Formal Verification of Distributed Branching Multiway Synchronization Protocols

Hugues EVRARD and Frédéric LANG

Inria Grenoble – Team CONVECS

Introduction

- Distributed System design is complex
  → Formal Methods can help!

- Formal specification **help finding design bugs early**, e.g., using process algebra and model checking
- But… **semantic gap between formal specs and implementation**

Introduction

- Distributed System design is complex
  → Formal Methods can help!

- Formal specification **help finding design bugs early**, e.g., using process algebra and model checking
- But… **semantic gap between formal specs and implementation**

- Automatic distributed code generation is a solution we want to implement
- A distributed implementation requires **synchronization protocols**

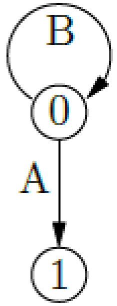
This Talk

Focus on the **correctness** of existing synchronization protocols

- Study of 3 protocols selected from the literature
- Formal specification in the language LNT
- Correctness verification using the toolbox CADP

The Specification Language LNT

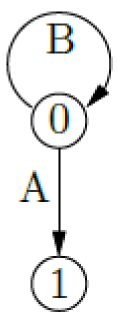
- Short for LOTOS NT, also inspired by E-LOTOS
- Process Algebra, with rendezvous on gates (actions)
- Labeled Transition System (LTS) semantics



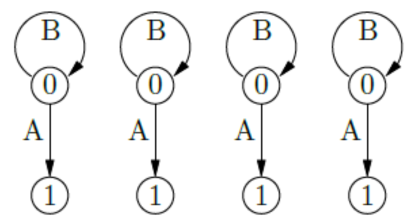

```
process p[A,B] is
loop
   select
      A ; stop
   []
      B
   end select
end loop
end process
```

The Specification Language LNT

- Short for LOTOS NT, also inspired by E-LOTOS
- Process Algebra, with rendezvous on gates (actions)
- Labeled Transition System (LTS) semantics
- Parallel composition operator more expressive than LOTOS (m-among-n synchronizations)



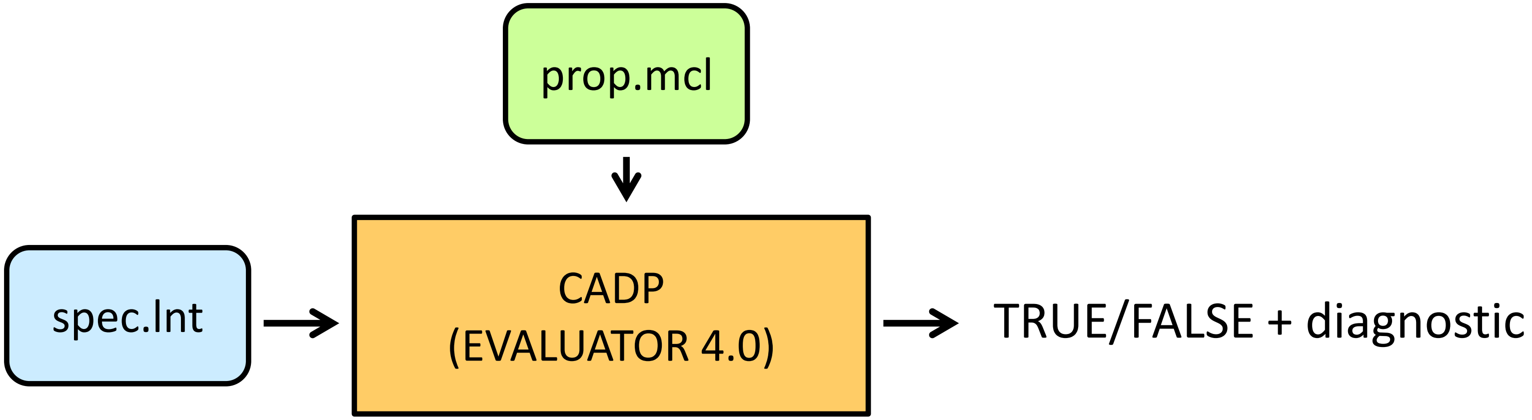

```
process p[A,B] is
loop
   select
      A ; stop
   []
      B
   end select
end loop
end process
```




```
process
   main[A,B]
is
par A#2, B in
   p[A,B]
|| p[A,B]
|| p[A,B]
|| p[A,B]
end par
end process
```

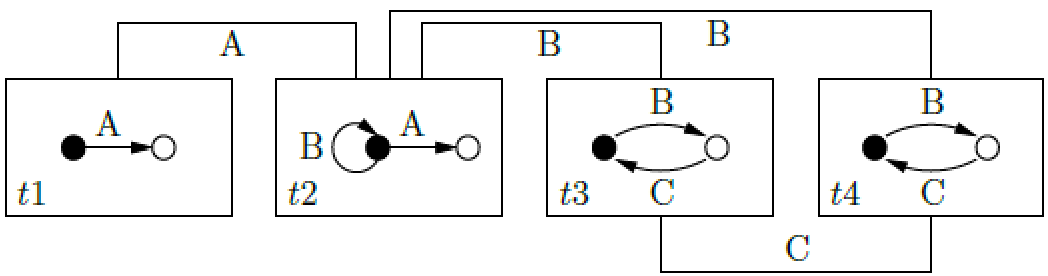
The Verification Toolbox CADP

- *Construction and Analysis of Distributed Processes*
  - Developed by Inria CONVECS (formerly VASY)
- Supports LNT specifications (among others)
  - Model Checker EVALUATOR 4.0 for MCL temporal logic
  - Equivalence Checker BISIMULATOR
  - …more tools, see cadp.inria.fr



Distributed System: a model

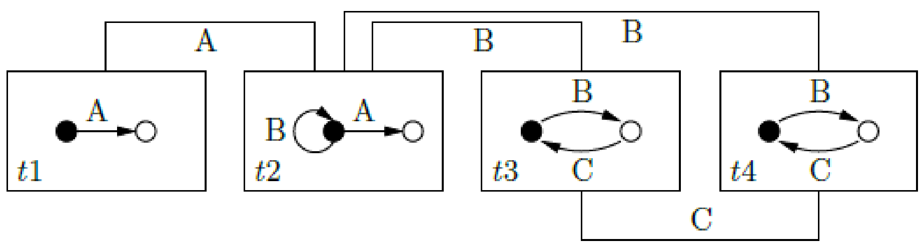
- Task specified as Labeled Transition System (LTS)
- Asynchronous execution
- Interaction by synchronization on ***gates*** (label names)
- General model of synchronization: multiway and non-deterministic



A **synchronization scenario** with 4 tasks synchronizing on 3 gates

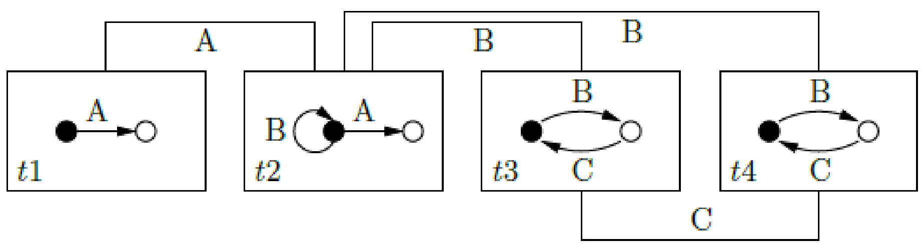
From Model to Implementation

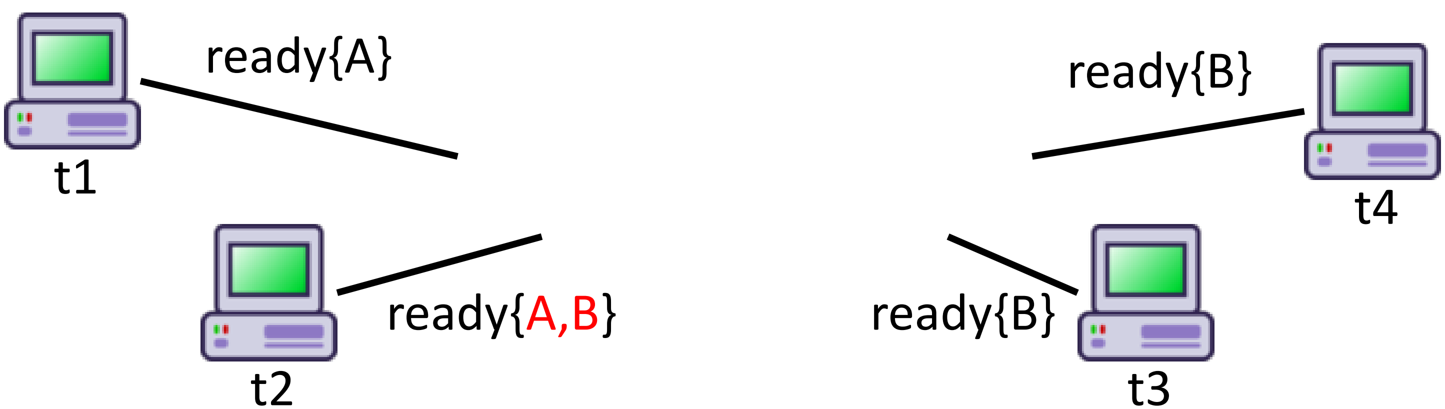
- Target: *Distributed* Implementation
- Each Task becomes a local, sequential process



From Model to Implementation

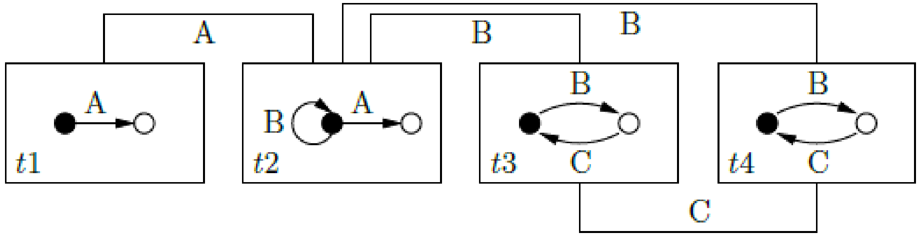
- Target: *Distributed* Implementation
- Each Task becomes a local, sequential process
- Tasks are branching: they may be ready on several gates a the same time (e.g. t2)

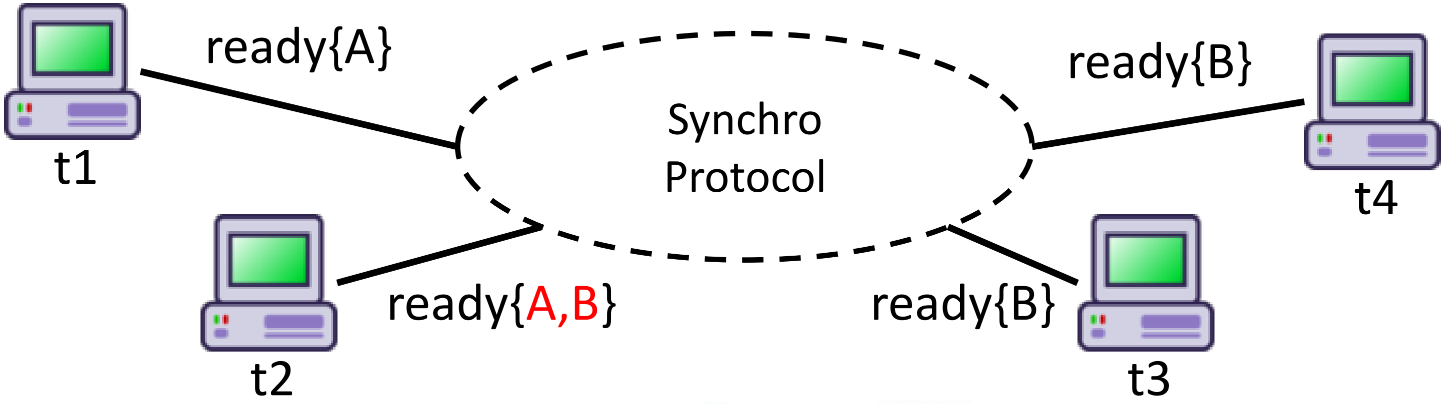




From Model to Implementation

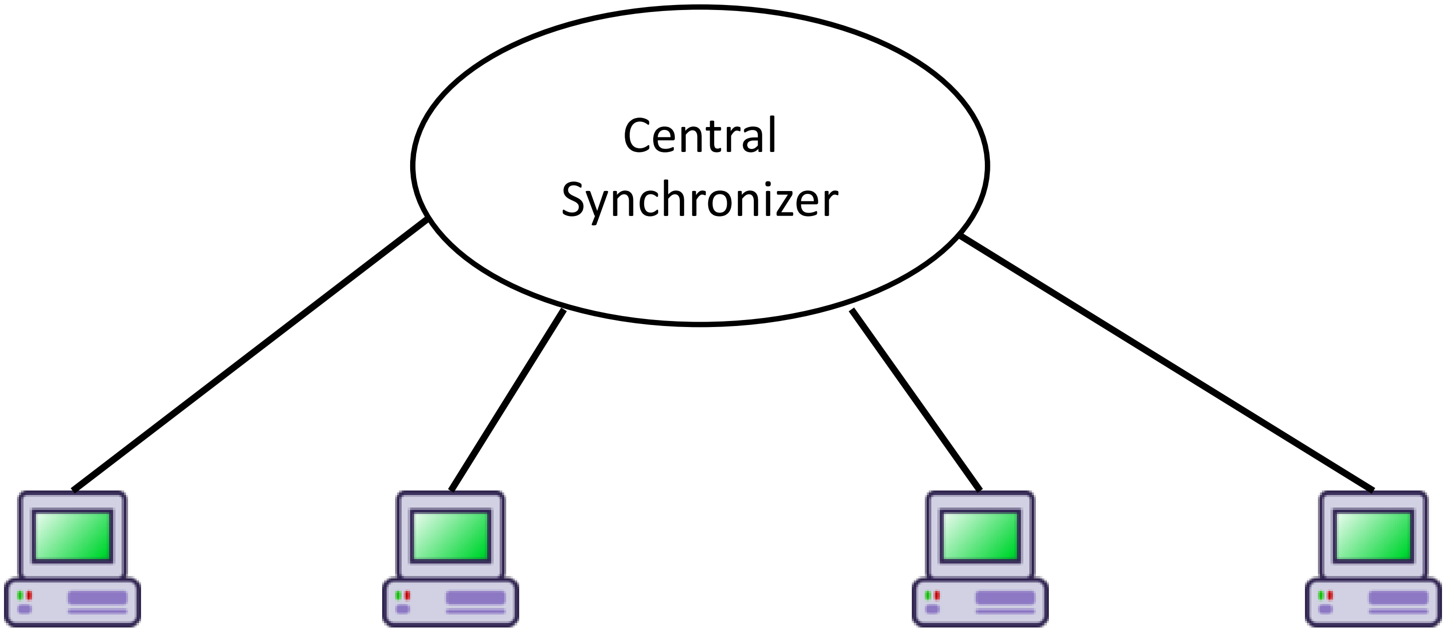
- Target: *Distributed* Implementation
- Each Task becomes a local, sequential process
- Tasks are branching: they may be ready on several gates a the same time (e.g. t2)
- A protocol is needed for task synchronizations





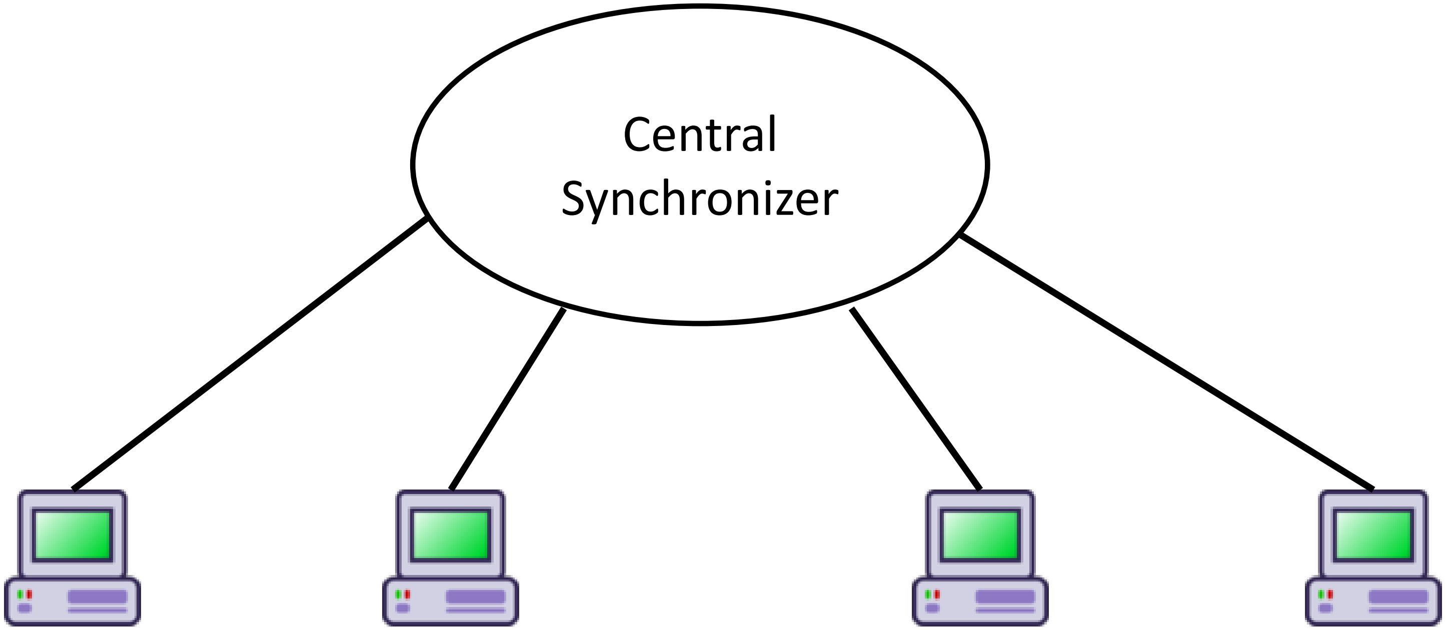
Naïve Solution

- Unique **central synchronizer**
- Knows all ready tasks, select possible synchros



Naïve Solution

- Unique **central synchronizer**
- Knows all ready tasks, select possible synchros
- Obvious bottleneck ☹
- … need distributed protocols!



Protocols under study

- Sisto, Ciminiera & Valenzano (1991)
  - *A protocol for multirendezvous of LOTOS processes*
- Sjödin (1991)
  - *From LOTOS Specifications to Distributed Implementations* (PhD Thesis)
- Parrow & Sjödin (1996)
  - *Designing a multiway synchronization protocol*

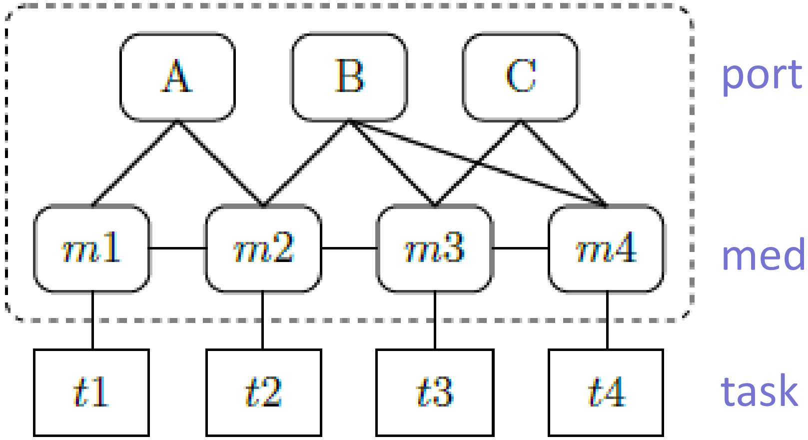
Protocols under study

- Sisto, Ciminiera & Valenzano (1991)
  - *A protocol for multirendezvous of LOTOS processes*
- Sjödin (1991)
  - *From LOTOS Specifications to Distributed Implementations* (PhD Thesis)
- **Parrow & Sjödin (1996)**
  - ***Designing a multiway synchronization protocol***

Parrow's Protocol

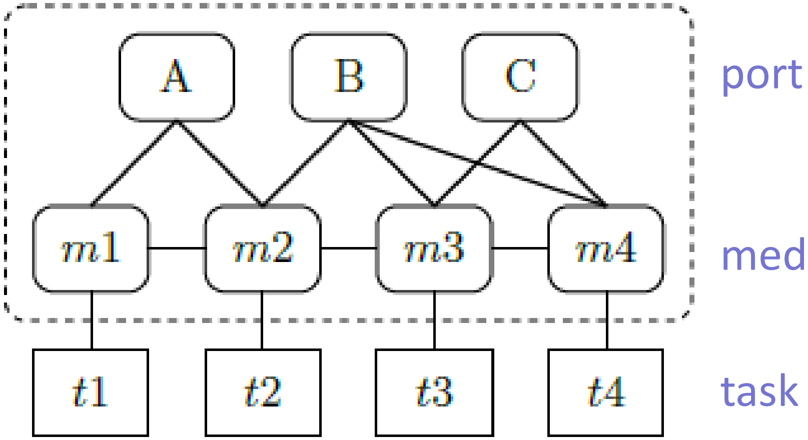
- **Ports** represent gates
- **Mediators** are attached to Tasks

- Negotiation:
  - Task sends **request** to Mediator
  - Mediator sends **ready** to Ports



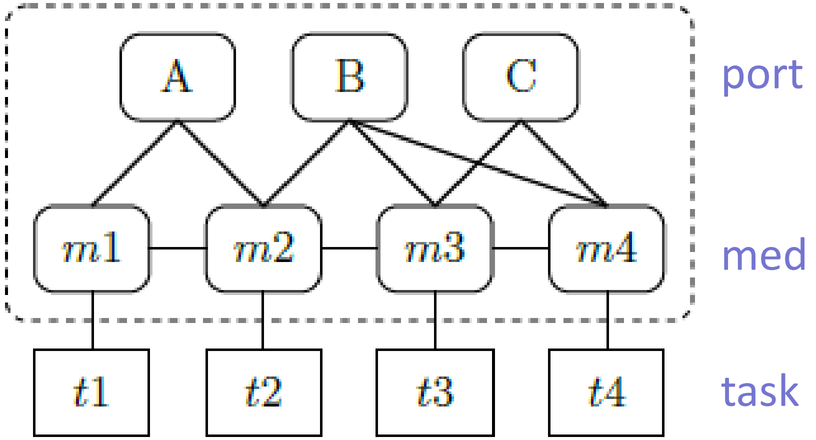
Parrow's Protocol

- **Ports** represent gates
- **Mediators** are attached to Tasks
- Negotiation:
  - Task sends **request** to Mediator
  - Mediator sends **ready** to Ports
  - Ports send **query** to 1st Mediator
  - Mediators propagate **locks** in order



Parrow's Protocol

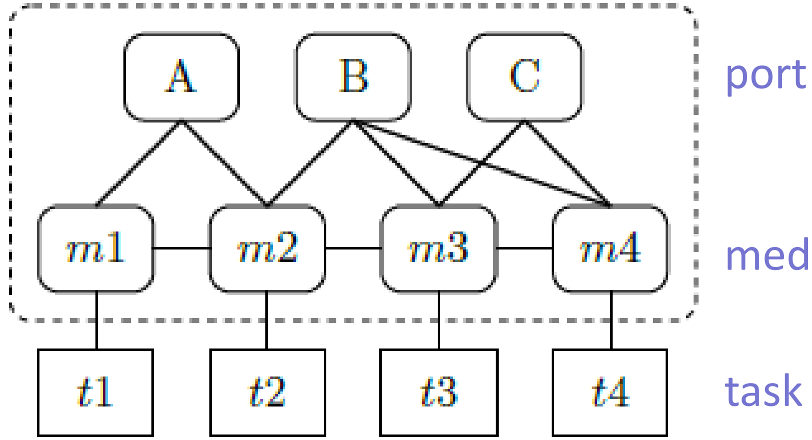
- **Ports** represent gates
- **Mediators** are attached to Tasks



- Negotiation:
  - Task sends **request** to Mediator
  - Mediator sends **ready** to Ports
  - Ports send **query** to 1st Mediator
  - Mediators propagate **locks** in order
  - Last Mediator replies **yes** to Port, **commit** to Mediators
  - Mediators propagate **commit**, and **confirm** their Task

Parrow's Protocol

- **Ports** represent gates
- **Mediators** are attached to Tasks

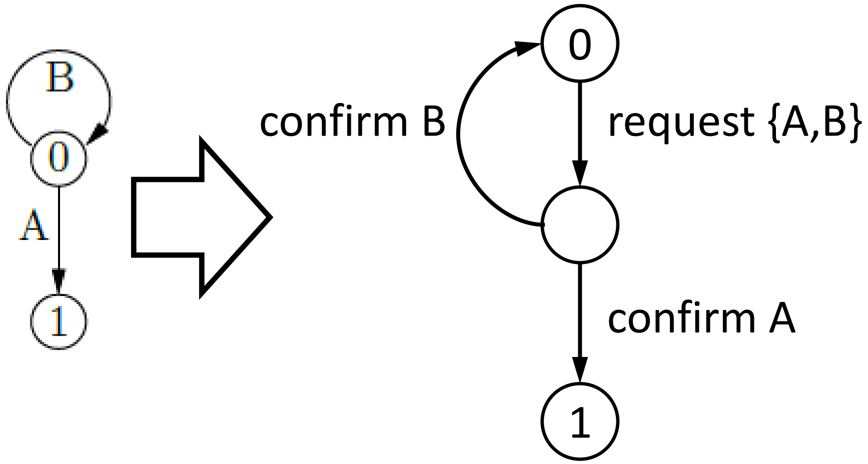


- Negotiation:
  - Task sends **request** to Mediator
  - Mediator sends **ready** to Ports
  - Ports send **query** to 1st Mediator
  - Mediators propagate **locks** in order
  - Last Mediator replies **yes** to Port, **commit** to Mediators
  - Mediators propagate **commit**, and **confirm** their Task

- Plus **abort** mechanism

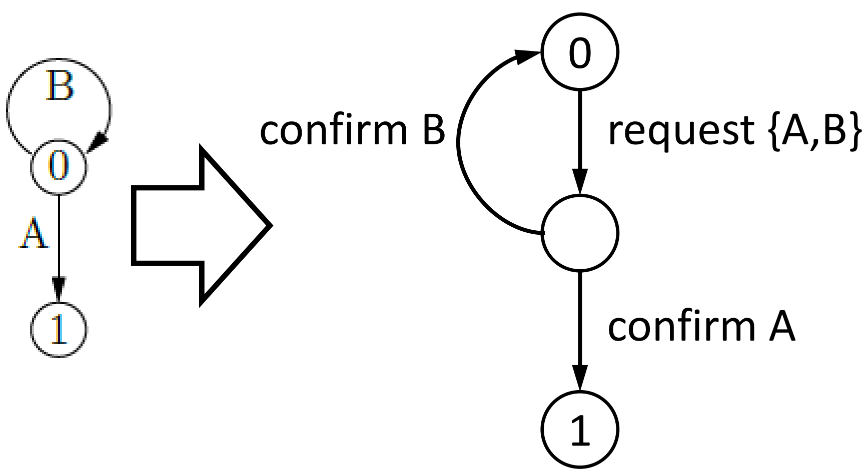
Task Model

- Task behavior: LTS
- Model:
  - send **request** message with set of gates
  - wait for a **confirm** answer with successful synchronization



Task Model

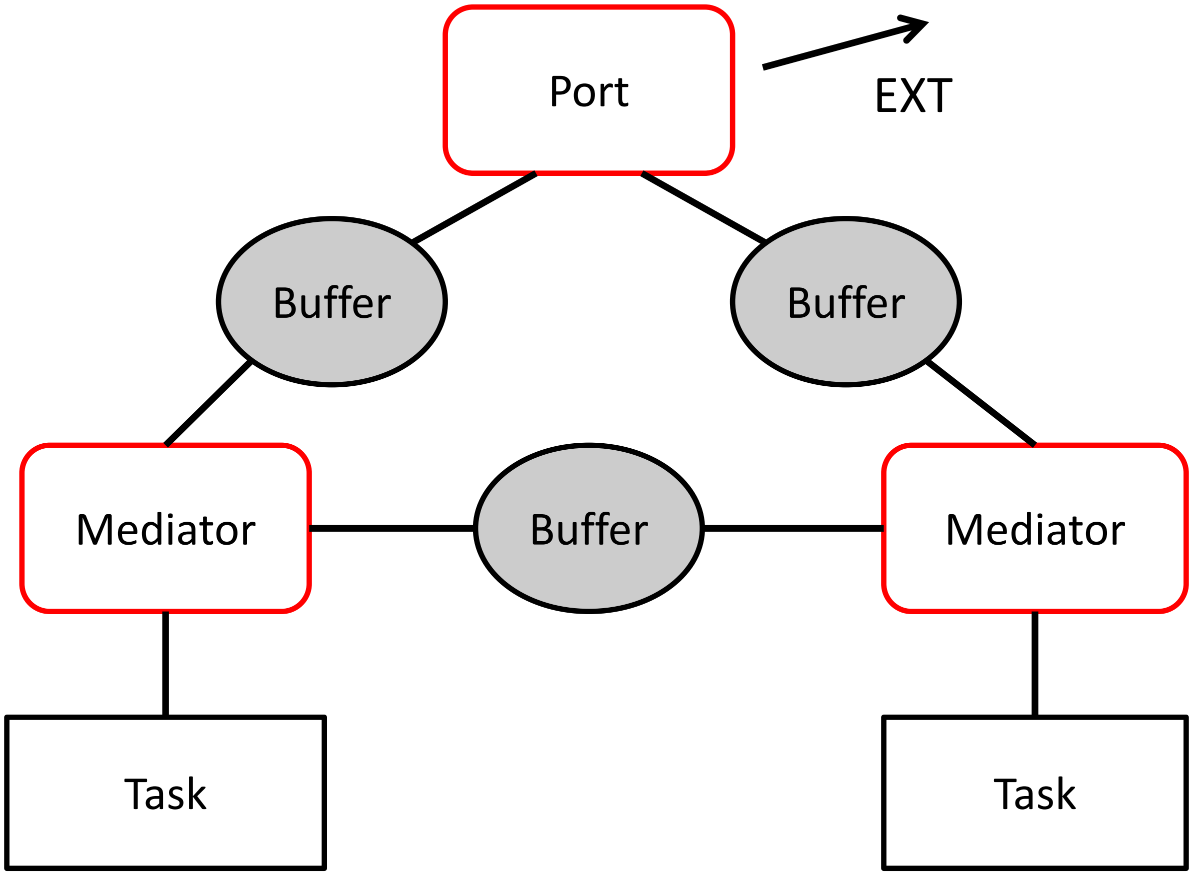
- Task behavior: LTS
- Model:
  - send **request** message with set of gates
  - wait for a **confirm** answer with successful synchronization



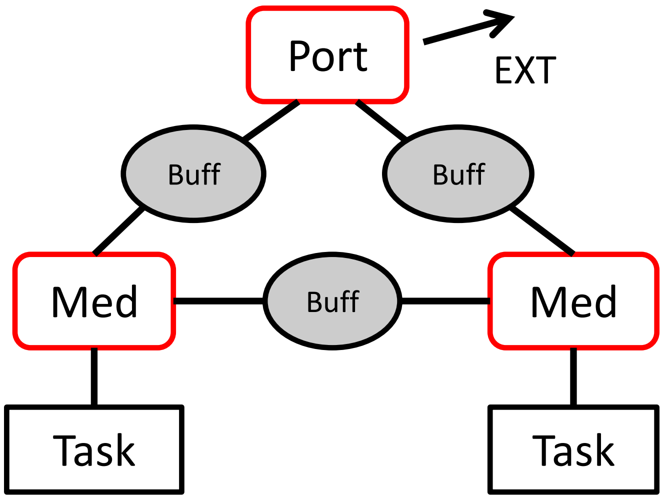

```
process task_t2 [M: msg_channel] is
   var sync_gate: gate in
      M(request, {A, B});
      M(confirm, ?sync_gate);
      case sync_gate in
         A -> task_t2_1[M]
      |  B -> task_t2[M]
      end case
   end var
end process

process task_t2_1 [M: msg_channel] is
   stop
end process
```

Protocol Model Overview

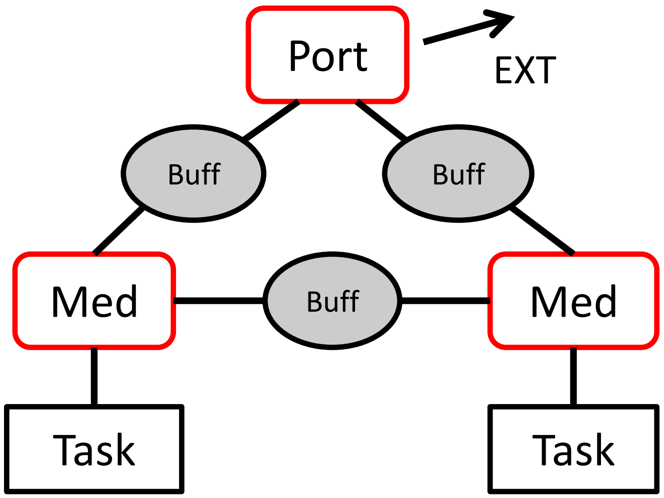


Protocol Model



- Mediators & Ports
  - Behavior specified in original publication
  - Write corresponding processes in LNT
  - Arguments (e.g. port: possible synchros on its gate)

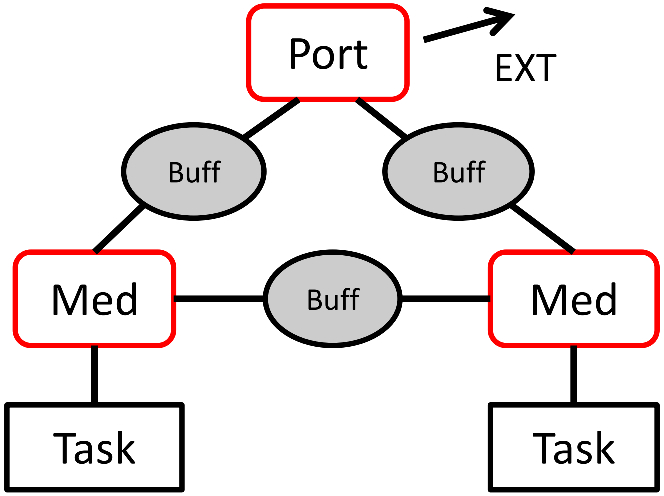
Protocol Model



- Mediators & Ports
  - Behavior specified in original publication
  - Write corresponding processes in LNT
  - Arguments (e.g. port: possible synchros on its gate)

- Inter-Process Communication
  - Asynchronous message passing: buffers on channels

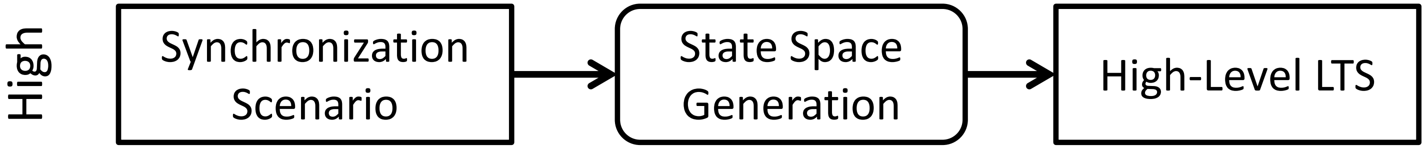
Protocol Model



- Mediators & Ports
  - Behavior specified in original publication
  - Write corresponding processes in LNT
  - Arguments (e.g. port: possible synchros on its gate)
- Inter-Process Communication
  - Asynchronous message passing: buffers on channels
- Trace successful synchronizations
  - Message on EXT (External World)

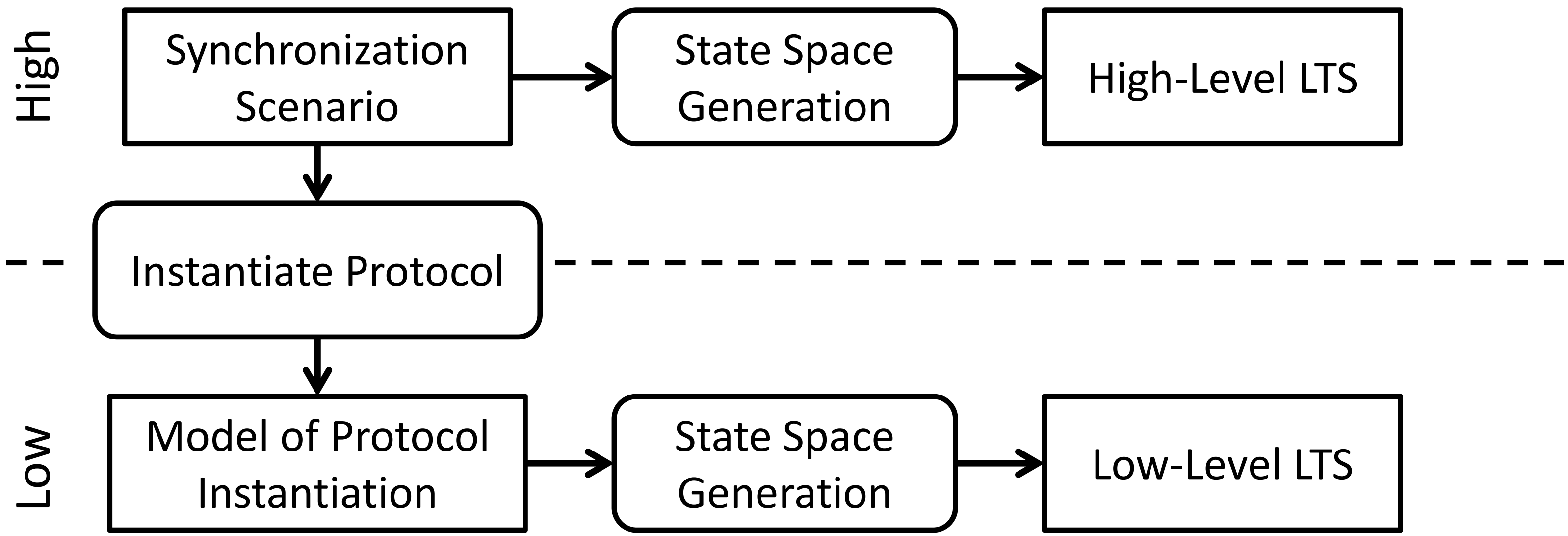
High-Level and Low-Level LTS

High-level LTS: all possible task synchronizations w.r.t. scenario



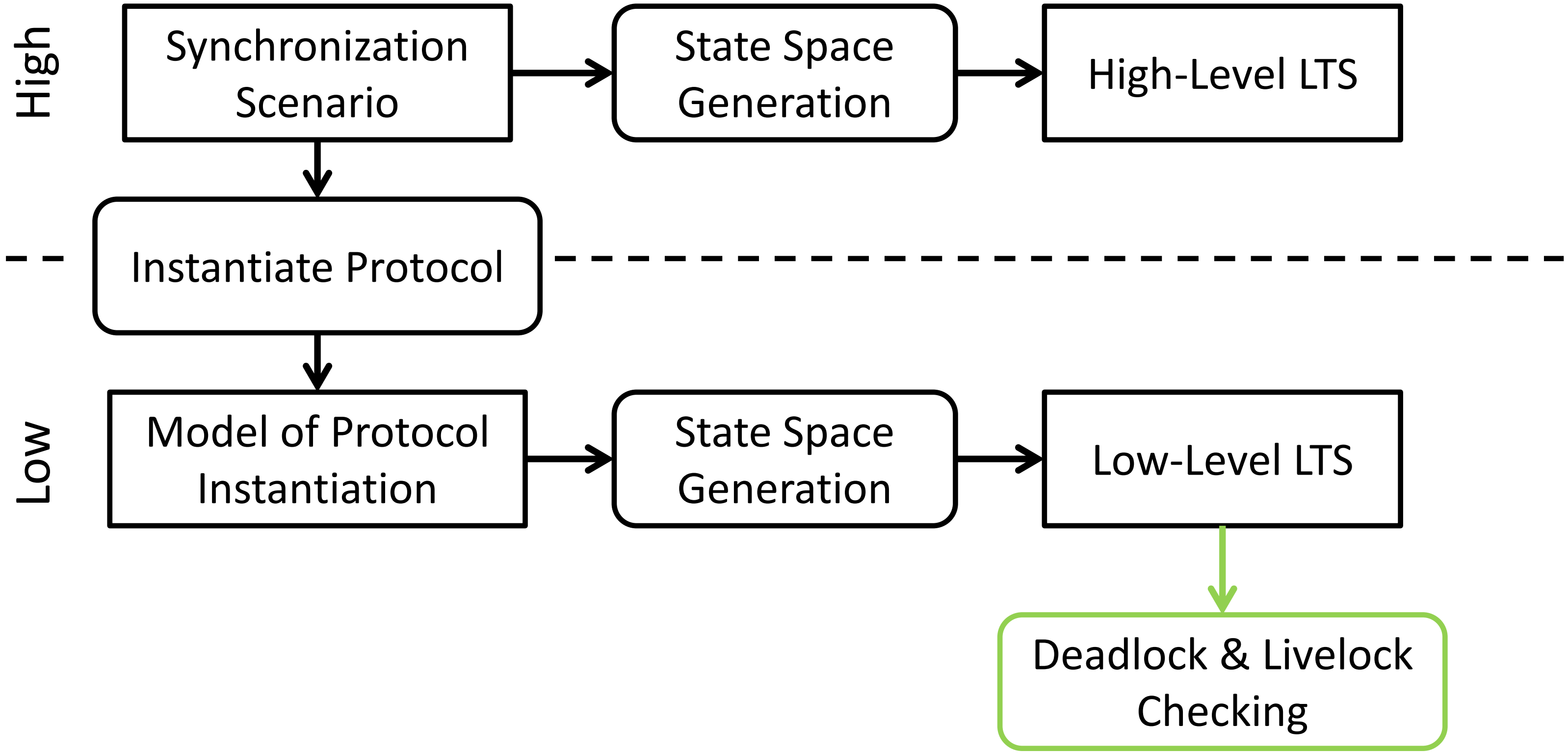
High-Level and Low-Level LTS

High-level LTS: all possible task synchronizations w.r.t. scenario



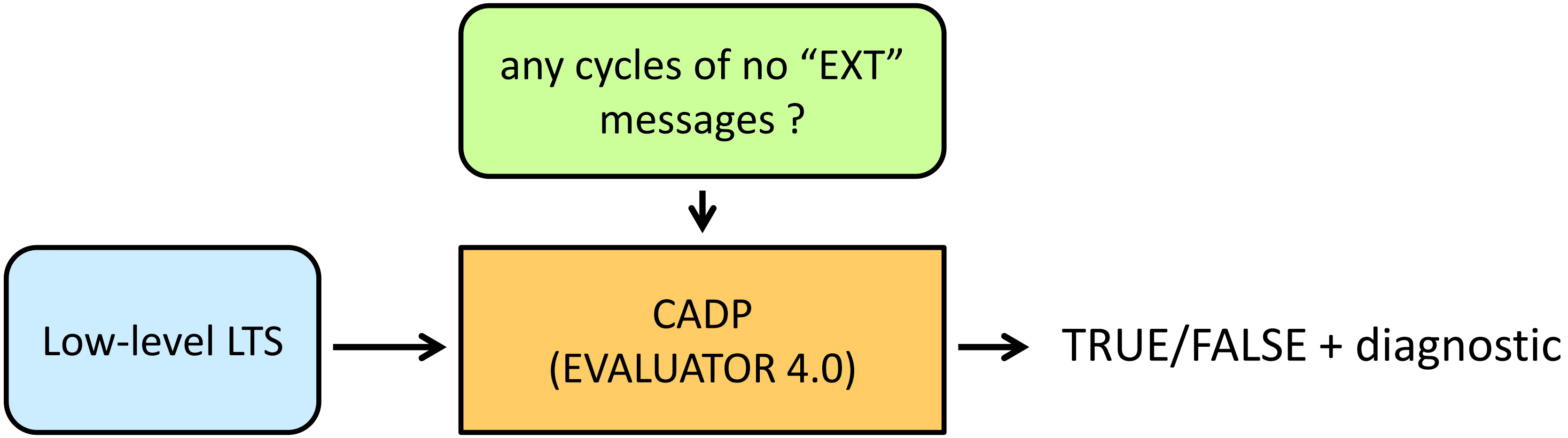
Low-level LTS: all possible sequences of protocol message
(messages on EXT announce high-level synchro)

High-Level and Low-Level LTS



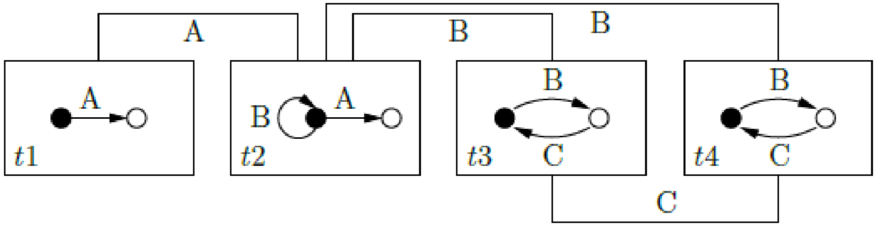
Livelock checking

- Infinite protocol message exchange without reaching a synchro
- Look for **cycle with no EXT message** in Low-level LTS
- Verified using EVALUATOR 4.0 on the LTS obtained from the protocol model.



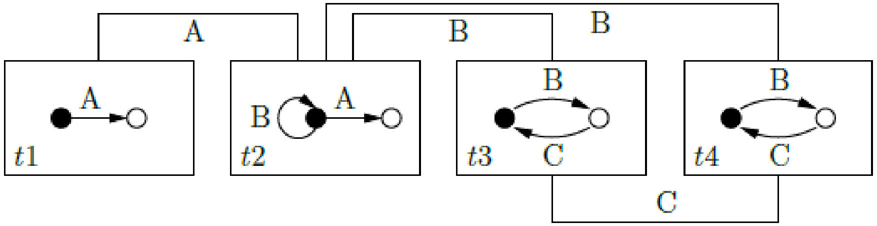
Deadlock checking

- Classic deadlock: a state with no outgoing transitions
  - Can be a High-level deadlock (e.g., after action A)



Deadlock checking

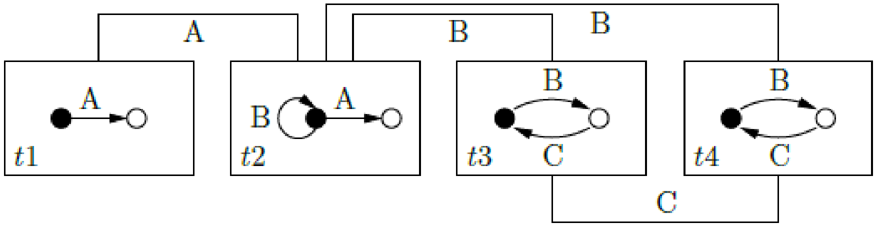
- Classic deadlock: a state with no outgoing transitions
  - Can be a High-level deadlock (e.g., after action A)



- Low-level deadlock == **triggered by the protocol**
  - Protocol is blocked after a sequence of messages, while a synchronization **could** have been reached

Deadlock checking

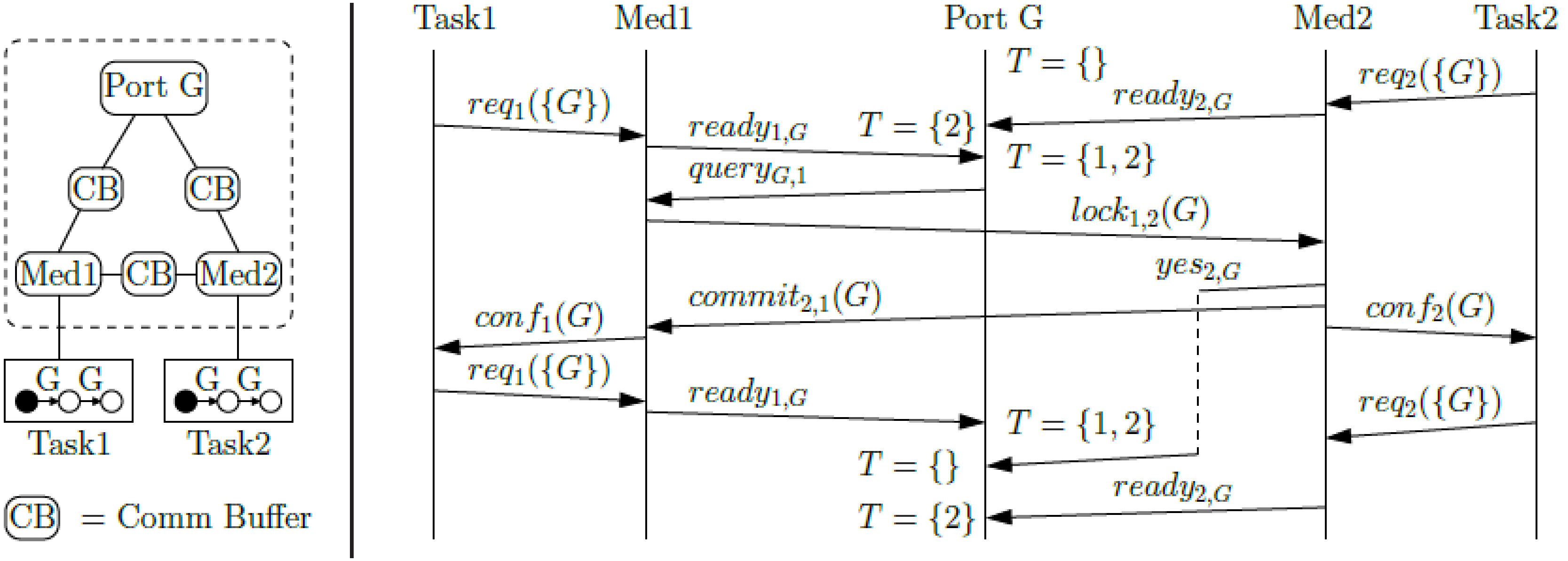
- Classic deadlock: a state with no outgoing transitions
  - Can be a High-level deadlock (e.g., after action A)



- Low-level deadlock == **triggered by the protocol**
  - Protocol is blocked after a sequence of messages, while a synchronization **could** have been reached
- In Low-level LTS, search for states from which there exists both:
  - a sequence leading to a classic deadlock with no EXT (protocol blocks with no synchronization)
  - a sequence which contains EXT (protocol may reach a synchro)
- If found, model checker EVALUATOR 4.0 provides an example

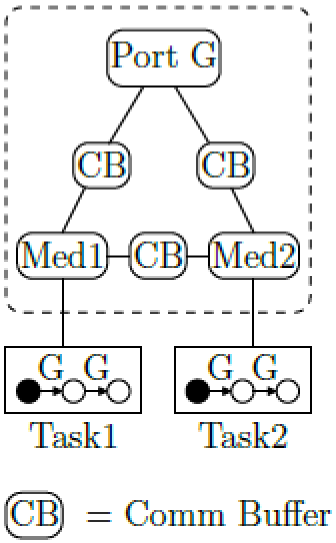
Deadlock in Parrow's protocol

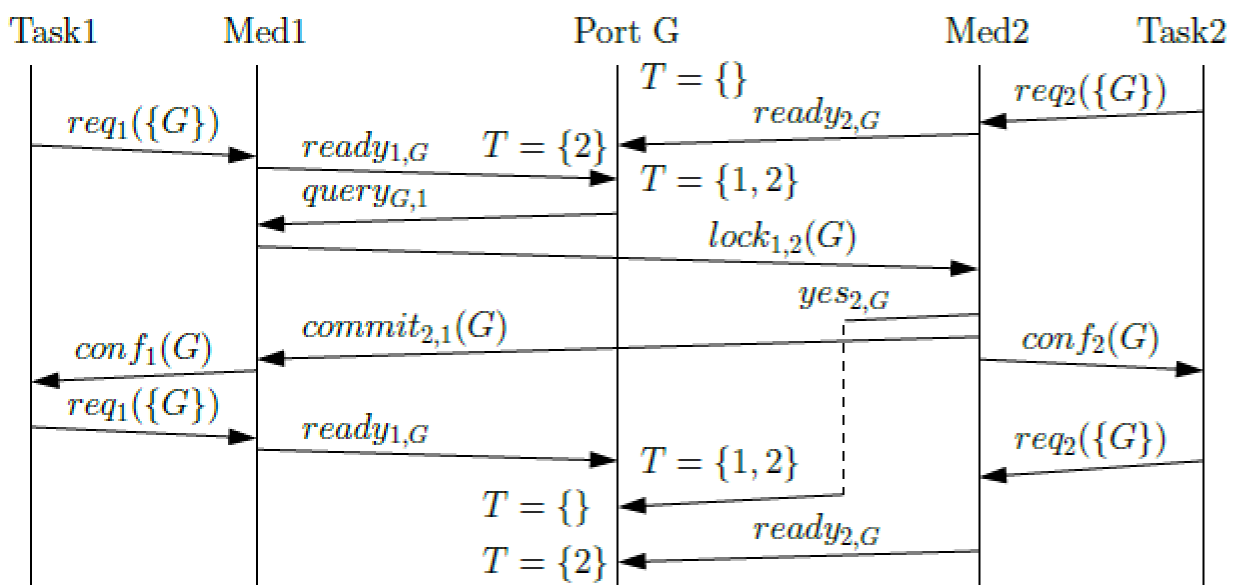
- Simple scenario: two similar tasks with two synchro on gate G:
  - on Port G, set T stores ready notifications



Deadlock in Parrow's protocol

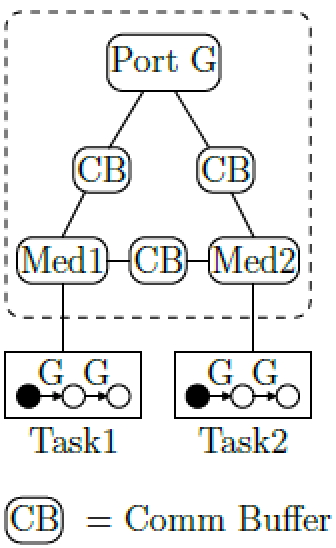
- Simple scenario: two similar tasks with two synchro on gate G:
  - on Port G, set T stores ready notifications
  - "yes" message is delayed (dashed line)

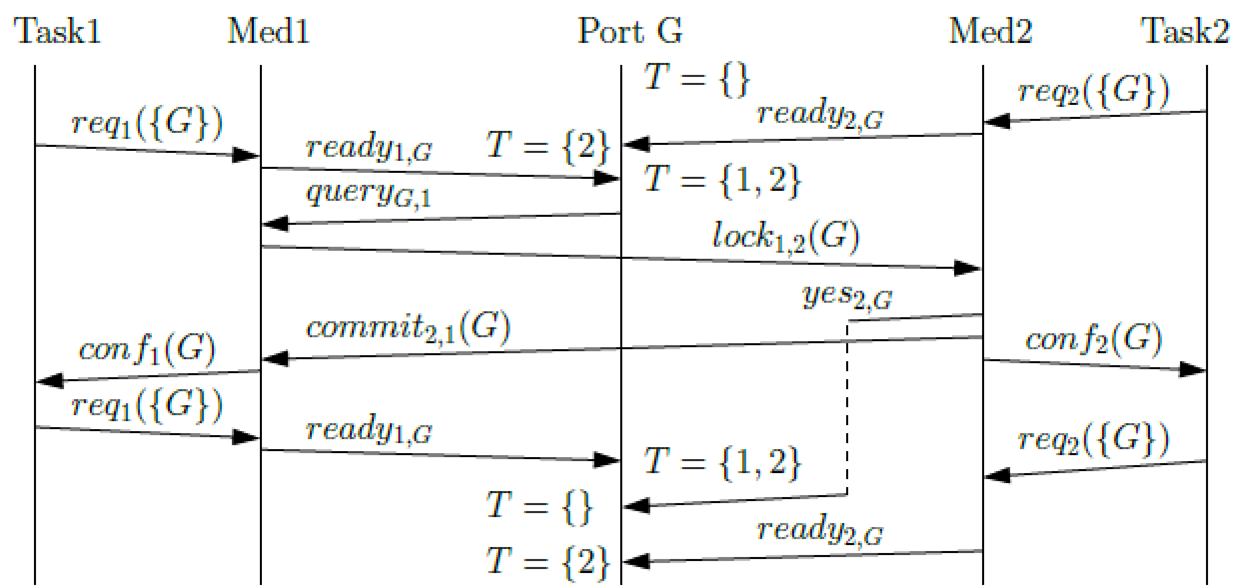




Deadlock in Parrow's protocol

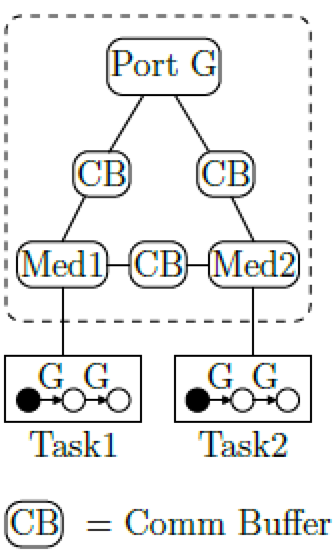
- Simple scenario: two similar tasks with two synchro on gate G:
  - on Port G, set T stores ready notifications
  - "yes" message is delayed (dashed line)
  - on reception of "yes", T := { }

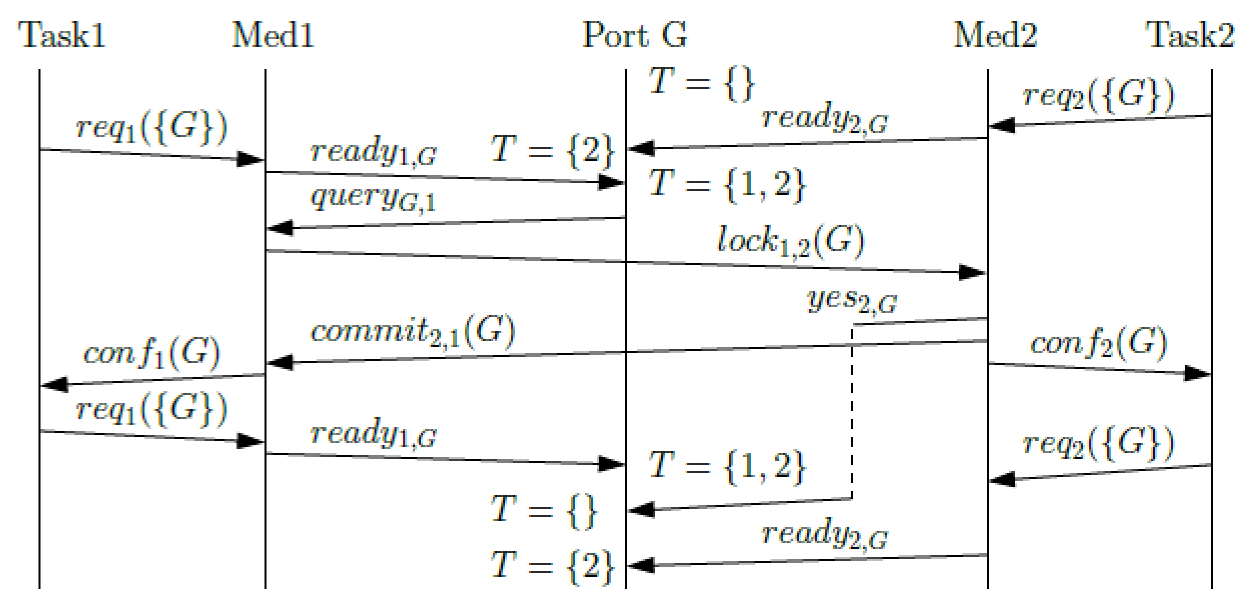




Deadlock in Parrow's protocol

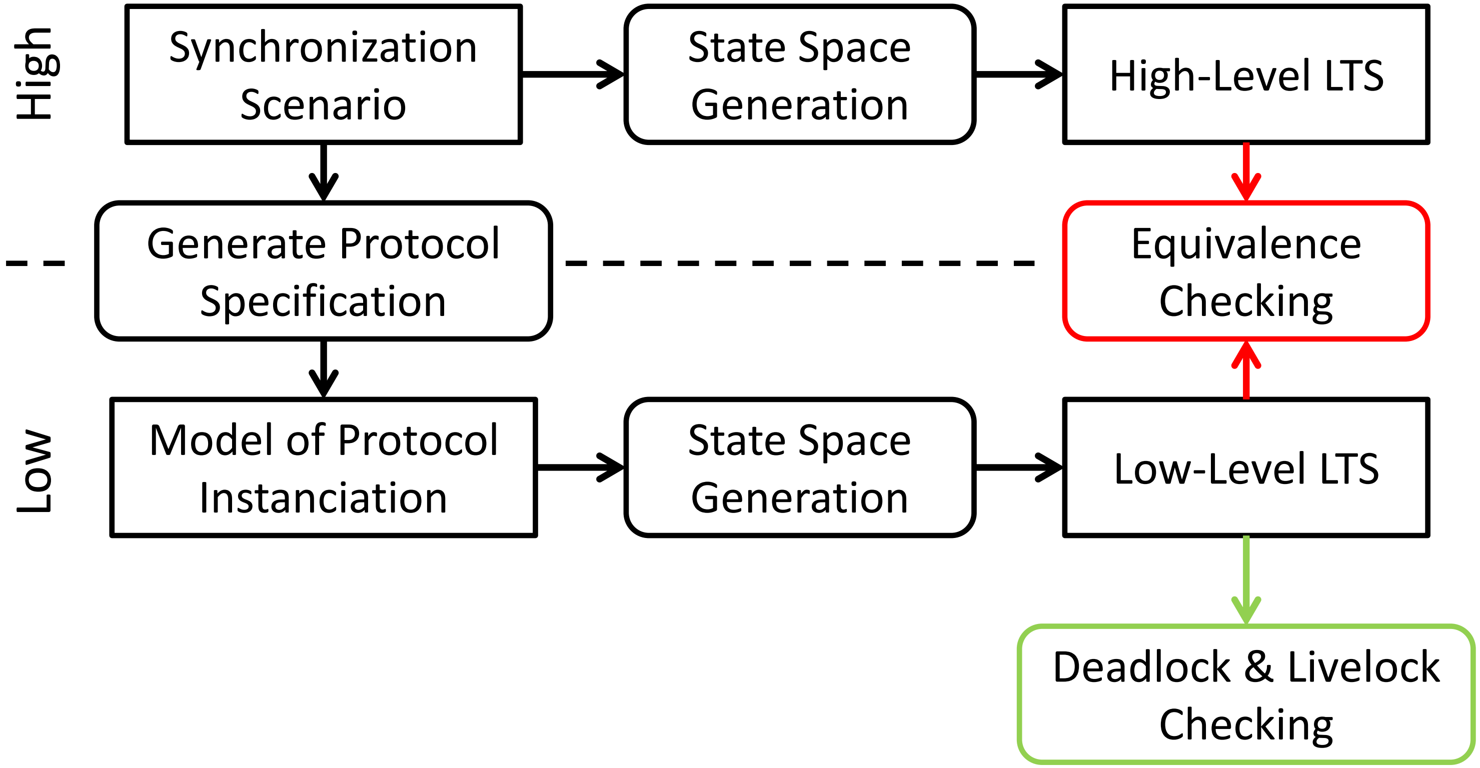
- Simple scenario: two similar tasks with two synchro on gate G:
  - on Port G, set T stores ready notifications
  - "yes" message is delayed (dashed line)
  - on reception of "yes", T := { }
  - no second synchro on G (OK if "yes" was received sooner by Port G…)
- High-level model is OK, this is a **protocol** deadlock





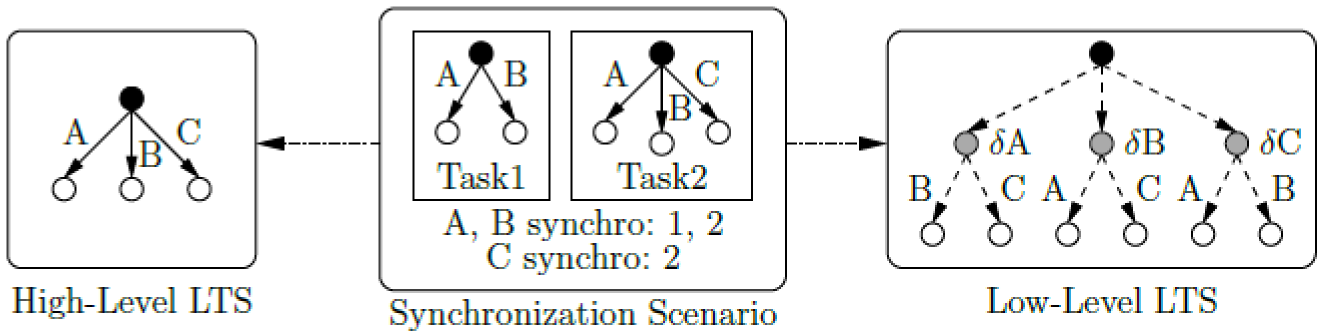
Synchronizations Consistency

- A protocol can be livelock- and deadlock-free…
- … but still not match synchronization semantics !



High and Low level LTS equivalence

- Compared using several equivalence relations
- Always have **safety** equivalence
- No **branching** bisimulation in some cases:



  - Three synchronizations are possible
  - Protocol negociation eliminates possible synchros step by step
  - Gray states in Low-level have no bisimilar state in High-level

Conclusion

- Complex synchronization protocols are required for automatic distributed implementation
- We modeled three protocols (LNT)
- We verified properties on 50+ scenarios (CADP)
- Some scenarios revealed possible deadlocks for one protocol
- Better understanding of difficulties of distributed synchronization

Conclusion

- Complex synchronization protocols are required for automatic distributed implementation
- We modeled three protocols (LNT)
- We verified properties on 50+ scenarios (CADP)
- Some scenarios revealed possible deadlocks for one protocol
- Better understanding of difficulties of distributed synchronization

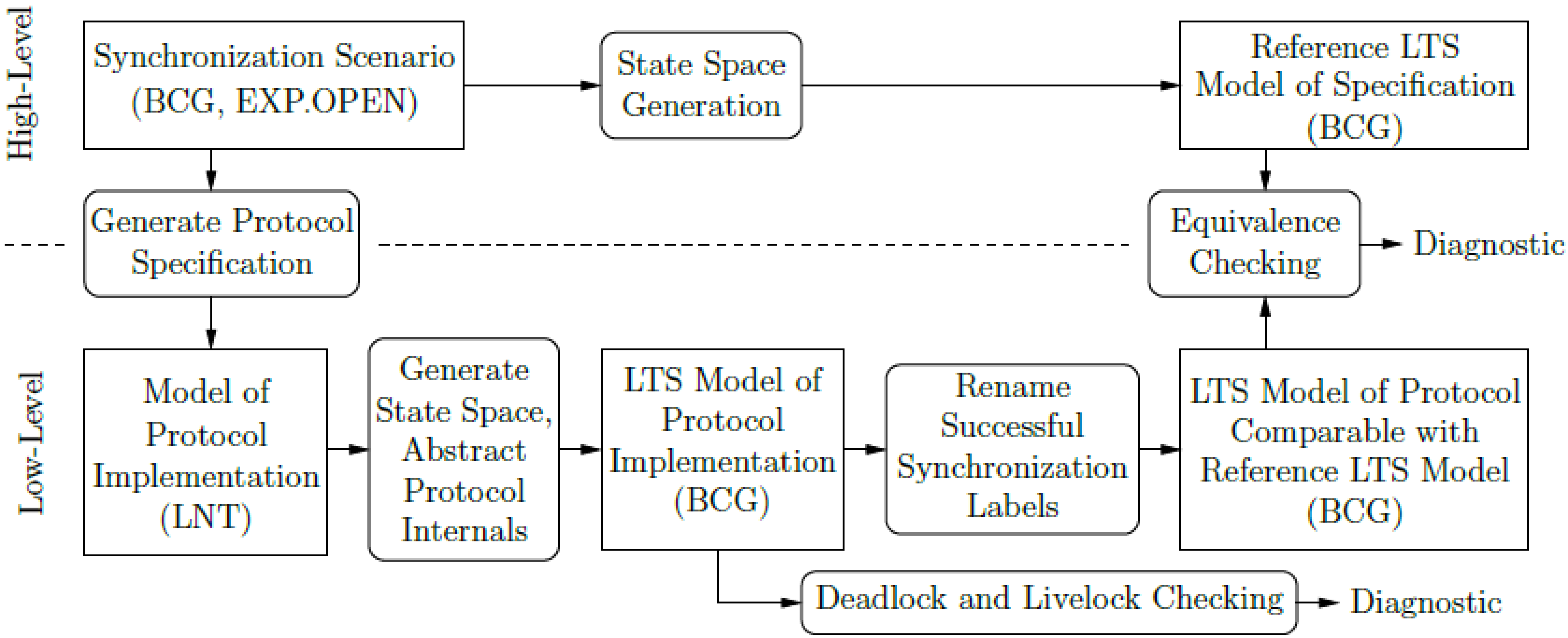
Future Work

- Reuse protocols models for **performance evaluation**
- **Rapid prototyping** of distributed systems from LNT specifications
- Consider **data exchange** at synchronization (with guards…)

Thank you for your attention

Questions?

Verification Overview



- High-level: original synchronization scenario (tasks + interactions)
- Low-level: protocol instantiation (task model, mediators, ports…)

Verification Overview (SVL)

- Generic script for any scenario

```
(* Generate low-level LTS *)
"raw_lowlevel.bcg" = generation of "main.lnt";
(* Hide protocol messages *)
"lowlevel.bcg" = hide all but "EXT.*" in "raw_lowlevel.bcg";

(* Model checking: livelock and deadlock *)
"diag_live.bcg" = livelock of "lowlevel.bcg";
"diag_dead.bcg" = verify "deadlock.mcl" in "lowlevel.bcg";

(* Generate reference LTS from high-level spec *)
"reference.bcg" = generation of "composition.exp";
(* Rename synchronization announcements *)
"renamed.bcg" = total rename "EXT !\(.*\)" -> "\1" in "lowlevel.bcg";

(* Equivalence checking: branching, safety, weaktrace *)
"diag_branching.bcg" = branching comparison "renamed.bcg" == "reference.bcg";
"diag_safety.bcg" = safety comparison "renamed.bcg" == "reference.bcg";
"diag_weaktrace.bcg" = weak trace comparison "renamed.bcg" == "reference.bcg";
```