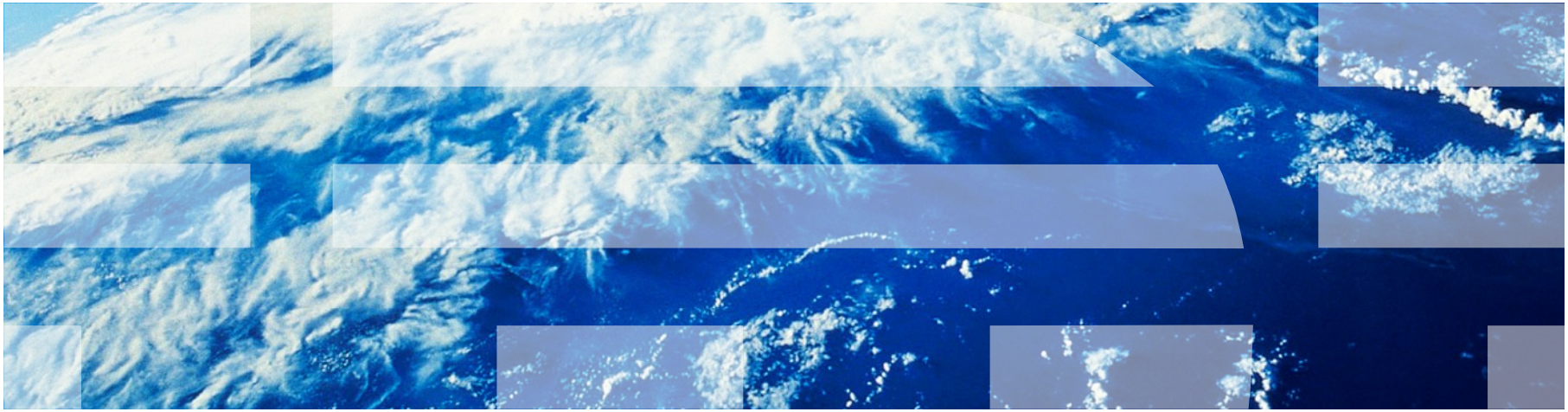
CP-based scheduling in examples

CP Optimizer Development Team

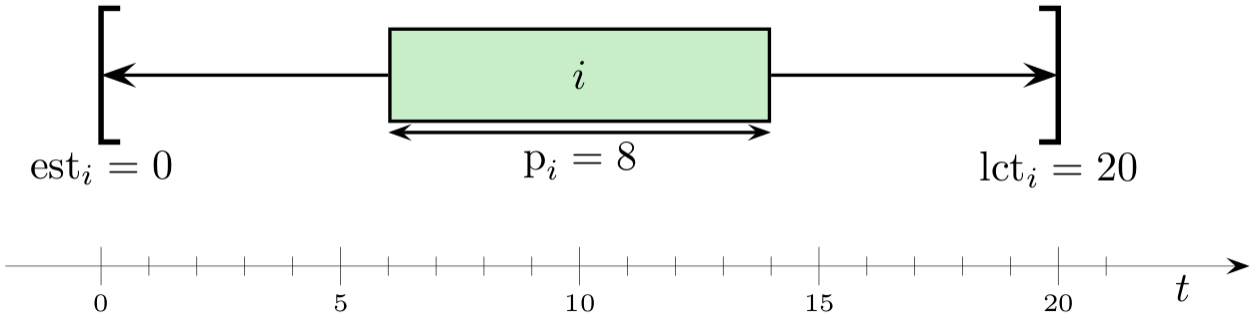
Petr Vilím



Scheduling, Interval Variable

[1] Laborie. *IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems*. CPAIOR 2009.

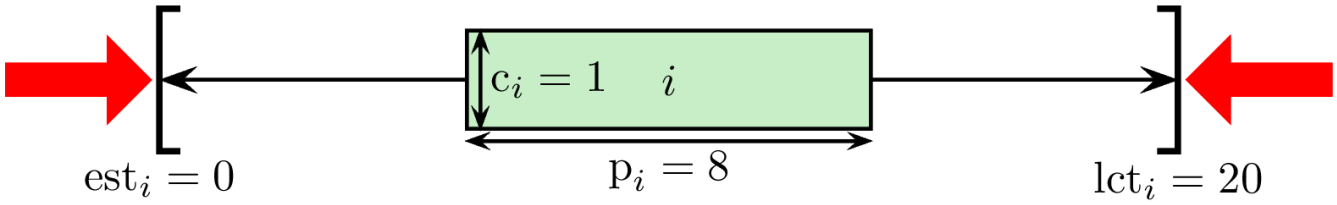
(Non-optional) Interval Variable



- **Interval variable** models an operation, task, activity, ..
  - Domain: Domain(i) $\subseteq$ { [s,e) | s,e$\in \mathbb{Z}$, s≤e }
  - Interval variables are not preemptive (cannot be interrupted).
  - All times are integers.
- Usually interval variables are characterized by:
  - Earliest (**minimum**) **start** time: $\text{est}_i \leq s$
  - Latest (**maximum**) completion (**end**) time (deadline) $\text{lct}_i \geq e$
  - Processing time (duration, size, **length**) $p_i = e - s$
- The task is to assign times to interval variables that satisfy all the constraints.

(Non-optional) Interval Variable: Propagation

- Internally, domain of an interval variable is represented by:
  - Range for start time: $est_i..lst_i$,
  - Range for end time: $ect_i..lct_i$,
  - Range for duration: $pmin_i..pmax_i$.
  - CP Optimizer does not maintain holes in those ranges.
- Interval $[est_i, lct_i]$ is called *time window* of activity i.
- Propagation aims to make the time window smaller by increasing est_i and decreasing lct_i:

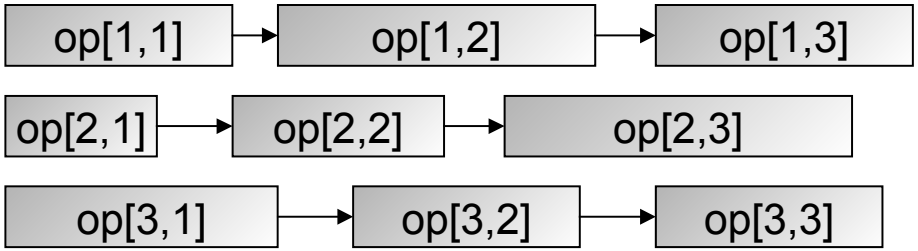


Example #1:

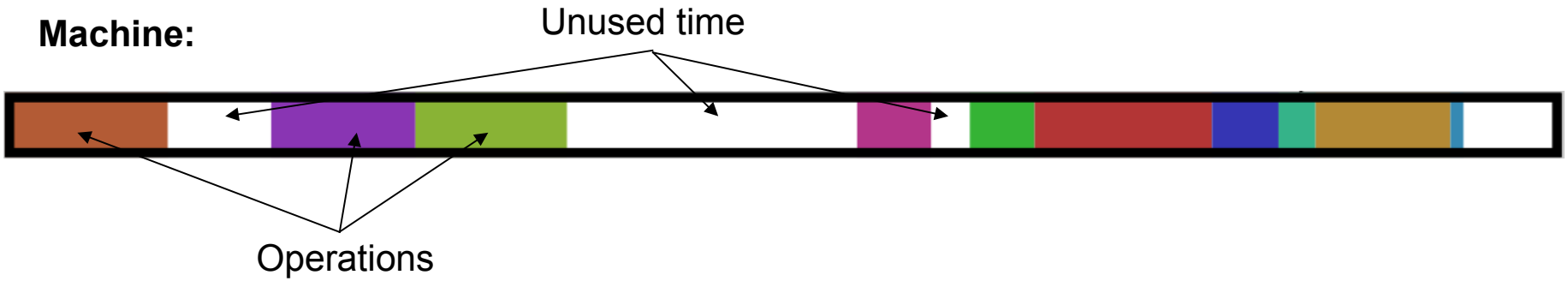
Job-shop

Example #1: Job-shop

- Schedule n jobs, each job consists of m consecutive operations (their order is fixed):

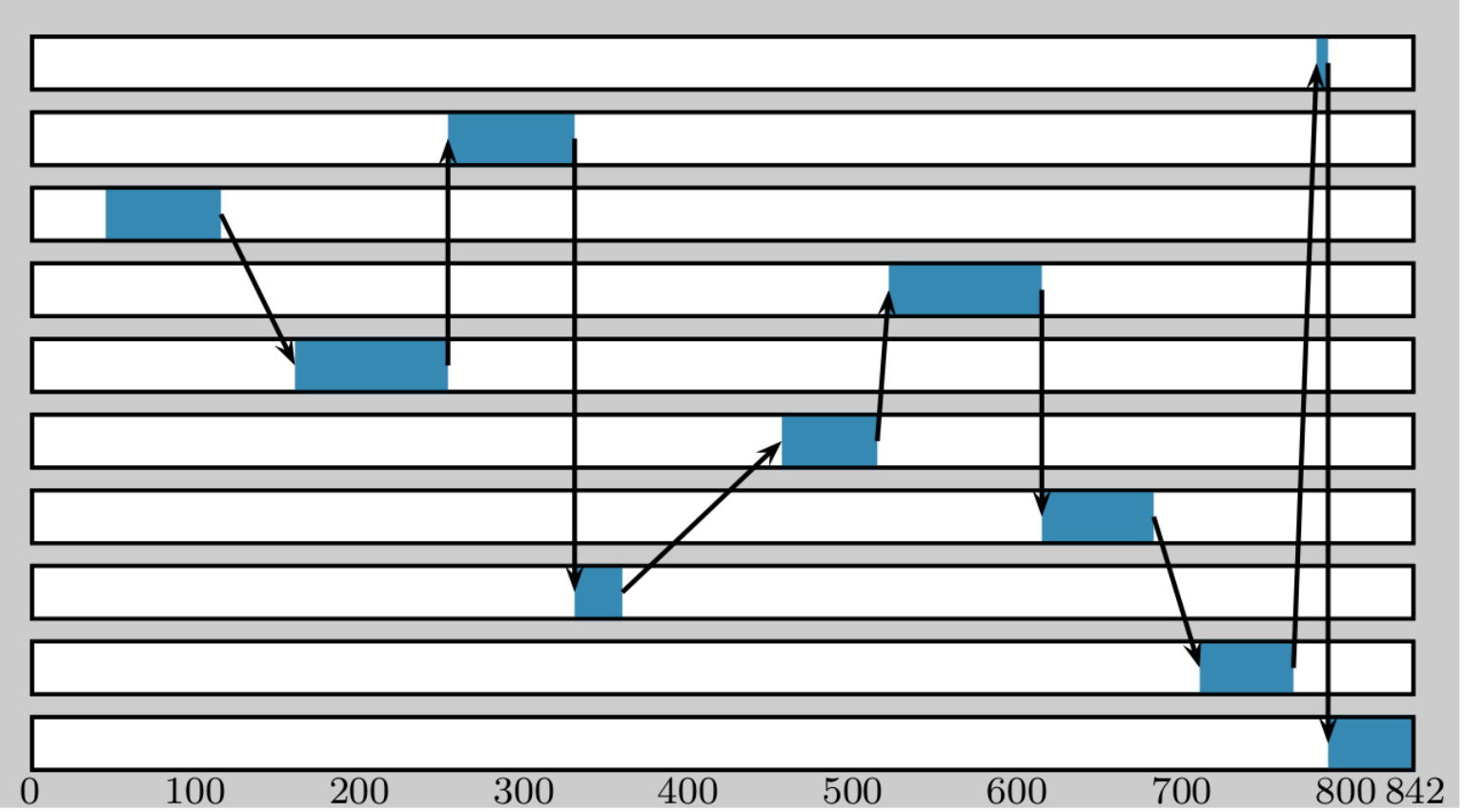


- Each operation requires exclusive use of one of m machines.
  - Each machine can handle only one operation at a time.
  - i.e. two operations on the same machine cannot overlap.



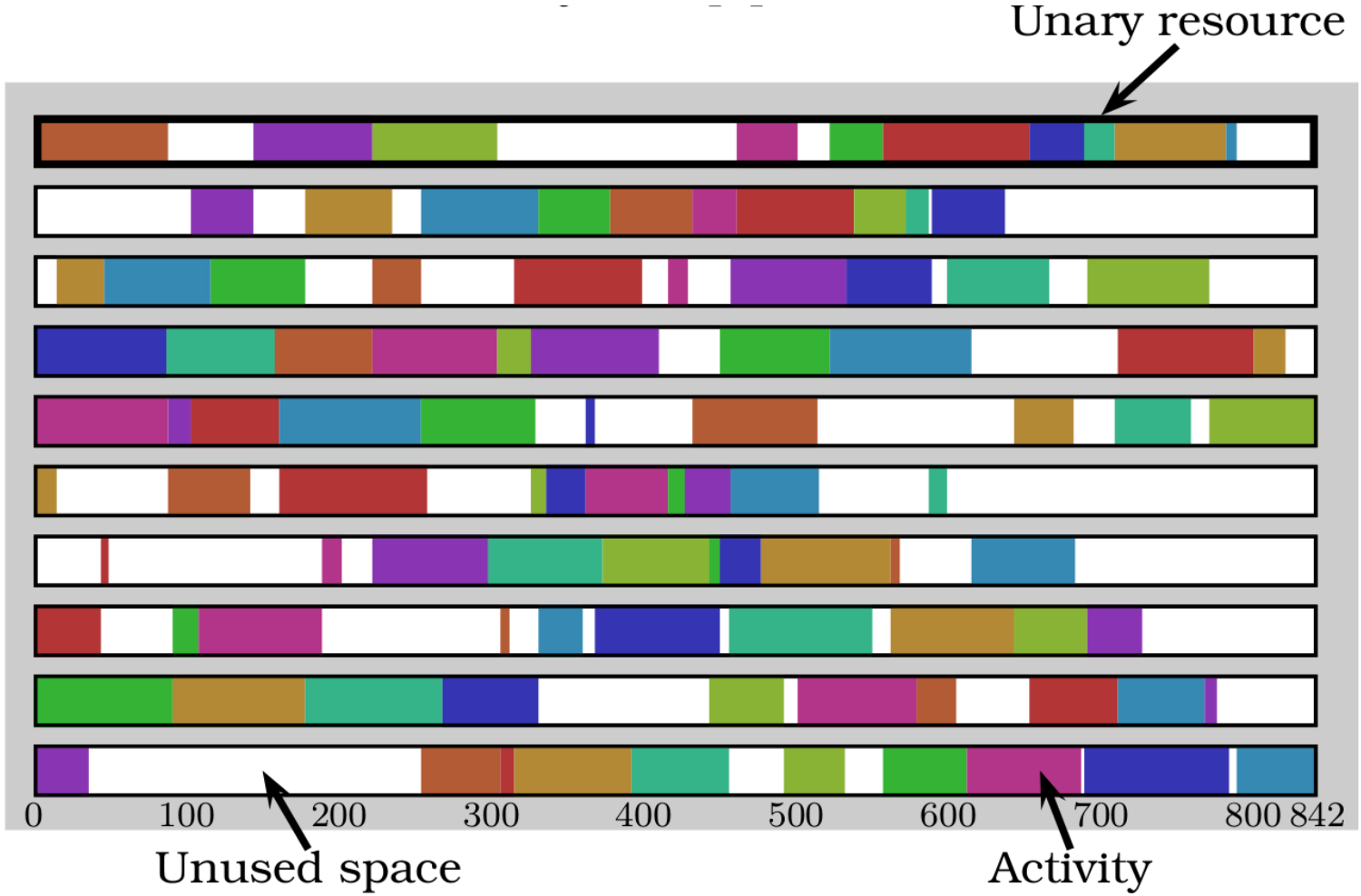
Example #1: Job-shop

- Each operation in a job require different machine.
- Order of required machines is different for each job.



Example #1: Job-shop

- Objective: find shortest possible schedule.
- Optimal solution of problem instance la19:



Example #1: Job-shop

```
1  using CP;
2  int n = ...; // Number of jobs
3  int m = ...; // Number of machines
4  range Jobs = 0..n-1;
5  range Mchs = 0..m-1;
6
7  tuple Operation {
8    int mch; // Machine
9    int pt;  // Processing time
10 };
11 Operation op[j in Jobs][i in Mchs] = ...;
12
13 dvar interval itvs[j in Jobs][o in Mchs] size op[j][o].pt;
14
15 minimize max(j in Jobs) endOf(itvs[j][m-1]);
16
17 subject to {
18   forall (j in Jobs, o in 0..m-2)
19     endBeforeStart(itvs[j][o], itvs[j][o+1]);
20   forall (i in Mchs)
21     noOverlap(all(j in Jobs, o in Mchs : op[j][o].mch == i) itvs[j][o]);
22 }
```

- Model is using CP Optimizer engine (not CPLEX)

Example #1: Job-shop

```
1  using CP;
2  int n = ...; // Number of jobs
3  int m = ...; // Number of machines
4  range Jobs = 0..n-1;
5  range Mchs = 0..m-1;
6
7  tuple Operation {
8    int mch; // Machine
9    int pt;  // Processing time
10 };
11 Operation op[j in Jobs][i in Mchs] = ...;
12
13 dvar interval itvs[j in Jobs][o in Mchs] size op[j][o].pt;
14
15 minimize max(j in Jobs) endOf(itvs[j][m-1]);
16
17 subject to {
18   forall (j in Jobs, o in 0..m-2)
19     endBeforeStart(itvs[j][o], itvs[j][o+1]);
20   forall (i in Mchs)
21     noOverlap(all(j in Jobs, o in Mchs : op[j][o].mch == i) itvs[j][o]);
22 }
```

Data

- Data reading:
  - Problem size (n, m)
  - Operations in jobs, each with machine (mch) and duration (pt)

Example #1: Job-shop

```
1  using CP;
2  int n = ...; // Number of jobs
3  int m = ...; // Number of machines
4  range Jobs = 0..n-1;
5  range Mchs = 0..m-1;
6
7  tuple Operation {
8    int mch; // Machine
9    int pt;  // Processing time
10 };
11 Operation op[j in Jobs][i in Mchs] = ...;
12
13 dvar interval itvs[j in Jobs][o in Mchs] size op[j][o].pt;
14
15 minimize max(j in Jobs) endOf(itvs[j][m-1]);
16
17 subject to {
18   forall (j in Jobs, o in 0..m-2)
19     endBeforeStart(itvs[j][o], itvs[j][o+1]);
20   forall (i in Mchs)
21     noOverlap(all(j in Jobs, o in Mchs : op[j][o].mch == i) itvs[j][o]);
22 }
```

Decisions

- Decision variables:
  - 2D array of interval variables (one for each operation.

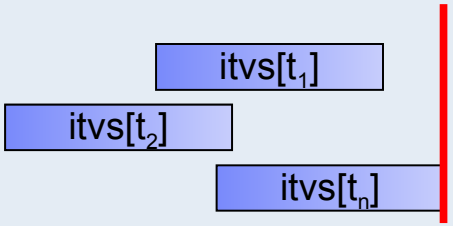
Example #1: Job-shop

```
1  using CP;
2  int n = ...; // Number of jobs
3  int m = ...; // Number of machines
4  range Jobs = 0..n-1;
5  range Mchs = 0..m-1;
6
7  tuple Operation {
8    int mch; // Machine
9    int pt;  // Processing time
10 };
11 Operation op[j in Jobs][i in Mchs] = ...;
12
13 dvar interval itvs[j in Jobs][o in Mchs] size op[j][o].pt;
14
15 minimize max(j in Jobs) endOf(itvs[j][m-1]);
16
17 subject to {
18   forall (j in Jobs, o in 0..m-2)
19     endBeforeStart(itvs[j][o], itvs[j][o+1]);
20   forall (i in Mchs)
21     noOverlap(all(j in Jobs, o in Mchs : op[j][o].mch == i) itvs[j][o]);
22 }
```

Objective

- Minimize objective function:
  - Makespan: total duration.

max(j in Jobs) endOf(itvs[j][m-1])

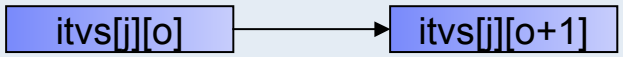


Example #1: Job-shop

```
1  using CP;
2  int n = ...; // Number of jobs
3  int m = ...; // Number of machines
4  range Jobs = 0..n-1;
5  range Mchs = 0..m-1;
6
7  tuple Operation {
8    int mch; // Machine
9    int pt;  // Processing time
10 };
11 Operation op[j in Jobs][i in Mchs] = ...;
12
13 dvar interval itvs[j in Jobs][o in Mchs] size op[j][o].pt;
14
15 minimize max(j in Jobs) endOf(itvs[j][m-1]);
16
17 subject to {
18   forall (j in Jobs, o in 0..m-2)
19     endBeforeStart(itvs[j][o], itvs[j][o+1]);
20   forall (i in Mchs)
21     noOverlap(all(j in Jobs, o in Mchs : op[j][o].mch == i) itvs[j][o]);
22 }
```

Precedences

- Precedence constraints for intervals in a job



Example #1: Job-shop

```
1  using CP;
2  int n = ...; // Number of jobs
3  int m = ...; // Number of machines
4  range Jobs = 0..n-1;
5  range Mchs = 0..m-1;
6
7  tuple Operation {
8    int mch; // Machine
9    int pt;  // Processing time
10 };
11 Operation op[j in Jobs][i in Mchs] = ...;
12
13 dvar interval itvs[j in Jobs][o in Mchs] size op[j][o].pt;
14
15 minimize max(j in Jobs) endOf(itvs[j][m-1]);
16
17 subject to {
18   forall (j in Jobs, o in 0..m-2)
19     endBeforeStart(itvs[j][o], itvs[j][o+1]);
20   forall (i in Mchs)
21     noOverlap(all(j in Jobs, o in Mchs : op[j][o].mch == i) itvs[j][o]);
22 }
```

noOverlap

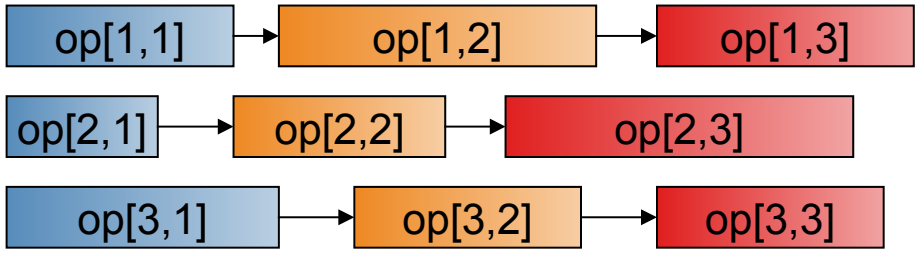
- noOverlap constraint for every machine:
  - Two intervals on the same machine cannot overlap

Example #2:

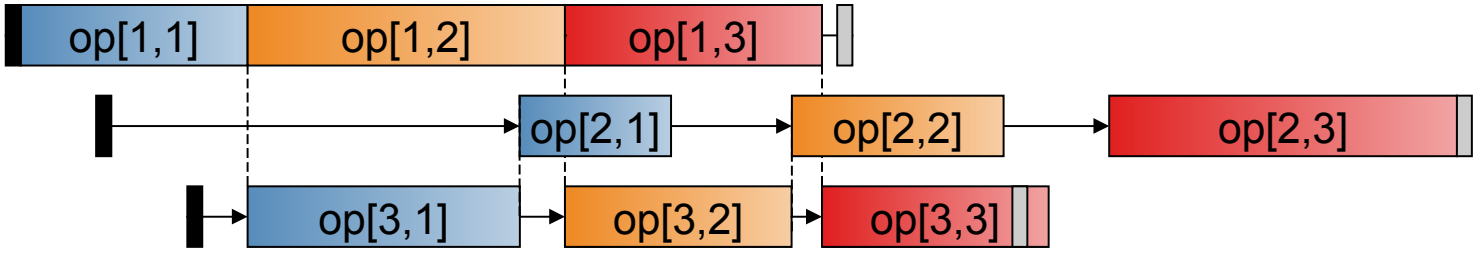
Flow-shop with Earliness/Tardiness

Example #2: Flow-shop with Earliness/Tardiness

- Classical Flow-Shop Scheduling problem:
  - n jobs, m machines



- Job release dates ( ), due dates ( ) and weight



Example #2: Flow-shop with Earliness/Tardiness

```
1  using CP;
2  int n = ...;
3  int m = ...;
4  int rd[1..n] = ...;
5  int dd[1..n] = ...;
6  float w[1..n] = ...;
7  int pt[1..n][1..m] = ...;
8  float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9  dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endOf(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13   forall(i in 1..n) {
14     rd[i] <= startOf(op[i][1]);
15     forall(j in 1..m-1)
16       endBeforeStart(op[i][j],op[i][j+1]);
17   }
18   forall(j in 1..m)
19     noOverlap(all(i in 1..n) op[i][j]);
20 }
```

- The model is using CP Optimizer engine

Example #2: Flow-shop with Earliness/Tardiness

```
1  using CP;
2  int n = ...;
3  int m = ...;
4  int rd[1..n] = ...;
5  int dd[1..n] = ...;
6  float w[1..n] = ...;
7  int pt[1..n][1..m] = ...;
8  float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9  dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endOf(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13   forall(i in 1..n) {
14     rd[i] <= startOf(op[i][1]);
15     forall(j in 1..m-1)
16       endBeforeStart(op[i][j],op[i][j+1]);
17   }
18   forall(j in 1..m)
19     noOverlap(all(i in 1..n) op[i][j]);
20 }
```

Data

- Data reading and computation:
  - Problem size (n, m)
  - Job release date (rd), due-date (dd), weight (w)
  - Operation processing time (pt)
  - Normalization factor (W)

Example #2: Flow-shop with Earliness/Tardiness

```
1  using CP;
2  int n = ...;
3  int m = ...;
4  int rd[1..n] = ...;
5  int dd[1..n] = ...;
6  float w[1..n] = ...;
7  int pt[1..n][1..m] = ...;
8  float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9  dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endOf(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13   forall(i in 1..n) {
14     rd[i] <= startOf(op[i][1]);
15     forall(j in 1..m-1)
16       endBeforeStart(op[i][j],op[i][j+1]);
17   }
18   forall(j in 1..m)
19     noOverlap(all(i in 1..n) op[i][j]);
20 }
```

Decisions

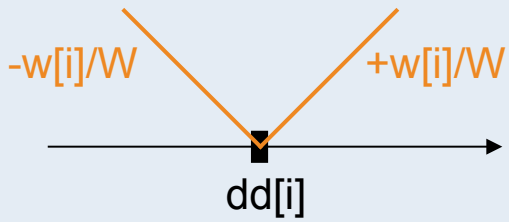
- Decision variables and expressions:
  - Operations: 2D array of **interval variables**
  - Jobs end time: 1D array of **integer expressions**

Example #2: Flow-shop with Earliness/Tardiness

```
1  using CP;
2  int n = ...;
3  int m = ...;
4  int rd[1..n] = ...;
5  int dd[1..n] = ...;
6  float w[1..n] = ...;
7  int pt[1..n][1..m] = ...;
8  float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9  dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endOf(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13   forall(i in 1..n) {
14     rd[i] <= startOf(op[i][1]);
15     forall(j in 1..m-1)
16       endBeforeStart(op[i][j],op[i][j+1]);
17   }
18   forall(j in 1..m)
19     noOverlap(all(i in 1..n) op[i][j]);
20 }
```

Objective

- Objective:
  - Weighted sum of earliness/tardiness costs



Problem #2: Flow-shop with Earliness/Tardiness

```
1  using CP;
2  int n = ...;
3  int m = ...;
4  int rd[1..n] = ...;
5  int dd[1..n] = ...;
6  float w[1..n] = ...;
7  int pt[1..n][1..m] = ...;
8  float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9  dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endOf(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13   forall(i in 1..n) {
14     rd[i] <= startOf(op[i][1]);
15     forall(j in 1..m-1)
16       endBeforeStart(op[i][j],op[i][j+1]);
17   }
18   forall(j in 1..m)
19     noOverlap(all(i in 1..n) op[i][j]);
20 }
```

Constraints

- Constraints:
  - Release dates of job i (using **startOf** integer expression)
  - **Precedence constraints** between operations of job i
  - **No-overlap** of operations on the same machine j

Example #3:

Resource-Constrained Scheduling Problem (Basic RCPSP)

Example #3: Basic RCPSP

```
using CP;
int NbTasks = ...;
int NbRsrcs = ...;
range RsrcIds = 1..NbRsrcs;
int Capacity[RsrcIds] = ...;
tuple Task { key int id; int pt; int dmds[RsrcIds]; {int} succs; }
{Task} Tasks = ...;
dvar interval itvs[t in Tasks] size t.pt;
minimize max(t in Tasks) endOf(itvs[t]);
subject to {
  forall (r in RsrcIds)
    sum (t in Tasks: t.dmds[r]>0) pulse(itvs[t], t.dmds[r]) <= Capacity[r];
  forall (t1 in Tasks, t2id in t1.succs)
    endBeforeStart(itvs[t1],itvs[<t2id>]);
}
```

- The model is using CP Optimizer engine

Example #3: Basic RCPSP

```
1  using CP;
2  int NbTasks = ...;
3  int NbRsrcs = ...;
4  range RsrcIds = 1..NbRsrcs;
5  int Capacity[RsrcIds] = ...;
6  tuple Task { key int id; int pt; int dmds[RsrcIds]; {int} succs; }
7  {Task} Tasks = ...;
8  dvar interval itvs[t in Tasks] size t.pt;
9  minimize max(t in Tasks) endOf(itvs[t]);
10 subject to {
11   forall (r in RsrcIds)
12     sum (t in Tasks: t.dmds[r]>0) pulse(itvs[t], t.dmds[r]) <= Capacity[r];
13   forall (t1 in Tasks, t2id in t1.succs)
14     endBeforeStart(itvs[t1],itvs[<t2id>]);
15 }
```

Data

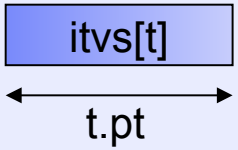
- Data reading:
  - Problem size (number of tasks and resources)
  - Resource capacities
  - Tasks with their processing time, resource demand and successors

Example #3: Basic RCPSP

```
1  using CP;
2  int NbTasks = ...;
3  int NbRsrcs = ...;
4  range RsrcIds = 1..NbRsrcs;
5  int Capacity[RsrcIds] = ...;
6  tuple Task { key int id; int pt; int dmds[RsrcIds]; {int} succs; }
7  {Task} Tasks = ...;
8  dvar interval itvs[t in Tasks] size t.pt;
9  minimize max(t in Tasks) endOf(itvs[t]);
10 subject to {
11   forall (r in RsrcIds)
12     sum (t in Tasks: t.dmds[r]>0) pulse(itvs[t], t.dmds[r]) <= Capacity[r];
13   forall (t1 in Tasks, t2id in t1.succs)
14     endBeforeStart(itvs[t1],itvs[<t2id>]);
15 }
```

Decisions

- Decision variables:
  - Tasks: array of **interval variables**

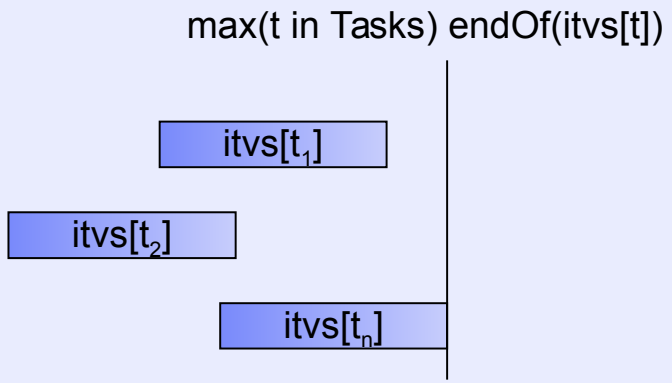


Example #3: Basic RCPSP

```
1  using CP;
2  int NbTasks = ...;
3  int NbRsrcs = ...;
4  range RsrcIds = 1..NbRsrcs;
5  int Capacity[RsrcIds] = ...;
6  tuple Task { key int id; int pt; int dmds[RsrcIds]; {int} succs; }
7  {Task} Tasks = ...;
8  dvar interval itvs[t in Tasks] size t.pt;
9  minimize max(t in Tasks) endOf(itvs[t]);
10 subject to {
11   forall (r in RsrcIds)
12     sum (t in Tasks: t.dmds[r]>0) pulse(itvs[t], t.dmds[r]) <= Capacity[r];
13   forall (t1 in Tasks, t2id in t1.succs)
14     endBeforeStart(itvs[t1],itvs[<t2id>]);
15 }
```

Objective

- Objective:
  - Minimize project makespan

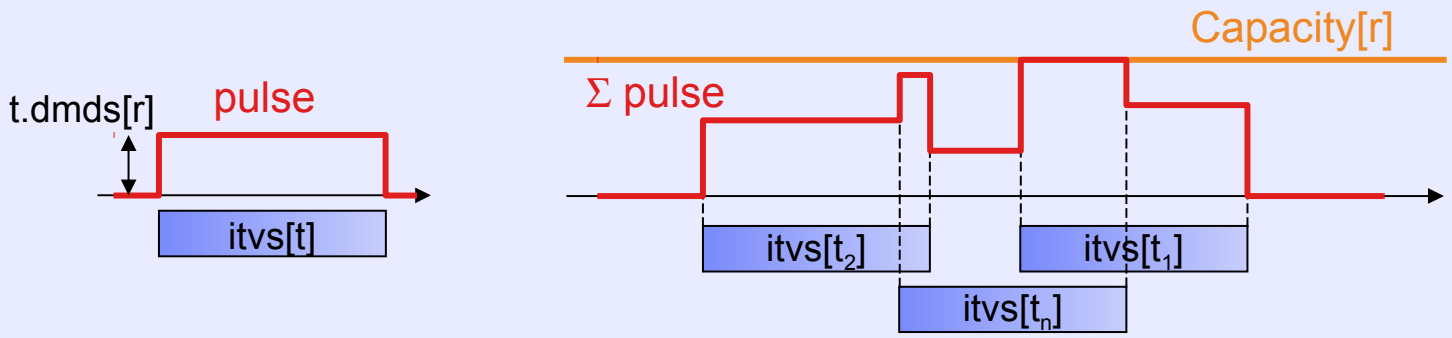


Example #3: Basic RCPSP

```
1  using CP;
2  int NbTasks = ...;
3  int NbRsrcs = ...;
4  range RsrcIds = 1..NbRsrcs;
5  int Capacity[RsrcIds] = ...;
6  tuple Task { key int id; int pt; int dmds[RsrcIds]; {int} succs; }
7  {Task} Tasks = ...;
8  dvar interval itvs[t in Tasks] size t.pt;
9  minimize max(t in Tasks) endOf(itvs[t]);
10 subject to {
11   forall (r in RsrcIds)
12     sum (t in Tasks: t.dmds[r]>0) pulse(itvs[t], t.dmds[r]) <= Capacity[r];
13   forall (t1 in Tasks, t2id in t1.succs)
14     endBeforeStart(itvs[t1],itvs[<t2id>]);
15 }
```

Constraints

- Constraints:
  - Resource capacity constraints (using **cumul functions**)

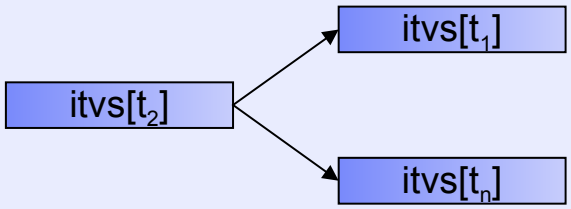


Example #3: Basic RCPSP

```
1  using CP;
2  int NbTasks = ...;
3  int NbRsrcs = ...;
4  range RsrcIds = 1..NbRsrcs;
5  int Capacity[RsrcIds] = ...;
6  tuple Task { key int id; int pt; int dmds[RsrcIds]; {int} succs; }
7  {Task} Tasks = ...;
8  dvar interval itvs[t in Tasks] size t.pt;
9  minimize max(t in Tasks) endOf(itvs[t]);
10 subject to {
11   forall (r in RsrcIds)
12     sum (t in Tasks: t.dmds[r]>0) pulse(itvs[t], t.dmds[r]) <= Capacity[r];
13   forall (t1 in Tasks, t2id in t1.succs)
14     endBeforeStart(itvs[t1],itvs[<t2id>]);
15 }
```

Constraints

- Constraints:
  - **Precedence constraints** between tasks



Modeling with Optional Interval Variables

Optional interval variables

Domain of values for an interval variable *a* :

$$\text{Domain(i)} \subseteq \{\perp\} \cup \{\, [s,e) \mid s,e \in \mathbb{Z},\ s \leq e \,\}$$

Absent

Interval of integers

Initially and during the search interval variable can be:

- Present if $\perp \notin$ Domain(i) (the time can still be unbound).
- Absent if Domain(i) = $\{\perp\}$.
- Optional otherwise.

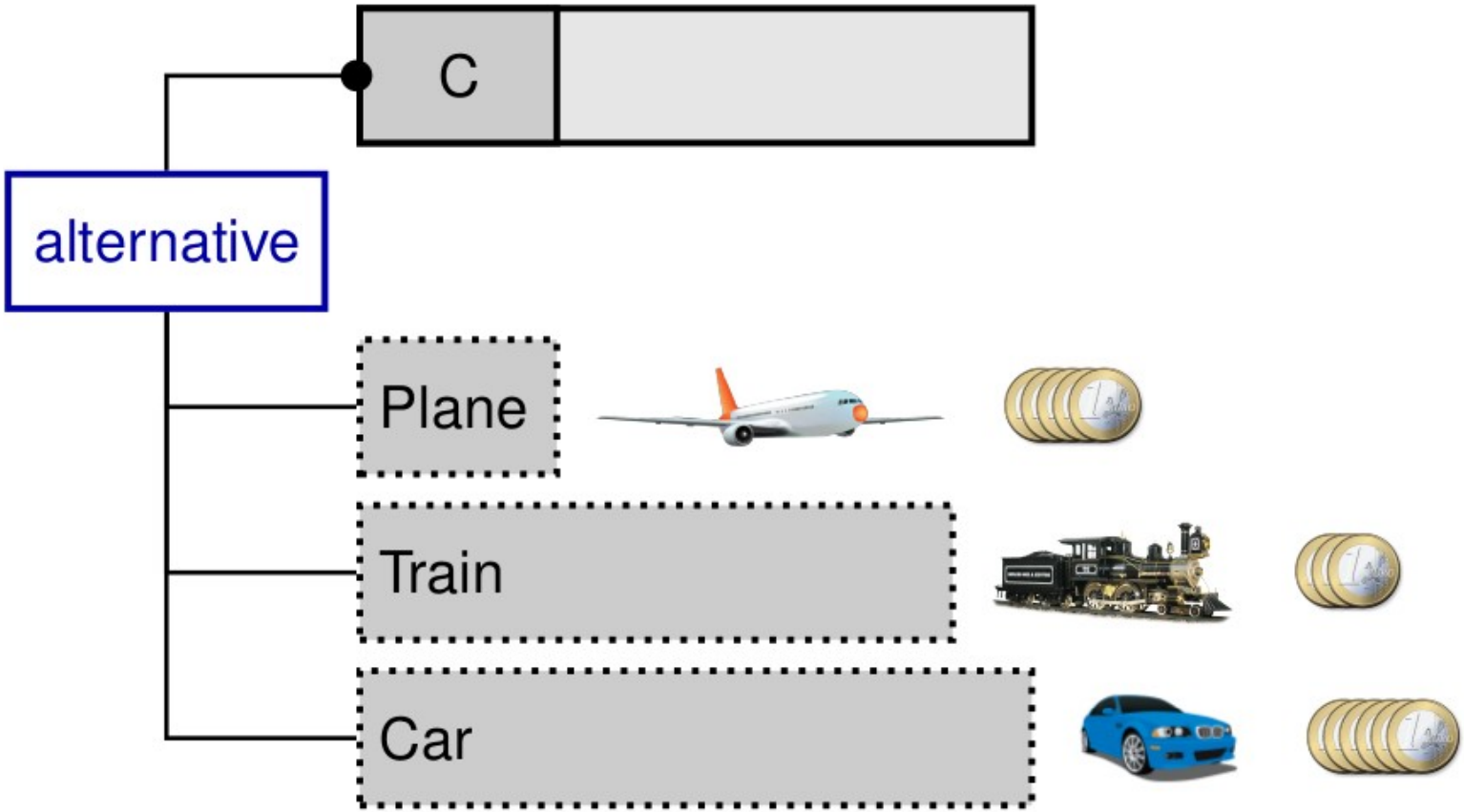
In a solution, interval variable can be:

- Absent i = $\perp$: it is left unperformed.
- Present i = [s, e) : it starts at time s and ends at time e .

Absent intervals are ignored by most of constraints.

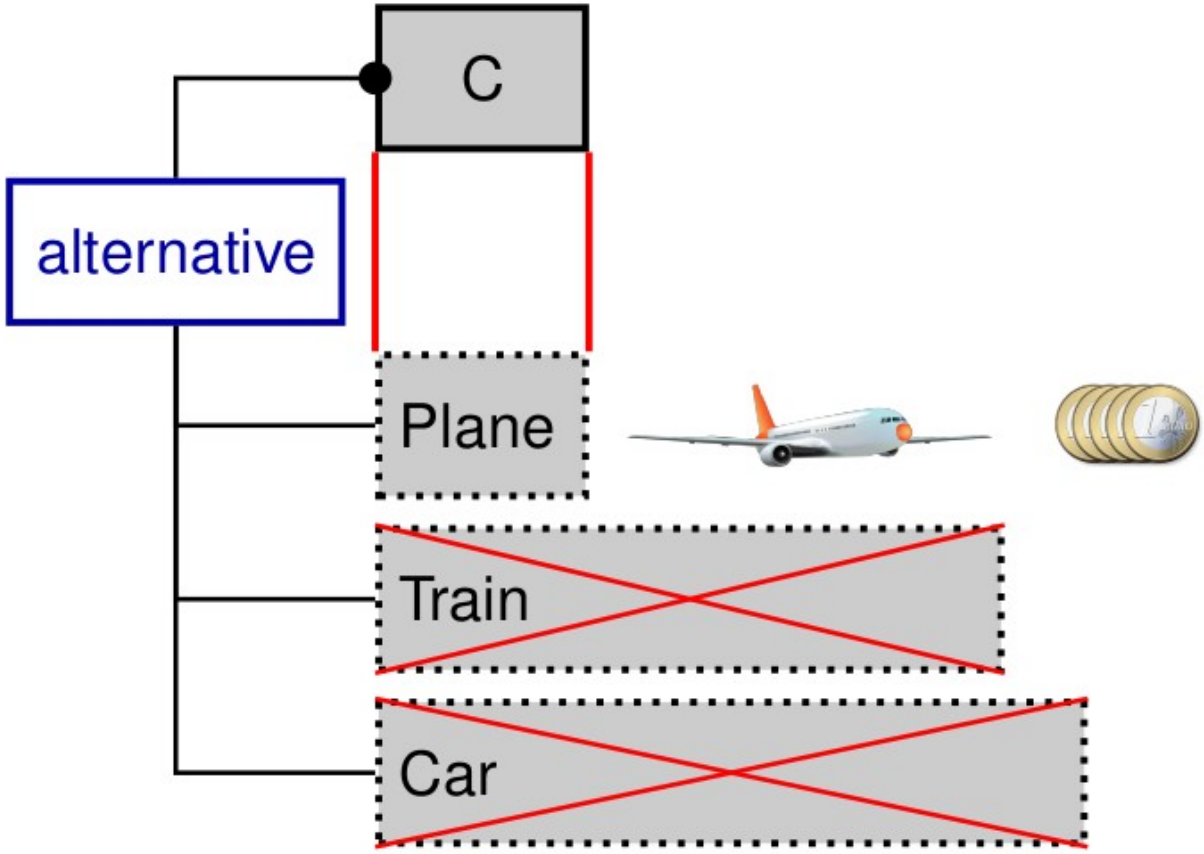
Alternative constraint

alternative(C, [Plane, Train, Car])



Alternative constraint

alternative(C, [Plane, Train, Car])



C = [10, 15)

Plane = [10, 15)

Train = ⊥

Car = ⊥

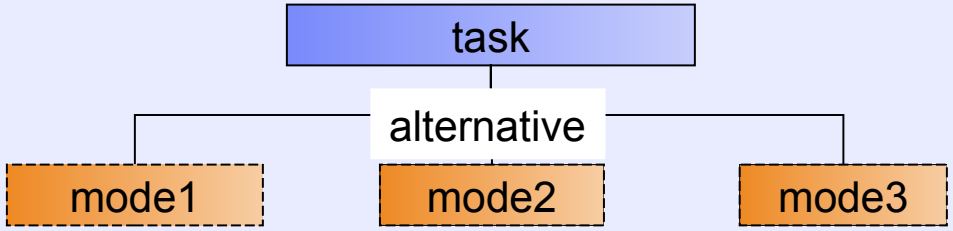
Example #4:

Multi-Mode RCPSP

Example #4: Multi-Mode RCPSP

Multi-Mode RCPSP

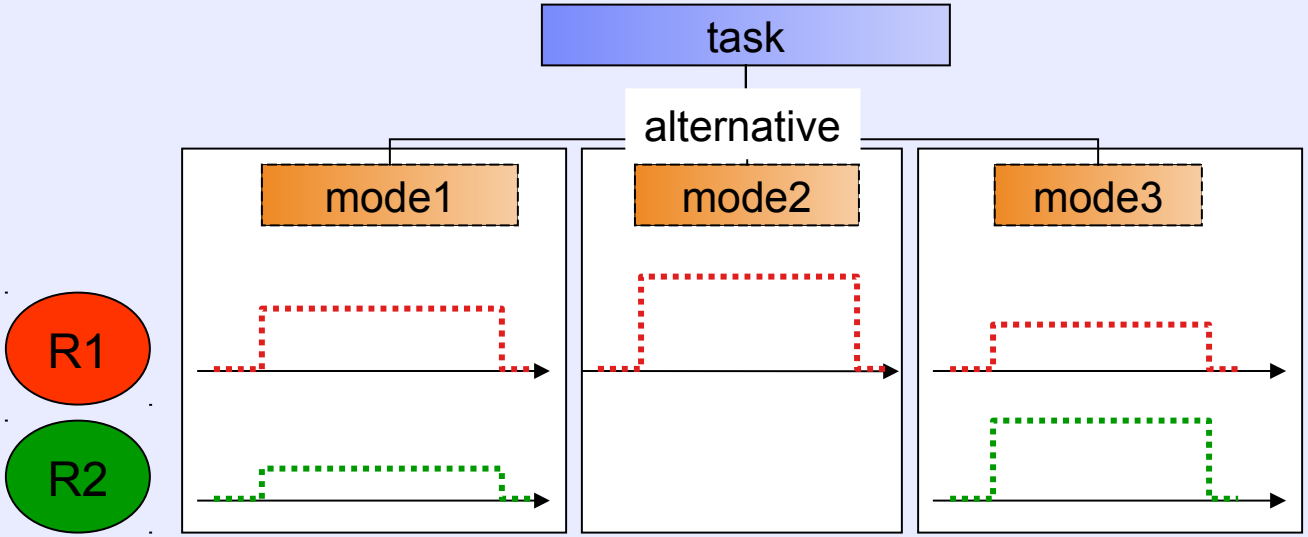
- Each task must be one among several alternative execution modes



Example #4: Multi-Mode RCPSP

Multi-Mode RCPSP

- Each task must be one among several alternative execution modes



Example #4: Multi-Mode RCPSP

```
using CP;
int NbTasks    = ...;
int NbRRsrcs   = ...;
int NbNRsrcs   = ...;
range RRsrcIds = 1..NbRRsrcs;
range NRsrcIds = 1..NbNRsrcs;
int CapRRsrc [RRsrcIds]  = ...;
int CapNRsrc [NRsrcIds] = ...;
tuple Task { key int id; {int} succs; }
{Task} Tasks = ...;
tuple Mode { int taskId; int id; int pt; int dmdR[RRsrcIds]; int dmdN[NRsrcIds]; }
{Mode} Modes = ...;
dvar interval task[t in Tasks];
dvar interval mode[m in Modes] optional  size m.pt;
minimize max(t in Tasks) endOf(task[t]);
subject to {
  forall (t in Tasks)
    alternative(task[t], all(m in Modes: m.taskId==t.id) mode[m]);
  forall (r in RRsrcIds)
    sum (m in Modes: m.dmdR[r]>0) pulse(mode[m], m.dmdR[r]) <= CapRRsrc[r];
  forall (r in NRsrcIds)
    sum (m in Modes: m.dmdN[r]>0) m.dmdN[r]*presenceOf(mode[m]) <= CapNRsrc[r];
  forall (t1 in Tasks, t2id in t1.succs)
    endBeforeStart(task[t1], task[<t2id>]);
}
```

- Alternative constraint:
  - Mode selection for each task (using **alternative** constraints)
  - Resource usage for each mode

Example #4: Multi-Mode RCPSP

```
1  using CP;
2  int NbTasks    = ...;
3  int NbRRsrcs   = ...;
4  int NbNRsrcs   = ...;
5  range RRsrcIds = 1..NbRRsrcs;
6  range NRsrcIds = 1..NbNRsrcs;
7  int CapRRsrc [RRsrcIds]  = ...;
8  int CapNRsrc [NRsrcIds] = ...;
9  tuple Task { key int id; {int} succs; }
10 {Task} Tasks = ...;
11 tuple Mode { int taskId; int id; int pt; int dmdR[RRsrcIds]; int dmdN[NRsrcIds]; }
12 {Mode} Modes = ...;
13 dvar interval task[t in Tasks];
14 dvar interval mode[m in Modes] optional  size m.pt;
15 minimize max(t in Tasks) endOf(task[t]);
16 subject to {
17   forall (t in Tasks)
18     alternative(task[t], all(m in Modes: m.taskId==t.id) mode[m]);
19   forall (r in RRsrcIds)
20     sum (m in Modes: m.dmdR[r]>0) pulse(mode[m], m.dmdR[r]) <= CapRRsrc[r];
21   forall (r in NRsrcIds)
22     sum (m in Modes: m.dmdN[r]>0) m.dmdN[r]*presenceOf(mode[m]) <= CapNRsrc[r];
23   forall (t1 in Tasks, t2id in t1.succs)
24     endBeforeStart(task[t1], task[<t2id>]);
25 }
```

- Non-renewable resources (e.g. budget):
  - Modeled using inequality constraint: 5*presenceOf(x) + 3*presenceOf(y) + … <= 15
  - Expression presenceOf has value 1 or 0.

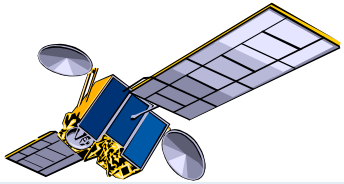
Example #5:

Oversubscribed Scheduling

Example #5: Oversubscribed Scheduling

- USAF Satellite Control Network scheduling problem [6]
- A set of *n* communication requests for Earth orbiting satellites must be scheduled on a total of 32 antennas spread across 13 ground-based tracking stations.
- Objective is to maximize the number of satisfied requests
- In the instances, *n* ranges from to 400 to 1300

Example #5: Oversubscribed Scheduling

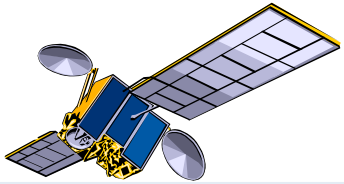


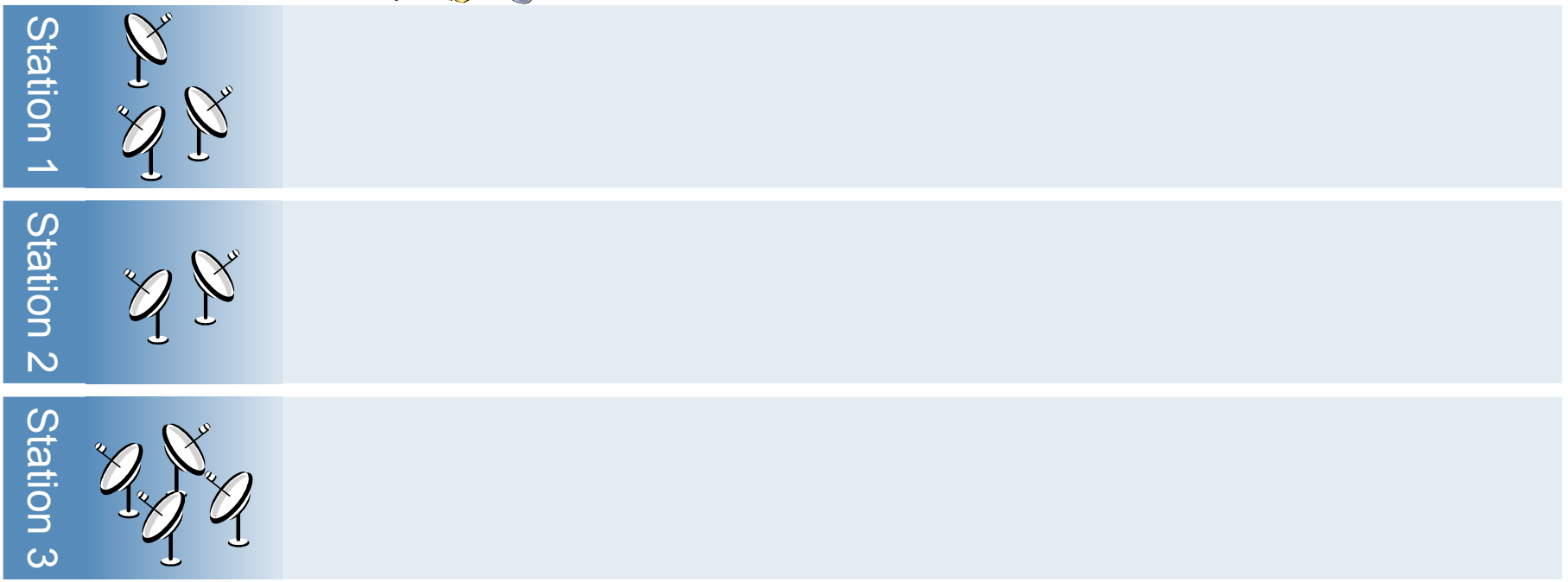
Station 1

Station 2

Station 3

Example #5: Oversubscribed Scheduling

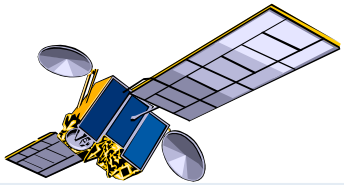


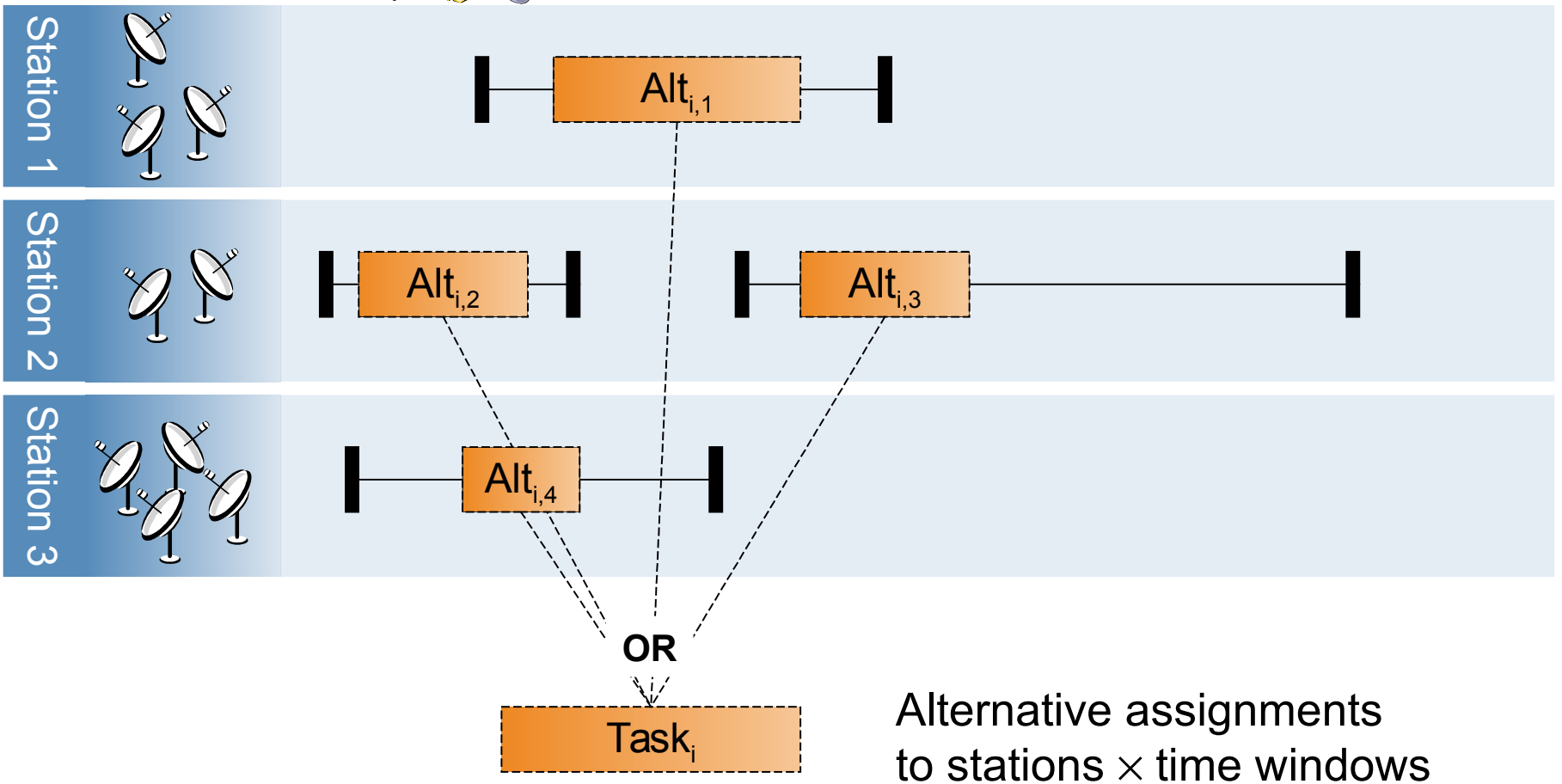


$Task_i$

Communication requests

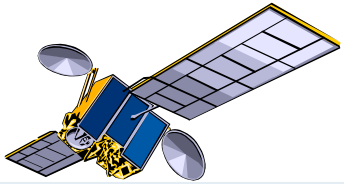
Example #5: Oversubscribed Scheduling





Alternative assignments
to stations × time windows

Example #5: Oversubscribed Scheduling



Station 1

$Alt_{i,1}$

Station 2

$Alt_{i,2}$

$Alt_{i,3}$

Station 3

$Alt_{i,4}$

$Task_i$

Selected alternative will use 1 antenna for communication with the satellite

Example #5: Oversubscribed Scheduling

```
1  using CP;
2  tuple Station { string name; int id; int cap; }
3  tuple Alternative { string task; int station; int smin; int dur; int emax; }
4  {Station} Stations = ...;
5  {Alternative} Alternatives = ...;
6  {string} Tasks = { a.task | a in Alternatives };
7  dvar interval task[t in Tasks] optional;
8  dvar interval alt[a in Alternatives] optional in a.smin..a.emax size a.dur;
9  maximize sum(t in Tasks) presenceOf(task[t]);
10 subject to {
11   forall(t in Tasks)
12     alternative(task[t], all(a in Alternatives: a.task==t) alt[a]);
13   forall(s in Stations)
14     sum(a in Alternatives: a.station==s.id) pulse(alt[a],1) <= s.cap;
15 }
```

Data

- Data reading and computation:
  - Tuple sets of Stations and Alternative assignments
  - Tasks: set of tasks of the problem

Example #5: Oversubscribed Scheduling

```
1  using CP;
2  tuple Station { string name; int id; int cap; }
3  tuple Alternative { string task; int station; int smin; int dur; int emax; }
4  {Station} Stations = ...;
5  {Alternative} Alternatives = ...;
6  {string} Tasks = { a.task | a in Alternatives };
7  dvar interval task[t in Tasks] optional;
8  dvar interval alt[a in Alternatives] optional in a.smin..a.emax size a.dur;
9  maximize sum(t in Tasks) presenceOf(task[t]);
10 subject to {
11   forall(t in Tasks)
12     alternative(task[t], all(a in Alternatives: a.task==t) alt[a]);
13   forall(s in Stations)
14     sum(a in Alternatives: a.station==s.id) pulse(alt[a],1) <= s.cap;
15 }
```

Decisions

- Decision variables:
  - Tasks: array of **optional** interval variables
  - Alternative assignments: array of **optional** interval variables,
    each possible assignment is defined with a specific time-window ([smin,emax]) and a size

Example #5: Oversubscribed Scheduling

```
1  using CP;
2  tuple Station { string name; int id; int cap; }
3  tuple Alternative { string task; int station; int smin; int dur; int emax; }
4  {Station} Stations = ...;
5  {Alternative} Alternatives = ...;
6  {string} Tasks = { a.task | a in Alternatives };
7  dvar interval task[t in Tasks] optional;
8  dvar interval alt[a in Alternatives] optional in a.smin..a.emax size a.dur;
9  maximize sum(t in Tasks) presenceOf(task[t]);
10 subject to {
11   forall(t in Tasks)
12     alternative(task[t], all(a in Alternatives: a.task==t) alt[a]);
13   forall(s in Stations)
14     sum(a in Alternatives: a.station==s.id) pulse(alt[a],1) <= s.cap;
15 }
```

Objective

- Objective:
  - Maximize number of executed tasks (modeled with a sum of **presenceOf** constraints)

Example #5: Oversubscribed Scheduling

```
1  using CP;
2  tuple Station { string name; int id; int cap; }
3  tuple Alternative { string task; int station; int smin; int dur; int emax; }
4  {Station} Stations = ...;
5  {Alternative} Alternatives = ...;
6  {string} Tasks = { a.task | a in Alternatives };
7  dvar interval task[t in Tasks] optional;
8  dvar interval alt[a in Alternatives] optional in a.smin..a.emax size a.dur;
9  maximize sum(t in Tasks) presenceOf(task[t]);
10 subject to {
11   forall(t in Tasks)
12     alternative(task[t], all(a in Alternatives: a.task==t) alt[a]);
13   forall(s in Stations)
14     sum(a in Alternatives: a.station==s.id) pulse(alt[a],1) <= s.cap;
15 }
```

Constraints

- Constraints:
  - Alternative assignments for a given task t using an **alternative constraint**
  - Maximal capacity of stations (number of antennas) using a constrained **cumul function**