Hierarchy & Model checking

Lecture Outline

- Recap 3d lecture/concept check
- Classroom exercise
- Hierarchy in Petri nets
- Model checking

Recap 3d lecture

- What do we mean by the "structure" of a Petri net ?
- What do we mean by "behaviour" of a Petri net ?
- If we characterize a Petri-net as a "WF-net", is that a structural or behavioural characterization?
- If we characterize a WF-net as a "sound" WF-net, is that a structural or behavioural characterization?

Recap 3d lecture

- Is characterization of a WF-net as :
  - A Free-choice net
  - A Well-structured net
  - An S-coverable net

  - A structural or a behavioural characterization?

Recap 3d lecture

- What do we mean if we talk about "state space explosion" ?

Recap 3 lecture

- Why is this state space explosion a “problem” ?

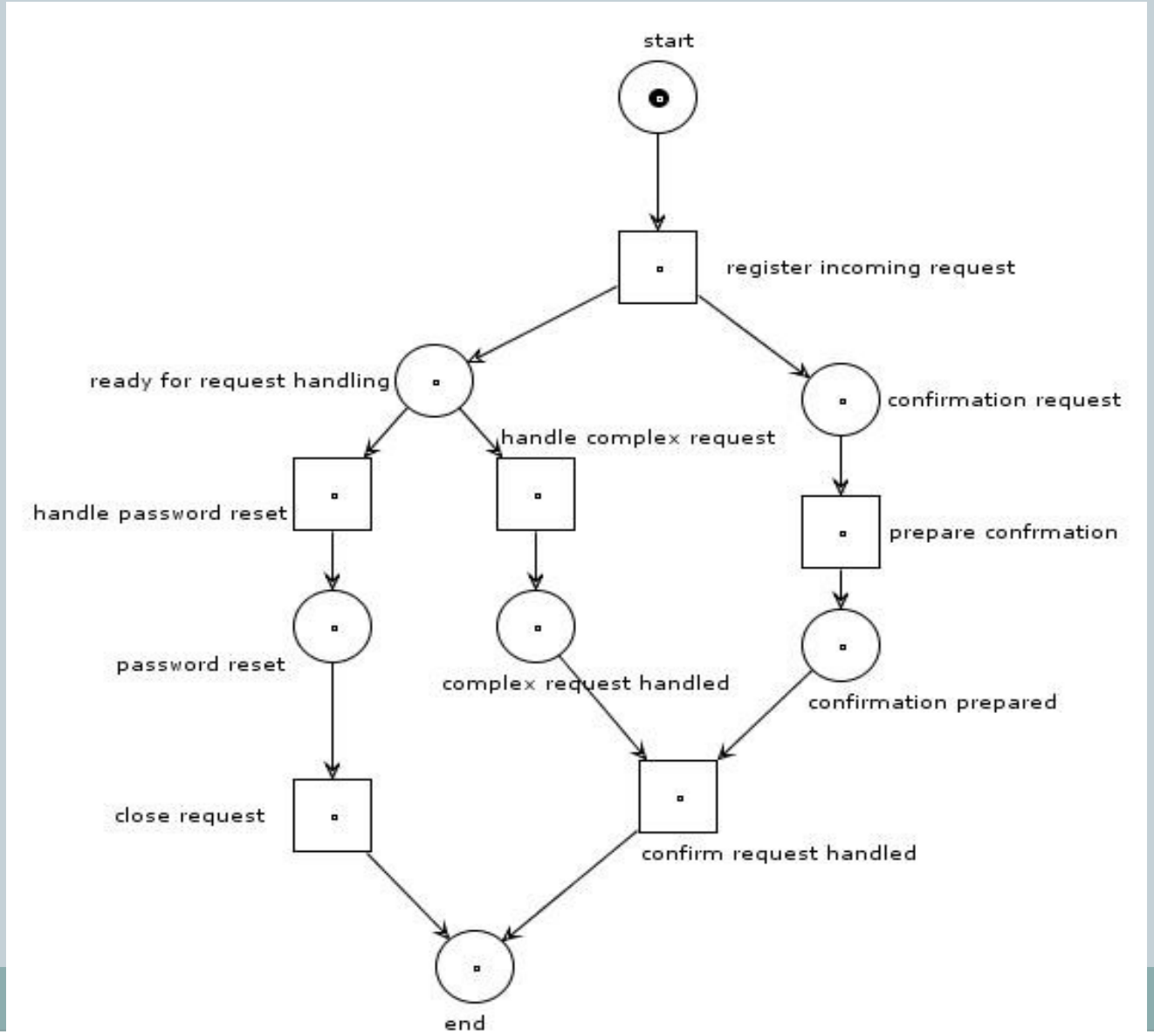
Recap 3 lecture

- If you know that a Petri net is a workflow net, what can you infer then about liveness of that Petri net?

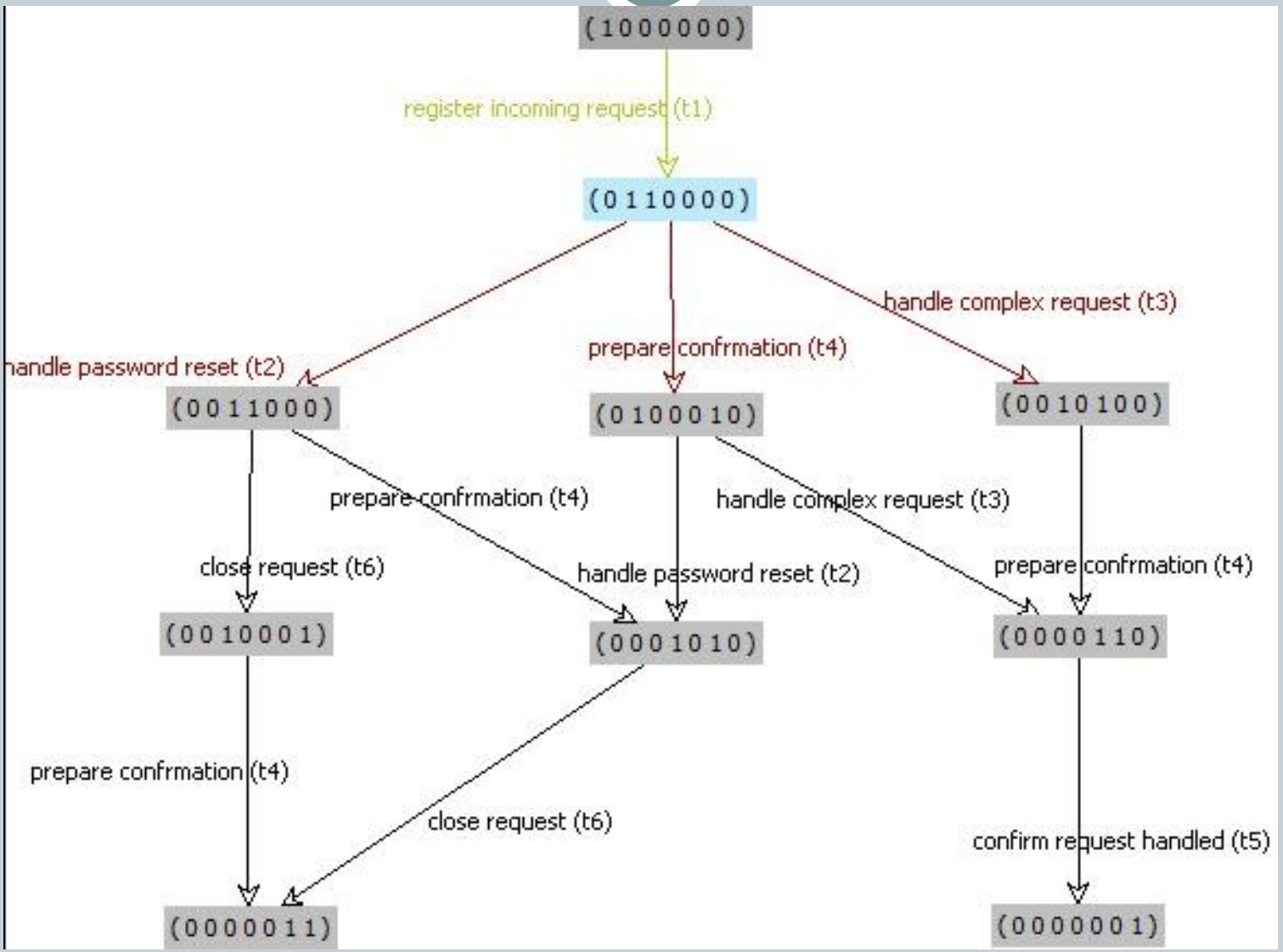
Lecture Outline

- Recap 3d lecture/concept check
- Classroom exercise
- Hierarchy in Petri nets
- Model checking

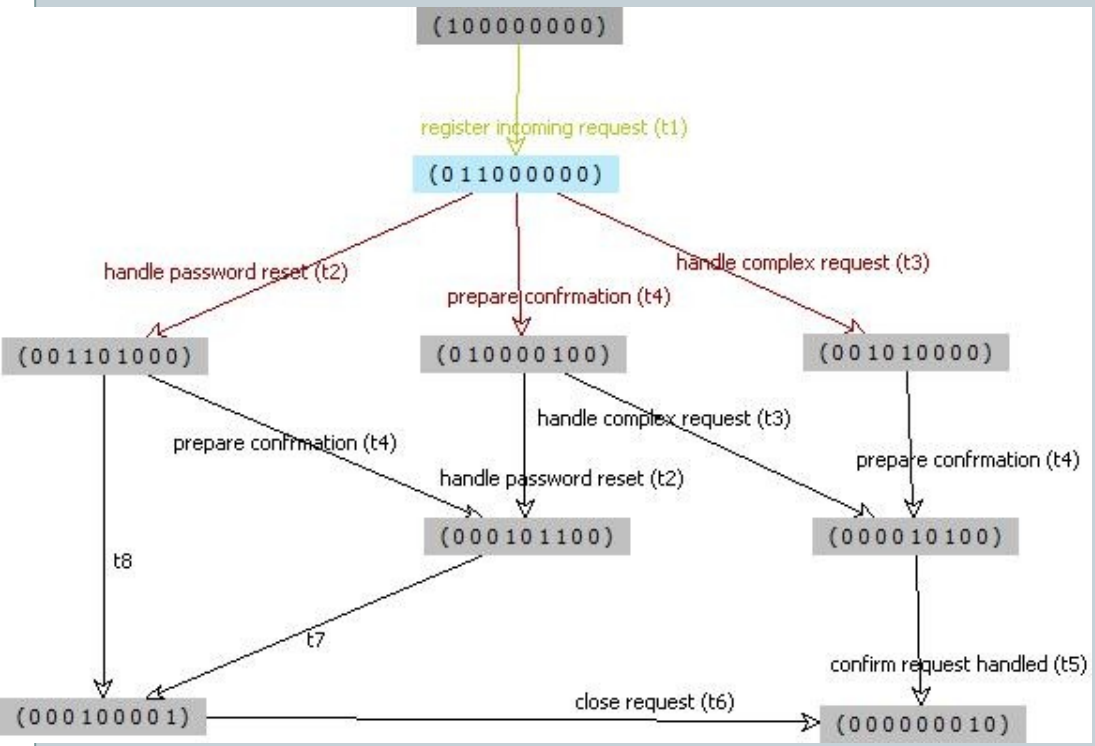
Classroom exercise

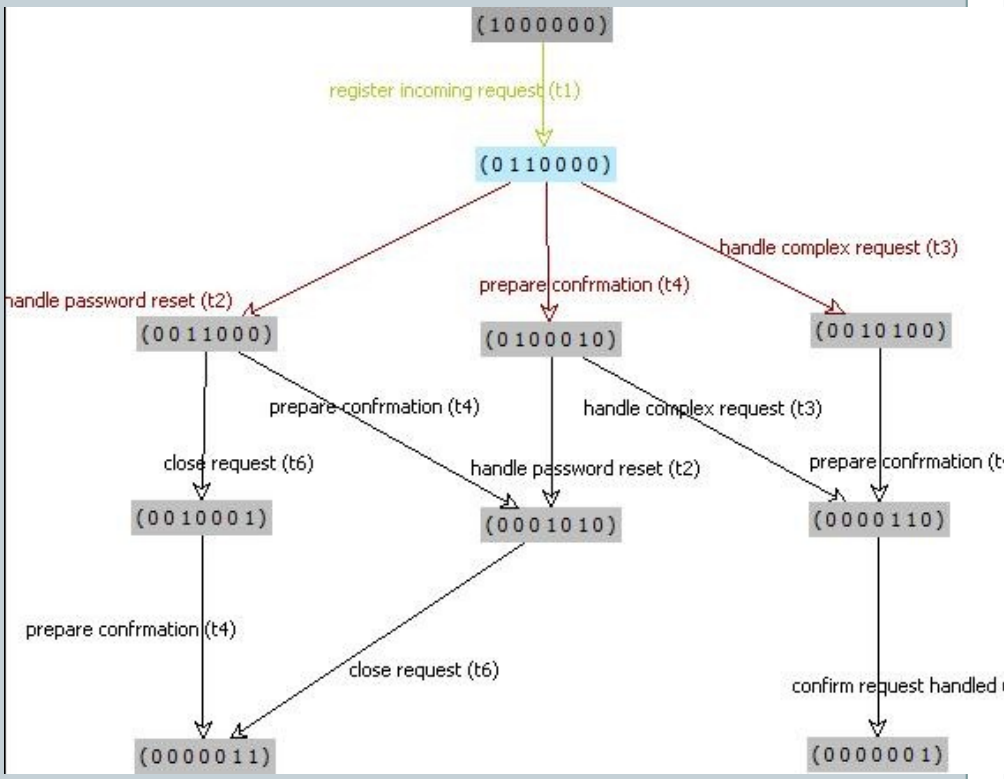


Classroom exercise



Classroom exercise



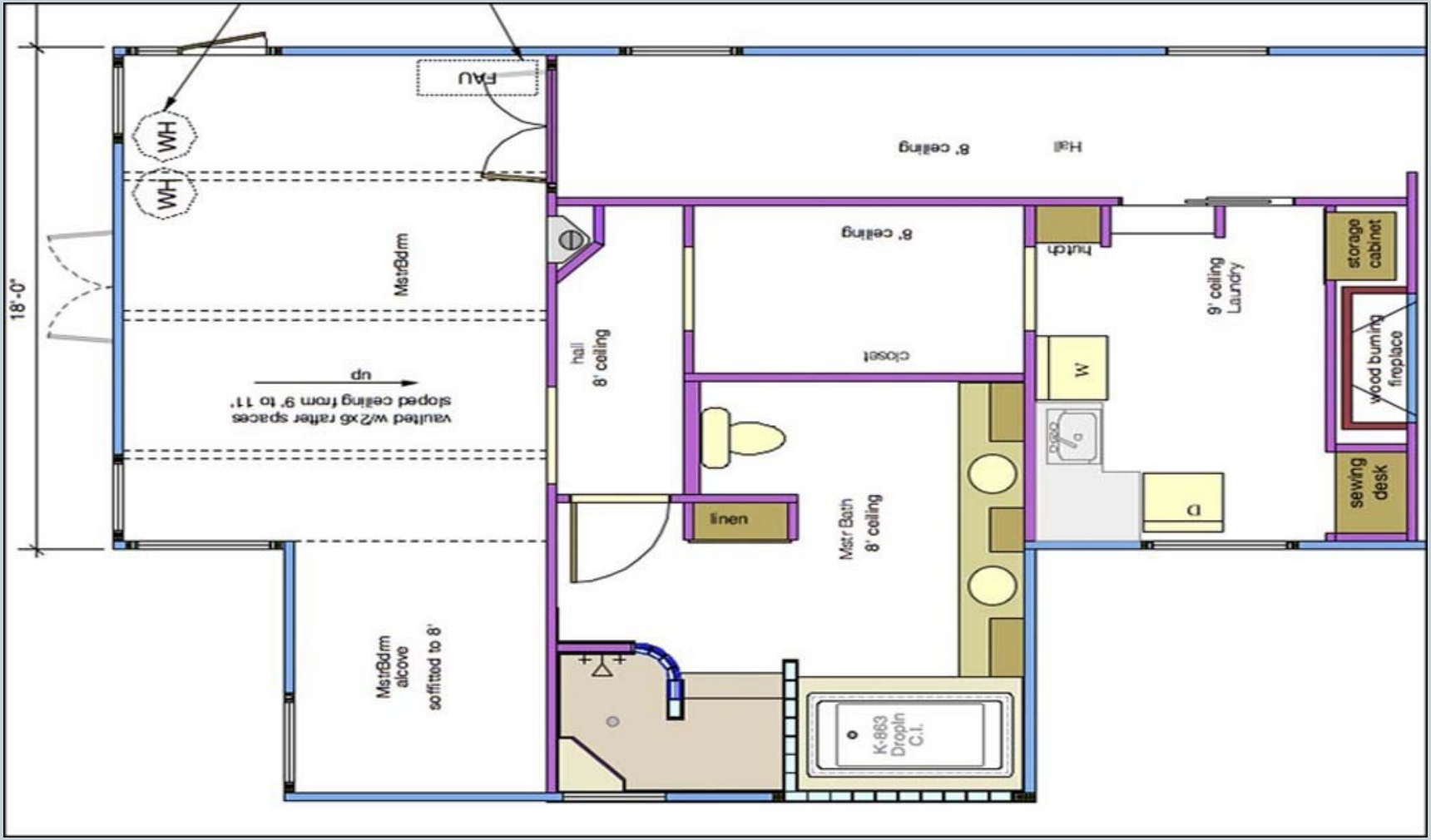


Lecture Outline

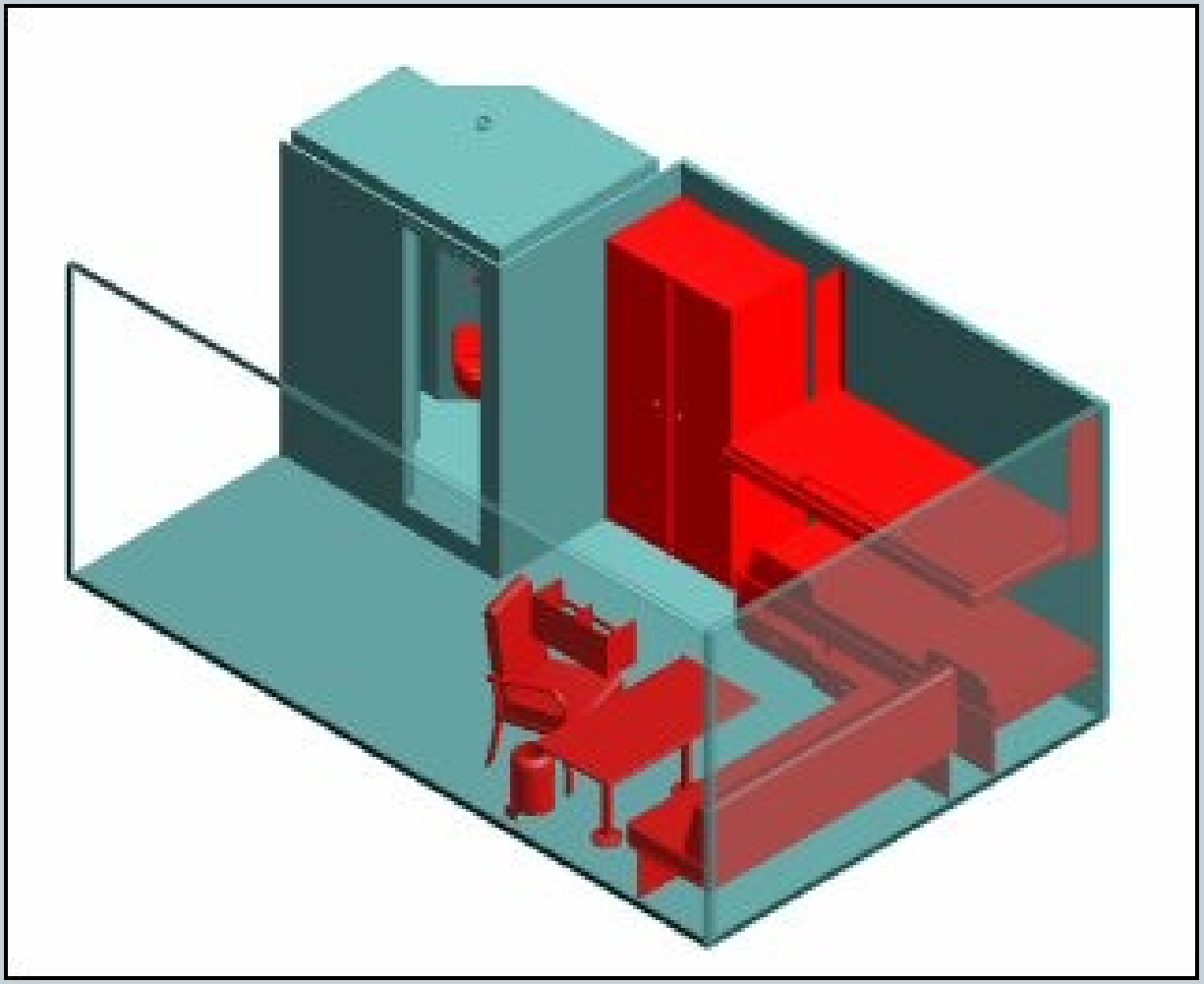
- Recap 3d lecture/concept check
- Classroom exercise
- Hierarchy in Petri nets 
- Model checking

Top down design (level 0 : a house)

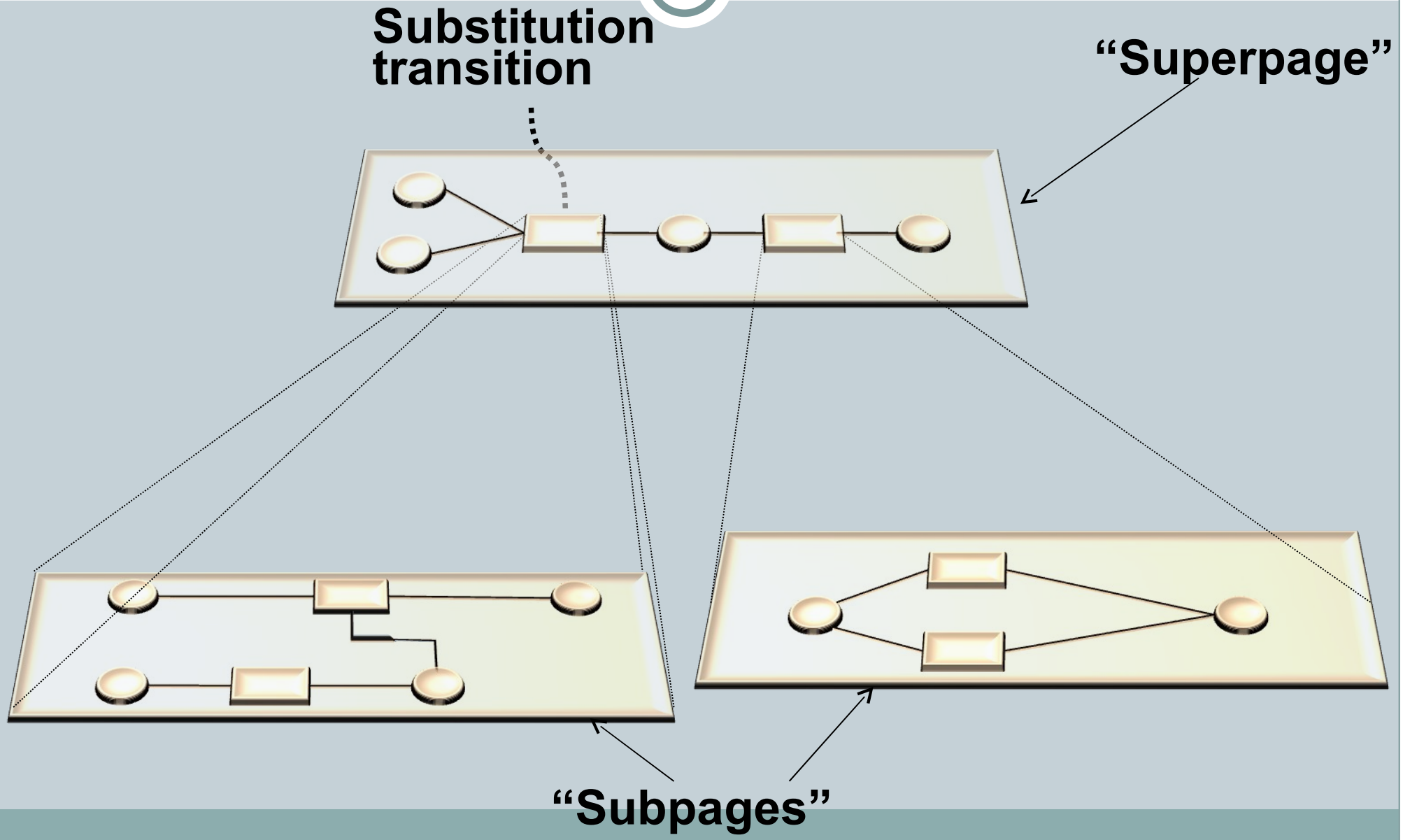
Top down design (level 1 ; a floor)



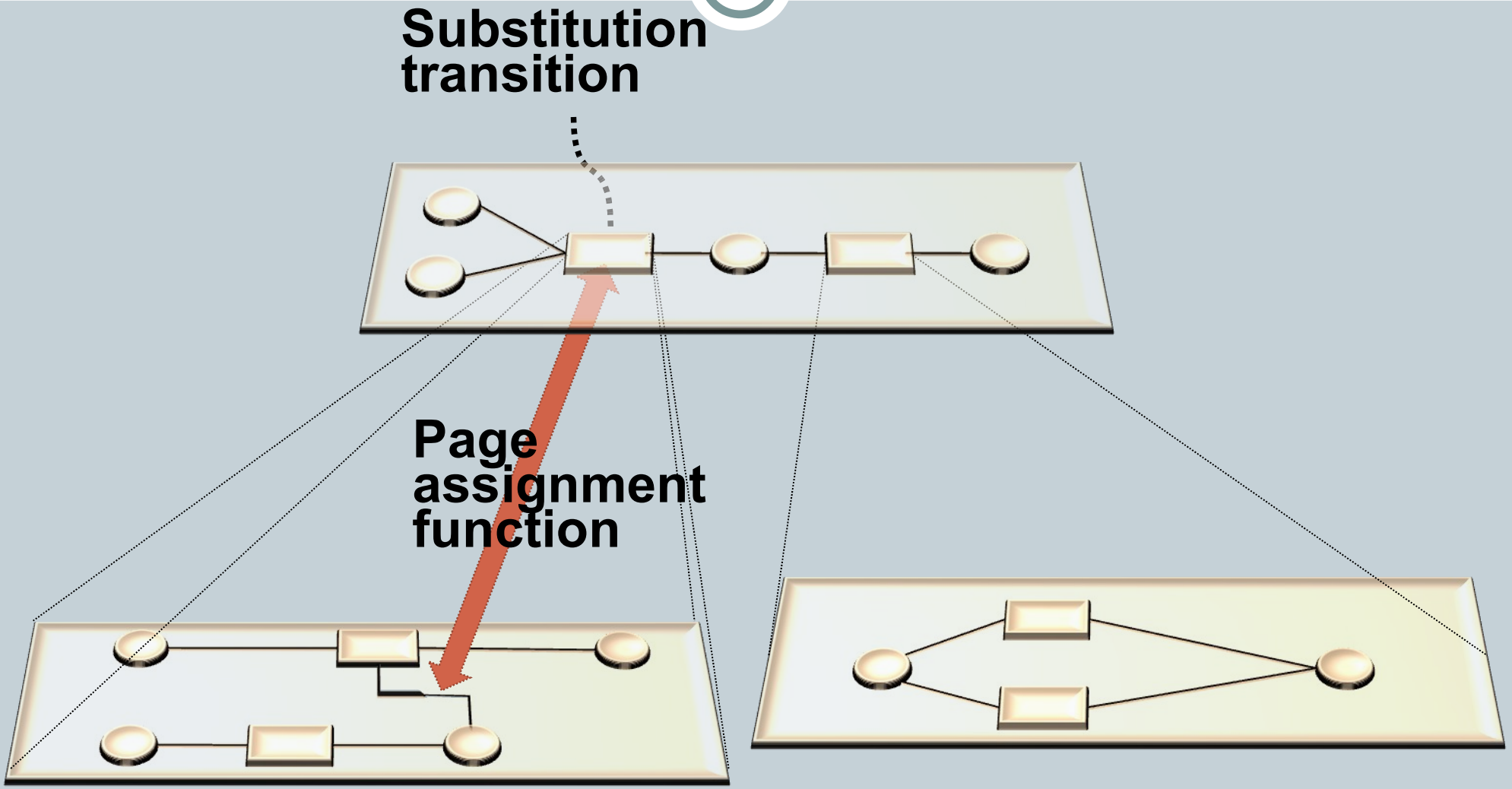
Top down design (level 2 : a room)



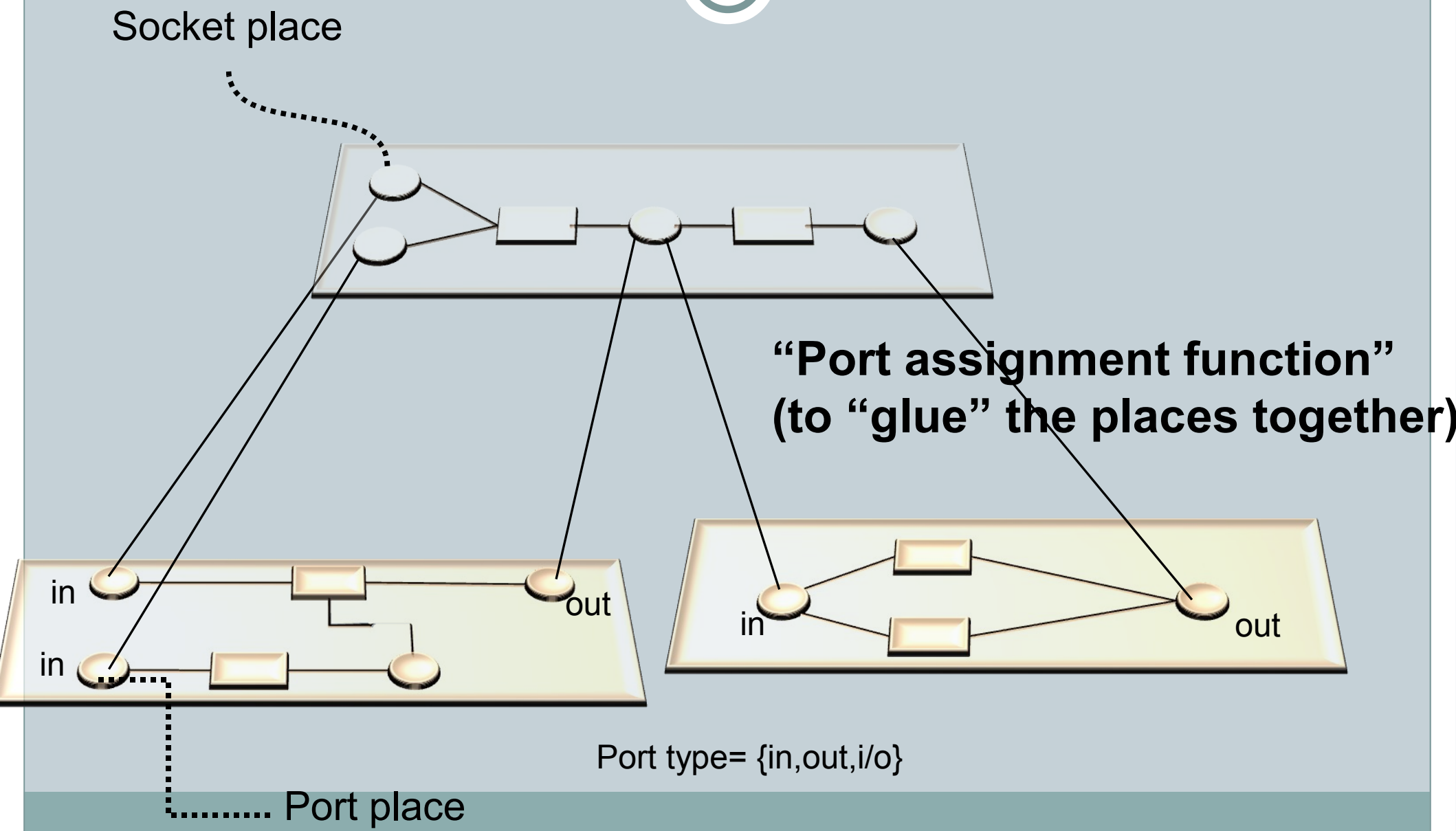
Hierarchical Coloured Petri nets



Hierarchical Coloured Petri nets

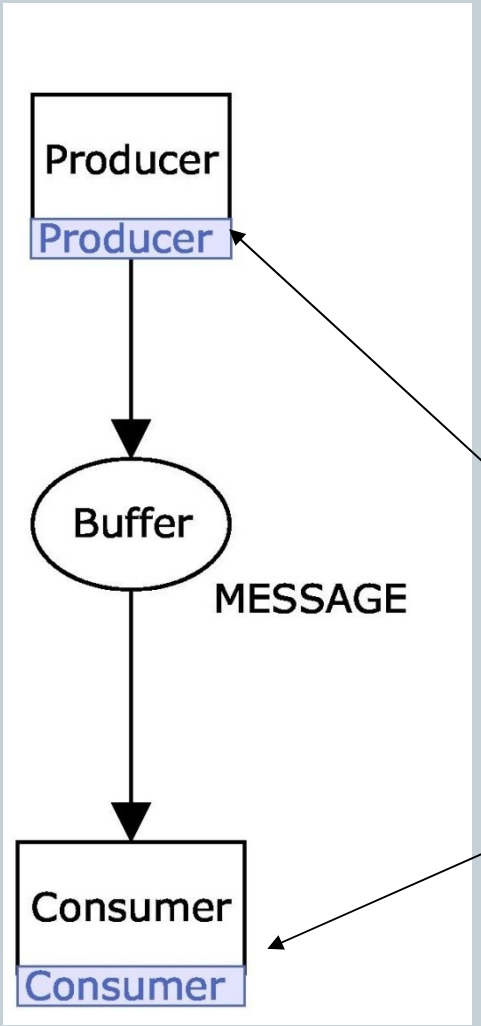


Hierarchical Coloured Petri nets



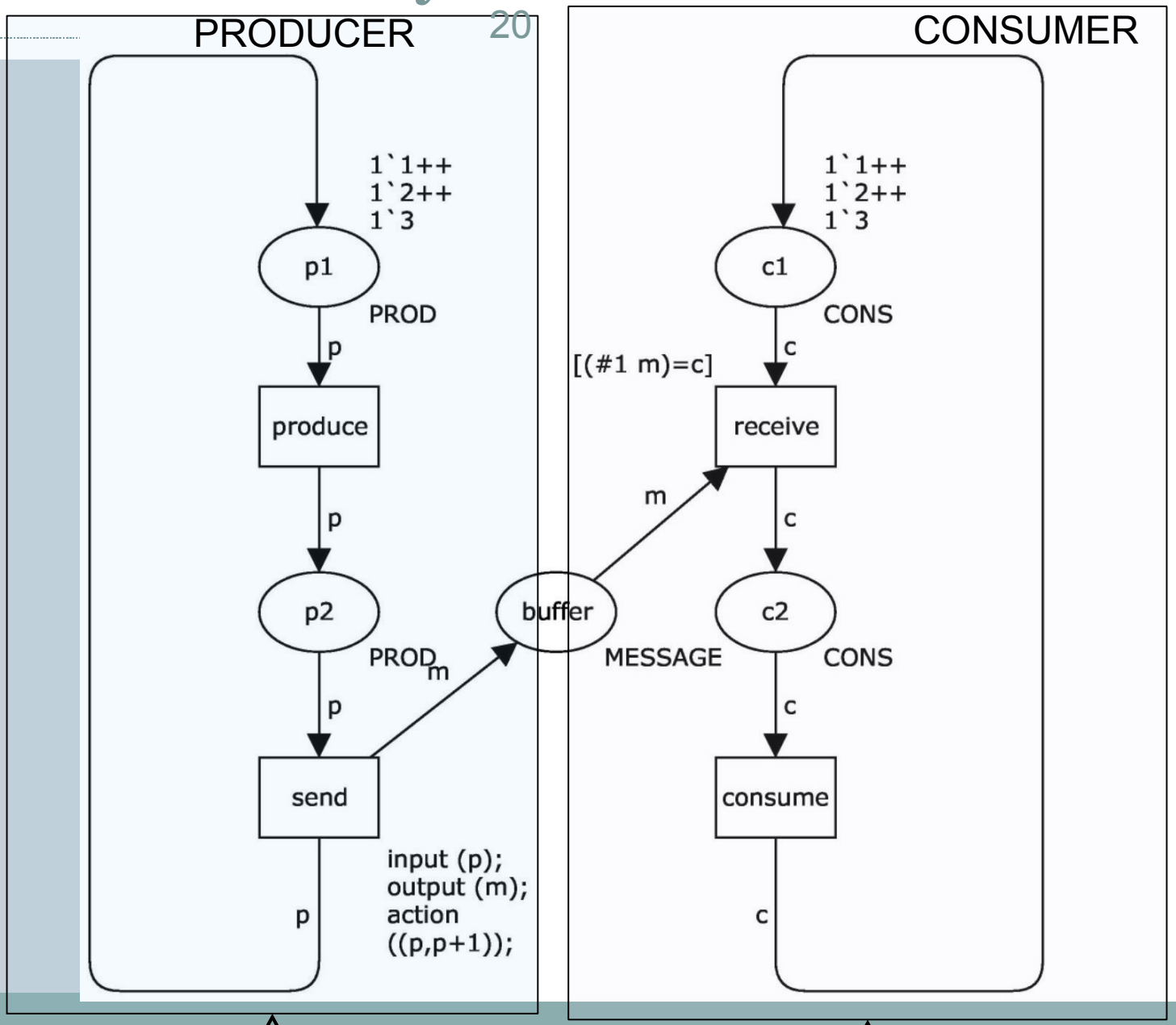
Example Hierarchical Coloured Petri net producer consumer system

Superpage



“Substitution transitions”

Example: CP net of our producer consumer system



Hierarchical Coloured Petri nets

- Even coloured petri nets can become quite large!
- If we extend CP nets to ***hierarchical*** CP nets they can become more readable and managable.
- It is similar to the concept of “subroutine” in programming languages, which makes programs smaller and easier to read.

Hierarchical coloured Petri nets

- The **subpage** in Hierarchical Petri nets is a bit like the **subroutine** in programming
- We can use for **top-down** process design, or **bottom-up**

Hierarchical coloured Petri nets

- **Multiple superpages** can be linked to one subpage definition

- Multiple subpage instances **each with their own markings** (Like a subprogram definition can have multiple instances running, each with private copies of local variables)

Lecture Outline

- Recap 3d lecture/concept check
- Classroom exercise
- Hierarchy in Petri nets
- Model checking

Model checking

- Introduction
- Formalizing specification of properties
- Examples

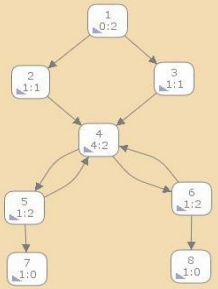
Introduction

- Definition model checking:
  - Given a model of a system, verify automatically whether this model meets a given specification
- Ingredients:
  - Model : eg. the Reachability Graph of a Petri net
  - Specification : eg. of properties like liveness, soundness etc.
  - Verification: automatically eg. with CPN-tools

Introduction

Specification :
- Liveness
- Soundness
- Etc.

Model of system

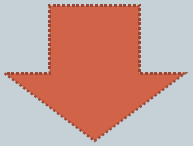


Model checker

Yes/No

Introduction

- Ingredients:
  - Model : eg. Petri net model (i.e. its RG)
  - Specification : eg. properties Liveness, soundness etc.
  - Verification: automatically eg. with CPN-tools



1. We need a formal language to write down the specification ➔ CTL

AND

2. We will show some (simple) examples how you can do model checking in CPN Tools

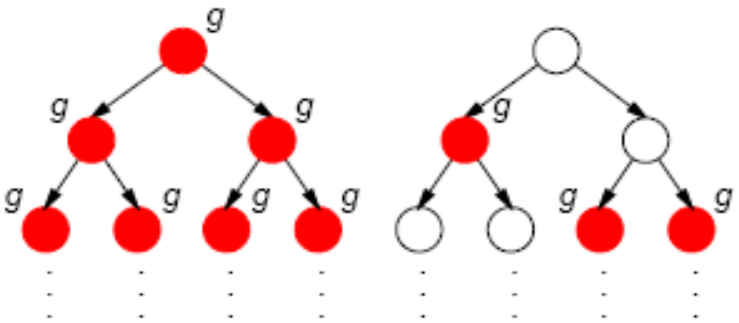
Specification of properties in CTL

Formulas are constructed from path quantifiers and temporal operators:

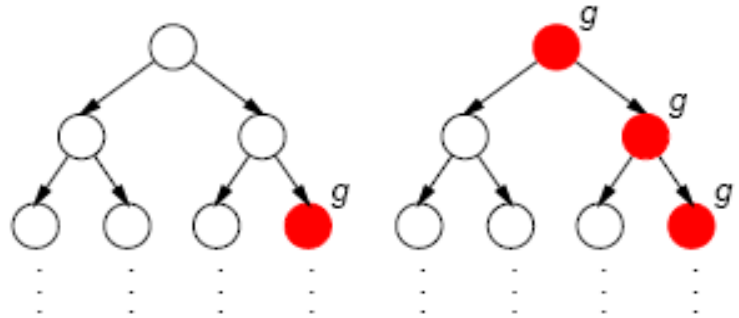
1. Path quantifier:
   - **A**—"for every path"
   - **E**—"there exists a path"
2. Temporal Operator:
   - **X***p*—*p* holds next time.
   - **F***p*—*p* holds sometime in the future
   - **G***p*—*p* holds globally in the future
   - *p***U***q*—*p* holds until *q* holds

Specification of properties in CTL

The four most widely used CTL operators are illustrated below. (s_0 is the root of each computation tree.)

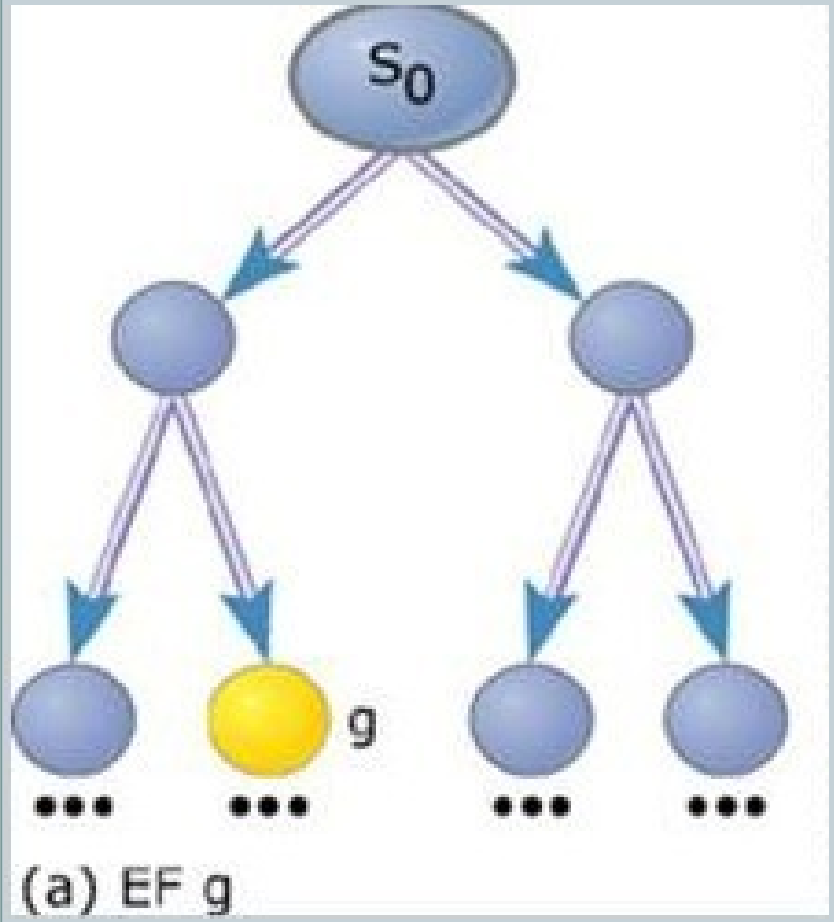


$M, s_0 \models \mathbf{AG}\, g$ $M, s_0 \models \mathbf{AF}\, g$



$M, s_0 \models \mathbf{EF}\, g$ $M, s_0 \models \mathbf{EG}\, g$

Specification of properties in CTL



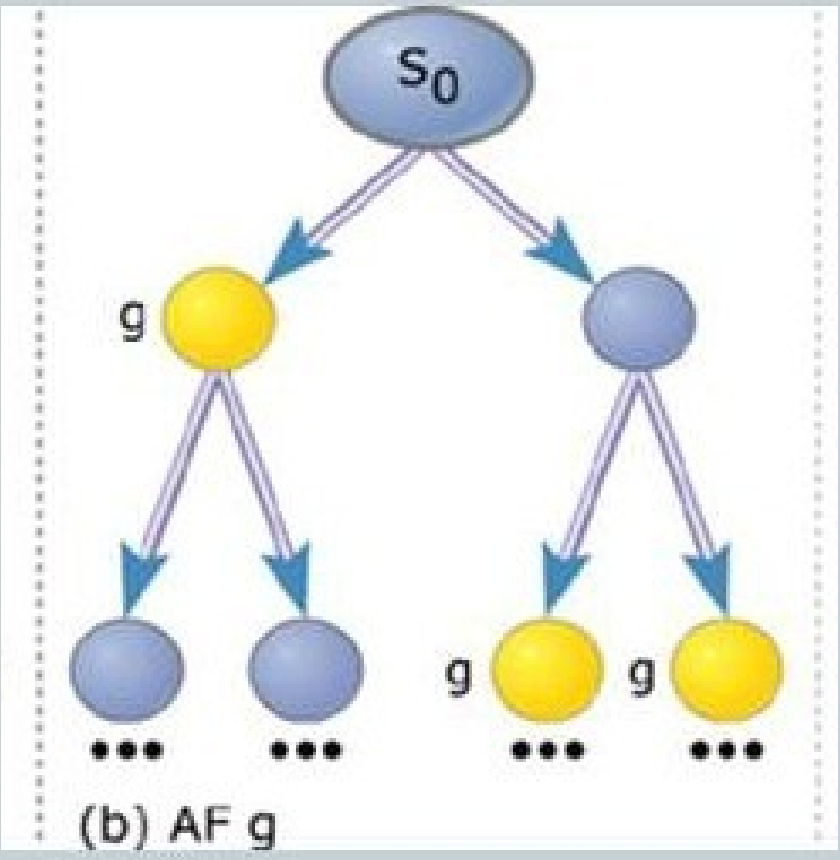
REACHABILITY :

EF(g) is true, if there is a path from S_0 to a state where g is true

EF(g) in CPN Tools (ASK-CTL):

POS(g)= EXISTS_UNTIL(TT,g)

Specification of properties in CTL



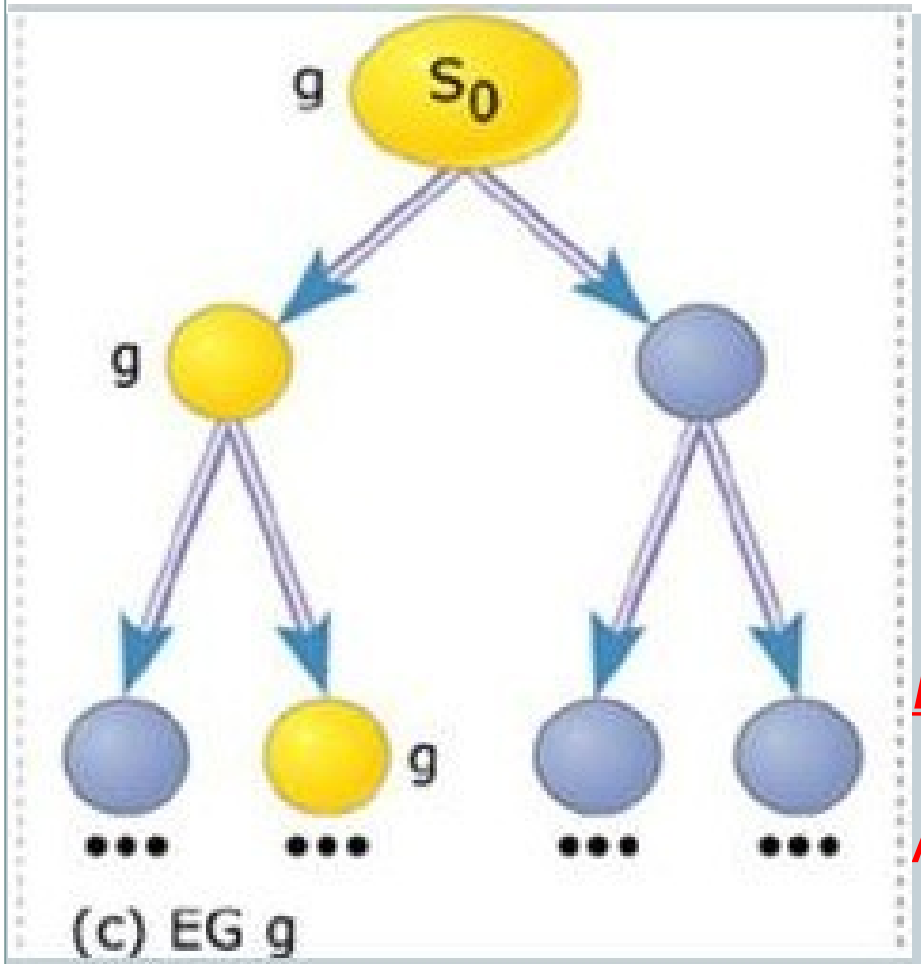
LIVENESS :

AF(g) is true, if all paths from S_0 eventually encounter a state where g is true

AF(g) in CPN Tools (ASK-CTL):

EV(p) =FORALL_UNTIL(p)

Specification of properties in CTL

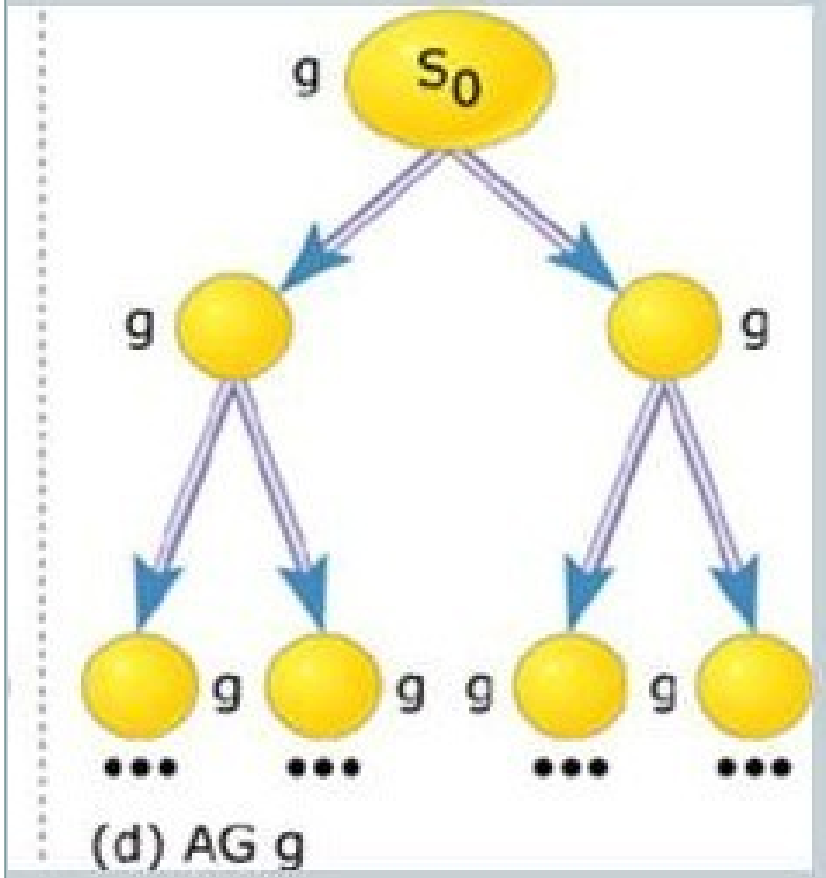


***EG(g)* is true, if there is a path from S_0 along which g is true in every state**

***EG(g) in* CPN Tools *ASK-CTL*:**

ALONG(g) = ¬EV(¬(g))

Specification of properties in CTL



BOUNDEDNESS :

AG(g) is true, if along all paths from S_0 , g is true in every state

AG(g) in CPN Tools ASK-CTL:

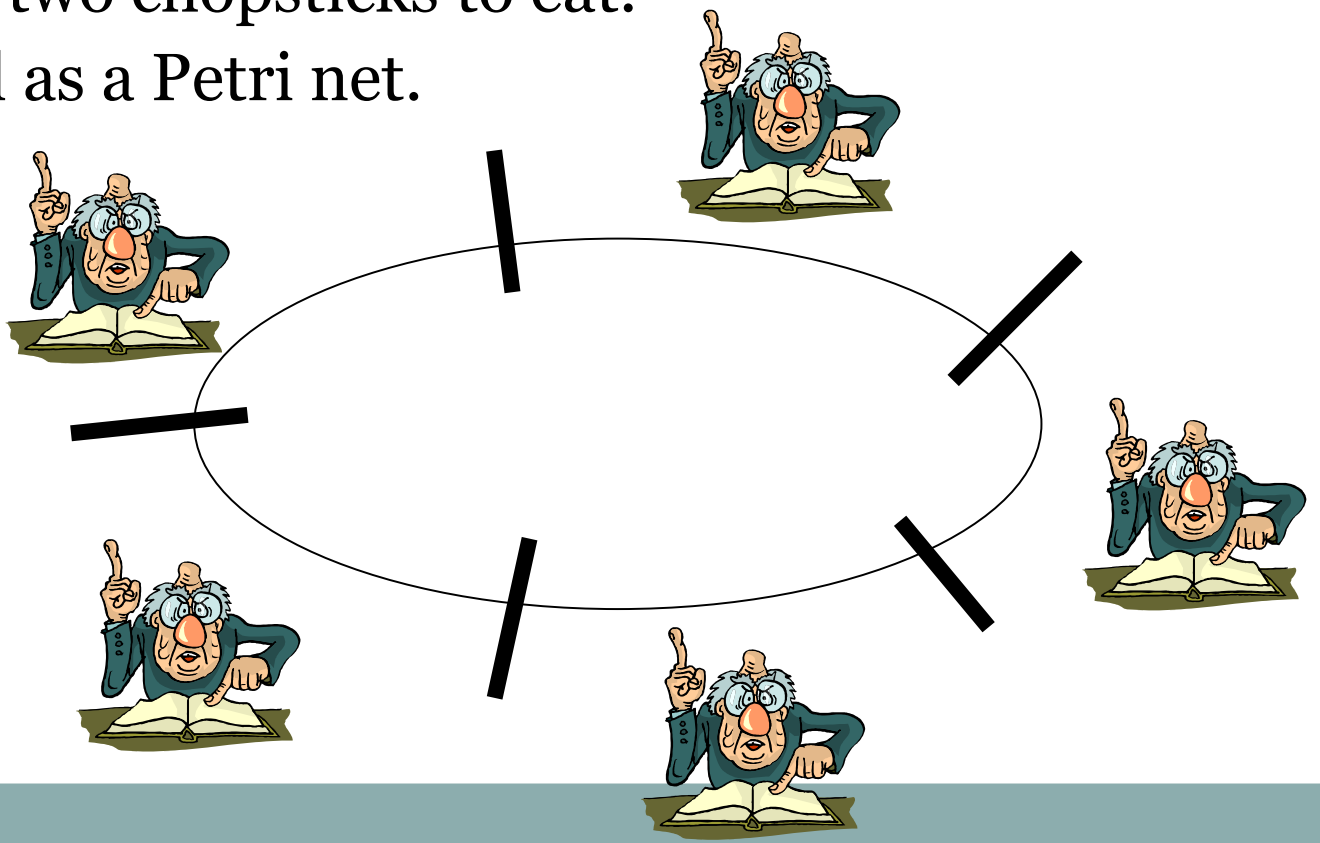
INV(g)= ¬POS(¬(g))
[AG(g)= ¬EF(¬(g))]
EF(p) =POS(g)=EXISTS_UNTIL(TT,g)

Specification of properties in CTL

- Translations CTL ➔ ASK-CTL (CPN-Tools):
  - AG(p)= INV(p)
  - EF(p) = POS(p)
  - AF(p)= EV(p)
  - EG(p)= ALONG(p)

Example: Dining philosophers

- 5 philosophers sharing 5 chopsticks: chopsticks are located in-between philosophers
- A philosopher is either in state eating or thinking and needs two chopsticks to eat.
- Model as a Petri net.



Class room Exercise: Dining philosophers

Now consider a variation of the Dining Philosophers example. A number of philosophers are initially outside a dining room. Each of them decides at some point to enter the room bringing along one chopstick (for the left hand) to be shared with the neighbor philosopher. Once in the room the philosopher sits down and starts thinking. If both left and right chopsticks are unused the philosopher can decide to start eating, thus occupying the two chopsticks. When nished eating the philosopher can decide to think again making available again the left and right chopsticks. As life is unpredictable, a philosopher can get poisoned and die while eating. In this case the chopsticks are forever lost.

The Petri net model is on the next sheet. The exercise is to calculate, by hand, the reachability graph and answer the questions:

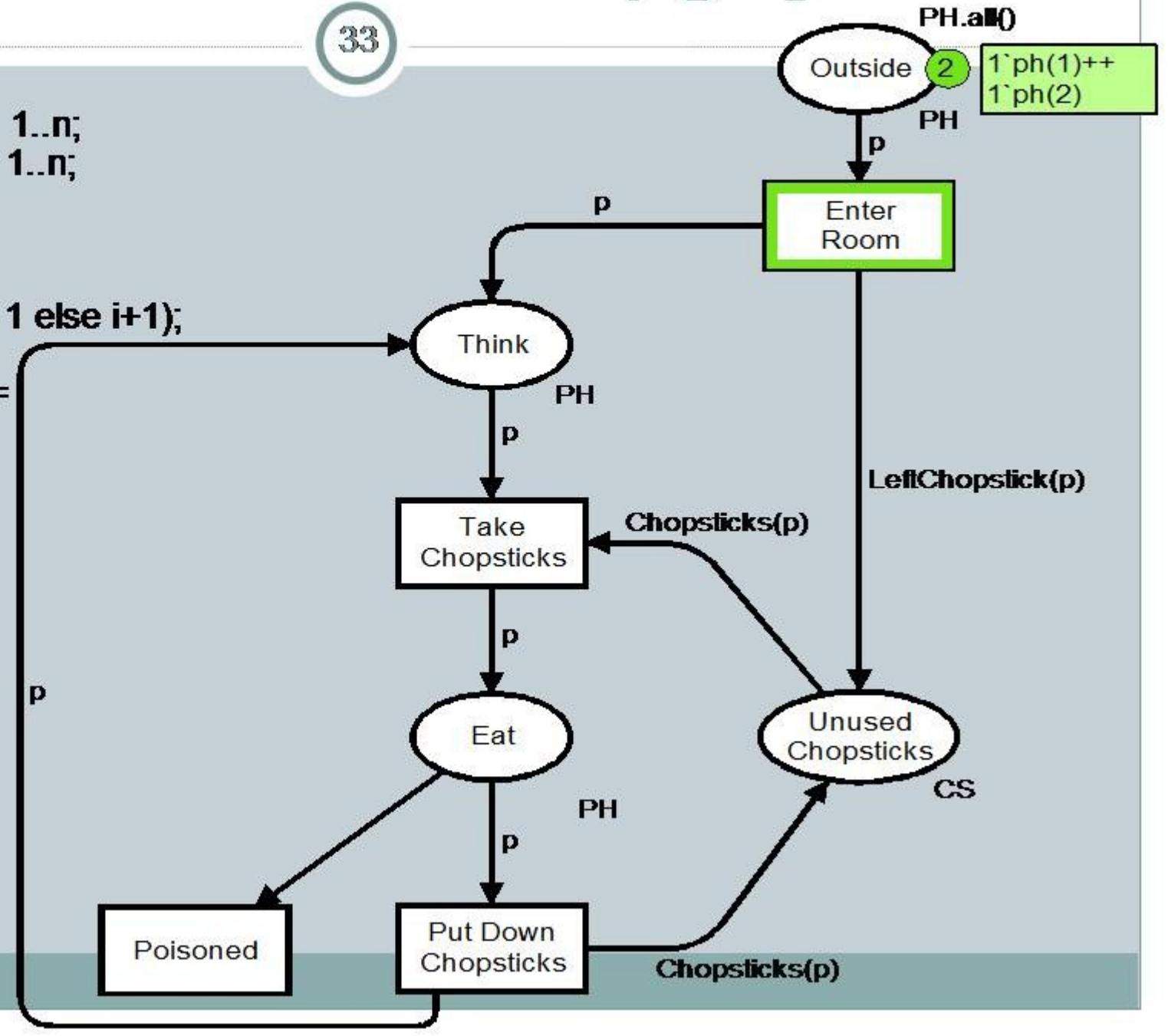
- Is the initial state a home state?
- Is the transition “take chopsticks” a live transition ?
- Is there a state where the Petri net reaches a deadlock ?

Classroom exercise: Calculate the reachability graph


```
val n = 2;
colset PH = index ph with 1..n;
colset CS = index cs with 1..n;
var p: PH;

fun Chopsticks(ph(i)) =
1`cs(i) ++ 1`cs(if i=n then 1 else i+1);

fun LeftChopstick (ph(i)) =
cs(i);
```



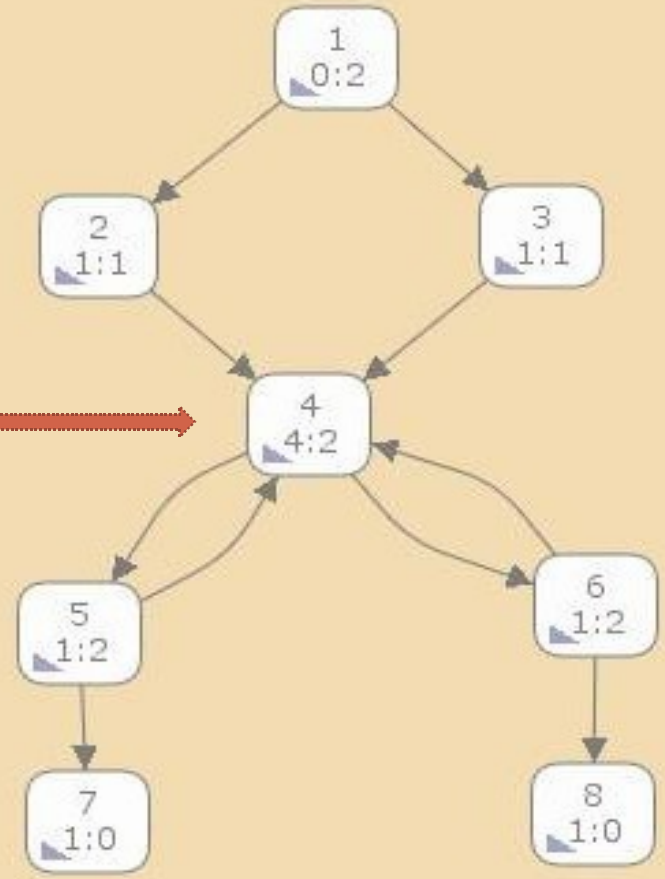
Reachability graph

Room entered (1)

Thinking

Eating

Poisoned



Examples

- Is initial marking a home marking ?
  - AG (EF (g)) , g: current state=initial marking

```
val myASKCTLformula =
INV(POS(NF("initial marking",IsInitialMarking))) ;
```

- Is a transition t live ?
  - AG (EF (g)), g: transition t is enabled in current state

```
val myASKCTLformula =
INV(POS(MODAL(AF("(take,<p=ph(2)>)",IsConsideredBE))));
```