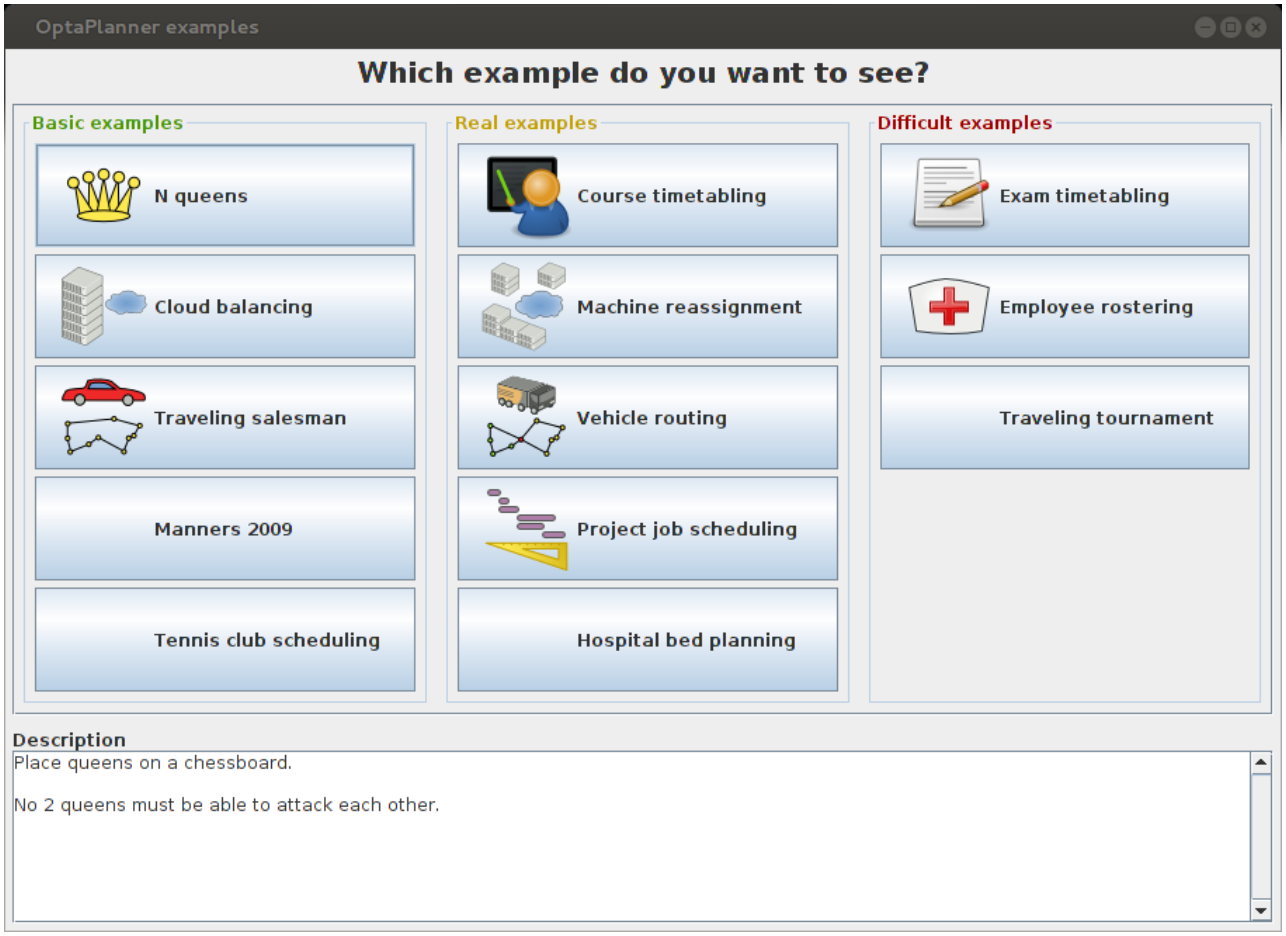
Do more business with less resources

by Geoffrey De Smet

OptaPlanner lead

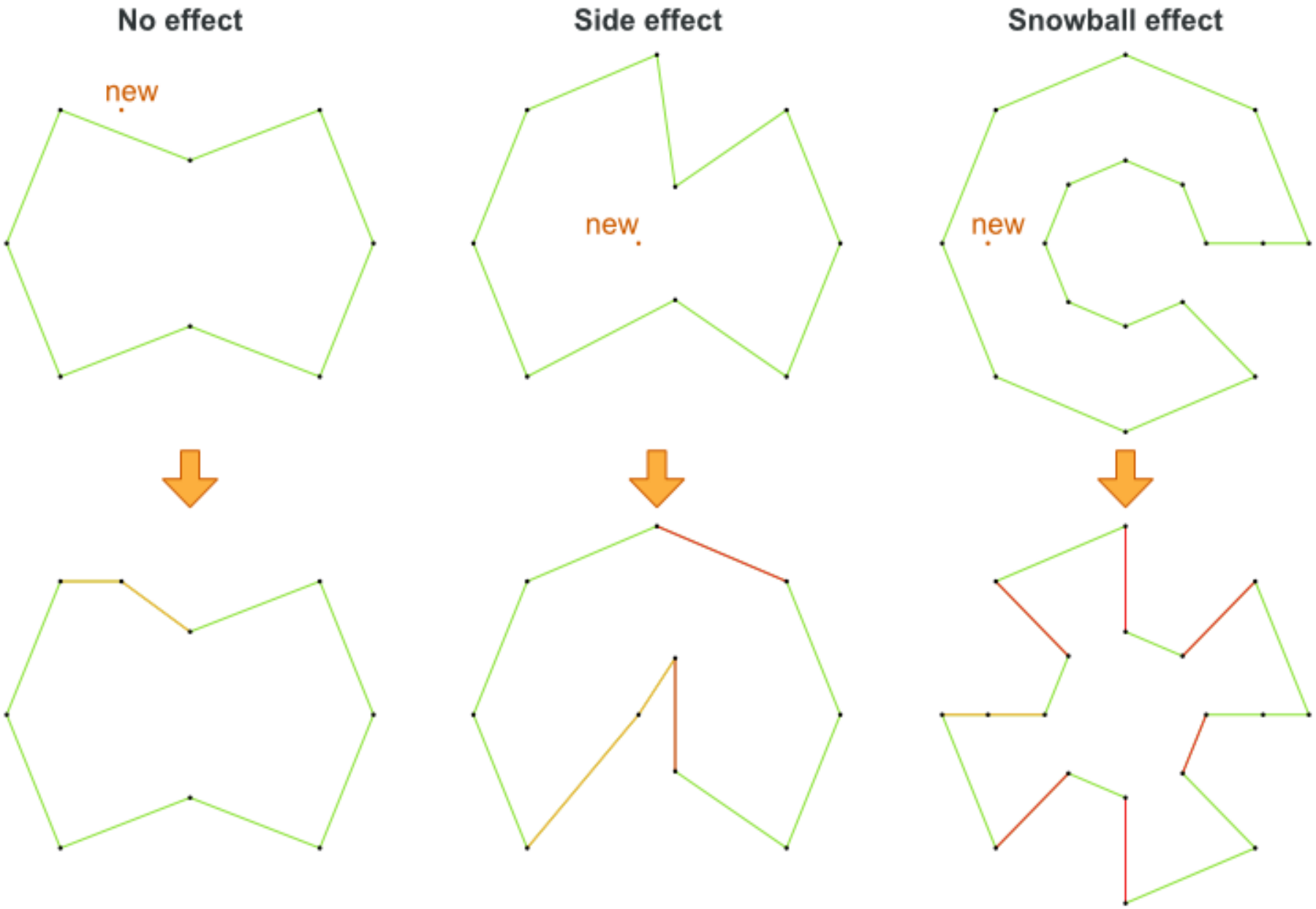
What is business resource optimization?



TSP demo

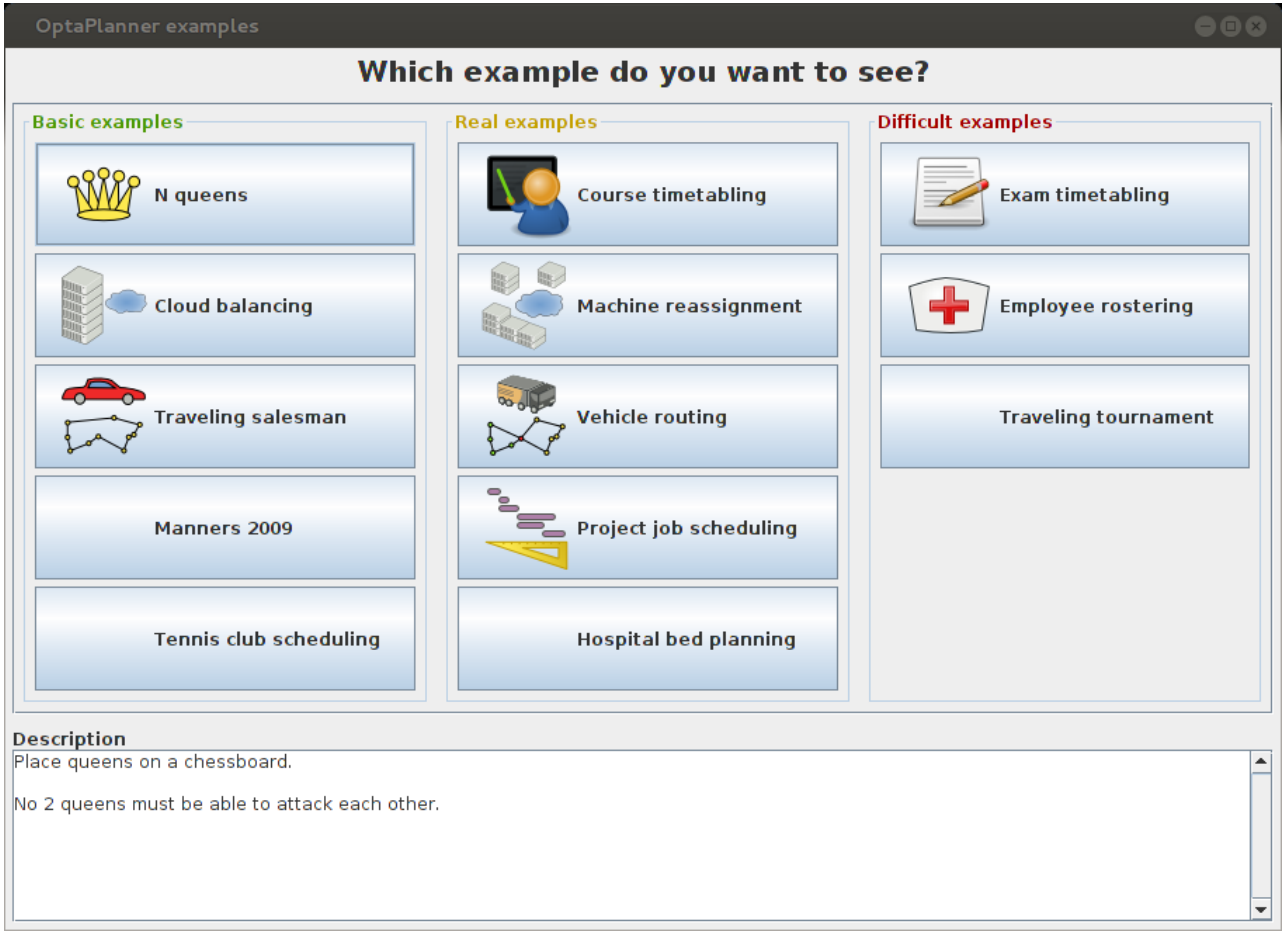
TSP optimal solution volatility

How much does the optimal solution change if we add 1 new location?



TSP
is an academic problem

What is realistic business resource optimization?



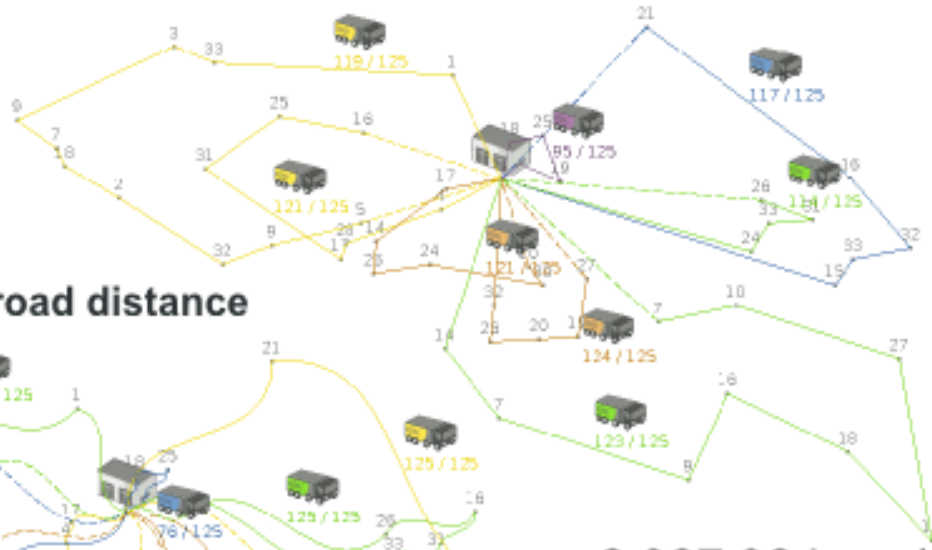
VRP demo

Vehicle routing distance type

Can we optimize for air distances, when we need road distances or driving times?

Optimized for ...

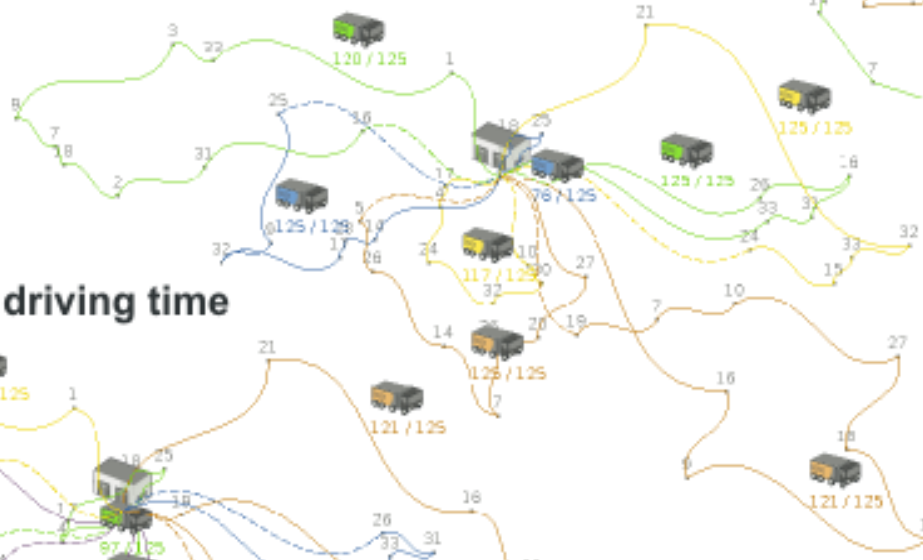
... air distance



2 327.32 km 118 632 sec

3.8% worse 4.0% worse

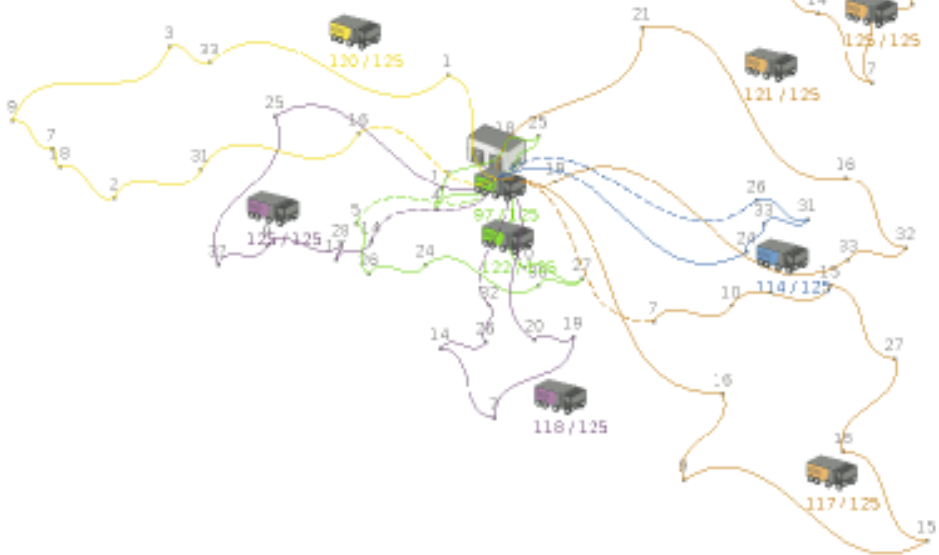
... road distance



2 243.15 km 115 516 sec

best 1.2% worse

... driving time



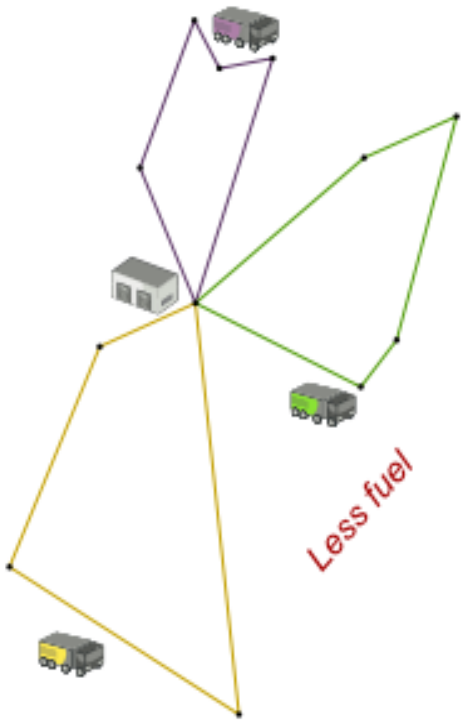
2 300.32 km 114 105 sec

2.5% worse best

What is a planning problem?

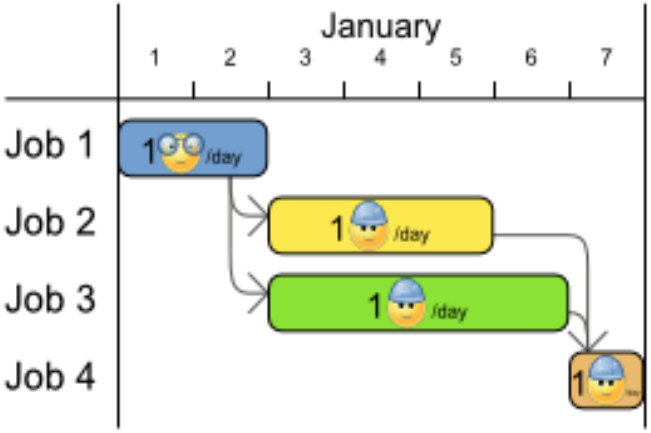
- Complete goals
- With limited resources
- Under constraints

Vehicle routing



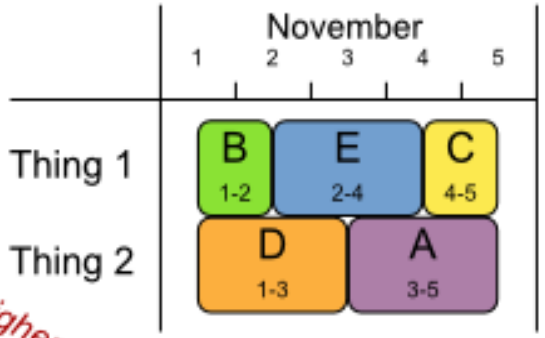
Less fuel

Job shop scheduling



Less makespan

Equipment scheduling



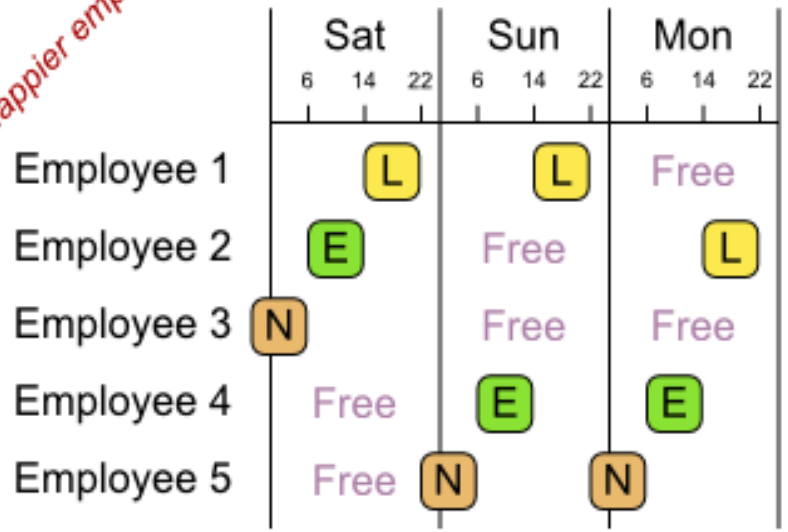
Higher utilization

OptaPlanner

Do more business with less resources

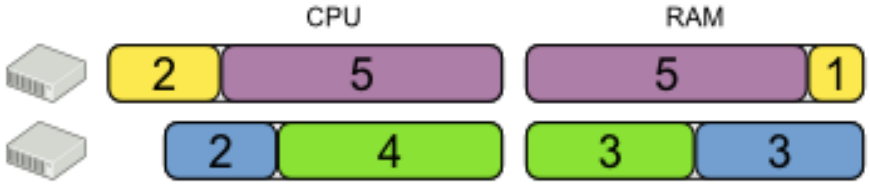
Happier employees

Employee rostering



Better allocations

Bin packing



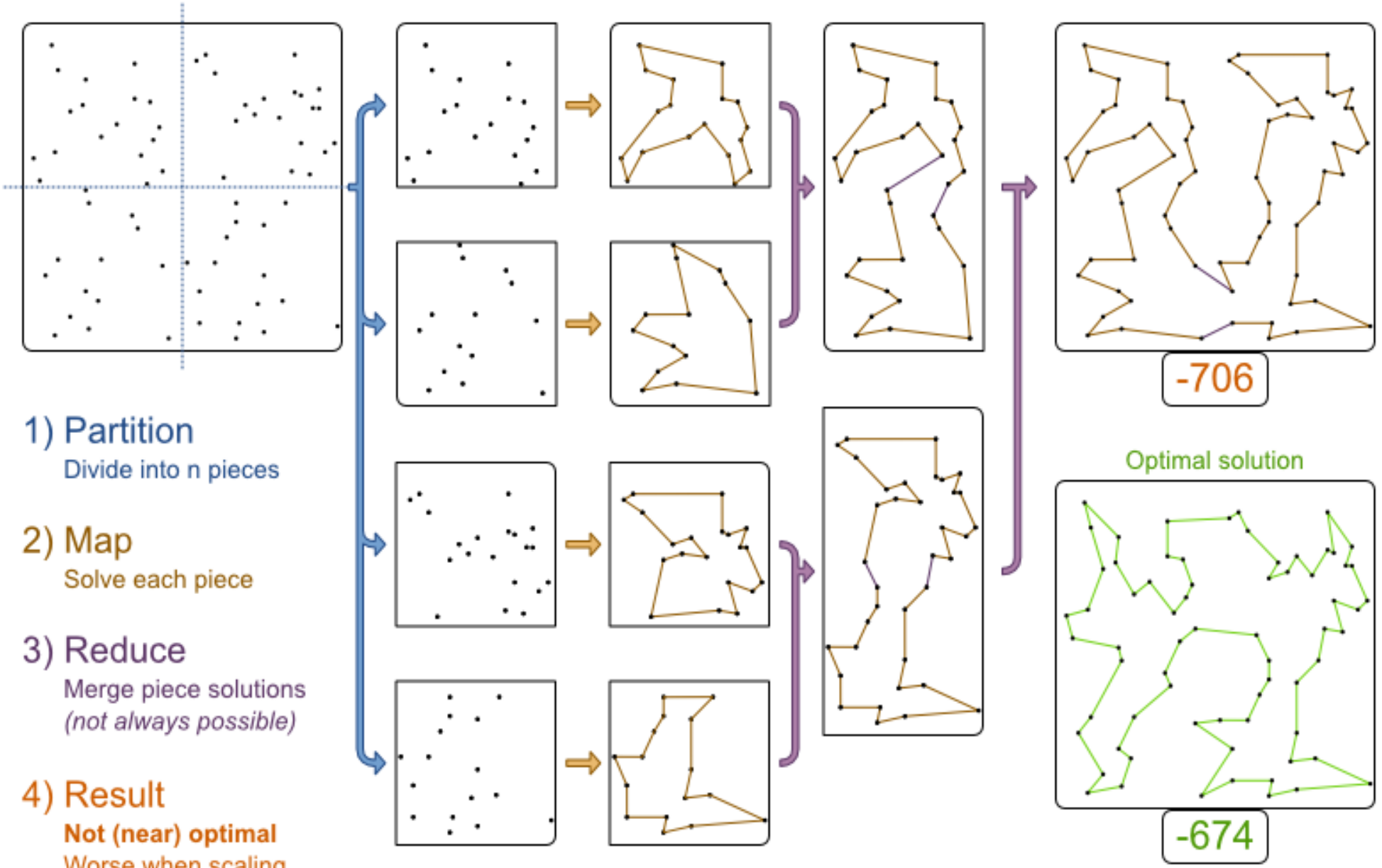
Planning problem use cases

- **Agenda scheduling**: doctor appointments, court hearings, maintenance jobs, TV advertisements, ...
- **Educational timetabling**: lectures, exams, conference presentations, ...
- **Task assignment**: affinity/skill matchmaking for tax audits, wage calc, ...
- **Employee shift rostering**: nurses, repairmen, help desk, firemen, ...
- **Vehicle routing**: route trucks, buses, trains, boats, airplanes, VIP escorts, ...
- **Bin packing**: fill containers, trucks, ships, storage warehouses, cloud computers nodes, prisons, hospitals, ...
- **Job shop scheduling**: assembly lines for cars, furniture, books, ...
- **Cutting stock**: minimize waste while cutting paper, steel, carpet, ...
- **Sport scheduling**: football/baseball league, tennis court utilization, ...
- **Financial optimization**: investment portfolio balance, risk spreading, ...

Are planning problems difficult to solve?

MapReduce is terrible for TSP

Why do MapReduce, Divide&Conquer and partitioning perform badly on NP-hard problems?



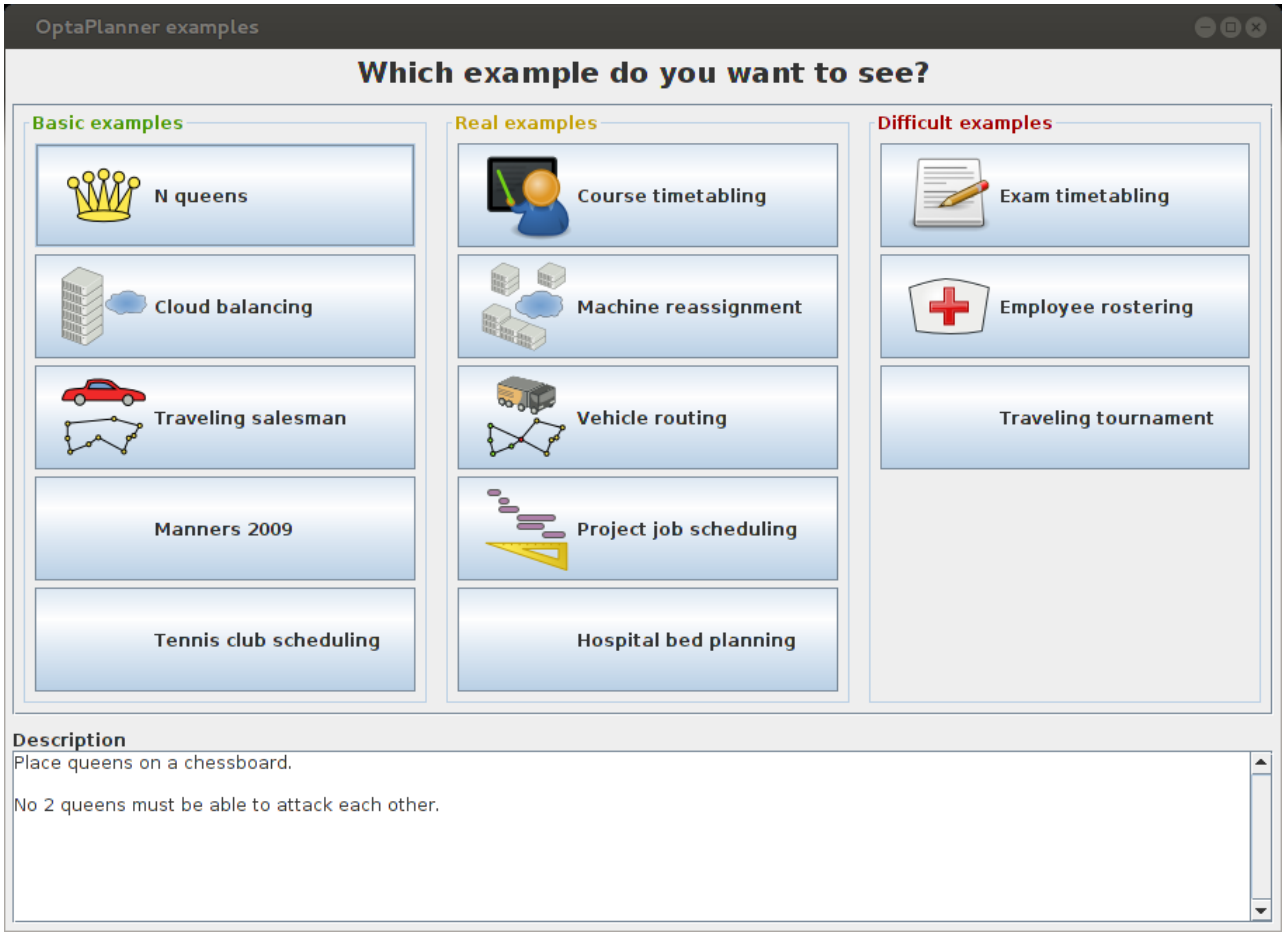
1) Partition
Divide into n pieces

2) Map
Solve each piece

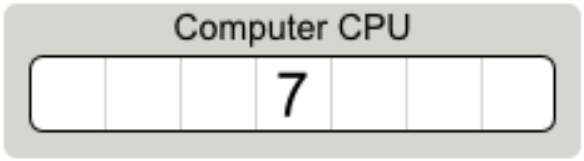
3) Reduce
Merge piece solutions
(not always possible)

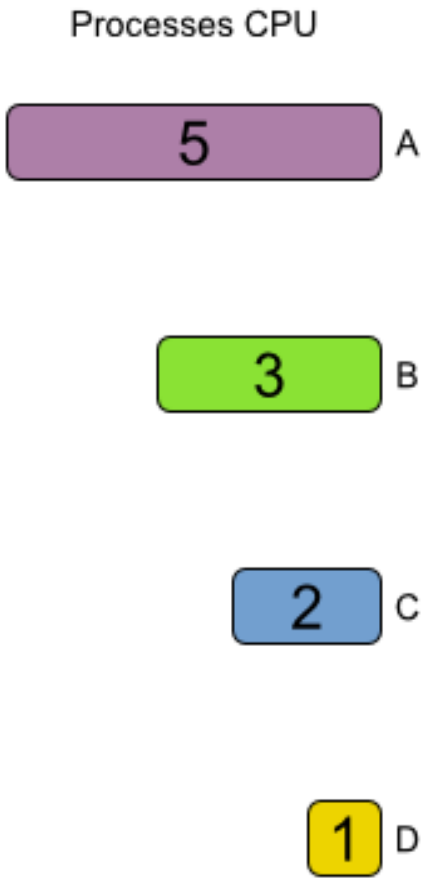
4) Result
Not (near) optimal
Worse when scaling

What is business resource optimization?

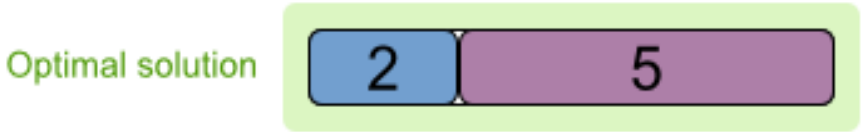


Cloud Balancing demo



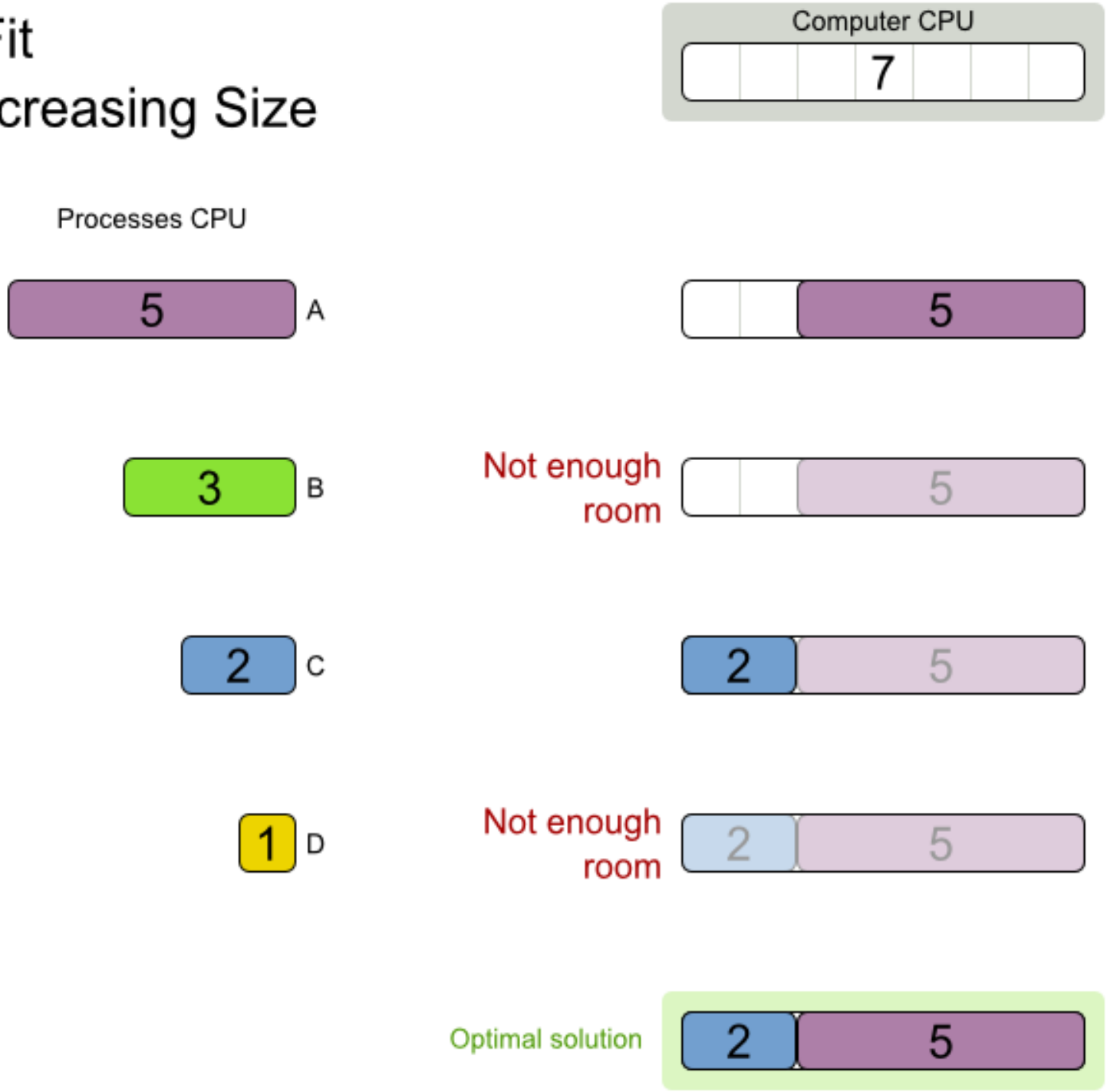


Which processes fill up this computer as much as possible?

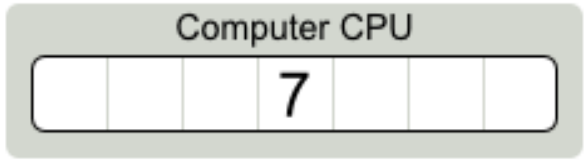


How did we find this solution?

First Fit
by Decreasing Size



First Fit Decreasing again...



Processes CPU

Not enough room

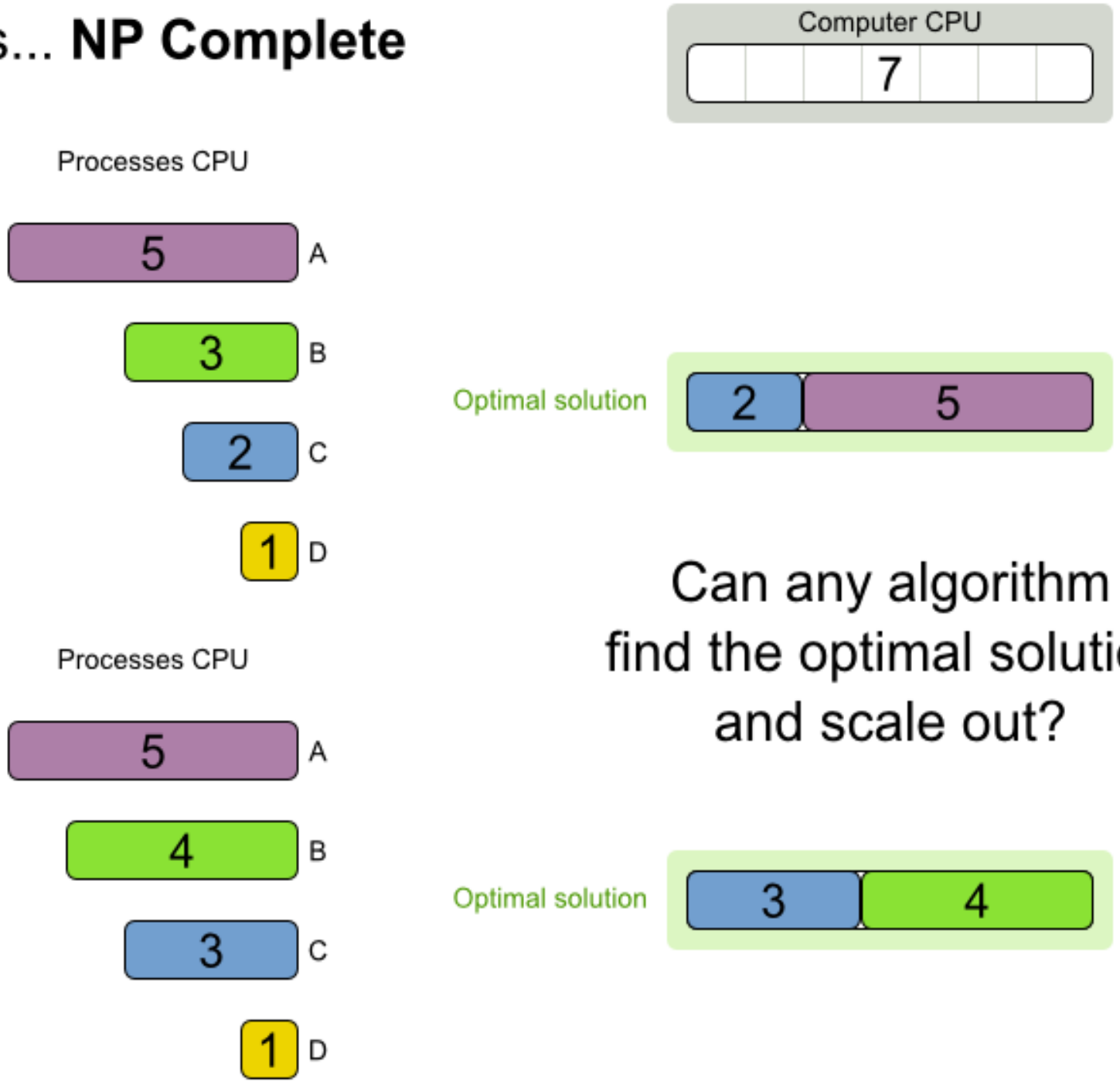
Not enough room

Not optimal!

FAIL

Optimal solution

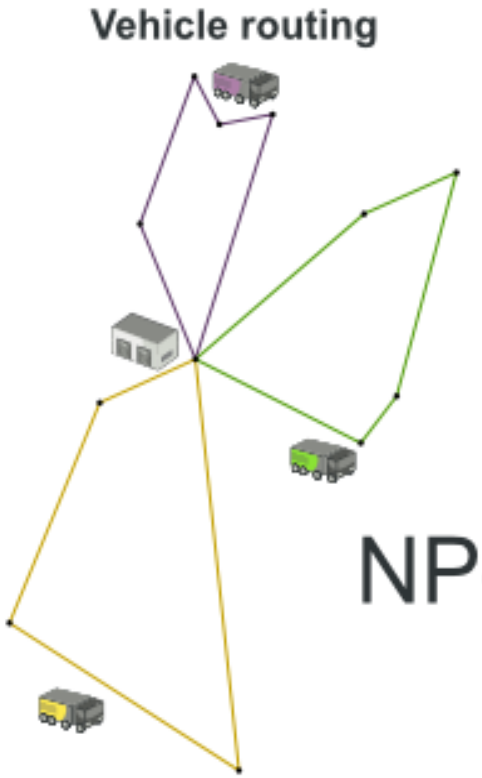
This is... **NP Complete**

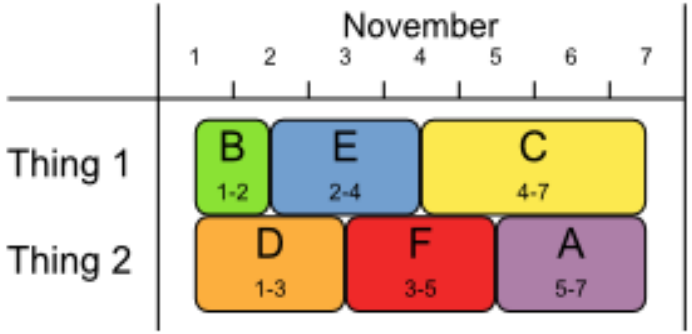


Find optimal solution and scale out for an NP-complete problem?

⇔ Is P = NP?

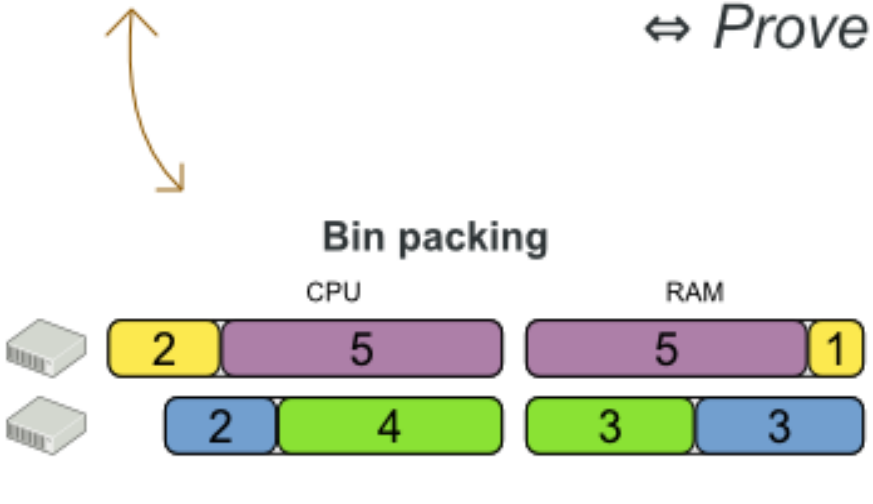
- Unresolved since 1971
- 1 000 000 $ reward since 2000
  - One of the 7 Millennium Problems (http://www.claymath.org/millennium/)
- Most believe P ≠ NP
  - ⇔ **Impossible to find optimal solution and scale out**
- 3000+ known NP-complete problems (wikipedia (http://en.wikipedia.org/wiki/List_of_NP-complete_problems))

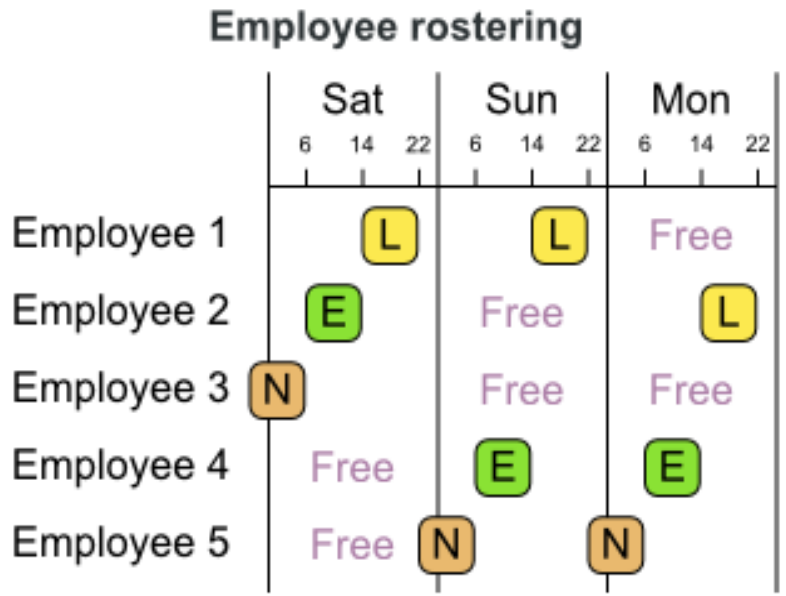




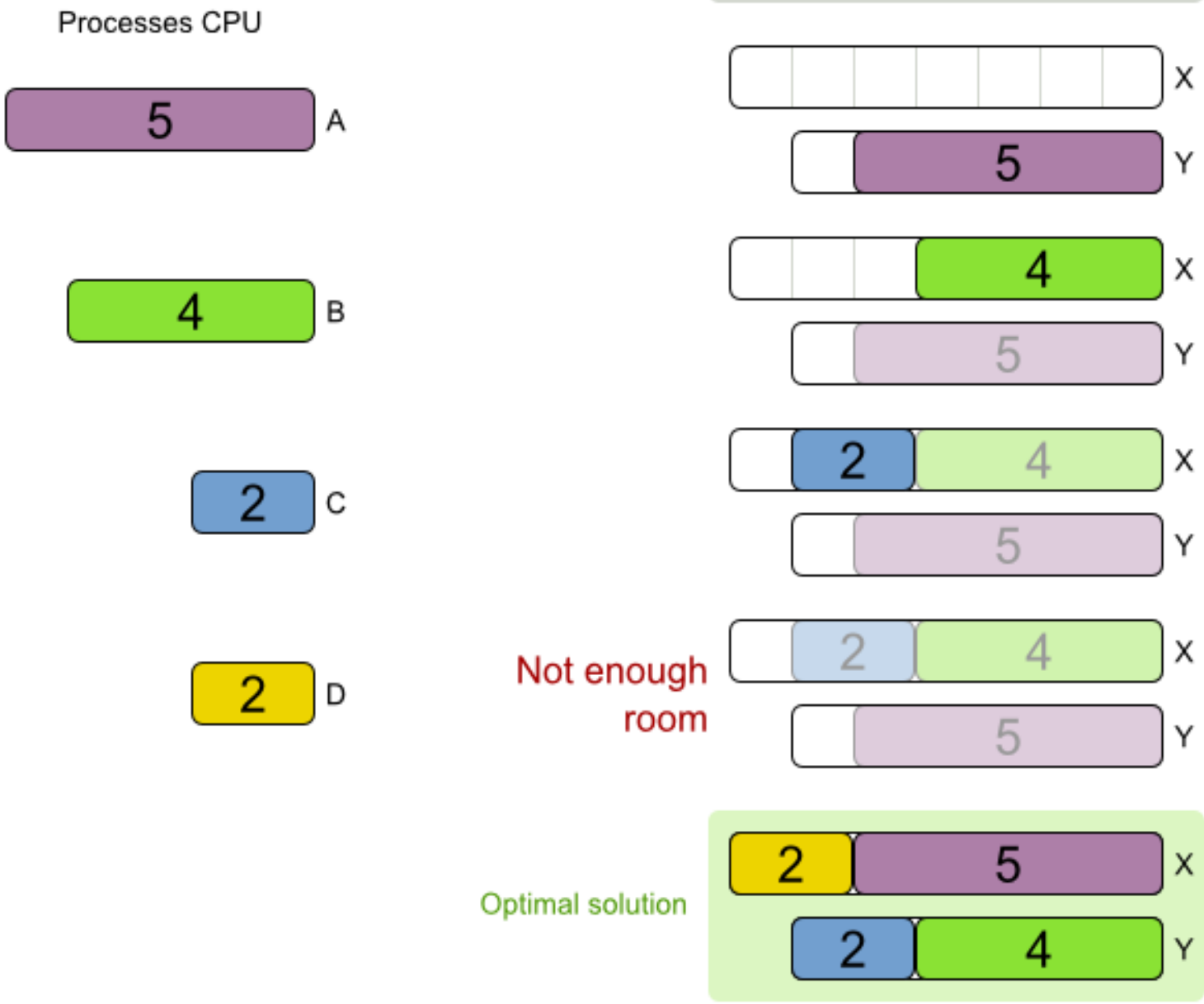
NP-complete interconnection

*Solve **one** use case*
*⇔ Solve **all** use cases*
⇔ Prove P = NP

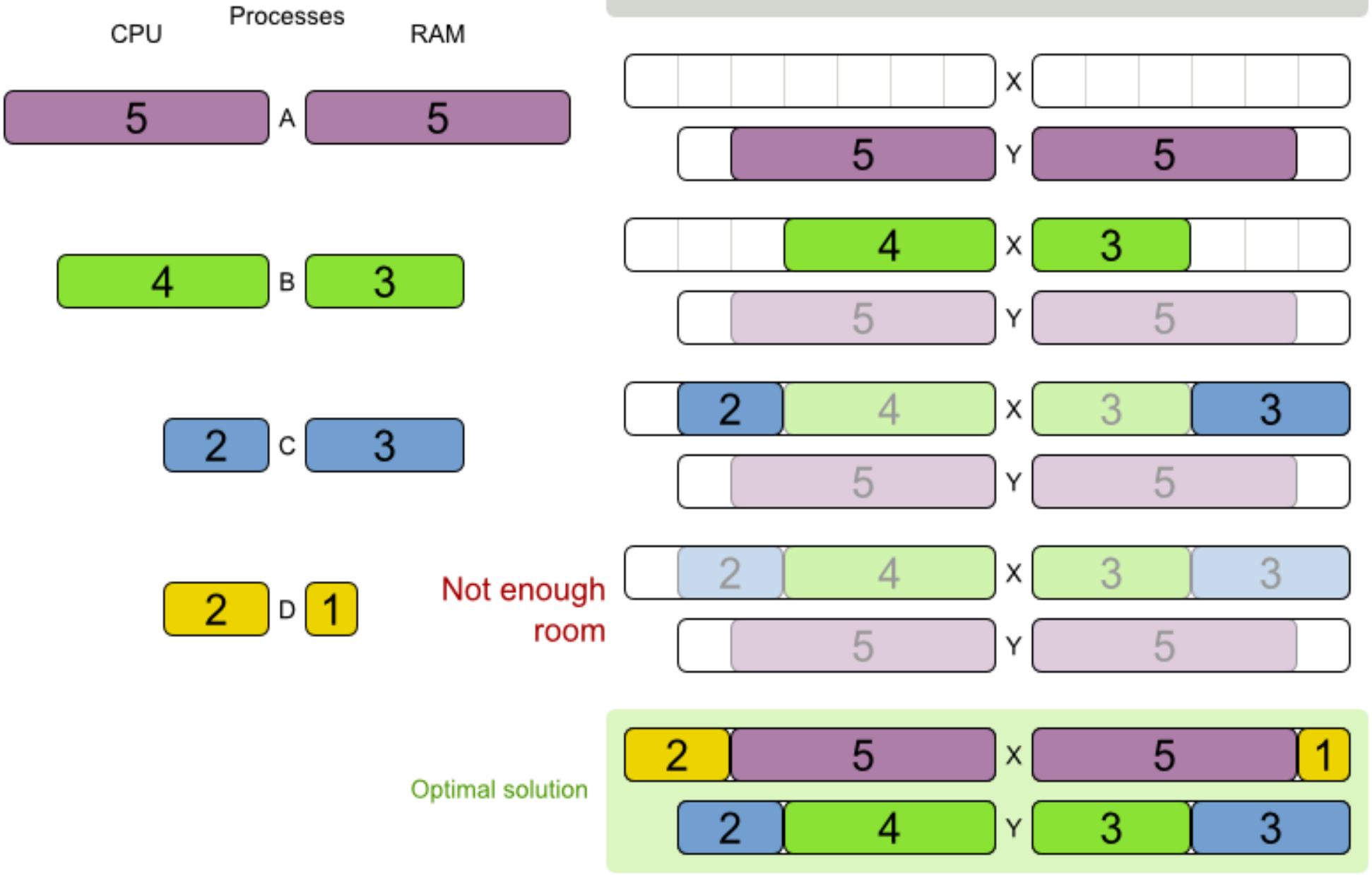




Multiple computers...
⇒ harder to solve

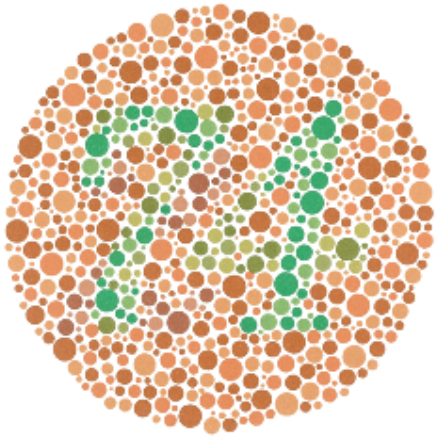


More constraints...
⇒ harder to solve



Planning problems are difficult to solve!

And human aren't good at it



But they don't realize it
(nor does their manager)

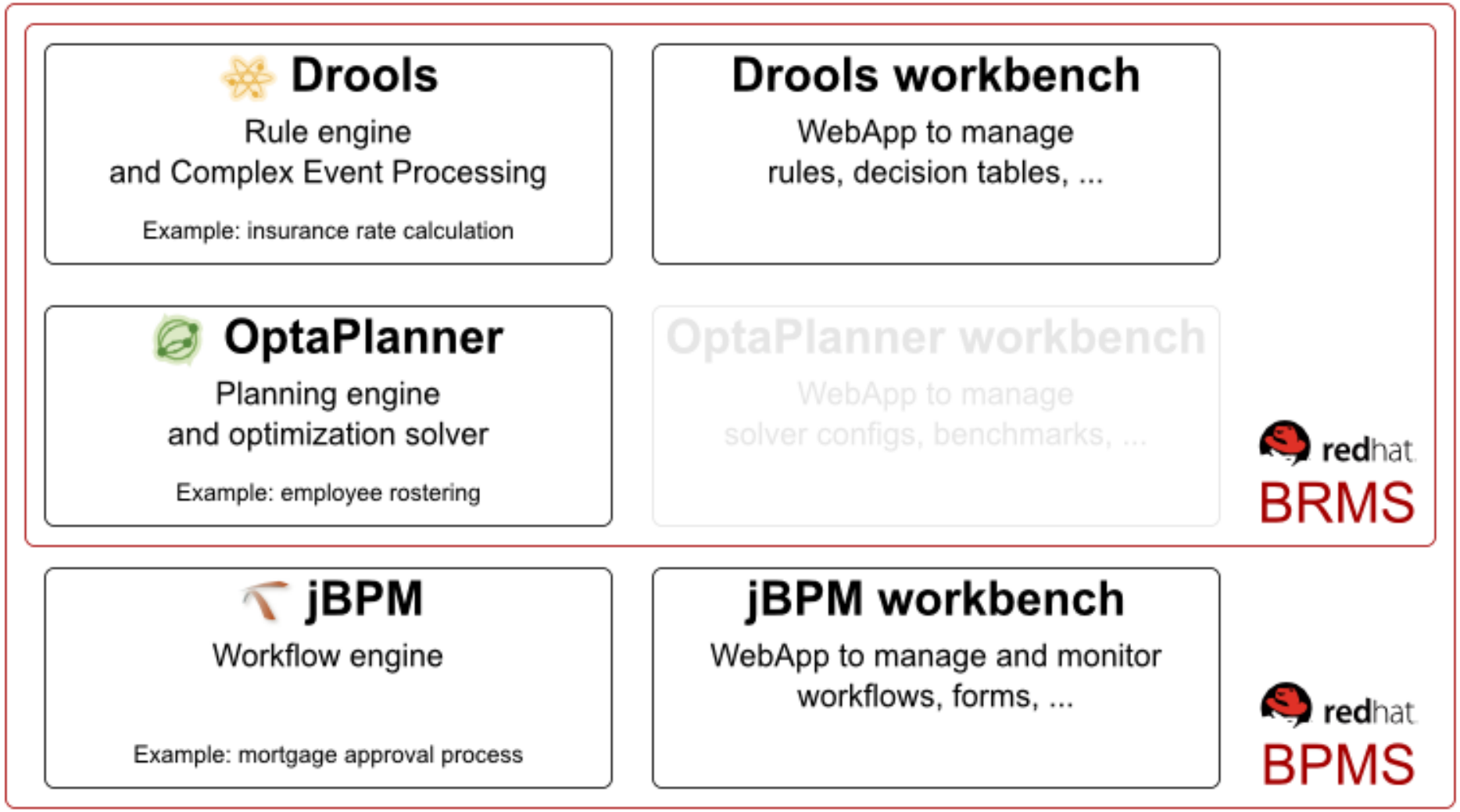
Reuse optimization algorithms

Find better solutions in time and scale out

- **Open source**
  Apache License
- **Regular releases**
  Download the zip or from Maven Central
- **Documented**
  Reference manual, examples, ...
- **Quality coverage**
  Unit, integration and stress tests

KIE functionality overview

What are the KIE projects?



Compatibility

OptaPlanner works on any Java Virtual Machine

Standard Java

Linux

Mac

Windows

...

Enterprise Java

JBoss AS

OPENSHIFT

...

JVM languages

Java

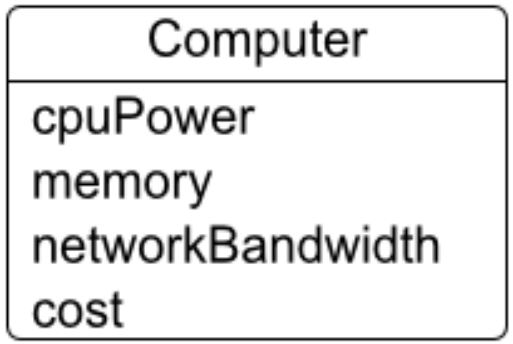
Scala

...

Cloud Balancing example

Domain model

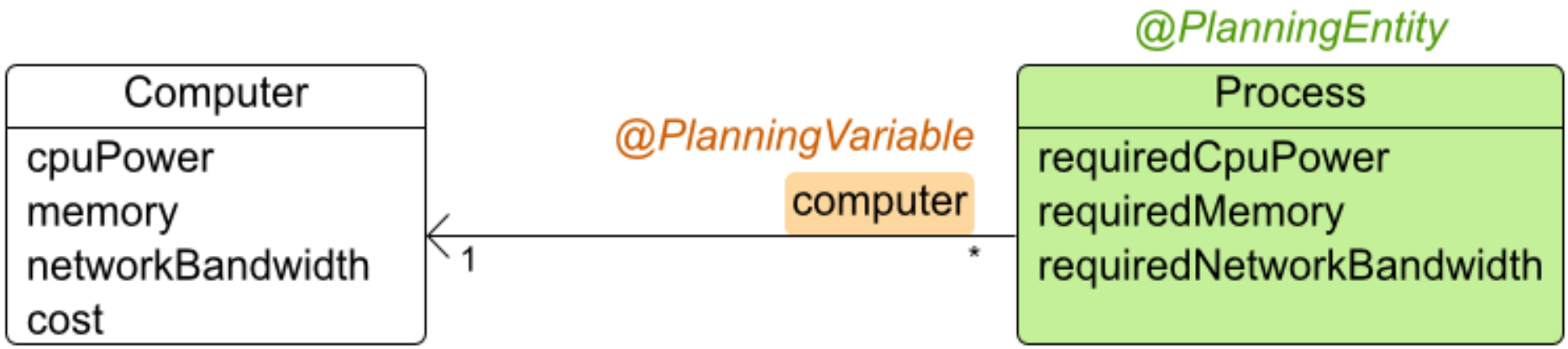
Cloud balance class diagram



Computer

```
public class Computer {

  private int cpuPower;
  private int memory;
  private int networkBandwidth;

  private int cost;

  // getters
}
```

Cloud balance class diagram



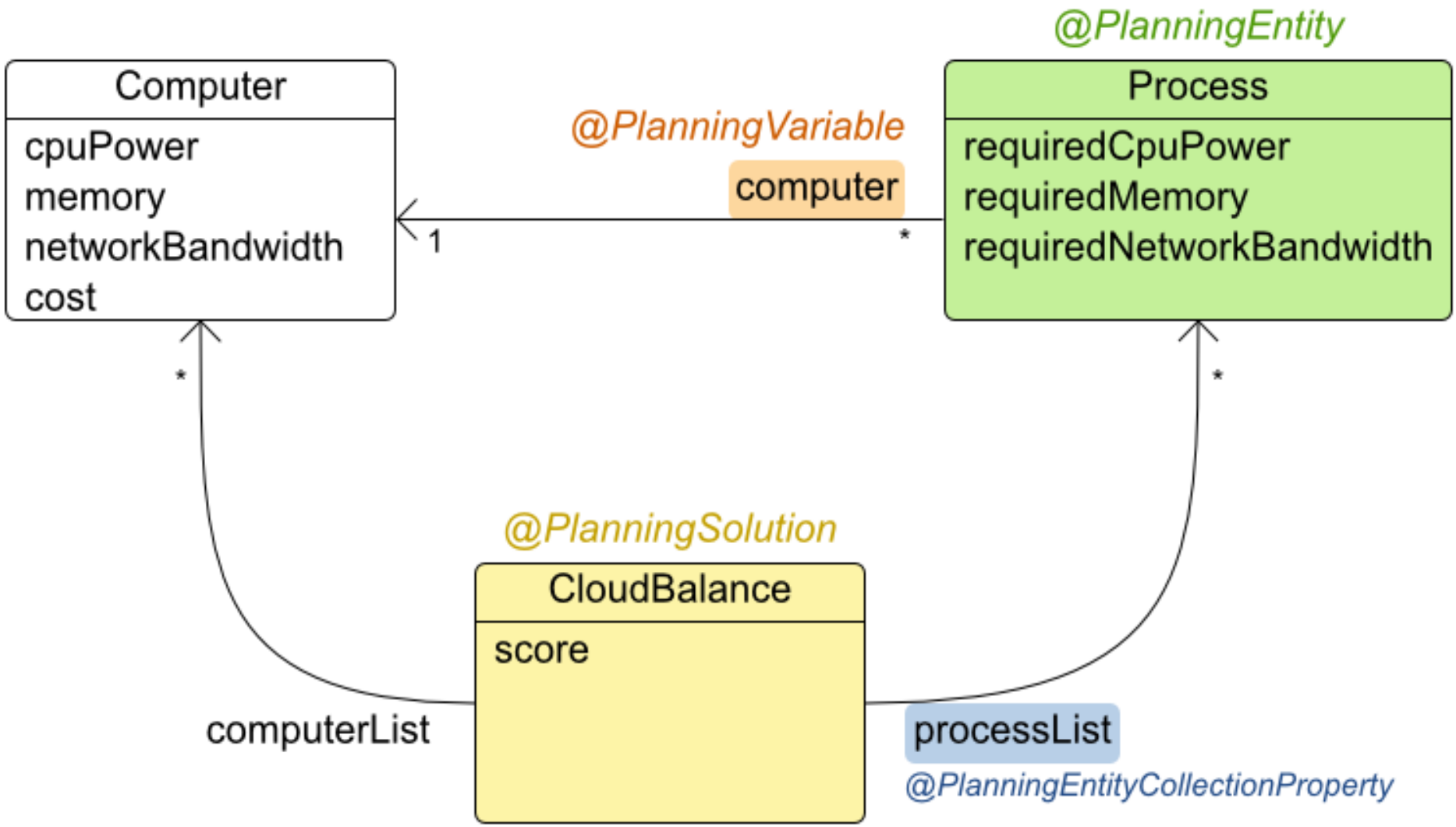
Process is a planning entity

```
@PlanningEntity
public class Process {

  private int requiredCpuPower;
  private int requiredMemory;
  private int requiredNetworkBandwidth;

  // getters

  ...
}
```

Process has a planning variable

```
@PlanningEntity
public class Process {
  ...

  private Computer computer;

  @PlanningVariable(valueRangeProviderRefs = {"computerRange"})
  public Computer getComputer() {
    return computer;
  }
  public void setComputer(Computer computer) {
    this.computer = computer;
  }

}
```

Cloud balance class diagram



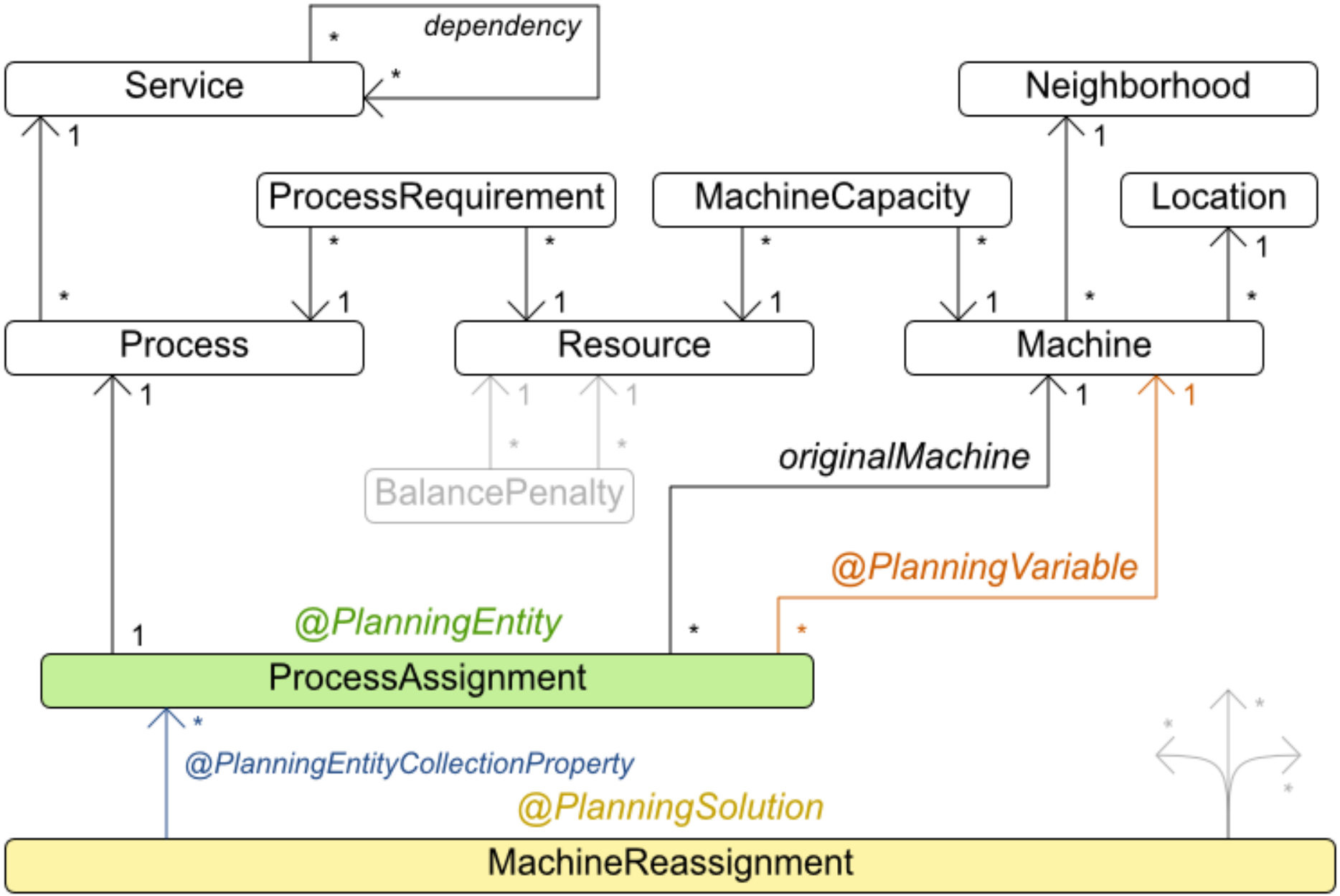
Solution CloudBalance

```
public class CloudBalance implements Solution<HardSoftScore> {

  private List<Computer> computerList;
  private List<Process> processList;

  @ValueRangeProvider(id = "computerRange")
  public List<Computer> getComputerList() {
    return computerList;
  }

  @PlanningEntityCollectionProperty
  public List<Process> getProcessList() {
    return processList;
  }

  ...
}
```

Solution CloudBalance: score

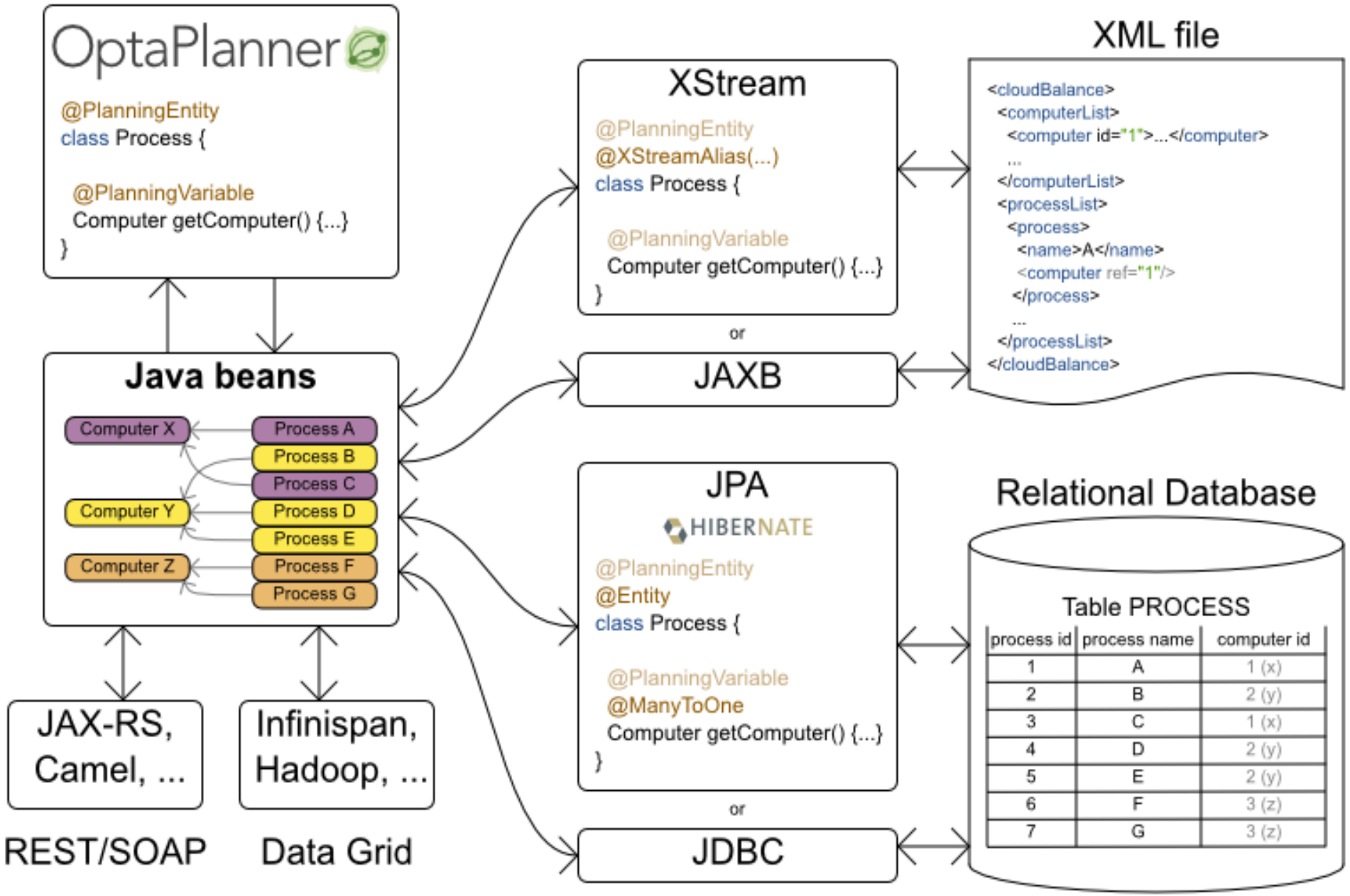
```
public class CloudBalance implements Solution<HardSoftScore> {
  ...

  private HardSoftScore score;

  public HardSoftScore getScore() {
    return score;
  }
  public void setScore(HardSoftScore score) {
    this.score = score;
  }

}
```

Machine reassignment class diagram



Integration overview

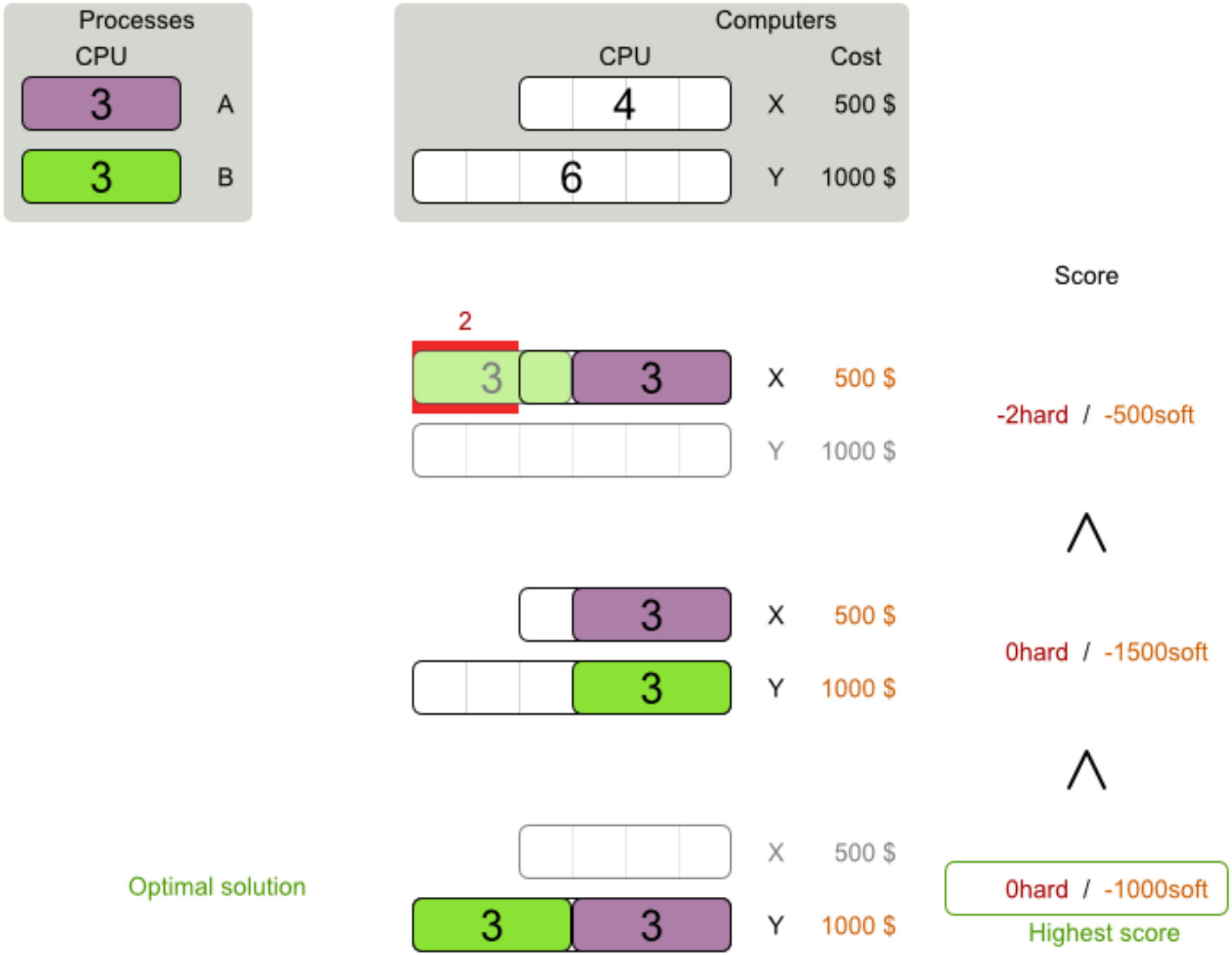
OptaPlanner combines easily with other Java and JEE technologies.



Cloud Balancing example

Score constraints

Given 2 solutions which one is better?


Processes
CPU
3
A
3
B
Computers
CPU
Cost
4
X
500 $
6
Y
1000 $
Score
2
3
3
X
500 $
Y
1000 $
-2hard / -500soft
3
X
500 $
3
Y
1000 $
0hard / -1500soft
X
500 $
3
3
Y
1000 $
0hard / -1000soft
Highest score
Optimal solution

Score calculation

- Easy Java
- Incremental Java
- Drools

Easy Java score calculation

- Easy to implement
- Bridge an existing system
- Slow

```
public class CloudBalancingEasyScoreCalculator
    implements EasyScoreCalculator<CloudBalance> {

  public Score calculateScore(CloudBalance cb) {
    ...
    return HardSoftScore.valueOf(hardScore, softScore);
  }

}
```

Incremental Java score calculation

- Fast
  - Solution changes ⇒ recalculate score delta only
- Hard to implement
  - Much boilerplate code

Drools score calculation

- Incremental
  - No boilerplate code
- Constraints in Drools Rule Language (DRL)
  - Declarative (like SQL, regular expression)
- Integration opportunities
  - Drools Workbench
  - Decision tables

Solution CloudBalance: getProblemFacts()

```
public class CloudBalance implements Solution<HardSoftScore> {

  private List<Computer> computerList;
  private List<Process> processList;
  ...

  // Used in Drools score rules (DRL)
  public Collection<Object> getProblemFacts() {
    List<Object> facts = new ArrayList<Object>();
    facts.addAll(computerList);
    return facts;
  }

  ...
}
```

DRL soft constraint: computer cost

```
rule "computerCost"
  when
    // there is a computer
    $s : Computer($c : cost)
    // there is a processes on that computer
    exists Process(computer == $s)
  then
    // lower soft score by the maintenance cost
    scoreHolder.addSoftConstraintMatch(kcontext, - $c);
end
```

DRL hard constraint: CPU power

```
rule "requiredCpuPowerTotal"
  when
    // there is a computer
    $s : Computer($cpu : cpuPower)
    // with too little cpu for its processes
    $total : Number(intValue > $cpu) from accumulate(
      Process(computer == $s, $requiredCpu : requiredCpuPower),
      sum($requiredCpu)
    )
  then
    // lower hard score by the excessive CPU usage
    scoreHolder.addHardConstraintMatch(kcontext,
        $cpu - $total.intValue());
end
```

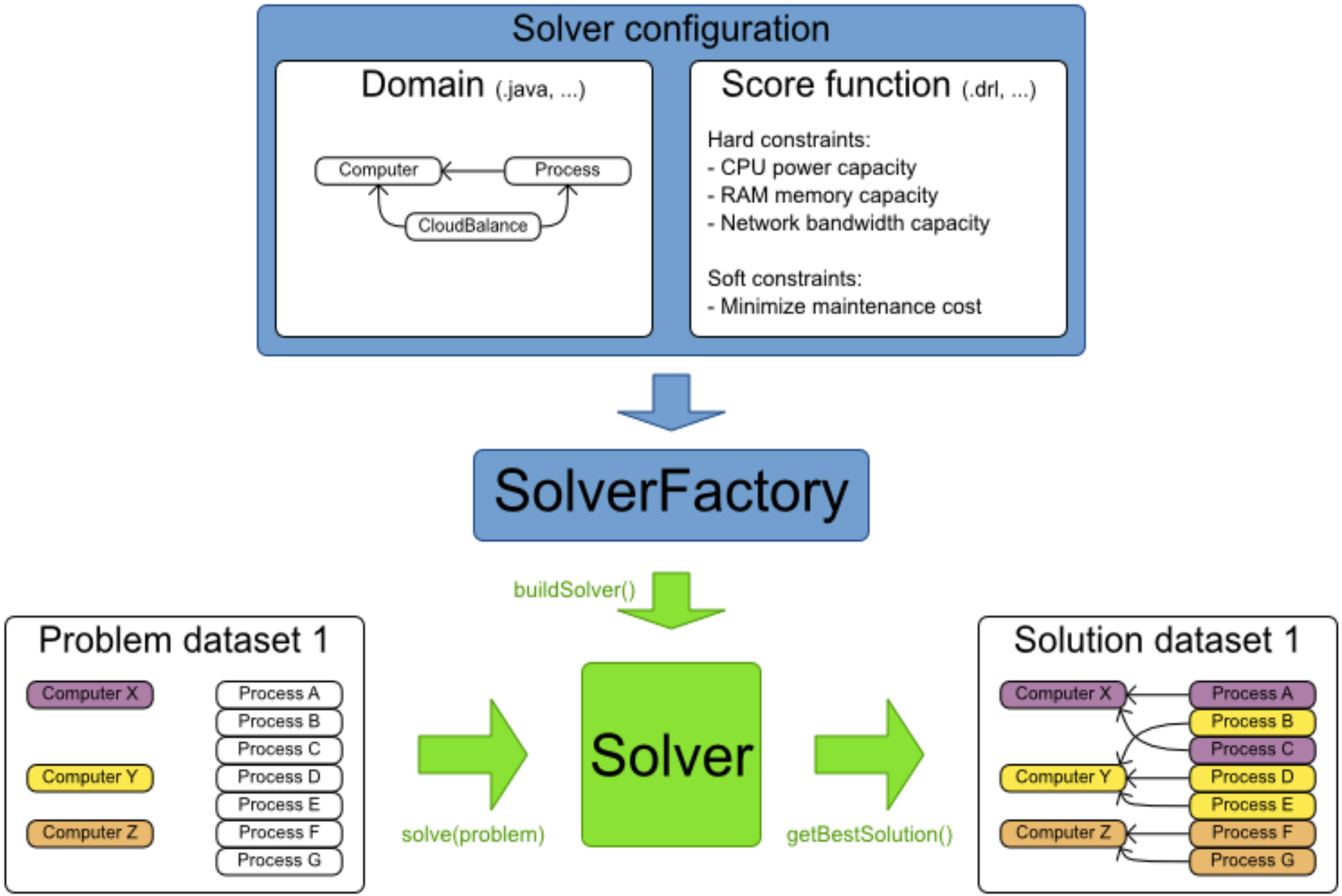
Score calculation must be flexible

- **Optimal solution for *almost* your business problem is useless**
- Model supports:
  - Reusing existing classes
  - Rich, OO class hierarchies (including polymorphism)
- Constraints supports:
  - Any constraint (no linear or quadratic restrictions!)
  - Reusing existing code
- Scoring supports:
  - Positive/negative mix
  - Score weights
  - Unlimited score levels

Cloud Balancing example

Solving it

Input/Output overview



Solver configuration by XML

```
<solver>
  <solutionClass>...CloudBalance</solutionClass>
  <entityClass>...Process</entityClass>

  <scoreDirectorFactory>
    <scoreDefinitionType>HARD_AND_SOFT</scoreDefinitionType>
    <scoreDrl>...ScoreRules.drl</scoreDrl>
  </scoreDirectorFactory>

  <!-- optimization algorithms -->
</solver>
```

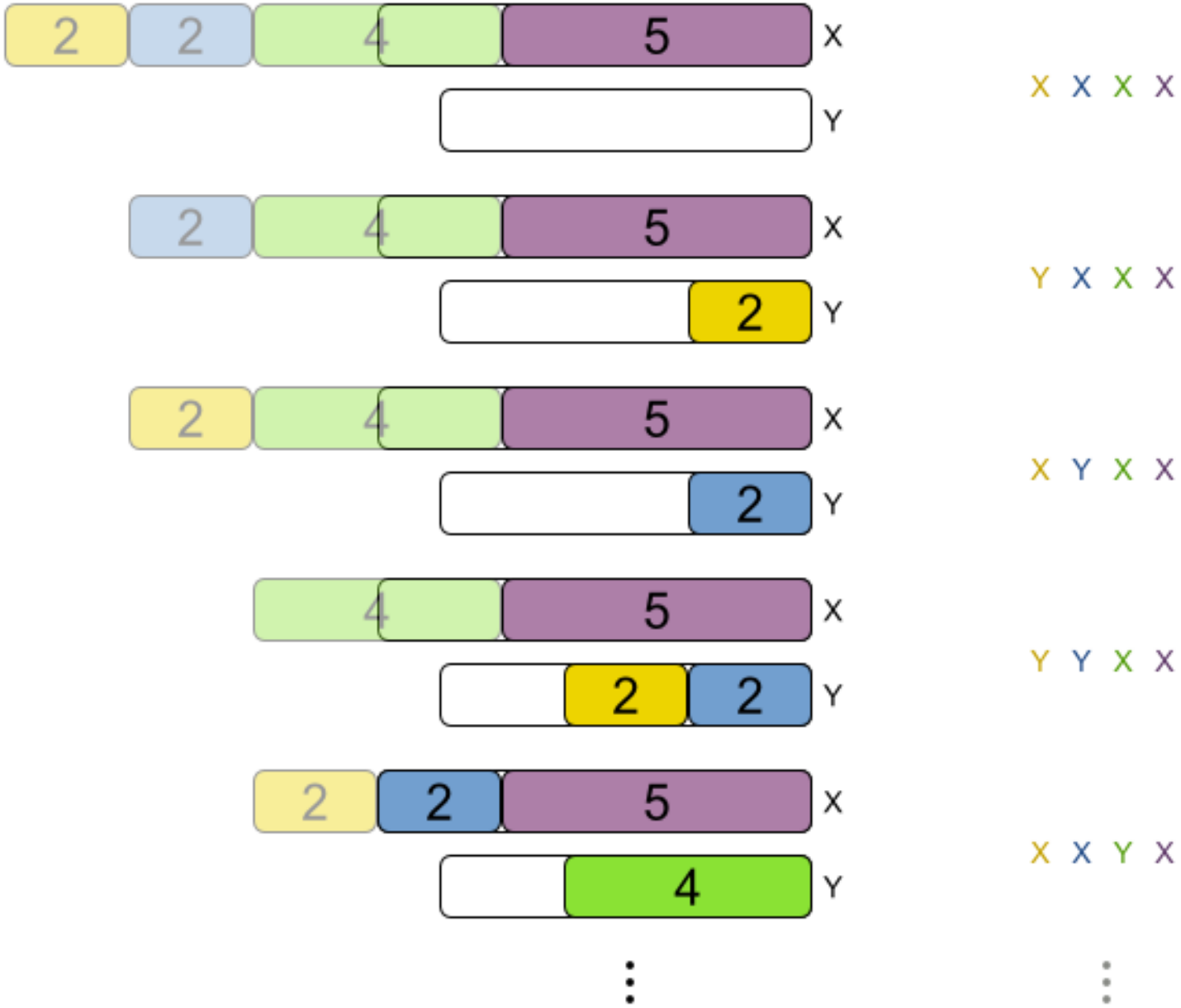
Solving

```
SolverFactory factory = SolverFactory.createFromXmlResource(
    "...SolverConfig.xml");
Solver solver = factory.buildSolver();

solver.solve(cloudBalance);
cloudBalance = (CloudBalance) solver.getBestSolution();
```

Cloud Balancing example

Optimization algorithms

Brute Force

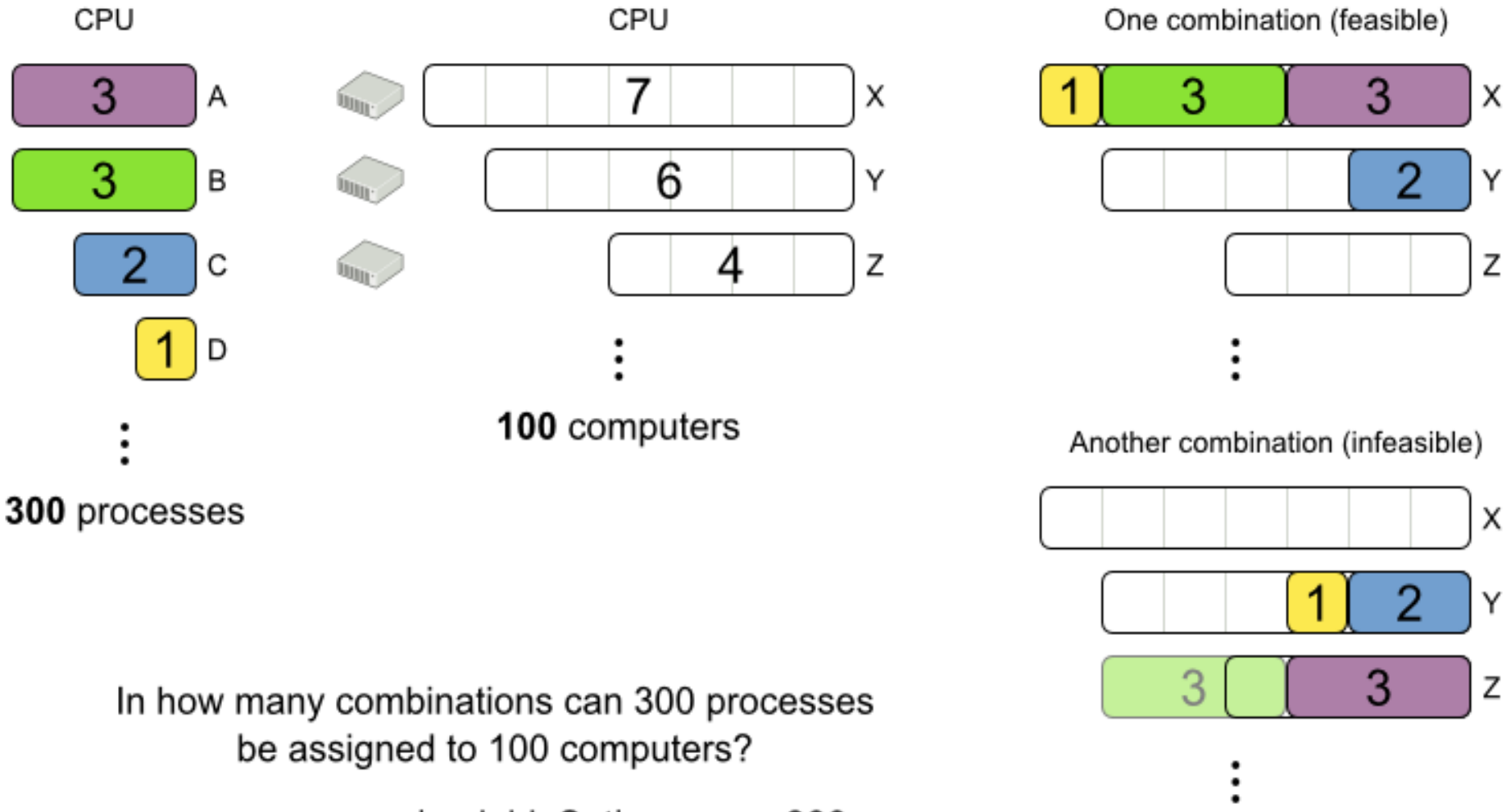

2
2
4
5
X
Y
2
4
5
X
2
Y
2
4
5
X
2
Y
4
5
X
2
2
Y
2
2
5
X
4
Y
X X X X
Y X X X
X Y X X
Y Y X X
X X Y X

Brute force config

```
<solver>
  ...

  <exhaustiveSearch>
    <exhaustiveSearchType>BRUTE_FORCE</exhaustiveSearchType>
  </exhaustiveSearch>
</solver>
```

What is the size of the search space?

How big is the haystack?

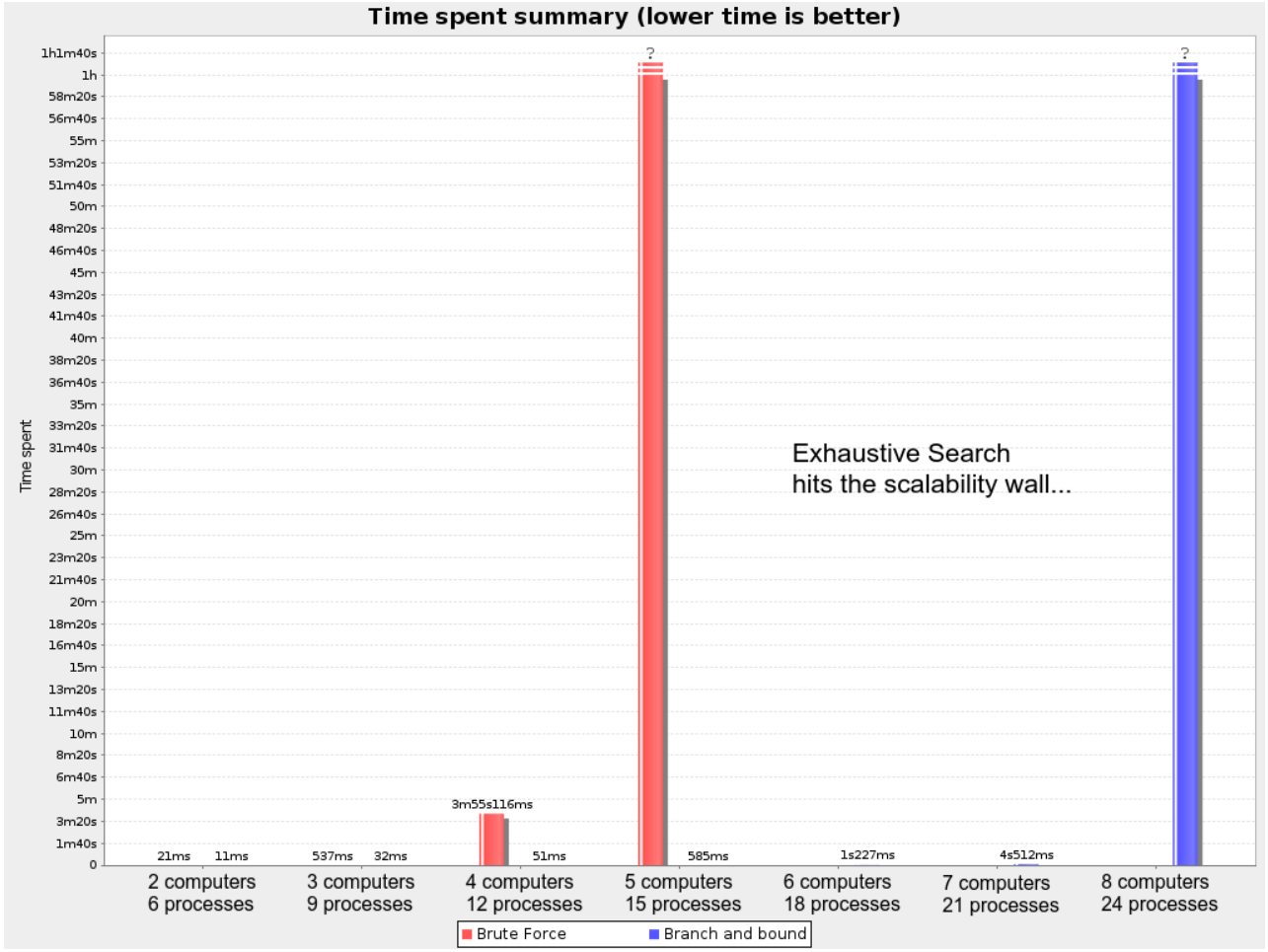


In how many combinations can 300 processes be assigned to 100 computers?

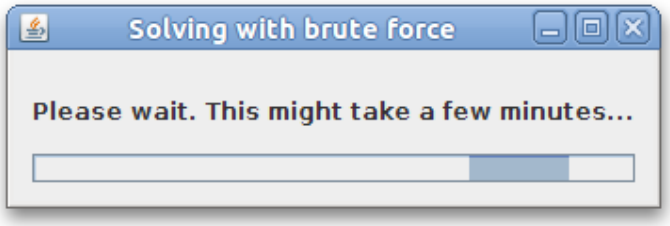
$$|\text{valueSet}|^{|\text{variableSet}|} = \mathbf{100^{300}}$$

$$= 10^{600} = 1000000000000000000000...$$

Brute force scalability



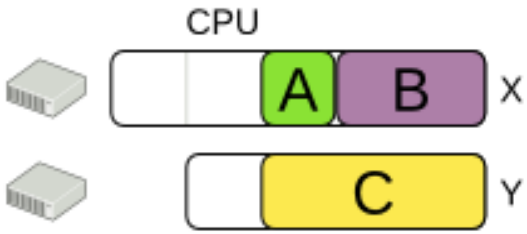
Plan 1200 processes with brute force?



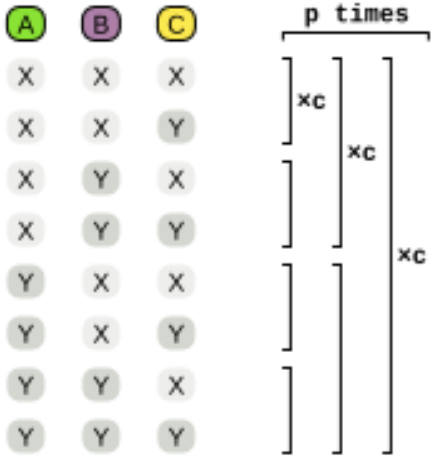
Calculate the size of the search space

Given a Solution model, how many different combinations can it represent?

Cloud balancing



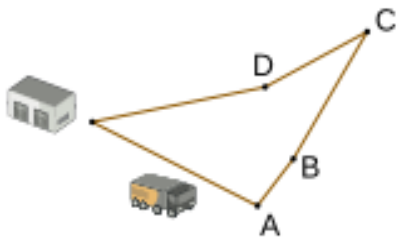
Model:



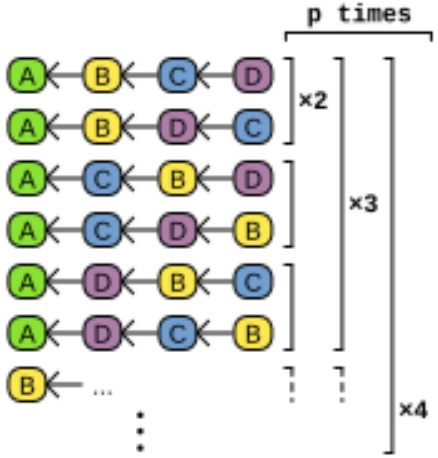
Search space: c^p

# computers	# processes	search space
2	3	8
100	300	10^{680}
200	600	10^{1380}
400	1200	10^{6967}

Traveling salesman (TSP)



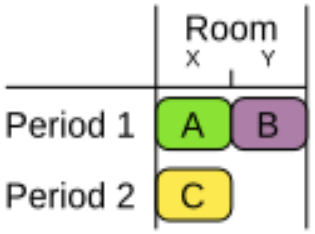
Model: linked list



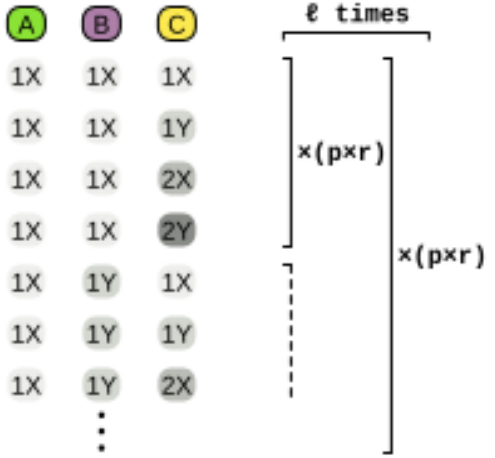
Search space: $n!$

# customers	search space
4	24
100	10^{157}
1000	10^{2567}
10000	10^{35659}

Course scheduling



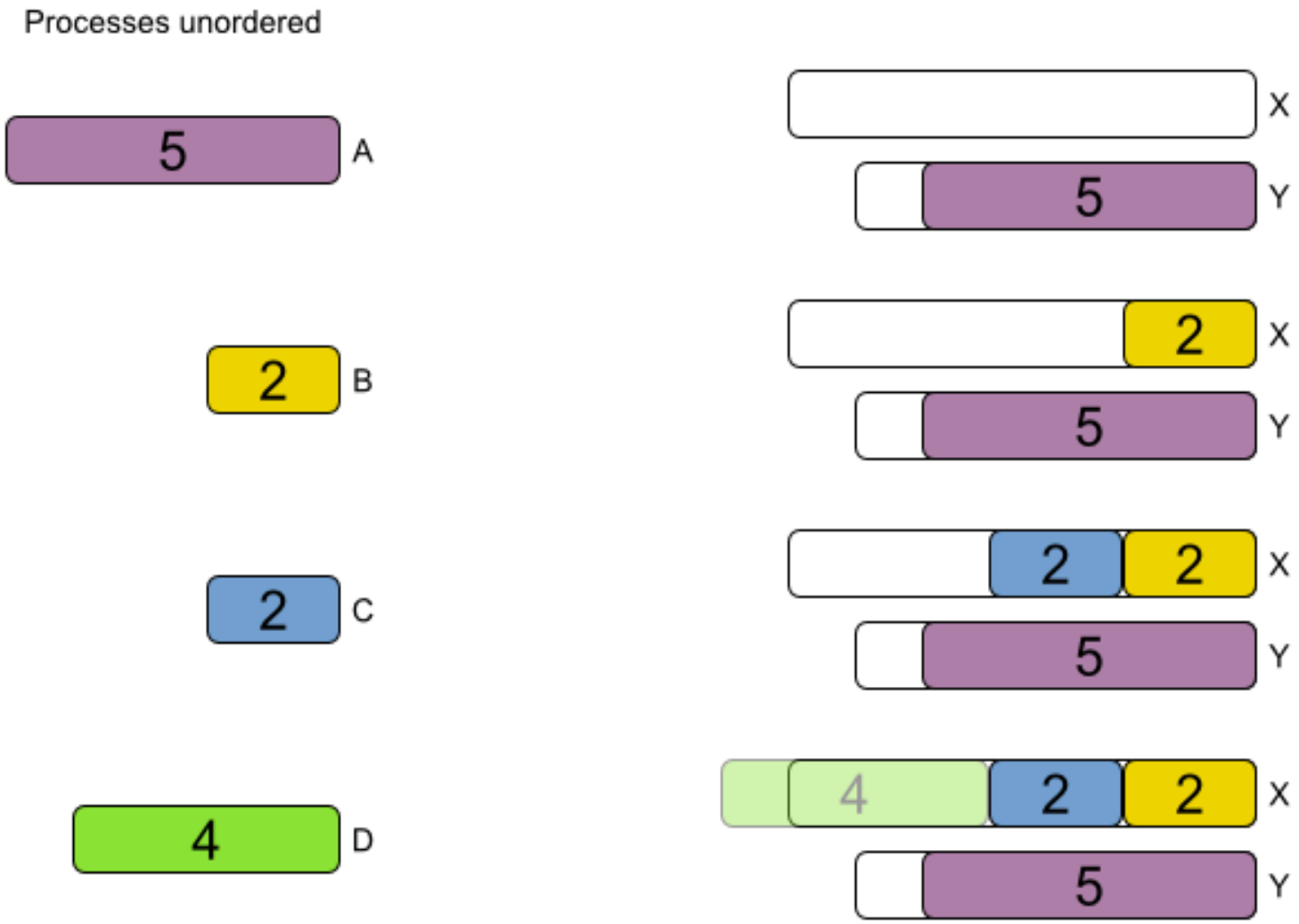
Model:



Search space: $(p \times r)^{\ell}$

# periods	# rooms	# lectures	space
2	2	3	64
36	6	100	10^{233}
36	18	400	10^{1124}
36	36	800	10^{2490}

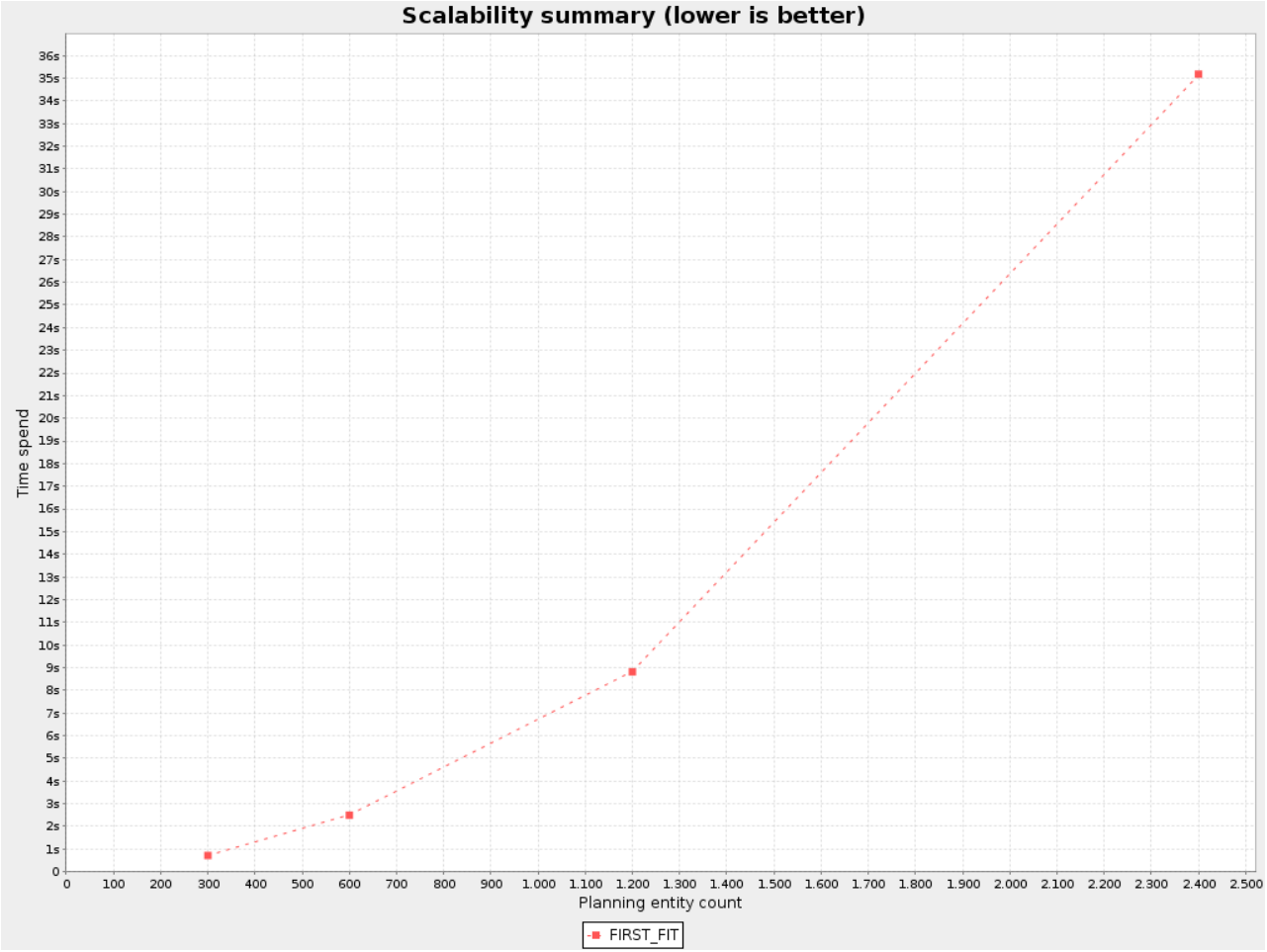
First Fit



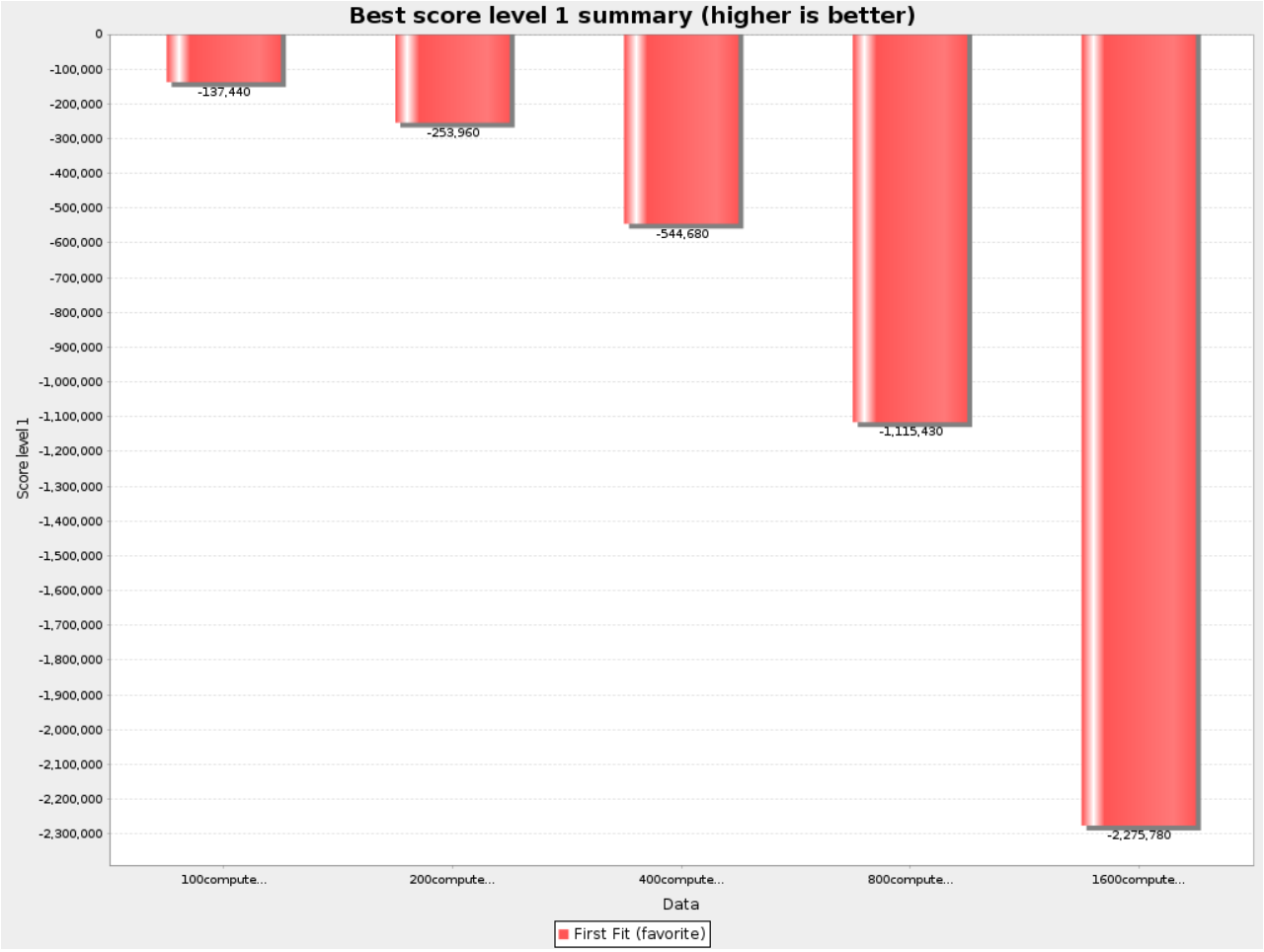
First Fit config

```
<solver>
  ...

  <constructionHeuristic>
    <constructionHeuristicType>FIRST_FIT</constructionHeuristicType>
  </constructionHeuristic>
</solver>
```

First Fit scalability

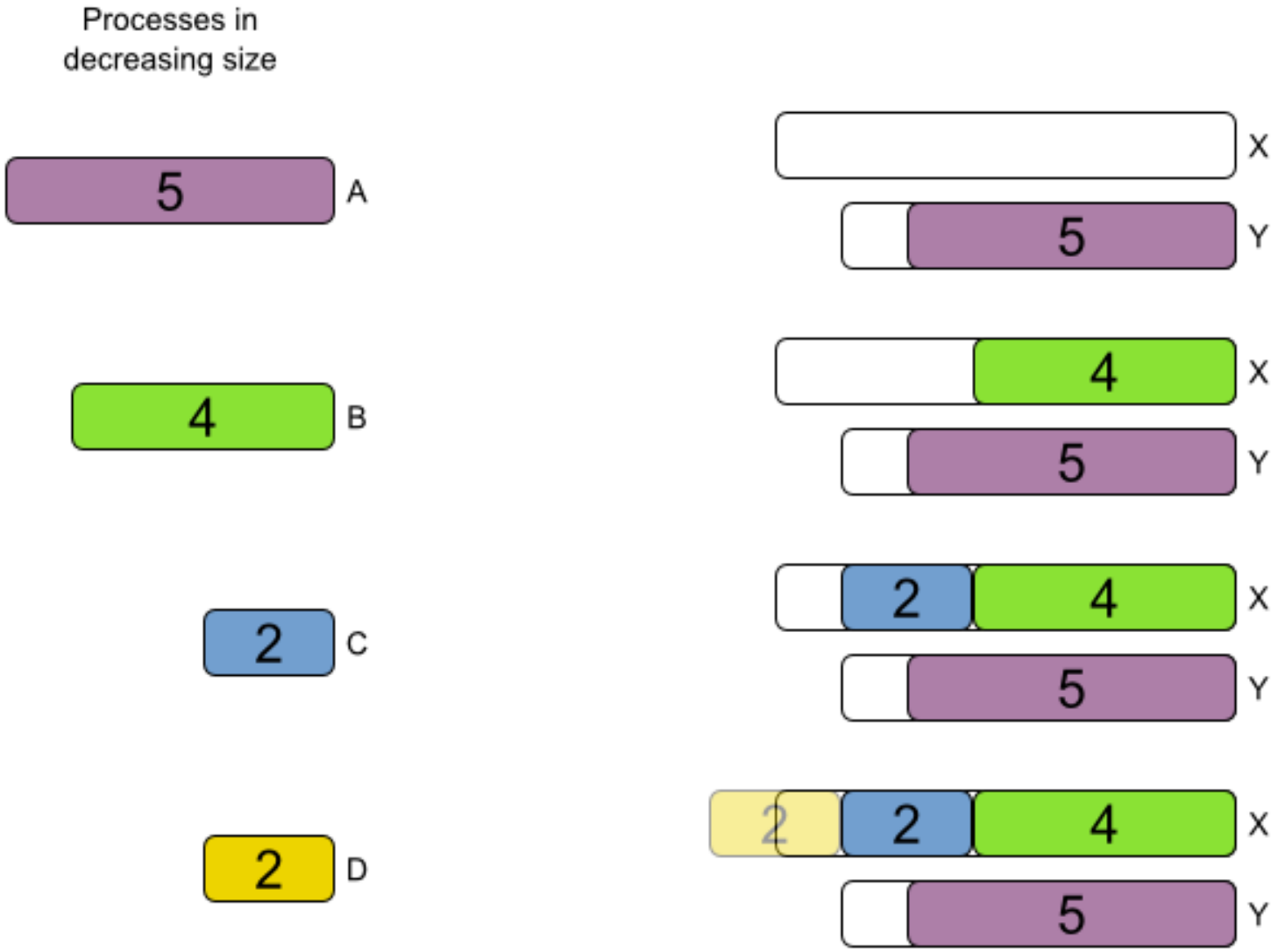


First Fit results



All hard constraints satisfied: maintenance cost shown

First Fit Decreasing



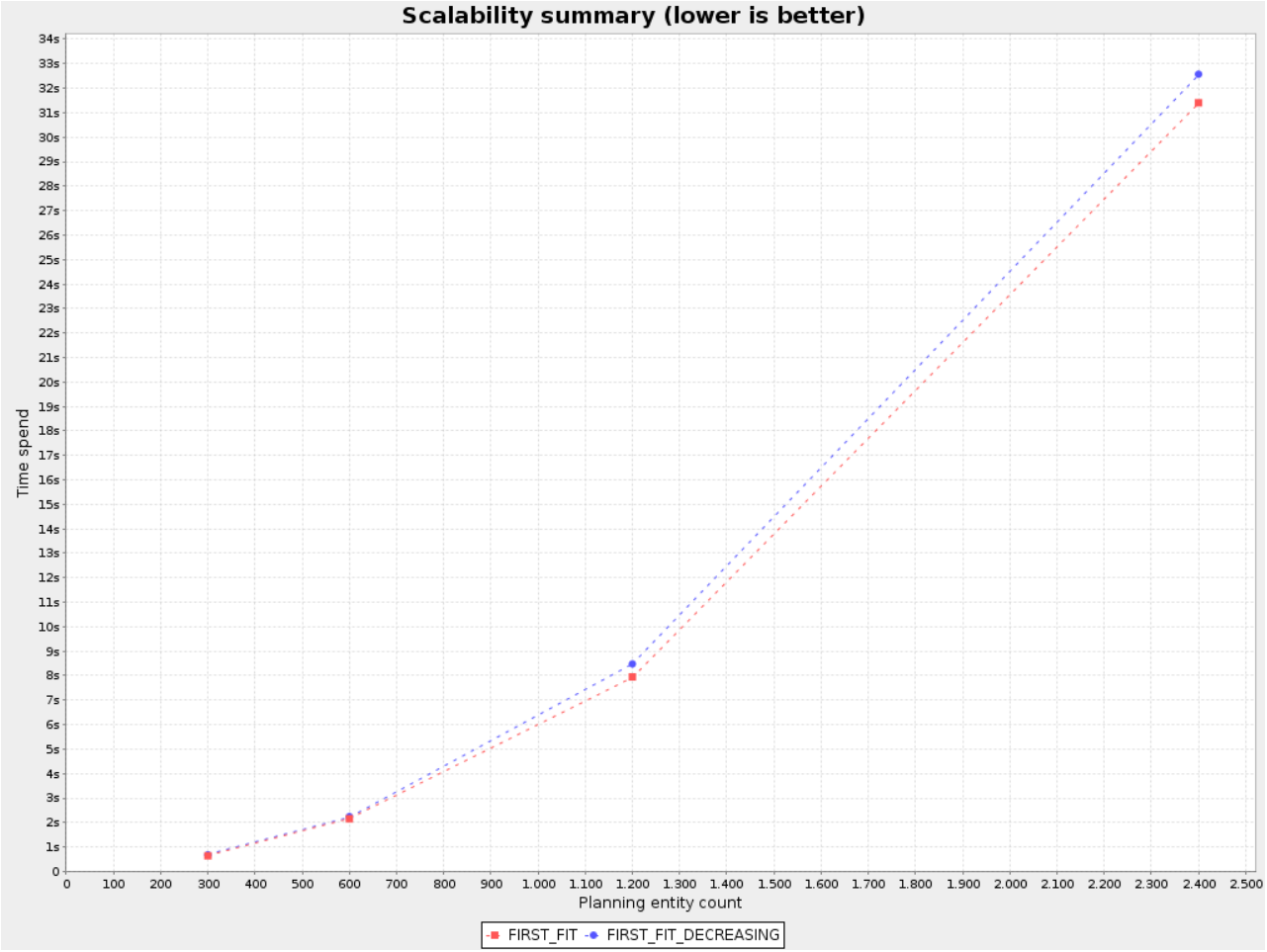
First Fit Decreasing config

```
<solver>
  ...

  <constructionHeuristic>
    <constructionHeuristicType>FIRST_FIT_DECREASING</constructionHeuristicTyp
e>
  </constructionHeuristic>
</solver>
```

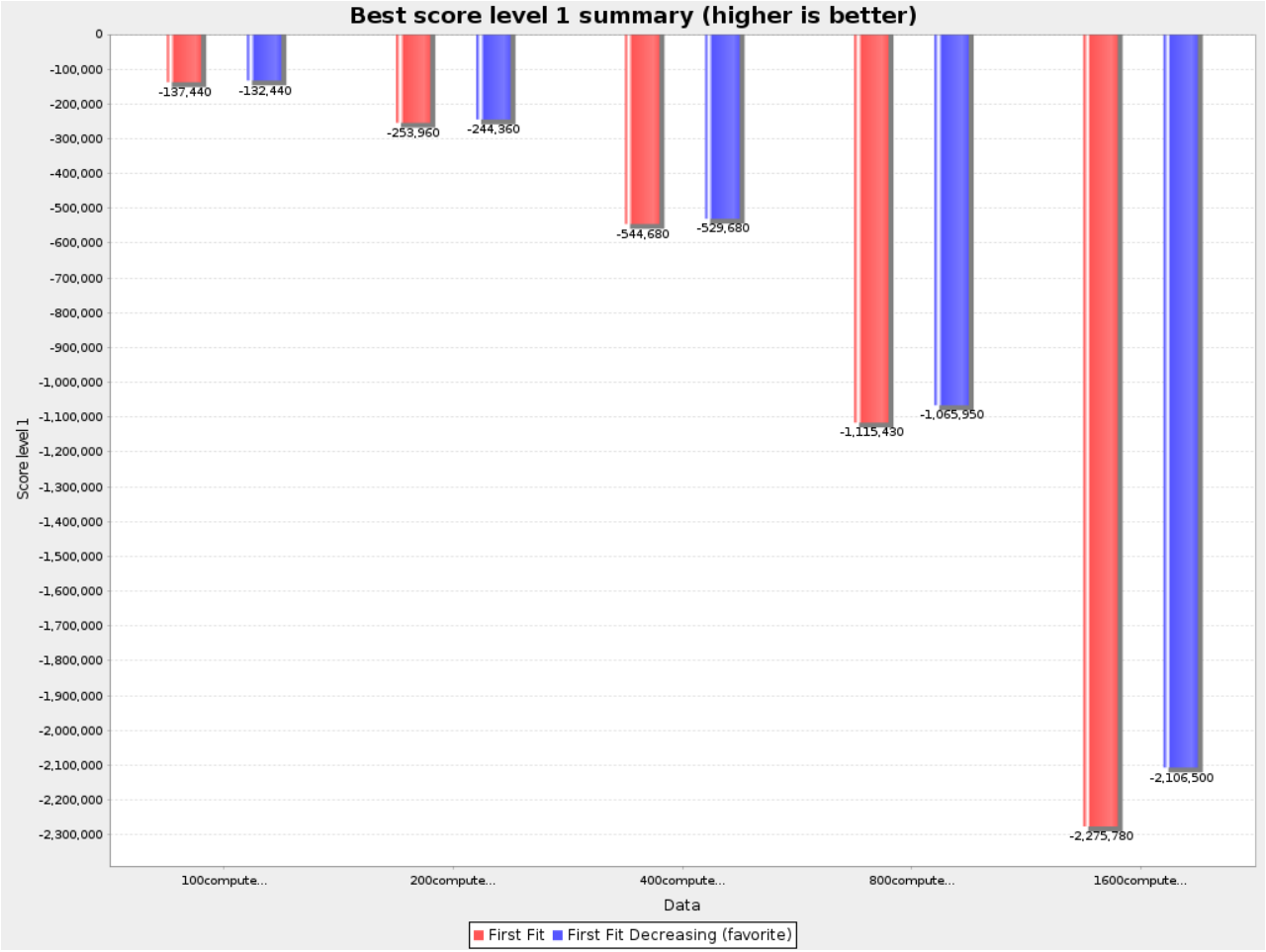
DifficultyComparator

```
public class ProcessDifficultyComparator
    implements Comparator<Process> {
  public int compare(Process a, Process b) {
    // Compare on requiredCpuPower * requiredMemory
    //     * requiredNetworkBandwidth
  }
}

@PlanningEntity(difficultyComparatorClass
    = ProcessDifficultyComparator.class)
public class Process  {
  ...
}
```

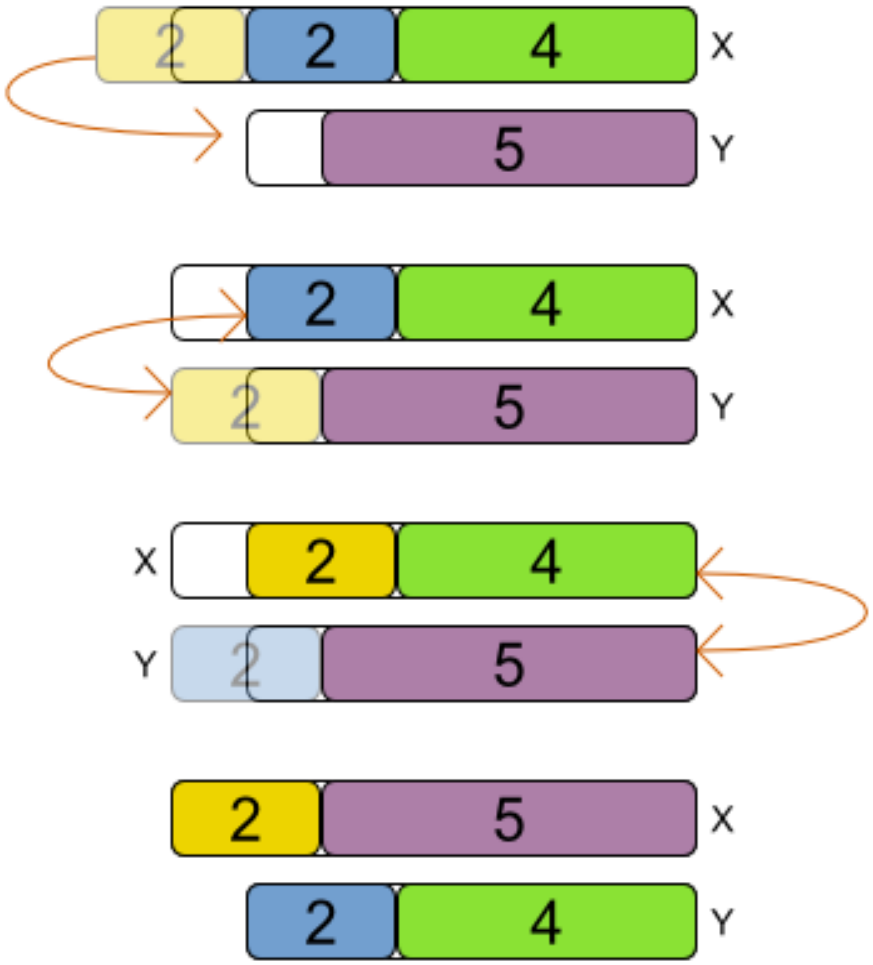
First Fit Decreasing scalability



First Fit Decreasing results

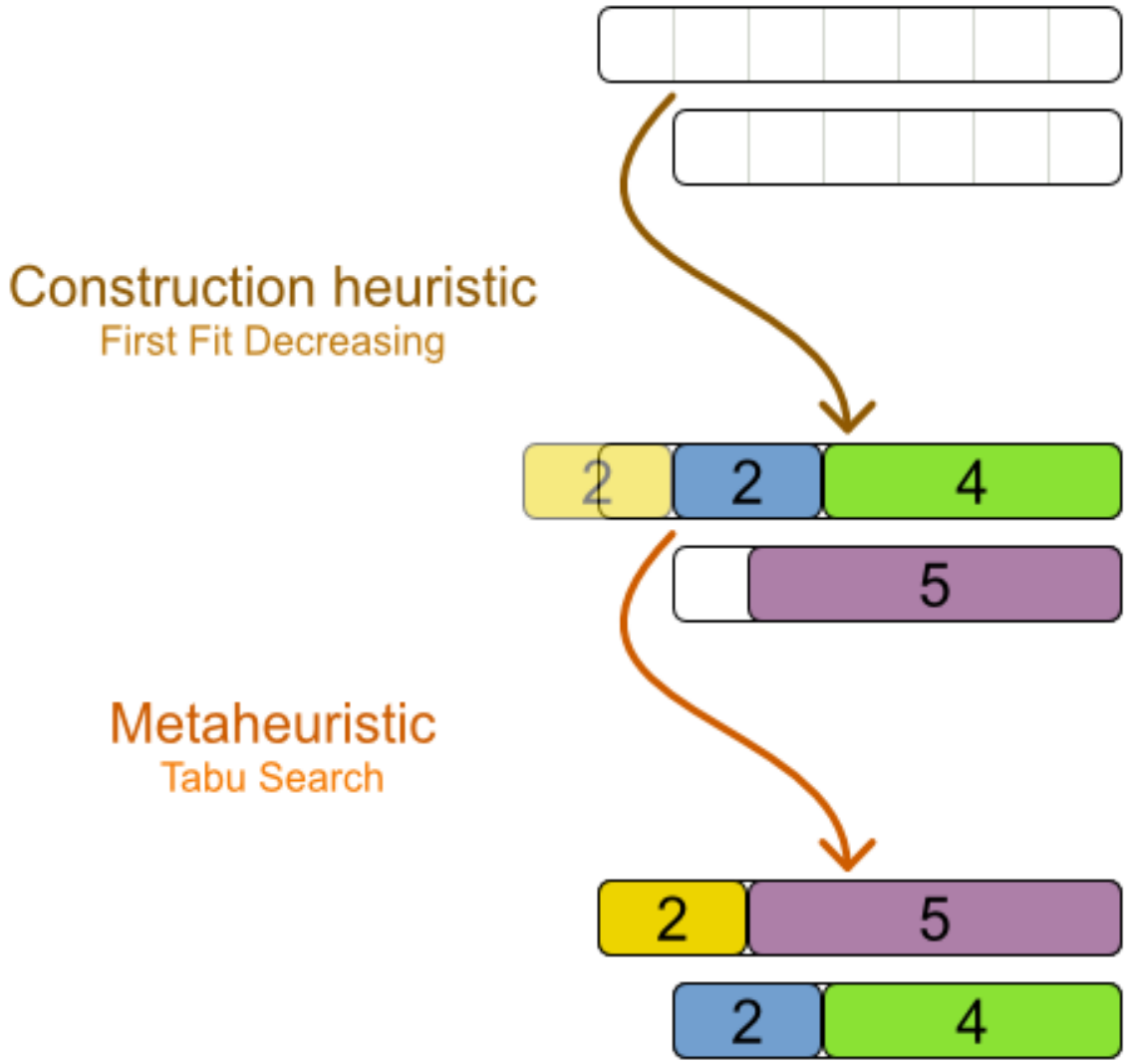


Local Search



General phase sequence

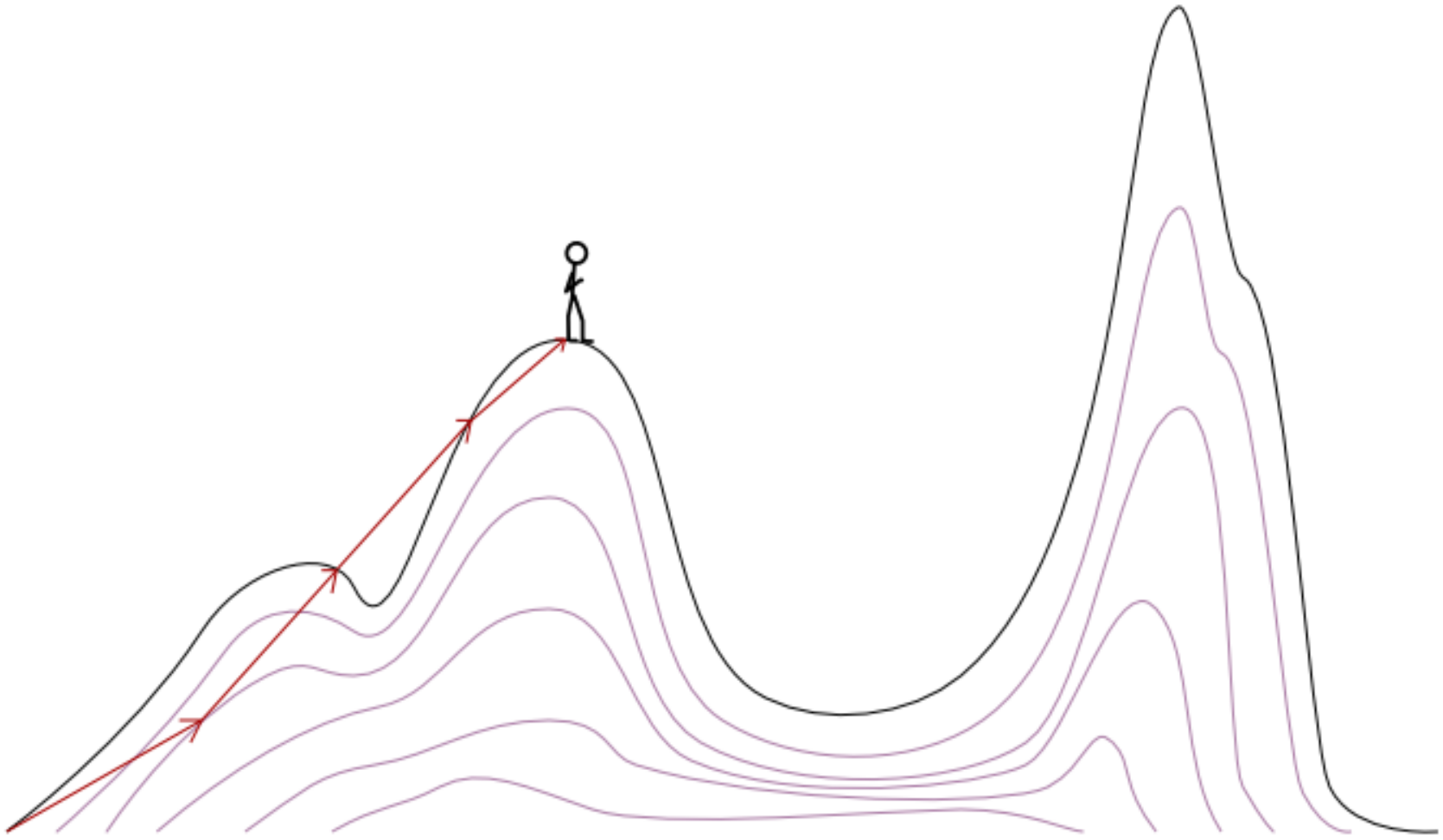
First a construction heuristic,
then metaheuristics



Construction Heuristics + Local Search

```
<solver>
  ...

  <constructionHeuristic>
    <constructionHeuristicType>FIRST_FIT_DECREASING</constructionHeuristicTyp
e>
  </constructionHeuristic>
  <localSearch>
    ...
  <localSearch>
</solver>
```

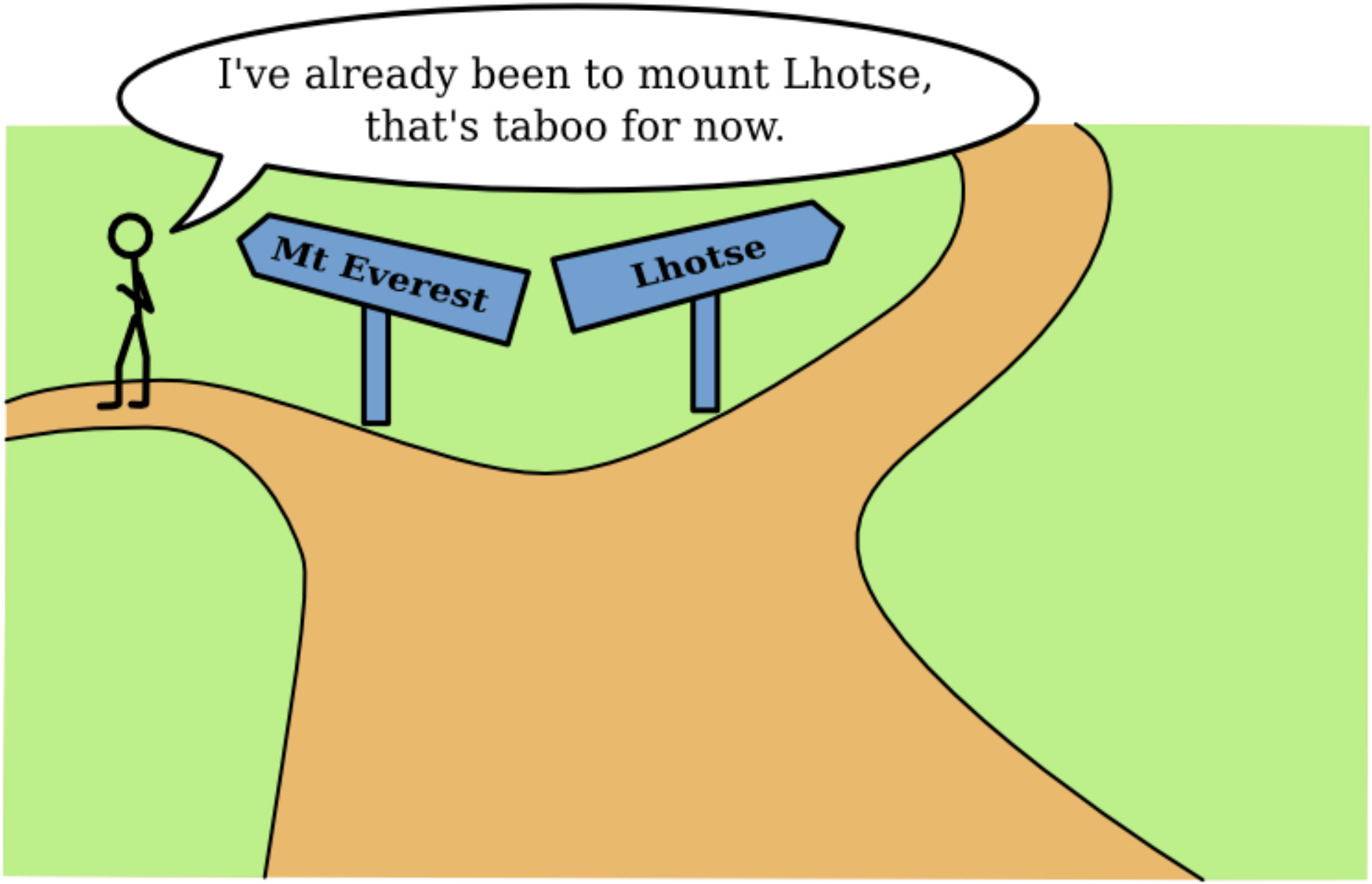
Hill climbing



Hill Climbing config

```
<localSearch>
  <forager>
    <!-- Untweaked standard value -->
    <acceptedCountLimit>1000</acceptedCountLimit>
  </forager>
</localSearch>
```

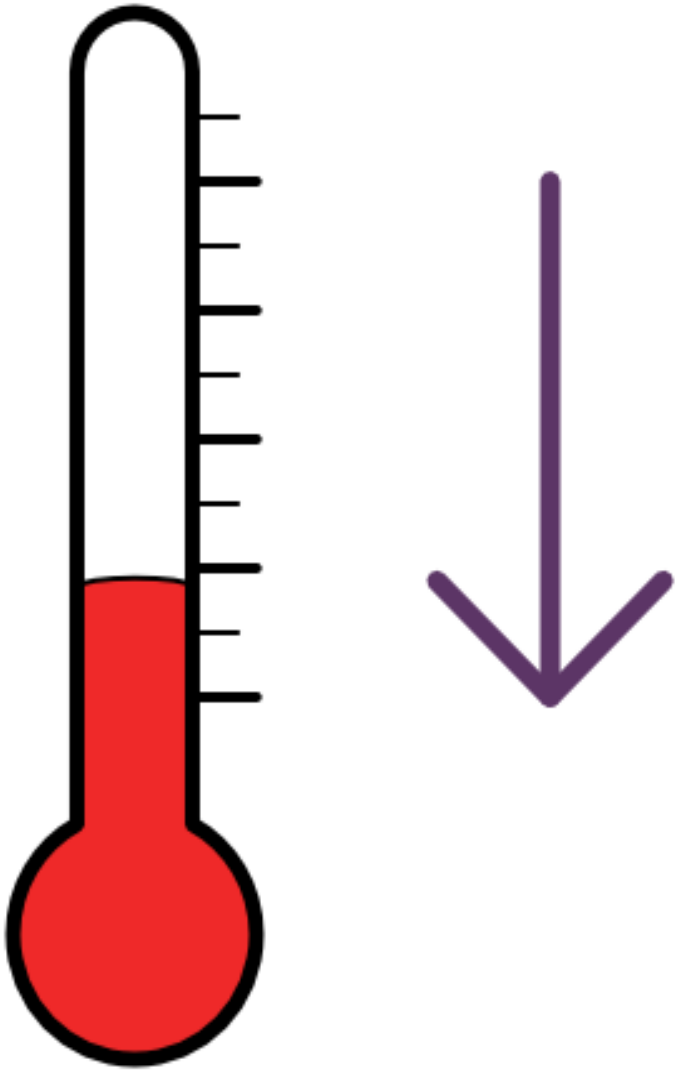
Tabu Search


I've already been to mount Lhotse,
that's taboo for now.
Mt Everest
Lhotse

Tabu Search config

```
<localSearch>
  <acceptor>
    <!-- Typical standard value -->
    <entityTabuSize>7</entityTabuSize>
  </acceptor>
  <forager>
    <!-- Typical value -->
    <acceptedCountLimit>1000</acceptedCountLimit>
  </forager>
</localSearch>
```

Simulated Annealing



Simulated Annealing config

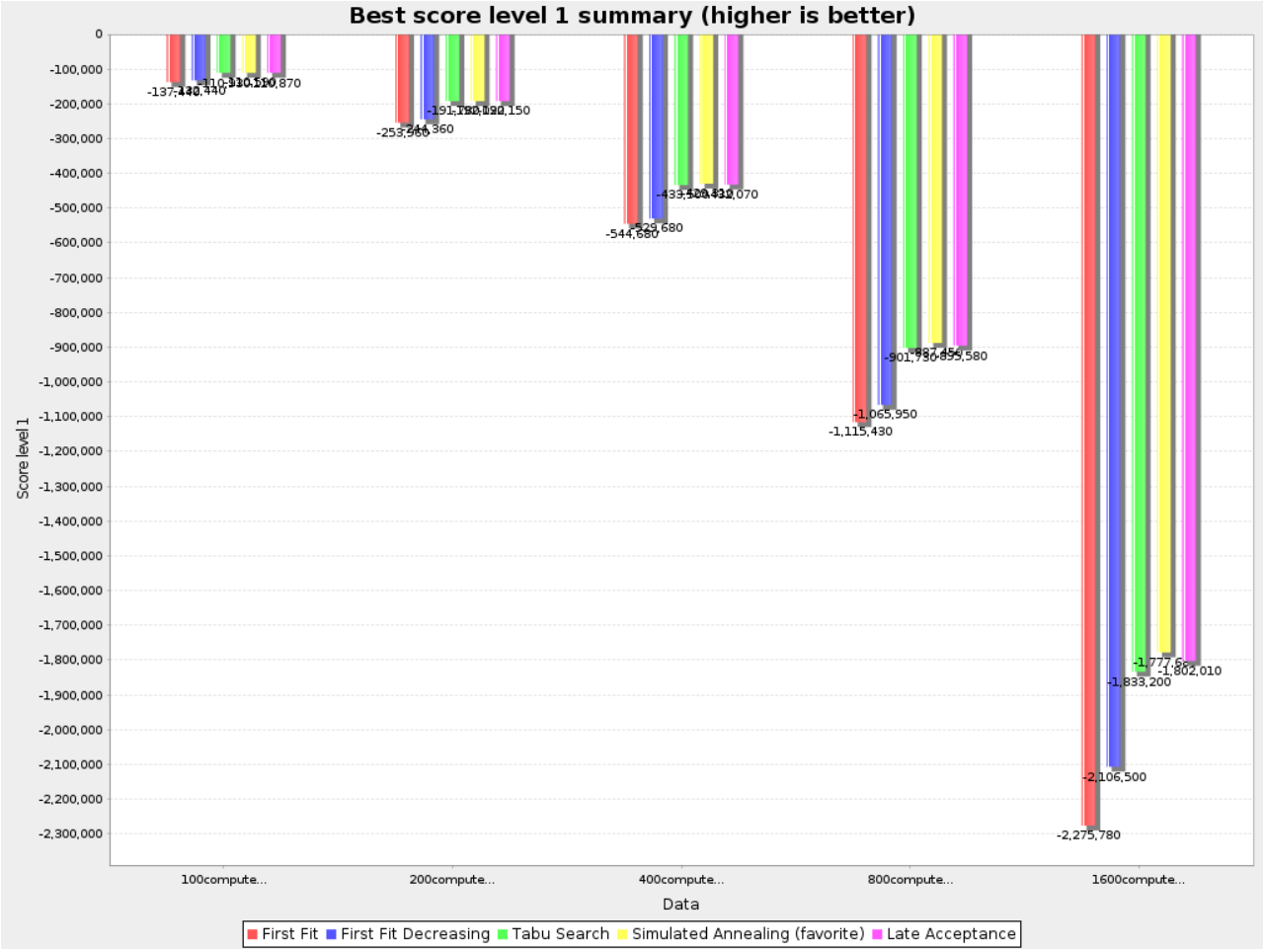
```
<localSearch>
  <acceptor>
    <!-- Tweaked value -->
    <simulatedAnnealingStartingTemperature>
        0hard/400soft
    </simulatedAnnealingStartingTemperature>
  </acceptor>
  <forager>
    <!-- Typical value -->
    <acceptedCountLimit>4</acceptedCountLimit>
  </forager>
</localSearch>
```

Late acceptance

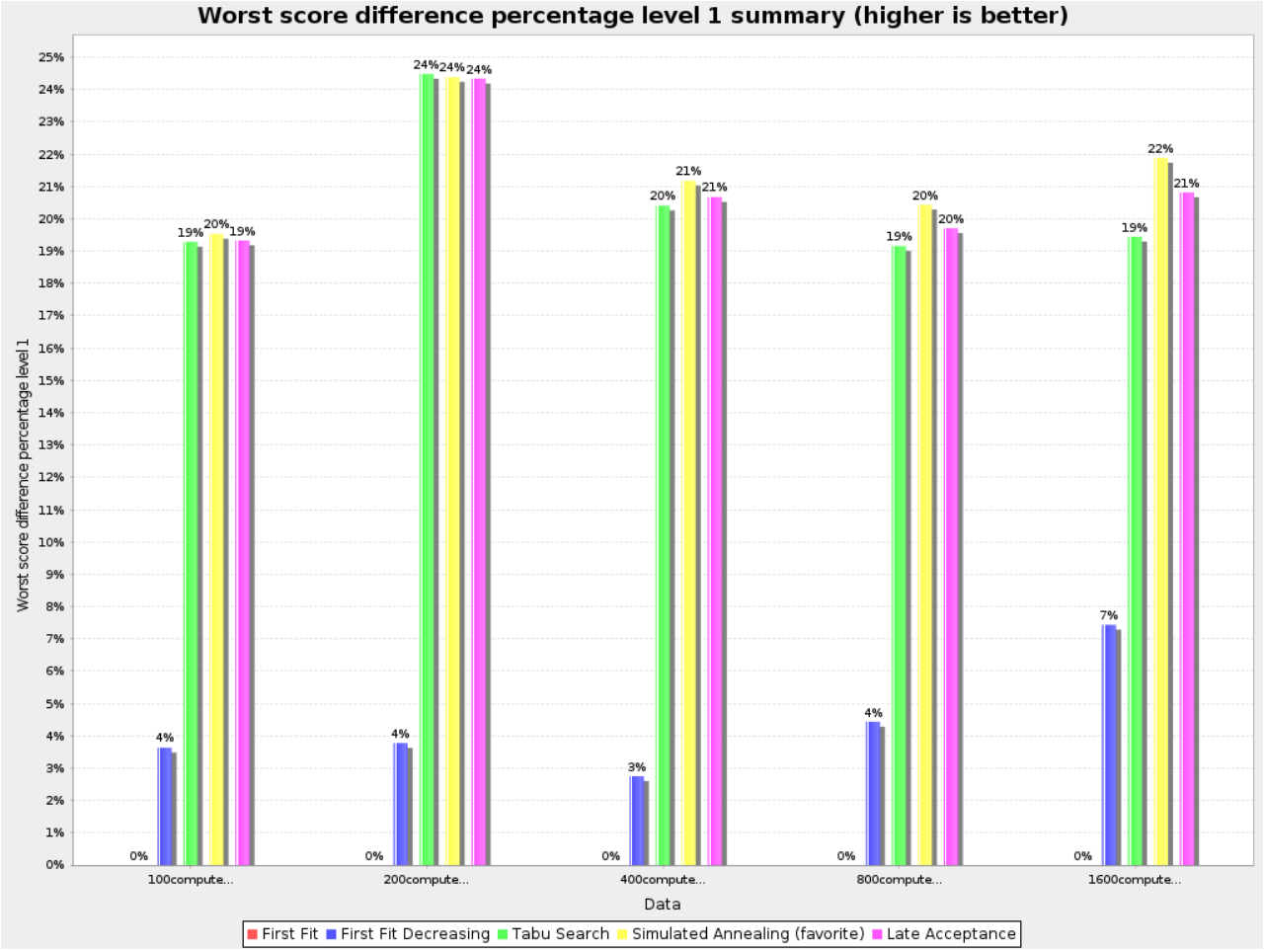
Late Acceptance config

```
<localSearch>
  <acceptor>
    <!-- Typical standard value -->
    <lateAcceptanceSize>400</lateAcceptanceSize>
  </acceptor>
  <forager>
    <!-- Typical value -->
    <acceptedCountLimit>4</acceptedCountLimit>
  </forager>
</localSearch>
```

Local Search results



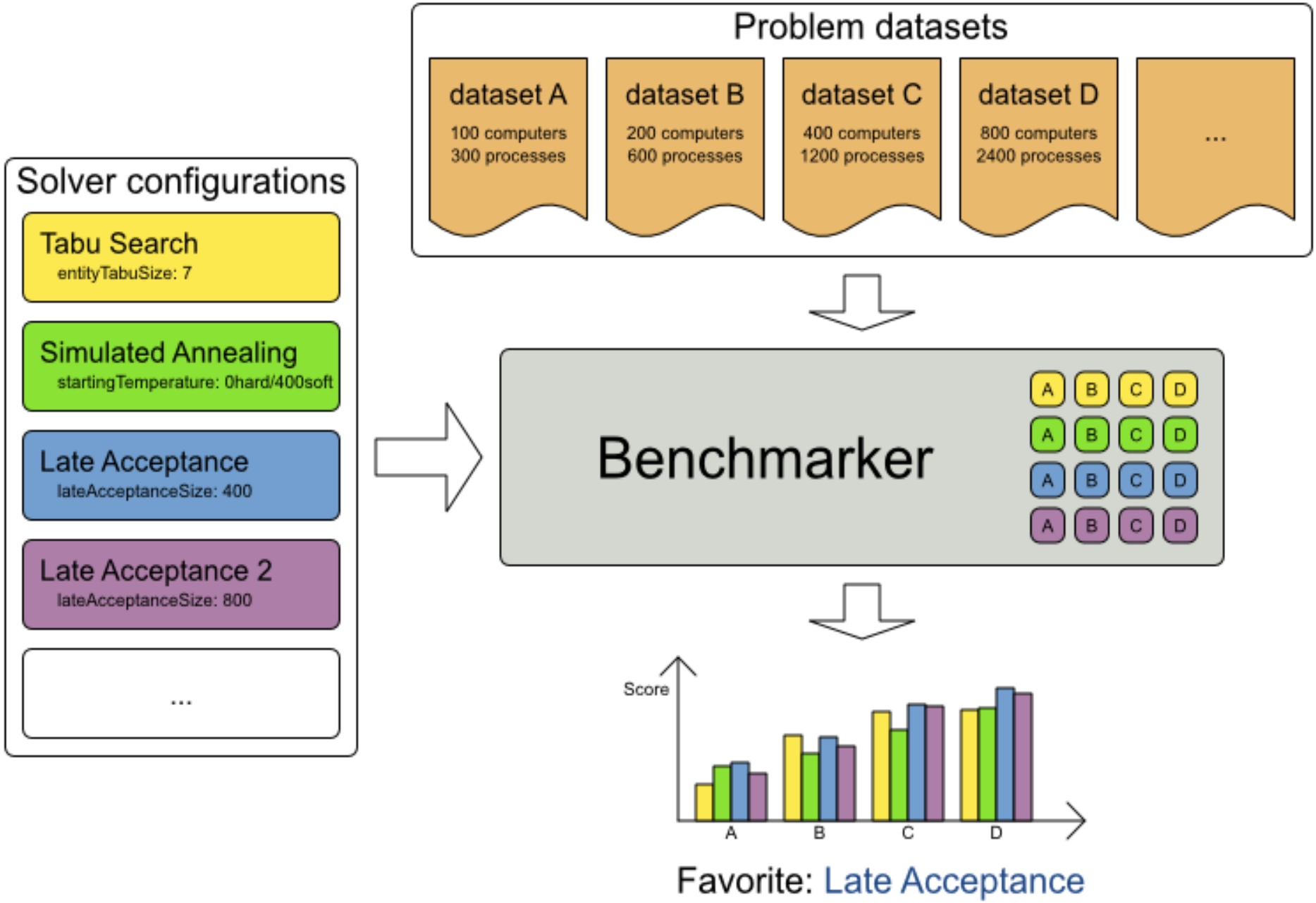
Cost ($) reduction



Optimization algorithms

- Exhaustive Search
  - Brute Force
  - Branch And Bound
- Construction Heuristics
  - First Fit (Decreasing)
  - Weakest/Strongest Fit (Decreasing)
  - Cheapest Insertion
- Local Search
  - Hill Climbing
  - Tabu Search
  - Strategic Oscillation Tabu Search
  - Simulated Annealing
  - Late Acceptance
  - Step Counting Hill Climbing

Benchmark overview



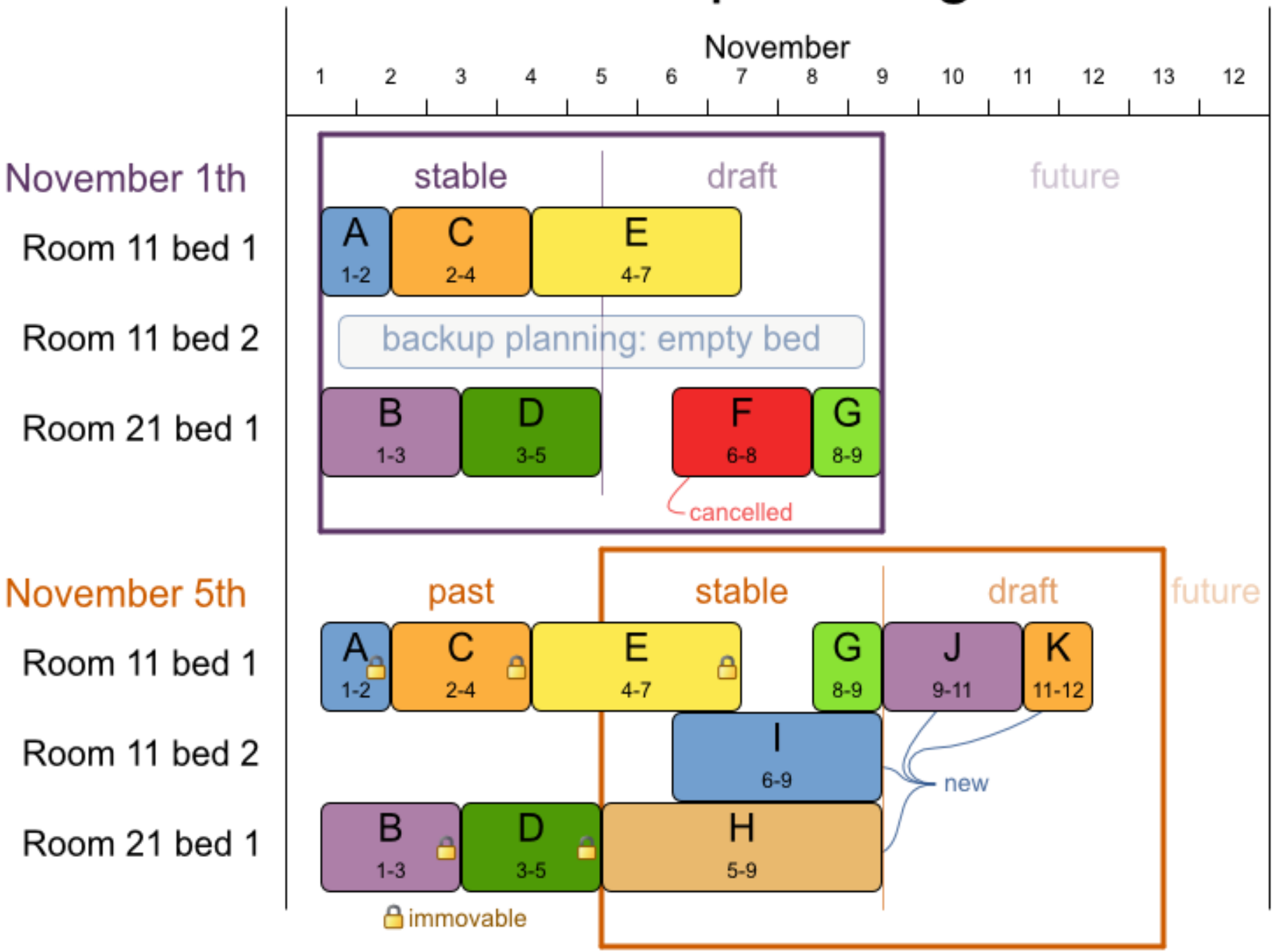
Favorite: Late Acceptance

Benchmark results

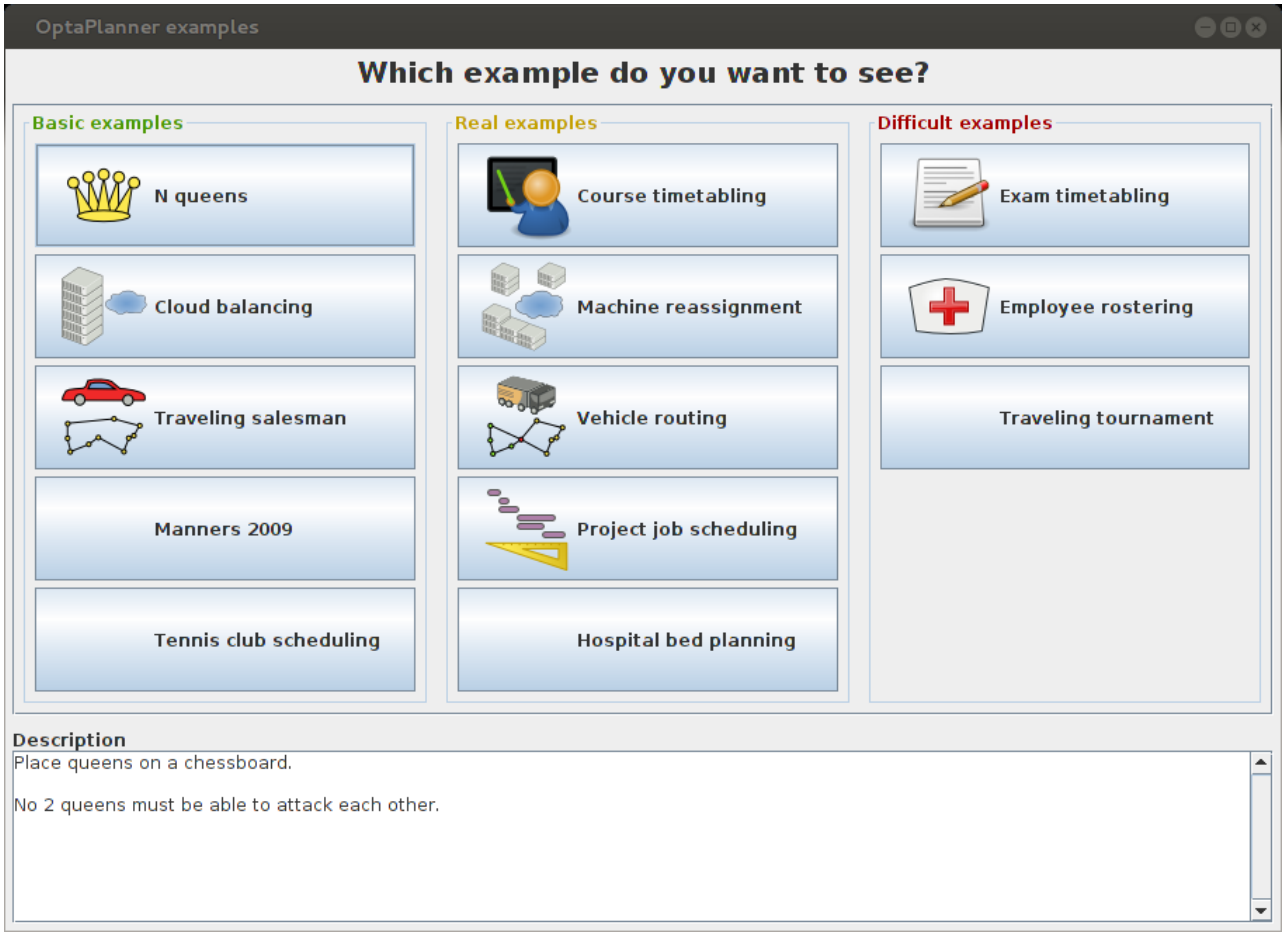
Demo

Repeated planning: continuous/real-time

Continuous planning



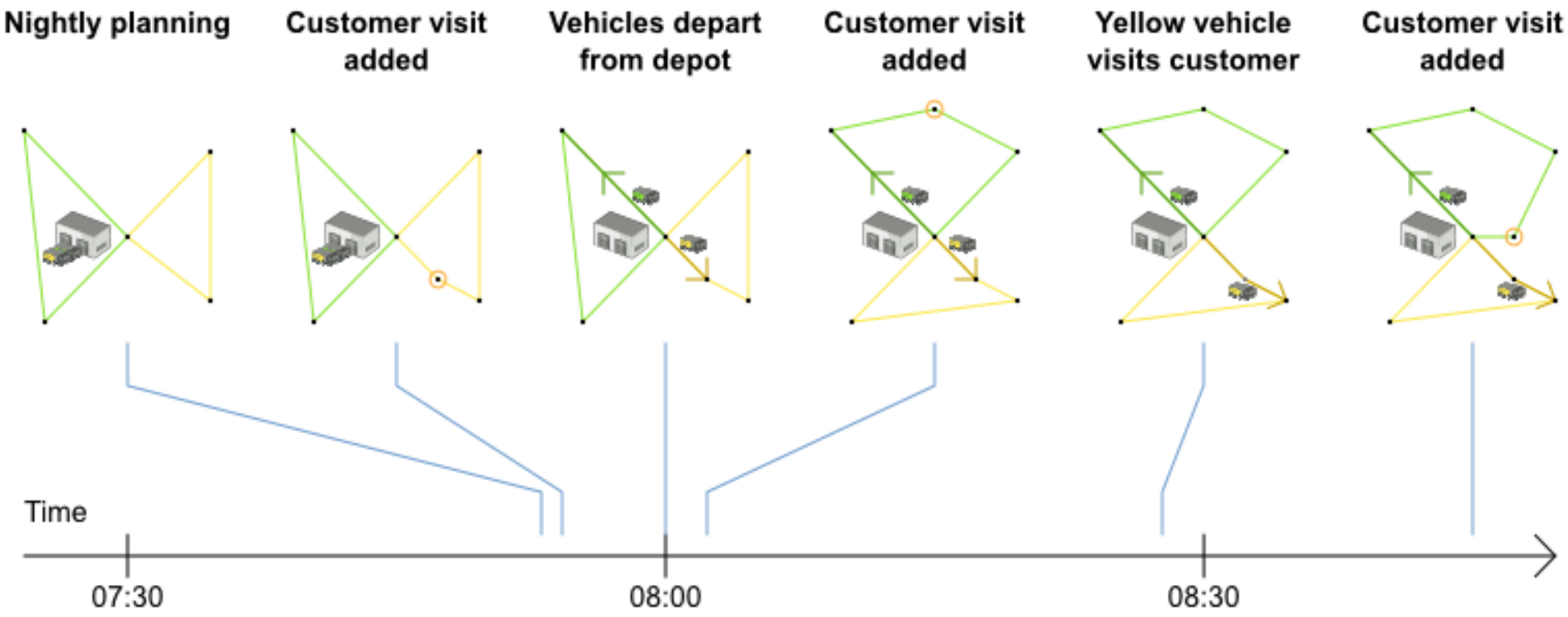
Continuous planning demo



Nurse rostering demo

Real-time planning

When the problem changes in real-time, the plan is adjusted in real-time.



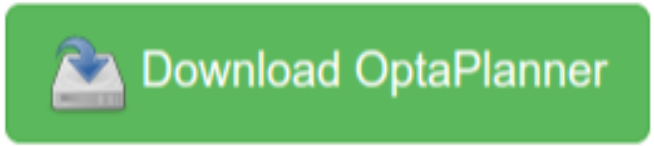
Summary

- OptaPlanner solves planning and scheduling problems
- Adding constraints: easy and scalable
- Switching/combining optimization algorithms: easy

Distribution zip

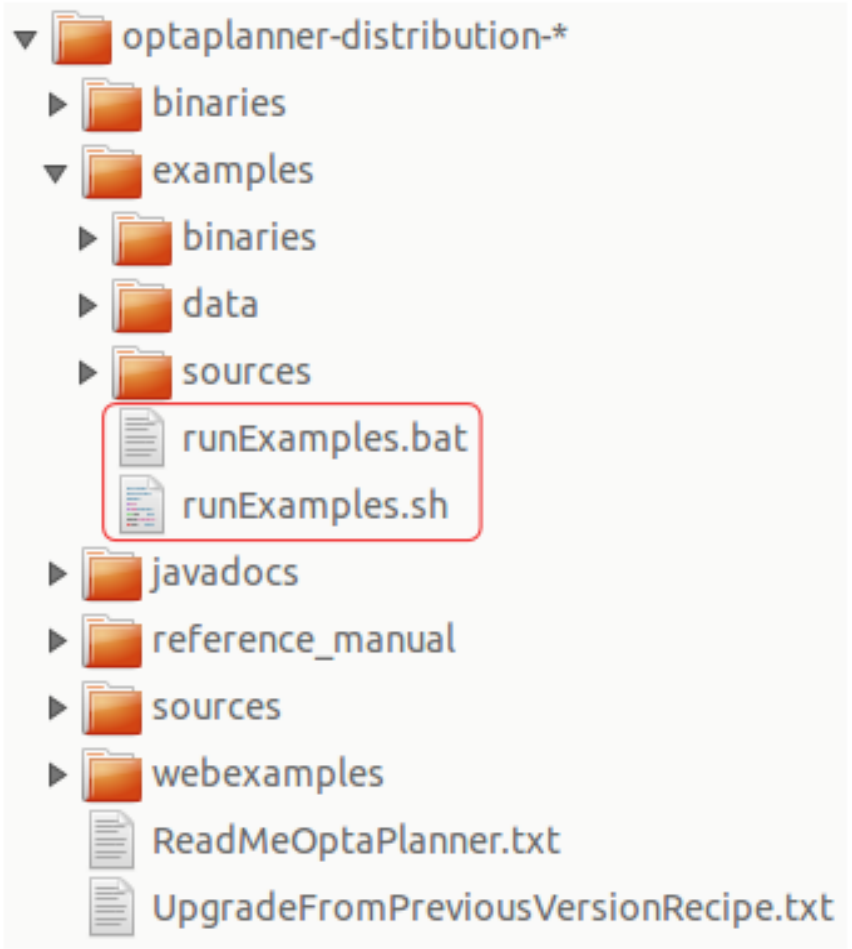
Running the examples locally

1. Surf to www.**optaplanner.org**

2. Click on 

3. Unzip optaplanner-distribution-*.zip

4. Open the directory `examples` and double click on `runExamples`



Build from source

- https://github.com/droolsjbpm/optaplanner (https://github.com/droolsjbpm/optaplanner)

```
$ git clone git@github.com:droolsjbpm/optaplanner.git
...
$ cd optaplanner
$ mvn clean install -DskipTests
...
$ cd optaplanner-examples
$ mvn exec:exec
...
```

Q & A

- OptaPlanner homepage
  - http://www.optaplanner.org (http://www.optaplanner.org)
- Reference manual
  - http://www.optaplanner.org/learn/documentation.html (http://www.optaplanner.org/learn/documentation.html)
- Download/fork this presentation
  - http://www.optaplanner.org/learn/slides.html (http://www.optaplanner.org/learn/slides.html)

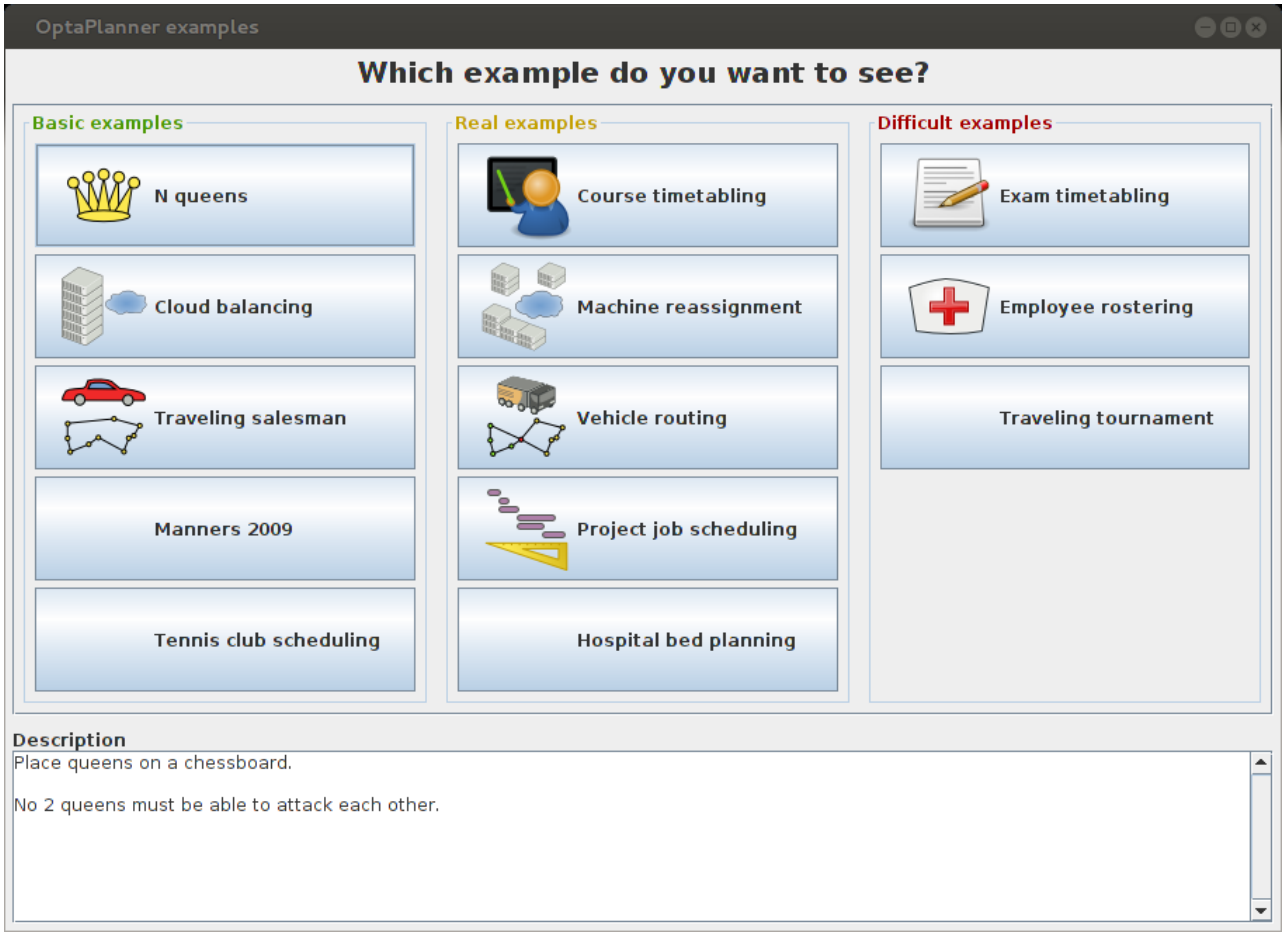
- What did you think of this presentation?
  - Twitter: @geoffreydesmet (http://twitter.com/geoffreydesmet)
  - Google+: +GeoffreyDeSmet (https://plus.google.com/+GeoffreyDeSmet)

Introduction to heuristics and metaheuristics for business resource optimization

by Geoffrey De Smet

OptaPlanner lead

N Queens demo

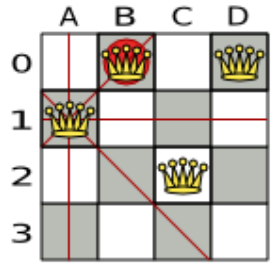


demo

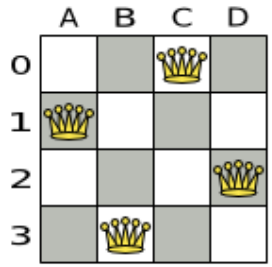
Simplified problem

N Queens

- Place n queens on a n sizes chessboard
- No 2 queens can attach each other



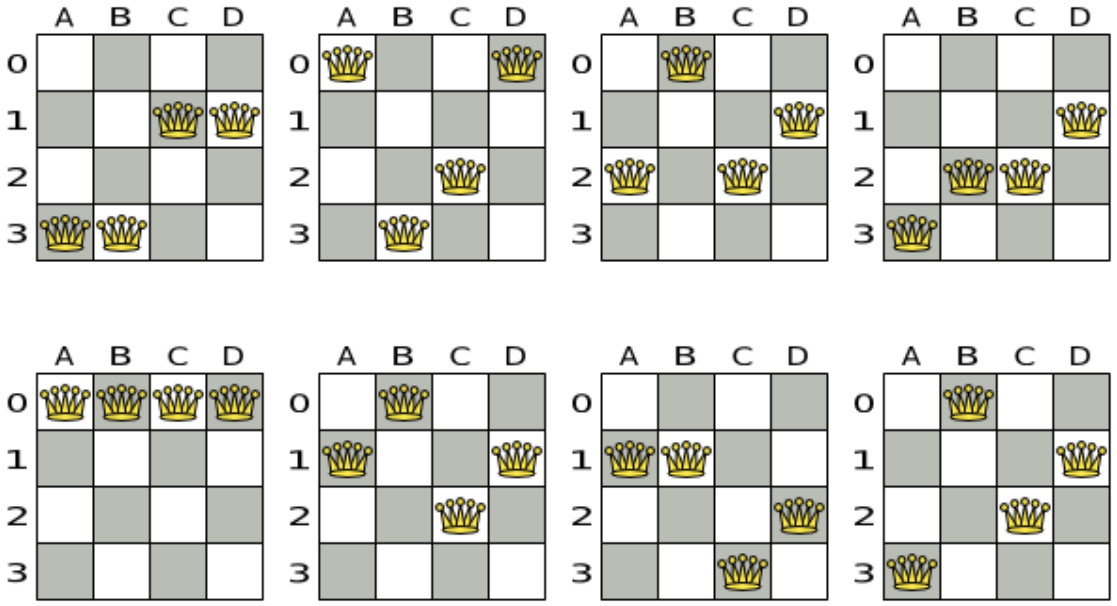
Bad



Good

- Imperfect example
  - Not NP-complete (shortcuts exist)

What solution is better?



- Need for **objective scoring**
- Better score ⇔ better solution
- Highest score ⇔ optimal solution

Positive constraints

Maximize apples

Maximize 🍏

⇒ 🍏 = 1

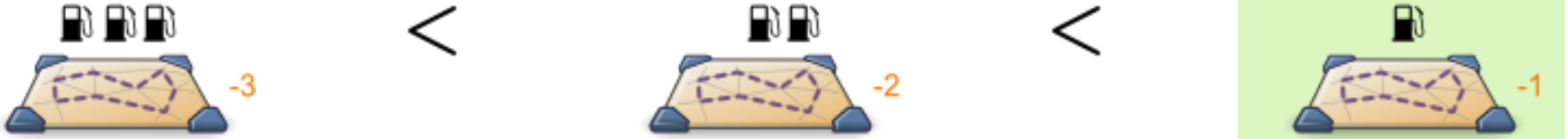
Optimal solution

Negative constraints

Minimize fuel usage

Minimize ⛽

⇒ ⛽ = -1



Optimal solution

Score weighting

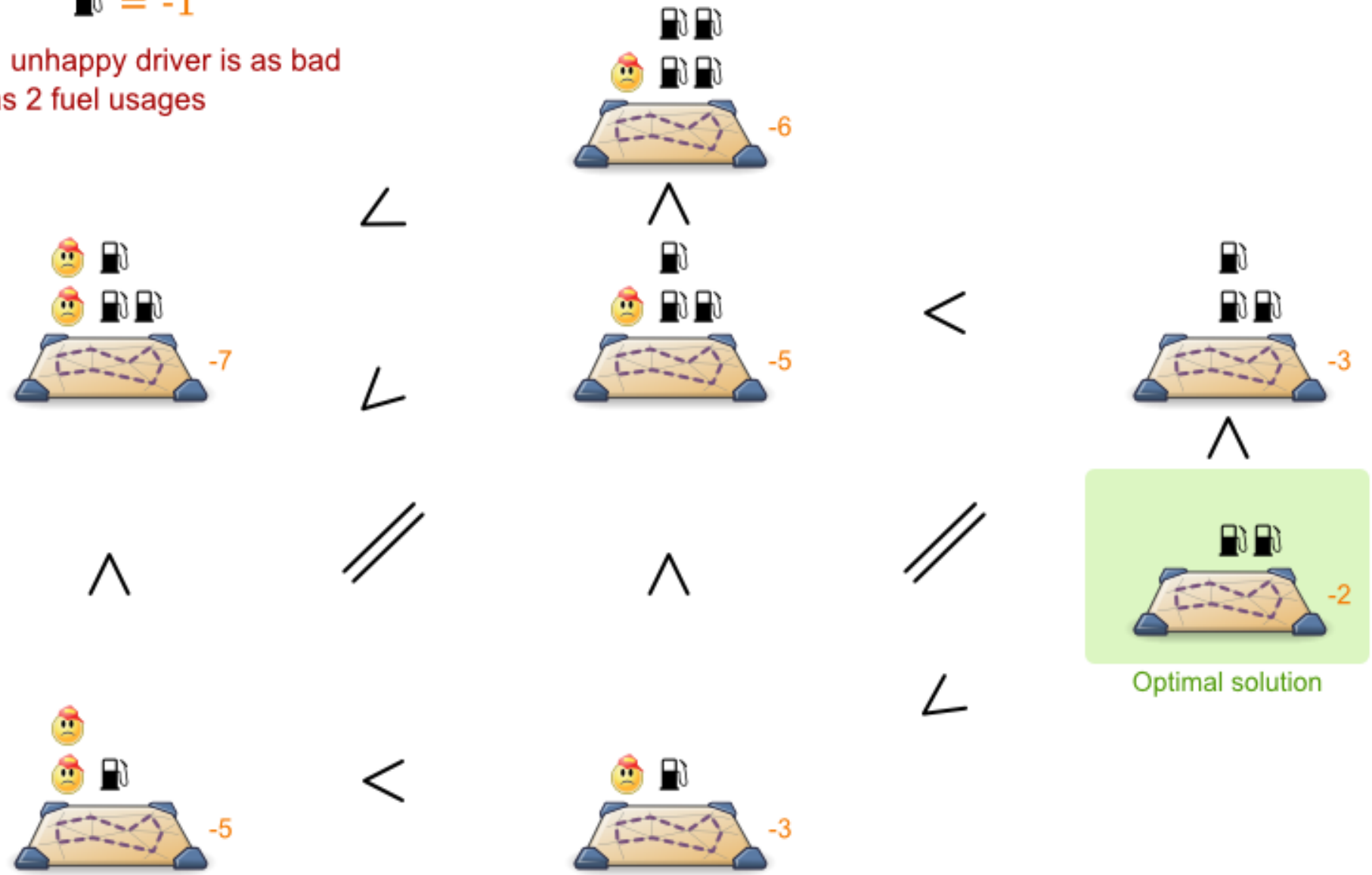
= 2

⇒ = -2

= -1

1 unhappy driver is as bad as 2 fuel usages

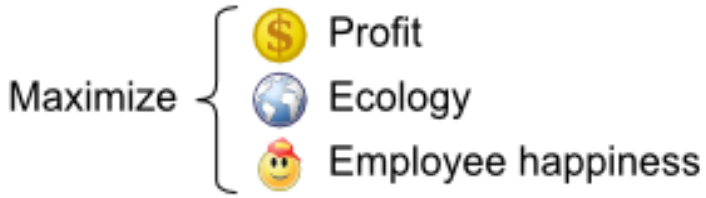
Minimize driver unhappiness
Minimize fuel usage

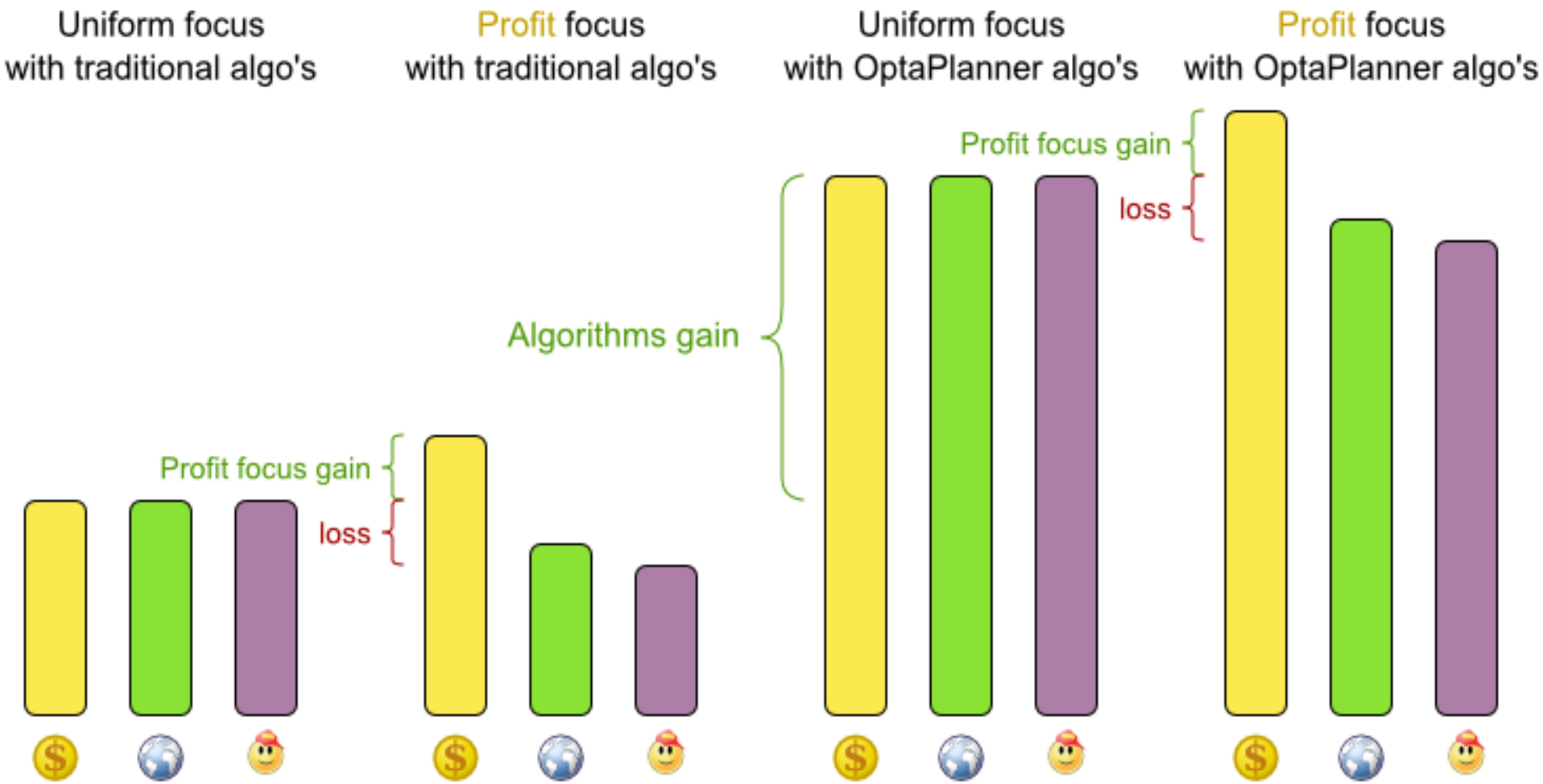


Optimal solution

Score tradeoff in perspective

Picking the right tradeoff is less important than using better algorithms.



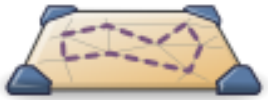


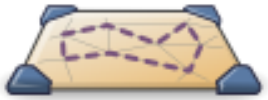
Score weight type

⛽ = 0.01 $

Use the correct number type

	Fuel usage	double double-precision 64-bit IEEE 754 floating point	BigDecimal arbitrary-precision signed decimal number
Vehicle X	⛽⛽⛽	0.03	0.03
Vehicle Y	⛽⛽⛽	0.03	0.03
Total		0.06	0.06 Highest score
Vehicle X	⛽	0.01	0.01
Vehicle Y	⛽⛽⛽⛽⛽	0.05	0.05
Total		0.060000000000000005 Highest score	0.06 Highest score



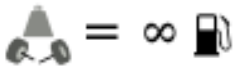


SimpleDoubleScore
score : double

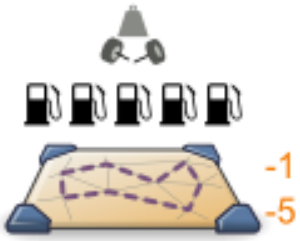
SimpleBigDecimalScore
score : BigDecimal

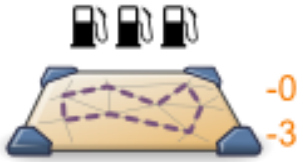
Score levels

First minimize overloaded truck axles,
then minimize fuel usage

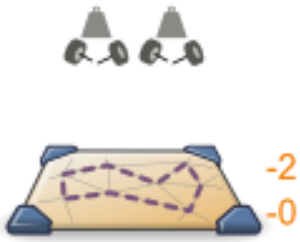


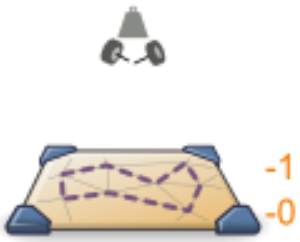
1 overloaded axle is worse
than any number of fuel usages





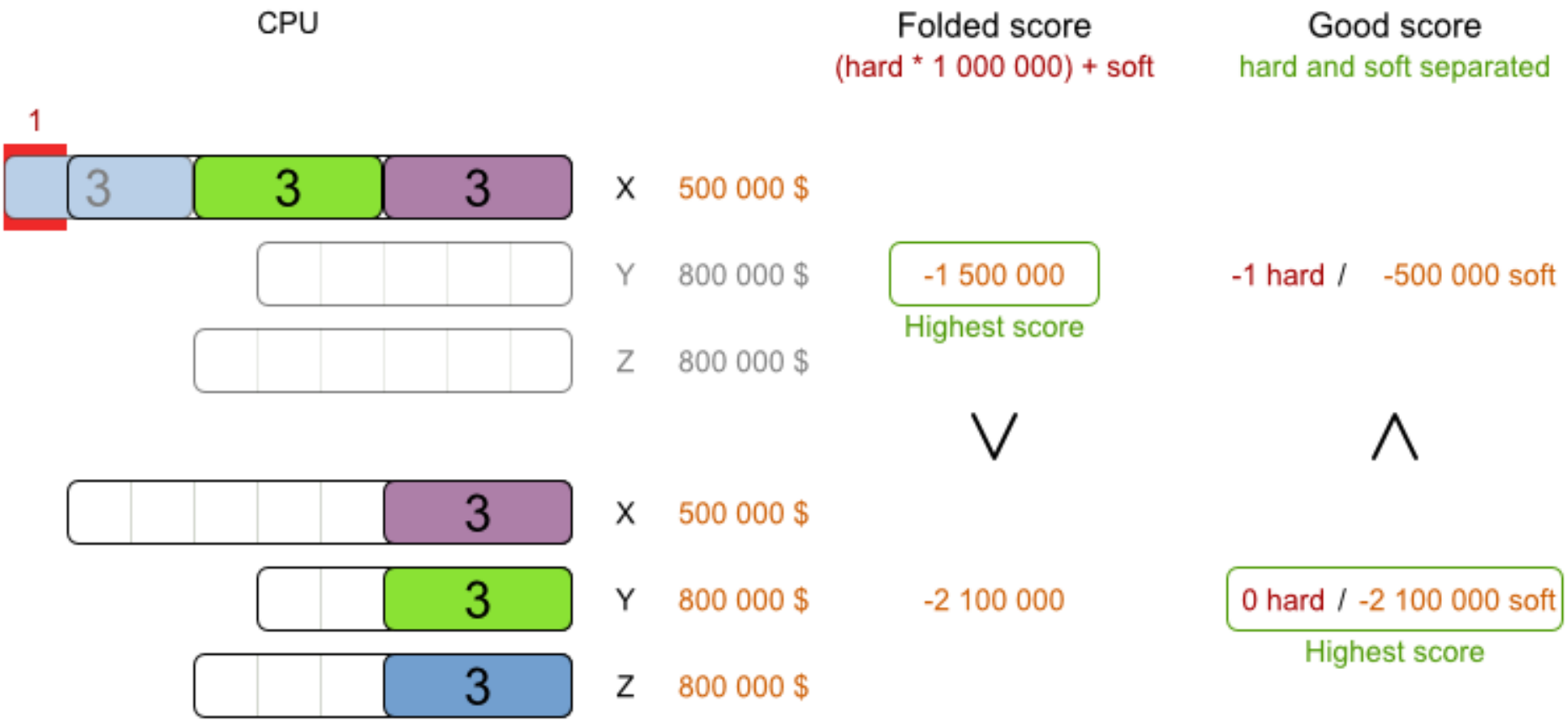
Optimal solution



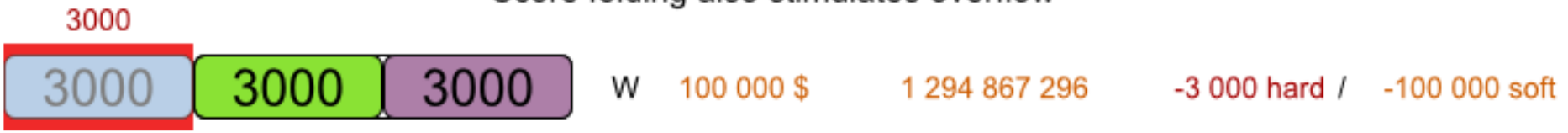


Score folding is broken

Don't mix score levels

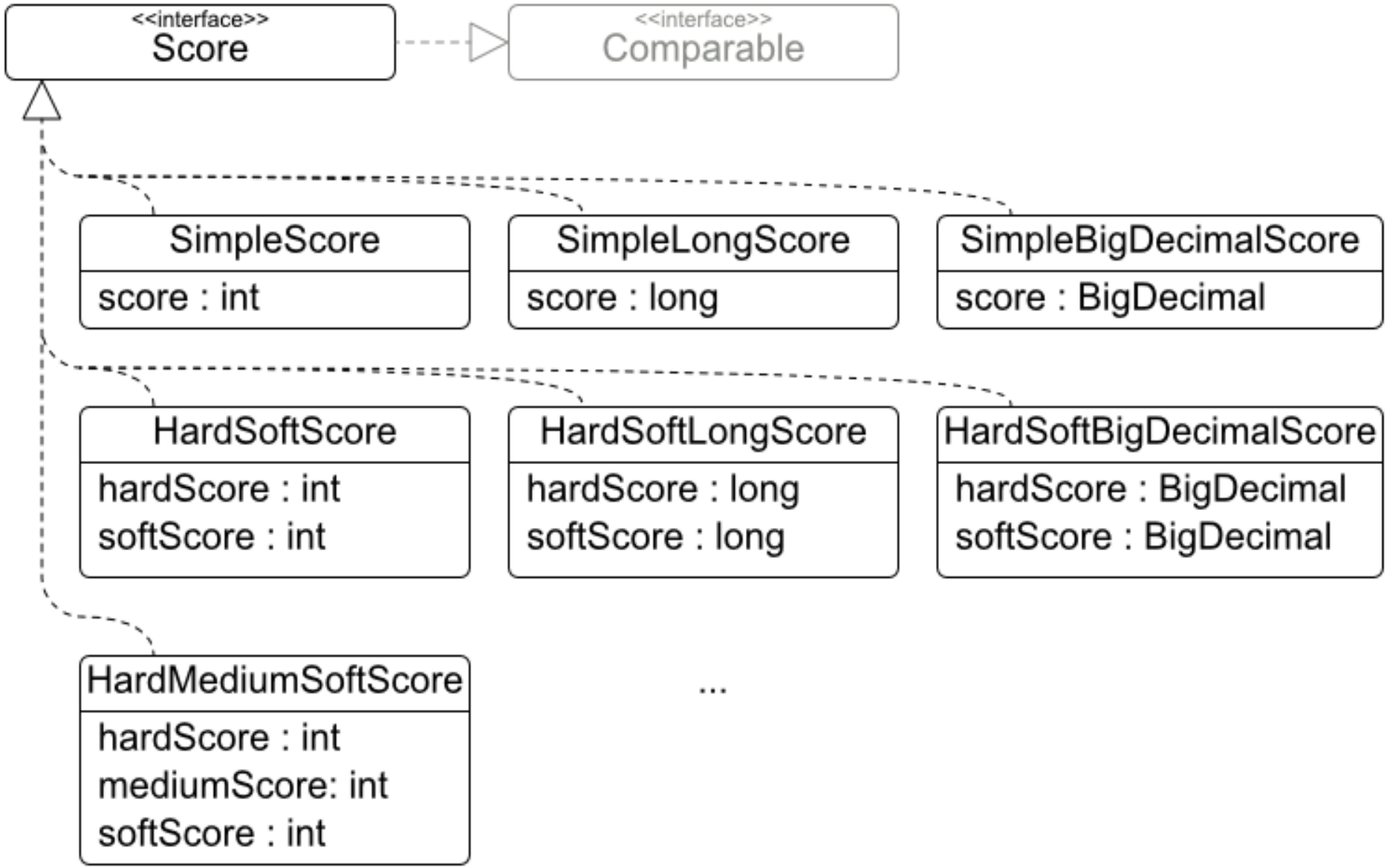


Score folding also stimulates overflow



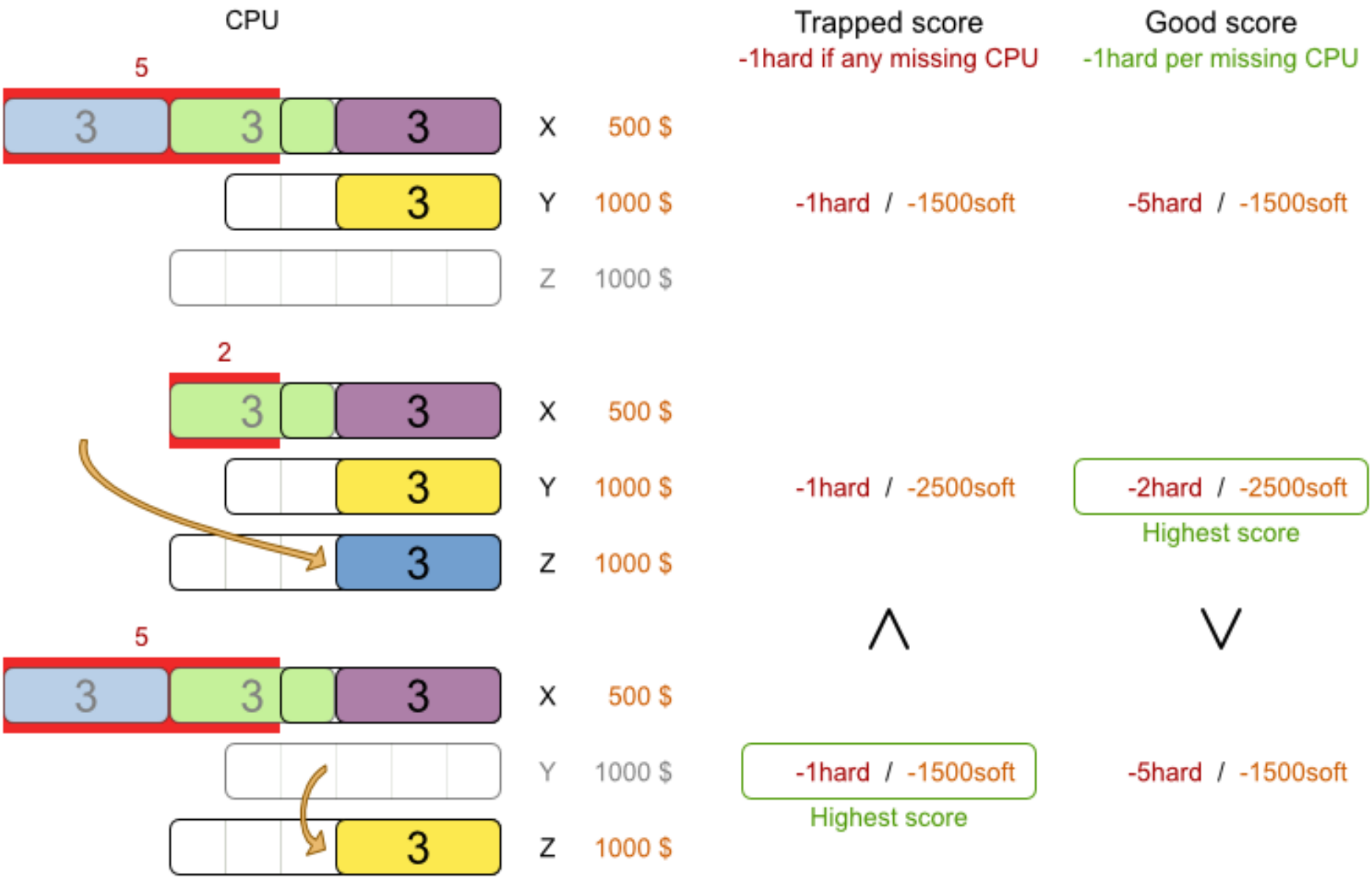
Score class diagram

Choose a Score implementation or write a custom one



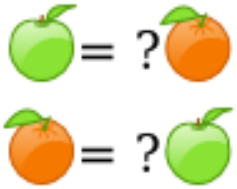
Score trap

There are degrees of infeasibility



Pareto optimization scoring

Maximize apples and oranges harvest
Don't compare apples and oranges



1 apple is worth an unknown number of oranges
1 orange is worth an unknown number of apples

Optimal solution A

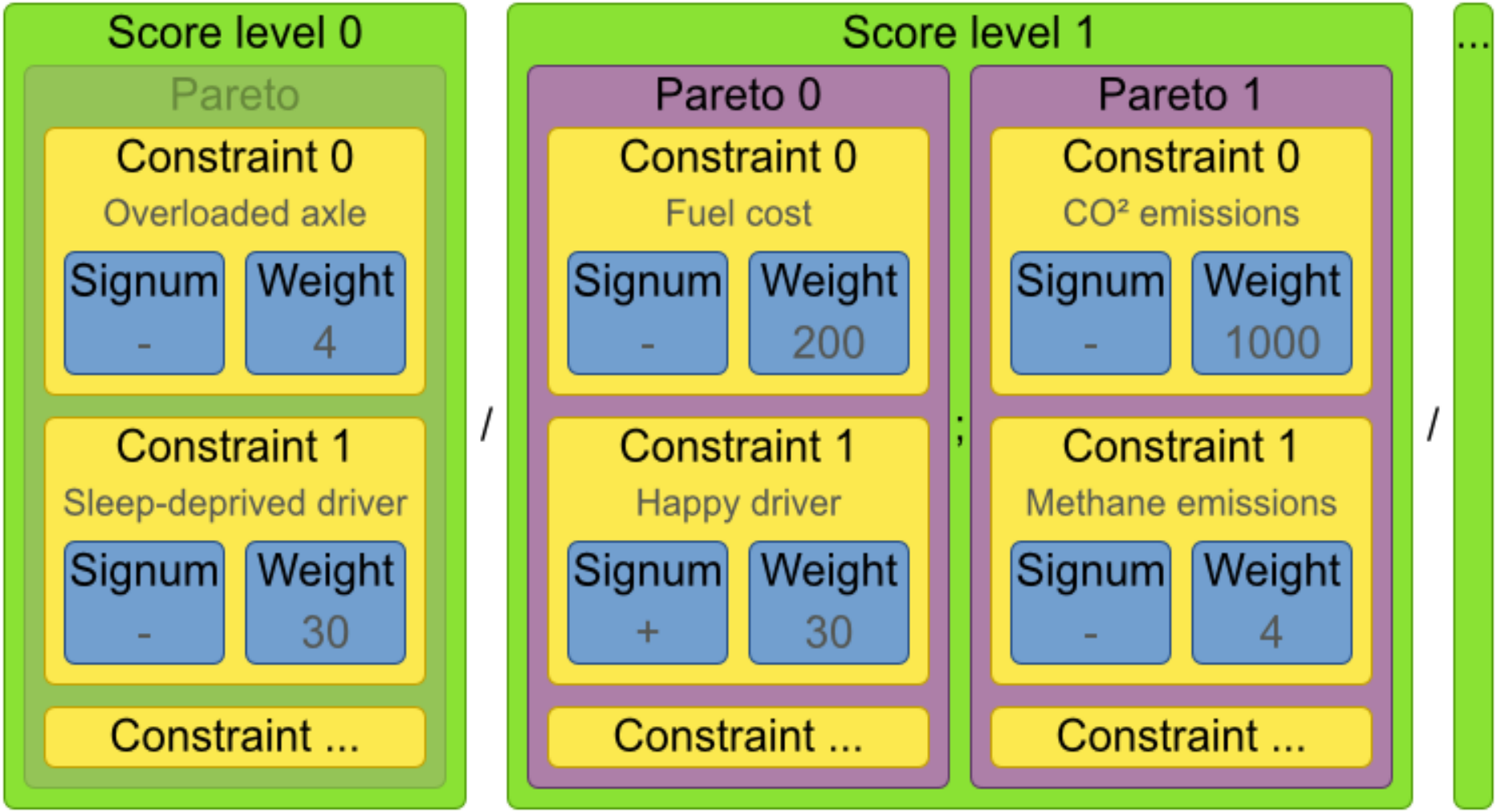
Optimal solution B

Pareto optimal

Only pareto optimal solutions
are shown to the user
User decides between A and B

Score composition

How are the score techniques combined?

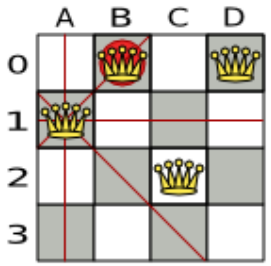


-34 / -170 ; -1004 / ...

Score for 1 solution

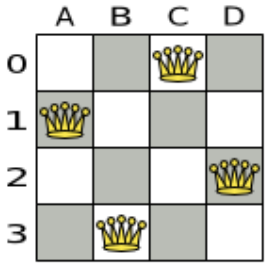
N Queens

- Hard constraints:
  - -1 for every pair of conflicting queens
- Soft constraints:
  - None



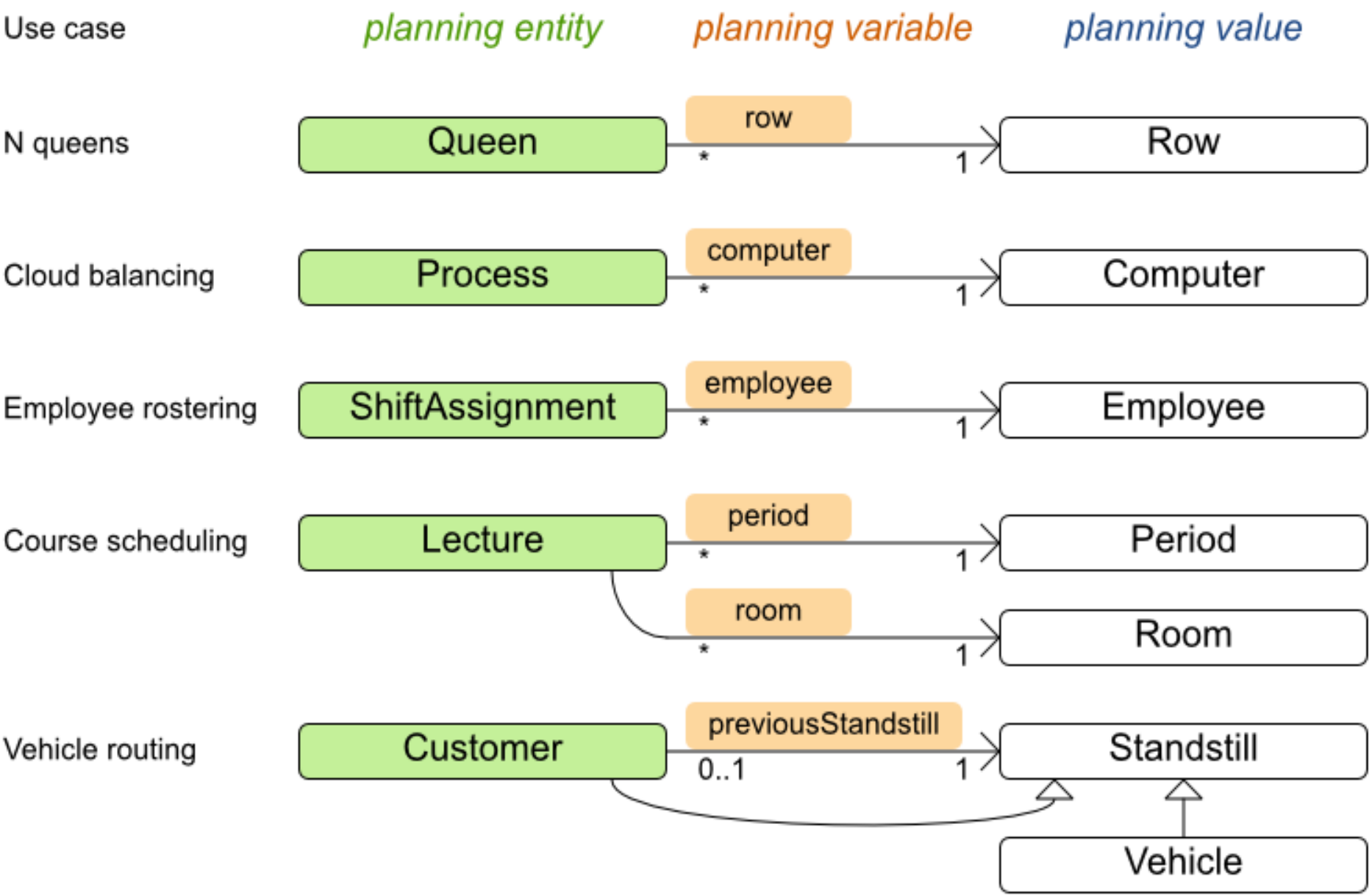
Score = -2

Conflicts: A-B, B-D

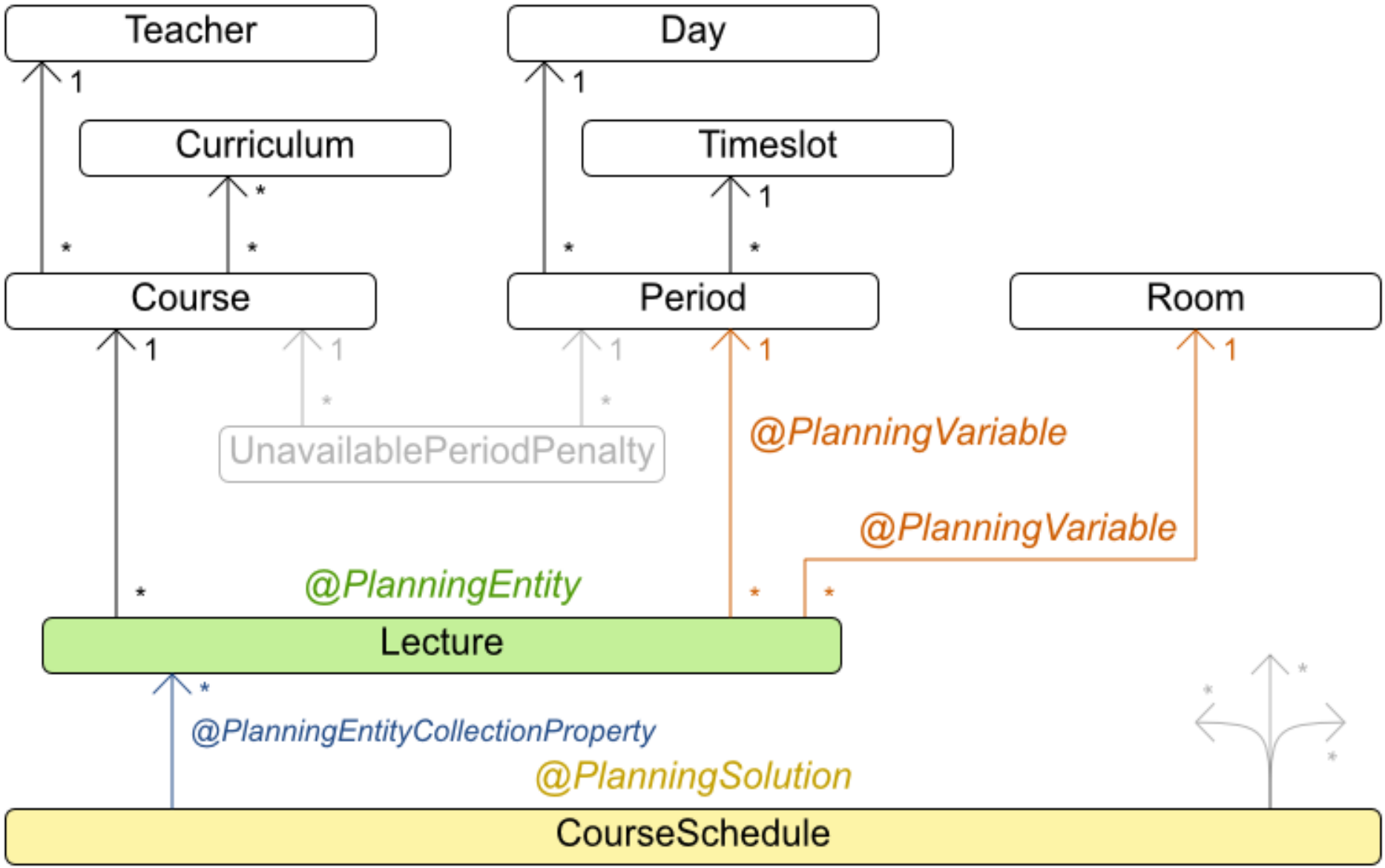


Score = 0

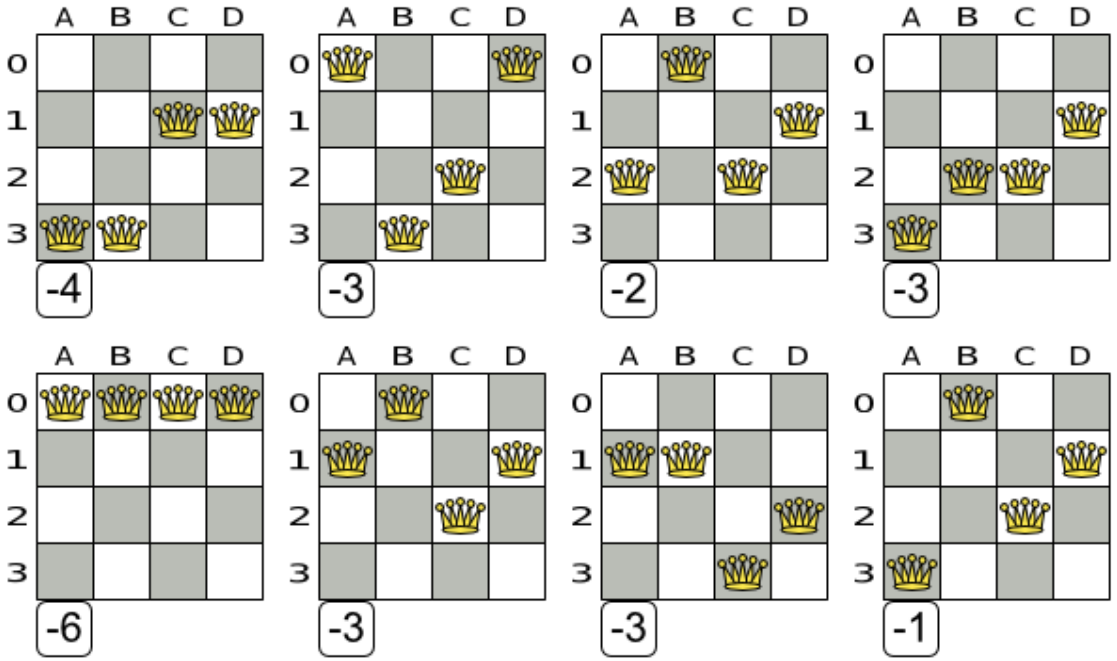
Entity, variable and value examples



Curriculum course class diagram

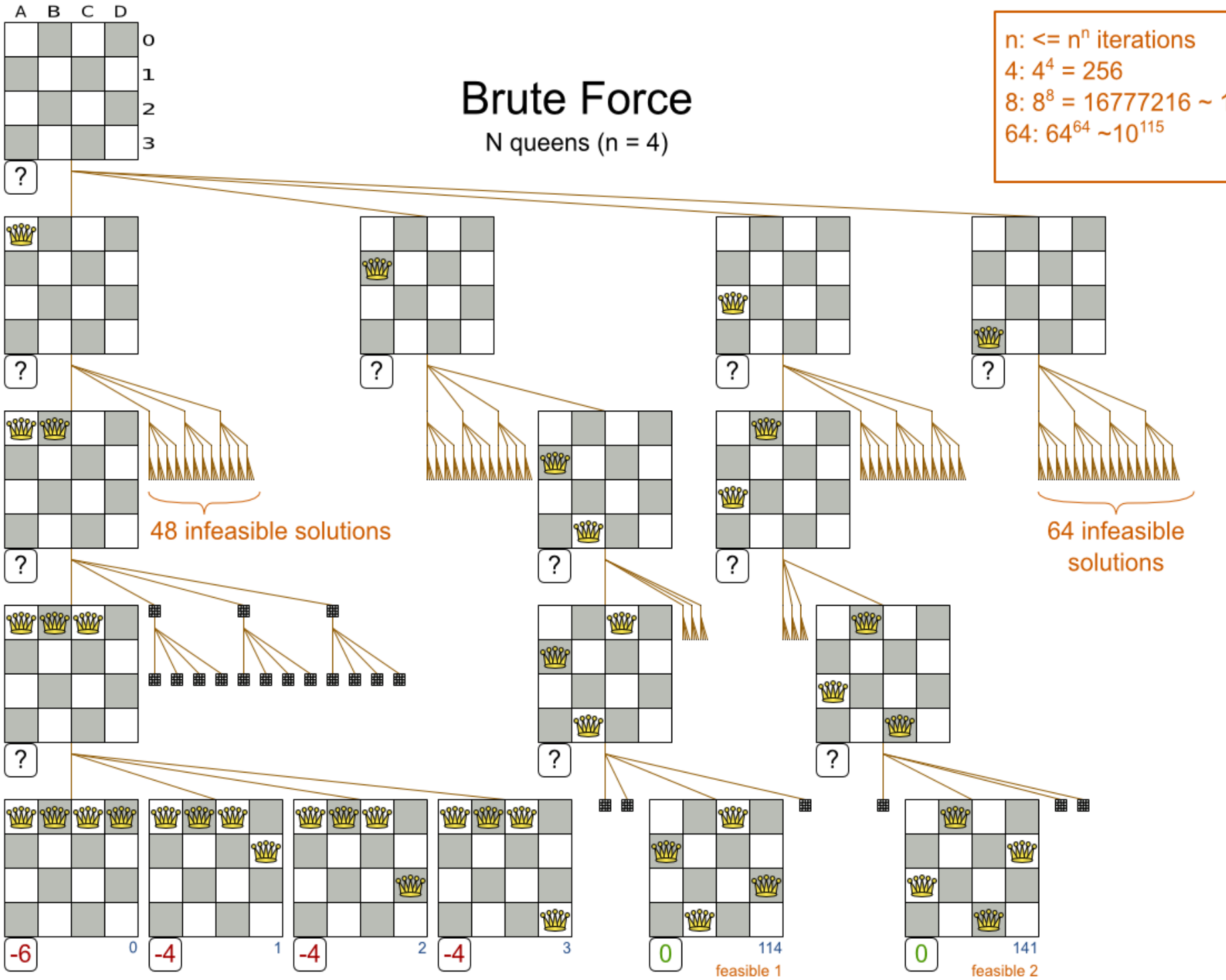


How do we find the best solution?

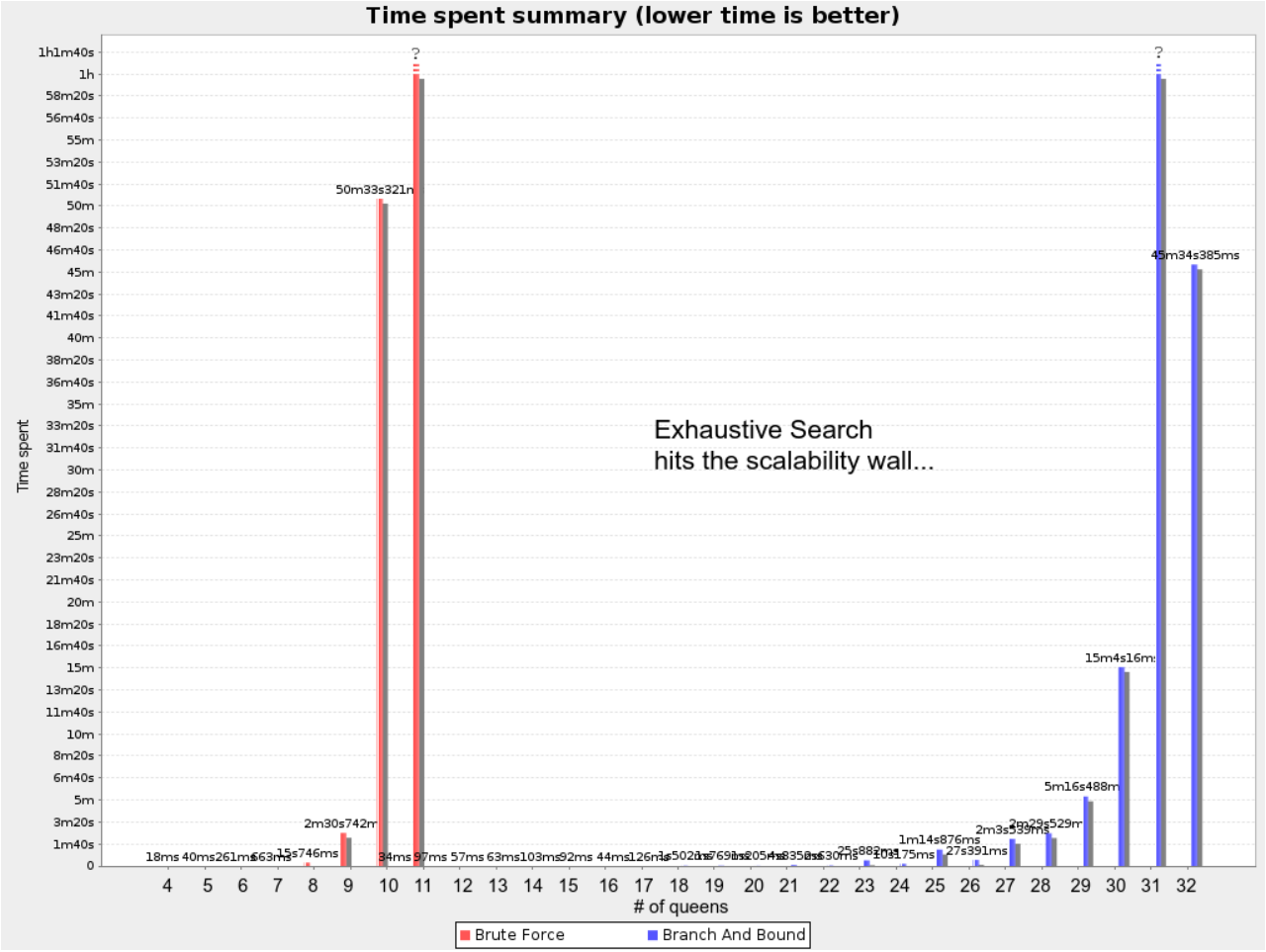


- Need for **optimization algorithms**
- Best solution in available time

Brute Force


A B C D
0
1
2
3
?
Brute Force
N queens (n = 4)
n: <= n^n iterations
4: 4^4 = 256
8: 8^8 = 16777216 ~ 10^7
64: 64^64 ~10^115
48 infeasible solutions
64 infeasible solutions
-6
0
-4
1
-4
2
-4
3
0
114
feasible 1
0
141
feasible 2

Brute Force scalability



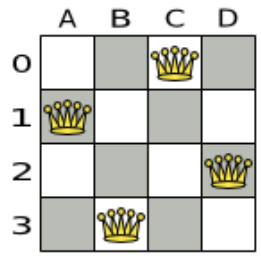
8 queens = 15.7 seconds

9 queens = 2.5 minutes (times 10)

10 queens = 0.83 hours (times 20)

How many combinations for 100 queens?

- 1 queen per column
- 100 queens ⇒ 100 variables
- 100 rows ⇒ 100 values per variable



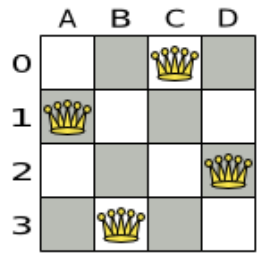
Source: NASA (wikipedia)

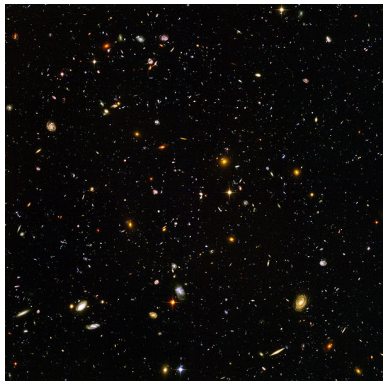
> humans?

7 000 000 000

How many combinations for 100 queens?

- 1 queen per column
- 100 queens ⇒ 100 variables
- 100 rows ⇒ 100 values per variable



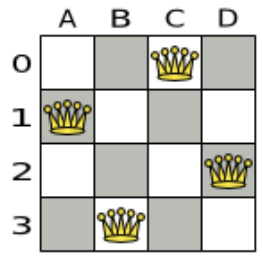


Source: NASA and ESA (wikipedia)

> minimum atoms
in the observable universe?
10^{80}

How many combinations for 100 queens?

- 1 queen per column
- 100 queens ⇒ 100 variables
- 100 rows ⇒ 100 values per variable

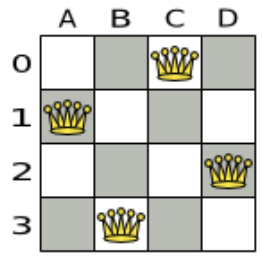


$$100^{100} = 10^{200}$$

```
1 0000000000 0000000000 0000000000 0000000000 0000000000
  0000000000 0000000000 0000000000 0000000000 0000000000
  0000000000 0000000000 0000000000 0000000000 0000000000
  0000000000 0000000000 0000000000 0000000000 0000000000
```

How many combinations for n queens?

- 1 queen per column
- n queens ⇒ n variables
- n rows ⇒ n values per variable



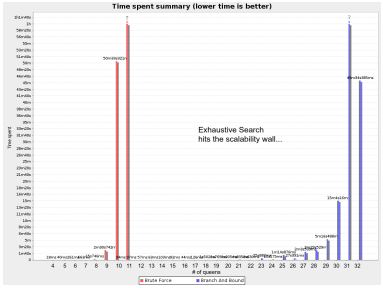
$$n^n$$

$$|valueSet|^{|variableSet|}$$

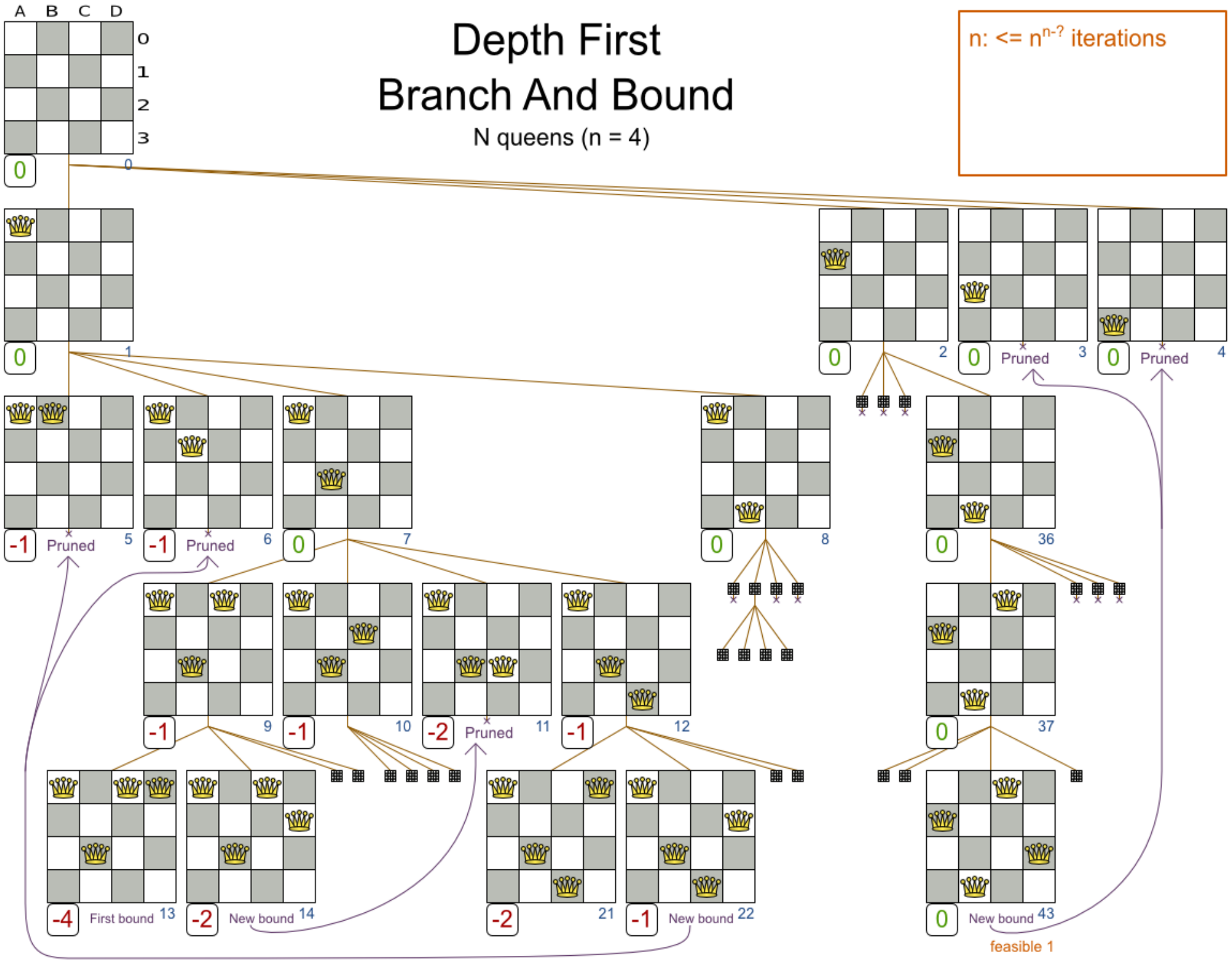
How long?

Presume 10^9 scores/ms ⇒ 10^{20} scores/year

Queens	Combinations	Calculation time
100	$100^{100} = 10^{200}$	10^{180} years
1000	$1000^{1000} = 10^{3000}$	10^{2980} years
10000	$10000^{10000} = 10^{40000}$	10^{39980} years



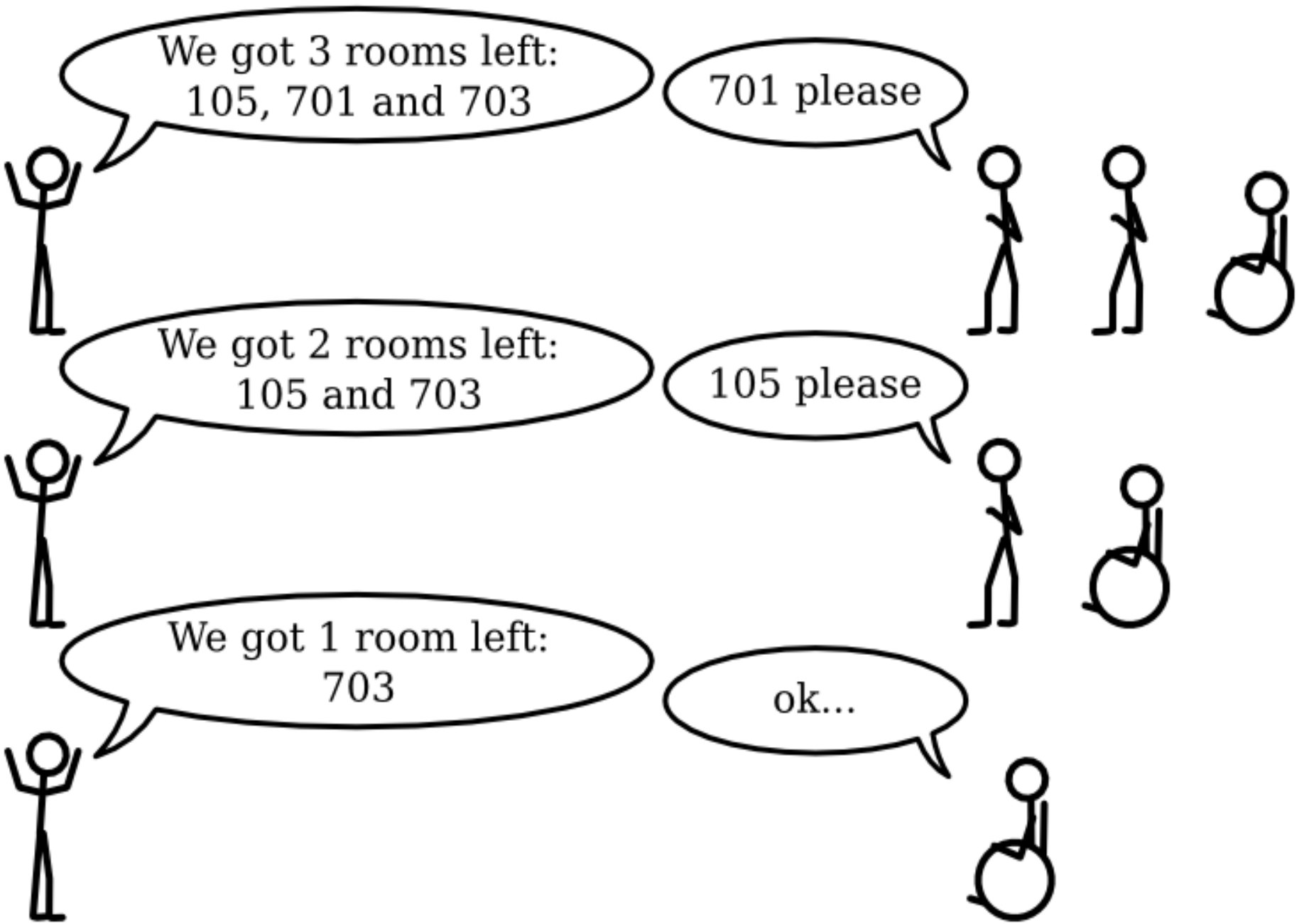
Moore's law == a drop in the ocean


Depth First
Branch And Bound
N queens (n = 4)
n: <= n^n-? iterations
A B C D
0
1
2
3
Pruned
First bound
New bound
feasible 1

Exhaustive Search doesn't scale

- Branches explode exponentially
- Not enough CPU
- Not enough memory

First Fit

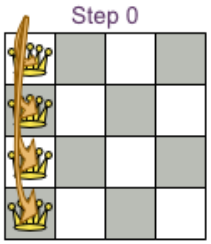

We got 3 rooms left:
105, 701 and 703
701 please
We got 2 rooms left:
105 and 703
105 please
We got 1 room left:
703
ok...

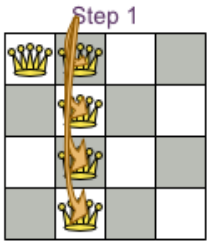
Construction heuristic: First fit

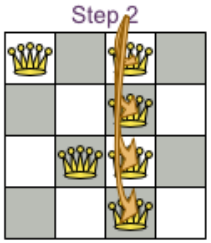
N queens (n = 4)

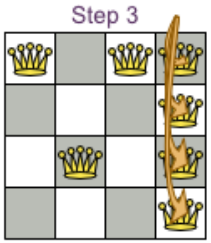
1 entity per step ordered arbitrary

n: <= n*n iterations
4: 4*4 = 16
8: 8*8 = 64
64: 64*64 = 4096

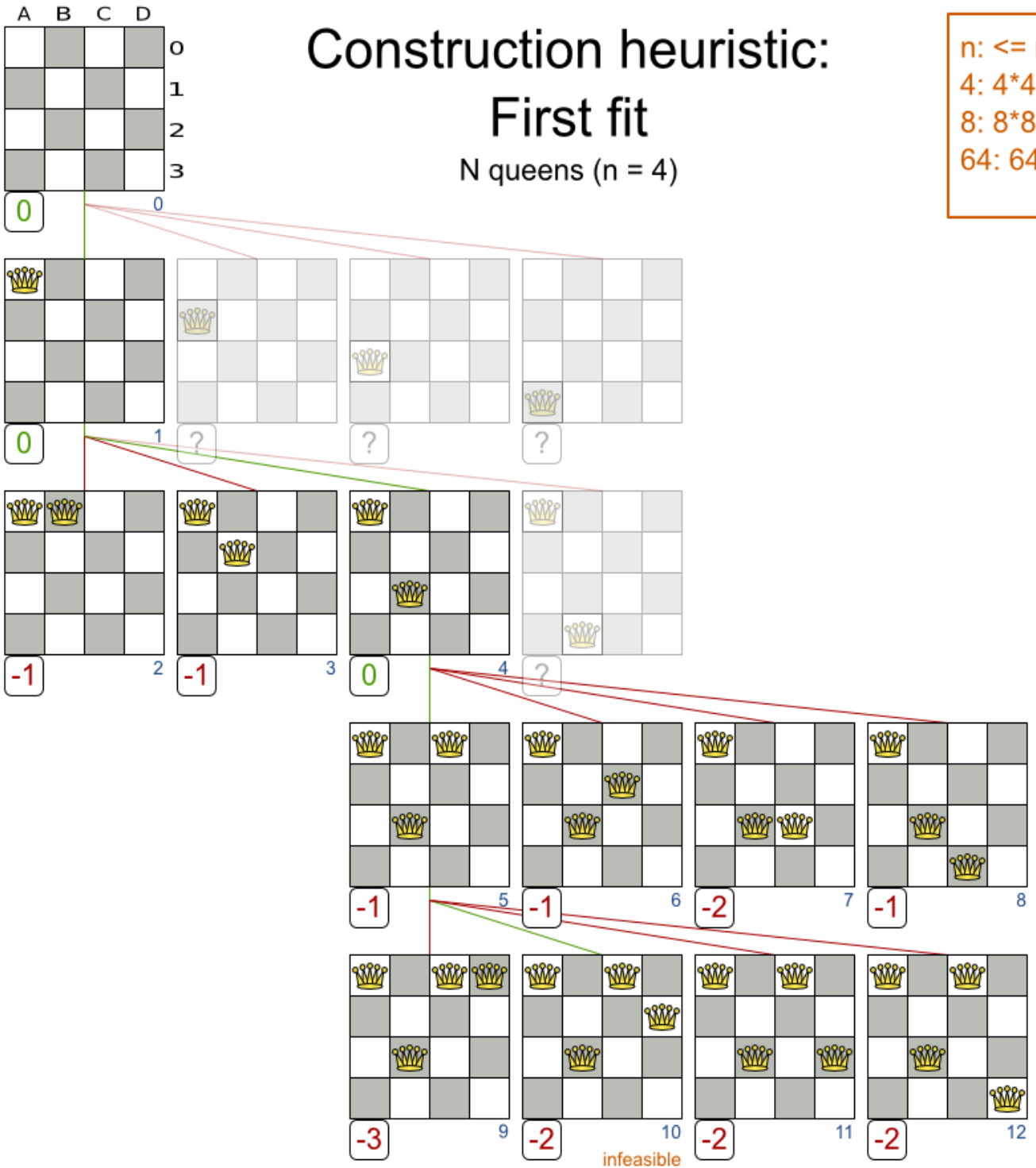




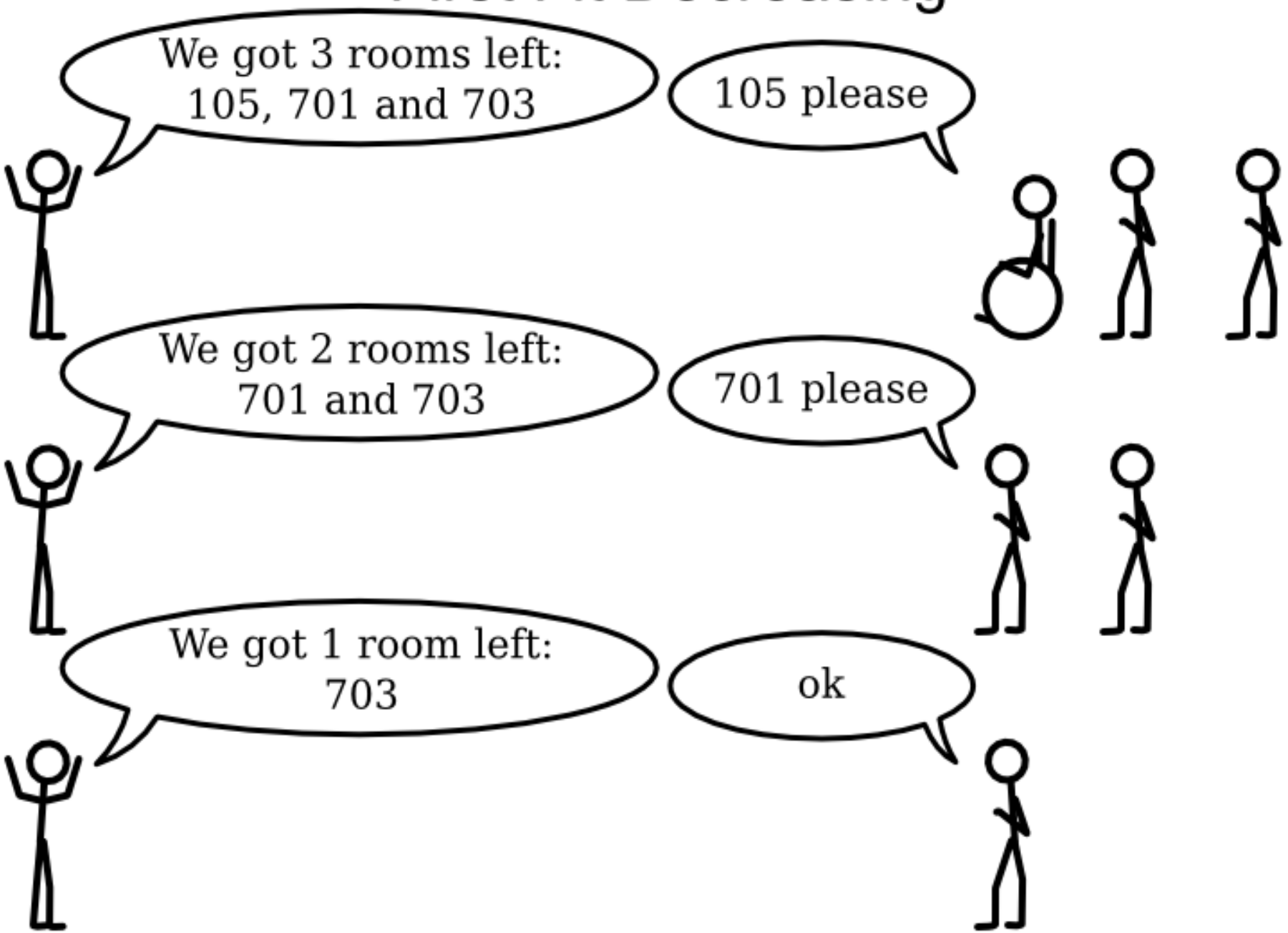




The end



First Fit Decreasing



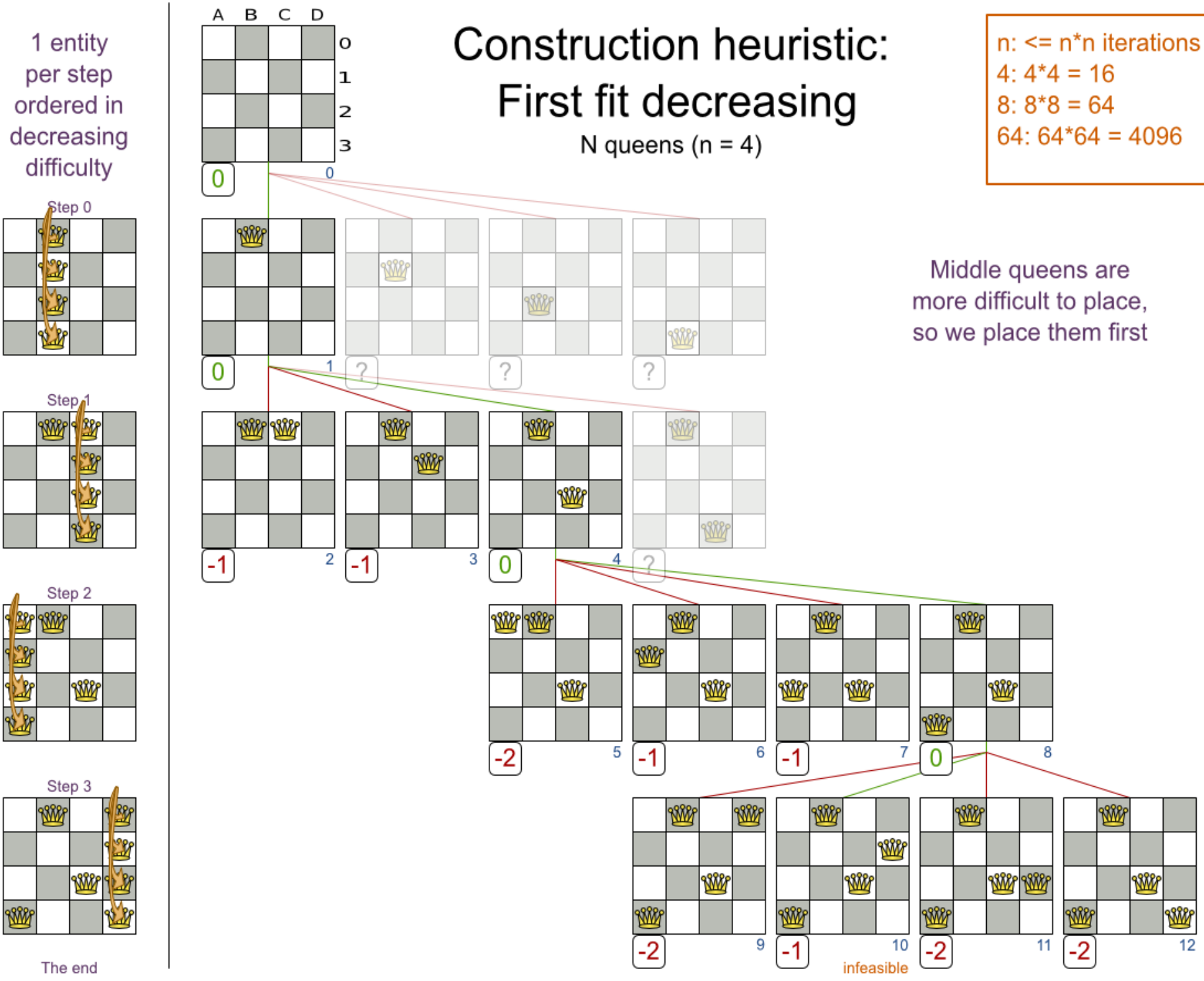
Construction heuristic: First fit decreasing

N queens (n = 4)

1 entity per step ordered in decreasing difficulty

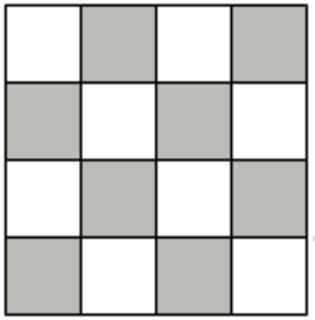
n: <= n*n iterations
4: 4*4 = 16
8: 8*8 = 64
64: 64*64 = 4096

Middle queens are more difficult to place, so we place them first



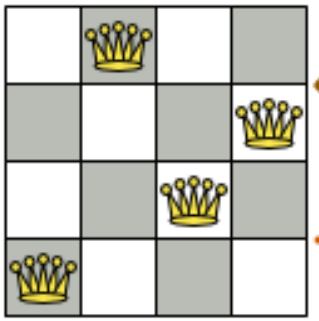
General phase sequence

First a construction heuristic,
then metaheuristics



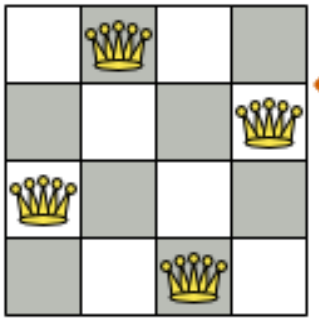
Construction heuristic

First Fit Decreasing



Metaheuristic

Tabu Search



Scope overview

Each scope triggers lifecycle events

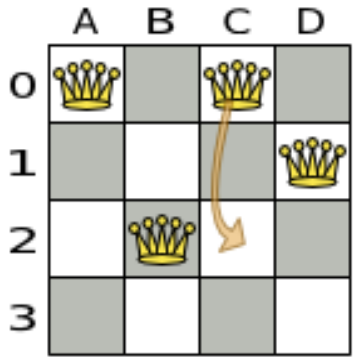
Move types

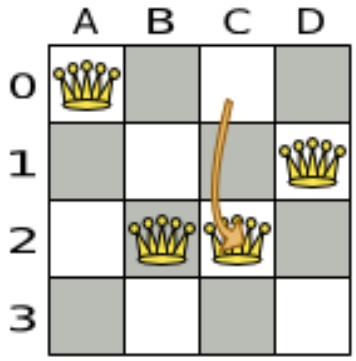
- Change move
- Swap move
- ...

ChangeMove

Change 1 variable of 1 entity

N queens

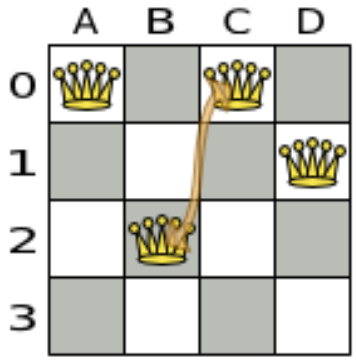


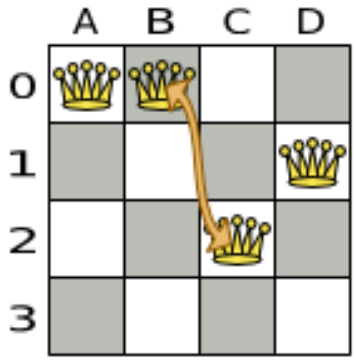


SwapMove

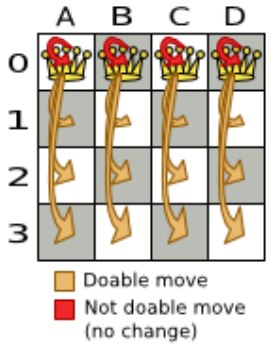
Swap all variables of 2 entities

N queens



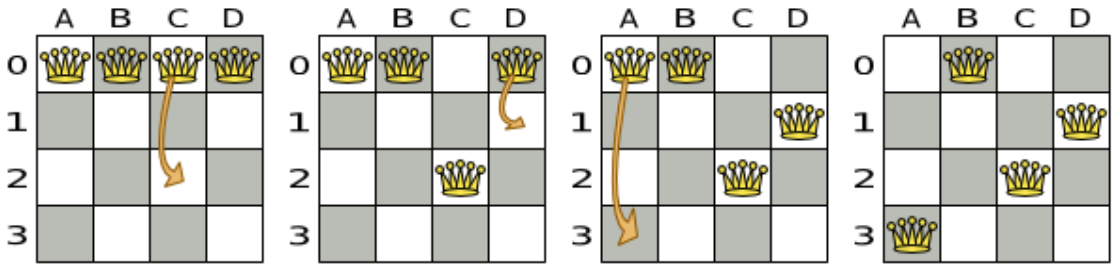


All change moves



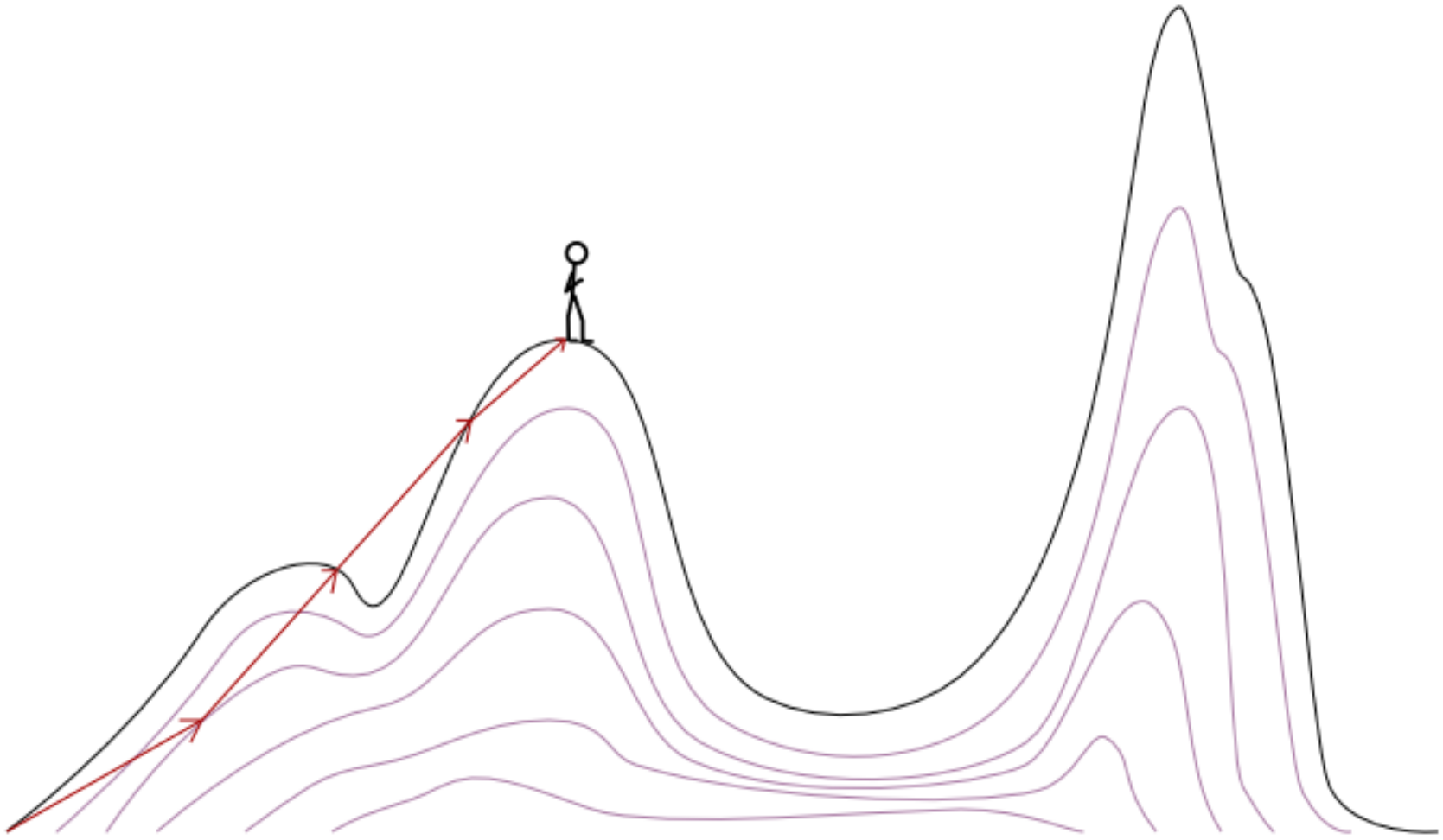
n	# moves	# solutions
4	16	256
8	64	16777216
64	4096	10^{116}
n	n^2	n^n

Multiple moves



Multiple moves can reach any solution

Hill climbing

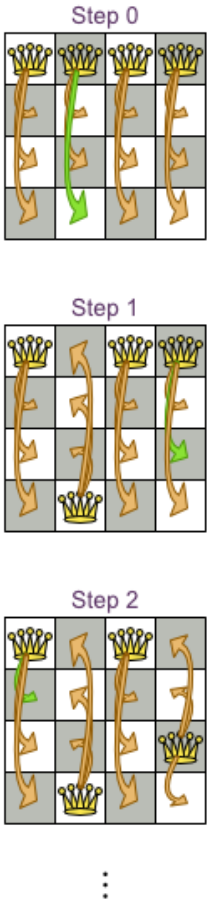


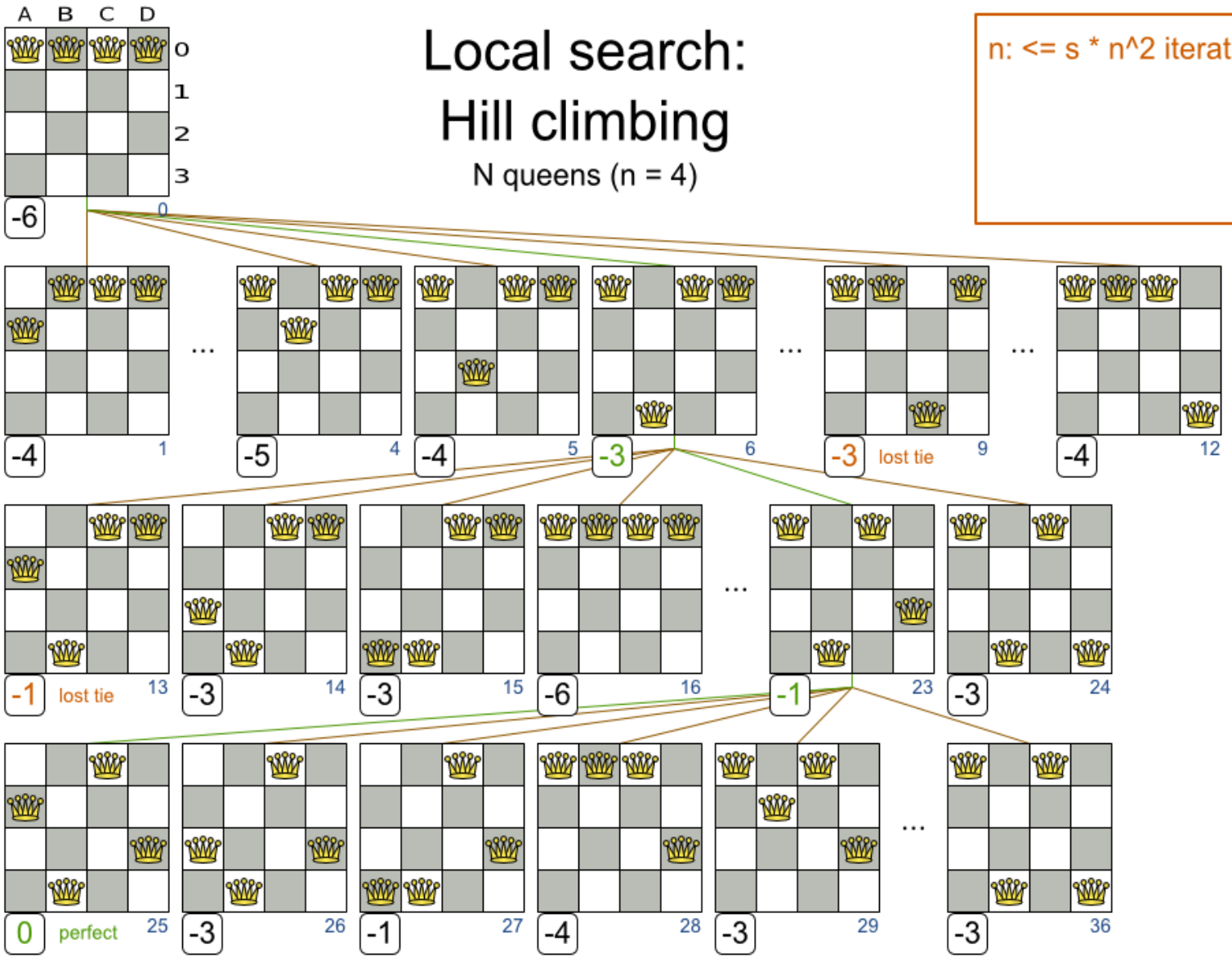
Local search: Hill climbing

N queens (n = 4)

n: <= s * n^2 iterations

Selected moves for each step





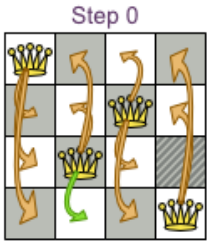
Uses a search path, not a search tree
=> highly scalable

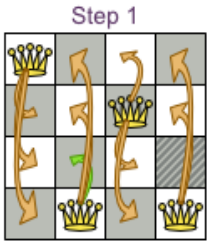
Hill climbing gets stuck in local optima

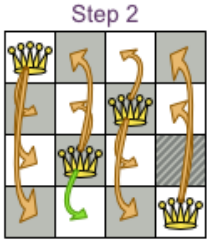
N queens (n = 4)

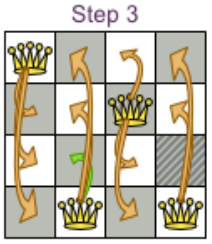
n: <= s * n^2 iterations

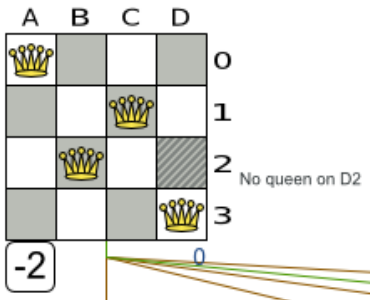
Selected moves for each step

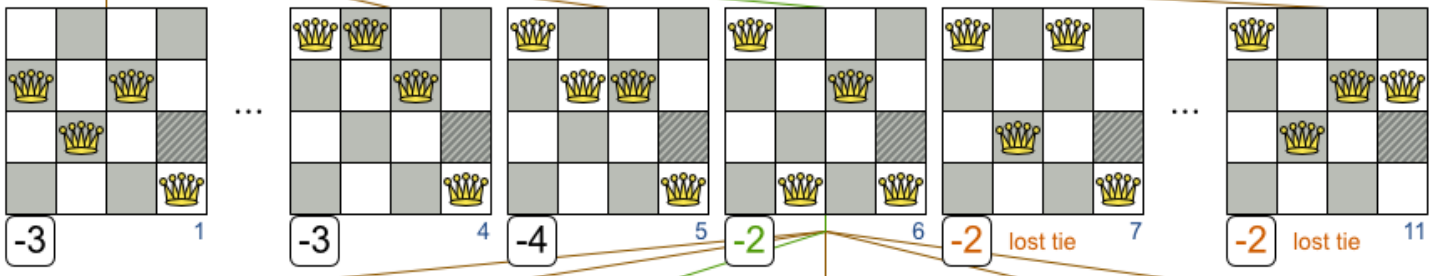


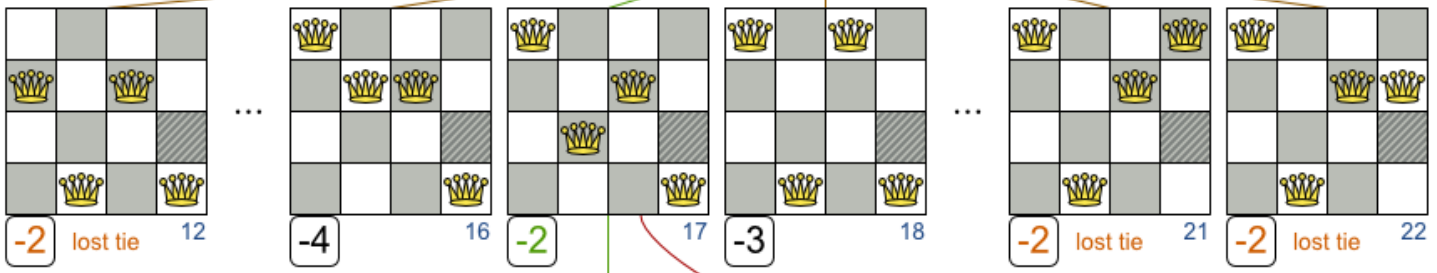


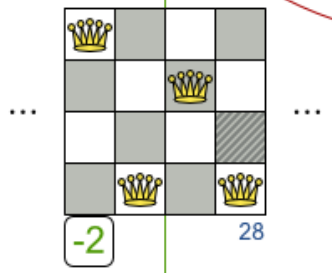


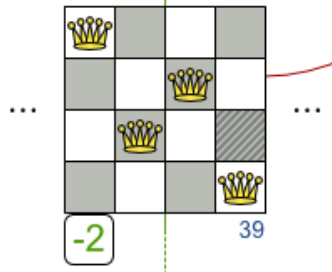






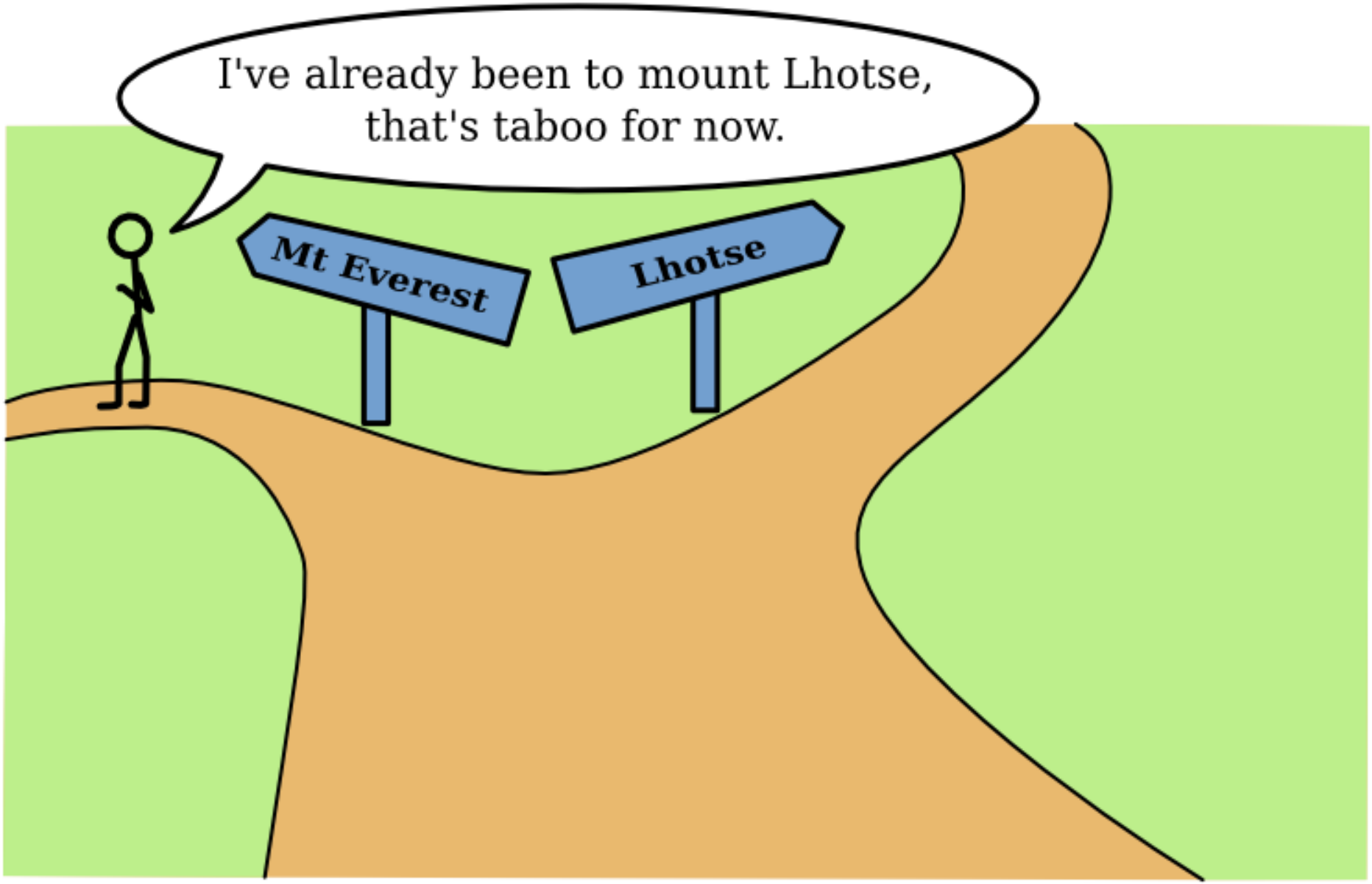






Solution already encountered:
same as starting solution
=> possibly stuck

Tabu Search


I've already been to mount Lhotse,
that's taboo for now.
Mt Everest
Lhotse

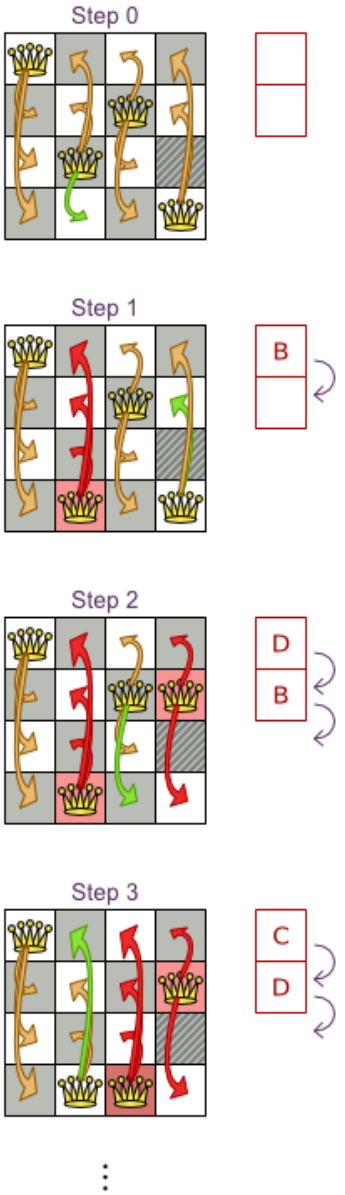
Tabu search: entity tabu

N queens (n = 4, entityTabuSize = 2)

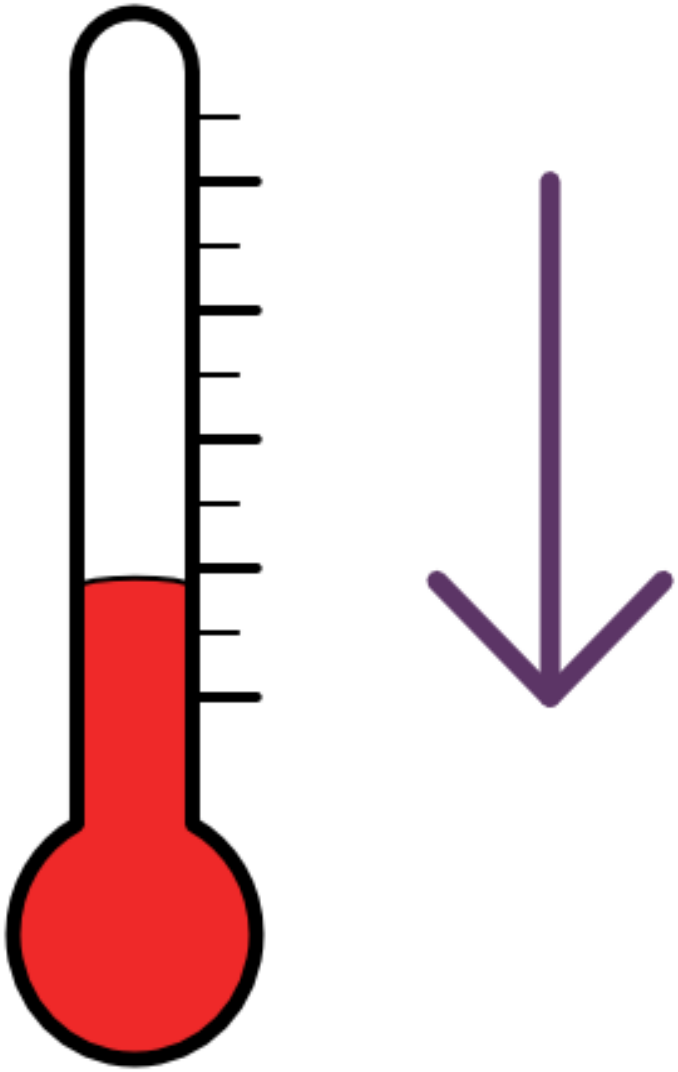
n: <= s * n^2 iterations

Selected moves for each step

Tabu list



Simulated Annealing



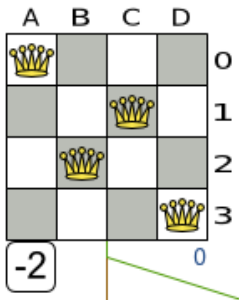
Simulated Annealing (Time Gradiant aware)

N queens (n = 4, startingTemperature = 2)

Temperature decreases for each step

n: <= s * m iterations

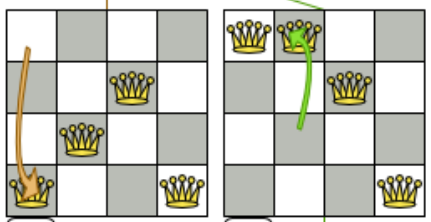
$\max \square = e^{\Delta/t}$



Step 0

t
2.0

Δ	max
≥0	any
-1	0.61
-2	0.37
-3	0.22
-4	0.14



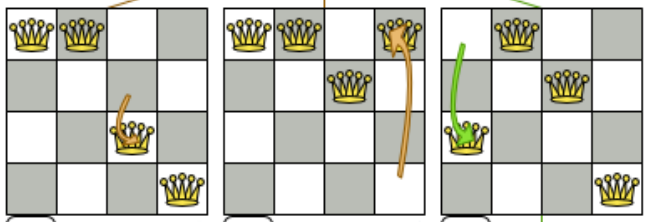
Δ = -2 ⬛ = 0.41 > 0.37

Δ = -1 ⬛ = 0.50 ≤ 0.61

Step 1

t
1.6

Δ	max
≥0	any
-1	0.54
-2	0.29
-3	0.15
-4	0.08



Δ = -1 ⬛ = 0.88 > 0.54

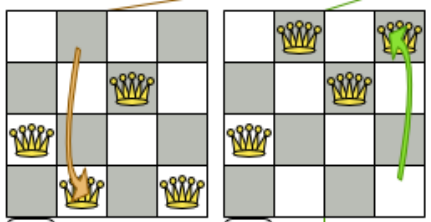
Δ = -2 ⬛ = 0.31 >0.29

Δ = +2 ⬛ = n/a ≤ any

Step 2

t
1.2

Δ	max
≥0	any
-1	0.43
-2	0.19
-3	0.08
-4	0.04



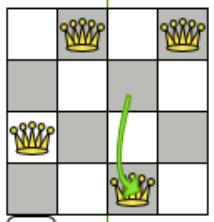
Δ = -1 ⬛ = 0.74 > 0.43

Δ = -2 ⬛ = 0.18 ≤ 0.19

Step 3

t
0.8

Δ	max
≥0	any
-1	0.29
-2	0.08
-3	0.02
-4	0.01

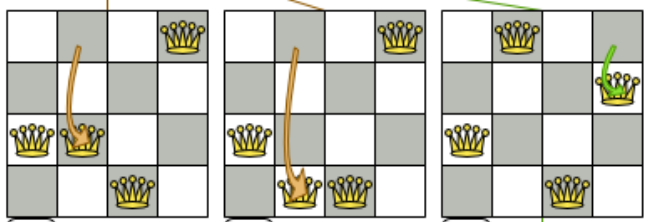


Δ = +2 ⬛ = n/a ≤ any

Step 4

t
0.4

Δ	max
≥0	any
-1	0.08
-2	0.01
-3	0.00
-4	0.00



Δ = -2 ⬛ = 0.97 > 0.01

Δ = -1 ⬛ = 0.11 > 0.08

Δ = +1 ⬛ = n/a ≤ any

⋮

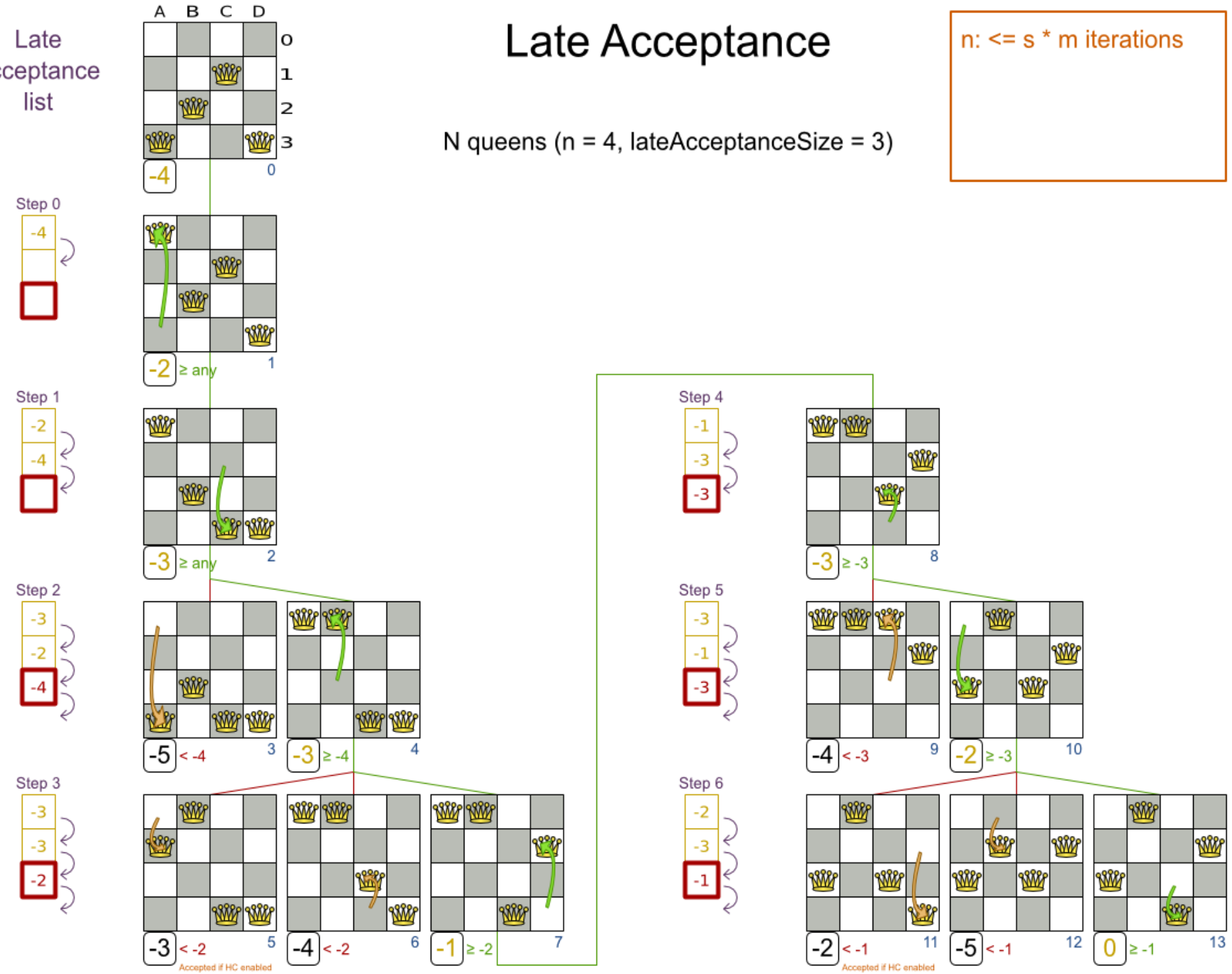
Late acceptance

Late Acceptance

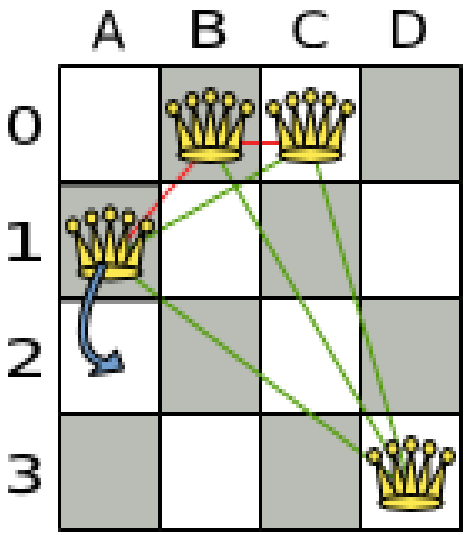
N queens (n = 4, lateAcceptanceSize = 3)

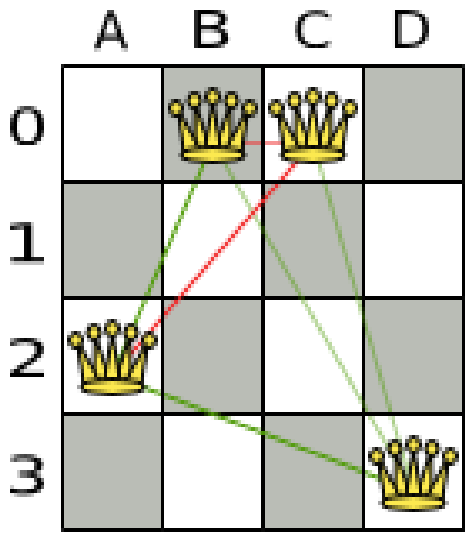
n: <= s * m iterations

Late acceptance list



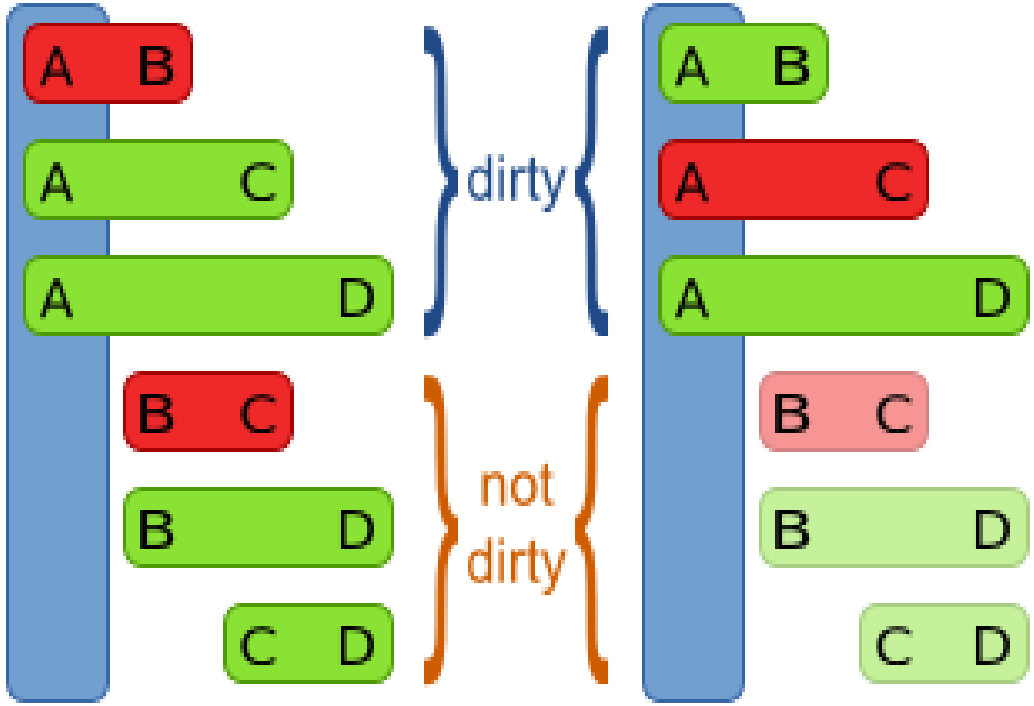
Performance tricks





Delta based score calculation

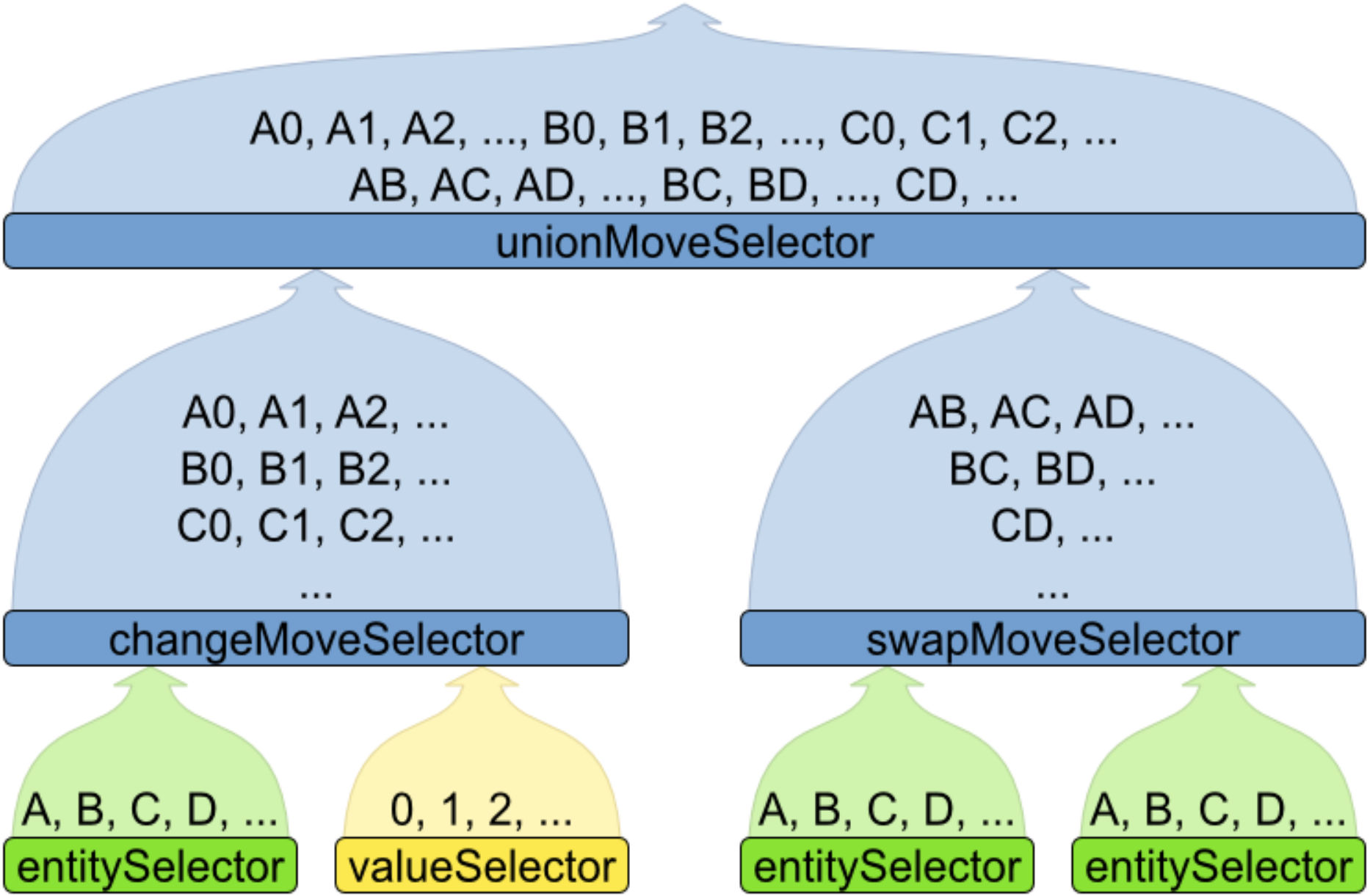
The rule engine
(with forward chaining)
only recalculates dirty tuples.

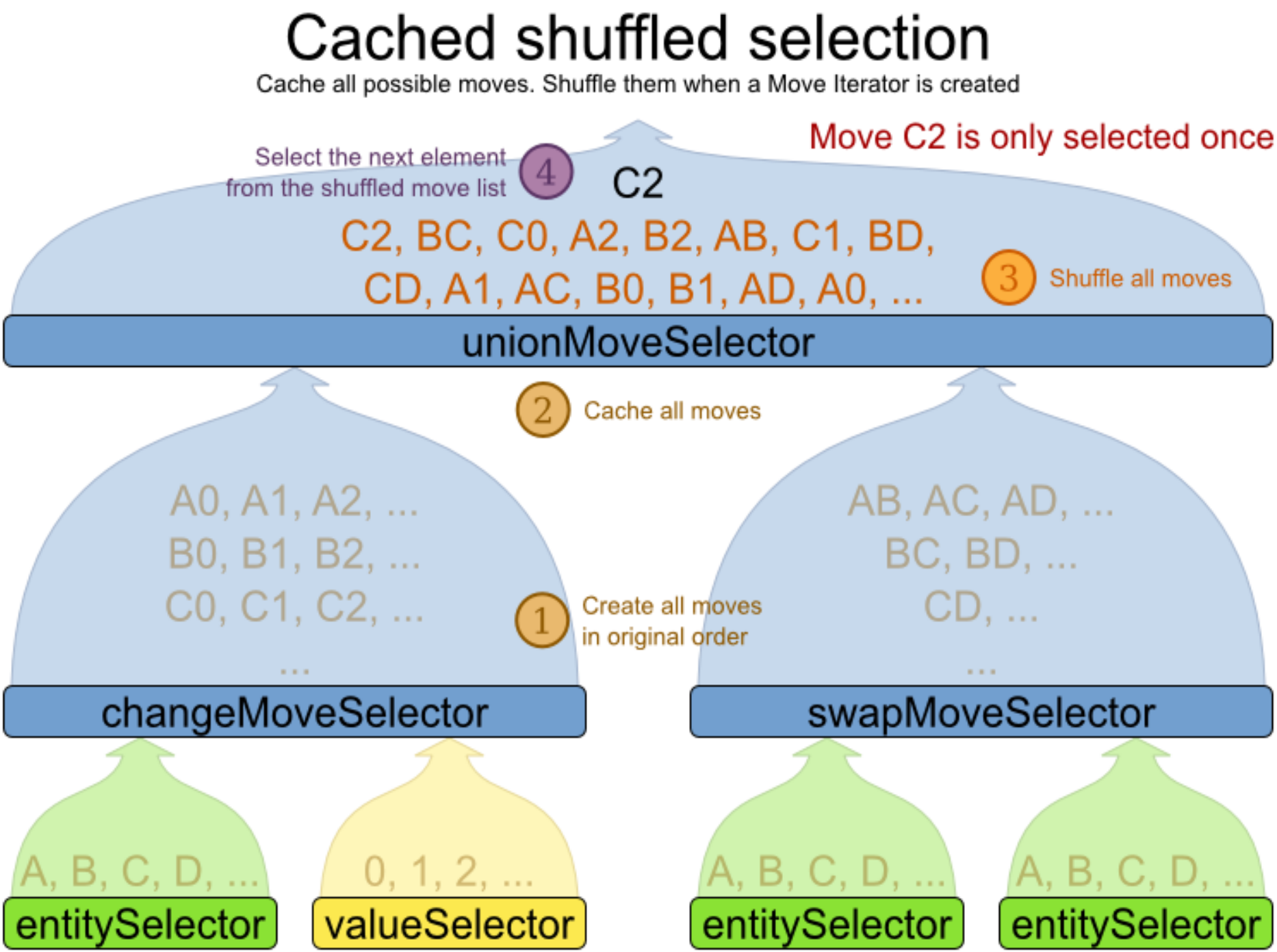


queens	dirty	total	speedup
4	3 of	6	time / 2
8	7 of	28	time / 4
16	15 of	120	time / 8
32	31 of	496	time / 16
64	63 of	2016	time / 32

Selector tree

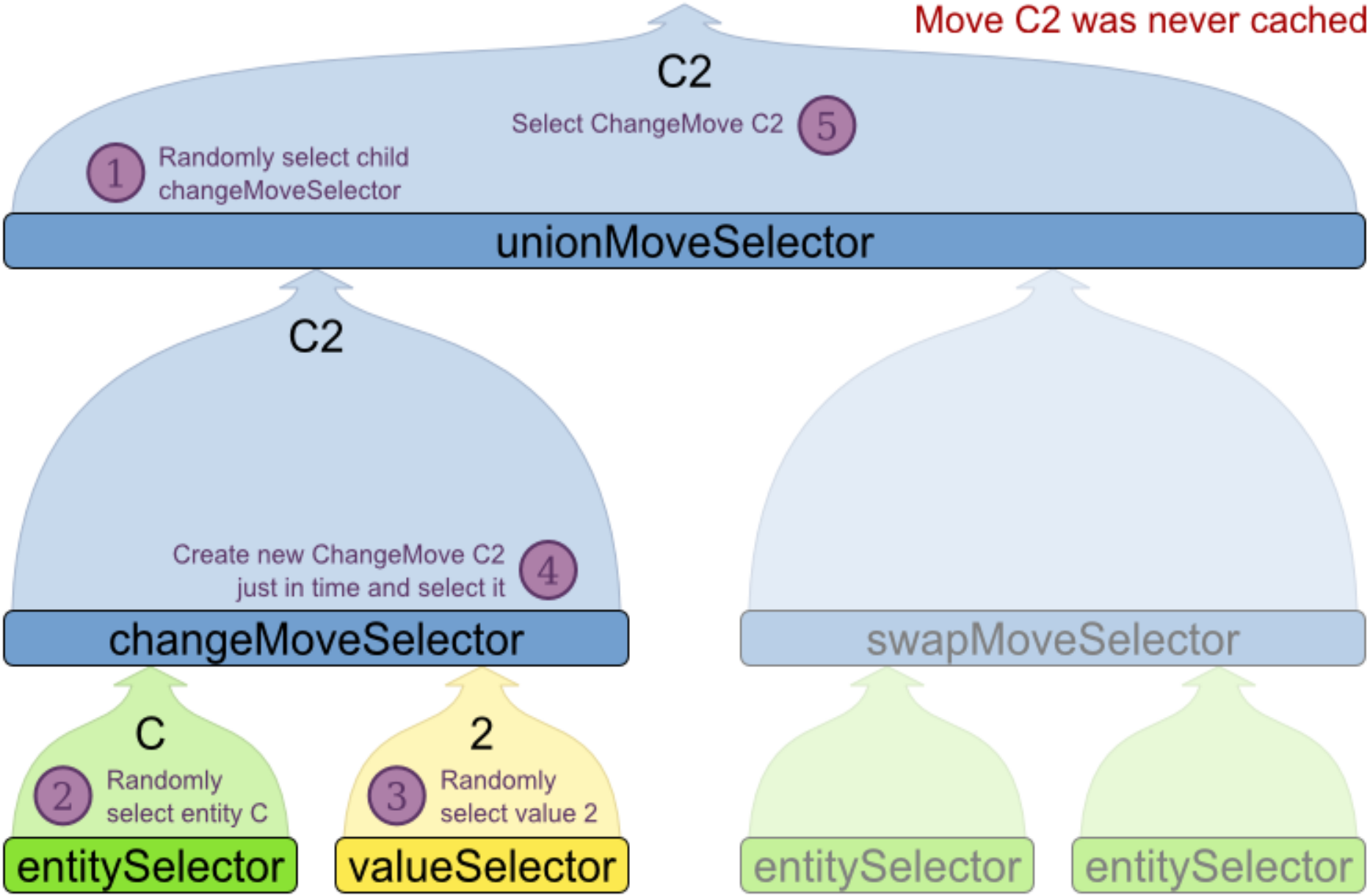
A MoveSelector can be composed out of other MoveSelectors, EntitySelectors and/or ValueSelectors.




Cached shuffled selection
Cache all possible moves. Shuffle them when a Move Iterator is created
Move C2 is only selected once
Select the next element
from the shuffled move list
4
C2
C2, BC, C0, A2, B2, AB, C1, BD,
CD, A1, AC, B0, B1, AD, A0, ...
3
Shuffle all moves
unionMoveSelector
2
Cache all moves
A0, A1, A2, ...
B0, B1, B2, ...
C0, C1, C2, ...
...
1
Create all moves
in original order
AB, AC, AD, ...
BC, BD, ...
CD, ...
...
changeMoveSelector
swapMoveSelector
A, B, C, D, ...
0, 1, 2, ...
A, B, C, D, ...
A, B, C, D, ...
entitySelector
valueSelector
entitySelector
entitySelector

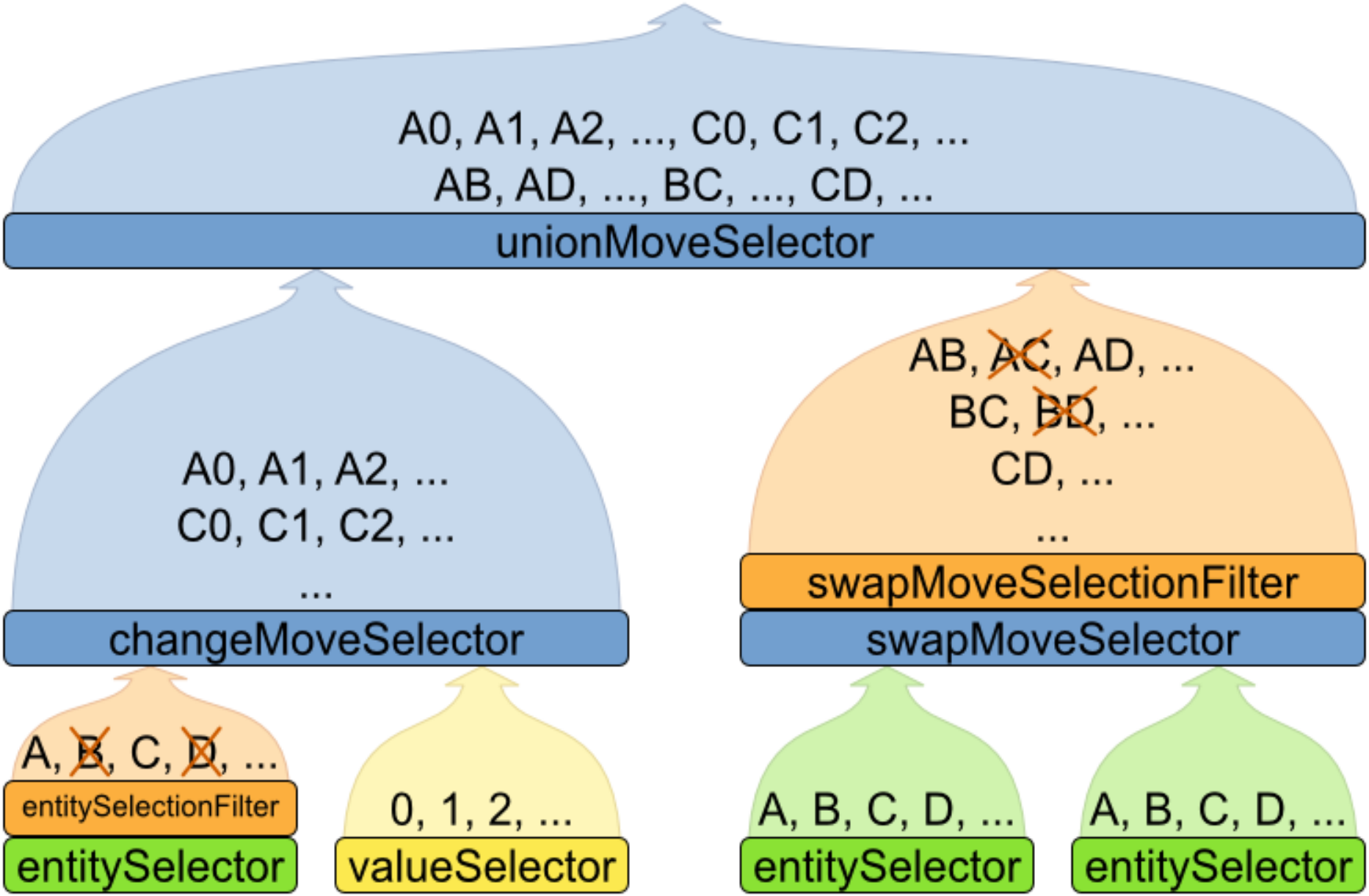
Just in time random selection

Create a random Move just before it's needed and no sooner



Filtered selection

The output of any Selector can be filtered with one or more SelectionFilters



Reuse existing business code

```
public class Nurse {

  private Country country;

  public boolean isHoliday(Date date) {
    if (country == Country.BE) {
      // true if date is 1-JAN, easter monday, 21-JUL, ...
    } else if (country == Country.FR) {
      // true if date is 1-JAN, easter monday, 14-JUL, ...
    } else if (...) {
      ...
    }
    ...
  }

}
```

Q & A

- OptaPlanner homepage
  - http://www.optaplanner.org (http://www.optaplanner.org)
- Reference manual
  - http://www.optaplanner.org/learn/documentation.html (http://www.optaplanner.org/learn/documentation.html)
- Download/fork this presentation
  - http://www.optaplanner.org/learn/slides.html (http://www.optaplanner.org/learn/slides.html)

- What did you think of this presentation?
  - Twitter: @geoffreydesmet (http://twitter.com/geoffreydesmet)
  - Google+: +GeoffreyDeSmet (https://plus.google.com/+GeoffreyDeSmet)