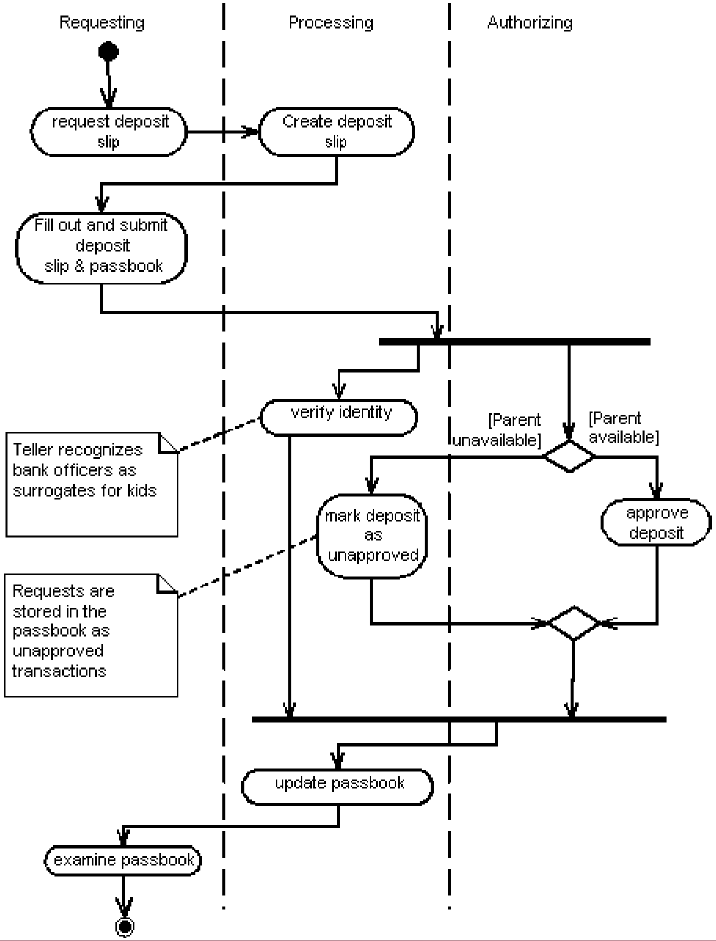
# Business Process Models in Practice

**Budapest University of Technology and Economics**
**Fault Tolerant Systems Research Group**

# UML Activity Diagram

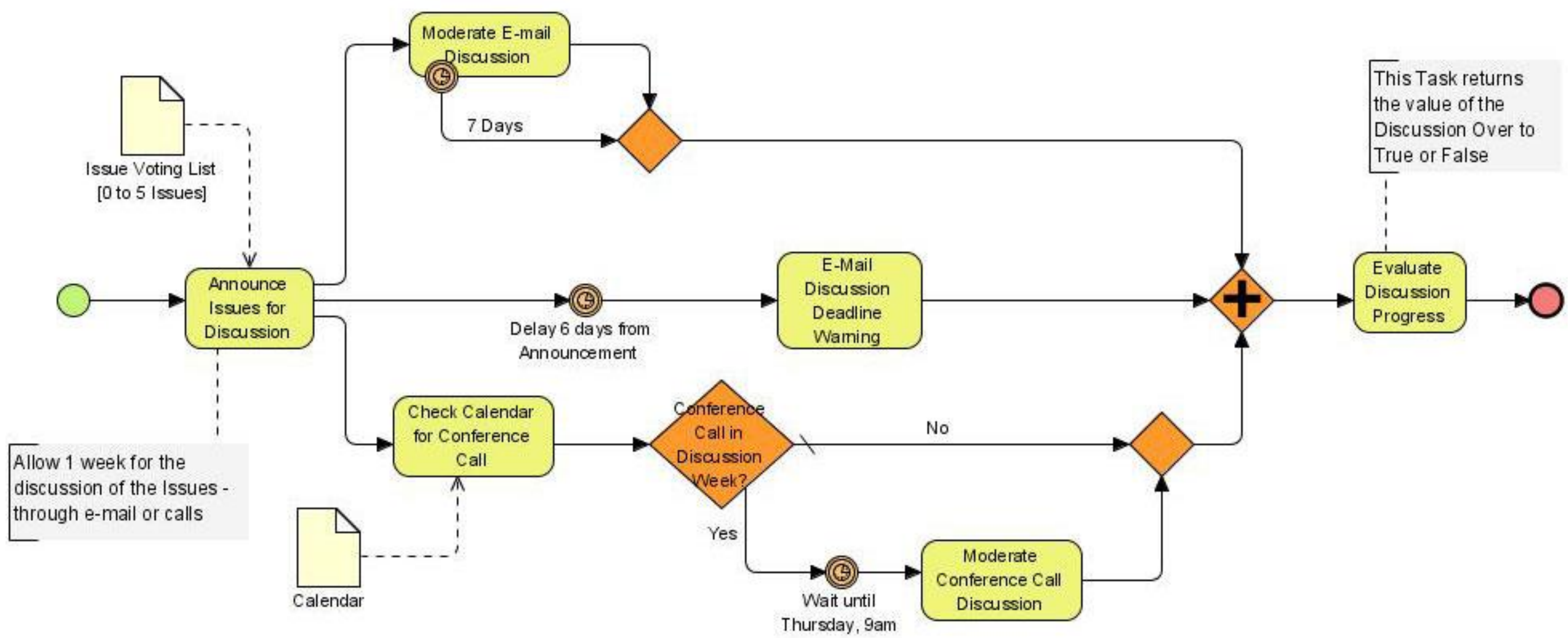
- Standardized syntax, with extensions
  - In details: see Software Technology course



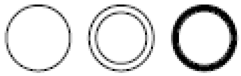
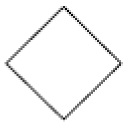
# Business Process Modeling Notation (BPMN)

- Business Process Management Initiative (BPMI)
  - May 2004: BPMN 1.0 specification
  - 2011: BPMN 2.0, final
- Goals
  - Clarity
    - User
  - Business analyst
    - Initial process plan
  - Technical developer
    - Implementation
    - Internal model for the purpose of automatic generation
    - BPEL4WS
  - End-user (monitoring, management)

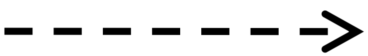
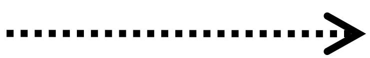
# BPMN example



# Data flow

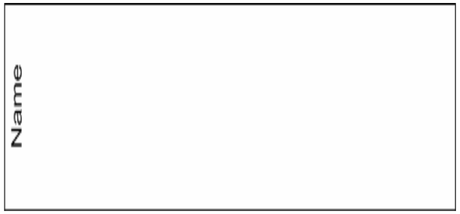
| | | |
|---|---|---|
| Event | State change<br>Cause-effect<br>Types of events:<br>Start, Intermediate, End |  |
| Activity | Atomic/composite<br>Task/subprocess |  |
| Gateway | Sequence<br>convergence/divergence<br>AND, OR, XOR, … |  |

# Connections

| | | |
|---|---|---|
| Sequence flow | Order of activities in the process |  |
| Message flow | Information exchange between two independent process participant |  |
| Association | Data, text, etc. |  |

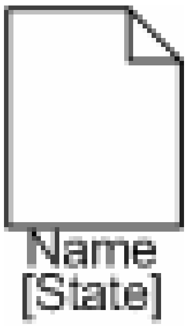
# Swim lanes

Pool | Represents a participant

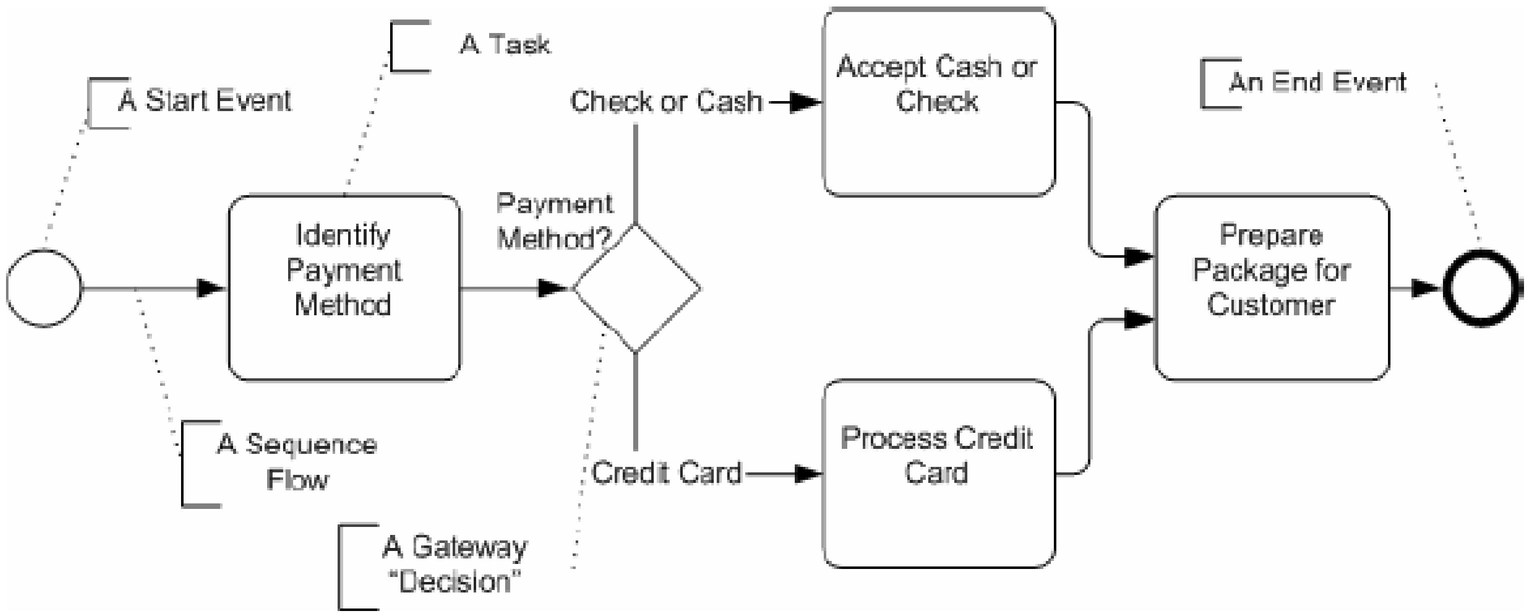



Lane | Categorisation of activities

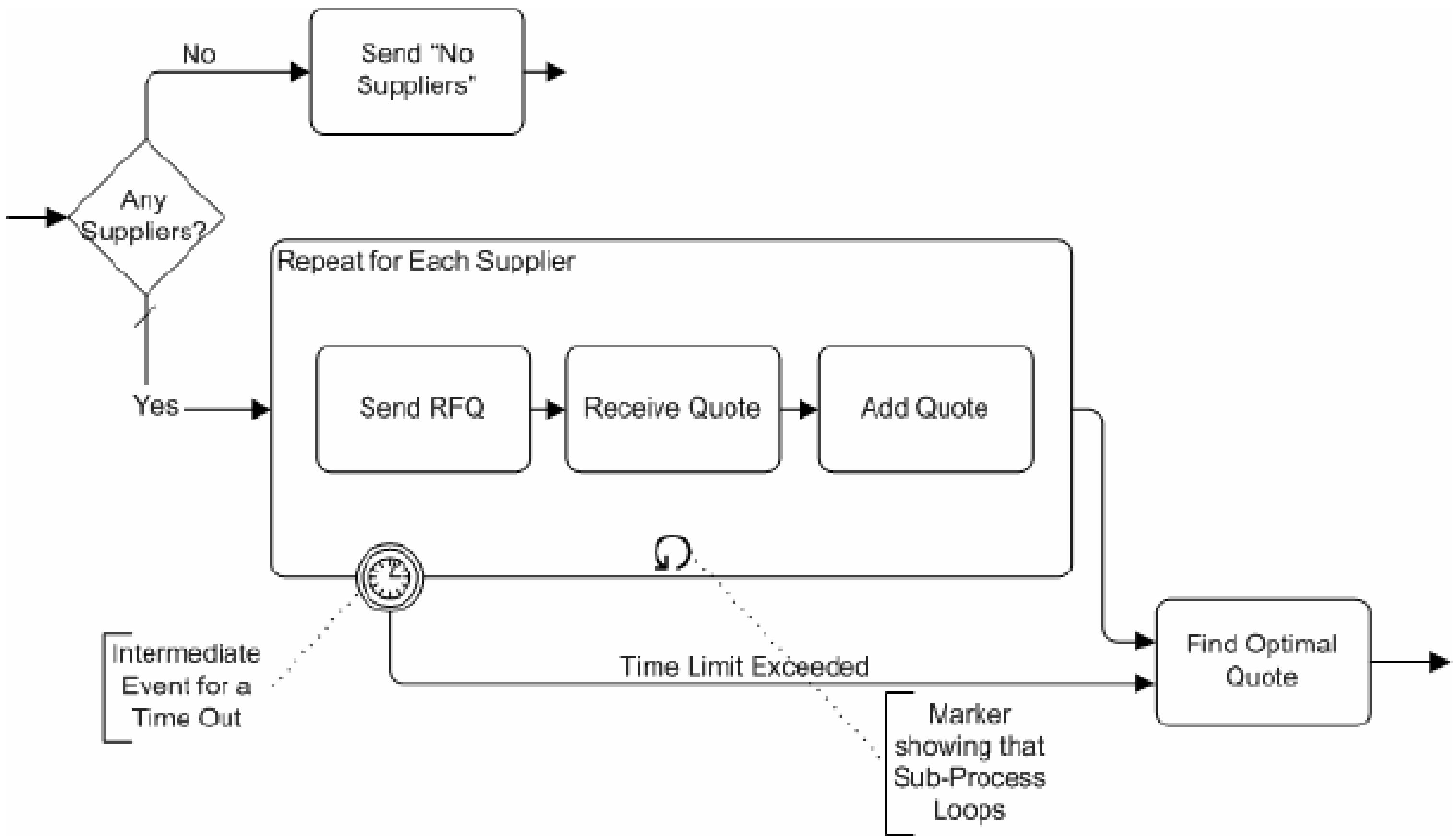
# Artefact

| | | |
|---|---|---|
| Data object | Symbolic token |  |
| Group | Grouping activities |  |
| Annotation | Additional text information (comment) |  |

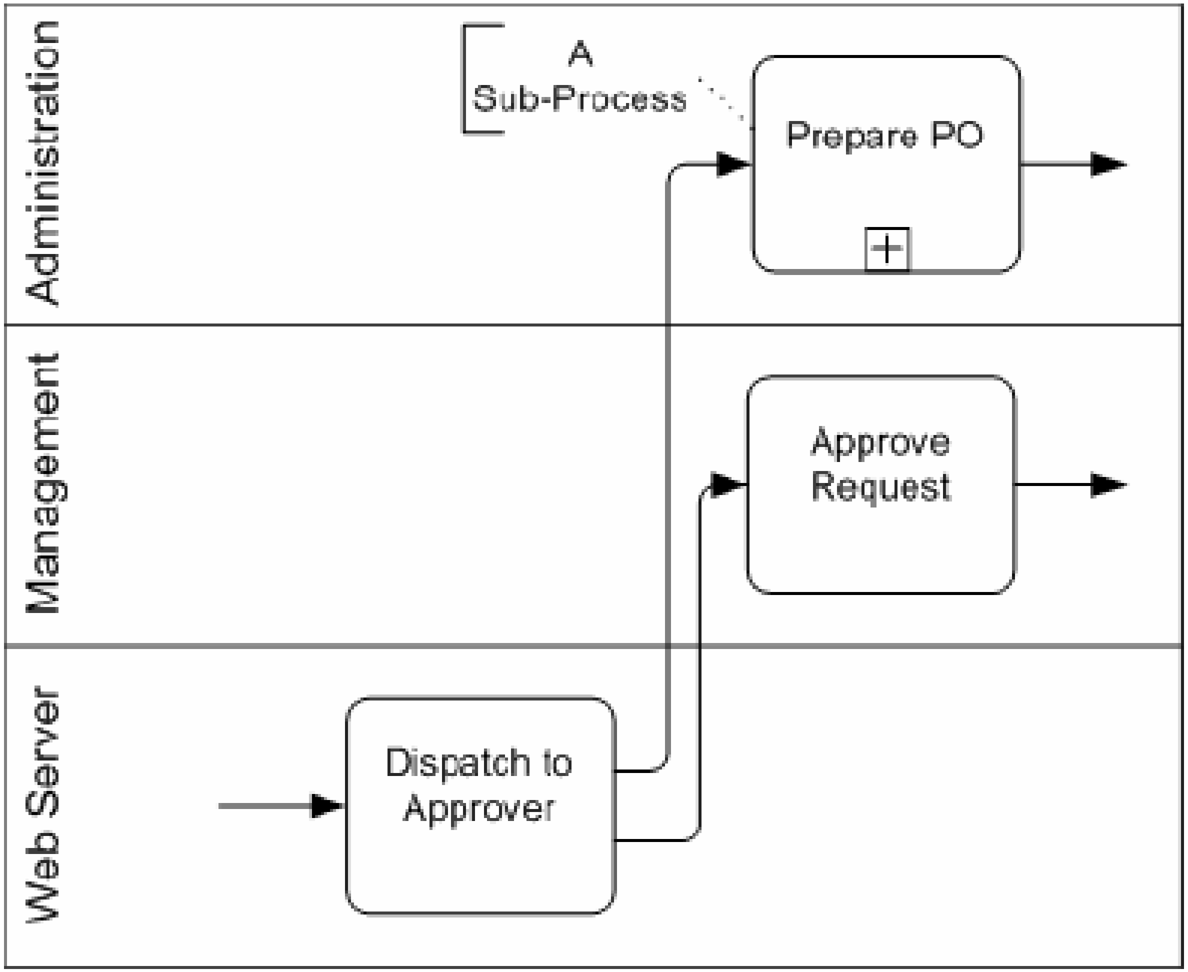
# Example


A Task
A Start Event
Check or Cash
Accept Cash or Check
An End Event
Identify Payment Method
Payment Method?
Prepare Package for Customer
A Sequence Flow
Credit Card
Process Credit Card
A Gateway "Decision"

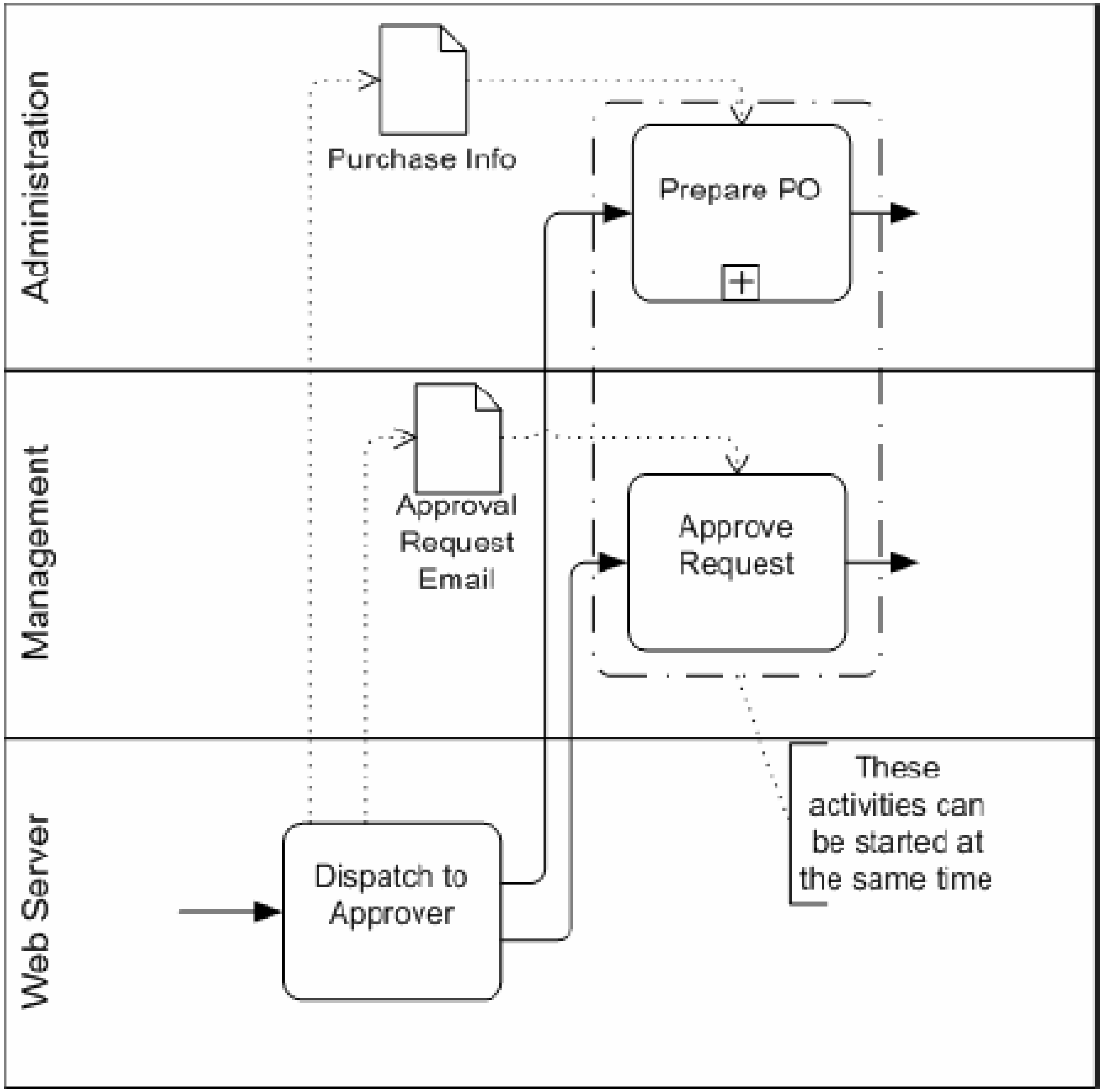
# Hierarchical Modelling



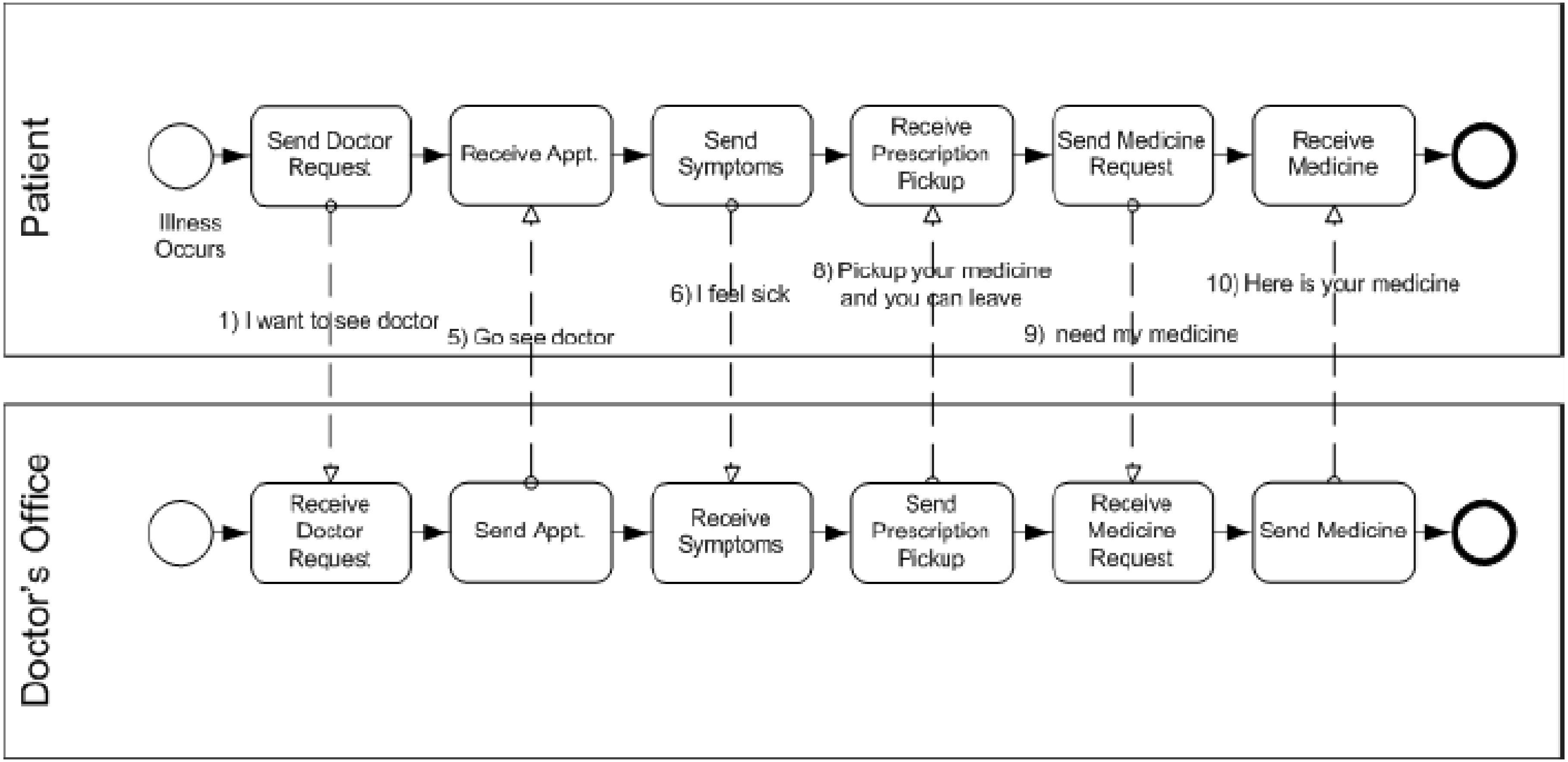
# Role Separation



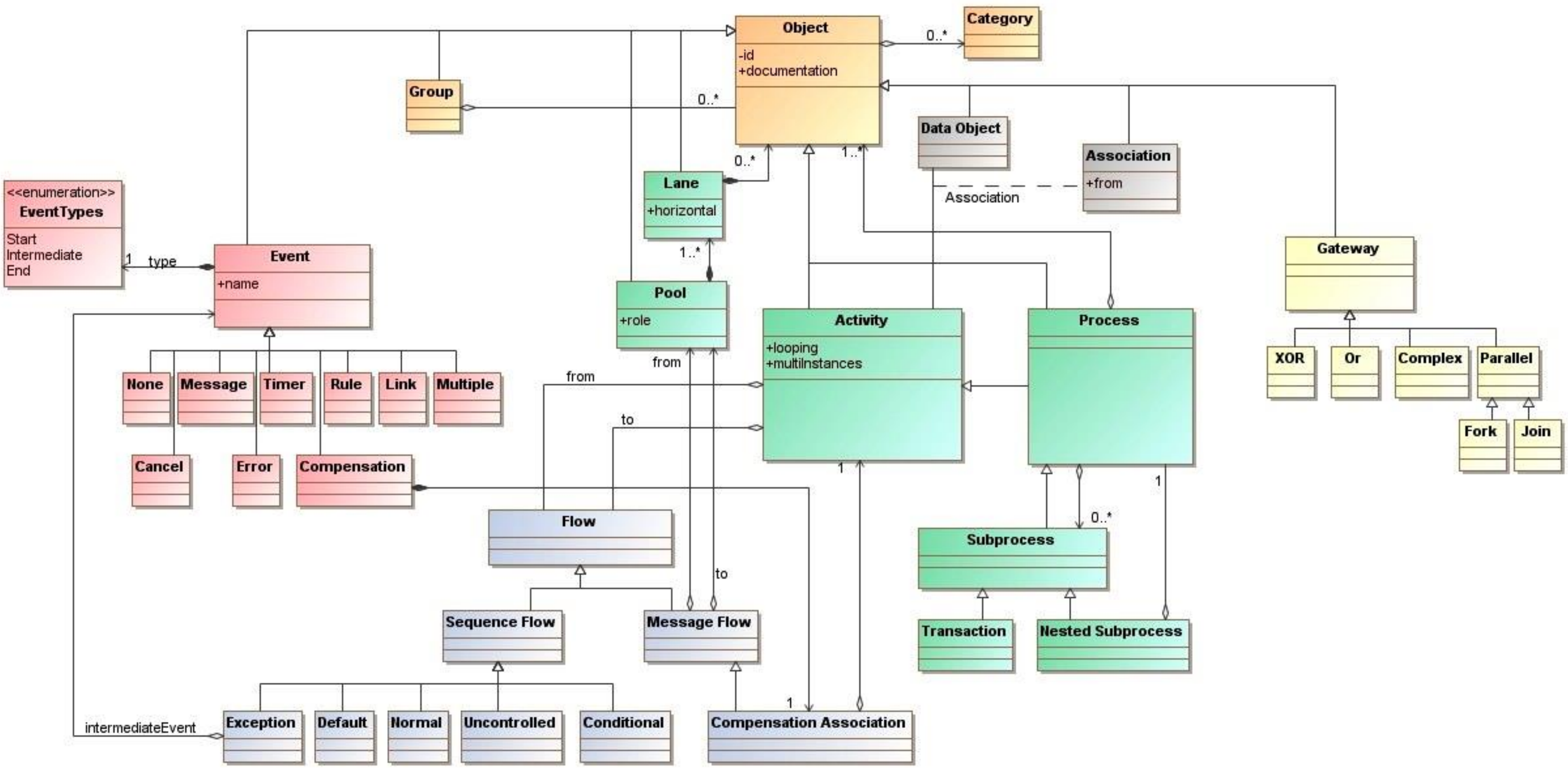
# Data, Grouping



# Cooperating (Sub)Processes

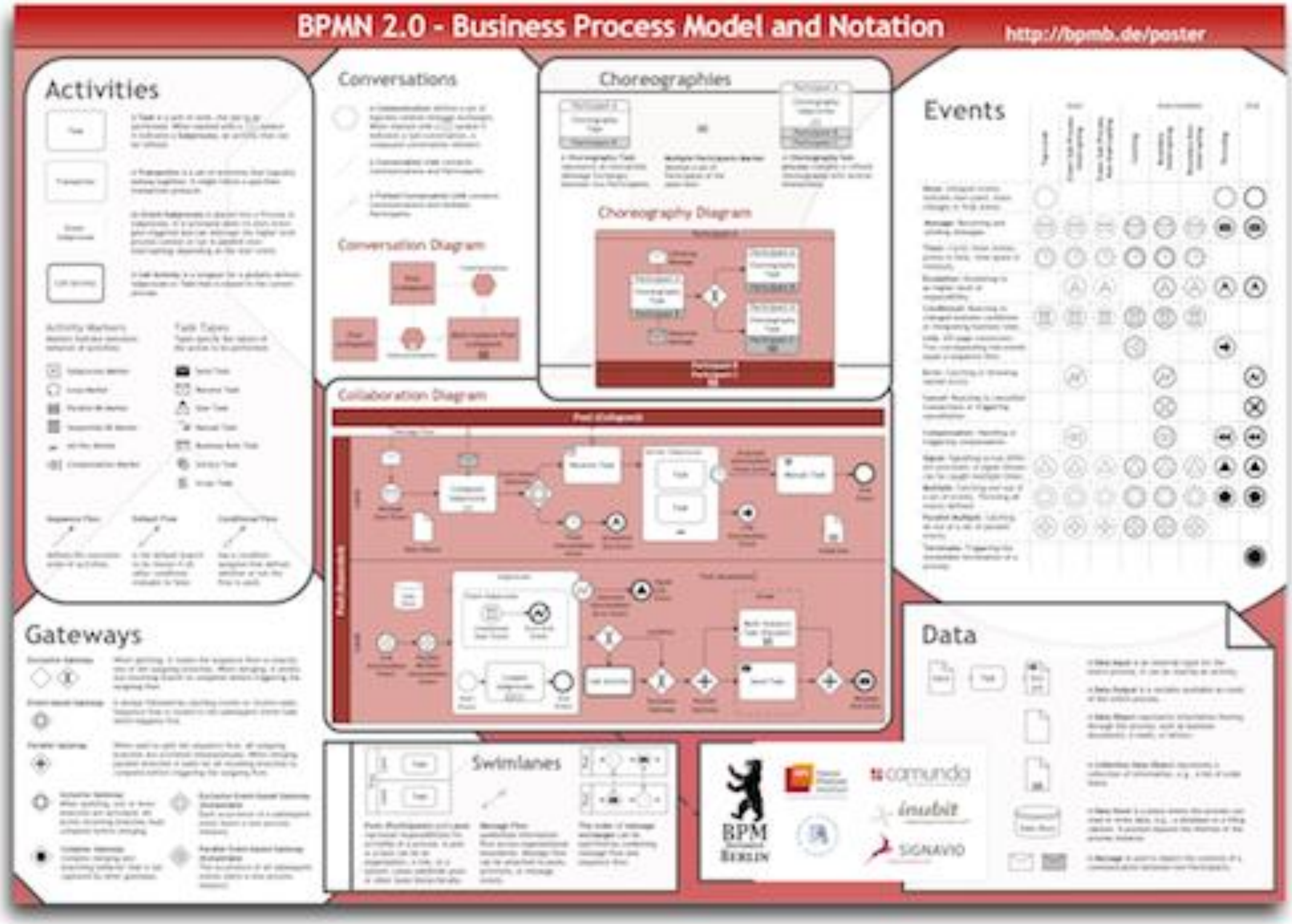


# BPMN Metamodel (simplified)




Source: http://www.wsper.org//

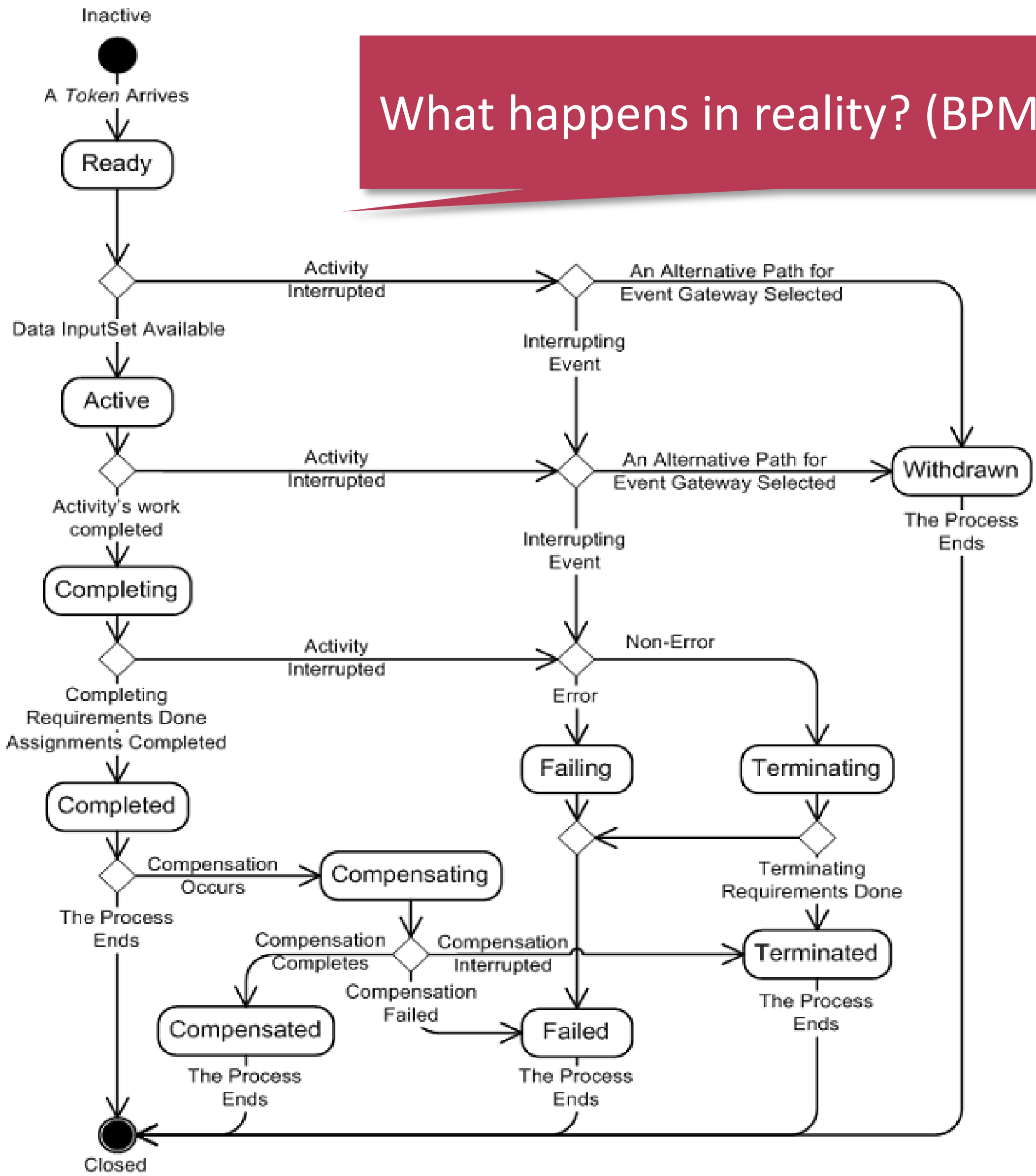
# Language Elements




Source: http://www.bpmb.de

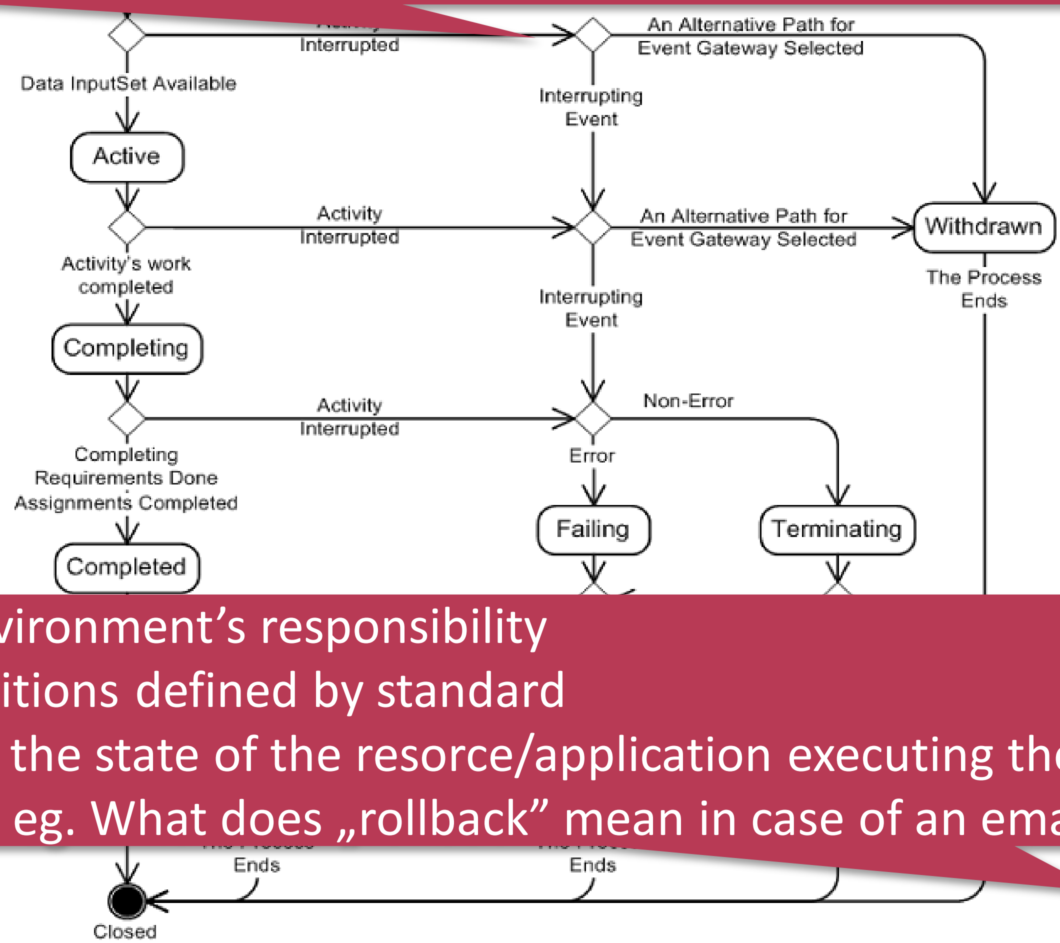
# Refined state machine of atomic activity

What happens in reality? (BPMN standard)



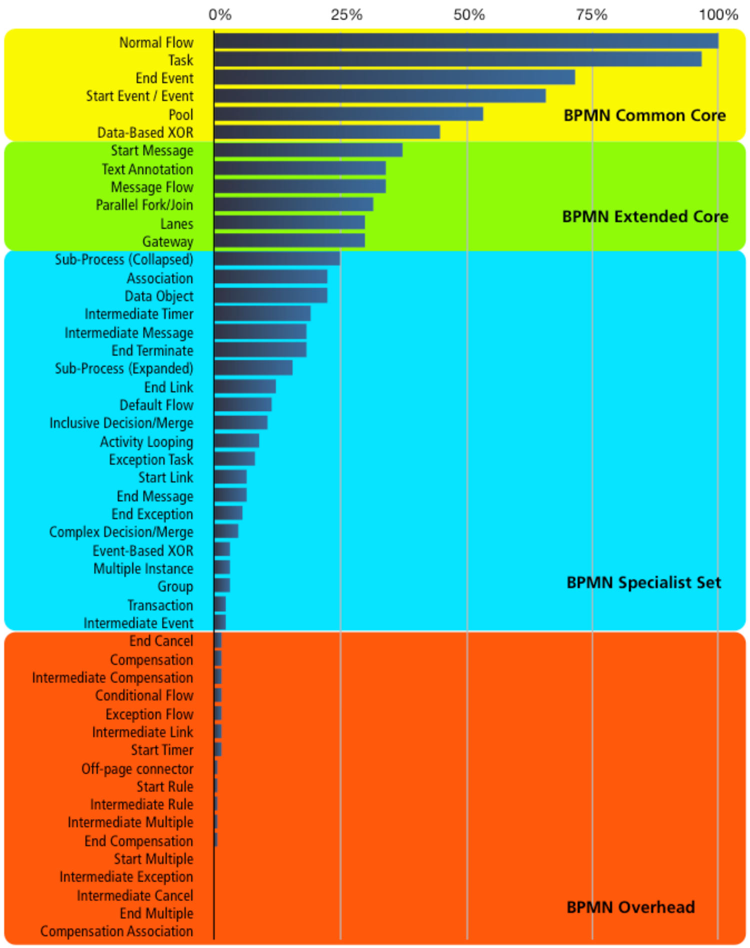
# Refined state machine of atomic activity

- Activity can be interrupted, rolled back, mistaken…




- Runtime environment's responsibility
- States/transitions defined by standard
- Differs from the state of the resorce/application executing the step!
- Design task: eg. What does „rollback" mean in case of an email

# „A statistics…”




Source: Process Modelling. What Really Matters
Keynote of Michael Rosemann @ UNISCON2009 conference

# Challenges

- Formalization of Domain Specific Knowledge
  - Libraries, templates
  - Inclusion of „Web2.0"
  - Efficient modelling (textual?)
- Consistency of the models
  - Static analysis: ~200 questions (BPEL2 standard)
  - Connecting process models and other ones
    - State machines, …
- Installation, resource configuration, ….

# BPMN Tools

- jBPM Designer
- Eclipse BPMN
- Tibco Business Studio
- **IBM Websphere Business Modeler**
- Intalio Designer
- BPMN Composer
- BPMN Designer
- Bonita Open Solution
- Adonis
- Activiti
- Obeo Designer

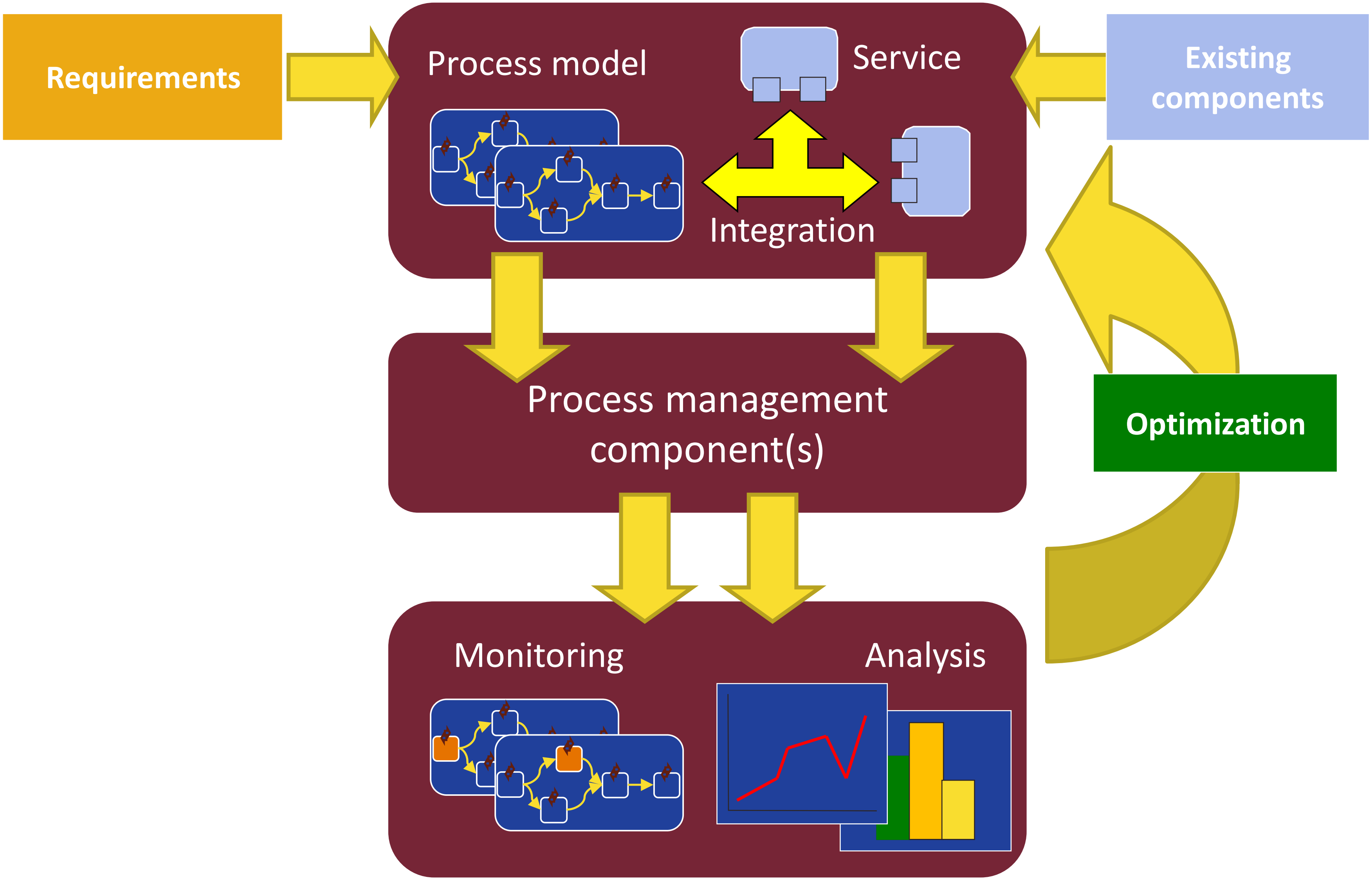
+ general modelling tools

# EXECUTING BUSINESS PROCESSES

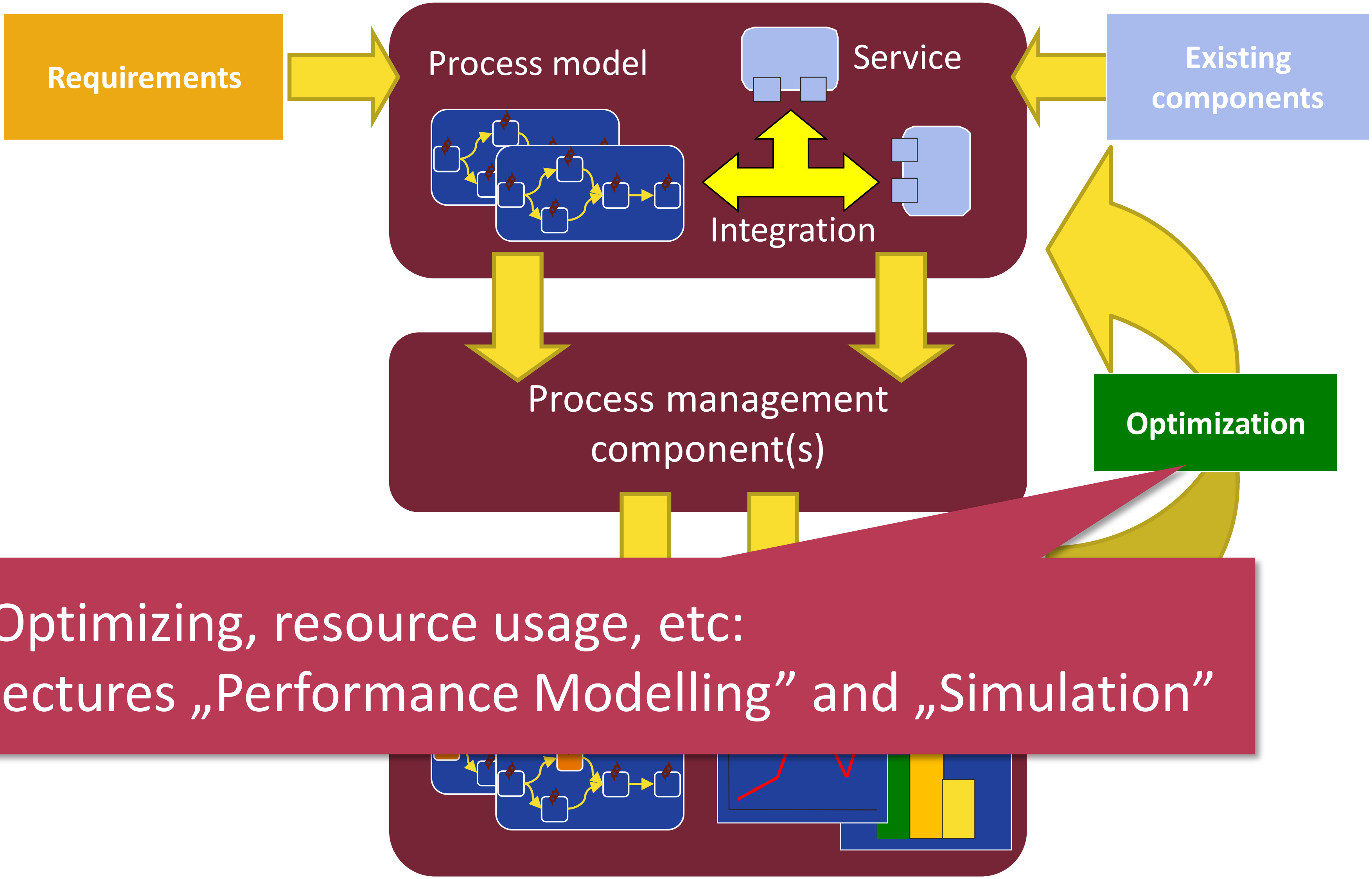
# Execution: “workflow engine”

- Managing life cycle of processes
  - Process templates
  - Instantiation, managing data
- Versioning, online update
- API for embedded/connected elements
  - REST, WS, EJB…
- Managing business rules (decisions)
- Human task
  - Can be displayed in browser
  - Managing permissions

# Process management



# Process management




Optimizing, resource usage, etc:
lectures „Performance Modelling” and „Simulation”

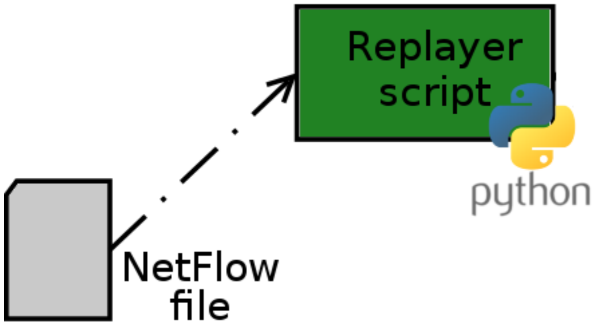
# CASE STUDY: STORM

Data processing using Apache Storm

(NÁDUDVARI Tamás:

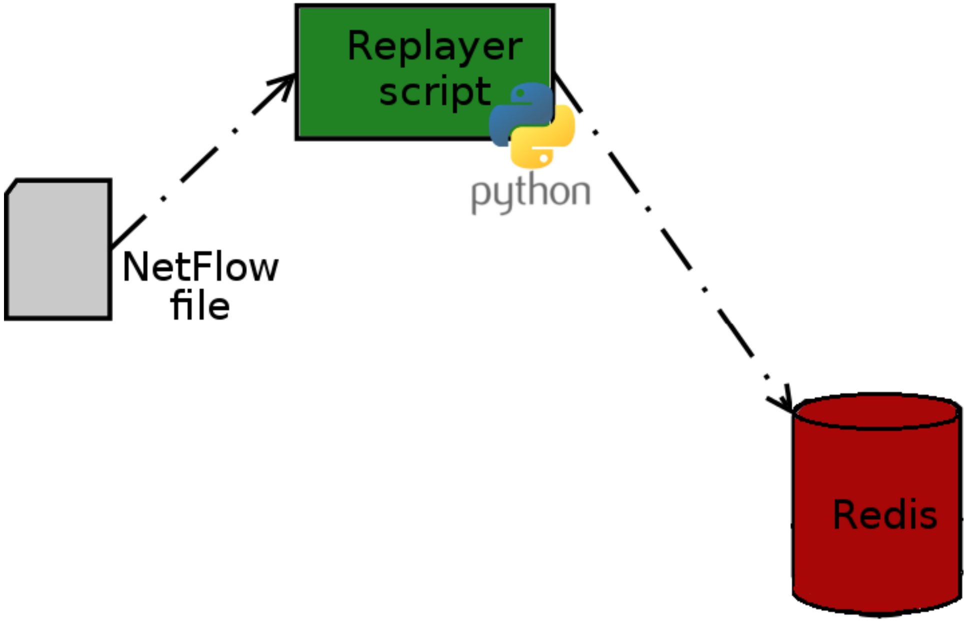
Stream processing based support for big data analysis)

# Application Data Flow



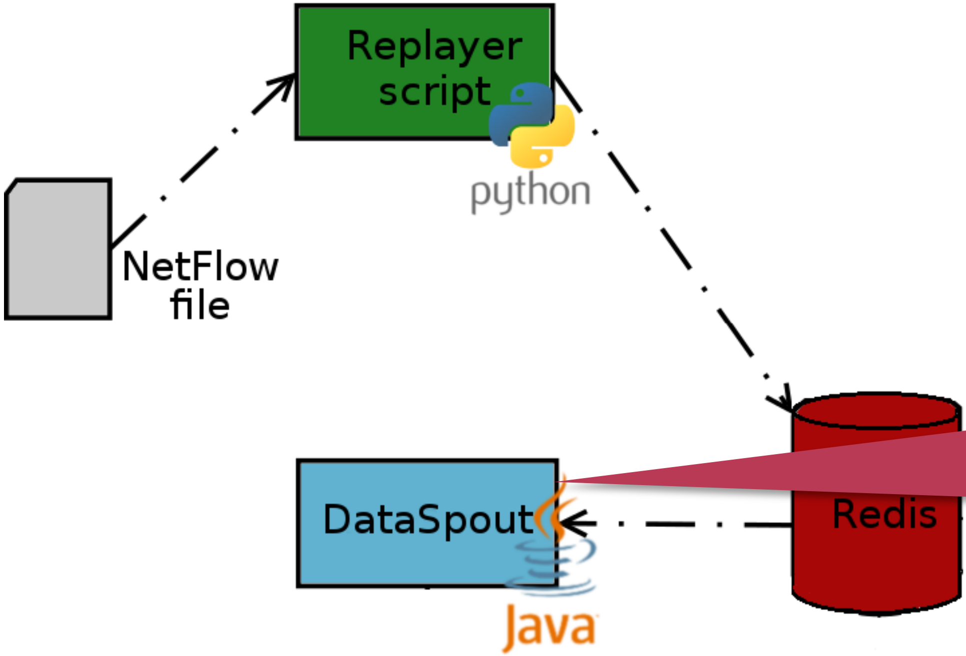

- Reading records that contain saved network data
- Records contain
  - source and target IP
  - time
  - # of distributed packages
  - quantity of data

# Application Data Flow



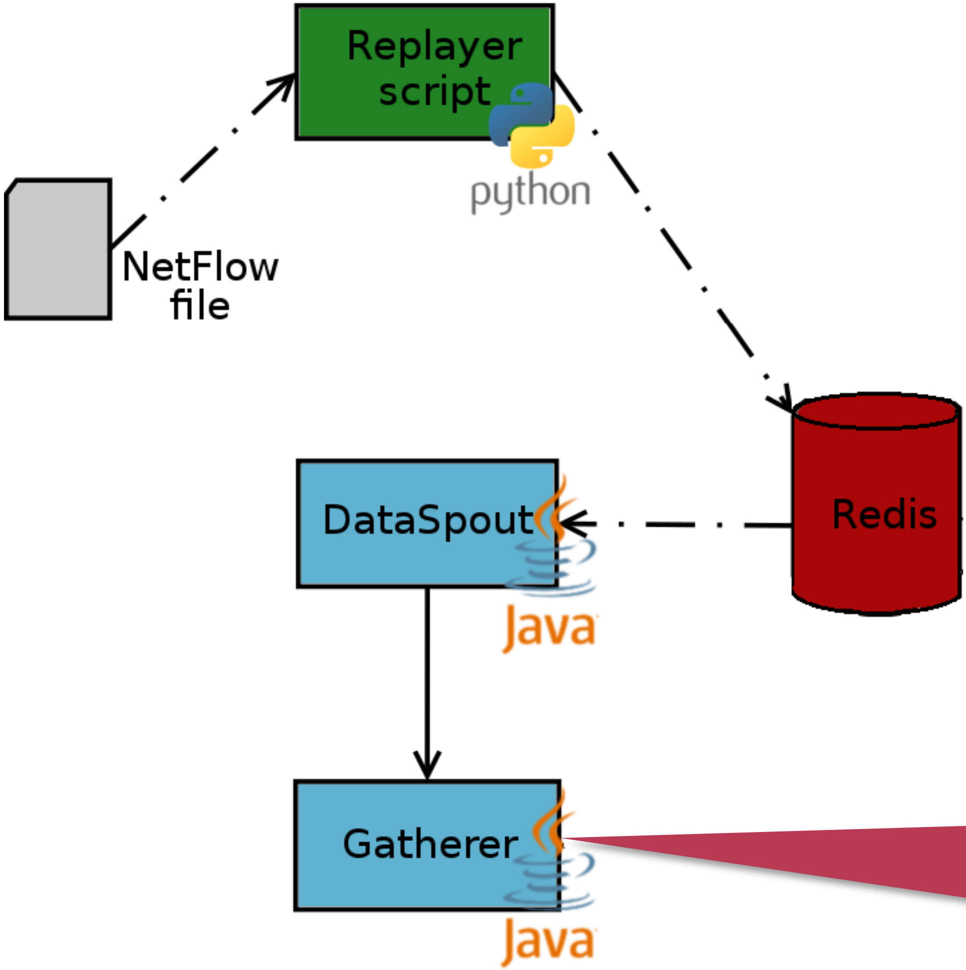

- Sending network records to a database

# Application Data Flow



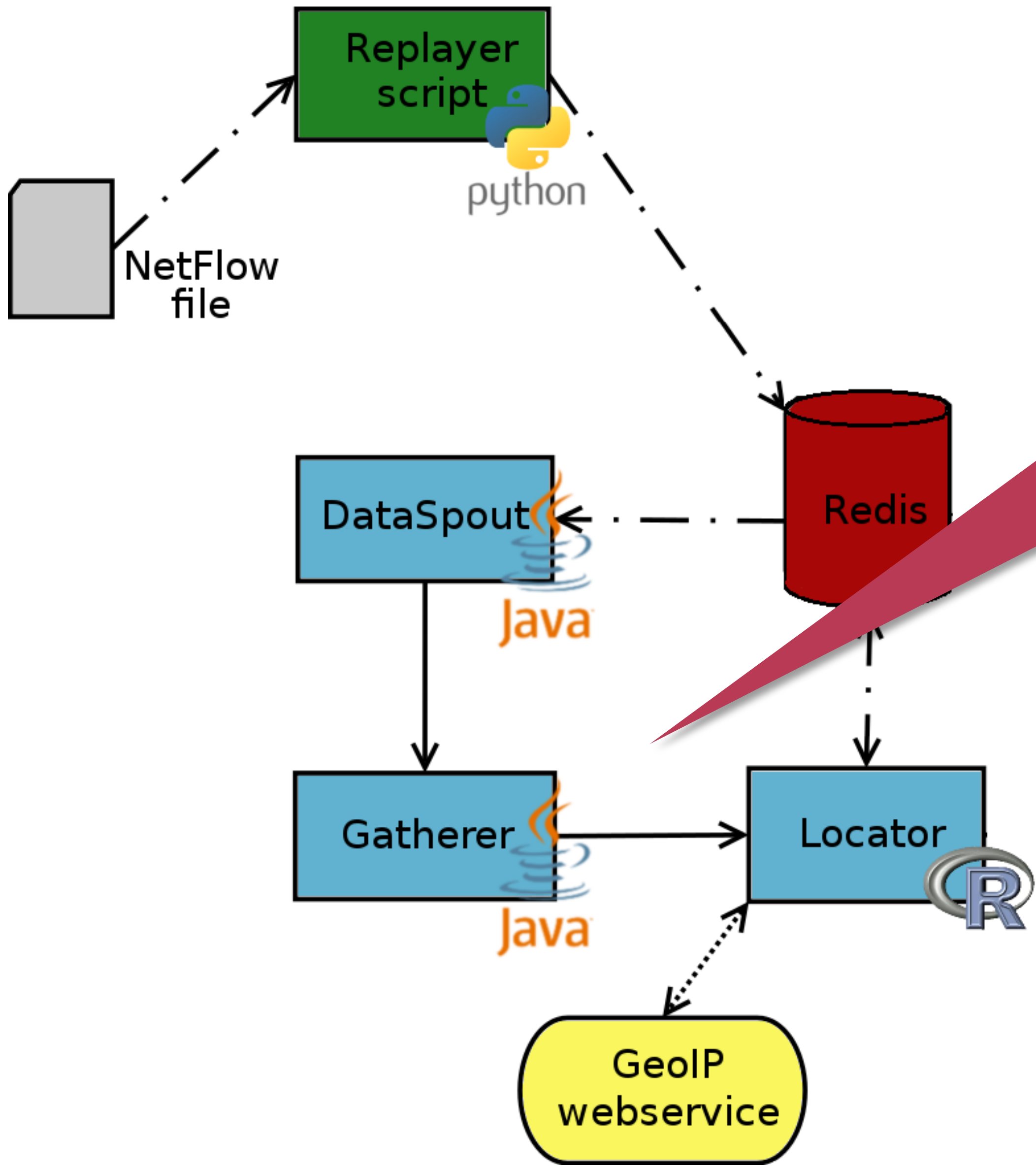

First Storm component reads records

# Application Data Flow



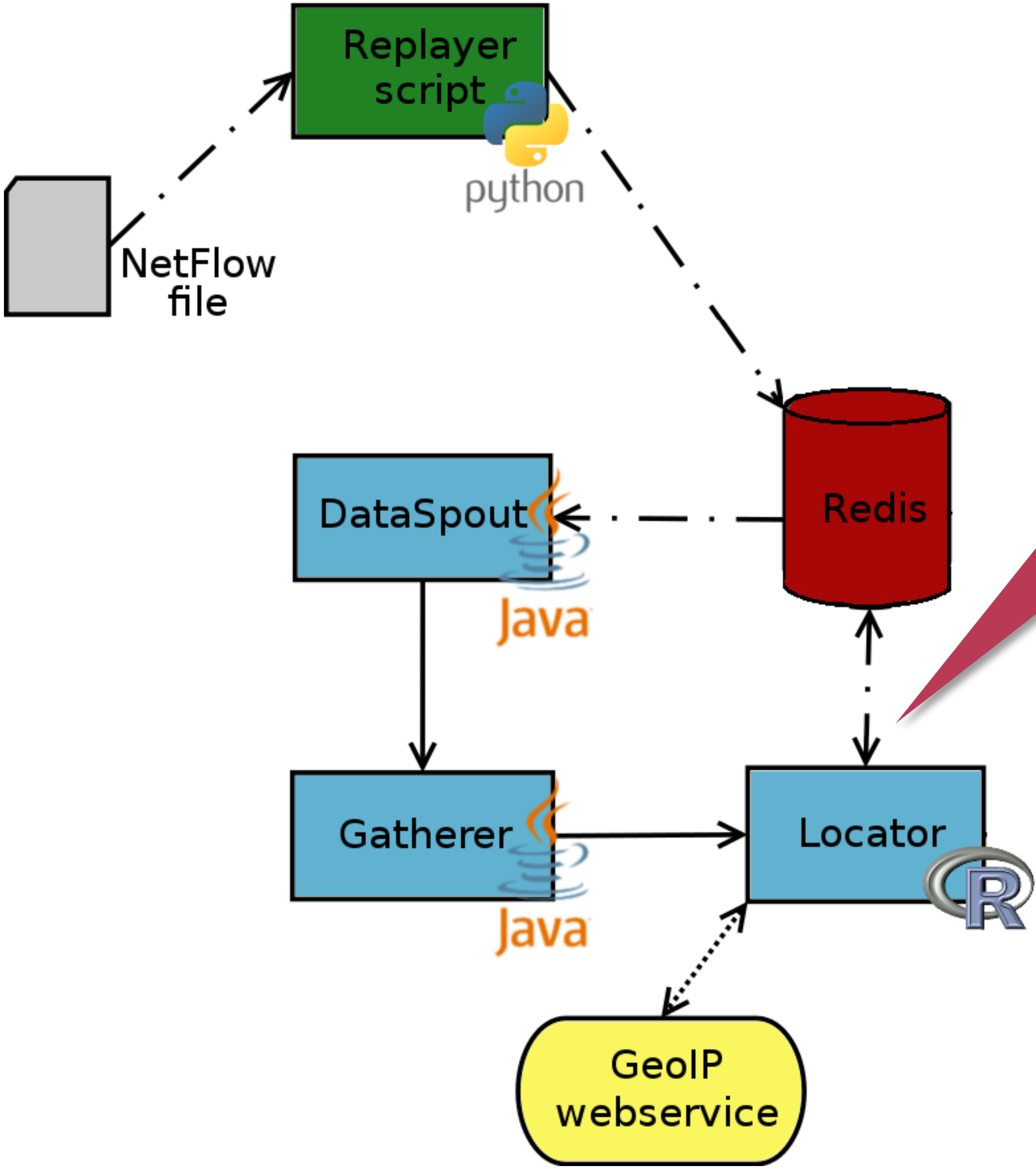

Cuts off irrelevant data from records

# Application Data Flow



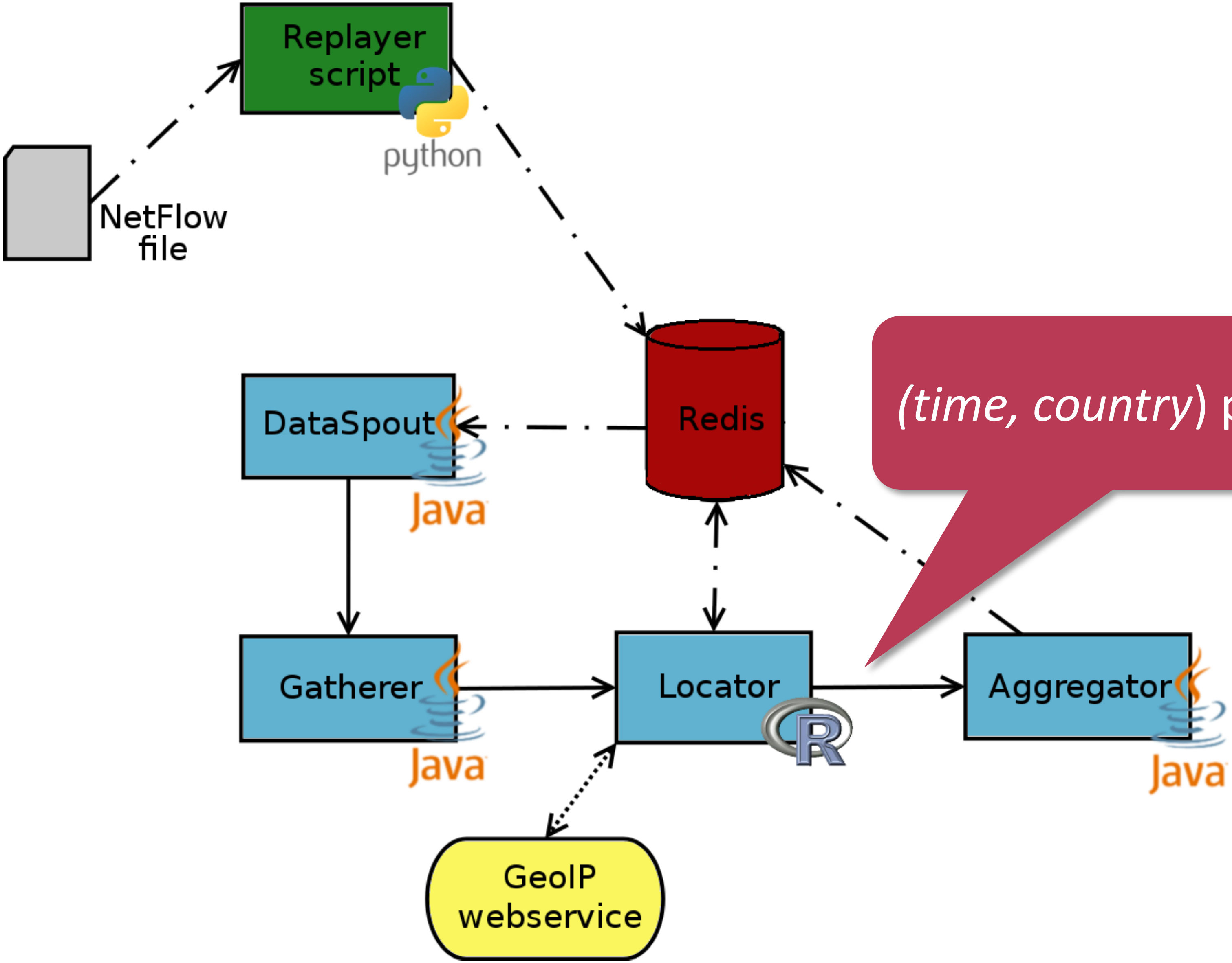

Only time – traget IP pairs are sent forward

# Application Data Flow



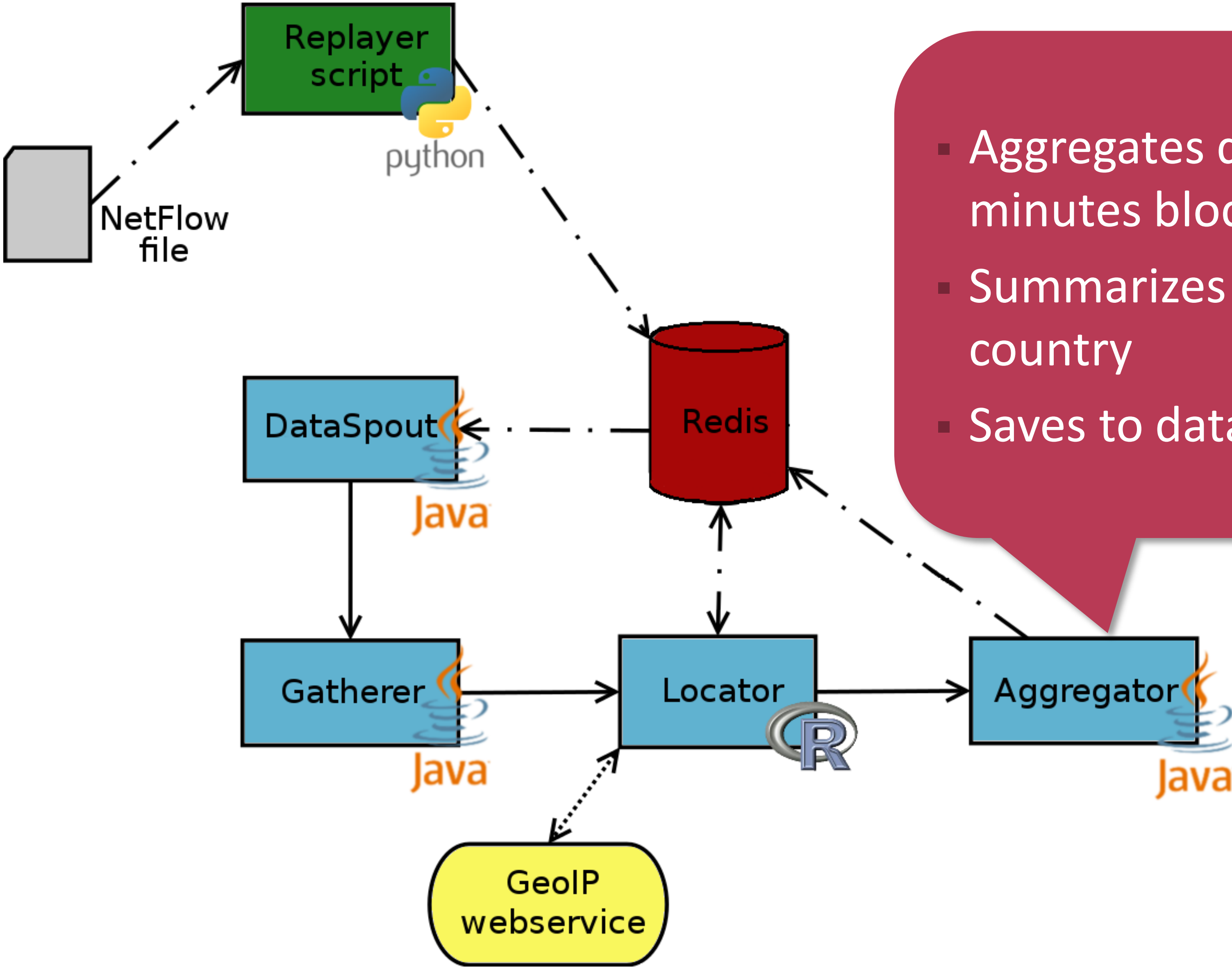

Finds the country corresponding to the IP address using an external service

# Application Data Flow



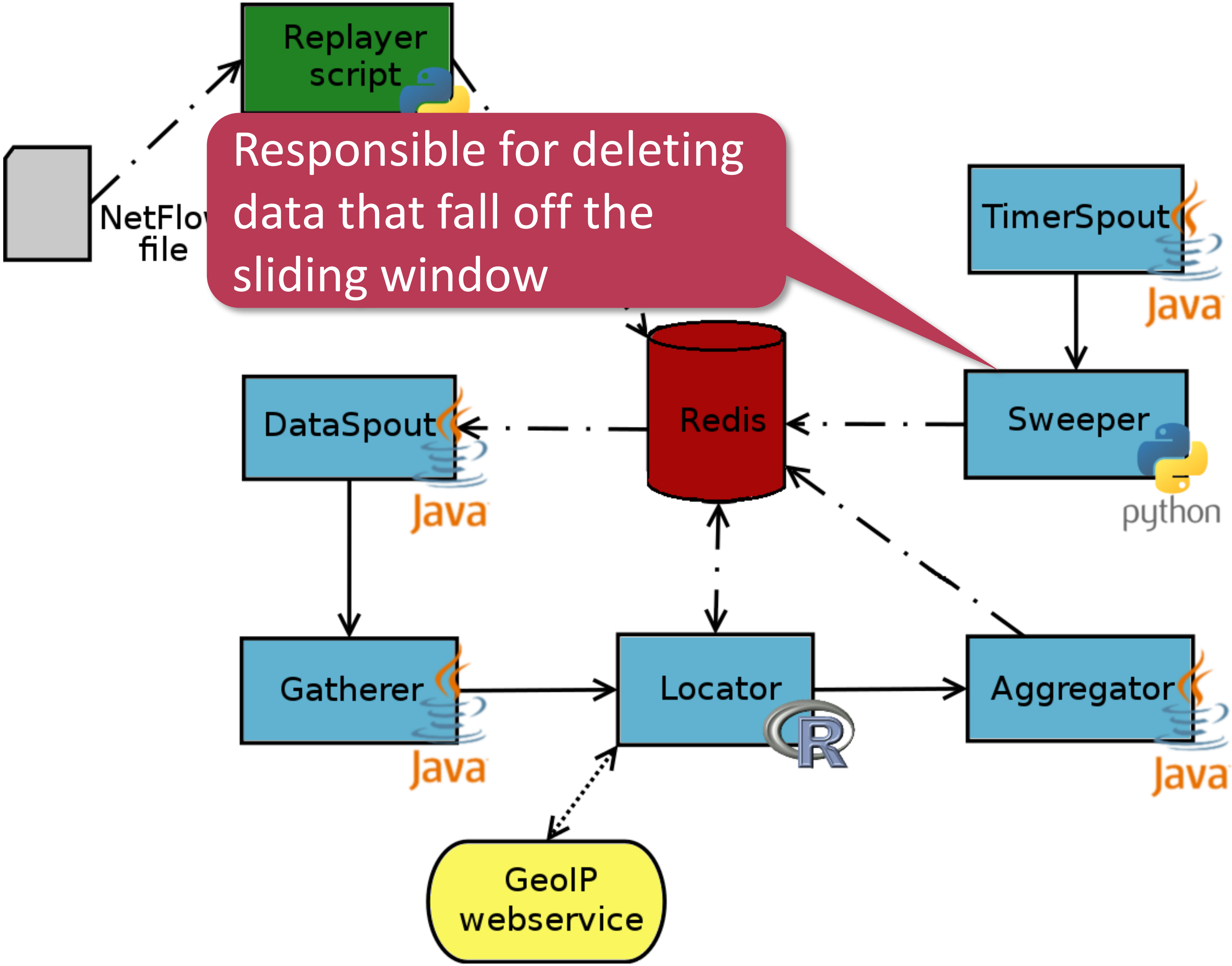

(*time, country*) pairs

# Application Data Flow

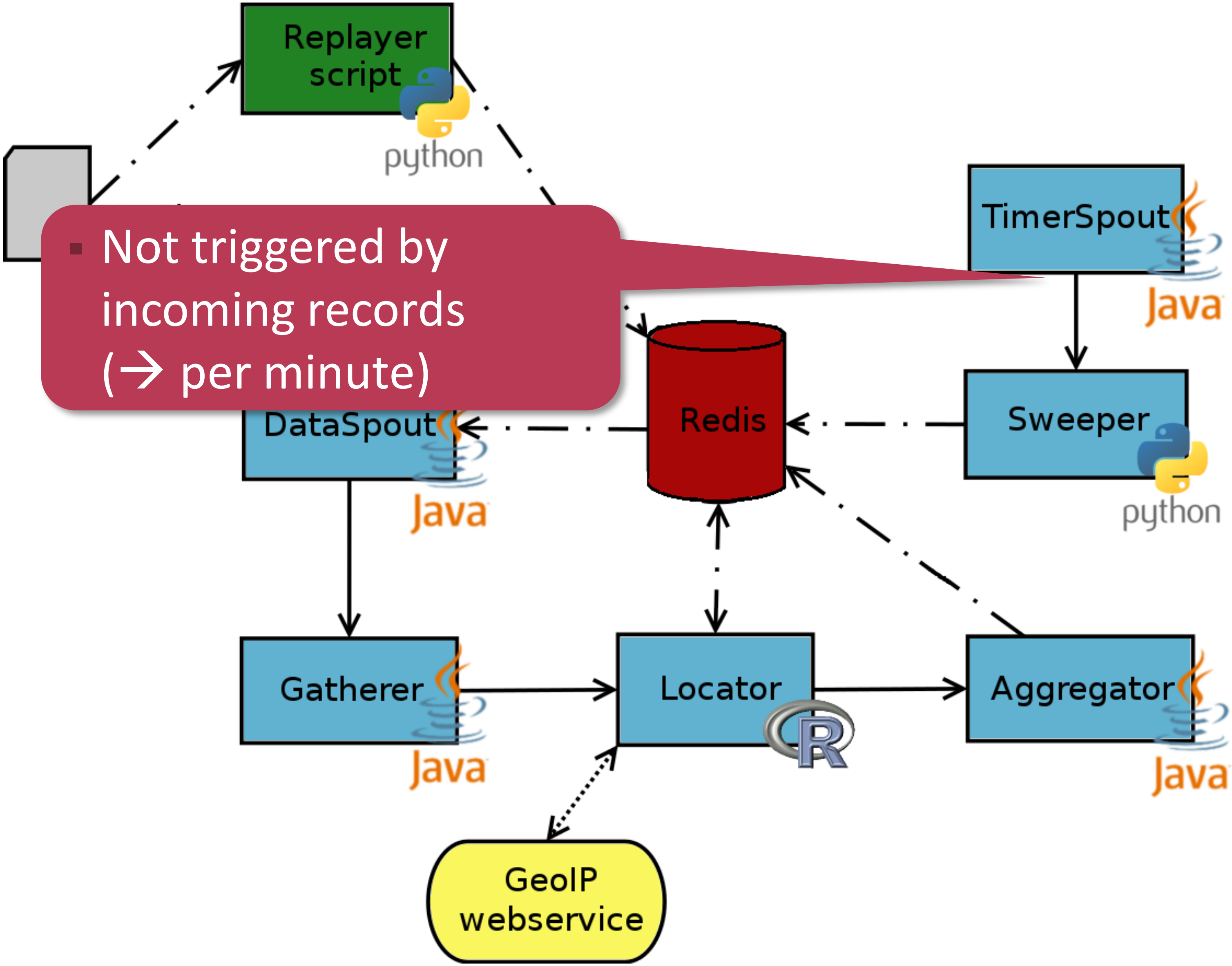



- Aggregates data to 3 minutes blocks
- Summarizes them by country
- Saves to database

# Application Data Flow

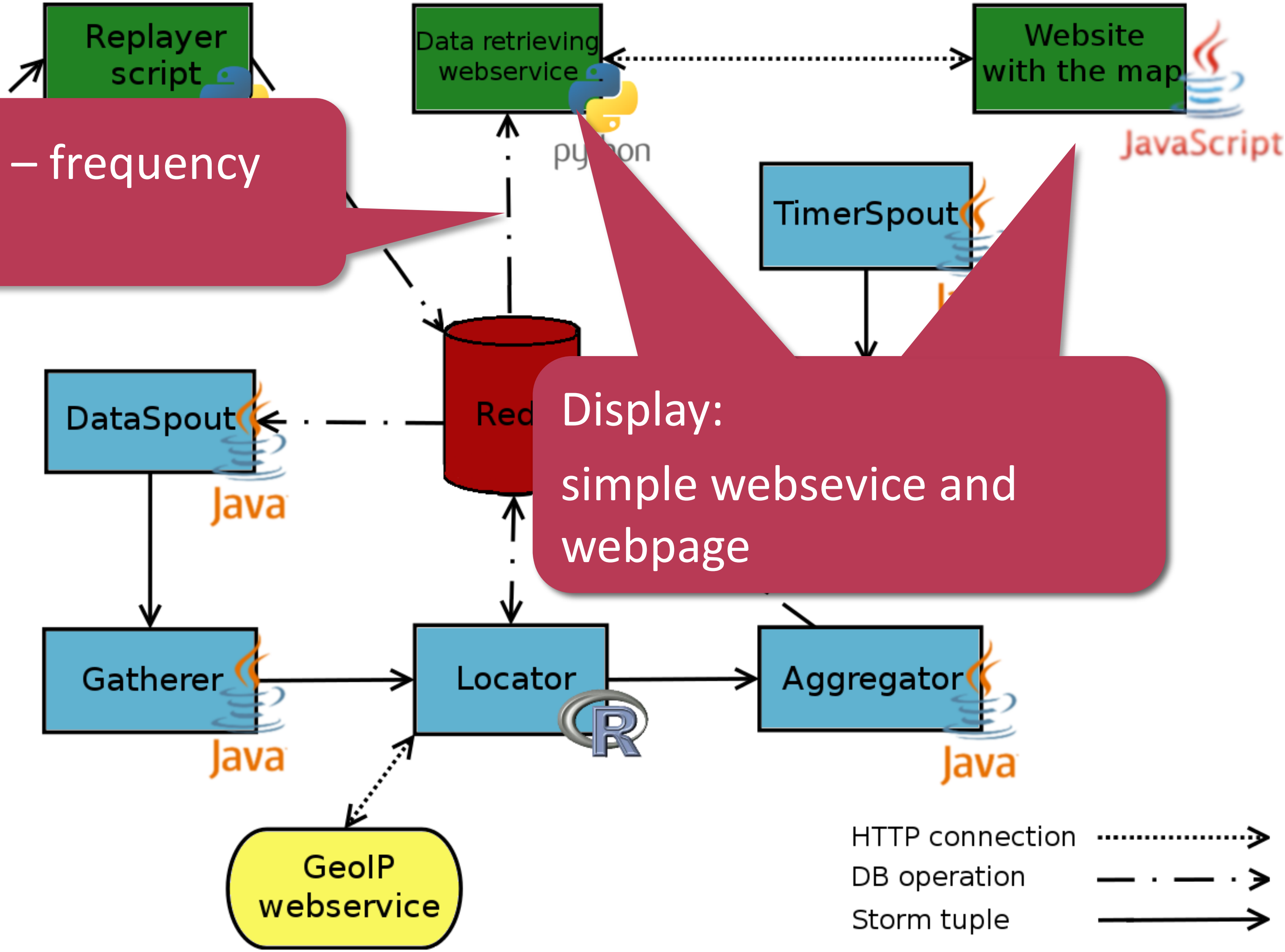


# Application Data Flow




- Not triggered by incoming records (→ per minute)

# Application Data Flow

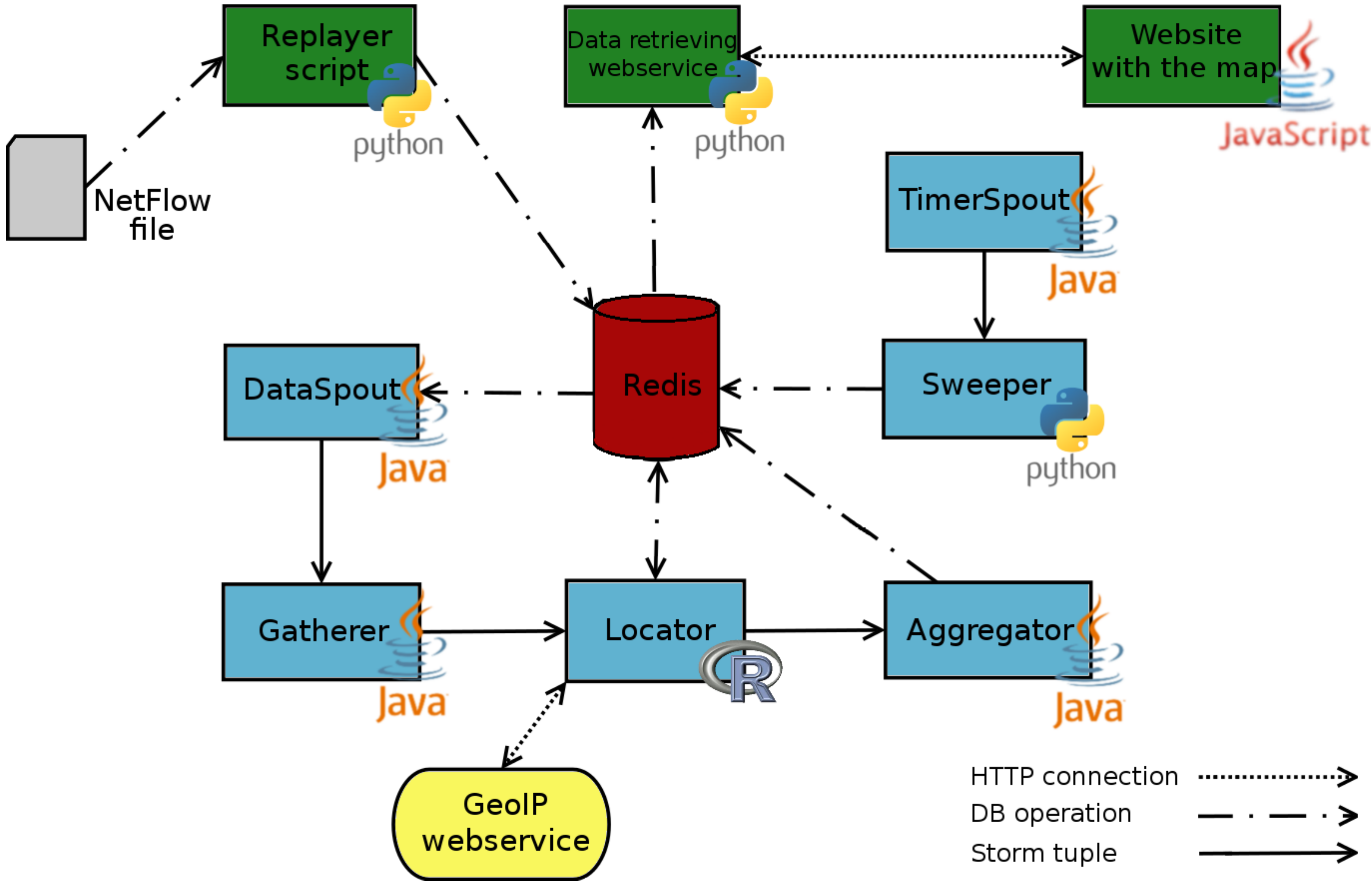



- Country – frequency pairs

Display:

simple websevice and webpage

# Application Data Flow

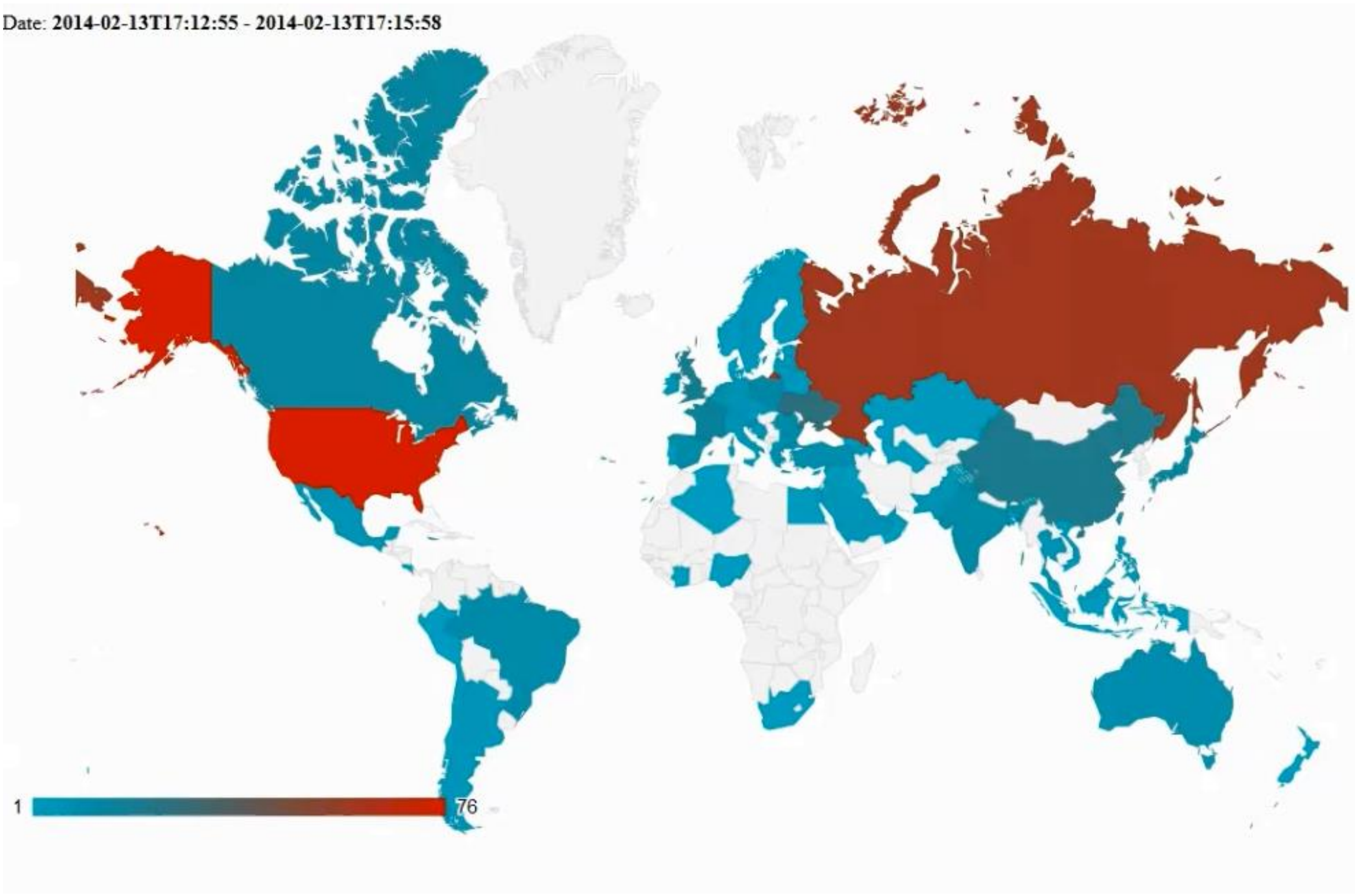


# Textual “Process” (Topology)

```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("redis_spout", new RedisSpout(), 1);
builder.setBolt("gatherer", new Gatherer(), 5)
              .shuffleGrouping("redis_spout");
builder.setBolt("locator", new GeoTagger(), 10)
                  .shuffleGrouping("gatherer");
builder.setBolt("aggregator", new Aggregator(), 10)
                  .fieldsGrouping("locator", new Fields("date"));
builder.setSpout("timer_spout", new TimerSpout(), 1);
builder.setBolt("sweeper", new Sweeper(), 5)
                  .shuffleGrouping("timer_spout");
```

# Output



# Why/how is it a process?

- Data flow appears explicitly
  - „Filter first, then summarize”
- Implicit dependencies (DB)
- Process template ~ topology
  - Own definition, not standard
- Not general
  - Specifically for data processing
  - (Originally: status updates)

# Data Flow Modelling

**Budapest University of Technology and Economics**
**Fault Tolerant Systems Research Group**

# 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”.

# Goal of Data Flow Modelling

- Nodes and communication
  - Identify system components and their interactions
- Nodes specified using a behaviour model…
  - State machine
  - (Process model?)
  - DFN
- Modelling hierarchy

# Communication of Components

- Loose coupling→ asynchronous composition
- Channel
  - FIFO or random access
  - Capacity (can be infinite)
  - Can be associated with a data model (eg. token set)
- Background technology
  - E.g. Message queue-based solutions

# Informal Definition – Data Flow Network

A **Data Flow Network** is a set of **nodes** which are connected and communicate over (unbounded) (FIFO) **queues**. Queues are called **channels**.

The bits of information that are communicated over the channels are called **tokens**.

- Nodes read their input channels is a blocking way.
- Nodes perform some computation on their input, and produce output. To start computing, nodes require „enough” tokens on their input channels. Nodes consume their input tokens.
- Nodes are stateless or stateful. Nodes fire one at a time.

# Data Flow Modelling

Non-deterministic DFN formalism

- [Jonsson, Cannata]

- Structure
  - Data flow graph (DFG)
    - nodes
    - directed arcs (FIFO channels)
- Behaviour
  - Firing rules: <s0; in=c0; s1; out=c2; $\pi$>
- Data
  - Tokens

# Data Flow Modelling

Non-deterministic DFN formalism

- [Jonsson, Cannata]

- Structure
  - Data flow graph (DFG)
    - no...
    - di... FI...
- Behaviour
  - **Firing rules: <s0; in=c0; s1; out=c2; π>**
- Data
  - Tok...

Initial state

Token taken from input channel

Output channel

Priority

Input channel

Target state

Token put to output channel

# Advantages of the method

| Property | Use case |
|---|---|
| Graphical, modular, compact, hierarchical | Clear model |
| Black and white box model | Early phase of modelling |
| Refinement rules | Multilevel modelling |
| Direct description of information flow | Modelling error propagation |
| Distributed model for both fine and coarse accuracy | Asynchronous, concurrent events |
| Data driven operation | Data driven real-time systems |
| Call transparency, atomic property, information hiding | Fault tolerant applications |
| Mathematical formalism | Formal methods |
| Transformation: TTPN, PA | Validation, temporal analysis |

# Formal Definition – Data Flow Network

**Data Flow Network** is a triple (N, C, S )

- N : set of nodes
- C : set of channels
  - I: input channels
  - O: output channels
  - IN: internal channels (between nodes)

  (I and O: Connection to the outside world)
- S : set of states

**Dataflow channel:**

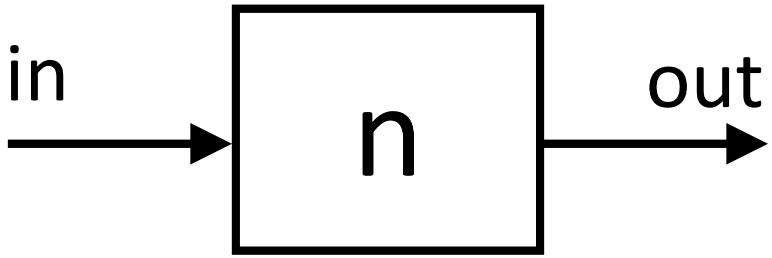
- FIFO channel with unlimited capacity
- Linked to one input and one output channel
- Channel state: $S_c = \times^{\infty} M_c$ token sequence

# Formal Definition – Data Flow Network

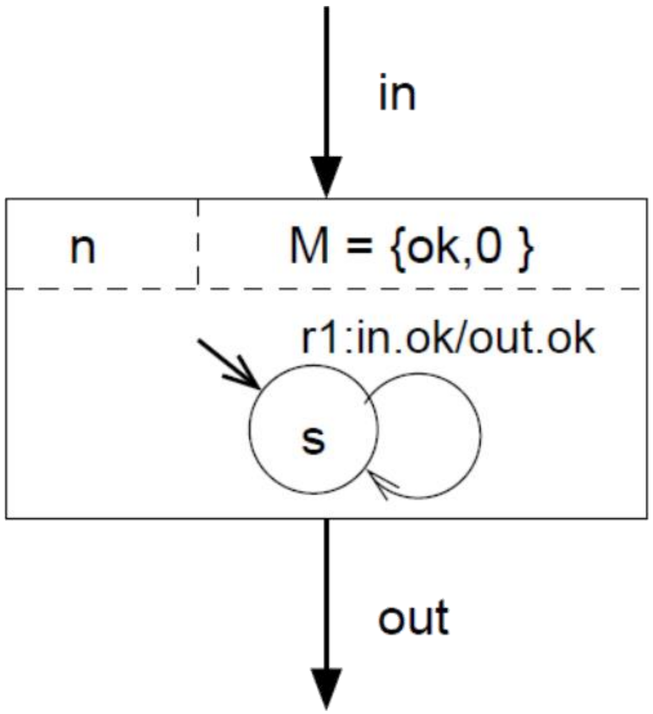
**Data flow node**: $n = (I_n, O_n, S_n, s_n^0, R_n, M_n)$, where

- $I_n$ – set of input channels
- $O_n$ – set of output channels
- $S_n$ – set of node states
- $s_n^0$ – initial state of node, $s_n^0 \in S_n$
- $M_n$ – set of tokens
- $R_n$ – set of firing rules, $r_n \in R_n$ a structure $(s_n, X_{in}, s'_n, X_{out}, \pi)$
  - $s_n$ – states before and after firing, $s'_n \in S$
  - $X_{in}$ – input mapping, $X_{in} : I_n \to M_n$
  - $X_{out}$ – output mapping, $X_{out} : O_n \to M_n$
  - $\pi$ – priority, $\pi \in N$

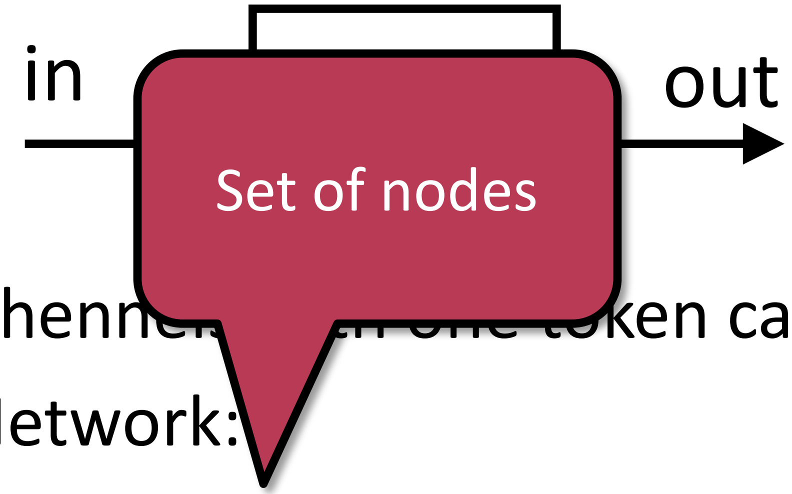
# Example



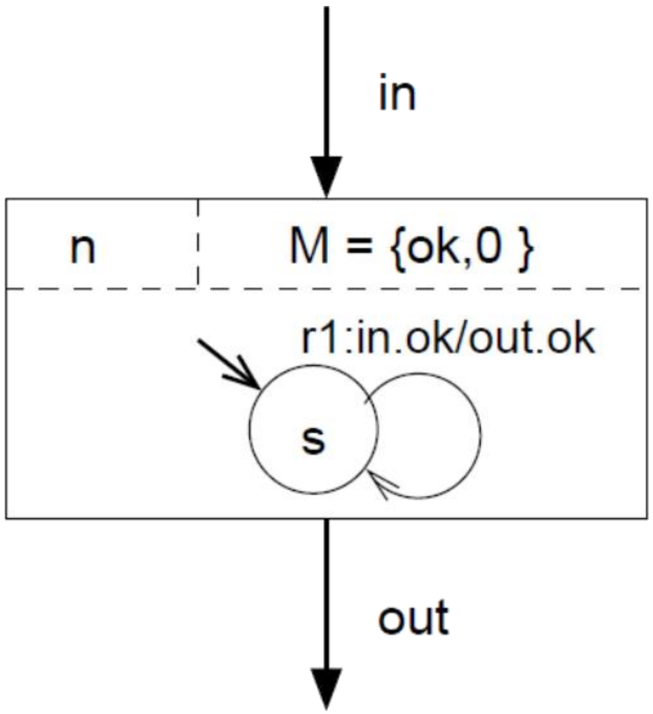

- Channels with one token capacity
- Network:
  - DFN = ({n}, {in, out}, {(s,0,0), (s,ok,0), (s,0,ok), (s,ok,ok)})
- Nodes:
  - n = ({in}, {out}, {s}, s, {ok,0}, {r1})
- Firing rules:
  - r1=<s; in=ok; s; out=ok; 0>



# Example




- Chenn… ken capacity
- Network:
  - DFN = ({n}, {in, out}, {(s,0,0), (s,ok,0), (s,0,ok), (s,ok,ok)})
- Nodes:
  - n = ({in}, {out}, {s}, s, {ok,0}, {r1})
- Firing rules:
  - r1=<s; in=ok; s; out=ok; 0>

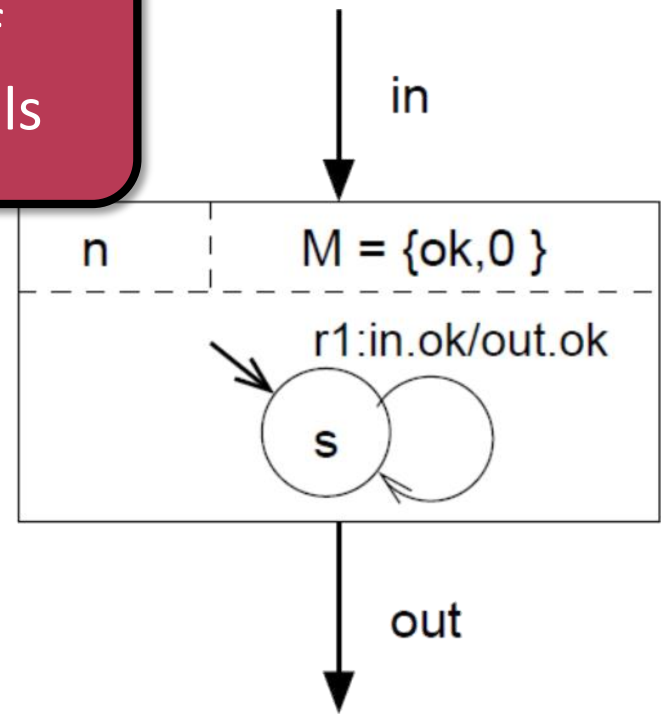


# Example


Set of nodes

Set of channels

- Chenn… …ken
- Network:
  - DFN = ({n}, {in, out}, {(s,0,0), (s,ok,0), (s,0,ok), (s,ok,ok)})
- Nodes:
  - n = ({in}, {out}, {s}, s, {ok,0}, {r1})
- Firing rules:
  - r1=<s; in=ok; s; out=ok; 0>



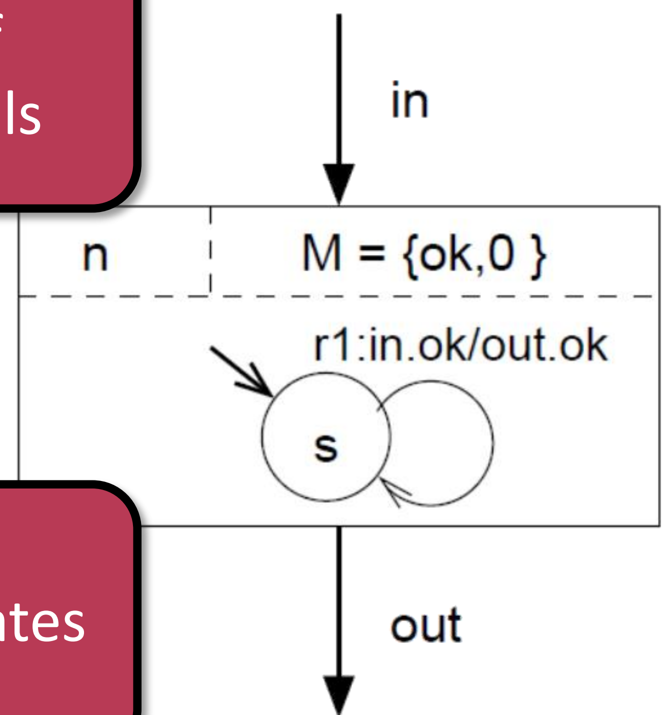
# Example


Set of nodes

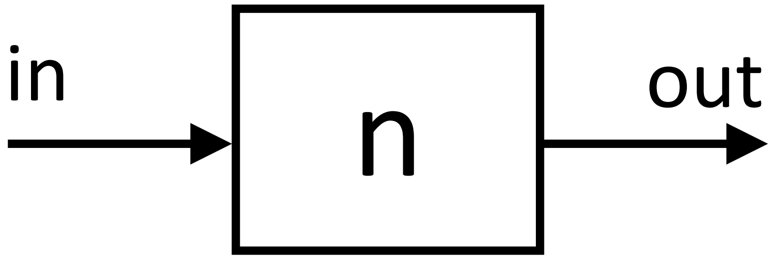
Set of channels

- Chenn… one token
- Network:
  - DFN = ({n}, {in, out}, {(s,0,0), (s,ok,0), (s,0,ok), (s,ok,ok)})
- Nodes:
  - n = ({in}, {out}, {s}, s, {ok,0},
- Firing rules:
  - r1=<s; in=ok; s; out=ok; 0>

Set of states



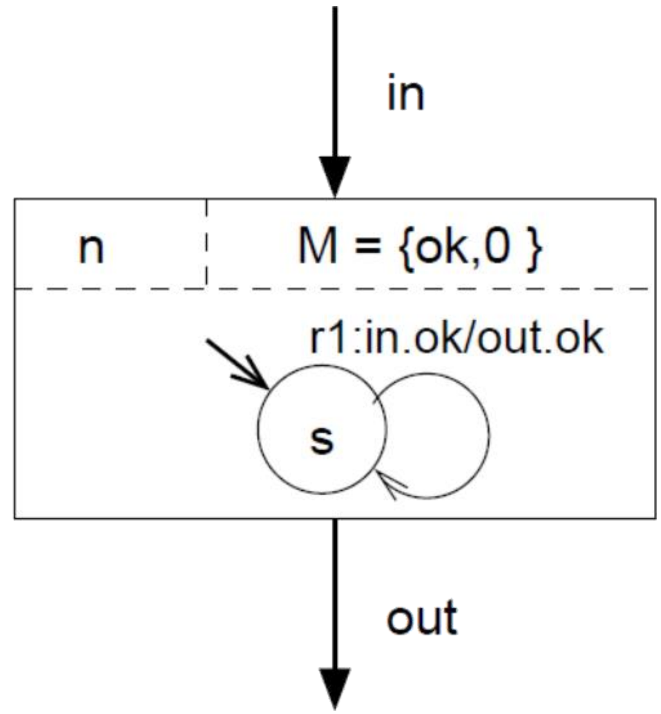
# Example



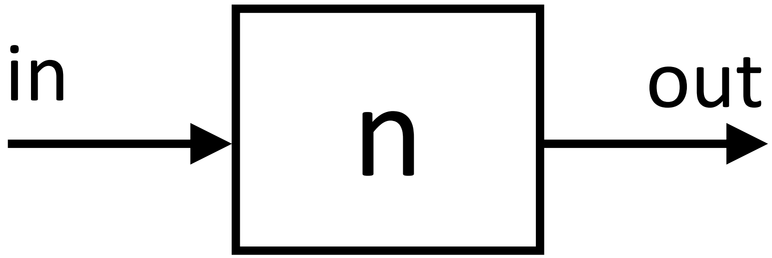

- one token capacity

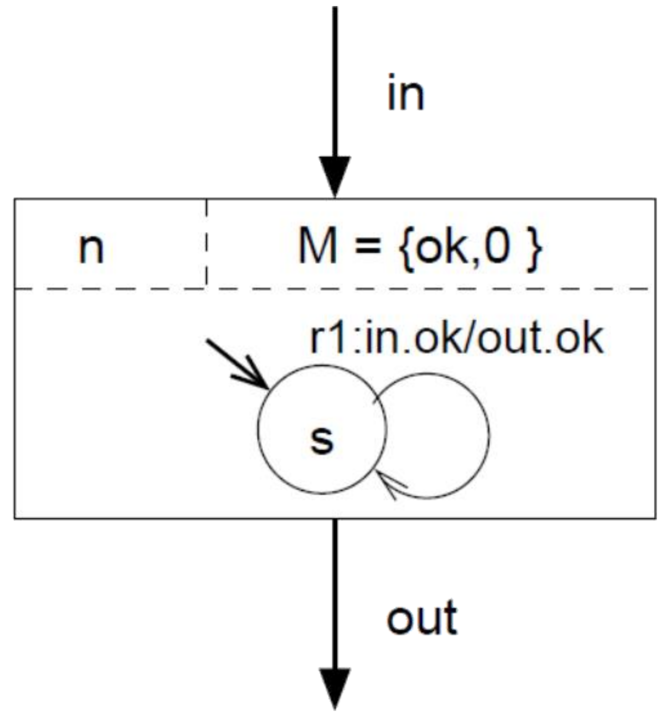
Set of input channels

- , out}, {(s,0,0), (s,ok,0), (s,0,o s,ok,ok)})
- Nodes:
  - n = ({in}, {out}, {s}, s, {ok,0}, {r1})
- Firing rules:
  - r1=<s; in=ok; s; out=ok; 0>



# Example



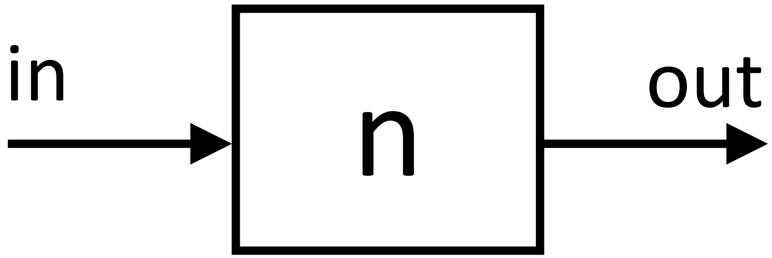




Set of input channels

Set of output channels

- o … y
- 
  …, ou … 0),
  (s,0,o … s,ok,ok)
- Nodes:
  - n = ({in}, {out}, {s}, s, {ok,0}, {r1})
- Firing rules:
  - r1=<s; in=ok; s; out=ok; 0>

# Example




Set of input channels

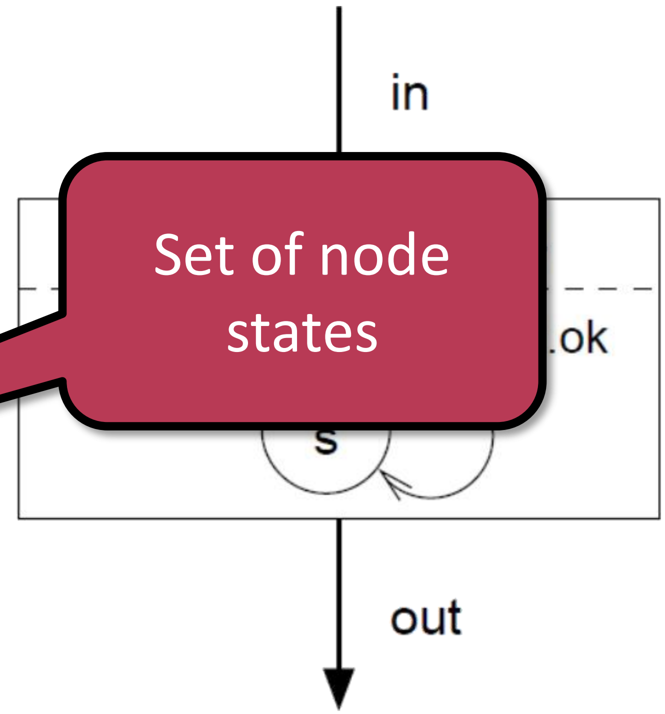
Set of output channels

Set of node states

o … y

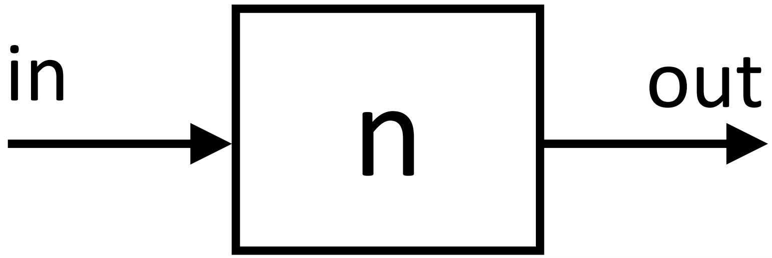
n, o … 0),

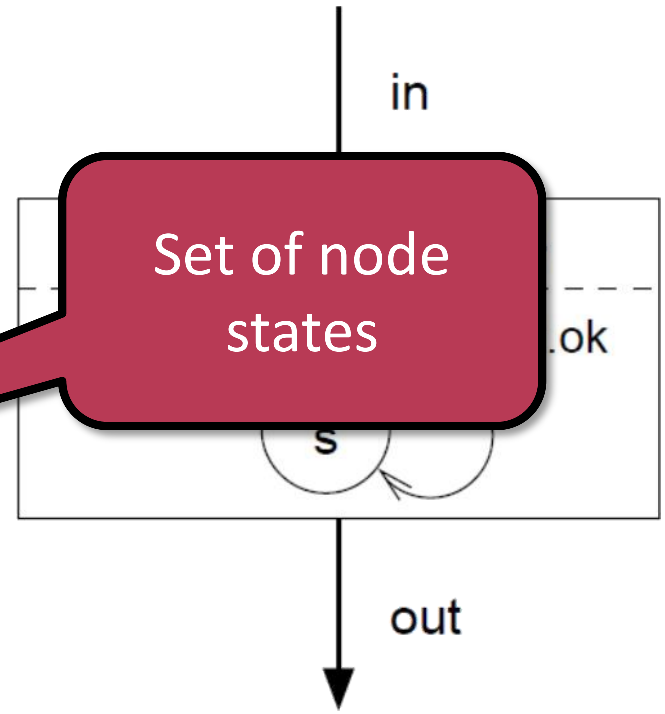
(s,0,o … s,ok,ok)




- Nodes:
  - n = ({in}, {out}, {s}, s, {ok,0}, {r1})
- Firing rules:
  - r1=<s; in=ok; s; out=ok; 0>

# Example




Set of input channels

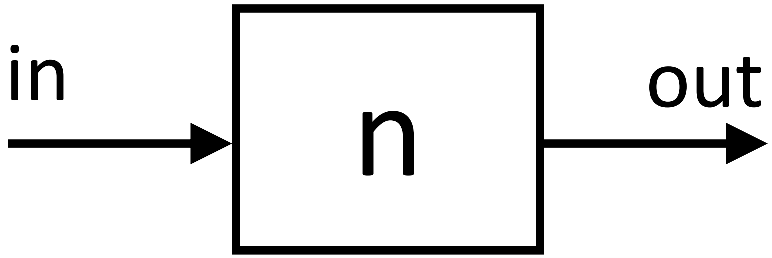
Set of output channels

Set of node states

- o … y
- …
  - …n, ou… 0),
  (s,0,o… s,ok,ok)…
- Nodes:
  - n = ({in}, {out}, {s}, s, {ok,0}, {r1})
- …
  - …; out=ok; 0>

Set of tokens

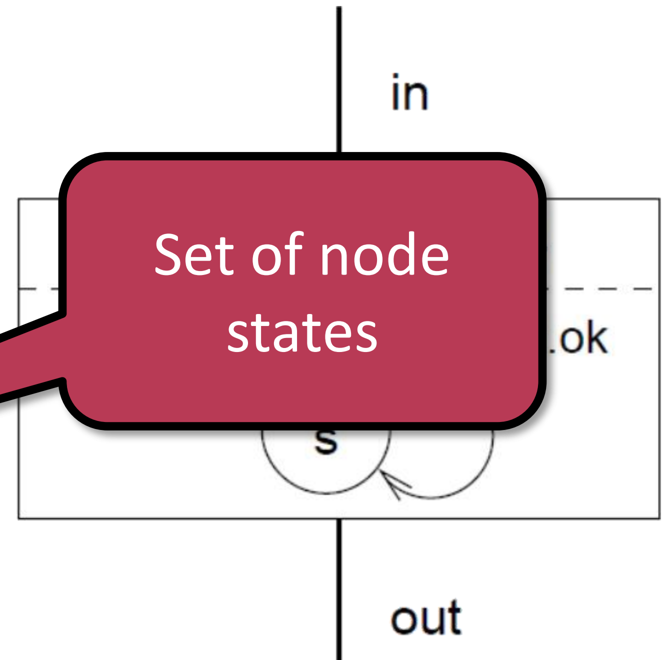
# Example




- Set of input channels
- Set of output channels

(fragments partly hidden: "o ... y"; "n, o ... 0),")

(s,0,o ... s,ok,ok)}




Set of node states

- Nodes:
  - n = ({in}, {out}, {s}, s, {ok,0}, {r1})

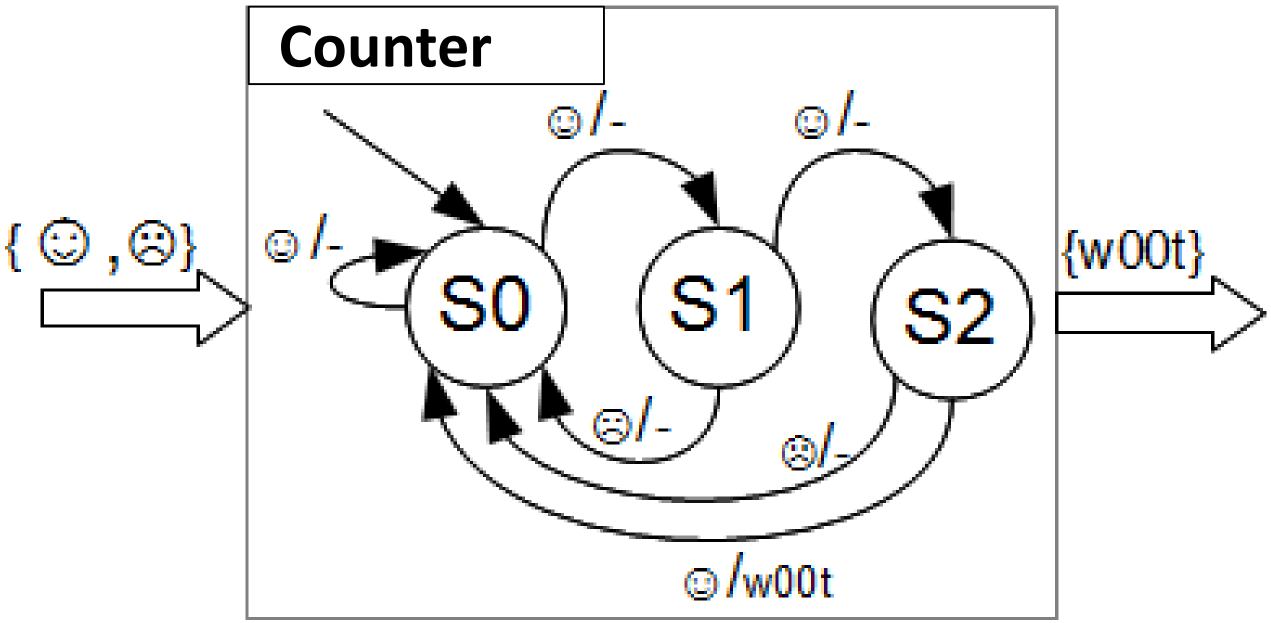
Set of tokens

Set of firing rules

; out=ok; 0>

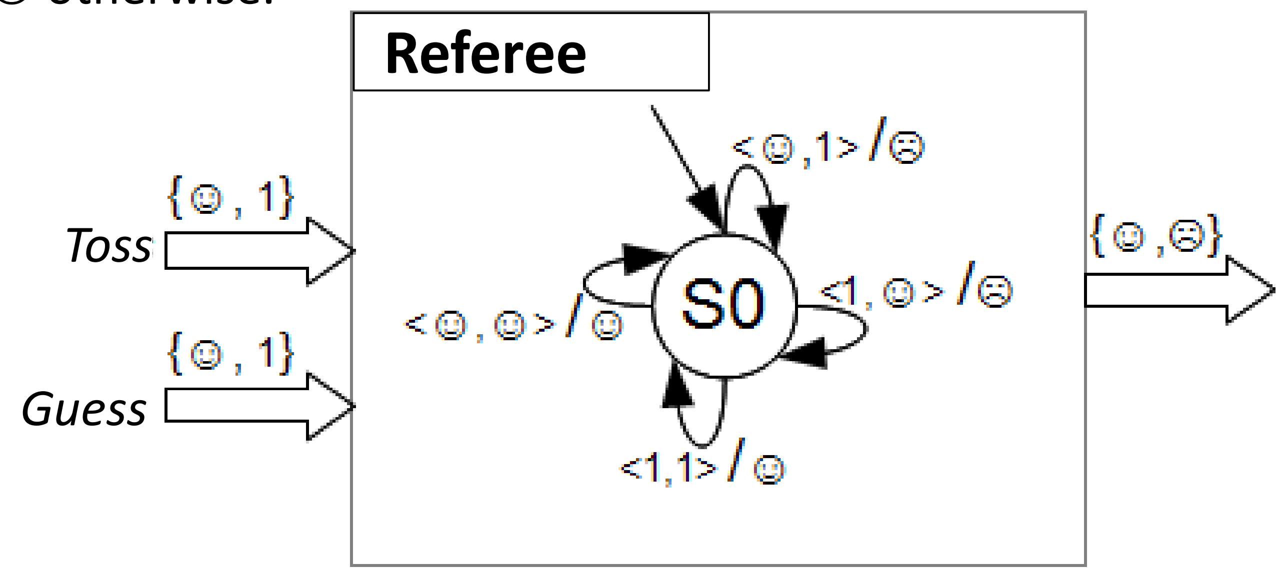
# Example - Counter

- Design the „Counter” node of a DFN
  - Input: ☺ and ☹ input tokens
  - Output: *w00t* token when reading 3 ☺s in a row.



# Example - Referee

- Design the „Referee” node of a DFN.
  - Input 1: result of a coin toss
  - Input 2: the player’s guess
  - Output:
    - ☺ if the toss and the guess match,
    - ☹ otherwise.



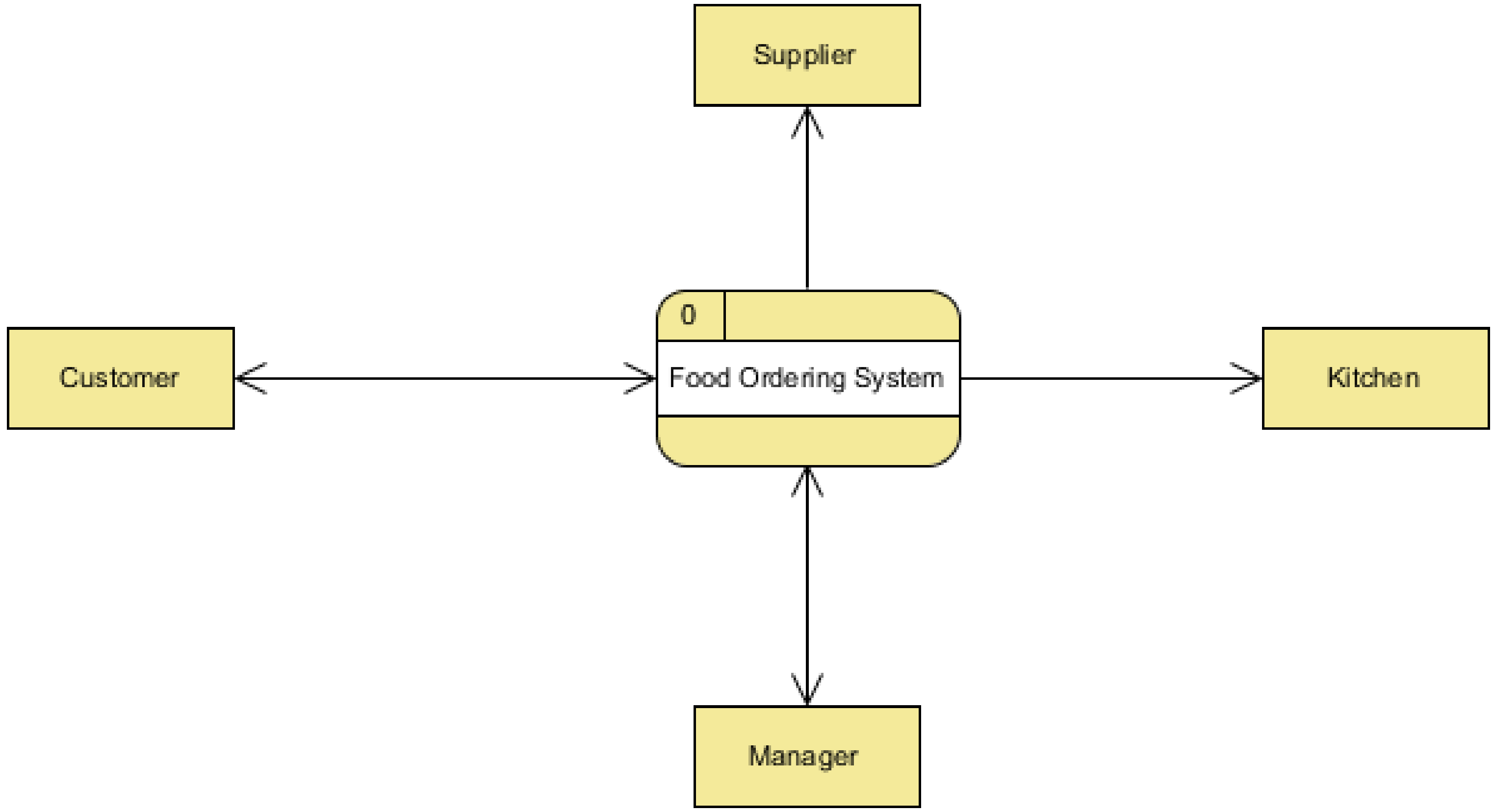

{Toss.1, Guess.☺} := <1, ☺>

# Execution Model

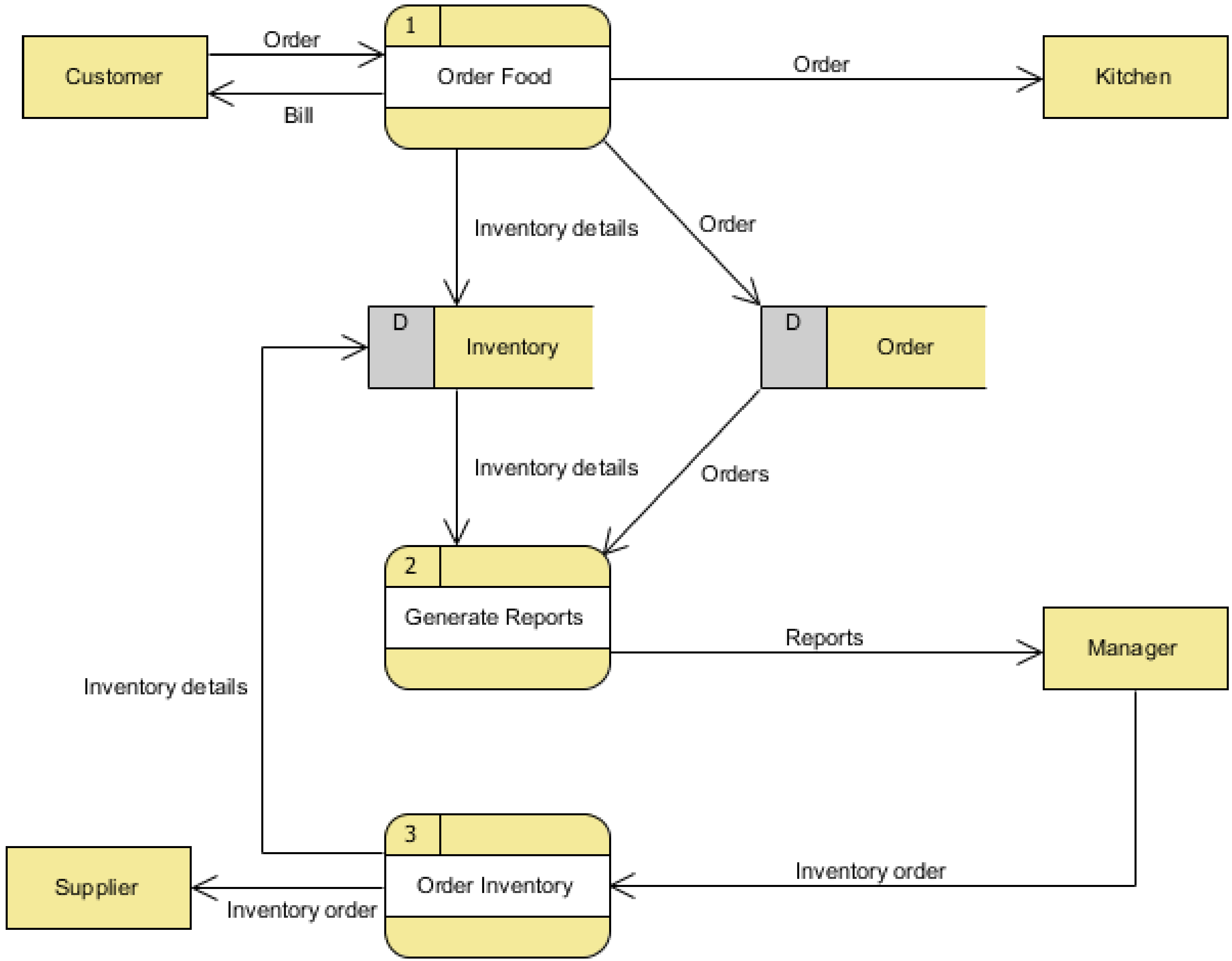
- How the model will be executed?
  - State based models
    - event handling
  - Process based models
    - keeping track of the current state of the instances
  - Data flow based models
    - independent/concurrent execution of all nodes
    - nodes only care for their own input/output channels

- Applications:
  - E.g. data processing, form processing, LabView, …

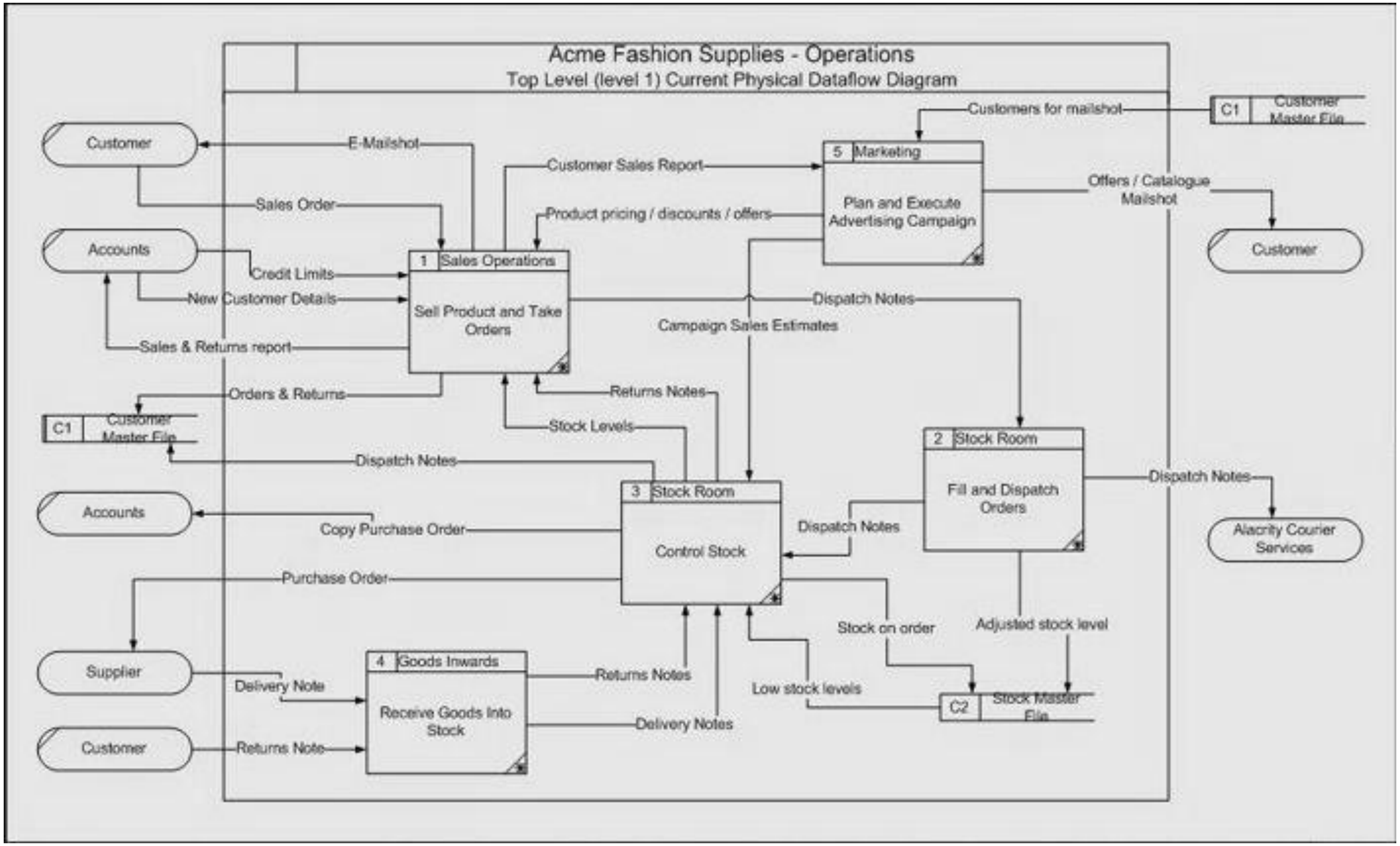
# Example – Food Ordering System



# Example – Food Ordering System



# Example – Fashion Supplies



# Example – Warning Triangle Manufacturing

1. Two machines
   - One produces light-resistant side panels, and places them on the conveyor belt.
   - The other one takes the panels off the belt, and besides produces an assembled triangle every once in a while.
2. First machine sometimes produces deformed side panels.
3. The assembler machine contains a testing equipment wired before the original functionality, that is able to get rid of the deformed panels.
4. After discarding deformed panels the assembler machine always waits until three light-resistant panels have arrived and assembles a triangle out of them.

# Example – Error Propagation Analysis

- Modelling the system as a data flow network
- The nodes are the system components
  - their behaviour is modelled in a state based way
  - states: correct operation, different erroneous operation modi
  - state transitions: corruption, repair
  - error handling features can be modelled
    - error detection, error correction, error confinement
- The channels are the communication channels where errors can propagate
- The tokens are messages: correct or erroneous ones
  - the content of the messages is not considered

- The big question:

  What kind of errors can be propagated to the output?