Discrete Optimization

Constraint-based Scheduling

Goals of the Lecture

- Scheduling with Constraint Programming
  - modeling
  - global constraints
  - and some nice techniques ...

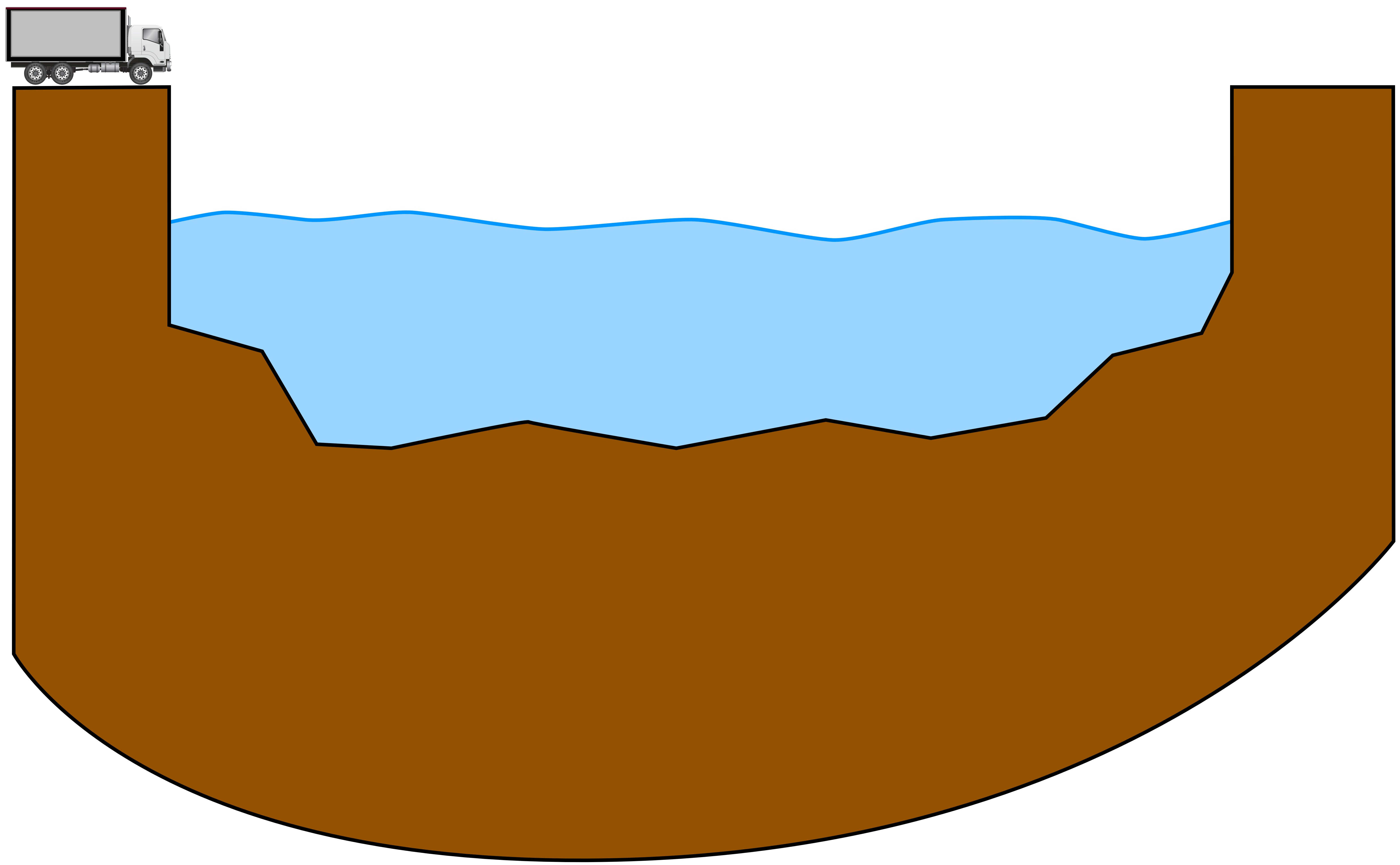
Motivation

- Very successful application area for constraint programming

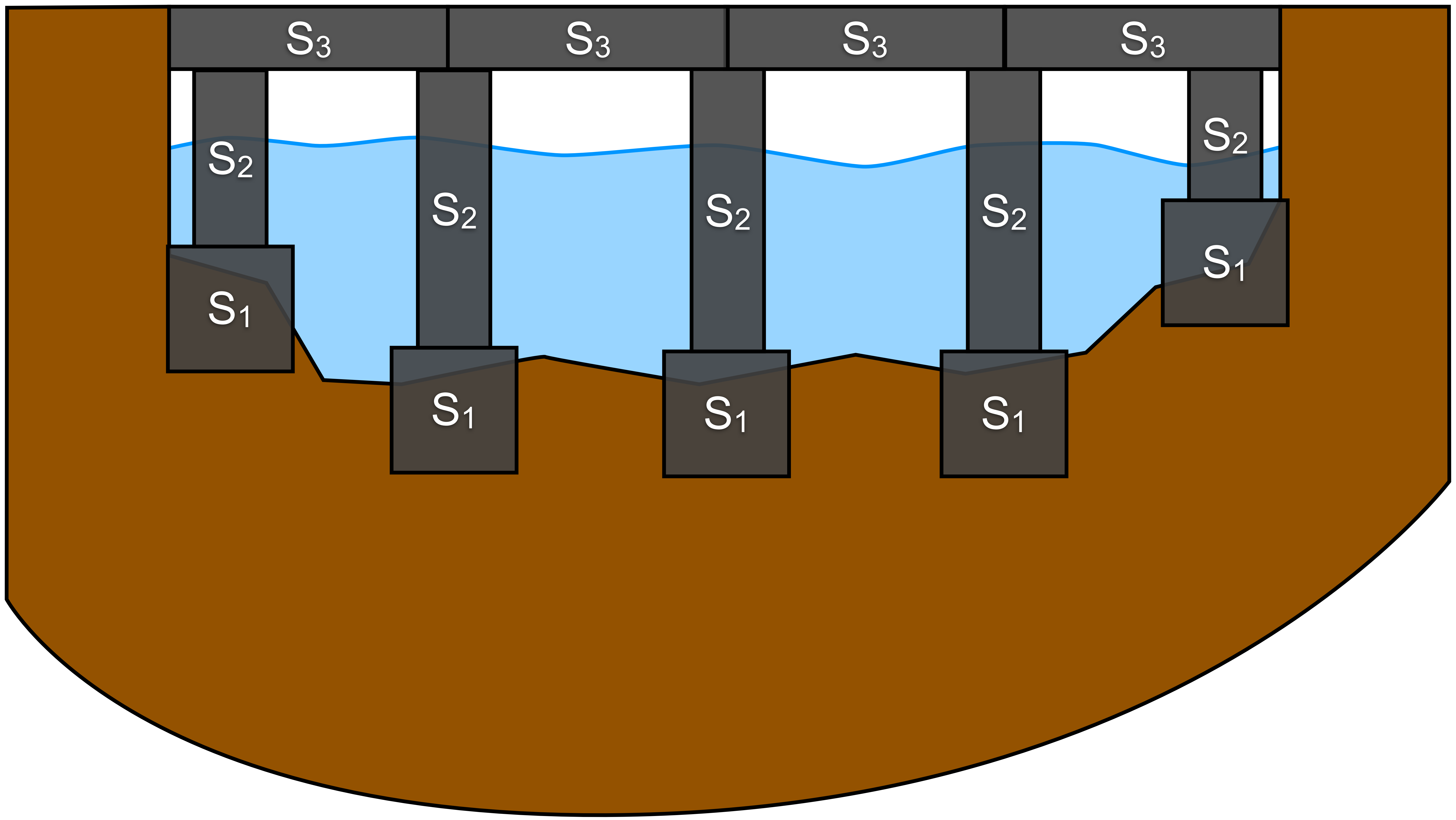
Motivation

- Very successful application area for constraint programming
- minimize project duration subject to
  - precedence constraints
  - disjunctive constraints: no two tasks scheduled on the same machine can overlap in time

Project Scheduling



Project Scheduling



Modeling

- Dedicated abstractions for scheduling
  - model-based computing

Modeling

- Dedicated abstractions for scheduling
  - model-based computing
- Domain-specific concepts such as
  - activities
  - resources
  - precedence constraints
  - ....

Modeling

- Dedicated abstractions for scheduling
  - model-based computing
- Domain-specific concepts such as
  - activities
  - resources
  - precedence constraints
  - ....
- Encapsulate
  - variables and global constraints

Modeling

- Dedicated abstractions for scheduling
  - model-based computing
- Domain-specific concepts such as
  - activities
  - resources
  - precedence constraints
  - ....
- Encapsulate
  - variables and global constraints
- Support
  - search procedures

Jobshop Scheduling

- The "TSP" of scheduling
  - standard benchmarks and open problems

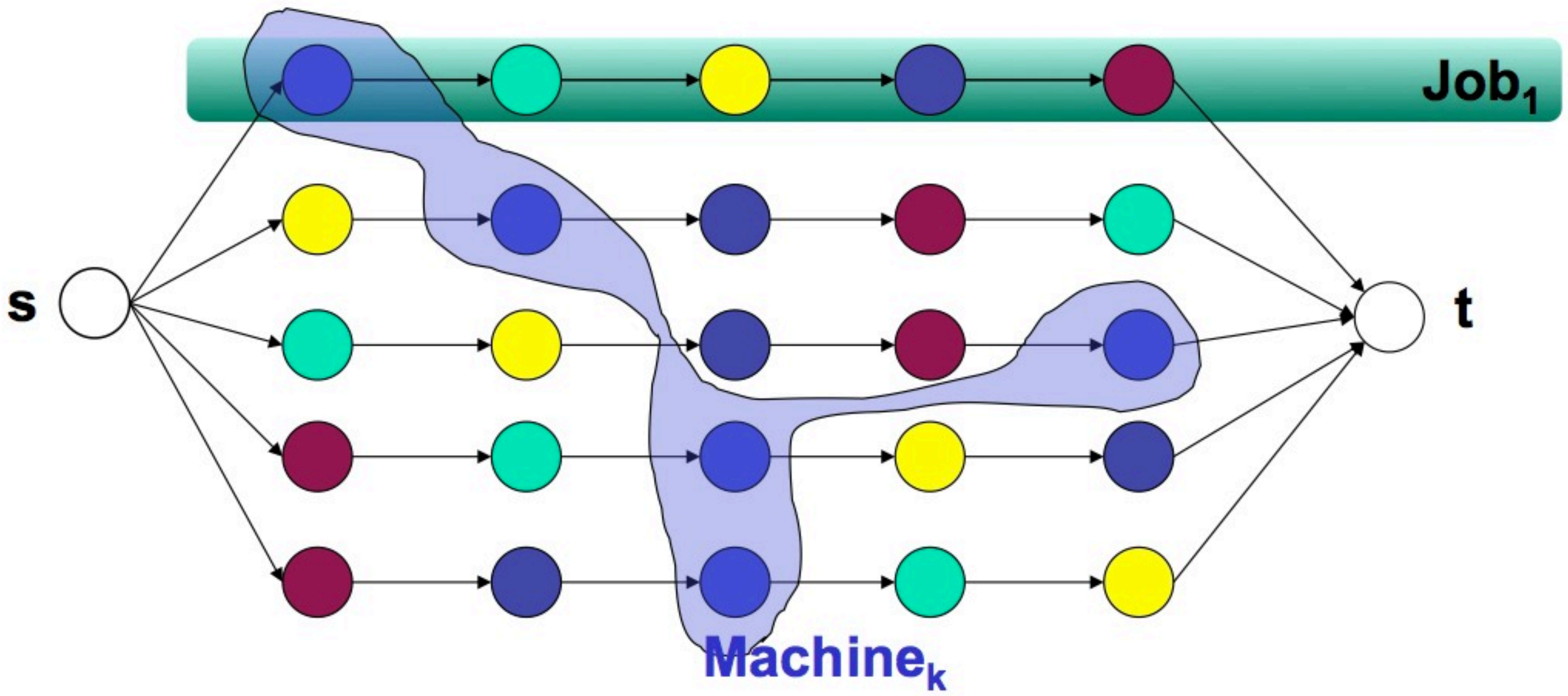
Jobshop Scheduling

- The “TSP” of scheduling
  - standard benchmarks and open problems
- Problem formulation
  - a set of tasks and
  - each task t has a duration d(t)
  - each task t executes on a machine m(t) and no two tasks scheduled on the same machine can overlap in time
  - a set of precedence constraints (b,a) stating that task a must start after task b has completed

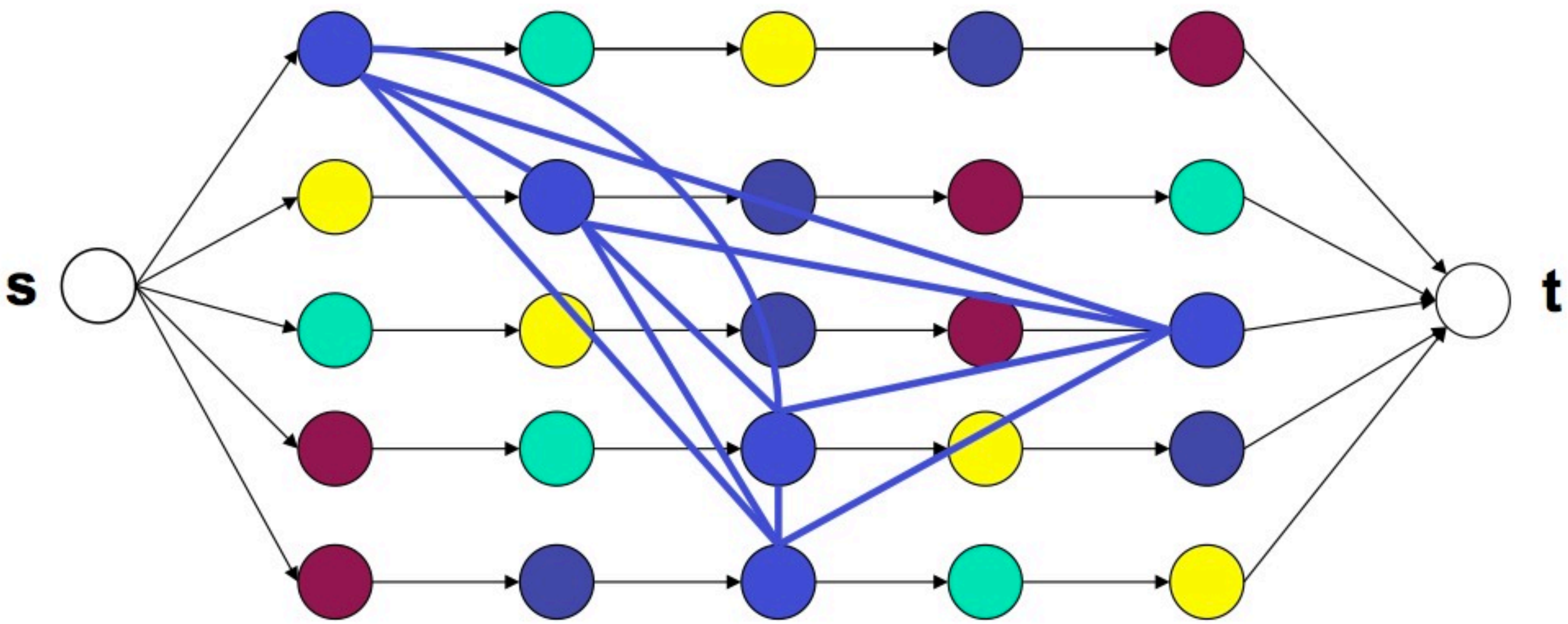
Jobshop Scheduling

- The “TSP” of scheduling
  - standard benchmarks and open problems
- Problem formulation
  - a set of tasks and
  - each task t has a duration d(t)
  - each task t executes on a machine m(t) and no two tasks scheduled on the same machine can overlap in time
  - a set of precedence constraints (b,a) stating that task a must start after task b has completed
- Objective
  - minimize the project completion time

Jobshop Scheduling

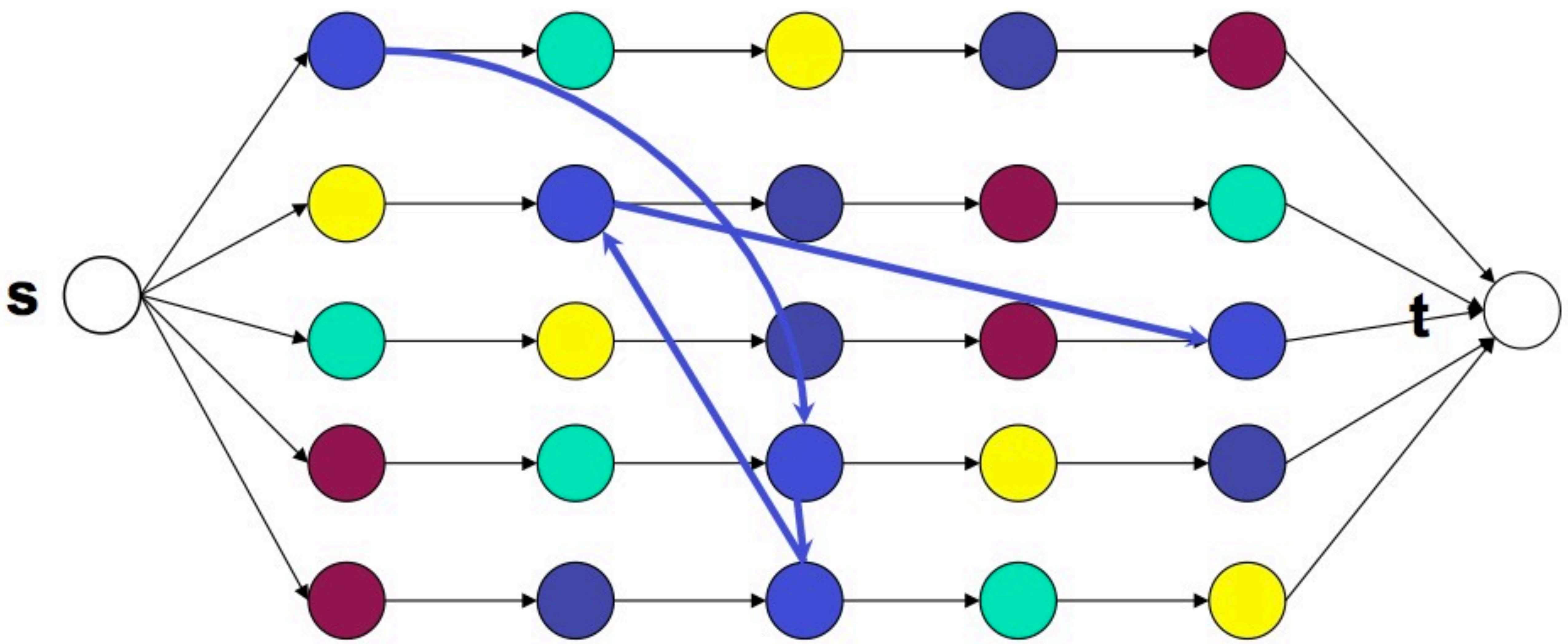


Jobshop Scheduling



- A machine must handle its tasks sequentially
  - a solution must find an ordering of the tasks on each machine

Jobshop Scheduling



- A machine must handle its task sequentially
  - a solution must find an ordering of the tasks on each machine

Jobshop Scheduling

- It is sufficient to order all machines
  - each ordering then introduces precedence constraints

Jobshop Scheduling

- It is sufficient to order all machines
  - each ordering then introduces precedence constraints
- Minimize project duration under precedence constraints
  - polynomial time
  - topological sorting (PERT)
  - transitive closure (Floyd-Warshall)

Jobshop Scheduling

```
int duration[Jobs,Tasks] = ...;
int machine[Jobs,Tasks] = ...;
int horizon = sum(j in Jobs,t in tasks) duration[j,t];

Scheduler sched(horizon);
Activity act[j in Jobs,t in Tasks](sched,duration[j,t]);
Activity makespan(sched,0);
UnaryResource r[Machines](sched);

minimize makespan.end
subject to {
   forall(j in Jobs,t in tasks: t != Tasks.high)
       act[j,t] precedes act[j,t+1];
   forall(j in Jobs)
       act[j,Tasks.high] precedes makespan;
   forall(j in Jobs,t in Tasks)
       act[j,t] requires r[machine[j,t]];
}
```

Jobshop Scheduling

```
int duration[Jobs,Tasks] = ...;
int machine[Jobs,Tasks] = ...;
int horizon = sum(j in Jobs,t in tasks) duration[j,t];

Scheduler sched(horizon);
Activity act[j in Jobs,t in Tasks](sched,duration[j,t]);
Activity makespan(sched,0);
UnaryResource r[Machines](sched);

minimize makespan.end
subject to {
   forall(j in Jobs,t in tasks: t != Tasks.high)
       act[j,t] precedes act[j,t+1];
   forall(j in Jobs)
       act[j,Tasks.high] precedes makespan;
   forall(j in Jobs,t in Tasks)
       act[j,t] requires r[machine[j,t]];
}
```

Jobshop Scheduling

```
int duration[Jobs,Tasks] = ...;
int machine[Jobs,Tasks] = ...;
int horizon = sum(j in Jobs,t in tasks) duration[j,t];

Scheduler sched(horizon);
Activity act[j in Jobs,t in Tasks](sched,duration[j,t]);
Activity makespan(sched,0);
UnaryResource r[Machines](sched);

minimize makespan.end
subject to {
   forall(j in Jobs,t in tasks: t != Tasks.high)
       act[j,t] precedes act[j,t+1];
   forall(j in Jobs)
       act[j,Tasks.high] precedes makespan;
   forall(j in Jobs,t in Tasks)
       act[j,t] requires r[machine[j,t]];
}
```

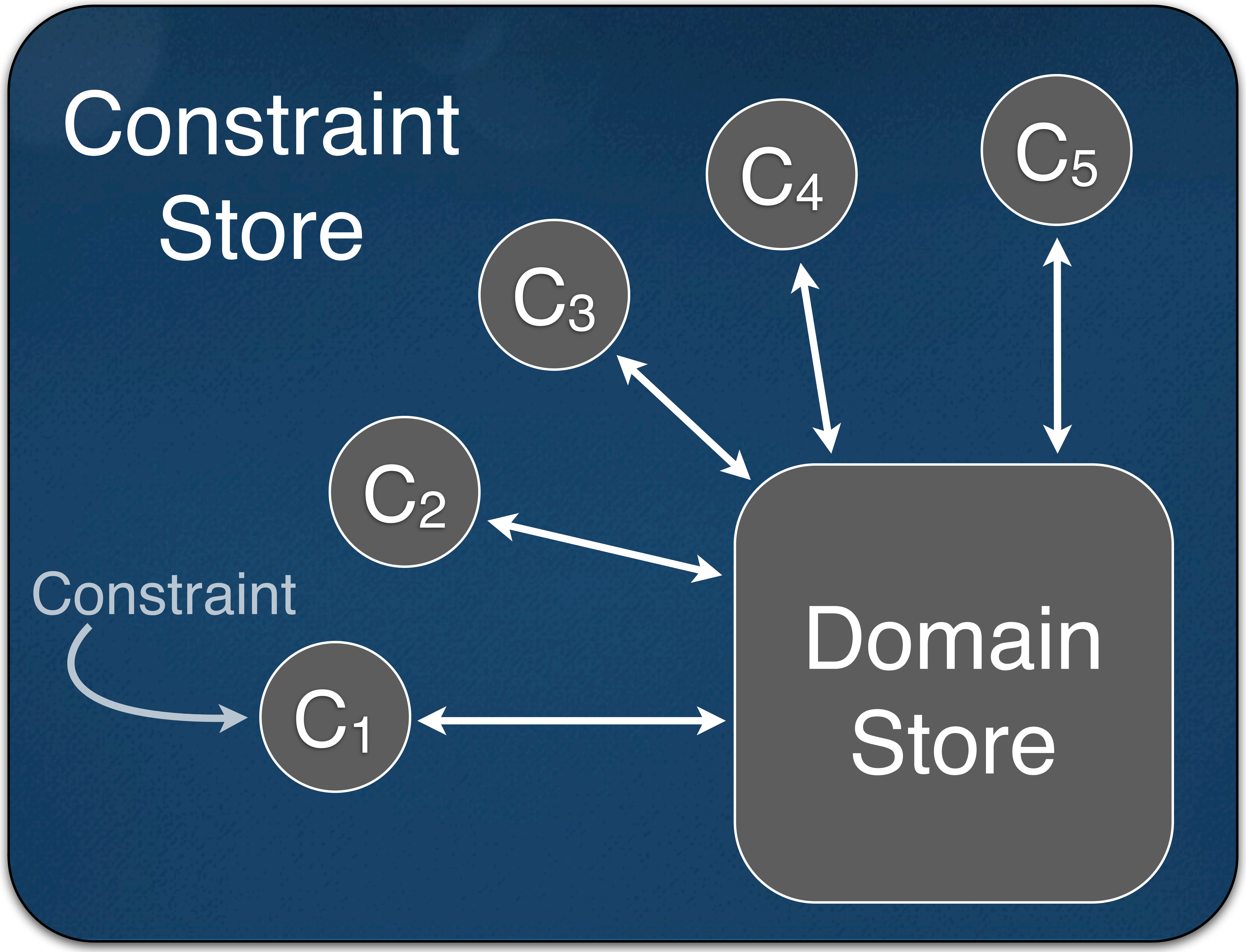
Jobshop Scheduling

```
int duration[Jobs,Tasks] = ...;
int machine[Jobs,Tasks] = ...;
int horizon = sum(j in Jobs,t in tasks) duration[j,t];

Scheduler sched(horizon);
Activity act[j in Jobs,t in Tasks](sched,duration[j,t]);
Activity makespan(sched,0);
UnaryResource r[Machines](sched);

minimize makespan.end
subject to {
   forall(j in Jobs,t in tasks: t != Tasks.high)
       act[j,t] precedes act[j,t+1];
   forall(j in Jobs)
       act[j,Tasks.high] precedes makespan;
   forall(j in Jobs,t in Tasks)
       act[j,t] requires r[machine[j,t]];
}
```

Model Compilation

- Each activity encapsulates
  - variables (e,s,d) for starting date, ending date, and duration
  - a constraint linking these three variables
    - $s + d = e$
- Each precedence constraint (b,a)
  - $s_a \geq e_b$
- Each machine m gives rise to a global constraint
  - disjunctive($t_1,\ldots,t_n$)

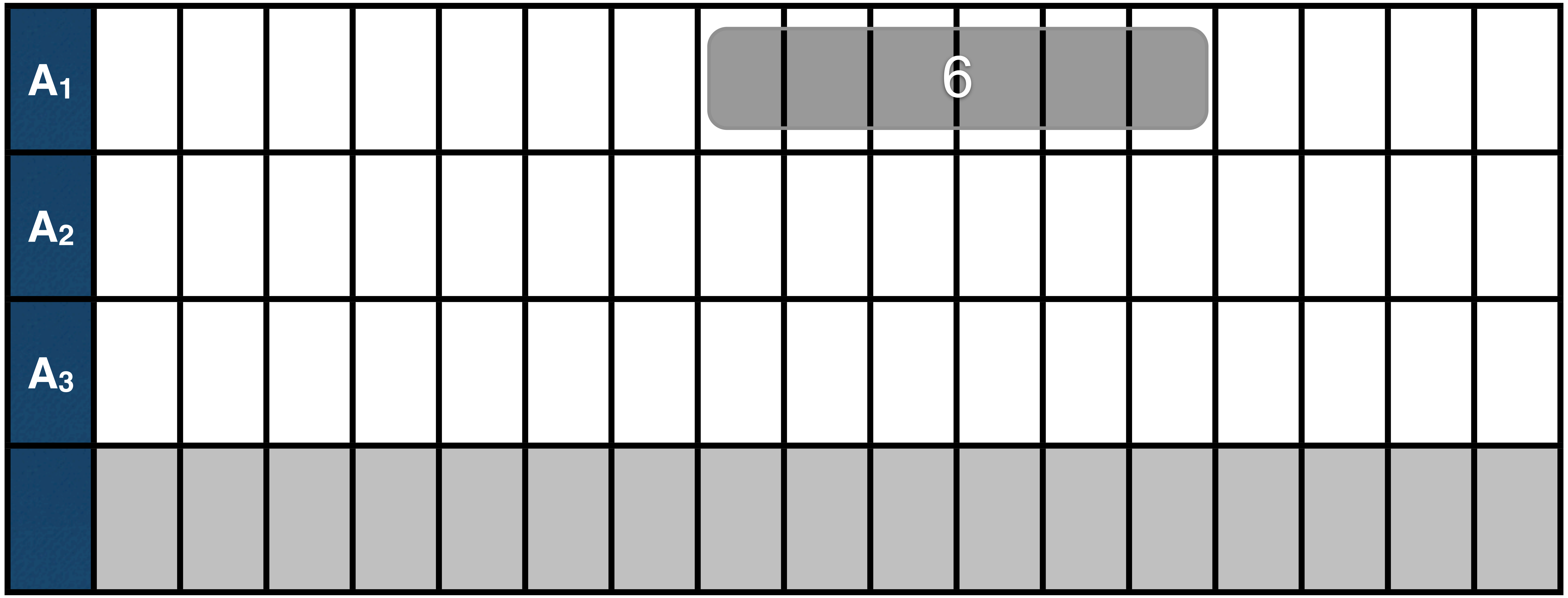
Jobshop Scheduling



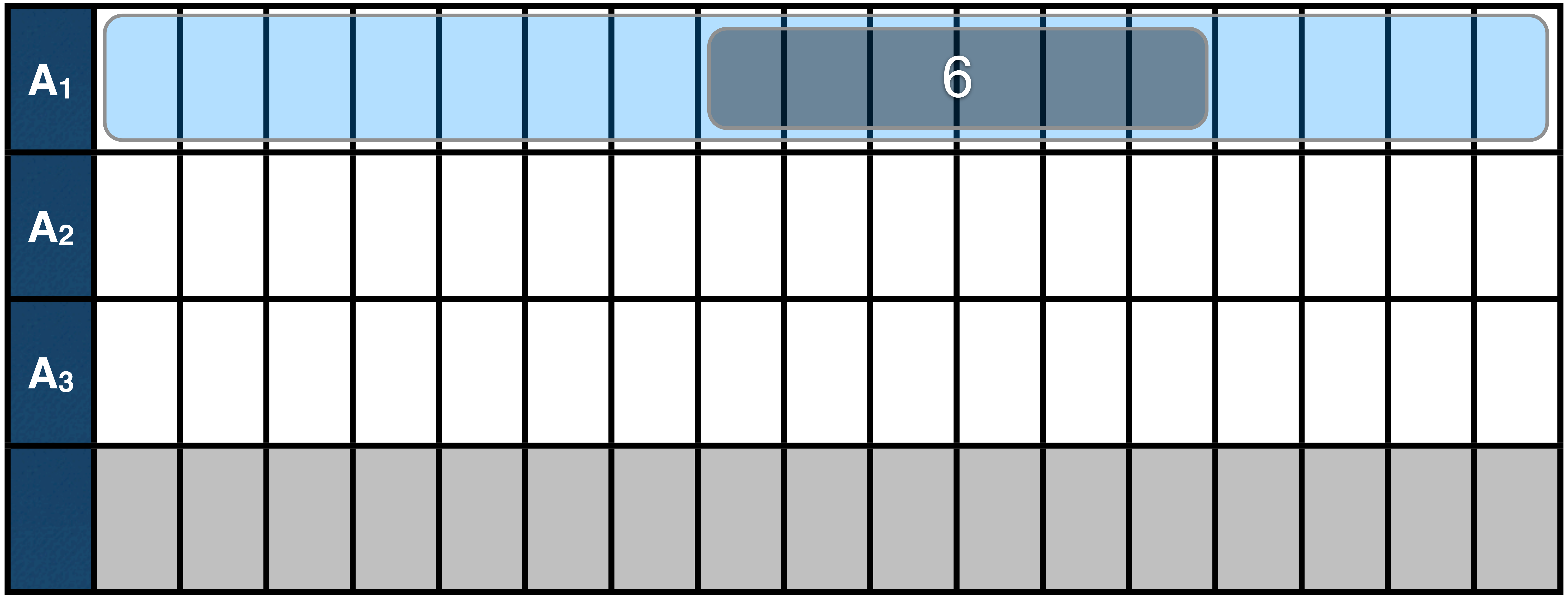
Disjunctive Constraint: Feasibility

A_1																	
A_2																	
A_3																	

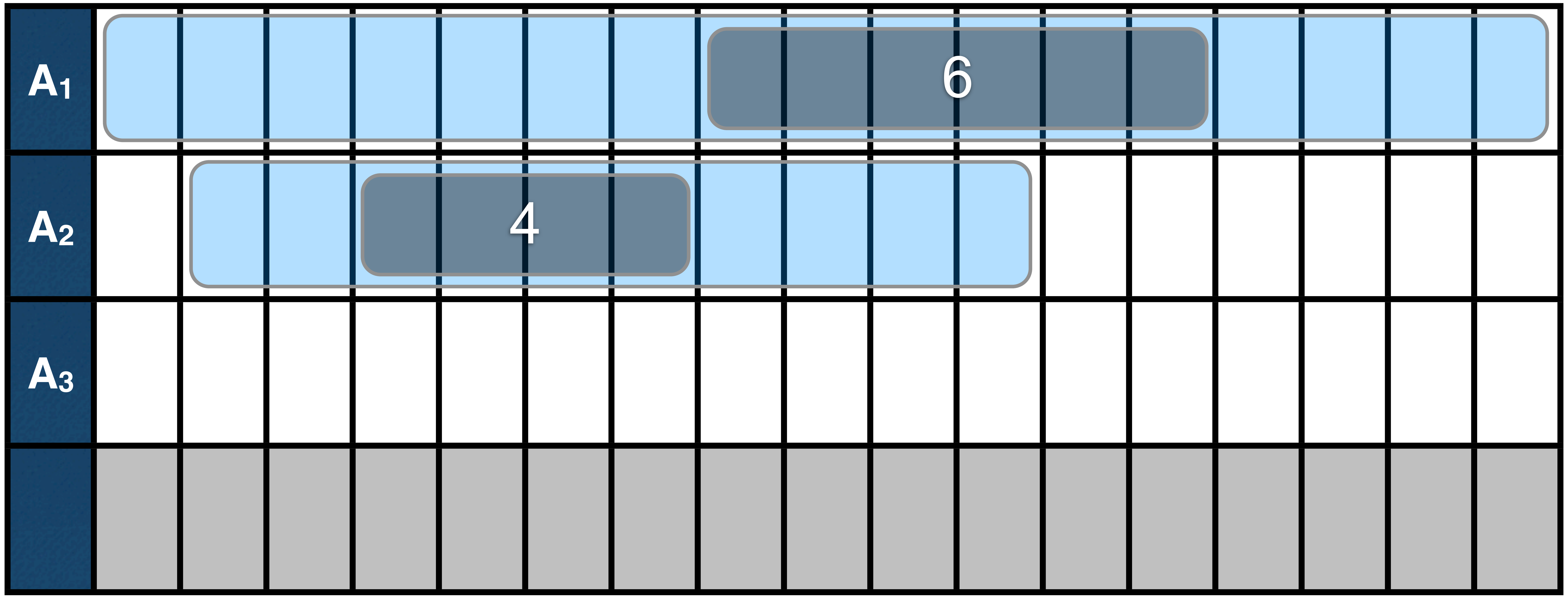
Disjunctive Constraint: Feasibility



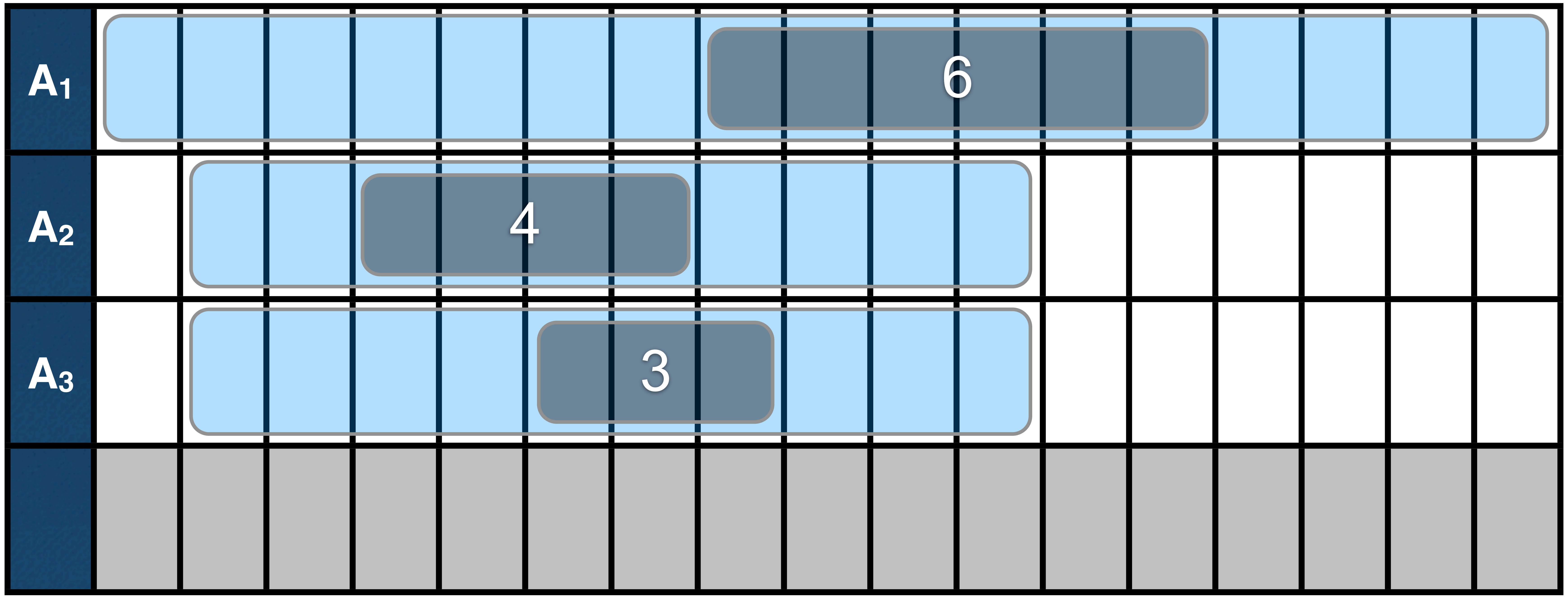
Disjunctive Constraint: Feasibility



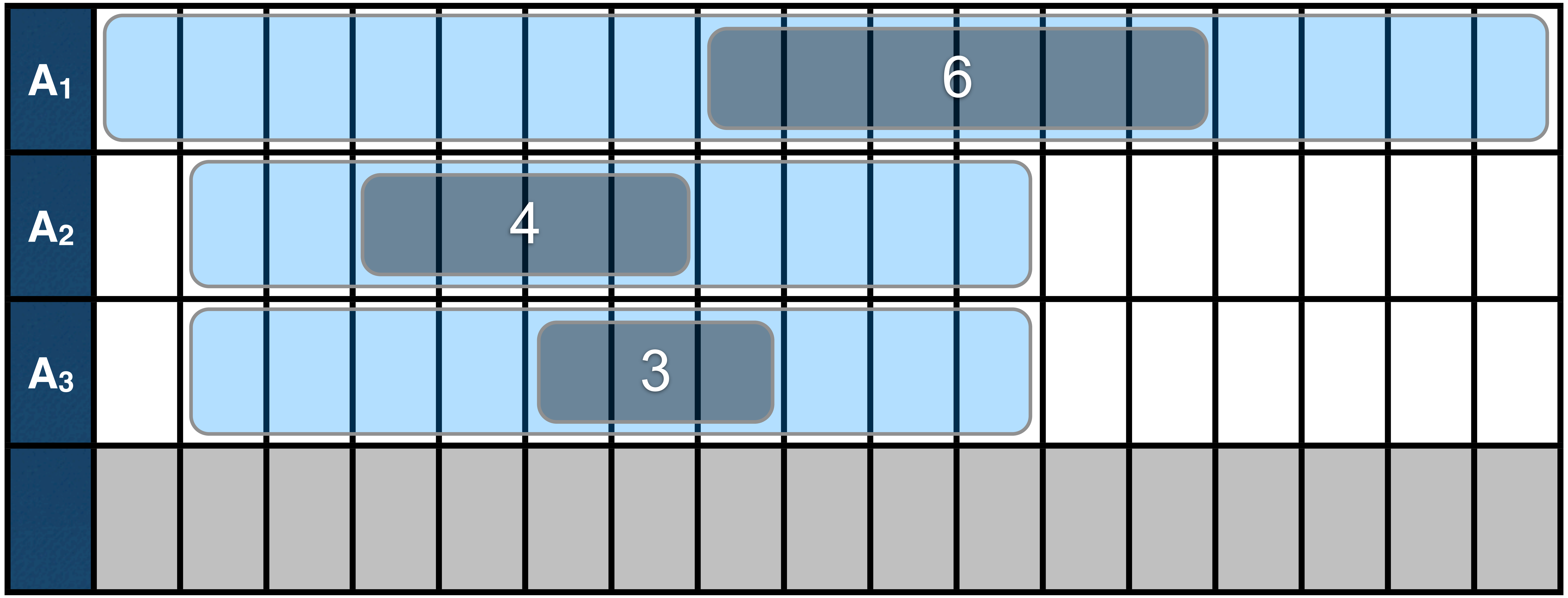
Disjunctive Constraint: Feasibility



Disjunctive Constraint: Feasibility



Disjunctive Constraint: Feasibility



- Detecting feasibility of a disjunctive constraint is NP-Complete

Feasibility of Disjunctive Constraints

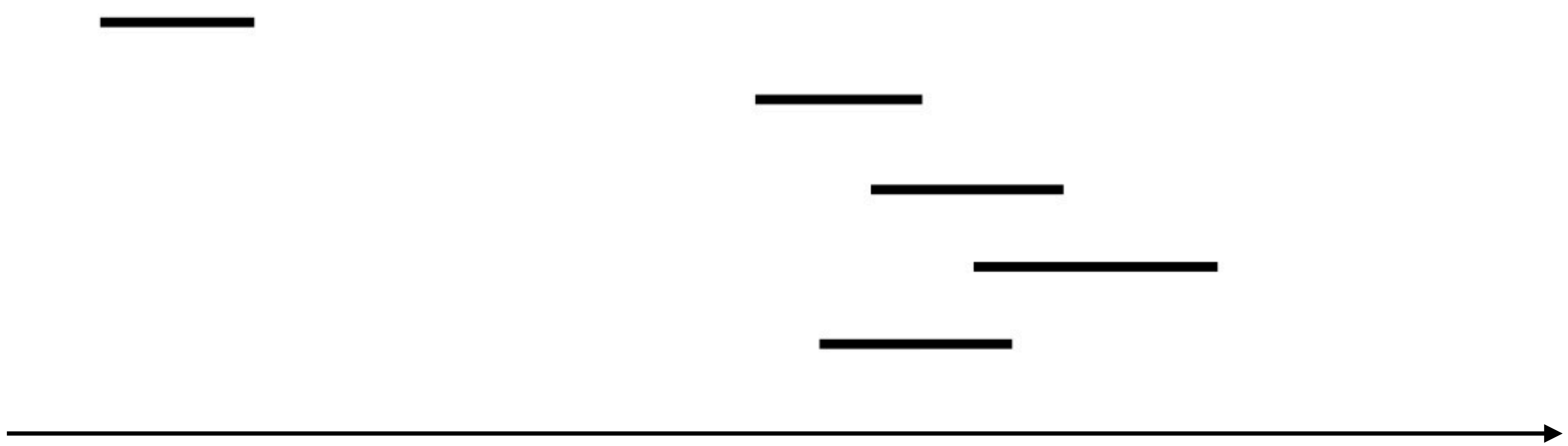
- Some basic intuition and algorithms
  - very rich domain
  - just making you curious
  - many interesting connections
- Notations
  - $s(\Omega)$ = min(t in Ω) min(s_t)
  - $e(\Omega)$ = max(t in Ω) max(e_t)
  - $d(\Omega)$ = sum(t in Ω) min(d_t)

Disjunctive Feasibility

- Feasibility test: tasks T
  - $s(T) + d(T) \leq e(T)$

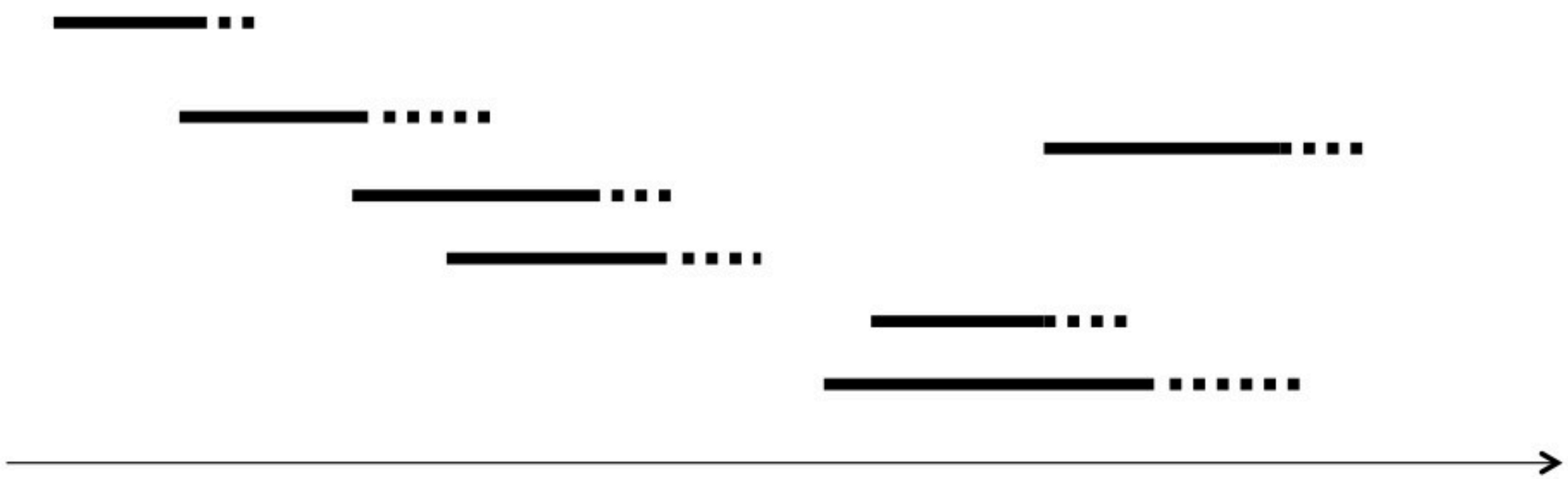
Disjunctive Feasibility

- Feasibility test: tasks T
  - $s(T) + d(T) \leq e(T)$



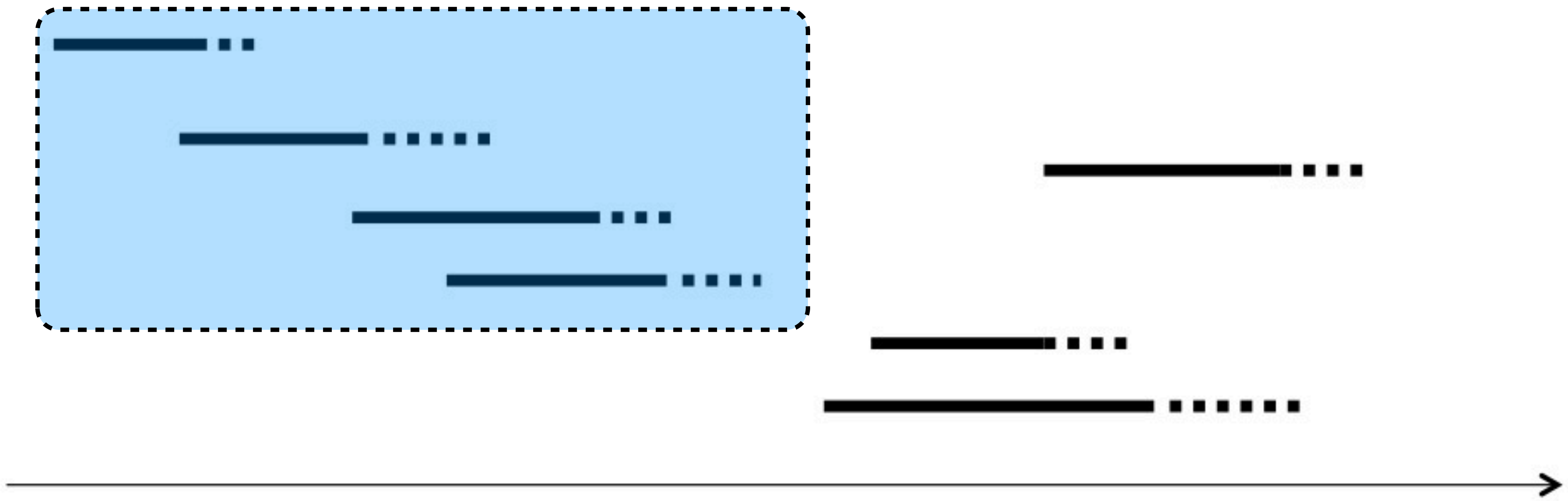
Disjunctive Feasibility

- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$



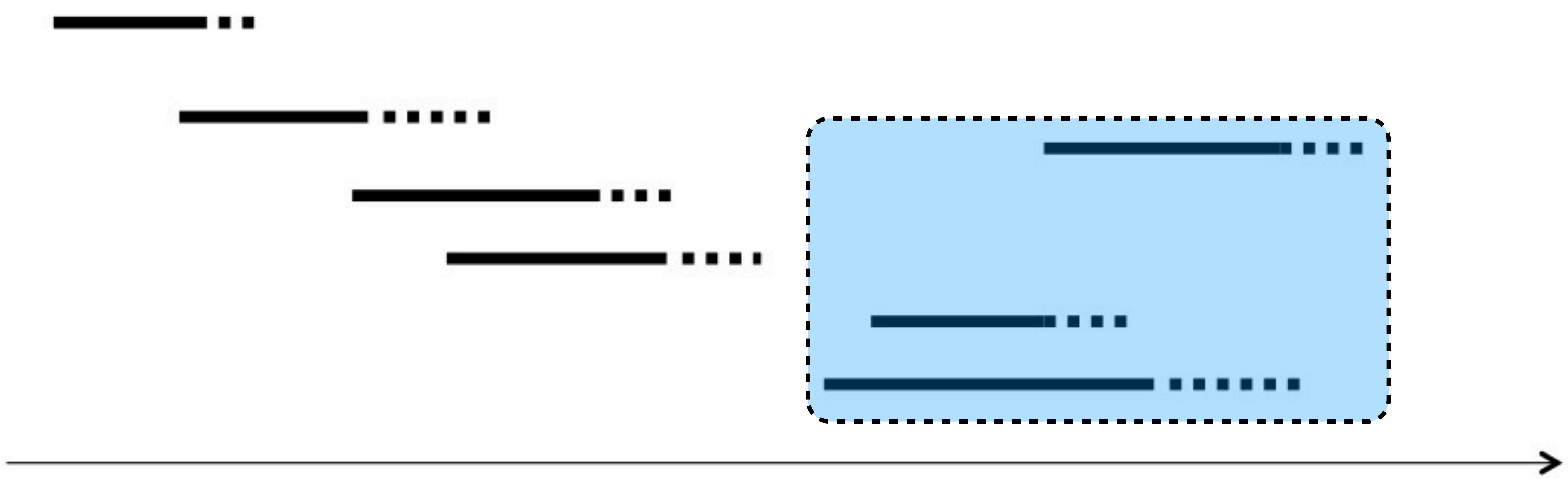
Disjunctive Feasibility

- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$



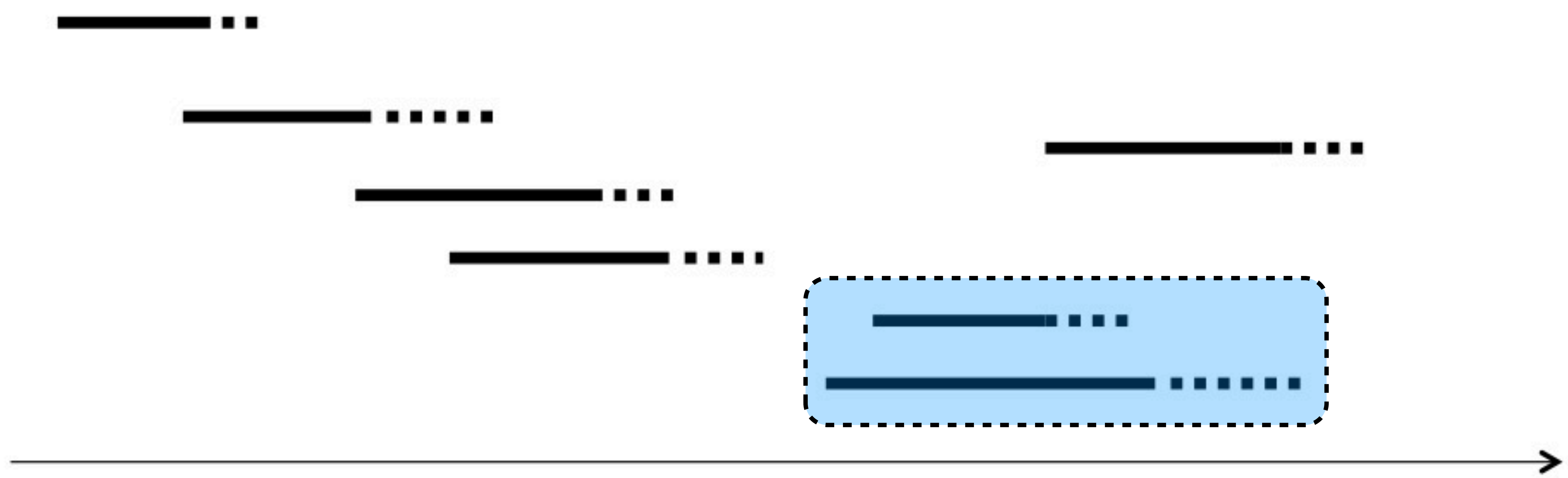
Disjunctive Feasibility

- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$



Disjunctive Feasibility

- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$

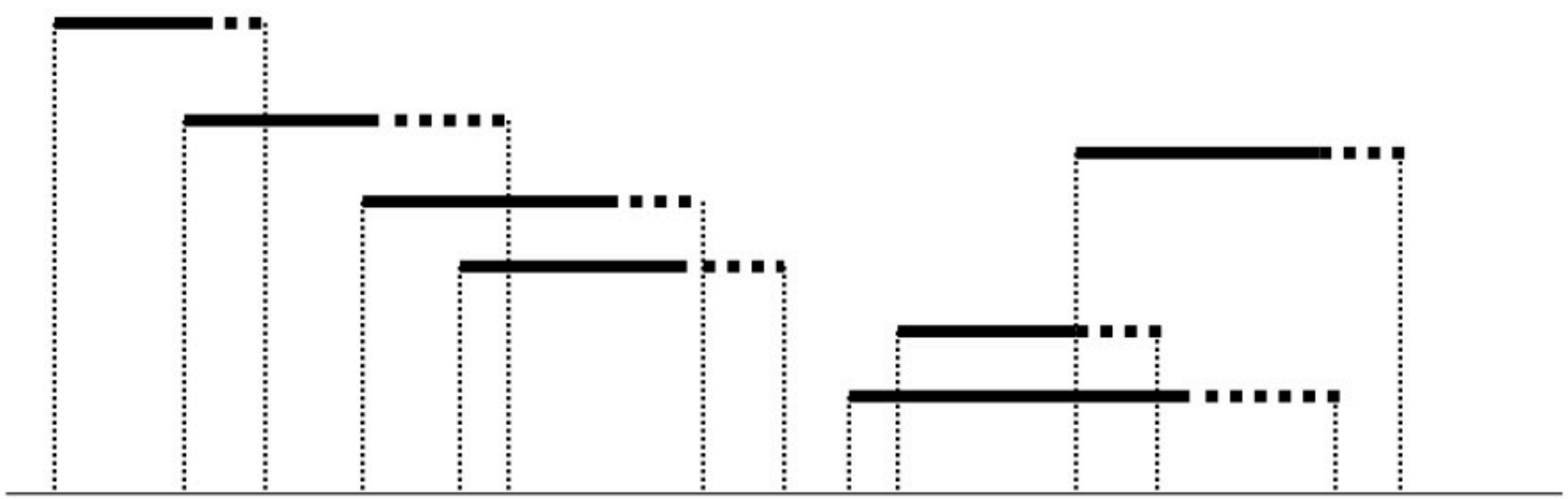


Disjunctive Feasibility

- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$
- What is the issue here?

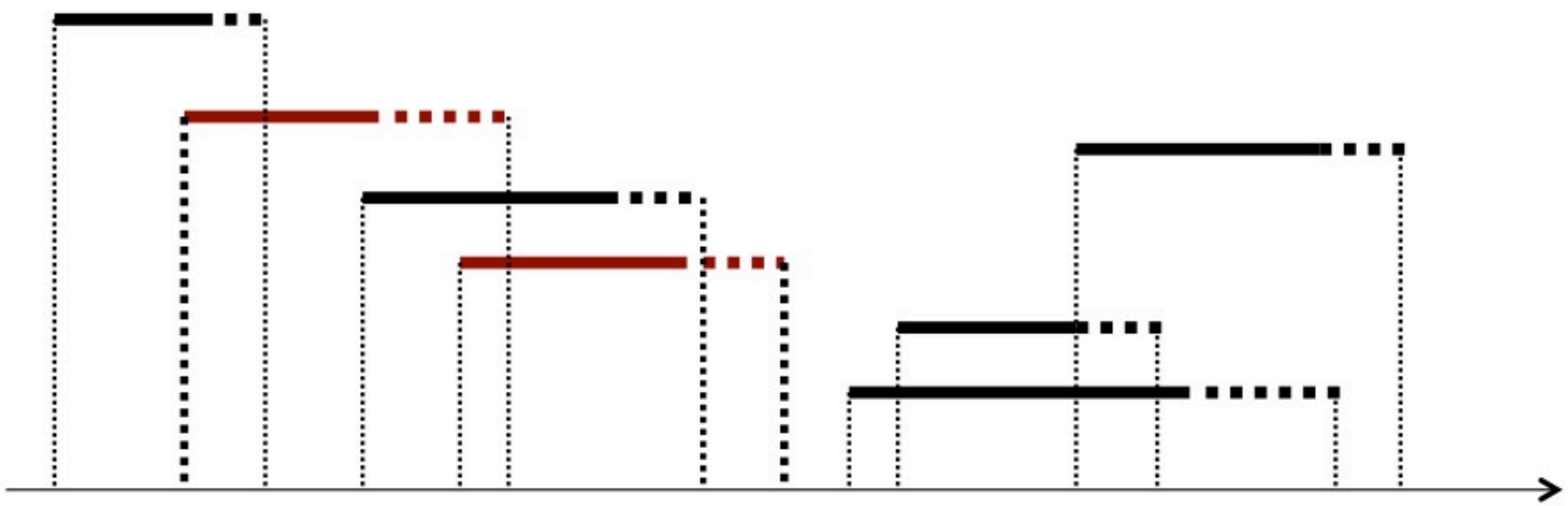
Disjunctive Feasibility

- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$
- We only need to look at a quadratic number of subsets.



Disjunctive Feasibility

- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$
- We only need to look at a quadratic number of subsets.



Disjunctive Feasibility

- Task intervals
  - $S(t_1,t_2) = \{ t \text{ in } R \mid s(t) \geq s(t_1) \ \& \ e(t) \leq e(t_2) \}$

Disjunctive Feasibility

- Task intervals
  - $S(t_1,t_2) = \{ t \text{ in } R \mid s(t) \geq s(t_1) \; \& \; e(t) \leq e(t_2) \}$
- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$

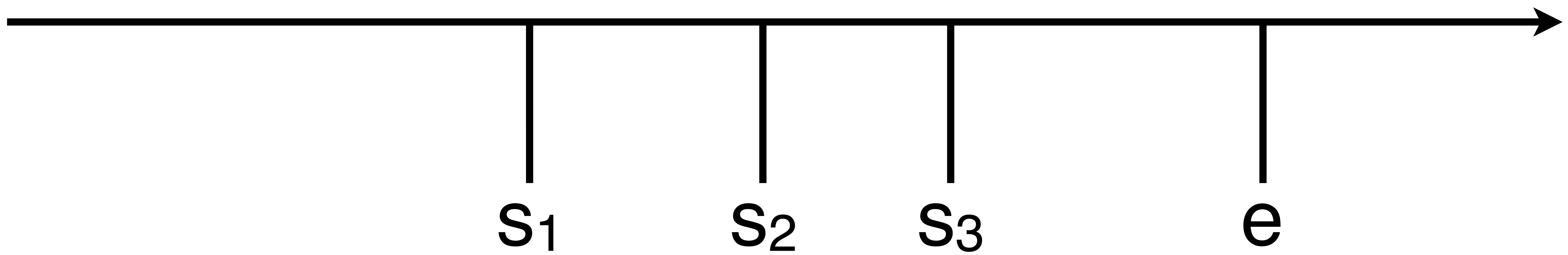
Disjunctive Feasibility

- Task intervals
  - $S(t_1,t_2) = \{ t \text{ in } R \mid s(t) \geq s(t_1) \ \& \ e(t) \leq e(t_2) \}$
- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$
- Implementation
  - apply the feasibility tests for all task intervals $S(t_1,t_2)$ with t_1,t_2 in T

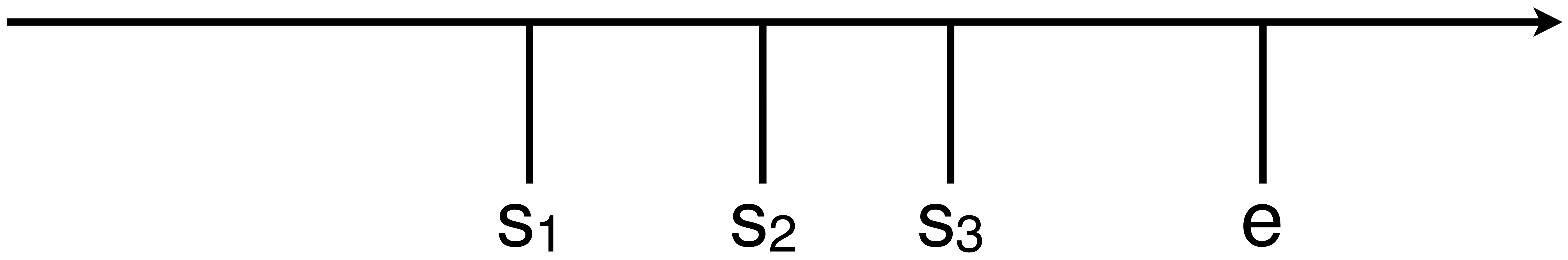
Disjunctive Feasibility

- Task intervals
  - $S(t_1,t_2) = \{ t \text{ in } R \mid s(t) \geq s(t_1) \ \& \ e(t) \leq e(t_2) \}$
- A better feasibility test: tasks T
  - for all $\Omega \subseteq T$: $s(\Omega) + d(\Omega) \leq e(\Omega)$
- Implementation
  - apply the feasibility tests for all task intervals $S(t_1,t_2)$ with t_1,t_2 in T
- Complexity
  - $O(|T|^3)$

Disjunctive Feasibility



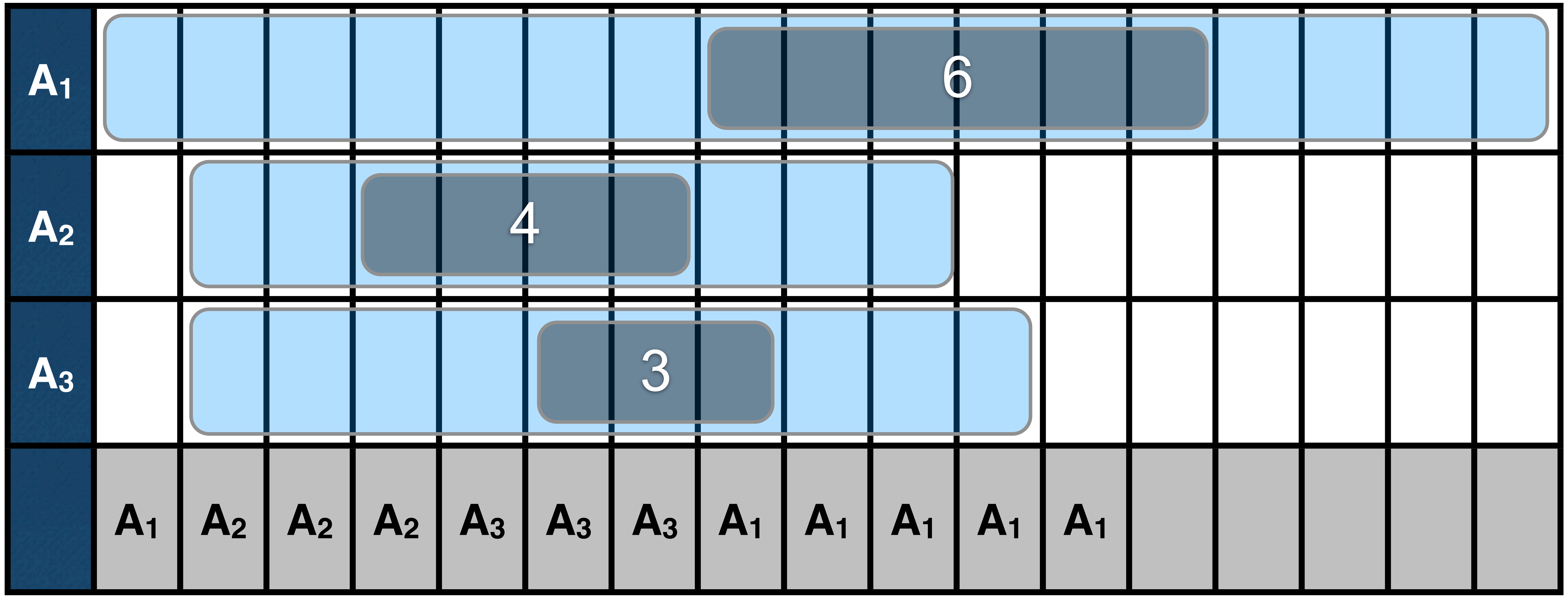
Disjunctive Feasibility




```
d := 0;
for each task t in decreasing order of st
   if et <= e
     d := d + dt;
     if st + d > e
       return failure;
return success;
```

Disjunctive Constraint: Feasibility

- Relax: feasibility of the preemptive schedule



- One-machine preemptive feasibility can be computed in O(|T| log |T|)

Disjunctive Pruning

- Edge finding rules
  - select a set Ω of tasks and a task $i \notin \Omega$

Disjunctive Pruning

- Edge finding rules
  - select a set Ω of tasks and a task $i \notin \Omega$
- Determine if i must start after all tasks in Ω
  - $s(\Omega \cup \{i\}) + d(\Omega \cup \{i\}) > e(\Omega)$

Disjunctive Pruning

- Edge finding rules
  - select a set Ω of tasks and a task $i \notin \Omega$
- Determine if i must start after all tasks in Ω
  - $s(\Omega \cup \{i\}) + d(\Omega \cup \{i\}) > e(\Omega)$
- Update the starting times of i to
  - $\max(\gamma \subseteq \Omega)\ s(\gamma) + d(\gamma)$

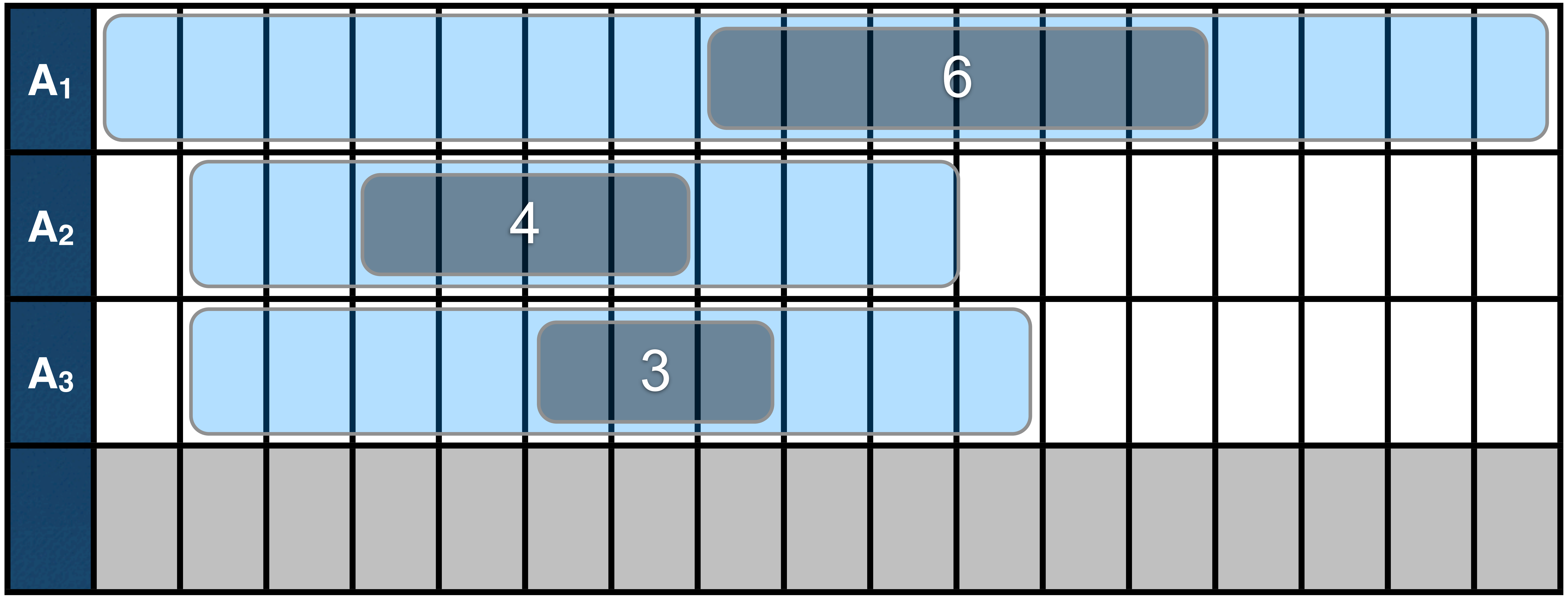
Disjunctive Pruning

- Edge finding rules
  - select a set Ω of tasks and a task $i \notin \Omega$
- Determine if i must start after all tasks in Ω
  - $s(\Omega \cup \{i\}) + d(\Omega \cup \{i\}) > e(\Omega)$
- Update the starting times of i to
  - $\max(\gamma \subseteq \Omega)\ s(\gamma) + d(\gamma)$
- Same for the ending dates

Disjunctive Pruning

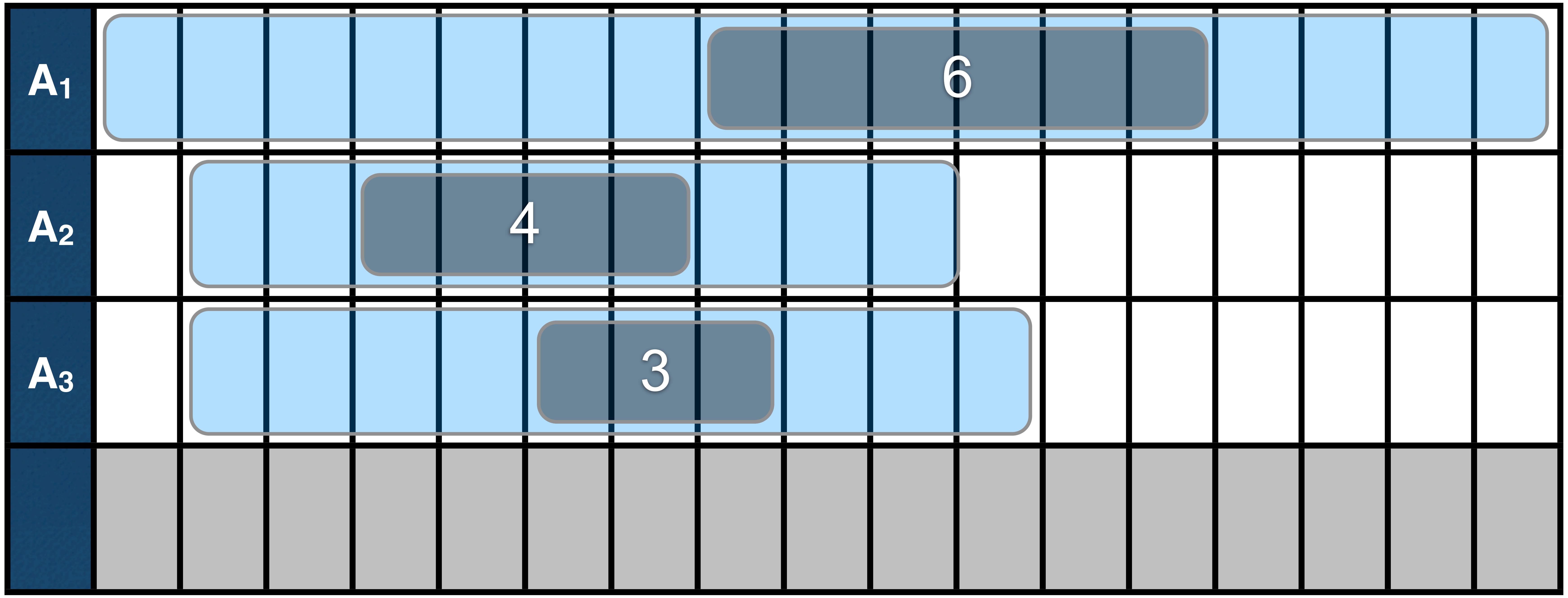
- Edge finding rules
  - select a set Ω of tasks and a task $i \notin \Omega$
- Determine if i must start after all tasks in Ω
  - $s(\Omega \cup \{i\}) + d(\Omega \cup \{i\}) > e(\Omega)$
- Update the starting times of i to
  - $\max(\gamma \subseteq \Omega)\ s(\gamma) + d(\gamma)$
- Same for the ending dates
- The edge finding rules can be enforced in strongly polynomial time

Disjunctive Pruning



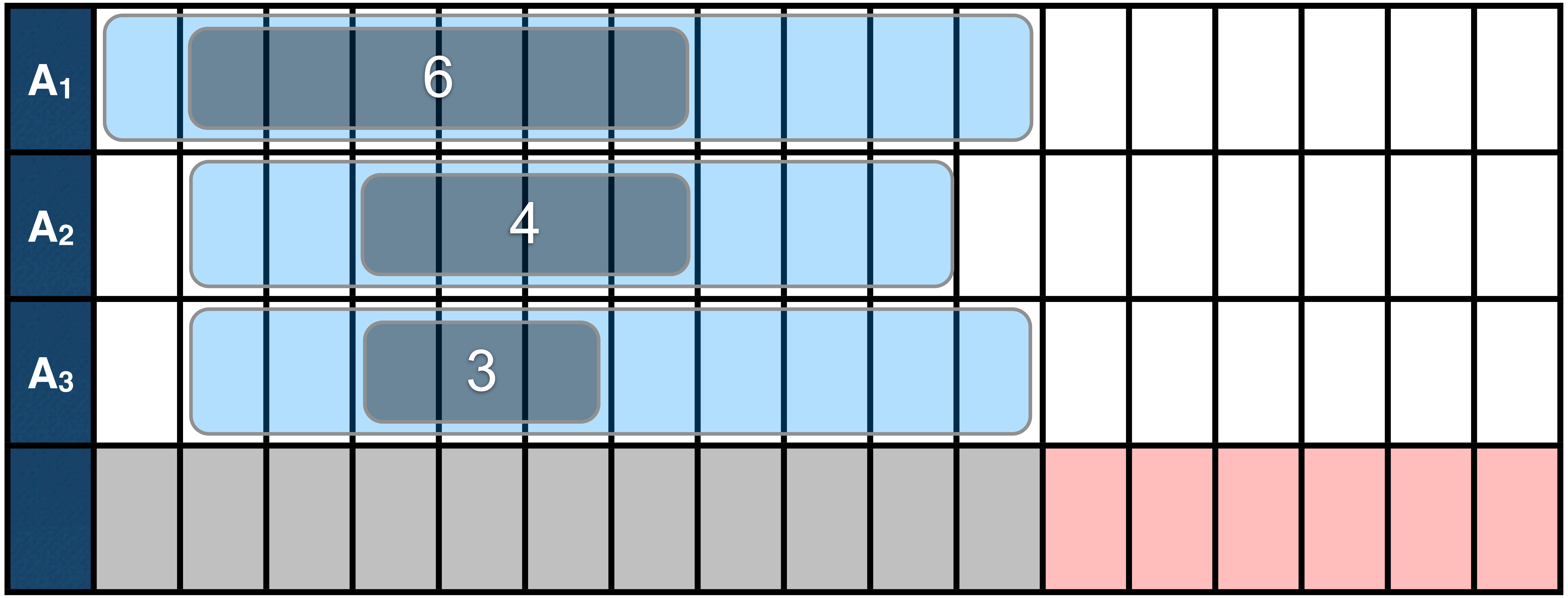
Disjunctive Pruning

- Can A_1 start before A_2 or A_3?



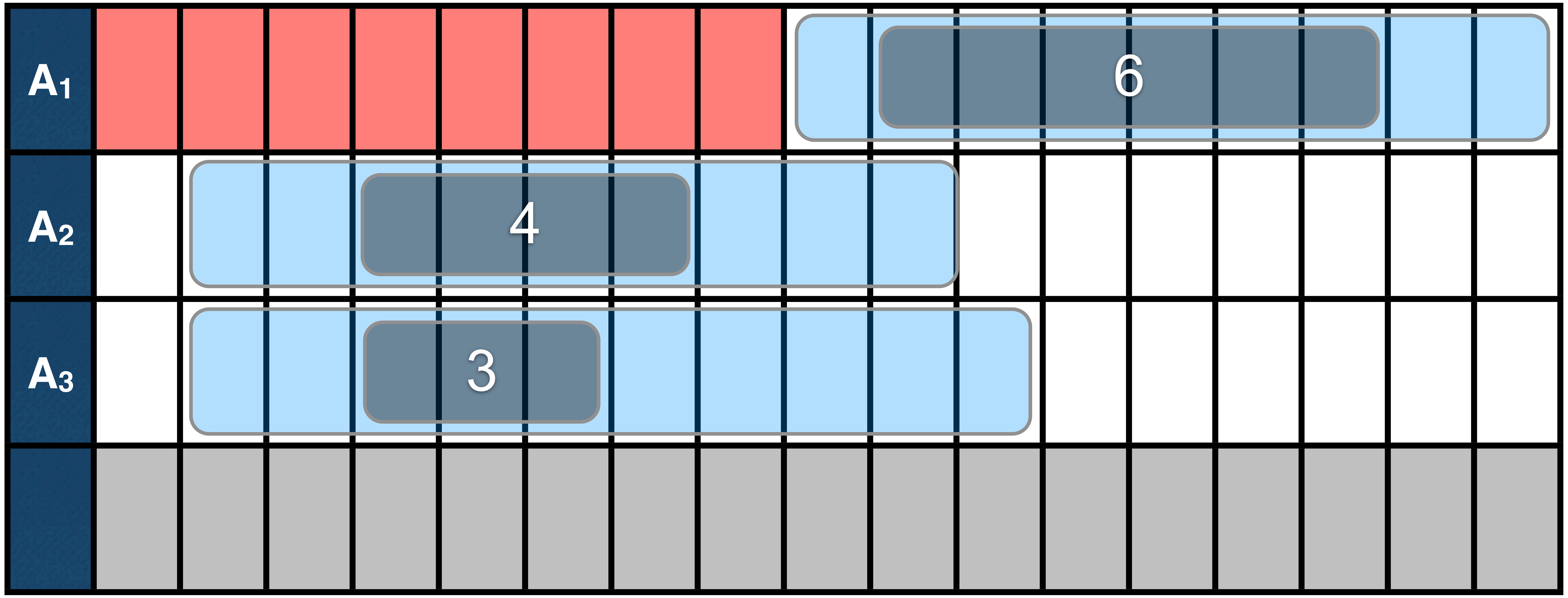
Disjunctive Pruning

- Can A_1 start before A_2 or A_3?



Disjunctive Pruning

- A1 must start after A2 and A3



Search for Disjunctive Scheduling

- Basic strategy
  - choose a machine
  - sequence that machine
  - repeat

Search for Disjunctive Scheduling

- Basic strategy
  - choose a machine
  - sequence that machine
  - repeat
- Which machine?
  - first-fail principle: the tightest machine

Search for Disjunctive Scheduling

- Basic strategy
  - choose a machine
  - sequence that machine
  - repeat
- Which machine?
  - first-fail principle: the tightest machine
- Which task?
  - a task that can be scheduled first (or last)
  - a task that is as tight as possible

Until Next Time