Towards Distributed Reo

José Proença

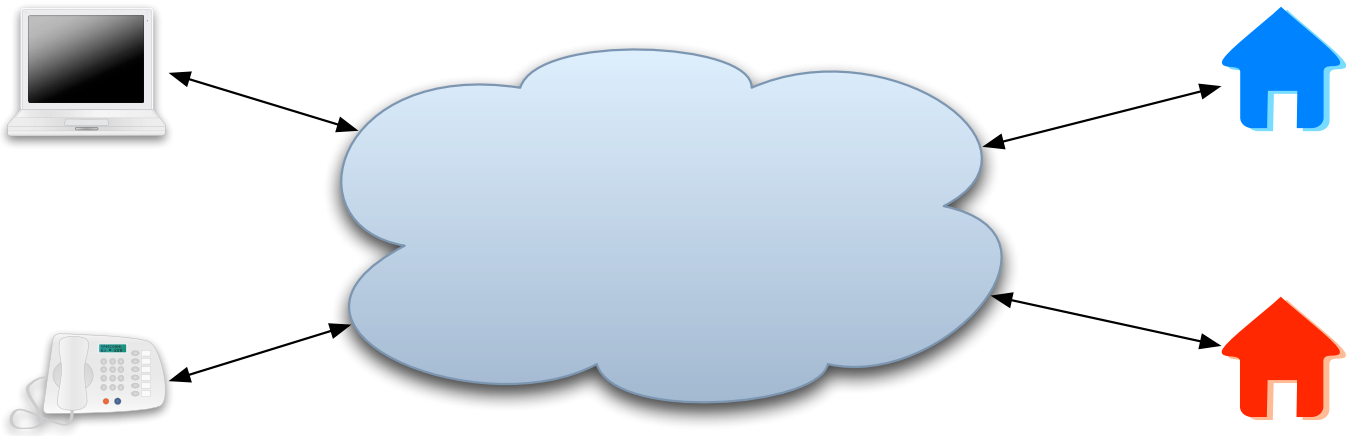
Centrum voor Wiskunde en Informatica

CIC, 2007

Outline

Outline

Motivation

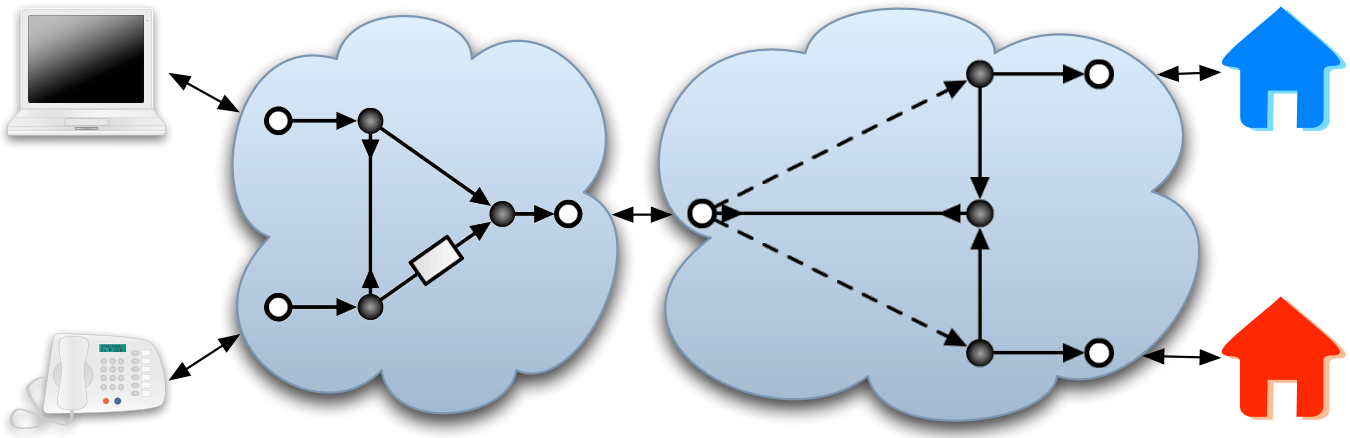


Coordination:

- How to implement it?
- Where to run it?

Distributed Coordination

Motivation

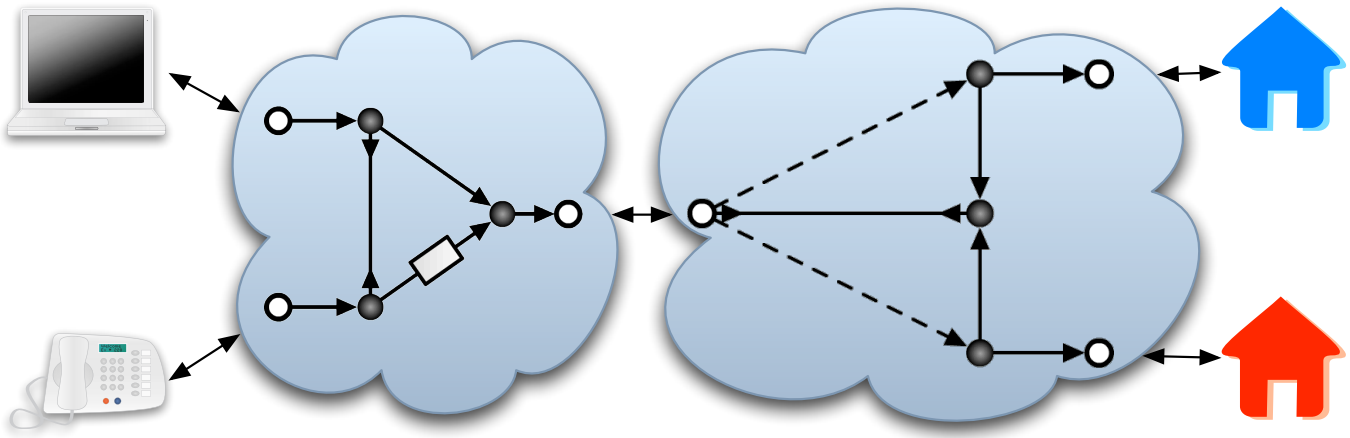


Coordination:

- How to implement it?
- Where to run it?

Distributed Coordination

Motivation



Coordination:

- How to implement it?
- Where to run it?

Distributed Coordination

Architecture

Designer

Deployment Resolver

Local Optimization

Instantiator

Kernel

Architecture

Designer

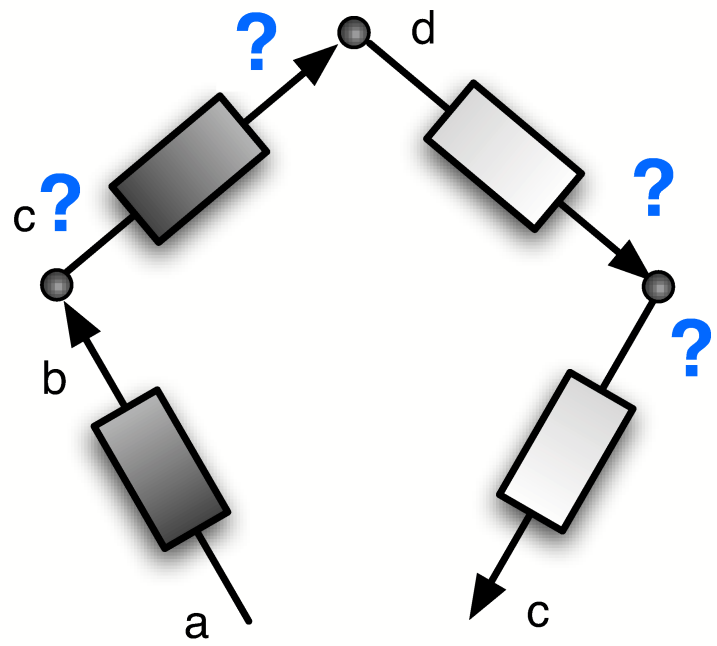
Use of tools, such as a GUI

Deployment Resolver

Local Optimization

Instantiator

Kernel



Architecture

Designer

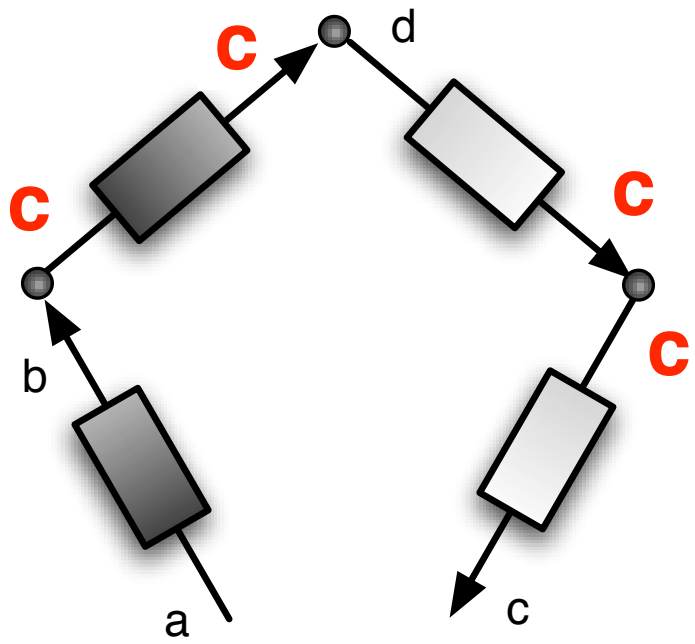
Deployment Resolver

Unspecified locations are resolved. Constraints and policies need to be considered.

Local Optimization

Instantiator

Kernel



Architecture

Designer

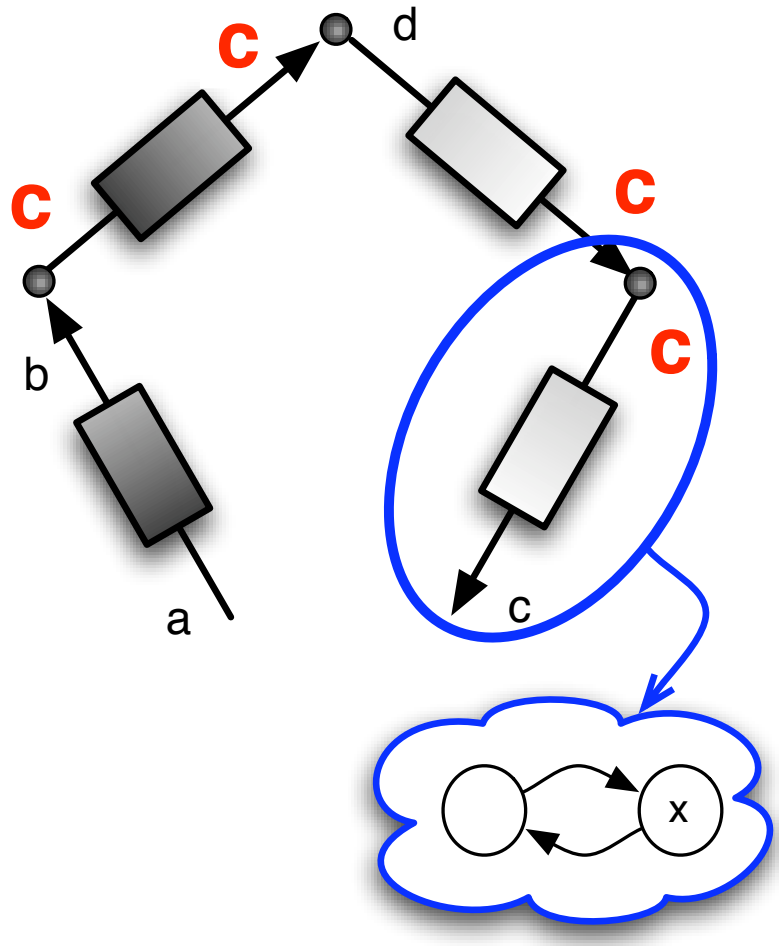
Deployment Resolver

Local Optimization

Plugins: | CA | CC | CSP |

Instantiator

Kernel



Architecture

Designer

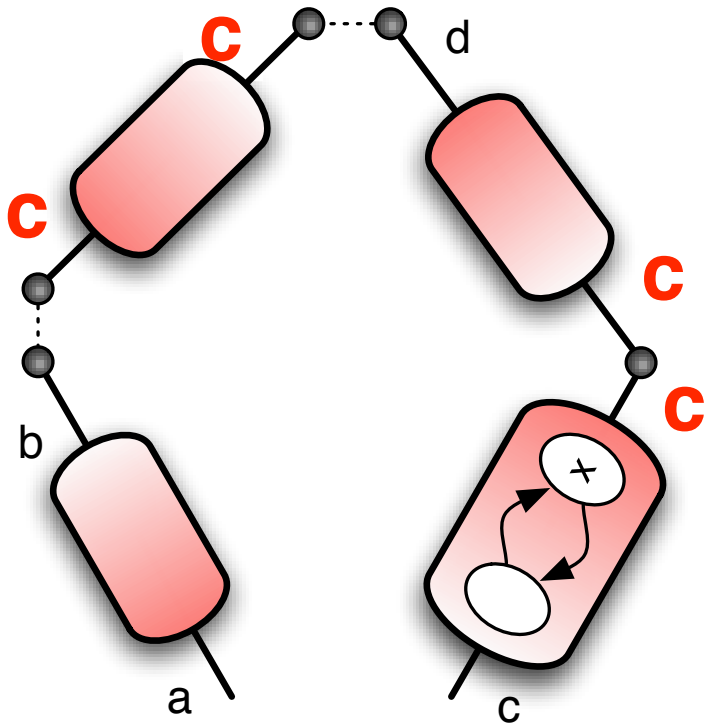
Deployment Resolver

Local Optimization

Instantiator

Creation of primitives

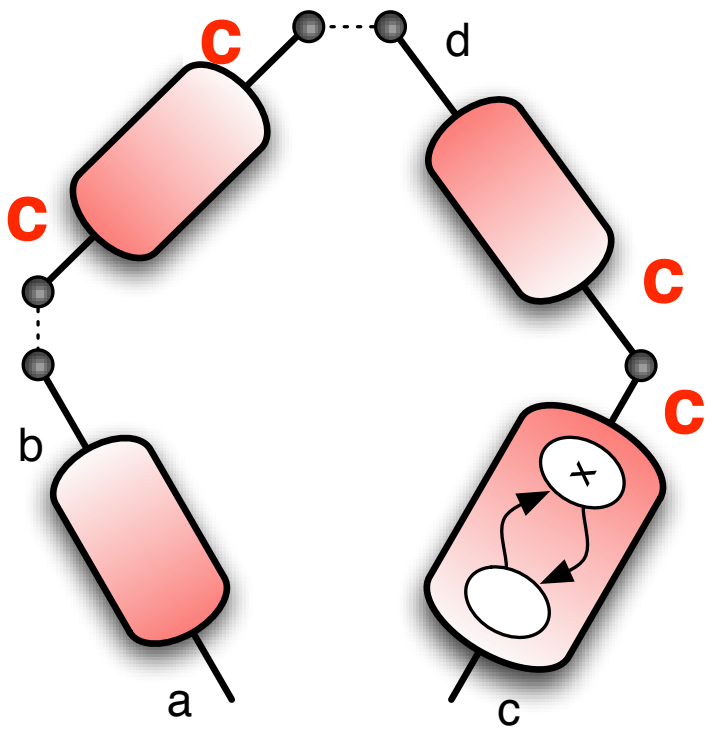
Kernel



Architecture

- Designer
- Deployment Resolver
- Local Optimization
- Instantiator
- **Kernel**

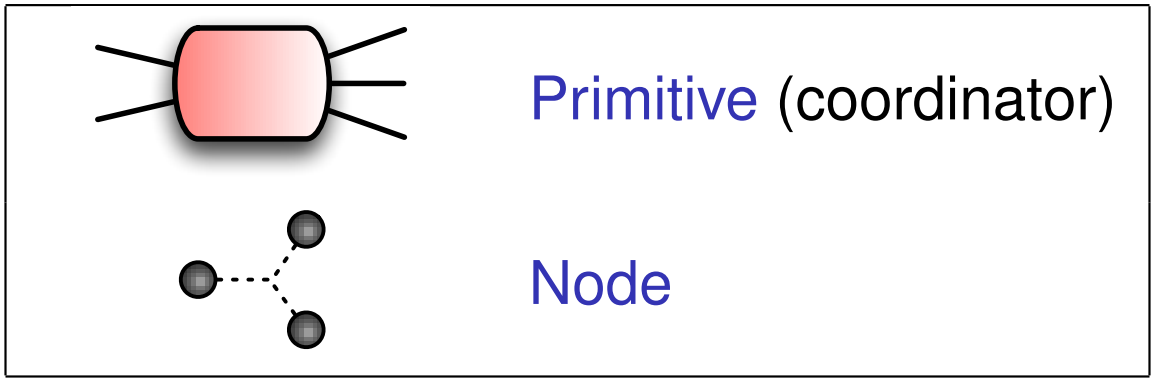
  Execution of the engine



Outline

Distributed Reo

Distributed: deals with partial knowledge.

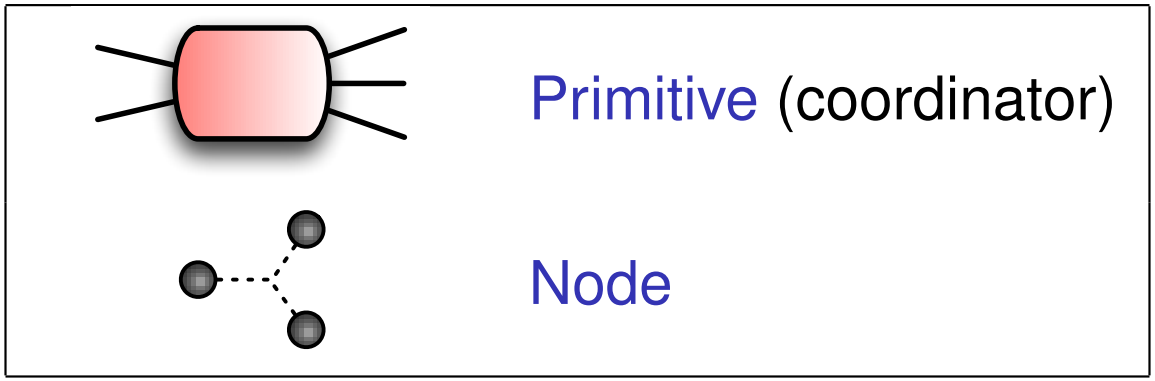


Implemention: Scala language

Integrates features of object-oriented and functional languages;
Fully interoperable with Java;
Actor model for communication.

Distributed Reo

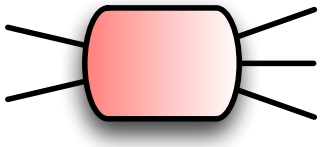
Distributed: deals with partial knowledge.



Implemention: Scala language

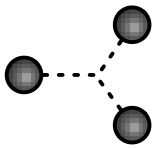
Integrates features of object-oriented and functional languages;
Fully interoperable with Java;
Actor model for communication.

Distributed Reo
Primitives and nodes



Each port has a *location*.
Must react to some messages:

- Request Behabiour
- Reply Behaviour
- Refuse (reason)
- Give Behaviour & Request/Give Data
- Reply Data

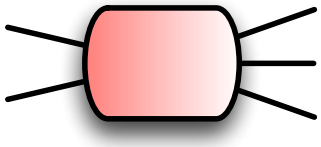


Can be seen as a particular case of a primitive that:

- has no state;
- can be distributed;
- propagates synchronous constraints.

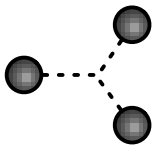
Distributed Reo

Primitives and nodes



Each port has a *location*.
Must react to some messages:

- Request Behabiour
- Reply Behaviour
- Refuse (reason)
- Give Behaviour & Request/Give Data
- Reply Data



Can be seen as a particular case of a primitive that:

- has no state;
- can be distributed;
- propagates synchronous constraints.

Behaviour
What is it?

What each primitive can do

Which end points can flow data, and relation between data flowing in the end points.

Join of behaviours

Given the behaviour of two primitives, the behaviour of the composition of both can also be obtained.

Example: Connector Colouring

- Colouring tables provide the behaviour of each primitive;
- Join of colouring tables is defined.

Behaviour
What is it?

What each primitive can do

Which end points can flow data, and relation between data flowing in the end points.

Join of behaviours

Given the behaviour of two primitives, the behaviour of the composition of both can also be obtained.

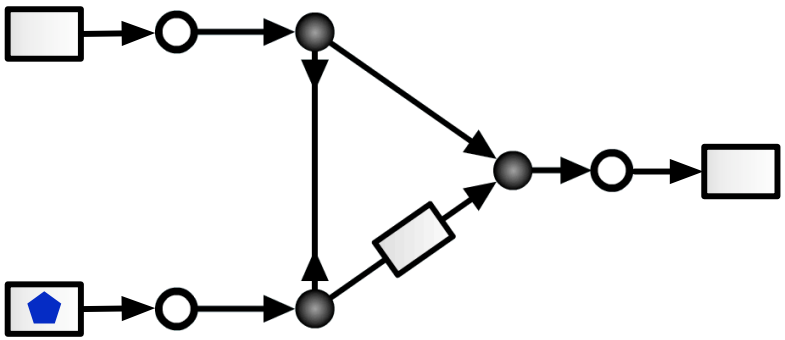
Example: Connector Colouring

- Colouring tables provide the behaviour of each primitive;
- Join of colouring tables is defined.

Outline

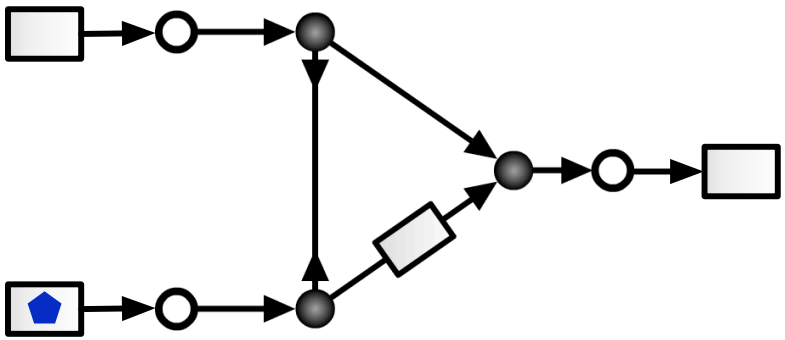
Commit to a behaviour

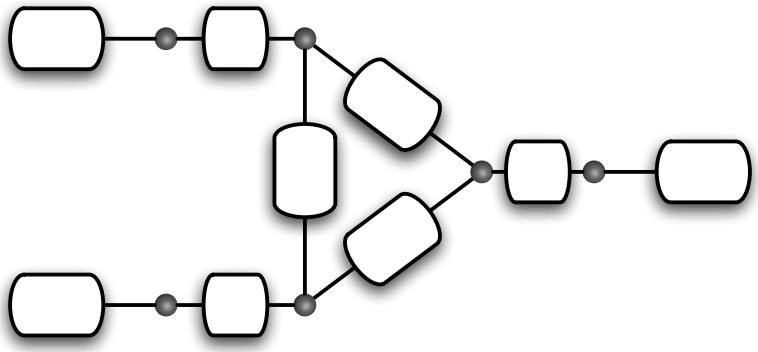
Locations are not relevant;
Assume partial knowledge (know only neighbours);
Two phase algorithm: Negotiation and Communication.



Commit to a behaviour

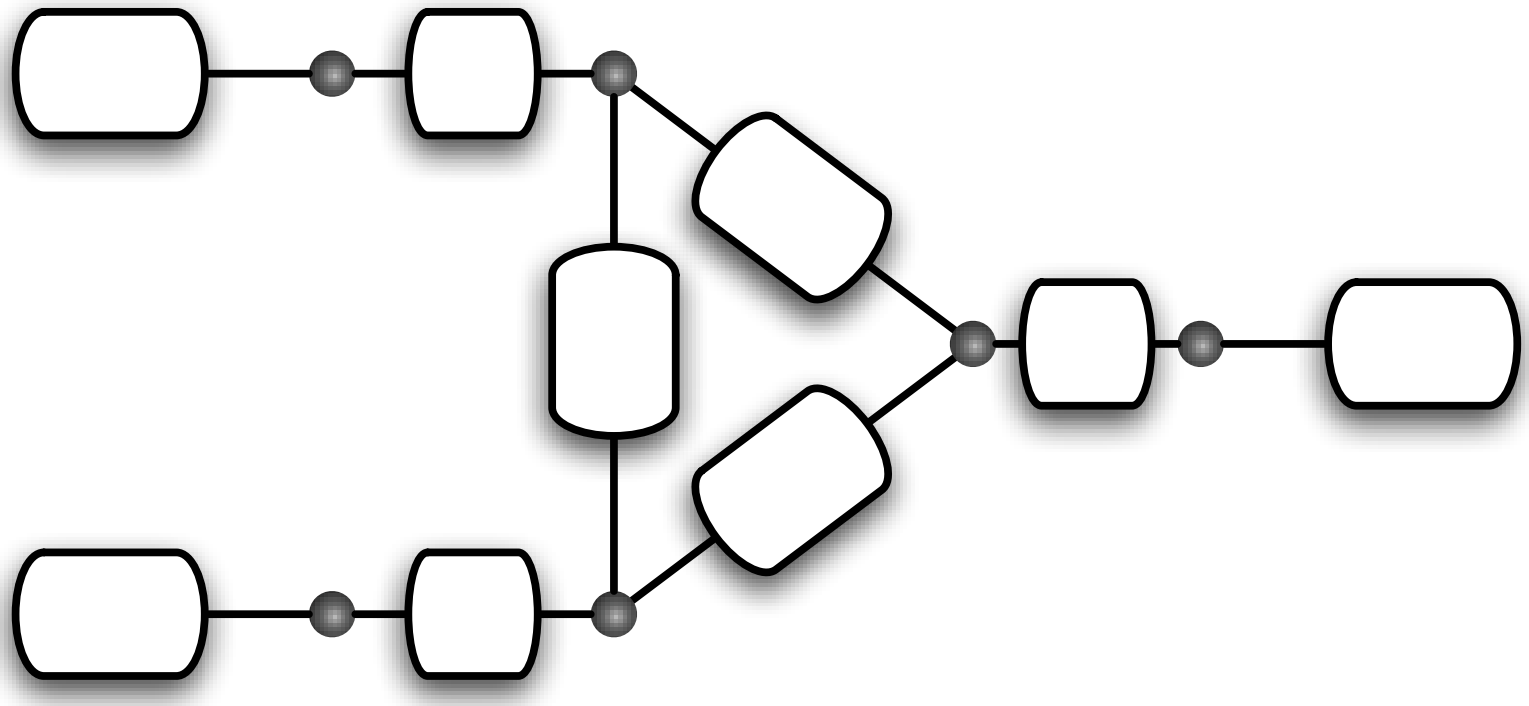
Locations are not relevant;
Assume partial knowledge (know only neighbours);
Two phase algorithm: Negotiation and Communication.





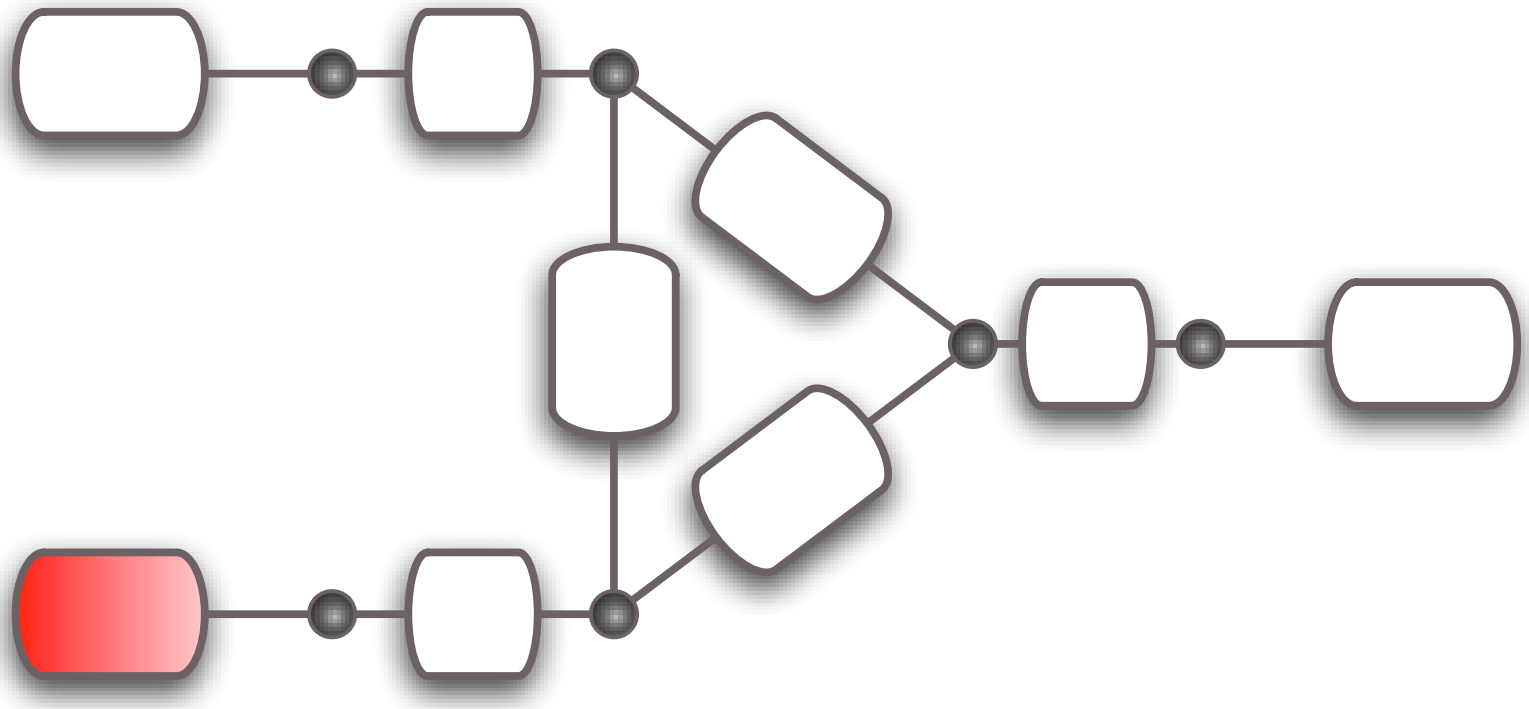
Commit to a behaviour

Basic case



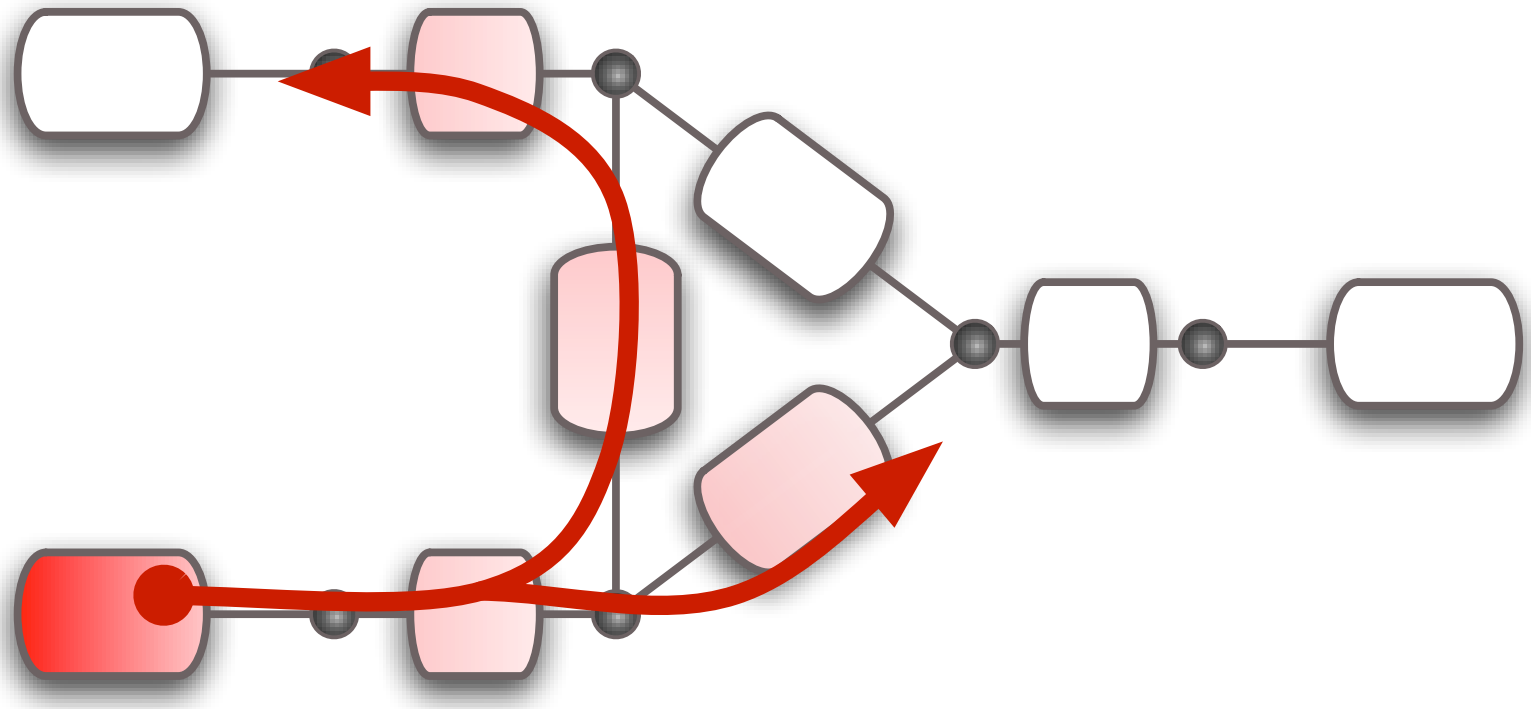
Commit to a behaviour

Basic case



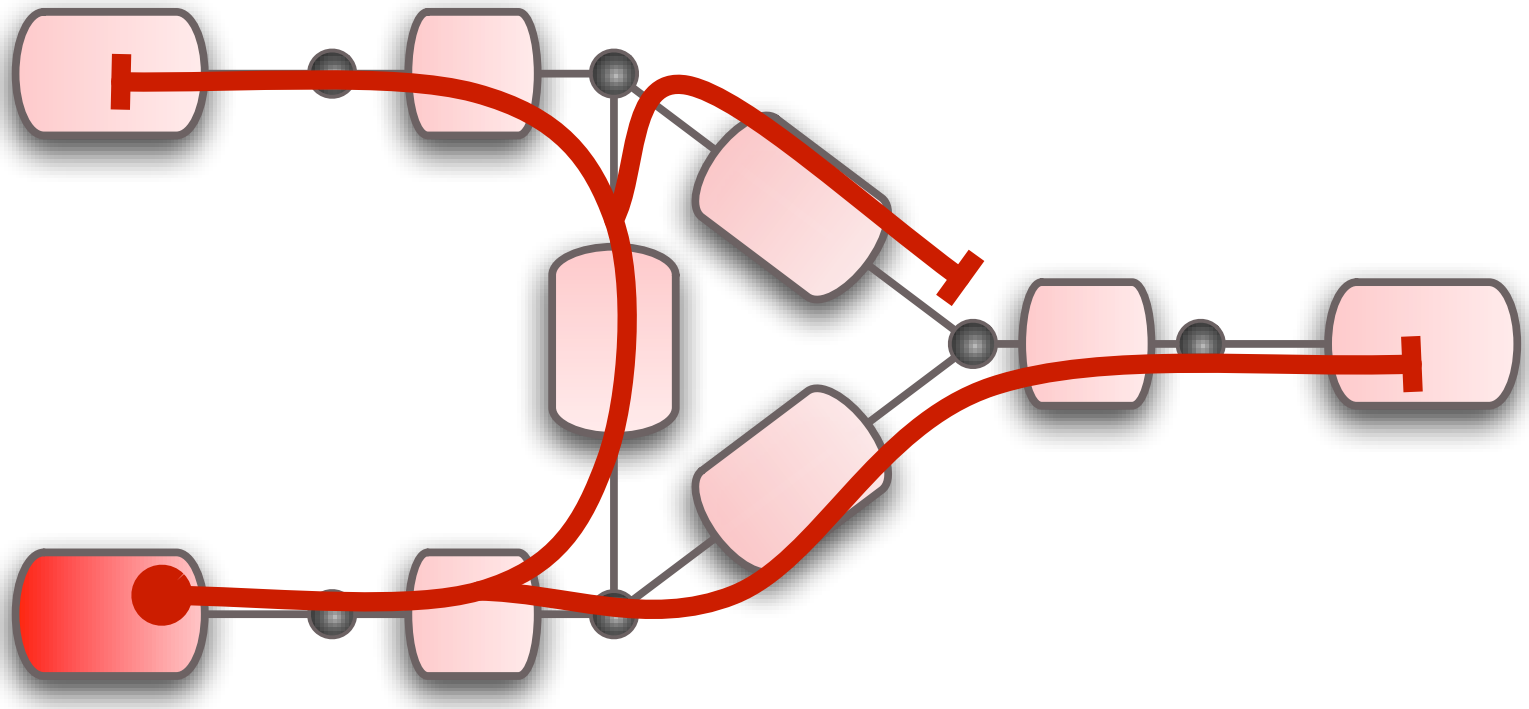
Commit to a behaviour

Basic case



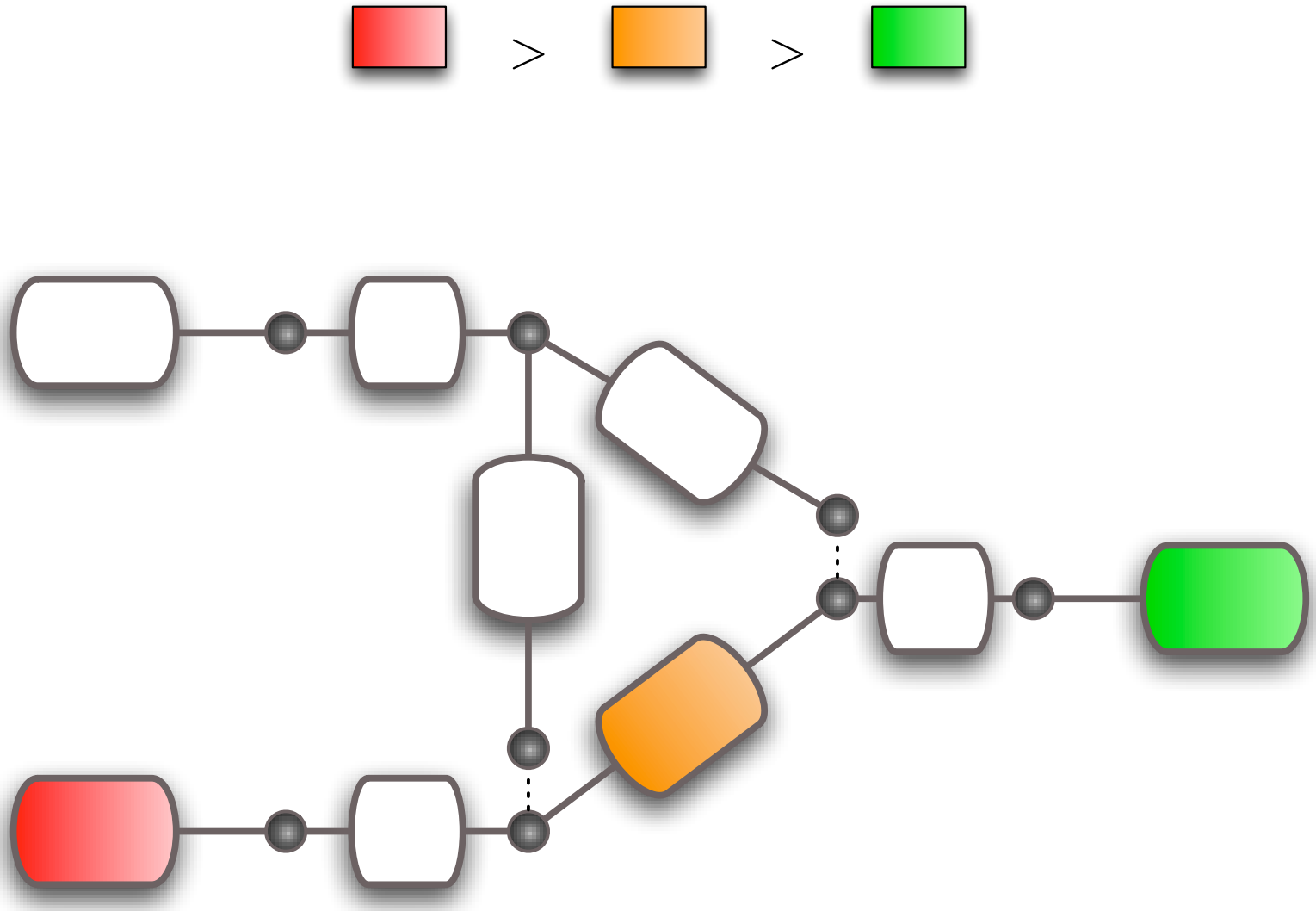
Commit to a behaviour

Basic case



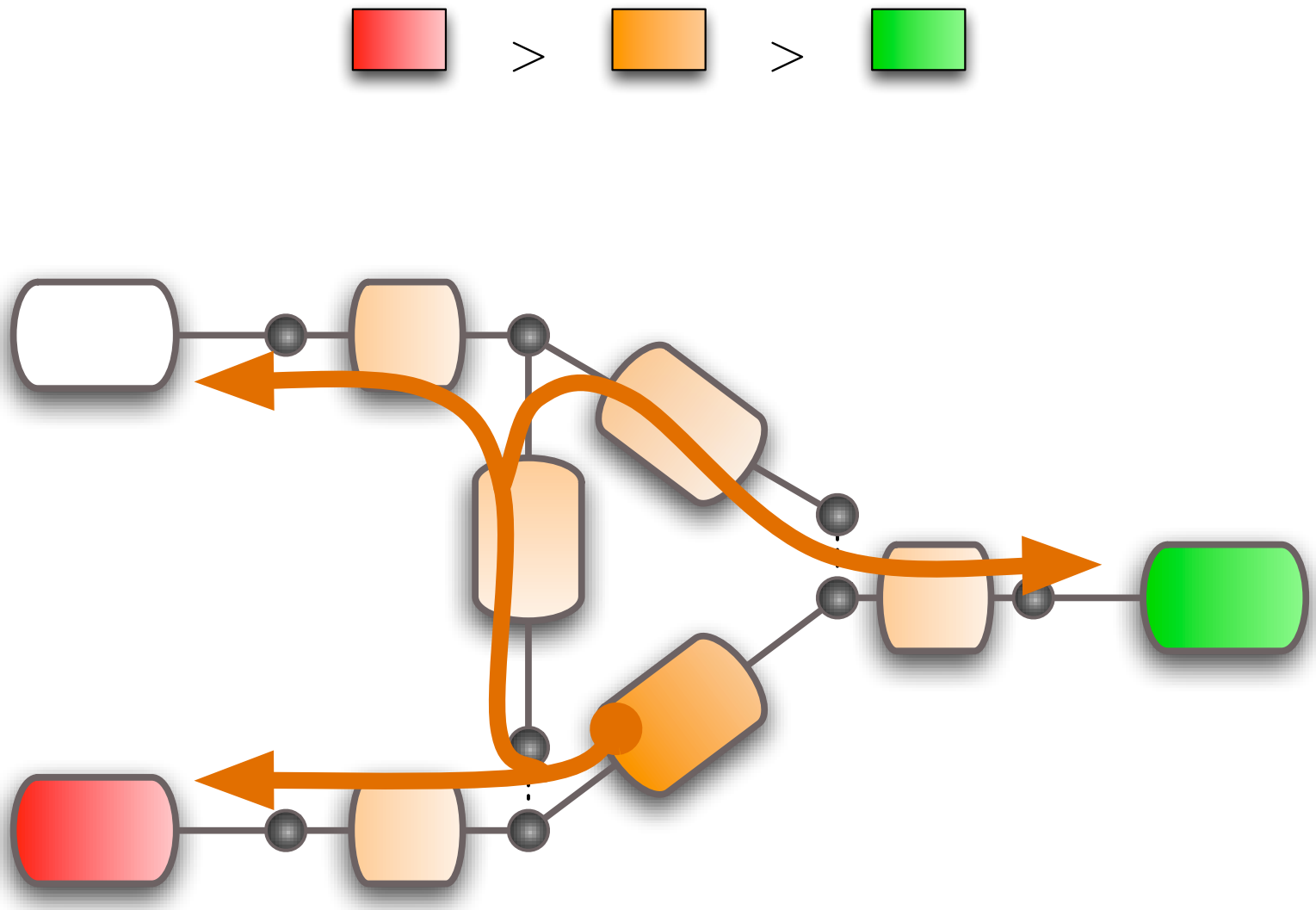
Commit to a behaviour

Multiple starting points



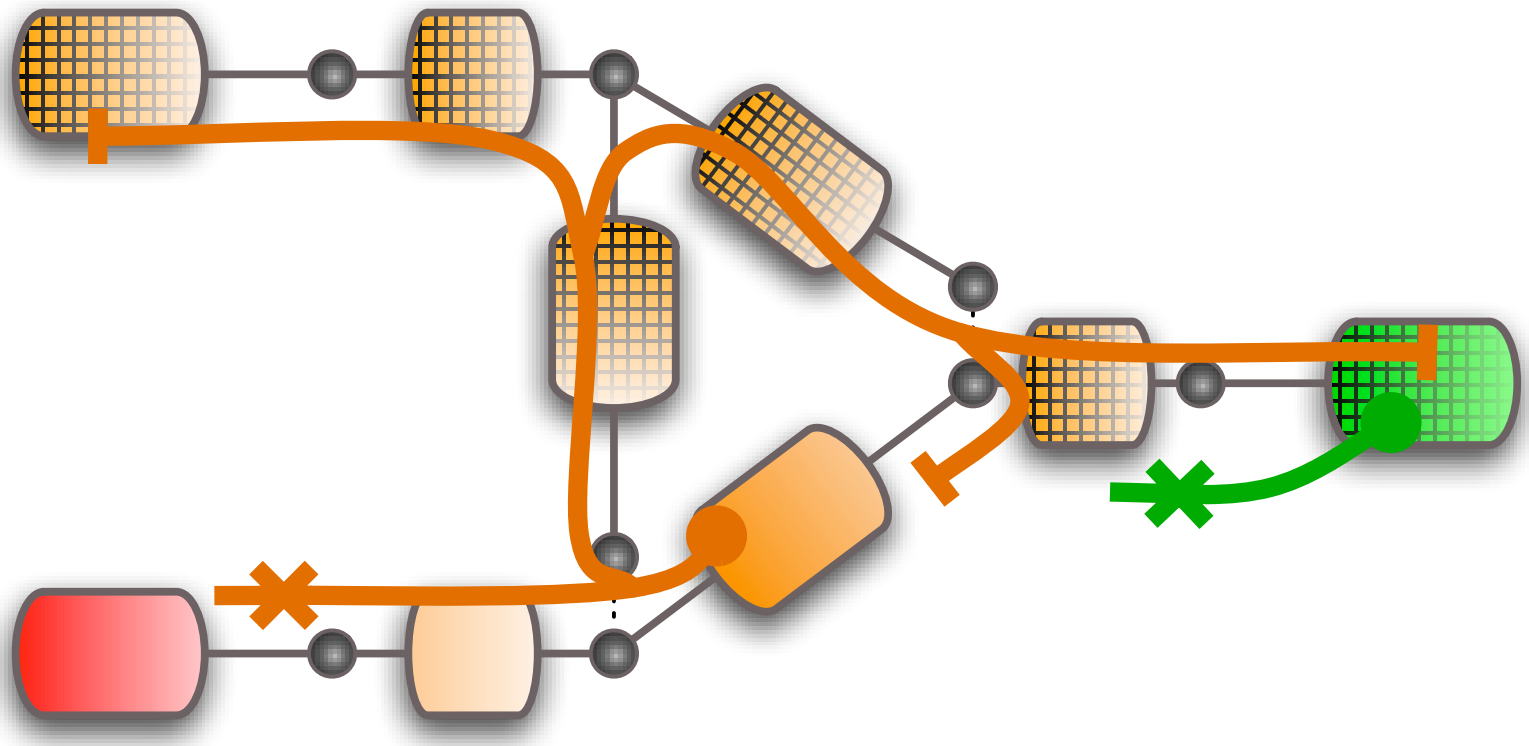
Commit to a behaviour

Multiple starting points



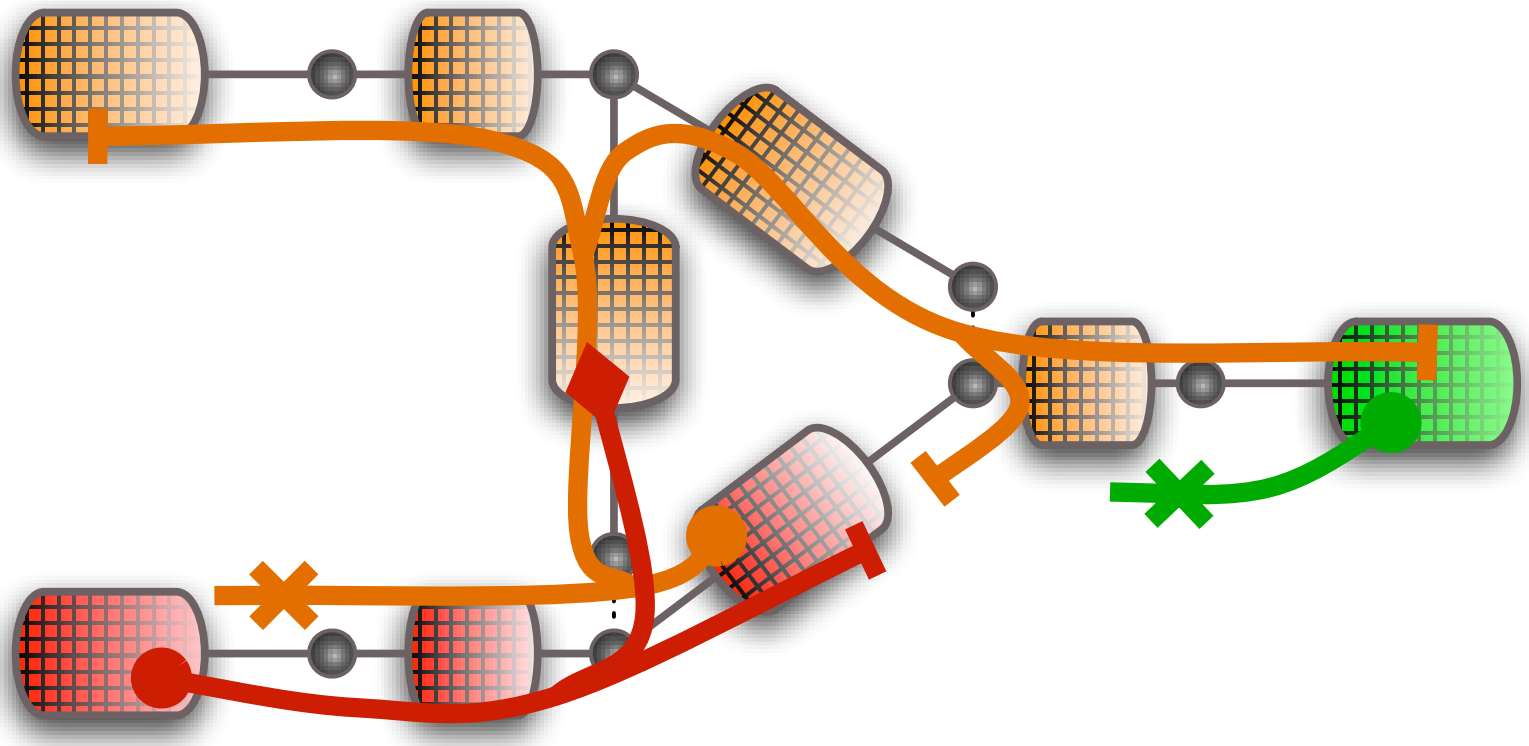
Commit to a behaviour

Multiple starting points



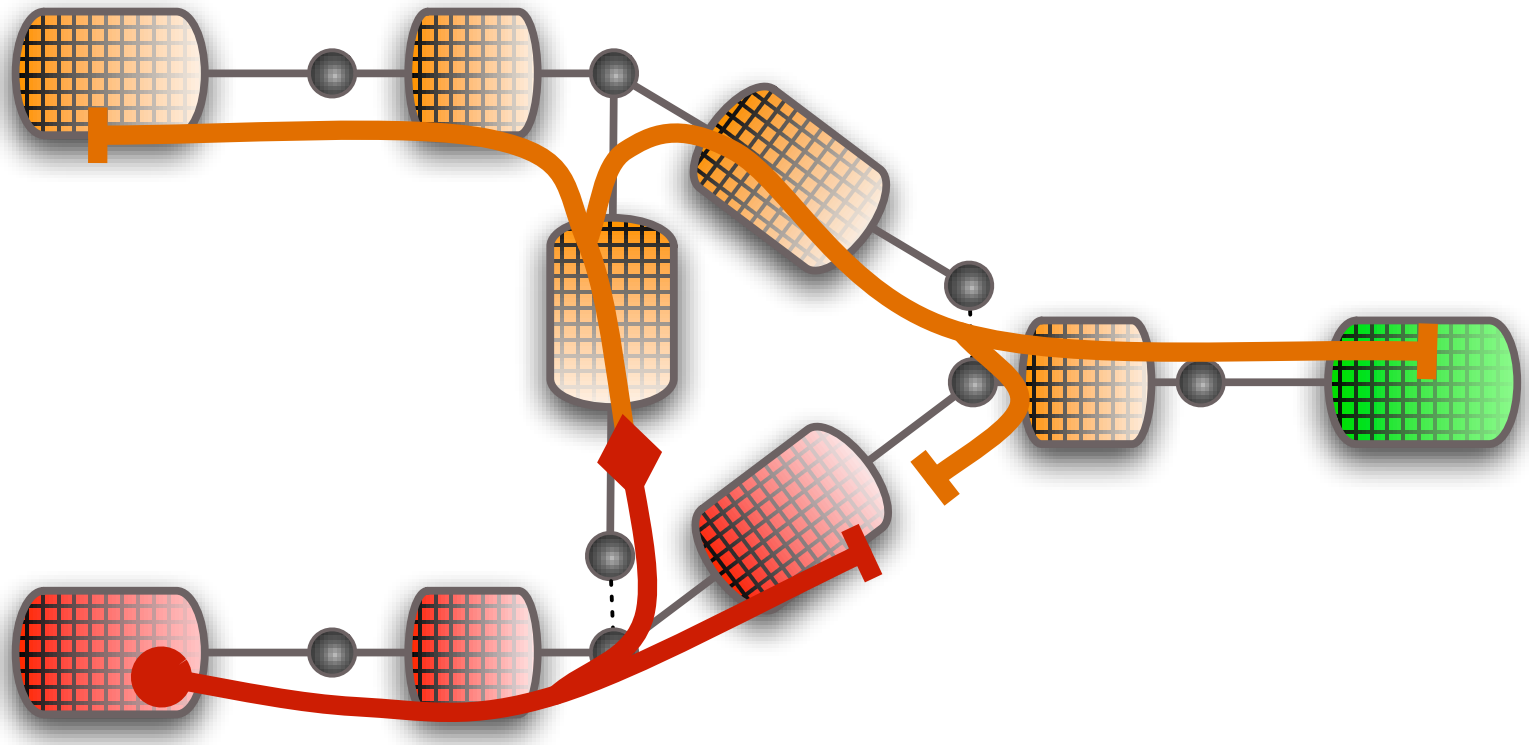
Commit to a behaviour

Multiple starting points



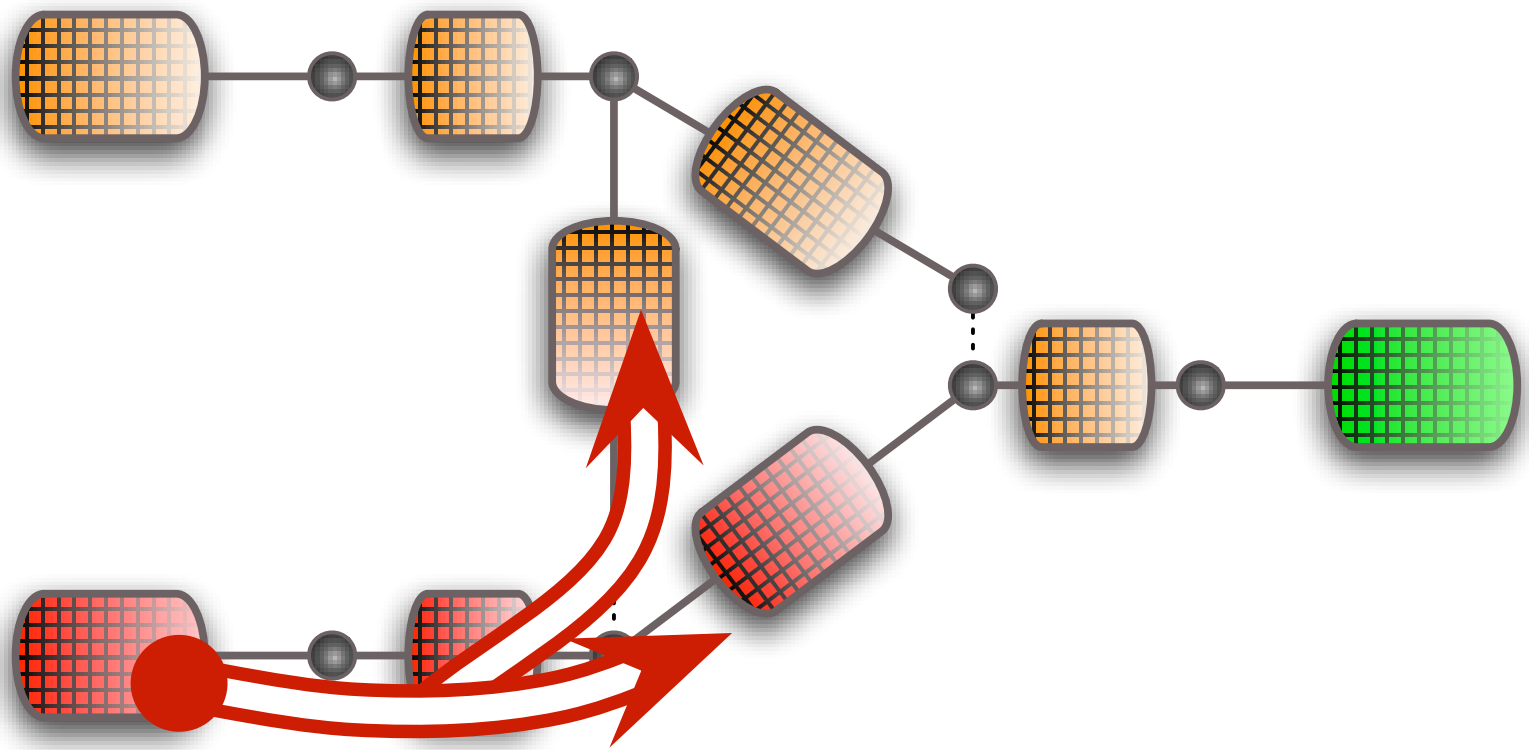
Commit to a behaviour

Multiple starting points



Commit to a behaviour

Multiple starting points



Outline

Conclusions

- Common architecture to include design and implementation;
- Implementation platform, where each (distributed) element knows only about its own neighbours;
- Resolve synchrony constraints (imposed by Reo) using asynchronous messages;
- The kernel supports messages for other purposes:
  - Fail/Abort;
  - Suspend – to allow reconfiguration.
  - ...
- How to determine the rank of the *inititiators*?
- A primitive(s) can be obtained from other coordination models other than Reo (*e.g.*, Orc);
- Allow unification of more coordination models;

Conclusions

- Common architecture to include design and implementation;
- Implementation platform, where each (distributed) element knows only about its own neighbours;
- Resolve synchrony constraints (imposed by Reo) using asynchronous messages;
- The kernel supports messages for other purposes:
  - Fail/Abort;
  - Suspend – to allow reconfiguration.
  - ...
- How to determine the rank of the *inititiators*?
- A primitive(s) can be obtained from other coordination models other than Reo (*e.g.*, Orc);
- Allow unification of more coordination models;

Conclusions

- Common architecture to include design and implementation;
- Implementation platform, where each (distributed) element knows only about its own neighbours;
- Resolve synchrony constraints (imposed by Reo) using asynchronous messages;
- The kernel supports messages for other purposes:
  - Fail/Abort;
  - Suspend – to allow reconfiguration.
  - . . .
- How to determine the rank of the *inititiators*?
- A primitive(s) can be obtained from other coordination models other than Reo (*e.g.*, Orc);
- Allow unification of more coordination models;

Conclusions

- Common architecture to include design and implementation;
- Implementation platform, where each (distributed) element knows only about its own neighbours;
- Resolve synchrony constraints (imposed by Reo) using asynchronous messages;
- The kernel supports messages for other purposes:
  - Fail/Abort;
  - Suspend – to allow reconfiguration.
  - . . .
- How to determine the rank of the *inititiators*?
- A primitive(s) can be obtained from other coordination models other than Reo (*e.g.*, Orc);
- Allow unification of more coordination models;