Scheduling Large Jobs by Abstraction Refinement

Thomas A. Henzinger Vasu Singh Thomas Wies
Damien Zufferey

IST Austria

June 16, 2011

Motivation (1)

Cloud computing gives the *illusion* of ∞ (virtual) resources.

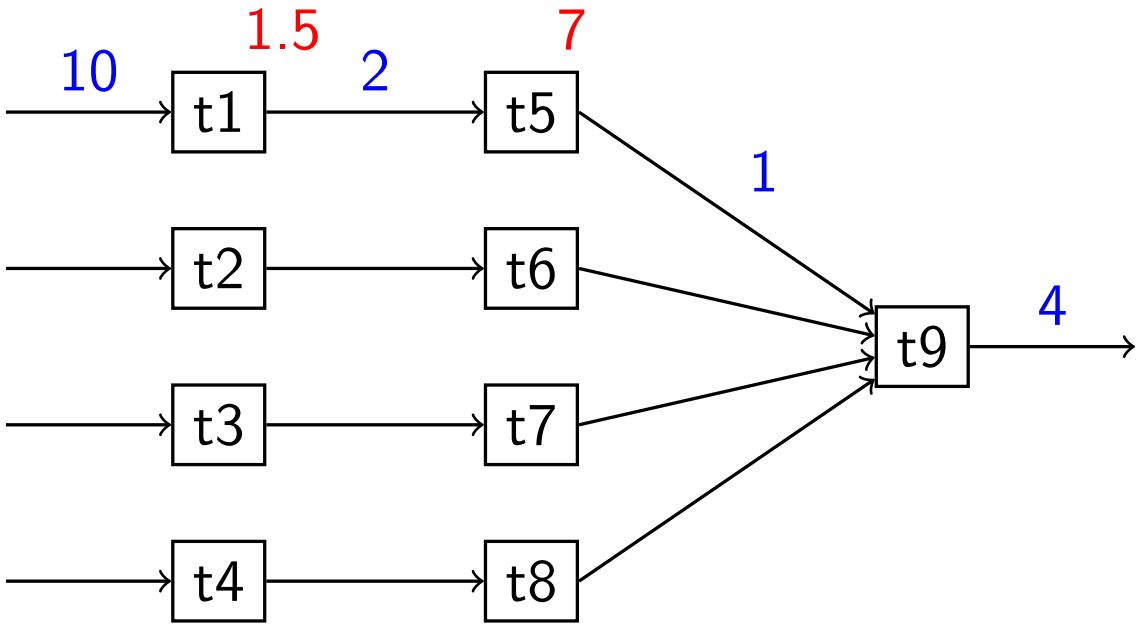
Actually there is a finite amount of (physical) resources.

We would like to efficiently share those resources:

1. being able to distinguish high priority (serving customer *now*) from low priority (batch) requests;
2. schedule accordingly.

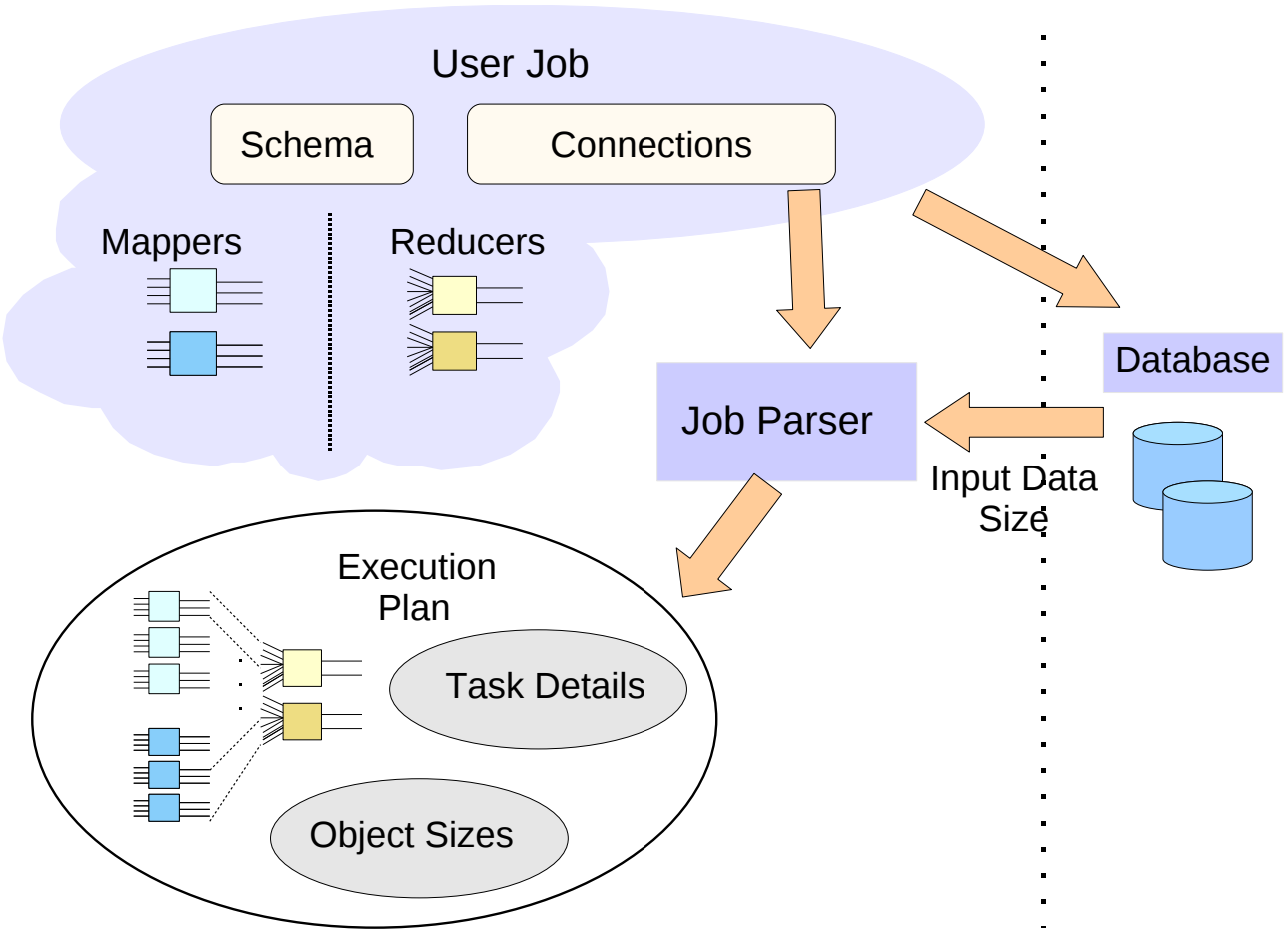
Therefore, we should be able to plan ahead computations.

Jobs Model

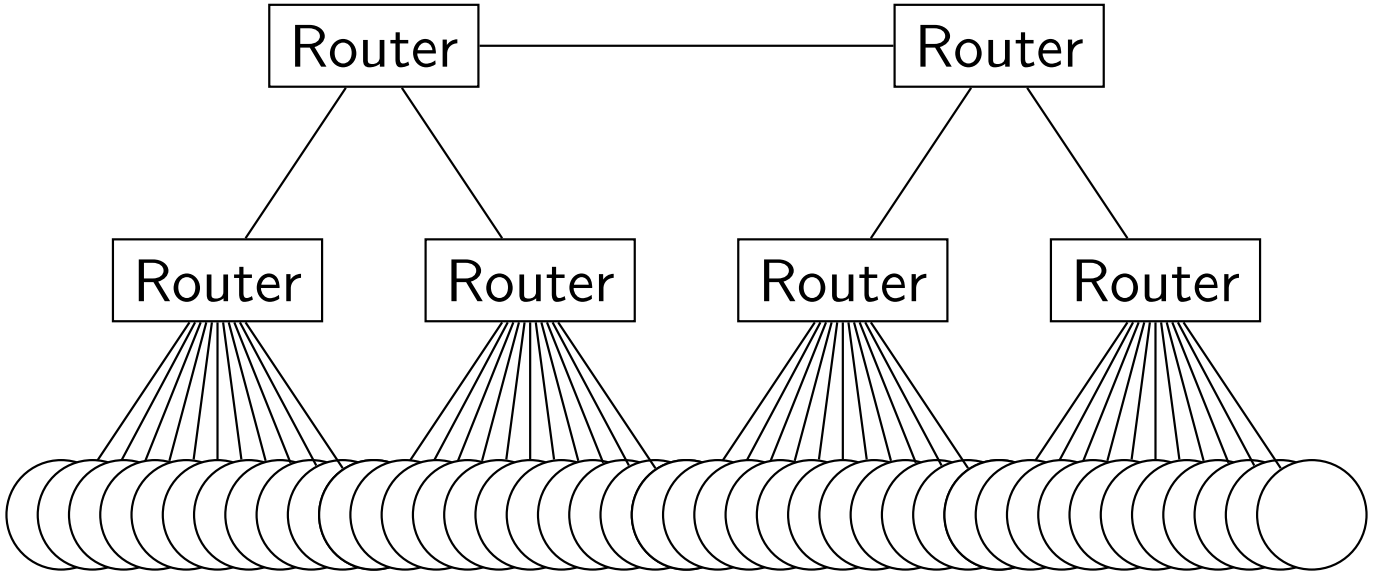


- A Job is a directed acyclic task (DAG) of tasks.
- Node are marked with worst case duration.
- Edges are marked with data transfer.
- duration and data can be parametric in the input.

Parametric Jobs



Infrastructure Model

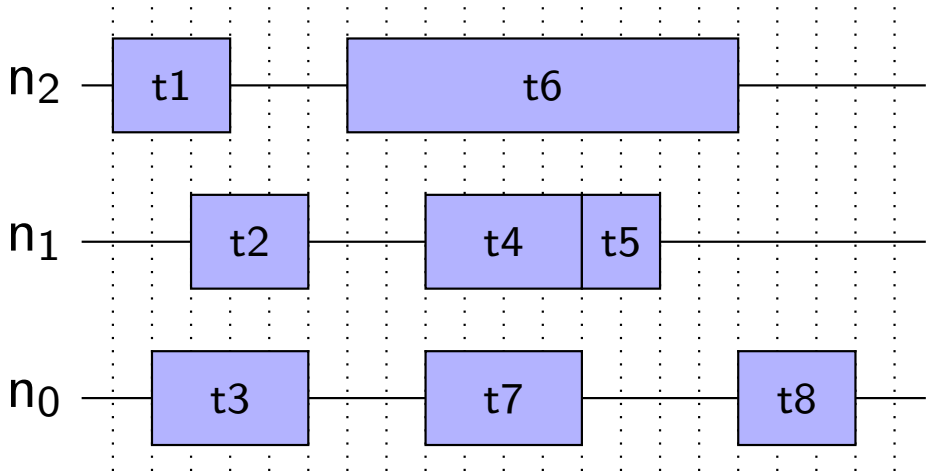


Datacentre as a tree-like graph:

- internal nodes are router;
- leaves are compute nodes (computation speed);
- edges specifies the bandwidth.

Schedule

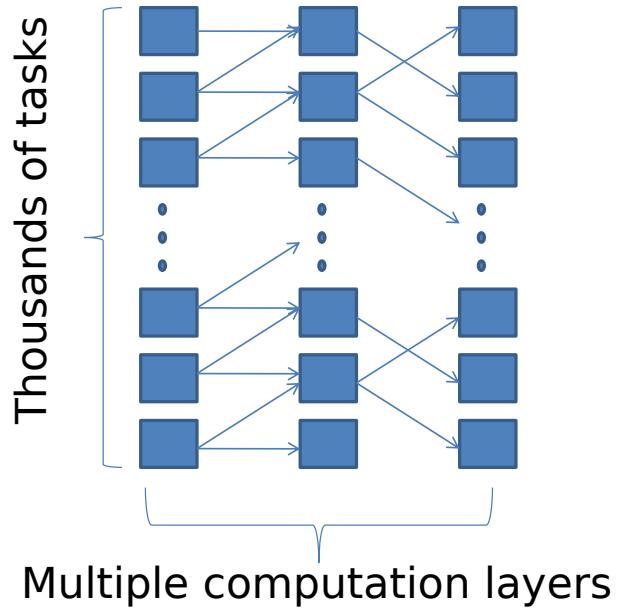
A relation between tasks, nodes, and time.



Problem: the scale

On the infrastructure side:

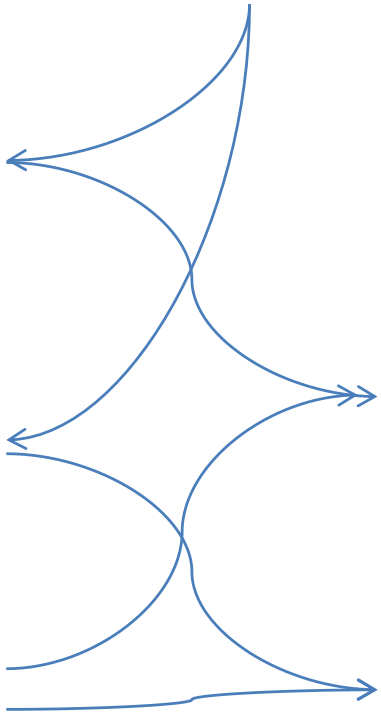
On the job side:



Given the scale, people have been using dynamic scheduling.

Example: Hadoop MapReduce engine

Job ⟹



Motivation (2)

Dynamic Scheduling: use work queues, priorities, but limited.

Without knowledge of jobs, this is the best you can do.

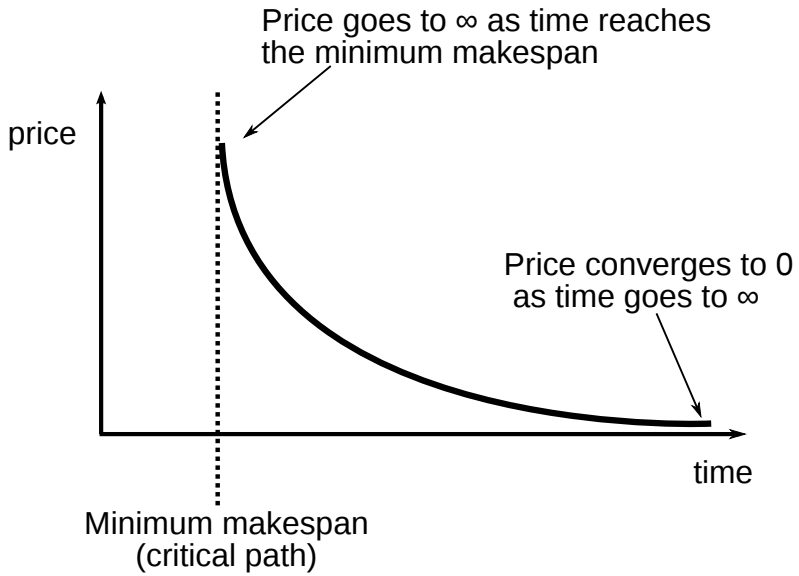
We need to ask the user submitting a job for:

- what kind of resources the job requires;
- a deadline/priority for the job.

In exchange we can give him an expected completion time.

Giving incentive to plan in advance

The scheduler returns not one but many possible schedules with different finish times.
Use a pricing model to associate a cost to the schedules.
Include the "scheduling difficulty" in the cost, give a discount to schedule with later finish time.



Problem: static scheduling is *hard*.
Only possible if the scheduler can handle the work load.

Static scheduling:

Computing optimal schedule: NP-hard

Heuristics (Greedy, deadline division etc.): $|J| \cdot |C|$

With 1000 tasks job and 200 nodes cloud, a greedy scheduler takes up to 5 minutes!

To do better we should try to solve only a simplified problem.

Core idea:

Assumption: job and infrastructure regularity

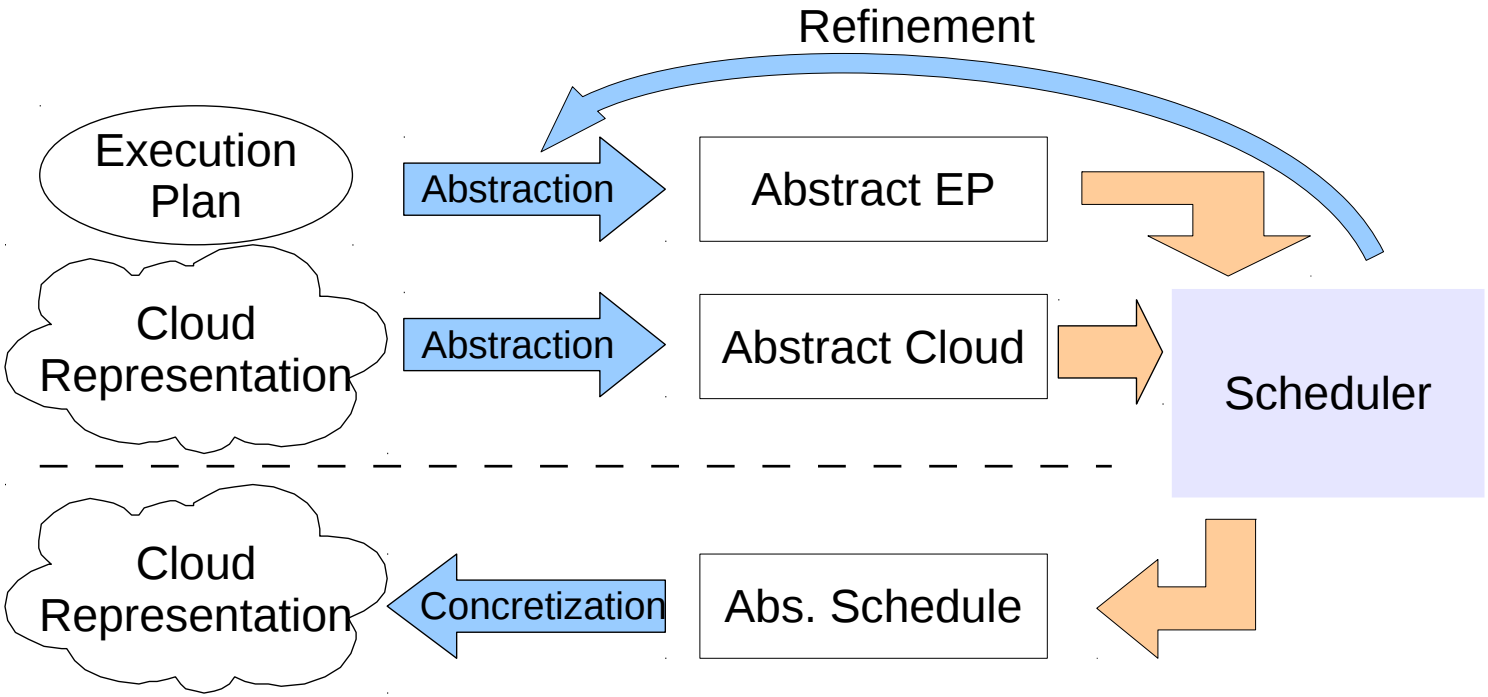
Idea: regularity makes large scale scheduling feasible

How: Using abstraction techniques

- Over-approximate the resources needed by the job J to get $J^{\#}$.
- Under-approximate the available resources for the cloud C to get $C^{\#}$.
- Get a schedule for $(J^{\#}, C^{\#})$ and use it to construct a schedule for (J, C).

Approximation is done in order to introduce regularity (symmetry).
Symmetry can be captured using multiplicity.
Over-/Under-approximation guarantees the existence of a schedule.

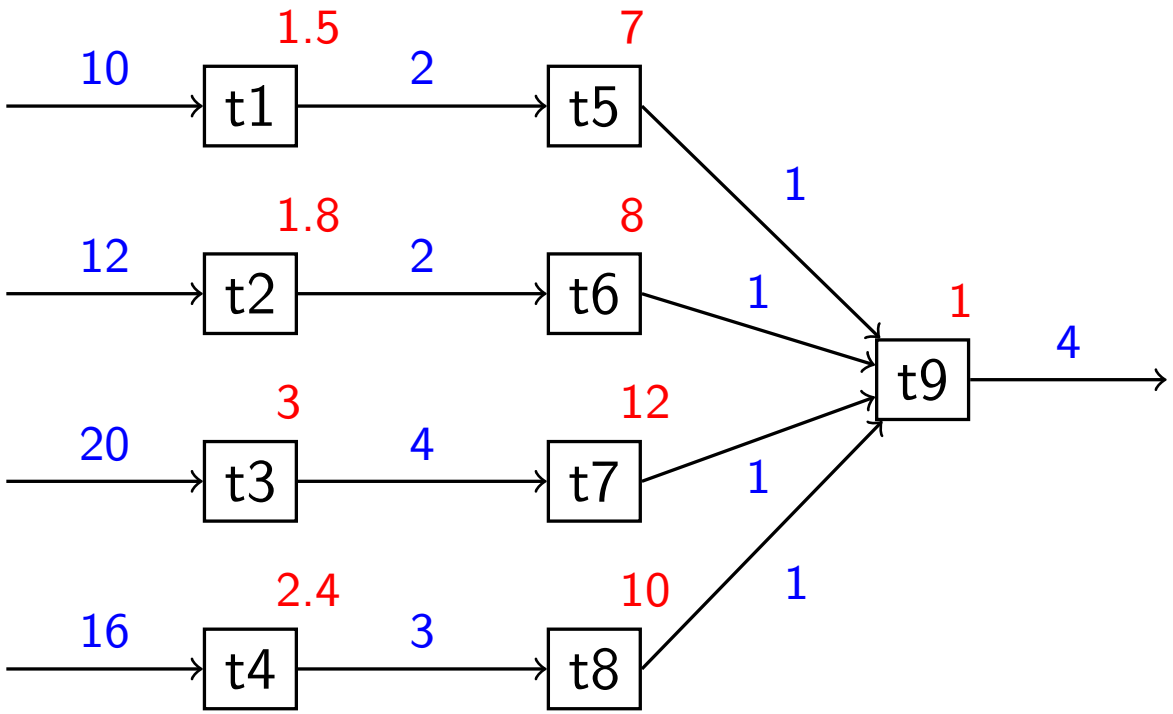
Scheduling using abstraction refinement



When to refine: not good enough (utilisation, makespan).

Abstraction for jobs:

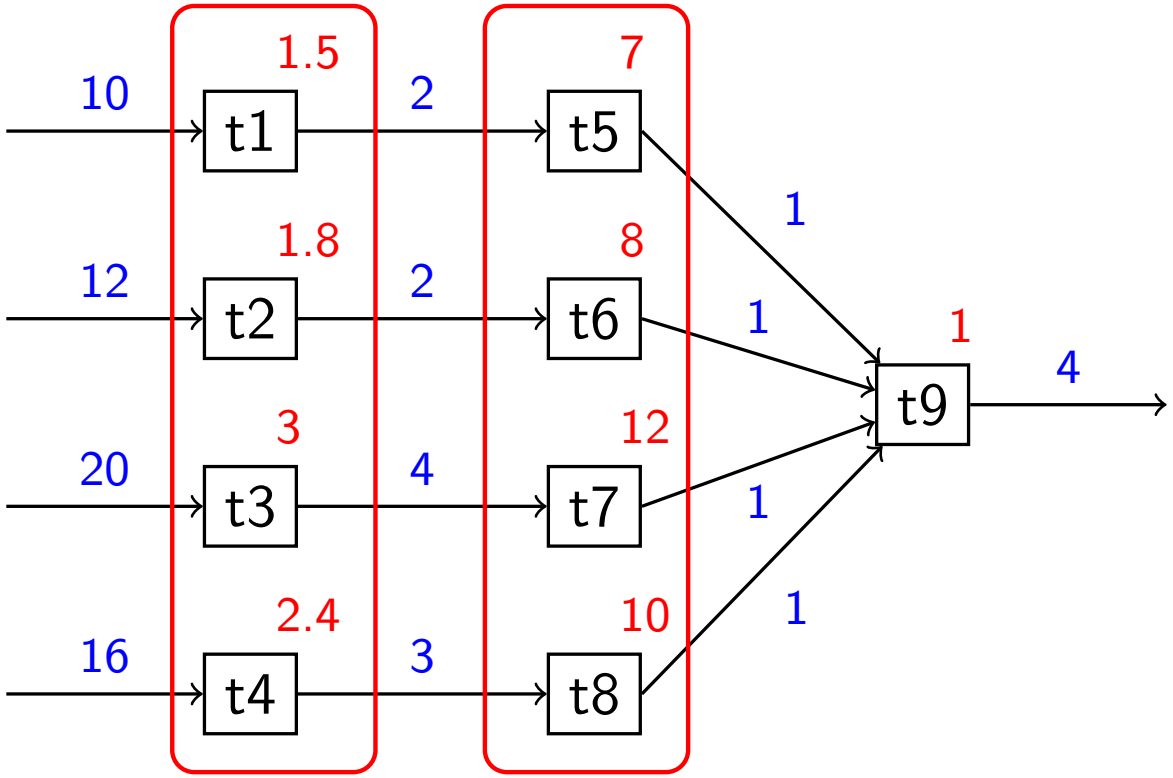
Group independent tasks as per a topological sort. Merge them into an abstract task.



Decide when to merge based on similarity.
Two tasks with duration d_1, d_2 are X-similar iff $1/X \leq d_1/d_2 \leq X$.

Abstraction for jobs:

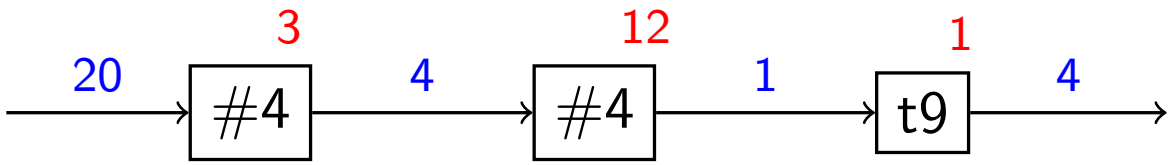
Group independent tasks as per a topological sort. Merge them into an abstract task.



Decide when to merge based on similarity.
Two tasks with duration d_1, d_2 are X-similar iff $1/X \leq d_1/d_2 \leq X$.

Abstraction for jobs:

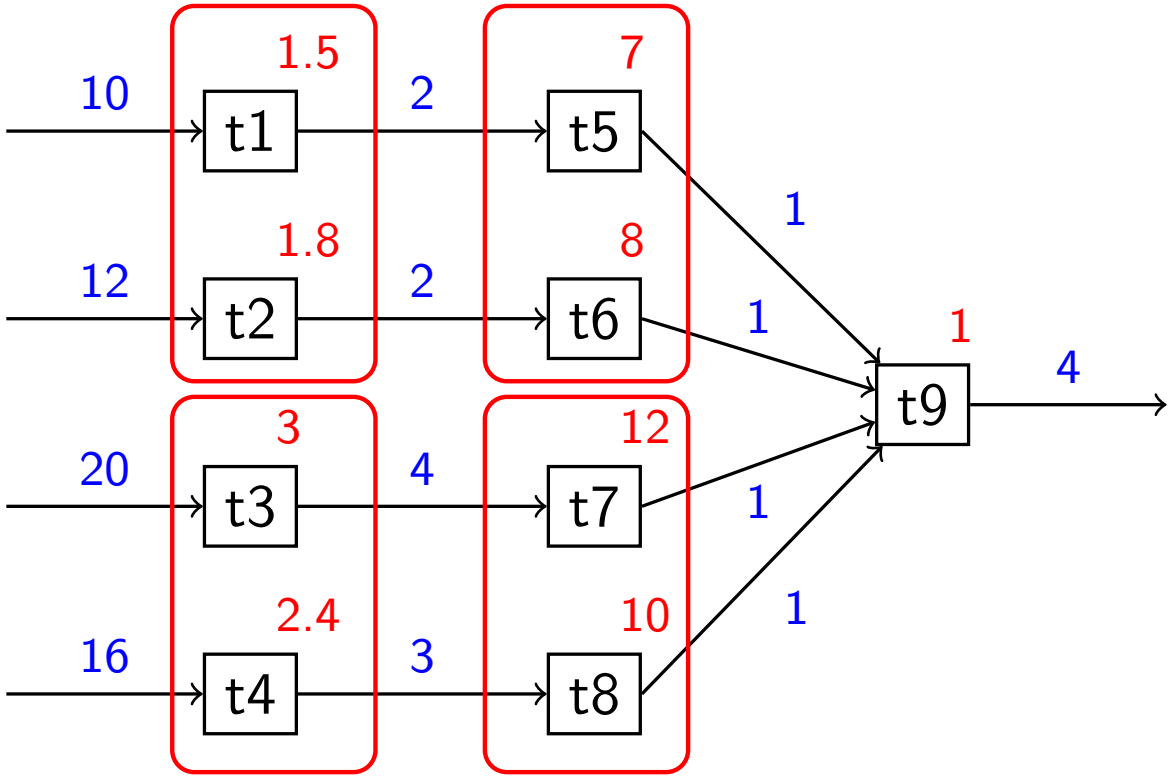
Group independent tasks as per a topological sort. Merge them into an abstract task.



Decide when to merge based on similarity.
Two tasks with duration d_1, d_2 are X-similar iff $1/X \leq d_1/d_2 \leq X$.

Abstraction for jobs:

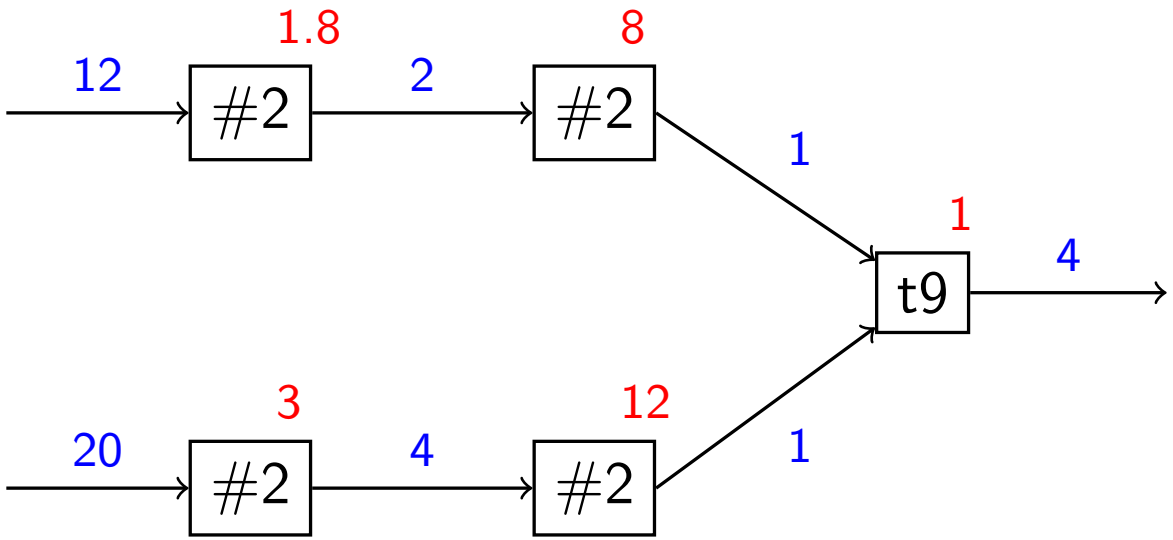
Group independent tasks as per a topological sort. Merge them into an abstract task.



Decide when to merge based on similarity.
Two tasks with duration d_1, d_2 are X-similar iff $1/X \leq d_1/d_2 \leq X$.

Abstraction for jobs:

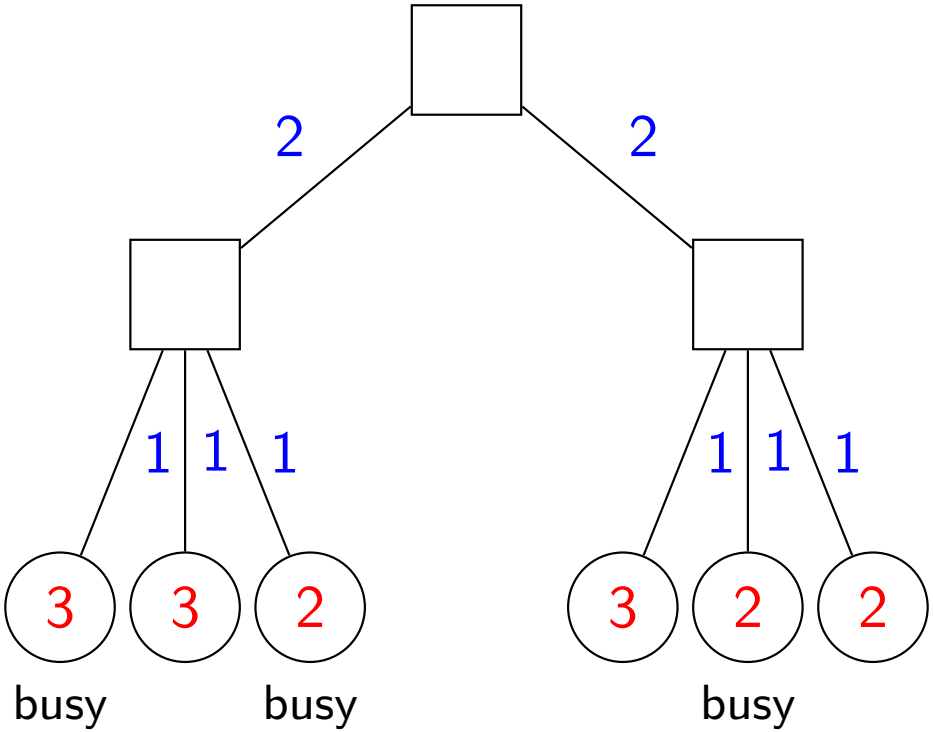
Group independent tasks as per a topological sort. Merge them into an abstract task.



Decide when to merge based on similarity.
Two tasks with duration d_1, d_2 are X-similar iff $1/X \leq d_1/d_2 \leq X$.

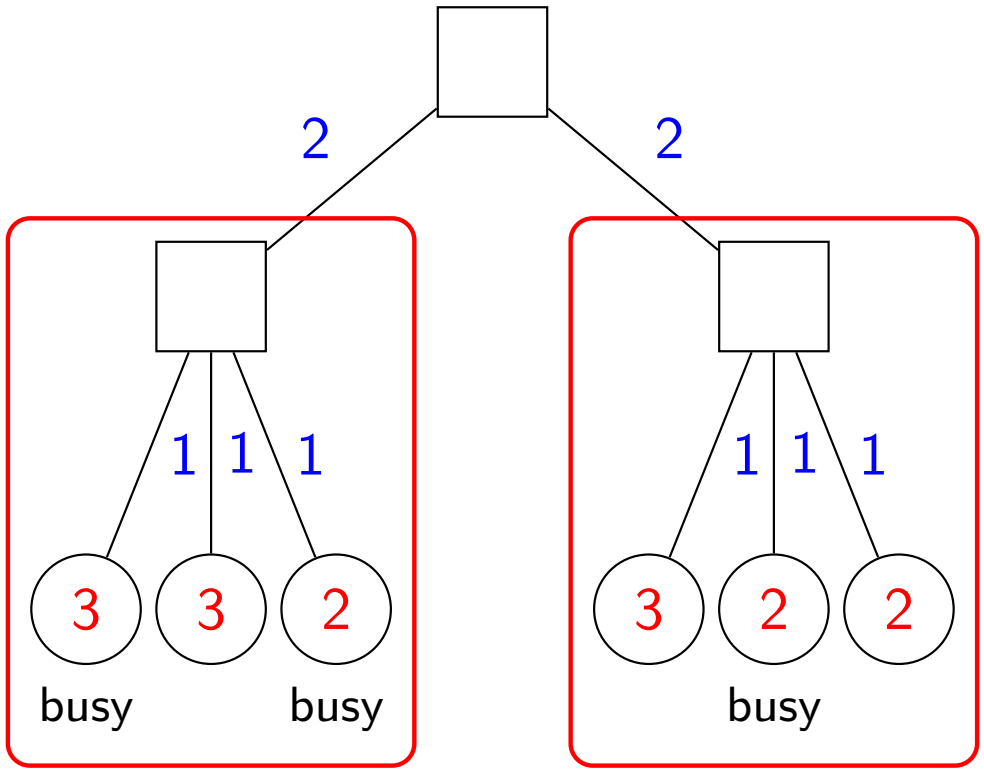
Datacentre abstraction

Rack abstraction: Create an abstract node for a group of nodes on a rack



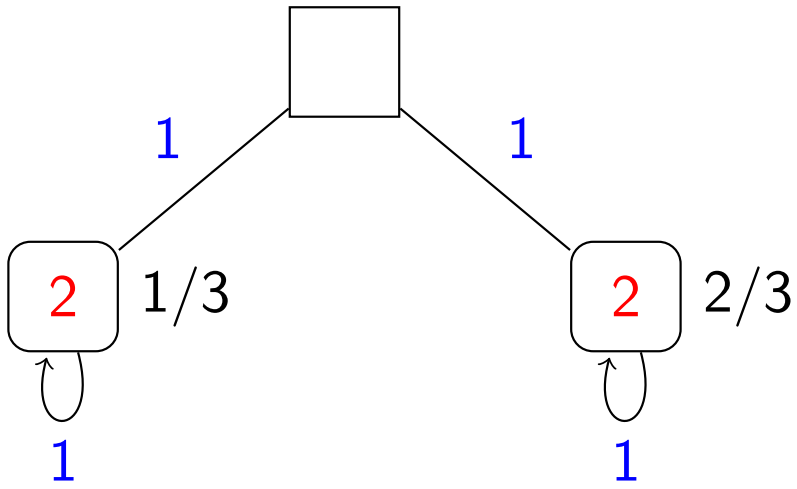
Datacentre abstraction

Rack abstraction: Create an abstract node for a group of nodes on a rack



Datacentre abstraction

Rack abstraction: Create an abstract node for a group of nodes on a rack



Two AR scheduler

Fisch: Free intervals scheduler

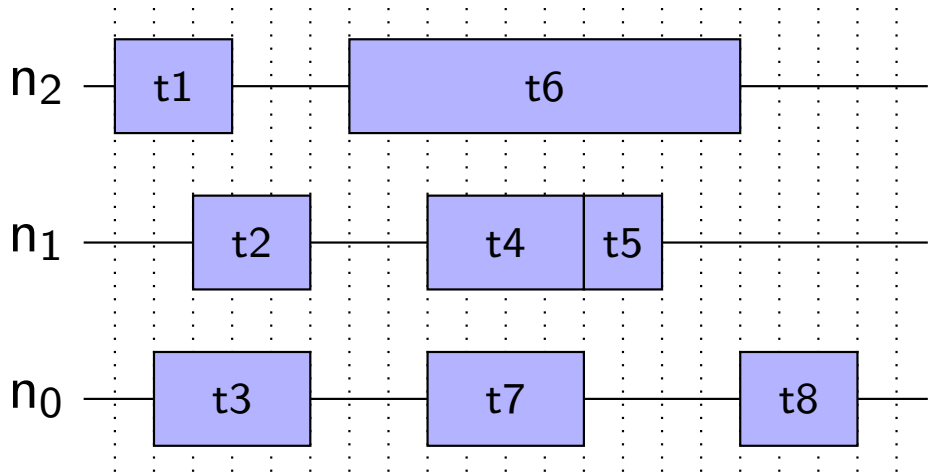
fixed datacentre abstraction,
refines the job abstraction.

Blind: Buddy-list in datacentre

fixed job abstraction,
refines the datacentre abstraction.

Fisch: Free intervals scheduler

Build an index (search engine) to track only the free intervals.



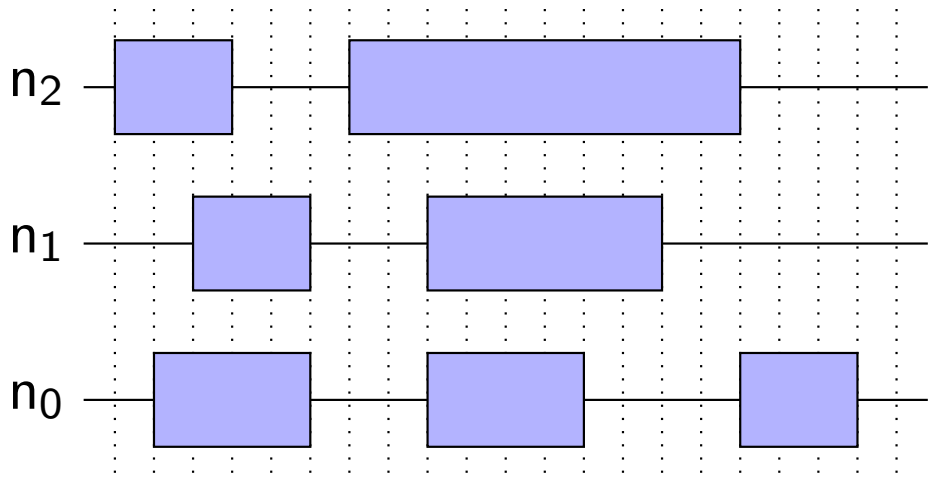
Index of the free intervals (computed at the rack level):

1 :
2 :
3 :
4 :
_ :

Starts with an X-similar job abstraction where $X = \infty$
Refinement lowers the X

Fisch: Free intervals scheduler

Build an index (search engine) to track only the free intervals.



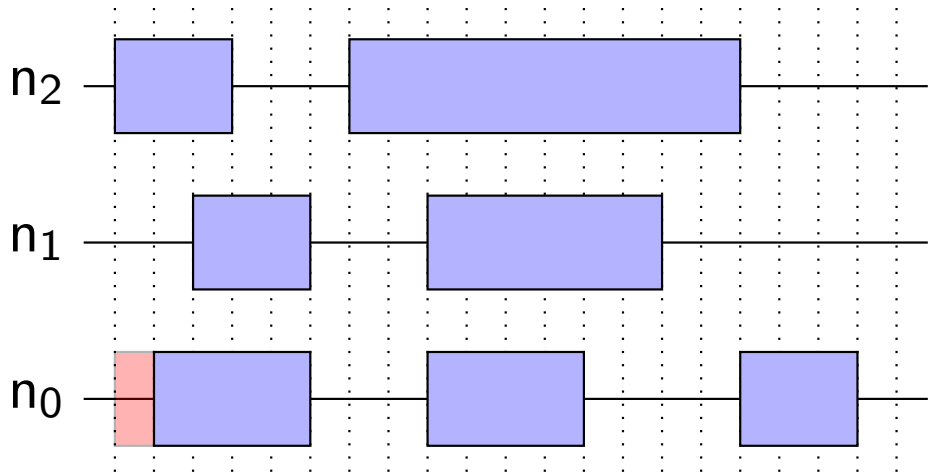
Index of the free intervals (computed at the rack level):

1 :
2 :
3 :
4 :
- :

Starts with an X-similar job abstraction where $X = \infty$
Refinement lowers the X

Fisch: Free intervals scheduler

Build an index (search engine) to track only the free intervals.



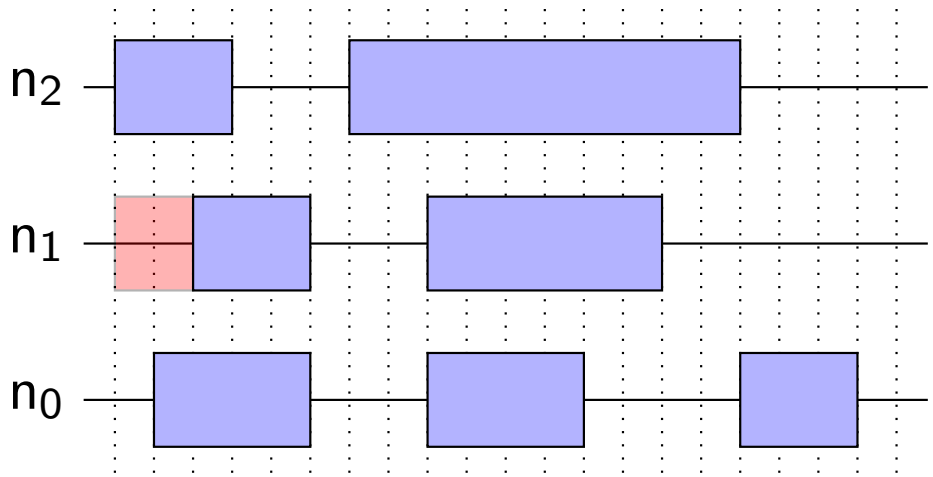
Index of the free intervals (computed at the rack level):

1	:	$(n_0,0)$
2	:	
3	:	
4	:	
_	:	

Starts with an X-similar job abstraction where $X = \infty$
Refinement lowers the X

Fisch: Free intervals scheduler

Build an index (search engine) to track only the free intervals.



Index of the free intervals (computed at the rack level):

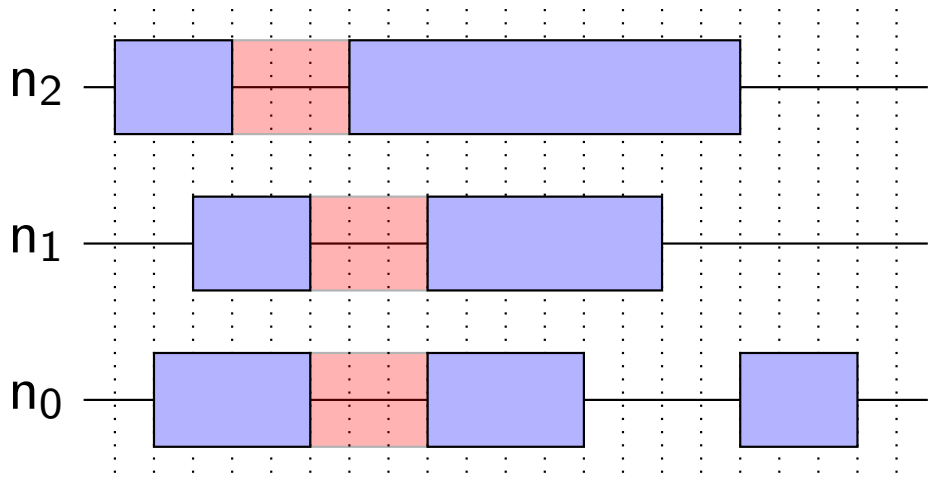
1	:	$(n_0,0)$
2	:	$(n_1,0)$
3	:	
4	:	
_	:	

Starts with an X-similar job abstraction where $X = \infty$

Refinement lowers the X

Fisch: Free intervals scheduler

Build an index (search engine) to track only the free intervals.



Index of the free intervals (computed at the rack level):

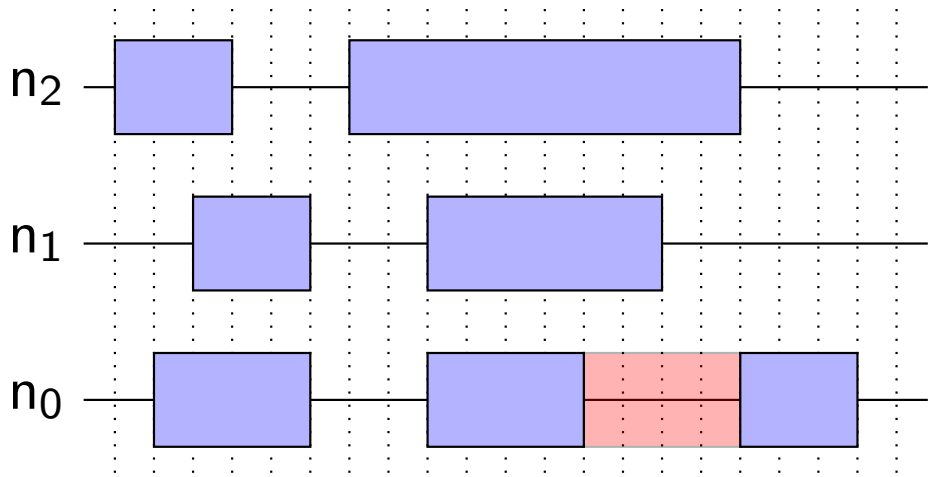
1	:	$(n_0,0)$
2	:	$(n_1,0)$
3	:	$(n_2,3)$, $(n_0,5)$, $(n_1,5)$
4	:	
_	:	

Starts with an X-similar job abstraction where $X = \infty$

Refinement lowers the X

Fisch: Free intervals scheduler

Build an index (search engine) to track only the free intervals.



Index of the free intervals (computed at the rack level):

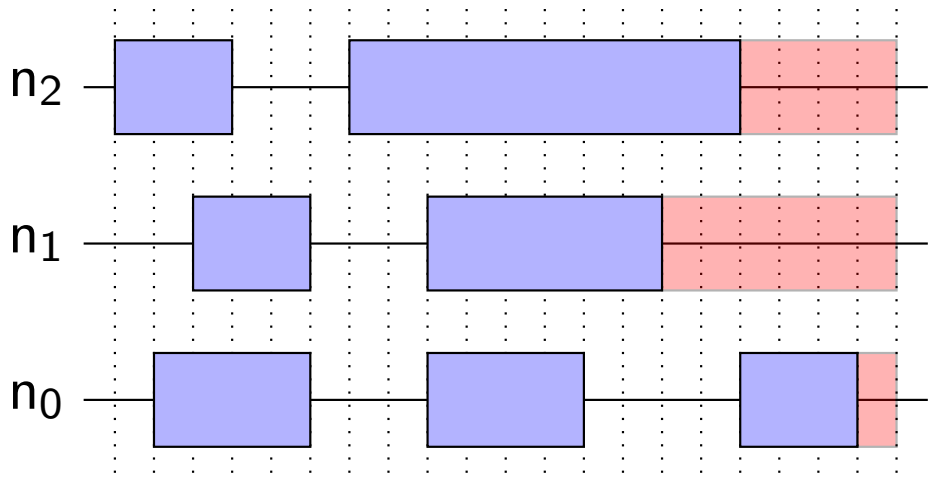
1	:	(n_0,0)
2	:	(n_1,0)
3	:	(n_2,3), (n_0,5), (n_1,5)
4	:	(n_0,12)
_	:	

Starts with an X-similar job abstraction where $X = \infty$

Refinement lowers the X

Fisch: Free intervals scheduler

Build an index (search engine) to track only the free intervals.



Index of the free intervals (computed at the rack level):

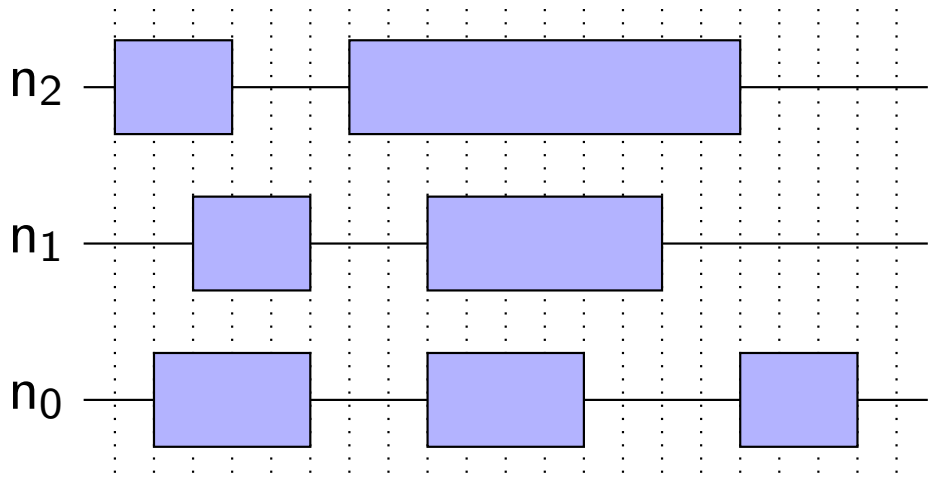
1	:	(n_0,0)
2	:	(n_1,0)
3	:	(n_2,3), (n_0,5), (n_1,5)
4	:	(n_0,12)
_	:	(n_1,14), (n_2,16), (n_0,19)

Starts with an X-similar job abstraction where $X = \infty$

Refinement lowers the X

Fisch: Free intervals scheduler

Build an index (search engine) to track only the free intervals.



Index of the free intervals (computed at the rack level):

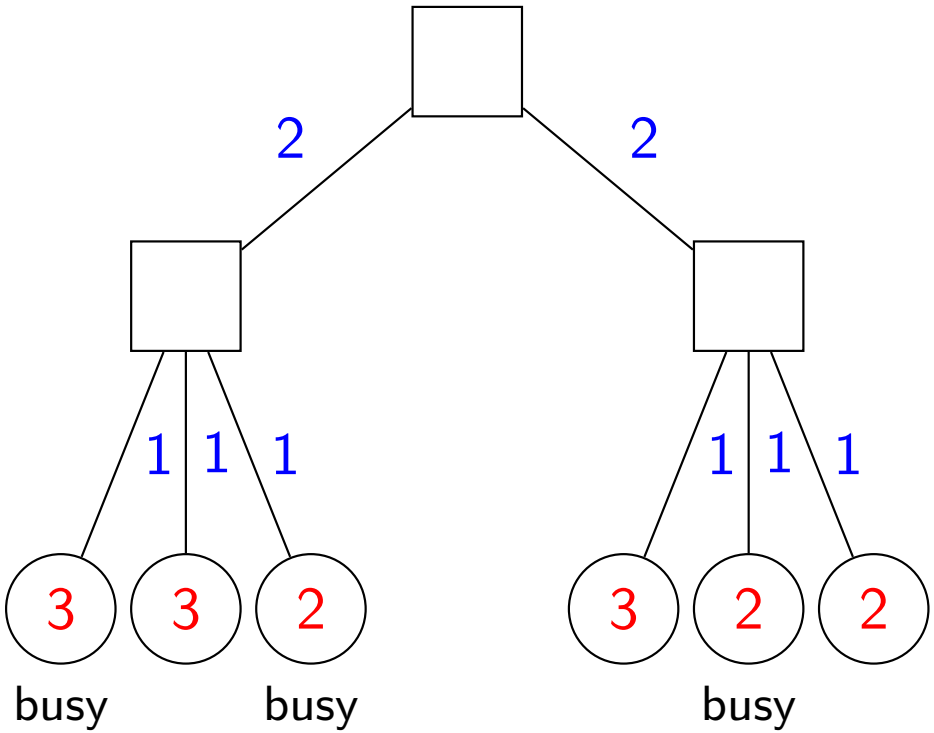
1	:	(n_0,0)
2	:	(n_1,0)
3	:	(n_2,3), (n_0,5), (n_1,5)
4	:	(n_0,12)
–	:	(n_1,14), (n_2,16), (n_0,19)

Starts with an X-similar job abstraction where $X = \infty$

Refinement lowers the X

Blind: Buddy-list in datacentre

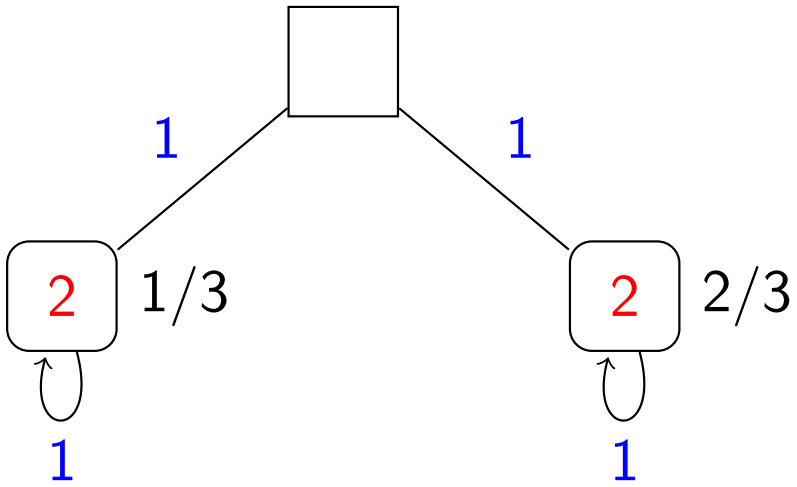
buddy-list (garbage collection) principle to group machines.



Uses utilisation as quality measure to decide when to refine a node.

Blind: Buddy-list in datacentre

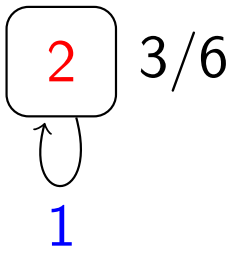
buddy-list (garbage collection) principle to group machines.



Uses utilisation as quality measure to decide when to refine a node.

Blind: Buddy-list in datacentre

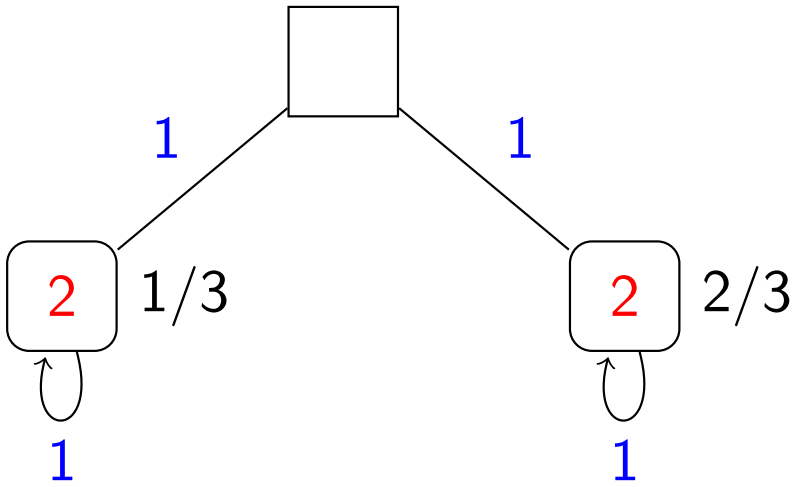
buddy-list (garbage collection) principle to group machines.



Uses utilisation as quality measure to decide when to refine a node.

Blind: Buddy-list in datacentre

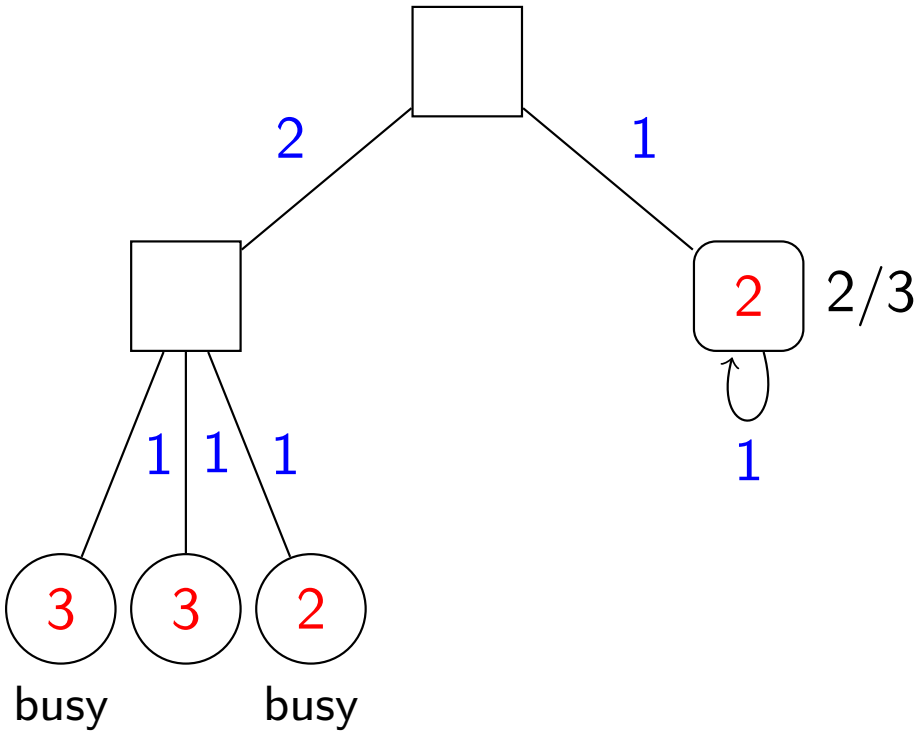
buddy-list (garbage collection) principle to group machines.



Uses utilisation as quality measure to decide when to refine a node.

Blind: Buddy-list in datacentre

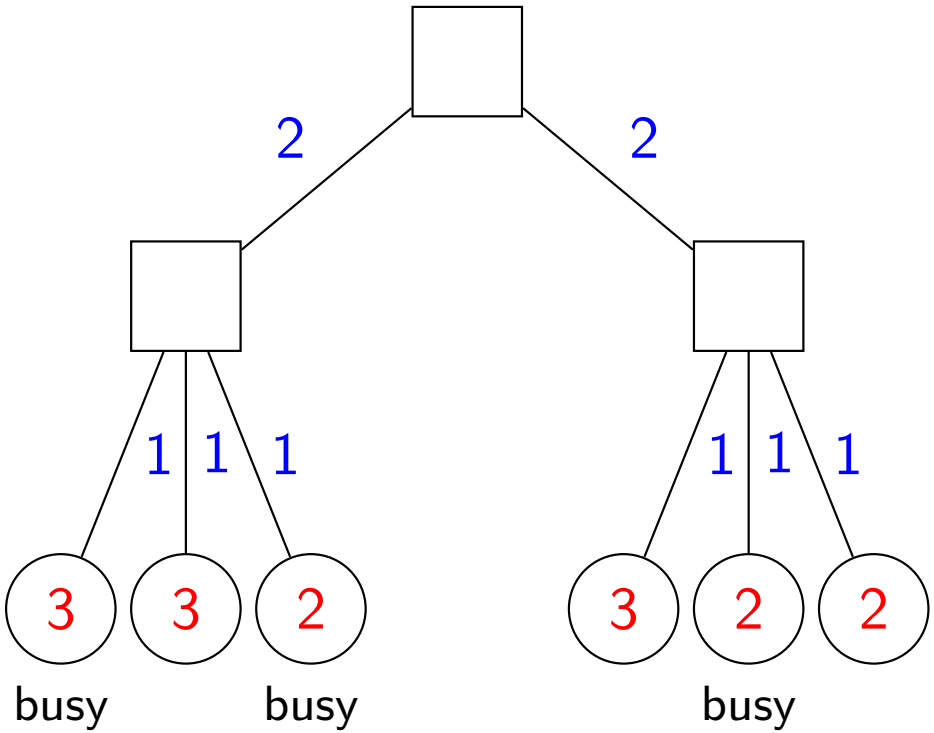
buddy-list (garbage collection) principle to group machines.



Uses utilisation as quality measure to decide when to refine a node.

Blind: Buddy-list in datacentre

buddy-list (garbage collection) principle to group machines.



Uses utilisation as quality measure to decide when to refine a node.

Infrastructure Refinement: a race to the bottom ?

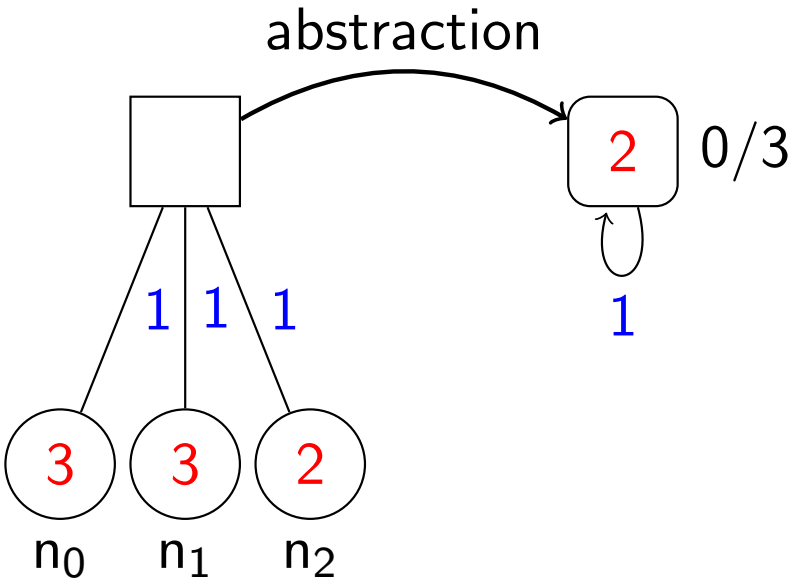
If we keep on refining we eventually get to the concrete system. Thus, an abstraction would only provide transient gain.

We also need to coarsen the abstraction.

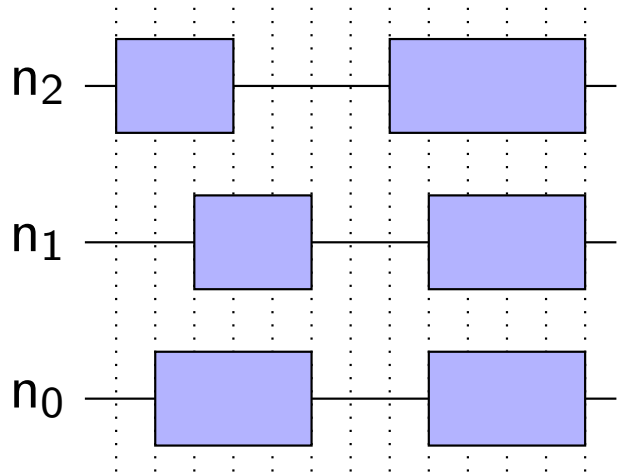
Caveat: for a given abstraction there are concrete schedules with no corresponding abstract schedules.

Fortunately, time is on our side. Tasks scheduled at a finer level of abstraction will eventually be executed. However, there is a transition period with two abstractions levels.

Allocation on an abstract node

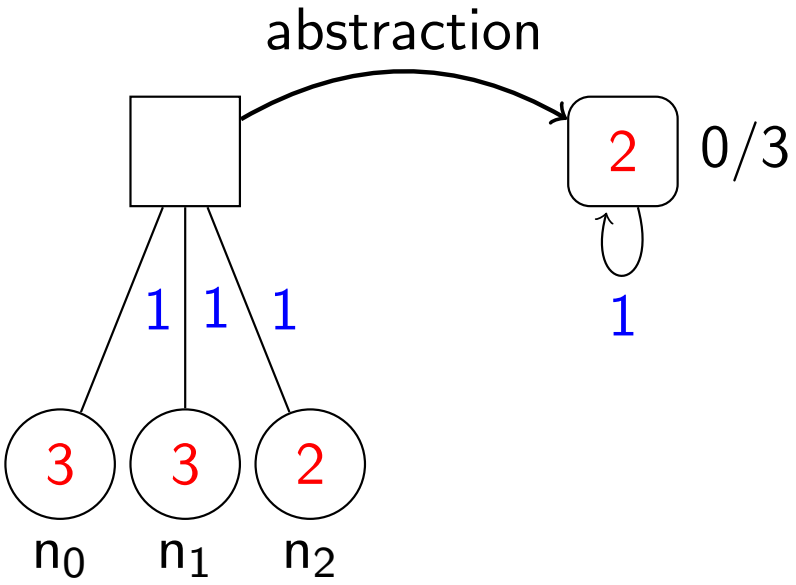


Schedule:

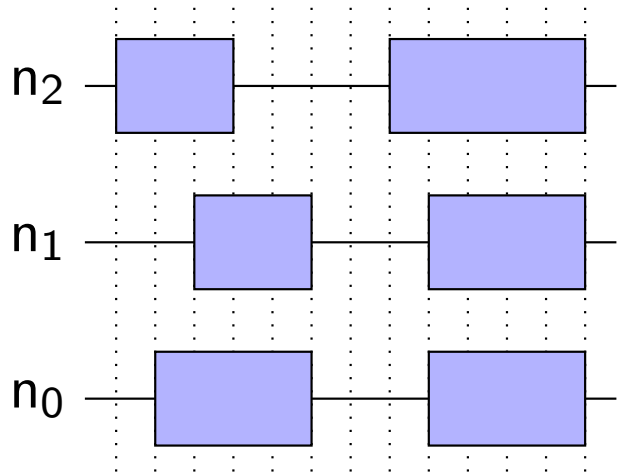


New request: 2 nodes for 3 units of time.

Allocation on an abstract node



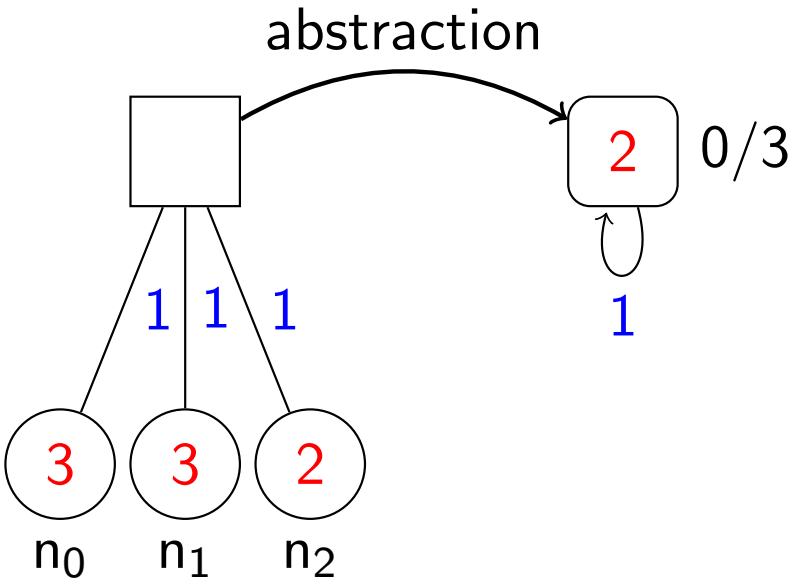
Schedule:



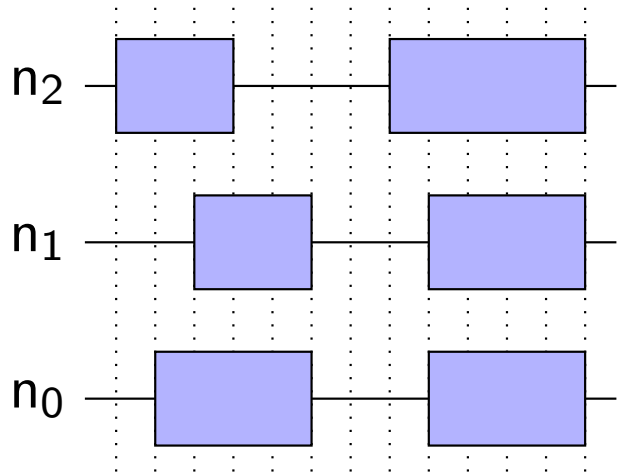
New request: 2 nodes for 3 units of time.

Events: (0,-1), (1,-1), (2,-1), (3,+1), (5,+2), (7,-1), (8,-2), (12,+3)

Allocation on an abstract node



Schedule:



New request: 2 nodes for 3 units of time.

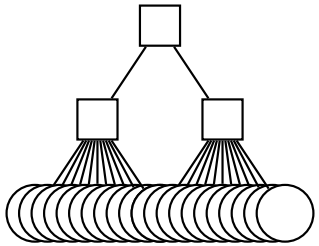
Events: (0,-1), (1,-1), (2,-1), (3,+1), (5,+2), (7,-1), (8,-2), (12,+3)

Traverses the events and shifts the allocations by -3+ϵ

Guarantees that there is two free nodes at 5, but does not tell which ones

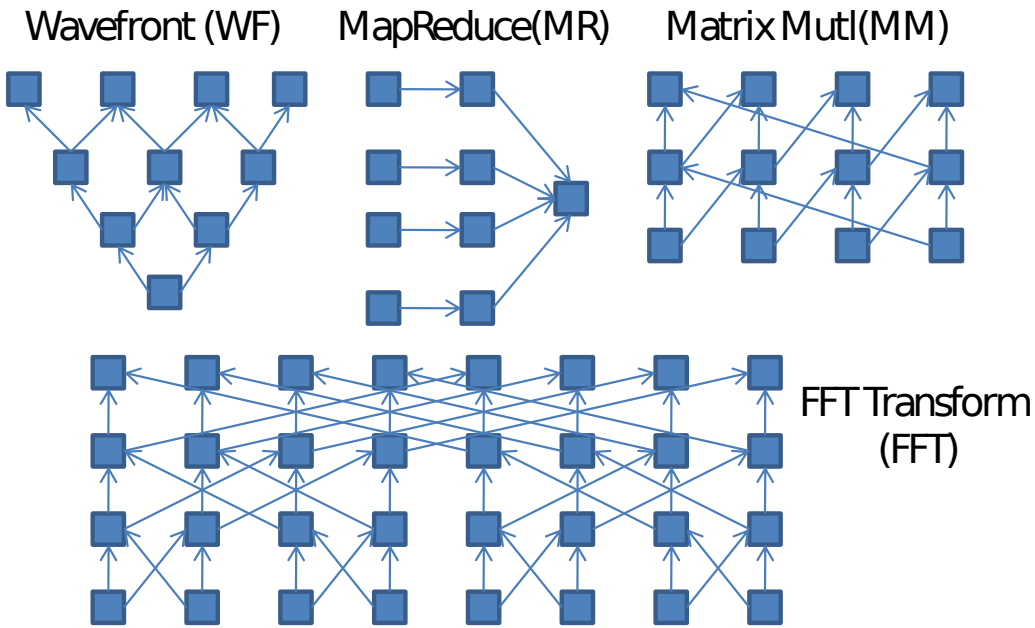
Experiments, part 1: simulation

datacenter:



2-tier datacenter
Half of the nodes: speed x,
Other half: speed 1.5 x

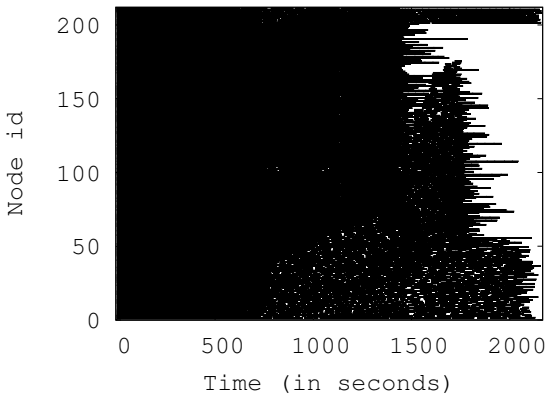
On the job side:



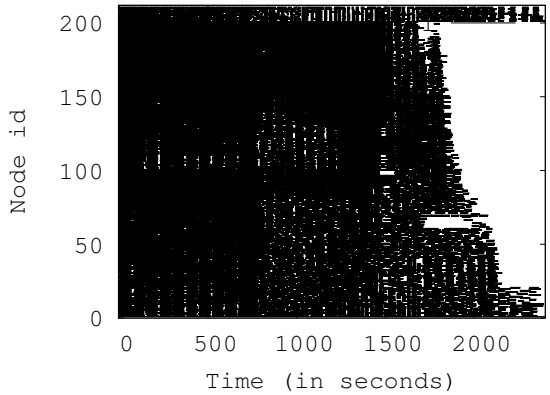
Experiments: the cost of abstraction

We then compare Fisch and Blind to a concrete greedy scheduler (baseline) on a sequence of 100 jobs (10-5000 tasks each). Latency is given per tasks.

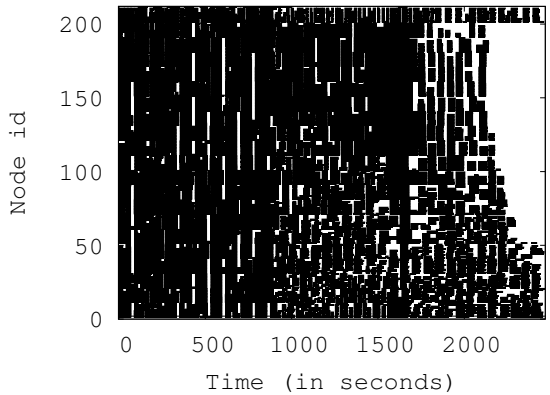
Scheduler	Latency (ms)	Utilization
Baseline	293	96 %
Fisch	0.27	92 %
Blind	0.16	91 %



(a) Baseline



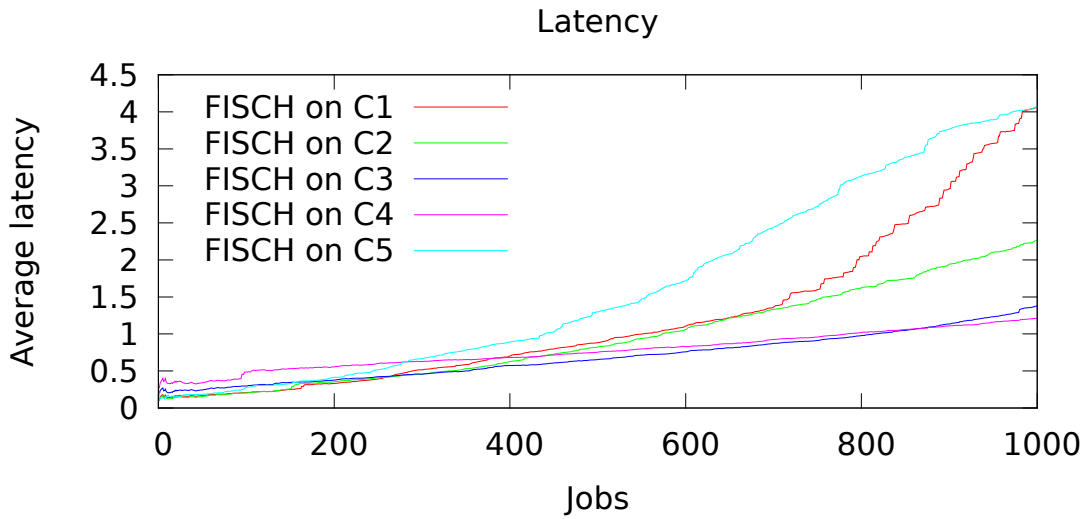
(b) Fisch

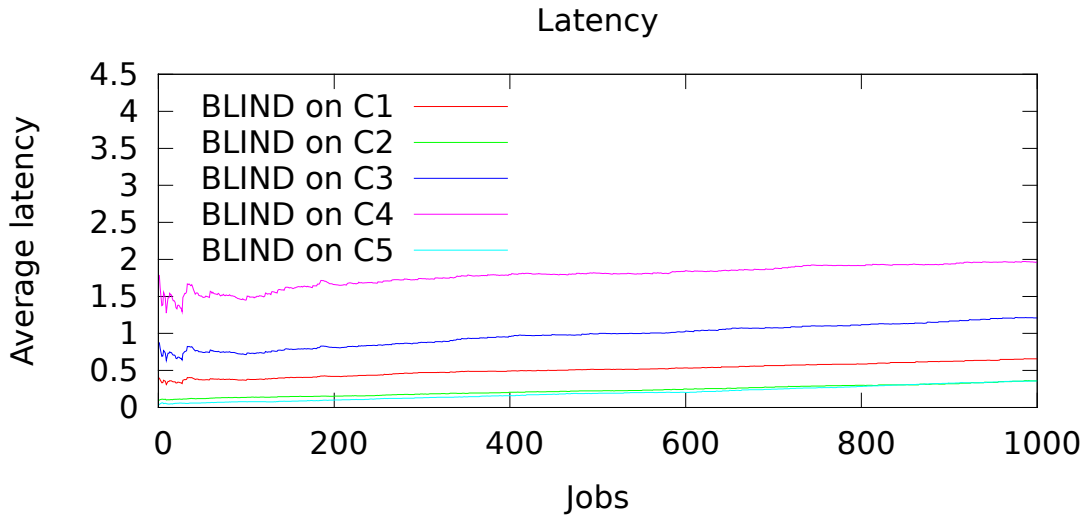


(c) Blind

Experiments: scaling

C1: 2000 nodes, 20 per rack
C2: 1600 nodes, 40 per rack
C3: 4000 nodes, 20 per rack
C4: 8000 nodes, 20 per rack
C6: 1000 nodes, 500 per rack





Experiments, part 2: real world

Caution: static scheduling alone will not work.

- Task duration are conservative estimates;
- Variability of the performance of the compute node.

We use static scheduling with backfilling.

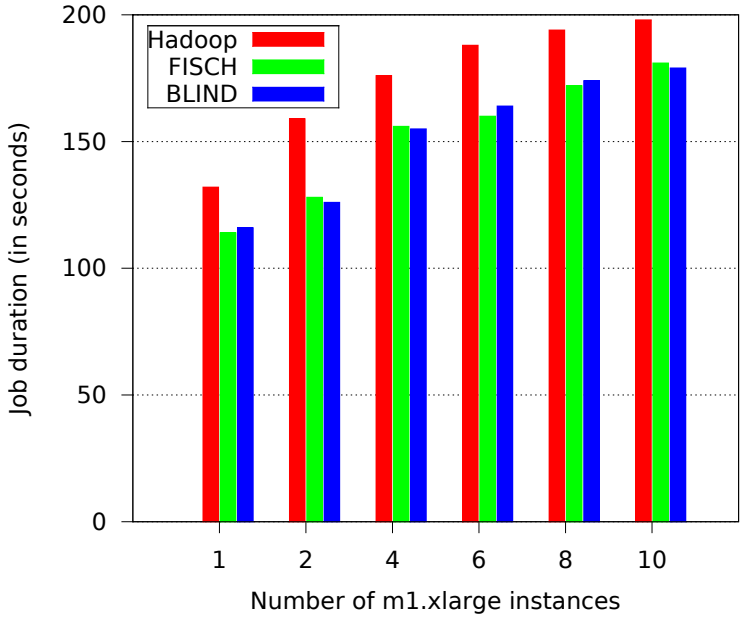
Job:

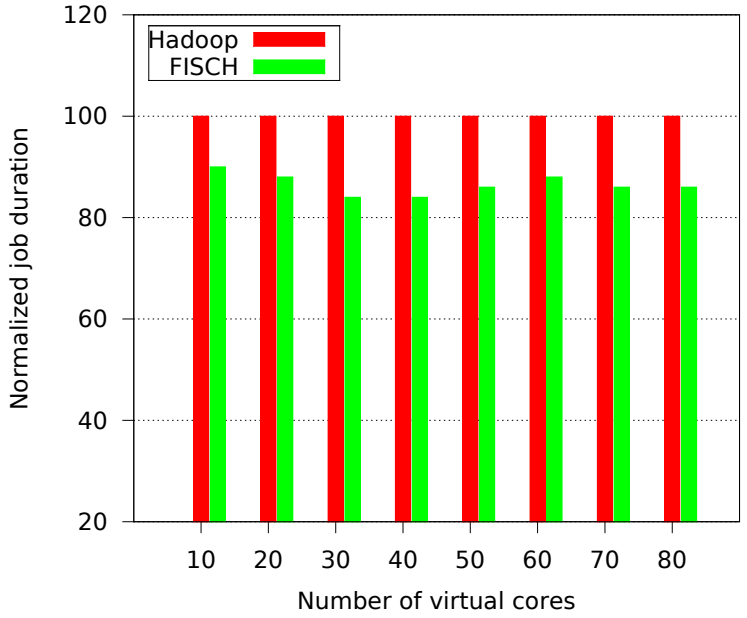
- The jobs are MapReduce jobs doing image transformation.
- Mapper: 8.1 seconds on average, estimate is 40 seconds
- Reducer: Identity operation

Infrastructure:

- Hadoop streaming version 0.19.0
- Amazon EC2 m1.xlarge instances (15GB RAM, 4 cores)
- Number of mappers = 50 * number of instances

Experiments: compared to Hadoop





Observations:

- The Hadoop framework requires large runtime overhead: results in slowdown of the job execution.
- Static scheduling allows to prefetch data, whereas dynamic scheduling does not

Conclusion

There is an opportunity to apply methods developed to solve computationally hard problem in verification to other area. While preserving a solid theoretical basis.

Questions ?