Optimized Distributed Implementation of Multiparty Interactions with Observation

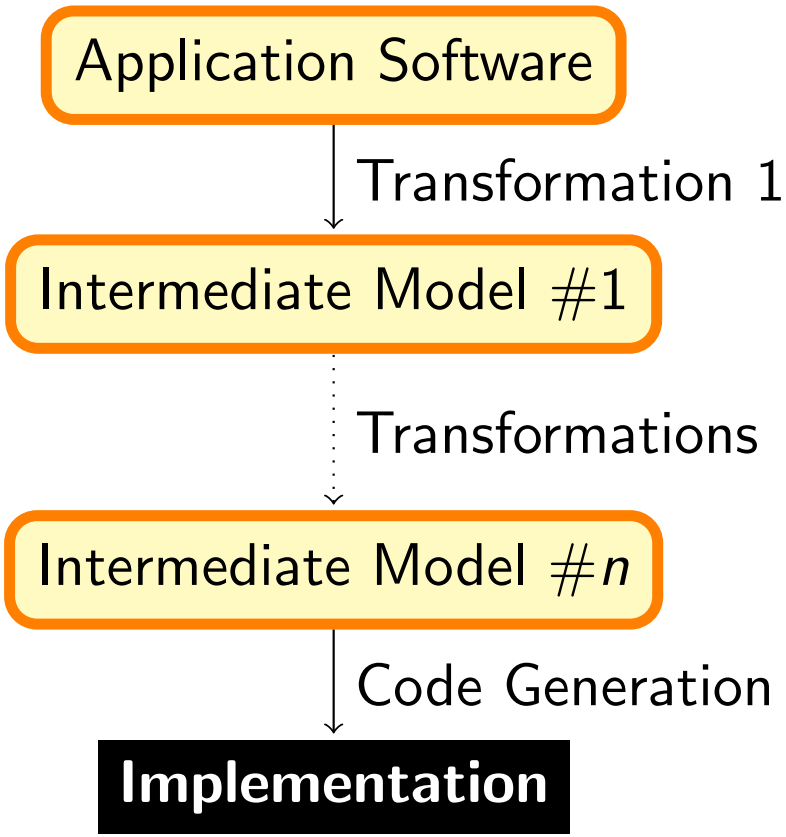
Saddek Bensalem, Marius Bozga, Jean Quilbeuf and Joseph Sifakis

Verimag

AGERE!12
21-22 October 2012
Tucson, Arizona, USA

Rigorous System Design

Sequence of transformations guaranteeing that essential requirements are met by the implementation.



Model-Based

- Unified semantic model
- Abstraction adapted to task (verification, code generation. . .)

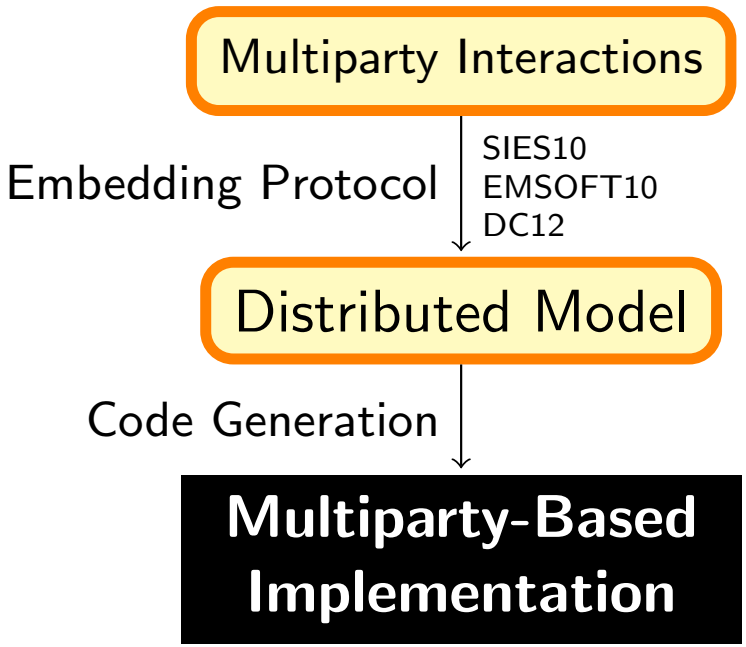
Component-Based

- Encompass heterogeneity
- Reusability

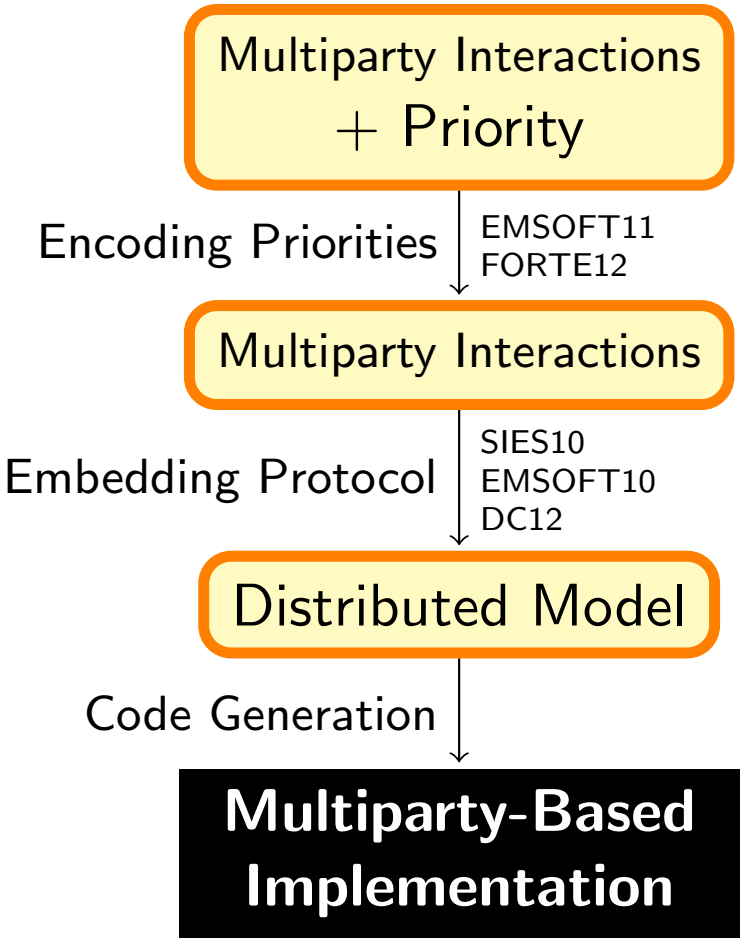
Correct-by-Construction

- Functional properties preserved by transformations
- Avoid a posteriori verification

Distributed BIP Design Flow



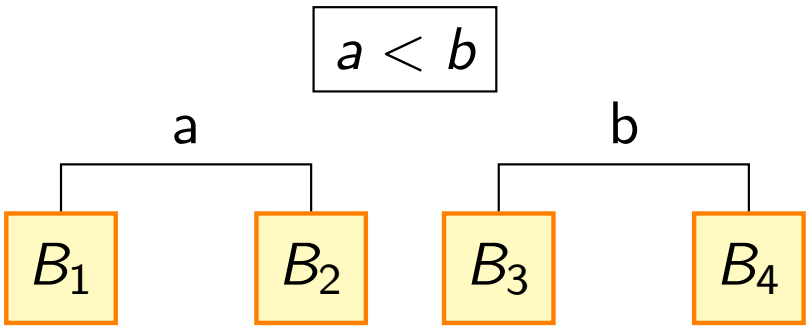
Distributed BIP Design Flow



Limits of Multiparty Interactions for encoding Priority

Assume components communicating through Multiparty Interactions.

Example of model with Priority:



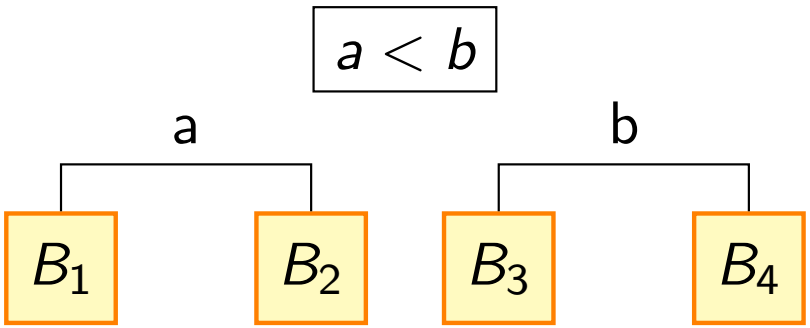
a can execute only if b cannot.

- Execution of a does not change the state of B_3 and B_4.
- Execution of a depends on the state of B_3 and B_4.

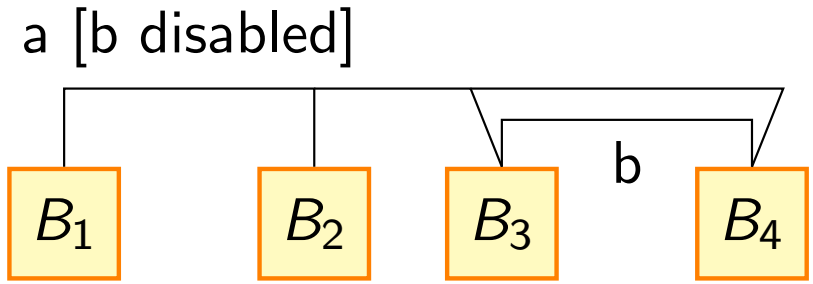
Limits of Multiparty Interactions for encoding Priority

Assume components communicating through Multiparty Interactions.

Example of model with Priority:



Same example encoded with Multiparty Interactions (and guard):



a can execute only if b cannot.

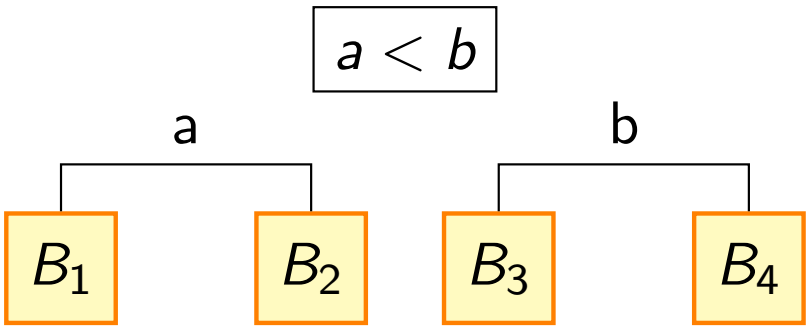
a checks that b cannot execute.

- Execution of a does not change the state of B_3 and B_4.
- Execution of a depends on the state of B_3 and B_4.

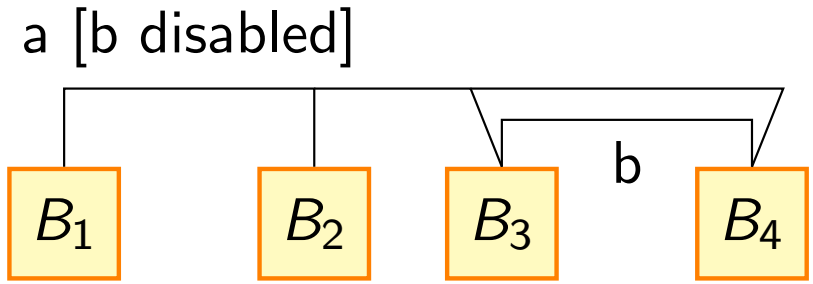
Limits of Multiparty Interactions for encoding Priority

Assume components communicating through Multiparty Interactions.

Example of model with Priority:



Same example encoded with Multiparty Interactions (and guard):



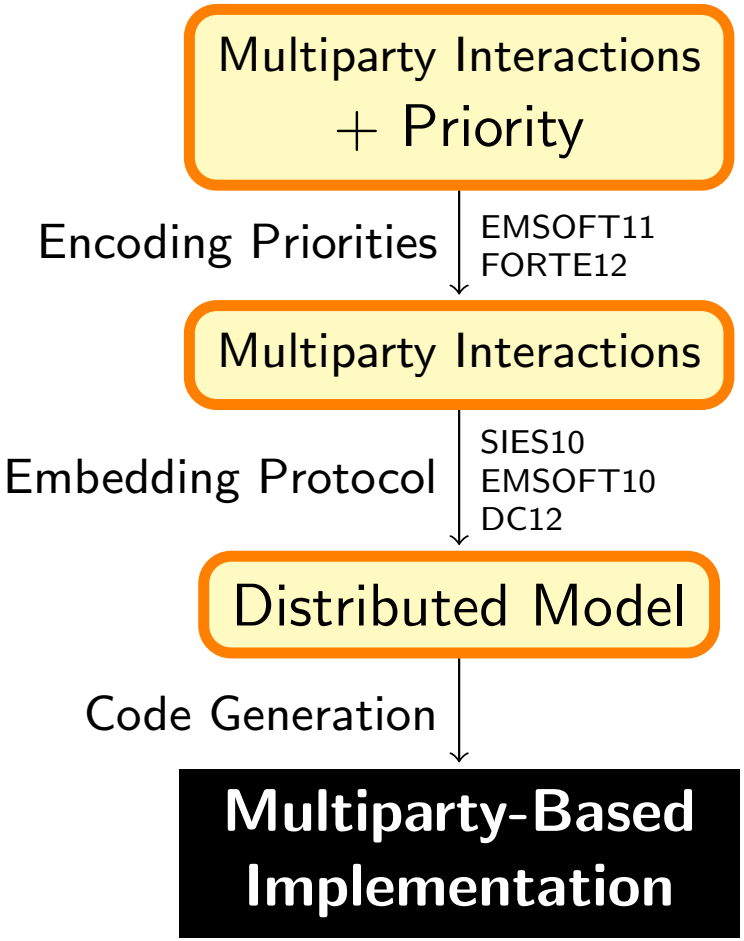
a can execute only if b cannot.

a checks that b cannot execute.

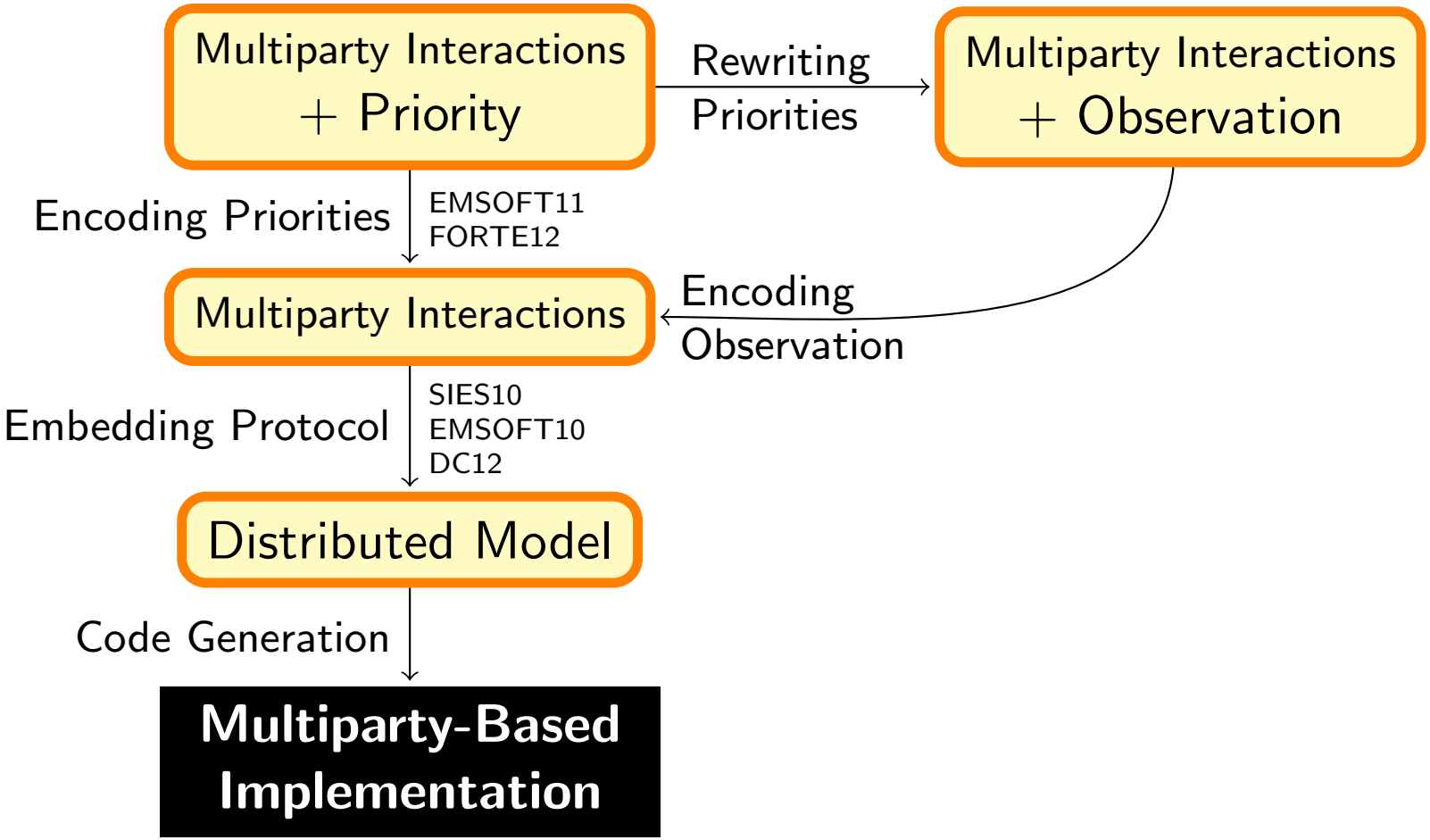
- Execution of a does not change the state of B_3 and B_4.
- Execution of a depends on the state of B_3 and B_4.

With Multiparty Interaction encoding, B_3 and B_4 become participants in a.
Need an intermediate between participation and non-participation.

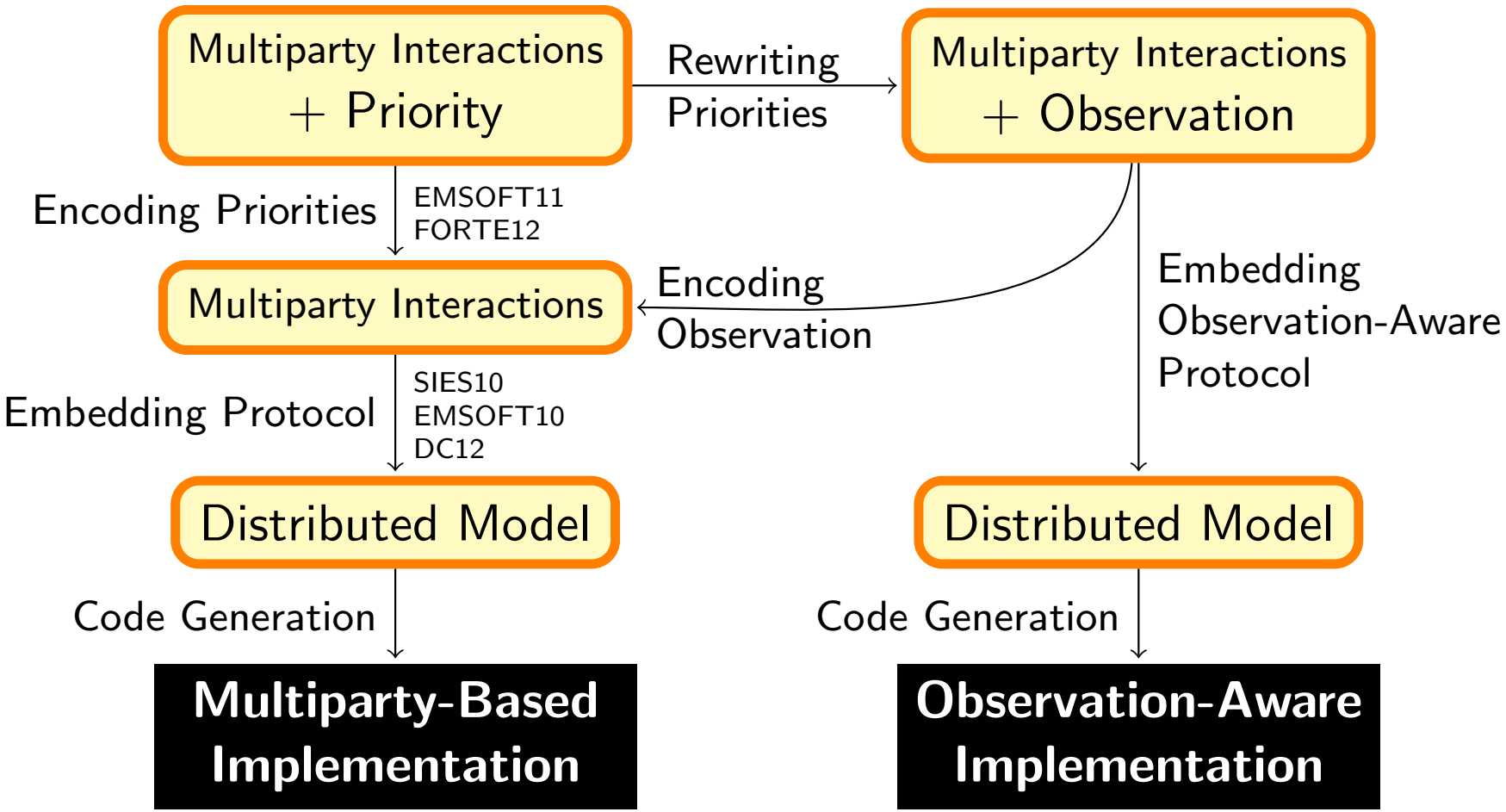
Distributed BIP Design Flow



Distributed BIP Design Flow

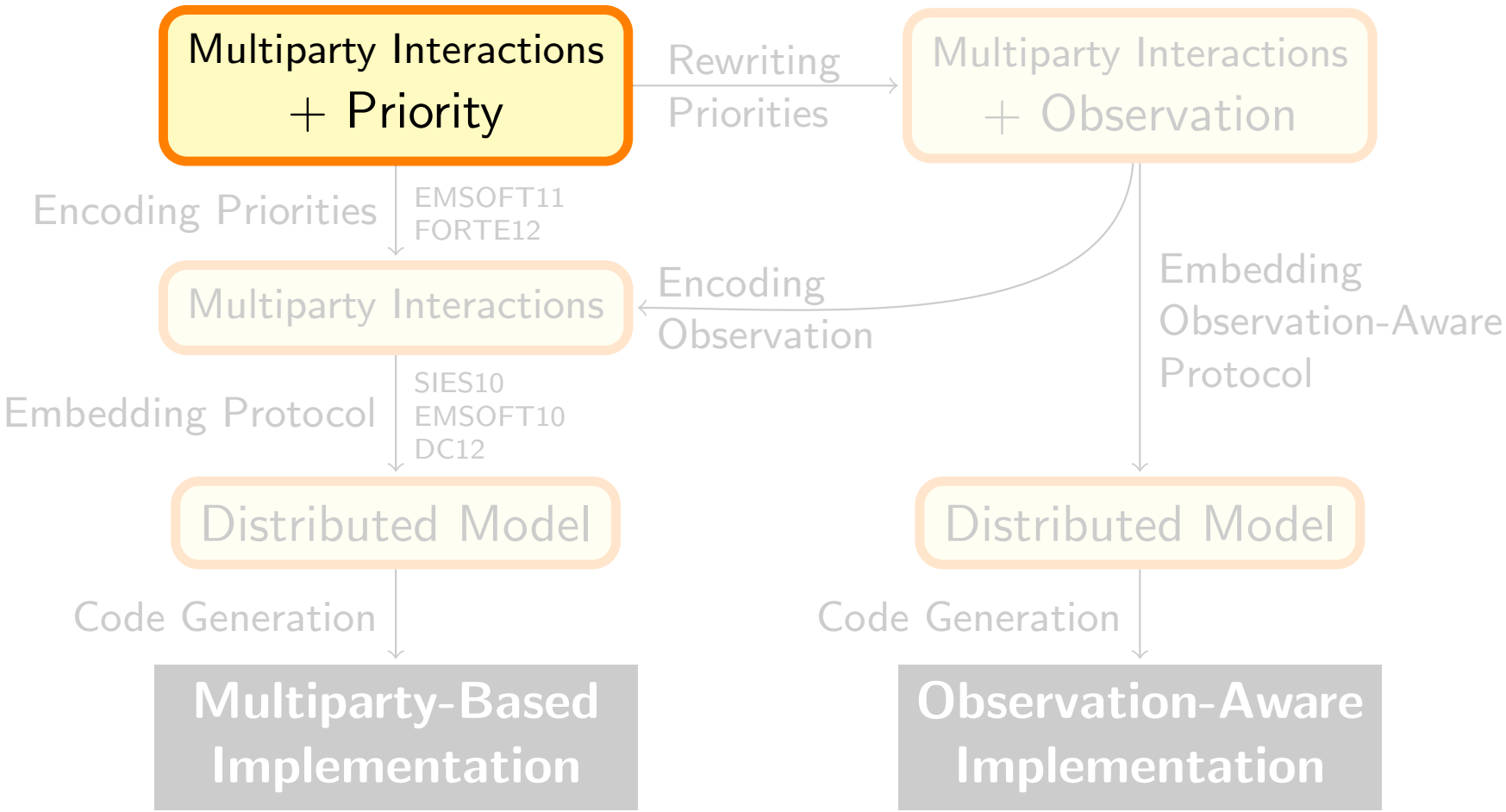


Distributed BIP Design Flow

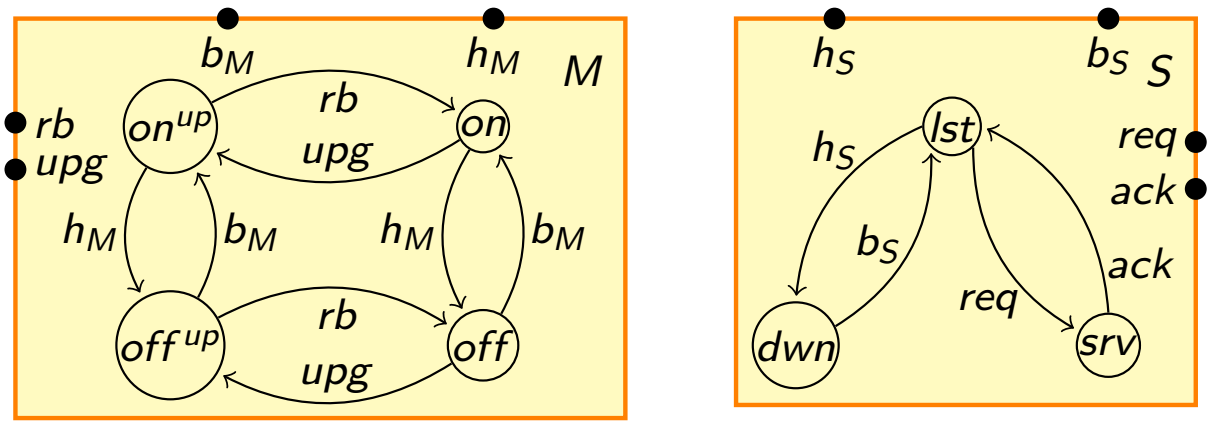


Outline

2 The BIP Model



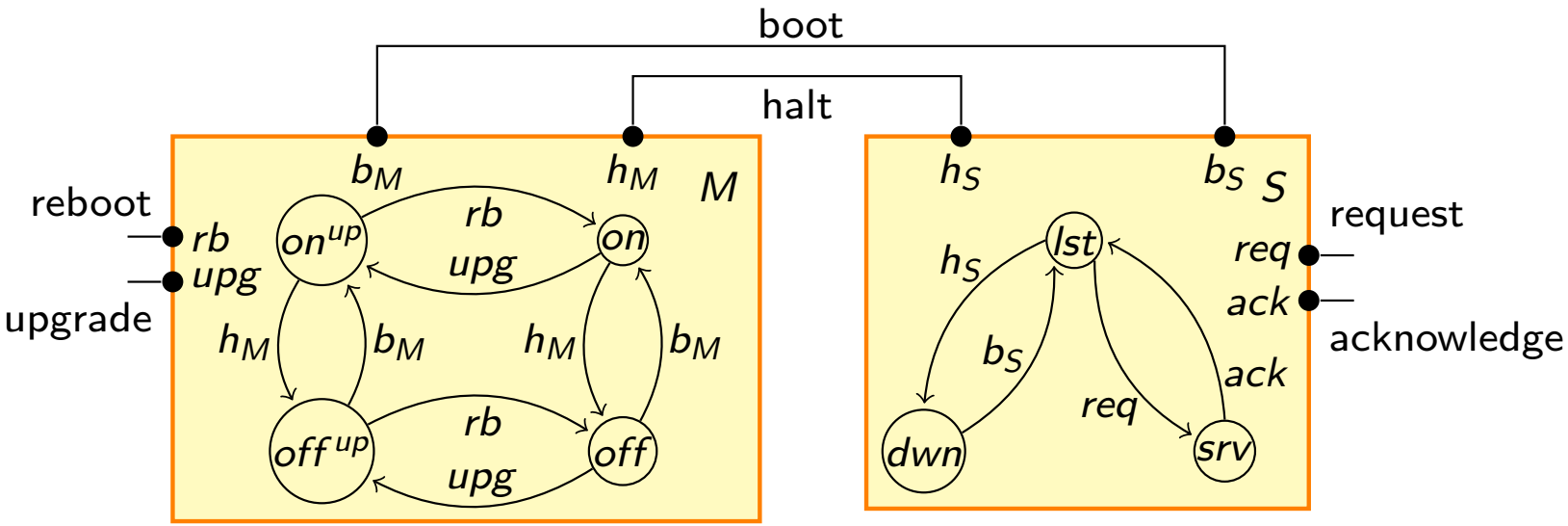
BIP



Behavior

- Ports (interface)
- Automaton

BIP



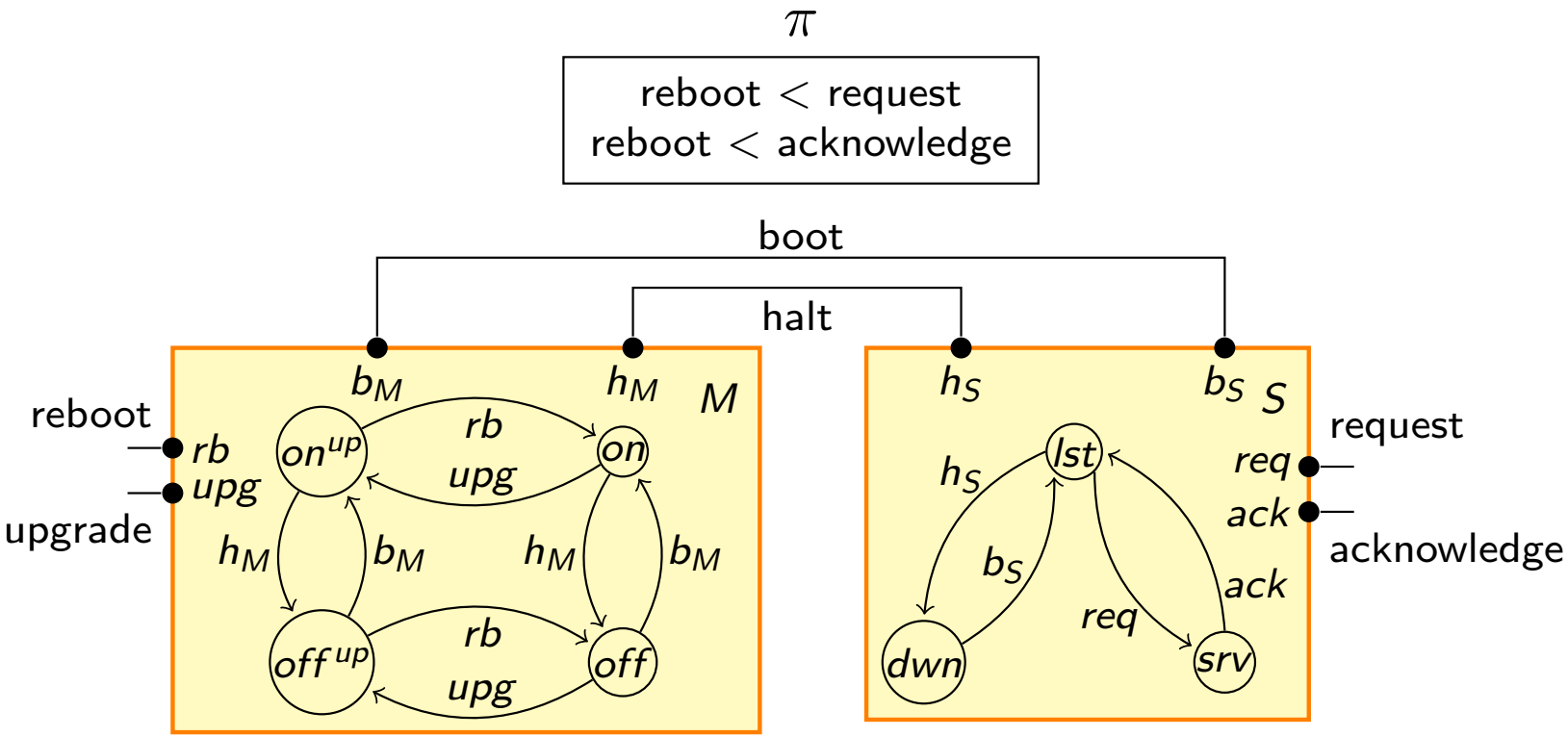
Behavior

- Ports (interface)
- Automaton

Interaction

- Set of ports

BIP



Behavior

- Ports (interface)
- Automaton

Interaction

- Set of ports

Priority

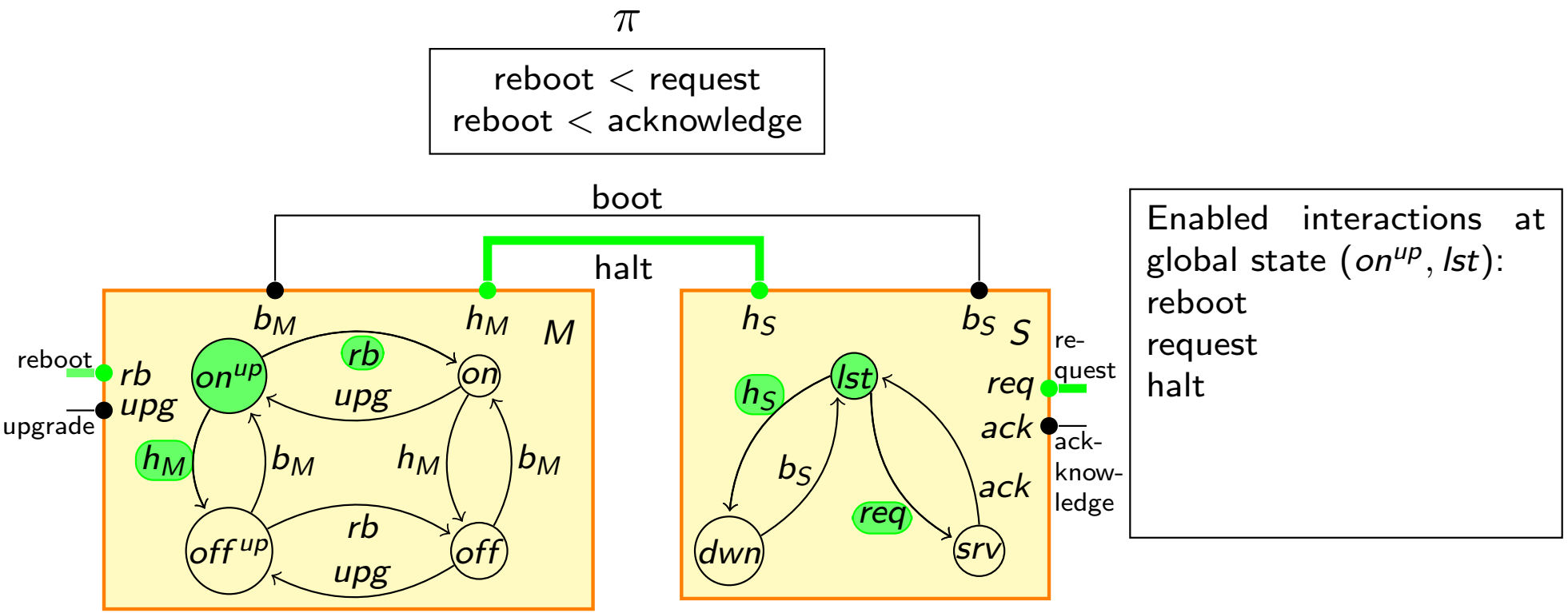
- Partial order over interactions

BIP Semantics

At global state q, interaction a can execute if:

- a is enabled, that is all its ports are enabled, denoted $EN_a(q)$
- there is no enabled interaction with higher priority

Execution of a is atomic and changes only the state of the participants.

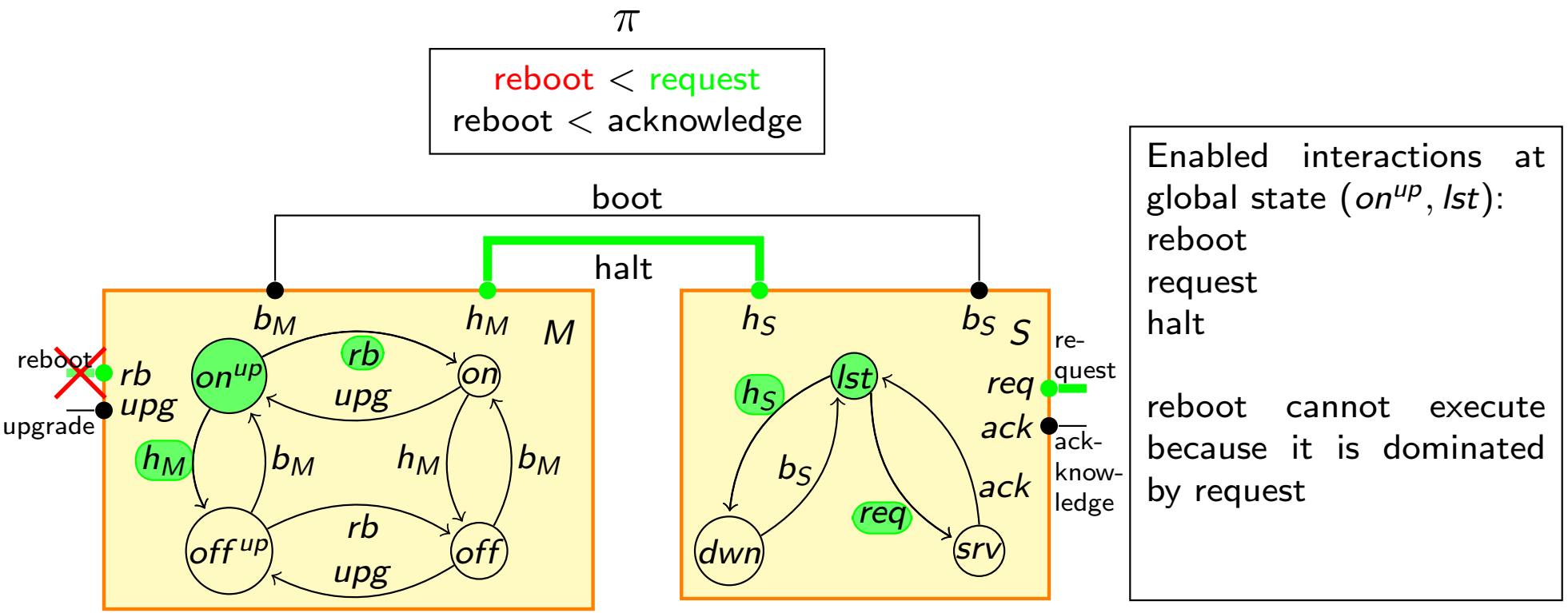


BIP Semantics

At global state q, interaction a can execute if:

- a is enabled, that is all its ports are enabled, denoted $EN_a(q)$
- there is no enabled interaction with higher priority

Execution of a is atomic and changes only the state of the participants.



Enabled interactions at global state (on^{up}, lst):
reboot
request
halt

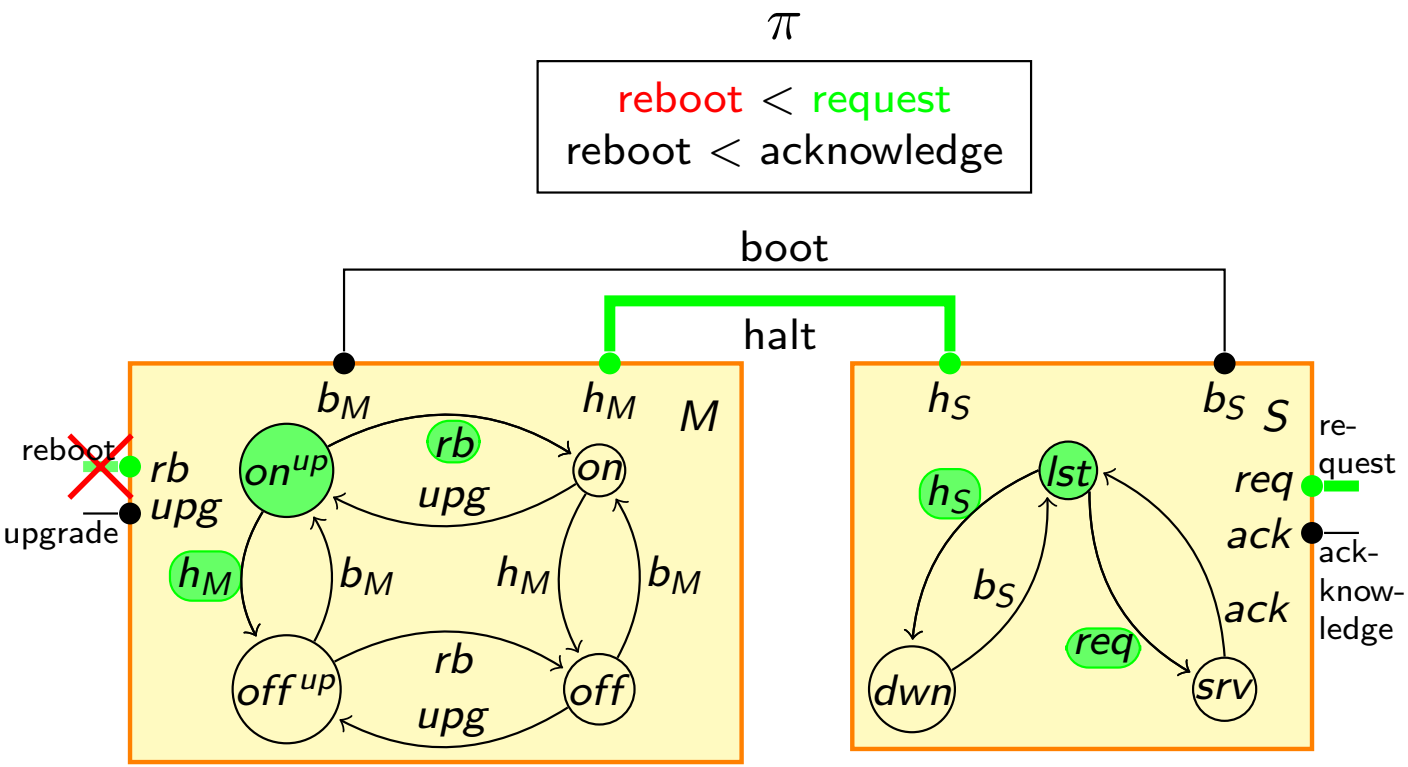
reboot cannot execute because it is dominated by request

BIP Semantics

At global state q, interaction a can execute if:

- a is enabled, that is all its ports are enabled, denoted $EN_a(q)$
- there is no enabled interaction with higher priority

Execution of a is atomic and changes only the state of the participants.

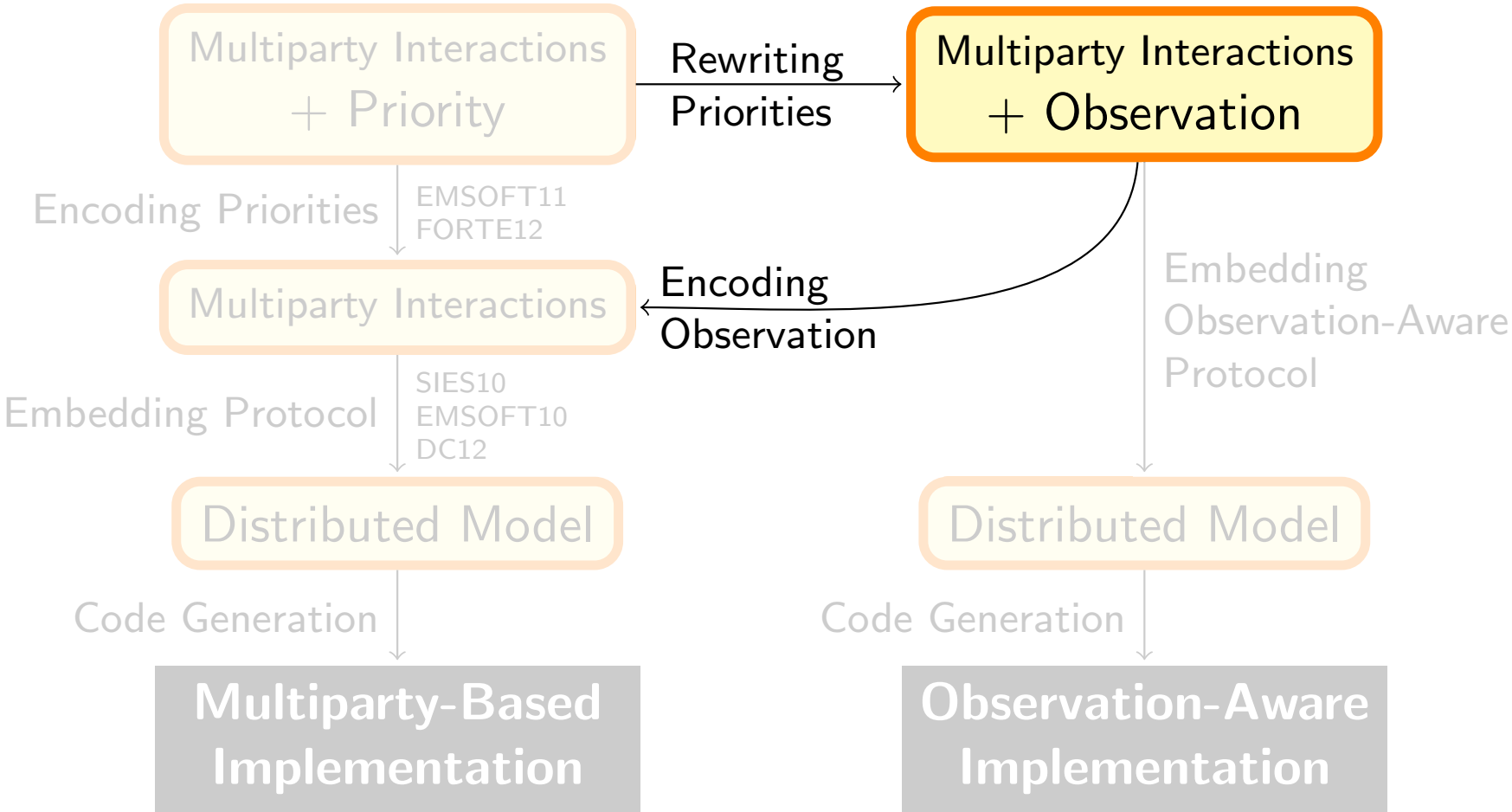


Enabled interactions at global state (on^{up}, lst):
reboot
request
halt

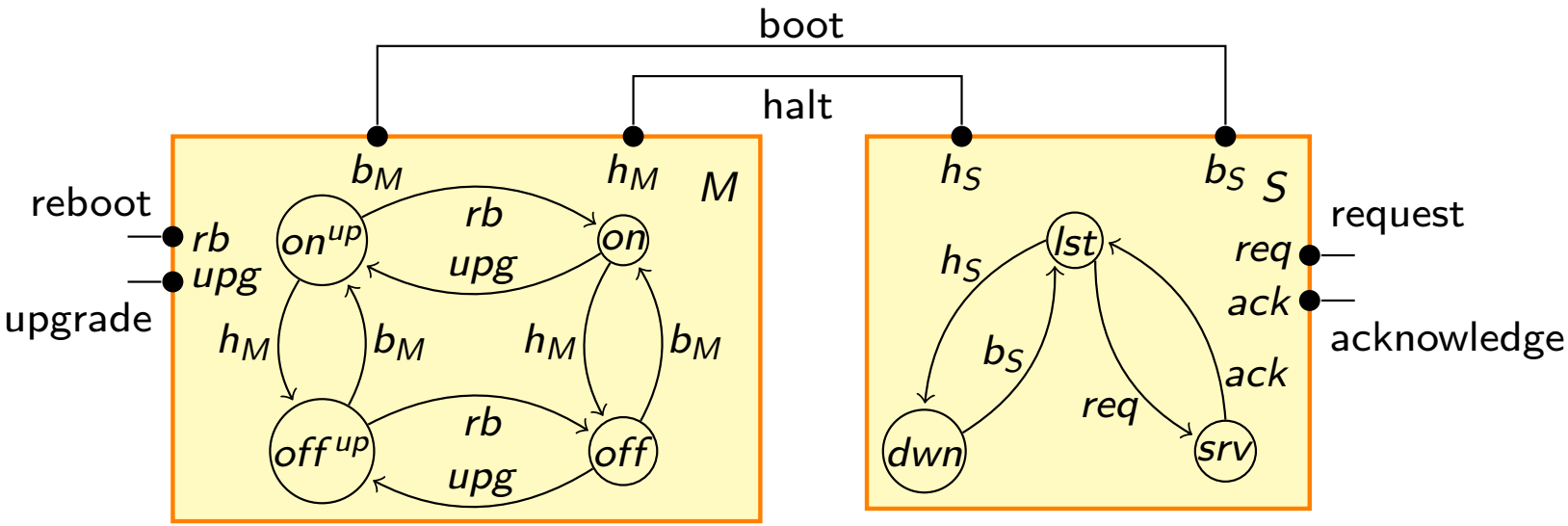
reboot cannot execute because it is dominated by request

Non-deterministic choice between request and halt.

3 Alternative Model: Observation



BIP



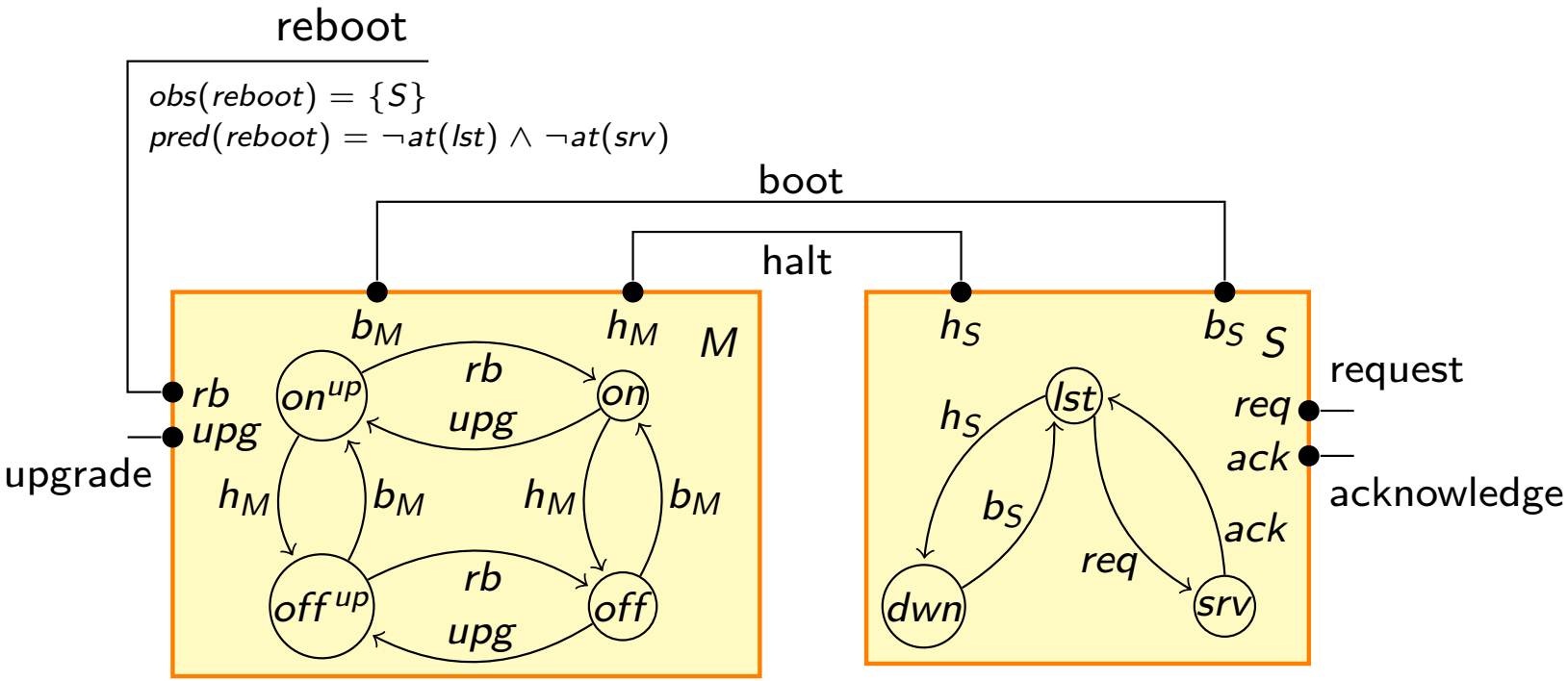
Behavior

- Ports (interface)
- Automaton

Interaction

- Set of ports

BIO



Behavior

- Ports (interface)
- Automaton

Interaction

- Set of ports

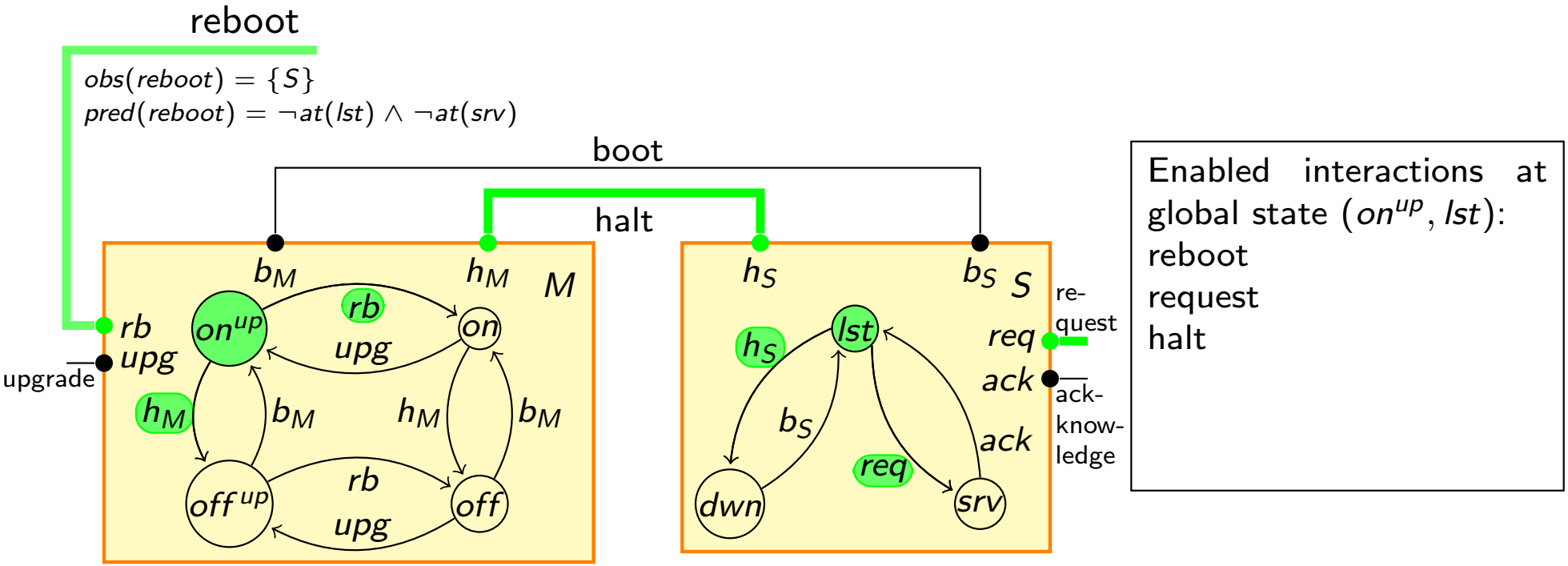
Observation (each interaction)

- observes a set of components
- is guarded by a state predicate

BIO Semantics

At global state q, interaction a can execute if:

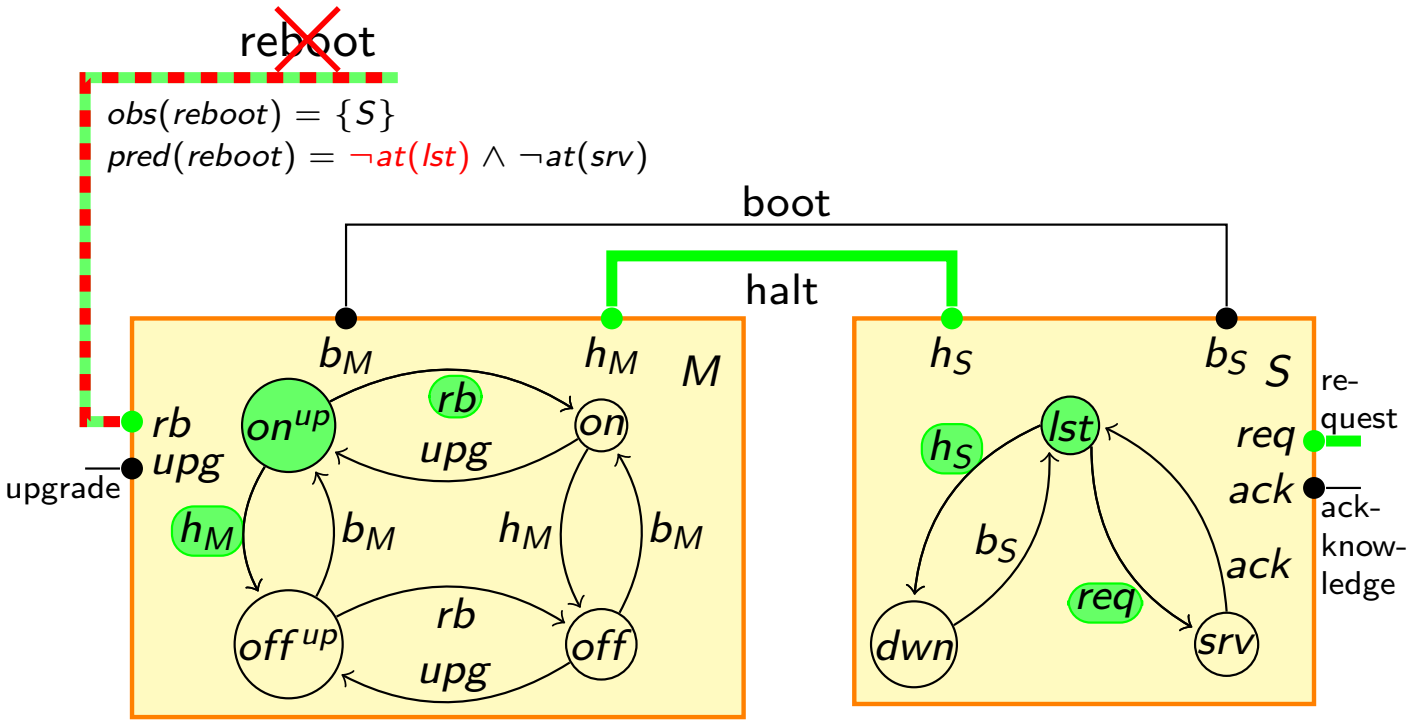
- a is enabled, that is all its ports are enabled, denoted $EN_a(q)$
- the predicate $pred(a)$ evaluates to $true$ at q



BIO Semantics

At global state q, interaction a can execute if:

- a is enabled, that is all its ports are enabled, denoted $EN_a(q)$
- the predicate $pred(a)$ evaluates to $true$ at q



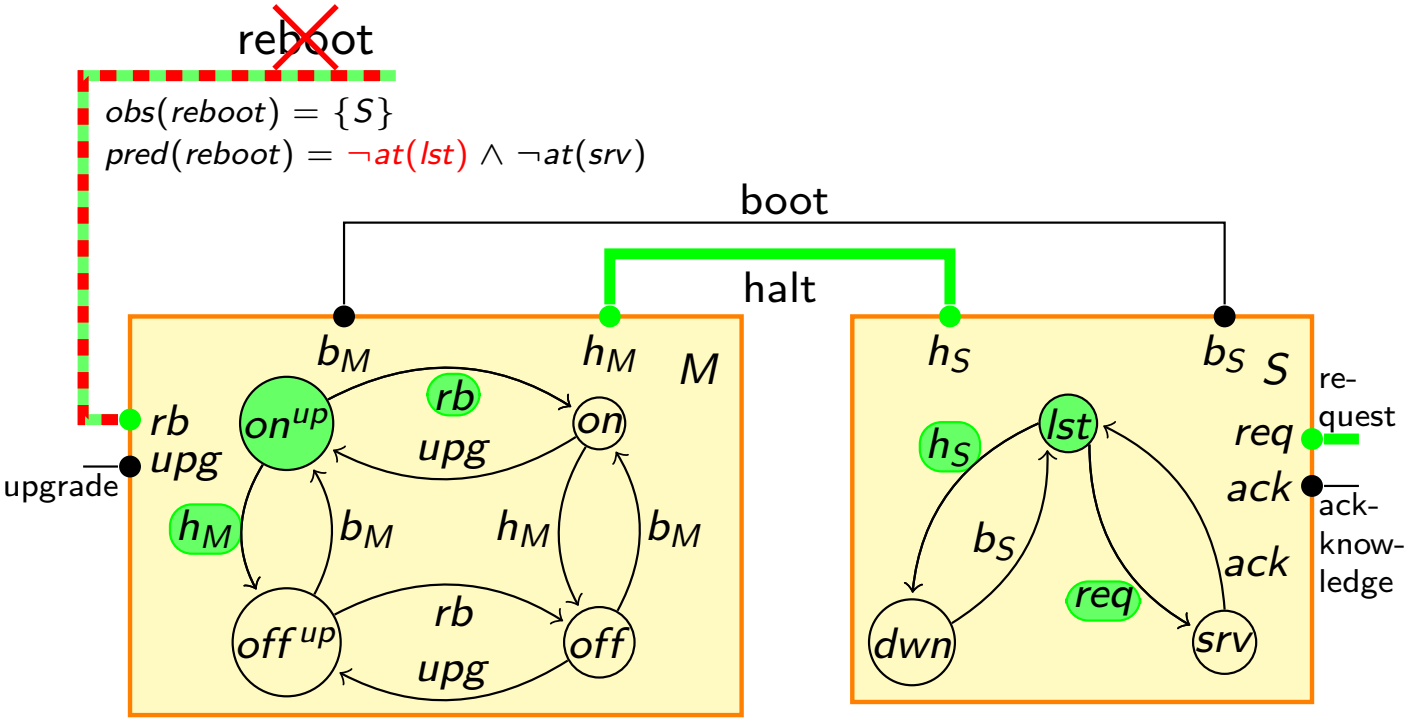
Enabled interactions at global state (on^{up}, lst):
reboot
request
halt

reboot cannot execute because its predicate is false

BIO Semantics

At global state q, interaction a can execute if:

- a is enabled, that is all its ports are enabled, denoted $EN_a(q)$
- the predicate $pred(a)$ evaluates to $true$ at q



Enabled interactions at global state (on^{up}, lst):
reboot
request
halt

reboot cannot execute because its predicate is false

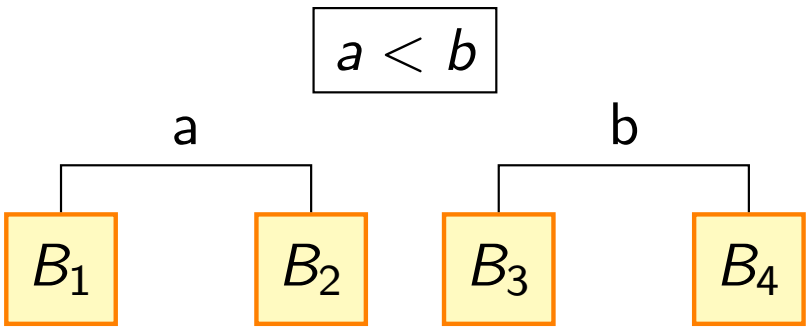
Non-deterministic choice between request and halt.

Expressing Priority with Observation

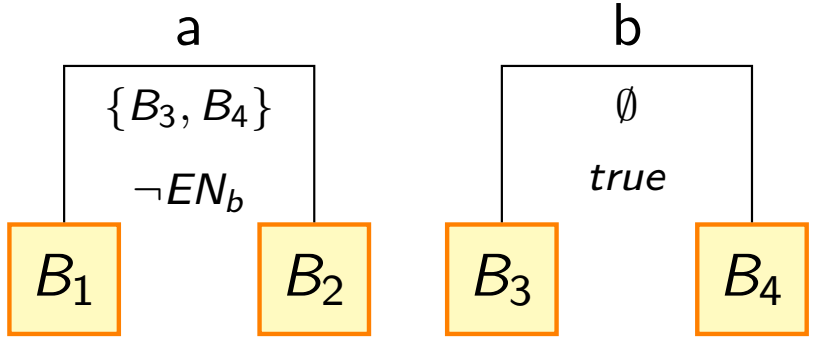
Given a priority π, we define the Observation $\mathcal{O}_\pi$,
such that each low priority interaction

- observes all components involved in higher priority interactions,
- checks disableness of all higher priority interactions.

Priority π:



Observation $\mathcal{O}_\pi$:

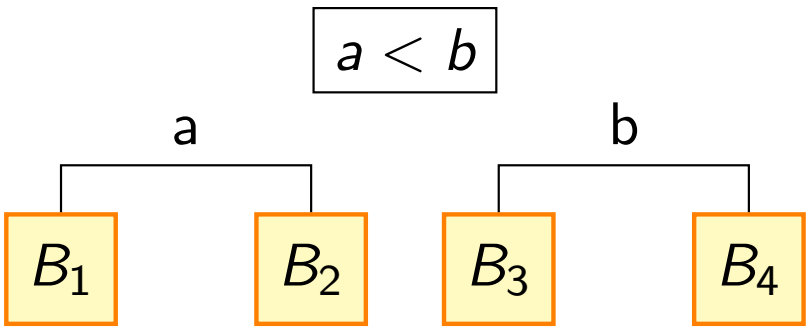


Expressing Priority with Observation

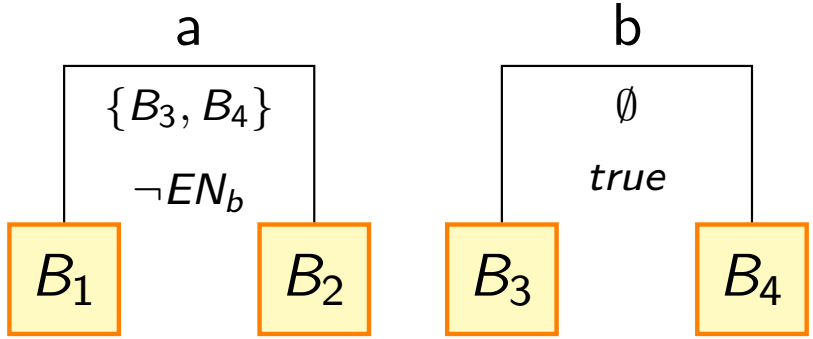
Given a priority π, we define the Observation $\mathcal{O}_\pi$,
such that each low priority interaction

- observes all components involved in higher priority interactions,
- checks disableness of all higher priority interactions.

Priority π:



Observation $\mathcal{O}_\pi$:



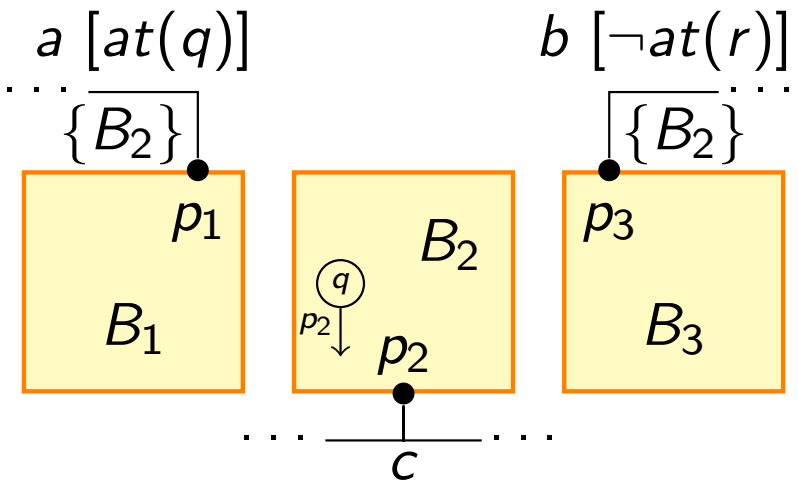
Proposition *The two models allow exactly the same transitions (are bisimilar).*

Expressing Observation with Multiparty Interactions

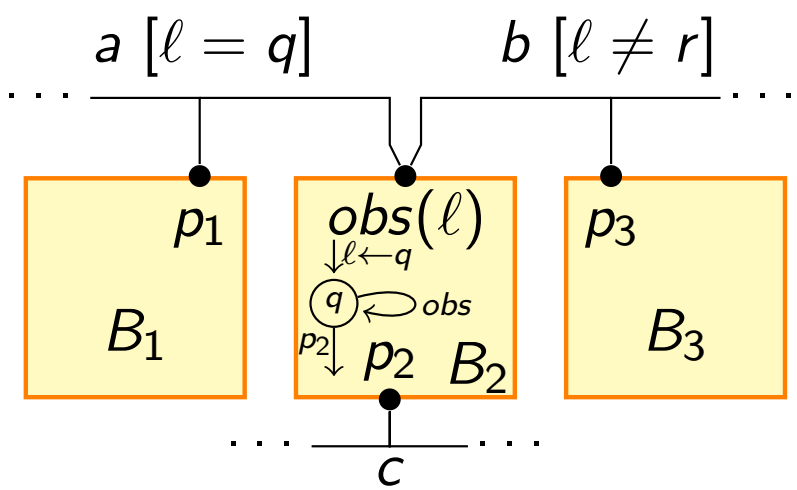
Given model with Observation $\mathcal{O}$ we define a model with Multiparty Interactions:

- each observed component exports its state through a new port *obs*,
- interactions are extended to port *obs* of the observed component and the predicate is transformed into a guard.

Observation $\mathcal{O}$:



Multiparty Interactions:

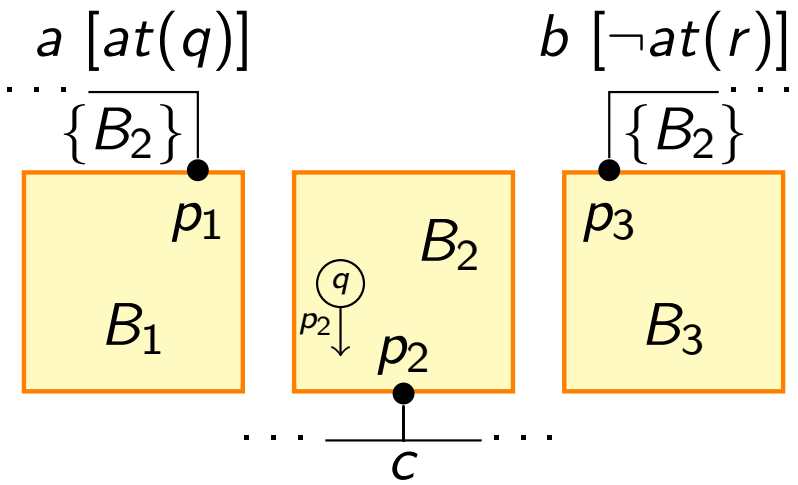


Expressing Observation with Multiparty Interactions

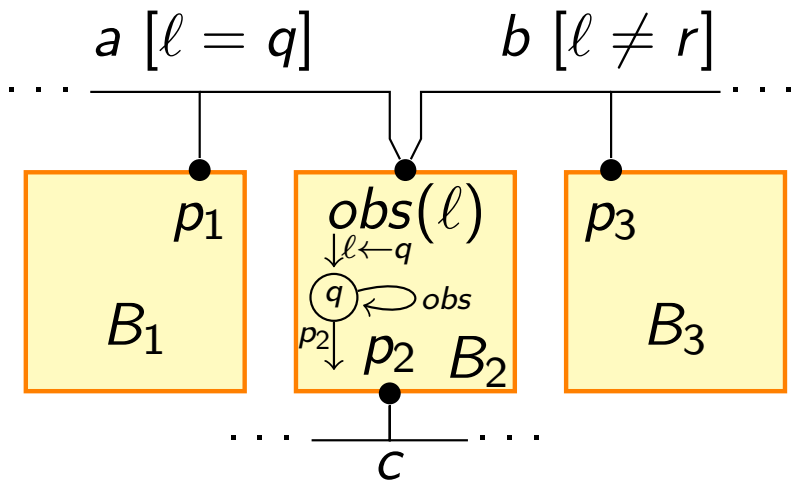
Given model with Observation $\mathcal{O}$ we define a model with Multiparty Interactions:

- each observed component exports its state through a new port *obs*,
- interactions are extended to port *obs* of the observed component and the predicate is transformed into a guard.

Observation $\mathcal{O}$:

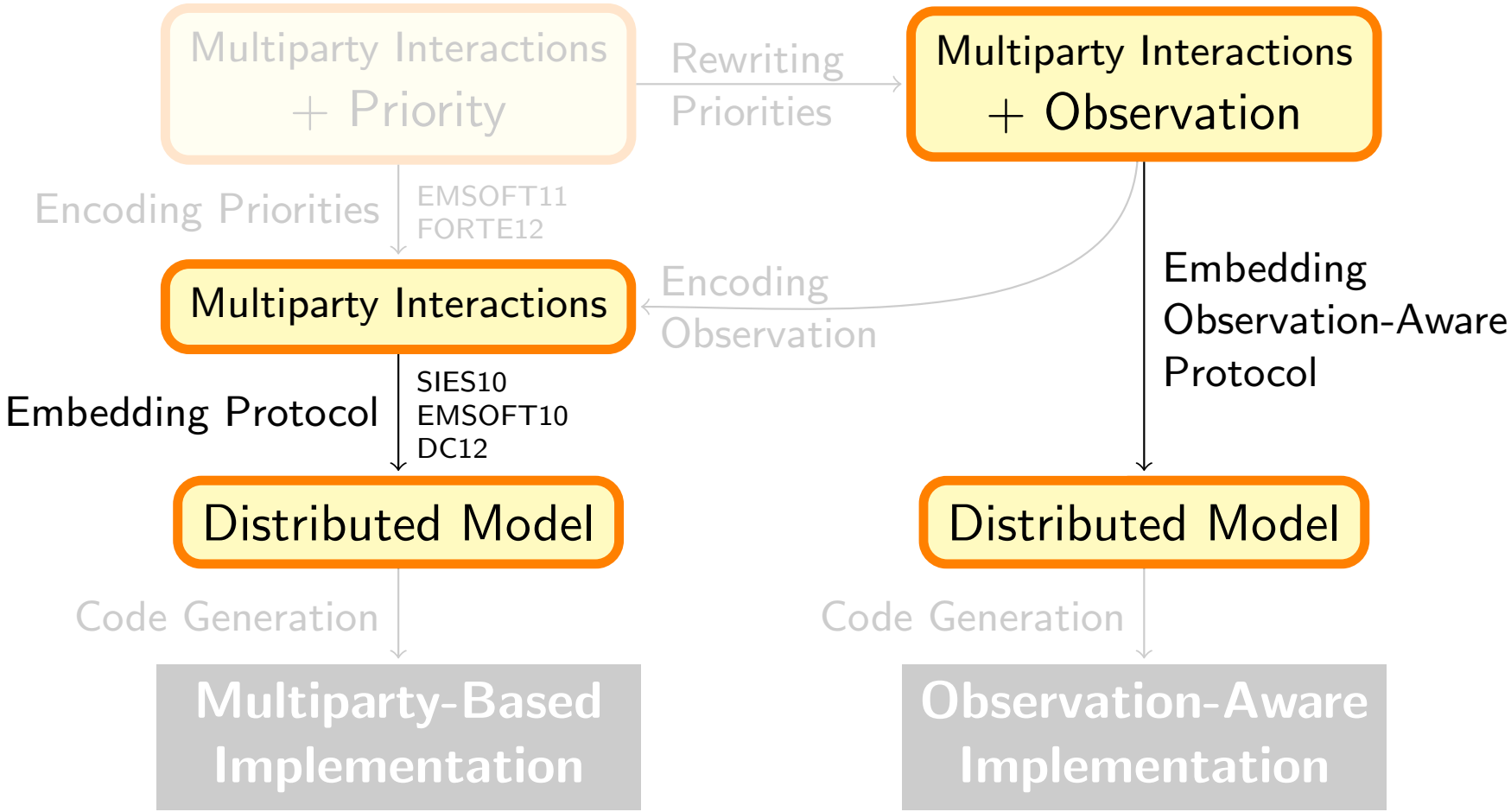


Multiparty Interactions:



Proposition *The two models allow exactly the same transitions (are bisimilar).*

4 Distributed Protocols



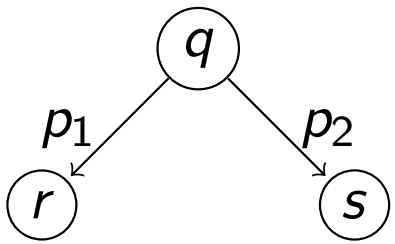
Distributed components

In a distributed context, atomicity of transitions is broken.

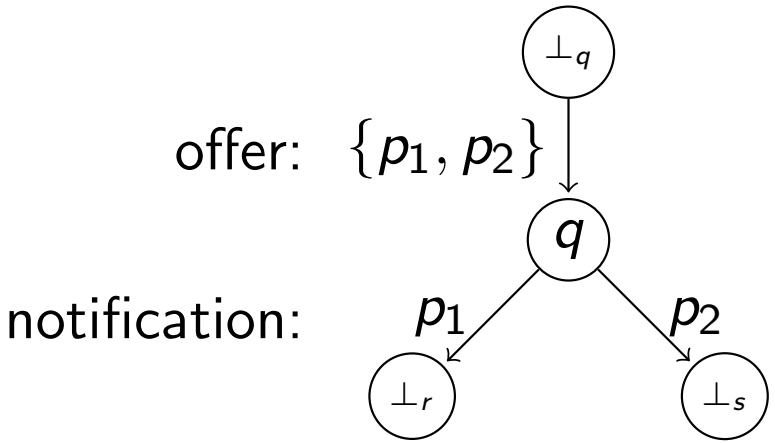
Each transition (in distributed atomic components) is done in two steps:

- sending an offer (indicating enabled ports), through a special port
- receiving a notification (indicating on which port to execute)

Original Behavior:



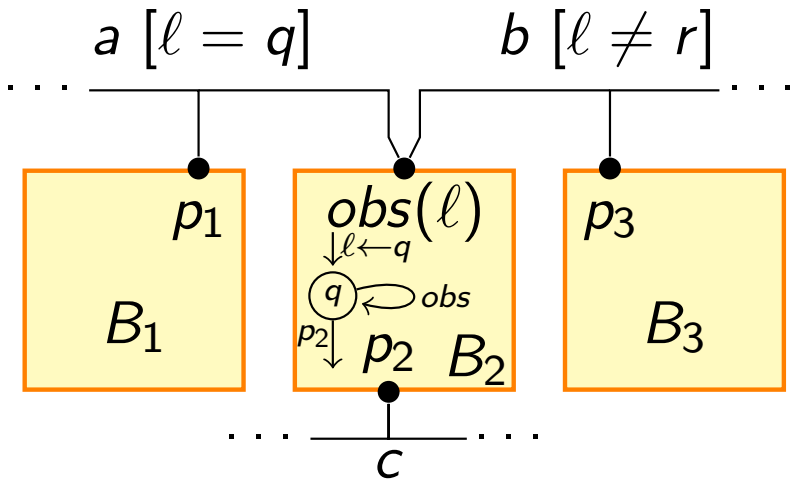
Distributed Behavior:



A distributed protocol gathers offer and sends notifications.

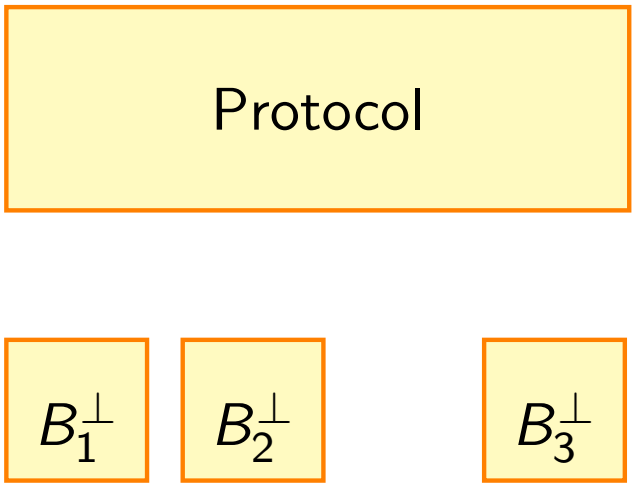
Distributed Protocol for Multiparty Interactions

Centralized Example



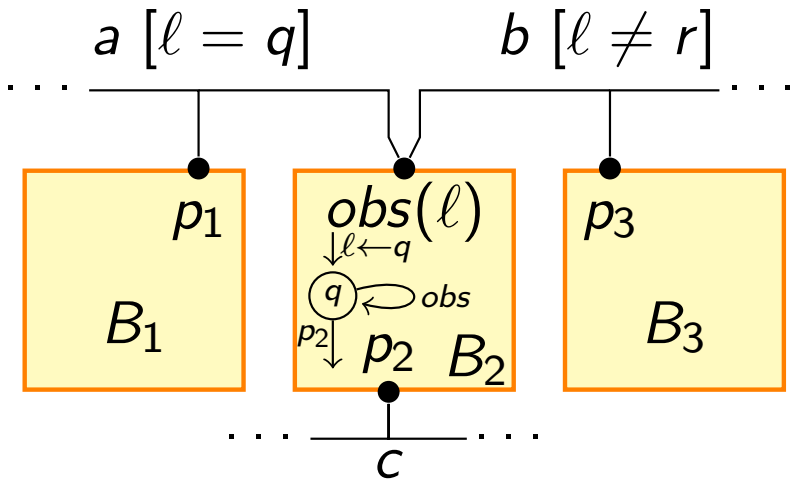
Protocol

Distributed Version

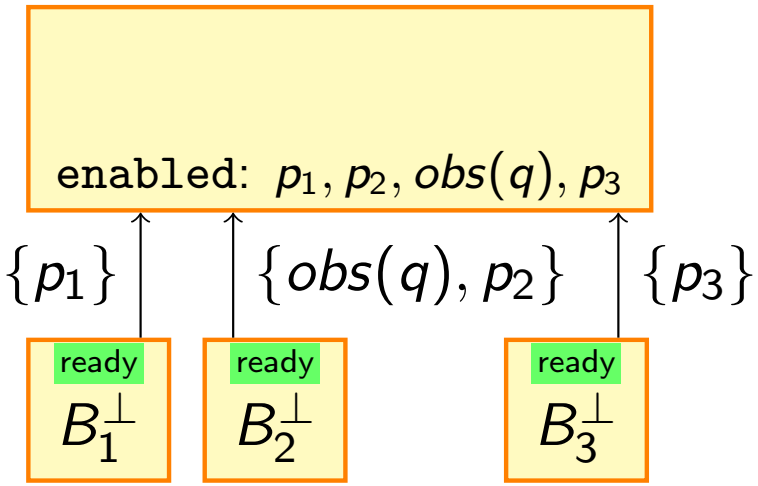


Distributed Protocol for Multiparty Interactions

Centralized Example



Distributed Version

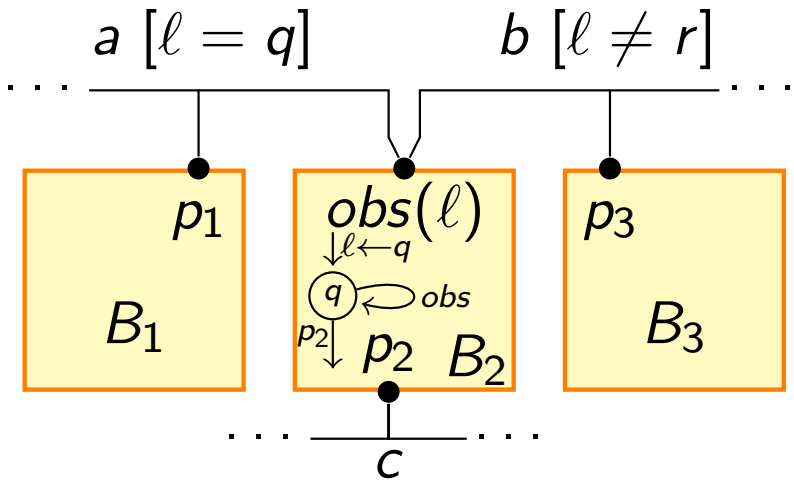


Protocol

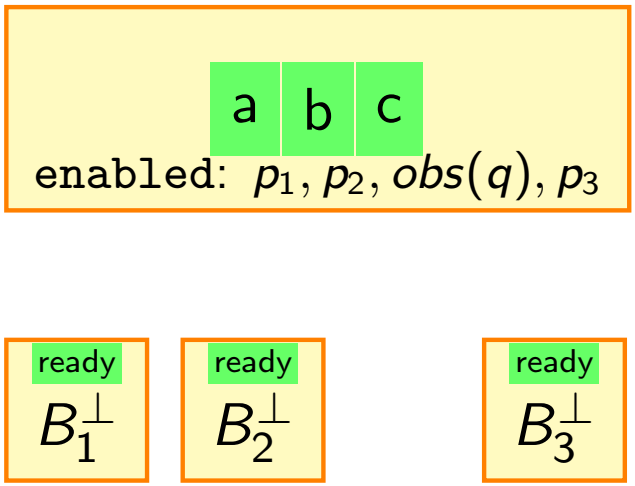
- Gather offers
  - Mark component as ready
  - Record enabled ports

Distributed Protocol for Multiparty Interactions

Centralized Example



Distributed Version

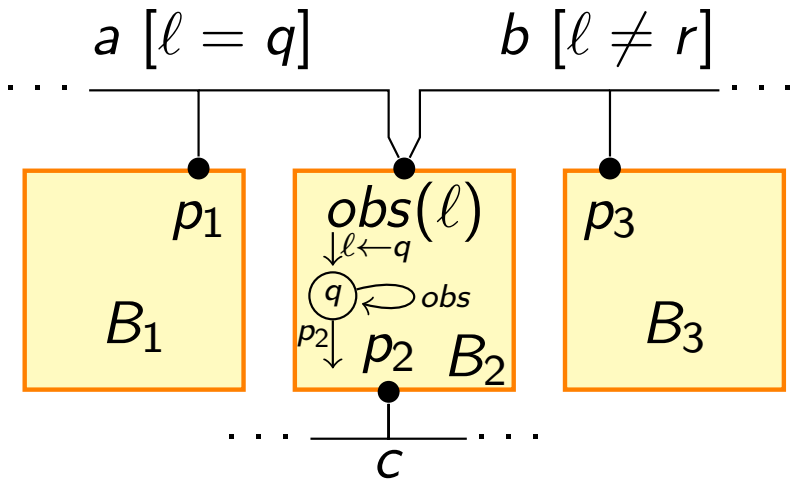


Protocol

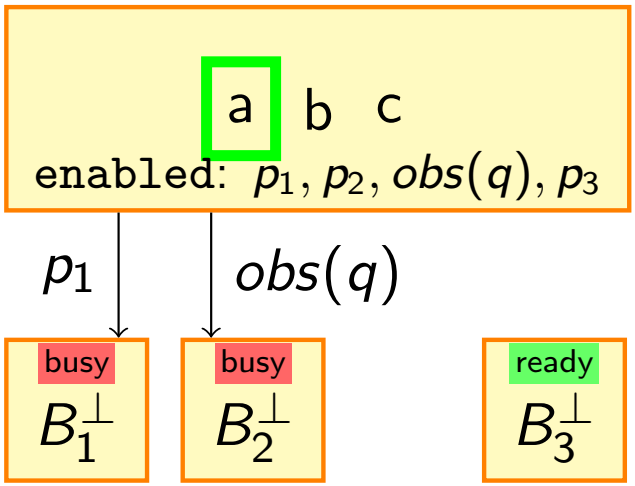
- Gather offers
  - Mark component as ready
  - Record enabled ports
- Detect enabled interactions
  - all ports are enabled
  - all participants are ready

Distributed Protocol for Multiparty Interactions

Centralized Example



Distributed Version

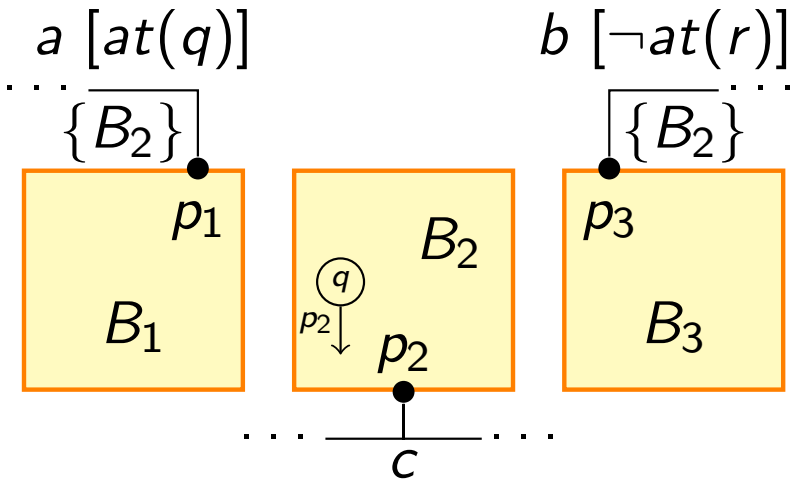


Protocol

- Gather offers
  - Mark component as ready
  - Record enabled ports
- Detect enabled interactions
  - all ports are enabled
  - all participants are ready
- Select an enabled interaction and execute it:
  - Send notifications to participants
  - Mark participants as not ready

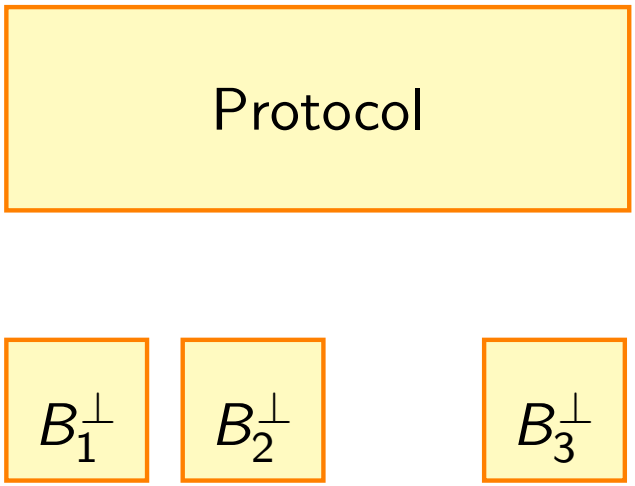
Observation-Aware Implementation

Centralized Example



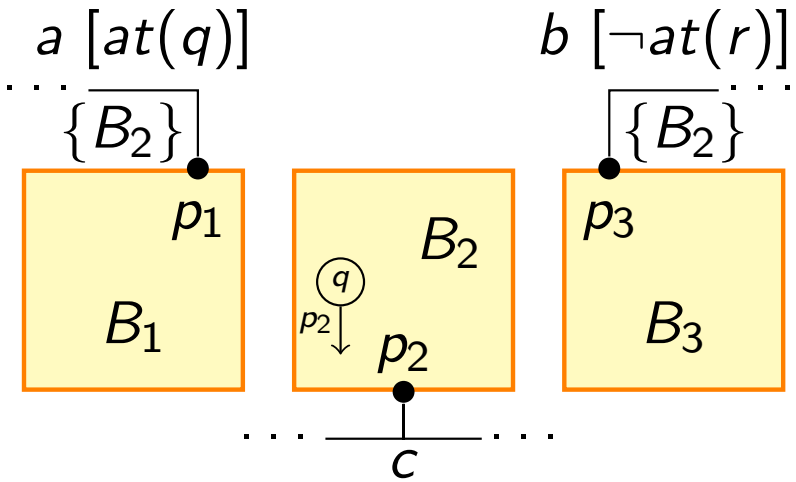
Protocol

Distributed Version



Observation-Aware Implementation

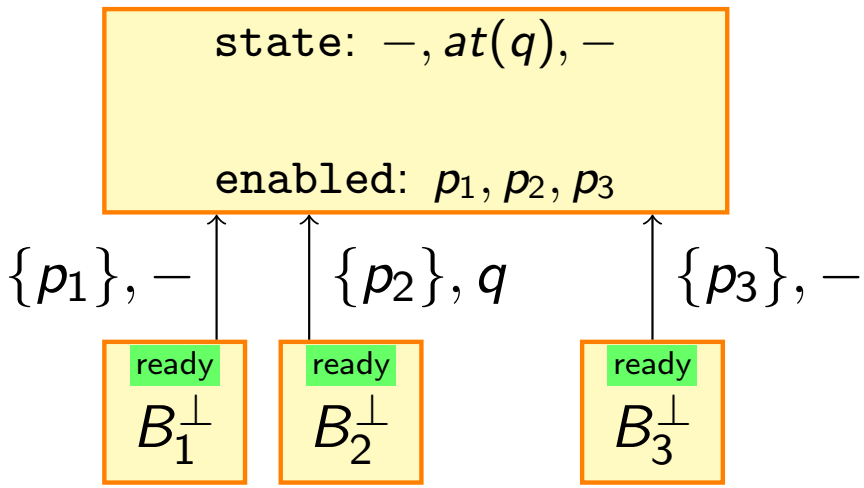
Centralized Example



Protocol

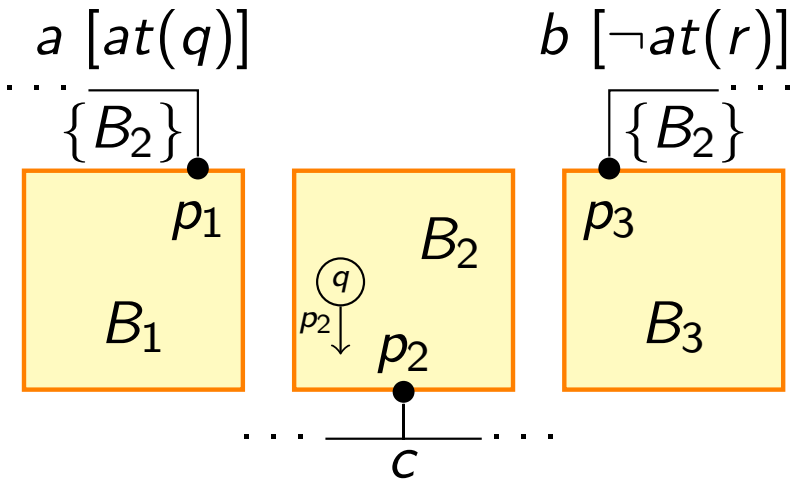
- Gather offers
  - Mark component as ready
  - Record enabled ports
  - **Record states**

Distributed Version

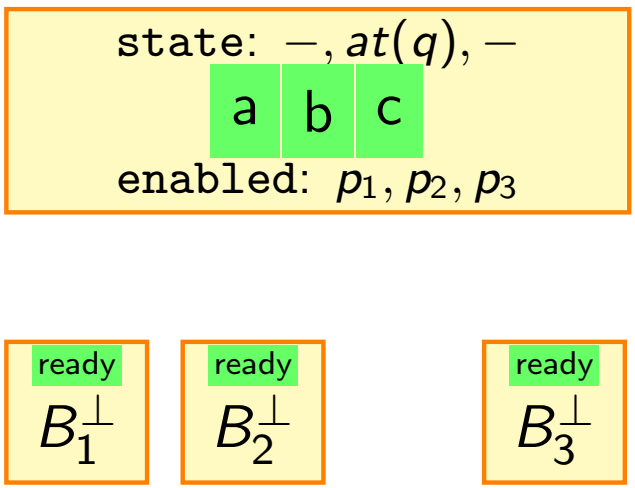


Observation-Aware Implementation

Centralized Example



Distributed Version

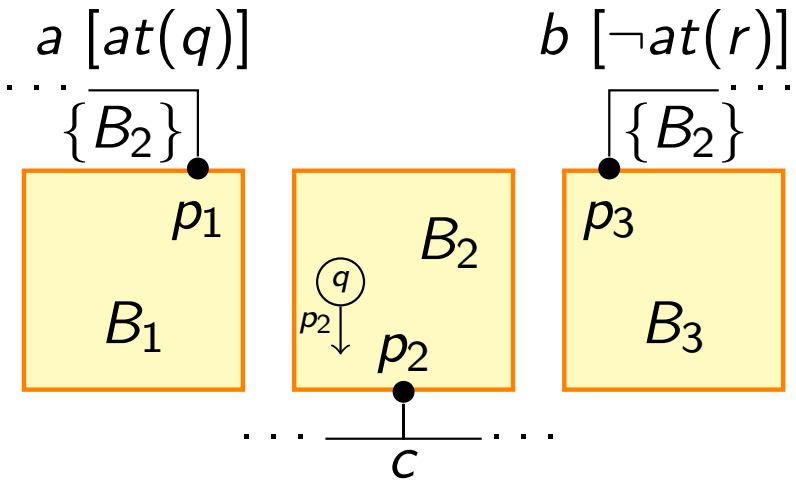


Protocol

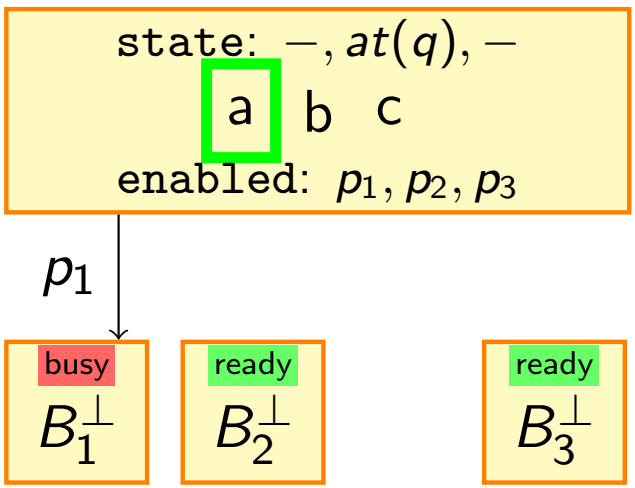
- Gather offers
  - Mark component as ready
  - Record enabled ports
  - **Record states**
- Detect possible interactions
  - all ports are enabled
  - participants **and observed components** are ready
  - **state predicate holds**

Observation-Aware Implementation

Centralized Example



Distributed Version

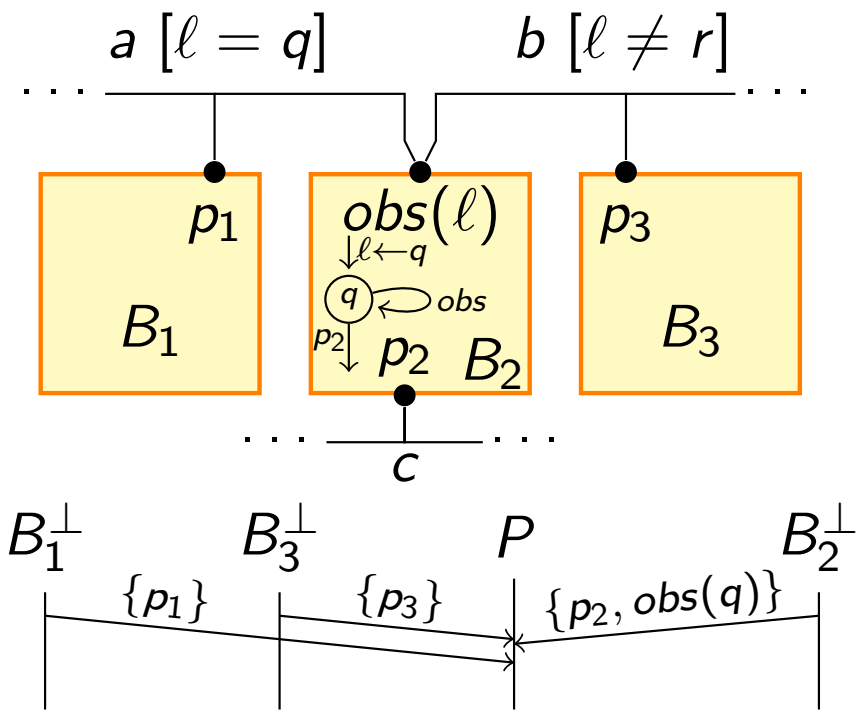


Protocol

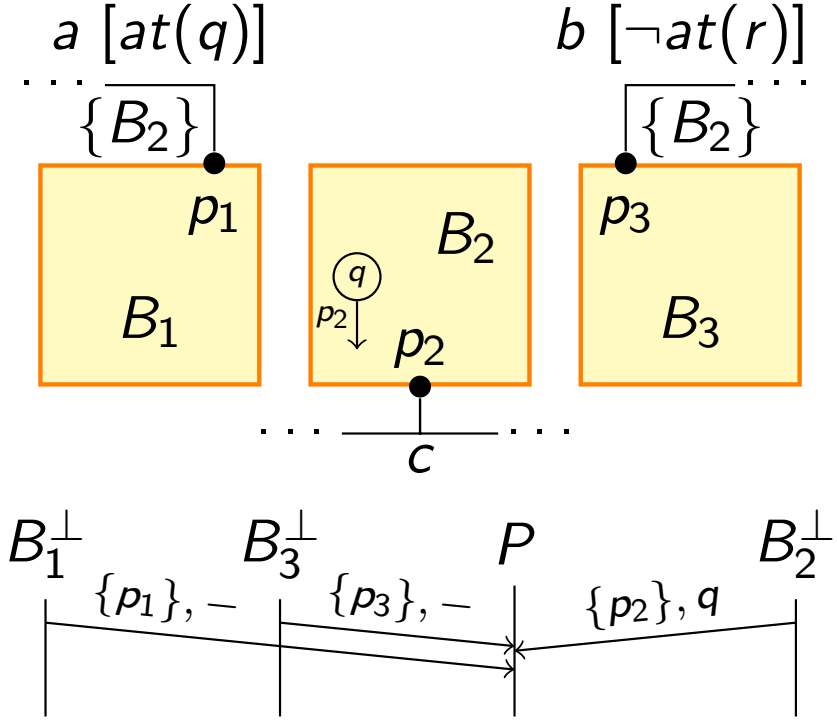
- Gather offers
  - Mark component as ready
  - Record enabled ports
  - **Record states**
- Detect possible interactions
  - all ports are enabled
  - participants **and observed components** are ready
  - **state predicate holds**
- Select an enabled interaction and execute it:
  - Send notifications to participants
  - Mark participants as not ready **but not observed components**

Message Efficiency Comparison

Multiparty-Based

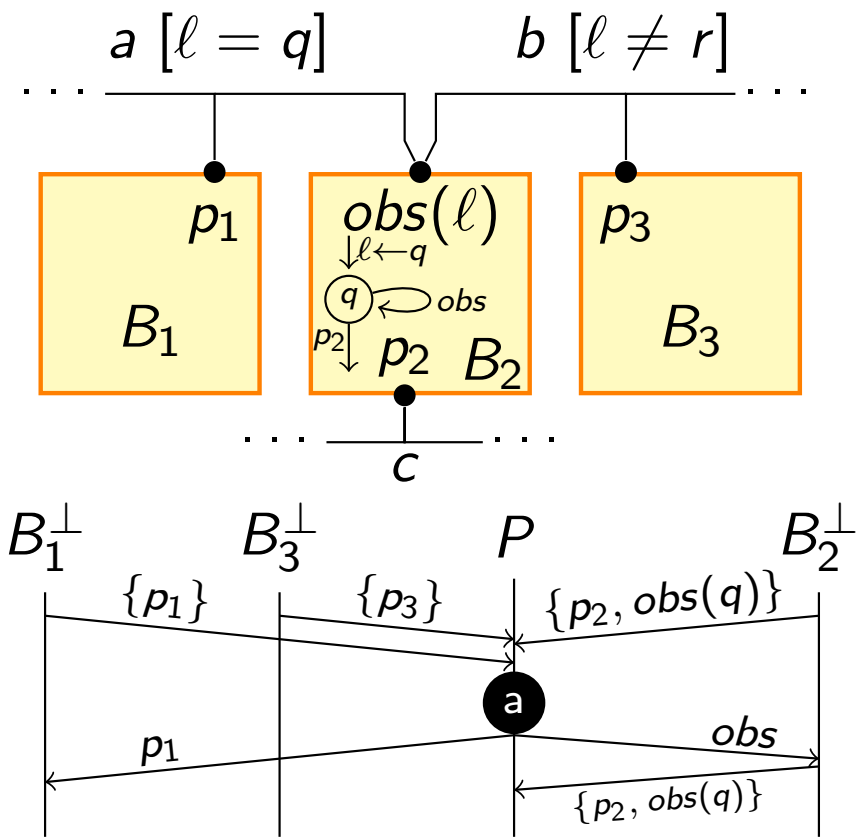


Observation-Aware

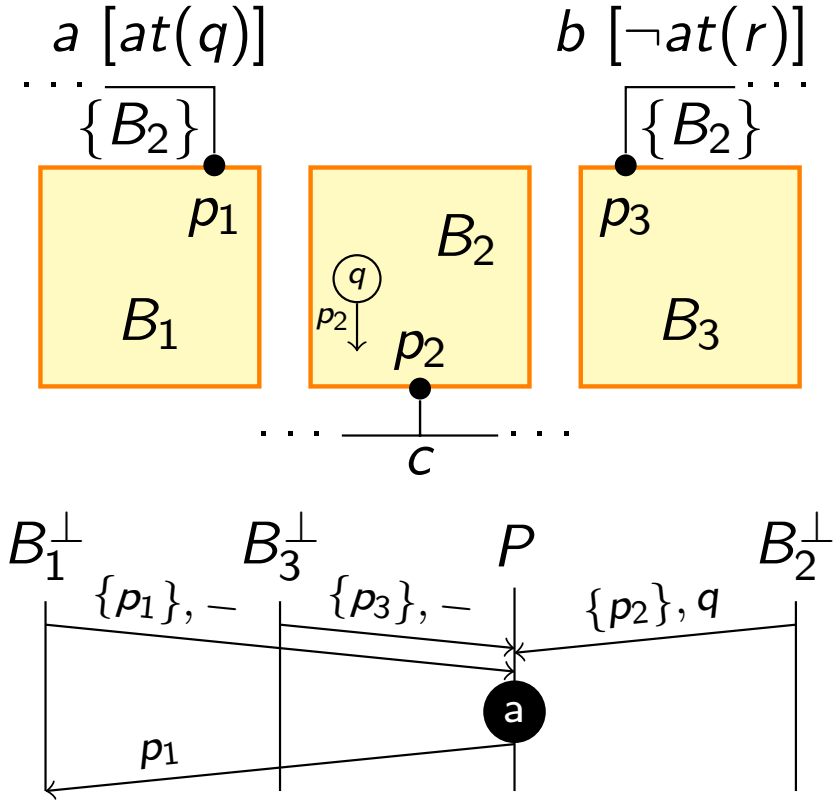


Message Efficiency Comparison

Multiparty-Based

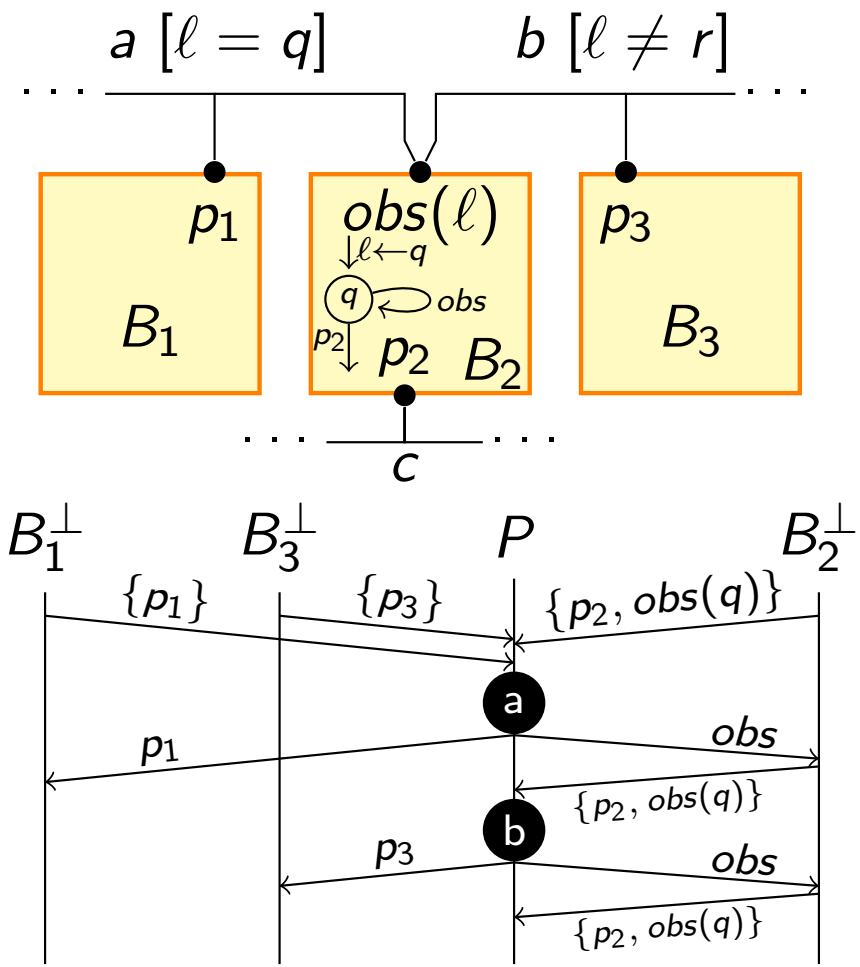


Observation-Aware

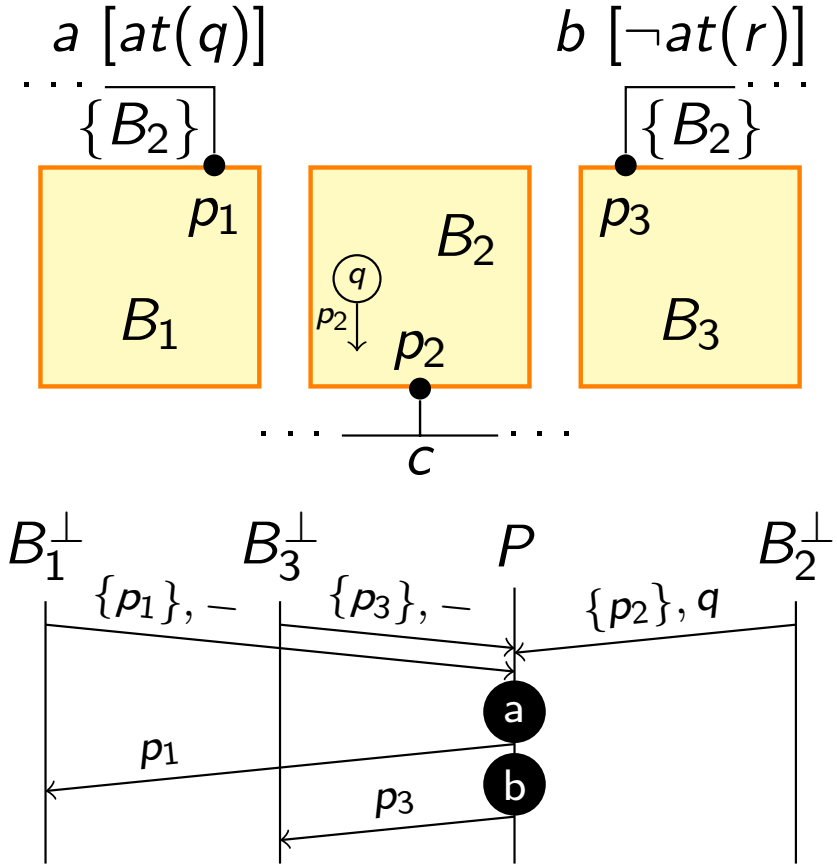


Message Efficiency Comparison

Multiparty-Based

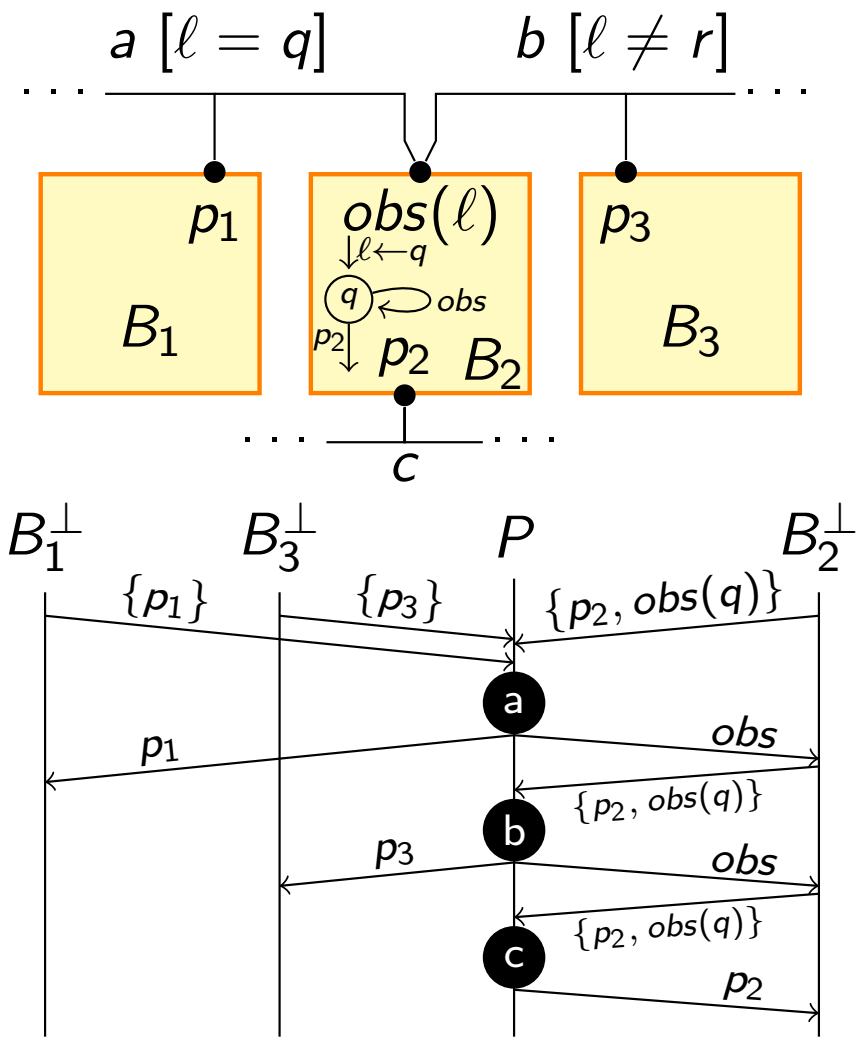


Observation-Aware

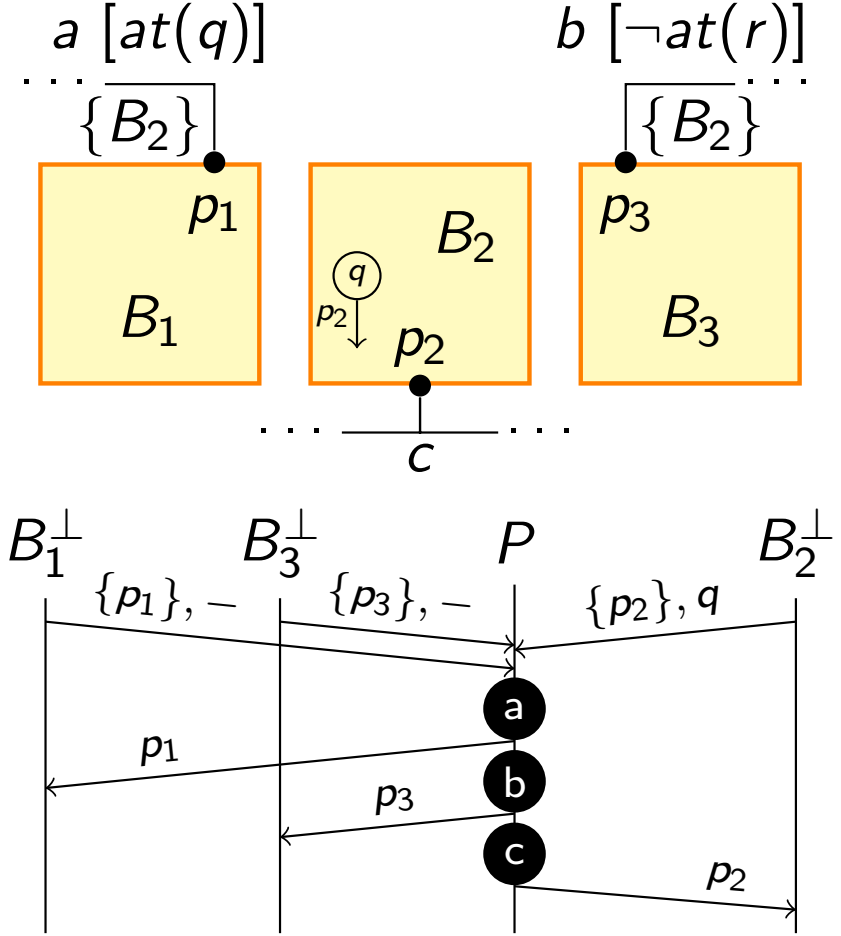


Message Efficiency Comparison

Multiparty-Based



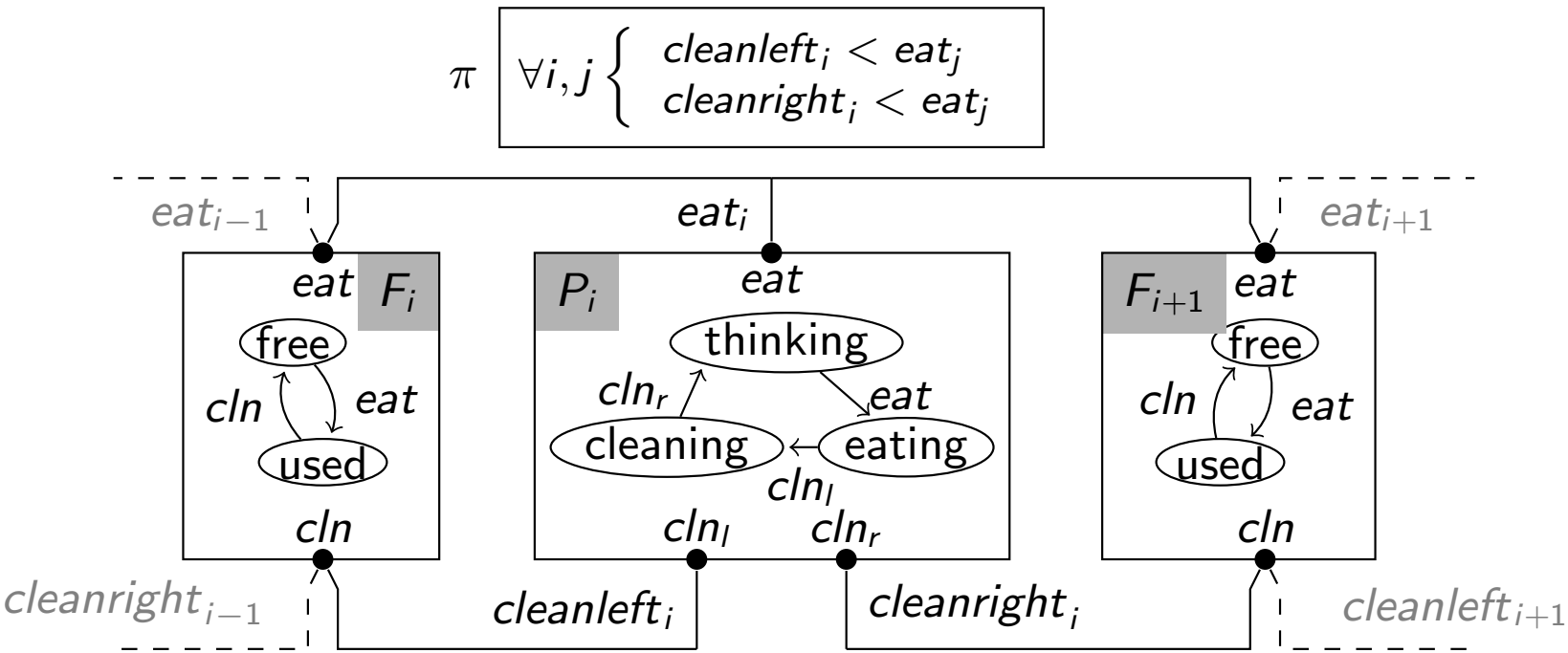
Observation-Aware



Outline

Example: Dining philosophers

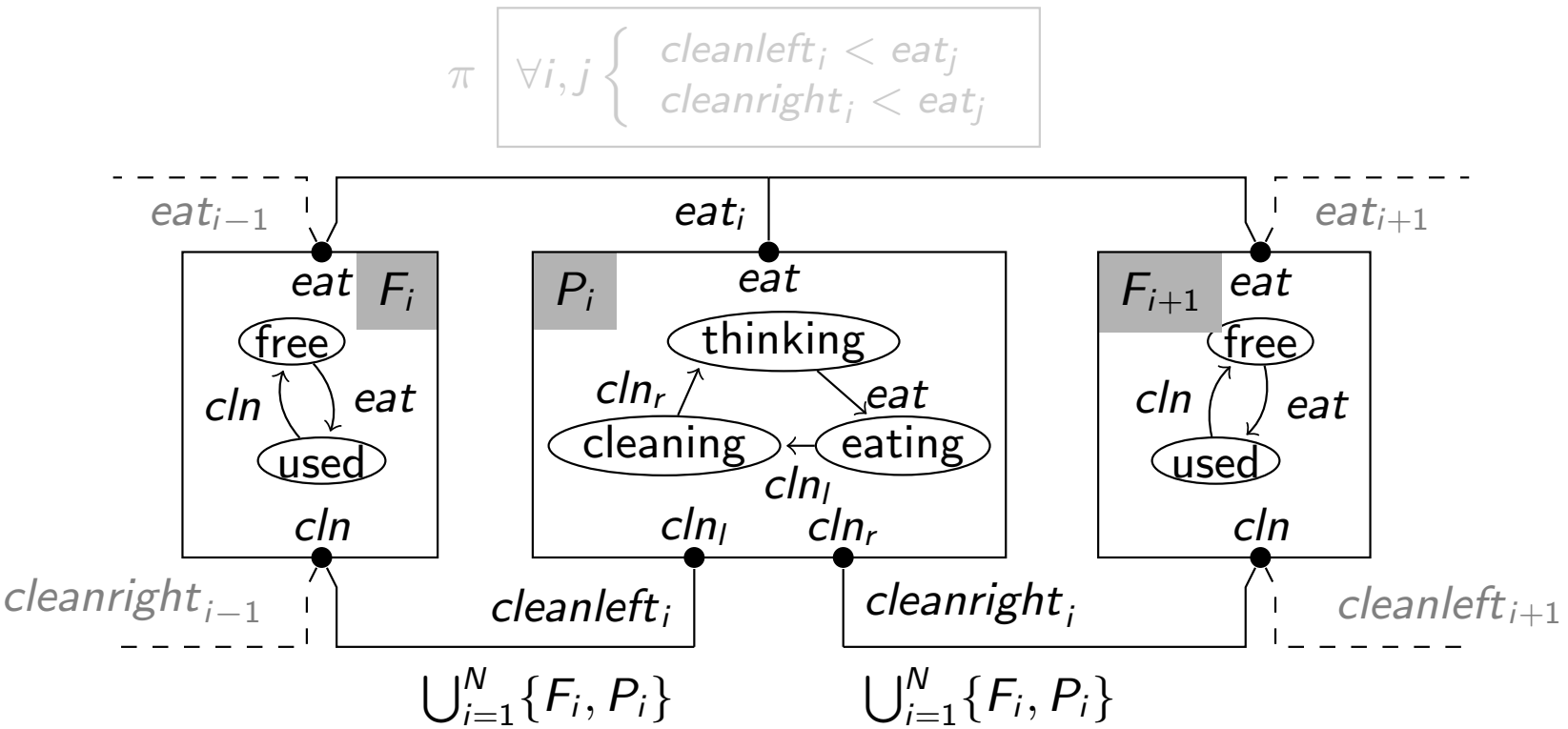
Model with N philosophers and N forks.



Only interactions $cleanright_i$ and $cleanleft_{i+1}$ have low priority.

Example: Dining philosophers

Model with N philosophers and N forks.

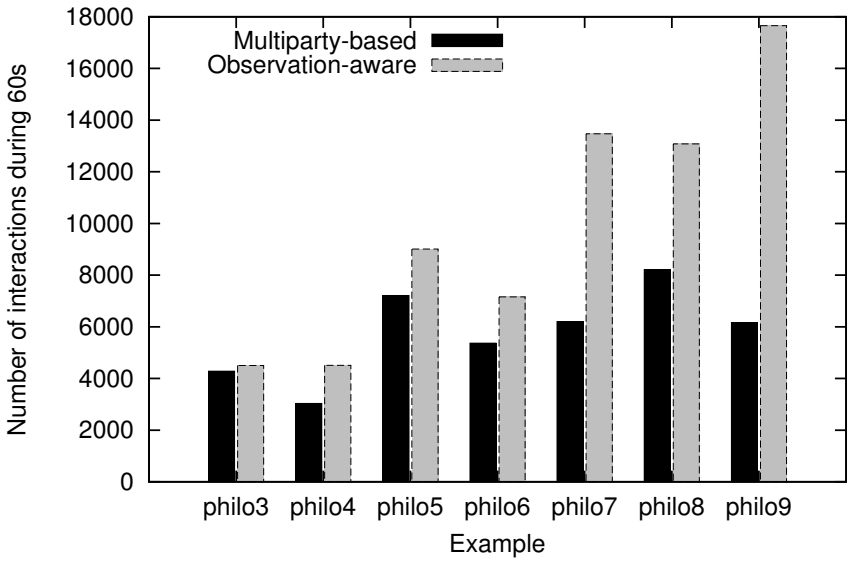


Only interactions $cleanright_i$ and $cleanleft_{i+1}$ have low priority.

They need to observe all components.

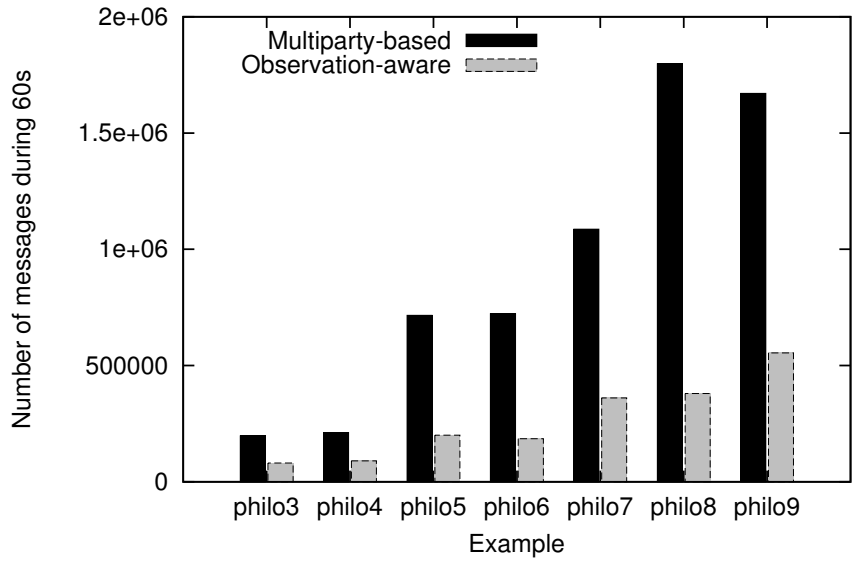
Dining Philosophers

Interactions executed (during 60s)



Observation-aware implementation gives better performance.

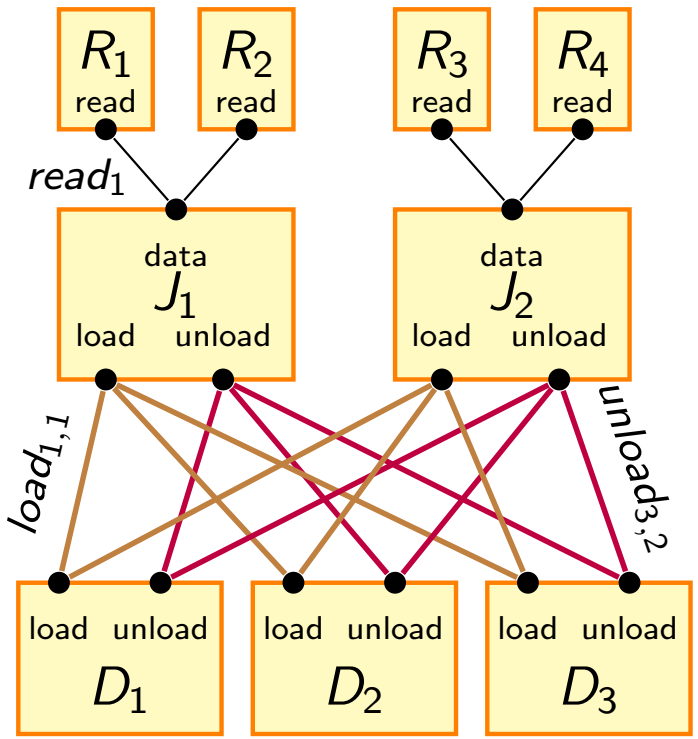
Messages Exchanged (during 60s)



Observation-aware implementation requires fewer messages.

Jukebox Example

$$\forall i, j, k,\ unload_{i,j} < read_k$$
$$\forall i \in \{2, 3, 4\},\ load_{1,1} < load_{i,1}$$



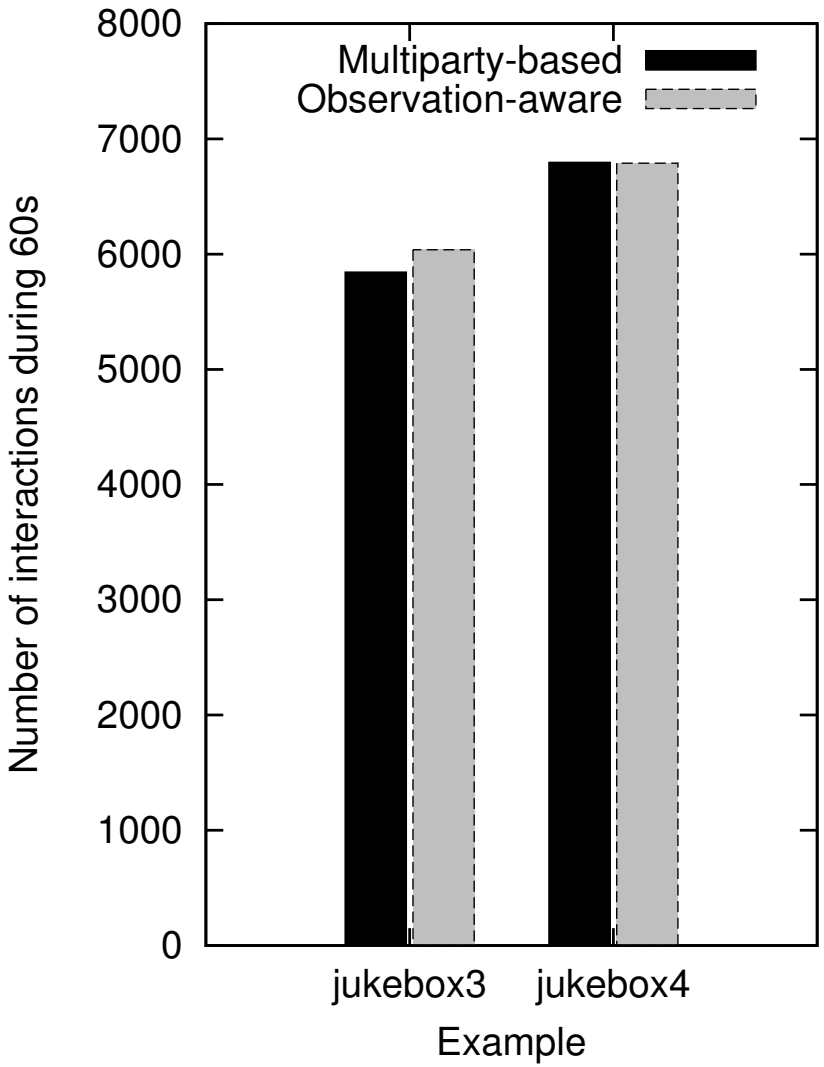
Each reader needs to access disks following a given sequence.

Each jukebox loads and unloads all 3 disks sequentially, in a random order.

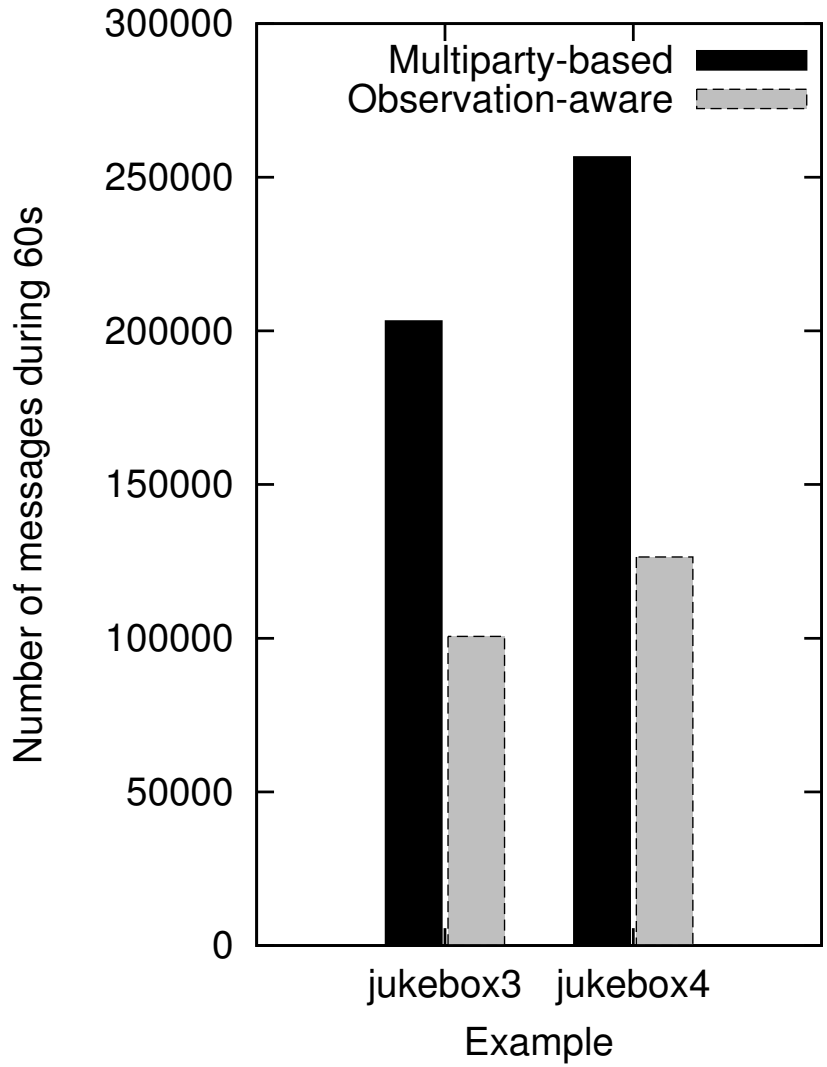
An interaction "read" is possible only if the disk loaded by the jukebox is the disk needed by the reader.

Jukebox

Interactions during 60s.



Messages exchanged during 60s.



Outline

Related Work

Protocols for Multiparty Interaction:

Bagrodia *Process Synchronization: Design and performance evaluation of distributed algorithms.* 1989

Kumar *An implementation of n-party synchronization using tokens.* 1990

Pérez,Corchuelo,Toro *An order-based algorithm for implementing multiparty synchronization* 2004.

Distributed implementation for BIP:

[DC12] **Bonakdarpour, Bozga, Jaber, Quilbeuf, Sifakis** *A framework for automated distributed implementation of component-based models.* 2012

[EMSOFT11] **Bonakdarpour, Bozga, Quilbeuf** *Automated distributed implementation of component-based models with priorities.* 2011

Reducing number of observed components using Knowledge:

[FORTE12] **Bensalem, Bozga, Quilbeuf, Sifakis** *Knowledge-Based distributed conflict resolution for multiparty interactions and priorities.* 2012

Conclusion

- We proposed a new glue operator, Observation, for scheduling multiparty interactions.
- We showed that it is expressive enough to encompass Priority.
- We proposed an Observation-Aware protocol that requires fewer messages than a Multiparty-Based protocol for Observation.
- Experiments showed that this message efficiency can improve performance.

Future Work

- Characterization of the expressiveness allowed by Observation, and comparison with other existing glues.
- Deeper experimental analysis of the new Protocol, in particular when combined with knowledge-based methods:
  - to reduce number of observed components [FORTE12]
  - to optimize the protocol itself [HVC'12]

Thank You