

Using Constraint Programming in Business Rules Environments

Jacob Feldman, PhD
OpenRules Inc., CTO
jacobfeldman@openrules.com
www.openrules.com
www.4c.ucc.ie

“I have concluded that decision making and the techniques and technologies to support and automate it will be the next competitive battleground for organizations. Those who are using business rules, data mining, analytics and optimization today are the shock troops of this next wave of business innovation”

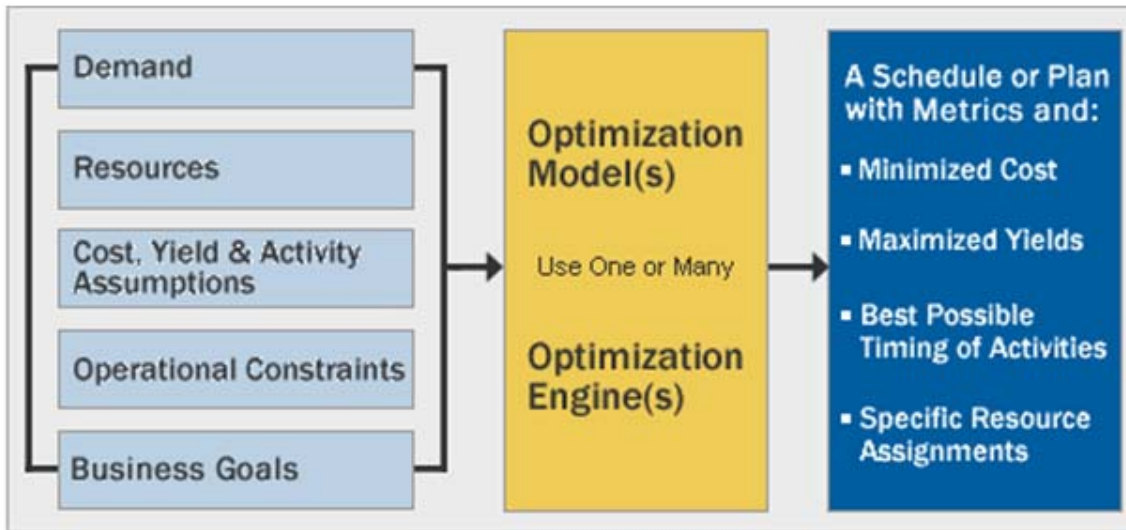
Tom Davenport

- ⌘ **Constraint Programming (CP) is a very powerful problem solving paradigm with strong roots in Operation Research and AI:**
 - ⌘ *Handbook of Constraint Programming* (Elsevier, 2006)
 - ⌘ Association for CP - <http://slash.math.unipd.it/acp/>
 - ⌘ Cork Constraint Computation Centre - <http://www.4c.ucc.ie/>
- ⌘ **This presentation concentrates on pragmatic aspects of CP:**
 - ⌘ Use of CP for real-world business application development
 - ⌘ Available tools and techniques
 - ⌘ Simple demo and practical examples
 - ⌘ Comparison with BR (pros and cons)
 - ⌘ BR+CP Integration - how different CP solvers can be incorporated into different BR products

New buzzword “Business Optimization”

- ≡ **Optimization** usually refers to a mathematical technique used to calculate the *best possible* (optimal) resource utilization to achieve a desired result such as:
 - /// minimizing expenses or travel time
 - /// maximizing ROI, service level,
 - /// other optimization objective
- ≡ **Business Optimization** helps *business* people to find optimal solutions among multiple alternatives subject to different *business* constraints.
- ≡ Optimization is at work everywhere: manufacturing, transportation, logistics, financial services, utilities, energy, telecommunications, government, defense, health care and retail

- ⚡ **Nowadays Optimization technology is quickly coming back to the business application development world as an important component of the EDM – Enterprise Decision Management**
- ⚡ **Both BR leaders ILOG and Fair Isaac put Optimization among key components of their EDM vision:**
 - ⚡ ILOG has for a long time the best tools for different optimization techniques including famous ILOG CPLEX and CP Solver
 - ⚡ Fair Isaac recently acquired Xpress-MP and incorporated it in their product offerings
- ⚡ **Many open source optimization products have achieved a competitive level and now are ready for the prime time**



- Optimization technology helps organizations make better plans and schedules
- A model captures your complex planning or scheduling problem. Then a mathematical engine applies the model to a scenario find the best possible solution
- When optimization models are embedded in applications, planners and operations managers can perform what-if analysis, and compare scenarios
- Equipped with intelligent alternatives, you make better decisions, dramatically improving operational efficiency

Constraint Programming: a bridge between academia and biz

- ⚡ **Constraint Programming (CP) is a proven optimization technology introduced to the business application development at the beginning of 1990s**
- ⚡ **During the 90s ILOG Solver became the most popular optimization tool that was widely used by commercial C++ developers. Being implemented not as a specialized language but rather as an API for the mainstream language of that time, ILOG Solver successfully built a bridge between the academic and business worlds**
- ⚡ **A few real world CP application examples**
- ⚡ **CP was especially successful dealing with real-world scheduling, resource allocation, and complex configuration problems:**
 - ⚡ CP clearly separates problem definition from problem resolution bringing declarative programming to the real-world
 - ⚡ CP made different optimization techniques handily available to regular software developers (without PhDs in Operation Research)

- ≡ **By 2000 the practical use of CP went down. Some reasons that contributed to this temporary slowdown:**
 - // Growing use of Business Rules that moved control over business logic to business people (while CP required experienced developers)
 - // Business manager reasoning: *“Let me first to externalize and arrange my business rules, then we will worry about optimization”*
 - // Java that quickly pushed C++ aside – but there were no good Java CP Solvers at that time
 - // A strong competition from a more straight-forward LP and MIP software products such as ILOG’s own CPLEX

Constraint Programming: coming back with EDM

- ⚡ The modern EDM cannot limit itself to the BR technology only. It also requires predictive analytics, CEP, constraint satisfaction, optimization, and other decision support techniques
- ⚡ Today CP is coming back as the most business friendly optimization technique. CP is rapidly becoming a ‘must-have’ capability for decision management:
 - ⚡ CP does well where BR stops short especially when :
 - ⚡ the number of alternatives goes beyond thousands and millions
 - ⚡ “closed to optimal” decisions are expected in real time
 - ⚡ a compromise between time and quality is required
 - ⚡ New powerful CP tools (both open source and commercial) with friendly business API became available
- ⚡ **Incorporating CP into Business Rules Management systems empowers a BRMS with much more sophisticated decision-support capabilities**

Both rules and constraints represent conditions which restrict our freedom of decision:

- /// The meeting must start no later than 3:30PM
- /// Glass components cannot be placed in the same bin with copper components
- /// The job requires Joe or Jim but cannot use John
- /// Mary prefers not to work on Wednesday
- /// The portfolio cannot include more than 15% of technology stocks unless it includes at least 7% of utility stocks

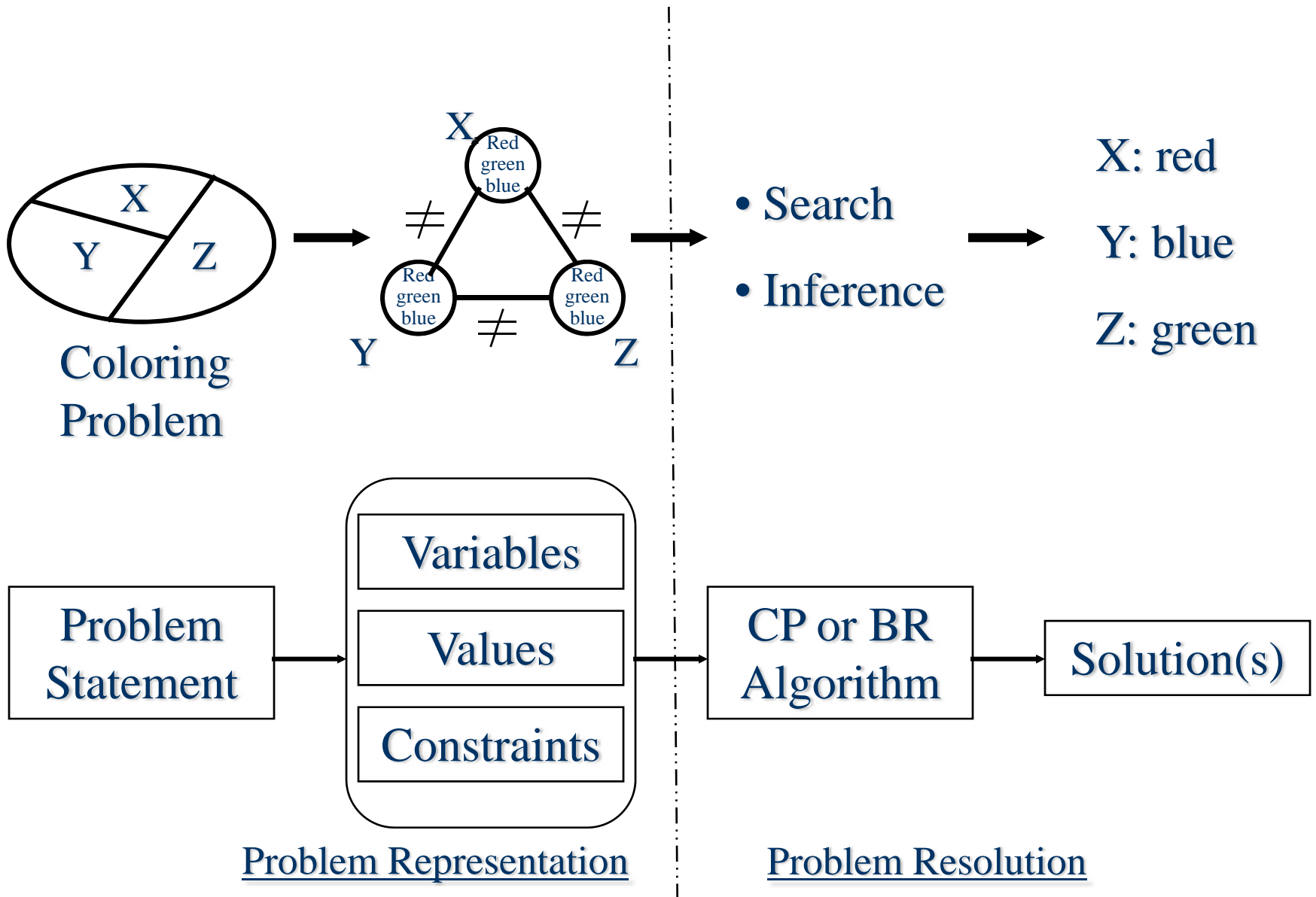
Both rules and constraints support declarative programming

- /// Concentrate on WHAT instead of HOW
- /// The same basic idea:
 - /// a user states the Rules (Constraints)
 - /// a general purpose Rule Engine (Constraint Solver) is used to solve them
- /// Note. In OpenRules we actually use the term “Rule Solver”

Problem Representation and Resolution

| BR | CP |
|--|--|
| <p><u>Business Objects</u> with some yet unknown characteristics</p> | <p><u>Decision variables</u> with yet unknown values from a finite domain</p> |
| <p><u>Rules specify relations</u> among these objects</p> | <p><u>Constraints specify relations</u> among these variables</p> |
| <p><u>Rule Engine</u> executes these rules using Rete or sequential algorithms to find unknown characteristics</p> | <p><u>Constraint Solver</u> uses different search strategies to find an assignment of values to variables that satisfies all constraints</p> |

Problem Representation and Resolution: trivial example



BR Advantage:

- Rules Repository is managed by business people while Constraint Store usually was under control of software developers

CP Advantage:

- Rules usually have to consider All (!) possible combinations of the problem parameters
- Constraints do not have to cover all situations but rather define an optimization objective and allow a search algorithm to find an optimal solution

BR+CP provides the best of both worlds:

- BR defines an optimization problem, CP solves it

How the constraint “ $X < Y$ ” works

Let’s assume X and Y are defined on the domain $[0,10]$

Initial constraint propagation after posting $X < Y$ constraint:

$X[0;9]$

$Y[1;10]$

Changes in X cause the changes in Y

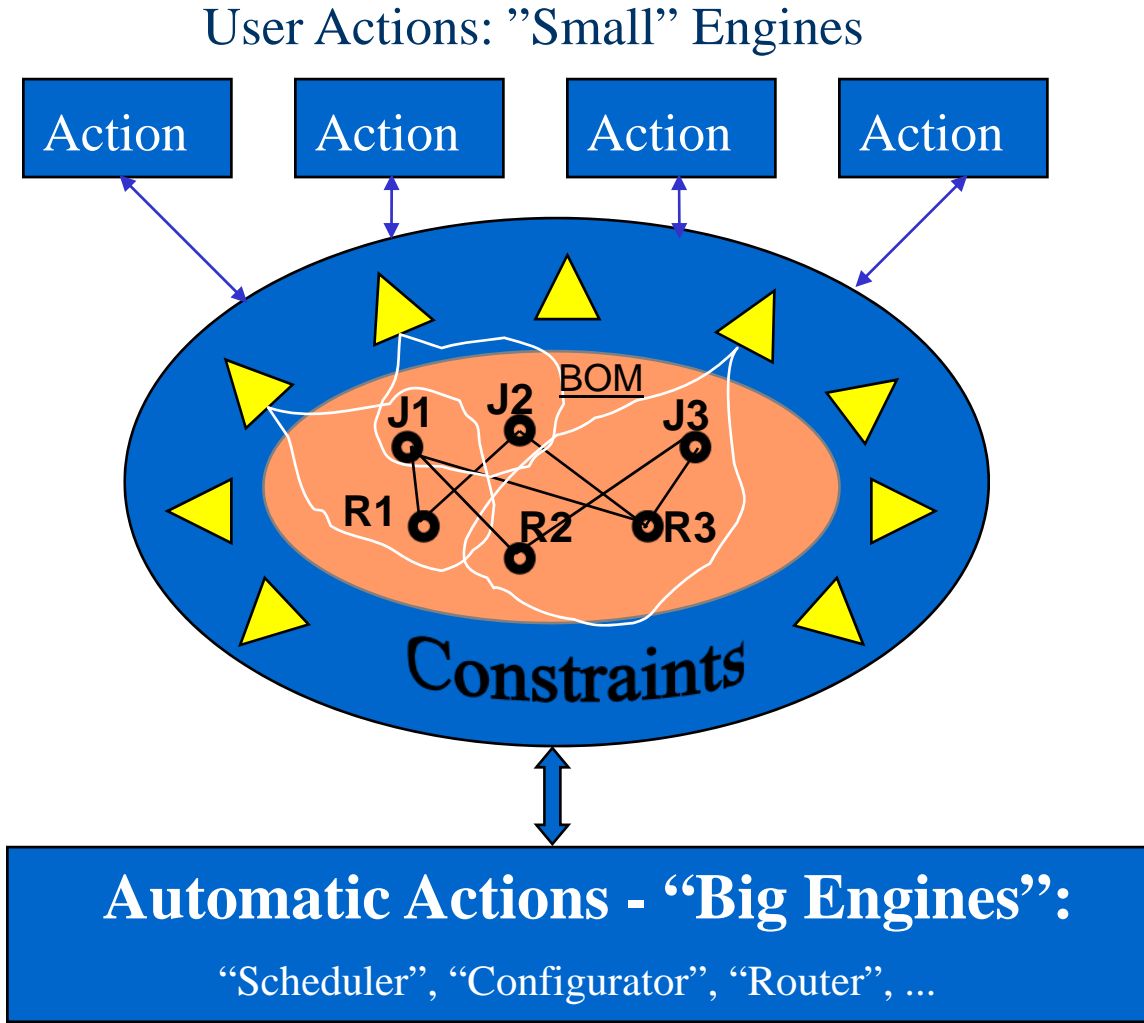
$X > 3 \Rightarrow Y > 4$

Changes in Y cause the changes in X

$Y \leq 8 \Rightarrow X \leq 7$

Bi-Directional constraint propagation

Constraint Propagation (intuitive view)

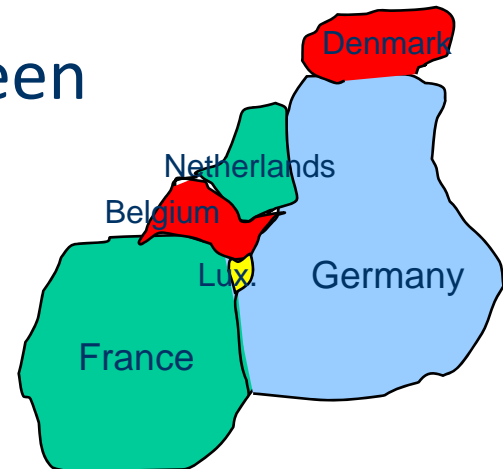


Both BR Engine and CP Solver usually provide necessary problem representation and problem resolution facilities:

| Rule Engine | CP Solver |
|---|--|
| Rules repository | Constraint Store |
| Working memory with business objects | Business objects inside a reversible environment |
| Rules execution mechanisms (predefined) | Constraint Propagation and Goal execution mechanisms (predefined or defined by a user) |

- ≡ **Business Rules always deal directly with business objects**
 - ≡ Business Rules are usually created from scratch on top of terms and facts defined in BOM
- ≡ **CP Solvers provide a rich library of predefined concepts to define and solve a related constraint satisfaction problem:**
 - ≡ **Problem Representation:**
 - ≡ Predefined Constrained Objects: Integer, Boolean, Real, and Set variables
 - ≡ Predefined Constraints:
 - ≡ Basic arithmetic and logical constraints and expressions
 - ≡ Global constraints (AllDifferent, Cardinality, ElementAt, ...)
 - ≡ **Problem Resolution:**
 - ≡ Predefined Search Algorithms (Goals)

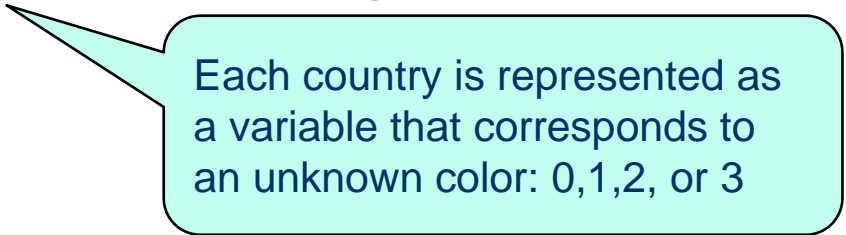
- ⌘ A map-coloring problem involves choosing colors for the countries on a map in such a way that at most 4 colors are used and no two neighboring countries have the same color
- ⌘ We will consider six countries: Belgium, Denmark, France, Germany, Netherlands, and Luxembourg
- ⌘ The colors are blue, white, red or green



Example “Map Coloring”: problem variables

```

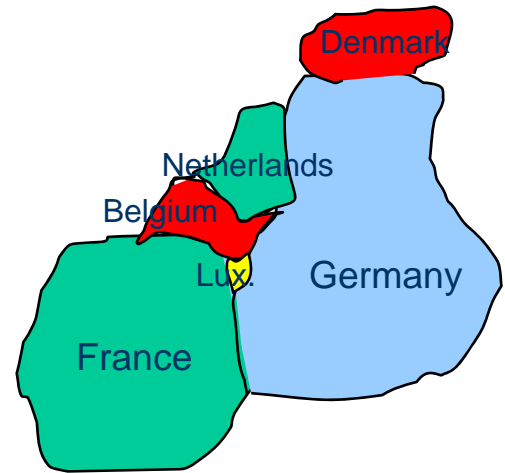
CSP p = new CSP("Map-coloring");
// Define Variables
Var Belgium      = p.addInt(0, 3, "Belgium");
Var Denmark      = p.addInt(0, 3, "Denmark");
Var France       = p.addInt(0, 3, "France");
Var Germany      = p.addInt(0, 3, "Germany");
Var Netherlands  = p.addInt(0, 3, "Netherlands");
Var Luxemburg    = p.addInt(0, 3, "Luxemburg");
    
```



Each country is represented as a variable that corresponds to an unknown color: 0,1,2, or 3

“Map Coloring”: problem constraints

```
// Define Constraints
France.ne(Belgium).post();
France.ne(Luxemburg).post();
France.ne(Germany).post();
Luxemburg.ne(Germany).post();
Luxemburg.ne(Belgium).post();
Belgium.ne(Netherlands).post();
Germany.ne(Netherlands).post();
Germany.ne(Denmark).post();
```



// We actually create a constraint and then post it
 Constraint c = Germany.ne(Denmark);
 c.post();

“Map Coloring”: solution search

// Solve

```
Goal goal = p.goalGenerate();
```

```
Solution solution = p.solve(goal);
```

```
if (solution != null) {
```

```
    for (int i = 0; i < p.getIntegers().length; i++) {
```

```
        Var var = p.getIntegers()[i];
```

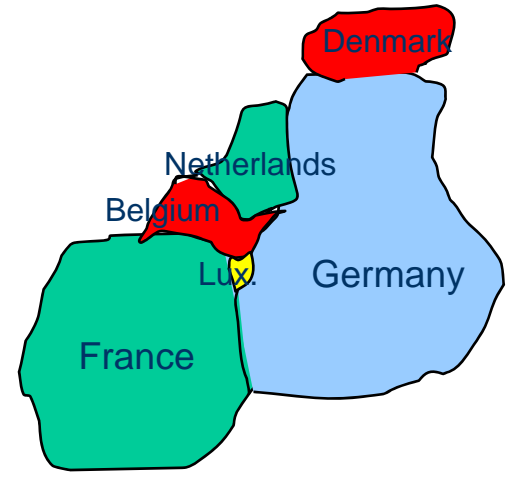
```
        p.log(var.getName() + " - " + colors[var.getValue()]);
```

```
    }
```

```
}
```

// Solution:

- Belgium – red
- Denmark – red
- France – green
- Germany – blue
- Netherlands – green
- Luxemburg - yellow



- ≡ **Predefined classes for Constrained Objects, Constraints, and Search Goals**
- ≡ **Domain representations for major constrained objects**
- ≡ **Generic reversible environment**
 - ≡ “Try/Fail/Backtrack” capabilities
 - ≡ Powerful customizable event management mechanism
 - ≡ Constraints use events to control states of all constrained objects
- ≡ **Constraint propagation mechanisms**
- ≡ **Ability to write problem-specific constraints and search goals**
- ≡ **Typical Solver Implementations:**
 - ≡ C++ , Java, Prolog, different CP Modeling Languages

CP Modeling Languages

- /// **OPL** from ILOG, France (www.ilog.com)
- /// **MiniZinc** from G12 group, Australia (<http://www.g12.cs.mu.oz.au>)
- /// **Comet**, Brown University (www.comet-online.org)
- /// **Prolog** (ECLiPSe, SICStus)

C++ API

- /// **ILOG CP** – Commercial from ILOG, France
- /// **Gecode** – Open Source (www.gecode.org)

Java API

- /// **Choco** - Open Source
- /// **ILOG JSolver** – Commercial
- /// **Constrainer** - Open Source

/// **20+ other CP Solvers:** <http://slash.math.unipd.it/cp/>

/// **CP Solvers are usually well integrated with other optimization tools (LP, MIP)**

Generic interface between different CP Solvers and Business Applications

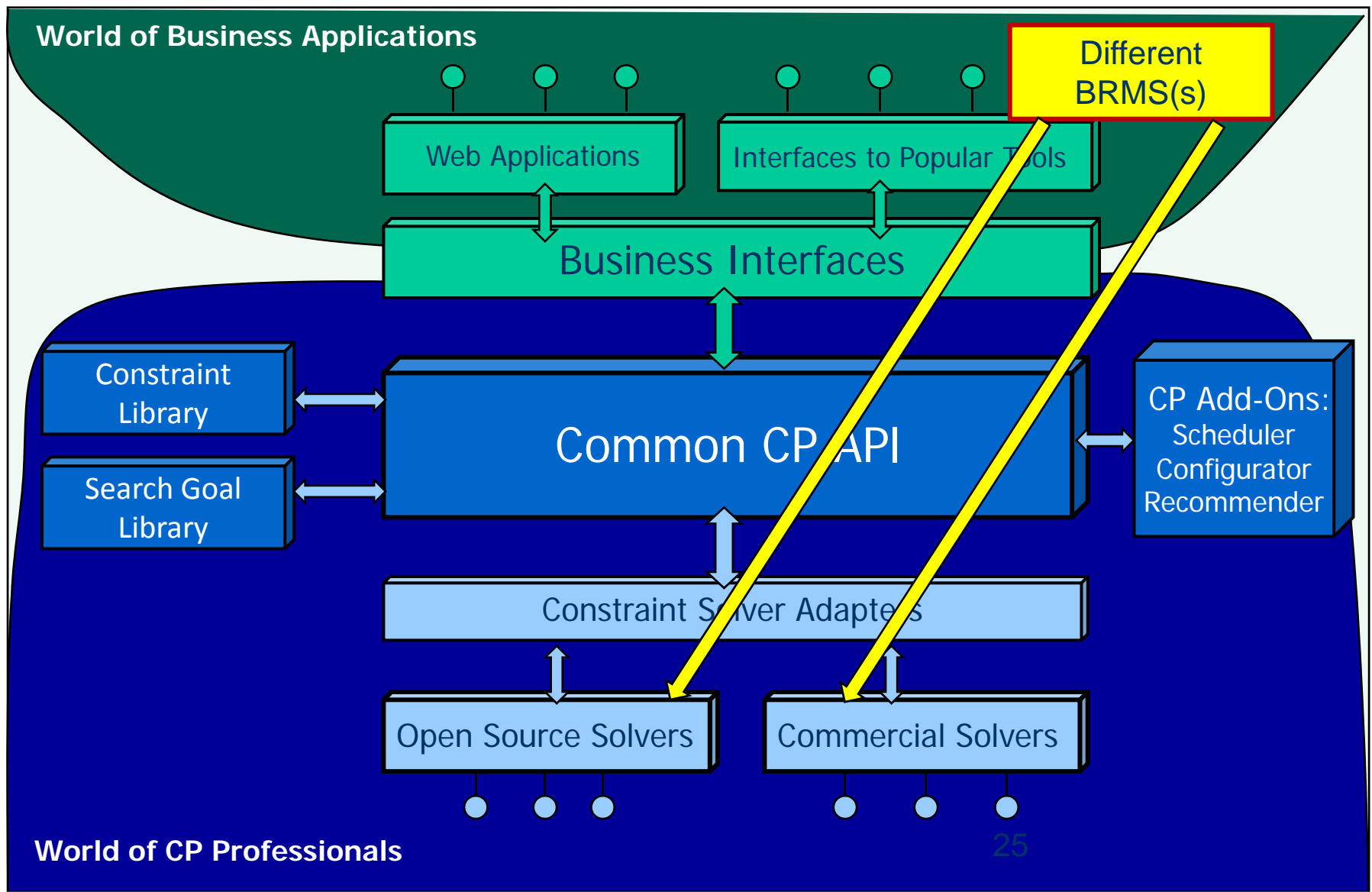
- Created by Cork Constraint Computation Centre (www.4C.ucc.ie) with support from Enterprise Ireland and Science Foundation Ireland

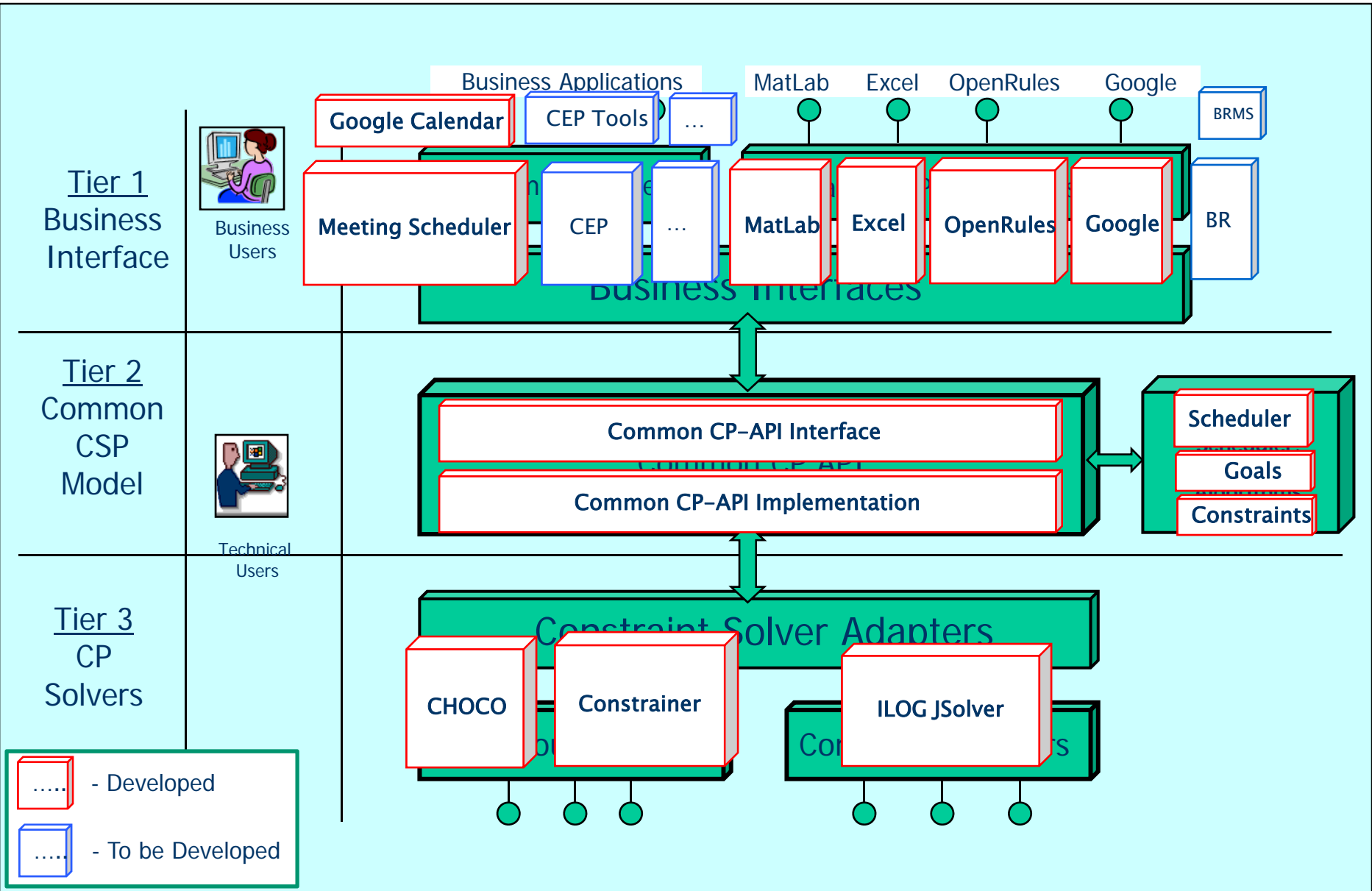
Provides a Vendor-Neutral CP API for Java

- Adapters to popular open source and commercial CP solvers
- Common library of constraints and goals
- Standardization efforts (OMG)

Can incorporate CP-based engines in popular software tools:

- MS Office (Excel), Rule Engines (OpenRules), Google Calendar and Facebook Events, MatLab, CEP tools, Lotus Notes, and more





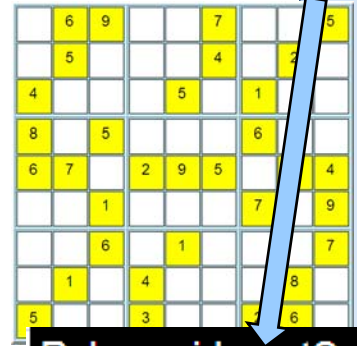
SUDOKU: integrated use of Rules and CP

| | | | | | | | | |
|---|---|---|--|--|---|--|--|---|
| | 6 | 9 | | | 7 | | | 5 |
| | 5 | | | | | | | |
| 4 | | | | | | | | |
| 8 | | | | | | | | |
| 6 | 7 | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | 1 | | | | | | | |
| 5 | | | | | | | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 9 | 8 | 2 | 7 | 3 | 4 | 5 |
| 7 | 5 | 8 | 1 | 3 | 4 | 9 | 2 | 6 |
| 4 | 3 | 2 | 9 | 5 | 6 | 1 | 7 | 8 |
| 8 | 9 | 5 | 7 | 4 | 1 | 6 | 3 | 2 |
| 6 | 7 | 3 | 2 | 9 | 5 | 8 | 1 | 4 |
| 2 | 4 | 1 | 6 | 8 | 3 | 7 | 5 | 9 |
| 3 | 2 | 6 | 5 | 1 | 8 | 4 | 9 | 7 |
| 9 | 1 | 7 | 4 | 6 | 2 | 5 | 8 | 3 |
| 5 | 8 | 4 | 3 | 7 | 9 | 2 | 6 | 1 |

Sudoku Constraints in Excel Rules Table

| 1 | 2 | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|----|---|-----------|-----|-----|-----|-----|-----|-----|-----|-----|---|---|---|
| | | Rules void postSudokuConstraints(CpProblem p) | | | | | | | | | | | | |
| | 6 | Array Name | Variables | | | | | | | | | | | |
| | 7 | row0 | x00 | x01 | x02 | x03 | x04 | x05 | x06 | x07 | x08 | | | |
| | 8 | row1 | x10 | x11 | x12 | x13 | x14 | x15 | x16 | x17 | x18 | | | |
| | 9 | row2 | x20 | x21 | x22 | x23 | x24 | x25 | x26 | x27 | x28 | | | |
| | 10 | row3 | x30 | x31 | x32 | x33 | x34 | x35 | x36 | x37 | x38 | | | |
| | 11 | row4 | x40 | x41 | x42 | x43 | x44 | x45 | x46 | x47 | x48 | | | |
| | 12 | row5 | x50 | x51 | x52 | x53 | x54 | x55 | x56 | x57 | x58 | | | |
| | 13 | row6 | x60 | x61 | x62 | x63 | x64 | x65 | x66 | x67 | x68 | | | |
| | 14 | row7 | x70 | x71 | x72 | x73 | x74 | x75 | x76 | x77 | x78 | | | |
| | 15 | row8 | x80 | x81 | x82 | x83 | x84 | x85 | x86 | x87 | x88 | | | |



Row Constraints

Rules void postSudokuConstraints(CpProblem p)

Action

```
CpVariable[] array = p.addArray(name,vars);
p.allDiff(array).post();
```

| String name | String[] vars |
|-------------|---------------|
| Array Name | Variables |

| | | | | | | | | | | | | | |
|----|---------|-----|-----|-----|-----|-----|-----|-----|-----|-----|--|--|--|
| 25 | block00 | x00 | x01 | x02 | x10 | x11 | x12 | x20 | x21 | x22 | | | |
| 26 | block01 | x03 | x04 | x05 | x13 | x14 | x15 | x23 | x24 | x25 | | | |
| 27 | block02 | x06 | x07 | x08 | x16 | x17 | x18 | x26 | x27 | x28 | | | |
| 28 | block10 | x30 | x31 | x32 | x40 | x41 | x42 | x50 | x51 | x52 | | | |
| 29 | block11 | x33 | x34 | x35 | x43 | x44 | x45 | x53 | x54 | x55 | | | |
| 30 | block12 | x36 | x37 | x38 | x46 | x47 | x48 | x56 | x57 | x58 | | | |
| 31 | block20 | x60 | x61 | x62 | x70 | x71 | x72 | x80 | x81 | x82 | | | |
| 32 | block21 | x63 | x64 | x65 | x73 | x74 | x75 | x83 | x84 | x85 | | | |
| 33 | block22 | x66 | x67 | x68 | x76 | x77 | x78 | x86 | x87 | x88 | | | |

Block Constraints

Method void createSudokuProblem(RuleSolver s)

```
CpProblem p = s.newProblem();
// Create 9X9 Square of constrained variables with values from 1 to 9
CpVariable[] vars = p.addSquare("x", 1, 9, 9);
postDataConstraints(p);
postSudokuConstraints(p);
```

Method boolean solveSudokuProblem(CpProblem p)

```
CpVariable[] x = p.getArray("x");
if (p.solve(x, new CpSelectorMinSize(x)) != null) {
    return true;
}
else {
    p.log("No Solutions");
    return false;
}
```

See more at www.openrules.com/FanLab.htm

Typical CSP structure:

1. Define Constrained Variables with all possible values
2. Define Constraints on the variables
3. Find Solution(s) that defines a value for each variable such that all constraints are satisfied

CP-Inside interface for Rules Environments

Rules void addCoreVariables(CSP p)

| Name | Min | Max | Is Objective | Array Size |
|---------|-----|-----|--------------|------------|
| X | 0 | 10 | FALSE | |
| Y | 0 | 10 | FALSE | |
| Z | 0 | 10 | FALSE | |
| Cost | 2 | 25 | TRUE | |
| CostVar | 10 | 20 | FALSE | 5 |

Rules void addExpressions(CSP p)

| Expression Name | Variable 1 | Expression Operator | Variable 2 | Value 2 |
|-----------------|------------|---------------------|------------|---------|
| XplusY | X | + | Y | |
| XplusY3 | XplusY | * | | 3 |
| z4 | Z | * | | 4 |
| CostExp | XplusY2 | * | 4 | |

Rules void addFormulas(CSP p)

| Name | Formula |
|-------------|--|
| Expression1 | <pre>{ Var x = p.getInt("X"); Var y = p.getInt("Y"); Var z = p.getInt("Z"); x.mul(3).mul(y).sub(z.mul(4)); // 3xy-4z }</pre> |

See more examples at [XYZ.xls](#)

≡ Field Service Scheduling for the Long Island Gas and Electric Utility

More than 1 million customers in Long Island, NY

More than 5000 employees

Service territory 1,230 square miles

Hundreds jobs per day

Job requires a mix of people skills, vehicles and equipment

≡ Multi-objective Work Planning and Scheduling:

Travel time minimization

Resource load levelization

Skill utilization (use the least costly skills/equipment)

Schedule jobs ASAP

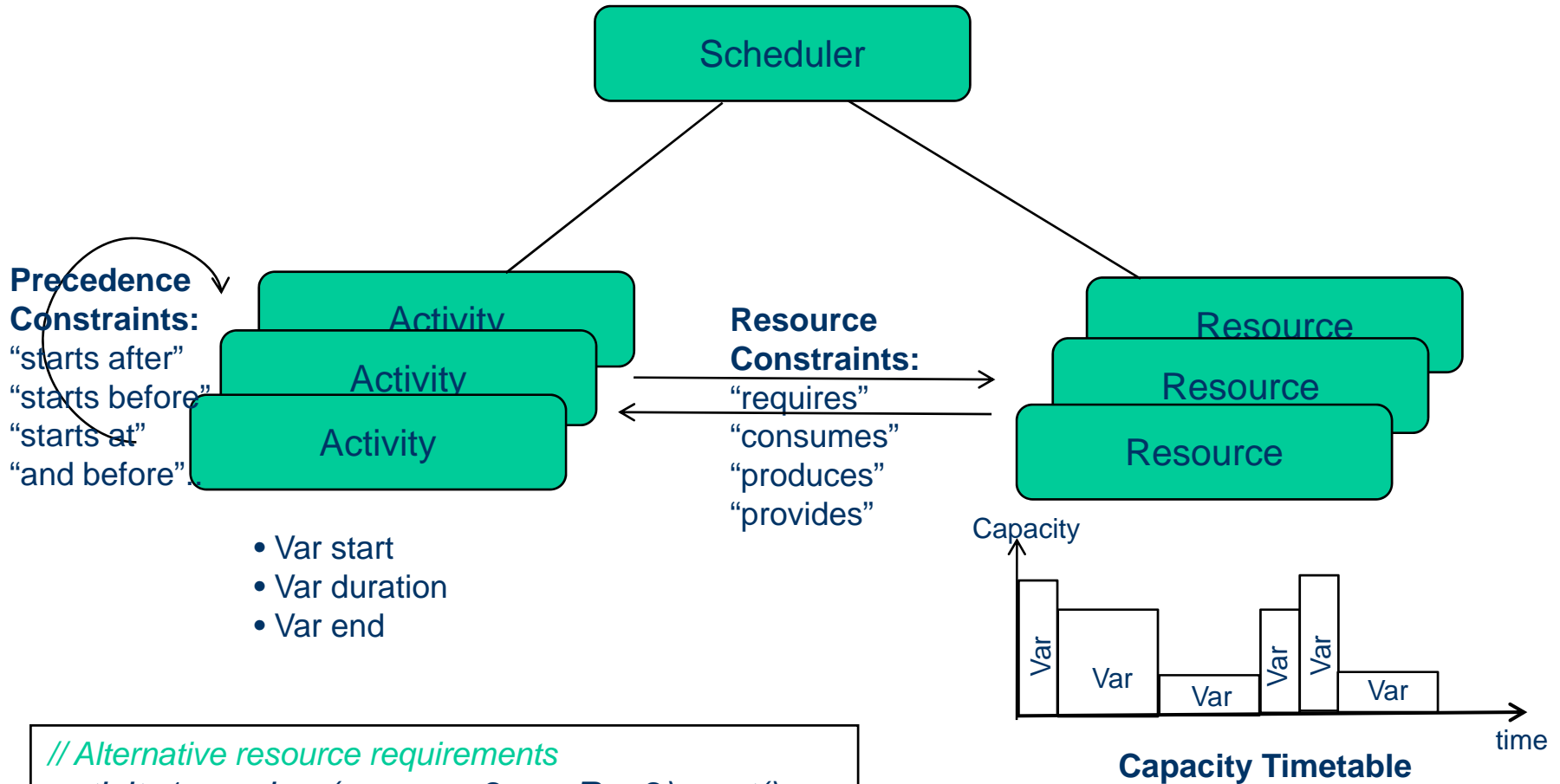
Honor user-defined preferences

⚡ Scheduling problems usually deals with:

- ⚡ Activities with yet unknown start times and known durations (not always)
- ⚡ Resources with limited capacities varying over time
- ⚡ Constraints:
 - ⚡ Between activities (e.g. Job2 starts after the end of Job1)
 - ⚡ Between activities and resources (e.g. Job1 requires a welder, where Jim and Joe both have a welder skills)

⚡ There are multiple scheduling objectives (e.g. minimize the makespan, utilize resources, etc.)

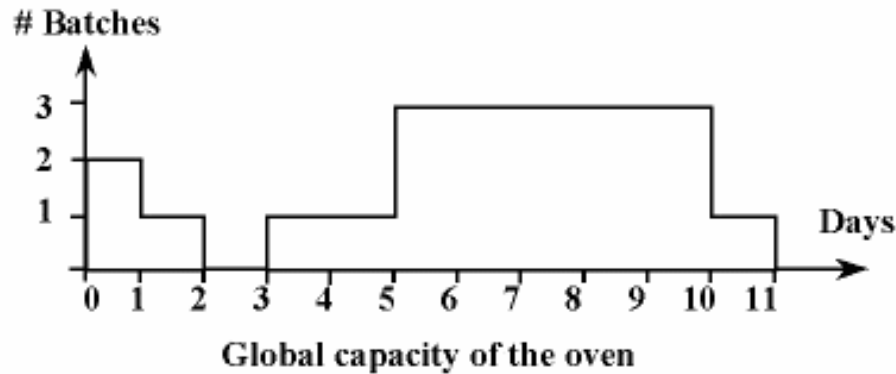
How we may create a CP-based Scheduler?



```
// Alternative resource requirements
activity1.requires(resource2, varReq2).post();
activity1.requires(resource3, varReq3).post();
varReq2.ne(varReq3).post();
```

Oven - job scheduling with one resource

There is an oven in which we can fire batches of bricks. There are five orders to fire X batches during Y days. Schedule all orders to be done in no more than 11 days taking into consideration the following oven availability:



A 2 batches, 1 day

B 1 batch, 4 days

C 1 batch, 4 days

D 1 batch, 2 days

E 2 batches, 4 days

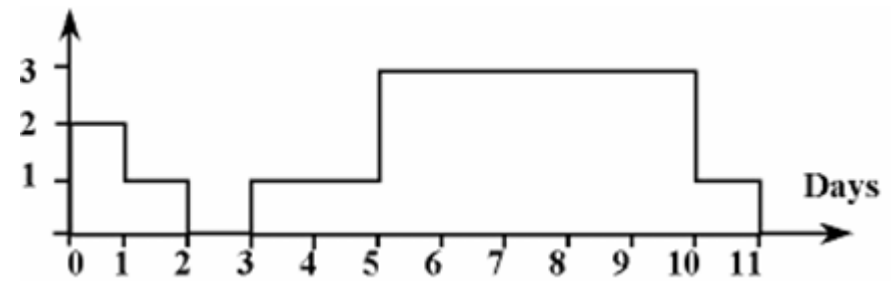
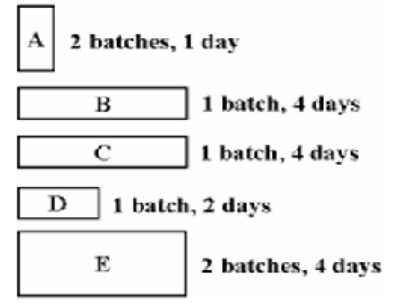
5 Activities

Scheduling Sample Implementation

```

CSP problem = new CSP("Oven Scheduling Example");
Schedule schedule = problem.addSchedule(0, 11);
Activity A = schedule.addActivity(1, "A");
Activity B = schedule.addActivity(4, "B");
Activity C = schedule.addActivity(4, "C");
Activity D = schedule.addActivity(2, "D");
Activity E = schedule.addActivity(4, "E");
Resource oven = schedule.addResource(3, "oven");
oven.setCapacityMax(0, 2);
oven.setCapacityMax(1, 1);
oven.setCapacityMax(2, 0);
oven.setCapacityMax(3, 1);
oven.setCapacityMax(4, 1);
oven.setCapacityMax(10, 1);

```



// Resource Constraints

```

A.requires(oven, 2).post();
B.requires(oven, 1).post();
C.requires(oven, 1).post();
D.requires(oven, 1).post();
E.requires(oven, 2).post();

```

// Find Solution

```

schedule.scheduleActivities();
schedule.displayActivities();

```

SOLUTION:
 A[5 -- 1 --> 6) requires oven[2]
 B[3 -- 4 --> 7) requires oven[1]
 C[7 -- 4 --> 11) requires oven[1]
 D[0 -- 2 --> 2) requires oven[1]
 E[6 -- 4 --> 10) requires oven[2]

- /// In real-world many problems are over-constrained. If this is a case, we may want to find a solution that minimizes the total constraint violation
- /// Consider a map coloring problem when there are not enough colors, e.g. only two colors:
 - /// Coloring violations may have different importance for France – Belgium and France – Germany
 - /// Find a solution that minimizes total constraint violations

“Map Coloring with Violations” implementation (1)

```
static final int MAX = 2;
```

```
// Variables
```

```
Var Belgium      = p.addInt(0, MAX - 1, "Belgium");
```

```
Var Denmark     = p.addInt(0, MAX - 1, "Denmark");
```

```
Var France      = p.addInt(0, MAX - 1, "France");
```

```
Var Germany     = p.addInt(0, MAX - 1, "Germany");
```

```
Var Netherlands = p.addInt(0, MAX - 1, "Netherlands");
```

```
Var Luxemburg   = p.addInt(0, MAX - 1, "Luxemburg");
```

```
Var[] countries = { Belgium, Denmark, France, Germany,  
                  Netherlands, Luxemburg };
```

```
// Hard Constraints
```

```
France.ne(Belgium).post();
```

```
France.ne(Germany).post();
```

```
Belgium.ne(Netherlands).post();
```

```
Germany.ne(Denmark).post();
```

```
Germany.ne(Netherlands).post();
```

```
// Soft Constraints
```

```
Var[] weights = {
```

```
    France.eq(Luxemburg).toInt().mul(257),
```

```
    Luxemburg.eq(Germany).toInt().mul(9043),
```

```
    Luxemburg.eq(Belgium).toInt().mul(568)
```

```
};
```

```
Var weightedSum = p.sum(weights);
```

“Map Coloring with Violations” Implementation (3)

```
// Optimal Solution Search
```

```
Solution solution =
```

```
    p.minimize(p.goalGenerate(countries),weightedSum);
```

```
if (solution == null)
```

```
    p.log("No solutions found");
```

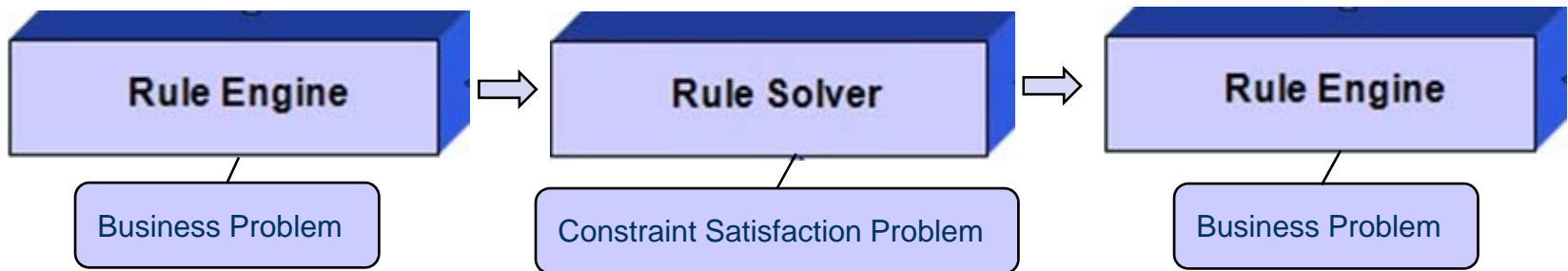
```
else
```

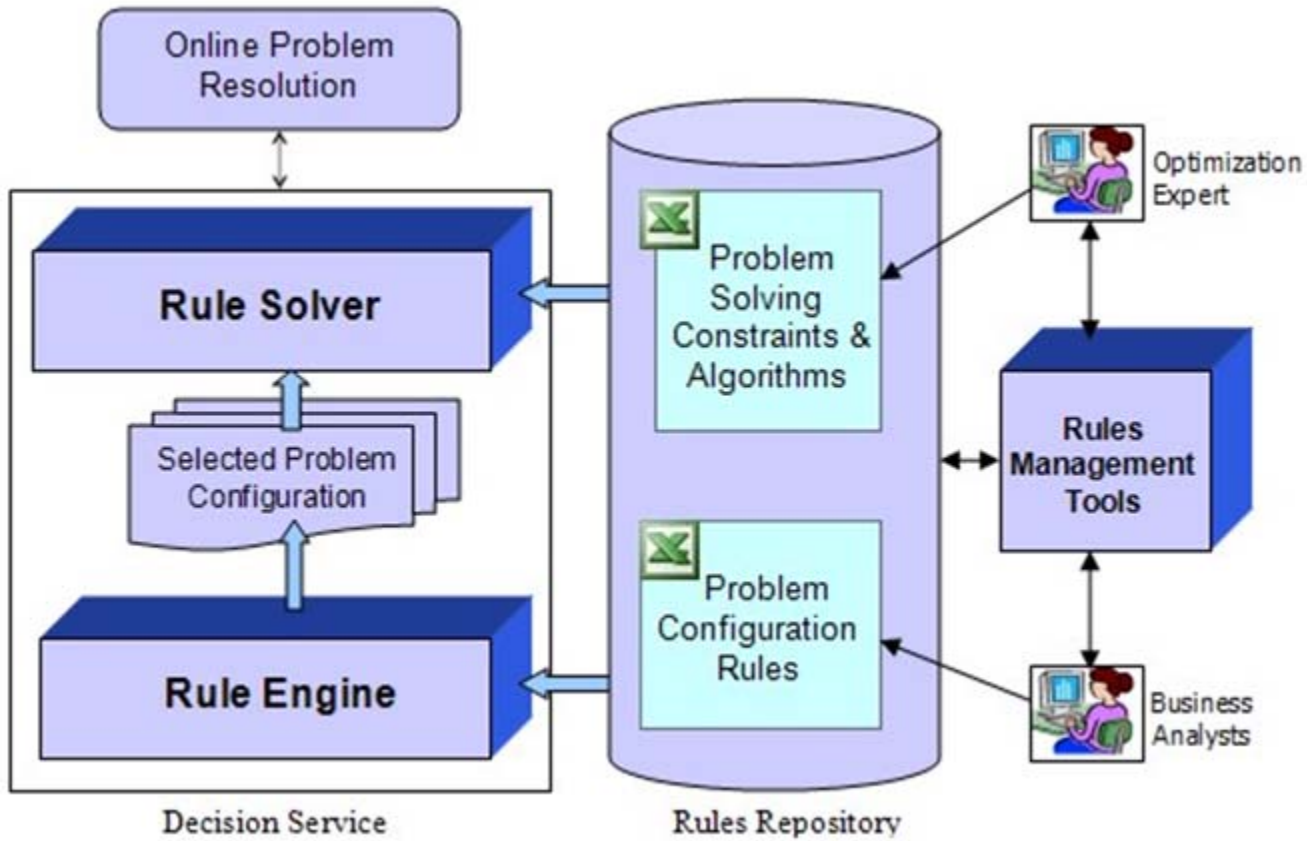
```
    solution.log();
```

Solution:

Belgium[0] Denmark[1] France[1] Germany[0] Netherlands[1] Luxemburg[1]

- ≡ **Business rules could be used to define and modify a business objects**
- ≡ **Rule Engine can generate a related constraint satisfaction problem/subproblem representing it in terms of constrained objects and constraints**
- ≡ **CP Solver can solve the optimization problems and return the results to the Rules Engine for further analysis**



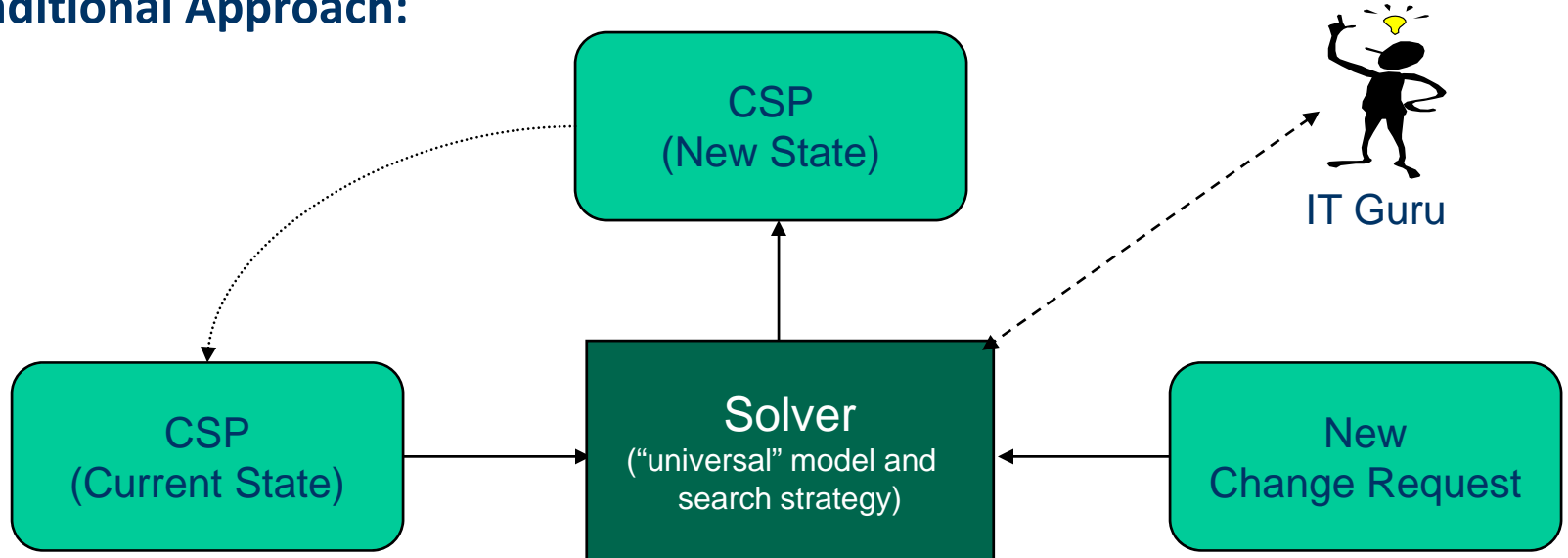


Online Decision Support: modeling and solving constraint satisfaction problems

Typical Online Systems with CP-based Solvers:

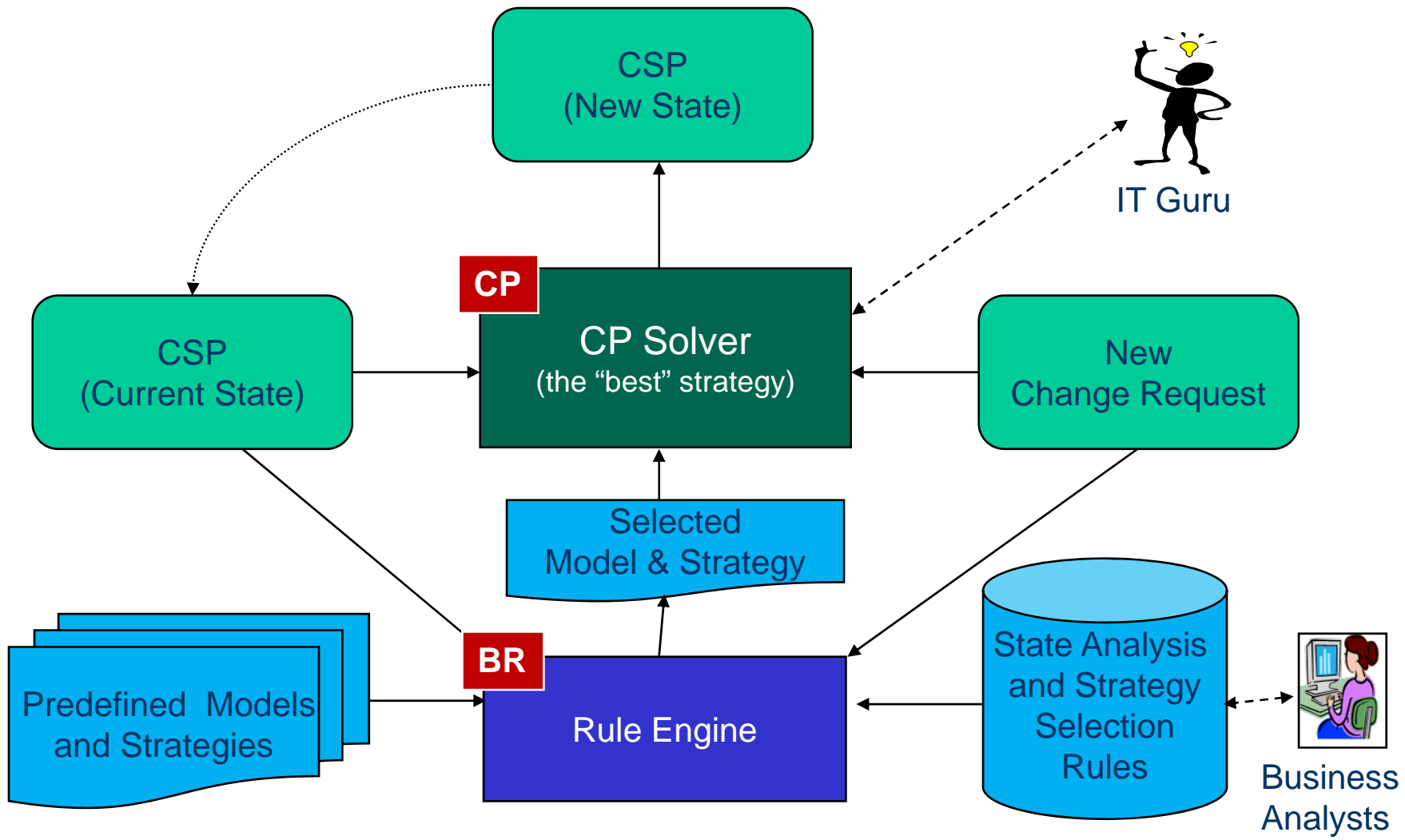
- /// Online Reservation systems (hotels, tours, vacations, ..)
- /// Event Scheduling (both business and personal events in social networks)
- /// Field Service Scheduling, Advertisement Scheduling, and more

Traditional Approach:



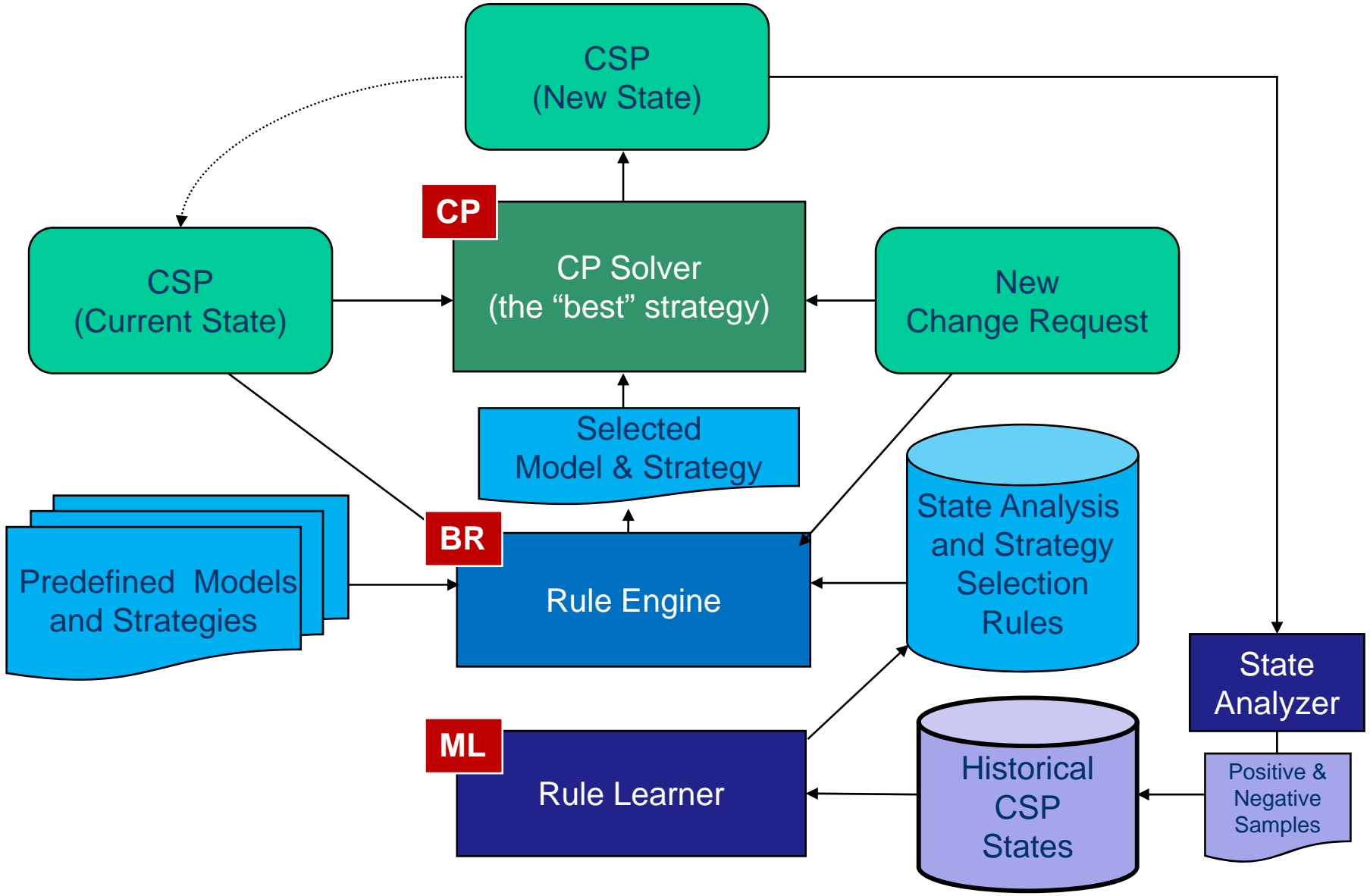
- /// “Fat” Problem Solver tuned for all possible problem states
- /// Complexity grows over time – hard to create and maintain

Online Decision Support: CP + BR adding Rule Engine to find the "best" strategy



Online Decision Support: CP + BR + ML

adding Rule Learner to find the "best" strategy



- /// **Integration of BR and CP empowers a BRMS with much more sophisticated decision-support capabilities**
- /// **BR+CP methodology and tools are available in a vendor-neutral way**
- /// **Online decision support may be done with**
 - /// CP or BR only: Hard to create and maintain “fat” Solvers controlled by IT
 - /// CP + BR: Rule Engine recommends a CSP model and search strategy based on state analysis rules controlled by business analysts
 - /// CP + BR + ML: Rule Learner discovers model/strategy selection rules based on historical Solver runs – “Ever-learning” decision support!



Thank you

Q & A