Systematic Development of Distributed Algorithms using Event-B Experiences, reviews and prospects

Dominique Méry
Telecom Nancy,Université de Lorraine
dominique.mery@loria.fr

ABZ @TOKYO , Japan, May 20, 2026

supported by ANR EBRP Plus

Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Distributed Algorithms as a Field of Experimentation

Distributed Algorithms as a Field of Experimentation

- Distributed algorithms are difficult to understand **how** and **why** they work properly.

Distributed Algorithms as a Field of Experimentation

- Distributed algorithms are difficult to understand **how** and **why** they work properly.
- Distributed algorithms are used in **critical systems** that require very strong correctness properties.

Distributed Algorithms as a Field of Experimentation

- Distributed algorithms are difficult to understand **how** and **why** they work properly.
- Distributed algorithms are used in **critical systems** that require very strong correctness properties.
- Application 1: I write a mail message and just click! That's it! And it works!
- Application 2: I pay my bill and use my credit card . . . Just confirm my password. That's it! And it works!
- Application 3: I got a confimation from my neighboors that's I win the election and That's it! And it works!

Distributed algorithms as players on graphs

Distributed algorithms as players on graphs

- Messages may disappear . . .

Distributed algorithms as players on graphs

- Messages may disappear . . .
- Messages may be modified . . .

Distributed algorithms as players on graphs

- Messages may disappear . . .
- Messages may be modified . . .
- Messages may be added . . .

Distributed algorithms as players on graphs

- Messages may disappear . . .
- Messages may be modified . . .
- Messages may be added . . .
- Messages may be cyclically circulating . . .

Distributed algorithms as players on graphs

- Messages may disappear ...
- Messages may be modified ...
- Messages may be added ...
- Messages may be cyclically circulating ...
- Peers should share, exchange, ...

Distributed algorithms as players on graphs

- Messages may disappear ...
- Messages may be modified ...
- Messages may be added ...
- Messages may be cyclically circulating ...
- Peers should share, exchange, ...
- Local informations help to get global informations

Distributed algorithms as players on graphs

- Messages may disappear ...
- Messages may be modified ...
- Messages may be added ...
- Messages may be cyclically circulating ...
- Peers should share, exchange, ...
- Local informations help to get global informations
- Environment may be critical and hostile

Distributed algorithms as players on graphs

- Messages may disappear ...
- Messages may be modified ...
- Messages may be added ...
- Messages may be cyclically circulating ...
- Peers should share, exchange, ...
- Local informations help to get global informations
- Environment may be critical and hostile

Modelling and verifying distributed algorithms

Modelling and verifying distributed algorithms

- Distributed algorithms as transition systems over local states (see G. Tel)

Modelling and verifying distributed algorithms

- Distributed algorithms as transition systems over local states (see G. Tel)
- Properties of transition sysrtems : safety, liveness, fairness, temporal properties, ...

Modelling and verifying distributed algorithms

- Distributed algorithms as transition systems over local states (see G. Tel)
- Properties of transition sysrtems : safety, liveness, fairness, temporal properties, ...
- Requirements: reliability of communications, sharing

Modelling and verifying distributed algorithms

- Distributed algorithms as transition systems over local states (see G. Tel)
- Properties of transition sysrtems : safety, liveness, fairness, temporal properties, ...
- Requirements: reliability of communications, sharing ressources, localisation of informations, graph properties, ...

Modelling and verifying distributed algorithms

- Distributed algorithms as transition systems over local states (see G. Tel)
- Properties of transition sysrtems : safety, liveness, fairness, temporal properties, ...
- Requirements: reliability of communications, sharing ressources, localisation of informations, graph properties, ...
- Modelling languages for distributed algorithms as guarded commands, action systems, I/O automata, UNITY, TLA$^+$, Graph Rewriting, ...
- Distributed programming languages as MPI-based programming or as coordination-based programming languages ...
- Verification of distributed algorithms : compositionality, R/G, ...

Modelling and verifying distributed algorithms

- Distributed algorithms as transition systems over local states (see G. Tel)
- Properties of transition sysrtems : safety, liveness, fairness, temporal properties, ...
- Requirements: reliability of communications, sharing ressources, localisation of informations, graph properties, ...
- Modelling languages for distributed algorithms as guarded commands, action systems, I/O automata, UNITY, TLA$^+$, Graph Rewriting, ...
- Distributed programming languages as MPI-based programming or as coordination-based programming languages ...
- Verification of distributed algorithms : compositionality, R/G, ... using model checking or proof assistants.

Correctness by Construction

Correctness by Construction

- Correctness by Construction is a method of building software -based systems with **demonstrable correctness** for security- and safety-critical applications.

Correctness by Construction

- Correctness by Construction is a method of building software -based systems with **demonstrable correctness** for security- and safety-critical applications.
- Correctness by Construction advocates a **step-wise refinement** process from specification to code using tools for checking and transforming models.

Correctness by Construction

- Correctness by Construction is a method of building software -based systems with **demonstrable correctness** for security- and safety-critical applications.
- Correctness by Construction advocates a **step-wise refinement** process from specification to code using tools for checking and transforming models.
- Correctness by Construction is an approach to distributed algorithms or distributed applications

Correctness by Construction

- Correctness by Construction is a method of building software -based systems with **demonstrable correctness** for security- and safety-critical applications.
- Correctness by Construction advocates a **step-wise refinement** process from specification to code using tools for checking and transforming models.
- Correctness by Construction is an approach to distributed algorithms or distributed applications
  - verification of distributed algorithms.

Correctness by Construction

- Correctness by Construction is a method of building software -based systems with **demonstrable correctness** for security- and safety-critical applications.
- Correctness by Construction advocates a **step-wise refinement** process from specification to code using tools for checking and transforming models.
- Correctness by Construction is an approach to distributed algorithms or distributed applications
  - verification of distributed algorithms.
  - starting from the service and peeling back the layers of the ISO by refinement.

Correctness by Construction

- Correctness by Construction is a method of building software -based systems with **demonstrable correctness** for security- and safety-critical applications.
- Correctness by Construction advocates a **step-wise refinement** process from specification to code using tools for checking and transforming models.
- Correctness by Construction is an approach to distributed algorithms or distributed applications
  - verification of distributed algorithms.
  - starting from the service and peeling back the layers of the ISO by refinement.
  - to highlight a simple explanation that helps us understand why and how the algorithm worked.

Correctness by Construction

- Correctness by Construction is a method of building software -based systems with **demonstrable correctness** for security- and safety-critical applications.
- Correctness by Construction advocates a **step-wise refinement** process from specification to code using tools for checking and transforming models.
- Correctness by Construction is an approach to distributed algorithms or distributed applications
  - verification of distributed algorithms.
  - starting from the service and peeling back the layers of the ISO by refinement.
  - to highlight a simple explanation that helps us understand why and how the algorithm worked.
  - to (re)develop distributed algorithms using Event-B, then to translate the Event-B models into TLA$^+$ to validate the invariant using TLAPS, and to express the liveness properties

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Distributed Algorithms: first steps in verification

First steps in a new world by proving the mutual exclusion algorithm of Ricart and Agrawala using a sound and semantically complete temporal proof system and a graphical notation called proof lattice

Distributed Algorithms: first steps in verification

First steps in a new world by proving the mutual exclusion algorithm of Ricart and Agrawala using a sound and semantically complete temporal proof system and a graphical notation called proof lattice



Ricart, Glenn; Agrawala, Ashok K. (1 January 1981). "An optimal algorithm for mutual exclusion in computer networks". Communications of the ACM. 24 (1): 9–17.

Distributed Algorithms: first steps in verification

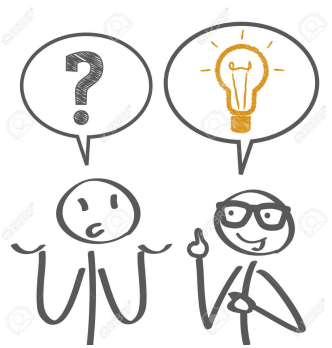
First steps in a new world by proving the mutual exclusion algorithm of Ricart and Agrawala using a sound and semantically complete temporal proof system and a graphical notation called proof lattice



Ricart, Glenn; Agrawala, Ashok K. (1 January 1981). "An optimal algorithm for mutual exclusion in computer networks". Communications of the ACM. 24 (1): 9–17.

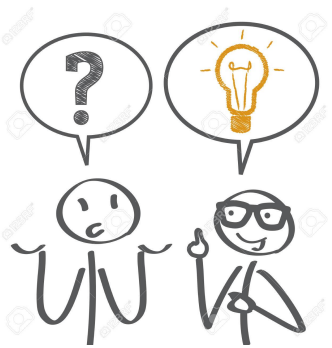
- Definition of sound and semantically complete temporal proof system.
- Annotations and proofs were not machine-assisted.
- How to explain why the algorithm was correct?
- Carvalho and Roucairol published an improvement of the RA algorithm using some kind of abstraction and simplification.

Verifying Distributed Algorithms: lessons learnt



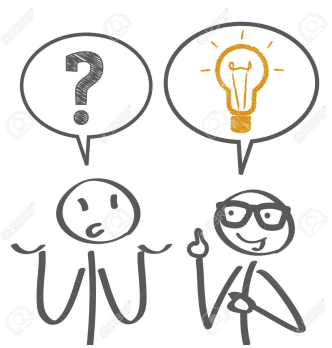
Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:



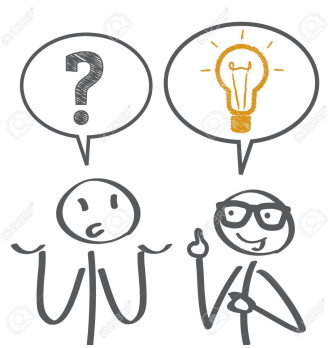
Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:
  - local annotation but global state



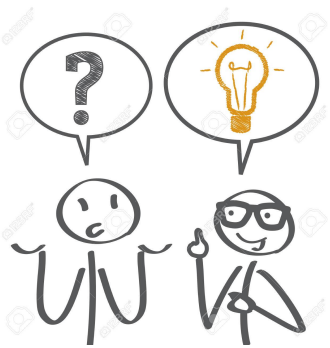
Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...



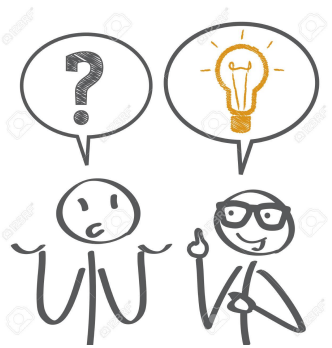
Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)

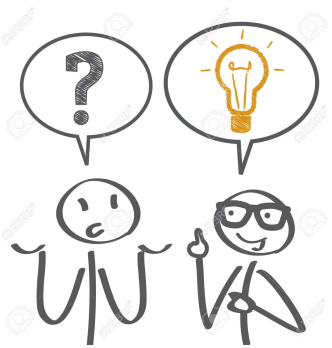


Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions

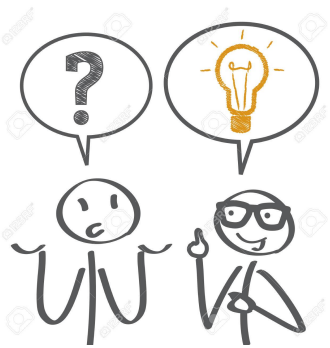


Verifying Distributed Algorithms: lessons learnt



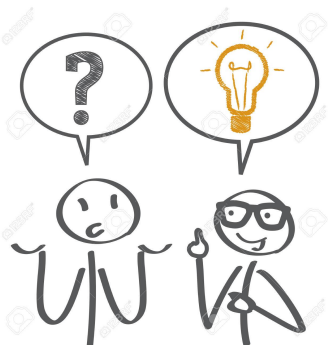
- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process

Verifying Distributed Algorithms: lessons learnt



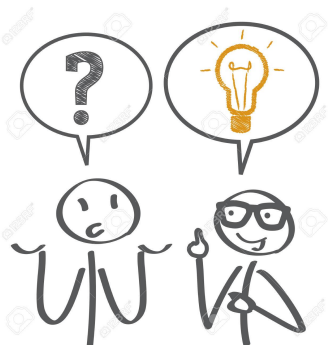
- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process
- Discovering why the distributed system is working

Verifying Distributed Algorithms: lessons learnt



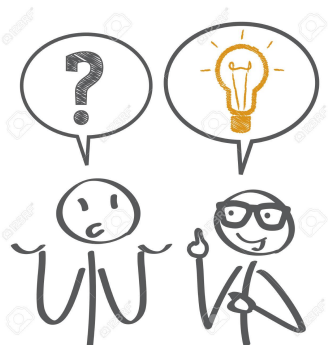
- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process
- Discovering why the distributed system is working
- Explaining in an abstract and simple way why it is working

Verifying Distributed Algorithms: lessons learnt



- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process
- Discovering why the distributed system is working
- Explaining in an abstract and simple way why it is working
- Crocos was an integrated environment for interactive verification of SDL specifications (CAV 1992) using Isabelle and Concerto

Verifying Distributed Algorithms: lessons learnt



- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process
- Discovering why the distributed system is working
- Explaining in an abstract and simple way why it is working
- Crocos was an integrated environment for interactive verification of SDL specifications (CAV 1992) using Isabelle and Concerto
- Discovering why the *distributed* process is correct by a simple **abstraction**.

Distributed Algorithms: using proof assistant



Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Distributed Algorithms: using proof assistant



Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

- Using the PVS proof assistant

Distributed Algorithms: using proof assistant



Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

- Using the PVS proof assistant
- Modelling in I/O automata

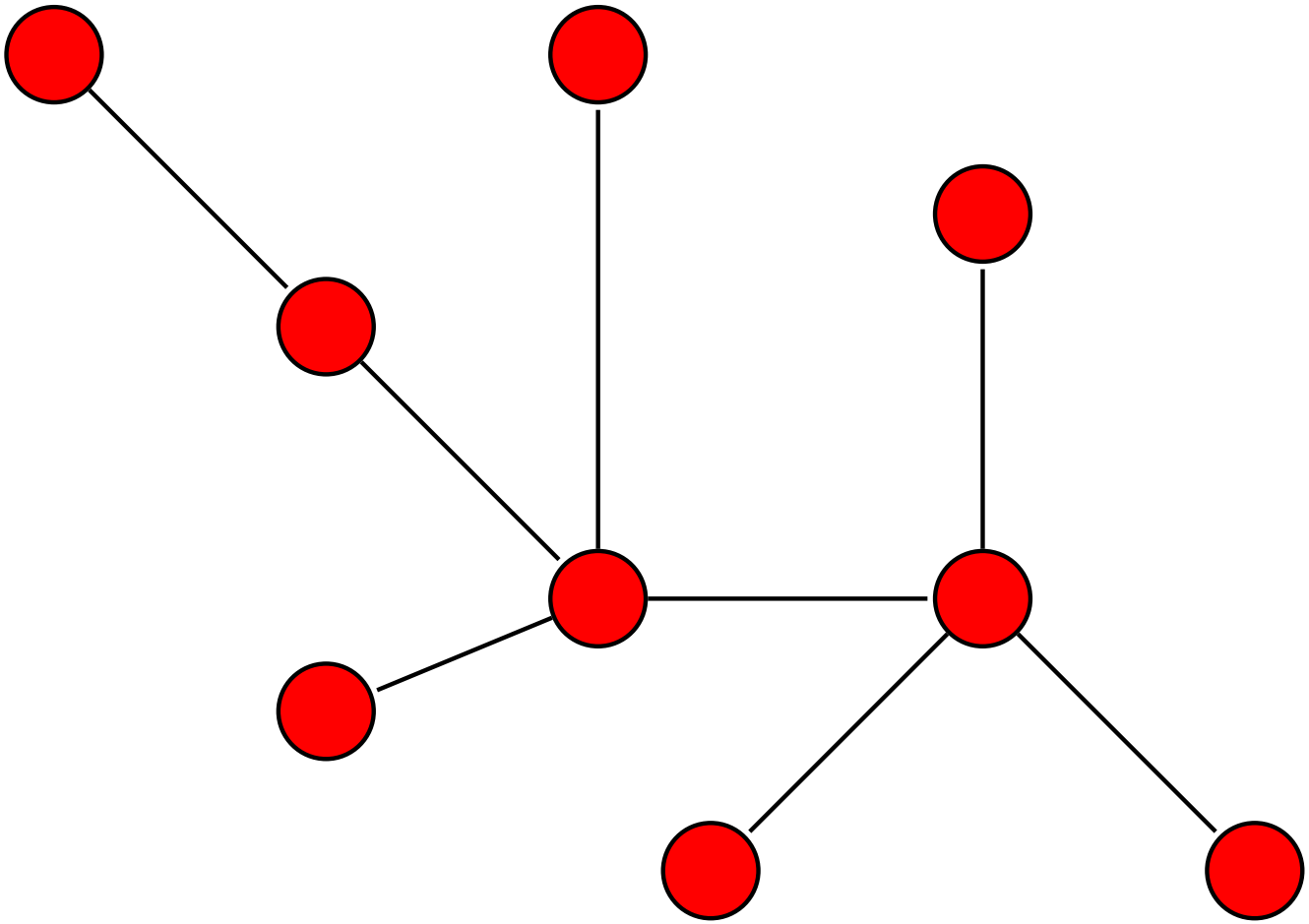
Distributed Algorithms: using proof assistant



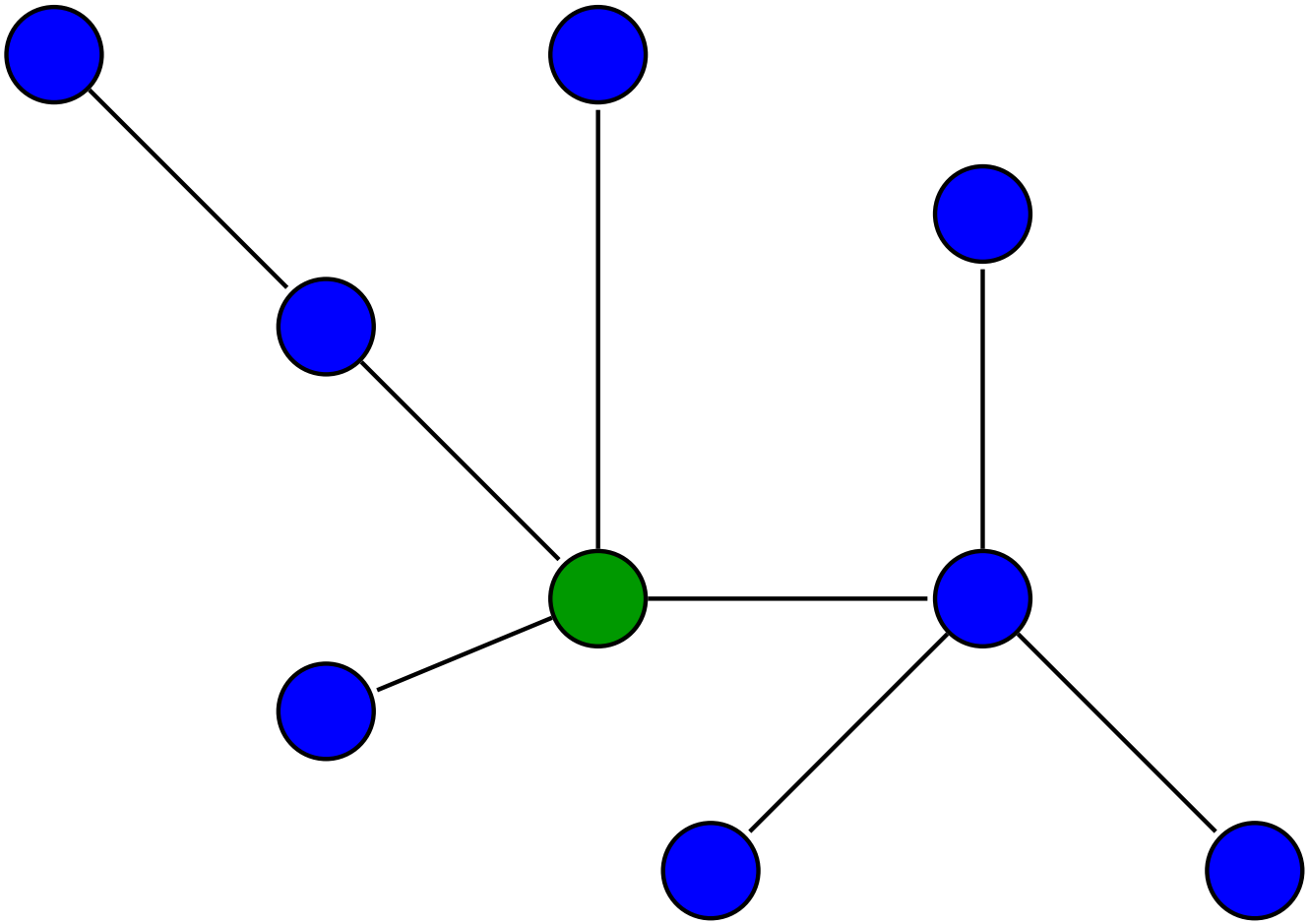
Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

- Using the PVS proof assistant
- Modelling in I/O automata
- Proofs difficult to read.

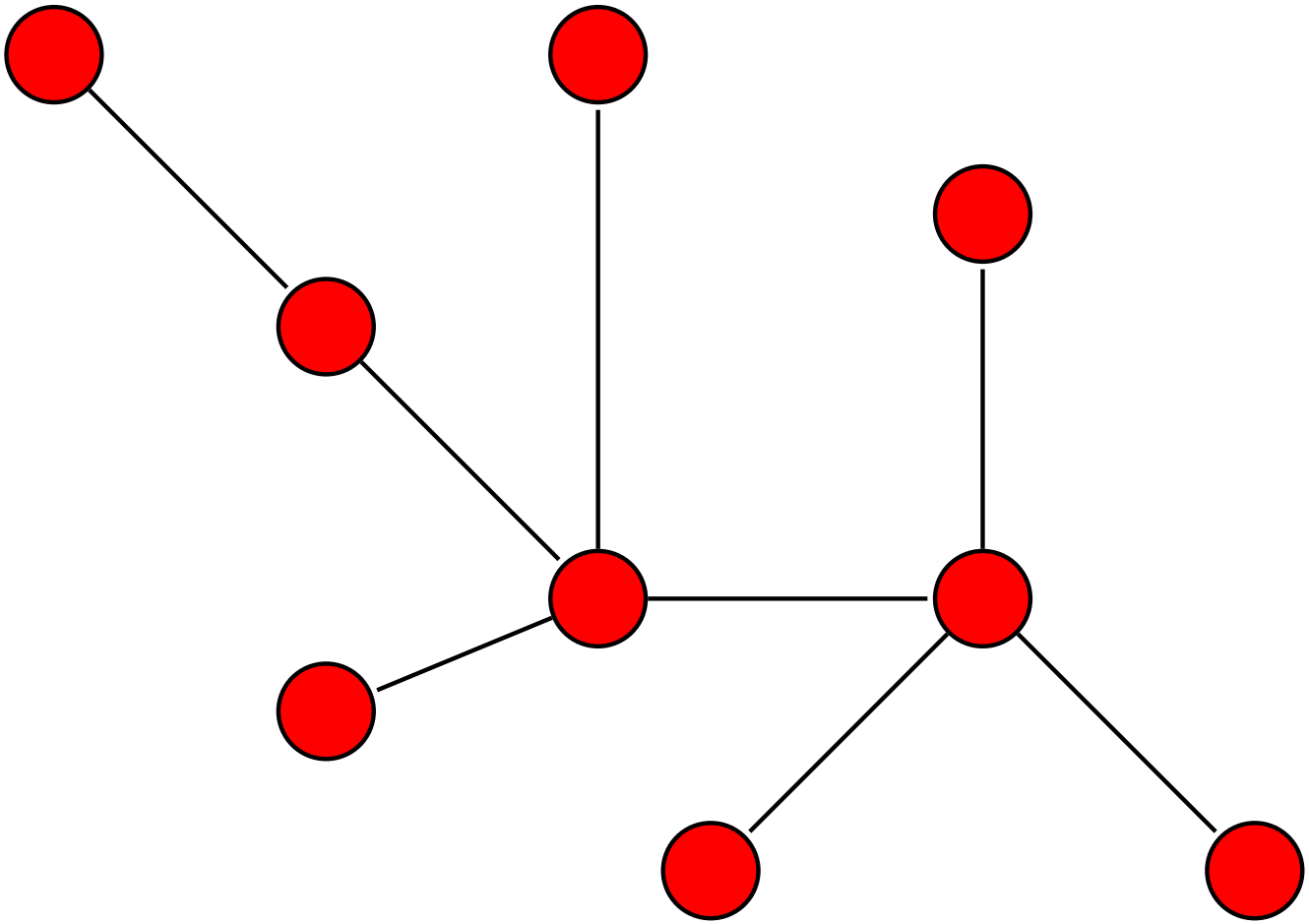
Scenario for the leader election protocol



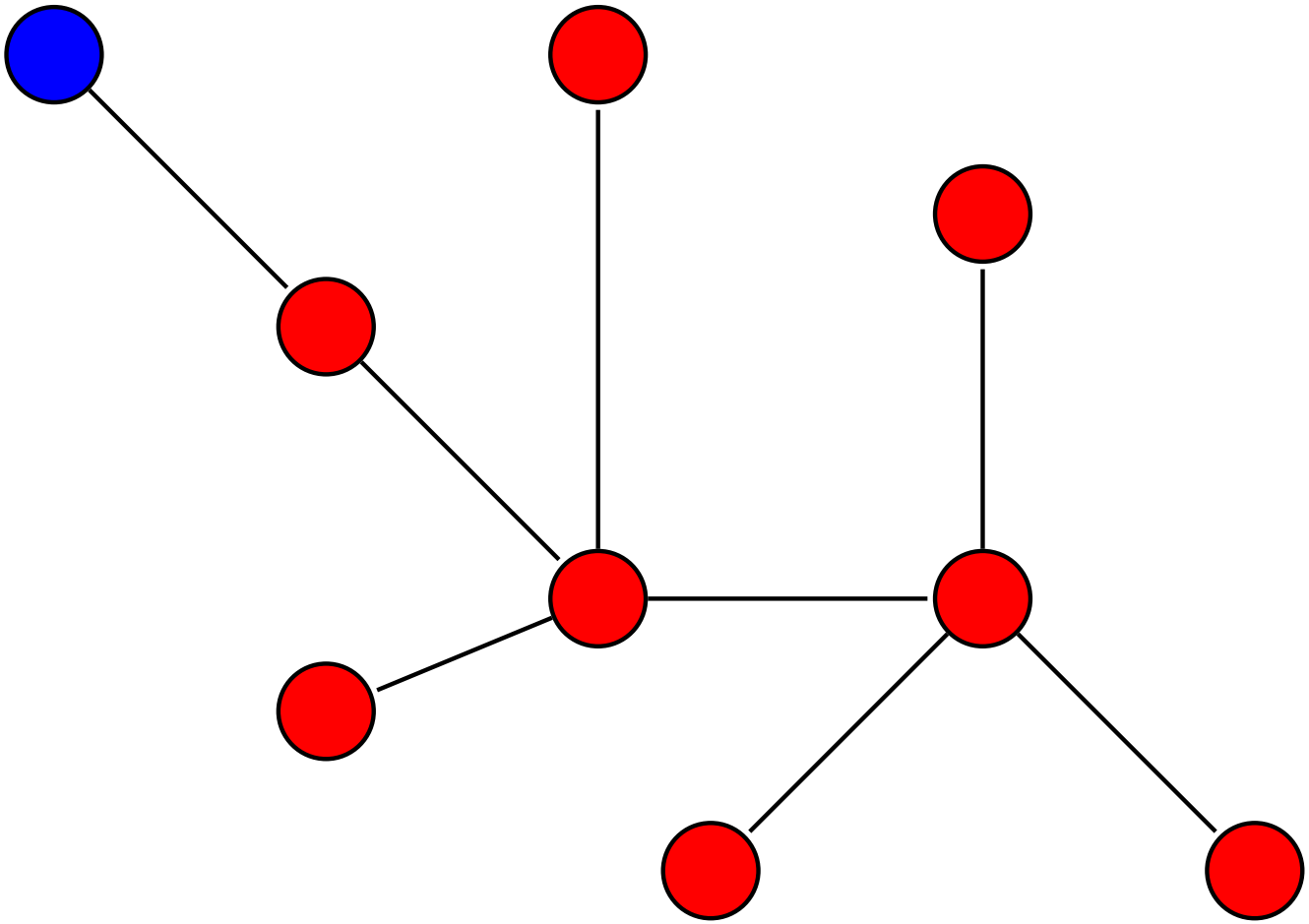
The leader election



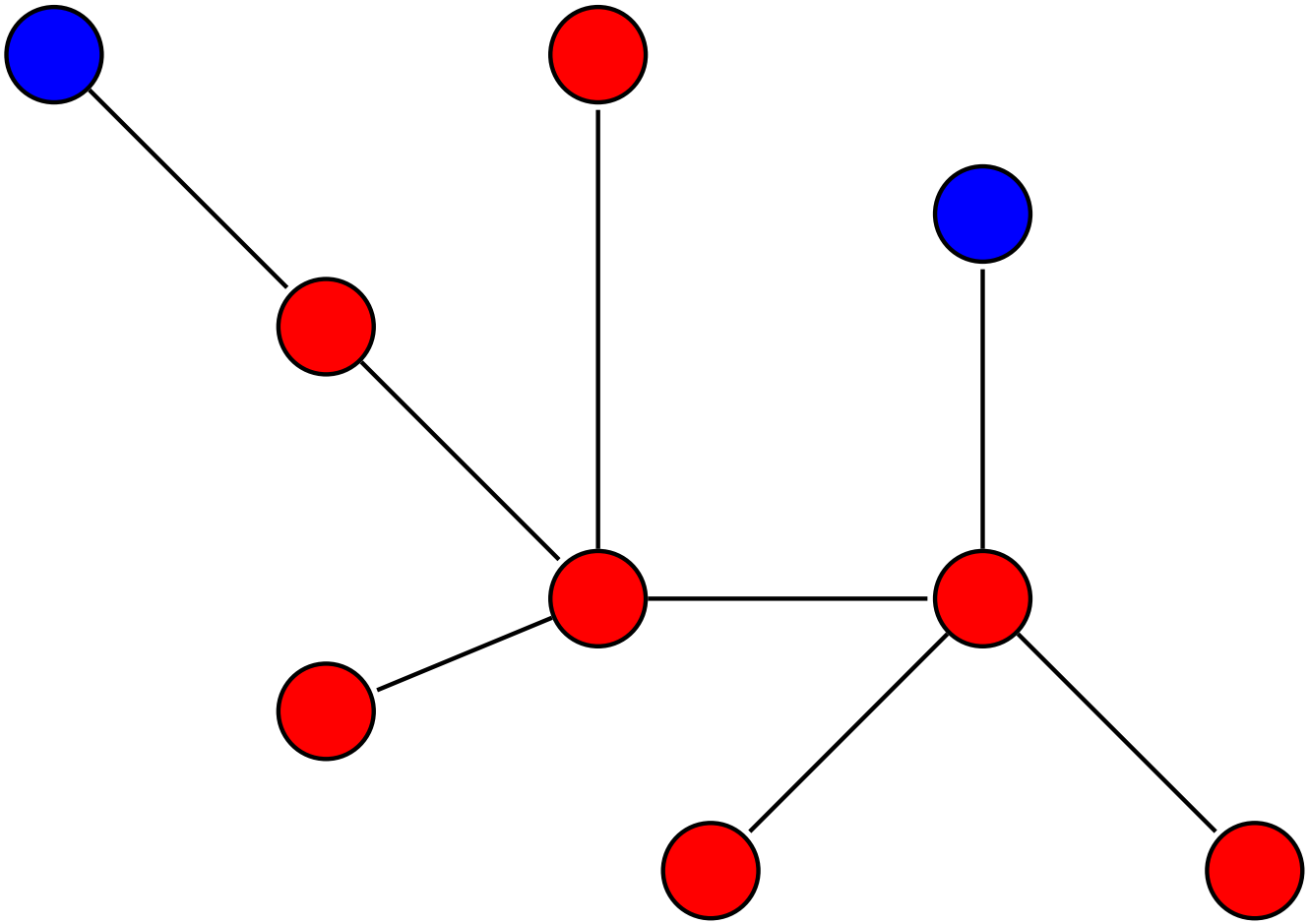
Scenario for the leader election protocol



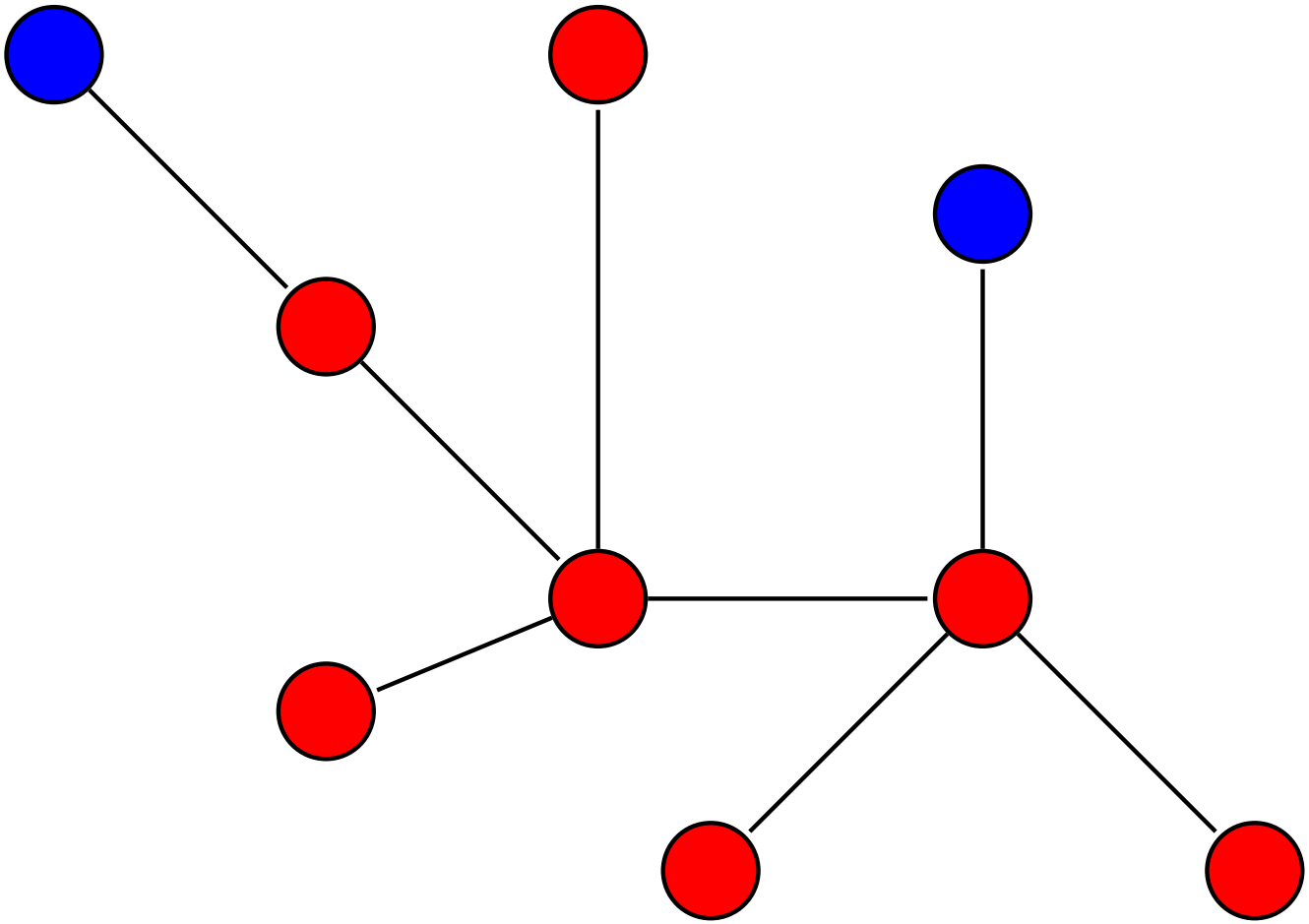
Scenario for the leader election protocol



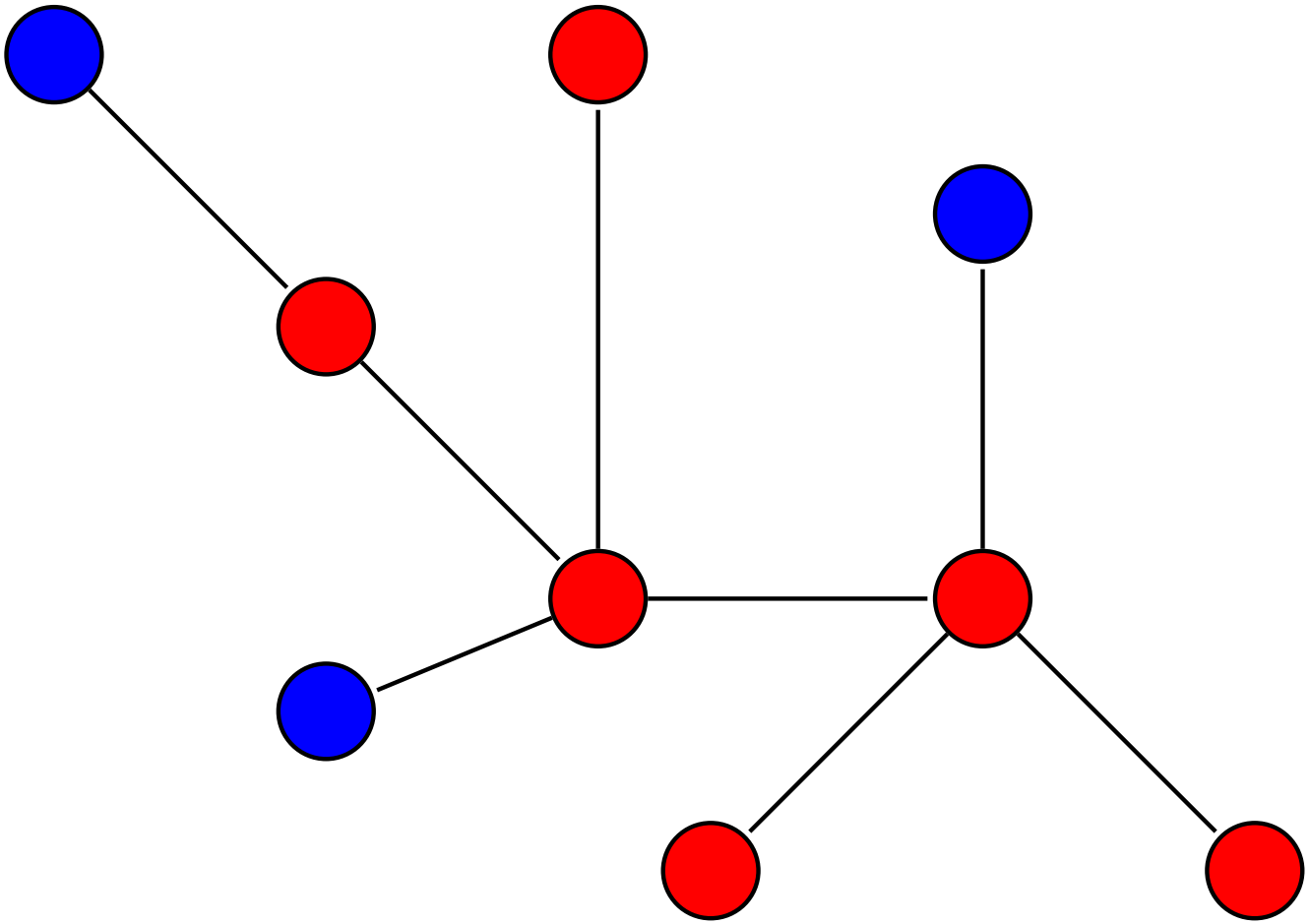
Scenario for the leader election protocol



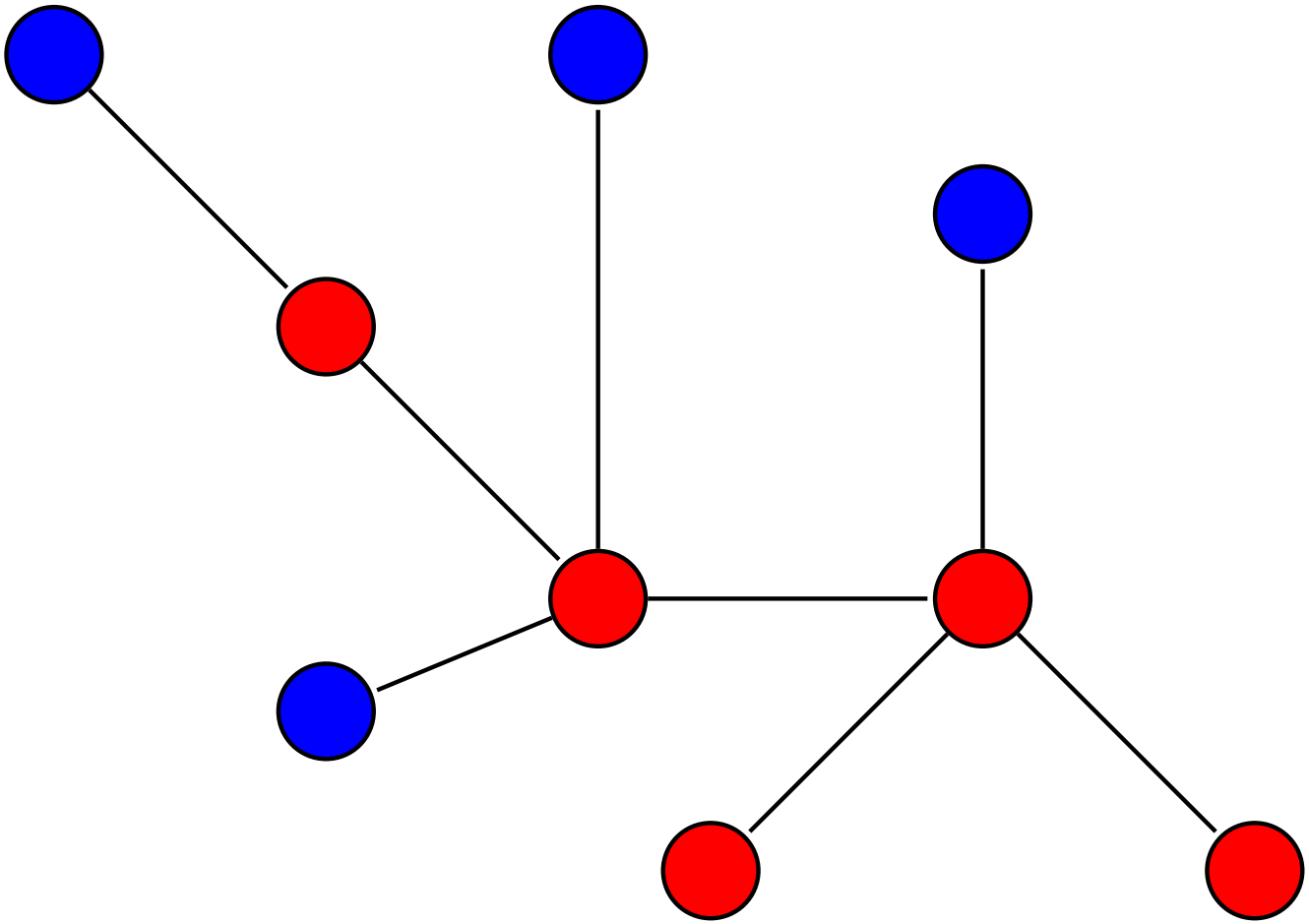
Scenario for the leader election protocol



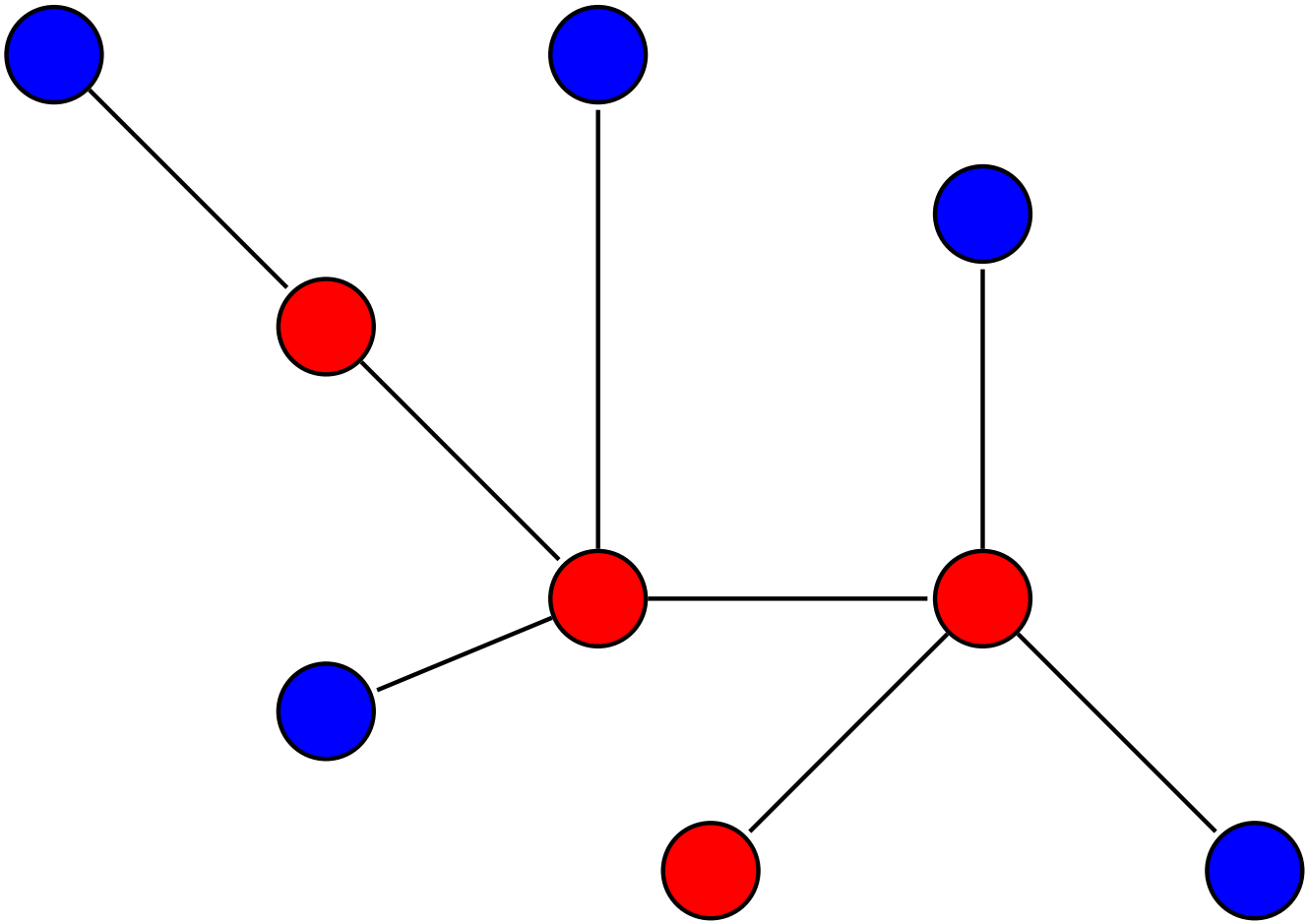
Scenario for the leader election protocol



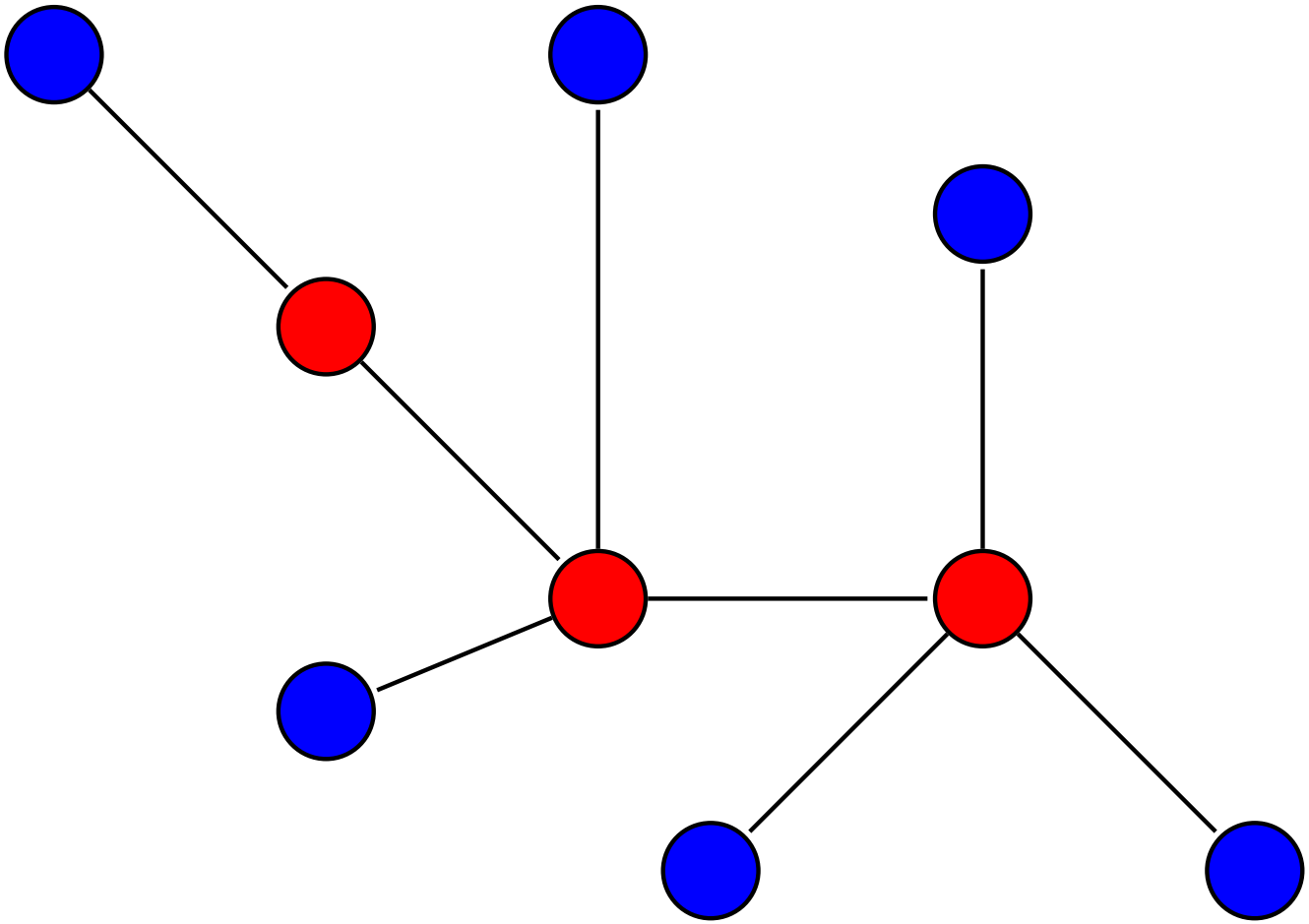
Scenario for the leader election protocol



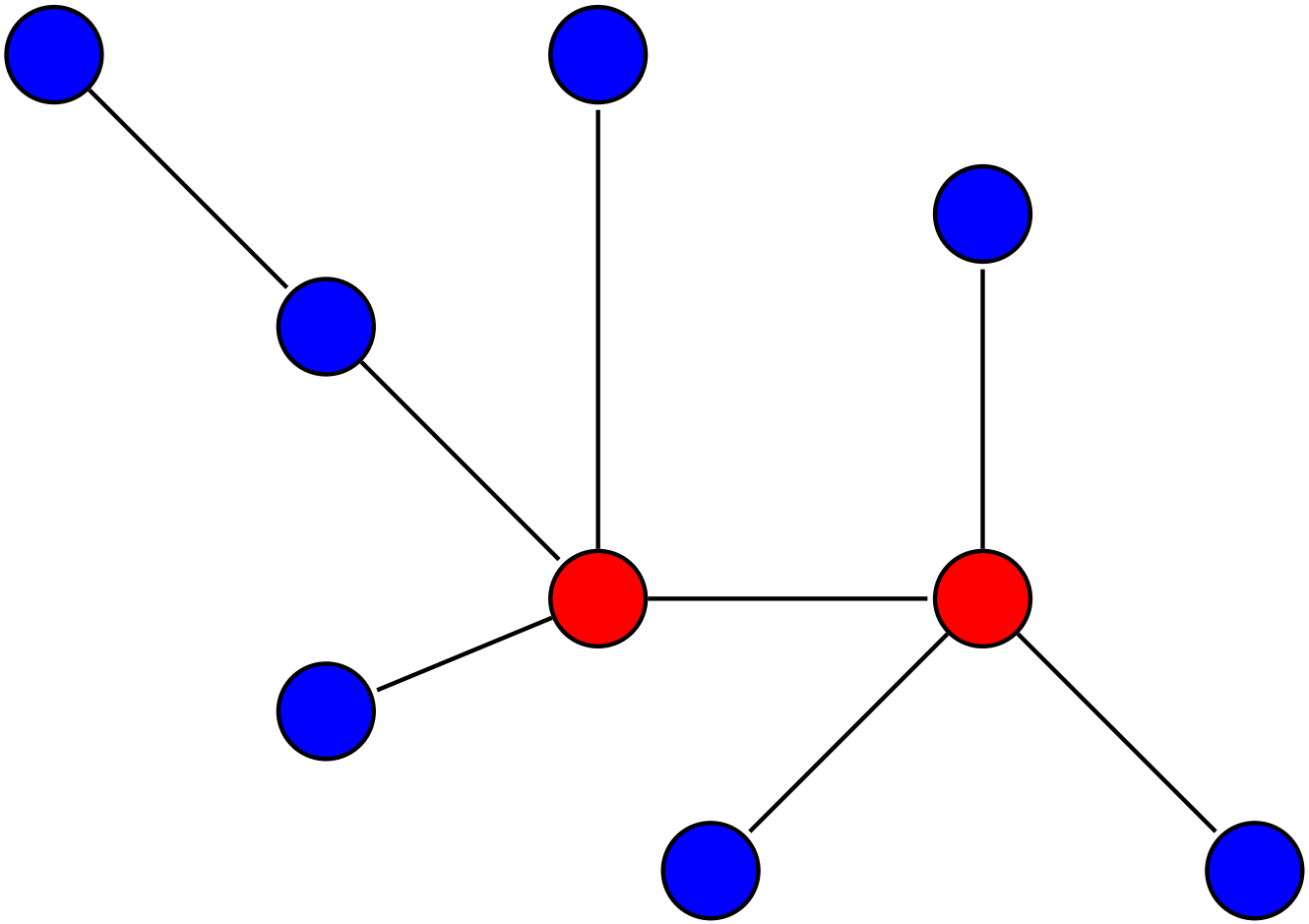
Scenario for the leader election protocol



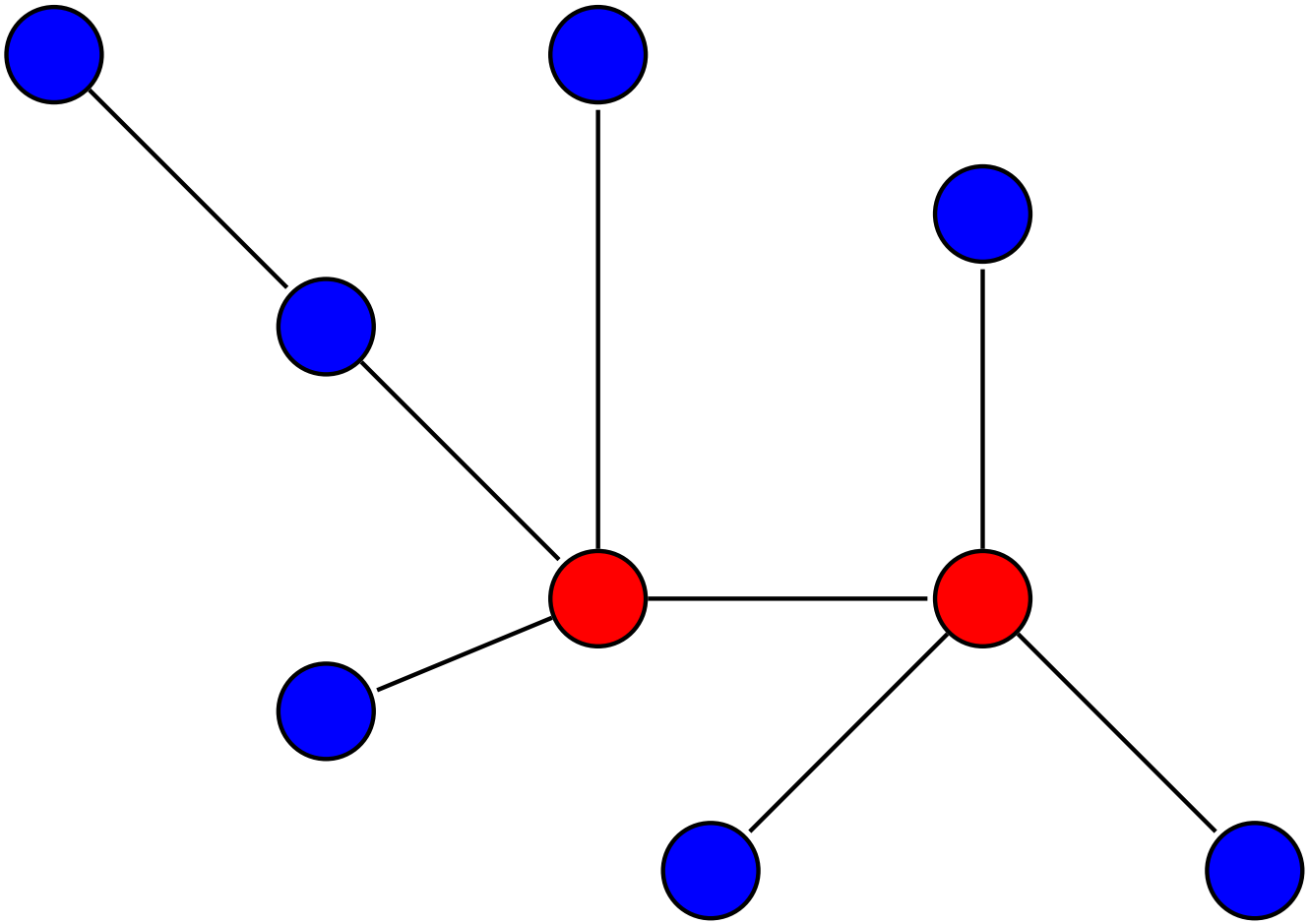
Scenario for the leader election protocol



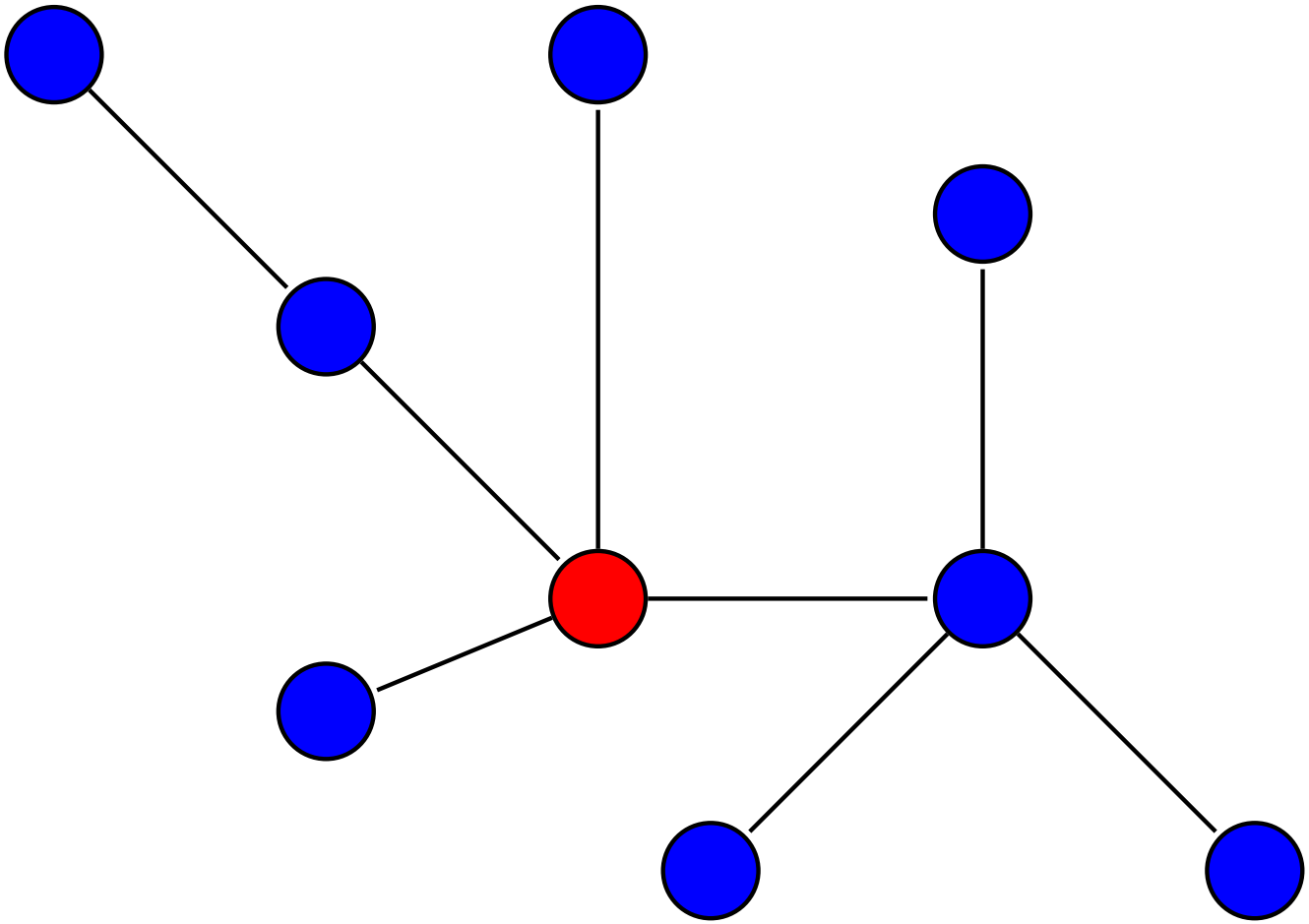
Scenario for the leader election protocol



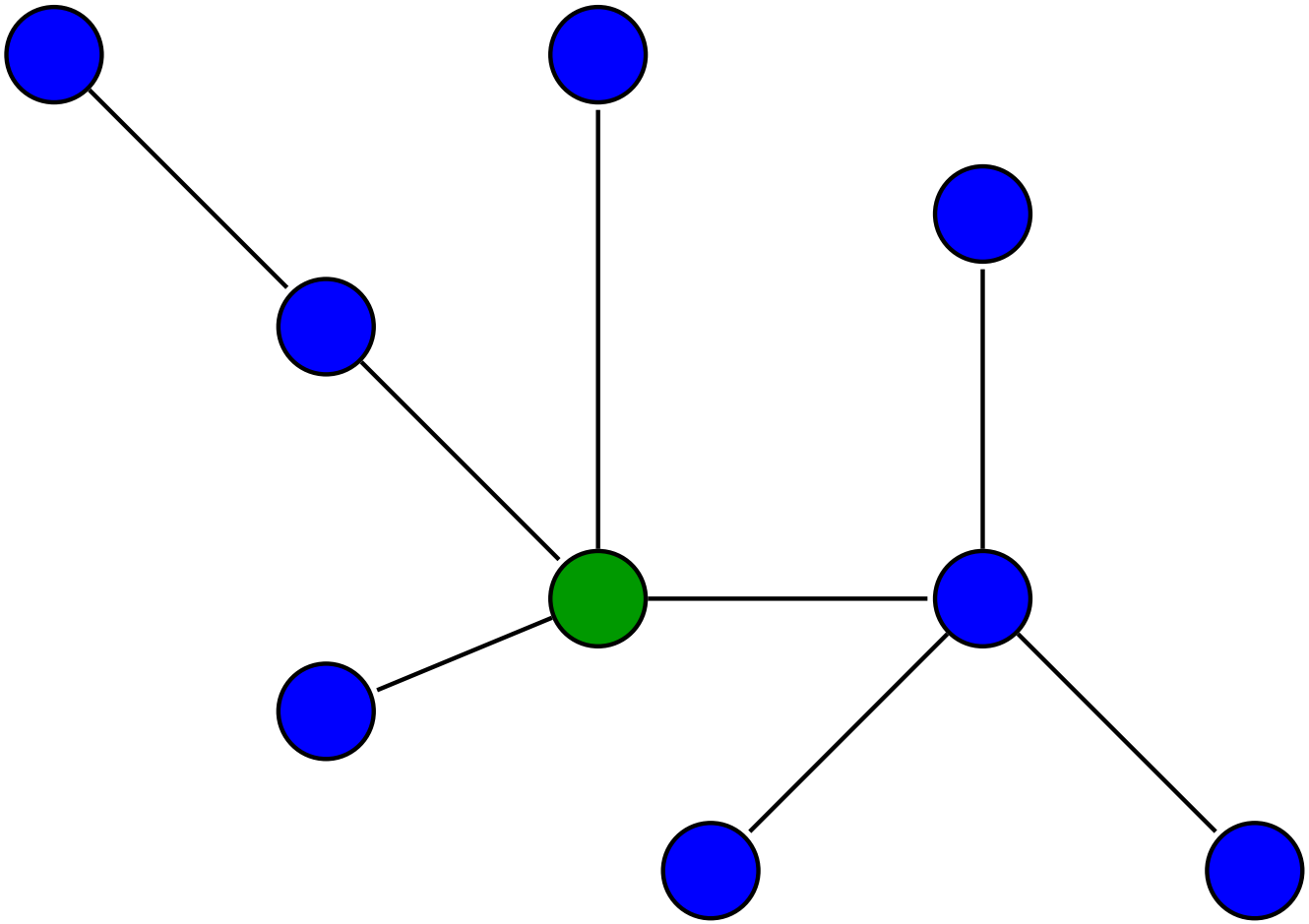
Scenario for the leader election protocol



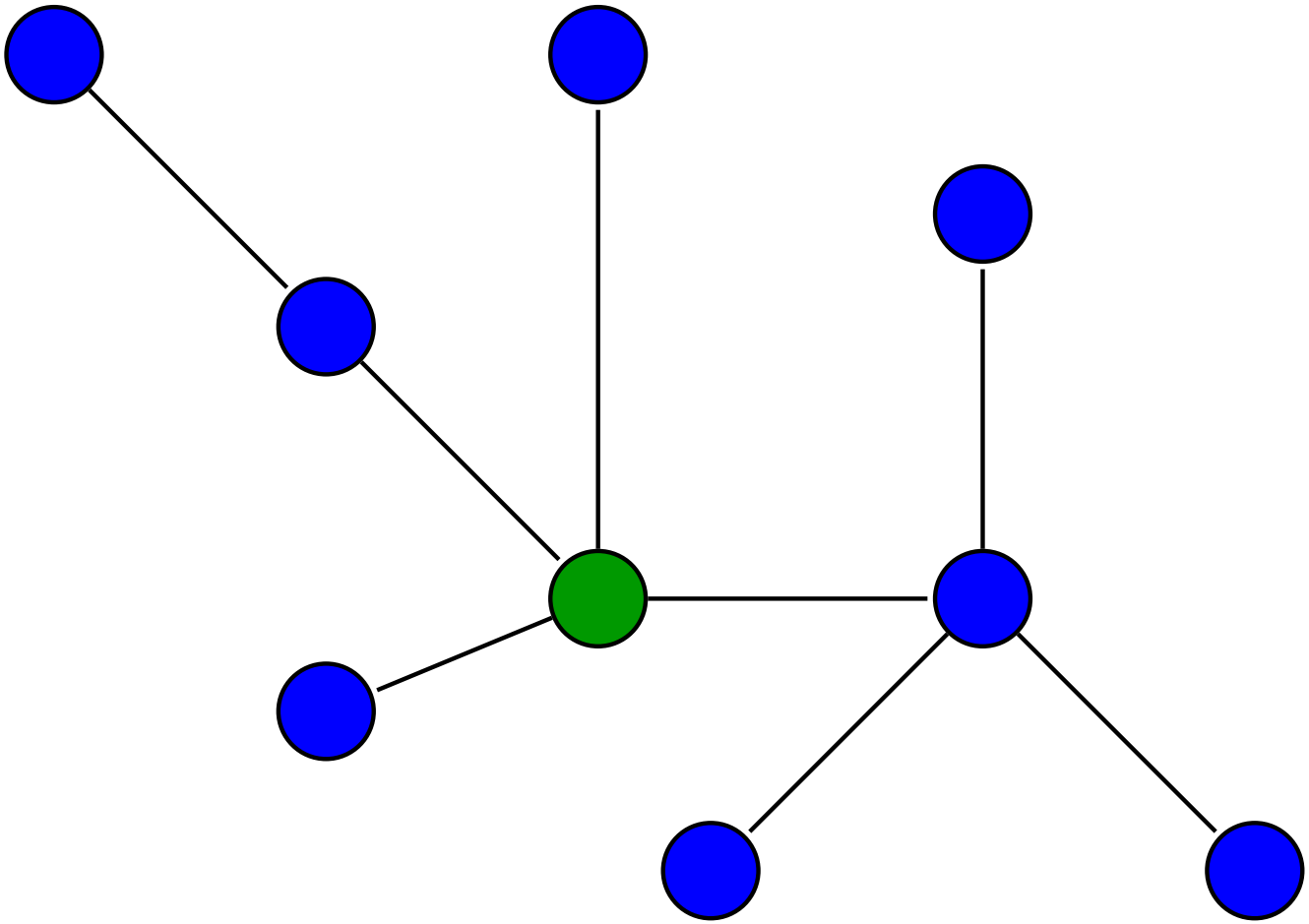
Scenario for the leader election protocol



Scenario for the leader election protocol



The leader election

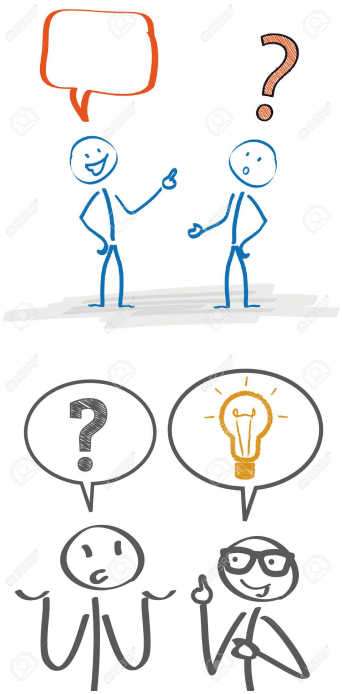


Distributed Algorithms: quest for verification



Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

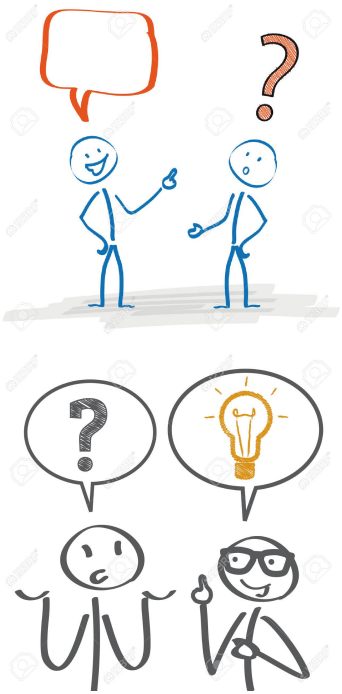
Distributed Algorithms: quest for verification



Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

Distributed Algorithms: quest for verification

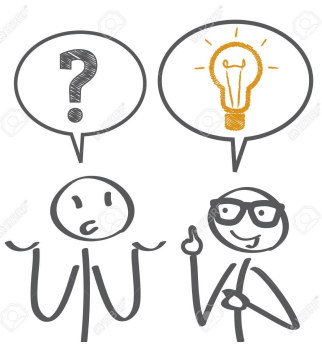


Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

Distributed Algorithms: quest for verification





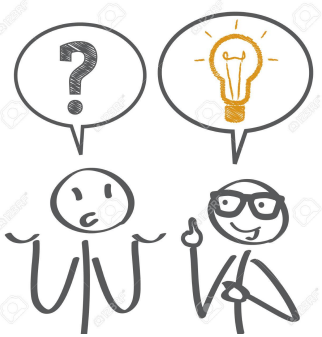
Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

- Main inductive property: *a forest is converging to a tree.*
- ...

Distributed Algorithms: quest for verification





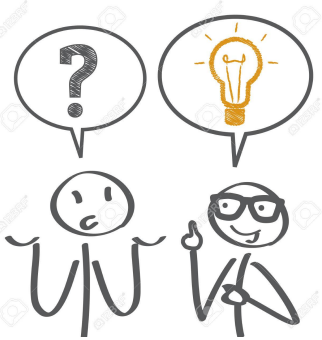
Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

- Main inductive property: *a forest is converging to a tree.*
- . . . but it should exist eventually a tree.

Distributed Algorithms: quest for verification





Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

- Main inductive property: *a forest is converging to a tree.*
- . . . but it should exist eventually a tree.
- Knowledges over graphs should be somewhere modelled.

Guidelines for a Modelling Language

How to Solve It by Pólya

If you can't solve a problem, then there is an easier problem you can solve: find it. or *If you cannot solve the proposed problem, try to solve first some related problem. Could you imagine a more accessible related problem?*.

- *patterns* are a key concept for solving problems;
- Moreover, another key concept is the refinement of models handling the complex nature of such systems: the refinement is used for constructing models or patterns.
- Revisit a list of patterns which can be used for developing programs or systems using the refinement and the proof as a mean to check the whole process.

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Motivations

Design and Verification of Distributed Algorithms

- Issues
  - Complexity of proof derivation of properties
    - Safety, liveness, fairness
  - Non-functional requirements
    - Probabilistic choices, temporal aspects, ...
    - Network properties: e.g. connectivity, (a)cyclicity
    - Fault-tolerance, security, reliability, hardware properties

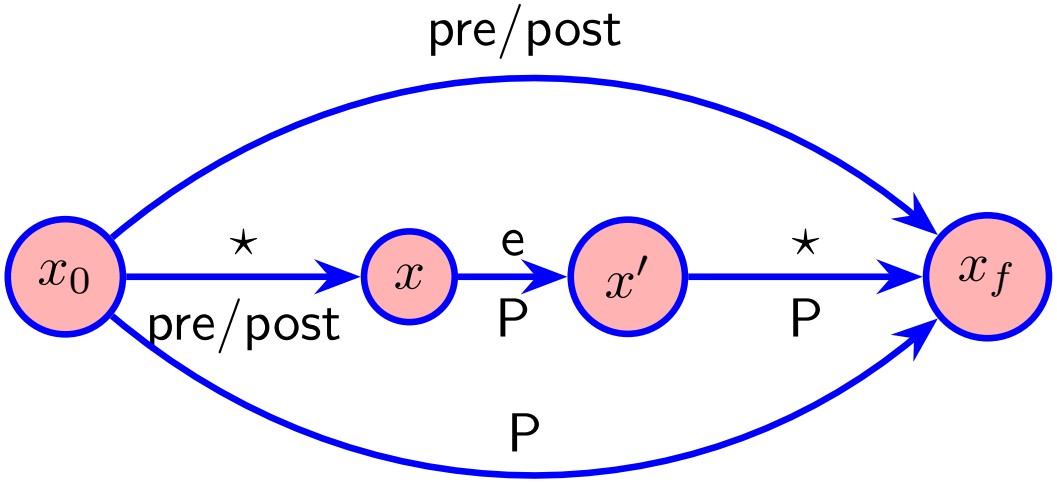
Motivations

Design and Verification of Distributed Algorithms

- Issues
  - Complexity of proof derivation of properties
    - Safety, liveness, fairness
  - Non-functional requirements
    - Probabilistic choices, temporal aspects, ...
    - Network properties: e.g. connectivity, (a)cyclicity
    - Fault-tolerance, security, reliability, hardware properties
- Proposed Methodologies
  - *"Correct-by-Construction"* Paradigm
    - Incremental (Re-) Construction of targeted distributed algorithms
    - Core : Refinement of models
    - Reusability: models, properties, proofs, correctness, ...
  - *"Service-as-Event"* Paradigm
    - Organisation/Structuration of incremental development of distributed algorithms
    - Semantical Framework for reasoning with liveness properties

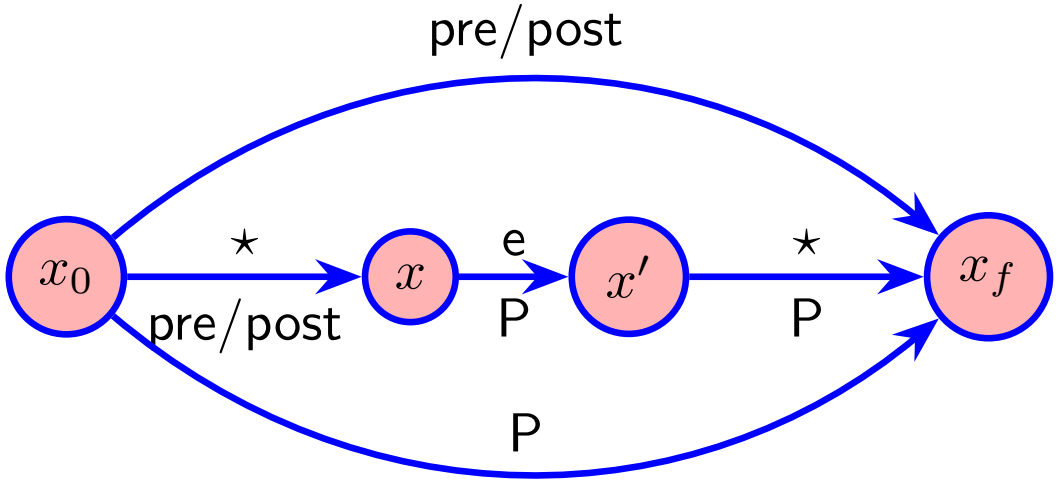
Big steps versus Small steps

Refining a pre/post specification into a sequential algorithm

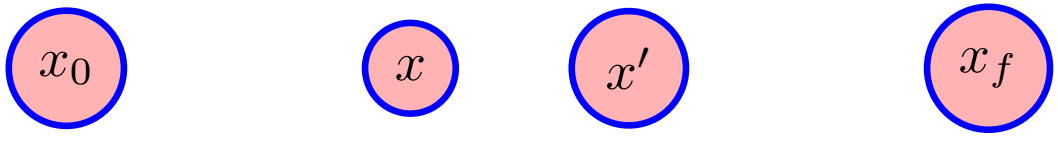


Big steps versus Small steps

Refining a pre/post specification into a sequential algorithm

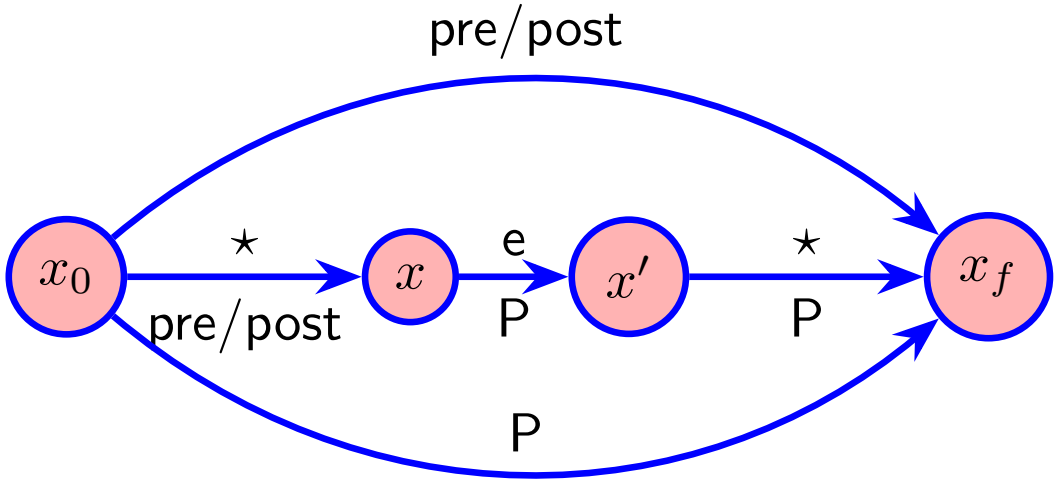


Refining a service specification into a distributed algorithm

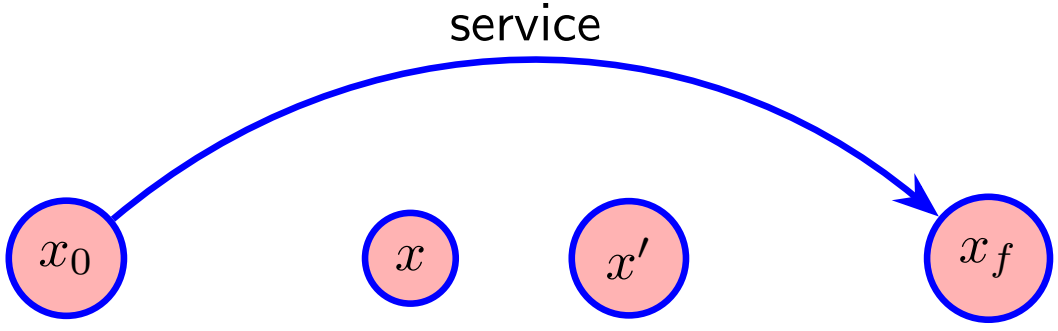


Big steps versus Small steps

Refining a pre/post specification into a sequential algorithm

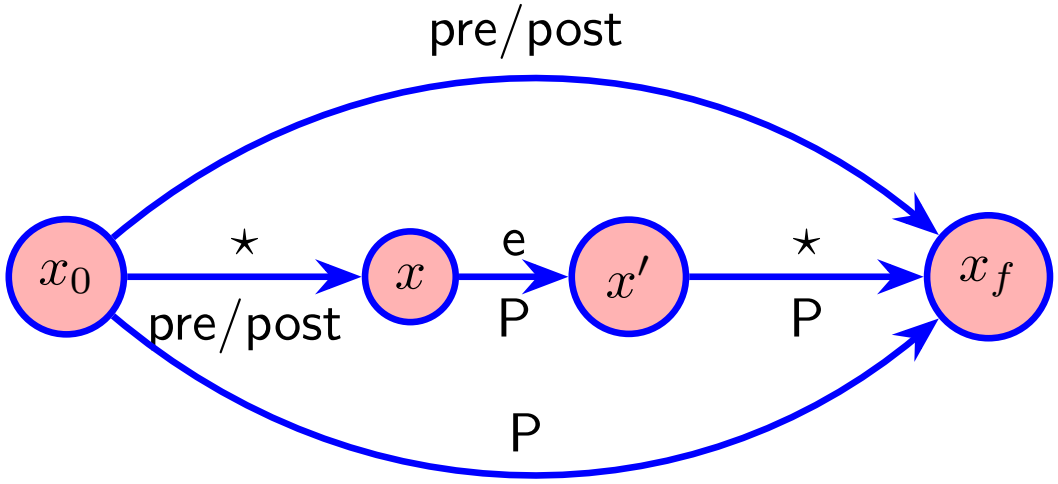


Refining a service specification into a distributed algorithm

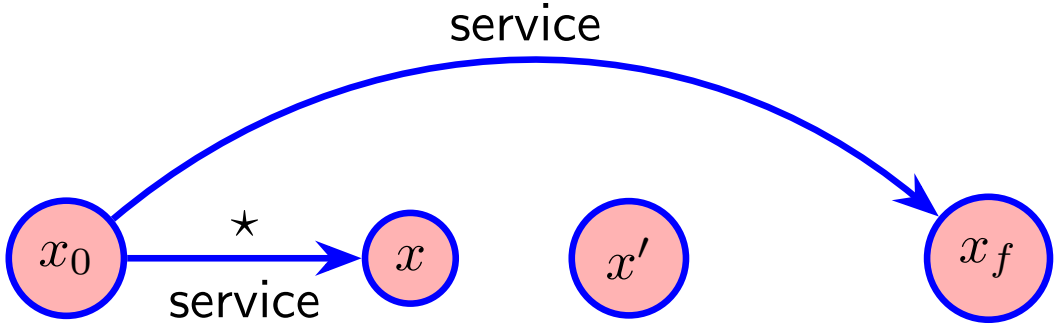


Big steps versus Small steps

Refining a pre/post specification into a sequential algorithm

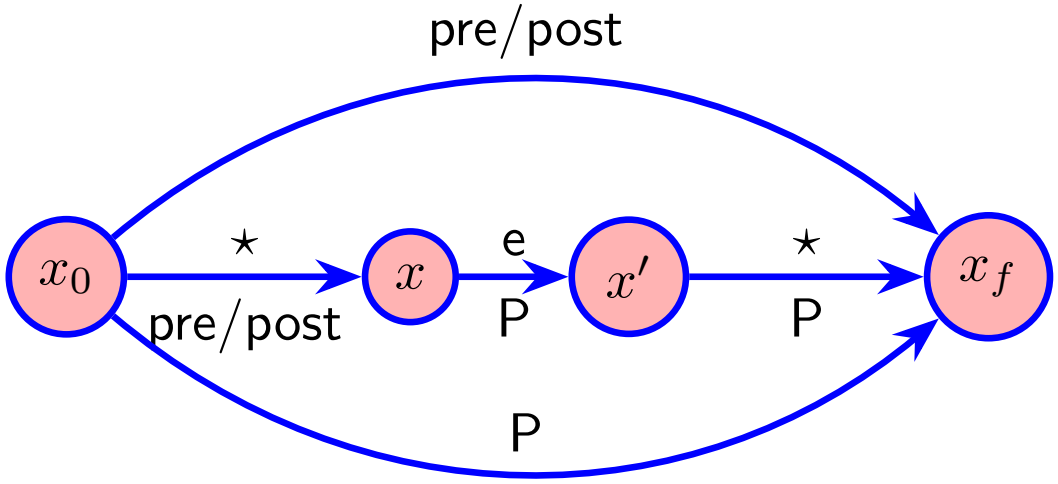


Refining a service specification into a distributed algorithm

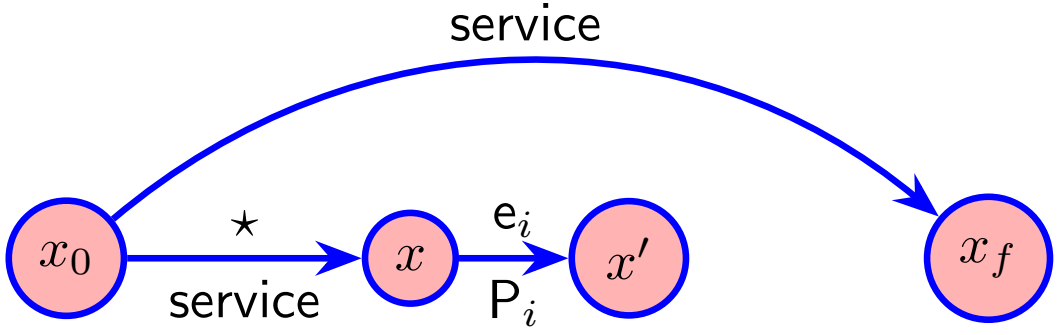


Big steps versus Small steps

Refining a pre/post specification into a sequential algorithm

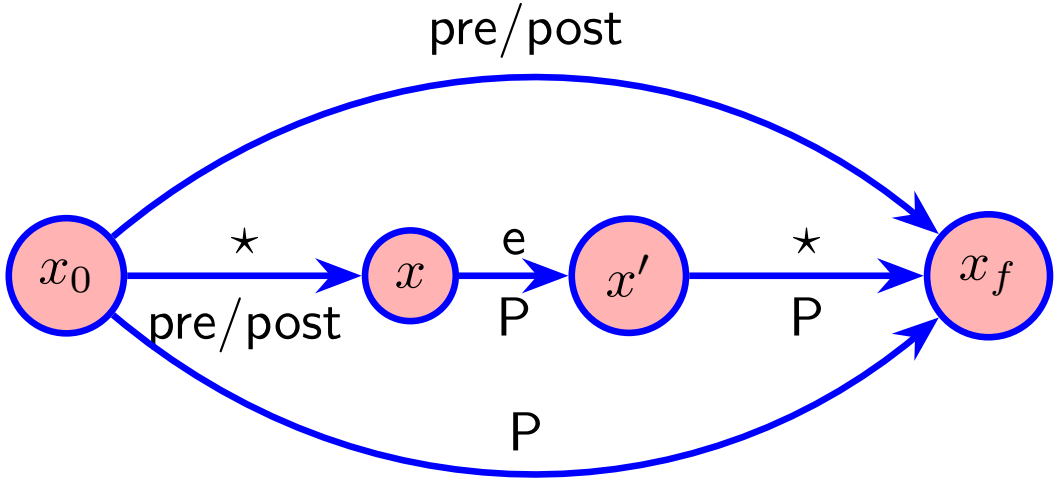


Refining a service specification into a distributed algorithm

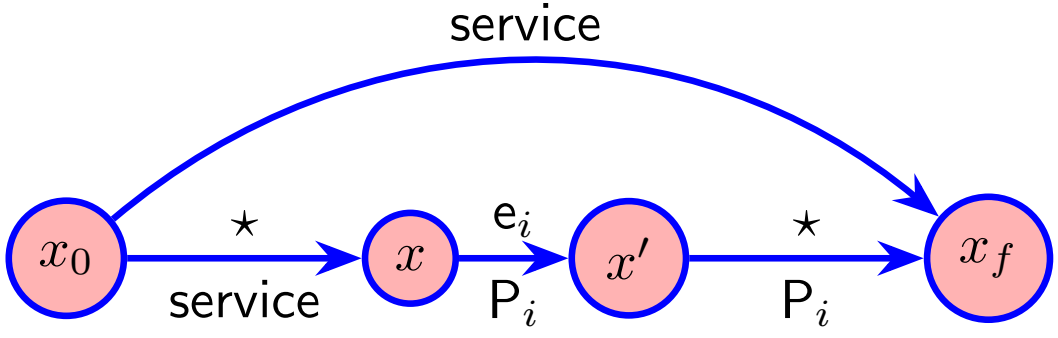


Big steps versus Small steps

Refining a pre/post specification into a sequential algorithm

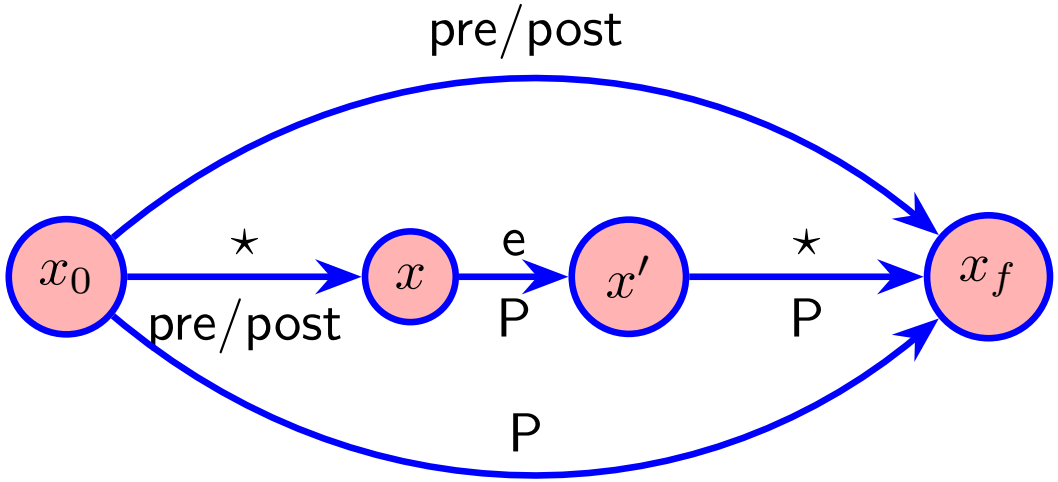


Refining a service specification into a distributed algorithm

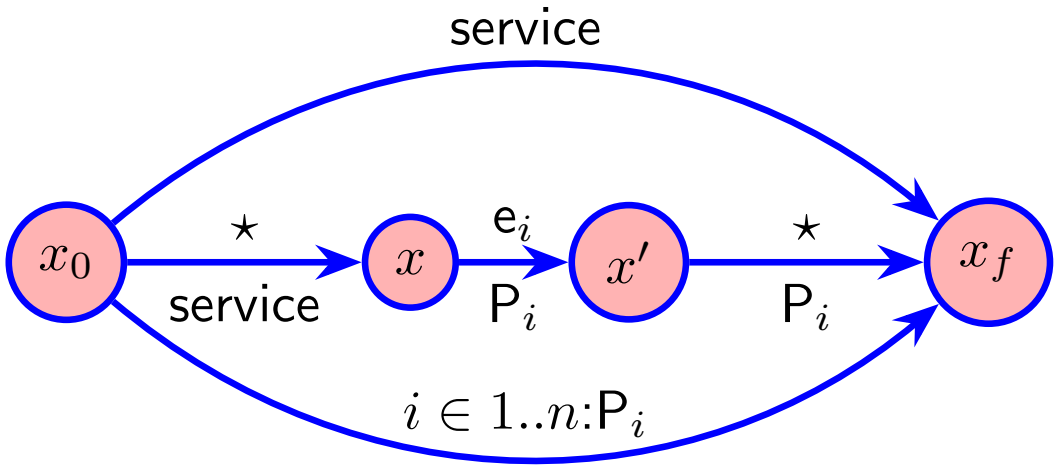


Big steps versus Small steps

Refining a pre/post specification into a sequential algorithm



Refining a service specification into a distributed algorithm



First steps using refinement for distributed algorithms

- The IEEE 1394 Tree Identify Protocol:
  - If the graph is connected and acyclic, then the reset eventually leads to an election.
  - If the graph is not connected or acyclic, then the reset eventually leads to the non election.

First steps using refinement for distributed algorithms

- The IEEE 1394 Tree Identify Protocol:
  - If the graph is connected and acyclic, then the reset eventually leads to an election:
    $connected(gr) \wedge acyclic(gr) \models reset \rightsquigarrow election.$
  - If the graph is not connected or acyclic, then the reset eventually leads to the non election:
    $\neg(connected(gr) \wedge acyclic(gr)) \models reset \rightsquigarrow nonelection.$

Lessons learnt from the IEEE 1394 leader election protocol

Lessons learnt from the IEEE 1394 leader election protocol

- Importance of properties of the underlying graph: connected, undirected, acyclic.

Lessons learnt from the IEEE 1394 leader election protocol

- Importance of properties of the underlying graph: connected, undirected, acyclic.
- The *Big-step view* of the *magic* event namely election.

Lessons learnt from the IEEE 1394 leader election protocol

- Importance of properties of the underlying graph: connected, undirected, acyclic.
- The *Big-step view* of the *magic* event namely election.
- Finding an induction over the graph: *a forest converging to a tree*

Lessons learnt from the IEEE 1394 leader election protocol

- Importance of properties of the underlying graph: connected, undirected, acyclic.
- The *Big-step view* of the *magic* event namely election.
- Finding an induction over the graph: *a forest converging to a tree*
- Localizing the events!

Lessons learnt from the IEEE 1394 leader election protocol

- Importance of properties of the underlying graph: connected, undirected, acyclic.
- The *Big-step view* of the *magic* event namely election.
- Finding an induction over the graph: *a forest converging to a tree*
- Localizing the events!

Main concepts

Abstraction and Induction (AI) are two key points helping to explain why and how it works.

Lessons learnt from the IEEE 1394 leader election protocol

- Importance of properties of the underlying graph: connected, undirected, acyclic.
- The *Big-step view* of the *magic* event namely election.
- Finding an induction over the graph: *a forest converging to a tree*
- Localizing the events!

Main concepts

Abstraction and Induction (AI) are two key points helping to explain why and how it works.

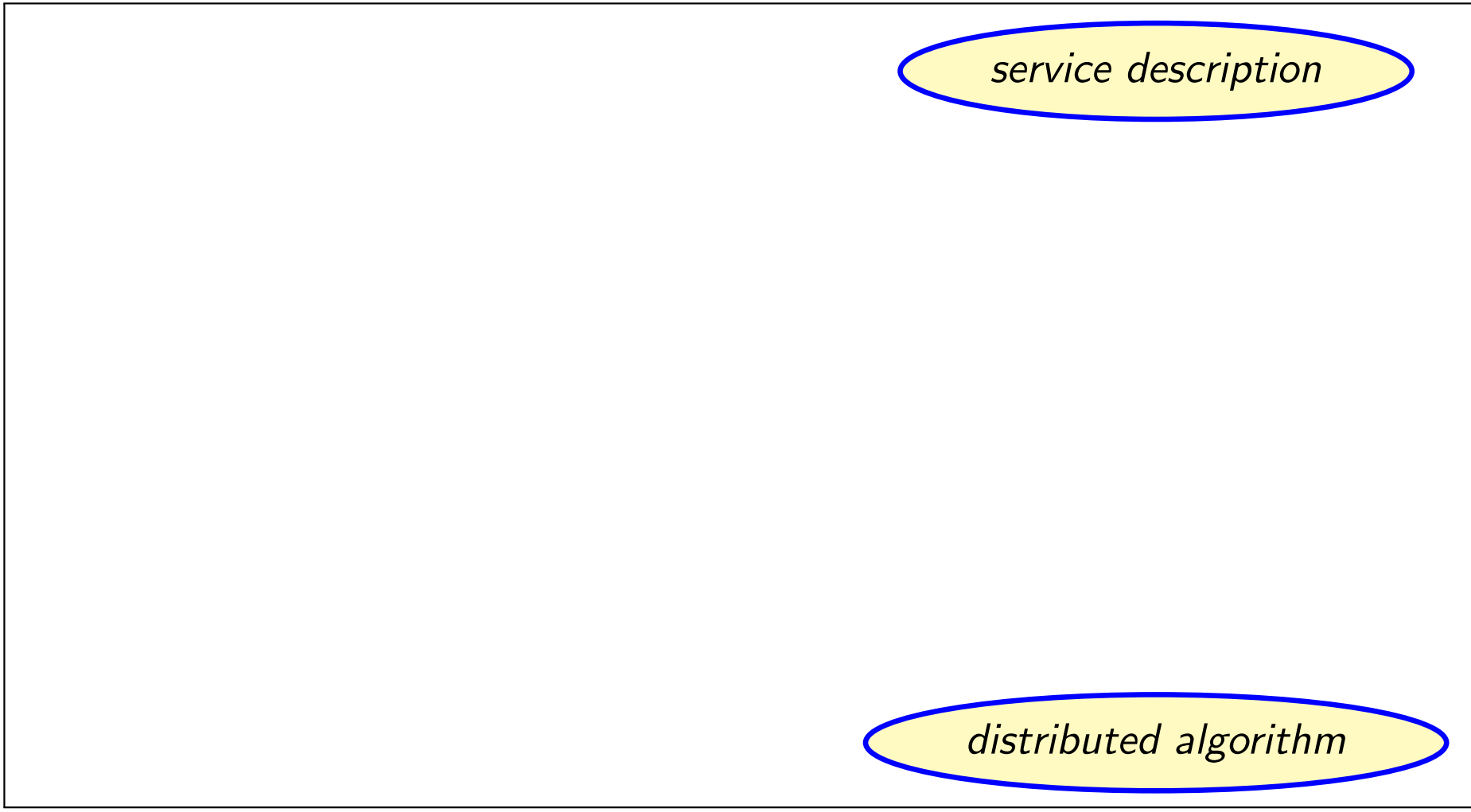
Missing concept

This algorithm posed a problem known as contention, and the contention is resolved using a probabilistic approach.

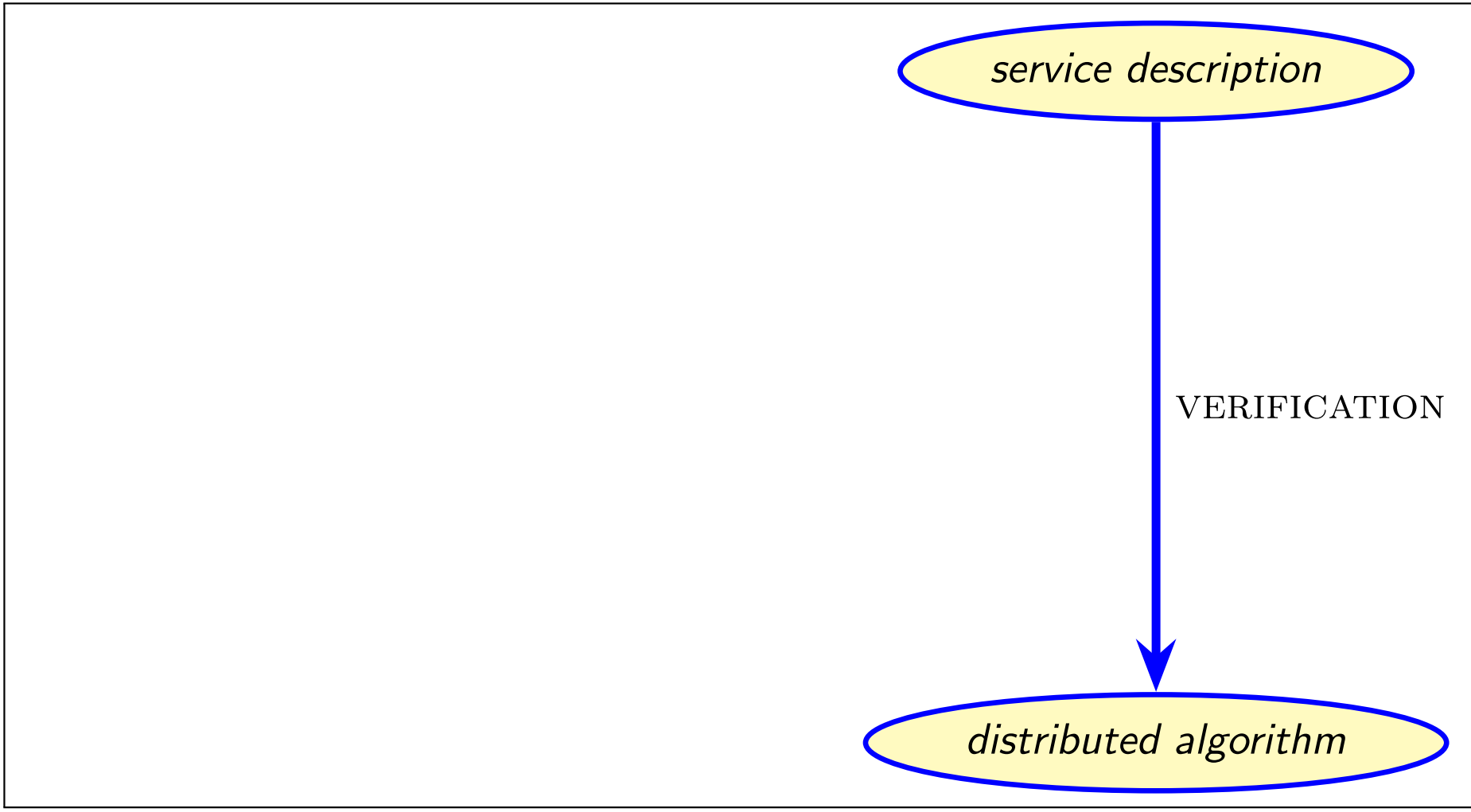
The distributed pattern



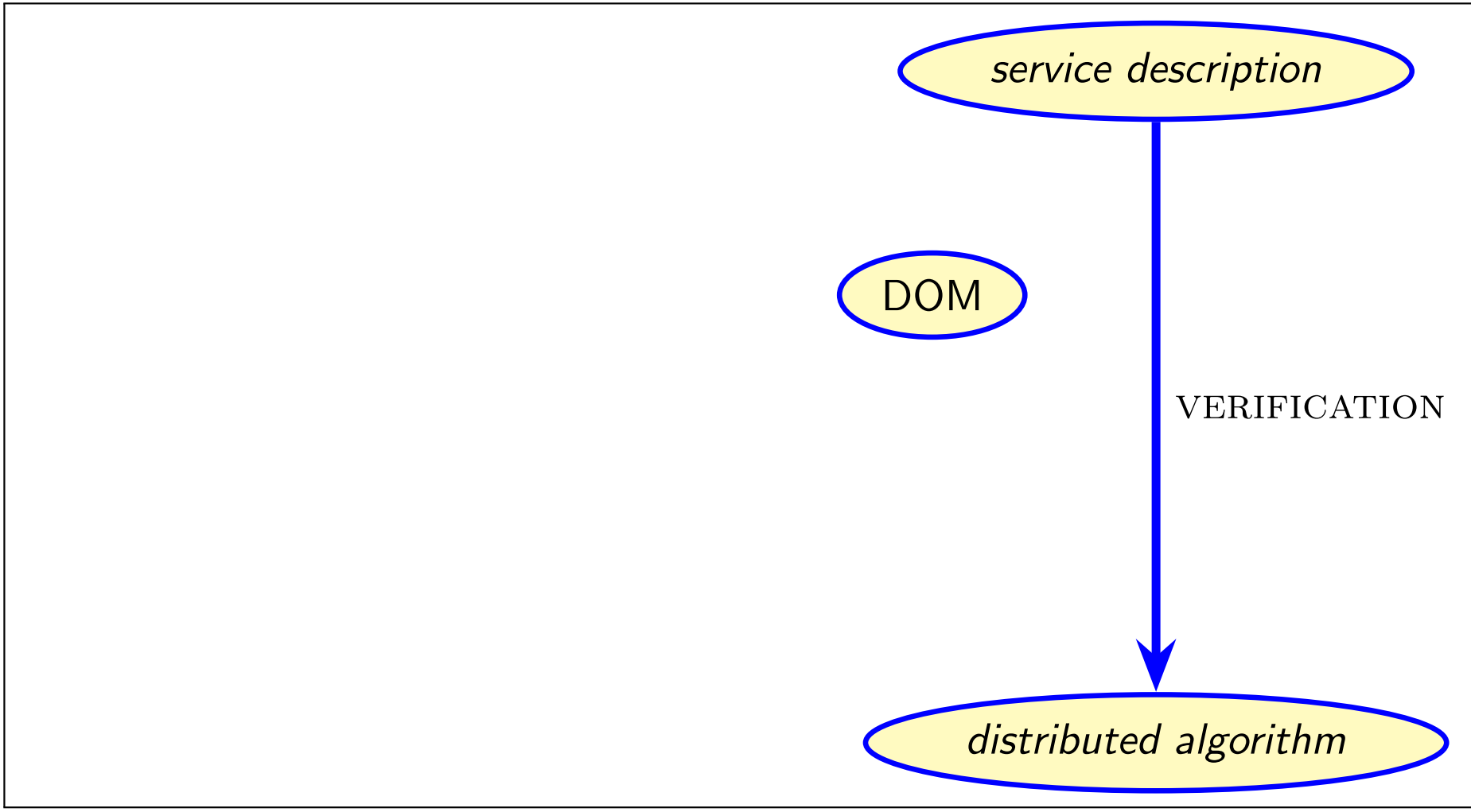
The distributed pattern



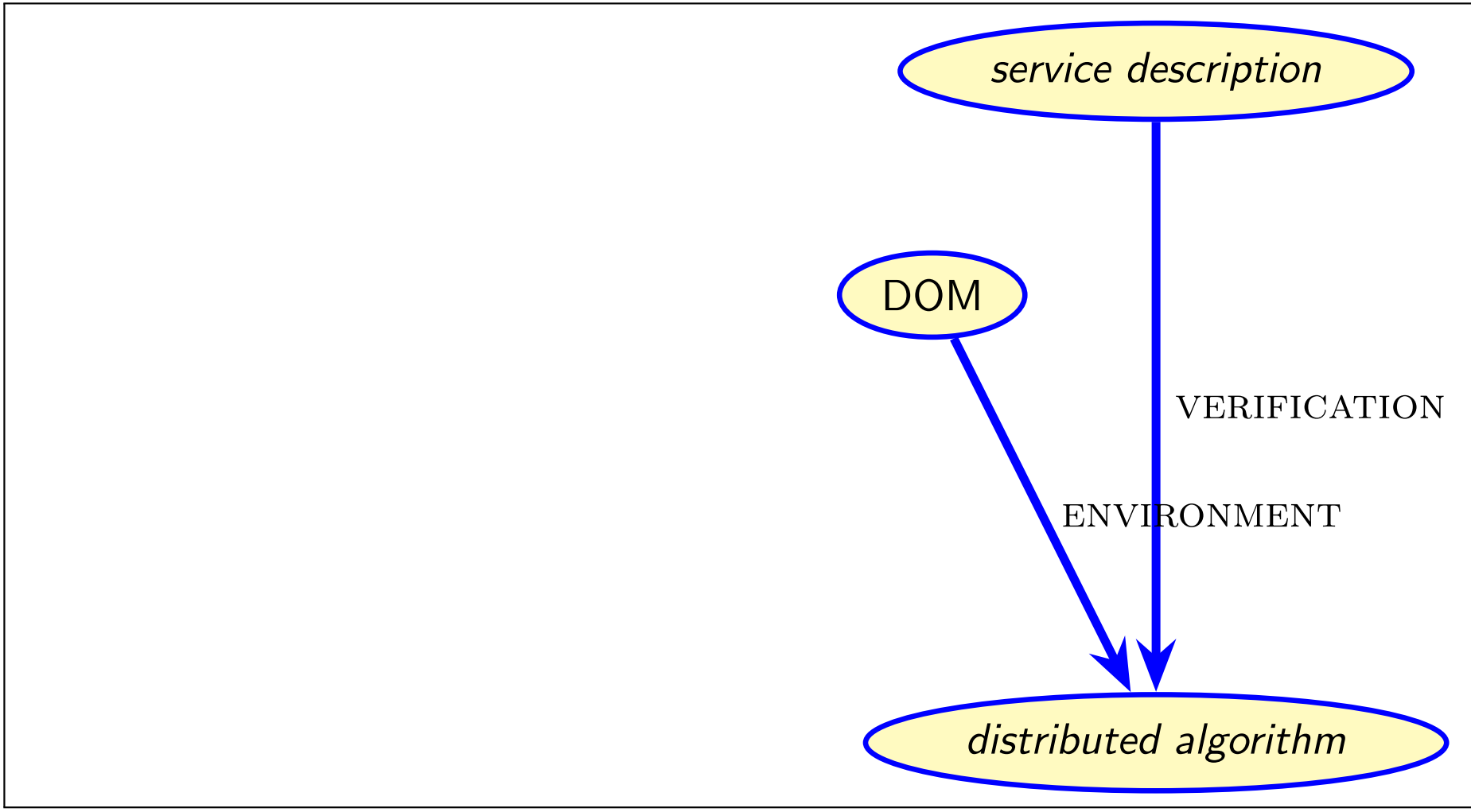
The distributed pattern



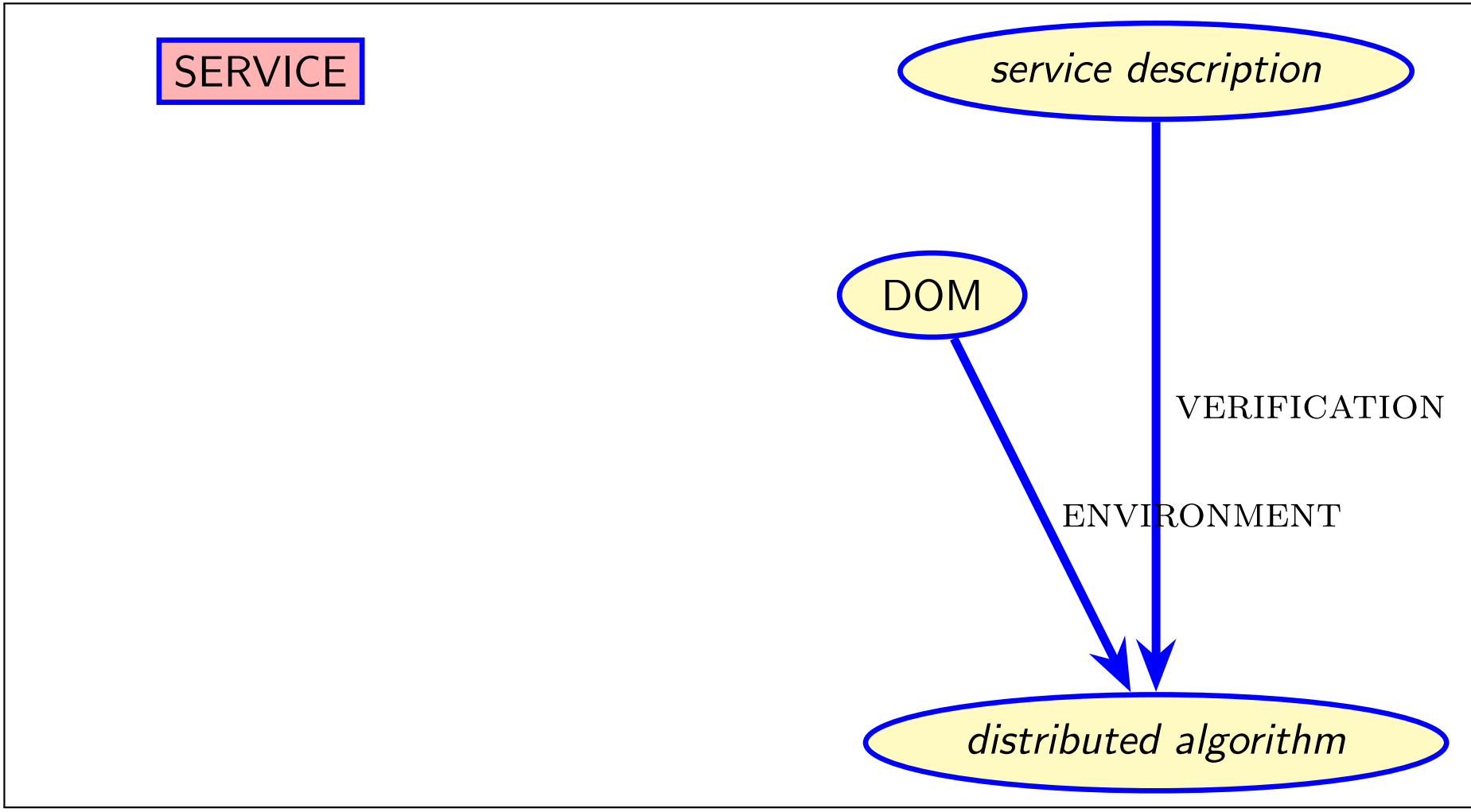
The distributed pattern



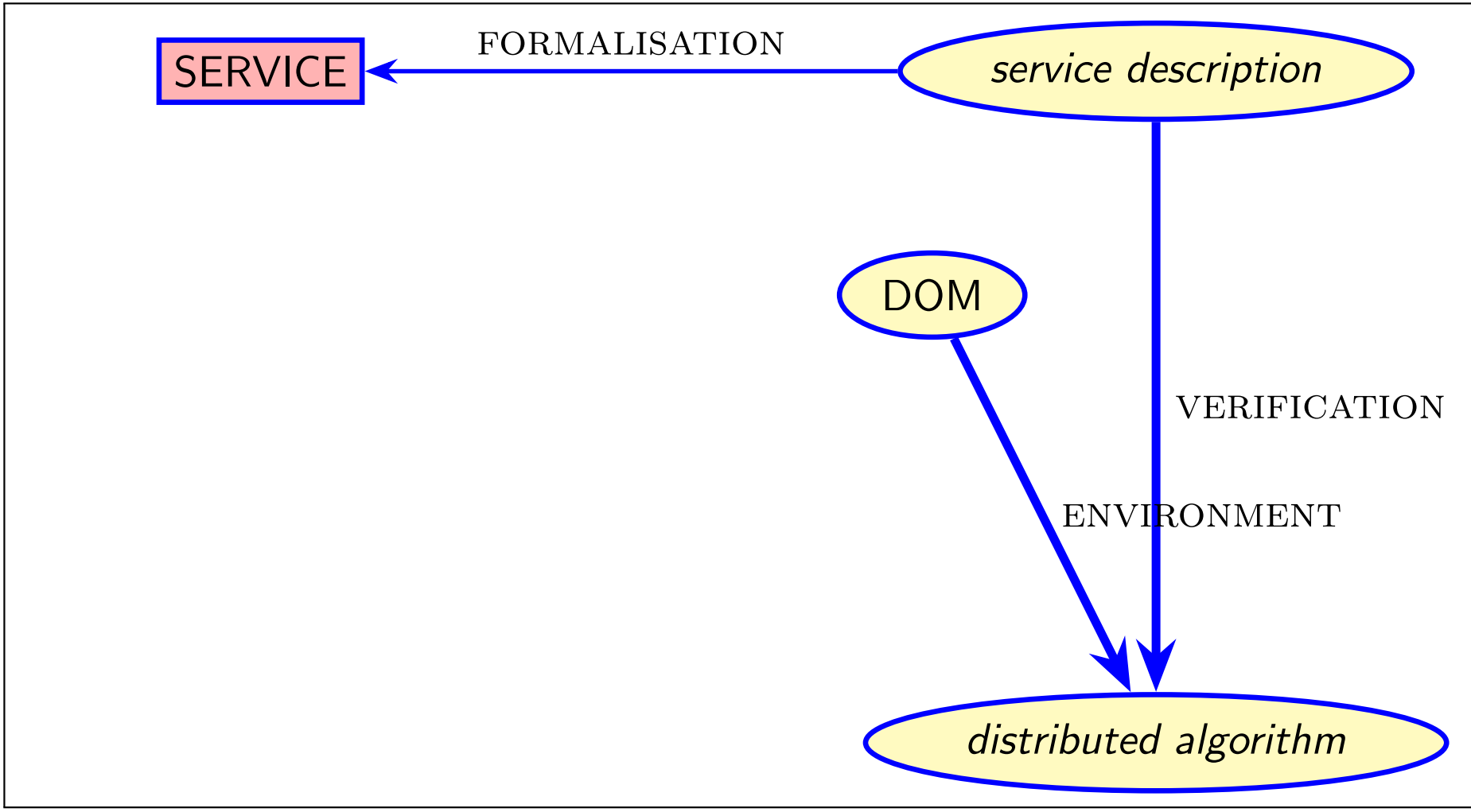
The distributed pattern



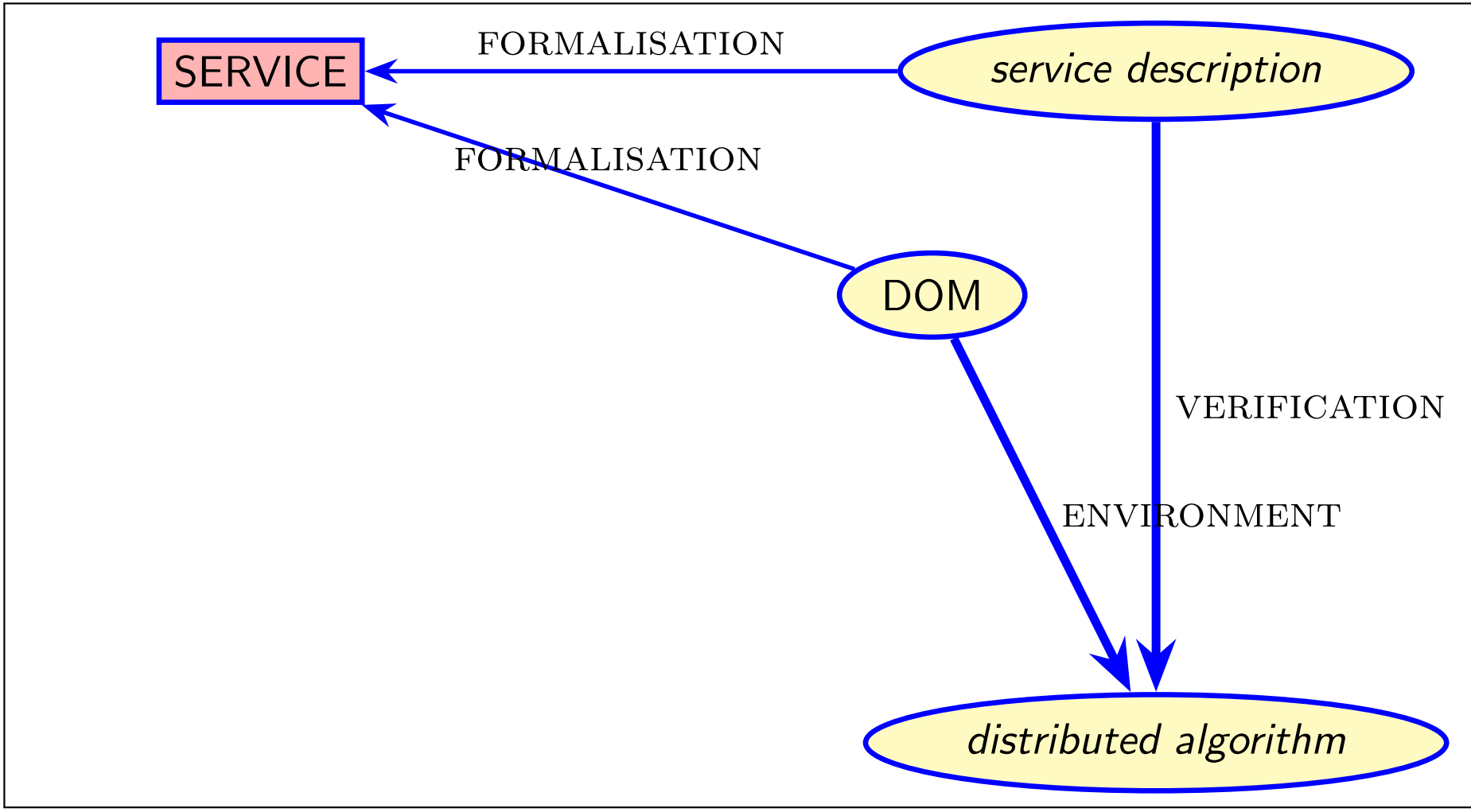
The distributed pattern



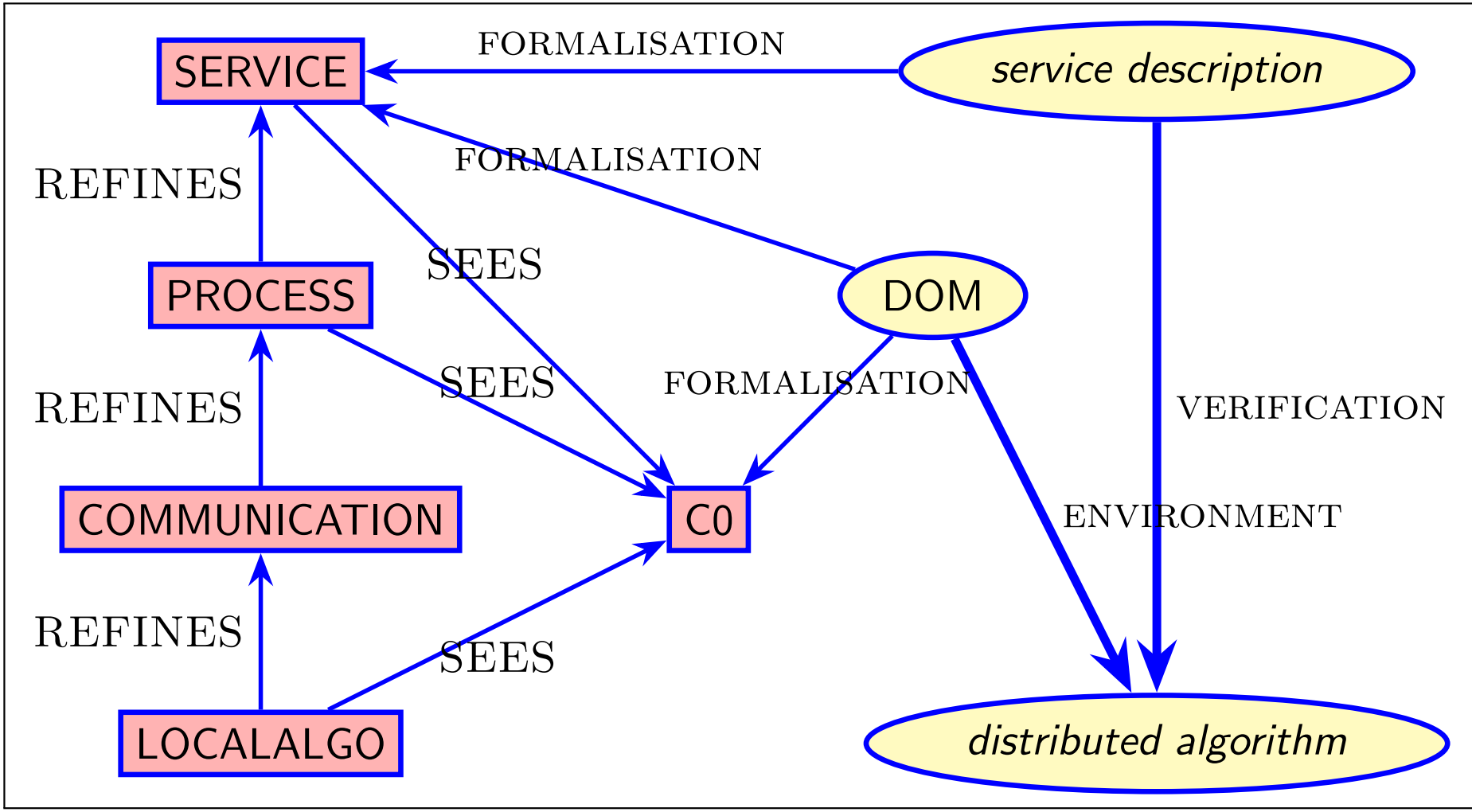
The distributed pattern



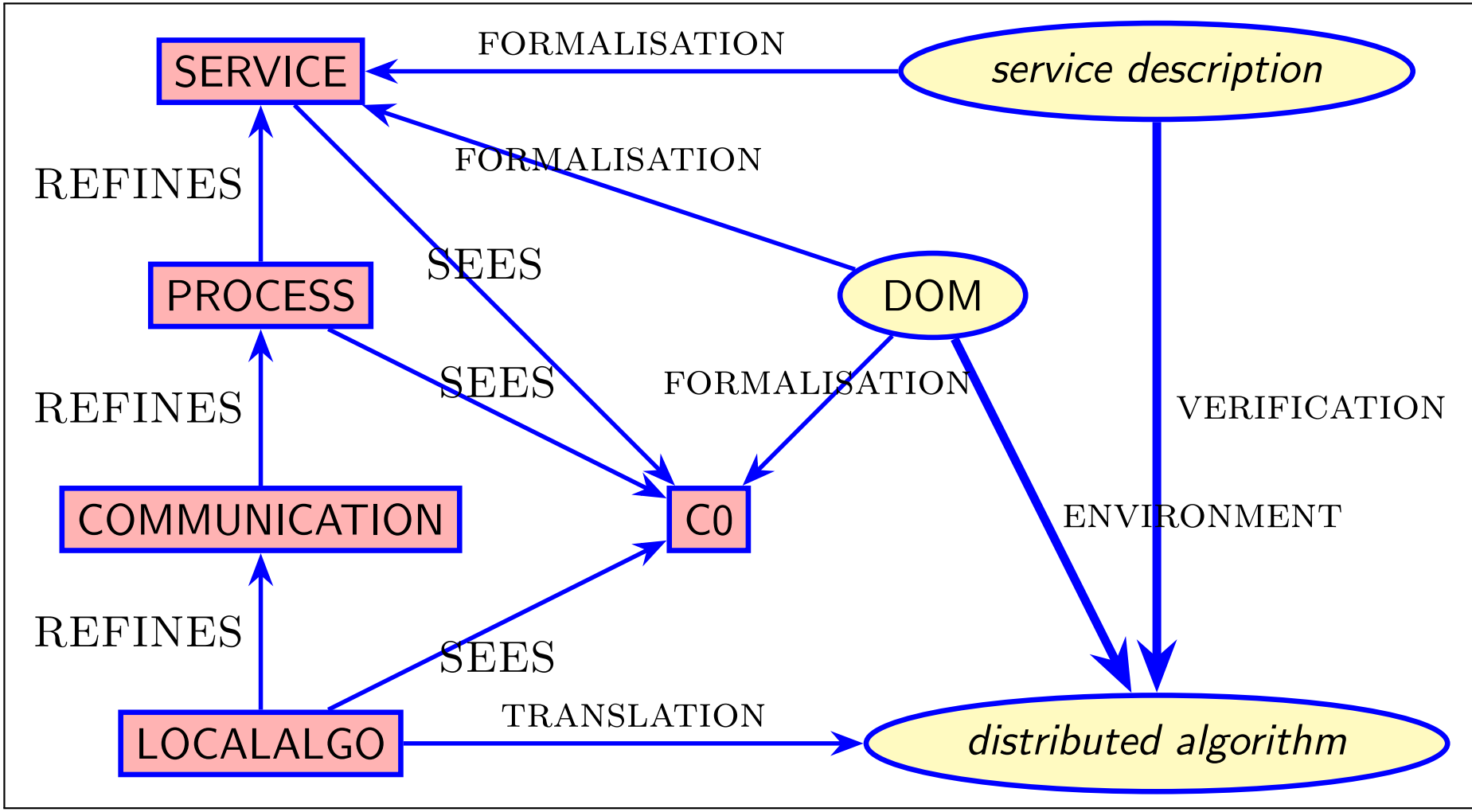
The distributed pattern



The distributed pattern



The distributed pattern



Refinement of a model by another one (I)

- Given an abstract and a corresponding concrete event

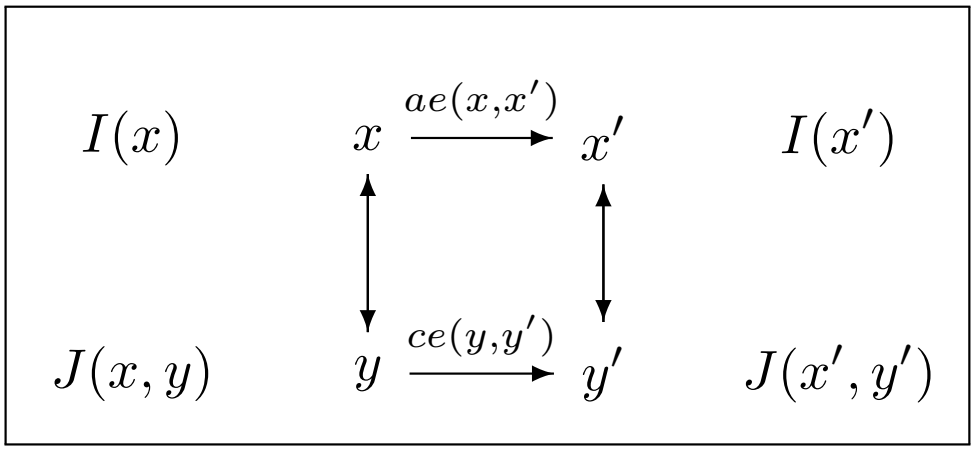
EVENT ae $\widehat{=}$
any v **where**
$G(x, v)$
then
$x := E(x, v)$
end

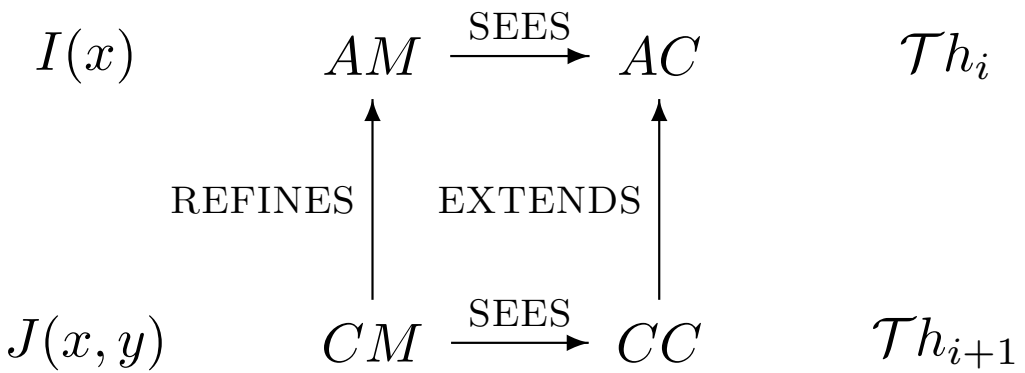
EVENT ce $\widehat{=}$
any w **where**
$H(y, w)$
then
$y := F(y, w)$
end

$$I(x) \quad \wedge \quad J(x, y) \quad \wedge \quad H(y, w)$$
$$\Longrightarrow$$
$$\exists v \cdot (G(x, v) \quad \wedge \quad J(E(x, v), F(y, w)))$$

Refinement of a model by another one (II)

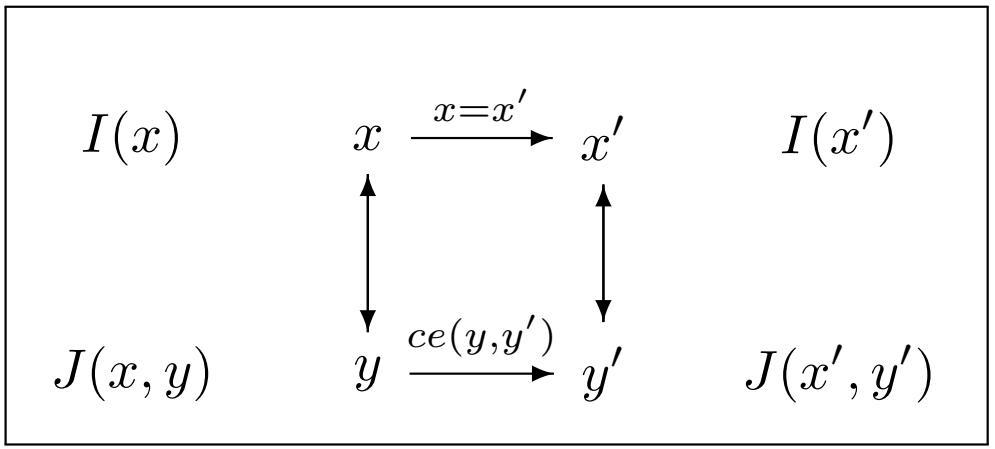
- event ce simulates event ae.
- $I(x)$ is already checked in the abstract model.
- $I(x) \wedge J(x,y) \wedge ce(y,y') \Rightarrow \exists x'.ae(x,x') \wedge J(x',y')$

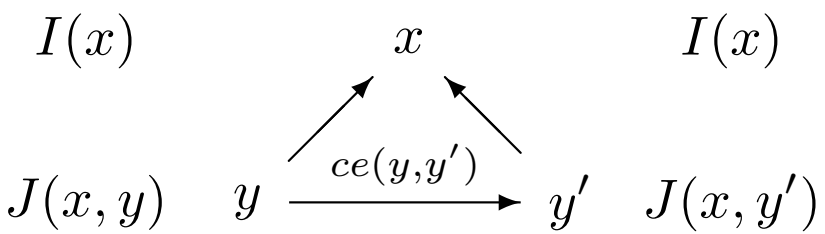




Refinement of a model by another one (III)

- event ce simulates event $skip$
- ce is a new event on new variables.
- $I(x)$ is already checked in the abstract model.
- $I(x) \wedge J(x,y) \wedge ce(y,y') \Rightarrow J(x,y')$





Distributed algorithms by *big-step* service

Distributed algorithms by *big-step* service

big-step service

The service is defined by an event called ‘magic’, which corresponds to the observation of the expected liveness property.

Distributed algorithms by *big-step* service

big-step service

The service is defined by an event called 'magic', which corresponds to the observation of the expected liveness property.



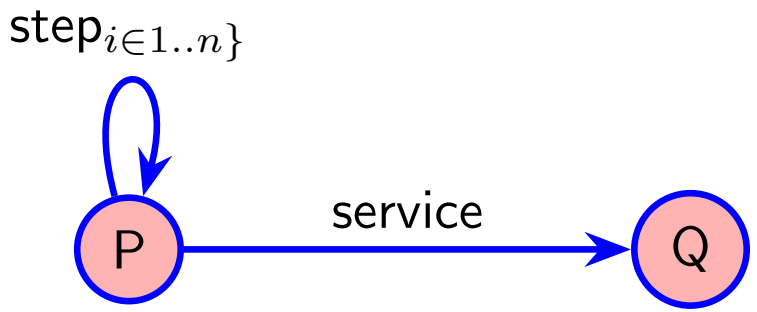
- The IEEE 1394 leader election protocol and other election algorithms
- The communication protocols as Steeening, ABP, SWP ...
- The snapshot algorithms as Chandy-Lamport or Lay and Yang.
- The naming algorithm

or $P \rightsquigarrow Q$ pronounces as P *leads to* Q

Distributed algorithms by *big-step* service

big-step service

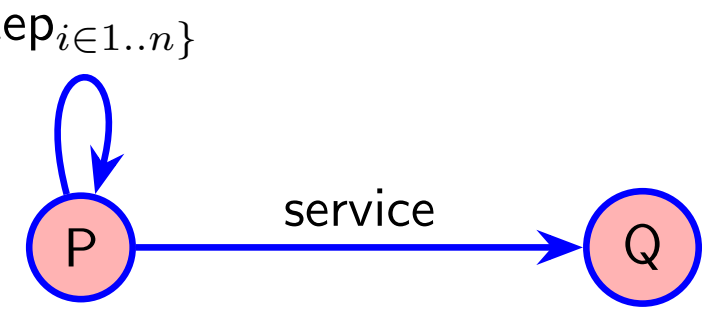
The service is defined by an event called ‘magic’, which corresponds to the observation of the expected liveness property.



Distributed algorithms by *big-step* service

big-step service

The service is defined by an event called 'magic', which corresponds to the observation of the expected liveness property.

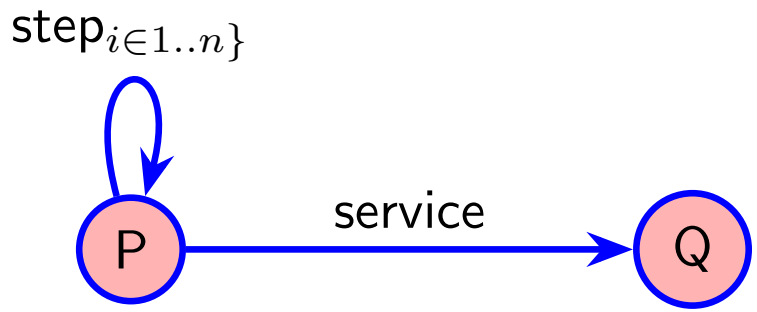


- The IEEE 1394 leader election protocol and other election algorithms

Distributed algorithms by *big-step* service

big-step service

The service is defined by an event called 'magic', which corresponds to the observation of the expected liveness property.

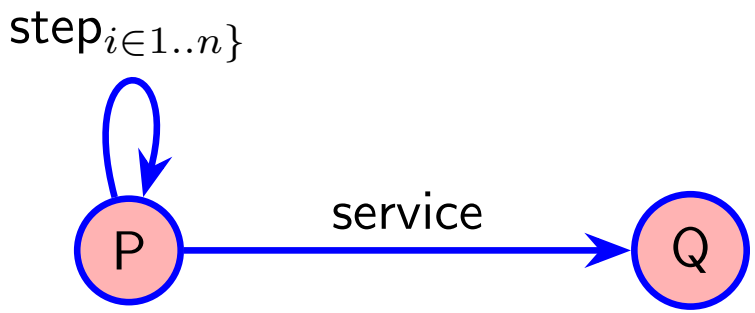


- The IEEE 1394 leader election protocol and other election algorithms
- The communication protocols as Steeening, ABP, SWP ...

Distributed algorithms by *big-step* service

big-step service

The service is defined by an event called 'magic', which corresponds to the observation of the expected liveness property.

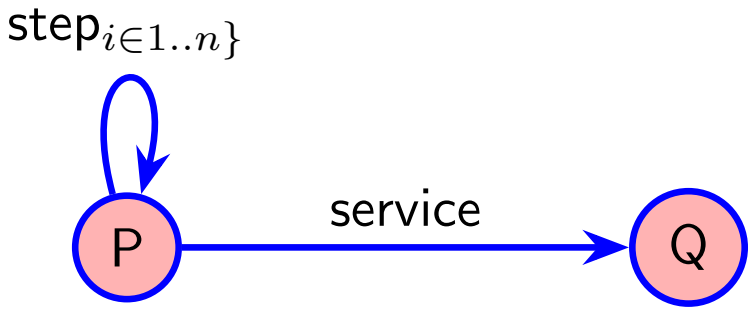


- The IEEE 1394 leader election protocol and other election algorithms
- The communication protocols as Steeening, ABP, SWP ...
- The snapshot algorithms as Chandy-Lamport or Lay and Yang.

Distributed algorithms by *big-step* service

big-step service

The service is defined by an event called 'magic', which corresponds to the observation of the expected liveness property.

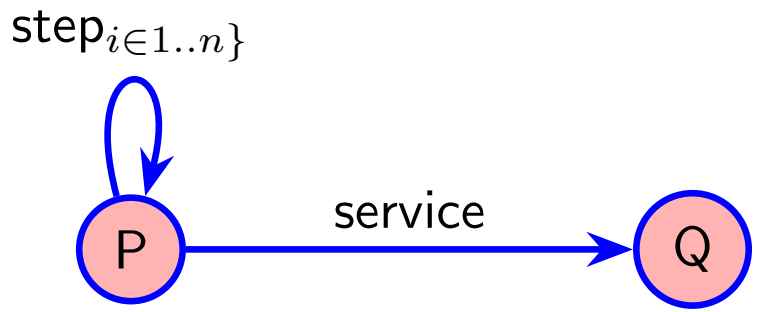


- The IEEE 1394 leader election protocol and other election algorithms
- The communication protocols as Steeening, ABP, SWP ...
- The snapshot algorithms as Chandy-Lamport or Lay and Yang.
- The naming algorithm

Distributed algorithms by *big-step* service

big-step service

The service is defined by an event called 'magic', which corresponds to the observation of the expected liveness property.



- The IEEE 1394 leader election protocol and other election algorithms
- The communication protocols as Steeening, ABP, SWP ...
- The snapshot algorithms as Chandy-Lamport or Lay and Yang.
- The naming algorithm

or $P \rightsquigarrow Q$ pronounces as P *leads to* Q

Teaching Distributed Algorithms and Systems

Abstraction

A critical reevaluation of pedagogical approaches in the context of algorithms and distributed systems.

Teaching Distributed Algorithms and Systems

Abstraction

A critical reevaluation of pedagogical approaches in the context of algorithms and distributed systems.

- Berlin 2001 at FM: *We've checked this for a graph of size 5, so we'll try it for a network of size 6*

Teaching Distributed Algorithms and Systems

Abstraction

A critical reevaluation of pedagogical approaches in the context of algorithms and distributed systems.

- Berlin 2001 at FM: *We've checked this for a graph of size 5, so we'll try it for a network of size 6* . . . but the induction lies elsewhere.

Teaching Distributed Algorithms and Systems

Abstraction

A critical reevaluation of pedagogical approaches in the context of algorithms and distributed systems.

- Berlin 2001 at FM: *We've checked this for a graph of size 5, so we'll try it for a network of size 6* . . . but the induction lies elsewhere.
- Construction of the ISO model from scratch

Teaching Distributed Algorithms and Systems

Abstraction

A critical reevaluation of pedagogical approaches in the context of algorithms and distributed systems.

- Berlin 2001 at FM: *We've checked this for a graph of size 5, so we'll try it for a network of size 6* . . . but the induction lies elsewhere.
- Construction of the ISO model from scratch
- Explanation from the service to kernel or peeling the onion . . .
- Supporting the explanation by following the refinement

Teaching Distributed Algorithms and Systems

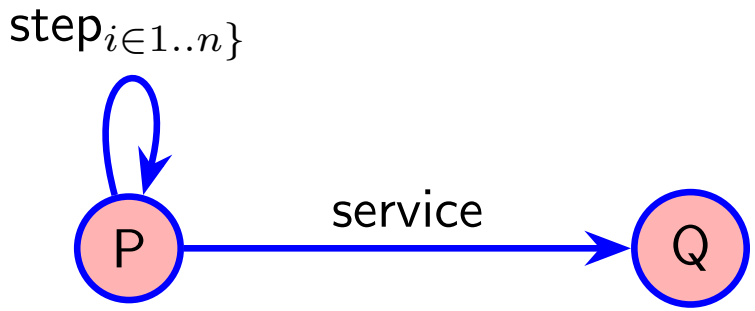
Abstraction

A critical reevaluation of pedagogical approaches in the context of algorithms and distributed systems.

- Berlin 2001 at FM: *We've checked this for a graph of size 5, so we'll try it for a network of size 6* . . . but the induction lies elsewhere.
- Construction of the ISO model from scratch
- Explanation from the service to kernel or peeling the onion . . .
- Supporting the explanation by following the refinement
- Combining Event-B and TLA$^+$.

The project ANR RIMEL

- Incremental event-based model refinement (RIMEL) focuses on proof-based patterns
- Integrating Event-B and Visidia
- Developing Bart
- Case studies related to the distributed algorithms.



Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Communications between peers

Communications between peers

- First problem in a distributed system: communication

Communications between peers

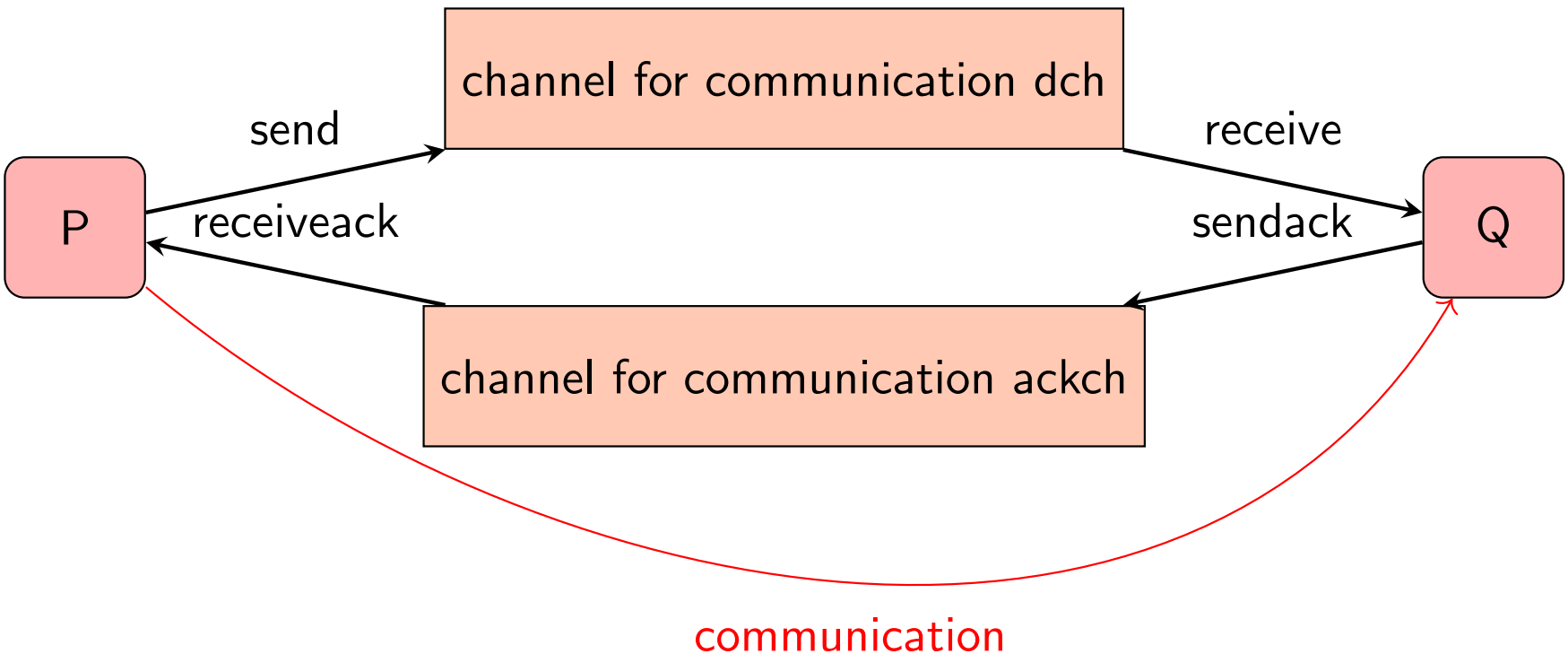
- First problem in a distributed system: communication
- Assumptions over channels as reliable, reordering, loss, mutation, attack, ...

Communications between peers

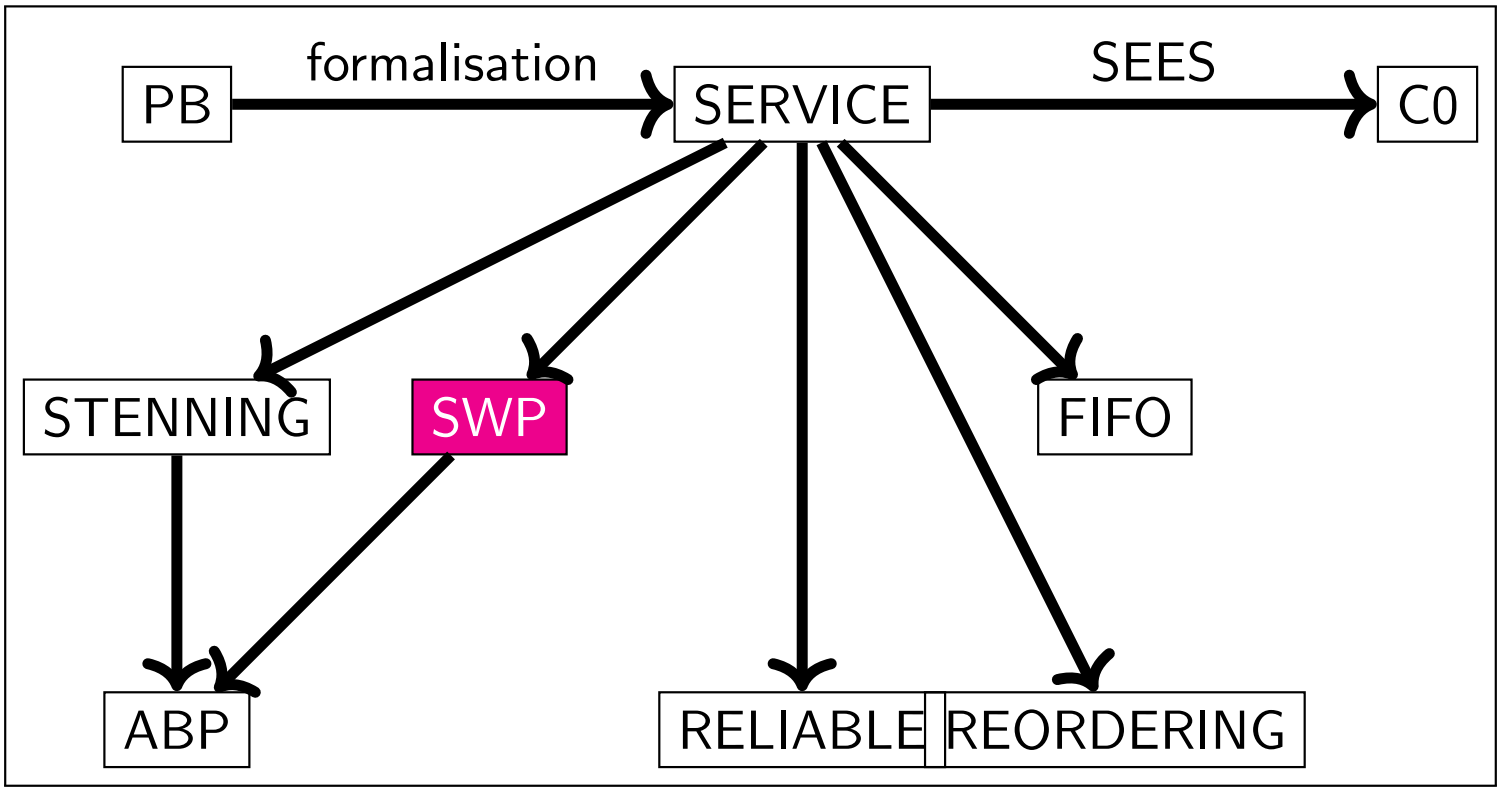
- First problem in a distributed system: communication
- Assumptions over channels as reliable, reordering, loss, mutation, attack, ...
- Protocols as Stenning, ABP, SWP ...

Communications between peers

- First problem in a distributed system: communication
- Assumptions over channels as reliable, reordering, loss, mutation, attack, ...
- Protocols as Stenning, ABP, SWP ...
- as well as authentication and key establishment protocols (see PhD N. Benaissa and composition of security mechanisms)



Relationship among protocols



SERVICE Sliding-Window Protocol



SERVICE Sliding-Window Protocol



CONTEXT $DATA$
SETS
D
CONSTANTS
n, IN, l
AXIOMS
$axm1 : n \in \mathbb{N}_1$
$axm2 : IN \in \mathbb{N} \pfun D$
$axm3 : dom(IN) = 0 \upto n$
$axm4 : l \in \mathbb{N}_1$
$axm5 : l \leq n$
END

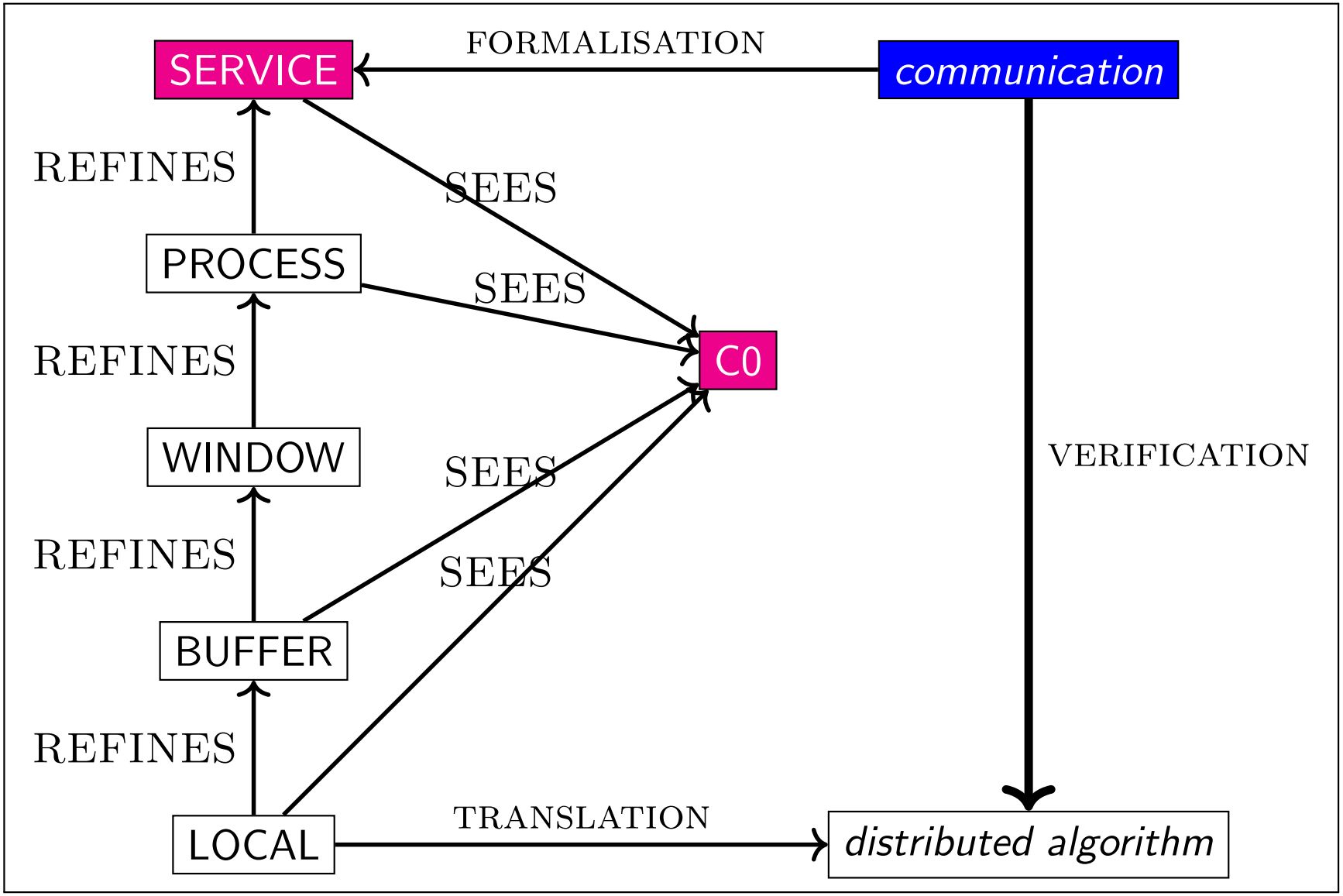
SERVICE Sliding-Window Protocol



CONTEXT $DATA$
SETS
D
CONSTANTS
n, IN, l
AXIOMS
$axm1 : n \in \mathbb{N}_1$
$axm2 : IN \in \mathbb{N} \pfun D$
$axm3 : dom(IN) = 0 \upto n$
$axm4 : l \in \mathbb{N}_1$
$axm5 : l \leq n$
END

INVARIANTS
$inv2 : OUT \in \mathbb{N} \pfun D$
EVENTS
EVENT INITIALISATION
BEGIN
$act2 : OUT := \varnothing$
END
EVENT communication
BEGIN
$act1 : OUT := IN$
END
EVENT protocoling
BEGIN
$act1 : OUT : |(OUT' \subseteq IN)$
END
END

The current project



PROCESS: the Sliding-Window Protocol

$OUT, i, chan, ack, $

$inv1 : OUT \in \mathbb{N} \pfun D$
$inv2 : i \in 0 \upto n+1$
$inv3 : 0 \upto i-1 \subseteq dom(OUT)$
$inv3bis : dom(OUT) \subseteq 0 \upto n$
$inv7 : chan \in \mathbb{N} \pfun D$
$inv8 : ack \subseteq \mathbb{N}$
$inv9 : ack \cup got \subseteq i \upto i+l \cap 0 \upto n$
$inv10 : got \subseteq \mathbb{N}$
$inv12 : dom(chan) \subseteq 0 \upto i+l \cap 0 \upto n$
$inv13 : got \subseteq dom(OUT)$
$inv14 : ack \subseteq dom(OUT)$
$inv16 : 0 \upto i-1 \domres OUT = 0 \upto i-1 \domres IN$
$inv17 : chan \subseteq IN$
$inv18 : OUT \subseteq IN \land got \subseteq dom(chan)$

PROCESS: the Sliding-Window Protocol

$$
\begin{array}{l}
inv1 : OUT \in \mathbb{N} \pfun D \\
inv2 : i \in 0 \upto n+1 \\
inv3 : 0 \upto i-1 \subseteq dom(OUT) \\
inv3bis : dom(OUT) \subseteq 0 \upto n \\
inv7 : chan \in \mathbb{N} \pfun D \\
inv8 : ack \subseteq \mathbb{N} \\
inv9 : ack \cup got \subseteq i \upto i+l \cap 0 \upto n \\
inv10 : got \subseteq \mathbb{N} \\
inv12 : dom(chan) \subseteq 0 \upto i+l \cap 0 \upto n \\
inv13 : got \subseteq dom(OUT) \\
inv14 : ack \subseteq dom(OUT) \\
inv16 : 0 \upto i-1 \domres OUT = 0 \upto i-1 \domres IN \\
inv17 : chan \subseteq IN \\
inv18 : OUT \subseteq IN \land got \subseteq dom(chan)
\end{array}
$$

$OUT, i, chan, ack, $

- Idea of the protocol: a sliding window

PROCESS: the Sliding-Window Protocol

$$
\begin{array}{l}
inv1 : OUT \in \mathbb{N} \nrightarrow D \\
inv2 : i \in 0 \mathinner{..} n+1 \\
inv3 : 0 \mathinner{..} i-1 \subseteq dom(OUT) \\
inv3bis : dom(OUT) \subseteq 0 \mathinner{..} n \\
inv7 : chan \in \mathbb{N} \nrightarrow D \\
inv8 : ack \subseteq \mathbb{N} \\
inv9 : ack \cup got \subseteq i \mathinner{..} i+l \cap 0 \mathinner{..} n \\
inv10 : got \subseteq \mathbb{N} \\
inv12 : dom(chan) \subseteq 0 \mathinner{..} i+l \cap 0 \mathinner{..} n \\
inv13 : got \subseteq dom(OUT) \\
inv14 : ack \subseteq dom(OUT) \\
inv16 : 0 \mathinner{..} i-1 \lhd OUT = 0 \mathinner{..} i-1 \lhd IN \\
inv17 : chan \subseteq IN \\
inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)
\end{array}
$$

$OUT, i, chan, ack, $

- Idea of the protocol: a sliding window
- Progressive reception of data while sliding

PROCESS: the Sliding-Window Protocol

$$
\begin{array}{l}
inv1 : OUT \in \mathbb{N} \pfun D \\
inv2 : i \in 0 \upto n+1 \\
inv3 : 0 \upto i-1 \subseteq dom(OUT) \\
inv3bis : dom(OUT) \subseteq 0 \upto n \\
inv7 : chan \in \mathbb{N} \pfun D \\
inv8 : ack \subseteq \mathbb{N} \\
inv9 : ack \cup got \subseteq i \upto i+l \cap 0 \upto n \\
inv10 : got \subseteq \mathbb{N} \\
inv12 : dom(chan) \subseteq 0 \upto i+l \cap 0 \upto n \\
inv13 : got \subseteq dom(OUT) \\
inv14 : ack \subseteq dom(OUT) \\
inv16 : 0 \upto i-1 \domres OUT = 0 \upto i-1 \domres IN \\
inv17 : chan \subseteq IN \\
inv18 : OUT \subseteq IN \land got \subseteq dom(chan)
\end{array}
$$

$OUT, i, chan, ack, $

- Idea of the protocol: a sliding window
- Progressive reception of data while sliding
- Two channels:

PROCESS: the Sliding-Window Protocol

$$
\begin{array}{l}
inv1 : OUT \in \mathbb{N} \pfun D \\
inv2 : i \in 0 \upto n+1 \\
inv3 : 0 \upto i-1 \subseteq dom(OUT) \\
inv3bis : dom(OUT) \subseteq 0 \upto n \\
inv7 : chan \in \mathbb{N} \pfun D \\
inv8 : ack \subseteq \mathbb{N} \\
inv9 : ack \cup got \subseteq i \upto i+l \cap 0 \upto n \\
inv10 : got \subseteq \mathbb{N} \\
inv12 : dom(chan) \subseteq 0 \upto i+l \cap 0 \upto n \\
inv13 : got \subseteq dom(OUT) \\
inv14 : ack \subseteq dom(OUT) \\
inv16 : 0 \upto i-1 \domres OUT = 0 \upto i-1 \domres IN \\
inv17 : chan \subseteq IN \\
inv18 : OUT \subseteq IN \land got \subseteq dom(chan)
\end{array}
$$

$OUT, i, chan, ack, $

- Idea of the protocol: a sliding window
- Progressive reception of data while sliding
- Two channels:
  - $chan$ is the data communication channel from IN into OUT

PROCESS: the Sliding-Window Protocol

$$
\begin{array}{l}
inv1 : OUT \in \mathbb{N} \nrightarrow D \\
inv2 : i \in 0 \mathinner{..} n+1 \\
inv3 : 0 \mathinner{..} i-1 \subseteq dom(OUT) \\
inv3bis : dom(OUT) \subseteq 0 \mathinner{..} n \\
inv7 : chan \in \mathbb{N} \nrightarrow D \\
inv8 : ack \subseteq \mathbb{N} \\
inv9 : ack \cup got \subseteq i \mathinner{..} i+l \cap 0 \mathinner{..} n \\
inv10 : got \subseteq \mathbb{N} \\
inv12 : dom(chan) \subseteq 0 \mathinner{..} i+l \cap 0 \mathinner{..} n \\
inv13 : got \subseteq dom(OUT) \\
inv14 : ack \subseteq dom(OUT) \\
inv16 : 0 \mathinner{..} i-1 \lhd OUT = 0 \mathinner{..} i-1 \lhd IN \\
inv17 : chan \subseteq IN \\
inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)
\end{array}
$$

$OUT, i, chan, ack,$

- Idea of the protocol: a sliding window
- Progressive reception of data while sliding
- Two channels:
  - $chan$ is the data communication channel from IN into OUT
  - ack is the acknowledgement channel.

PROCESS: the Sliding-Window Protocol

$$
\begin{array}{l}
inv1 : OUT \in \mathbb{N} \pfun D \\
inv2 : i \in 0 \upto n+1 \\
inv3 : 0 \upto i-1 \subseteq dom(OUT) \\
inv3bis : dom(OUT) \subseteq 0 \upto n \\
inv7 : chan \in \mathbb{N} \pfun D \\
inv8 : ack \subseteq \mathbb{N} \\
inv9 : ack \cup got \subseteq i \upto i+l \cap 0 \upto n \\
inv10 : got \subseteq \mathbb{N} \\
inv12 : dom(chan) \subseteq 0 \upto i+l \cap 0 \upto n \\
inv13 : got \subseteq dom(OUT) \\
inv14 : ack \subseteq dom(OUT) \\
inv16 : 0 \upto i-1 \domres OUT = 0 \upto i-1 \domres IN \\
inv17 : chan \subseteq IN \\
inv18 : OUT \subseteq IN \land got \subseteq dom(chan)
\end{array}
$$

$OUT, i, chan, ack, $

- Idea of the protocol: a sliding window
- Progressive reception of data while sliding
- Two channels:
  - $chan$ is the data communication channel from IN into OUT
  - ack is the acknowledgement channel.

PROCESS: the Sliding-Window Protocol

$OUT, i, chan, ack, $

$inv1 : OUT \in \mathbb{N} \nrightarrow D$
$inv2 : i \in 0 \,..\, n+1$
$inv3 : 0 \,..\, i-1 \subseteq dom(OUT)$
$inv3bis : dom(OUT) \subseteq 0 \,..\, n$
$inv7 : chan \in \mathbb{N} \nrightarrow D$
$inv8 : ack \subseteq \mathbb{N}$
$\mathbf{inv9 : ack \cup got \subseteq i \,..\, i+l \cap 0 \,..\, n}$
$inv10 : got \subseteq \mathbb{N}$
$inv12 : dom(chan) \subseteq 0 \,..\, i+l \cap 0 \,..\, n$
$inv13 : got \subseteq dom(OUT)$
$inv14 : ack \subseteq dom(OUT)$
$\mathbf{inv16 : 0 \,..\, i-1 \lhd OUT = 0 \,..\, i-1 \lhd IN}$
$inv17 : chan \subseteq IN$
$\mathbf{inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)}$

- Idea of the protocol: a sliding window
- Progressive reception of data while sliding
- Two channels:
  - $chan$ is the data communication channel from IN into OUT
  - ack is the acknowledgement channel.

Initialisation of the protocol

EVENT INITIALISATION
BEGIN
$act1 : OUT := \varnothing$
$act2 : i := 0$
$act4 : chan := \varnothing$
$act5 : ack := \varnothing$
$act6 : got := \varnothing$
END

Sending/Receiving

EVENT send
ANY
j
WHERE
$grd1 : j \in i \,..\, i + l$
$grd2 : j \leq n$
$grd3 : j \notin got$
THEN
$act1 : chan(j) := IN(j)$
END

EVENT receive
REFINES protocoling
ANY
j
WHERE
$grd1 : j \in dom(chan)$
$grd2 : j \in i \,..\, i + l$
THEN
$act1 : OUT(j) := chan(j)$
$act2 : ack := ack \cup \{j\}$
END

Acknowledgement/Completion

EVENT receiveack
ANY
k
WHERE
$grd1 : k \in ack$
THEN
$act1 : got := got \cup \{k\}$
$act2 : ack := ack \setminus \{k\}$
END

EVENT completion
REFINES communication
WHEN
$grd1 : i = n + 1 \wedge got = \varnothing$
THEN
$skip$
END

Managing an implicit window

The window is sliding when possible: $i + l < n$.

EVENT sliding
WHEN
$grd1 : got \neq \varnothing$
$grd3 : i \in got$
$grd4 : i + l < n$
THEN
$act1 : i := i + 1$
$act2 : got := got \setminus \{i\}$
$act3 : ack := ack \setminus \{i\}$
END

The window does not slide and slowly disappears: $i + l \geq n$.

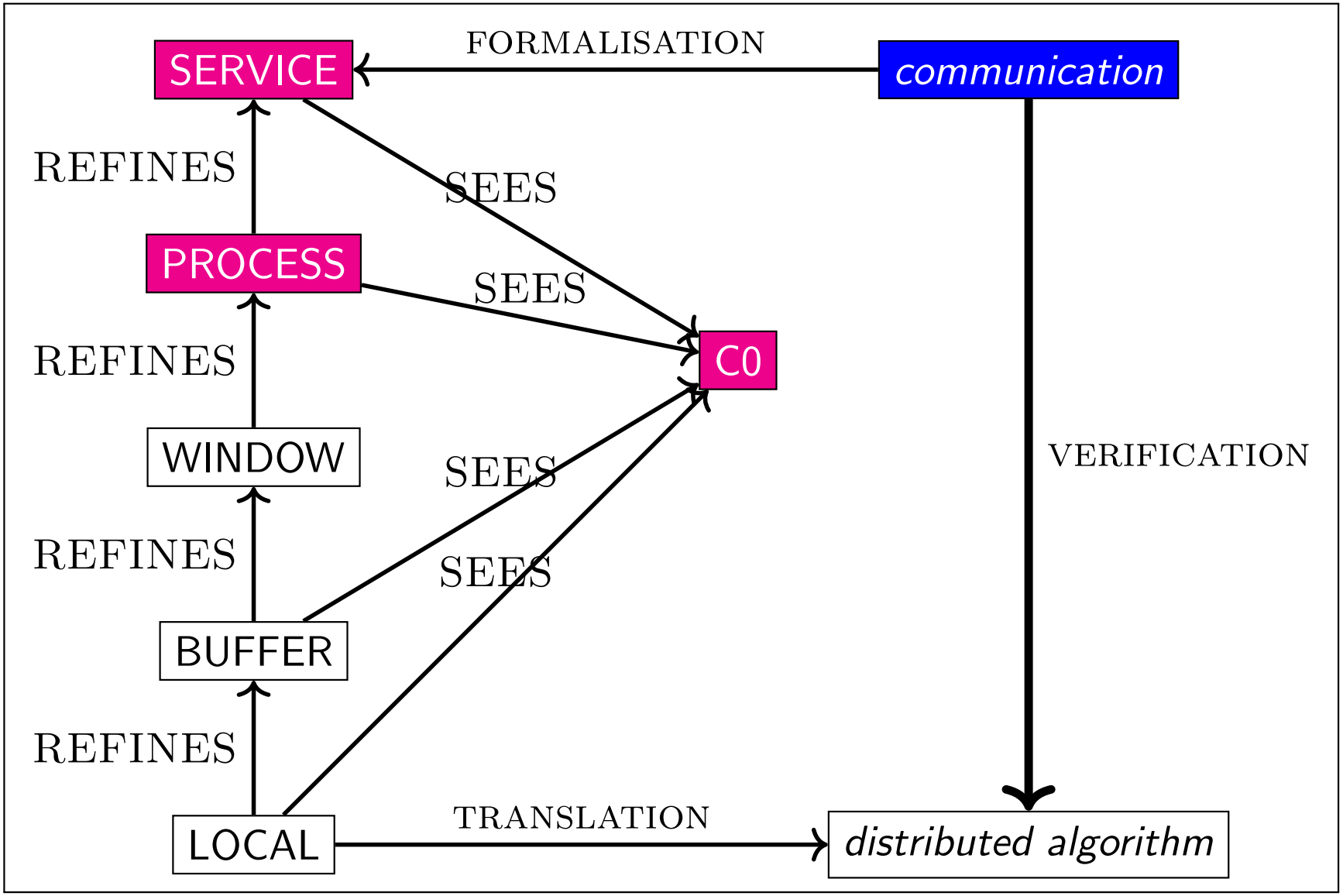
EVENT emptywindow
WHEN
$grd1 : got \neq \varnothing$
$grd2 : i \in got$
$grd3 : i + l \geq n$
$grd4 : i \leq n$
THEN
$act1 : i := i + 1$
$act2 : got := got \setminus \{i\}$
$act3 : ack := ack \setminus \{i\}$
END

Modelling loss of messages on the two channels

EVENT loosingchan
ANY
j
WHERE
$grd1 : j \in i \mathinner{..} i + l$
$grd2 : j \in dom(chan)$
$grd3 : j \notin got$
THEN
$act1 : chan := \{j\} \lhd chan$
END

EVENT loosingack
ANY
k
WHERE
$grd1 : k \in ack$
THEN
$act1 : ack := ack \setminus \{k\}$
END

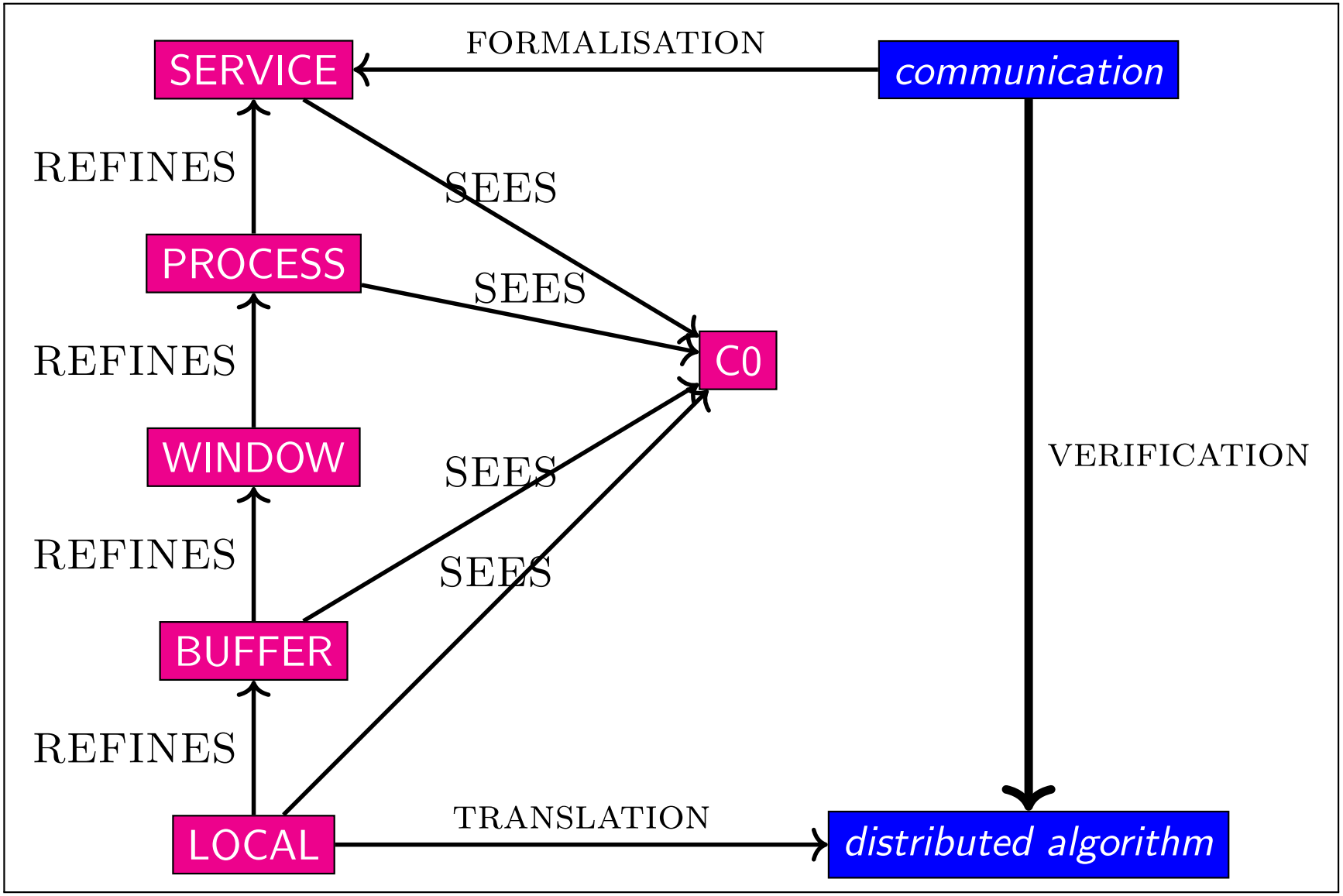
The current project



Completing the development

- WINDOW: introducing the explicit window as a new variable w.
- BUFFER: introducing the explicit buffer as a new variable b.
- LOCAL: hiding the window and the channel w and ch.

The current project



Translation Process into Programming Language



VARIABLES

OUT (local to Q)
i (local to P)
ack (channel for communicating between Q to P)
got (local to P)
b (buffer between P and Q)

INVARIANTS

$inv1 : OUT \subseteq IN$
$inv2 : 0 \mathinner{..} i-1 \lhd OUT = 0 \mathinner{..} i-1 \lhd IN$
$inv3 : i \in 0 \mathinner{..} n+1$
$inv4 : got \subseteq i \mathinner{..} i+l \cap 0 \mathinner{..} n$
$inv5 : ack \subseteq dom(OUT)$

P is sending a value to Q through b

EVENT P.sending(IN(j),Q,b(j-i)) REFINES send

ANY

j

WHERE

$grd1 : j \in i \,..\, i + l$

$grd2 : j \leq n$

$grd3 : j \notin got$

$grd4 : j - i \in 0 \,..\, l$

THEN

$act3 : b(j - i) := IN(j)$

END

P is sending a value to Q through b

P.sending(IN(j),Q,b(j-i)) $=$

$$
\boxed{
\begin{array}{l}
\textbf{if} \quad \left(\begin{array}{l} j \in i \,..\, i+l \\ j \leq n \\ j \notin got \\ j - i \in 0 \,..\, l \end{array} \right) \\
\textbf{then} \\
\quad b(j-i) := IN(j) \\
\textbf{fi}
\end{array}
}
$$

Q is receiving a value from P/ sending back an ack to P

EVENT Q.receivingandsendingack(IN(j)) REFINES receive
ANY
j
WHERE
$grd1 : j \in i \,..\, i + l$
$grd3 : j - i \in dom(b)$
THEN
$act2 : ack := ack \cup \{j\}$
$act3 : OUT(j) := b(j - i)$
END

Q is receiving a value from P/ sending back an ack to P

Q.receivingandsendingack(IN(j)) =

$$
\begin{array}{l}
\textbf{if} \quad \left(\begin{array}{l} j \in i \,..\, i+l \\ j-i \in dom(b) \end{array} \right) \\
\textbf{then} \\
\quad ack := ack \cup \{j\} \\
\quad OUT(j) := b(j-i) \\
\textbf{fi}
\end{array}
$$

Q is receiving a value from P/ sending back an ack to P

Q.receivingandsendingack(IN(j)) =

$$\textbf{if} \quad \left(\begin{array}{l} j \in i \,..\, i+l \\ j-i \in dom(b) \end{array} \right)$$
$$\textbf{then}$$
$$sending(<\text{"}ack\text{"}, j>, P)$$
$$OUT(j) := b(j-i)$$
$$\textbf{fi}$$

Producing programming statements

P.sending(IN(j),Q,b(j-i)) =

$$\textbf{if} \left(\begin{array}{l} j \in i \,..\, i+l \\ j \leq n \\ j \notin got \\ j - i \in 0 \,..\, l \end{array} \right) \textbf{then} \; b(j-i) := IN(j) \; \textbf{fi}$$

Q.receivingandsendingack(IN(j)) =

$$\textbf{if} \left(\begin{array}{l} j \in i \,..\, i+l \\ j - i \in dom(b) \end{array} \right) \textbf{then} \; sending(<\text{"}ack\text{"}, j>, P) \; OUT(j) := b(j-i) \; \textbf{fi}$$

Summary on the sliding window protocol

- Using refinement for hiding the channel and the window.
- Integration of loss of messages
- Manual proofs for manipulating arrays.

Name	Total	Automatic	Interactive
SWP	124	101	23
C0	1	1	0
SERVICE	4	2	2
PROCESS	63	51	12
WINDOW	19	13	6
BUFFER	21	18	3
LOCAL	16	16	0

Summary on protocols

- The refinement helps to understand the different steps of the *implementation*.
- Daemons are useful mates for handling loss of messages in an explicit way.
- Relationship between solutions using refinement.
- Management of attaccks is an important issue.

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Examples

Exemple 1 sensor network collecting data

- In a network where each sensor transmits measurements to its neighbours for distributed aggregation.
- The DS algorithm determines when all sensors have finished their transmissions.
- Once the data has been propagated and no messages are in transit, the initiator detects the end of the calculation.

Examples

Exemple 2 Multi-agent system performing distributed exploration

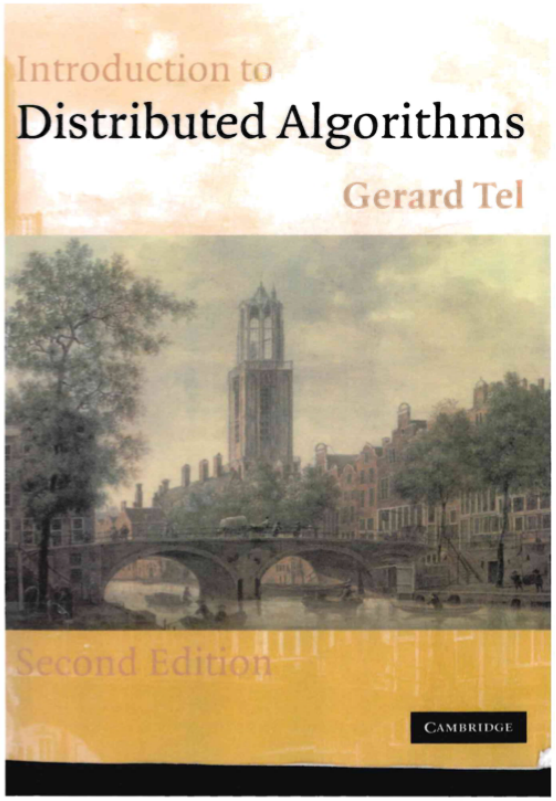
- In a graph exploration by several autonomous agents, each agent activates other agents according to the areas to be explored.
- The algorithm ensures that we know exactly when all agents have completed their exploration and that no tasks are pending.

Examples

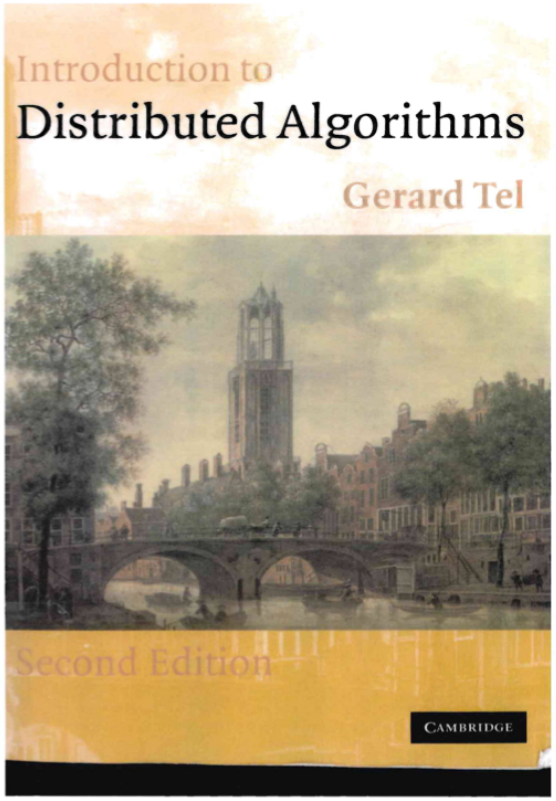
Exemple 3 Parallel computating using a cluster

- During parallel computation where nodes dynamically distribute subtasks, it can be difficult to know when no more subtasks are in progress.
- The Dijkstra-Scholten algroithm ensures correct detection of the end of processing without heavy centralised coordination.

The Dijkstra-Scholten algorithm



The Dijkstra-Scholten algorithm




```
var state_p  : (active, passive)   init if p = p_0 then active else passive ;
    sc_p     : integer             init 0 ;
    father_p : P                   init if p = p_0 then p else udef ;

S_p: { state_p = active }
   begin send ⟨mes, p⟩ ; sc_p := sc_p + 1 end

R_p: { A message ⟨mes, q⟩ has arrived at p }
   begin receive ⟨mes, q⟩ ; state_p := active ;
         if father_p = udef then father_p := q else send ⟨sig, q⟩ to q
   end

I_p: { state_p = active }
   begin state_p := passive ;
         if sc_p = 0 then (* Delete p from T *)
           begin if father_p = p
                    then Announce
                    else send ⟨sig, father_p⟩ to father_p ;
                 father_p := udef
           end
   end

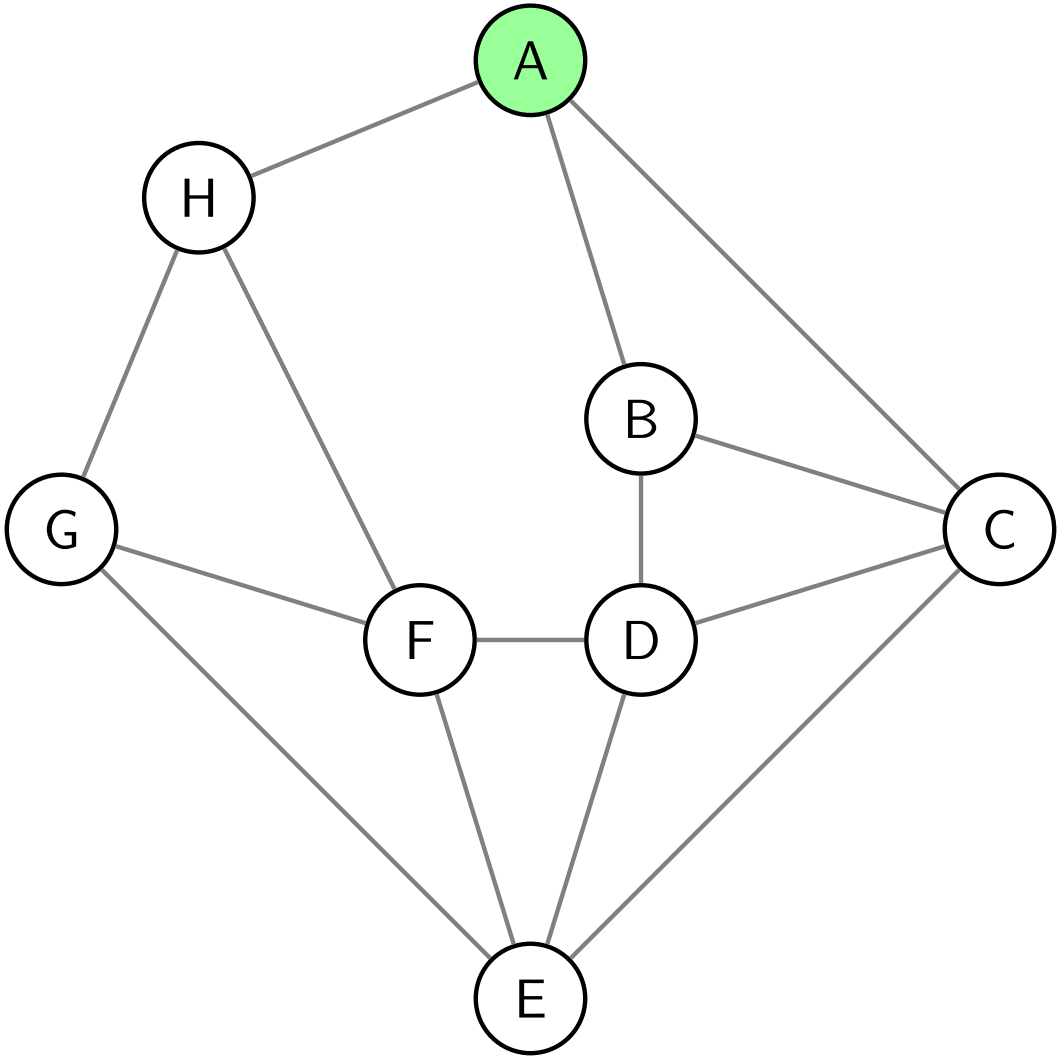
A_p: { A signal ⟨sig, p⟩ arrives at p }
   begin receive ⟨sig, p⟩ ; sc_p := sc_p − 1 ;
         if sc_p = 0 and state_p = passive then
           begin if father_p = p
                    then Announce
                    else send ⟨sig, father_p⟩ to father_p ;
                 father_p := udef
           end
   end
```

Algorithm 8.3 THE DIJKSTRA–SCHOLTEN ALGORITHM.

The Dijkstra-Scholten algorithm

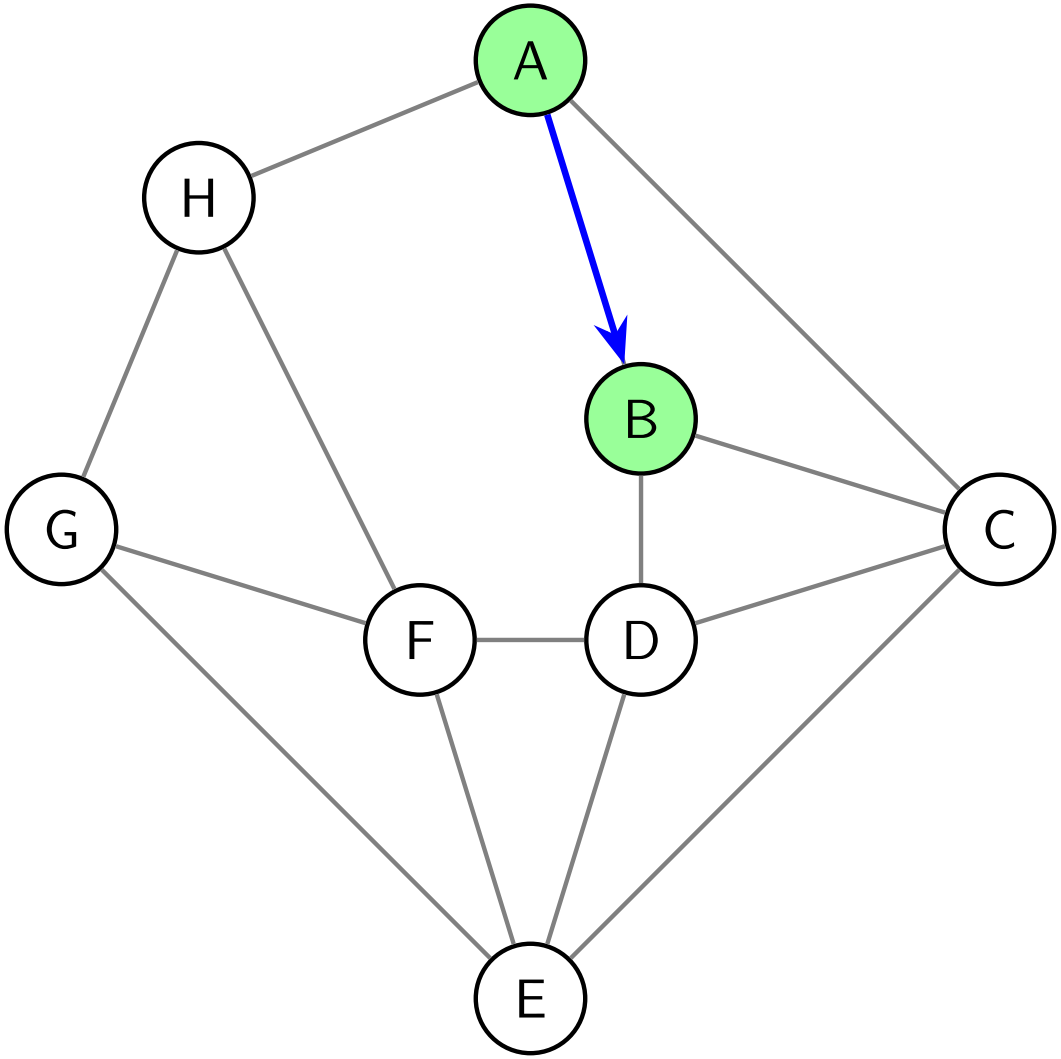
- The Dijkstra-Scholten algorithm determines when all processes in a distributed computation have completed their work and when no messages are still in transit.
- The main idea is to implicitly construct a *directed spanning tree* from the initiating process becoming the root of the tree.
- Each process that becomes active due to a received message becomes a child in this tree.
- When a process finishes its local activity and no longer has any active children, it sends a return message (acknowledgement) to its parent.
- When the root (starting process) receives all the returns from its children, global termination is detected.

Building a spanning tree



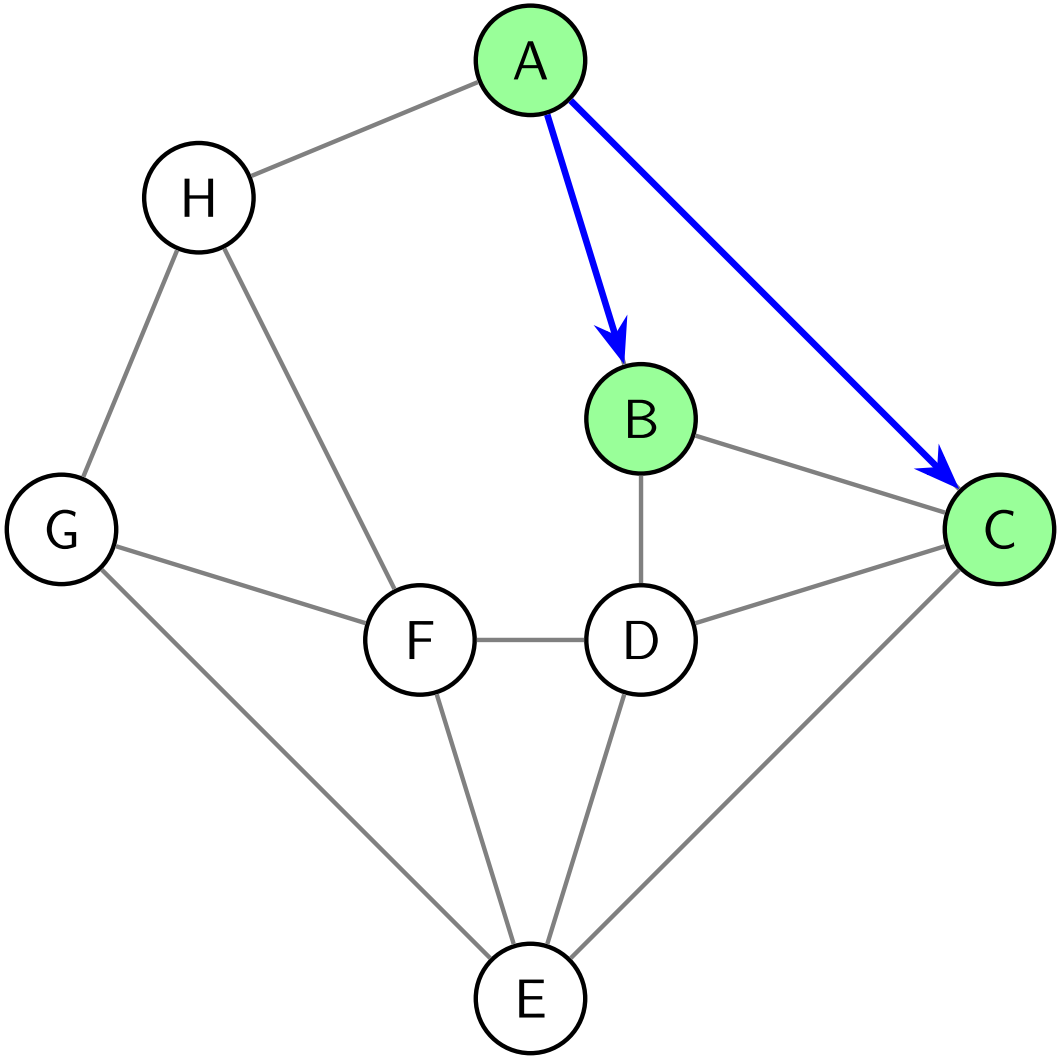
Initial undirected graph

Building a spanning tree



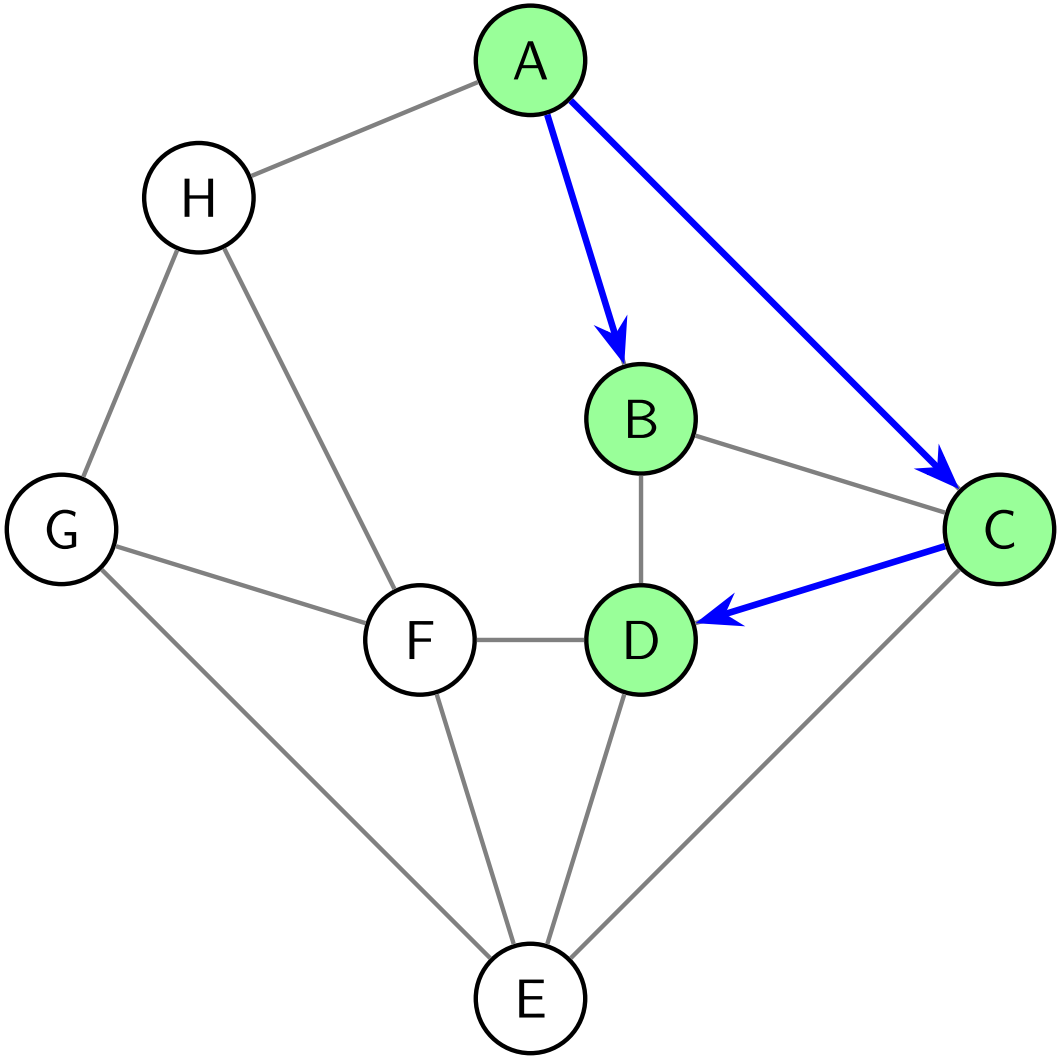
Adding the link A→B

Building a spanning tree



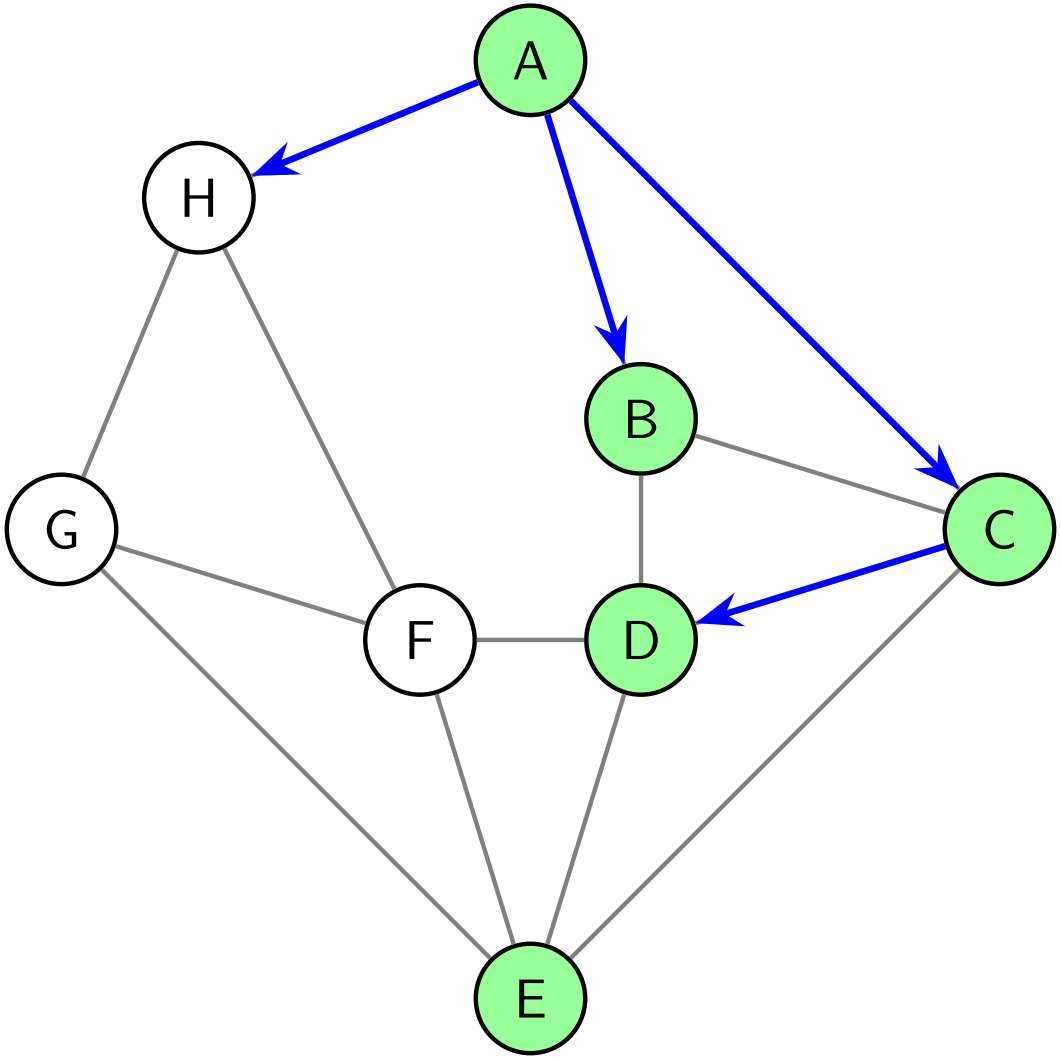
Adding the link A→C

Building a spanning tree



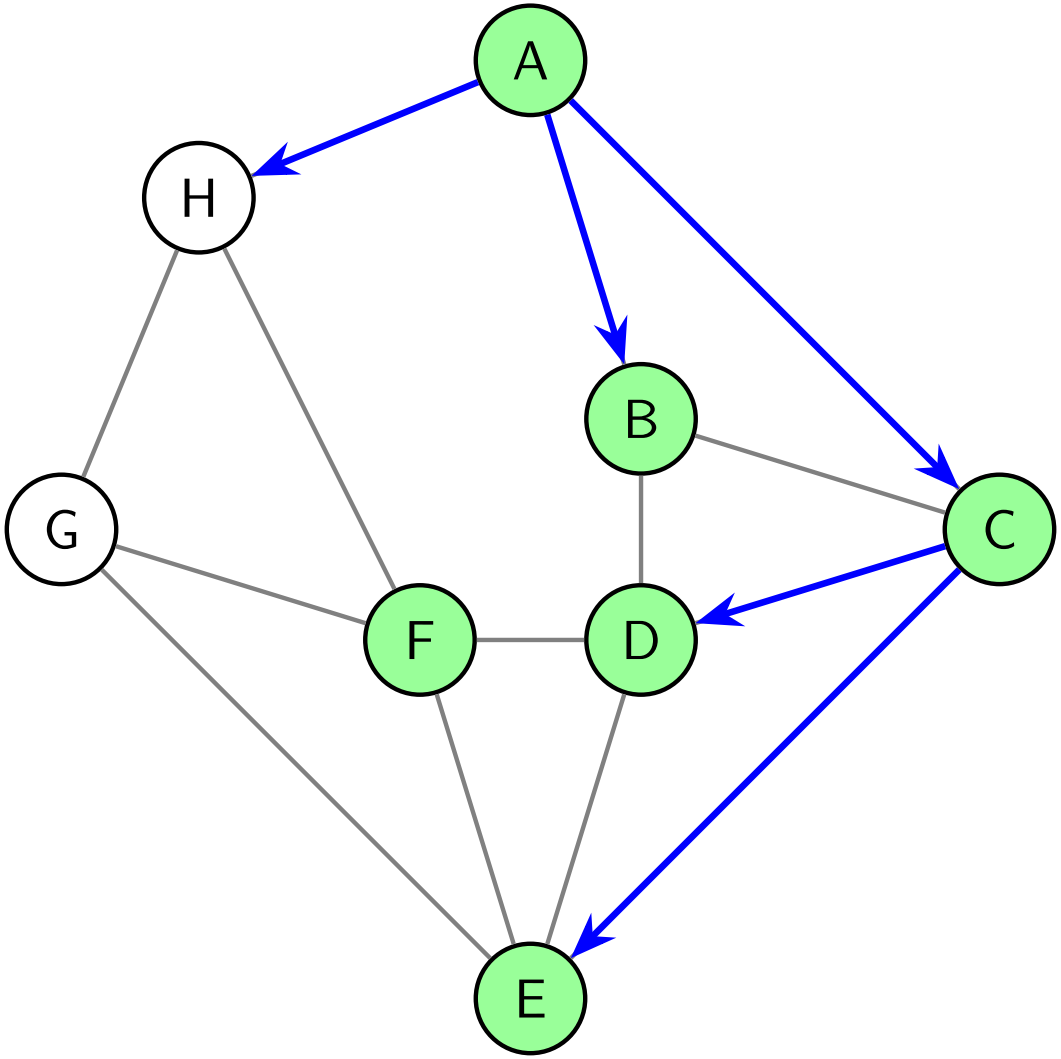
Adding the link C→D

Building a spanning tree



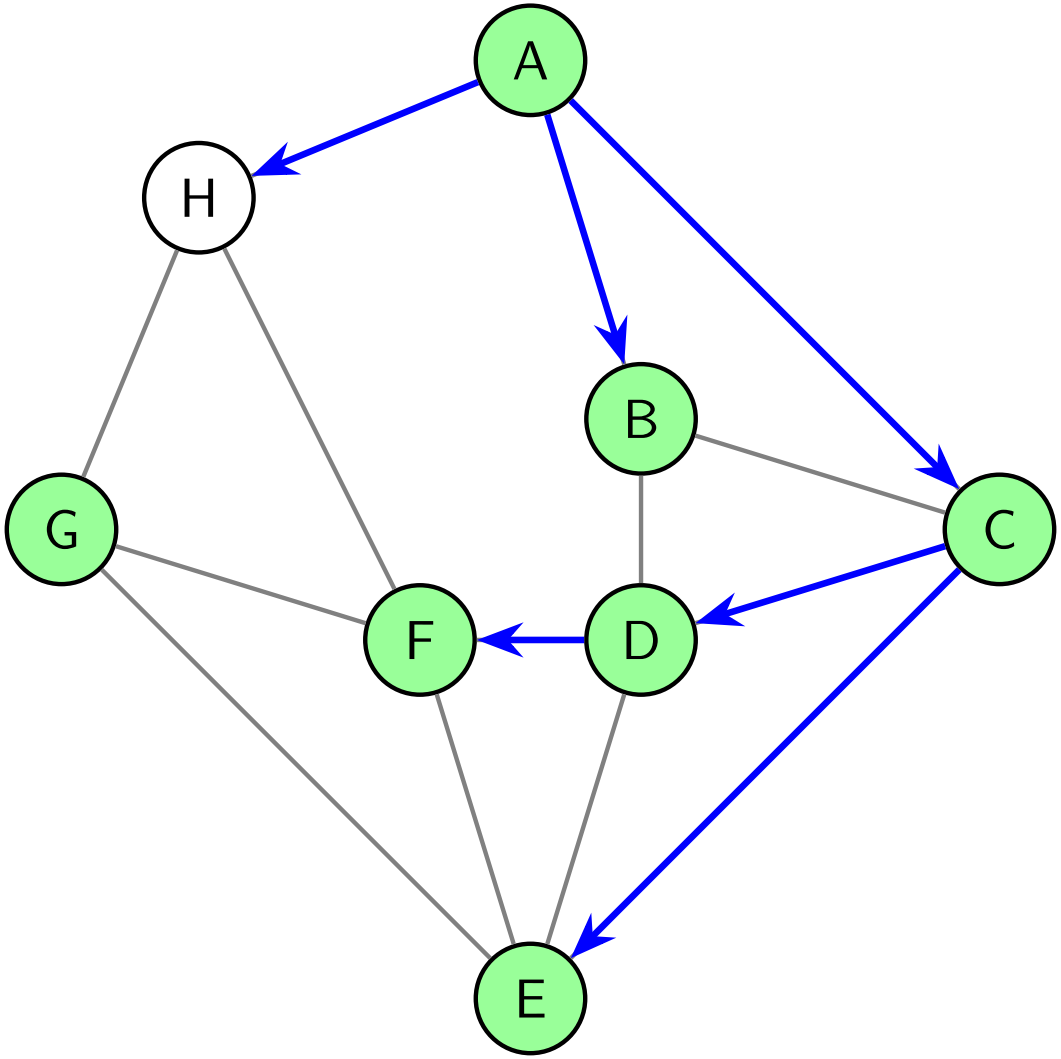
Adding the link A→H

Building a spanning tree



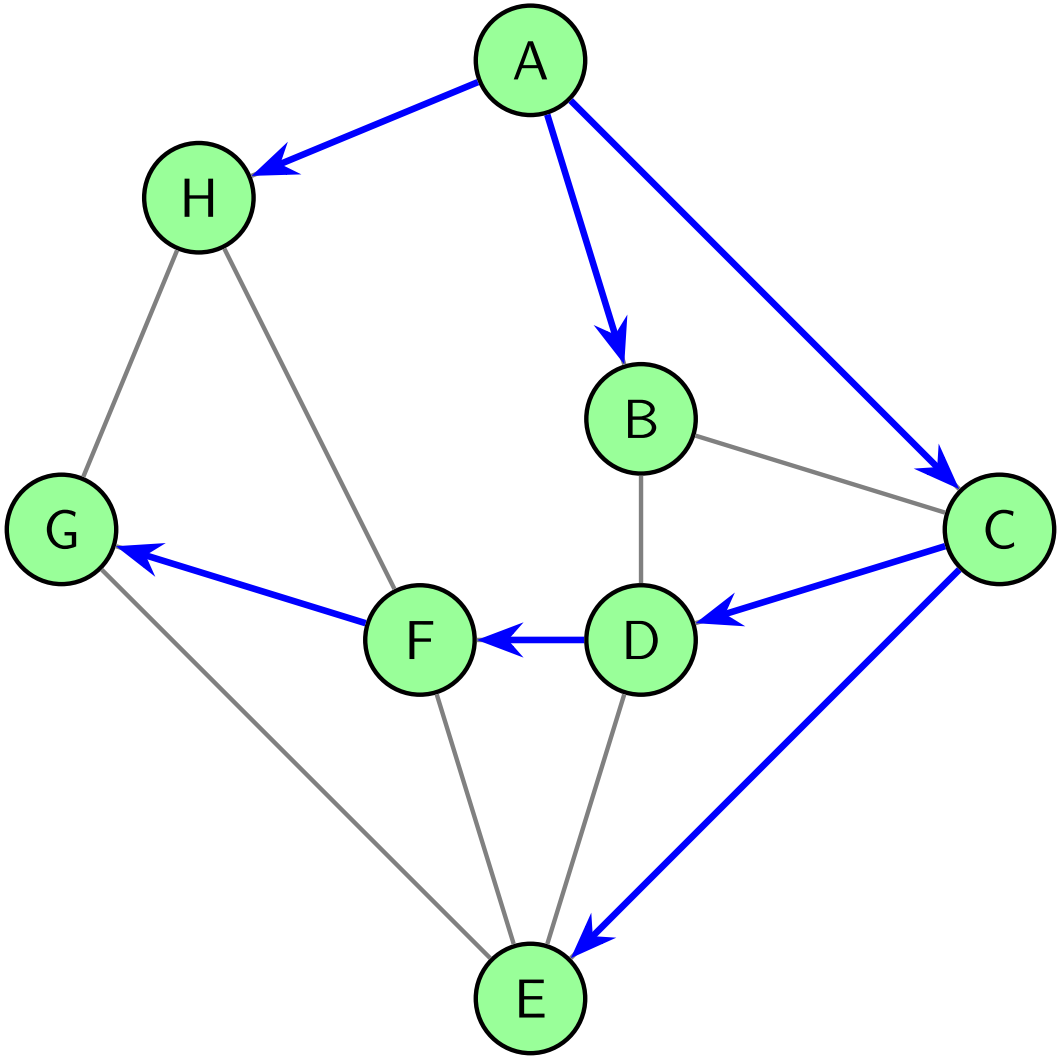
Adding the link C→E

Building a spanning tree



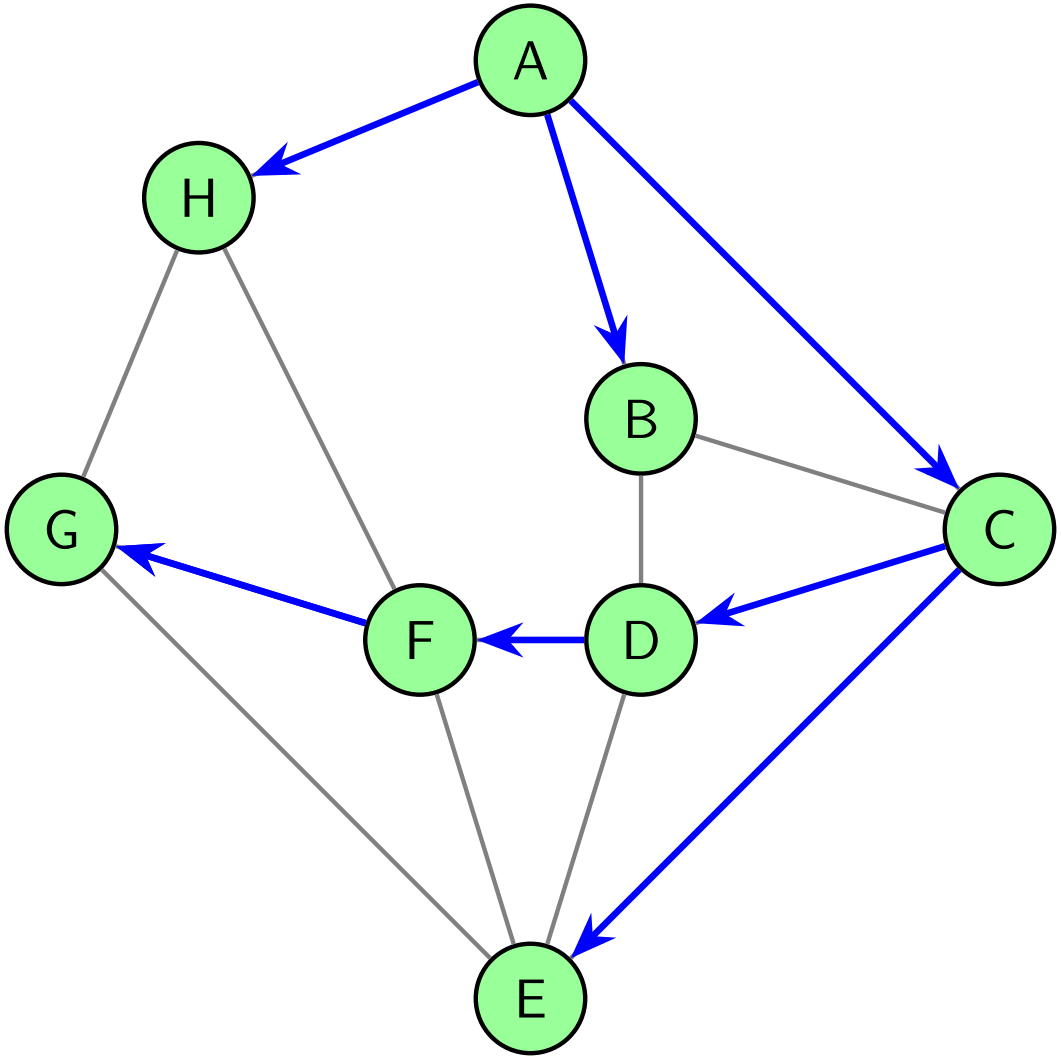
Adding the link D→F

Building a spanning tree



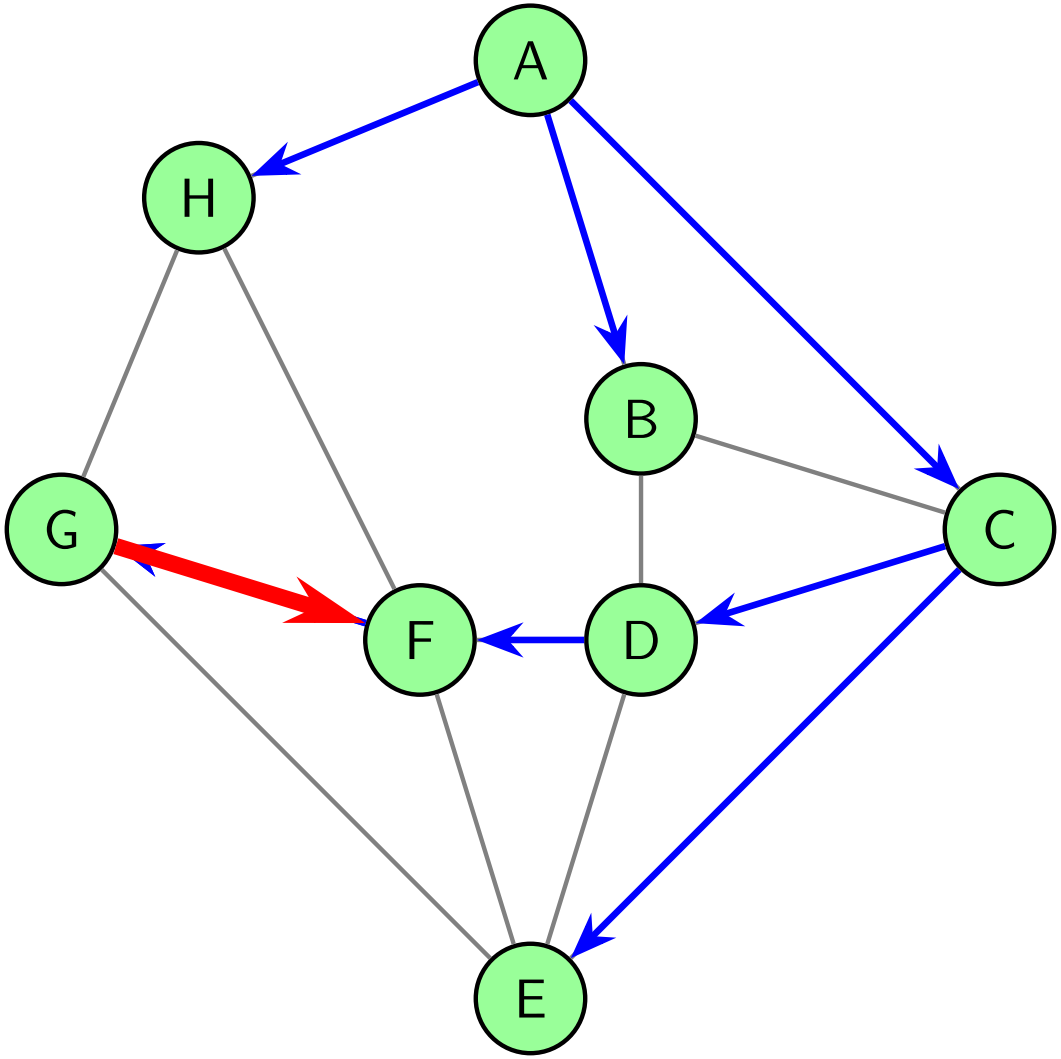
Adding the link F→G

Building a spanning tree



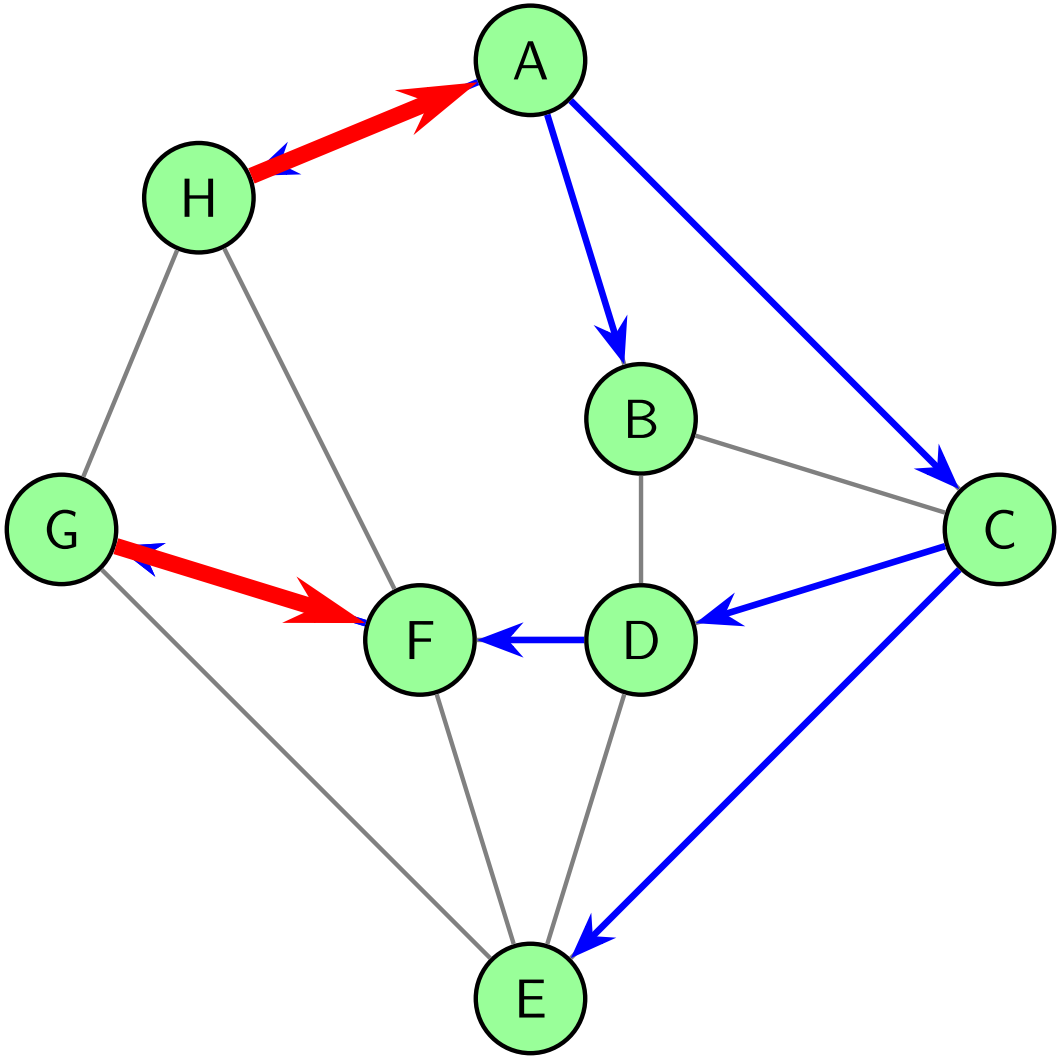
Final spanning tree

Building a spanning tree



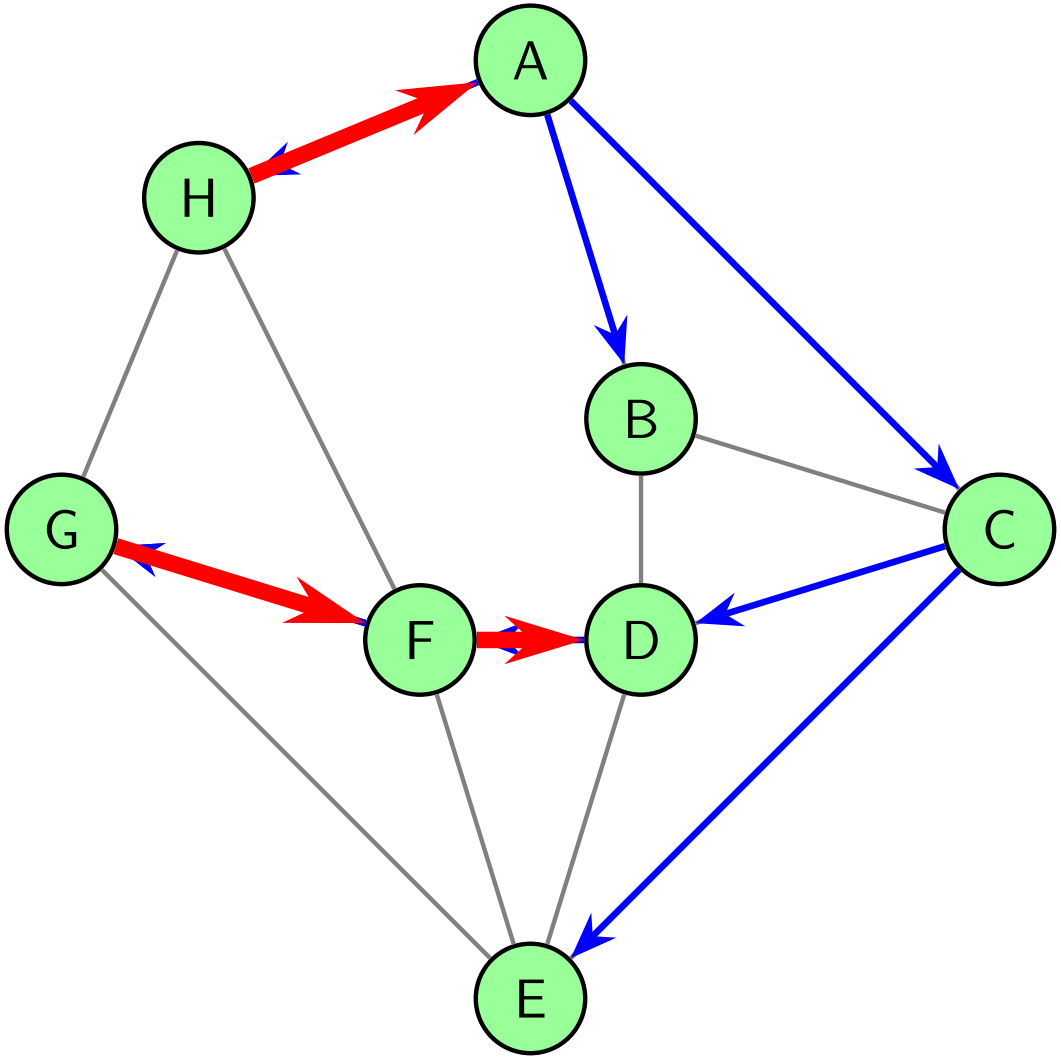
Final spanning tree

Building a spanning tree



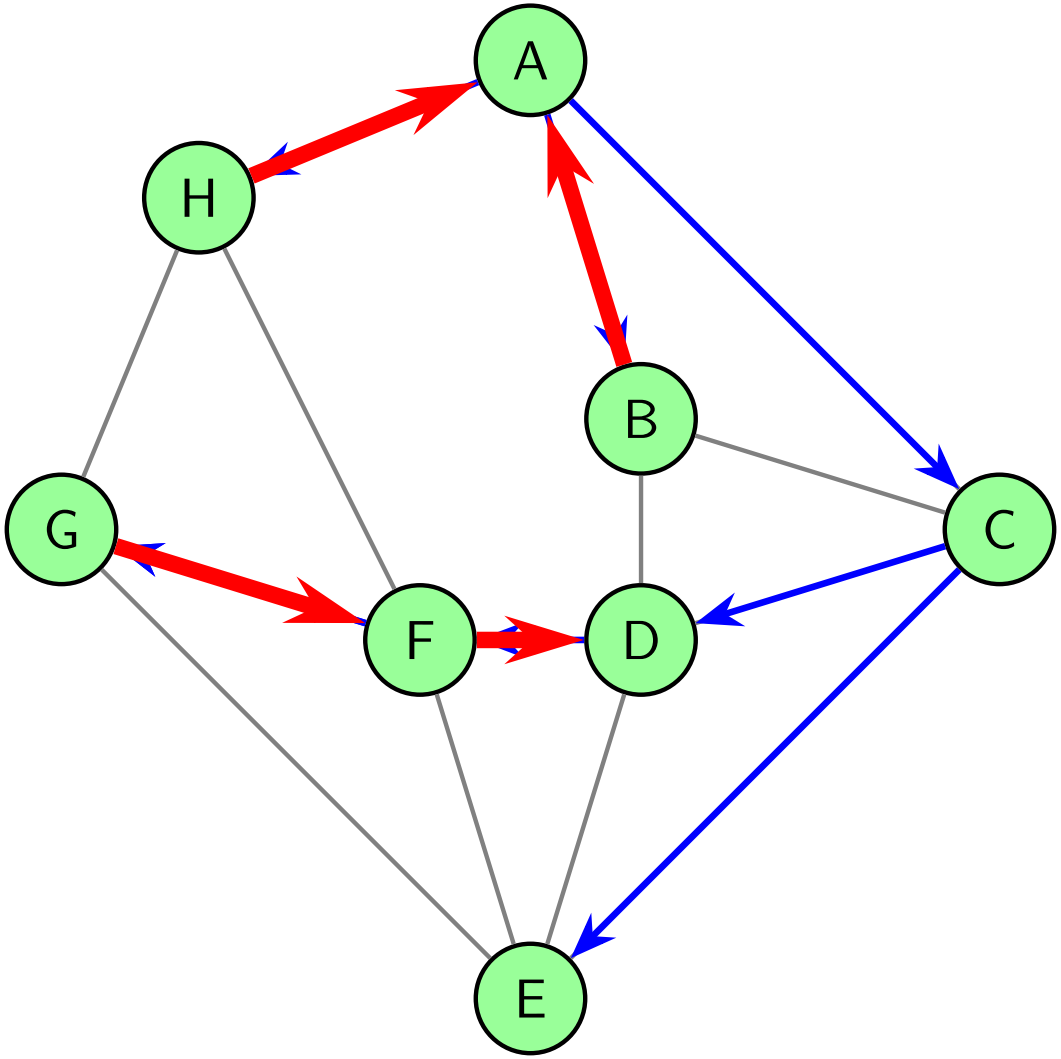
Final spanning tree

Building a spanning tree



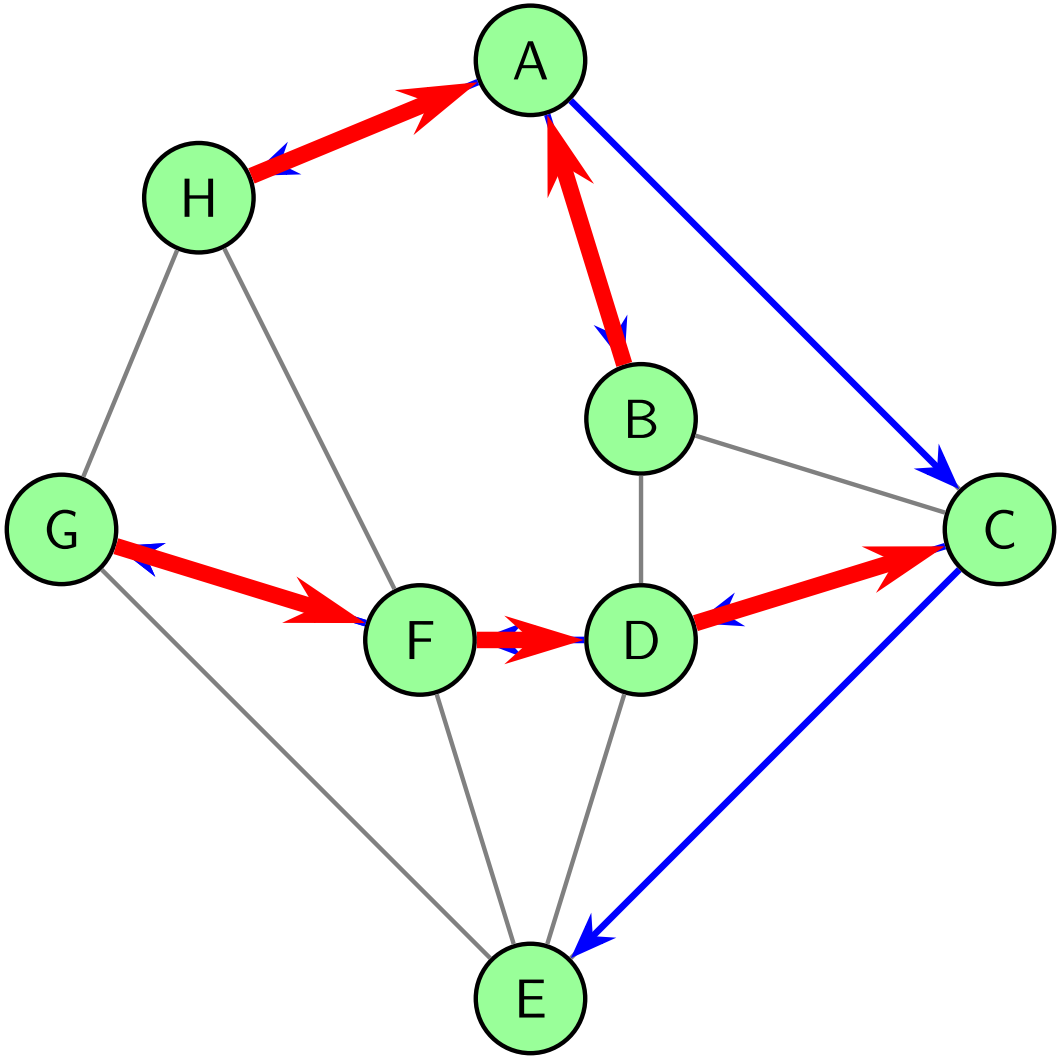
Final spanning tree

Building a spanning tree



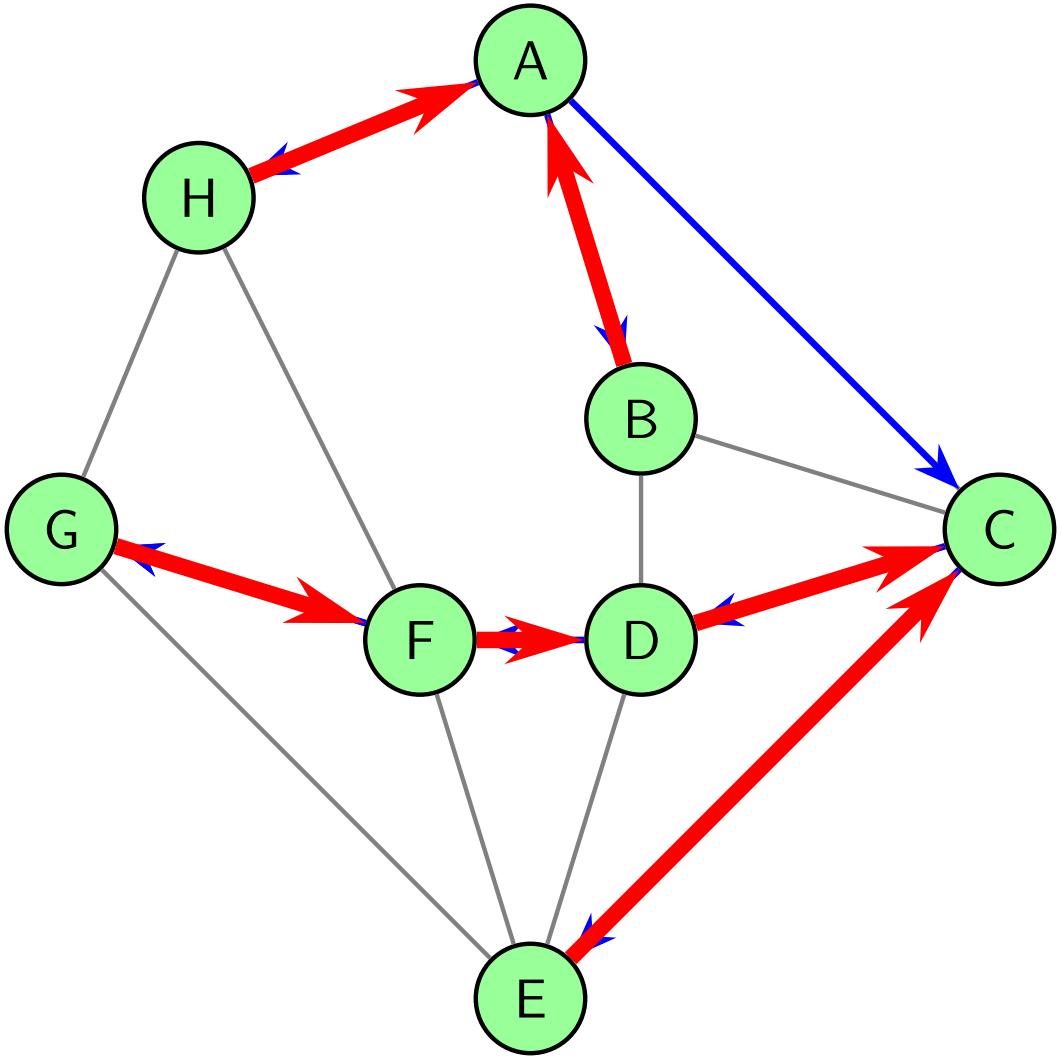
Final spanning tree

Building a spanning tree



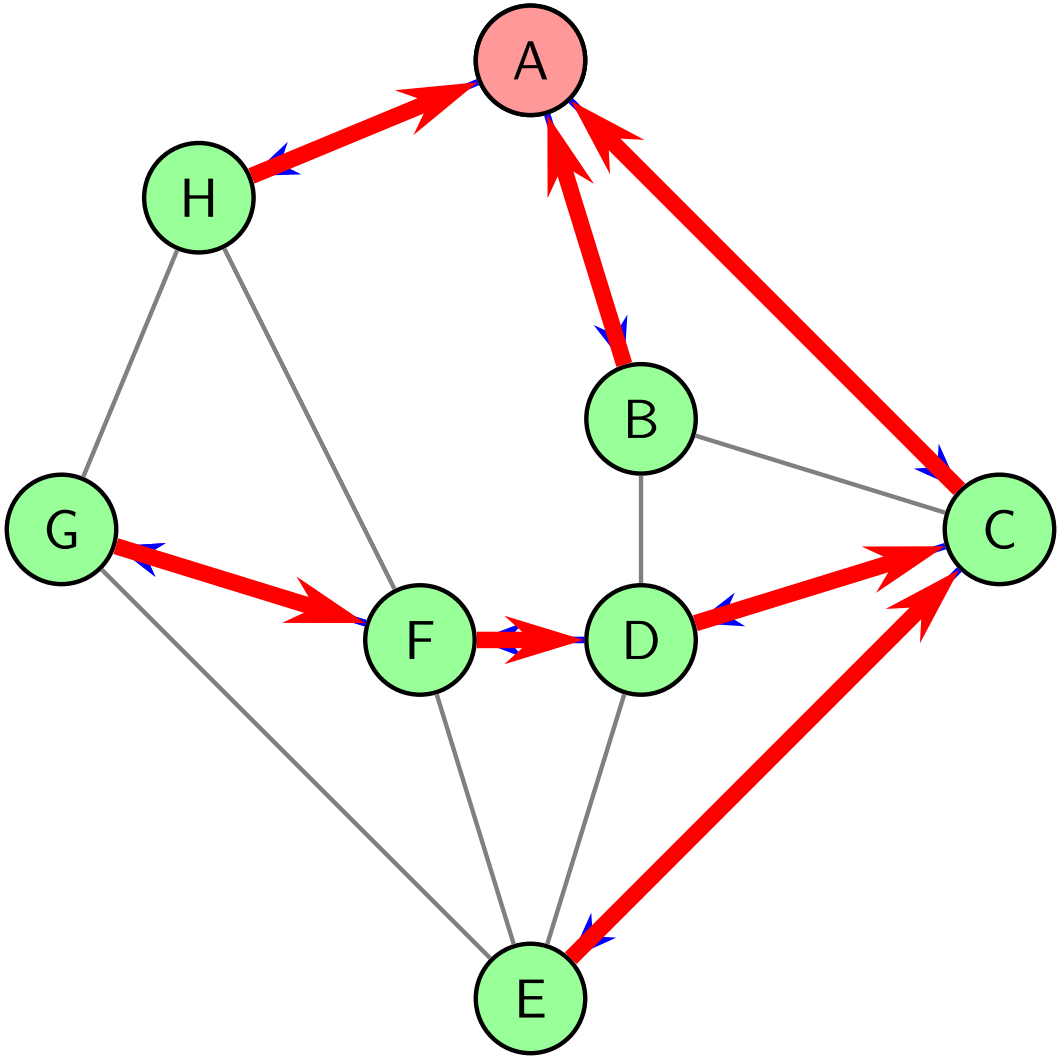
Final spanning tree

Building a spanning tree



Final spanning tree

Building a spanning tree



Final spanning tree A knows that every node has terminated

The Dijkstra-Scholten algorithm

- A set of nodes N and a relation between the nodes form a connected graph.
- A node is distinguished and launches an exploration of the nodes that will execute an algorithm.
- The idea is to construct a dynamic covering tree that will have locally executed algorithms. These are calculations by diffusion.
- The algorithm controls the construction and disappearance of the tree, which means that the initiating node knows when all the nodes have finished their respective tasks.

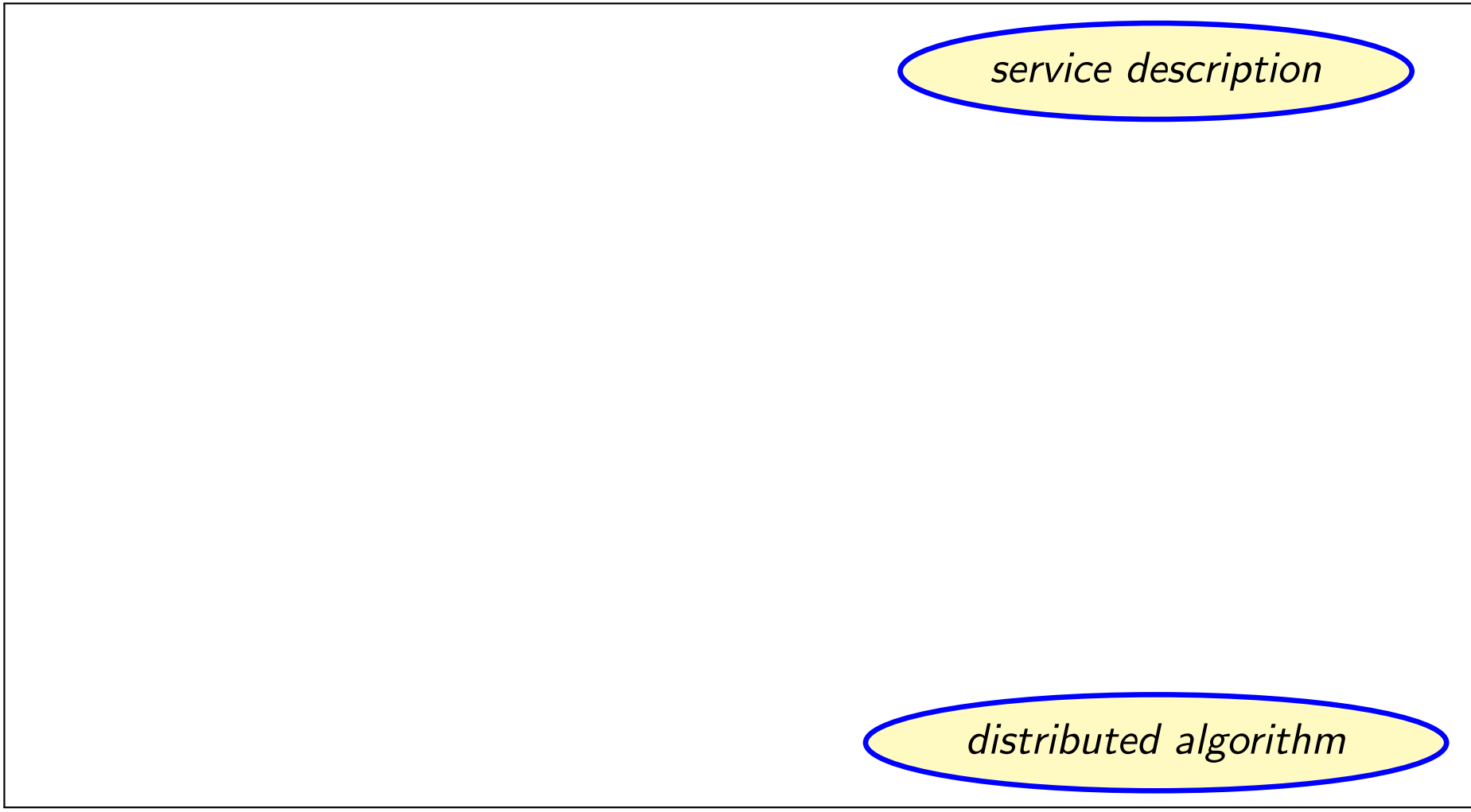
The spanning tree for the control of computing

- A two phases process
  - with a up phase building the spanning tree
  - and a down phase for controling that children of a node have terminated the local computations.
- Each node become active when he is added to the spanning tree.
- When a node has terminated his local computation, then he waits for the terminatation of his children and then send a message to his parent.

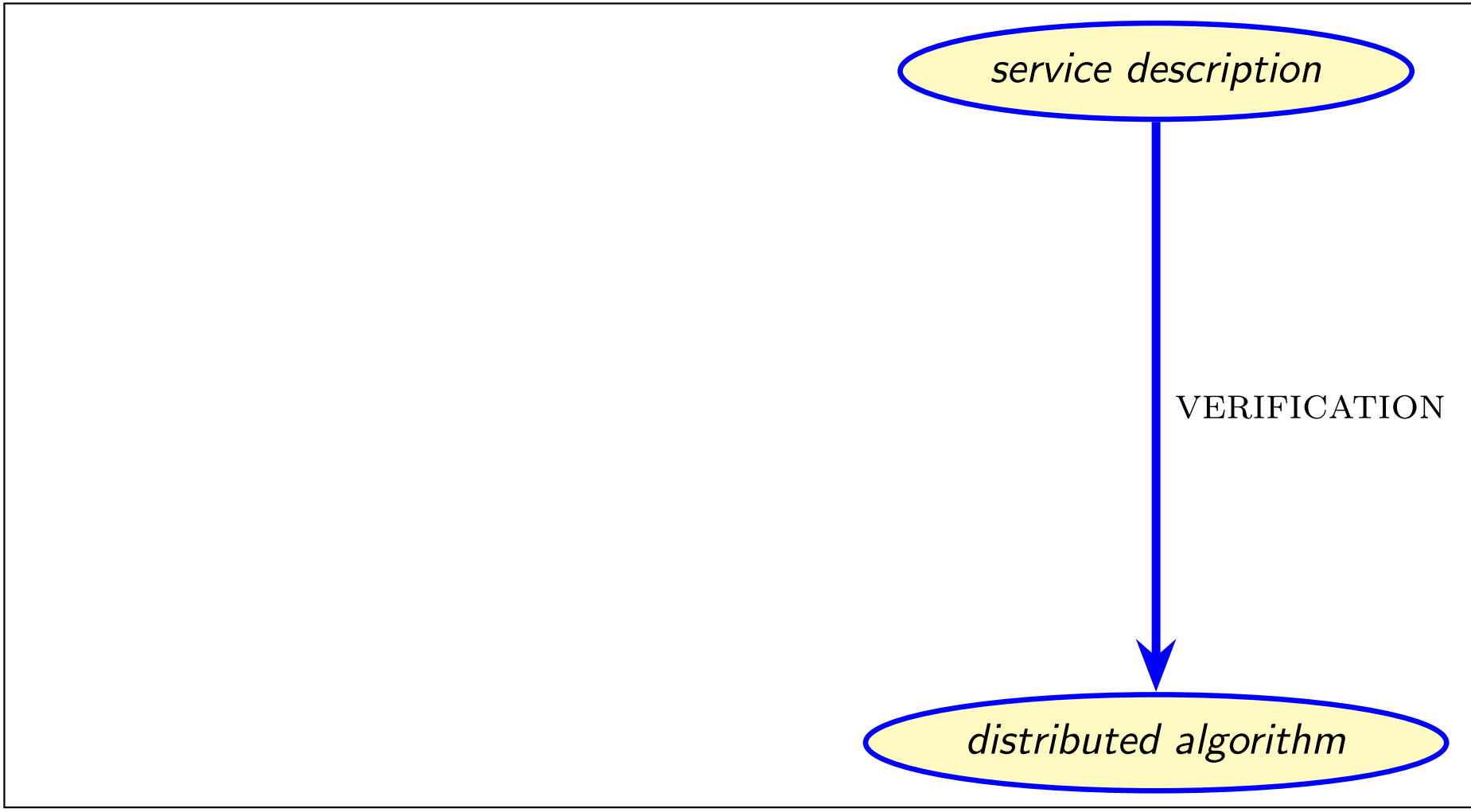
The distributed pattern



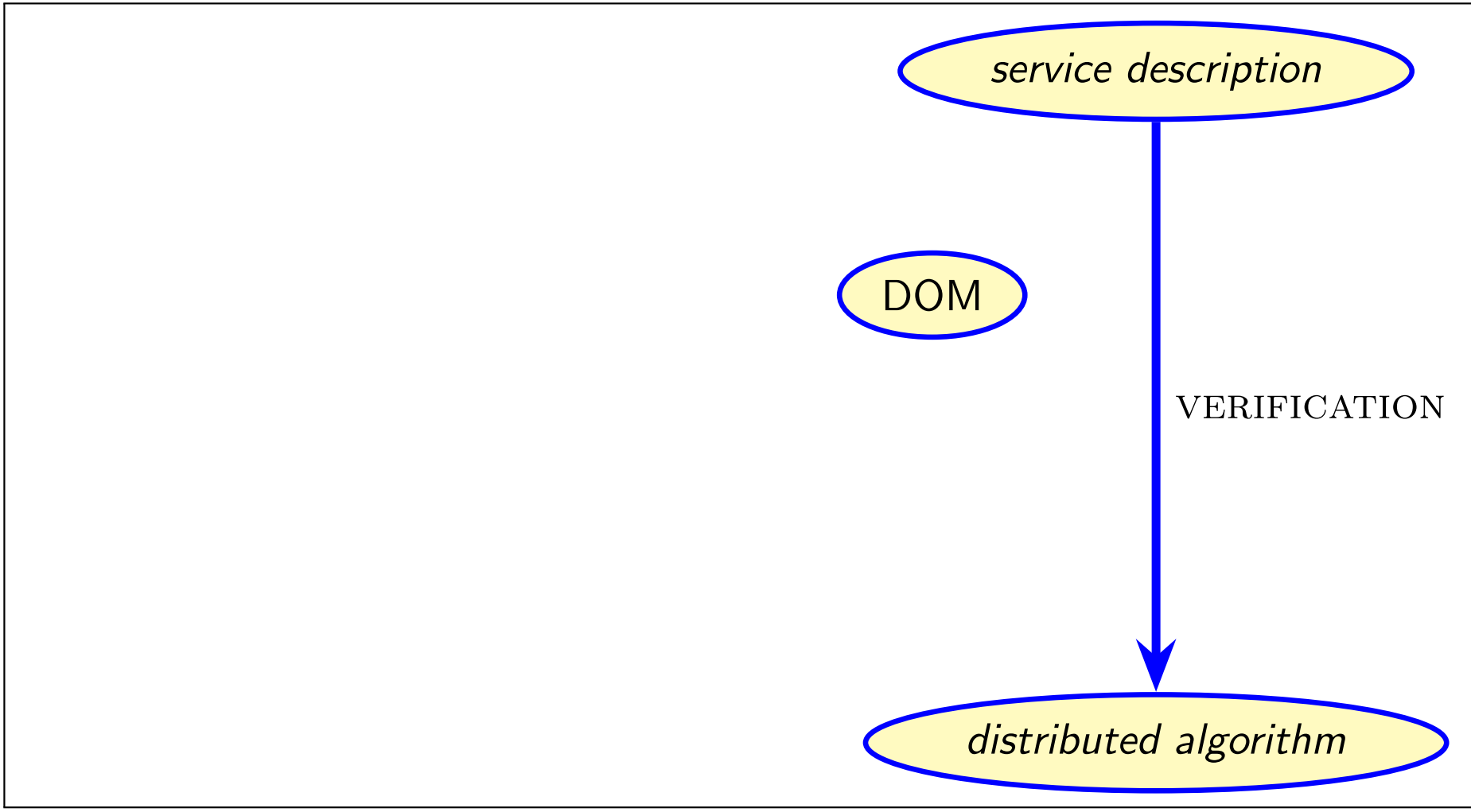
The distributed pattern



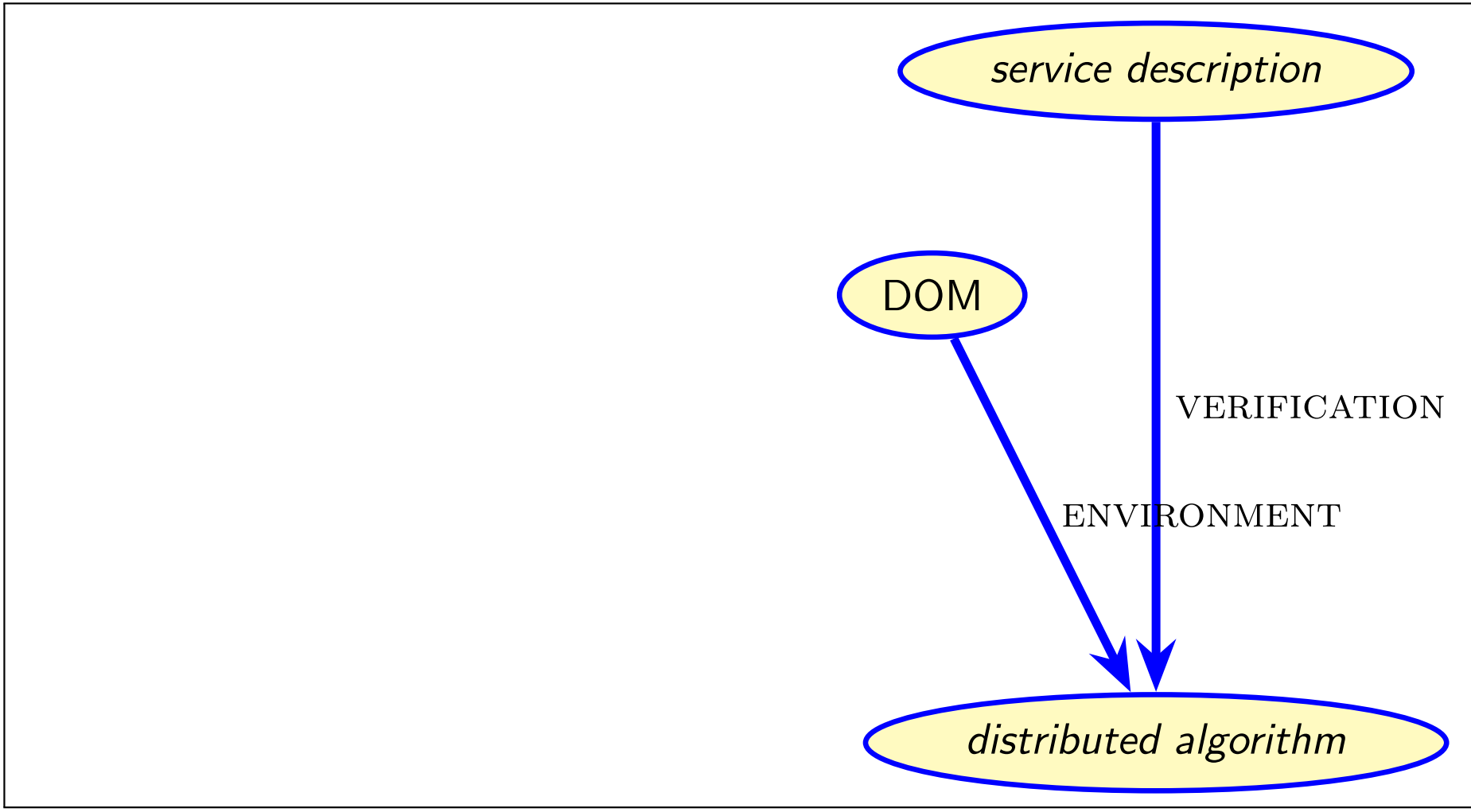
The distributed pattern



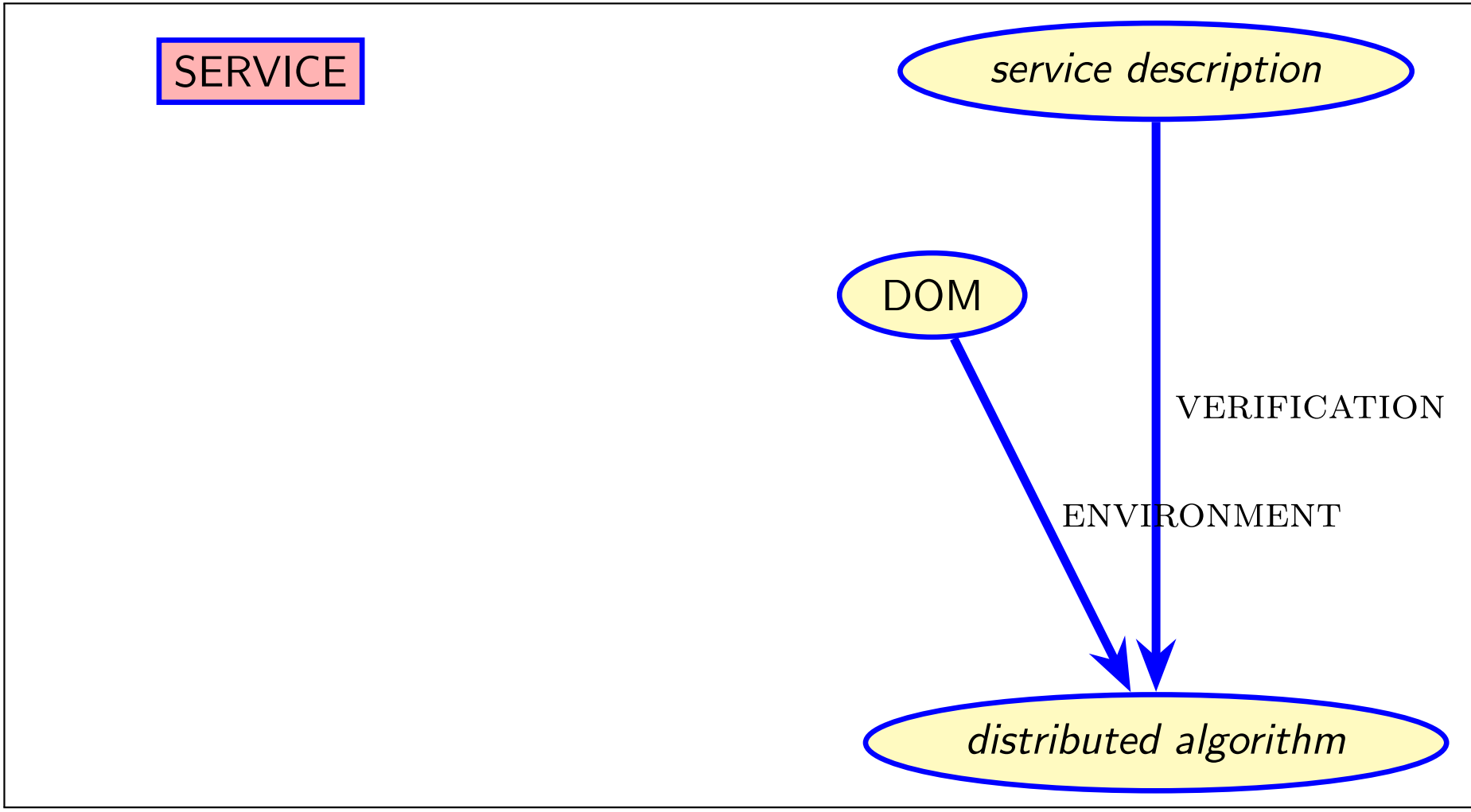
The distributed pattern



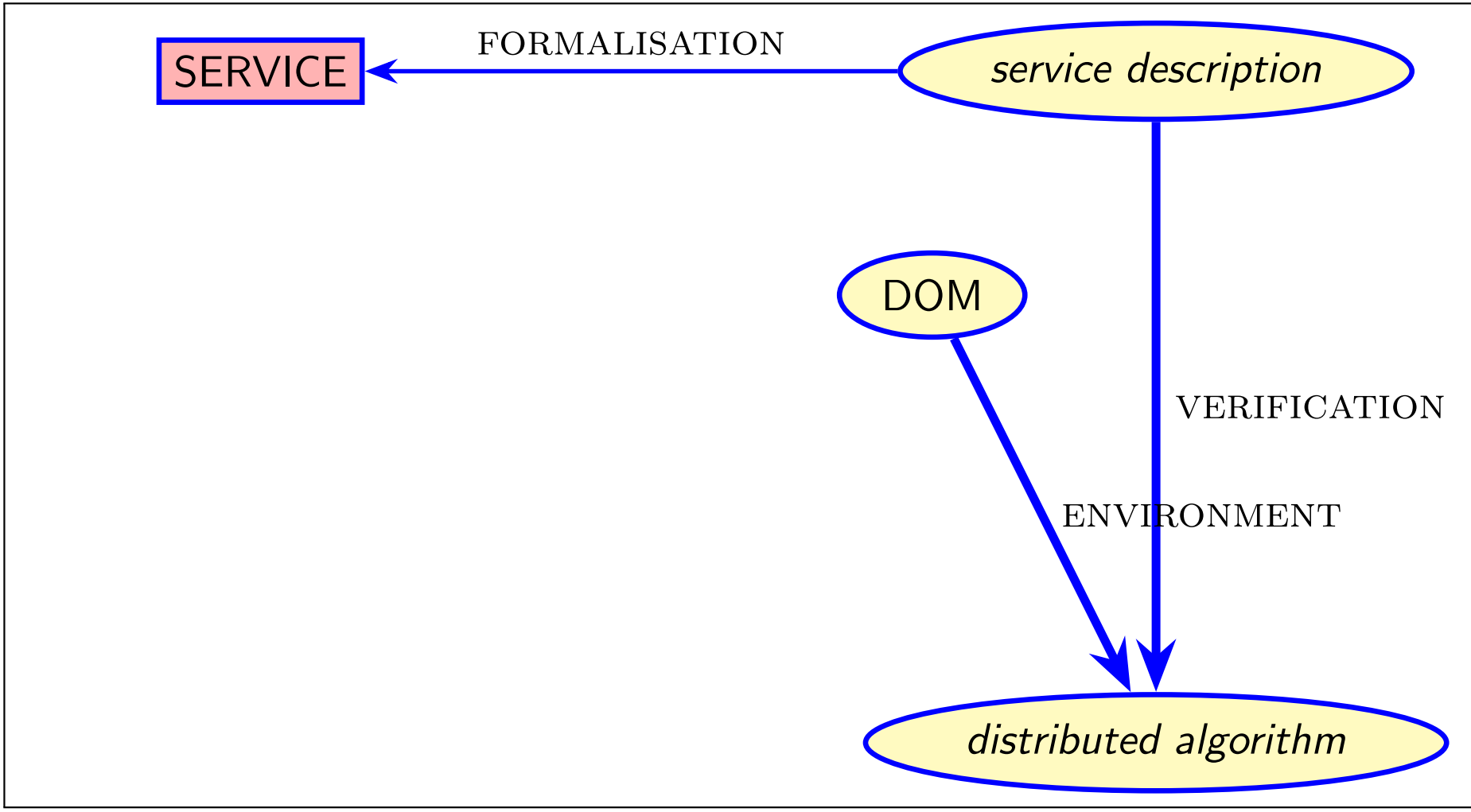
The distributed pattern



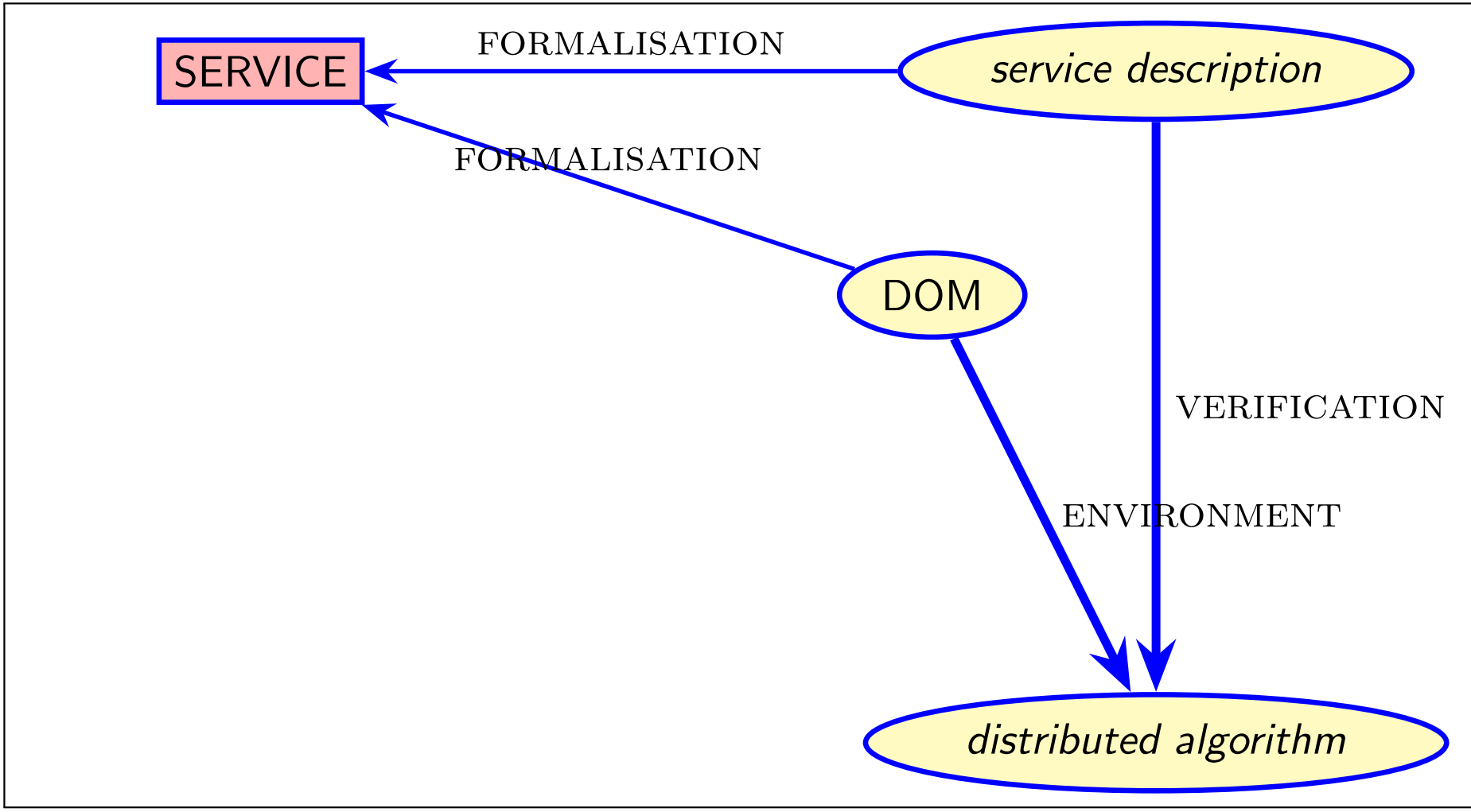
The distributed pattern



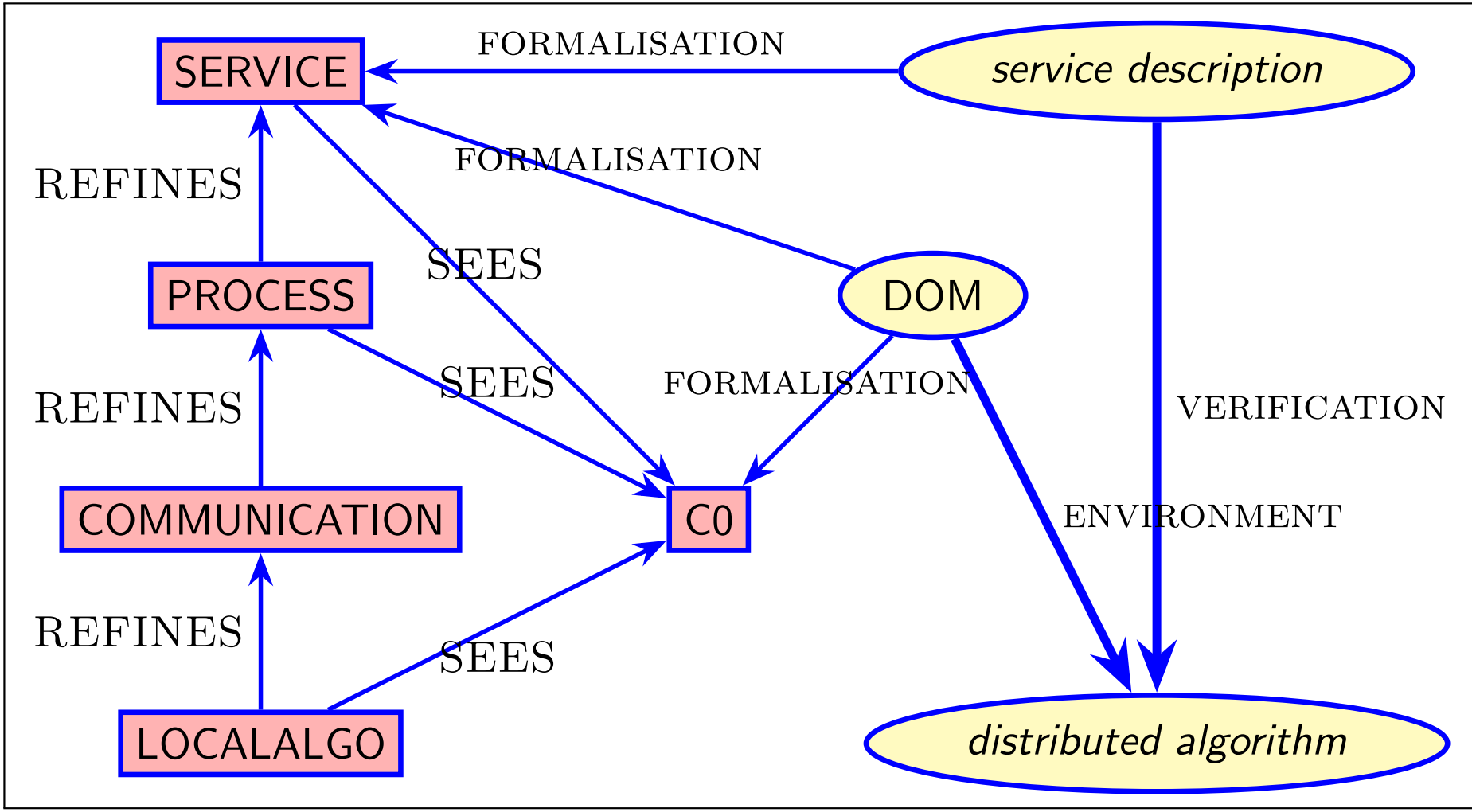
The distributed pattern



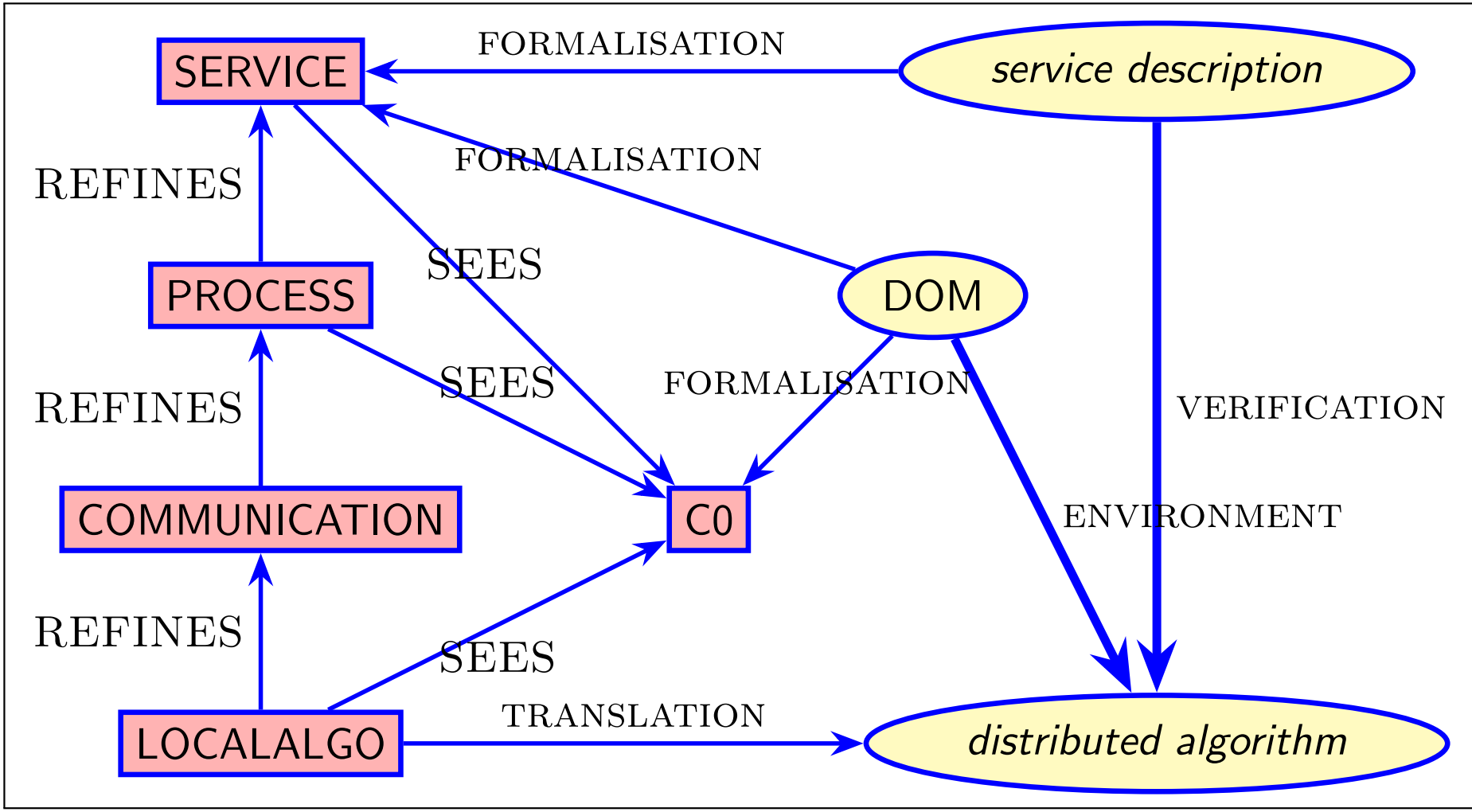
The distributed pattern



The distributed pattern



The distributed pattern



Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

DS1 SEES DS0

EVENT $detection$
where
$@grd1\ ok = N \ \wedge \ detection = FALSE$
then
$@act1\ detection := TRUE$
end

DS1 SEES DS0

EVENT $detection$
where
$@grd1\ ok = N \ \wedge \ detection = FALSE$
then
$@act1\ detection := TRUE$
end

- *detection* observes the termination of the computation.
- *detection* expresses the service.
- Each node is in the set ok
- the nodes of N are passive but r is the last active

DS1 SEES DS0

VARIABLES $ok\ detection$

INVARIANTS

$@inv1\ ok \subseteq N \land detection \in BOOL$

$@inv2\ detection = TRUE \Rightarrow ok = N$

EVENTS

EVENT $INITIALISATION$

then

$@act1\ ok := \varnothing$

$@act2\ detection := FALSE$

end

EVENT $detection$

where

$@grd1\ ok = N \land detection = FALSE$

then

$@act1\ detection := TRUE$

end

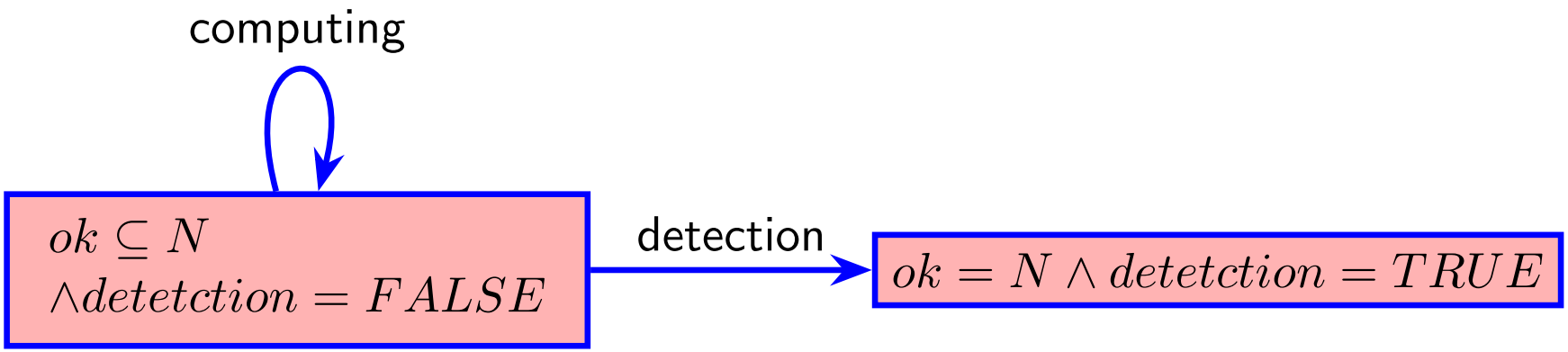
$anticipated$ **EVENT** $computing$

then

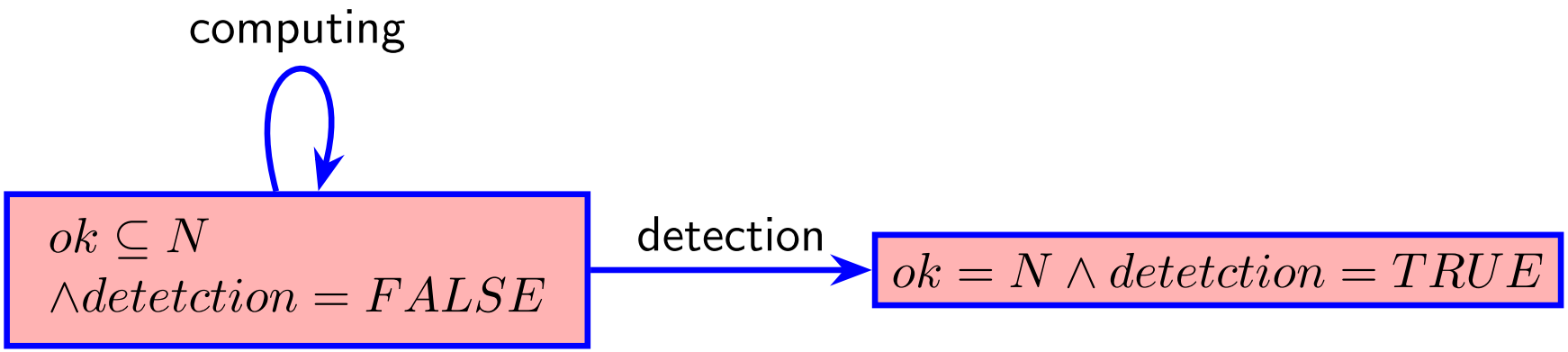
$@act1\ ok, detection :| (ok' \subseteq N \land detection' \in BOOL \land detection' = TRUE \Rightarrow ok' = N)$

end

Abstract transition system labelled by events

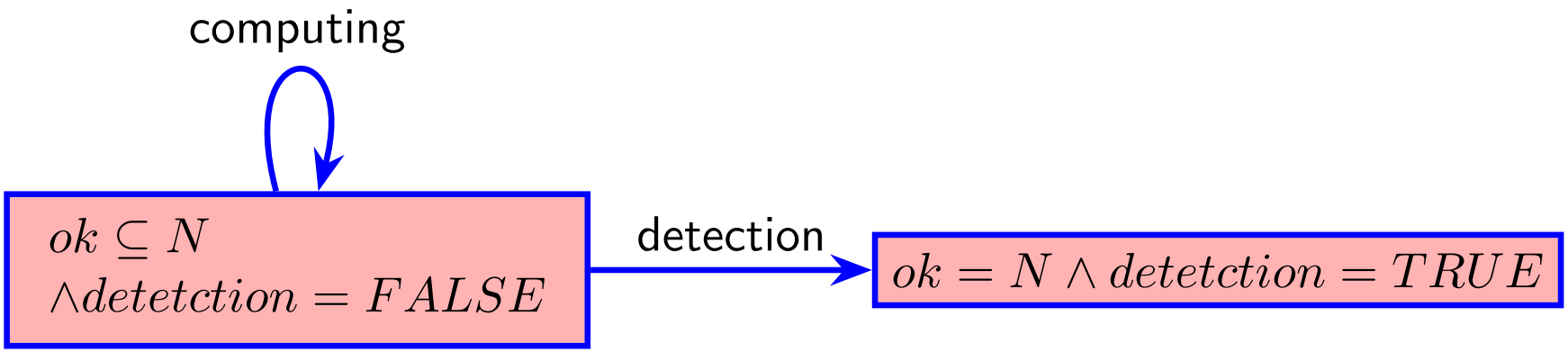


Abstract transition system labelled by events



- *computing* expresses the future protocol to discover;
- Condition for convergence of the protocol with *anticipated* status.

Abstract transition system labelled by events



- *computing* expresses the future protocol to discover;
- Condition for convergence of the protocol with *anticipated* status.

Next step!

How to refine *computing*?
What is the trick?

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Introducing the two waves

- tr, a, rt, utr are new variables for controling the graph and the defined spanning tree.
- a contains active nodes

$$@inv1\ a \subseteq N$$
$$@inv2\ tr \subseteq G \land tr^{-1} \in N \nrightarrow N \land r \in a$$
$$@inv3\ a = \{r\} \cup dom(tr) \cup ran(tr)$$
$$@inv4\ \forall u.\ u \subseteq a \land r \in u \land tr[u] \subseteq u \Rightarrow a \subseteq u$$
$$@inv6\ rt \subseteq G \land rt \subseteq tr \land ok \subseteq a$$
$$theorem\ @inv12\ \forall x, y.\ x \in a \land y \in a \land x \mapsto y \in tr \land x \in ok \Rightarrow y \in ok$$
$$theorem\ @inv13\ \forall x, y.\ x \in a \land y \in a \land x \mapsto y \in tr \land y \notin ok \Rightarrow x \notin ok$$
$$theorem\ @inv19\ rt^{-1} \in N \nrightarrow N$$
$$@inv22\ rt[ok] \subseteq ok$$
$$@inv18\ \forall x, y.\ x \in ok \land x \mapsto y \in tr \Rightarrow y \in ok$$
$$@inv23\ \forall x, y.\ x \in ok \land x \mapsto y \in closure(rt^{-1}) \Rightarrow y \in ok$$
$$@inv24\ \forall x, y, z.\ x \in ok \land x \mapsto y \in closure(rt^{-1}) \land y \mapsto z \in closure(rt$$
$$@inv25\ \forall x, y.\ x \in ok \land x \mapsto y \in closure(rt^{-1}) \Rightarrow y \in ok$$
$$theorem\ @inv30\ ok = N \Rightarrow G \subseteq tr \cup utr$$
$$@inv34\ \forall u.\ u \subseteq ok \land r \in u \land rt[u] \subseteq u \Rightarrow ok \subseteq u$$

Introducing the two waves

- tr, a, rt, utr are new variables for controling the graph and the defined spanning tree.
- a contains active nodes

$$@inv4\ \forall u.\ u \subseteq a \land r \in u \land tr[u] \subseteq u \Rightarrow a \subseteq u$$
$$theorem\ @inv8\ dom(tr) \subseteq a \land ran(tr) \subseteq a$$
$$theorem\ @inv19\ rt^{-1} \in N \nrightarrow N$$
$$@inv22\ rt[ok] \subseteq ok$$
$$@inv18\ \forall x, y.\ x \in ok \land x \mapsto y \in tr \Rightarrow y \in ok$$
$$@inv23\ \forall x, y.\ x \in ok \land x \mapsto y \in closure(rt^{-1}) \Rightarrow y \in ok$$
$$theorem\ @inv30\ ok = N \Rightarrow G \subseteq tr \cup utr$$

Introducing the two waves

- tr, a, rt, utr are new variables for controling the graph and the defined spanning tree.
- a contains active nodes

$$@inv4\ \forall u.\ u \subseteq a \wedge r \in u \wedge tr[u] \subseteq u \Rightarrow a \subseteq u$$
$$theorem\ @inv8\ dom(tr) \subseteq a \wedge ran(tr) \subseteq a$$
$$theorem\ @inv19\ rt^{-1} \in N \nrightarrow N$$
$$@inv22\ rt[ok] \subseteq ok$$
$$@inv18\ \forall x, y.\ x \in ok \wedge x \mapsto y \in tr \Rightarrow y \in ok$$
$$@inv23\ \forall x, y.\ x \in ok \wedge x \mapsto y \in closure(rt^{-1}) \Rightarrow y \in ok$$
$$theorem\ @inv30\ ok = N \Rightarrow G \subseteq tr \cup utr$$

- a is satisfying an inductive property namely a lfp property.

Introducing the two waves

- tr, a, rt, utr are new variables for controling the graph and the defined spanning tree.
- a contains active nodes

$$@inv4\ \forall u.\ u \subseteq a \wedge r \in u \wedge tr[u] \subseteq u \Rightarrow a \subseteq u$$
$$theorem\ @inv8\ dom(tr) \subseteq a \wedge ran(tr) \subseteq a$$
$$theorem\ @inv19\ rt^{-1} \in N \nrightarrow N$$
$$@inv22\ rt[ok] \subseteq ok$$
$$@inv18\ \forall x, y.\ x \in ok \wedge x \mapsto y \in tr \Rightarrow y \in ok$$
$$@inv23\ \forall x, y.\ x \in ok \wedge x \mapsto y \in closure(rt^{-1}) \Rightarrow y \in ok$$
$$theorem\ @inv30\ ok = N \Rightarrow G \subseteq tr \cup utr$$

- a is satisfying an inductive property namely a lfp property.
- Thanks to the **Instantiation** plugin for providing the indutive definitions as closure ...

Introducing the two waves

- tr, a, rt, utr are new variables for controling the graph and the defined spanning tree.
- a contains active nodes

$$@inv4\ \forall u.\ u \subseteq a \wedge r \in u \wedge tr[u] \subseteq u \Rightarrow a \subseteq u$$
$$theorem\ @inv8\ dom(tr) \subseteq a \wedge ran(tr) \subseteq a$$
$$theorem\ @inv19\ rt^{-1} \in N \nrightarrow N$$
$$@inv22\ rt[ok] \subseteq ok$$
$$@inv18\ \forall x, y.\ x \in ok \wedge x \mapsto y \in tr \Rightarrow y \in ok$$
$$@inv23\ \forall x, y.\ x \in ok \wedge x \mapsto y \in closure(rt^{-1}) \Rightarrow y \in ok$$
$$theorem\ @inv30\ ok = N \Rightarrow G \subseteq tr \cup utr$$

- a is satisfying an inductive property namely a lfp property.
- Thanks to the **Instantiation** plugin for providing the indutive definitions as closure ...
- ... Thanks to the ANR Project EBRP Plus and to Jean-Raymond for the idea of having *reusable* and *parametric* components.

Introducing the two waves

EVENT *detection* REFINES *detection*

where

$@grd1\ ok = N \ \wedge \ detection = FALSE$

$theorem\ @grd2\ \forall x.\ x \in ok \Rightarrow G[\{x\}] \subseteq (tr \cup utr)[\{x\}]$

$theorem\ @grd3\ \forall u.\ u \subseteq N \wedge r \in u \wedge rt[u] \subseteq u \Rightarrow N \subseteq u$

then

$@act1\ detection := TRUE$

end

EVENT *up* REFINES *computing*

any $x\ y$

where

$@grd1\ x \in a \wedge y \notin a \wedge x \mapsto y \in G$

$@grd2\ x \mapsto y \notin tr$

$@grd3\ y \mapsto x \notin tr$

$@grd4\ x \mapsto y \notin utr$

$@grd5\ y \mapsto x \notin utr$

$theorem\ @grd6\ x \notin ok$

then

$@act1\ tr := tr \cup \{x \mapsto y\}$

$@act2\ a := a \cup \{y\}$

end

Introducing the two waves

EVENT *up−*

EVENT *downone* . REFINES *computing*

EVENT *down* . REFINES *computing*

- up- means that a request for achild did not succeed
- down events are refining computing and are refined later.

Introducing the two waves

EVENT *detection* REFINES *detection*

where

$@grd1\ ok = N \ \wedge \ detection = FALSE$

$theorem\ @grd2\ \forall x.\ x \in ok \Rightarrow G[\{x\}] \subseteq (tr \cup utr)[\{x\}]$

$theorem\ @grd3\ \forall u.\ u \subseteq N \wedge r \in u \wedge rt[u] \subseteq u \Rightarrow N \subseteq u$

then

$@act1\ detection := TRUE$

end

EVENT *up* REFINES *computing*

any $x\ y$

where

$@grd1\ x \in a \wedge y \notin a \wedge x \mapsto y \in G$

$@grd2\ x \mapsto y \notin tr$

$@grd3\ y \mapsto x \notin tr$

$@grd4\ x \mapsto y \notin utr$

$@grd5\ y \mapsto x \notin utr$

$theorem\ @grd6\ x \notin ok$

then

$@act1\ tr := tr \cup \{x \mapsto y\}$

$@act2\ a := a \cup \{y\}$

end

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Adding communication variables

- Adding a variable called msg which is containing the messages sent by a node resuesting to anotherode to be his client and to be active.
- New events are added for managing the new variable.
- Communications are added progressively from machine DS3 till DS8.
- DS8 is a machine with variables and properties that are preparing the localization.

Last machine DS8

$ok\ detection\ tr\ a\ rt\ utr\ wmsg\ loc\ gsm\ col$
$ploc\ pgsm\ nomsg\ pnomsg\ ack\ wack\ pack\ chi\ wchi$
$parent\ noack\ confloc\ nb\ ch\ noch\ chan$
$inport$ // *input of messageswhich are first received and then a procees*
$ackchan\ nackchan\ par\ act\ confchan\ okchan$

- The machine DS8 has variables modelling the state of the channels as ackchan and variables as wack standing for witness variables related to the ack messages.
- The choice of our modelling is to keep explicit model or ghost variables as long as necessary . . .
- . . . to get an invariant strong enough for localizing the events.

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Leaving ghost variables in previous machines

- Remaining visible variables *ok,detection,tr,a, rt, utr, wmsg, gsm, col, wack, wchi, noack, confloc, nb, ch , noch, chan, inport, ackchan, nackchan, par, act, confchan, okchan*
- events are named using an explicit role as local computation, sending a message of receiving a message.
- Communications are assumed reliable.
-

$$\left\{ \begin{array}{l} theorem\ @grd2\ \forall\, x.\ x \in ok \Rightarrow G[\{x\}] \subseteq (tr \cup utr)[\{x\}] \\ theorem\ @grd3\ \forall\, u.\ u \subseteq N \wedge r \in u \wedge rt[u] \subseteq u \Rightarrow N \subseteq u \end{array} \right.$$

EVENT *detection* REFINES *detection*
where
$@grd1\ ok = N \wedge detection = FALSE$
then
$@act1\ detection := TRUE$

Leaving ghost variables in previous machines

- Remaining visible variables *ok,detection,tr,a, rt, utr, wmsg, gsm, col, wack, wchi, noack, confloc, nb, ch , noch, chan, inport, ackchan, nackchan, par, act, confchan, okchan*
- events are named using an explicit role as local computation, sending a message of receiving a message.
- Communications are assumed reliable.
-

$$\left\{\begin{array}{l} theorem\ @grd2\ \forall\, x.\ x \in ok \Rightarrow G[\{x\}] \subseteq (tr \cup utr)[\{x\}] \\ theorem\ @grd3\ \forall\, u.\ u \subseteq N \wedge r \in u \wedge rt[u] \subseteq u \Rightarrow N \subseteq u \end{array}\right.$$

EVENT $sending(x, MSG, y)(x \mapsto y \notin msg)$
local event@x-to-y REFINES $sending(x, MSG, y)(x \mapsto y \notin msg)$
any $x\ y$ where
$@grd1\ x \in N \wedge y \in N \wedge y \in nb(x)$
$@grd2\ x \in a \wedge y \notin wmsg(x)$
$@grd3\ x \mapsto y \mapsto MSG \notin chan$
then
$@act1\ chan := chan \cup \{x \mapsto y \mapsto MSG\}$
$@act2\ wmsg(x) := wmsg(x) \cup \{y\}$

Leaving ghost variables in previous machines

- Remaining visible variables *ok,detection,tr,a, rt, utr, wmsg, gsm, col, wack, wchi, noack, confloc, nb, ch , noch, chan, inport, ackchan, nackchan, par, act, confchan, okchan*
- events are named using an explicit role as local computation, sending a message of receiving a message.
- Communications are assumed reliable.
-

$$\left\{\begin{array}{l} theorem\ @grd2\ \forall\, x.\ x \in ok \Rightarrow G[\{x\}] \subseteq (tr \cup utr)[\{x\}] \\ theorem\ @grd3\ \forall\, u.\ u \subseteq N \wedge r \in u \wedge rt[u] \subseteq u \Rightarrow N \subseteq u \end{array}\right.$$

EVENT $receiving(y, MSG, x)$ // $localevent@y$
REFINES $receiving(y, MSG, x)$
any $x\ y$
where
$@grd1\ x \mapsto y \mapsto MSG \in chan$
then
$@act1\ chan := chan \setminus \{x \mapsto y \mapsto MSG\}$
$@act2\ inport := inport \cup \{x \mapsto y \mapsto MSG\}$
end

Leaving ghost variables in previous machines

- Remaining visible variables *ok,detection,tr,a, rt, utr, wmsg, gsm, col, wack, wchi, noack, confloc, nb, ch , noch, chan, inport, ackchan, nackchan, par, act, confchan, okchan*
- events are named using an explicit role as local computation, sending a message of receiving a message.
- Communications are assumed reliable.
-

$$\left\{\begin{array}{l} theorem\ @grd2\ \forall x.\ x \in ok \Rightarrow G[\{x\}] \subseteq (tr \cup utr)[\{x\}] \\ theorem\ @grd3\ \forall u.\ u \subseteq N \wedge r \in u \wedge rt[u] \subseteq u \Rightarrow N \subseteq u \end{array}\right.$$

EVENT $sending(y, ACK, x)$ // $localevent@ytox$

any $x\ y$ where

$@grd1\ x \mapsto y \mapsto MSG \in inport$

$@grd3\ y \neq r$

$@grd9\ y \notin dom(wack)$

$@grd5\ y \notin a$

then

$@act1\ wack := wack \cup \{y \mapsto x\}$

$@act2\ inport := inport \setminus \{x \mapsto y \mapsto MSG\}$

$@act3\ ackchan := ackchan \cup \{x \mapsto y \mapsto ACK\}$

The full machine DS9

EVENT $detection$
EVENT $sending(x, MSG, y)(x \mapsto y \notin msg!)$ // $localevent@xy$
EVENT $receiving(y, MSG, x)$ // $localevent@y$
EVENT $sending(y, ACK, x)$ // $localevent@yxtoyx$
EVENT $sending(y, NACK, x)$ // $localevent@yxtoyx$
EVENT $receiving(x, NACK, y)$ // $localevent@x$
EVENT $updating(x, noch, y)$ // $localevent@x$
EVENT $receiving(x, ack, y)$ // $localevent@x$
EVENT $sending(x, CONF, y)$ // $localevent@x$
EVENT $receiving(y, CONF, x)$ // $localevent@y$
EVENT $sending(i, OK)_to_par(i)_on_okchan$ // $localevent@i$
EVENT $receiving(x, OK, y)$ // $localevent@x$
EVENT $(x)getting_ok_from_children$ // $localevcent@x$

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Statistics

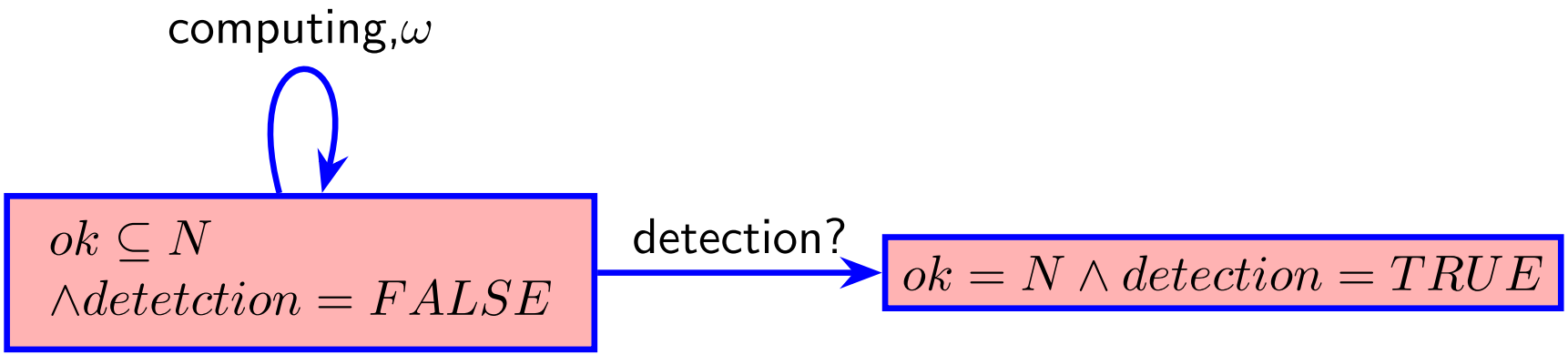
Component	Total	Auto	Manual	% Manual
ADS1	4	4	0	0
ADS2	83	68	15	18 %
ADS3	39	35	4	10%
ADS4	150	121	29	19 %
ADS5	80	59	21	26 %
ADS6	32	23	9	28 %
ADS7	31	21	10	32 %
ADS8	54	35	19	35 %
ADS9	20	20	0	0

The Distributed Pattern

- The better way to explain the algorithm DS is to say: *It is a game with a spanning tree to build and then to delete nodes from the so called leaves*.
- Possibility to superpose local computations for each node when active.
- Applying our fusion technique using merging of events for local events and then getting the derived local algorithm.
- Thanks to the Instantiation plugin for providing the fixed-point theorems.
- But the Kumar algorithm is not clearly in this family of algorithms

The Distributed Pattern

However, the Kumar algorithm does not terminate . . .



Non trivial invariant! (see paper by Méry and Merz in the volume Gedenkschrift Jean-Raymond Abrial).

Current Summary

1. Overview
2. Distributed Algorithms
3. Short Summary on Event-B
4. The Service-as-Event Paradigm
5. Protocols of communication
6. The Self-Healing P2P based Protocol
7. The termination detection problem
   - Overview
   - Service for DS (DS1)
   - Process for DS (DS2)
   - Communication for DS (DS(3..8)))
   - Localization for DS
   - Final comments
8. Conclusion

Experiences- Teaching and modelling distributed algorithms

Families of distributed algorithms

1. Communicate: communication protocols
2. Coordinate: timing and scheduling algorithms
3. Share: mutual exclusion algorithms
4. Organise: leader election algorithms
5. Control: snapshot and termination detection algorithms
6. Stabilise: self-stabilisation algorithms

Experiences- Teaching and modelling distributed algorithms

Families of distributed algorithms

1. Communicate: communication protocols
2. Coordinate: timing and scheduling algorithms
3. Share: mutual exclusion algorithms
4. Organise: leader election algorithms
5. Control: snapshot and termination detection algorithms
6. Stabilise: self-stabilisation algorithms

- Each distributed algorithm was explained through applications of induction and refinement rules.
- Making explicit each refinement step helps to explain why the algorithm works.
- We developed and taught courses on distributed algorithms by combining Event-B and TLA$^+$.

Reviews - Lessons learnt from Event-B developments

Representative case studies

- IEEE1394 leader election protocol: convergence from a forest to a tree.
- Ricart–Agrawala mutual exclusion algorithm: requests are organised through abstract timestamp queues.
- Self-* algorithms: decomposition into convergent refinement steps.
- Security and distributed coordination protocols.

Reviews - Lessons learnt from Event-B developments

Representative case studies

- IEEE1394 leader election protocol: convergence from a forest to a tree.
- Ricart–Agrawala mutual exclusion algorithm: requests are organised through abstract timestamp queues.
- Self-* algorithms: decomposition into convergent refinement steps.
- Security and distributed coordination protocols.

Methodological contributions

- Paradigms for planning refinements:
  - Service-as-Event paradigm
  - Composition/Decomposition paradigm
- Introduction of graphical refinement diagrams.
- Pedagogical support for explaining distributed behaviours: the parachutist process.

Prospects - Future directions

Prospects - Future directions

- Transformation of Event-B models into executable distributed programs.

Prospects - Future directions

- Transformation of Event-B models into executable distributed programs.
- Modelling and verification with probabilistic features.

Prospects - Future directions

- Transformation of Event-B models into executable distributed programs.
- Modelling and verification with probabilistic features.
- Construction of an atlas of *correct-by-construction* distributed algorithms.

- Thanks to my partners of projects DESIRS, RIMEL, IMPEX, EBRP Plus, TAPAS, Mosel/Veridis for their support and help.

- Thanks to my partners of projects DESIRS, RIMEL, IMPEX, EBRP Plus, TAPAS, Mosel/Veridis for their support and help.
- Special thank to Jean-Raymond Abrial

- Thanks to my partners of projects DESIRS, RIMEL, IMPEX, EBRP Plus, TAPAS, Mosel/Veridis for their support and help.
- Special thank to Jean-Raymond Abrial
- see my webpage https://mery54.github.io/mery/

- Thanks to my partners of projects DESIRS, RIMEL, IMPEX, EBRP Plus, TAPAS, Mosel/Veridis for their support and help.
- Special thank to Jean-Raymond Abrial
- see my webpage https://mery54.github.io/mery/
- Questions ?