A Simple Model of Communication APIs – Application to Dynamic Partial-order Reduction

Cristian Rosa Stephan Merz Martin Quinson

AVOCS 2010
22/09/2010

Motivation

- Distributed Algorithms are hard to get right:
  - lack of a shared clock
  - lack of a global view of the state
- Errors are hard to find and reproduce
- State-exploration techniques
  - exhaustive
  - counter-examples
  - reproducible

Model-checking Distributed Systems

We define a Distributed System (DS) as:

- processes running distributed across several networked hosts
- no shared clock
- no shared memory
- communication by message exchange

Model-checking Distributed Systems

We define a Distributed System (DS) as:

- processes running distributed across several networked hosts
- no shared clock
- no shared memory
- communication by message exchange

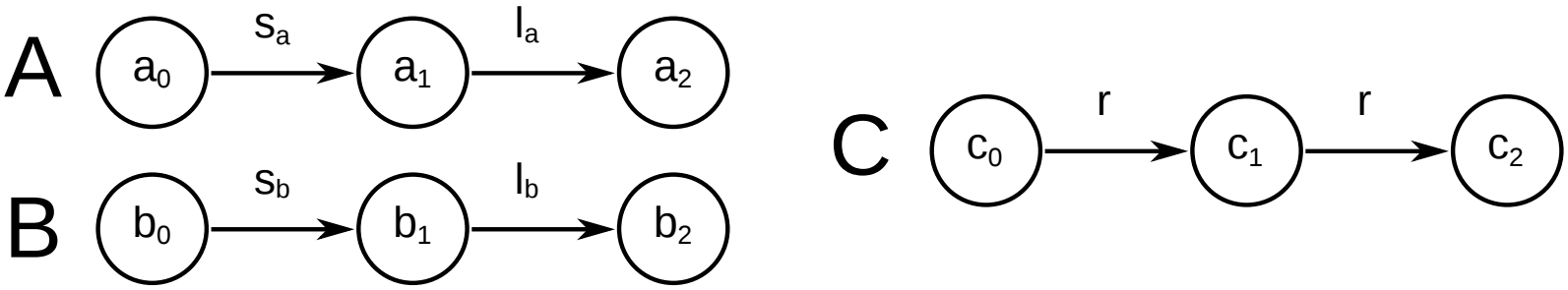
We verify the real C implementation of the programs:

- no models, no abstraction, the processes are really executed
- from a given initial state
- state space generated by the interleaving of the messages
- exploration is depth bounded (useful for debugging anyway)

It its mandatory to apply partial-order techniques!

Happened-before Relation

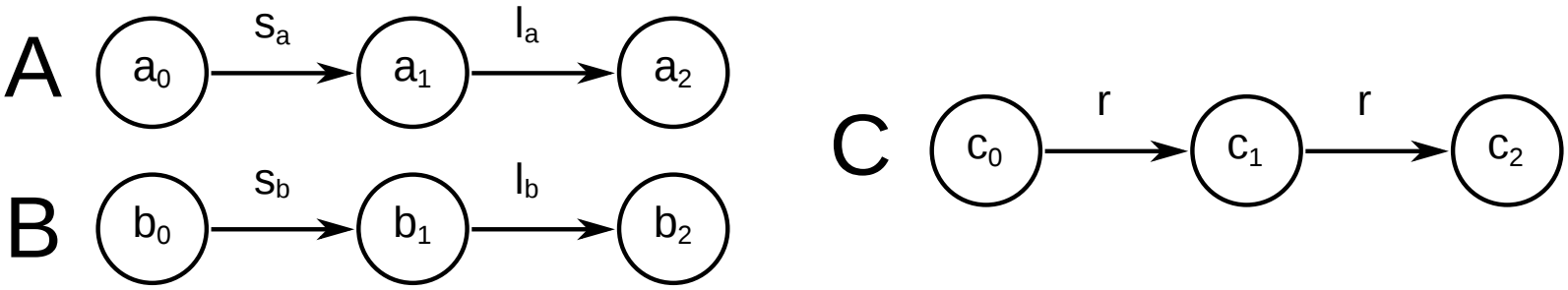
Consider the following processes:



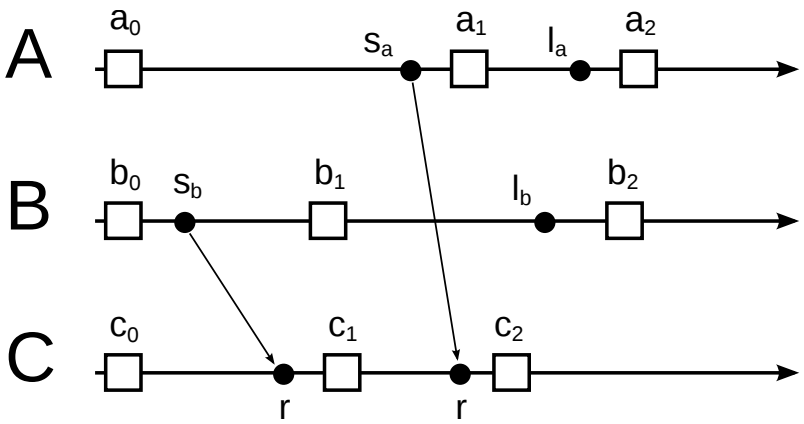
Execution traces:

Happened-before Relation

Consider the following processes:

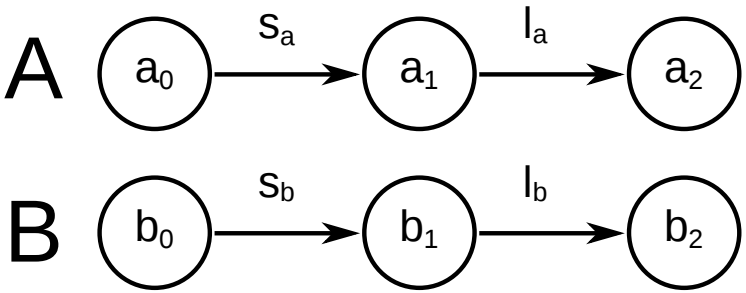


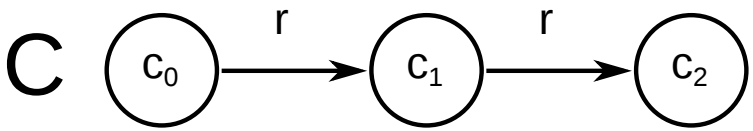
Execution traces:



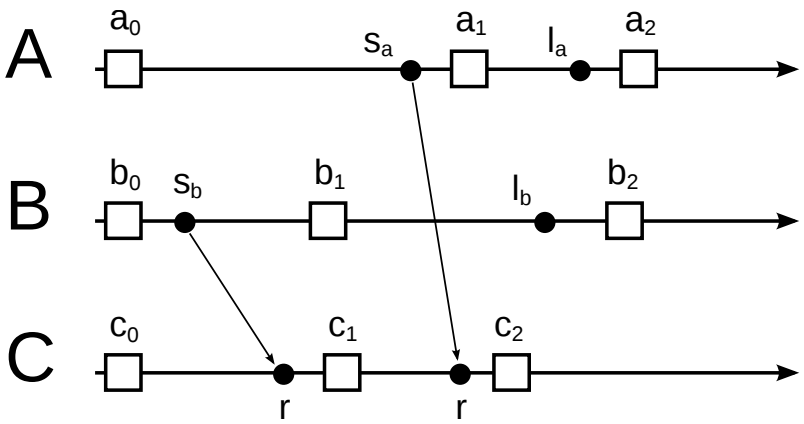
Happened-before Relation

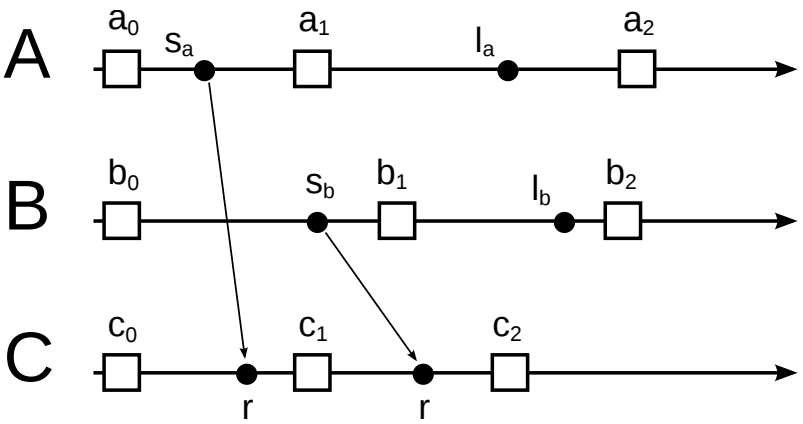
Consider the following processes:



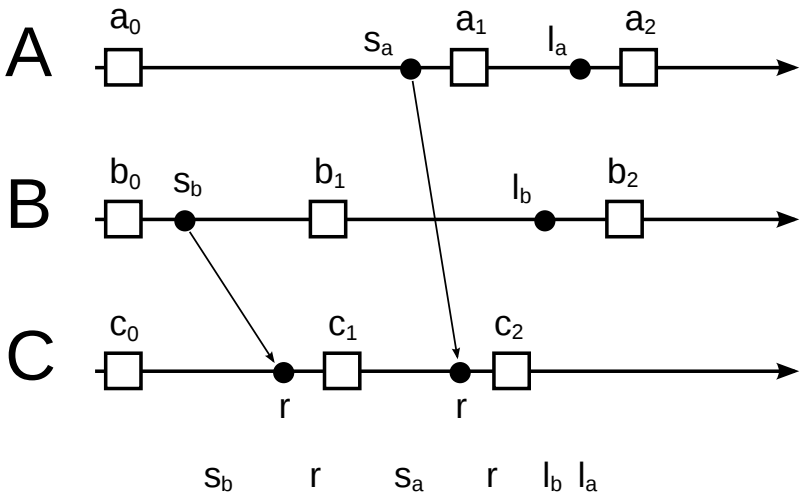


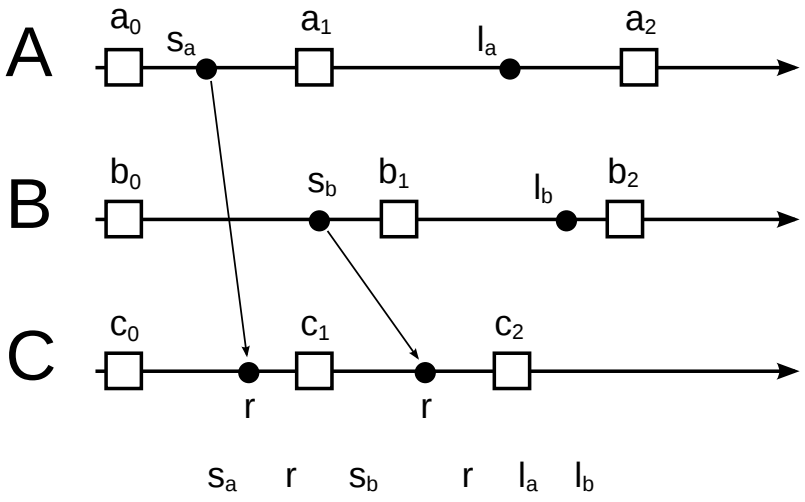
Execution traces:



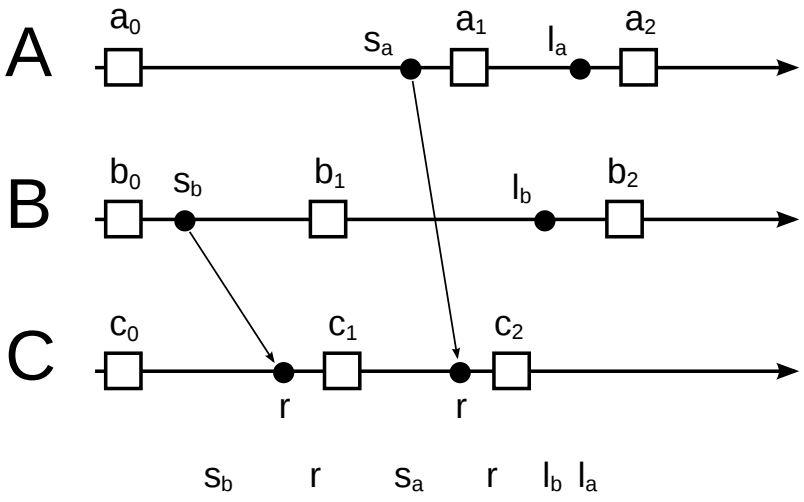


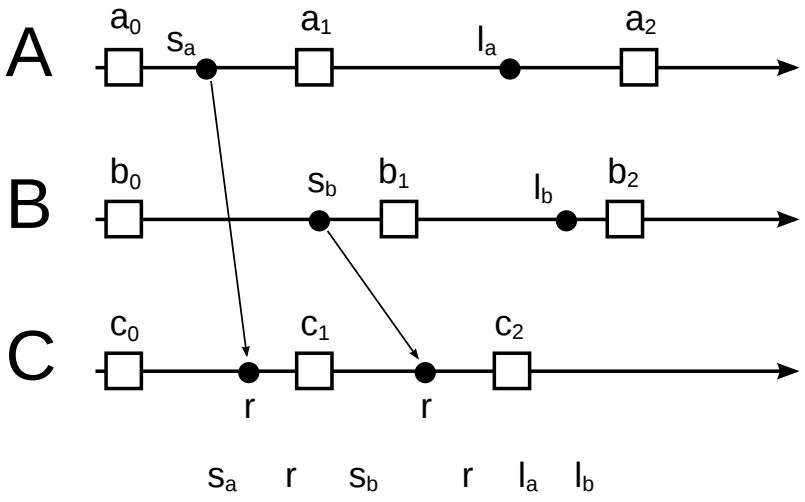
Depth-first Search Exploration



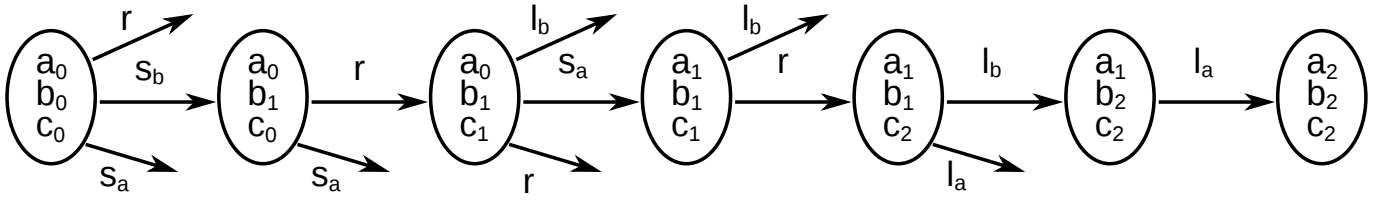


Depth-first Search Exploration

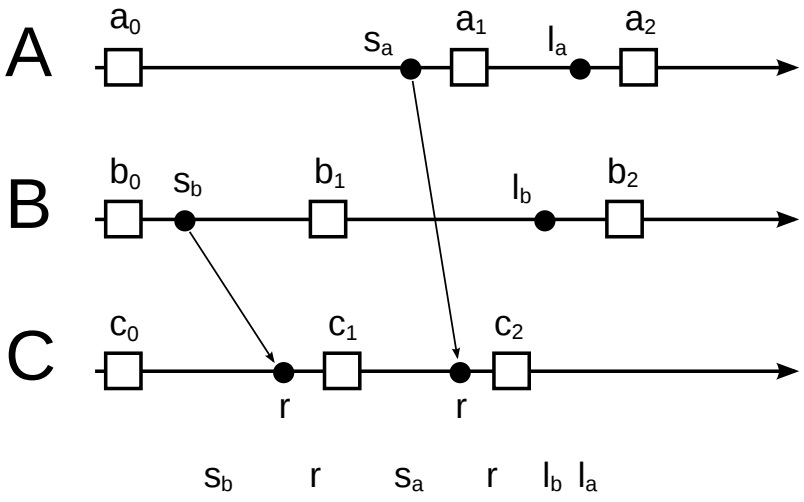


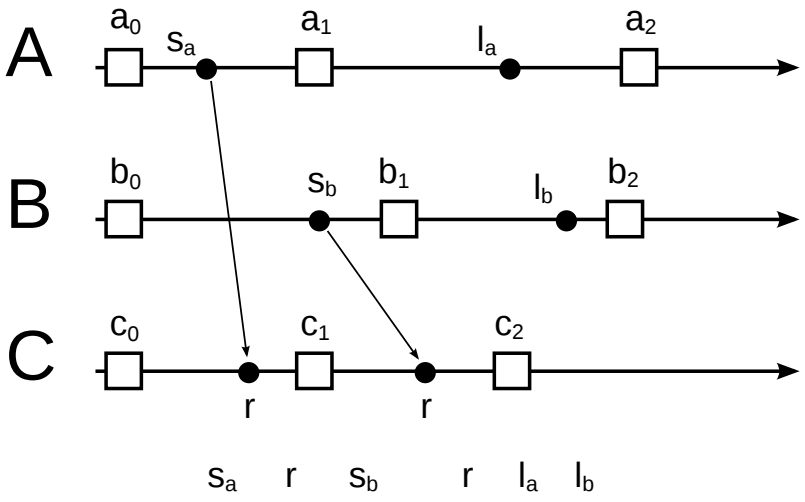


Depth-first search state exploration algorithm:

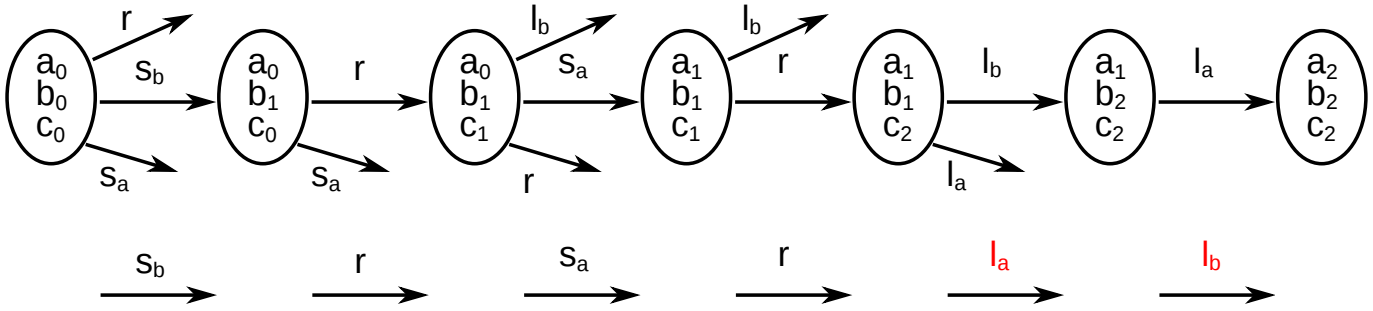


Depth-first Search Exploration





Depth-first search state exploration algorithm:



It's another serialization of the same partial-order!

Dynamic Partial-order Reductions by Example

What are the transitions that we should interleave?

or equivalently ...

How do we generate a serialization of a different partial-order?

Dynamic Partial-order Reductions by Example

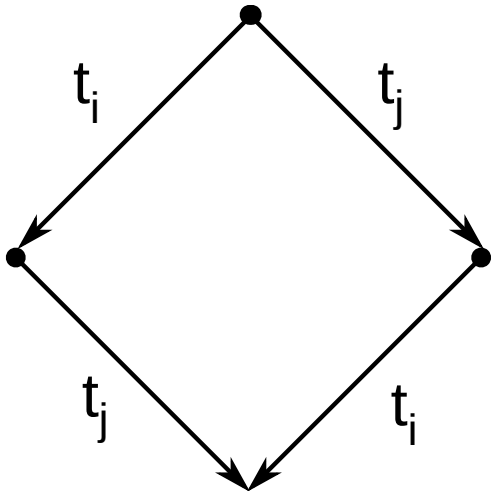
What are the transitions that we should interleave?

or equivalently ...

How do we generate a serialization of a different partial-order?

Interleaving dependent transitions!

$$D(t_i, t_j) = \neg I(t_i, t_j)$$



Computing D Efficiently

How do we get the predicate D?

- Using the semantics of the transitions
- Proof of "*independence theorems*" for each pair of transitions
- The I predicate is the disjunction of these cases

$$I(t_i, t_j) = (t_i = A \wedge t_j = B) \vee \ldots$$

Computing D Efficiently

How do we get the predicate D?

- Using the semantics of the transitions
- Proof of "*independence theorems*" for each pair of transitions
- The I predicate is the disjunction of these cases

$$I(t_i, t_j) = (t_i = A \wedge t_j = B) \vee \ldots$$

- It can be overaproximated by a D' such that

$$D(A, B) \Rightarrow D'(A, B)$$

but the converse might not be true

$$D'(A, B) \not\Rightarrow D(A, B)$$

If we don't know if $I(t_i, t_j)$ we assume $D'(t_i, t_j)$ (for soundness).

API Functions as Transitions

- The transitions are the calls to the communication API
- The problem: almost no API has a formal specification of their semantics
- Manual specification is required based on:
  - informal API references
  - experiments
  - user experience
- It is a tedious and time consuming job (i.e. +100 pages for a subset of MPI [PGK+07])
- It should be done for each API

Contributions

The contributions of this work are:

- A core set of five networking primitives
- Formal specification in $\mathrm{TLA}^+$of it's semantics
- Theorems of independence between certain primitives
- Implementation of the networking core inside of SimGrid simulator [CLQ08]
- A DPOR exploration algorithm using the SimGrid simulation framework

The Communication Model

The communication model is based on mailboxes:

- processes post send/receive request into mailboxes
- requests are queued and matched in FIFO order

The Communication Model

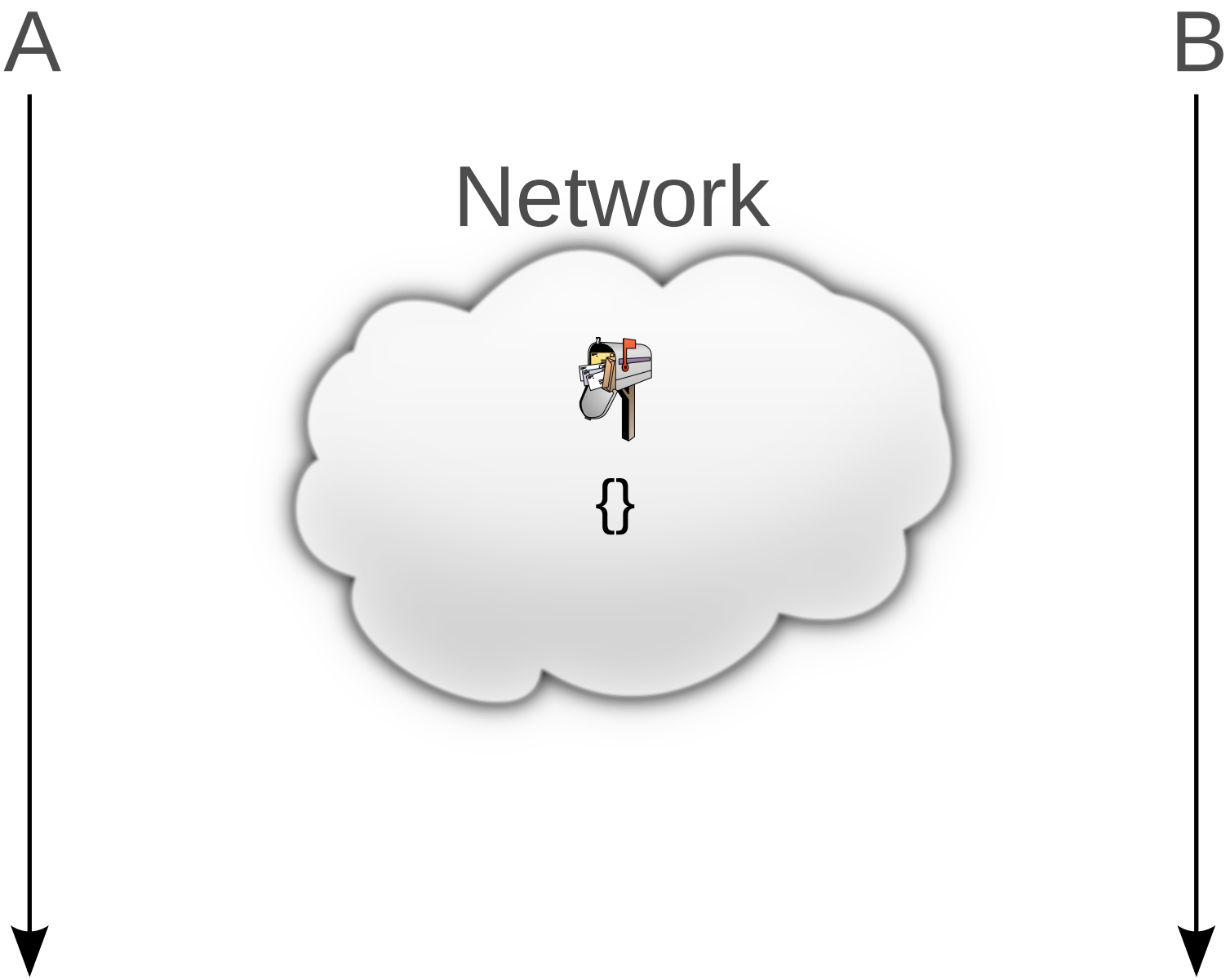
The communication model is based on mailboxes:

- processes post send/receive request into mailboxes
- requests are queued and matched in FIFO order

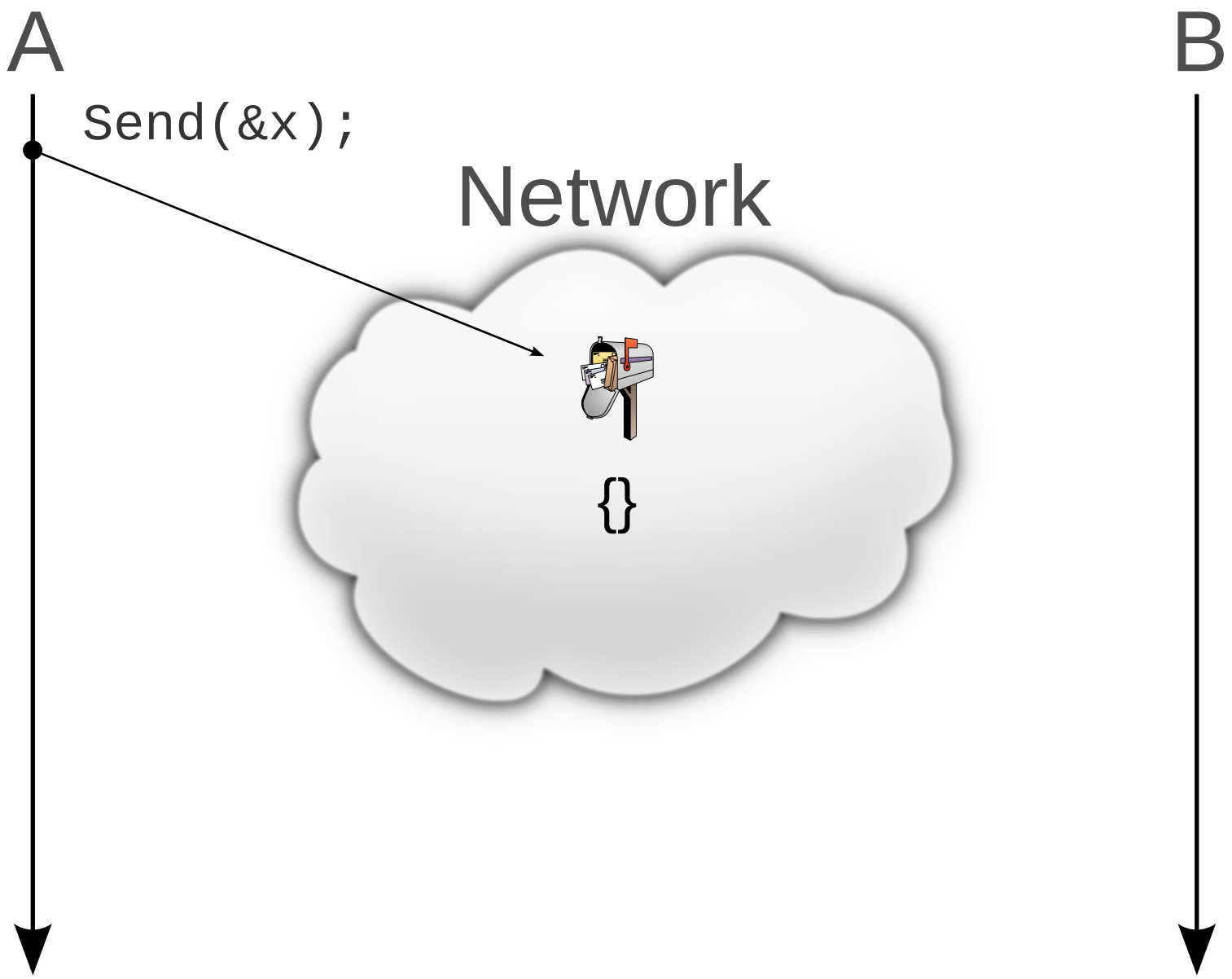
There are four primitives:

- `Send` – asynchronous send
- `Recv` – asynchronous receive
- `WaitAny` – block until completion of a communication
- `Test` – test for completion without blocking

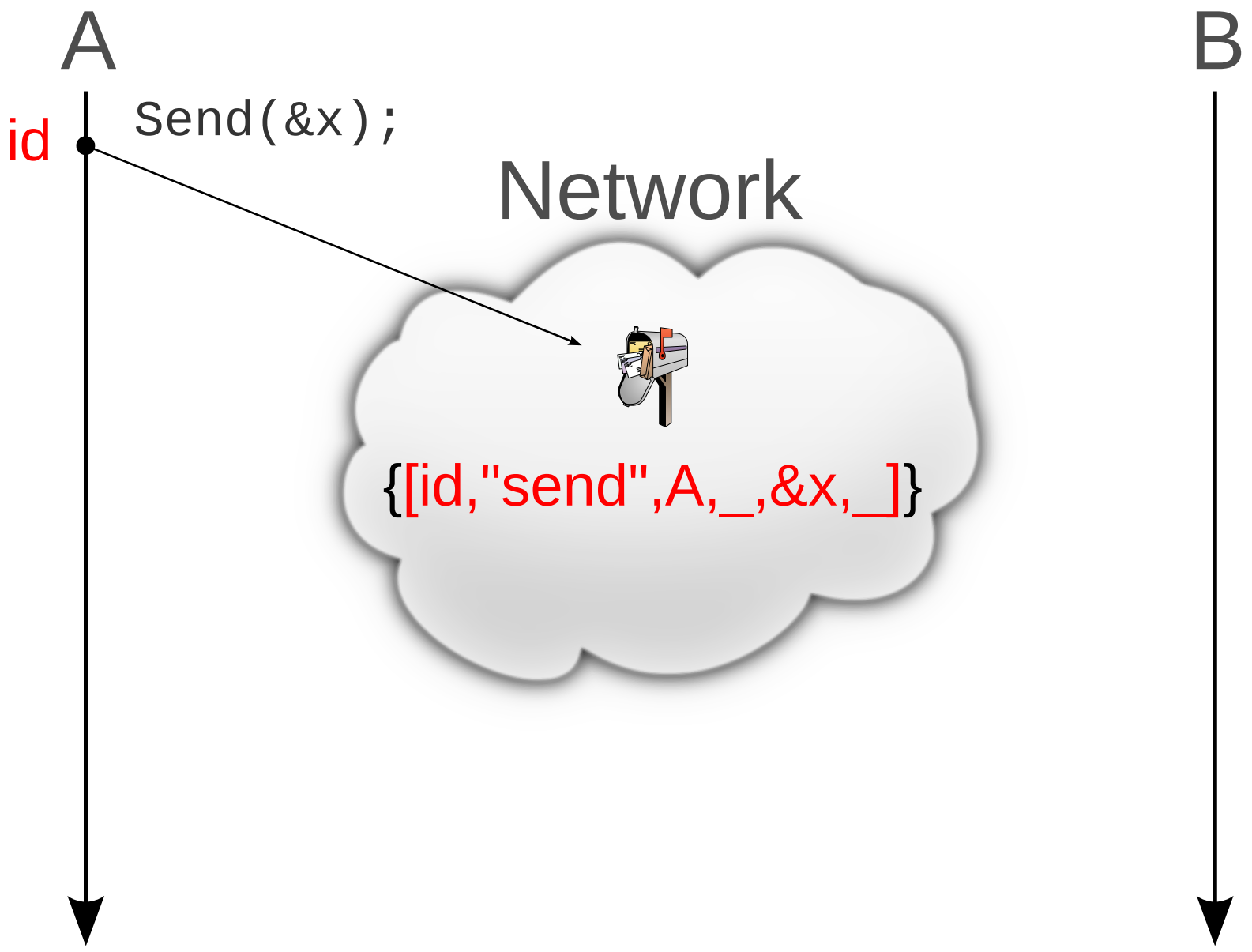
The Communication Model



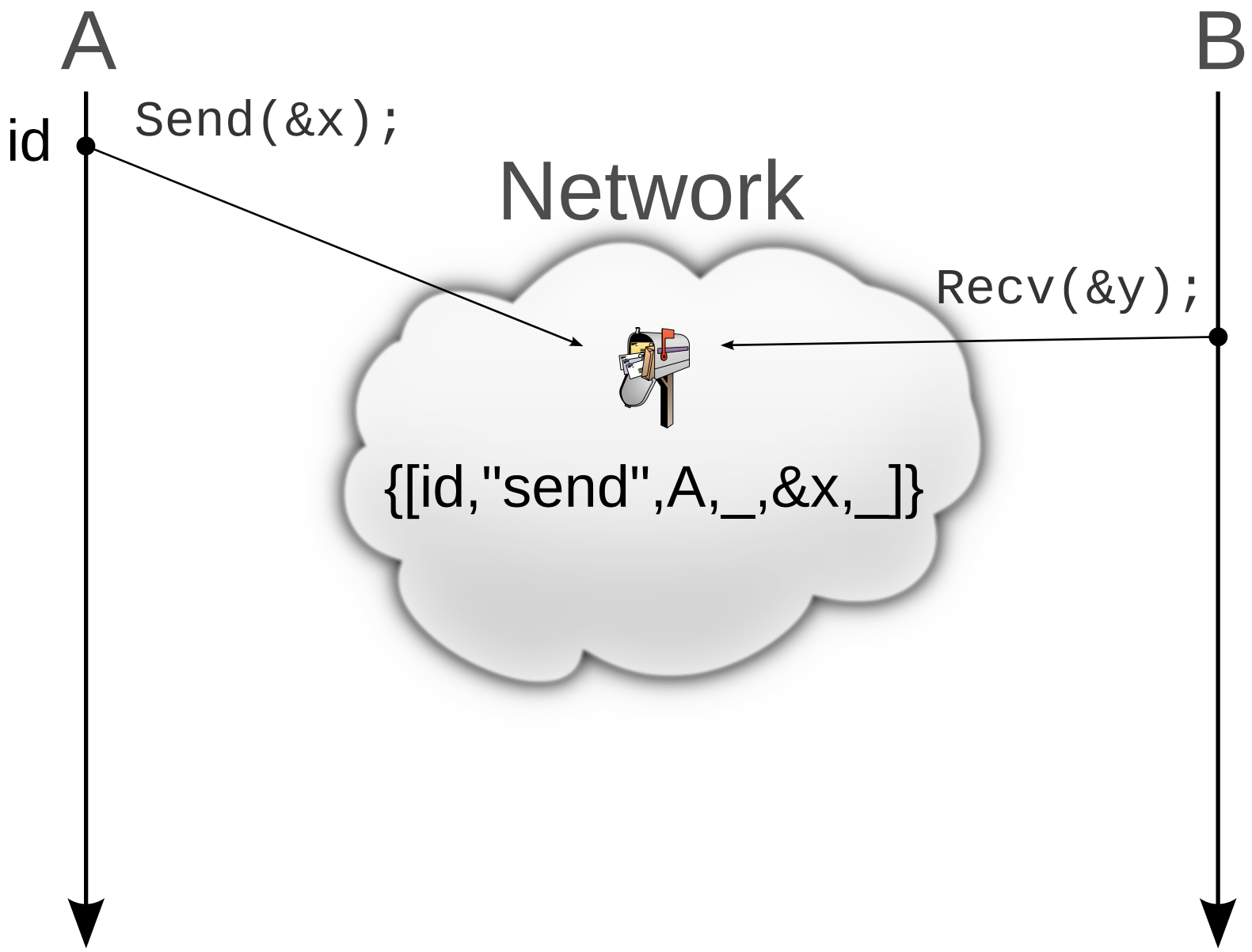
The Communication Model



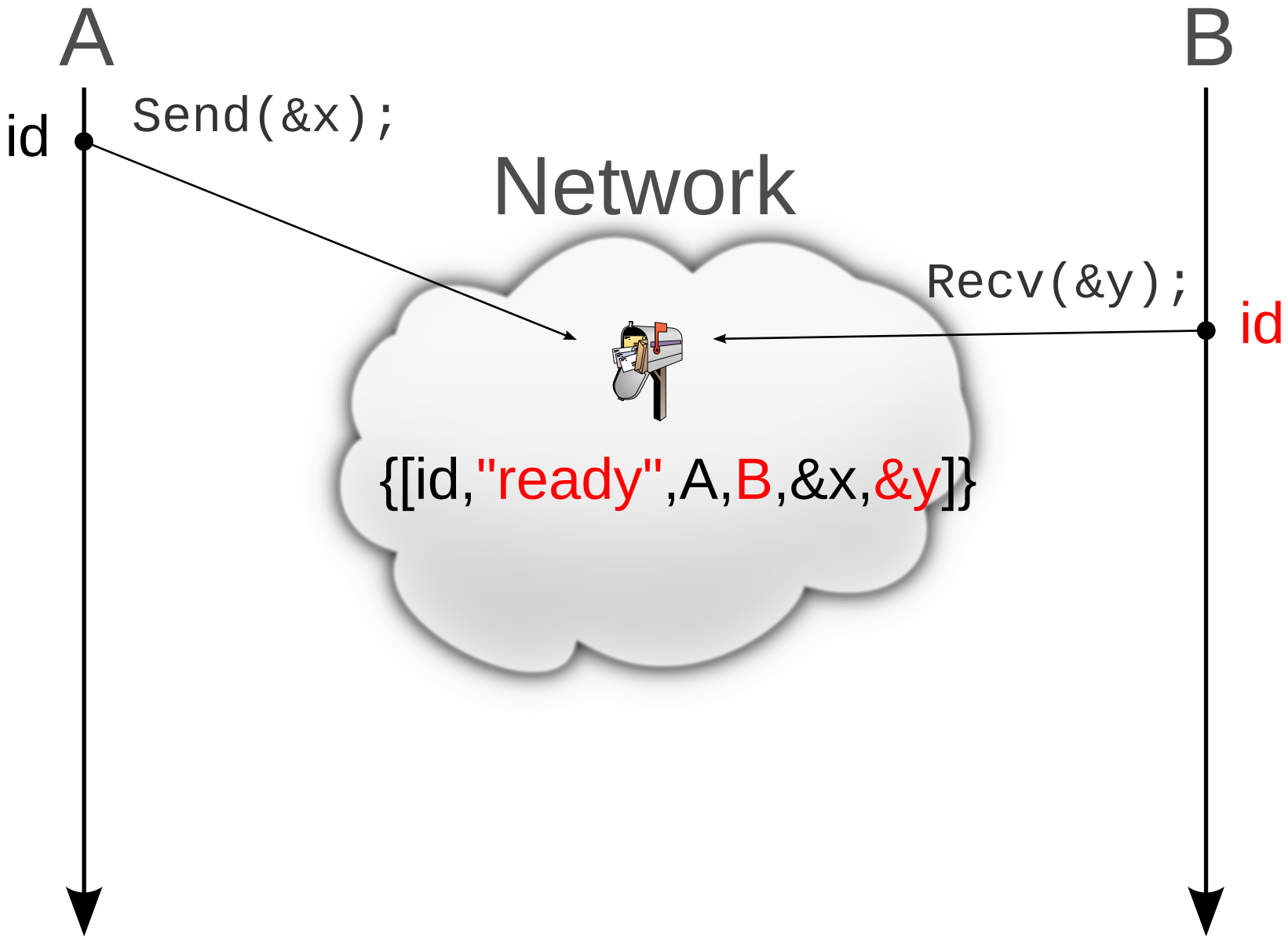
The Communication Model



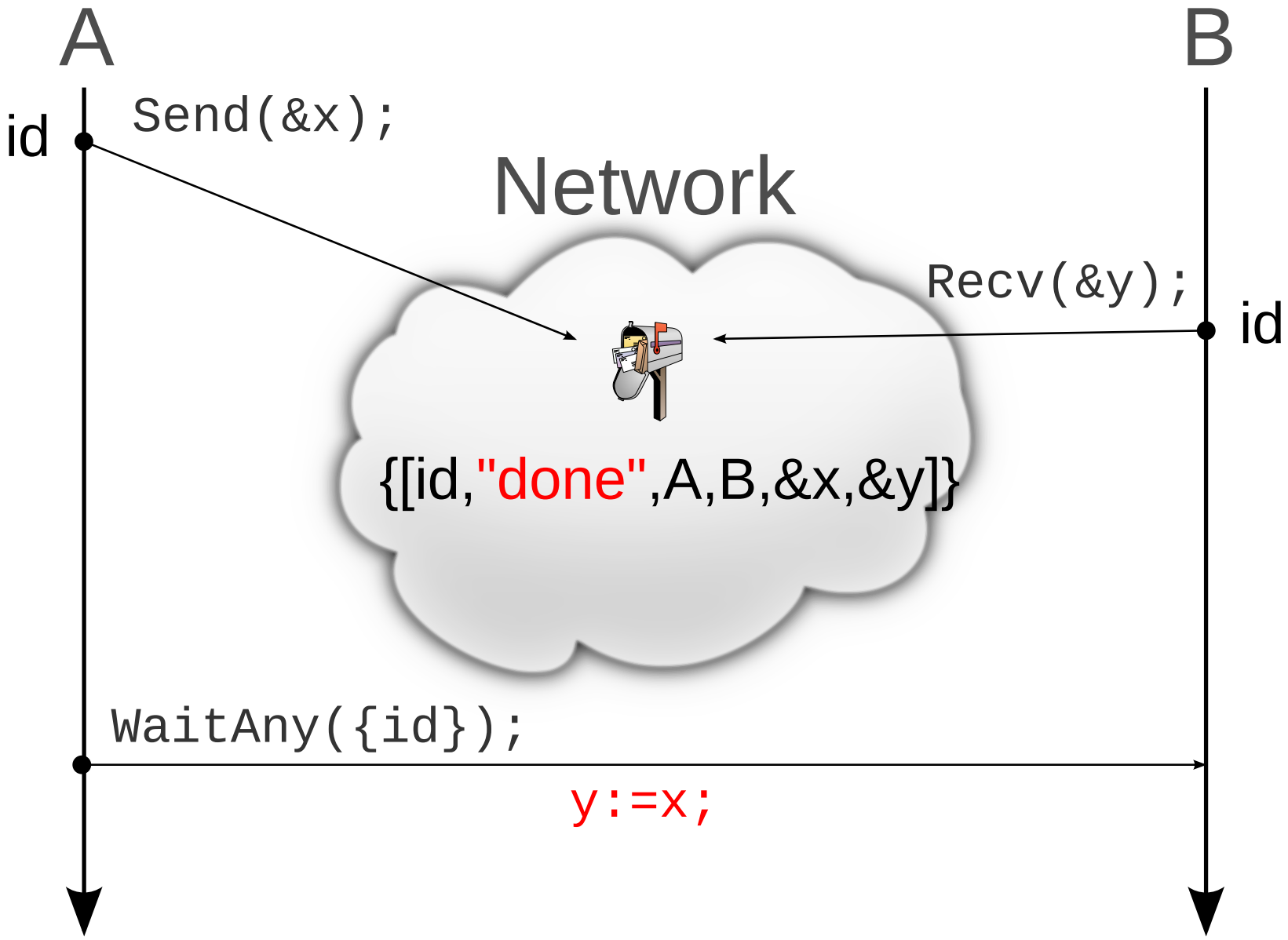
The Communication Model



The Communication Model



The Communication Model

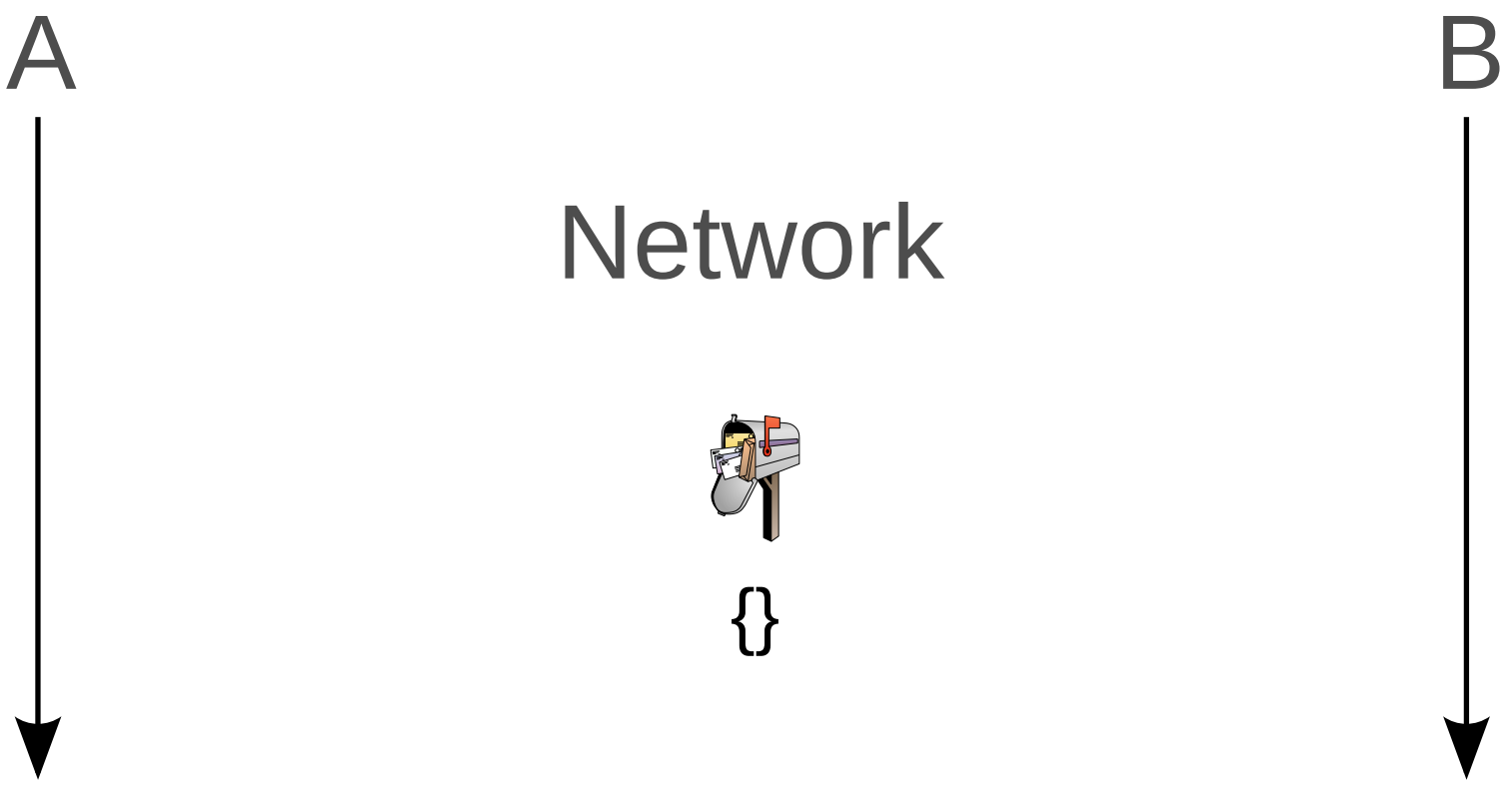


(In)Dependence Theorems

Theorem

Any two Send and Recv transitions are independent.

$$\forall p_1, p_2 \in Proc, rdv_1, rdv_2 \in RdV, d_1, d_2 \in Addr, c_1, c_2 \in Addr : \\ I(Send(p_1, rdv_1, d_1, c_1), Recv(p_2, rdv_2, d_2, c_2))$$

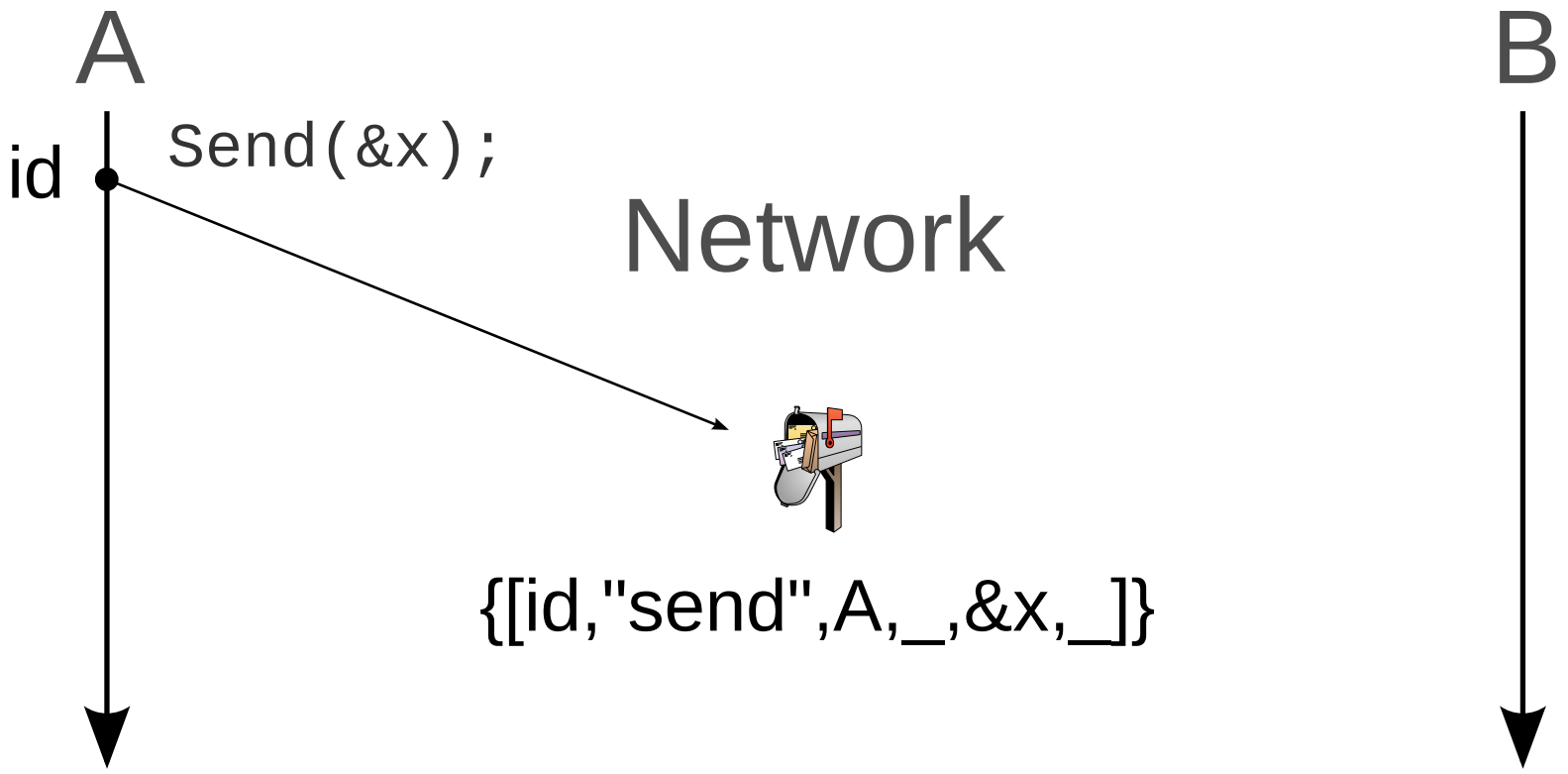


(In)Dependence Theorems

Theorem

Any two Send and Recv transitions are independent.

$$\forall p_1, p_2 \in Proc, rdv_1, rdv_2 \in RdV, d_1, d_2 \in Addr, c_1, c_2 \in Addr :$$
$$I(Send(p_1, rdv_1, d_1, c_1), Recv(p_2, rdv_2, d_2, c_2))$$

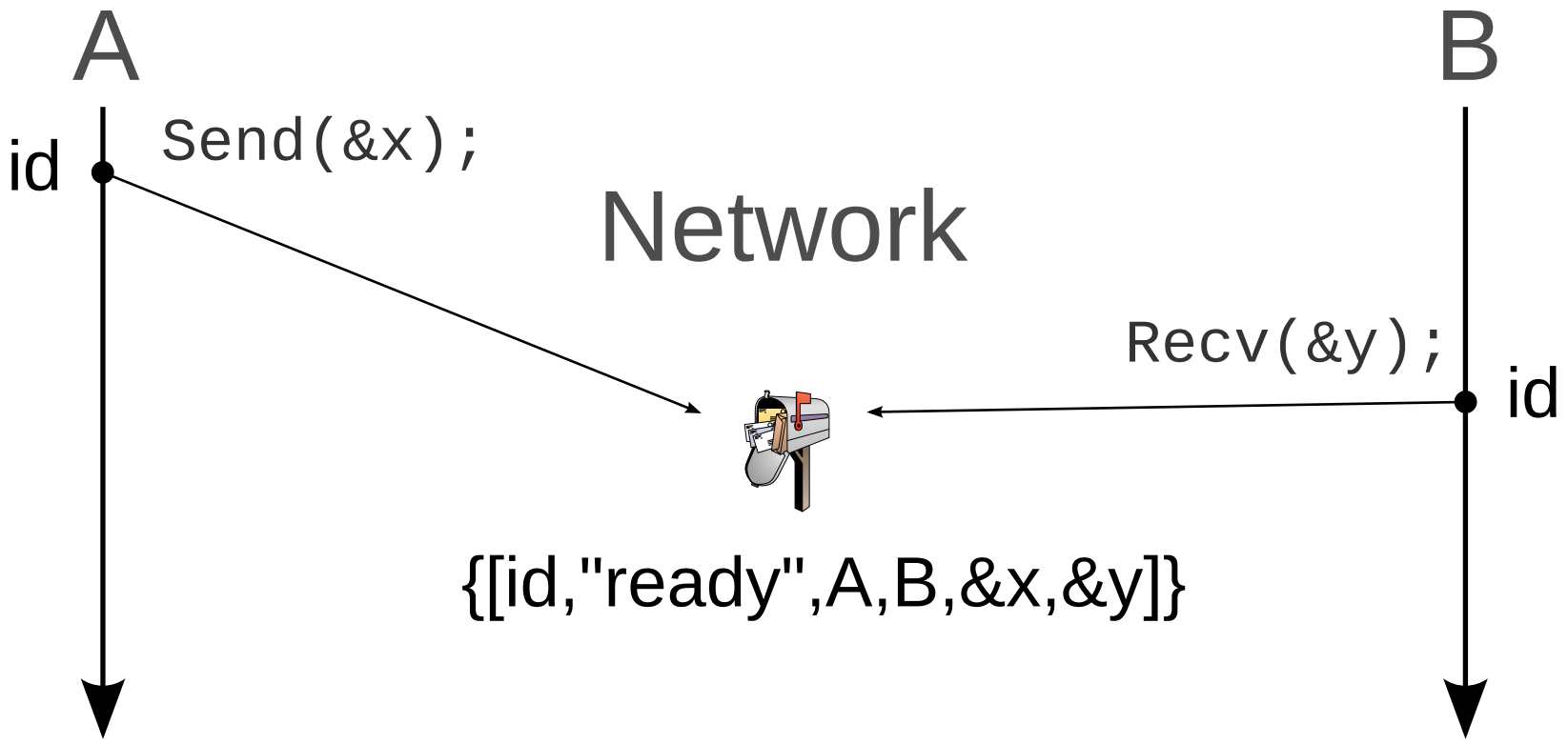


(In)Dependence Theorems

Theorem

Any two Send and Recv transitions are independent.

$$\forall p_1, p_2 \in Proc, rdv_1, rdv_2 \in RdV, d_1, d_2 \in Addr, c_1, c_2 \in Addr : \\ I(Send(p_1, rdv_1, d_1, c_1), Recv(p_2, rdv_2, d_2, c_2))$$

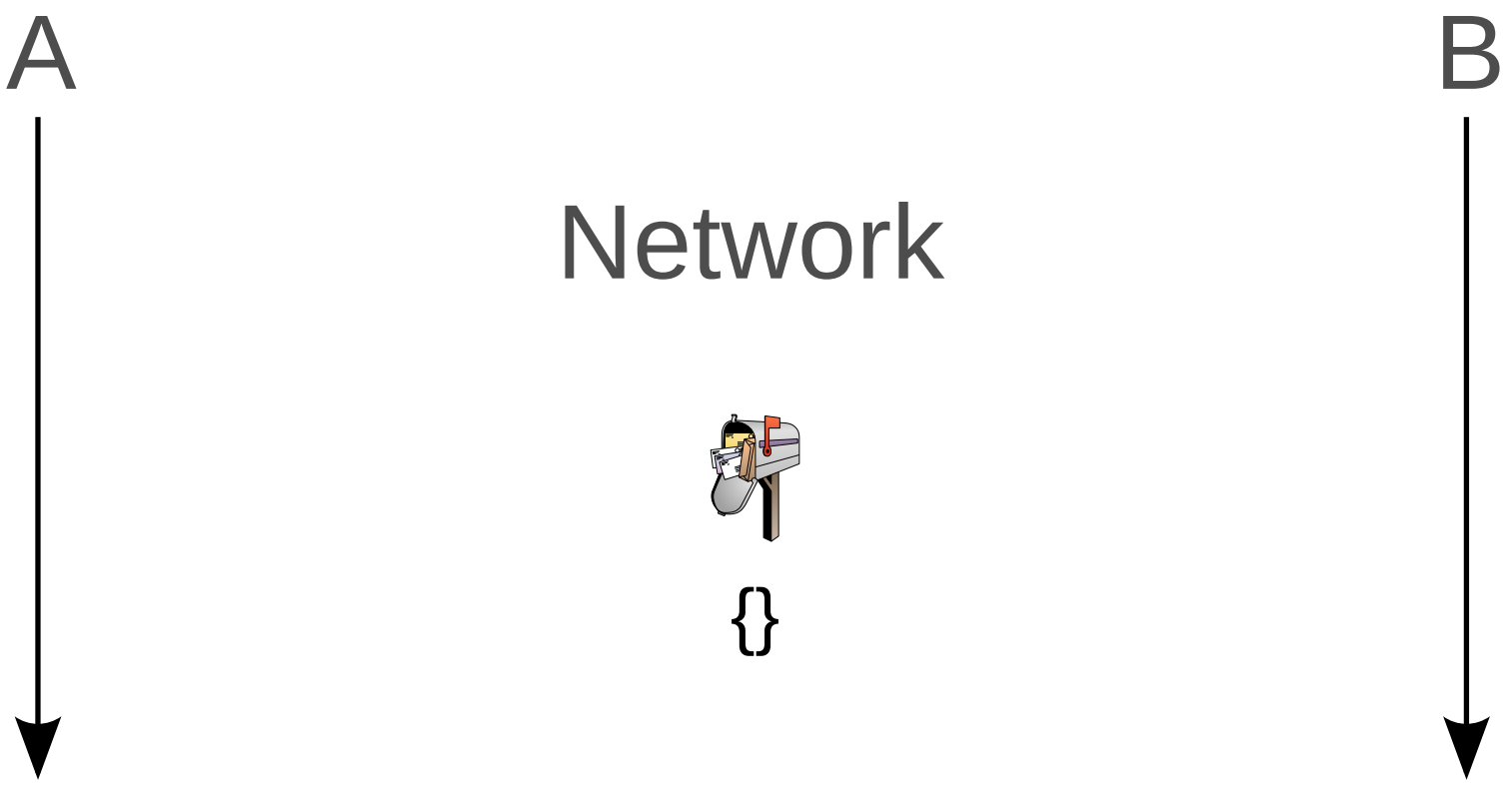


(In)Dependence Theorems

Theorem

Any two Send and Recv transitions are independent.

$$\forall p_1, p_2 \in Proc, rdv_1, rdv_2 \in RdV, d_1, d_2 \in Addr, c_1, c_2 \in Addr : \\ I(Send(p_1, rdv_1, d_1, c_1), Recv(p_2, rdv_2, d_2, c_2))$$

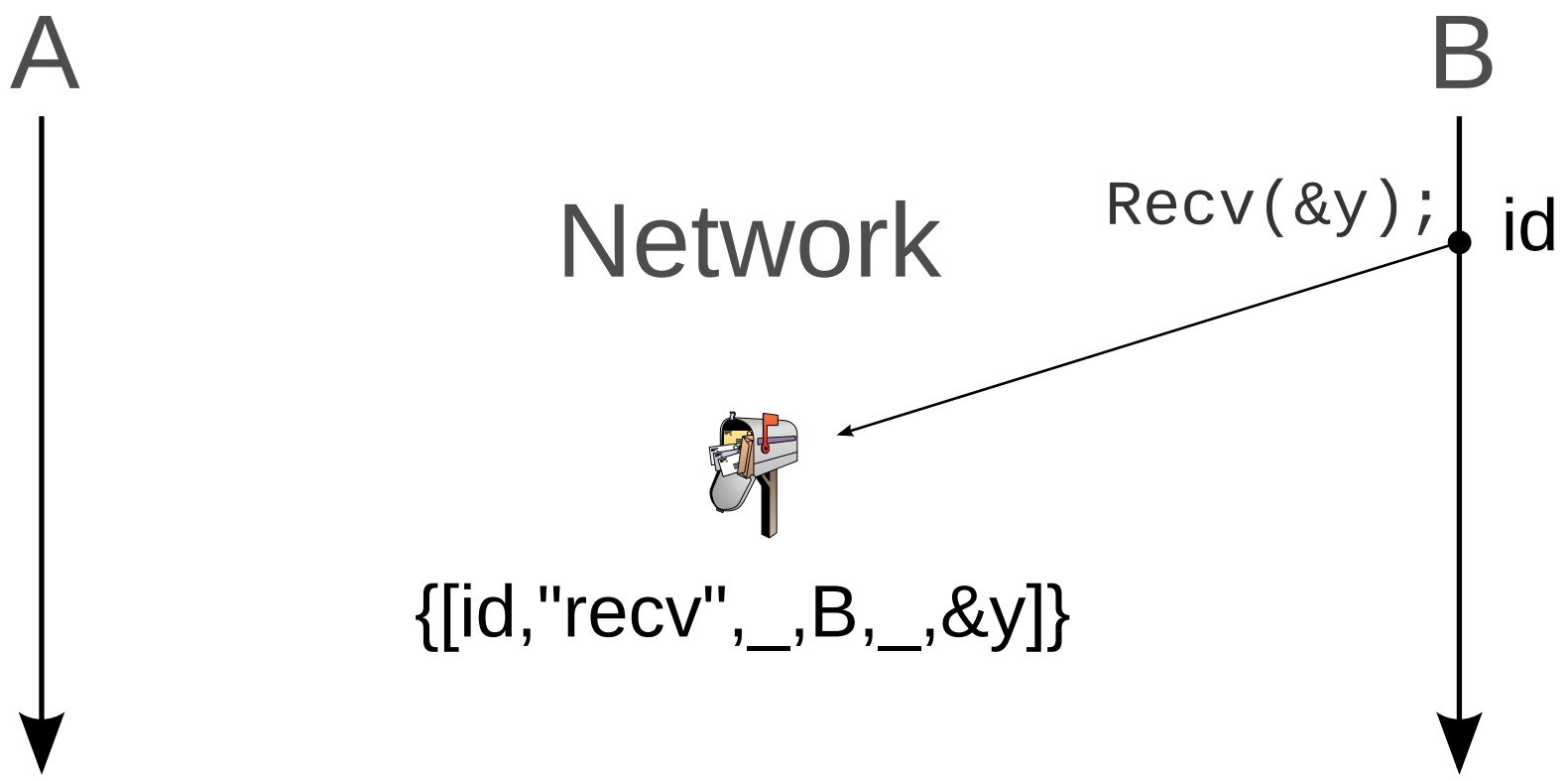


(In)Dependence Theorems

Theorem

Any two Send and Recv transitions are independent.

$$\forall p_1, p_2 \in Proc, rdv_1, rdv_2 \in RdV, d_1, d_2 \in Addr, c_1, c_2 \in Addr : \\ I(Send(p_1, rdv_1, d_1, c_1), Recv(p_2, rdv_2, d_2, c_2))$$

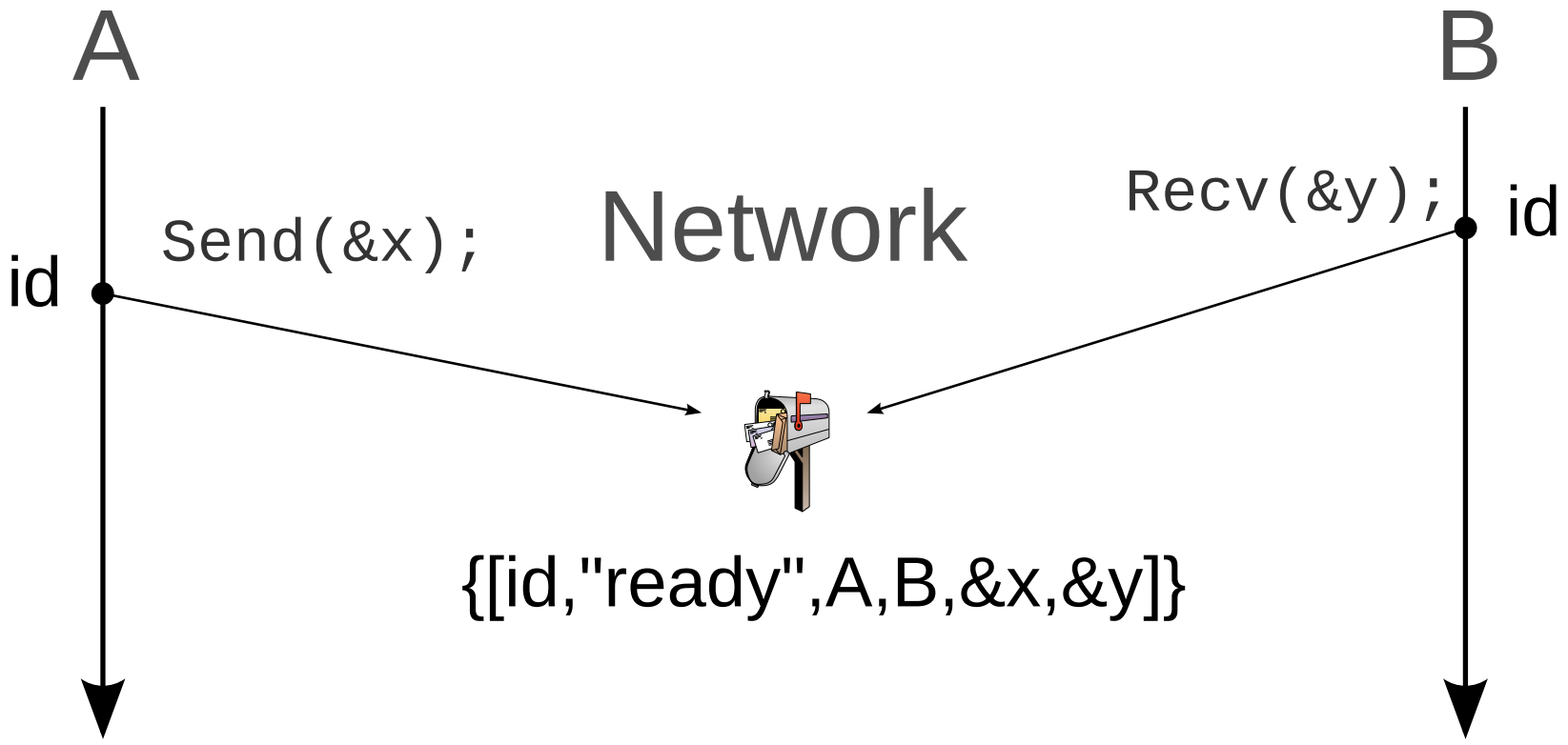


(In)Dependence Theorems

Theorem

Any two Send and Recv transitions are independent.

$$\forall p_1, p_2 \in Proc, rdv_1, rdv_2 \in RdV, d_1, d_2 \in Addr, c_1, c_2 \in Addr : \\ I(Send(p_1, rdv_1, d_1, c_1), Recv(p_2, rdv_2, d_2, c_2))$$

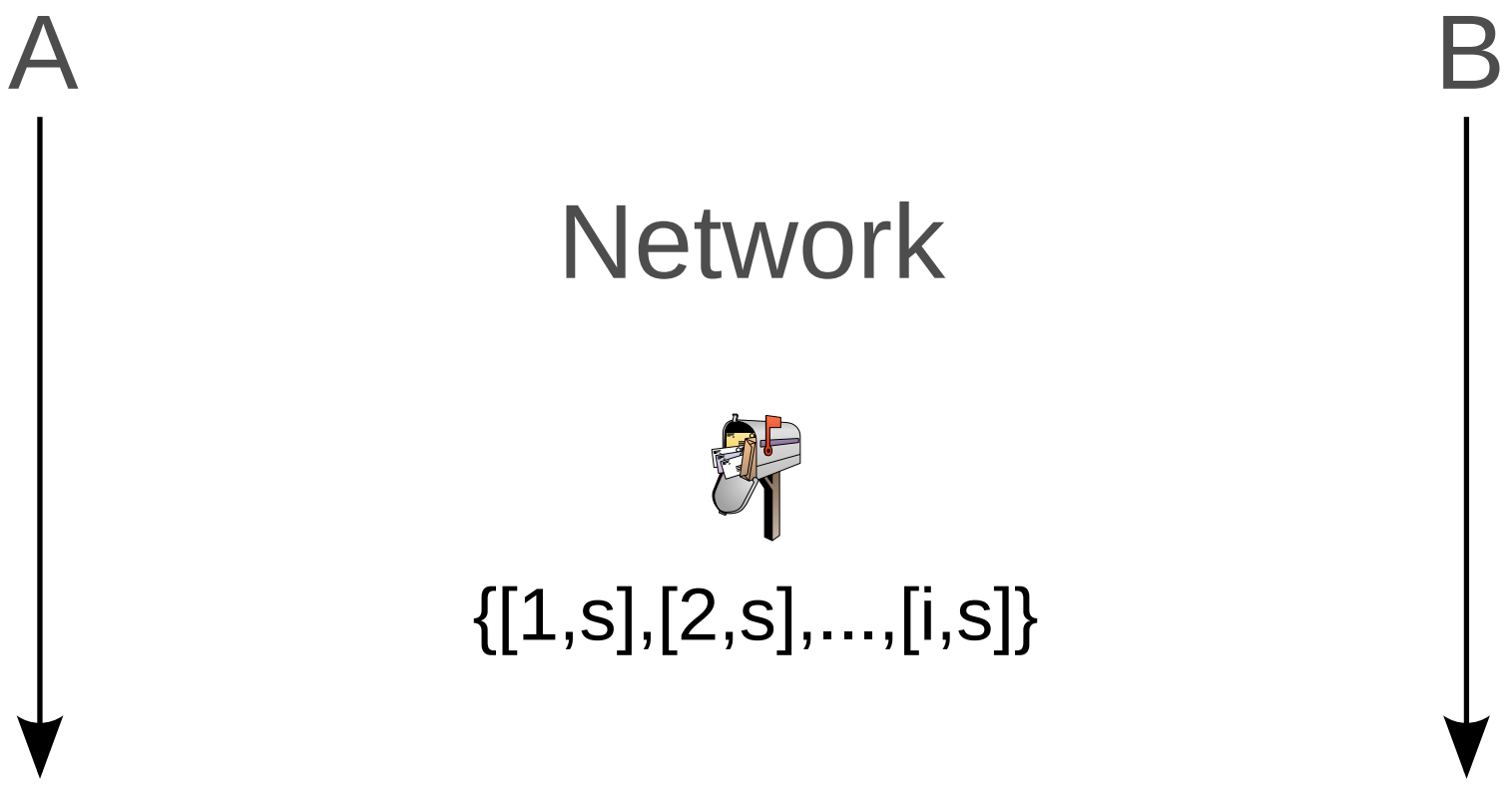


(In)Dependence Theorems

Theorem

Any two Send and Recv transitions are independent.

$$\forall p_1, p_2 \in Proc, rdv_1, rdv_2 \in RdV, d_1, d_2 \in Addr, c_1, c_2 \in Addr : \\ I(Send(p_1, rdv_1, d_1, c_1), Recv(p_2, rdv_2, d_2, c_2))$$

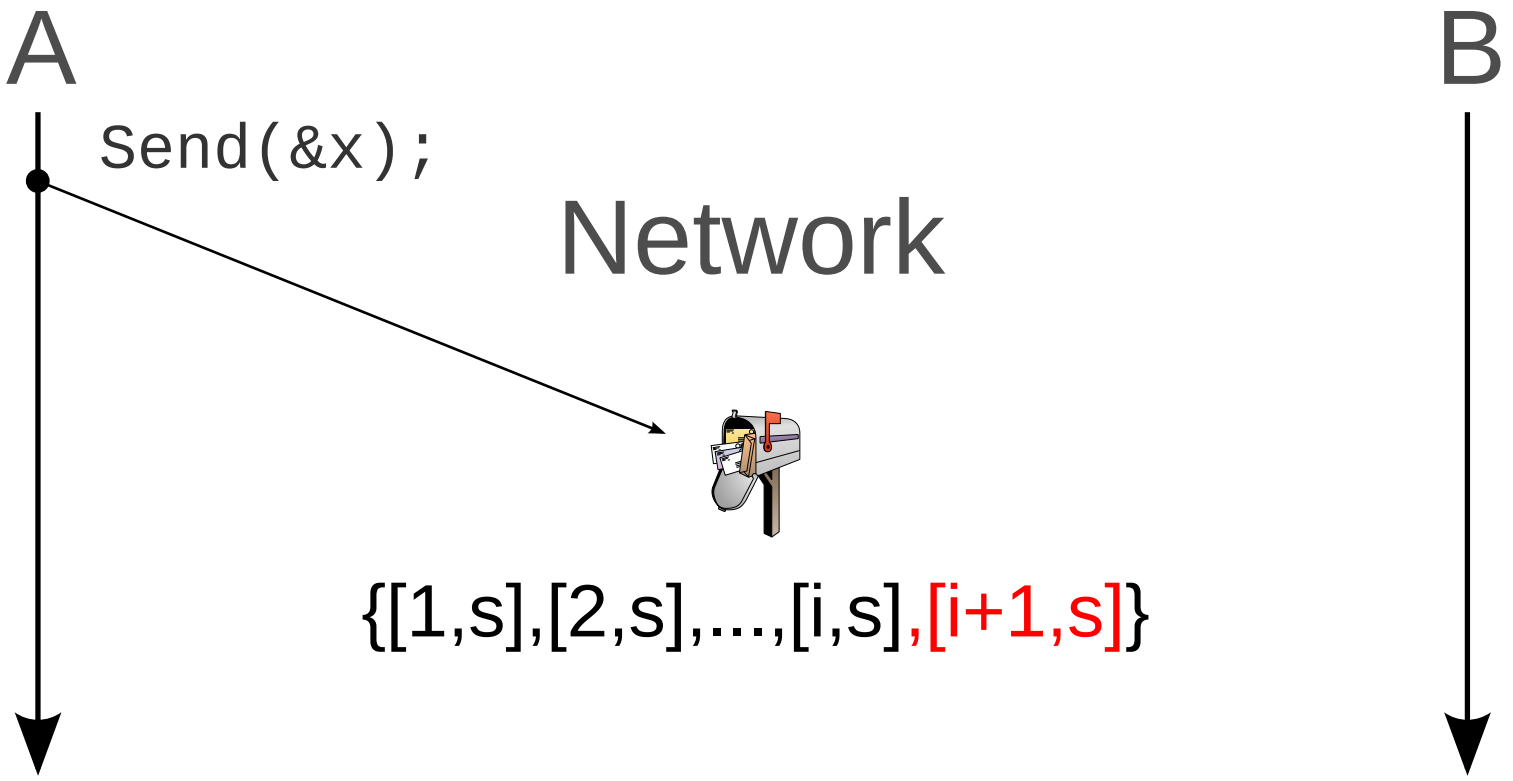


(In)Dependence Theorems

Theorem

Any two Send and Recv transitions are independent.

$$\forall p_1, p_2 \in Proc, rdv_1, rdv_2 \in RdV, d_1, d_2 \in Addr, c_1, c_2 \in Addr :$$
$$I(Send(p_1, rdv_1, d_1, c_1), Recv(p_2, rdv_2, d_2, c_2))$$

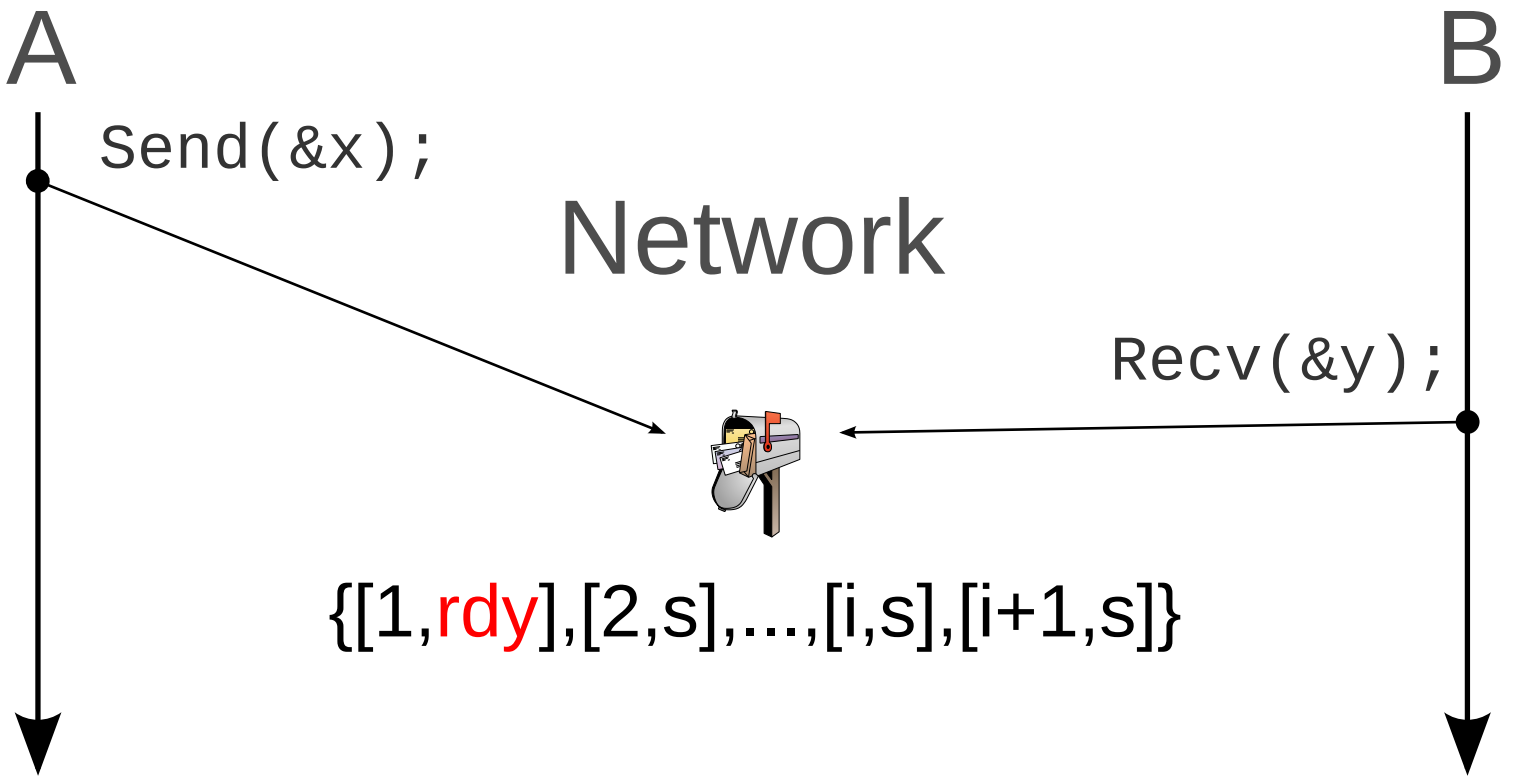


(In)Dependence Theorems

Theorem

Any two Send and Recv transitions are independent.

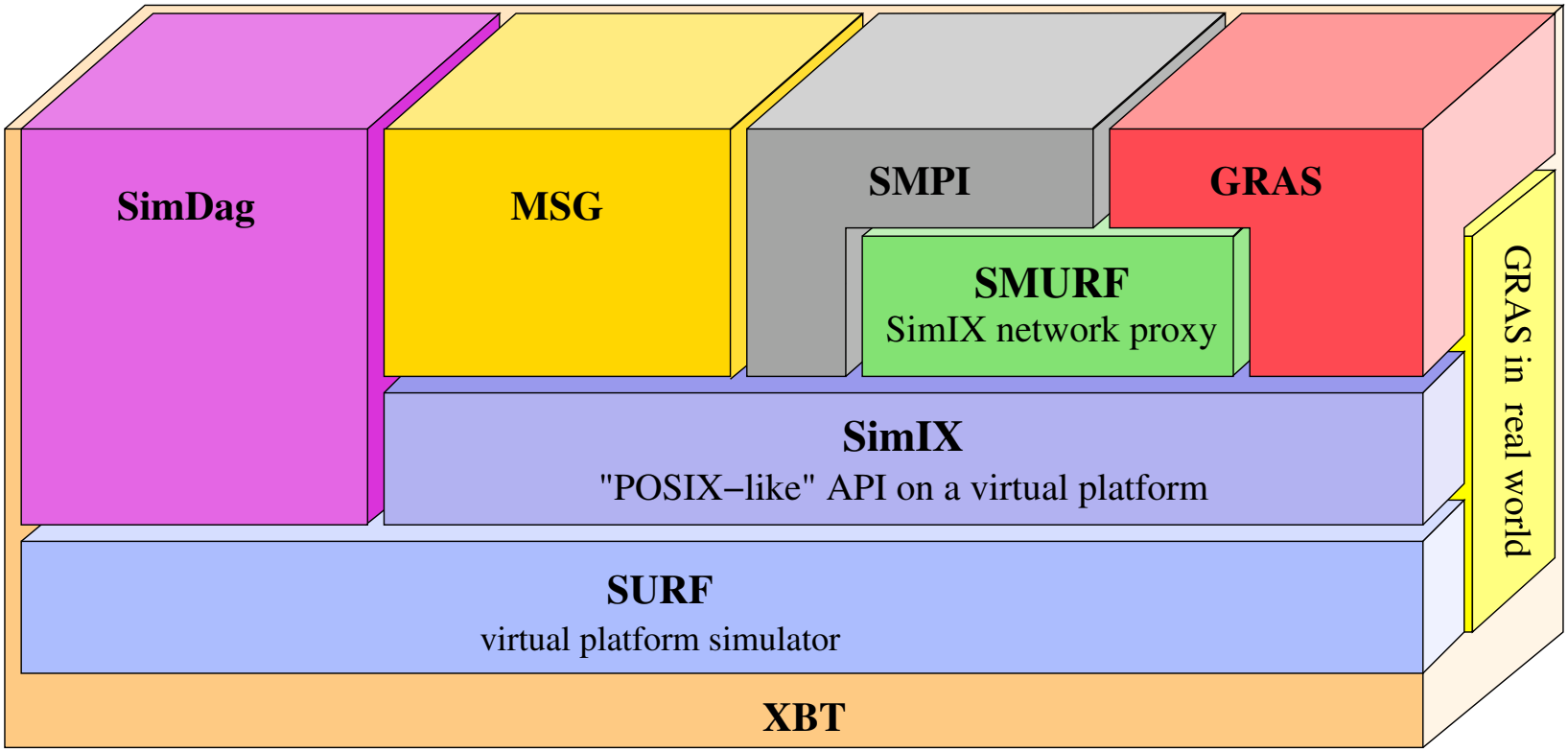
$$\forall p_1, p_2 \in Proc, rdv_1, rdv_2 \in RdV, d_1, d_2 \in Addr, c_1, c_2 \in Addr :$$
$$I(Send(p_1, rdv_1, d_1, c_1), Recv(p_2, rdv_2, d_2, c_2))$$



The Predicate I

$$
\begin{aligned}
I(t_i, t_j) \quad \triangleq \; & \vee \; t_i = Send(\_, \_, \_, \_) \wedge t_j = Recv(\_, \_, \_, \_) \\
& \vee \; \wedge \; t_i = Send(p_1, rdv_1, \_, \_) \wedge t_j = Send(p_2, rdv_2, \_, \_) \\
& \quad \wedge \; p_1 \neq p_2 \wedge rdv_1 \neq rdv_2 \\
& \vee \; \wedge \; t_i = Recv(p_1, rdv_1, \_, \_) \wedge t_j = Recv(p_2, rdv_2, \_, \_) \\
& \quad \wedge \; p_1 \neq p_2 \wedge rdv_1 \neq rdv_2 \\
& \vee \; t_i = WaitAny(\_, \{c\}) \wedge t_j = WaitAny(\_, \{c\}) \\
& \vee \; t_i = Test(\_, c, \_) \wedge t_j = Test(\_, c, \_) \\
& \vee \; t_i = WaitAny(\_, \{c\}) \wedge t_j = Test(\_, c,_{)} \\
& \vee \; t_i = Local(\_) t_j = Local(\_)
\end{aligned}
$$

SimGrid's Internals



Experiment 1

Server code

```
count = 0
while (count<3) {
 value = receive_from("M")
 count++
}
assert( value == 3 )
```

Code of clients 1..3

```
client(int ID) {
  // do something
  if(done){
    send_to(ID, "M")
  }
}
```

Number of explored traces:

- Classic DFS: 20064
- With DPOR: 14778

First Counter-example

```
[server]  Recv
[client1] Send
[client3] Send
[client2] Send
[server]  Wait
[client1] Wait
[server]  Recv
[client3] Wait
[server]  Wait
[server]  Recv
[client2] Wait
[server]  Wait
```

Experiment 2

Server code

```
// step 1
val1 = receive_from("M")
val2 = receive_from("M")
assert(min(val1,val2) == 1)
// step 2
val1 = receive_from("M")
val2 = receive_from("M")
assert(min(val1,val2) == 1)
```

Code of client 1

```
send_to(1,"M") // step 1
send_to(1,"M") // step 2
```

Code of client 2

```
send_to(2,"M") // step 2
send_to(2,"M") // step 2
```

Number of explored traces:

- Classic DFS: 96420
- With DPOR: 32268

Second Counter-example

```
[server]  Recv
[client1] Send
[server]  Wait
[client1] Wait
[server]  Recv
[client1] Send
[client2] Send
[server]  Wait
[client1] Wait
[server]  Recv
[client2] Wait
[server]  Wait
[client2] Send
...
```

Conclusion and Future Work

Conclusion:

- Model-checking actual distributed programs is a current challenges in verification.
- It is feasible only if reduction algorithms such as DPOR are employed.
- Realistic communication APIs for distributed programming are large and complex.
- We propose instead to identify a small set of core primitives that are sufficiently expressive for encoding realistic APIs, with a formal semantic.
- Implementation in the SimGrid framework.

Future Work:

- Validate our approach over larger distributed programs
- Extend it to cover more complex properties, including liveness properties

Bibliography I

Henri Casanova, Arnaud Legrand, and Martin Quinson.
SimGrid: a Generic Framework for Large-Scale Distributed Experiments.
In *10th IEEE International Conference on Computer Modeling and Simulation*, March 2008.

Salman Pervez, Ganesh Gopalakrishnan, Robert M. Kirby, Robert Palmer, and Rajeev Thakur.
Practical model checking method for verifying correctness of MPI programs.
In *EuroPVM/MPI*, pages 344–353. Springer, 2007.