Process modeling

Vince Molnár

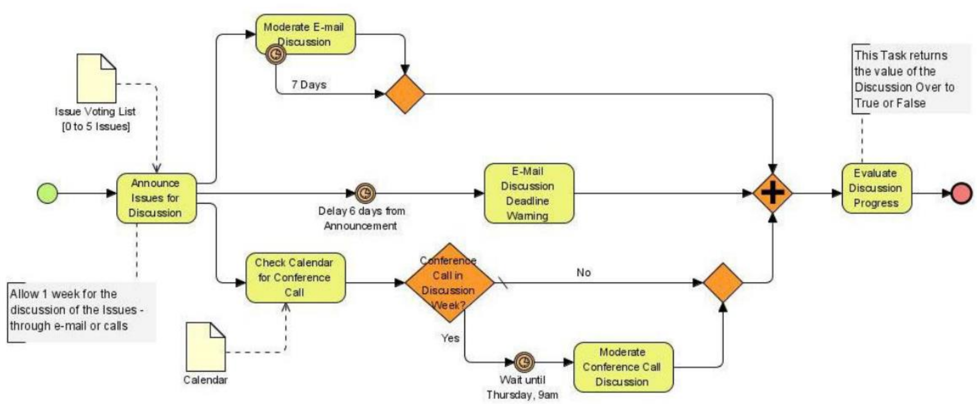
Informatikai Rendszertervezés

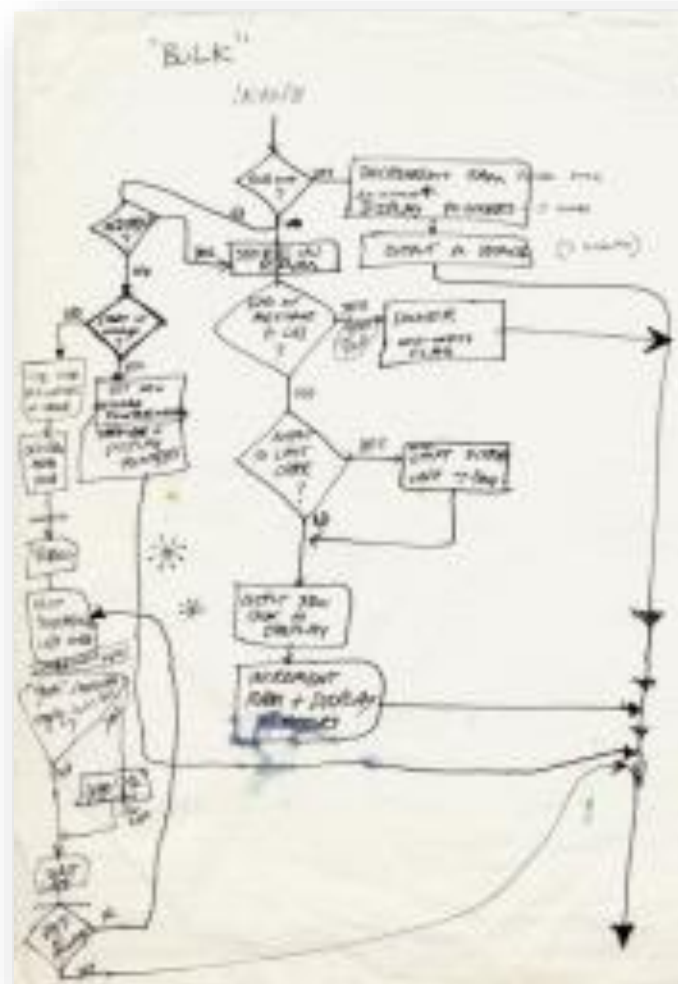
BMEVIMIAC01

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group

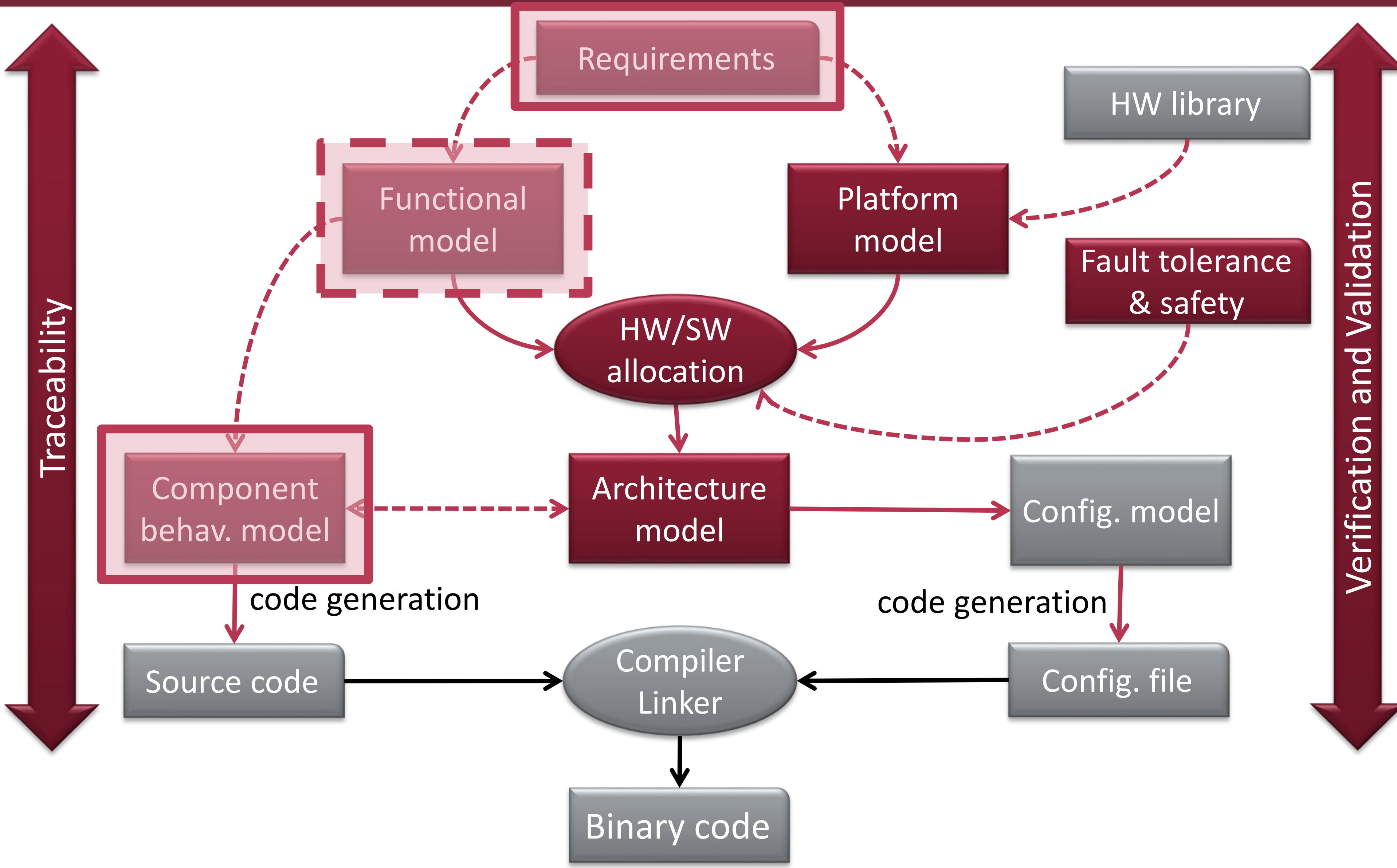
Roots & Relations

- Flow-sheets and flow-charts are used everywhere...
  - Brainstorming
  - Computer algorithms
  - Business processes





Platform-based systems design



Learning Objectives

Process modeling

- Understand the basic blocks control and data flow modeling
- Identify the steps of, the data being used by and the logical flow of a process
- Understand the syntactic building blocks of UML Activity Diagrams
- Understand the semantics of UML Activity Diagrams
- Use hierarchy to structure the models and express abstraction-refinement of actions
- Build clean and expressive models by using best practices
- Be able to use Activity Diagrams in high-level process modeling and low-level behavior modeling

PROCESS MODELING

Objectives

Main aspects

Objectives

- Transformation of **inputs to outputs** through a sequence of actions
- Model **control flow** and **data flow**
- Definition of **high-level** processes
  - Elaborate use cases
  - Functional decomposition
- Definition of **low-level** activities
  - Specific behavior executed at given points
    - E.g. reaction to an event

Main aspects

- **Atomic activities (Actions)**
  - An activity that is not detailed further
  - Depens on the level of abstraction
    - Use case
    - Informal description of some activity
    - Primitive operation (e.g. object access/update, messaging)
  - May be refined later (see Activity Decomposition)
- **Control flow**
  - Specifies the order in which activities can be executed
  - Also: **cuncurrency** and **exceptions**

Main aspects

- **Data flow**
  - Specifies the flow of data between activities
    - Where can a certain data element propagate?
  - Facilitates data flow analysis
    - …to reveal opportunities for optimization
    - …to avoid errors caused by improper data usage
- **Activity allocation**
  - Which functional block will execute an activity?
- **Activity decomposition**
  - Refine and/or reuse activities

UML ACTIVITY DIAGRAM

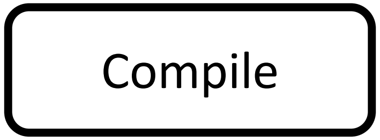
Control flow

Activity refinement

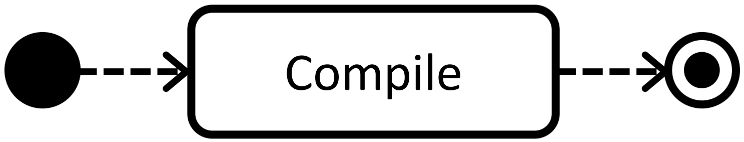
Data flow

Allocation

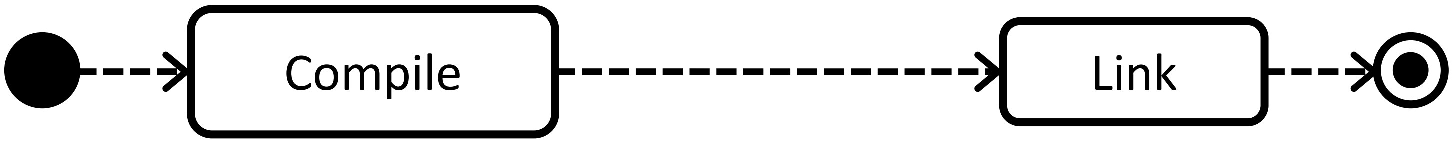
Basic control flow – Atomic Activity



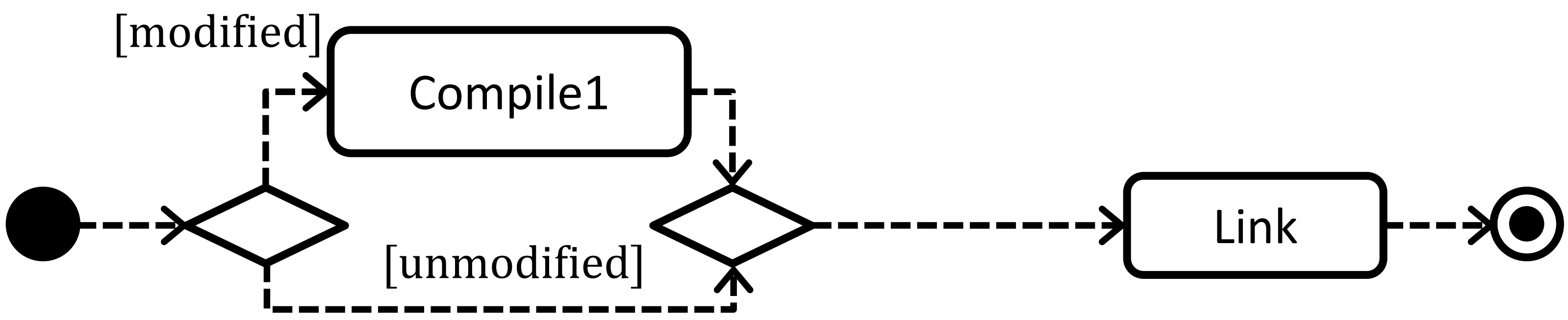
Basic control flow – Initial & Final node



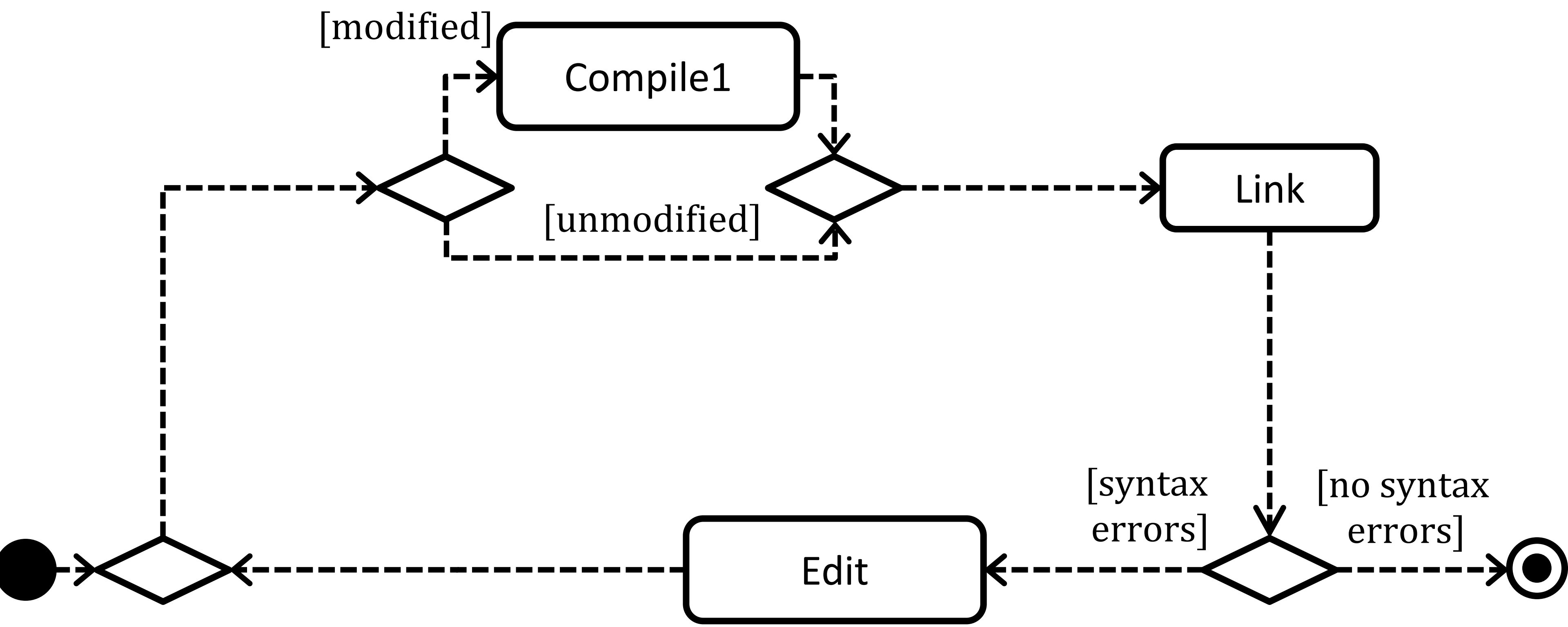
Basic control flow – Sequence



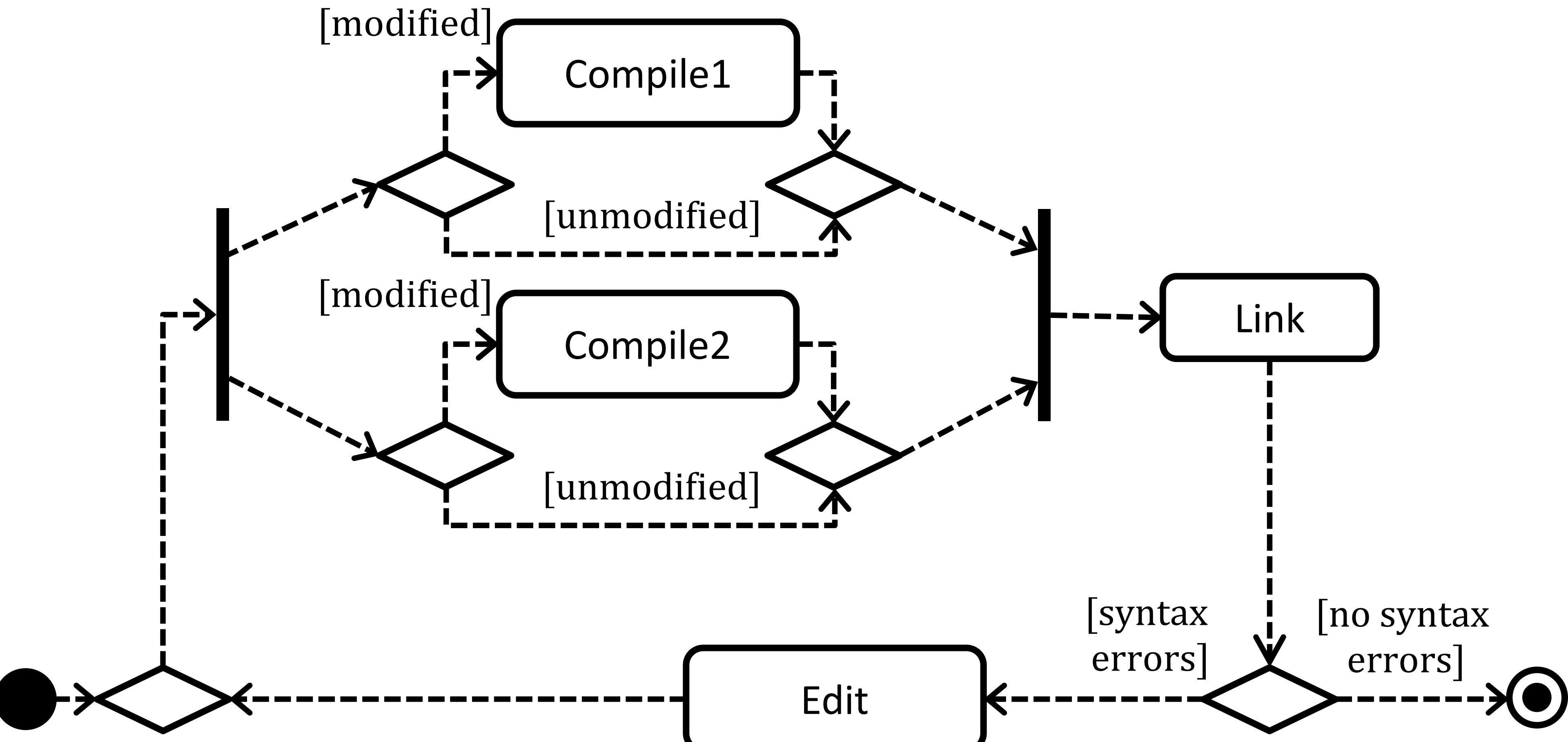
Basic control flow – Decision & Merge



Basic control flow – Loop

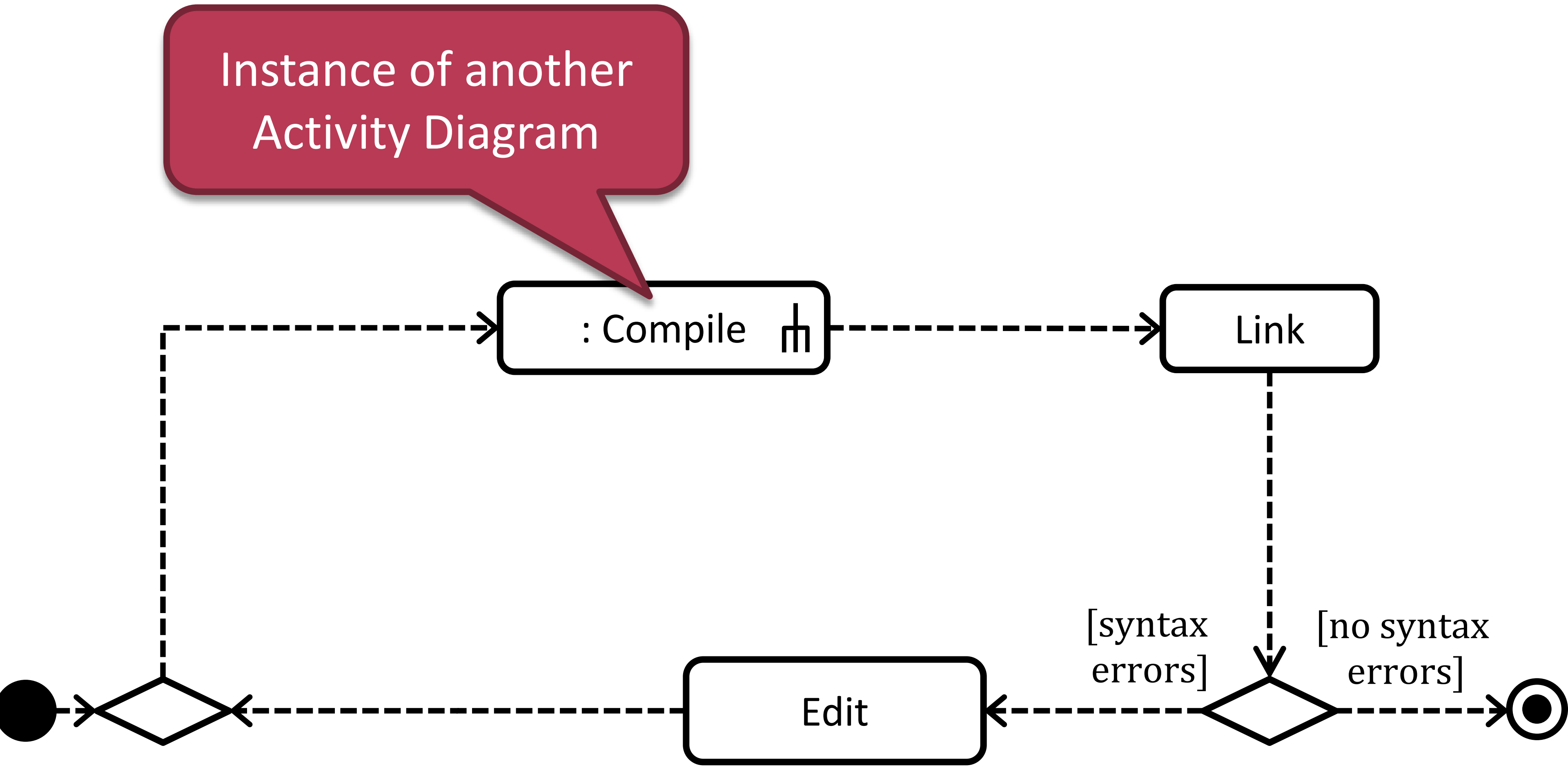


Basic control flow – Fork & Join

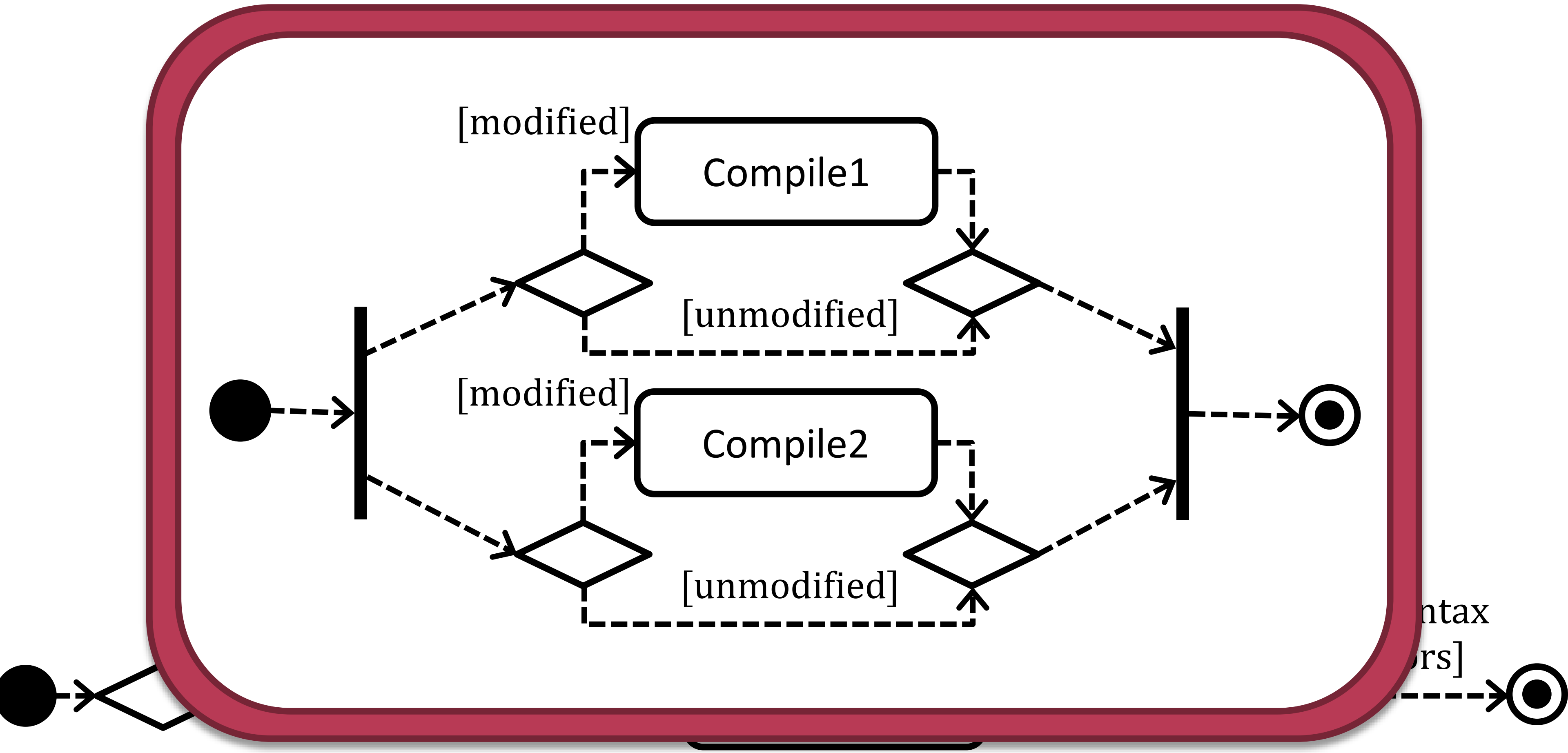


MŰEGYETEM 1782

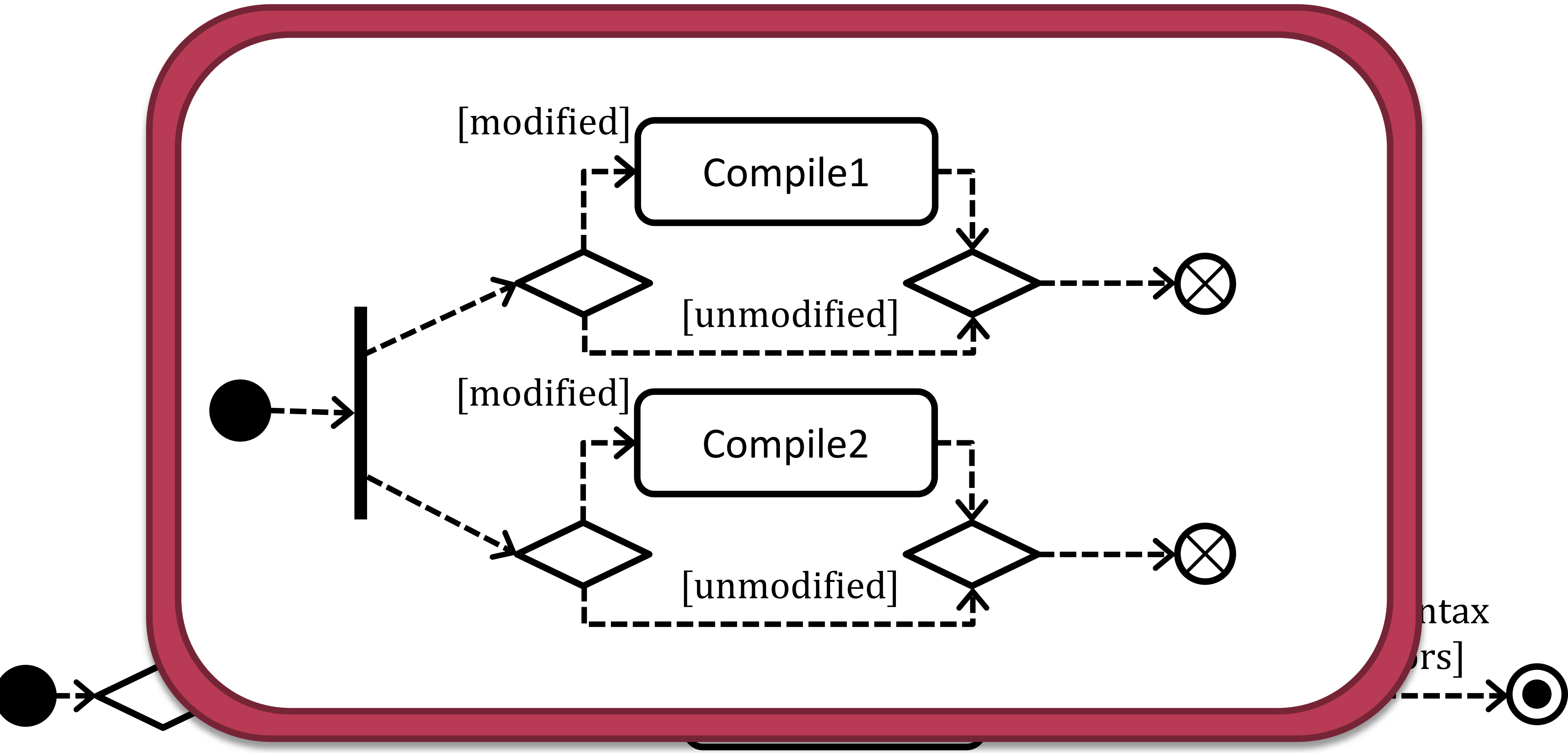
Activity refinement



Activity refinement

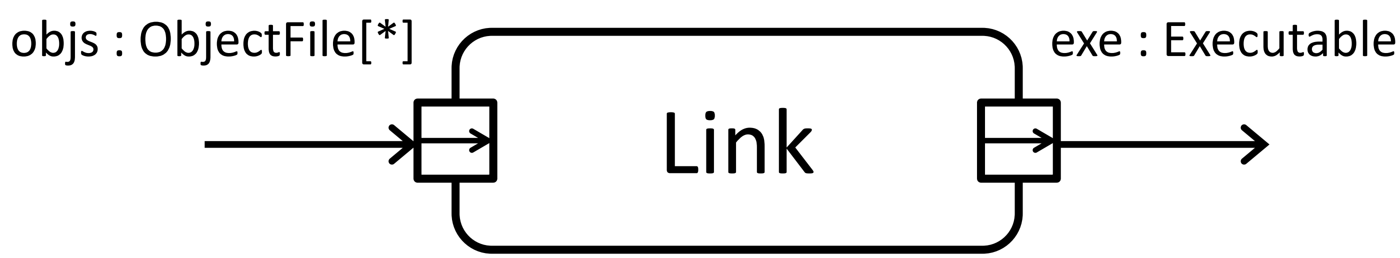


Basic control flow – Flow end



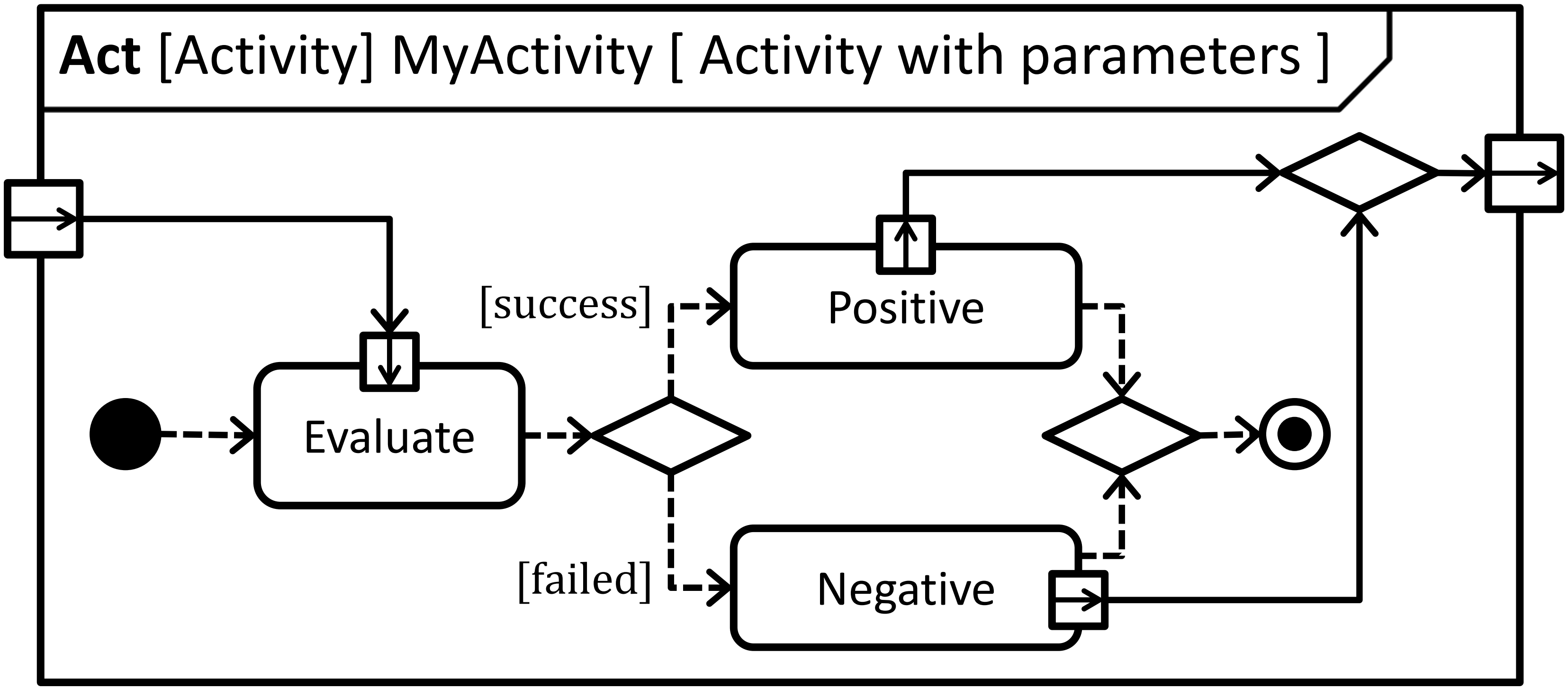
Modeling data flow

- Activities usually **consume and produce data**
  - Data can mean physical atrifacts
  - The produced data can be an input for another activity
- Notation: **Input/Output pins**
  - Can have name and type
  - Data flow is denoted by **solid arrows**



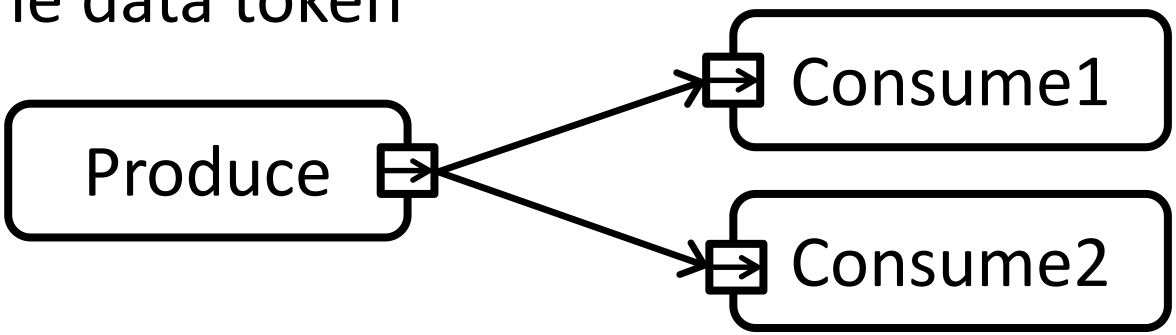
Modeling data flow

- An Activity can have **parameters**
  - **Parameter pins:** similar to Input/Output pins
  - Appear on the frame of an Activity Diagram

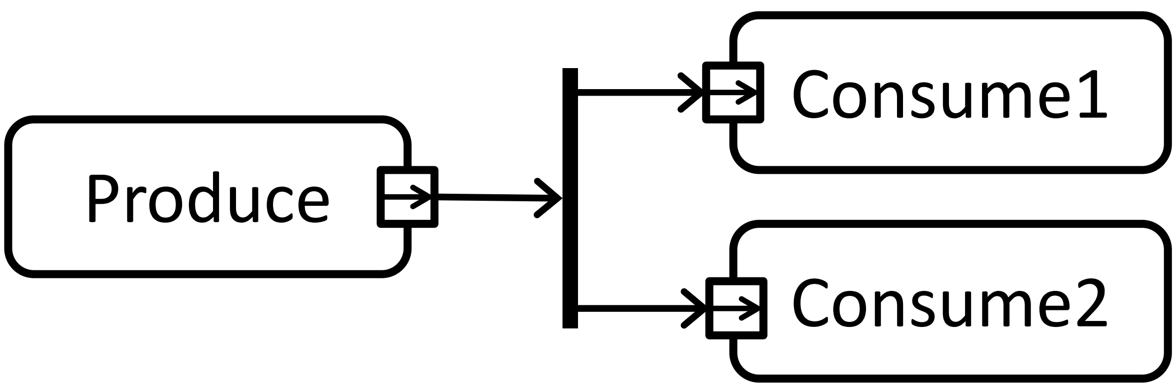


Exclusive/Shared data

- Output pins „emit" a **single data token**
  - Input pins connected to the same output pin **compete** for the data token



- Use a **fork** to produce multiple data tokens



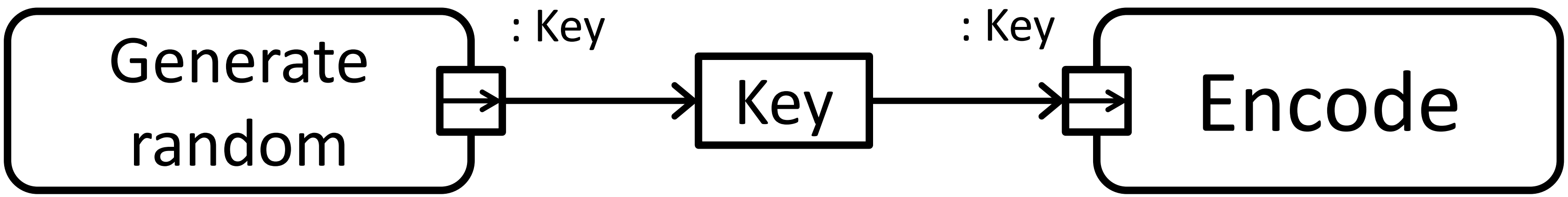
Control flow vs. Data flow

- **Data flow** denotes data dependencies
  - Some step **requires data** from another
- **Control flow** denotes control dependencies
  - Some step can be **executed only after** another

- Data flow can **substitute** control flow
  - Modeling control flow is not mandatory if there is a data flow between two actions
    - Still, it is sometimes useful to have them separately
  - Control flow can be regarded as a “void” data flow

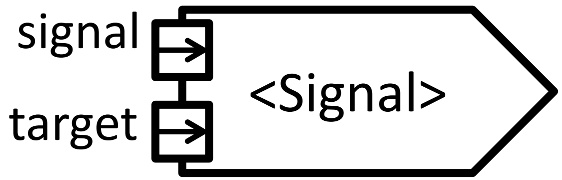
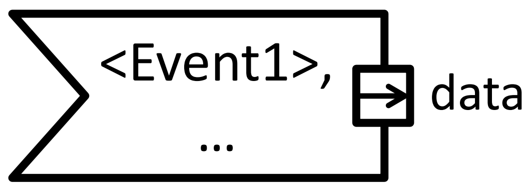
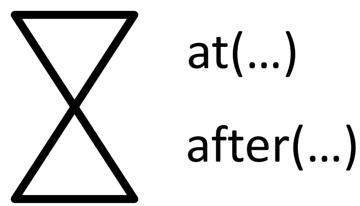
Object node

- Use an **object node** to emphasize the flowing data



- Built-in object nodes in SysML:
  - **Central buffer:**
    - Can model a message queue or pool
    - Same behavior as an output pin, but not related to an action
  - **Datastore:**
    - Denotes a permanent storage
    - Data tokens are stored and retreived

Atomic activities (Actions)

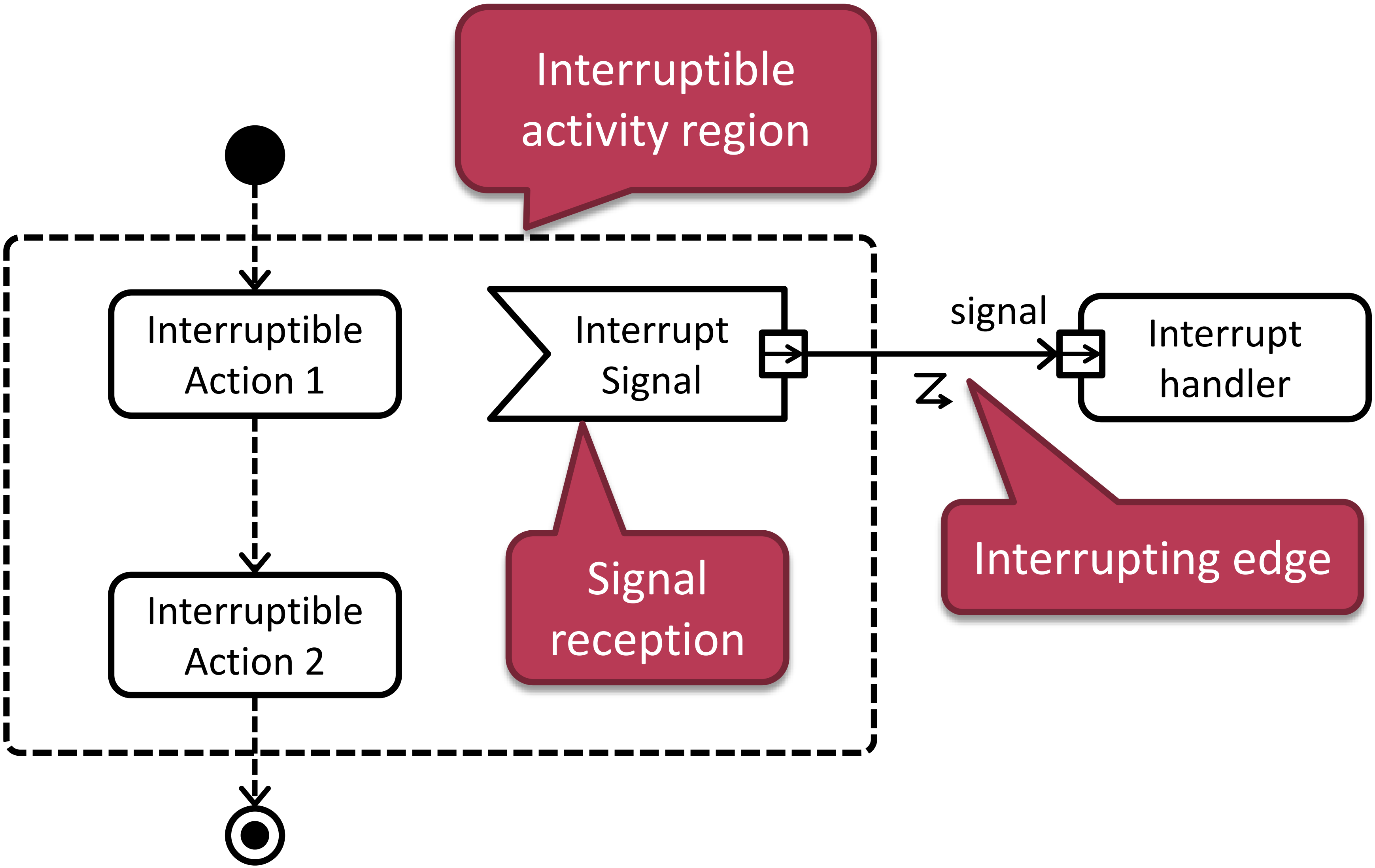
Primitive action		E.g. object access, update and manipulation actions.
Send signal		Send a signal to the specified target.
Accept event		Accepts incoming events. Typically outputs received data.
Accept time event		Raised by an expiration of an (implicit) timer.
Call behavior		Executes another behavior (e.g. another Activity).

Interruptible activity region

Interruptible activity region

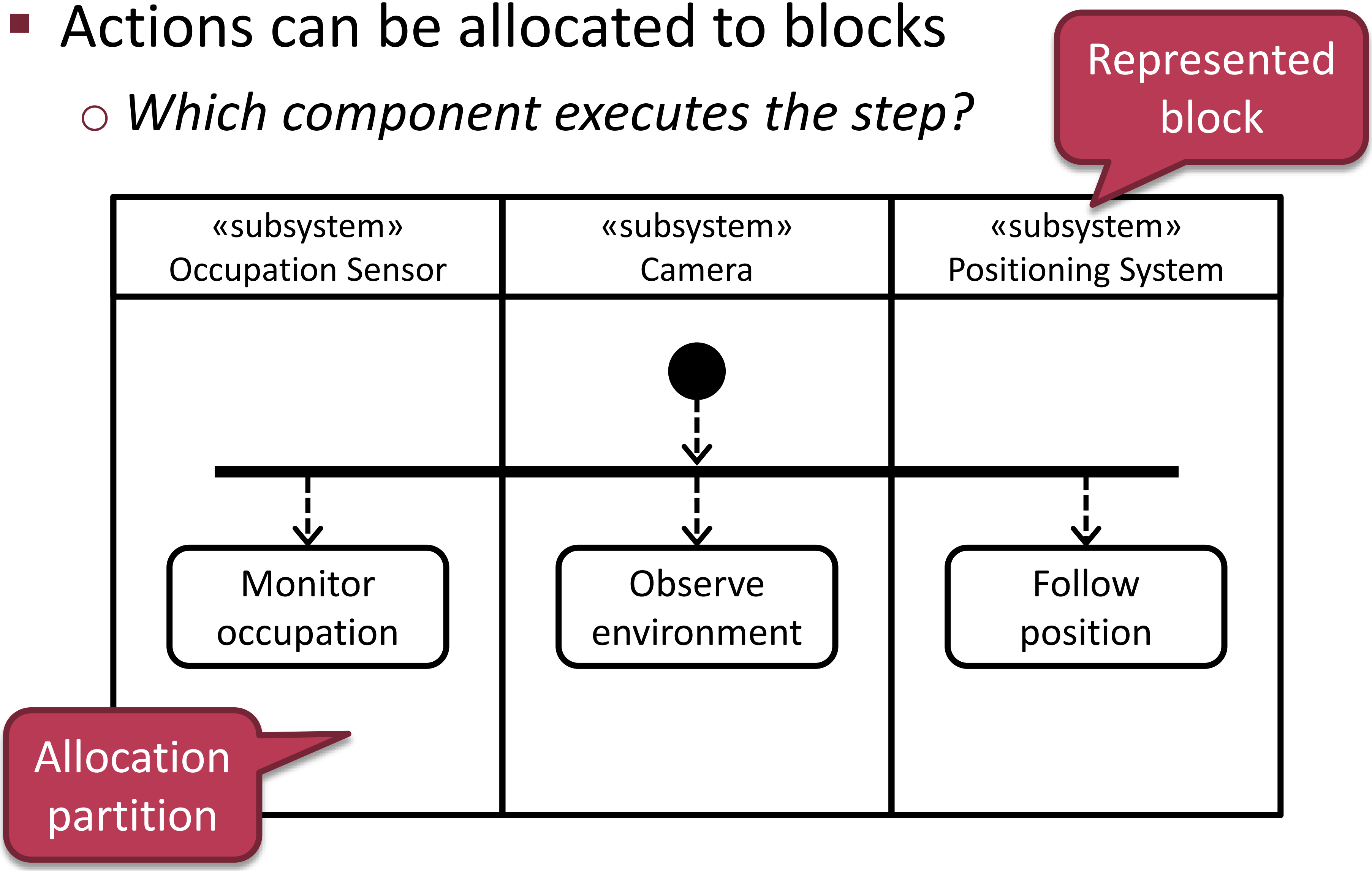
- Specifies a part of the activity that will be interrupted if a certain event occurs
  - Control is transferred to an **exception handler**
    - + Some data regarding the event
  - Similar to a try-catch block
- Interrupt…?
  - **Not** in the sense of HW interrupts
    - See State Machines
  - Rather like **SIGINT**, execution of the activity stops

Interruptible activity region



Allocation of Actions

- Actions can be allocated to blocks
  - *Which component executes the step?*



Summary

- **Atomic activities (Actions)**
  - Primitive actions
  - Send signal
  - Accept (time) event
  - Call behavior
- **Control flow**
  - Initial/Final node, Flow final
  - Decision & Merge
  - Fork & Join
  - Interruptible activity region

Summary

- **Data flow**
  - Input/Output pins
  - Parameter pins
  - Object nodes
- **Activity allocation**
  - Allocation partition
- **Activity decomposition**
  - Call behavior actions
  - Parameter pins

SEMANTICS

Tokens & Channels

Actions

Control structures

Tokens and Channels

- Represent the "*right to execute*" and data elements as **tokens**
  - **Data tokens** have type and value
  - **Control tokens** are typeless (like *void*)
- A channel is a **buffer** where tokens can be put to and read from
  - Usually FIFO (can be modified by stereotypes)
- What counts as a channel?

  - Output pin/Object node → Input pin/Object node
  - The buffer is "in" the starting point

Actions

- To execute (*fire*) an action, it needs
  - A **data token** on all of its input pins
    - **Type conformance:** argument type < parameter type
  - A **control token** from all incoming control flow connectors
  - \+ An incoming event in case of **“accept” actions**
  - Actions connected to the same output **compete** for the data/control token
- An executed (*fired*) action produces
  - A **control token** on all outgoing control flow connectors
  - A **data token** on all output pins

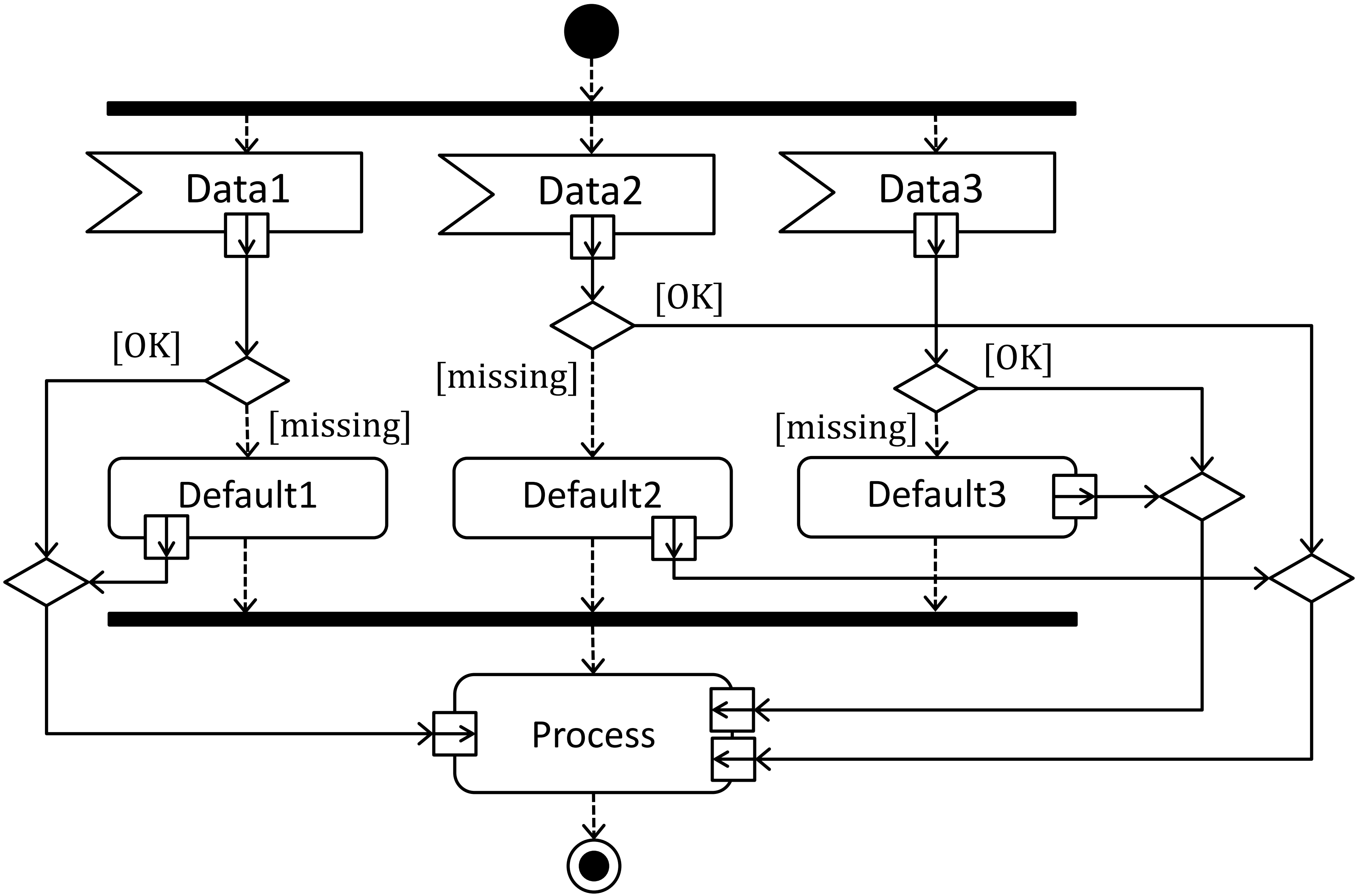
Control structures

- **Initial node:**
  - **Produces** a control token when the Activity is invoked
- **Final node:**
  - **Removes all control tokens** and returns from the Activity
- **Flow final:**
  - **Consumes** a control token
- **Decision:**
  - Forwards incoming token to **selected output**
- **Merge:**
  - Forwards incoming token from **any input** to output

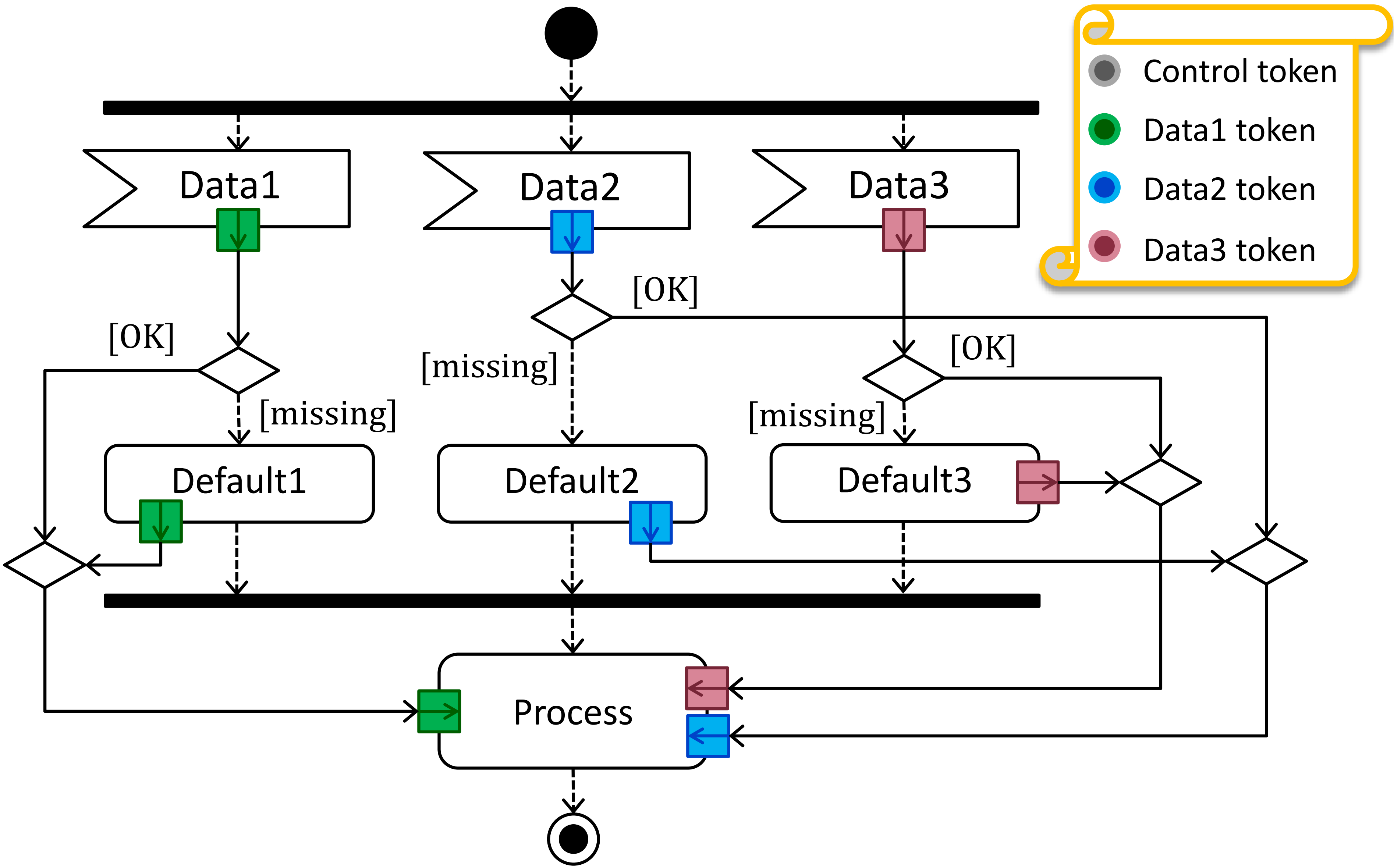
Control structures

- **Fork:**
  - Copies and forwards incoming token to **all outputs**
- **Join:**
  - **Waits for a token on all inputs** then forwards one
- **Interruptible activity region:**
  - **Removes all control tokens** from region when interrupted

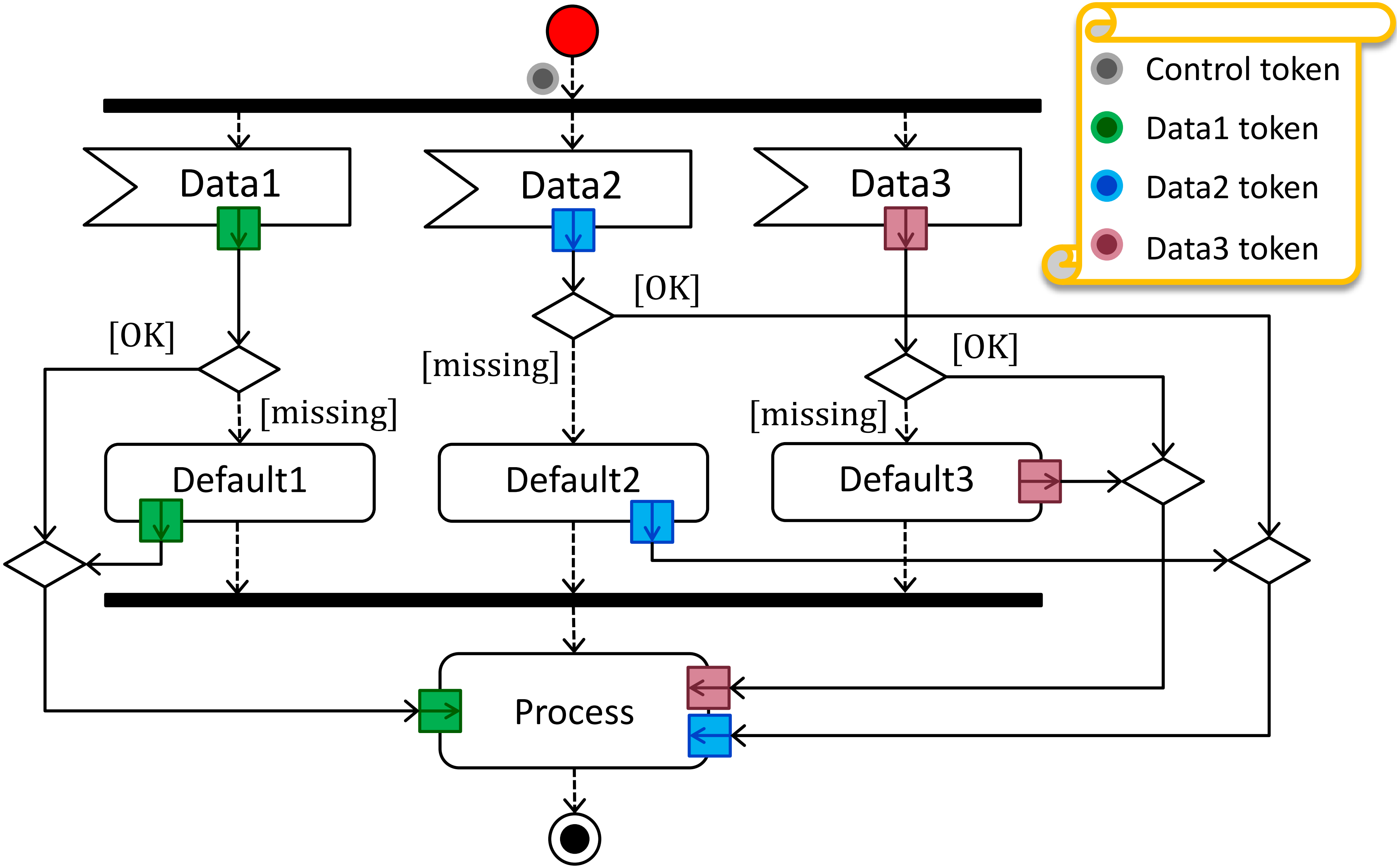
Example: Process collected data



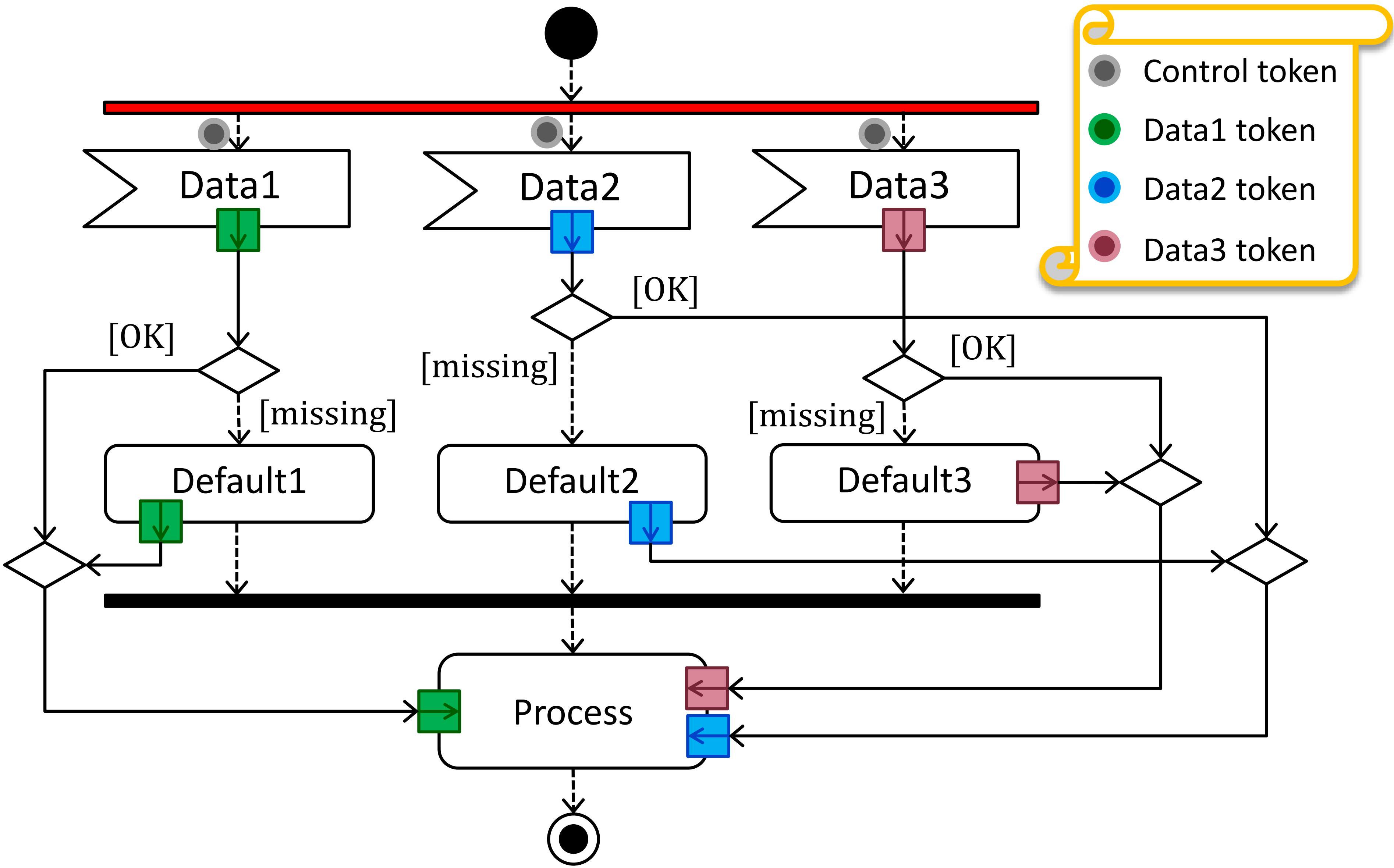
Example: Process collected data



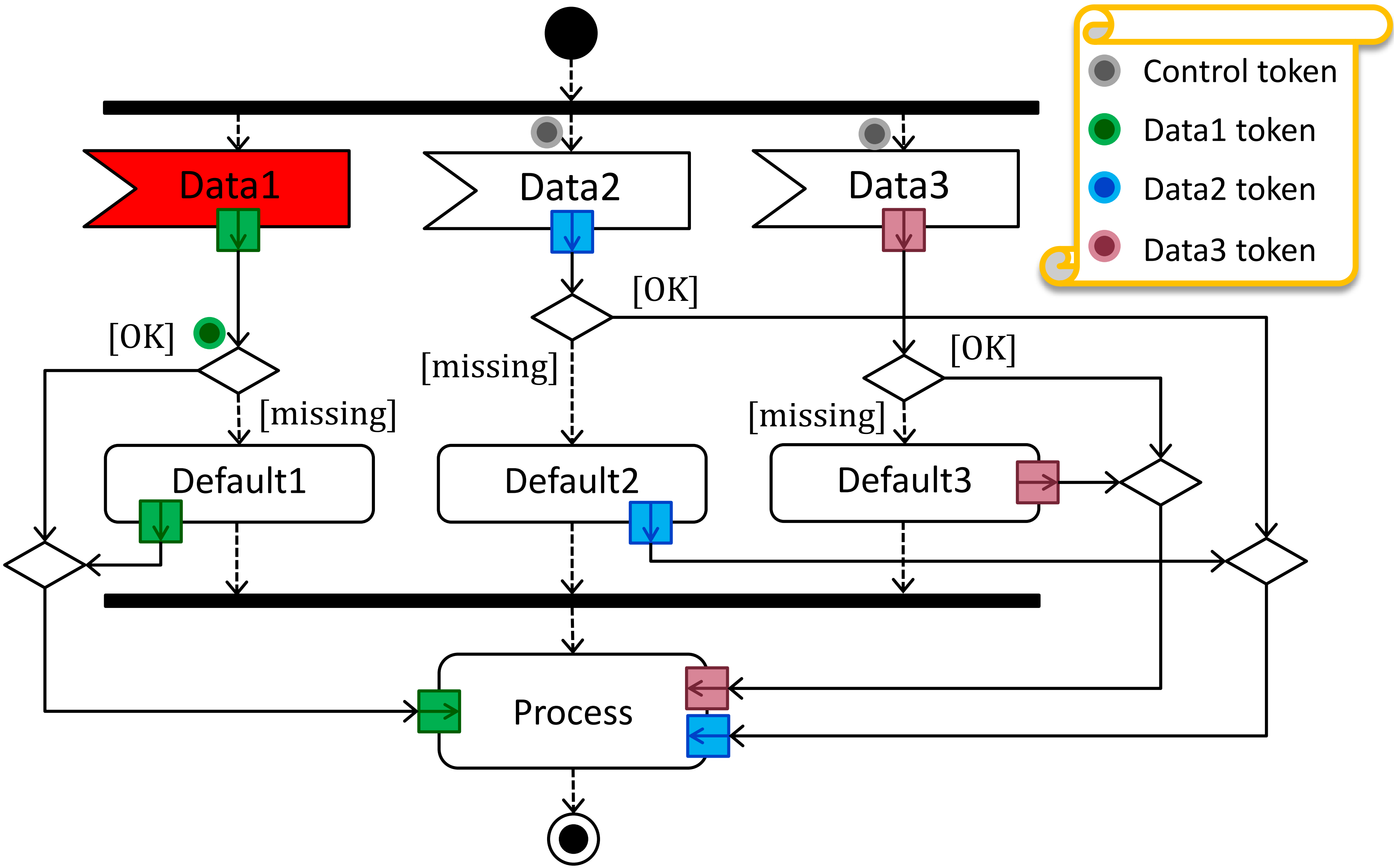
Example: Process collected data



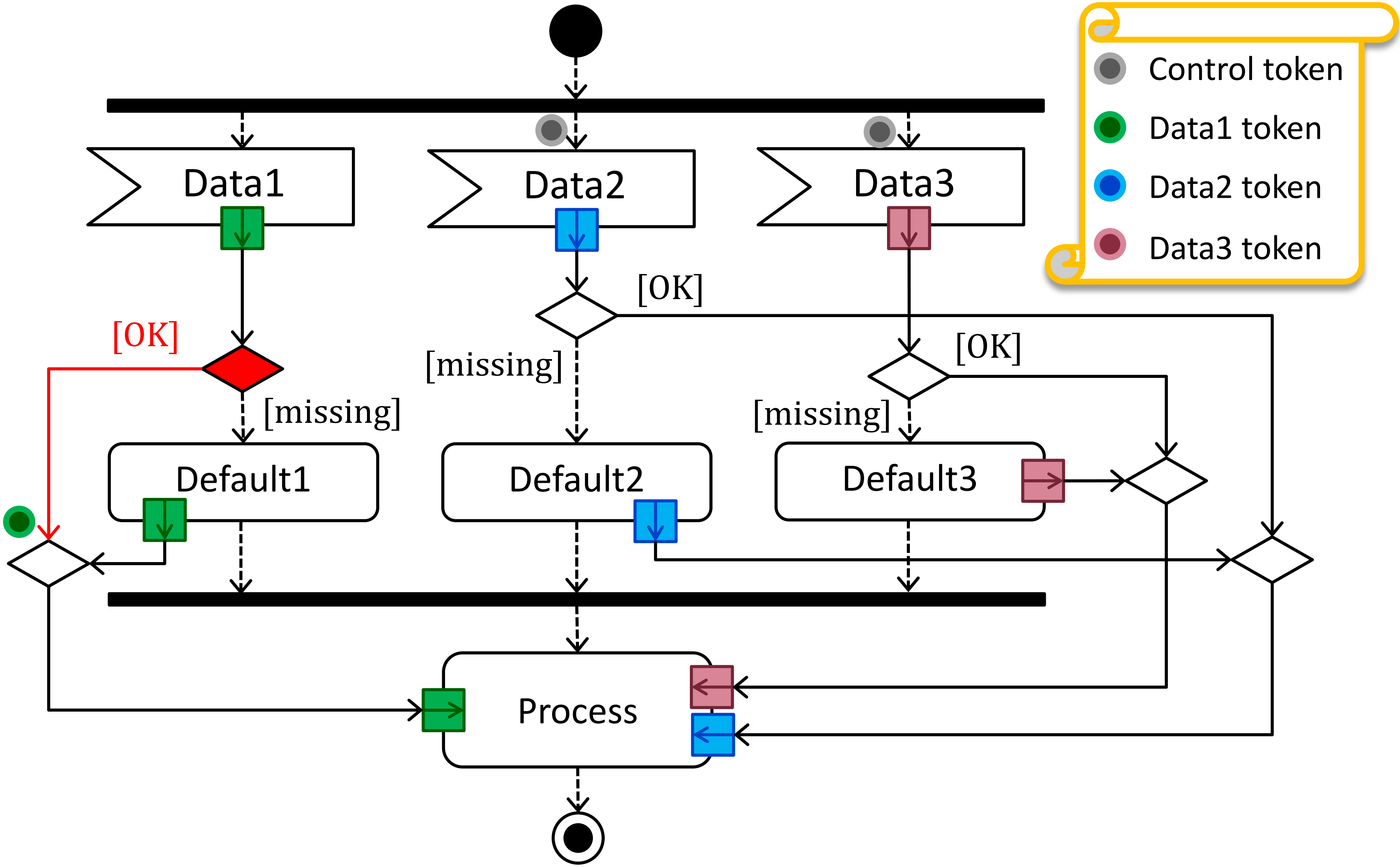
Example: Process collected data



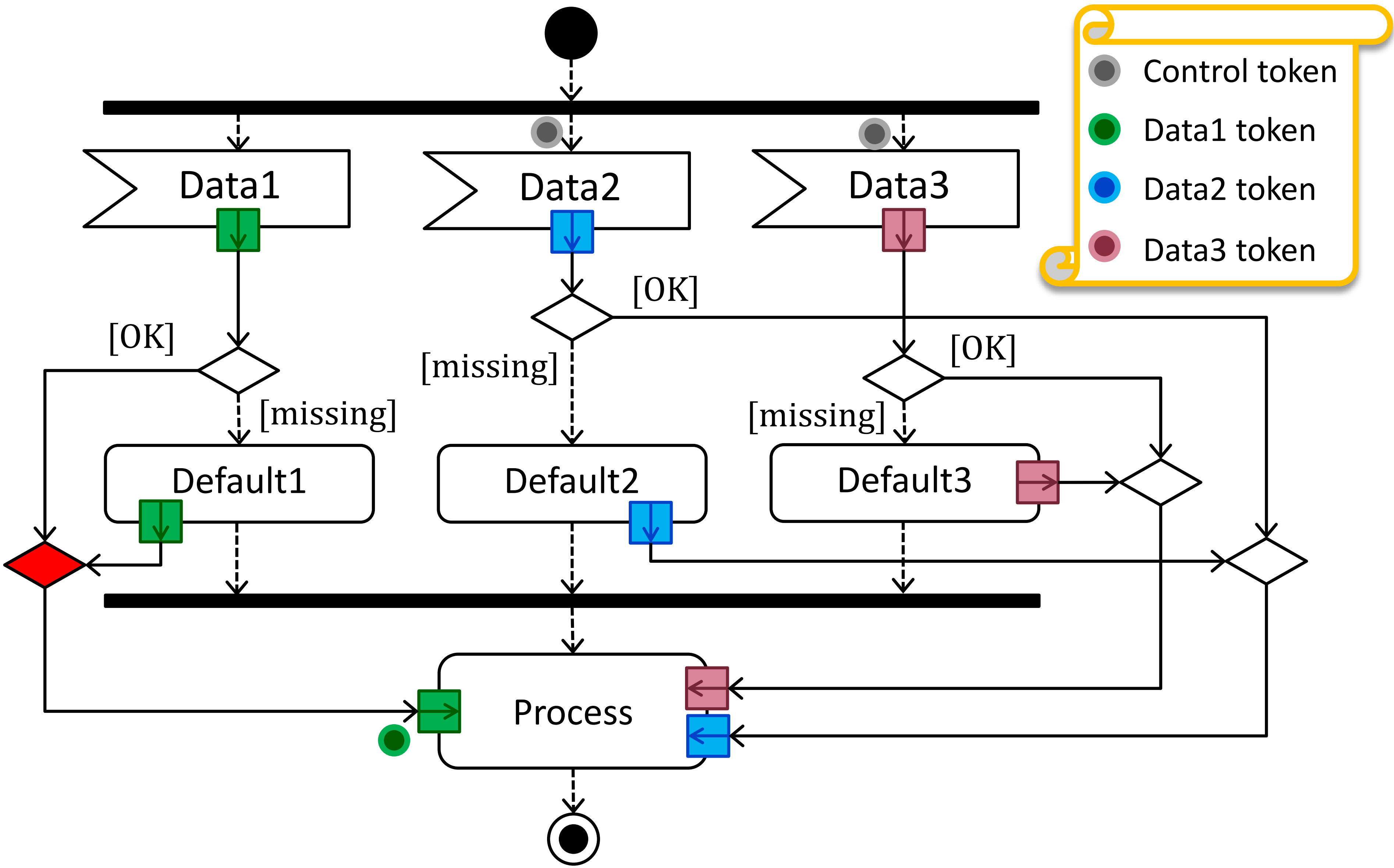
Example: Process collected data



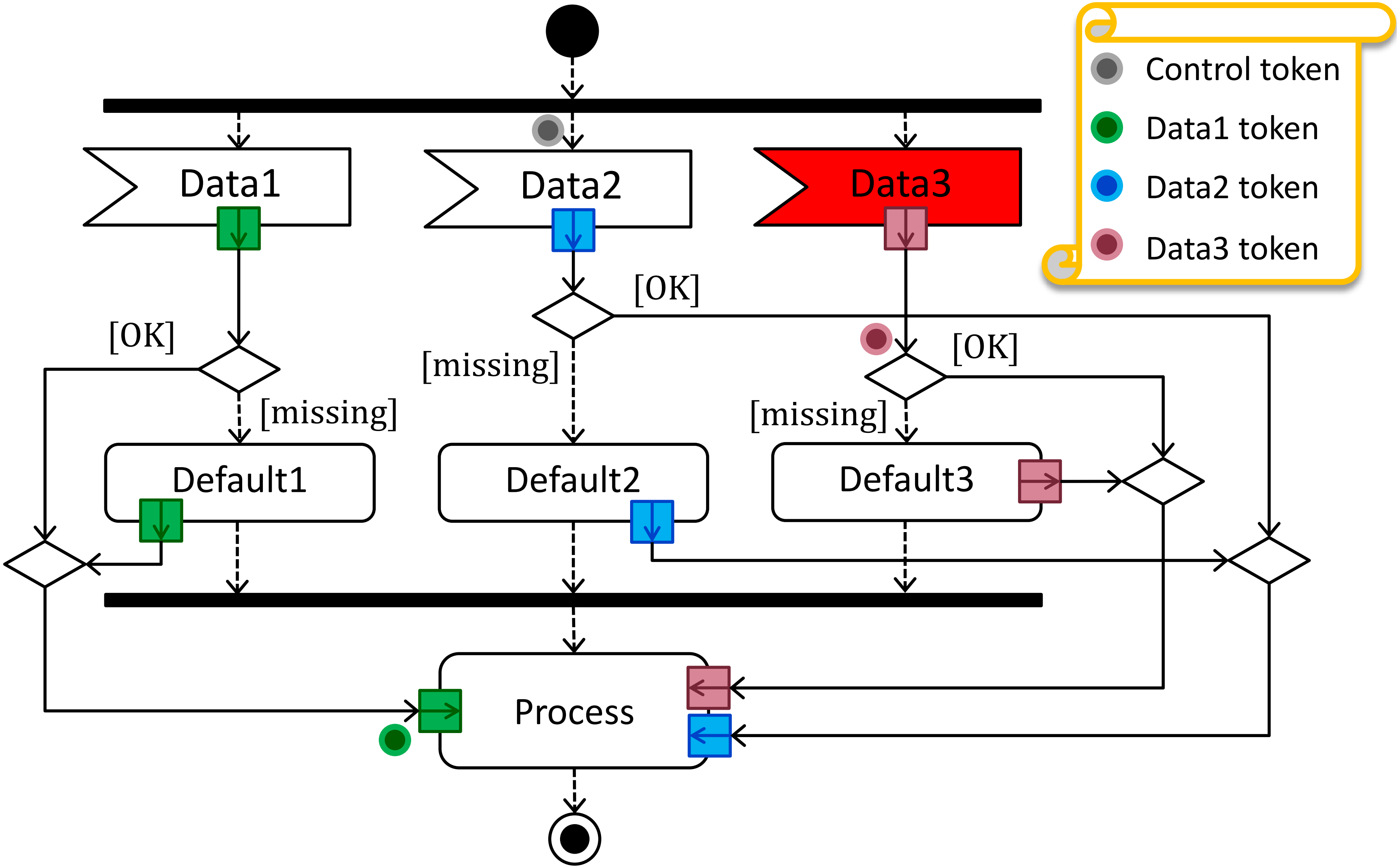
Example: Process collected data



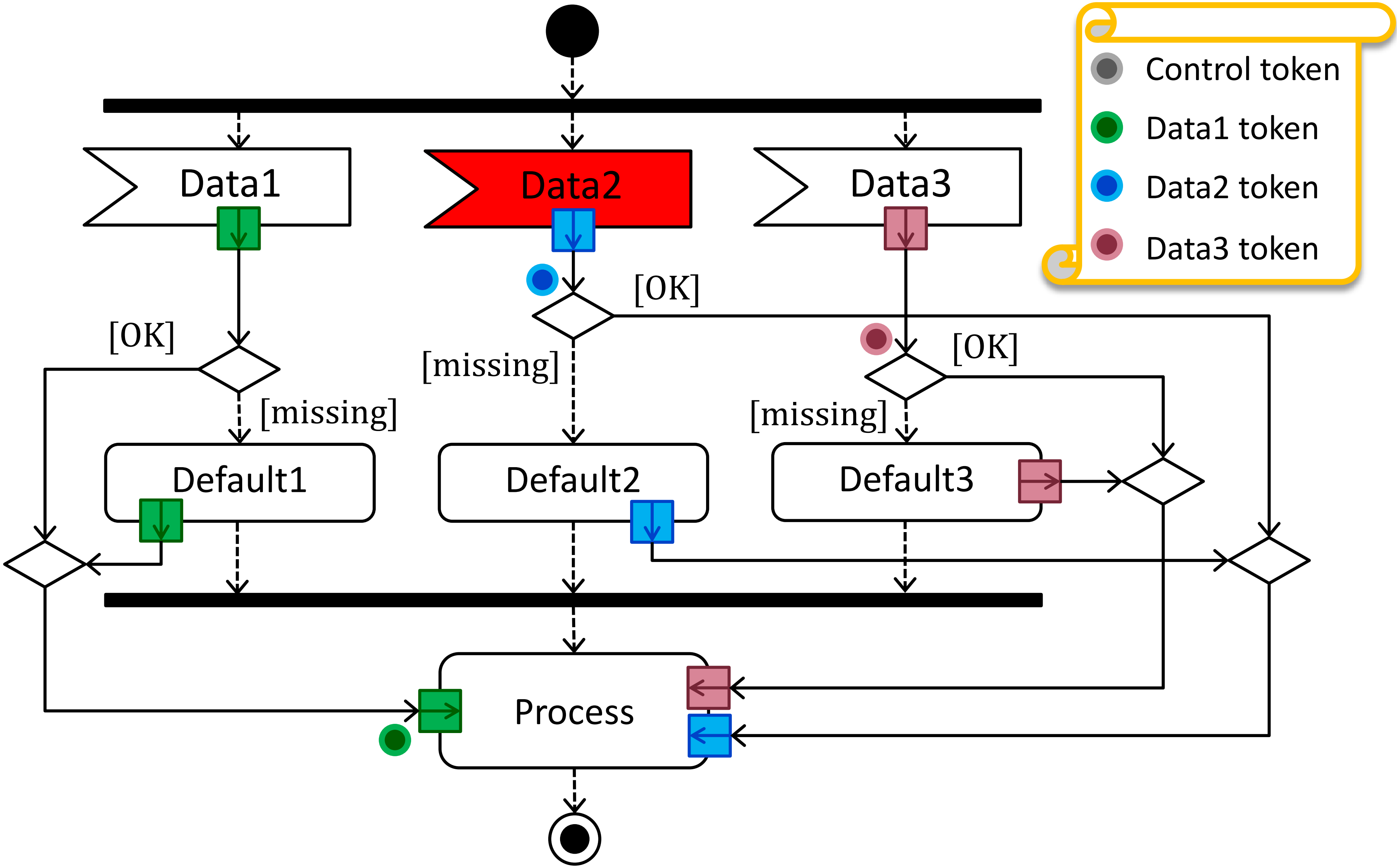
Example: Process collected data



Example: Process collected data

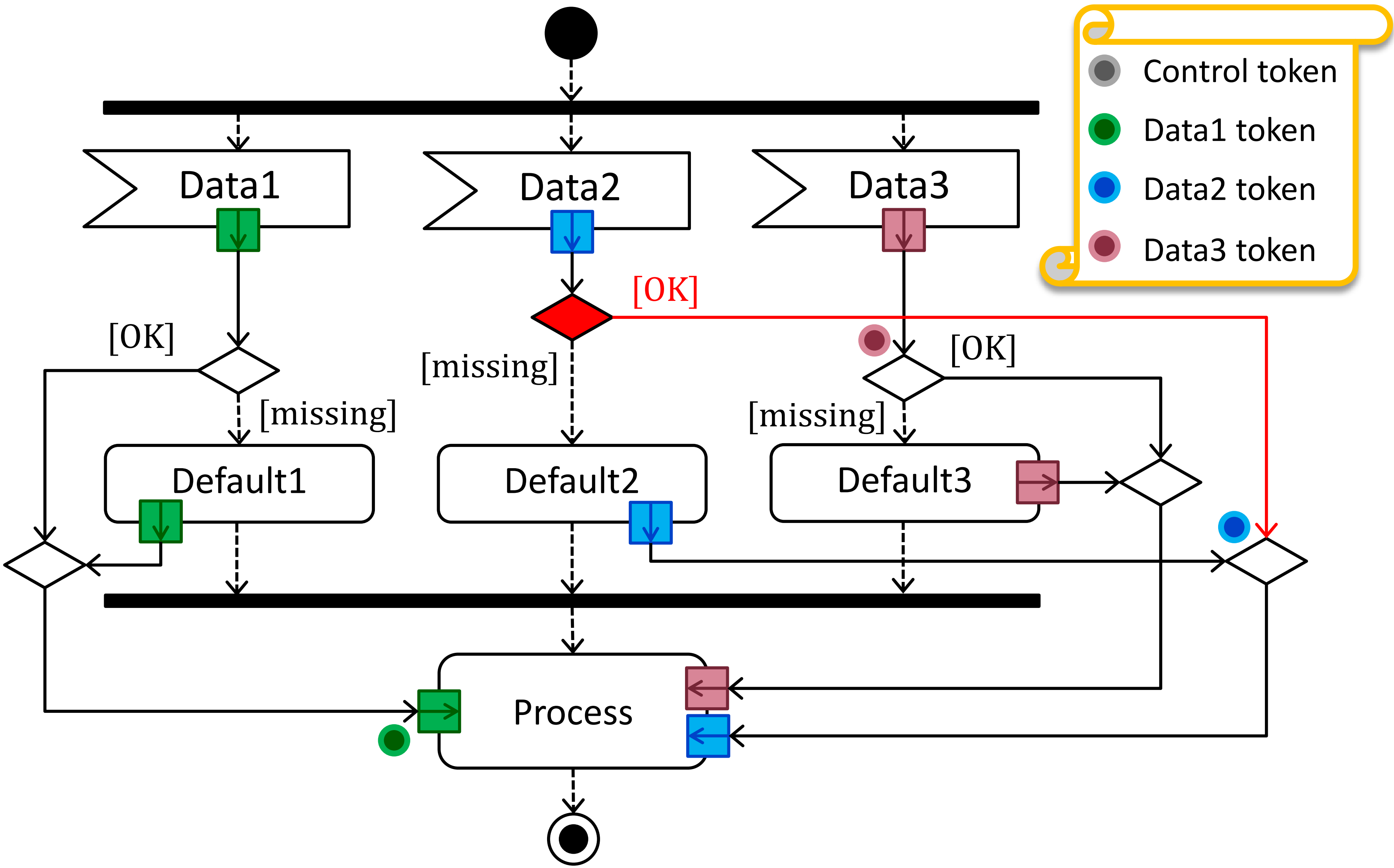


Example: Process collected data

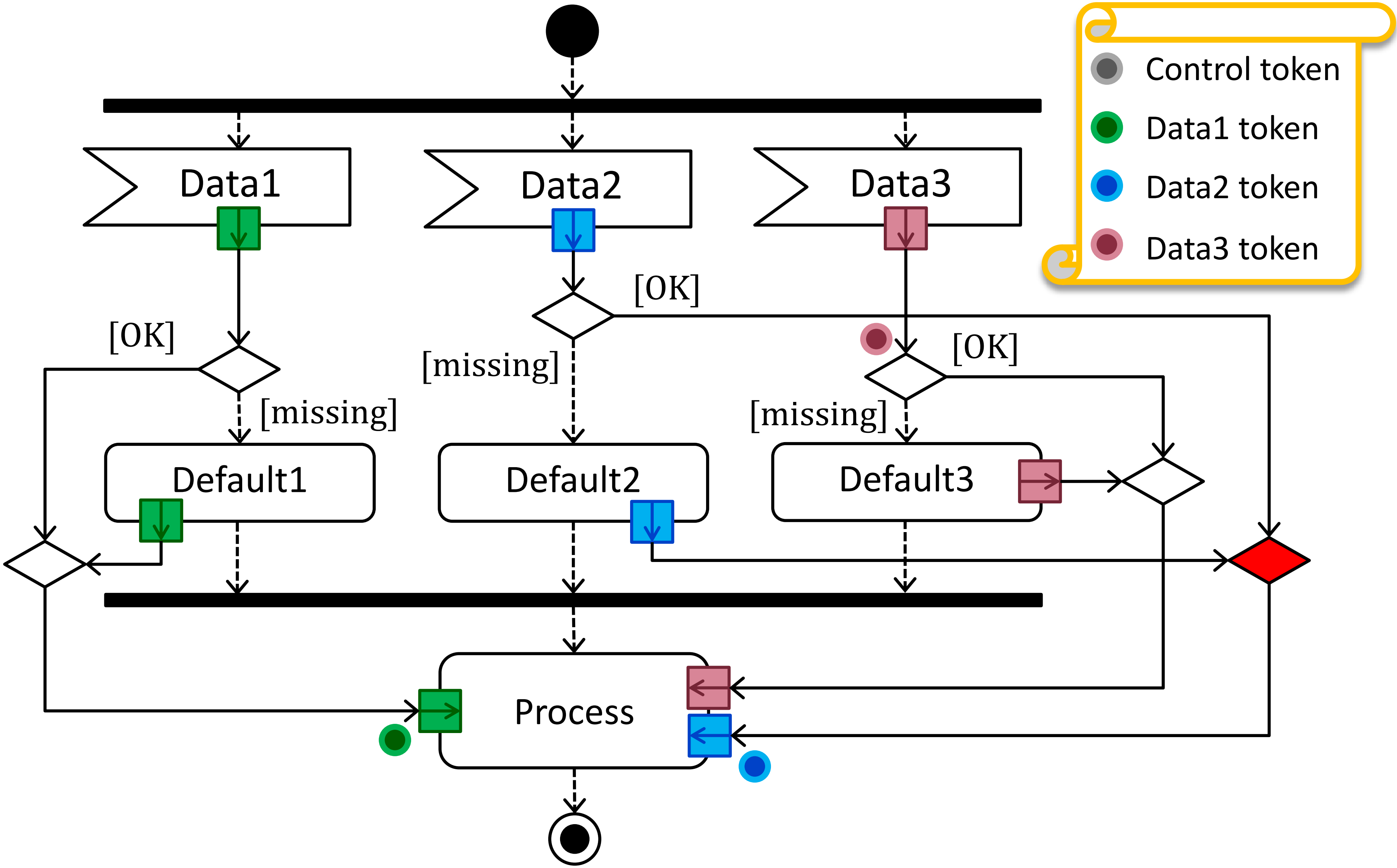


MŰEGYETEM 1782

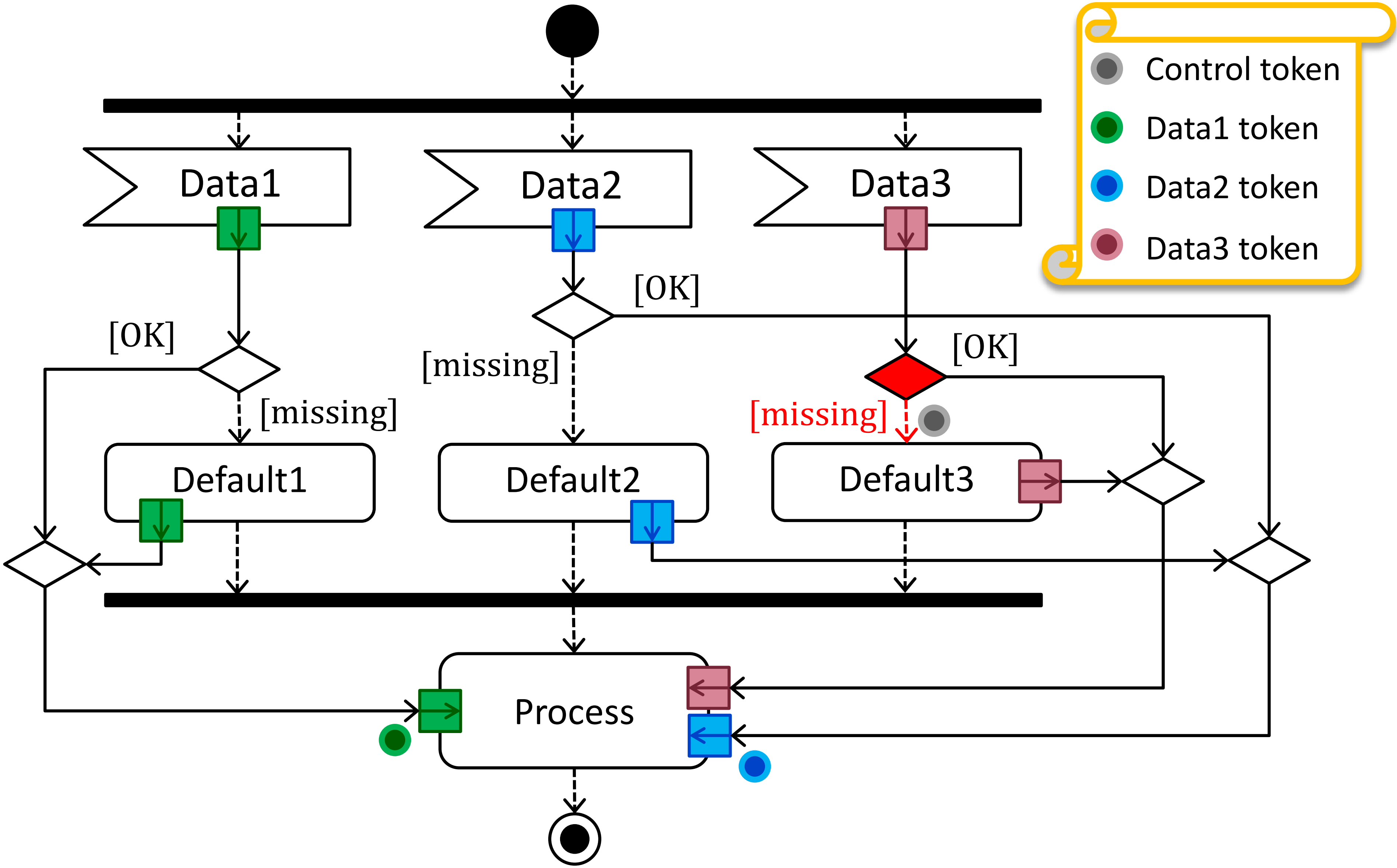
Example: Process collected data



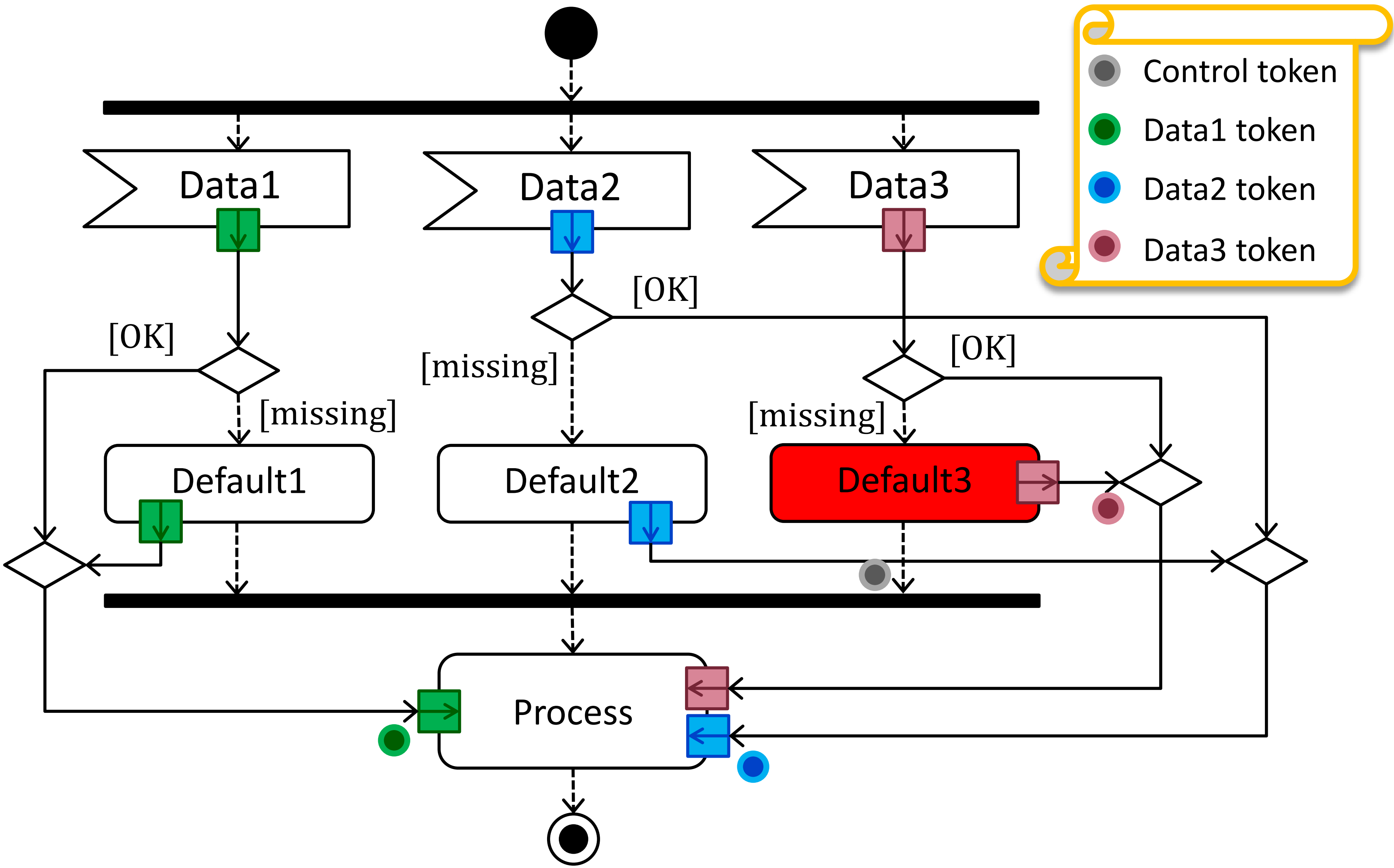
Example: Process collected data



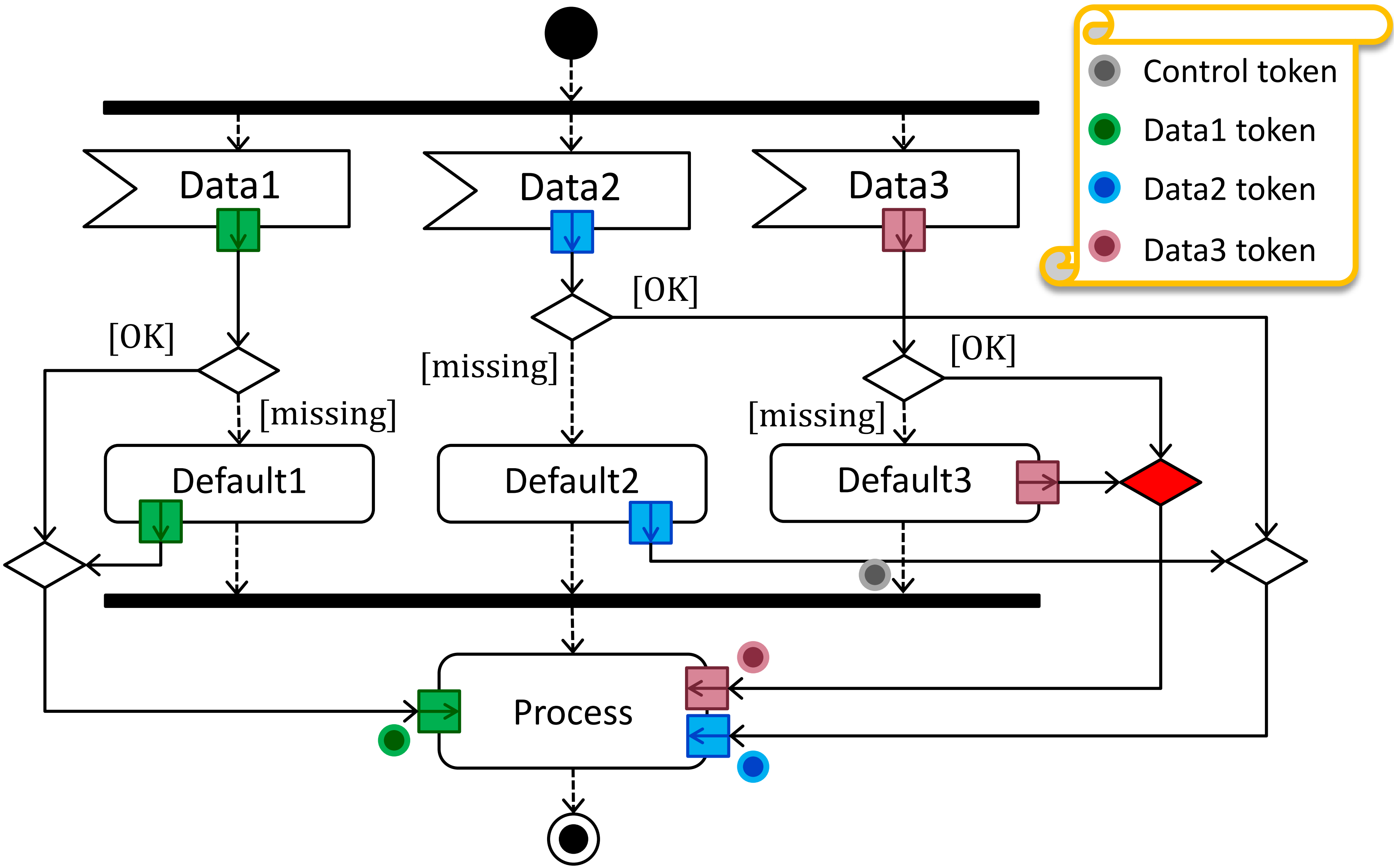
Example: Process collected data



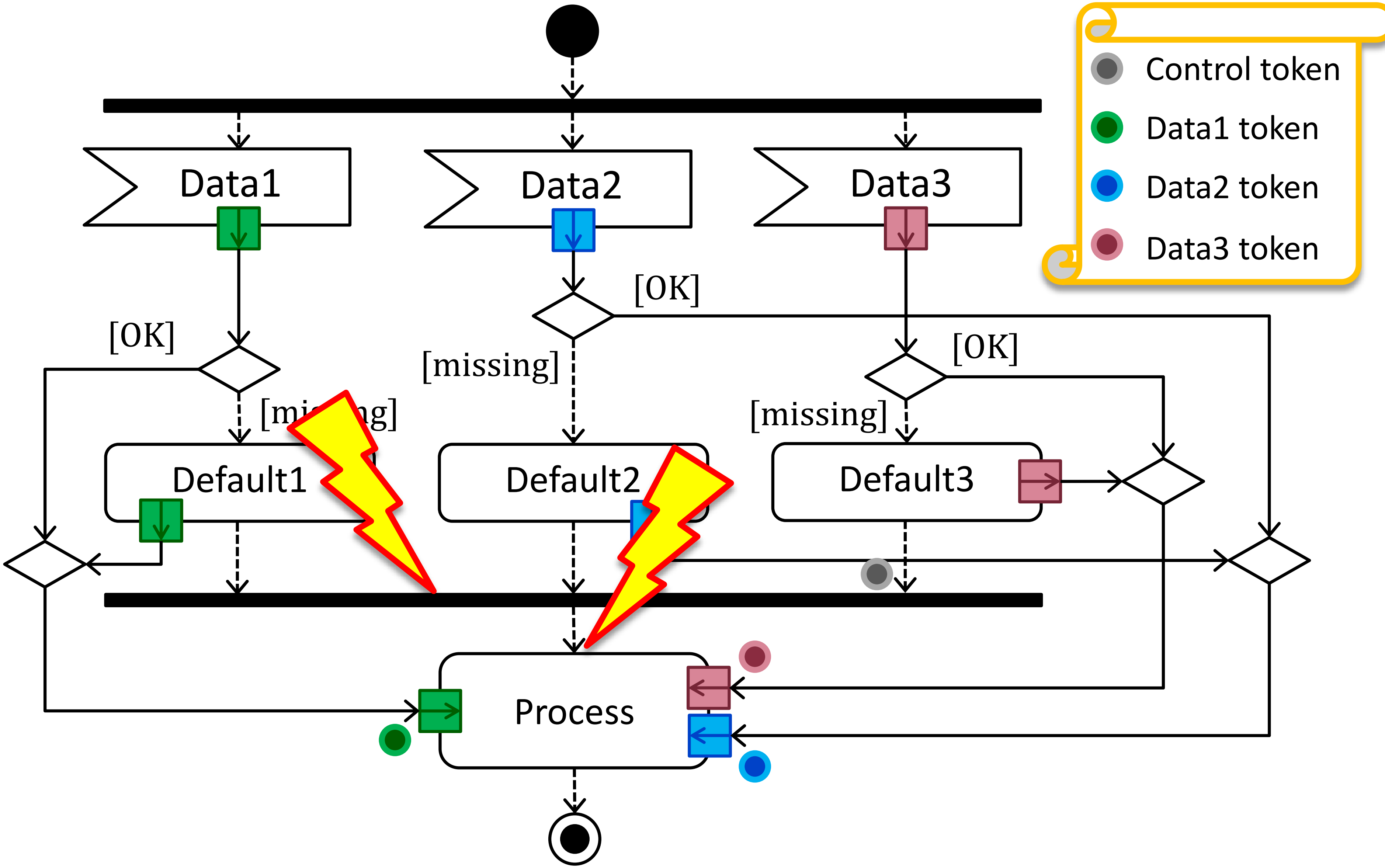
Example: Process collected data



Example: Process collected data



Example: Process collected data



MODELING WITH UML ACTIVITY DIAGRAMS

Modeling high-level processes

Modeling low-level activites

Deadlock, Ambiguity & Completeness

Best practices

Modeling high-level processes

Describe **system-level processes**

- As a refinement for Use Case Diagrams
  - **Use case flows:**
    - Action = Use Case
    - *“In what order can use cases be executed?”*
    - Typical and exceptional scenarios
  - **Use case scenarios:**
    - Actions are informal steps
    - *“What happens when a Use Case is executed?”*
    - Actions may later be refined into Activities
      - …and allocated to functional blocks

Modeling low-level activities

Describe low-level behavior to be implemented

- As a refinement of **operations**
  - Can describe the control and data flow of the method
  - With fUML: **executable models**
- As the behavior to execute in **state-based models**
  - Reactions to an event/signal
  - Continuous behavior in a certain state
- As an alternative to **Interaction Digrams**
  - Specify communication **and** internal behavior
  - Relying on Allocation Partitions

Deadlock, Ambiguity & Completeness

- **Deadlock-freeness:**
  - Control must always reach a Final node
  - Often due to incorrect use of control structures
    - Can be avoided by well-structuredness (see System Modeling)
- **Unambiguity:**
  - There shall never be more than one condition that evaluates to true at the same decision

    → Non-deterministic behavior
- **Completeness:**
  - There shall always be at least one condition that evaluates to true at the same decision

    → Deadlock

Best practices

- How to **build** a model?
  1. Model the **typical (primary) control flow** first
  2. Add **alternate** and **exceptional** paths
  3. Identify and model **data** and **data flows**
  4. Decompose the initial model by **refining Actions**
  5. **Allocate** Actions to functional blocks
- How to build a **good** model?
  - Always add a Final node to indicate the end of activity
    - Add multiple ones to indicate different or abnormal outcomes
  - Avoid ambiguity and incompleteness
  - Strive to build a **well-structured** model

RELATIONS TO OTHER DIAGRAMS

Class/Block Diagram

Activity Diagram

Interactions

Use case diagram, Block diagram

Use Case Diagram

- Refines use cases
  - Use case flow
  - Use case scenario

Block Diagram

- Allocates activities to blocks
- Defines main (continuous) behavior of blocks
- Defines behavior of operations
- Defines usage of data in a process

Interactions, State Machines

Interactions

- Refines/Extends Interactions
  - Modeling of communication and internal behavior

State Machines

- Defines behavior of actions in State Machines
  - How to react to an event
  - What to do in a state