Utrecht University

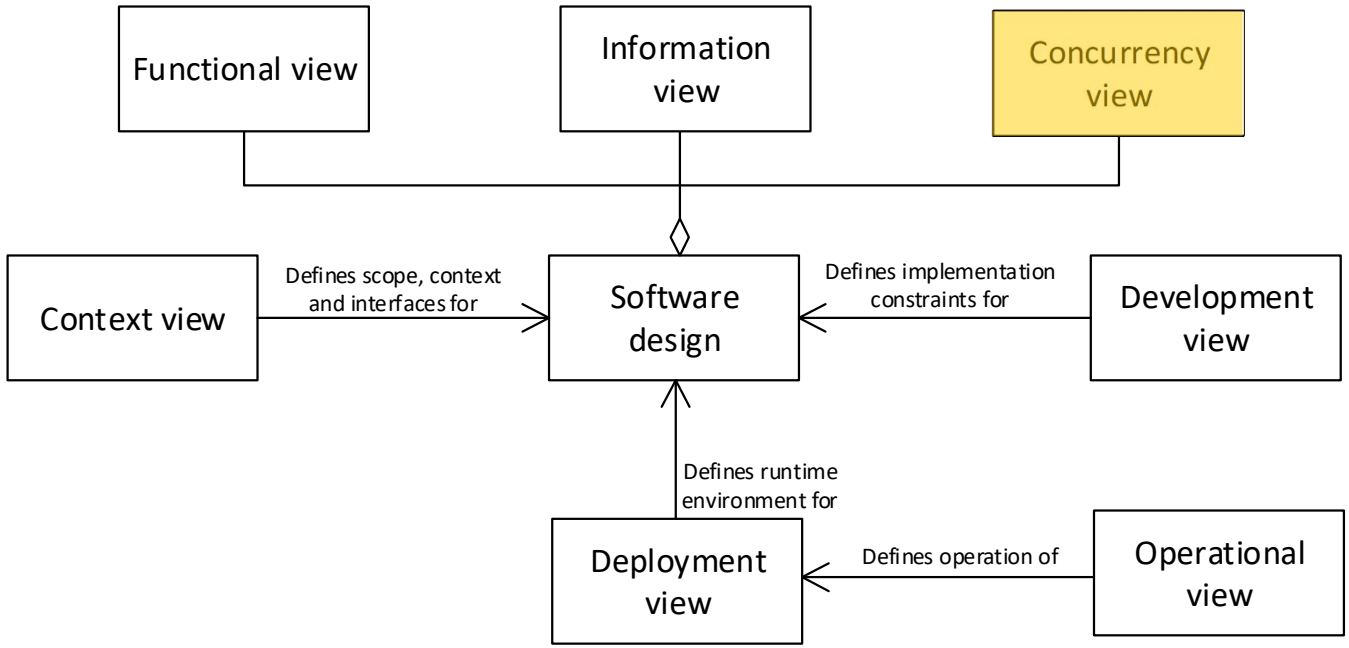
26-2-2024

Lecture 6: concurrency viewpoint

Jan Martijn van der Werf

Utrecht University

Viewpoint catalog

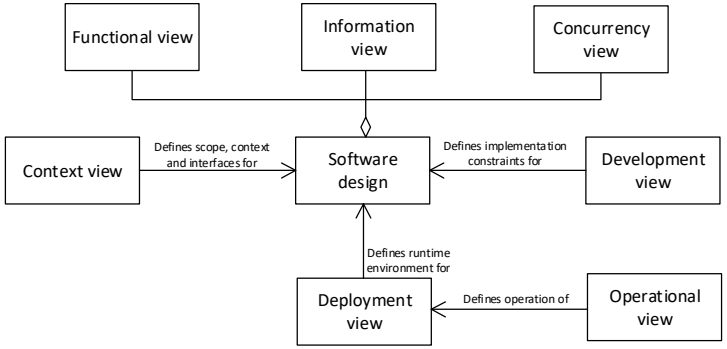


Viewpoint:

Collection of patterns, templates and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views

Utrecht University

Concurrency view



- Concurrency view:
  - **Describes the concurrency structure of the system and maps functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently and how this is coordinated and controlled**
- Concerns
  - **Task structure, mapping of functional elements to tasks,**
  - **Inter-process communication,**
  - **State management,**
  - **Synchronization and integrity,**
  - **Supporting scalability, task failure,**
  - **Startup and shutdown, re-entrancy**
- Models and views
  - **System level concurrency models**
  - **State models, protocol models**

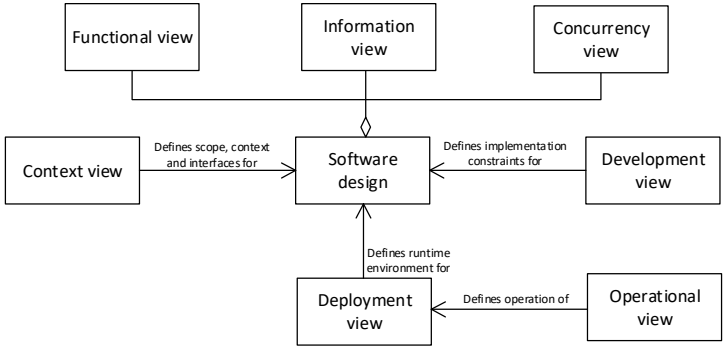
Utrecht University

About today's papers...

Utrecht University

Concurrency view (2)

Relation with Shaw & Garlan (1995)?



- Problems and pitfalls
  - **Modelling the wrong concurrency**
  - **Modelling the concurrency wrongly**
  - **Excessive complexity**
  - **Resource contention,**
  - **Deadlocks, livelocks**
  - **Race conditions**
- Applicability
  - **Systems with a number of concurrent threads of execution**
  - **Component-based systems with loosely coupled elements**
  - **Interorganizational systems**

We tend to illogically reason over concurrency!
It is genuinely difficult to grasp how systems are intertwined and communicate!

My position: semiformal models are insufficient to understand concurrency!

Utrecht University

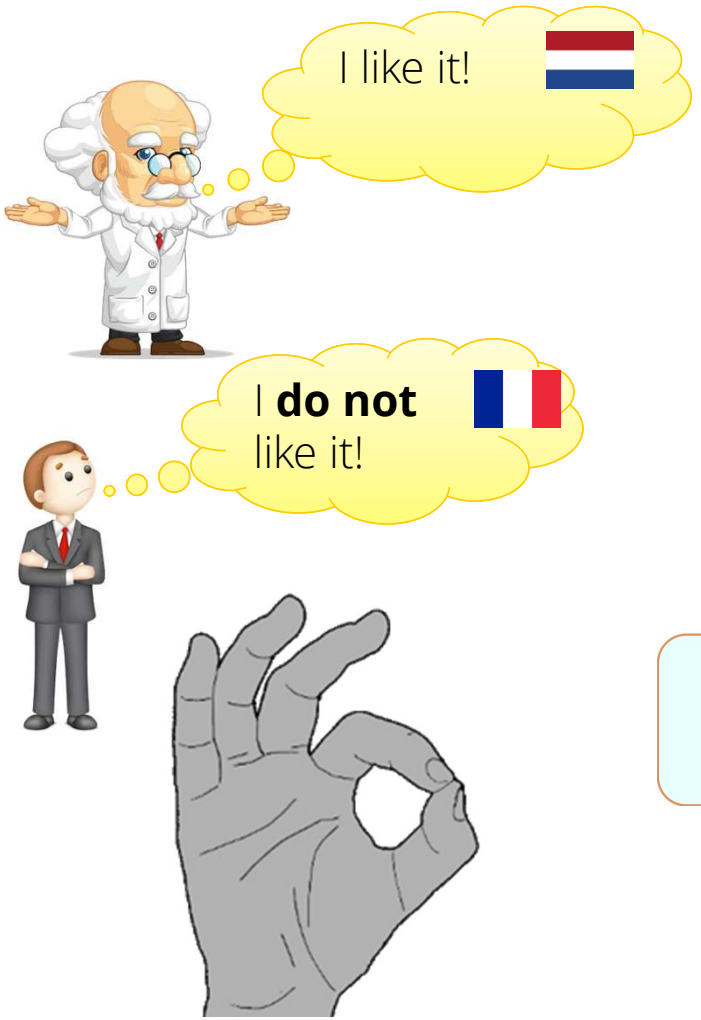
Models vs. pictures

What do you see?

Pictures: Everybody sees something different…

Utrecht University

Models

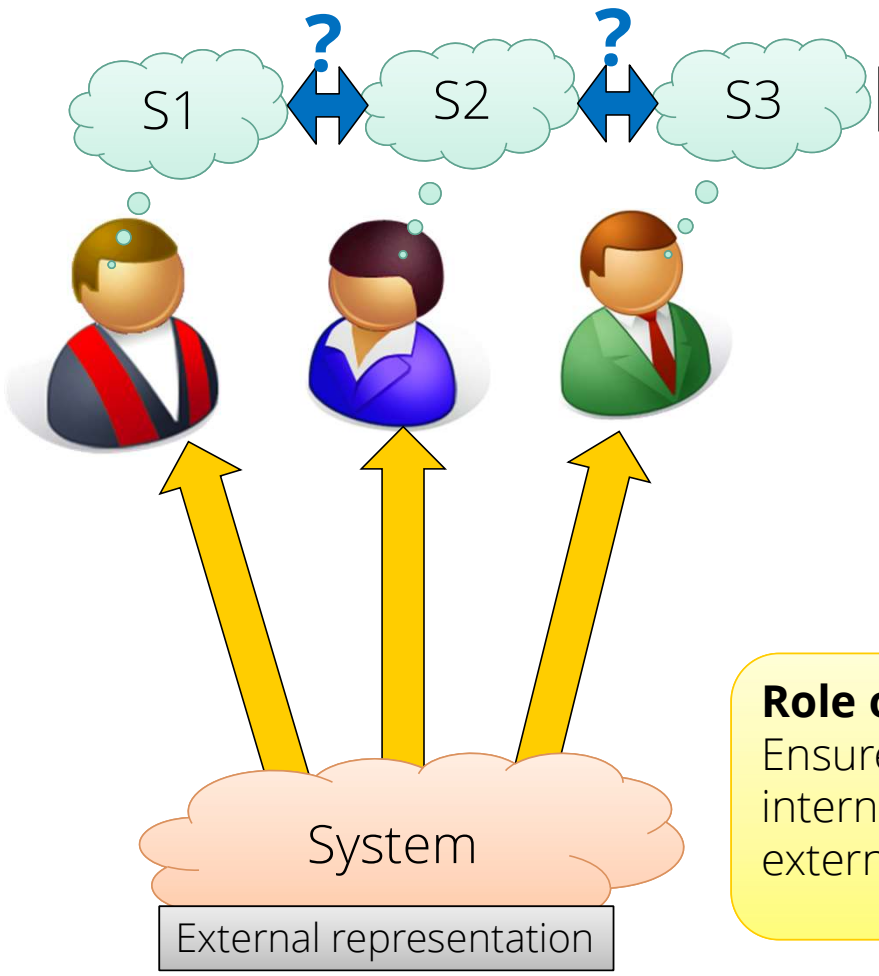


- Syntax:
  - **What elements are there?**
  - **How are the elements related to each other?**
- Notation:
  - **How do you denote the different elements?**
- Semantics
  - **What is the (mathematical) meaning of the model?**
- Pragmatics
  - **How do you use and create the models?**

- Intention
  - **What does the modeler want to express with their model?**

Utrecht University

Intention: ensure the right internal representation!



Internal representation

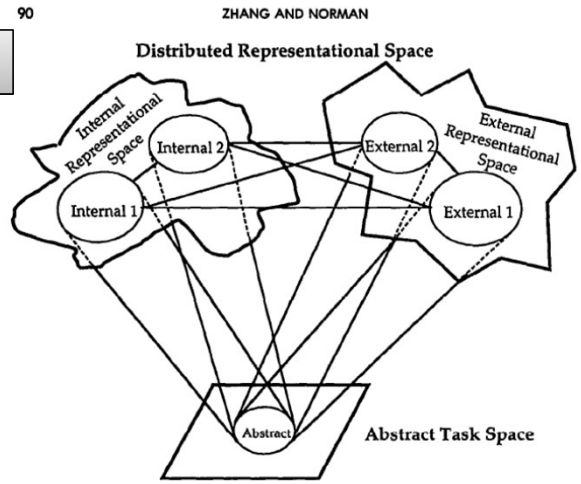
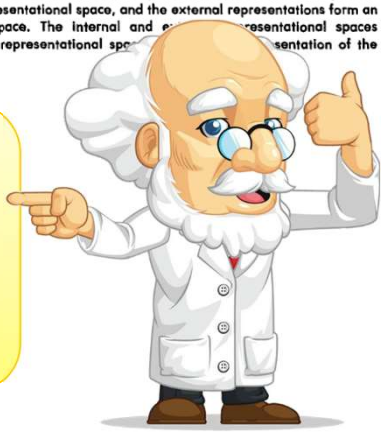


Figure 1. The theoretical framework of distributed representations. The internal representations form an internal representational space, and the external representations form an external representational space. The internal and e... resentational spaces together form a distributed representational spa... sentation of the abstract task space.

Role of Mathematics:
Ensures that everyone has the same internal representation of some external representation

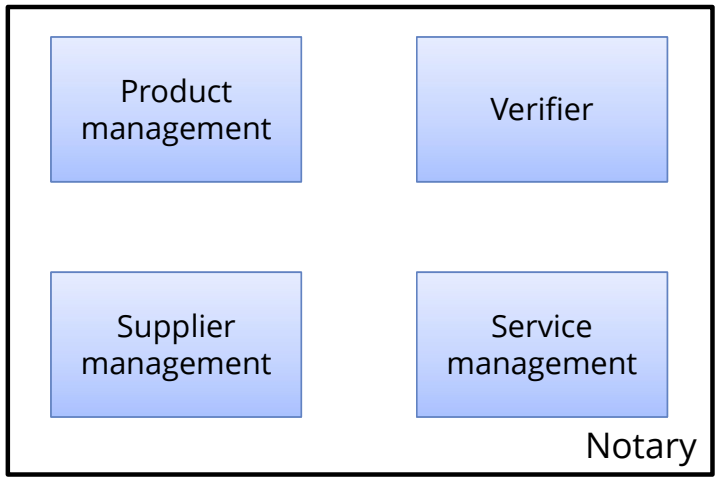


Utrecht University

Back to software architecture...

Utrecht University

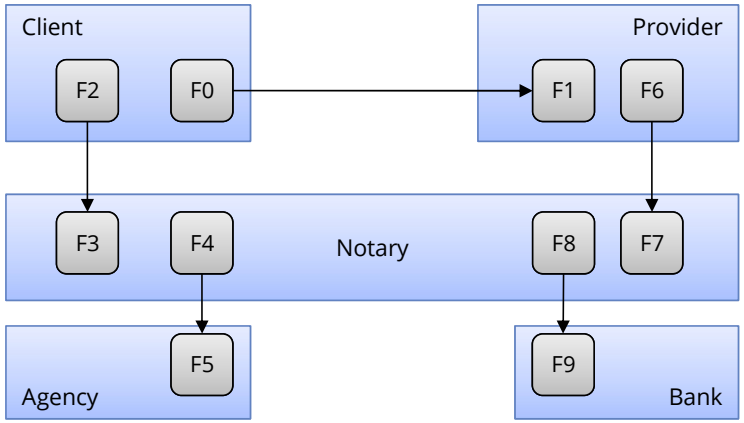
Create a functional architecture for this context:



- Notary is a platform with providers and clients.
- Providers deliver services to clients.
- Clients directly approach a provider.
- The notary is a trusted party.
- Clients need to be checked by the Notary
- Clients send their approval to a preferred provider
- Payment is only done once the client signed the bill
- Multiple payments per service possible
- Payment is done by the notary
- The notary can use several agencies to get more info upon a client's request

Utrecht University

Create a functional architecture for this context:

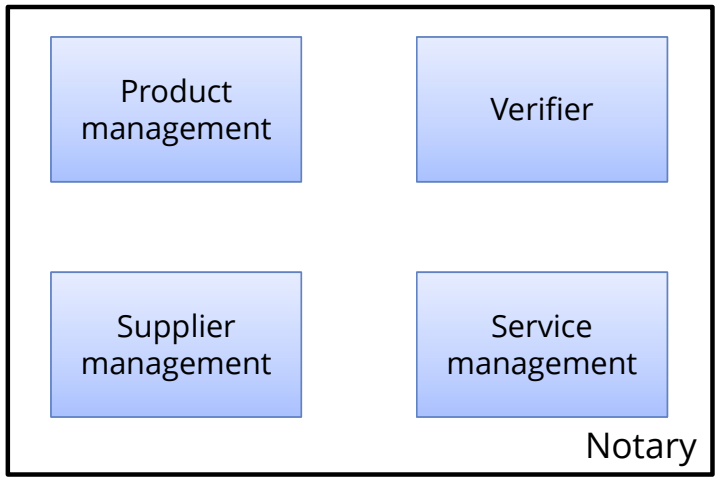


- Notary is a platform with providers and clients.
- Providers deliver services to clients.
- Clients directly approach a provider.
- The notary is a trusted party.
- Clients need to be checked by the Notary
- Clients send their approval to a preferred provider
- Payment is only done once the client signed the bill
- Multiple payments per service possible
- Payment is done by the notary
- The notary can use several agencies to get more info upon a client's request

Utrecht University

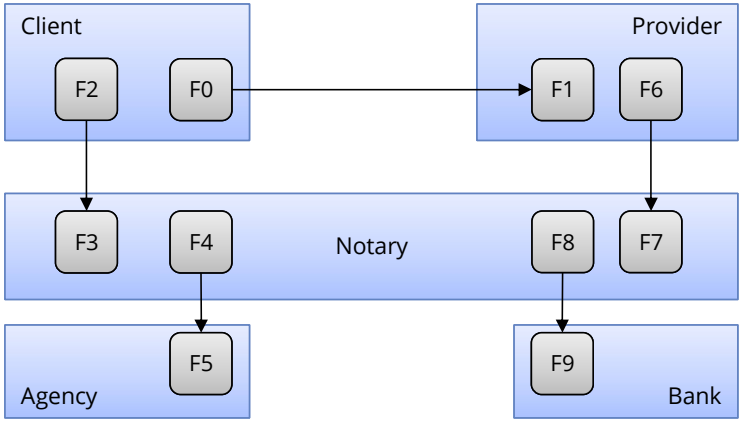
Create a functional architecture for this context:

Functional architecture



mapping

Logical architecture



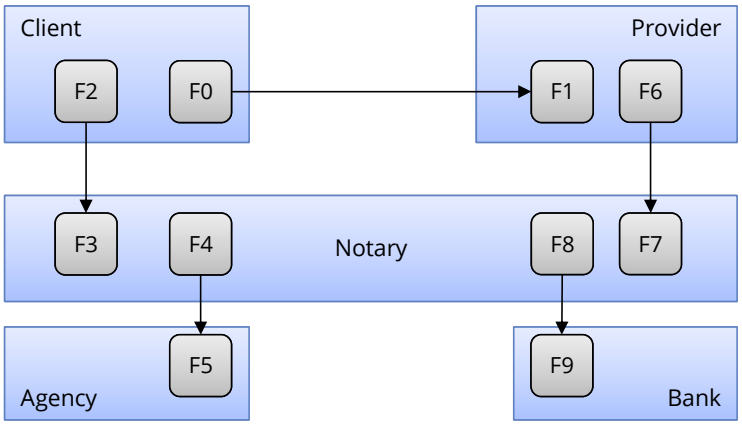
From a “usage” perspective

From a “resource” perspective

Here, you already see the envisioned “services”!

Utrecht University

Logical architecture model

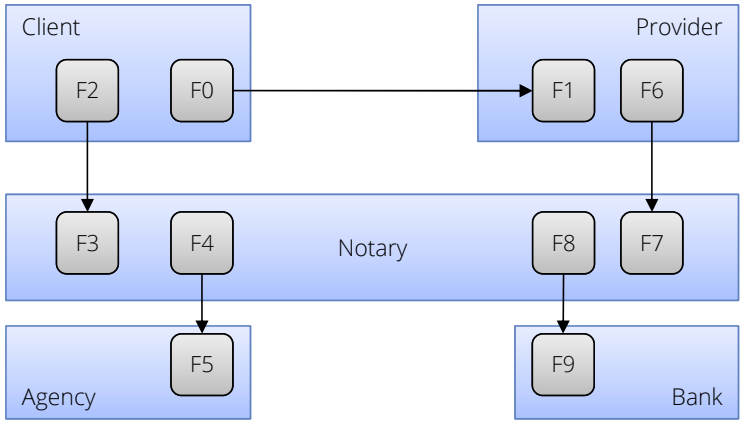


- Hierarchical structure:
  Atomic and composite containers
- Atomic Container
  Groups a coherent set of functions
- Composite Containter
  Only contains containers
- Function calls:
  Always between functions
  Arrow: function "converses" with another function

Remember: "just" a lines-and-boxes diagram

Utrecht University

From logical model to concurrency…

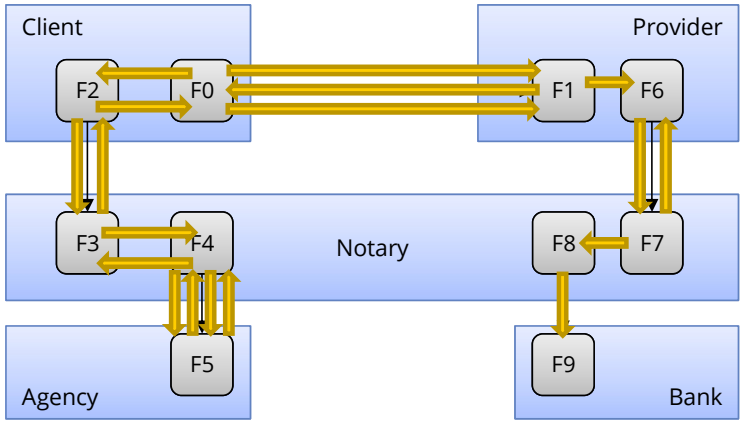


Each system is running concurrently with all the other systems!

- Notary is a platform with providers and clients.
- Providers deliver services to clients.
- Clients directly approach a provider.
- The notary is a trusted party.
- Clients need to be checked by the Notary
- Clients send their approval to a preferred provider
- Payment is only done once the client signed the bill
- Multiple payments per service possible
- Payment is done by the notary
- The notary can use several agencies to get more info upon a client's request

Utrecht University

Logical model and scenarios



- Scenario is a sequence of function calls
- Formal definition:

  Given a logical model $(C, F, h, \rightarrow)$**,**
  a scenario is a partial order over the function calls,
  i.e., $\sigma \in (\rightarrow)^*$ **such that:**

  Functions can only start if being called before:

  $$\forall 1 < i < |\sigma| : \left(\exists 1 < j < i : \pi_3(\sigma(j)) = \pi_1(\sigma(i))\right)$$

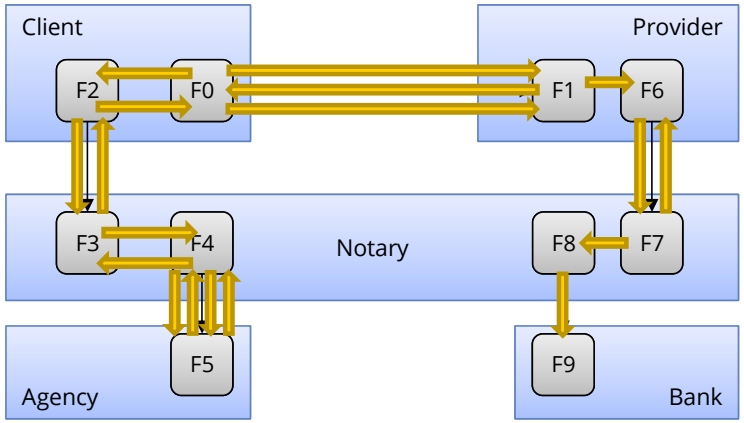
Key:
F0: Request service
F1: Handle service request
F2: Request approval
F3: Receive approval request
F4: Validate client
F5: Do credibility check
F6: Request payment
F7: Check payment request
F8: Send payment
F9: Make payment

Given a set A, a sequence of length $n \in \mathbb{N}$ is a function $\sigma : \{1..n\} \rightarrow A$

- We write $\sigma = \langle a_1, \ldots, a_n \rangle$ if $\sigma(i) = a_i$ for all $1 \leq i \leq n$.
- We denote its length by $|\sigma|$
- If $n = 0$, we call it the empty sequence, and denote it with ϵ
- The set of all finite sequences over A is denoted by A^*

Utrecht University

Logical model and scenarios



- Scenario is a sequence of function calls
- Formal definition:

  Given a logical model $(C, F, h, \rightarrow)$**,**
  a scenario is a partial order over the function calls,
  i.e., $\sigma \in (\rightarrow)^*$ **such that:**

  Functions can only start if being called before:

  $$\forall 1 < i < |\sigma| : \left(\exists 1 < j < i : \pi_3(\sigma(j)) = \pi_1(\sigma(i))\right)$$

Problem: scenarios can be conflicting!

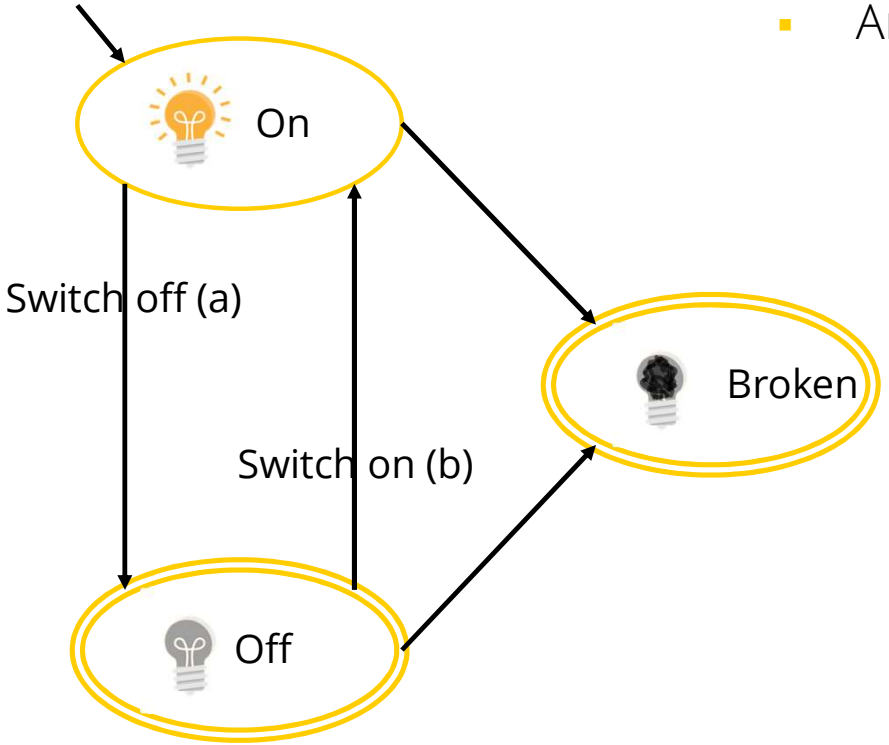
Key:
F0: Request service
F1: Handle service request
F2: Request approval
F3: Receive approval request
F4: Validate client
F5: Do credibility check
F6: Request payment
F7: Check payment request
F8: Send payment
F9: Make payment

Given a set A, a sequence of length $n \in \mathbb{N}$ is a function $\sigma : \{1..n\} \rightarrow A$

- We write $\sigma = \langle a_1, \ldots, a_n \rangle$ if $\sigma(i) = a_i$ for all $1 \leq i \leq n$.
- We denote its length by $|\sigma|$
- If $n = 0$, we call it the empty sequence, and denote it with ϵ
- The set of all finite sequences over A is denoted by A^*

Utrecht University

Generalizing scenarios: Labeled transition systems

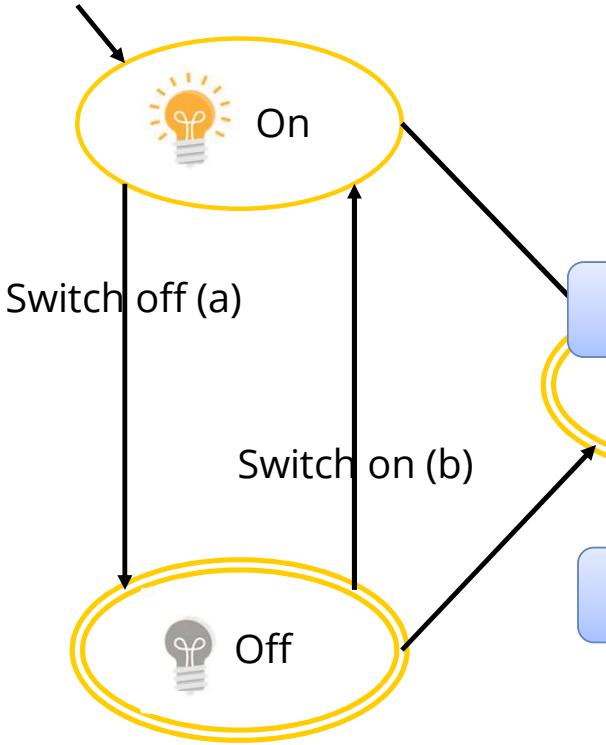


- An automaton is a 5-tuple $(Q, A, \rightarrow, s_0, \Omega)$
  - Q **is a set of states**
  - A **is a set of observable action labels**
  - **Flow relation** $\rightarrow \subseteq Q \times (A \cup \{\tau\}) \times Q$ **with**
    - τ **the unobservable action, so** $\tau \notin A$
  - $s_0 \in Q$ **is the *initial state***
  - $\Omega \subseteq Q$ **is the set of *final (accepting) states***

Q = { On, Off, Broken }
A = { a, b },
$\rightarrow$ = { (On, a, Off), (Off, b, On),
(Off, τ, Broken), (On, τ, Broken) }
s_0 = On
Ω = { Off, Broken }

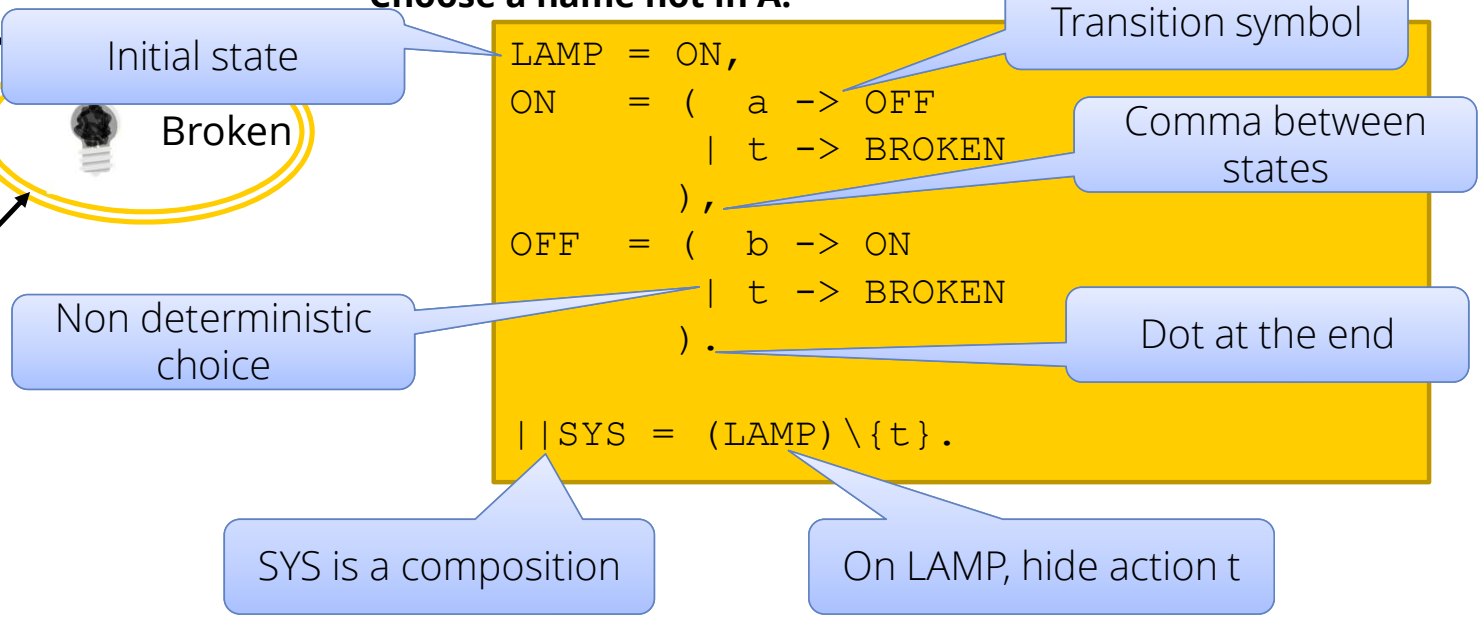
Utrecht University

Modeling Label Transition Systems in LTSA



- States in capital
- Actions in small letters
- List all transitions per state:

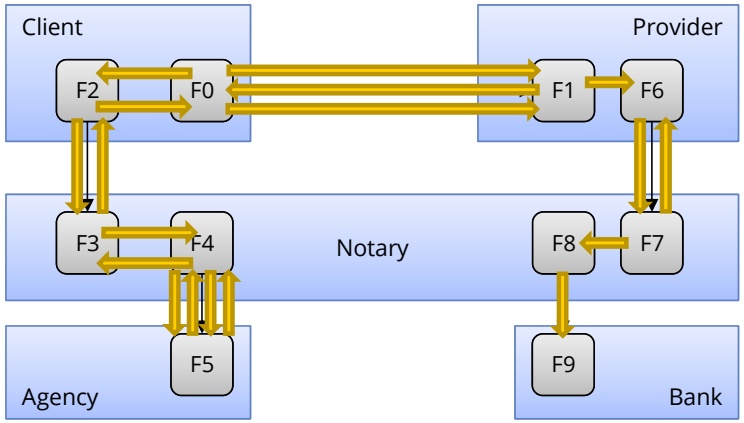
Note: silent action cannot be implemented directly!
Choose a name not in A.




```
LAMP = ON,
ON   = (  a -> OFF
        | t -> BROKEN
       ),
OFF  = (  b -> ON
        | t -> BROKEN
       ).

||SYS = (LAMP)\{t}.
```

Utrecht University

Define an LTS for F0 in LTSA

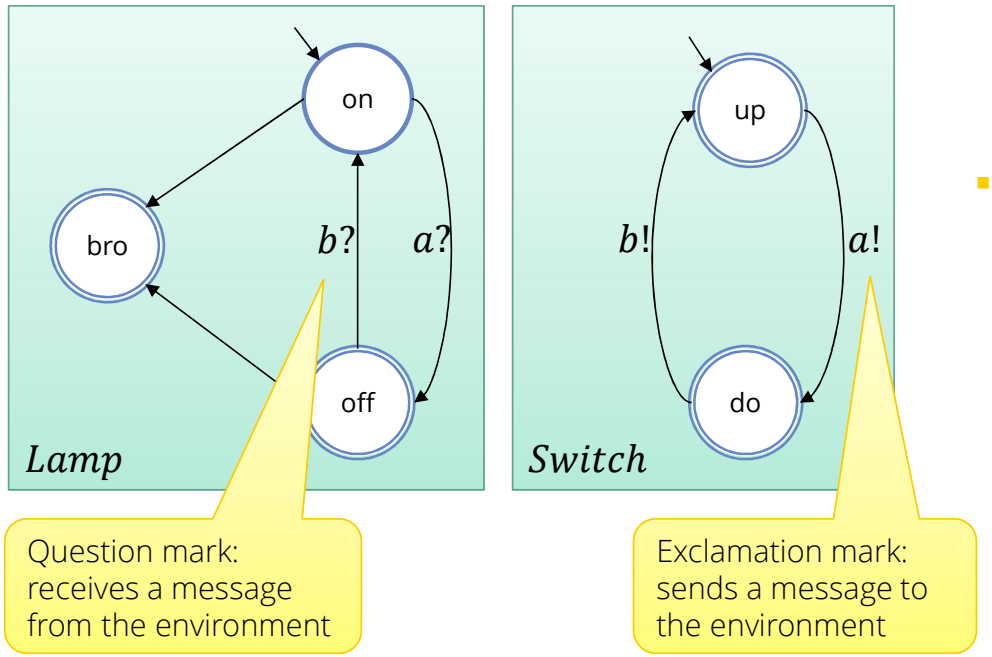


- The client asks for permission. Upon receiving the permission, (s)he asks for a service of the provider, and either accepts or denies the offer received by the provider. Upon accepting, the provider offers the service, and regularly sends an invoice, which needs to be signed and returned by the client. Once denied, either (s)he stops, or (s)he asks for a better offer.
  In case the client does not receive permission, (s)he tries again.

Key:
F0: Request service
F1: Handle service request
F2: Request approval
F3: Receive approval request
F4: Validate client
F5: Do credibility check
F6: Request payment
F7: Check payment request
F8: Send payment
F9: Make payment

Utrecht University

Interface automata
(Also referred to as Finite State Processes)

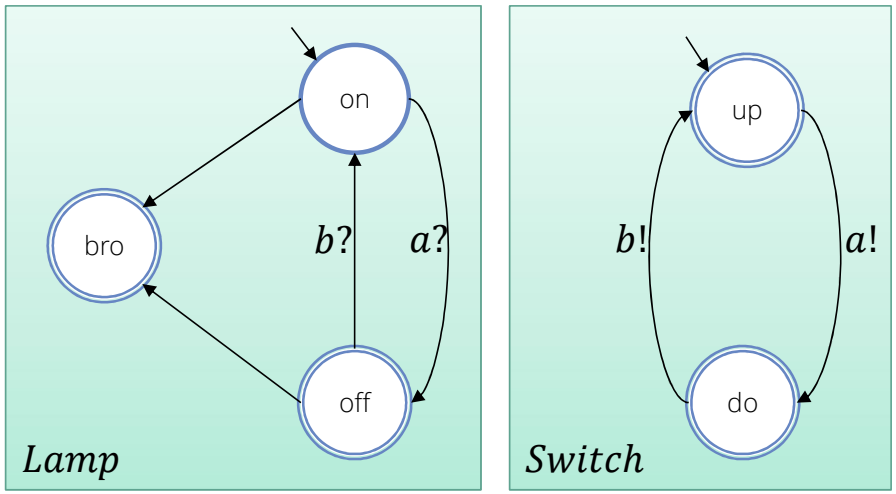


- Actions of LTS divided in three sets:
  - **Internal action**
  - **Sending action: denoted with an exclamation mark**
  - **Receiving action: denoted with a question mark**
- Synchronization:
  - **Sync is a function: if a! and a? are both enabled in the two LTSs, they synchronize in action a**

Utrecht University

Composing interface automata

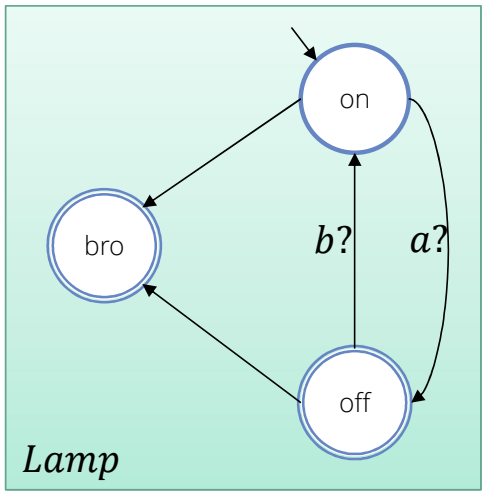
Composition of interface automata

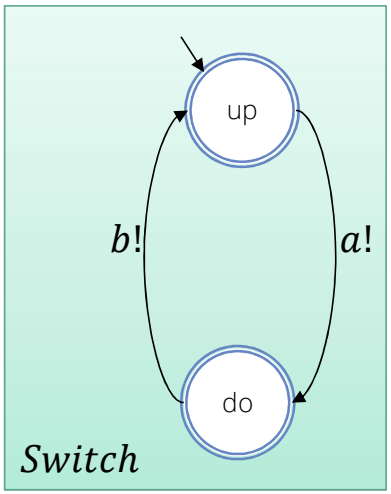


Utrecht University

Composing interface automata

Composition of interface automata





Let $L_1 = (S_1, A_1, \rightarrow_1, s_0^1, \Omega_1)$ **and** $L_2 = (S_2, A_2, \rightarrow_2, s_0^2, \Omega_2)$**,**
Let $R \subseteq A$ **be a set of actions to synchronize on**
i.e. $\forall a \in R: (a? \in S_1 \wedge a? \in S_2) \vee (a! \in S_1 \wedge a? \in S_2)$
Define $R^+ = \bigcup_{a \in R} \{a!, a?\}$

Then: $L_1 \otimes_R L_2 = (\mathrm{S}, \mathrm{A}, \rightarrow, \mathrm{s}_0, \Omega)$ **with**

$$S = S_1 \times S_2$$

$$A = (R \cup A_1 \cup A_2) \setminus \mathrm{R}^+$$

$$\rightarrow = \left\{ \left((s_1, s_2), a, (s_1', s_2)\right) \middle| (s_1, a, s_1') \in \rightarrow_1 \wedge\, a \in A_1 \setminus R^+ \right\}$$
$$\cup \left\{ \left((s_1, s_2), a, (s_1, s_2')\right) \middle| (s_2, a, s_2') \in \rightarrow_2 \wedge\, a \in A_2 \backslash \mathrm{R}^+ \right\}$$
$$\cup \left\{ \left((s_1, s_2), a, (s_1', s'_2)\right) \middle| (s_1, a?, s_2) \in \rightarrow_1, (s_2, a!, s_2') \in \rightarrow_2, a \in \mathrm{R} \right\}$$
$$\cup \left\{ \left((s_1, s_2), a, (s_1', s_2')\right) \middle| (s_1, a!, s_2) \in \rightarrow_1, (s_2, a?, s_2') \in \rightarrow_2, a \in R \right\}$$

$$s_0 = (s_0^1, s_0^2)$$

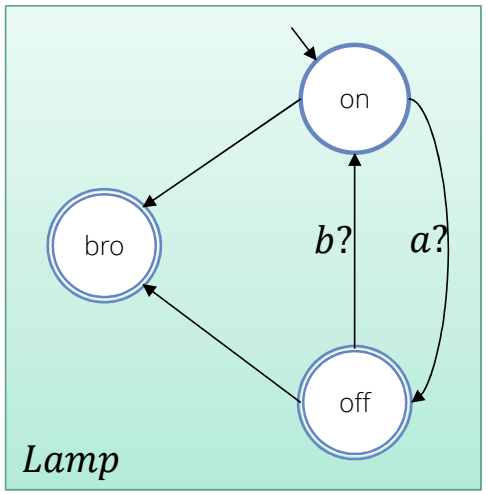
$$\Omega = \Omega_1 \times \Omega_2$$

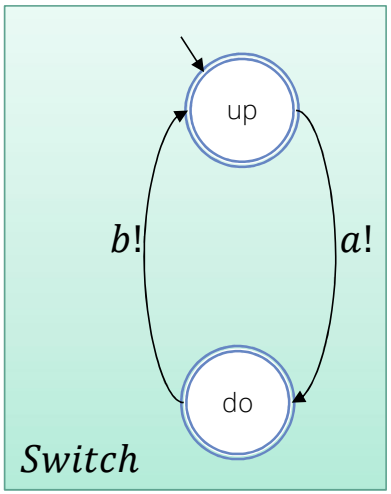
Notice: this is interleaving semantics!

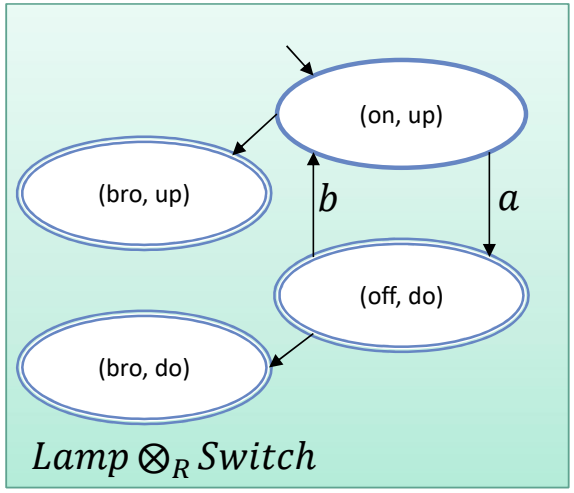
ReqEng: In FSP: $(a \rightarrow S) \parallel (b \rightarrow T) \equiv \left(a \rightarrow \left(S' \parallel (b \rightarrow T)\right) \middle| b \rightarrow \left((a \rightarrow S) \parallel T\right)\right)$

Utrecht University

Composing interface automata



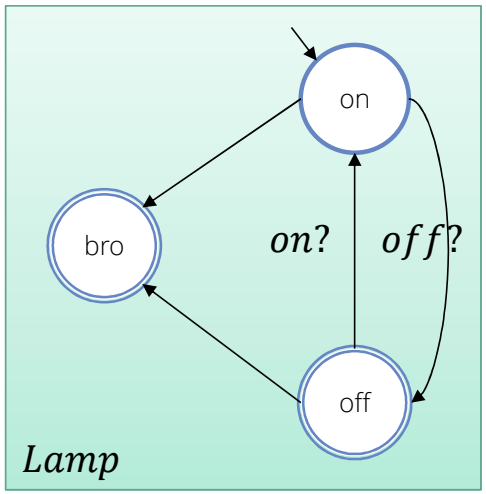


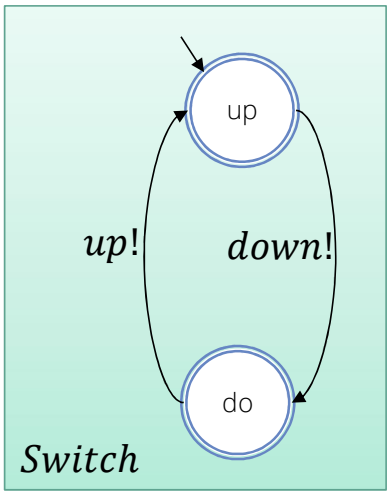


$R = \{a, b\}$

Utrecht University

Composing interface automata in LTSA



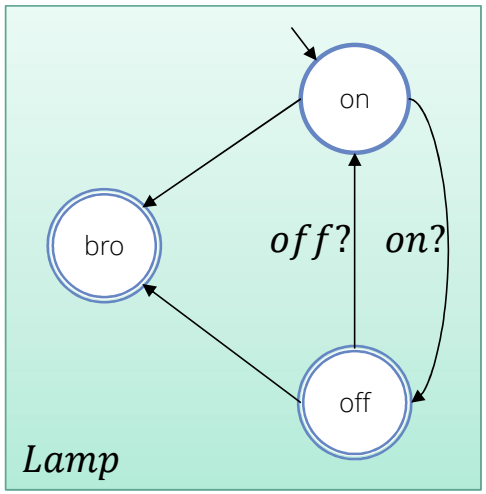


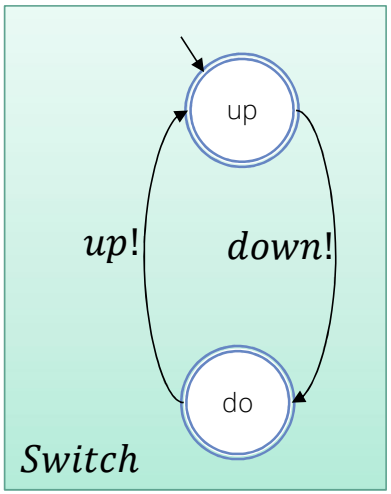
- Define each system separately
  - **Prefix a send action with an s**
  - **Prefix a receive action with an r**

```
LAMP = ON,
ON   = (  rOff -> OFF | t -> BROKEN),
OFF  = (  rOn -> ON   | t -> BROKEN).

SWITCH = UP,
UP     = (sDown -> DOWN),
DOWN   = (sUp   -> UP).
```

Utrecht University

Composing interface automata in LTSA





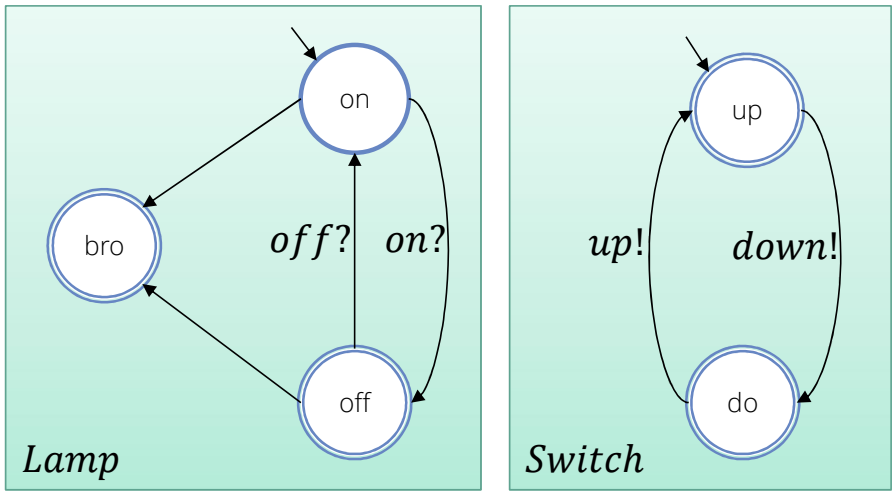
- Define each system separately
  - **Prefix a send action with an s**
  - **Prefix a receive action with an r**
- Compose the systems (||)

```
LAMP = ON,
ON   = (  rOff -> OFF | t -> BROKEN),
OFF  = (  rOn -> ON   | t -> BROKEN).

SWITCH = UP,
UP     = (sDown -> DOWN),
DOWN   = (sUp   -> UP).

||SYS = ( LAMP || SWITCH ).
```

SYS is a composition of lamp with switch

Utrecht University

Composing interface automata in LTSA



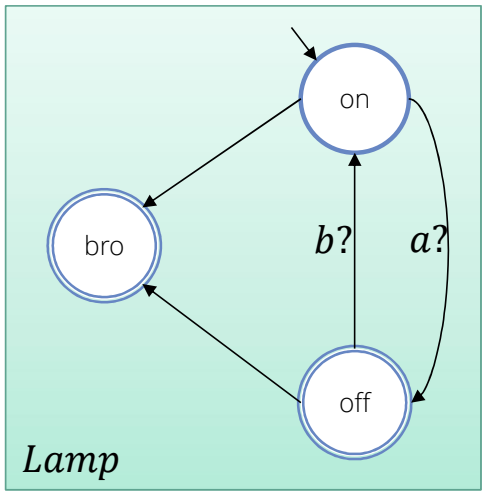
- Define each system separately
  - **Prefix a send action with an s**
  - **Prefix a receive action with an r**
- Compose the systems (||)
- Specify synchronization relation with relabelling:
  - **rename the actions to create the synchronization: /{sSendAction/rReceiveAction}**

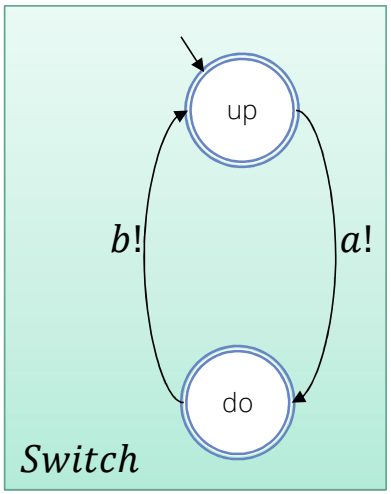
```
LAMP = ON,
ON   = (  rOff -> OFF | t -> BROKEN),
OFF  = (  rOn -> ON   | t -> BROKEN).

SWITCH = UP,
UP     = (sDown -> DOWN),
DOWN   = (sUp   -> UP).

||SYS = ( LAMP || SWITCH )/{sDown/rOn, sUp/rOff}.
```

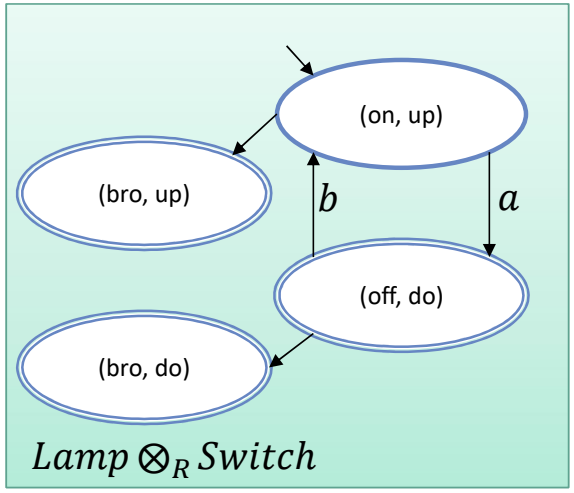
/ denotes a renaming action!
s/r renames an r into s

Utrecht University

Composing interface automata







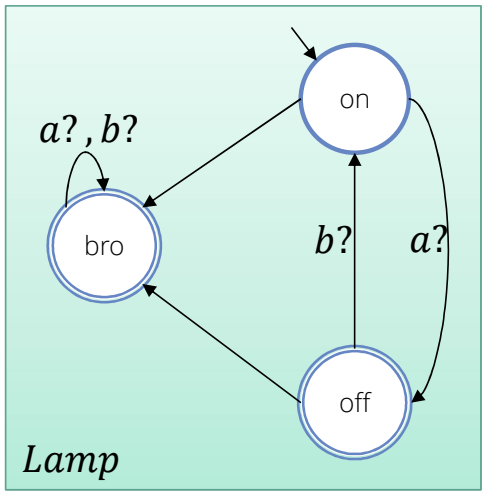
$R = \{a, b\}$

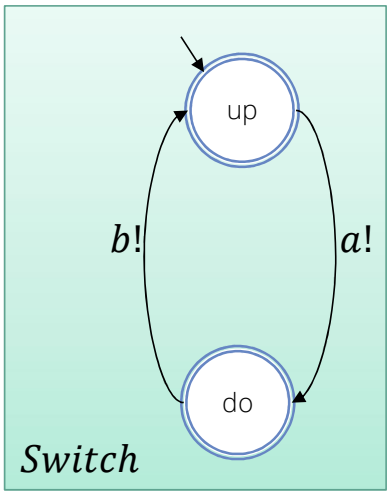
Notice: once the lamp is broken, it is not possible to use the switch anymore! This is due to the fact that the actions of the switch need to be synchronized with the lamp. As it does not react on any messages anymore, the system is in deadlock.

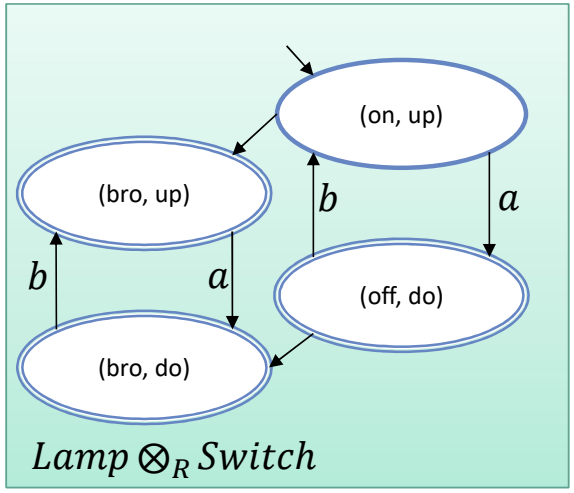
Deadlock: a state with no outgoing actions. Possible repair: add a "self-loop" on state "bro" labeled $a?$ and $b?$

Utrecht University

Composing interface automata







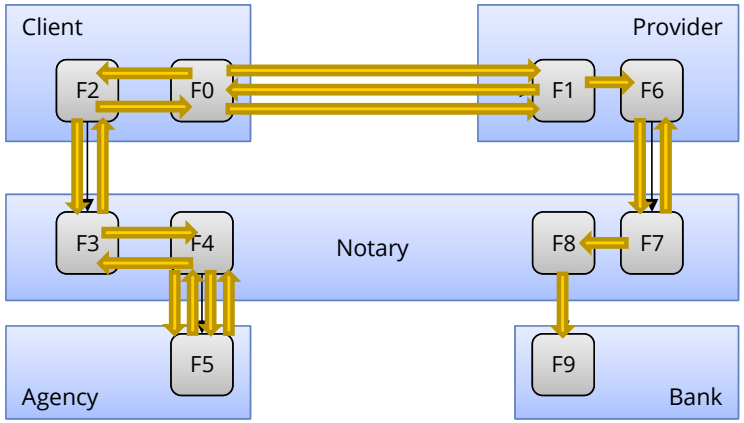
$R = \{a, b\}$

Notice: once the lamp is broken, it is not possible to use the switch anymore! This is due to the fact that the actions of the switch need to be synchronized with the lamp. As it does not react on any messages anymore, the system is in deadlock.

Deadlock: a state with no outgoing actions. Possible repair: add a "self-loop" on state "bro" labeled $a?$ and $b?$

Utrecht University

Define the following LTS for F1 in LTSA, and compose it with F0



- Upon receiving a request for an offer from a client, the provider creates and sends an offer. If the offer is rejected, the provider will create a new offer, and send it.
- Once the offer is accepted, the service is provided, and on a regular basis an invoice is sent, which is signed and returned by the client. This signed bill is then passed to the administration.

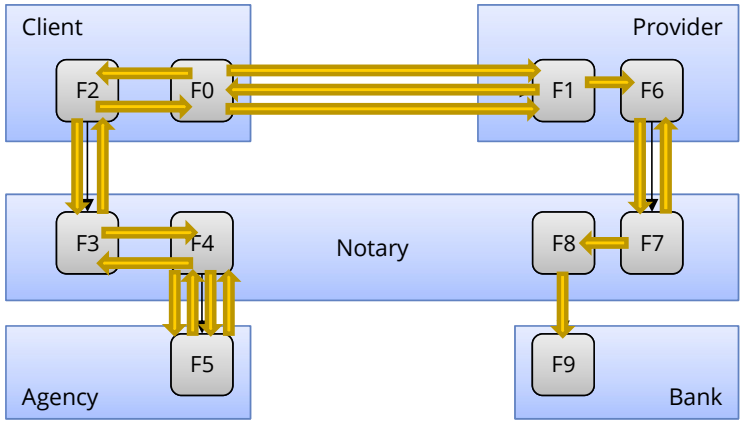
Key:
F0: Request service
F1: Handle service request
F2: Request approval
F3: Receive approval request
F4: Validate client
F5: Do credibility check
F6: Request payment
F7: Check payment request
F8: Send payment
F9: Make payment

What are your observations?

Designing the individual systems is “easy”. Their composition is exponentially more difficult!

Utrecht University

Communication mechanisms



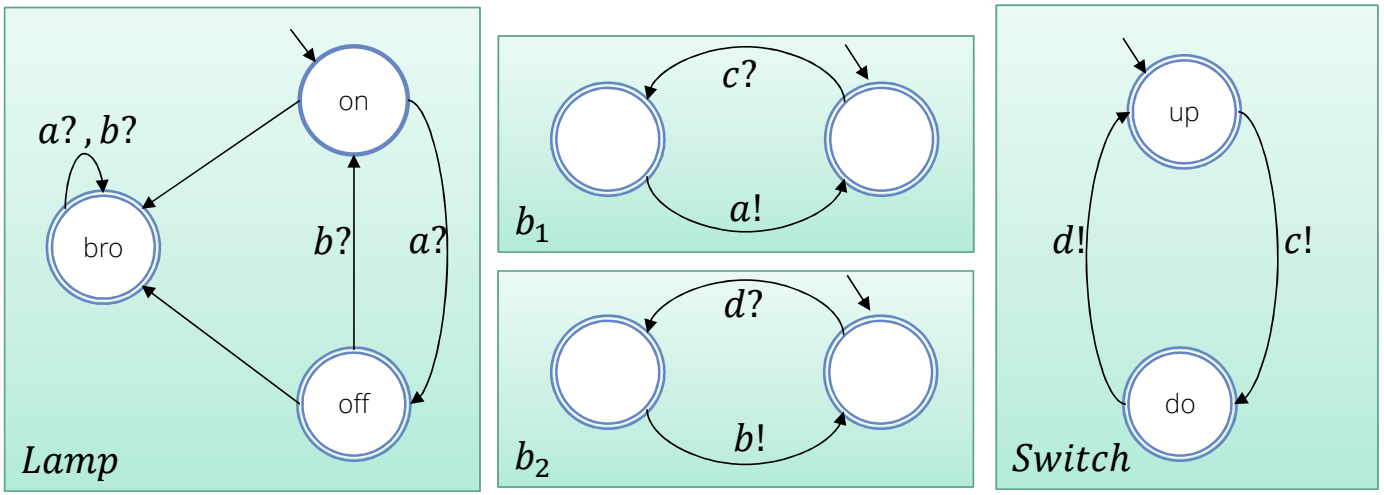
- Synchronous communication
  You have to know in which state the other is!
- Asynchronous communication
  Send messages to the other, the state of the other is unknown!

FSP, Interface automata only allow for synchronous communication!

Key:
F0: Request service
F1: Handle service request
F2: Request approval
F3: Receive approval request
F4: Validate client
F5: Do credibility check
F6: Request payment
F7: Check payment request
F8: Send payment
F9: Make payment

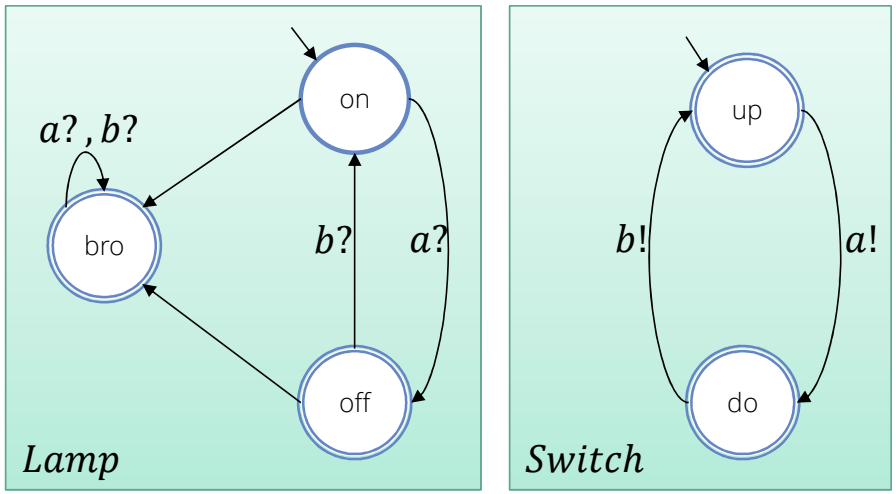
Utrecht University

Simulating asynchronous behavior with buffers



Utrecht University

I/O automata



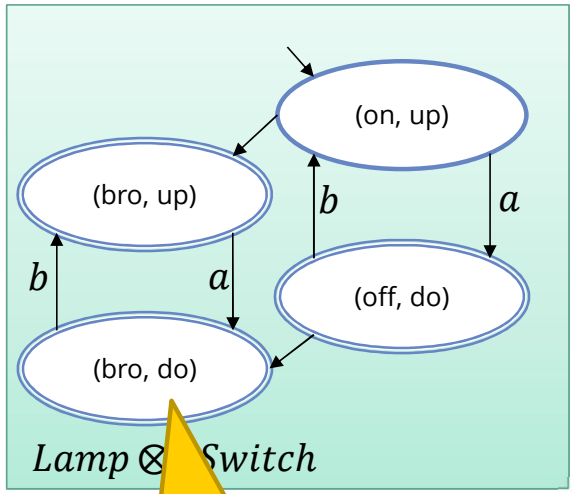
- Message queue to delay messages
  - **Queue per automaton / global queue**
  - **Messages are handled in order.**
  - **Capacity of the queue?**

Utrecht University

Petri nets

Utrecht University

Labeled Transition Systems have global states!



- Instead of global states: distributed states!

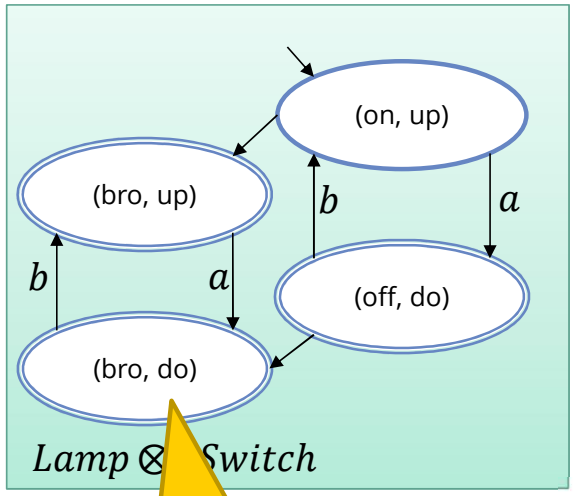
- Transitions: state changes are local
  check locally if a transition can fire:
  transition firing changes the state locally

State resembles:
- State of switch
- State of light

How can we make advantage of this split state?

Utrecht University

Labeled Transition Systems have global states!



State resembles:
- State of switch
- State of light

How can we make advantage of this split state?

- Instead of global states: distributed states!

Places: indicate a condition of the system
Tokens: represents elementary entities of interest
State: the configuration of tokens over the different places

- Transitions: state changes are local

check locally if a transition can fire:
transition firing changes the state locally

Utrecht University

Petri nets: an informal introduction

- Instead of global states: distributed states!

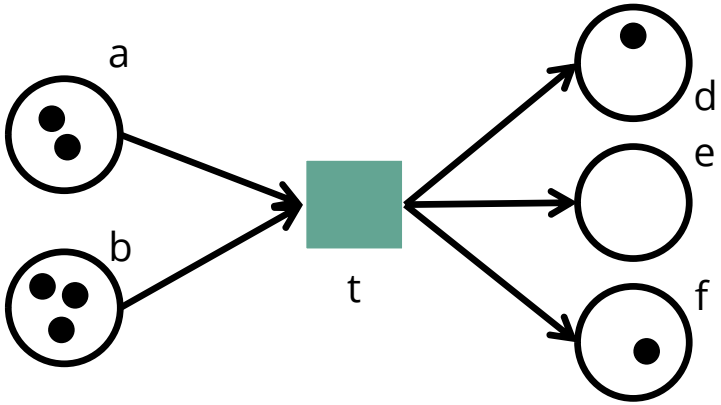
Places: indicate a condition of the system
Tokens: represents elementary entities of interest
State: the configuration of tokens over the different places

- Transitions: state changes are local

check locally if a transition can fire:
transition firing changes the state locally

Utrecht University

Petri nets: an informal introduction



- Instead of global states: distributed states!

Places: indicate a condition of the system
Tokens: represents elementary entities of interest
State: the configuration of tokens over the different places

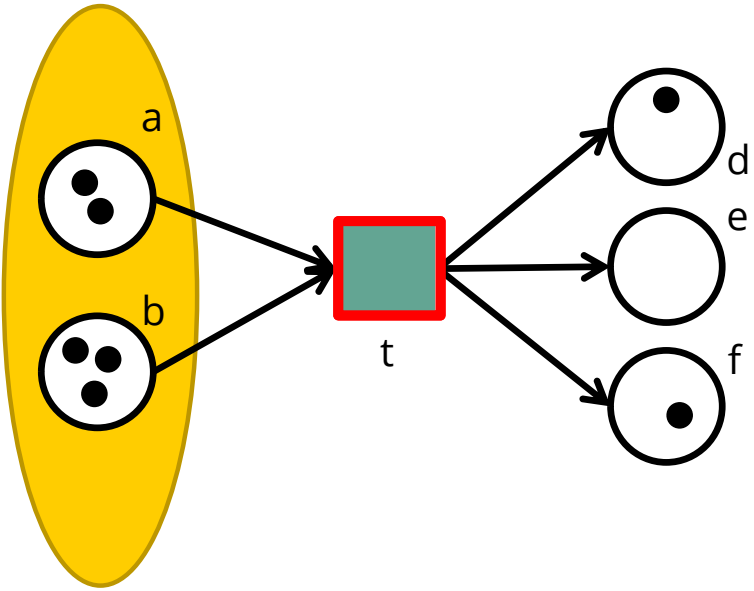
- Transitions: state changes are local

check locally if a transition can fire:
transition firing changes the state locally

Arcs are always from a place to a transition, or
From a transition to a place

Utrecht University

Petri nets: an informal introduction



Input places of transition t

- Instead of global states: distributed states!

Places: indicate a condition of the system
Tokens: represents elementary entities of interest
Marking: configuration of tokens over the different places

- Transitions: state changes are local

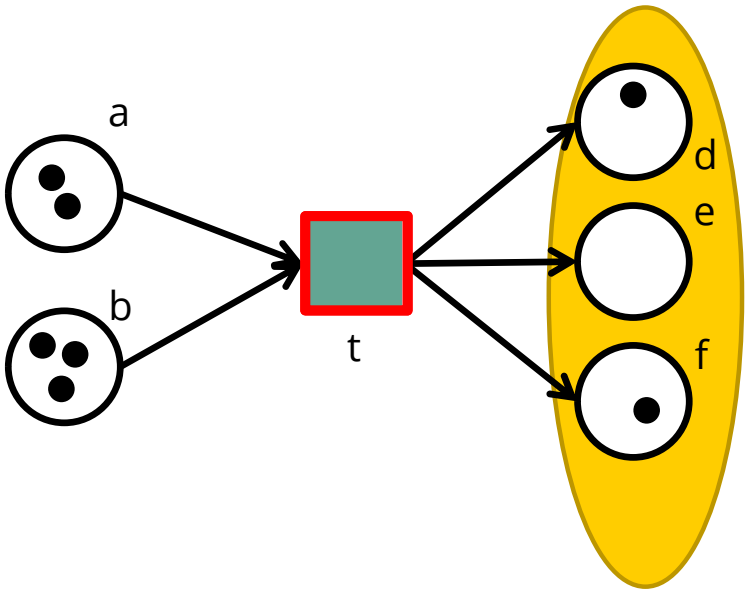
Input places: check locally if a transition can fire:
Output places: transition firing changes the state locally

- Transition enabled:

all input places have at least 1 token

Utrecht University

Petri nets: an informal introduction



output places of transition t

- Instead of global states: distributed states!

Places: indicate a condition of the system
Tokens: represent elementary entities of interest
Marking: configuration of tokens over the different places

- Transitions: state changes are local

Input places: check locally if a transition can fire:
Output places: transition firing changes the state locally

- Transition enabled:

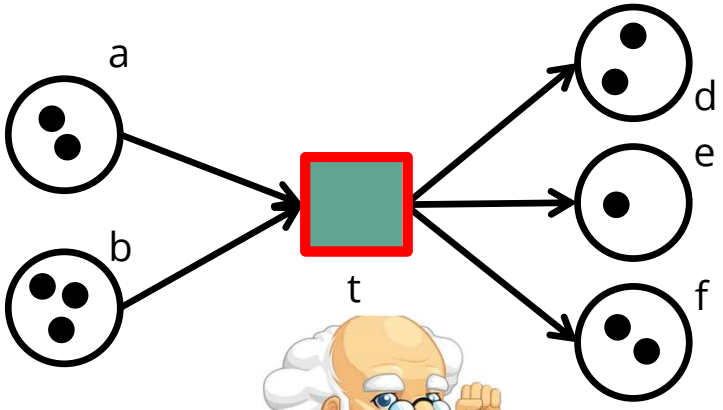
all input places have at least 1 token

- Transition firing (executing the transition):

Consumes 1 token from each input place
Produces 1 token in each output place

Utrecht University

Petri nets: an informal introduction



- Instead of global states: distributed states!

Places: indicate a condition of the system
Tokens: represent elementary entities of interest
Marking: configuration of tokens over the different places

- Transitions: state changes are local

Input places: check locally if a transition can fire:
Output places: transition firing changes the state locally

- Transition enabled:

all input places have at least 1 token

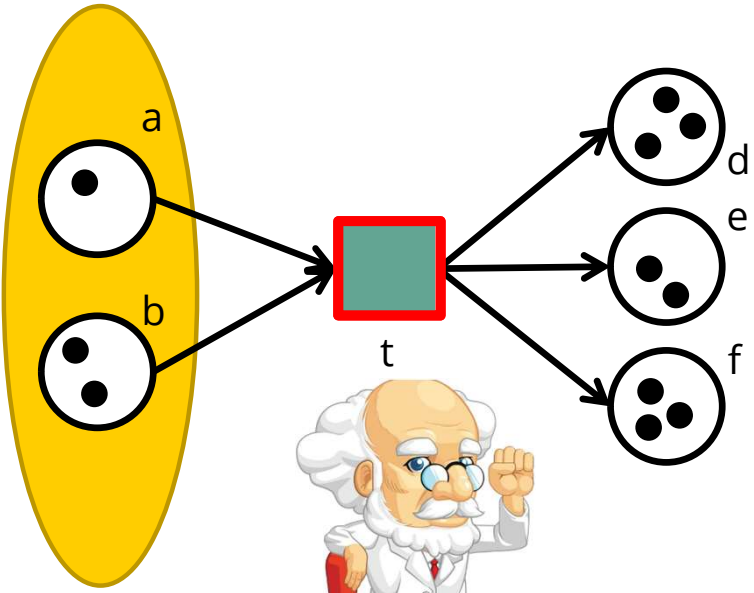
- Transition firing (executing the transition):

Consume 1 token from each input place
Produce 1 token in each output place

There is no law of token preservation!

Utrecht University

Petri nets: an informal introduction



- Instead of global states: distributed states!

Places: indicate a condition of the system
Tokens: represent elementary entities of interest
Marking: configuration of tokens over the different places

- Transitions: state changes are local

Input places: check locally if a transition can fire:
Output places: transition firing changes the state locally

- Transition enabled:

  all input places have at least 1 token

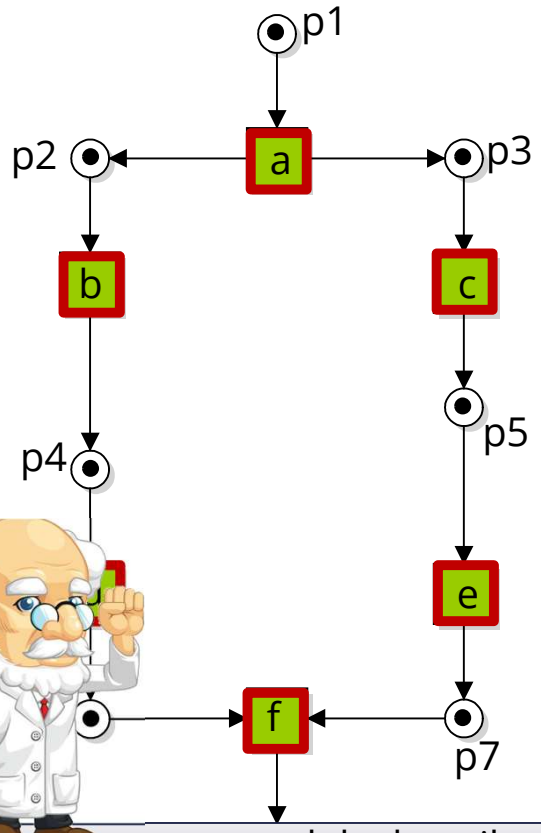
- Transition firing (executing the transition):

  Consume 1 token from each input place
  Produce 1 token in each output place

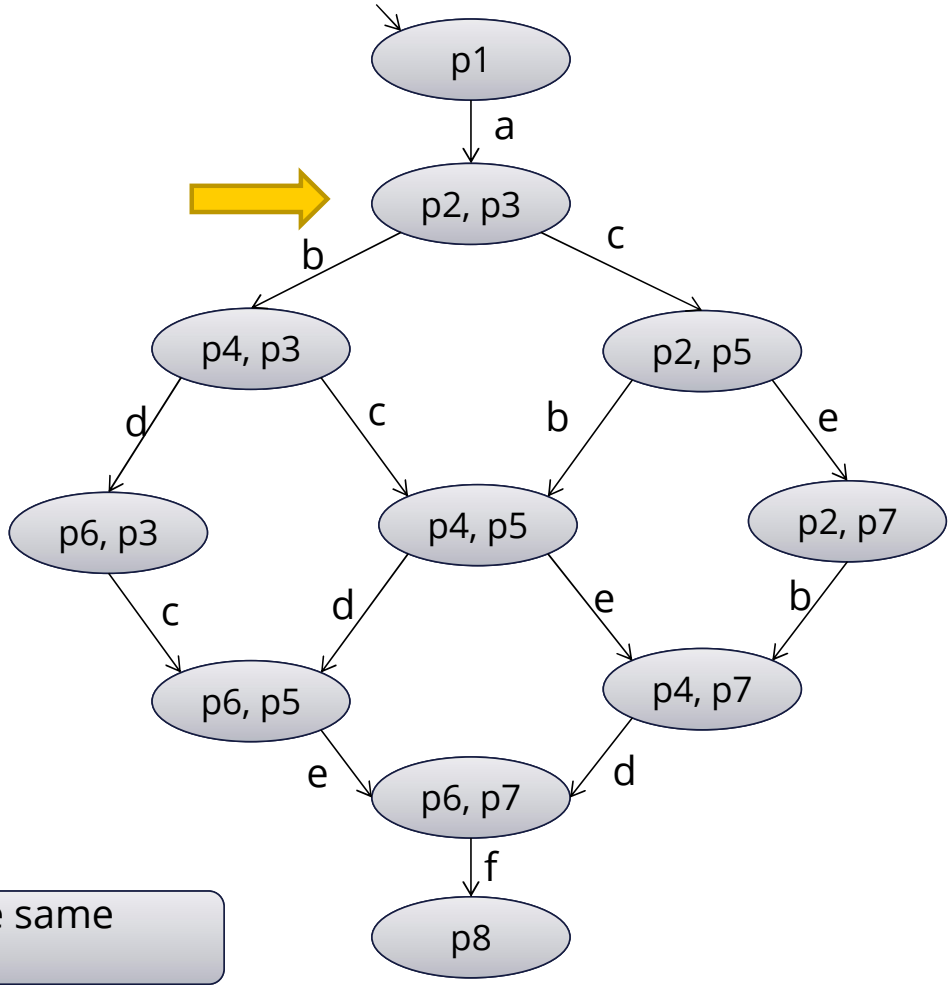
There is no law of token preservation!

Utrecht University

Petri nets: behavior in terms of LTSs

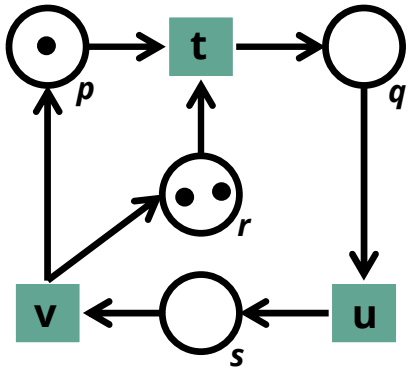


These two models describe the same behavior!



Utrecht University

Petri nets: formal definition



$N = ((P, T, F), m_0)$ with:

$P = \{p, q, r, s\}$

$T = \{t, u, v\}$

$F = \{(p,t), (r,t), (t,q), (q,u)$
$(u,s), (s,v), (v,r), (v,p)\}$

$m_0 = [\, p,\ r^2 \,]$

(reads as: $[\, p^1,\ r^2, q^0, s^0]$)

- A Petri net is a 4-tuple N = $((P, T, F), m_0)$ with:

- P: a (finite) set of places
- T: a (finite) set of transitions
- P and T are disjoint ($P \cap T = \emptyset$)
- F is the flow relation that defines the arcs

$$F \subseteq (P \times T) \cup (T \times P)$$

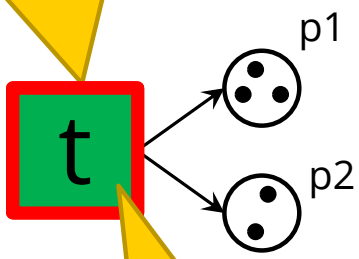
- m_0 is the initial marking, gives the tokens per place

$$m_0 : P \rightarrow \mathbb{N}$$

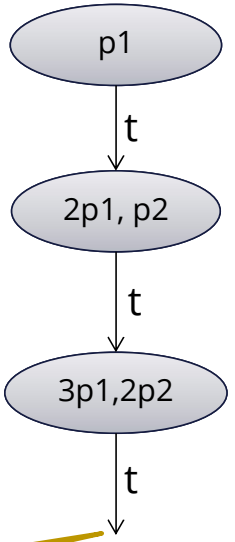
Utrecht University

Petri nets vs. LTSs

All input places always have a token
(empty set, so always true!)



Always enabled!

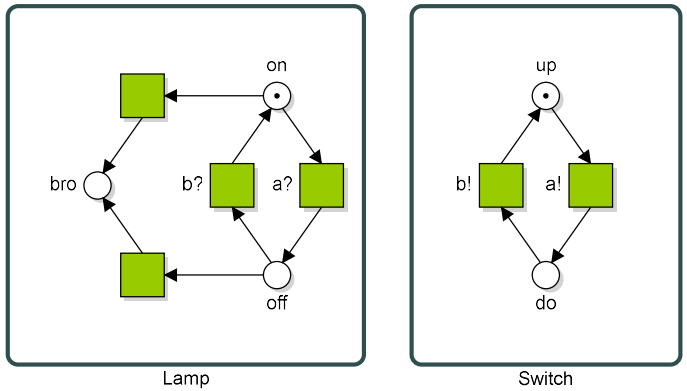


Infinite sequence

Infinite Automata!

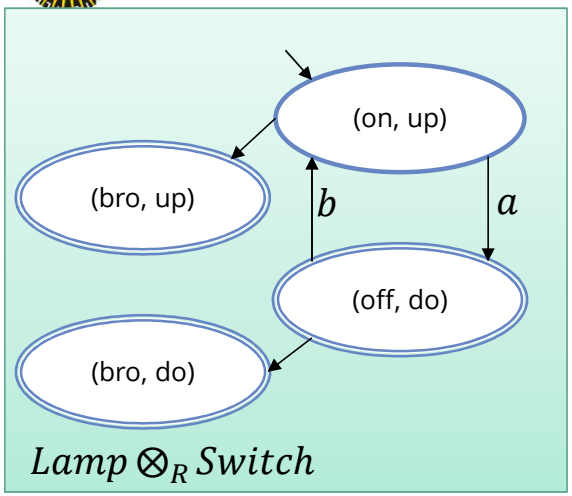
Utrecht University

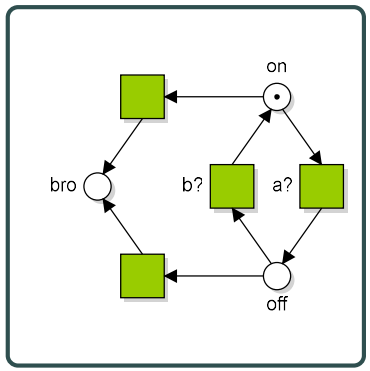
Back to our lamps: now as Petri nets



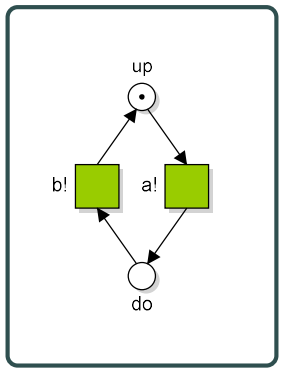
Utrecht University

Back to our lamps: now as Petri nets

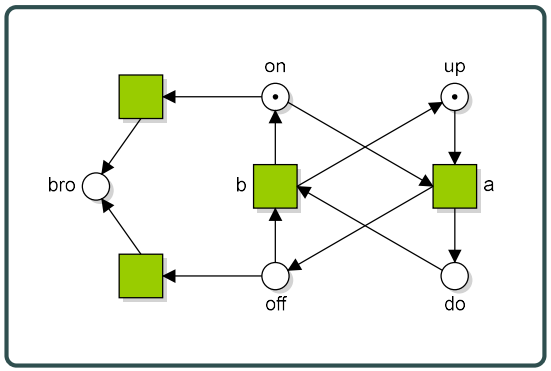




Lamp

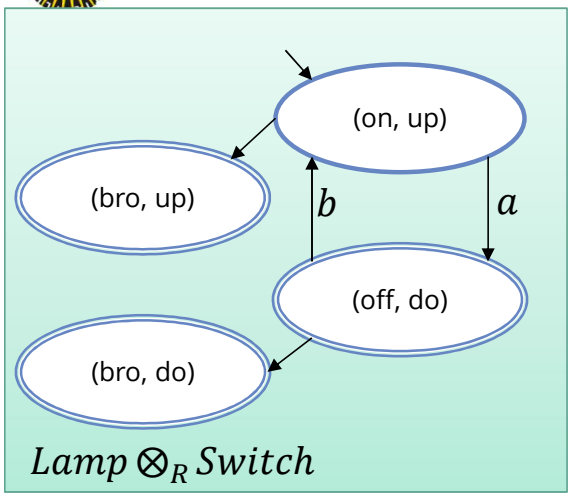


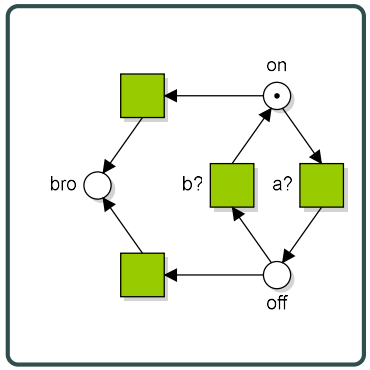
Switch



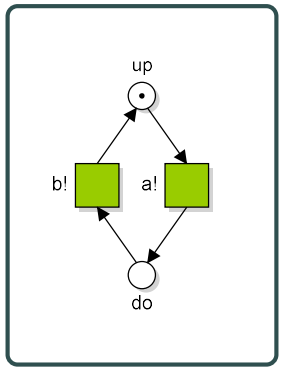
Utrecht University

Back to our lamps: now as Petri nets

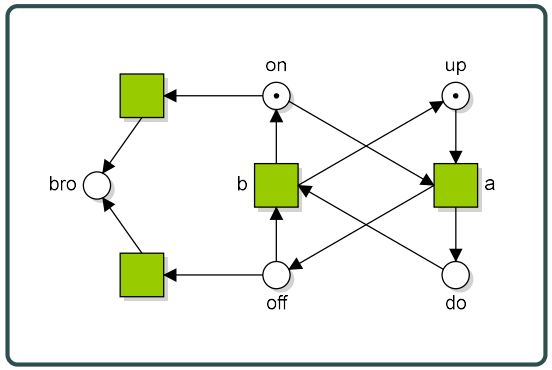


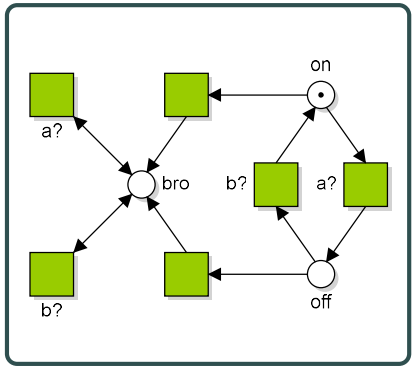


Lamp

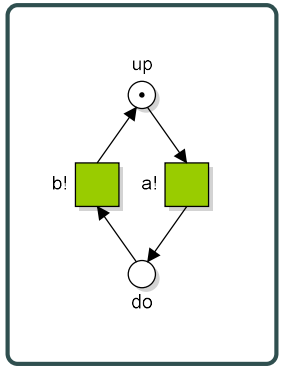


Switch





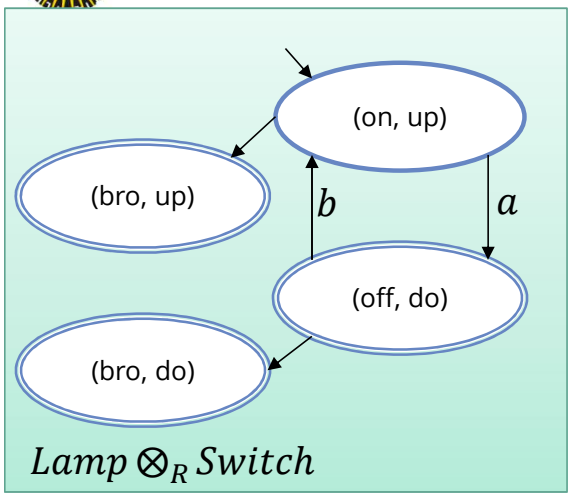
Lamp

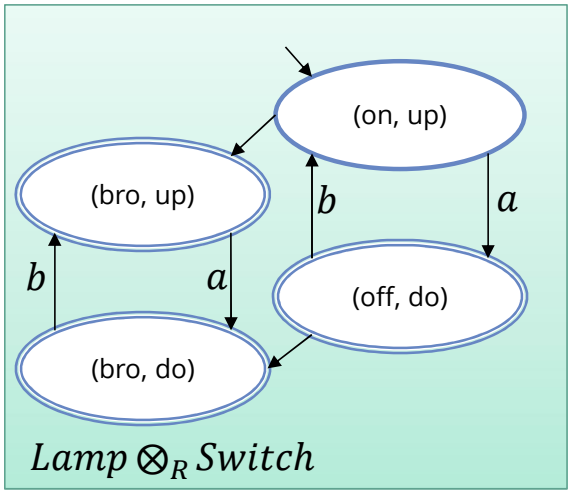


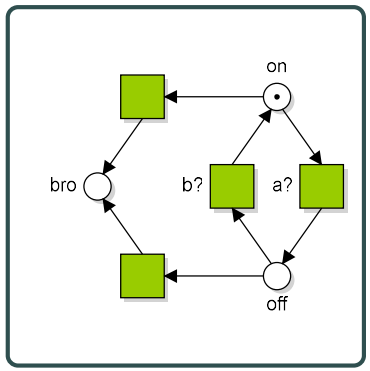
Switch

Utrecht University

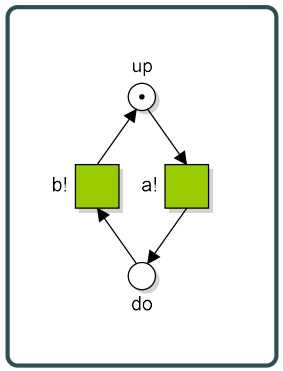
Back to our lamps: now as Petri nets
Synchronous communication



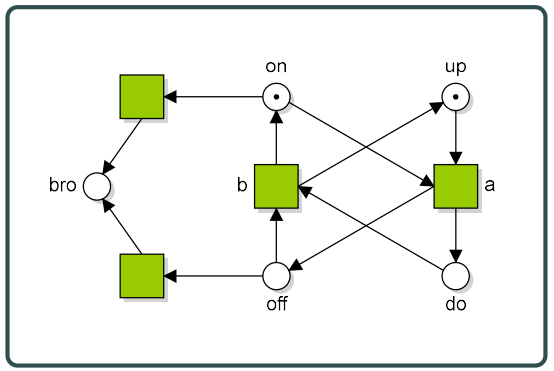


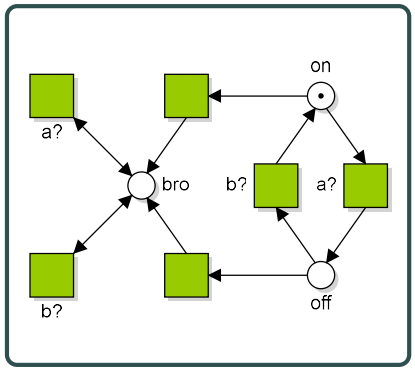


Lamp

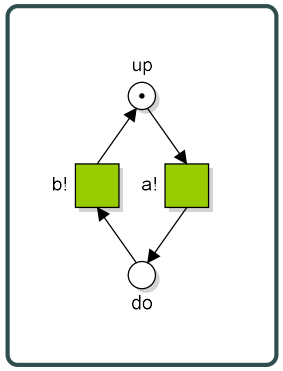


Switch

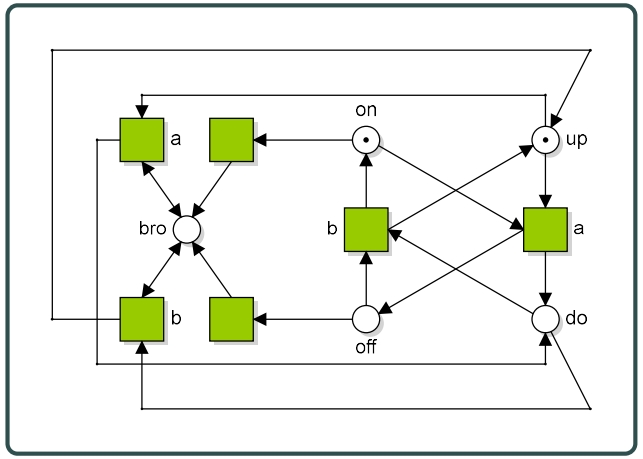




Lamp

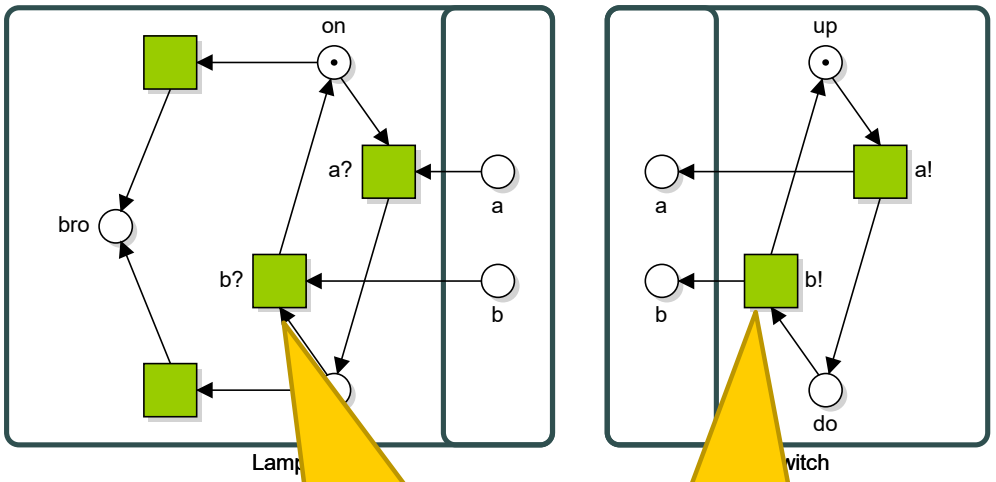


Switch



Utrecht University

Back to our lamps: now as Petri nets
Asynchronous communication!

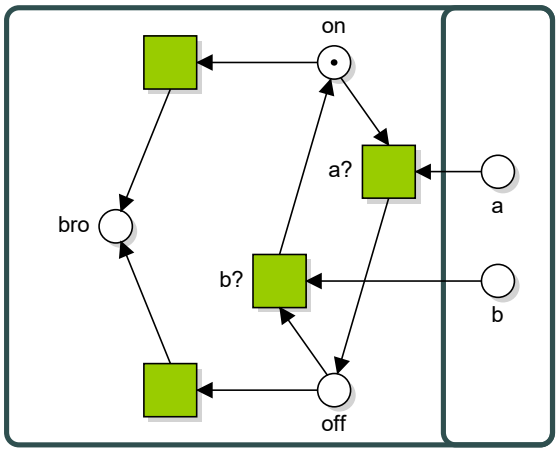


Transition receives a message by consuming a token from “message place” b

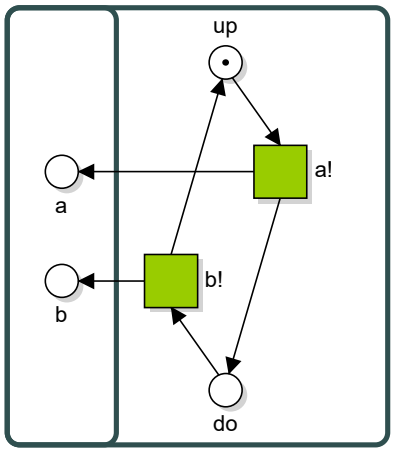
Transition sends a message by placing a token in the “message place” b

Utrecht University

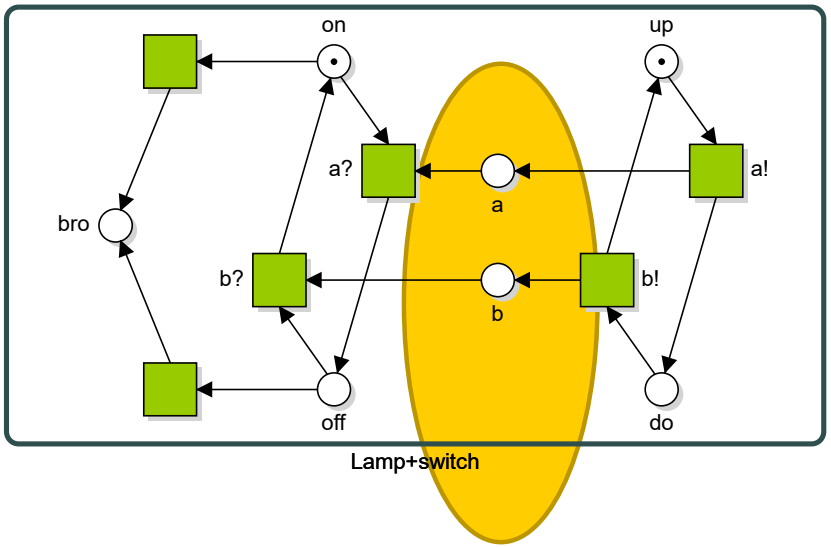
Back to our lamps: now as Petri nets
Asynchronous communication!



Lamp



Switch

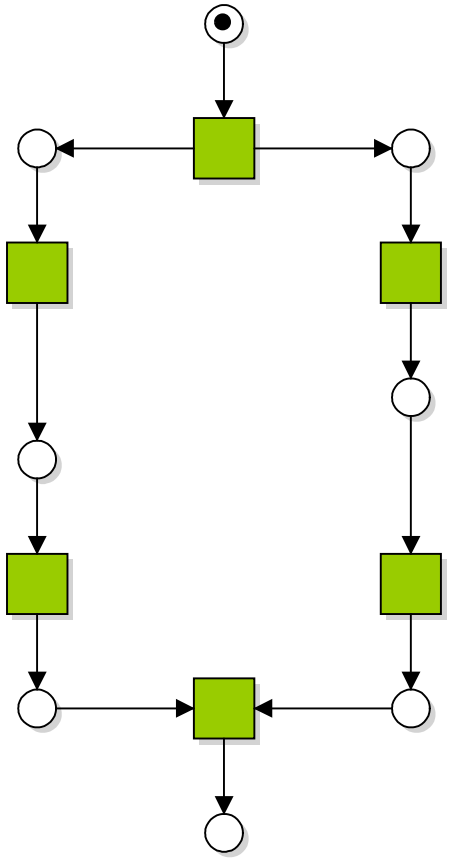


Lamp+switch

Message buffers

Utrecht University

Petri nets and communication mechanisms

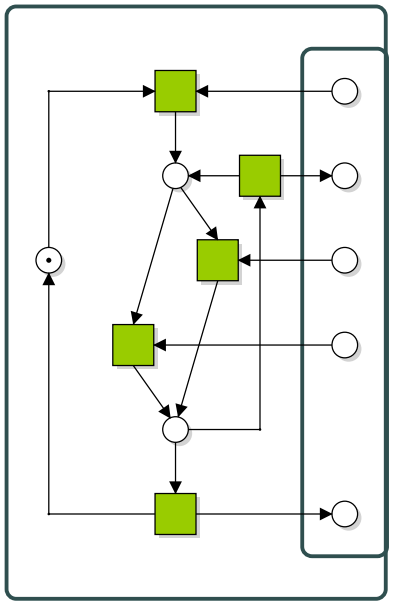


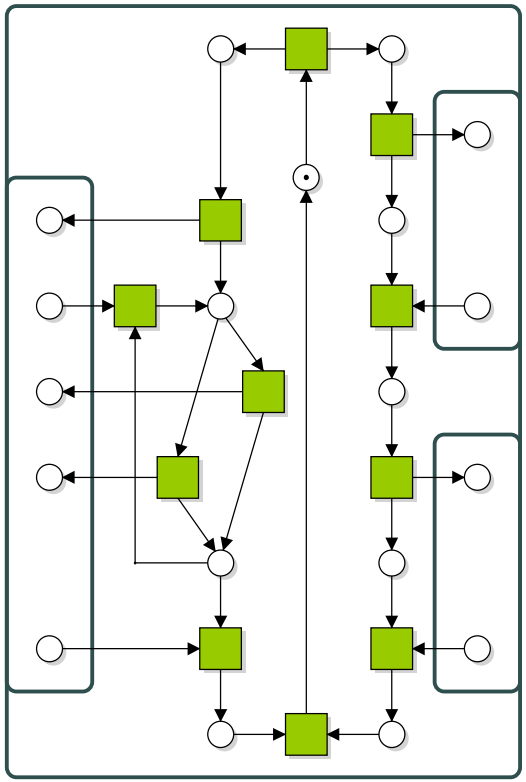
- Synchronous communication
  - **Transitions that consume from multiple places**
- Asynchronous communication
  - **Places resemble pools of messages to be handled**
  - **Random access of messages in these pools**

Both communication mechanisms in a single, **graphical** formalism!

Utrecht University

Teaser for next week:





- Modelling communication between components
- Different perspectives on message handling

Utrecht University

For the remainder of today:

- Work on the assignment

- Thursday: first presentation session!
  - ➔ **No need for a complete presentation**
  - ➔ **Schedule will be made available Thursday morning**

Utrecht University

DISCLAIMER

The information in this presentation has been compiled with the utmost care,
but no rights can be derived from its contents.