UML Statecharts

Emmanuel Fleury
B1-201
fleury@cs.aau.dk

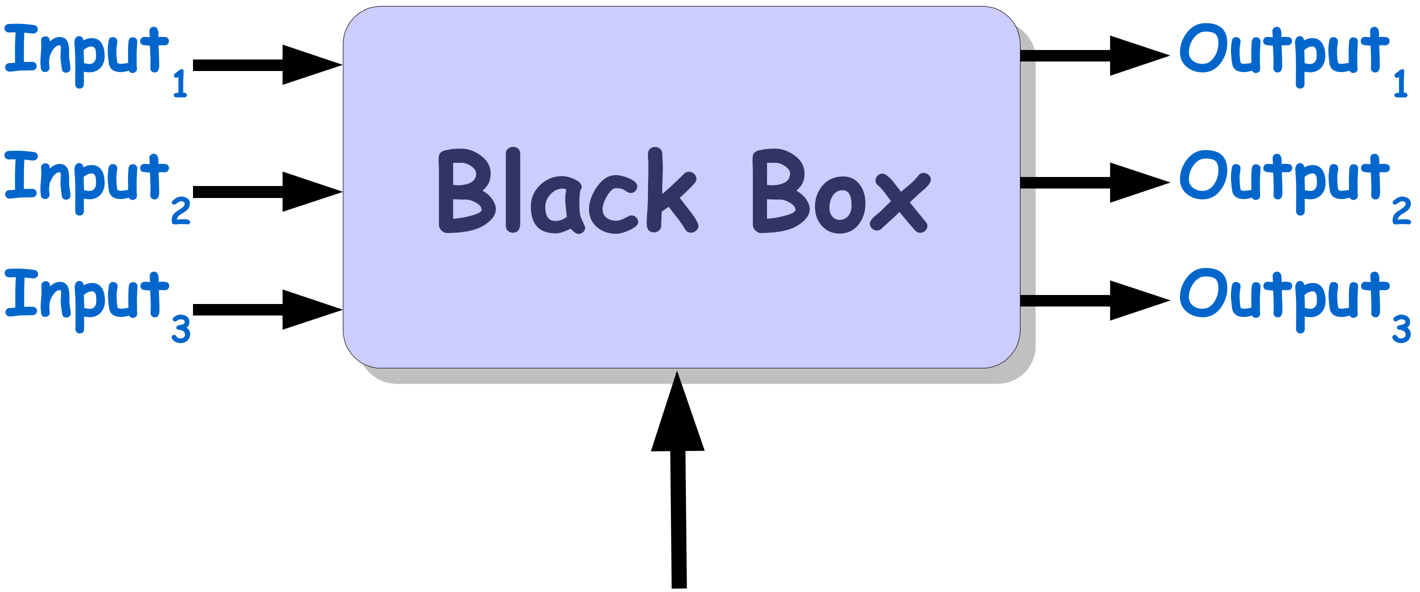
Outline

- Motivations
- State-Diagrams
- Finite State Machines
- Statecharts
- UML Statecharts
- Modelling Tools

Motivations

Motivations

A computer system can be seen as a reactive system



System to be modelled

Our Goal ?

Represents
Computer Systems
in an "efficient" way !

The Problem

Computer systems are more and more complex and this complexity prevent:

- Global Understanding of the system
- Global Analysis of the system
- Flexibility/Adaptability of the system
- Modularity of the system
- Maintainability of the system

Success Criteria

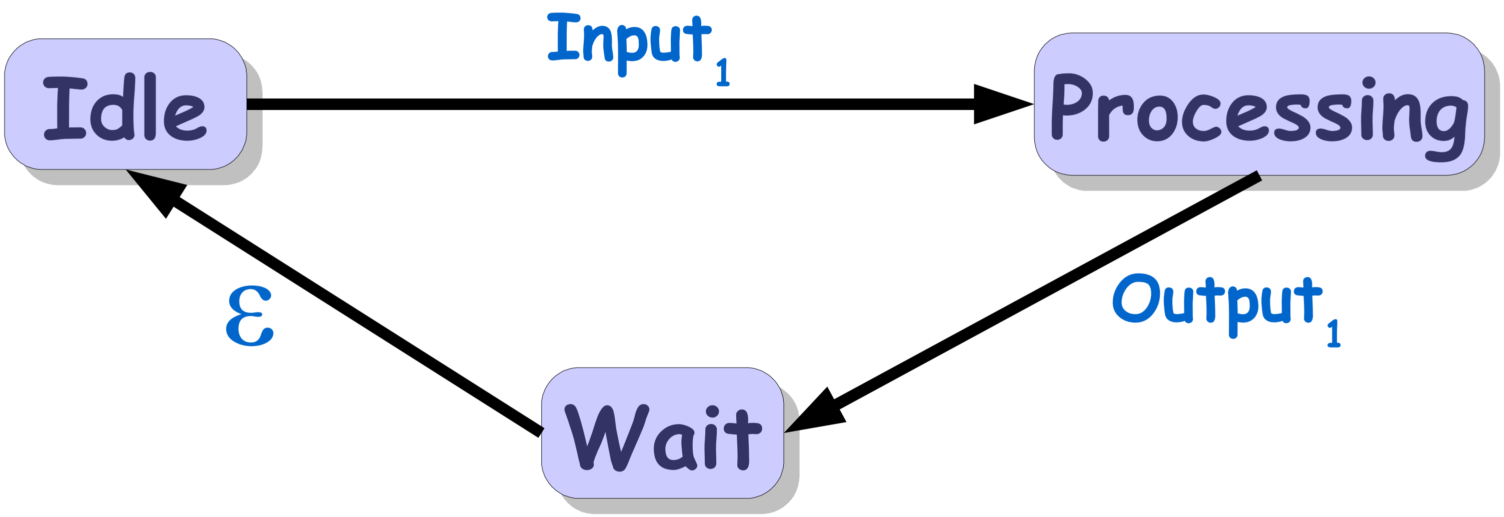
Good Properties to get:

- Human Readable (a visual formalism)
- Concise
- Formal
- Abstract
- Just as expressive as needed
  (important properties can be checked)

State-Diagrams

Why States ?

Computer systems are naturally organized in "states". Where each state is an abstraction for a position in the process of achieving the task of the system.



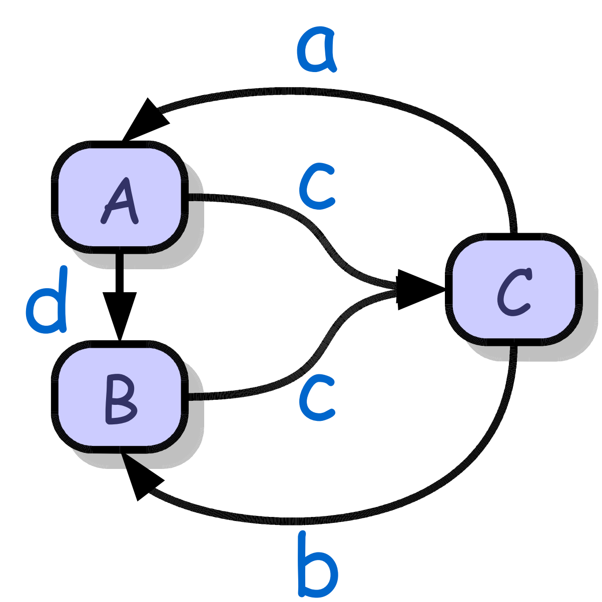
State-Diagrams

- Labelled Directed Graphs
- Hypergraphs
- Hierarchical Graphs
- Higraphs

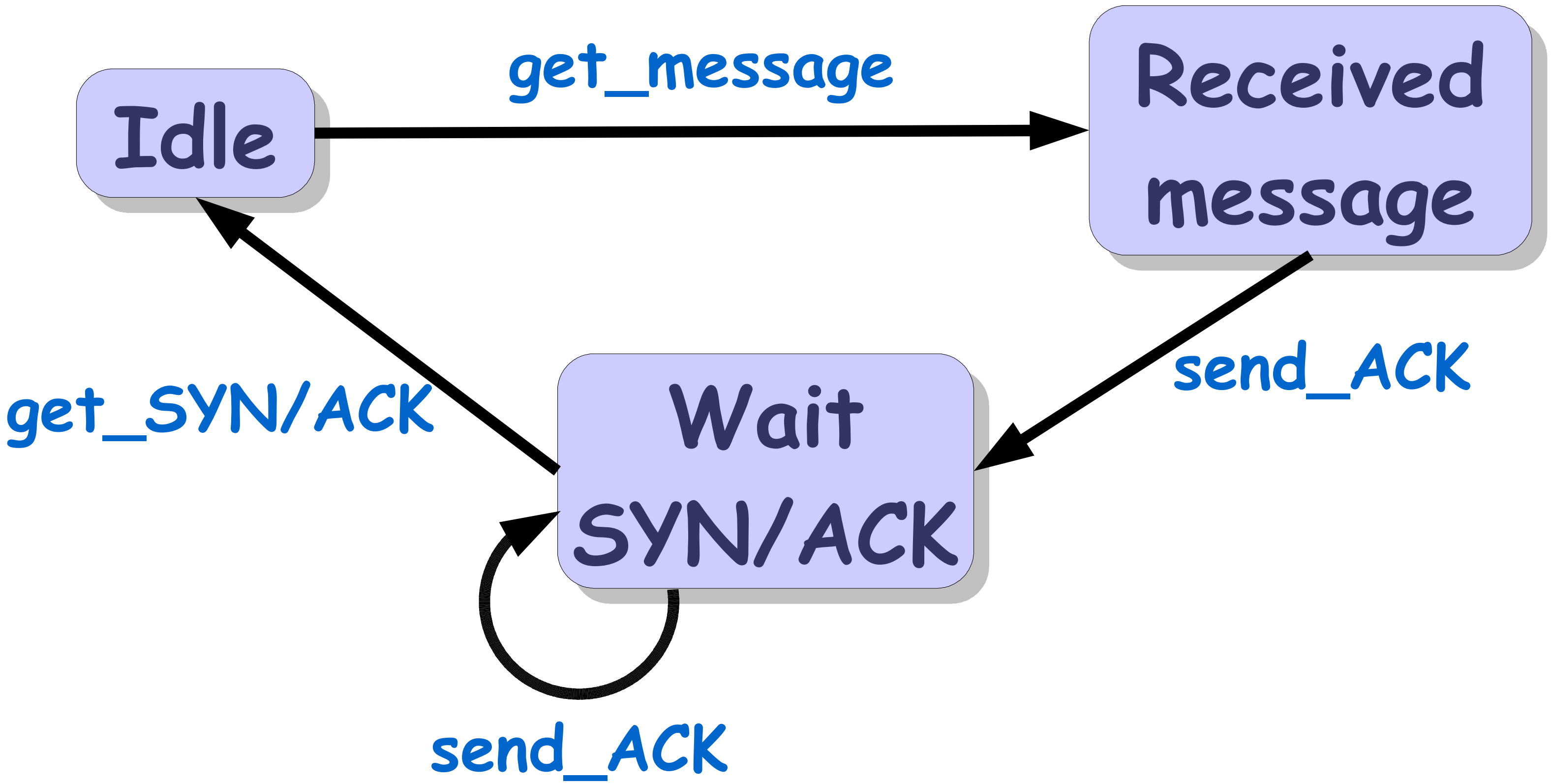
Labelled Directed Graphs

A Labelled Directed Graph is a triplet (Q, A, T):

- **Q** a finite set of states;
- **A** a (possibly infinite) alphabet;
- T⊆QxAxQ a finite set of transitions.



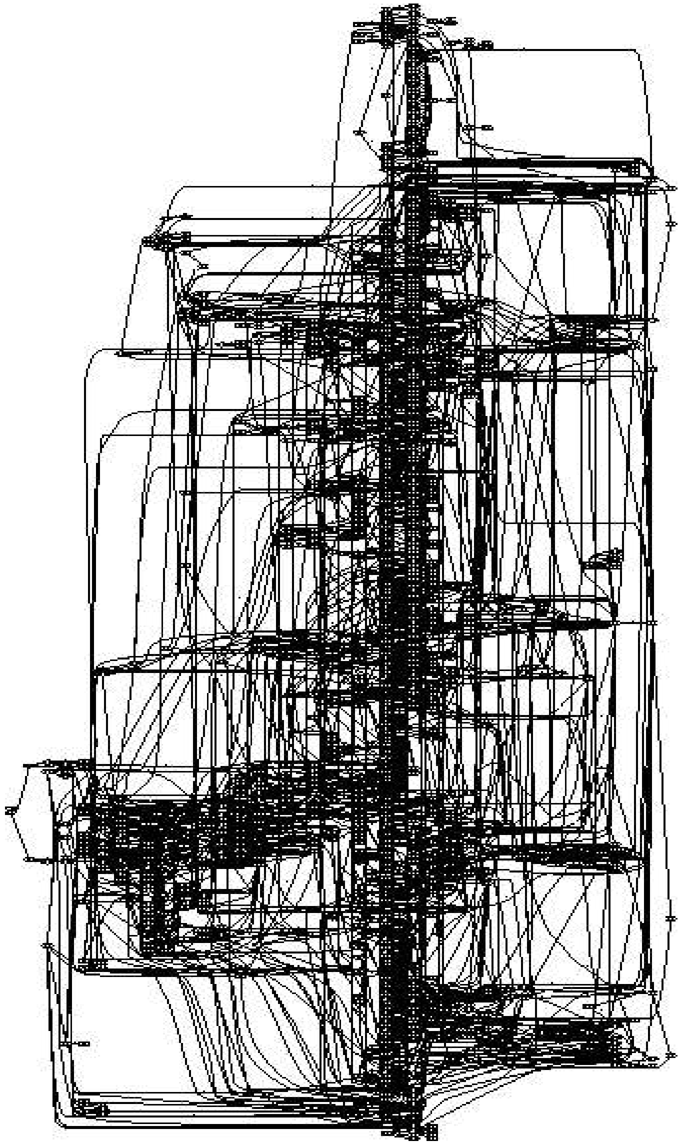
Example


get_message
Idle
Received message
send_ACK
get_SYN/ACK
Wait SYN/ACK
send_ACK

Advantages/Drawbacks

- Good Points:
  - Human readable
  - Formal
- Bad Points:
  - Not concise enough !!!

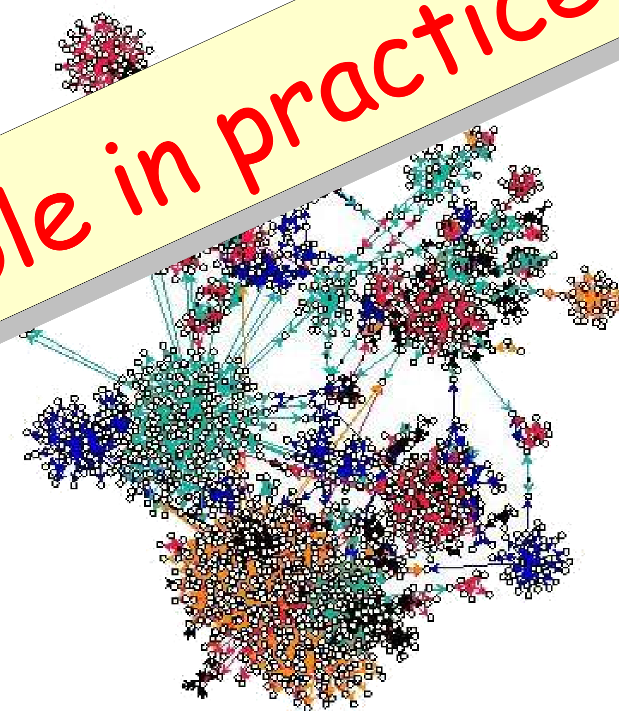
We need more abstraction !!!



Hypergraphs

An Hypergraph is a triplet (Q, A, H):

- **Q** a finite set of states;
- ***A*** a (possibly infinite) alphabet;
- **H⊆QxAxP(** … e set of h… ges.

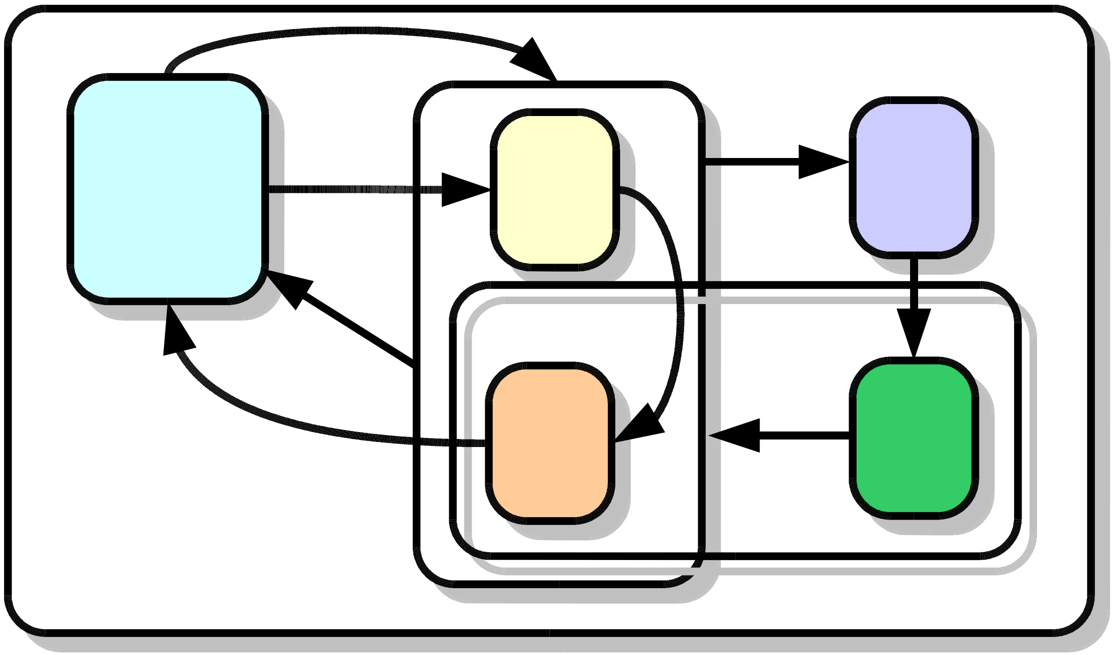


Nice in theory but unusable in practice !

Try to factorize the number of edges and generalize the notion of graphs !

Hierarchical Graphs

Notion of **Hierarchy** among the states



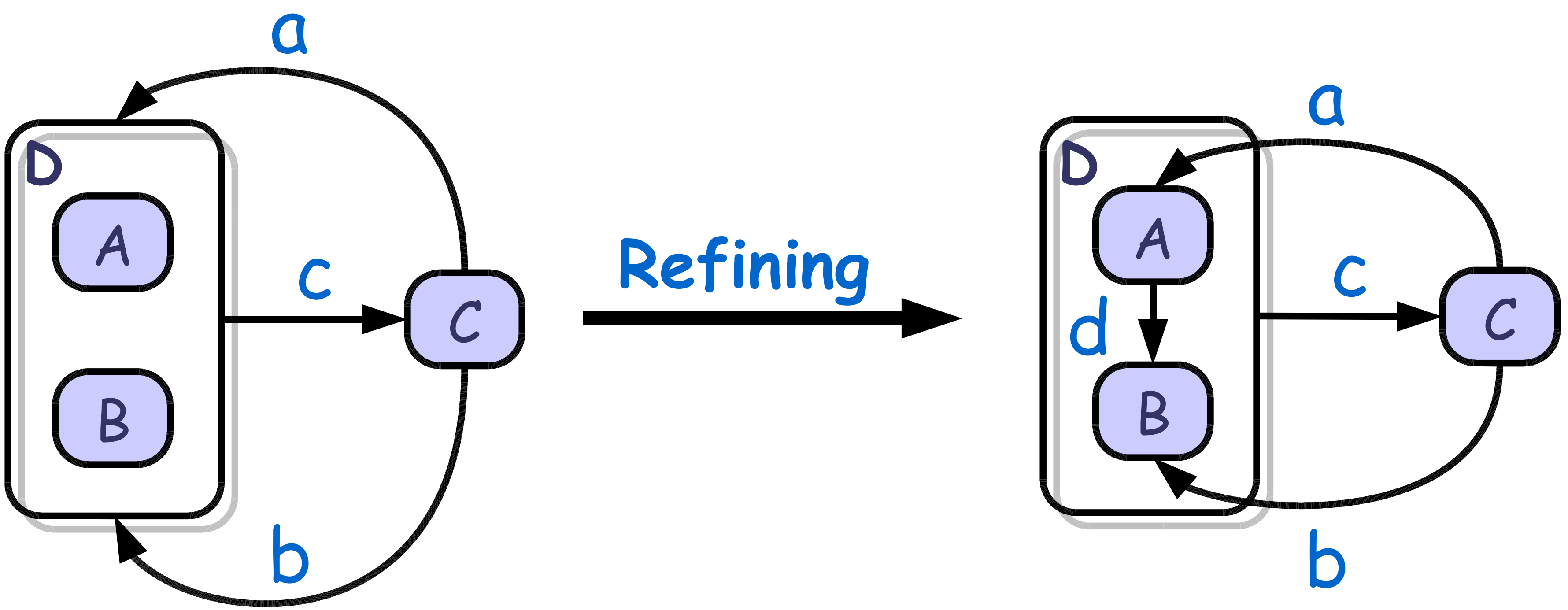
Allows Clustering, Refining and Zooming !

Clustering

Using states hierarchy to **factorize** behaviours

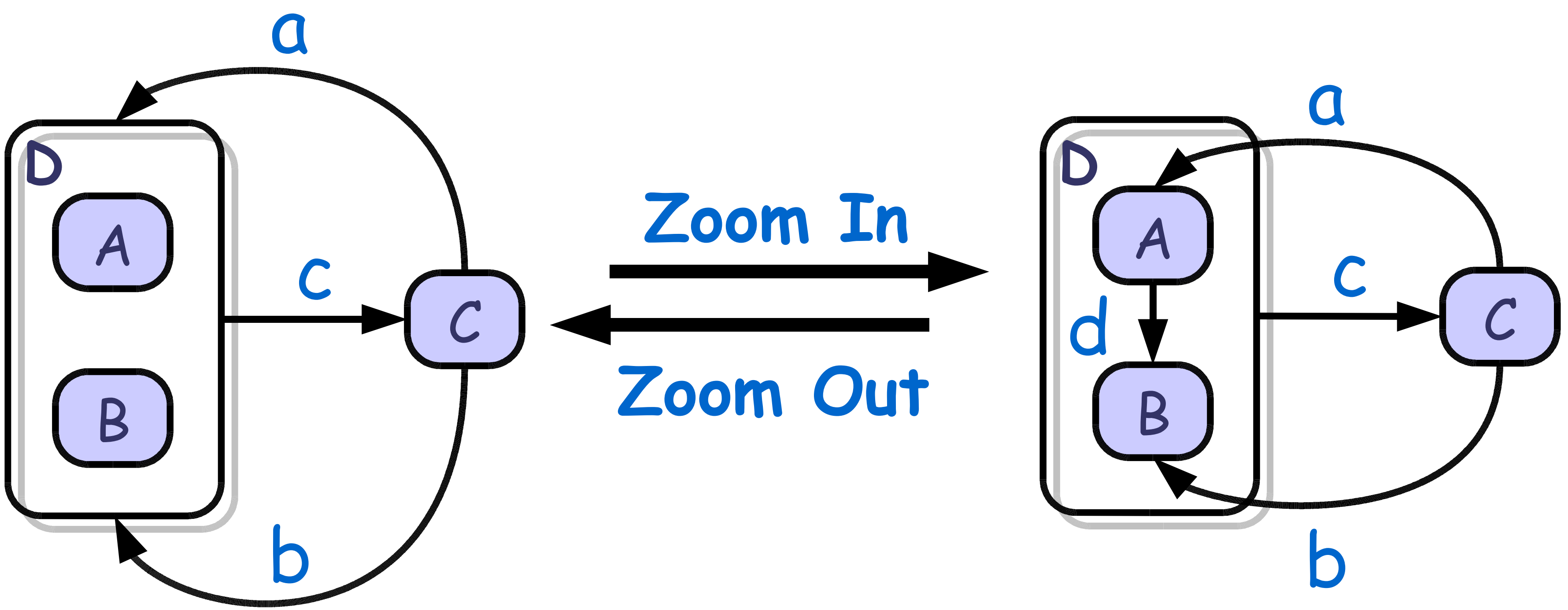
Refining

Using state hierarchy to implement "a posteriori" the internal behaviour of a state

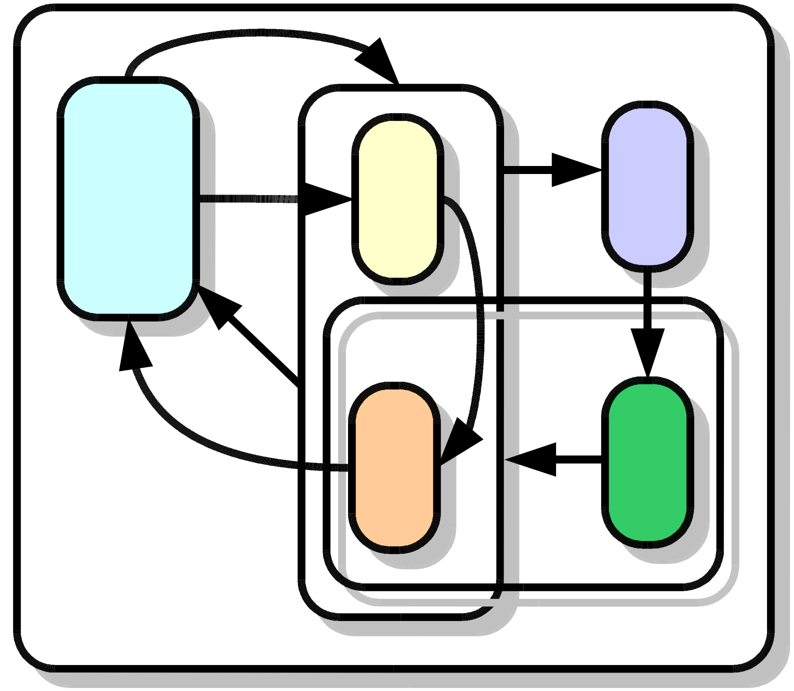


Zooming In/Out

Using state hierarchy to **hide/unveil** implementation's details



Formal Definition

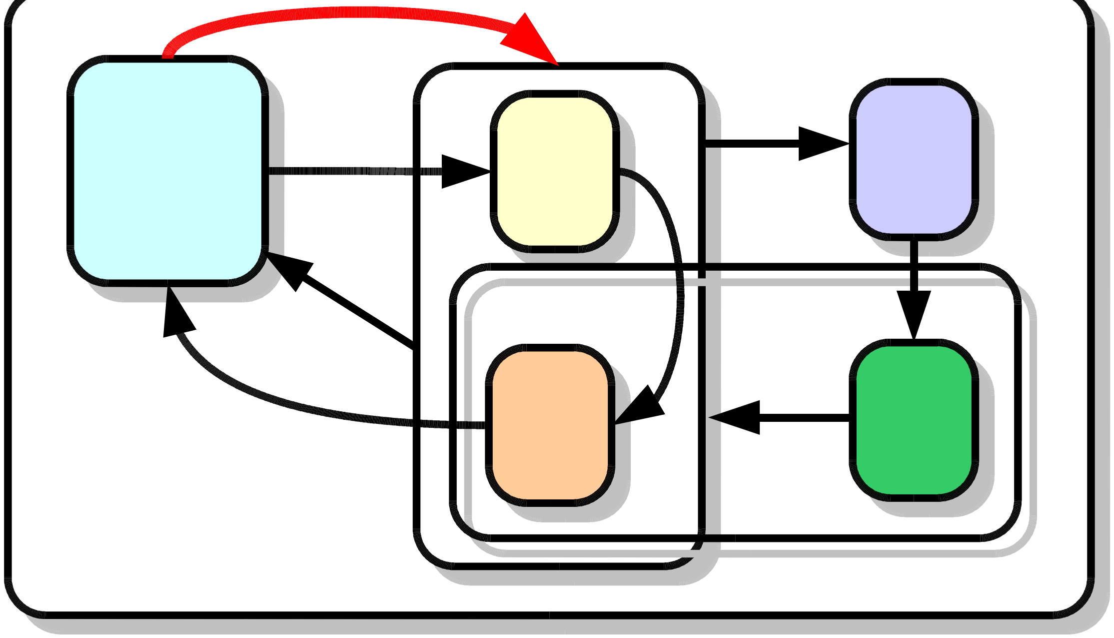


A Hierarchical Graph is a triplet (Q, E, >):

- **Q** a finite set of states;
- **E⊆QxQ** a finite set of edges;
- **>⊆QxQ** the hierarchical relations in between states.

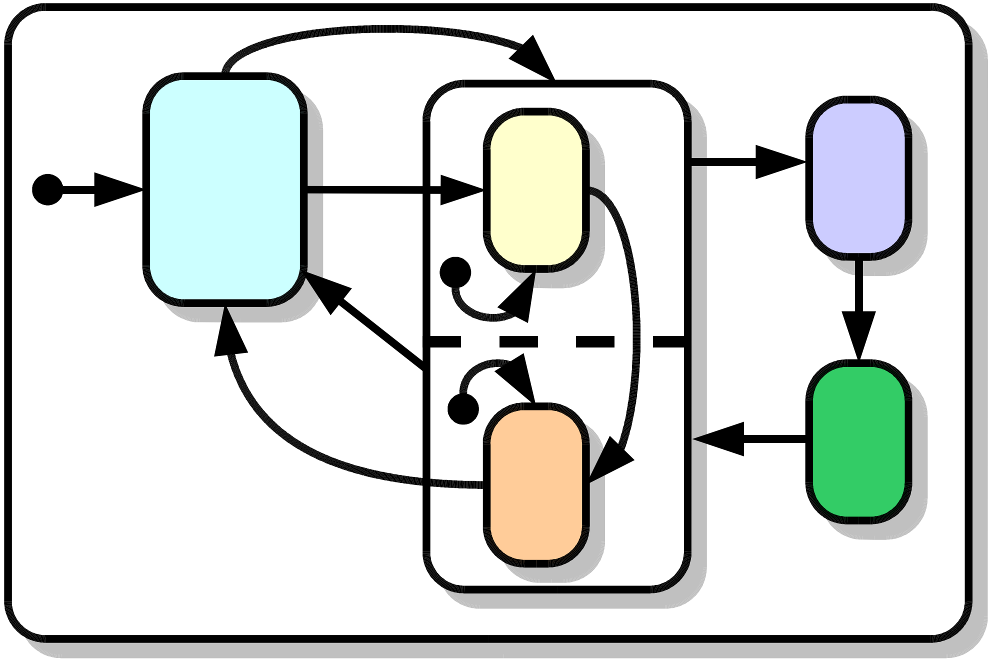
But...

A lot of semantics problems !!!



Higraphs

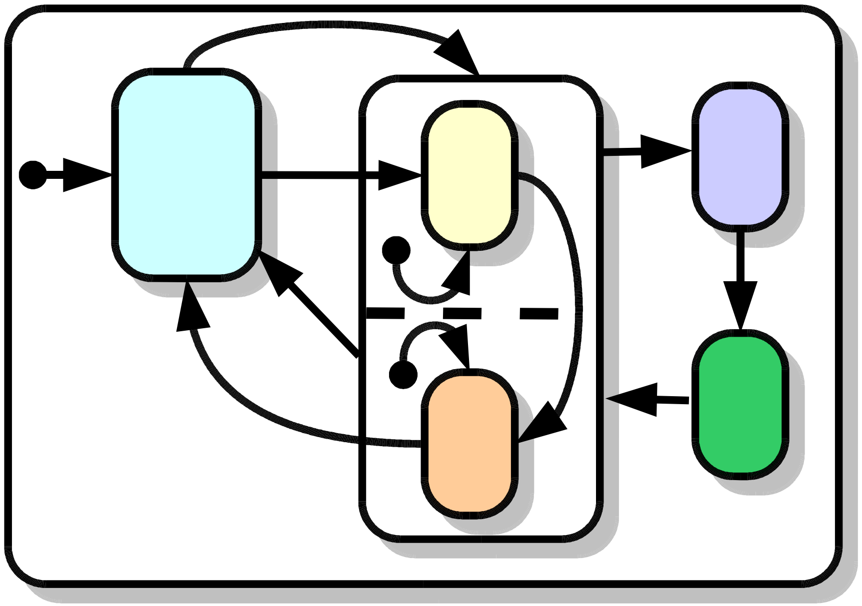
higraphs = graphs + depth + orthogonality



Introduced by David Harel in 1987
for Avionic Systems Modelling.

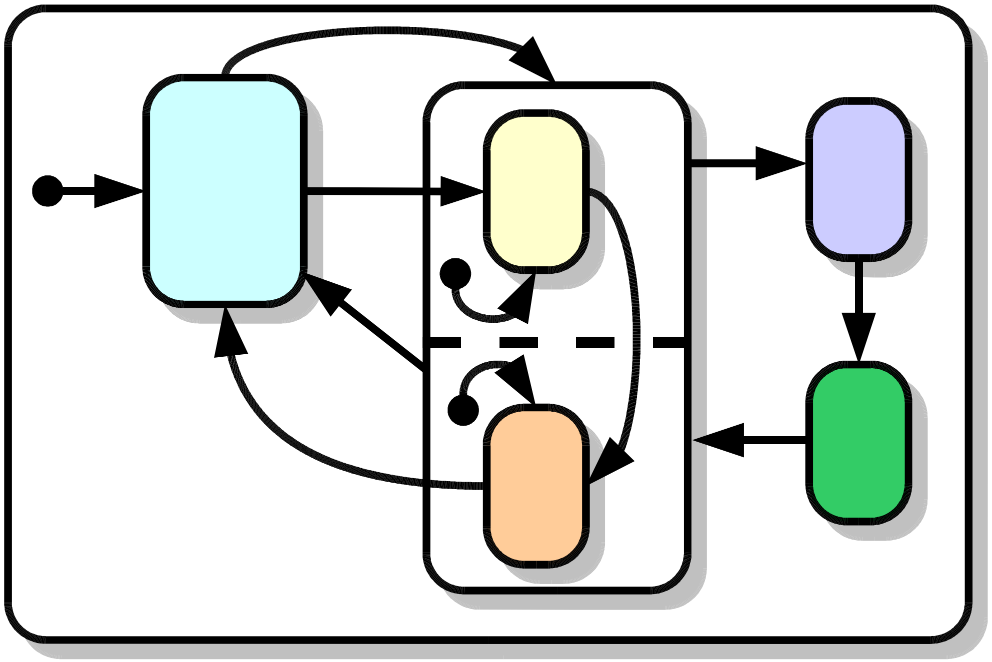
Superstates

- A state which contain state(s)
- All of them have a **default entry**
- There are two types of superstates:
  - **AND-States**
  - **OR-States**



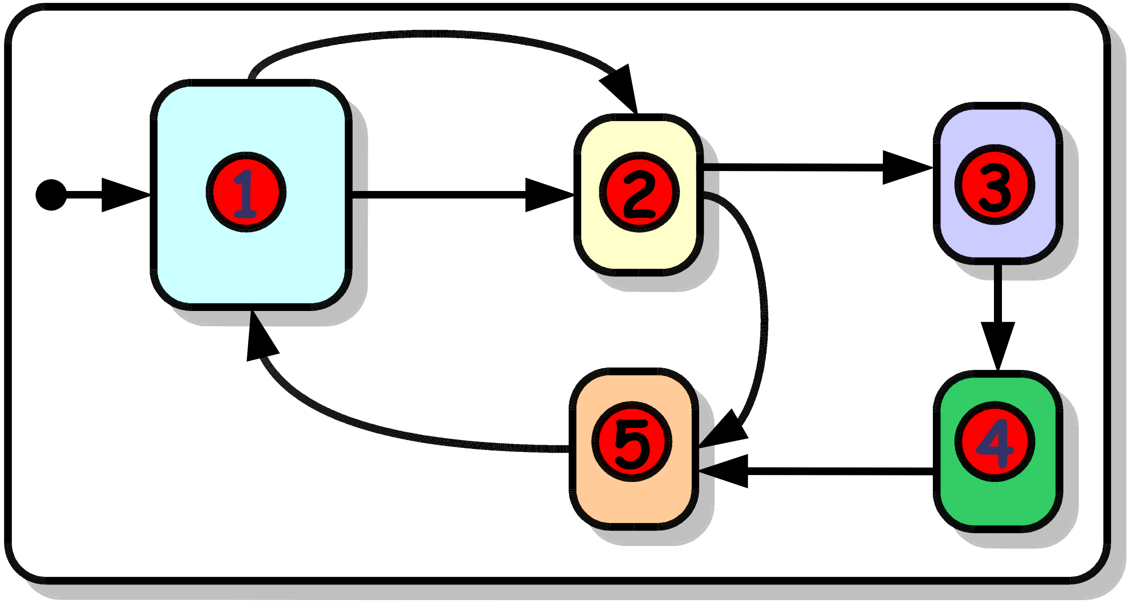
Default Entry

- **One and only one for each superstate.**
- Indicate the default initial state whenever the superstate is entered.



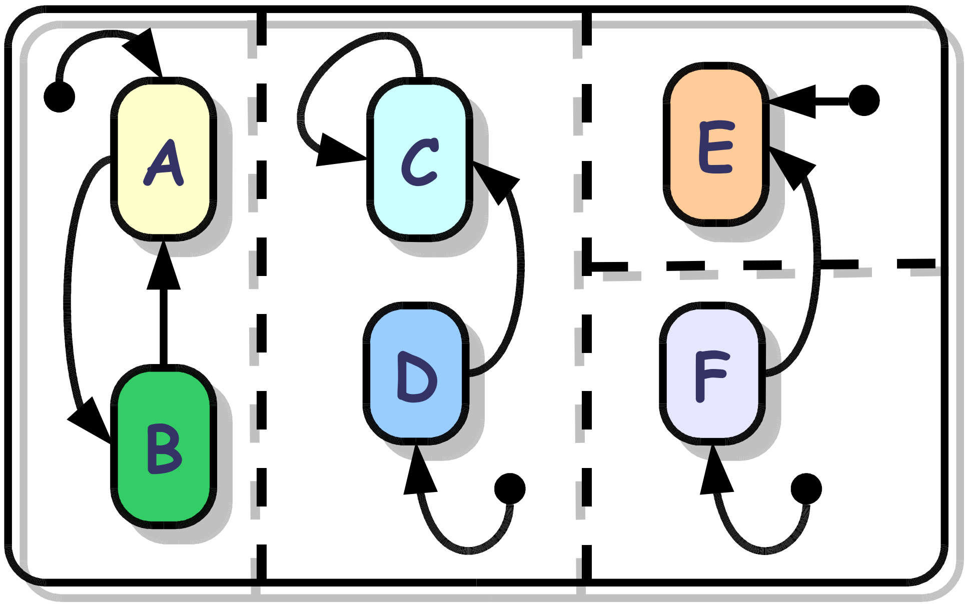
OR-States

At any given time, we are in **one and only one** OR-State.



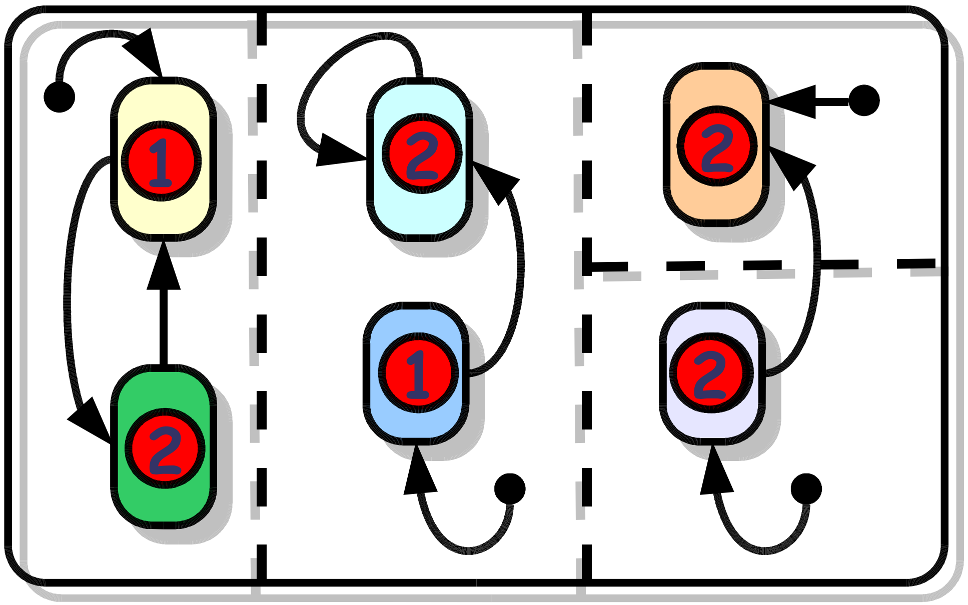
AND-States

- Parallel composition via split of a superstate (dashed line)
- Each split is said to be an OR-state.
- No limitation in the number of OR-states.



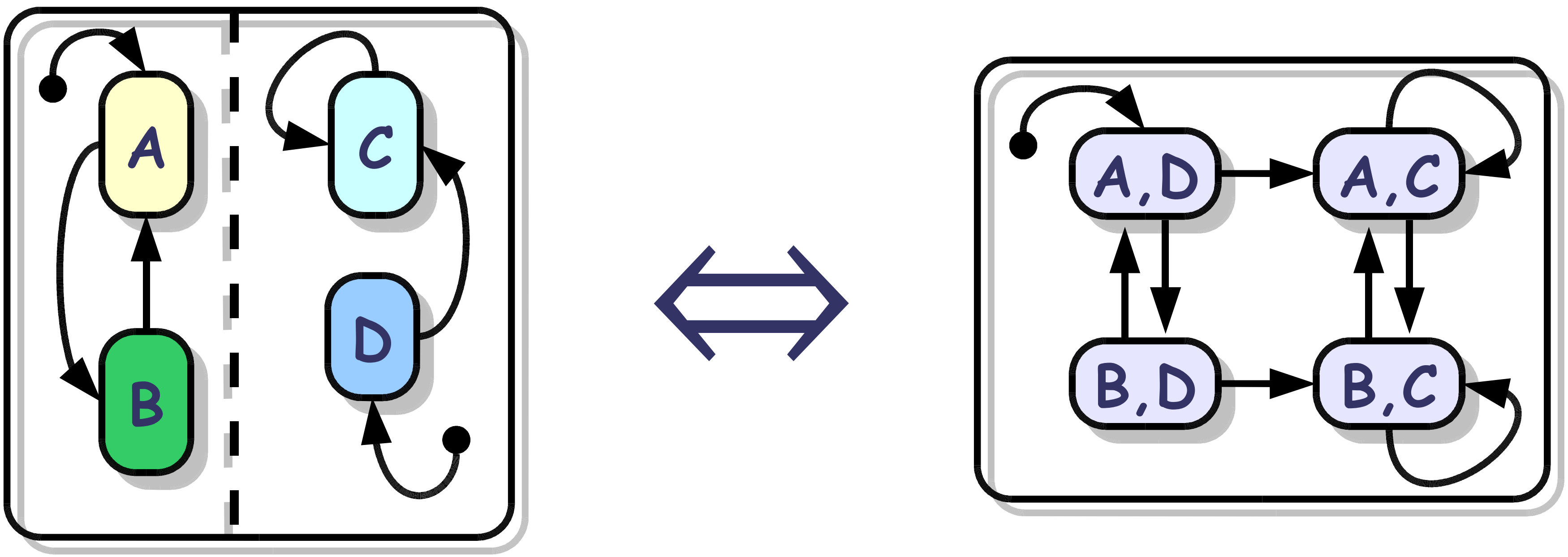
AND-States

At any given time, we are in **every** AND-States.



AND-States

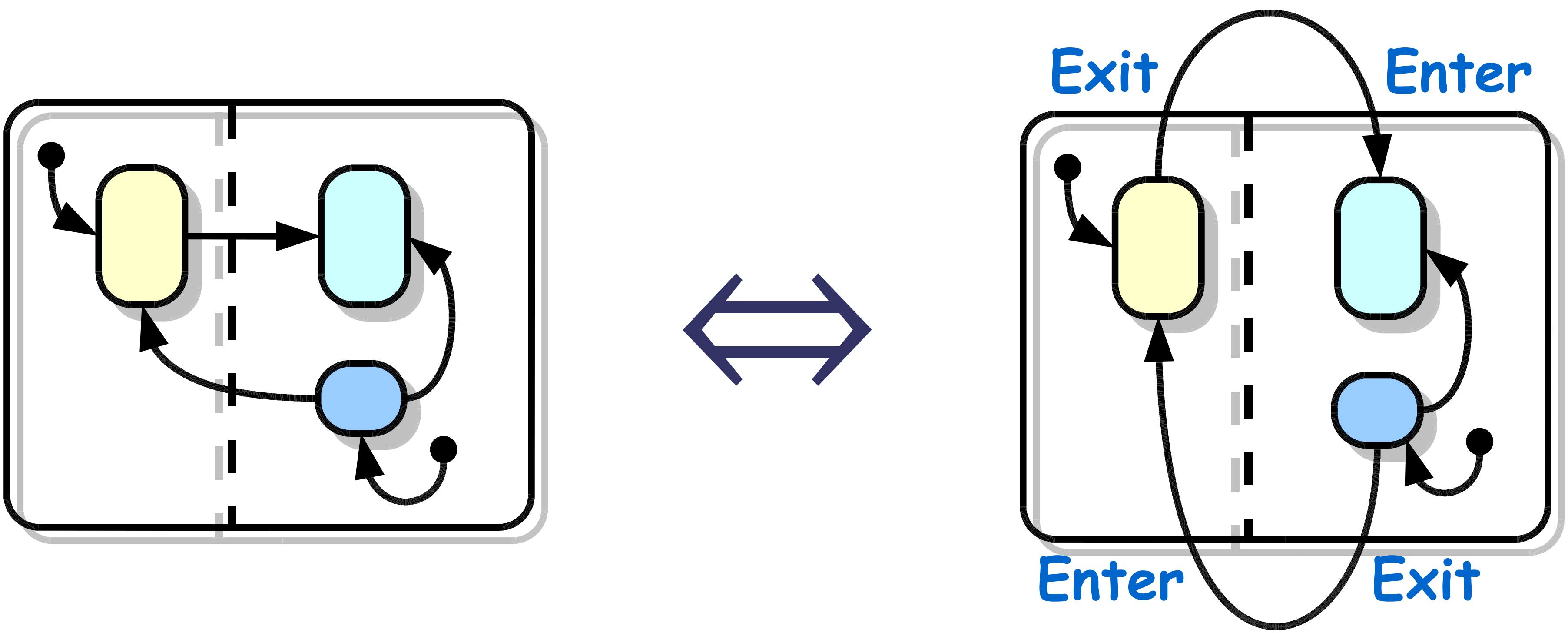
Semantically equivalent to the cartesian product of each automaton contained in one AND-state



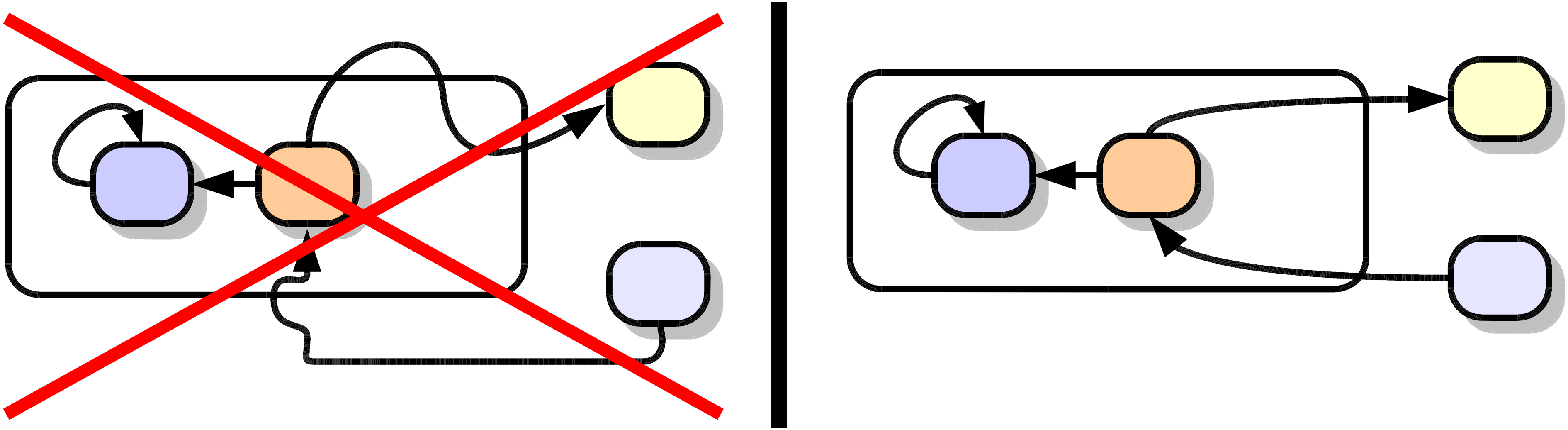
Intend to reduce the number of states !

Crossing Dashed Line Edges

What should happen here ????

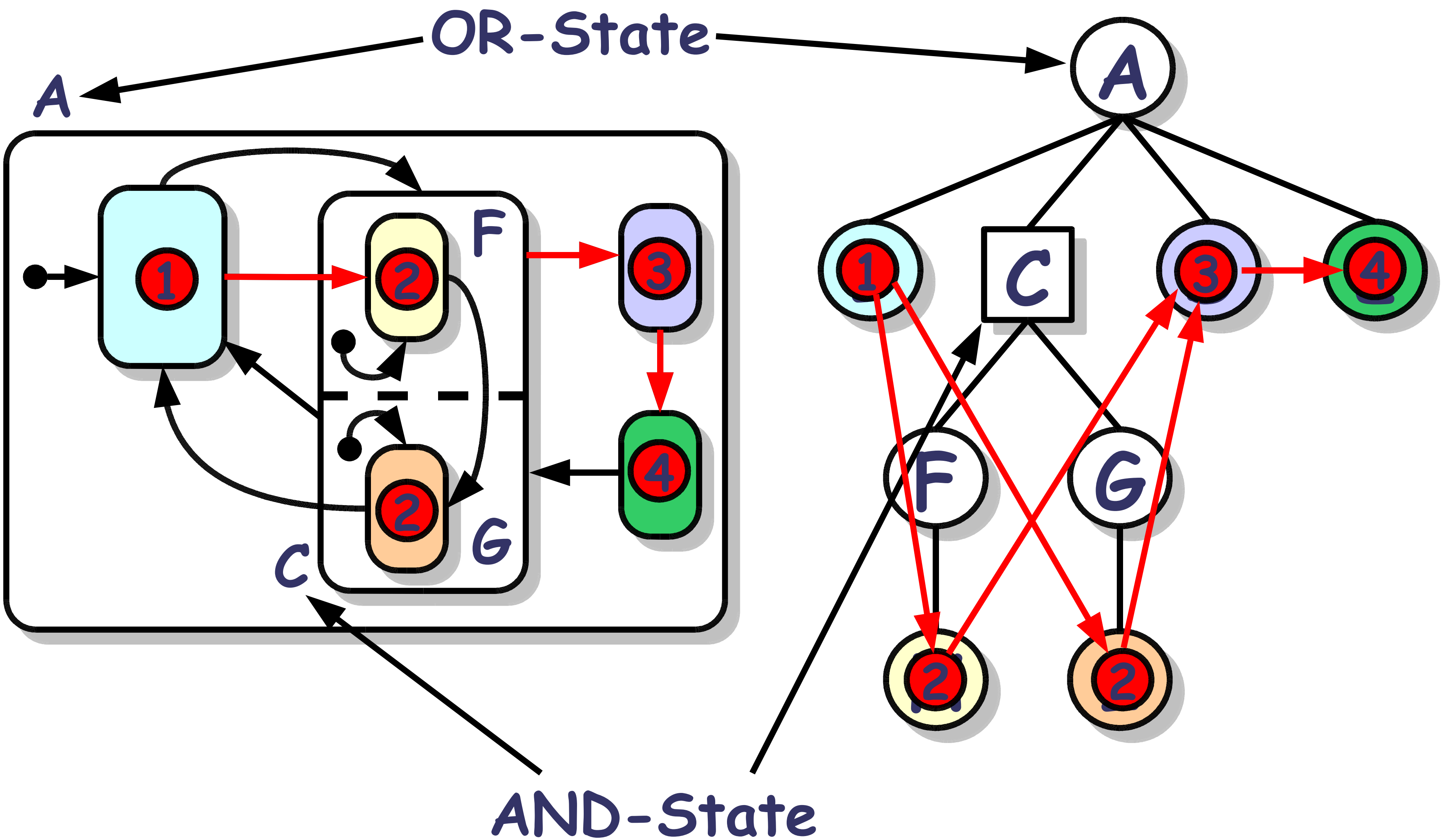


Multiple-Crossing Edges

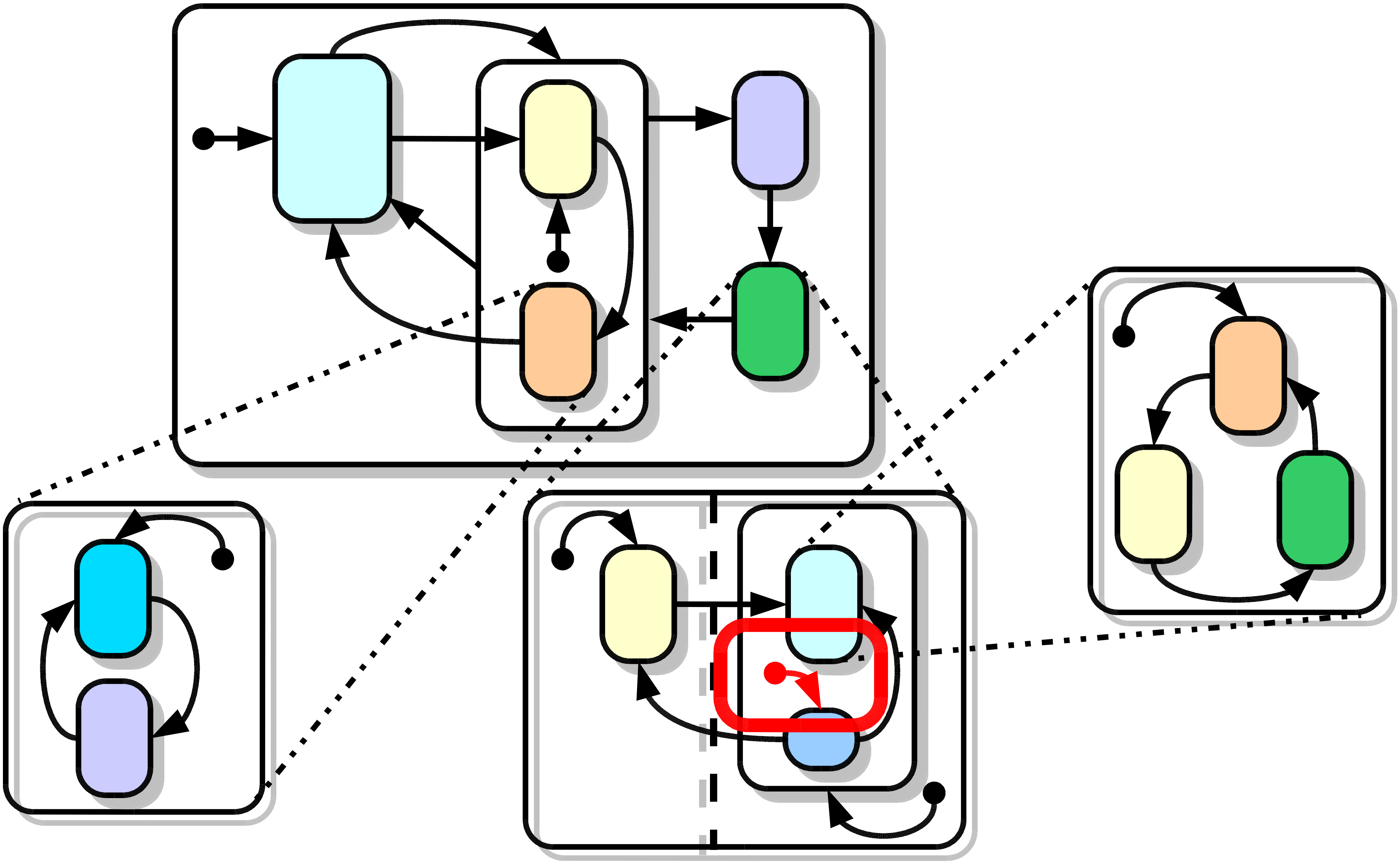


- Each edge cross a superset border only once.
- Redundant edges are pruned.
- Intended to remove unnecessary complexity of the model

Symbolic Representation



Example



And... Something's missing

We still need to handle input/output messages !!!
And higraphs do not provide any mechanism for this...

Finite State Machines

Acceptors/Recognizers

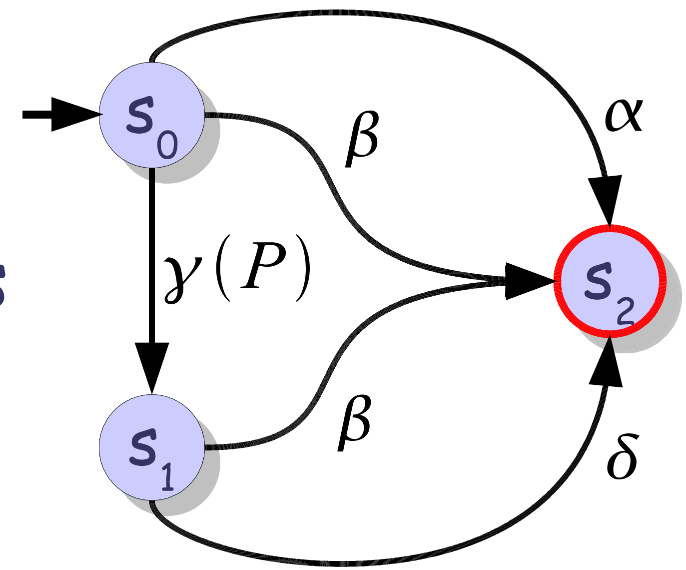
- Finite Automata
- Büchi Automata
- ... etc ...

The acceptor can decide if a word is in his language or not

Acceptors/Recognizers

An acceptor is a 5-tuple (Q, A, T, s_0, F):

- **Q** a finite set of states;
- ***A*** a (possibly infinite) alphabet;
- $T \subseteq Q \times A \times Q$ a finite set of edges
- s_0 is the initial state
- $F \subseteq Q$ the set of accepting states

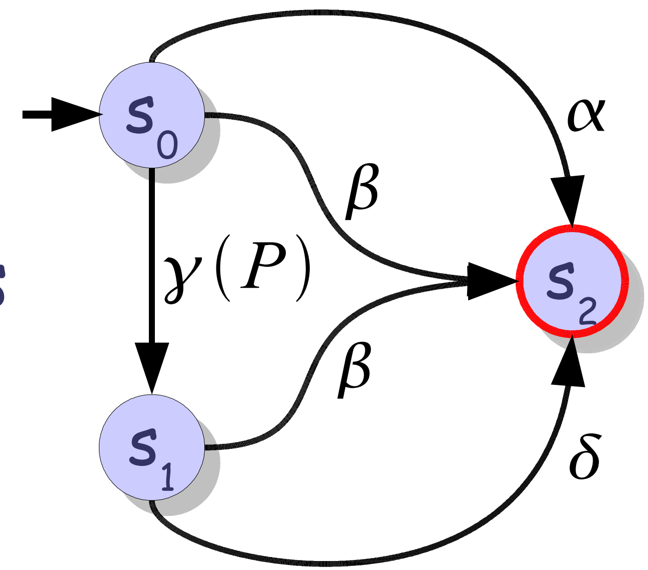


The accepting condition change depending on the kind of acceptor we consider.

Finite Automata

A Finite automata is a 5-tuple (Q, A, T, s_0, F):

- **Q** a finite set of states;
- **A** a (possibly infinite) alphabet;
- $T \subseteq Q \times A \times Q$ a finite set of edges
- s_0 is the initial state
- $F \subseteq Q$ the set of accepting states



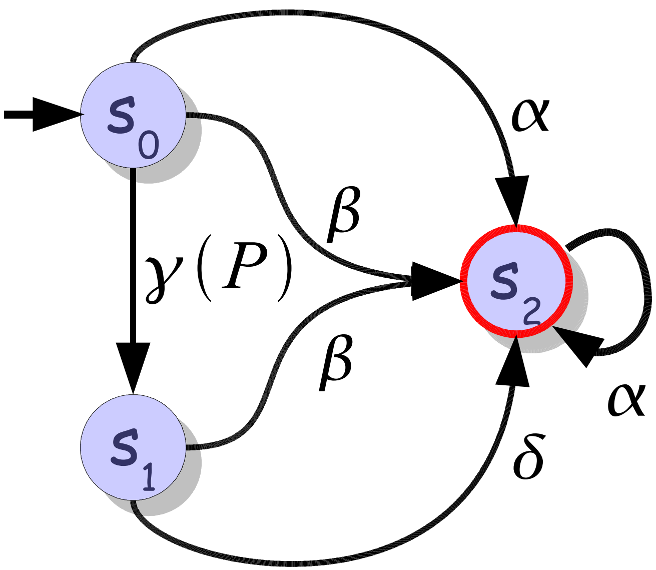
The word should <u>end</u> in one final state !

$$L = \{\{\alpha\}, \{\beta\}, \{\delta\}, \{\gamma, \beta\}, \{\gamma, \delta\}\}$$

Büchi Automata

A Büchi automata is a 5-tuple (Q, A, T, s_0, F):

- **Q** a finite set of states;
- **A** a (possibly infinite) alphabet;
- $T \subseteq Q \times A \times Q$ a finite set of edges
- s_0 is the initial state
- $F \subseteq Q$ the set of accepting states

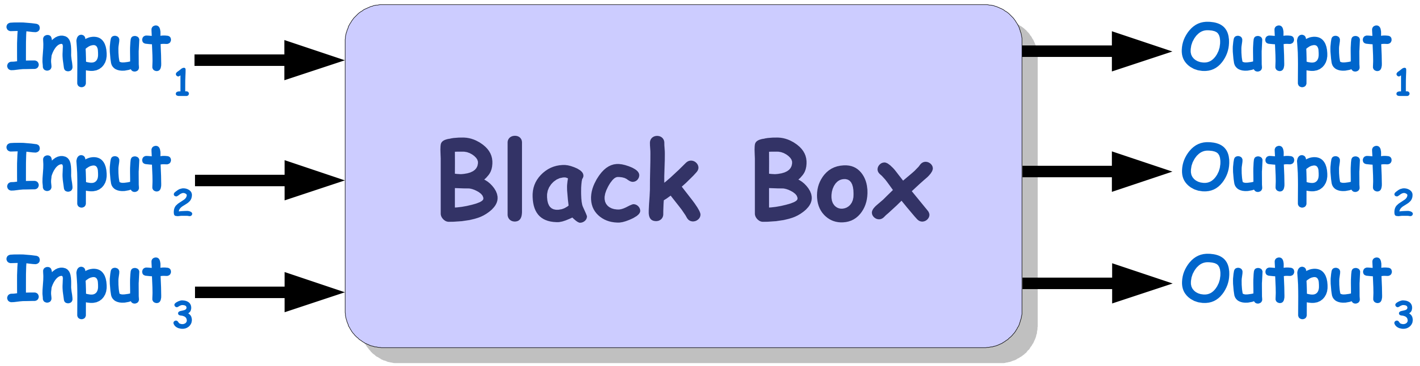


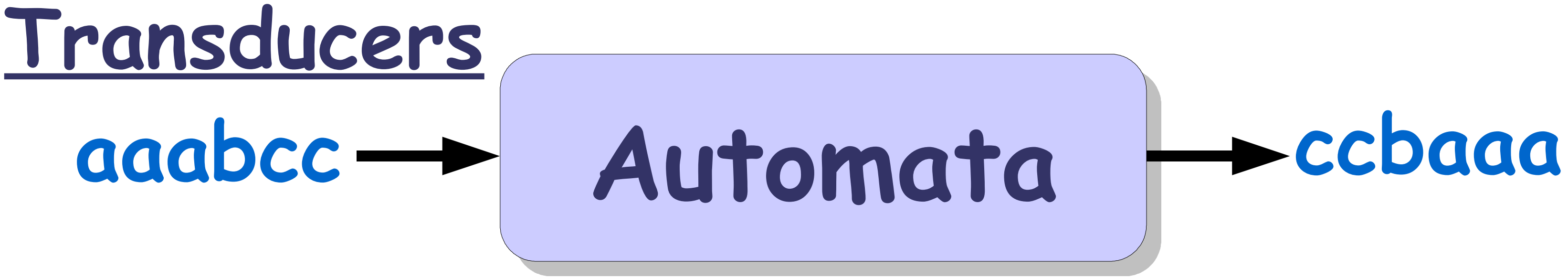
The word should go <u>infinitely often</u> in one final state

$$L = \{\{\alpha^+\}, \{\beta, \alpha^+\}, \{\delta, \alpha^+\}, \{\gamma, \beta, \alpha^+\}, \{\gamma, \delta, \alpha^+\}\}$$

Still not quite that...

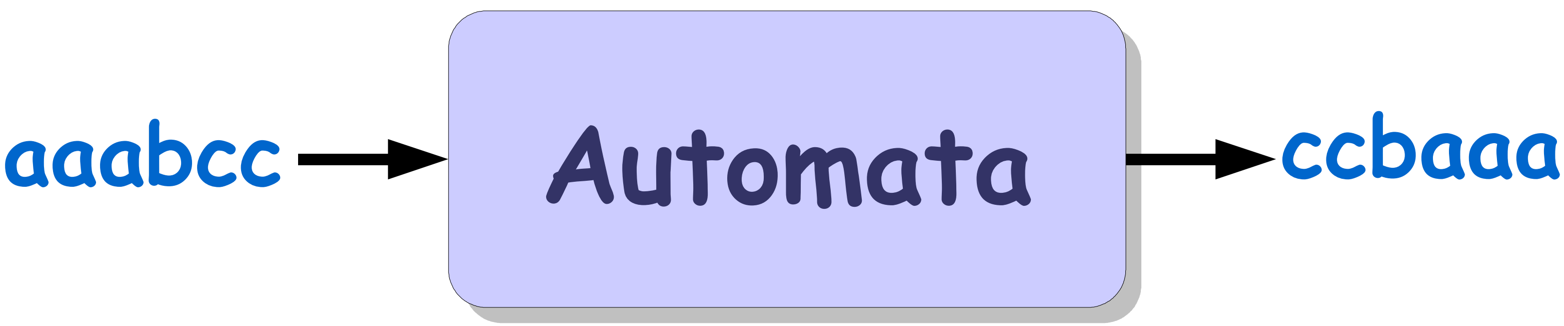
We want more than a "YES" or "NO" answer.
We want to output another word.





Transducers

- Moore Machines
- Mealy Machines
- ... etc ...



Transducers

A Transducer is a 6-tuple $(Q, A_{in}, A_{out}, T, O, s_0)$:

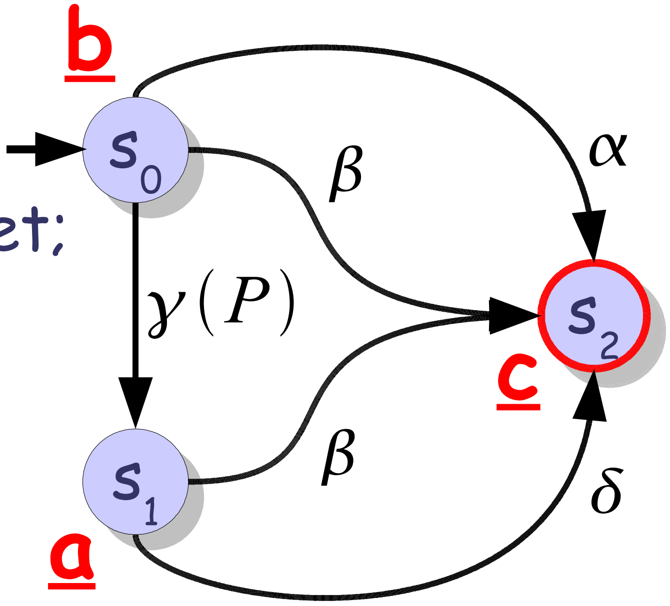
- **Q** a finite set of states;
- A_{in} a (possibly infinite) input alphabet;
- A_{out} a (possibly infinite) output alphabet;
- $T \subseteq Q \times A_{in} \times Q$ a finite set of edges
- $O \subseteq Q \times A_{in} \times A_{out}$ a set of outputs
- s_0 is the initial state

There are several ways to define the output function.

Moore Machines

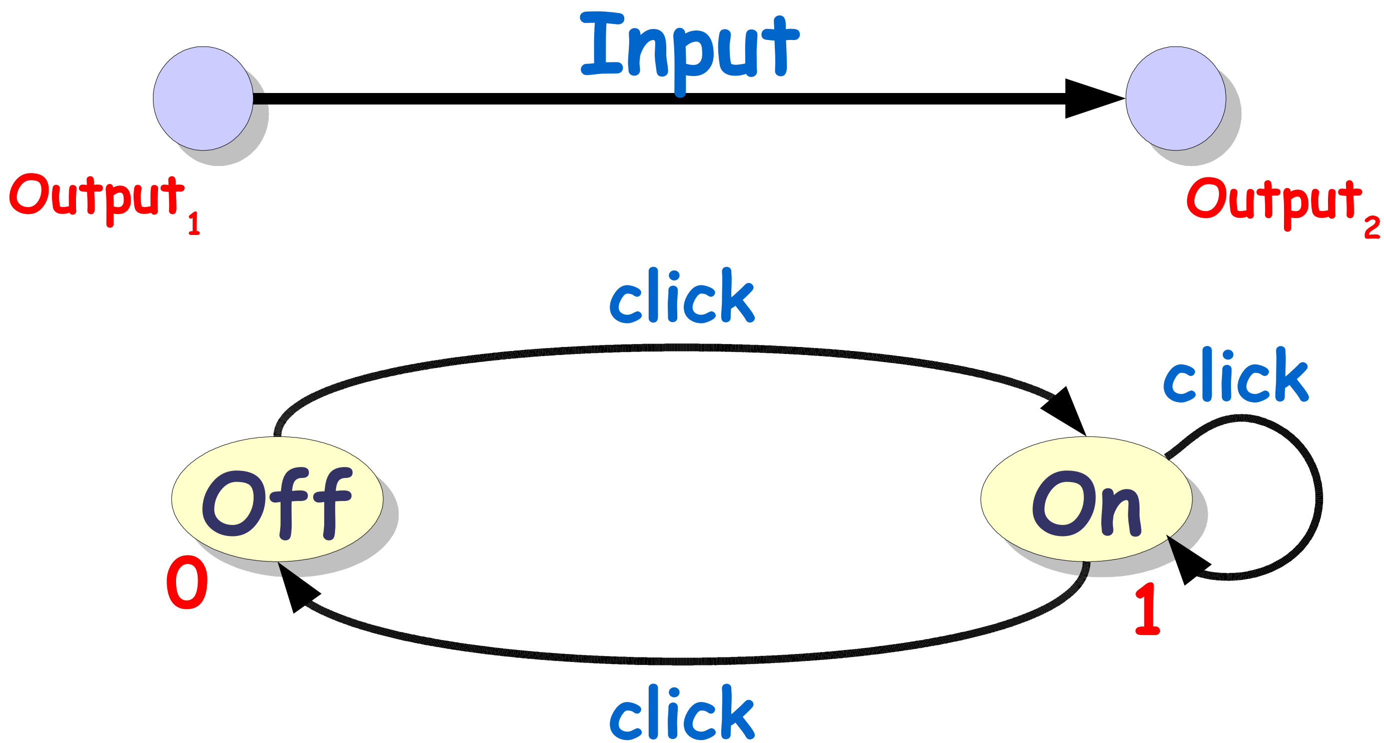
A Transducer is a 6-tuple $(Q, A_{in}, A_{out}, T, O, s_0)$:

- **Q** a finite set of states;
- A_{in} a (possibly infinite) input alphabet;
- A_{out} a (possibly infinite) output alphabet;
- $T \subseteq Q \times A_{in} \times Q$ a finite set of edges
- $O \subseteq Q \times A_{out}$ a set of outputs
- s_0 is the initial state



The output depends in which state you are in

Example (Moore Machine)



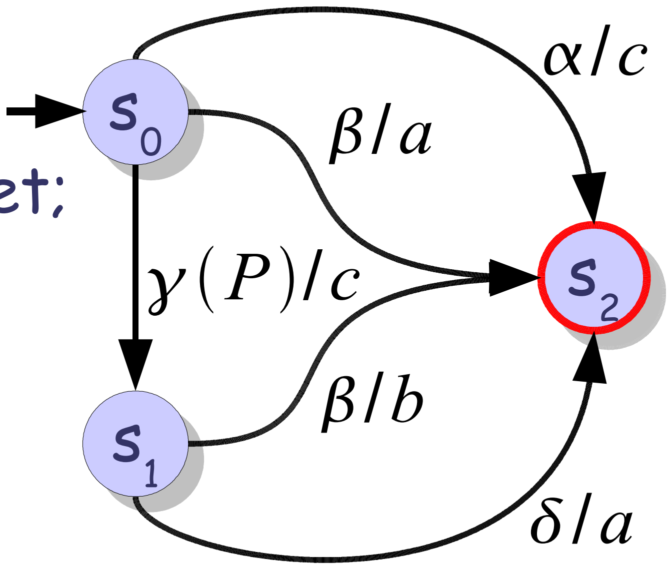
Input String: click, click, click, ...

Output String: 0, 1, 1, 1, 0, ...

Mealy Machines

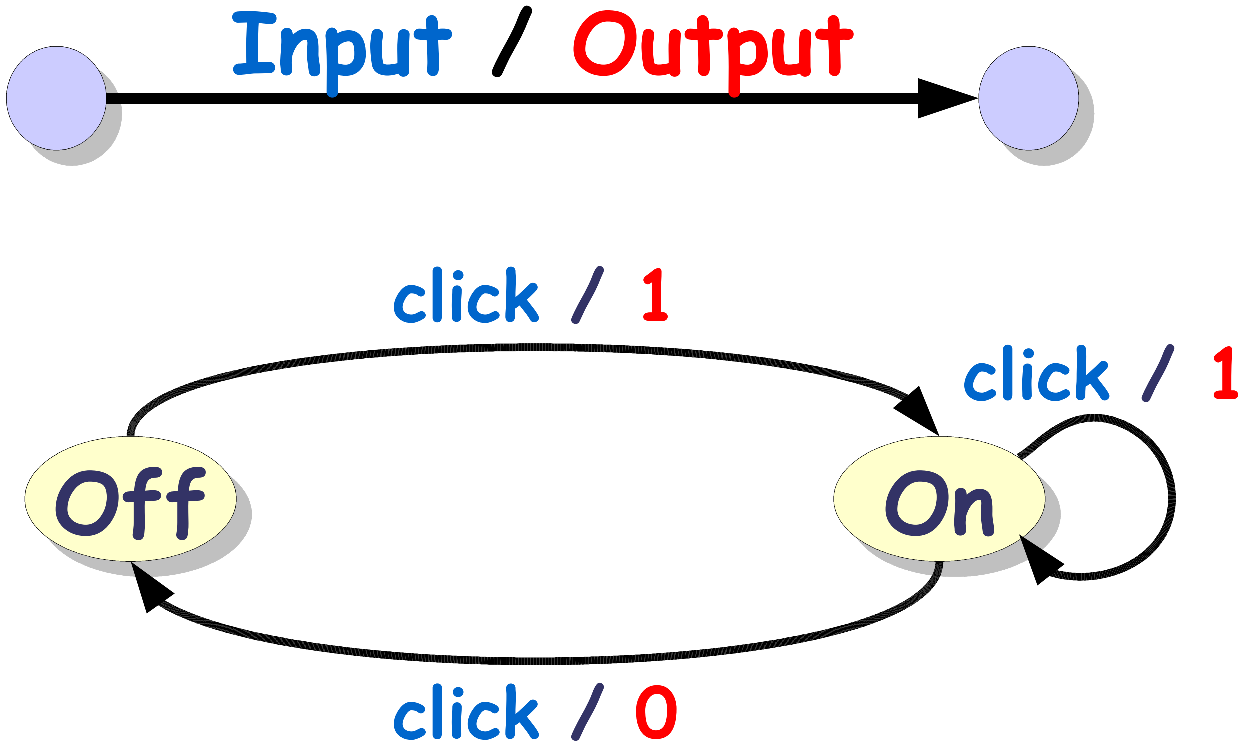
A Transducer is a 6-tuple $(Q, A_{in}, A_{out}, T, O, s_0)$:

- **Q** a finite set of states;
- A_{in} a (possibly infinite) input alphabet;
- A_{out} a (possibly infinite) output alphabet;
- $T \subseteq Q \times A_{in} \times Q$ a finite set of edges
- $O \subseteq Q \times A_{in} \times A_{out}$ a set of outputs
- s_0 is the initial state



The output depends on which transition you are taking

Example (Mealy Machine)



Input String: click, click, click, ...

Output String: 1, 1, 1, 0, 1, 0, ...

Statecharts

Statecharts in a Nutshell

- What are the Statecharts ?
  - Represent the behavioural view of the system.
  - Visual formalism for describing states and transitions in modular way.
- What is the purpose of using Statecharts ?
  - To suppress and organize details.
  - Best if graphical. The clarity they provide can be lost if they are represented in tabular form.

Advantages of Statecharts

- A hierarchical structure to reduce complexity and support abstraction
- AND/OR superstates
- Concurrency & Orthogonality
- Compact & Expressive
- Global Communication Mechanism
- Formal enough to avoid ambiguity (code generation is possible)

Two Types of Models

2 Types of statechart development:

- Harel's Statechart
  - Developed by David Harel.
  - First developed for function-oriented systems.
  - Later extended for OO systems with few changes.
- UML Statechart
  - Developed by Object Management Group (OMG).
  - Extend the properties of Harel's statecharts with some new features.

Harel's Statecharts

Harel's Statecharts

Harel's Statecharts =
state-diagrams + depth + orthogonality + broadcast

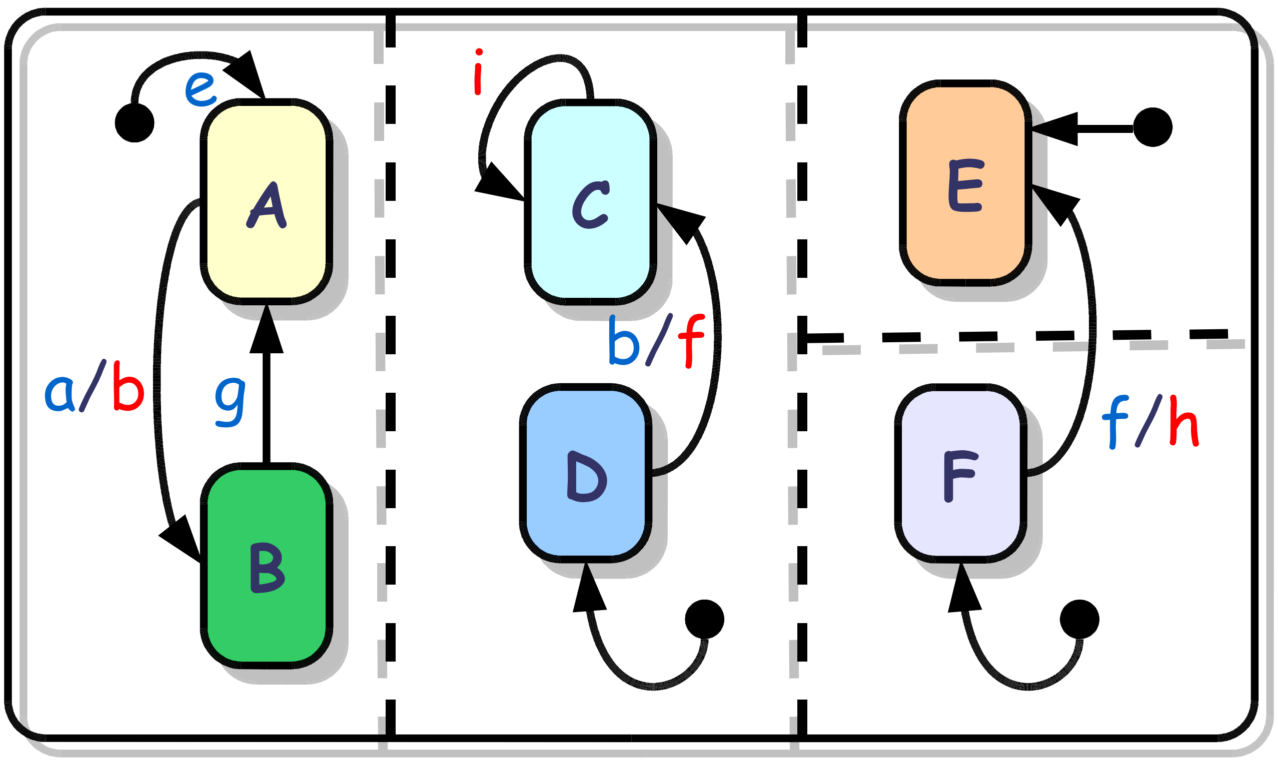
Harel's Statecharts is an higraph with mealy machine's like communication and extra macro-commands like:

- (Deep/Shallow) History
- Joins and Forks
- Conditional
- Selection
- Timeout

Synchronization/Broadcast

Mealy machine's communication which allows:

- Synchronizations
- Broadcasts



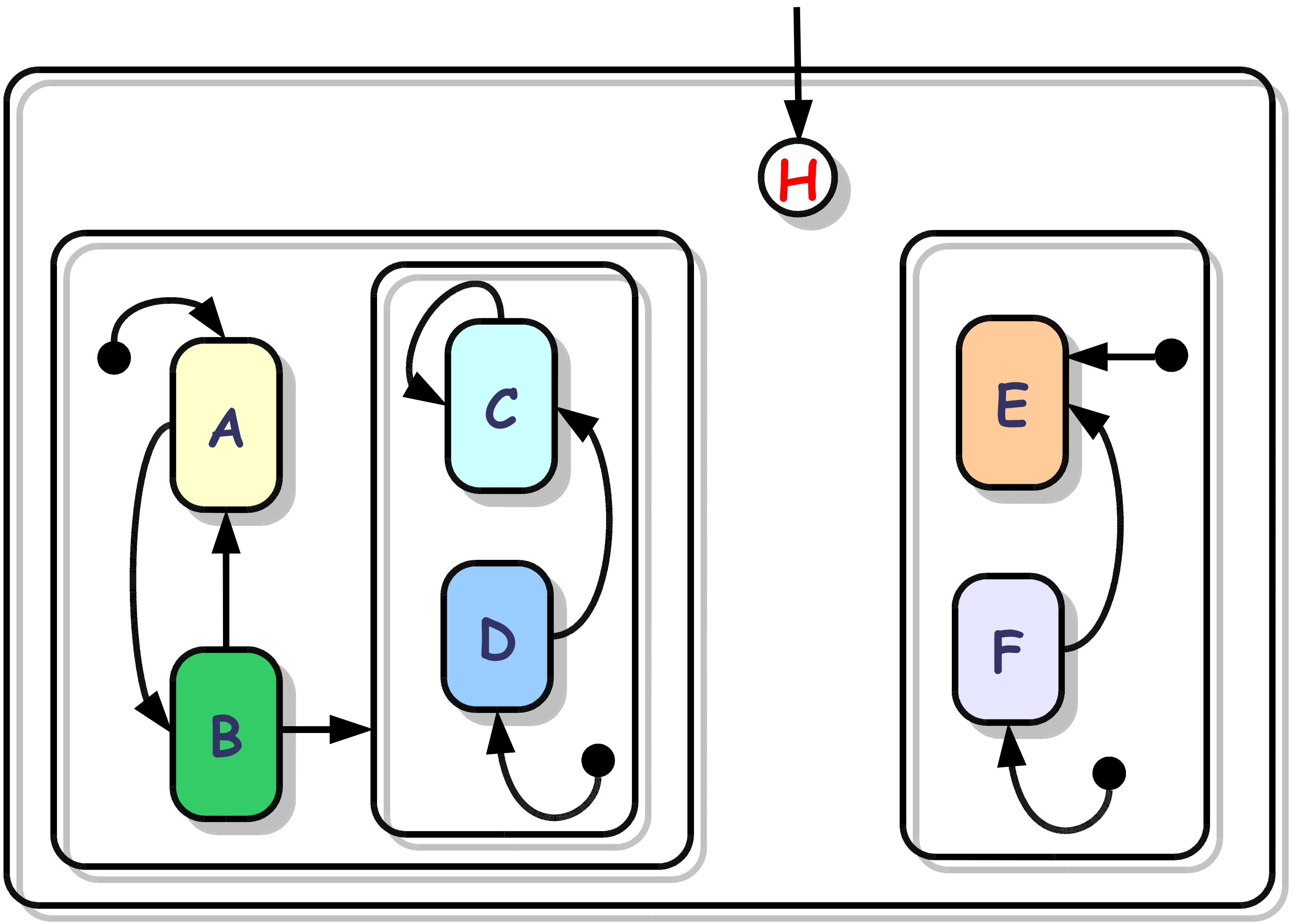
History Entries

An History Entry give the most recently visited state of the entered superstate

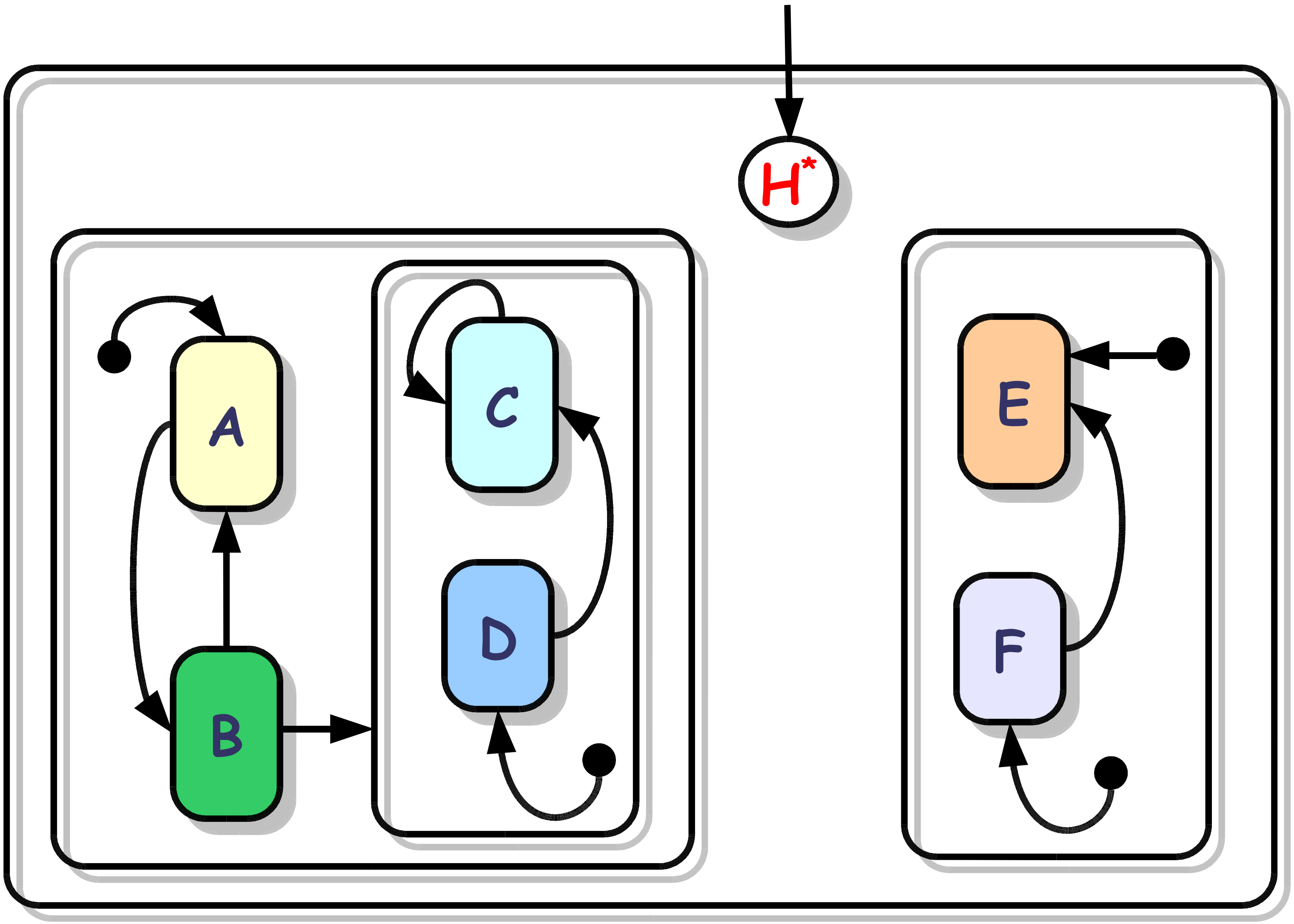
2 Types of History Entries:

- Shallow History (H): Represents the most recently entered state at the same level.
- Deep History (H*): Represents the most recently visited state whatever how deep is the state.

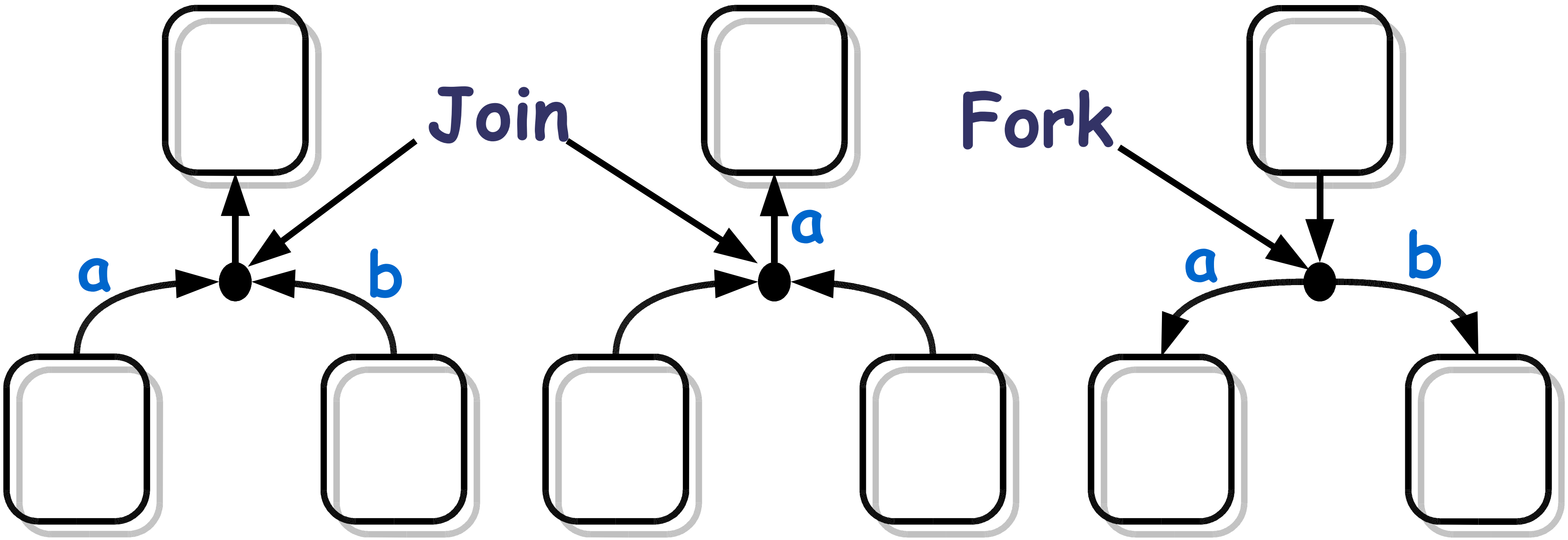
Shallow History



Deep History



Joins & Forks



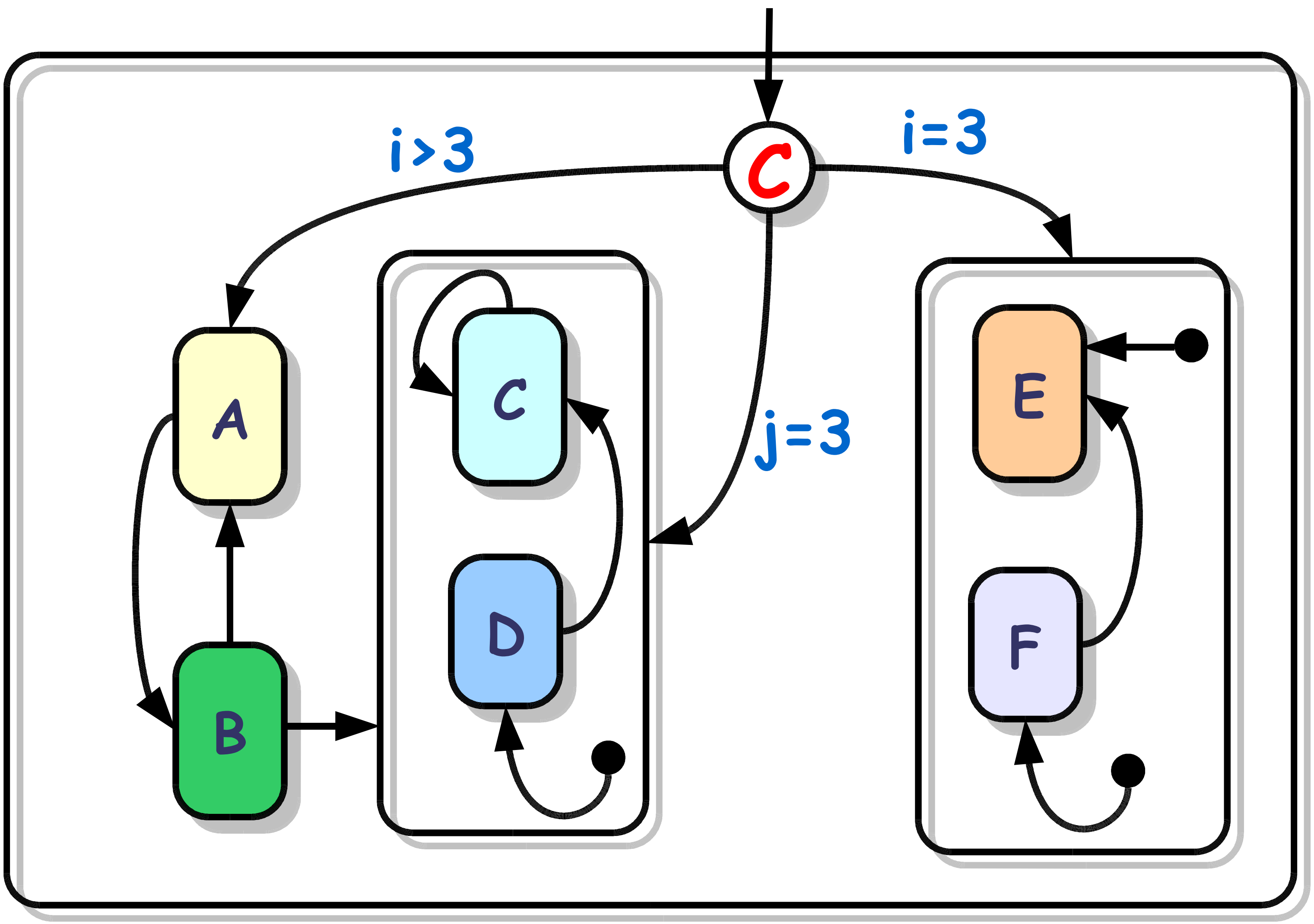
Take care of the guards !!!

Condition & Selection

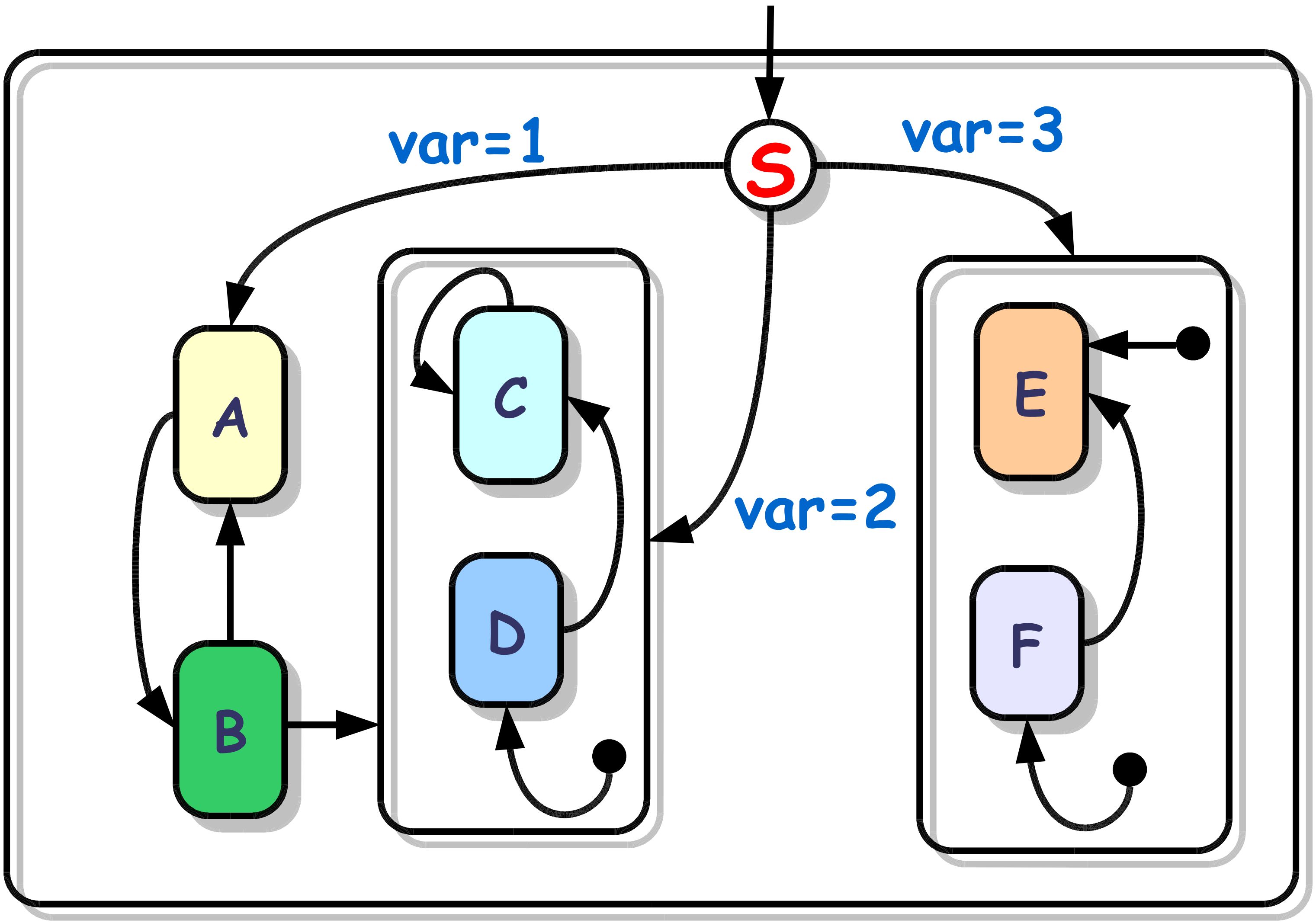
- **Condition (C):**
  When entering the superstate, a condition is checked and a sub-state is chosen.

- **Selection (S):**
  When entering the superstate, a variable is checked and a sub-state is chosen.

Condition

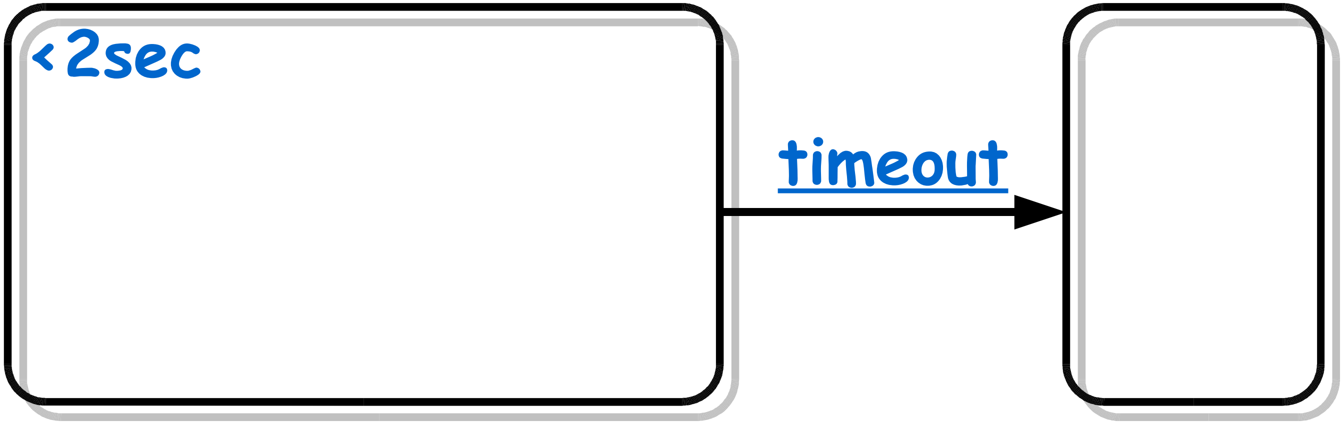


Selection



Time-out

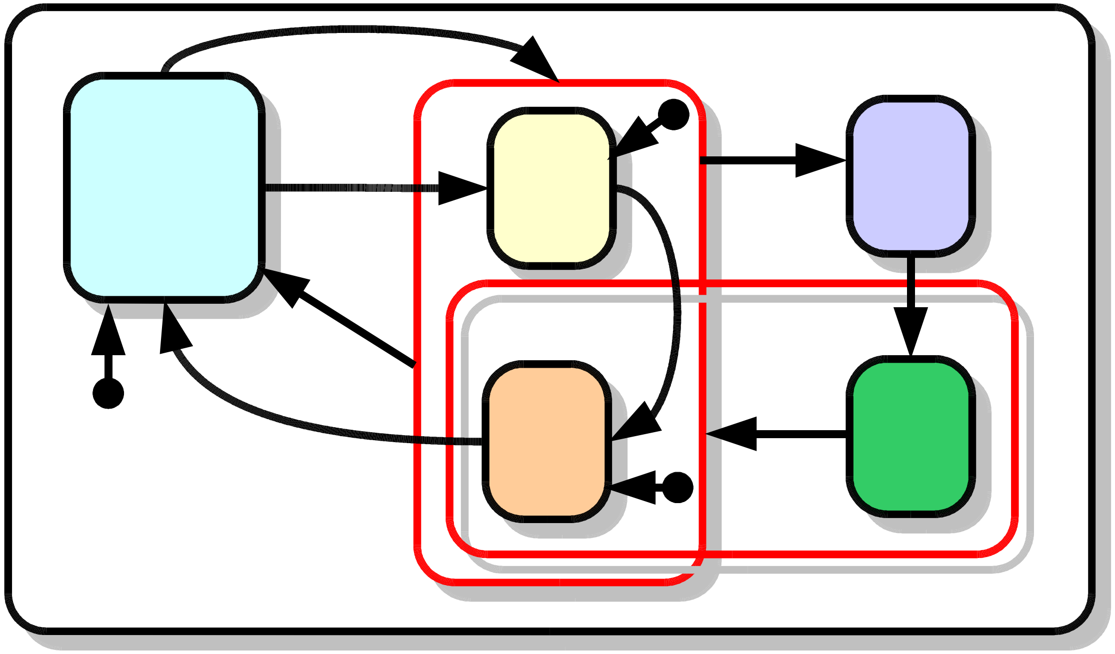
Force to leave the (super)state
after the time-out has expired
through the timeout transition.



But, time is not inherent in the model
So be careful !

But...

Still some semantics problems !!!



Problems of Harel's Model

- **Semantics:**

  Many papers published with flaws or ambiguities. None giving the complete formal semantics

- **Notion of Time:**

  Each transition is supposed to take no time which is an unrealistic assumption in RT systems

- **Determinism:**

  The model is easily made non-deterministic which is a problem at code generation

- **... etc ...**

UML
Statecharts

What is UML ?

UML = Unified Modelling Language

Introduced by the Object Management Group (OMG) in 1997 as a standard modelling language for object-oriented applications

UML defines a the **syntax** and the **semantics** of a set of visual formalisms (**diagrams**) which gives different perspectives of a software system

UML 1.0

- **Use Case Diagram:** Interactions between the system and the users
- **Class Diagram:** Class hierarchy and data layout over the classes
- **State Diagram:** Behaviour of the system abstracted as a set of states and transitions
- **Communication Diagram:** Object interactions
- **Sequence Diagram:** Time sequence of object interactions
- **Component Diagram:** High-level packaged structure of the code
- **Deployment Diagram:** Physical architecture and deployment of components on the hardware architecture

UML 2.0

- **Structural Modelling Diagrams**
  - Package Diagrams
  - Class Diagrams
  - Object Diagrams
  - Composite Structure Diagrams
  - Component Diagrams
  - Deployment Diagrams
- **Behavioural Modelling Diagrams**
  - Use Case Diagrams
  - Activity Diagrams
  - State Machine Diagrams
  - Communication Diagrams
  - Sequence Diagrams
  - Timing Diagrams
  - Interaction Overview Diagrams

UML Statecharts

Harel's Statecharts was introduced in UML with modification of the semantics and some additional elements

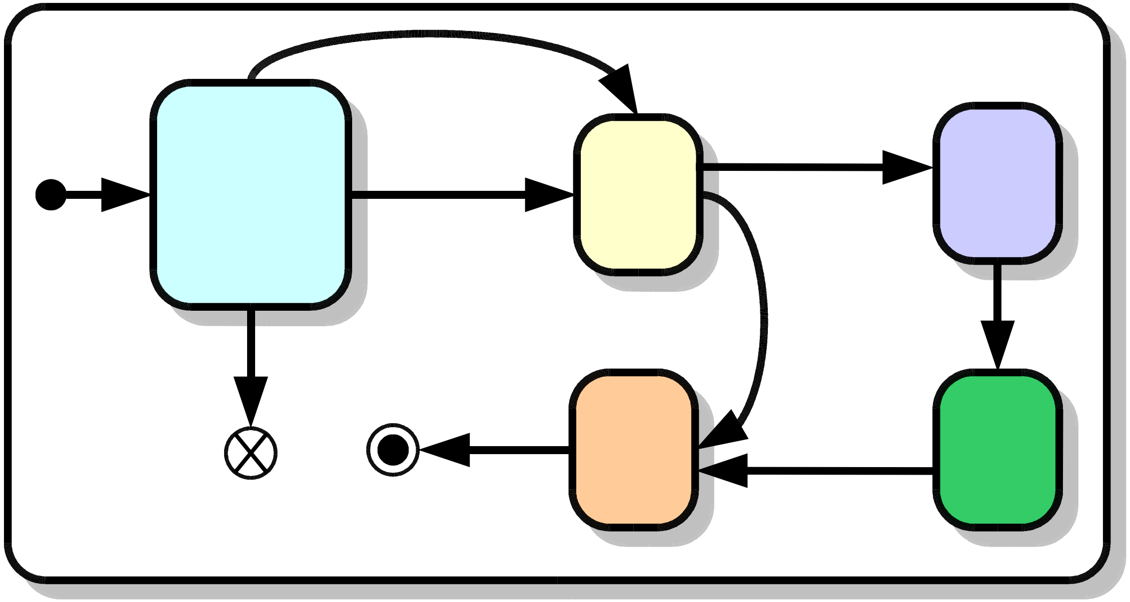
- **UML Statecharts inherits:**
  AND/OR-states, (Shallow/Deep) History, Fork/Join, Condition, Time-out.
- **UML Statecharts introduces:**
  Synch, Terminal, Junction, Stub.

New Properties

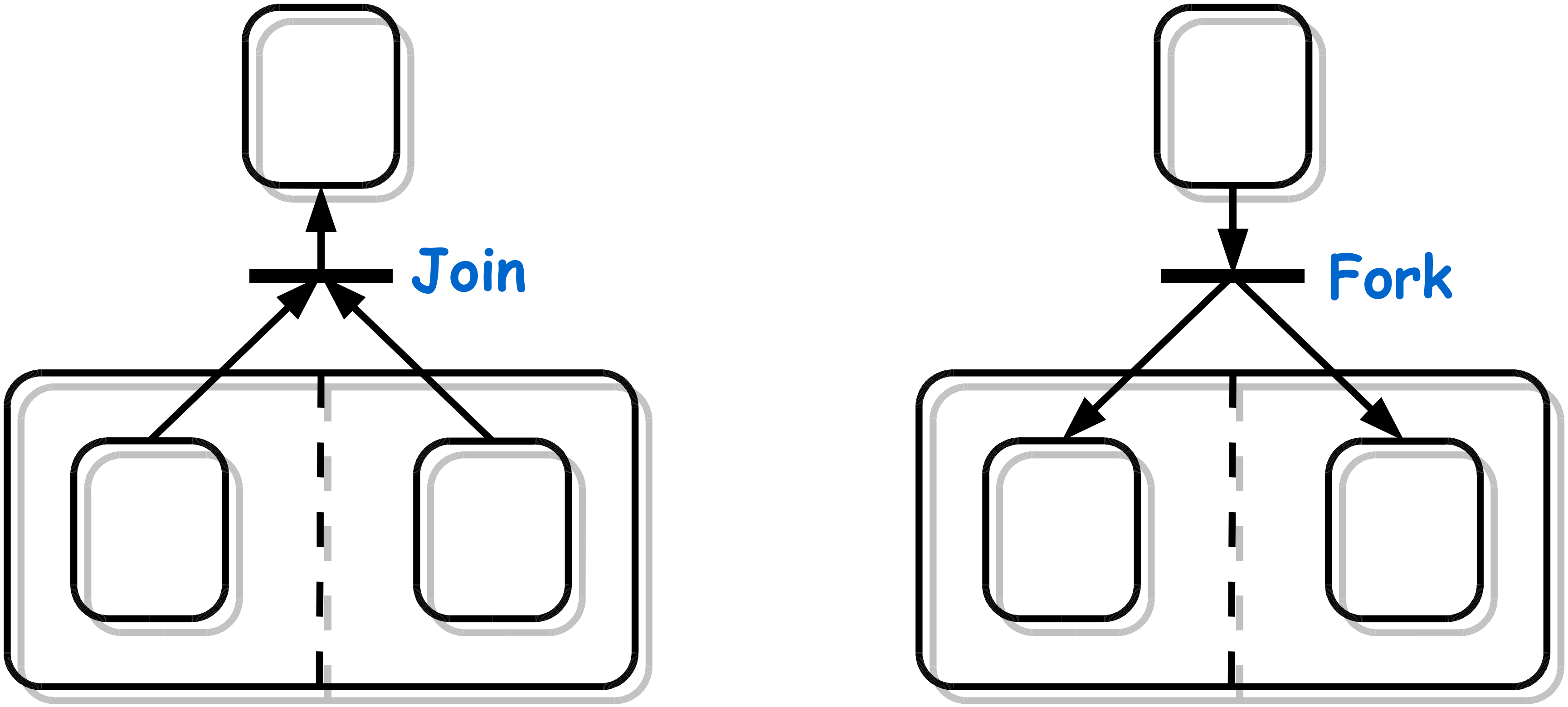
- Transitions are supposed to take an "insignificant" amount of time... (sigh!)
- The model doesn't support overlapping superstates (clearer semantics)
- Events can carry parameters and variables
- Introduce a "pseudo-states" class (history, fork, join, condition, ...)

Starting/Ending

- Initial pseudo-state
- Final pseudo-state
- End of subtask

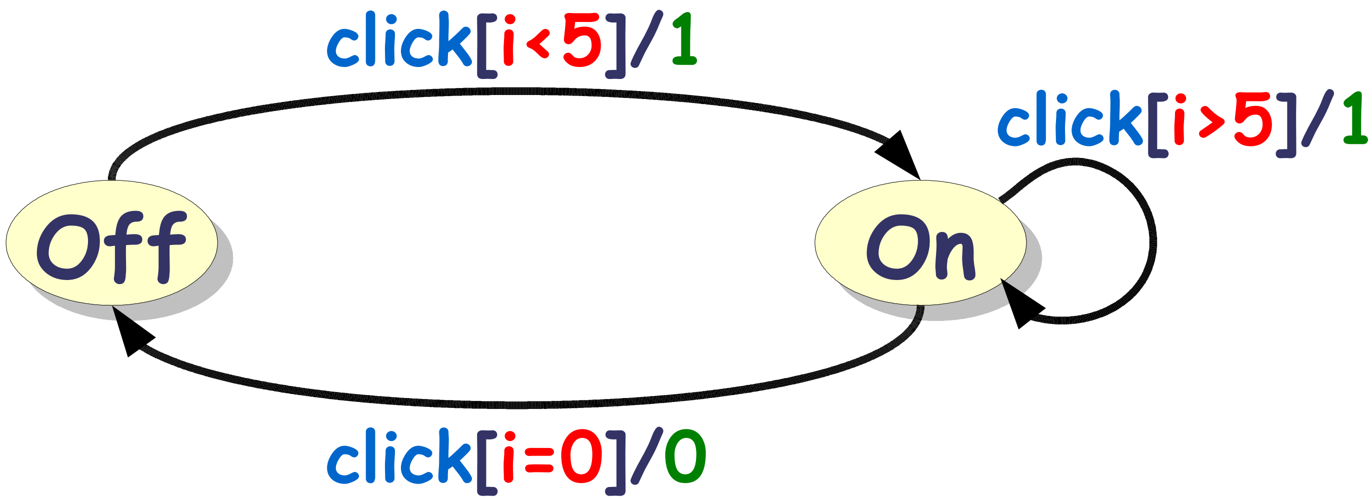


Synchronization (Fork/Join)

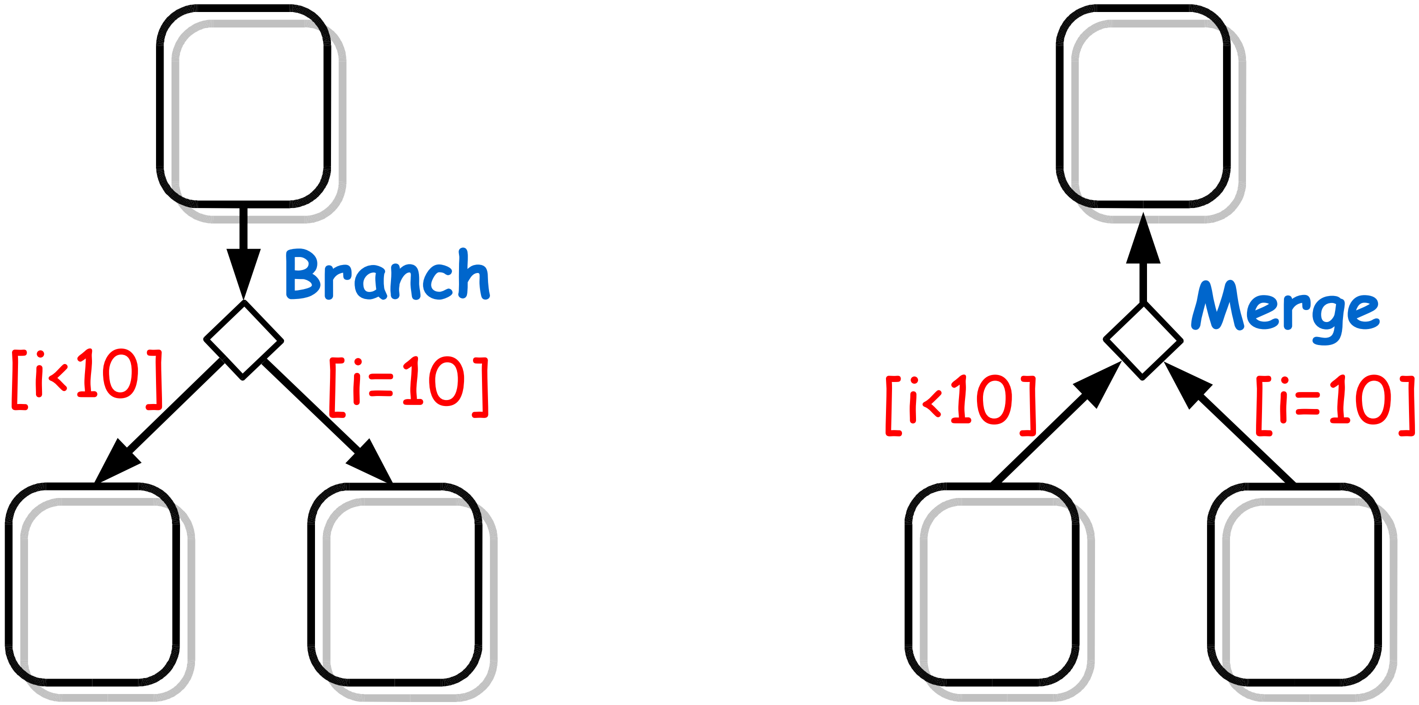


Used to enter or exit AND-states

Transitions

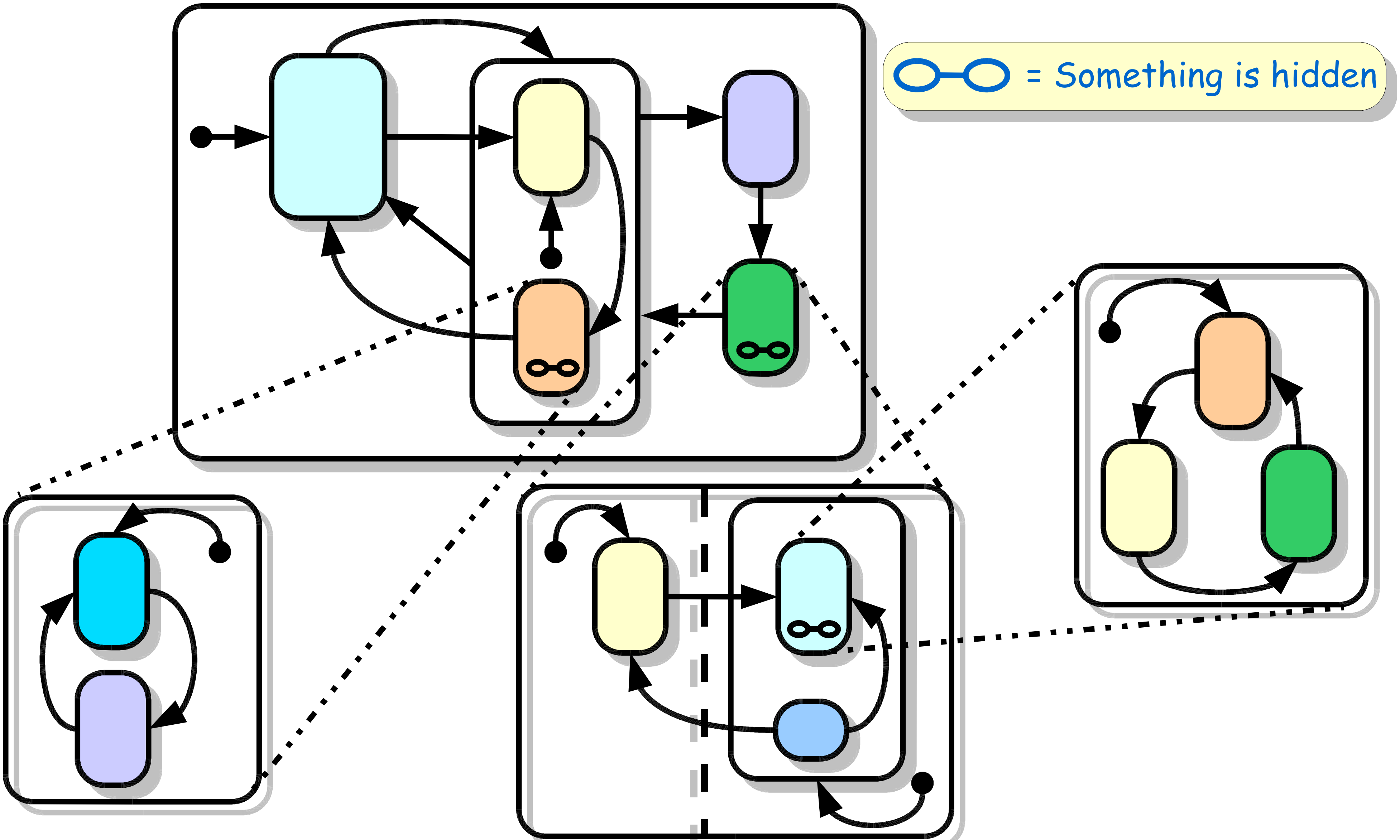


Decision (Branch/Merge)

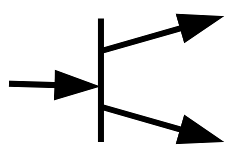
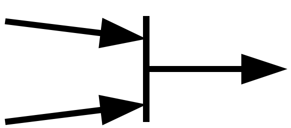


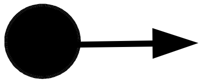
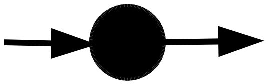
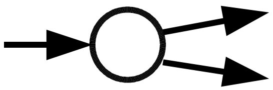
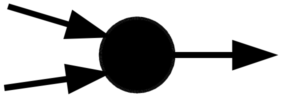
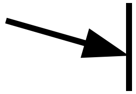
Take care of the guards !!!

Hiding Superstates



Pseudo-states

Symbol	Name
Ⓒ OR ◇	Conditional
Ⓣ OR ◉	Terminal
(*) OR (n)	Synch
	Fork
	Join

Symbol	Name
Ⓗ	Shallow History
(H*)	Deep History
	Initial/Default
	Junction
	Choice Point
	Merge Junction
	Stub

Problems of UML Model

- **Semantics:**

  Many papers published with flaws or ambiguitie
  giving the complete formal semantics

- **Notion of Time:**

  Each transition is supp time which is an
  unrealistic assu ystems

- **Dete**

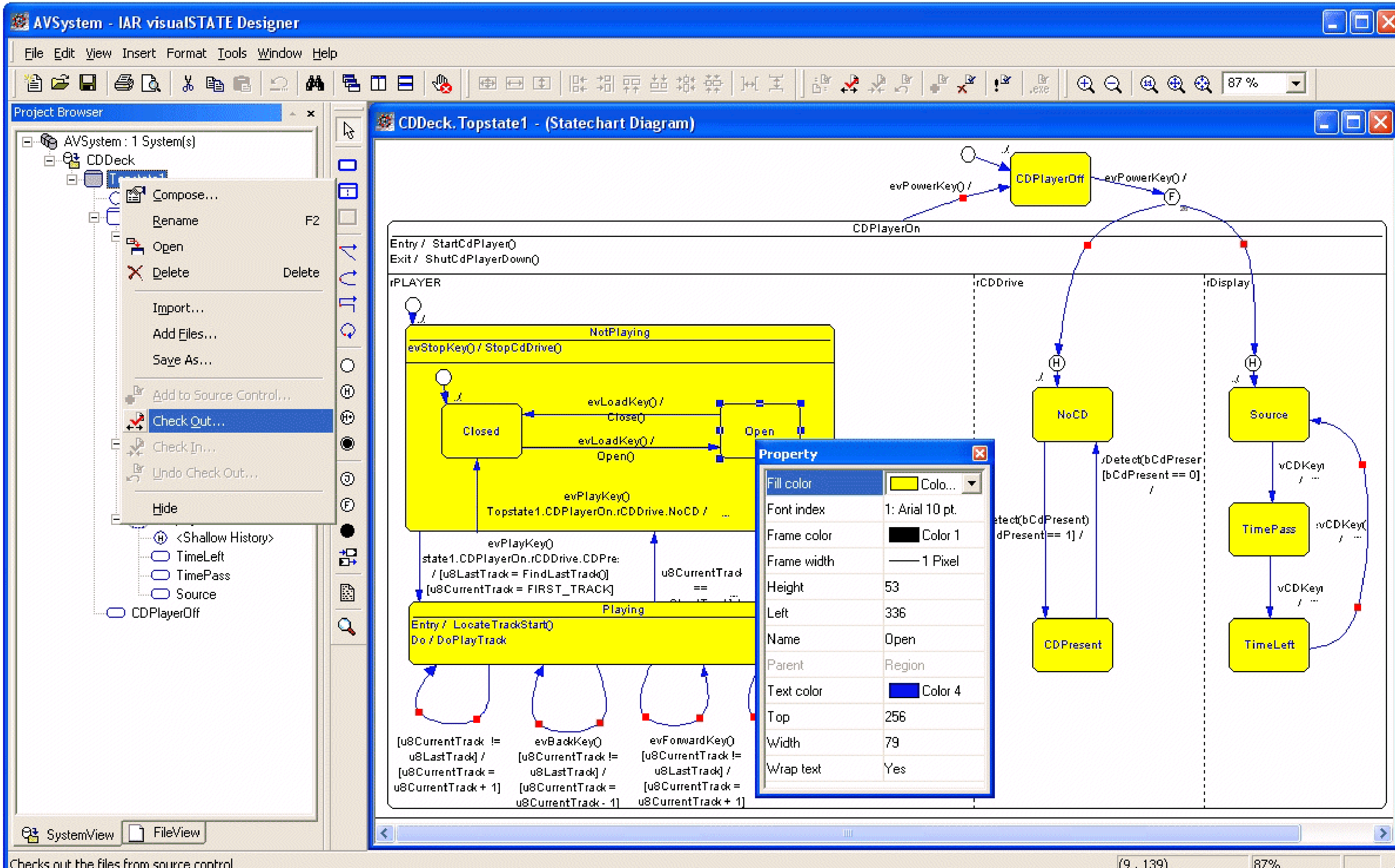
  s easily made non-deterministic which is a
  em at code generation

- **... etc ...**

None of the problems have been solved !

Modelling Tools

VisualSTATE 5.1



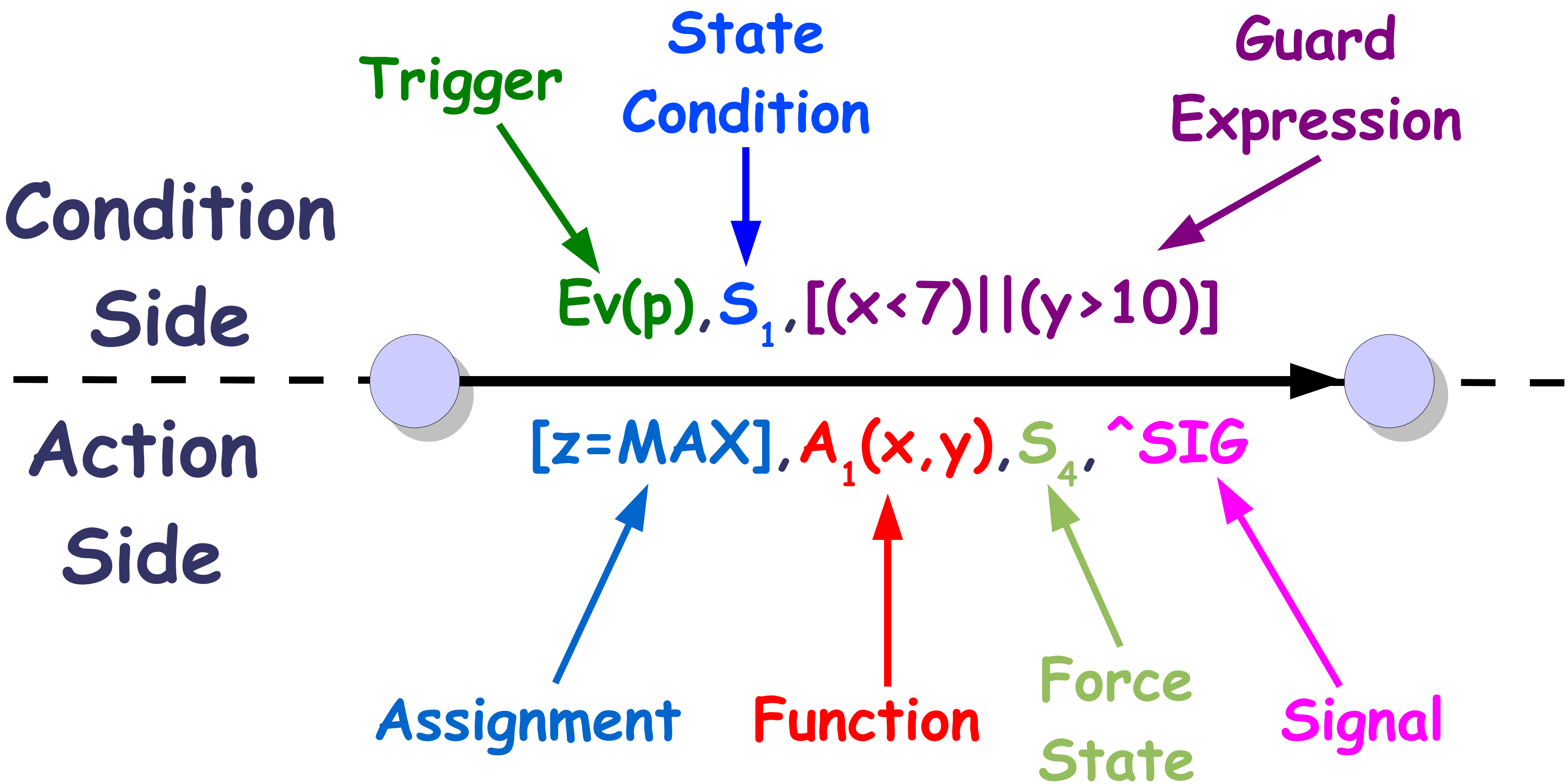
VisualSTATE Components

- 6 Components:
  - **Navigator**: Project Manager
  - **Designer**: Statecharts diagrams editor
  - **Verificator**: Model Verifier
  - **Validator**: Model Simulator and Tester
  - **Coder**: Code Generator
  - **Documenter**: Documentation Writer

VisualSTATE Features

- From the original Statecharts
  - Superstates
  - Mealy transitions (I/O)
  - Orthogonality
  - Default Entry states
  - (Deep) History states
- Added to the formalism
  - Variables (VS_INT, VS_UINT, VS_FLOAT, ...)
  - Signals (m, ^m)
  - C functions on transitions

VisualSTATE Transitions



Statemate 4.0

Statemate is development plate-form for object-oriented embedded systems applications.

* Enables complete systems design at the highest level
* Eliminates ambiguities common in written specifications
* Validates system behaviour early in the design process
* Generates an executable specification of the system
* Links designers, developers, and users for collaboration on a design, increasing the level of communication and cooperation
* Simplifies understanding of operation with animation of graphical models during code execution
* Accelerates the rapid prototyping process by providing C or Ada code for virtual and physical prototypes
* Produces production quality code generation from the design model
* Enables hardware/software co-specification
* Automatically generates complete, consistent, and formal documentation

Rhapsody 6.0

Rhapsody is an UML 2.0 based Model-Driven Development environment for systems and software engineering. It allows to specify systems and software design graphically, execute and validate the system as building it, and ultimately produce full production code from the model for the target system.

* Environment for Systems and Software Development
* Requirements Modelling
* Design-level Debugging on Target
* Directly Deployable C, C++, and Ada Code Generation
* Automatic Test Vector Generation

Questions ?

Next Week

- Concurrent Programming Basic Principles
  - Processes, Threads, Fibers
  - Atomicity
  - Synchronization
  - Mutual Exclusion
- Processes
  - Process Basics
  - System Calls (exec, fork, sleep, wait, ...)