LARVA – Logical Automata for Runtime Verification & Analysis

Christian Colombo
Gordon Pace
June 2009

A Day in the Life of a Software Developer

- We are given a very vague idea of what the client wants
- Which we try to understand and concretise
- And eventually transform into a real system
- Which the client uses ... and discovers bugs.

Bugs? So what!

- Well, there are bugs, and bugs
- We can live with certain bugs
  - A system crashing every so often, UI bugs, etc
  - Most of us still use Windows
  - Undesirable but acceptable
- It is more difficult to live with others:
  - A financial transaction handler may accidentally give an extra 10c to users whose name starts with a W.
  - A bug in your code may lead to sensitive data being shared with unauthorised users.

Reducing Bugs

- Usually we test against the specification before deployment.
  - "Testing shows the presence, not the absence of bugs" E. Dijkstra
  - What does coverage really mean?
- Research into automated analysis of programs is underway but still far, far from being useful for real-life systems.
- So do we have choice other than to live with bugs?

Back to Testing

- The testing process usually requires two components:
  - Traces/sequences of events/inputs to be simulated
  - A collection of properties which must be satisfied during the running of these traces
- Some properties are easy to specify:
  - "At login, the user must be in the database"
  - "At every point in the lifetime of a financial transaction, the incoming and outgoing funds must balance"
- It is relatively easy to check these

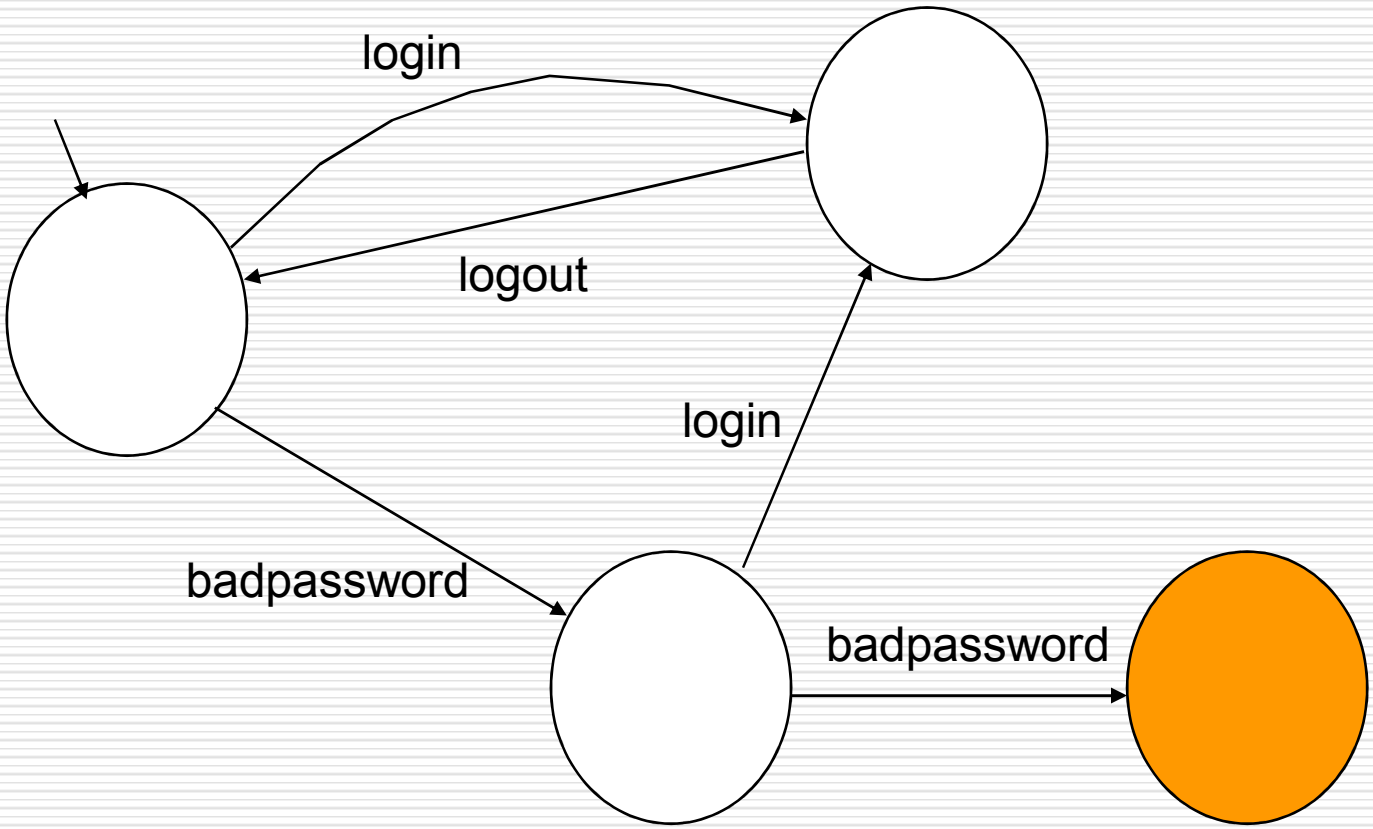
Back to Testing

- Some properties are less easy to specify:
  - "The user has the right to change her user name the first three times she logs in".
  - "If a financial transaction takes longer than 30 days to complete, the operator must be notified".
  - "A login may only occur 5 times in 30 minutes except in the first hour after user creation, during which 10 logins are allowed".
- Assertions for these properties are less easy to write
  - And usually involve extra variables (flags), timers, program logic to implement,
  - Which may in turn contain its own bugs.

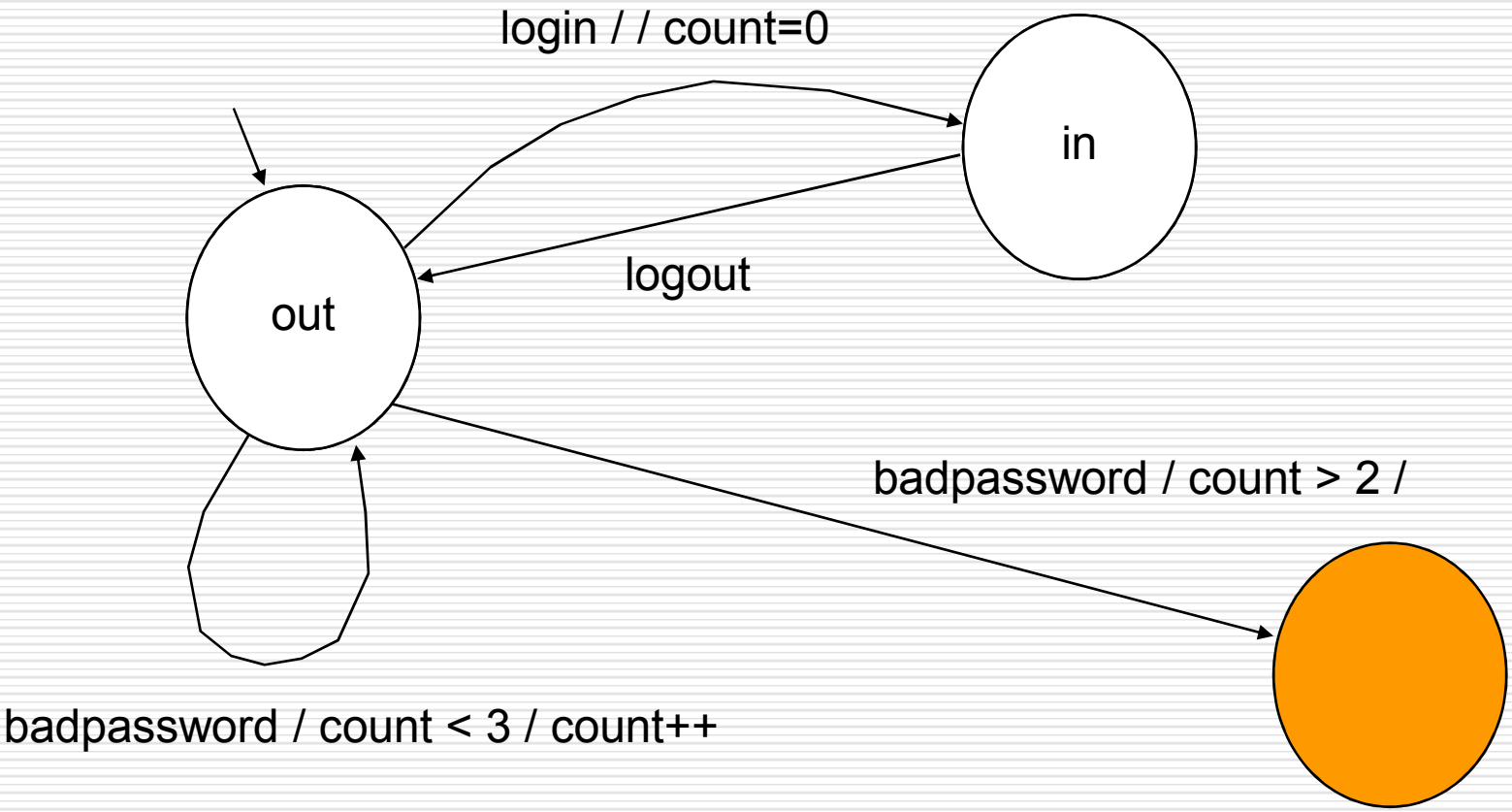
Complex Assertions

- What is needed is:
  - A higher-level way of writing these assertions
  - And an automated way of transforming/compiling these high level assertions into our code
- Various such languages exist.
- Managers, prefer a visual notation.
- Developers are happy with this as long as it can also be written in textual form.

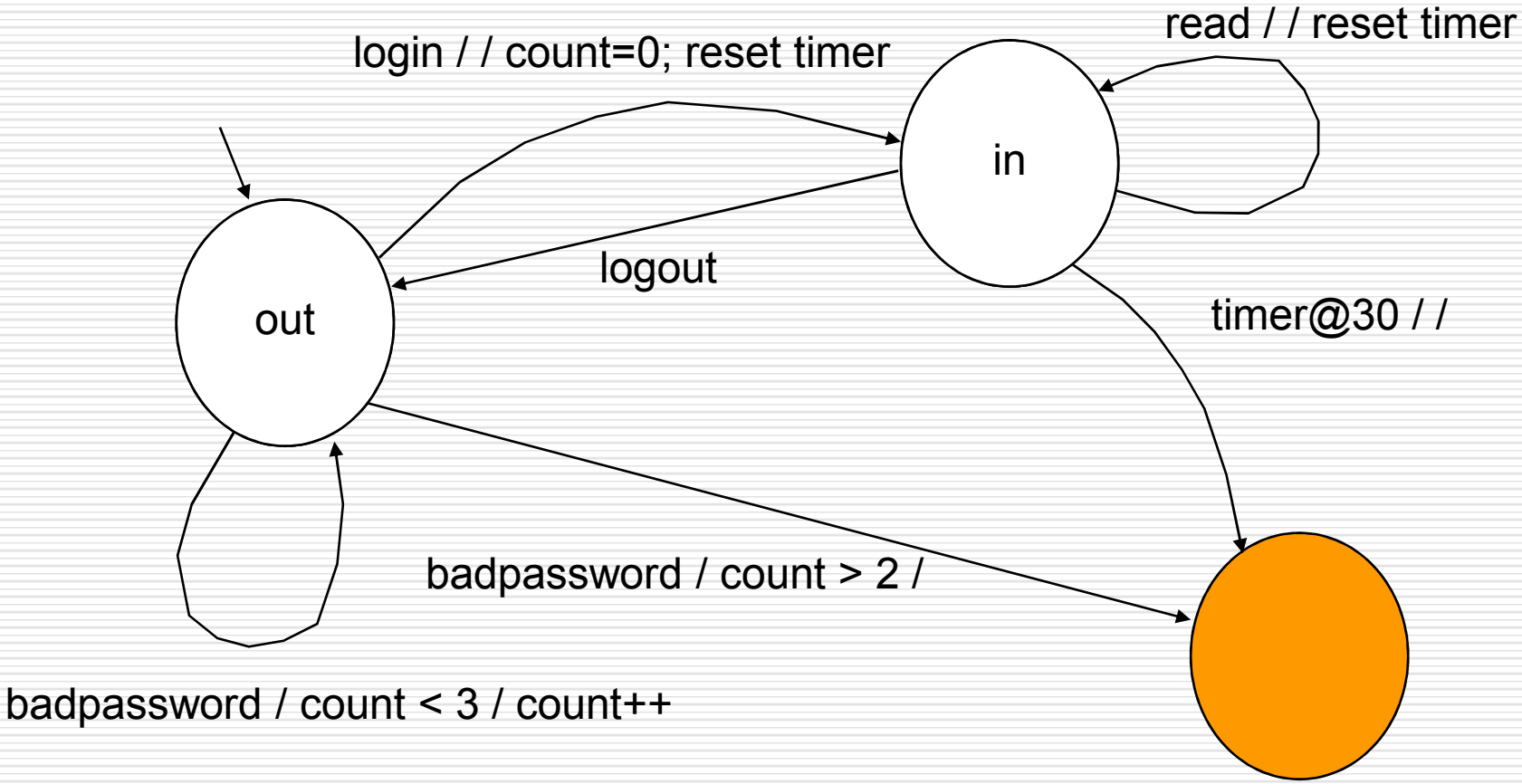
Properties for Testing



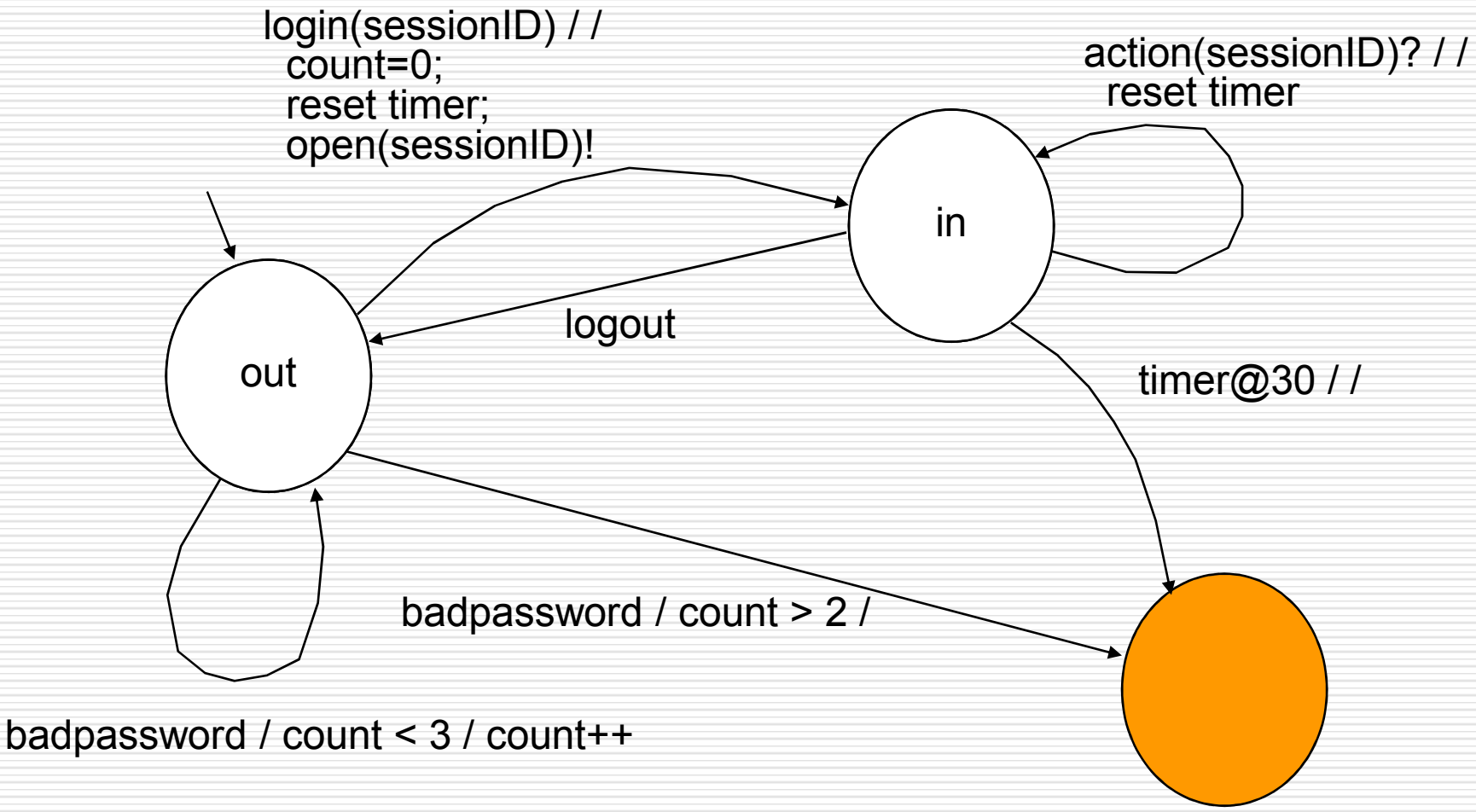
Properties for Testing



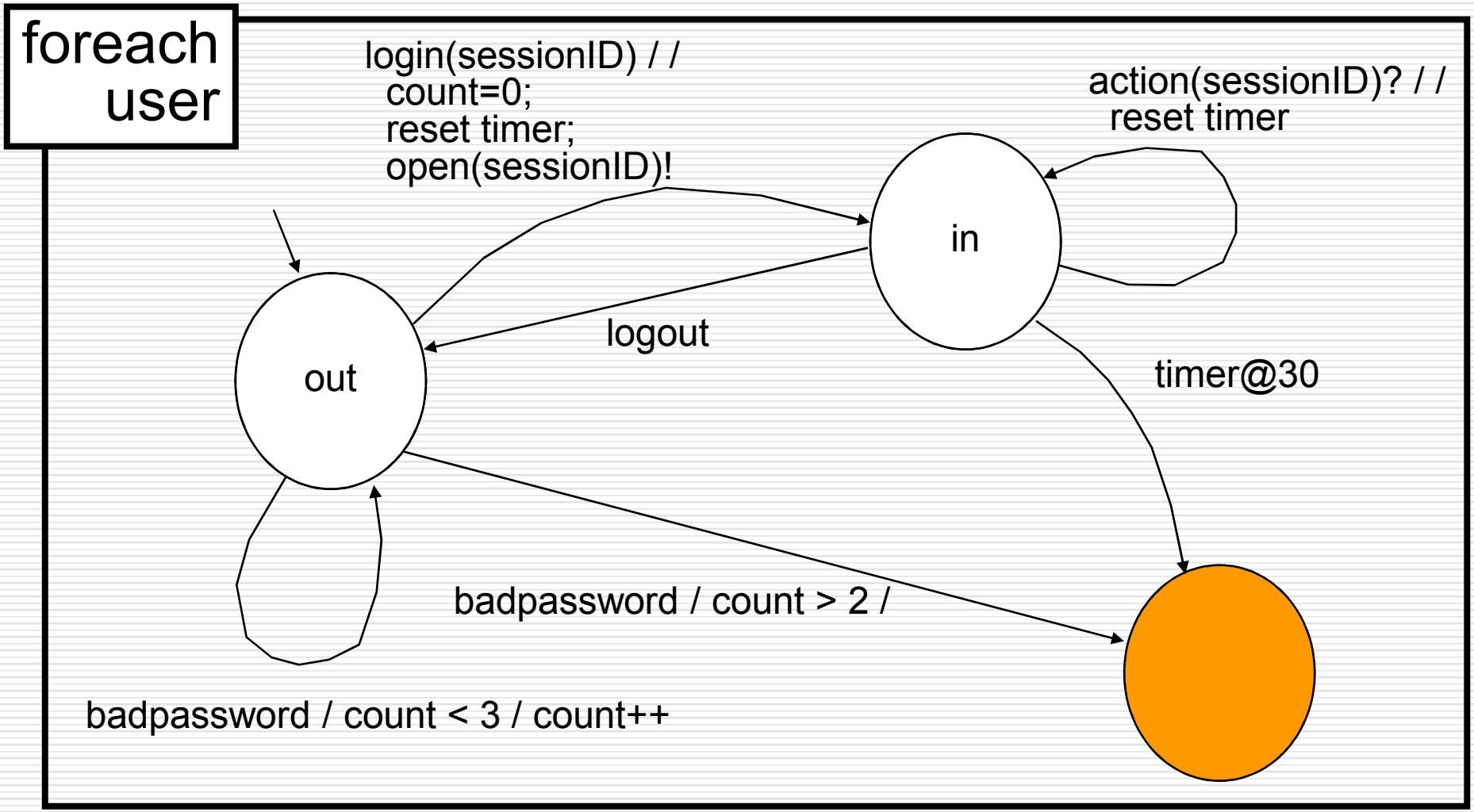
Properties for Testing



Properties for Testing



Properties for Testing



Properties for Testing and Beyond

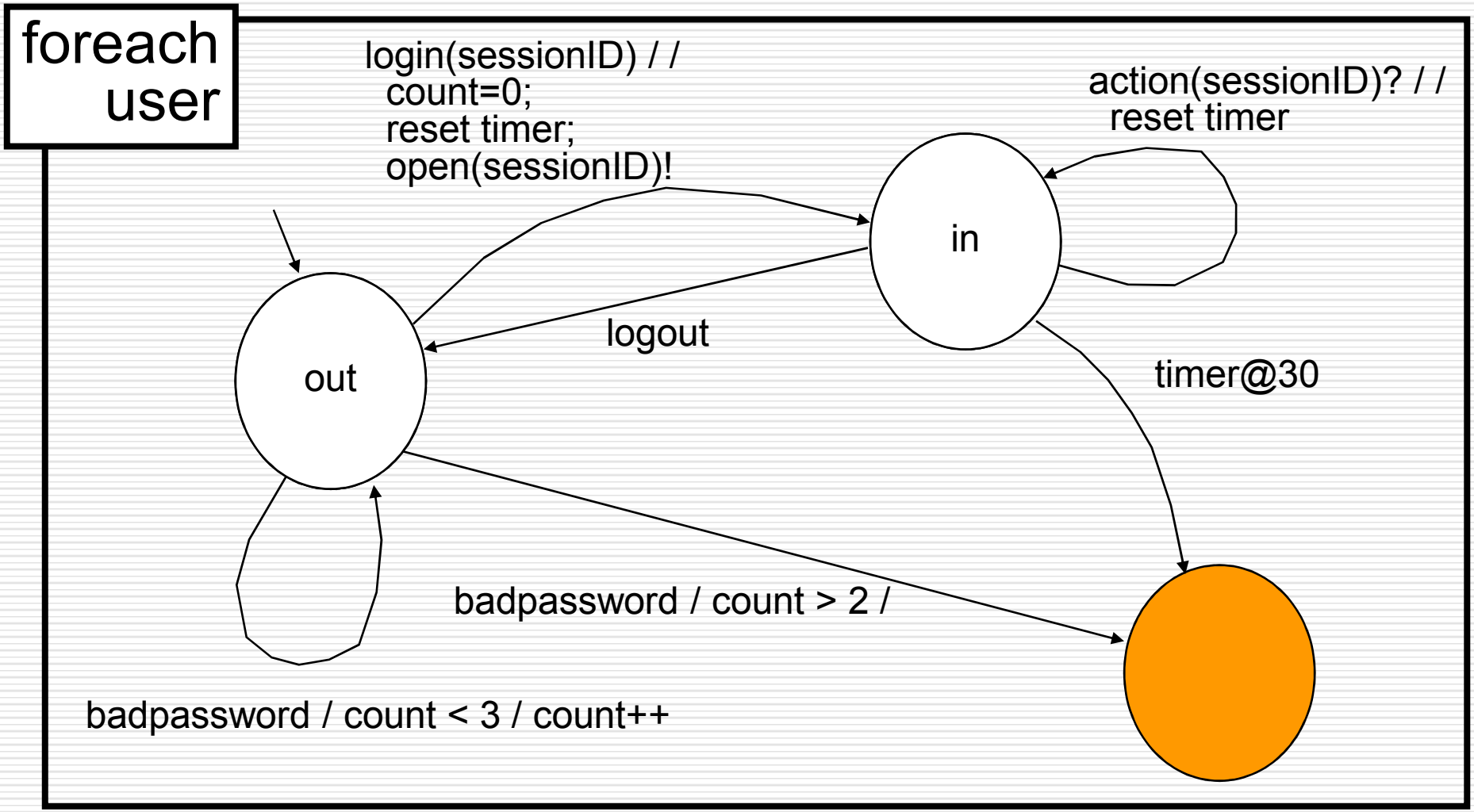
```
Property property = new Property();
while (!bugs.empty()) {
   bugs := program.test(property);
   if (!bugs.empty()) program.debug();
}
finalize(property);
```

- But these properties are of greater value:
  - Reduce, Recycle, Reuse ...

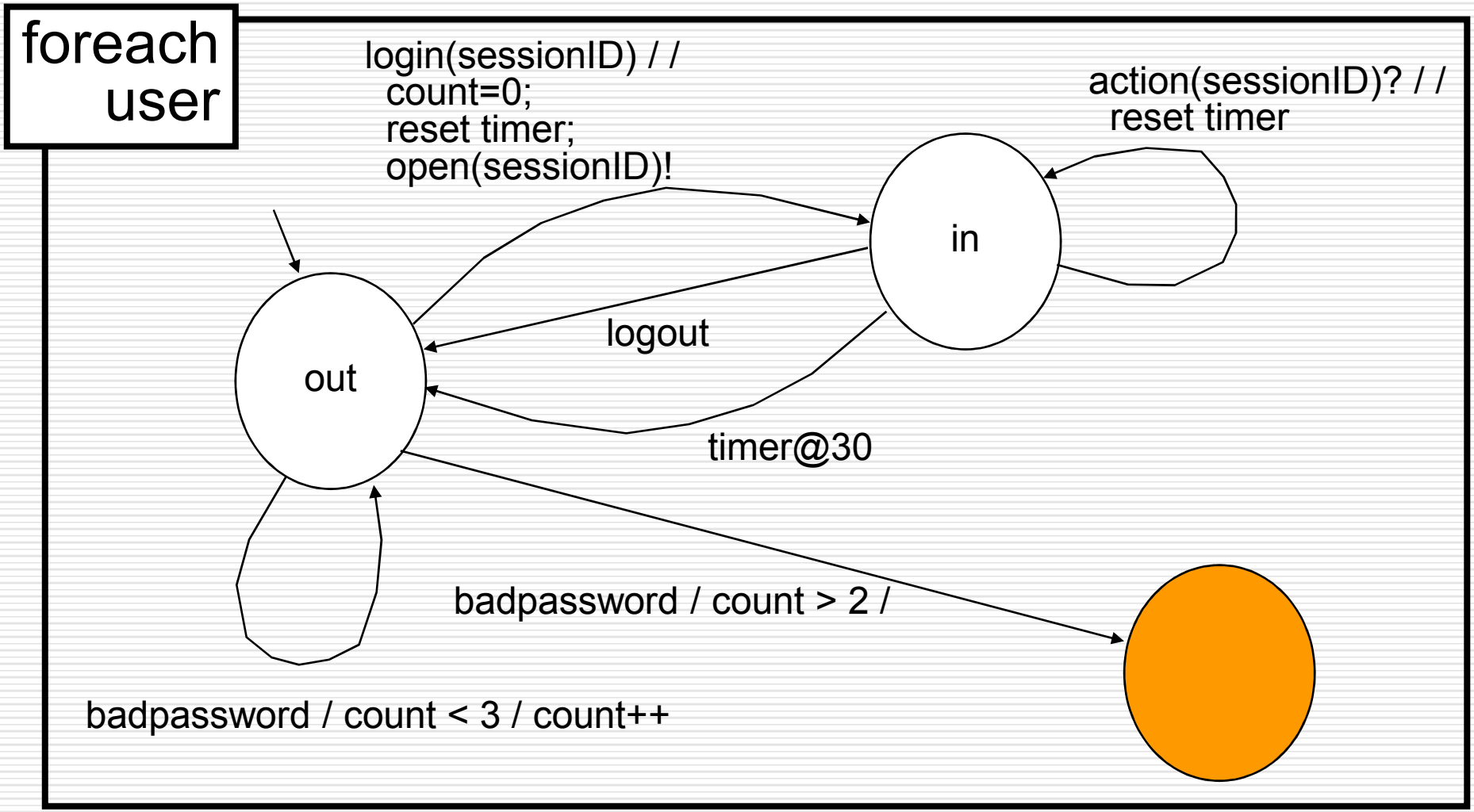
Properties Beyond Testing

- The monitors we inserted for testing can be kept at runtime:
  - At a cost – lower performance
  - But guaranteeing that we know when a property is violated
- But what can be done when a bug is discovered at runtime:
  - Take precautionary measures if potentially serious:
    - Stop the transaction
    - Disable the user
  - Just log the information if not so serious
  - Or even notify a human operator to decide what is to be done

Properties Beyond Testing



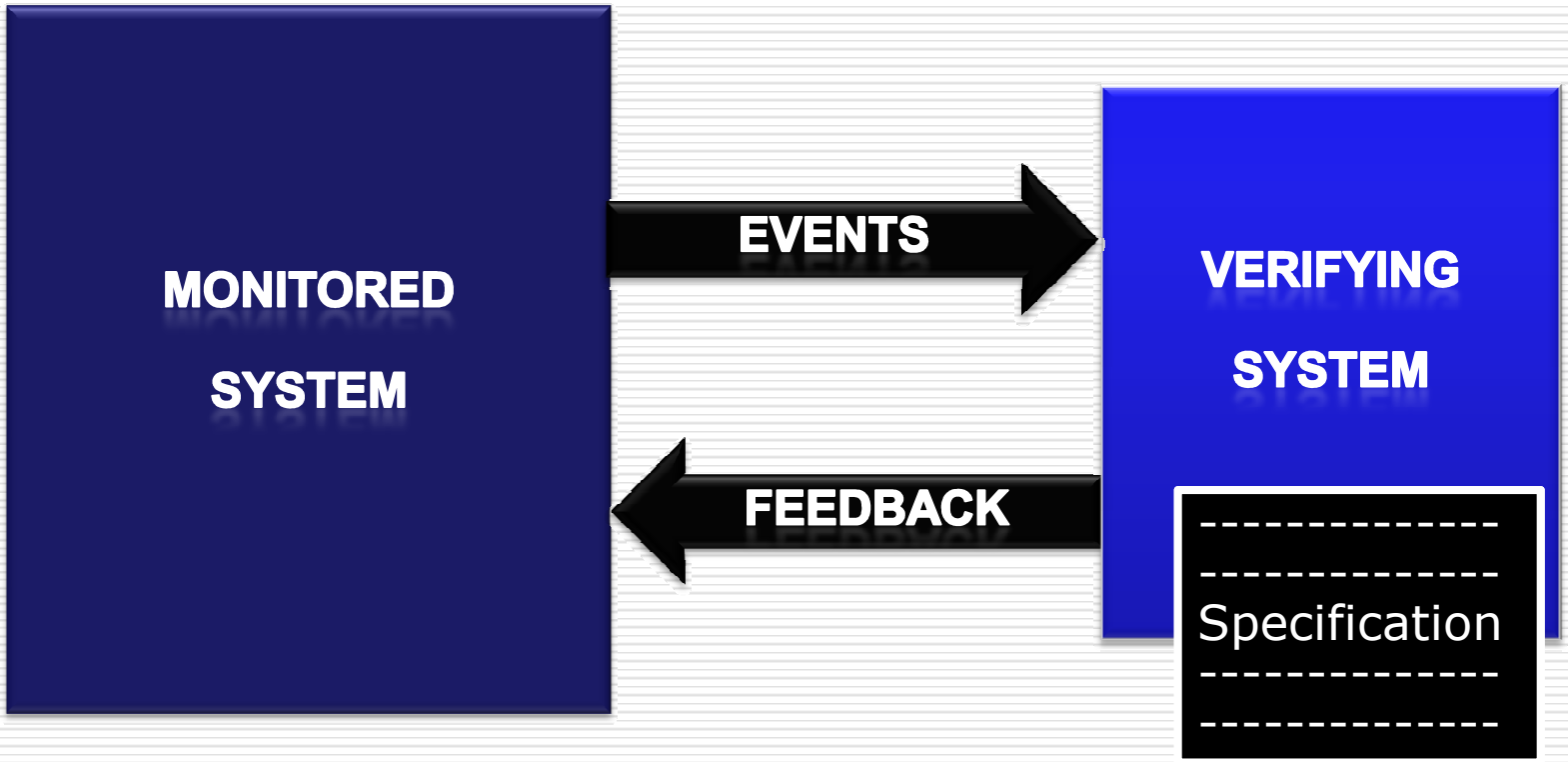
Properties Beyond Testing



General Architecture

General Architecture

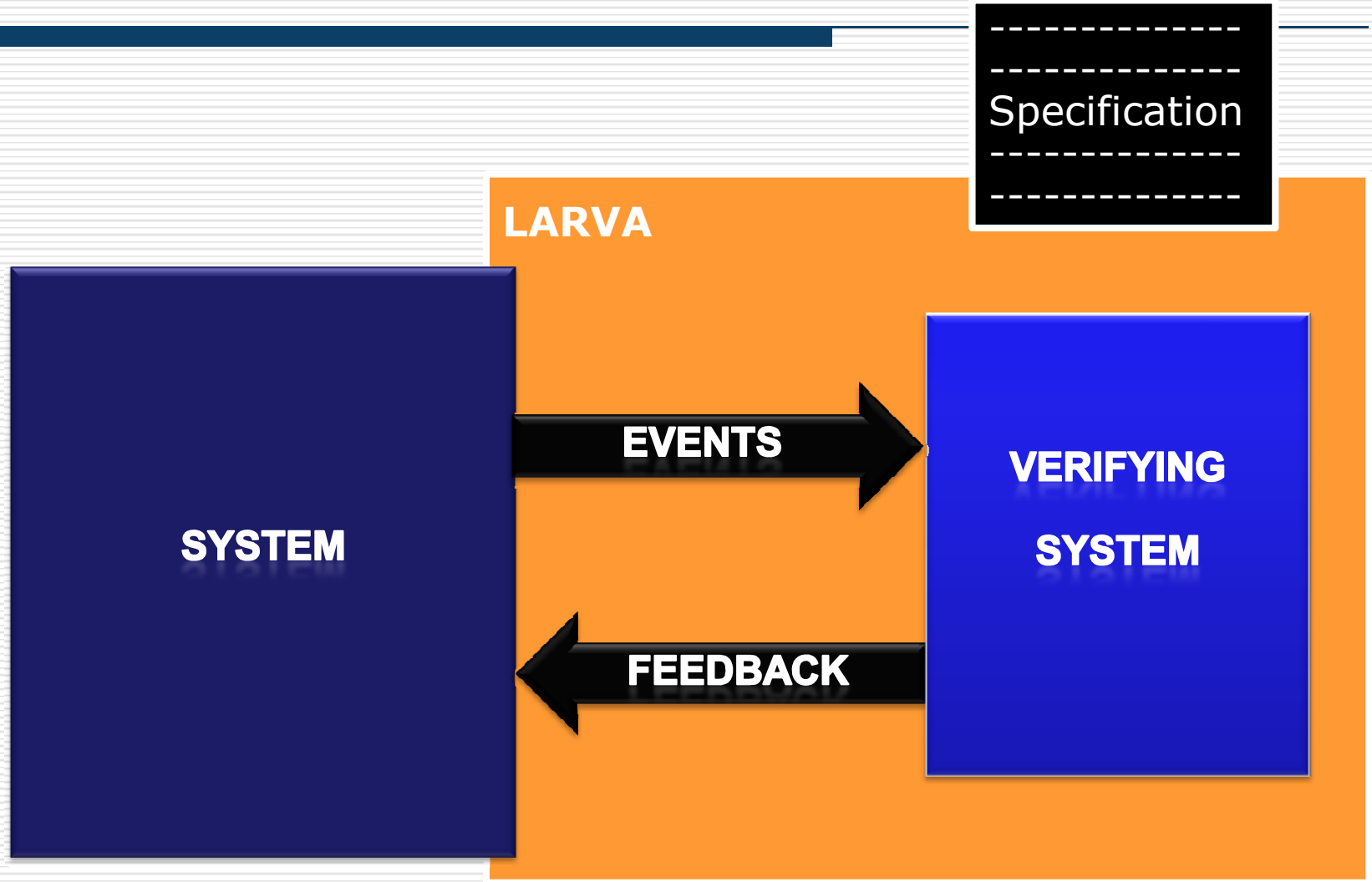
General Architecture



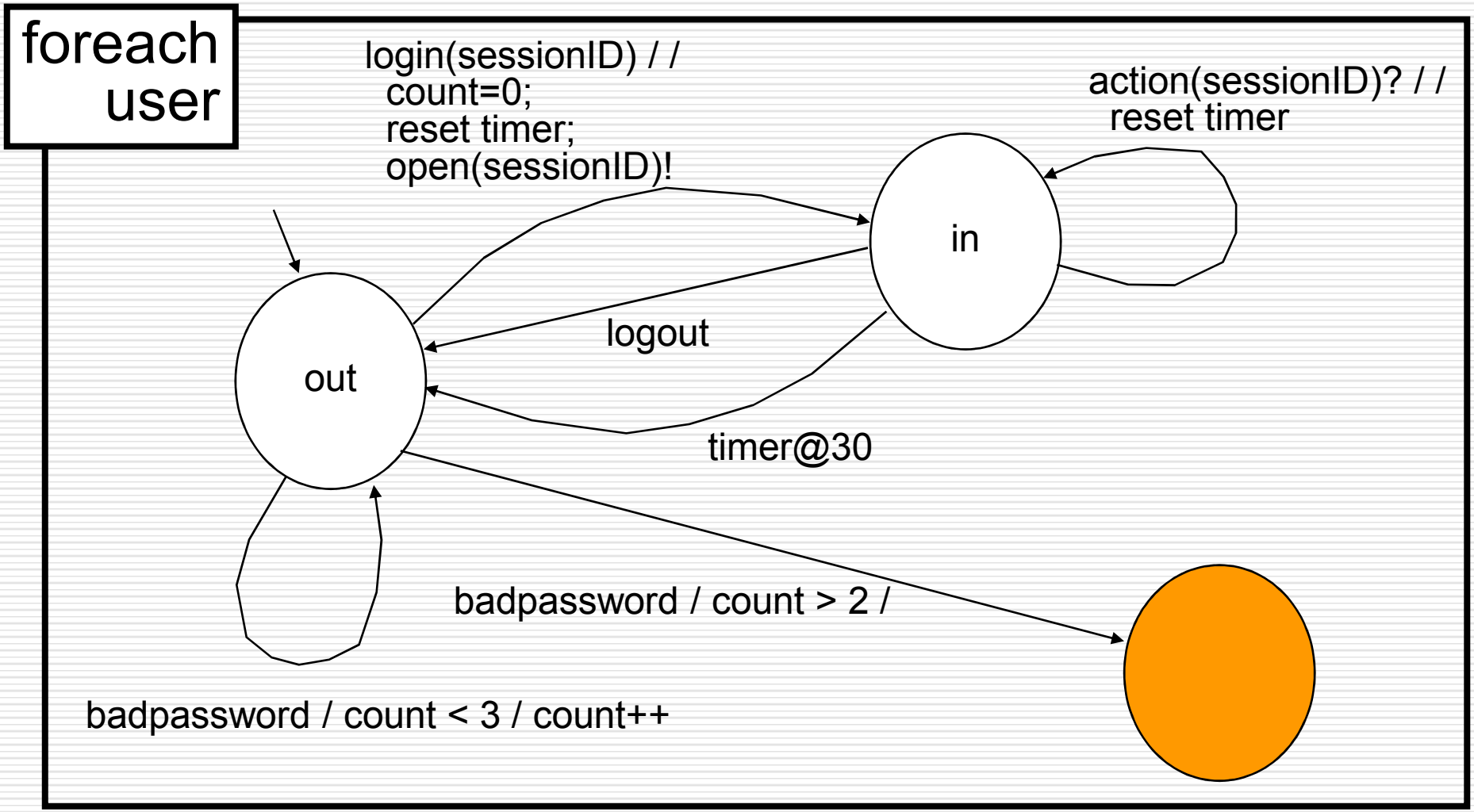
LARVA

- A framework for the specification of real-time properties using a dedicated visual property language.
  - The properties correspond to the diagrams shown earlier.
  - The properties refer to events in the programs via the program method calls and exception handlers.
  - Monitors for the properties are automatically generated and linked to the program to run together.
- Developed to work with Java programs, although linking with the byte-code, not the source code.
- Available for use upon request.

General Architecture



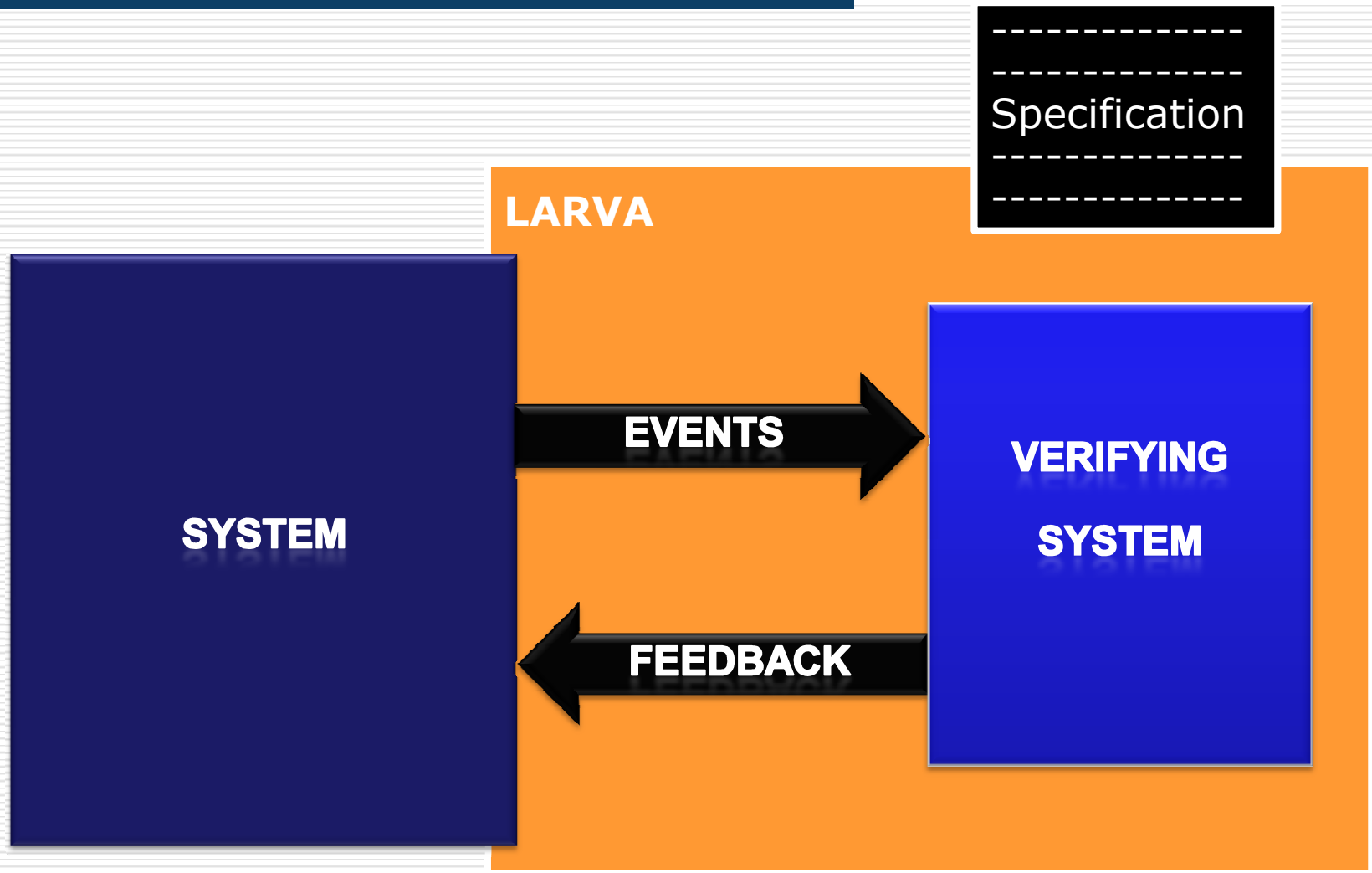
A Property in LARVA



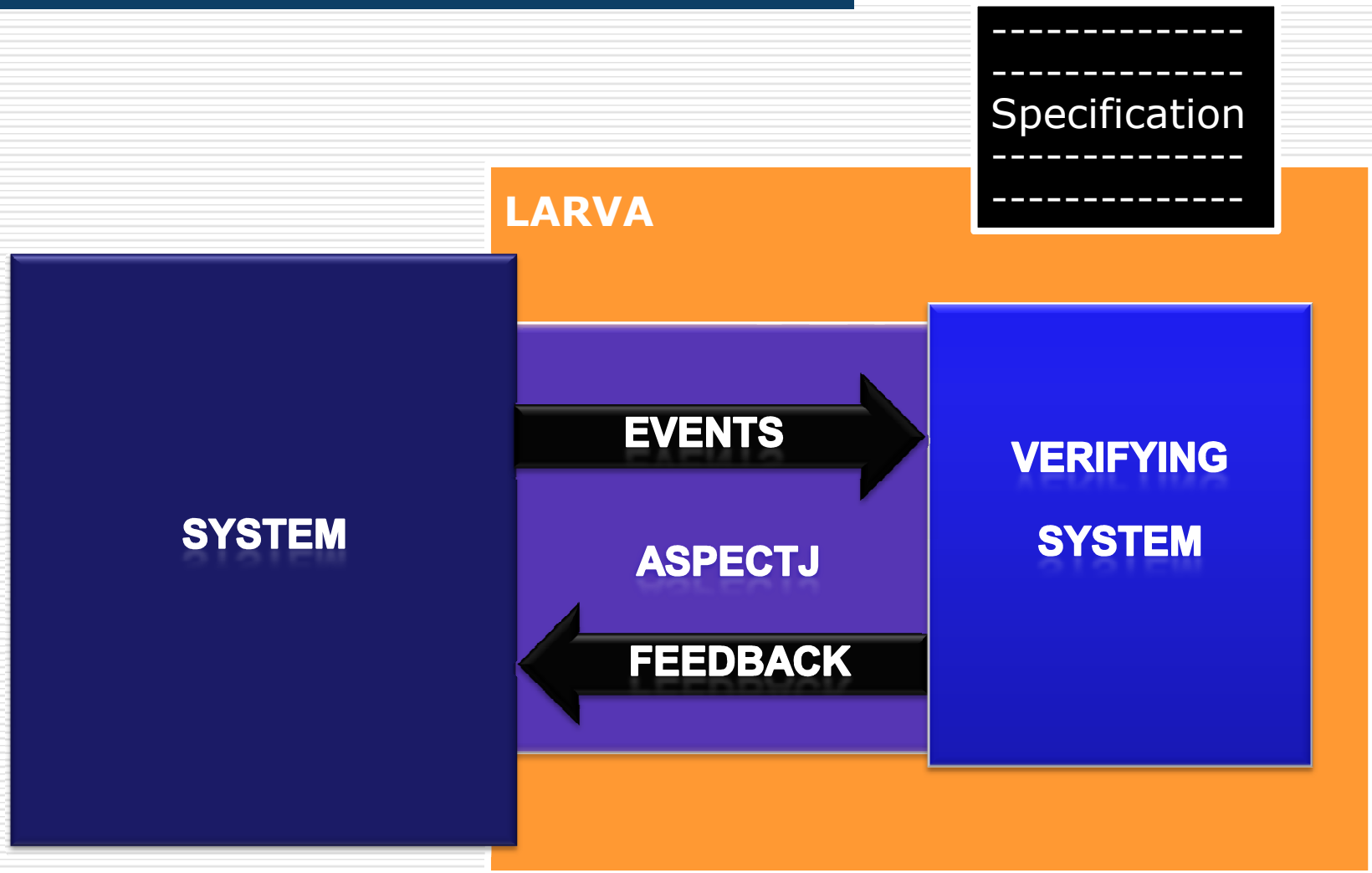
A Property in LARVA

```
FOREACH (User u) {
  EVENTS {
    ...  }
  PROPERTY badlogins {
    STATES {
      BAD { bad }
      NORMAL { loggedin }
      STARTING { loggedout }
    }
    TRANSITIONS {
      loggedout -> loggedin   [ login(sessionID) // count=0; timer.reset(); open(sessionID)! ]
      loggedin  -> loggedin   [ action(sessionID)? // timer.reset(); ]
      loggedin  -> loggedout  [ logout ]
      loggedout -> loggedout  [ badpassword / count < 3 / count++; ]
      loggedout -> bad        [ badpassword / count > 2 / ]
      loggedin  -> loggedout  [ timer@30 / / ]
}}}
```

General Architecture



General Architecture



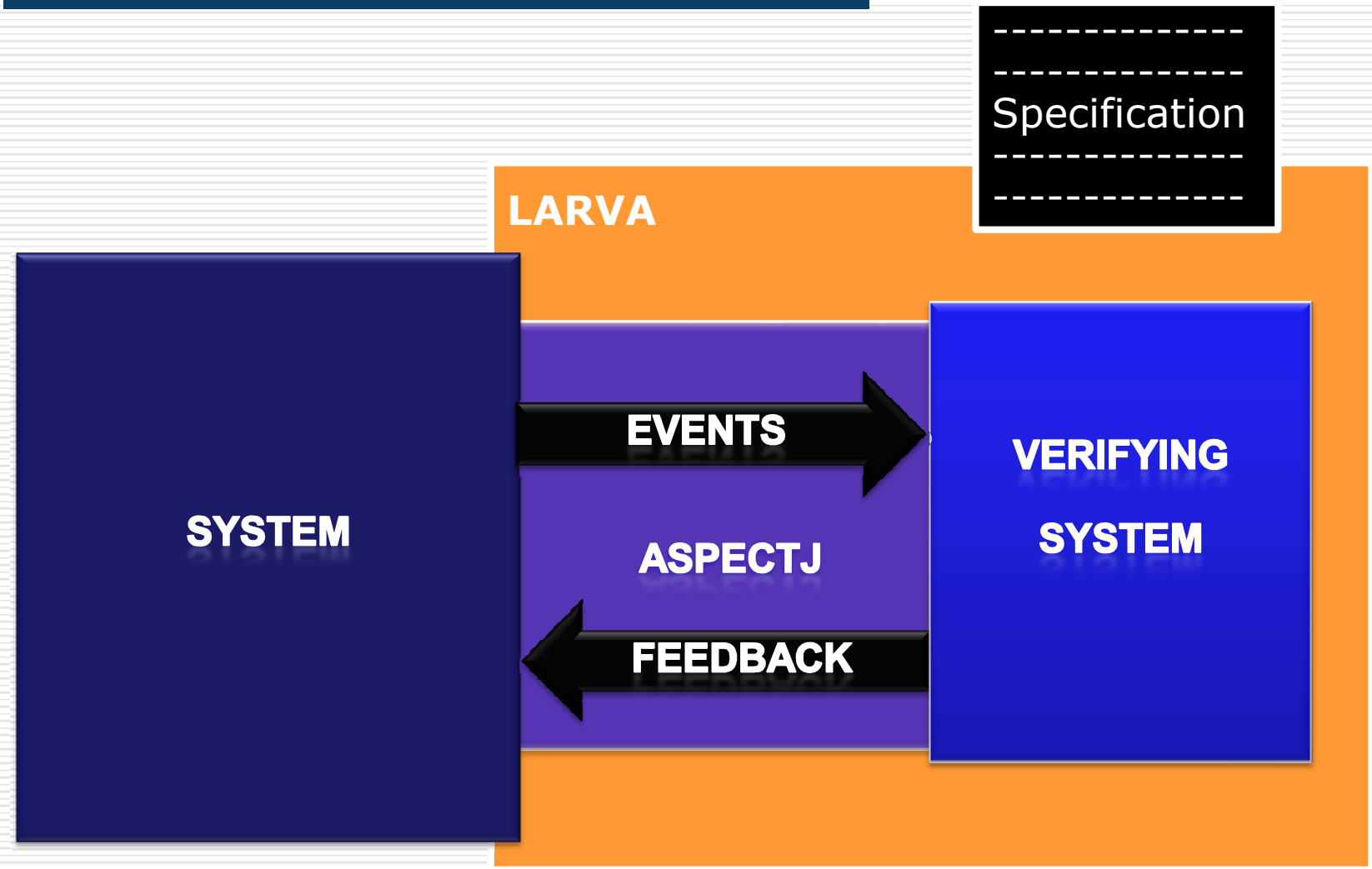
Generating AspectJ

ASPECTJ

```
before (User u) : (call(* User.logout(..)) && target(u)
                    && !cflow(adviceexecution()))
 {
   Automaton instance = getInstance(u);
   instance.call(thisJoinPoint.getSignature().toString(), 4/*logout*/);
}

before (int sessionID, User u) : (call(* User.login(..)) && target(u)
                    && args(sessionID) && !cflow(adviceexecution()))
{
   Automaton  instance = getInstance(u);
   instance.call(thisJoinPoint.getSignature().toString(), 0/*login*/);
}
```

General Architecture



Generating the Verifying System

VERIFYING SYSTEM

```
else if (_state_id_badlogins==1) {
if (1==0){}
else if ((_occurredEvent (_event, 2 /*actionevent*/
))) {

  _state_id_badlogins = 1;//moving to state loggedin
  _goto_badlogins(_info);
}

else if ((_occurredEvent (_event, 4 /*logout*/ ))){

_state_id_badlogins = 2;//moving to state loggedout
_goto_badlogins(_info);
}

else if ((_occurredEvent (_event, 8 /*cAT30*/ ))){

_state_id_badlogins = 2;//moving to state loggedout
_goto_badlogins(_info);

} }
```

Runtime Verification in SDLC

- Specifications are written anyway
- Testing is done anyway
- Runtime verification
  - requires precise specifications
  - Assertions are automatically added to your code
  - Checks can be kept during deployment
  - Alerts/actions can be triggered automatically

More about Larva

- Has been used in two industrial case-studies.
- Currently exploring its use with Ixaris Ltd on a financial transaction system.
- Its use is lightweight enough that it is viable even for small case-studies.
- We are looking for more case studies.
- Tool is available upon demand at: http://www.cs.um.edu.mt/svrg/Tools/LARVA/

Current Directions

- Extending the tool as an audit tool – checking properties on past logs.
- Linking the tool with the testing phase of software development.
- Looking into the development of more analyst-friendly property specification.
- Making the tool more and more efficient.

The End