Process Modelling

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group

Table of Contents

Table of contents

Structure and Behaviour Modelling

- *Structural*
  - Static
  - Whole and part, components
  - Connections
- *Behavioural*
  - Dynamic
  - Timeliness
  - State, Process
  - Reaction to the environment (context)
- Modelling does not cover all aspects, aspects cannot be separated…

The main components of the robot vacuum cleaner are the control unit, the roller gear and the vacuum cleaner.

For the command „to right” changes the roller gear its operational mode to „turn”.

Main Questions of the Behavioural Models

- What the system „does”?

$\Longrightarrow$ Event based model

$\boxed{\text{Process based model}}$

…

- What are the properties of the system now, and how is it changing?

$\Longrightarrow$ State based models

Main Questions of the Behavioural Models

- State Based Approach
  - the system changes (its properties)
  - as a reaction to (external) events
  - input/output channels
- Process Based Approach
  - the system changes the *work item*
  - as a series of activities
  - data flow

Definition: Process

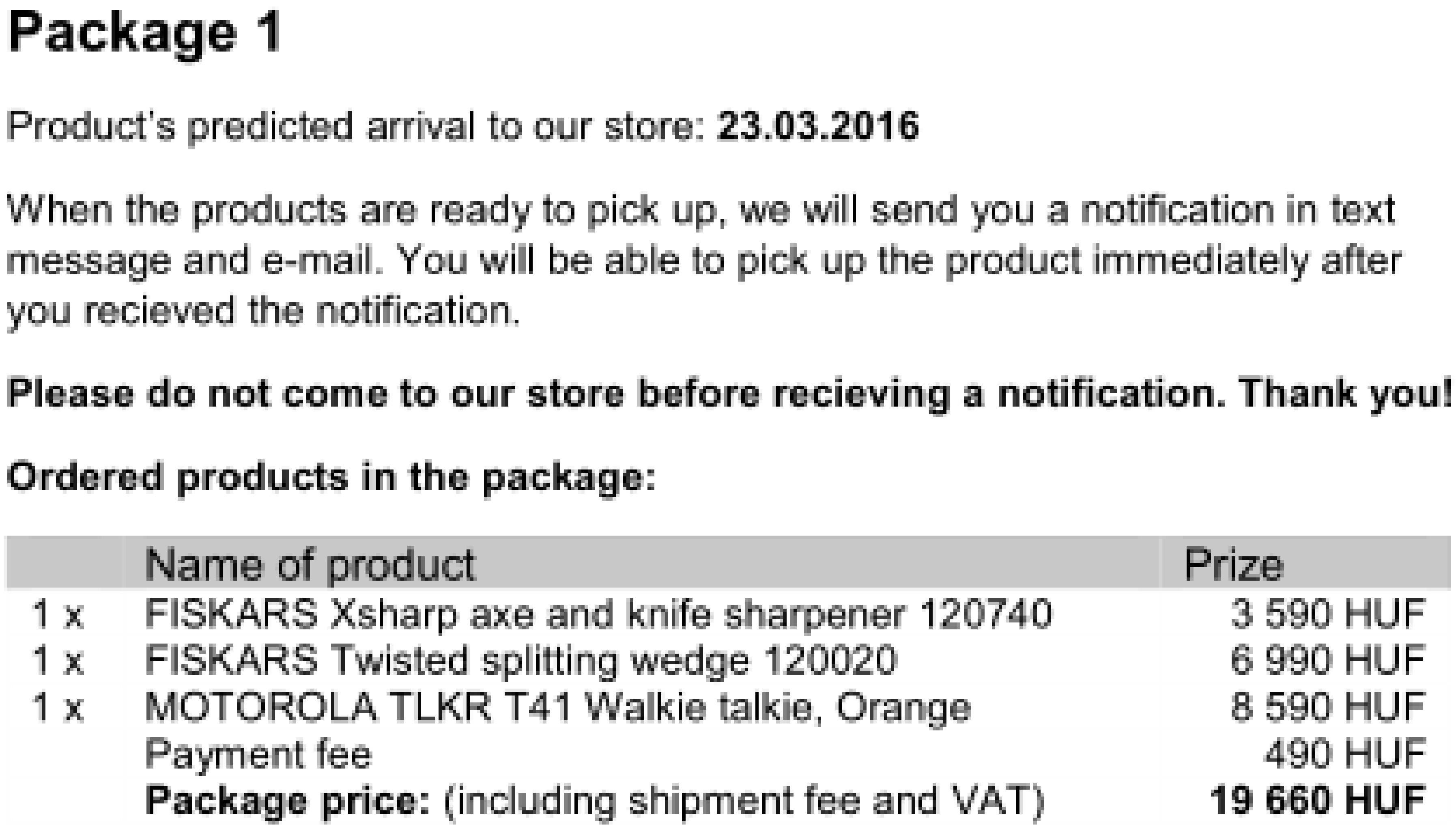
Process: series of steps that achieve purpose when executed in the right order

Table of Contents

Role of Process Modelling

- Specification
- Design
- Implementation
  - Executable models
  - Code generation
- Model verification
  - Simulation
  - Monitoring
  - Automated model checking
- Documentation

Example: How Does the Product Arrive?



Package 1

Product's predicted arrival to our store: **23.03.2016**

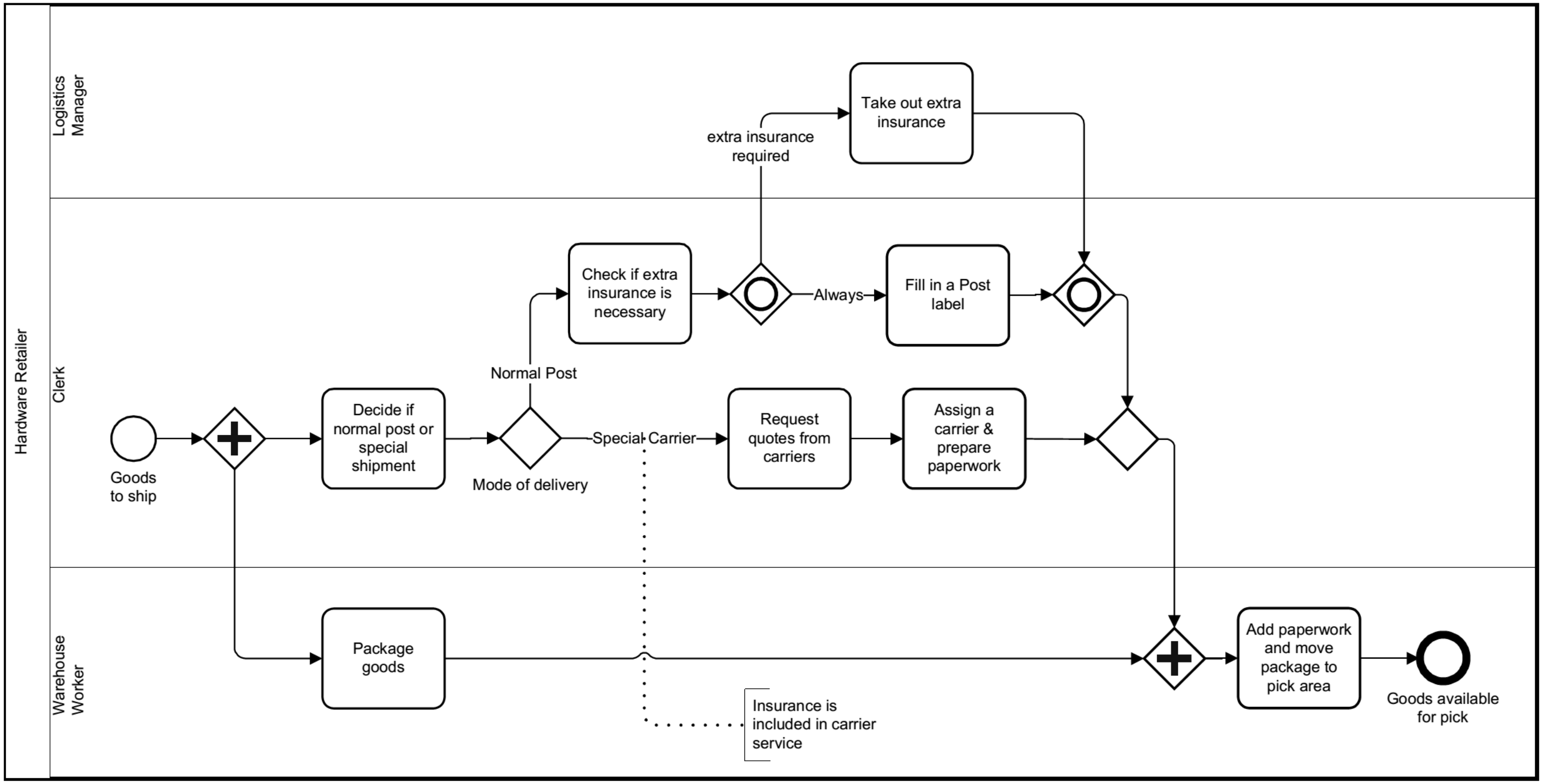
When the products are ready to pick up, we will send you a notification in text message and e-mail. You will be able to pick up the product immediately after you recieved the notification.

Please do not come to our store before recieving a notification. Thank you!

Ordered products in the package:

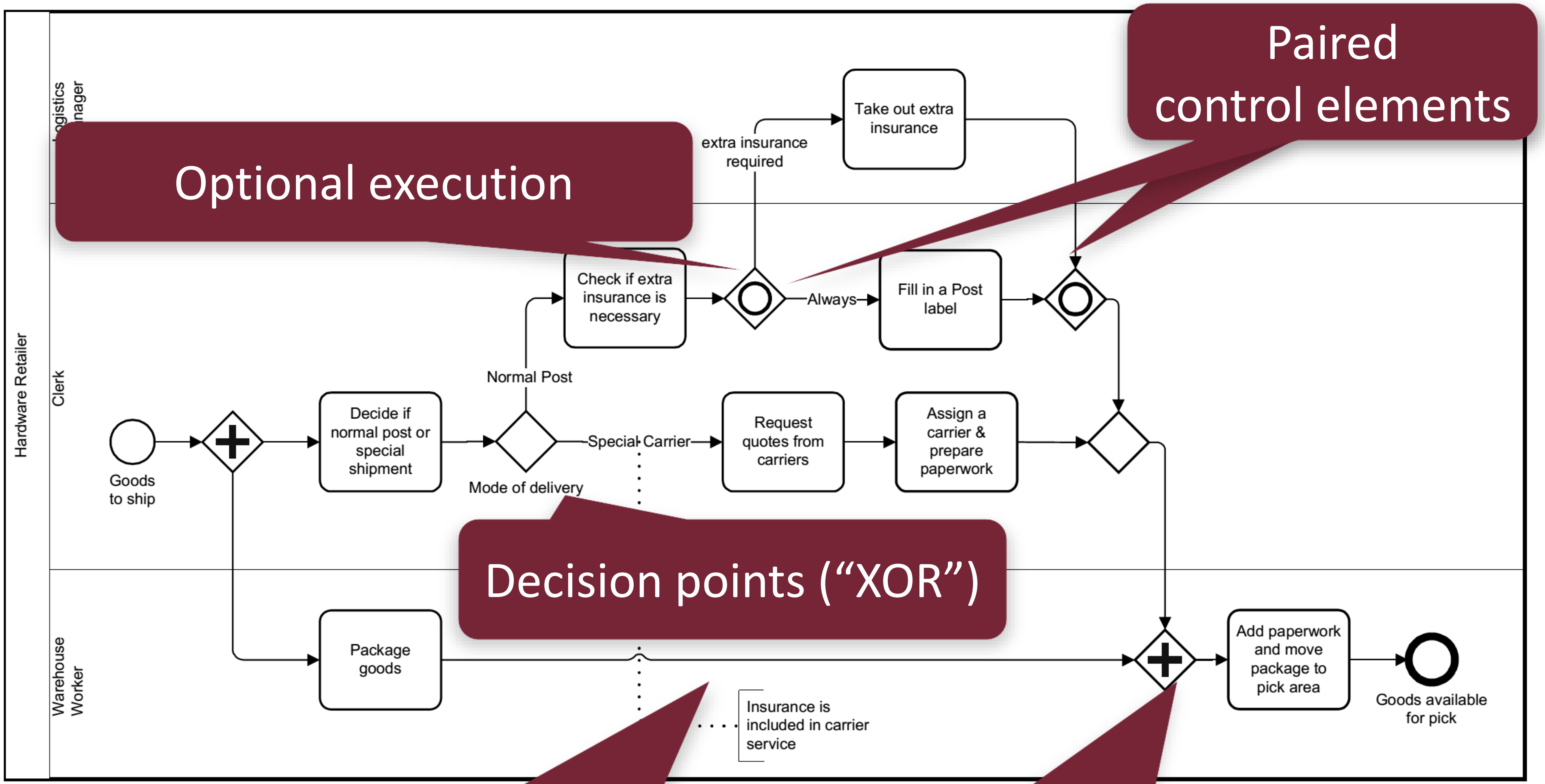
	Name of product	Prize
1 x	FISKARS Xsharp axe and knife sharpener 120740	3 590 HUF
1 x	FISKARS Twisted splitting wedge 120020	6 990 HUF
1 x	MOTOROLA TLKR T41 Walkie talkie, Orange	8 590 HUF
	Payment fee	490 HUF
	Package price: (including shipment fee and VAT)	**19 660 HUF**

Example: HW Delivery


Hardware Retailer
Logistics Manager
Clerk
Warehouse Worker
Goods to ship
Decide if normal post or special shipment
Mode of delivery
Normal Post
Special Carrier
Check if extra insurance is necessary
extra insurance required
Take out extra insurance
Always
Fill in a Post label
Request quotes from carriers
Assign a carrier & prepare paperwork
Package goods
Insurance is included in carrier service
Add paperwork and move package to pick area
Goods available for pick

omg.org, BPMN 2.0 by Example

Example: HW Delivery


Hardware Retailer
Logistics Manager
Clerk
Warehouse Worker
Goods to ship
Decide if normal post or special shipment
Normal Post
Mode of delivery
Special Carrier
Check if extra insurance is necessary
extra insurance required
Take out extra insurance
Always
Fill in a Post label
Request quotes from carriers
Assign a carrier & prepare paperwork
Package goods
Insurance is included in carrier service
Add paperwork and move package to pick area
Goods available for pick

Paired control elements

Optional execution

Decision points ("XOR")

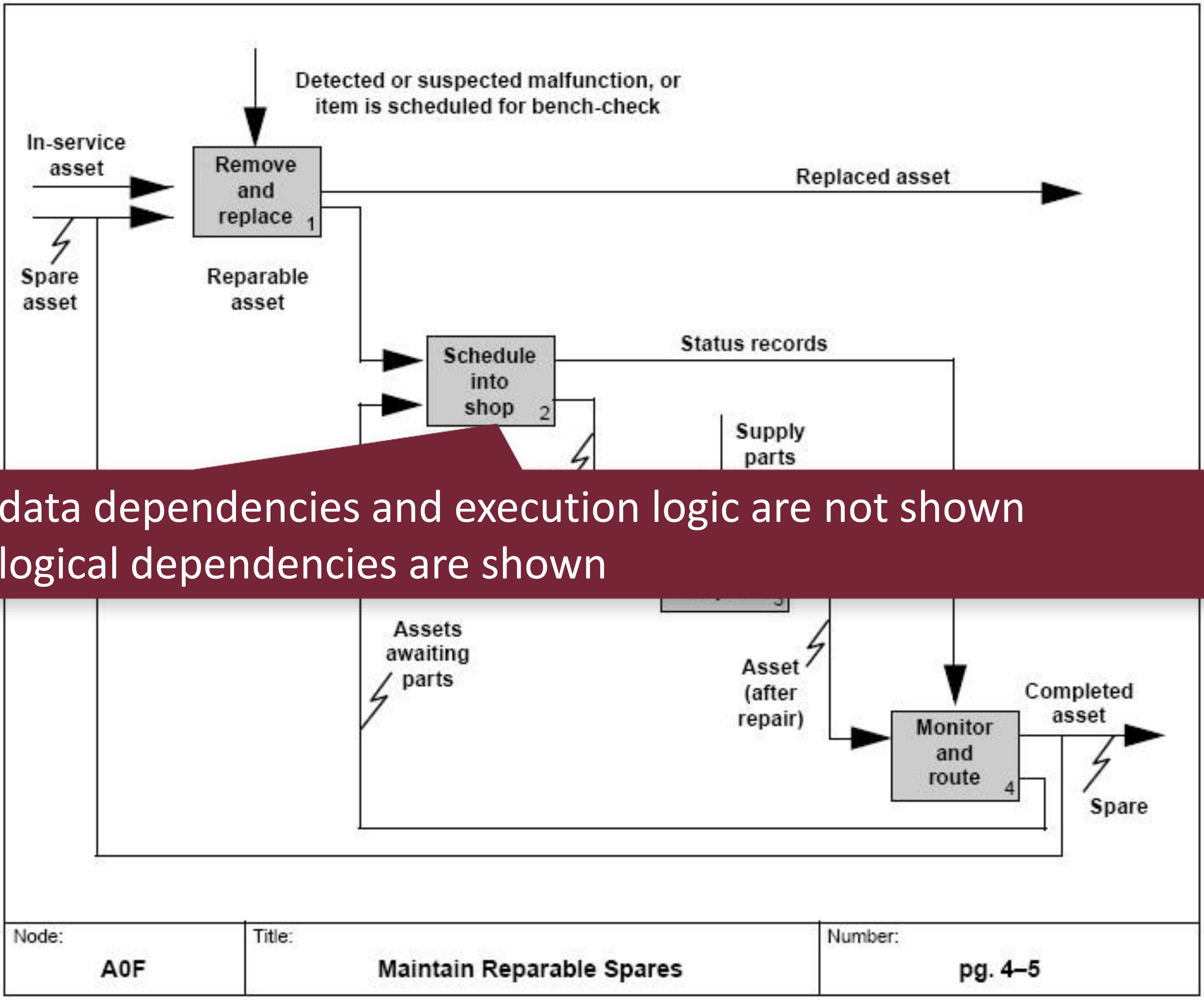
Order of execution

"Parallel" (independent) execution ("AND")

What It’s Based On

- History
  - Programs control structures
  - Scheduling (eg. GANTT diagrams)
  - Modelling manufacturing/office processes
  - IDEF-0: 1980’s, US AirForce
  - Describing logistic processes
  - System operator’s/administrator’s “runbook”
- Common elements
  - There are atomic steps
  - Dependencies between them (time? data? order?)
  - Decision points
  - → general-purpose process modelling languages (eg. BPMN)

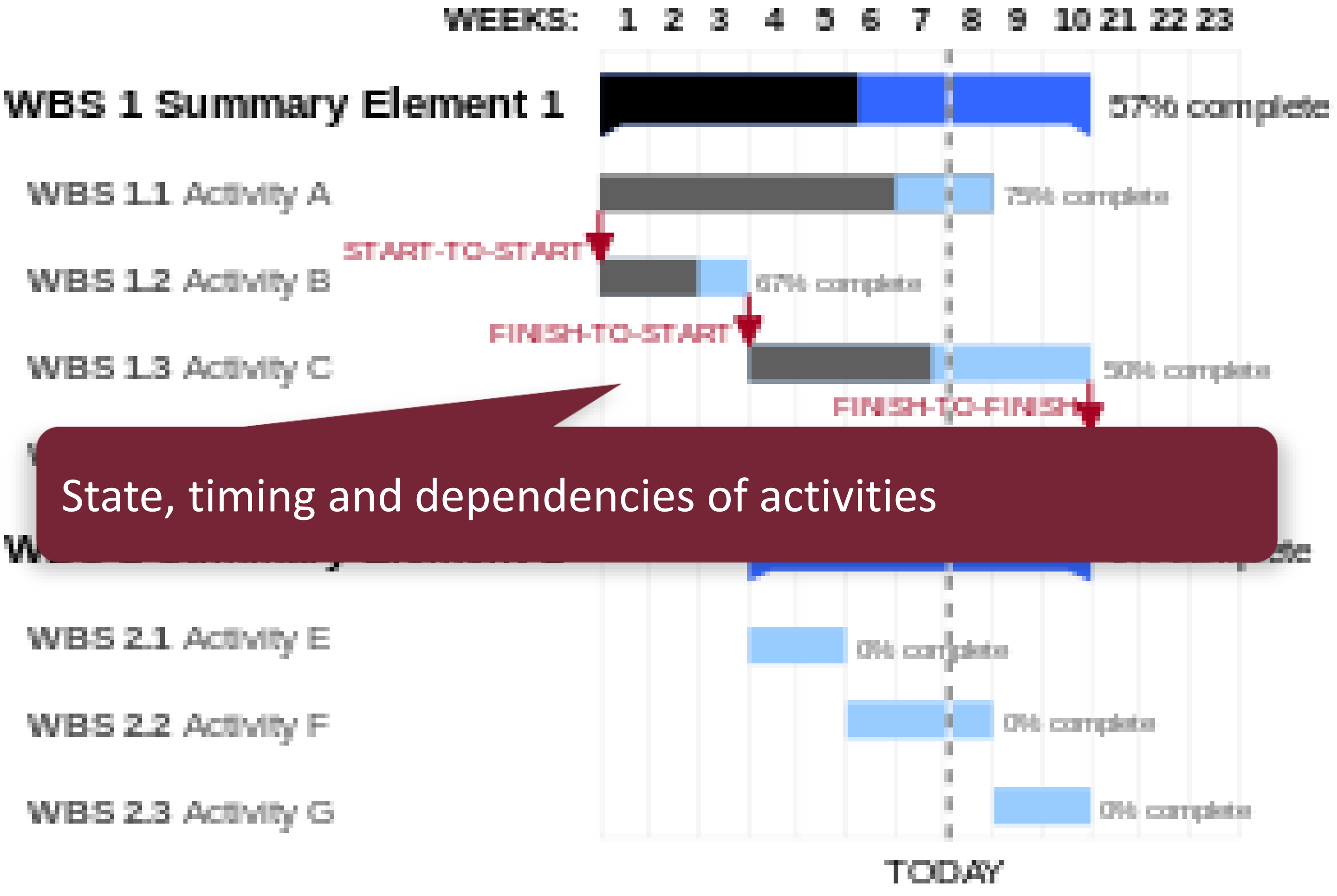
Example: IDEF-0



data dependencies and execution logic are not shown
logical dependencies are shown

Defense Acquisition University - *Systems Engineering Fundamentals.* Defense Acquisition University Press, 2001

Example: GANTT



State, timing and dependencies of activities

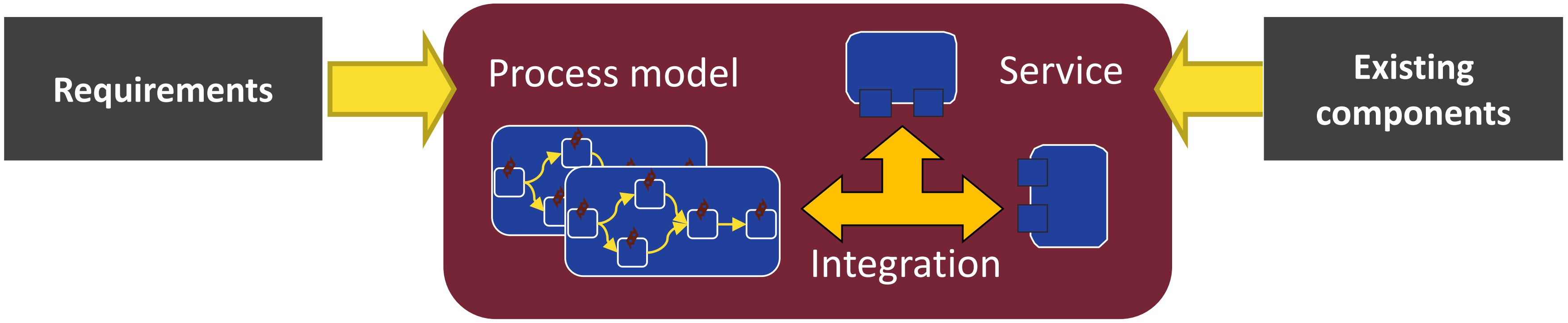
wikipedia.org

What It Uses

- Idea in system/software design:
  - Use existing elements
  - Describe how the complex system operates
- Basic elements can be many
  - webform validation, sending email, database operation, remote web service, human interaction, sending text message, drawing diagram, etc.

What is Derived from the Control Logic?

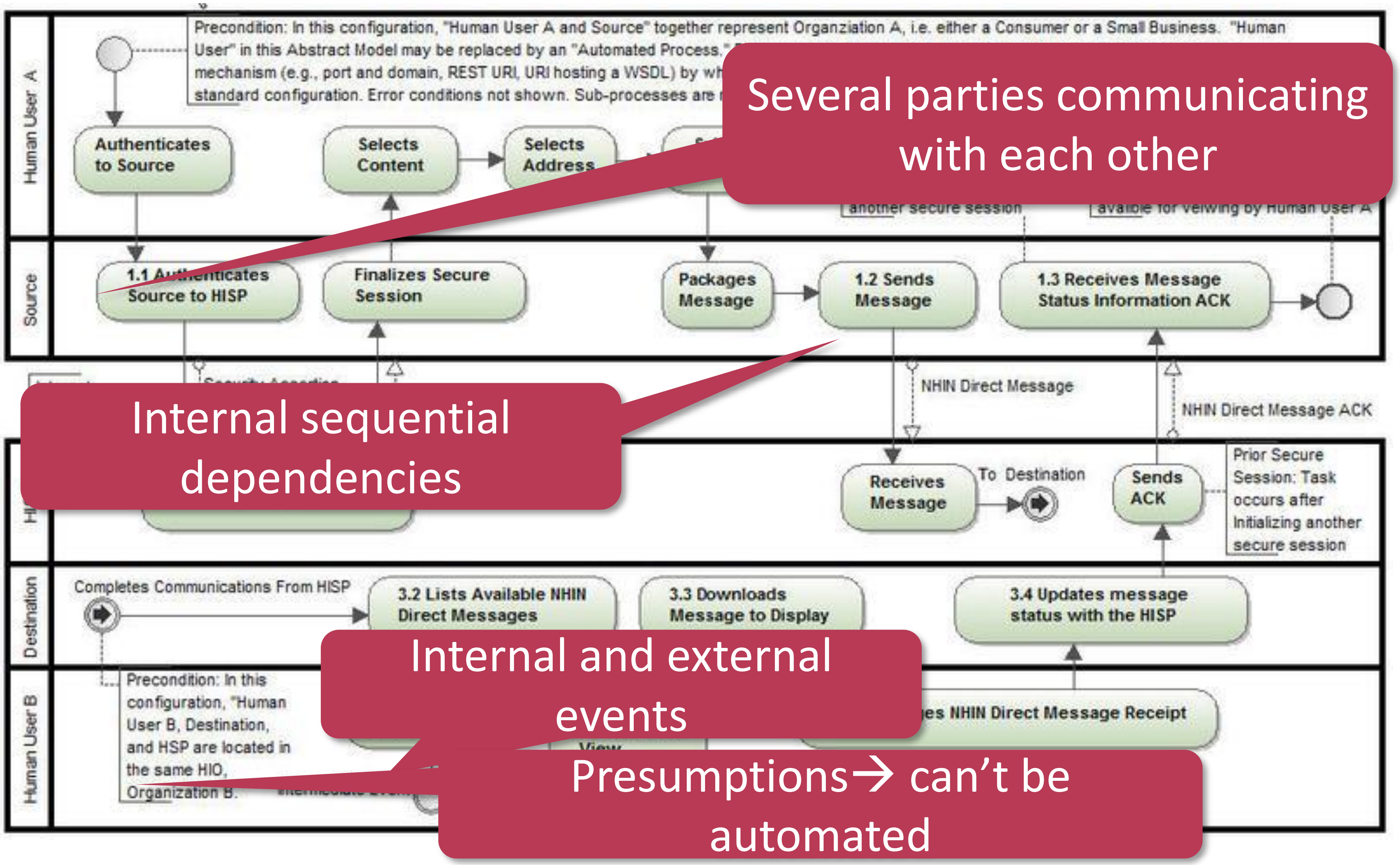
- Program code directly (C/C++, C#, Java, …)
- Input of an executing environment
  - “Create this process for me”



Other Uses of Process Models

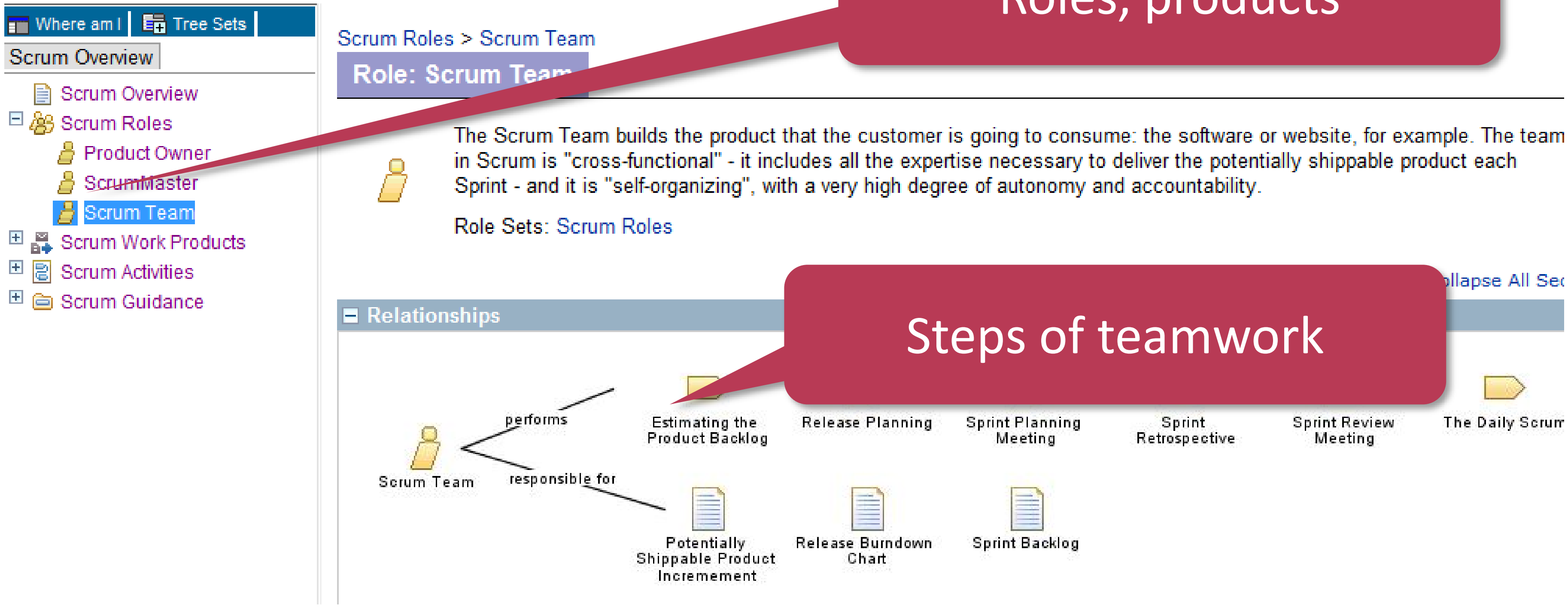
- Operating IT systems
  - ITIL, UK Gov. initiative
- Protocol specification
  - Cooperation between elements of a complex system
  - Roles of components
- Designing executable processes
  - Order evaluation, credit assessment preparation, …
- Data processing/analysing processes

Example: Managing Health Data


Human User A
Source
Destination
Human User B
Precondition: In this configuration, "Human User A and Source" together represent Organziation A, i.e. either a Consumer or a Small Business. "Human User" in this Abstract Model may be replaced by an "Automated Process," mechanism (e.g., port and domain, REST URI, URI hosting a WSDL) by wh standard configuration. Error conditions not shown. Sub-processes are
Authenticates to Source
Selects Content
Selects Address
another secure session
avalble for viewing by Human User A
1.1 Authenticates Source to HISP
Finalizes Secure Session
Packages Message
1.2 Sends Message
1.3 Receives Message Status Information ACK
Security Assertion
NHIN Direct Message
NHIN Direct Message ACK
Receives Message
To Destination
Sends ACK
Prior Secure Session: Task occurs after Initializing another secure session
Completes Communications From HISP
3.2 Lists Available NHIN Direct Messages
3.3 Downloads Message to Display
3.4 Updates message status with the HISP
Precondition: In this configuration, "Human User B, Destination, and HSP are located in the same HIO, Organization B.
es NHIN Direct Message Receipt
View
Several parties communicating with each other
Internal sequential dependencies
Internal and external events
Presumptions→ can't be automated

http://wiki.directproject.org/Abstract+Model+Examples

Example: Agile Development, as a Process



The Scrum Team builds the product that the customer is going to consume: the software or website, for example. The team in Scrum is "cross-functional" - it includes all the expertise necessary to deliver the potentially shippable product each Sprint - and it is "self-organizing", with a very high degree of autonomy and accountability.

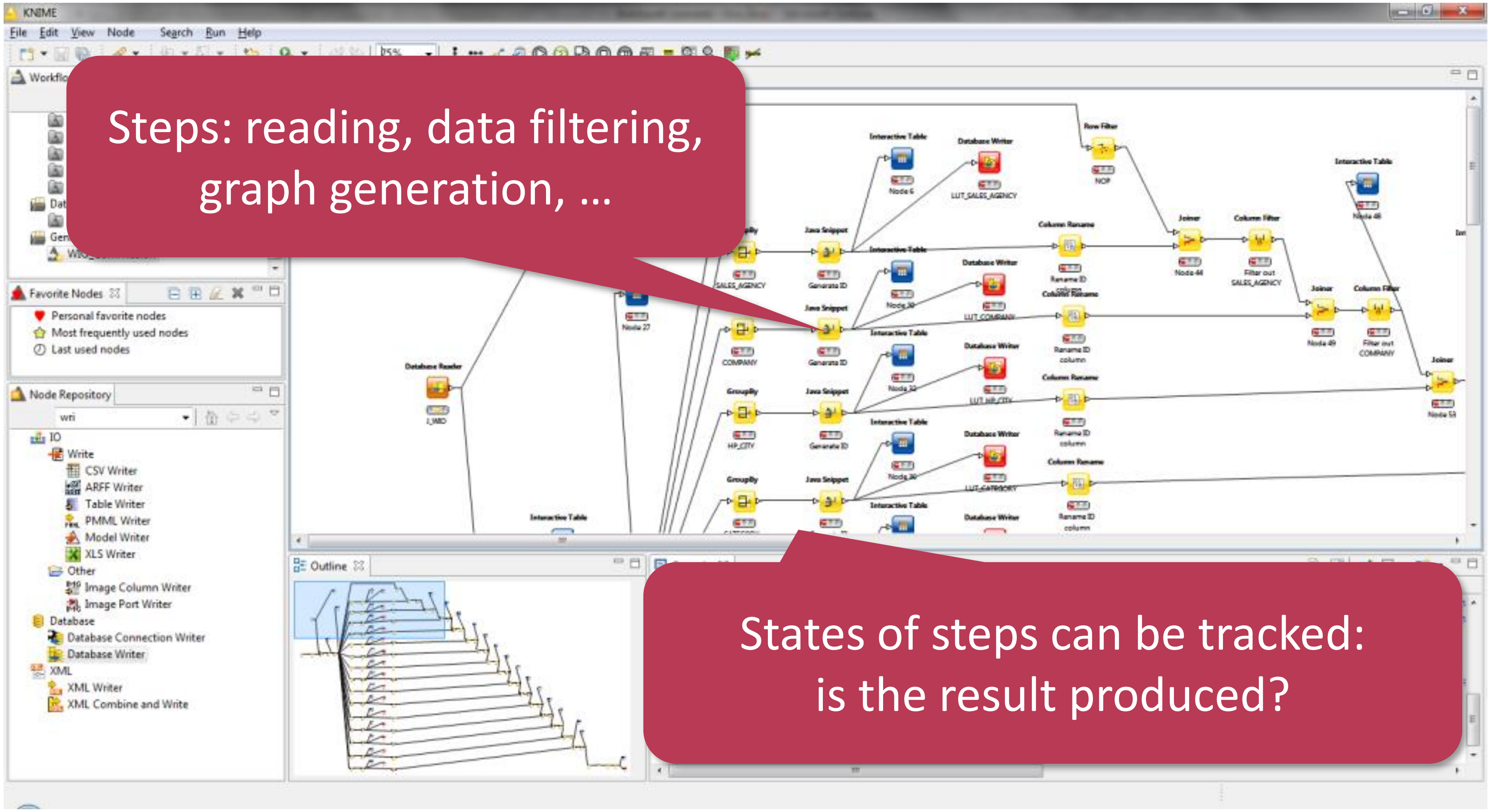
Role Sets: Scrum Roles

http://www.eclipse.org/epf/

Examples

- Modelling banking processes
  - What activities are executed closing time?
  - Could the bank switch to transferring multiple times a day?
- Modelling manufacturing process
  - Optimal production scheduling: convert or fabricate?
  - What happens in the factory?
  - (see the lecture on Simulation)
- Modelling business transactions
  - Where are recurring communication patterns?
  - Model based data processing

Example: Data Processing



Steps: reading, data filtering, graph generation, …

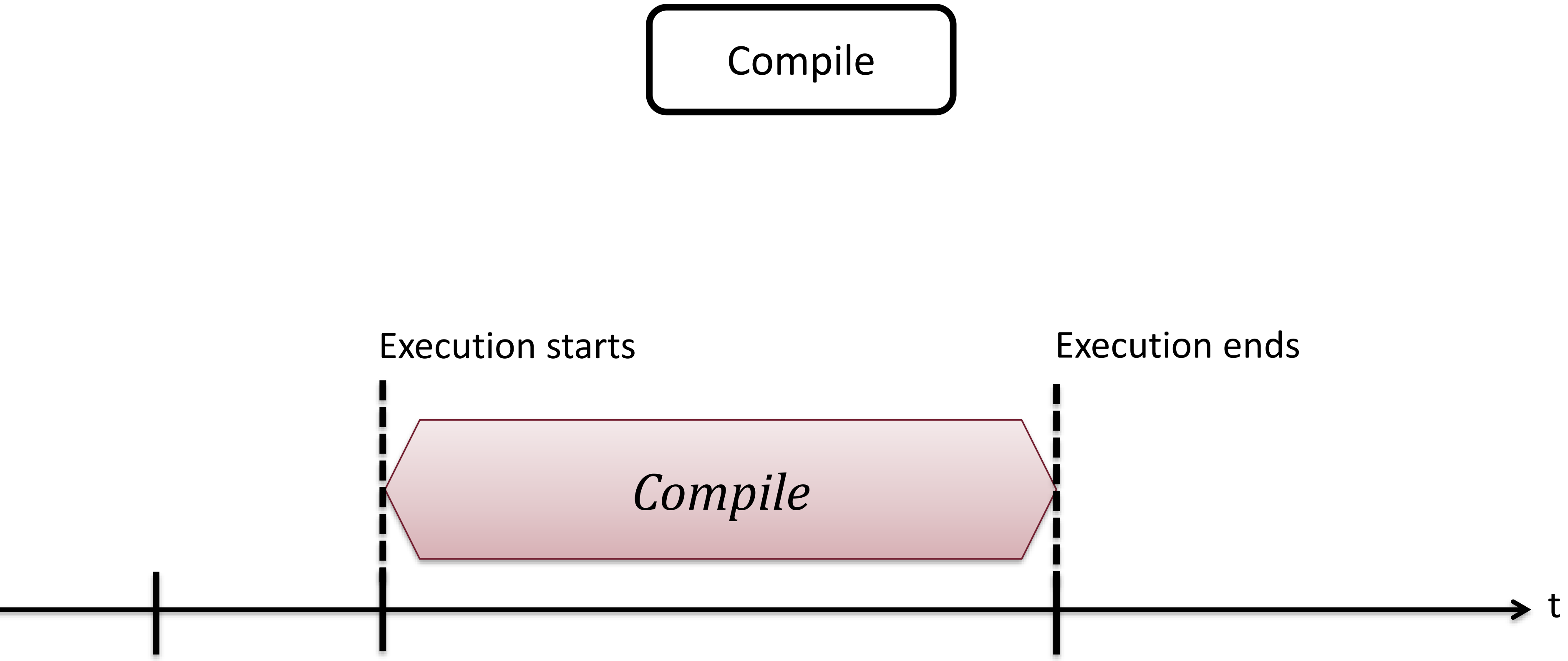
States of steps can be tracked: is the result produced?

Basic concepts of designing processes

- Process description languages
  - BPMN, jPDL, XPDL, BPEL, UML AD, …
- Process model
  - Control, dataflow
  - Data structures can be linked to a process model
  - Definition of steps to execute
  - Timings, resources
- Process (template) vs. process instance
  - E.g. „Booking tickets" as a process
  - „László Gönczy books a ticket to Lisbon" is an instance

Table of contents

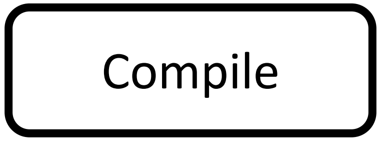
Elementary Activity (Task)



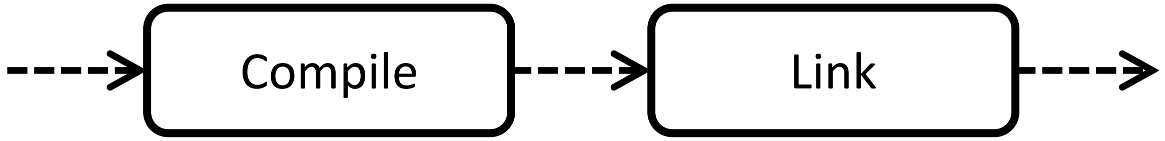
Definition: Elementary Activity

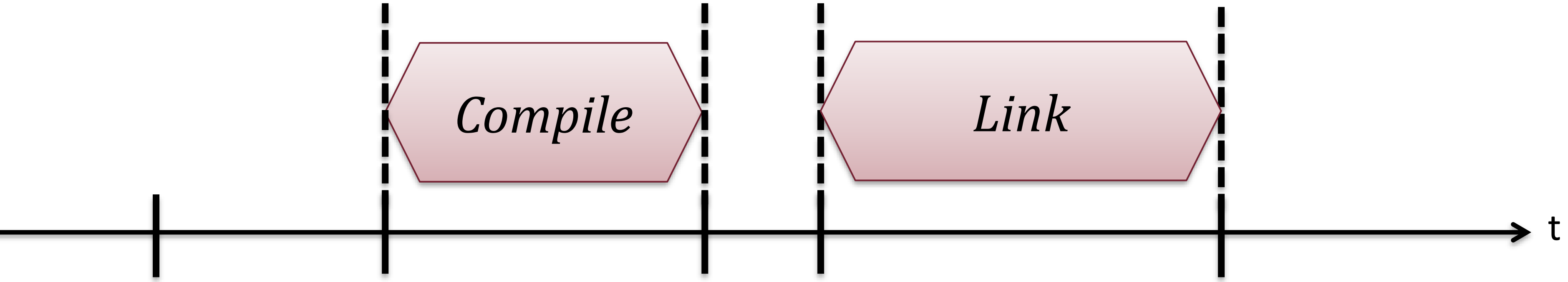
An **elementary activity** is an activity that

- has a positive temporal duration
- is *not* modelled beyond its start and end.



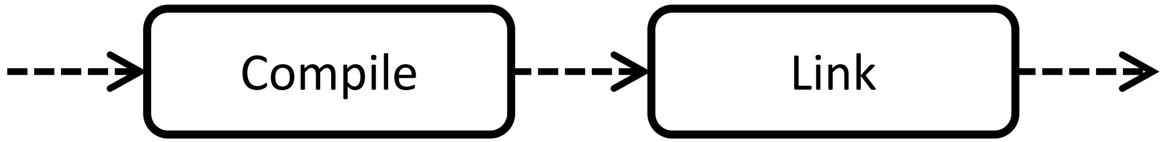
Sequence, Control Flow



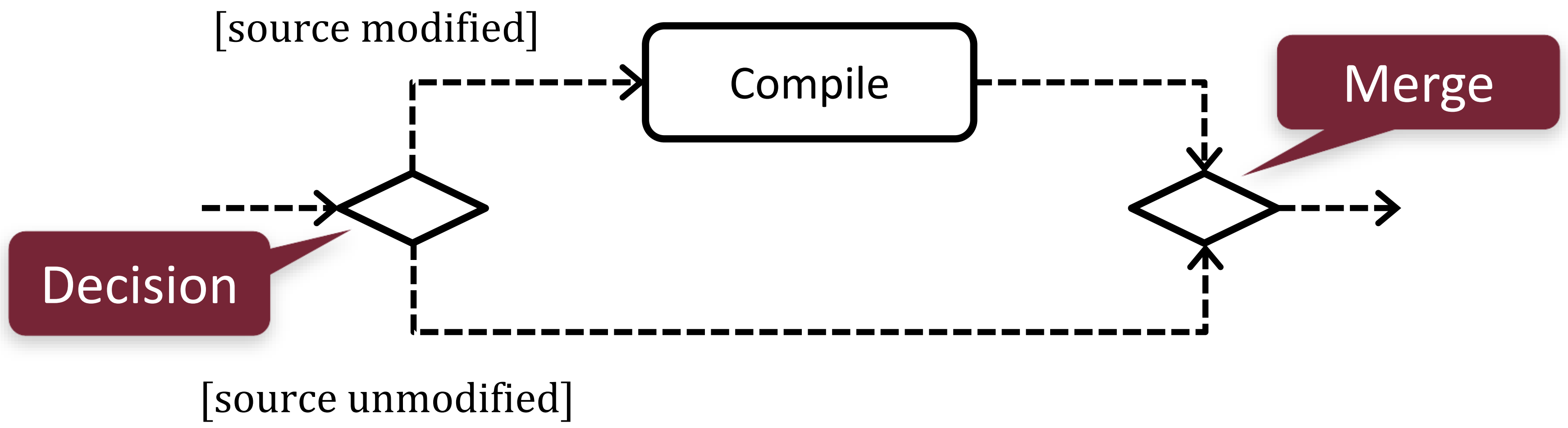


Definition: Sequence

Sequence defines the order of execution of activities.



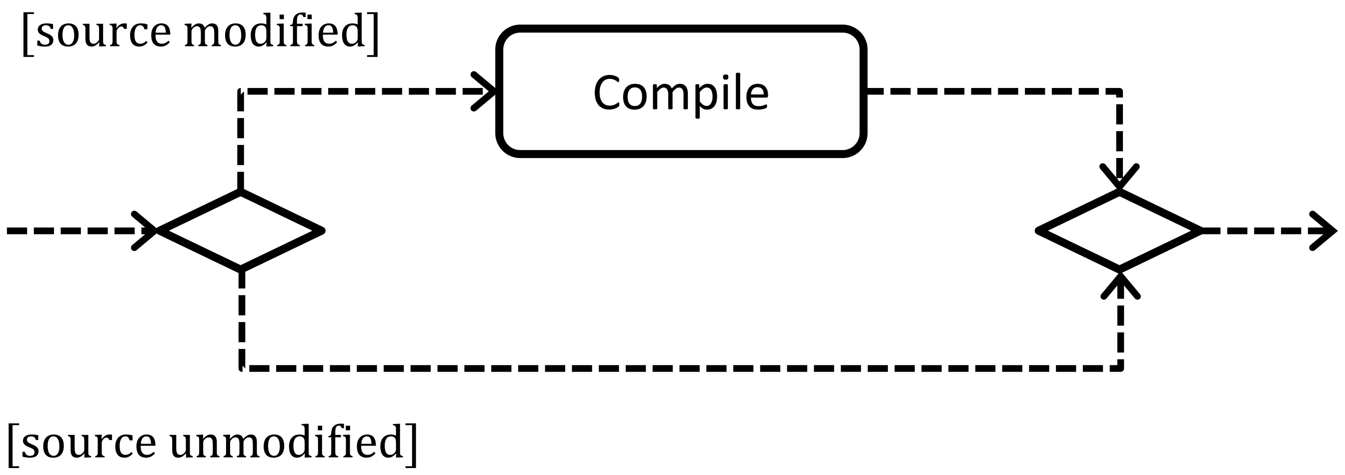
Guard Condition, Branches



- Semantics:
  - Only one branch is executed
  - Possibility of nondeterminism
    - Overlapping guard conditions
    - Or simply no guard conditions

Definition: Control Element

A **control element** is a junction of the process choosing one or more activities to execute.



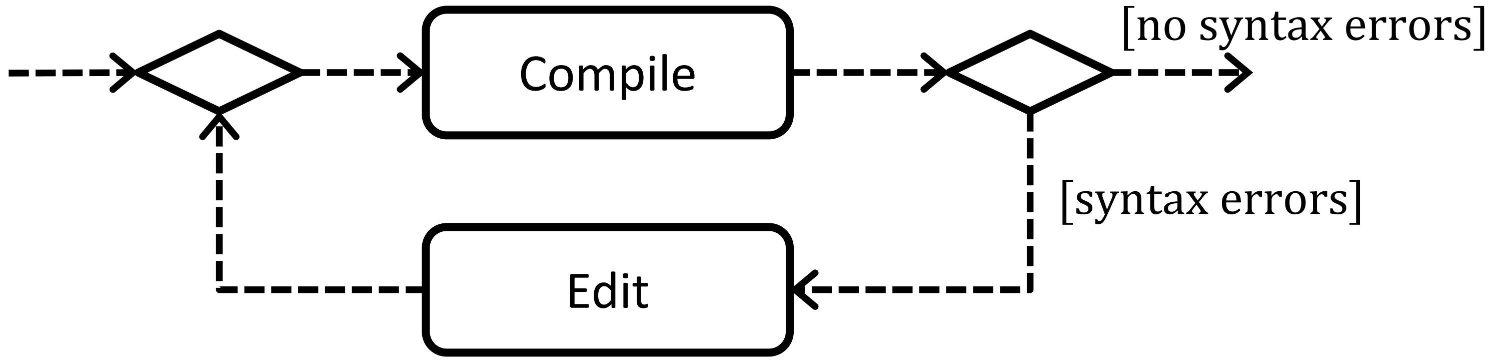
Definition: Decision-Merge

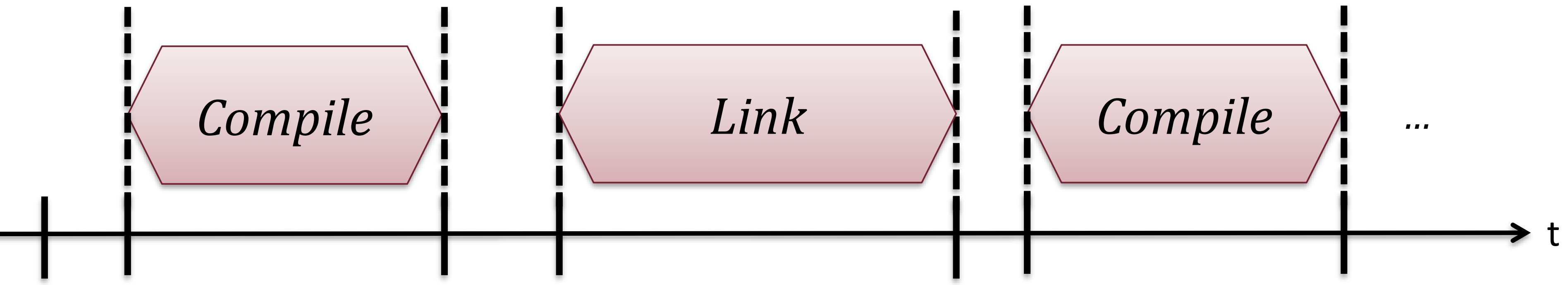
Decision-Merge is a control structure

- consisting of a **Decision** and a **Merge** control element, where
- the decision node has at least two **outputs** from which we choose where to put the control token by evaluating the **guard conditions,**
- the chosen output (branch) can contain an arbitrary number of elements, and
- each branch leads to the merge node.

- Here we use branch as an exclusive or (XOR gate), which means that as a result of an evaluation only one of the decision branch is chosen.
- A branch can be multiple or binary, in the course we use binary decisions (two outputs).

Loop


Compile
[no syntax errors]
[syntax errors]
Edit

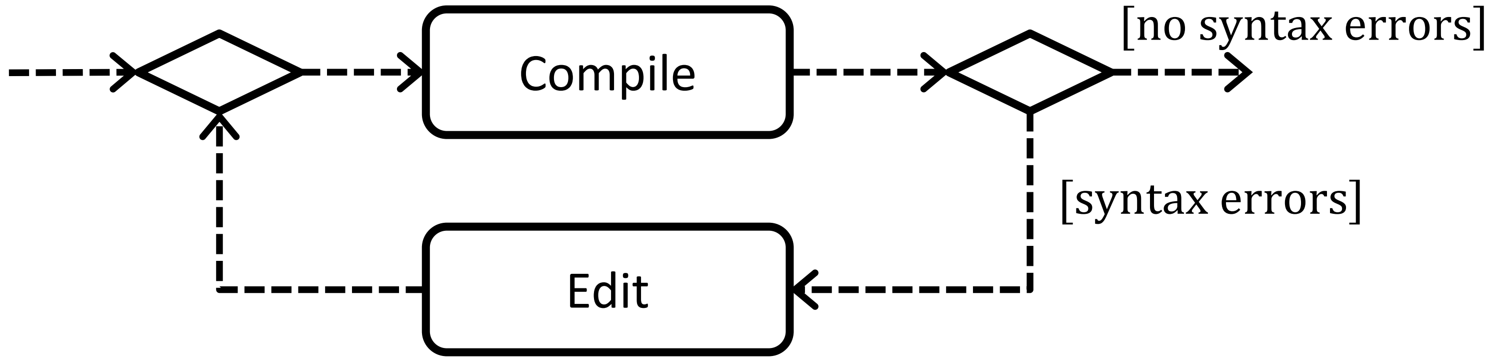

Compile
Link
Compile
...
t

Definition: Loop

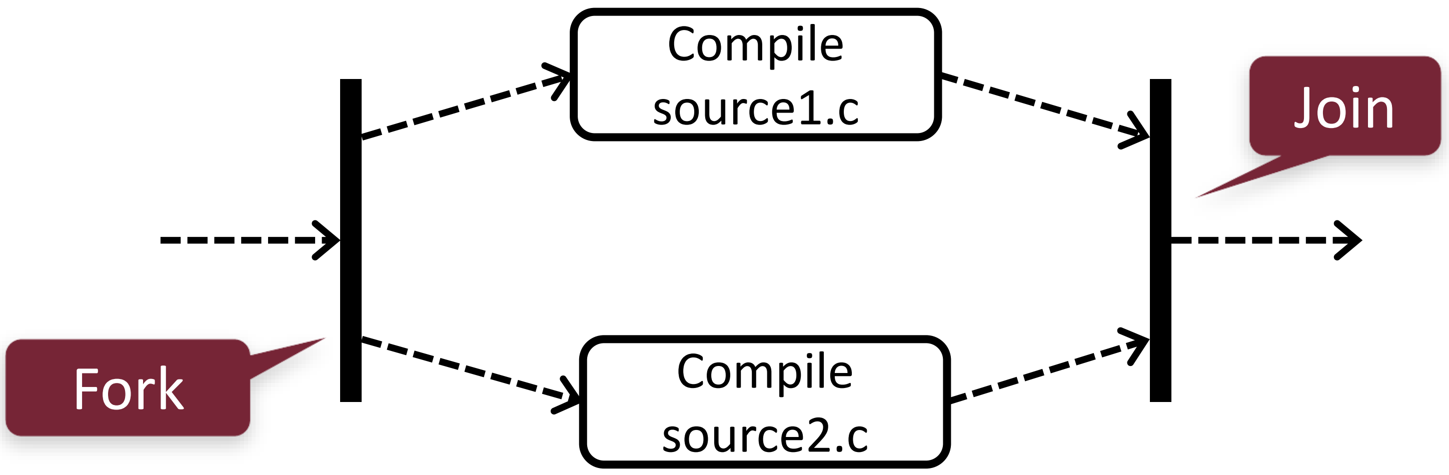
A **loop** is a control structure that defines multiple execution. The loop

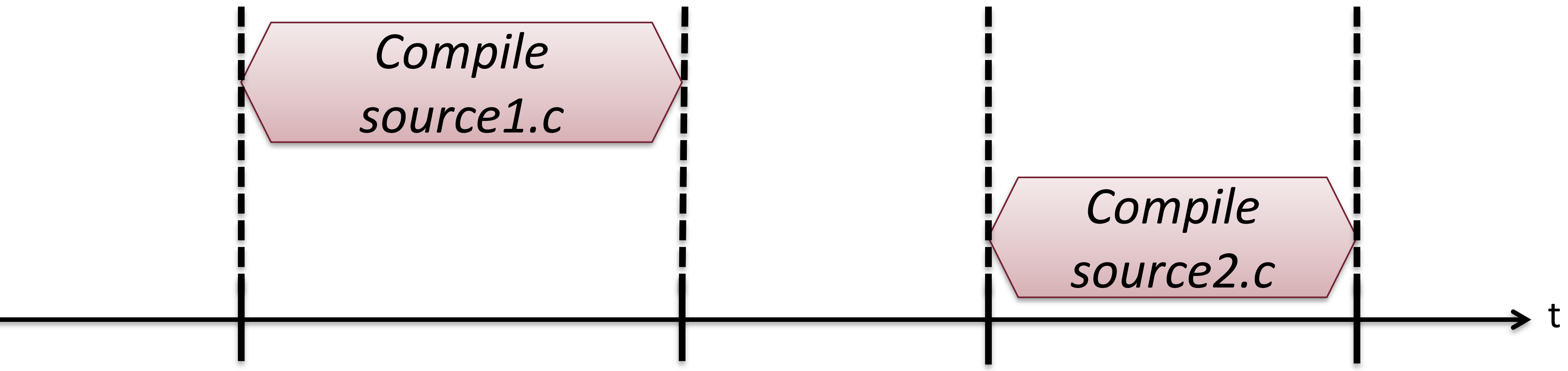
- consists of a **Merge** and a **Decision** element, where
- one of the branches of the decision node leads back to the merge node.

- *Note*: this corresponds to a **`repeat – until`** loop

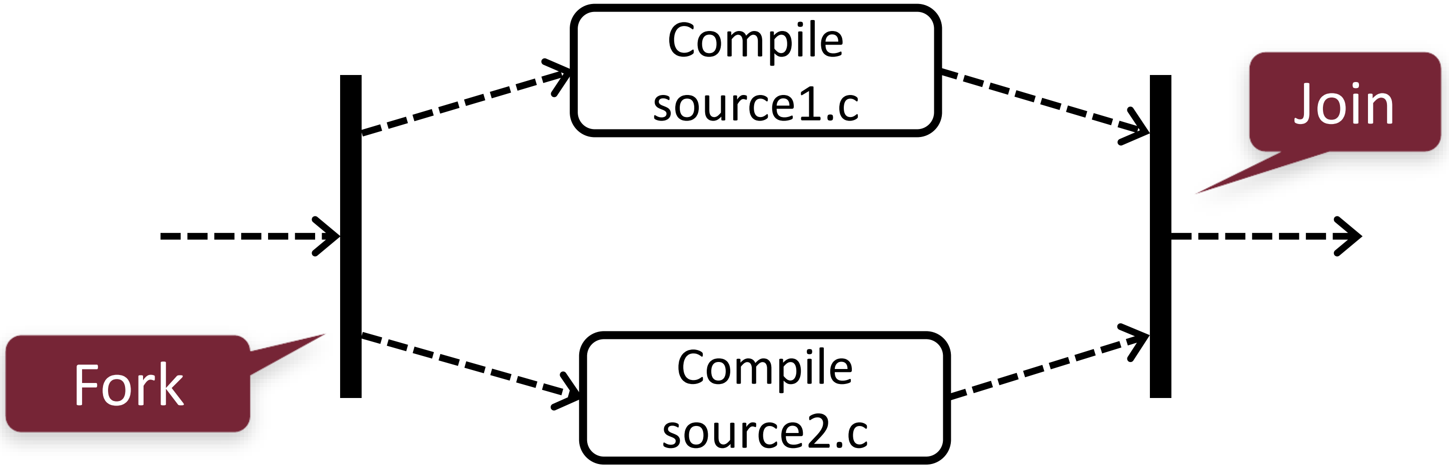


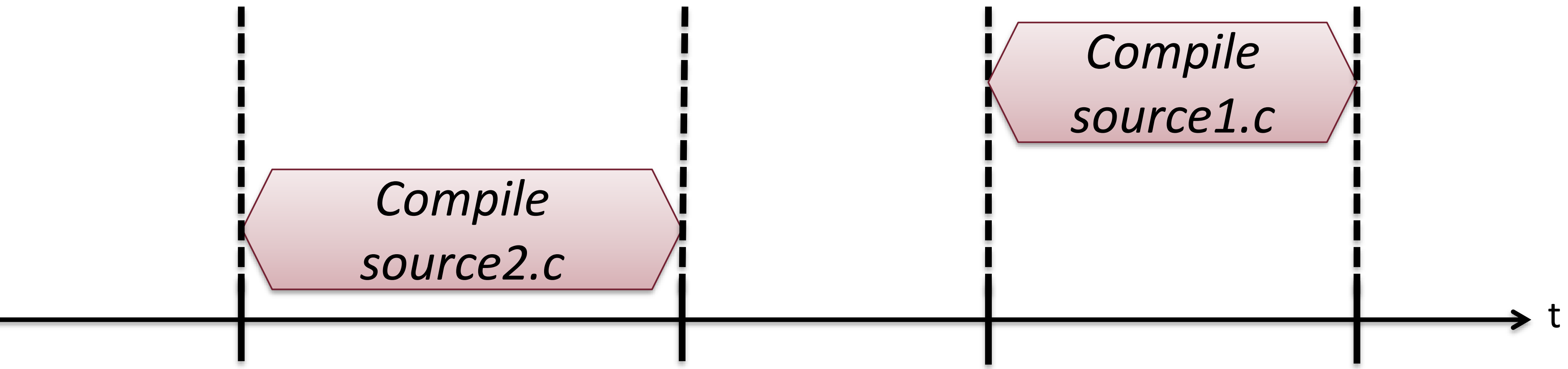
Fork / Join


Compile source1.c
Join
Fork
Compile source2.c

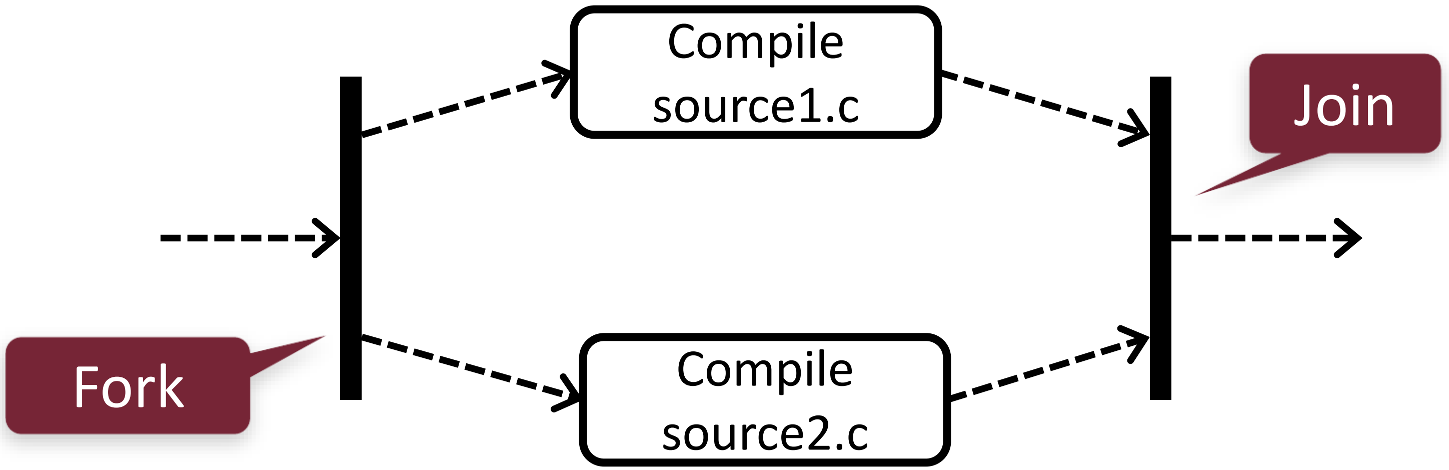

Compile source1.c
Compile source2.c
t

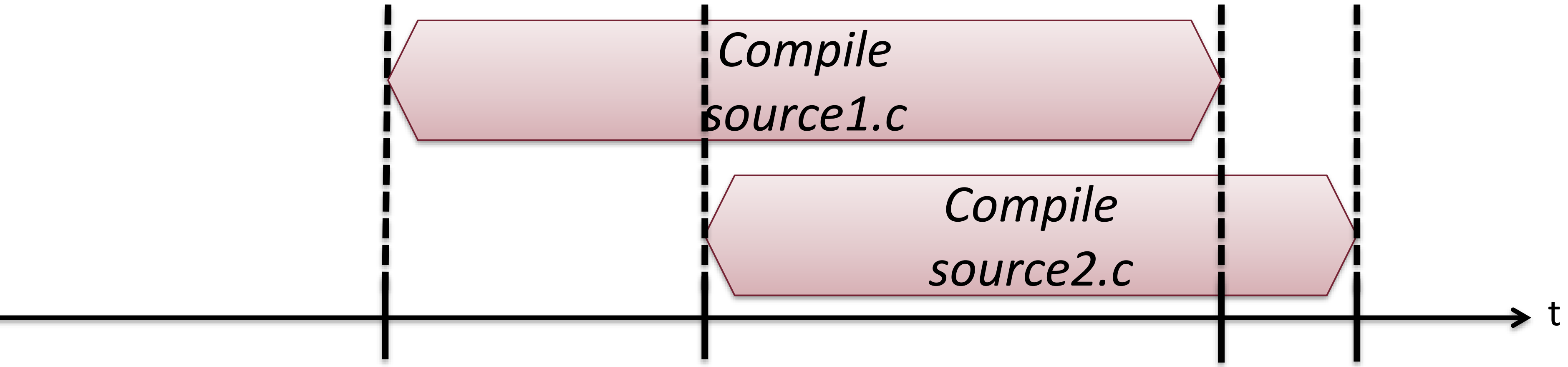
Fork / Join


Compile source1.c
Join
Fork
Compile source2.c

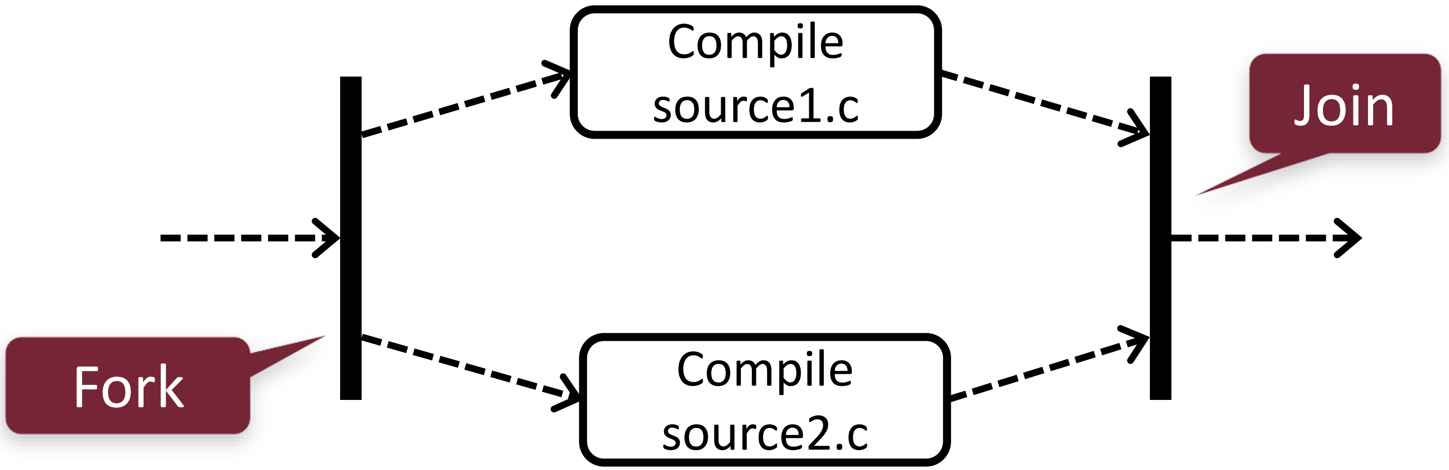

Compile source1.c
Compile source2.c
t

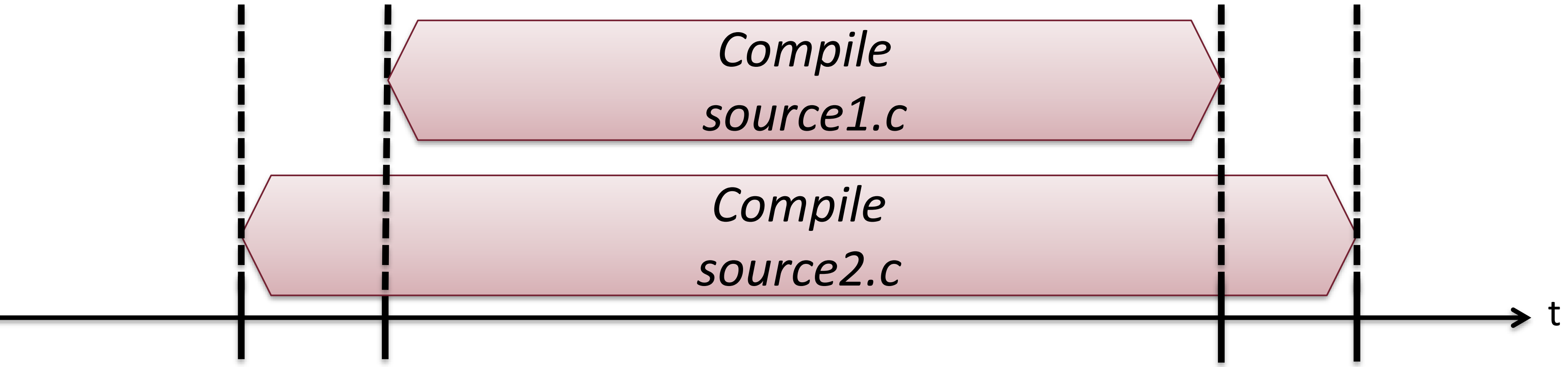
Fork / Join


Compile source1.c
Compile source2.c
Fork
Join

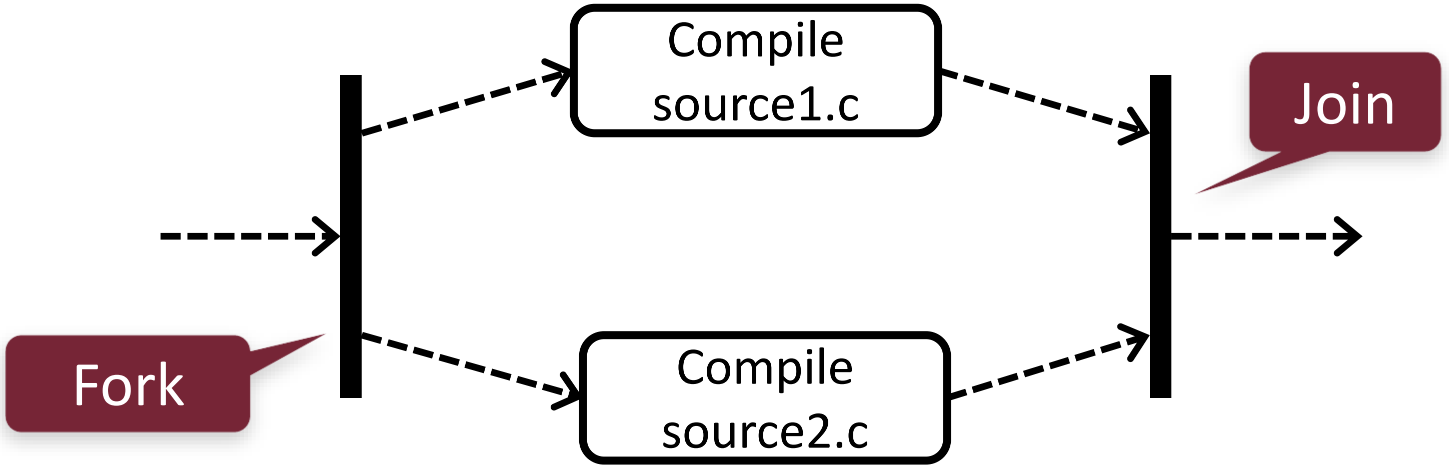

Compile source1.c
Compile source2.c
t

Fork / Join


Compile source1.c
Compile source2.c
Join
Fork


Compile source1.c
Compile source2.c
t

Fork / Join



- Semantics:
  - Execution sequence is not specified
  - Parallel/ overlapped execution is possible
- See: Computer architectures course

Definition: Parallel Execution

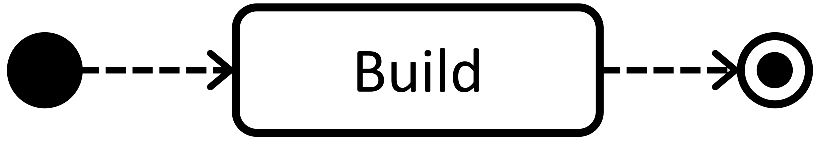
Parallel execution (Fork-Join)

- contains a **Fork** and a **Join** control element, where
- the fork can have an arbitrary number of outputs (branches).
- branches can be executed **concurrently**,
- all branches lead to the join node, and
- parallel execution ends, when all branches terminate.

Two activities are **concurrent** if the order of their execution is not controlled.

- Note: we are going to work with two parallel branches.
- **NOT equivalent to Decision-Merge!**

Flow Begin / Flow End



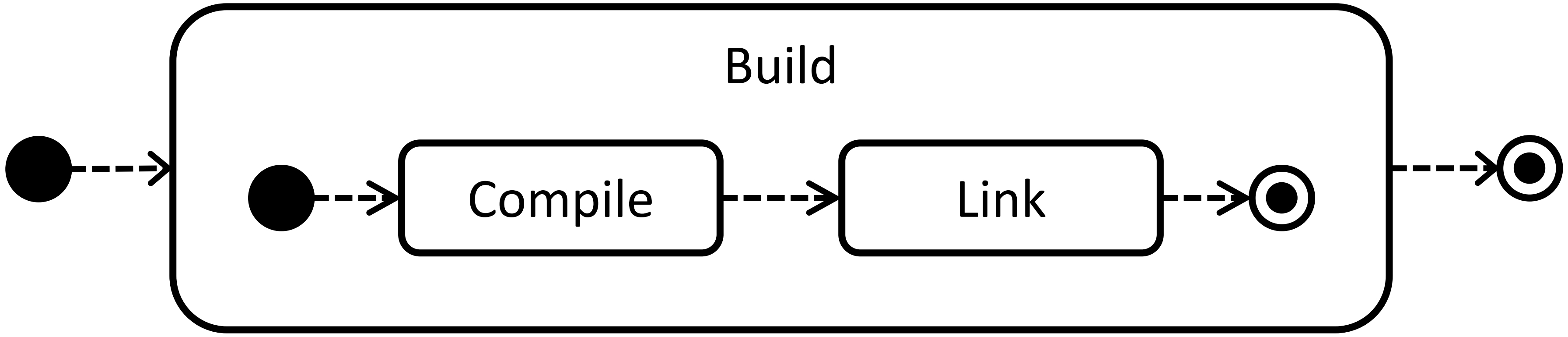
Definition: Flow Begin/End

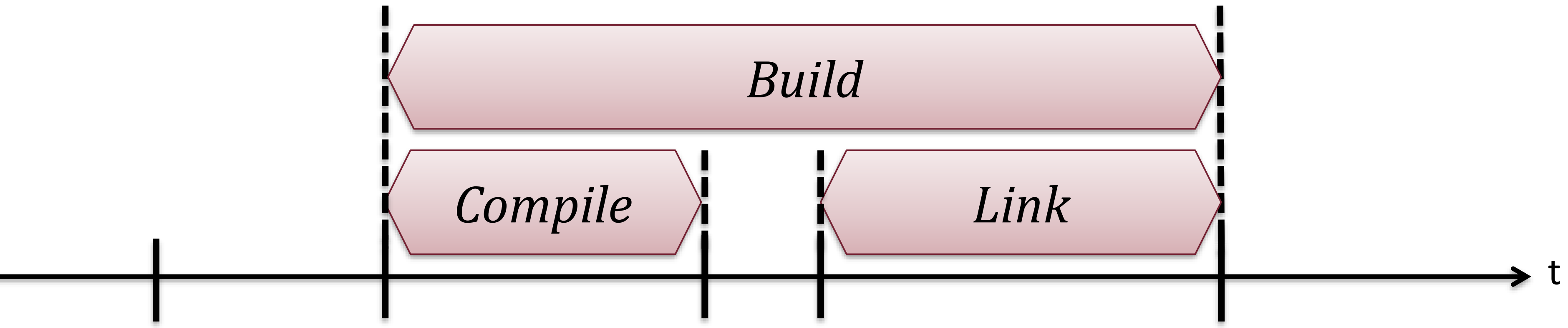
Process starts with a Flow Begin control element and ends with a Flow End element.

- The **begin node** is the first node of the process, with exactly one output.
- The **end node** is the last node of the process with exactly one input.

- Note: we do not model what causes the process to start

Hierarchy



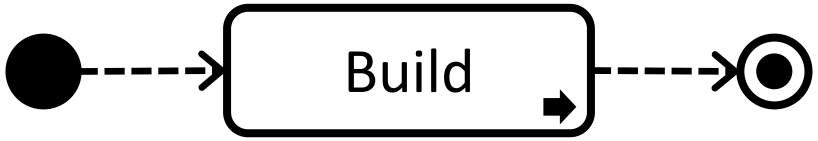


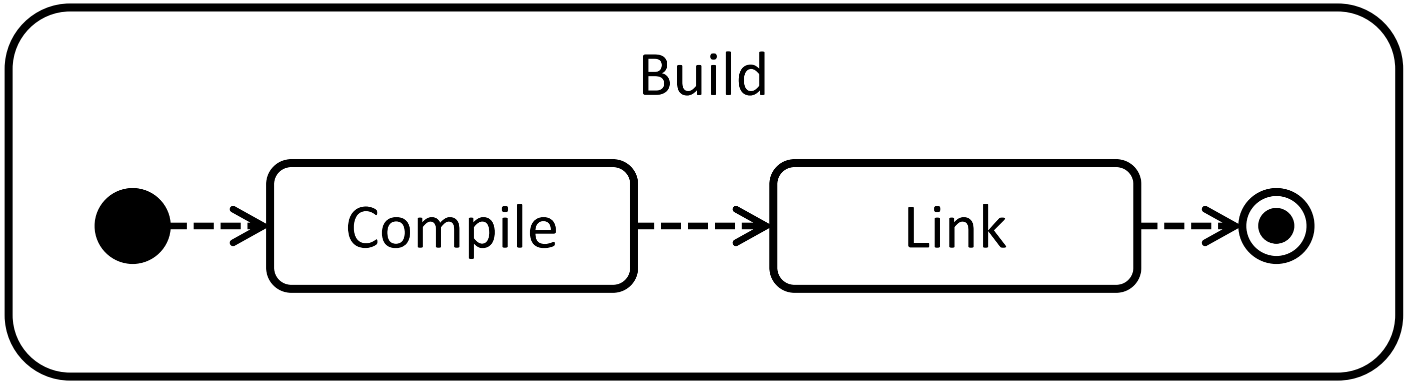
Definition: Hierarchy

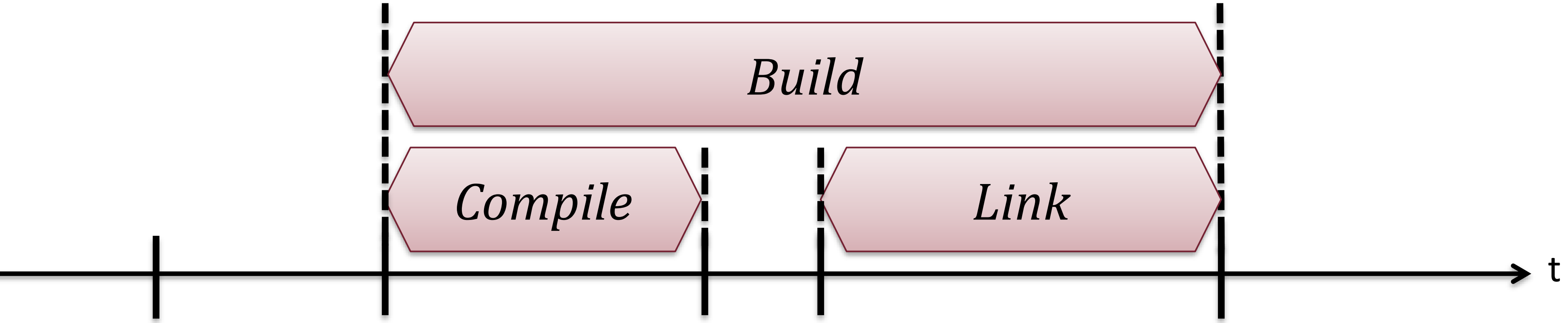
Hierarchical process model:

- Instead of an atomic activity it can contain a submodel described by a process model (hierarchical refinement).

References / Calls

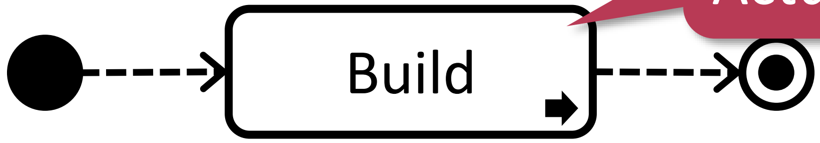


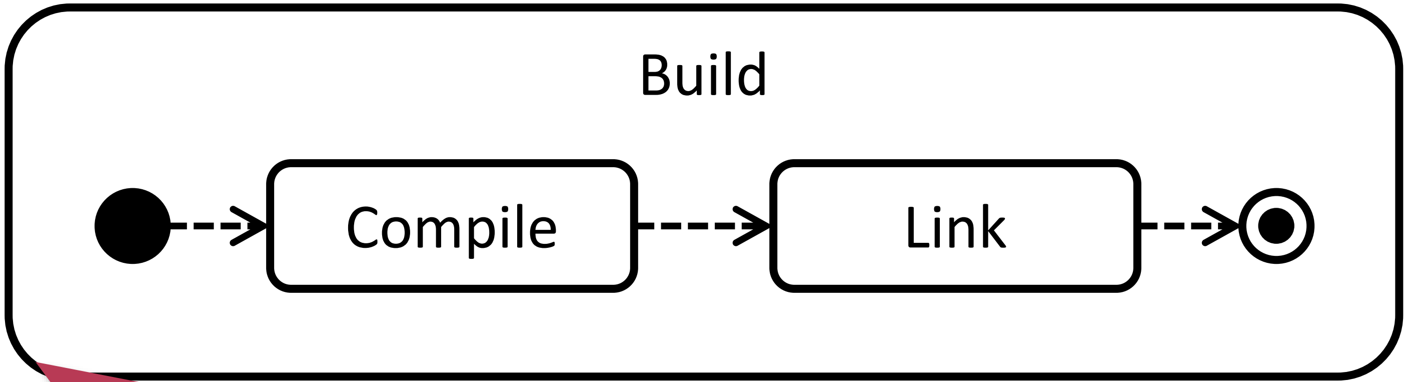




References / Calls

Elementary task?
Actually a subprocess!

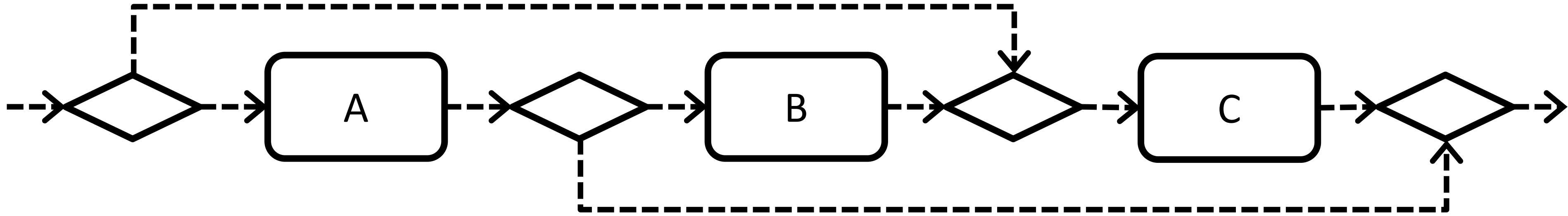




Can be embedded into the main process if the refinement is valid:
- The steps combined produce the same thing as the process
- No unhandled case on caller level
  (Input/output consistency)

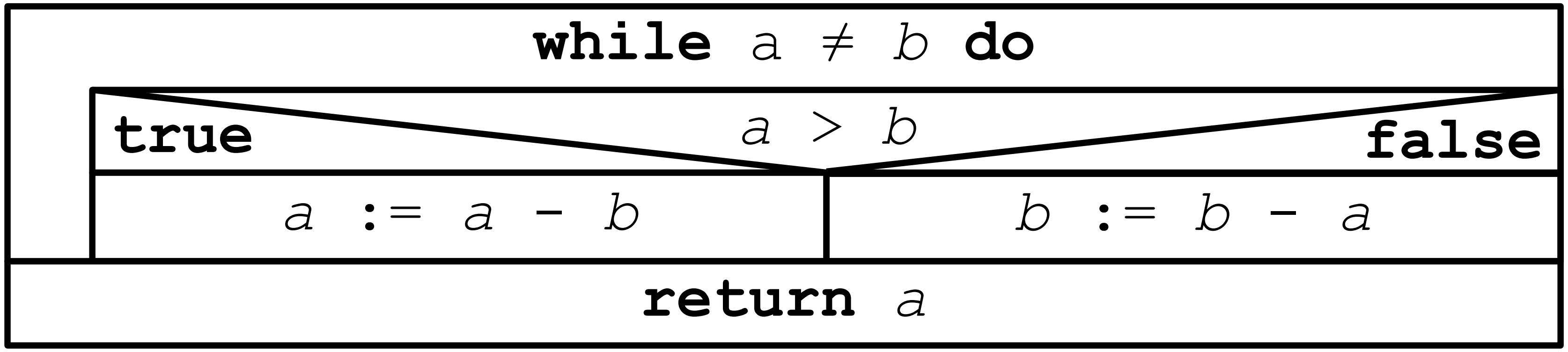
Well Structured Process

- Building from control blocks
  - **One entry point, one exit**
  - Sequence, decision-merge and fork-join blocks, loop, elementary activity, (empty control section)
- Analogy: structured programming
  - Control structures instead of **`goto`**
- Example of a non-well-structured process

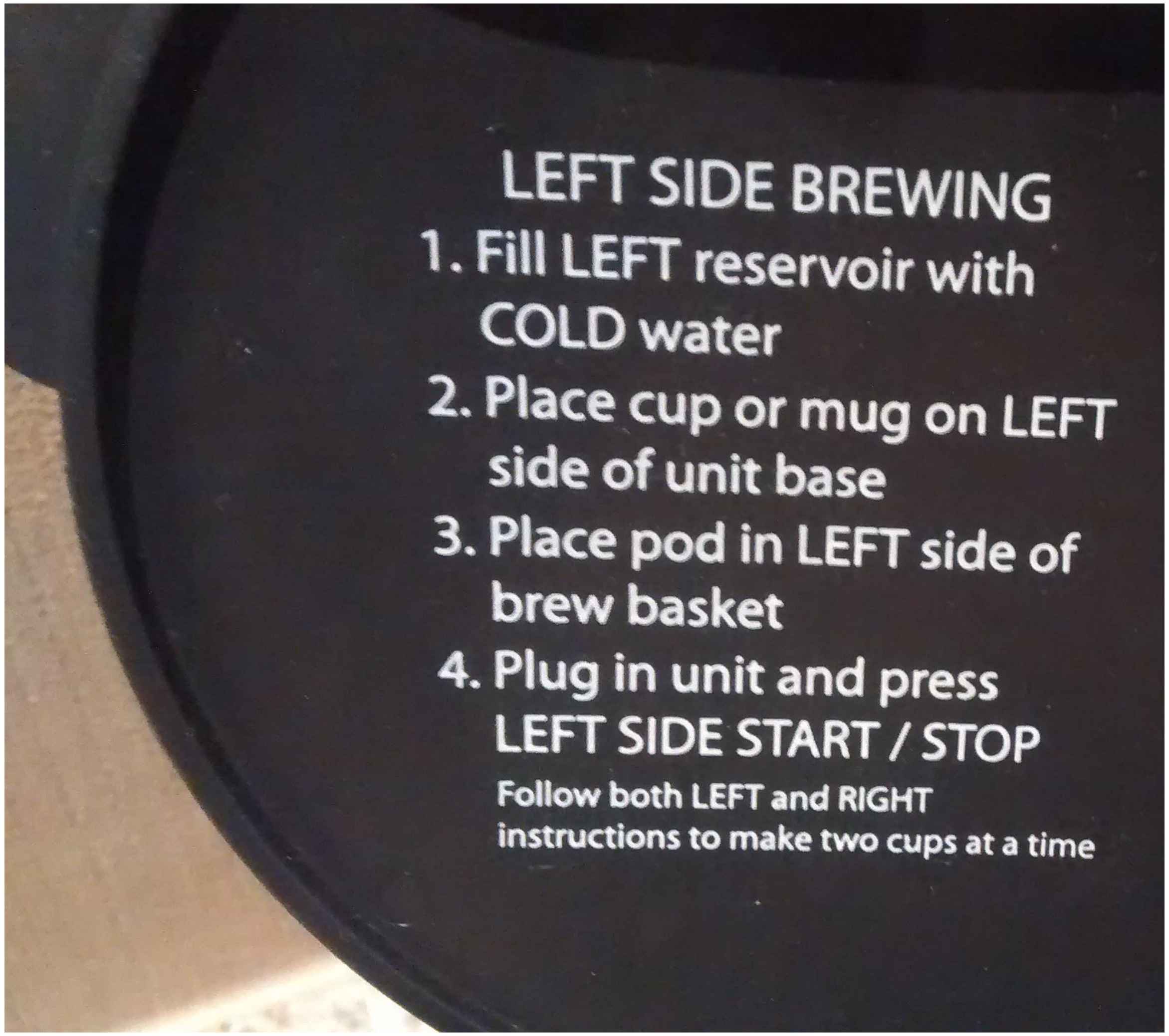


Well Structured Process

- Some formalisms enforce it
  - eg. BPEL (business process over web services)
  - eg. Structogram (Nassi-Shneiderman)
  - programming languages without goto, break, etc.



Example: Coffee Making Process

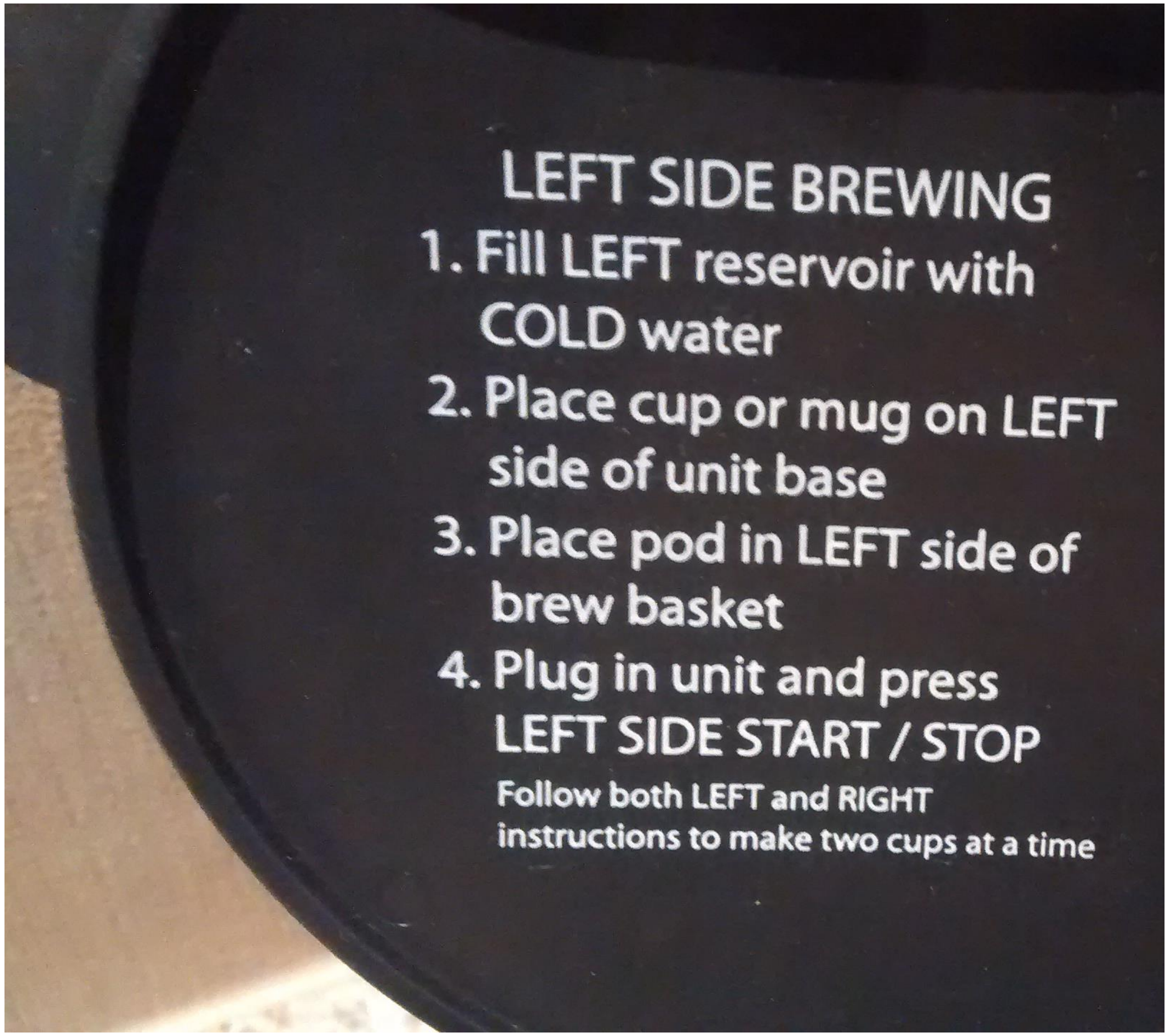


LEFT SIDE BREWING

1. Fill LEFT reservoir with COLD water
2. Place cup or mug on LEFT side of unit base
3. Place pod in LEFT side of brew basket
4. Plug in unit and press LEFT SIDE START / STOP

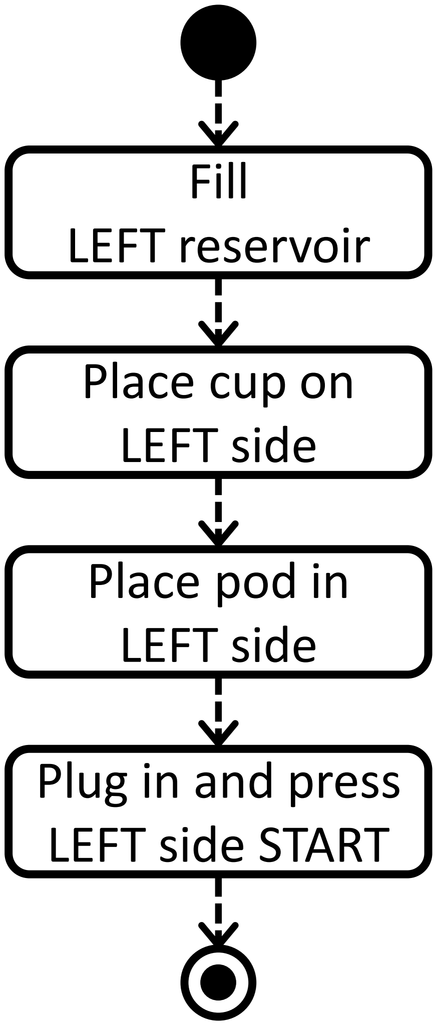
Follow both LEFT and RIGHT instructions to make two cups at a time

Example: Coffee Making Process

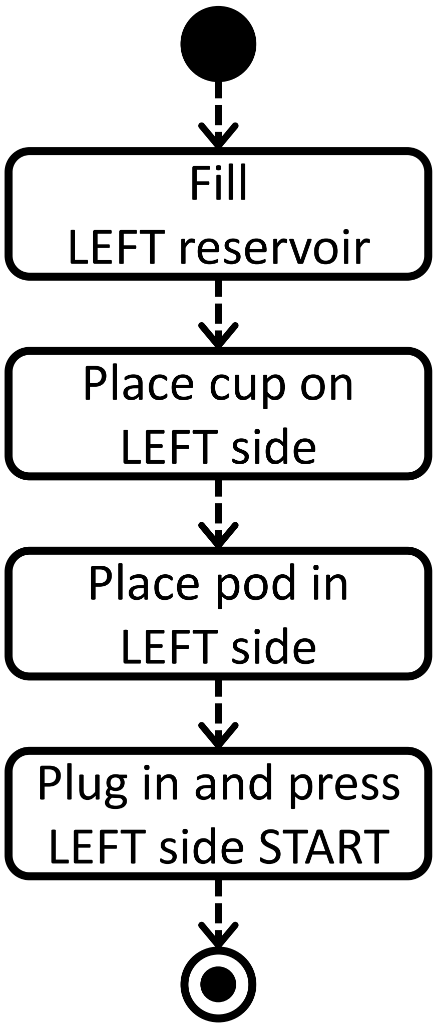

LEFT SIDE BREWING

1. Fill LEFT reservoir with COLD water
2. Place cup or mug on LEFT side of unit base
3. Place pod in LEFT side of brew basket
4. Plug in unit and press LEFT SIDE START / STOP

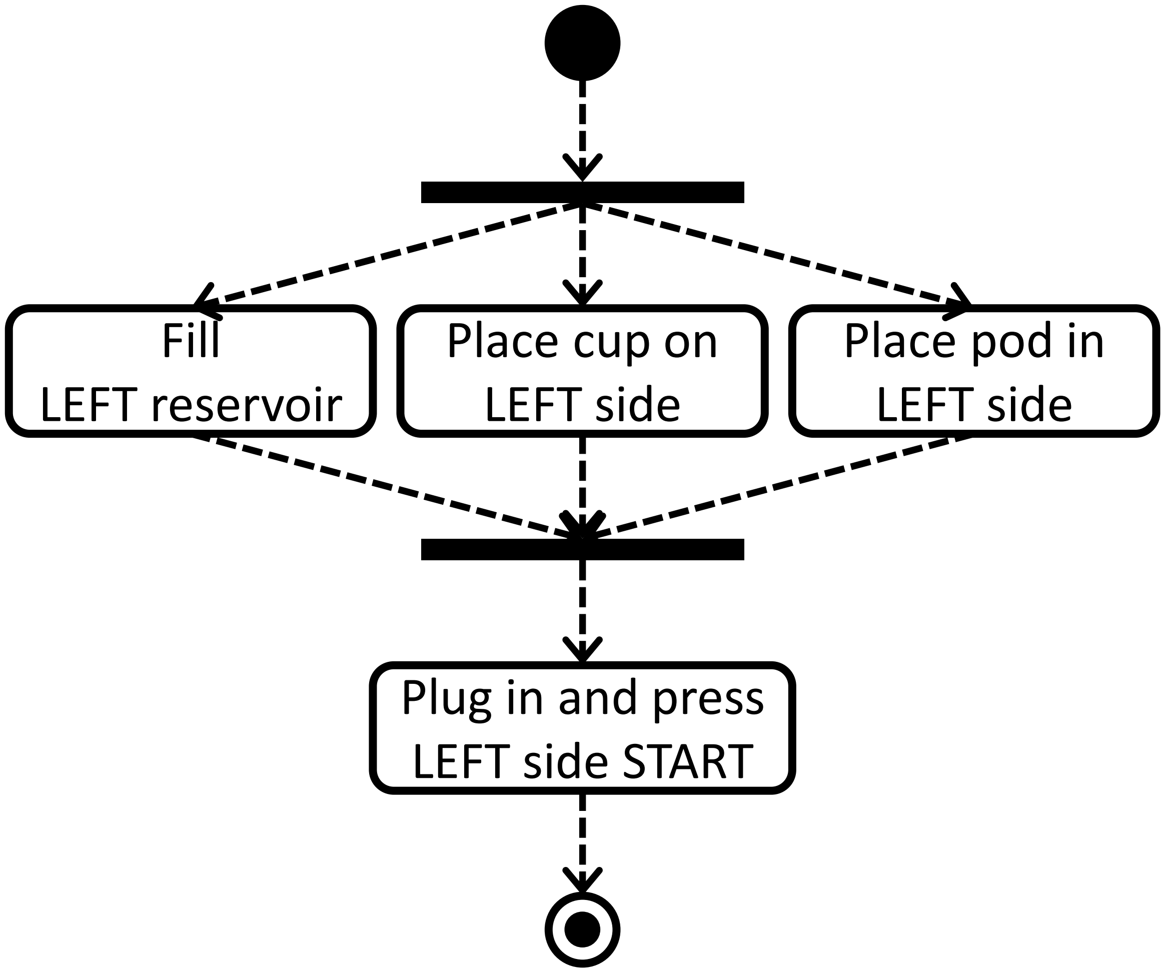
Follow both LEFT and RIGHT instructions to make two cups at a time



Example: Coffee Making Process

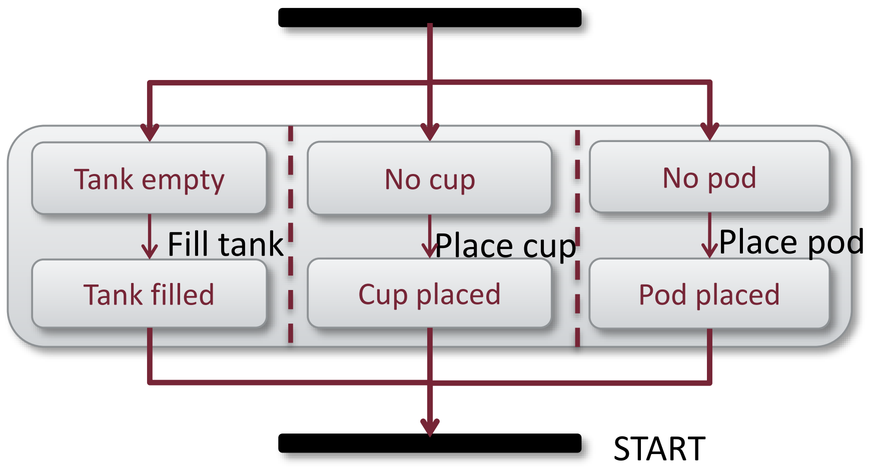


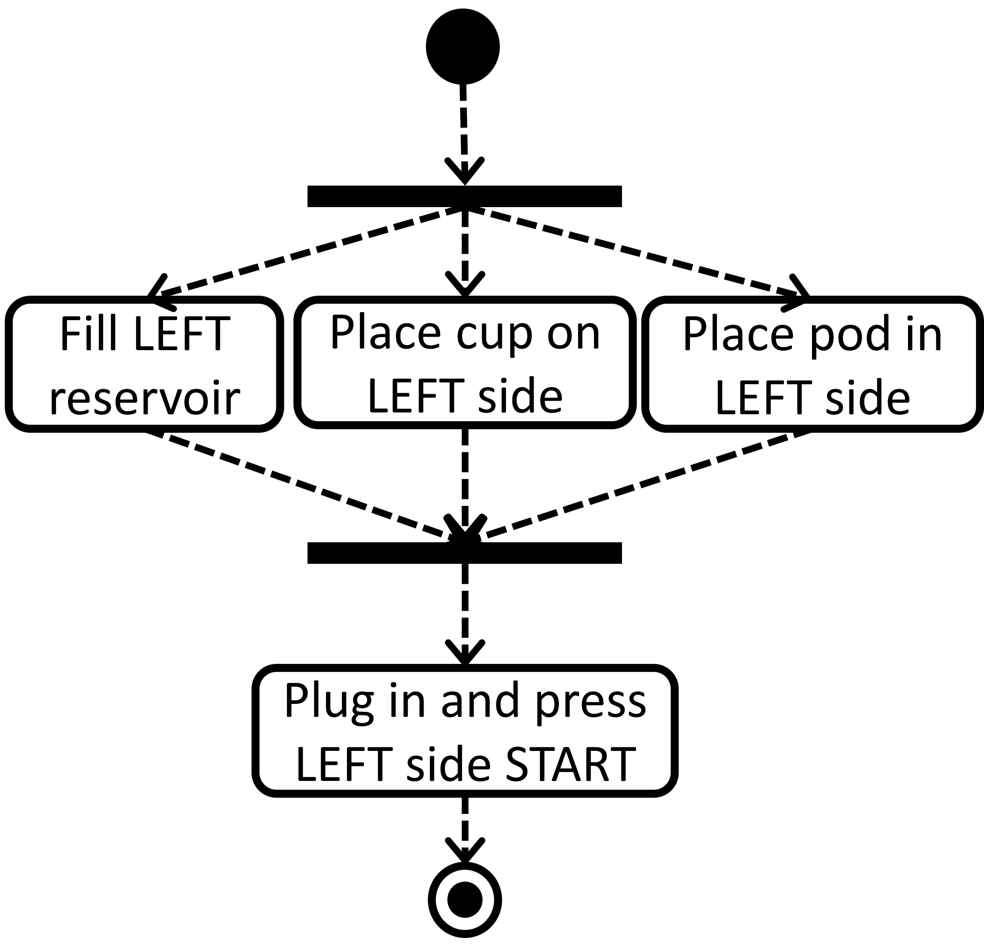
Example: Coffee Making Process



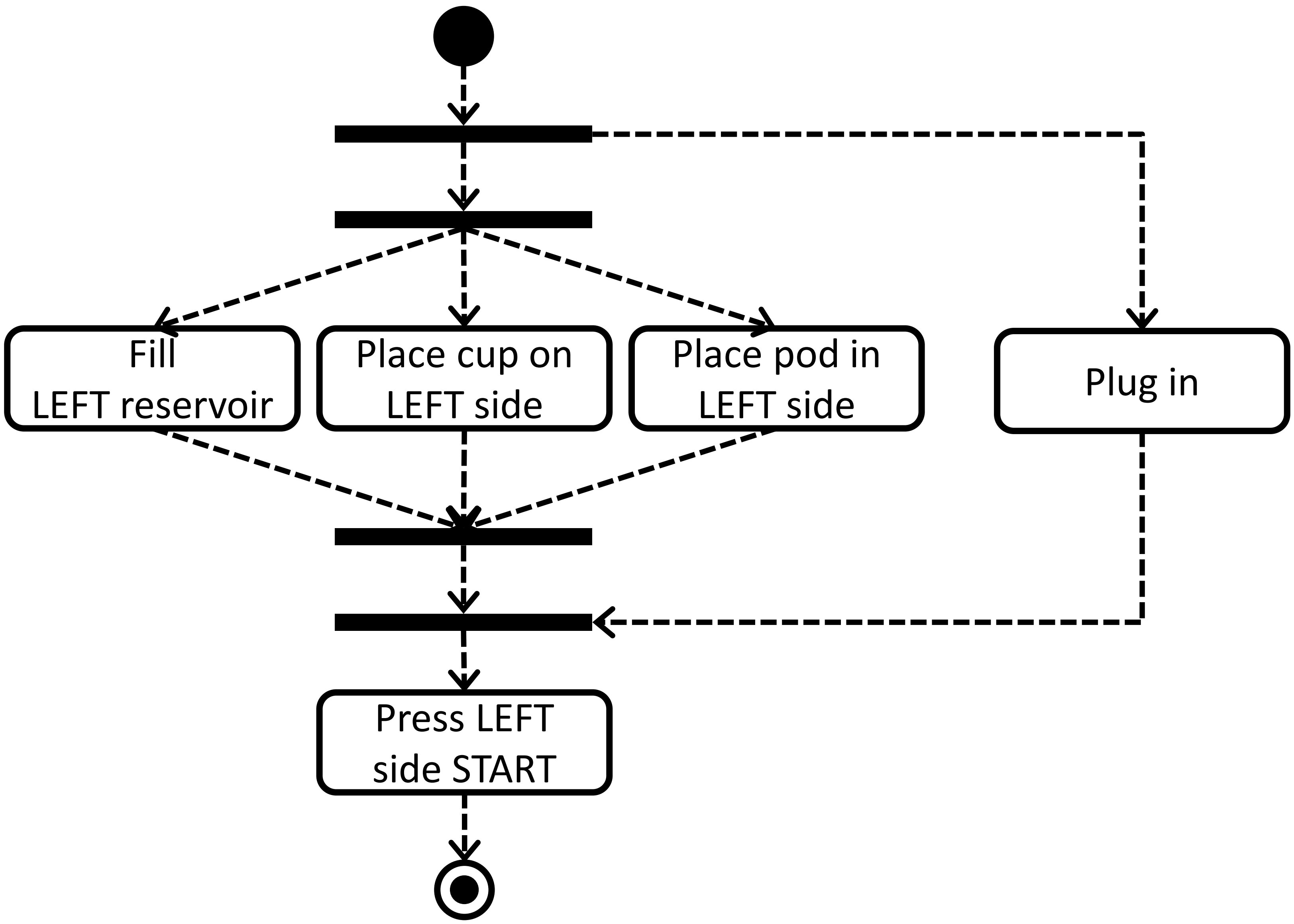
Comparison

- State machine
- Process

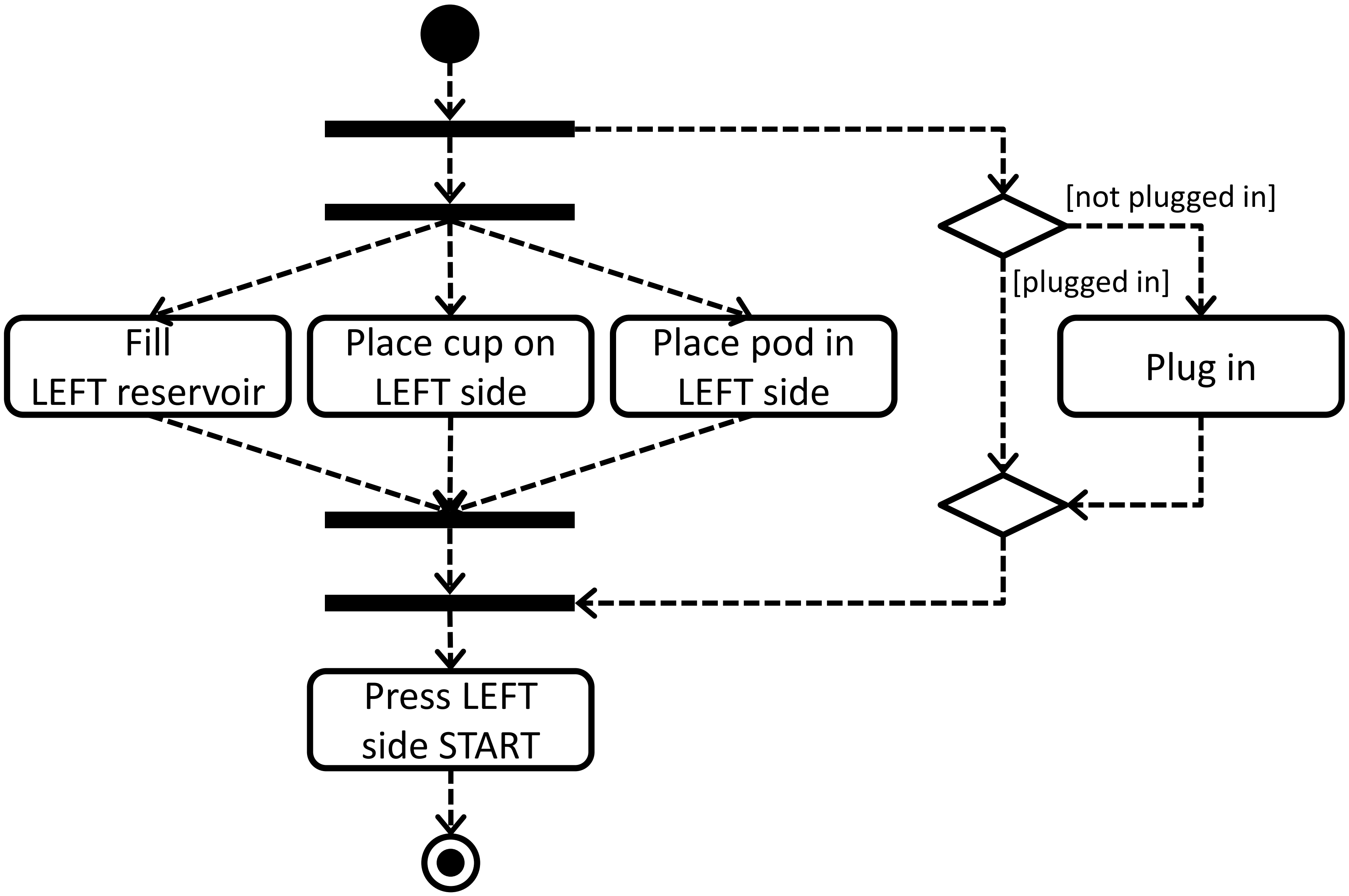




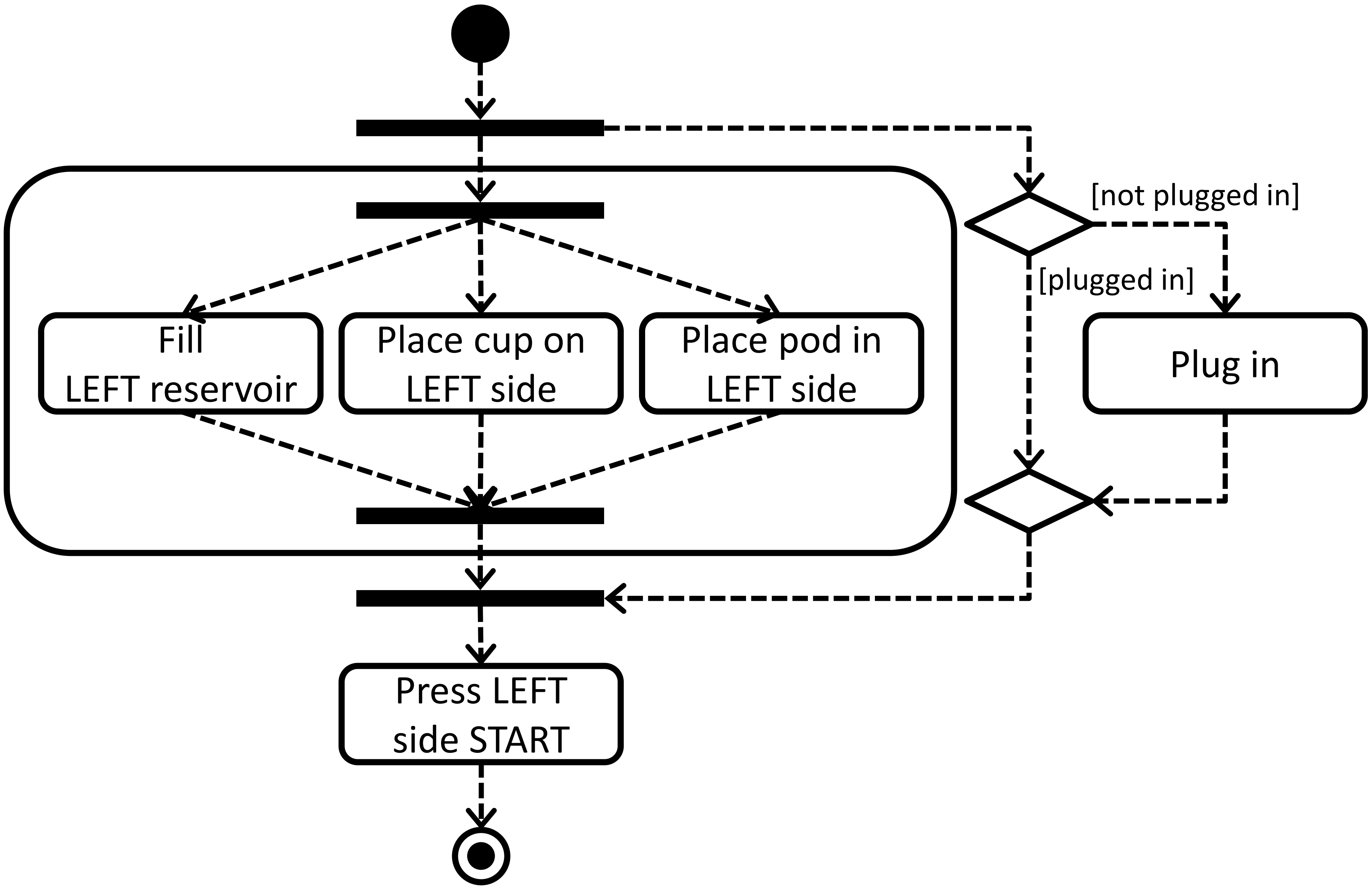
Example: Coffee Making Process



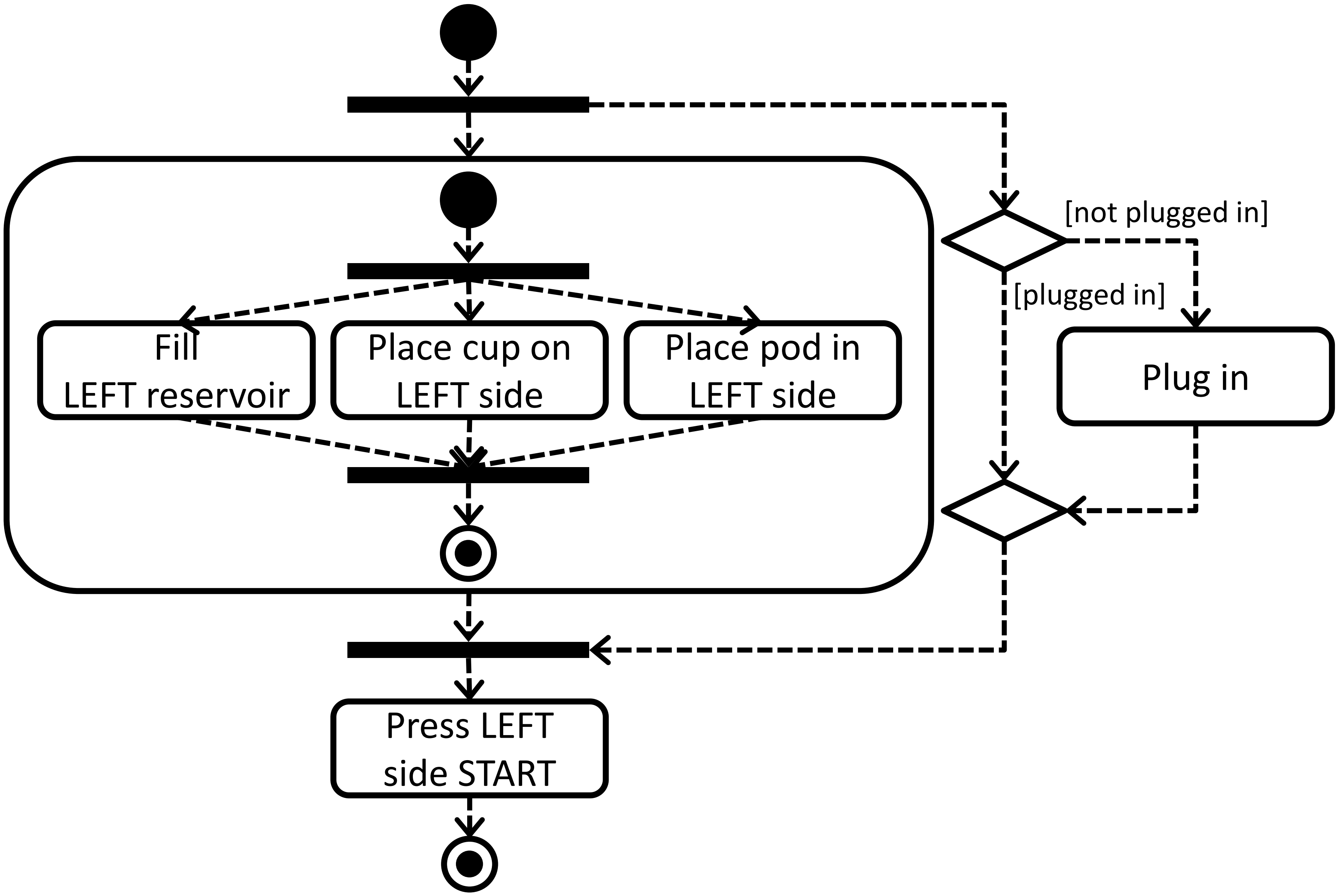
Example: Coffee Making Process



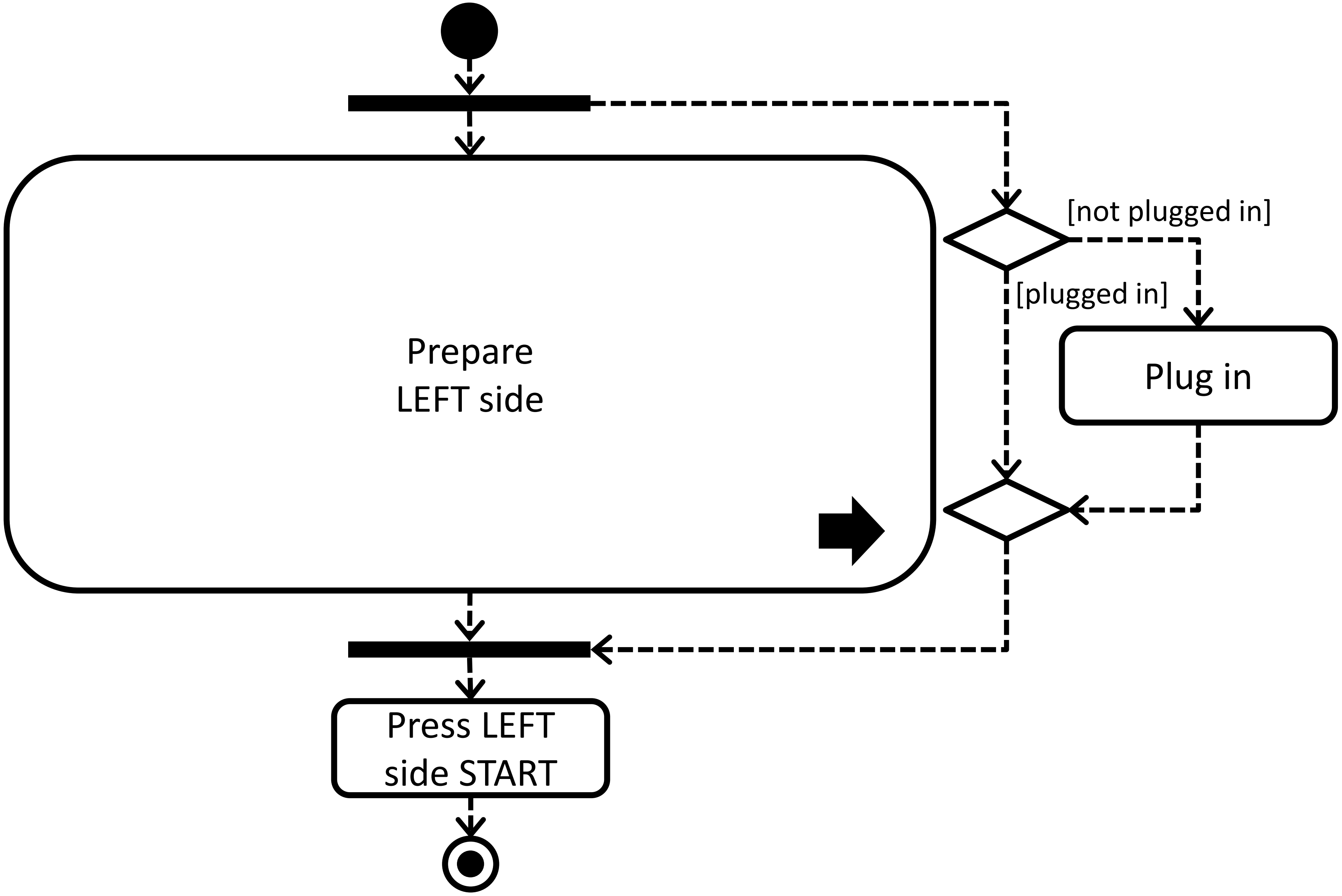
Example: Coffee Making Process



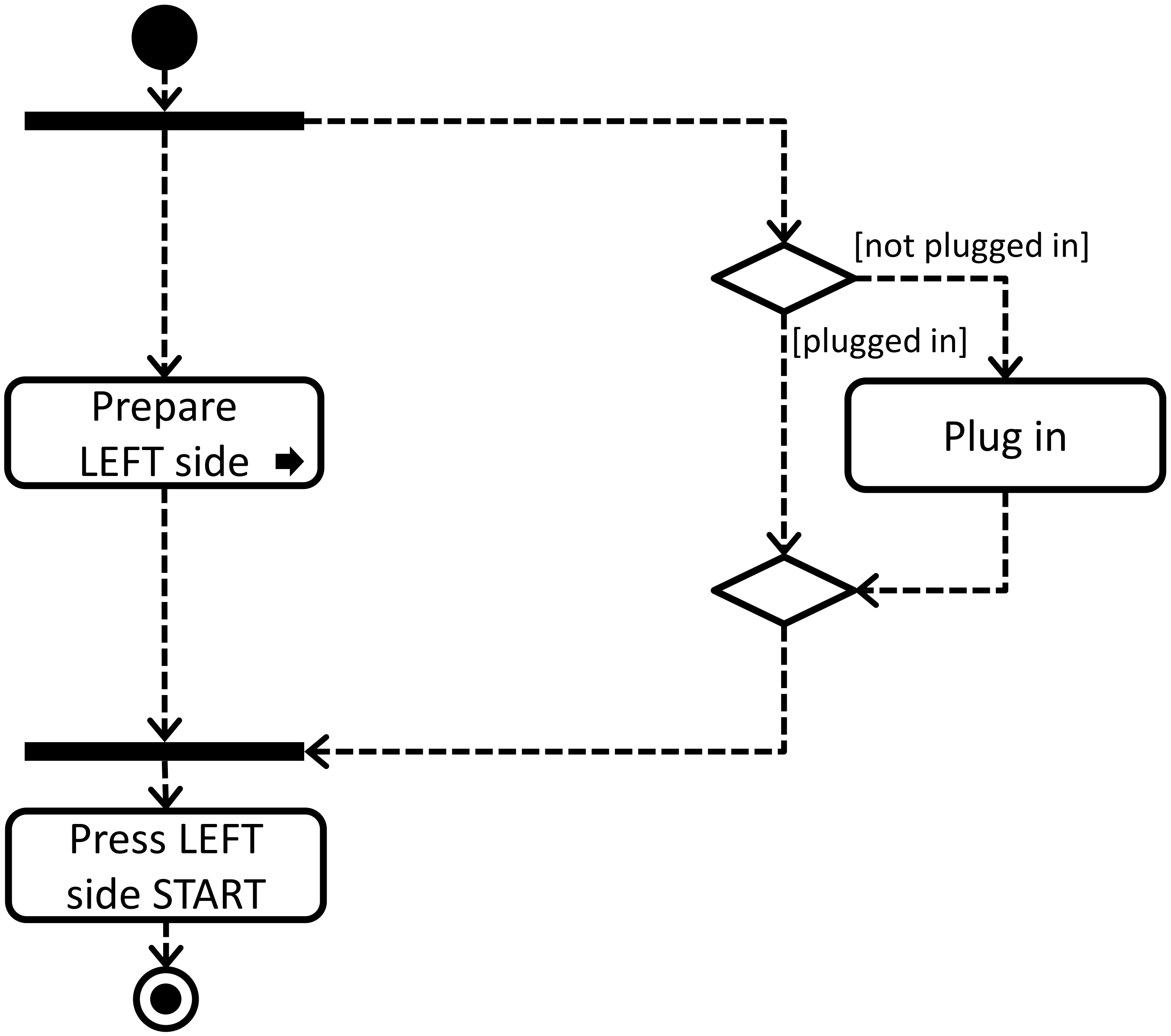
Example: Coffee Making Process



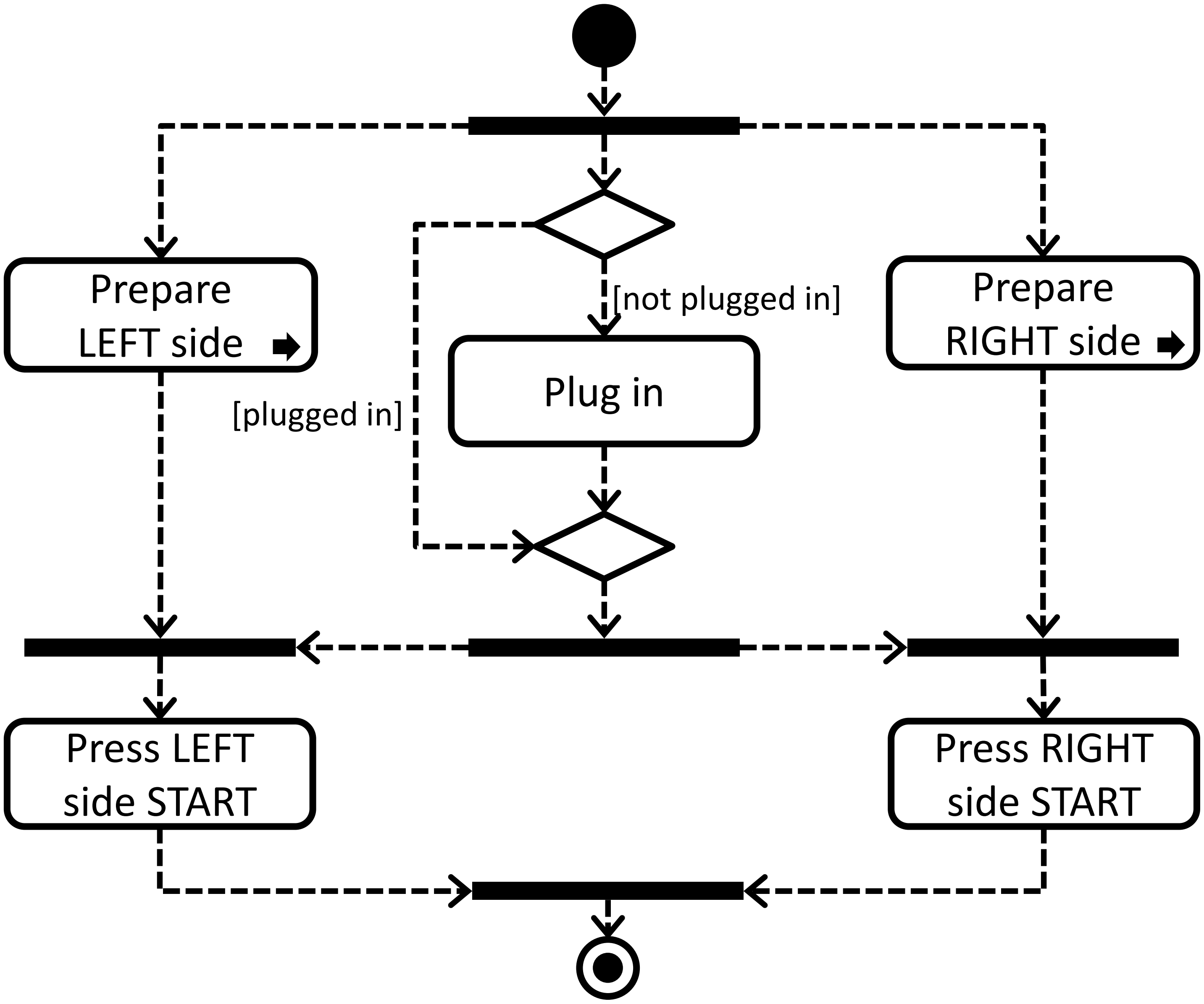
Example: Coffee Making Process



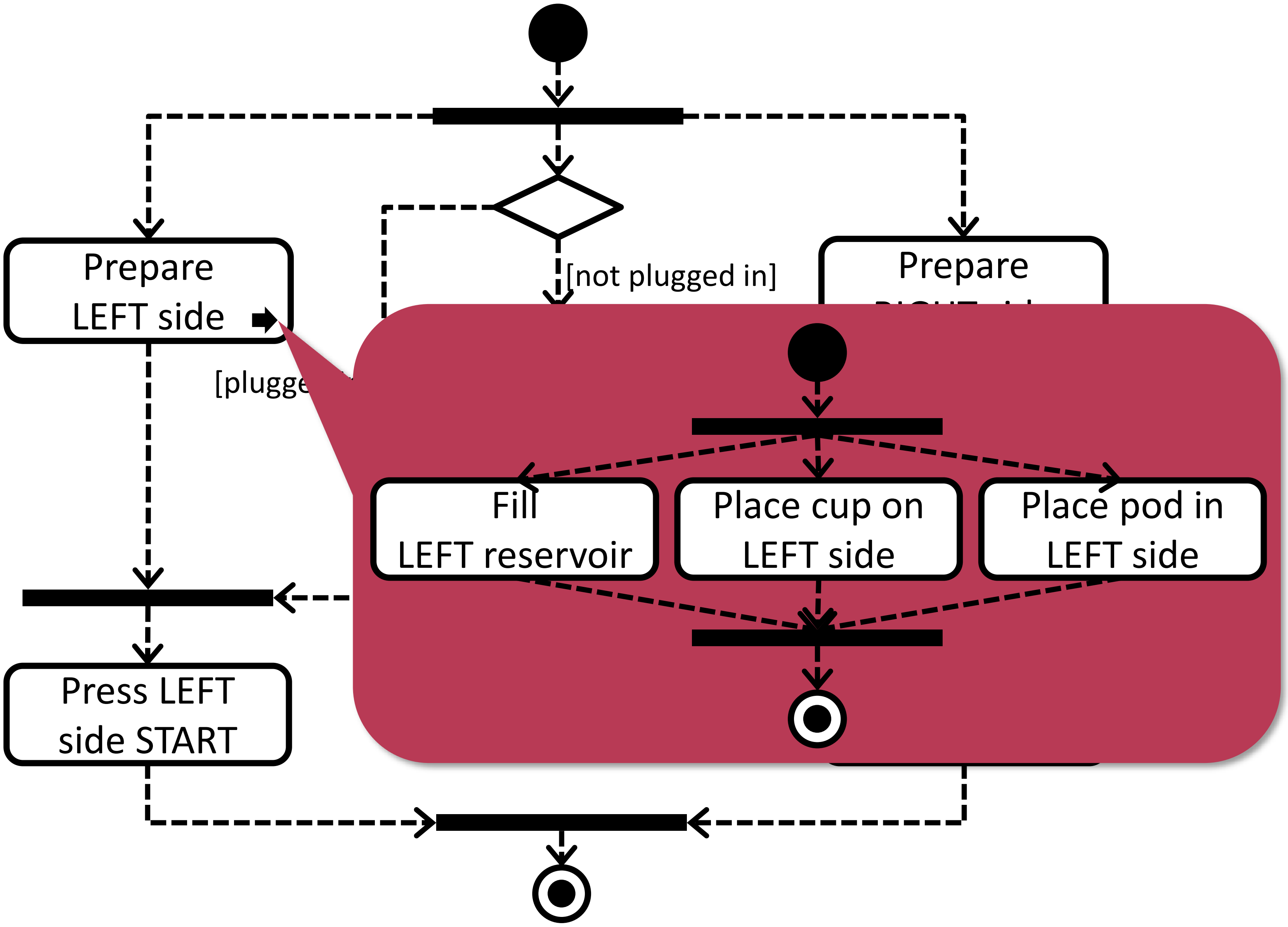
Example: Coffee Making Process



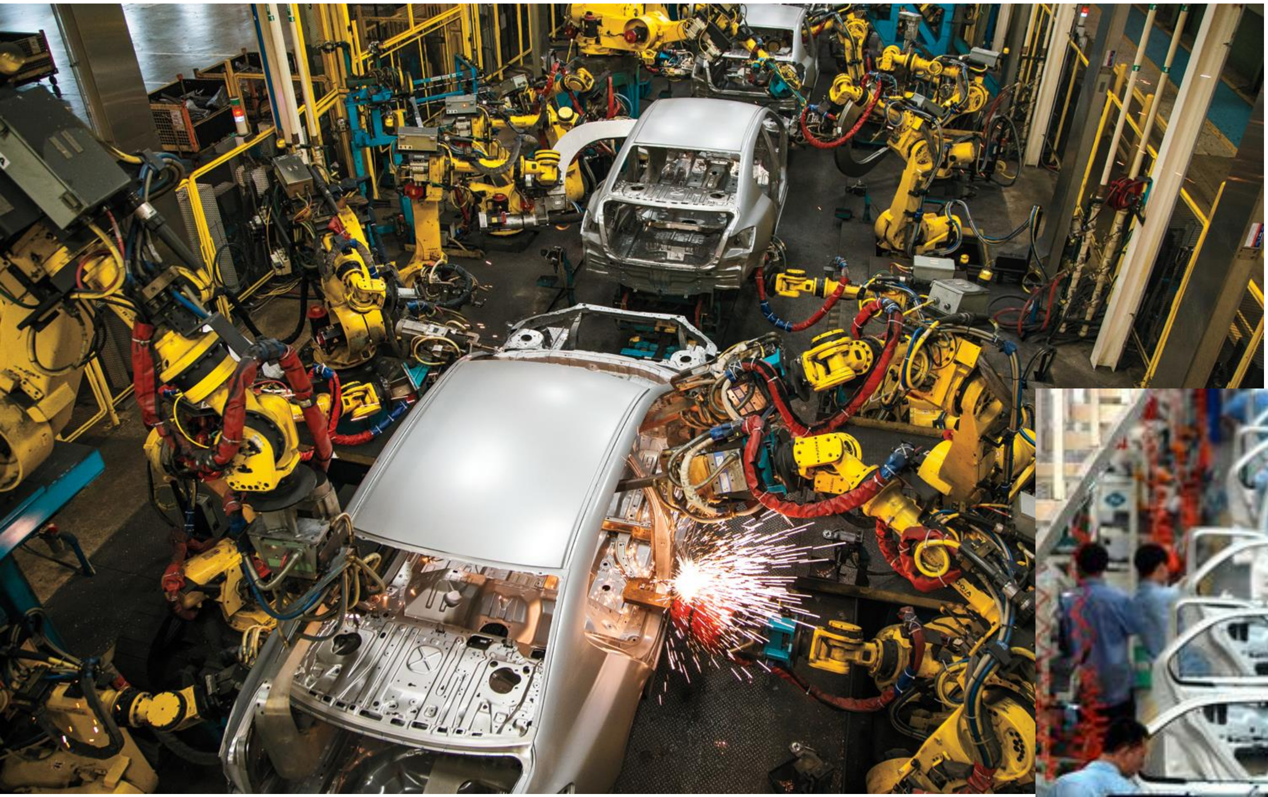
Example: Coffee Making Process



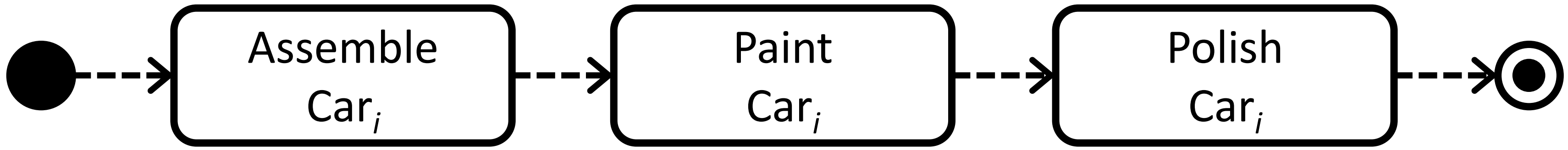
Example: Coffee Making Process



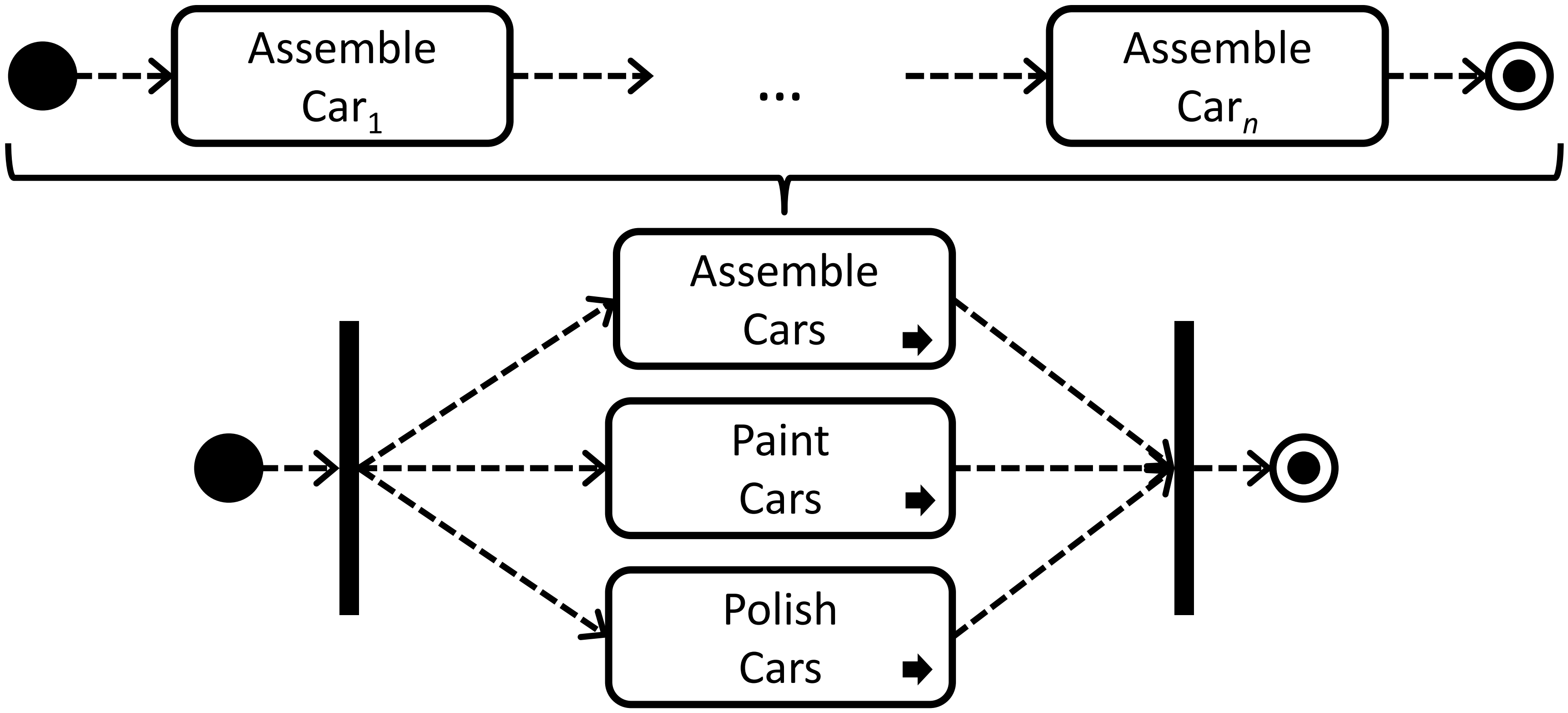
Modeling based on different aspects



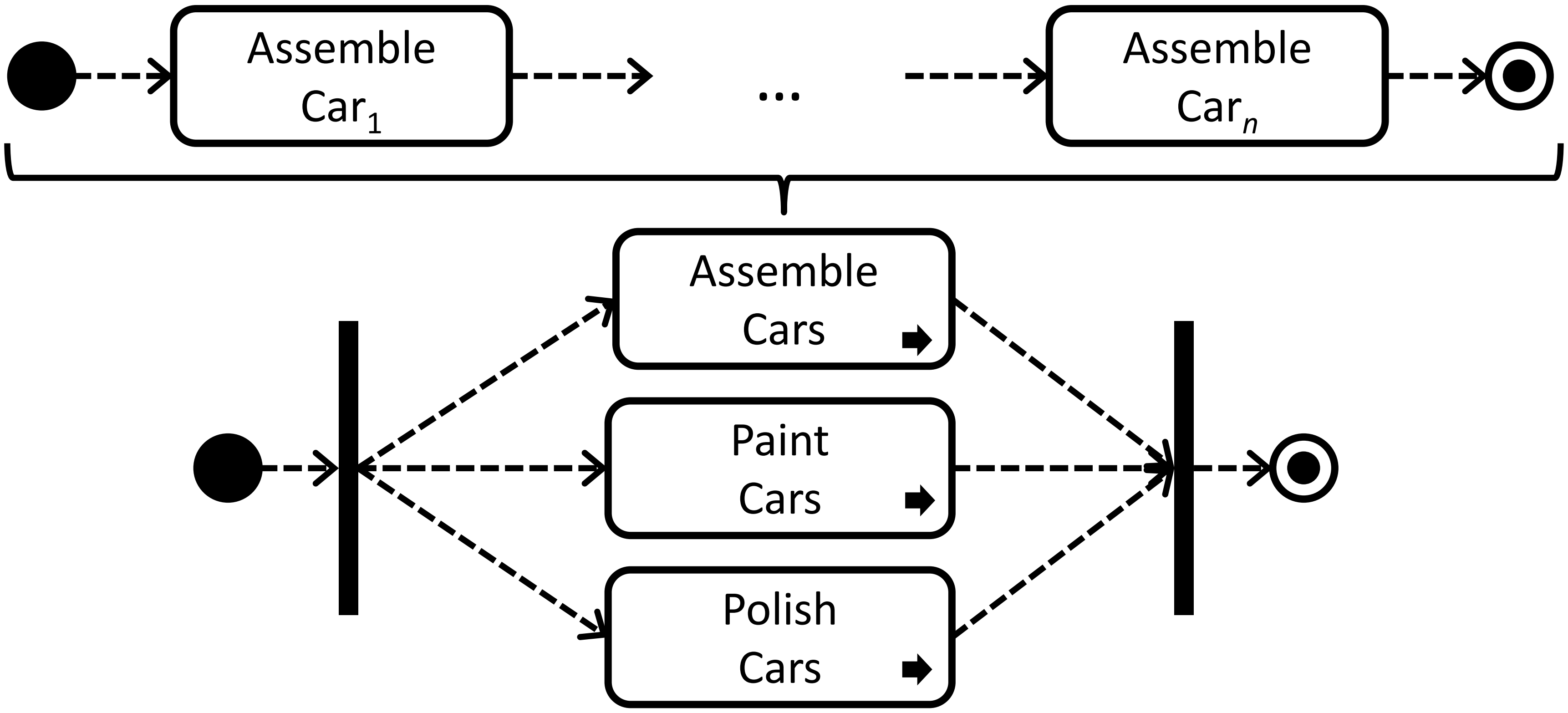
What happens to a car?

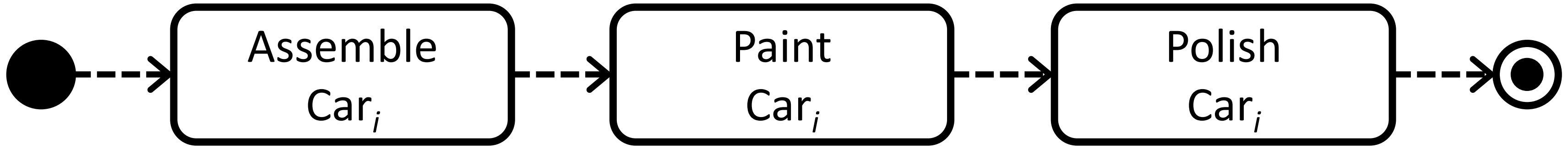


What happens on the production line?

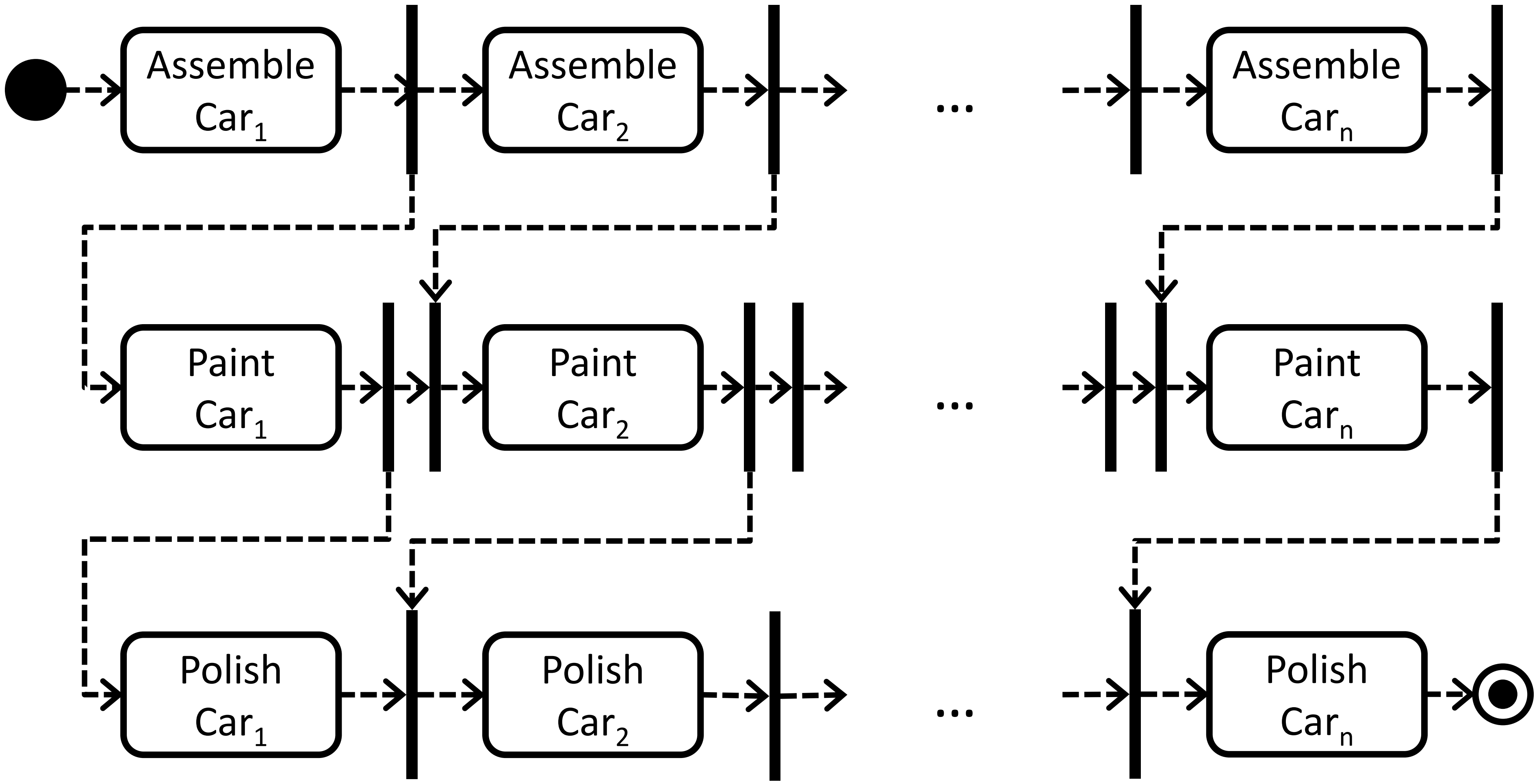


Modeling based on different aspects



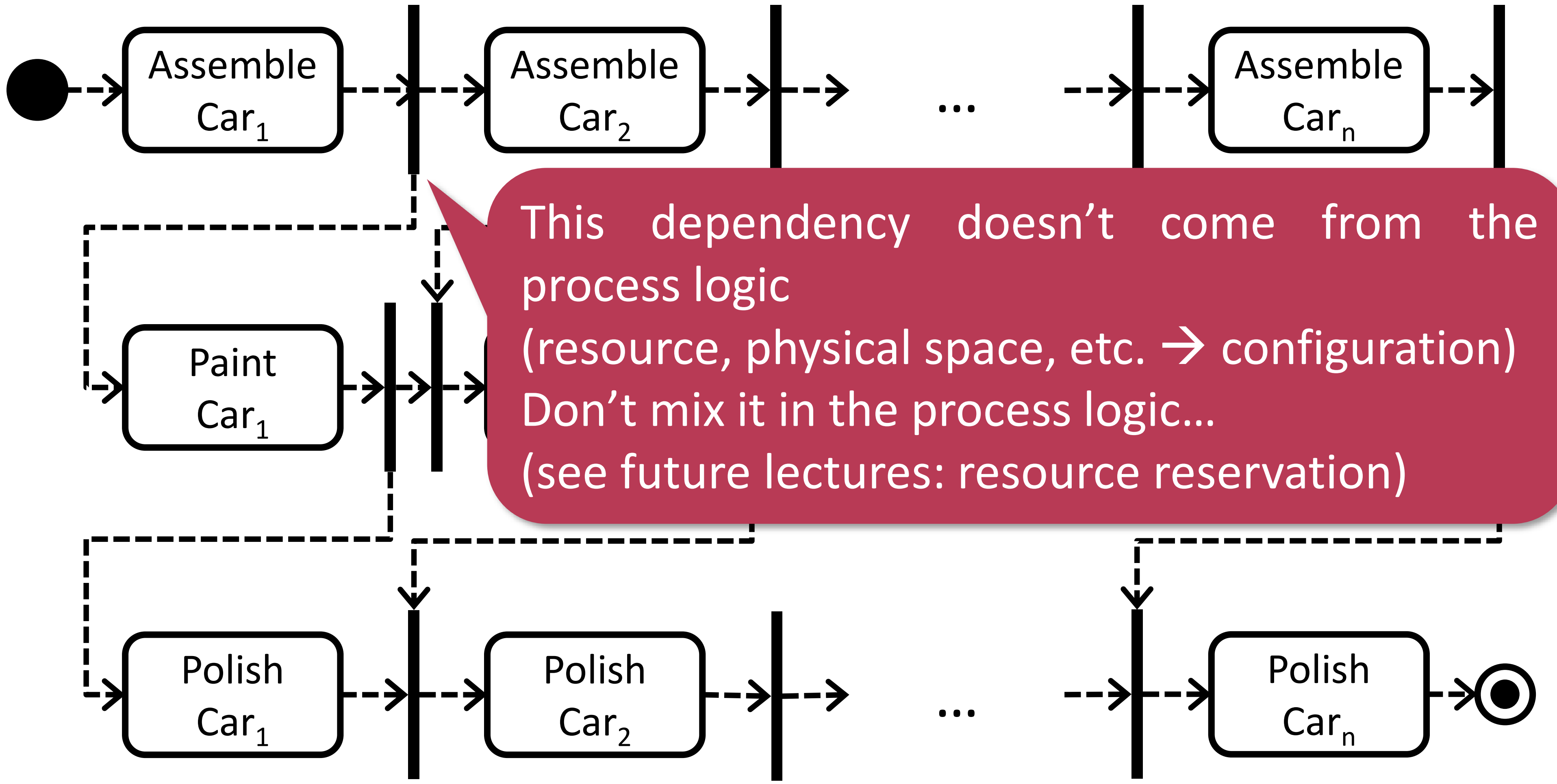


Joint View



- Includes everything but not very practical

Joint View



This dependency doesn’t come from the process logic
(resource, physical space, etc. → configuration)
Don’t mix it in the process logic…
(see future lectures: resource reservation)

- Includes everything but not very practical

Joint View

- 2D fork-join net isn't very practical
  - Different processes for different aspects (car's and machine's lifetime)
- Multiple fork-join pairs in a compact way?
  → PERT chart
  - Program Evaluation and Review Technique
    - For analyzing execution time
    - (No branching here)

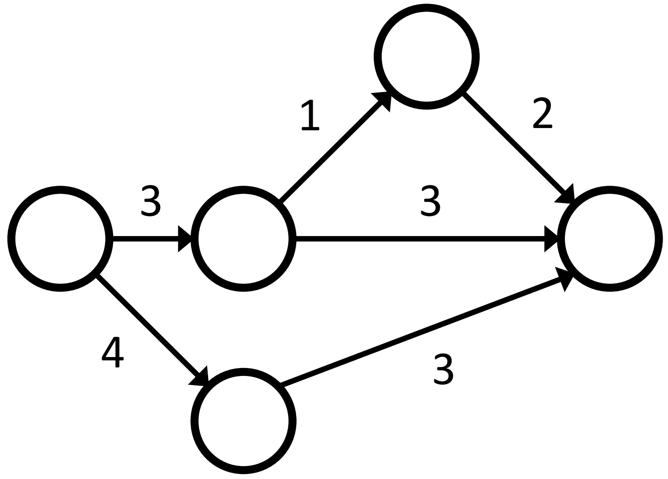
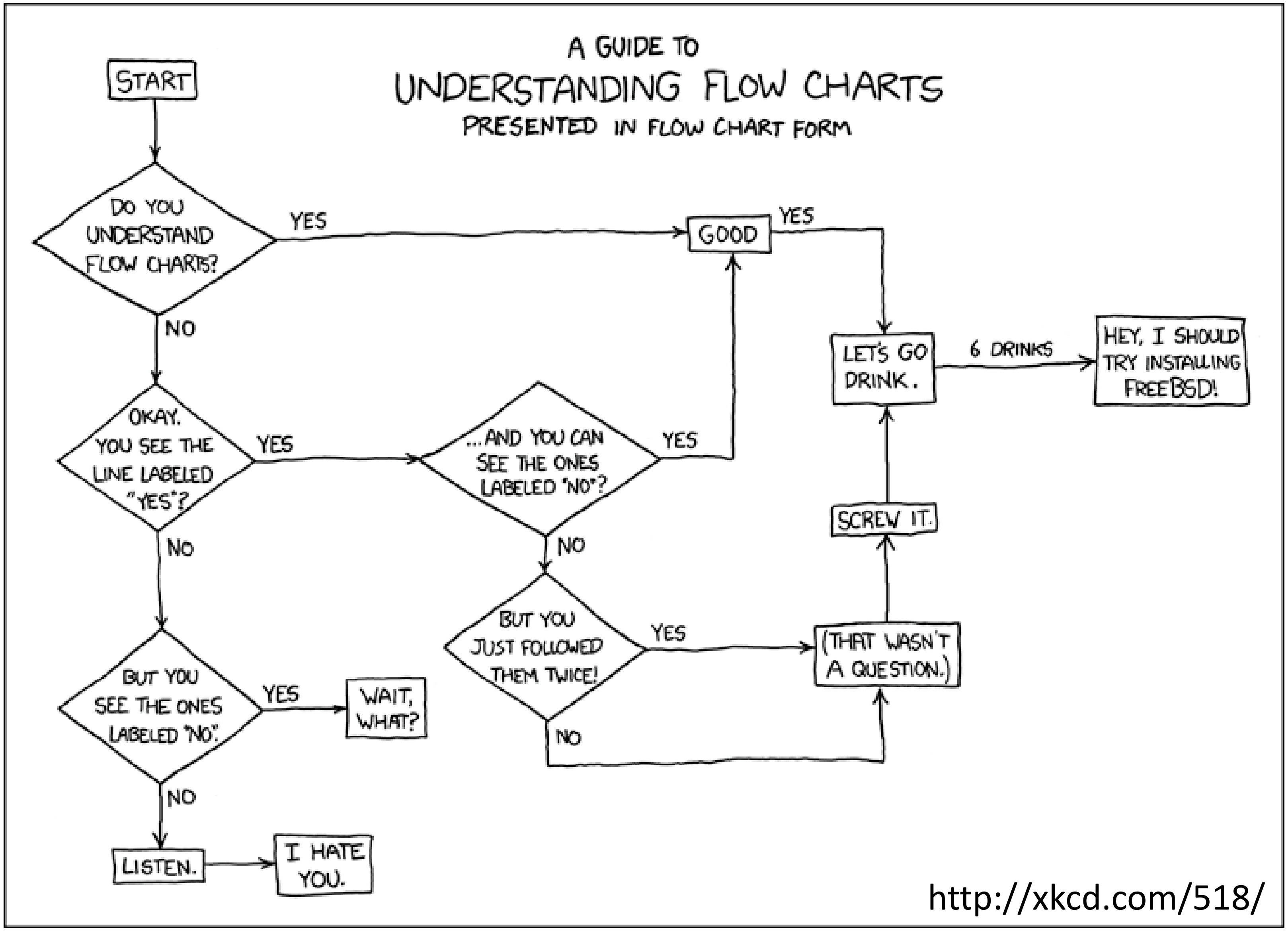


Table of contents

Flowchart

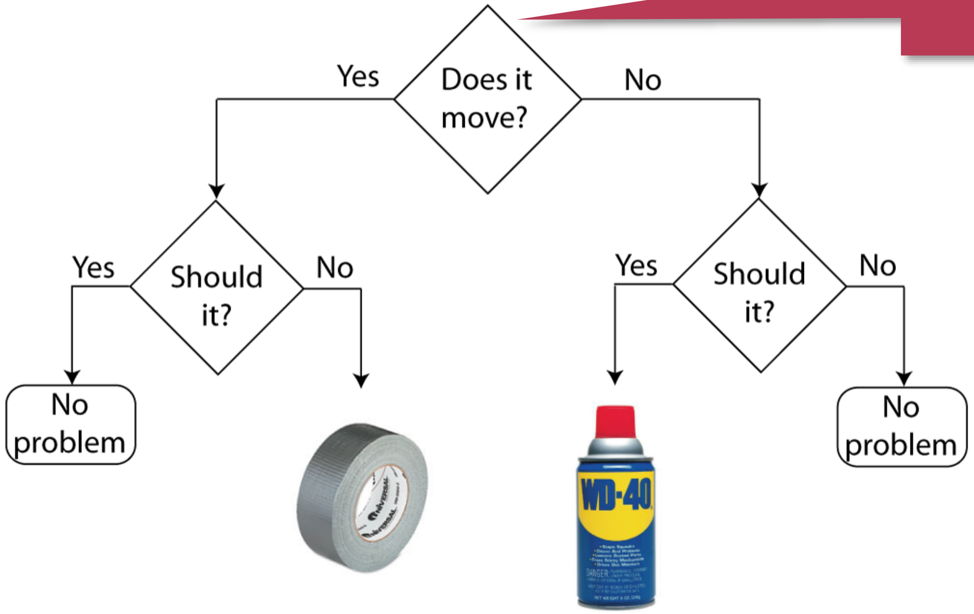

A GUIDE TO
UNDERSTANDING FLOW CHARTS
PRESENTED IN FLOW CHART FORM
START
DO YOU UNDERSTAND FLOW CHARTS?
YES
GOOD
YES
NO
LET'S GO DRINK.
6 DRINKS
HEY, I SHOULD TRY INSTALLING FREEBSD!
OKAY. YOU SEE THE LINE LABELED "YES"?
YES
...AND YOU CAN SEE THE ONES LABELED "NO"?
YES
NO
SCREW IT.
NO
BUT YOU JUST FOLLOWED THEM TWICE!
YES
(THAT WASN'T A QUESTION.)
NO
BUT YOU SEE THE ONES LABELED "NO".
YES
WAIT, WHAT?
NO
LISTEN.
I HATE YOU.

http://xkcd.com/518/

Flowchart

- Flowchart / decision diagram
  - Describes a train of thought for decision making
    - Leads to a conclusion
  - No temporal sequence
- Special case: decision tree

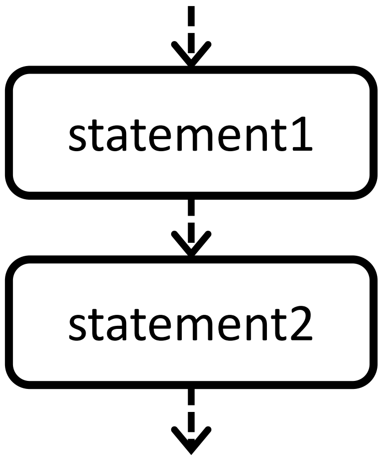
Describing decision points and their order is difficult for real problems



Control Flow

<statement1>

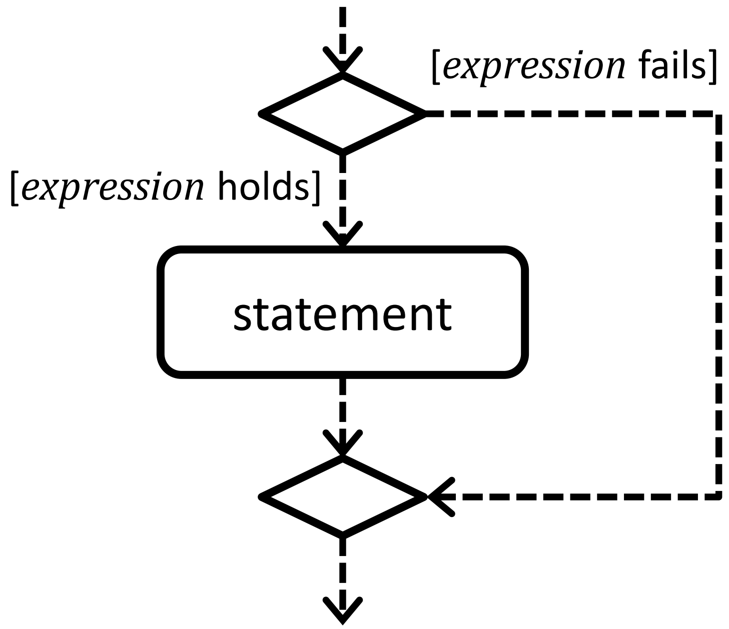
<statement2>



Control Flow

`if` (*<expression>*)

  <statement>



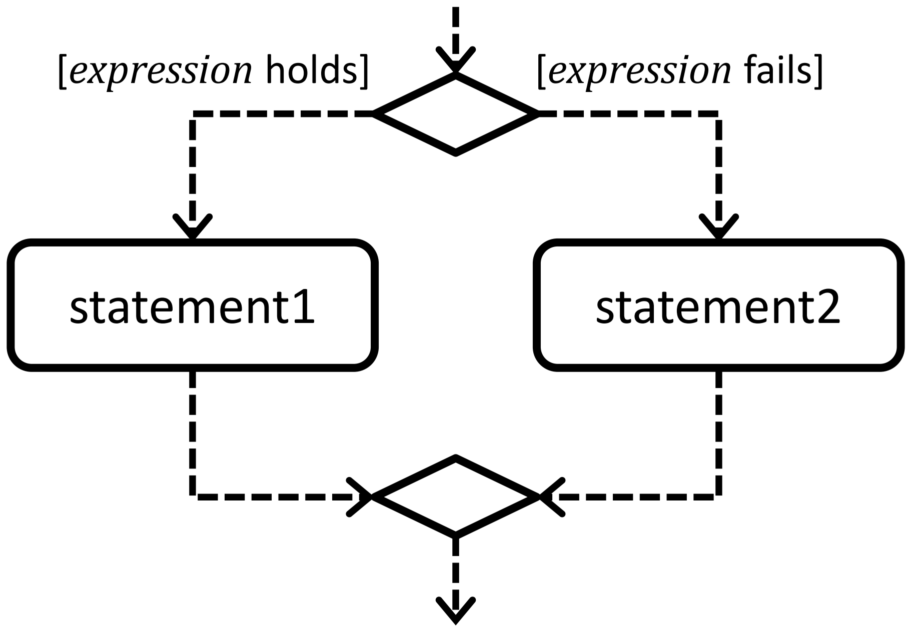
Control Flow

if (*<expression>*)

    <statement1>

else

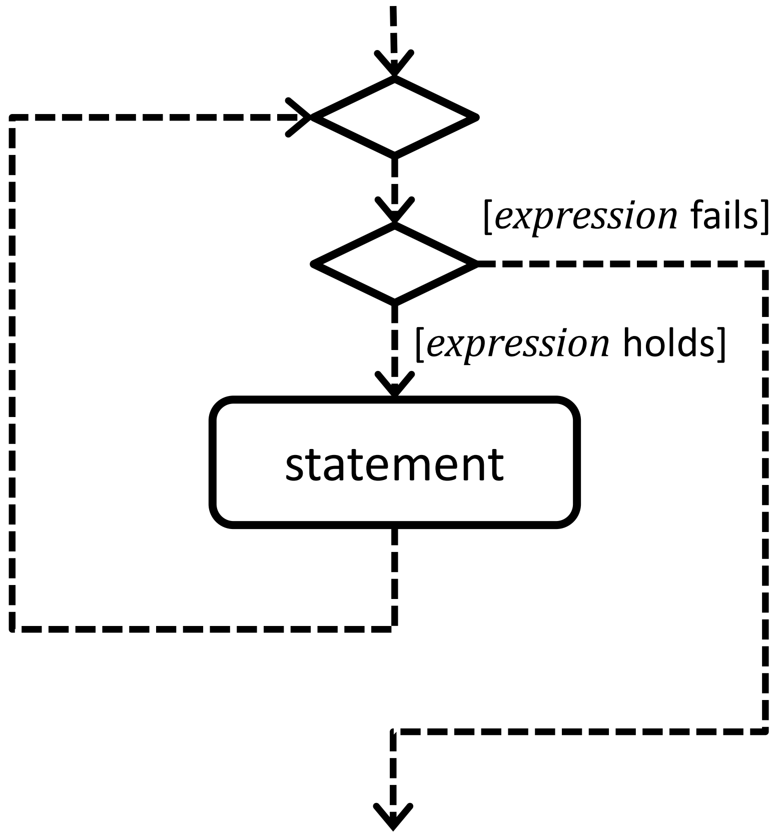
    <statement2>



Control Flow

`while` (*<expression>*)

  <statement>

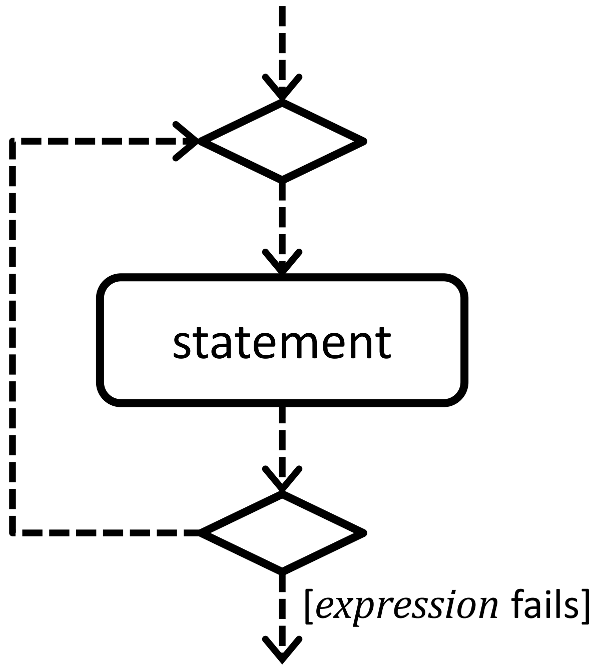


Control Flow

```
do
```

<statement>

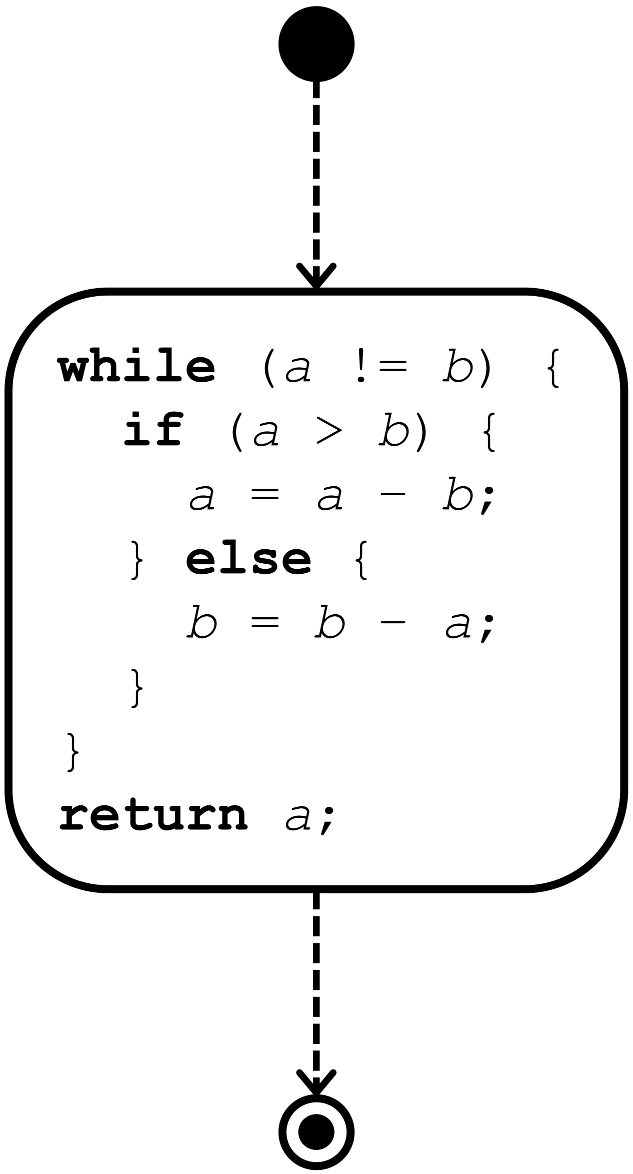
`while` (*<expression>*)



Control Flow - Example

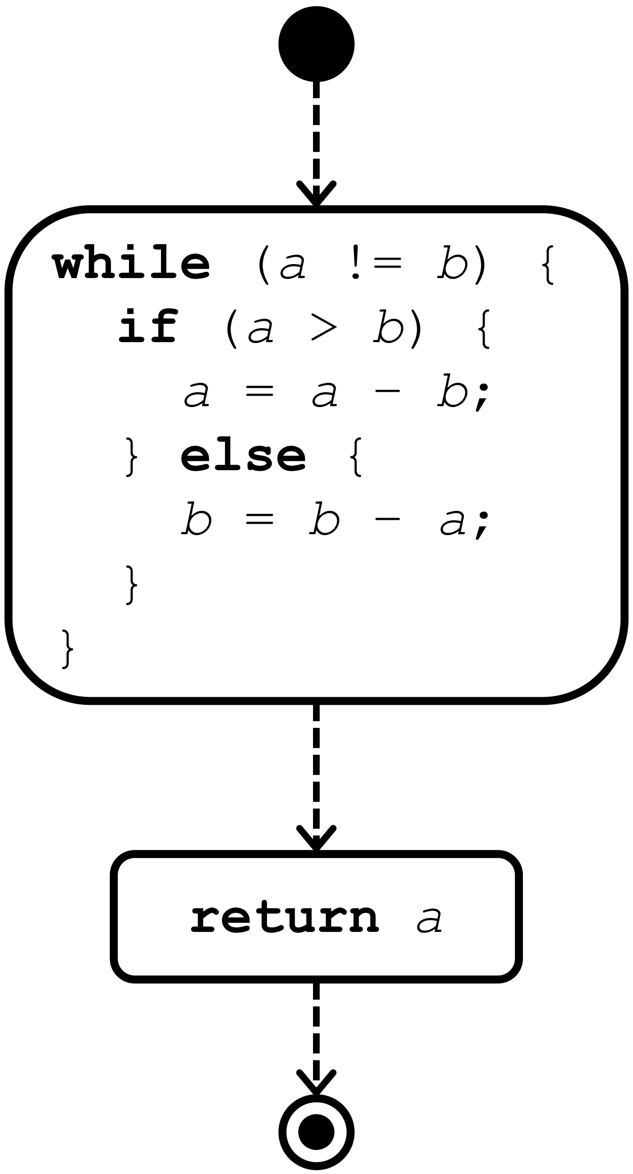
```
while (a != b) {
  if (a > b) {
    a = a - b;
  } else {
    b = b - a;
  }
}
return a;
```

Control Flow - Example



```
while (a != b) {
  if (a > b) {
    a = a - b;
  } else {
    b = b - a;
  }
}
return a;
```

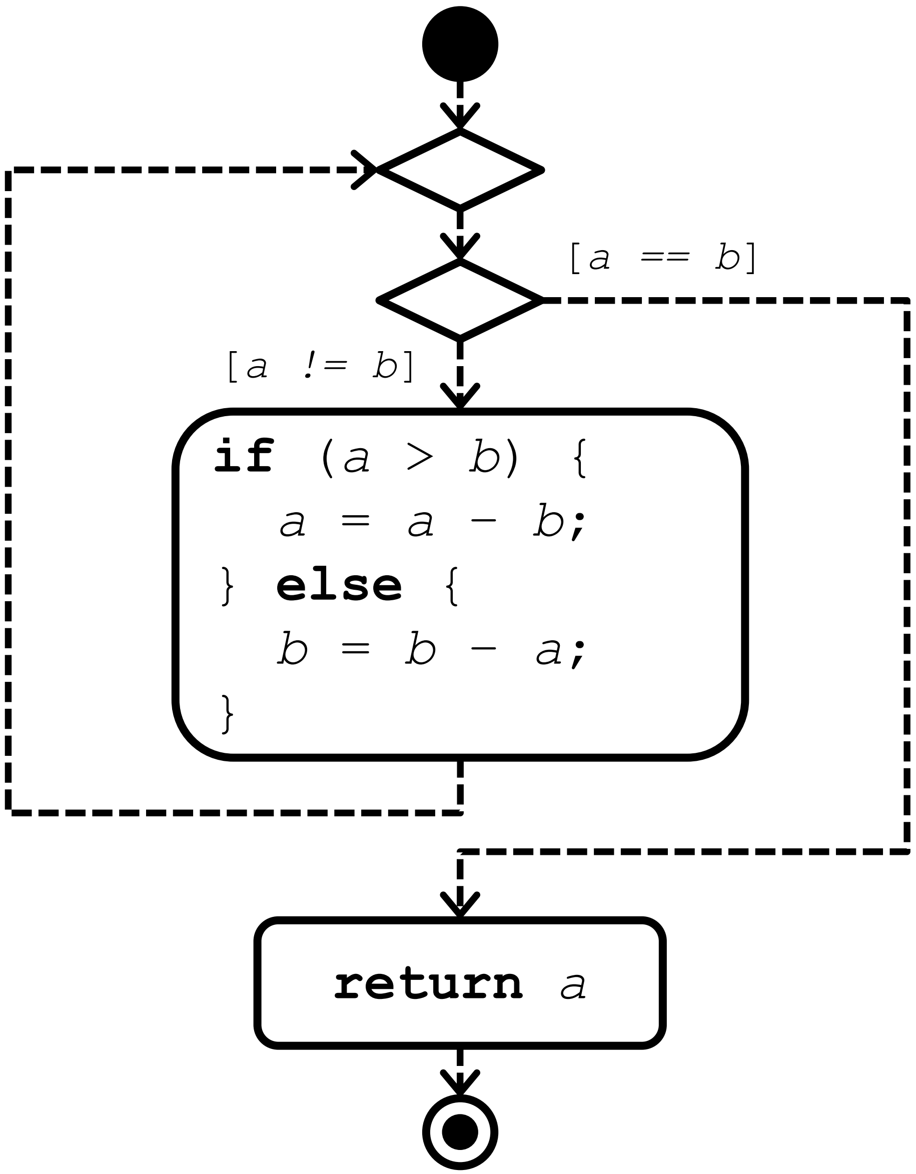
Control Flow - Example



```
while (a != b) {
  if (a > b) {
    a = a − b;
  } else {
    b = b − a;
  }
}
```

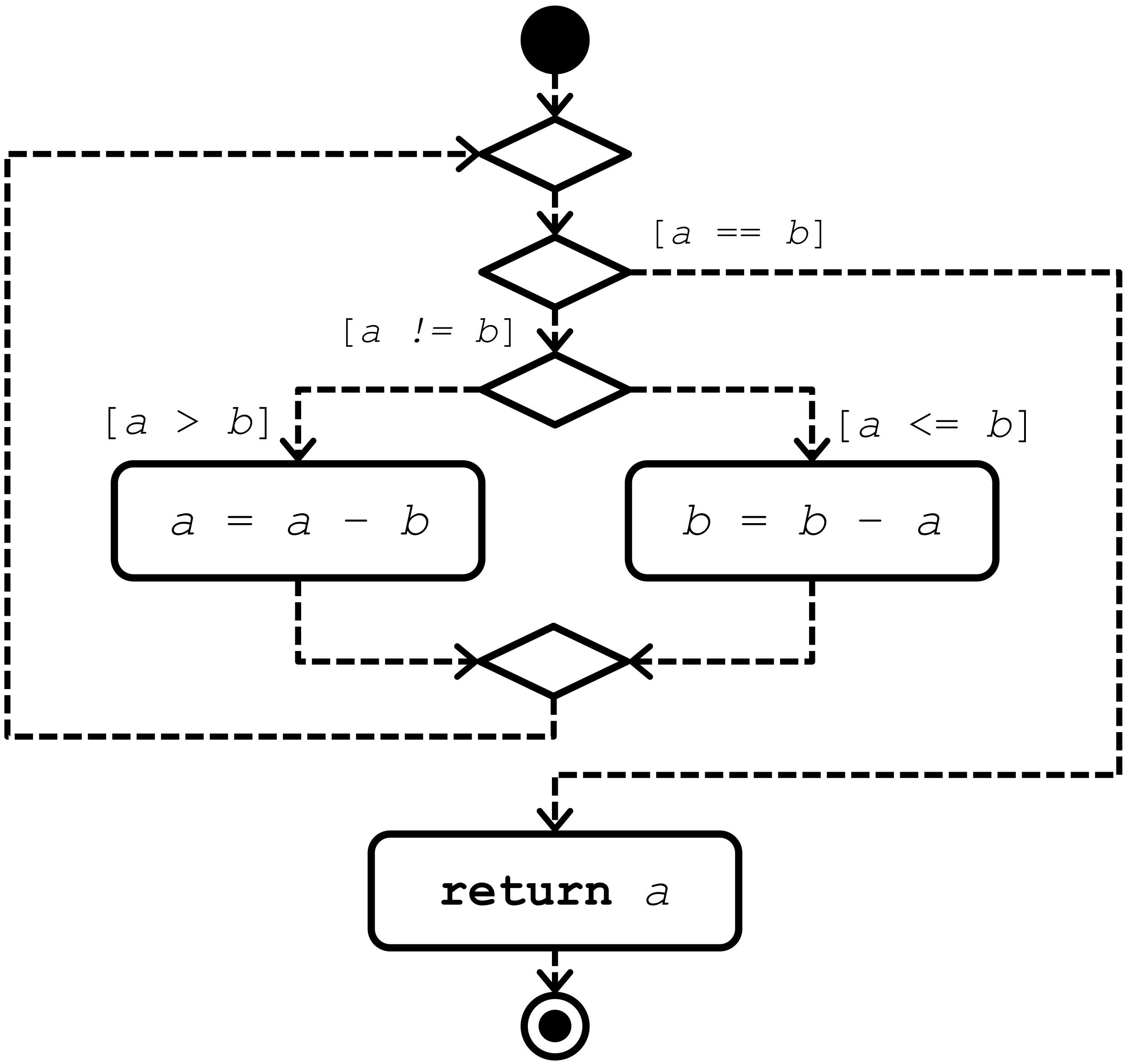
```
return a
```

Control Flow - Example

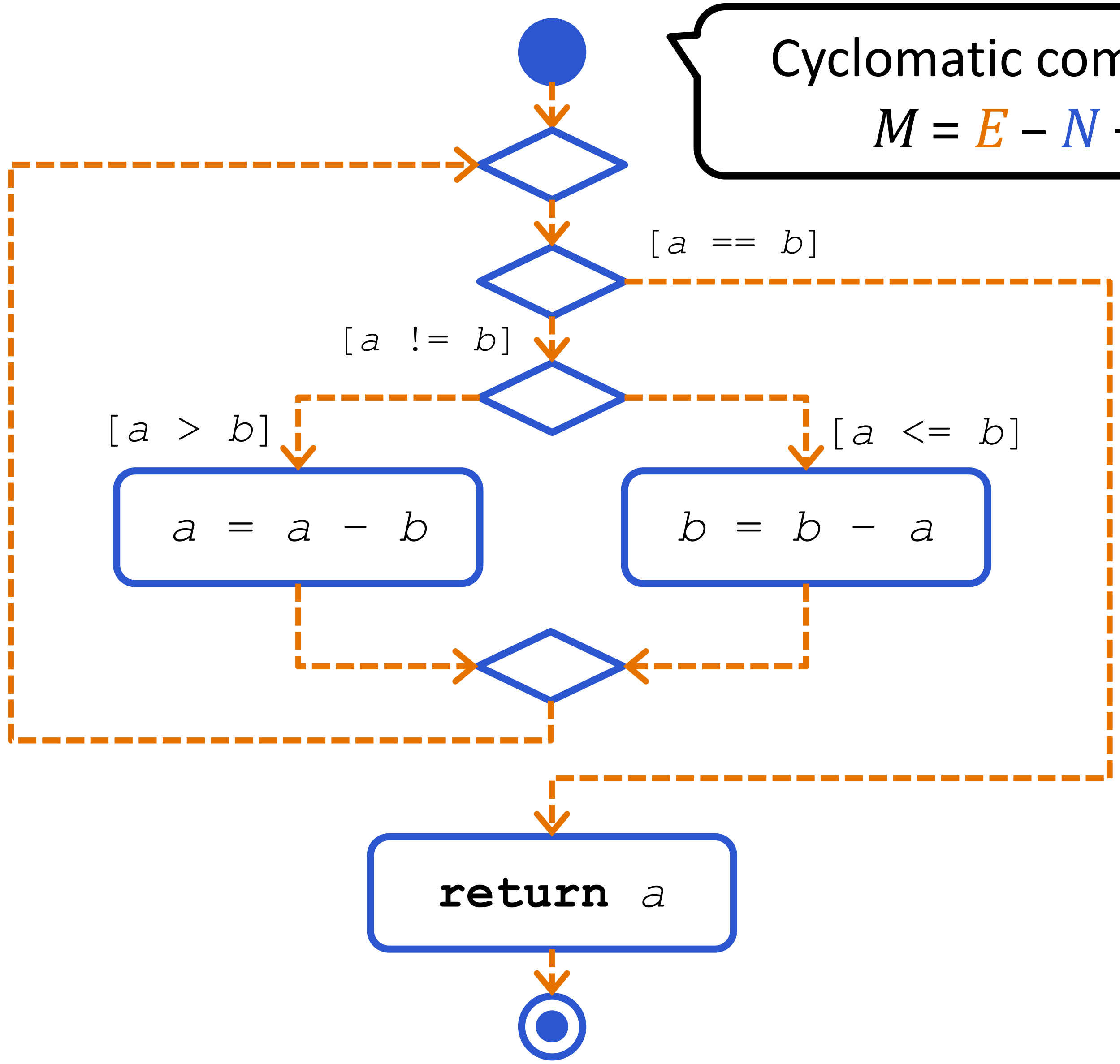


```
if (a > b) {
   a = a - b;
} else {
   b = b - a;
}
```

Control Flow - Example



Control Flow - Complexity



Cyclomatic complexity

$M = E - N + 2$

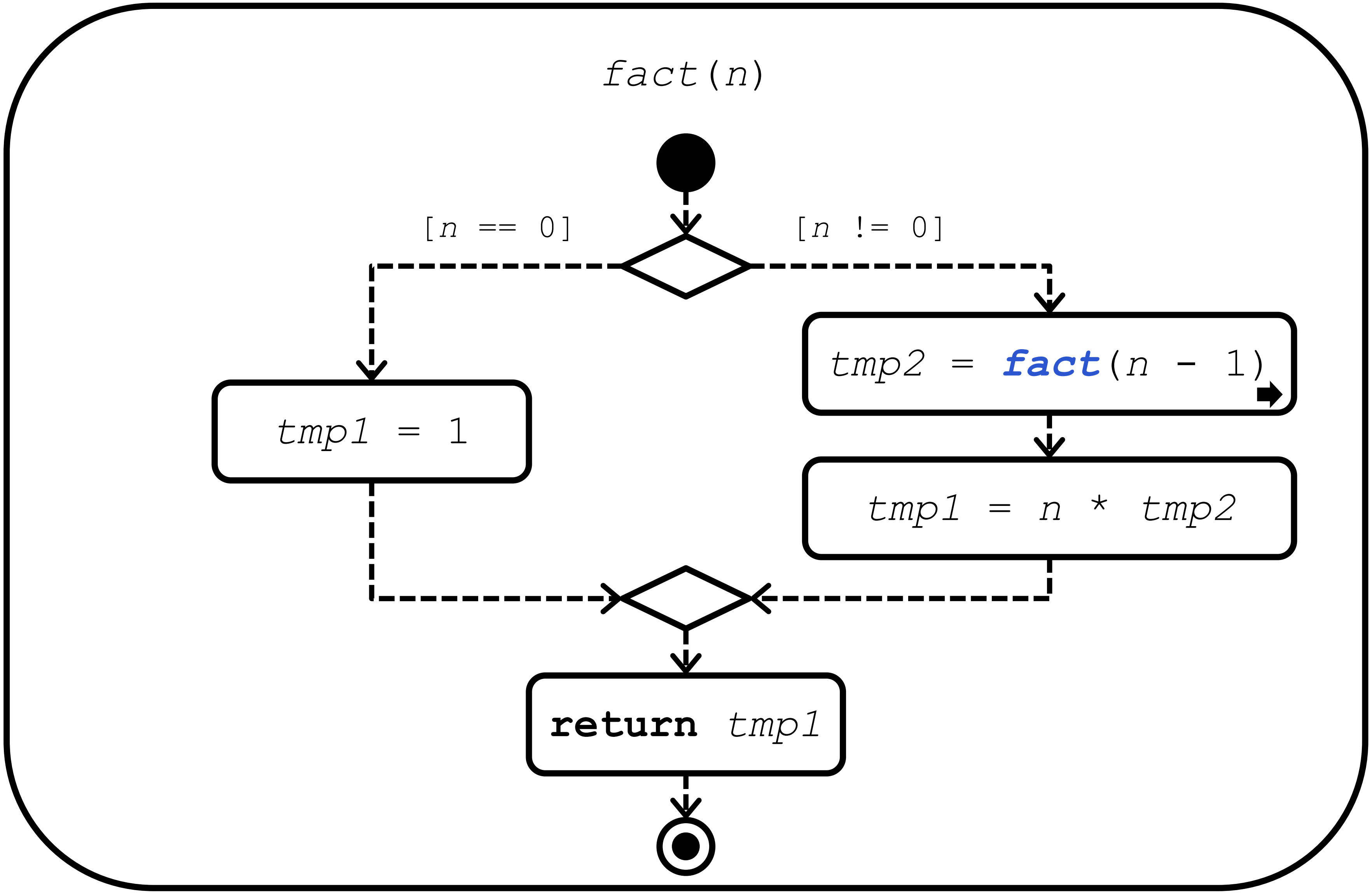
Control Flow - Recursion

```
int fact(int n) {
  return
    (n == 0) ? 1 : n * fact(n - 1);
}
```

Control Flow - Recursion

```
int fact(int n) {
  int tmp1;
  if (n == 0) {
    tmp1 = 1;
  } else {
    int tmp2 = fact(n - 1);
    tmp1 = n * tmp2;
  }
  return tmp1;
}
```

Control Flow - Recursion



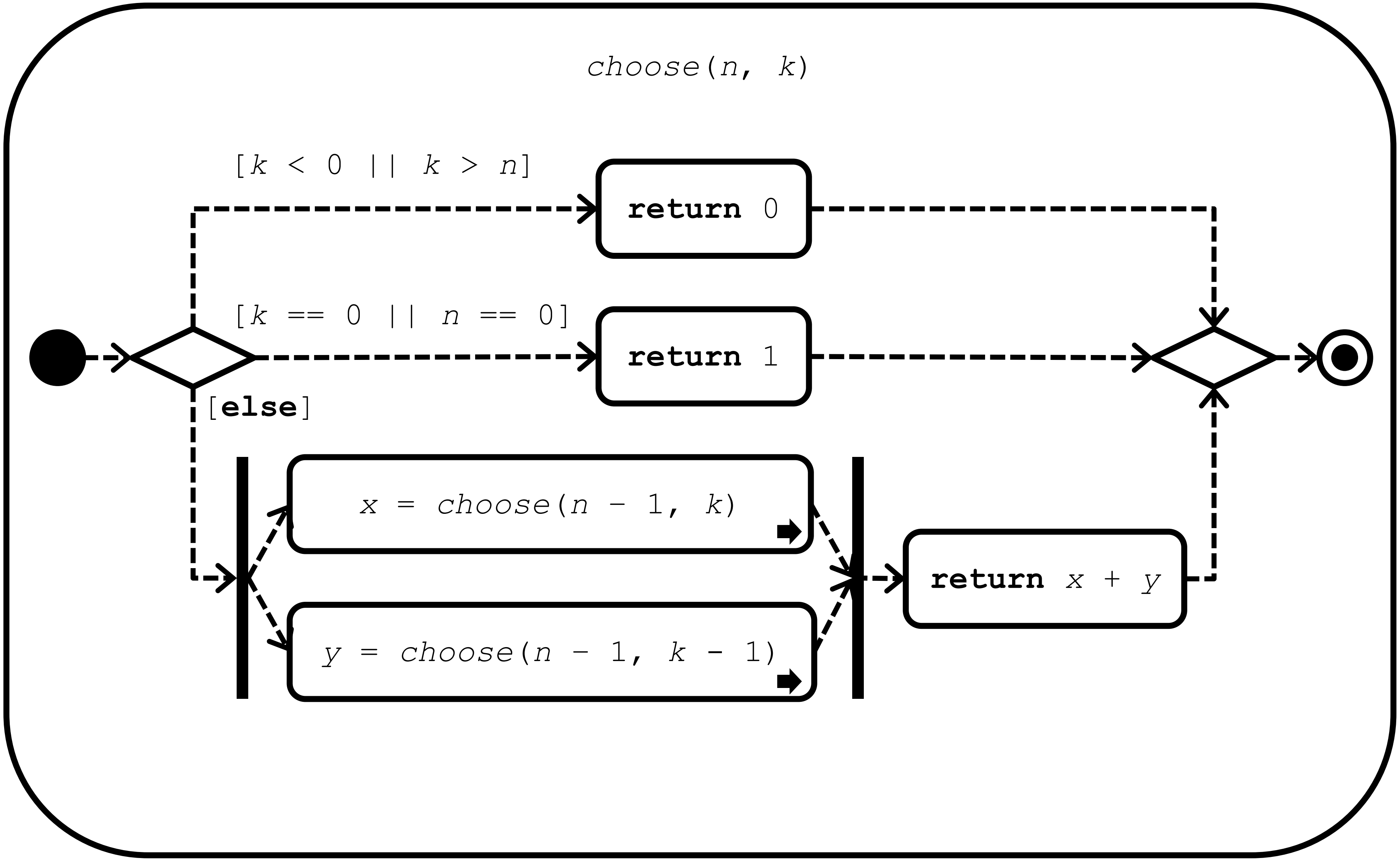
Example: *n* choose *k*

```
int choose(int n, int k) {
  if (k < 0 || k > n) {
    return 0;
  } else if (k == 0 && n == 0) {
    return 1;
  } else {
    int x = spawn choose(n - 1, k);
    int y = spawn choose(n - 1, k - 1);
    sync;
    return x + y;
  }
}
```

$\binom{0}{0}$=1

$\binom{n}{k}$=$\binom{n-1}{k} + \binom{n-1}{k-1}$

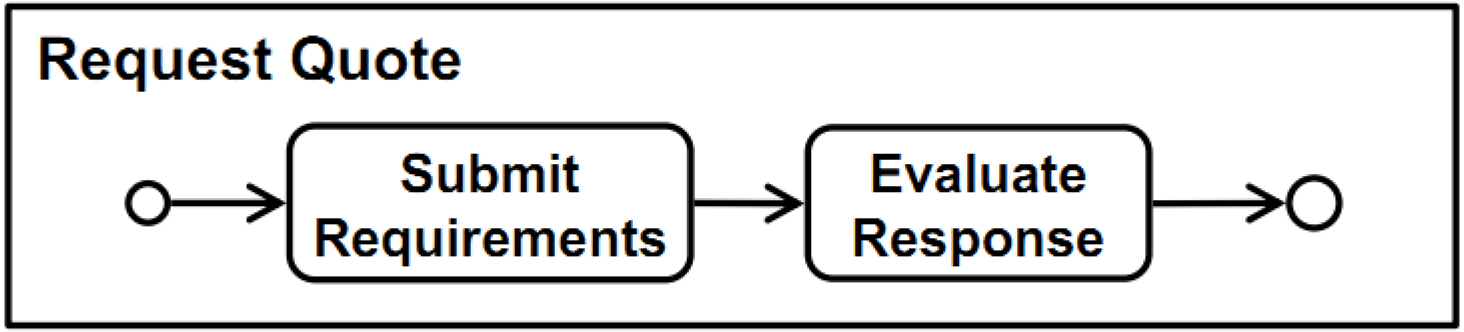
Example: *n* choose *k*



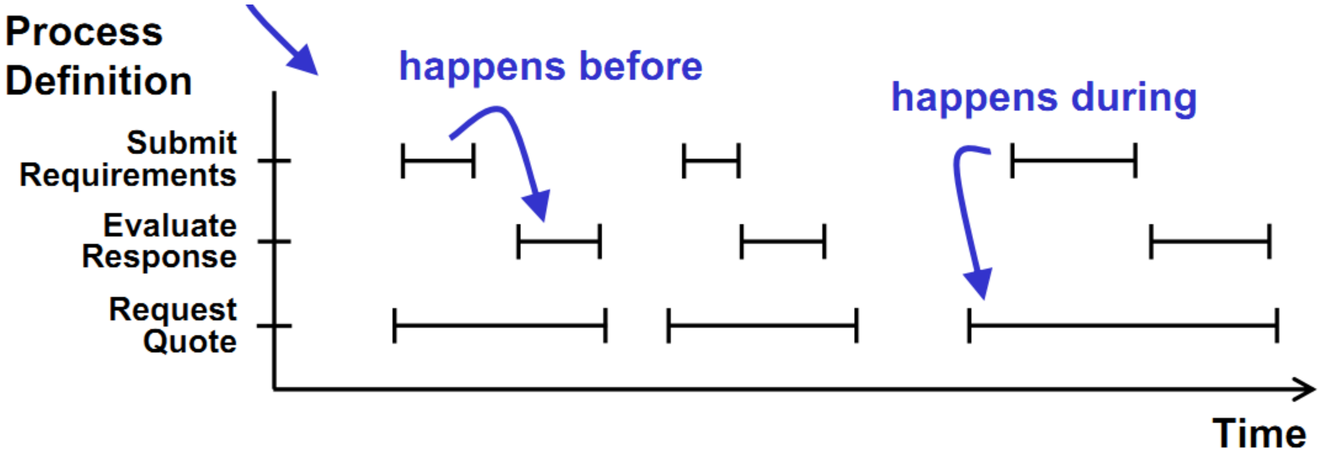
EXECUTION OF BUSINESS PROCESSES

The Semantics of Processes

- The modelling perspective

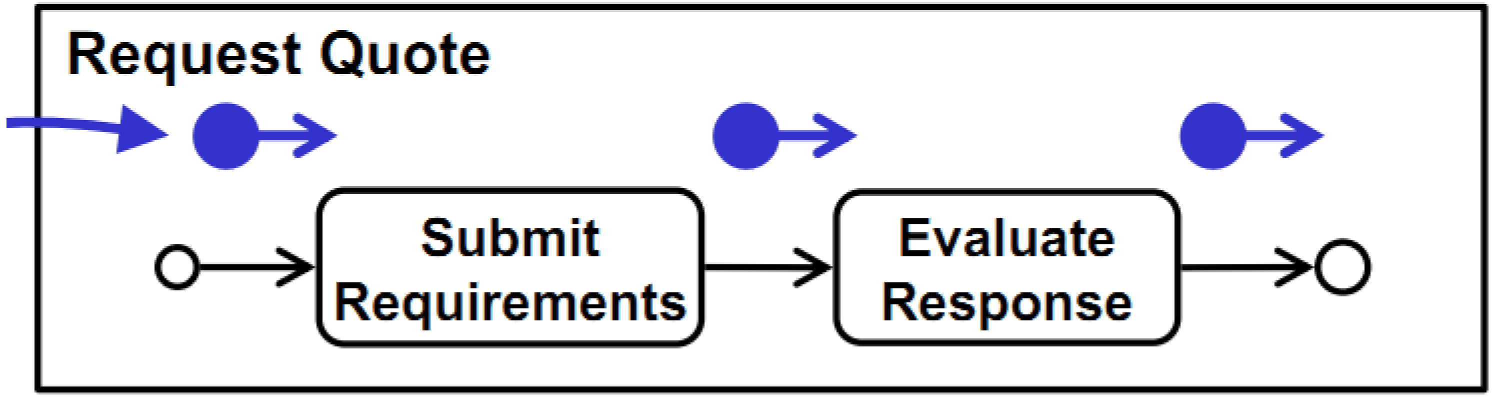


- The intended execution

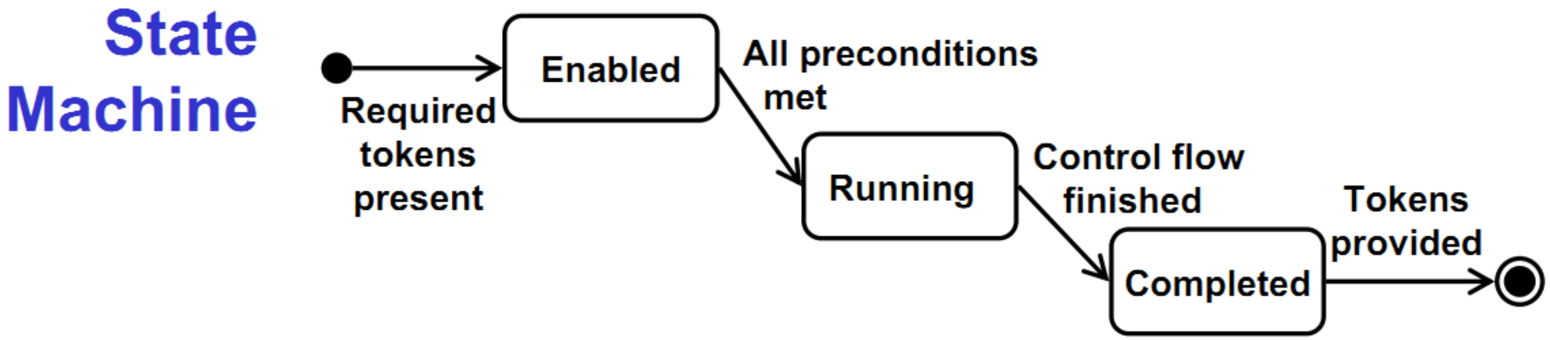


Process Execution

- Token flow

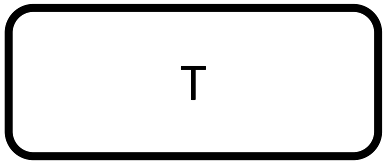


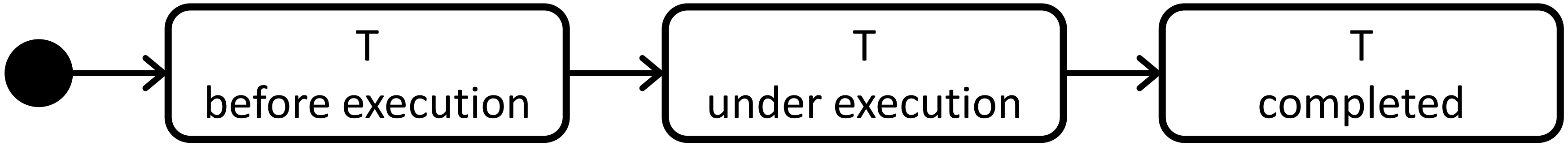
- The states of the process

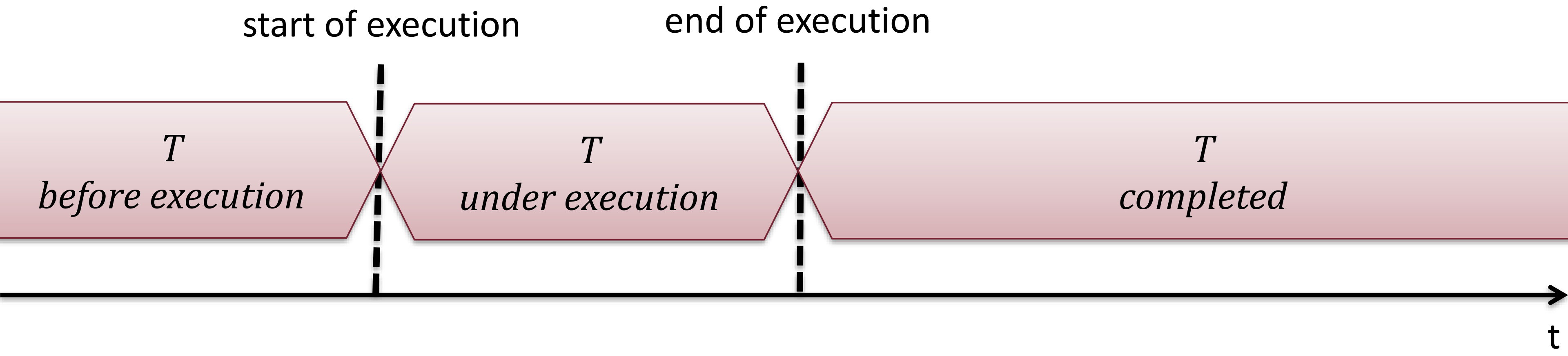


States of an Elementary Activity

Activity T:

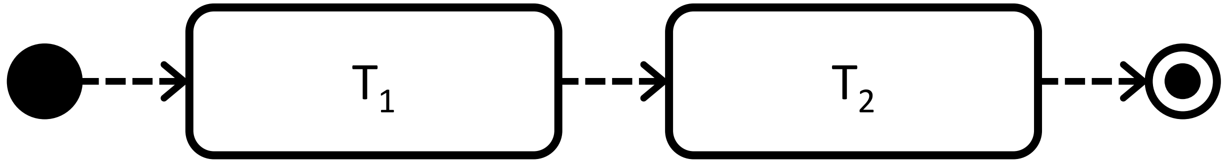


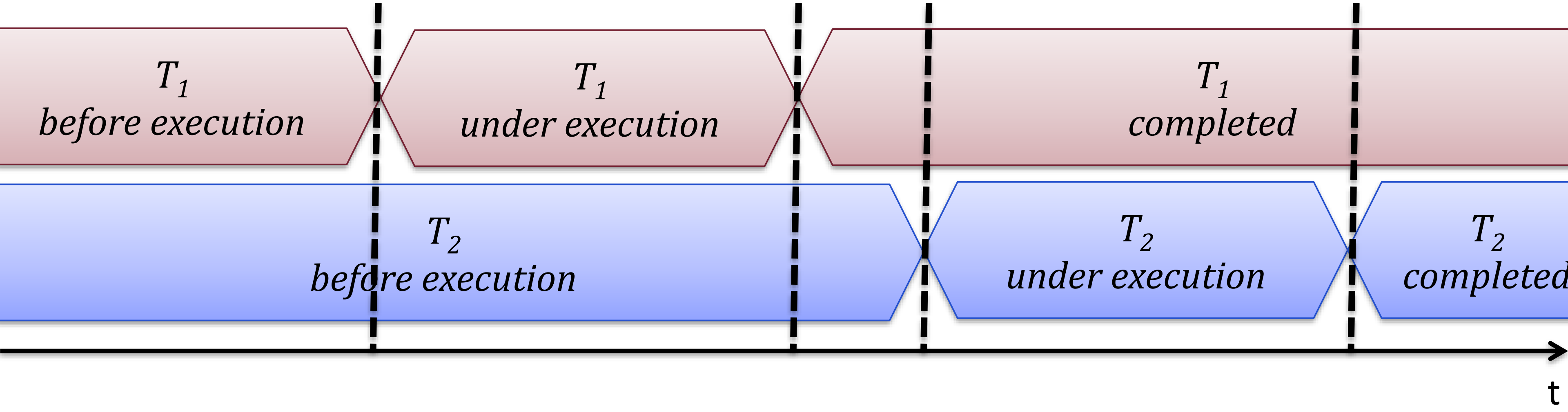




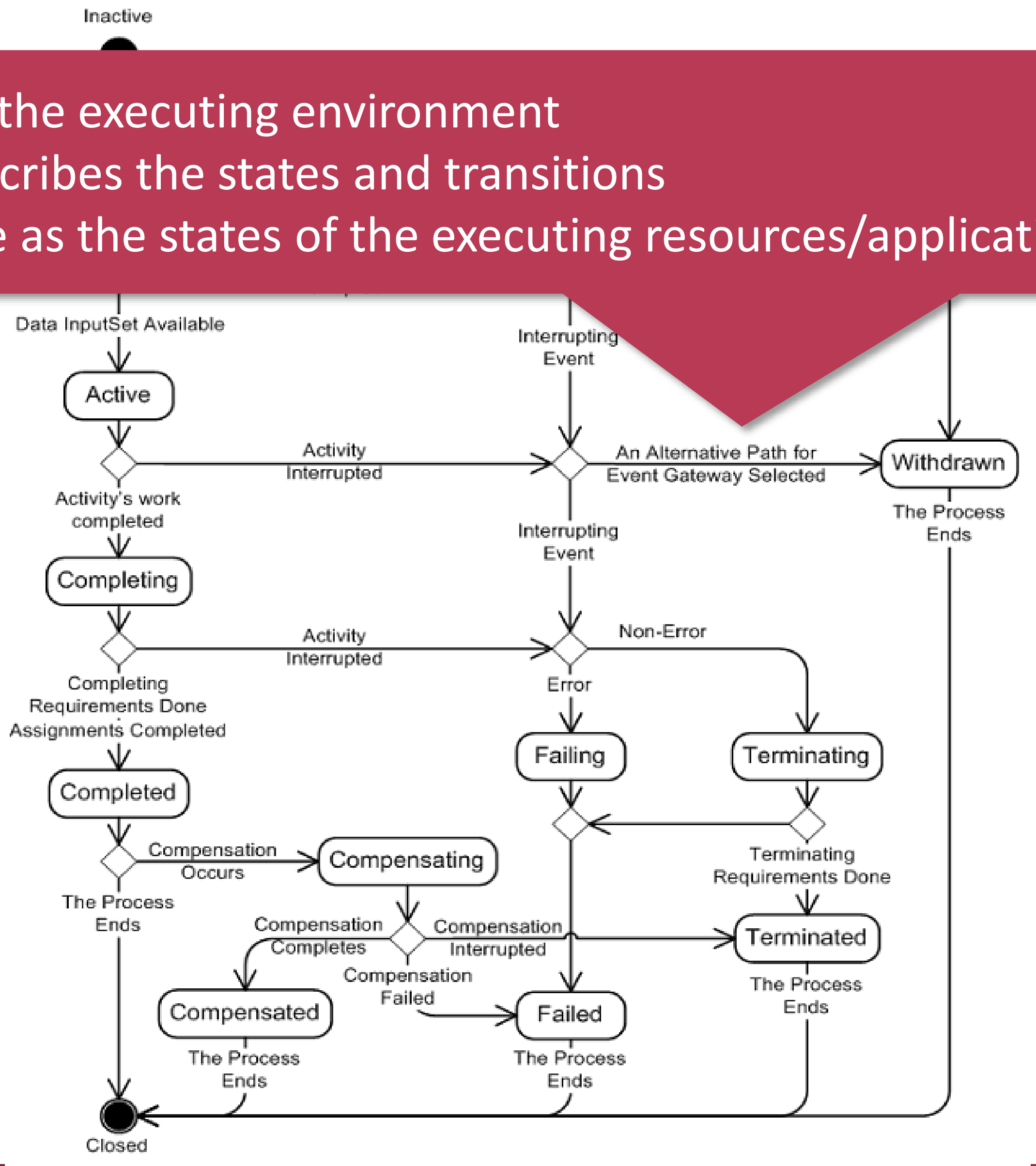
States of a Process

Process T= $T_1;T_2$





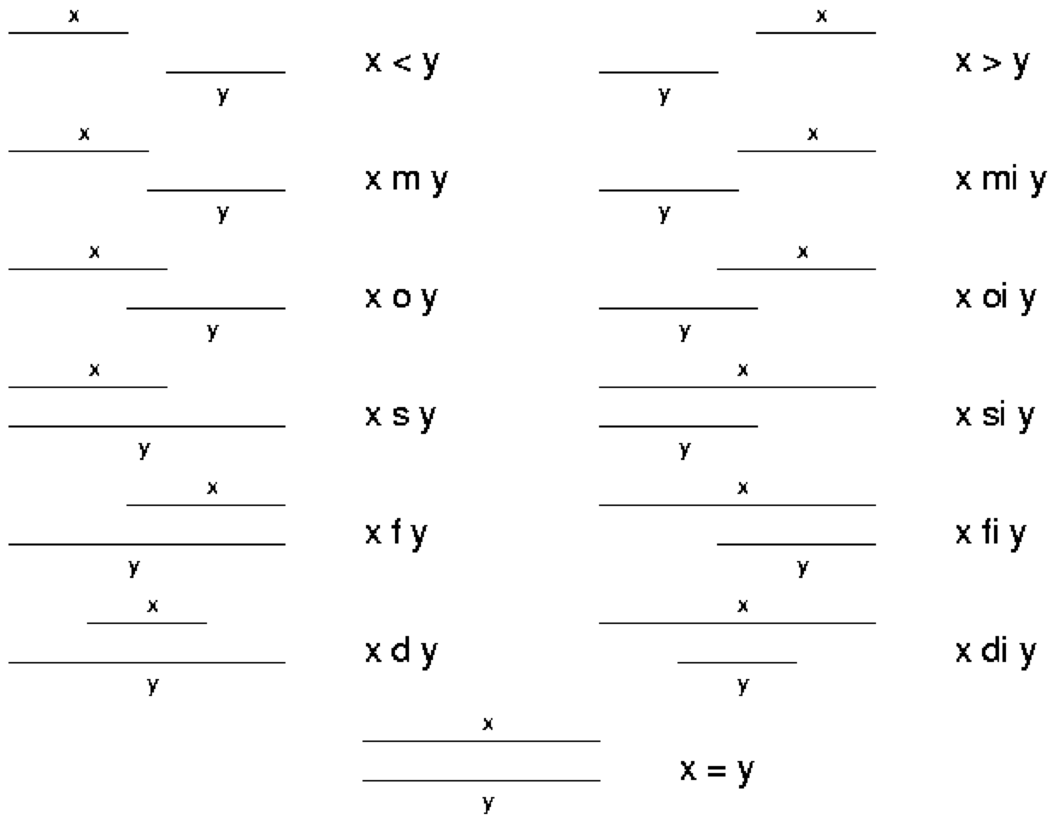
Symplified State Machine of an Activity



- Managed by the executing environment
- Standard describes the states and transitions
- Not the same as the states of the executing resources/applications

Background: Mathematical Model

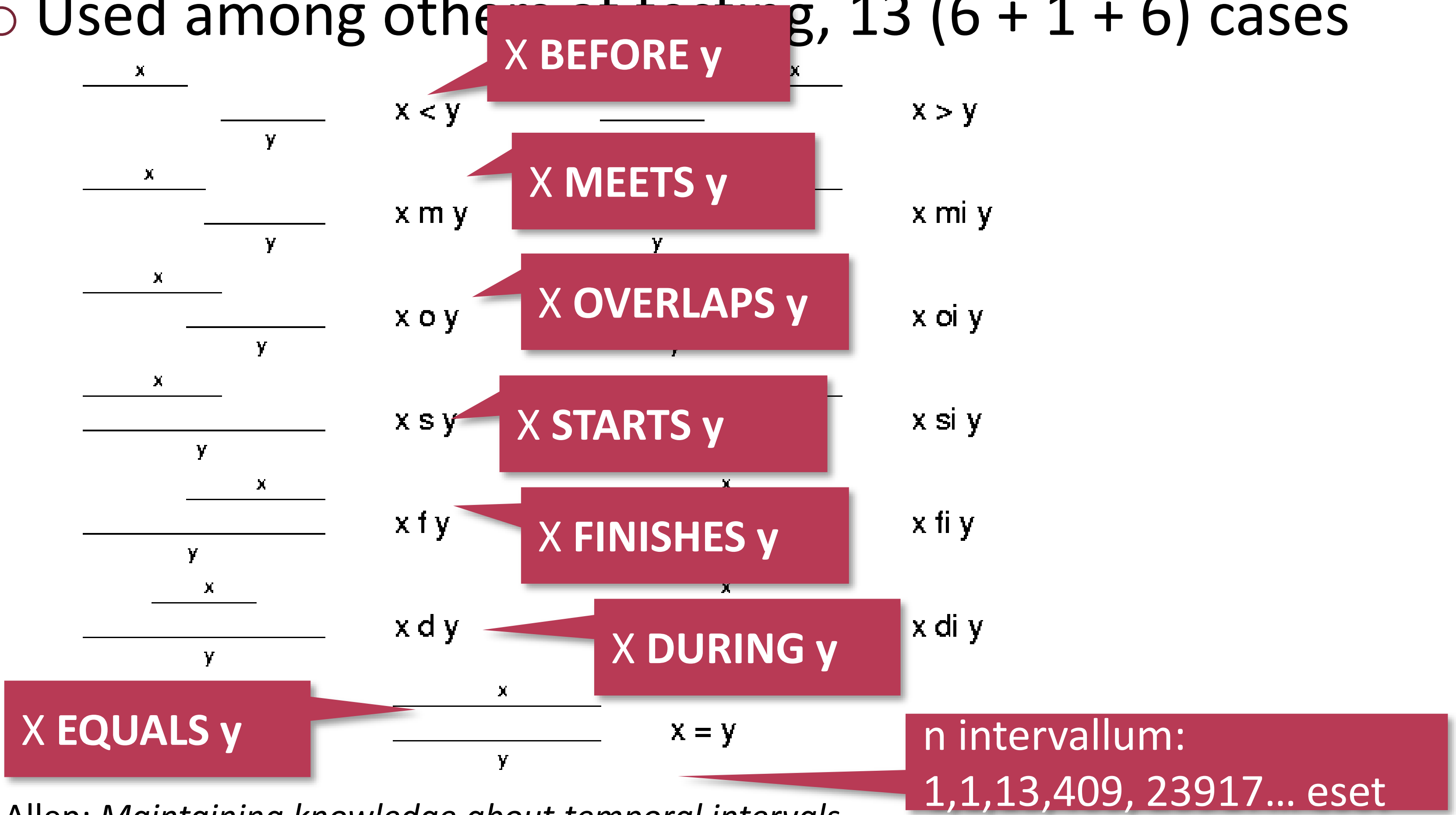
- Allen's interval algebra (1983)
  - Used among others at testing, 13 (6 + 1 + 6) cases



James F. Allen: *Maintaining knowledge about temporal intervals*.
In: *Communications of the ACM*. 26 November 1983. ACM Press. pp. 832–843, ISSN 0001-0782

Background: Mathematical Model

- Allen's interval algebra (1983)
  - Used among others at testing, 13 (6 + 1 + 6) cases



James F. Allen: *Maintaining knowledge about temporal intervals*.
In: *Communications of the ACM*. 26 November 1983. ACM Press. pp. 832–843, ISSN 0001-0782

What Can Be Checked?

- The execution is not based on the given process
  - Satisfaction of assumptions (order, independence)?
- What is the „process” behind system/execution?
  - Workflow mining
- If e.g. the execution environment is permissive
  - Steps can be skipped, ….
  - Are the requirements still satisfied?
- Tooling: formal methods
  - (Temporal)Logics, Petri nets, model checking, etc.