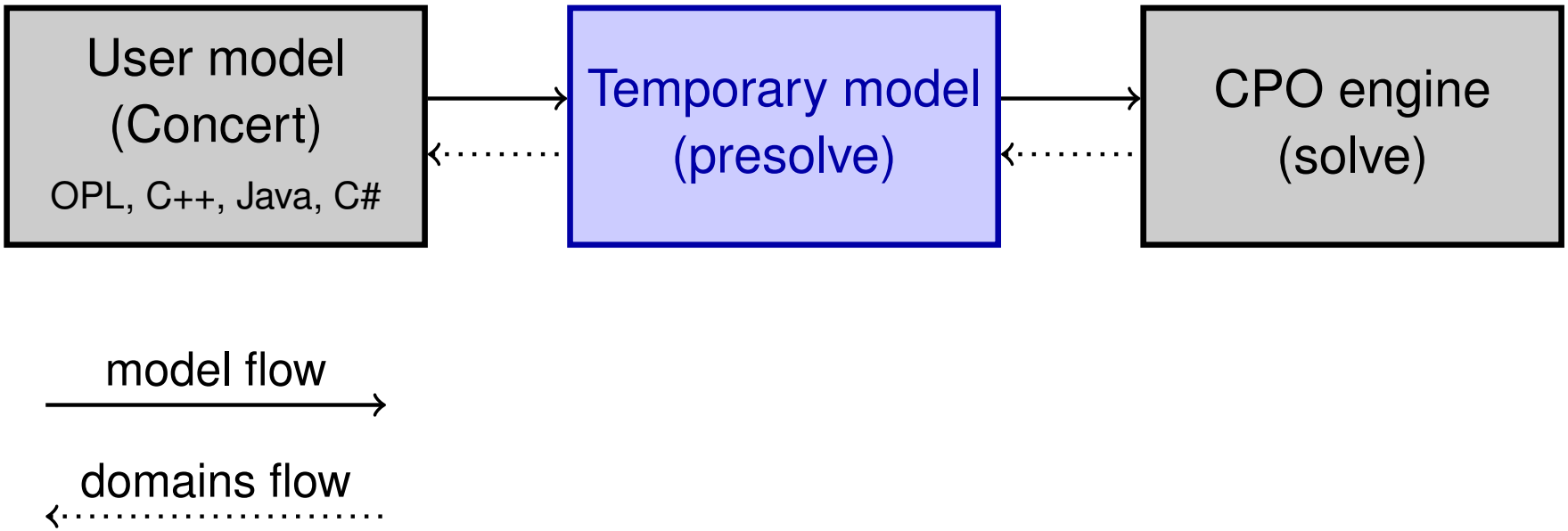
Presolve

CP Optimizer scheme (new)

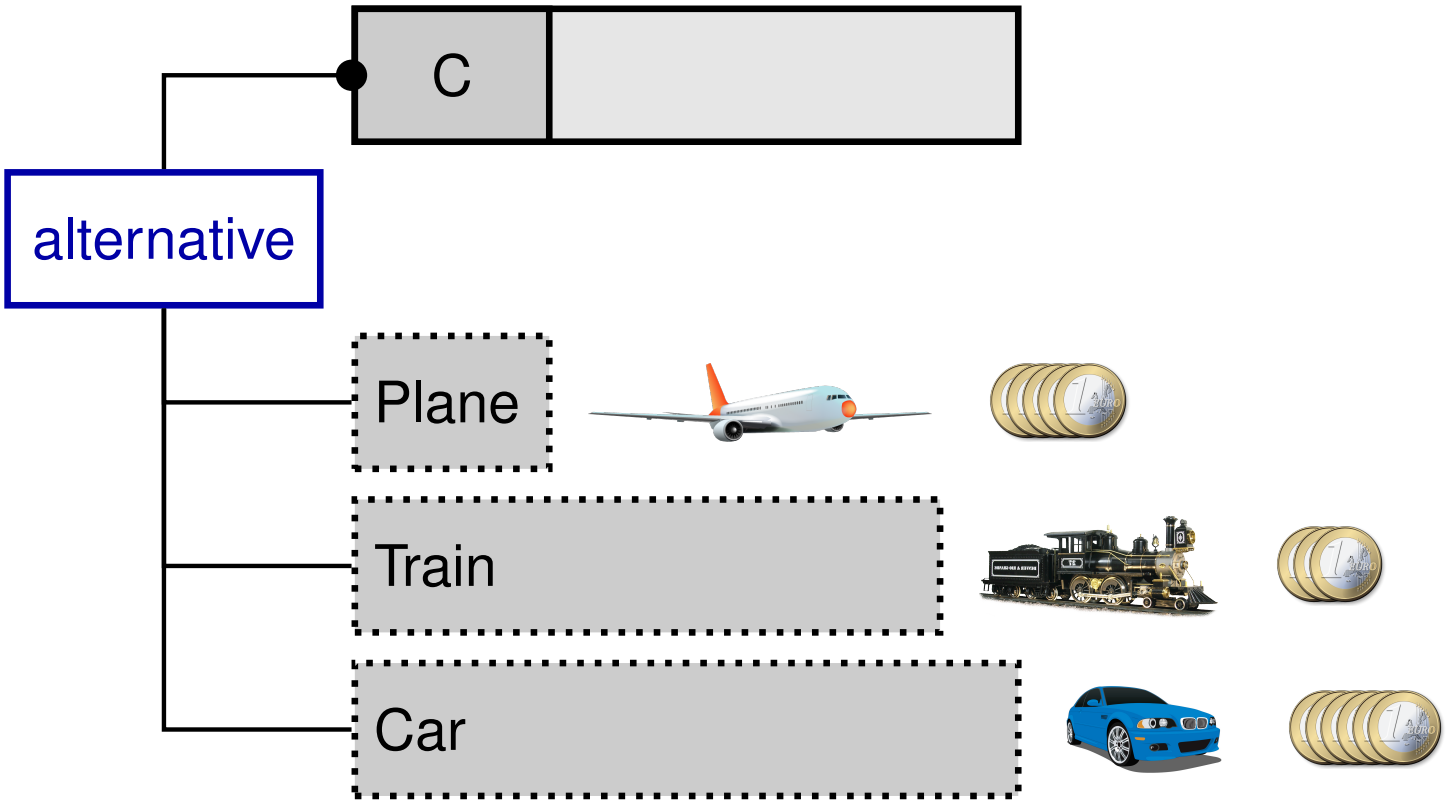


- All presolves are done in the temporary model.
- Each module uses completely different way to store the model (as the requirements are different).

Examples of presolves

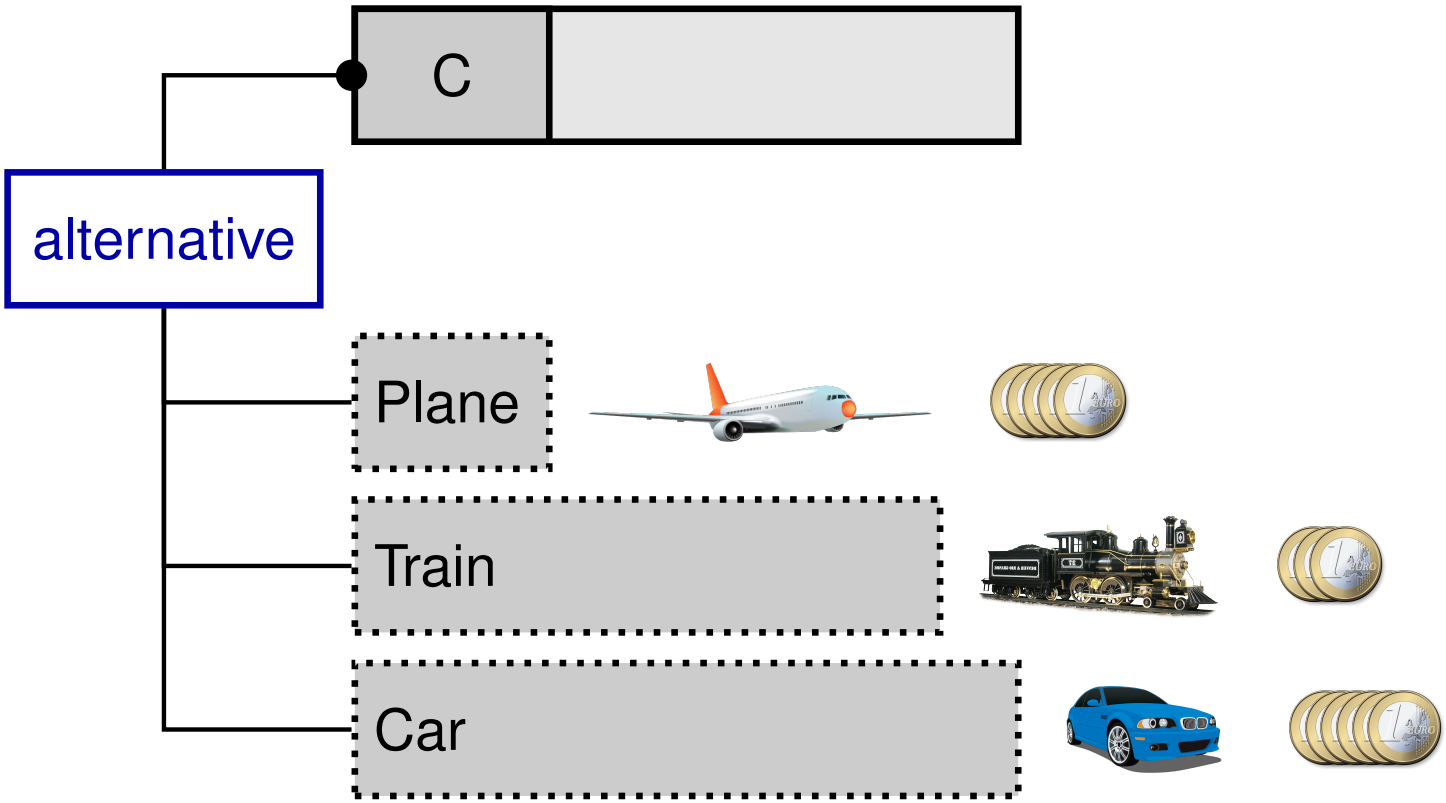
- Partial expression evaluation
- Common sub-expression elimination
- Precedence strengthening
  - If a and b cannot overlap and **startsBeforeStart**(a,b)
  - Then **endsBeforeStart**(a,b)
- Precedence recognition
  - If **endOf**$(a, -\infty) \leq$ **startOf**(b, ∞)
  - Then **endsBeforeStart**(a,b)
  - Precedences are aggreated into “time net” for faster and stronger propagation.
- 2-SAT clauses recognition
  - **presenceOf**$(a) \leq$ **presenceOf**(b)
  - **presenceOf**$(a) =$ **presenceOf**(b)
  - Such clauses are aggregated into “logical net” for stronger propagation.
- Strong constraint

Cost of an alternative




```
alternative(C, [Plane, Train, Car]);
cost = 5 * presenceOf(Plane) +
       3 * presenceOf(Train) +
       6 * presenceOf(Car);
```

Cost of an alternative




```
alternative(C, [Plane, Train, Car]);
cost = 5 * presenceOf(Plane) +
       3 * presenceOf(Train) +
       6 * presenceOf(Car);
```

It does not propagate well:
$\texttt{cost} \in [0, 14]$ versus $[3, 6]$.

The problem: How to model alternative cost?

It is a design problem for the modelling language.

```
alternative(C, [Plane,Train,Car], [5,3,6], cost);
```

- Does not work for hierarchy of alternatives.
- Mixes constraint with objective.

```
cost = 3 + 2*!presenceOf(Train) + presenceOf(Car);
```

- No one will write this.
- Hard to extend to more variables.

```
alternative(C, [Plane,Train,Car], indexVar);
cost = element(indexVar, [5,3,6])
```

- Requires variable `indexVar`.
- Does not work for hierarchy of alternatives.

The problem: How to model alternative cost?

IBM

It is a design problem for the modelling language.

```
alternative(C, [Plane,Train,Car], [5,3,6], cost);
```

- Does not work for hierarchy of alternatives.
- Mixes constraint with objective.

```
cost = 3 + 2*!presenceOf(Train) + presenceOf(Car);
```

- No one will write this.
- Hard to extend to more variables.

```
alternative(C, [Plane,Train,Car], indexVar);
cost = element(indexVar, [5,3,6])
```

- Requires variable `indexVar`.
- Does not work for hierarchy of alternatives.

All those possibilities are clumsy and non-intuitive.

Alternative cost

Our decision:

- Use the simplest expression:

```
cost = 5 * presenceOf(Plane) +
       3 * presenceOf(Train) +
       6 * presenceOf(Car);
```

- Presolve it into expression that propagates better.

Benefits:

- Intuitive language, no specialized function to learn.
- Easy to upgrade. No need to rewrite the model.
- Internal implementation can change at any time.

Alternative cost

Our decision:

- Use the simplest expression:

```
cost = 5 * presenceOf(Plane) +
       3 * presenceOf(Train) +
       6 * presenceOf(Car);
```

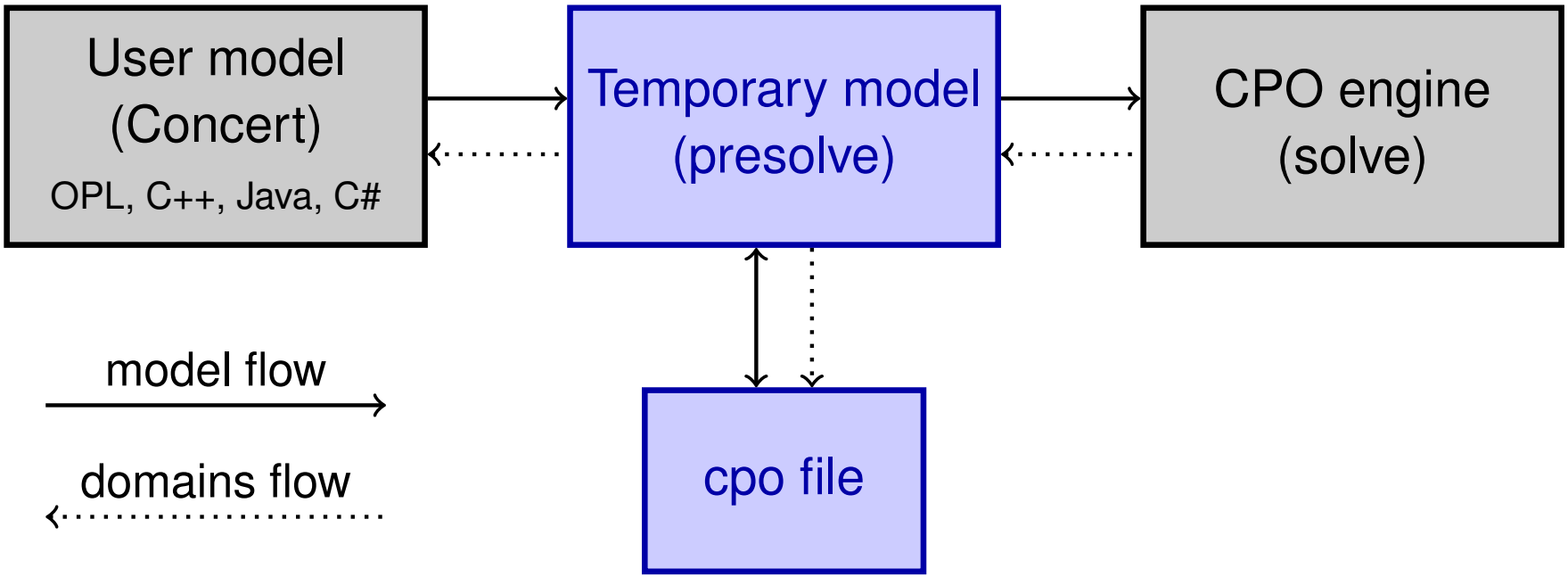
- Presolve it into expression that propagates better.

Benefits:

- Intuitive language, no specialized function to learn.
- Easy to upgrade. No need to rewrite the model.
- Internal implementation can change at any time.

Capabilities of presolve affect language design.

CP Optimizer scheme (new)



Facilities of cpo files:

- Export model before/instead solve.
- Import model instead of normal modelling.
- Export model during solve (with current domains).
- Developers only: export model after presolve.

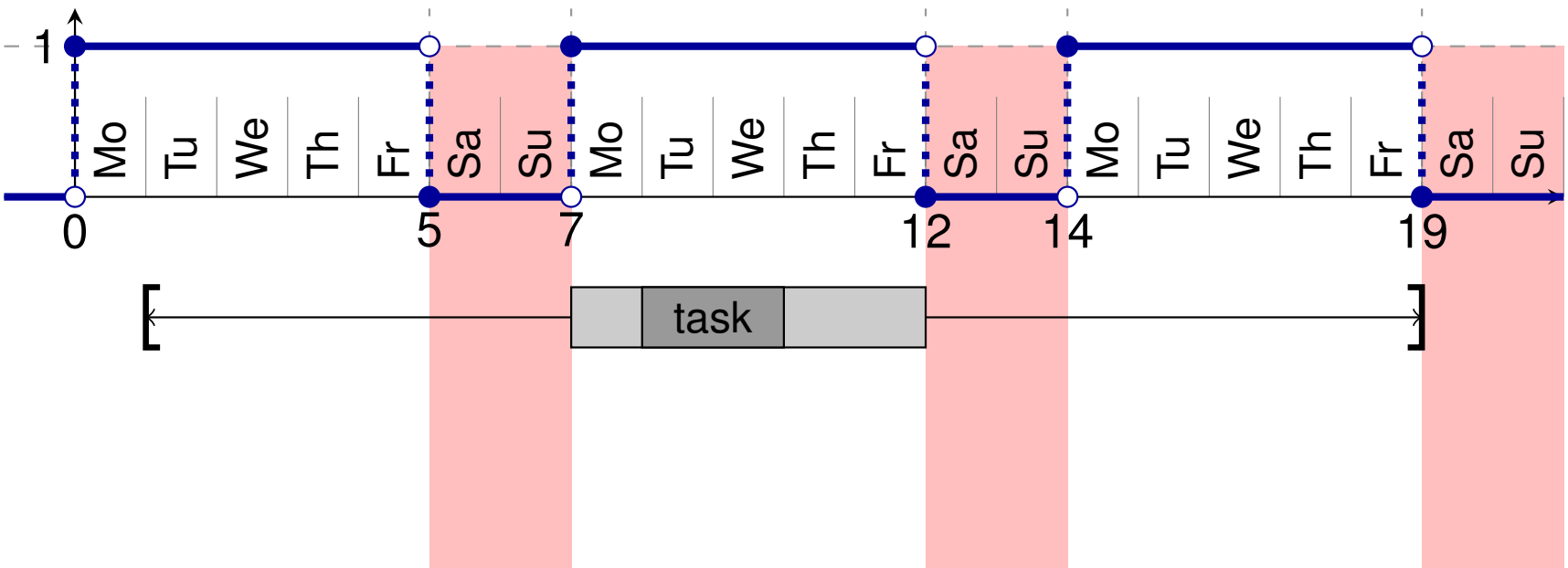
When I want to improve performance of some model:

- Dump the model multiple times during the solve.
  - E.g. failing nodes or branches that is hard to escape.
  - Those models contain current domains.
  - And they are typical infeasible.
- Import those models back and use conflict refiner to find minimum infeasible submodel (conflict).
- See if a pattern emerge and look for improvements:
  - Add redundant constraint?
  - Improve propagation of some constraint?
  - Add some presolve? Add `strong` constraint?

Often I think that I know how the model and solver is working. And then I'm surprised by the result of this analysis.

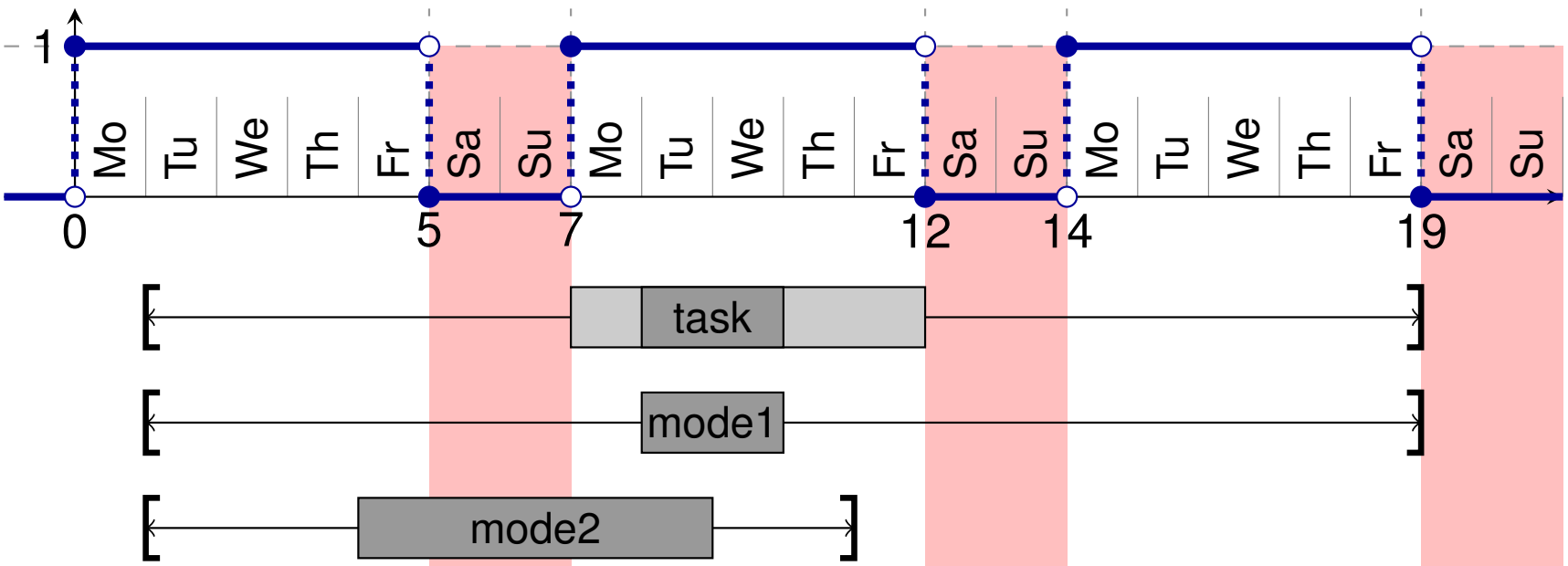
Example of a conflict

```
task = intervalVar(present, length=2..5, start=1..17, end=3..19);
weekends = stepFunction((0,1),(5,0),(7,1),(12,0),(14,1),(19,0));
forbidExtent(task, weekends)
```



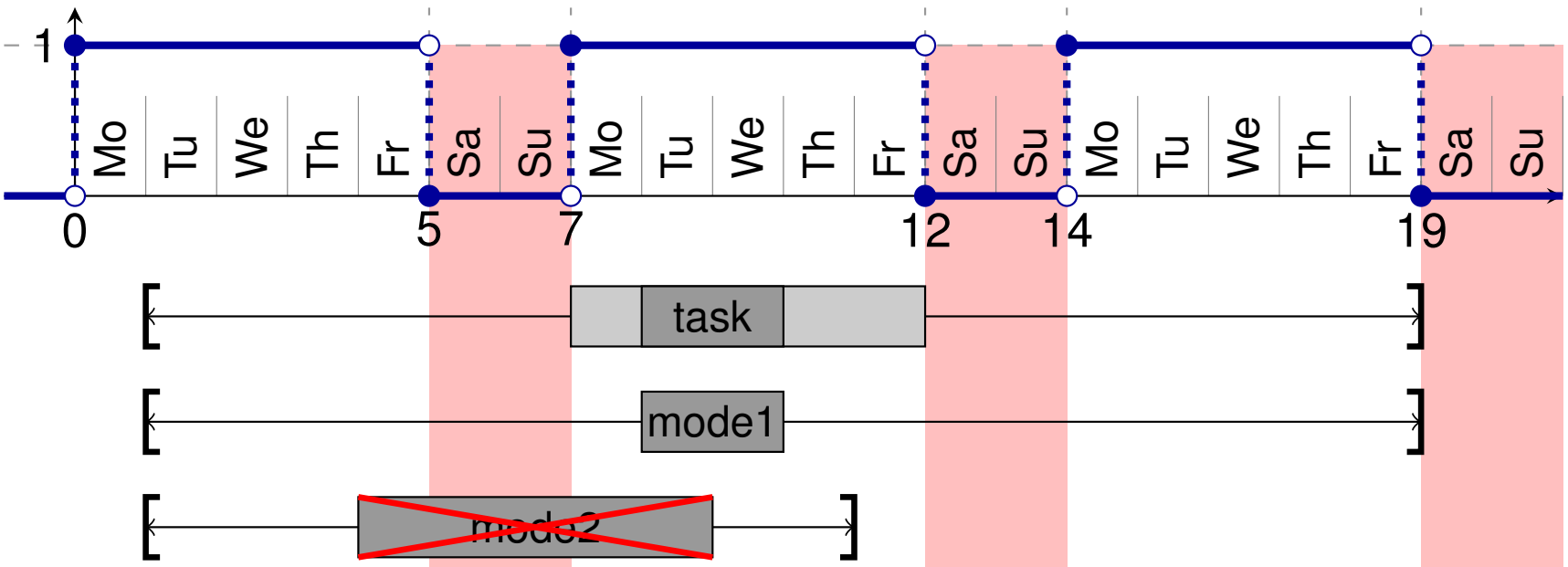
Example of a conflict

```
task = intervalVar(present, length=2..5, start=1..17, end=3..19);
weekends = stepFunction((0,1),(5,0),(7,1),(12,0),(14,1),(19,0));
forbidExtent(task, weekends)

mode1 = intervalVar(optional, length=2, start=1..17, end=3..19);
mode2 = intervalVar(optional, length=5, start=1..6, end=6..11);
alternative(task, [mode1, mode2]);
```



Example of a conflict

```
task = intervalVar(present, length=2..5, start=1..17, end=3..19);
weekends = stepFunction((0,1),(5,0),(7,1),(12,0),(14,1),(19,0));
forbidExtent(task, weekends)

mode1 = intervalVar(optional, length=2, start=1..17, end=3..19);
mode2 = intervalVar(optional, length=5, start=1..6, end=6..11);
alternative(task, [mode1, mode2]);
presenceOf(mode2); // Search decision
```



Example of a conflict

```
task = intervalVar(present, length=2..5, start=1..17, end=3..19);
weekends = stepFunction((0,1),(5,0),(7,1),(12,0),(14,1),(19,0));
forbidExt
```

New presolve: alternatives should inherit **forbidExtent** constraint from the master.

```
mode1 = i                                                    nd=3..19);
mode2 = intervalVar(optional, length=5, start=1..6, end=6..11);
alternative(task, [mode1, mode2]);
presenceOf(mode2); // Search decision
```