

Tasks Optimization

Understand the chained models

Donato Marrazzo

*EMEA Senior Solutions Architect for Business Process Automation,
Decision Management and Business Optimization*



Tasks are everywhere!
A common organization problem is to distribute
tasks to people or machines.



The challenger

Customer Profile

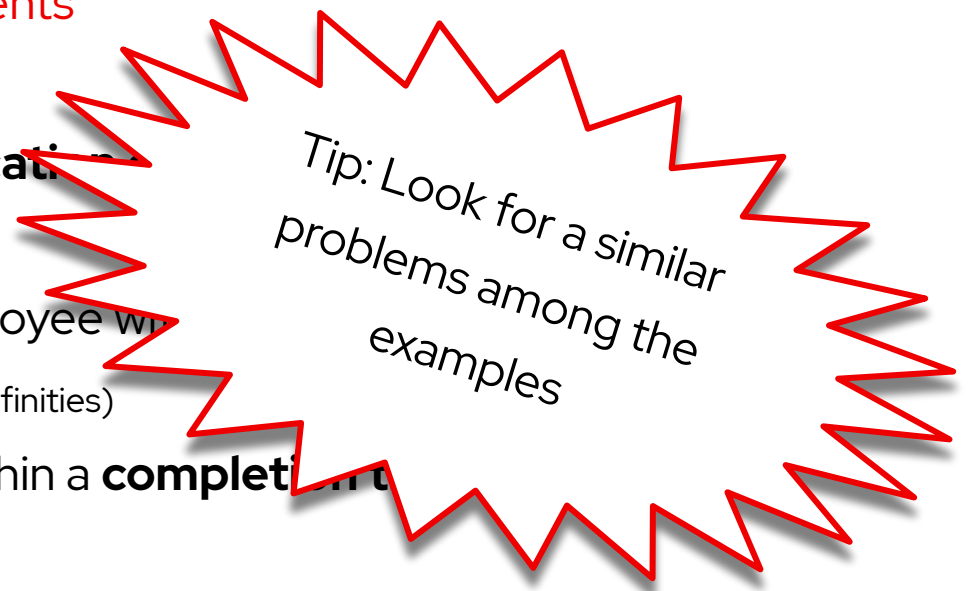
A big **retail group**

- ▶ Omni-channel, multi-local, multi-format and
- ▶ Operating in 30 countries
- ▶ 300000+ employees
- ▶ 12000+ stores

The challenge

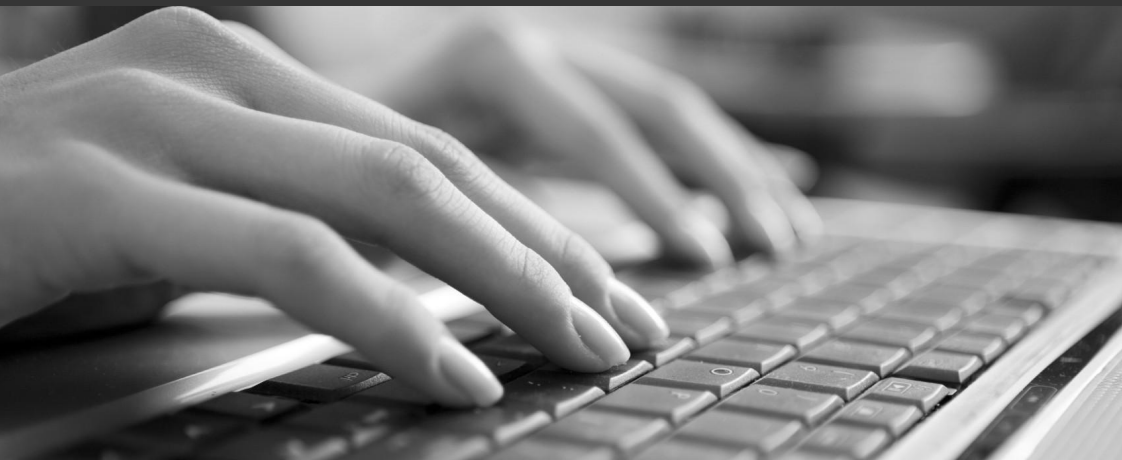
AKA Requirements

- ▶ The final goal is to optimize the **allocation** on daily basis
- ▶ A task has to be assigned to an employee with
- ▶ An employee has a set of **skills** (and affinities)
- ▶ Some tasks has to be performed within a **completion time**
- ▶ Tasks have different durations
- ▶ **A task can be shared** among different employees, each task has a maximum number of employees that can take over it
- ▶ All employees have a **meeting** once a day before store opening: no task can overlap the meeting.



Tip: Look for a similar problems among the examples

Run the examples



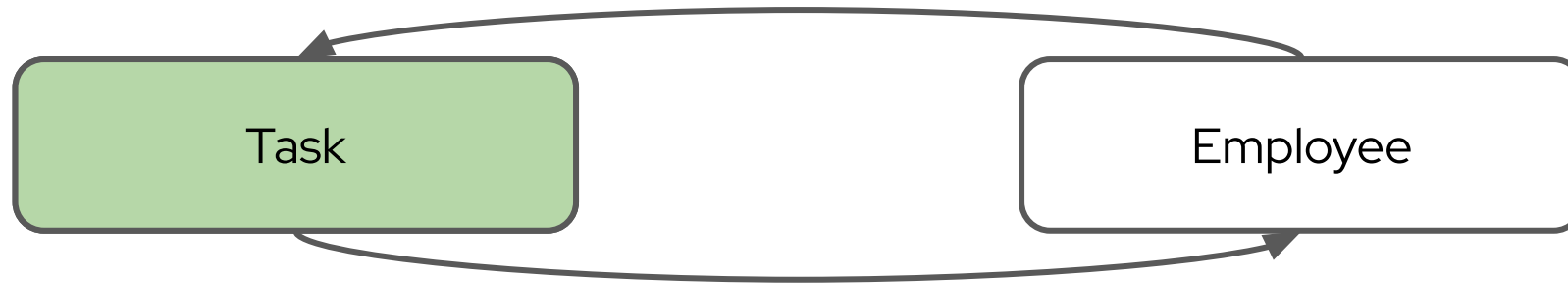
Tasks Domain Model



Which is the Planning Entity?

Which is the limited resource?

The employees! So the **planning entity** is the **task**!

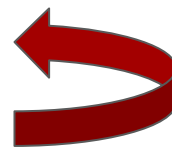


The sequence matters

Task assignment is not an employee rostering problem

Tip: Visualize the problem!

	Day 1	Day 2	Day 3
Shift 1	Lucy	John	
Shift 2	Ann	Lucy	Ann
Shift 3	Ali	Ann	Ali

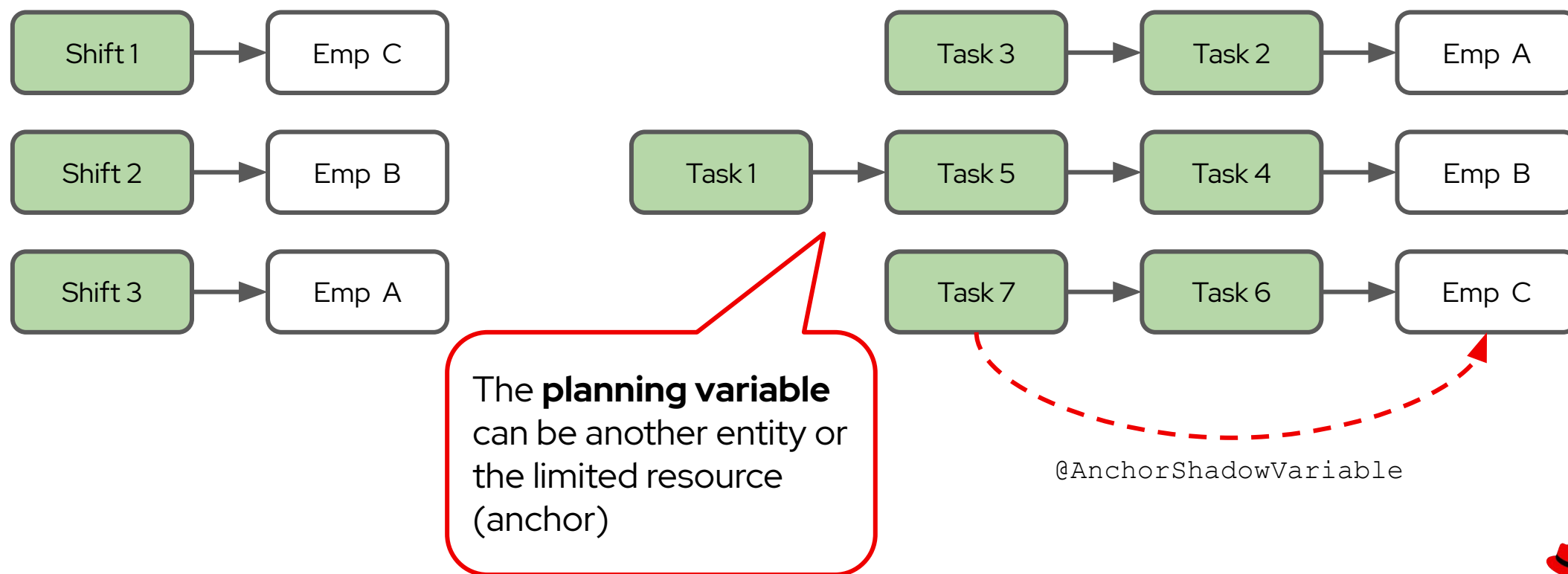


Emp 1
Emp 2
Emp 3

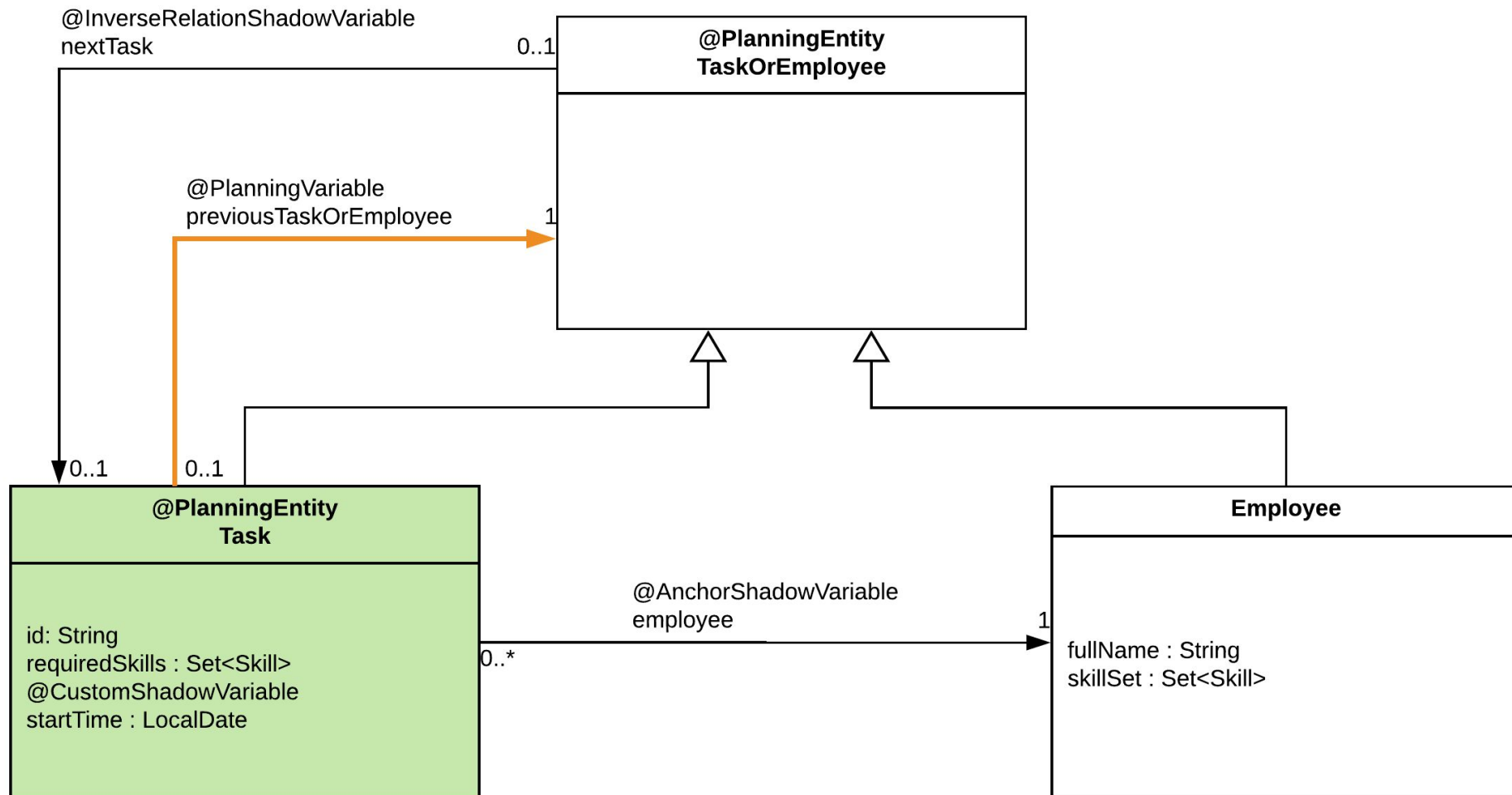
1	2	3
4	5	1
6	7	

Moving a task affects the starting time of following tasks

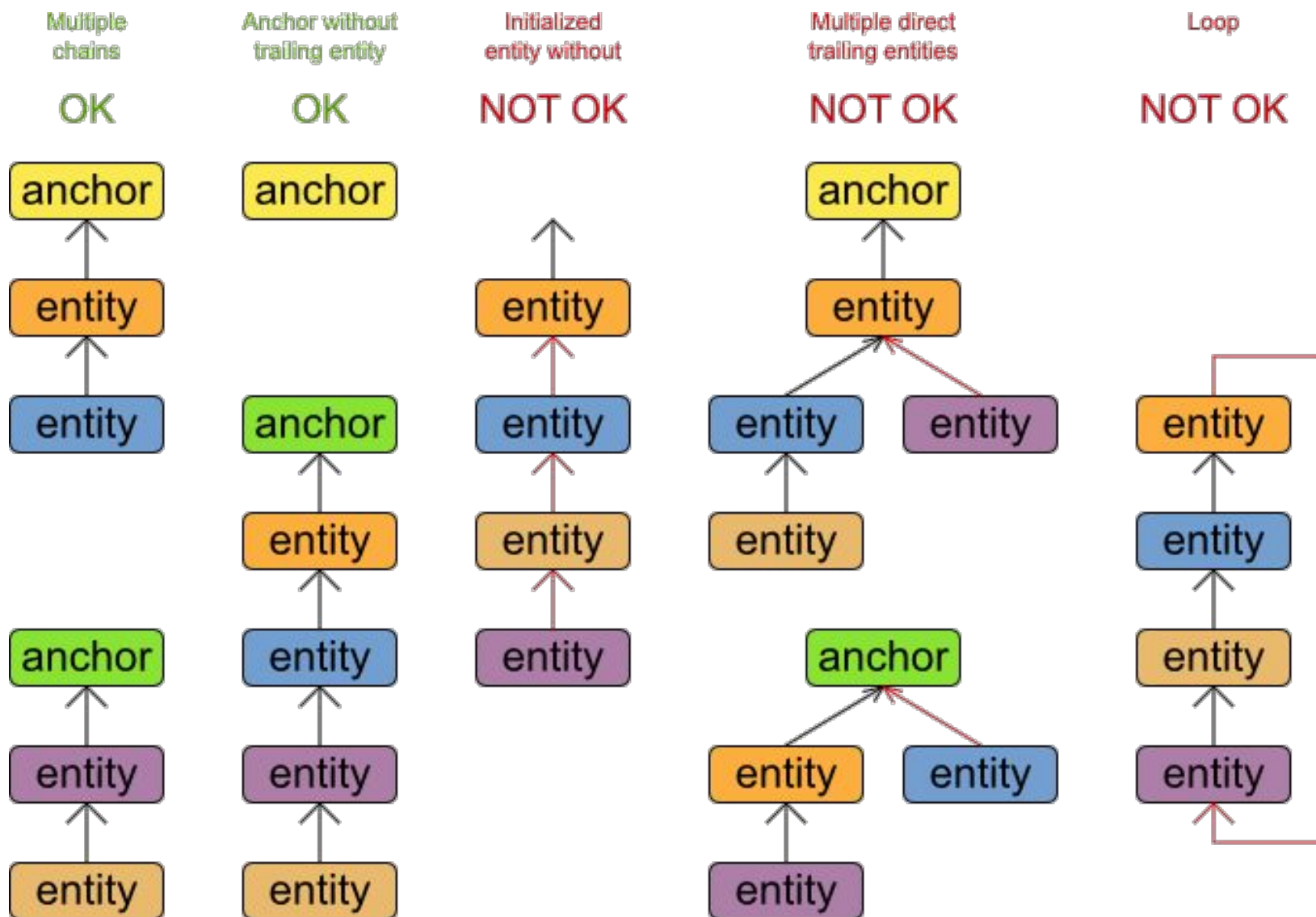
Normal vs Chained



Chained Model UML Diagram



Chain Principles

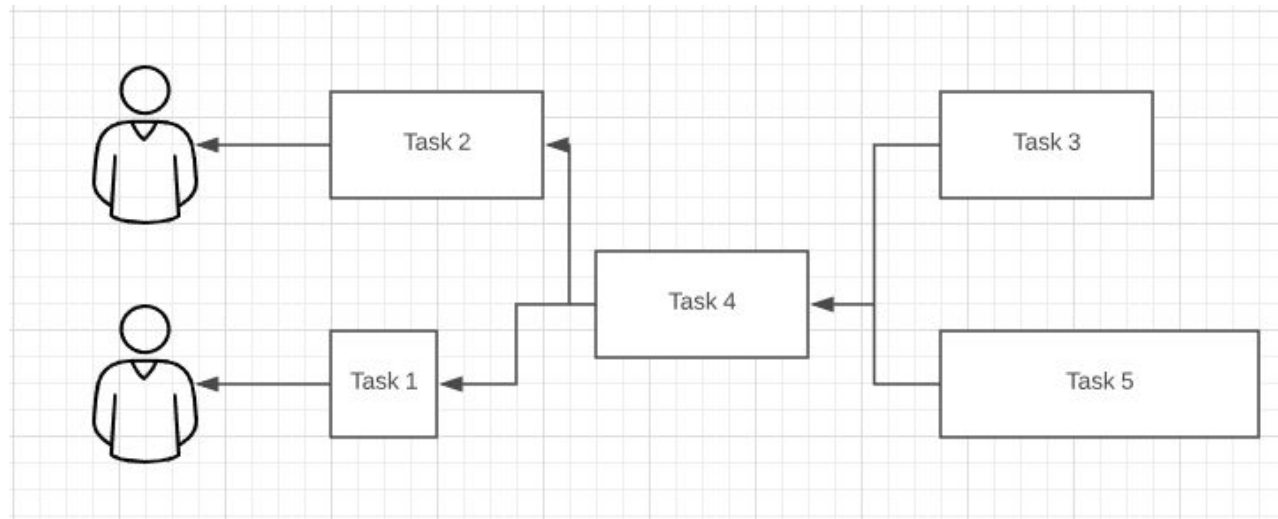


Shared Task

How to split a task among different users

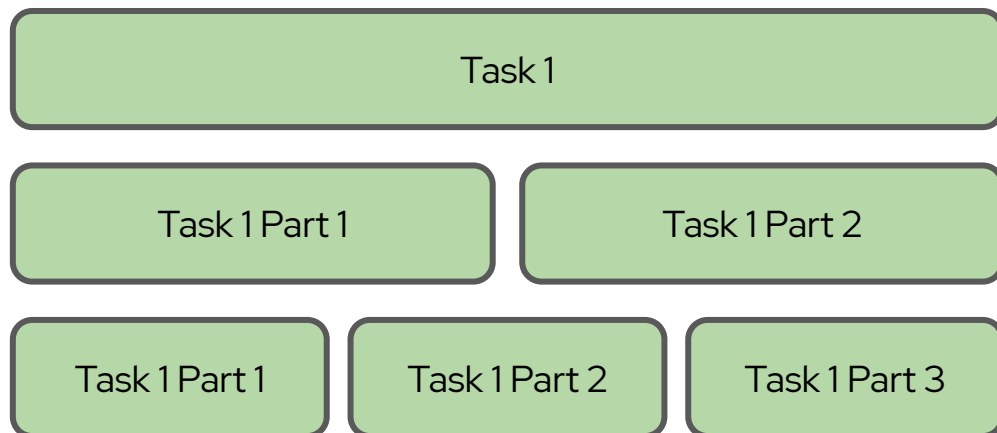
Some task can be shared among the employees!

But this contradict one of the chain principles: *a chain is never a tree, it is always a line. Every anchor or planning entity has at most one trailing planning entity.*

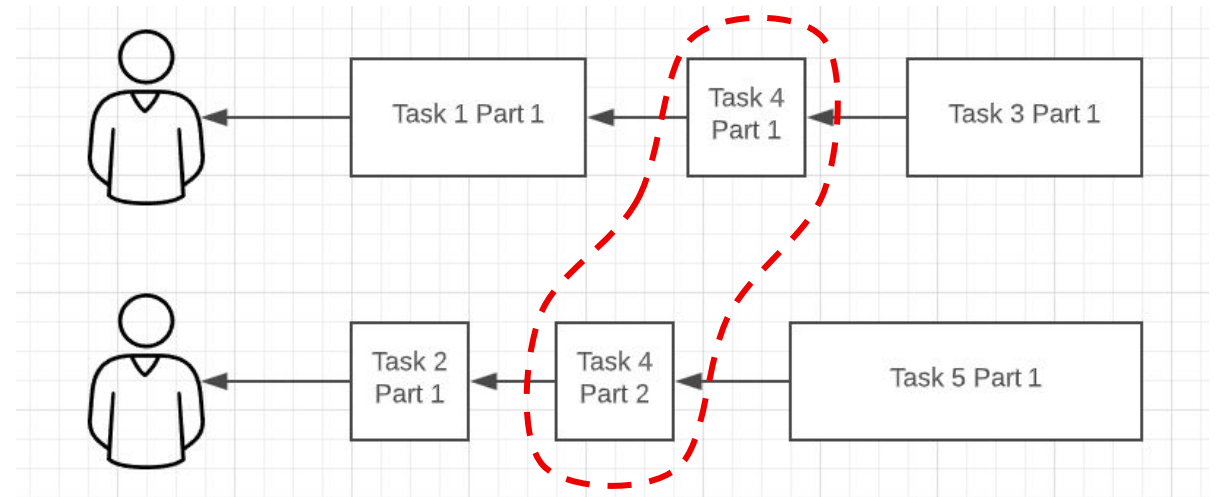


From Shared Task To Splitted Task

How to consider the possible number of splits? A task can be splitted in 2 parts or in 3 parts: the number of splits is potentially a **new planning variable**...



How to keep the part aligned?



A Fair Simplification

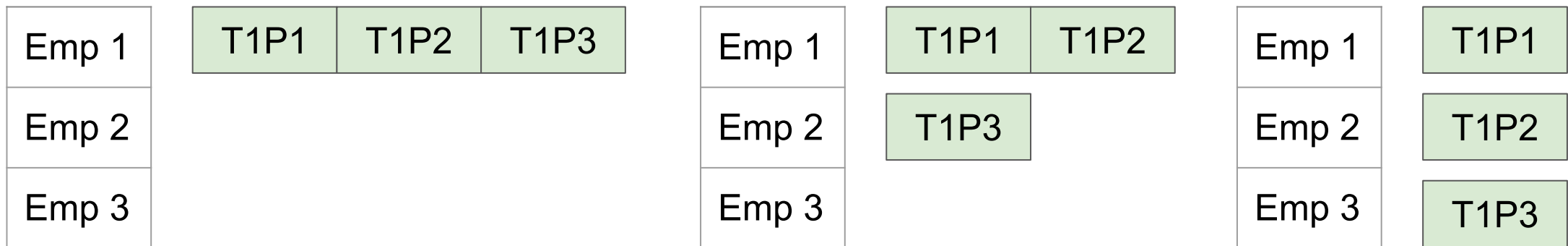
"The best is the enemy of the good"

Task have a fixed number of splits

- ▶ E.g. A task lasts 90 minutes and can be shared by 3 employees

It's modelled as 3 task part of 30 minutes

Allowed task parts topologies:

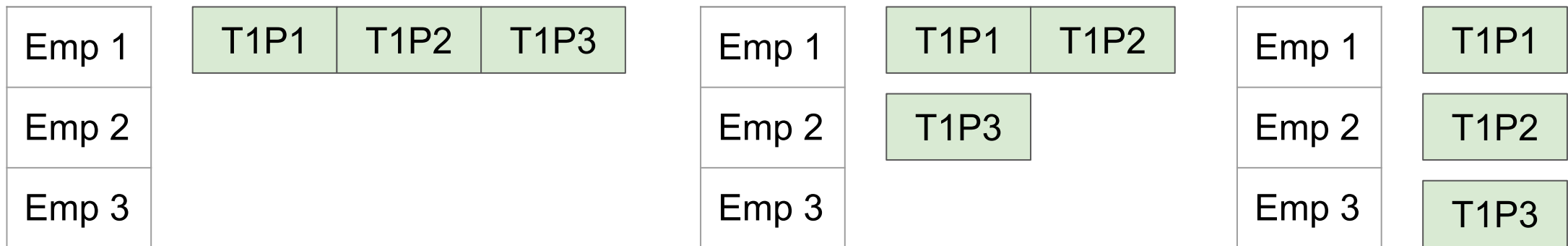


Minor Drawback:
Odd distribution

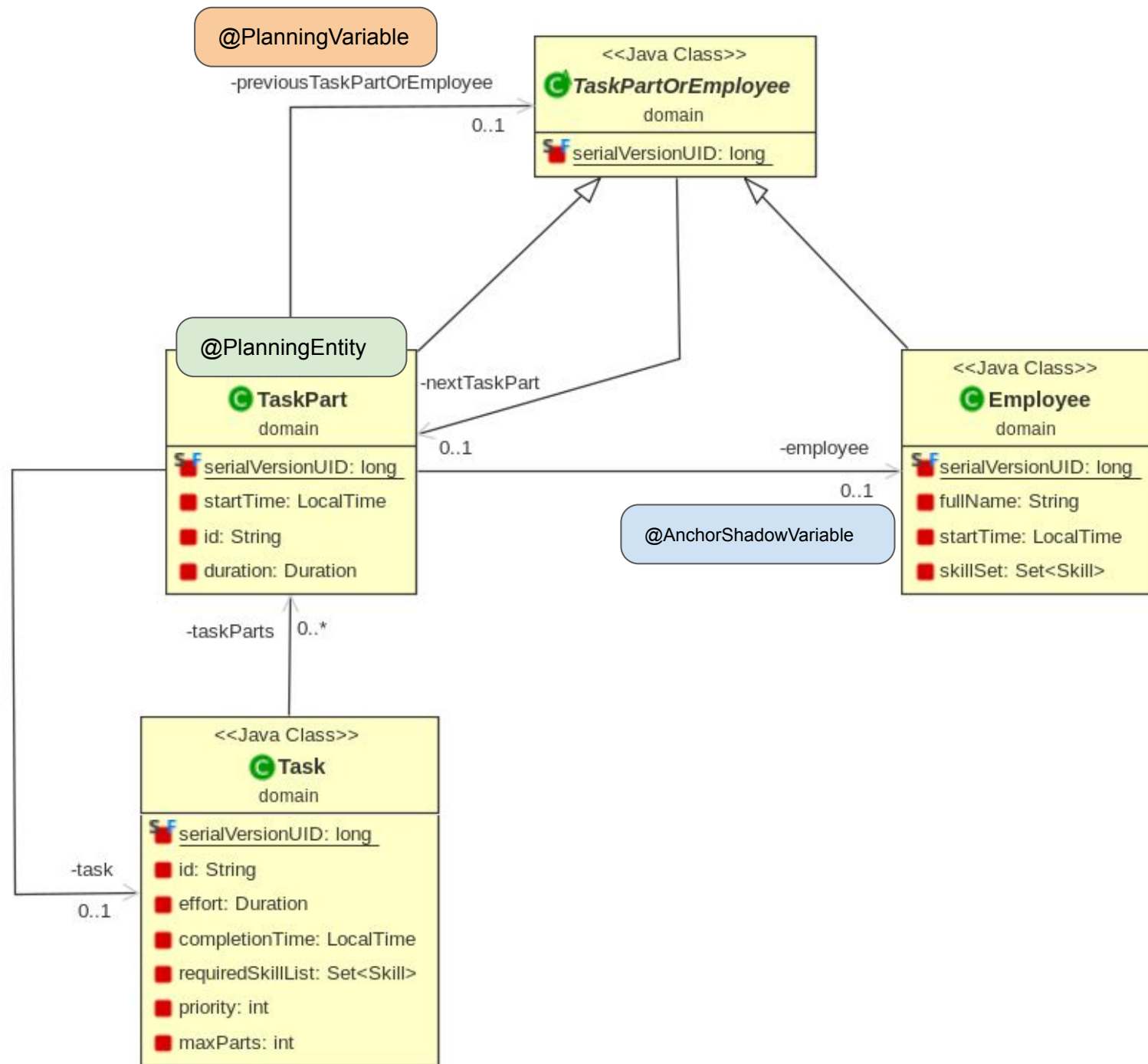
Enforce The Allowed Topologies

Two scoring rules:

- ▶ Same employee - Keep parts together (no task fragmentation)
- ▶ Different employees - Keep parts aligned (same start time)



FINAL MODEL



Task Constraints & Scoring



Scoring

To enforce constraints and to reward better solutions

Hard Constraints:

- ▶ Skill requirements
- ▶ High priority task must be accomplished on time

Soft Constraints

- ▶ Same employee - Keep parts together
- ▶ Same employee - Avoid gaps between parts
- ▶ Different employees - Keep parts aligned
- ▶ Prefer assigning all parts to same employee
- ▶ Minimize the number of employees
- ▶ Priority order

Rule "Same employee - Keep parts together"

When

```
// There is an assigned Task $t with more than 1 part
TaskPart (task.maxParts > 1, $e : employee, $t : task)

// Let $parts be the list of all task parts belonging the same employee
$parts : List( size > 1 ) from collect ( TaskPart ( task == $t, employee == $e ) )

// Does it exists an isolated task part?
exists TaskPart ( employee == $e,
                    task == $t,
                    previousTaskPartOrEmployee not memberOf $parts,
                    nextTaskPart not memberOf $parts )
```

then

```
scoreHolder.addSoftConstraintMatch(kcontext, 0, -1 );
```

TIP: Test with the score verifier

Unit test for the specific rule

```
// Fill in data in the model
(...)
solution.setTaskList(taskList);
// Check expected results
scoreVerifier.assertSoftWeight("Same employee -
Keep parts together", 0, 0, solution);
```

Just Another Requirement

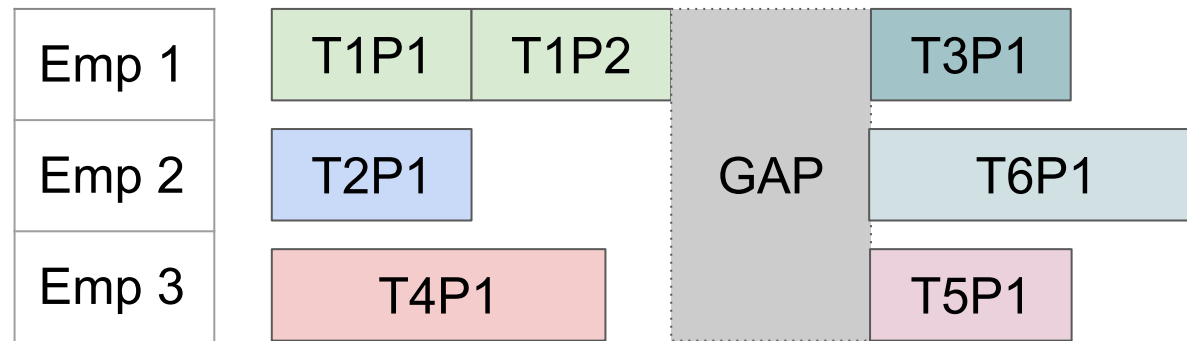
- ▶ All employees have a meeting once a day before shop opening: no task can overlap the meeting
- ▶ A task cannot be interrupted: all the parts must complete before the meeting or all parts must be shifted after the meeting

In general gaps could be enforced by scoring rules

But in chained tasks, the start time is not a planning variable: it is CONSEQUENCE of the task chain!

Mind The Gap

- ▶ All employees have a meeting once a day before shop opening: no task can overlap the meeting
- ▶ A task cannot be interrupted: all the parts must complete before the meeting or all parts must be shifted after the meeting

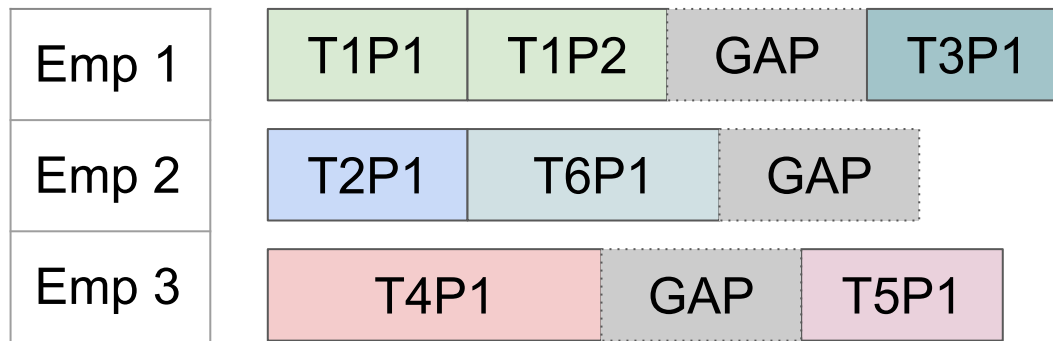


Mind The Gap

The start time are equal to end time of the previous task **by definition**:

- ▶ the start time is not a planning variable: it is CONSEQUENCE of the task chain

First **bad** idea: introducing a special task that acts as a gap



Solution: change the **definition**!

The start time can be calculated taking in account the gap interval

Start Time Calculation

`startTime` is a field of `TaskPart` Class defined as ***custom shadow variable***

```
@CustomShadowVariable(variableListenerClass =  
    StartTimeUpdatingVariableListener.class, sources = {  
    @PlanningVariableReference(variableName = "previousTaskPartOrEmployee") })  
private LocalTime startTime;
```

In `StartTimeUpdatingVariableListener` class:

1. Loop on the chain following `nextTaskPart` relationship
2. `startTime` = end time of the previous task
3. if the task overlap the gap move it at the end of the gap

Further Information

Source code:

<https://github.com/dmarrazzo/task-assignment-optimizer>

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)

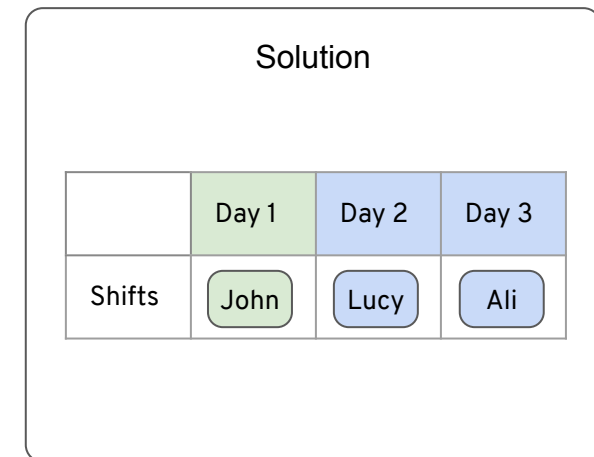
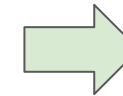
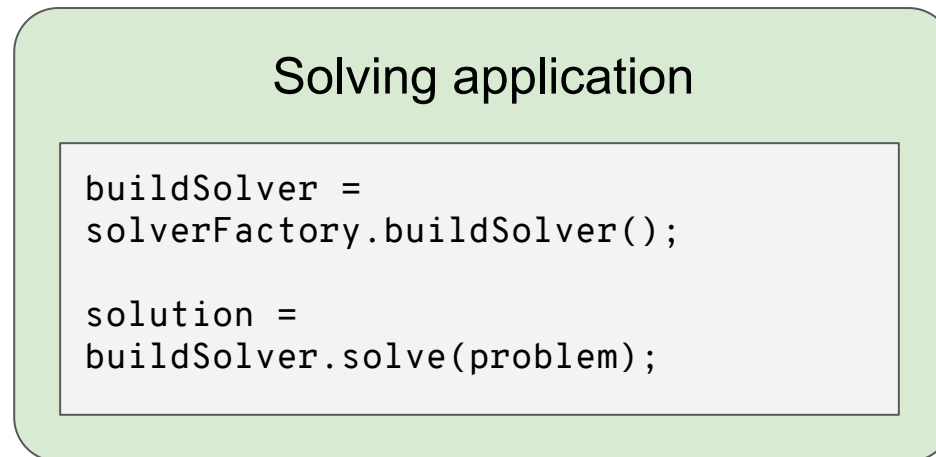
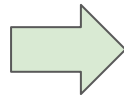
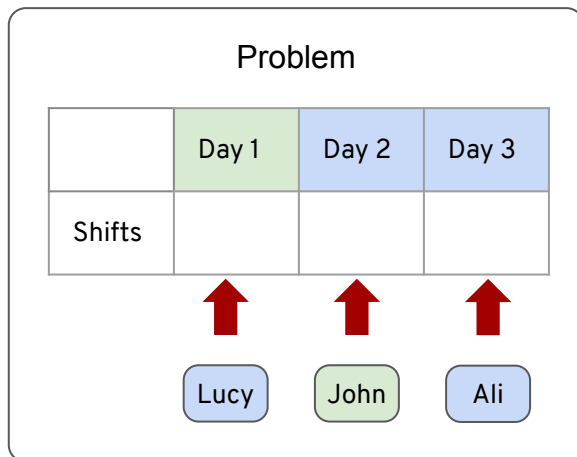
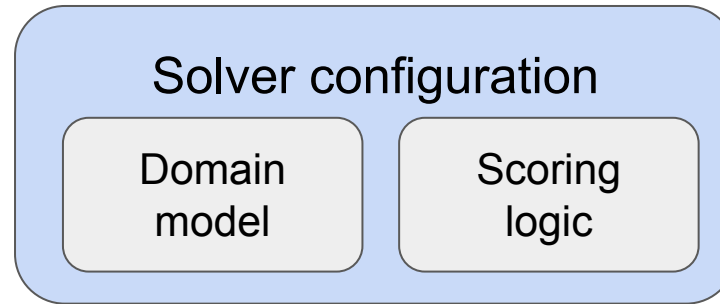


twitter.com/RedHat

How It Works



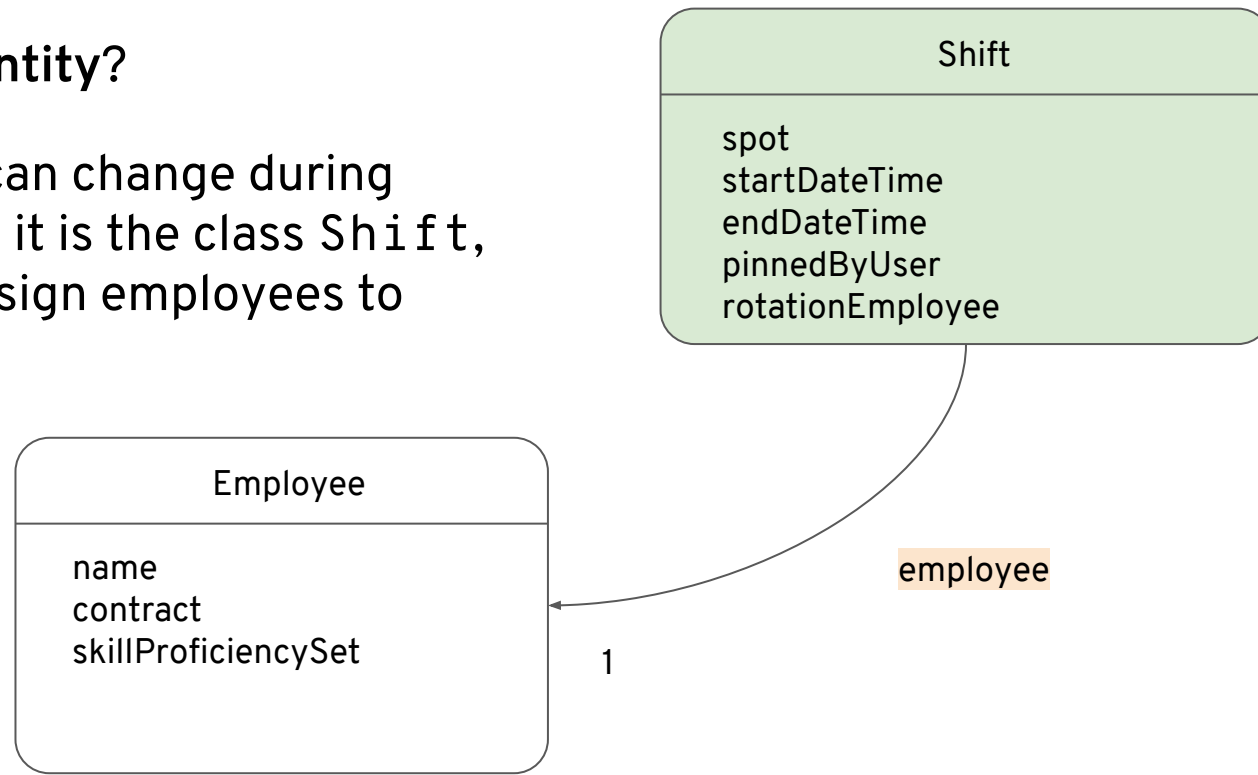
FIND THE SOLUTION



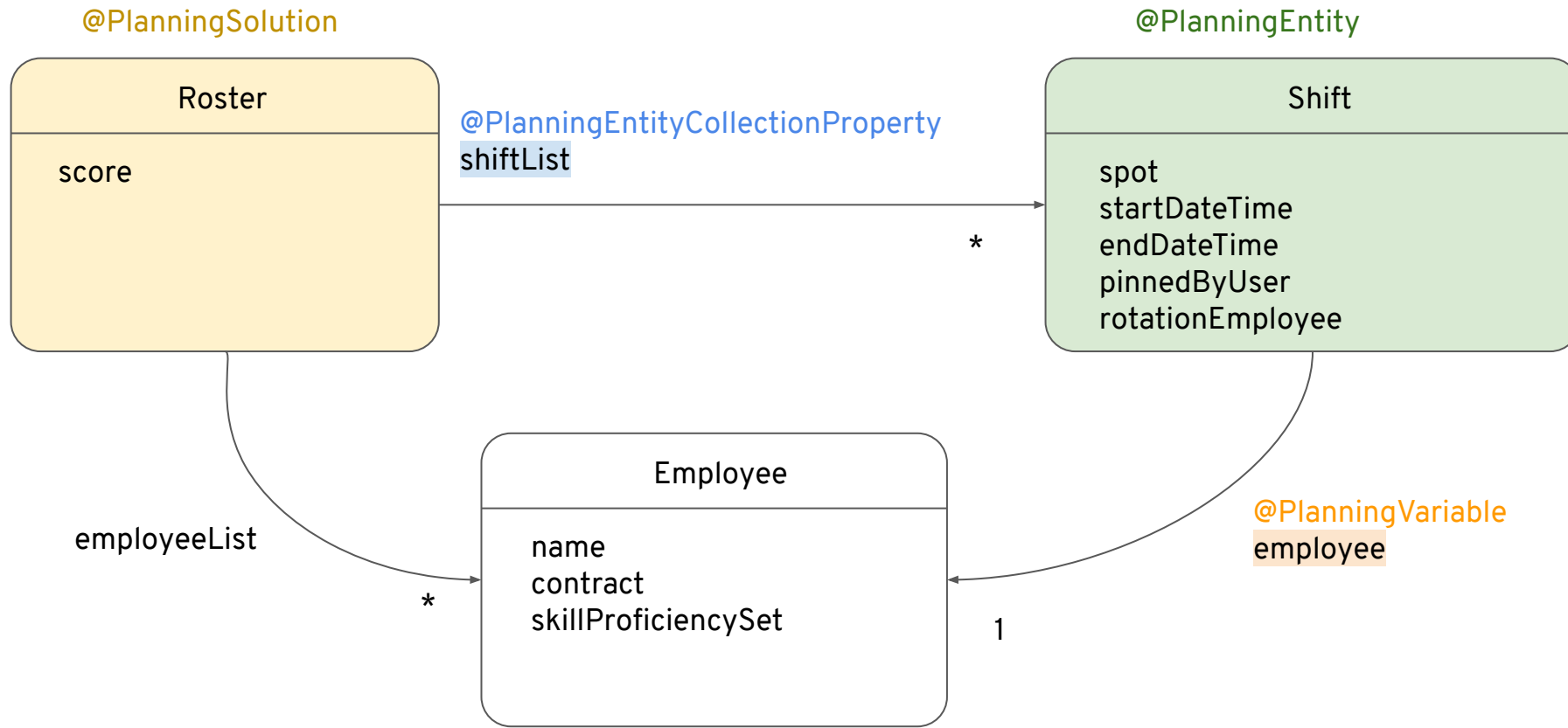
DESIGN THE DOMAIN MODEL

Which is the **planning entity**?

The *class* that Planner can change during solving. In this example, it is the class `Shift`, because Planner can assign employees to shifts.



DESIGN THE DOMAIN MODEL



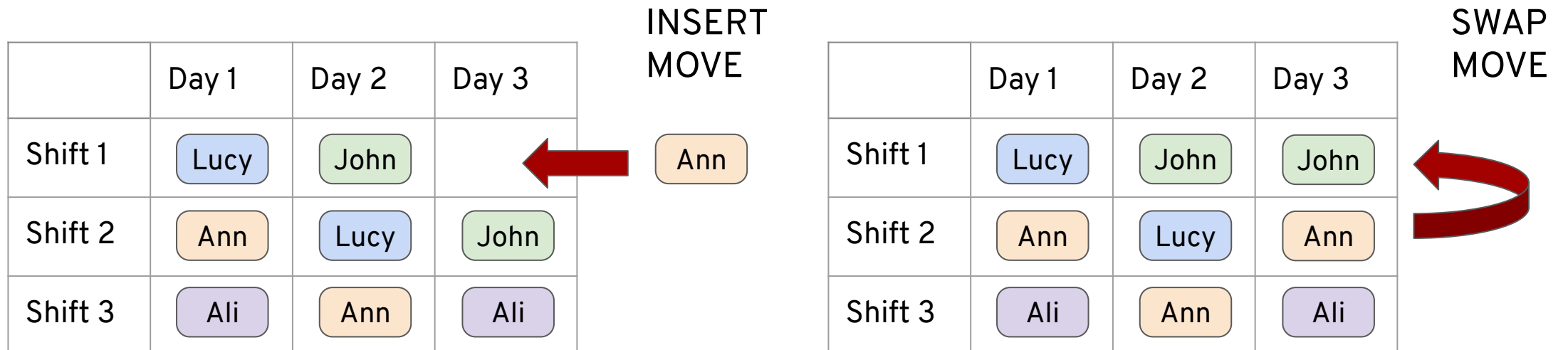
SCORING A SOLUTION

- ▶ The score is an objective way to compare two solutions.
- ▶ The Solver aims to find the Solution with the highest Score of all possible solutions.
- ▶ What are the employee rostering goals?
 - Assign all the shifts an employee with the required skills
 - Satisfy as much as possible employee needs
- ▶ Often a score constraint outranks another score constraint
- ▶ In that case, those score constraints are in different levels (**Hard** constraints and **Soft** Constraints)
 - Skills for a shift are fulfilled, then satisfy the employee day preference.

CALCULATE SCORING

Score logic: when the employee has less than 12 hours rest add constraint match -1

Incremental: means that you avoid re-evaluate all the employee but just calculate the difference

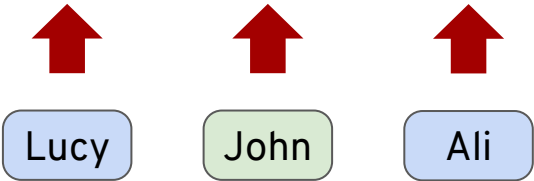


Which solution is the best?

DESIGN YOUR SCORING LOGIC

Problem

	Day 1	Day 2	Day 3
Shifts			



Possible Solutions

	Day 1	Day 2	Day 3
Shifts	Lucy	John	Ali

	Day 1	Day 2	Day 3
Shifts	John	Ali	Lucy

	Day 1	Day 2	Day 3
Shifts	John	Lucy	Ali

Scoring

Hard score: -2
Soft score: 0

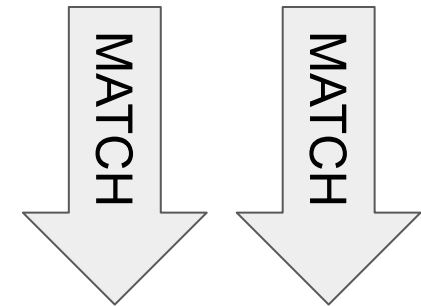
Hard score: 0
Soft score: -3

Hard score: 0
Soft score: -2

CALCULATE COMPUTER COST

```
rule "Required skill for a shift"
when
    // There is a shift whose skills are not fulfilled
    Shift(
        employee != null,
        !employee.hasSkills( spot.getRequiredSkillSet() )
    )
then
    scoreHolder.addHardConstraintMatch(kcontext, -100);
end
```

	Day 1	Day 2	Day 3
Shifts	Lucy	John	Ali



-200