OptaPlanner

Do more business with less resources

by Geoffrey De Smet
OptaPlanner lead

TSP & VRP

The story
of
the ultimate American road trip

Wonkblog

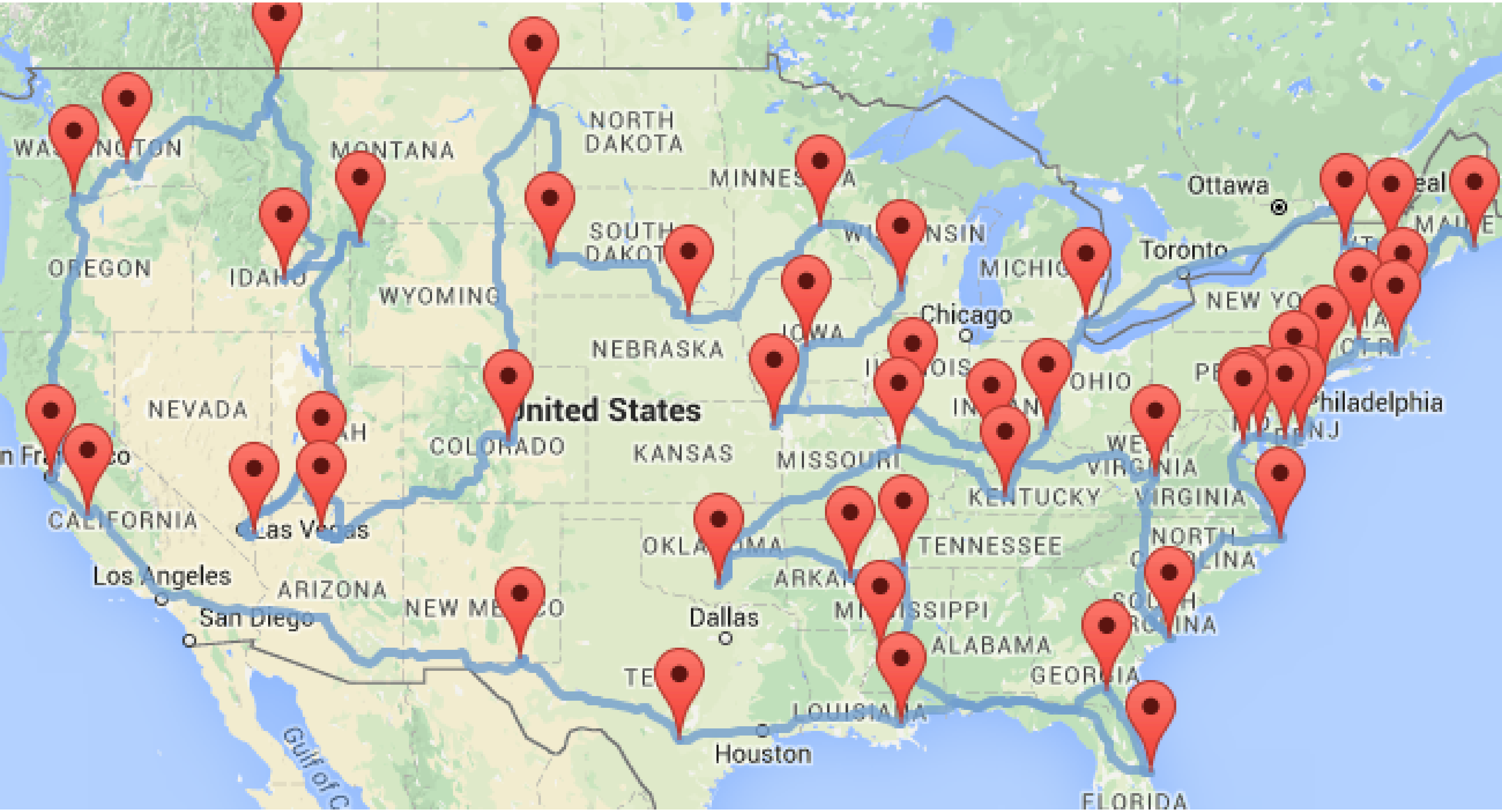
A data genius computes the ultimate American road trip

By Ana Swanson March 10, 2015 Follow @anaswanson

This post comes via Know More, Wonkblog's social media site.

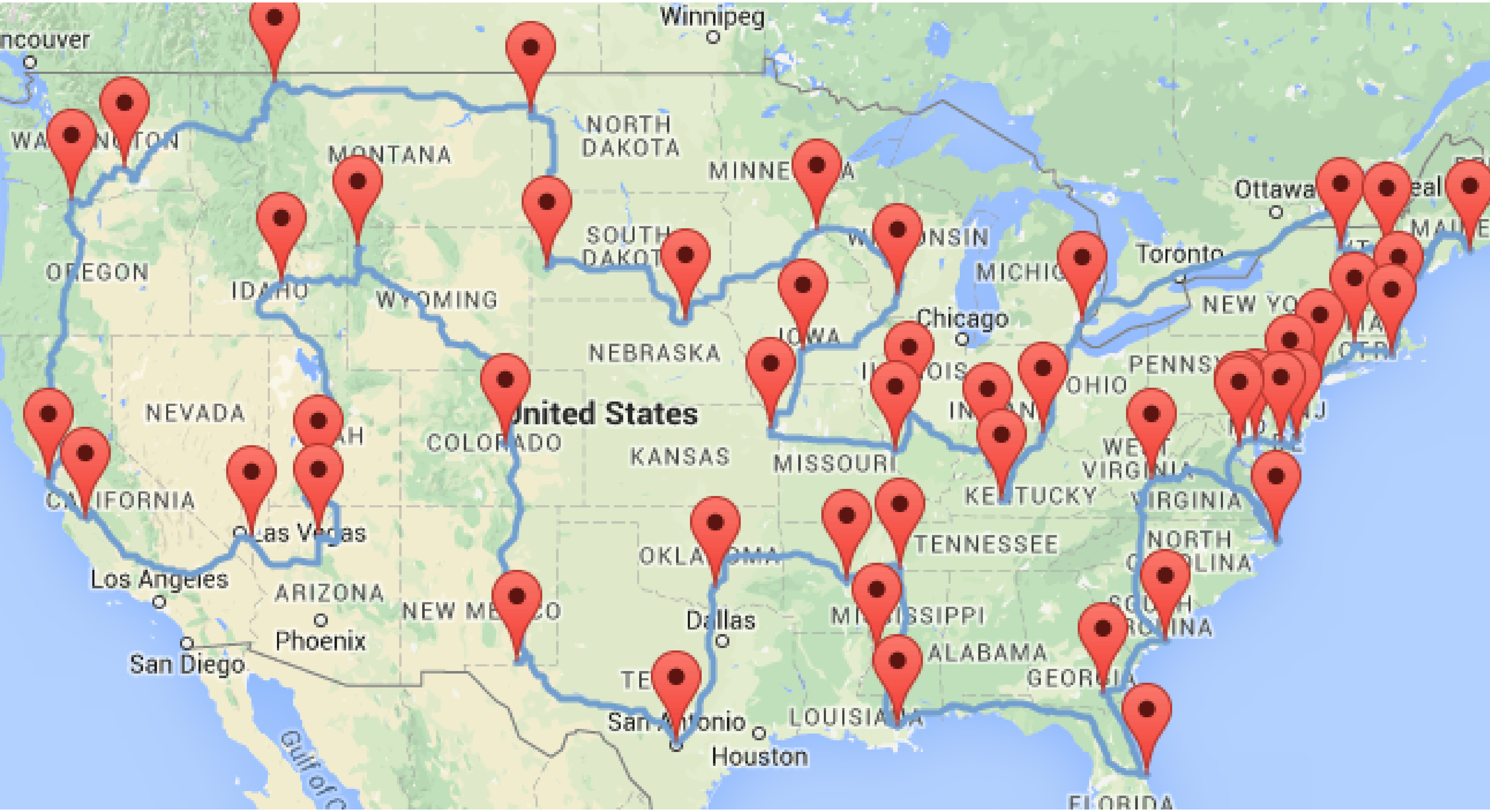
Who needs an atlas when you have an algorithm? Data tinkerer Randy Olson, who previously developed the optimal search path for finding the bespectacled main character of the "Where's Waldo?" books, has used this same algorithm to compute the ultimate American road trip.

Road trip for 50 landmarks, monuments, etc.



Traditional algorithm: 271h 35m 16s

Is it optimal?

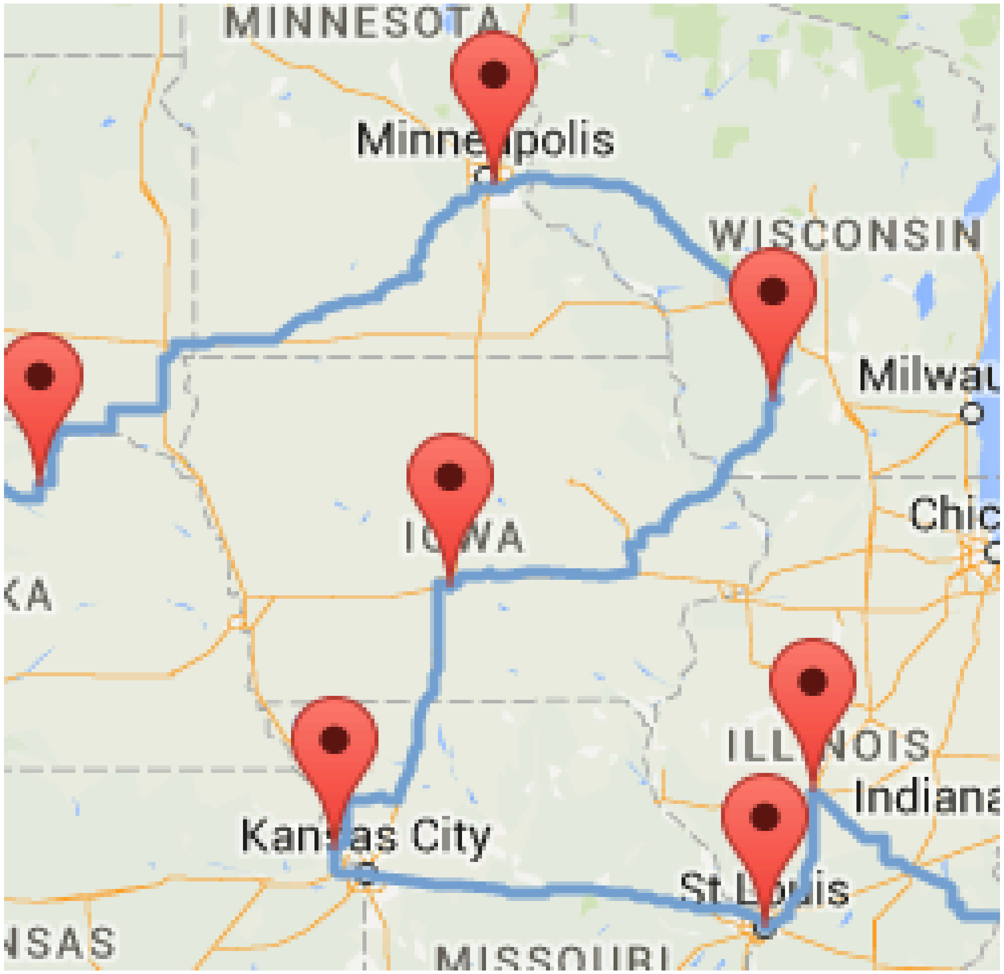


Olson's trip: 232h 43m 10s

⇒ 38h 52m 6s faster (14%)

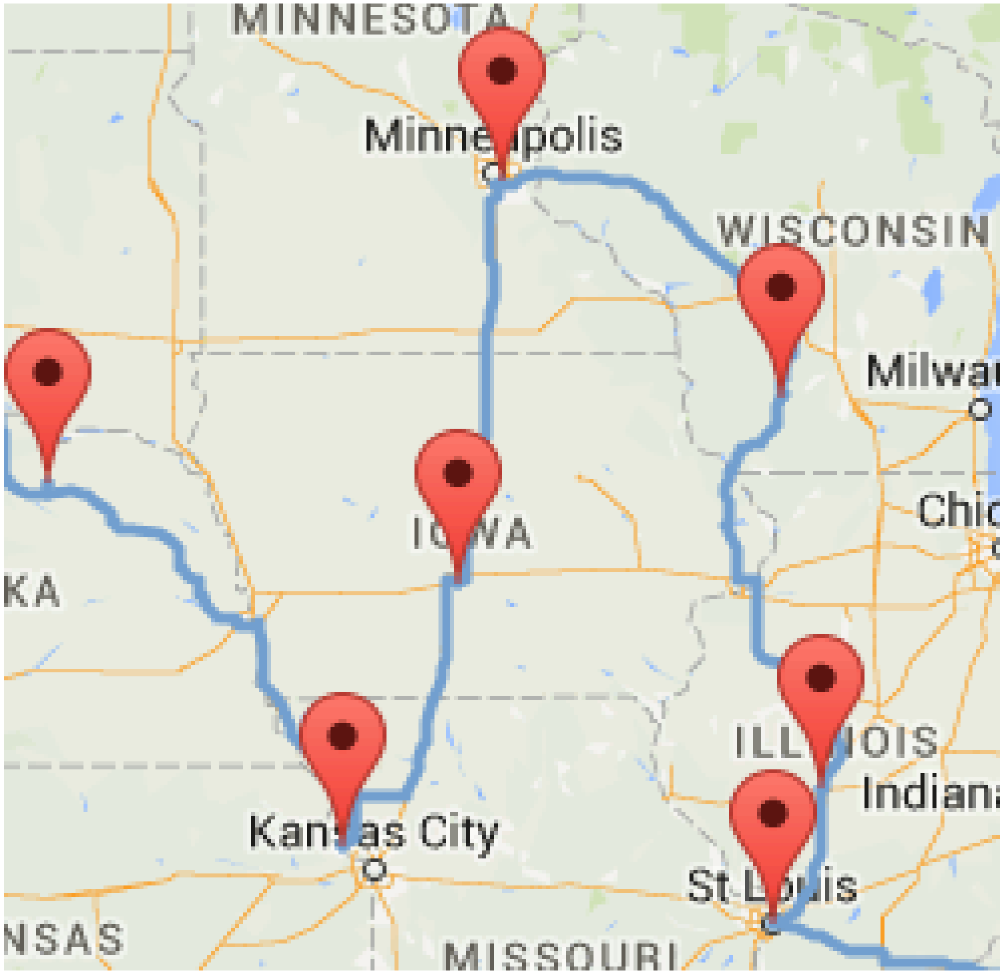
Is it optimal?

Better algorithms



Olson

232h 43m 10s

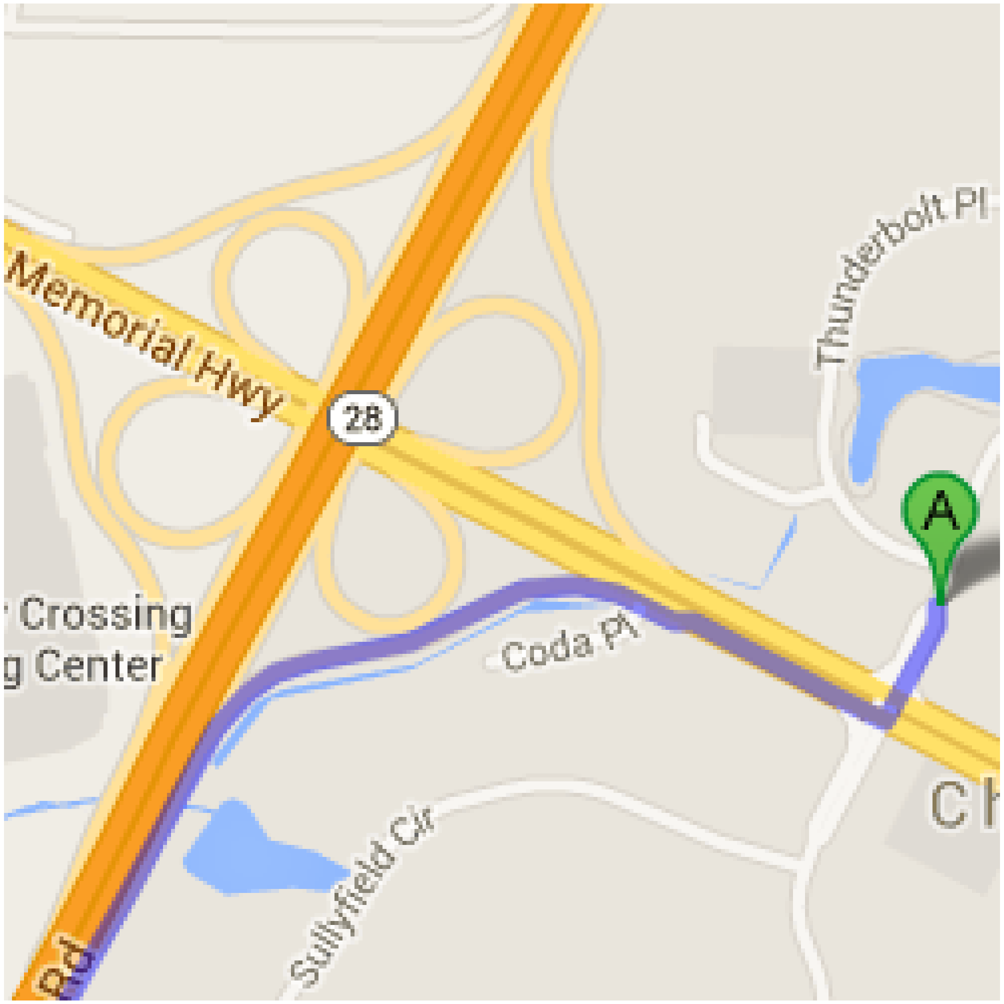


OptaPlanner

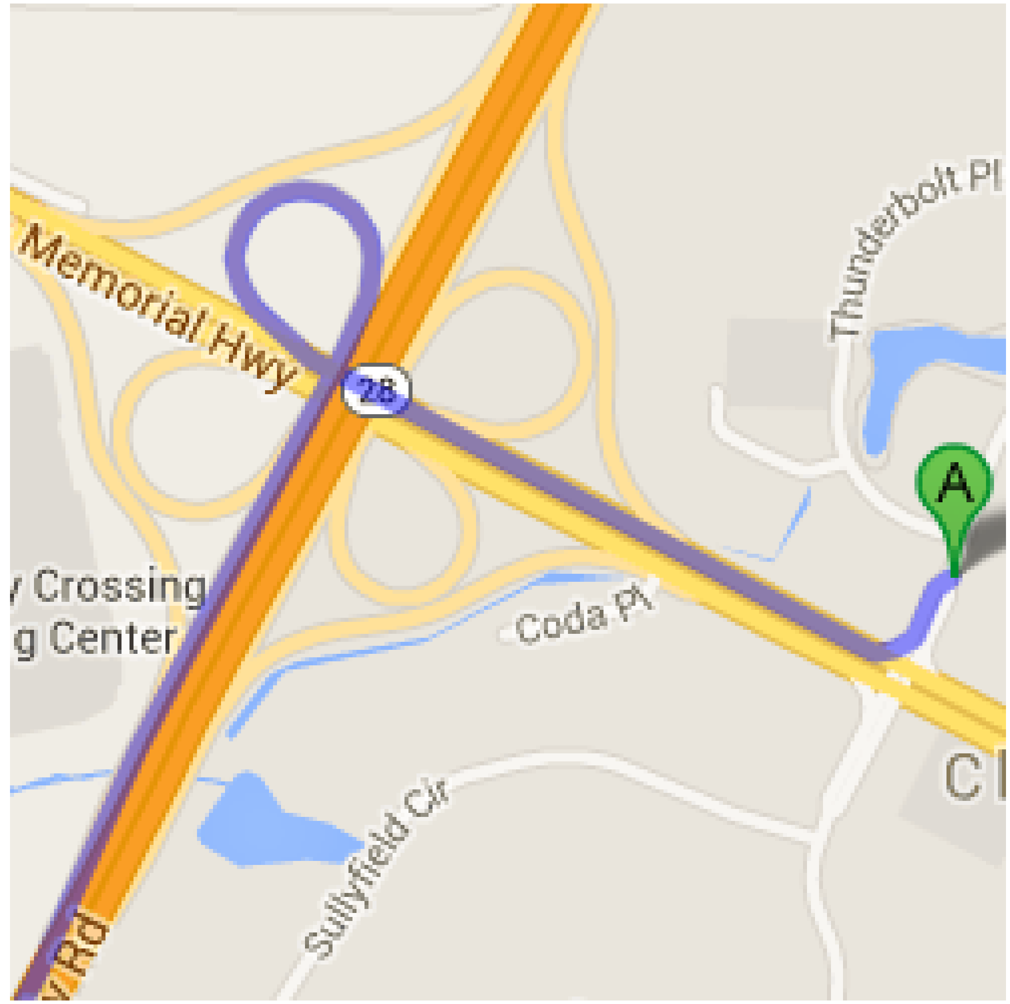
231h 7m 30s

- 1h 35m 40s

Road are asymmetric

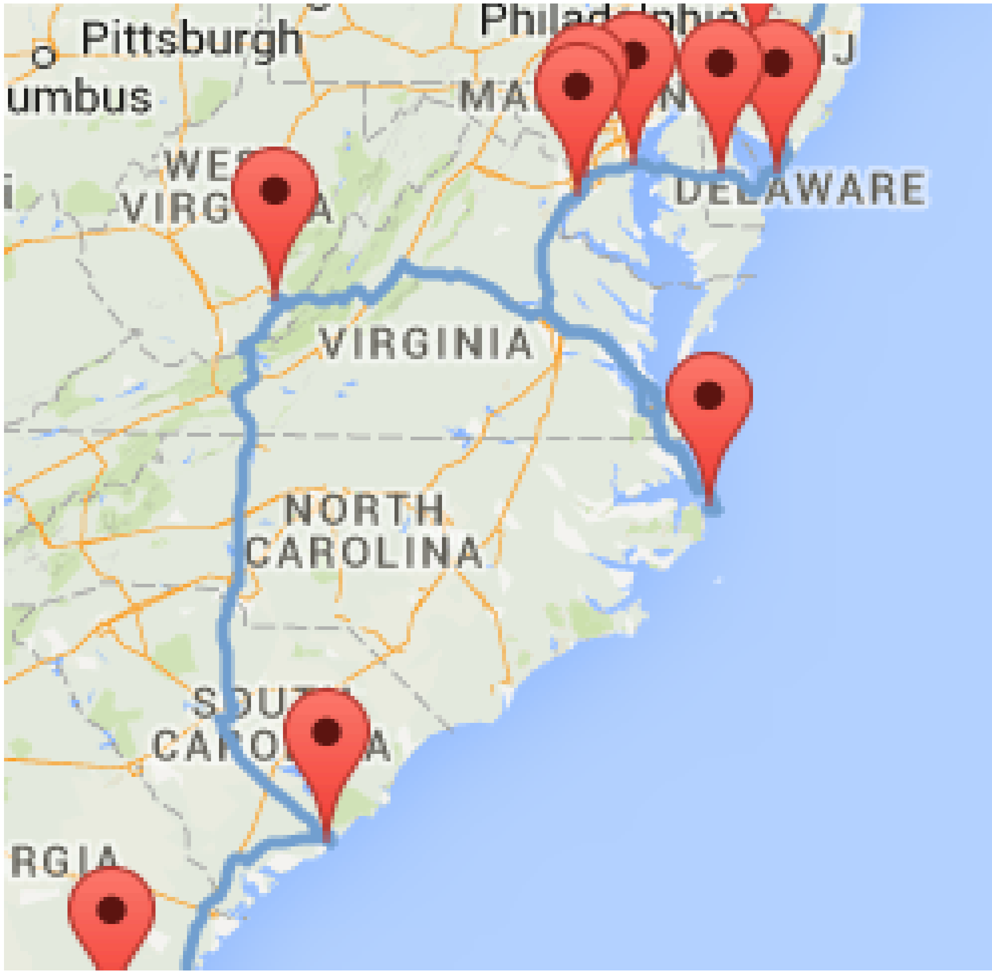


To location A



From location A

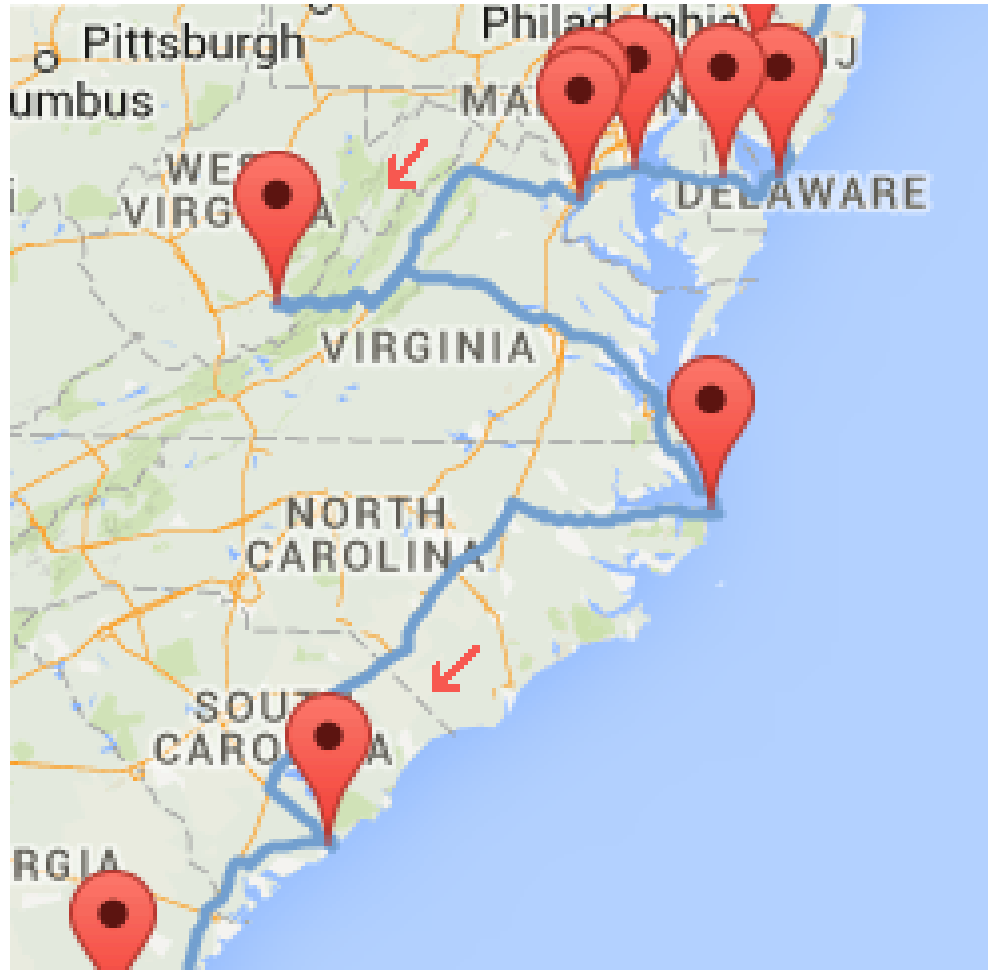
Road are asymmetric



Use symmetric data

231h 7m 30s

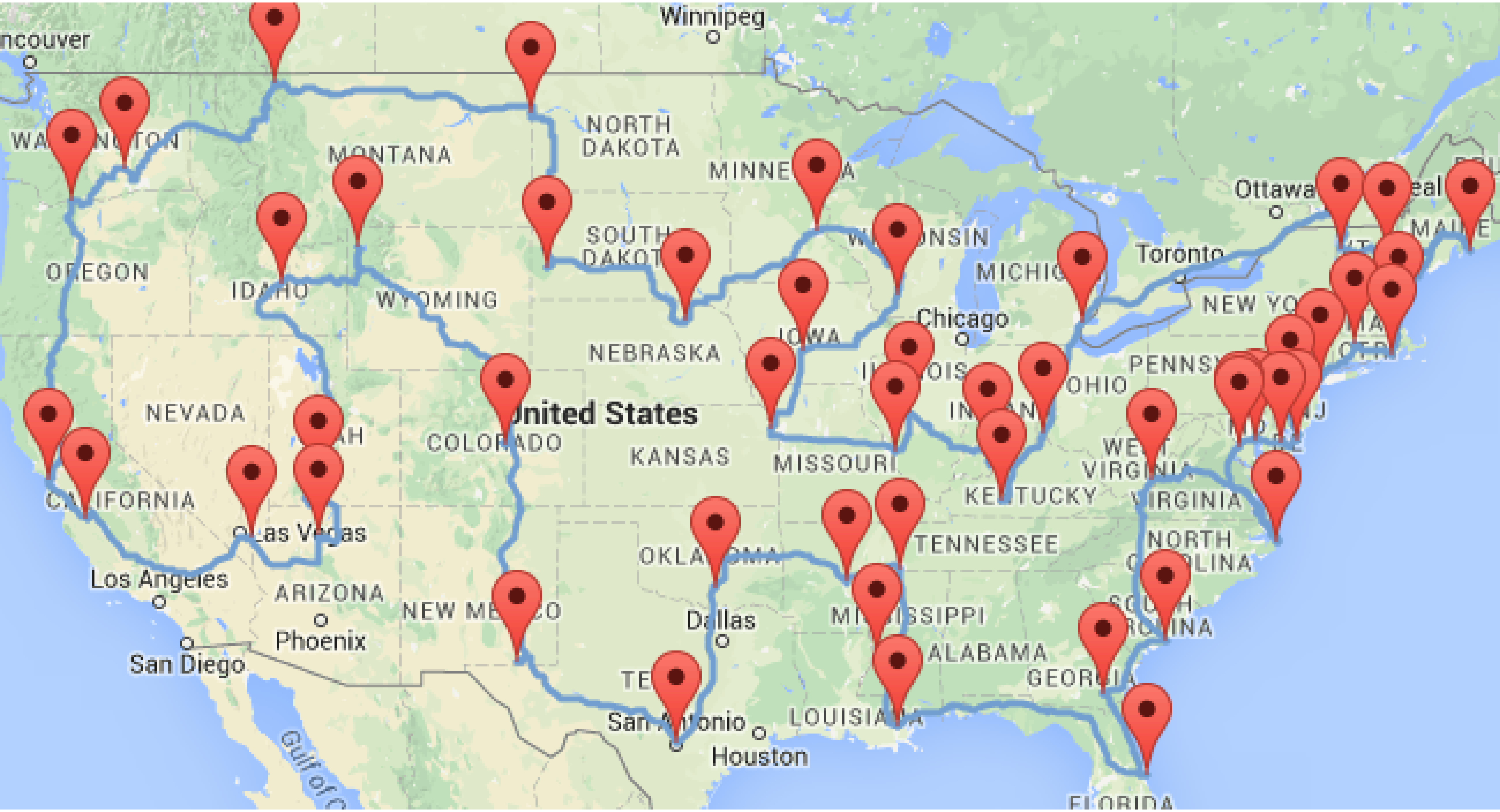
(asymmetric calculation)



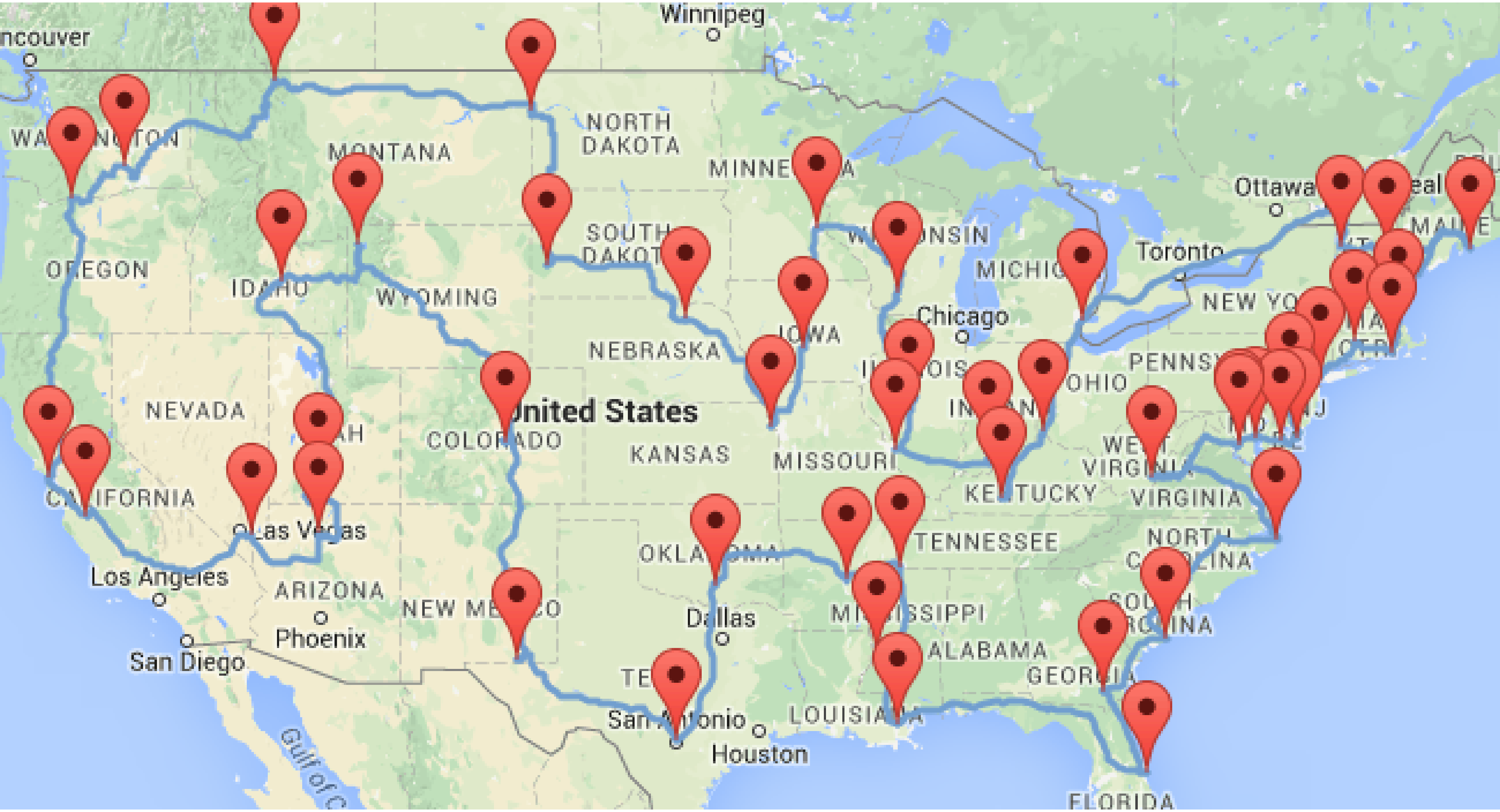
Use asymmetric data

230h 17m 54s

- 49m 36s



Olson's trip: `232h 43m 10s`
Not optimal



OptaPlanner's trip: `230h 17m 54s`

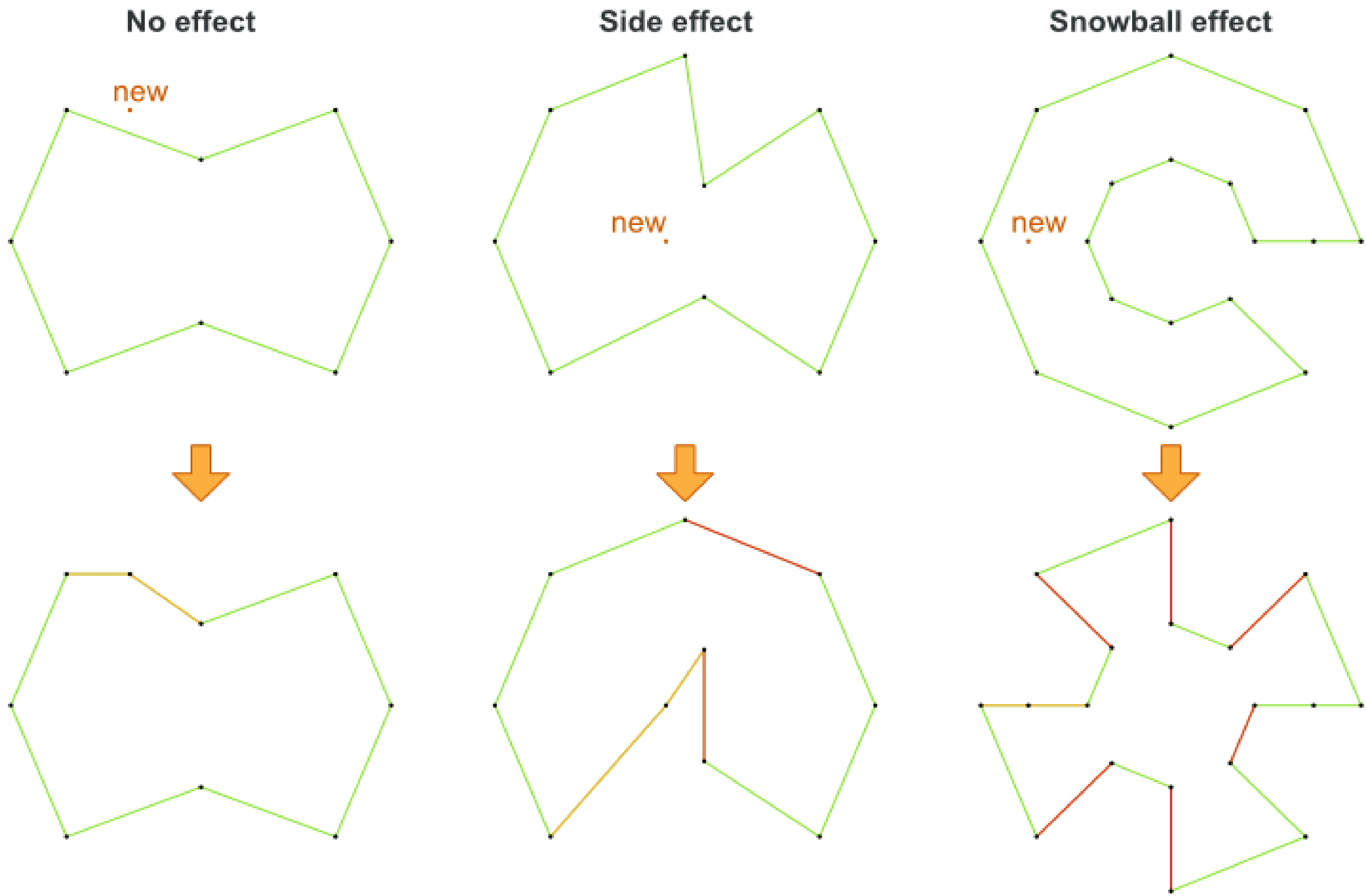
⇒ Another `2h 25m 16s` faster (1% ⇒ 15% in total)

Optimal, also `33km 710m` (= `20.95 miles`) shorter

What is business resource optimization?

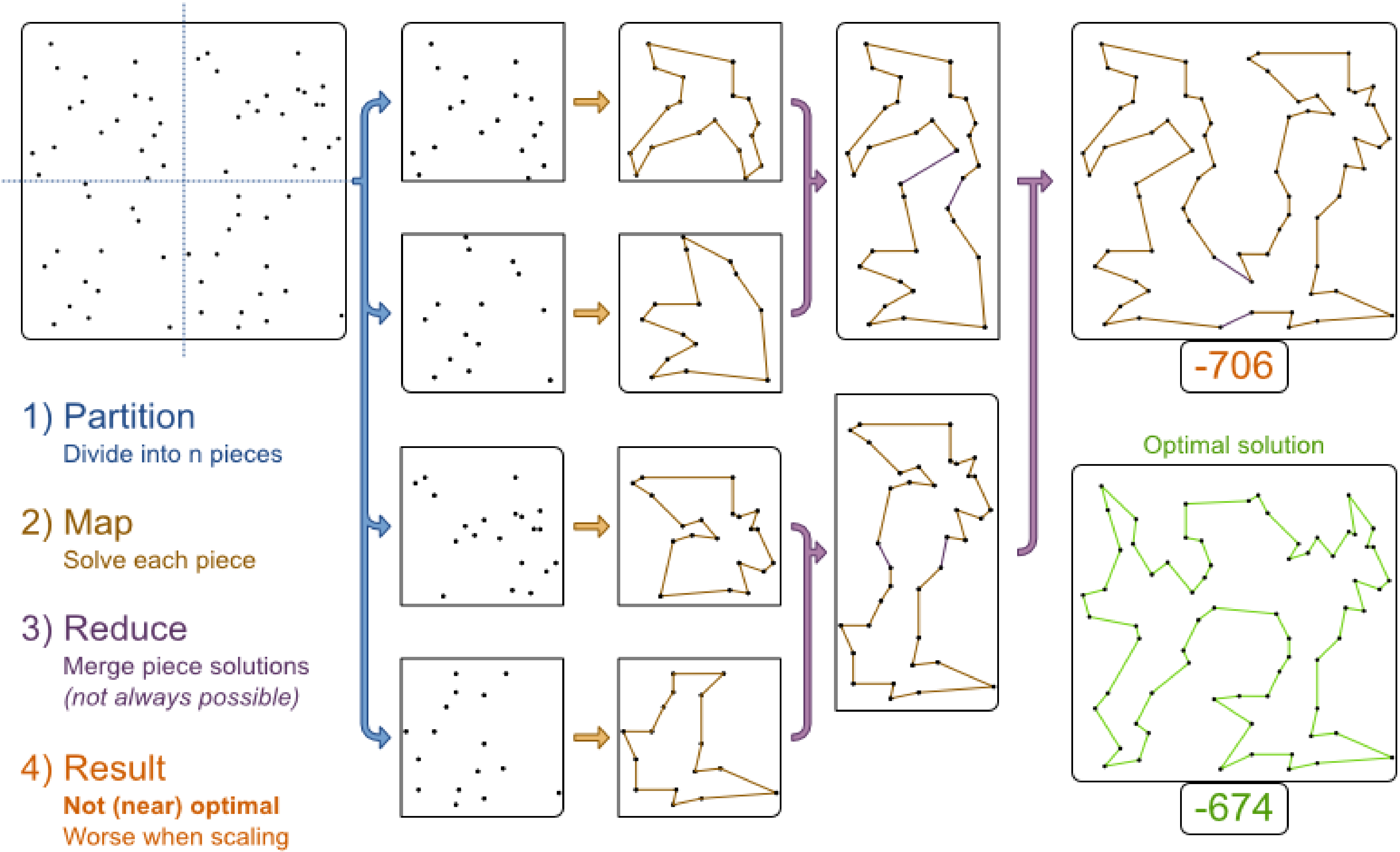
TSP optimal solution volatility

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



MapReduce is terrible for TSP

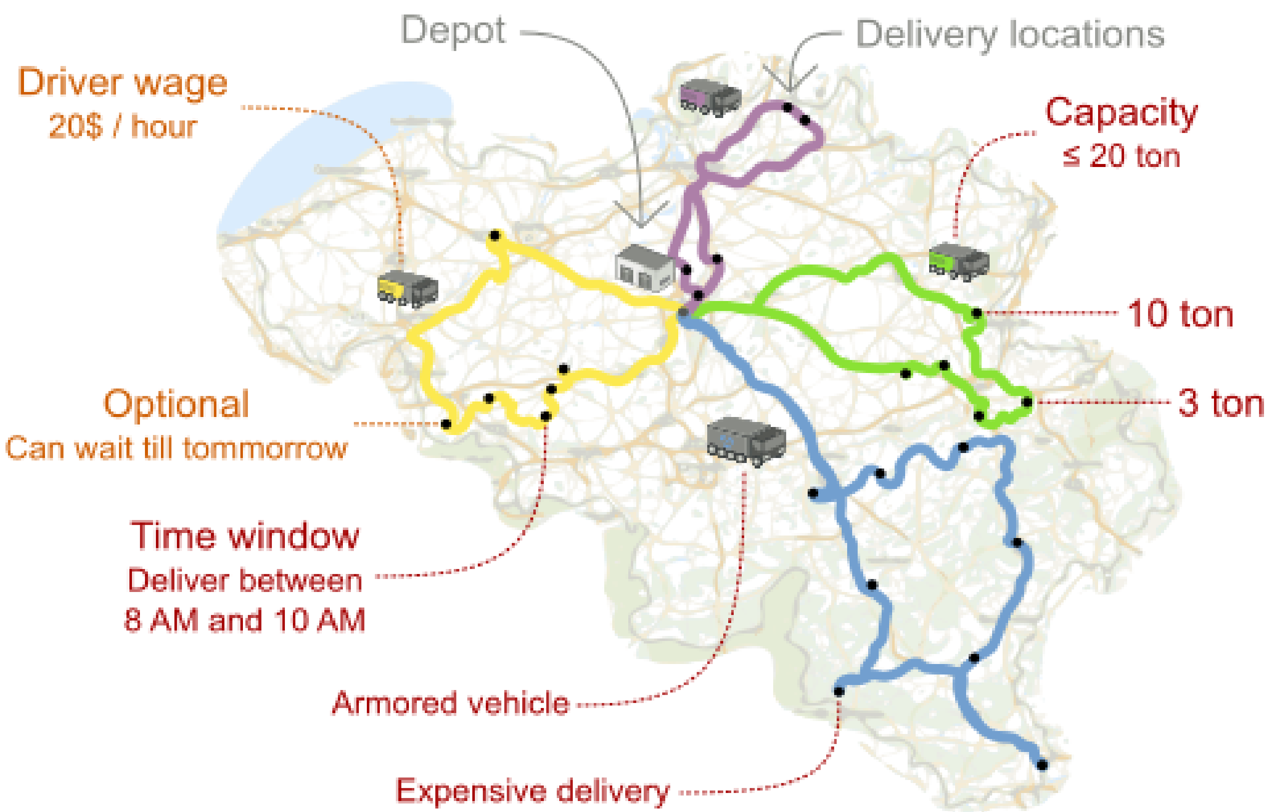
Why do MapReduce, Divide&Conquer and partitioning perform badly on NP-hard problems?



TSP
is an academic problem

Vehicle routing

Assign the delivery order of vehicles more efficiently.



Users

Supermarkets & retail stores

Freight transportation

Buses, taxi's & airlines

Technicians on the road

VehicleRouting benchmark (Belgium datasets)	Average	Min/Max	# datasets	Biggest dataset
Driving time	**-15%**	-9% -18%	5	2750 deliveries 55 vehicles

OptaPlanner versus traditional algorithm with domain knowledge

5 mins Late Acceptance Nearby vs First Fit Decreasing

Don't believe us? Run our open benchmarks yourself: http://www.optaplanner.org/code/benchmarks.html

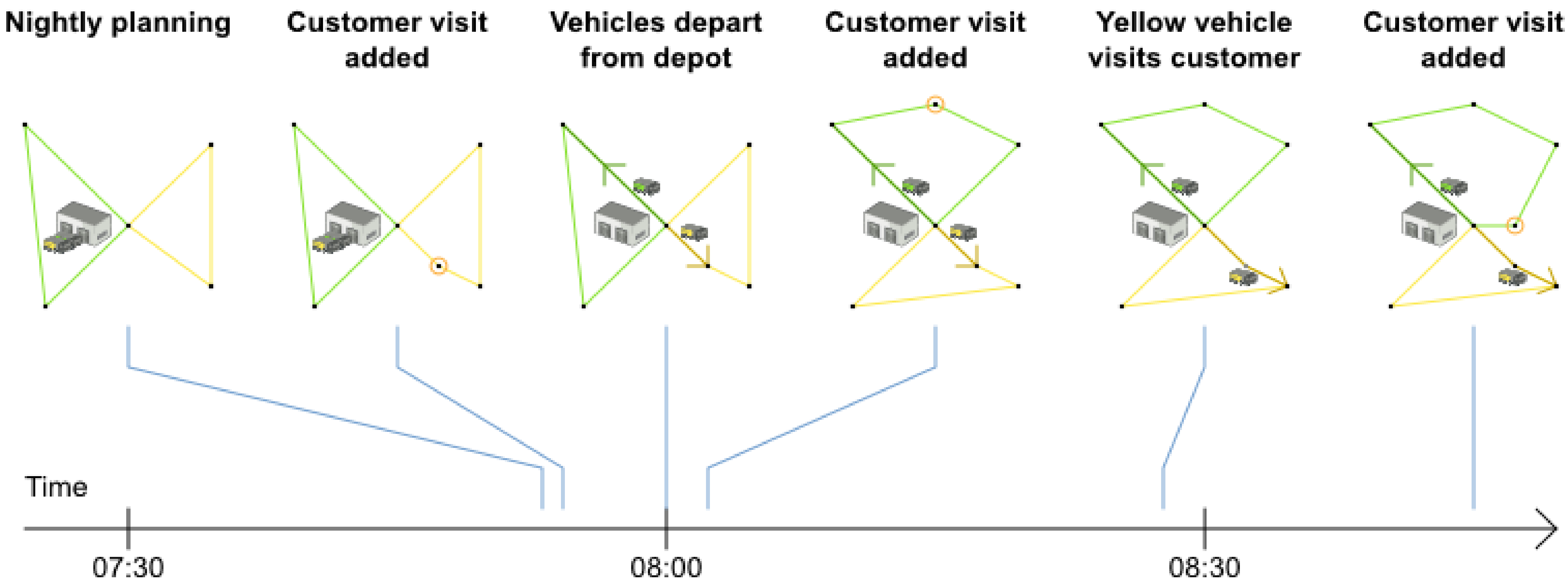
Realistic business resource optimization

Real-time planning

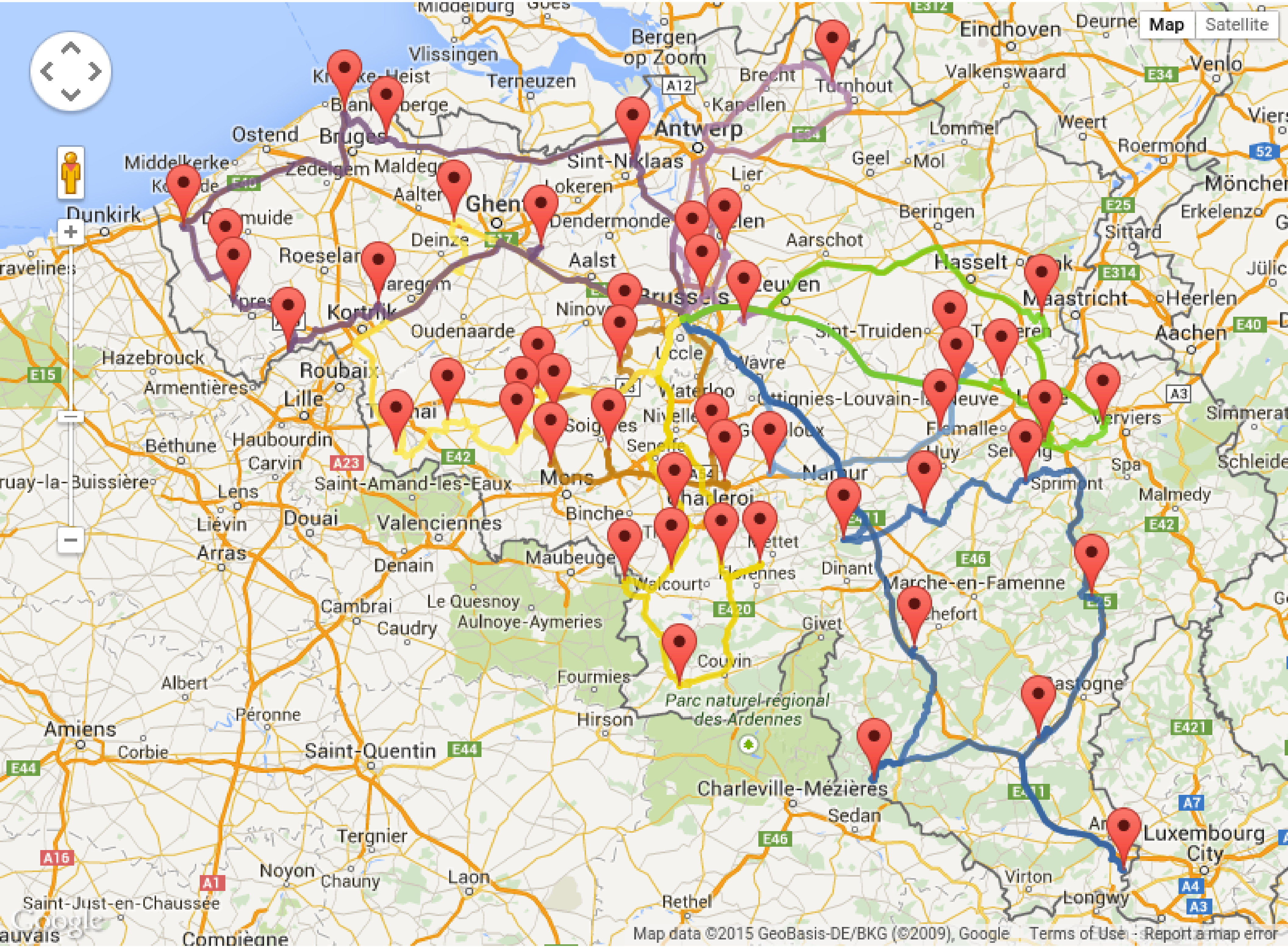
Warm starts to solve in milliseconds

Real-time planning

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

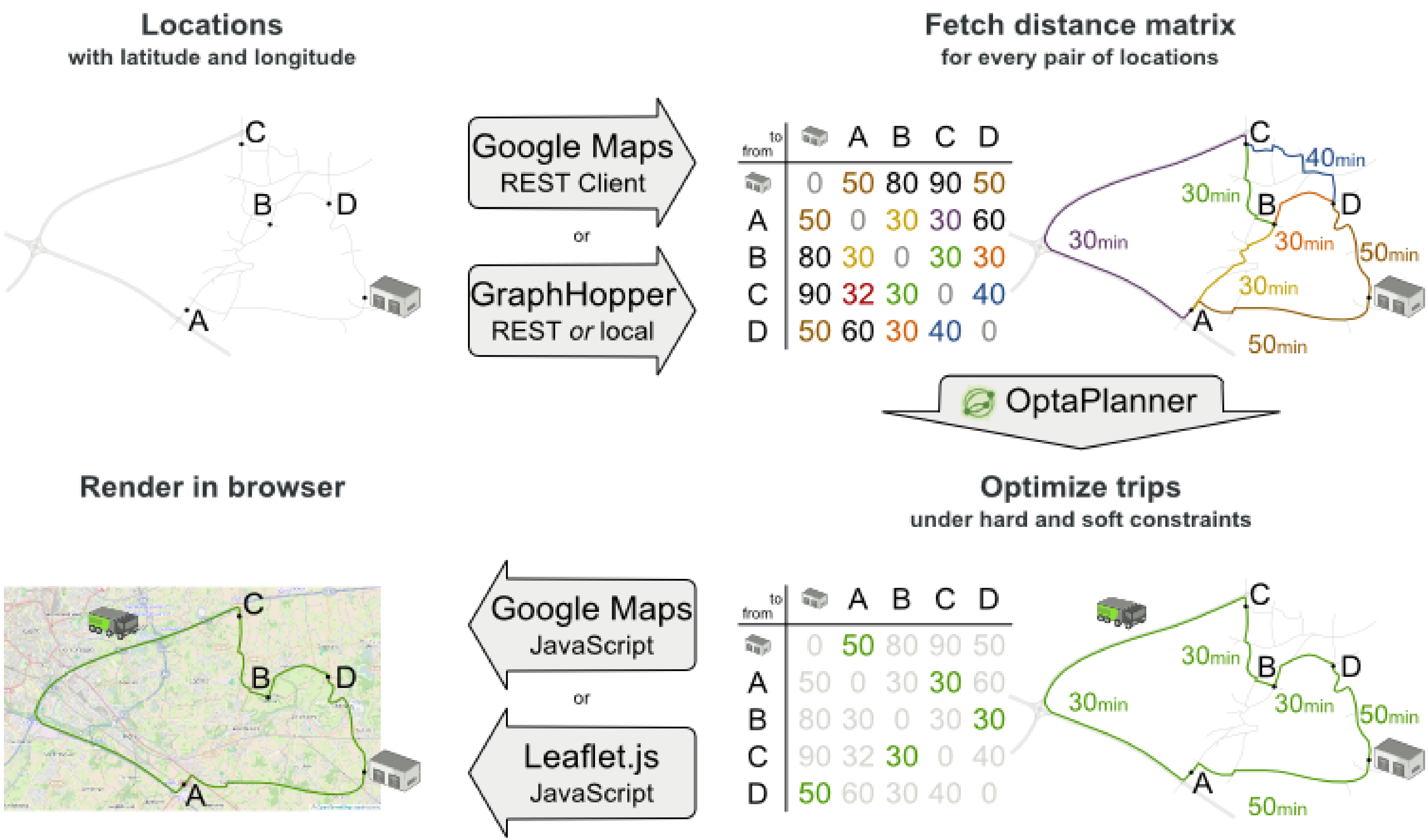


Real roads

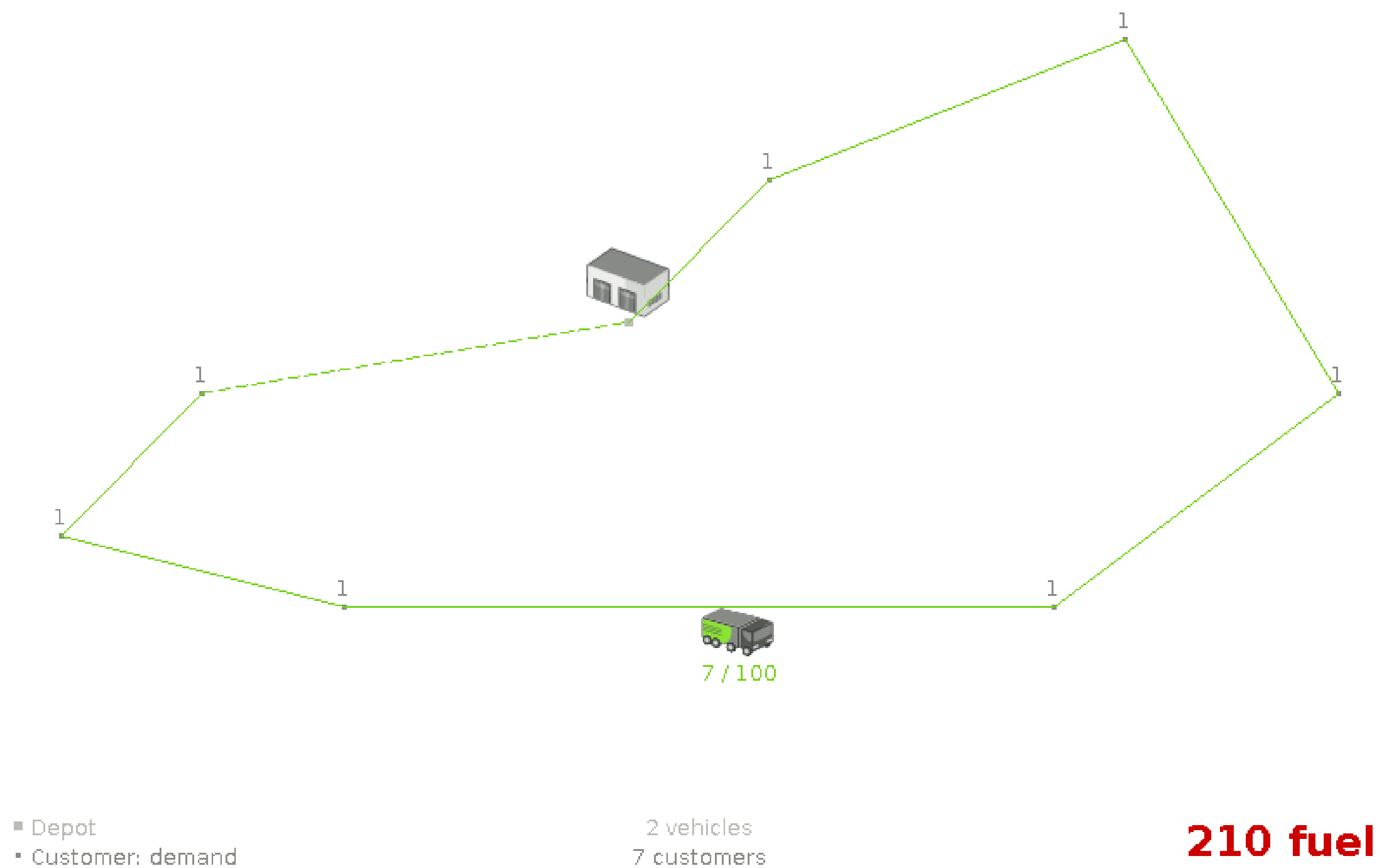

Map
Satellite
Middelburg
Vlissingen
Terneuzen
Bergen op Zoom
A12
Brecht
Eindhoven
Deurne
Valkenswaard
Venlo
E34
Weert
Lommel
Roermond
Kapellen
Ostend
Middelkerke
Aalter
Lokeren
Lier
Geel
Mol
Beringen
Erkelenz
Sittard
E25
Dunkirk
Dendermonde
Deinze
Aarschot
Aalst
Hasselt
E314
Heerlen
Oudenaarde
Sint-Truiden
Aachen
E40
Wavre
Hazebrouck
E15
Armentières
Roubaix
Lille
A3
Béthune
Haubourdin
Carvin
A23
E42
Saint-Amand-les-Eaux
Mons
Spa
Lens
Liévin
Douai
Valenciennes
Binche
Malmedy
Arras
Denain
Maubeuge
Dinant
E46
Marche-en-Famenne
Cambrai
Le Quesnoy
Aulnoye-Aymeries
Caudry
Givet
Fourmies
Albert
Péronne
Parc naturel régional des Ardennes
Hirson
Amiens
Corbie
Saint-Quentin
E44
E421
Charleville-Mézières
Sedan
A7
Luxembourg City
Tergnier
A16
Noyon
Chauny
Laon
A1
Virton
Longwy
A4
Rethel
Saint-Just-en-Chaussée
Compiègne
Map data ©2015 GeoBasis-DE/BKG (©2009), Google
Terms of Use
Report a map error

Integration with real maps

Google Maps or GraphHopper (OpenStreetMap) calculate distances, OptaPlanner optimizes the trips.

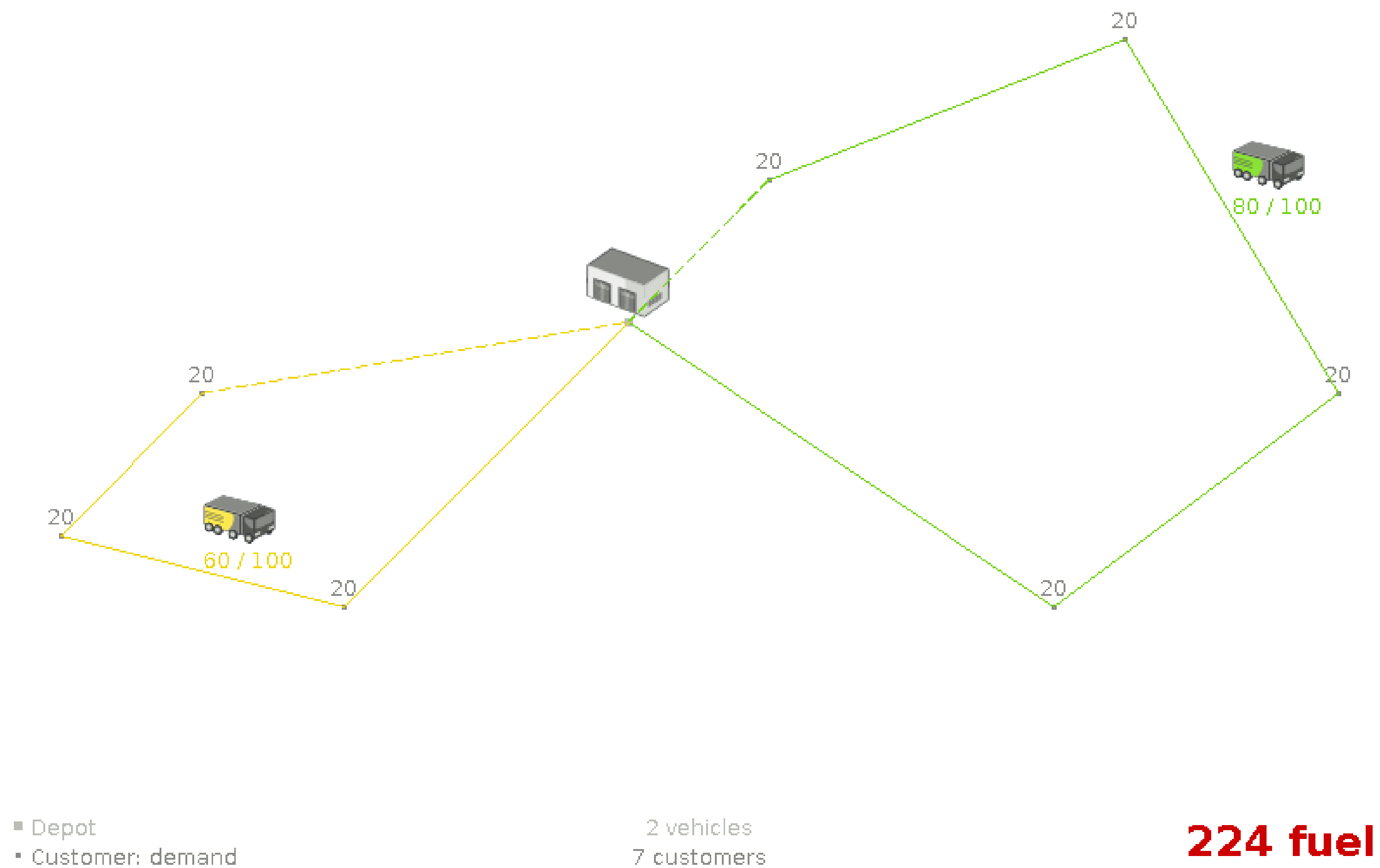


False presumptions VRP



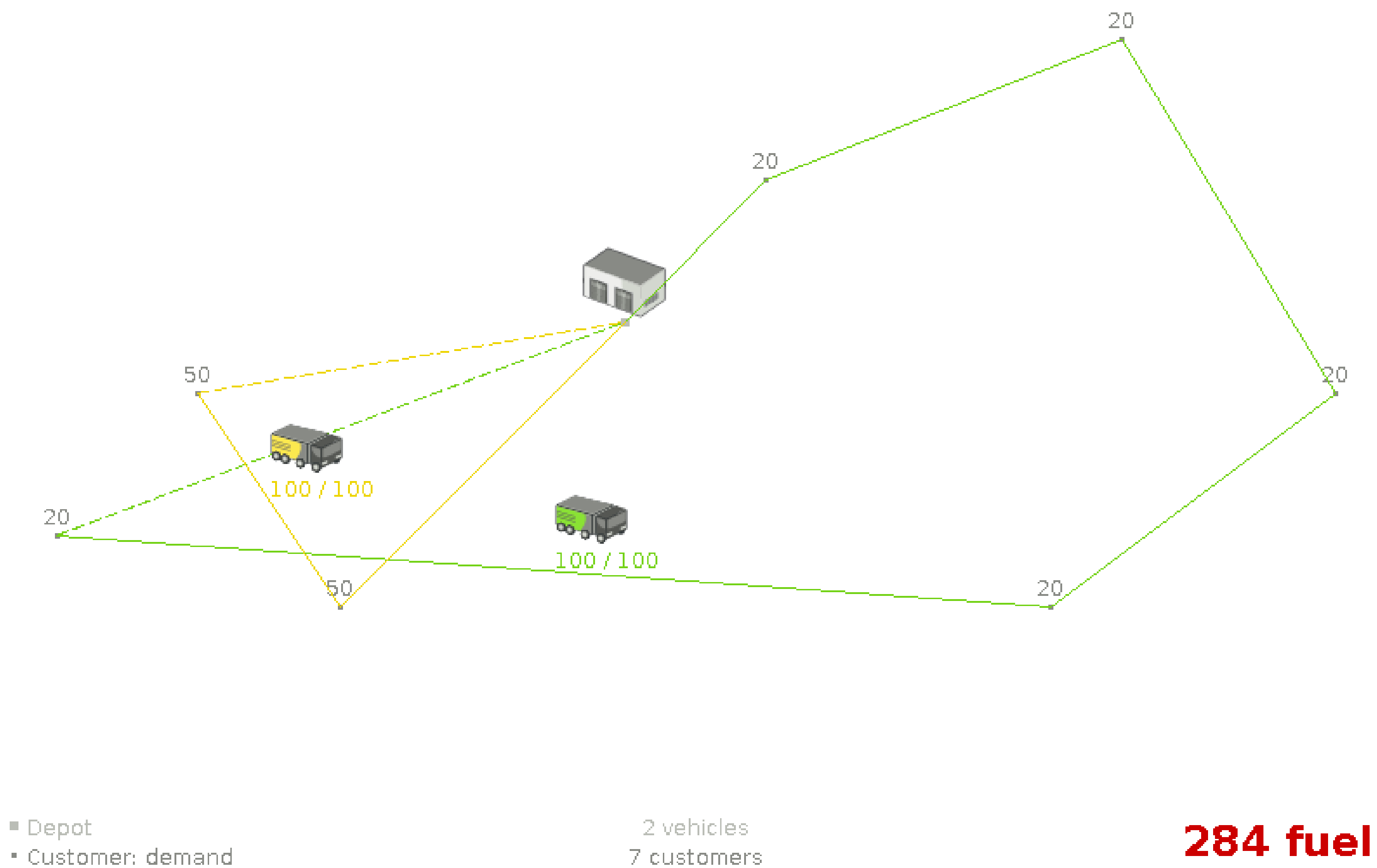
Assumption: **An optimal VRP route uses only 1 vehicle.**

Assumption: **An optimal VRP route uses only 1 vehicle. (false)**



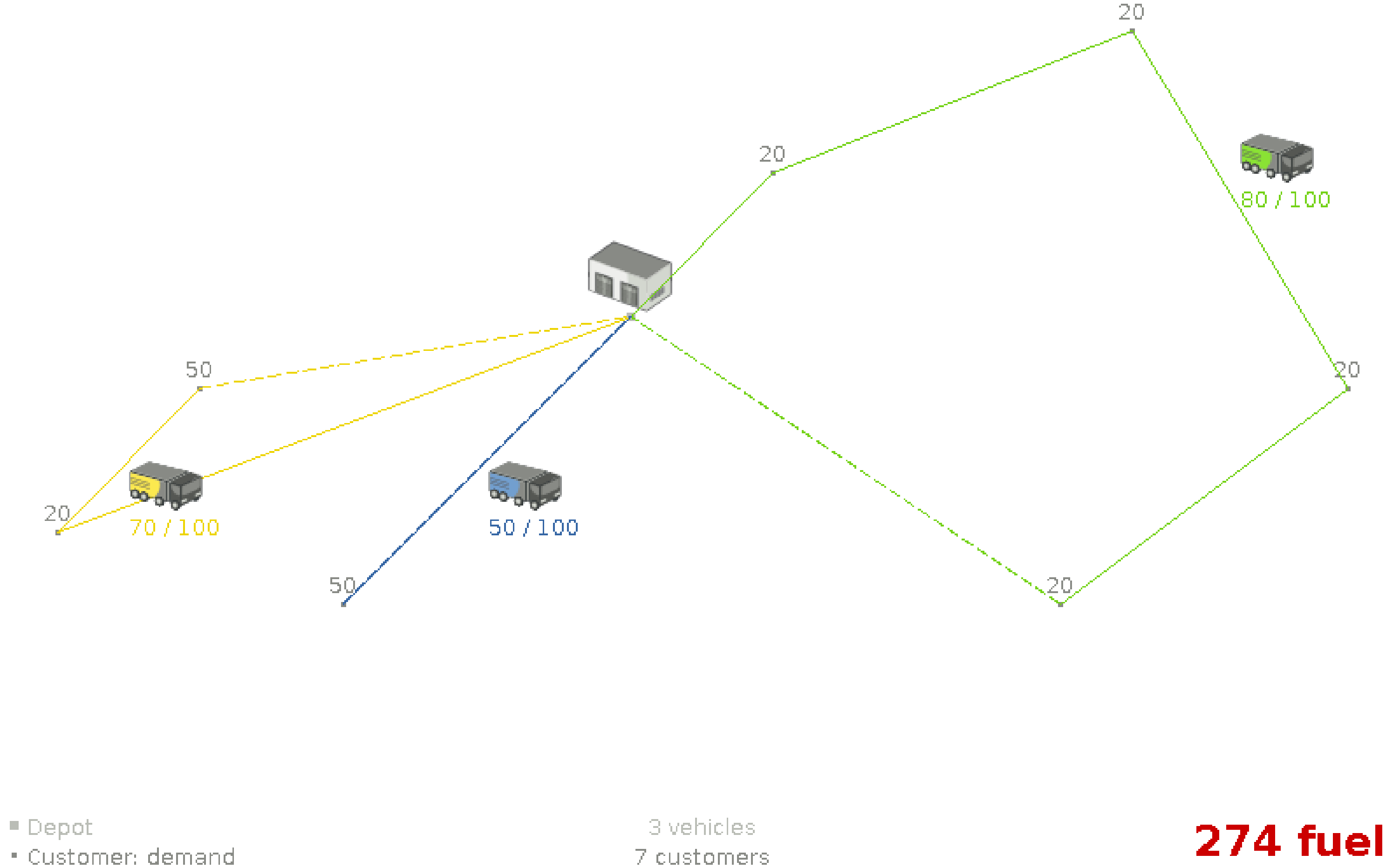
Assumption: **An optimal VRP route has no crossing lines.**

Assumption: **An optimal VRP route has no crossing lines. (false)**



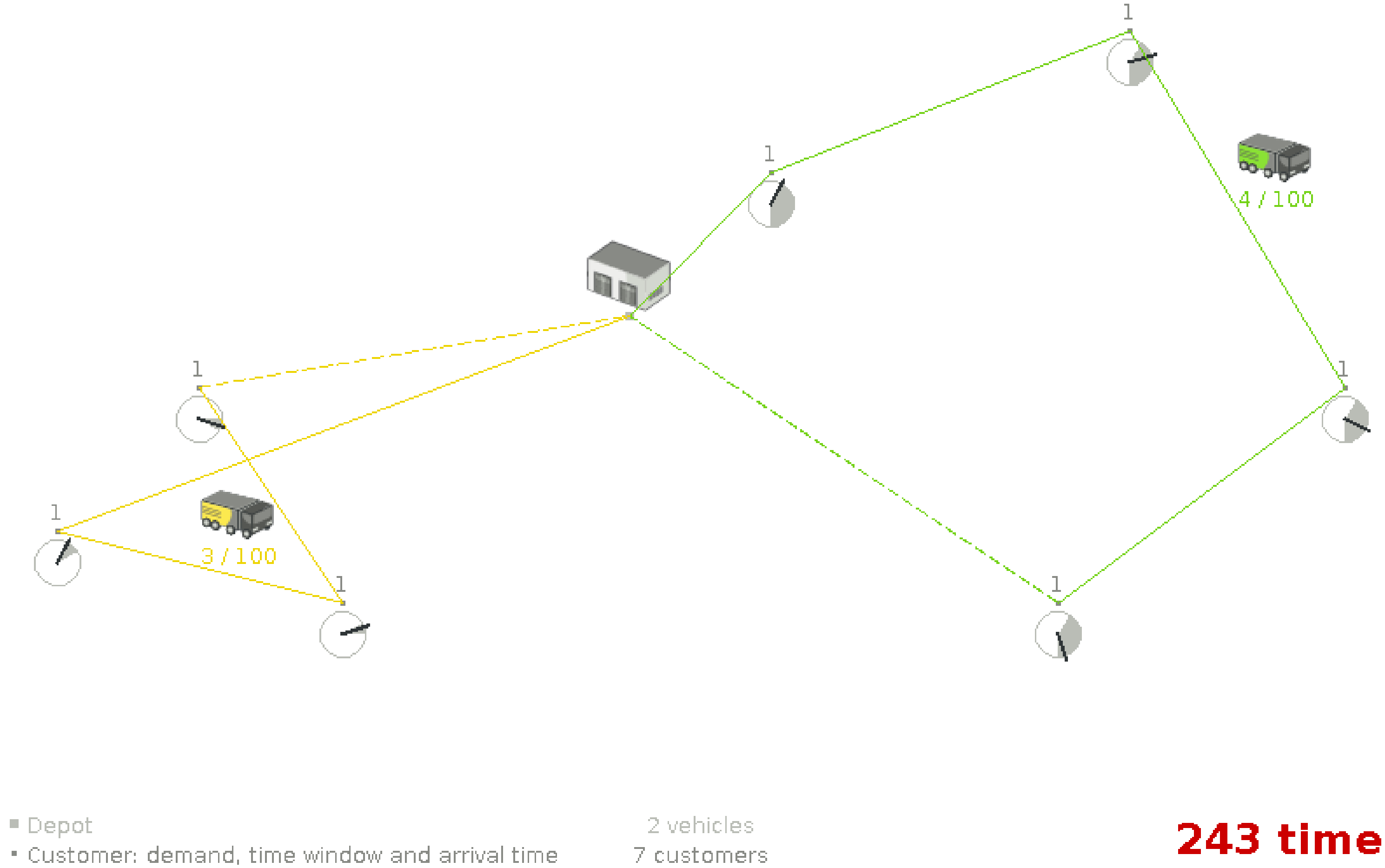
Assumption: **An optimal, feasible VRP route with n vehicles is still optimal for n+1 vehicles.**

Assumption: **An optimal, feasible VRP route with n vehicles is still optimal for n+1 vehicles. (false)**



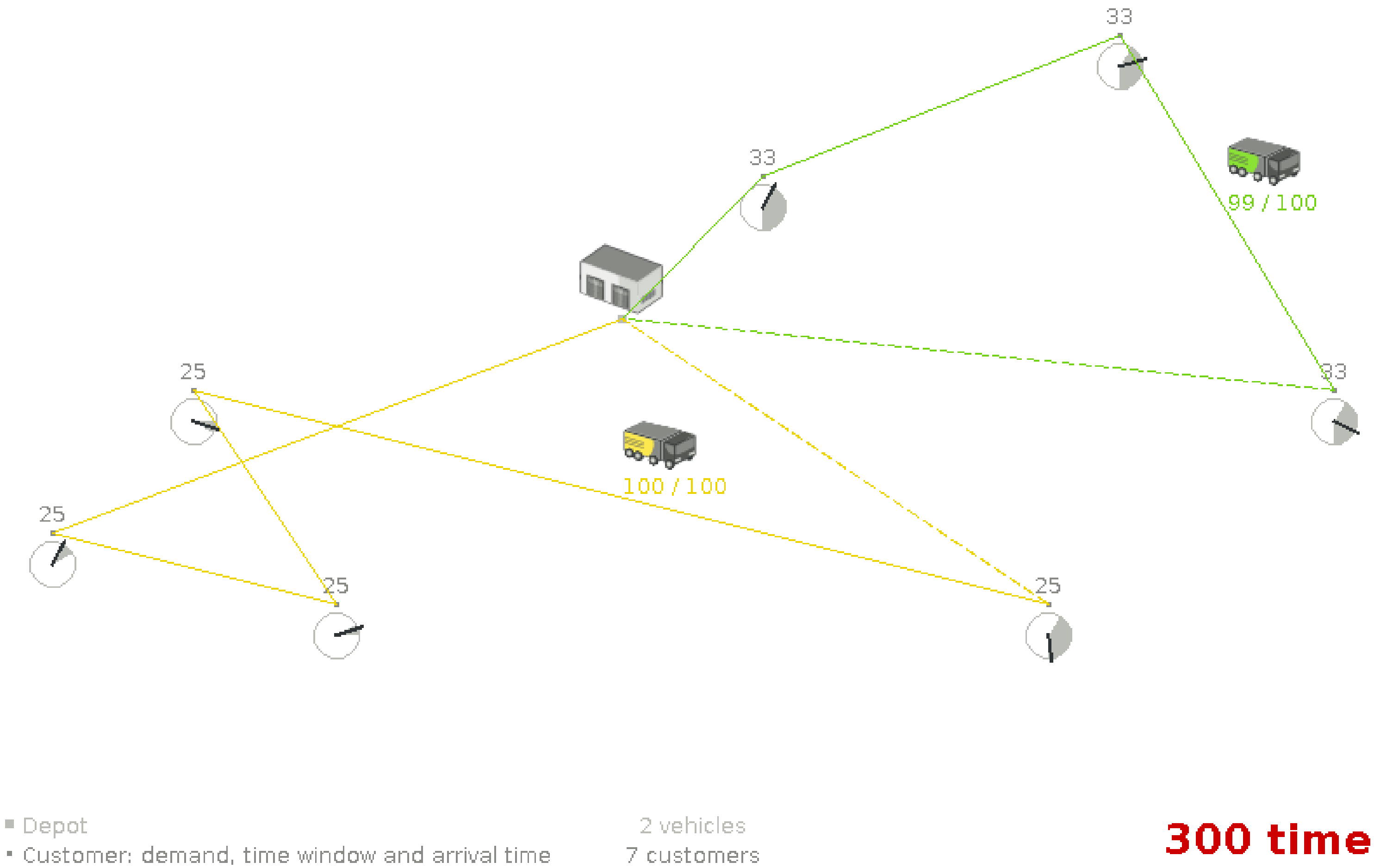
Assumption: **An optimal VRP route has no crossing lines of the same color.**

Assumption: **An optimal VRP route has no crossing lines of the same color. (false)**



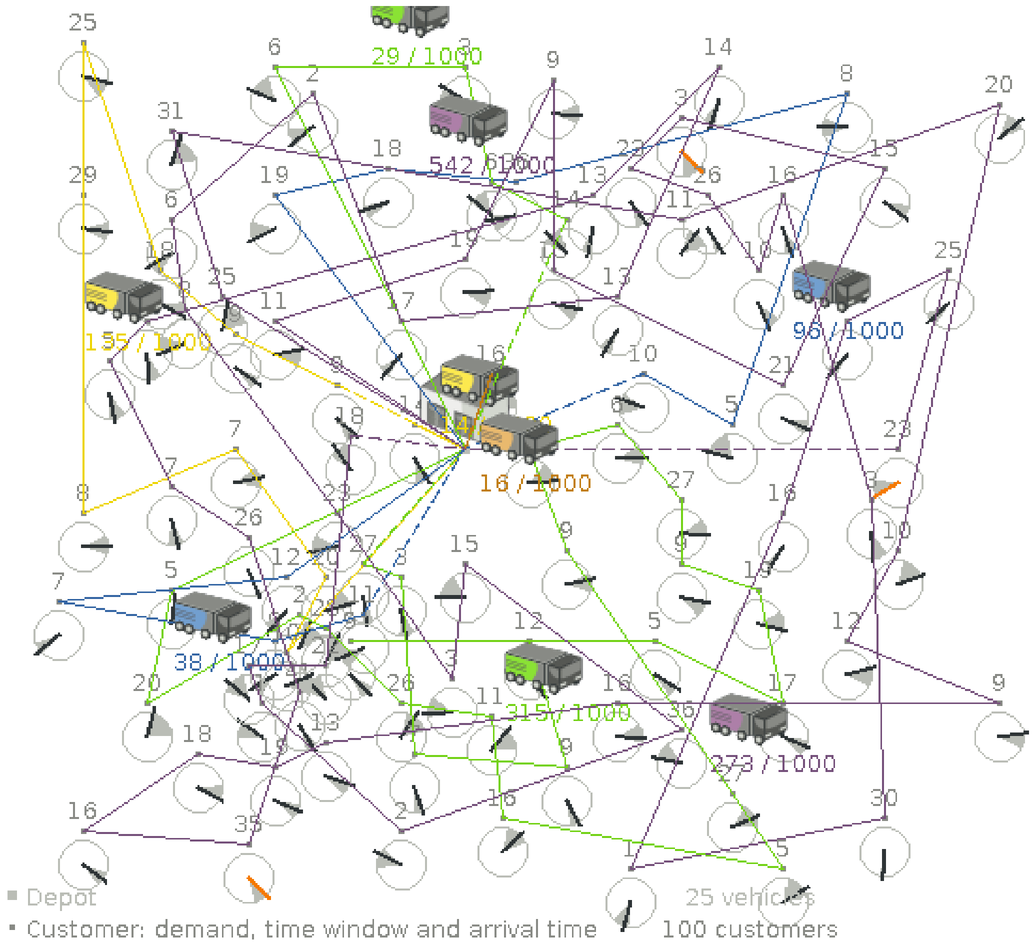
Assumption: **We can focus on time windows before focusing on capacity (or vice versa).**

Assumption: **We can focus on time windows before focusing on capacity (or vice versa). (false)**



Assumption: **Humans optimize VRP optimally.**

Assumption: **Humans optimize VRP optimally. (false)**



1,610 time

Can a manager reasonable judge if this is optimal?

What is a planning problem?

Optimize goals with limited resources under constraints

Optimize goals

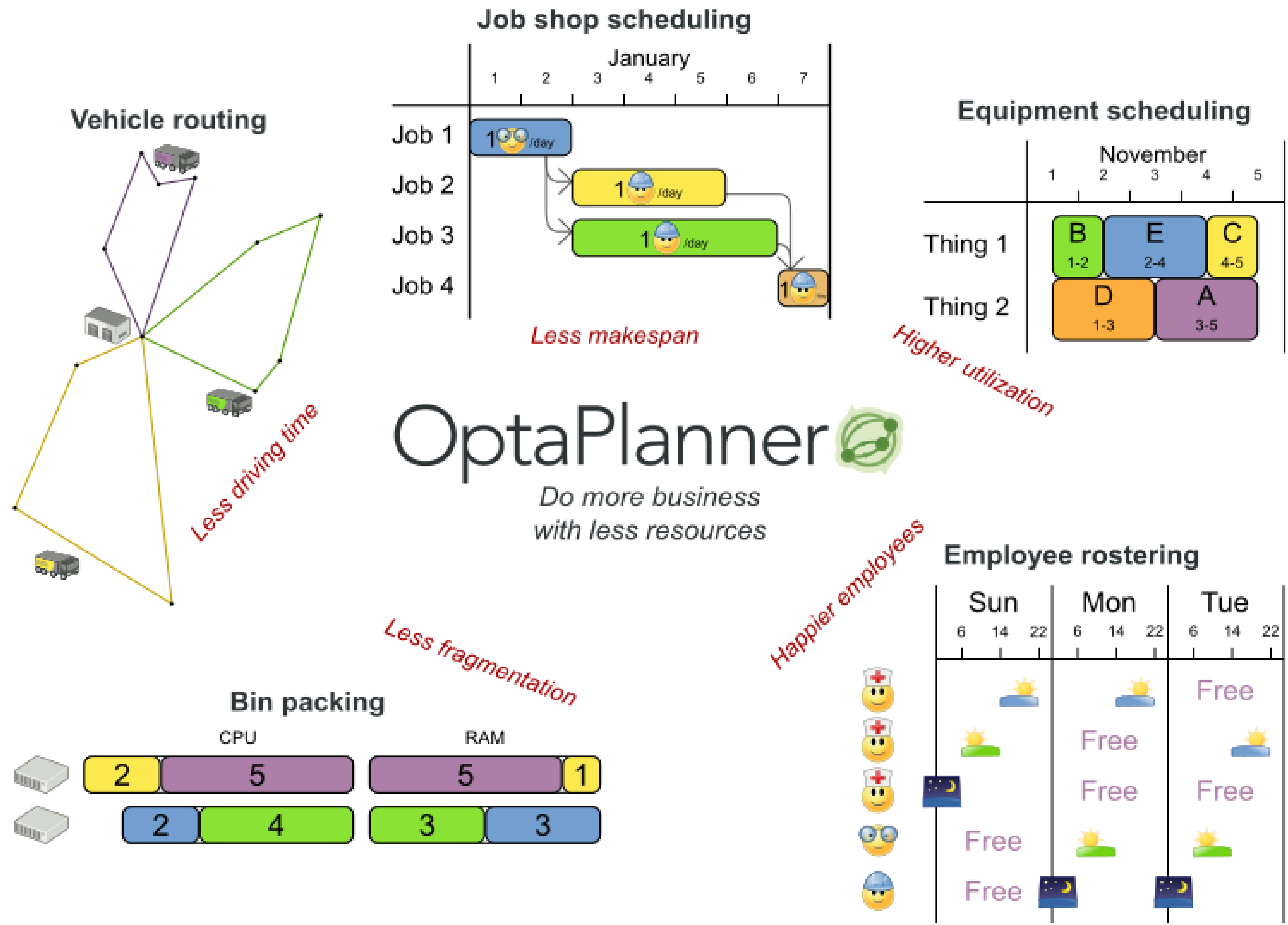
- Maximize profit
- Minimize ecological footprint
- Maximize happiness of employees / customers
- ...

With limited resources

- Employees
- Assets (machines, buildings, vehicles, ...)
- Time
- Budget

Under constraints

- vs Working hours
- vs Skills / affinity
- vs Logistic conflicts
- ...


Job shop scheduling
January
1 2 3 4 5 6 7
Job 1
Job 2
Job 3
Job 4
1 /day
1 /day
1 /day
1
Less makespan
Vehicle routing
Less driving time
Equipment scheduling
November
1 2 3 4 5
Thing 1
Thing 2
B 1-2
E 2-4
C 4-5
D 1-3
A 3-5
Higher utilization
OptaPlanner
Do more business
with less resources
Happier employees
Employee rostering
Sun
Mon
Tue
6 14 22
Free
Less fragmentation
Bin packing
CPU
RAM
2
5
5
1
2
4
3
3

Planning problem use cases

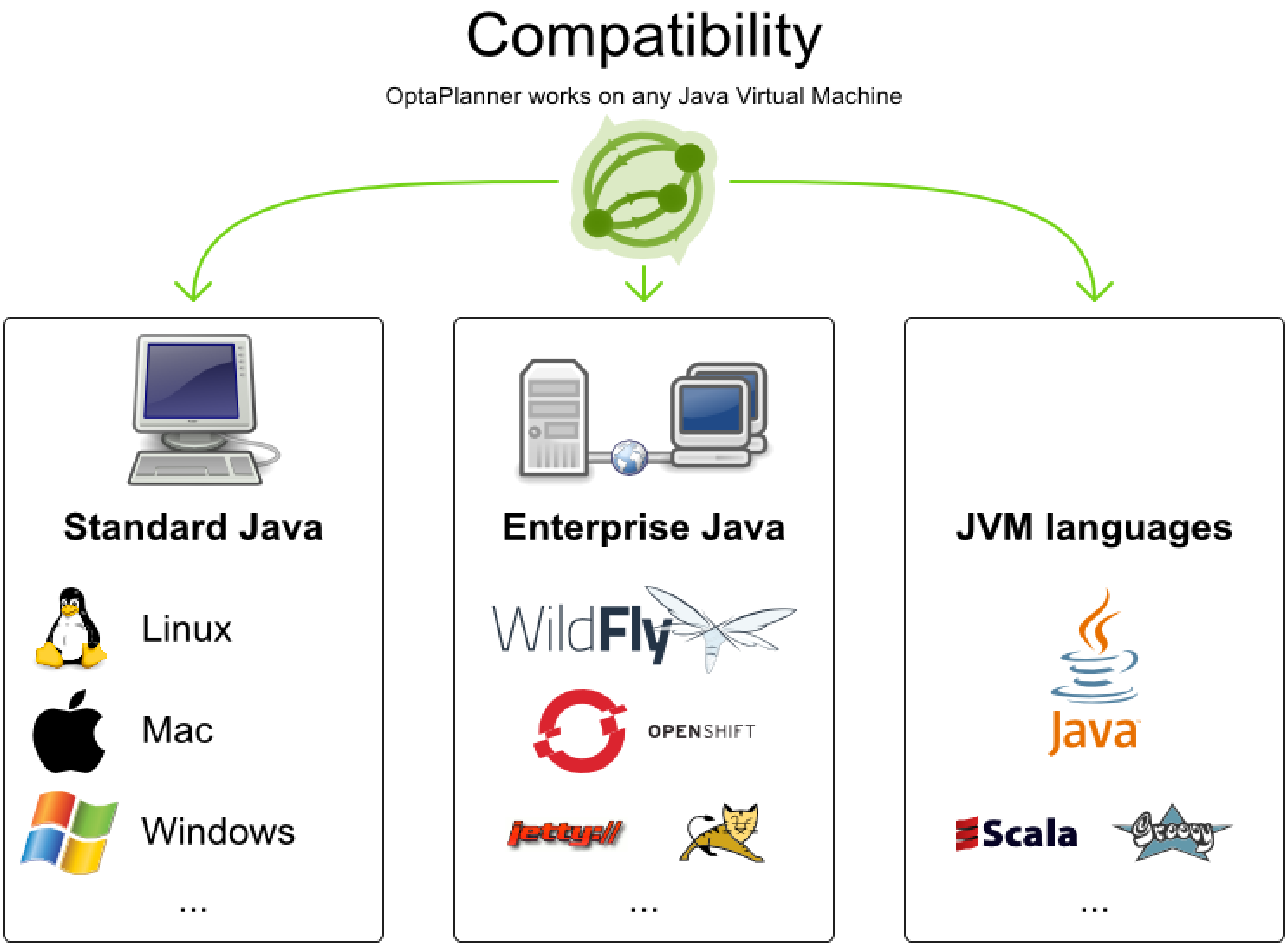
- **Agenda scheduling**: doctor appointments, court hearings, maintenance jobs, TV advertisements, ...
- **Educational timetabling**: lectures, exams, conference presentations, ...
- **Task assignment**: affinity/skill matchmaking for tax audits, wage calc, ...
- **Employee shift rostering**: nurses, repairmen, help desk, firemen, ...
- **Vehicle routing**: route trucks, buses, trains, boats, airplanes, ...
- **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, ...

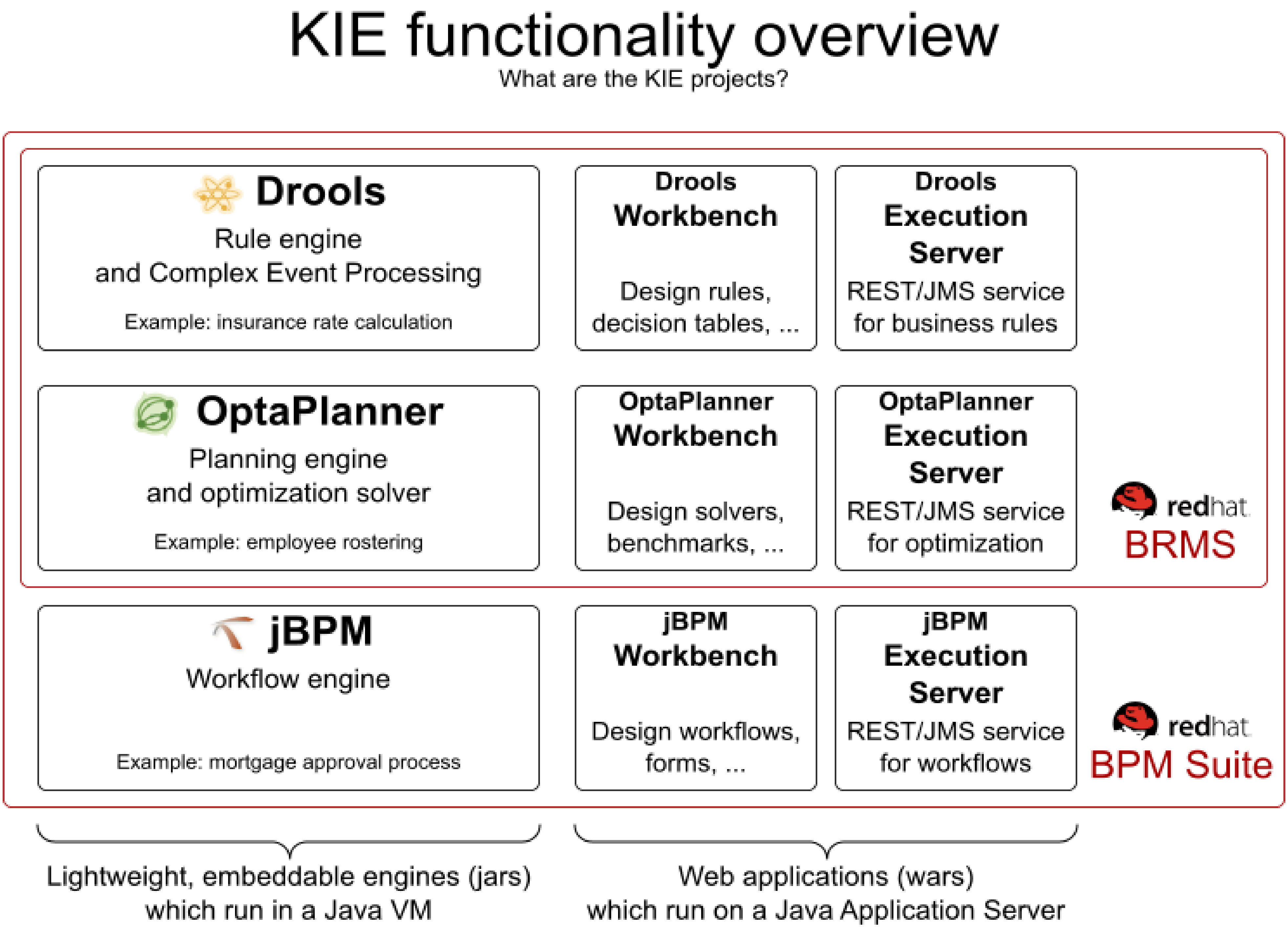
Reuse optimization algorithms

OptaPlanner

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

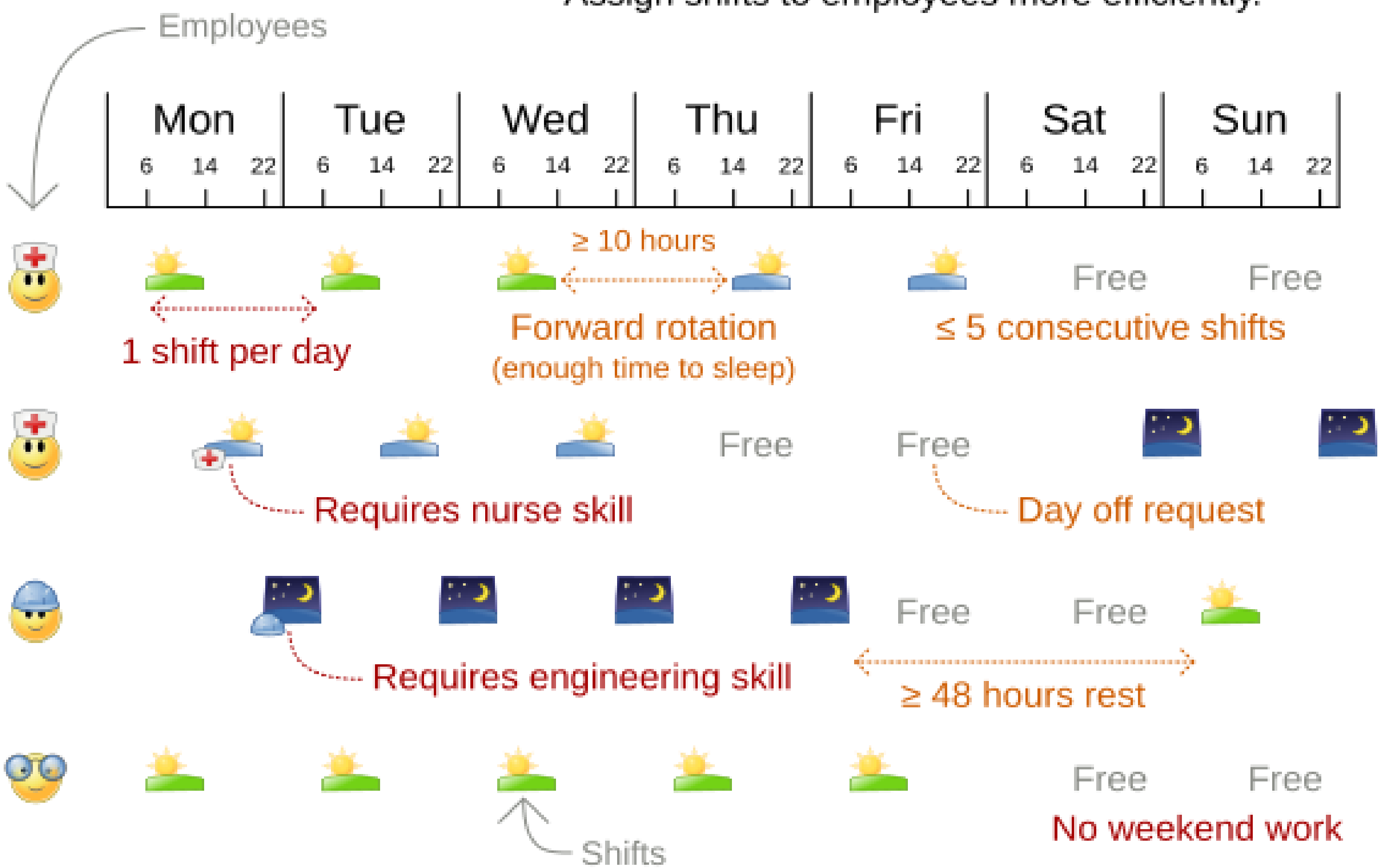

Compatibility
OptaPlanner works on any Java Virtual Machine
Standard Java
Linux
Mac
Windows
...
Enterprise Java
WildFly
OPENSHIFT
jetty://
...
JVM languages
Java
Scala
Groovy
...


KIE functionality overview
What are the KIE projects?
Drools
Rule engine
and Complex Event Processing
Example: insurance rate calculation
Drools
Workbench
Design rules,
decision tables, ...
Drools
Execution
Server
REST/JMS service
for business rules
OptaPlanner
Planning engine
and optimization solver
Example: employee rostering
OptaPlanner
Workbench
Design solvers,
benchmarks, ...
OptaPlanner
Execution
Server
REST/JMS service
for optimization
redhat.
BRMS
jBPM
Workflow engine
Example: mortgage approval process
jBPM
Workbench
Design workflows,
forms, ...
jBPM
Execution
Server
REST/JMS service
for workflows
redhat.
BPM Suite
Lightweight, embeddable engines (jars)
which run in a Java VM
Web applications (wars)
which run on a Java Application Server

Employee shift rostering

Employee rostering

Assign shifts to employees more efficiently.



Users

Hospitals

Courts of Justice

Call centers

Police and fire department

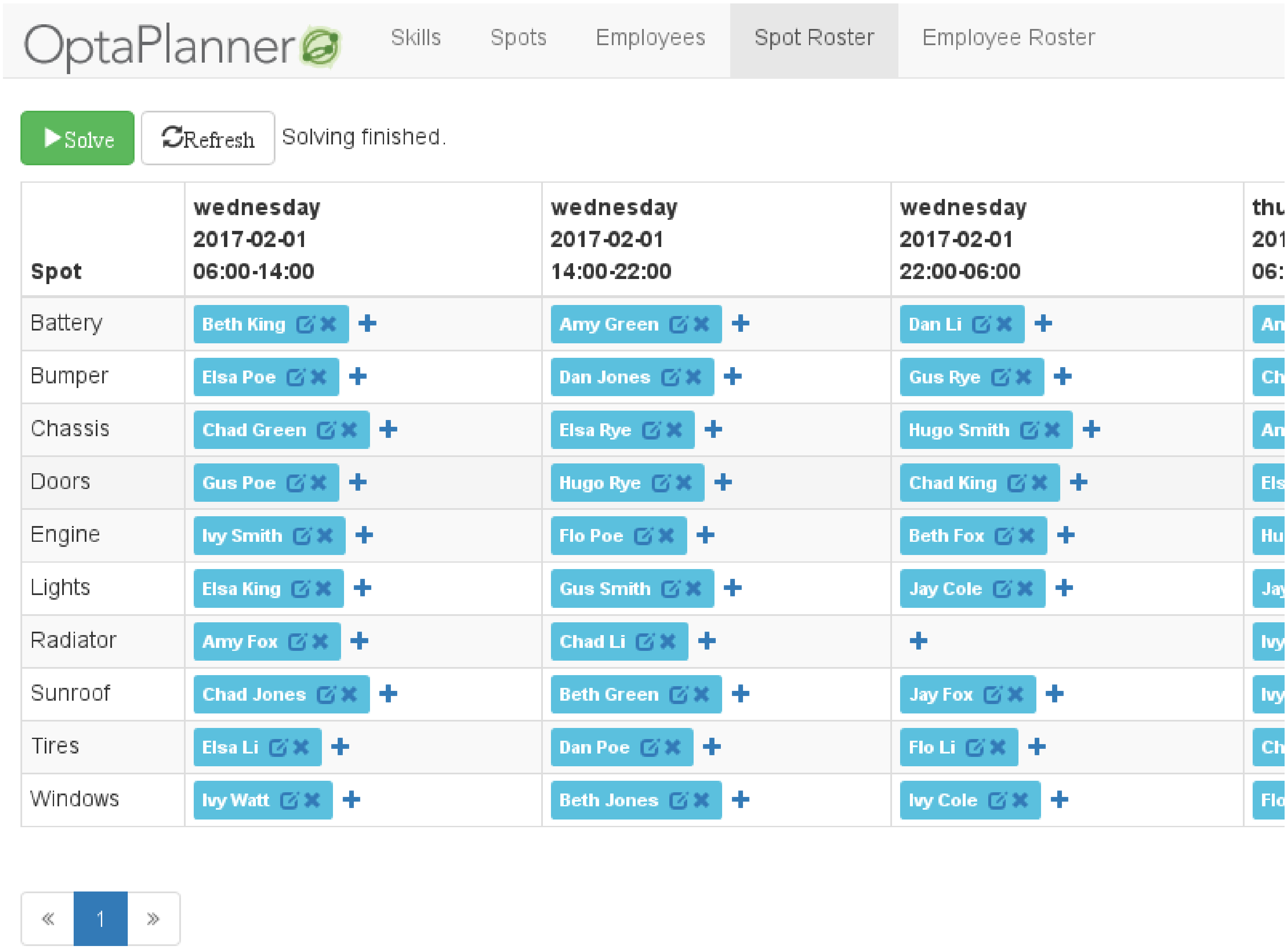
NurseRostering benchmark	Average	Min/Max	# datasets	Biggest dataset
Employee well-being	**+53%**	+19% +85%	26	752 assignments 50 employees

OptaPlanner versus traditional algorithm with domain knowledge

5 mins Tabu Search vs First Fit Decreasing

Don't believe us? Run our open benchmarks yourself: https://www.optaplanner.org/code/benchmarks.html

What is constraint solving?

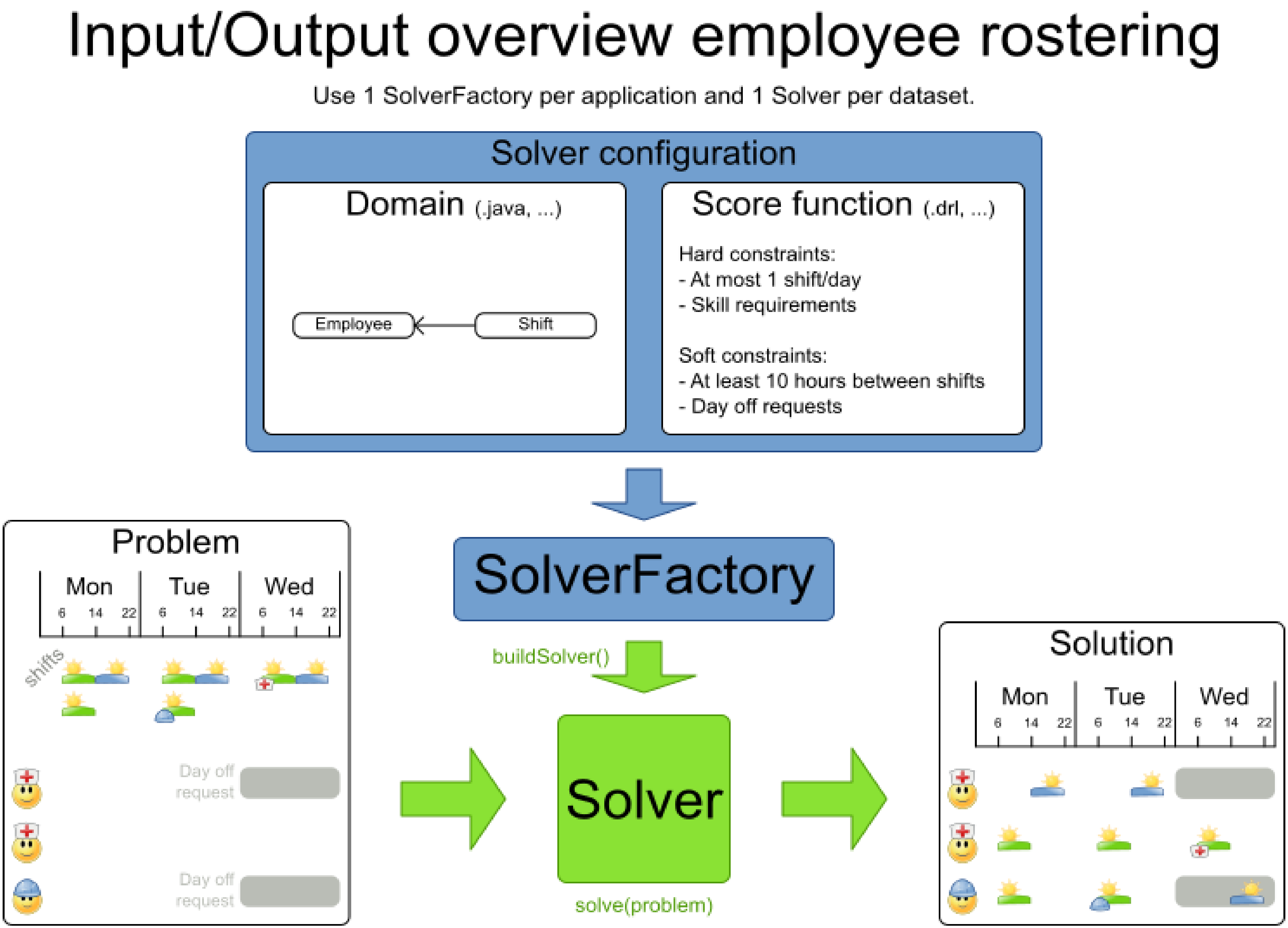


Spot	wednesday 2017-02-01 06:00-14:00	wednesday 2017-02-01 14:00-22:00	wednesday 2017-02-01 22:00-06:00	thu 201 06:
Battery	Beth King	Amy Green	Dan Li	An
Bumper	Elsa Poe	Dan Jones	Gus Rye	Ch
Chassis	Chad Green	Elsa Rye	Hugo Smith	An
Doors	Gus Poe	Hugo Rye	Chad King	Els
Engine	Ivy Smith	Flo Poe	Beth Fox	Hu
Lights	Elsa King	Gus Smith	Jay Cole	Jay
Radiator	Amy Fox	Chad Li		Ivy
Sunroof	Chad Jones	Beth Green	Jay Fox	Ivy
Tires	Elsa Li	Dan Poe	Flo Li	Ch
Windows	Ivy Watt	Beth Jones	Ivy Cole	Flo

OptaShift Employee Rostering demo

Implementation

1. Define domain
2. Define constraints
3. Solve

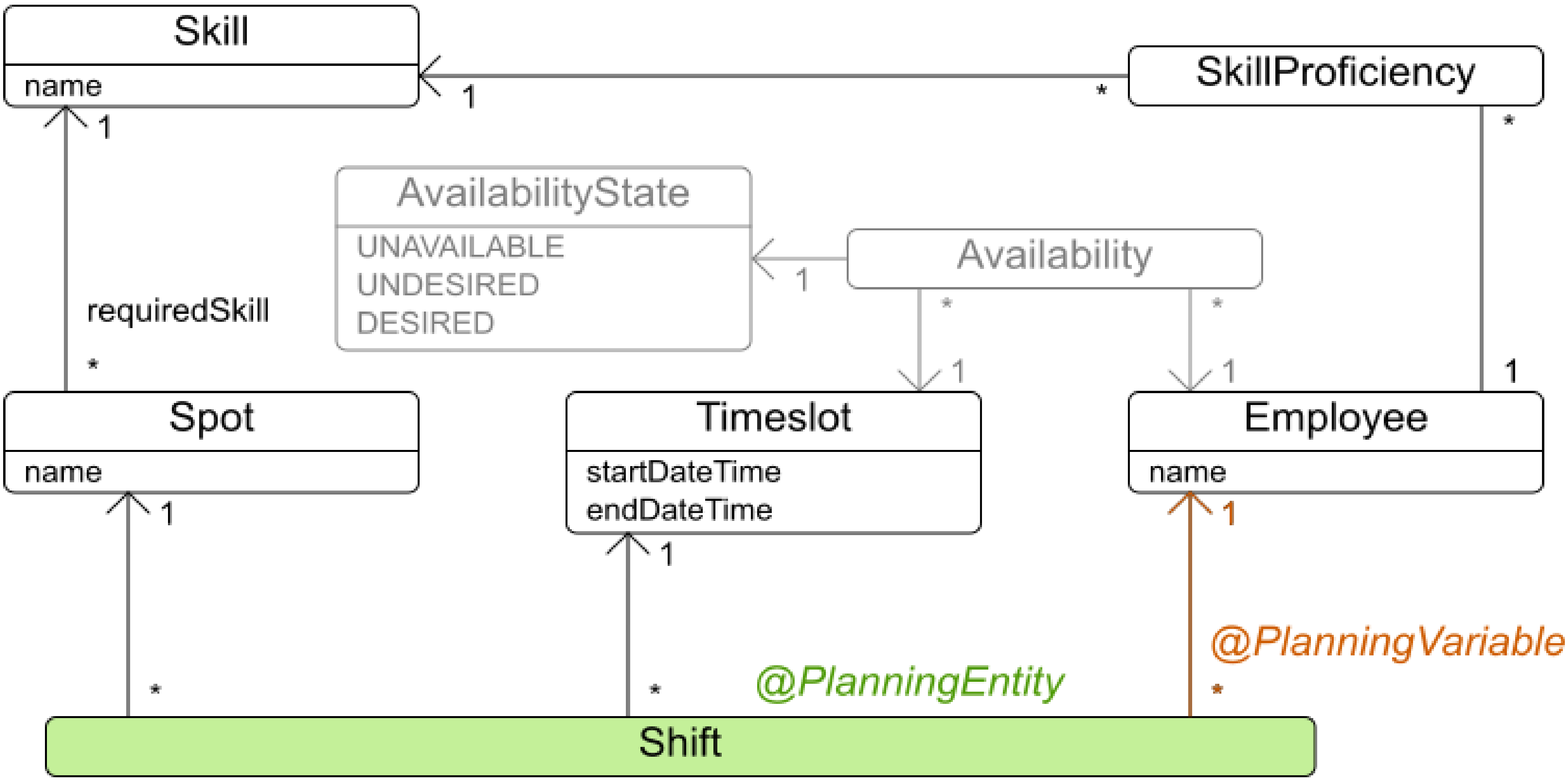

Input/Output overview employee rostering
Use 1 SolverFactory per application and 1 Solver per dataset.
Solver configuration
Domain (.java, ...)
Employee
Shift
Score function (.drl, ...)
Hard constraints:
- At most 1 shift/day
- Skill requirements
Soft constraints:
- At least 10 hours between shifts
- Day off requests
Problem
Mon
Tue
Wed
6
14
22
shifts
Day off request
Day off request
SolverFactory
buildSolver()
Solver
solve(problem)
Solution
Mon
Tue
Wed
6
14
22

Calling the solve method

```
// My domain specific class as input
Roster problem = ...;

SolverFactory<Roster> factory = SolverFactory
    .createFromXmlResource(".../mySolverConfig.xml");
Solver<Roster> solver = factory.buildSolver();

// My domain specific class as output
Roster solution = solver.solve(problem);

for (Shift shift : solution.getShiftList()) {
  // Each shift is now assigned to an employee
  System.out.println(shift + " assigned to " + shift.getEmployee());
}
```

Define domain

Employee rostering class diagram



Spot.java

```
public class Spot {

  private String name;
  private Skill requiredSkill;

  ...

}
```

Plain old java class

Employee.java

```
public class Employee {

  private String name;

  private List<Skill> skillList;

  ...

  public boolean hasSkill(Skill skill) {
    return skillList.contains(skill);
  }

}
```

Plain old java class

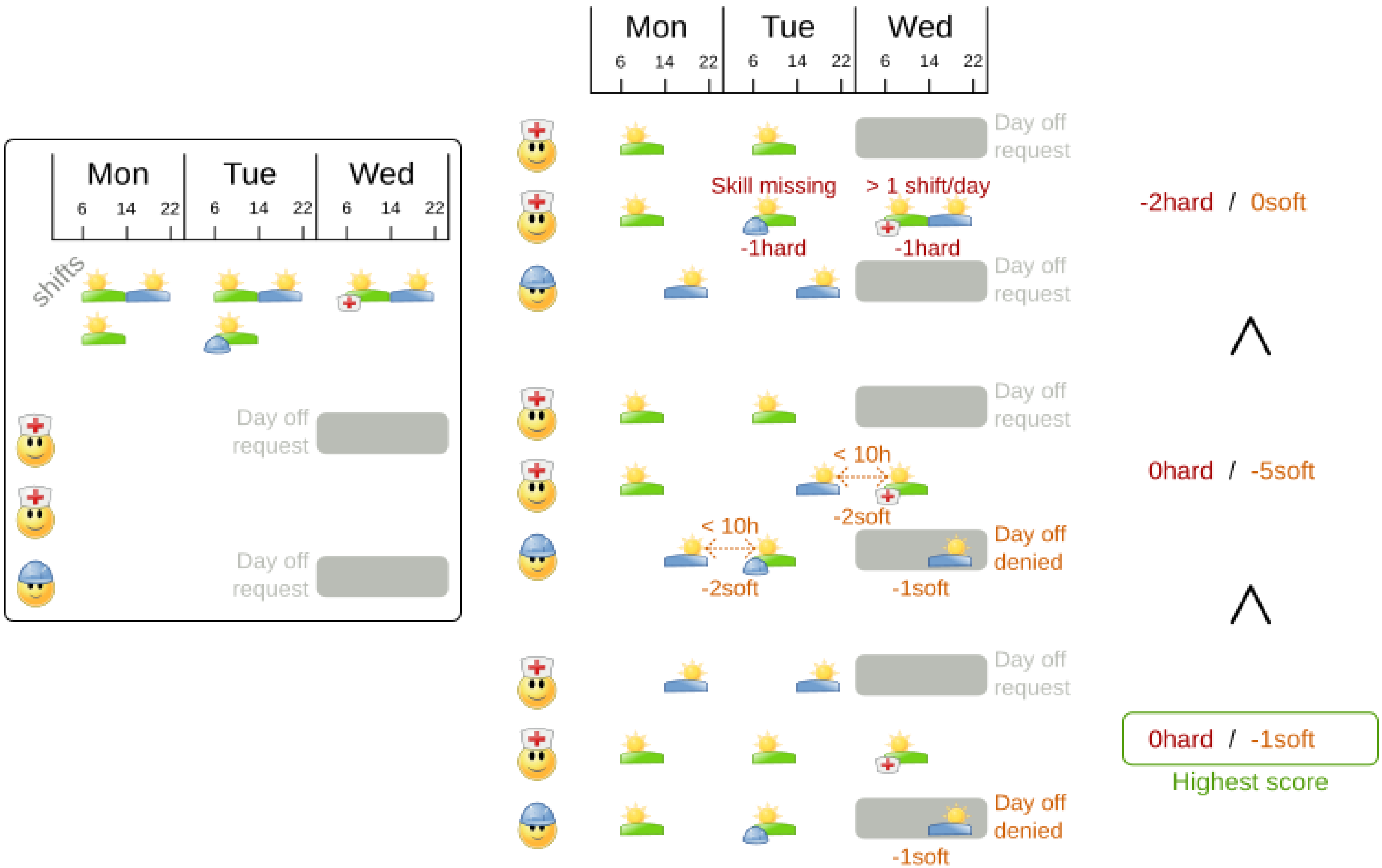
Shift.java

```java
@PlanningEntity
public class Shift {

  private Spot spot;
  private TimeSlot timeSlot;

  @PlanningVariable(...)
  private Employee employee; // Changes during solve()

  ...

}
```

Plain old java class with OptaPlanner annotations

Score Comparison Employee Rostering

Hard constraints always outweight soft constraints.



Score calculation

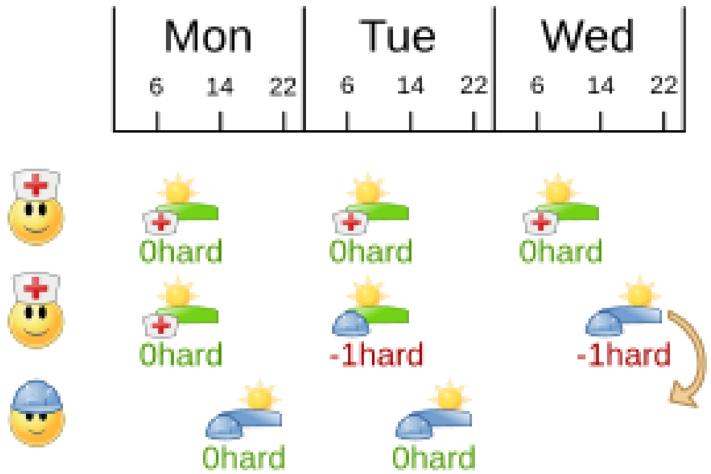
- Easy Java
- Incremental Java
- Drools DRL (also incremental)

Required skill constraint (easy Java)

```
public class MyScoreCalculator
    implements EasyScoreCalculator<Roster> {

  public Score calculateScore(Roster roster) {
    int hardScore = 0;
    for (Shift shift : roster.getShiftList()) {
      Skill requiredSkill = shift.getSpot().getRequiredSkill();
      if (shift.getEmployee() != null
          // Employee lacks required skill
          && !shift.getEmployee().hasSkill(requiredSkill)) {
        // Lower hard score
        hardScore--;
      }
    }
    ...
    return HardSoftScore.valueOf(hardScore, softScore);
  }
```

Incremental score calculation

Calcuting delta's is much faster than calculating the entire's solution's score.



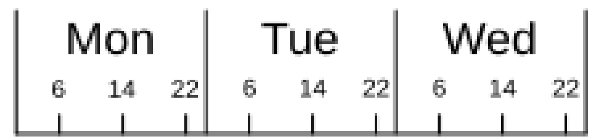
Check every shift:

0 + 0 + 0 + 0 - 1 - 1 + 0 + 0

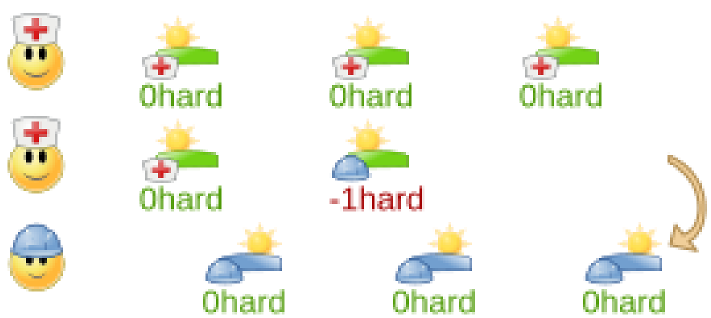
Required skill score: -2hard

BigO for n shifts

Constraint	**From scratch**	**Incremental**
Required skill	O(n)	O(1)
At most 1 shift/day	O(n²)	O(n)
...	...	...



Calculation from scratch (easy java)



Check every shift again:

0 + 0 + 0 + 0 - 1 + 0 + 0 + 0

Required skill score: -1hard

Incremental calculation (inc. java, drools)

Check one shift (old & new)

-2 + 1 - 0

Required skill score: -1hard

Required skill constraint (Drools DRL)

```
rule "Required skill"
  when
    Shift(
        getEmployee() != null,
        // Employee lacks required skill
        !getEmployee().hasSkill(getSpot().getRequiredSkill()))
  then
    // Lower hard score
    scoreHolder.addHardConstraintMatch(kcontext, -1);
end
```

Availability.java

```
public class Availability {

  private Employee employee;
  private TimeSlot timeSlot;

  ...

}
```

Plain old java class

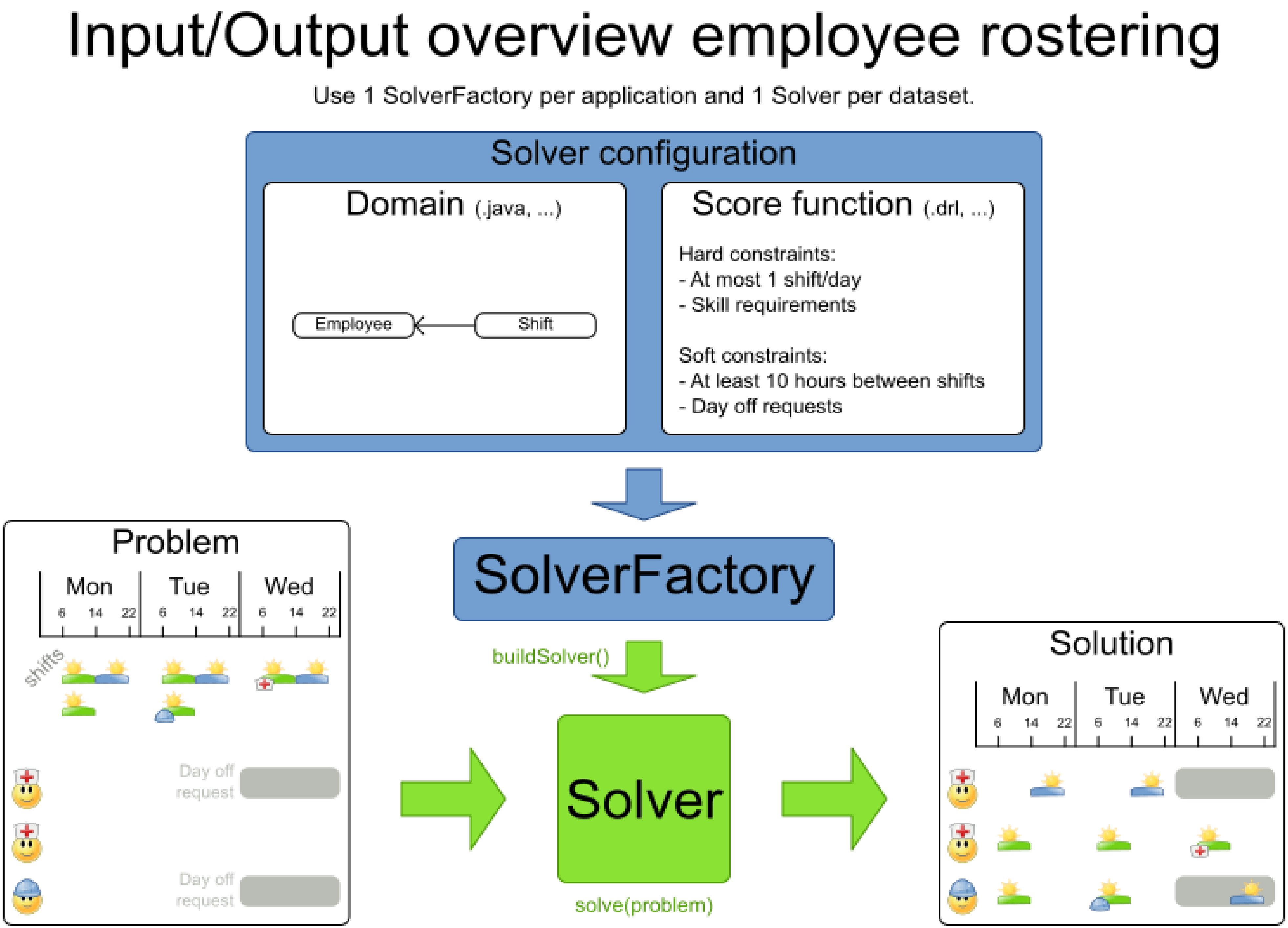
All instances handled by only one constraint

Time off request constraint (easy Java)

```
public class MyScoreCalculator
    implements EasyScoreCalculator<Roster> {

  public Score calculateScore(Roster roster) {
    ...
    int softScore = 0;
    for (Availability availability : roster.getAvailabilityList()) {
      for (Shift shift : roster.getShiftList()) {
          if (shift.getEmployee() == availability.getEmployee()
              && shift.getTimeSlot() == availability.getTimeSlot()) {
            // Lower soft score
            softScore--;
          }
      }
    }
    return HardSoftScore.valueOf(hardScore, softScore);
  }
```

Time off request constraint (Drools DRL)

```
rule "Time off request"
  when
    Availability(
        state == AvailabilityState.UNDESIRED,
        $e : employee,
        $t : timeSlot)
    Shift(
        employee == $e,
        timeSlot == $t)
  then
    // Lower soft score
    scoreHolder.addSoftConstraintMatch(kcontext, -1);
end
```



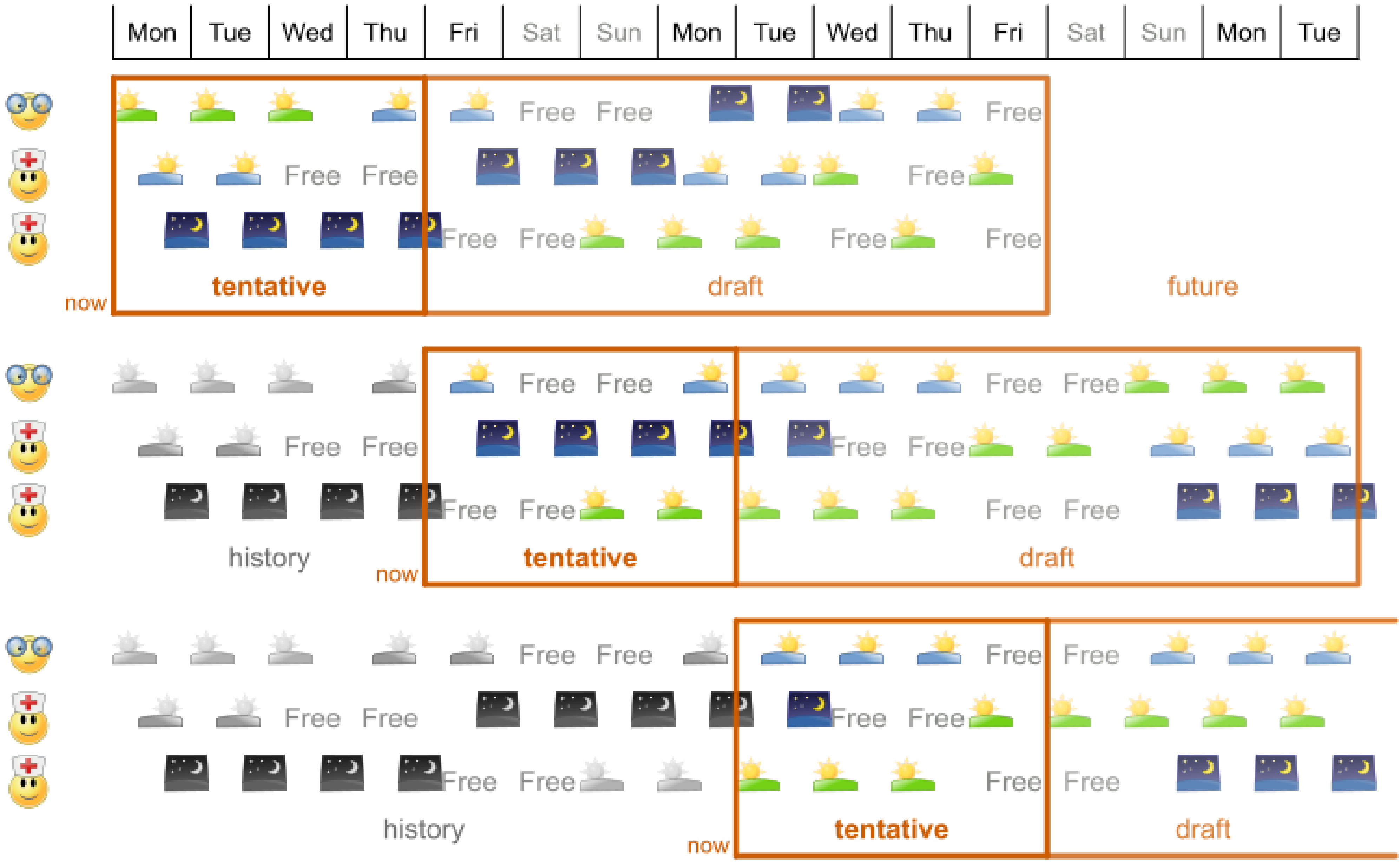
Input/Output overview employee rostering
Use 1 SolverFactory per application and 1 Solver per dataset.
Solver configuration
Domain (.java, ...)
Employee
Shift
Score function (.drl, ...)
Hard constraints:
- At most 1 shift/day
- Skill requirements
Soft constraints:
- At least 10 hours between shifts
- Day off requests
SolverFactory
buildSolver()
Solver
solve(problem)
Problem
Mon
Tue
Wed
6
14
22
shifts
Day off request
Day off request
Solution
Mon
Tue
Wed
6
14
22

When do we solve?

- Publish schedule weeks in advance
  - Affects family/social lives
- Ad hoc changes
  - Sick employees
  - Shift changes

Continuous planning

Replan at the start of every period. Plan 3 periods, but only share the first period.



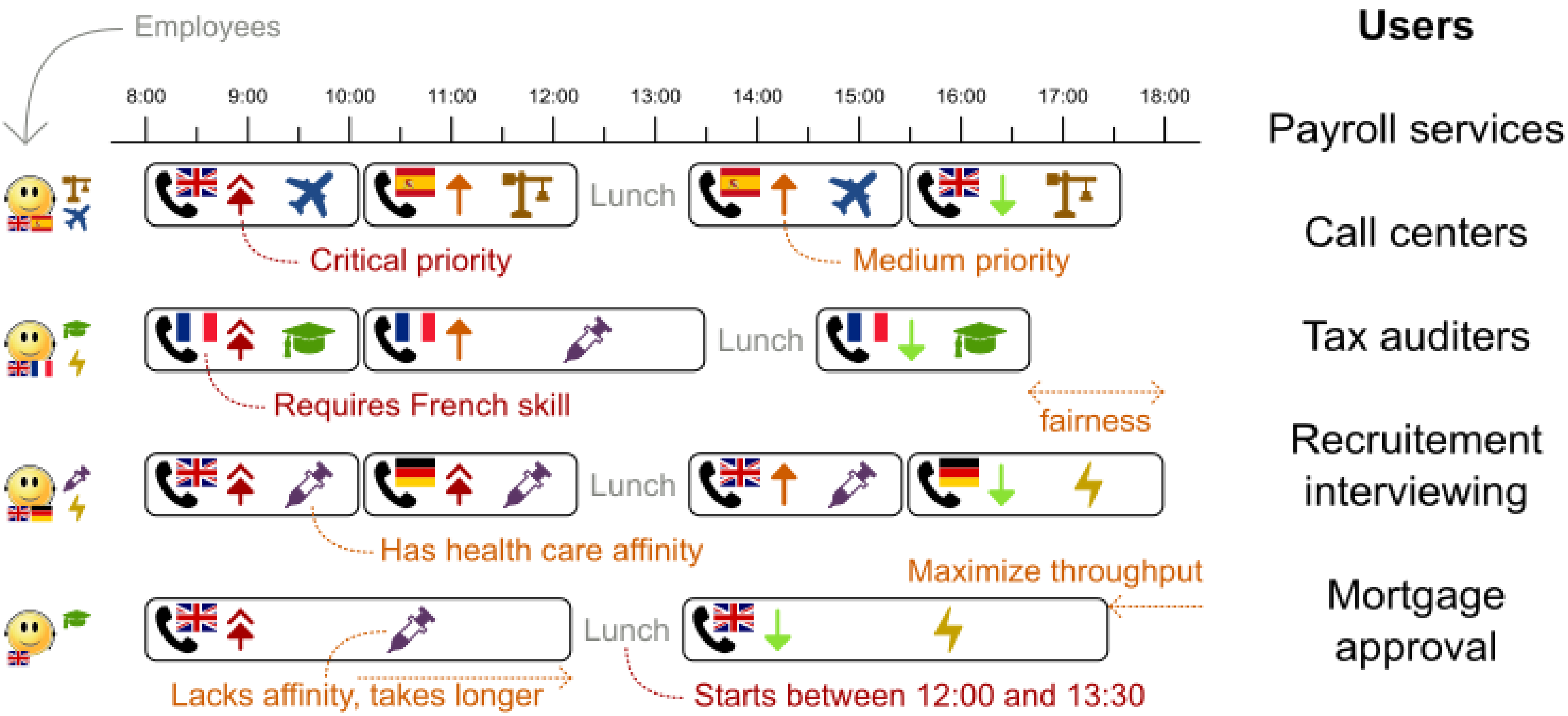
Who's in control?

The user

Task assigning

Task assigning

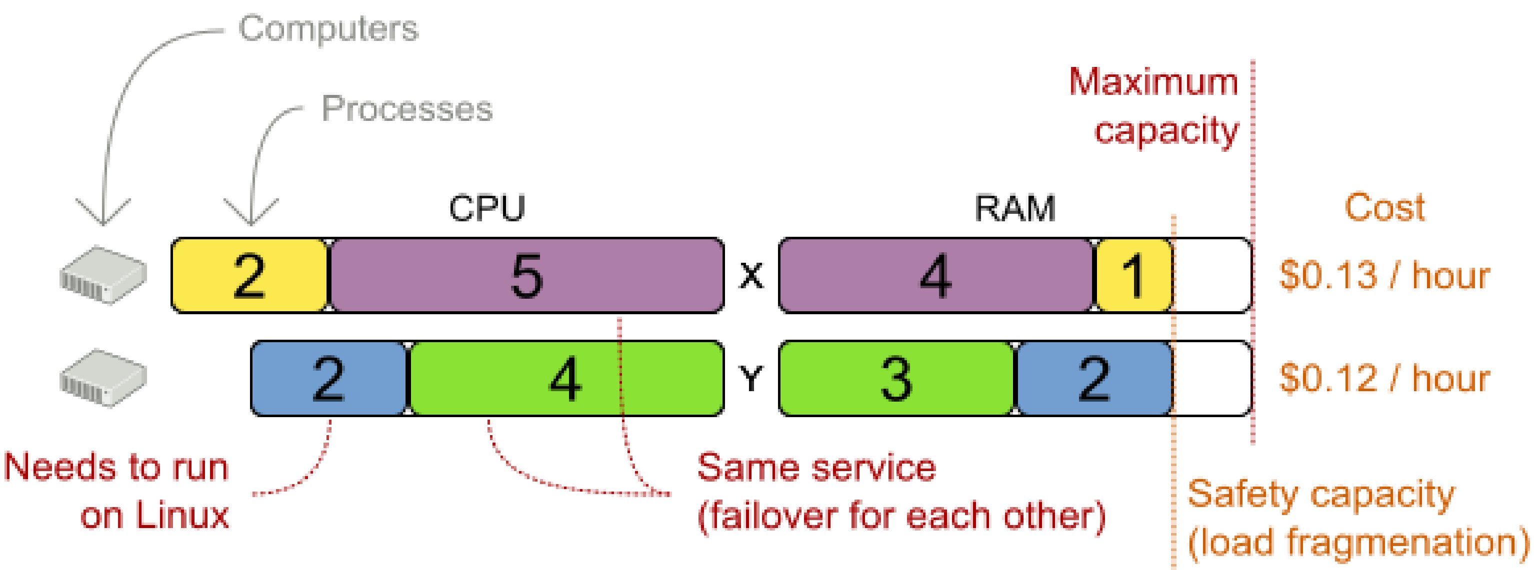
Optimize the task queue of every employee by reassigning and reordering tasks.



Cloud optimization

Cloud optimization

Assign processes to machines more efficiently.



Users

oVirt

CloudBalancing benchmark	Average	Min/Max	# datasets	Biggest dataset
Cloud hosting cost	-18%	-16% -21%	5	1600 computers 4800 processes

OptaPlanner versus traditional algorithm with domain knowledge

5 mins Simulated Annealing vs First Fit Decreasing

MachineReassignment benchmark	Average	Min/Max	# datasets	Biggest dataset
Hardware congestion	-63%	-25% -97%	20	50k machines 5k processes

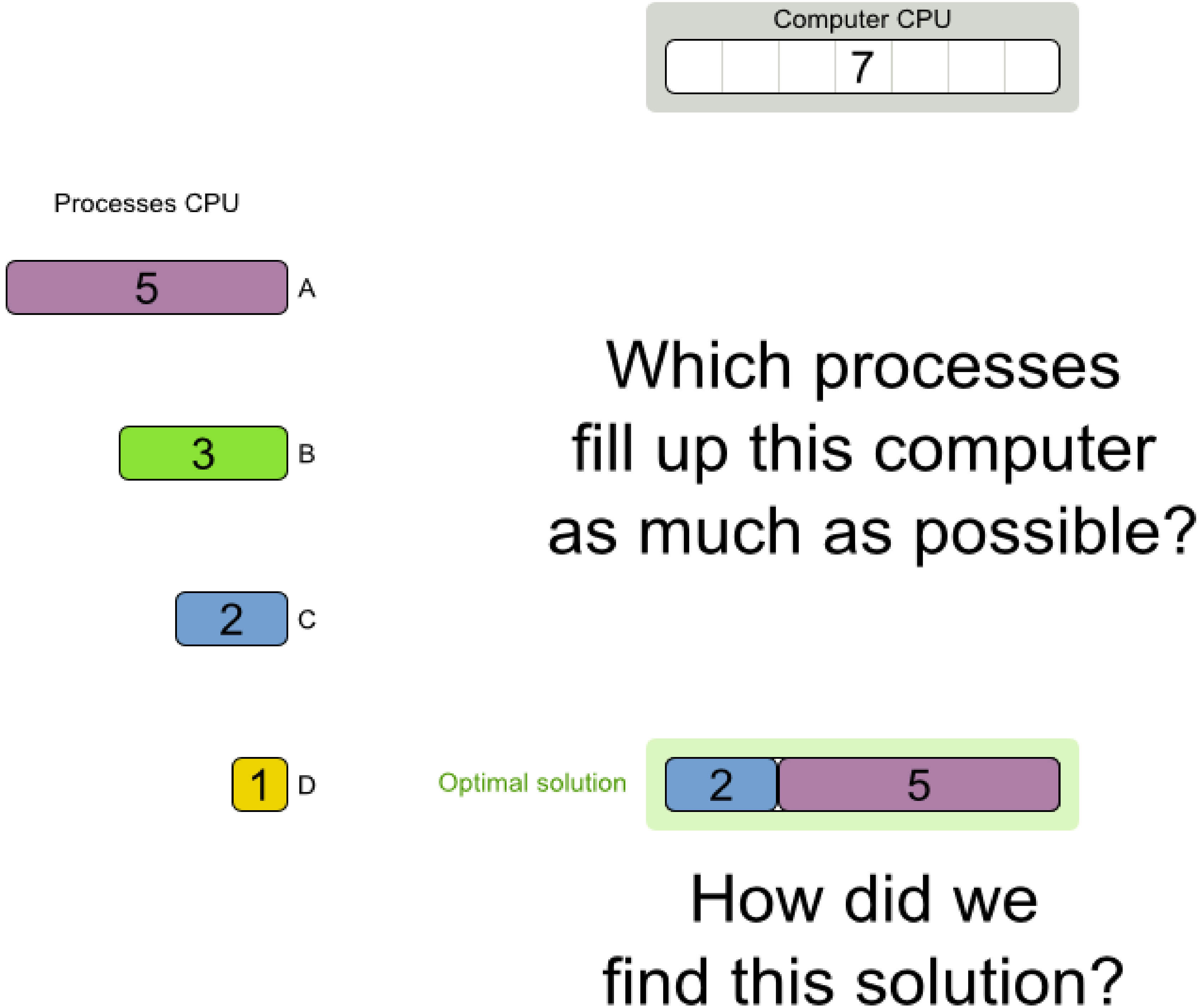
OptaPlanner versus arbitrary feasible assignments

5 mins Tabu Search vs First Feasible Fit

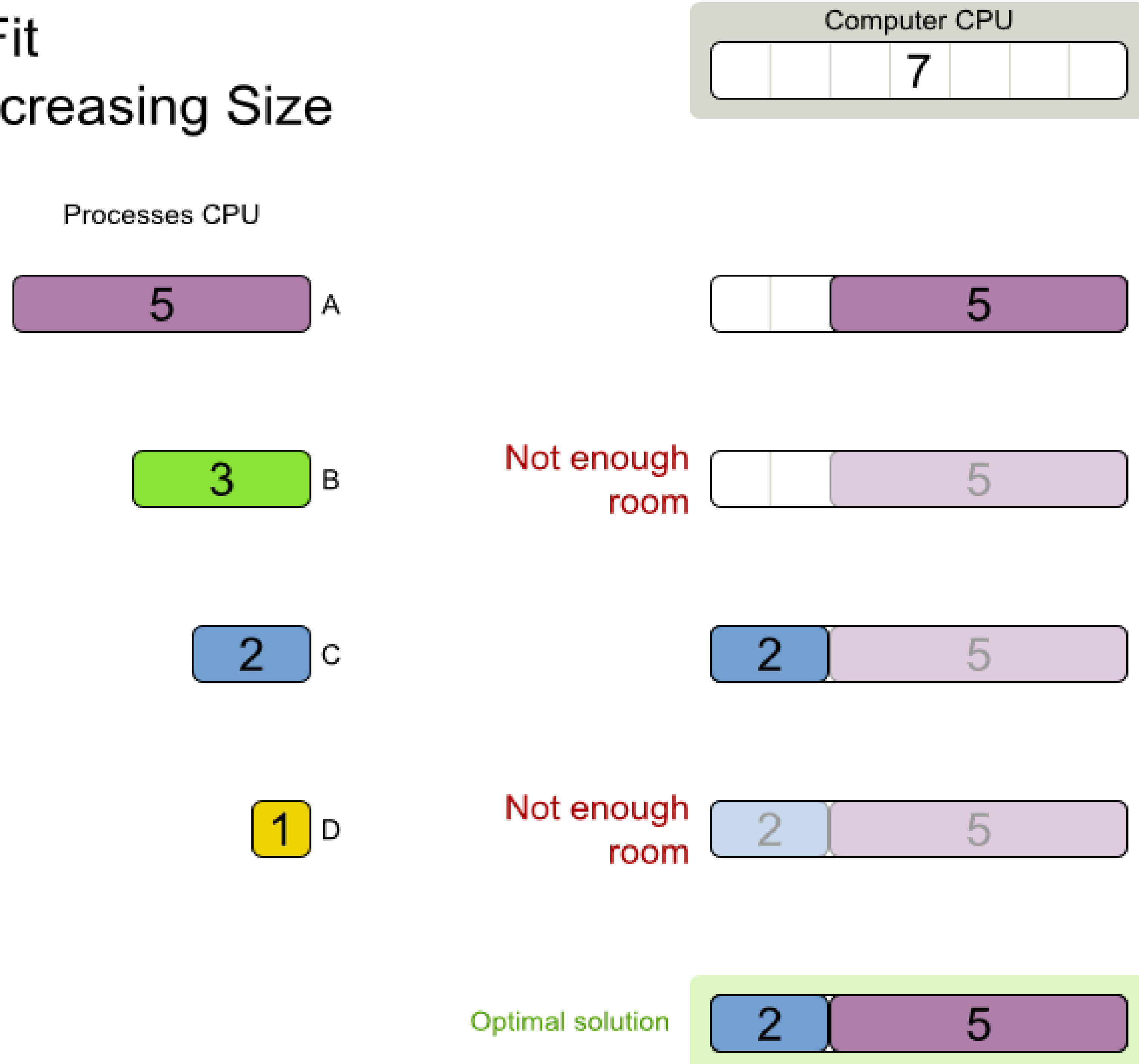
Don't believe us? Run our open benchmarks yourself: http://www.optaplanner.org/code/benchmarks.html

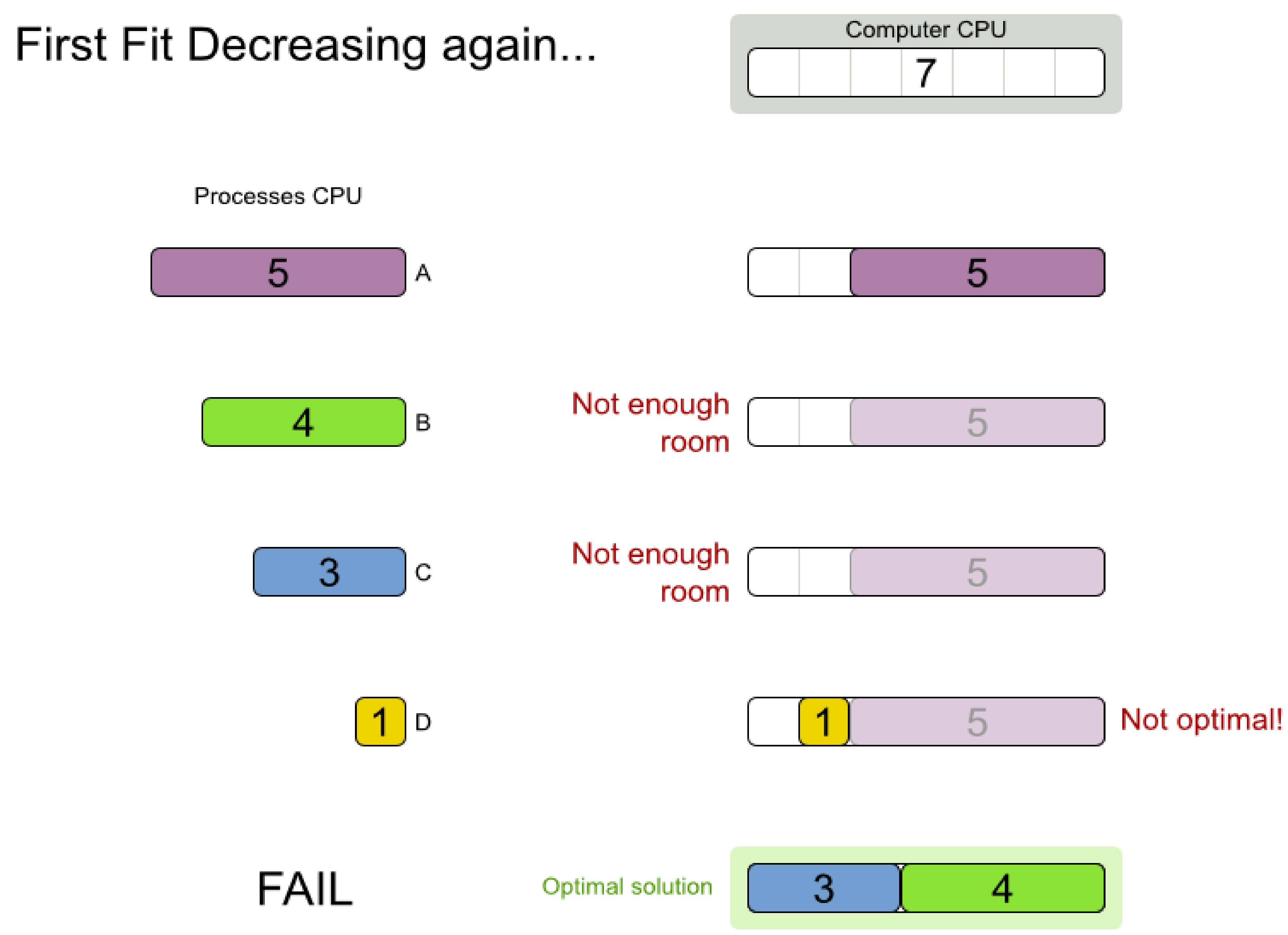
What is business resource optimization?

Are planning problems difficult to solve?

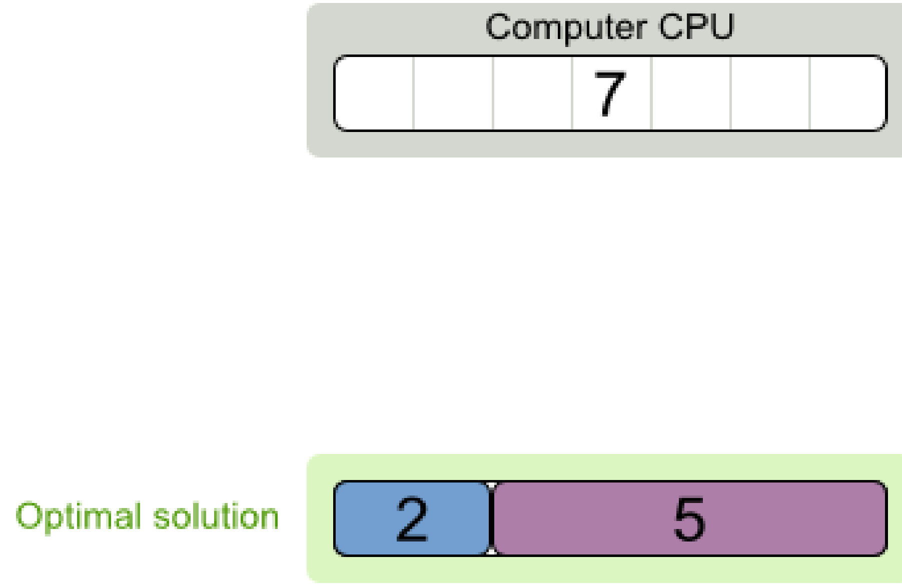

Computer CPU
7
Processes CPU
5
A
3
B
2
C
1
D
Which processes
fill up this computer
as much as possible?
Optimal solution
2
5
How did we
find this solution?

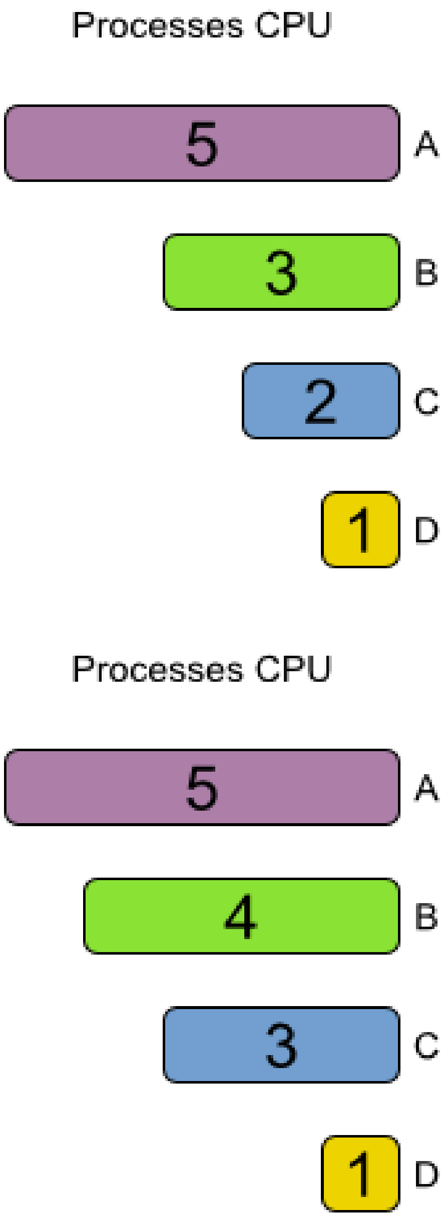
First Fit
by Decreasing Size




First Fit Decreasing again...
Computer CPU
7
Processes CPU
5
A
5
4
B
Not enough room
5
3
C
Not enough room
5
1
D
1
5
Not optimal!
FAIL
Optimal solution
3
4

This is... **NP Complete**



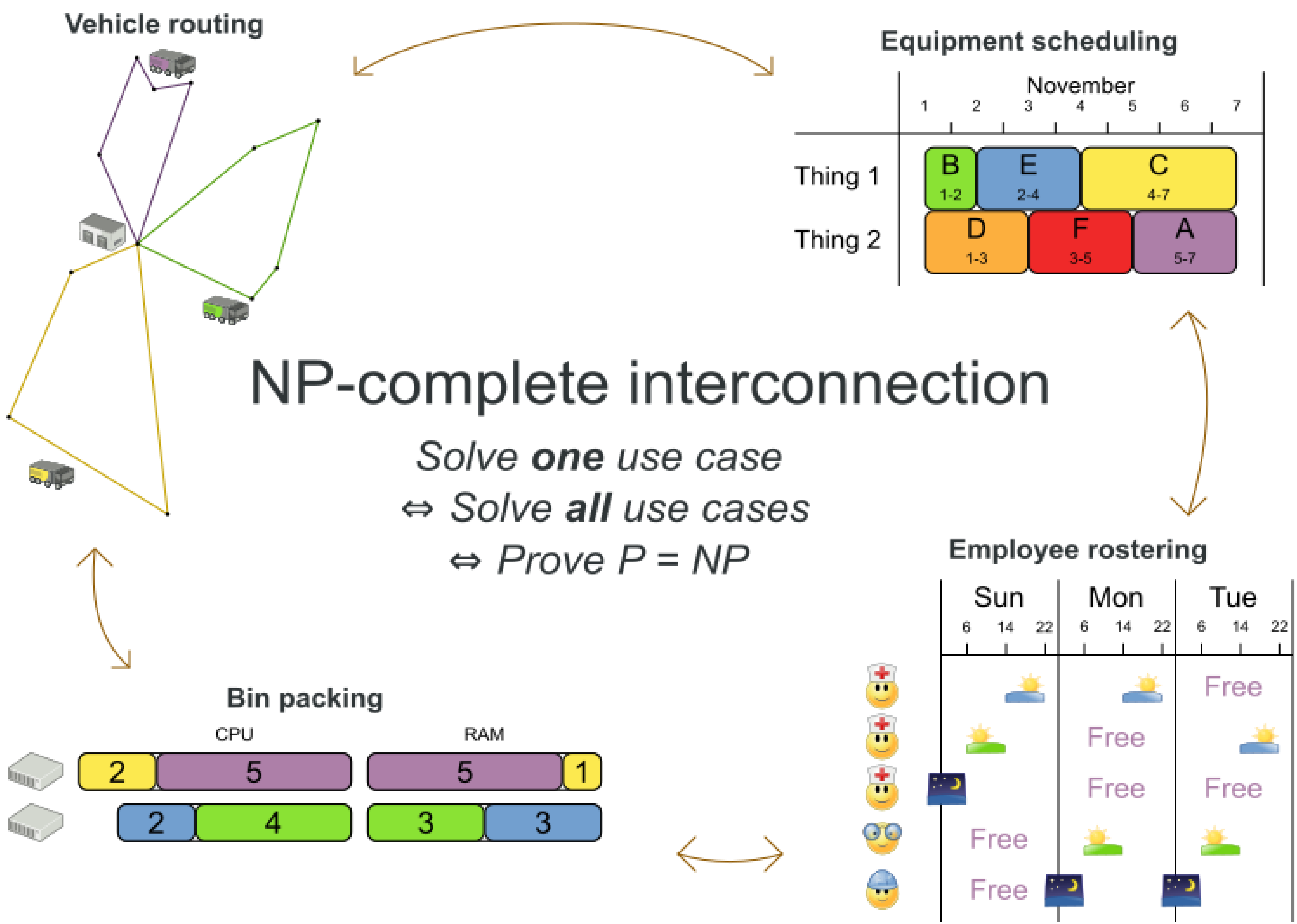


Can any algorithm
find the optimal solution
and scale out?

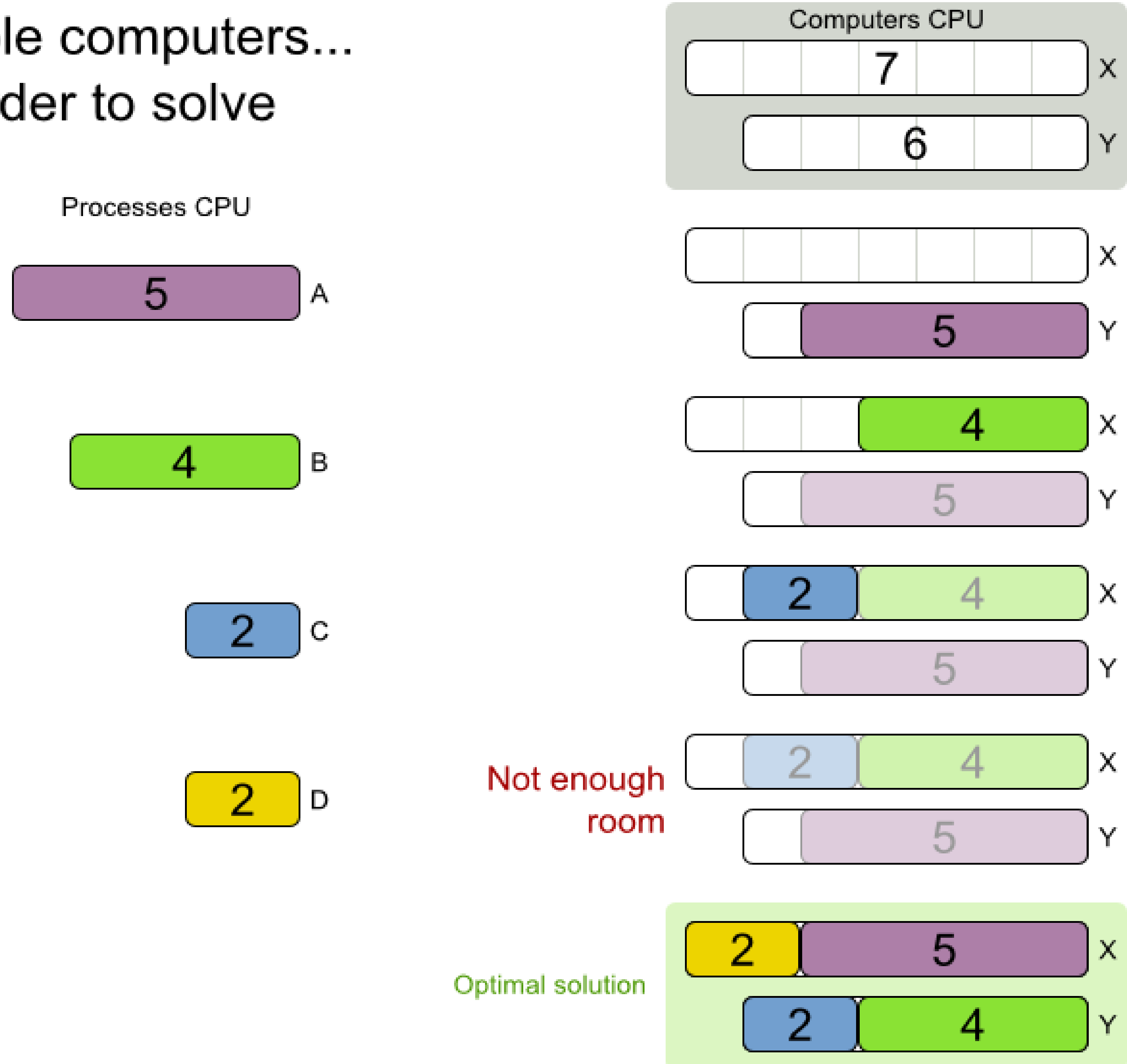
Find optimal solution and scale out for an NP-complete problem?

⇔ Is P = NP?

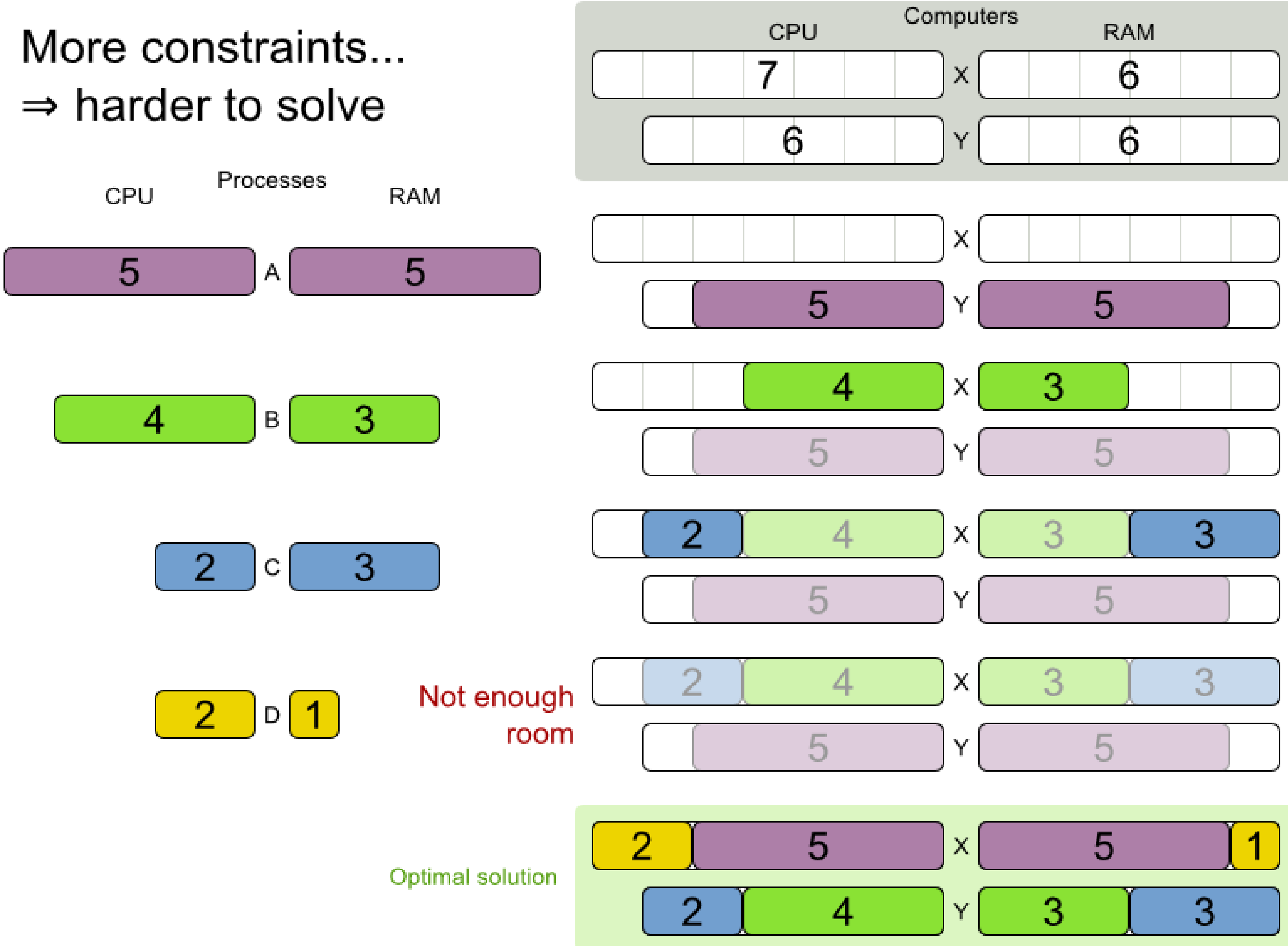
- Unresolved since 1971
- 1 000 000 $ reward since 2000
  - One of the 7 Millennium Problems (http://www.claymath.org/millennium-problems)
- Most believe P ≠ NP
  - ⇔ **Impossible to find optimal solution and scale out**
- 3000+ known NP-complete problems (wikipedia (http://en.wikipedia.org/wiki/List_of_NP-complete_problems))


Vehicle routing
Equipment scheduling
November
1 2 3 4 5 6 7
Thing 1
Thing 2
B 1-2
E 2-4
C 4-7
D 1-3
F 3-5
A 5-7
NP-complete interconnection
Solve **one** use case
⇔ Solve **all** use cases
⇔ Prove P = NP
Employee rostering
Sun
Mon
Tue
6 14 22
Free
Bin packing
CPU
RAM
2 5
2 4
5 1
3 3

Multiple computers...
⇒ harder to solve

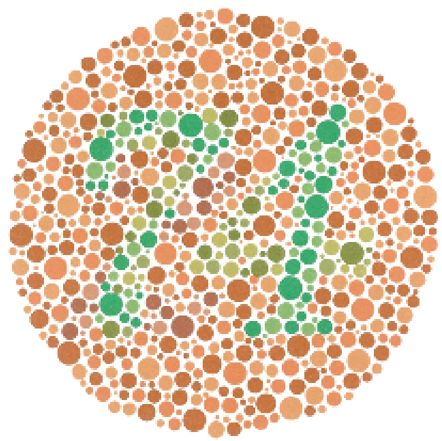


More constraints...
⇒ harder to solve



Planning problems are difficult to solve!

And human aren't good at it

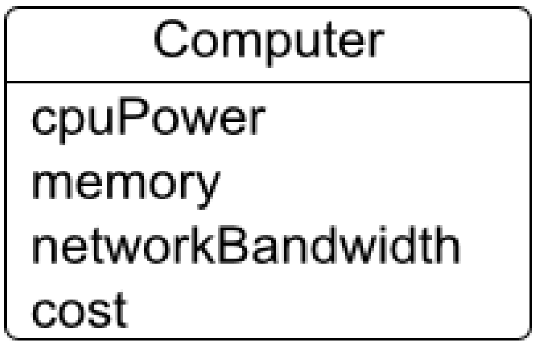


But they don't realize it
(nor does their manager)

Cloud Balancing example

Domain model

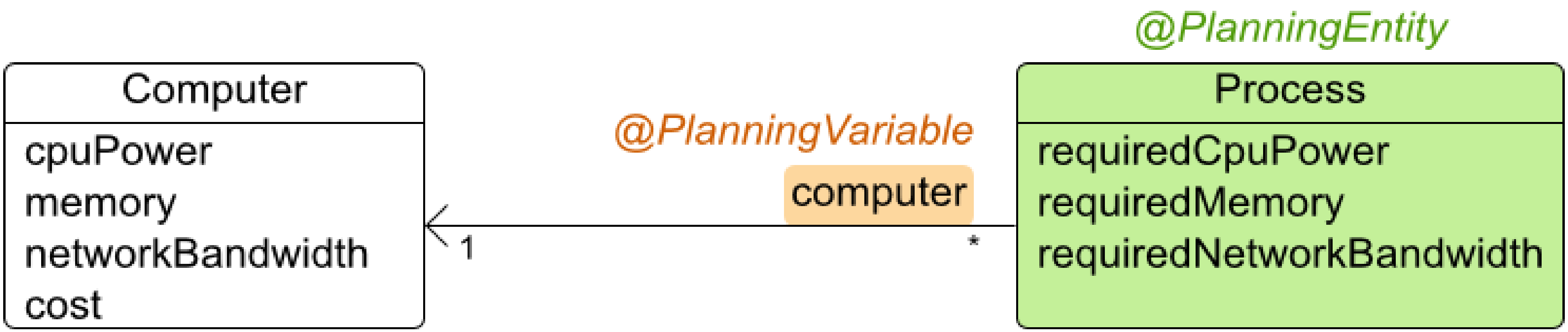
Cloud balance class diagram



Computer

```
public class Computer {

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

  private int cost;

  // getters
}
```

Cloud balance class diagram



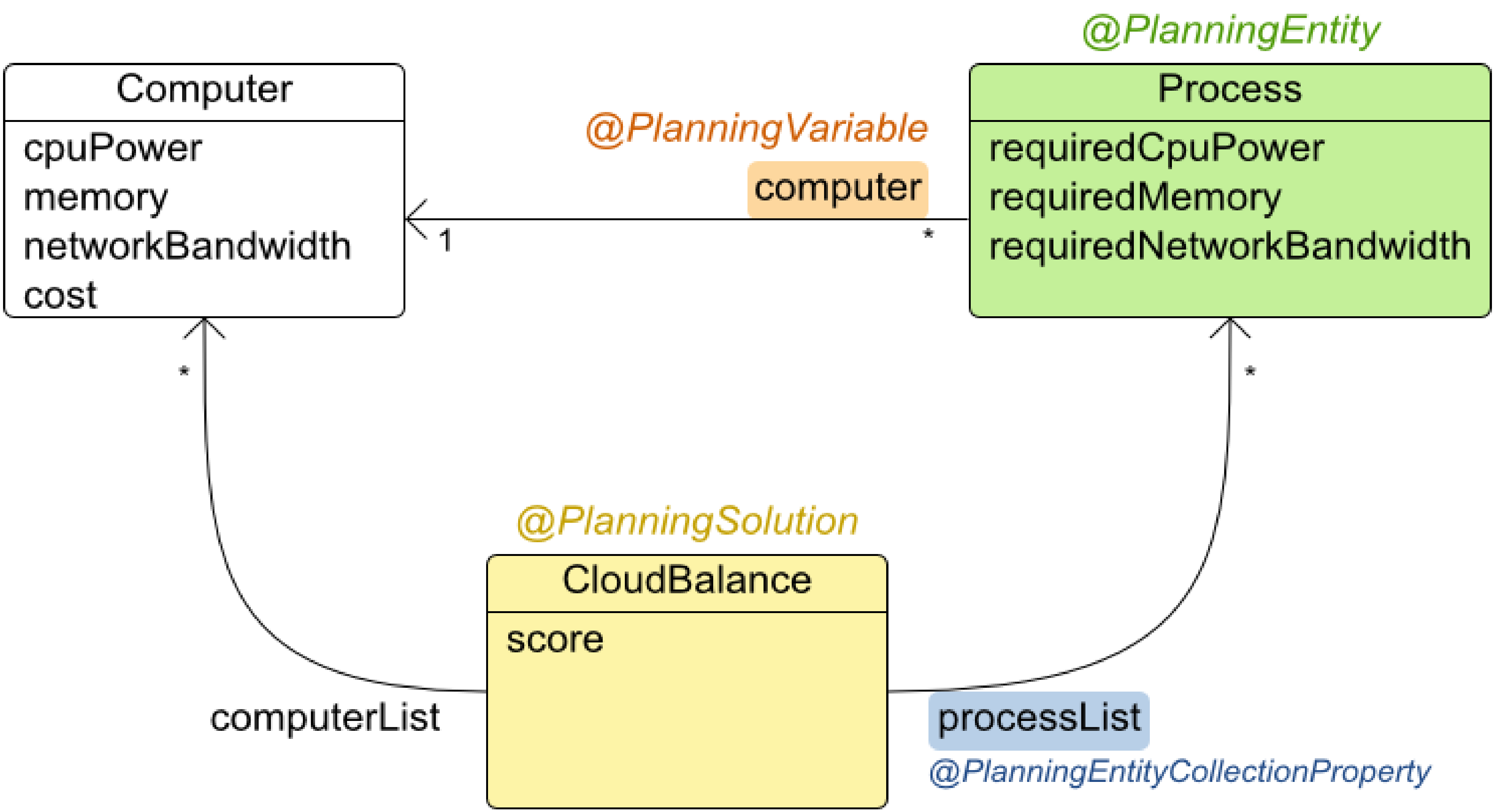
Process is a planning entity

```
@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



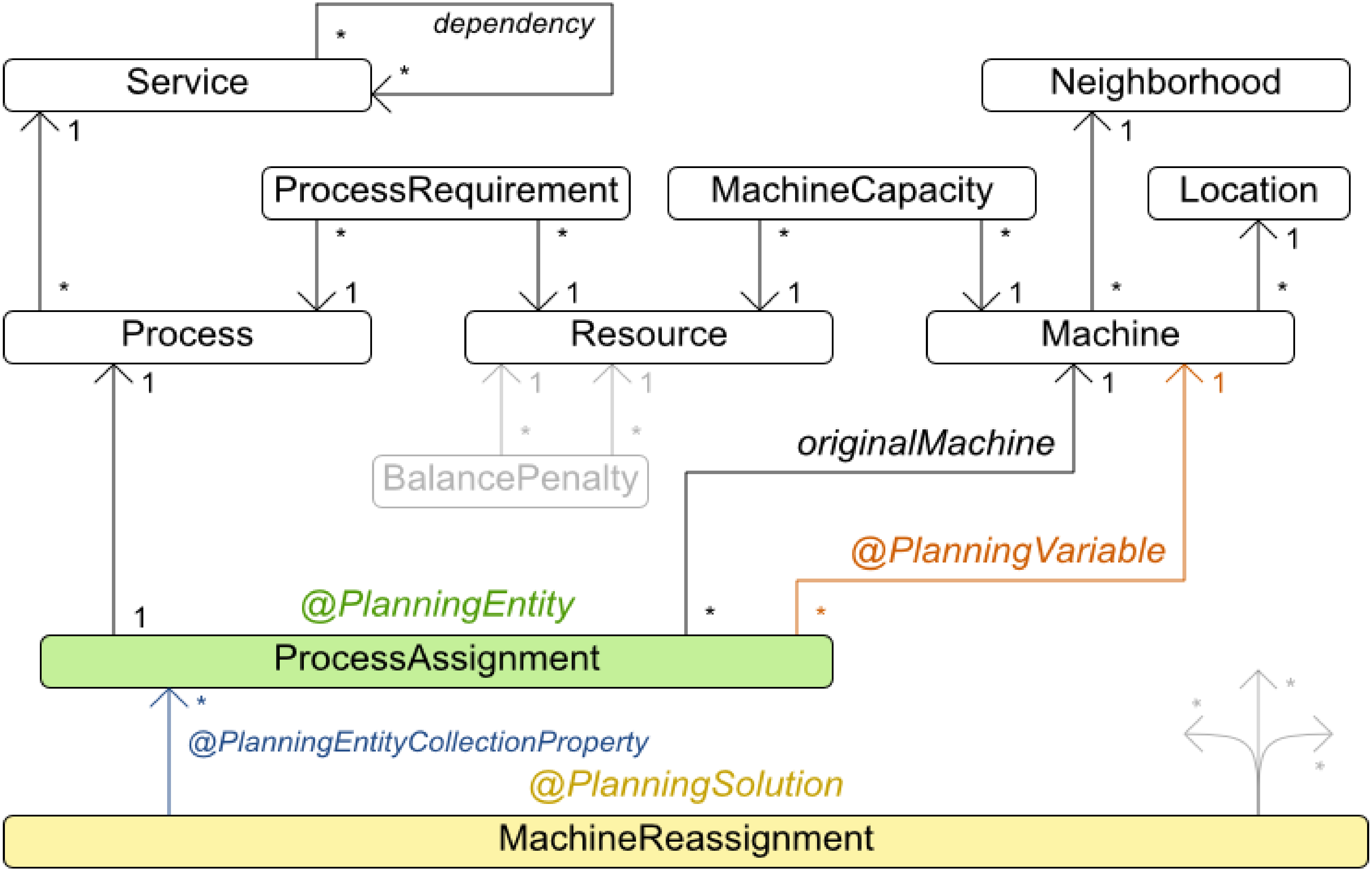
Solution CloudBalance

```
@PlanningSolution
public class CloudBalance {

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

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

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

Solution CloudBalance: score

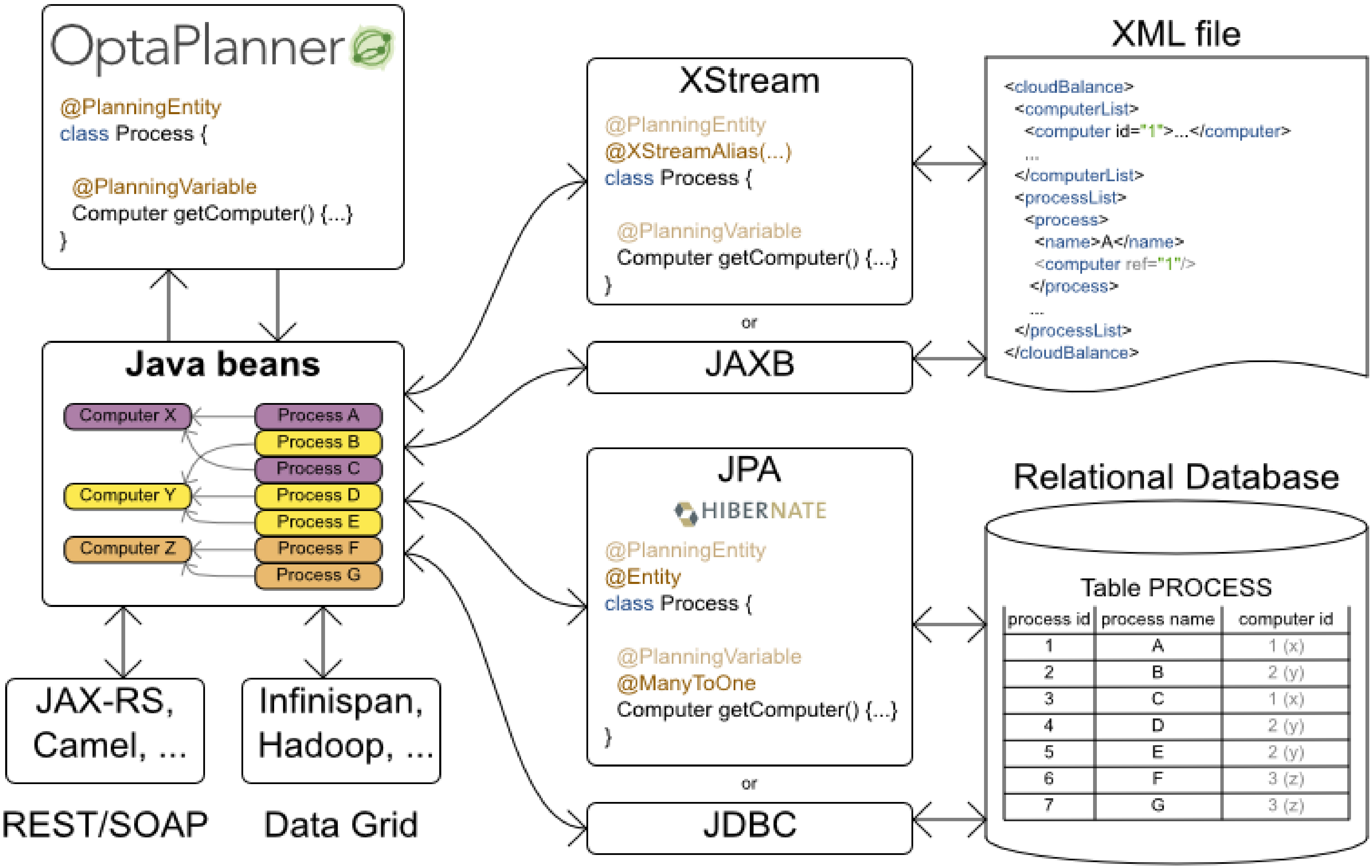
```
@PlanningSolution
public class CloudBalance {
  ...

  private HardSoftScore score;

  @PlanningScore
  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 $
Optimal solution
3
3
Y
1000 $
0hard / -1000soft
Highest score

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 HardSoftScore 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

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

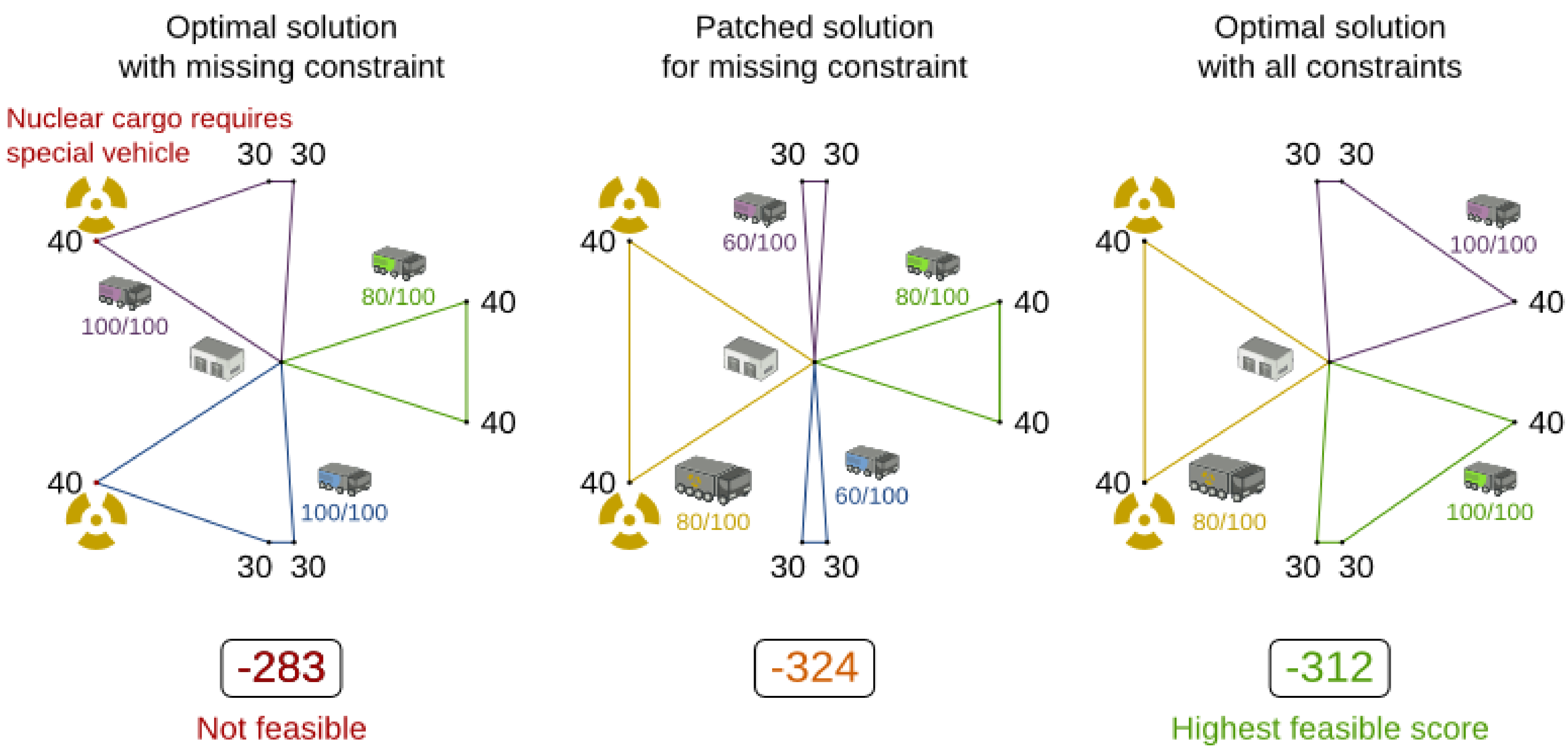
```
rule "requiredCpuPowerTotal"
  when
    // there is a computer
    $s : Computer($cpu : cpuPower)
    // with too little cpu for its processes
    accumulate(
      Process(computer == $s, $requiredCpu : requiredCpuPower);
      $total : sum($requiredCpu);
      $total > $cpu
    )
  then
    //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

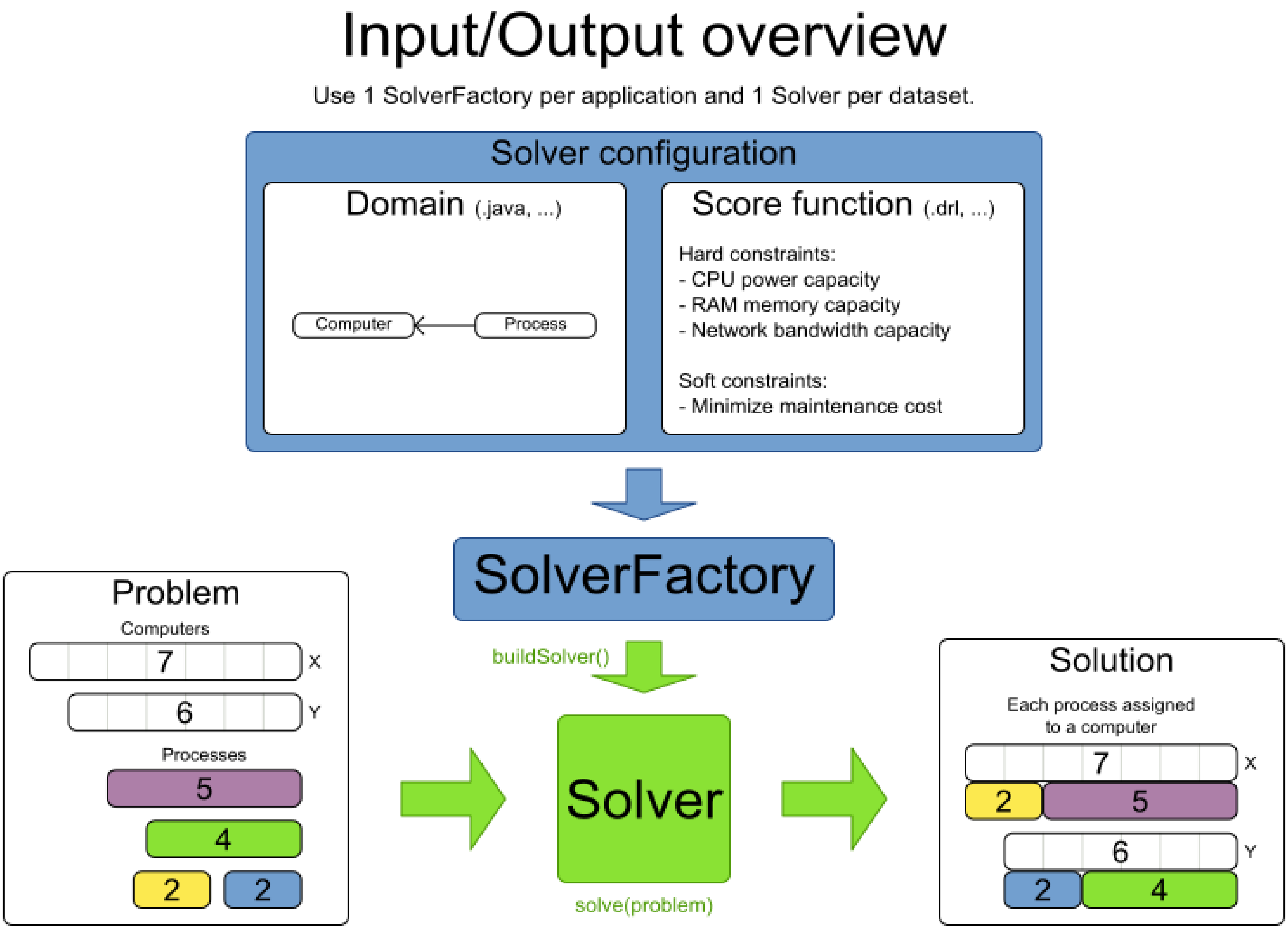
Optimal with incomplete constraints

The optimal solution for a problem that misses a constraint is probably useless.



Cloud Balancing example

Solving it


Input/Output overview
Use 1 SolverFactory per application and 1 Solver per dataset.
Solver configuration
Domain (.java, ...)
Computer
Process
Score function (.drl, ...)
Hard constraints:
- CPU power capacity
- RAM memory capacity
- Network bandwidth capacity
Soft constraints:
- Minimize maintenance cost
SolverFactory
buildSolver()
Problem
Computers
7
X
6
Y
Processes
5
4
2
2
Solver
solve(problem)
Solution
Each process assigned to a computer
7
X
2
5
6
Y
2
4

Solver configuration by XML

```xml
<solver>
  <scanAnnotatedClasses/>

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

</solver>
```

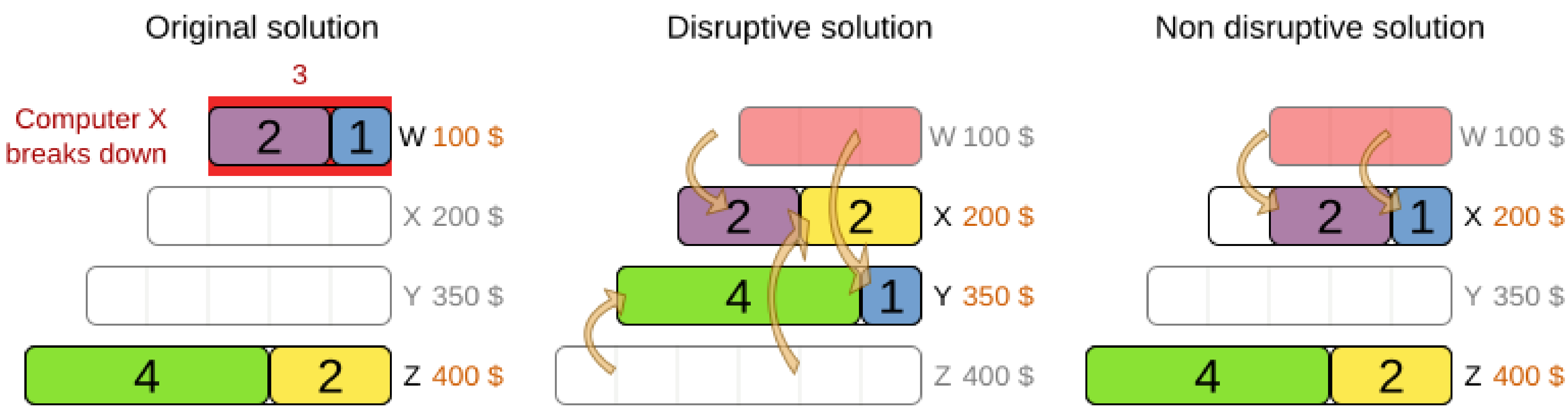
Solving

```
SolverFactory<CloudBalance> factory
        = SolverFactory.createFromXmlResource("...SolverConfig.xml");

Solver<CloudBalance> solver = factory.buildSolver();
CloudBalance problem = ... // Load problem
CloudBalance solution = solver.solve(problem);
```

Repeated planning

Non disruptive replanning

Real-time planning must not distort the entire plan to deal with a real-time change.



Normal score

Original solution	Disruptive solution	Non disruptive solution
-3hard / -500soft	0hard / -550soft Highest score	0hard / -600soft

Adjusted score (-100 per moved process)

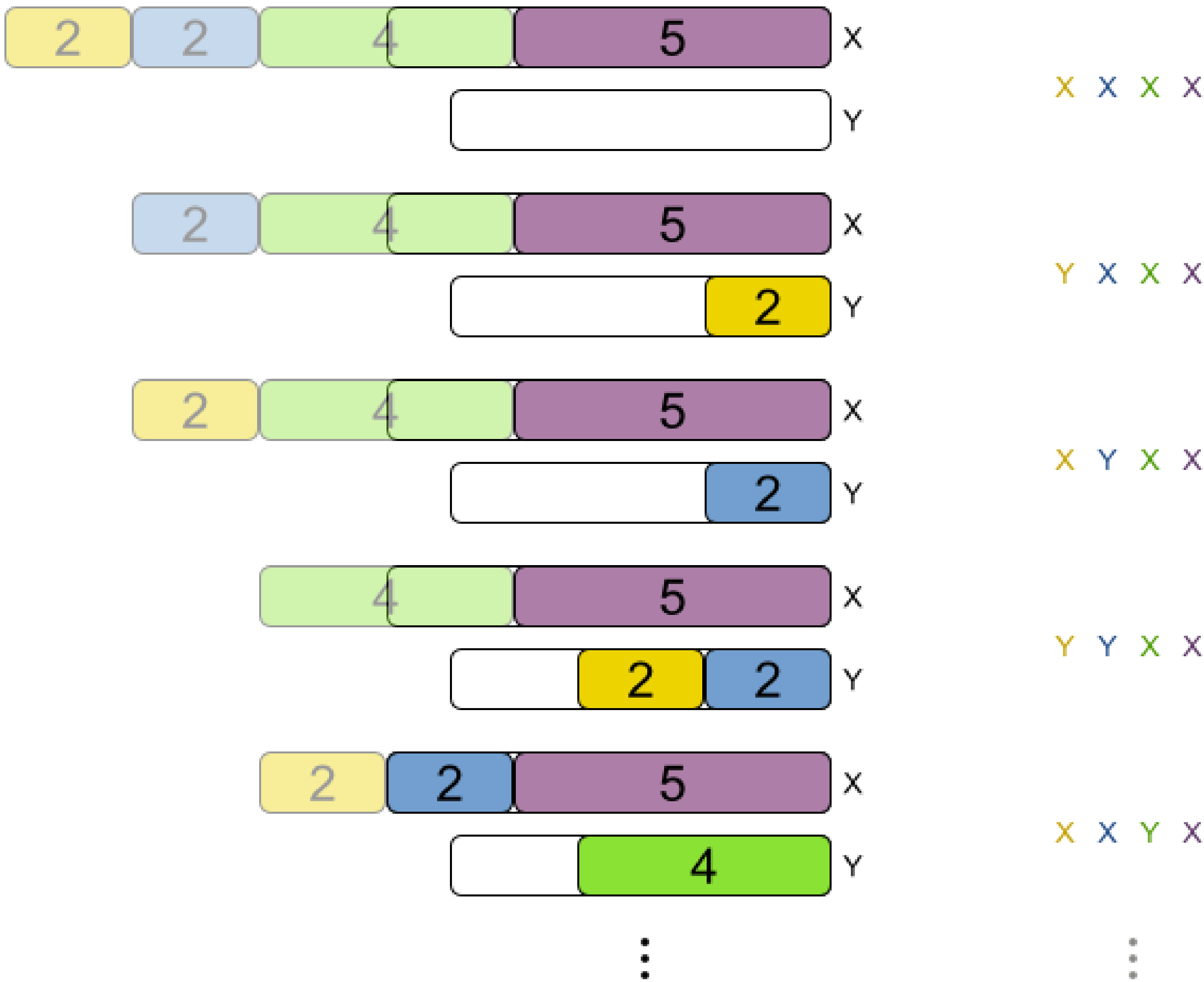
Original solution	Disruptive solution	Non disruptive solution
no moved processes: 0soft	4 moved processes: -400soft	2 moved processes: -200soft
-3hard / -500soft	0hard / -950soft	0hard / -800soft Highest score

Power tweaking and benchmarking optimization algorithms

Cloud Balancing example

Optimization algorithms

Brute Force



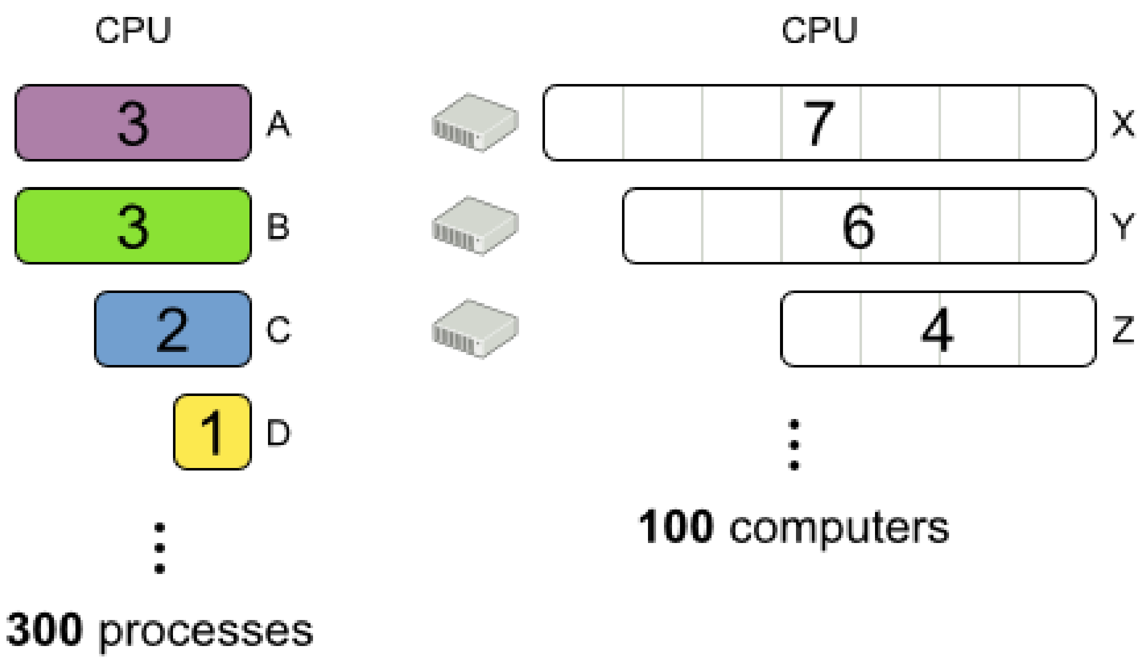
2 2 4 5 X
Y
X X X X
2 4 5 X
2 Y
Y X X X
2 4 5 X
2 Y
X Y X X
4 5 X
2 2 Y
Y Y X X
2 2 5 X
4 Y
X X Y X

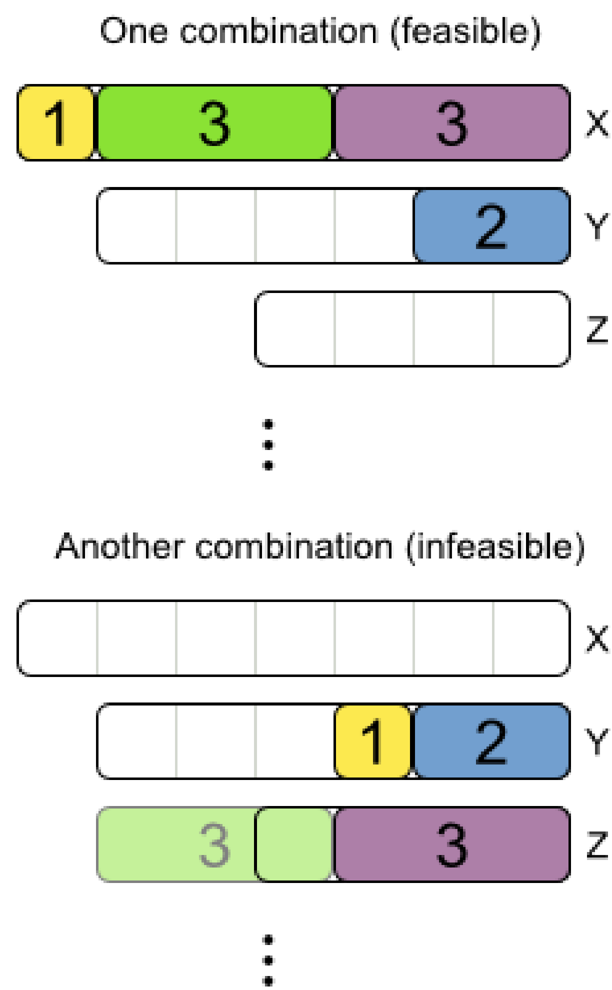
Brute Force config

```
<solver>
  ...

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

What is the size of the search space?

How big is the haystack?





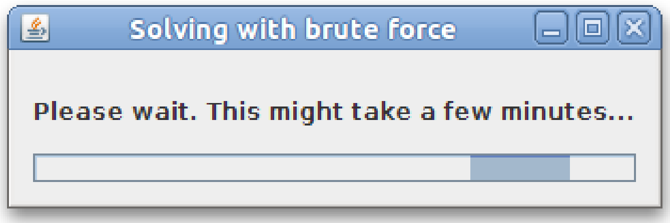
In how many combinations can 300 processes
be assigned to 100 computers?

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

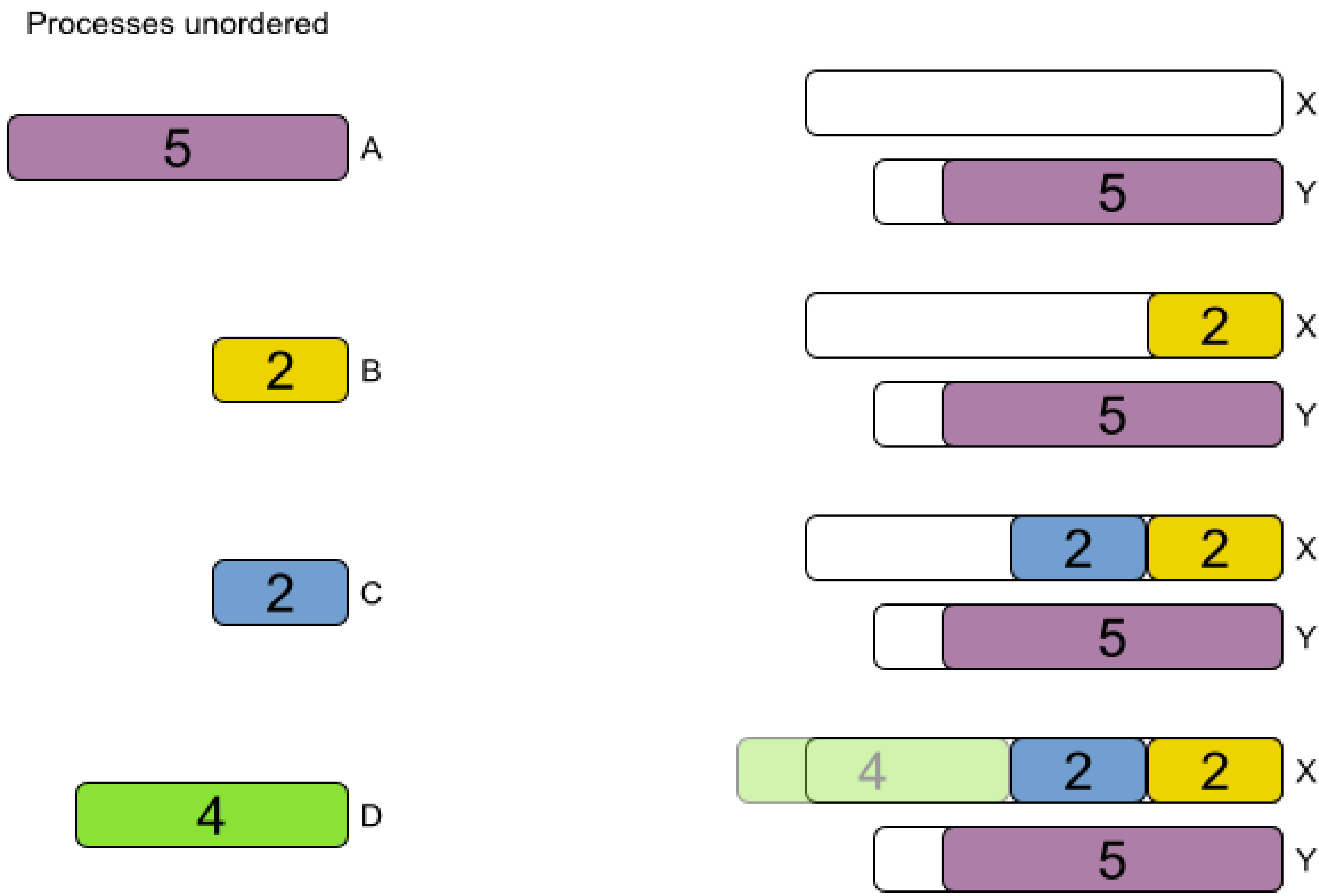
$$= 10^{600} = 1000000000000000000000\ldots$$

Brute Force scalability

Plan 1200 processes with Brute Force?



First Fit



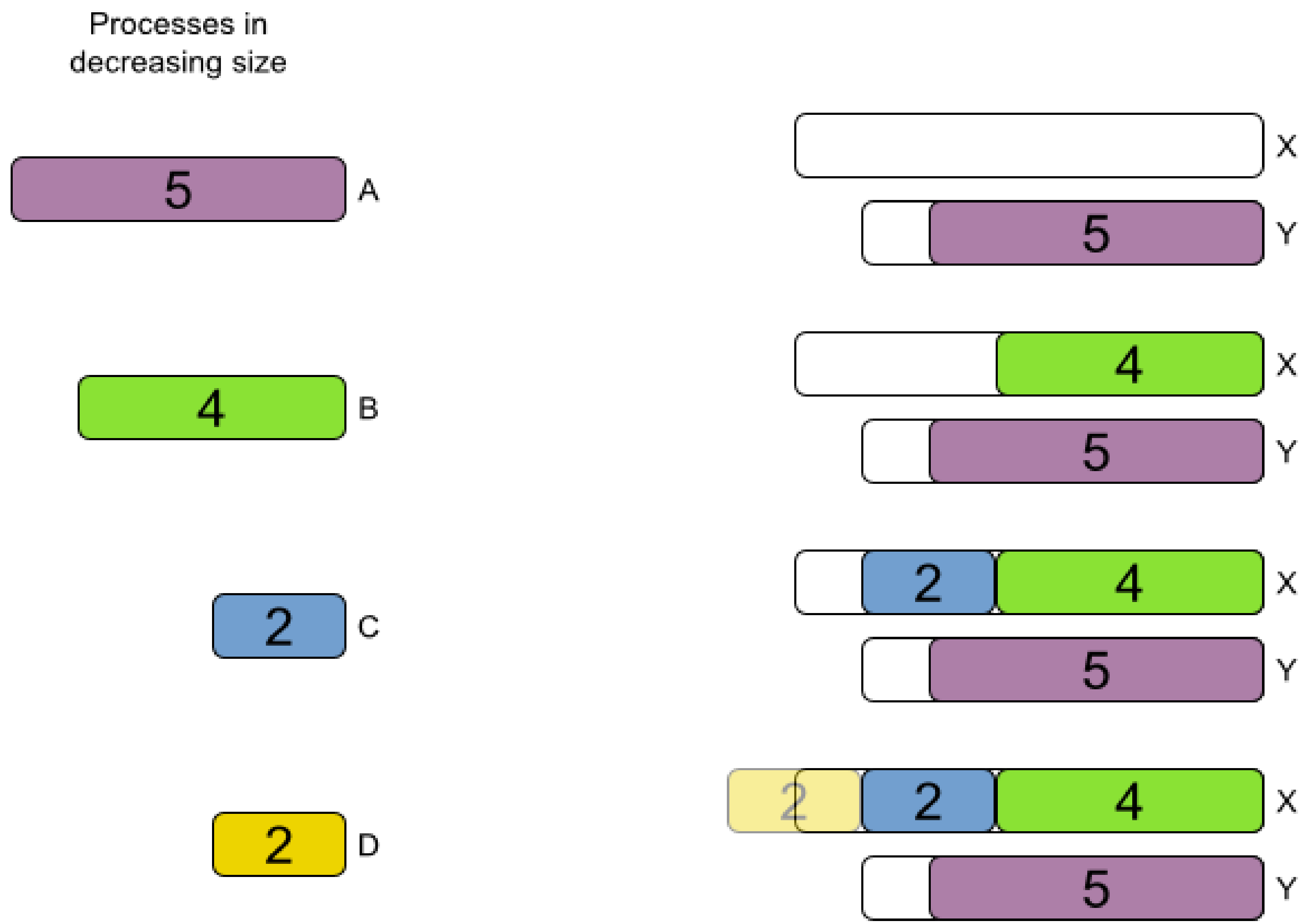
First Fit config

```xml
<solver>
  ...

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

First Fit scalability

First Fit results

First Fit Decreasing



First Fit Decreasing config

```
<solver>
  ...

  <constructionHeuristic>
    <constructionHeuristicType>FIRST_FIT_DECREASING</constructionHeuristicTy
pe>
  </constructionHeuristic>
</solver>
```

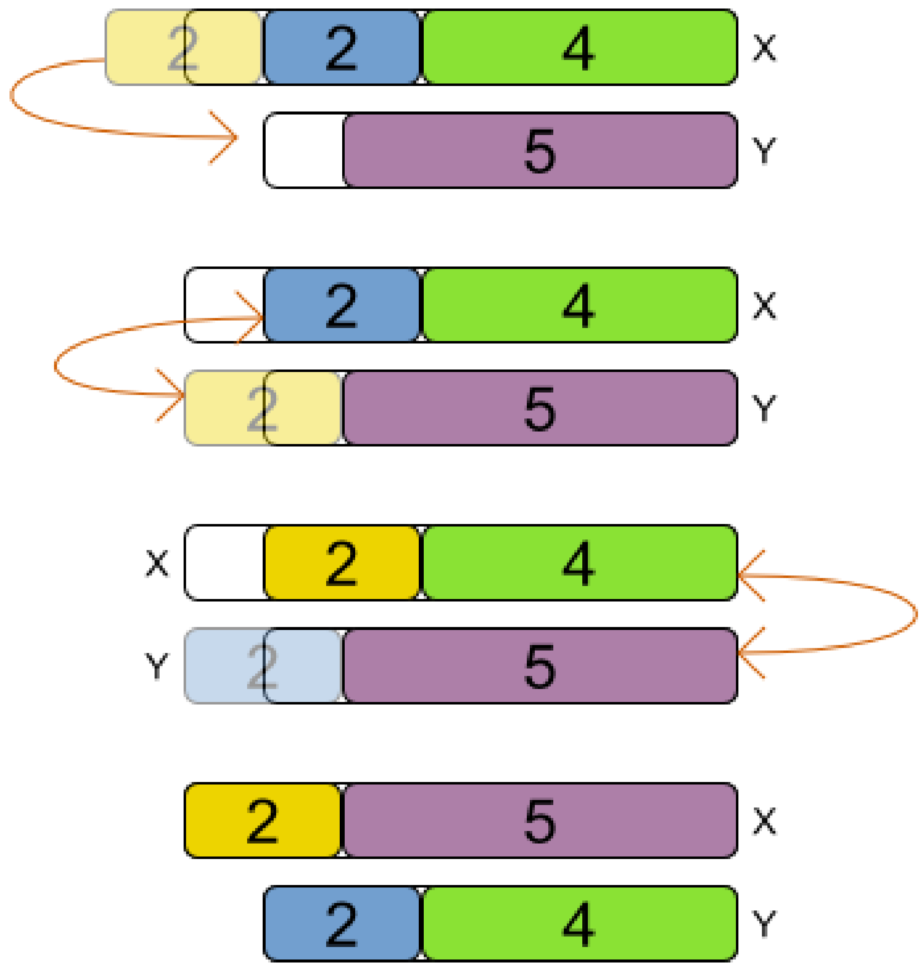
DifficultyComparator

```
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

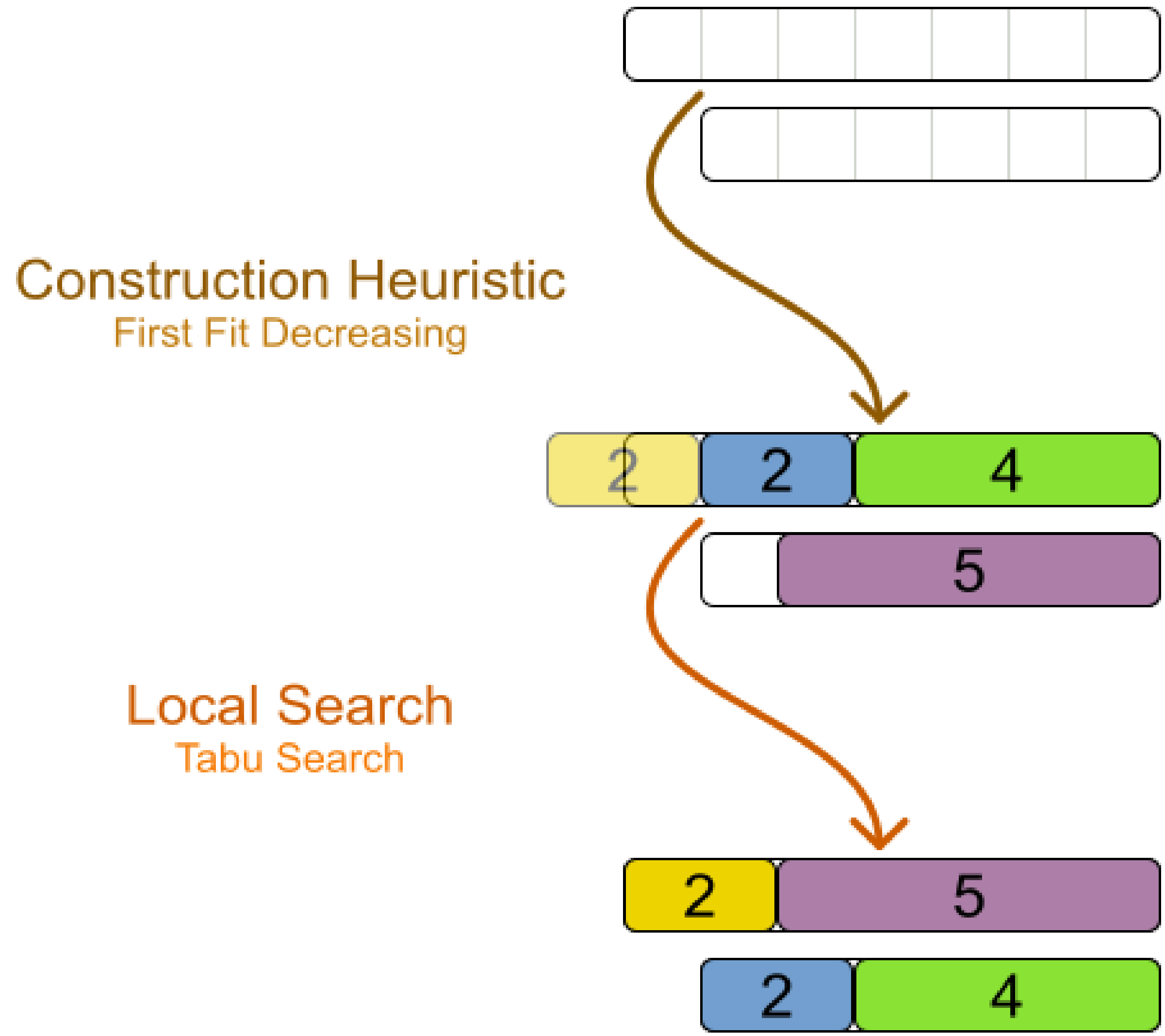
First Fit Decreasing results

Local Search



General phase sequence

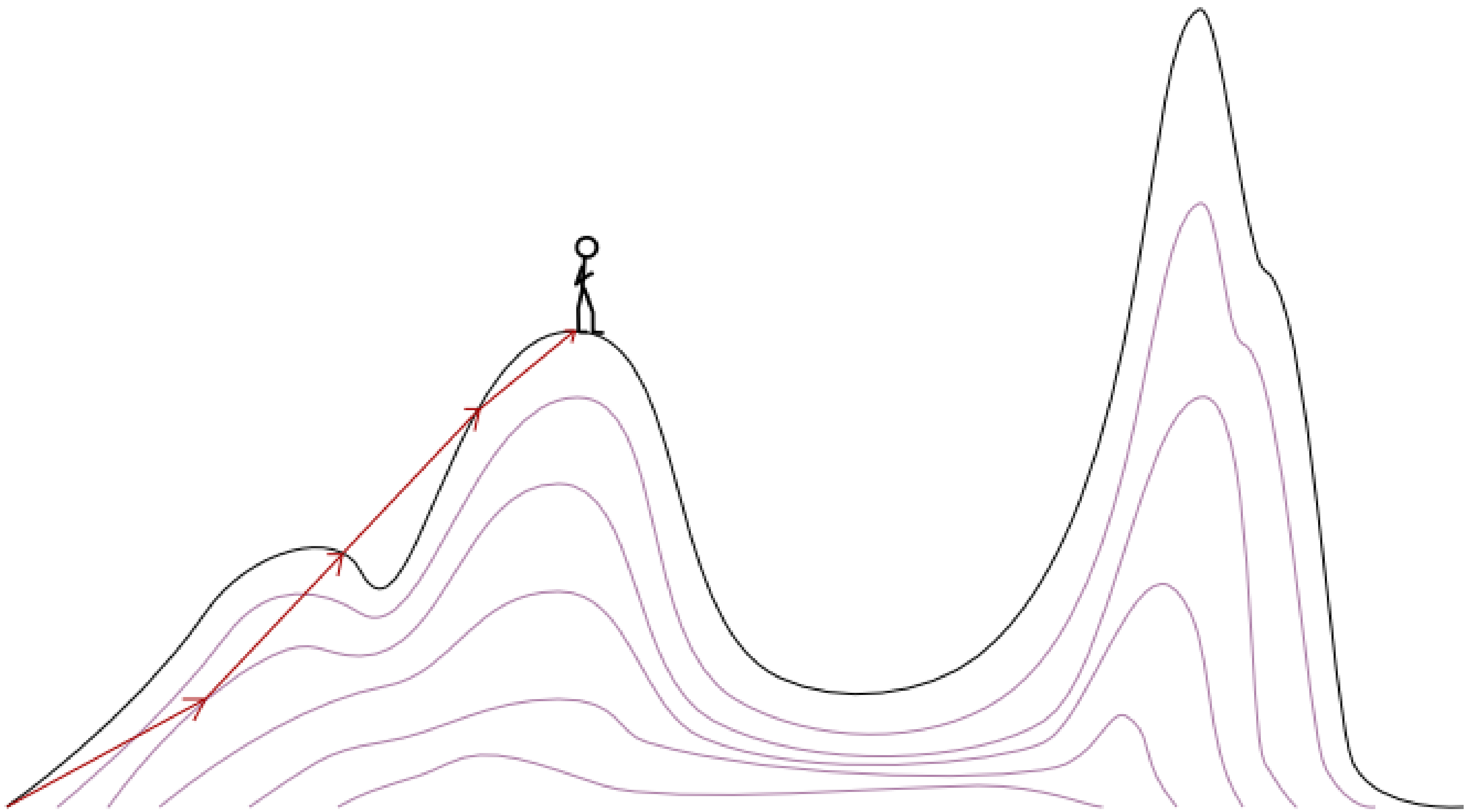
First a Construction Heuristic,
then a Local Search



Construction Heuristics + Local Search

```
<solver>
  ...

  <constructionHeuristic>
    <constructionHeuristicType>FIRST_FIT_DECREASING</constructionHeuristicTy
pe>
  </constructionHeuristic>
  <localSearch>
    ...
  <localSearch>
</solver>
```

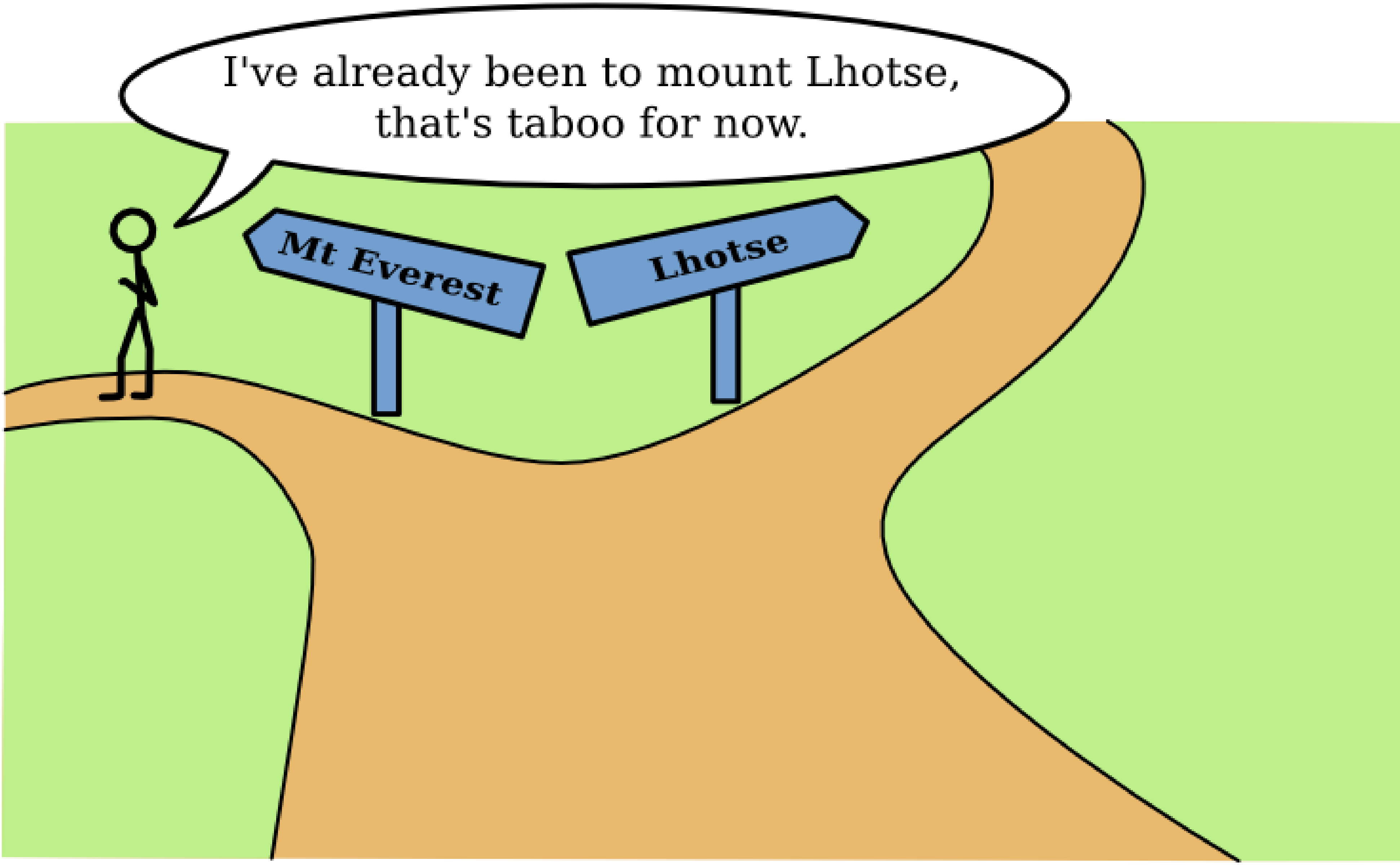
Hill climbing



Hill Climbing config

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

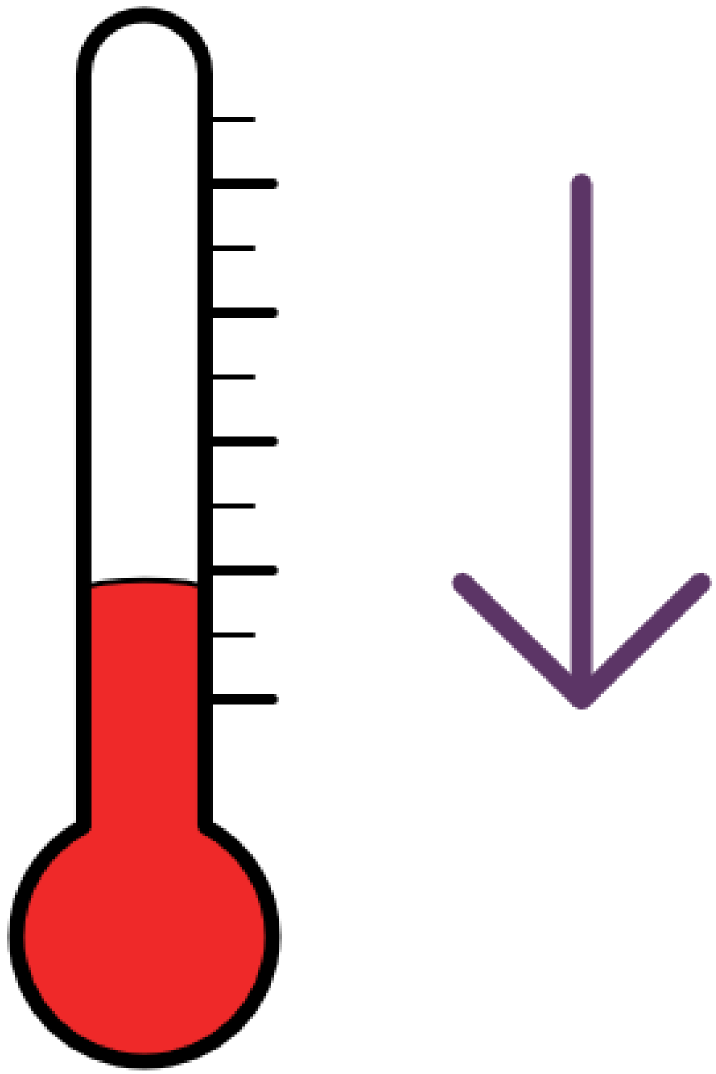
Tabu Search


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

Tabu Search config

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

Simulated Annealing



Simulated Annealing config

```xml
<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

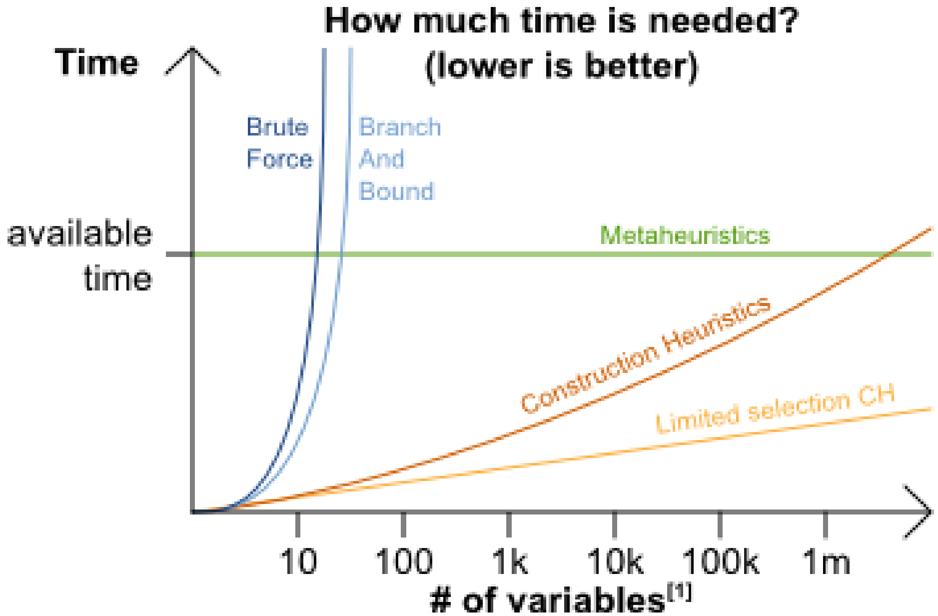
Cost ($) reduction

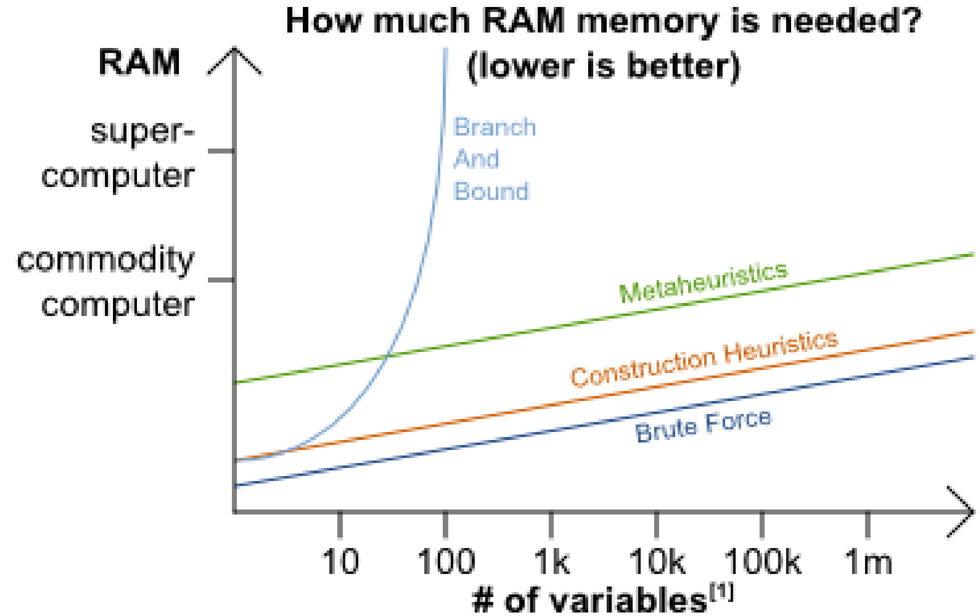
Optimization algorithms

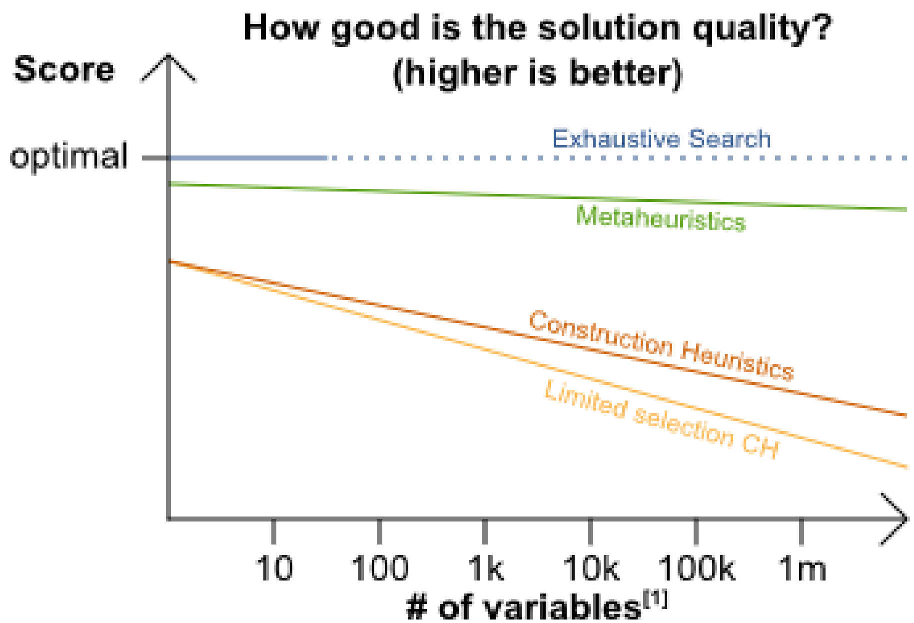
- Exhaustive Search
  - Brute Force
  - Branch And Bound
- Construction Heuristics
  - First Fit (Decreasing)
  - Weakest/Strongest Fit (Decreasing)
  - Cheapest Insertion
- Metaheuristics (Local Search, ...)
  - Hill Climbing
  - Tabu Search
  - Strategic Oscillation Tabu Search
  - Simulated Annealing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

Scalability of optimization algorithms

When scaling out, metaheuristics deliver the best solution in reasonable time on realistic hardware.







Effects of scaling out:

Exhaustive Search delivers the optimal solution but takes forever.

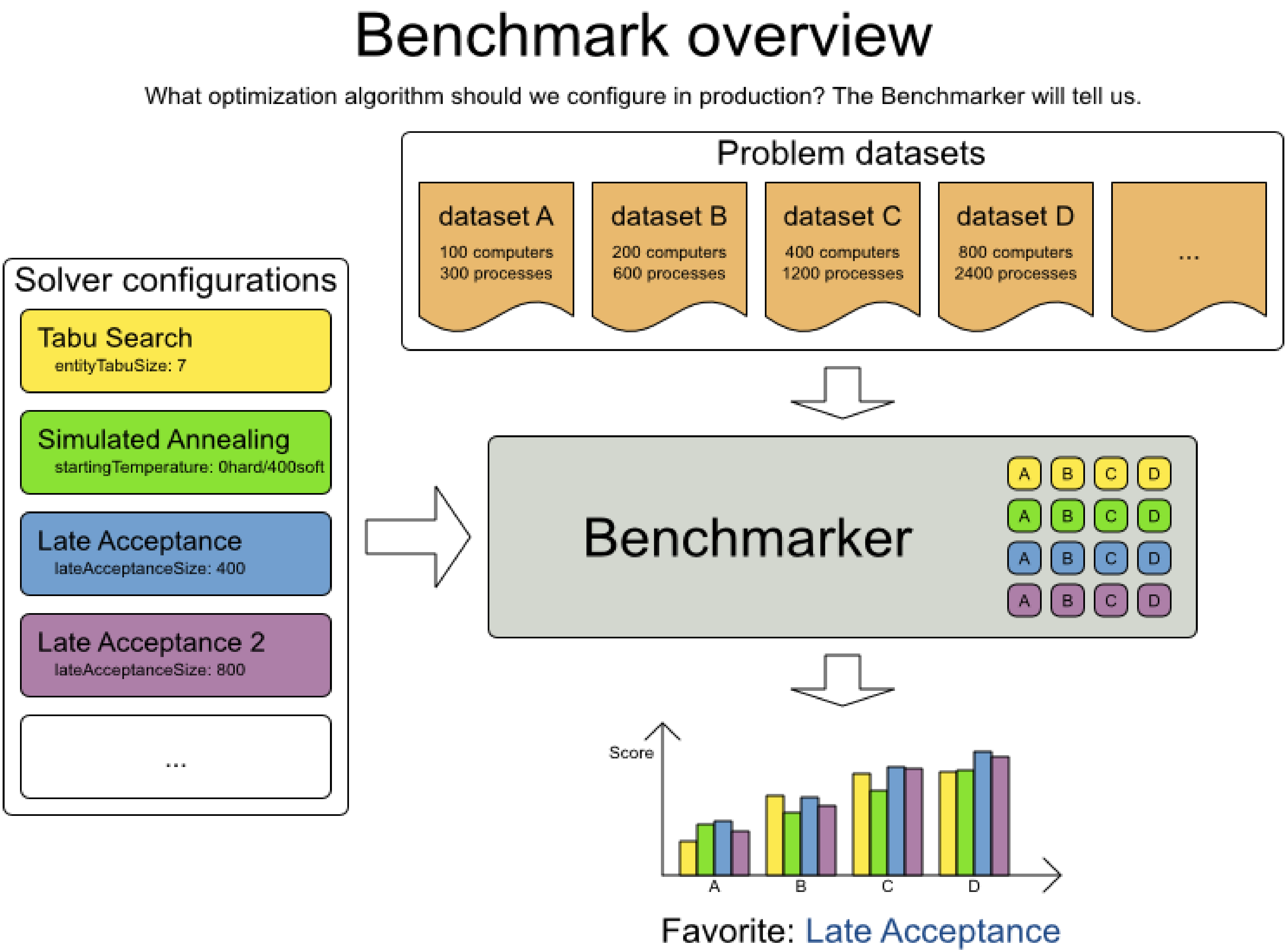
Construction Heuristics (including greedy algorithms) deliver poor quality in time.

Metaheuristics deliver good quality in time.

Note: Metaheuristics include a CH to initialize.

This is a rough generalization, based on years of experience and a large number of benchmarks on realistic use cases. Results may differ per use case and per solver configuration.

[1] Vars with a large value range (binary vars scale much more)

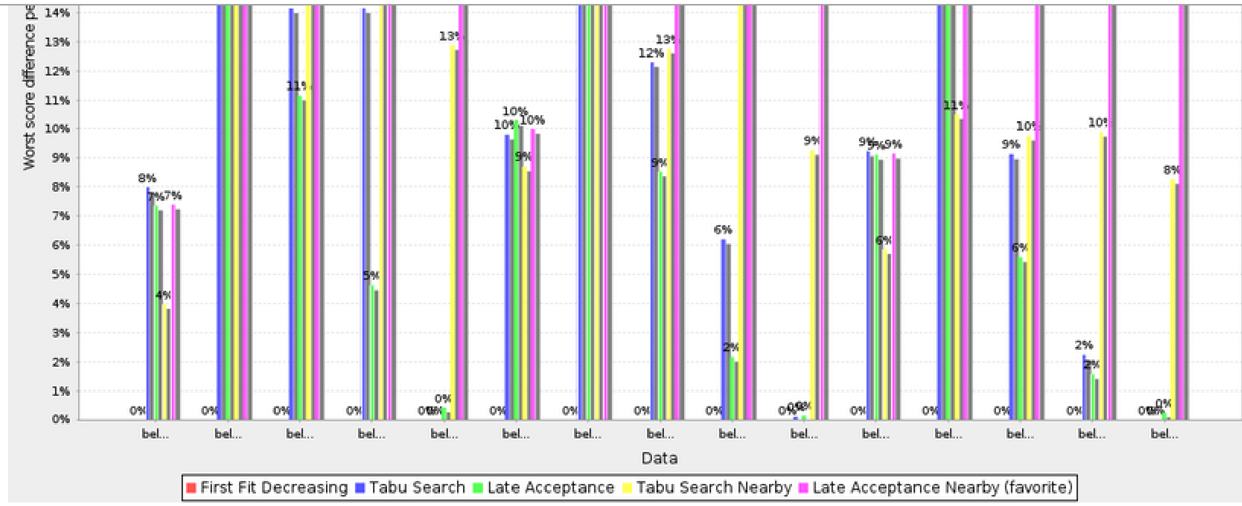

Benchmark overview
What optimization algorithm should we configure in production? The Benchmarker will tell us.
Problem datasets
dataset A
100 computers
300 processes
dataset B
200 computers
600 processes
dataset C
400 computers
1200 processes
dataset D
800 computers
2400 processes
...
Solver configurations
Tabu Search
entityTabuSize: 7
Simulated Annealing
startingTemperature: 0hard/400soft
Late Acceptance
lateAcceptanceSize: 400
Late Acceptance 2
lateAcceptanceSize: 800
...
Benchmarker
A B C D
Score
A
B
C
D
Favorite: Late Acceptance

Benchmarker toolkit

OptaPlanner

- Summary
  - Result
  - Performance

- Problem benchmarks
  - belgium-n50-k10
  - belgium-n100-k10
  - belgium-n500-k20
  - belgium-n1000-k20
  - belgium-n2750-k55
  - belgium-road-km-n50-k10
  - belgium-road-km-n100-k10
  - belgium-road-km-n500-k20
  - belgium-road-km-n1000-k20
  - belgium-road-km-n2750-k55



Solver	Average	Problem						
		belgium-n50-k10	belgium-n100-k10	belgium-n500-k20	belgium-n1000-k20	belgium-n2750-k55	belgium-road-km-n50-k10	belgium-road-km-n100-k10
First Fit Decreasing 4	0.00%/0.00%	0.00%/0.00% 4	0.00%/0.00% 4	0.00%/0.00% 4	0.00%/0.00% 4	0.00%/0.00% 3	0.00%/0.00% 4	0.00%/0.00 4
Tabu Search 2	0.00%/9.79%	0.00%/7.99% 0	0.00%/24.40% 2	0.00%/14.14% 2	0.00%/14.15% 2	0.00%/0.00% 3	0.00%/9.80% 2	0.00%/18.9 2
Late Acceptance 3	0.00%/8.42%	0.00%/7.36% 2	0.00%/25.68% 0	0.00%/11.15% 3	0.00%/4.62% 3	0.00%/0.42% 2	0.00%/10.28% 0	0.00%/21.2 1
Tabu Search Nearby 1	0.00%/11.74%	0.00%/3.98% 3	0.00%/16.74% 3	0.00%/15.49% 1	0.00%/20.96% 1	0.00%/12.88% 1	0.00%/8.71% 3	0.00%/15.2 3

Benchmark report demo

Gain by using OptaPlanner

Use case	Gain type	Avg	Min	Max
Cloud Balancing	Maintenance cost [1]	-18%	-16%	-21%
Machine Reassignment	Unbalanced load [2]	-63%	-25%	-97%
Vehicle Routing (Belgium datasets)	Distance [1]	-20%	-7%	-27%
Nurse rostering	Happiness [1]	+53%	-19%	-85%
Course scheduling	Unhappiness [1]	-66%	-26%	-100%

OptaPlanner in a 5 minute run

[1] Compared to traditional algorithms with domain knowledge.

[2] Compared to initial assignments.

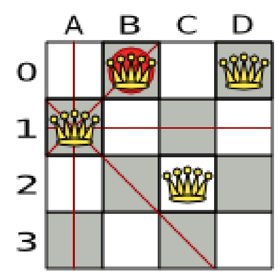
Optimization algorithms 101
Take the red pill

N Queens demo

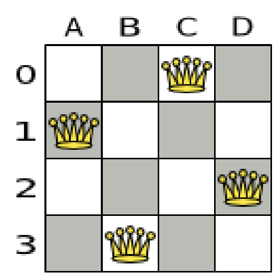
Simplified problem

N Queens

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



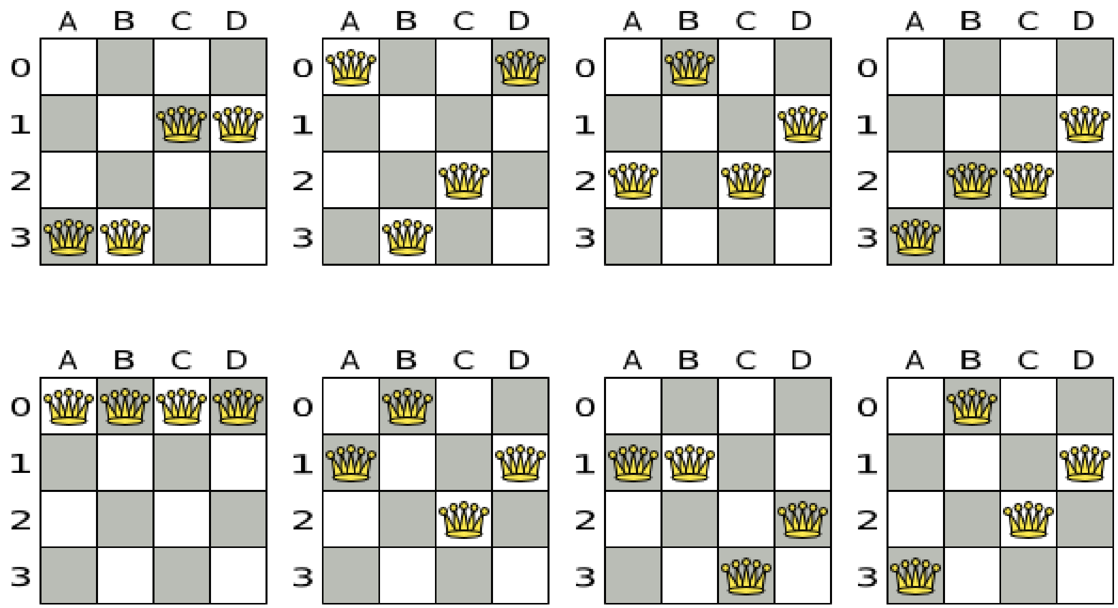
Bad



Good

- Imperfect example

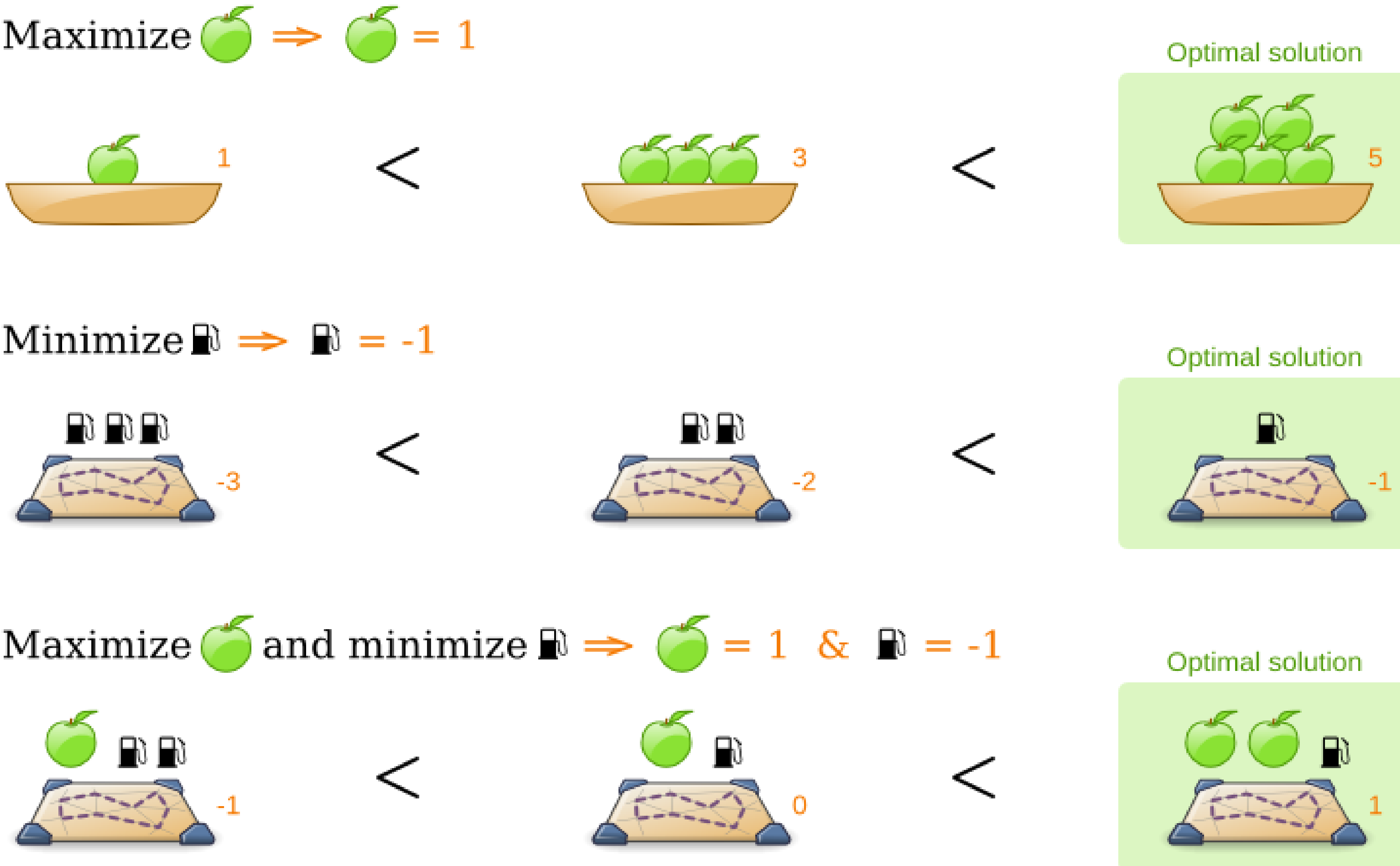
What solution is better?

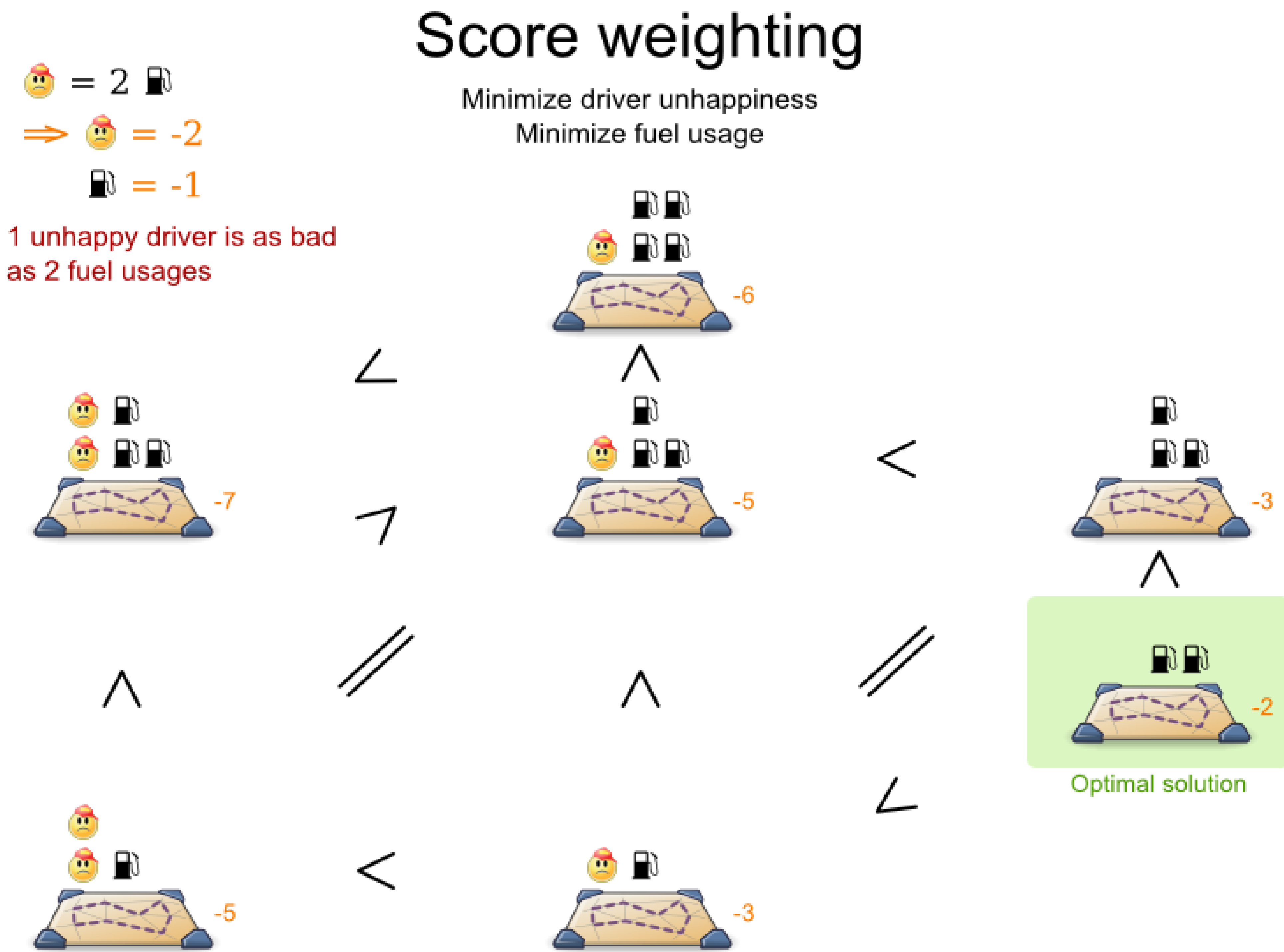


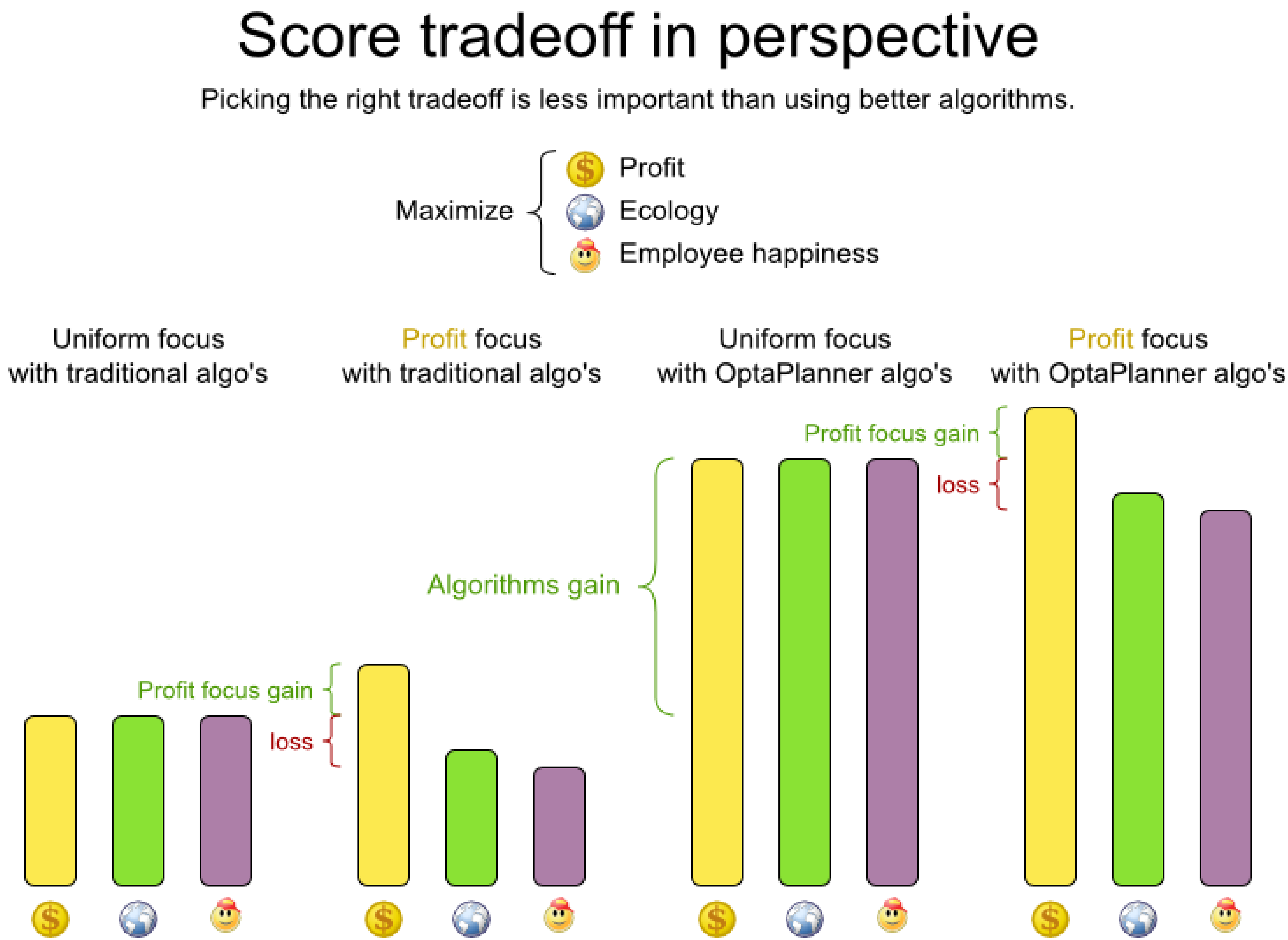
- Need for **objective scoring**
- Better score ⇔ better solution
- Highest score ⇔ optimal solution

Positive and negative constraints

Pick the solution which maximizes apples and minimizes fuel usage

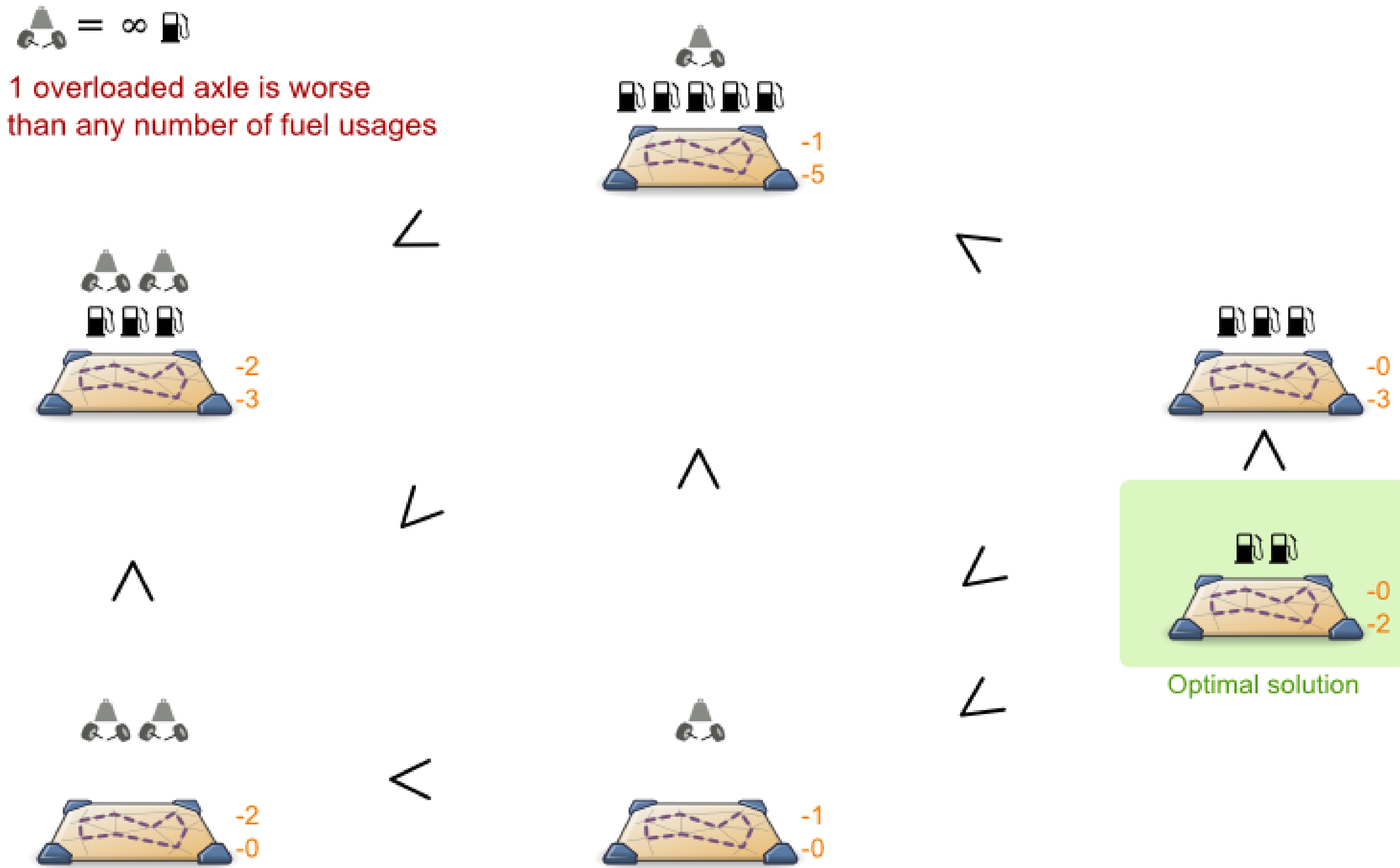



Score weighting
Minimize driver unhappiness
Minimize fuel usage
= 2
= -2
= -1
1 unhappy driver is as bad as 2 fuel usages
-6
-7
-5
-3
-2
Optimal solution
-5
-3


Score tradeoff in perspective
Picking the right tradeoff is less important than using better algorithms.
Maximize
Profit
Ecology
Employee happiness
Uniform focus
with traditional algo's
Profit focus
with traditional algo's
Uniform focus
with OptaPlanner algo's
Profit focus
with OptaPlanner algo's
Profit focus gain
loss
Algorithms gain
Profit focus gain
loss

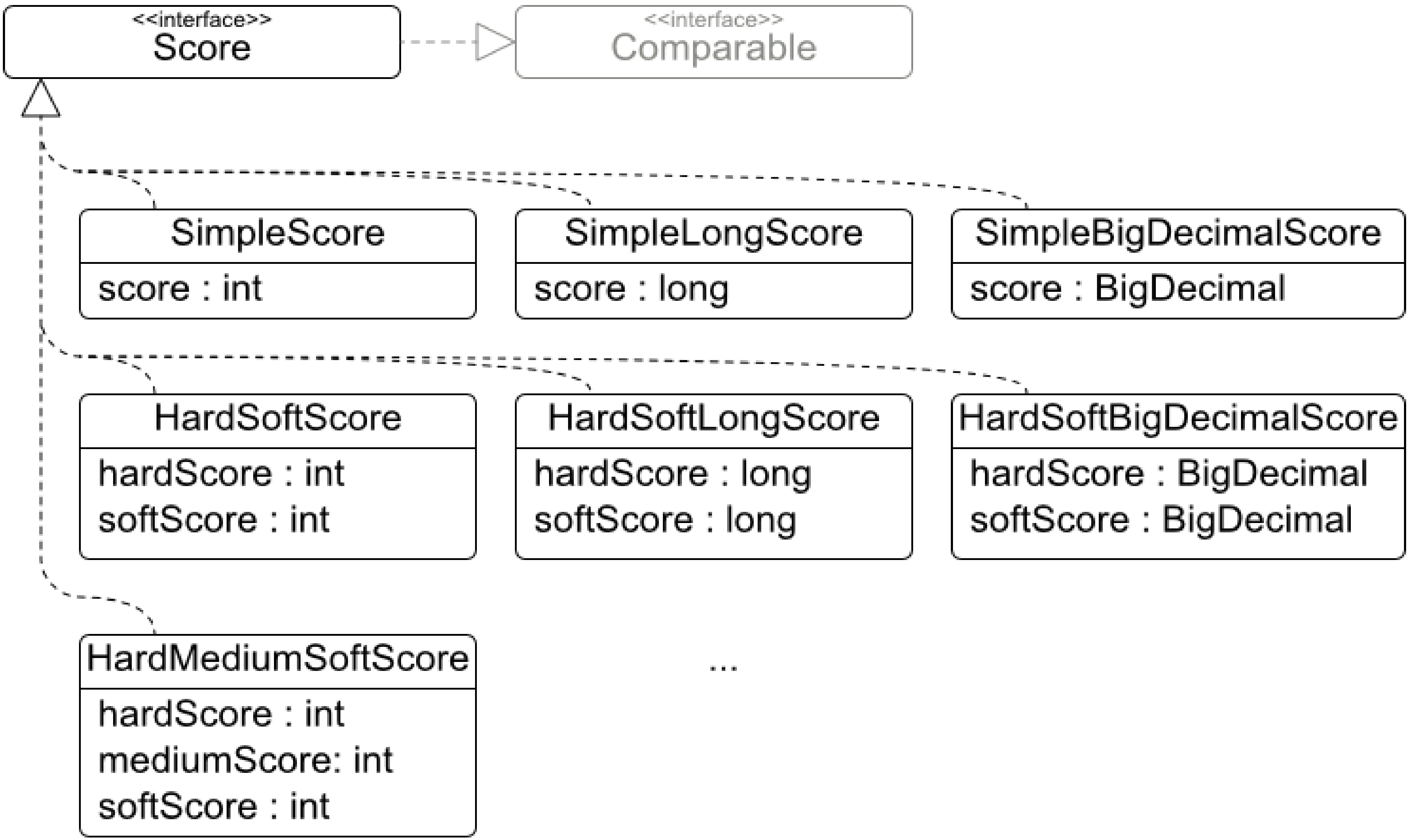
Score levels

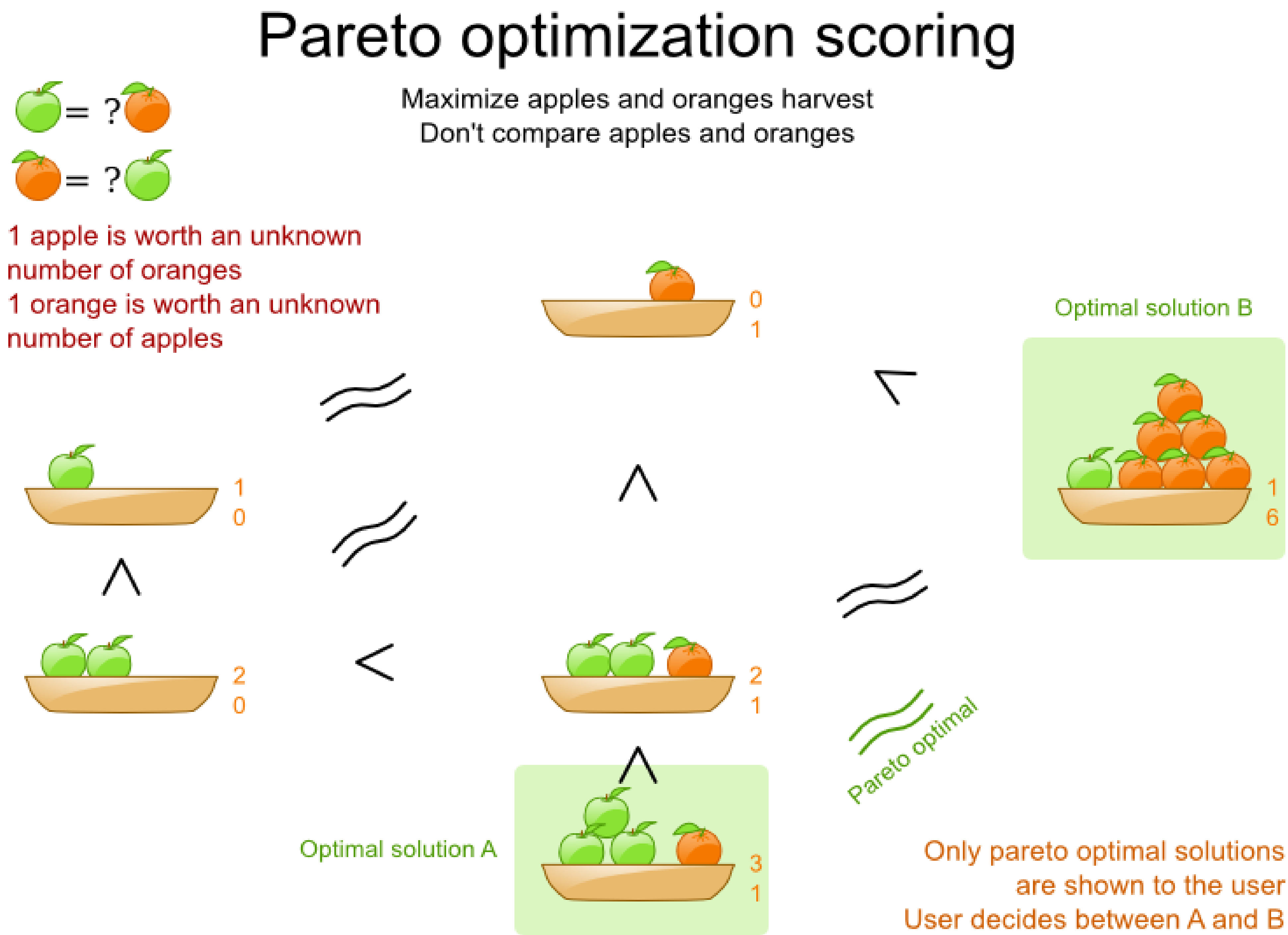
First minimize overloaded truck axles,
then minimize fuel usage



Score class diagram

Choose a Score implementation or write a custom one

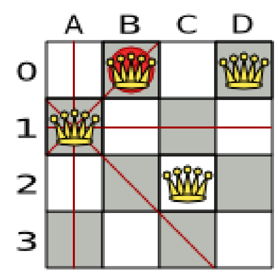




Pareto optimization scoring
= ?
= ?
Maximize apples and oranges harvest
Don't compare apples and oranges
1 apple is worth an unknown number of oranges
1 orange is worth an unknown number of apples
0
1
Optimal solution B
1
6
1
0
2
0
2
1
Pareto optimal
Optimal solution A
3
1
Only pareto optimal solutions are shown to the user
User decides between A and B

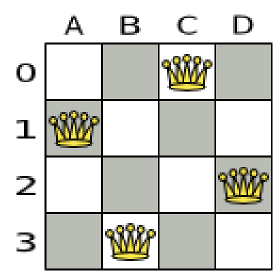
N Queens

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

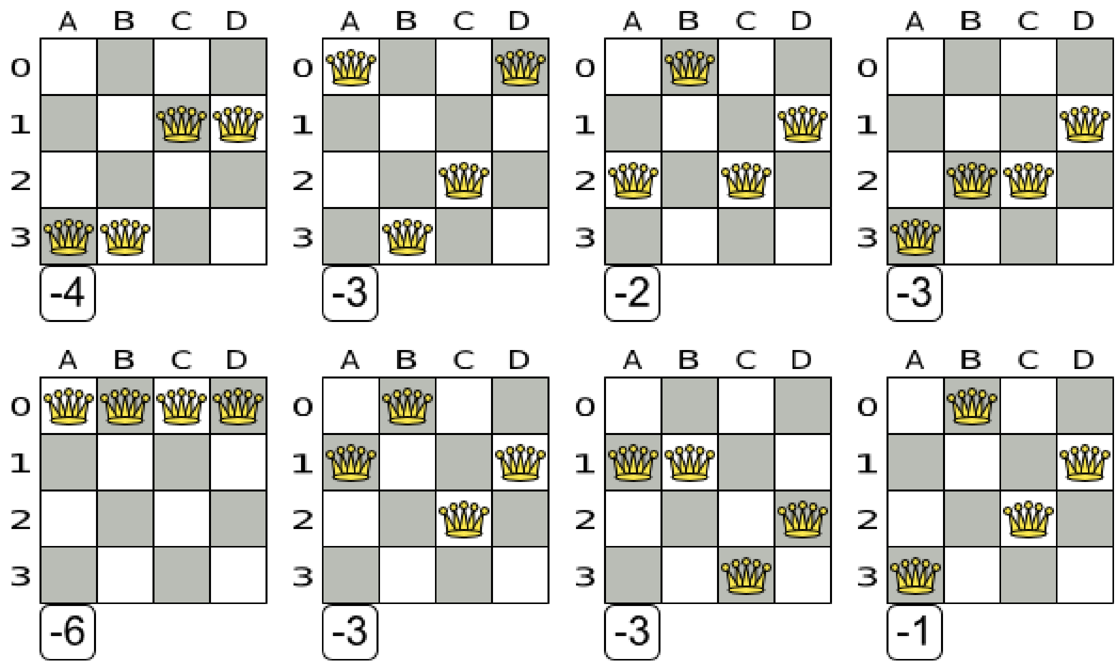


Score = -2

Conflicts: A-B, B-D

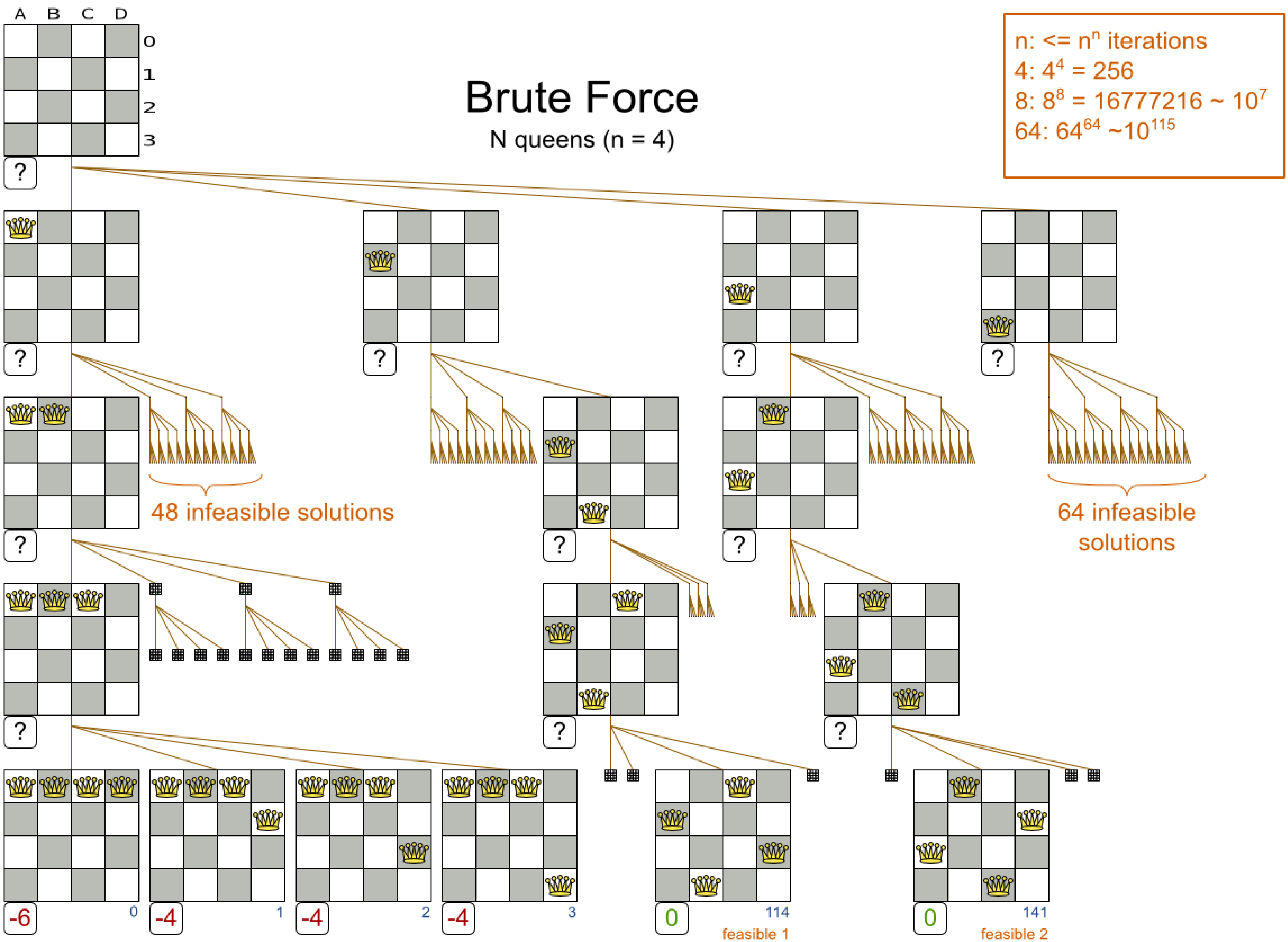


How do we find the best solution?



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

Brute Force

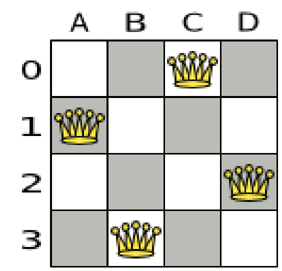


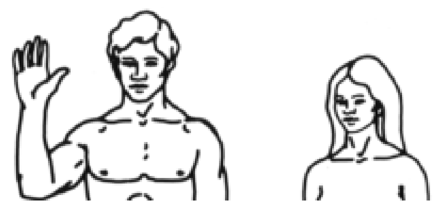
Brute Force
N queens (n = 4)
n: <= n^n iterations
4: 4^4 = 256
8: 8^8 = 16777216 ~ 10^7
64: 64^64 ~10^115
A B C D
0
1
2
3
48 infeasible solutions
64 infeasible solutions
-6
0
-4
1
-4
2
-4
3
0
114
feasible 1
0
141
feasible 2

Brute Force scalability

How many combinations for 100 queens?

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





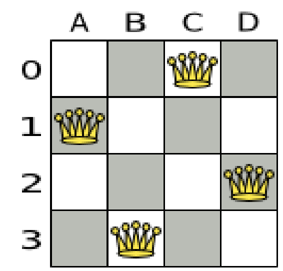
Source: NASA (wikipedia)

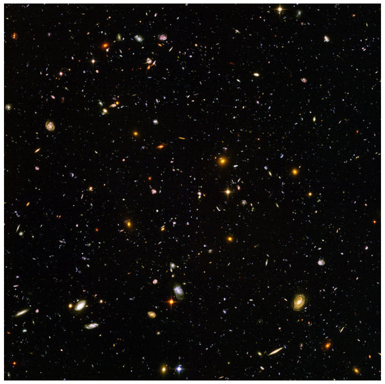
> humans?

7 000 000 000

How many combinations for 100 queens?

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



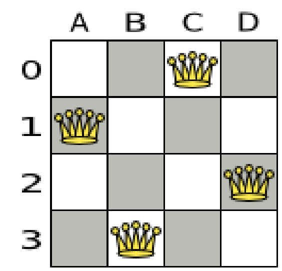


Source: NASA and ESA (wikipedia)

> 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

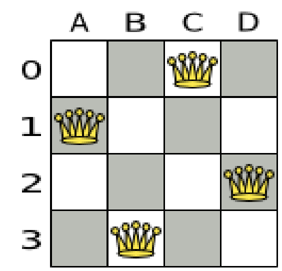


$$100^{100} = 10^{200}$$

```
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



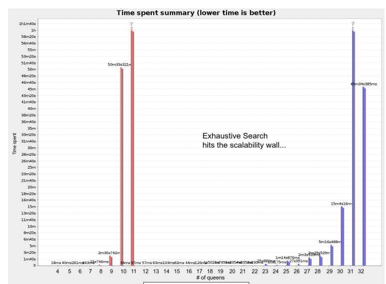
$$n^n$$

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

How long?

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

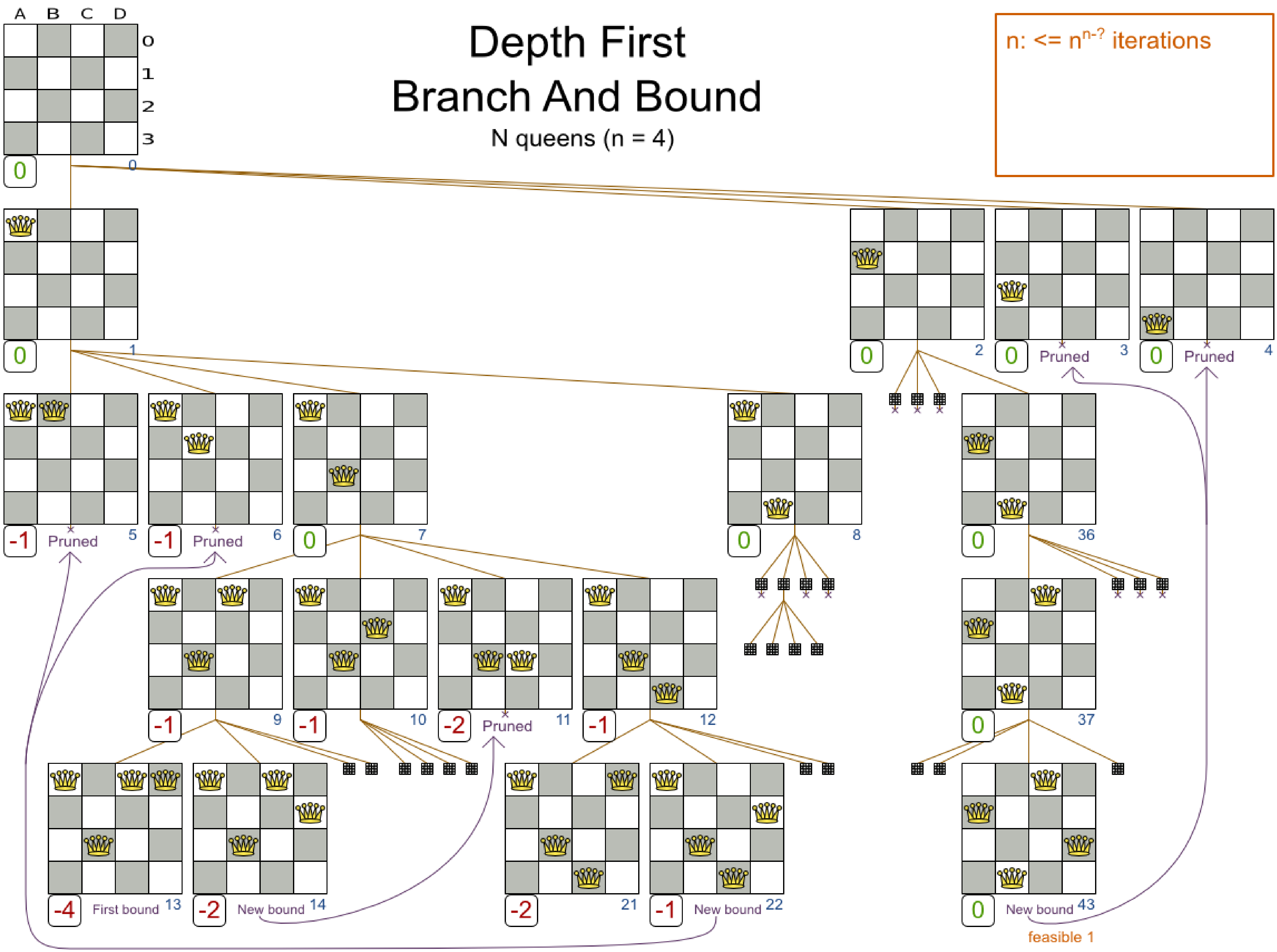
Queens	Combinations	Calculation time
100	$100^{100} = 10^{200}$	10^{180} years
1000	$1000^{1000} = 10^{3000}$	10^{2980} years
10000	$10000^{10000} = 10^{40000}$	10^{39980} years



Moore's law == a drop in the ocean

Depth First Branch And Bound

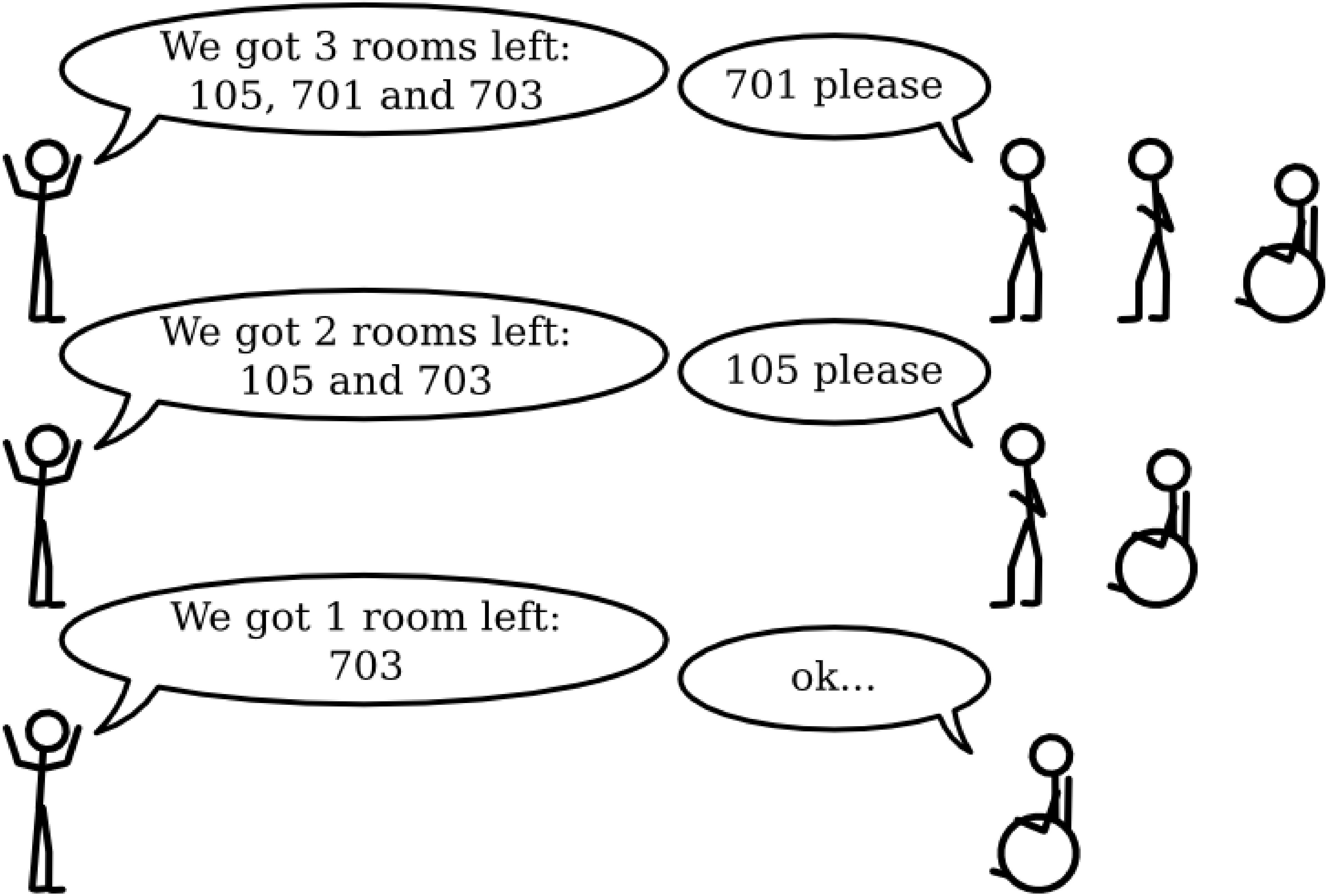
N queens (n = 4)

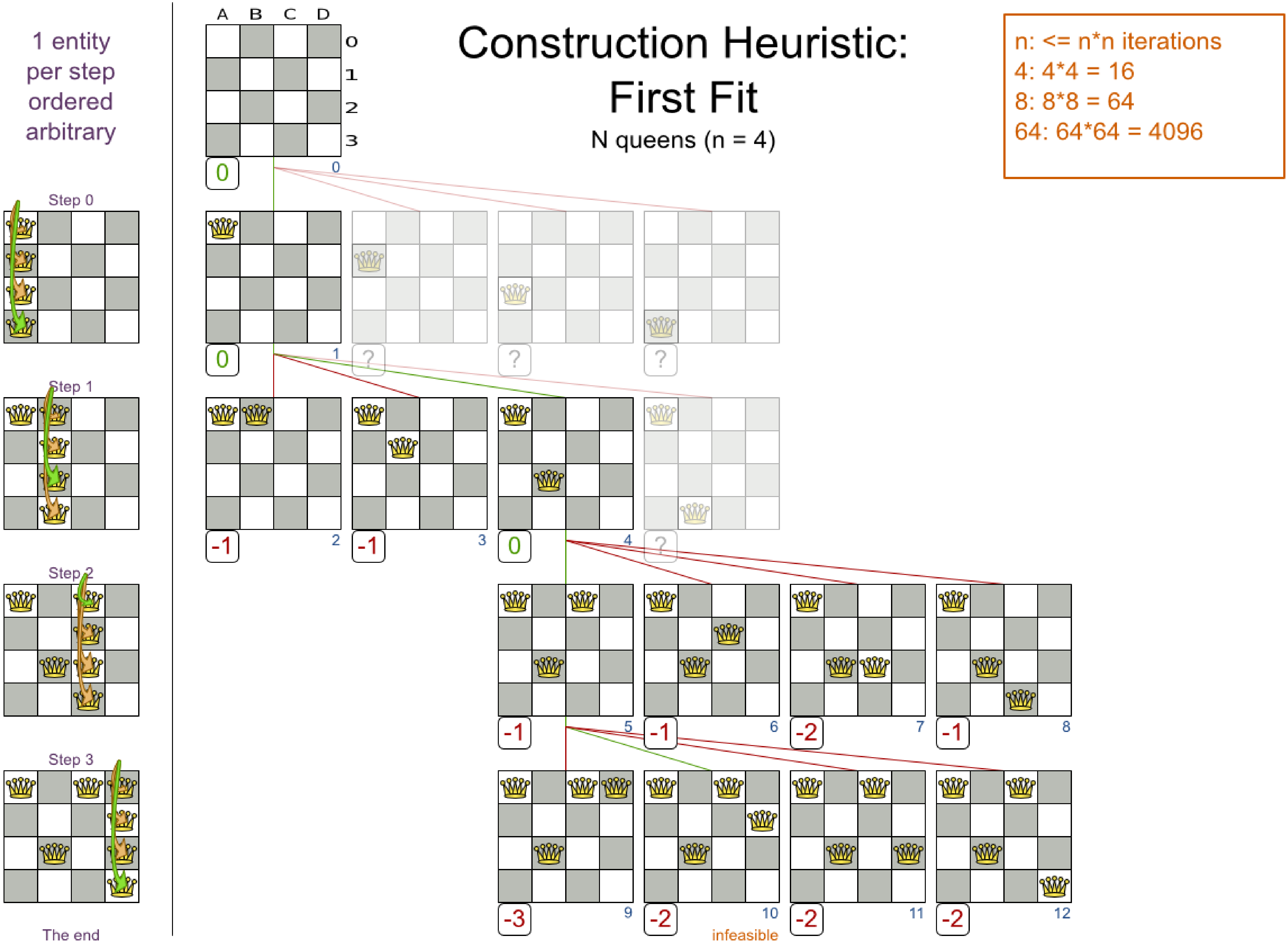


Exhaustive Search doesn’t scale

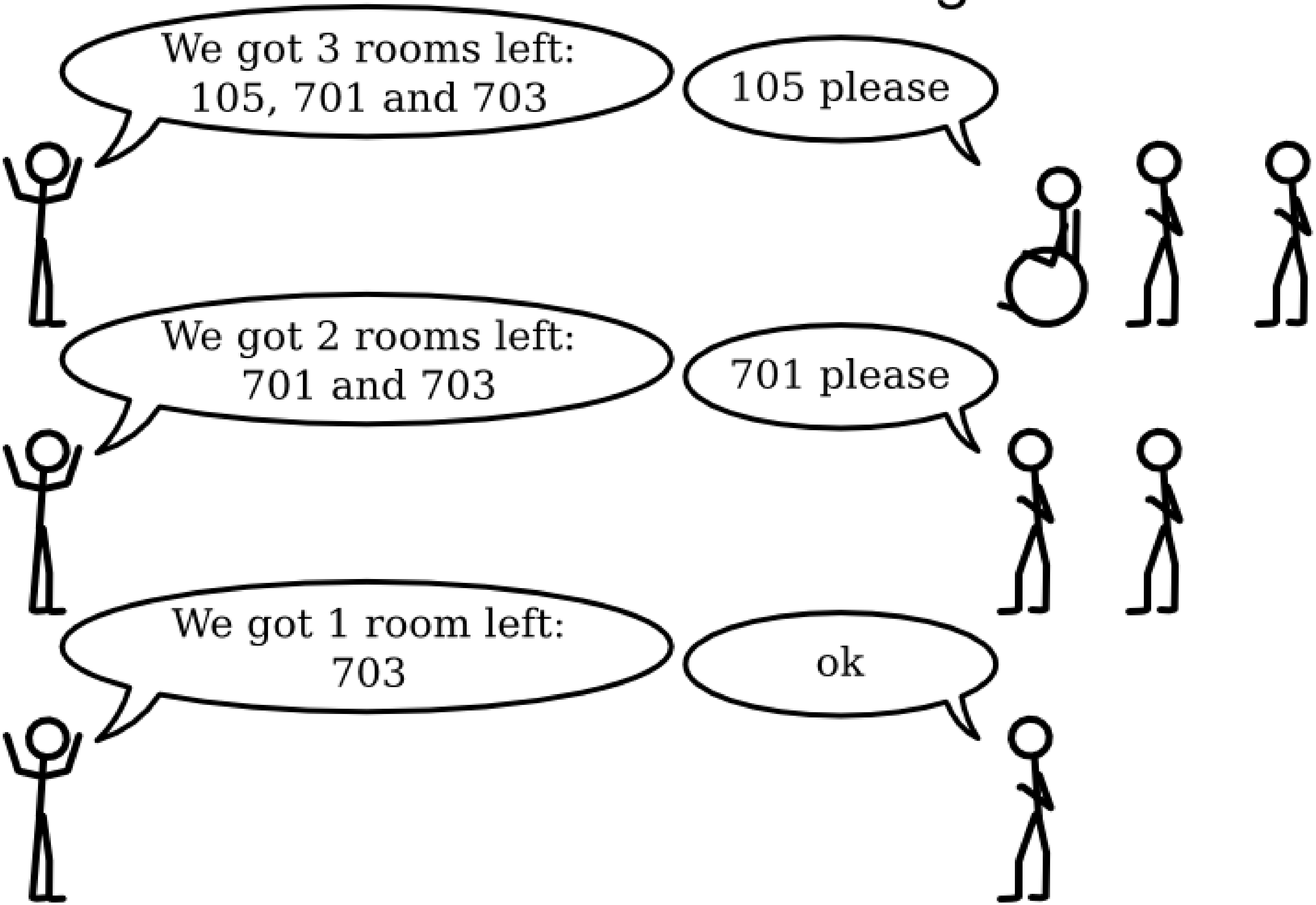
- Branches explode exponentially
- Not enough CPU
- Not enough memory

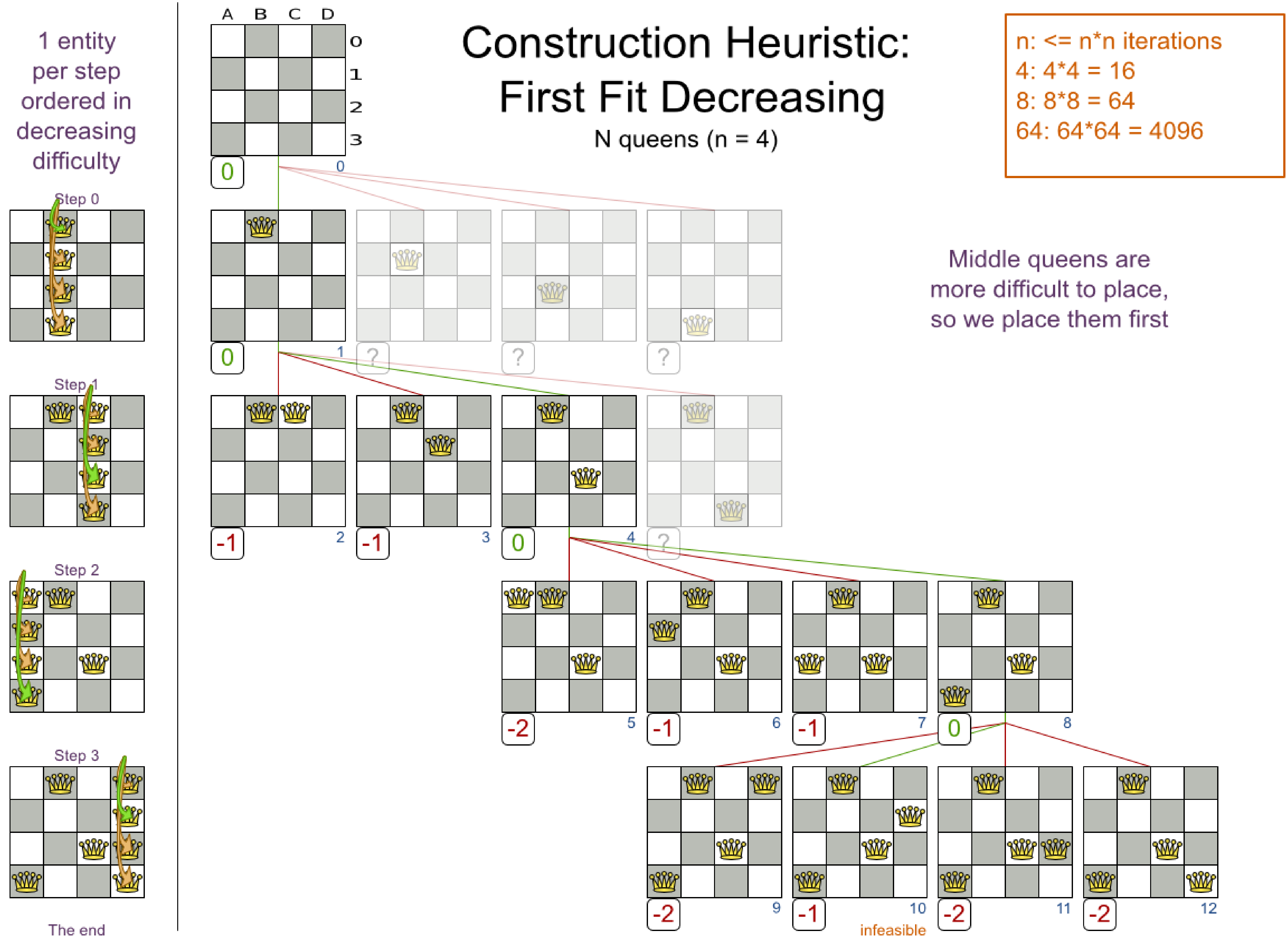
First Fit


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)
n: <= n*n iterations
4: 4*4 = 16
8: 8*8 = 64
64: 64*64 = 4096
1 entity
per step
ordered
arbitrary
Step 0
Step 1
Step 2
Step 3
The end
A B C D
0
1
2
3
infeasible

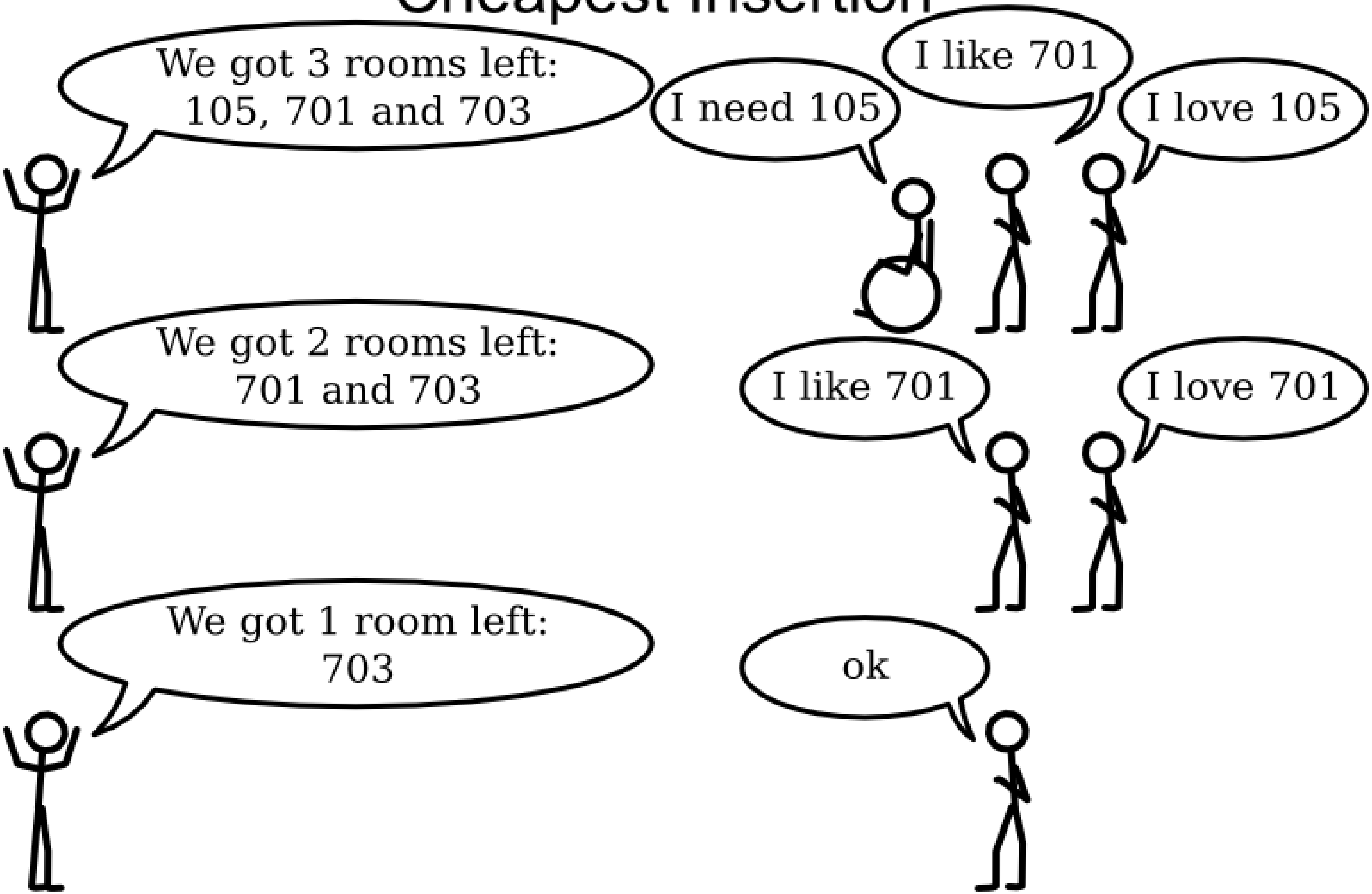
First Fit Decreasing

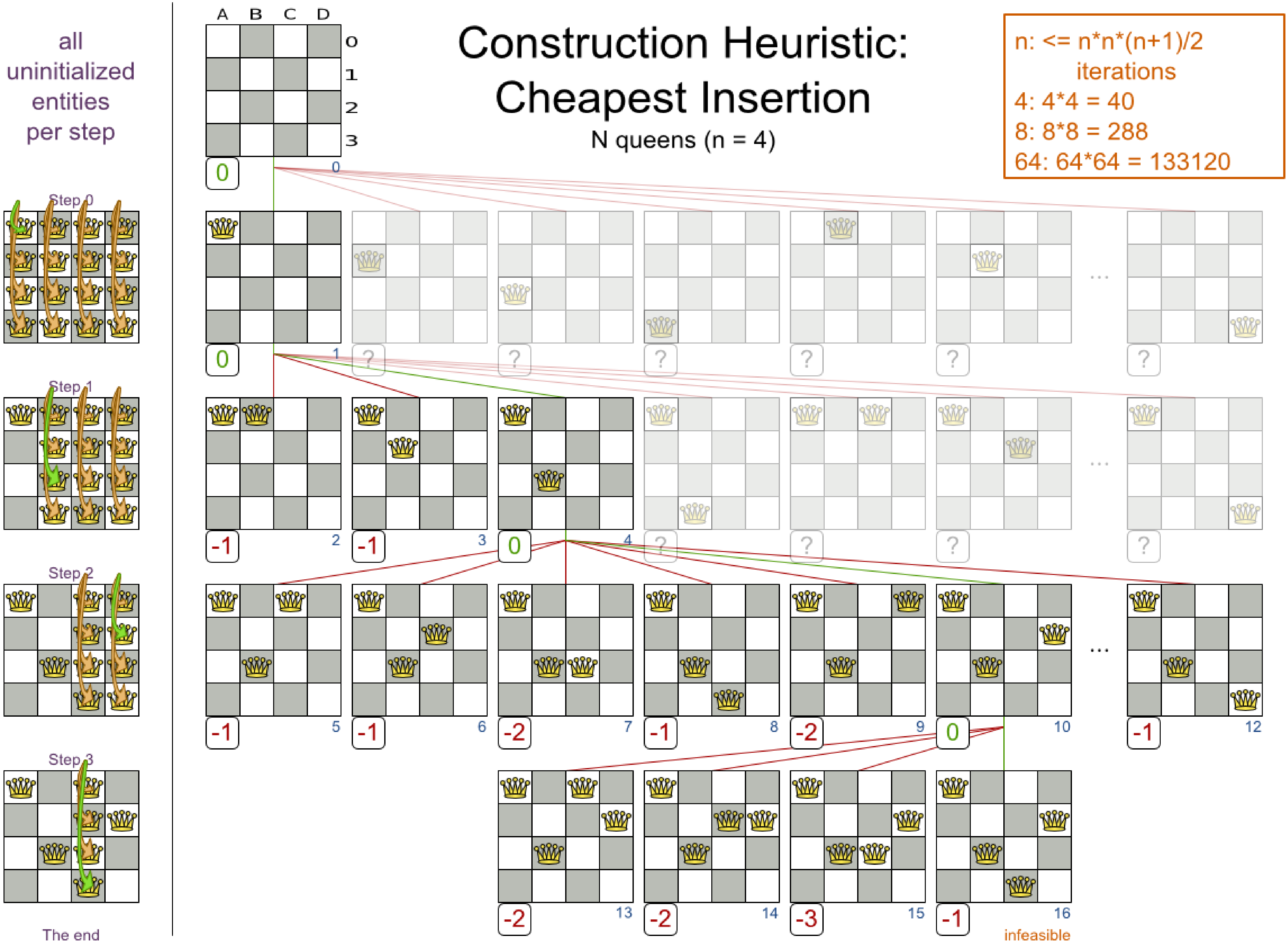




Construction Heuristic:
First Fit Decreasing
N queens (n = 4)
n: <= n*n iterations
4: 4*4 = 16
8: 8*8 = 64
64: 64*64 = 4096
1 entity per step ordered in decreasing difficulty
Step 0
Step 1
Step 2
Step 3
The end
A B C D
0 1 2 3
Middle queens are more difficult to place, so we place them first
infeasible

Cheapest Insertion

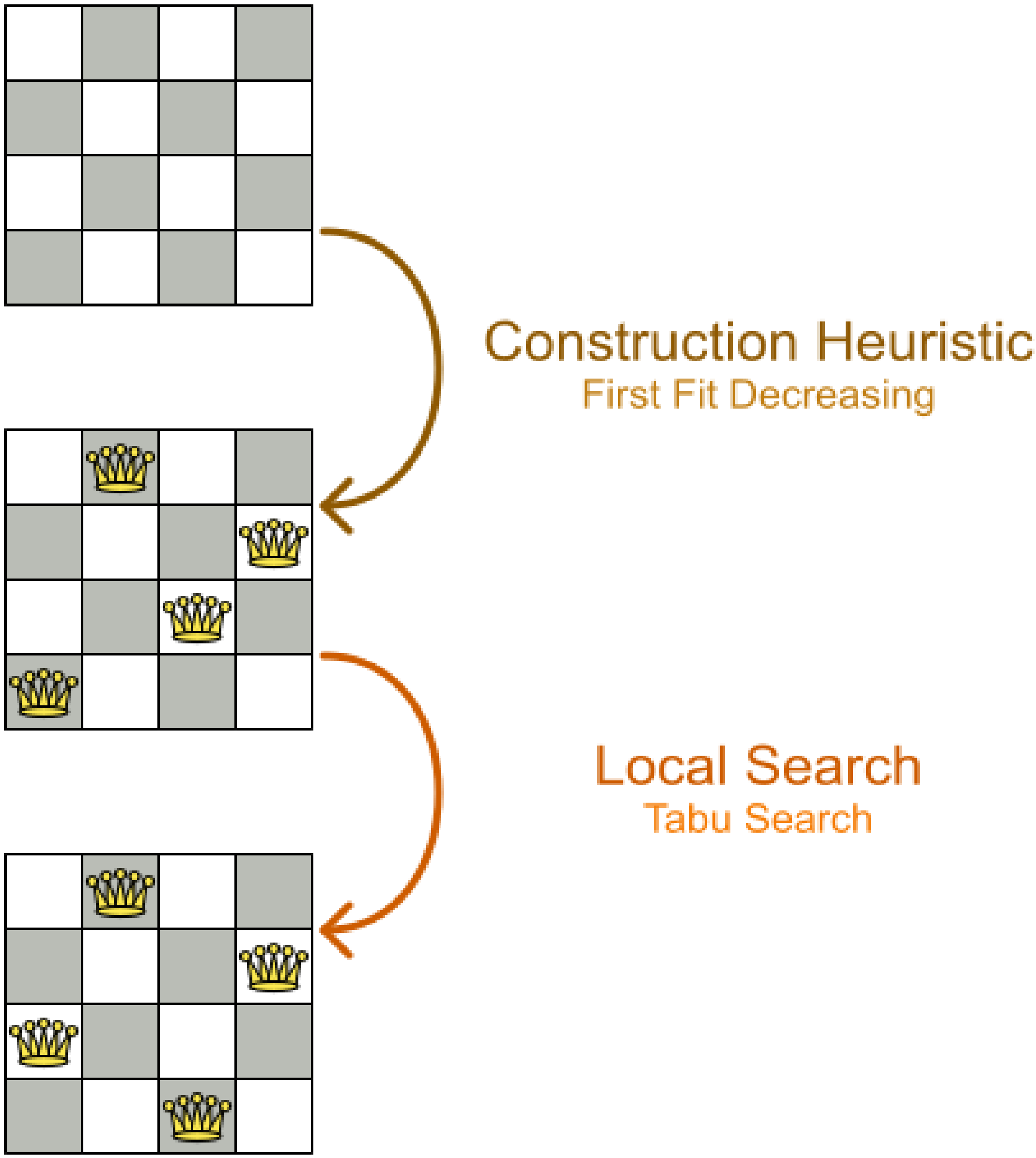




Construction Heuristic:
Cheapest Insertion
N queens (n = 4)
all
uninitialized
entities
per step
n: <= n*n*(n+1)/2
iterations
4: 4*4 = 40
8: 8*8 = 288
64: 64*64 = 133120
A B C D
0 1 2 3
Step 0
Step 1
Step 2
Step 3
The end
infeasible

General phase sequence

First a Construction Heuristic,
then a Local Search



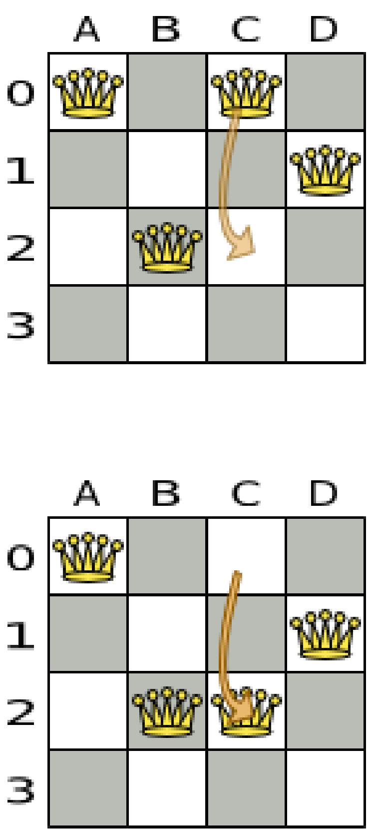
Local Search:
Move types

- Change move
- Swap move
- ...

ChangeMove

Change 1 variable of 1 entity

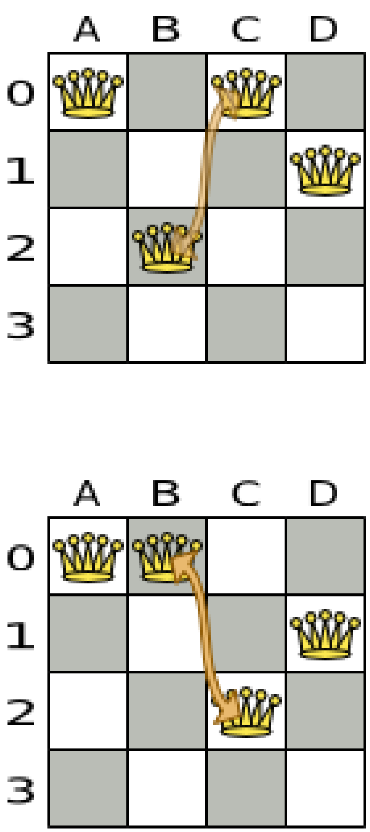
N queens



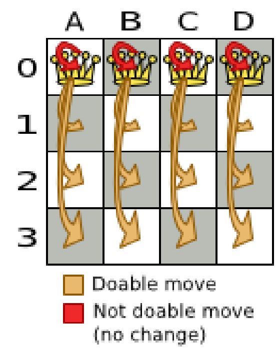
SwapMove

Swap all variables of 2 entities

N queens

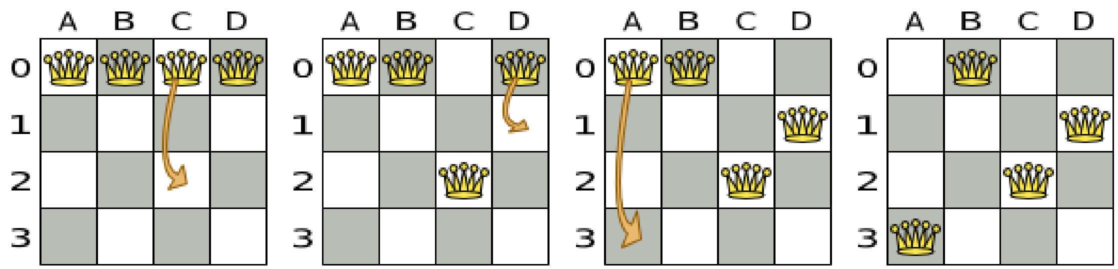


All change moves



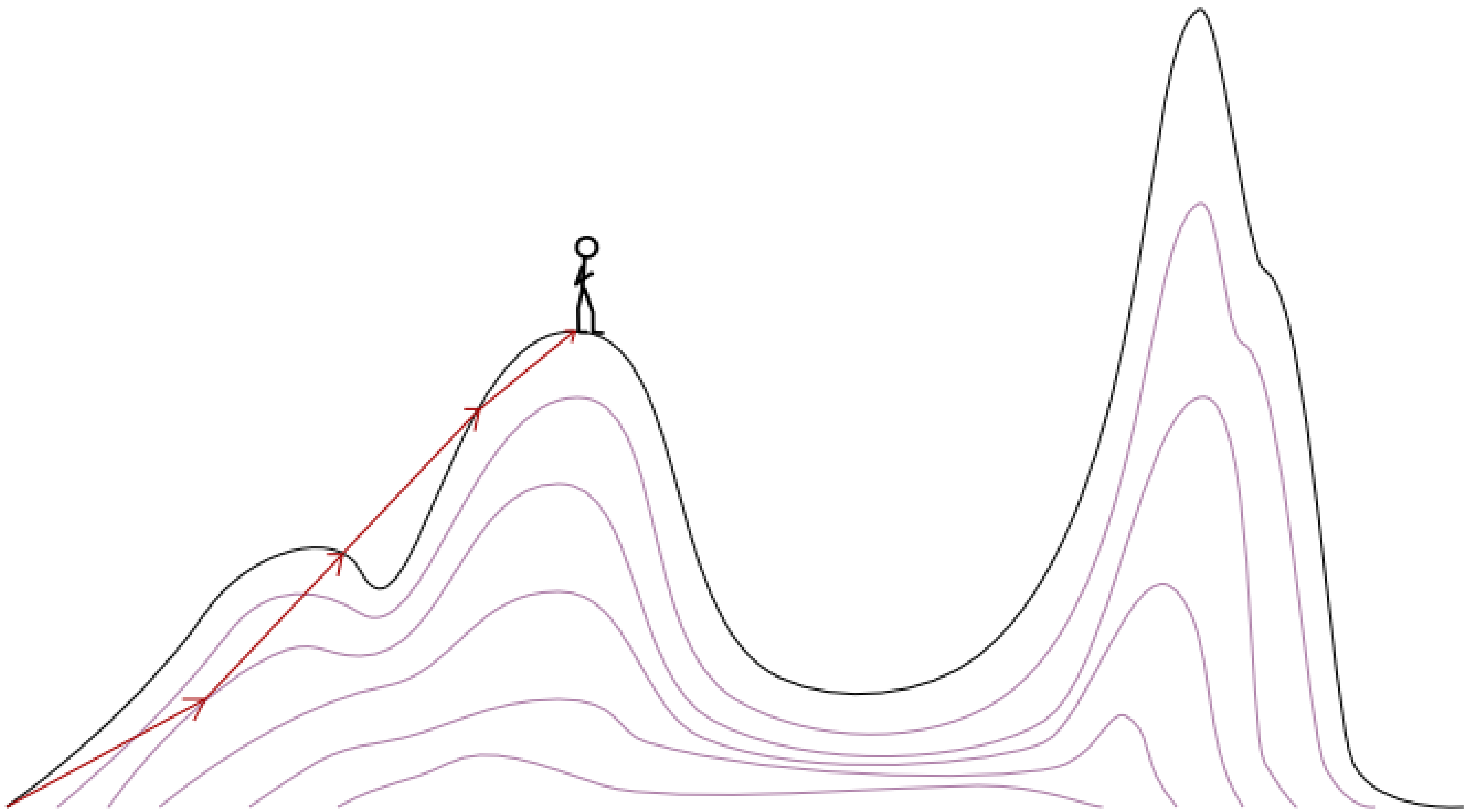
n	# moves	# solutions
4	16	256
8	64	16777216
64	4096	10^{116}
n	n^2	n^n

Multiple moves



Multiple moves can reach any solution

Hill climbing

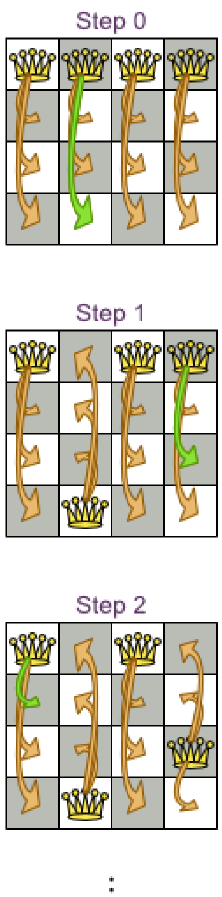


Local Search: Hill Climbing

N queens (n = 4)

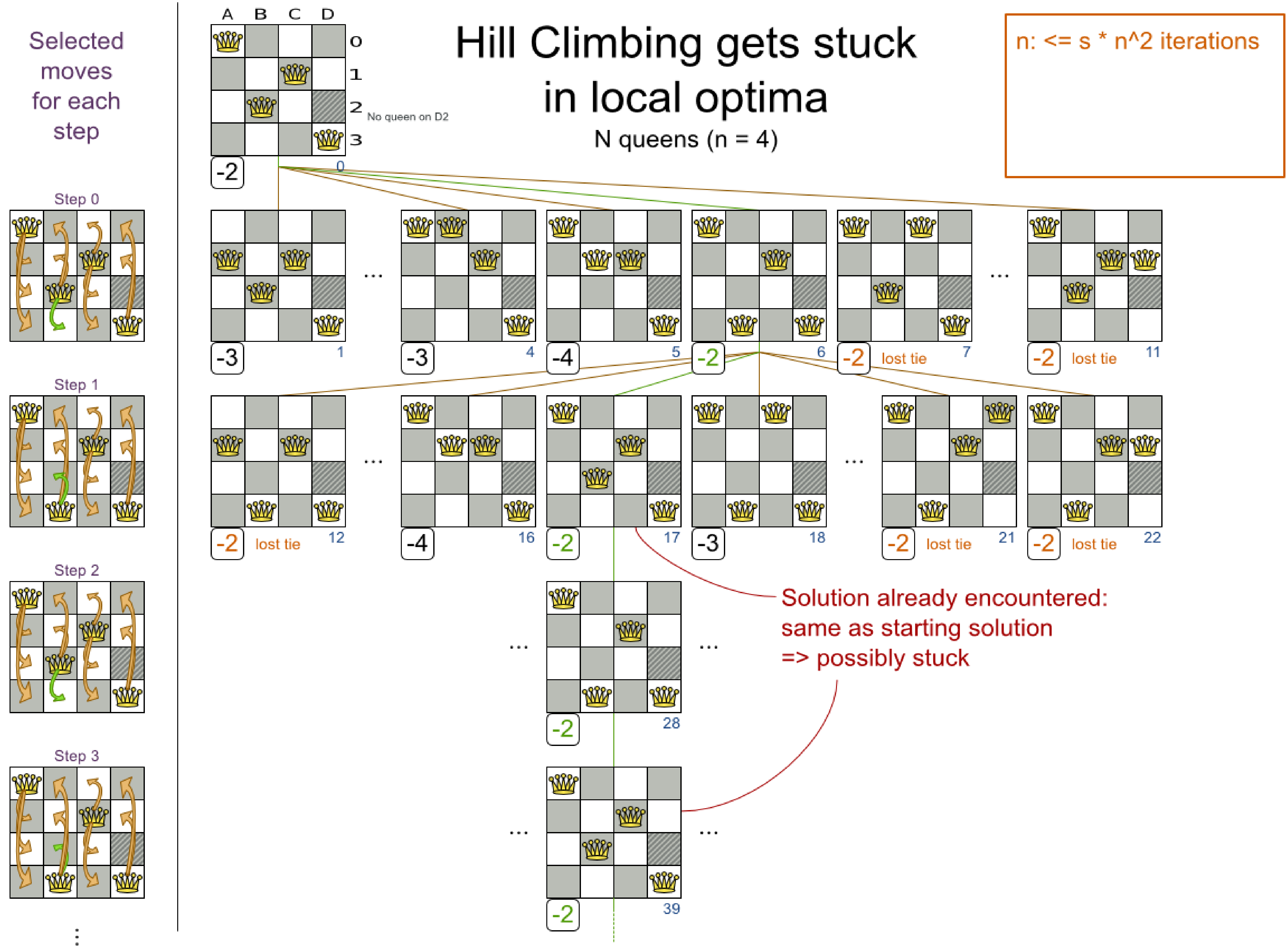
n: <= s * n^2 iterations

Selected moves for each step

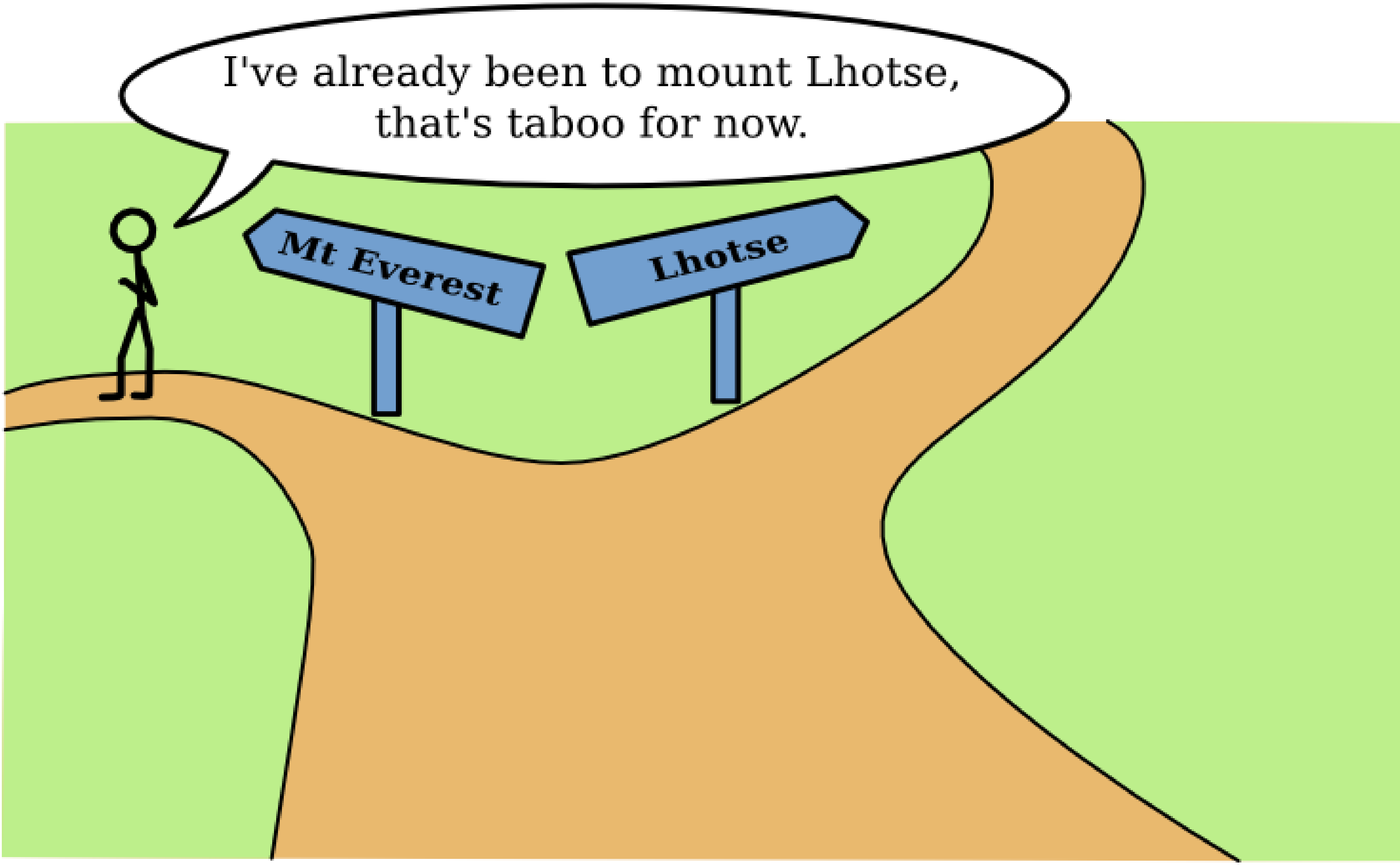


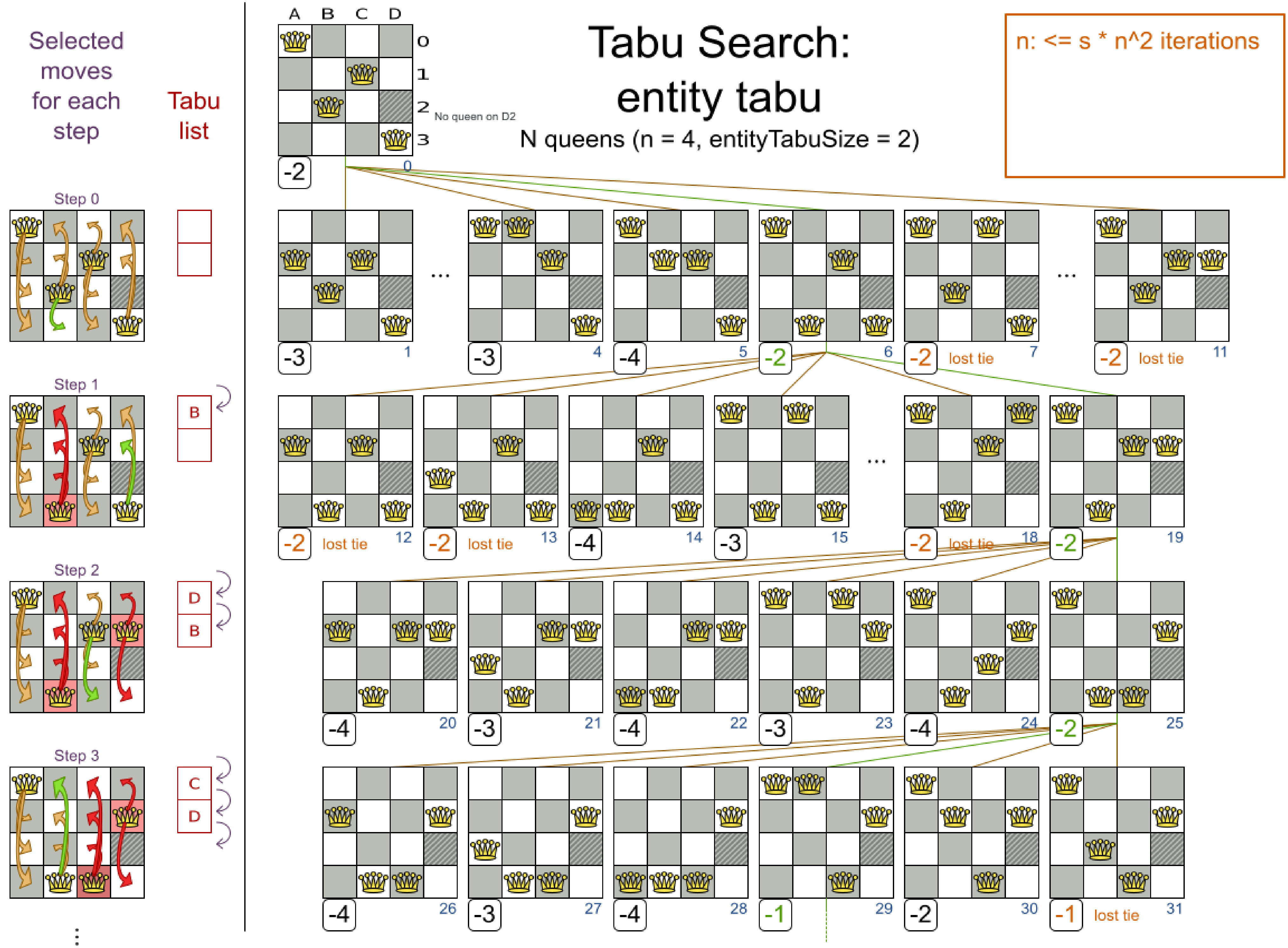
Uses a search path, not a search tree

⇒ highly scalable


Hill Climbing gets stuck in local optima
N queens (n = 4)
n: <= s * n^2 iterations
Selected moves for each step
Step 0
Step 1
Step 2
Step 3
A B C D
0 1 2 3
No queen on D2
-2 0
-3 1
-3 4
-4 5
-2 6
-2 lost tie 7
-2 lost tie 11
-2 lost tie 12
-4 16
-2 17
-3 18
-2 lost tie 21
-2 lost tie 22
-2 28
-2 39
Solution already encountered:
same as starting solution
=> possibly stuck

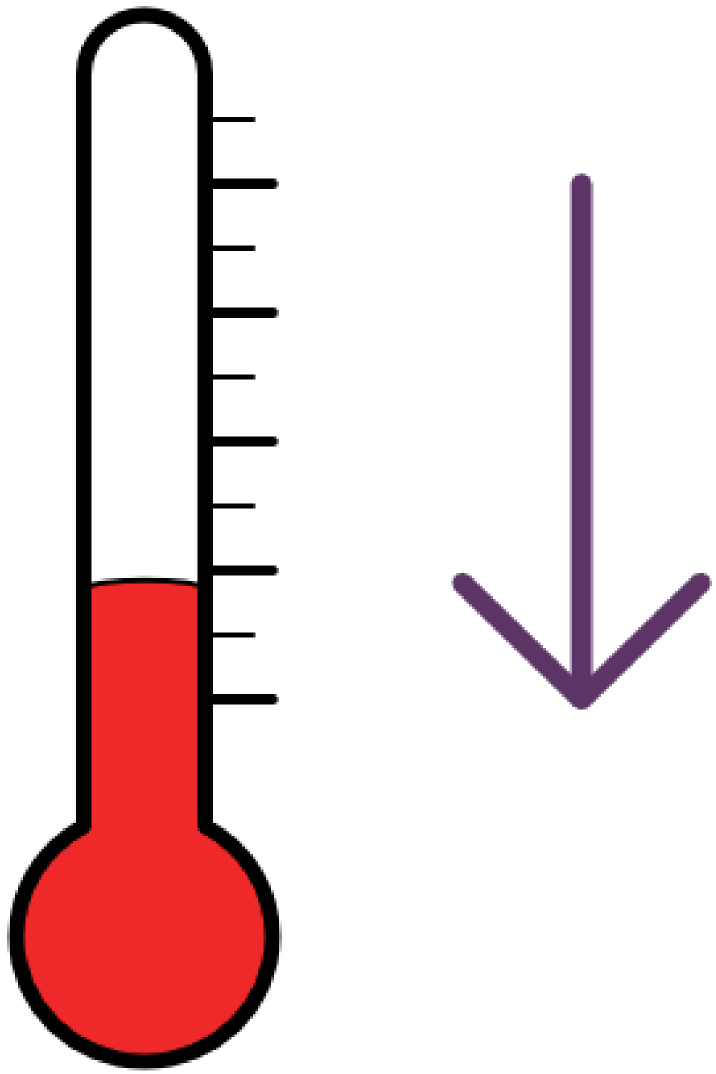
Tabu Search

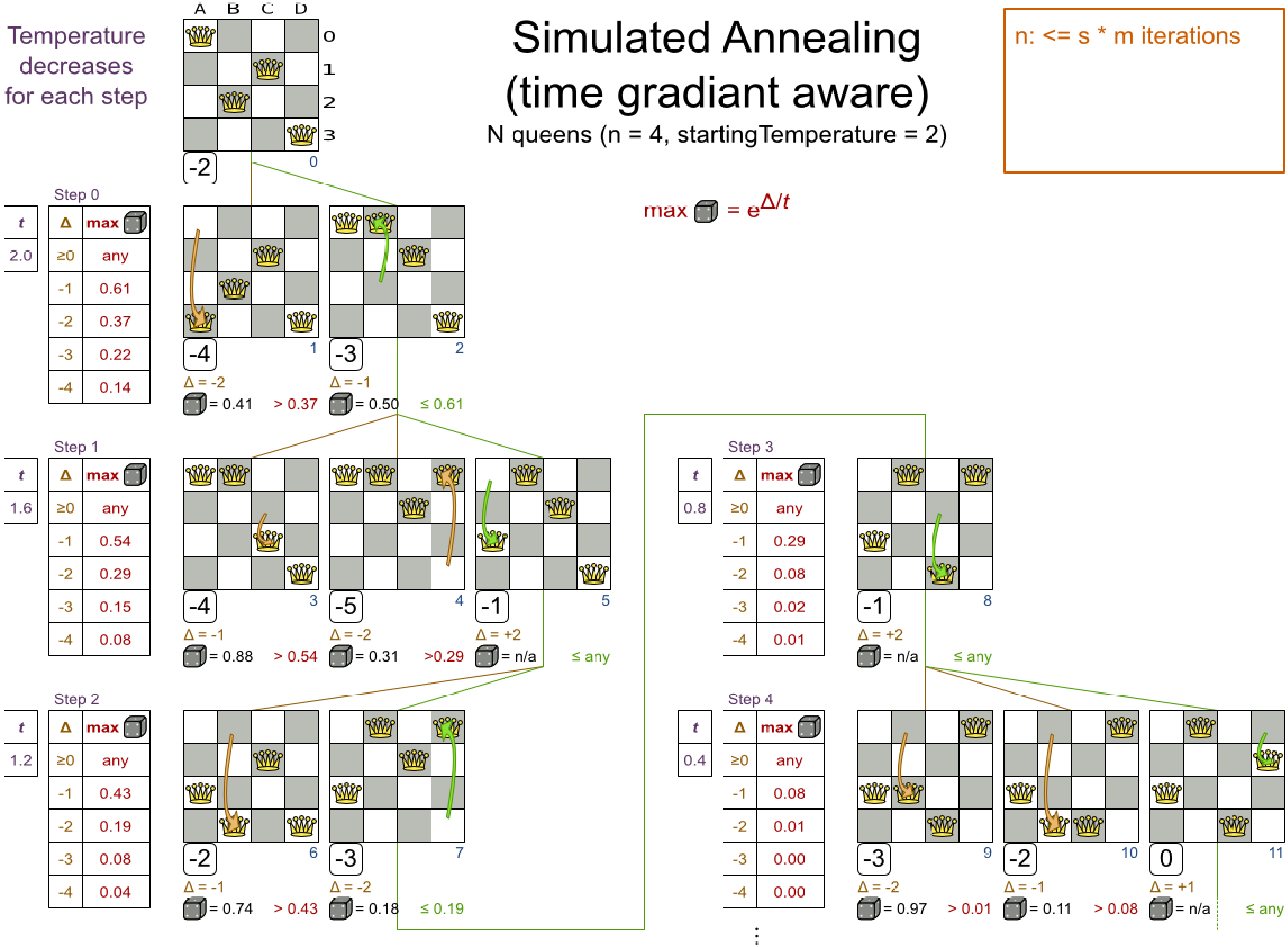

I've already been to mount Lhotse,
that's taboo for now.
Mt Everest
Lhotse



Tabu Search:
entity tabu
N queens (n = 4, entityTabuSize = 2)
n: <= s * n^2 iterations
Selected moves for each step
Tabu list
A B C D
0 1 2 3
No queen on D2
Step 0
Step 1
Step 2
Step 3
B
D
B
C
D
-2 0
-3 1
-3 4
-4 5
-2 6
-2 lost tie 7
-2 lost tie 11
-2 lost tie 12
-2 lost tie 13
-4 14
-3 15
-2 lost tie 18
-2 19
-4 20
-3 21
-4 22
-3 23
-4 24
-2 25
-4 26
-3 27
-4 28
-1 29
-2 30
-1 lost tie 31

Simulated Annealing





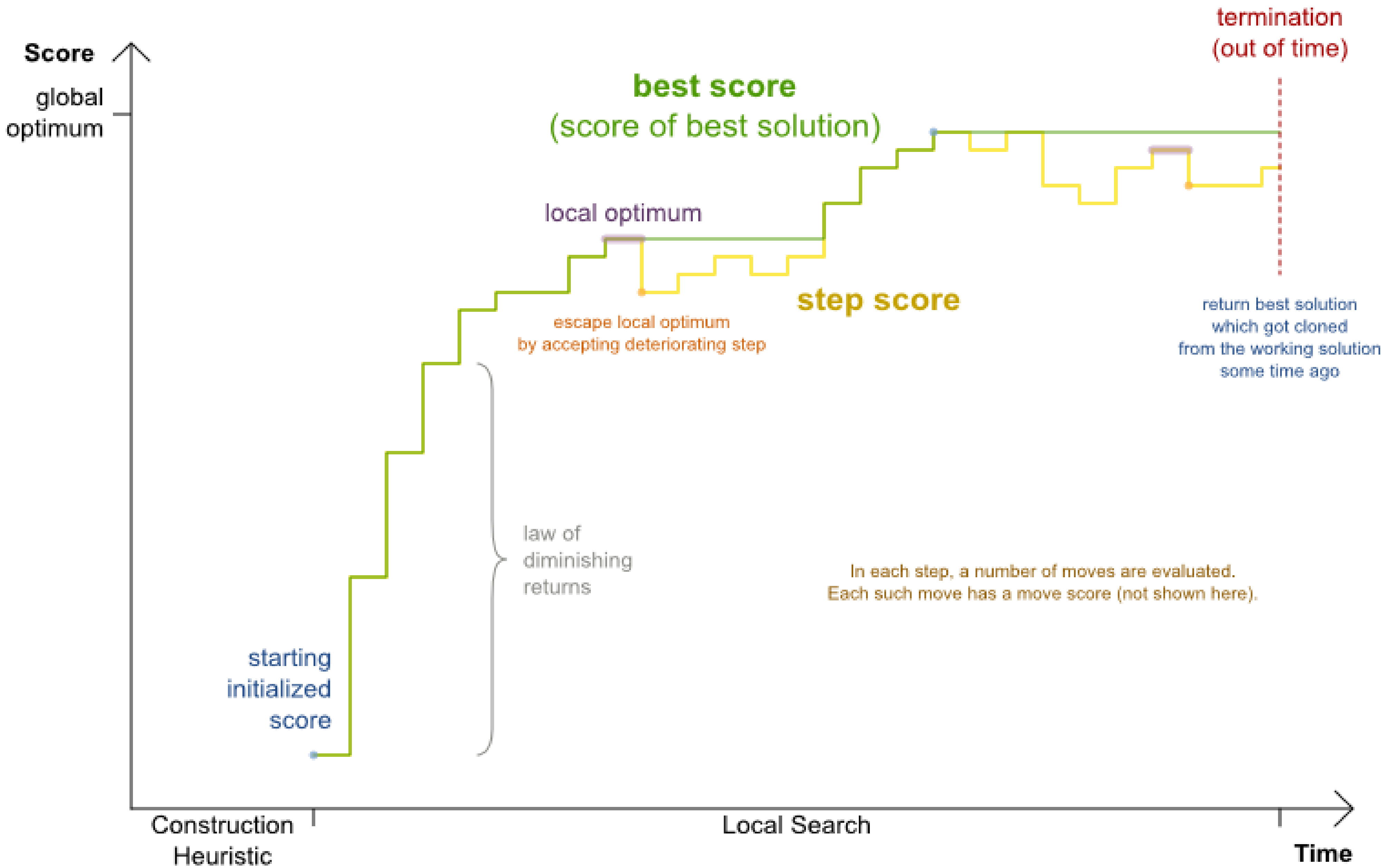
Simulated Annealing
(time gradiant aware)
N queens (n = 4, startingTemperature = 2)
n: <= s * m iterations
Temperature decreases for each step
max = e^Δ/t
A B C D
0
1
2
3
-2
Step 0
t
2.0
Δ max
≥0 any
-1 0.61
-2 0.37
-3 0.22
-4 0.14
-4
1
Δ = -2
= 0.41 > 0.37
-3
2
Δ = -1
= 0.50 ≤ 0.61
Step 1
t
1.6
Δ max
≥0 any
-1 0.54
-2 0.29
-3 0.15
-4 0.08
-4
3
Δ = -1
= 0.88 > 0.54
-5
4
Δ = -2
= 0.31 >0.29
-1
5
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
-2
6
Δ = -1
= 0.74 > 0.43
-3
7
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
-1
8
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
-3
9
Δ = -2
= 0.97 > 0.01
-2
10
Δ = -1
= 0.11 > 0.08
0
11
Δ = +1
= n/a ≤ any

Late acceptance

Late Acceptance
N queens (n = 4, lateAcceptanceSize = 3)
n: <= s * m iterations
Late acceptance list
A B C D
0 1 2 3
Step 0
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
-4 0
-2 ≥ any 1
-3 ≥ any 2
-5 < -4 3
-3 ≥ -4 4
-3 < -2 5
Accepted if HC enabled
-4 < -2 6
-1 ≥ -2 7
-3 ≥ -3 8
-4 < -3 9
-2 ≥ -3 10
-2 < -1 11
Accepted if HC enabled
-5 < -1 12
0 ≥ -1 13

Local Search score over time

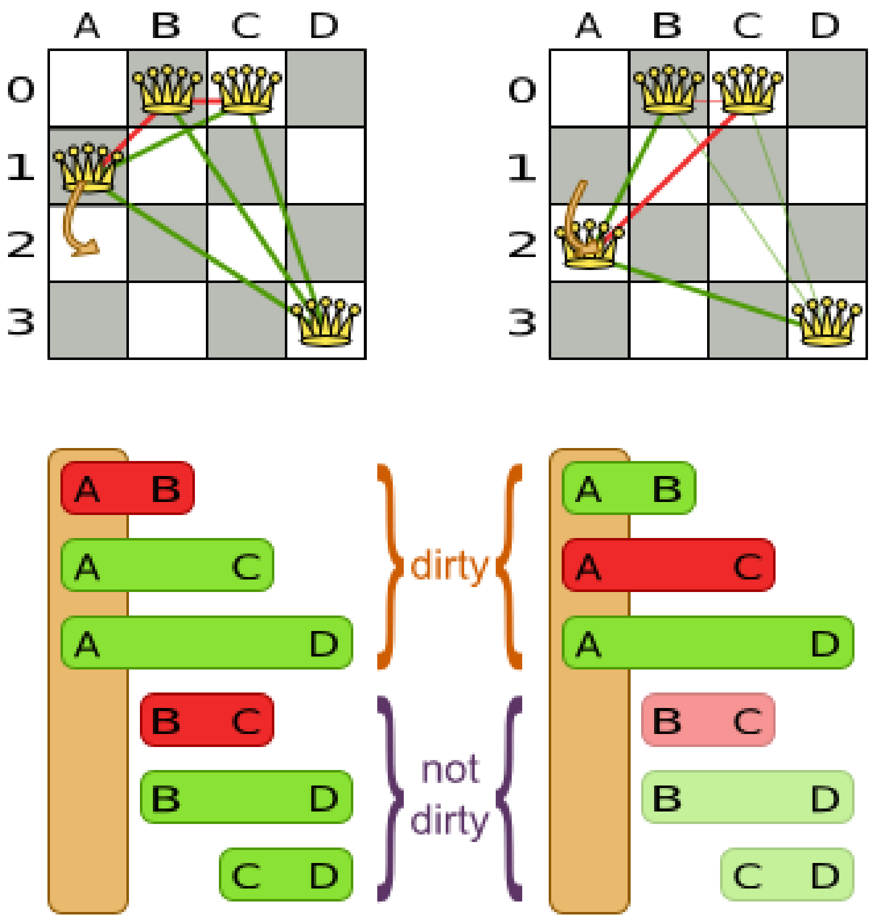
In 1 Local Search run, do not confuse starting initialized score, best score, step score and move score.



Performance tricks

Incremental score calculation

Incremental score calculation is much more scalable because only the delta is calculated.



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
n	*n-1*	*of*	*n*(n-1)/2*	time /	*(n/2)*

Summary

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

Distribution zip

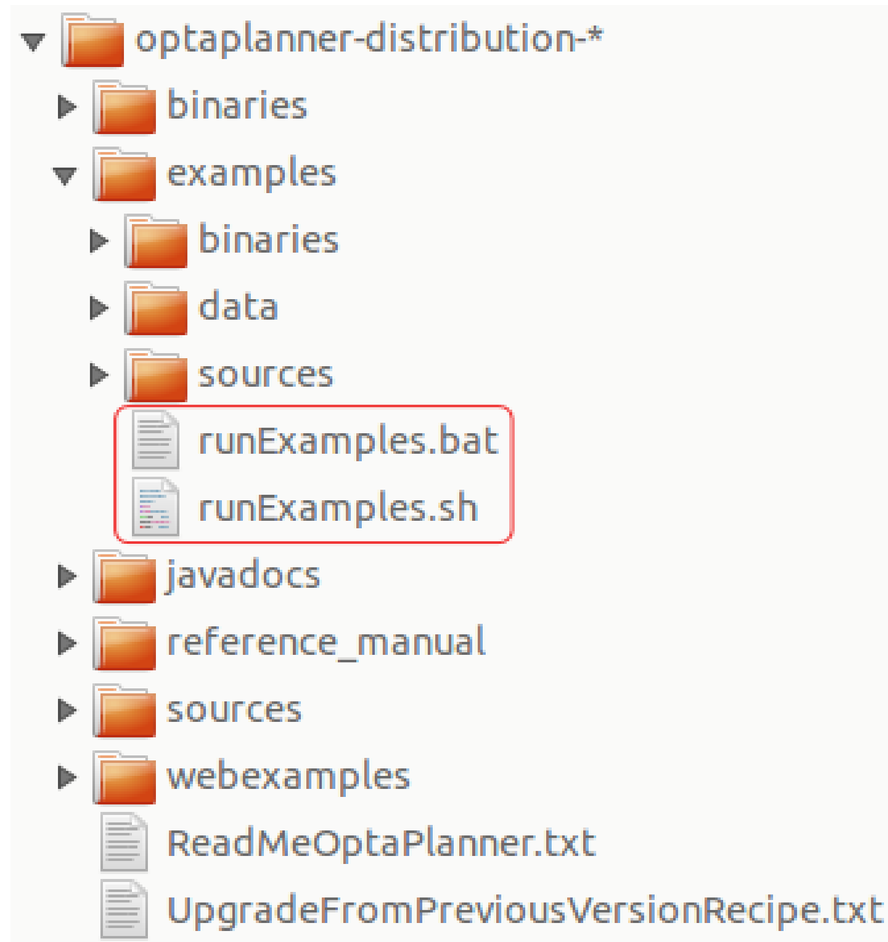
Running the examples locally

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/kiegroup/optaplanner (https://github.com/kiegroup/optaplanner)

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

Q & A

Homepage	www.optaplanner.org (https://www.optaplanner.org)
Slides	www.optaplanner.org/learn/slides.html (https://www.optaplanner.org/learn/slides.html)
User guide	www.optaplanner.org/learn/documentation.html (https://www.optaplanner.org/learn/documentation.html
Feedback	@GeoffreyDeSmet (https://twitter.com/GeoffreyDeSmet)