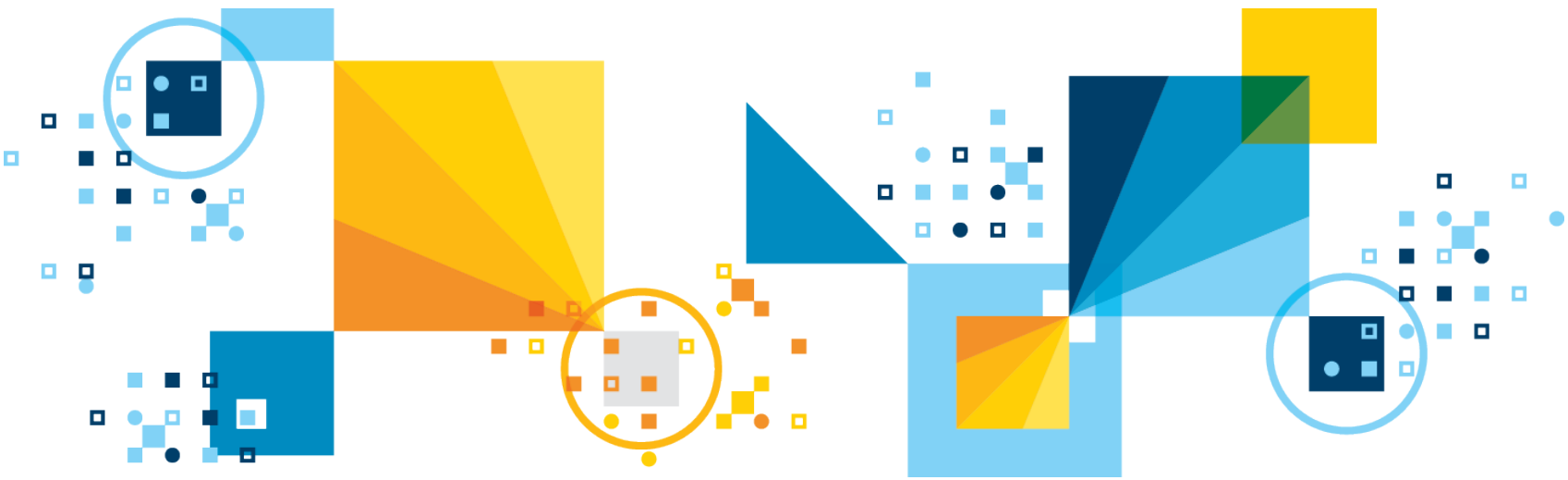
Making of a commercial constraint-based scheduling solver

Petr Vilím

IBM Czech Republic

May 17, 2015



Outline

From my talk in 2011:

- What are the features of an ideal/future industrial solver.
- How this vision drives development of CP Optimizer.
  - CP Optimizer versus ILOG Solver and Scheduler.
  - Optional interval variables
  - Automatic search
  - ...

This talk:

- Revisit features of ideal solver.
- Review advances of CP Optimizer and design changes.
  - presolve, automatic search, modeling aids (warnings and failure explanations), cpo file format, optimization as a service, isomorphism constraint, ..

Ideal solver (from business point of view)

- "Model and Run" paradigm
  - Rich and intuitive modelling language
  - Strong default search
  - Explain problems
  - Optimization as a service
- State of the art performance
  - Portfolio of methods (CP/AI/OR) and hybrids
  - Model analysis
  - Machine learning
- Easy to get support

Ideal solver (from business point of view)

- "Model and Run" paradigm
  - Rich and intuitive modelling language →2011, isomorphism
  - Strong default search → 2011, FDS
  - Explain problems → warnings, failure explanations
  - Optimization as a service → demo
- State of the art performance
  - Portfolio of methods (CP/AI/OR) and hybrids → 2011
  - Model analysis → presolve
  - Machine learning → 2011
- Easy to get support → cpo file format

Presolve

Interval variable, optionality

Interval variable models possibly optional activity that has start and end time. Its domain is:

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

Initially and during the search interval variable can be:

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

In a solution, interval variable can be:

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

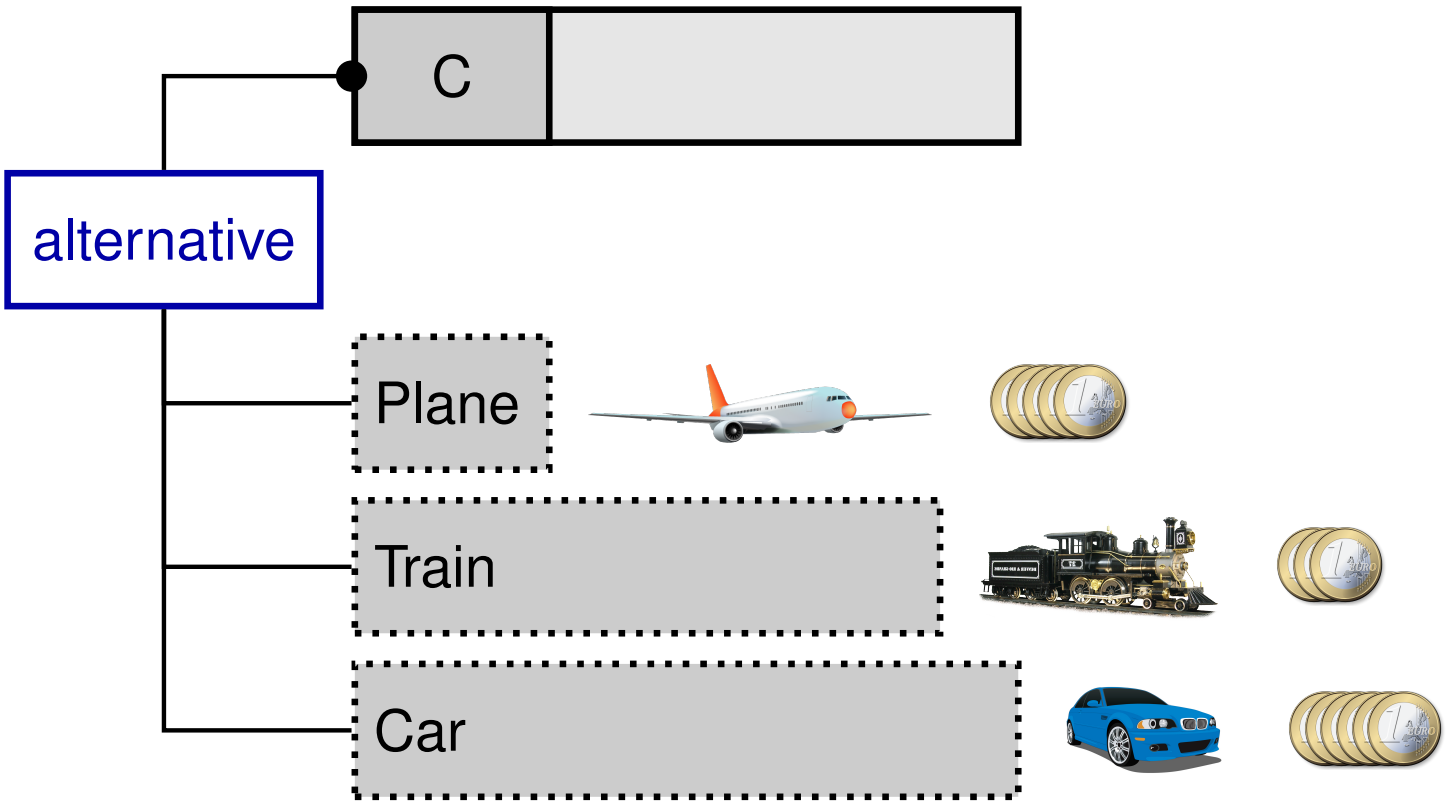
Semantics of optional interval variables

- Most constraints ignore absent intervals. For example:

  endsBeforeStart(a, b)

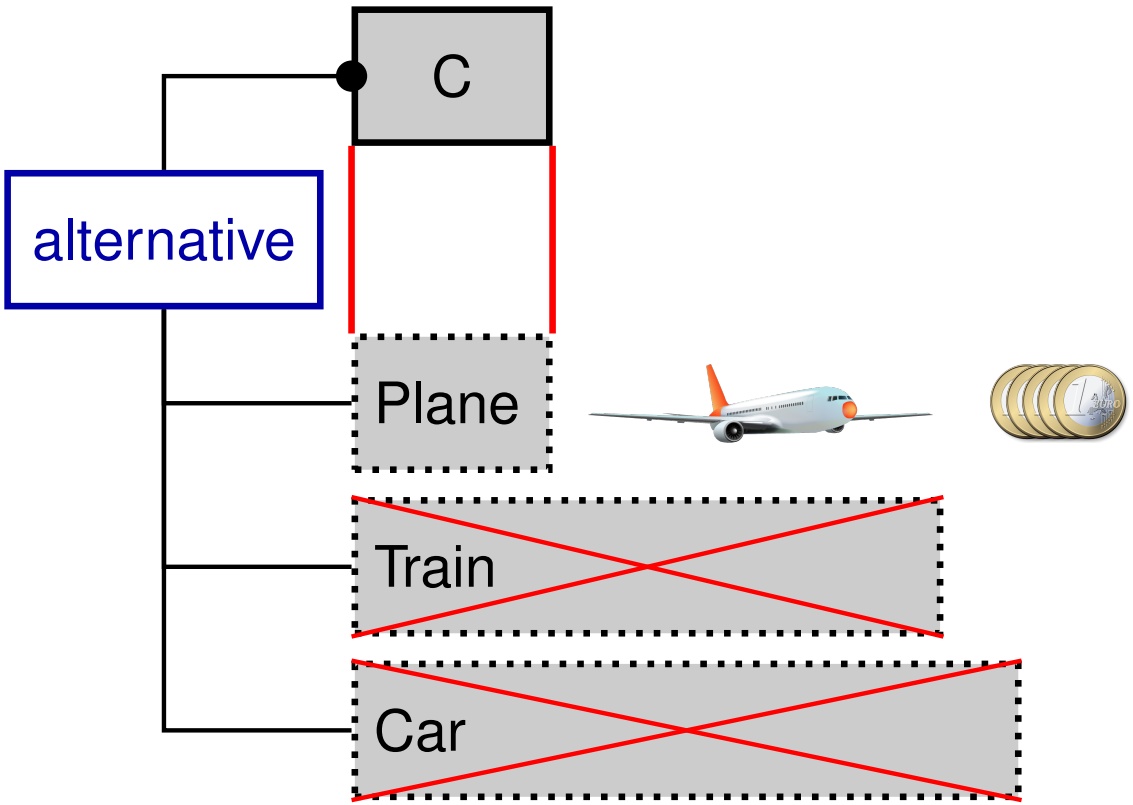
  is automatically satisfied if a or b are absent.
- Resource requirements of absent intervals are ignored.
- There are "accessor functions" for attributes of interval variable:
  - **presenceOf**(a): a constraints, values 0/1.
  - **startOf**(a): integer expression, 0 when absent.
  - **endOf**(a, 1000): integer expression, 1000 when absent.
  - **lengthOf**(a): integer expression, 0 when absent.

Alternative constraint



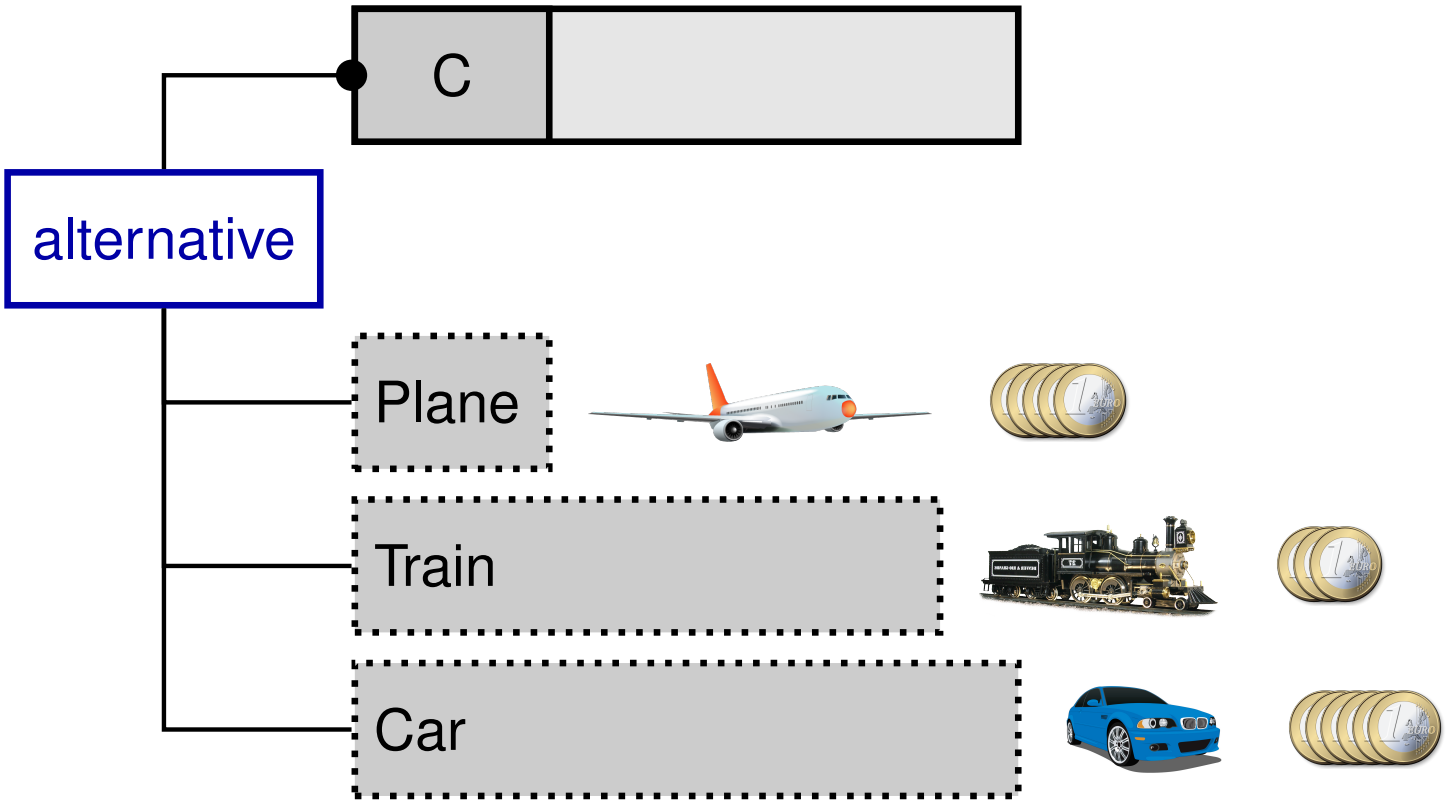

```
alternative(C, [Plane, Train, Car]);
```

Alternative constraint



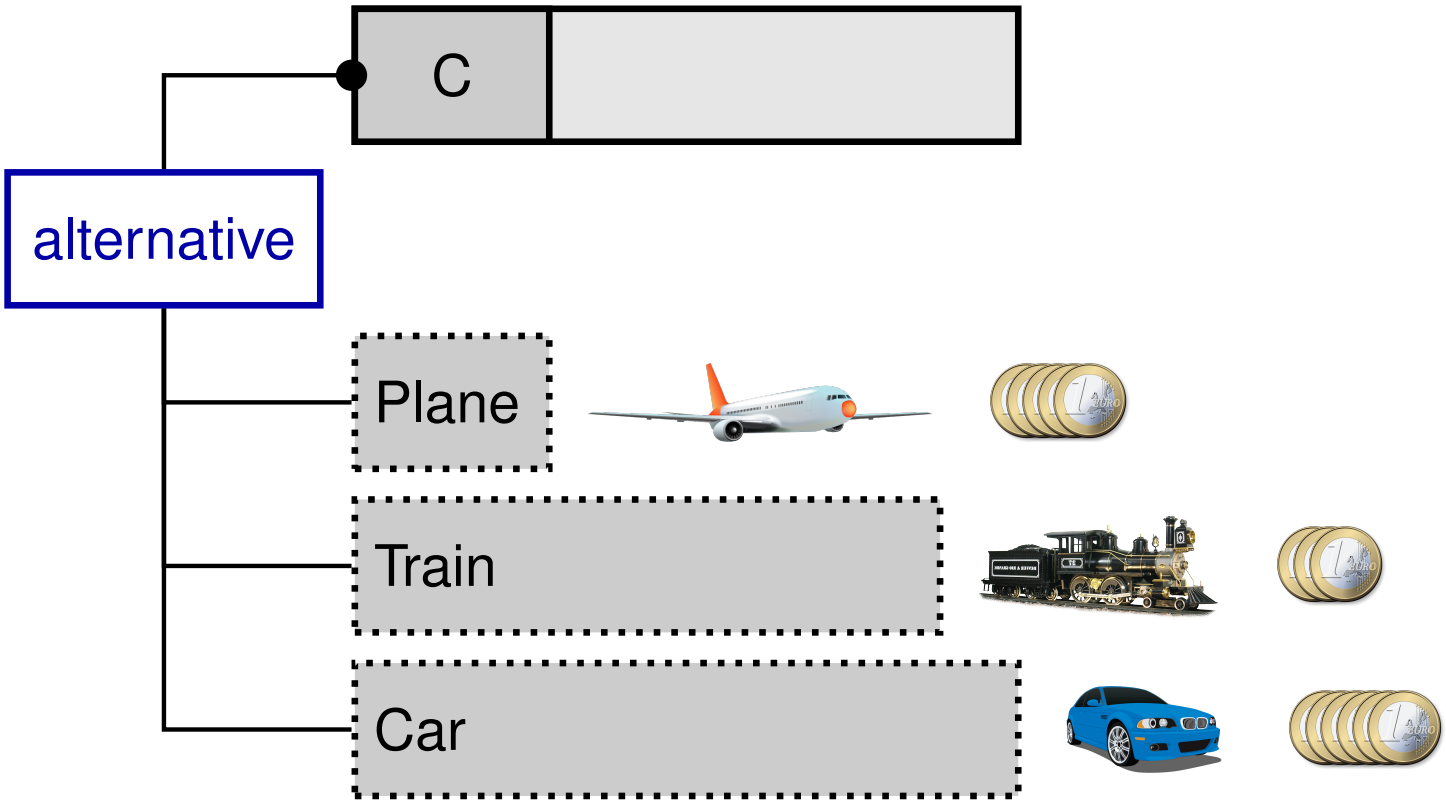

```
alternative(C, [Plane, Train, Car]);
```

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);
```

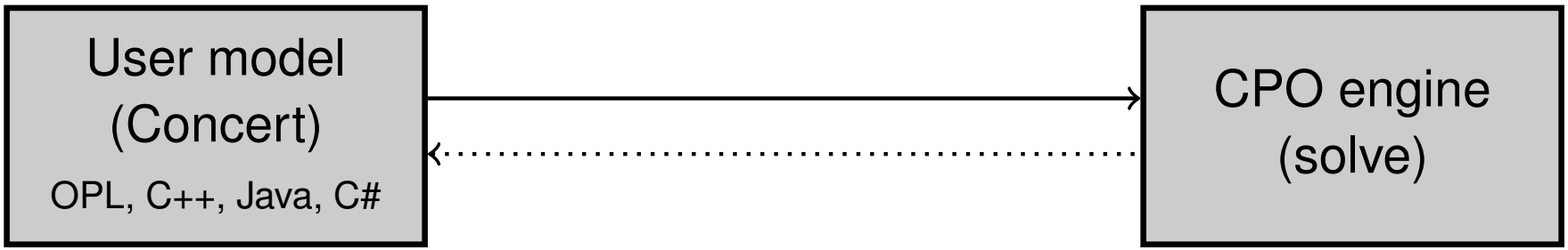
- Presolve it into expression that propagates better.

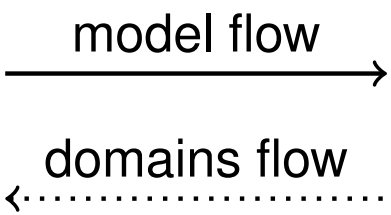
Benefits:

- 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.

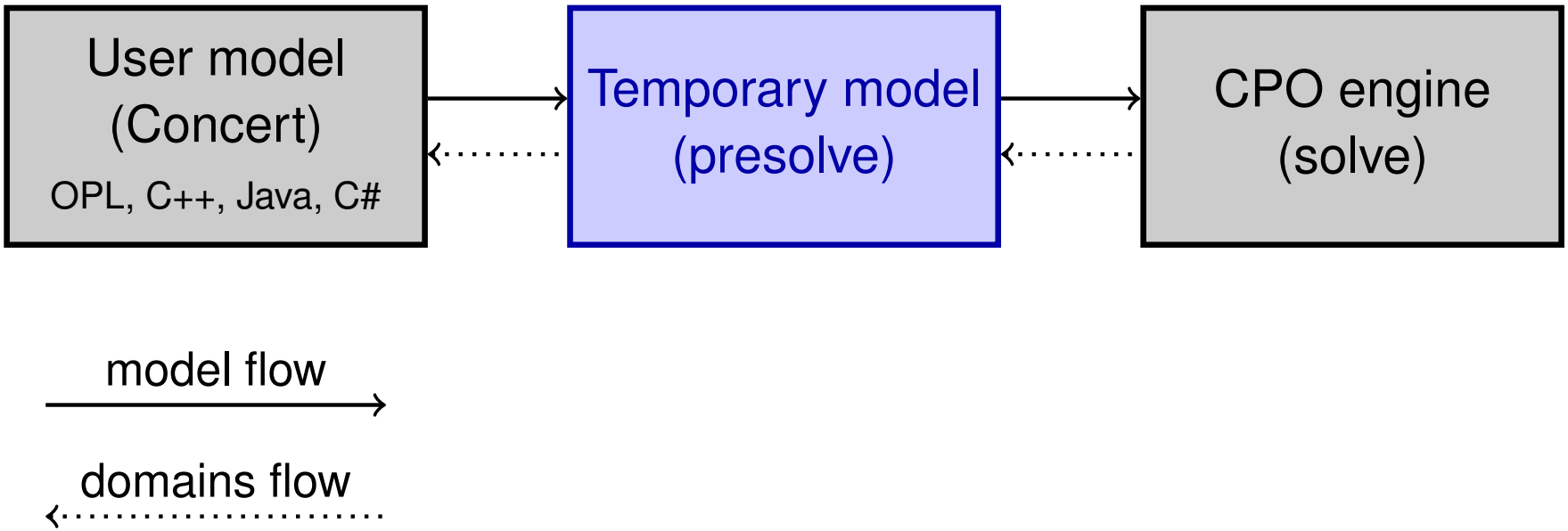
CP Optimizer scheme (old)





- Limitted presolve done during translation of user model into engine.

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

Strong constraint

- Sometimes, there is a small group of variables tightly tied together by a set of constraint.
  - However those constraint do not propagate well together (global view is missing).
- Standard recommendation used to be to replace those constraint by a table constraint (`allowedAssignments` or `forbiddenAssignments`).
  - However it is a pain to do and hard to maintain.
- The new recommendation is to use constraint `strong`. It automatically:
  - Computes all feasible tuples over given set of variables.
    - It uses whole model to verify the feasibility.
  - Creates table constraint from them.
  - Removes now redundant original constraints.
- Strong constraint is handled during presolve.

Strong constraint: example

```
...
home1x1 != away1x1;
game1x1 == 9*home1x1 + away1x1 - (away1x1 > home1x1);
strong([home1x1, away1x1, games1x1]);
...
```

Strong constraint: example

```
...
home1x1 != away1x1;
game1x1 == 9*home1x1 + away1x1 - (away1x1 > home1x1);
strong([home1x1, away1x1, games1x1]);
...
```

```
...
allowedAssignments([home1x1, away1x1, games1x1], [
  [1, 2, 10], [1, 3, 11], [1, 4, 12], [1, 5, 13], [1, 6, 14],
  [1, 7, 15], [1, 8, 16], [1, 9, 17], [2, 0, 18], [2, 1, 19],
  .. // 54 tuples instead of 90,000 possibilities
]);
...
```

Optimization as a service

http://www-969.ibm.com/software/analytics/docloud/

The problem

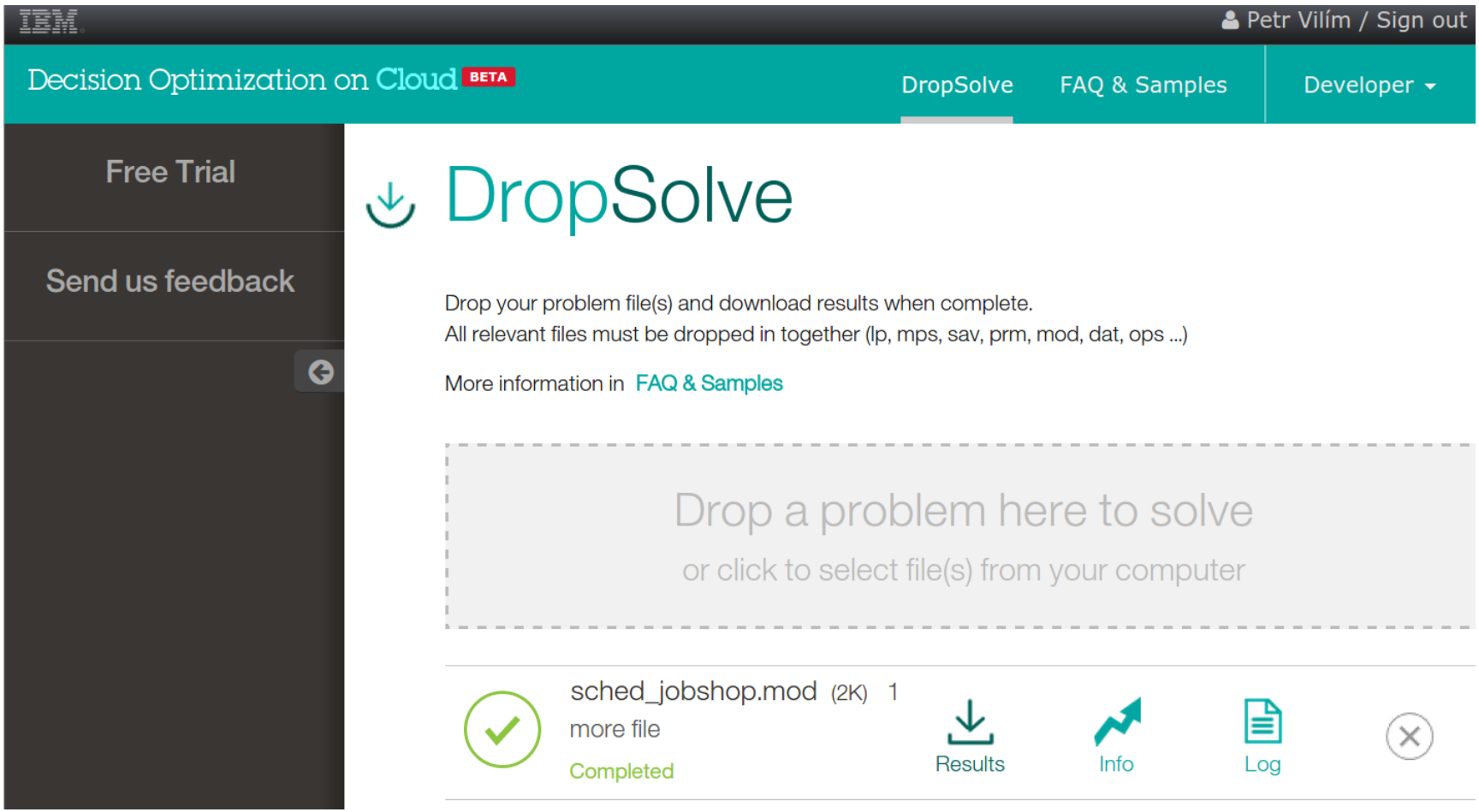
- Some customers ask for cloud
  - Powerful machines
  - Or on contrary, use optimization only occasionally
  - No need to buy and maintain HW and SW
- Great for consulting companies, proof of concepts, sizing, ..
- Free to try, attract more customers

Challenges

- Safety
- Reliability
- Split modelling and solving, network API
- Data throughput
- ...

Optimization as a service: demo

http://www-969.ibm.com/software/analytics/docloud/



cpo file format

The problem

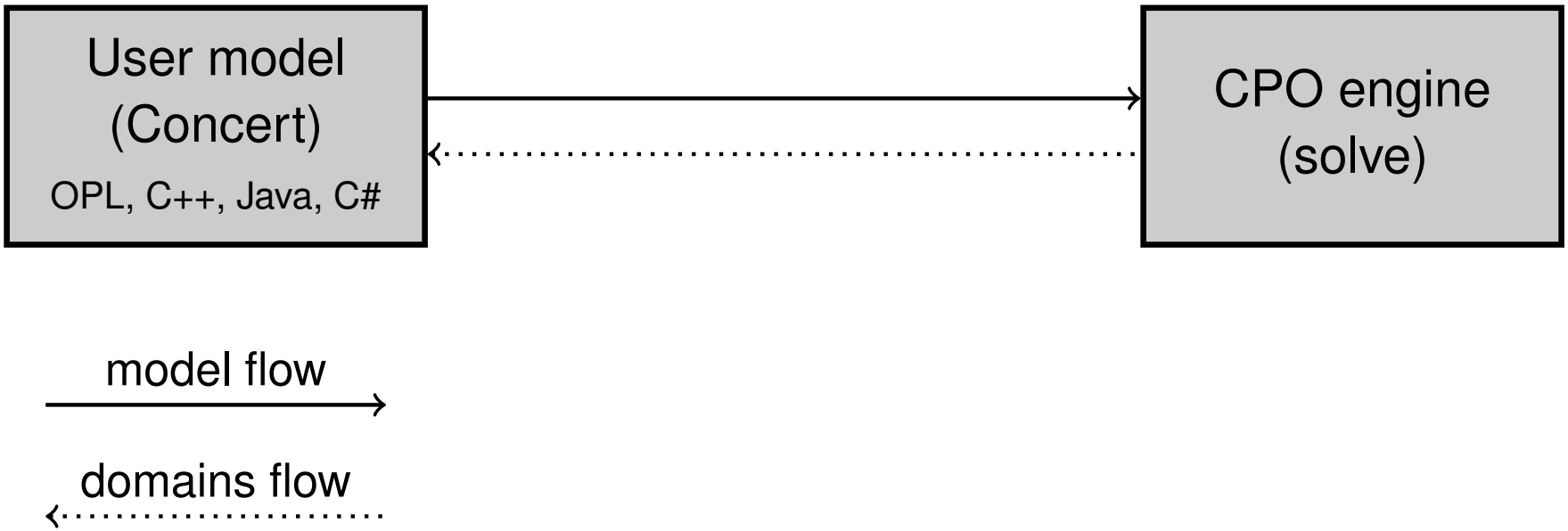
- Client makes changes in existing application and has a performance problem.
- Client asks for support.
- Application is written in C++, Java or C#, connects to databases and receives data over network.
  - Could be debugged only on the client side.
- The only solution is to send an expert to the client and go through the code of the whole application.
- Slow responds, time consuming work, ineffective, expensive, frustrating.

The problem

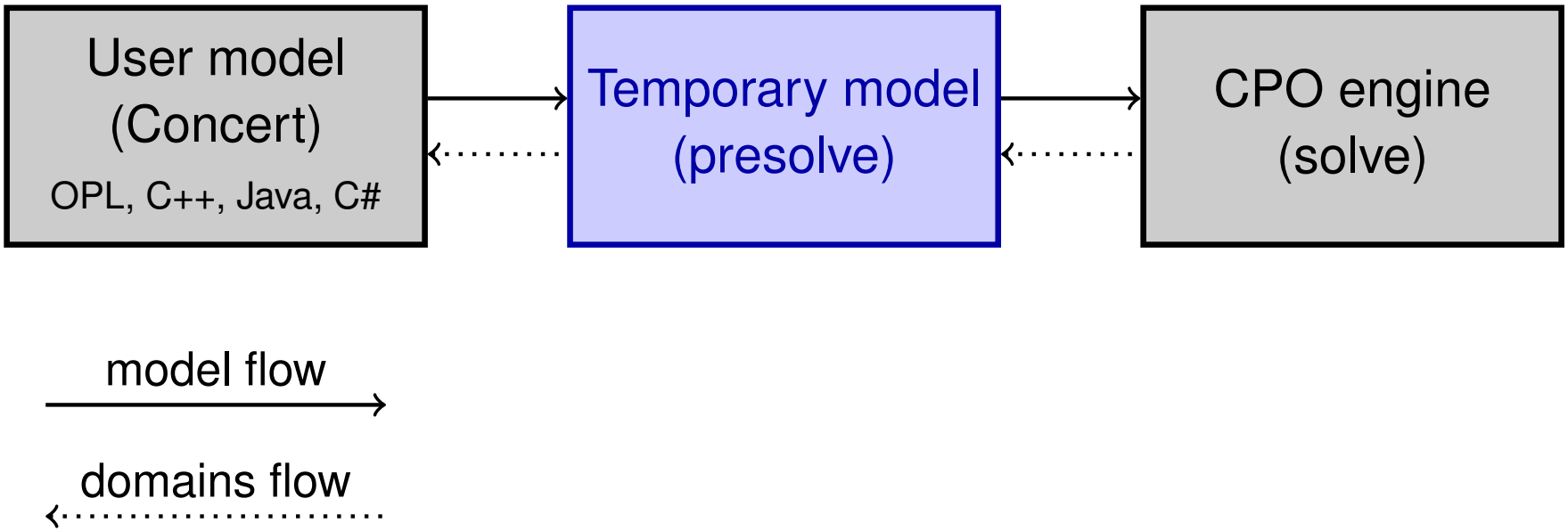
- Client makes changes in existing application and has a performance problem.
- Client asks for support.
- Application is written in C++, Java or C#, connects to databases and receives data over network.
  - Could be debugged only on the client side.
- The only solution is to send an expert to the client and go through the code of the whole application.
- Slow responds, time consuming work, ineffective, expensive, frustrating.

Need to export the model regardless what API was used to make it. And import it back.

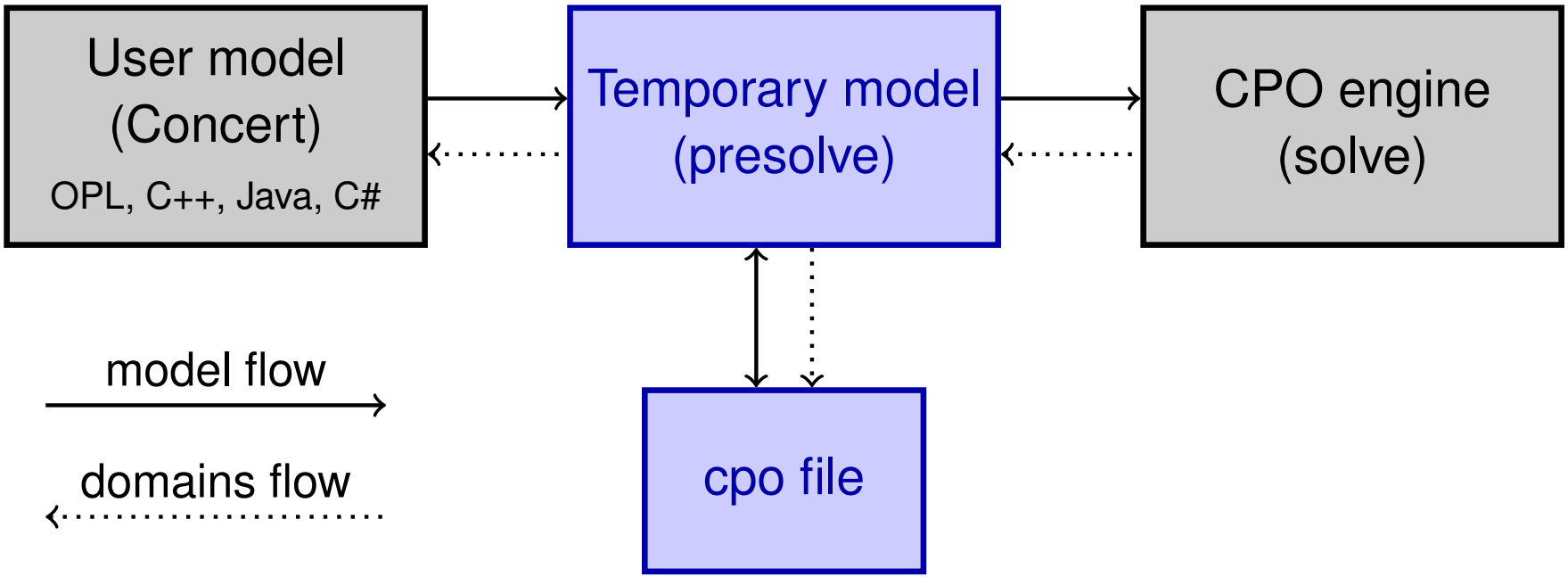
CP Optimizer scheme (old)



CP Optimizer scheme (new)



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.

- Human readable, but not expected to be written by humans.
- Flat. No cycles, forall statements etc.
- No user defined data types.
- Internal information such as version of CP Optimizer or platform used.
- Easy to parse (25MB/s on my laptop).

Benefits

- Complete serialization of the model. Possibility to transmit the model over a network.
- Debugging: user can see the actual model.
- User can send the model to IBM and get help.

Example of a cpo file

```
internals {
  version(12.6.1.0);
  architecture("x86-64_linux/static_pic", 64);
  ids(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17);
}

// Interval-related variables:
a = intervalVar(size=5, optional);
b = intervalVar(start=1..intervalmax, size=5);
c = intervalVar(start=2..intervalmax, end=0..14, size=7);
x = intervalVar(start=-100..intervalmax, end=intervalmin..intervalmax, size=1);

// Objective:
minimize(endOf(a) + endOf(b) + endOf(c) + endOf(x));

// Constraints:
noOverlap([a, b, c, x]);
startBeforeEnd(b, a, -6);
#line foo.cpp 150
startBeforeEnd(a, x, -10);

parameters {
  LogVerbosity = Quiet;
}
```

Why we did not chose flatzinc?

- Support for interval variables.
- Do not split expressions into `int_plus`, `int_times` etc.
  - Not necessary for CP Optimizer.
  - Easier to read by humans.
- Function names are the same as in C++ or OPL.
  - Simpler to understand by users.
- Possibility to include non-model information such as parameters.
- Serialization of the model including internal IDs of objects.

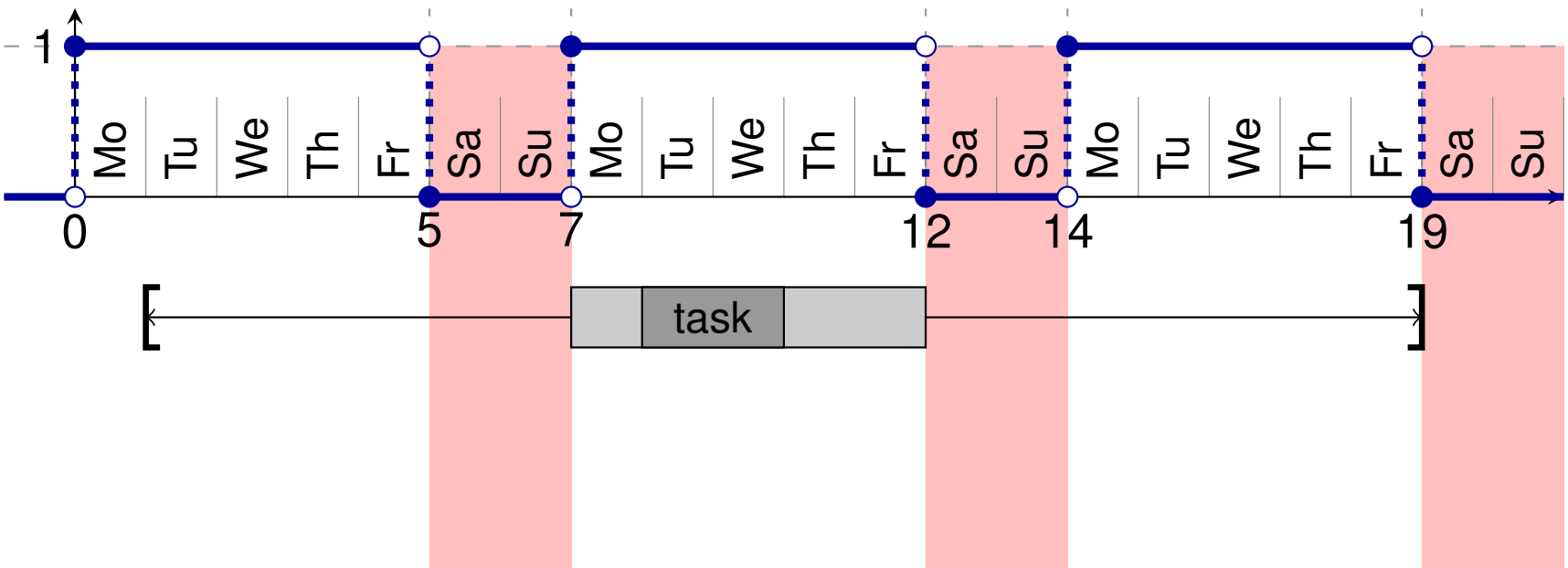
When I want to improve performance of some model:

- 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.

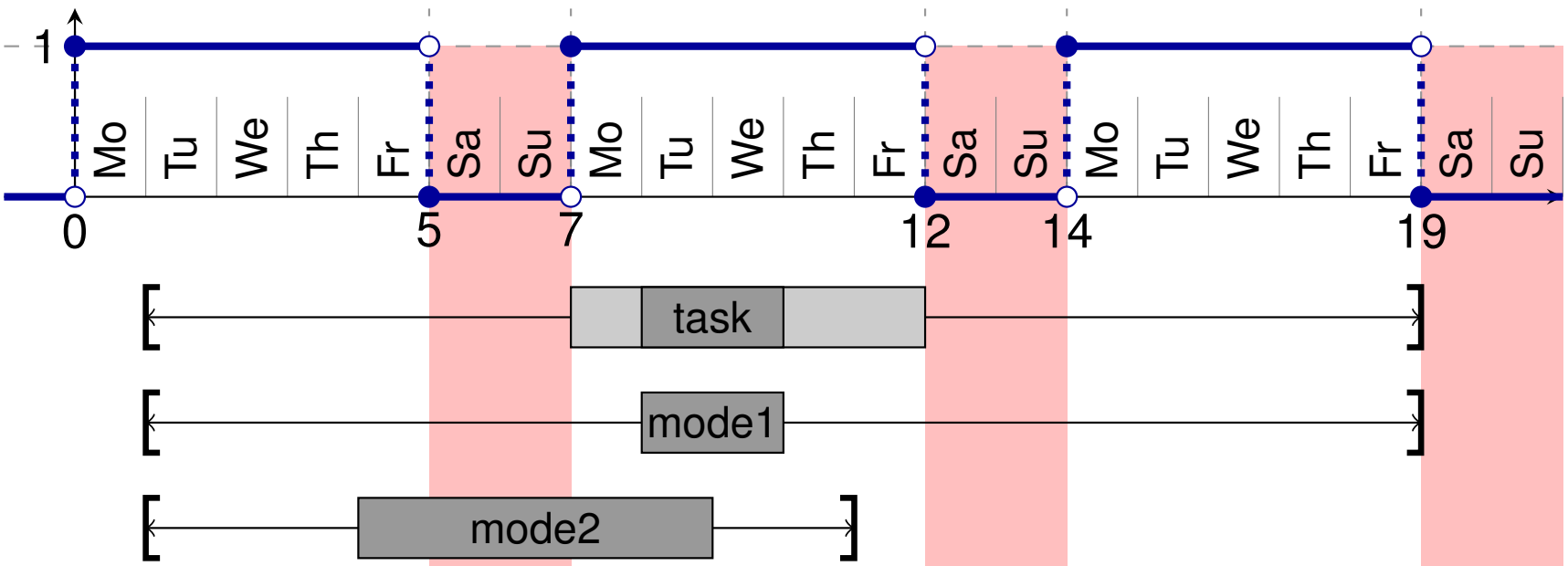
Example of a conflict

```
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)
```



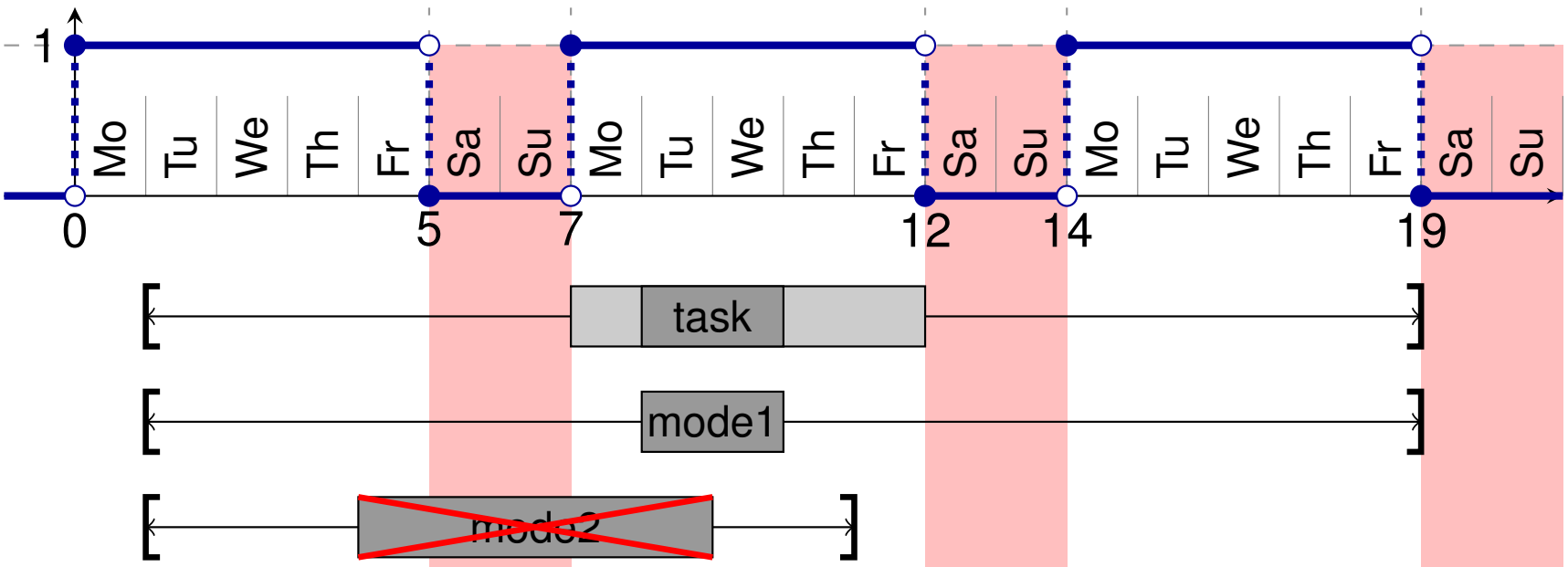
Example of a conflict

```
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
```

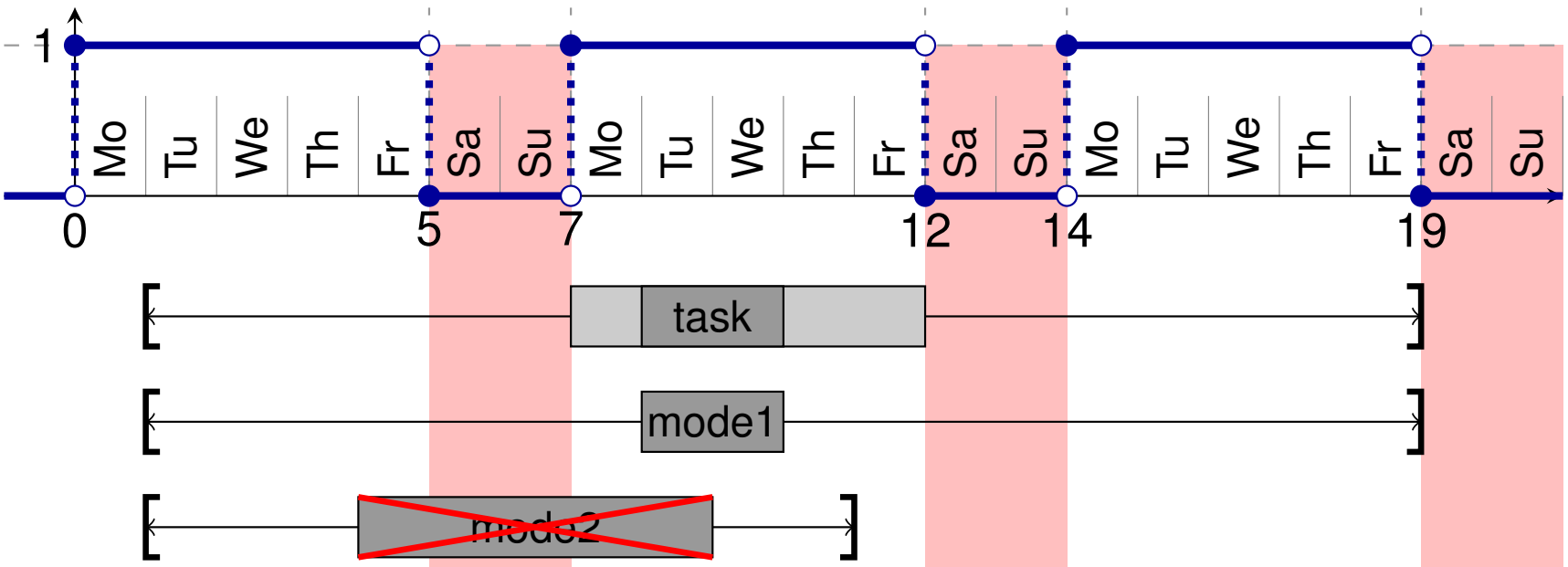


Example of a conflict

```
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
```



Explain problems

Failure explanations

```
// Build model:
...
// Create CP object:
IloCP cp(model);
// Use only one thread:
cp.setParameter(IloCP::Workers, 1);
// Simple tree search:
cp.setParameter(IloCP::SearchType, IloCP::DepthFirst);
// Show failure numbers:
cp.setParameter(IloCP::LogSearchTags, IloCP::On);
// Explain particular failures:
cp.explainFailure(IloIntArray(env, 4, 3, 10, 11, 12));
// Solve and explain:
cp.solve();
```

Failure explanation

```
 - Failure #1
 - Failure #2
 - Failure #3
-- Possible conflict explaining failure
// Model constraints
element(store1, [location1, location2, location3, location4, location5]) == 1;
element(store2, [location1, location2, location3, location4, location5]) == 1;
element(store3, [location1, location2, location3, location4, location5]) == 1;
element(store4, [location1, location2, location3, location4, location5]) == 1;
element(store5, [location1, location2, location3, location4, location5]) == 1;
element(store6, [location1, location2, location3, location4, location5]) == 1;
element(store7, [location1, location2, location3, location4, location5]) == 1;
element(store8, [location1, location2, location3, location4, location5]) == 1;
count([store1, store2, store3, store4, store5, store6, store7, store8], 0) <= 3;
count([store1, store2, store3, store4, store5, store6, store7, store8], 3) <= 4;
// Branch constraints
location2 == 0;
location3 == 0;
location5 == 0;
```

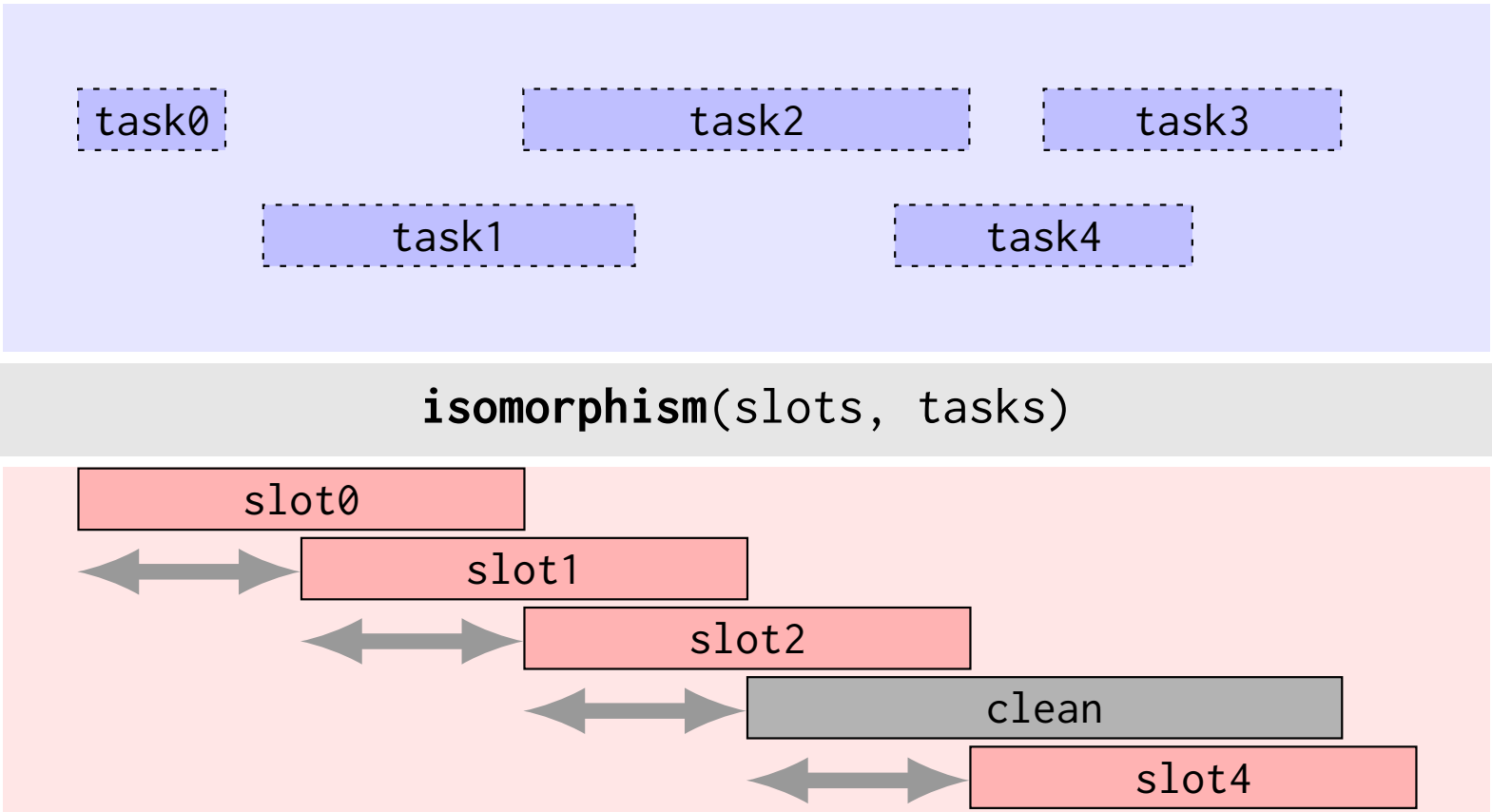
Like a compiler, CP Optimizer can print warnings

- When there is something suspicious in the model.
  - Especially for scheduling
- Regardless how the model was created (C++, OPL, ..)
- Including guilty part of the model in the cpo file format
- Including source code line numbers (if known)
- 3 levels of warnings, more than 50 types of warnings

```
foo.cpp:24: Warning: Unused interval variable 'x'.
              x = intervalVar(start=1..50, size=5..10)
foo.cpp:30: Warning: Constraint 'alternative': master interval variable 'task'
                     is optional but alternative interval 'mode1' is present.
              alternative(task, [mode1, mode2])
file.cpo:7: Warning: Constraint 'alternative': array of alternatives is empty.
                     Interval variable 'A' will be set to absent.
              alternative(A, [])
```

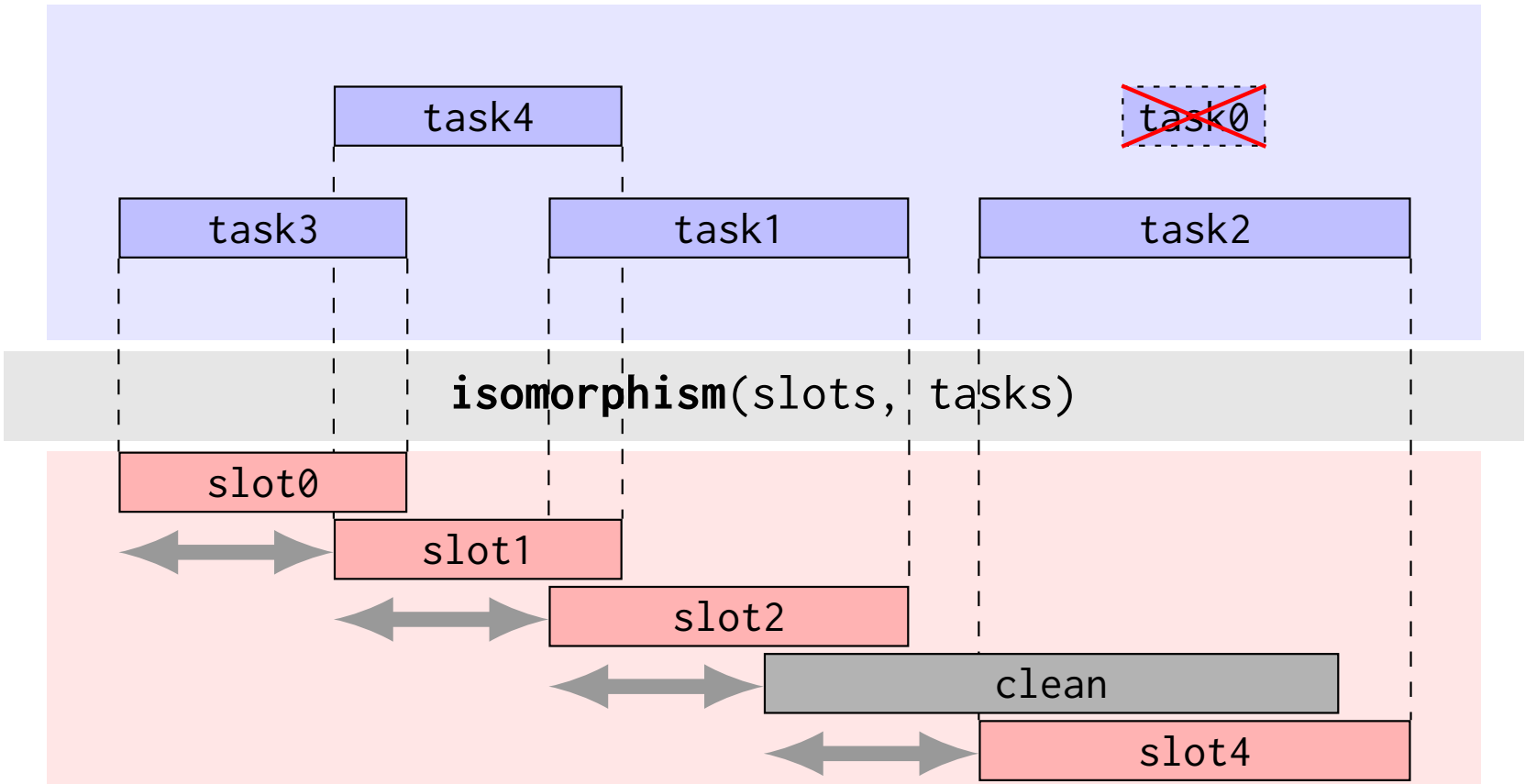
Isomorphism constraint

Isomorphism constraint



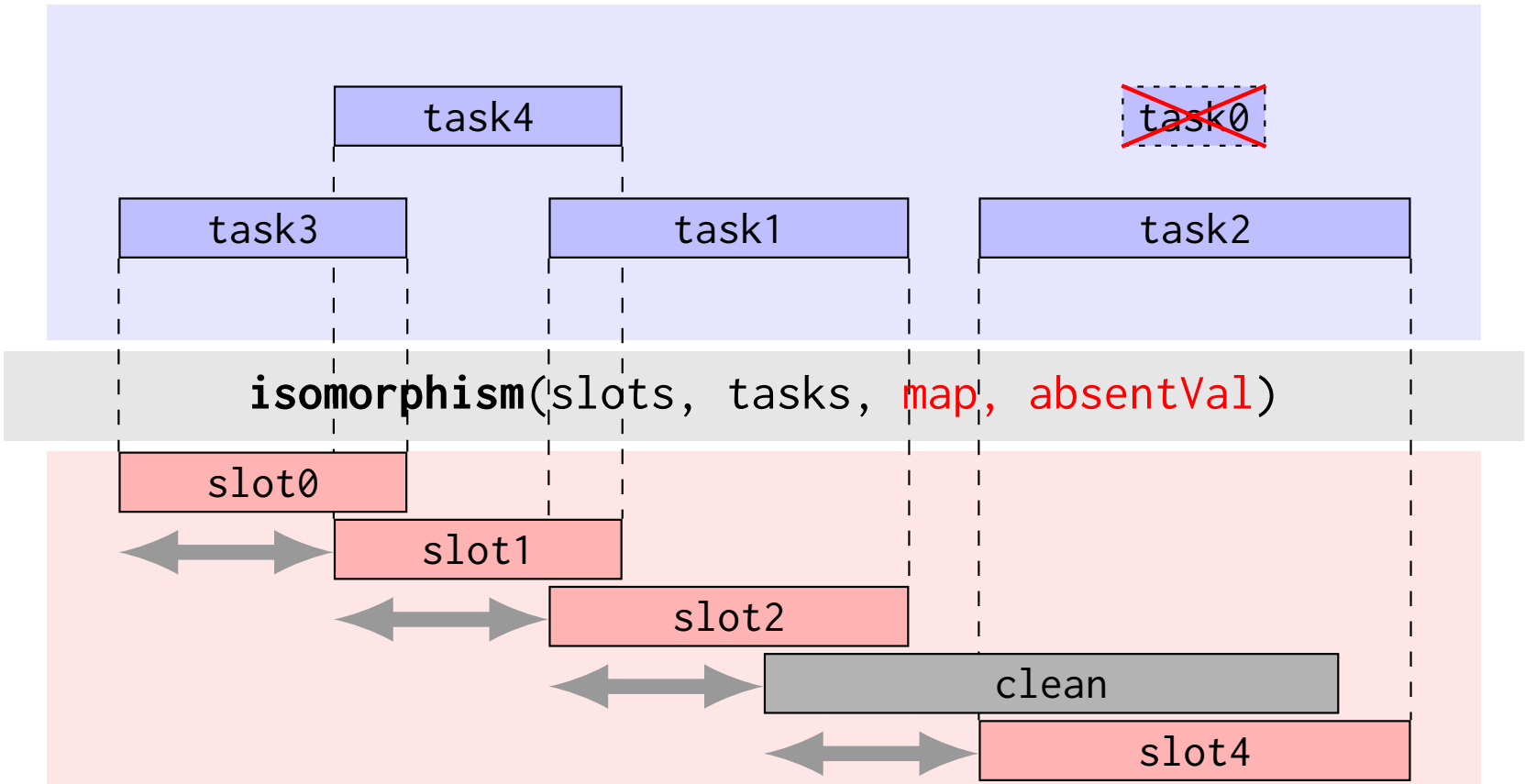
- `tasks`, `slots`: arrays of interval variables.
- Map 1:1 `tasks` on `slots`. Absent intervals are not mapped.

Isomorphism constraint



- `tasks`, `slots`: arrays of interval variables.
- Map 1:1 `tasks` on `slots`. Absent intervals are not mapped.

Isomorphism constraint



- `map`: Array of integer variables.
- `absentVal`: Value for absent variables (integer constant).
- `map[3]=0`, `map[4]=1`, `map[1]=2`, `map[2]=4`, `map[0]=absentVal`.

Isomorphism and presolve

There are many possibilities for presolve

- **noOverlap**(slots) ⇒ **noOverlap**(tasks)
- slots have common **forbidExtent** ⇒ same **forbidExtent** on tasks.
- **alldiff**(map)
- ...

Disclaimers

- The information contained in this publication is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this publication or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.
- References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.
- Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.
- IBM, the IBM logo, CPLEX and SPSS are trademarks of International Business Machines Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Questions?