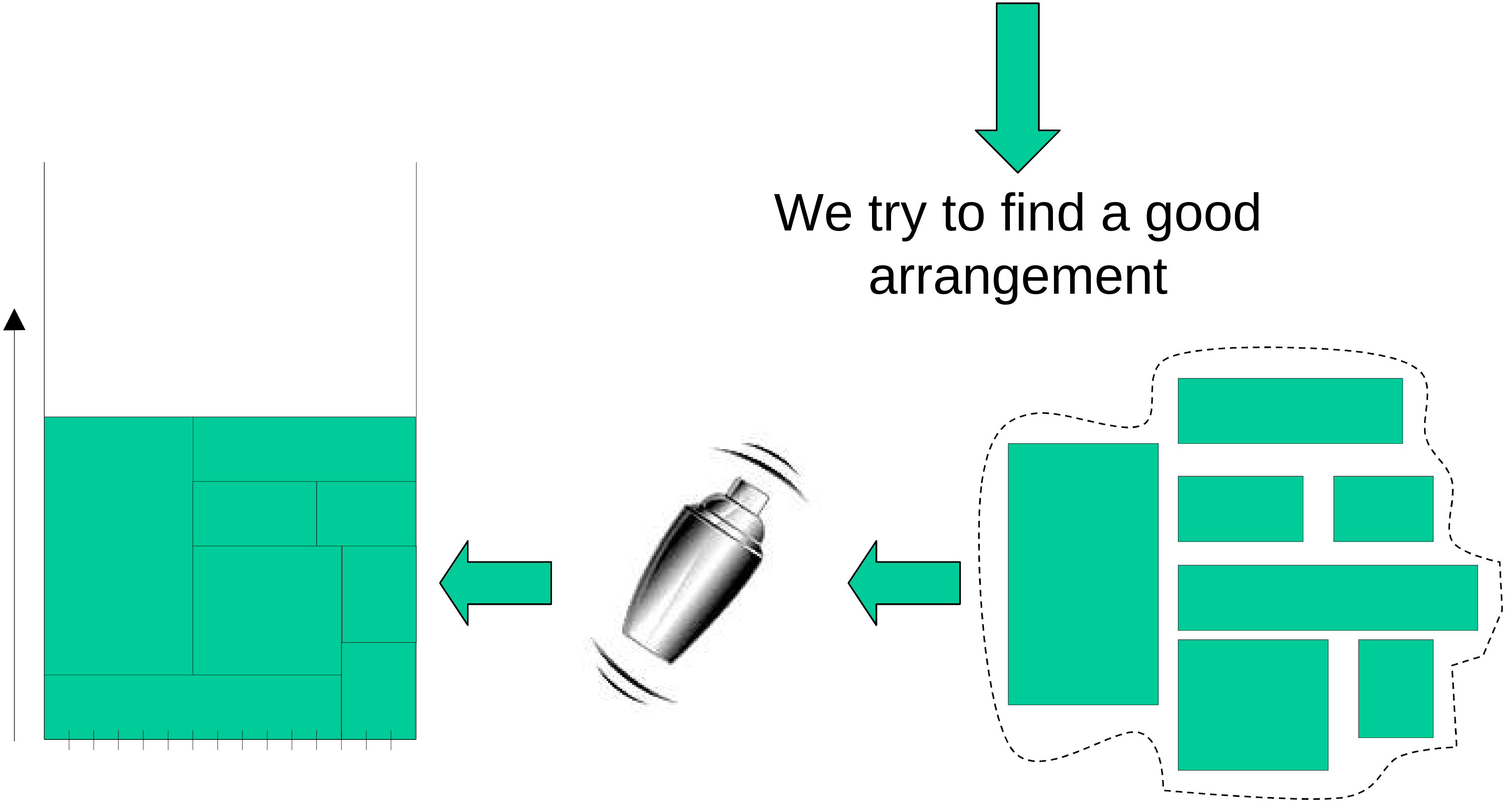


We take the scheduling decision while other jobs arrive

Scheduling: on-line vs off-line

Off-line: we have a finite set of works

We try to find a good arrangement



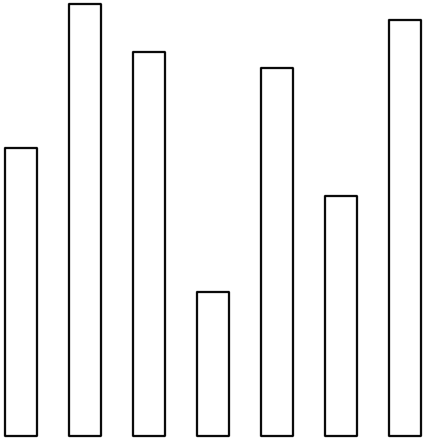
Off-line scheduler

Problem:

Schedule a set of independent moldable jobs (clairvoyant).

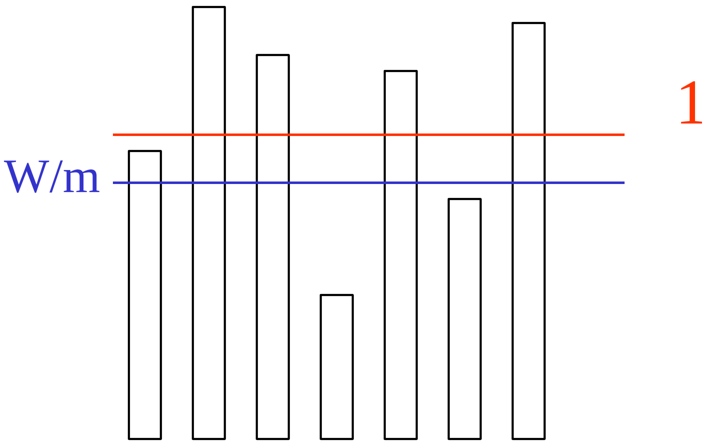
Penalty functions have somehow to be estimated (using complexity analysis or any prediction-measurement method like the one obtained by the log analysis).

Example

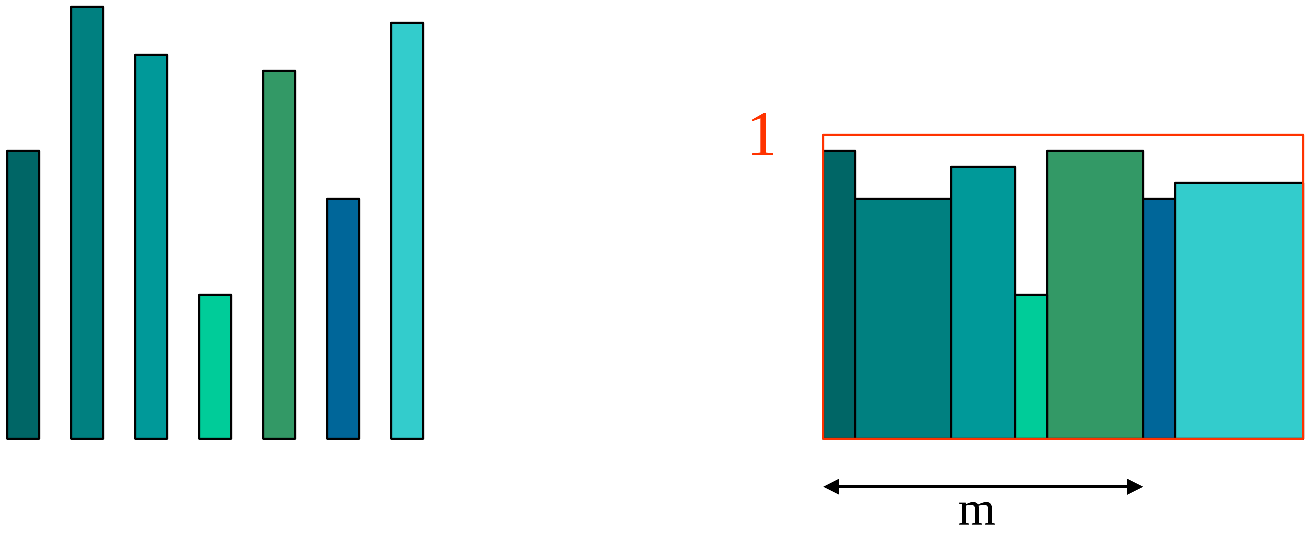


Let us consider 7 MT to be scheduled on m=10 processors.

Canonical Allotment



Canonical Allotment



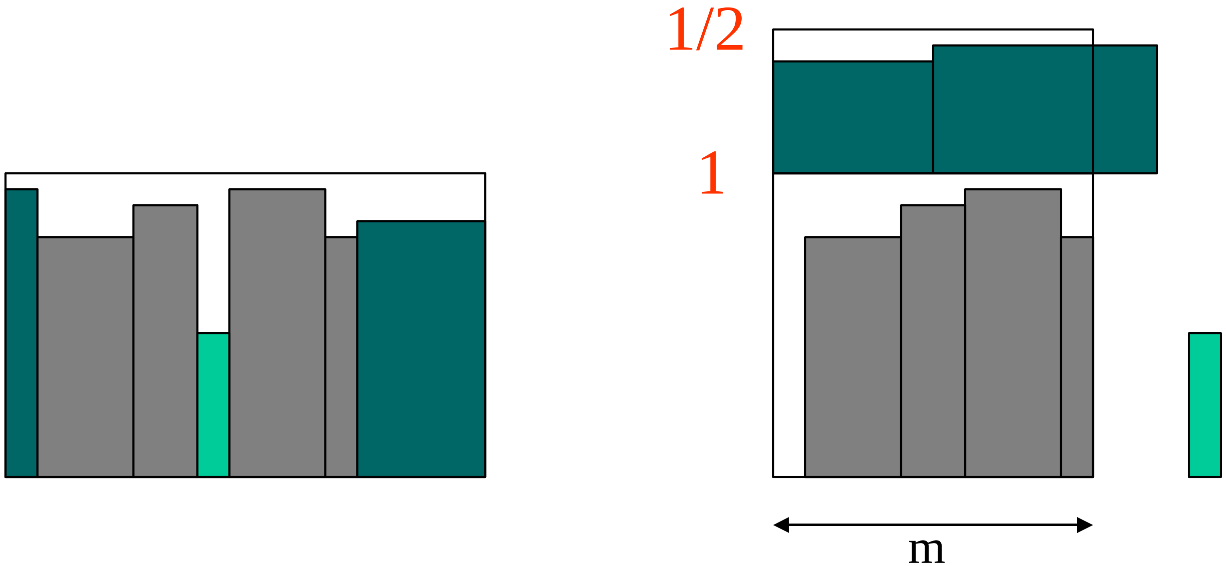
Maximal number of processors needed for executing the tasks in time lower than 1.

2-shelves scheduling

Idea: to analyze the structure of the optimum where the tasks are either greater than 1/2 or not.

Thus, we will try to fill two shelves with these tasks.

2 shelves partitioning



Knapsack problem: minimizing the global surface under the constraint of using less than m processors in the first shelf.

Dynamic programming

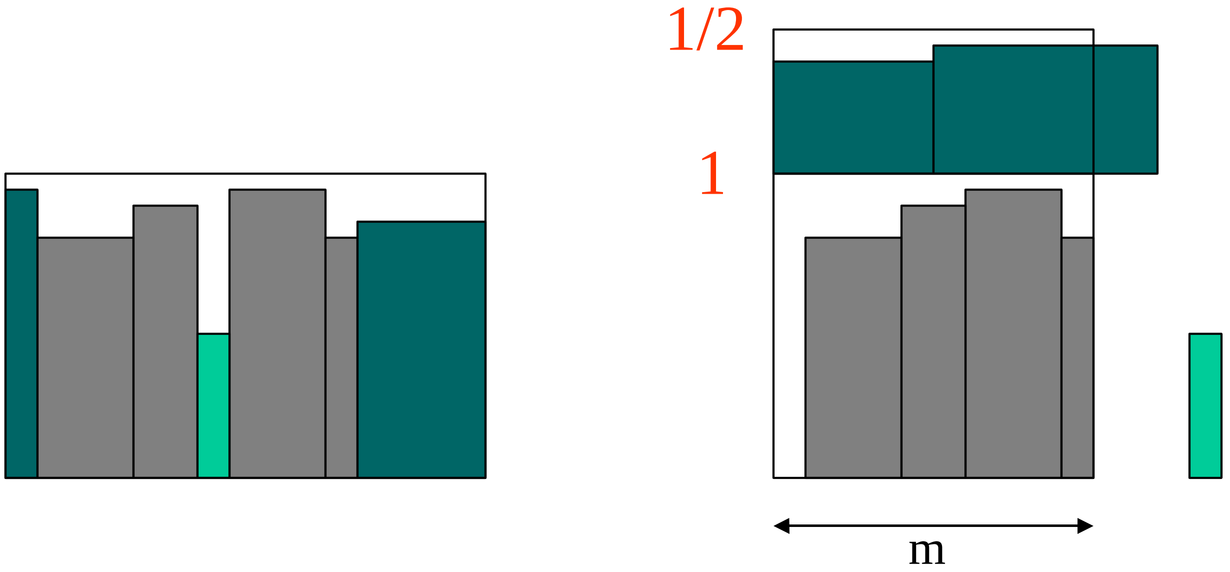
For i = 1..n // # of tasks

- for j = 1..m // #proc.
  - Wi,j = min(
    - – Wi,j-minalloc(i,1) + work(i,minalloc(i,1))
    - – Wi,j + work(i,minalloc(i,1))
    -)

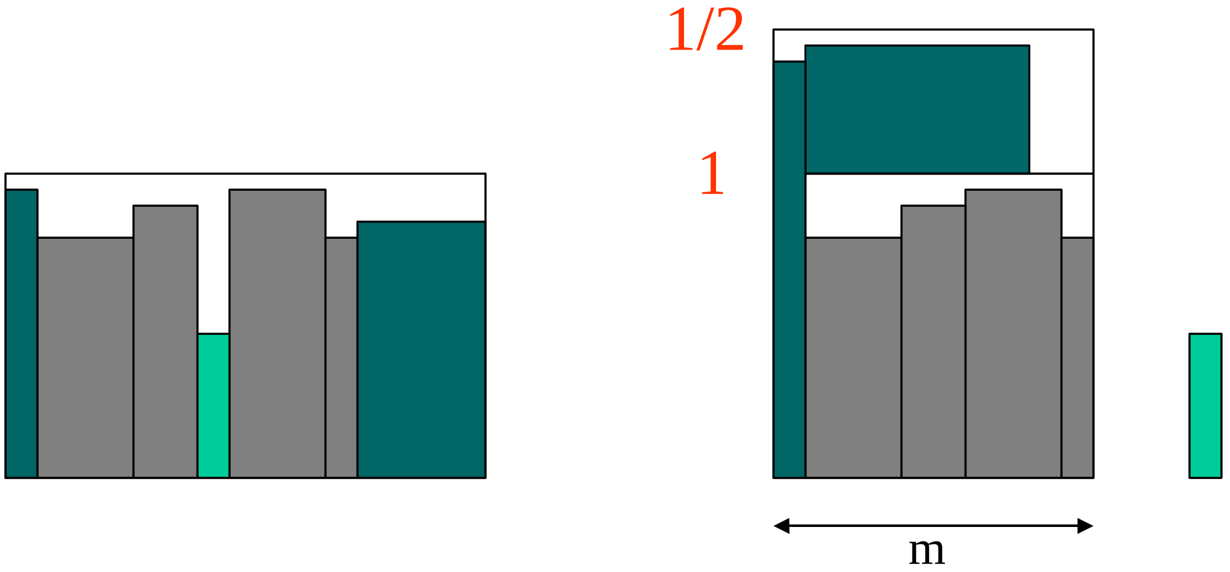
work Wn,m

- <= work of an optimal solution
- but the half-sized shelf may be overloaded

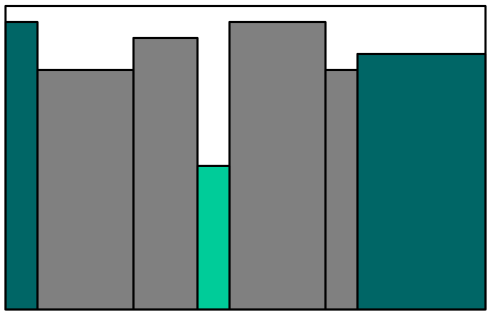
2 shelves partitioning

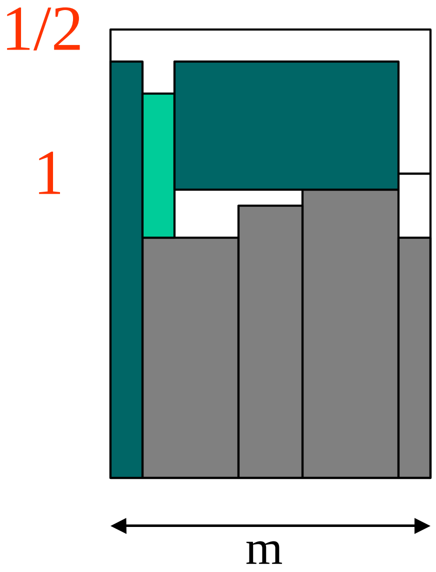


Drop down



Insertion of small tasks





Analysis

- These transformations donot increase the work
- If the 2nd shelf is used more than m, it is always possible to do one of the transformations (using a global surface argument)
- It is always possible to insert the « small » sequential tasks (again by a surface argument)

Guaranty

- The 2-shelves algorithm has a performance guaranty of $3/2+\varepsilon$

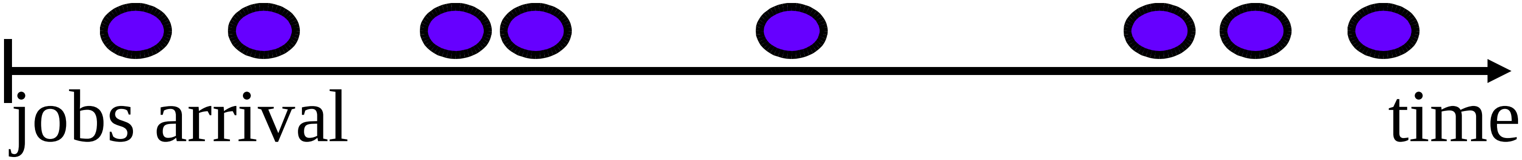
(SIAM J. on Computing, to appear)

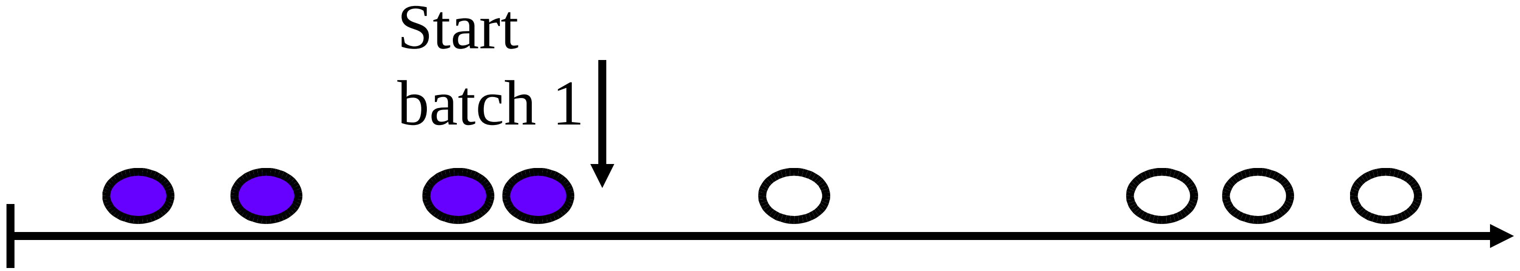
- Rigid case: 2-approximation algorithm (Graham resource constraints)

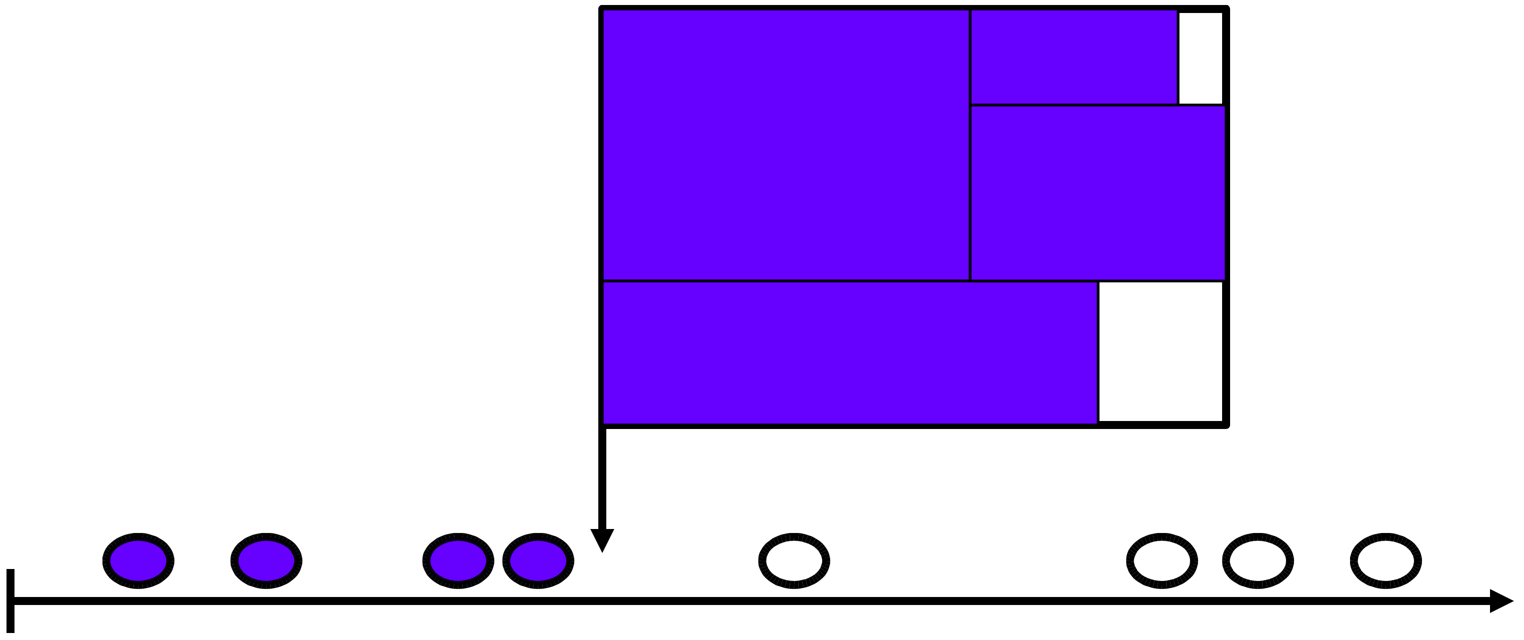
Batch scheduling

Principle: several jobs are treated at once using off-line scheduling.

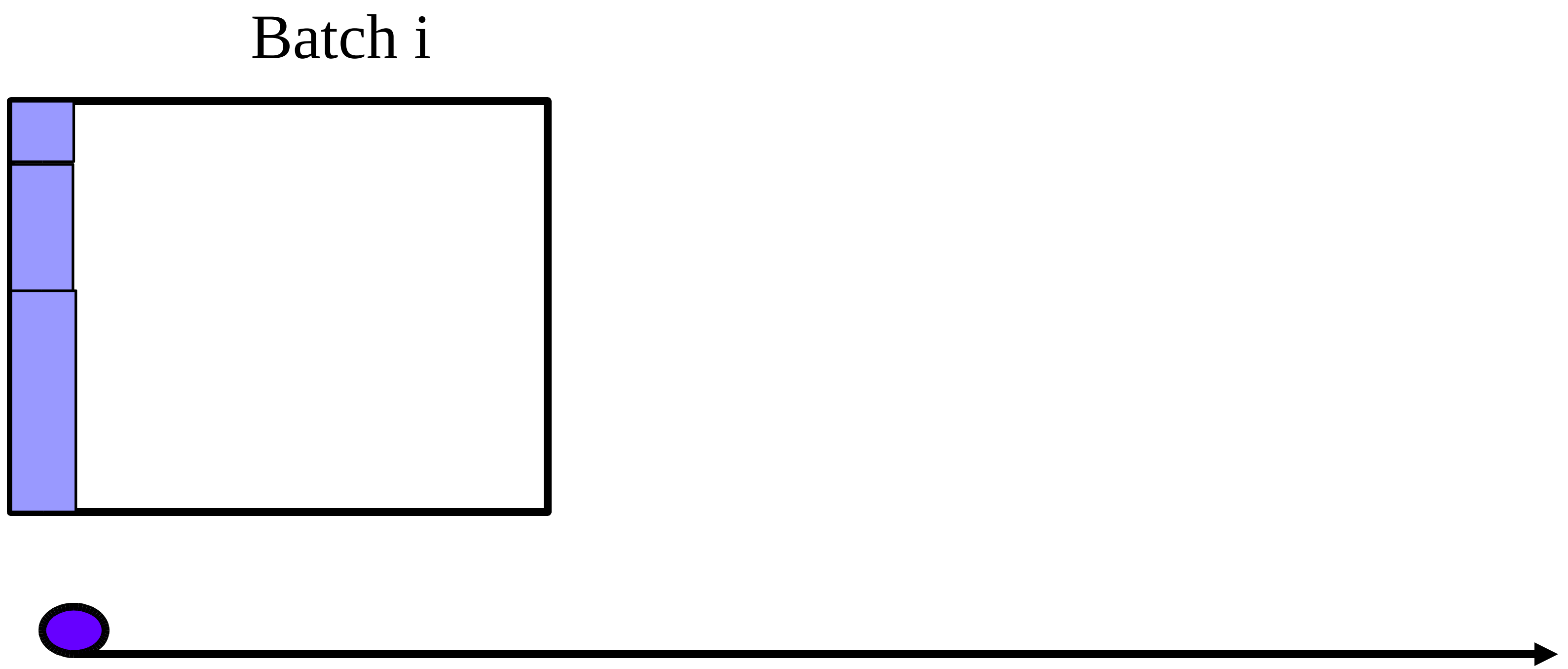
Principle of batch



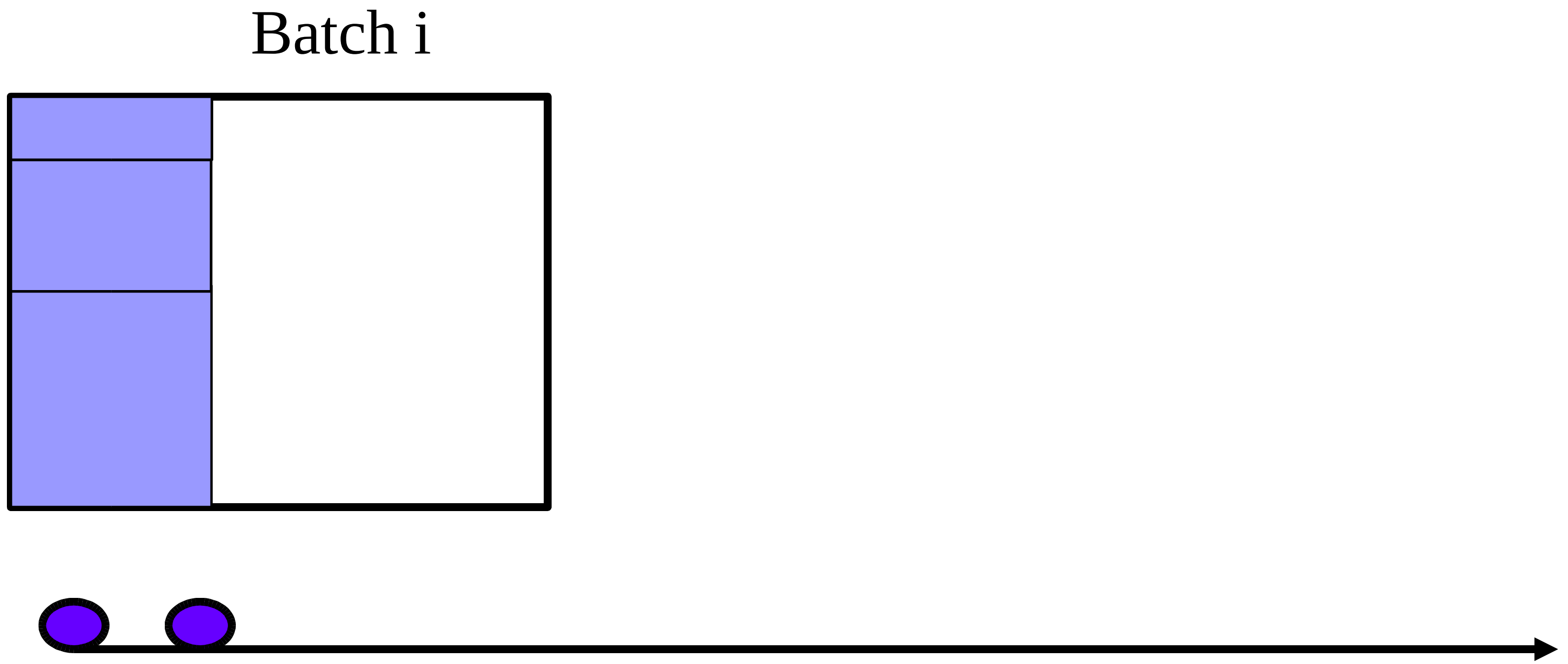

Start
batch 1



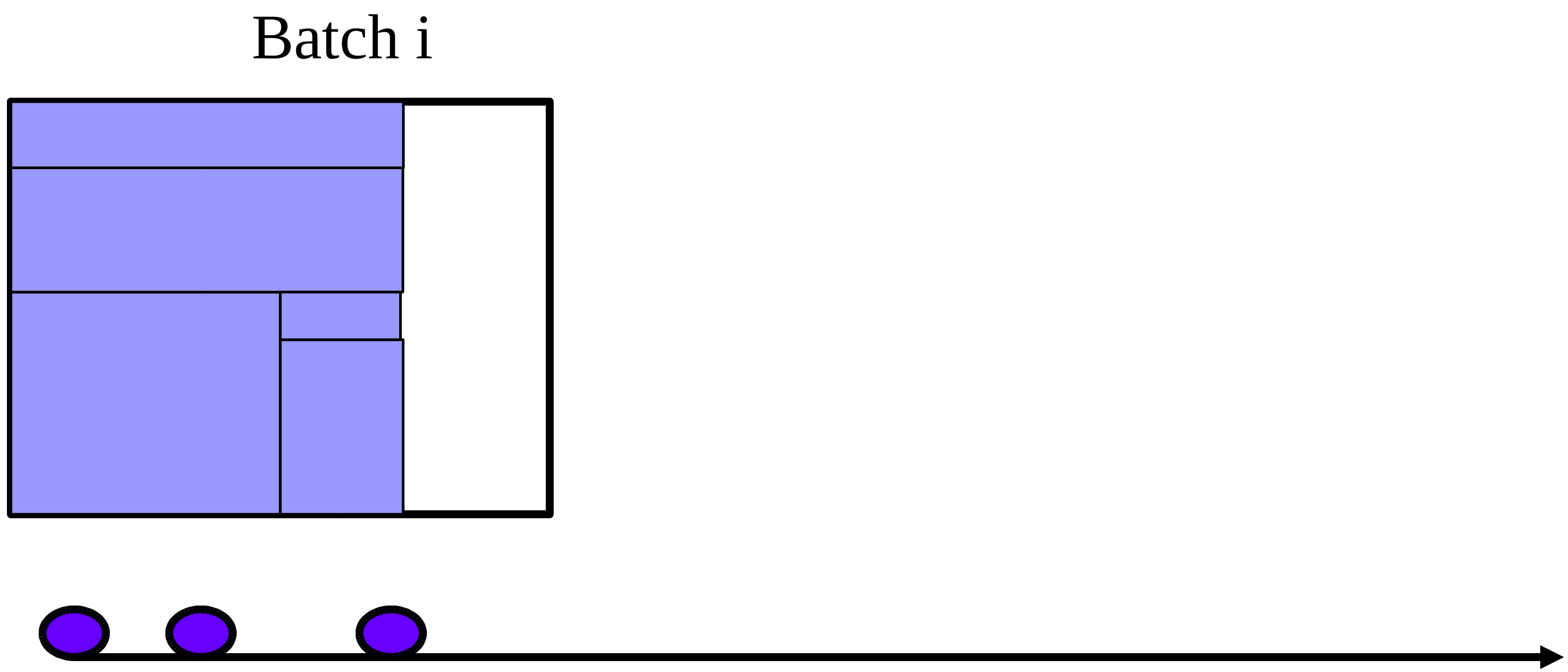
Batch chaining



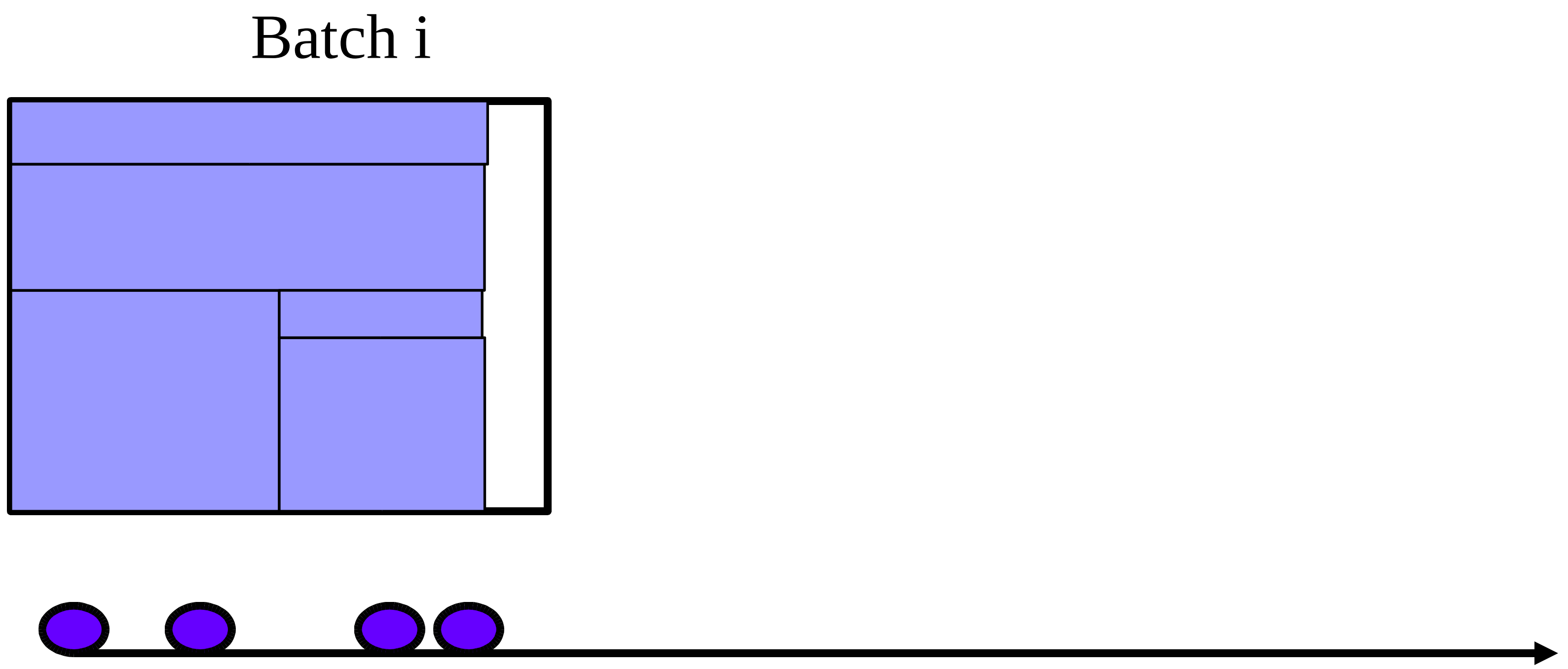
Batch chaining



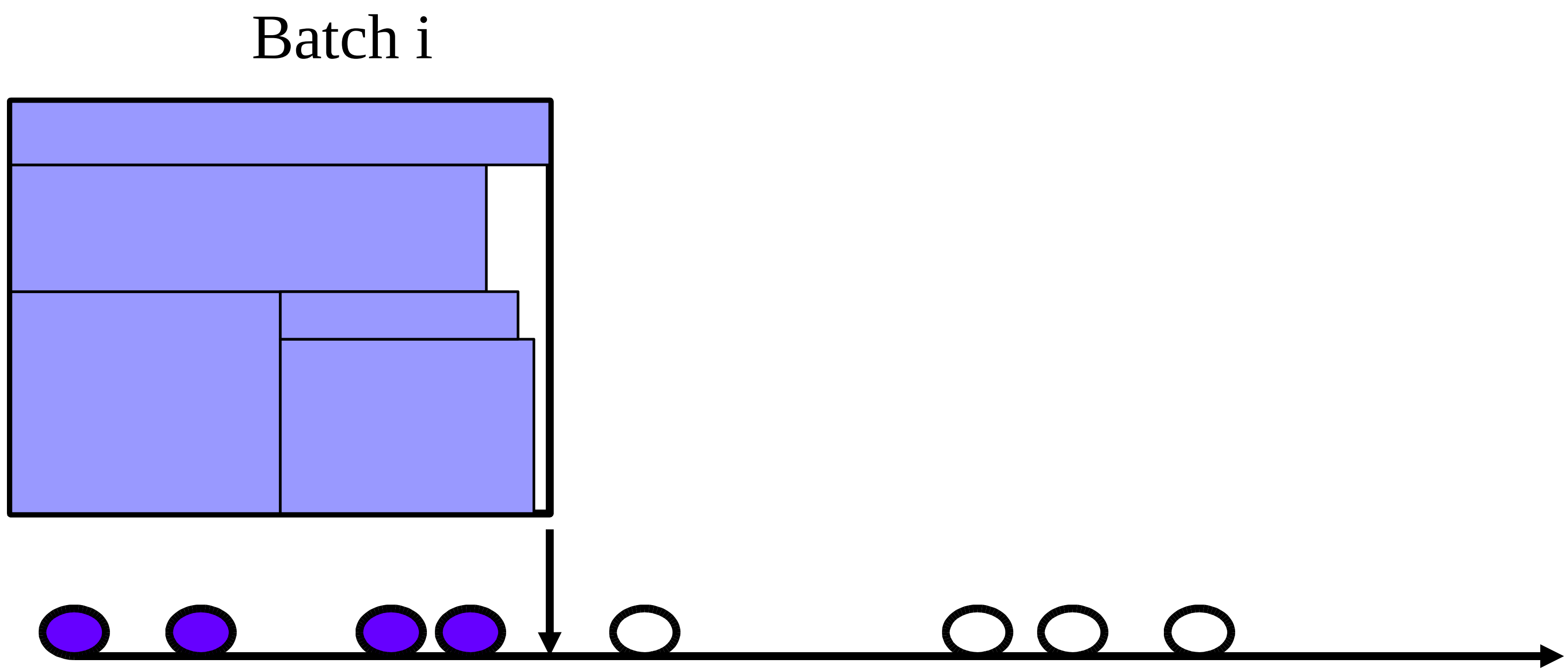
Batch chaining



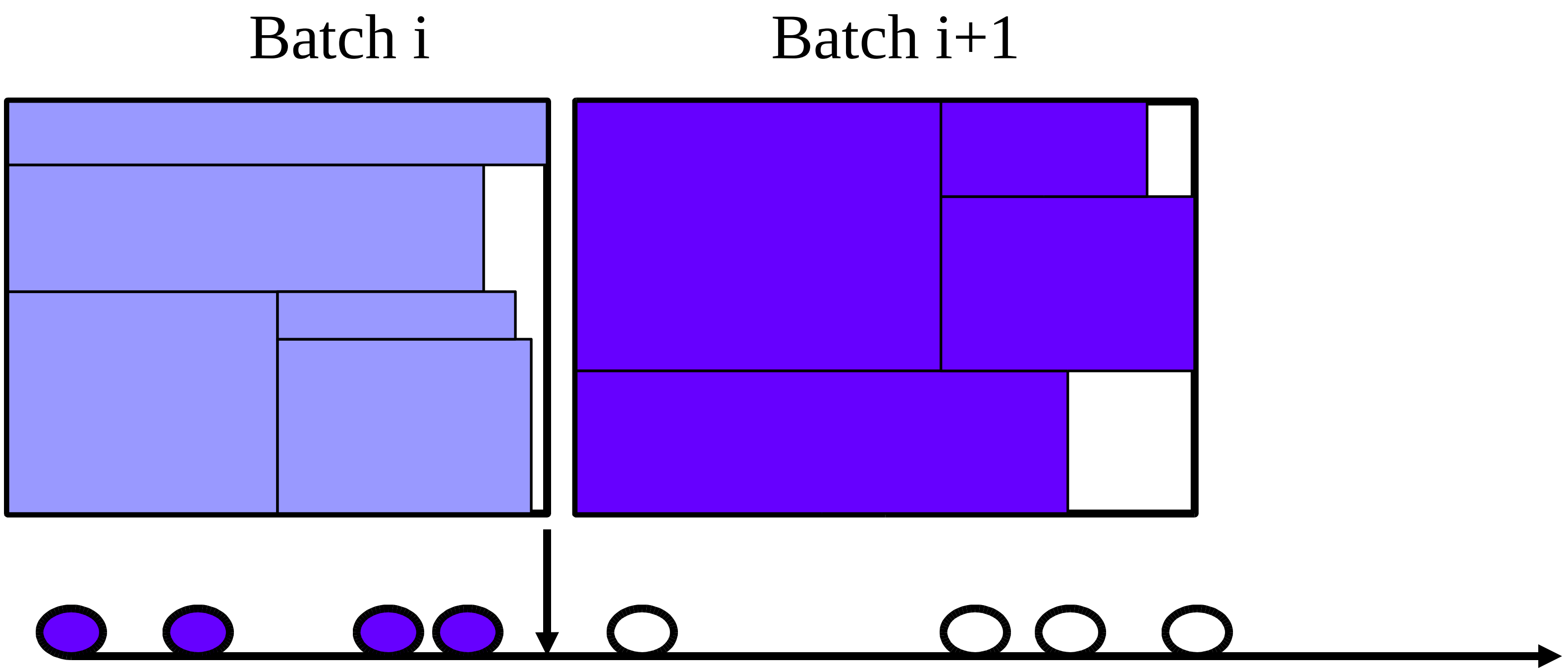
Batch chaining



Batch chaining



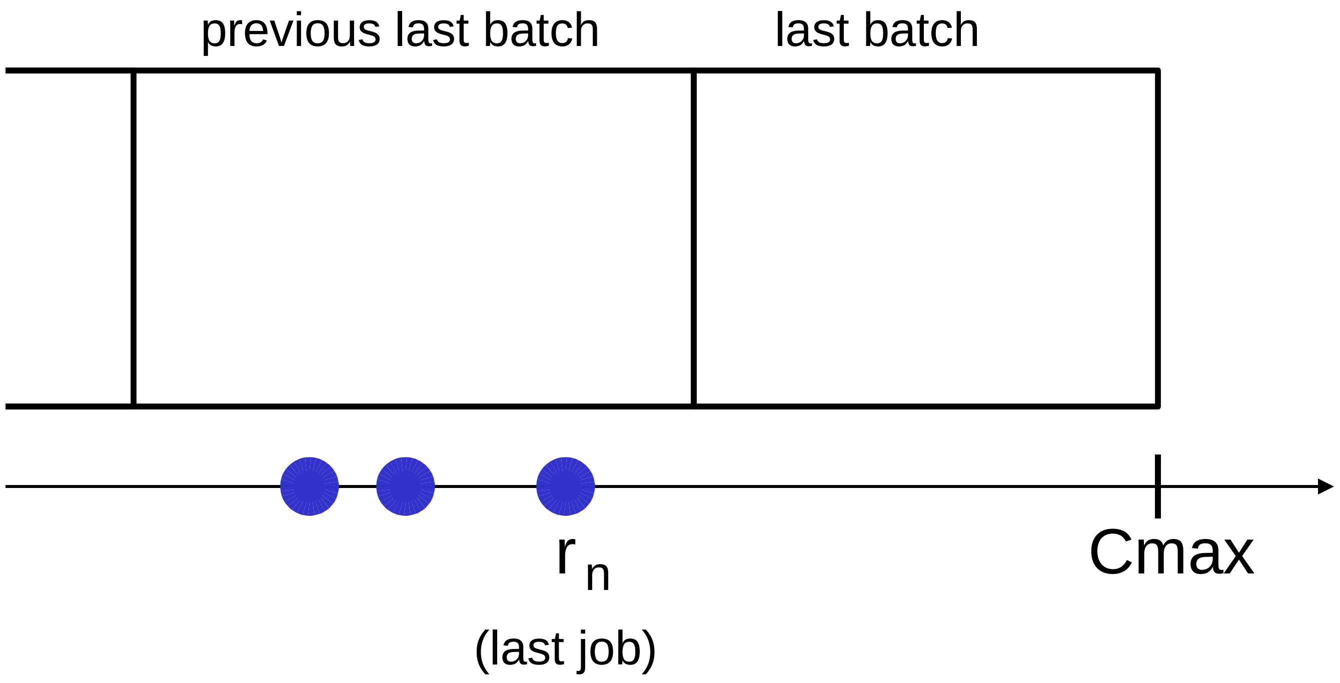
Batch chaining

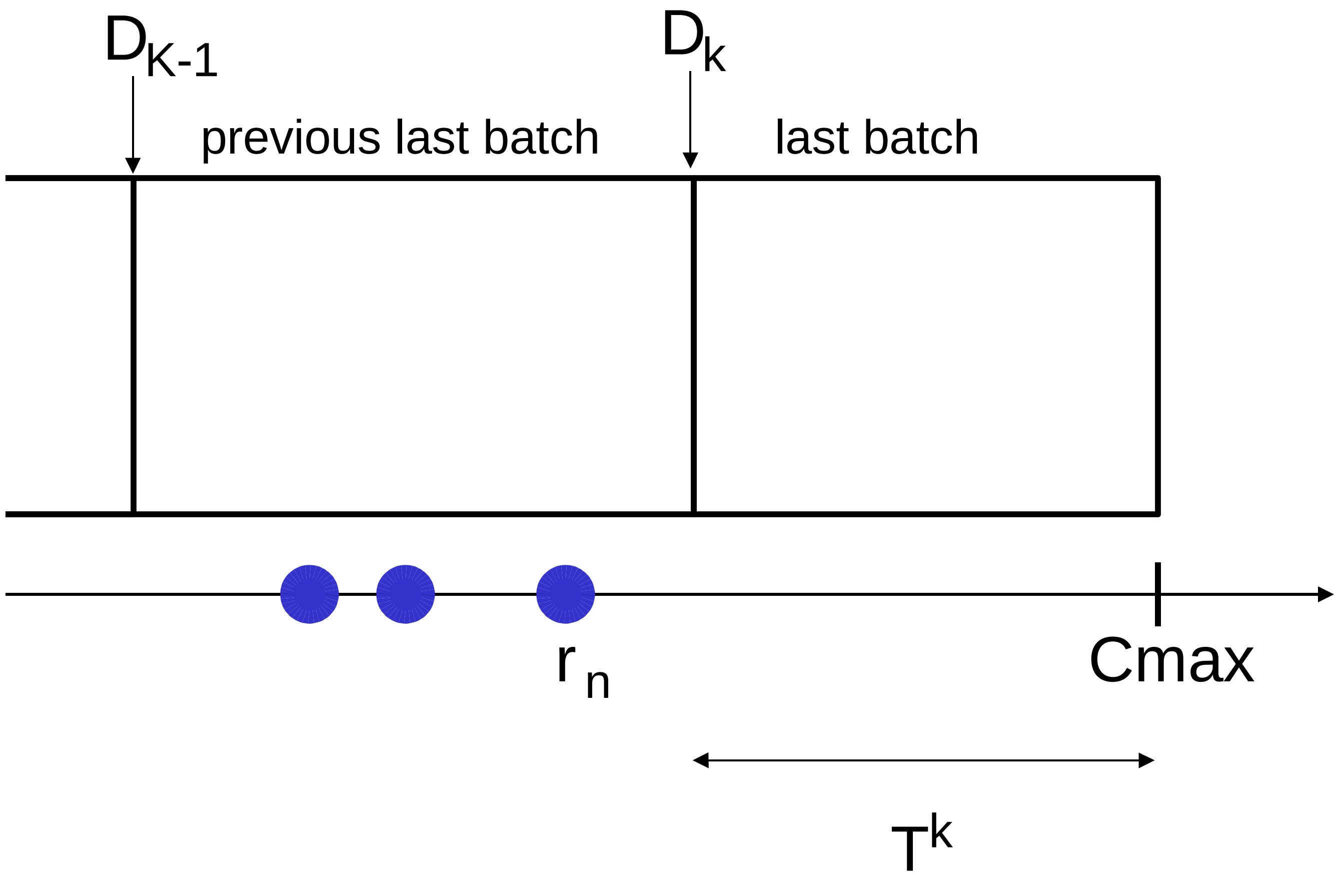


Constructing a batch scheduling

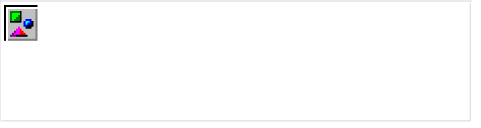
Analysis: there exists a nice (simple) result which gives a guaranty for an execution in batch mode using the guaranty of the off-line scheduling policy inside the batches.

Analysis [Shmoys]



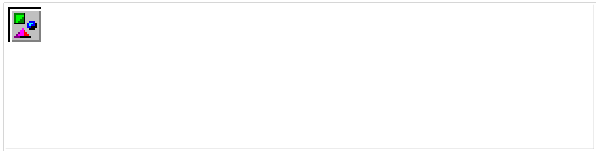

D_{K-1}
D_k
previous last batch
last batch
r_n
Cmax
T^k

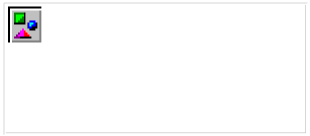
Proposition

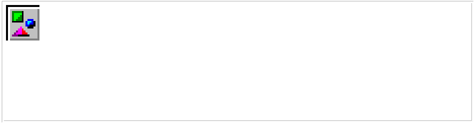


Analysis

Tk is the duration of the last batch



On another hand,  and 

Thus: 

Application

Applied to the best off-line algorithm for moldable jobs (3/2-approximation), we obtain a 3-approximation on-line batch algorithm for Cmax.

This result holds also for rigid jobs (using the 2-approximation Graham resource constraints), leading to a 4-approximation algorithm.

Multi criteria

Cmax is not always the adequate criterion.

User point of view:

Average completion time (weighted or not)

Other criteria:

Stretch, Asymptotic throughput, fairness, …

How to deal with this problem?

Hierachal approach: one criterion after the other

(Convex) combination of criteria

Transforming one criterion in a constraint

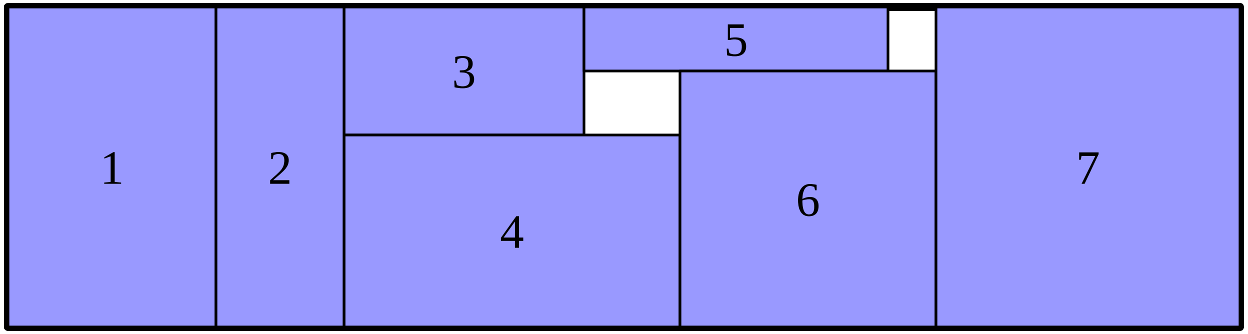
Better - but harder - ad hoc algorithms

A first solution

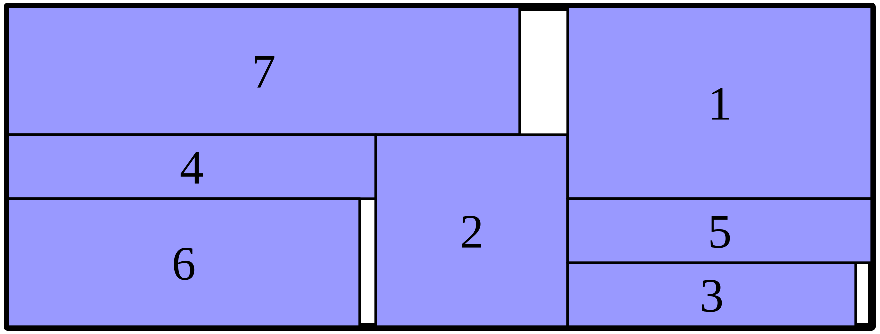
Construct a feasible schedule from two schedules of guaranty r for minsum and r’ for makespan with a guaranty (2r,2r’) [Stein et al.].

Instance: 7 jobs (moldable tasks) to be scheduled on 5 processors.

Schedules s and s’

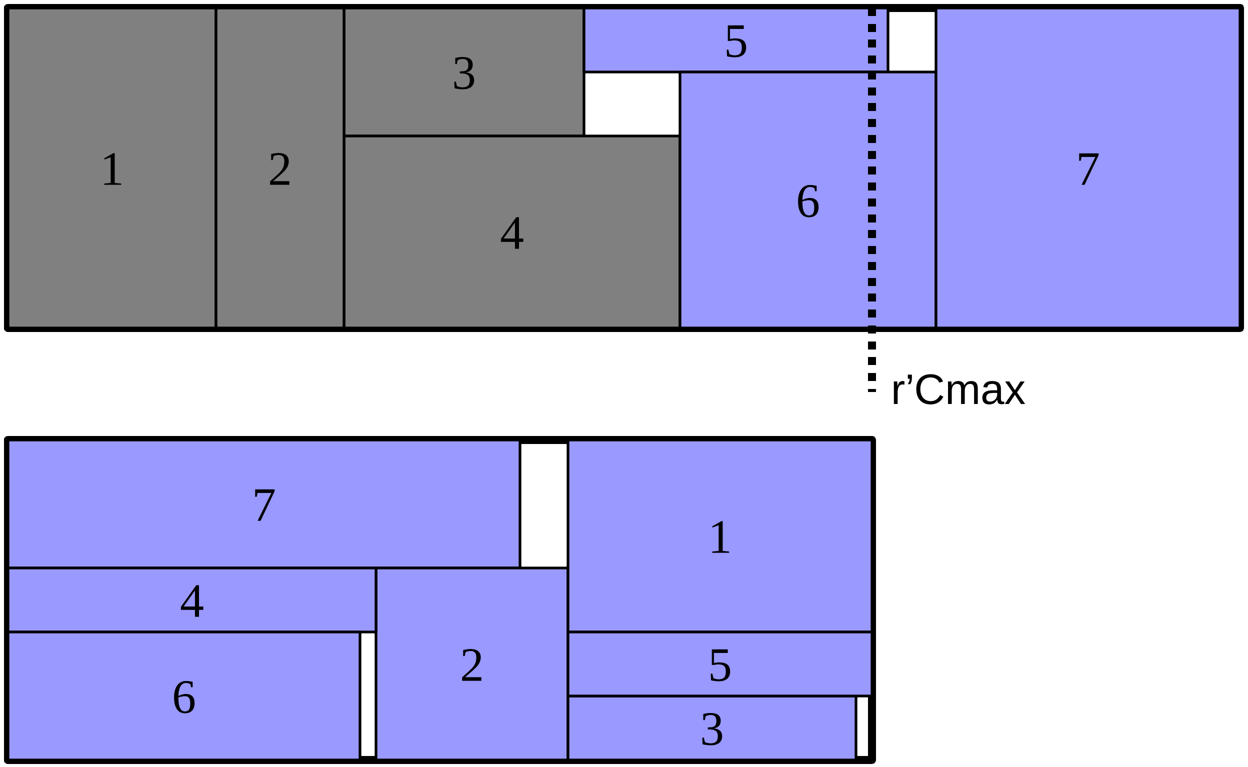


Schedule s
(minsum)

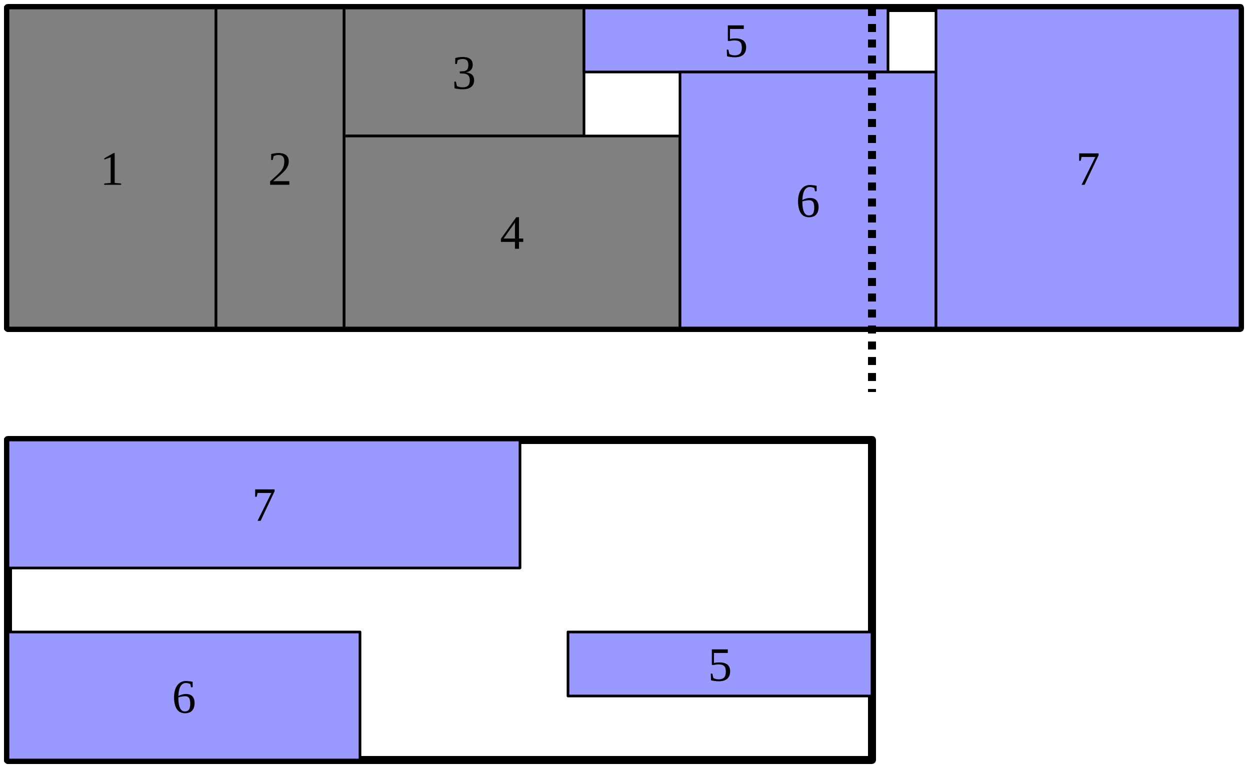


Schedule s’
(makespan)

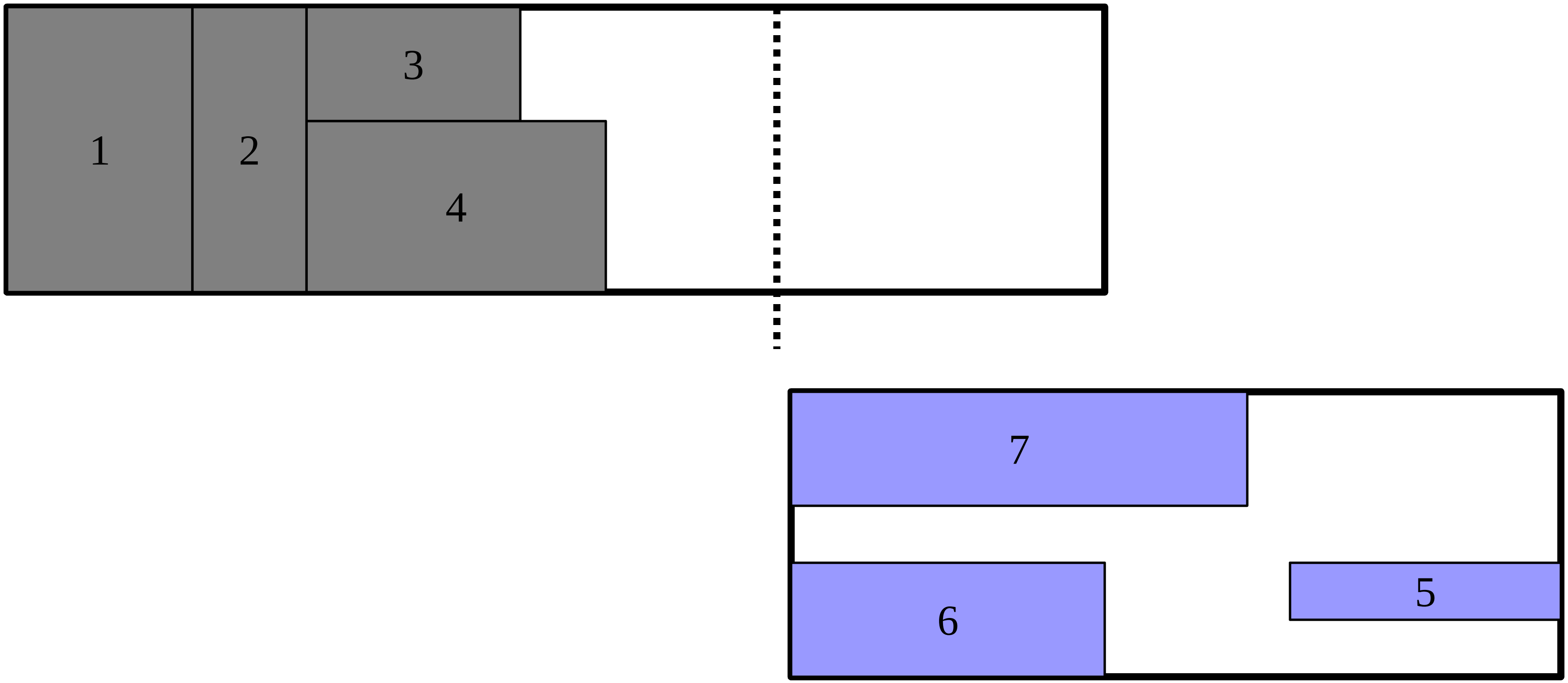
New schedule



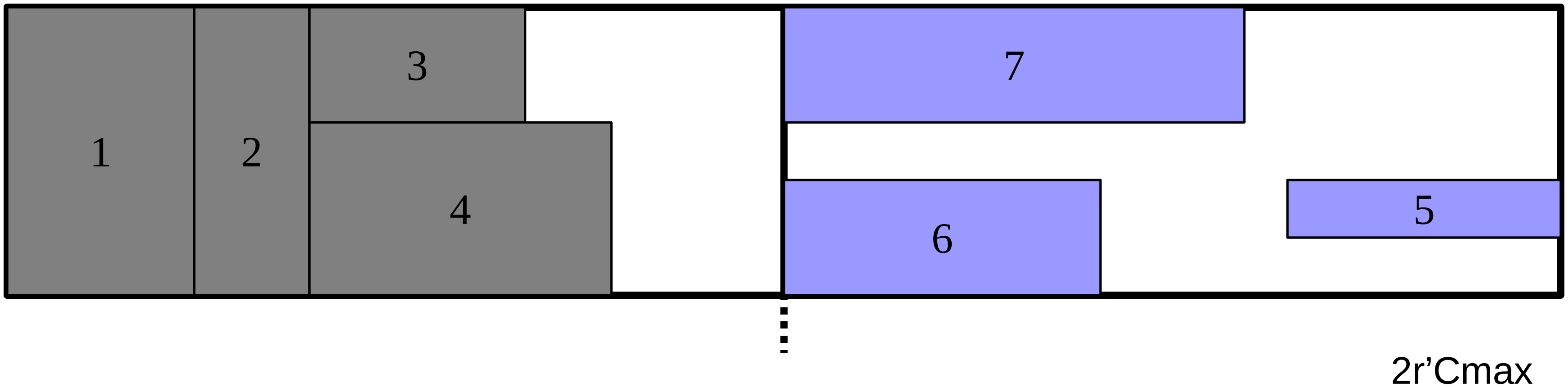
New schedule



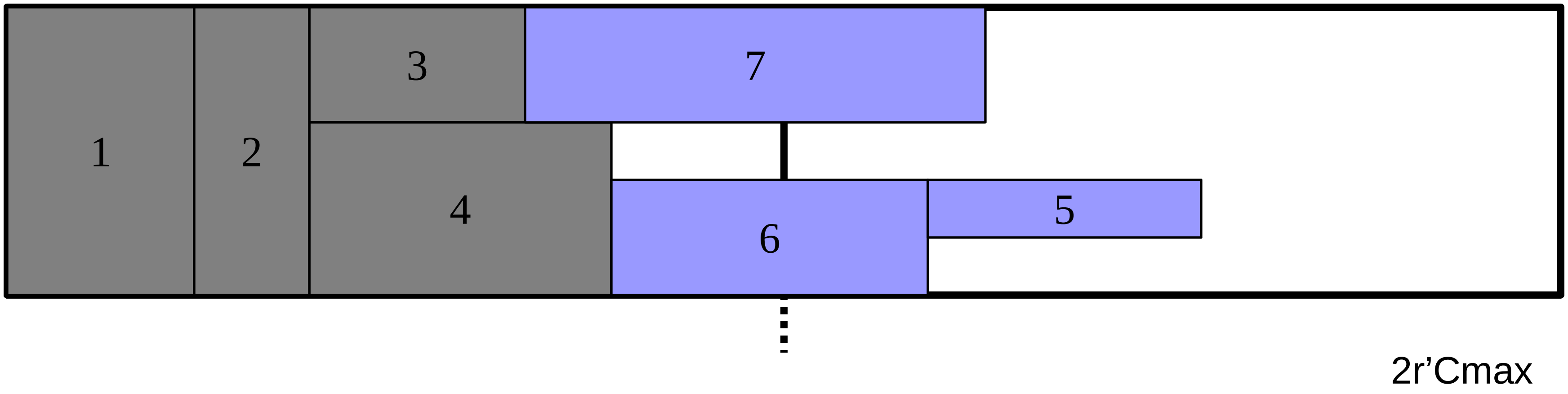
New schedule



New schedule



New schedule



Similar bound for the first criterion

Analysis

The best known schedules are:
8 [Schwiegelsohn] for minsum and 3/2 [Mounie et al.] for makespan leading to (16;3).

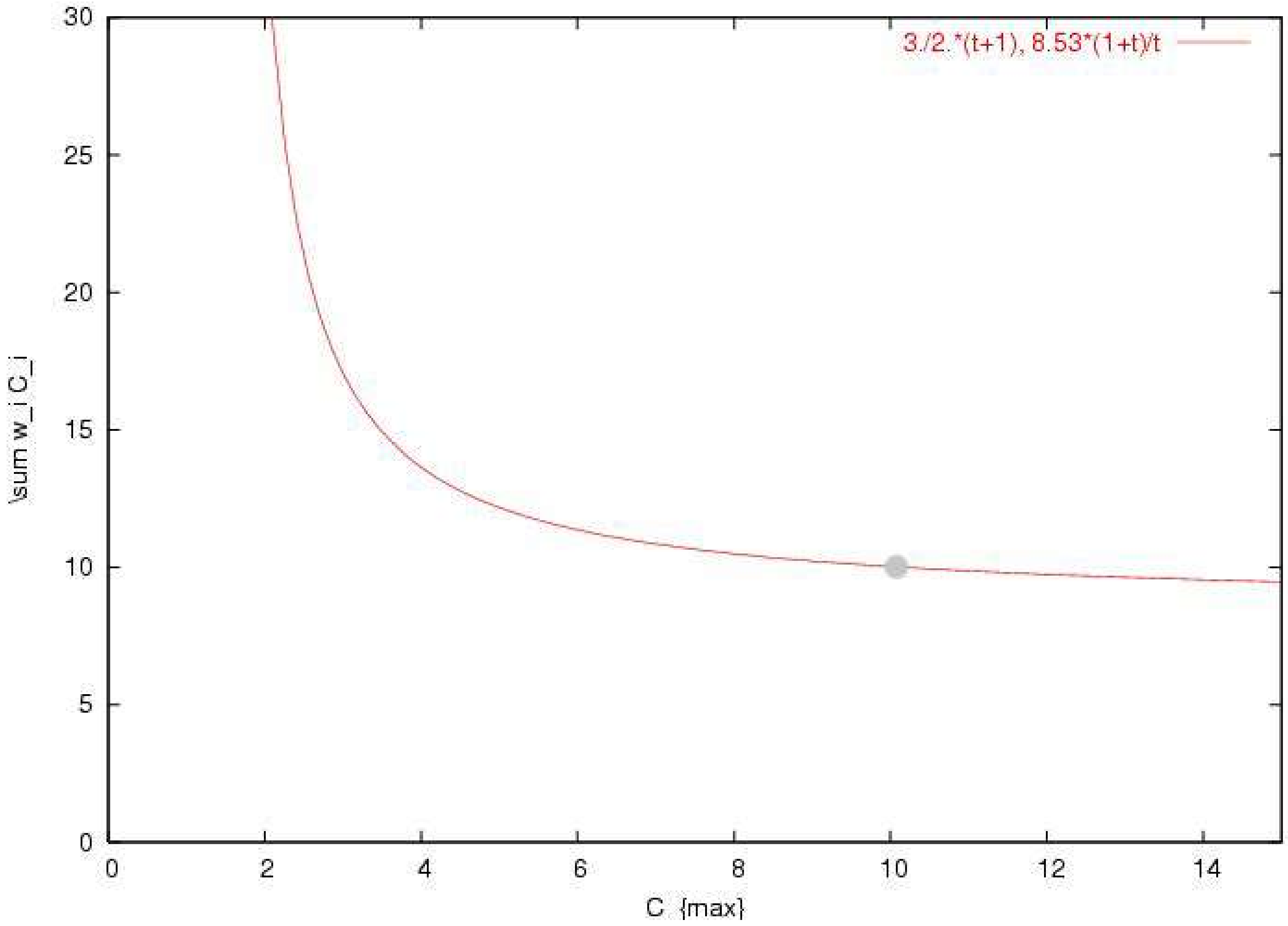
Similarly for the weighted minsum (ratio 8.53 for minsum).

Improvement

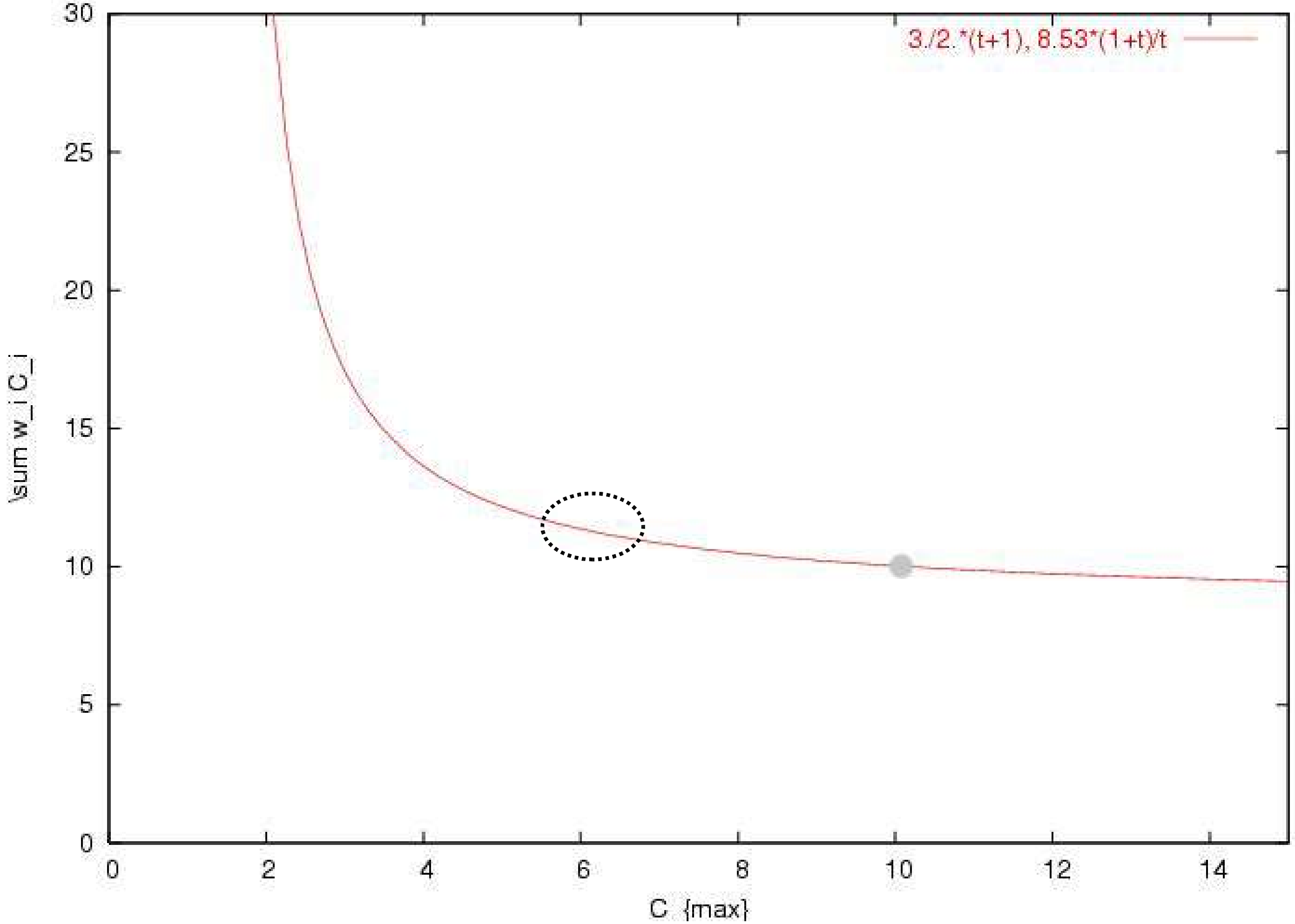
We can improve this result by determining the Pareto curves (of the best compromises):
$(1+\lambda)/\lambda\, r$ and $(1+\lambda)r'$

Idea:
take the first part of schedule s up to $\lambda\, r' C_{max}$

Pareto curve



Pareto curve



Another way for designing better schedules

We proposed [SPAA'2005] a new solution for a better bound which has not to consider explicitly the schedule for minsum (based on a dynamic framework).

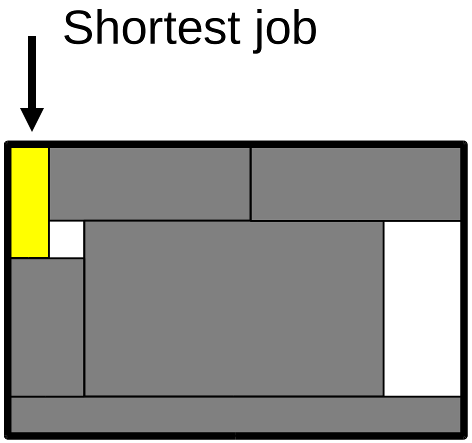
Principle: recursive doubling with smart selection (using a knapsack) inside each interval.
Starting from the previous algorithm for Cmax, we obtain a (6;6) approximation.

Bi criteria: C_{max} and $\Sigma_{w_i C_i}$

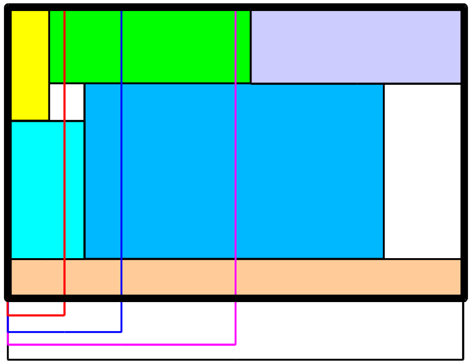
- Generic On-line Framework [Shmoys et al.]
  - Exponantially increasing time intervals
  - Uses a max-weight ρ approximation algorithm
    - If the optimal schedule of length *d* has weight *w**, provides a schedule of length ρ*d* and weight ≥ *w**
  - Yields a (4ρ, 4ρ) approximation algorithm
    - For moldable tasks, yields a (12, 12) approximation
    - With the 2-shelf algorithm, yields a (6, 6) approximation [Dutot et al.]

Example for $\rho = 2$

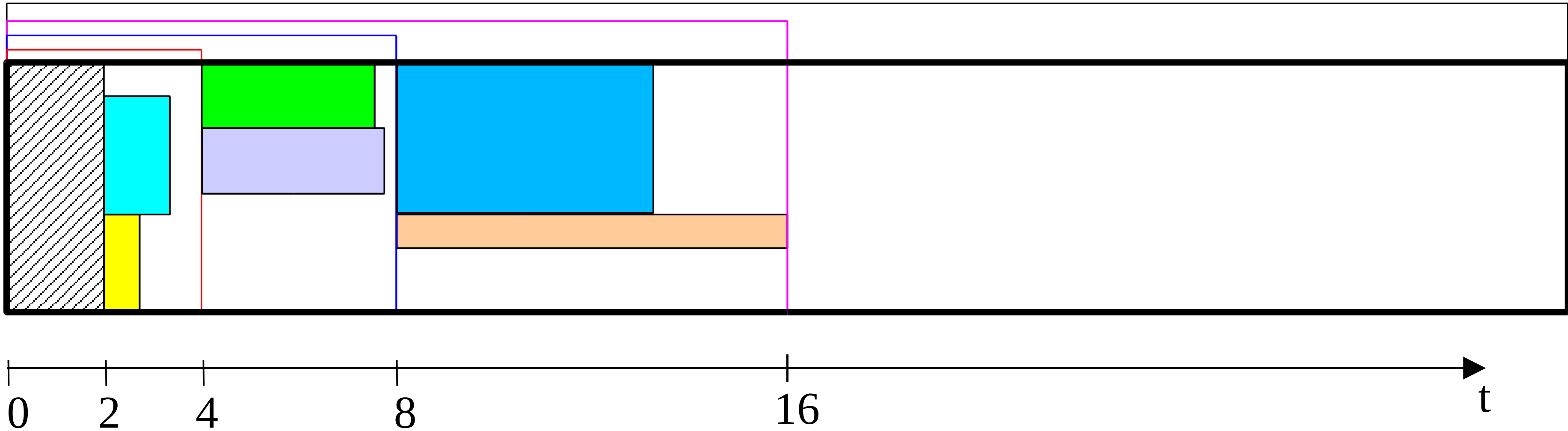
Schedule for makespan



Example for ρ = 2



→ "Contains more weight"



A last trick

The intervals are shaked (like in 2-opt local optimization techniques).

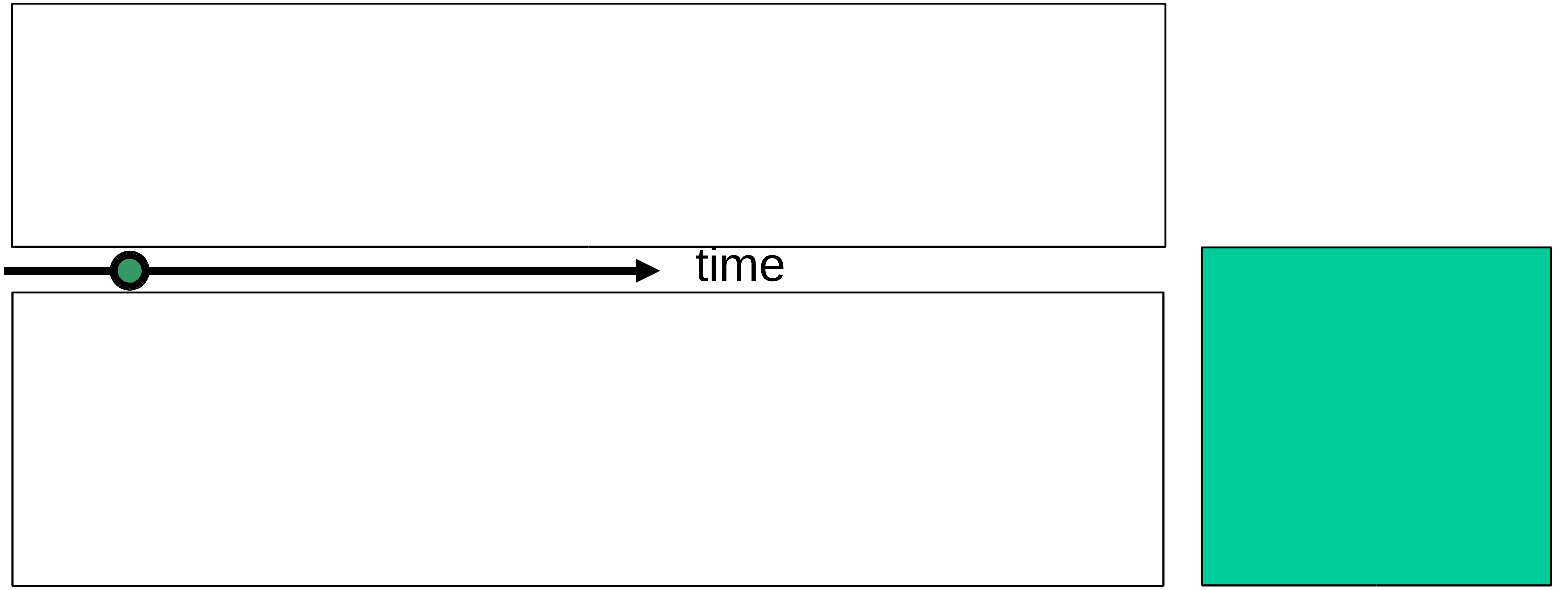
This algorithm has been adapted for rigid tasks.

It is quite good in practice, but there is no theoretical guaranty…

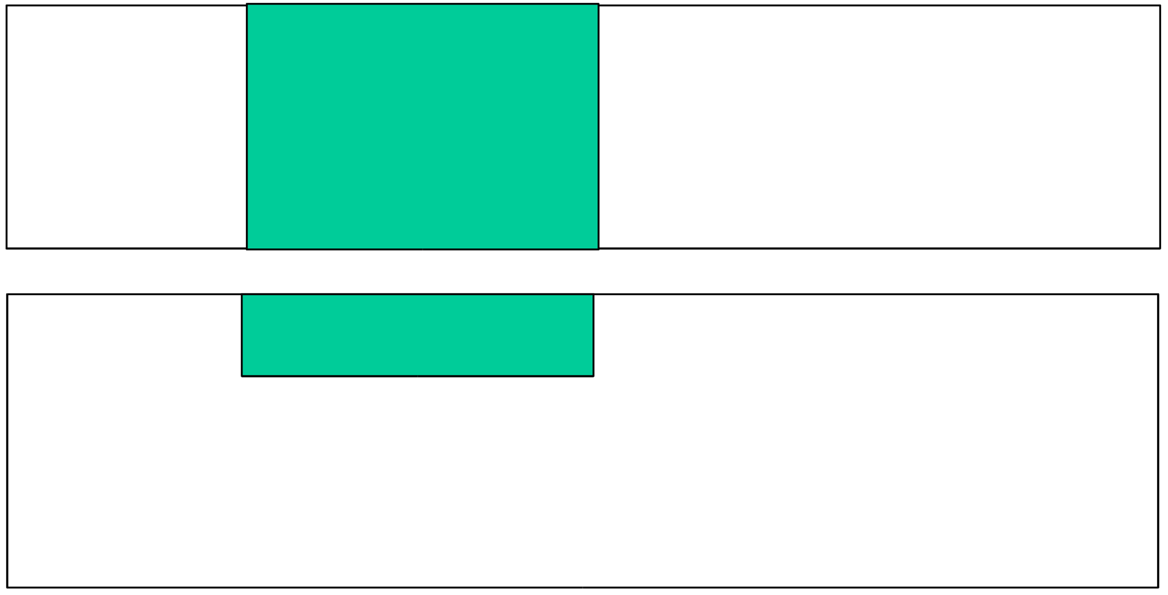
Reservations

Motivation:

Execute large jobs that require more than m processors.

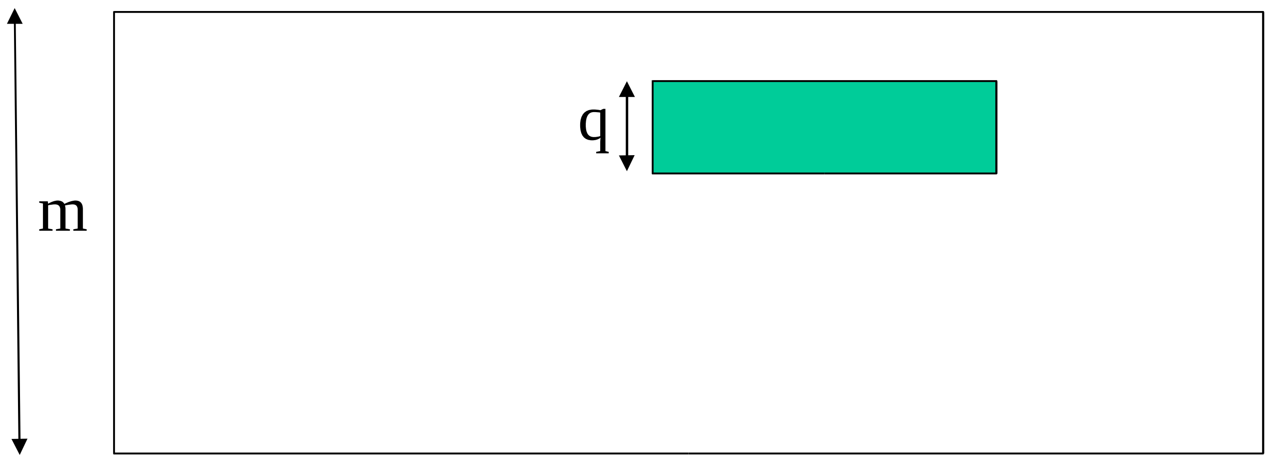


Reservations



Reservations

The problem is to schedule n independent parallel rigid tasks such that the last finishing time is minimum.



At each time t, $r(t) \leq m$ processors are not available

State of the art

Most existing results deal with sequential tasks (qj=1).

Without preemption:

Decreasing reservations

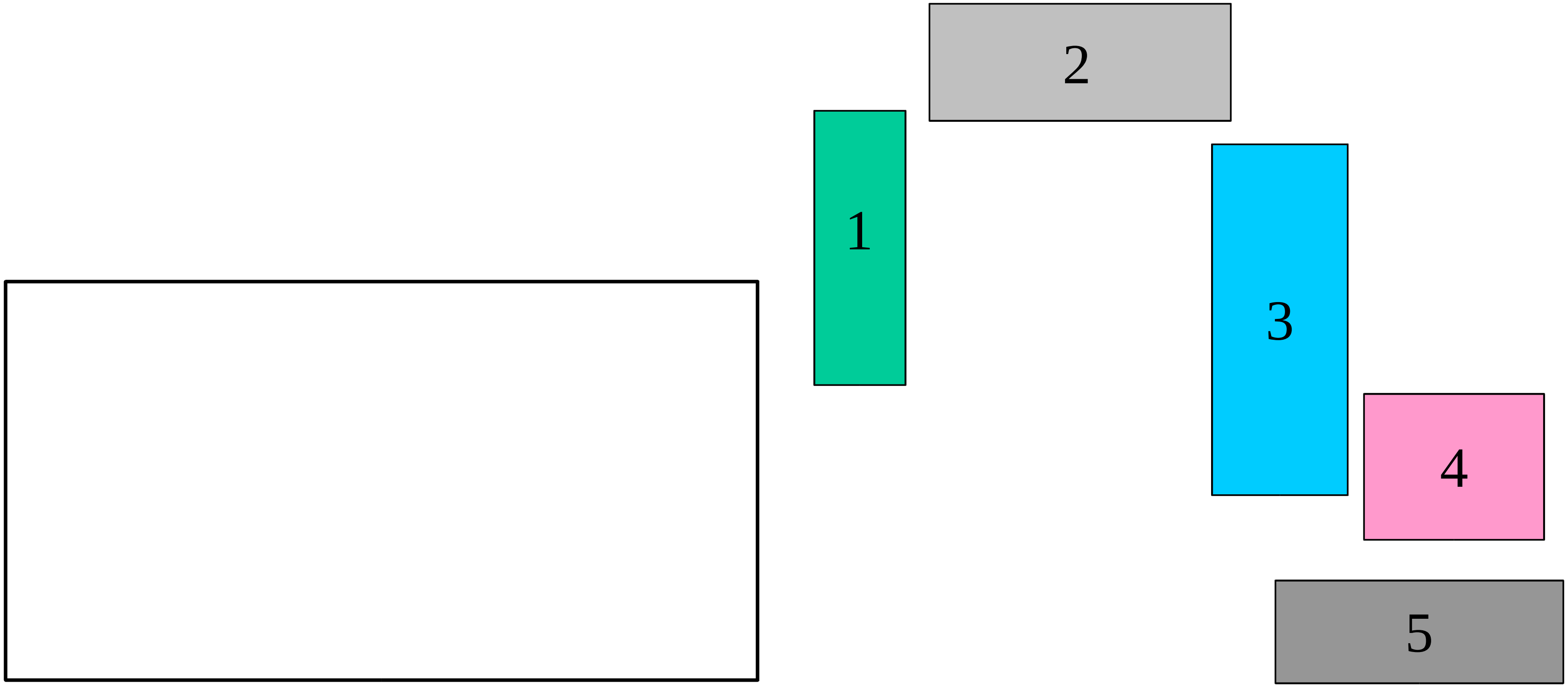
Only one reservation per machine

With preemption:

Optimal algorithms for independent tasks

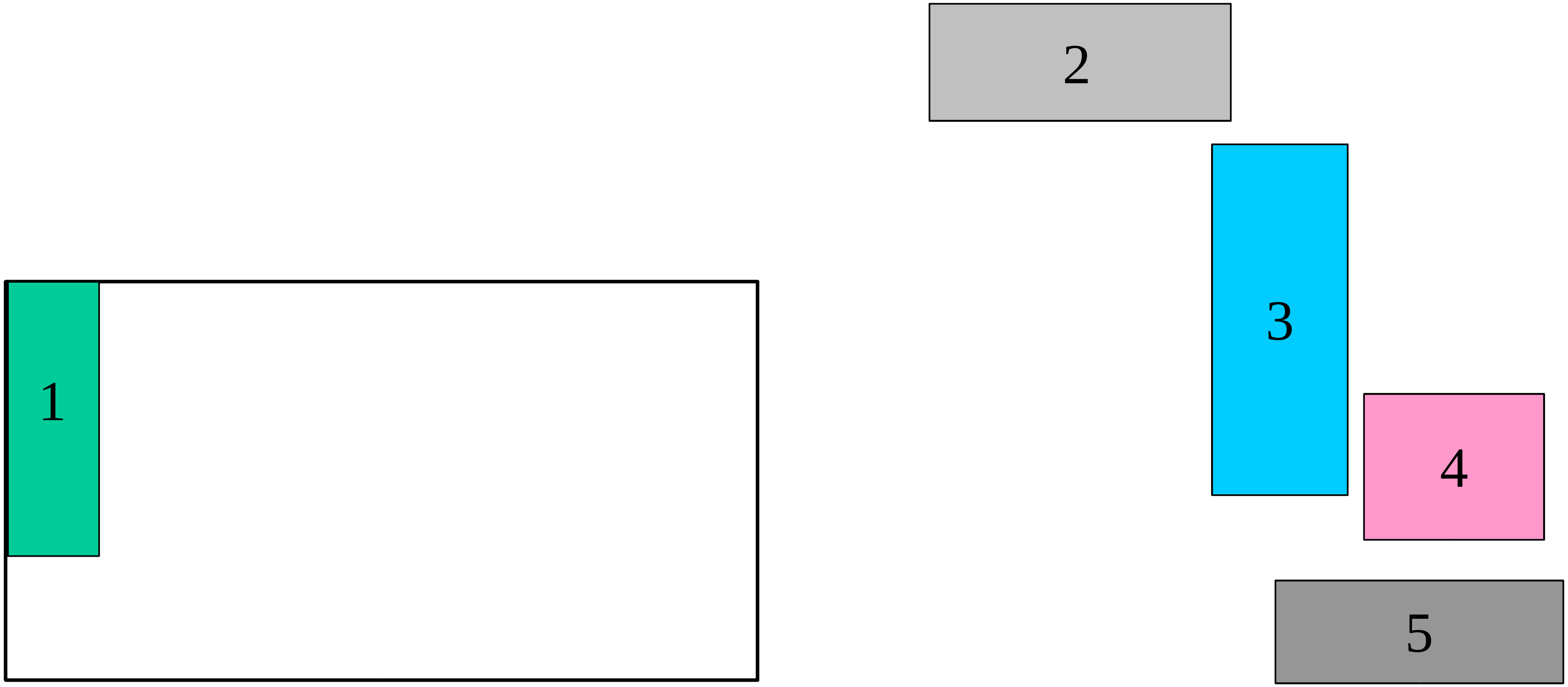
Optimal algorithms for some simple task graphs

Without reservation



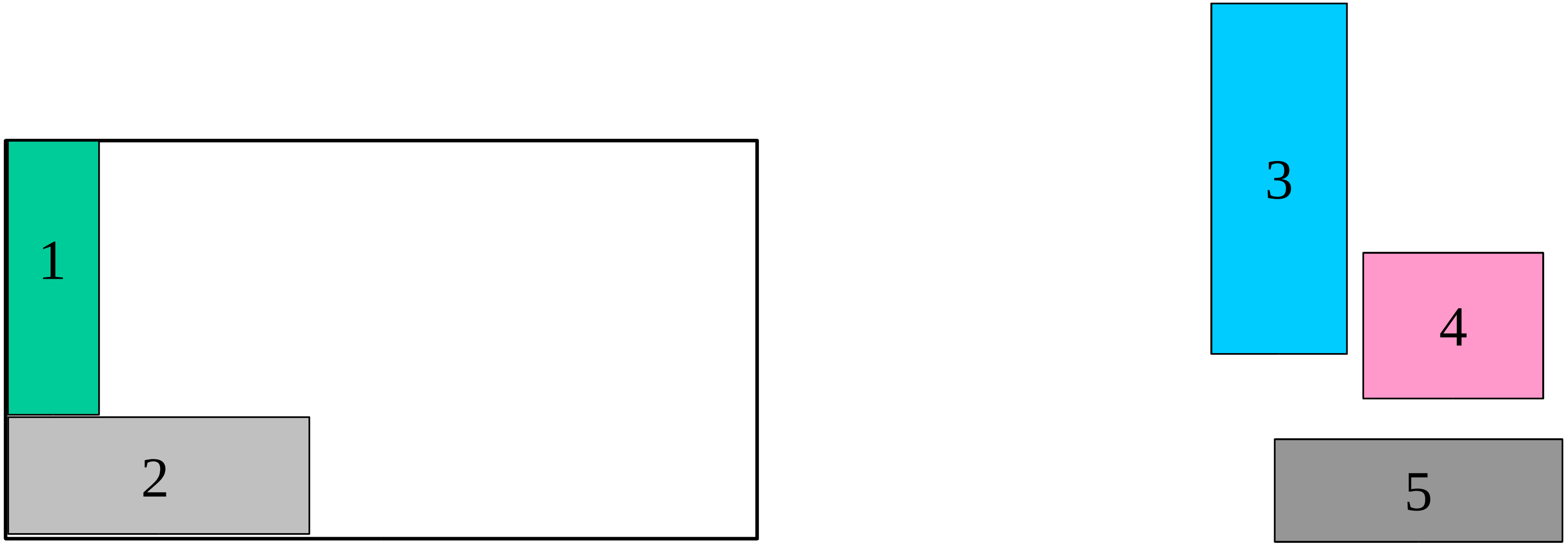
FCFS with backfilling

Without reservation



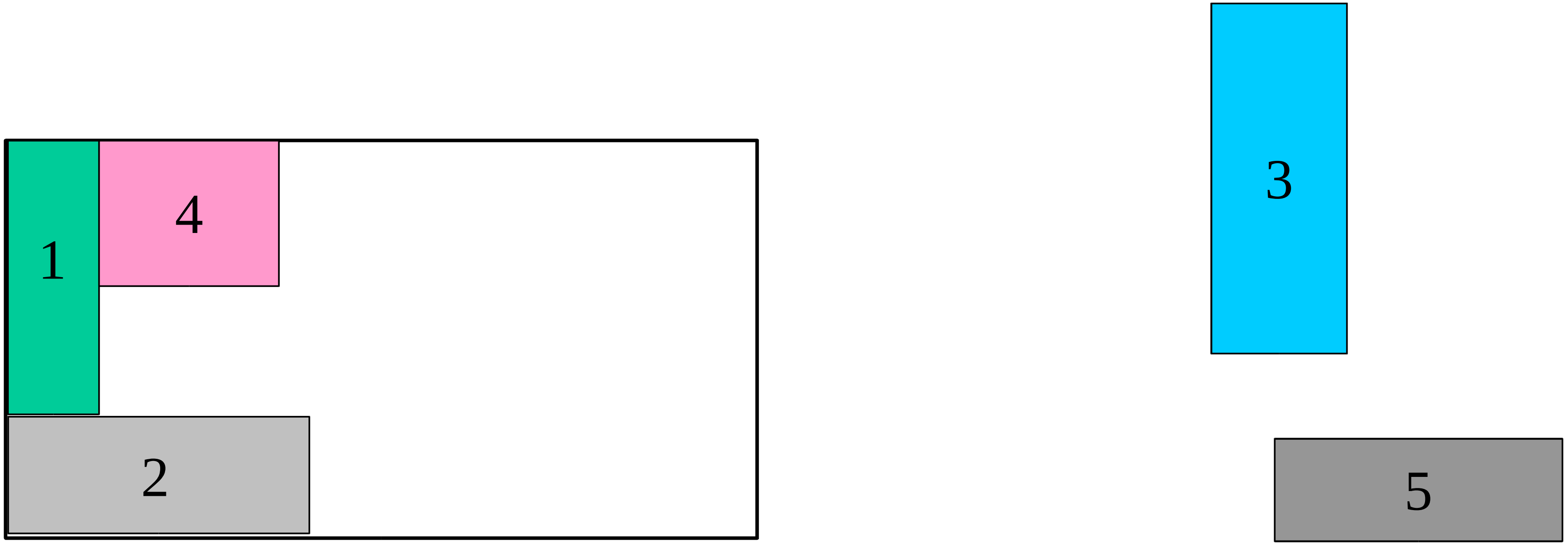
FCFS with backfilling

Without reservation



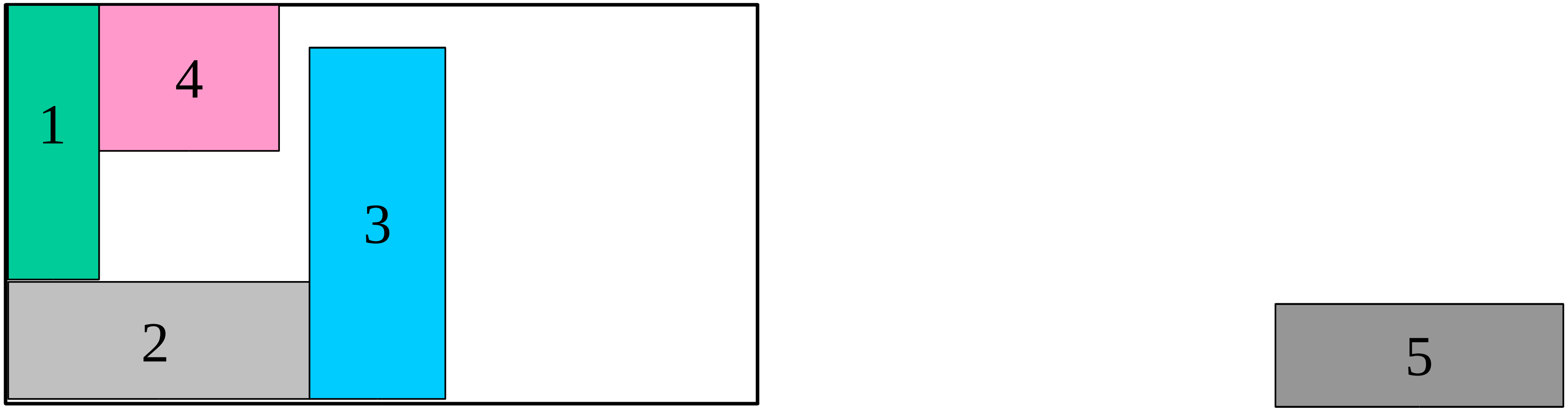
FCFS with backfilling

Without reservation



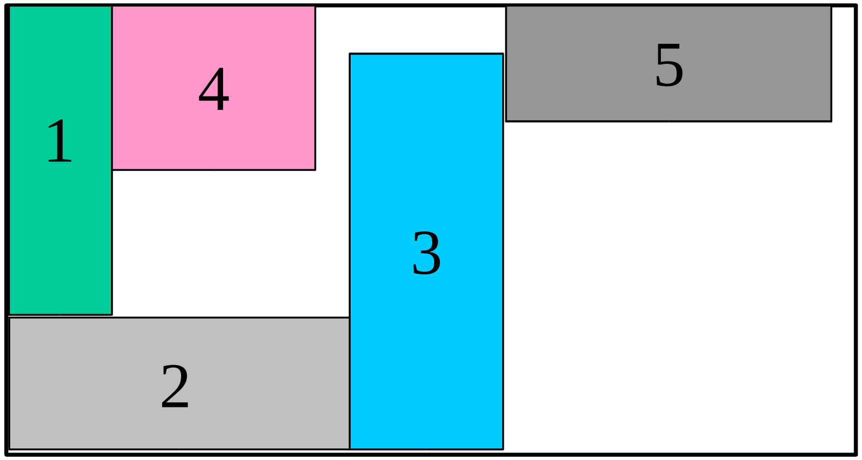
FCFS with backfilling

Without reservation



FCFS with backfilling

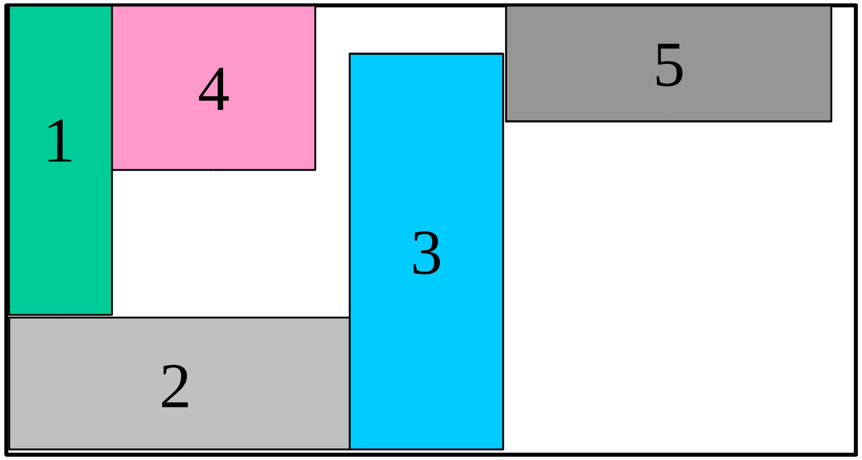
Without reservation



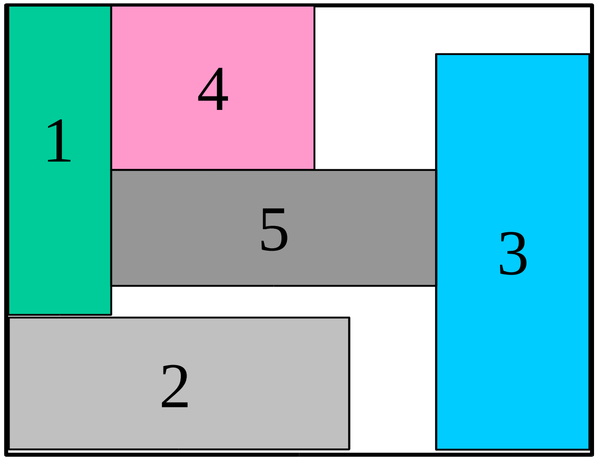
FCFS with backfilling

Without reservation

List algorithms: use available processors for executing the first possible task in the list.



FCFS with backfilling



list algorithm

Without reservation

Proposition: list algorithm is a 2-1/m approximation.

This is a special case of Graham 1975 (resource constraints), revisited by [Eyraud et al. IPDPS 2007].

The bound is tight (same example as in the well-known case in 1969 for sequential tasks).

With reservation

The guaranty is not valid.

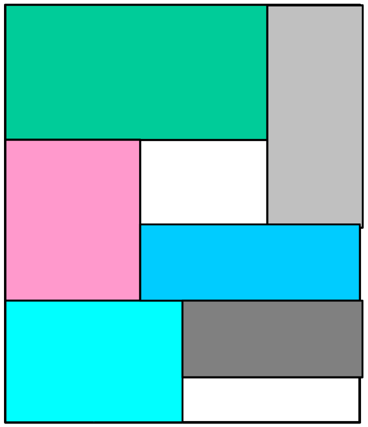
This is a special case of Graham 1975 (resource constraints), revisited by [Eyraud et al. IPDPS 2007].

The bound is tight (same example as in the well-known case in 1969 for sequential tasks).

Complexity

The problem is already NP-hard with no reservation.

Even worse, an optimal solution with arbitrary reservation may be delayed as long as we want:

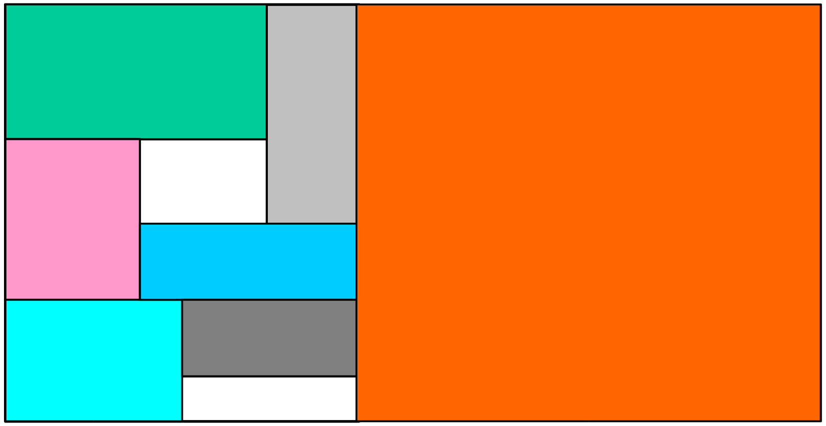


Cmax*

Complexity

The problem is already NP-hard with no reservation.

Even worse, an optimal solution with arbitrary reservation may be delayed as long as we want:

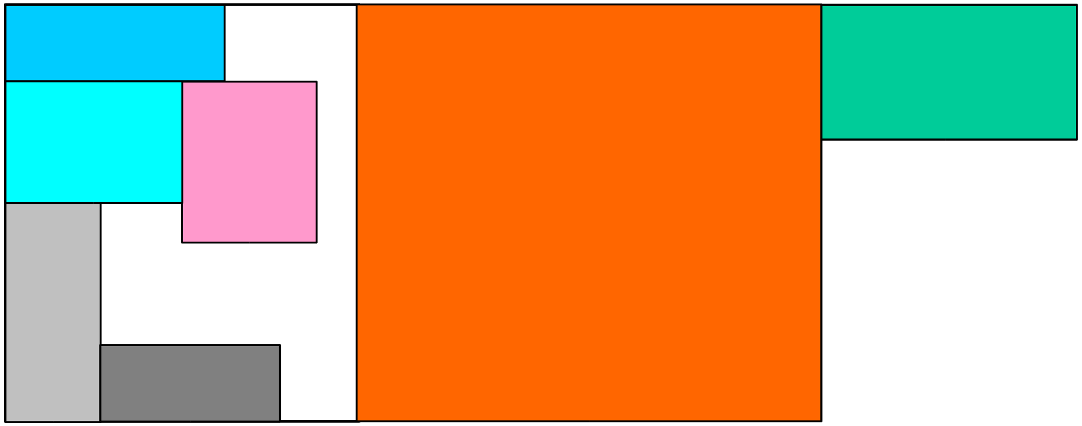


Cmax*

Complexity

The problem is already NP-hard with no reservation.

Even worse, an optimal solution with arbitrary reservation may be delayed as long as we want:

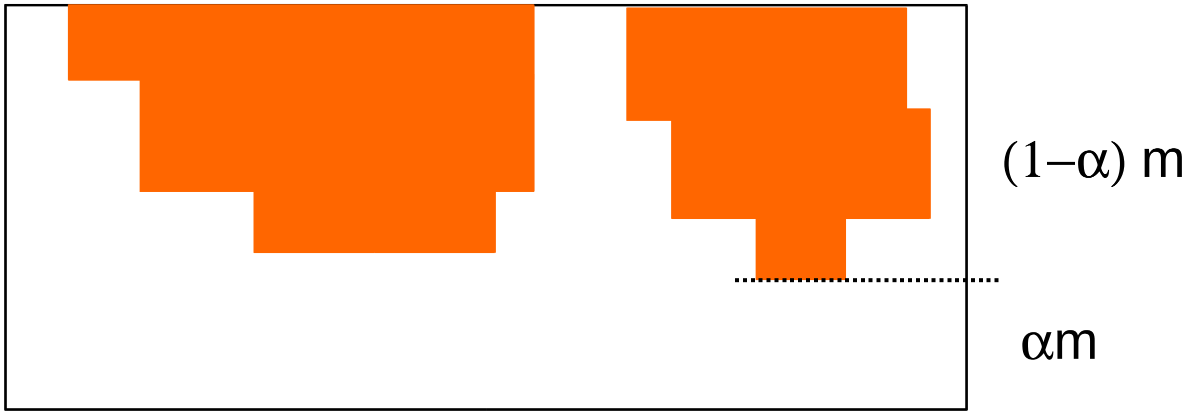


Conclusion: can not be approximated unless P=NP, even for m=1

Two preliminary results

Decreasing number of available processors

Restricted reservation problem: always a given part α of the processors is available $r(t) \leq (1-\alpha)m$ and for all task i, $q_i \leq \alpha m$.



Analysis

Case 1. The same approximation bound 2-1/m is still valid

Case 2. The list algorithm has a guaranty of $2/\alpha$

Insight of the proof: while the optimal uses m processors, list uses only α processors with a approximation of 2…

Analysis

Case 1. The same approximation bound 2-1/m is still valid

Case 2. The list algorithm has a guaranty of $2/\alpha$

Insight of the proof: while the optimal uses m processors, list uses only α processors with a approximation of 2…

There exists a lower bound which is arbitrary close to this bound:

$2/\alpha - 1 + \alpha/2$ if $2/\alpha$ is an integer

Conclusion

It remains a lot of interesting open problems with reservations.

Using preemption
Not rigid reservations
Better approximation (more costly)