Formal verification of SoC transactional models

Matthieu MOY – Ph.D Student

STMicroelectronics – Verimag

Outline

Outline

- Introduction about Systems on a Chip

Outline

- Introduction about Systems on a Chip
- Verification approach and tool chain

Outline

- Introduction about Systems on a Chip
- Verification approach and tool chain
- Model of SystemC

Outline

- Introduction about Systems on a Chip
- Verification approach and tool chain
- Model of SystemC
- An example

Building complex systems

- Moore's Law and the Design Gap
  - Physical capacity grows faster than design capability
- Hardware / Software
- ASIC / General purpose processors / SoC
- RTL / Transaction Level

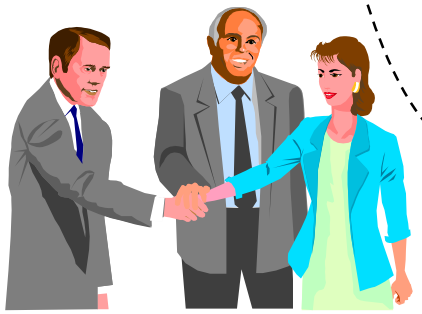
Concurrent Hardware/Software Design

Standard Flow

Concurrent Hardware/Software Design

Standard Flow

SysAr Extension



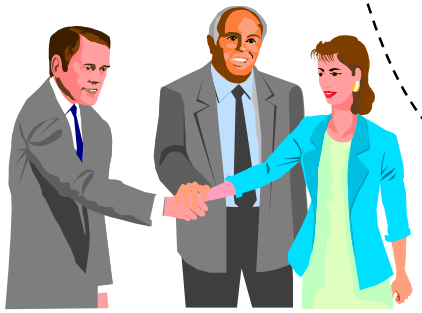
SoC TLM

Time

Concurrent Hardware/Software Design

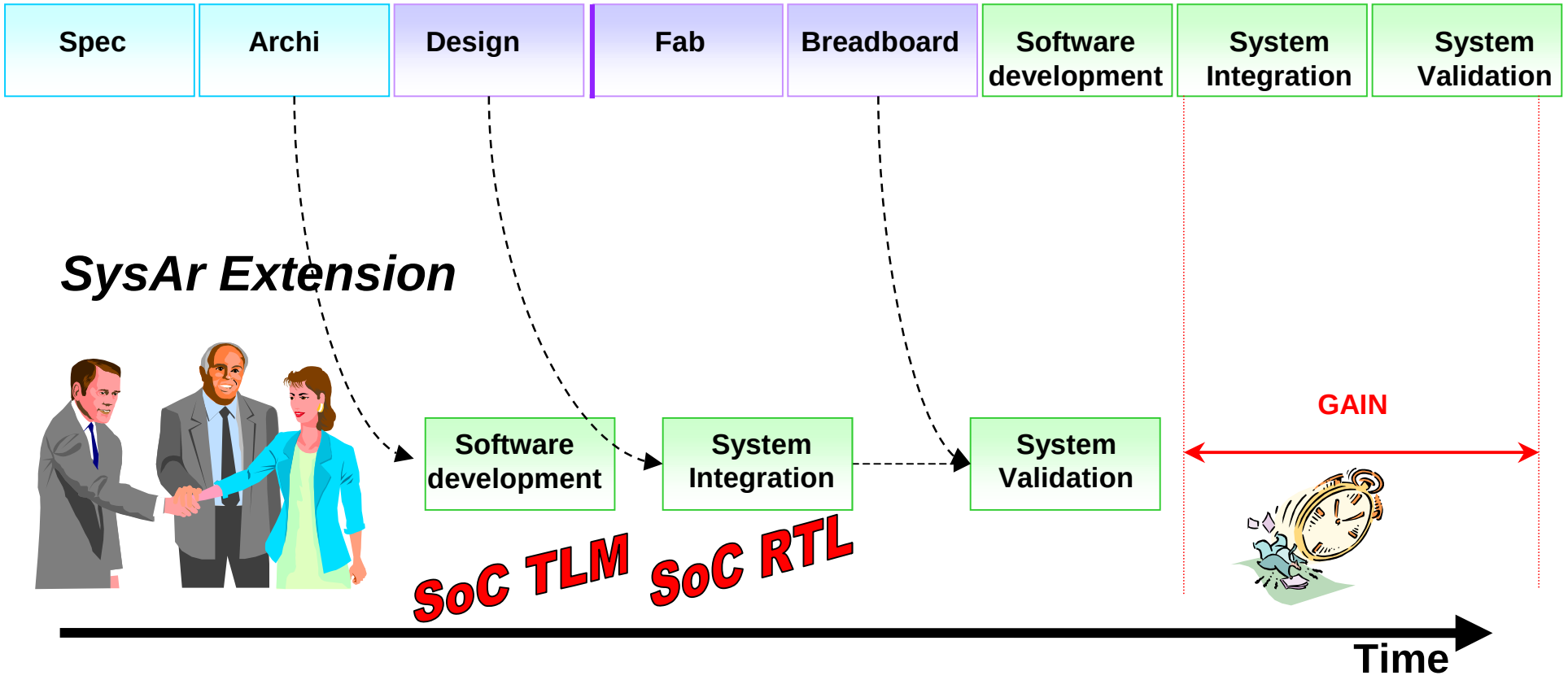
Standard Flow

SysAr Extension



Concurrent Hardware/Software Design

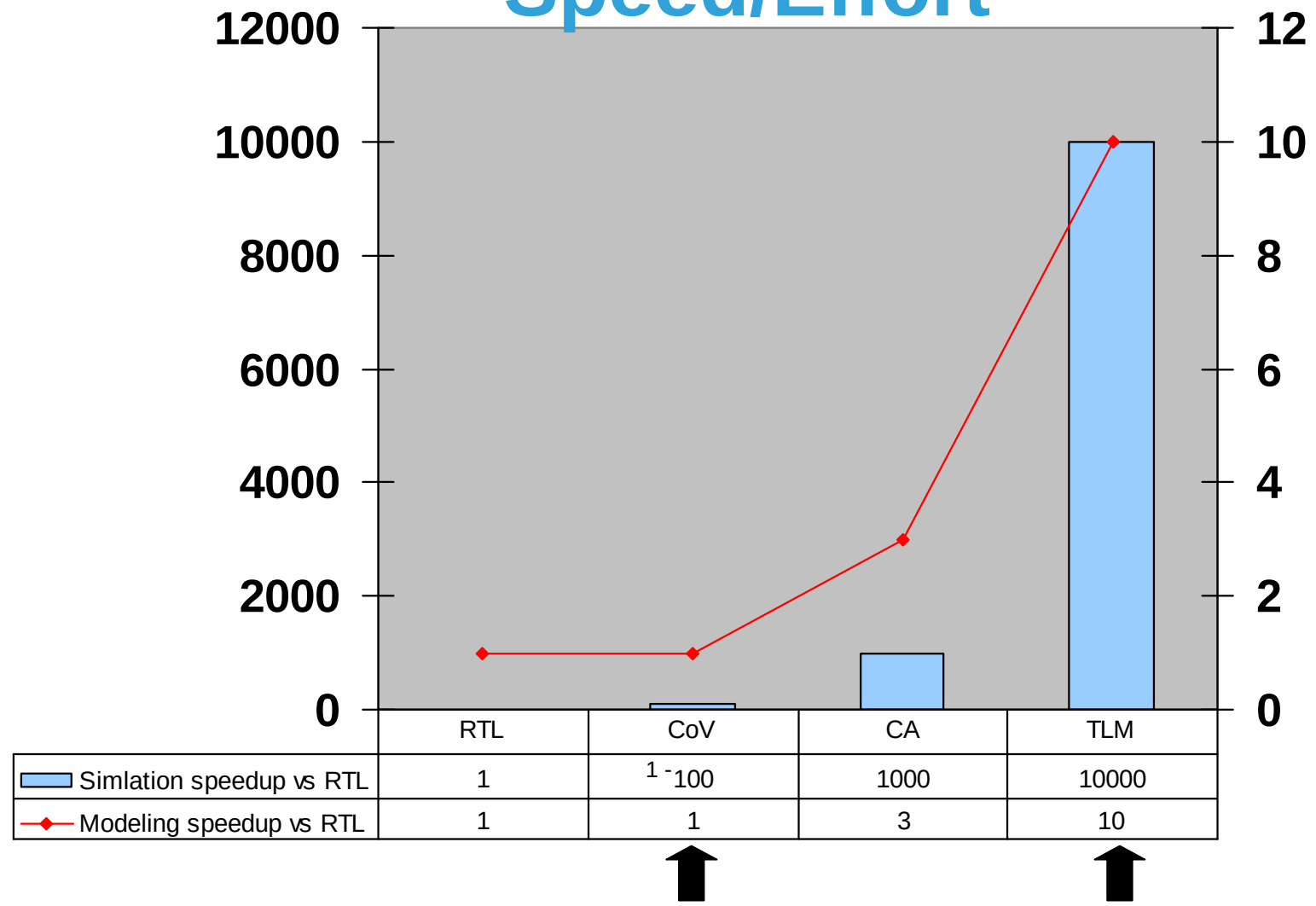
Standard Flow



System 2 RTL design flow

- SoC Micro-architecture (SoC RTL platform)
  - Accurately models cycles, protocols, IP internal states, etc.
  - Accuracy at expense of speed
  - ➢ Reuse RTL models to reduce development cost
- SoC Architecture (SoC TLM platform)
  - Corresponds to *contract* between embedded software and hardware
  - Accurately models register banks, data transfers, system synchronisations
  - Speed at expense of accuracy
  - ➢ Require light-weight modeling effort

SoC Platforms: RTL vs TLM Speed/Effort



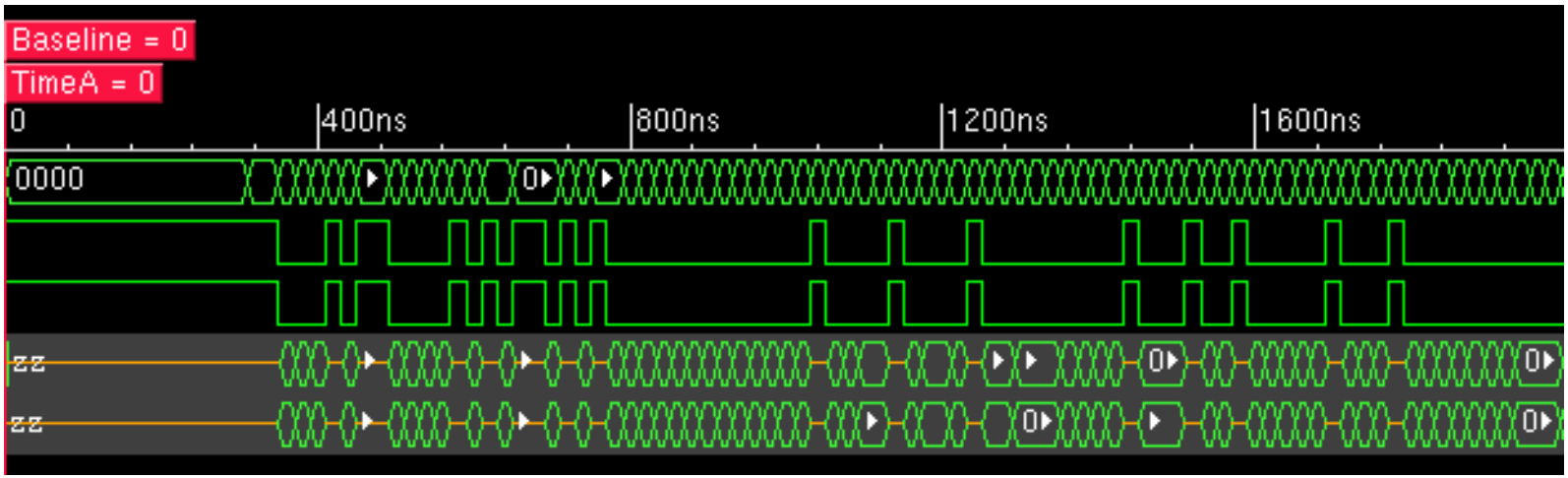
	RTL	CoV	CA	TLM
Simlation speedup vs RTL	1	1 - 100	1000	10000
Modeling speedup vs RTL	1	1	3	10

Disclaimer: numbers provided as rule of thumb…

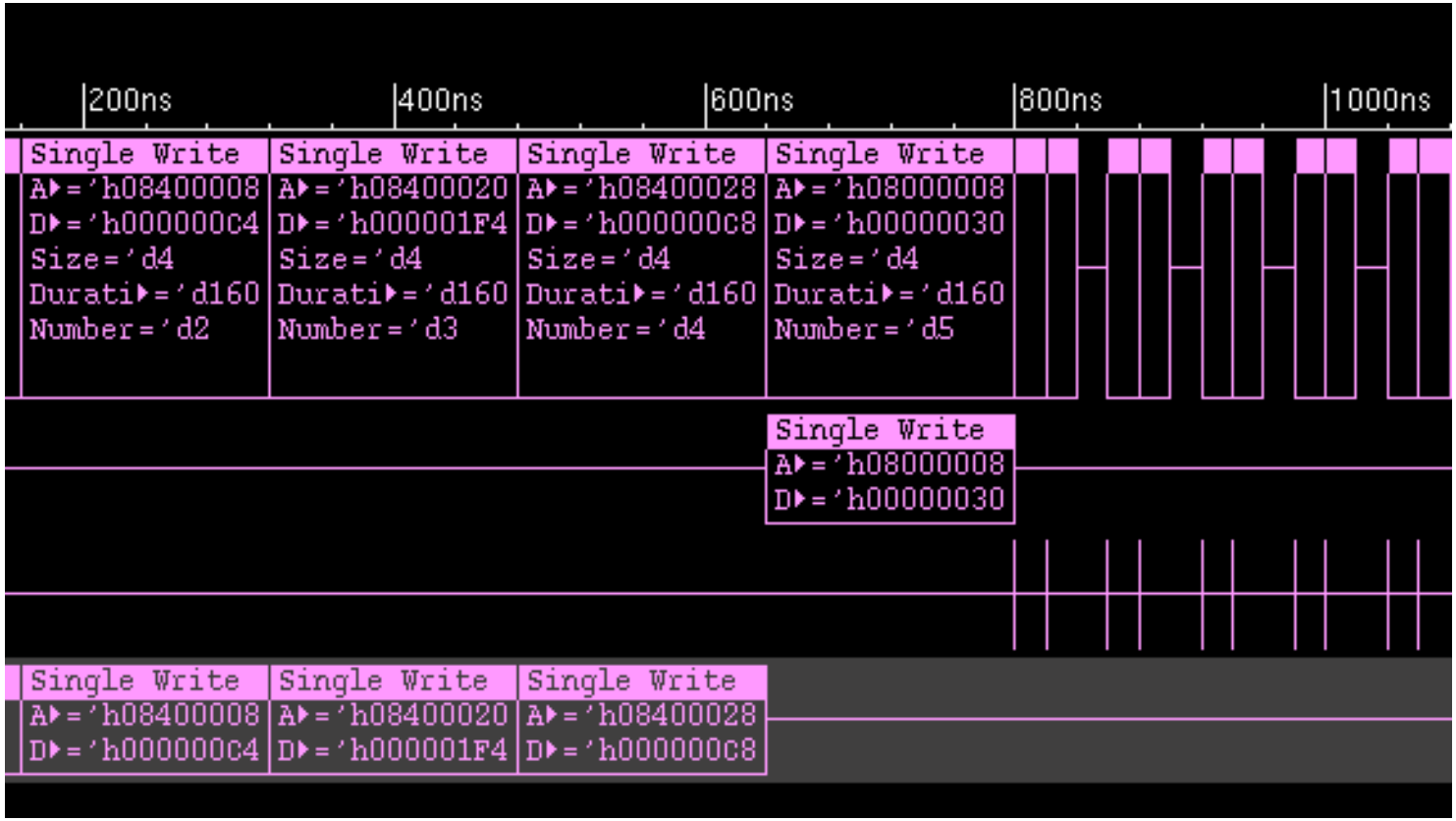
Transaction Level modeling: Definitions

- A system model is composed of a set of modules, that:
  - Execute part of the expected system behavior thanks to one or several concurrent threads
  - Communicate data between each other.
    - Each data set is called a transaction
    - Data sets of variable size
  - Synchronize between each other. A system synchronization is an explicit action
- The abstraction level is the SOC architecture level

RTL traces



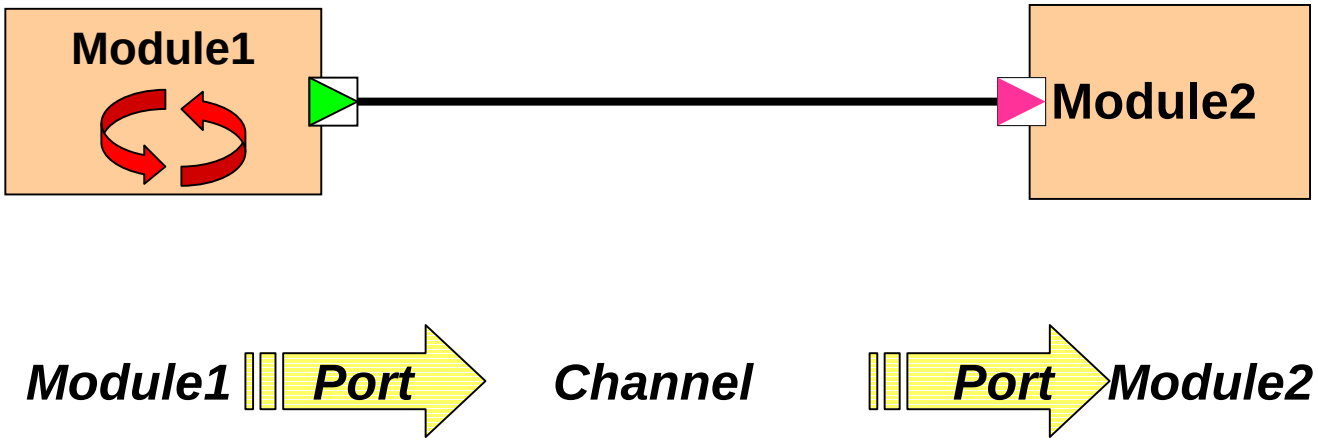
TLM Traces


200ns
400ns
600ns
800ns
1000ns
Single Write
A='h08400008
D='h000000C4
Size='d4
Durati='d160
Number='d2
Single Write
A='h08400020
D='h000001F4
Size='d4
Durati='d160
Number='d3
Single Write
A='h08400028
D='h000000C8
Size='d4
Durati='d160
Number='d4
Single Write
A='h08000008
D='h00000030
Size='d4
Durati='d160
Number='d5
Single Write
A='h08000008
D='h00000030
Single Write
A='h08400008
D='h000000C4
Single Write
A='h08400020
D='h000001F4
Single Write
A='h08400028
D='h000000C8

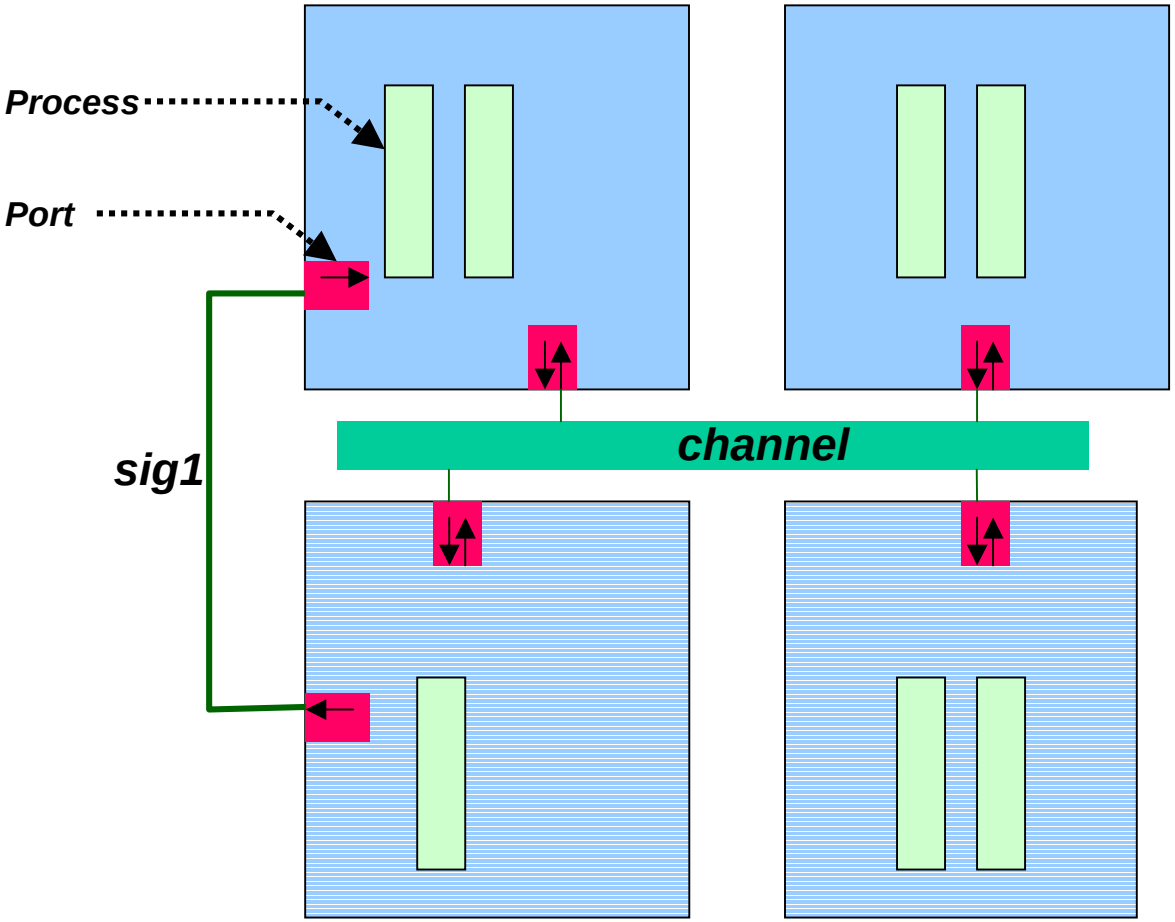
SystemC

- Used to build TLM platforms
- Built on top of C++ (Only additional classes)
- Provides
  - A scheduler
  - Notion of component (hierarchical)
  - Some basic communication elements
  - The infrastructure to build more abstract channels

The TLM interface



Example of system



Outline

- Introduction about Systems on a Chip
- Verification approach and tool chain
- Model of SystemC
- An example

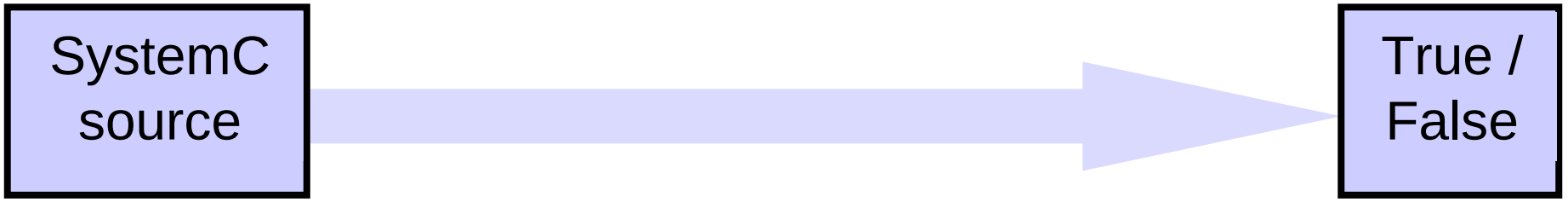
Formal Verification

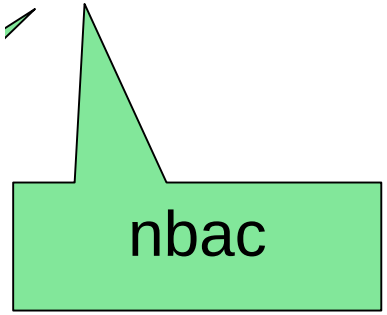
- Automatic Vs Interactive
- Property-checking Vs equivalence checking
- System-level Vs Block-level
- Safety Vs Liveness

Difficulties

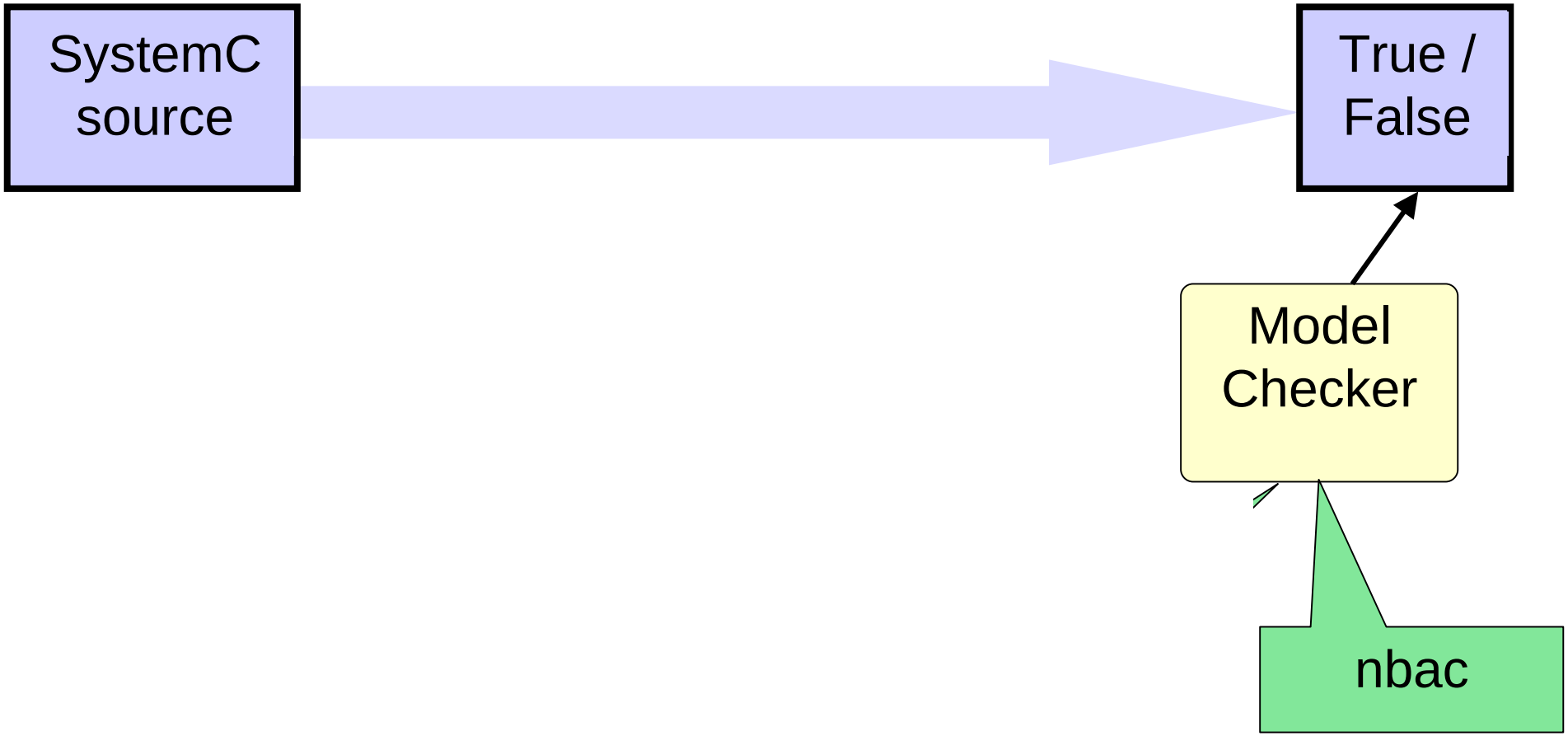
- Extract a formal model of the system to be verified (which is written in C++ !!)
- Deal with large systems
- Deal with data-dependant properties

Tool chain

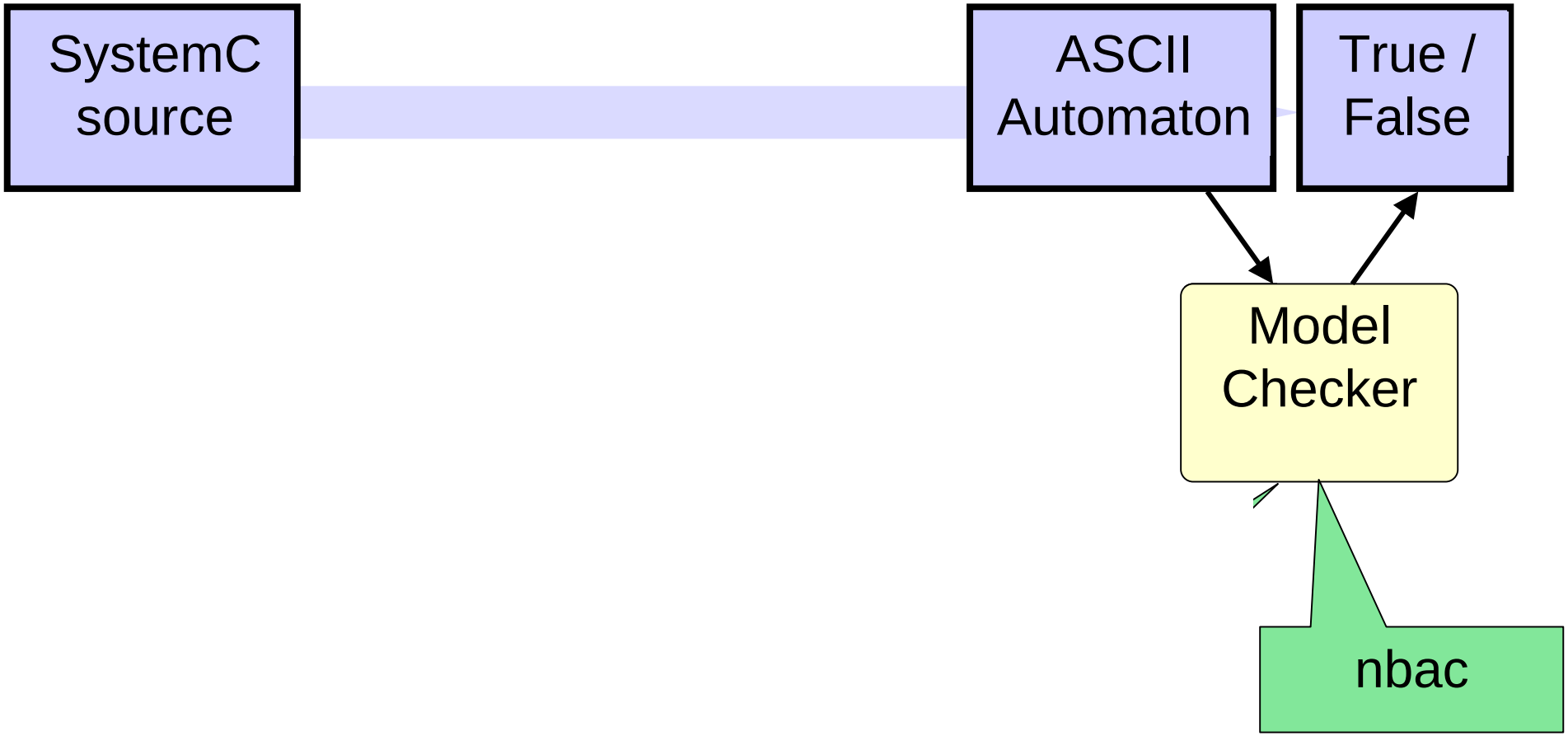




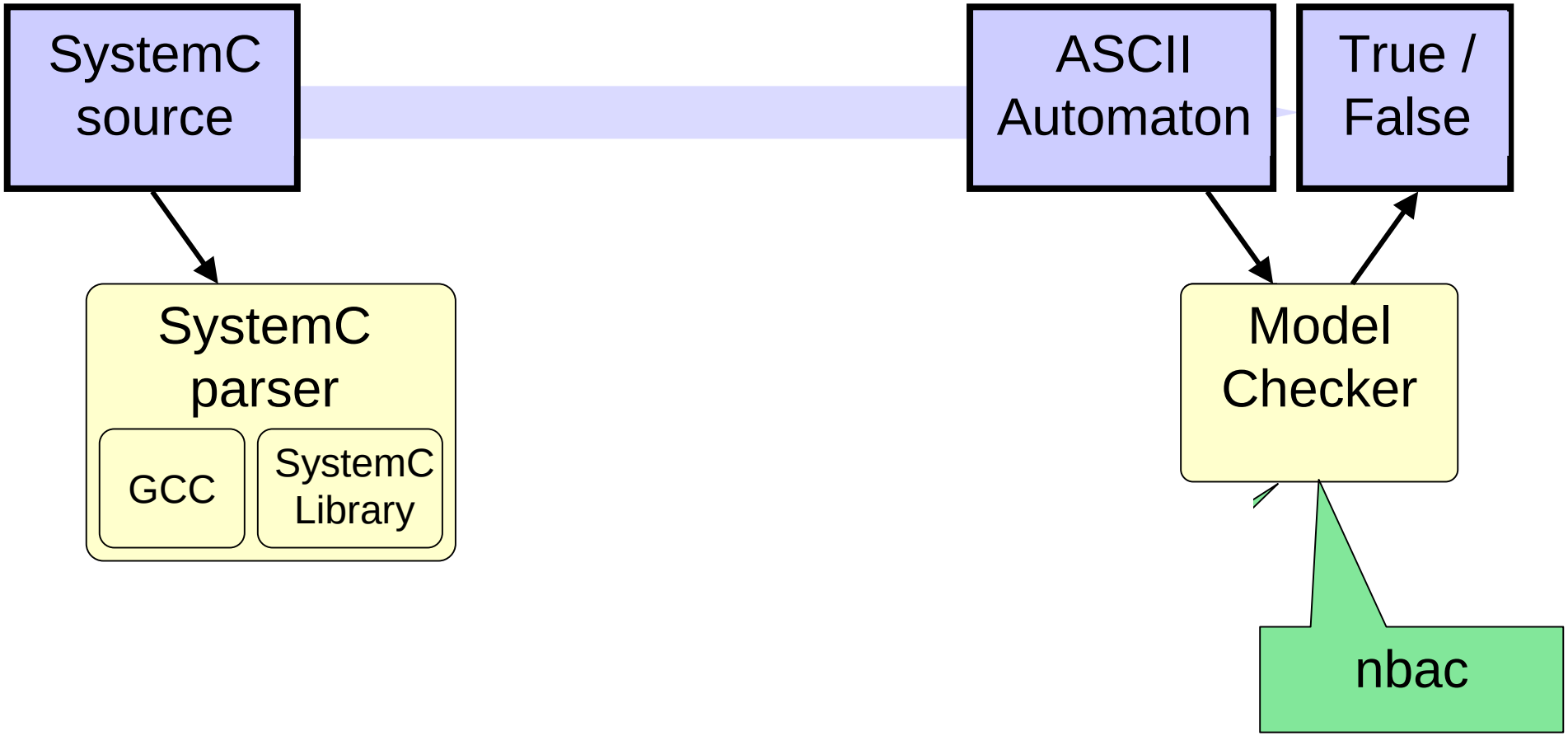
Tool chain



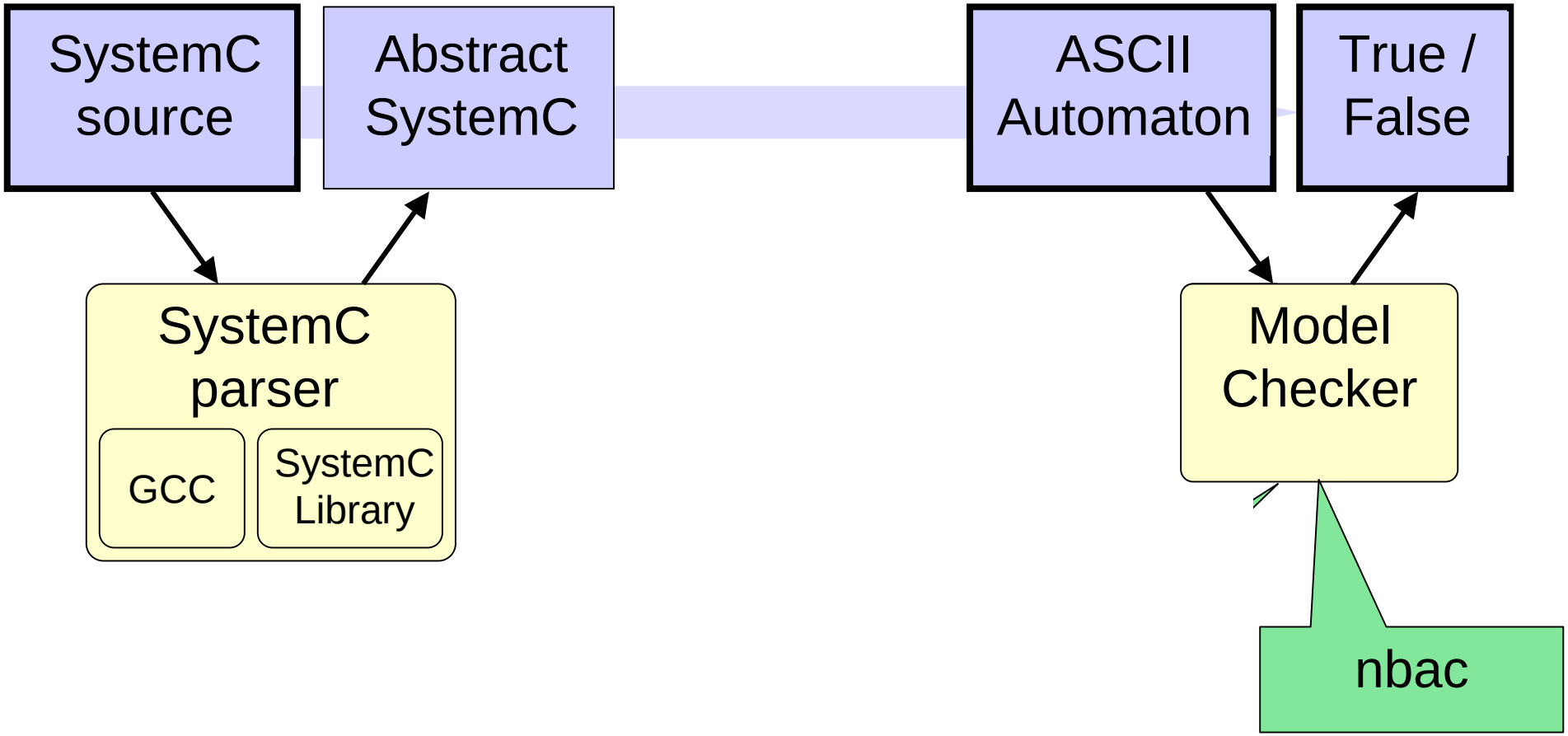
Tool chain


SystemC source
ASCII Automaton
True / False
Model Checker
nbac

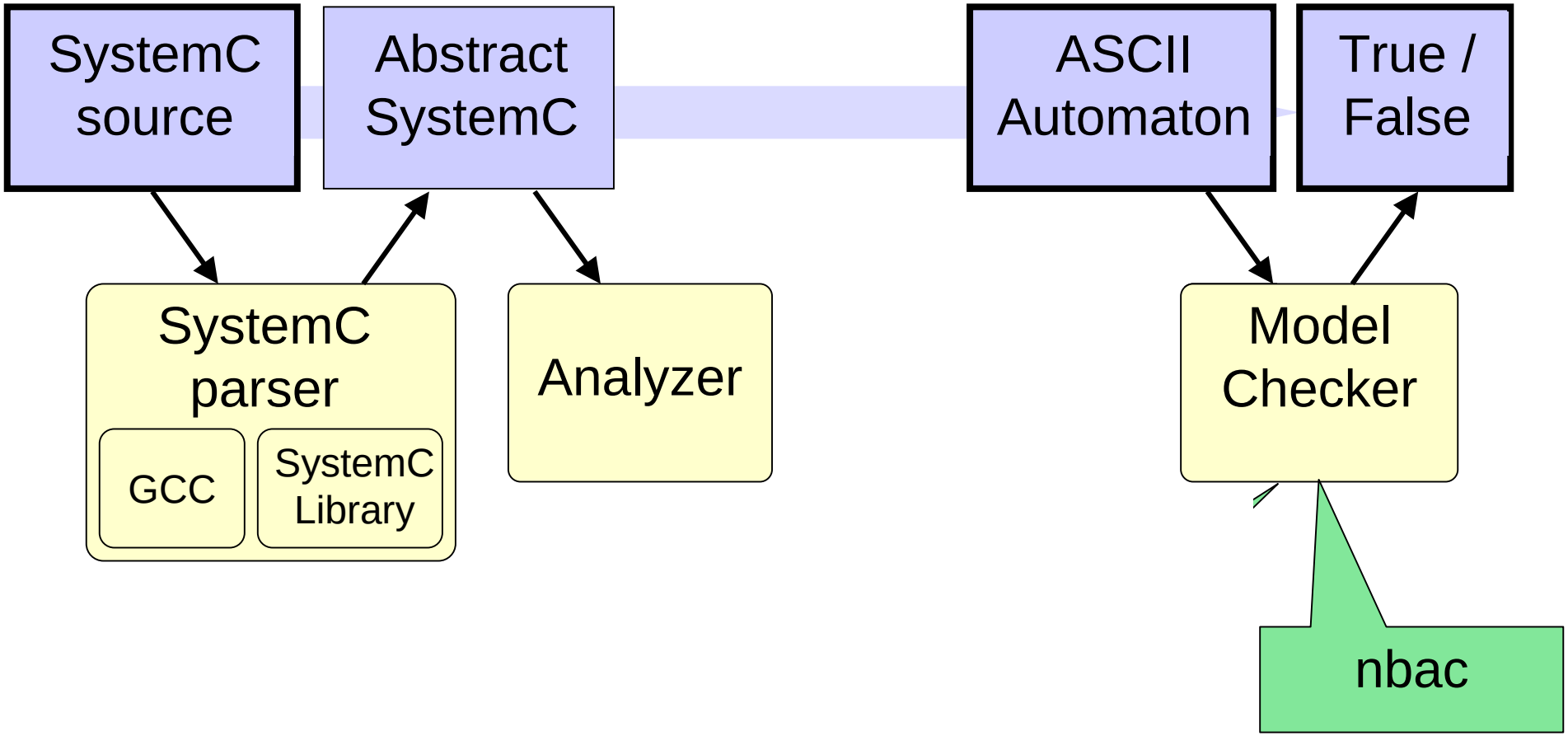
Tool chain


SystemC source
ASCII Automaton
True / False
SystemC parser
GCC
SystemC Library
Model Checker
nbac

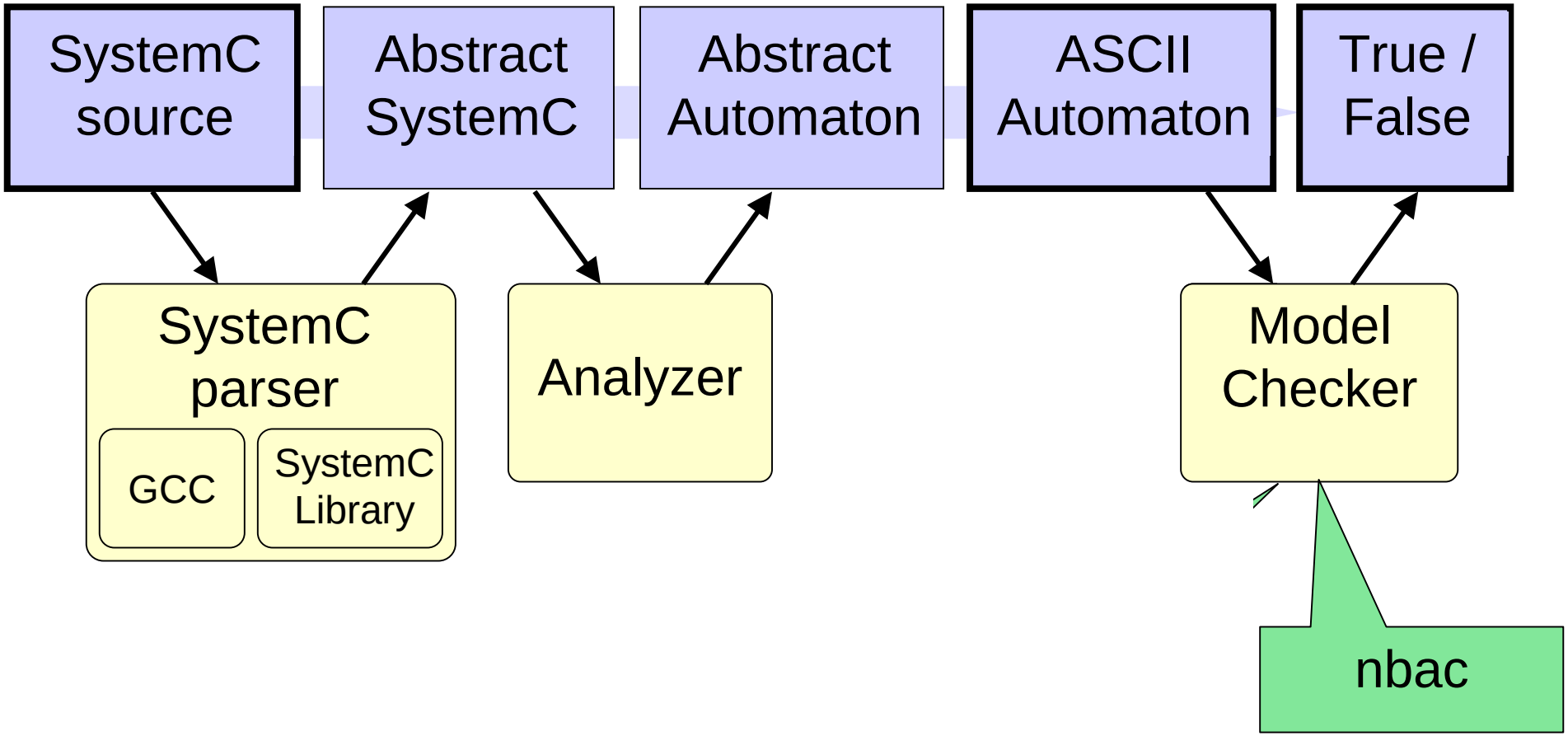
Tool chain


SystemC source
Abstract SystemC
ASCII Automaton
True / False
SystemC parser
GCC
SystemC Library
Model Checker
nbac

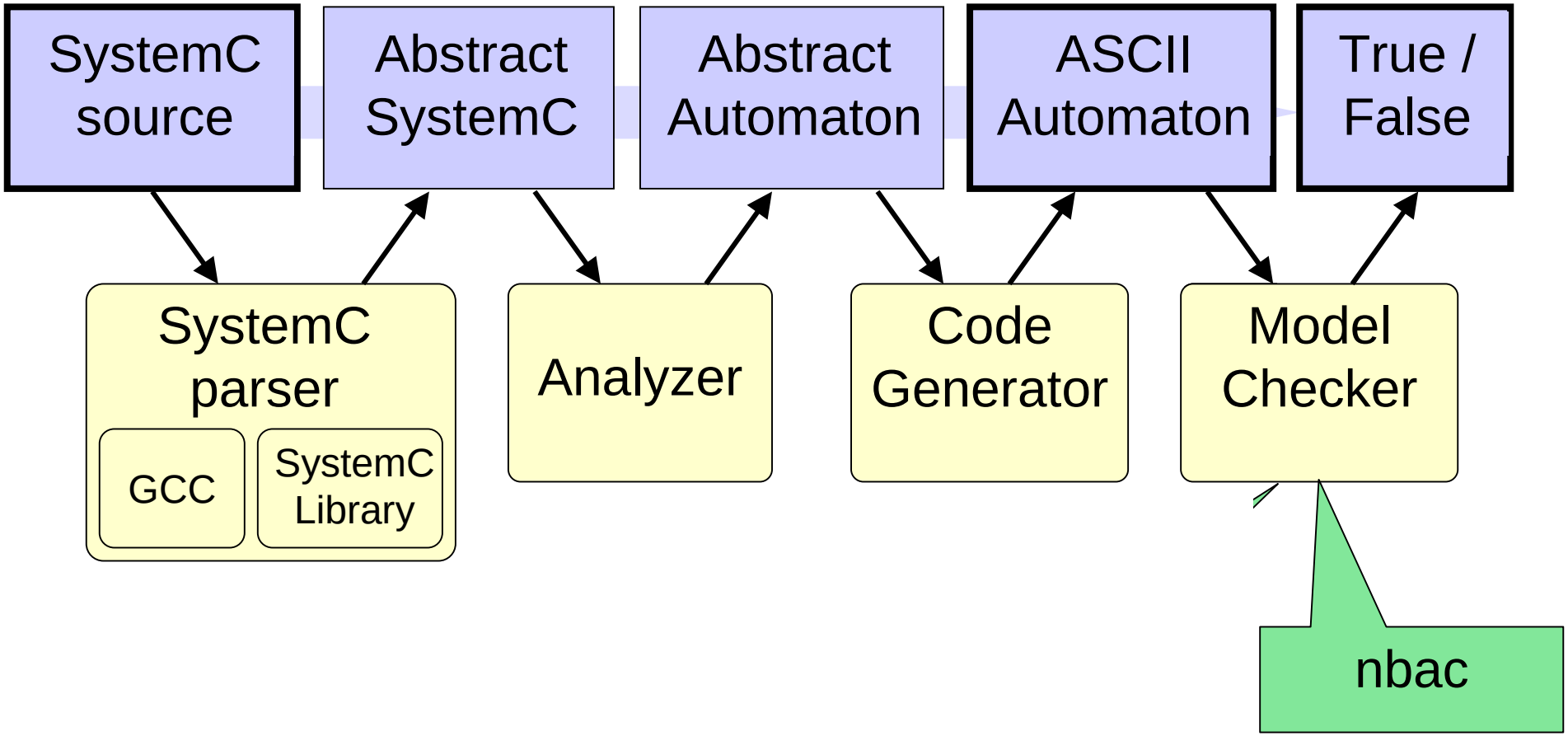
Tool chain


SystemC source
Abstract SystemC
ASCII Automaton
True / False
SystemC parser
GCC
SystemC Library
Analyzer
Model Checker
nbac

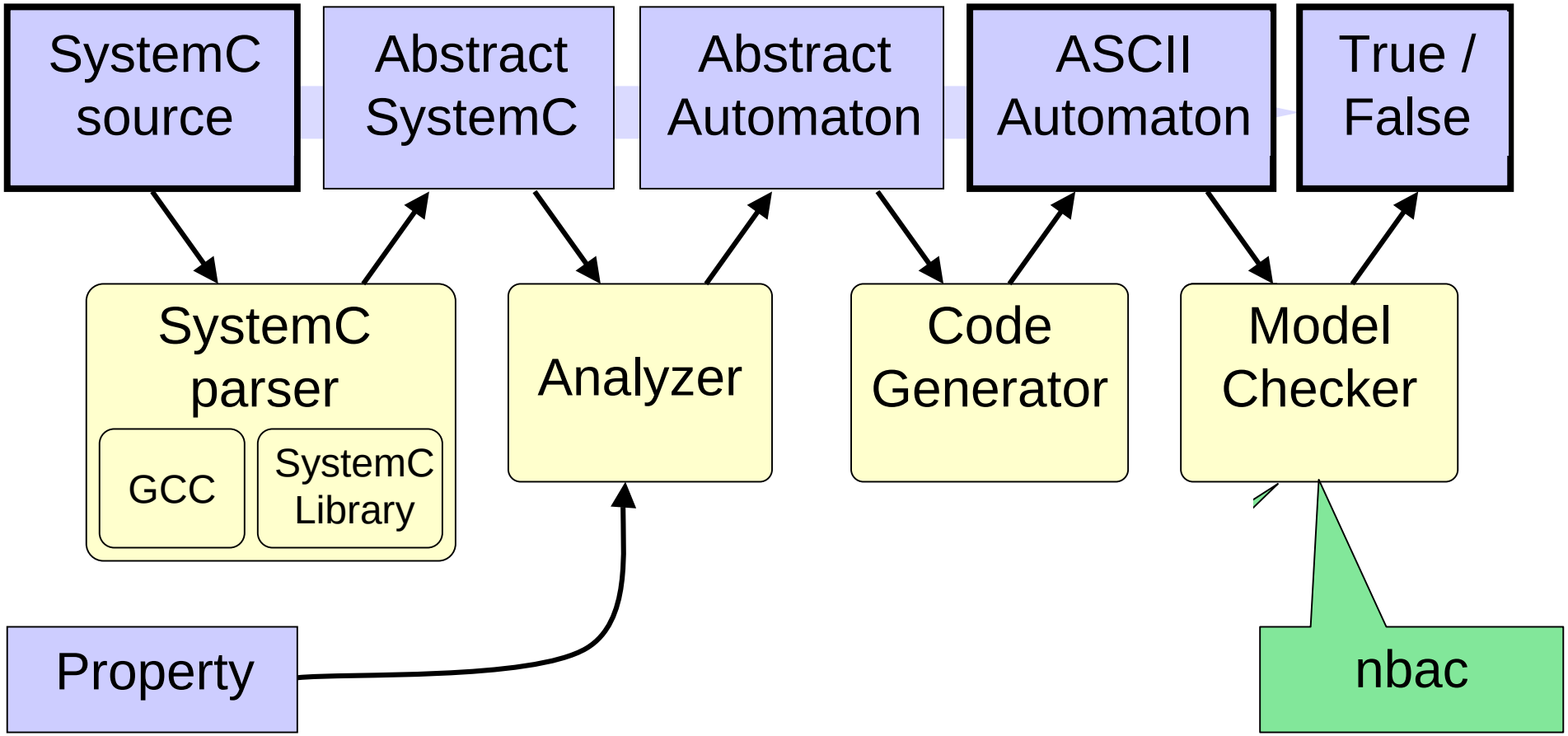
Tool chain


SystemC source
Abstract SystemC
Abstract Automaton
ASCII Automaton
True / False
SystemC parser
GCC
SystemC Library
Analyzer
Model Checker
nbac

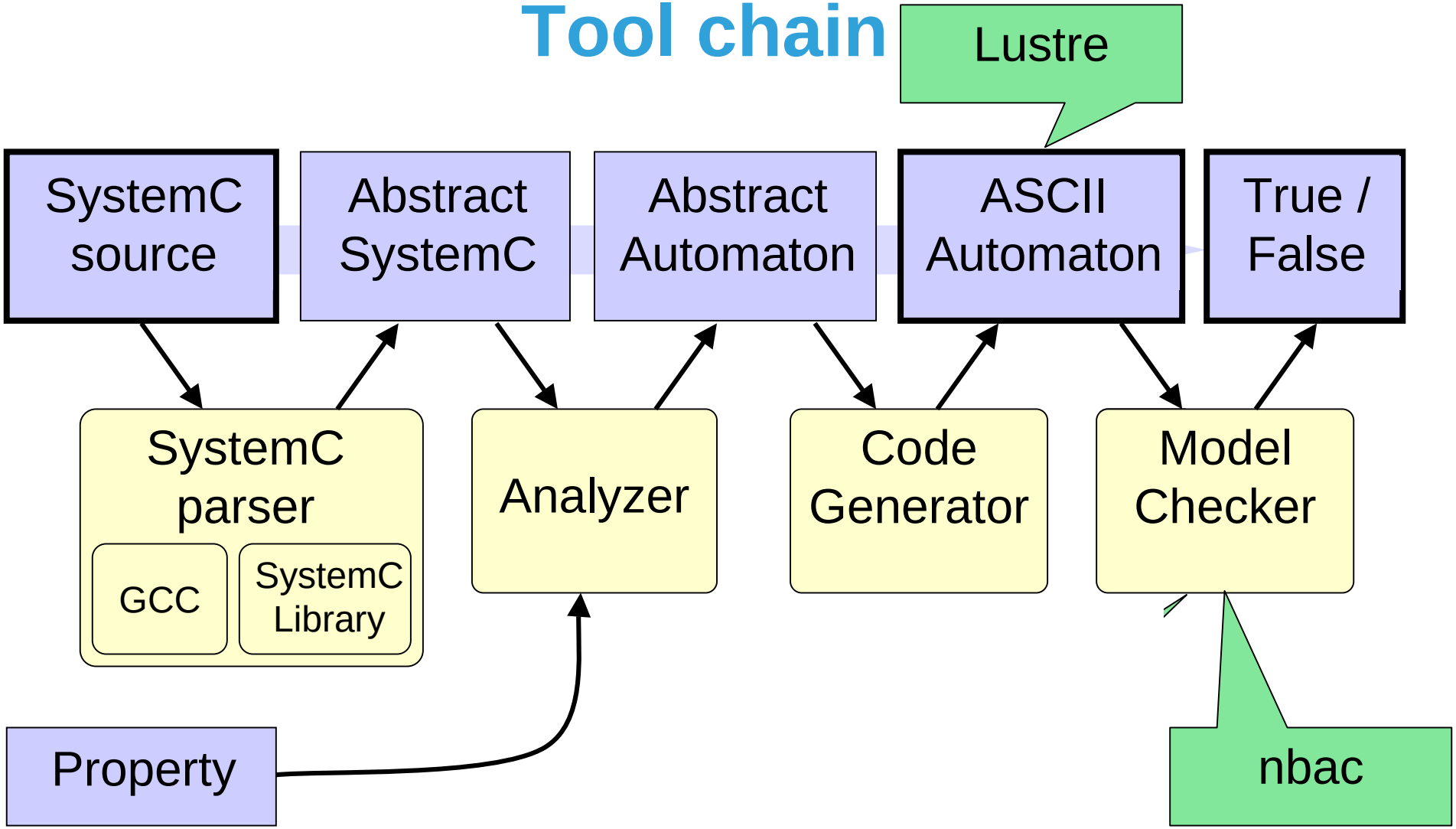
Tool chain


SystemC source
Abstract SystemC
Abstract Automaton
ASCII Automaton
True / False
SystemC parser
GCC
SystemC Library
Analyzer
Code Generator
Model Checker
nbac

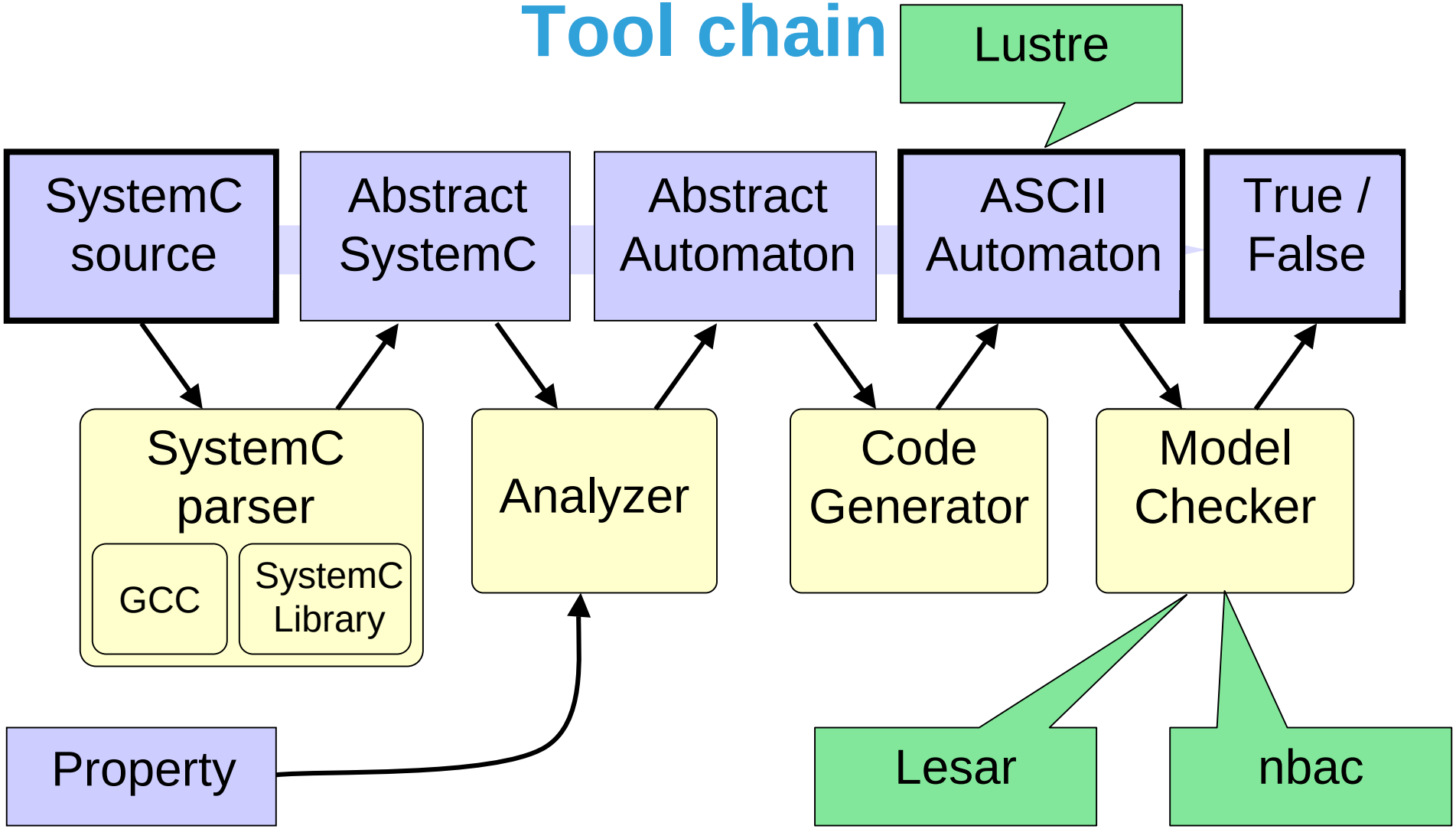
Tool chain


SystemC source
Abstract SystemC
Abstract Automaton
ASCII Automaton
True / False
SystemC parser
GCC
SystemC Library
Analyzer
Code Generator
Model Checker
Property
nbac

Tool chain


Lustre
SystemC source
Abstract SystemC
Abstract Automaton
ASCII Automaton
True / False
SystemC parser
GCC
SystemC Library
Analyzer
Code Generator
Model Checker
Property
nbac

Tool chain


Lustre
SystemC source
Abstract SystemC
Abstract Automaton
ASCII Automaton
True / False
SystemC parser
GCC
SystemC Library
Analyzer
Code Generator
Model Checker
Property
Lesar
nbac

Parsing SystemC

- Idea
  - Parse the body of processes with GCC
  - Execute the constructors to get the netlist
- Implementation
  - Modified version of both GCC and SystemC/TLM
  - The parser is linked to the system

Expression of the property

- Assertions in the source code (→reachability)

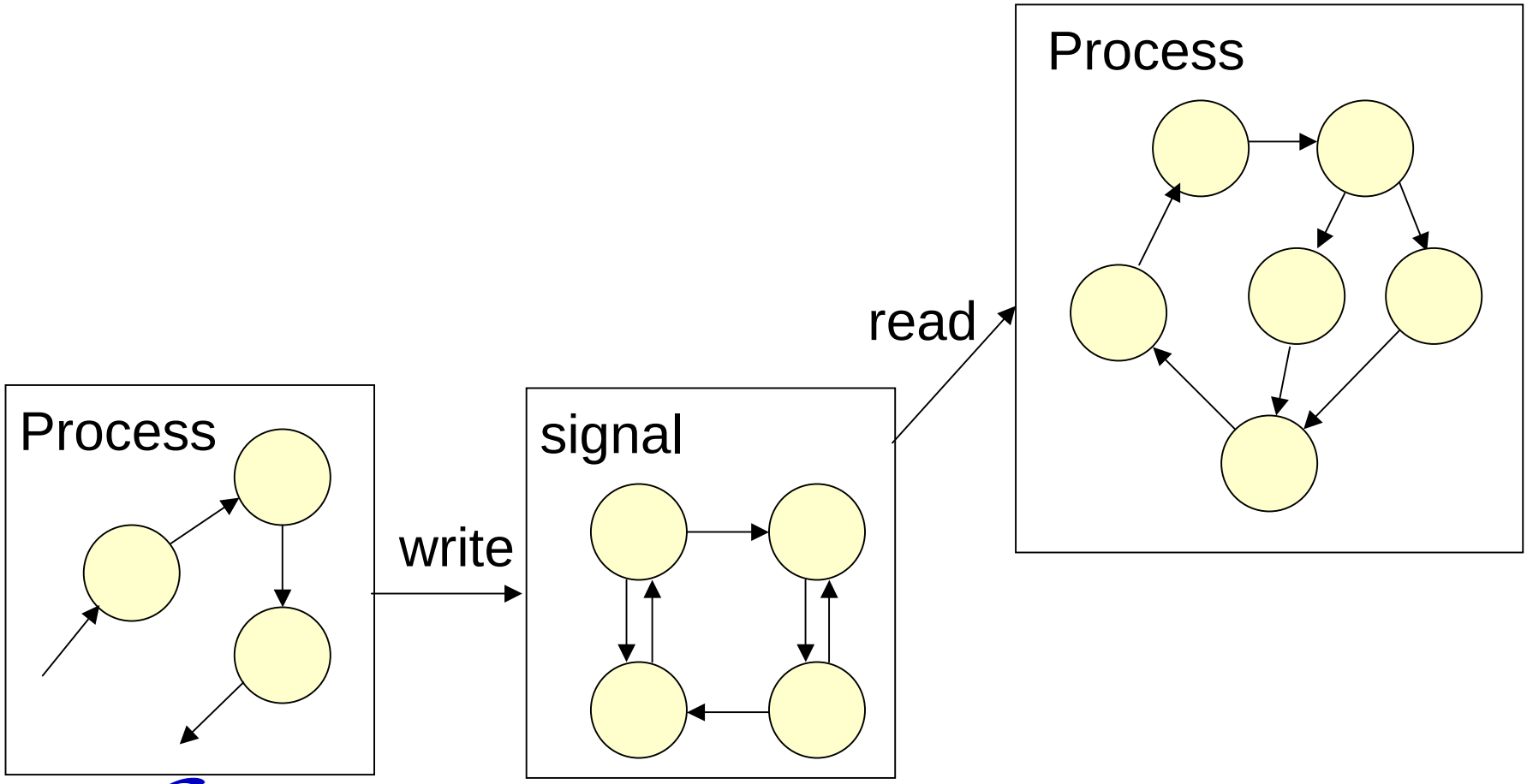
```
ASSERT(x != 3);
→if (!(x!=3)) {is_this_reachable();}
```

- Generic properties
  - A signal is never written on twice
  - Mutual exclusion

Outline

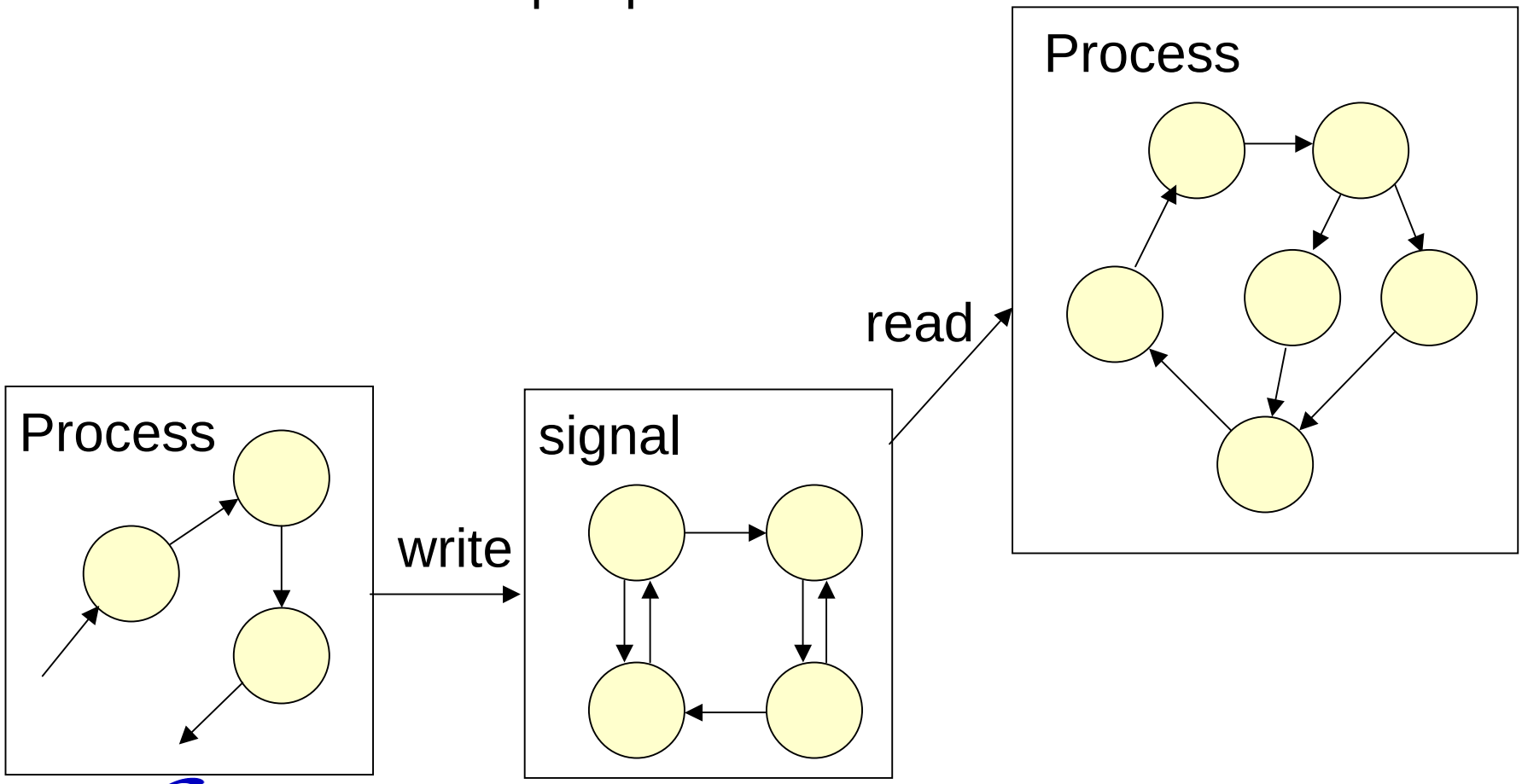
- Introduction about Systems on a Chip
- Verification approach and tool chain
- Model of SystemC
- An example

Generation of the automata



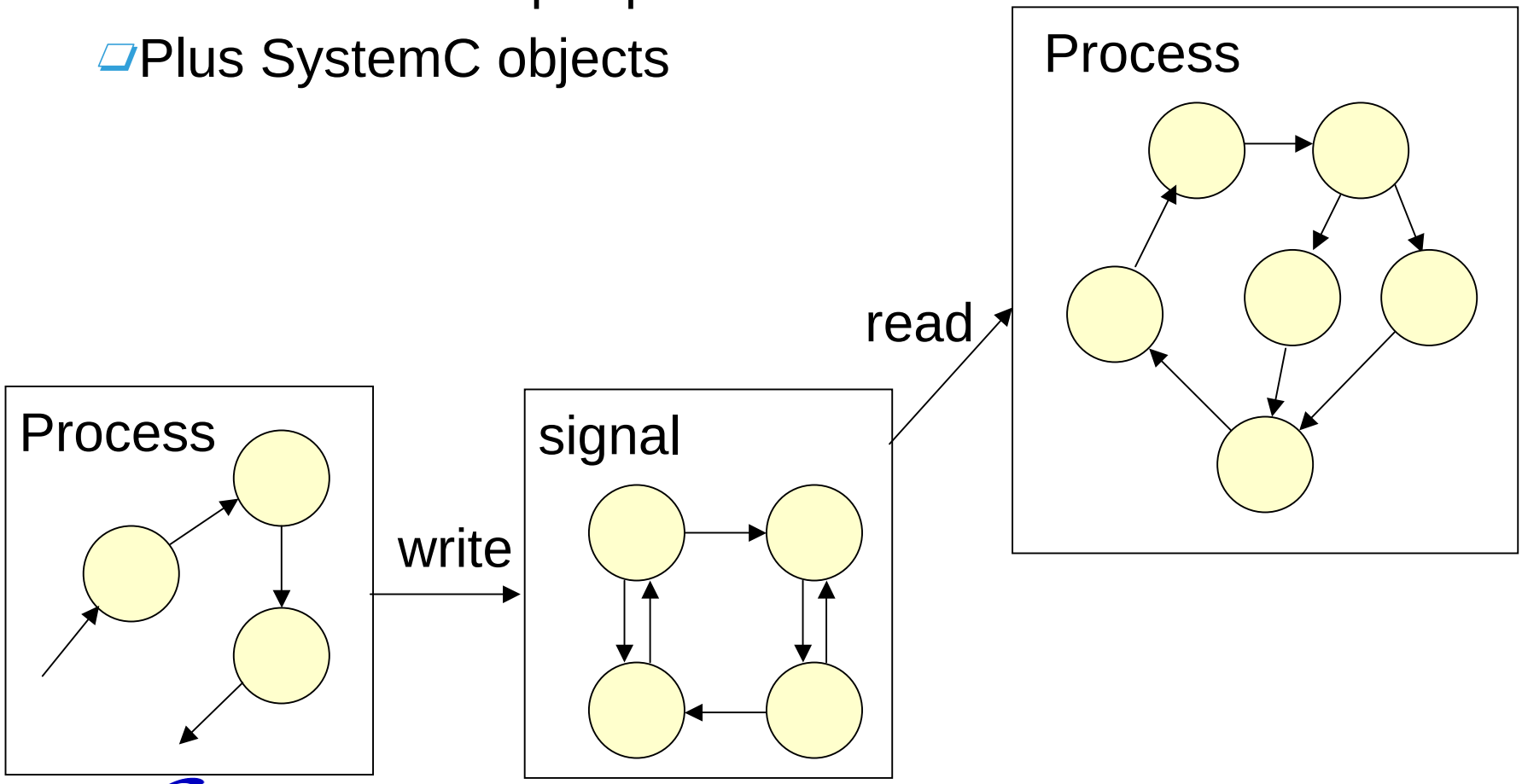
Generation of the automata

- One automaton per process



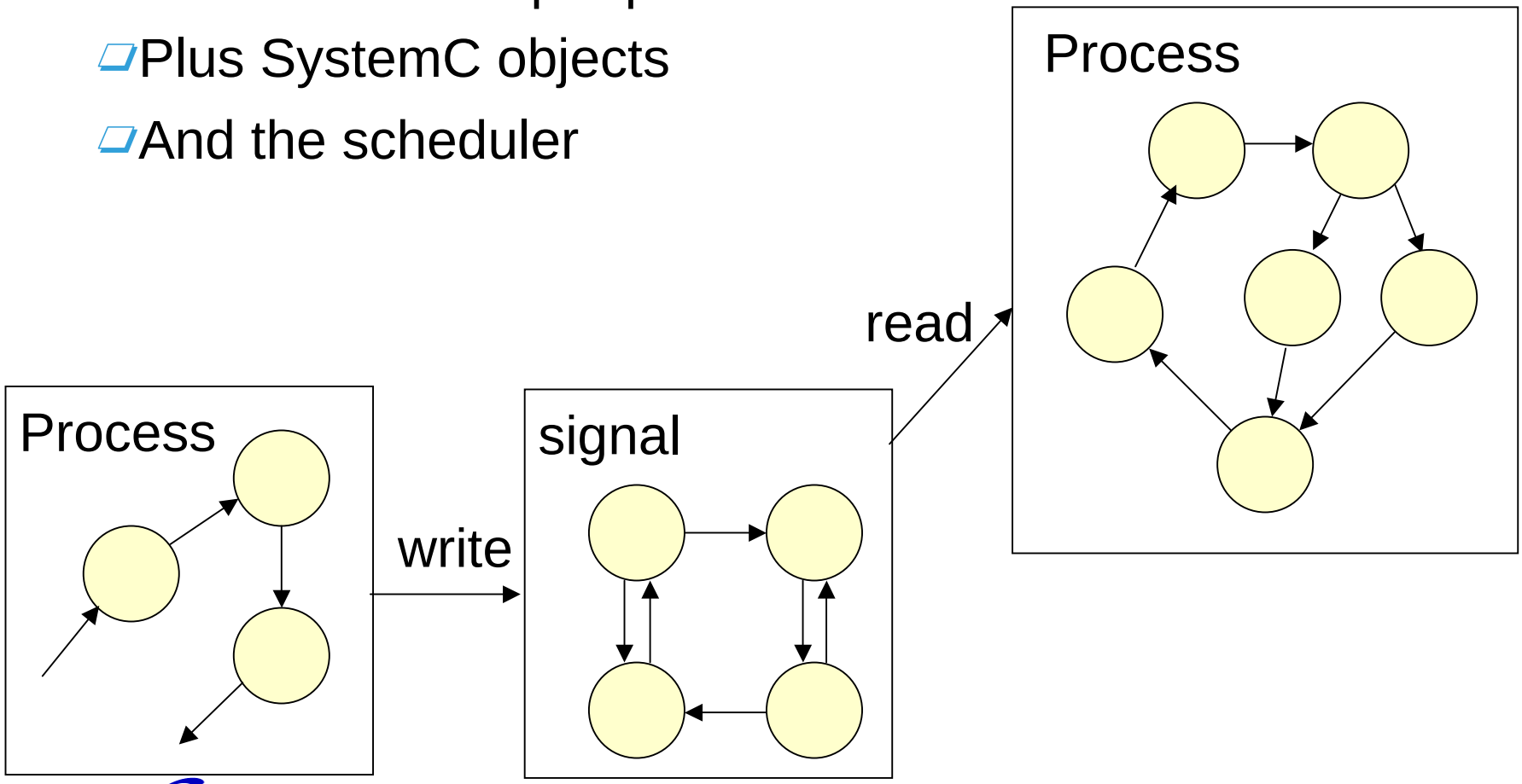
Generation of the automata

- One automaton per process
- Plus SystemC objects



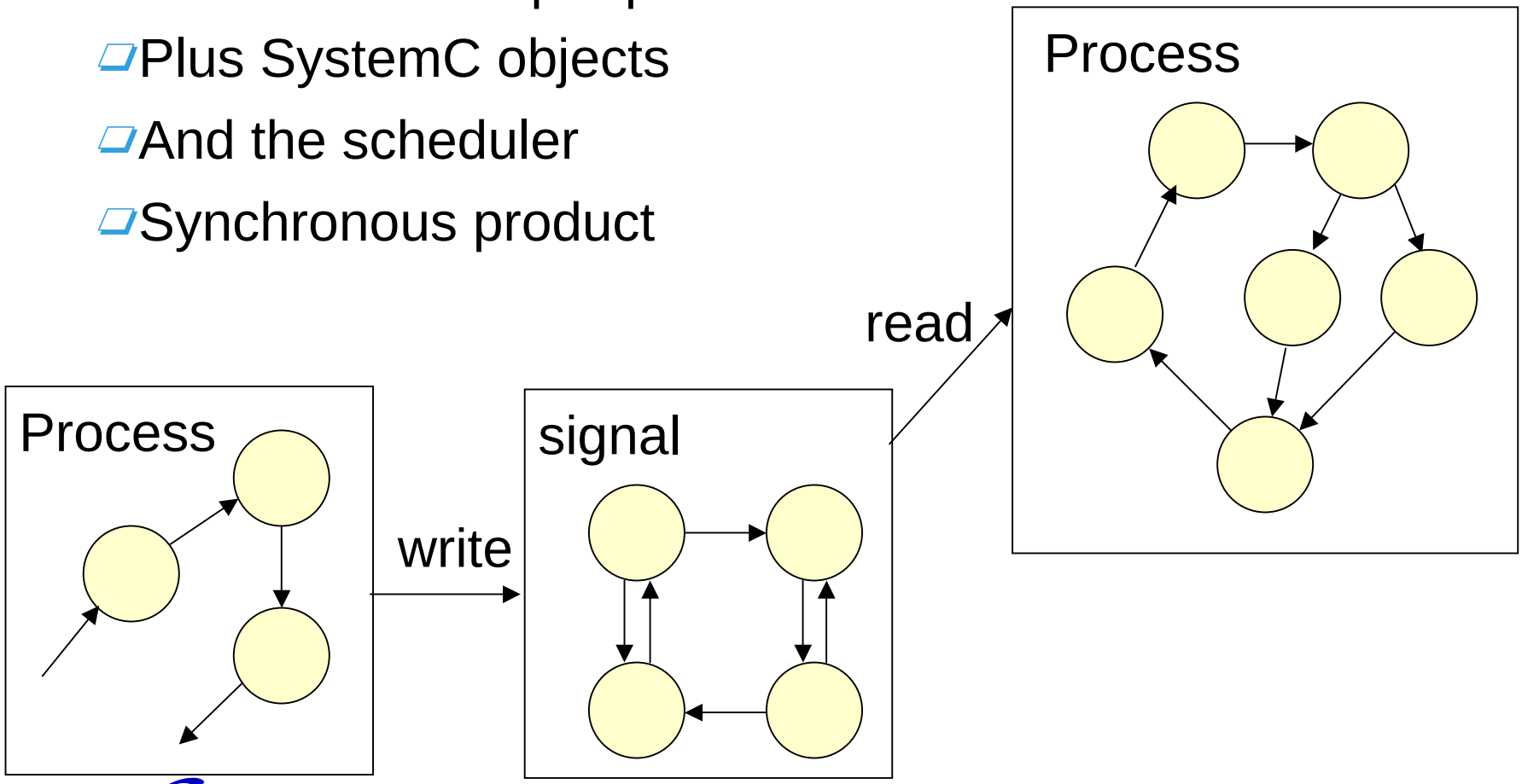
Generation of the automata

- One automaton per process
- Plus SystemC objects
- And the scheduler



Generation of the automata

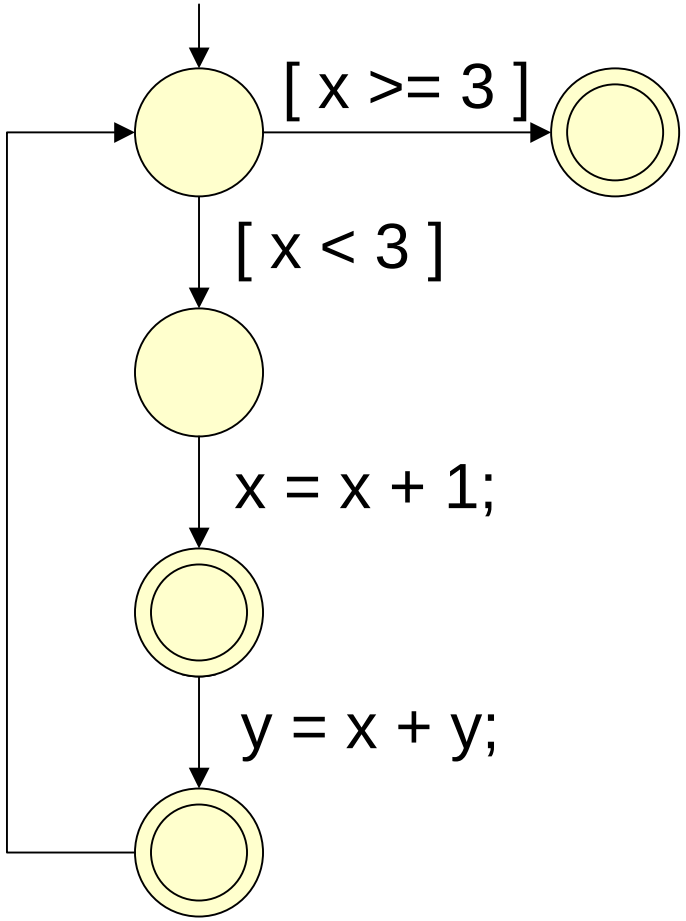
- One automaton per process
- Plus SystemC objects
- And the scheduler
- Synchronous product



C++ constructs

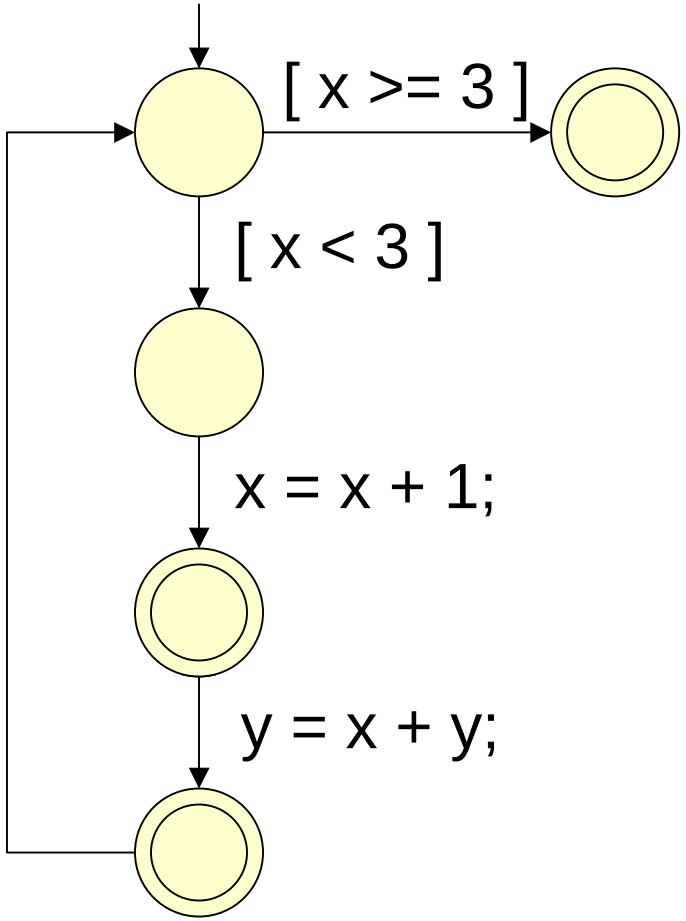
- Control flow graph extracted from the source
- Inductive construction
- One case per C++ instruction

C++ constructs : The While loop



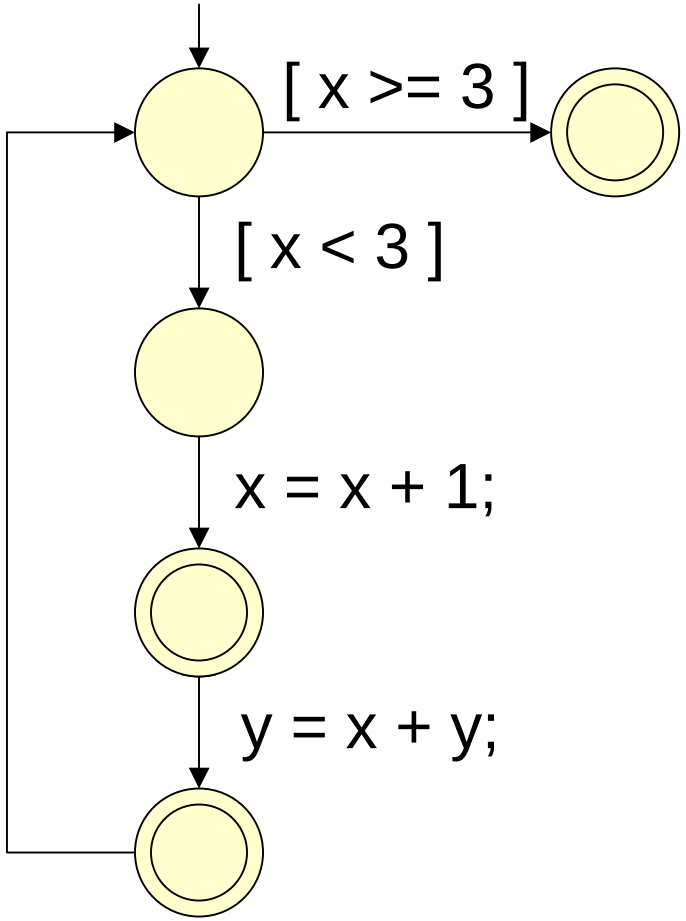
C++ constructs : The While loop

```
x = x + 1;
```



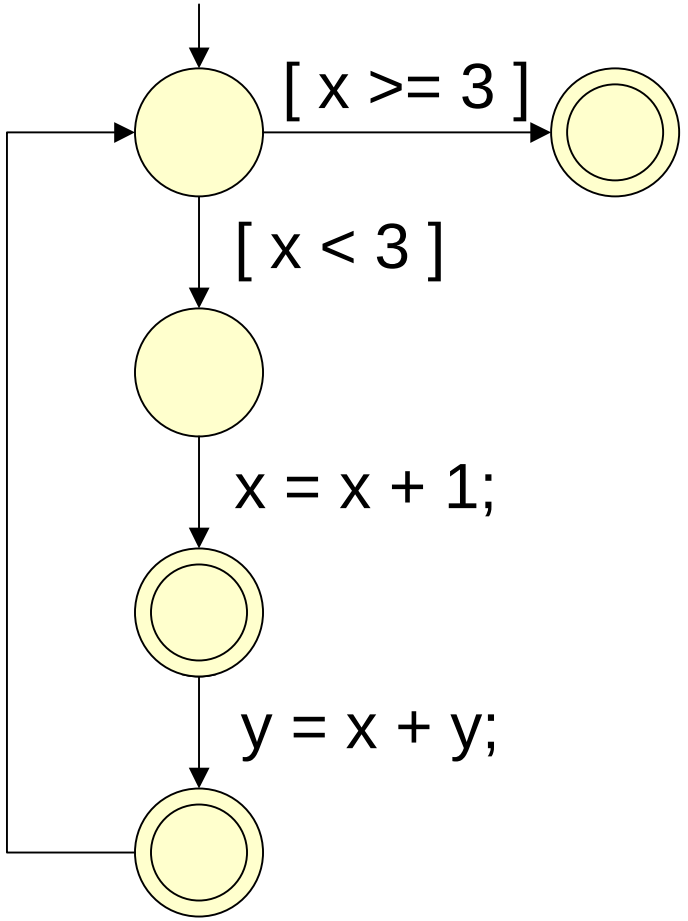
C++ constructs : The While loop

```
x = x + 1;
y = x + y;
```



C++ constructs : The While loop

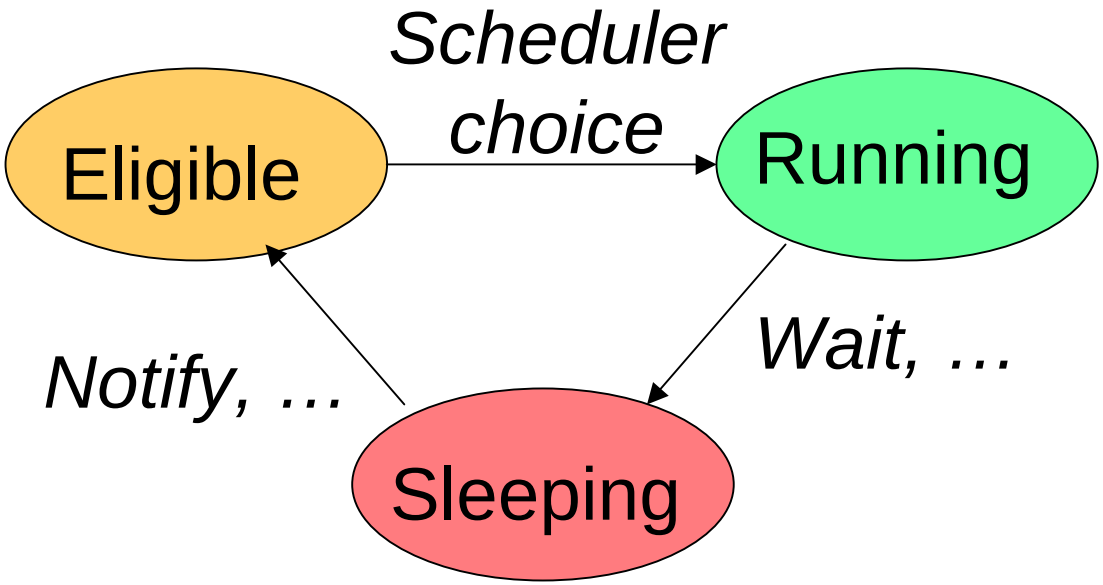
```
While (x < 3) {
   x = x + 1;
   y = x + y;
}
```



SystemC constructs

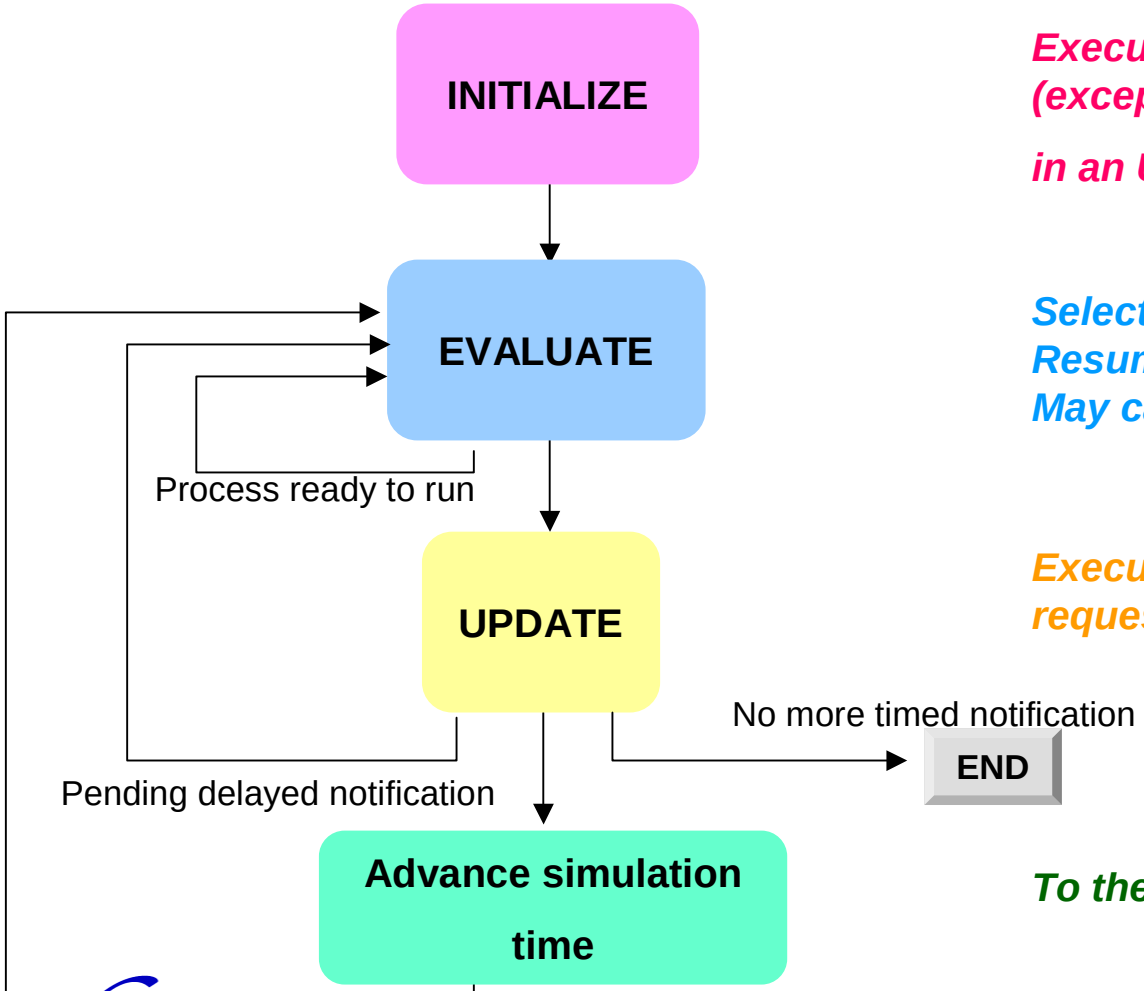
- Automaton pattern for
  - Each object
  - The scheduler
  - The “state” of a process : Running, Eligible, or Sleeping
- Instantiation depends on the model

SystemC scheduler



Note: Non-preemptive scheduler

SystemC scheduler



Execute all processes (except SC_CTHREADs)

in an UNSPECIFIED order

Select a process ready to run
Resume execution
May cause immediate notification

Execute update() calls from request_update() calls (prim channels)

To the earliest timed notification

Pattern for a signal

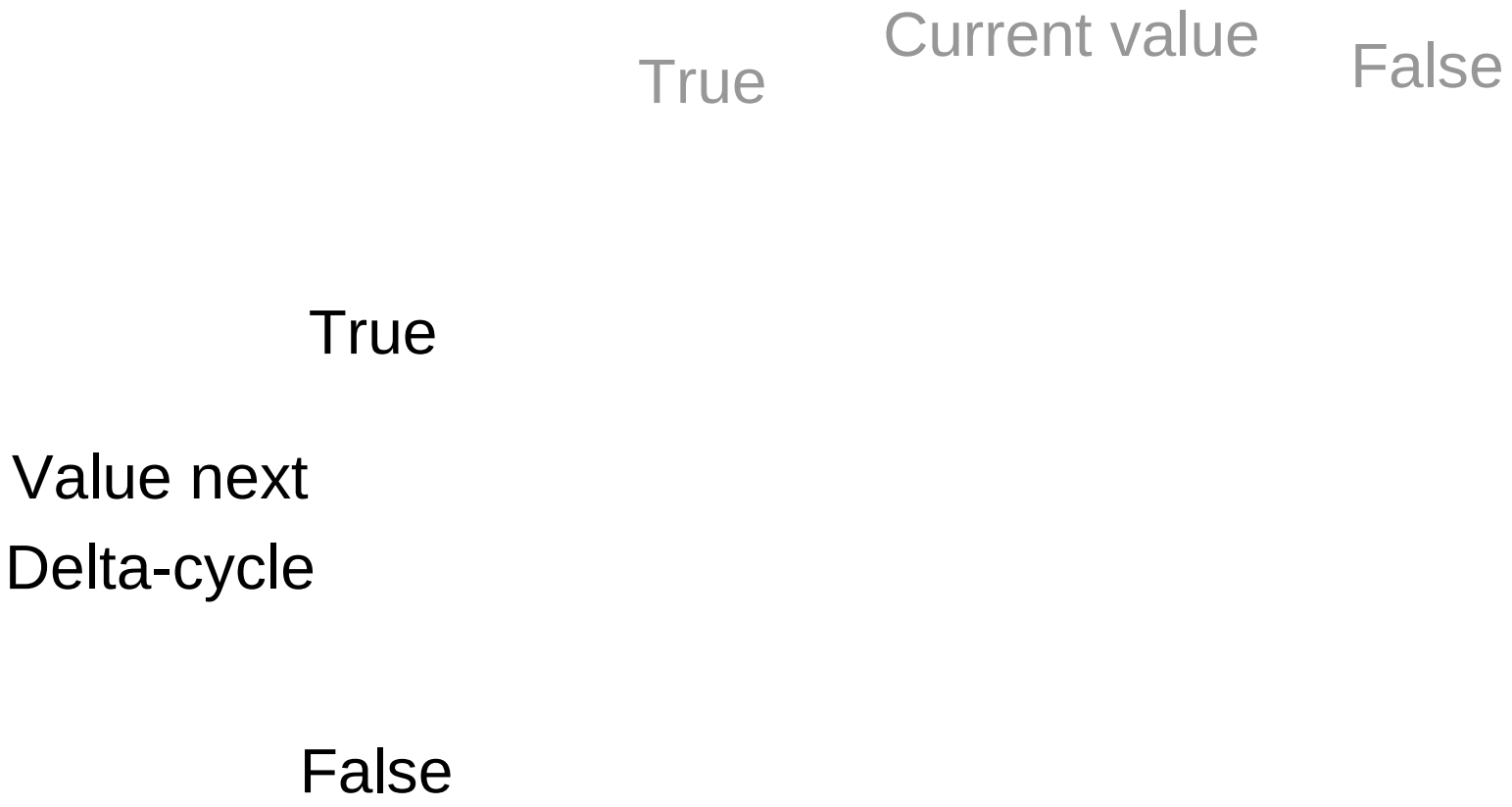
Pattern for a signal

Current value

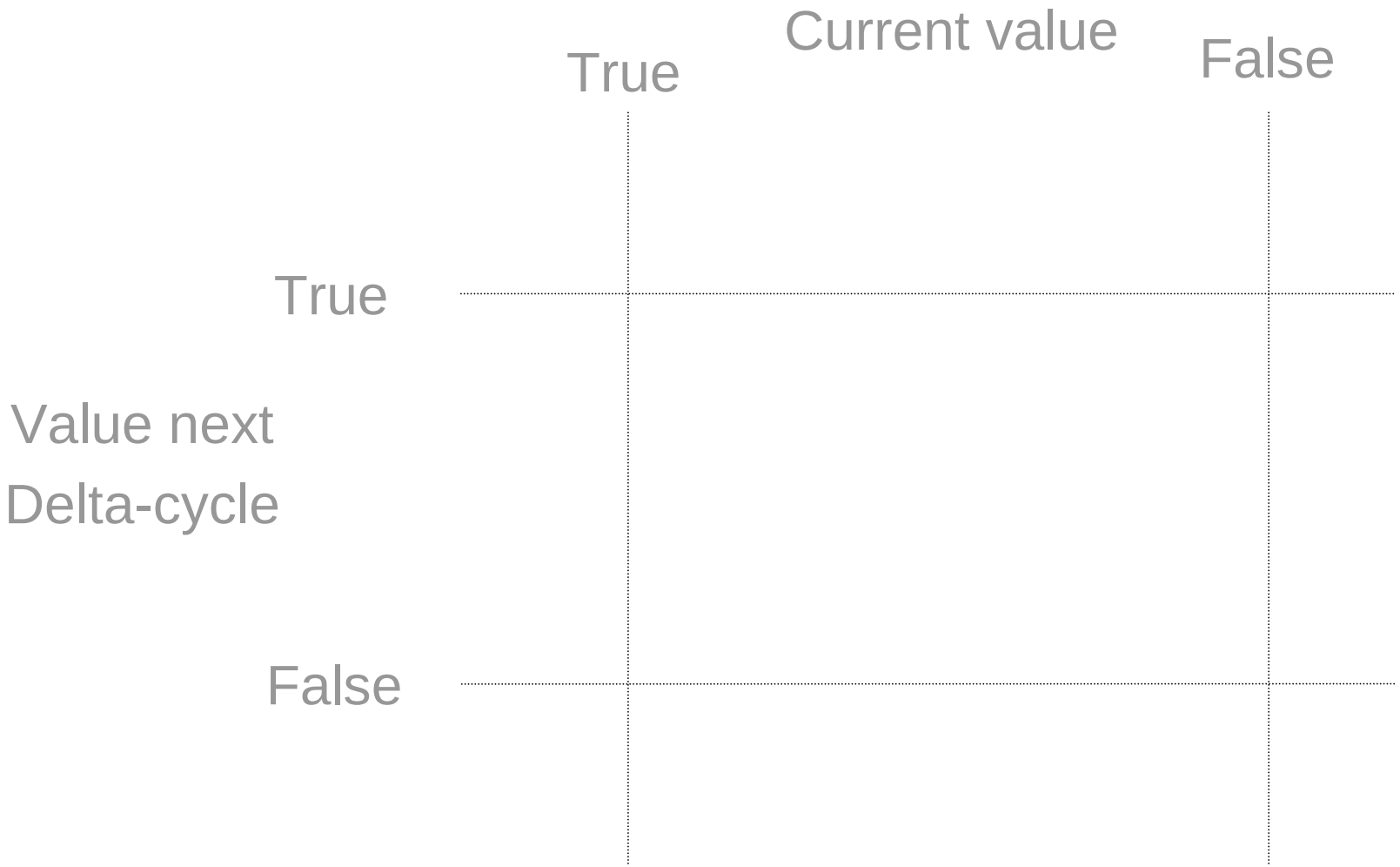
True

False

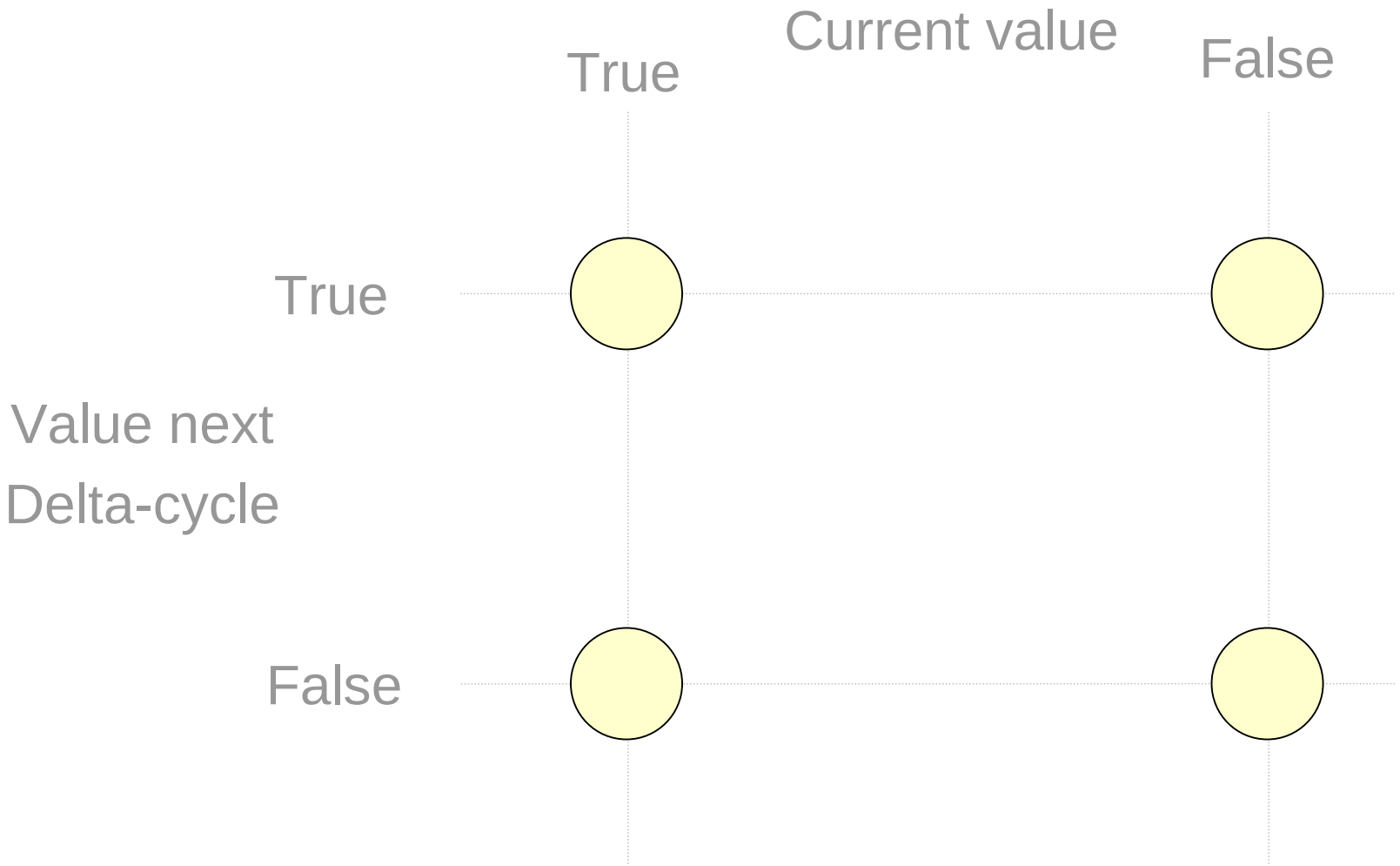
Pattern for a signal



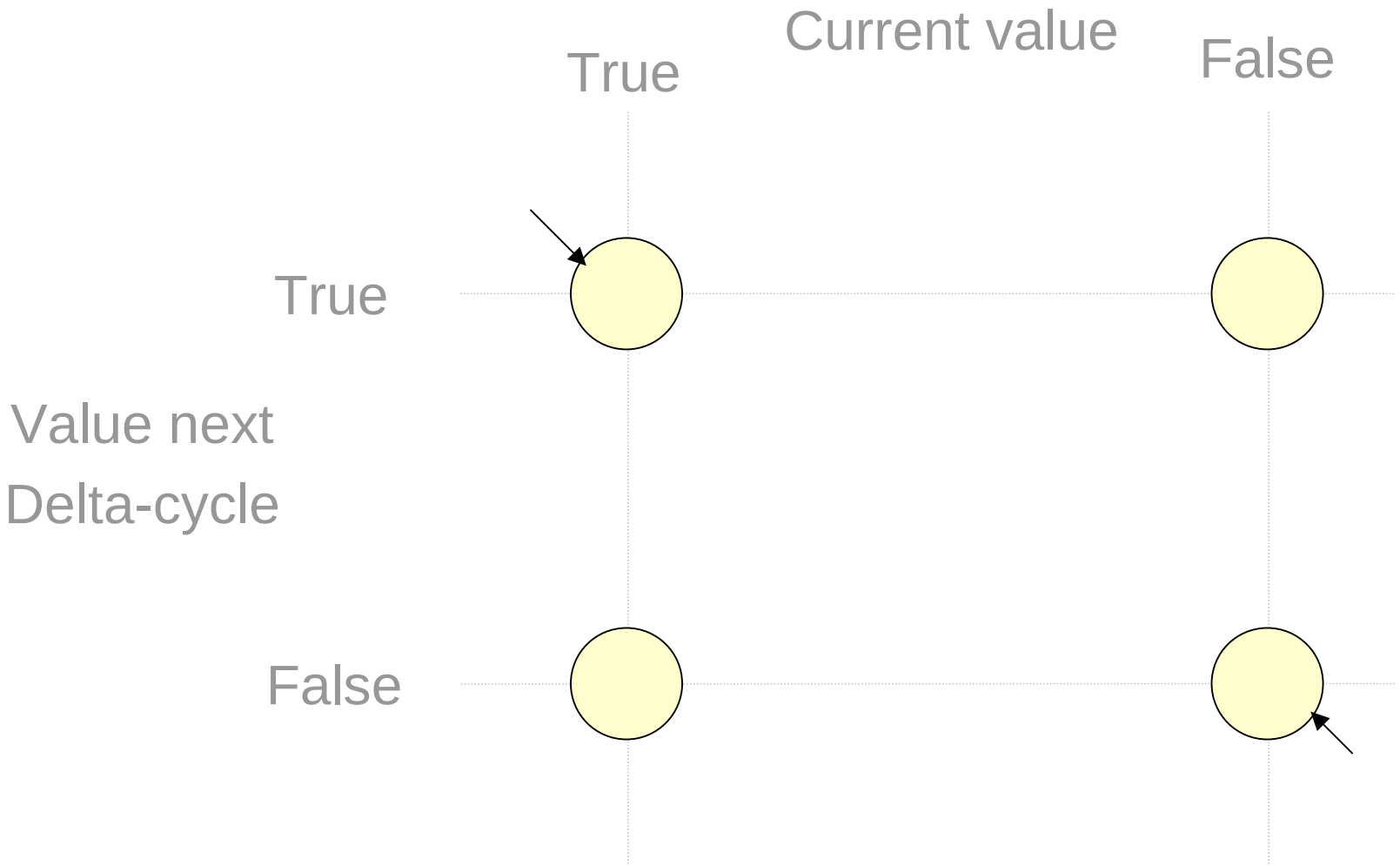
Pattern for a signal



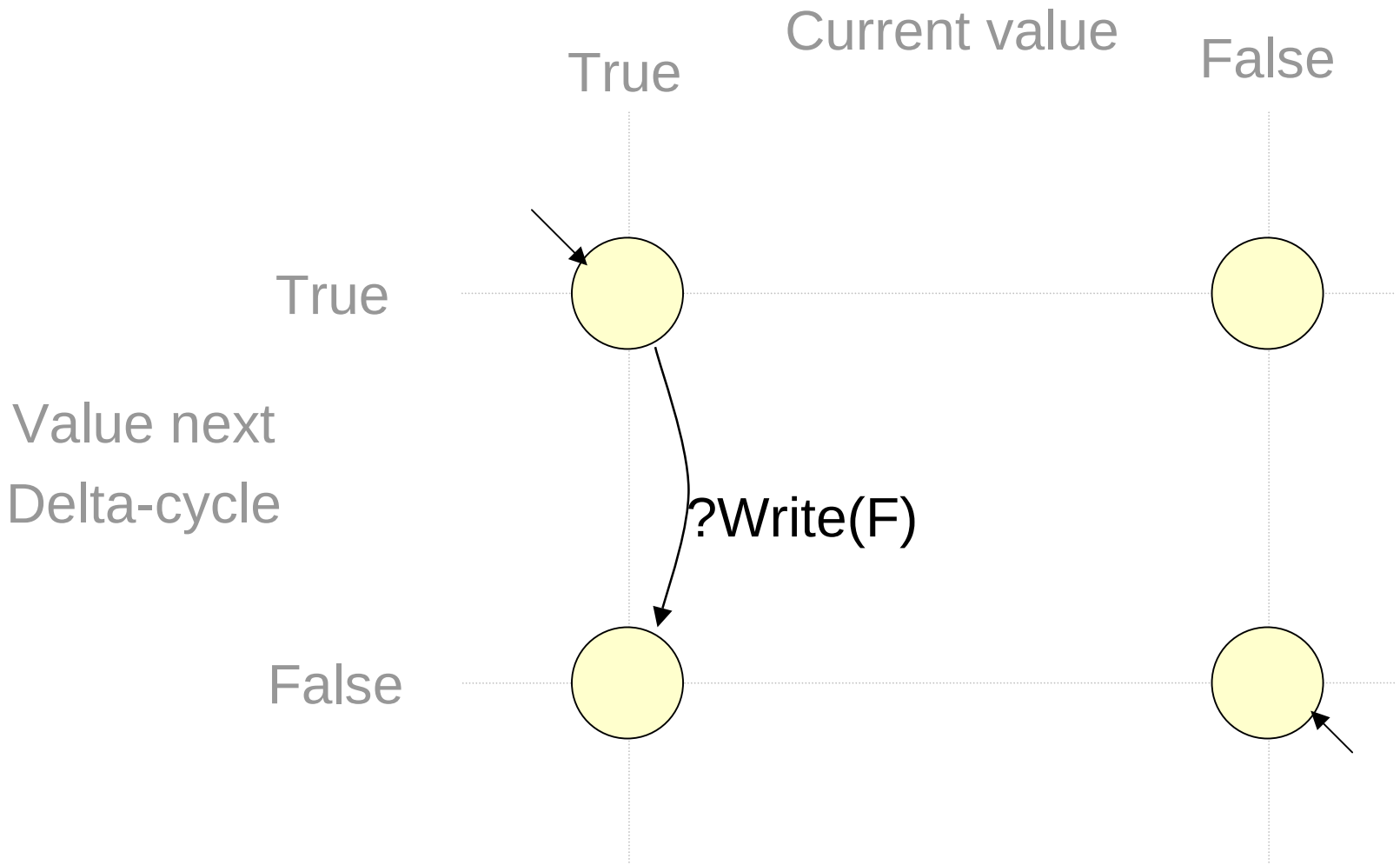
Pattern for a signal



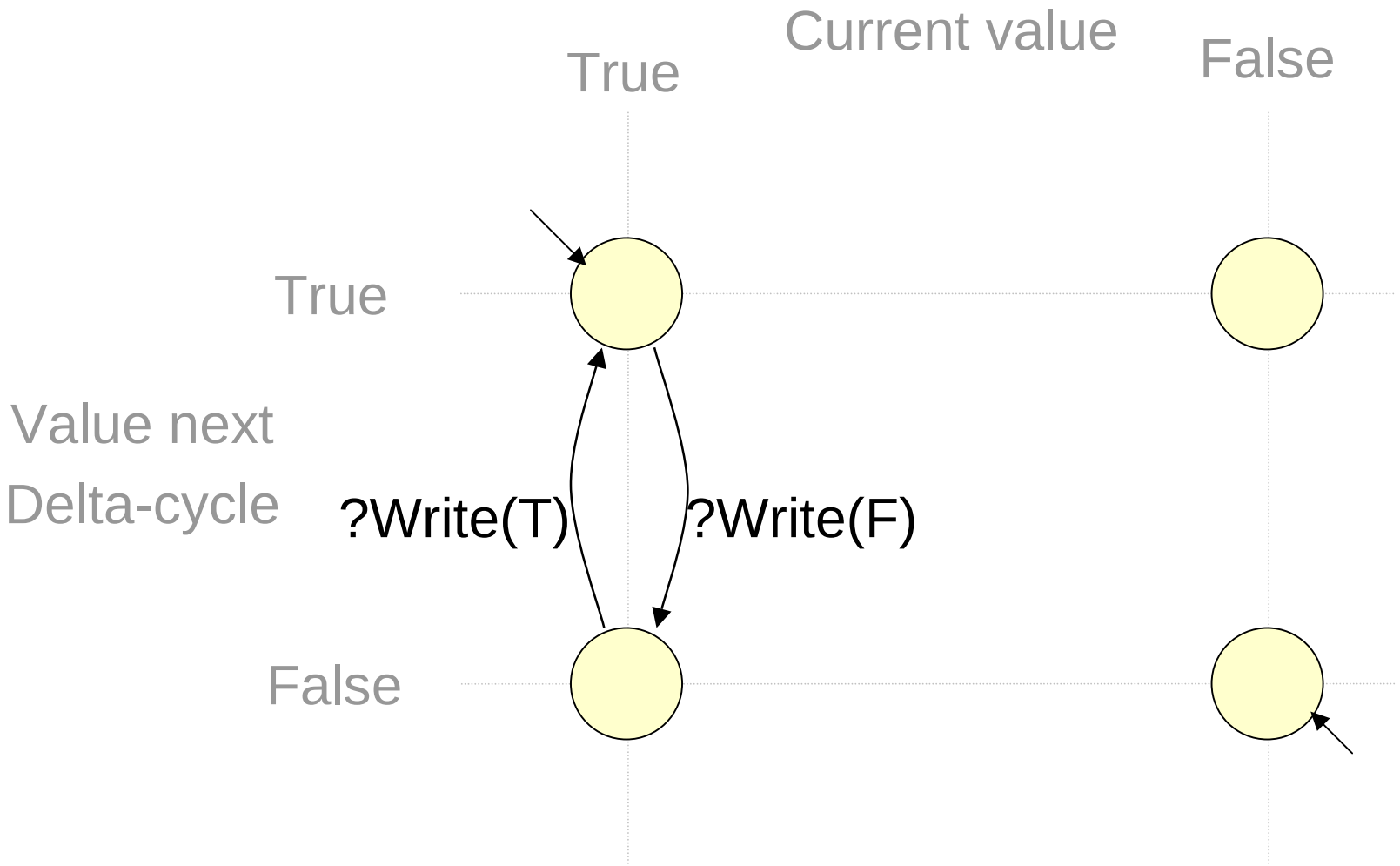
Pattern for a signal



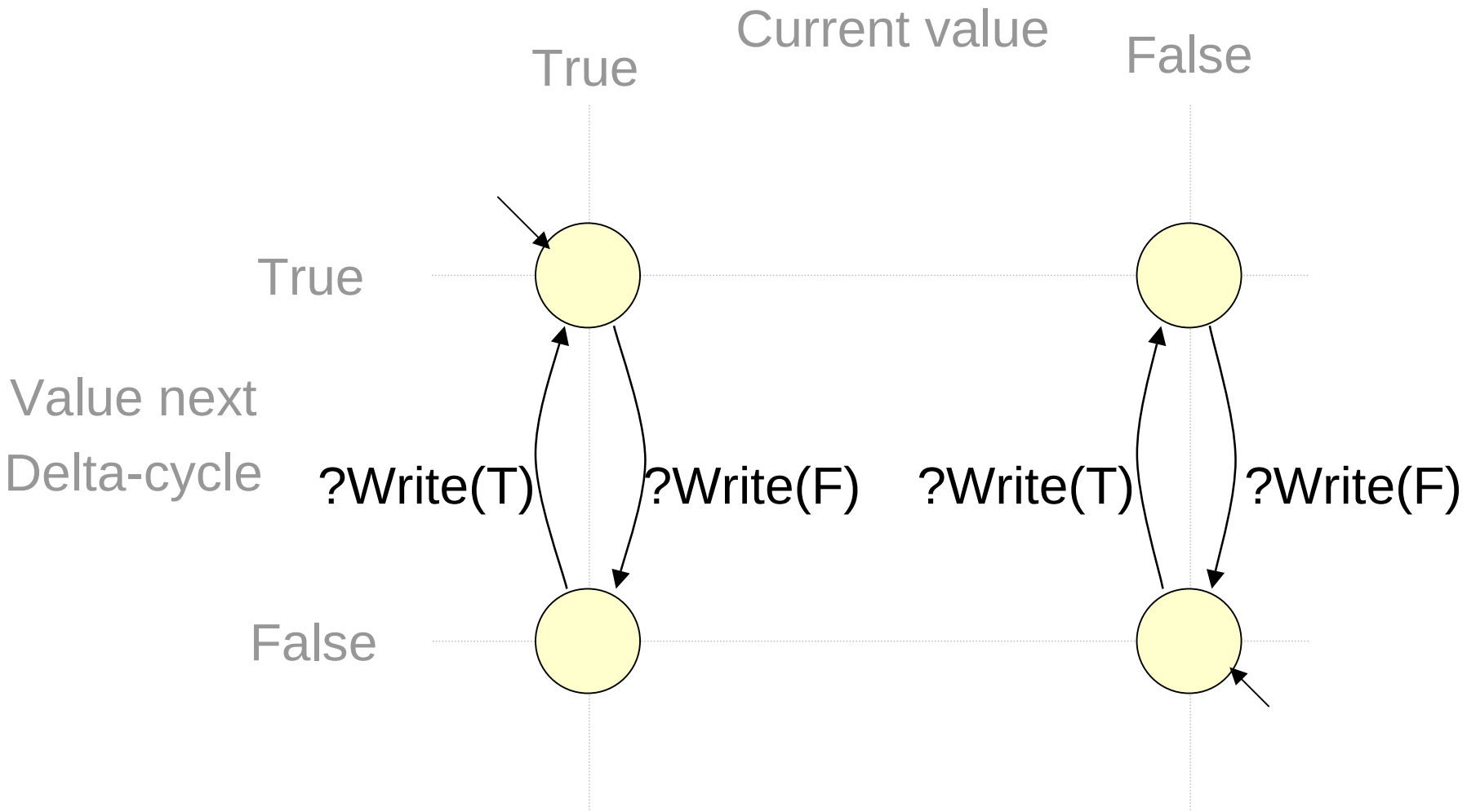
Pattern for a signal



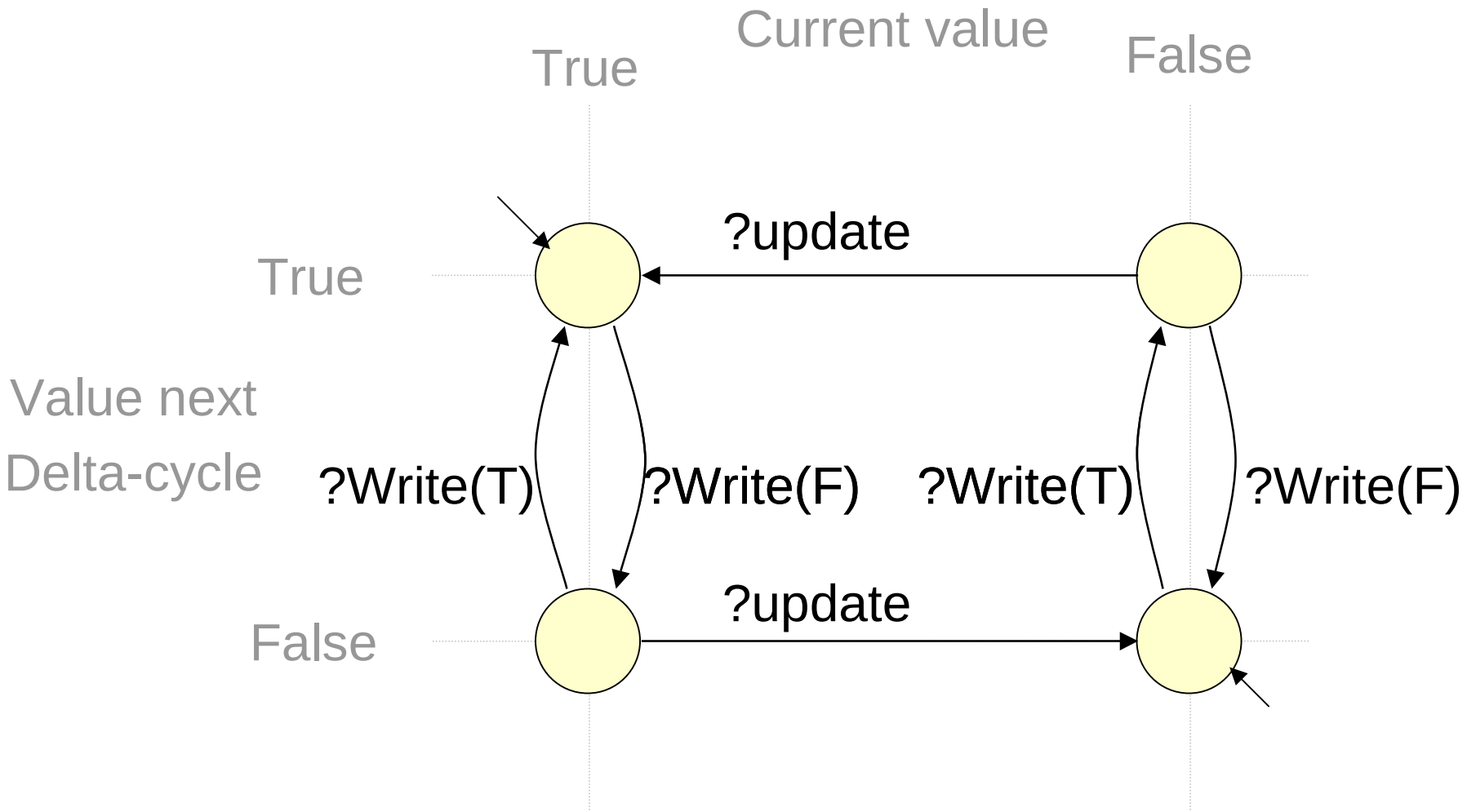
Pattern for a signal



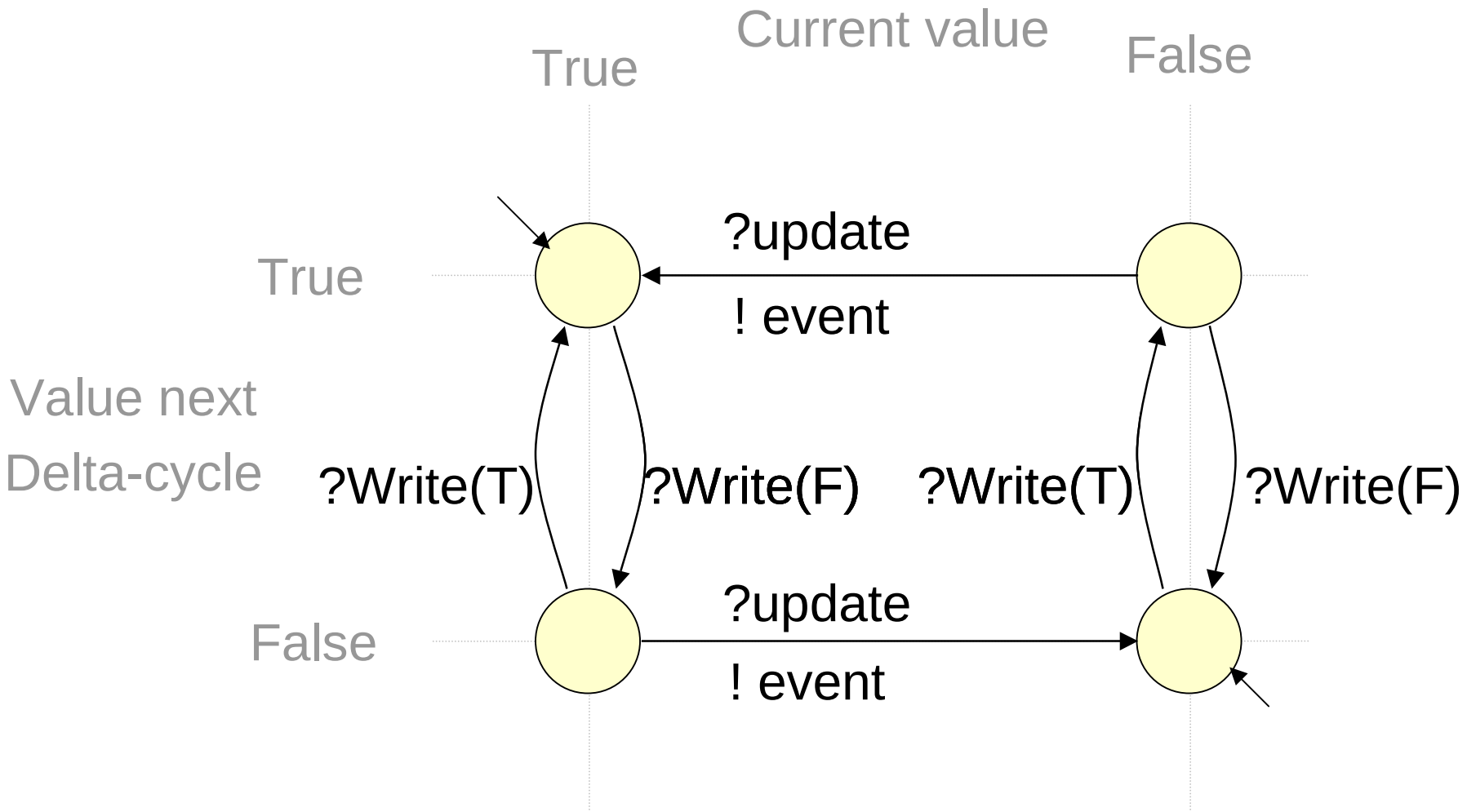
Pattern for a signal



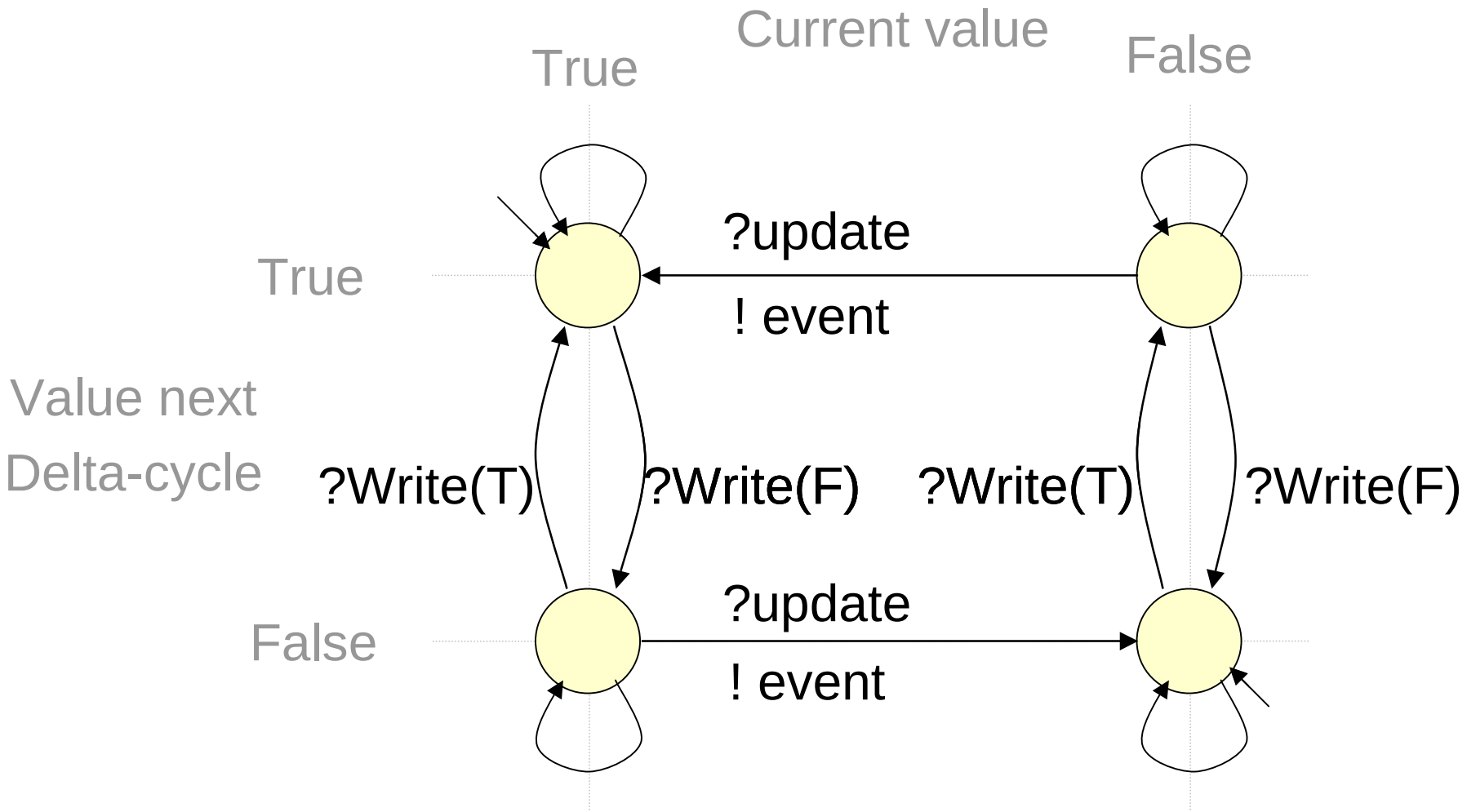
Pattern for a signal



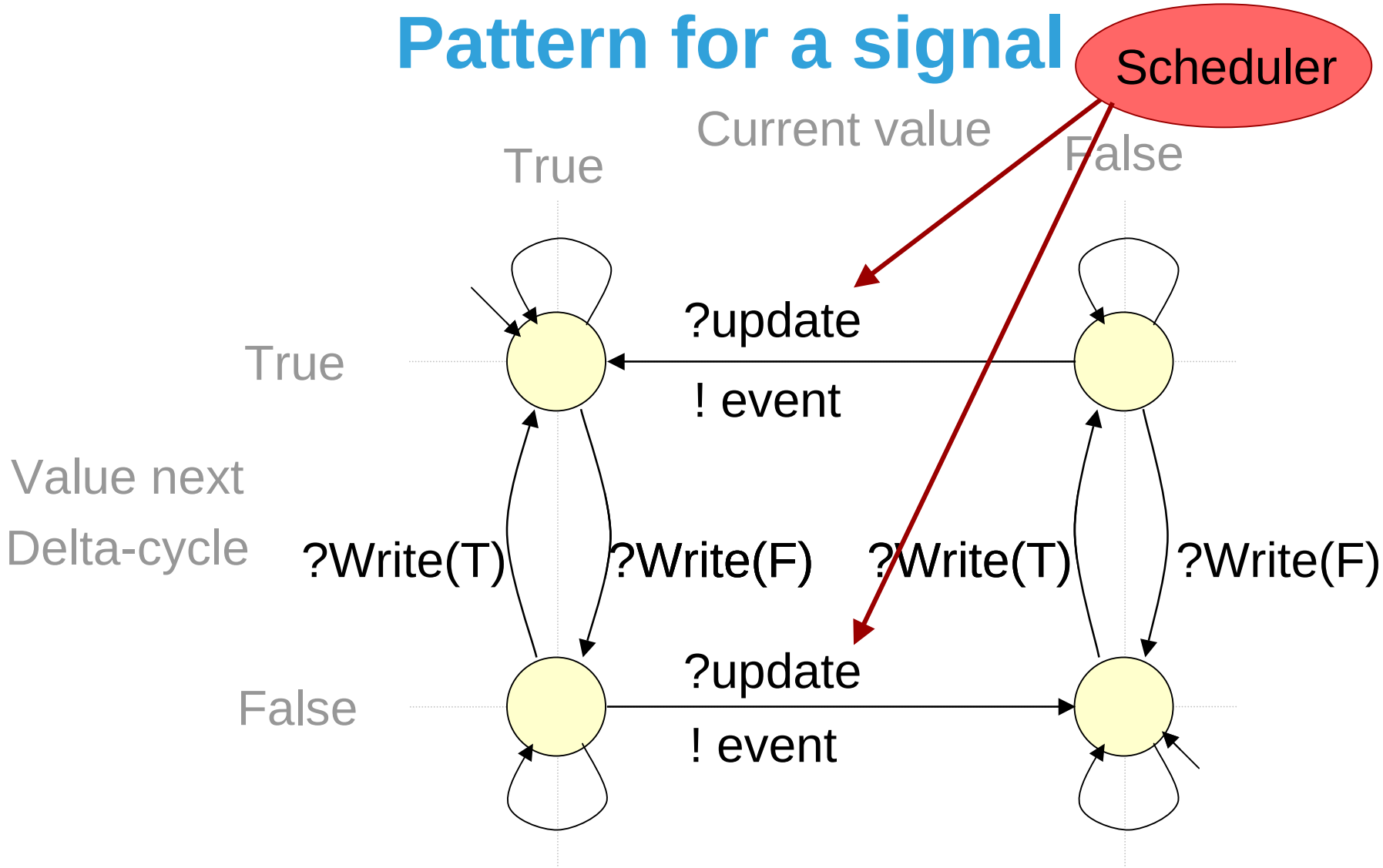
Pattern for a signal



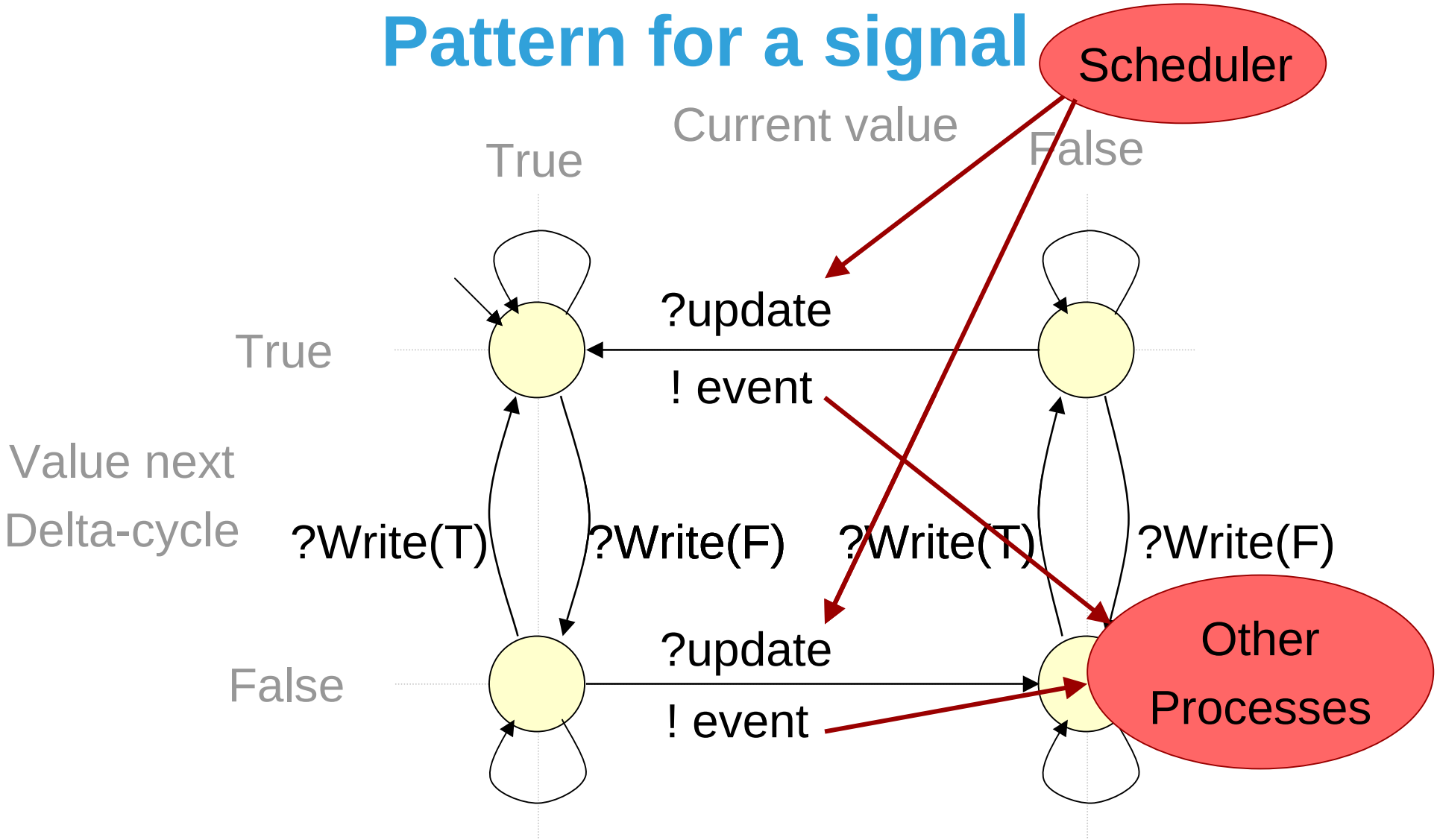
Pattern for a signal



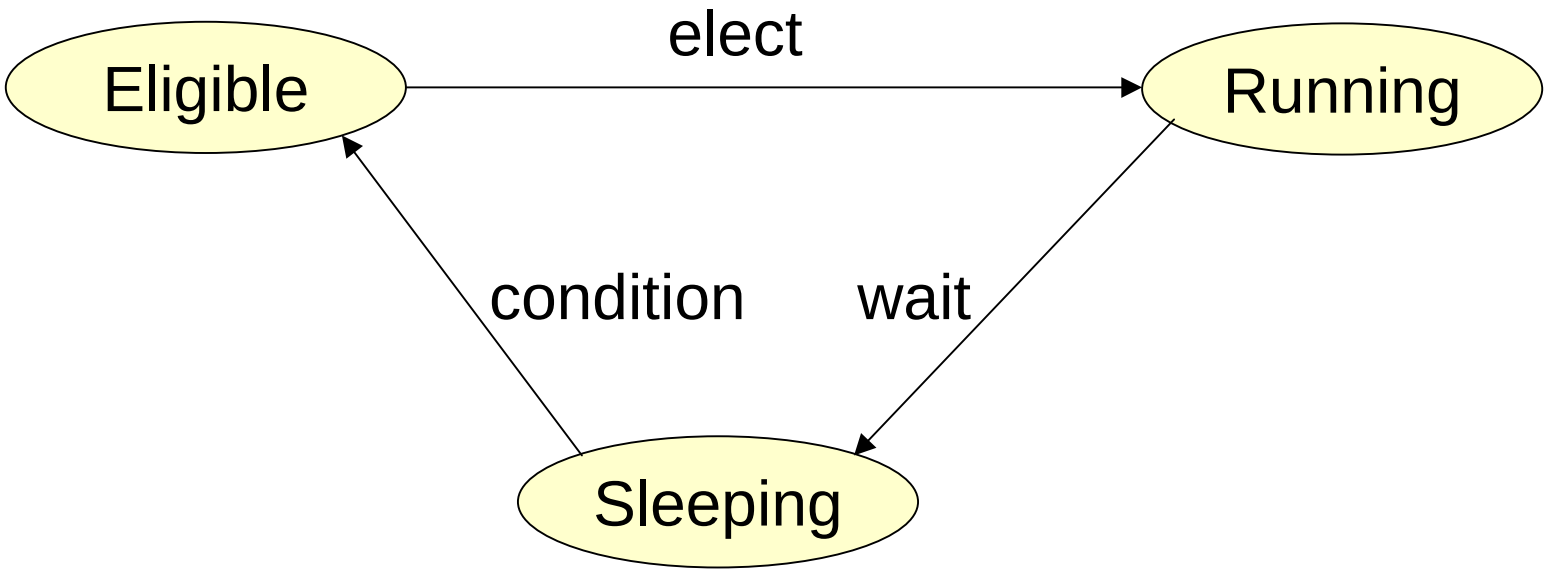
Pattern for a signal



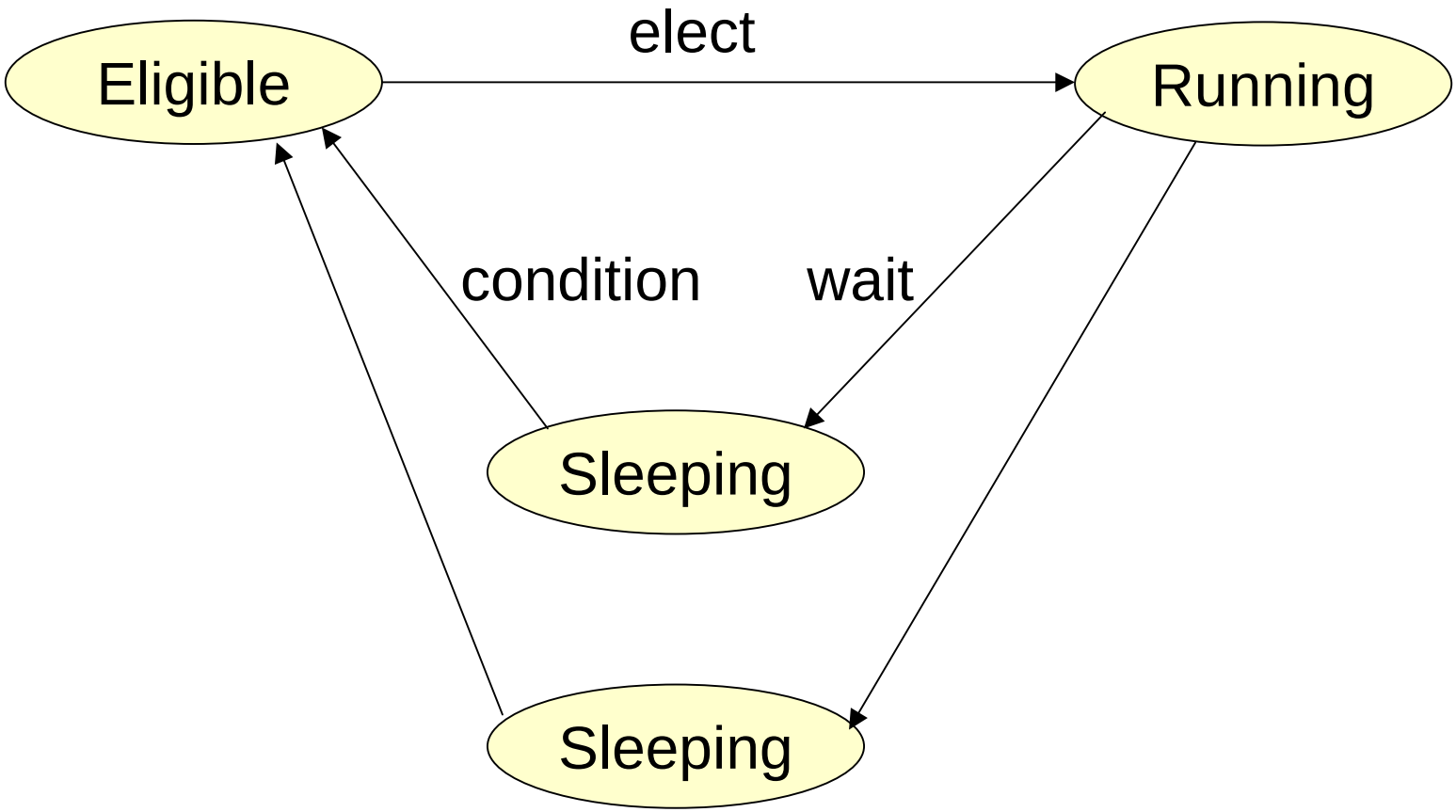
Pattern for a signal



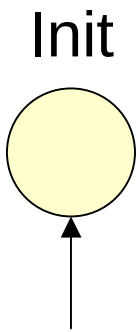
State of a process



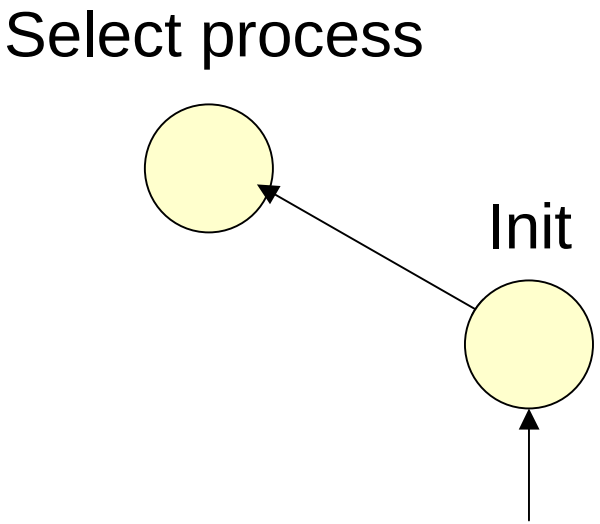
State of a process



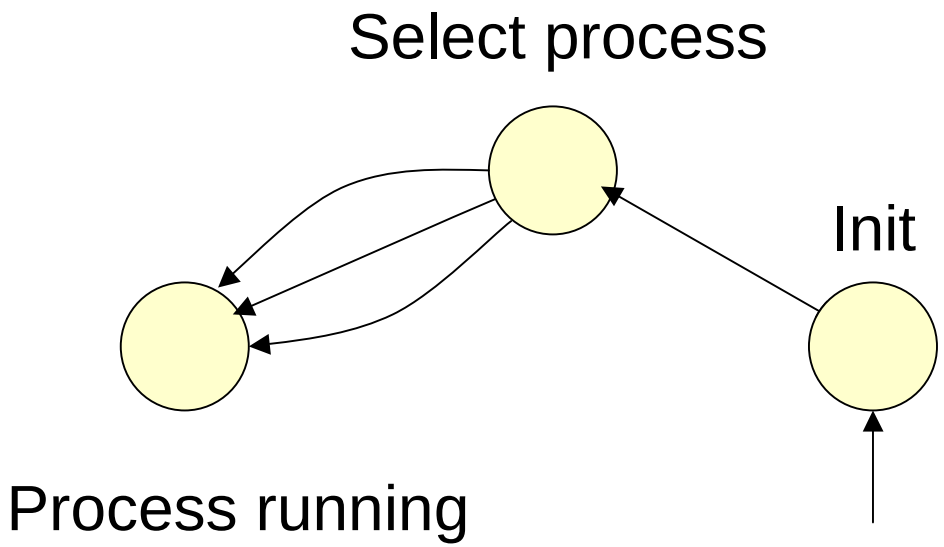
The scheduler



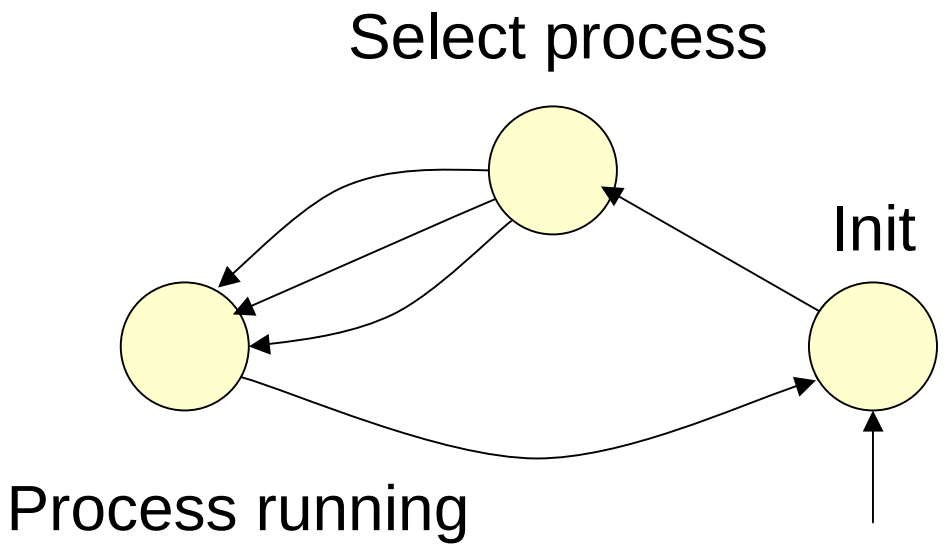
The scheduler



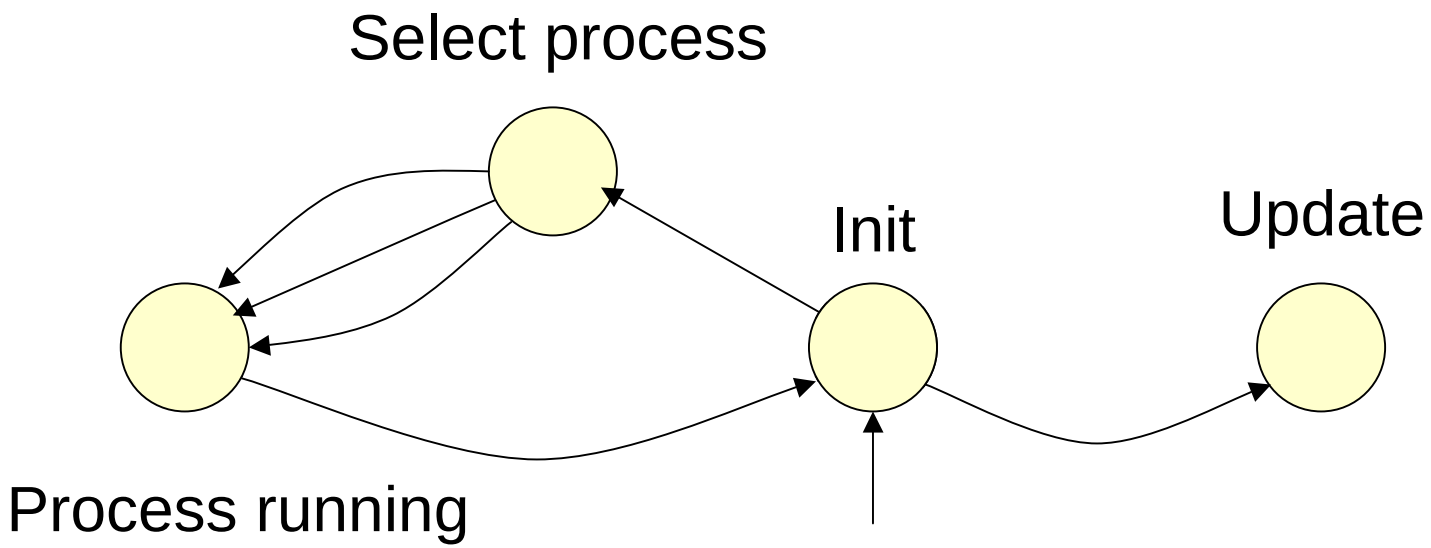
The scheduler



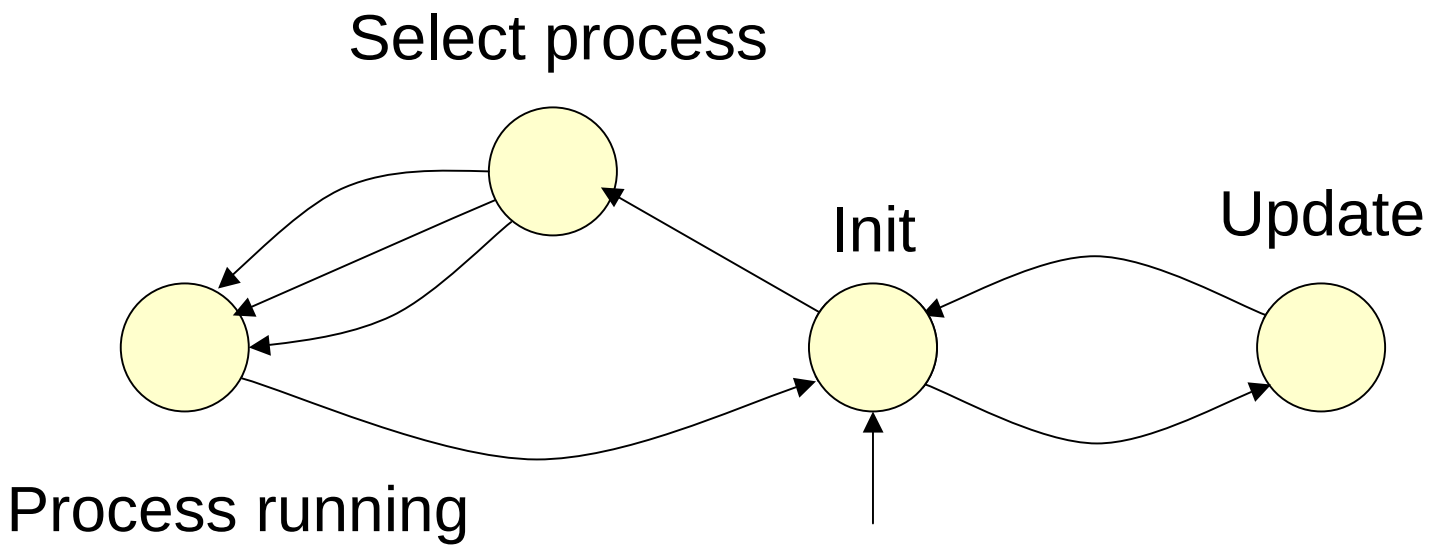
The scheduler



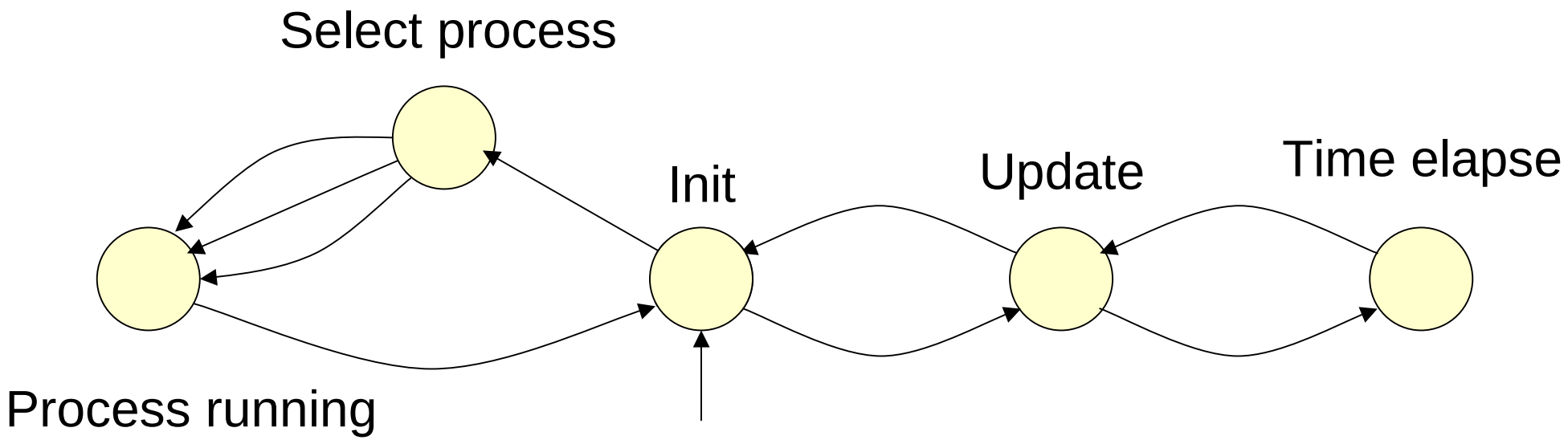
The scheduler



The scheduler

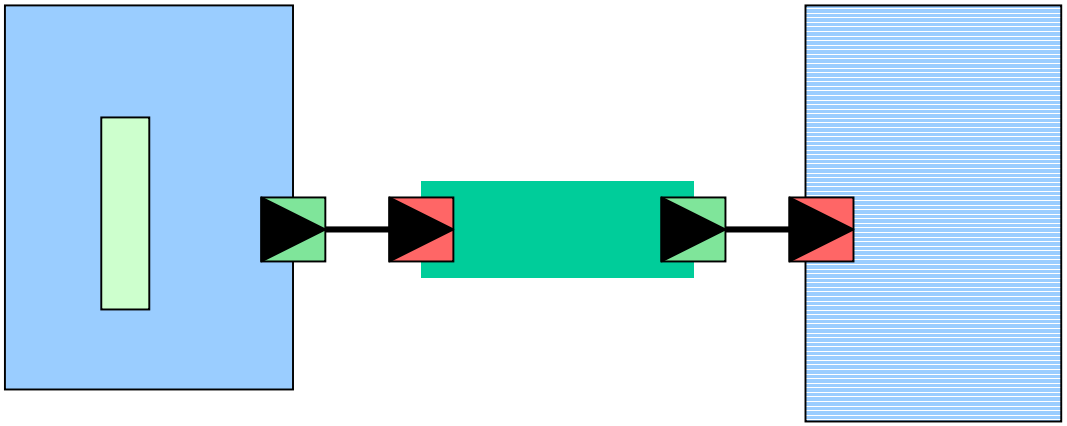


The scheduler

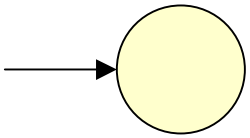


The basic protocol

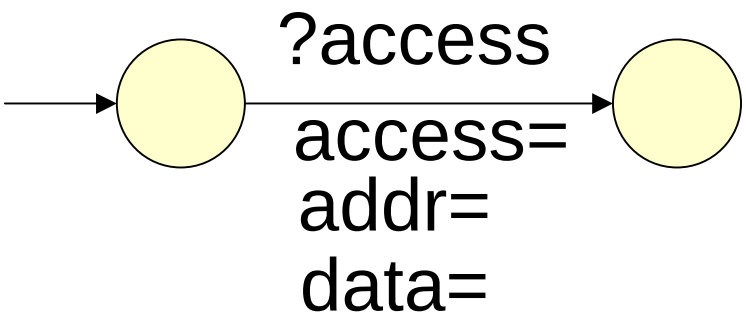
- Ports
  - Initiator
  - Target
- Channels
  - Router
  - Arbiter
  - *(Sequentializer)*



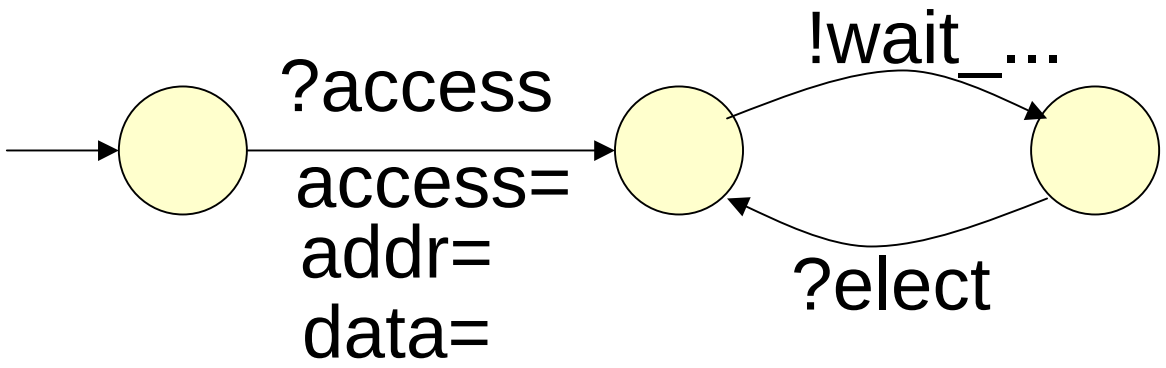
Basic initiator port



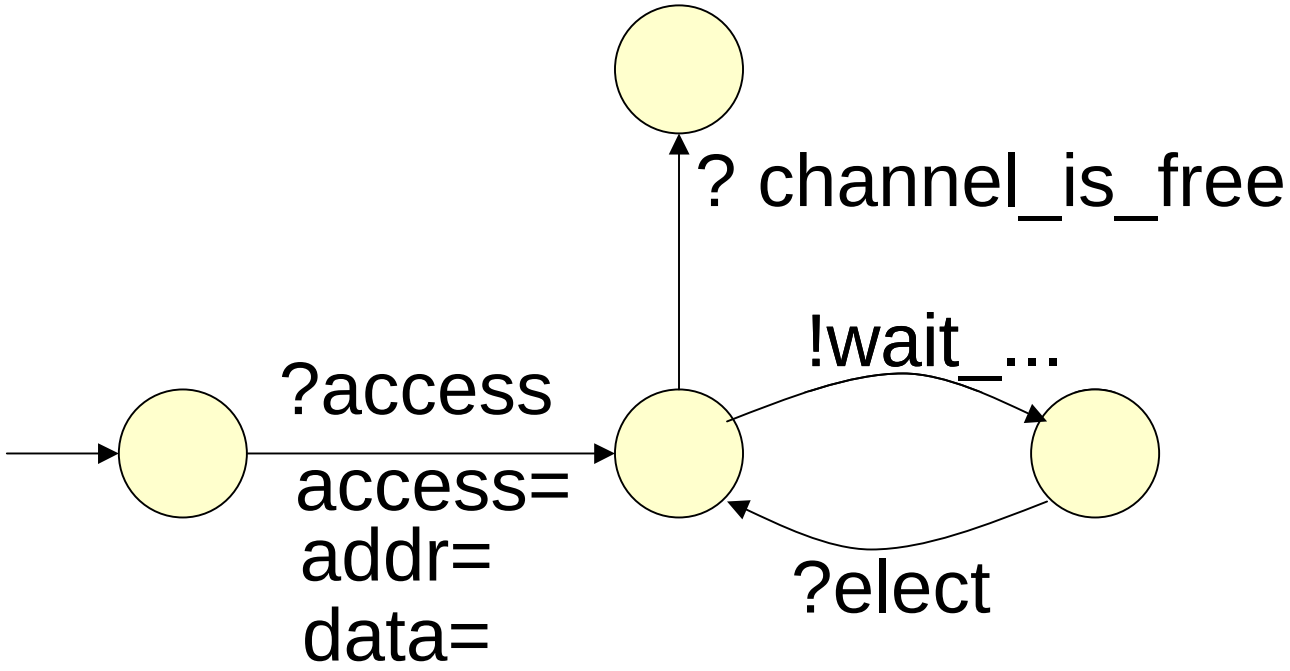
Basic initiator port



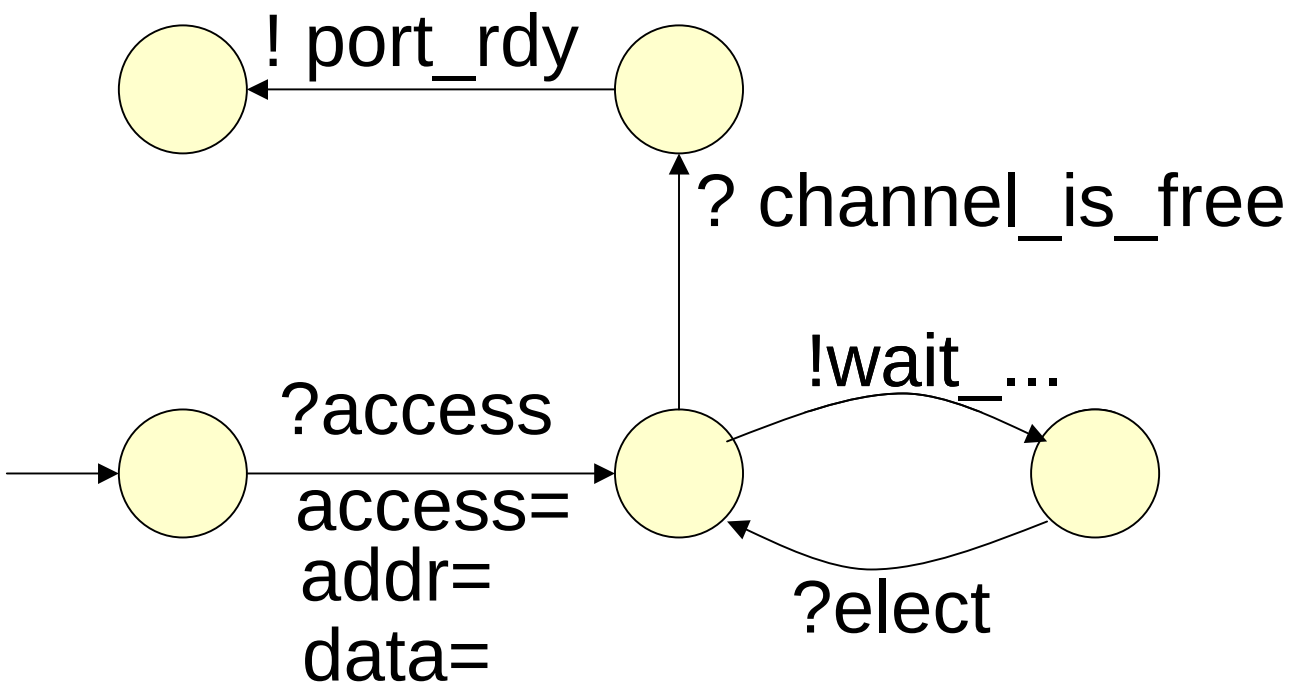
Basic initiator port



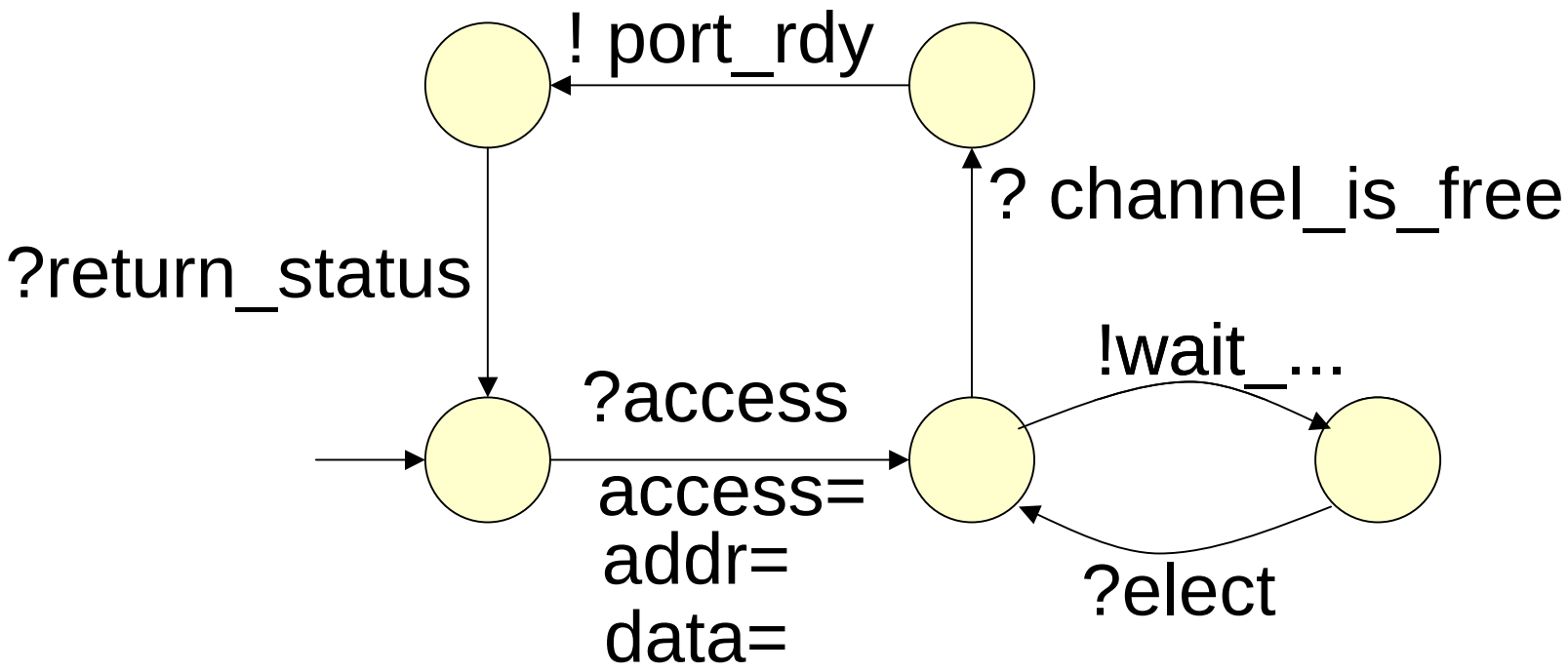
Basic initiator port



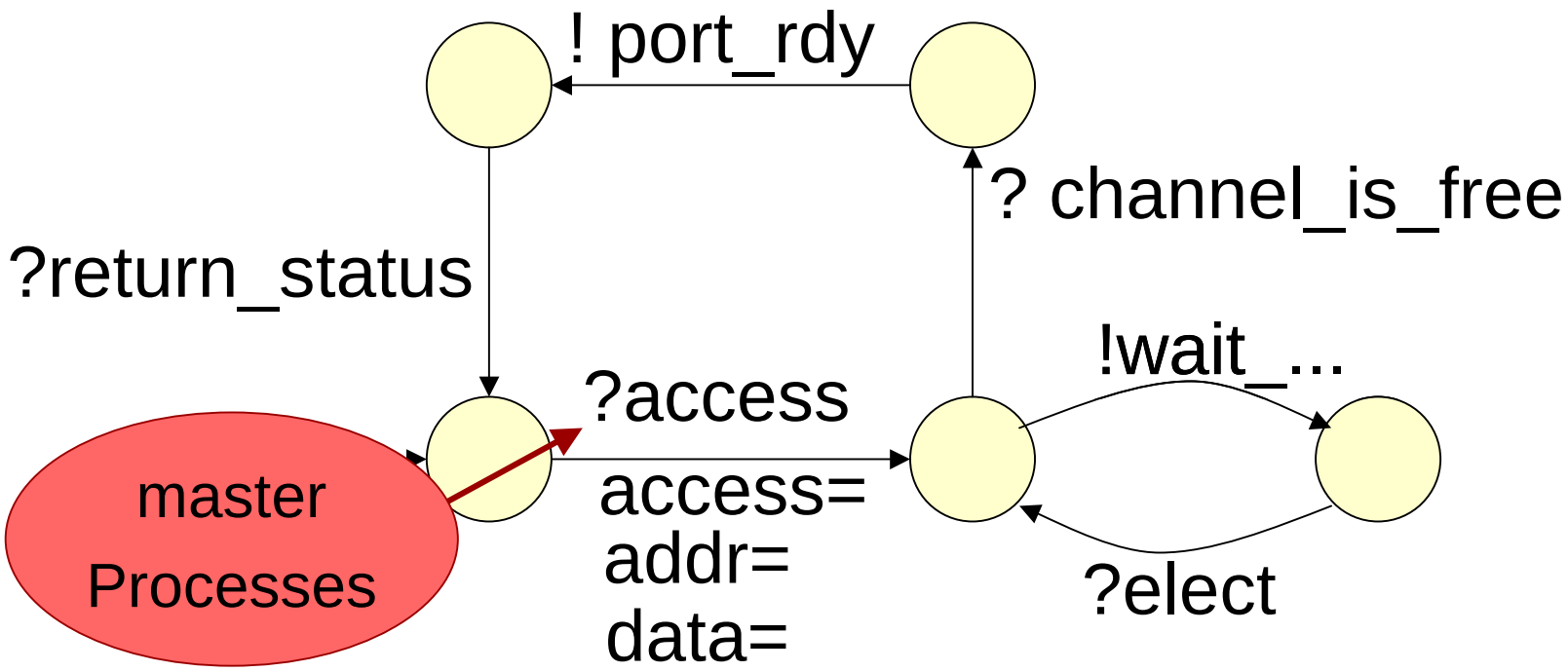
Basic initiator port



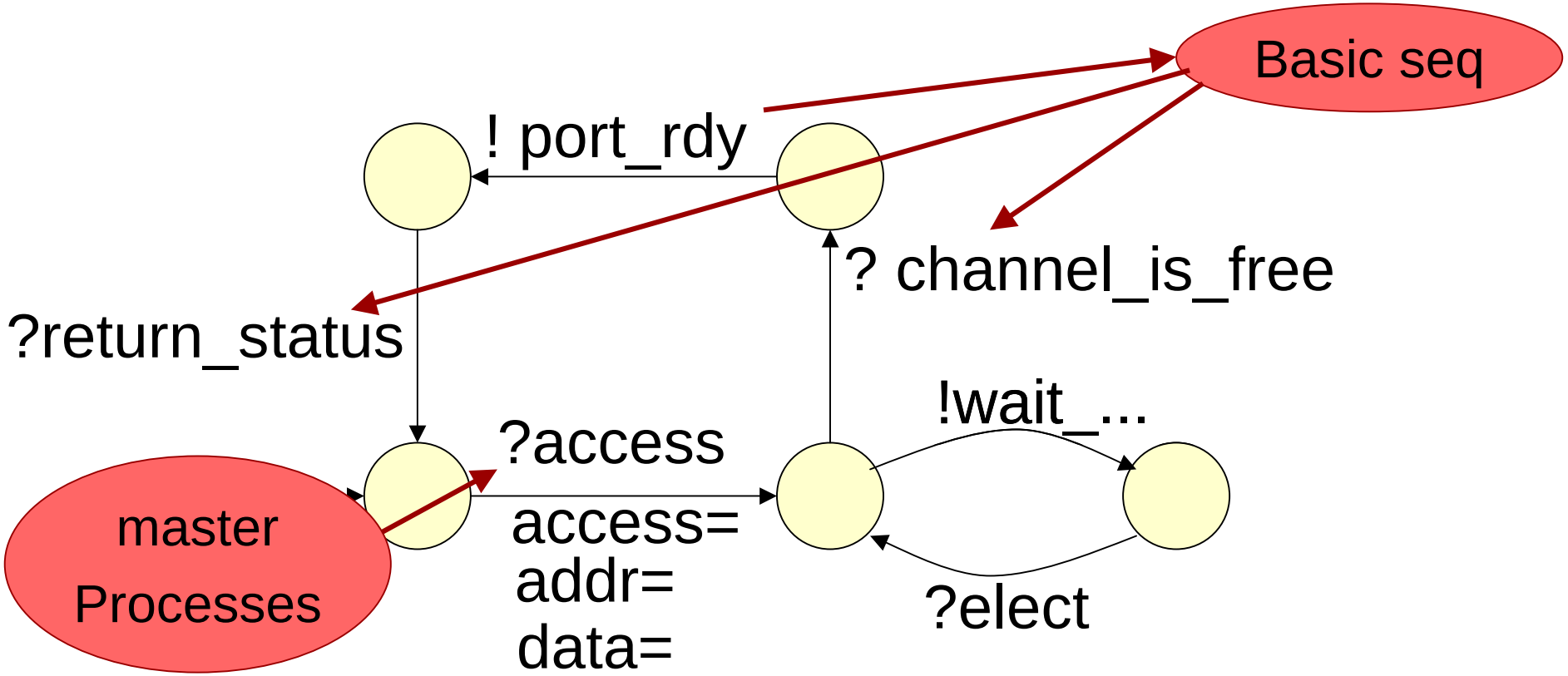
Basic initiator port



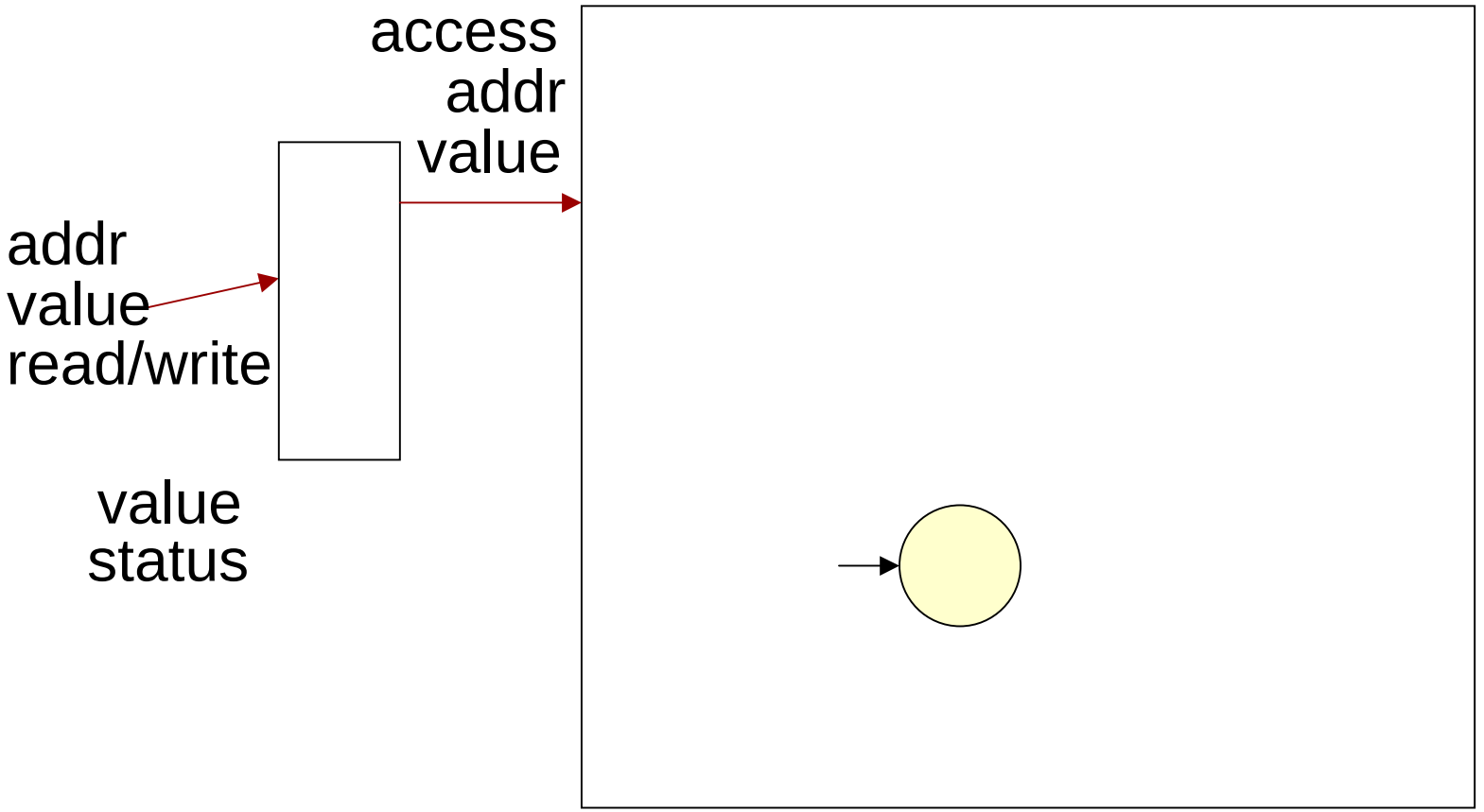
Basic initiator port



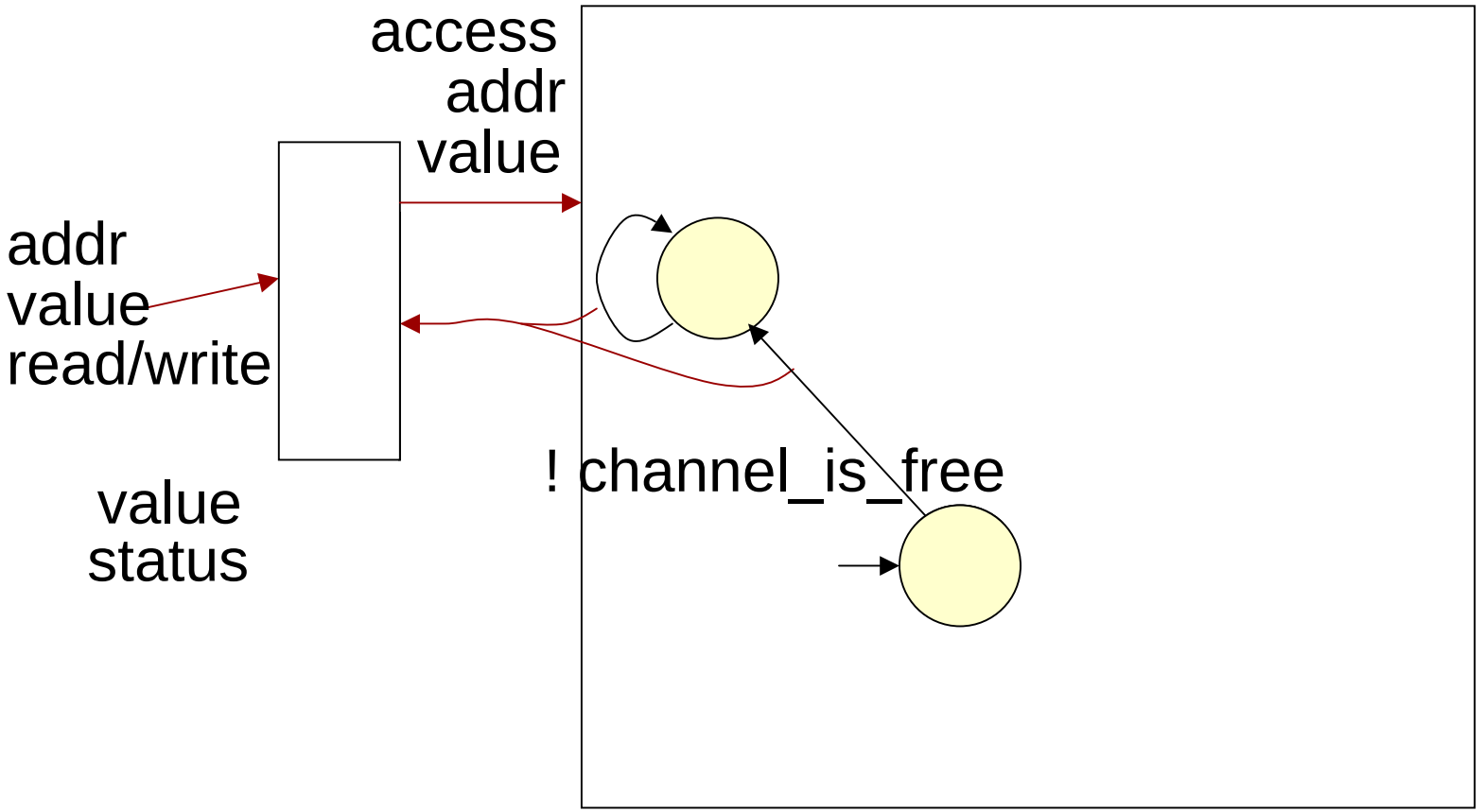
Basic initiator port



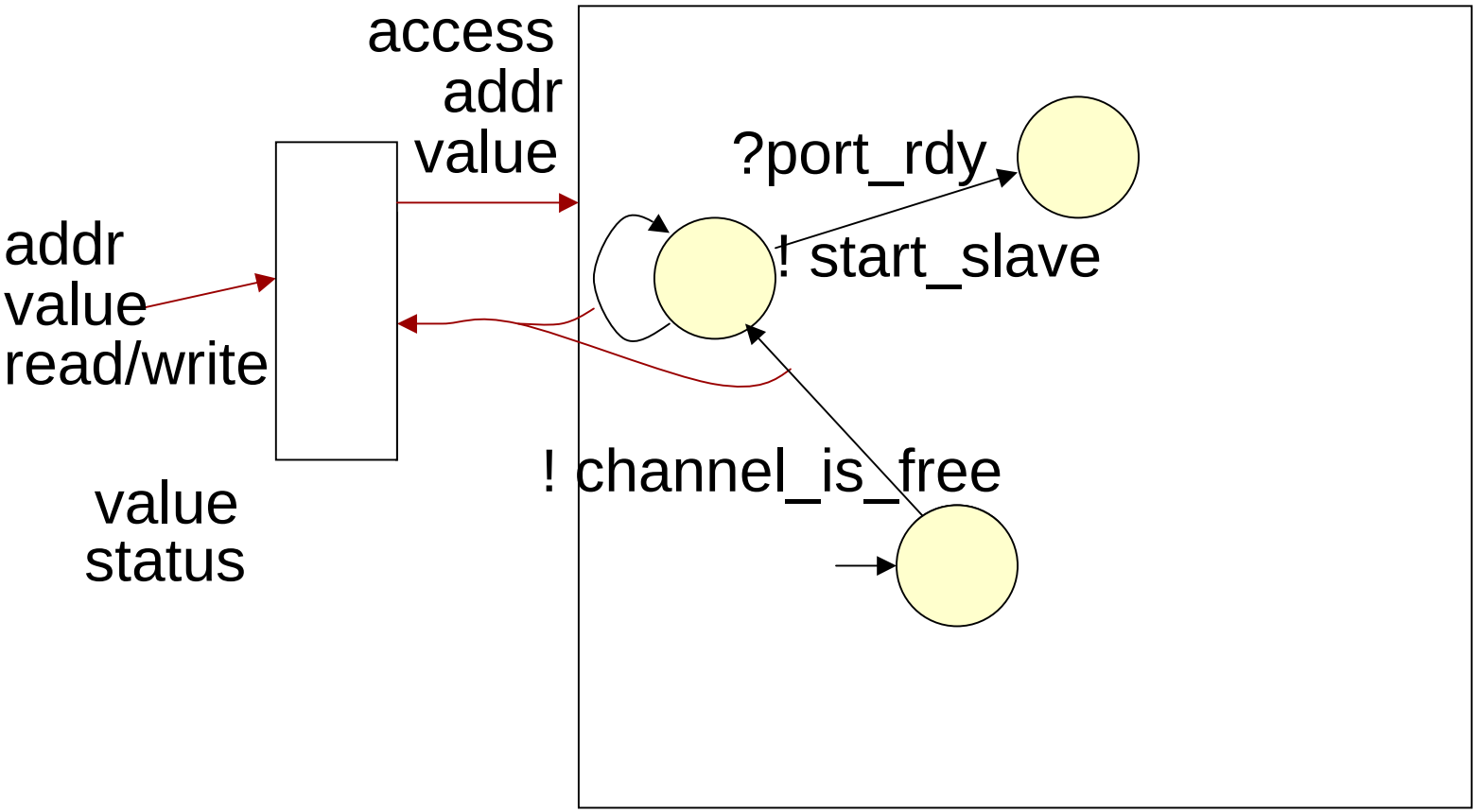
The basic_seq



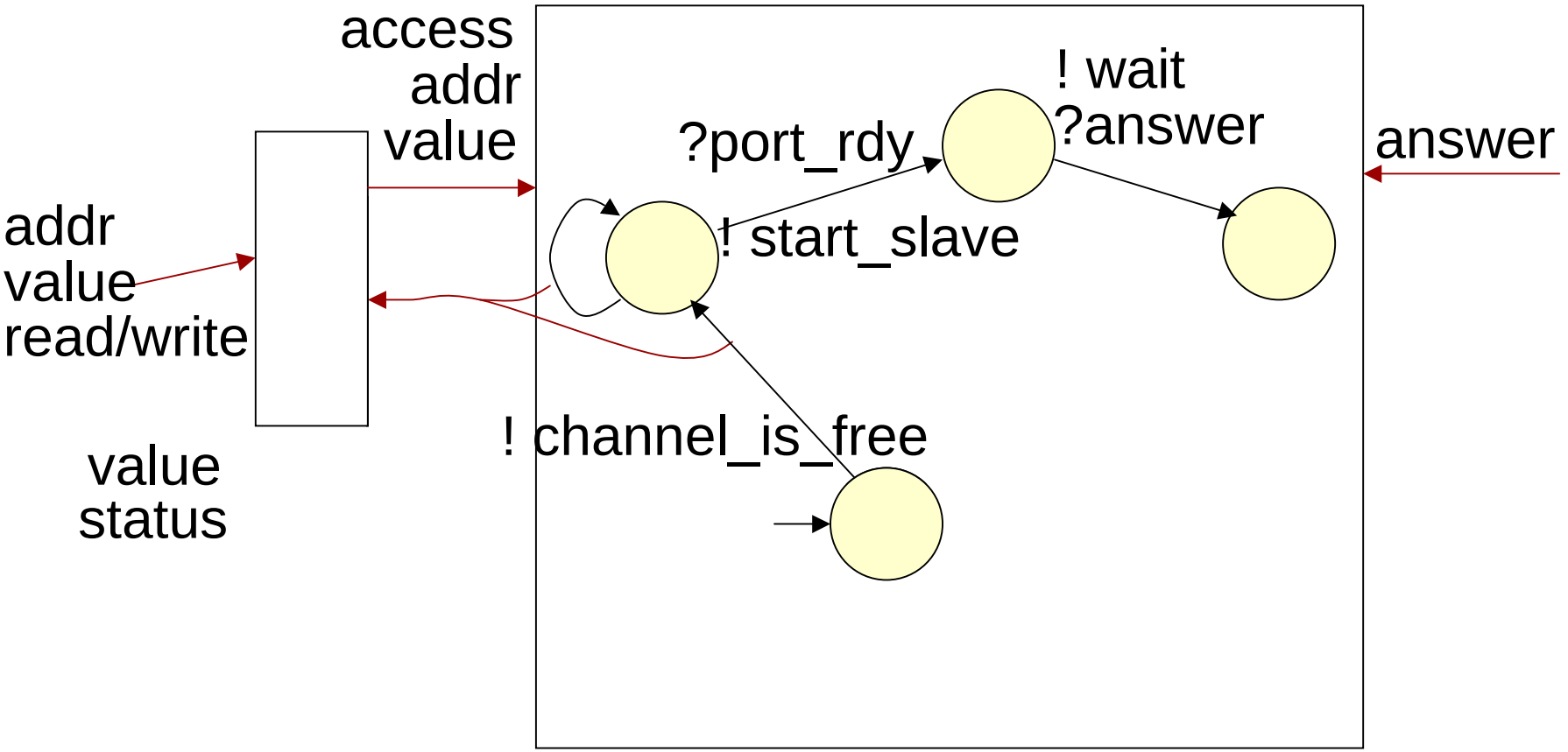
The basic_seq



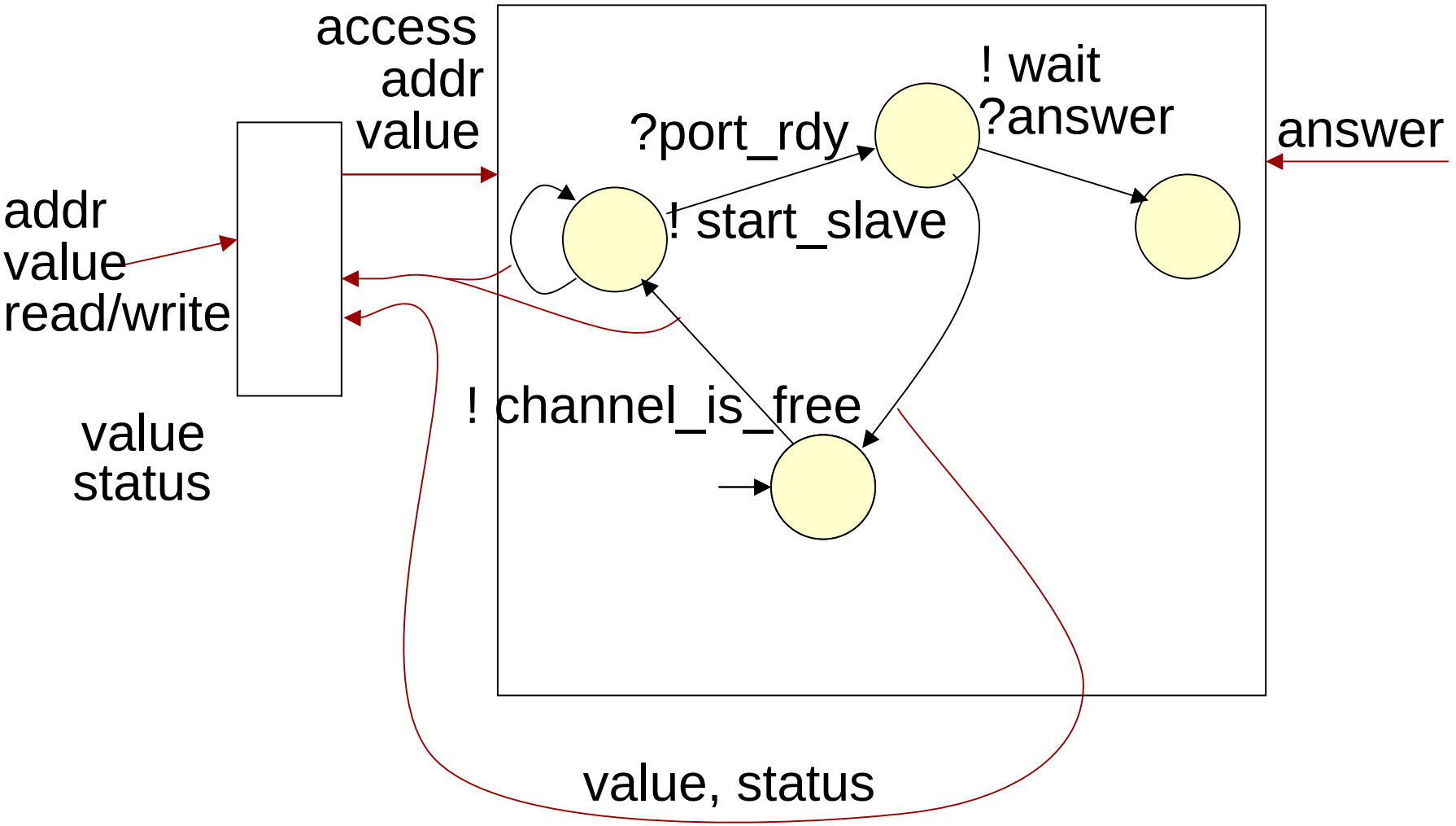
The basic_seq



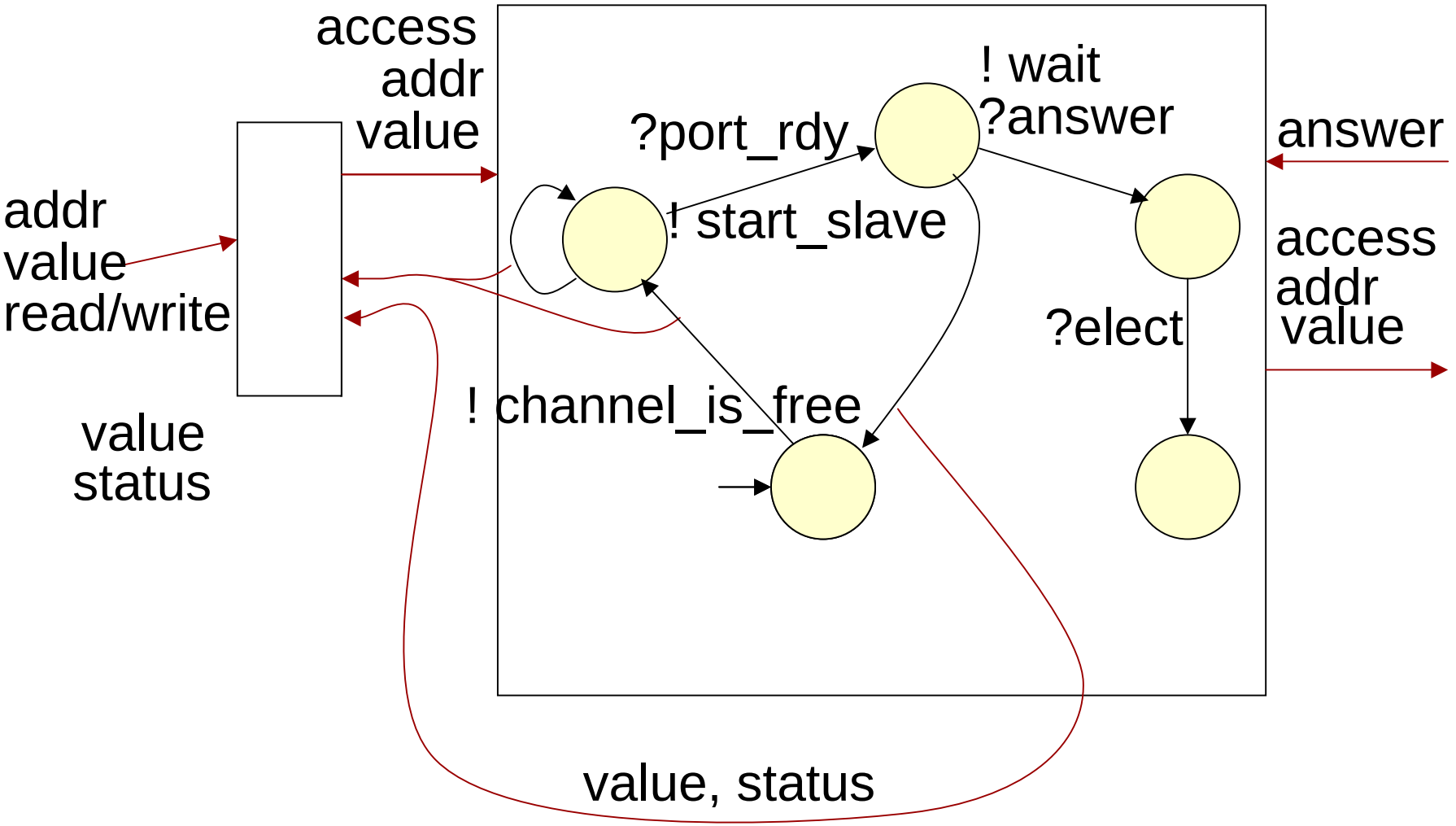
The basic_seq



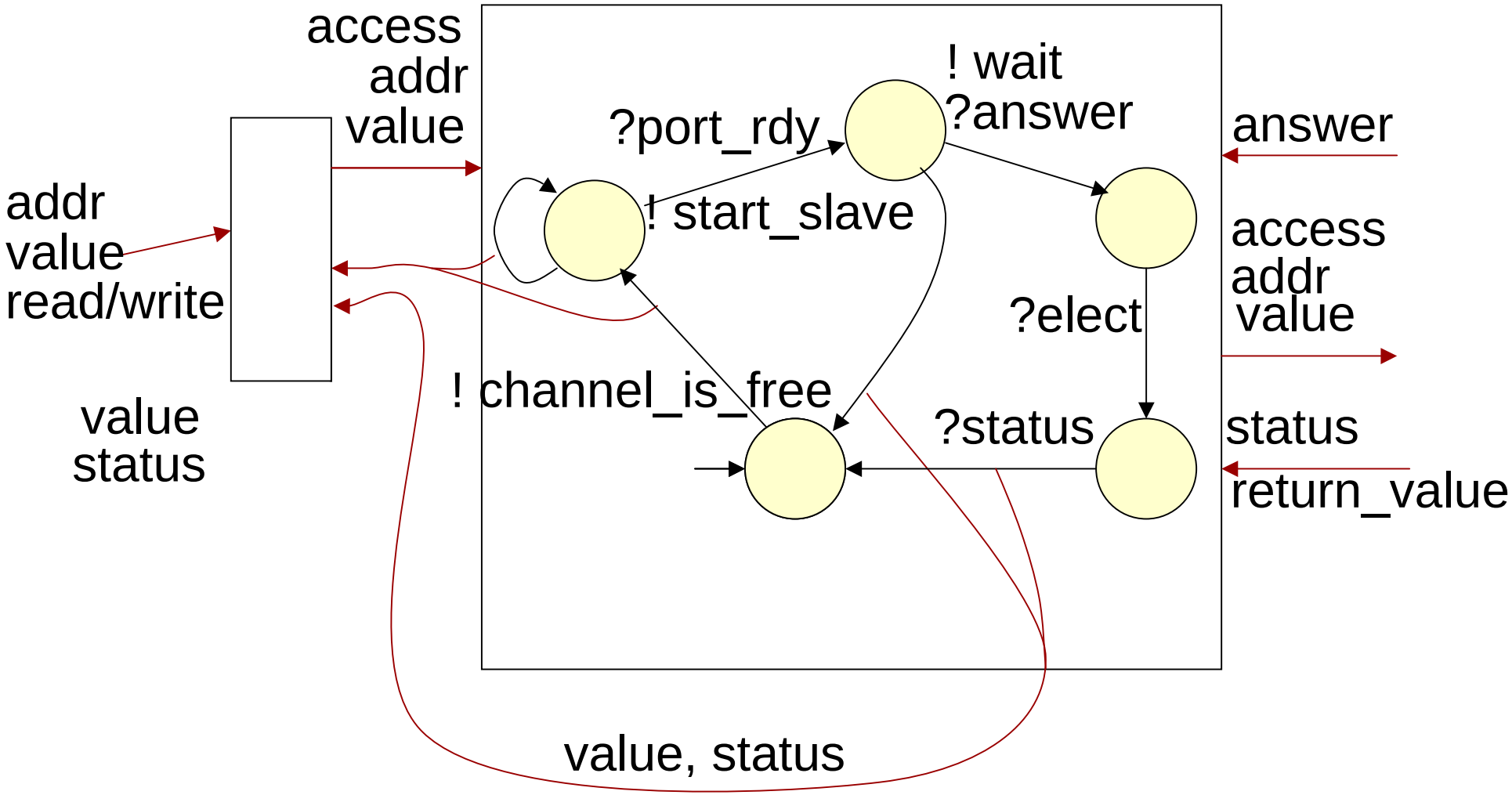
The basic_seq



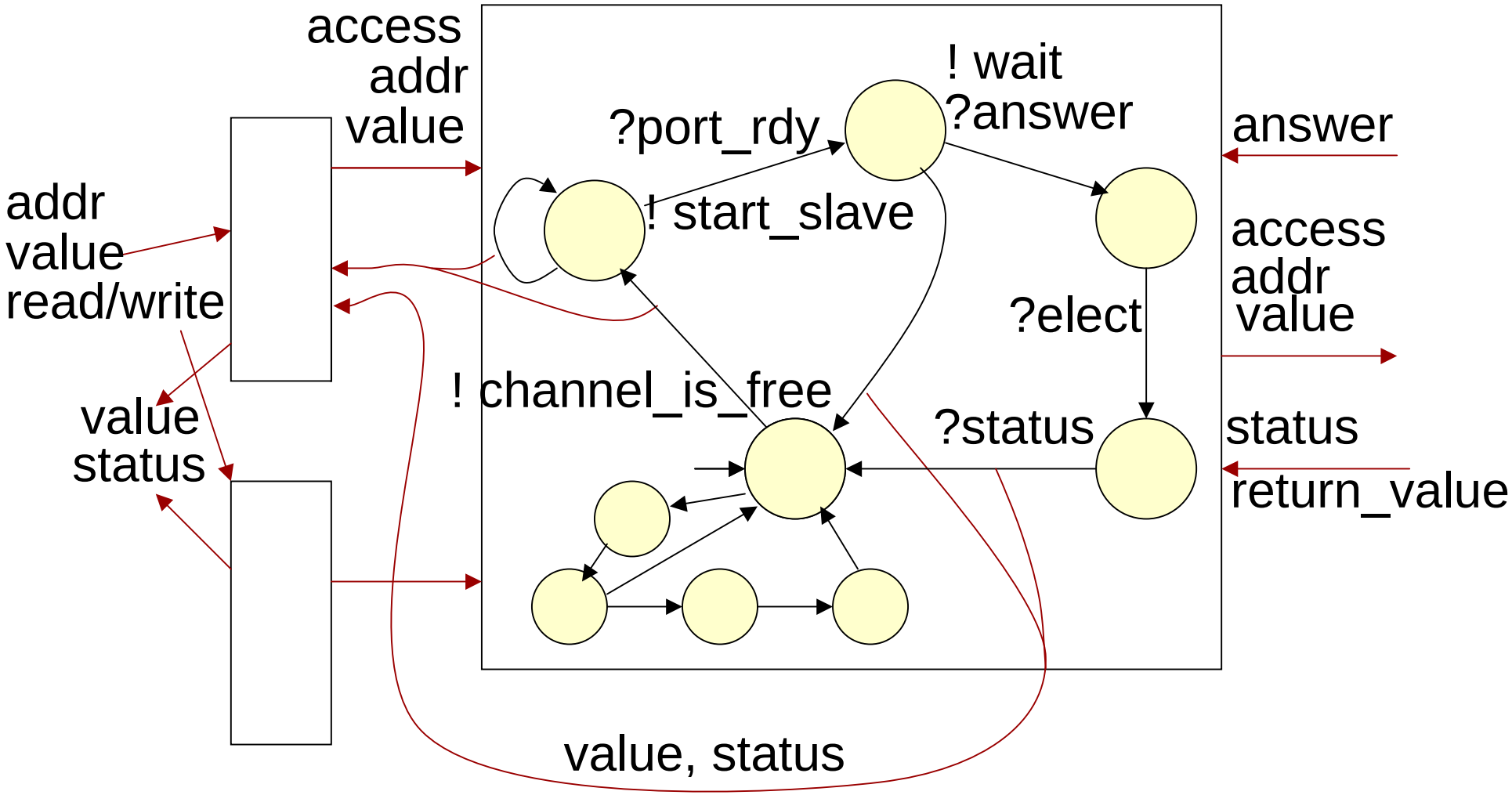
The basic_seq



The basic_seq



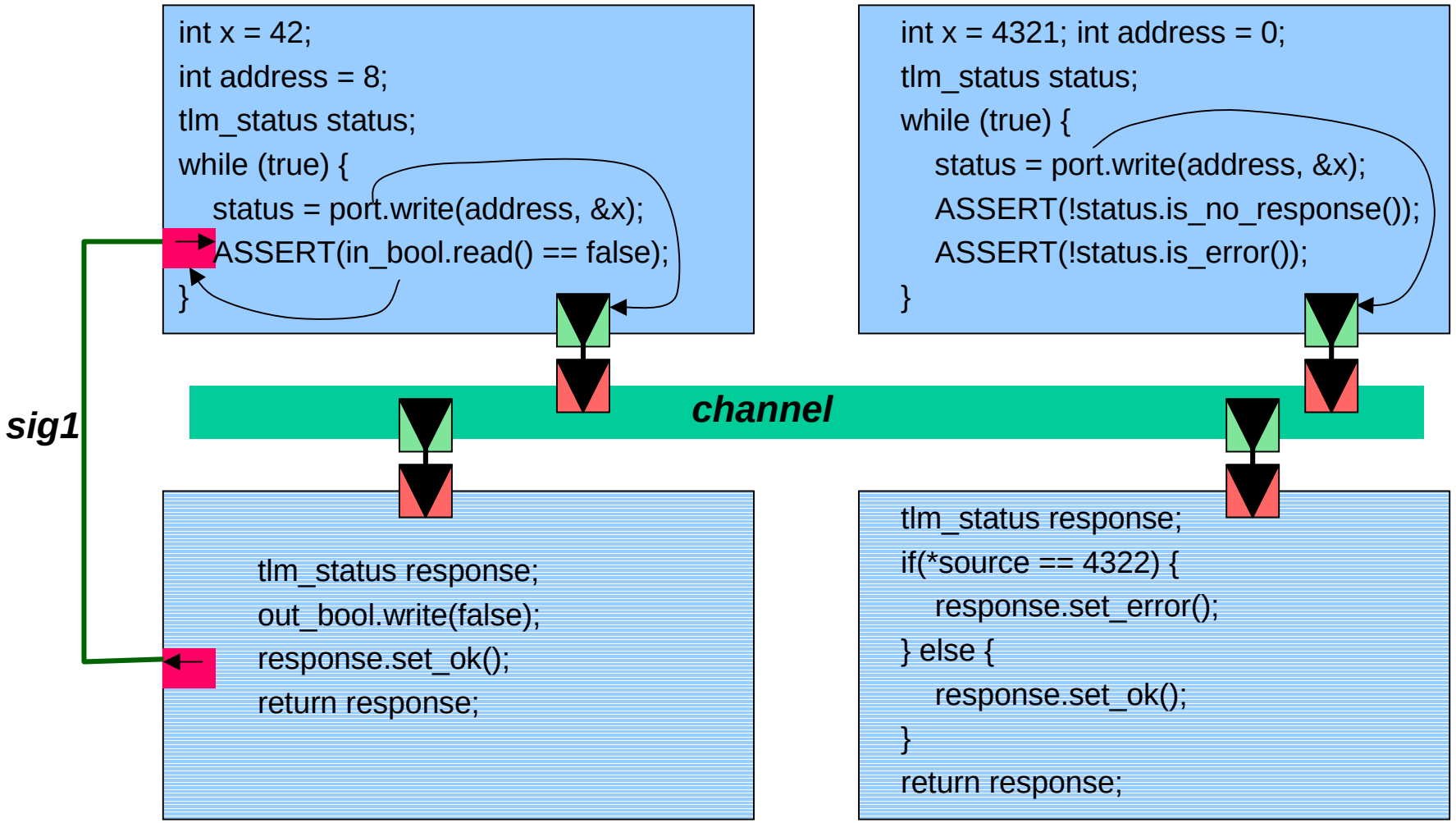
The basic_seq



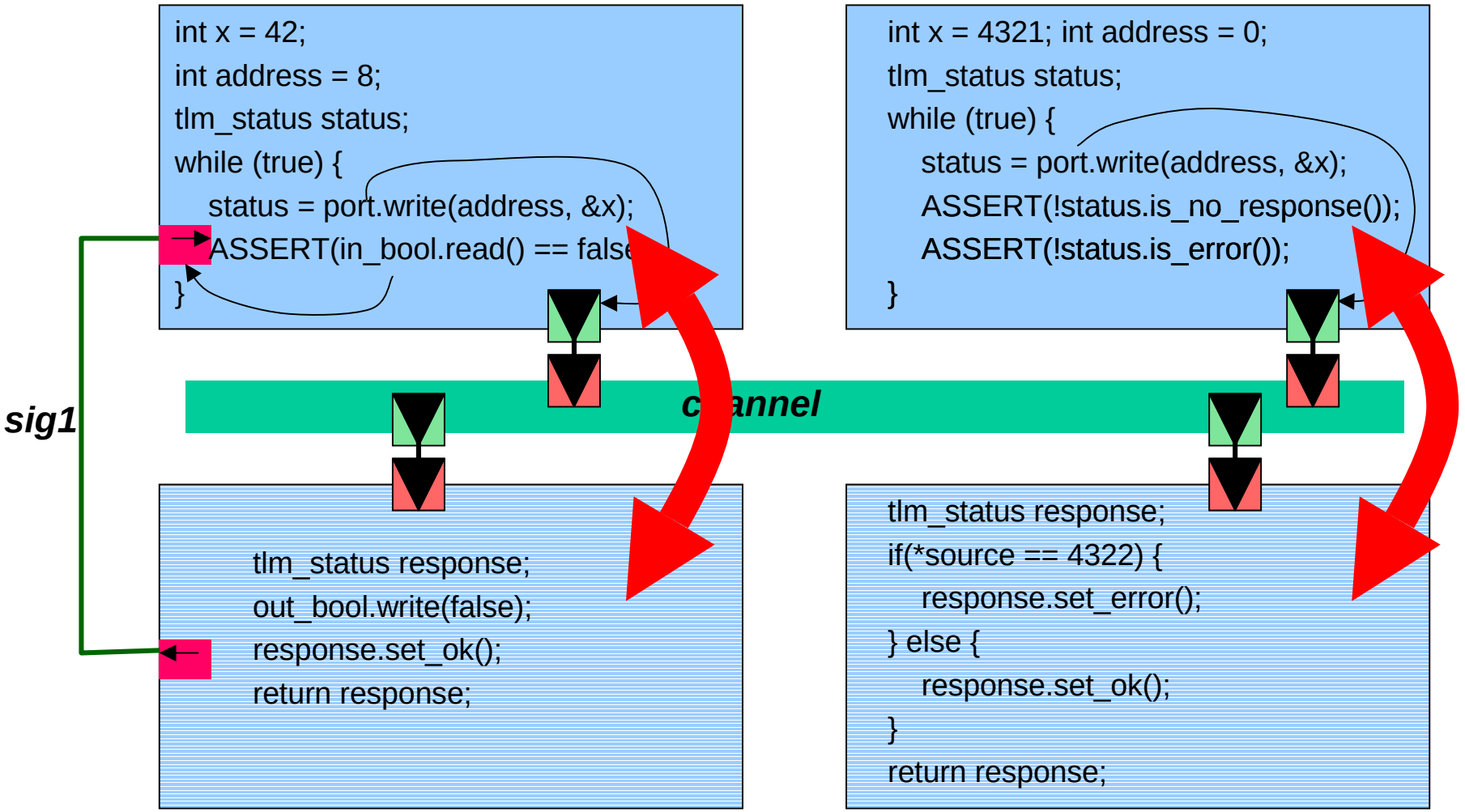
Outline

- Introduction about Systems on a Chip
- Verification approach and tool chain
- Model of SystemC
- An example

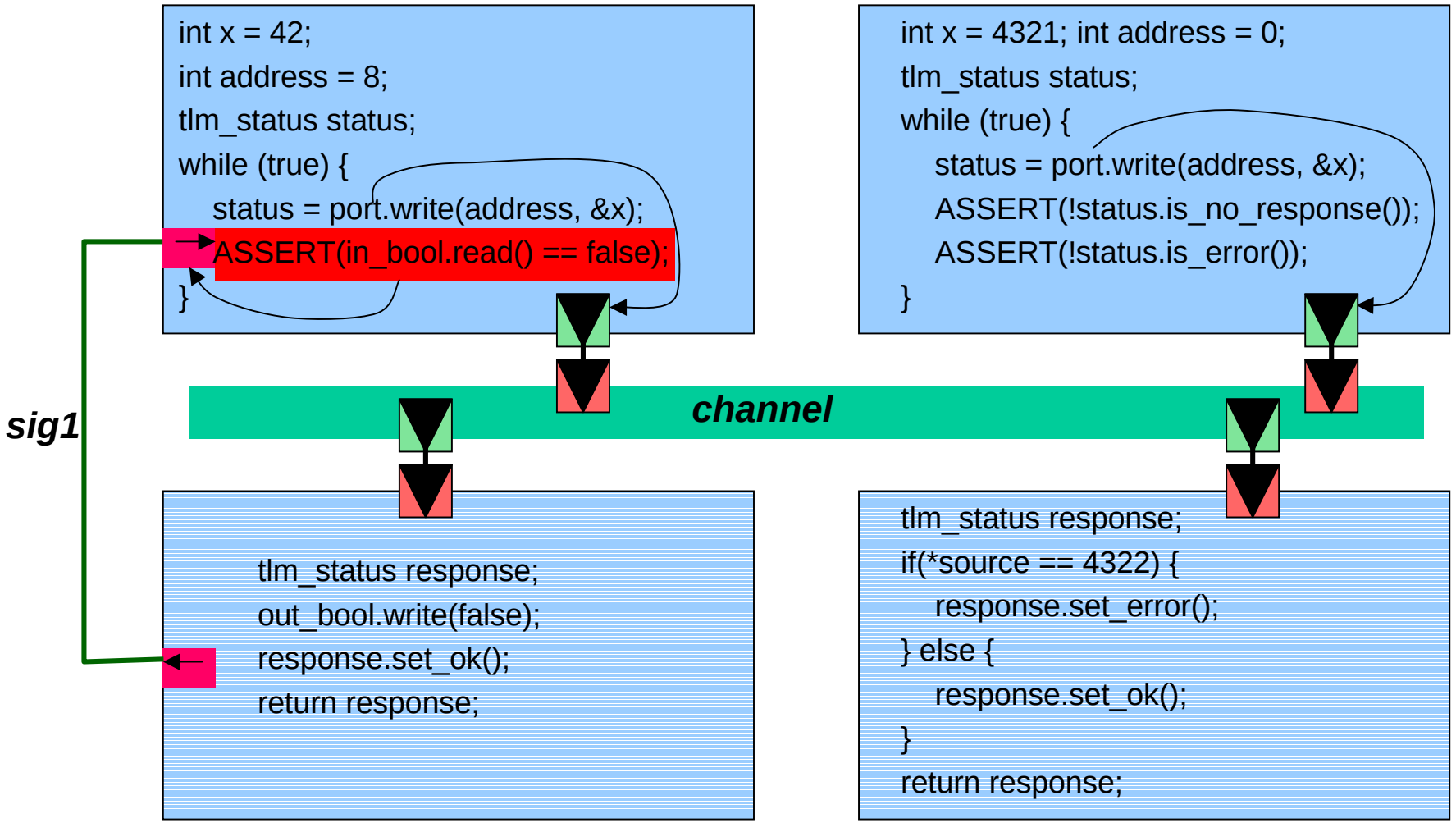
An example


int x = 42;
int address = 8;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(in_bool.read() == false);
}
int x = 4321; int address = 0;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(!status.is_no_response());
ASSERT(!status.is_error());
}
sig1
channel
tlm_status response;
out_bool.write(false);
response.set_ok();
return response;
tlm_status response;
if(*source == 4322) {
response.set_error();
} else {
response.set_ok();
}
return response;

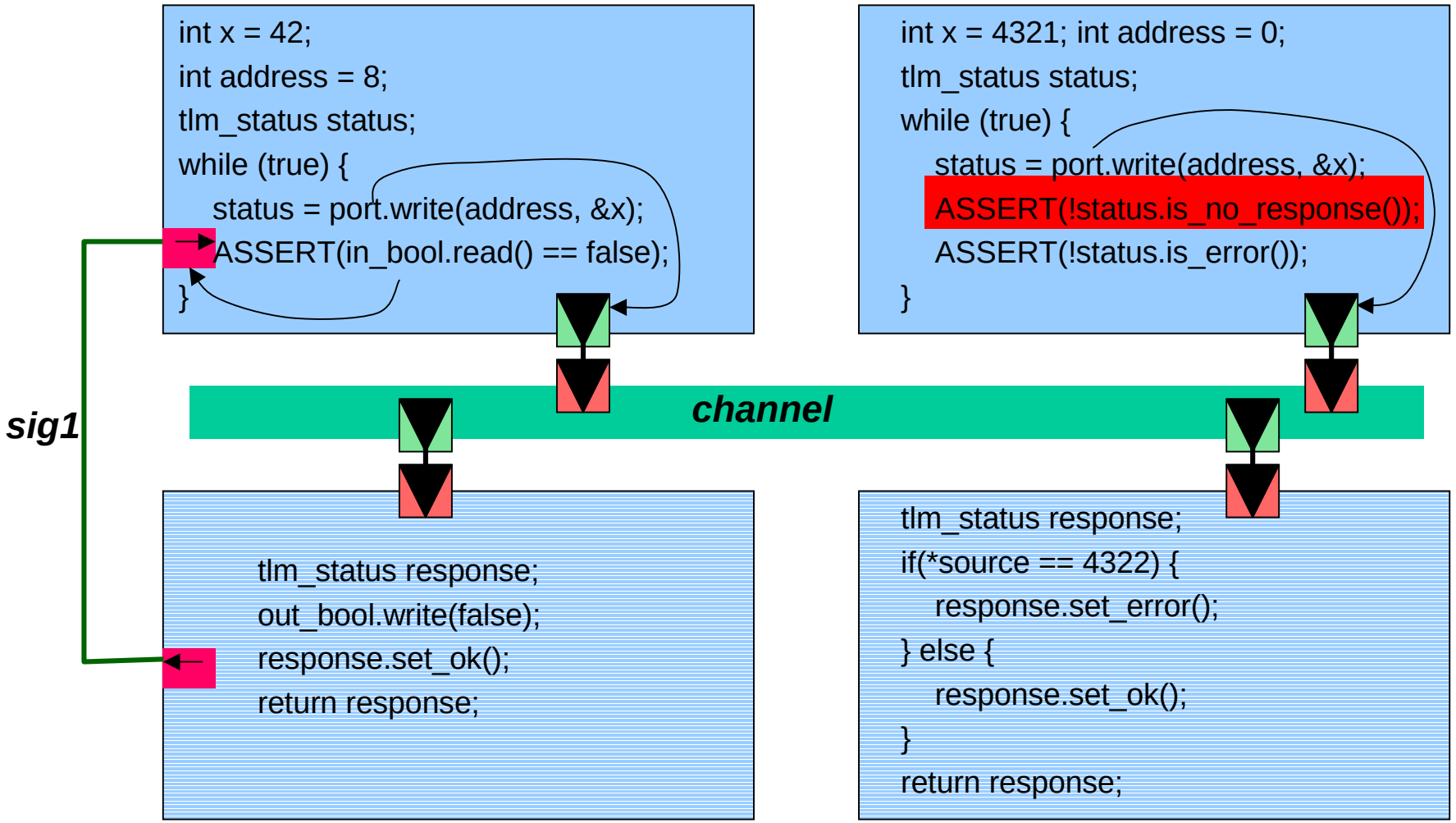
An example


int x = 42;
int address = 8;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(in_bool.read() == false
}
int x = 4321; int address = 0;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(!status.is_no_response());
ASSERT(!status.is_error());
}
sig1
channel
tlm_status response;
out_bool.write(false);
response.set_ok();
return response;
tlm_status response;
if(*source == 4322) {
response.set_error();
} else {
response.set_ok();
}
return response;

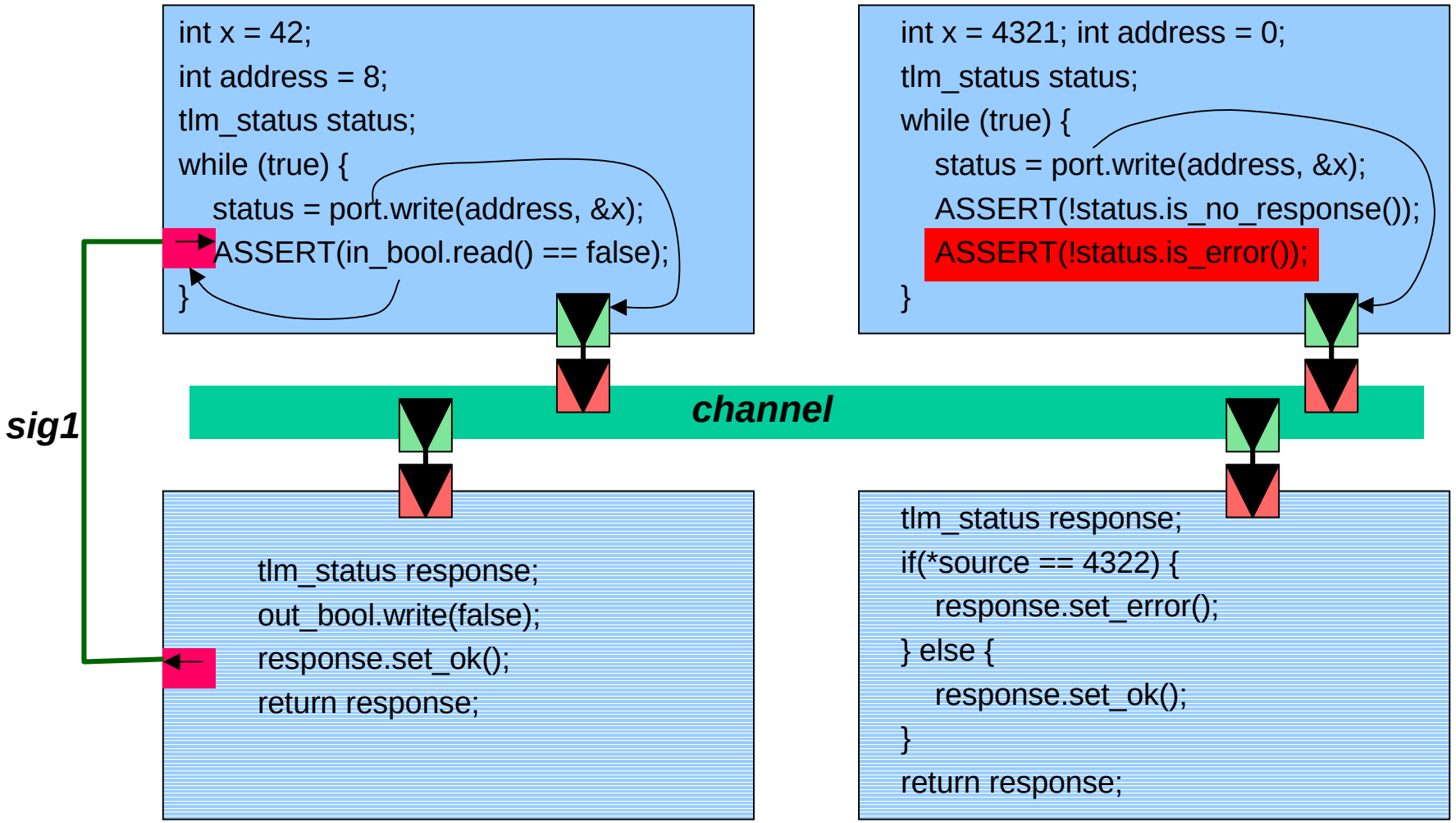
An example


int x = 42;
int address = 8;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(in_bool.read() == false);
}
int x = 4321; int address = 0;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(!status.is_no_response());
ASSERT(!status.is_error());
}
sig1
channel
tlm_status response;
out_bool.write(false);
response.set_ok();
return response;
tlm_status response;
if(*source == 4322) {
response.set_error();
} else {
response.set_ok();
}
return response;

An example


int x = 42;
int address = 8;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(in_bool.read() == false);
}
int x = 4321; int address = 0;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(!status.is_no_response());
ASSERT(!status.is_error());
}
sig1
channel
tlm_status response;
out_bool.write(false);
response.set_ok();
return response;
tlm_status response;
if(*source == 4322) {
response.set_error();
} else {
response.set_ok();
}
return response;

An example


int x = 42;
int address = 8;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(in_bool.read() == false);
}
int x = 4321; int address = 0;
tlm_status status;
while (true) {
status = port.write(address, &x);
ASSERT(!status.is_no_response());
ASSERT(!status.is_error());
}
sig1
channel
tlm_status response;
out_bool.write(false);
response.set_ok();
return response;
tlm_status response;
if(*source == 4322) {
response.set_error();
} else {
response.set_ok();
}
return response;

Still to be done

- Implement more C++/SystemC/TLM constructs
- Implement more generic properties
- Scale up
- *Try to* apply on a real-world case study

Interesting perspectives

- Better diagnosis
- Test-case generation
- Tools for debugging

Conclusion

- Work accomplished : A running prototype
- Encouraging but still much to be done

Questions ?