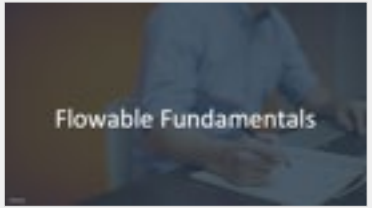

Flowable
making process personal

Flowable Trainings

Modelling Standards with Flowable

Agenda



Introduction

Your Trainer:

Andreas Isler
Business Process Modeler and Requirements Engineer

Contact us:
training@flowable.com
https://forum.flowable.com/

Organization

First things first!

Social

- First name basis

Breaks

- Regular breaks

Questions and Exercises

- **Don’t mind to interrupt for questions –**
  but maybe write a short message in the chat!
- We will have a number of exercises you can finish in your own time.
  We will also give you time to explore.
- Longer / specific discussions during the break or in workshop

Objectives

What you will know…

- **Understand the concepts**
  - Learn the most important Flowable concepts
  - Understand their role within your Flowable Apps
- **Learn how to model**
  - Processes with BPMN
  - Cases with CMMN
  - Decision Models with DMN
- **Designing Forms**
  - Know how to create simple forms with Flowable Forms
- **Running your Apps**
  - Create and deploy your first few apps and execute them

Background and Expectations

- I am…
- I attend this course because…
- I want to learn…

The Flowable Development Cycle

Developing a Flowable App

- Although not required, Flowable lends itself well to an **agile development approach**.
- You can ship new versions of code and models **at a high pace** to gain feedback.
- Finding the right process for your organization and project needs **careful thought**.
- Don’t forget that every Flowable project is also a **software project**!

Roles in Flowable Projects

In every Flowable project, you will find similar roles.

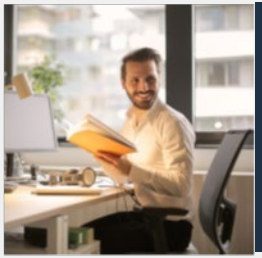
Often, these roles are fluid though!

Developer

Project Manager

Scrum Master

Architect



BA

Modeler

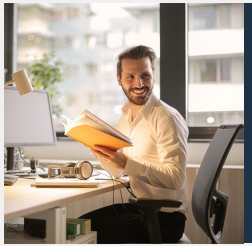
Sysadmin

Tester

Modeling – different approaches

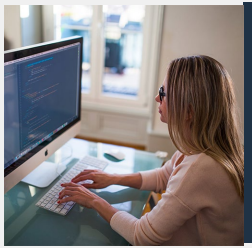
In many projects, certain roles take over different responsibilities.

As always, the roles are not set in stone!



Modeler

- Responsible for **CMMN** and **high-level models**.
- Models business-relevant **BPMN processes** as well.
- Creates **Forms** and **Pages** with great user experience.
- Takes care of **permissions** and **process/case flow**.



Developer

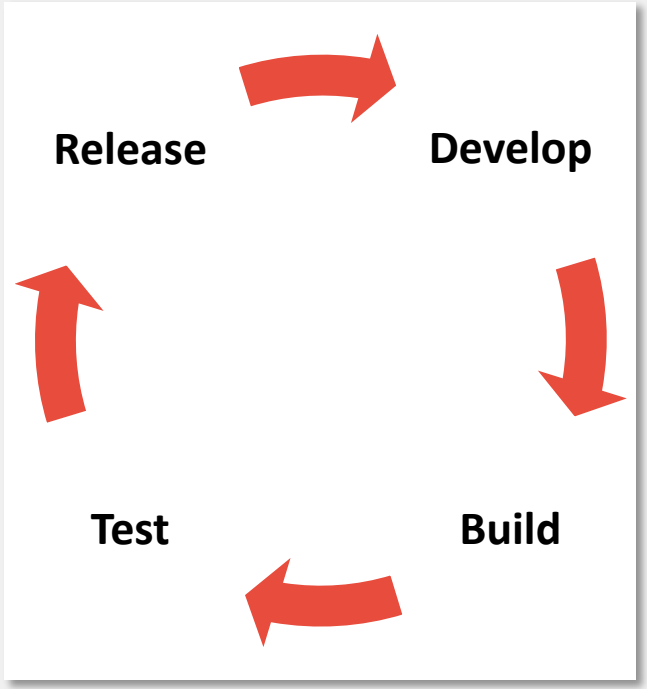
- Focus on technical tasks, e.g. implementation and modeling **Service Tasks** backed by Java Logic.
- Creates more **technical models**, e.g. User Definitions, Data Object Models etc.
- Provides and implements **REST endpoints** in Forms.

And what about me, **the user**?

Of course, the user should be at the center of your attention!

The Flowable Development Cycle

- Inspired by **agile methods**
- Works well in **DevOps** environments
- Consists of **Release**, **Develop**, **Test** and **Build** Phases
- The cycle recognizes that a Flowable Project is also always a **software project**

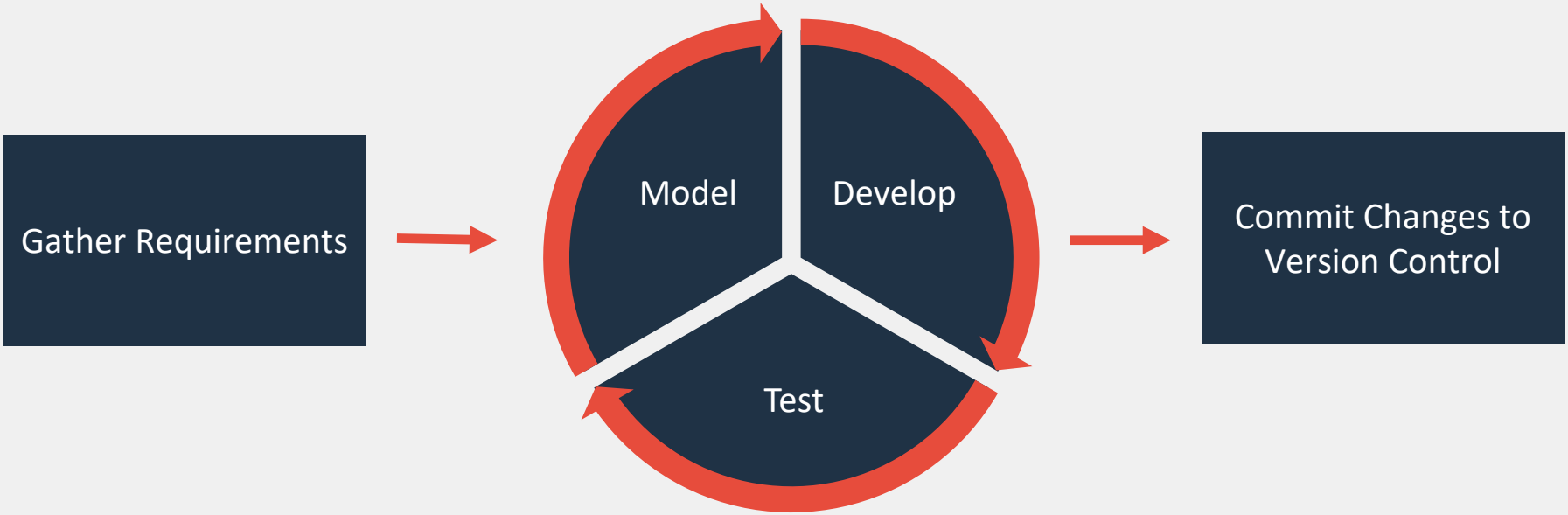


Development - Tasks

The development phase is where the **development** takes place.

- Architecture and setup
  Architects, Developers and Sysadmins
- Requirements gathering
  Modelers / BAs
- Process and Case Modeling
  Modelers / BAs
- Development of integrations/service/APIs
  Developers
- Intermediate testing
  Developers, Modelers, Testers

Development Example Workflow

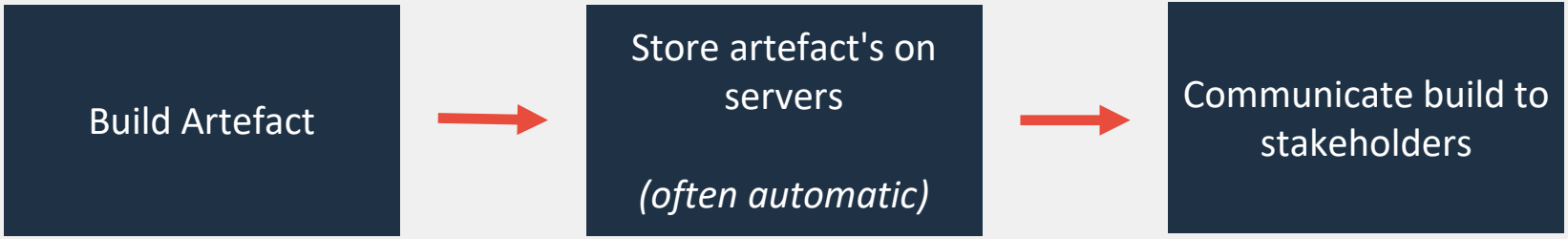


Build - Tasks

During the deployment phase, the application will be rolled out.

- Setup of Continuous Integration / Build
  Architects
- Start of Build
  Sysadmins and developers
- Environment-specific adaptations
  Architects and Developers, implemented by Sysadmins
- Communication to Stakeholders
  Modelers / BAs and Developers

Build
Example Workflow

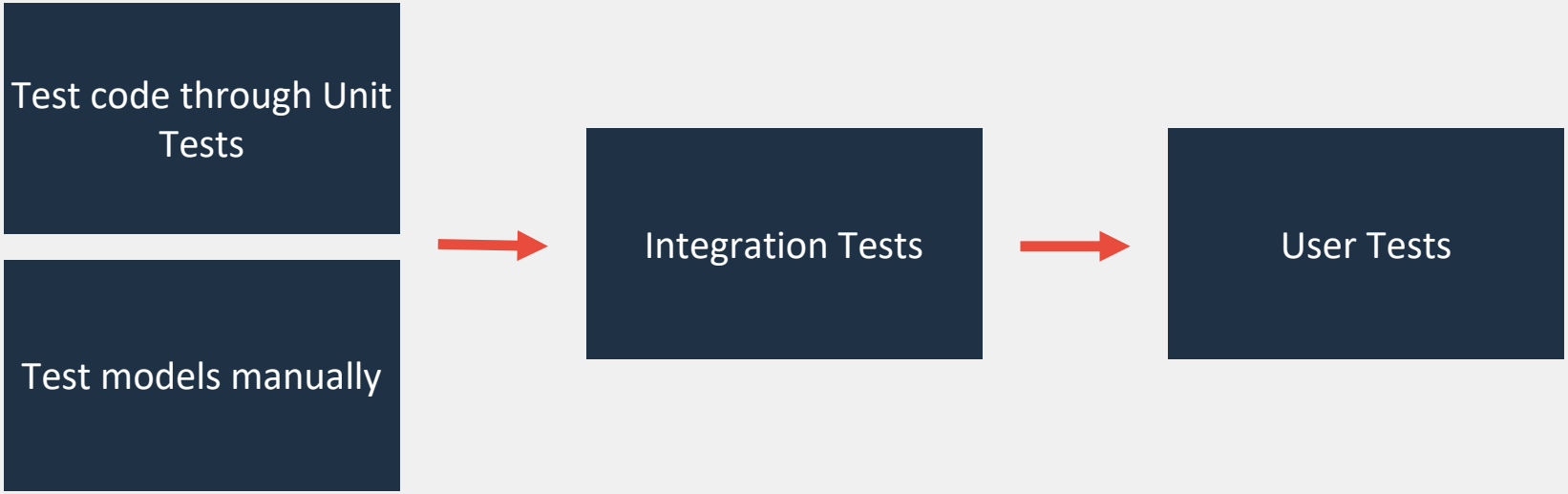


Test - Tasks

Testing is performed during development as well as upon release

- Creation of Unit Tests
  Developers
- Creation of (automated) Integration Tests
  Testers and Developers
- Manual tests of models
  Modelers
- User Tests
  End Users

Test
Example Workflow

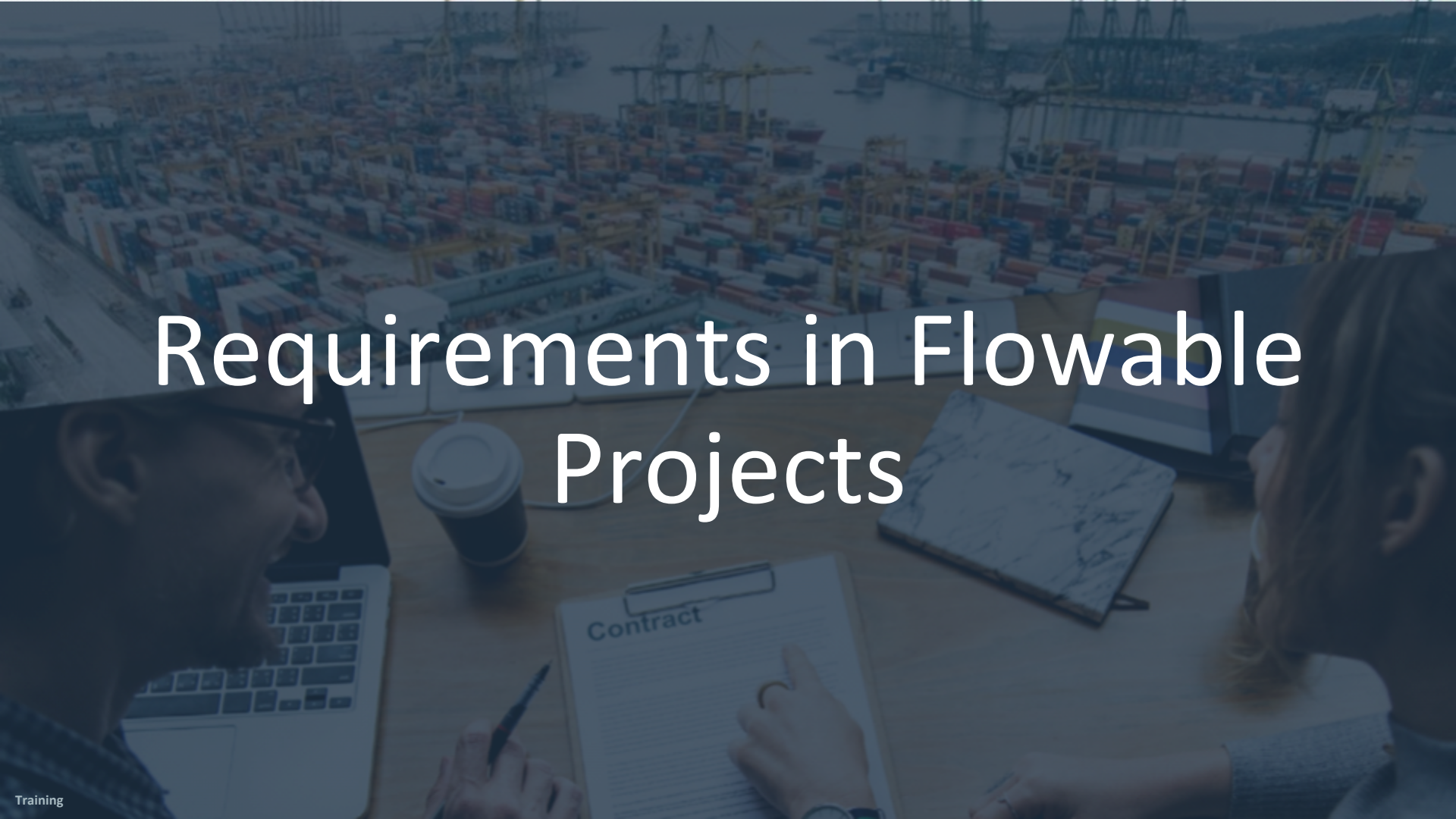


Release - Tasks

Once a new iteration is done, it can be released.

- Build of final artefacts
  Developers and Sysadmins
- Deployment on customer servers or cloud
  Sysadmins
- User training
  Modelers / BAs

Release
Example Workflow



Requirements in Flowable Projects

Interviews

+ Very powerful source of requirements

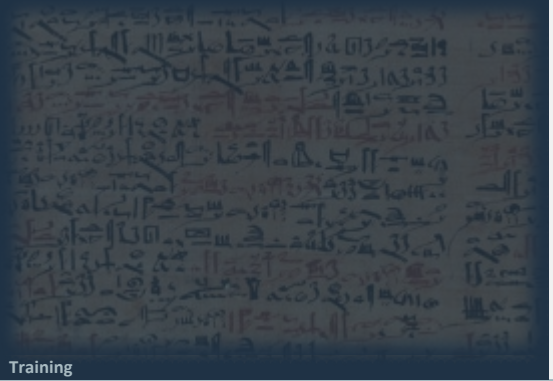
- Depends on skills of interviewer and interviewee

- Asking the right questions is hard

- To get a first feel for a process, a lot of times, **simply talking** to a user or a BA will help a lot.
- A lot of times, people will simply **tell** you what a process is about.
- It’s simple: **Take down notes** and try to get the most out of these sessions.
- Try to perform **structured** and **unstructured** interviews.
- **Don’t stop** – keep interviewing even in later project phases.

Textual Descriptions

+ No tools required
+ No limitations
+ Easy to iterate on
- A lot of interpretation
- No clear flows



- Write down requirements in a **structured** or **unstructured** way.
- Store requirements in a **central location** (e.g. a wiki such as Confluence).
- Make sure that you use **clear terminology**, establish a glossary to define important terms early on.
- **Split up** your requirements in digestible bits and keep them up to date.

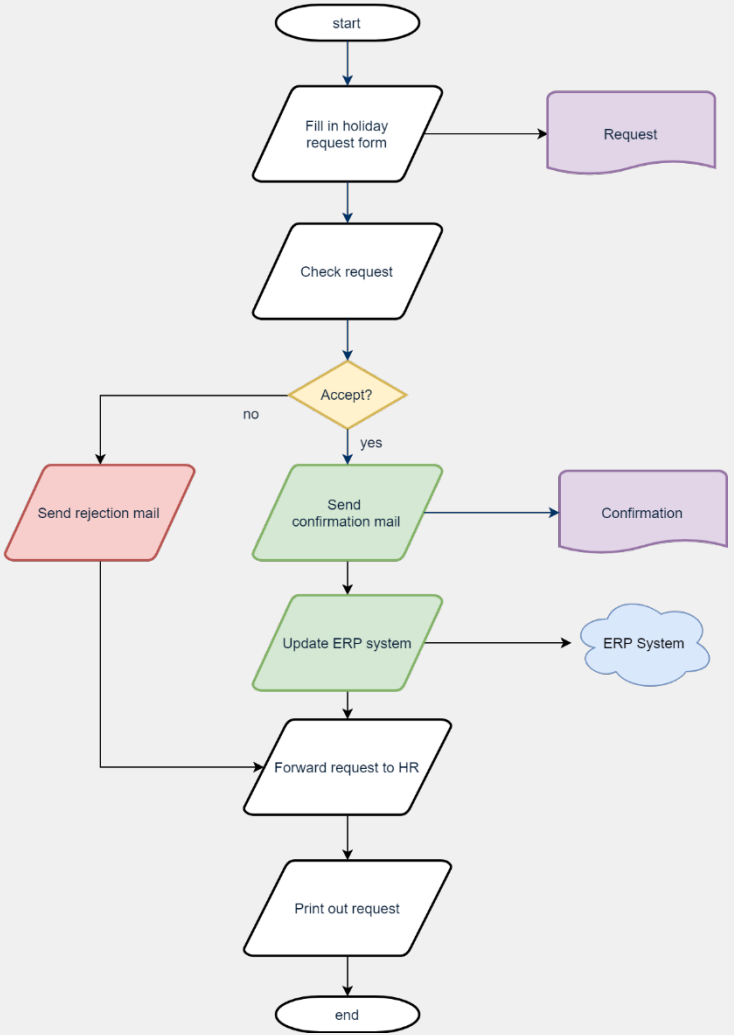
User Stories

+ No tools required
+ Detailed description
+ Benefit per user role
- Time consuming
- No clear flows

- As an employee I want to fill in a form for my holiday request, so that the supervisor can check it.
- Acceptance criteria:
  - The employee must fill out the following fields:
    - Start date
    - End date
    - Type (regular, unpaid, military)
    - Comment
  - The employee can send the request to the supervisor by clicking the “Send to supervisor” button.
  - The button is only active once all fields are correctly filled out.

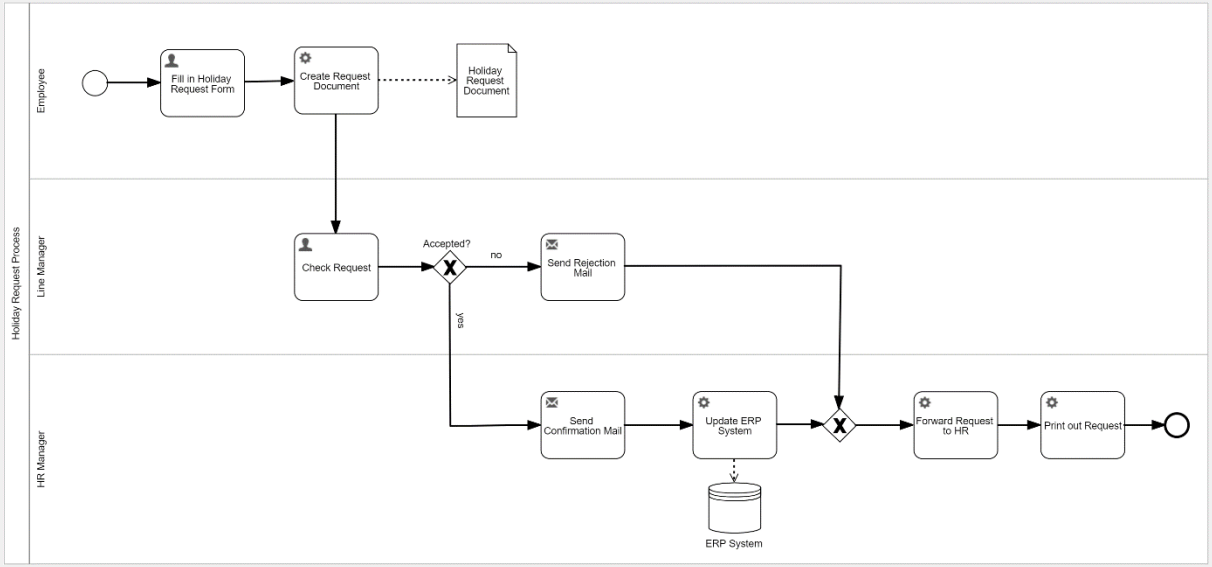
Flowcharts

+ Easy to understand
+ Clear Flow
+ Tool: Whiteboards
- No responsibilities
- Unclear semantics



Representation: BPMN

+ Clear semantics
+ Visible responsibilities (BPMN)
+ Clear structure
+ Executable
- Requires tools
- Training necessary



<u>Out of personal experience</u>:

A bad BPMN diagram is worse than none at all!

Process Landscapes

Often, it is helpful to get an **overview** of the company by creating or consulting a **process landscape** first:

Requirements for Business Rules

«If the client is less than 18 years old, he or she can apply for a Junior Prepaid Card. If the client is older and has a monthly income of more than CHF 50’000, he or she can apply for a Silver Credit Card. If the monthly income is more than CHF 80’000 and the client has a good credit rating, it is possible to apply for a Gold Credit Card. Finally, if the credit rating of the client is bad, the application will be rejected»

* Credit Card Ap···

U	Age age *number*		Credit Rating credit_rating *number*		Income income *number*		Product product *string*
1	<	18	==	-	==	-	Junior Prepaid Card
2	>=	18	>	5	>=	50000	Silver
3	>=	18	==	8	>=	80000	Silver,Gold
4	==	-	==	-	==	-	Not Eligible
5	==	-	==	-	==	-	

Requirement Exercise

1. Read the requirement
2. Find a better way to represent it

10 minutes

- First, the employee fills in a form and indicates the length of the holiday. The form is then sent to the supervisor.
- The supervisor checks the holiday request and either accepts or rejects the request.
- If the request is accepted, it will be confirmed via email on behalf of HR and a new holiday entry will be created in the ERP system.
- The request is forwarded to an HR Manager and at the same time, HR prints out the document.
- If the request is rejected, the employee will be informed via email of that decision and no further steps will be taken.

Flowable Fundamentals

Basic Concepts

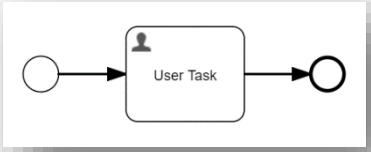
To get the most out of Flowable, you need to understand

- Apps, Deployments, Definitions and Models
- Processes
- Cases
- Permissions
- Hierarchies
- Forms

Models, Definitions, Instances

Model are the blueprint or “source code” of what you want to create:

- BPMN model in XML format
- Form model in JSON format

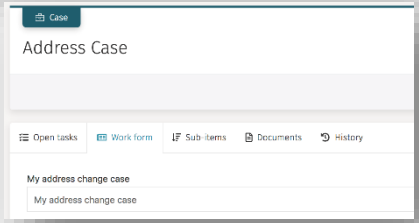


Definitions are what is created when you publish a model to an engine

- Process Definition for an Expense Process
- Case Definitions for a Repair Case

Instances can be created based on definitions after that.

- Concrete Process Instances for expense reports
- Concrete Case Instances for Repair Cases



A real-life analogy

Model
Blueprint

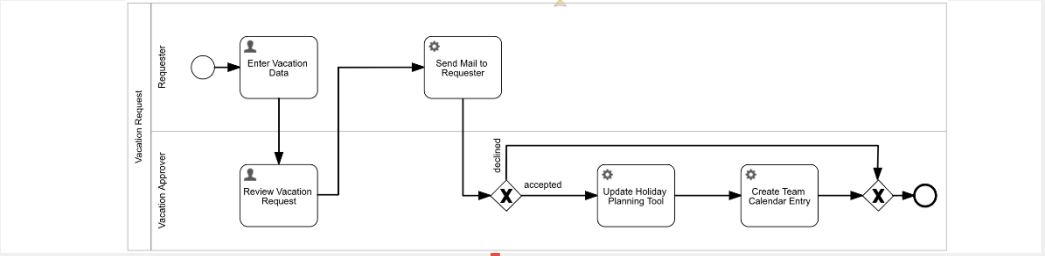
Definition
Approved Blueprint

Instance
Constructed House

Example: Process Model to Instance

1. Create a model
2. Publish/deploy it
3. Start creating instances

Process Model *(Flowable Design)*



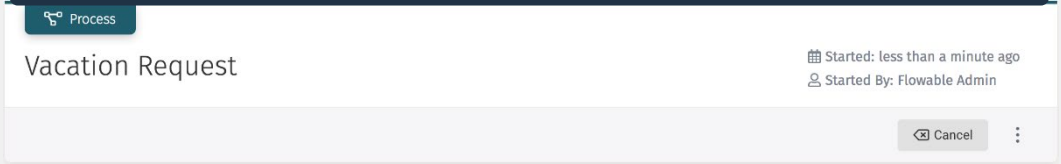
Publish

Process Definition *(Flowable Application Repository)*


```
{
  id: "PRC-VacationRequest:1:07e9f043-2ac0-11e9-9ae6-0242ac120021",
  url: "http://dev-work.flowable.io/flowable-work/process-api/repository/process-definitions/PRC-VacationRequest:1:07e9f043-2ac0-11e9-9ae6-0242ac120021",
  key: "VacationRequest",
  version: 1,
  name: "Vacation Request",
  description: null,
  tenantId: "",
  deploymentId: "PRC-07d297af-2ac0-11e9-9ae6-0242ac120021",
  deploymentUrl: "http://dev-work.flowable.io/flowable-work/process-api/repository/deployments/PRC-07d297af-2ac0-11e9-9ae6-0242ac120021",
  resource: "http://dev-work.flowable.io/flowable-work/process-api/repository/deployments/PRC-07d297af-2ac0-11e9-9ae6-0242ac120021/resources/VacationRequest.bpmn",
  diagramResource: null,
```

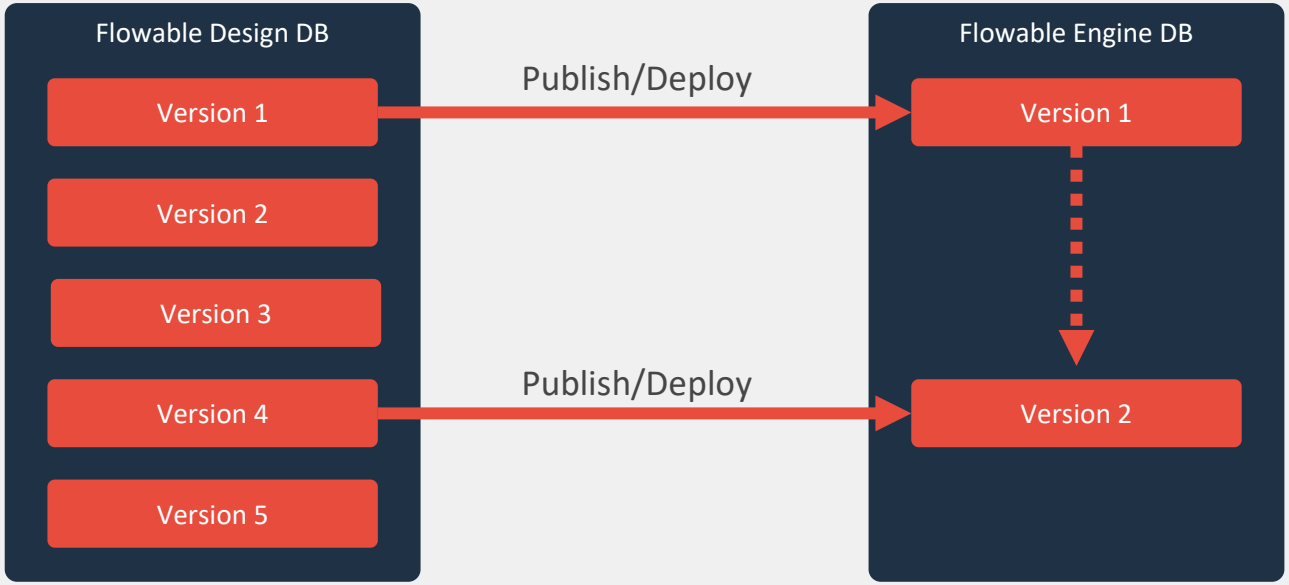

Create

Process Instance *(Flowable Application Runtime)*



Versions: Definitions vs. Models

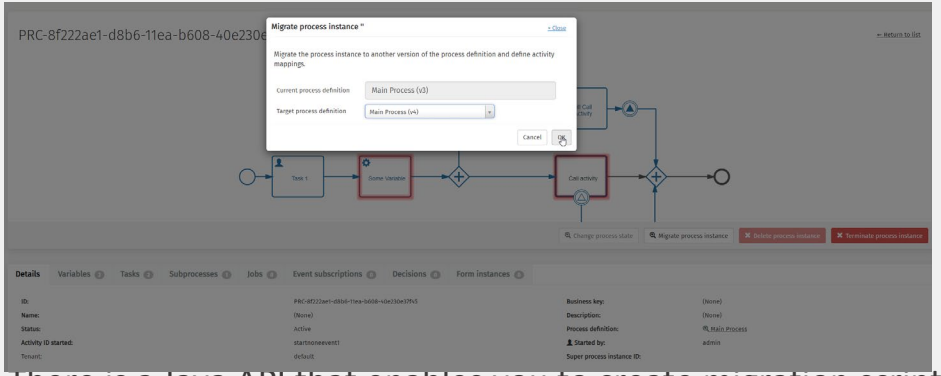
Models and definitions are not in sync!
Every time an app is published, all definitions are updated.



Migration

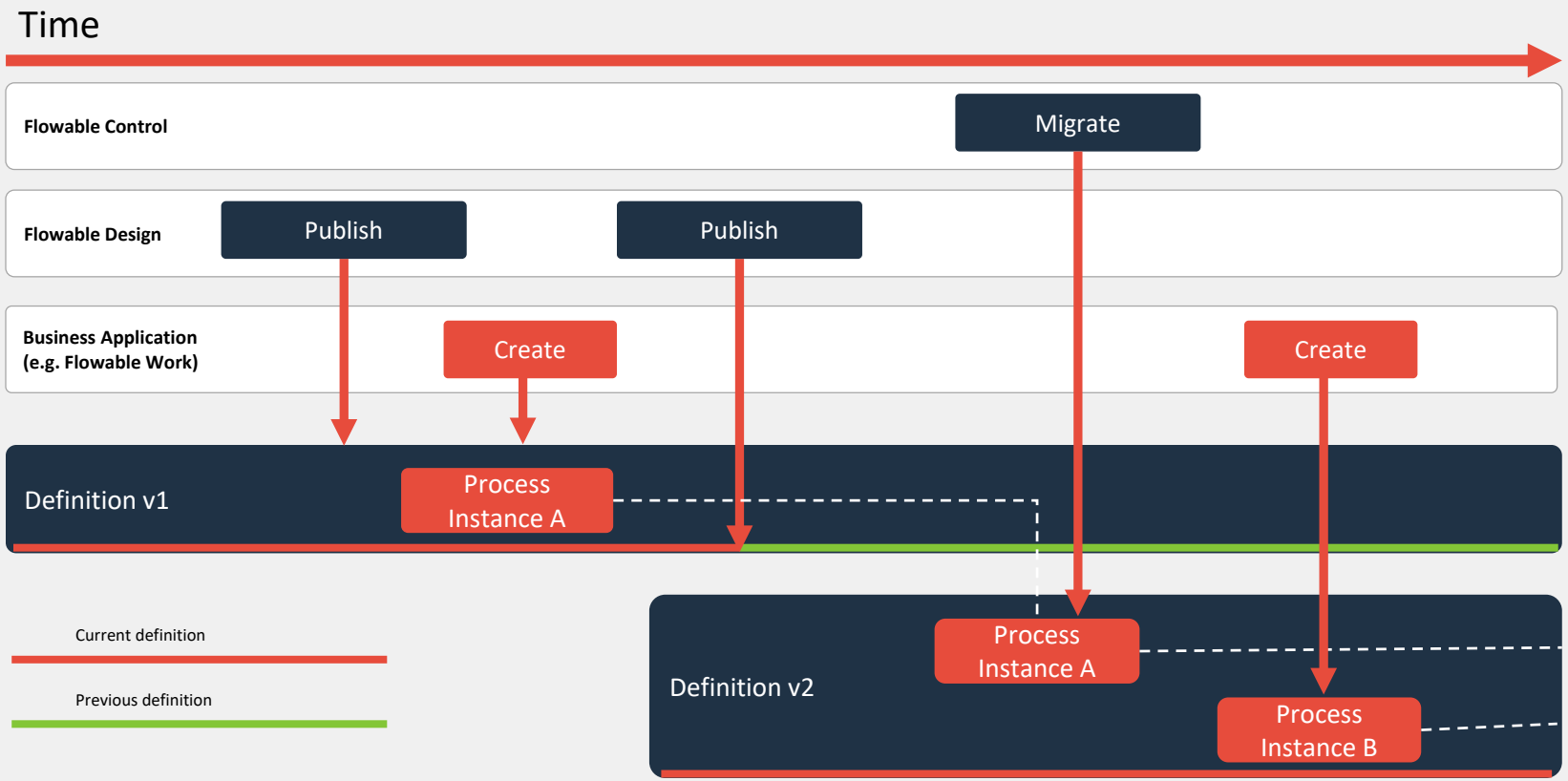
Bring your models up to speed!

- It is possible to update the version of your **process** and **case definitions**.
- Individual instances can be migrated via Flowable Control:



- There is a Java API that enables you to create migration scripts with complex mappings, variable changes etc.
- Large-scale migrations are supported via the “Batch Process Migration”.

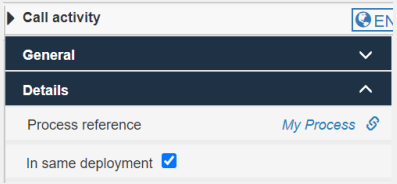
Definition Versions



Same Deployment

Define the definition loading process.

- When referencing a model, e.g. a process or form, you can define which definition will be used with the «**Same Deployment**» flag.



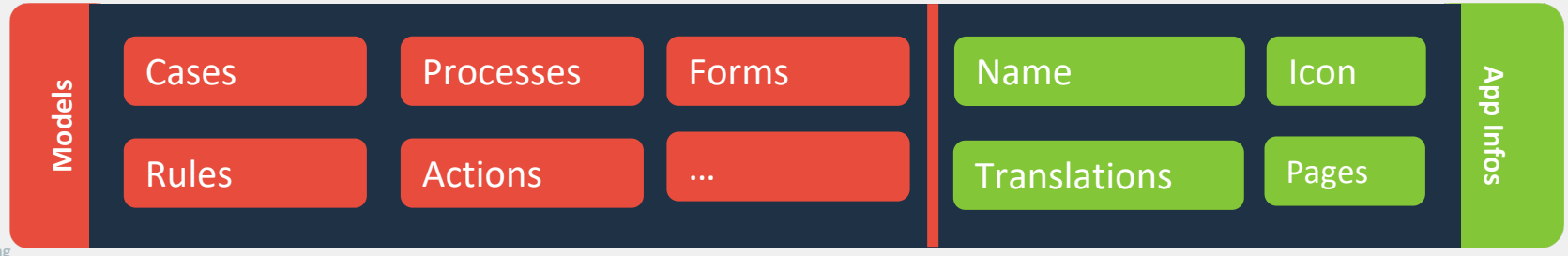
- If the flag is **ticked**:
  The definition that was created **along with the referencing model** will be used.

- If the flag is **not ticked**:
  The **newest** definition was created **along with the referencing model** will be used.

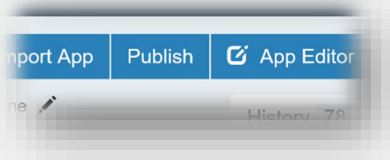
- The default behavior is «in same deployment» which guarantees **consistency**.
  However, newer versions will not be considered.

Apps are containers

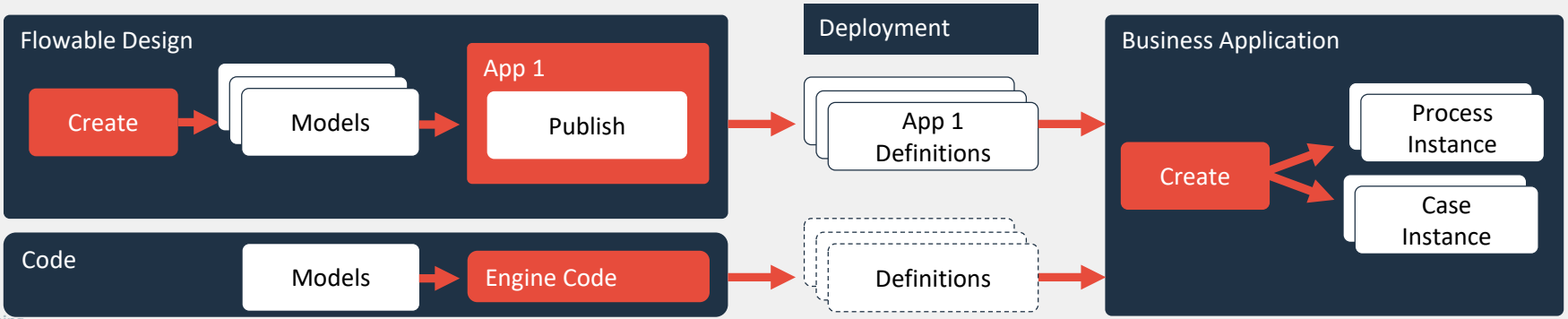
- Apps are versioned **containers** created in **Flowable Design** which must be **published** to a Flowable Application.
- They contain **models**, most notably of **Cases**, **Processes**, **Forms** and **Rules**.
- Flowable Platform also offers **Actions**, **Content** and **Page Models etc**.
- They do **not** contain Users, Groups, case instances or process instances!



Deployment: Export Apps

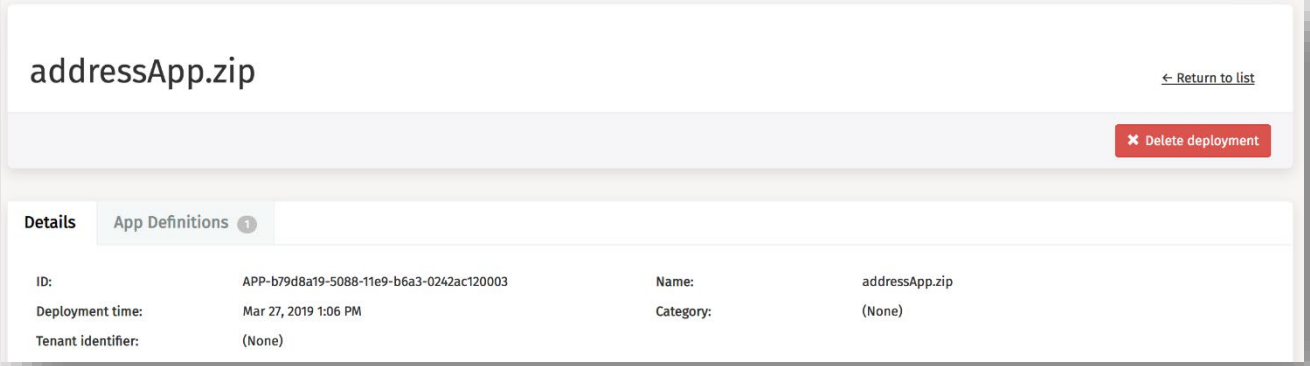


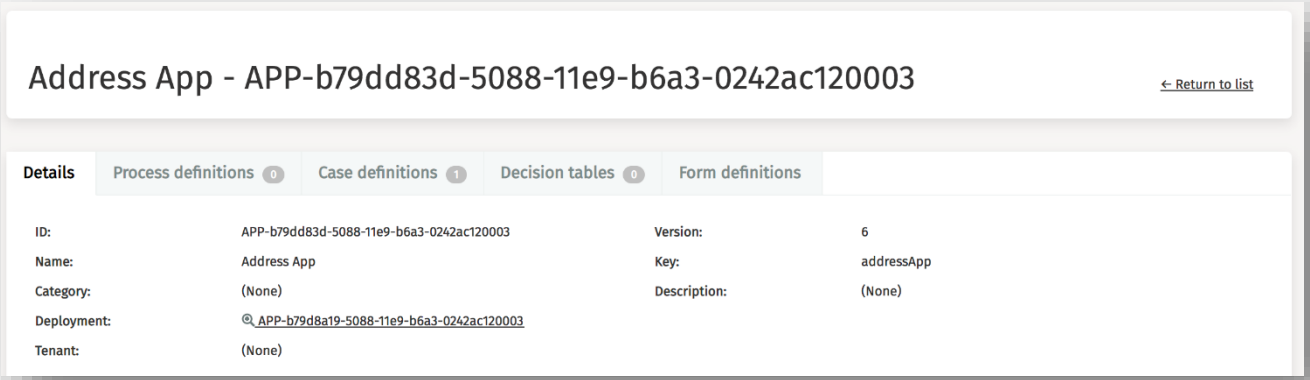
- Save and import your apps in Flowable Design as **zip archives**.
- Either **publish from Design directly** or **export** deployable ZIP files which can then be deployed through REST or via code.
- Models can also be provided as individual files.

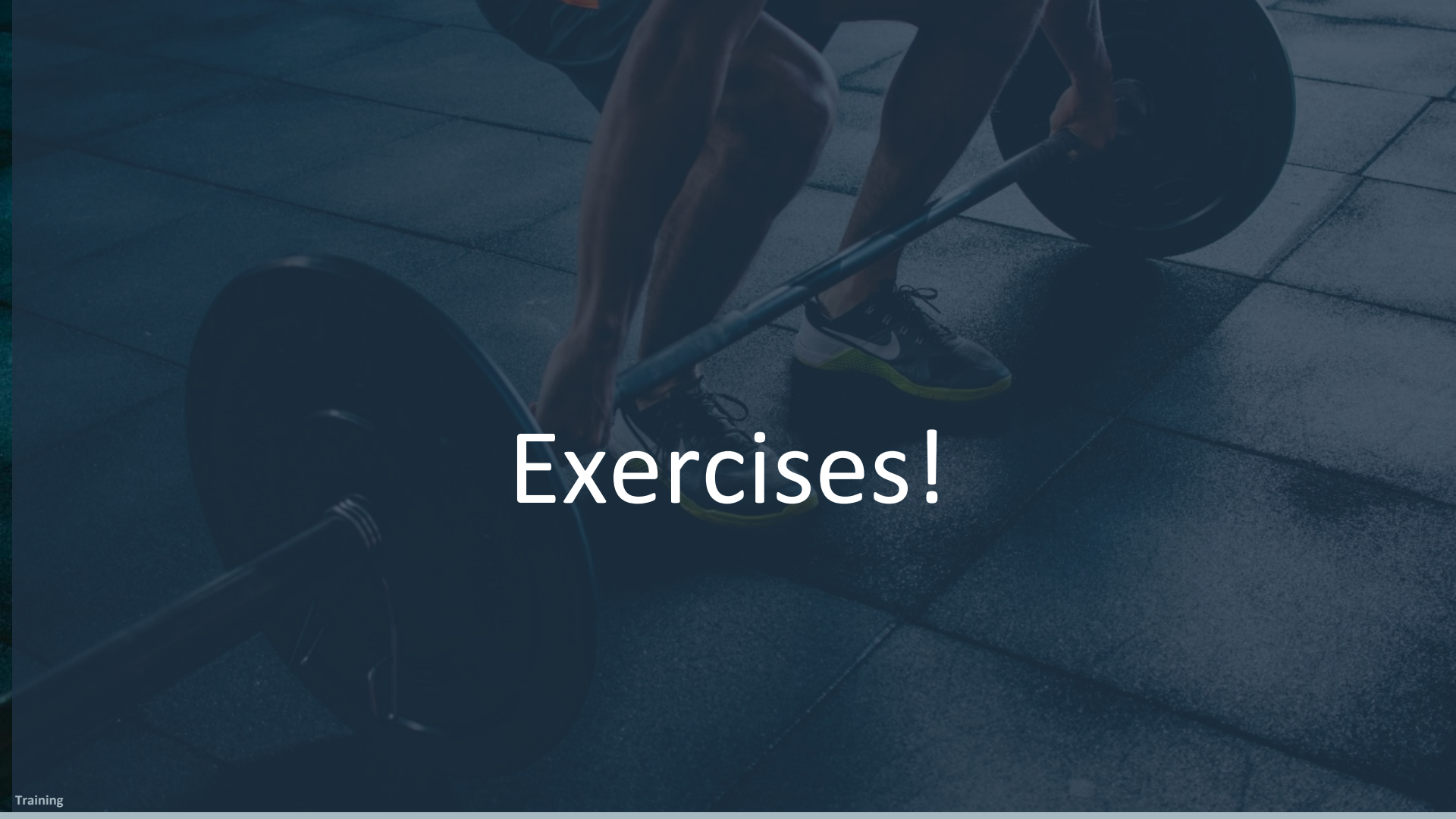


Deployments in Flowable Control

- Deployments
- App Definitions
- Included Definitions:
  - Process
  - Case
  - Decision Table
  - Forms

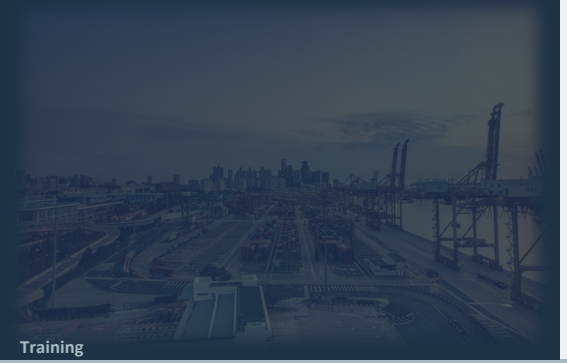





Exercises!

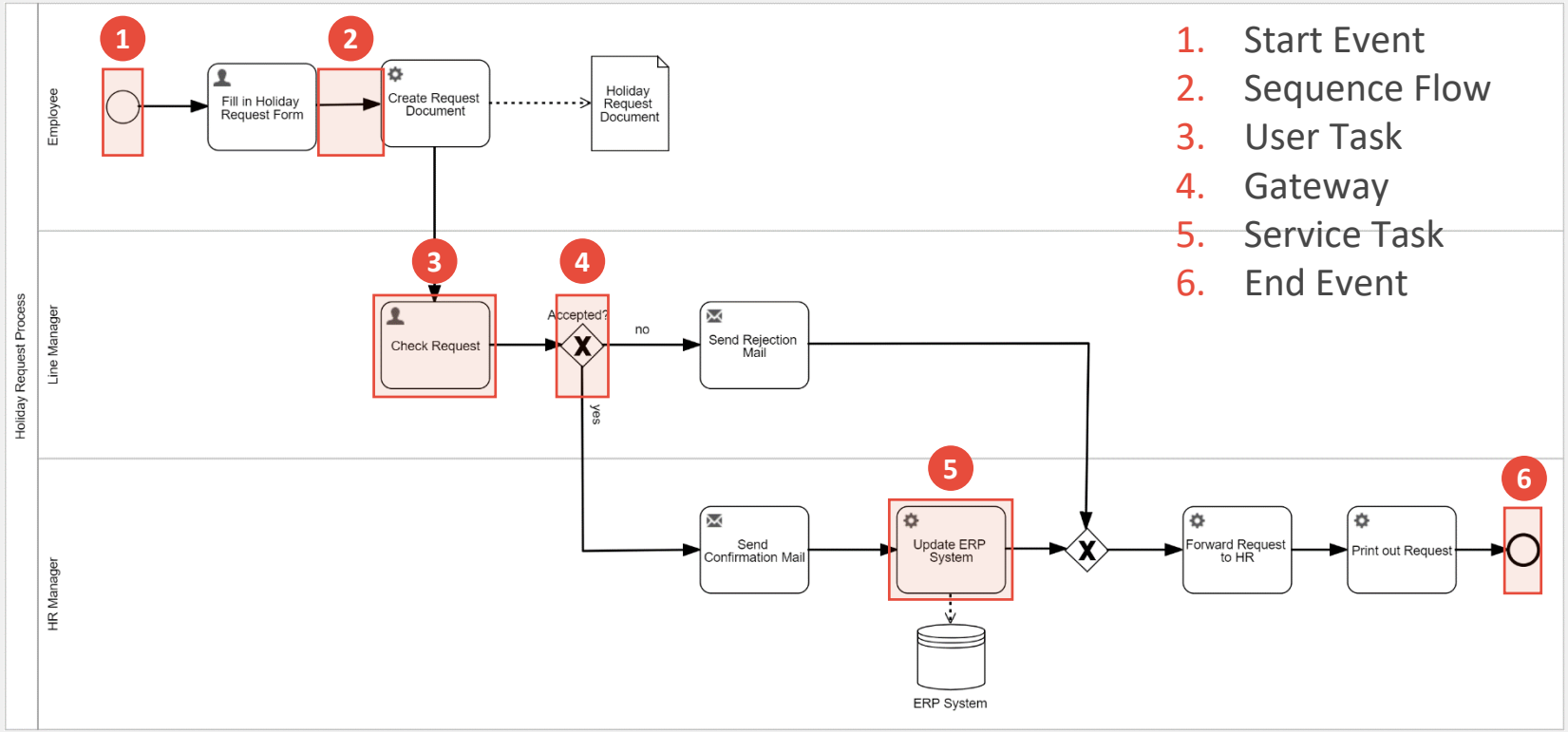

Cases
Processes
Hierarchies

Processes in Flowable



- **BPMN Processes** are the building blocks of your application.
- They can be started **manually**, **through Events**, through the **Java API**, **from Cases** or **from Processes**.
- Processes contain connected **Activities** and **Events**. The flow can be controlled through **Gateways**.
- To control the flow, **variables** are stored on the process.

Executable Process Example

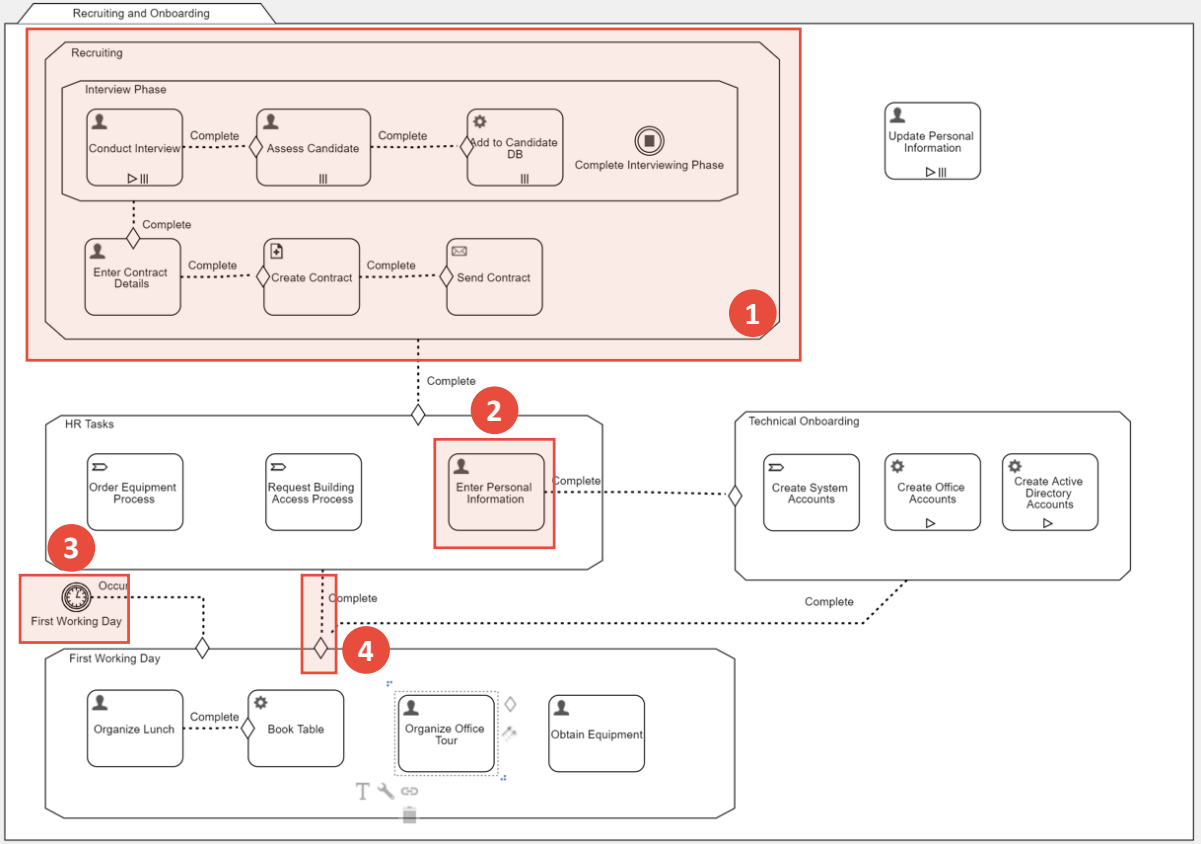


1. Start Event
2. Sequence Flow
3. User Task
4. Gateway
5. Service Task
6. End Event

Cases in Flowable

- Cases offer a way to model an executable **high-level** view.
- They can be started **manually**, **from code**, **from processes,** through the **Java API** or **from other Cases**.
- Cases mainly consist of **Tasks**, **Events**, **Stages** and **Sentries**.
- There is **no direct flow**, case are re-evaluated if any part of them change. Flow is controlled through **Sentries**.
- Case data is stored in **variables**.

Executable Case Example

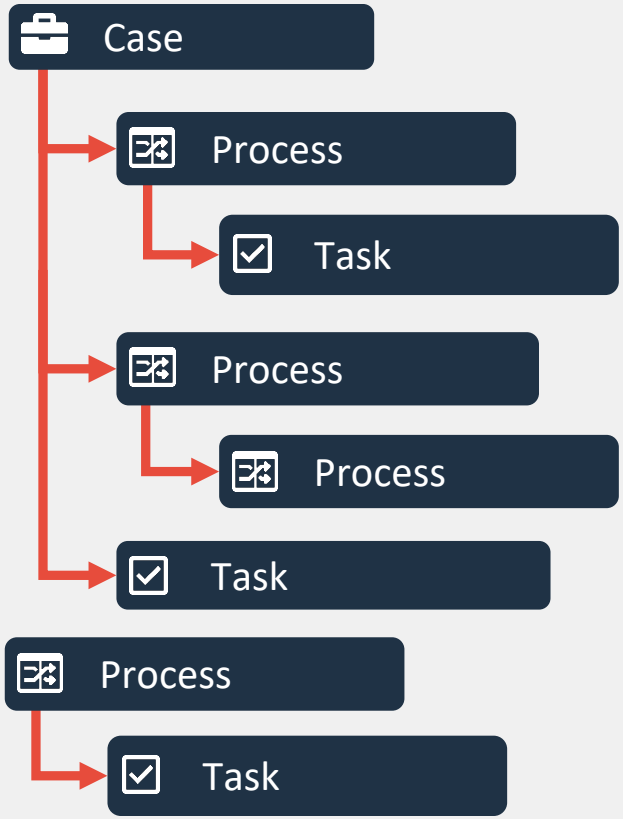


1. Stage
2. Task
3. Event
4. Sentry

Tasks

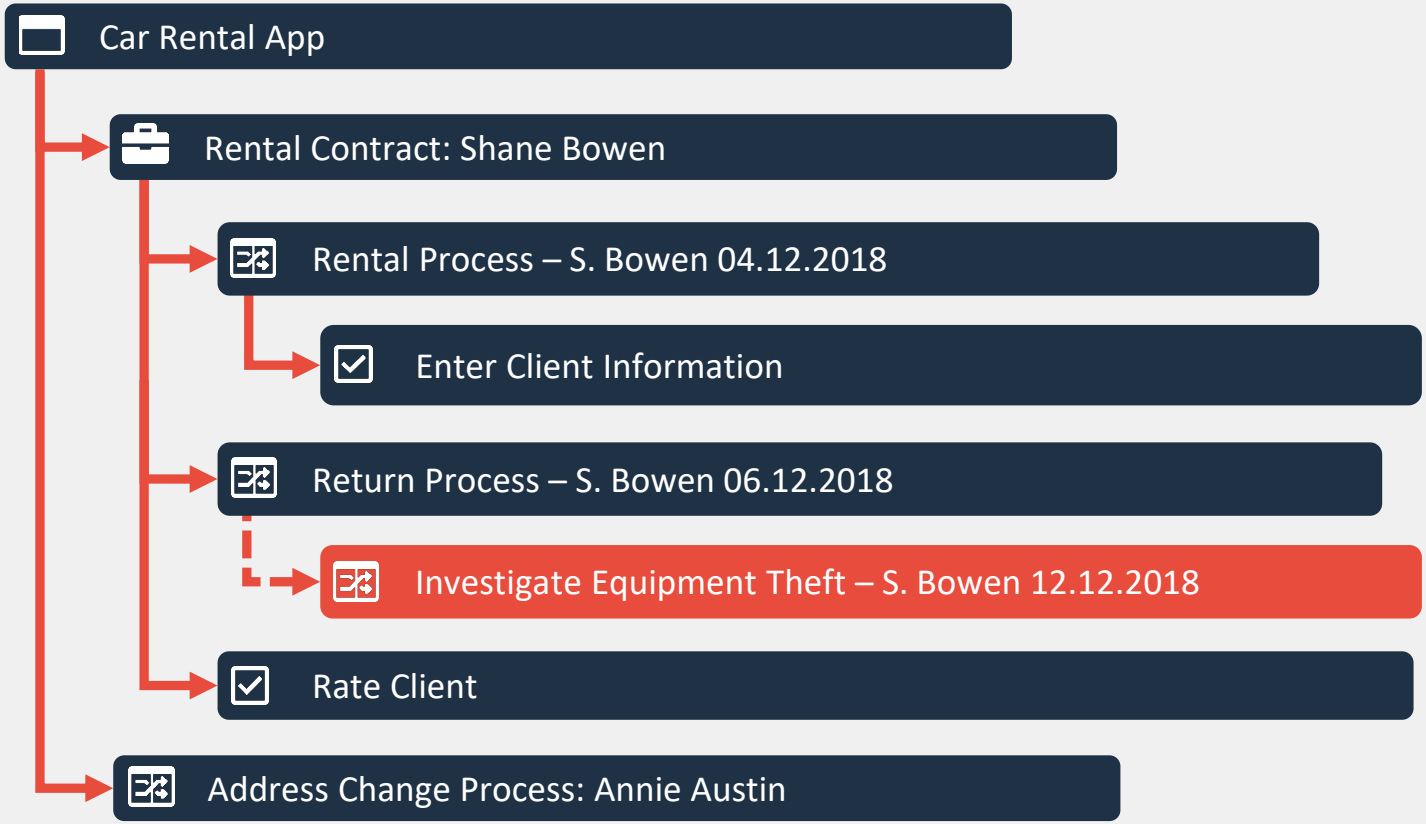
- Tasks allow users to **capture data** through Forms
- They allow you to store variables on a **process** or a **case** level.
- They are represented as “**User Tasks**” in BPMN and “**Human Tasks**” in CMMN
- Completing a task leads to an **advancement** within a workflow
- Access to Tasks is controlled through **Identity Links** (more on that later)

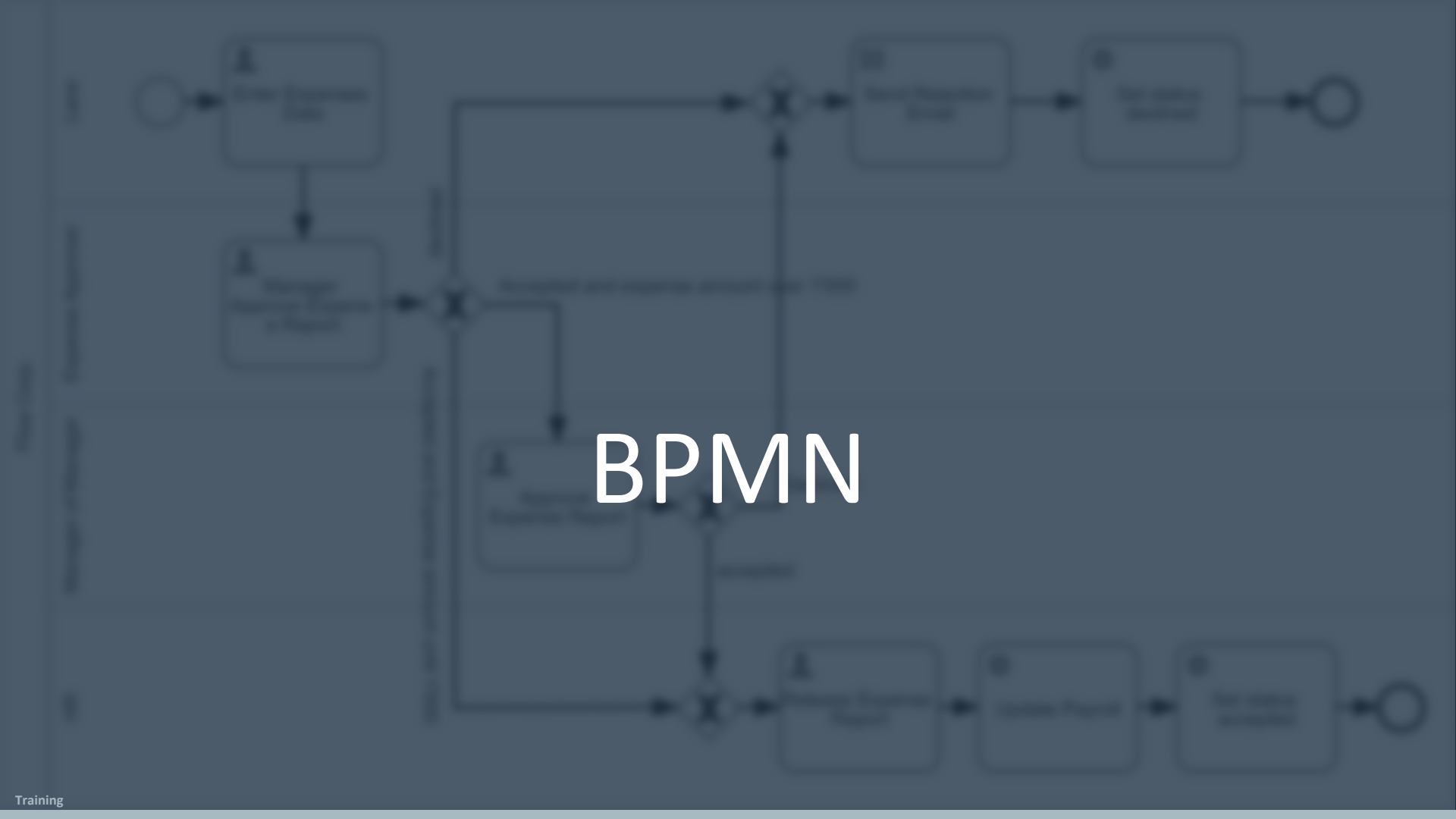
Hierarchies



- The “root” you interact with is either a **Case** or a **Process**.
- **Entity Links** on Cases and Processes can be used to build arbitrary hierarchies.
- They will usually be **automatically** created when they make sense.
- Refer to the next higher level with the keyword **parent** and **root**.

Example: Car Rental App





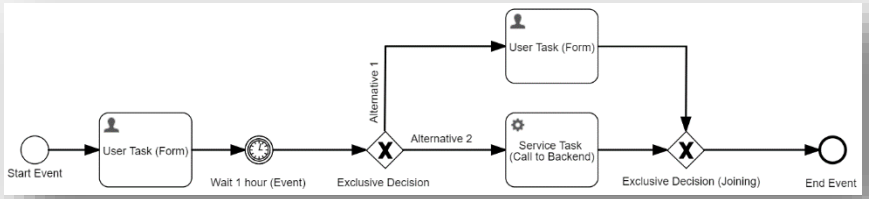
BPMN

BPMN in a Nutshell

Modeling Processes

- BPMN is used to model **processes**.
- **Activities** define what is happening in a process. They are at the heart of your app.
- There is always at least one **Start** and one **End Event** in each process.
- There are optional **Intermediate Events**.
- Activities are connected with **Sequence Flows**.
- You store information from forms, external data sources etc. in **variables**.

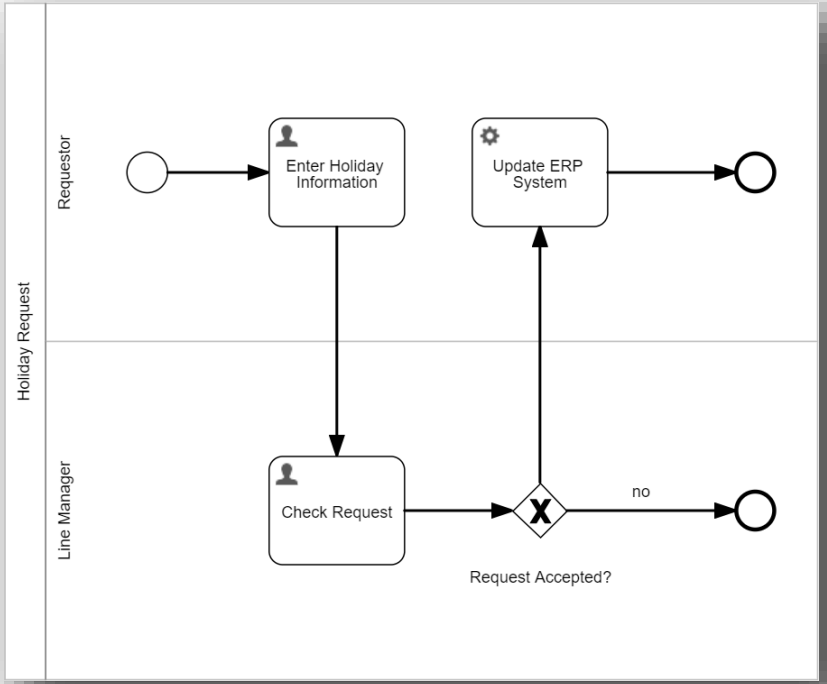
Basic Example



To the left, you see a simple process with:

- A **Start Event**
- Some **User Tasks** (forms)
- A **Gateway** (decision)
- A **Service Task** (backend call)
- An **Intermediate Event** (wait)
- An **End Event**
- Lots of **Sequence Flows**

Pools and Lanes



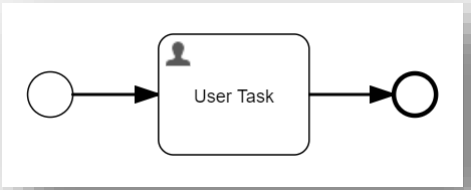
- Most Processes are organized in **Pools** and **Lanes**.
- A Pool usually represents a **business entity** or a Process and can be either be executable or not.
- A Lane represents an **actor** within that Process, e.g. a department or a person.
- There can only be **one executable Pool**.

BPMN Activities

- There are three types of activities
  - **Tasks** that do something
  - **Call Activities** that will *call* another process
  - **Sub-Processes** that allow you to nest processes

- Activities are connected to each other with **Sequence Flows**

- Example

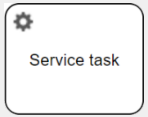


Important BPMN Tasks

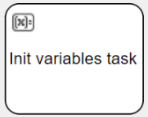
User Task: Prompts a form to the user. This is the most common way to interact with users.

Email Task: An email containing arbitrary content will be sent to a receiver.

Decision Task: Stores variables based on rules defined in a DMN table.



Service Task: Calls Java Logic in the backend and optionally stores the result. Used to interact with other systems, make calculations etc.



Init Variables Task: Initialize or change arbitrary variables.

Gateways

Exclusive: Chooses **one and only one** path based on a condition defined on an outgoing sequence flow.
If no condition is satisfied, the **default flow** will be chosen.
If more than one condition is true, the first defined path will be chosen!

Parallel: Executes **all** paths. Conditions will not have any impact.

Inclusive: Executes **at least one** of the outgoing sequence flows which may or may not have a condition.

Event-based: More advanced gateway that listens to several intermediate events and takes the route of the **first occurring event**.

Events

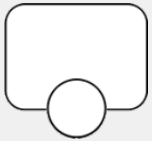
- Events react to the outside world
- There are event families: Signals, Timers, Messages etc.

- There are four types of Events:

Start Events: Triggers the **start** of a process.

End Events: Triggers the **end** of a process.

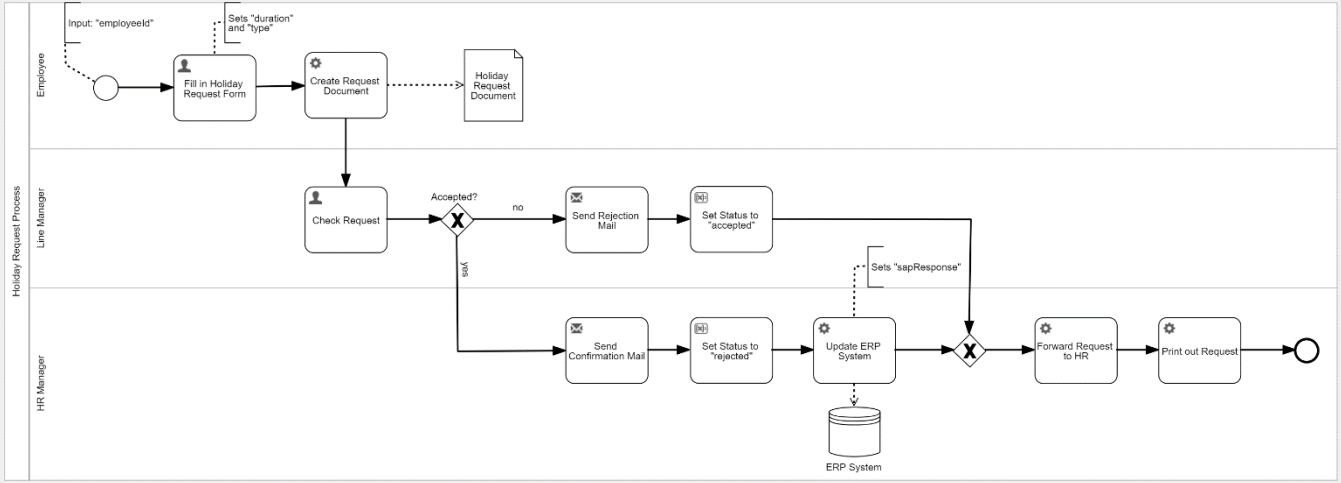
Intermediate Events: **Waits** for something to happen.



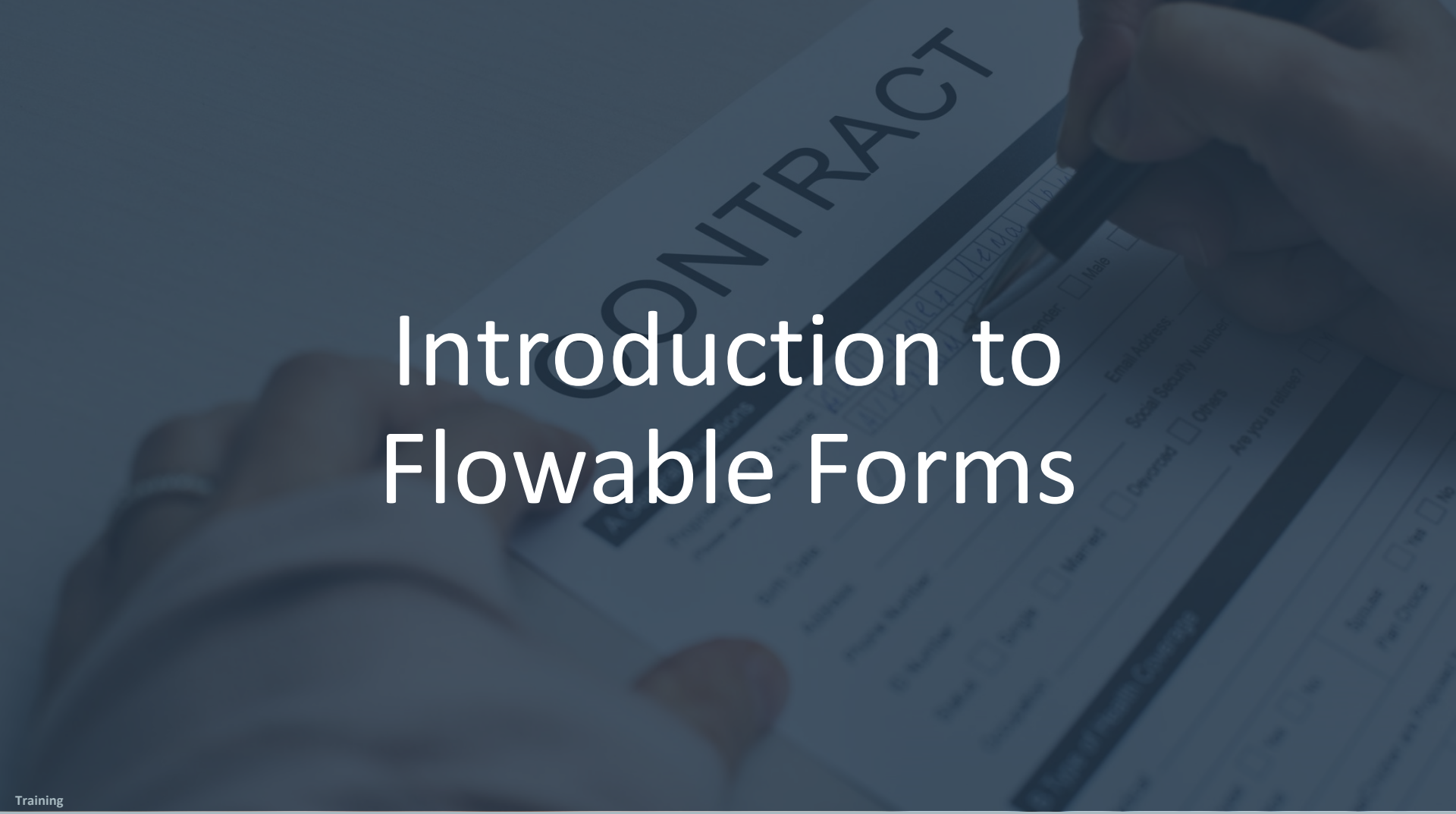
Boundary Events: **Waits** for something to happen as long as the attached activity is active.

How to model a process

1. Define responsibilities (Lanes and Pools)
2. Model the rough flow (Start and End Events, Tasks…)
3. Fill in technical tasks, add expressions and references
4. Add further documentary elements

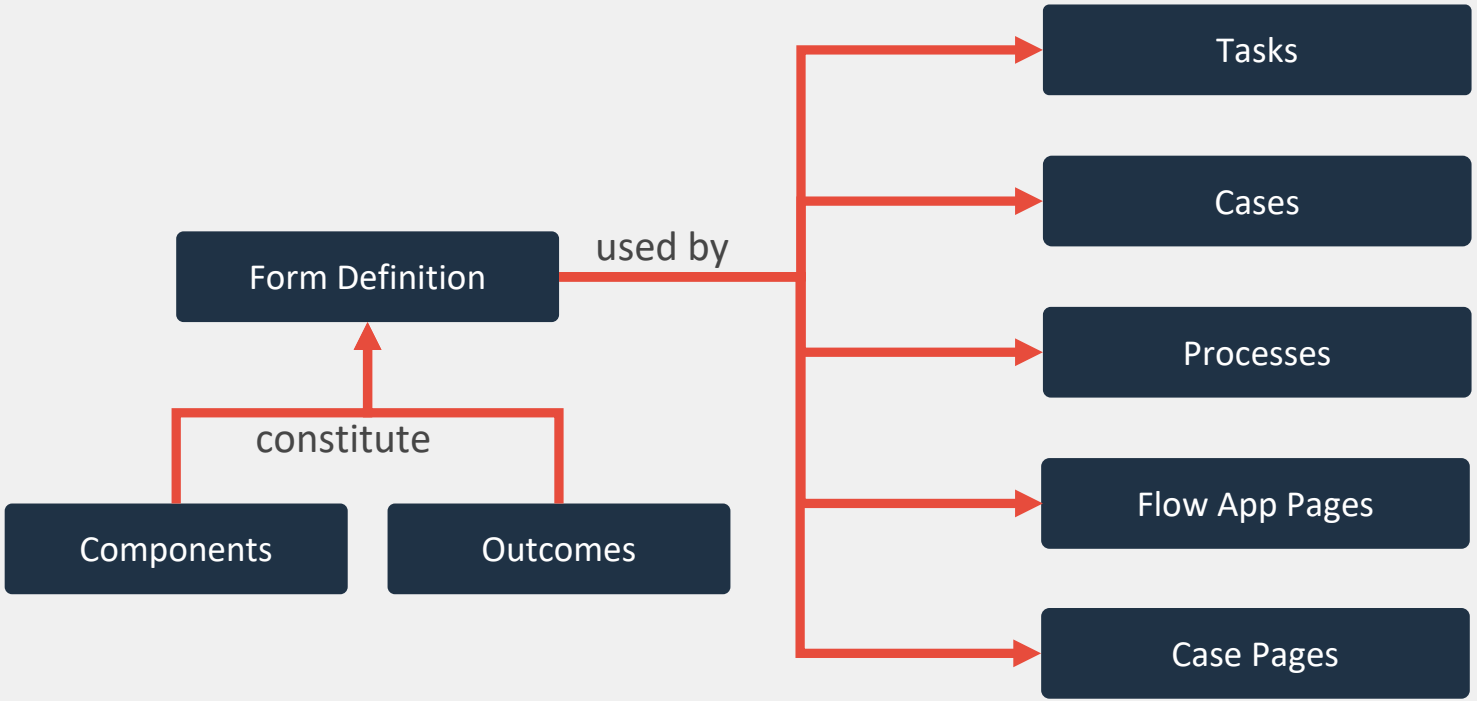



Let’s continue the
Flowable Journey

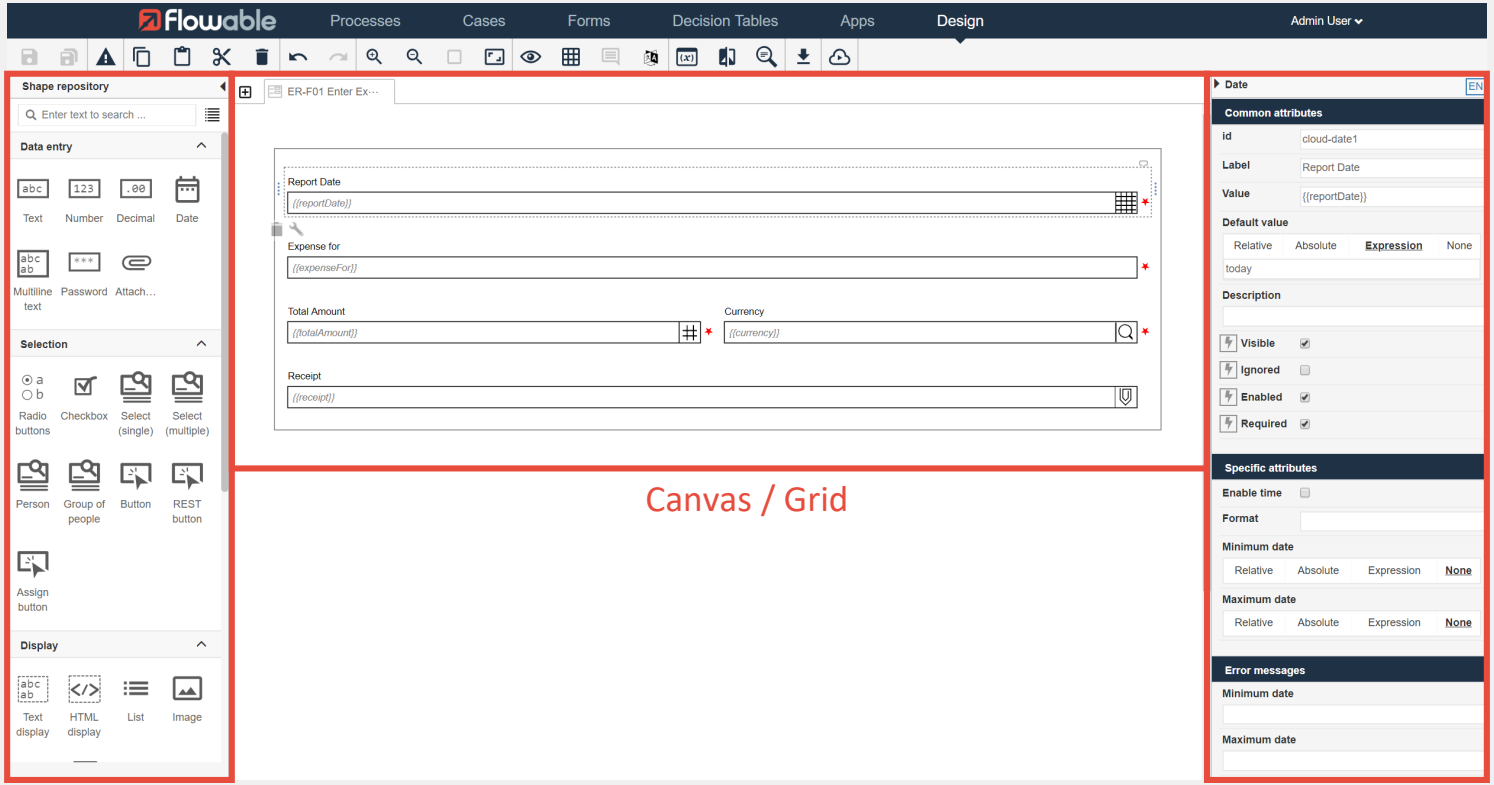


Introduction to Flowable Forms

Forms Are the Visuals of Your App



Forms Designer

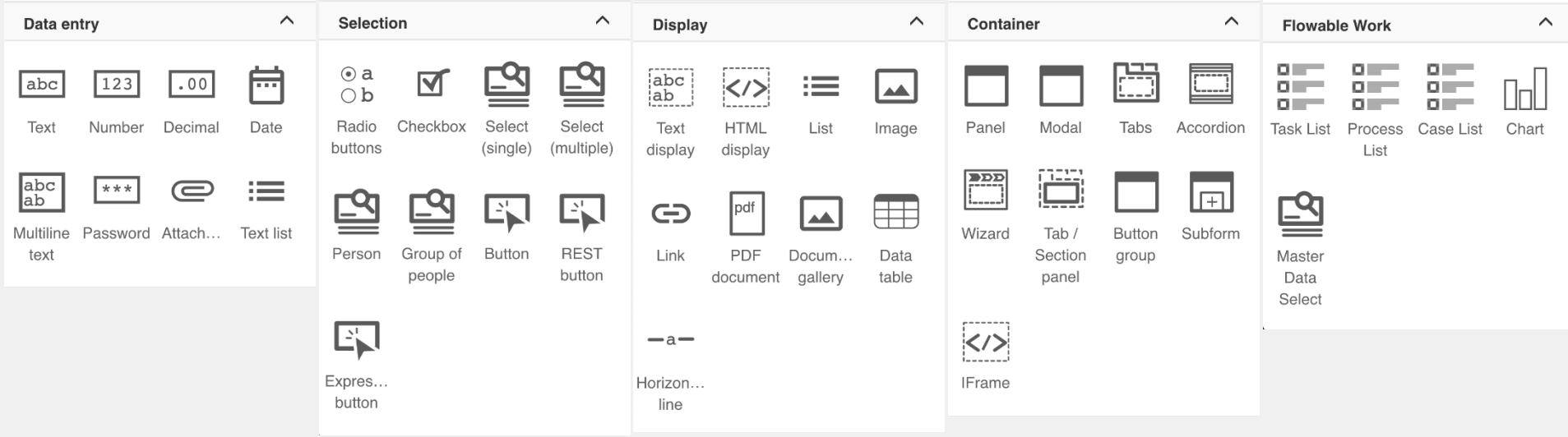


Palette

Canvas / Grid

Attribute Panel

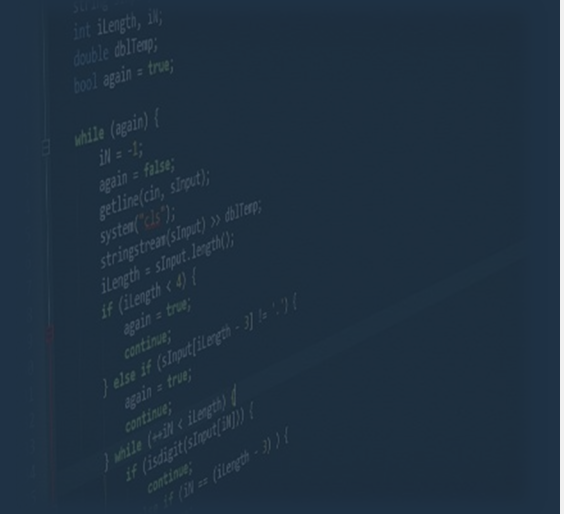
Types of Components




Variables and Expressions

Flowable Data

Store all your data!



- In Flowable, when dealing with cases or processes, we store and access information in **variables**.
  You do not directly access the Flowable DB.

- A variable has a **name,** a **value**, a **scope** and a **type**: Name: `"city"` Value: `"Valencia"`
  Scope: `"PRC-1234-2452-1234"` Type: `"string"`

- There are a number of **variable containers**:
  Process instances, case instances, tasks etc.

- **Not everything is a variable though!**
  Some fields such as IDs, names, business keys etc. are “**native fields**”.

Example: Data in Flowable

Case: Car Rental Case

`"id": "CAS-4fd3a362-84bb-412a-9738"`	Native Field
`"name": "Annie Austin's Card Rental Contract"`	Native Field
`"businessKey": "REN-2009-5231"`	Native Field
`"carType": "Audi A8"`	String Variable
`"startDate": "2019-02-14"`	Date Variable
`"startMileage": 9001`	Number Variable

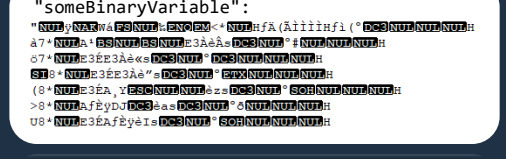
Process: Damage Report Process

`"id":" "PRC-e5834f60-3ff6-4bc1-8235"`	Native Field
`"damagedParts": ["Driver Seat", "Window"]`	List Variable

```
"analysis": {
    "text": "Clearly something is broken."
    "estimatedCost" : 1200
}
```

JSON Variable

"someBinaryVariable":



Binary Variable

Variable Types

Variable Types define what your variables behave like.

Each variable has a **type** in Flowable, for instance:

- **String** Text values
- **Boolean** Boolean values (true or false)
- **Integer, *Short, Long*** Numbers without fractions
- **Double** Numbers with fractions
- **Date, *Instant*** Dates including time
- **LocalDate, *LocalDateTime*** Dates w/o time (e.g. birthdays)
- **JSON** Complex object stored as JSON

- **ContentItemVariable** Content Items (e.g. attachments)
- **DataObjectVariable** Data Object references
- **JPAEntityVariable** Reference to a JPA Entity
- **Serializable (😈)** Binary content (use JSON instead!)

It is possible to define your own types.

How to store variables

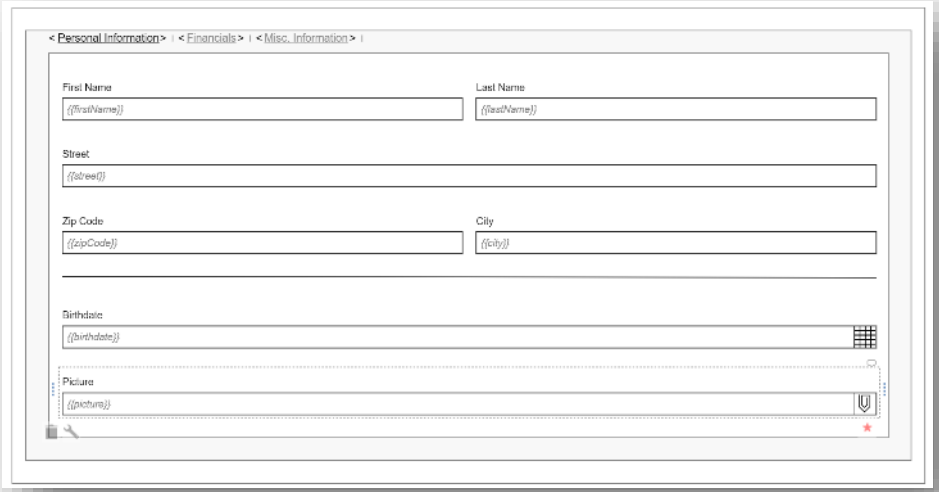
Forms and friends

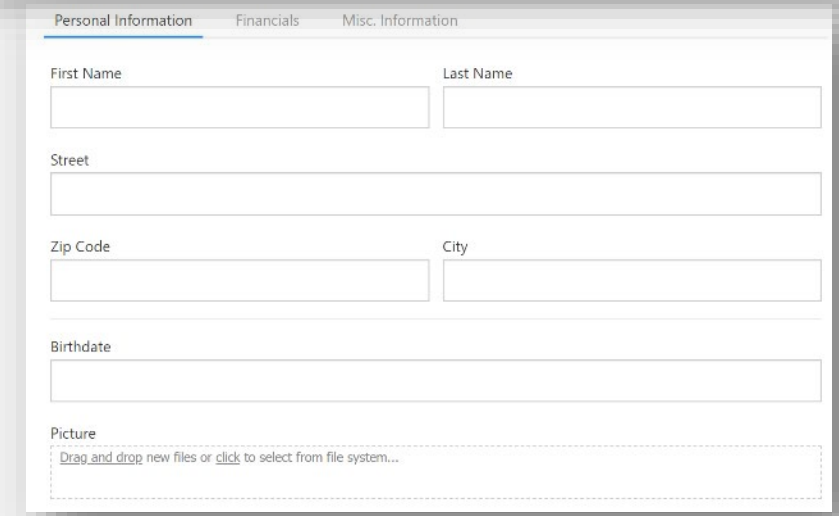
- There are many ways to store variables, e.g.:
  - Through **Form Bindings** (more on that later)
  - As the result of **Service Tasks**
  - Through **Initialize Variables Tasks**
  - As part of some **backend logic**
  - As output of **DMN Decision Tables**
  - Through the **REST API**
- While possible, **avoid changing** the **type of variable** after you defined it!
- Variables can be **deleted** or set to «**null**»

Store Variables with Forms

There are many ways to store variables on a process or case.

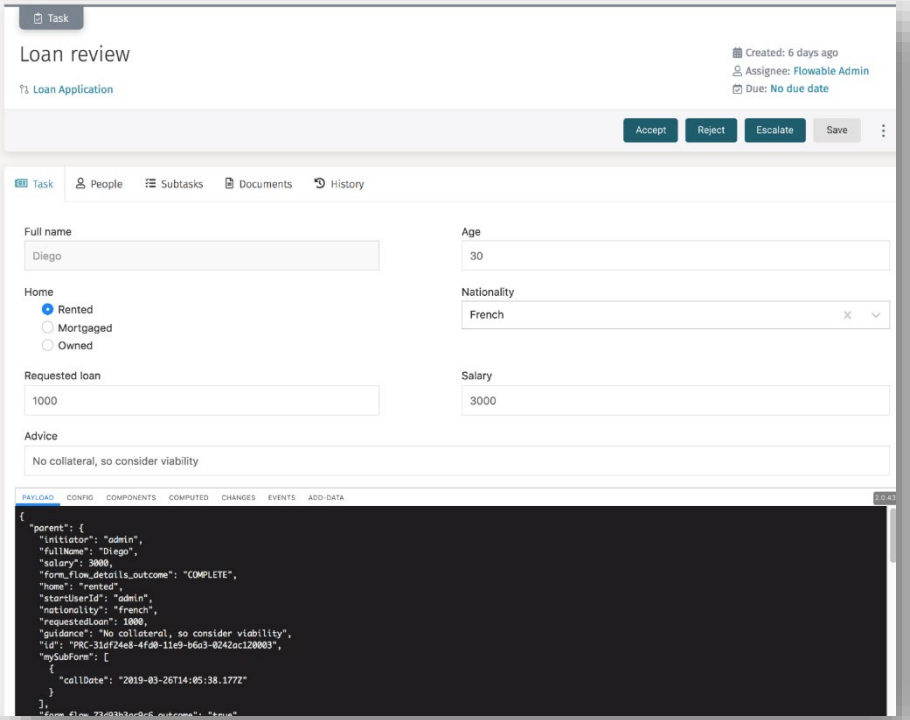
The easiest one is through forms.





Form Payload

- The **payload** determines what will be saved when a form is completed.
- It contains **all variables** of the **parent, root** and **task scope** (e.g. the process or the case).
- Every variable that is added to the payload, will be **stored** in the defined scope.
- **$temp** variables will not be stored!



JSON Data Notation

In Forms, the simple **JSON format** is used to represent data.

```
{
  "name": "Annie Austin",                          ← String, used for text
  "employeeNo": 5921,                                    ← Number, with or without decimals
  "birthday": "1975-04-30T00:00:00.000Z",          ← Date , used to represent points in time
  "isActive": true,                                ← Boolean, “true” (on) or “false” (off)
  "groups":      [                                 ← Array, a list of things
                    "employees",
                    "technical_staff"
                ],
  "address":{                                                ← Object, a complex group of properties
                    "street": "Manessestrasse 87",
                    "zipCode": "8045",
                    "city": "Zurich",
                    "country": "Switzerland“
              }
}
```

JSON Data Types

Number (Integer and Decimal/Float)

Positive and negative numbers, with or without fraction.

```
currentStockPrice = 114.5
```

String

A String is a text value. String are enclosed by quotes.

```
city = "Zurich"
```

Boolean

Expresses the idea of «yes» or «no», «on» and «off». There are only two values: true and false

```
hasAccessToBuilding = true
```

Date and Time

Dates are usually stored as strings, however you can us math operations for calculate dates

```
contractDate = "2020-02-21T14:00:00.000Z"
```

Form Expressions

- To access and store variables in tasks, processes and cases **in Forms**, you use **Form Expressions**.
- Enclose variables in **curly braces**: `{{yourVariable}}`
- You can refer to other parts of complex objects with the . operator:
- You can use most common operators within your expressions:
  - Add values: `{{var1 + variable2}}`
  - Compare: `{{(var1 == var2) || (var3 > var4)}}`
- More information can be found here:
  https://forms.flowable.io/docs/basic-expressions.html

Backend Expressions

- Sometimes, it is necessary to access data in your process. You can use **JUEL** or **Backend Expressions** to do so.
- They look like this: `${yourVariable}`
- You can use the most common **operators** (+, -, /, * etc.).
- You can directly refer to **beans**: `${myBean.myMethod(processVar)}`
- More information can be found here: https://documentation.flowable.com/latest/develop/be/be-expressions/

Example Backend Expressions

Expression	Description
`${firstName}`	Get the first name of a person
`${amount > 10000}`	Checks if the amount is greater than 10000
`${department == 'hr'}`	Checks if the department is ‘hr’
`${name != ''}`	Checks if the name is not empty
`${flwTimeUtils.formatDate( startTime, 'hh:mm')}`	Shows only the hours and minutes of start time
`${isInternalEmployee}`	Checks if ‘isInternalEmployee’ is true
`${hasGoldCard && salary > 10000}`	Checks if somebody has a gold card and that the salary is higher than 10000
`${isSuspicious || forcedCheck}`	Checks if one of the two variables ‘isSuspicious’ and ‘forcedCheck’ is true

Logical Operators

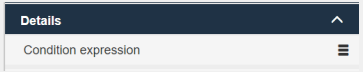
There are a number of **logical operators** in expressions.
The most important ones are:

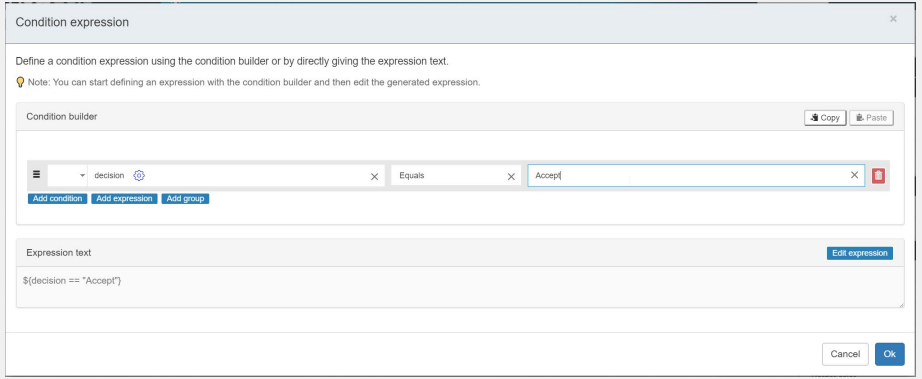
Name	Symbol	Example
Equality	==	country == "ch"
Non-Equality	!=	country != "ch"
Less / Greater than	< >	amount > 0
Less / Greater equals	<= >=	amount <= 100
Logic AND	&&	isSwiss && amount > 100
Logical OR	||	isSwiss || isEu
Precedence	( )	(isSwiss || isEu) && amount > 10

Condition Builder

Your gateway to the world of expressions.

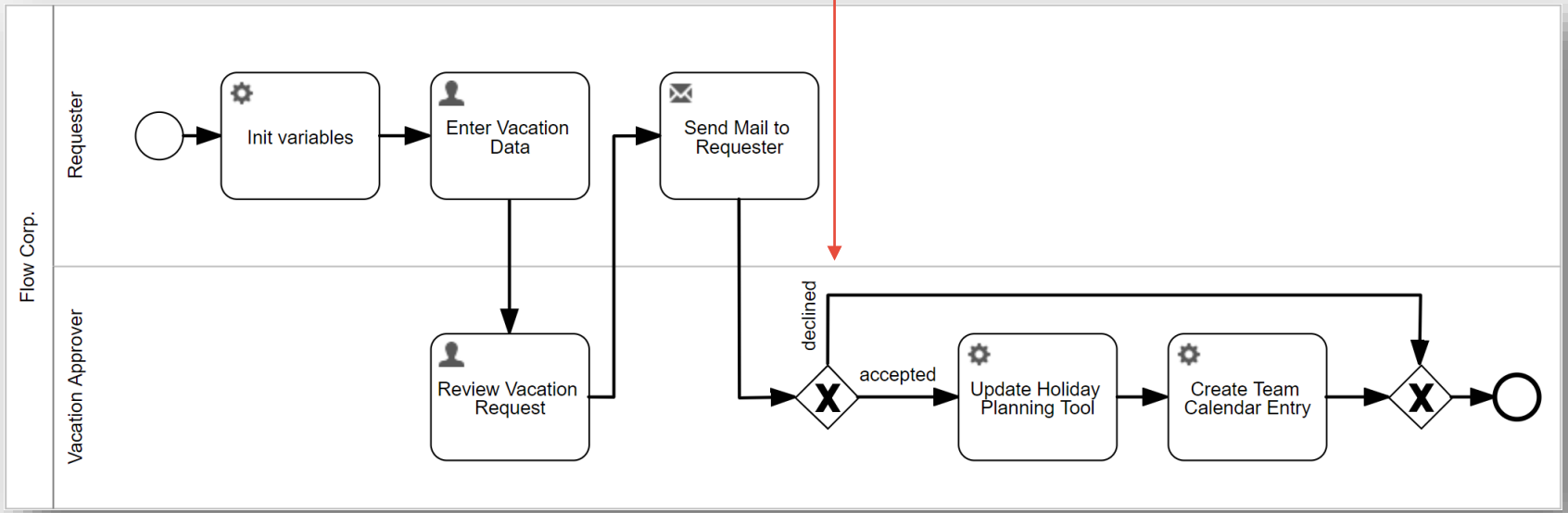
- In Flowable Design, you can use the **Condition Builder** to create boolean expressions.
- These can be used in **gateways**, to create **dynamic visbility rules** in forms etc.
- Based on the data source, you may get useful **options**, for instance outcomes, drop-down values etc.





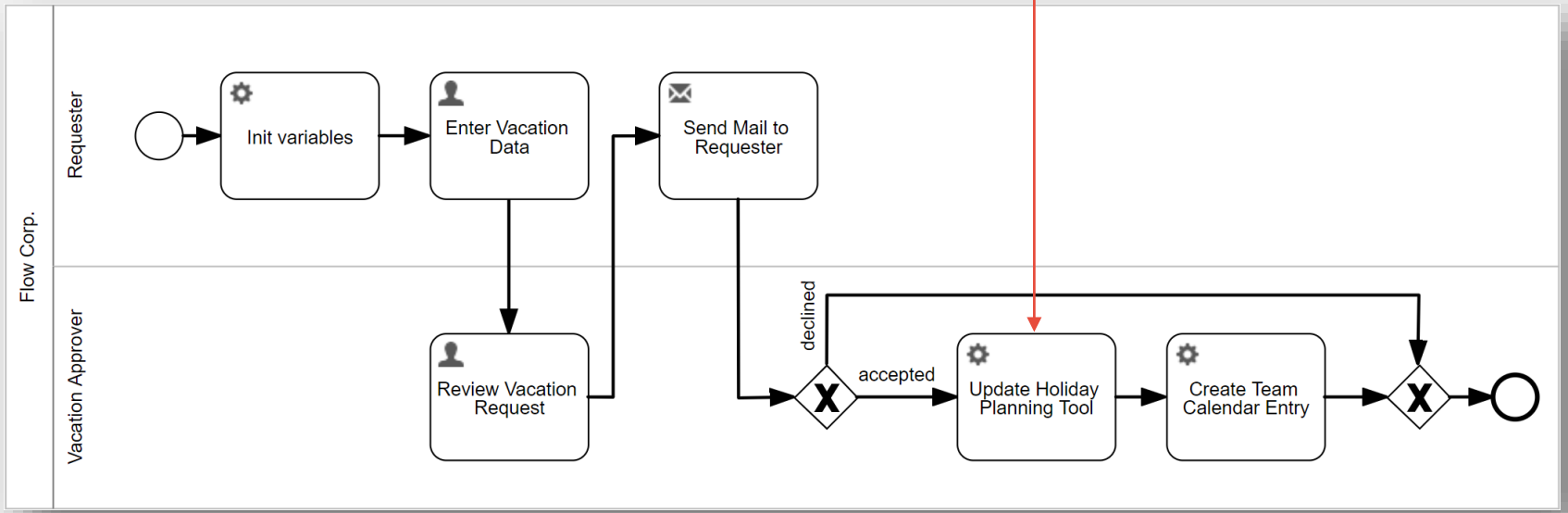
Usage Example: Condition

```
Condition:
${requestStatus == 'declined'}
```

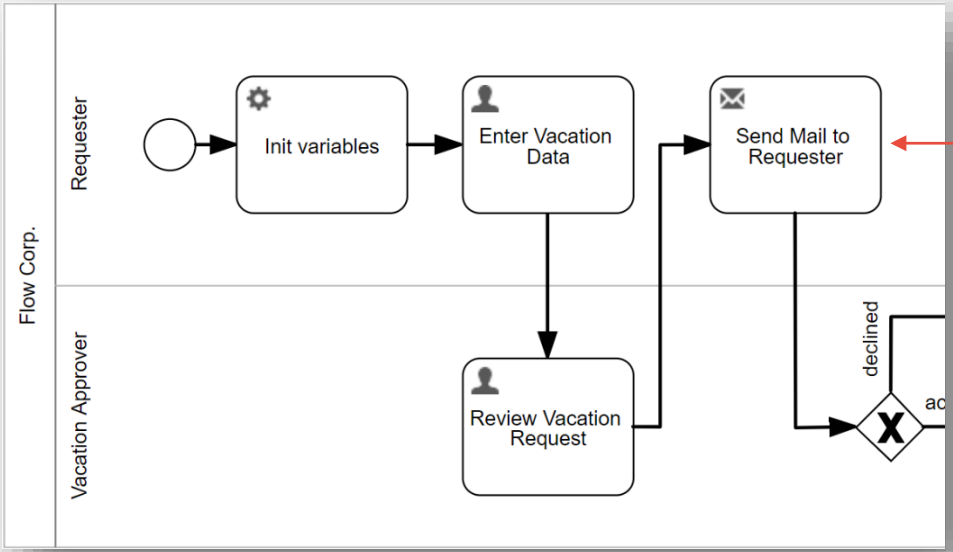


Usage Example: Service Task

Expression:
`${erpService.updateERP(data)}`



Usage Example: Mail Task



${recipient}

Specific attributes

Global ID: GEAR-71936de7-5958-4e87-a33b-24856a323895

Email properties

from:info@flowcorp.com, to:${recipient}, cc:, bcc:, subject:Vacation Request Status, html: -

From: info@flowcorp.com

To: ${recipient}

Cc:

Bcc:

Subject: Vacation Request Status

Html: Go fullscreen for more control

Dear ${firstName},

Your Vacation Request has been #{status}

Form vs. Backend Expressions

When to use what –
the rules are simple!

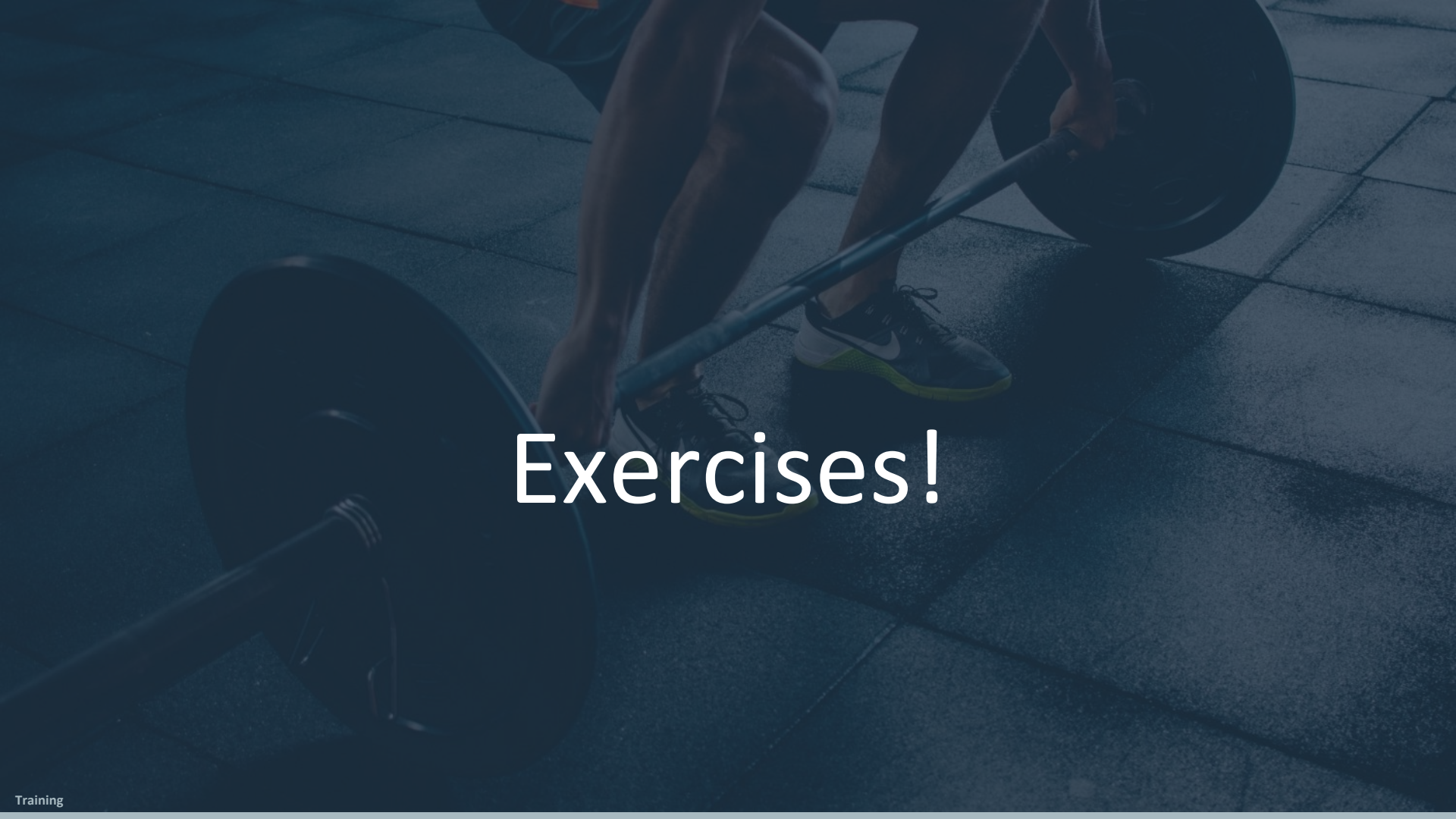
Form Expressions **{{expression}}**

- Forms

Backend Expressions **${expression}**

- Any condition (BPMN, CMMN)
- Expressions in Service Tasks
- Initialize Variables Tasks
- Mail Templates
- Everywhere else **except for Forms**

Remember: **ONLY use {{}} in Forms!**


Exercises!

Exercise 1

- Create a basic process for a vacation request:
  - Add a UserTask to fill the request information
  - You need to allow the process initiator to complete the task
  - Add a basic form to get the name and the number of days

Exercise 2

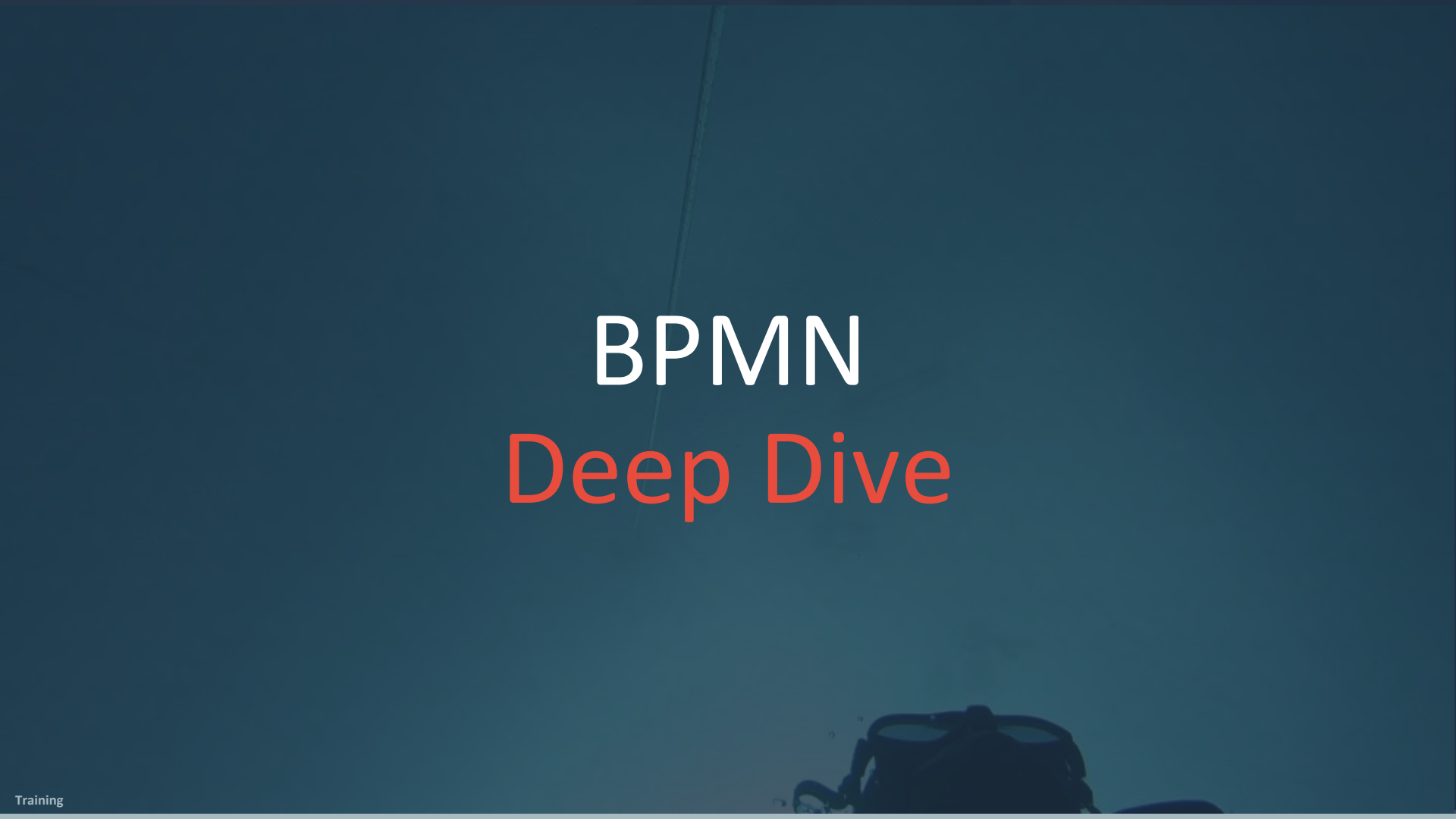
- Extend your process with the following requirements:
  - If number of days >10 create a User Task for the HR department, having as name “Vacation request for ‘*employeeName*’” (no need to create the form)
    - Use Pool/Lanes
  - Create another user task to notify the employee (independently of the number of days)

Exercise 3

- Extend your model with this requirement:

- Two validations are required from the HR department
  - The two user task must be created at the same time

BPMN
Deep Dive



Essential Start Events

None: Most common Start event, the default if the requirements are not specifically demanding something else.

Timer: Start a process at a specific point in time or in regular intervals.
Example: Start process every day at 11 PM.

Message: Start a process as soon as a specific “message” is sent to the engine. This requires you to implement some backend logic.
Example: Start process when new client was opened in CRM

Signal: Start a process when a “signal” is sent. A signal can be local or global and can be sent from another process or in the backend.
Example: start process when monthly close started

Essential Intermediate Events

Timer: Wait for a certain amount of time until to proceed. As a boundary event, it acts as a kind of “countdown”.
Example: Wait for 1 week or until a certain date.

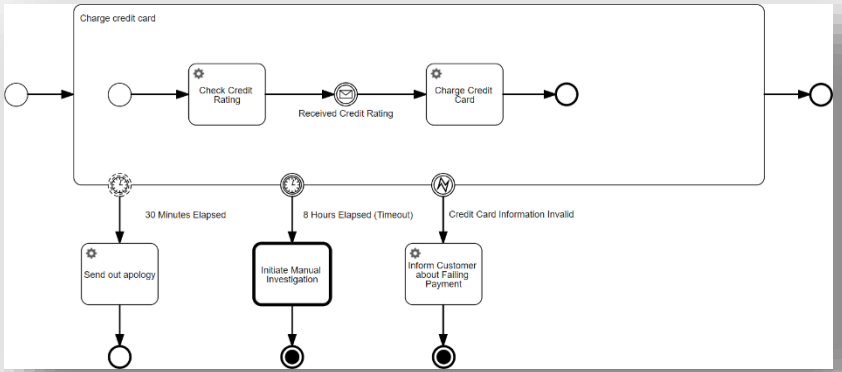
Message: Waits for an external message to arrive.
Example: Proceed as soon as an external system sends a confirmation.

Signal (Catching): Waits for a signal to arrive.
Example: Wait until the end-of-month process was completed

Signal (Throwing): Throws a signal to be consumed by other processes.
Example: Indicate that the end-of-month process was completed.

Boundary Events

- Some events can be placed on the «**boundary**» of an activity (subprocesses, tasks, call activities…)
- They allow you to **react to anything that happens** anywhere within that activity.
- Such events can be «**Interrupting**» or «**Non-interrupting**».



- Interrupting events will “**stop**” what is happening within the activity to which it is attached.

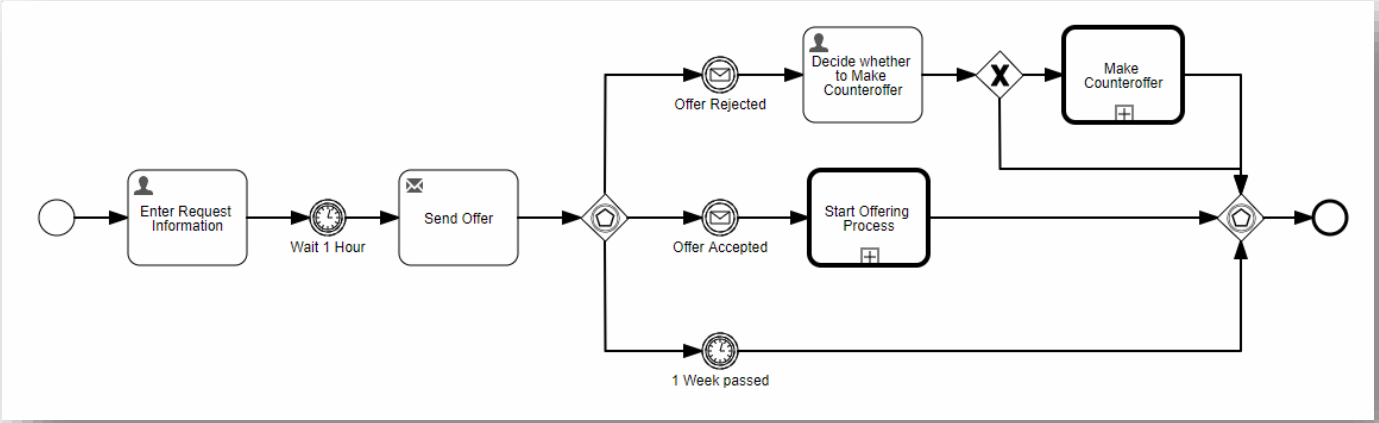
Essential End Events

None: Most common End event. The current execution path will simply be ended. If the last token arrives at such an event, the process will be completed.

Terminate: Terminates/completes the current process and, optionally, the parent case if there is one. It will not wait for other execution paths to complete.

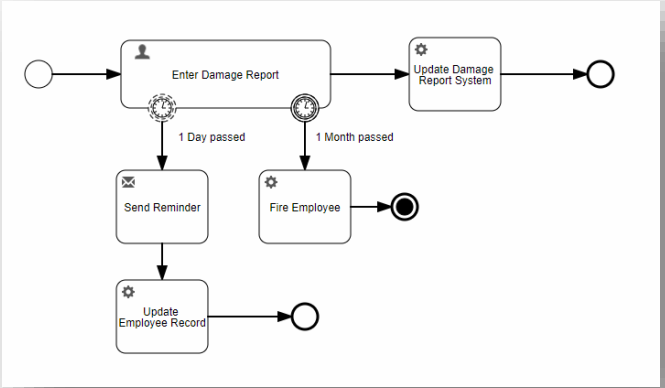
Example Events

- After the request has been entered, the process stops for one hour.
- If an «**Accept**» message is received, an offer will be sent out.
- If a «**Reject**» message is received, an optional counteroffer is sent out.
- If one week passes without a response, the process will be completed.



Example Events

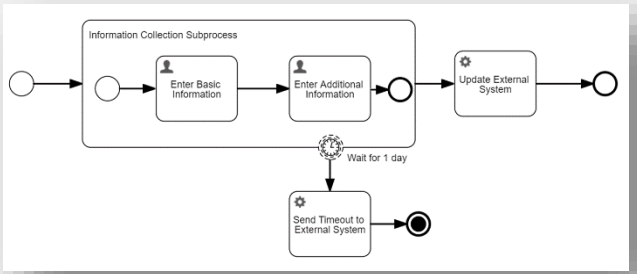
- An employee is prompted to fill in a damage report.
- Each day, a reminder will be sent out and a record will be made.
  However, the process still continues.
- After a month, the process will continue and the employee will be fired which will then terminate the process.
- If the employee fills in the damage report, the process continues.



Call Activities and Sub-Processes

Call Activities call another process. This allows you to better manage complexity.

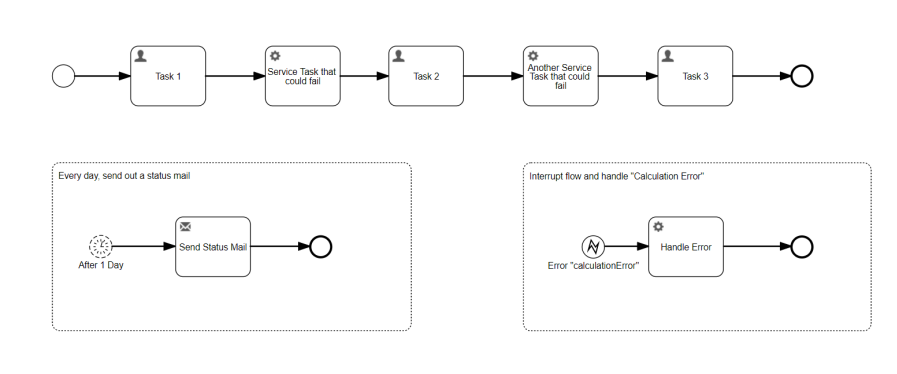
Sub Processes allow you to model a process within a process.



This is helpful in more advanced use cases such as error handling, looping etc.

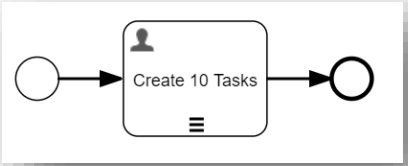
Event Sub-Processes

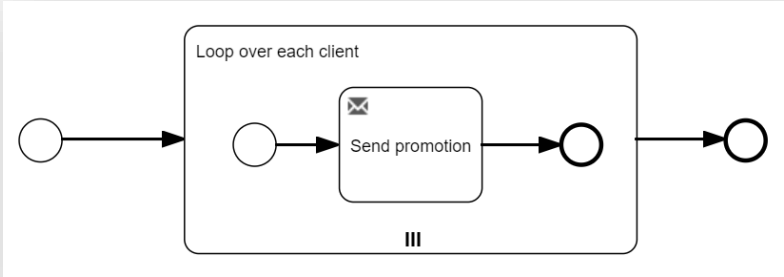
- With **Event Sub-Processes**, you can listen to events at any point during the execution of your process.
- They support many events, e.g. **Timer, Error**, **Signal** or **Message** events.
- They can be wrapped inside sub-processes, if desired.
- The events can either be **interrupting** or **non-interrupting**.



Loops

- It is possible to loop (over a list) in BPMN.
- You have to set the **Loop Type** on an activity to either «**MI Parallel**» or «**MI Sequential**»
- Either loop a fixed amount of times (**Loop Cardinality**) or specify a **Loop Collection** and a **Loop Element Variable**.

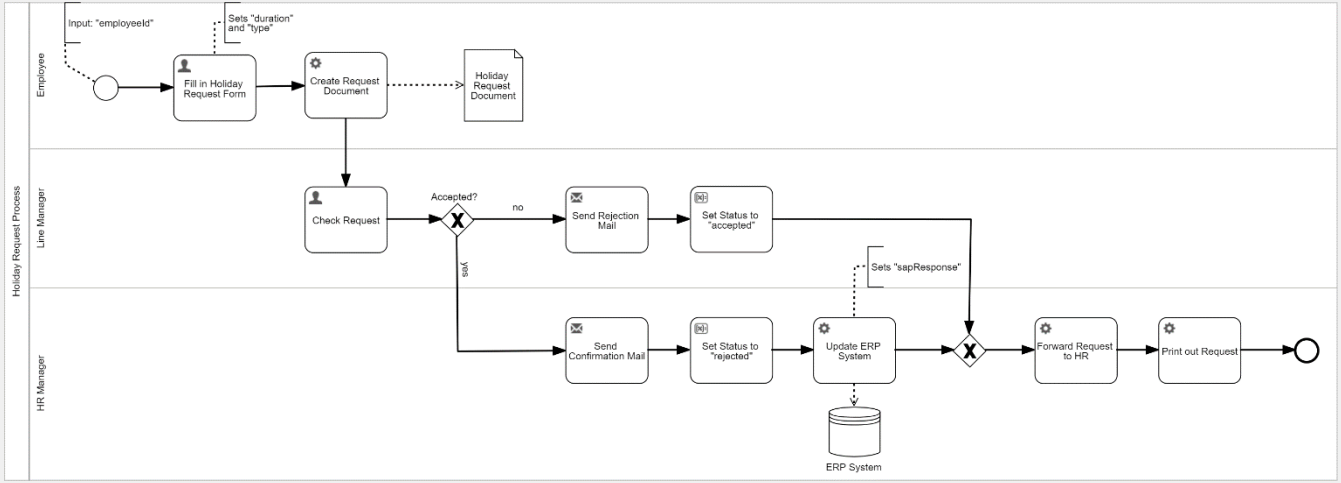




How to model a process?

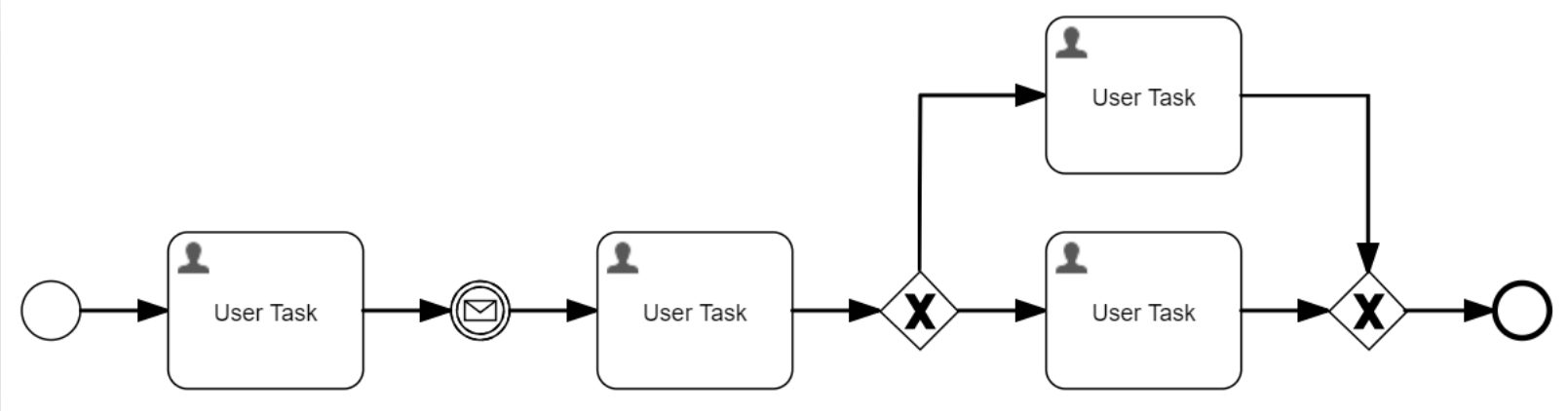
1. Define responsibilities (Lanes and Pools)
2. Model the rough flow (Start and End Events, Tasks…)
3. Fill in technical tasks, add expressions and references
4. Add further documentary elements

RECAP

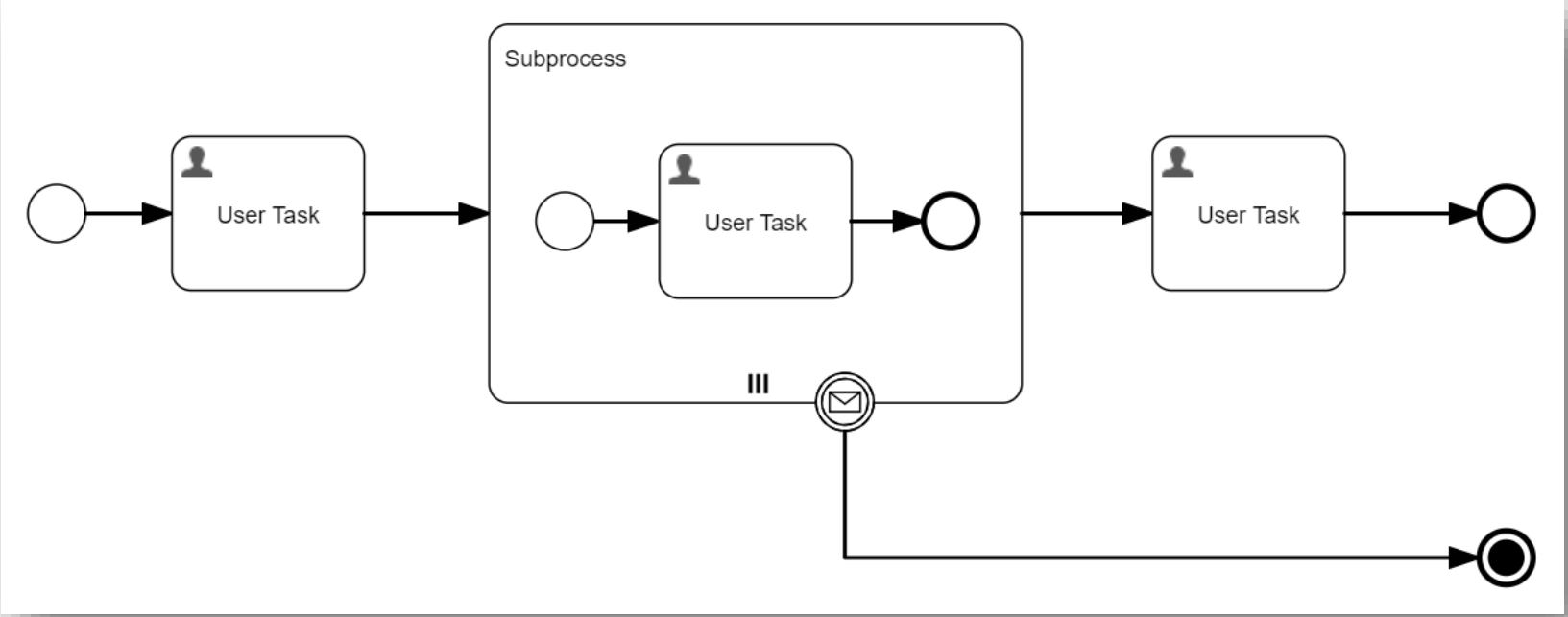



Be creative,
try it yourself!

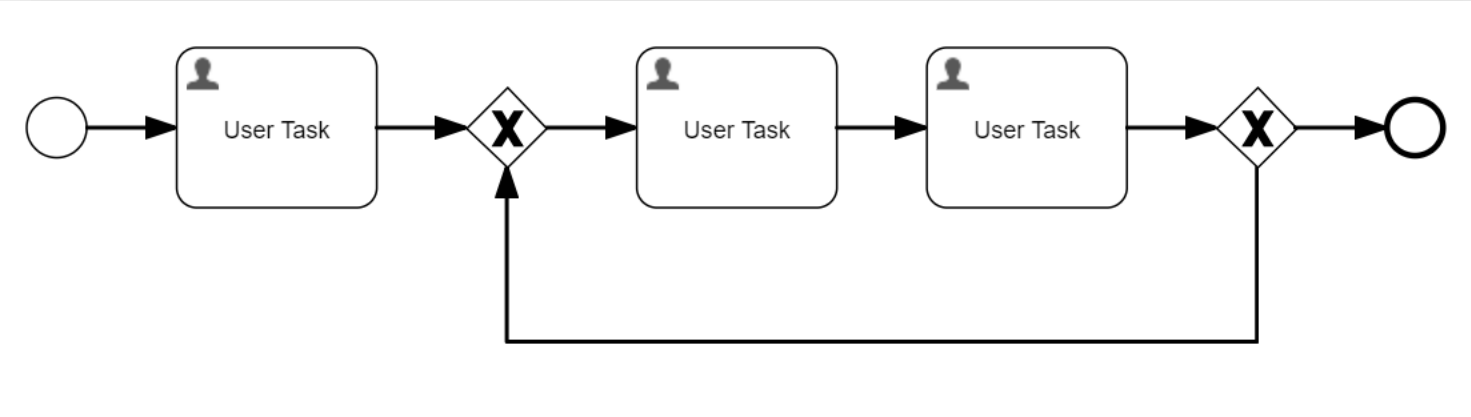
Explain what's Happening 1



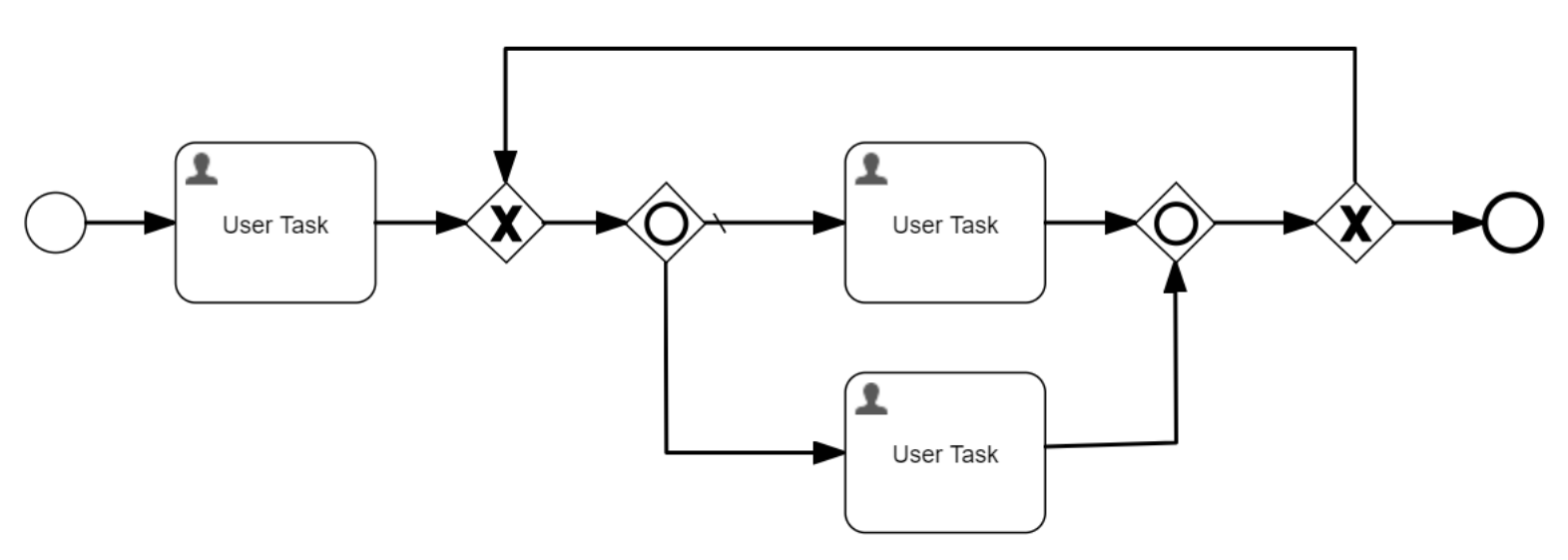
Explain what's Happening 2

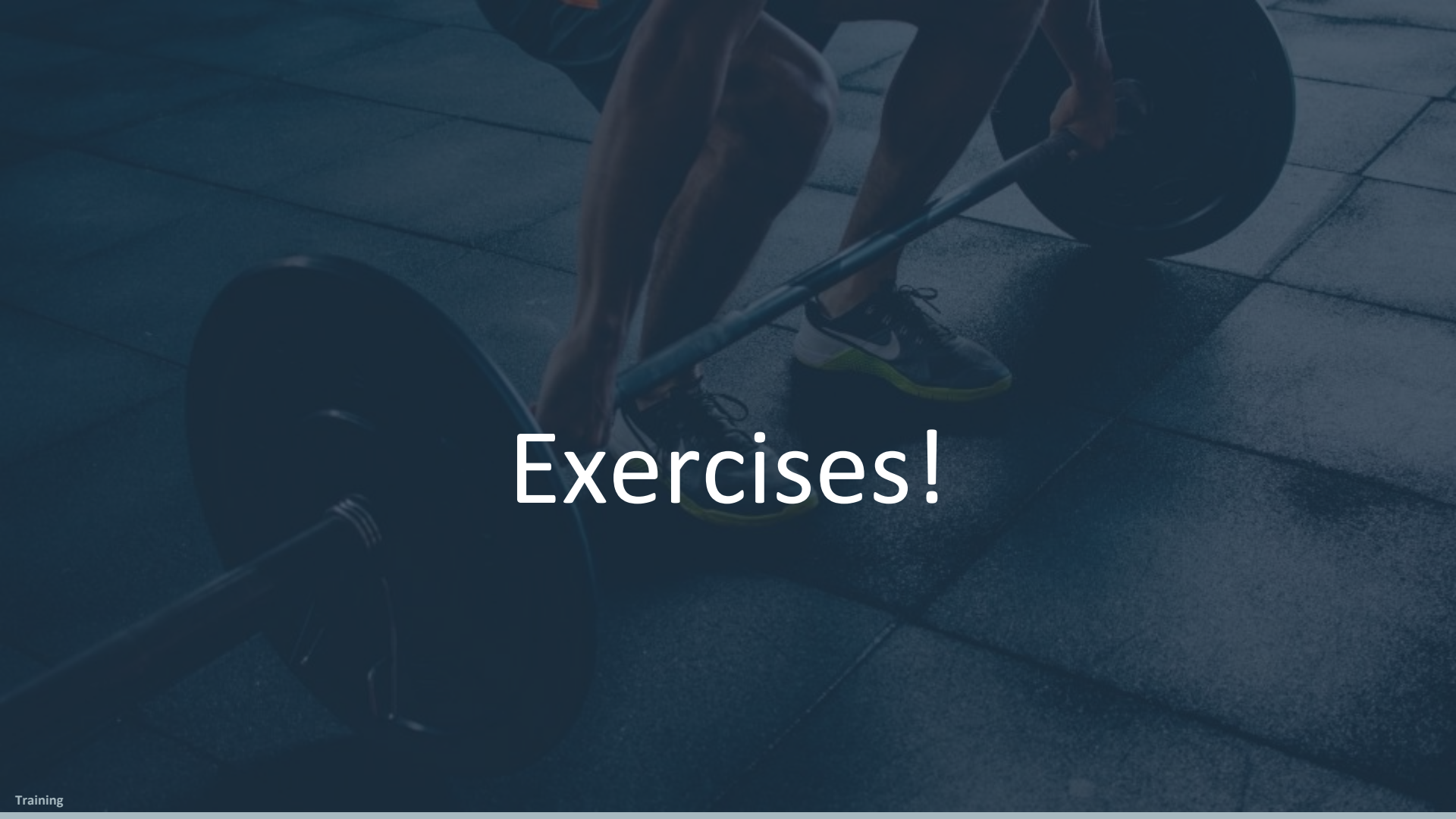


Explain what’s Happening 3



Explain what's Happening 4

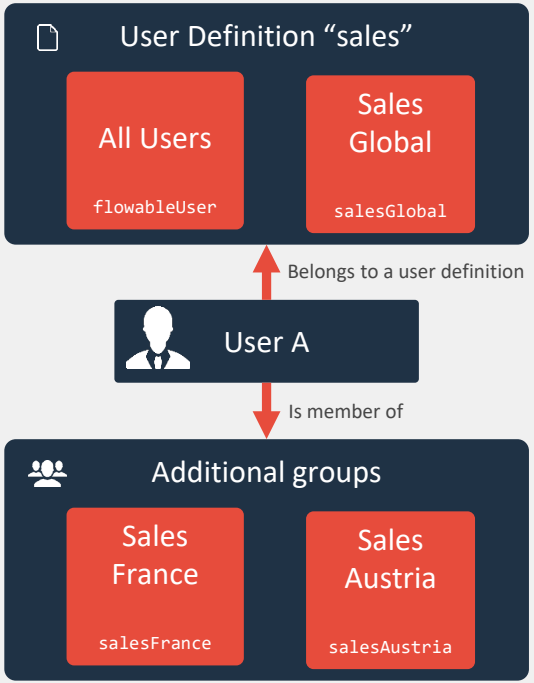



Exercises!

Permissions and Security

Users, Groups, User Definitions

- **Every actor** which interacts with tasks, processes or cases is backed by a **user**.
- Users can be members of **one or more groups**.
- Each user belongs to one **User Definition**.
  User definitions are templates for a user.
- They define the **available features**, **standard groups** etc. for a set of similar users.
- Users and groups can be **stored in Flowable internally** or in an **external identity provider**
  (e.g. LDAP).

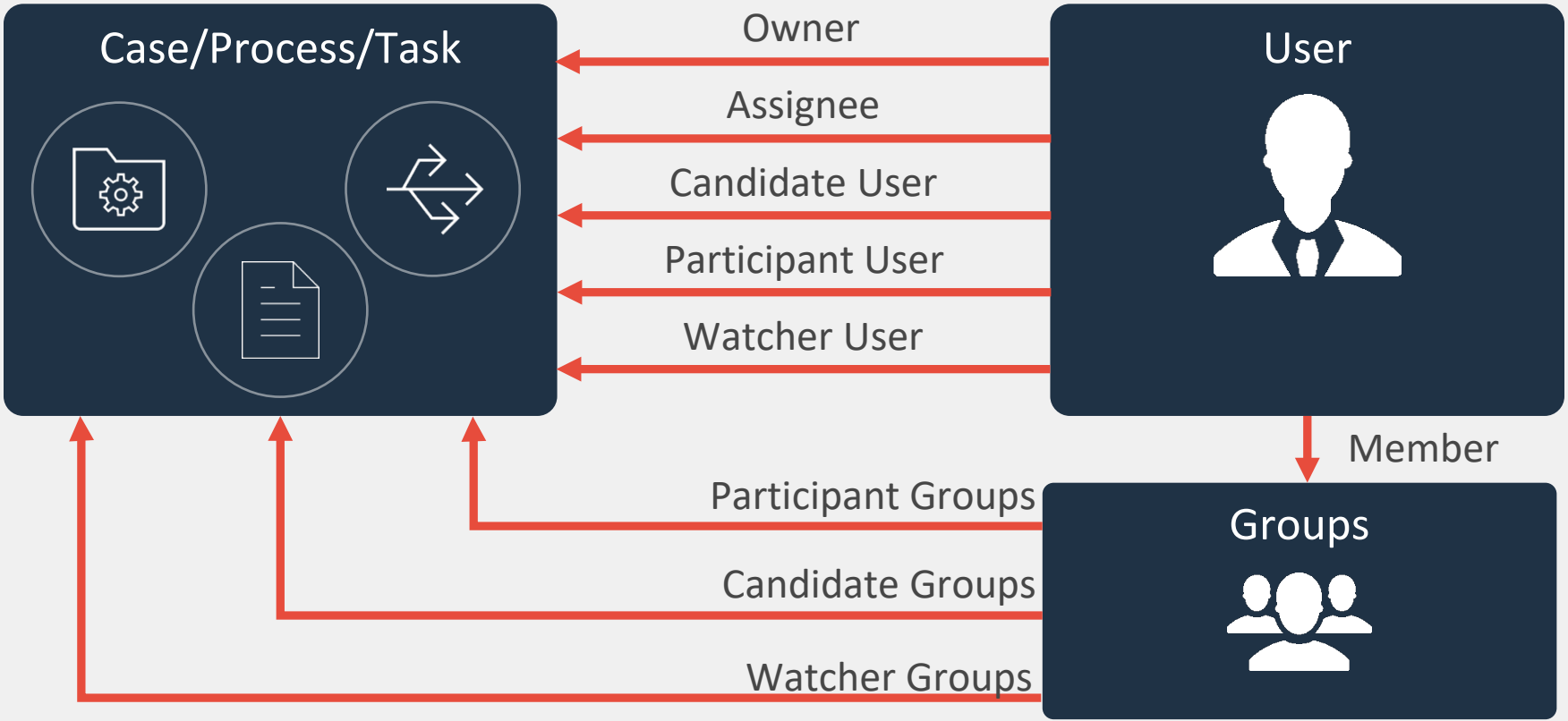


Identity Links

Connect your entities

- Users and groups can be connected to **tasks**, **processes** and **cases** through **Identity Links**
- There are a number of predefined **link types**, e.g. Assignee, Candidate, Owner…
- You can also define **custom** Identity Link Types
- Their concrete impact is determined by **Security Policies**

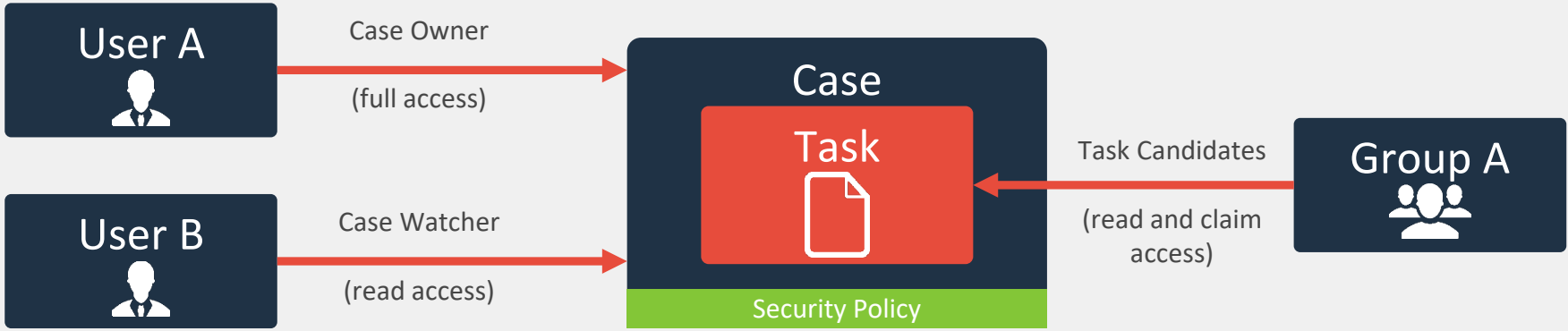
Default Identity Links



Visibility and Access

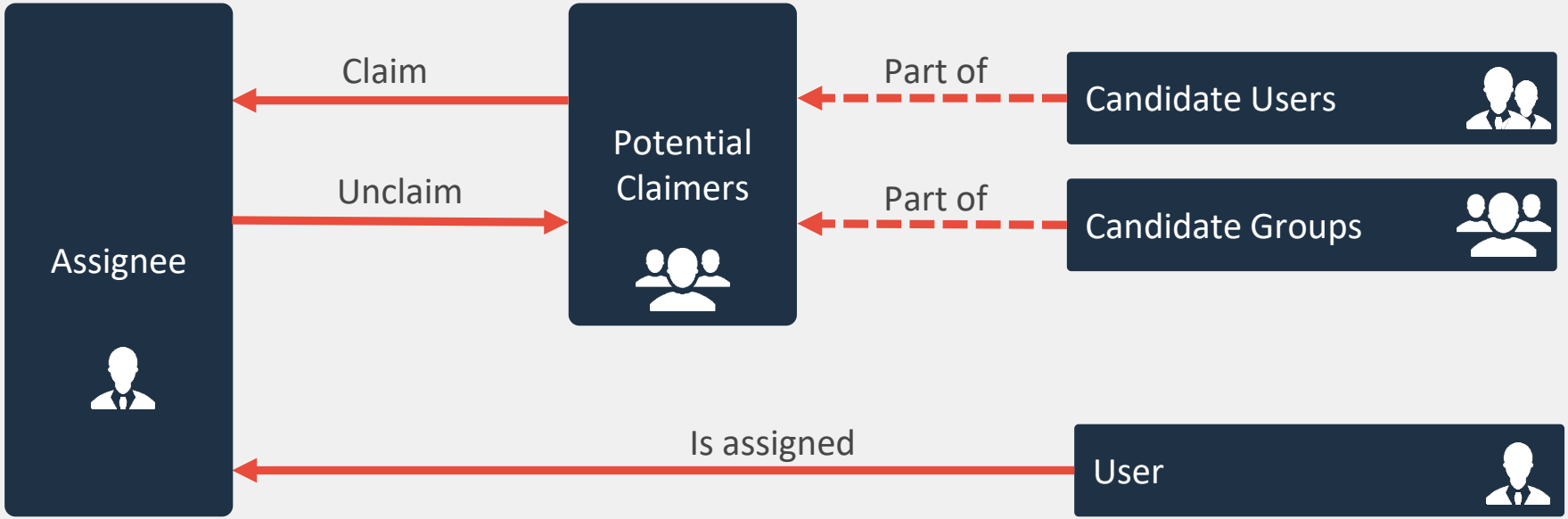
- The **Security Policy** of cases and processes define actions that can be performed (**view**, **edit**, **delete**, etc.)

- Grant access by adding an **identity link** pointing to a **user or group**.



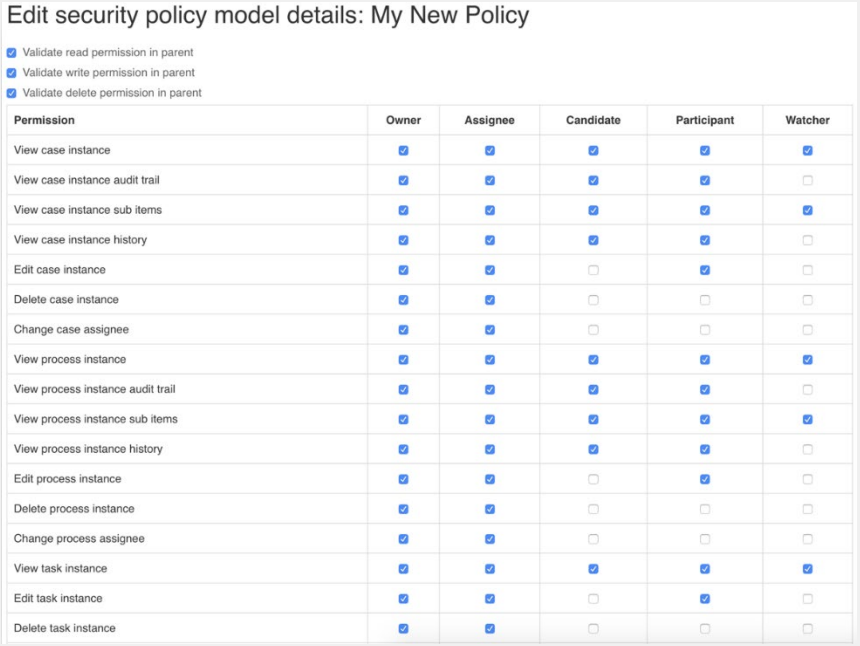
Default Task Assignment

The assignee is the person who can **complete** a task.



Security Policies

- Security Policies offer a way to **customize permissions**
- Default policies: **Basic** and **Strict**
- Policies can be created in **Flowable Design** or as part of your code
- **Reference** them in a process, case or task to apply specific security concepts

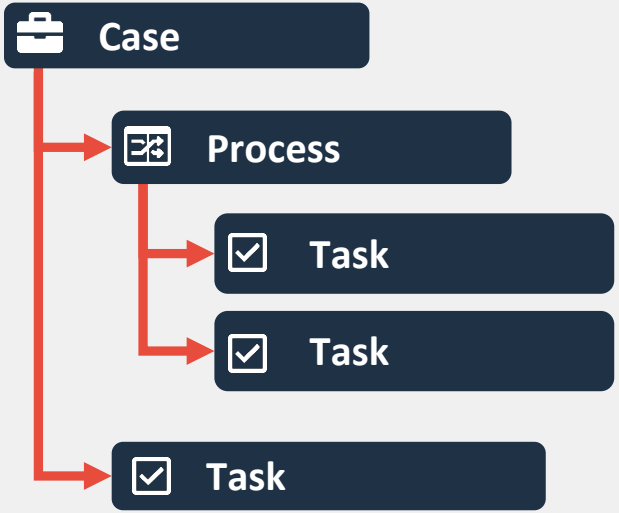


Permission Hierarchy: Downwards

Access propagation:

- Starter User
- Assignee User
- Owner User
- Participant Users

If such a user has access to a parent instance then the user will also have access to the child instance, and the children of the child instances.



Note:
Candidate users and groups have only access to a particular instance.

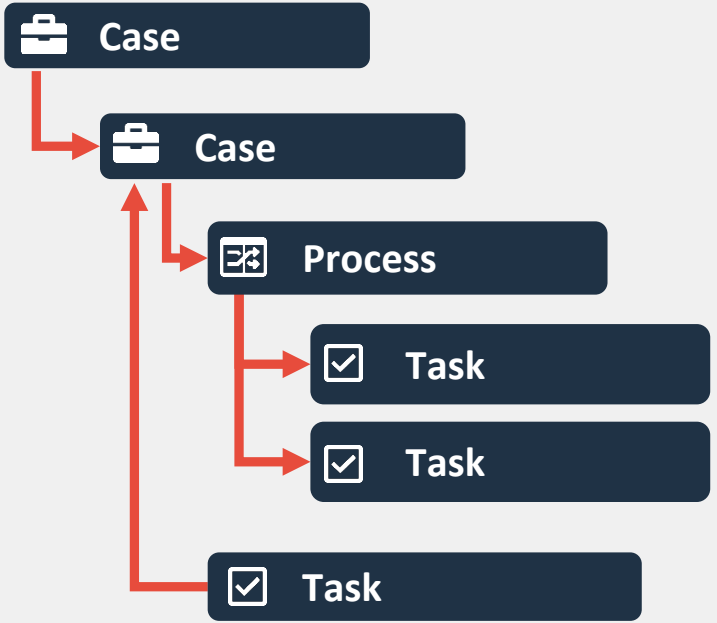
Permission Hierarchy: Upwards

Access propagation:

- Assignee User
- Owner User
- Participant User

When a user is involved within a task then the user will also become a participant of the parent of the task.

However, it will not gain access to the parent of the parent.



Permissions Example: Onboarding

Identity links in the context of a concrete case

(Case) Owner: HR Manager who starts onboarding of new employee.

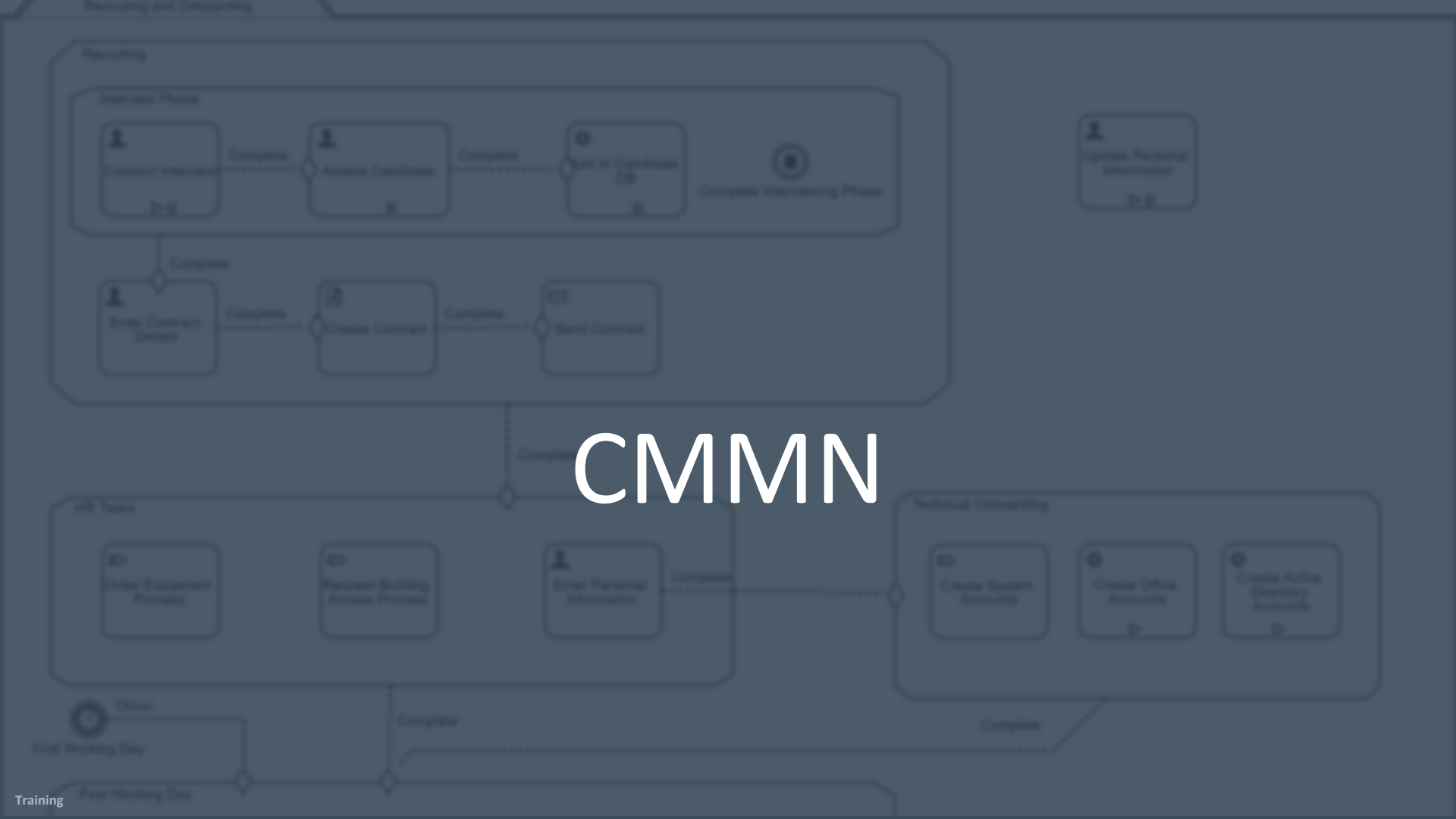
(Task) Assignee: The line manager of the new employee is the assignee of the «Organize Office Tour» task

(Task) Candidates: Two HR Managers responsible for the employee's team, both are candidates for the «Enter Compensation Info» task.

(Case) Participant Users: After they finished their tasks, the HR managers become participants with basic edit rights on the case.

(Task) Candidate Group: All employees belonging to the «Purchasing Department» group are candidates for the «Order Equipment Task».

(Case) Watcher: The boss of the HR department can see case, process and task information of the onboarding.


Recruiting and Onboarding
Recruiting
Interview Phase
Conduct Interview
Complete
Assess Candidate
Complete
Add to Candidate DB
Complete Interviewing Phase
Update Personal Information
Complete
Enter Contract Details
Complete
Create Contract
Complete
Send Contract
HR Tasks
Order Equipment Process
Request Building Access Process
Enter Personal Information
Complete
Technical Onboarding
Create System Accounts
Create Office Accounts
Create Active Directory Accounts
Complete
Once
First Working Day
Complete
First Working Day

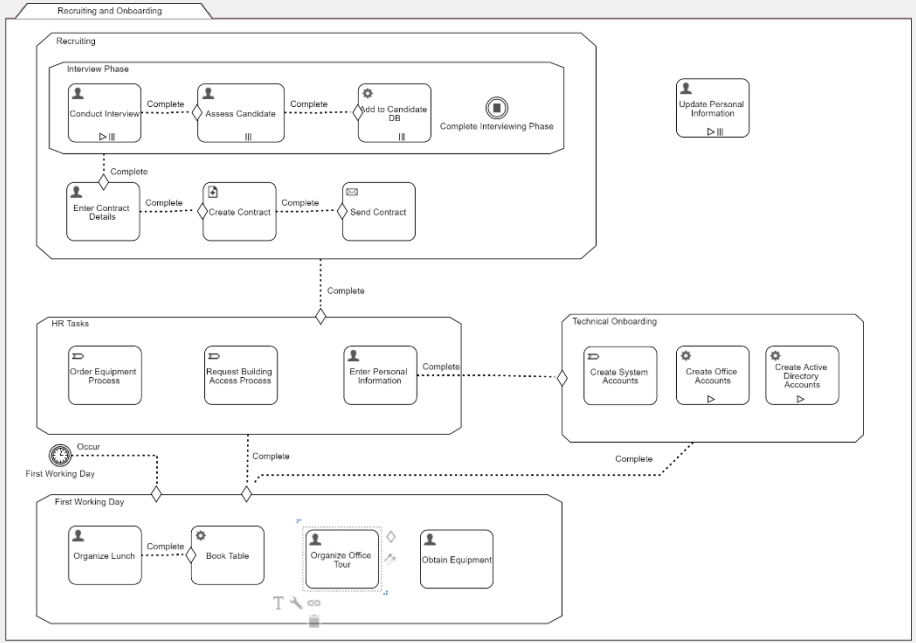
CMMN

CMMN in a Nutshell

Modeling Cases

- CMMN is used to model **cases**
- There are no **Start** and **End Events**, instead, the state of the whole model is taken into account
- **Plan Items** define your case:
  - **Tasks** define what is happening in a case
  - **Event Listeners** react to an external event
  - **Milestones** define that a certain state was achieved
  - **Stages** are groups of above elements
- Each **Plan Item** has a **state** and can **transition**
- **Sentries** define criteria to enable (enter) or terminate (exit)
- **Connectors** are used to connect PlanItems.

Example CMMN Case

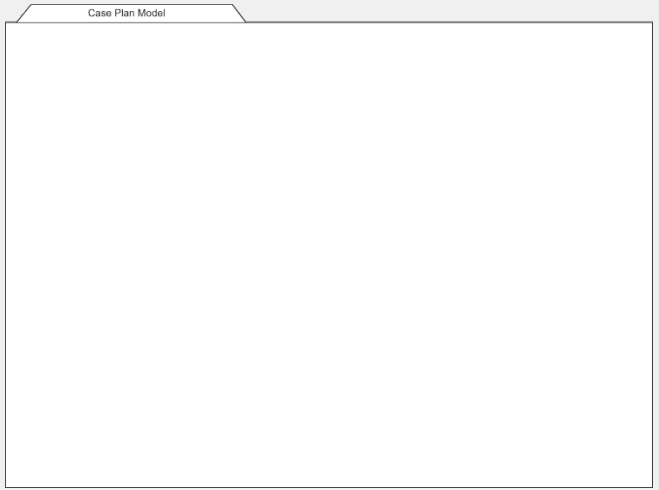


To the left, you see a case with:

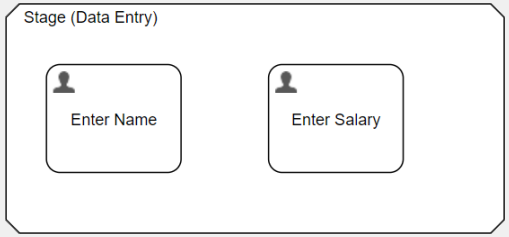
- Many **Plan Items**
  - 5 **Stages**
  - 17 **Tasks**
- A **(Timer) Event**
- **Sentries** that check whether tasks and stages can be activated
- **Connectors** that connect PlanItems with each other

Case Plan Model

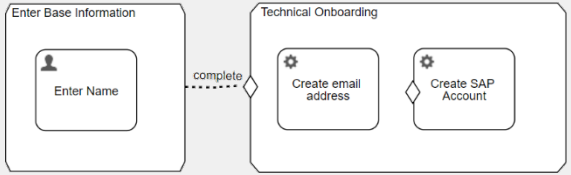
- The only required part of a case modeled in CMMN is the «**Case Plan Model**».
- The Case Plan Model represents the case.
- It essentially acts as the outermost stage.
- Once all case plan items in a case plan model are terminated or completed, **the case is completed**.



Stages

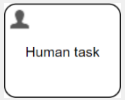


- **Stages** group related task and have a state, e.g. active, enabled, disabled etc.
- Like other Plan Items, stages can be connected through **Sentries**.

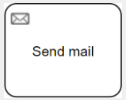


- If all Plan Items (Tasks or other stages) in a stage are **completed** or **terminated**, the stage is completed.

Important CMMN Tasks

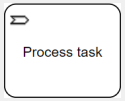


Human Task: Prompts a form to the user. This is the most common way to interact with users.

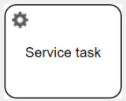


Email Task: An email containing arbitrary content will be sent a receiver.

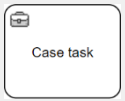
Rule Task: Stores variables based on rules defined in a DMN table.



Process Task: Starts a new BPMN Process.



Service Task: Calls Java Logic in the backend and optionally stores the result. Used to interact with other systems, make calculations etc.

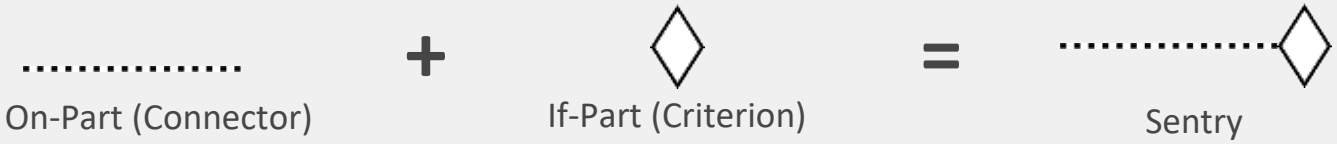


Case Task: Starts a new CMMN Case.

Sentries: Theory 1/2

- You can only connect plan items through sentries.
  They control the **flow** of your case!

- A **Sentry** consists of two parts:
  - The **On-Part** specifies the **trigger** of the condition
  - The **If-Part** specifies a **condition**

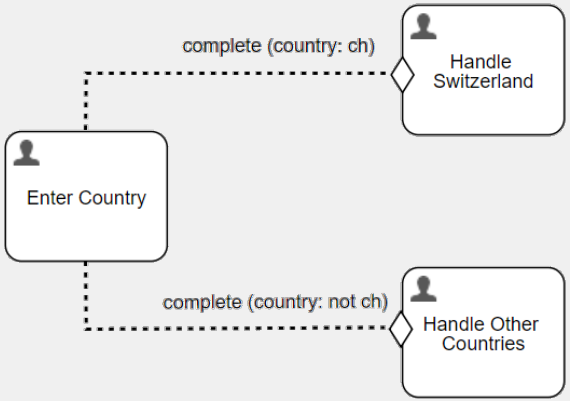


Sentries: Theory 2/2

- The On-Part is **optional**
  - There are many **event types**, e.g. complete, start, create or disable
- The If-Part is **mandatory**
  - Entry criteria (white diamond) **enable** planning items
  - Exit criteria (black diamond) **terminate** planning items

  - Use **Expressions** for conditions, e.g. `${var:equals(country, 'ch')}`
  - The conditions must evaluate to a **Boolean value**
  - If there is no condition, the If-Part evaluates to the Boolean value **`true`**

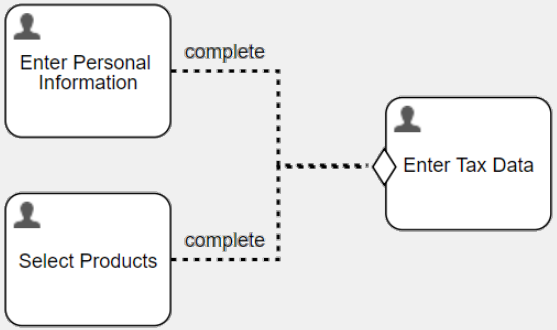
Sentry Patterns: OR and AND



Task «*Handle Switzerland*» is started if the country is «ch», **otherwise** «*Handle Other Countries*» is started.

On-Part: «Enter Country» Task was completed

If-Part: `${caseInstance.getVariable(country) == 'ch'}`



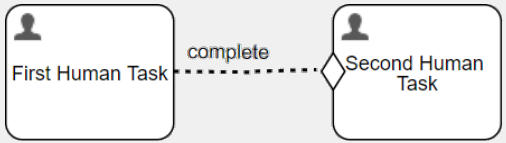
Task «*Enter Tax data*» is started once «*Select Products*» **and** «*Enter Personal Information*» was completed.

On-Part: «*Select Products*» **and** «*Enter Personal Information*» Tasks were completed

If-Part: None

Basic CMMN Flow

- In CMMN, Plan Items have a state. The flow is dependent on **state changes**!
- For example, whenever a **variable changes**, or a **task is completed**, the model will be **re-evaluated**. This means that each sentry is **checked**.
- If all their conditions are **satisfied** they will **activate** or **terminate** their plan item. Plan Items without sentry will simply be activated.

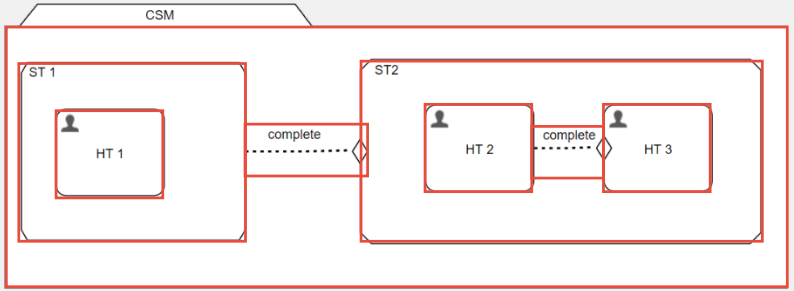


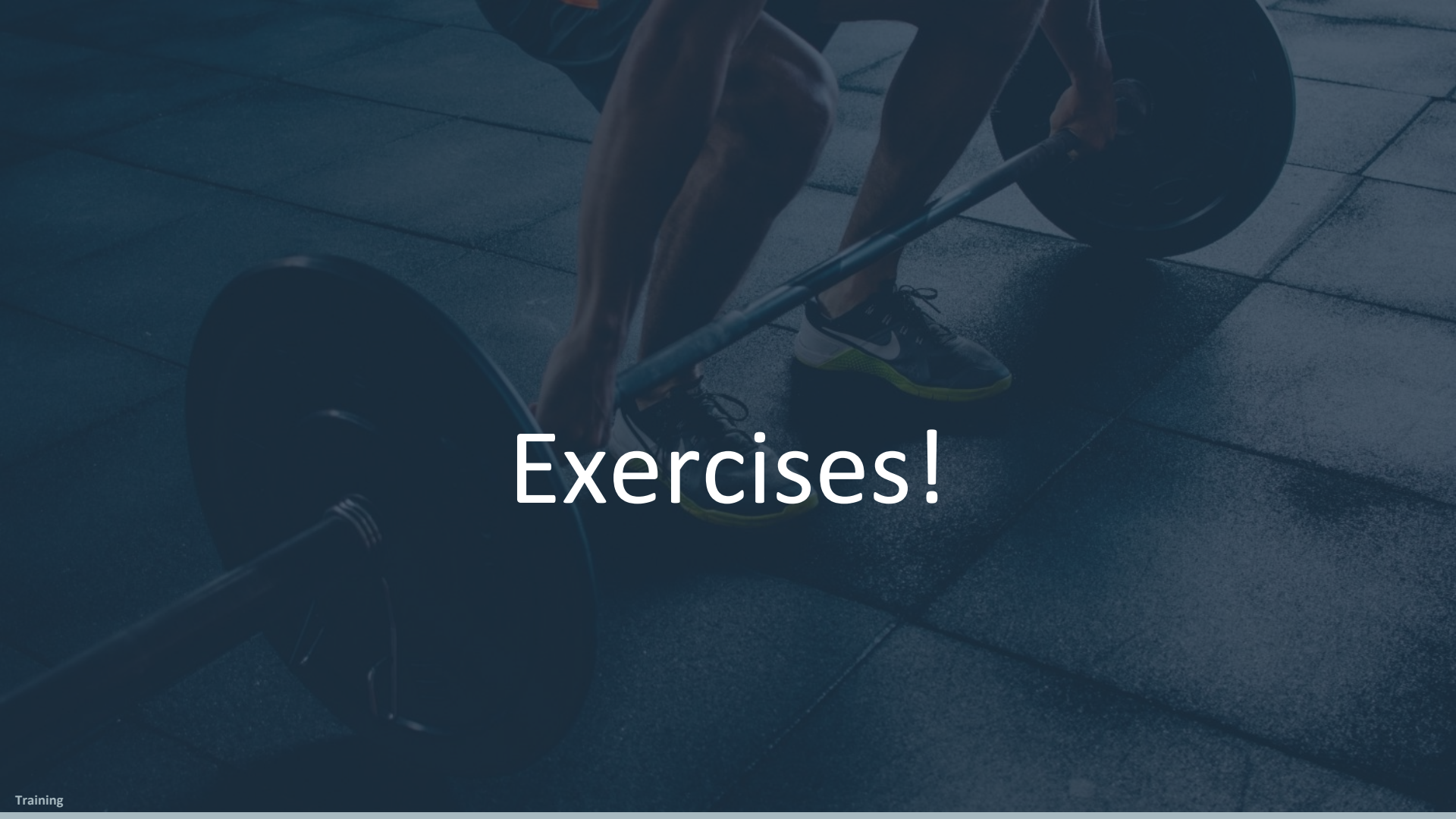
Simple Example:

- First Human Task is completed
- Model re-evaluates
- *On-Part* of the following sentry is satisfied
- *If-Part* of the following sentry is satisfied
- Second Human Task is activated

Basic CMMN Flow Example

- Start Case (CSM): → Activate
  - ST 1: No sentry → Activate
  - HT 1: No sentry → Activate
- Complete HT 1 → Fire «Complete» event
  - ST 1: No more items → Fire «Complete» event
  - ST 2: Sentry satisfied → Activate
  - HT 2: No sentry → Activate
- Complete HT 2: → Fire «Complete» event
  - HT 3: Sentry satisfied → Activate
- Complete HT 3: → Fire «Complete» event
  - ST 2: No more items → Fire «Complete» event
  - CSM: No more items → Fire «Complete» event




Exercises!

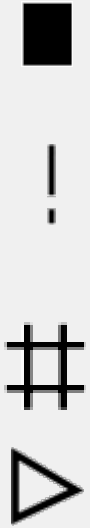
Simple Exercise

- Help me organise the next trainings!
- Model a “Training” case with three stages:
  - Negotiation:
    - First we decide the dates and then we confirm the training
  - Preparation:
    - I need to prepare the material and the environments
  - Execution:
    - I execute the training and then I send a survey

Plan Item Decorators

Every Plan Item can have one or more **decorators** which modify the way the model works. They can be combined and can be evaluated dynamically.

- **Auto Complete**: Stages close as soon as all **active**, and **required** Tasks are **completed** or **terminated**.
- **Required**: Plan Item must be completed in **auto complete** stages.
- **Repetition**: Plan Item can be activated more than once.
- **Manual Activation**: Plan Item is not automatically activated.



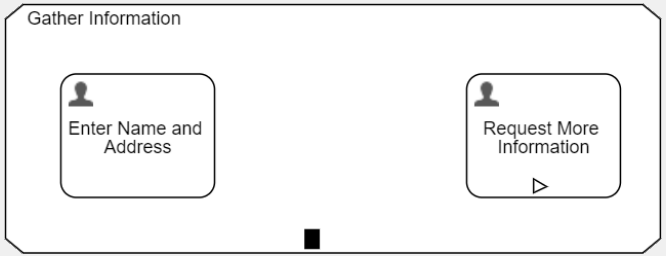
For applicability rules, see Table 6.1 of the standard.

Decorator: Manual Activation

- By default, Plan Items are **immediately activated** once their sentry's criteria are satisfied or if there is no sentry.
- «Manual Activation» defers that process. The Plan Item is then activated **on demand**, e.g. through a button.
- This adds flexibility and allows to add optionality!
- Example: Only compensate the client when the user decides to do so.

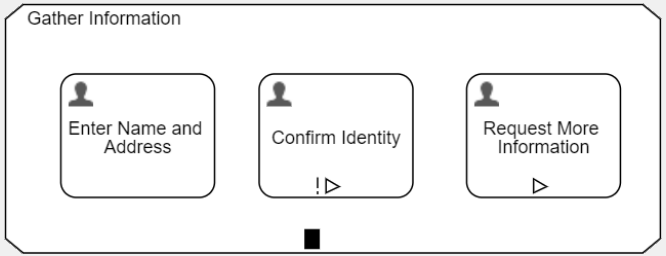
Decorator: Auto Complete

- **Auto Complete**: Stages close as soon as all **active**, and required Tasks are **completed** or **terminated**.
- In combination with «Manual Activation», they allow you to model optionality.
- Example: The first task is activated directly. The second task is optional. Once the first task is completed, the stage is closed.



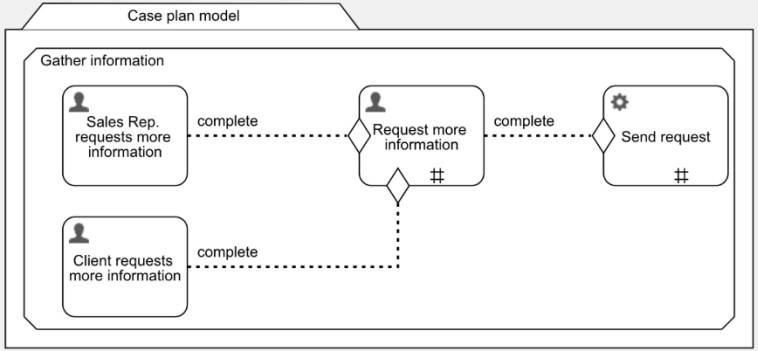
Decorator: Required

- Ensures that the Plan Item is **completed** or **terminated** even when the surrounding stage is «**Auto Complete**».
- Also marks the Plan Item as «required» which can be helpful when building **graphical applications**.
- Example: The first task is activated directly, the second one later on. Both tasks must be completed. The third task is optional.



Decorator: Repetition

- Allows a plan item to be activated **more than once**.
- Usually, there is a **condition** tied to it. When the condition evaluates to true, a new instance will be created
- Example: «Request More Information» and «Send Request» can be activated twice through two different Human Tasks.



Events

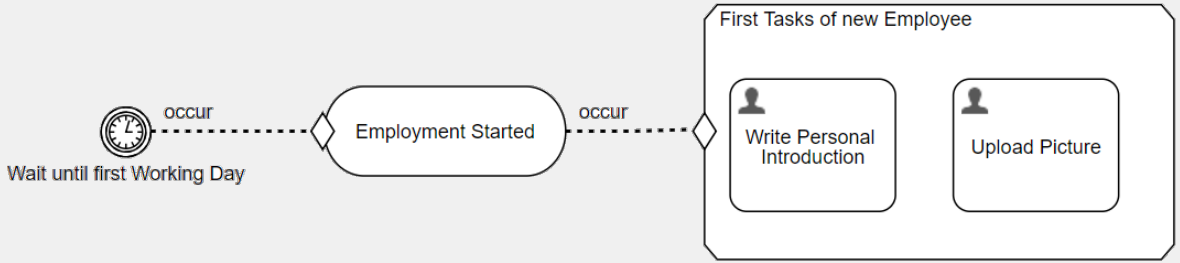
Events listen for outside triggers.
They can **activate** and **exit** plan items through sentries

- **User Event Listeners** listen for a user's input, e.g. the click of a button in the frontend.

- **Timer Events** wait for a certain duration or until a point in time.

Milestones

- **Milestones** indicate that some state has been achieved.
- They do **not** have any runtime behaviour but act as a normal plan item. This means that they can for instance block the execution etc.
- Use them to enhance the clarity of your model.

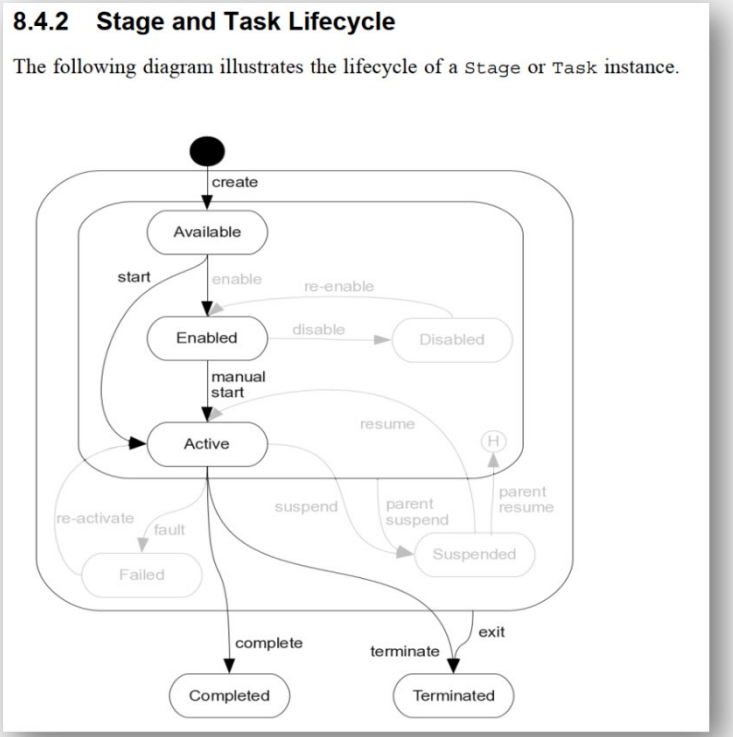


How to slice a case

Cases can be complex beasts.

- Cases often offer a **high-level overview**.
  If your case becomes too complex, redesign it!
- They trigger smaller, well-defined steps through **Process Tasks**.
- Task chains of **more than two elements** are good candidates for a process.
- Move **loops** and **complex decisions** to processes as well.
- Don’t shy away from **sub cases**!

A Word about State…



- There are a lot of **states** and **transitions** – but don't be intimidated.
- The most important states are «**Active**», «**Terminated**» and «**Completed**».
- Check the standard (chapter 8.4.2 and others) for status state diagrams!

Process or Case?

As always: It depends!

- Cases provide a **high-level overview** and start further processes. They often represent your **business data**.
- In principle, **every process** can be represented as case.
- Cases usually involve **repetition** and **optionality**.
- **Simple** sequences and **event-dependent** flows are candidates for processes.

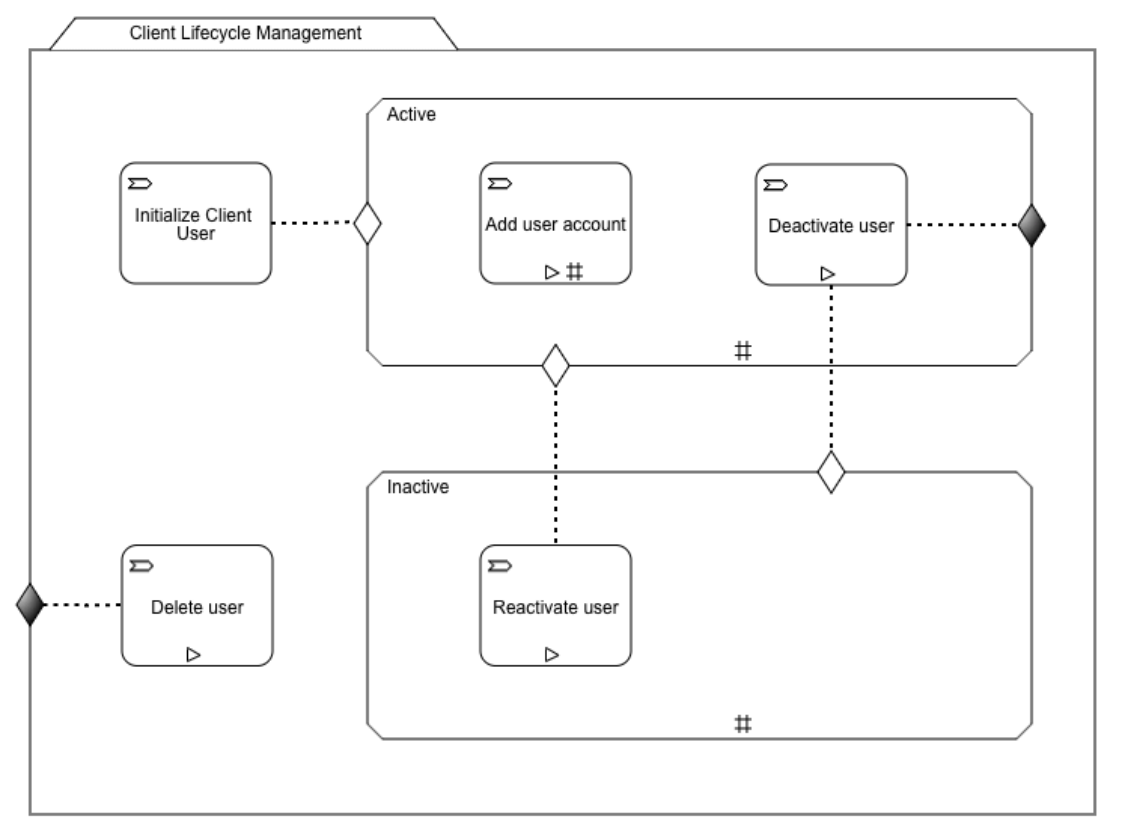
Process vs Cases

static

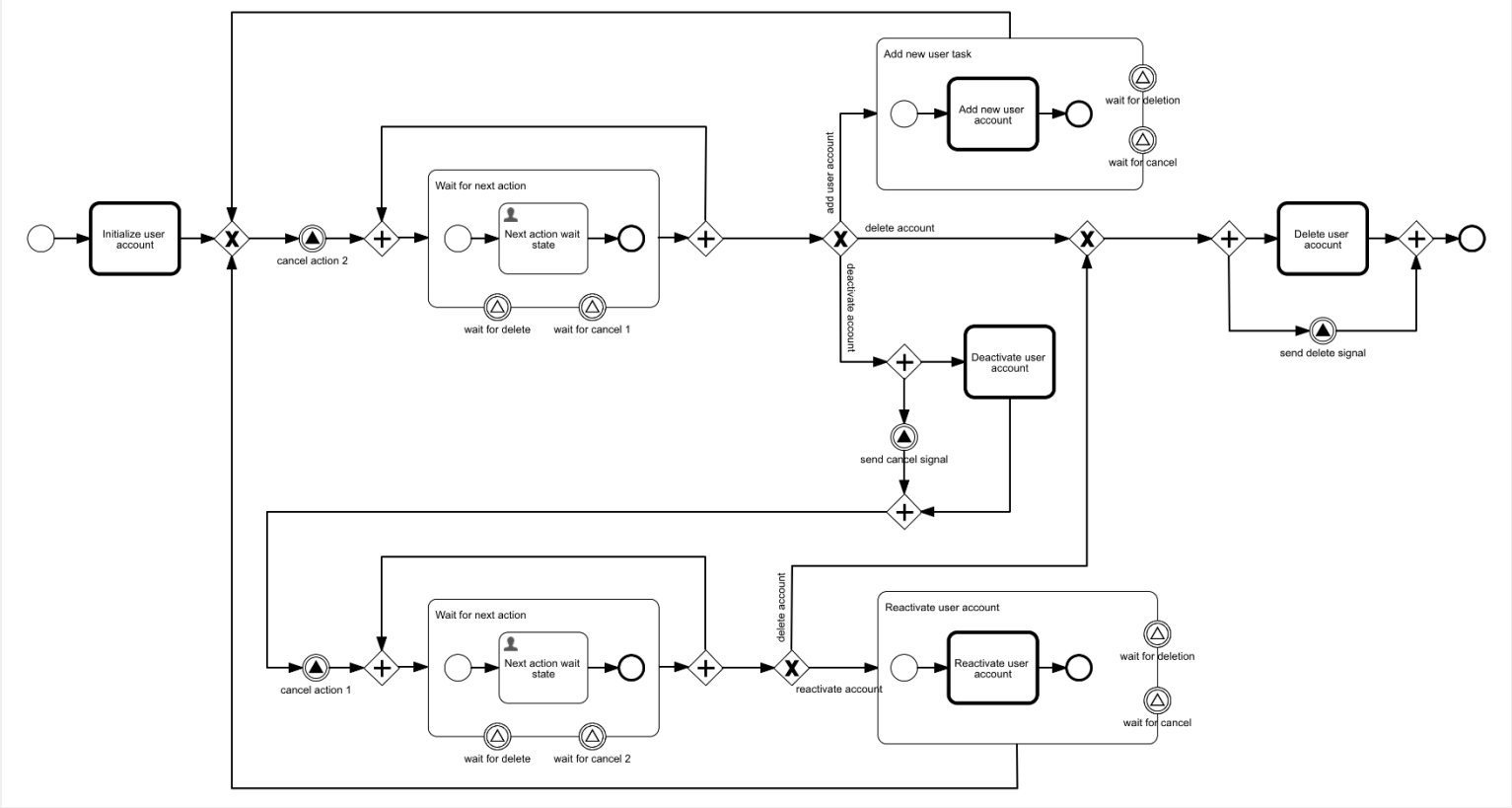
dynamic

- Cases provide a **high-level overview**. Every element is “business-relevant”.
  Processes can be **purely technical**.
- Processes define a clear **control flow**.
  Cases are more **reactive**.
- **Repetition**, **optionality** and **ad-hoc actions** are easy to implement in CMMN.
  Processes offer a number of **events**.
- Do not choose one over the other, combine them!
  It’s a matter of how dynamic your use case is!

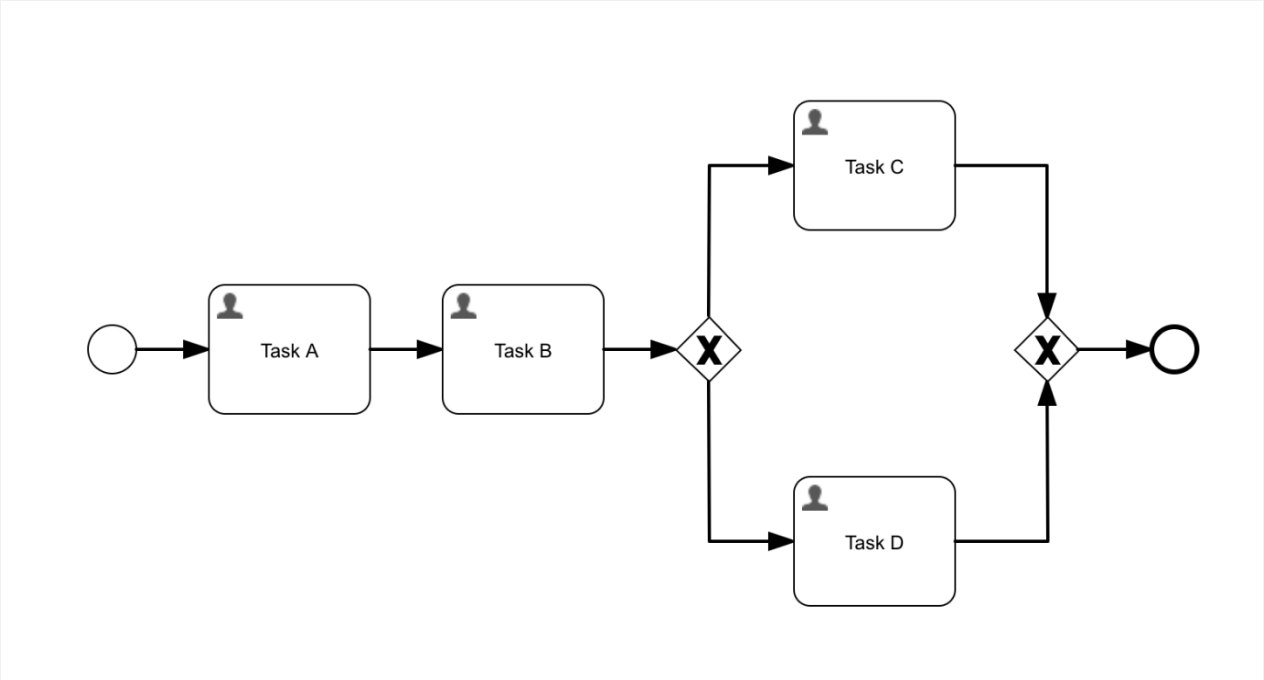
Example as a Case



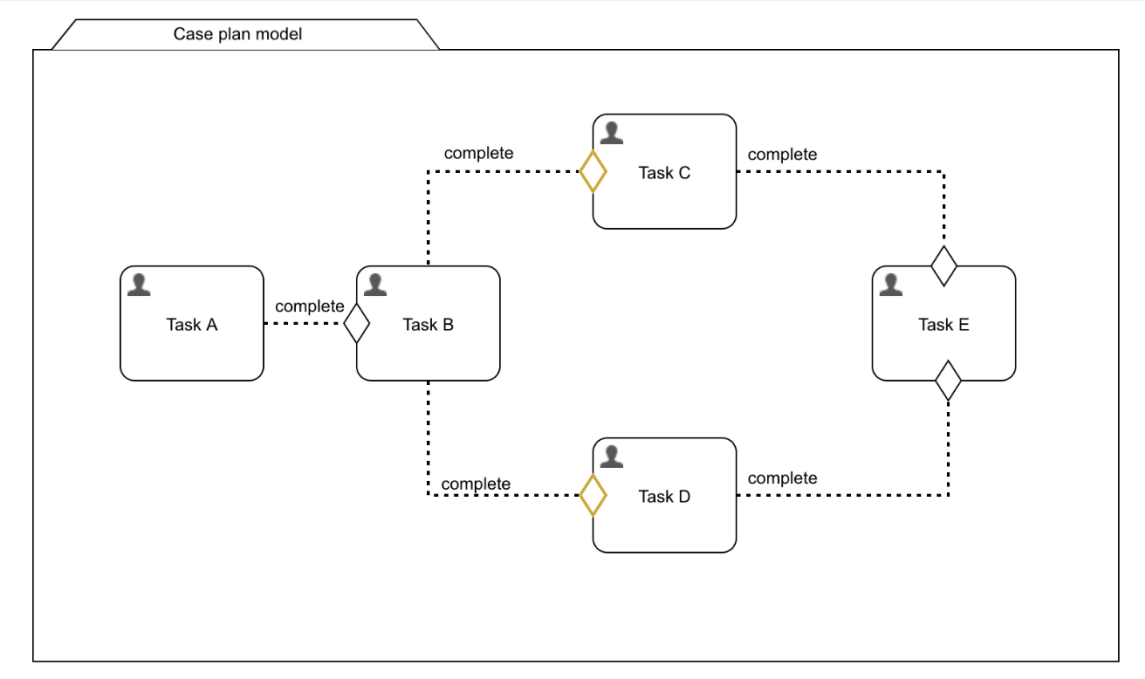
Same Example as a Process



Simple Process Flow



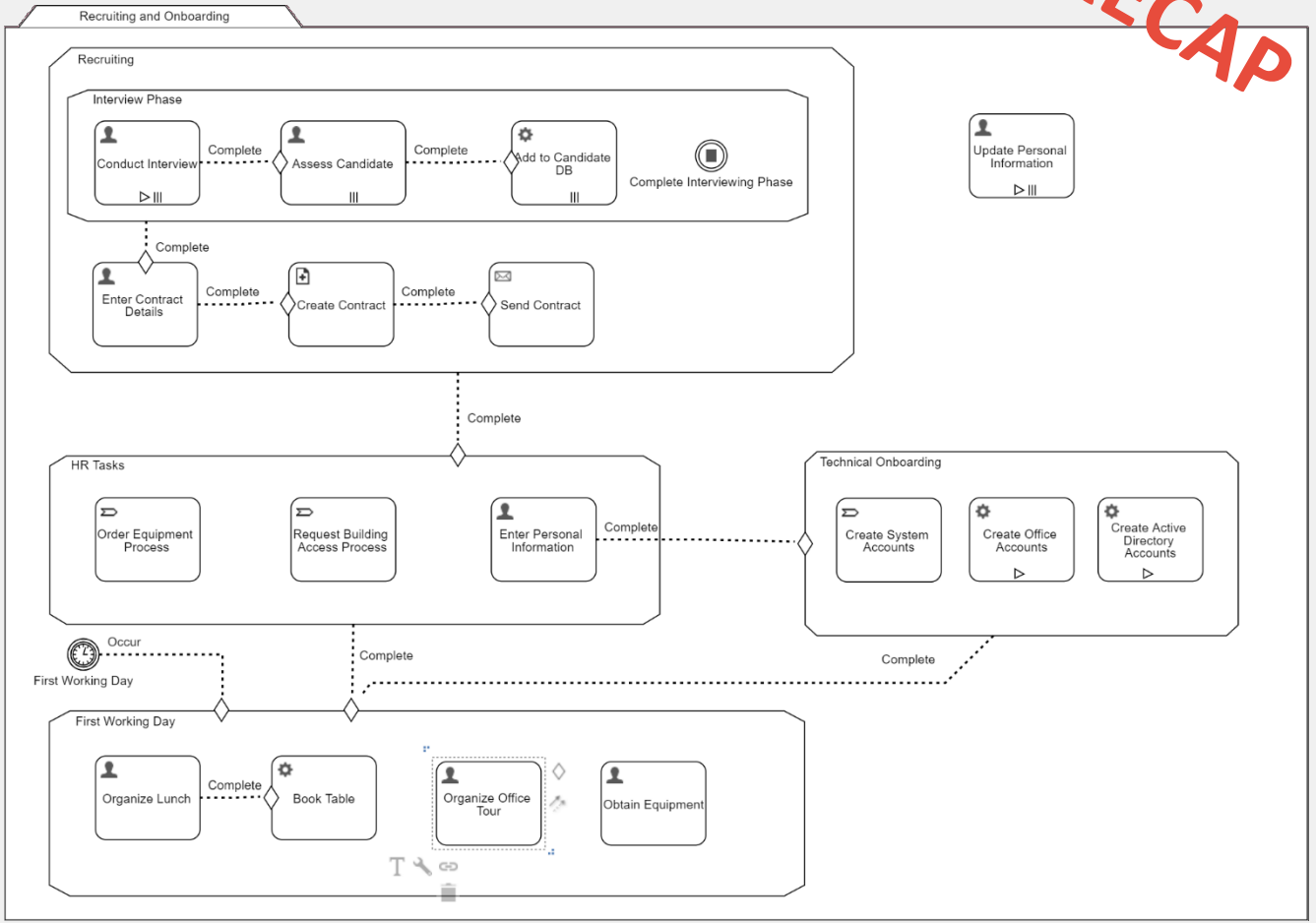
Same flow as Case



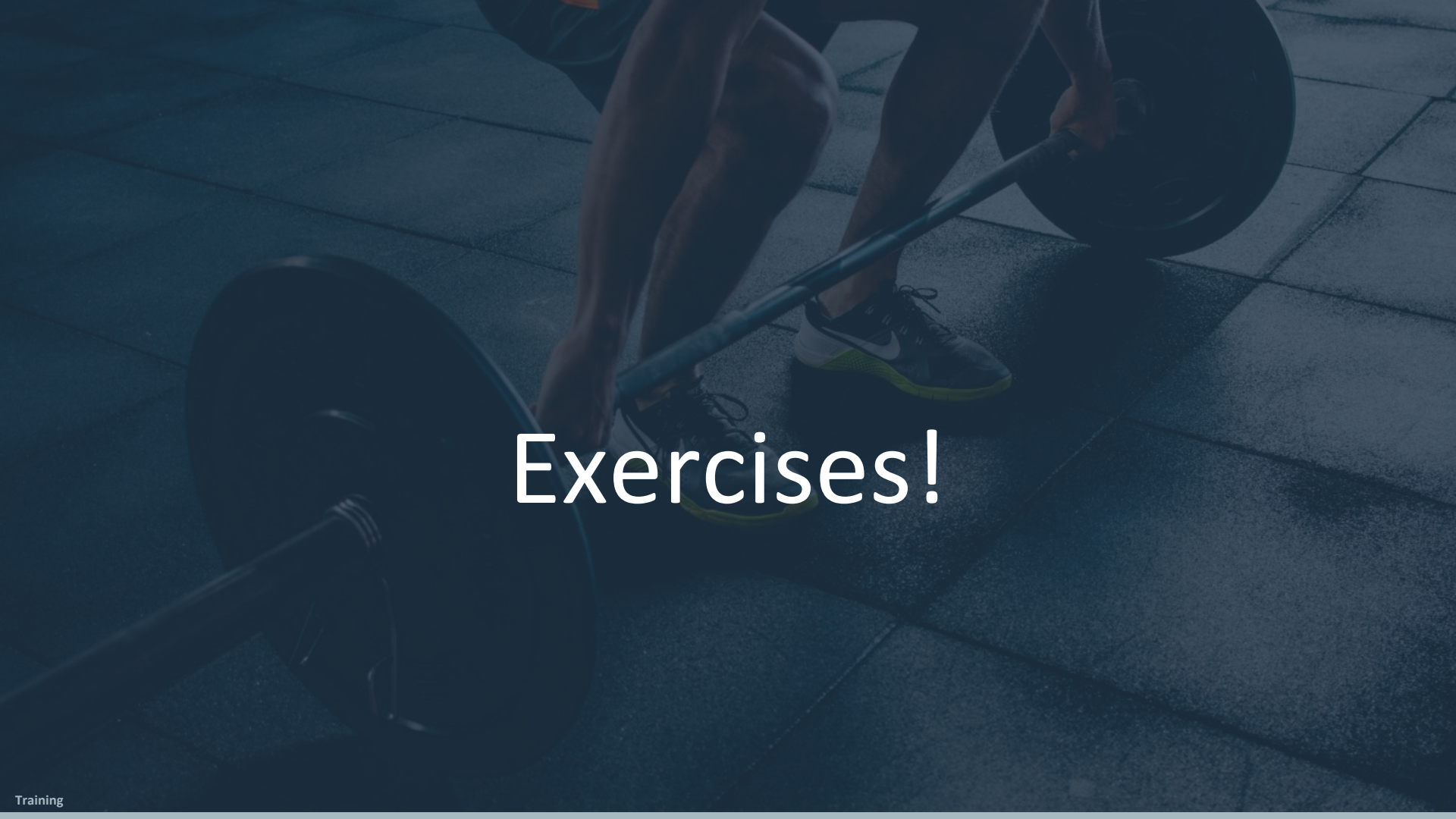
How to model a case?

1. Define Scope of Case
2. Set up stages/phases in a temporal order
3. Define tasks and rough flow
4. Refine, fill in tasks add attributes and add further logic

RECAP




Be creative,
try it yourself!


Exercises!

Exercise 2

- During the negotiation we can change the dates several times. We also need a checkbox for “Environment needed?”
- The environments setup task depend on the value of that variable.
- After the training execution task we need to send the certificates
- We need to be able to set up a call anytime during the case.
- We need to be able to cancel the case anytime during the case (UserEvent)


Risk Routing
Risk Category
riskCategory
string
[Low,Medium,High]
Age
clientAge
number
1
2 == High
3 == Medium
4 == Medium
5
18
-
25
25
-

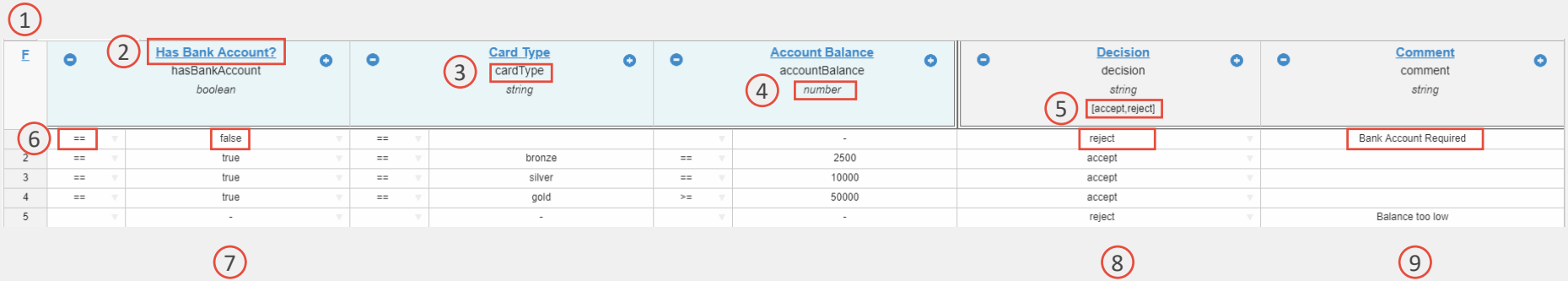
DMN

DMN in a Nutshell

Modeling Business Rules

- DMN allows you to define **Business Rules** in the form of **decision tables**
- You can define **input** and **output columns**
- Columns are bound to process or case **variables**
- There are different ways to evaluate the tables called **Hit Policies**
- There are “**Single**” and “**Multi**” Hit Policies which either return one or more results.

Decision Table Overview



1 F	2 Has Bank Account? hasBankAccount *boolean*		3 Card Type cardType *string*		4 Account Balance accountBalance *number*		5 Decision decision *string* [accept,reject]	Comment comment *string*
6	==	false	==			-	reject	Bank Account Required
2	==	true	==	bronze	==	2500	accept	
3	==	true	==	silver	==	10000	accept	
4	==	true	==	gold	>=	50000	accept	
5		-		-		-	reject	Balance too low

7 8 9

1. Hit Policy
2. Column Name
3. Column Variable
4. Column Variable Type
5. (possible) Output Values
6. Row Operators
7. Row Value
8. Restricted Output Value
9. Optionally empty Output Value

Input / Output Columns

- Multiple **inputs** and **outputs** can be defined.

Input

- **Types**: String, Number, Boolean, Date, Collection.
- Possibility to define **allowed values**, e.g. “accept” and “reject”.

Output

- **Types**: String, Number, Boolean, Date.
- Possibility to define priority with “output values” order (only relevant for hit policy “Priority” and “Output Order”).

Hit Policy: First (Single)

- **First** hit is returned - order matters!
- Multiple rules can match (with different output entries).
- Easy rule design but considered **not good practice**.

Determine Ag···

F	Age age *number*	Age Category ageCategory *string*
1	<= 3	Baby
2	<= 14	Child
3	< 20	Teenager
4	<= 62	Adult
5	> 62	Senior

Hit Policy: First (Single)

- **First** hit is returned - order matters!
- Multiple rules can match (with different output entries).
- Easy rule design but considered **not good practice**.

FIRST	Age `{age} (number)`	Age Category `{ageCategory} (string)`
	19	Teenager
1 ✗	<= 3	"Baby"
2 ✗	<= 14	"Child"
3 ✓	< 20	"Teenager"
4	<= 62	"Adult"
5	> 62	"Senior"

Hit Policy: Unique (Single)

- Only **single rule** matches.
- **No overlap** is possible.
- If there is more than match, an **error** will be thrown.

Calculate Prize V···

U	Rank rank *number*		Rank rank *number*		Prize prize *number*
1	==	1	==	1	10000
2	==	2	==	2	3000
3	==	3	==	3	1000
4	>=	4	<	10	200
5	>=	10		-	50

Hit Policy: Unique (Single)

- Only **single rule** matches.
- **No overlap** is possible.
- If there is more than match, an **error** will be thrown.

UNIQUE	Rank `{rank}` (number)	Rank `{rank}` (number)	Prize `{prize}` (number)
	6	6	200
1 ✗	== 1	== 1	10000
2 ✗	== 2	== 2	3000
3 ✗	== 3	== 3	1000
4 ✓	>= 4	< 10	200
5 ✗	>= 10	-	50

Hit Policy: Any (Single)

- **Multiple rules** can match (but must have equal output entries).
- If the output entries are non-equal, an **error** will be thrown.
- Points out **inconsistencies** in the rules.

Risk Routing

A	Risk Category riskCategory *string* [Low,Medium,High]		Age age *number*		Routing routing *string* [Decline,Further evaluati…
1		-	<=	18	Decline
2	==	High		-	Decline
3	==	Medium	<=	25	Further evaluation
4	==	Medium	>	25	Accept

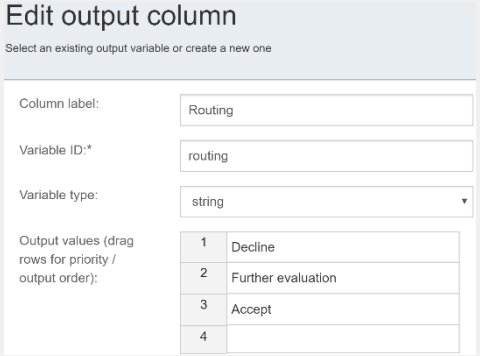
Hit Policy: Any (Single)

- **Multiple rules** can match (but must have equal output entries).
- If the output entries are non-equal, an **error** will be thrown.
- Points out **inconsistencies** in the rules.

ANY	Risk Category `{riskCategory} (string)`	Age `{clientAge} (number)`	Routing `{routing} (string)`
	High	17	Decline
1 ✓	-	<= 18	"Decline"
2 ✓	== "High"	-	"Decline"
3 ✗	== "Medium"	<= 25	"Further evaluation"
4 ✗	== "Medium"	> 25	"Accept"

Hit Policy: Priority (Single)

- **Multiple rules** can match (with different outputs)
- Entry with the **highest priority** will be match
- Priority is defined by the order of **Output Values**



Risk Routing

P	Risk Category riskCategory *string* [Low,Medium,High]		Age clientAge *number*		Routing routing *string* [Decline,Further evaluati…
1	==	Medium	<=	25	Further evaluation
2	==	Medium	>	25	Accept
3		-	<=	18	Decline
4	==	High		-	Decline
5		-		-	

Hit Policy: Priority (Single)

- **Multiple rules** can match (with different outputs)
- Entry with the **highest priority** will be match
- Priority is defined by the order of **Output Values**

PRIORITY	Risk Category `{riskCategory} (string)`	Age `{clientAge} (number)`	Routing `{routing} (string)`
	Medium	17	Decline
1 ✓	== "Medium"	<= 25	"Further evaluation"
2 ✗	== "Medium"	> 25	"Accept"
3 ✓	-	<= 18	"Decline"
4 ✗	== "High"	-	"Decline"

Hit Policy: Rule Order (Multiple)

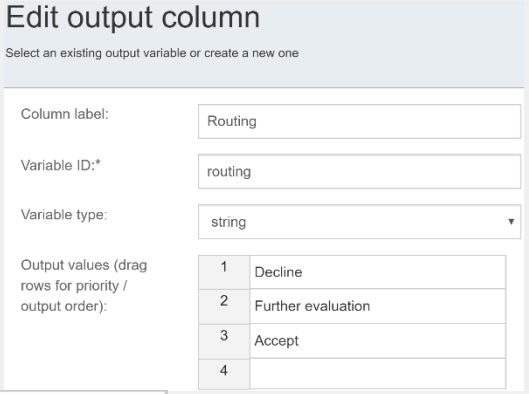
- **Multiple rules** can match (with different outputs).
- Simply returns all hits in **rule order**.
- The results will be stored in a **list**.

Available Movies···

R	Age age *number*		Movie movie *string*
1	>=	7	Despicable Me 3
2	>=	7	Ferdinand
3	>=	13	Inception
4	>=	13	Forrest Gump
5	>=	16	Deadpool 2

Hit Policy: Output Order (Multiple)

- Returns all hits in decreasing **output priority order**
- Based on **Output priorities**
- Useful when the output value matters.



Risk Routing

O	Risk Category riskCategory *string* [Low,Medium,High]		Age clientAge *number*		Routing routing *string* [Decline,Further evaluati…
1		-	<=	18	Decline
2	==	High		-	Decline
3	==	Medium	<=	25	Further evaluation
4	==	Medium	>	25	Accept

Hit Policy: Collect (Multiple)

- Can apply an **operator** on the matched output values.

- Sum (C+): Returns **sum** of all output values — Example: Shipping cost
- Min: (C<): Returns **lowest** output value — Example: Fee calculation
- Max (C>): Returns **highest** output value — Example: Priority
- Count (C#): Returns **number** of hits — Example: Eligible Products
- None: (C): Returns hits in **random** order

Calculate Discount

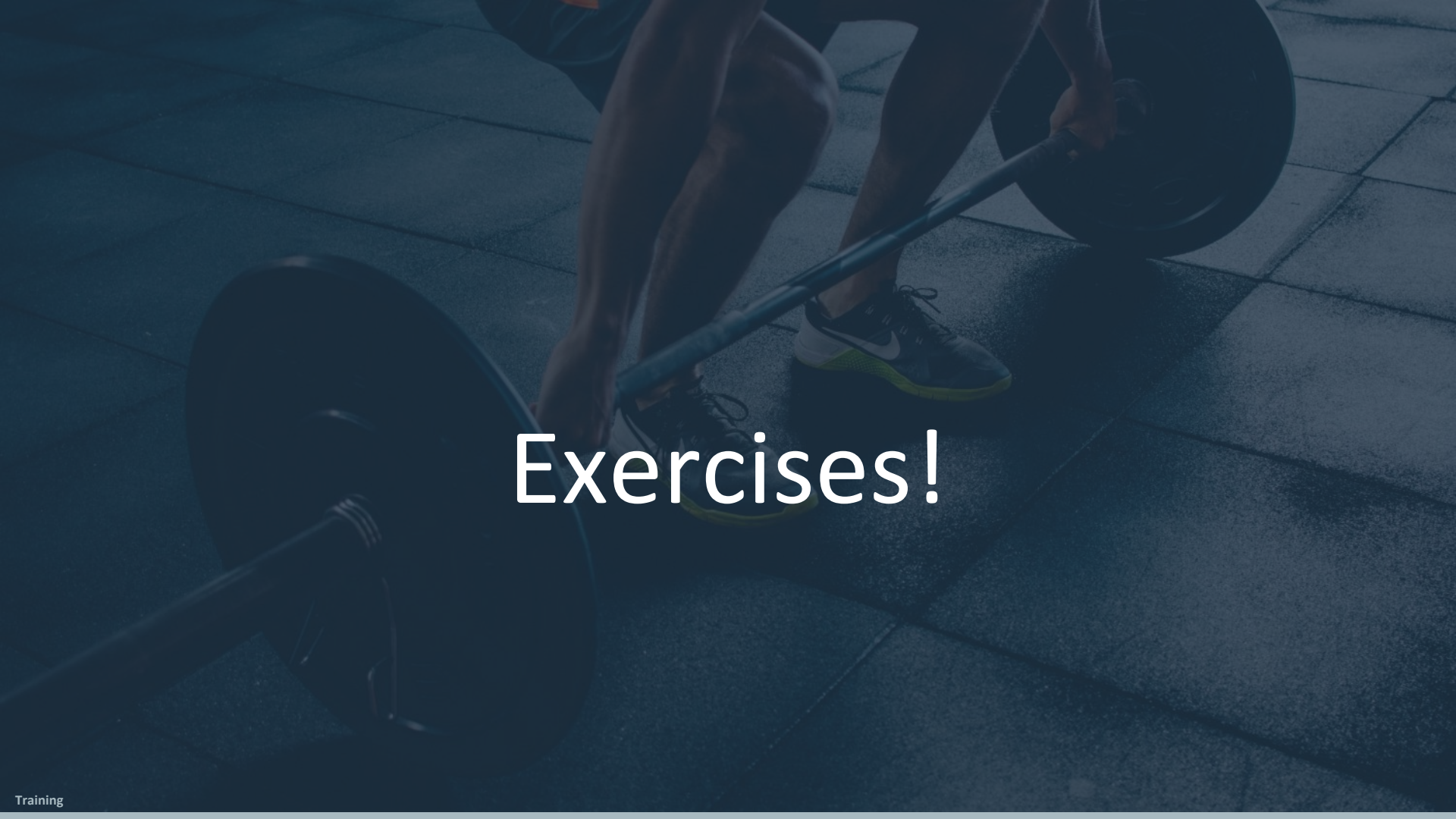
C>	Purchase Value purchaseValue *number*	Discount (in %) discount *number*
1	<= 50	0
2	<= 200	5
3	<= 5000	10
4	> 5000	15
5	-	

Hit Policy: Collect (Multiple)

- Applies an **operator** on the matched output values.
- Sum: Returns **sum** of all output values — Example: Shipping cost
- Min: Returns **lowest** output value — Example: Fee calculation
- Max: Returns **highest** output value — Example: Priority
- Count: Returns **number** of hits — Example: Eligible Products

COLLECT	Purchase Value `{purchaseValue} (number)`	Discount (in %) `{discount} (number)`
	5	10
1 ✓	<= 50	0
2 ✓	<= 200	5
3 ✓	<= 5000	10
4 ✗	> 5000	15


Let’s find business rules
on this mountain


Exercises!

Excercise

- Model this usecase:
  - A user creates an incident ticket: (form: subject(text) and severity(number))
  - Another user handles the ticket.
  - If the task is not completed, after a period of time, a escalation user task is created.

Severity	Time
1	PT5S
2	PT10S
3	PT15S


Flowable
making process personal