CS:4980
Foundations of Embedded Systems

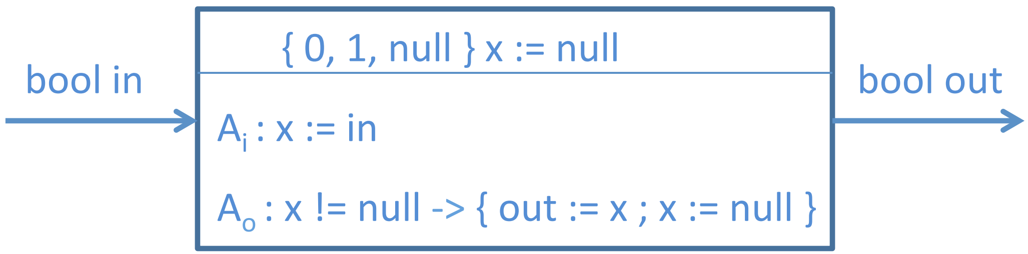
The Asynchronous Model
Part I

Asynchronous Model

- Recall: In the Synchronous Model, all components execute in a sequence of (logical) rounds in lock-step
- In the Asynchronous Model instead the speeds at which different components execute are independent, or unknown

  Examples:
  - Processes in a distributed system
  - Threads in a typical operating system such as Linux/ Windows
- Key design challenge: how to achieve coordination?

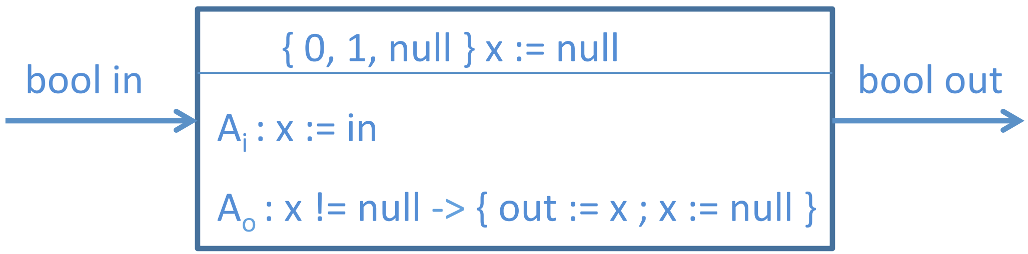
Example: Asynchronous Buffer



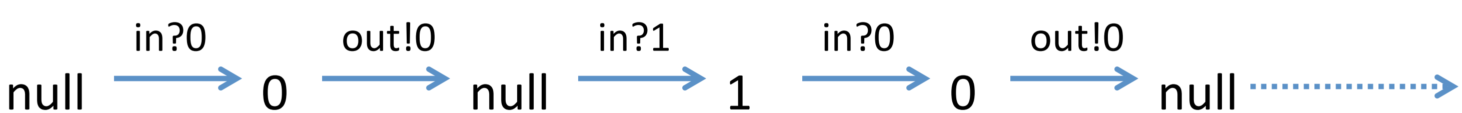
- Input channel: in of type Boolean
- Output channel: out of type Boolean
- State variable: x; can be empty (null), or hold 0/1 value
- Initialization of state variables: assignment x := null
- Input task A_i for processing of inputs: code: x := in
- Output task A_o for producing outputs:

  Guard: x != null *Update:* out := x ; x := null

Example: Asynchronous Buffer



- Execution Model: In one step, only a single task is executed
  - processing of inputs (by input tasks) is decoupled from production of outputs (by output tasks)
- A task can be executed if it is *enabled*, i.e., its guard condition holds
  - If multiple tasks are enabled, one of them is executed
- Sample Execution:

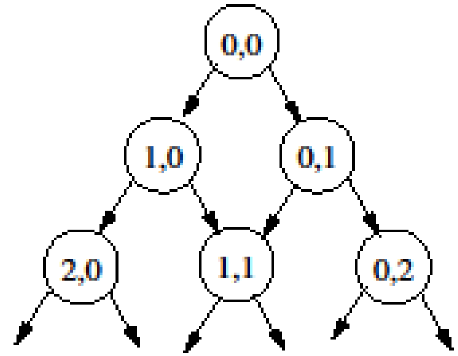


Example: Asynchronous Increments

nat x := 0 ; y := 0

A_x : x := x+1

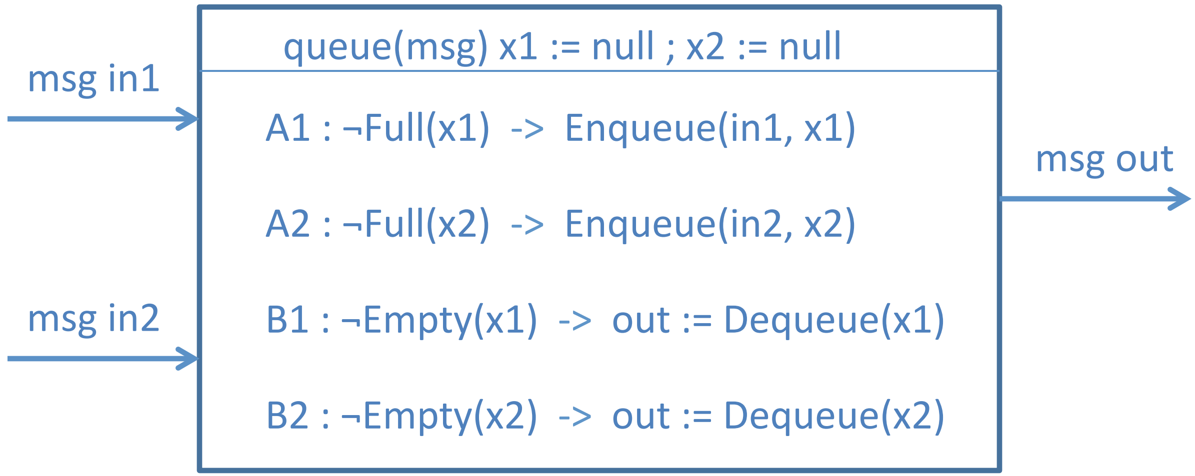
A_y : y := y+1



- An *internal task* does not involve input or output channels
  - Can have guard condition and update code
  - the execution of internal task in an internal action
- In each step, execute, either task A_x or task A_y
- Sample Execution:

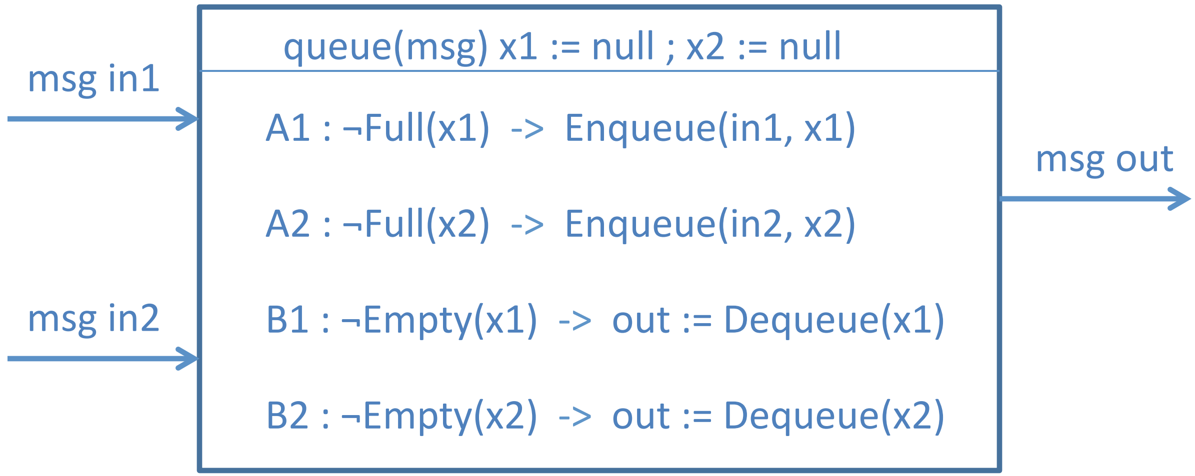
  (0,0) -> (1,0) -> (1,1) -> (1,2) -> (1,3) -> … -> (1,105) -> (2, 105) …
- For every m, n, state {x := m, y := n} is reachable
  - *Interleaving* model of concurrency

Asynchronous Merge



Sequence of messages on output channel is an arbitrary merge of sequences of values on the two input channels

Asynchronous Merge

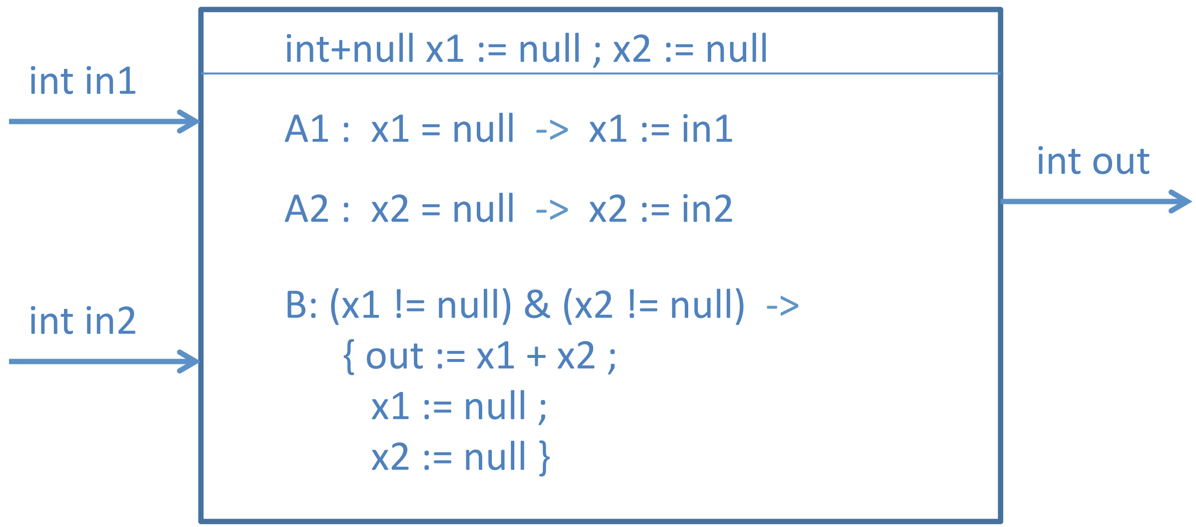


At every step exactly one of the four tasks executes, provided its guard condition holds

Sample Execution:

([],[]) –in1?5-> ([5], []) –in2?0-> ([5],[0]) –out!0-> ([5],[]) –in1?6-> ([5,6],[]) –in2?3-> ([5,6],[3]) –out!5-> ([6],[3]) …

What does this process do?



Asynchronous Process P

- Set I of (typed) *input channels*
  - Defines the set of inputs of the form x?v, where x is an input channel and v is a value
- Set O of (typed) *output channels*
  - Defines the set of outputs of the form y!v, where y is an output channel and v is a value
- Set S of (typed) *state variables*
  - Defines the set of states Q_S
- An initialization Init
  - Defines the set [Init] of initial states

Asynchronous Process P (cont.)

- Set of *input tasks*, each associated with an input channel x
  - Guard condition over state variables S
  - Update code from *read-set* $S \cup \{x\}$ to *write-set* S
  - Defines a set of *input actions* of the form $s \xrightarrow{x?v} t$

- Set of *output tasks*, each associated with an output channel y
  - Guard condition over state variables S
  - Update code from *read-set* S to *write-set* $S \cup \{y\}$
  - Defines a set of *output actions* of the form $s \xrightarrow{y!v} t$

- Set of *internal tasks*
  - Guard condition over state variables S
  - Update code from *read-set* S to *write-set* S
  - Defines a set of *internal actions* of the form $s \xrightarrow{\varepsilon} t$

Asynchronous Gates

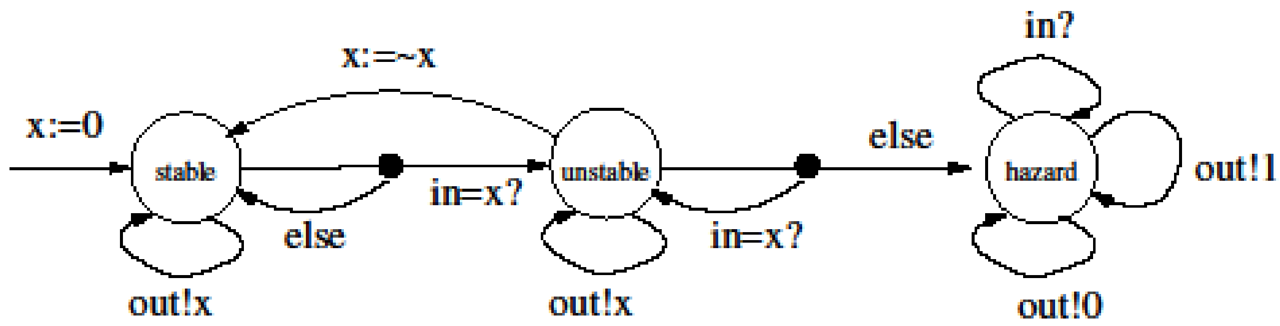
Why design asynchronous circuits?

- Input can be changed even before the effect propagates through the entire circuit
- Can be faster than synchronous circuits, but design is more complex

Example: modeling a NOT gate

- When input changes, gate enters *unstable* state until it gets a chance to update its output value
- If input changes again in unstable state, then this leads to a state with unpredictable behavior

Asynchronous NOT Gate as an ESM



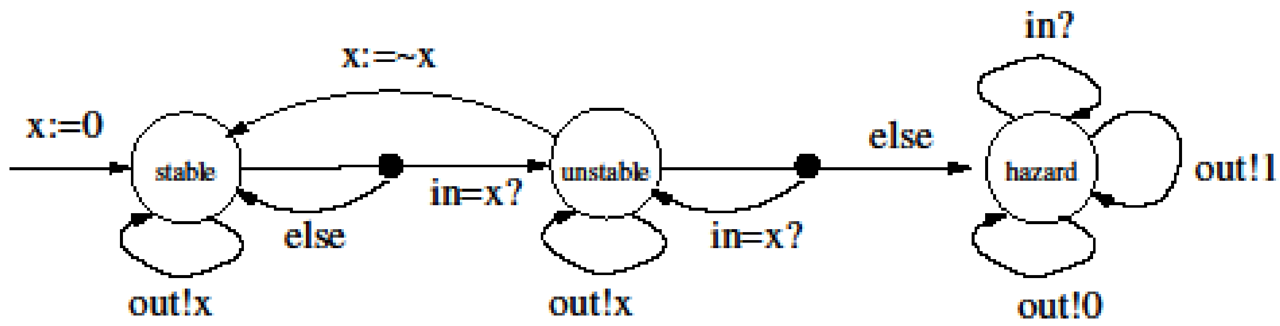
Sample Execution:

(stable,0) –out!0-> (stable,0) –in?0-> (unstable,0) –else->
(stable,1) –out!1-> (stable,1) –in?1-> (unstable,1) –out!1->
(unstable,1) –in?0-> (hazard,1) –out!0-> (hazard,1) –out!1->
(hazard,1) …

How to ensure that the gate does not enter hazard state?

Environment should wait to see a chance in value of output before toggling input again

Executing an ESM

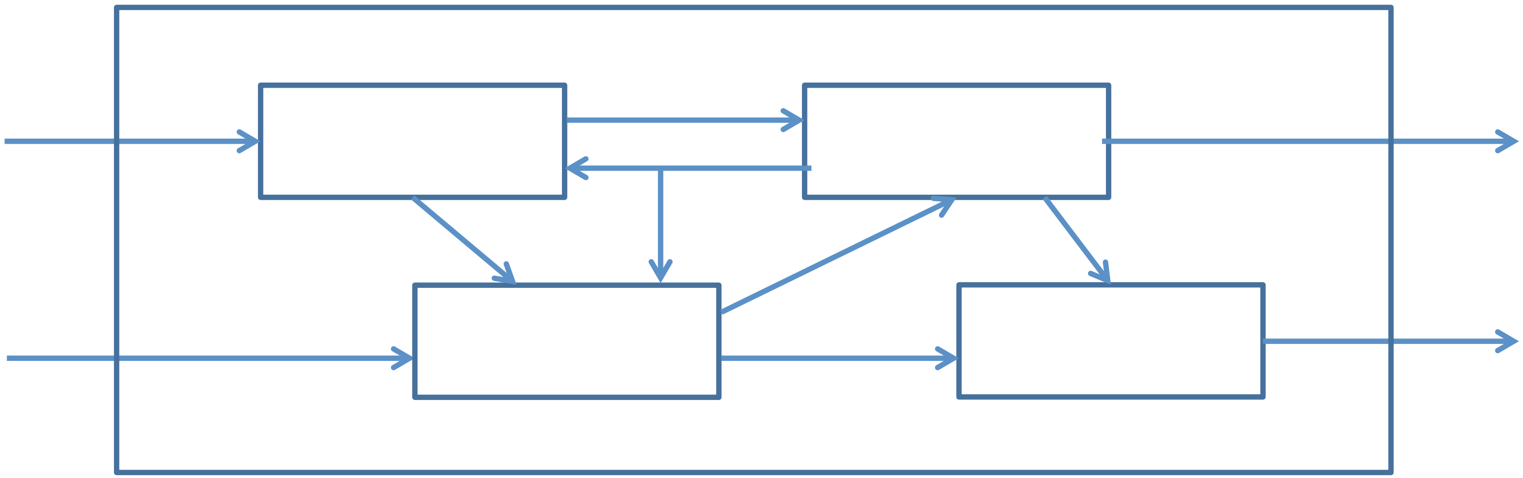


Each mode-switch corresponds to a task

Examples

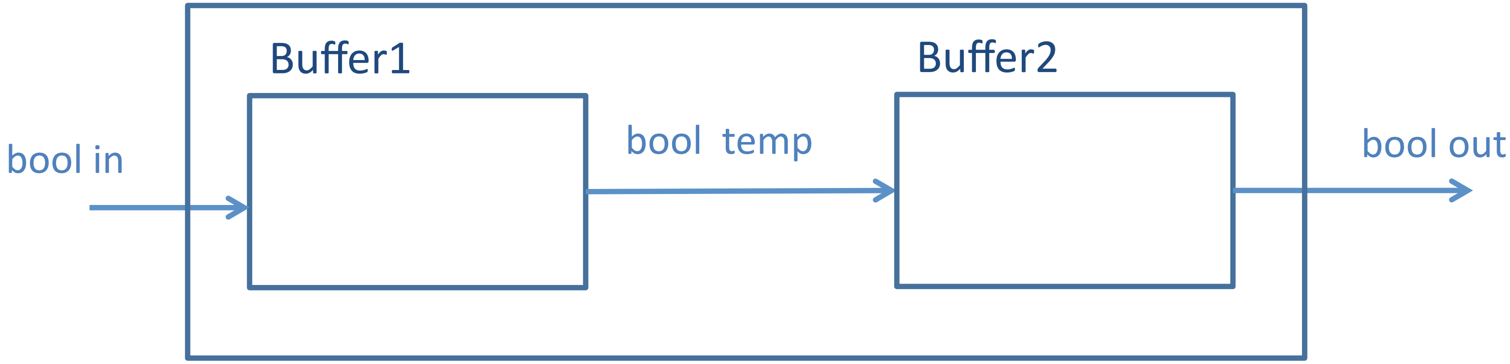
- Input task: (mode = stable) -> if (in = x) then mode := unstable
- Output task: (mode = stable) -> out := x
- Internal task: (mode = unstable) -> { x := ¬x ; mode := stable }

Block Diagrams



- Visually the same as the synchronous case
- However, their execution semantics is different !

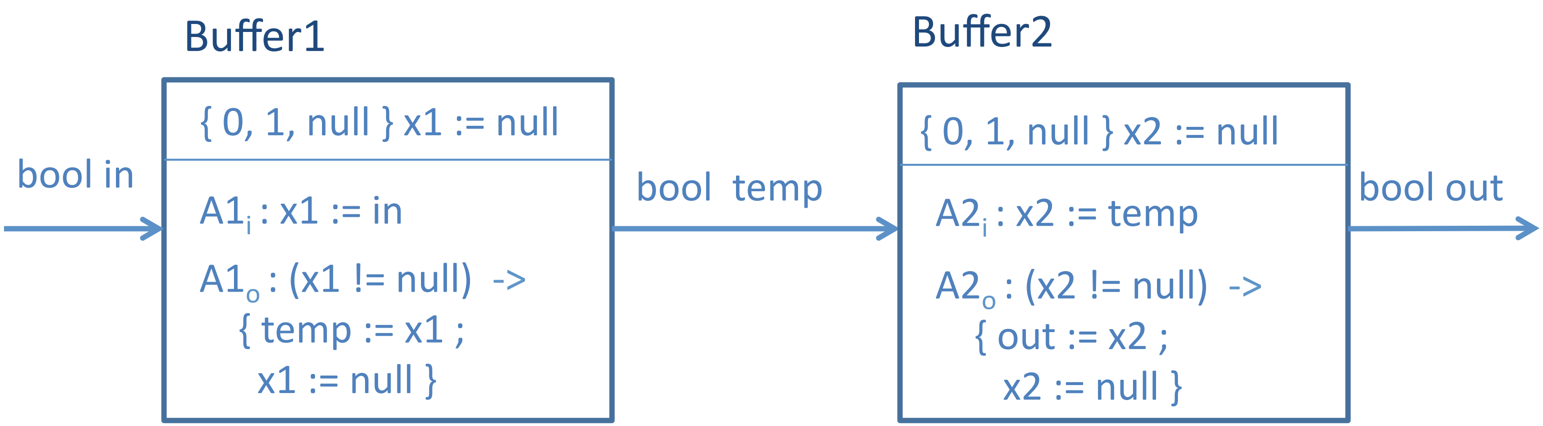
DoubleBuffer



(Buffer[out -> temp] | Buffer[in -> temp]) \ temp

- *Instantiation*: Create two instances of Buffer
  - output of Buffer1 = input of Buffer2 = variable temp
- *Parallel composition*: Asynchronous concurrent execution of Buffer1 and Buffer2
- *Variable hiding*: Encapsulation (temp becomes local)

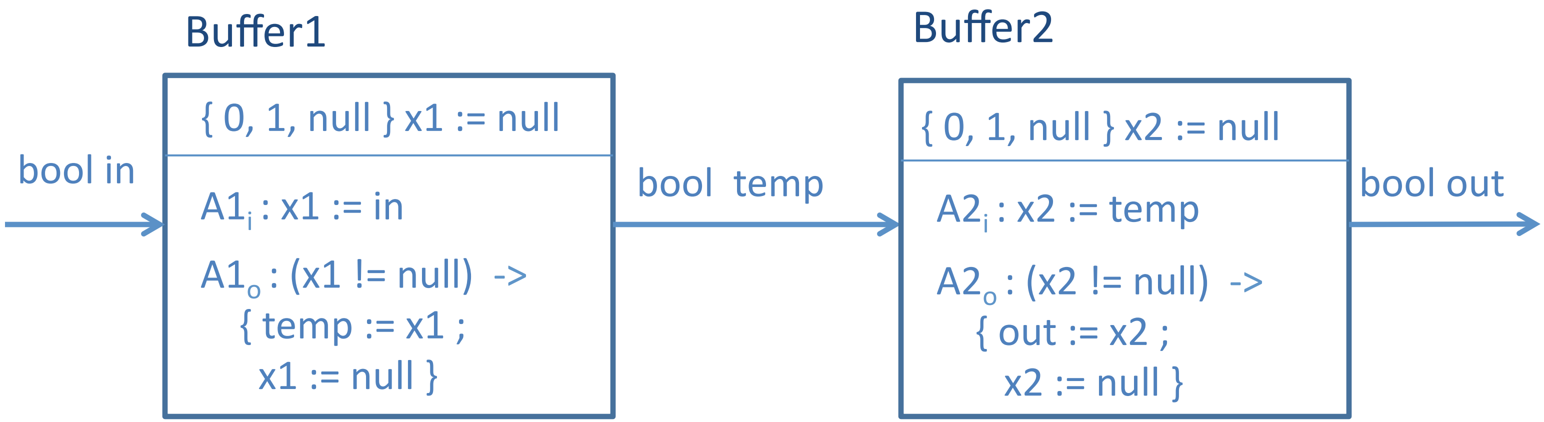
Composing Buffer1 and Buffer2

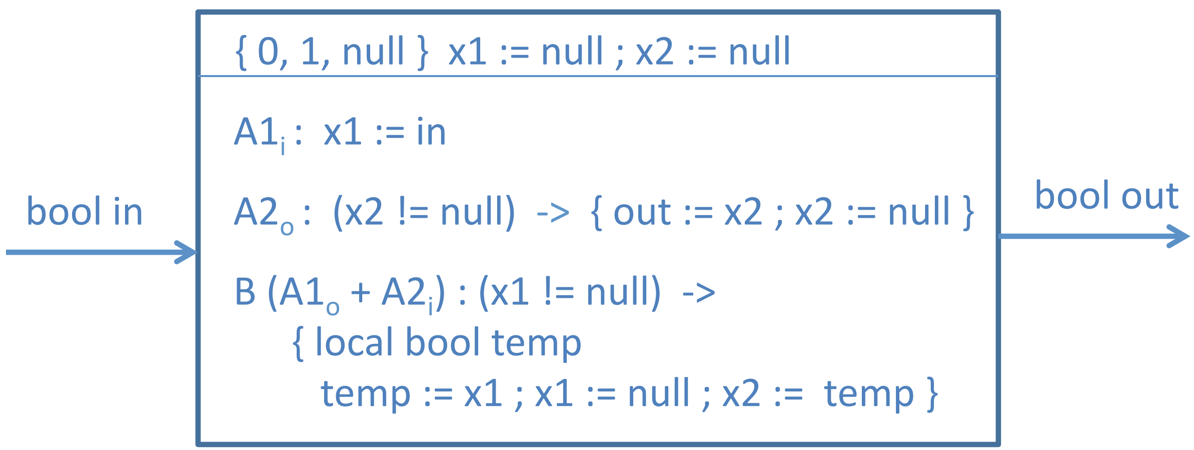


- Inputs, outputs, states, and initialization for composition obtained as in synchronous case
- What are the tasks of the composition?

  Production of output on temp by Buffer1 synchronized with consumption of input on temp by Buffer2

Compiled DoubleBuffer





Asynchronous Composition

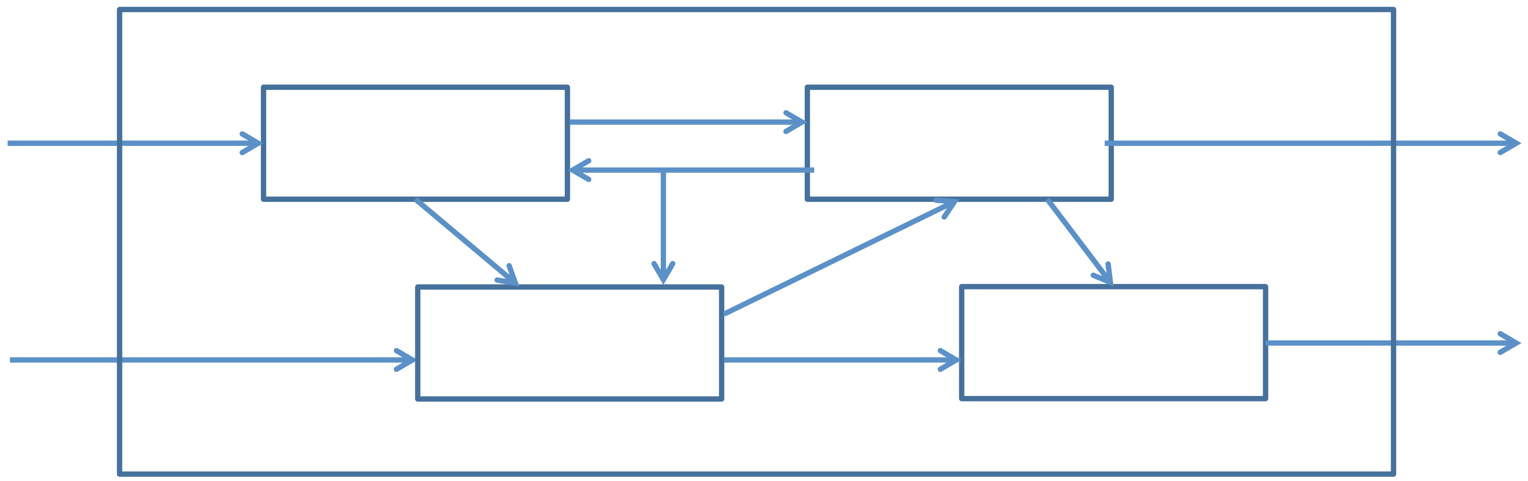
- Given asynchronous processes P_1 and P_2, how to define $P_1 \mid P_2$?
- In each step of execution, only one task is executed
  - Concepts such as await-dependencies, compatibility of interfaces, are not relevant
- Sample case (see textbook for complete definition):

  If
  - y is an output channel of P_1 and input channel of P_2,
  - A_1 is an output task of P_1 for y with code: $Guard_1$ -> $Update_1$,
  - A_2 is an input task of P_2 for y with code: $Guard_2$ -> $Update_2$,

  then
  - $P_1 \mid P_2$ has an output task for y with code:

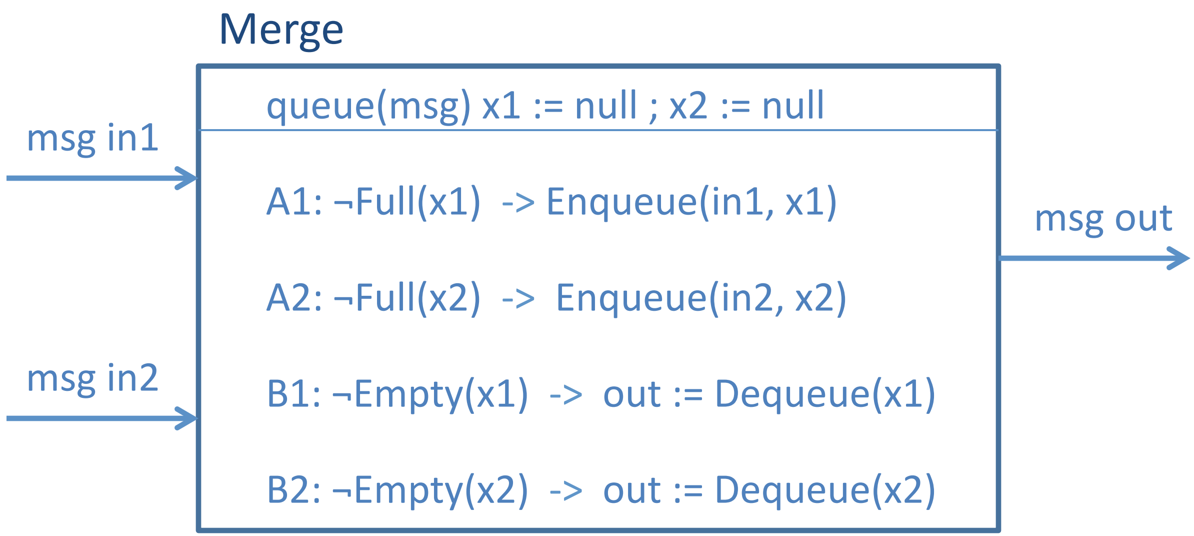
    ($Guard_1$ & $Guard_2$) -> $Update_1$; $Update_2$

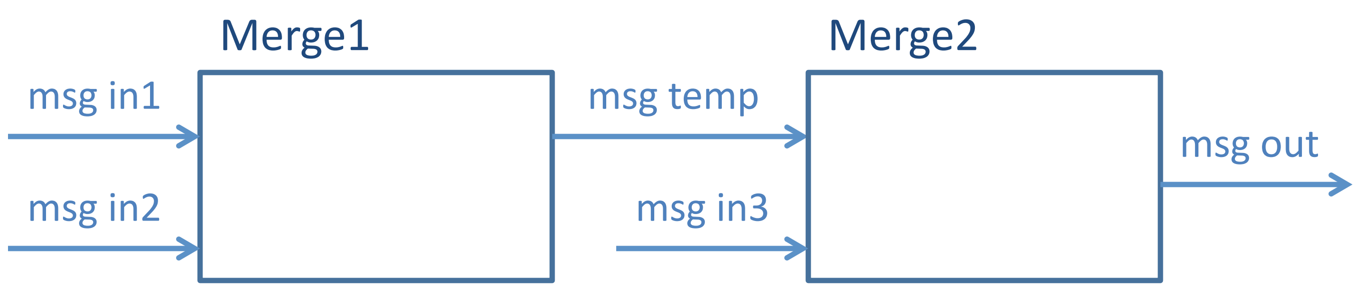
Execution Model: Another View



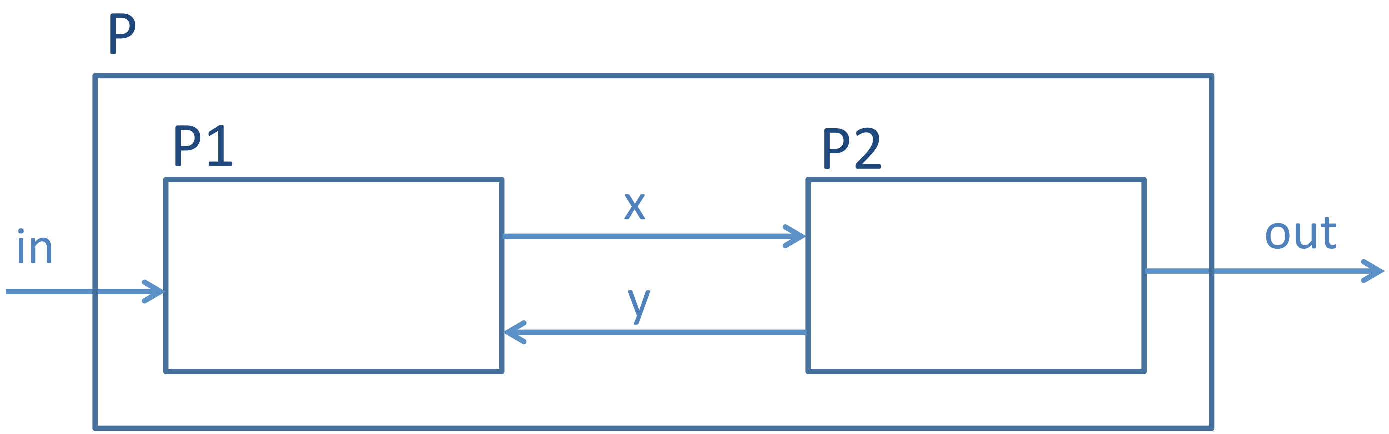
- A single step of execution
  - Execute an internal task of one of the processes, or
  - Process input on an external channel x: execute an input task for x of every process to which x is an input, or
  - Execute an output task for an output y of some process, followed by an input task for y for every process to which y is an input
- If multiple enabled choices, choose one non-deterministically
  - No constraint on relative execution speeds

Asynchronous Merge





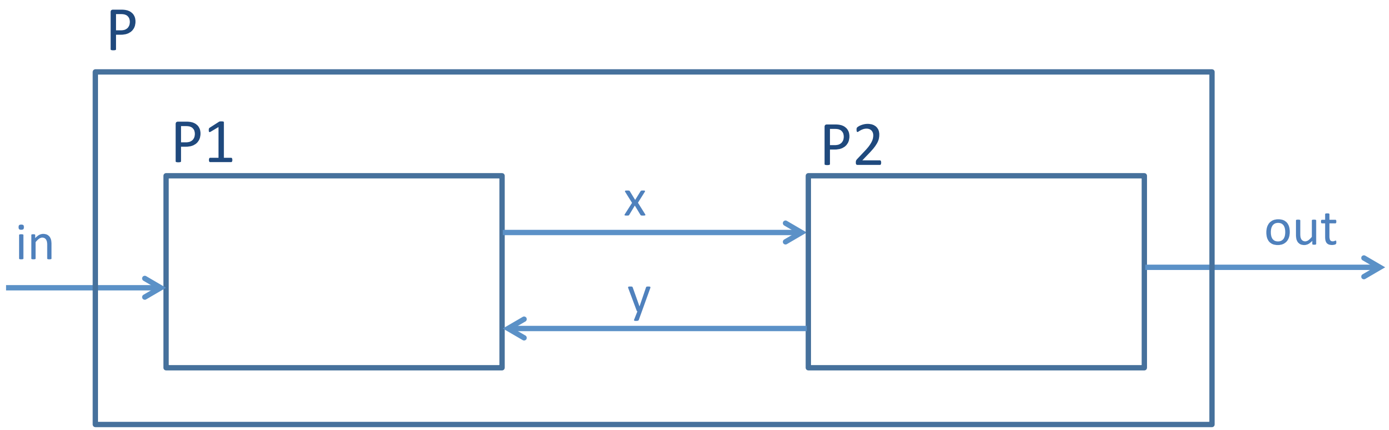
Asynchronous Execution



What can happen in a single round of this asynchronous model P?

- P1 synchronizes with the environment to accept input on in
- P2 synchronizes with the environment to send output on out
- P1 performs some internal computation (one of its internal tasks)
- P2 performs some internal computation (one of its internal tasks)
- P1 produces output on x, followed by its immediate consumption by P2
- P2 produces output on y, followed by its immediate consumption by P1

Asynchronous Execution



- **Note.** Interprocess communication is *blocking*: if no task of P2 associated with x is enabled in a round then P1 cannot write to x in that round
- A process P is *non-blocking* if for every input channel x and state s of P, some task of P associated with x is enabled in state s
- In designs with non-blocking processes, a receiving process is often expected to send an acknowledgement back to the sender of a message m that it did receive m

Credits

Notes based on Chapter 4 of

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