Resource-Bounded Monitoring of Java Programs

Christian Colombo
Gordon J Pace
Gerardo Schneider

SYNCHRON 2009

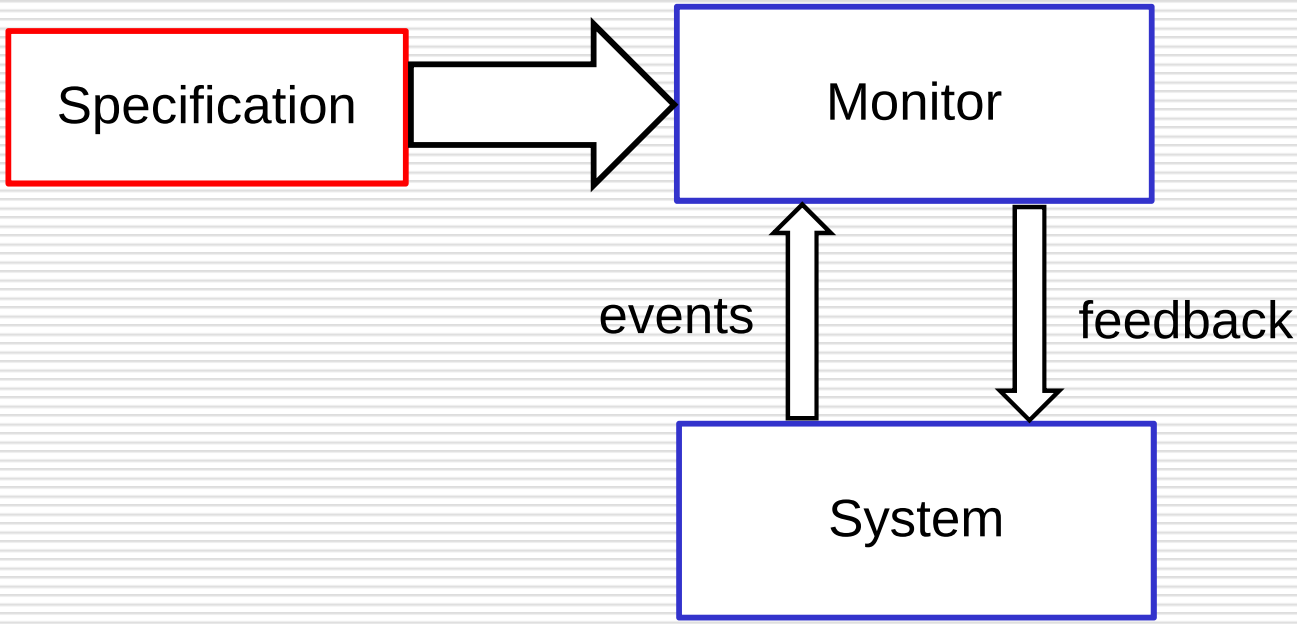
Motivation

- Security critical systems surround us
  - Power plants
  - Train systems
  - Financial transactions
- An error may cost a lot of money or loss of lives
- Testing lacks coverage
- Model checking lacks scalability

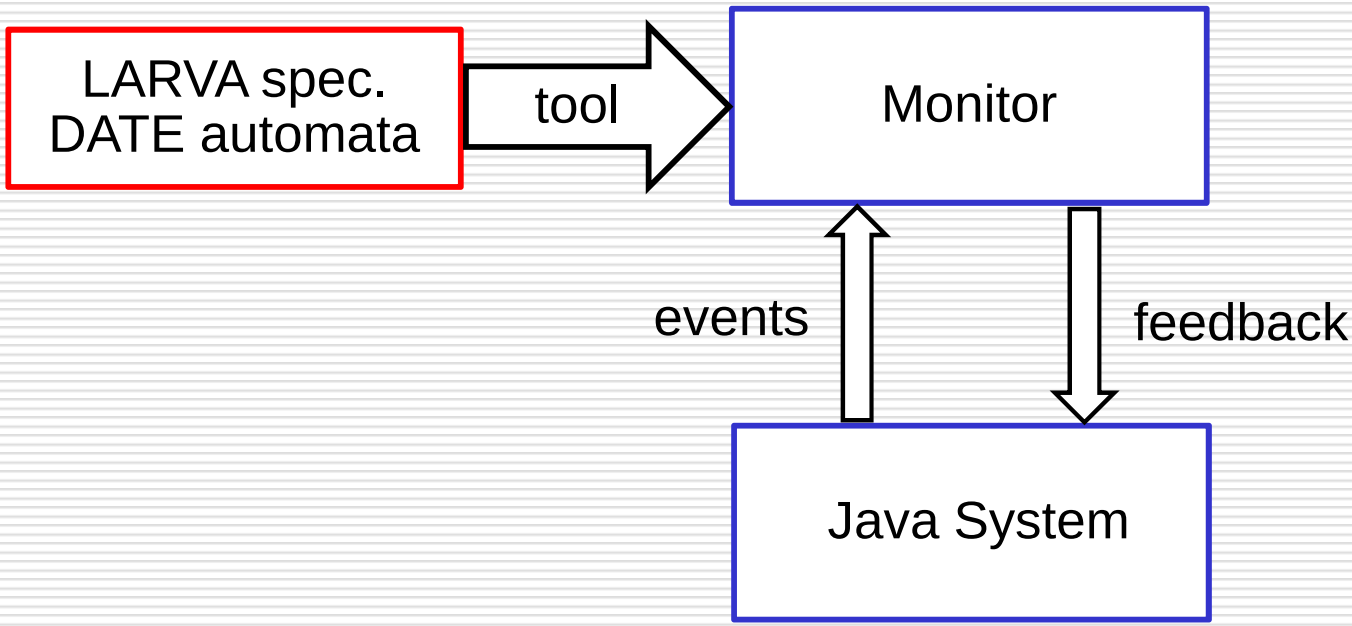
Motivation

- Runtime verification checks current execution path
- Resource overhead induced by monitor
- Guarantee an upperbound of resources per monitored object

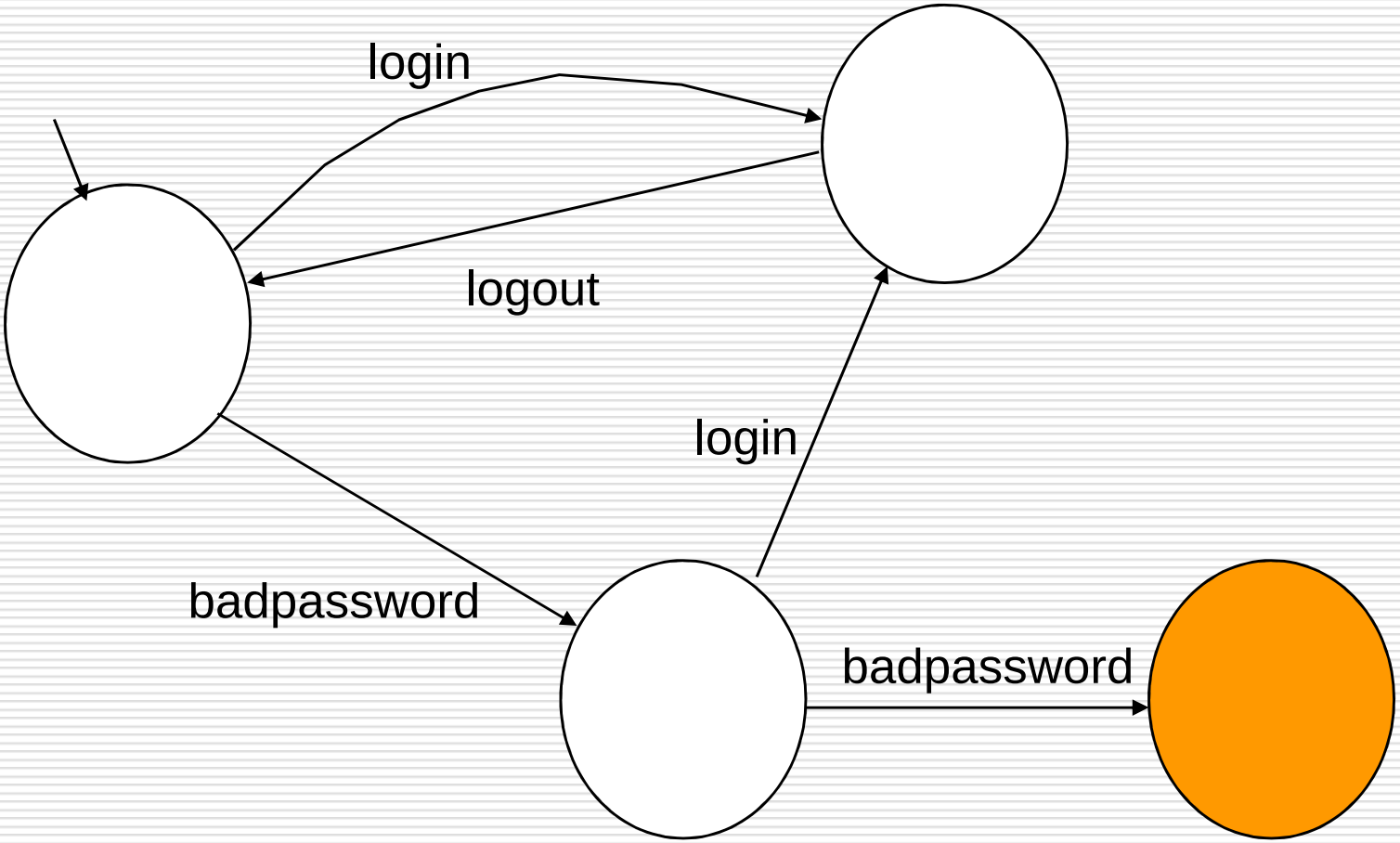
Runtime Verification Architecture



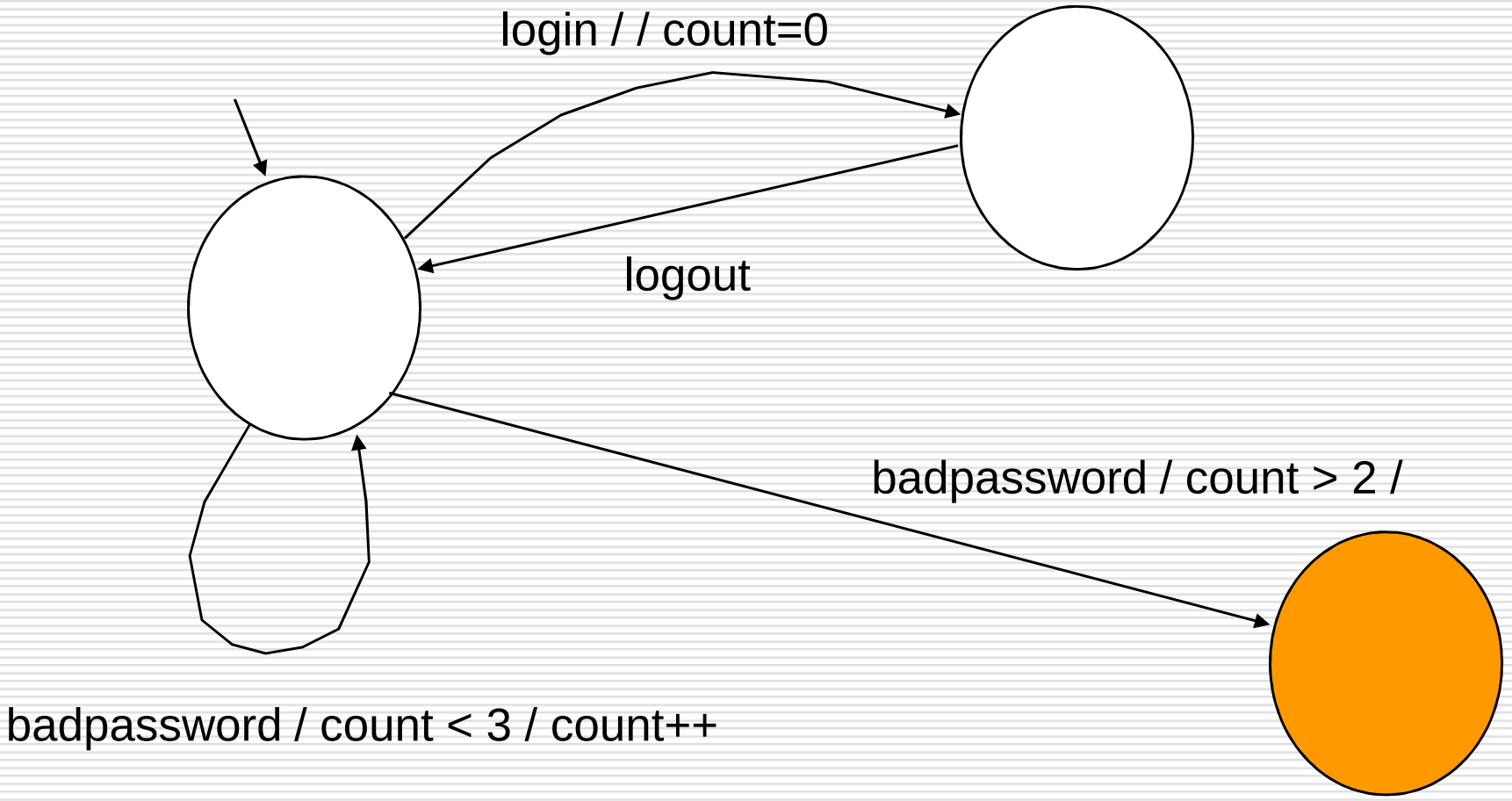
Larva Architecture



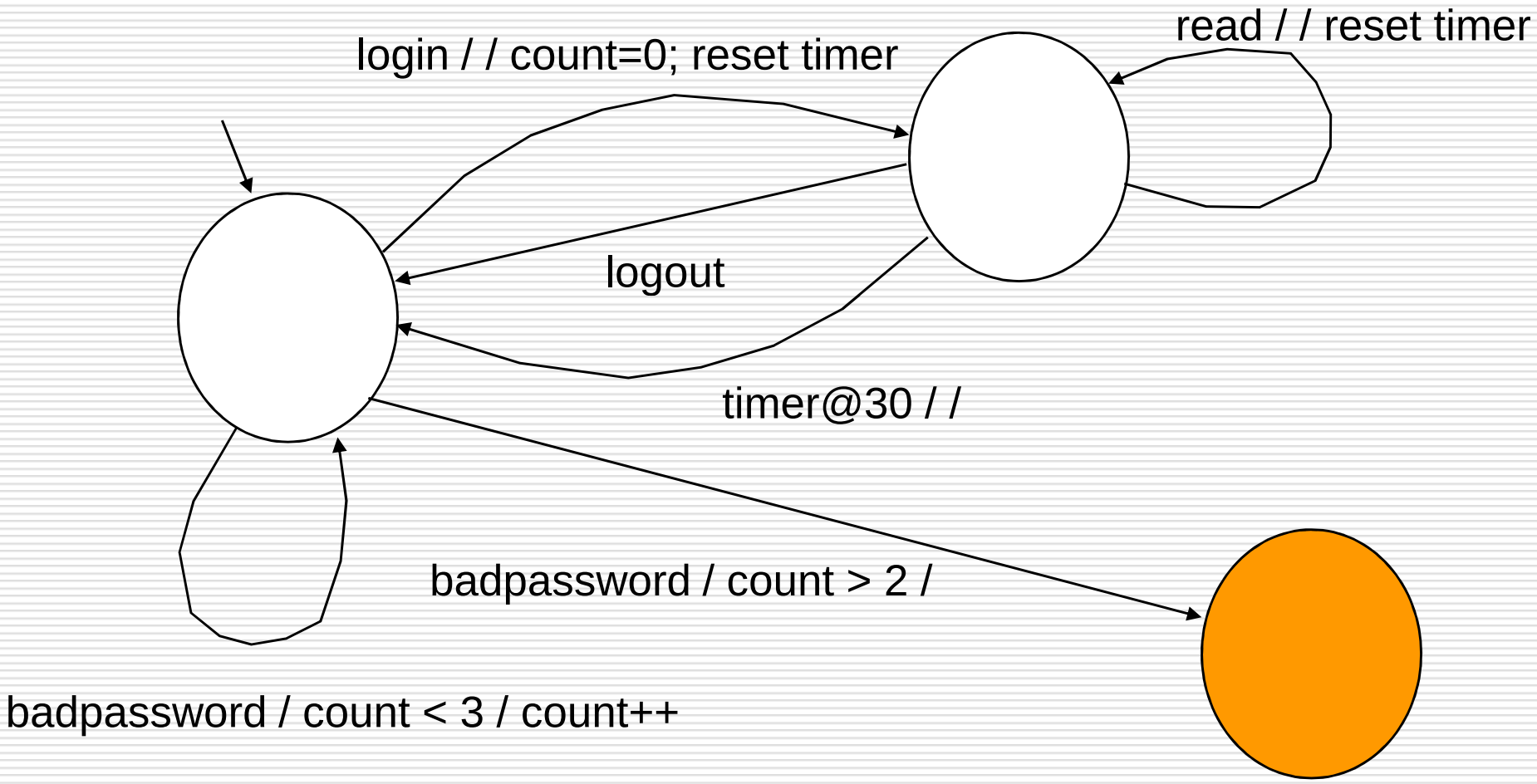
Properties in LARVA



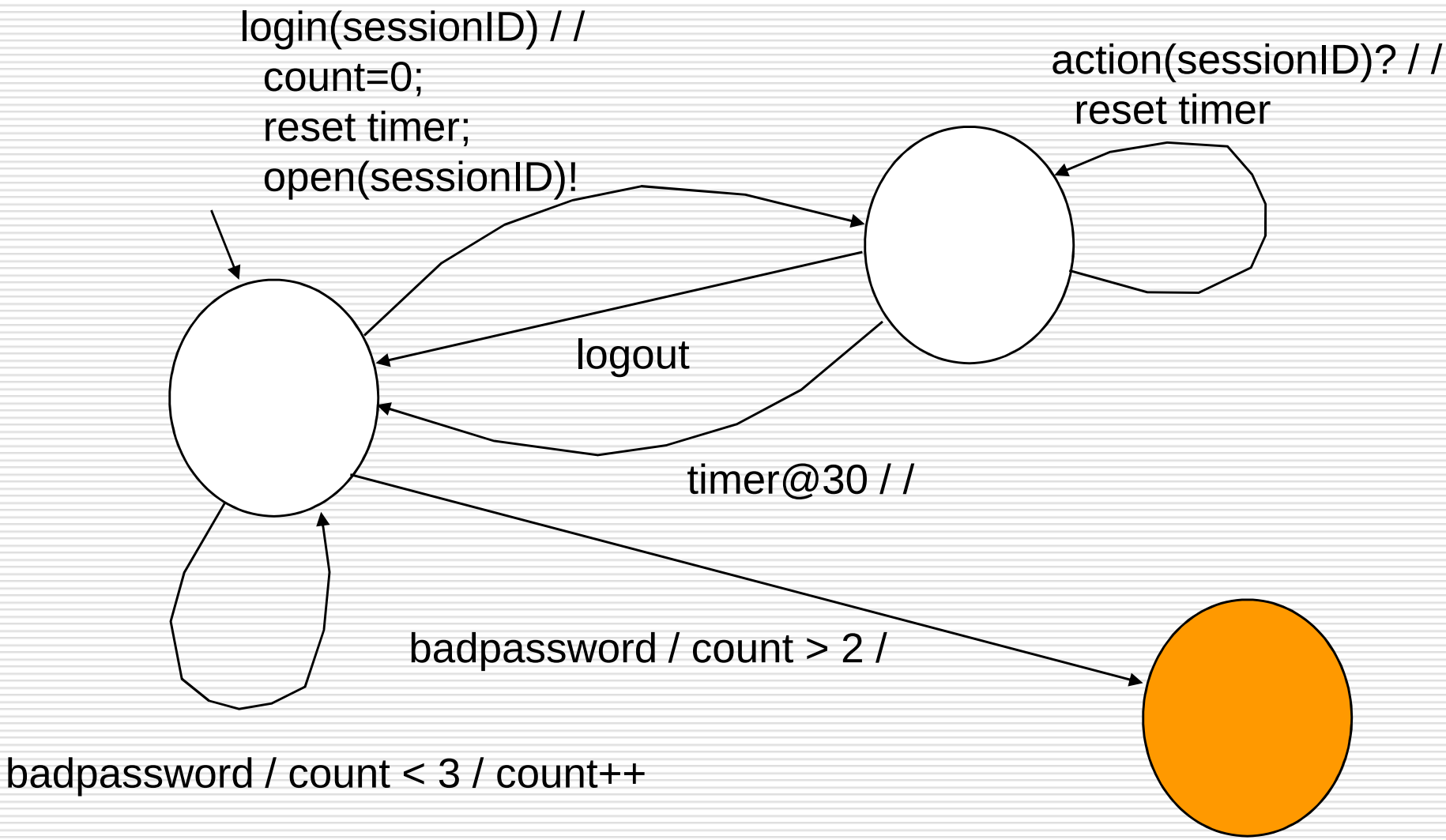
Properties in LARVA



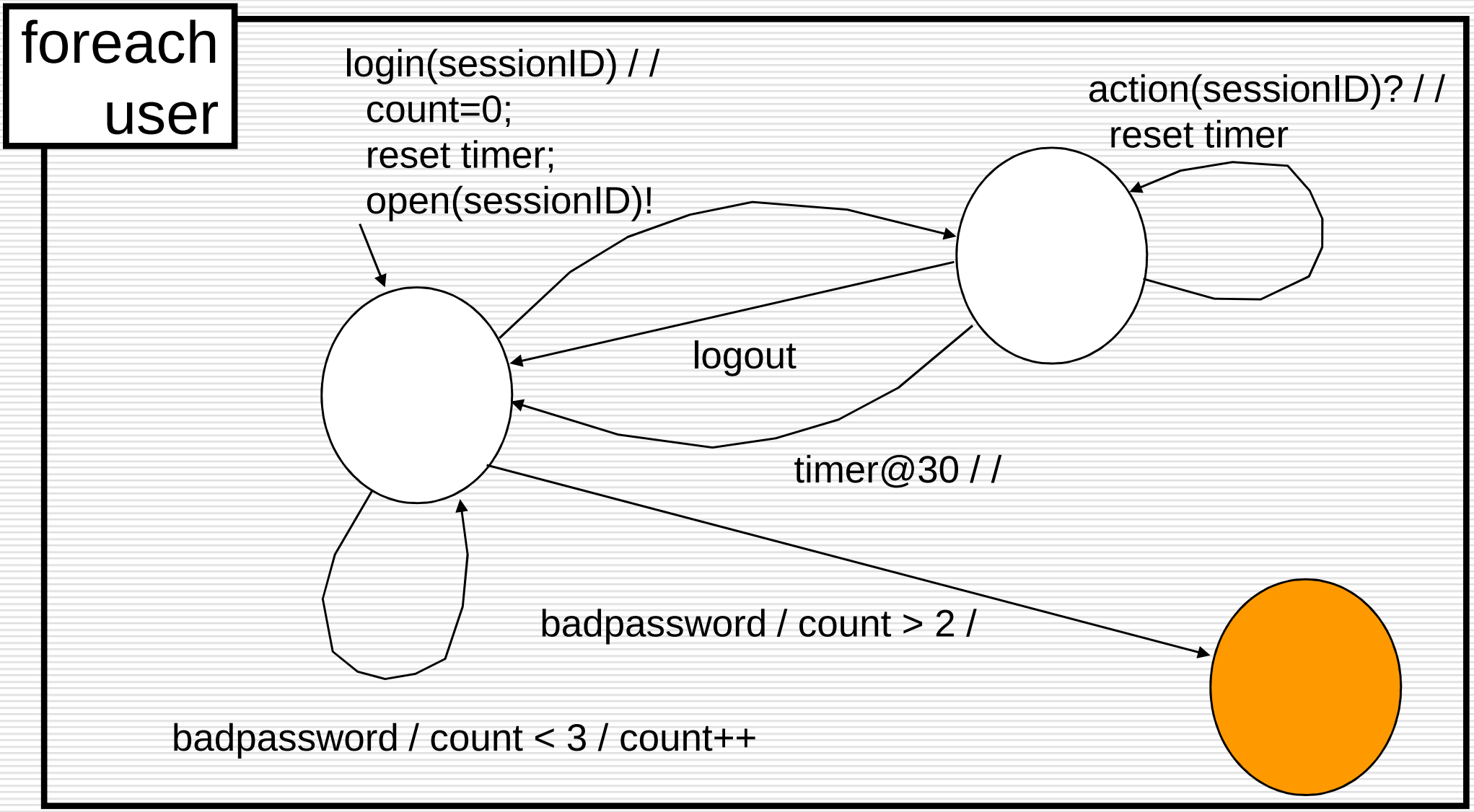
Properties in LARVA



Properties in LARVA



Properties in LARVA



The Problem

- The actions on transitions can include any Java code
- This allows the introduction of arbitrary overheads
- Actions are not only used as recovery but are crucial for property definitions

Other Considerations

- The automaton is created from the start so there is no risk of it growing
- We are working on a dynamic version of Larva by which the automaton is created on-the-fly

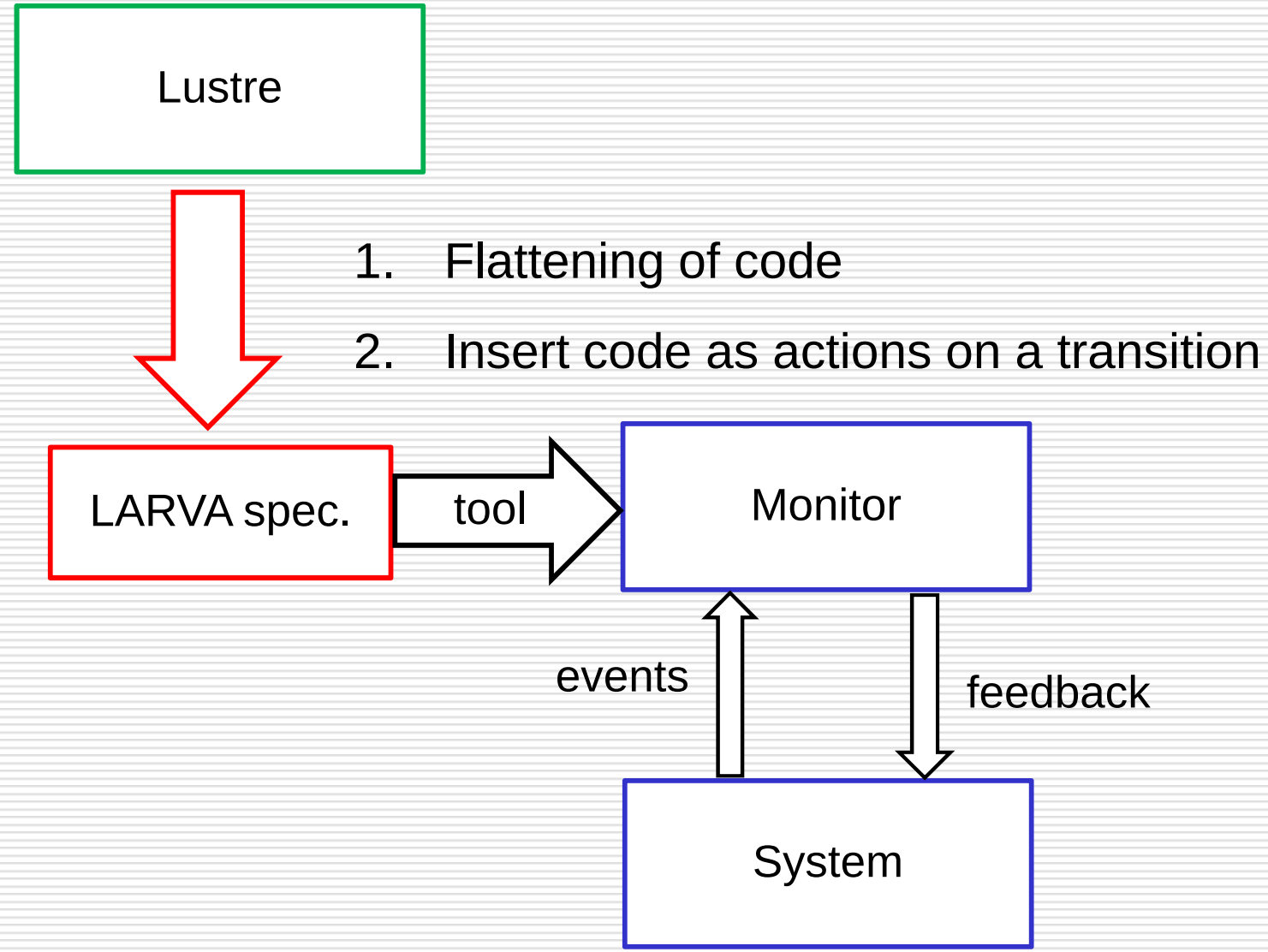
Lustre

- Can be used as a specification language
- Resource calculation at compile-time
- Symbolic automata
- Translation simply involves normal compilation process

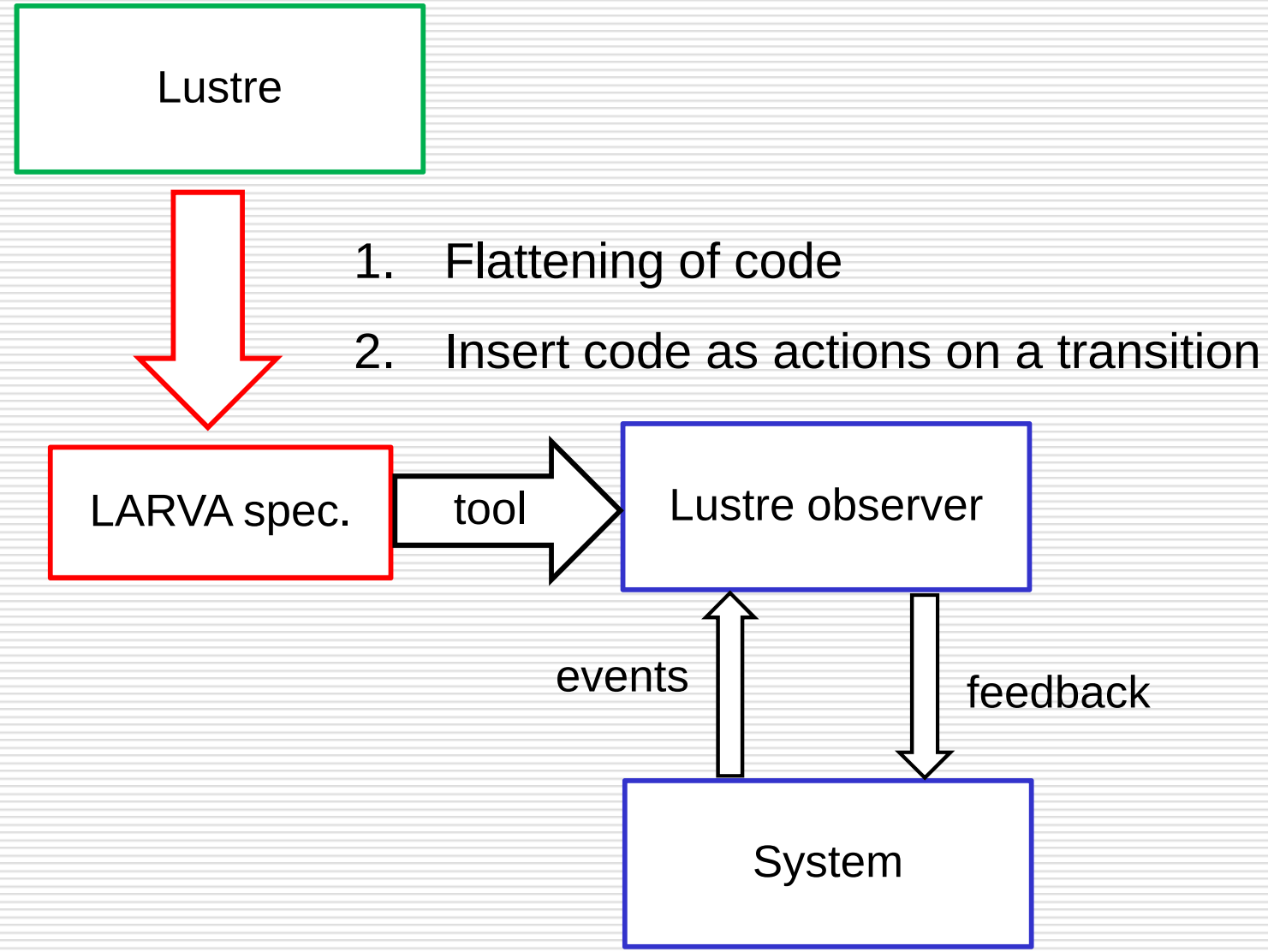
Lustre Example

```
node BadAccess (w,r,i,o:bool) returns
   (bw,br:bool);
var l:bool;
let
l = not o and (i or (false->pre(l)));
bw = w and not(l);
br = r and not(l);
tel
```

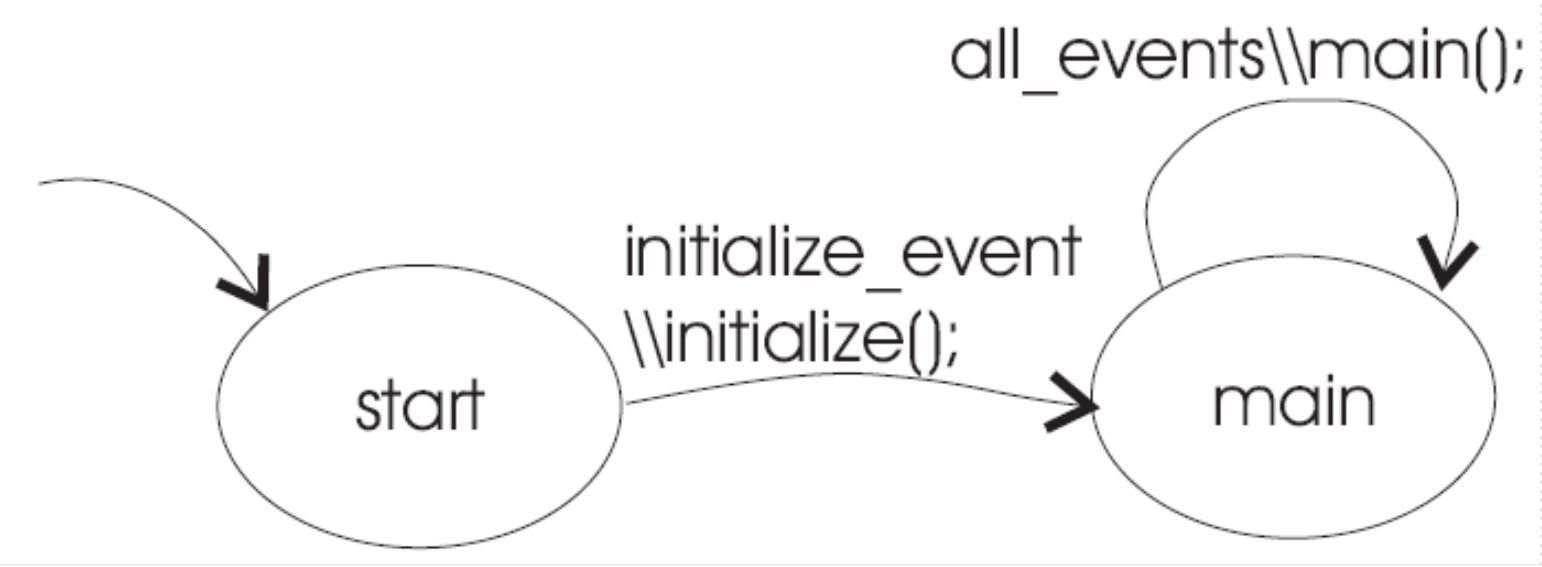
Architecture



Architecture



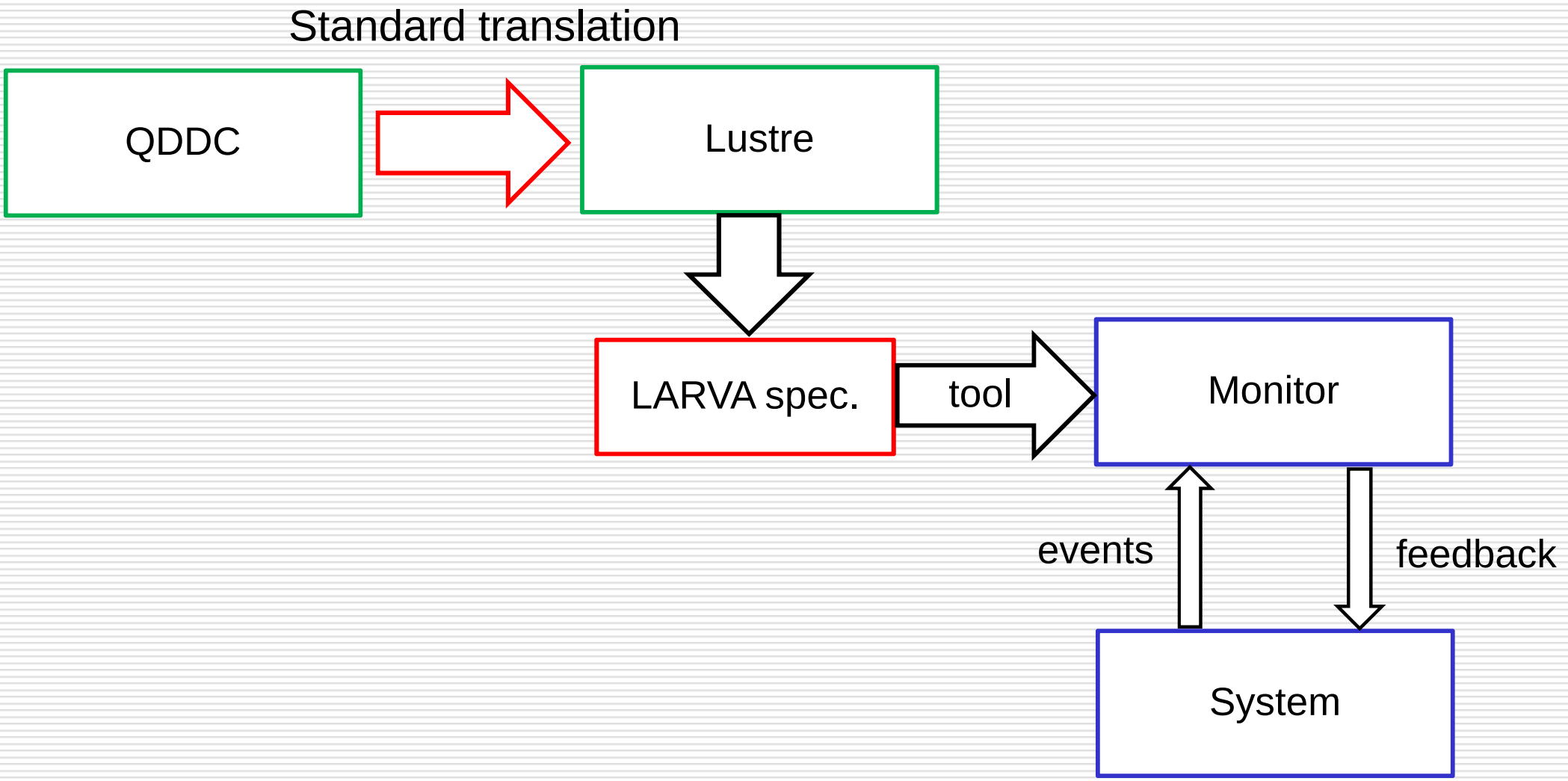
Larva Automaton



QDDC

- A real-time logic
- A fragment of which is translatable into Lustre
- Eg:
  - (age(Danger) < 5) ∨ (age(Alarm) > 0)

Architecture



The Complete Package

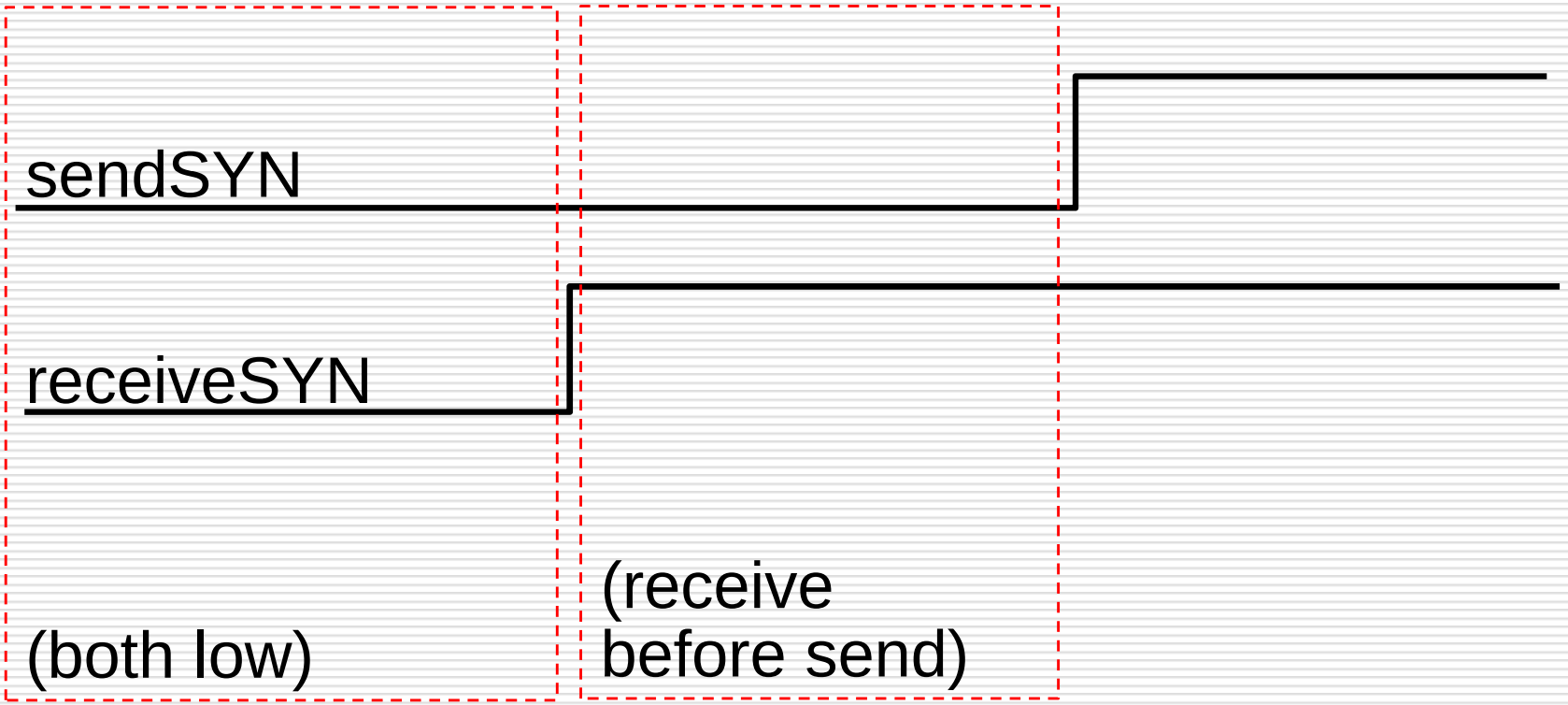
- Larva offers the possibility of monitoring Java programs
- Can monitor properties for each object
- Resource overhead size guarantee for monitoring each object through Lustre
- Specification of real-time properties through QDDC

Case Study

- An intrusion detection system
  - Refusing incoming connections
  - Denial of service attack
  - Port scan attack

Initiating Connection - QDDC

$$[\![\neg sendSYN \wedge \neg receiveSYN]\!] \ then \ begin(receiveSYN)$$



Denial of Service Attack

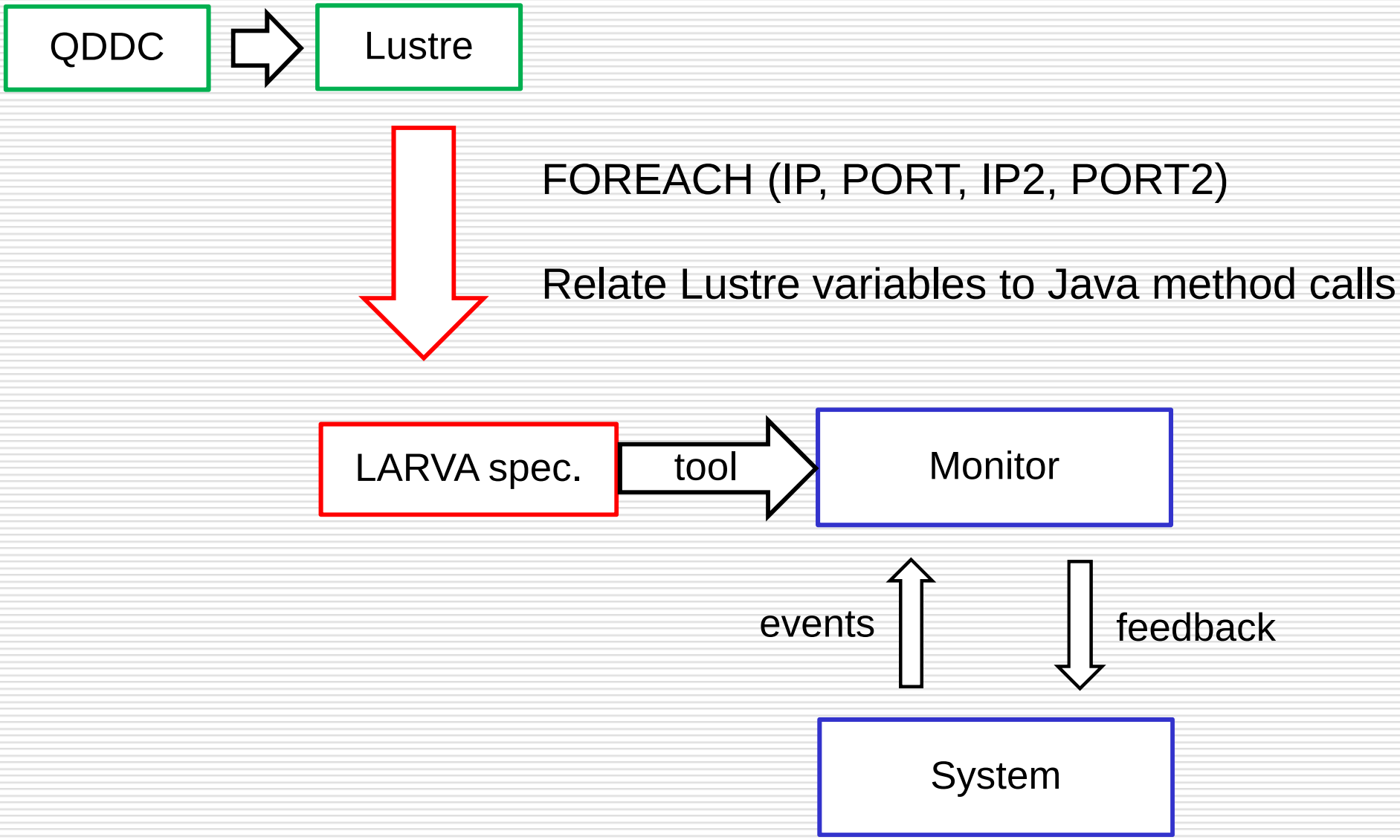
- Bounding the number of events over a period of time

```
node bounded (b:bool; rt_clock:time; redirect:bool; const n:int;
 period:time) returns (p:bool);
var now:bool; count:int; prev_n:time^n;
let
  now = redirect and after(b);
  count = if (now) then (0->pre count)+1 else (0->pre count);
  prev_n = rt_clock | prev_n[0..n-2];
  p = if (count > n and now
      and (rt_clock-(0-> pre(prev_n[n-1])) <= period))
        then false else (true->pre p);
tel
```

Port Scan

- Two Lustre nodes:
  - One to detect varying port numbers
  - Another for bounding the frequency

Using the Architecture



Conclusions

- Gluing together existing theory
- Obtaining more specific specification formalisms
- Per-object resource-bounded runtime monitoring of Java programs