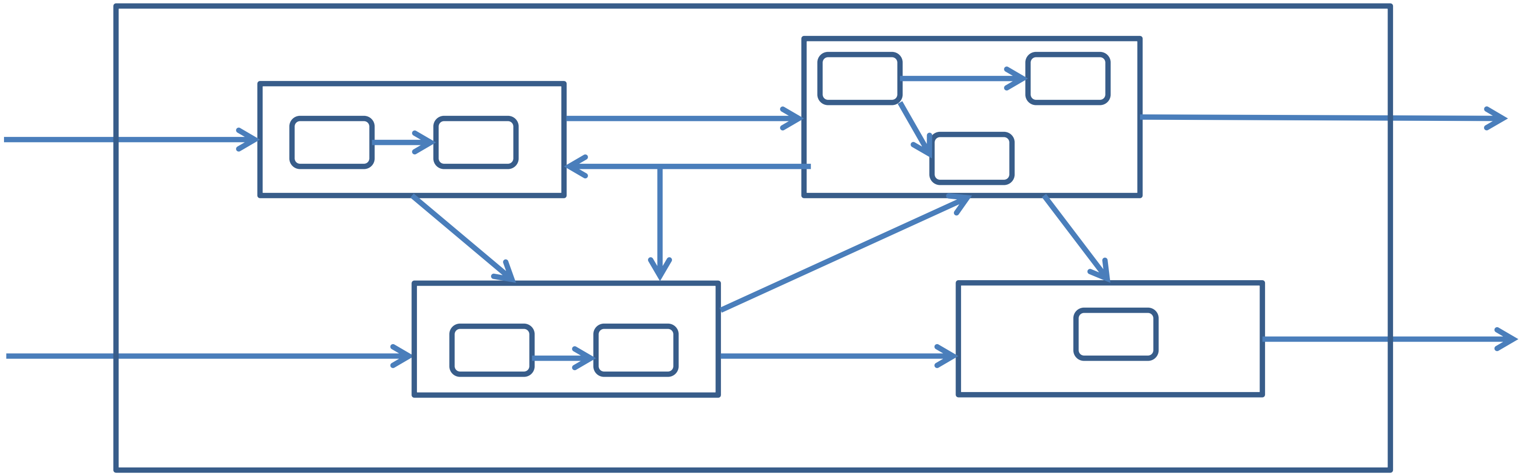
CS:4980
Foundations of Embedded Systems

Synchronous Model
Part III

Synchronous Design

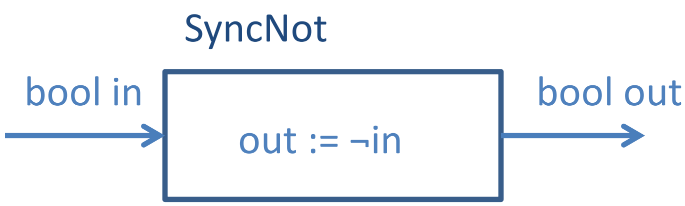


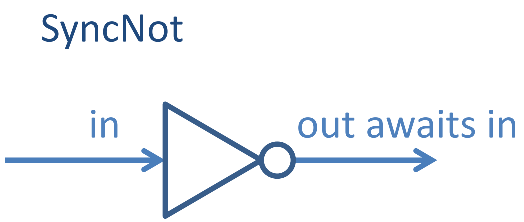
Component (de)composition can be stratified

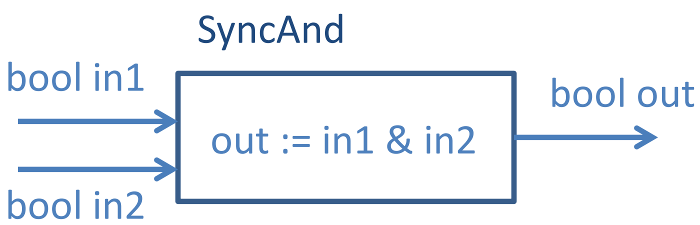
Bottom-Up Design

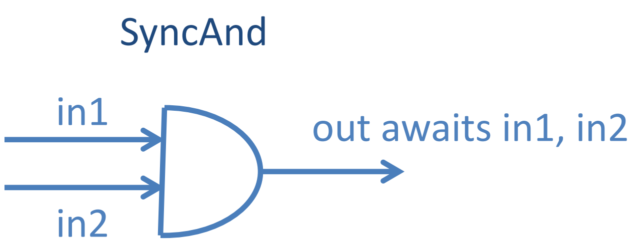
- Design basic components
- Compose existing components in block-diagrams to build new components
- Maintain a library of components, and try to reuse at every step
- Canonical example: Synchronous circuits

Combinational Circuits

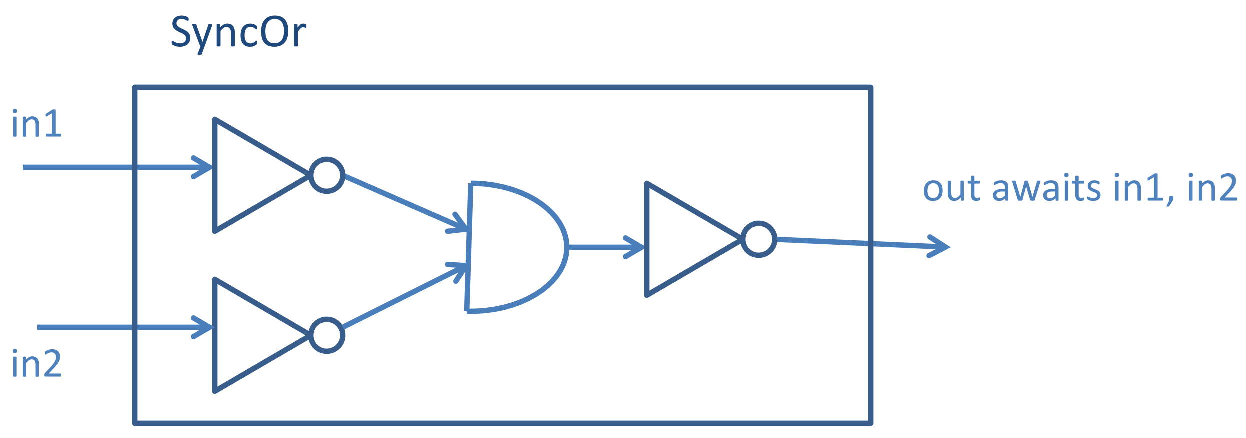






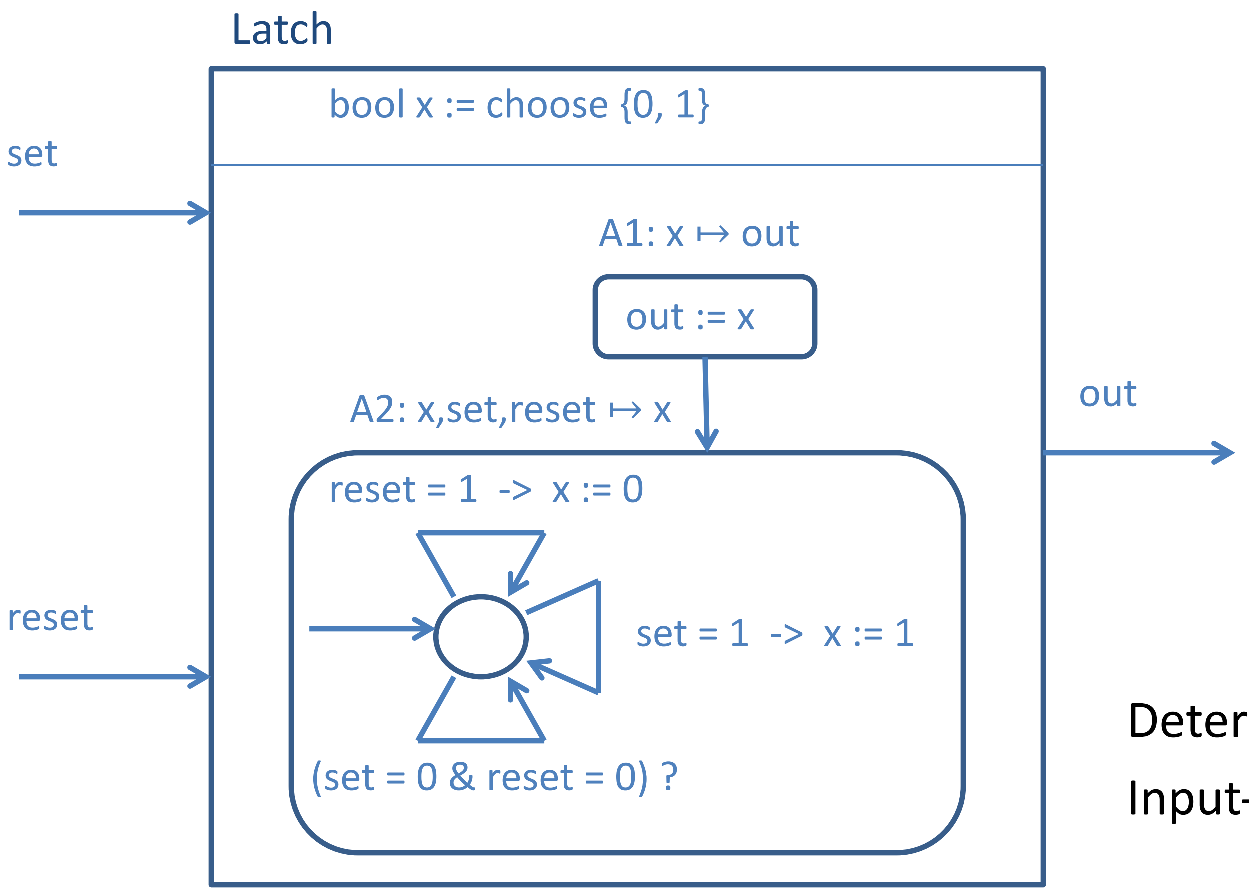


Design OR gate



Recall: $A \lor B \equiv \neg(\neg A \land \neg B)$

Synchronous Latch

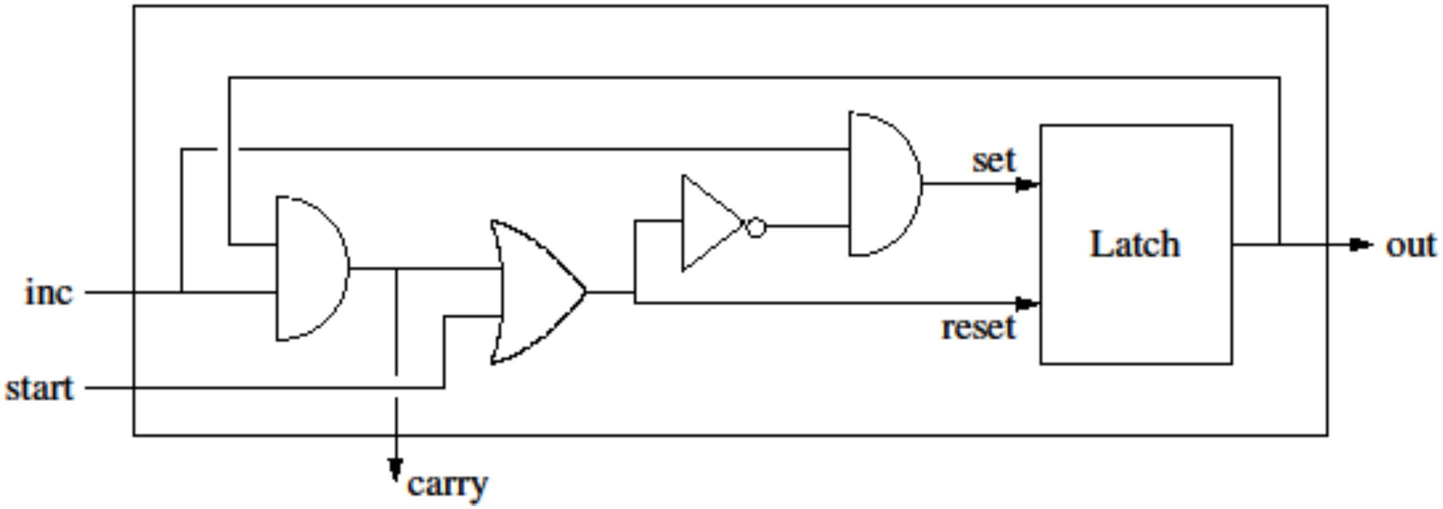


Deterministic?

Input-enabled?

Designing Counter Circuit (1)

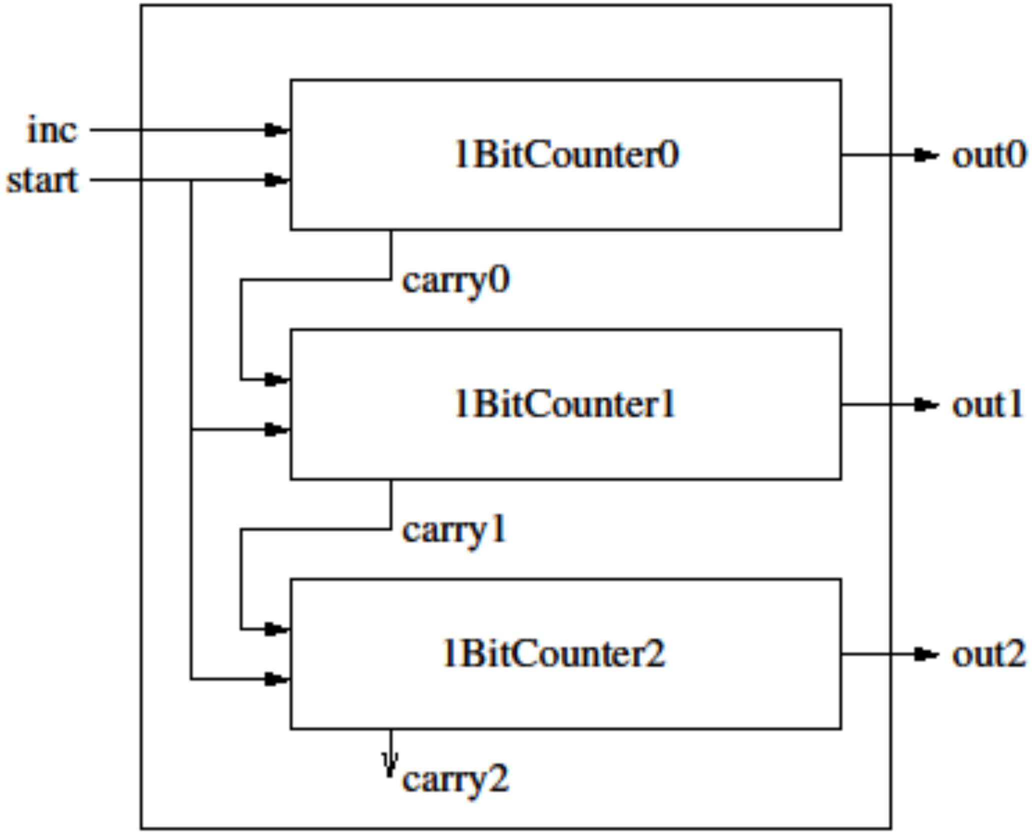
1BitCounter



Are await-dependencies acyclic?

Designing Counter Circuit (2)

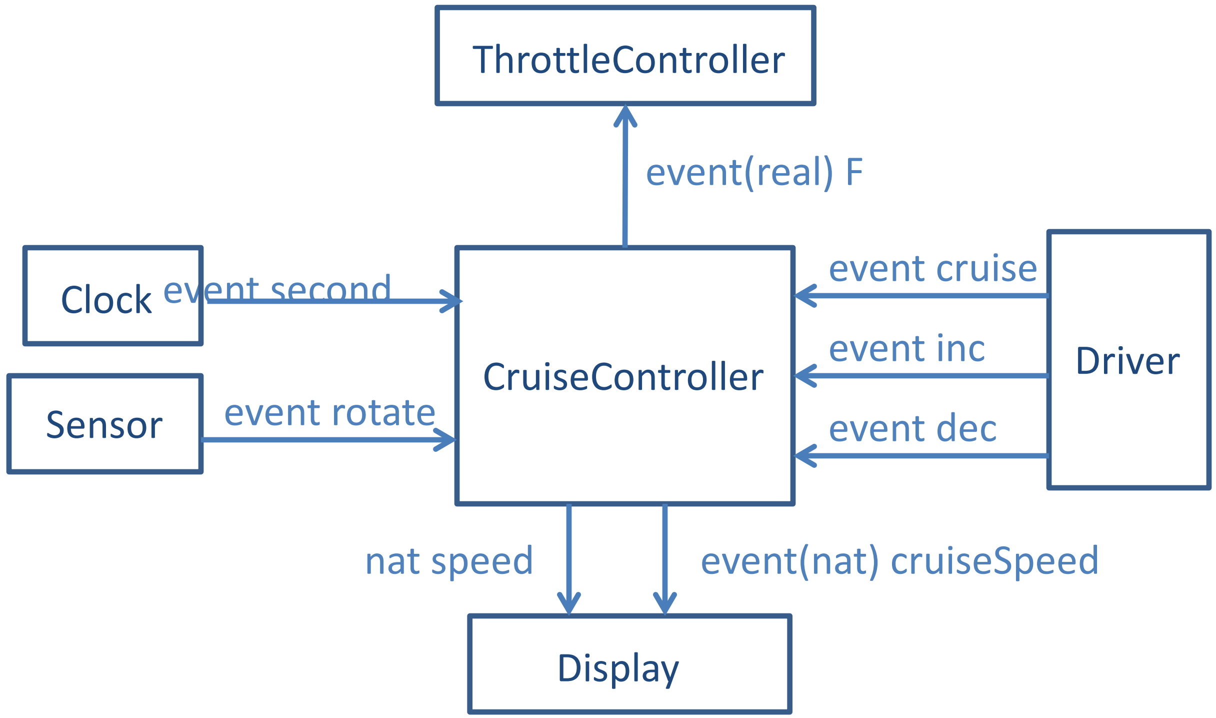
3BitCounter



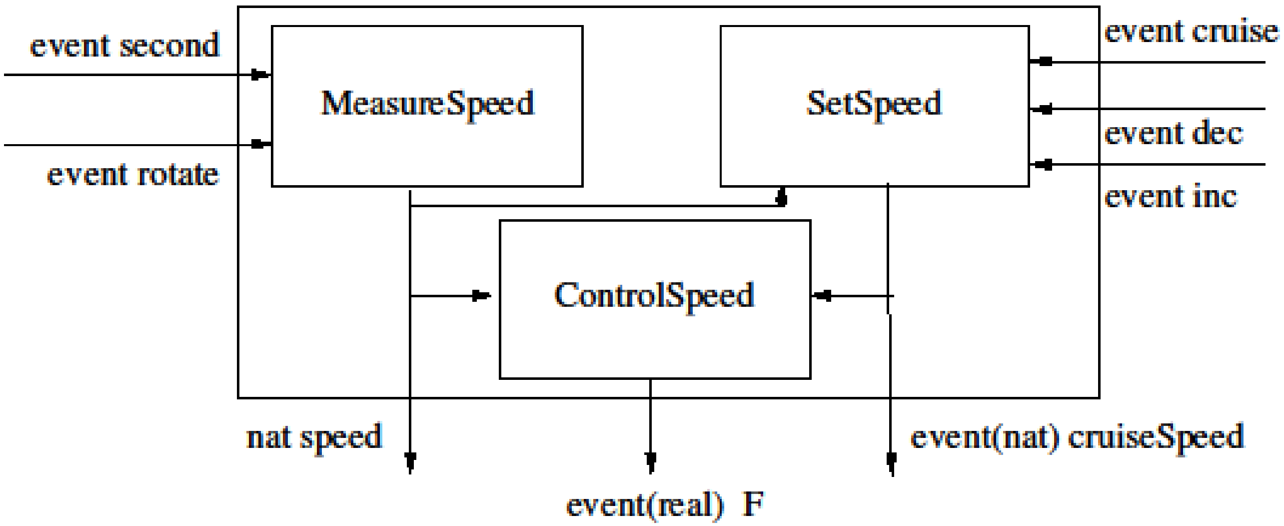
Top-Down Design

- Starting point: Inputs and outputs of desired design C
- Models/assumptions about the environment C operates in
- Informal/formal description of desired behavior of C
- Example: Cruise Controller

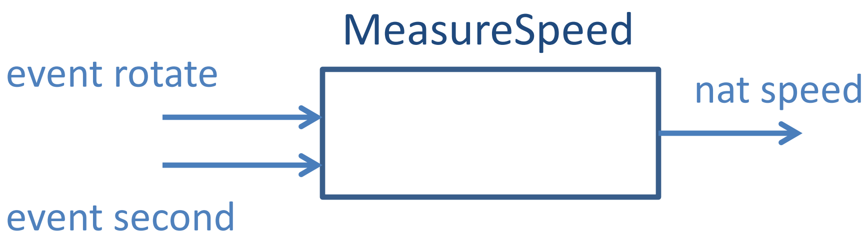
Top-Down Design of a Cruise Controller



Decomposing CruiseController

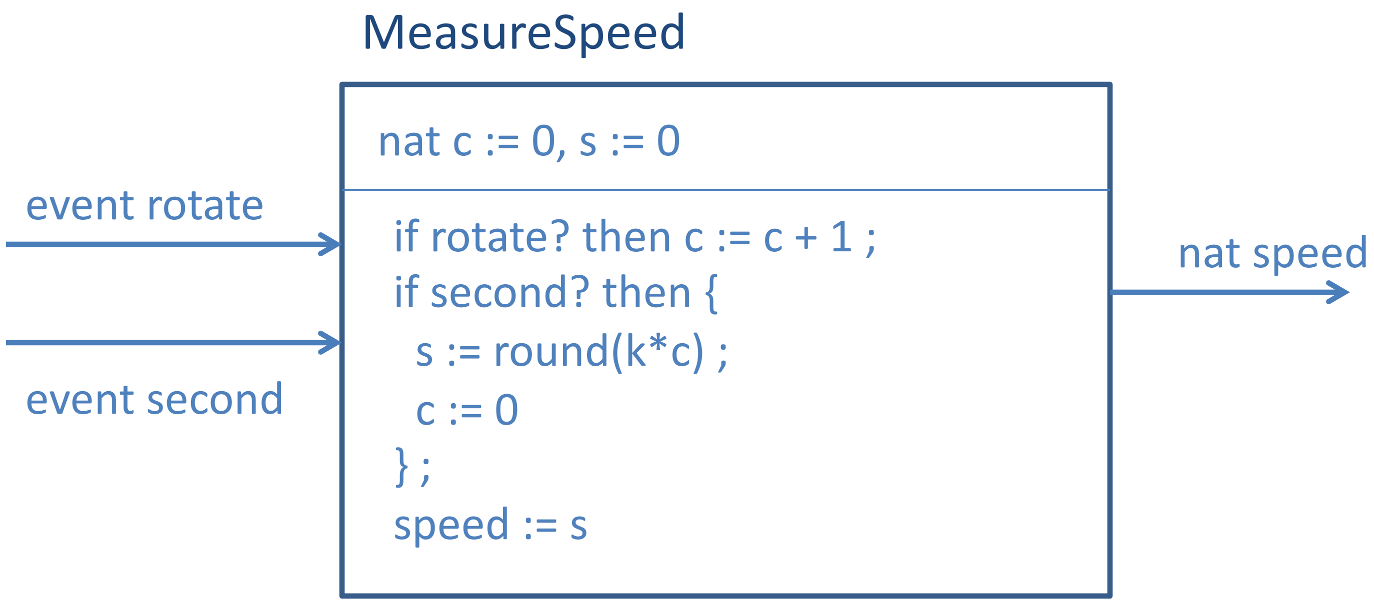


Tracking Speed

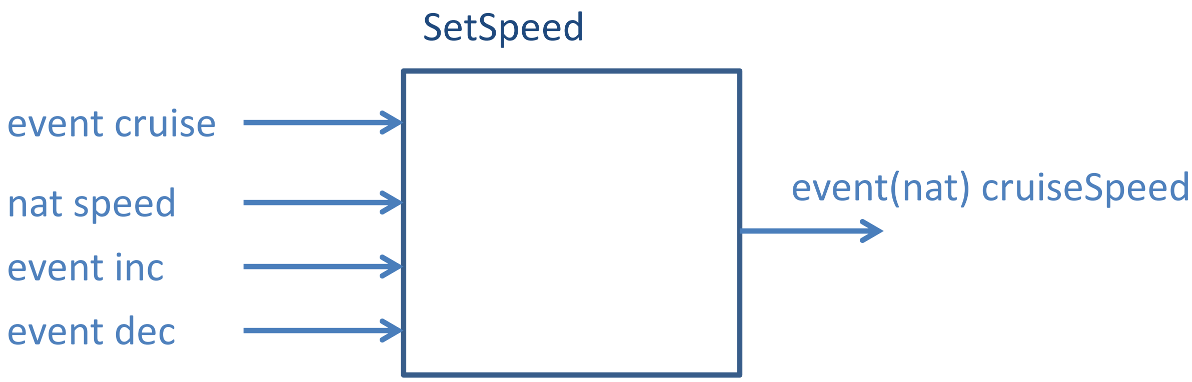


- **Inputs:** Events rotate and second
- **Output:** current speed
- Computes the number of rotate events per second

Tracking Speed

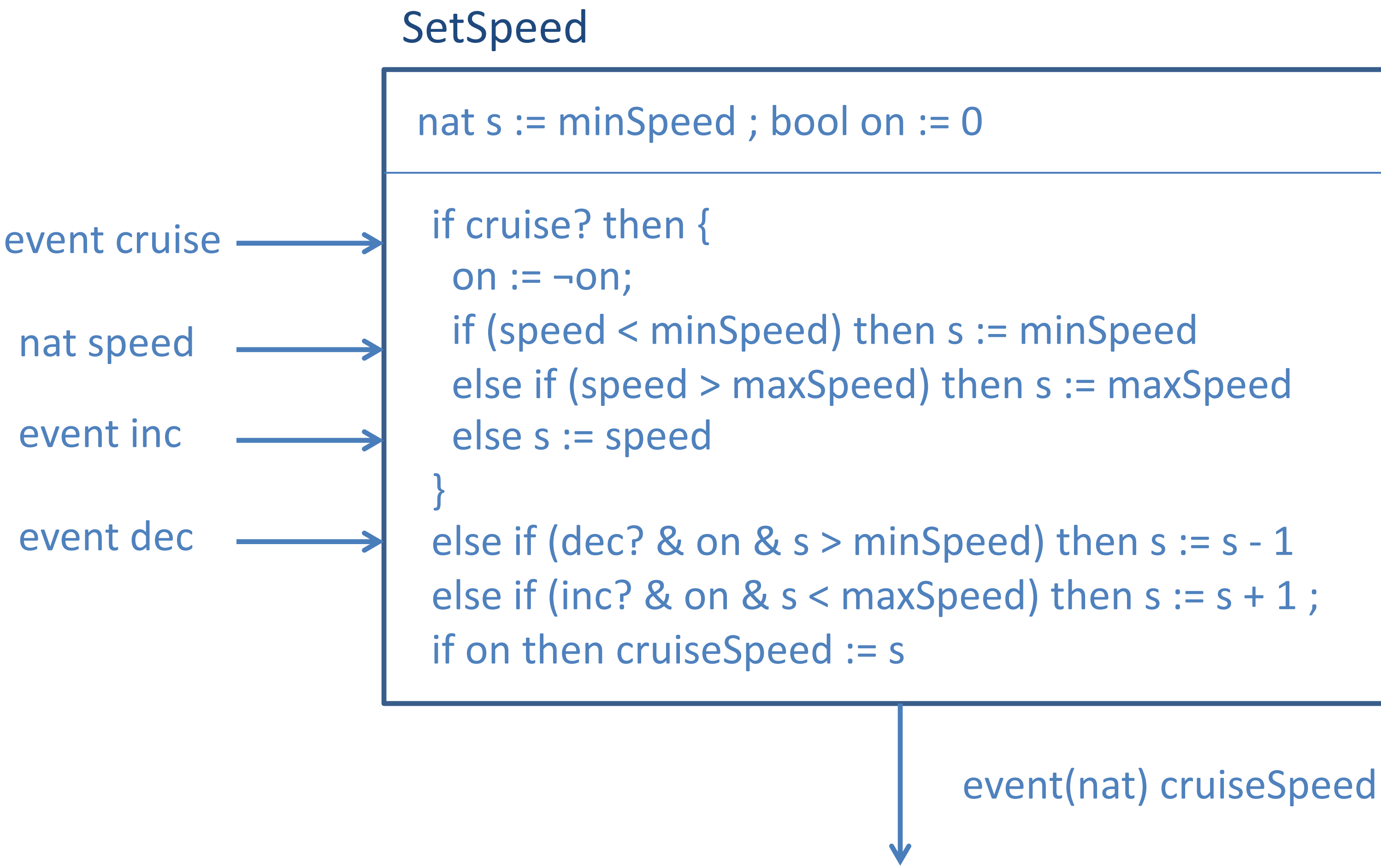


Tracking Cruise Settings

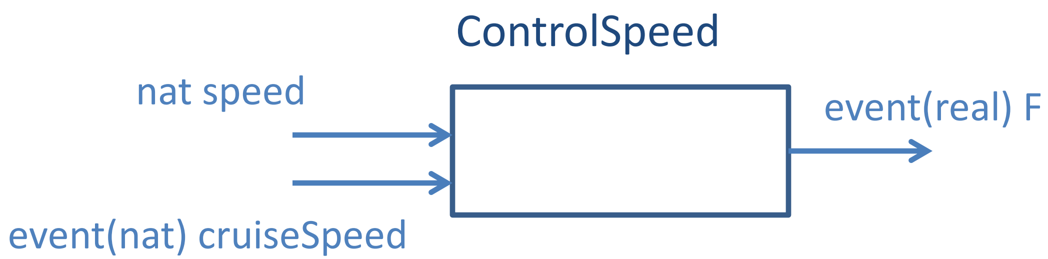


- **Inputs from driver:** commands to turn the cruise-control on/off and increment/decrement desired cruising speed from driver
- **Input from** MeasureSpeed**:** current speed
- **Output:** Desired cruising speed
- What assumptions can we make about simultaneity of events?
- Should we include safety checks to keep desired speed within bounds?

Tracking Cruise Settings

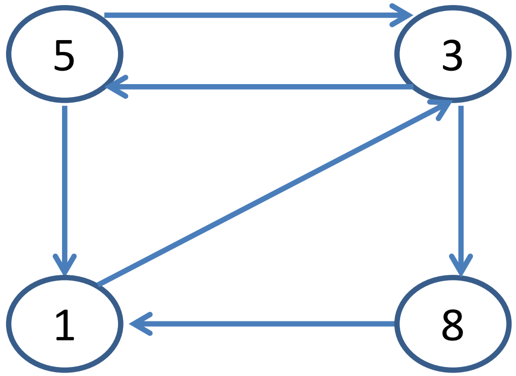


Controlling Speed



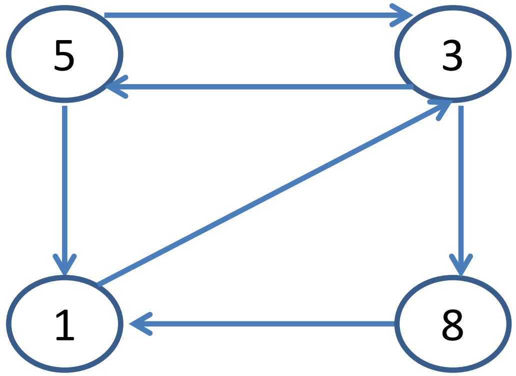
- **Inputs:** Actual speed and desired speed
- **Output:** Pressure on the throttle
- **Goal:** Make actual speed equal to the desired speed (while maintaining key physical properties such as stability)
- Design relies on theory of dynamical systems (Chapter 6)

Synchronous Networks



- Time divided into slots, with all nodes synchronized
- In one round, each node can get a message from each neighbor
- Design abstraction for simplicity
- Some implementation platforms directly support such a *time-triggered* network: WirelessHART (control), CAN (automotive)

Modeling Synchronous Networks



Assume: Each link is directed and connects two nodes

Alternative: Broadcast communication (everyone can listen)

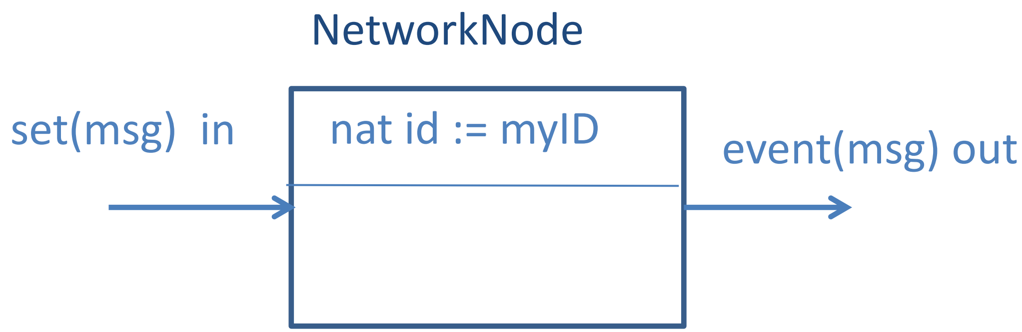
Assume: Communication is reliable

Alternative: Messages may be lost, collisions in broadcast

Network is a directed graph

Each link can carry one message in each slot

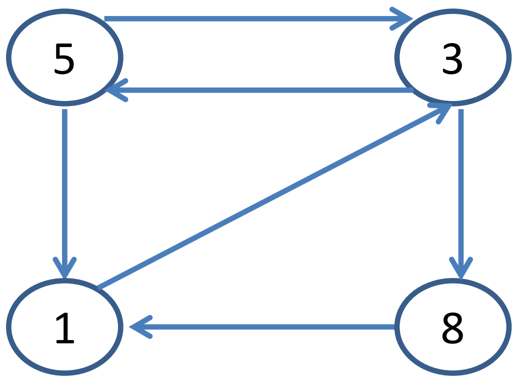
Component for a Network Node



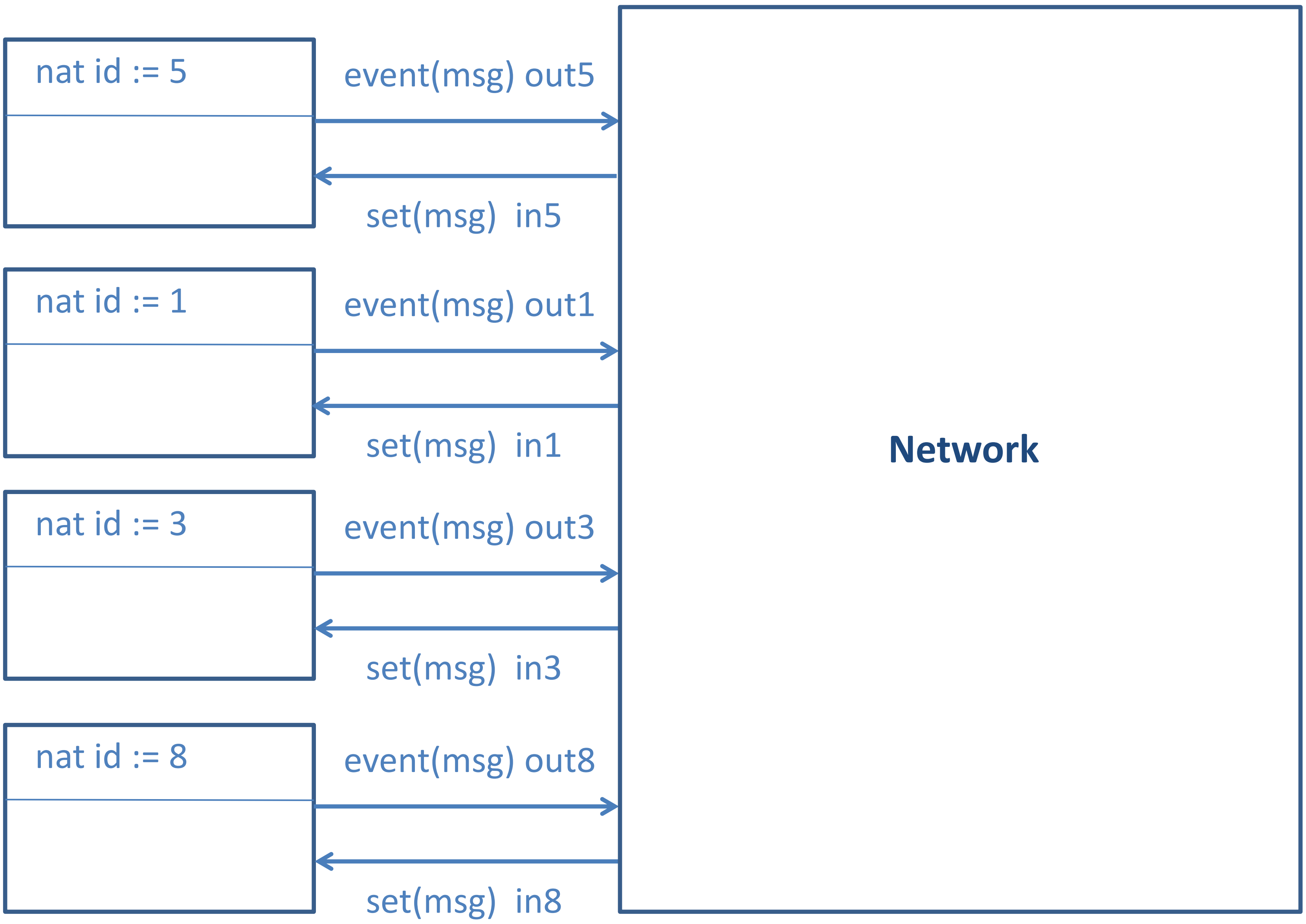
- A node does not know network topology
  - Each node has unique identifier, myID
  - Does not know which nodes it is connected to
  - Useful for network identification problems

- Interface for each node:
  - Output is an event carrying msg (may be absent in some rounds)
  - Input is a set of msg (delivered by the network)
  - Output should not await input

Modeling Synchronous Networks



- Description of each node does not depend on the network
- Network itself is modeled as a synchronous component
- Description of network depends on the network graph
  - Input variables: for each node n, out_n of type event(msg)
  - Output variables: for each node n, in_n of type set(msg)
- Network is a combinational component (simply routes messages)


nat id := 5
event(msg) out5
set(msg) in5
nat id := 1
event(msg) out1
set(msg) in1
nat id := 3
event(msg) out3
set(msg) in3
nat id := 8
event(msg) out8
set(msg) in8
Network

Network

event(msg) out5

set(msg) in5

event(msg) out1

set(msg) in1

event(msg) out3

set(msg) in3

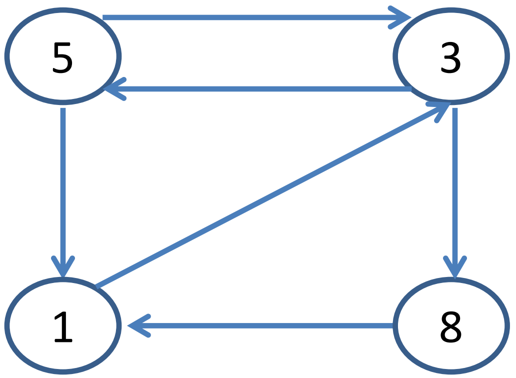
event(msg) out8

set(msg) in8

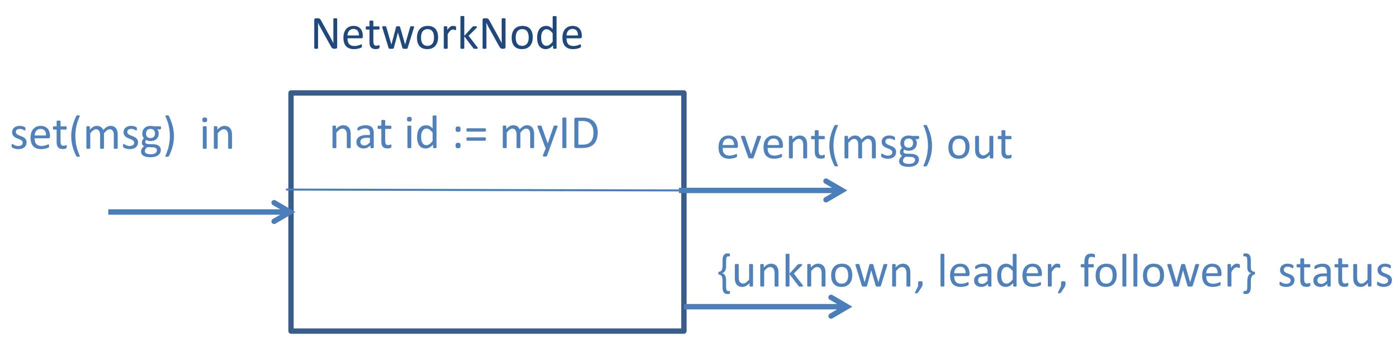
- Value of in1 should equal the set of messages sent on links incoming to node 1
- Sample code:

```
in1 := EmptySet ;
if out5? then Insert(out5, in1) ;
if out8? then Insert(out8, in1) ;
```

- Update of in5, in3, in8 is similar



Leader Election



Classical coordination problem: Elect a unique node as a leader

- Exchange messages to find out which nodes are in network
- Output the decision using the variable status

Requirements:

1. Eventually every node sets status to either leader or follower
2. Only one node sets its status to leader

Leader Election: Flooding Algorithm

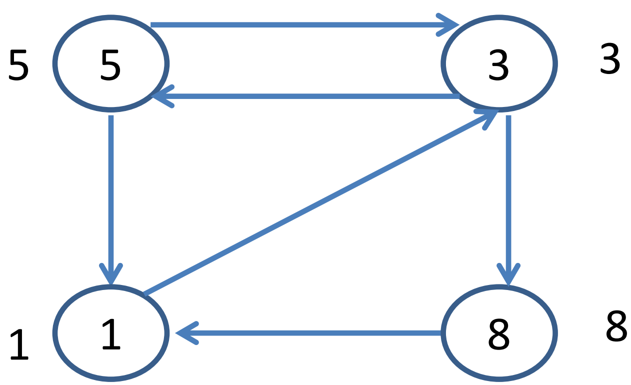
Goal: Elect the node with highest identifier as the leader

Strategy: Transmit to your neighbors the highest id you have encountered so far

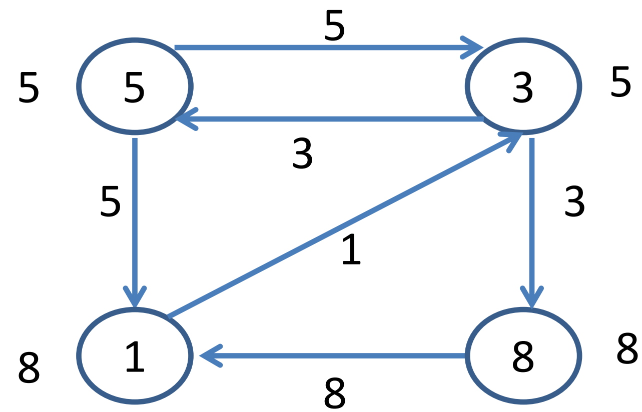
Implementation:

- Maintain a state variable, id, initialized to your own identifier
- In each round, transmit value of id on output
- Receive input values from the network
- If a value higher than id received, then update id

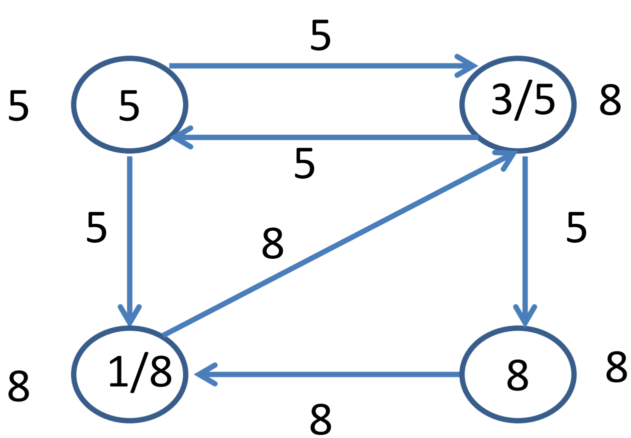
Execution of Leader Election



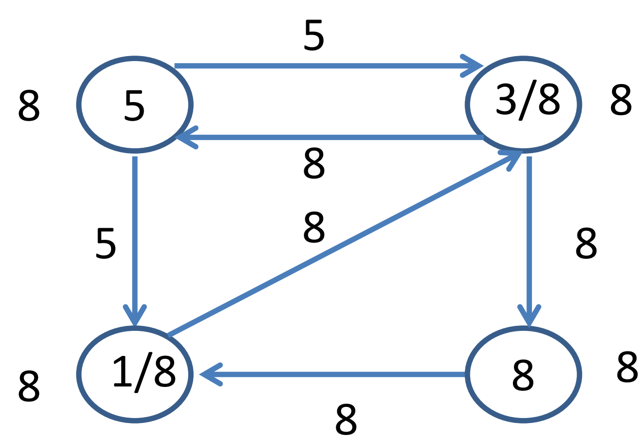
Initially



First round



Second round

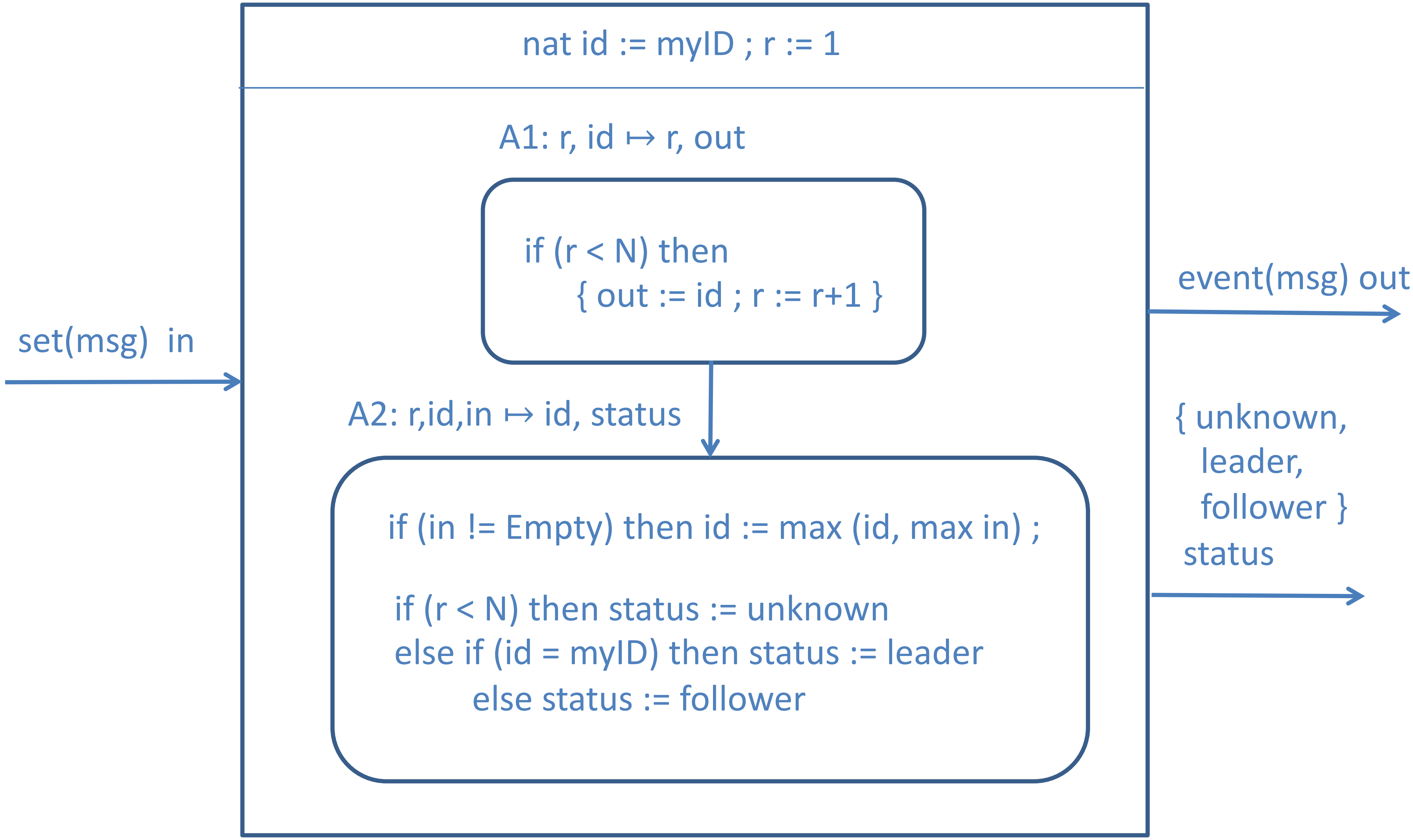


Third round

Leader Election

- When should a node stop and make a decision?
- When it knows that enough rounds have elapsed for message from every node to reach every other node
- Correctness depends on following assumptions:
  1. Network is strongly connected: for every pair of nodes m and n, there is a directed path from node m to node n
  2. Each node knows an upper bound N on total number of nodes
- Implementation of decision rule:
  - Maintain a state variable r to count rounds, initially 1
  - In each round, r is incremented
  - When r = N, decide
- What should the decision be?

Node Component for Leader Election



Leader Election

- Does a node really have to wait for N rounds?
- If a node receives a value higher than its own identifier, can it stop participating (i.e. transmit no more messages)?
- Does a node have to transmit in each round? When can it choose to skip a round without affecting correctness?

Credits

Notes based on Chapter 2 of

Principles of Cyber-Physical Systems
by Rajeev Alur
MIT Press, 2015