Sudoku, Square Tiling / Scheduling

CP Course, Lecture 10

Sudoku

Fill in the grid such that every row, every column and every 3x3 box contains the digits 1...9

	9		6		4		8	
		8	5		8	9		
7								6
8			4		7			9
		9				5		
7			9		6			4
8								7
		7	7		9	9		
	4		8		8		7	

Sudoku: Model

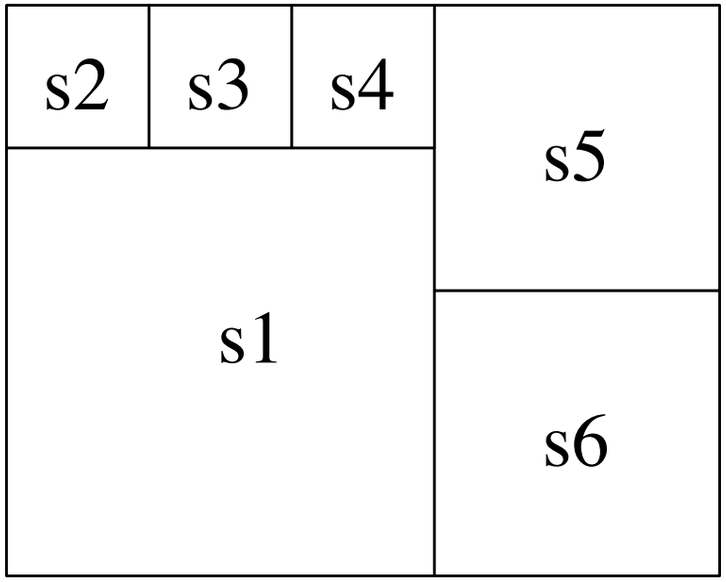
- fd var for each cell
- for each row and column: distinct
- for each 3x3 box: distinct

	9		6		4		8	
		8	5		8	9		
7								6
8			4		7			9
		9				5		
7			9		6			4
8								7
		7	7		9	9		
	4		8		8		7	

Solves most instances without search!

Square Tiling

- Given: *n*×*m* board, squares defined by dimension.
- Goal: place all squares on the board - no overlaps, board fully covered.



Square	Size
s1	3
s2	1
s3	1
s4	1
s5	2
s6	2

Naive model

- For each square *s*, introduce fd vars for s_x and s_y
- Express with reification that two squares *s* and *t* do not overlap:

  s left of *t*, or *t* left of *s*, or *s* above *t*, or *t* above

- In a column with coordinate *x*, the sum of the sizes occupying space at col. *x* must be the size of the board in *y*-direction. (Similar for rows)

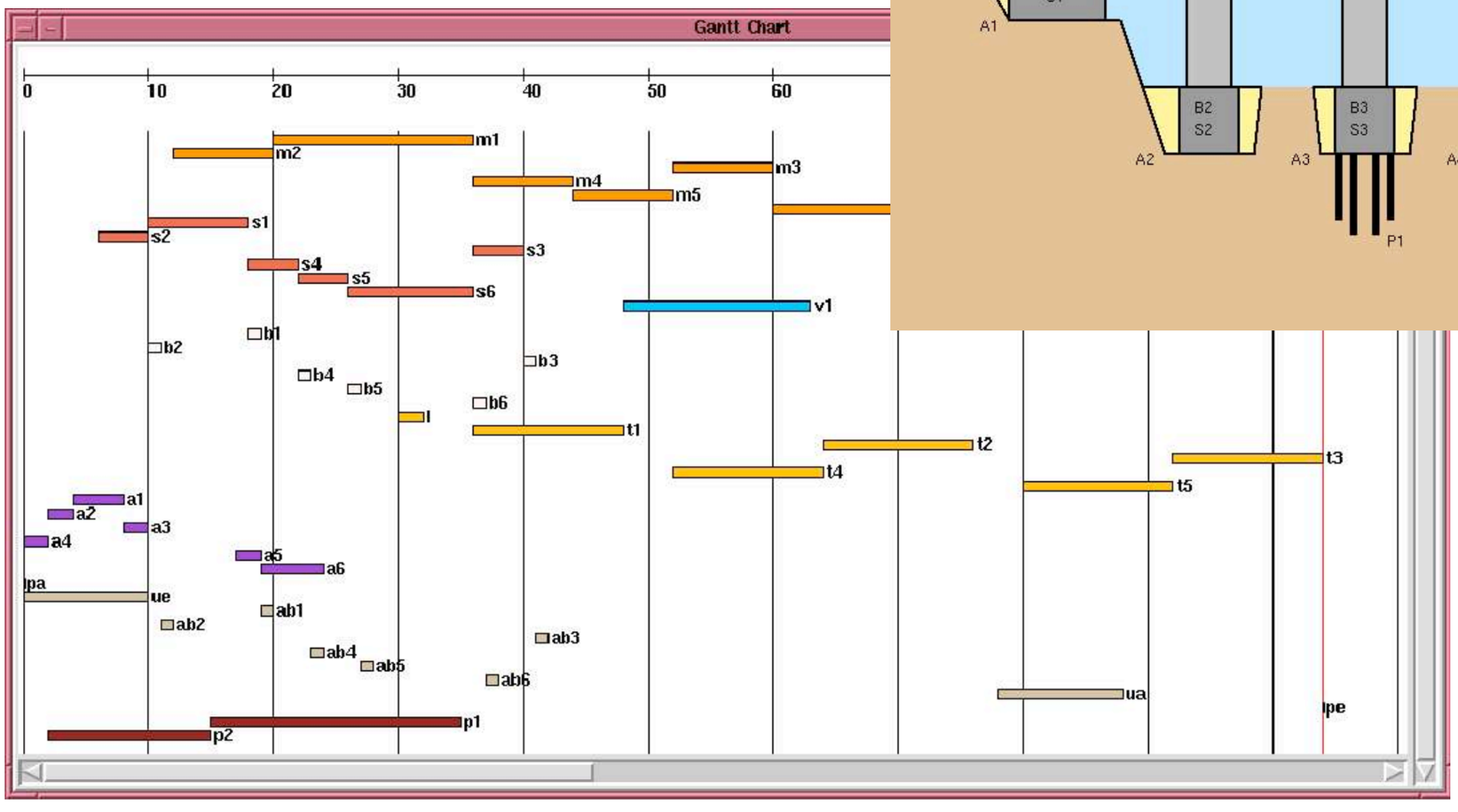
Improvements

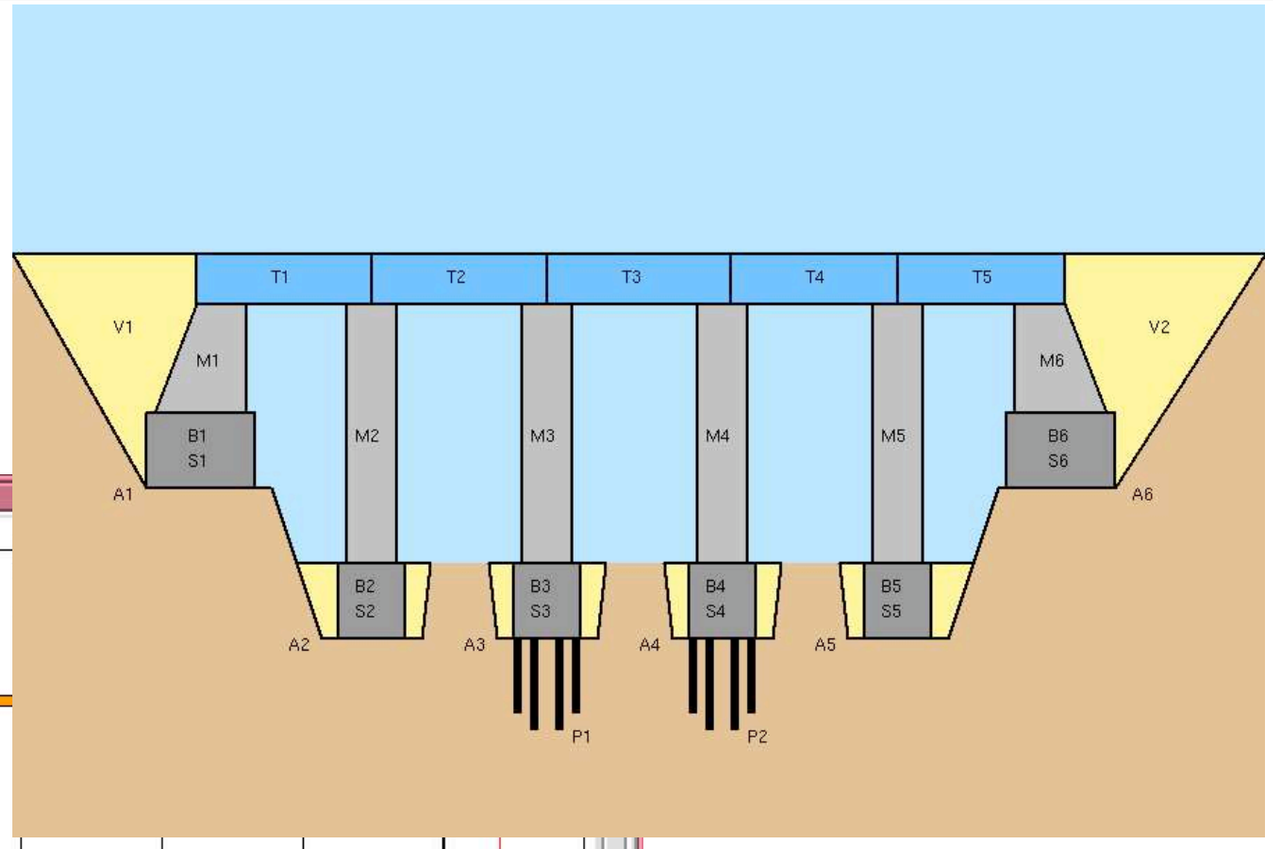
- Branching:
  1. Assign *x*-coordinates, then *y*-coordinates
  2. Try bigger squares first
  3. Place from left to right and top to bottom
- Symmetry breaking?

Scheduling

- Tasks *a* (aka activities)
  - duration *dur(a)*
  - resource *res(a)*
- Precedence constraints
  - determine order among two tasks
- Resource constraints
  - e.g. at most one task per resource

Building a Bridge





Application Areas

- creating time tables
- planning workflow
- scheduling instruction sequences in a compiler
- ...

Model in CP

- Variable for start-time of task *a*
- Precedence constraints:

$$a \textbf{ before } b$$

- Resource constraints:

$$a \textbf{ before } b \ \vee \ b \textbf{ before } a$$

similar to temporal relations

Model in CP

- Variable for start-time of task *a*
- Precedence constraints:

$$start(a) + dur(a) \leq start(b)$$

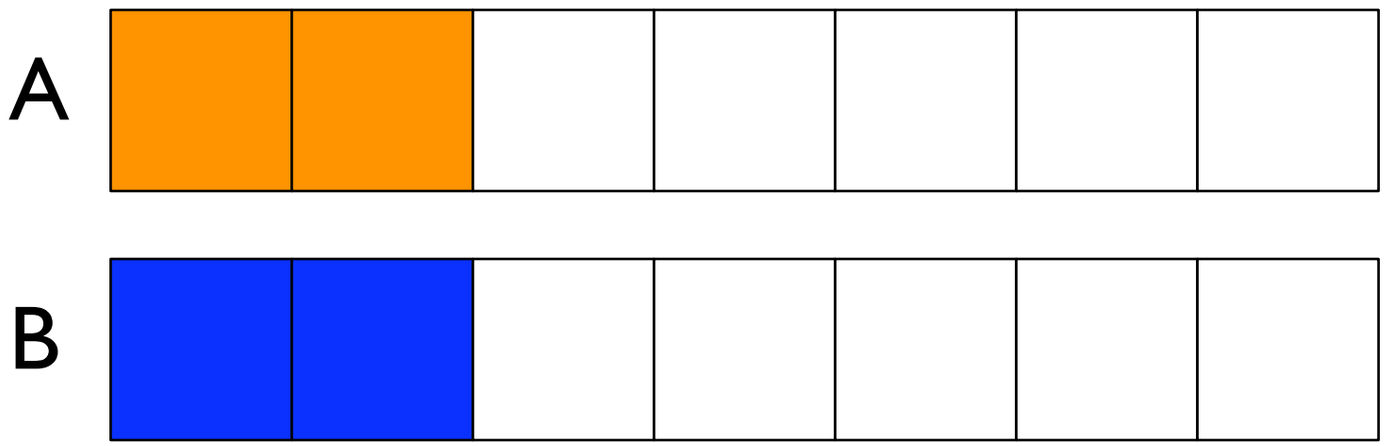
- Resource constraints:

$$a \textbf{ before } b \ \vee \ b \textbf{ before } a$$

reification

Propagate Precedence

A before B



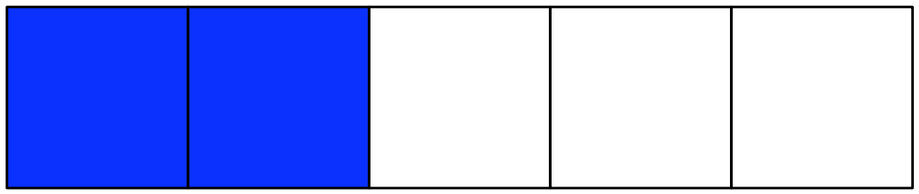
$start(A) \in \{0,\ldots,5\}$ $dur(A) = 2$

$start(B) \in \{0,\ldots,5\}$ $dur(B) = 2$

Propagate Precedence

A before B

A 

B 

$start(A) \in \{0,\ldots,3\}$ $dur(A) = 2$

$start(B) \in \{2,\ldots,5\}$ $dur(B) = 2$

So what's new?

- We are interested in the *concrete start and end times*
- We want to *optimize*

  (e.g. find earliest completion time)
- This makes the problem hard

Classes of problems I

- Resource type

  disjunctive (at most one task at a time)

  cumulative (fixed capacity per resource)

- Task type

  non-preemptive (not interruptible)

  preemptive

  e*lastic* (stretchable)

Classes of problems II

- Optimization: Minimize

  makespan (latest end time of any task)

  number of late jobs (that miss their due date)

  ...

- Give more weight to more important jobs

Special case we discuss

- disjunctive, non-preemptive
- cumulative (briefly)

Baptiste, Le Pape, Nuijten, Constraint-based Scheduling. Kluwer, 2001.

Again: Model in CP

- Variable for start-time of task *a*
- Precedence constraints:

$$start(a) + dur(a) \leq start(b)$$

- Resource constraints:

$$a \textbf{ before } b \ \vee \ b \textbf{ before } a$$

reification

Think global

- Model employs local view:
  - constraints on pairs of tasks
    - $O(n^2)$ propagators for n tasks
- Global view:
  - order all tasks on one resource
  - employ smart global propagator

Ordering Tasks: Serialization

- Consider all tasks on one resource
- Deduce their order as much as possible
- Propagators:
  - Timetabling: look at free/used time slots
  - Edge-finding: which task first/last?
  - Not-first / not-last

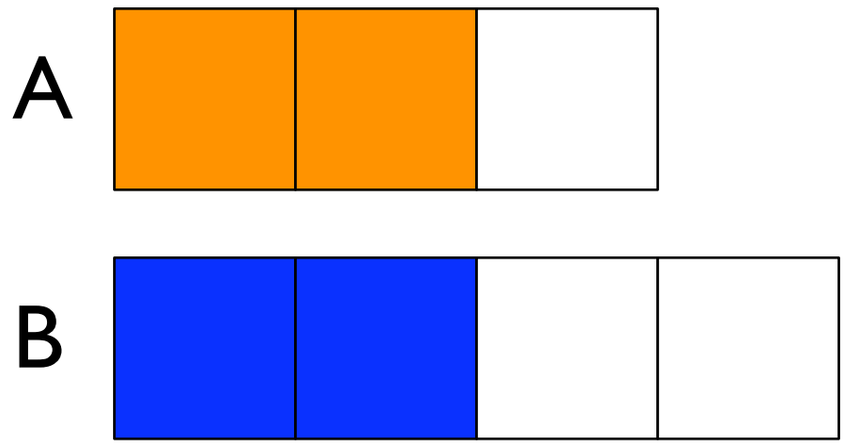
Special case

- Consider disjunctive, non-preemptive scheduling where the duration is one for all tasks
- Do you know a good propagator for serialization?

Timetable propagation

- Timetable: data structure that records per resource where some task definitively uses the resource
- Propagate
  - from tasks to timetable
  - from timetable to tasks

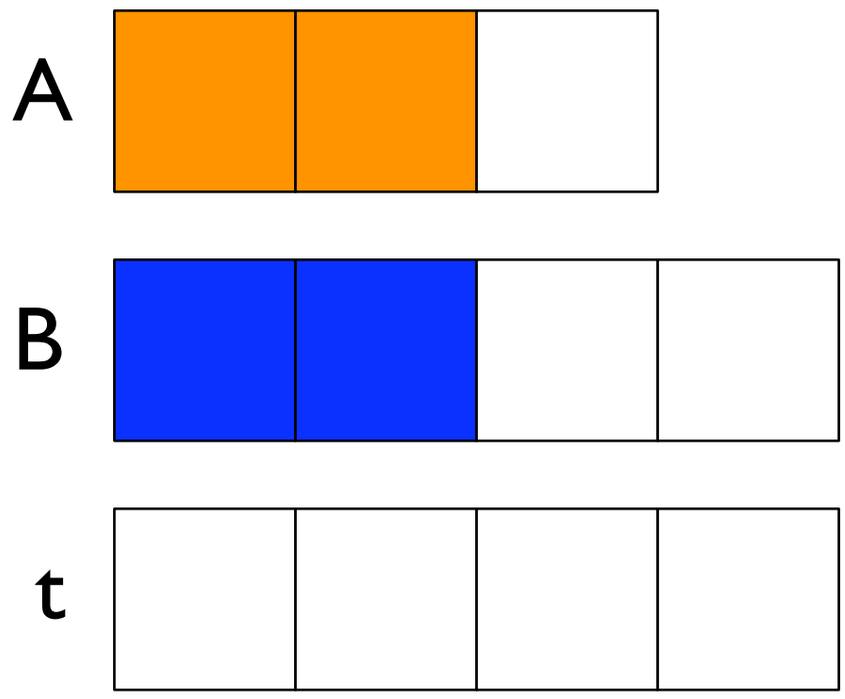
Example: Timetable



$start(A) \in \{0,1\}$ $dur(A) = 2$

$start(B) \in \{1,2,3\}$ $dur(B) = 2$

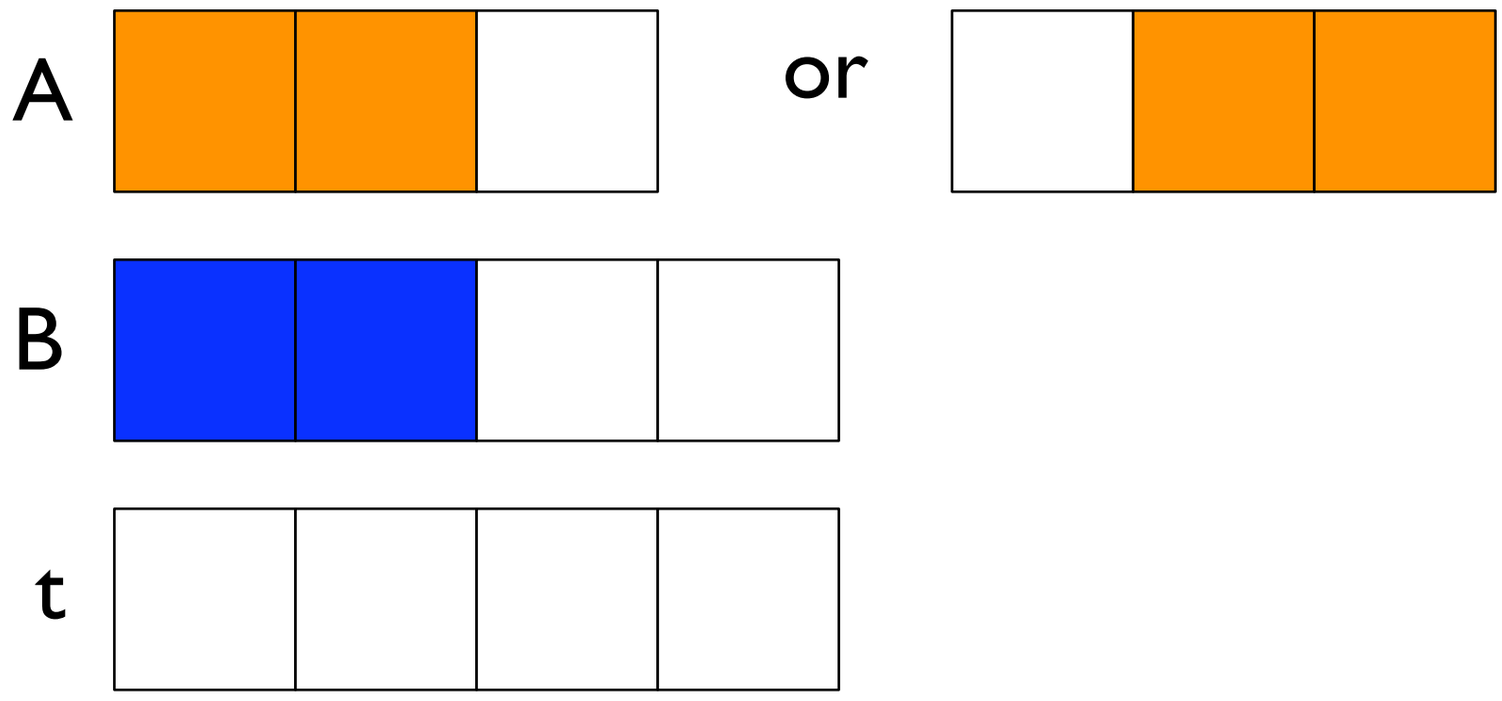
Example: Timetable



$start(A) \in \{0,1\}$ $dur(A) = 2$

$start(B) \in \{1,2,3\}$ $dur(B) = 2$

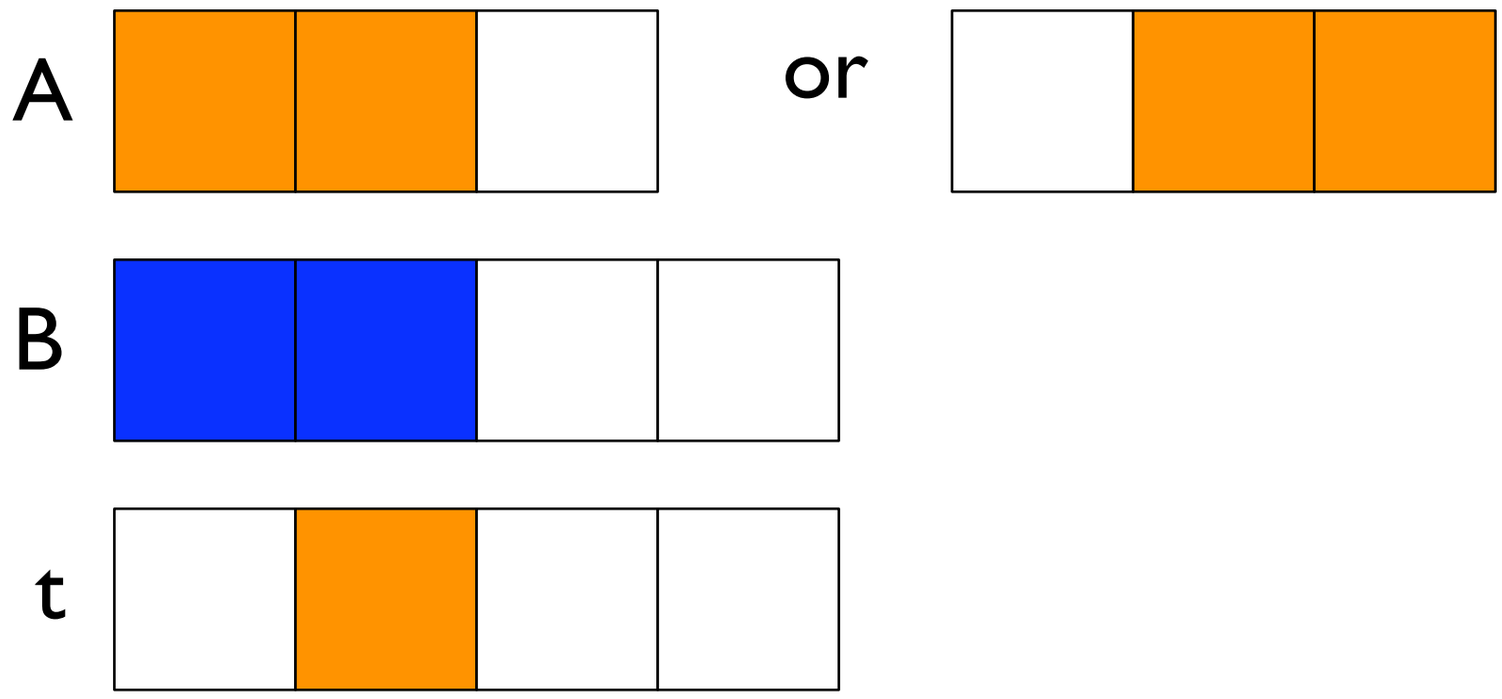
Example: Timetable



$start(A) \in \{0,1\}$ $dur(A) = 2$

$start(B) \in \{1,2,3\}$ $dur(B) = 2$

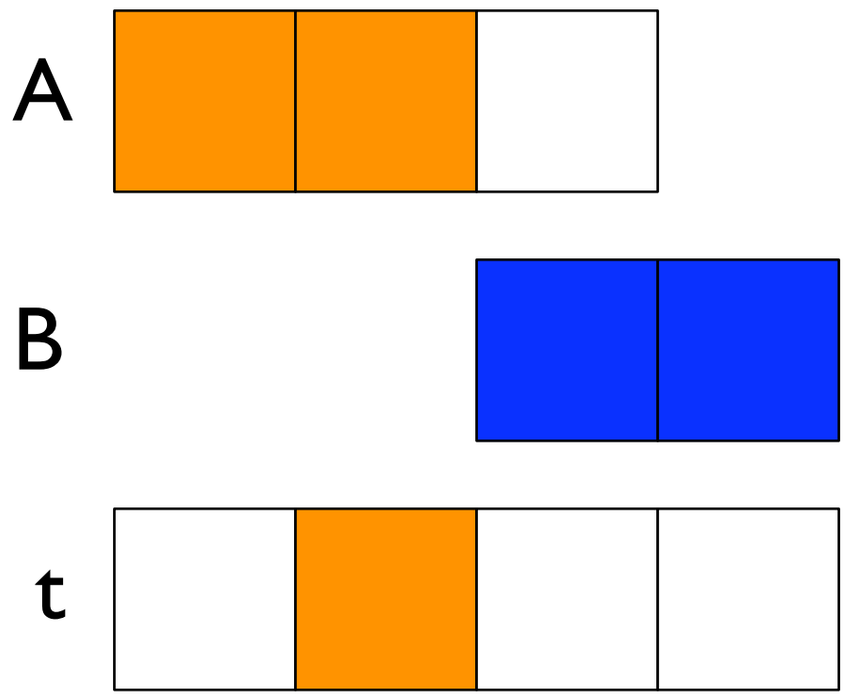
Example: Timetable



$start(A) \in \{0,1\}$ $dur(A) = 2$

$start(B) \in \{1,2,3\}$ $dur(B) = 2$

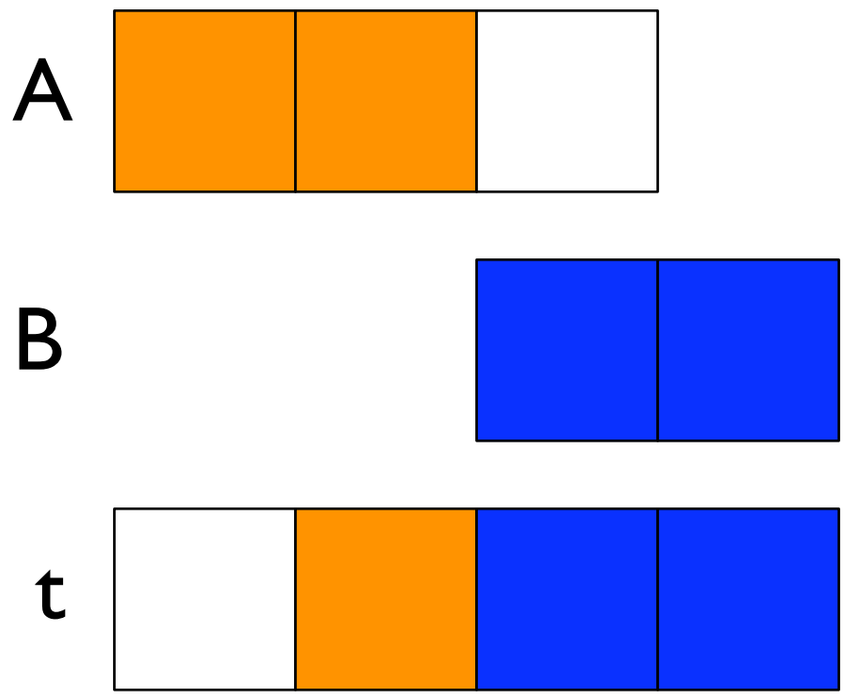
Example: Timetable



$start(A) \in \{0, 1\}$ $dur(A) = 2$

$start(B) \in \{2\}$ $dur(B) = 2$

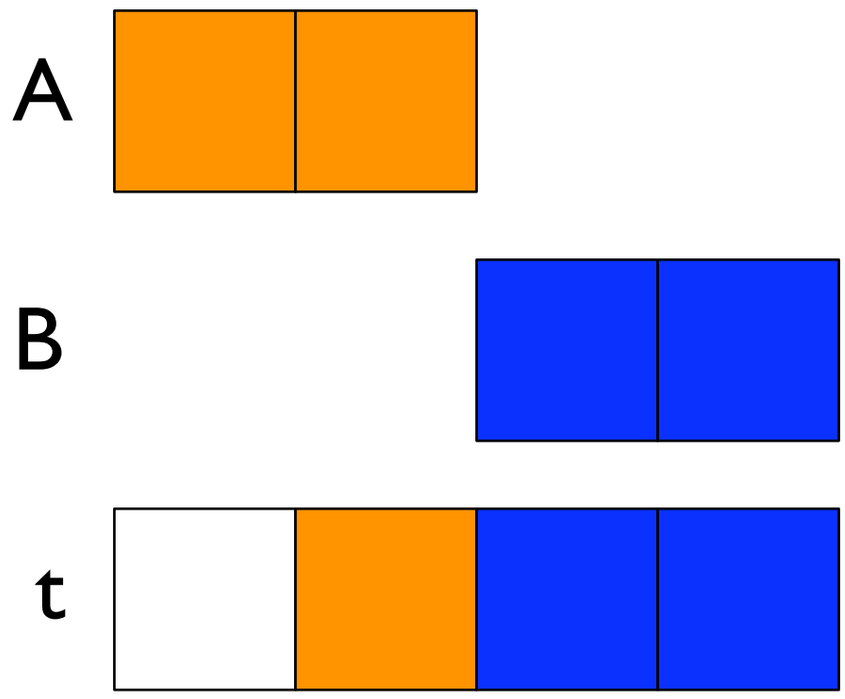
Example: Timetable



$start(A) \in \{0, 1\}$ $dur(A) = 2$

$start(B) \in \{2\}$ $dur(B) = 2$

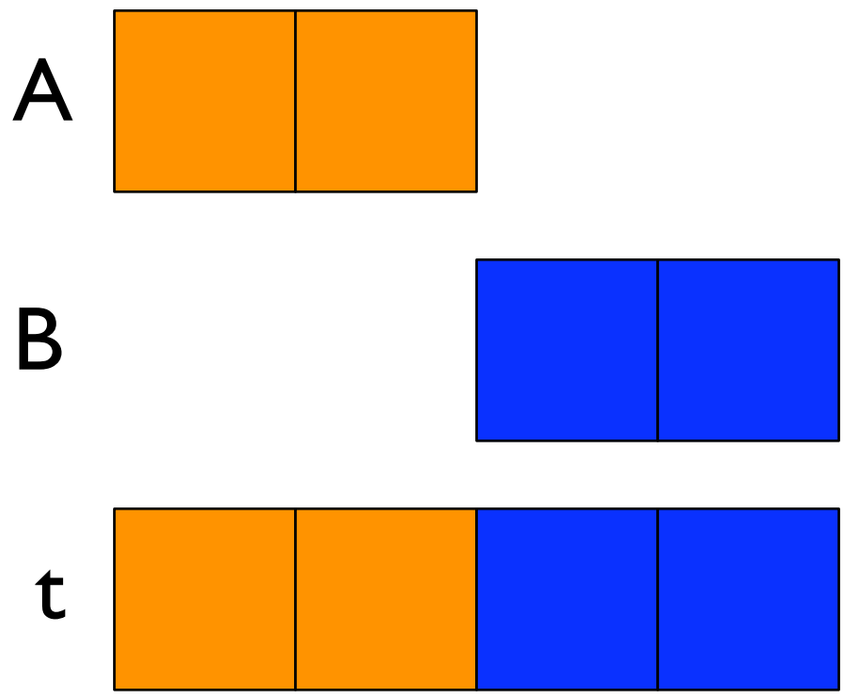
Example: Timetable



$start(A) \in \{0\}$ $dur(A) = 2$

$start(B) \in \{2\}$ $dur(B) = 2$

Example: Timetable



$start(A) \in \{0, 1\}$ $dur(A) = 2$

$start(B) \in \{2\}$ $dur(B) = 2$

Timetable propagation

- Timetable: data structure that records per resource where some task definitively uses the resource
- Propagate
  - from tasks to timetable
  - from timetable to tasks

Edge Finding

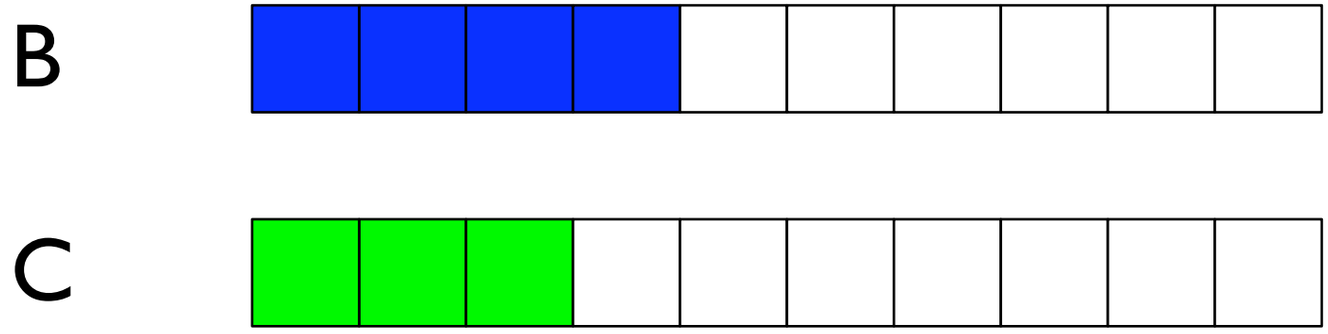
- Time tabling is often weaker than reification
- But important idea:

  record when resource is used
- Edge finding is a more general scheme to propagate order between tasks

Edge finding: Idea

- Assume a subset *O* of tasks and $T \in O$
  - constrain *T* to execute first
    - find out whether tasks in $O-\{T\}$ must, can, or cannot execute first or last
  - symmetrically for *T* last
- Can be done in $O(n^2)$ for *n* tasks

Example: Edge finding

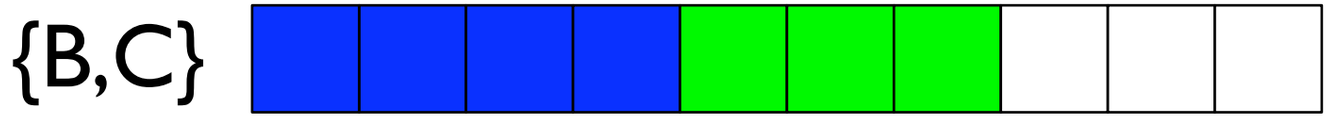


$start(A) \in \{0,...,11\}$ $dur(A) = 6$

$start(B) \in \{1,...,7\}$ $dur(B) = 4$

$start(C) \in \{1,...,8\}$ $dur(C) = 3$

Example: Edge finding



consider {B,C}
A cannot be before {B,C}

start(A) ∈ {*0,...,11*} *dur(A)* = *6*

start(B) ∈ {*1,...,7*} *dur(B)* = *4*

start(C) ∈ {*1,...,8*} *dur(C)* = *3*

Example: Edge finding

A

{B,C}

consider {B,C}
A cannot be before {B,C}

$start(A) \in \{8,...,11\}$ $dur(A) = 6$

$start(B) \in \{1,...,7\}$ $dur(B) = 4$

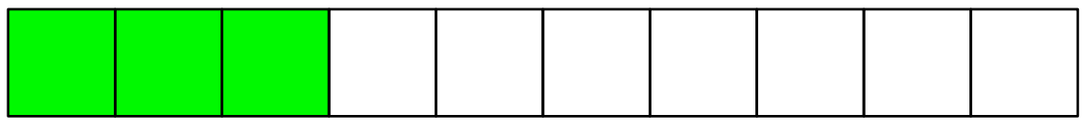
$start(C) \in \{1,...,8\}$ $dur(C) = 3$

Example: Edge finding

A

B

C



timetable and reification do not propagate anything!

$start(A) \in \{8,...,11\}$ $dur(A) = 6$

$start(B) \in \{1,...,7\}$ $dur(B) = 4$

$start(C) \in \{1,...,8\}$ $dur(C) = 3$

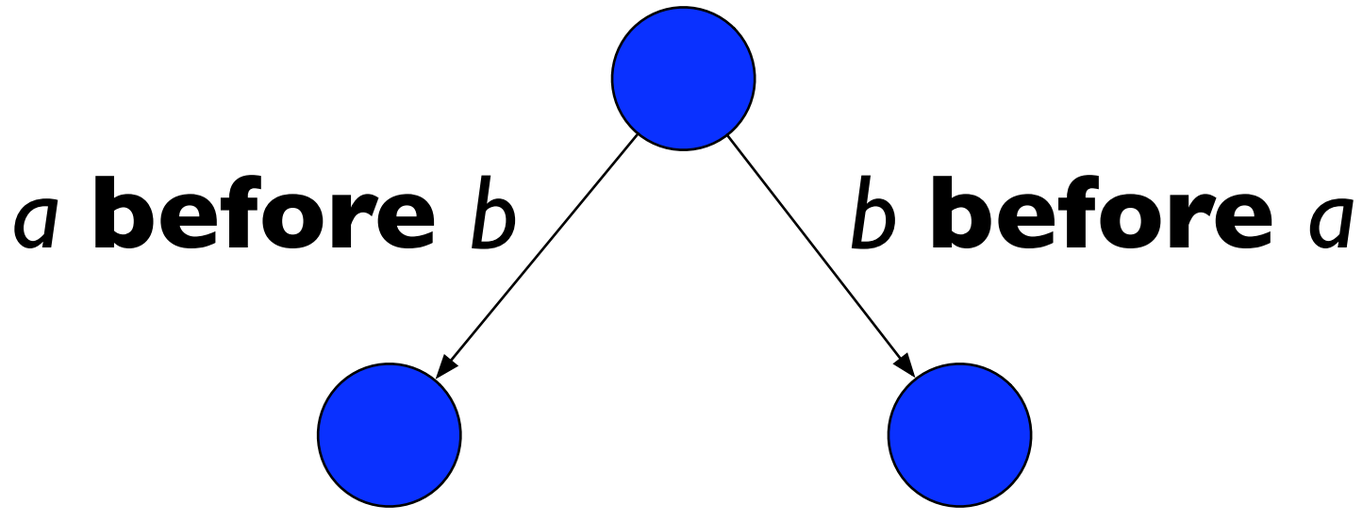
Branching

- General idea of any heuristic:
  determine *critical* variables early!
- A critical variable is one that makes a big difference if determined

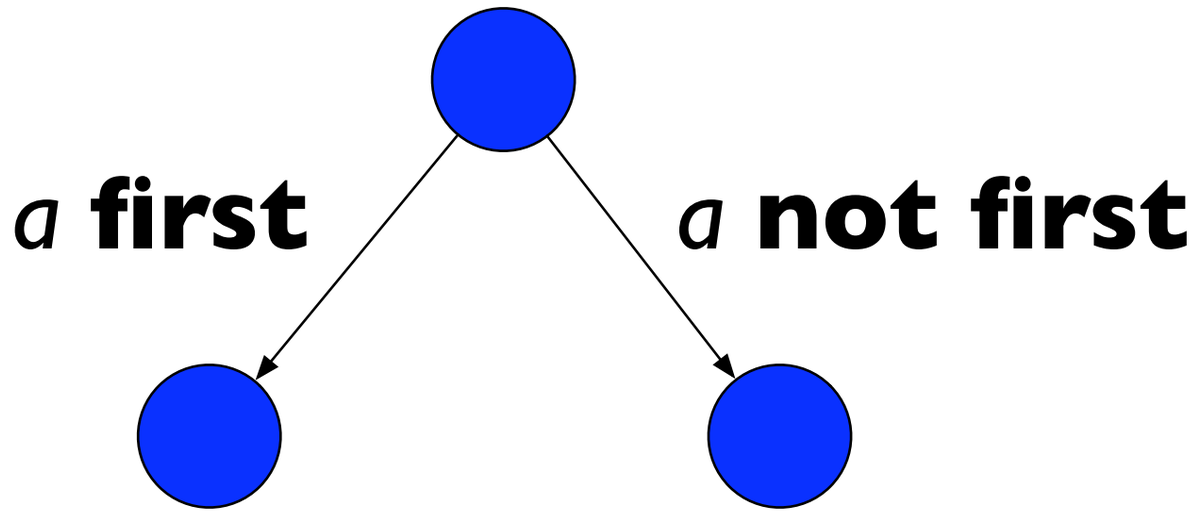
Branching

- Heuristic: establish order among tasks
  - which resource to choose
    - guess first task on resource
- After ordering: assign start times
  - solution exists => no branching required
  - after assigning, propagate precedence constraints

Branching

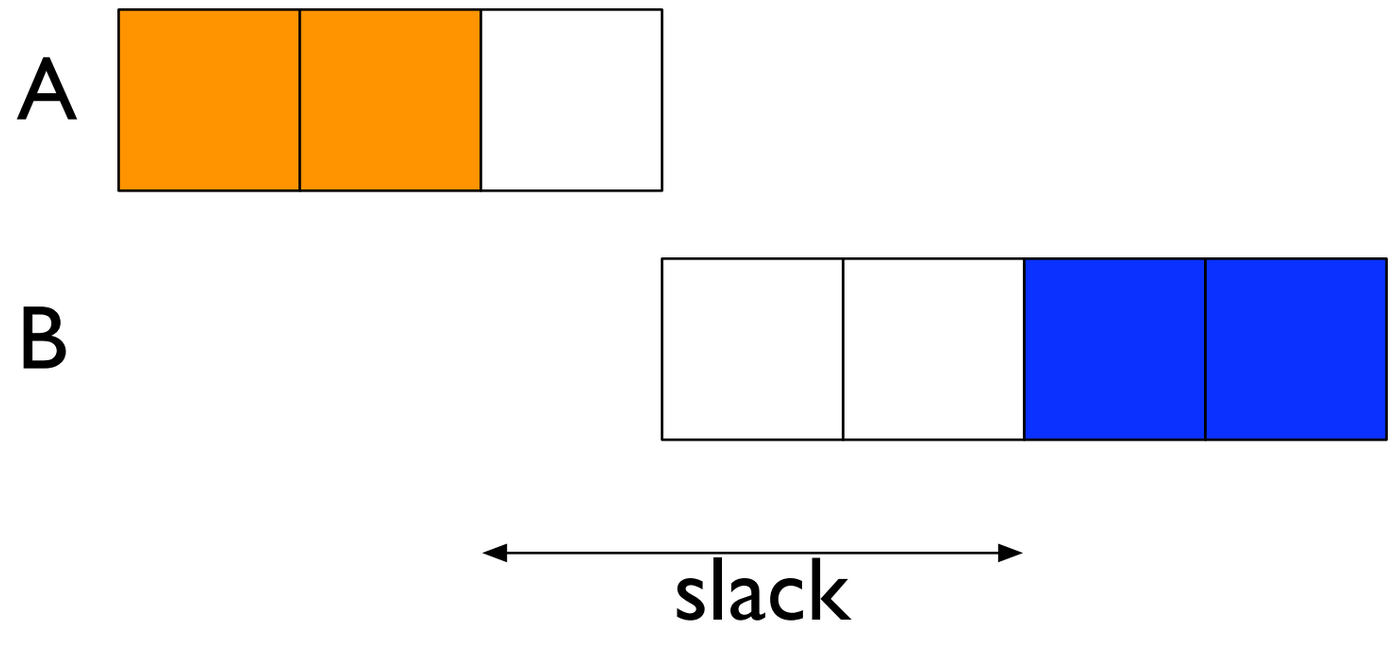


Branching



Branching: Slack

- How to choose *q* ?
- Good heuristic: minimum *slack*

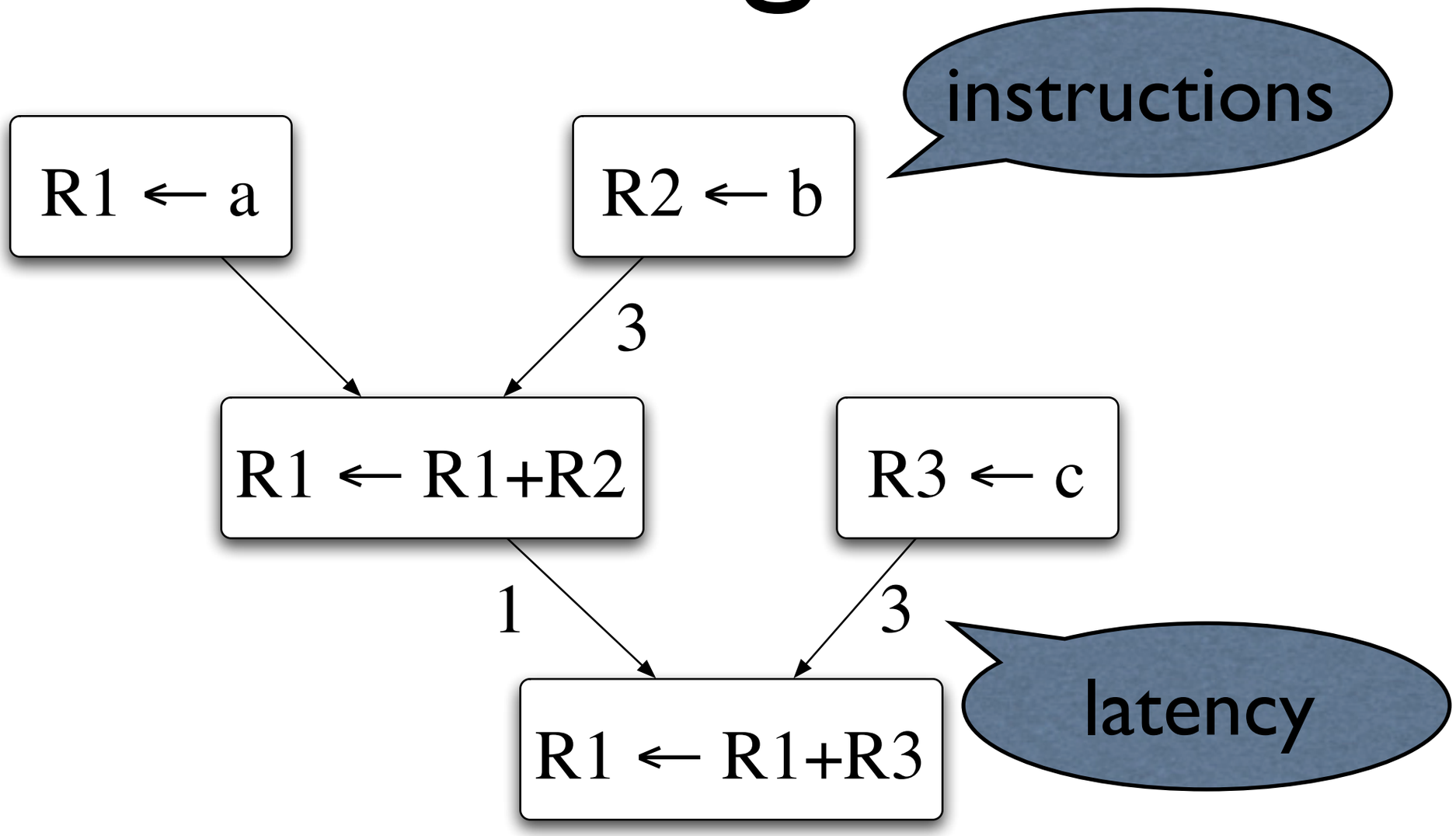


Example: Instruction Scheduling

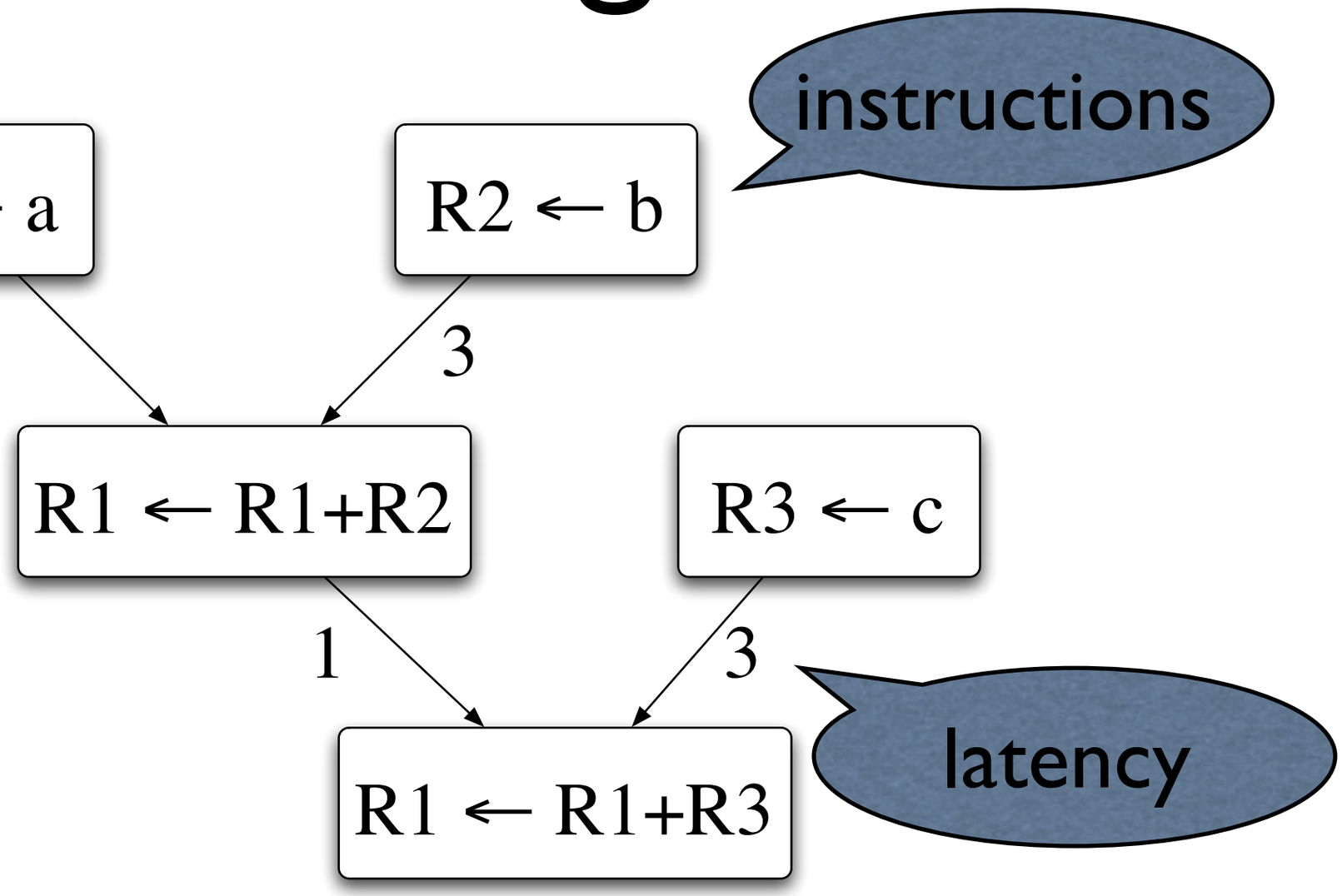
- Optimize machine code
- Goal: minimum length instruction schedule

Best paper CP 2001 (Peter van Beek and Kent Wilken)

Example: Instruction Scheduling



Example: Instruction Scheduling



Find *issue time s(i)* such that

1. $i \neq j \Rightarrow s(i) \neq s(j)$

2. $s(i) + l(i,j) \leq s(j)$

Minimize *max s(i)*

Example: Instruction Scheduling

- Constraints:
  all *s(i)* must be distinct
- Precedence constraints for latency
- Plus special case of edge finding

Results

- Built into gcc
- SPEC95 FP
- Large basic blocks (up to 1000 instructions)
- Optimally solved
- Far better than ILP approach

Cumulative Scheduling

- Each resource *R* has a capacity *cap(R)*
- Each task T on R uses amount *use(R)*
- Tasks can overlap but never exceed capacity

Cumulative: Disjunctive Propagation

- For tasks *A* and *B* on resource *R*:

$$use(A) + use(B) \leq cap(R)$$

or $start(A) + dur(A) \leq start(B)$

or $start(B) + dur(B) \leq start(A)$

Cumulative

- Techniques from disjunctive scheduling carry over to cumulative scheduling
- Generalized timetabling, edge-finding, not-first/not-last

Geometric Interpretation

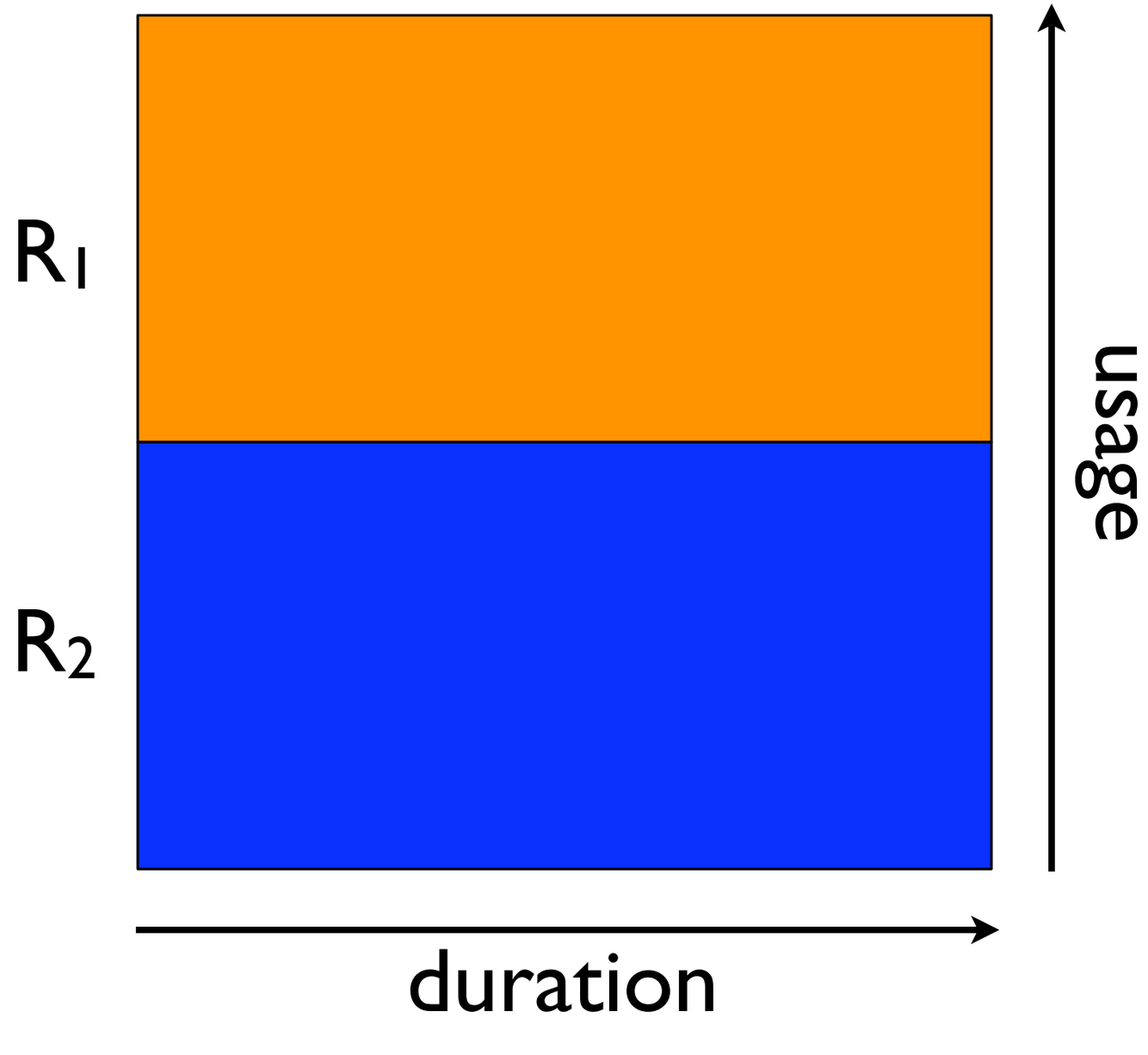
- Task is rectangle

  dimension *dur(T)* × *use(T)*

  places at x-coordinate *start(T)*

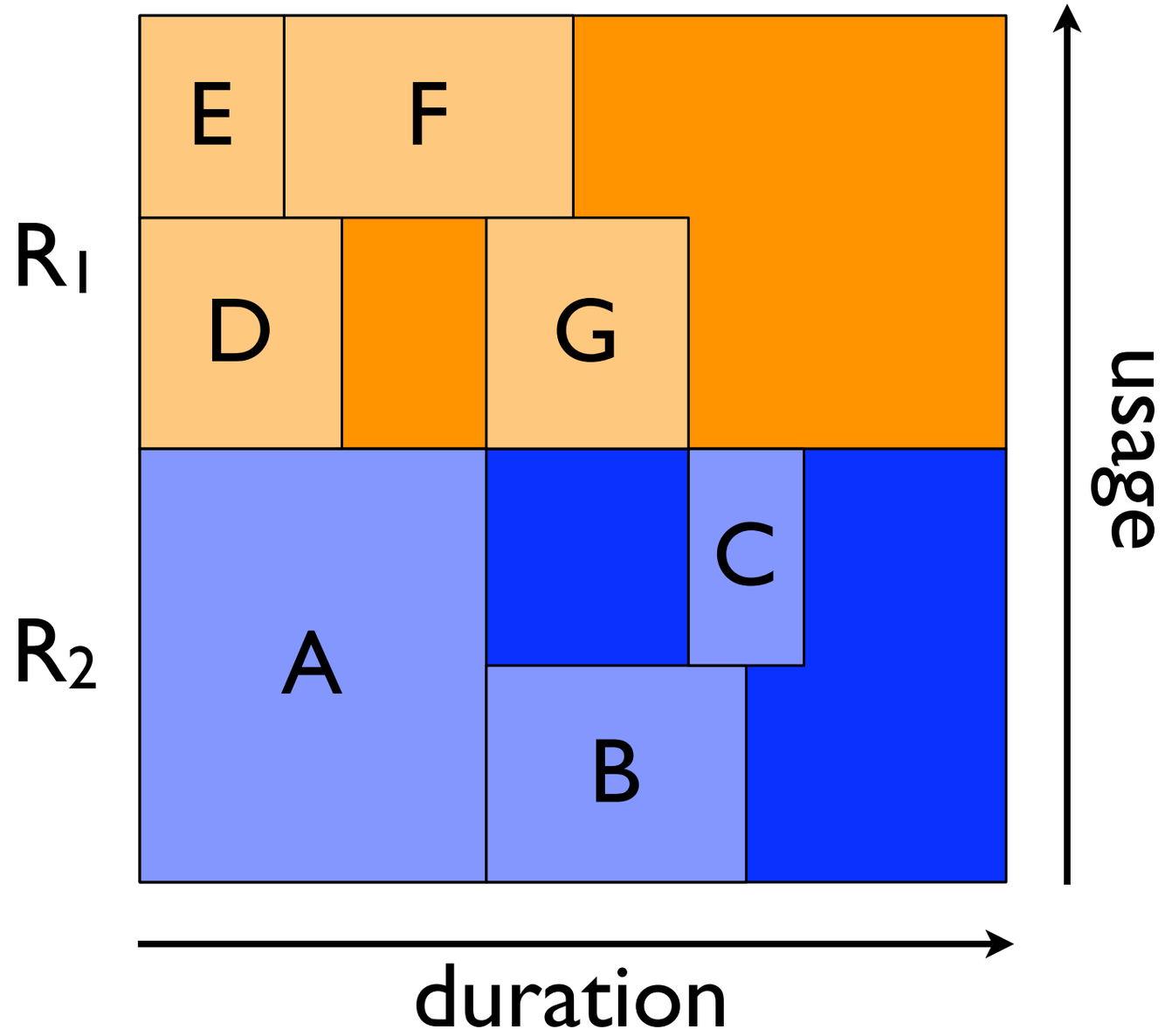
- Resources are rectangles
  - enclose task-rectangles
  - rectangles never exceed *y*-cordinates

Geometric Interpretation



Geometric Interpretation

We've seen something like this before...



Summary

- Scheduling: hard, real life problems
- Model similar to temporal relations, but:

  interested in actual solution + optimization
- Global constraints: timetable, edge-finding, not-first/not-last