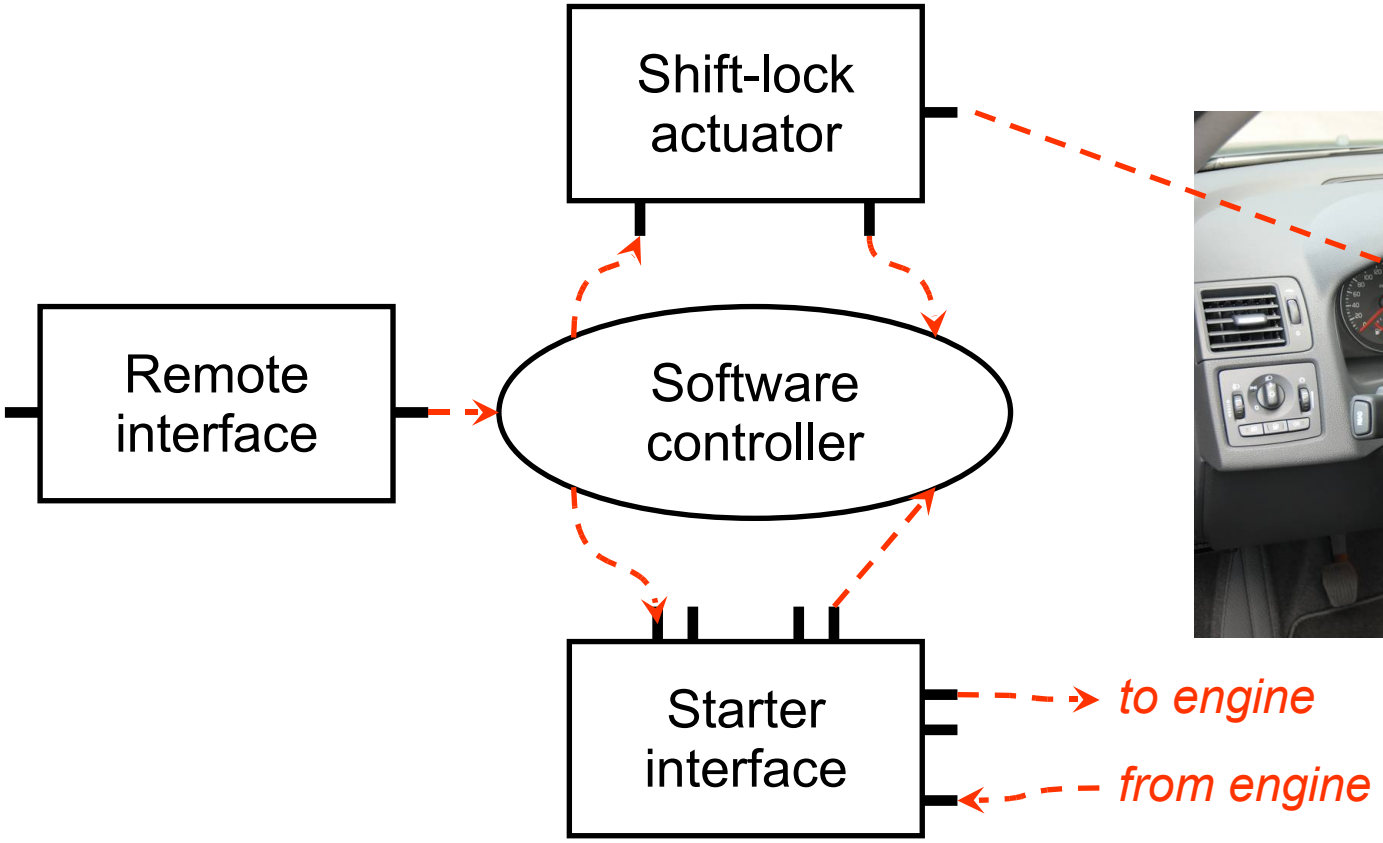
Systems Modeling

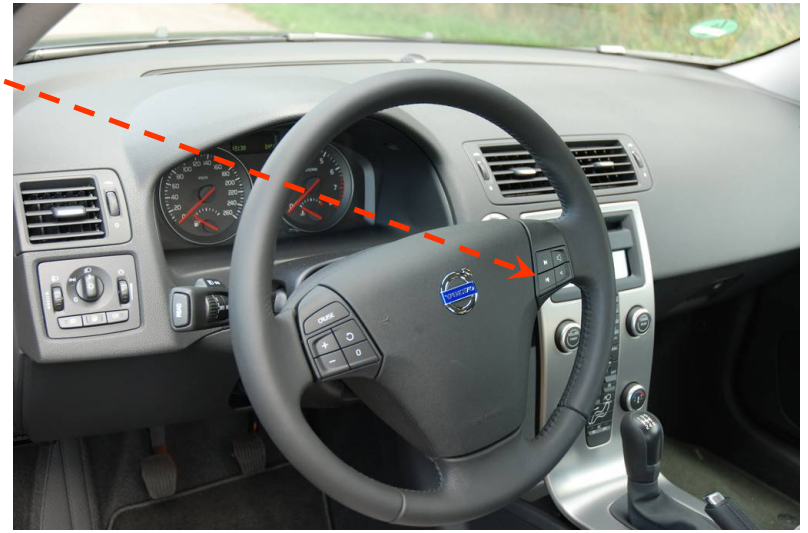
Lecture
2009/03/05

State Diagrams

- Most slides adapted or taken from Eileen Kraemer (University of Georgia)
- Compare "Object Oriented Modeling and Design with UML", M. Blaha et al., 5th Chapter

Locking System Car





Picture taken from kberberi, flickr

Potential Problems

- Controller
  - in non *receptive* state
    - signals arrive but no effect
    - preventing issuing of signals
      (greyed out buttons in dialog box)
  - receptive to some signals, but not those that are being offered
  - expecting a peer to be in a state that is
    ready to receive a signal,
    sends the signal,
    → peer is not ready

Potential Problems (continued)

Bad news:

Most of these problems *cannot* be reliably detected and fixed via testing

- Some are “race conditions”
  - Depend on how the various actors are scheduled by OS
  - Difficult to reproduce
  - Instrumentation (for diagnosis) may make them go away! → Heisenbugs!
- Very difficult to simulate all possible interactions with an environment

How can these problems be solved?

- *Finite state modeling and analysis* of software architectures
  - Model each entity in the system as a communicating finite state machine
  - Simulate interactions between state machines, looking for flaws
- *Model checking:* Exhaustive, formal analysis of a system specified in this fashion

Finite-State Models

Describe temporal/behavioral view of a system

Specify *control:*

- Sequence operations in response to stimuli
- Distinguish *states, events,* and *transitions*
- Especially useful during design

Lots of variants:

- E.g., StateCharts, UML state diagrams
- E.g., FSP (textual notation)

Key terms

Event: occurrence at a point in time
- instantaneous
- often corresponds to verb in past tense
  - e.g., alarm set, powered on, closed, opened
- or onset of a condition
  - e.g., paper tray becomes empty, temperature drops below freezing

State: behavioral condition that persists in time
- often corresponds to verbs with suffix of “-ing”
  - e.g., *Boiling, Waiting, Dialing*
- in OO terms: an abstraction of values of attributes and configuration of objects

Transition: instantaneous change in state
- triggered by an event

State diagrams

Graphical state-modeling notation:

- States: labeled roundtangles
- Transitions: directed arcs, labeled by triggering event, guard condition, and/or effects

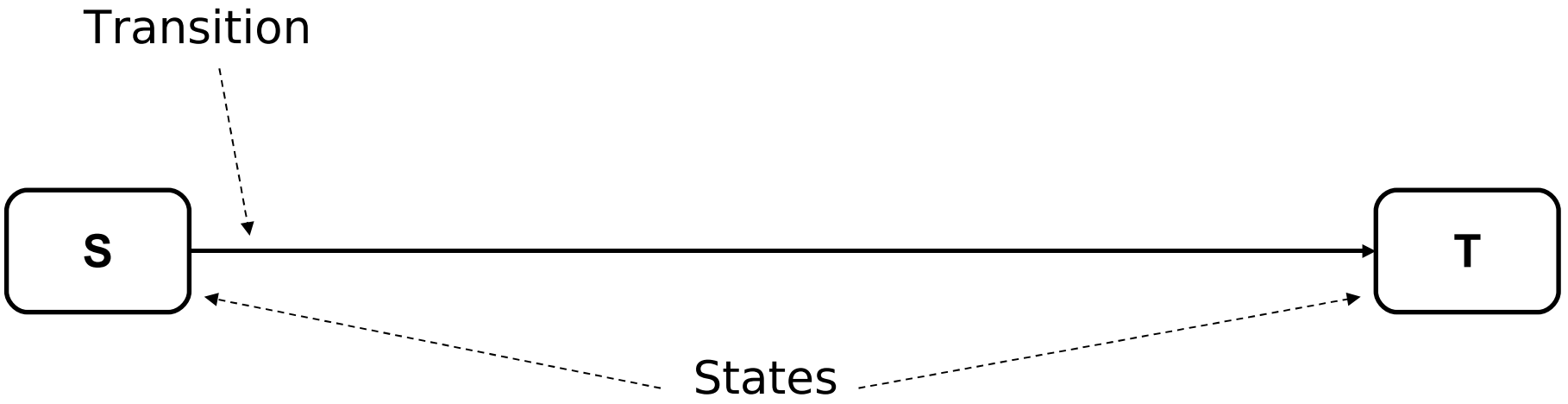
Example:

State diagrams

Graphical state-modeling notation:

- States: labeled roundtangles
- Transitions: directed arcs, labeled by triggering event, guard condition, and/or effects

Example:

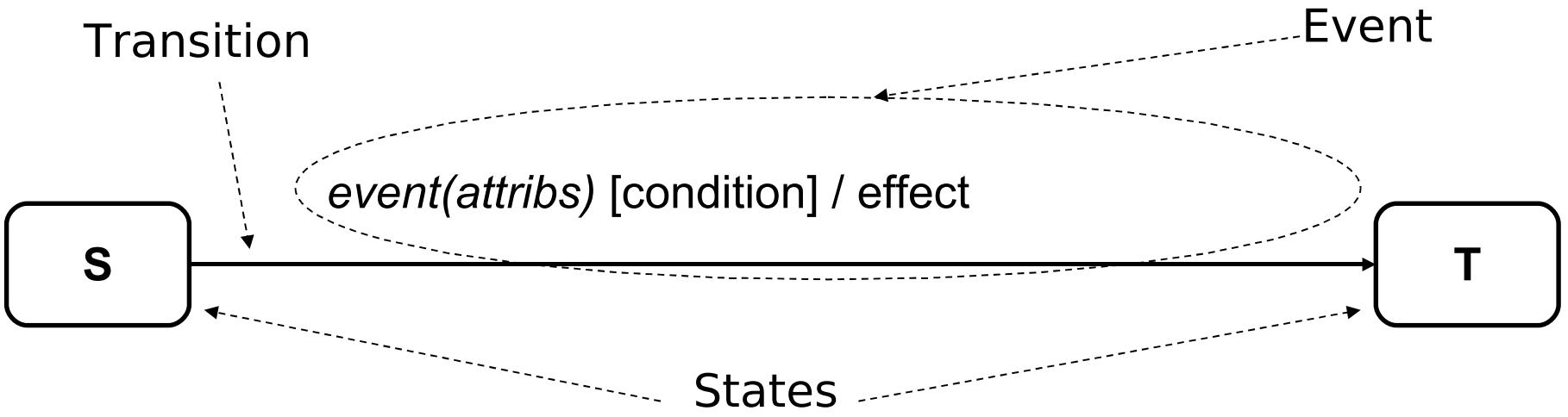


State diagrams

Graphical state-modeling notation:

- States: labeled roundtangles
- Transitions: directed arcs, labeled by triggering event, guard condition, and/or effects

Example:



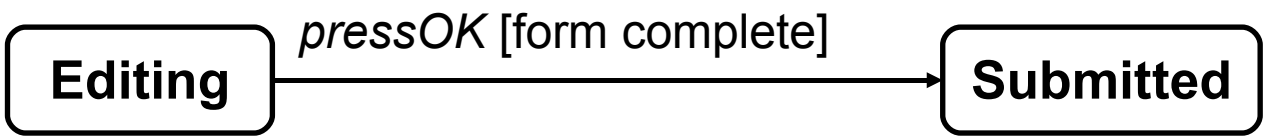
Enabling and firing of transitions

Transition is:

- *enabled* when source state is *active* and guard condition *satisfied*
- *fires* when enabled and the triggering event occurs

Example below:

- enabled when current state is Editing and the form is complete
- fires when the user presses the “OK” button



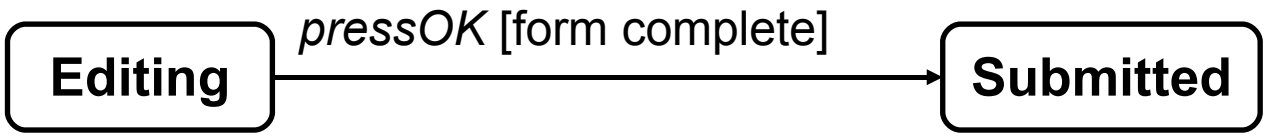
Enabling and firing of transitions

Transition is:

- *enabled* when source state is *active* and guard condition *satisfied*
- *fires* when enabled and the triggering event occurs

Example below:

- enabled when current state is Editing and the form is complete
- fires when the user presses the “OK” button



Question: What happens if user presses “OK” when transition not enabled?

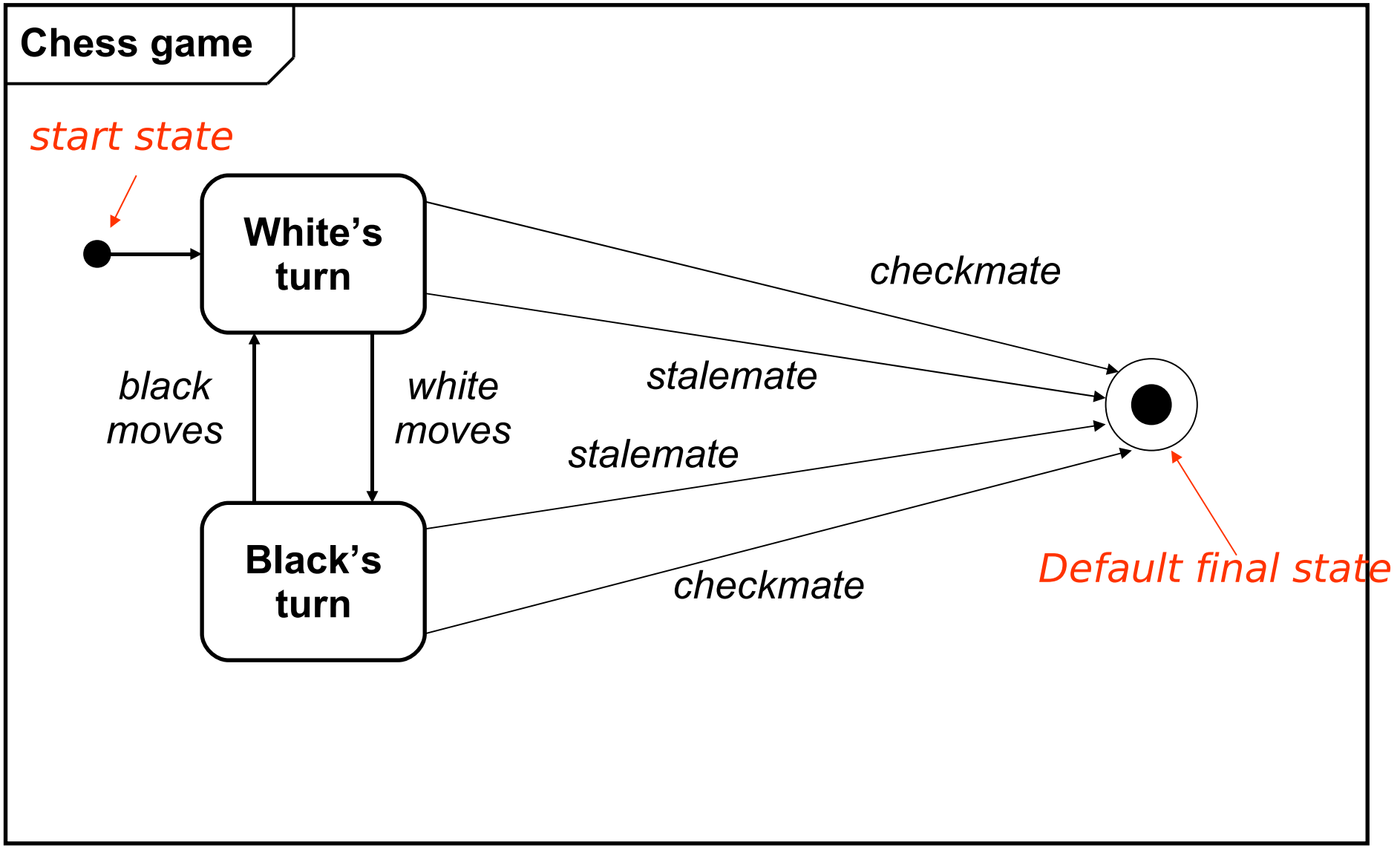
Guard Condition

- Boolean expression that must be true for transition to occur
- Checked only once, at time event occurs; transition fires if true

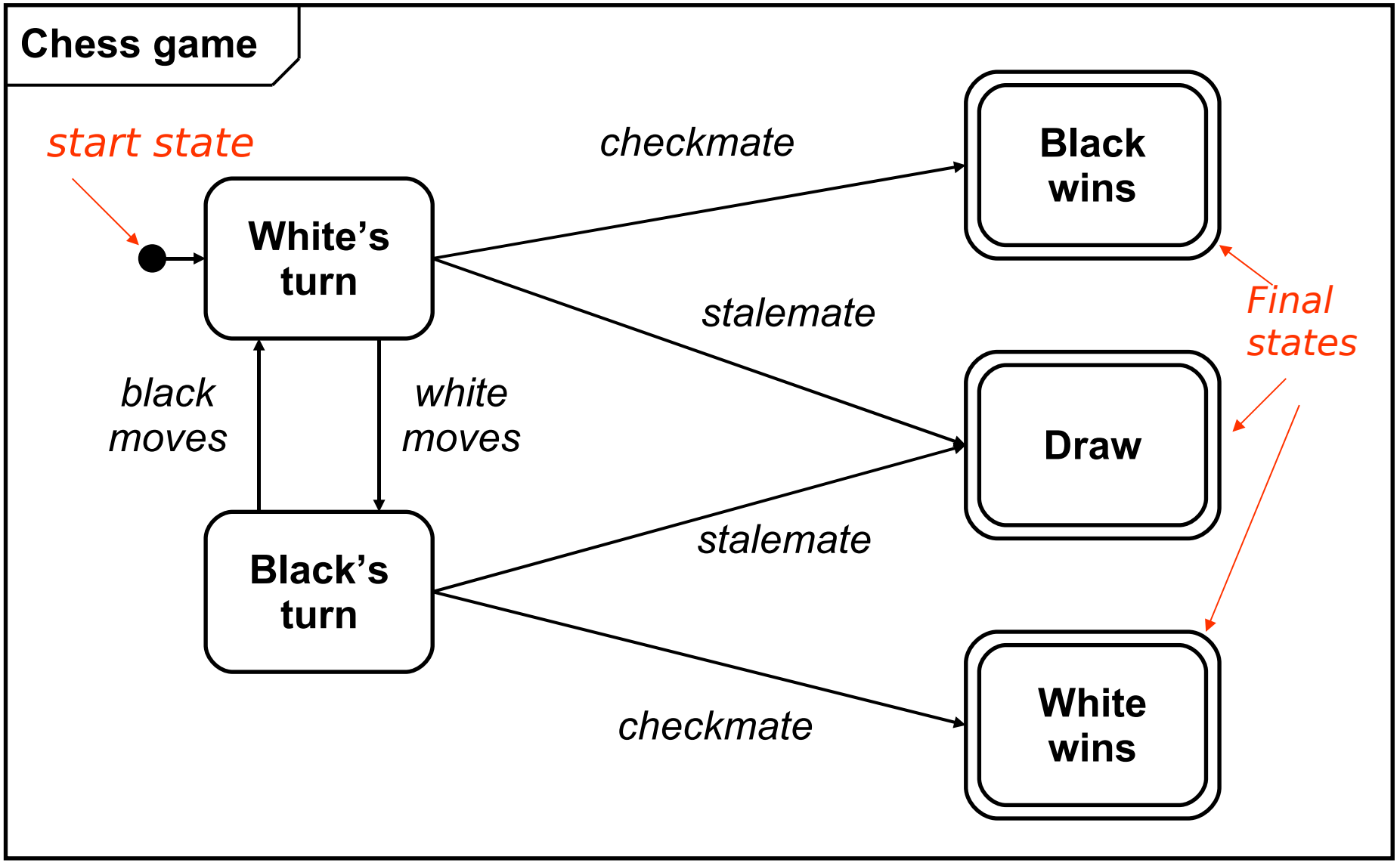
“One-shot” state diagrams

- represent objects with finite lives
  - have initial and finite states
- initial state - entered on object creation
- final state - entry implies destruction of object

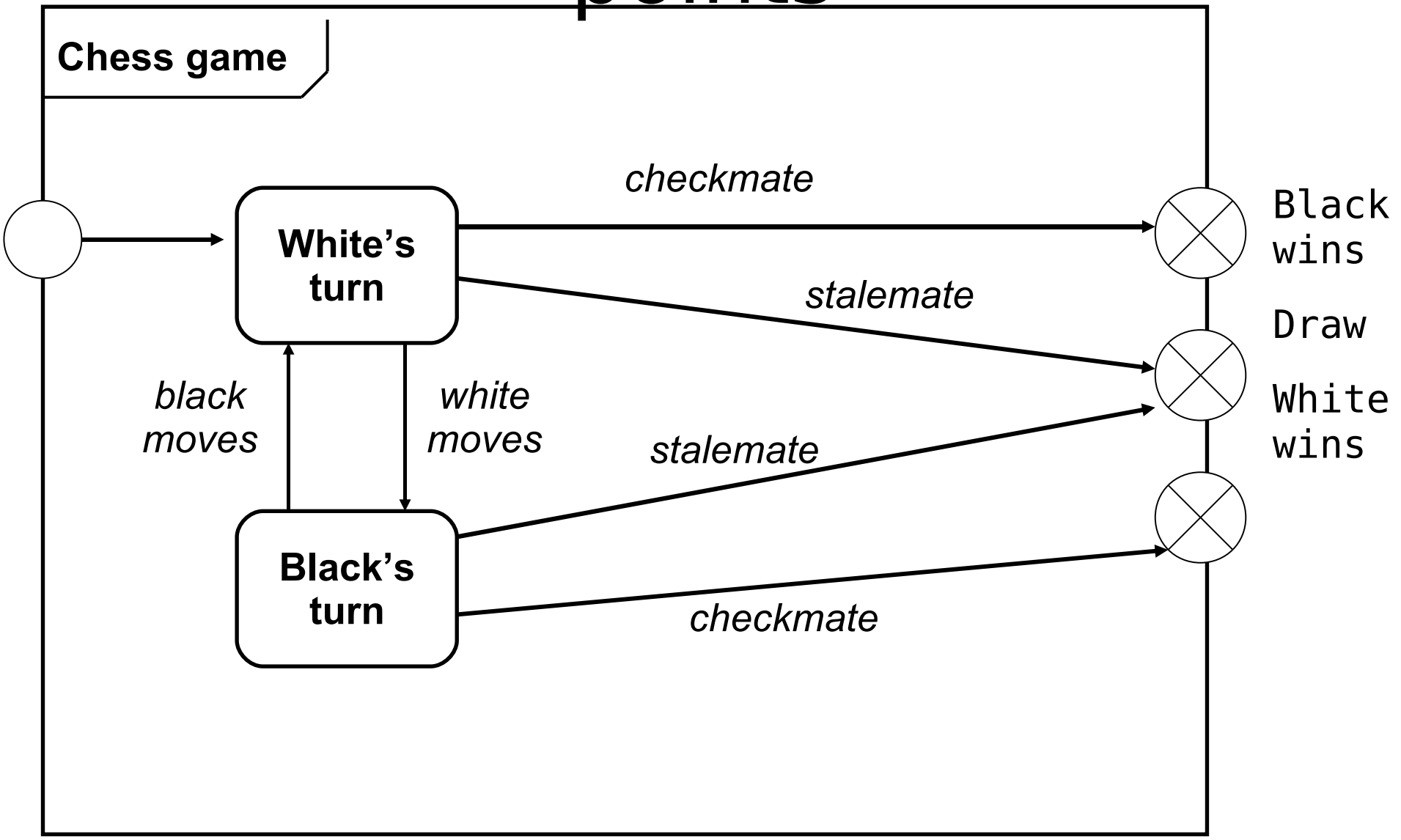
Example


Chess game
start state
White's turn
checkmate
stalemate
black moves
white moves
stalemate
Black's turn
checkmate
Default final state

Example


Chess game
start state
White's turn
checkmate
Black wins
stalemate
Final states
black moves
white moves
Draw
Black's turn
stalemate
checkmate
White wins

Example - entry and exit points



Elsim Elevator Door

Events

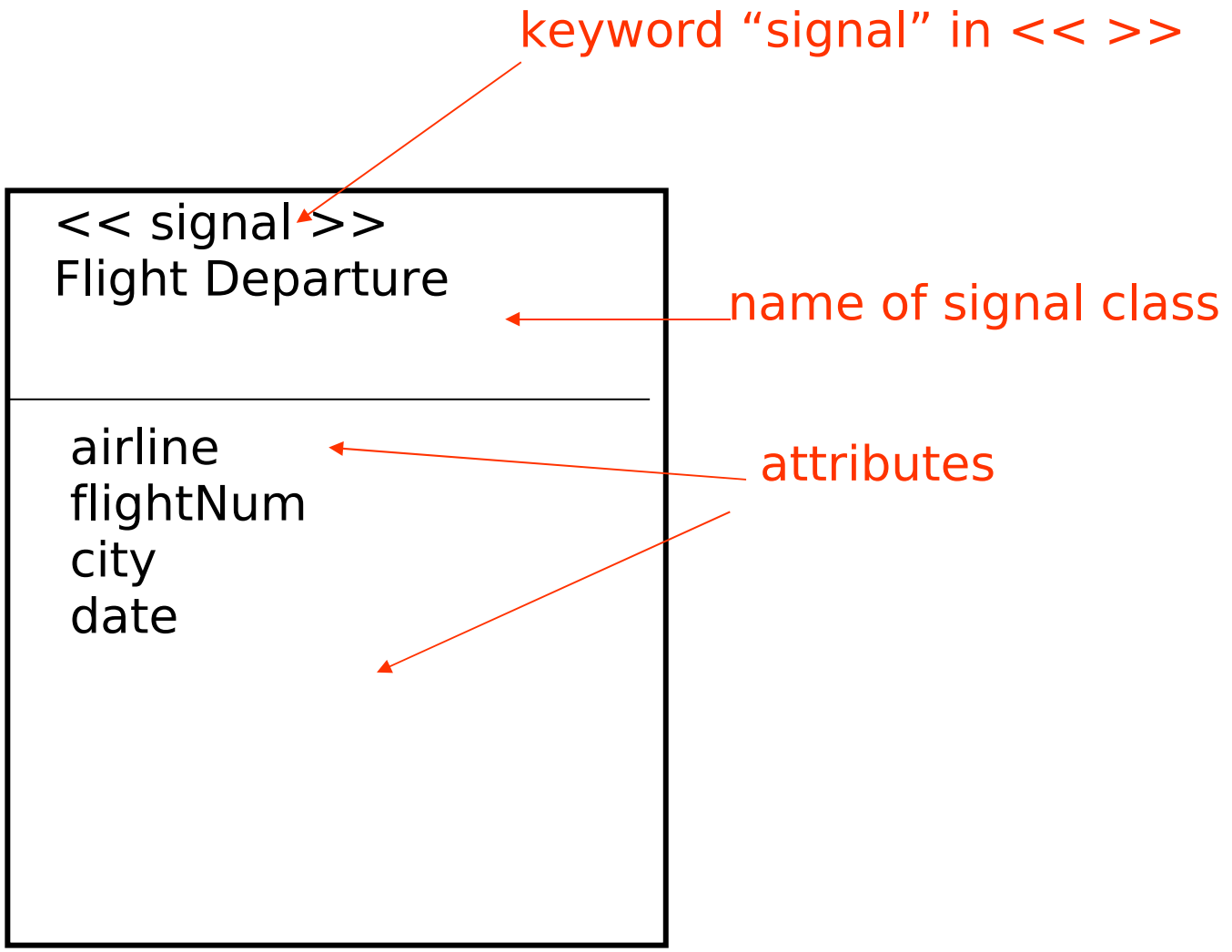
Event : occurrence at a point in time

Concurrent events : causally unrelated; have no effect on one another

Kind of Events: Signal Event

- the sending or receiving of a signal
  - an explicit one-way transmission of information
  - may be parameterized
    - E.g., *stringEntered(“Foo”)*
- Sending of a signal by one object is a distinct event from its reception by another

Signal class - UML notation



More Kinds of Events

- ***Change event***
  - Caused by satisfaction of a boolean expression
  - Intent: Expression continually tested; when changes from false to true, the event happens
  - Notation**: when**(val1 < val2)
- ***Time event***
  - Caused by the occurrence of an absolute time or the elapse of a time interval
  - **when** (time = some_time)
  - **when** (date = some_date_
  - **after** (n time_units)

State Model

- Multiple state diagrams, one for each class with important temporal behavior
  - diagrams interact by passing events and through side effects of guard conditions
  - events and guard conditions must match across diagrams in the model

Effect

- *effect* = a behavior executed in response to an event
  - can be attached to a transition or a state
  - listed after a “/”
  - multiple effects separated with a “,” and are performed concurrently

Activity Effects

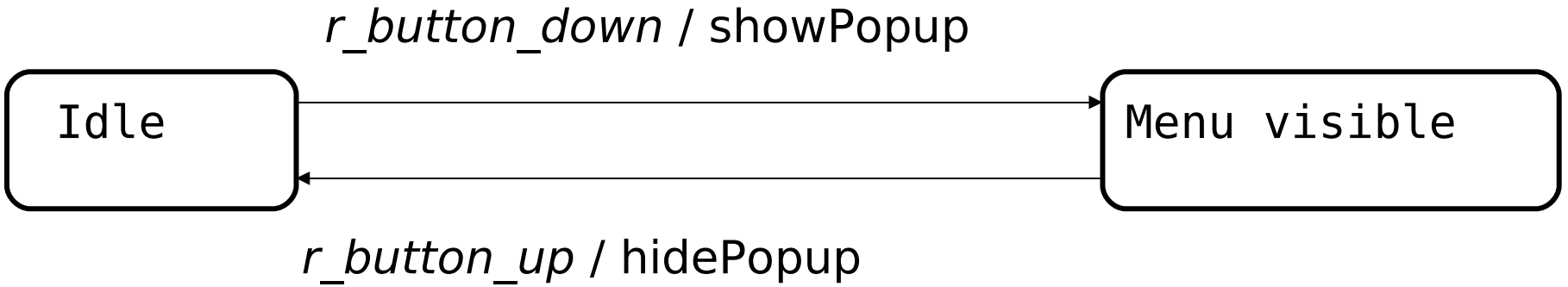
- *Activity* = behavior that can be invoked by any number of effects
- May be performed upon:
  - a transition
  - entry to or exit from a state
  - some event within a state
- Notation:
  - *event* / resulting-activity

Activities

Often useful to specify an *activity* that is performed within a given state

- E.g., while in **PaperJam** state, the warning light should be flashing
- E.g., on entry into the **Opening** state, the motor should be switched on
- E.g., upon exit of the **Opening** state, the motor should be switched off

Activity effects



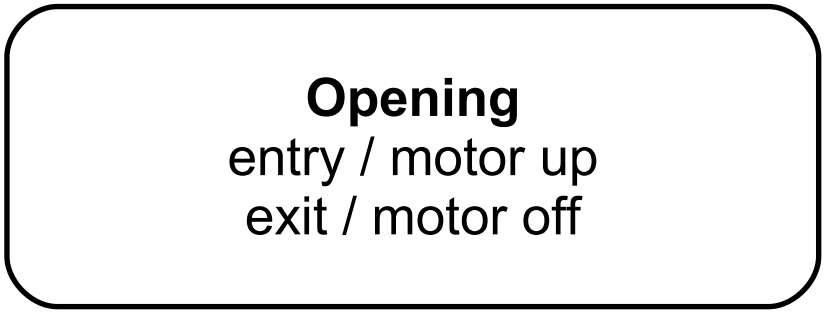
Do-Activities

- continue for an extended time
- can occur only within a state
- can not be attached to a transition
- may be performed for all or part of time object is in state
- may be interrupted by event received during execution; event may or may not cause state transition

PaperJam
do/ flash warning light

Entry and Exit Activities

can bind activities to entry to/ exit from a state

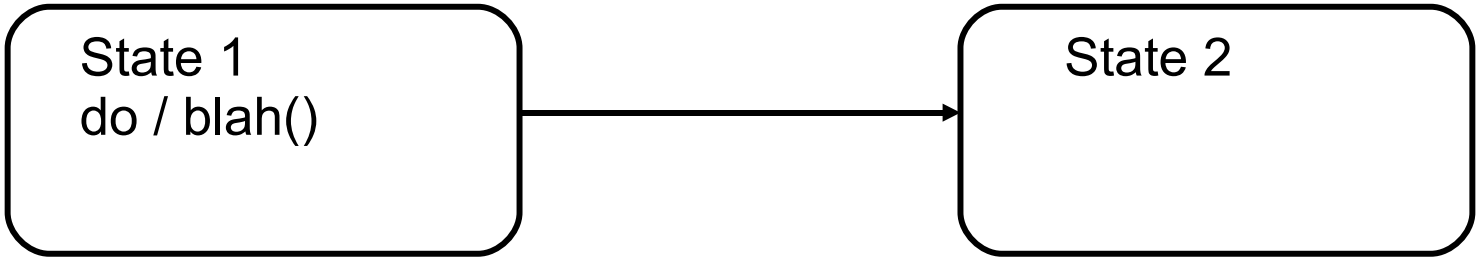


Order of activities

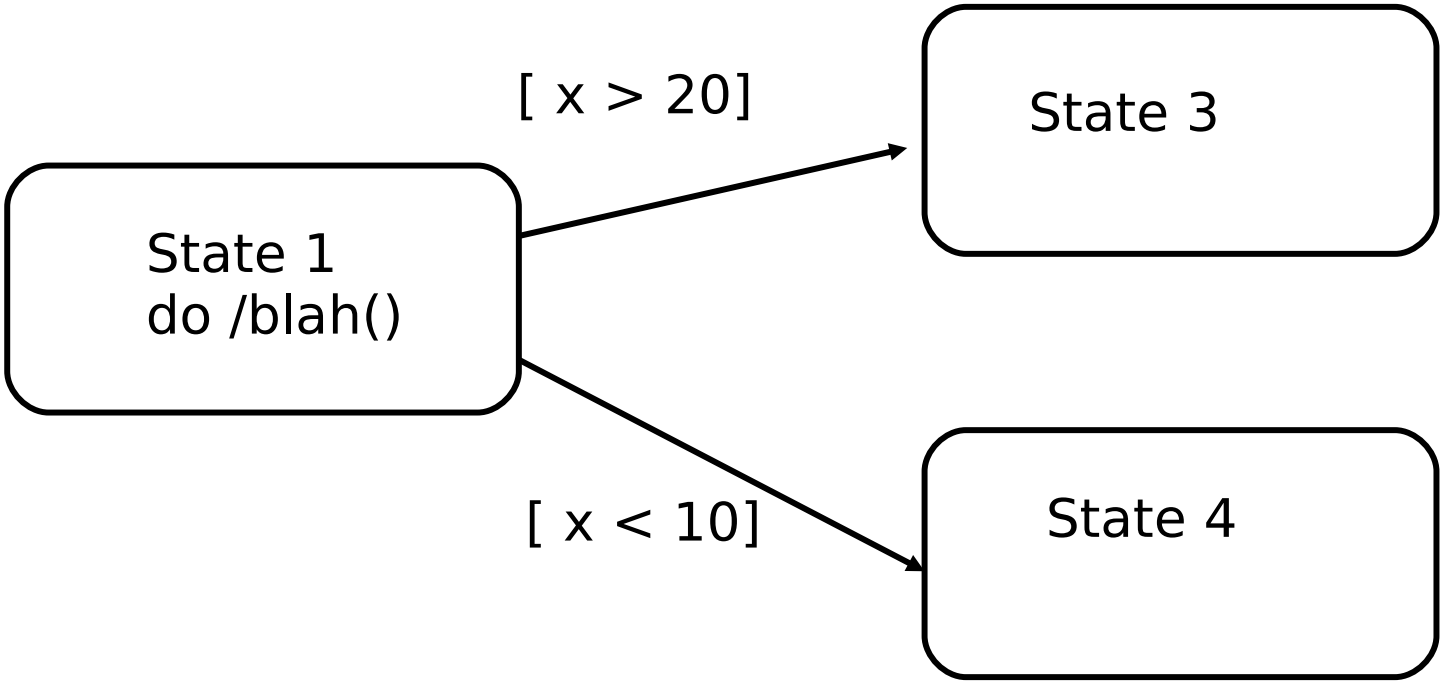
1. activities on incoming transition
2. entry activities
3. do-activities
4. exit activities
5. activities on outgoing transition

Completion Transition

- triggered by completion of activity in the source state



Potential problem ...

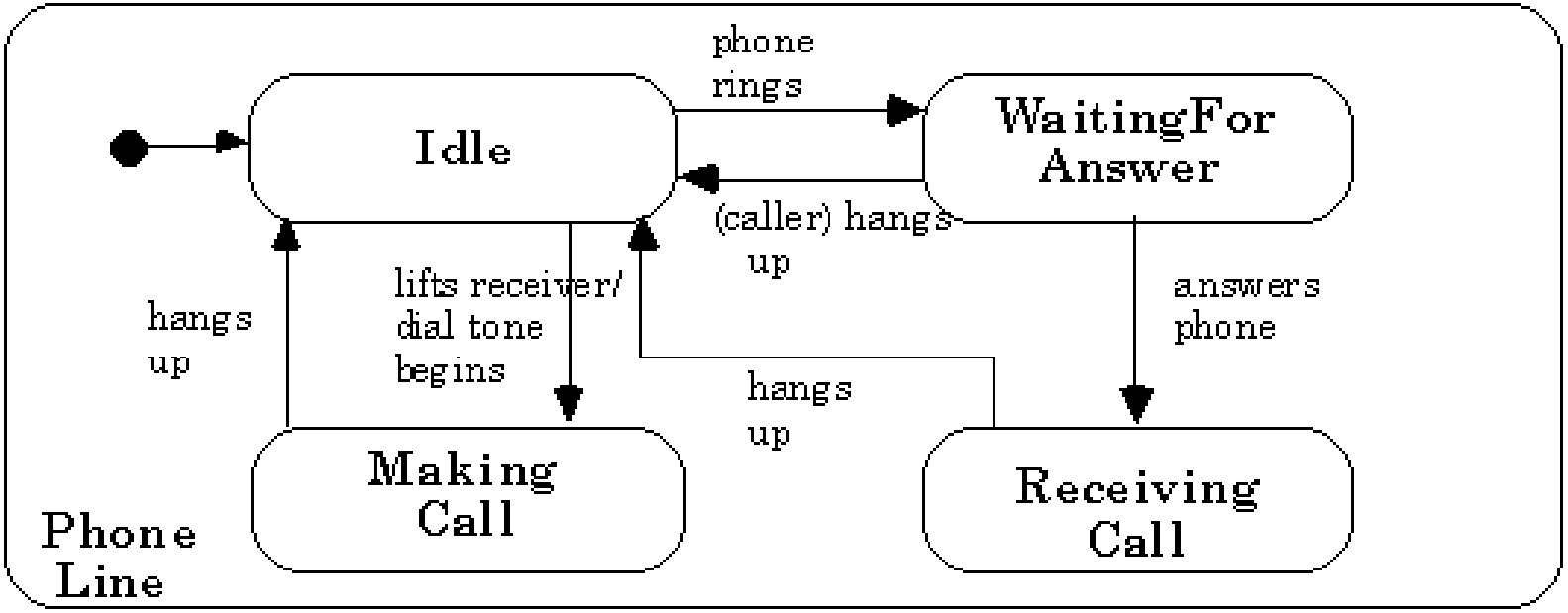


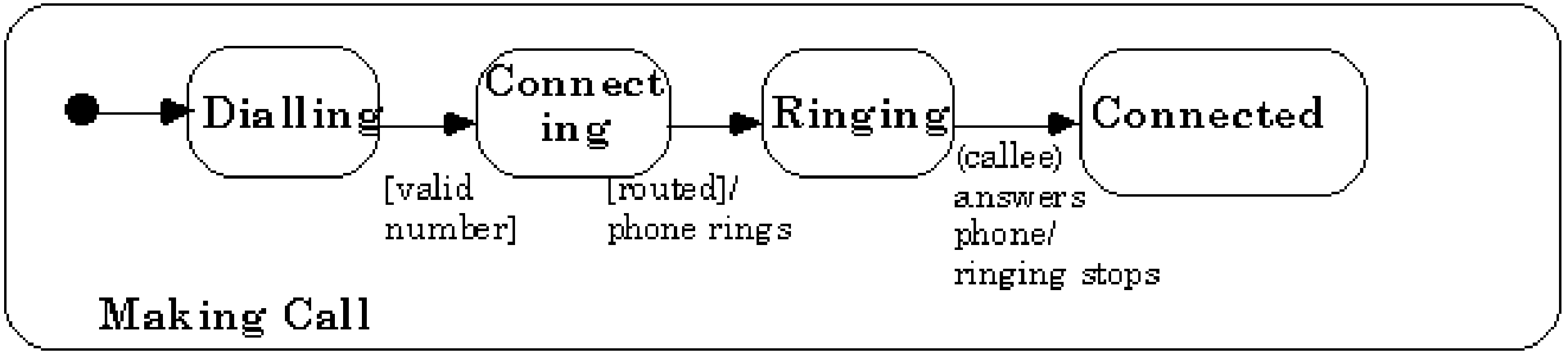
if (x ==12) when blah() completes??

Problems with FSMs

- Difficult to read with lots of states and transitions
- Two sources:
  - Multiple transitions with same triggering event, guard condition, and response but different source and/or target states
  - *State explosion* due to concurrency and/or orthogonality
- A bit better by modularity features:
  - State generalization
  - Parallel composition

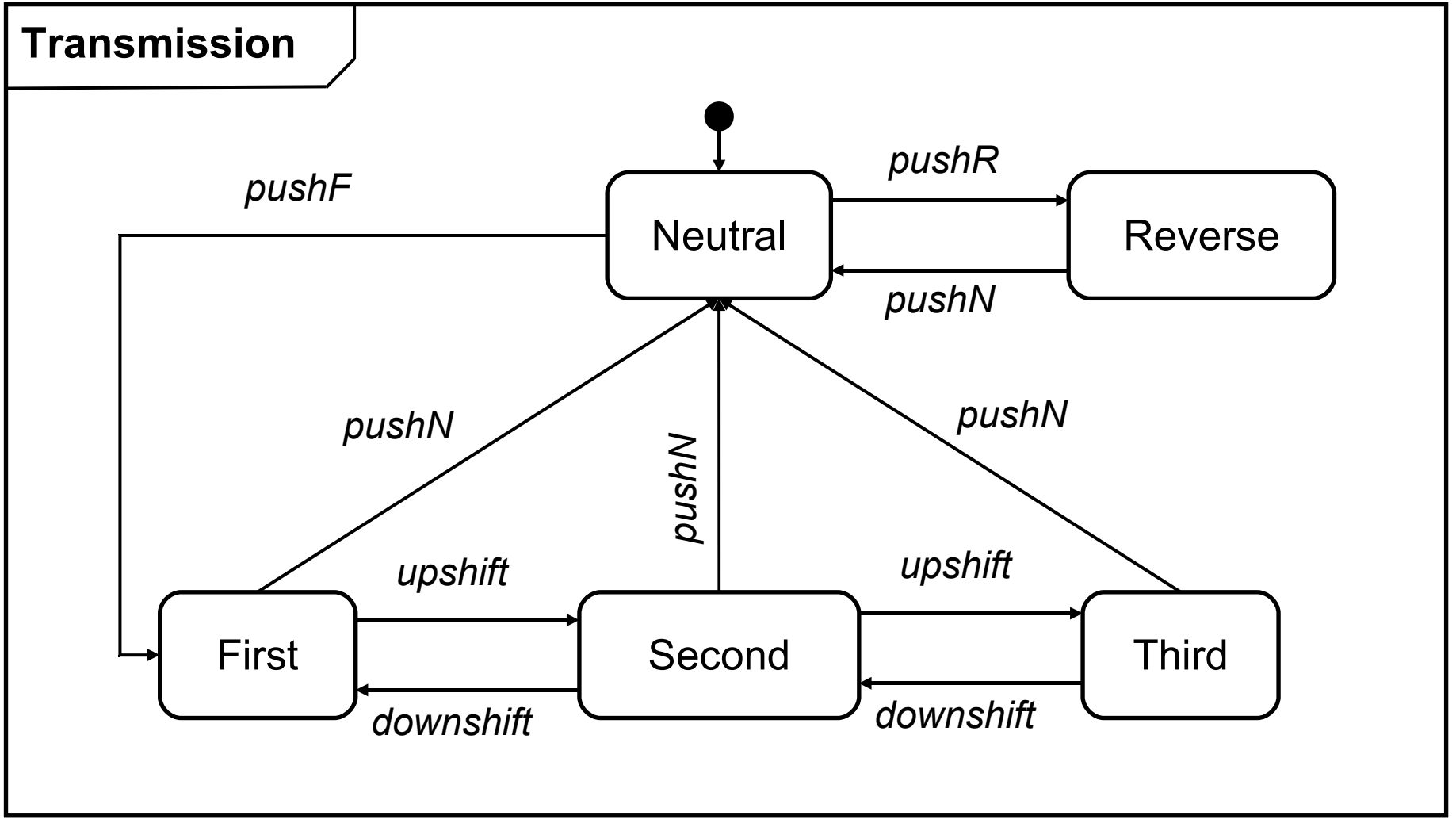
Phone Line example



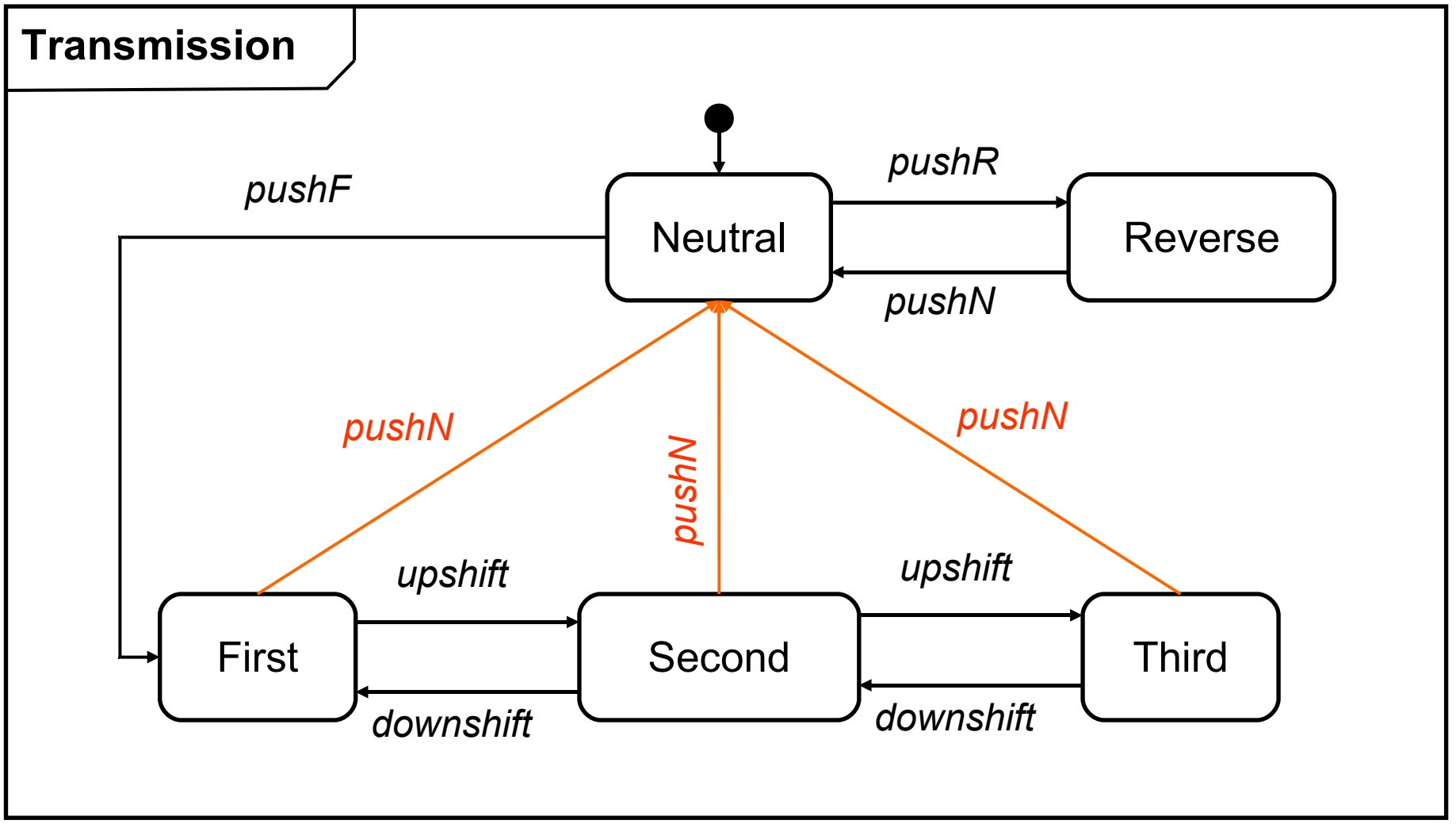


Taken from M. Woods, University of Hertfordshire Higher Education Corporation

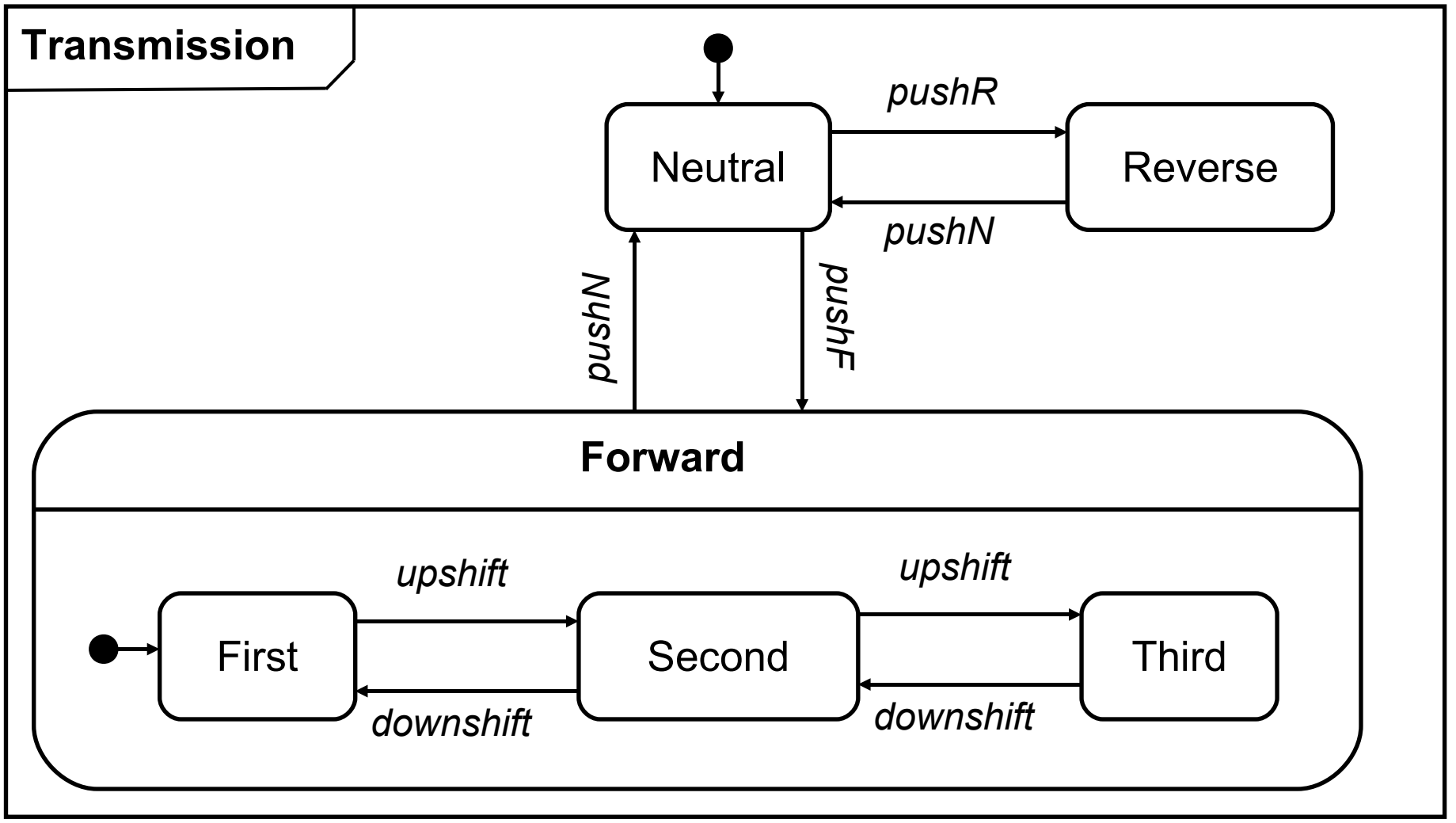
Example: Automatic transmission



Problem: Multiple similar transitions



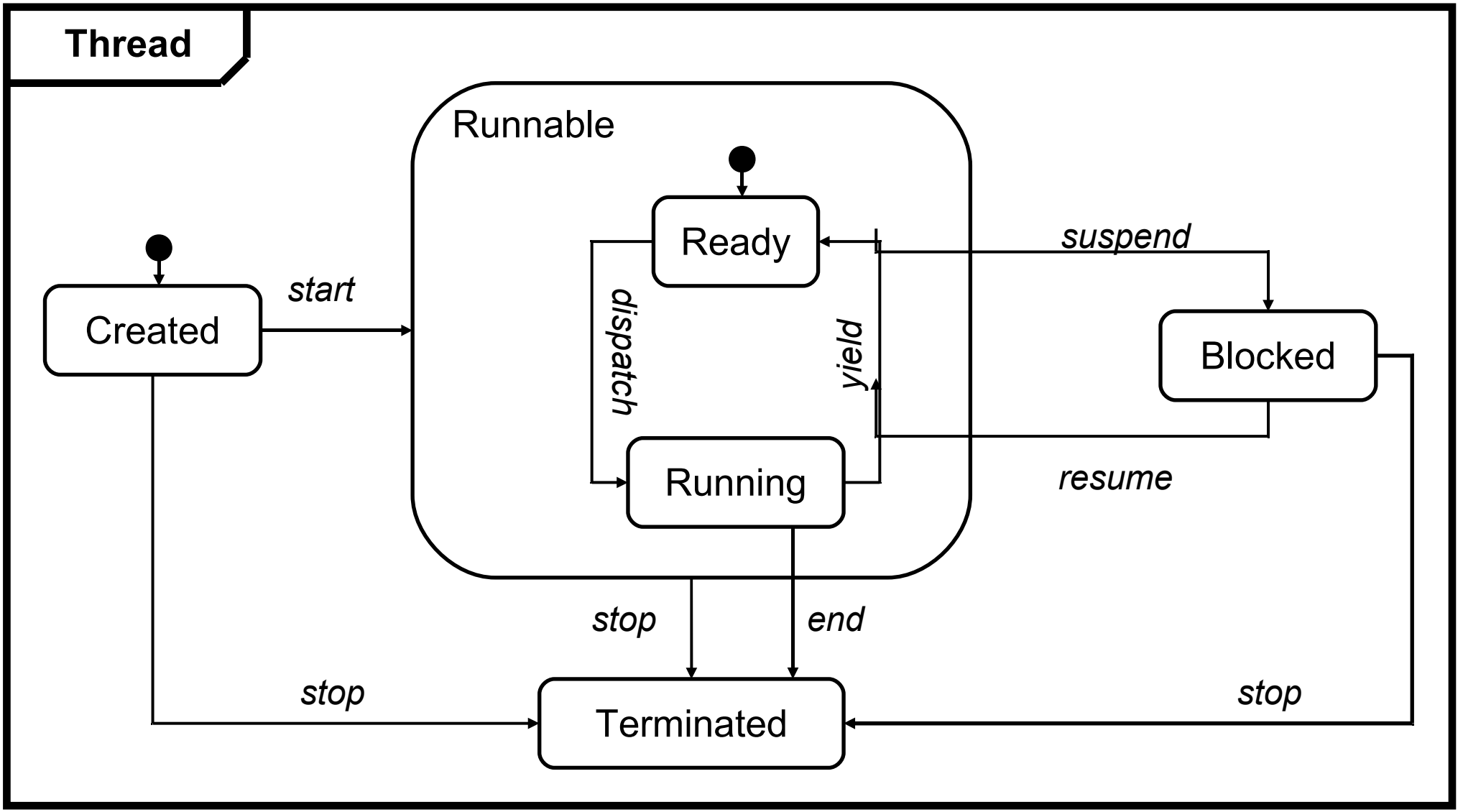
Solution: State generalization



State generalization

- Introduces an abstract “super state”:
  - decomposes into multiple sub-states
  - when super state is active, exactly one of its sub-states is active
- Outbound transition incident on super-state abbreviates set of transitions, one from each sub-state
- Inbound transition incident on super-state enters sub-state that is distinguished as the start state

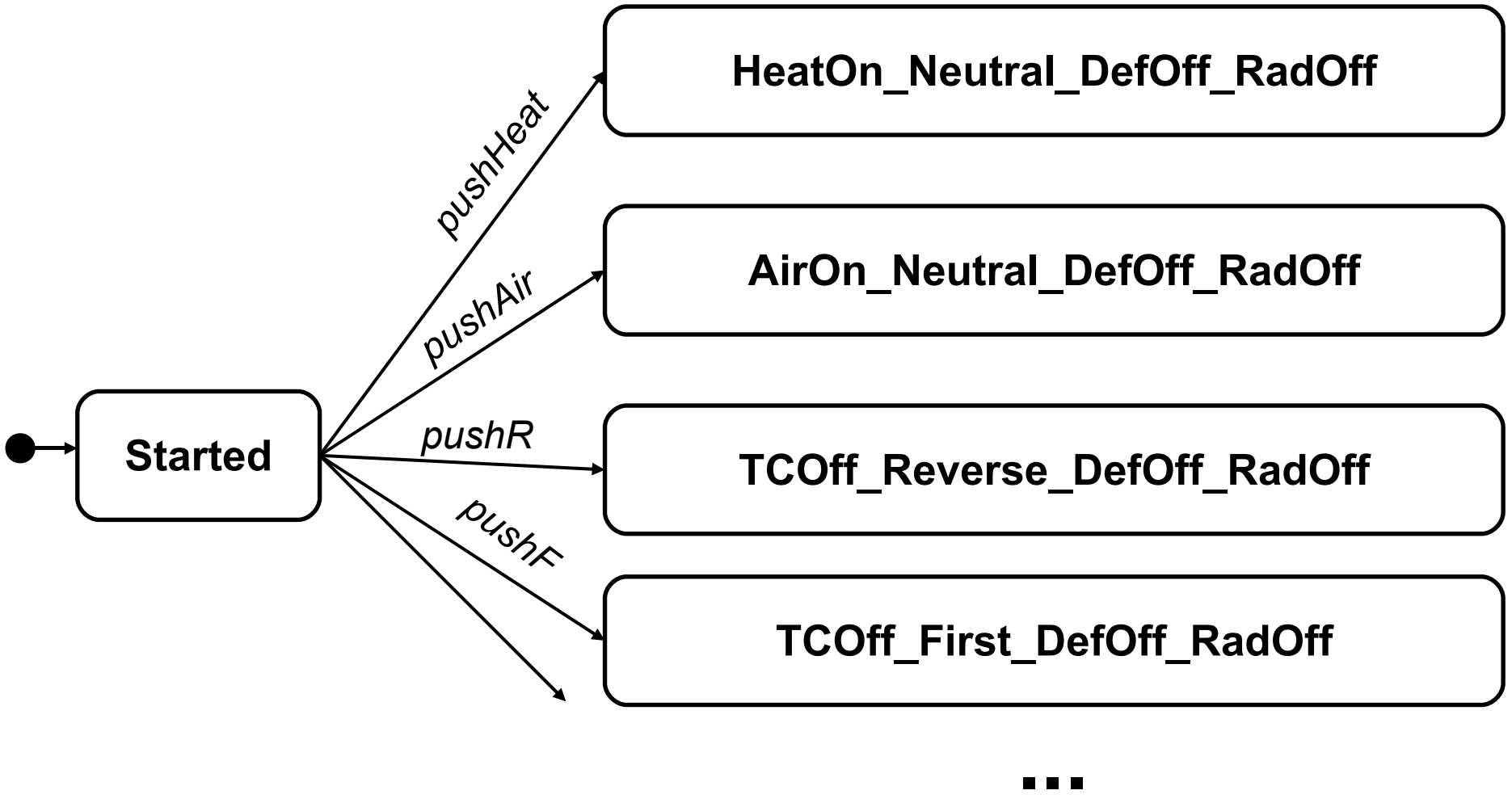
Example: Lifecycle of a thread



Problem: Composite behaviors

- Consider an automobile with multiple options:
  - Automatic transmission
  - Temperature control (heating/air)
  - Rear-window defroster
  - Stereo system
- Suppose we wish to construct a state diagram for the automobile:
  - Assume car starts with transmission in neutral and temp control, rear defroster, and stereo are all off
  - What are the possible next states?

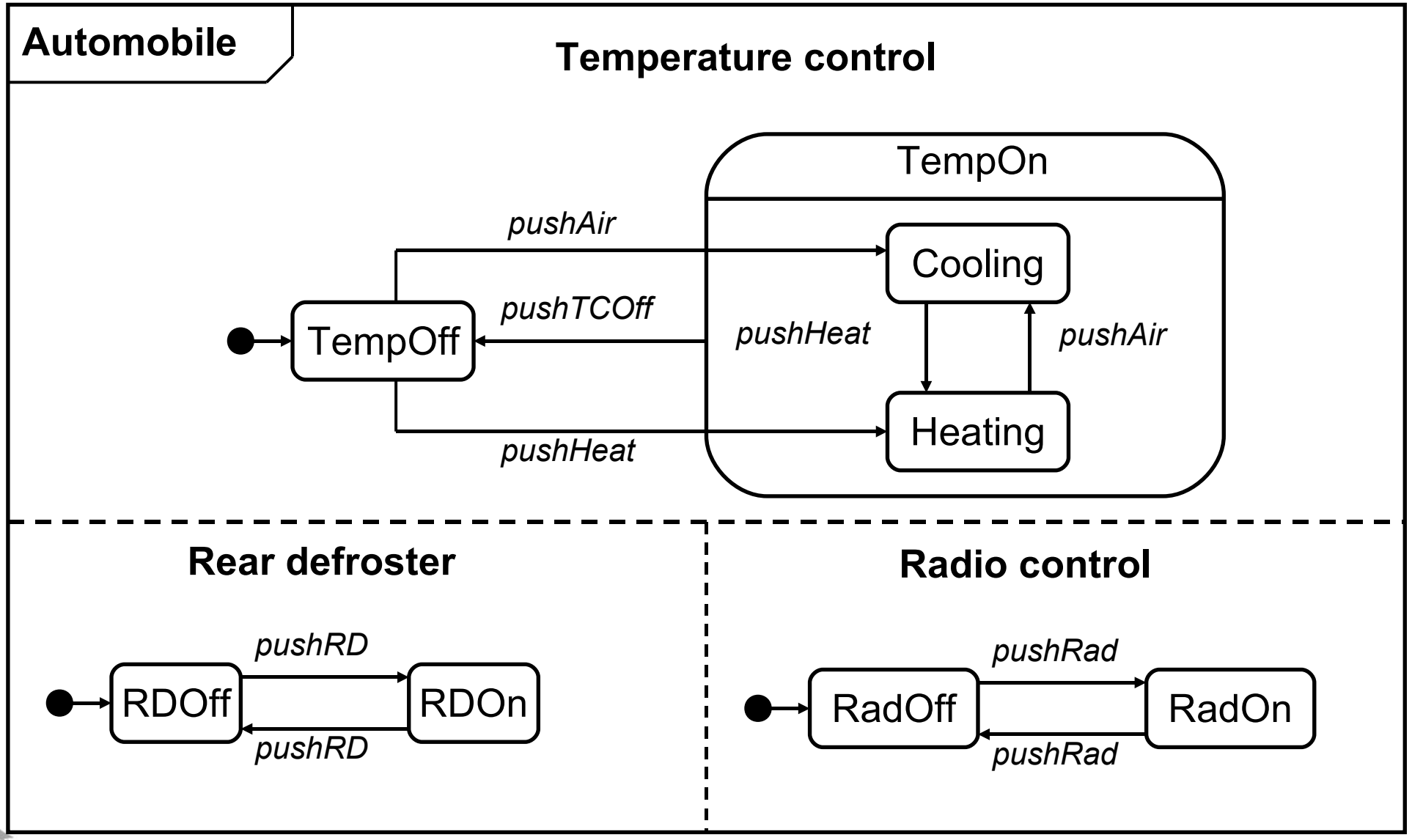
Example: Automobile states



State explosion problem

- Number of states in a composite diagram is product of the number of states in component diagrams
- Major impediment to understanding:
  - Impossible to visualize in any meaningful way
  - Requires the use of analysis tools to *verify properties*
- Managing state explosion:
  - Concurrent state diagrams
  - Highly effective when diagram can be separated into truly orthogonal components

Example


Automobile
Temperature control
TempOn
pushAir
Cooling
pushTCOff
TempOff
pushHeat
pushAir
Heating
pushHeat
Rear defroster
pushRD
RDOff
RDOn
pushRD
Radio control
pushRad
RadOff
RadOn
pushRad

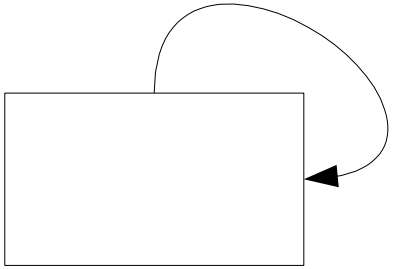
Semantics of parallel composition

Multiple interpretations:

- Concurrent regions execute independently
  - What happens if transitions in different regions are triggered by same event?
  - Do both execute simultaneously? Does one "consume" the event to the exclusion of the other?
- Concurrent regions communicate with one another, synchronizing on common events
  - Regions can only proceed when all are ready to proceed
  - Regions transfer data values during a concurrent transition
- Do we distinguish *internal* and *external* events?

Code Generation & Visual Prototyping

Code Generation



- Test: Self-reflective class
- Argo-UML
  - Nice for drawing, Code generation only very little
- Eclipse 3.4 with UML 2.0 Tools
  - Design does not work, totally unintuitive
- Netbeans
  - Very nice, some annoying bugs
- Visual Studio
  - Very robust, limited expressiveness
- Fujaba
  - Excellent Code generation
  - Extremely buggy, very hard to install
- Standard might be
  - Rose (not tested), Visual Paradigm

Fujaba

- From UML to Java and back again
- Initially from Prof. Albert Zündorf
- Active Community in Germany
- Supports also other languages
- Currently shift to Fujaba4Eclipse (Code generation does not work)
- Presenting here version 5.04

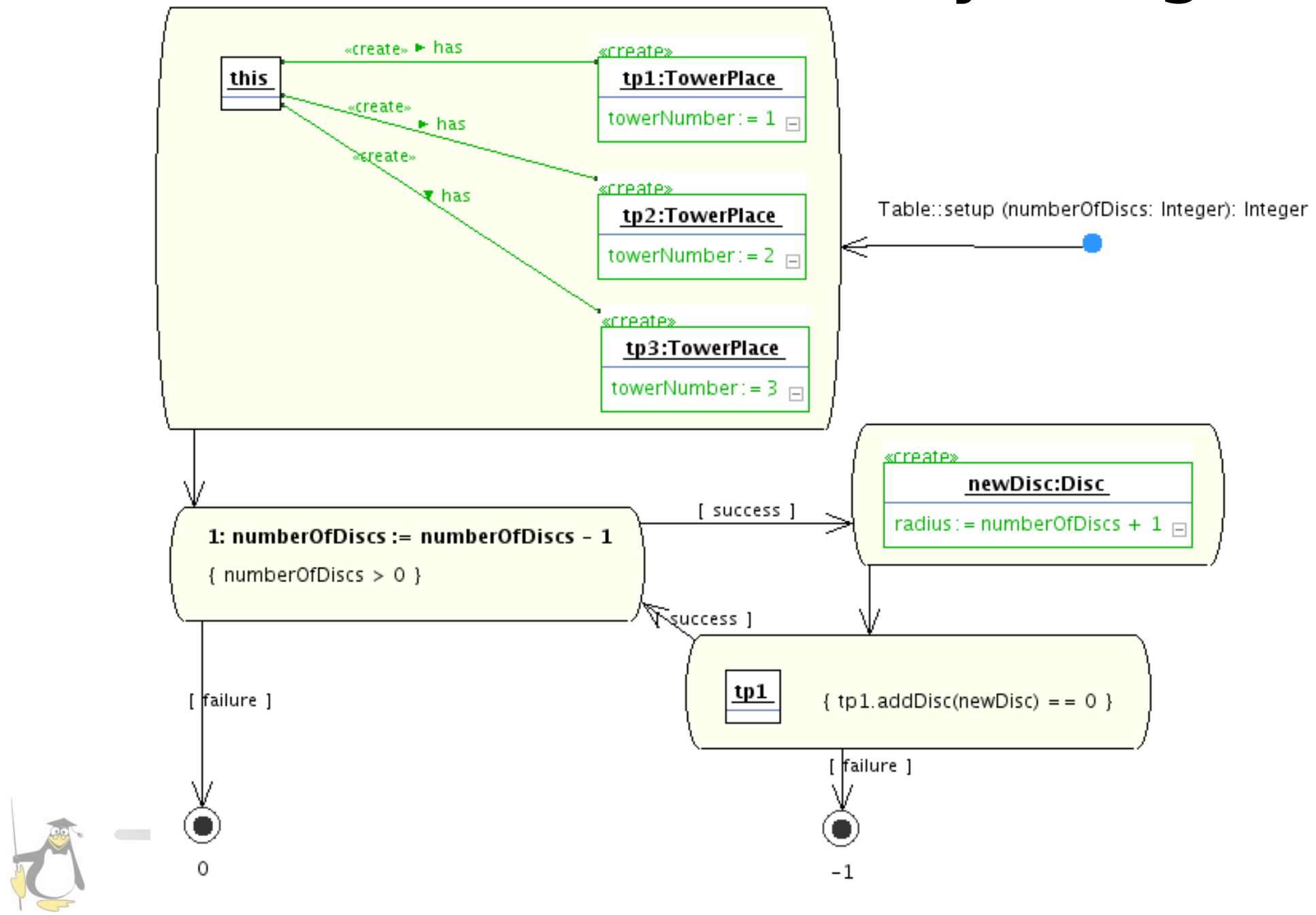
Demo

- Reflective Class and corresponding Code

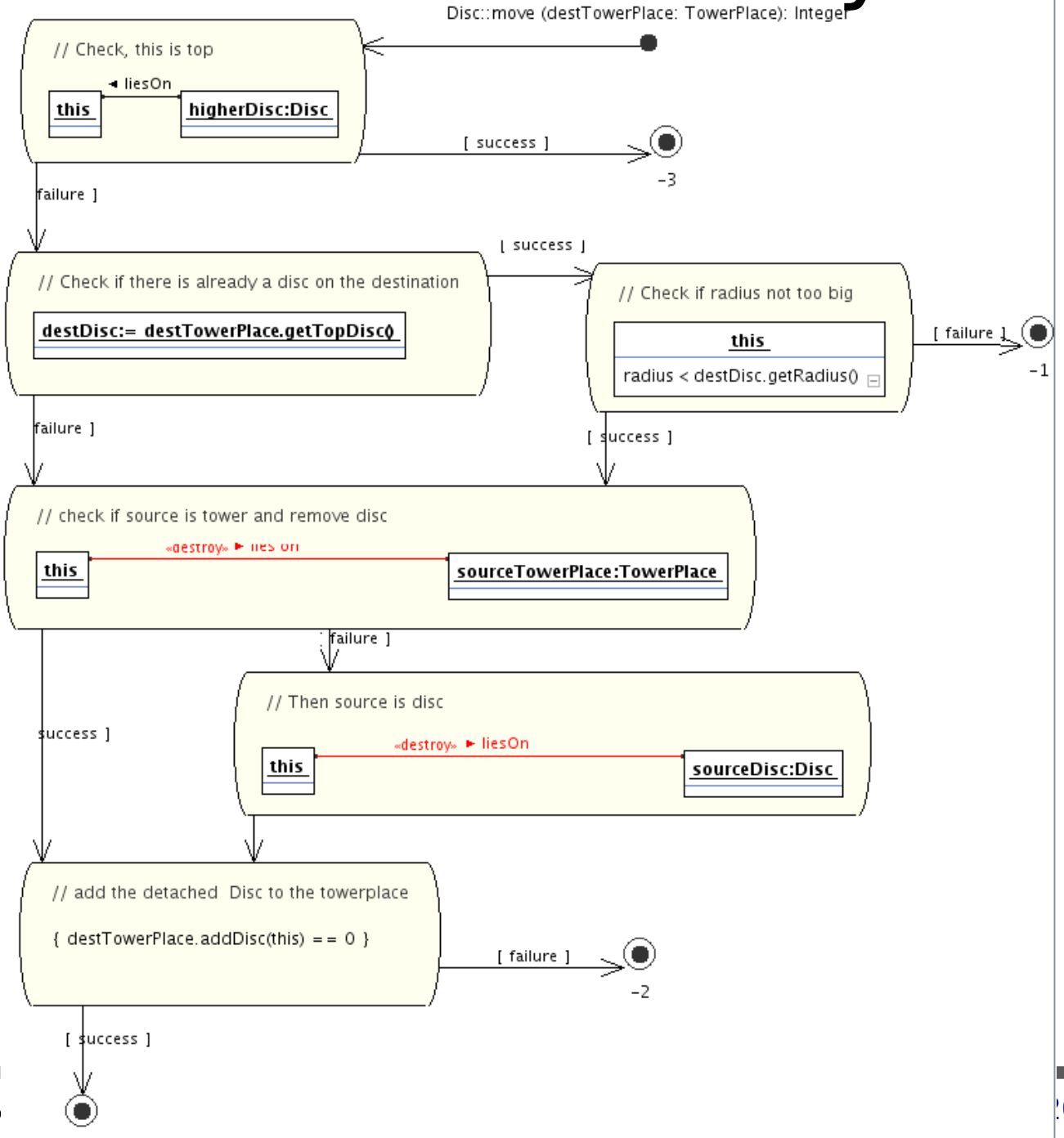
Story Diagrams

- Fujaba
- Merge of State Diagrams, Activity Diagrams, and Sequence Diagrams
- Allow to visually model code

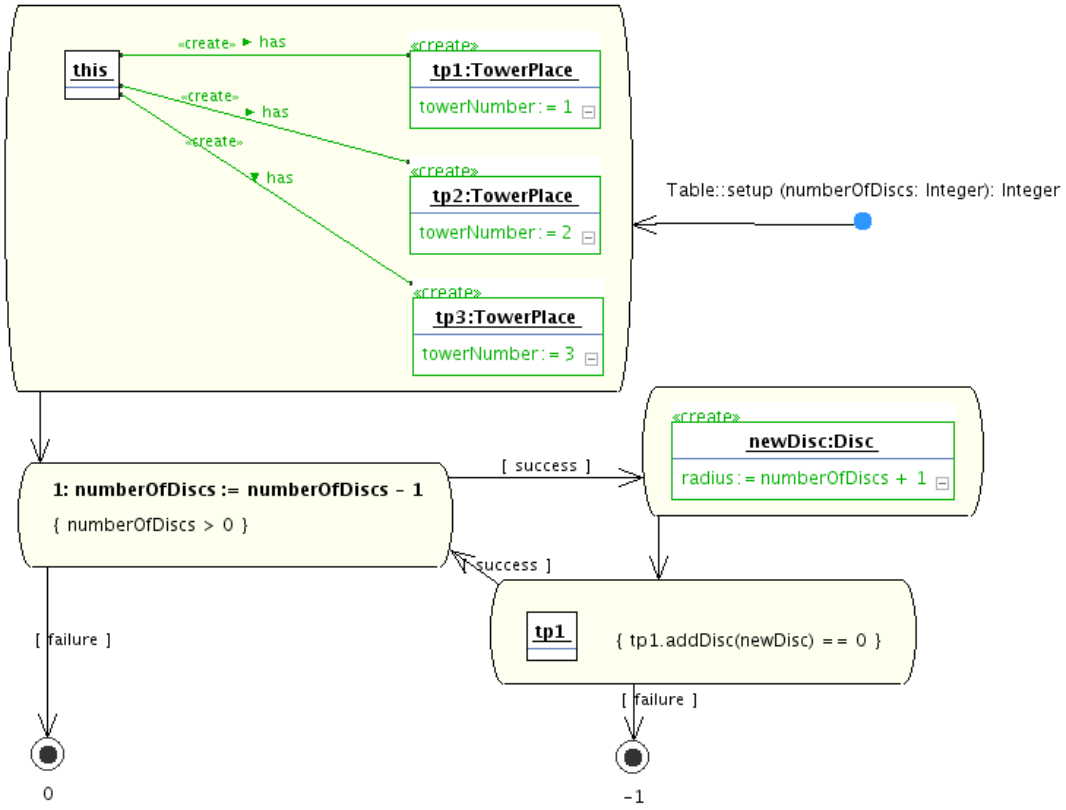
Some Elements of Story Diagrams



Some Elements of Story Diagrams



Story Diagram in Code + Demo



mpEDIT - Fujaba

File Edit Options Window Help

```
public int setup ( int numberOfDiscs)
{
   boolean fujaba__Success = false;
   TowerPlace tp1 = null;
   TowerPlace tp2 = null;
   TowerPlace tp3 = null;
   Disc newDisc = null;

   // story pattern
   try
   {
      fujaba__Success = false;

        // create object
      tp1 = new TowerPlace ( );
        // create object
      tp2 = new TowerPlace ( );
        // create object
      tp3 = new TowerPlace ( );
        // assign statement
      tp1.setTowerNumber (1);
       // assign statement
      tp2.setTowerNumber (2);
       // assign statement
      tp3.setTowerNumber (3);
        // create link
      tp1.setTable (this);

        // create link
      tp2.setTable (this);

        // create link
      tp3.setTable (this);

      fujaba__Success = true;
   }
   catch ( JavaSDMException fujaba__InternalException )
   {
      fujaba__Success = false;
   }

   do
   {
   // story pattern
   try
   {
      fujaba__Success = false;

      // constraint
      JavaSDM.ensure ( numberOfDiscs > 0 );
      // collabStat call
      numberOfDiscs = numberOfDiscs - 1;
      fujaba__Success = true;
```

Dynamic Object Browsing in Eclipse (eDobs) + Demo