Santa Claus – *with Mobile Reindeer and Elves*

Peter Welch (**phw@kent.ac.uk**)
Matt Pedersen (**matt@faculty.egr.unlv.edu**)

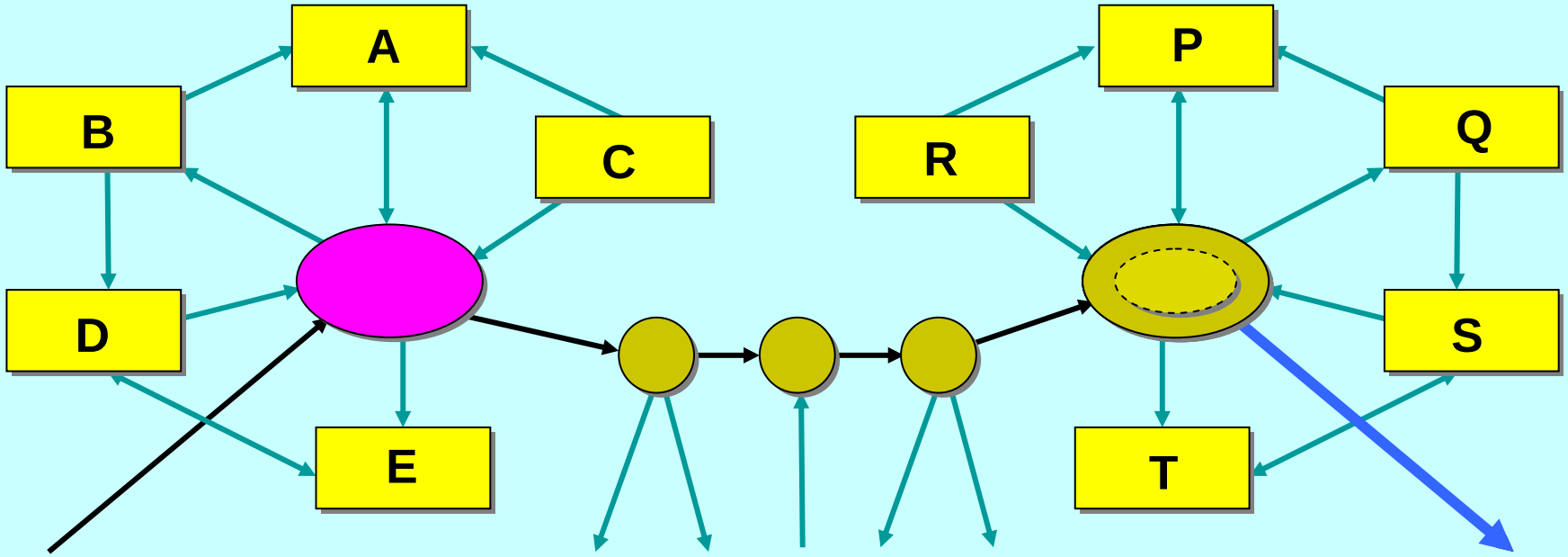
CPA 2008 (09/07/2008)

MOBILE processes …

Santa Claus …

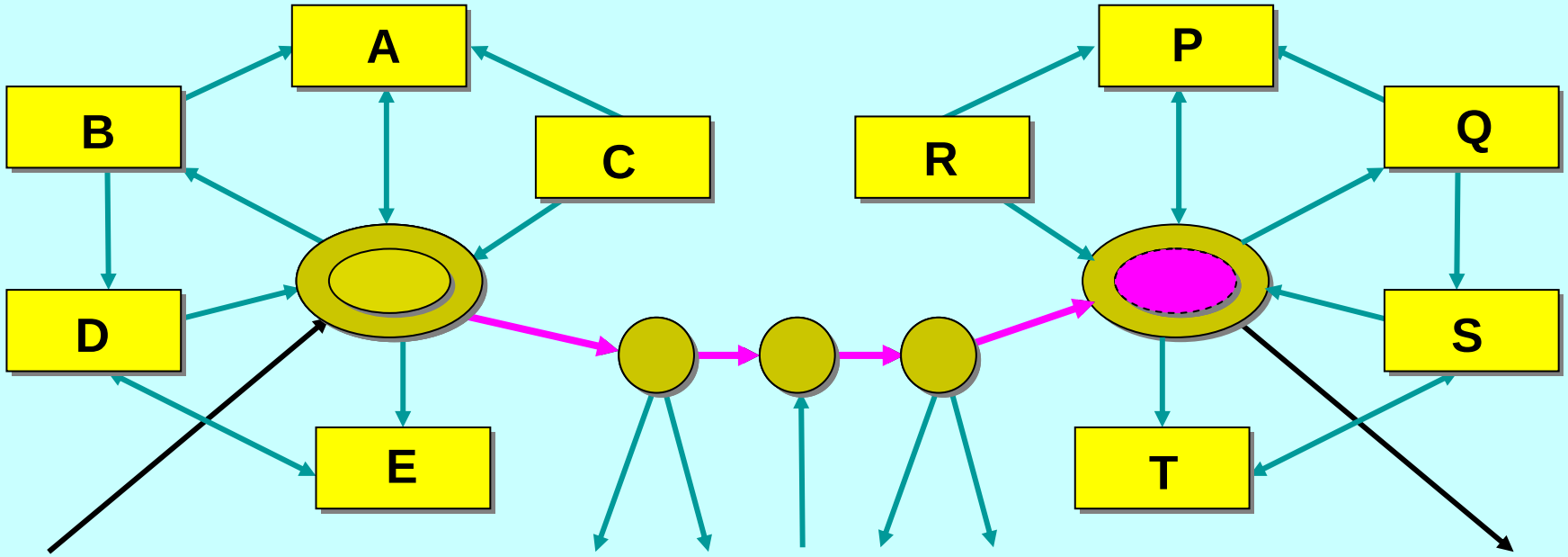
Mobile *Process* Types

An ***occam-π*** mobile process, embedded anywhere in a dynamically evolving network, may ***suspend*** itself mid-execution, be safely ***disconnected*** from its local environment, ***moved*** (by channel communication) to a new environment, ***reconnected*** to that new environment and ***reactivated***.



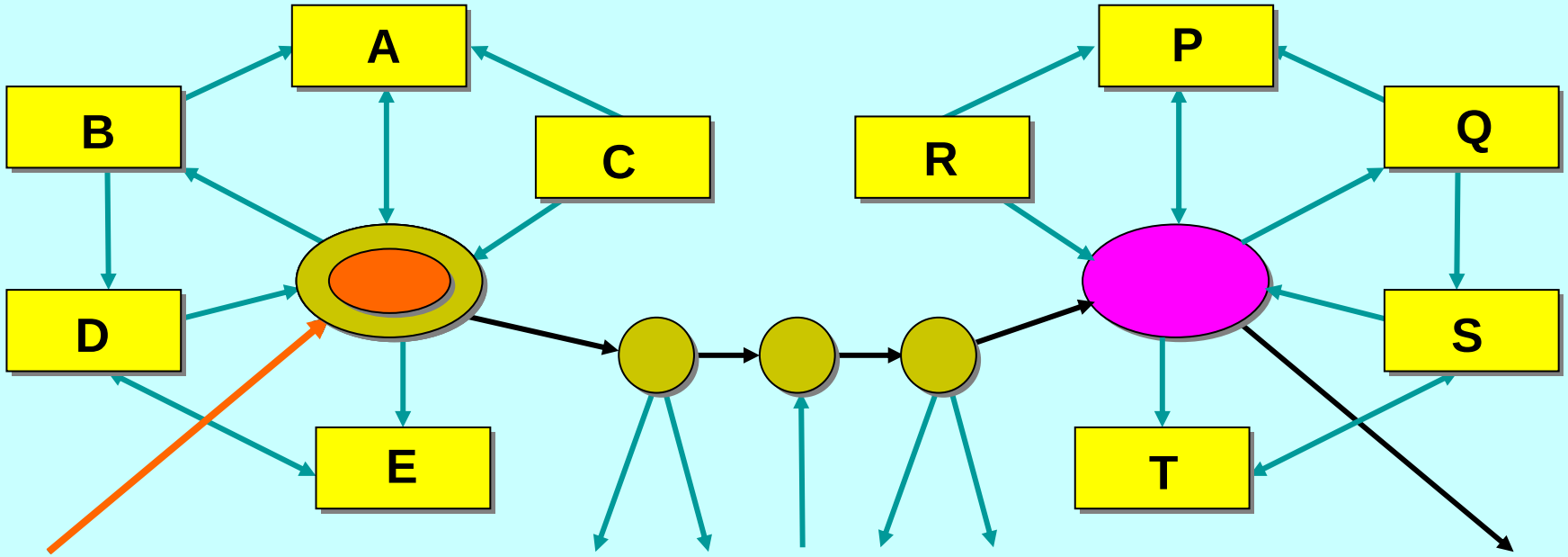
Mobile *Process* Types

An **occam-π** mobile process, embedded anywhere in a dynamically evolving network, may ***suspend*** itself mid-execution, be safely ***disconnected*** from its local environment, ***moved*** (by channel communication) to a new environment, ***reconnected*** to that new environment and ***reactivated***.



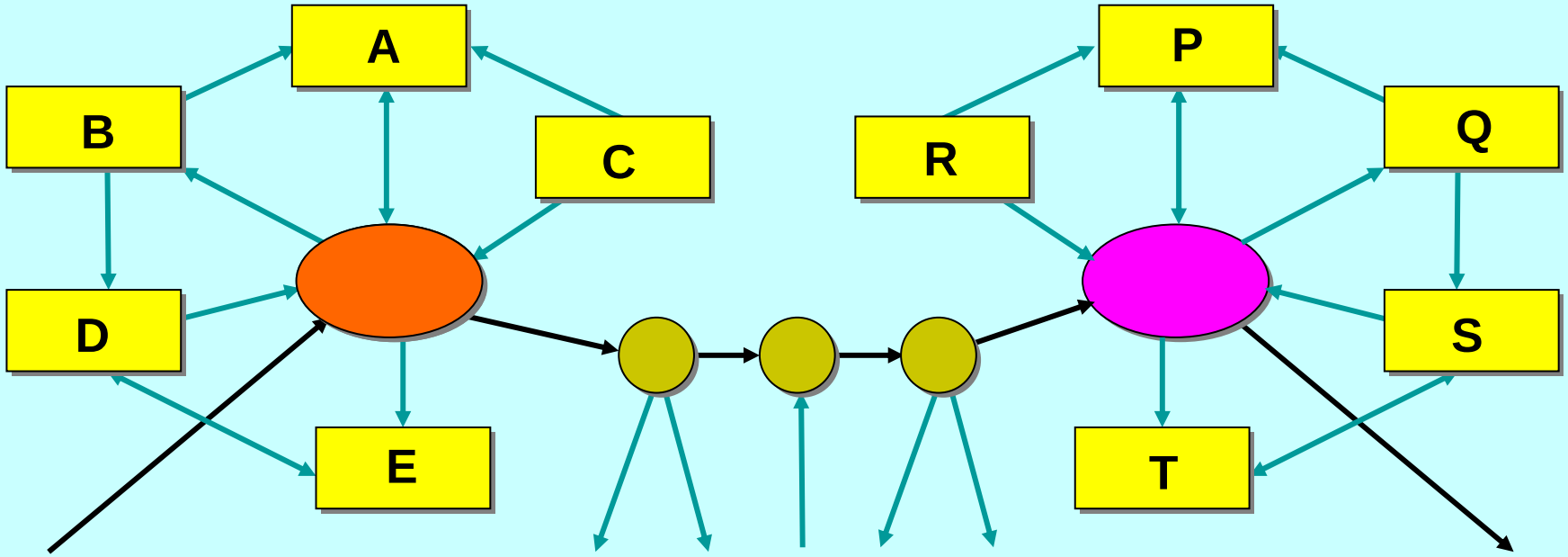
Mobile *Process* Types

An ***occam-π*** mobile process, embedded anywhere in a dynamically evolving network, may ***suspend*** itself mid-execution, be safely ***disconnected*** from its local environment, ***moved*** (by channel communication) to a new environment, ***reconnected*** to that new environment and ***reactivated***.



Mobile *Process* Types

An ***occam-π*** mobile process, embedded anywhere in a dynamically evolving network, may ***suspend*** itself mid-execution, be safely ***disconnected*** from its local environment, ***moved*** (by channel communication) to a new environment, ***reconnected*** to that new environment and ***reactivated***.



Mobile *Process* Types

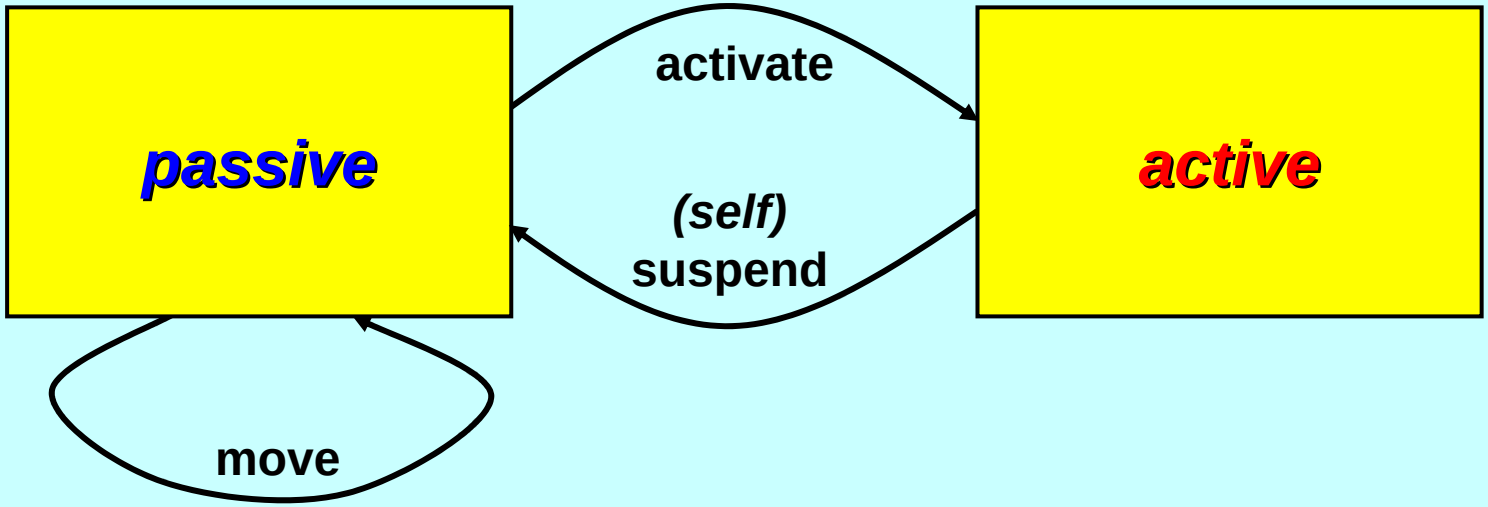
An ***occam-π*** mobile process, embedded anywhere in a dynamically evolving network, may ***suspend*** itself mid-execution, be safely ***disconnected*** from its local environment, ***moved*** (by channel communication) to a new environment, ***reconnected*** to that new environment and ***reactivated***.

Upon reactivation, the process resumes from the same state ***(i.e. data values and code positions)*** it held when suspended. Its view of that environment is unchanged, ***since that is abstracted by its channel interface***. The environment on the other side of that abstraction, however, will usually be different.

The mobile process may itself contain ***any number of levels*** of dynamically evolving parallel sub-network.

Mobile *Process* Types

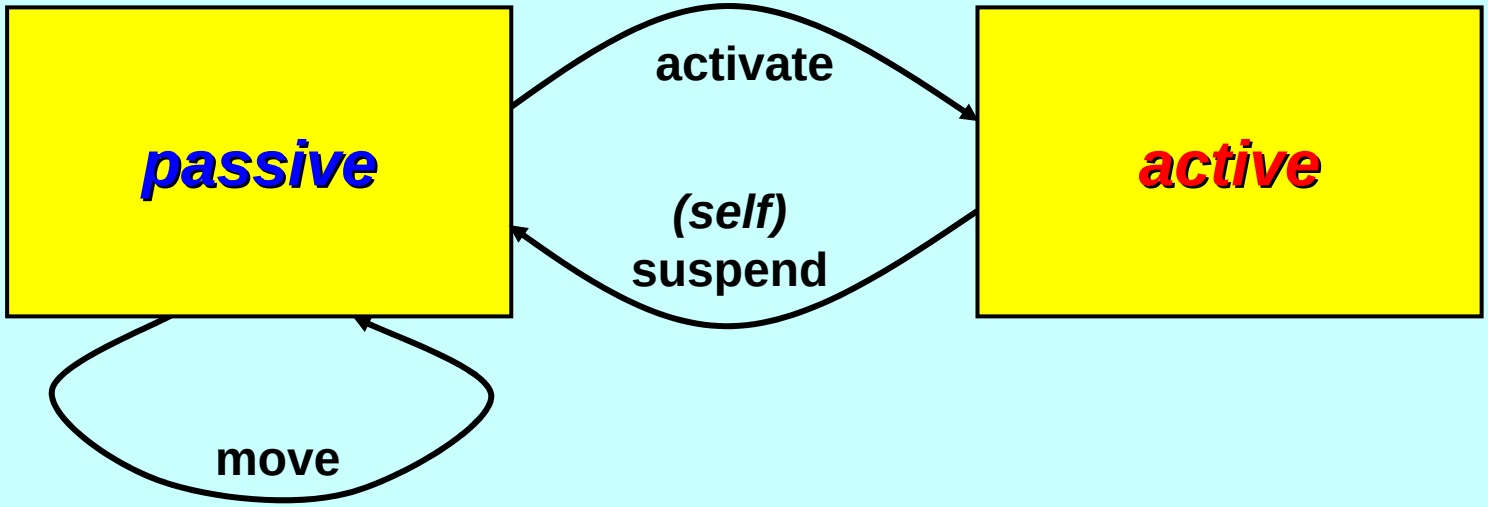
Mobile processes are entities encapsulating state and code. They may be ***active*** or ***passive***. Initially, they are ***passive***.



The state of a mobile process can only be felt by interacting with it when ***active***. When ***passive***, its state is locked – even against reading.

Mobile *Process* Types

When ***passive***, they may be ***activated*** or ***moved***. A ***moved*** process remains ***passive***. An ***active*** process cannot be ***moved*** or ***activated*** in parallel.



When an ***active*** mobile process ***suspends***, it becomes ***passive*** – retaining its state and code position. When it moves, its state moves with it. When re-***activated***, it sees its previous state and continues from where it left off.

Mobile *Process* Types

Mobile processes exist in many technologies – such as ***applets***, ***agents*** and in distributed operating systems.

occam-π offers (will offer) support for them with a formal ***denotational*** and ***refinement*** semantics, safety and very low overheads.

Process mobility semantics follows naturally from that for mobile data and mobile channel-ends.

We need to introduce a concept of process ***types*** and ***variables***.

Mobile *Process* Types

Process ***type*** declarations give names to `PROC` header templates. Mobile processes may implement types with synchronisation parameters only (i.e. ***channels***, ***barriers***, ***buckets***, etc.) plus records and fixed-size arrays of the same. For example:

```
PROC TYPE IN.OUT.SUSPEND (CHAN INT in?, out!, suspend?):
```

The above declares a process ***type*** called `IN.OUT.SUSPEND`. Processes implementing this will be given three channels by the (re-)activating host process: two for input (`in?, suspend?`) and one for output (`out!`), all carrying `INT` traffic.

Process ***types*** are used in two ways: for the declaration of process ***variables*** and to define the ***connection interface*** to a mobile process.

Mobile *Process* Example


```
MOBILE PROC integrate.suspend (CHAN INT in?, out!, suspend?)
IMPLEMENTS IN.OUT.SUSPEND
  INITIAL INT total IS 0:    -- local state
  WHILE TRUE
    INT x:
    PRI ALT
      suspend ? x
        SUSPEND  -- control returns to activator
        -- control resumes here when next activated
      in ? x
        SEQ
          total := total + x
          out ! total
:
```

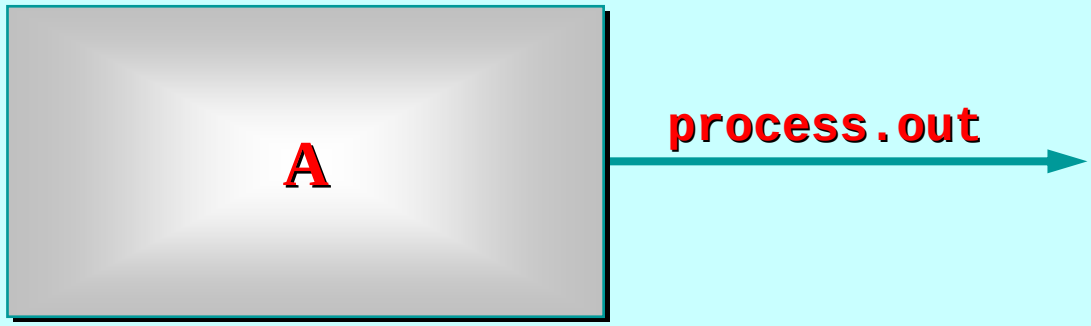
Mobile Processes and Types

A process ***type*** may be implemented by many mobile processes – each offering different behaviours.

The mobile process from the last slide, **`integrate.suspend`**, implements the process type, **`IN.OUT.SUSPEND`**, defined earlier. Other processes could implement the same type.

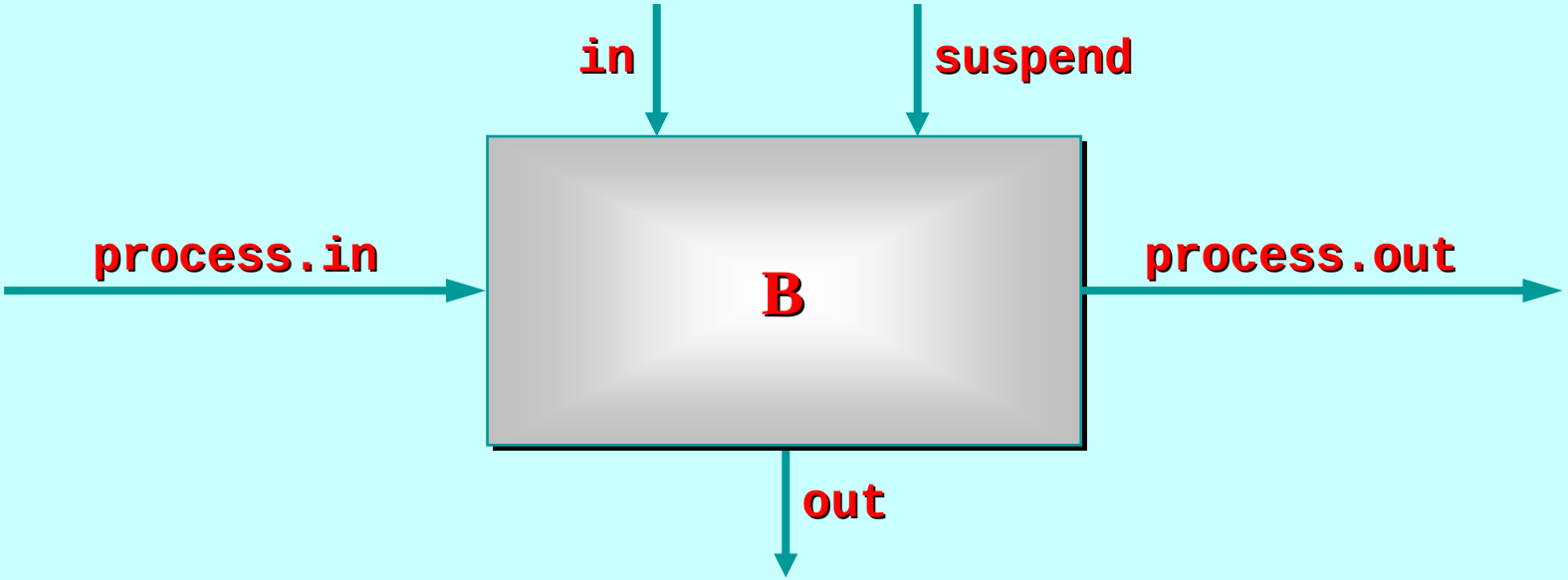
A process ***variable*** has a specific process type. Its value may be ***undefined*** or ***some mobile process*** implementing its type. A process variable may be bound to different mobile processes, offering different behaviours, at different times in its life. When ***defined***, it can only be activated according to that type.

Mobile *Process* Example



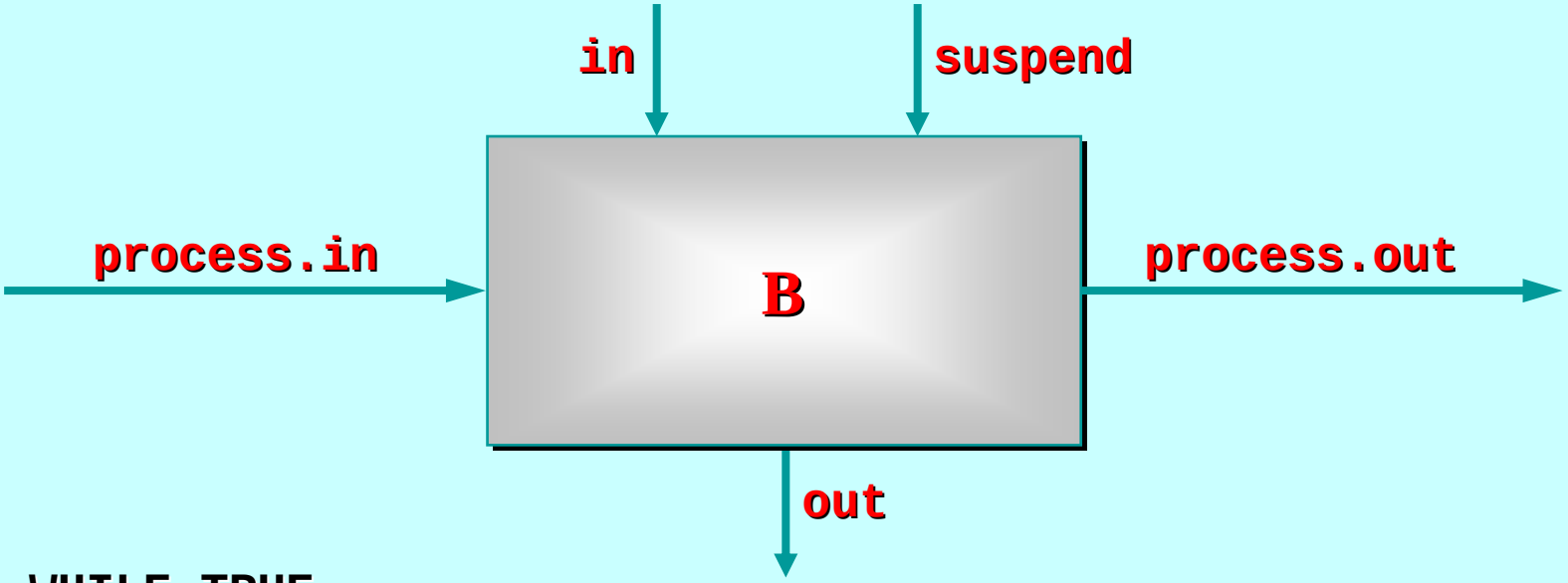

```
PROC A (CHAN IN.OUT.SUSPEND process.out!)
  IN.OUT.SUSPEND p:
  SEQ
    -- p is not yet defined (can't move or activate it)
    p := MOBILE integrate.suspend
    -- p is now defined (can move and activate)
    process.out ! p
    -- p is now undefined (can't move or activate it)
:
```

Mobile *Process* Example



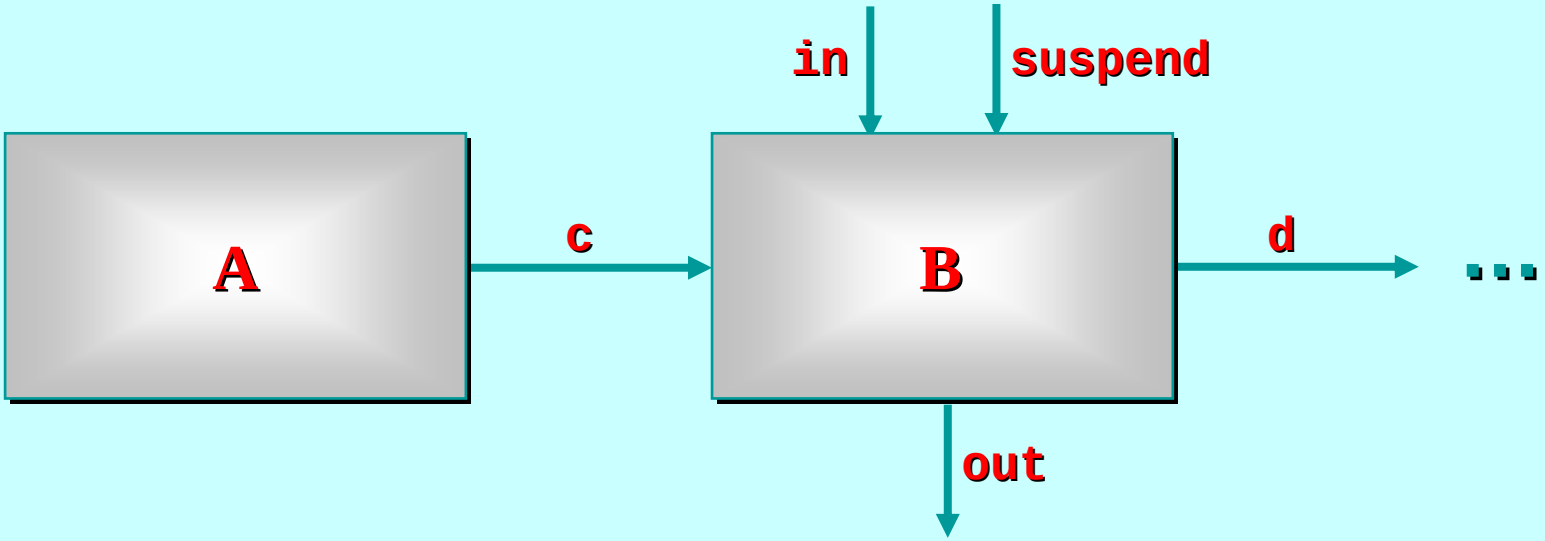

```
PROC B (CHAN IN.OUT.SUSPEND process.in?, process.out!,
        CHAN INT in?, out!, suspend?)
  WHILE TRUE
    IN.OUT.SUSPEND q:
    SEQ
      ...  input a process to q
      ...  plug into local channels and activate q
      ...  when finished, send it on its way
:
```

Mobile *Process* Example




```
WHILE TRUE
  IN.OUT.SUSPEND q:
  SEQ
    -- q is not yet defined (can't move or activate it)
    process.in ? q
    -- q is now defined (can move and activate)
    q (in?, out!, suspend?)
    -- q is still defined (can move and activate)
    process.out ! q
    -- q is now undefined (can't move or activate it)
```

Mobile *Process* Example




```
CHAN IN.OUT.SUSPEND c, d:
CHAN INT in, out, suspend:
...  other channels
PAR
  A (c!)
  B (c?, d!, in?, out!, suspend?)
  ...  other processes
```

MOBILE processes …

Santa Claus …

Santa repeatedly sleeps until wakened by either ***all*** of his ***nine*** reindeer (back from their holidays) or by a group of ***three*** of his ***ten*** elves (who have left their workbenches).

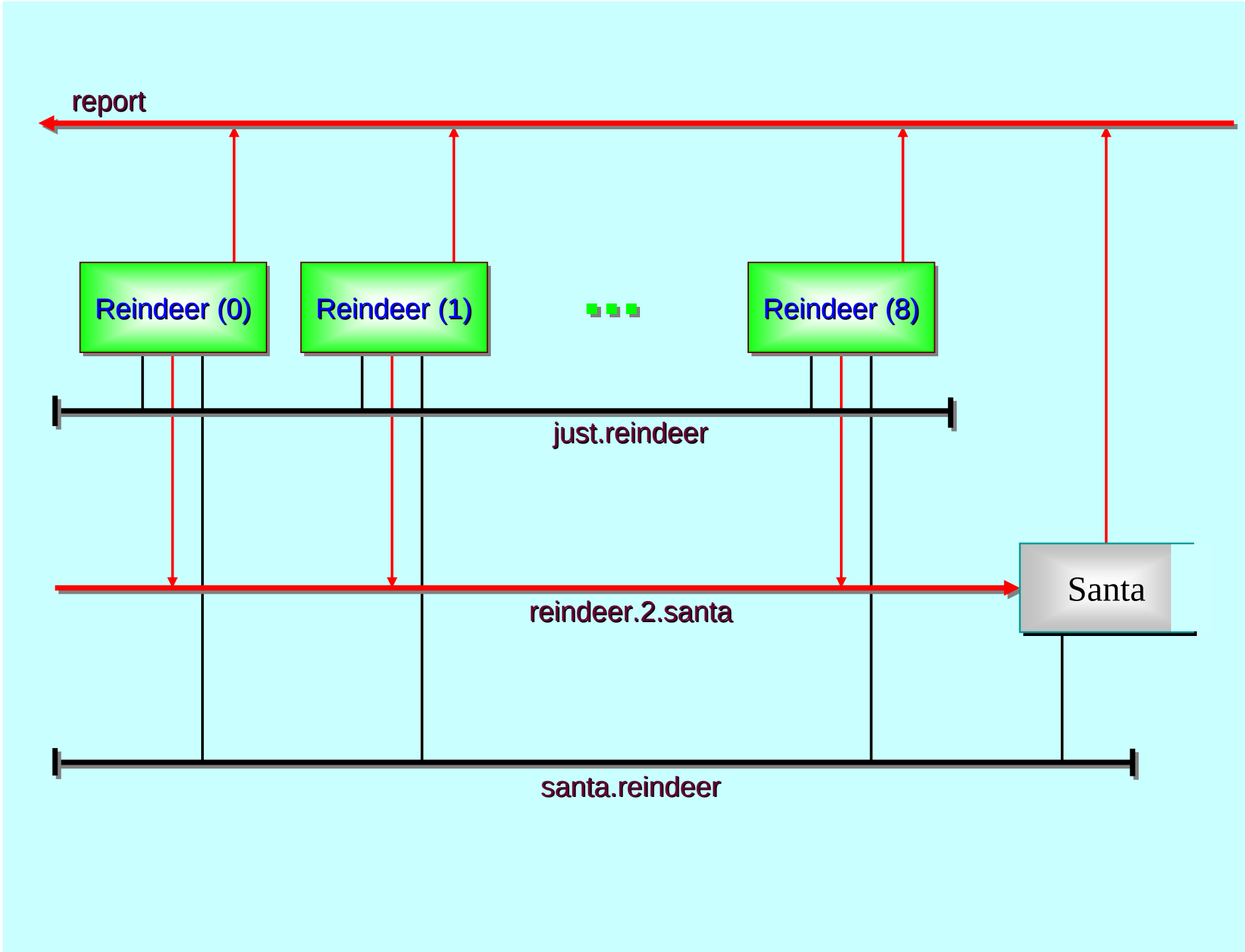
If awakened by the reindeer, he harnesses each of them to his sleigh, delivers toys with them and finally unharnesses them (allowing them to go back on holiday).

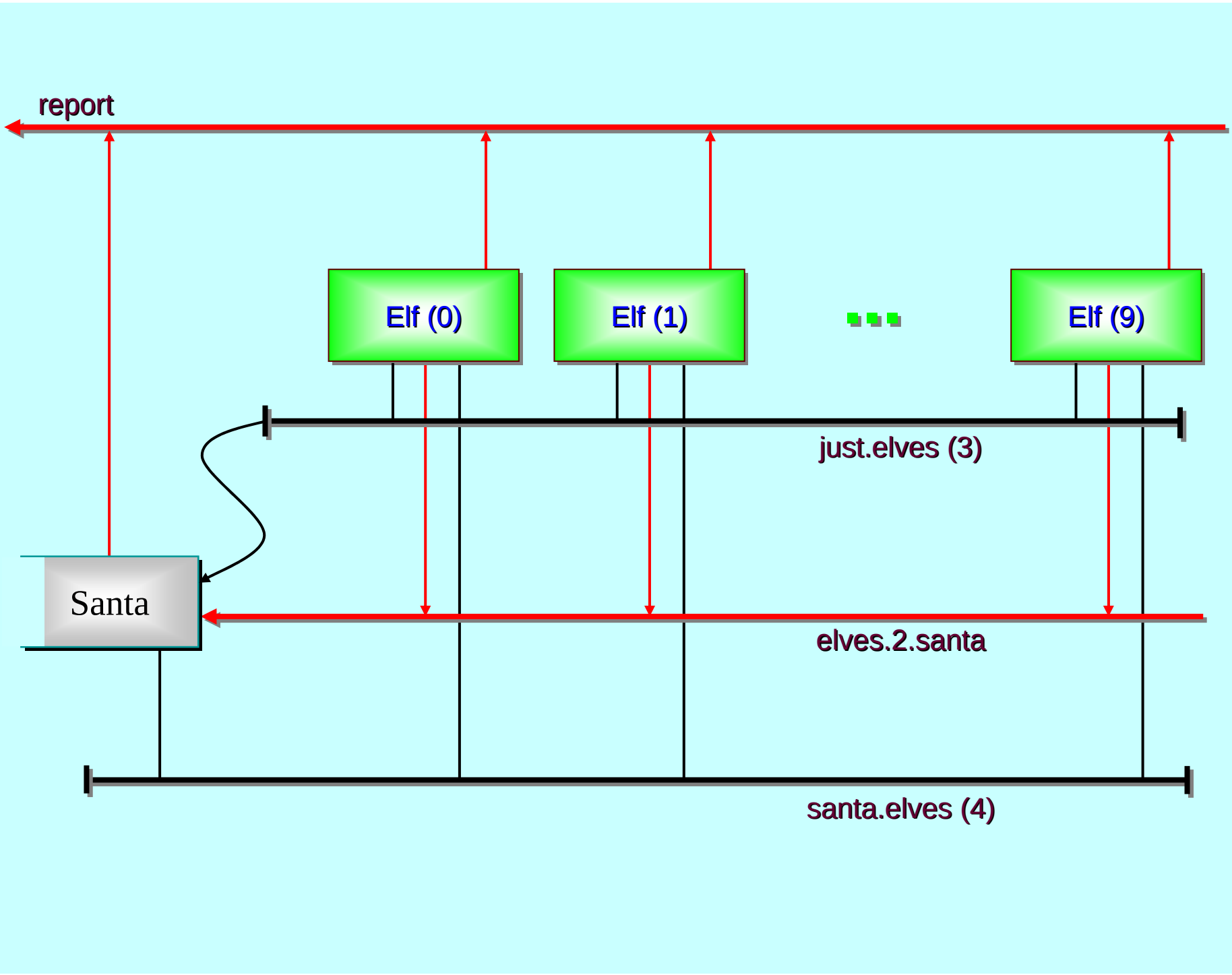
If awakened by a group of elves, he shows each of the group into his study, consults with them on toy R&D and finally shows each of them out (allowing them to go back to work).

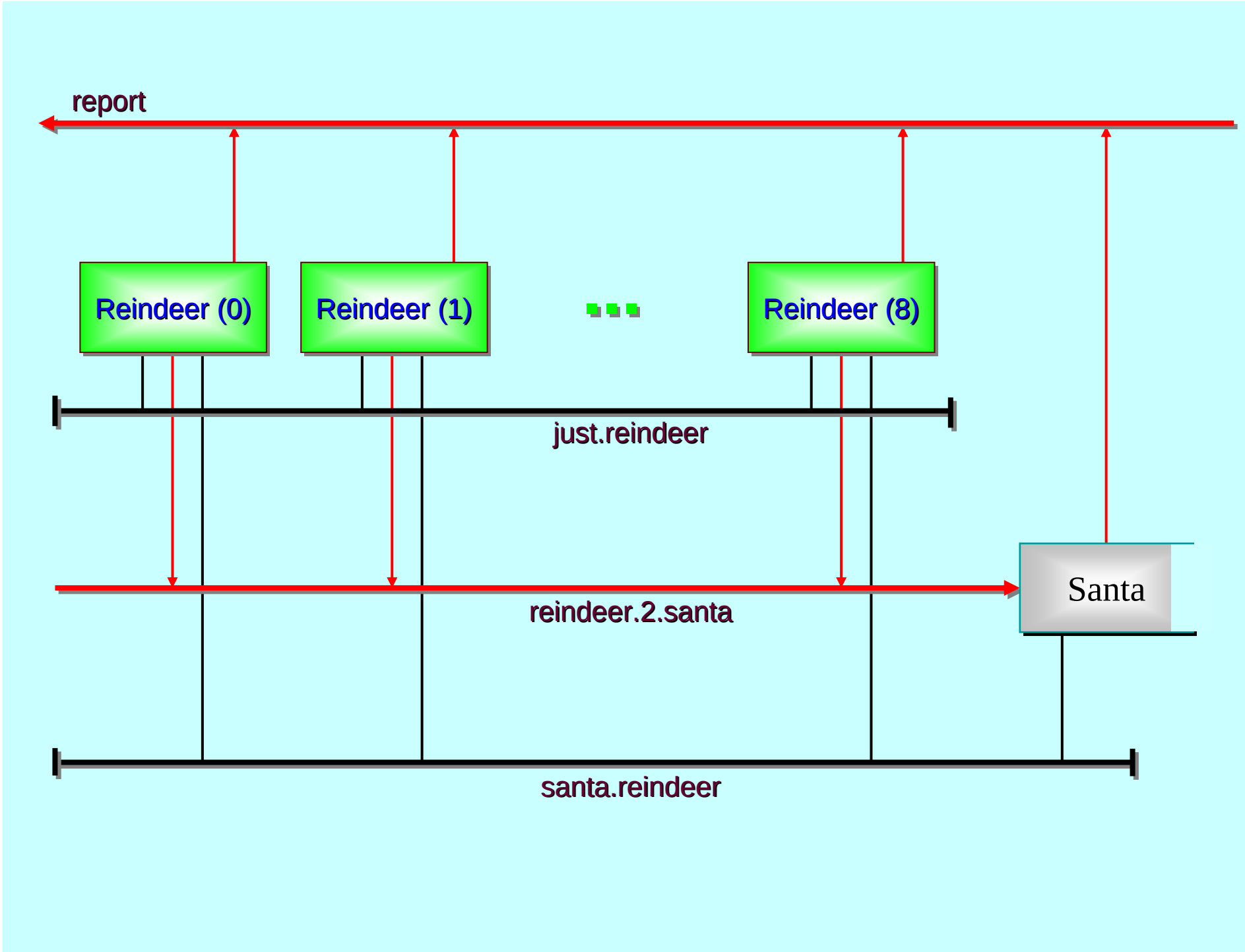
Santa should give priority to the reindeer in the case that there is both a group of elves and a group of reindeer waiting.

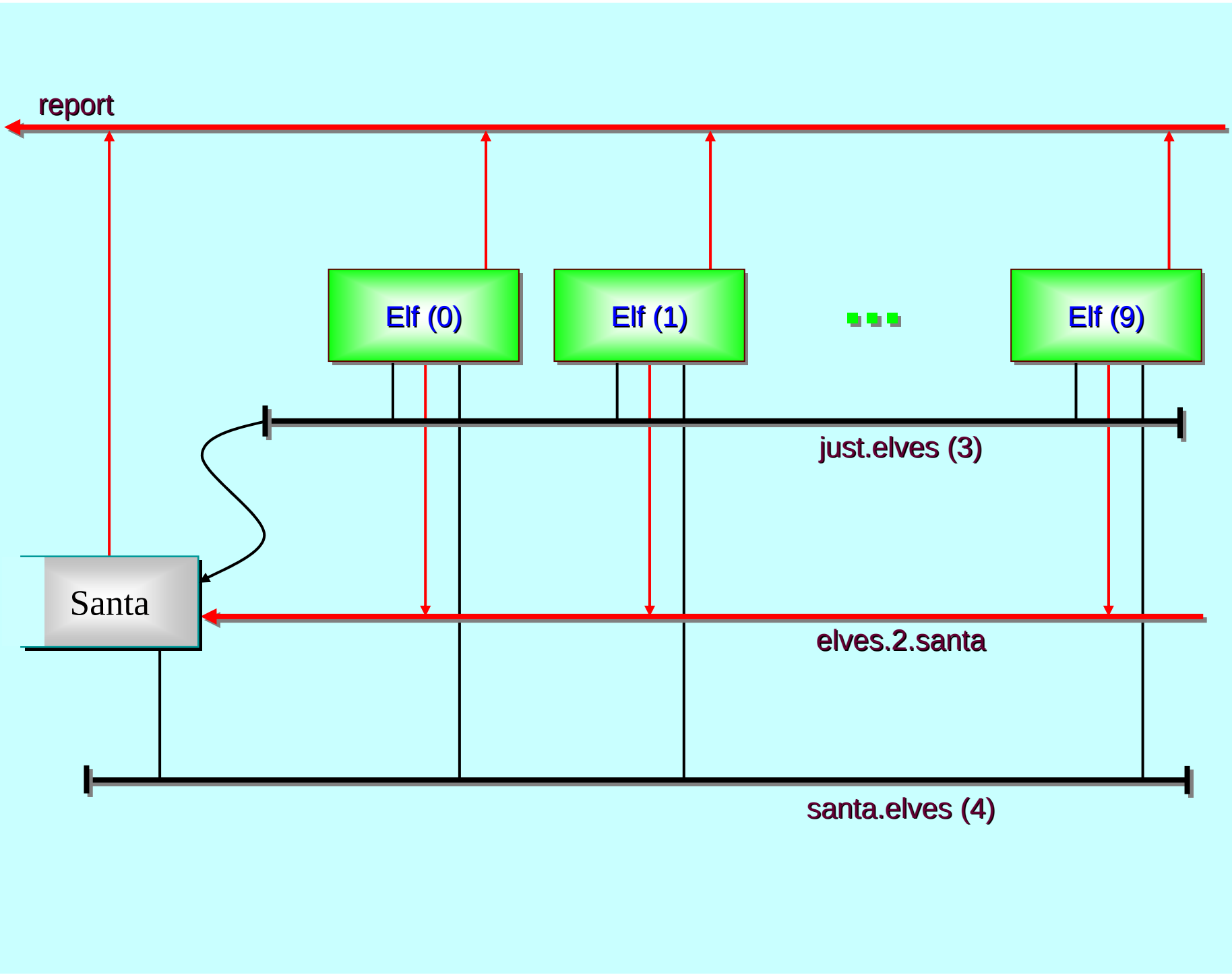
J.A.Trono, “A new exercise in concurrency”, SIGCSE Bulletin 26(3), pp. 8-10, 1994.

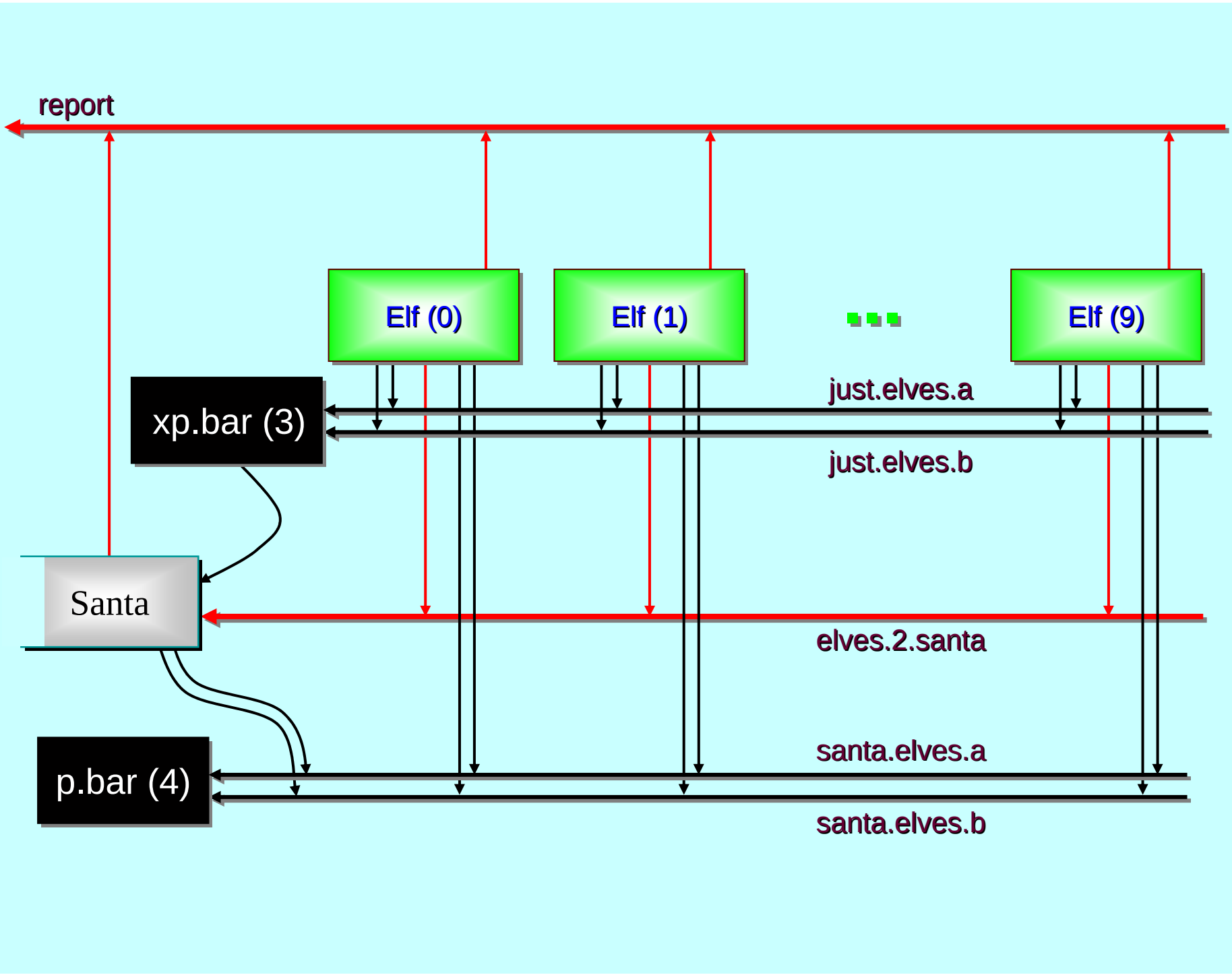
First: a static network (classical occam, shared channels, barriers and partial barriers)


report
Reindeer (0)
Reindeer (1)
Reindeer (8)
just.reindeer
reindeer.2.santa
Santa
santa.reindeer


report
Elf (0)
Elf (1)
Elf (9)
just.elves (3)
Santa
elves.2.santa
santa.elves (4)


report
Reindeer (0)
Reindeer (1)
Reindeer (8)
just.reindeer
reindeer.2.santa
Santa
santa.reindeer

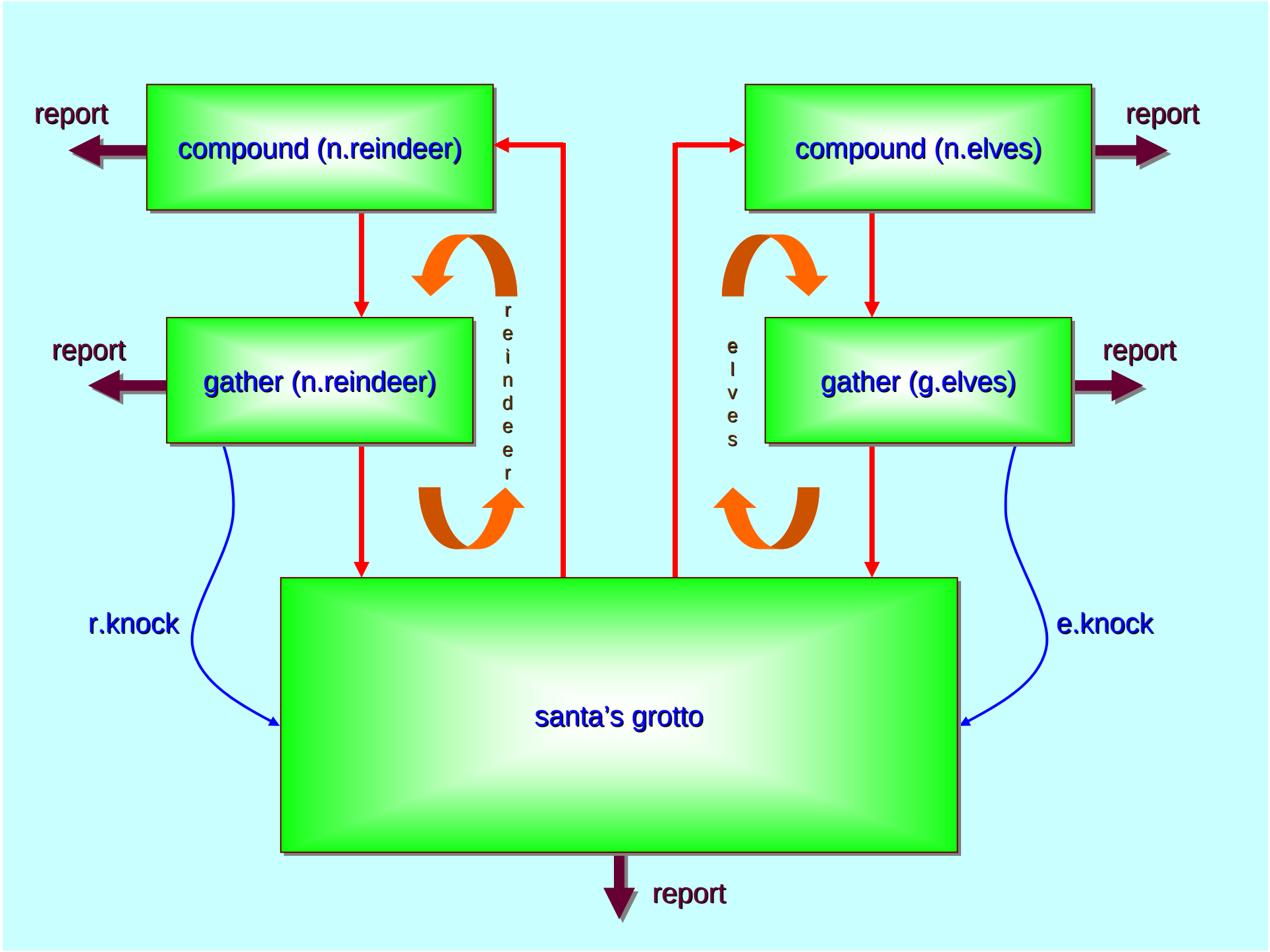

report
Elf (0)
Elf (1)
Elf (9)
just.elves (3)
Santa
elves.2.santa
santa.elves (4)

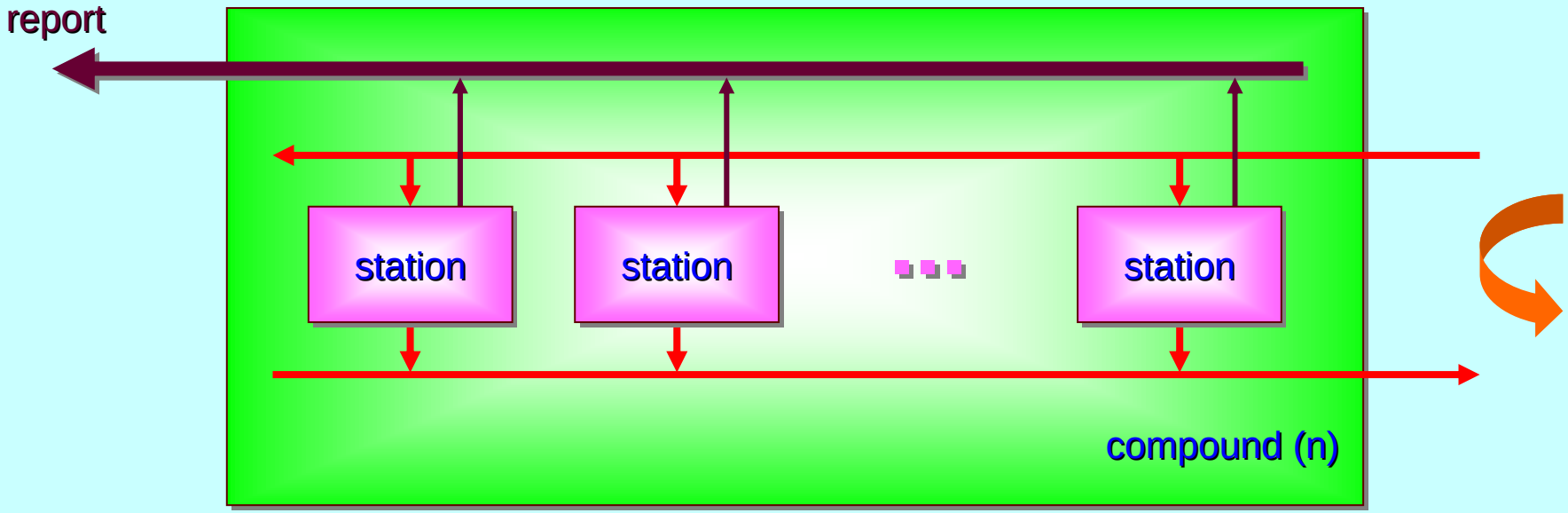

report
Elf (0)
Elf (1)
Elf (9)
xp.bar (3)
just.elves.a
just.elves.b
Santa
elves.2.santa
p.bar (4)
santa.elves.a
santa.elves.b

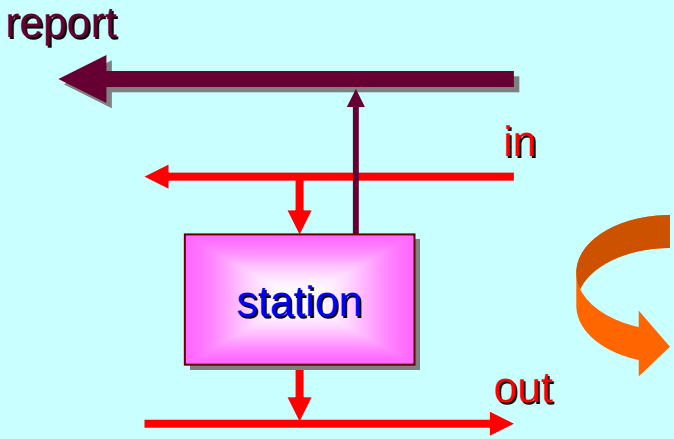
Second: a dynamic network (mobile channels)

Deferred … (ask Adam!)

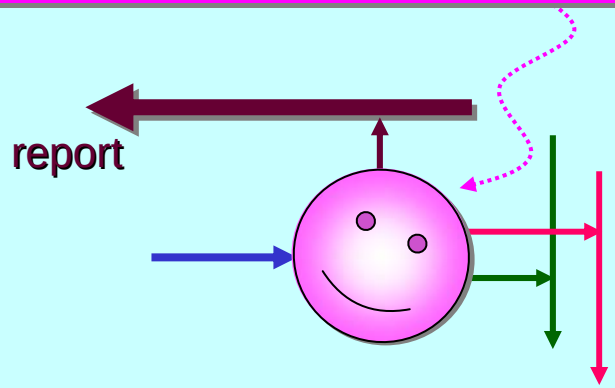
Third: a dynamic network (mobile processes)


report
compound (n.reindeer)
compound (n.elves)
report
report
gather (n.reindeer)
reindeer
elves
gather (g.elves)
report
r.knock
e.knock
santa's grotto
report

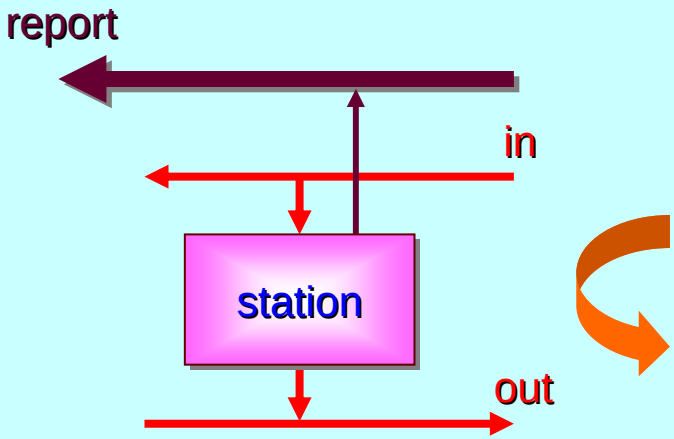

report
station
station
station
compound (n)



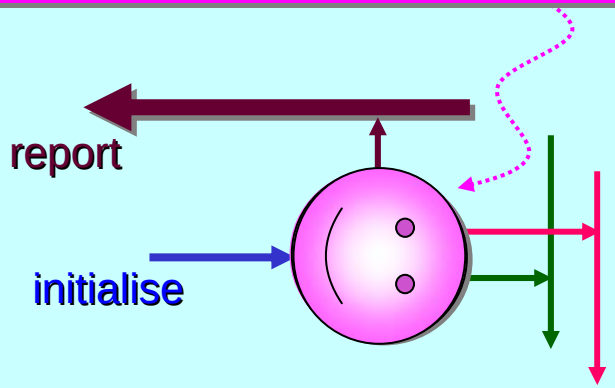
A reindelf : either a reindeer or an elf



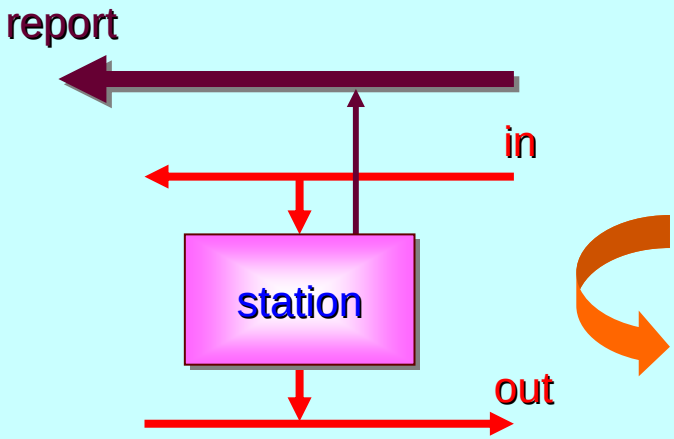

```
PROC station (VAL INT id, seed, kind, away.time,
              SHARED CHAN MOBILE AGENT in?, out!,
              SHARED CHAN AGENT.MESSAGE report!)
  MOBILE AGENT agent:
  SEQ
    agent := MOBILE reindelf
    ...  initialise agent
    ...  loop (send agent; receive agent; run agent)
:
```



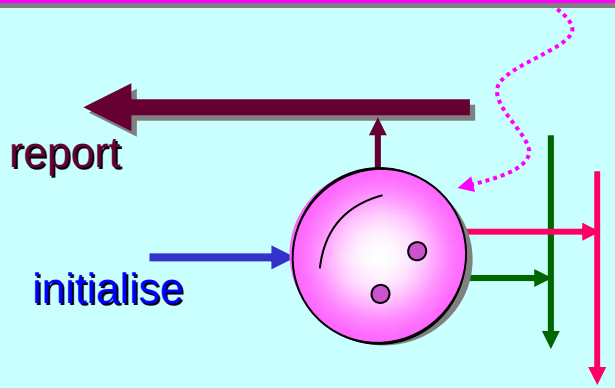
A reindelf : either a reindeer or an elf



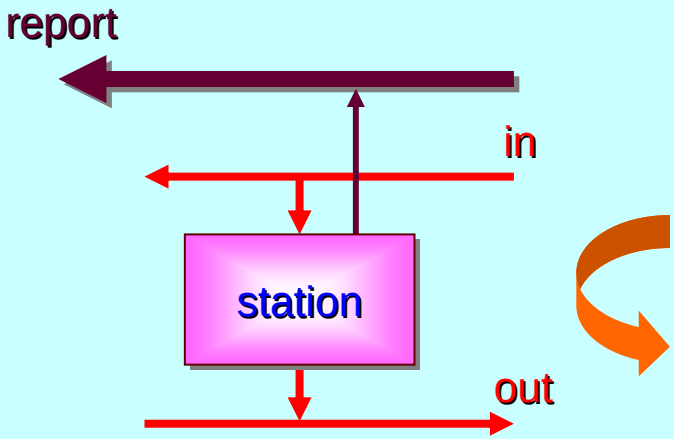

```
{{{  initialise agent
CHAN AGENT.INITIALISE initialise:
...  some dummy channels
PAR
  initialise ! id; seed; kind; away.time
  agent (initialise?, report!, ...)
}}}
```



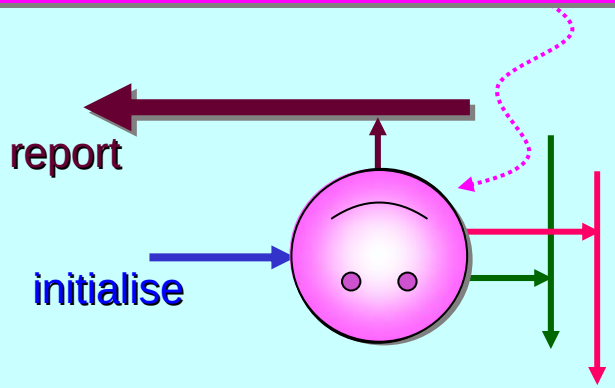
A reindelf : either a reindeer or an elf




```
PROC station (VAL INT id, seed, kind, away.time,
              SHARED CHAN MOBILE AGENT in?, out!,
              SHARED CHAN AGENT.MESSAGE report!)
  MOBILE AGENT agent:
  SEQ
    agent := MOBILE reindelf
    ...  initialise agent
    ...  loop (send agent; receive agent; run agent)
:
```

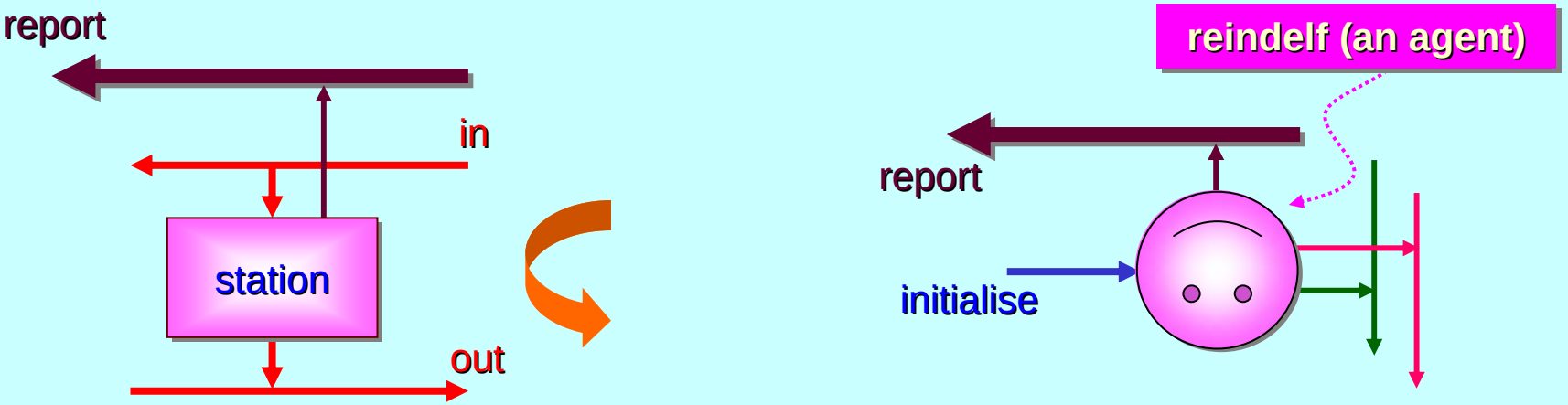


A reindelf : either a reindeer or an elf




```
{{{  loop (send agent; receive agent; run agent)
WHILE TRUE
  SEQ
    CLAIM out ! agent
    CLAIM in ? agent
    CHAN AGENT.INITIALISE dummy.init:
    ...  more dummy channels
    agent (dummy.init?, report!, ...)
}}}
```

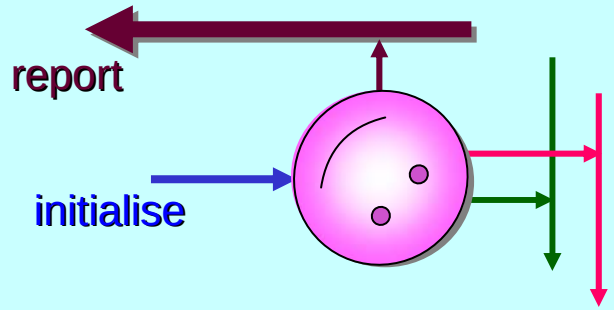
```
PROC TYPE AGENT IS (CHAN AGENT.INITIALISE initialise?,
                    SHARED CHAN AGENT.MESSAGE report!,
                    ... ):
```



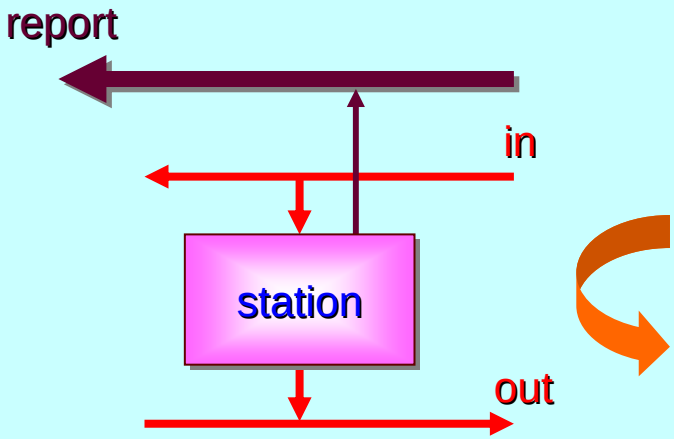

```
{{{  loop (send agent; receive agent; run agent)
WHILE TRUE
  SEQ
    CLAIM out ! agent
    CLAIM in ? agent
    CHAN AGENT.INITIALISE dummy.init:
    ...  more dummy channels
    agent (dummy.init?, report!, ...)
}}}
```

```
PROC TYPE AGENT IS (CHAN AGENT.INITIALISE initialise?,
                    SHARED CHAN AGENT.MESSAGE report!,
                    ... ):
```

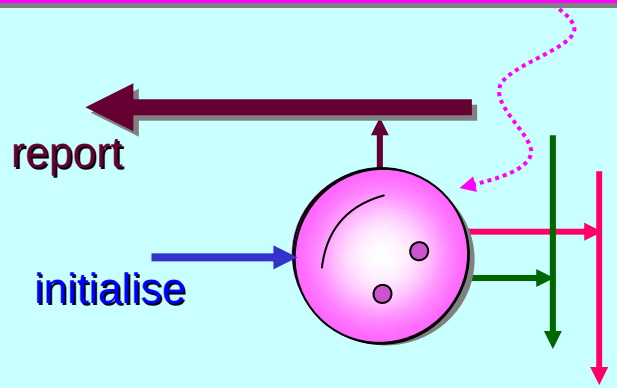
```
MOBILE PROC reindelf (CHAN AGENT.INITIALISE initialise?,
                      SHARED CHAN AGENT.MESSAGE report!,
                      ... ) IMPLEMENTS AGENT
  INT id, seed, kind, away.time:
  SEQ
    initialise ? id; seed; kind; away.time
    WHILE TRUE
      SEQ
        CLAIM report ! away; kind; id
        ...  away time (random delay up to away.time)
        CLAIM report ! ready; kind; id
        SUSPEND       -- move to gathering place
        ...
        SUSPEND       -- move to santa's grotto
        ...
        SUSPEND       -- move to compound
:
```



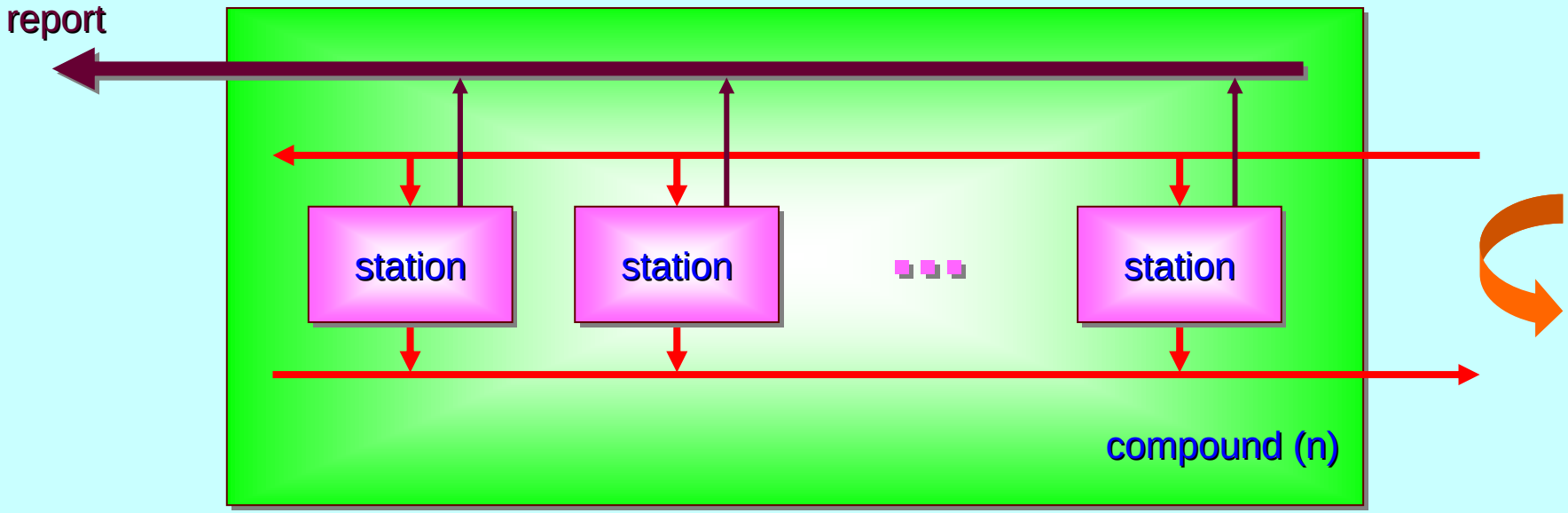

```
MOBILE PROC reindelf (CHAN AGENT.INITIALISE initialise?,
                      SHARED CHAN AGENT.MESSAGE report!,
                      ... ) IMPLEMENTS AGENT
  INT id, seed, kind, away.time:
  SEQ
    initialise ? id; seed; kind; away.time
    WHILE TRUE
      SEQ
        CLAIM report ! away; kind; id
        ...  away time (random delay up to away.time)
        CLAIM report ! ready; kind; id
        SUSPEND       -- move to gathering place
        ...
        SUSPEND       -- move to santa's grotto
        ...
        SUSPEND       -- move to compound
:
```

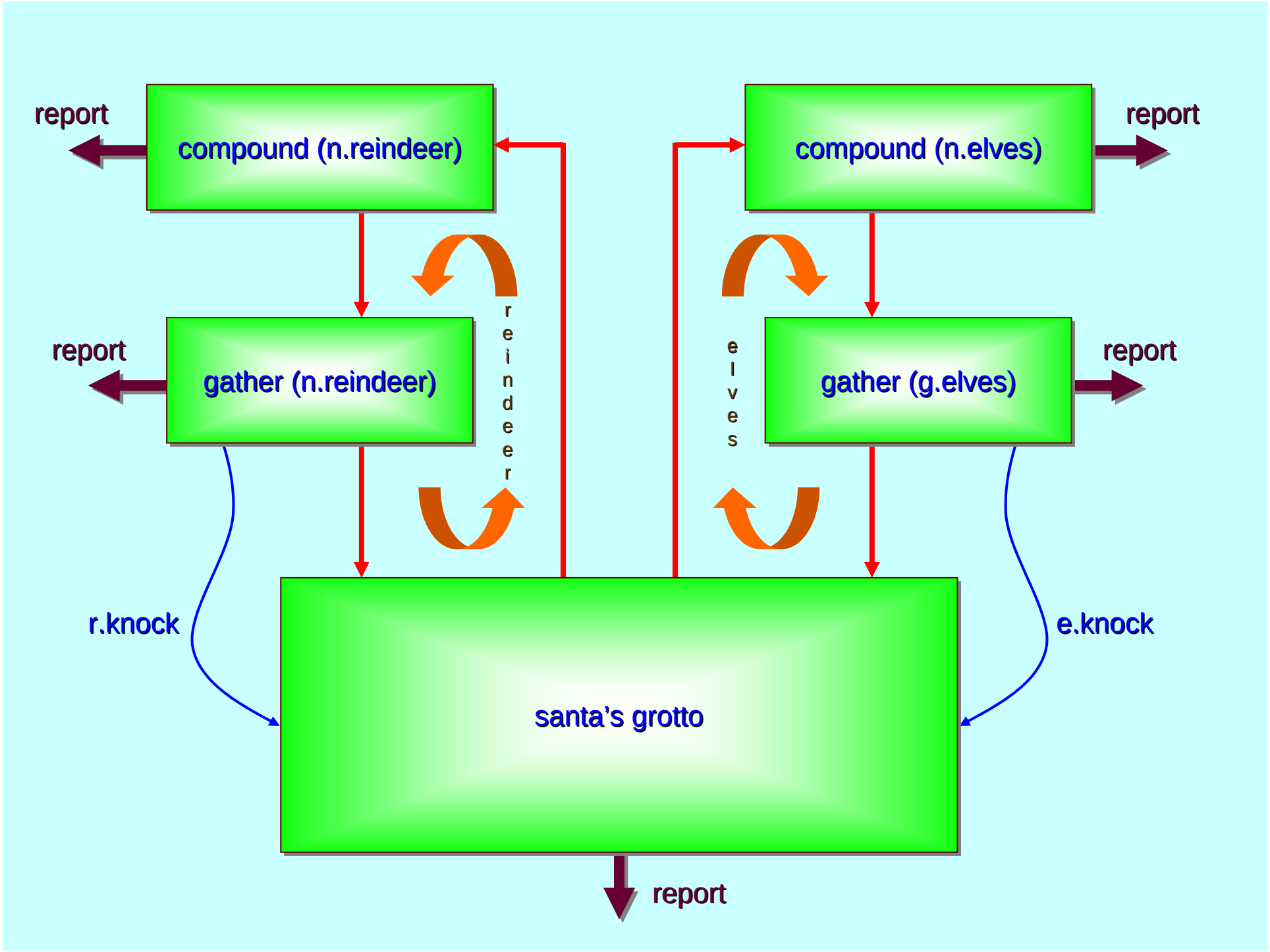


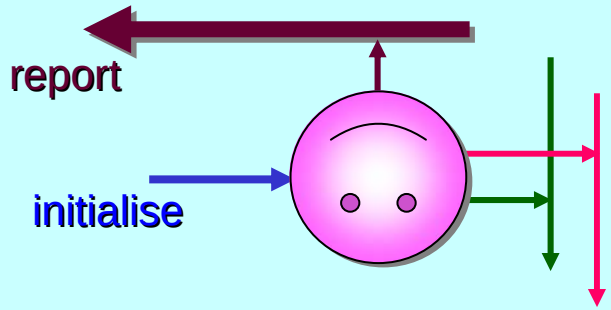
A reindelf : either a reindeer or an elf

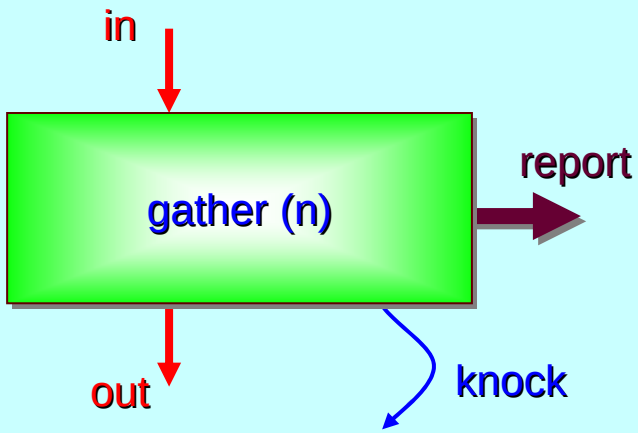



```
PROC station (VAL INT id, seed, kind, away.time,
              SHARED CHAN MOBILE AGENT in?, out!,
              SHARED CHAN AGENT.MESSAGE report!)
  MOBILE AGENT agent:
  SEQ
    agent := MOBILE reindelf
    ...  initialise agent
    ...  loop (send agent; receive agent; run agent)
:
```

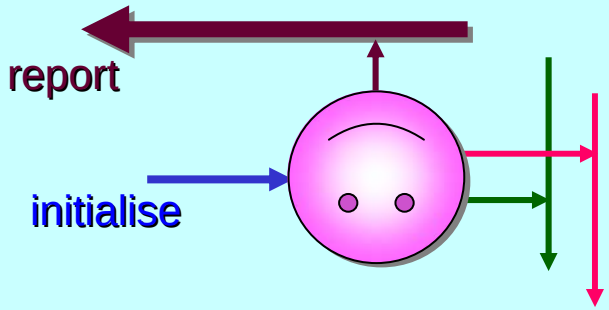

report
station
station
station
compound (n)

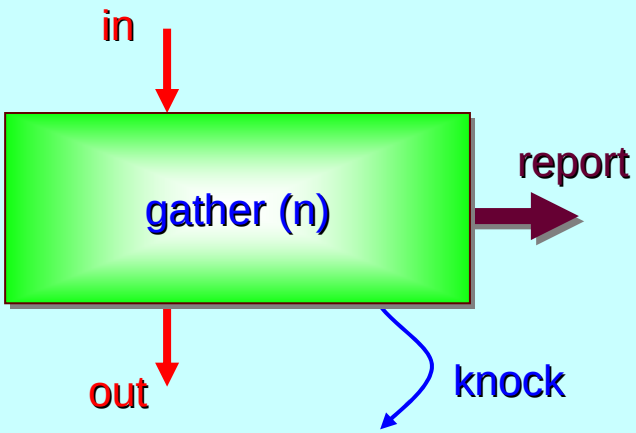

report
compound (n.reindeer)
compound (n.elves)
report
report
gather (n.reindeer)
reindeer
elves
gather (g.elves)
report
r.knock
e.knock
santa's grotto
report



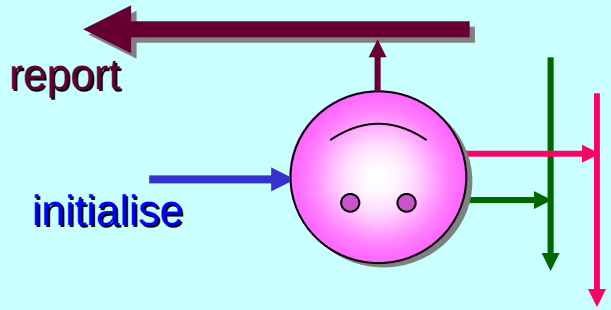


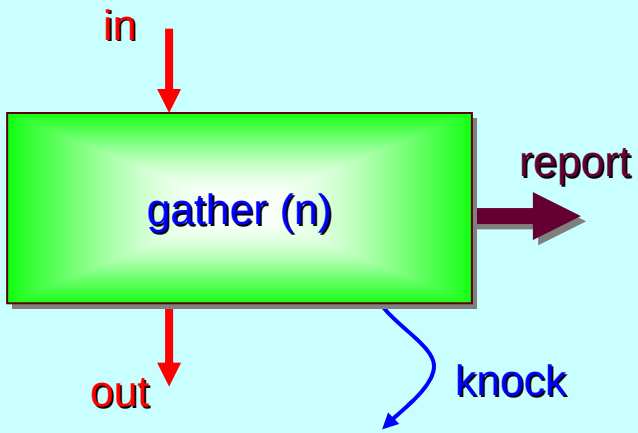

```
PROC gather (VAL INT n, CHAN MOBILE AGENT in?, out!,
             SHARED CHAN AGENT.MESSAGE report!,
             CHAN BOOL knock!)
  WHILE TRUE
    [n]MOBILE AGENT agent:
    SEQ
      SEQ i = 0 FOR n
        SEQ
          in ? agent[i]
          ...  plug in agent (let it make brief report)
      knock ! TRUE          -- knock on santa's door
      SEQ i = 0 FOR n
        out ! agent[i]
      knock ! TRUE          -- wait for door to slam
:
```



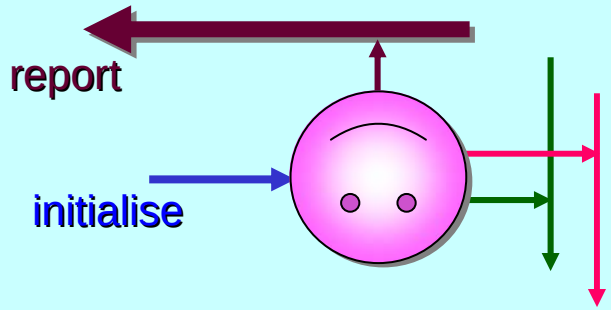


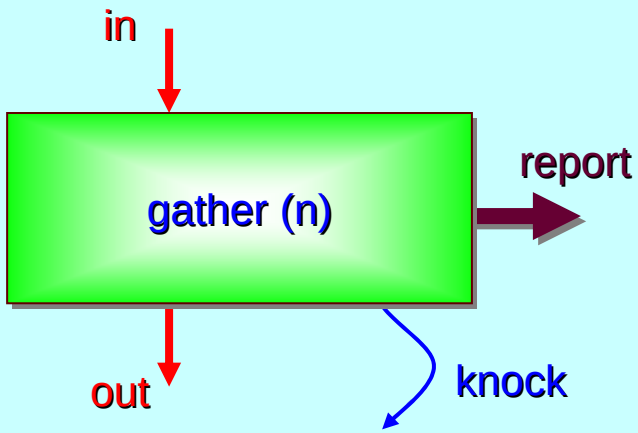

```
{{{  plug in agent (let it make brief report)

CHAN AGENT.INITIALISE dummy.init:
...  more dummy channels
agent[i] (dummy.init?, report!, ...)

}}}
```



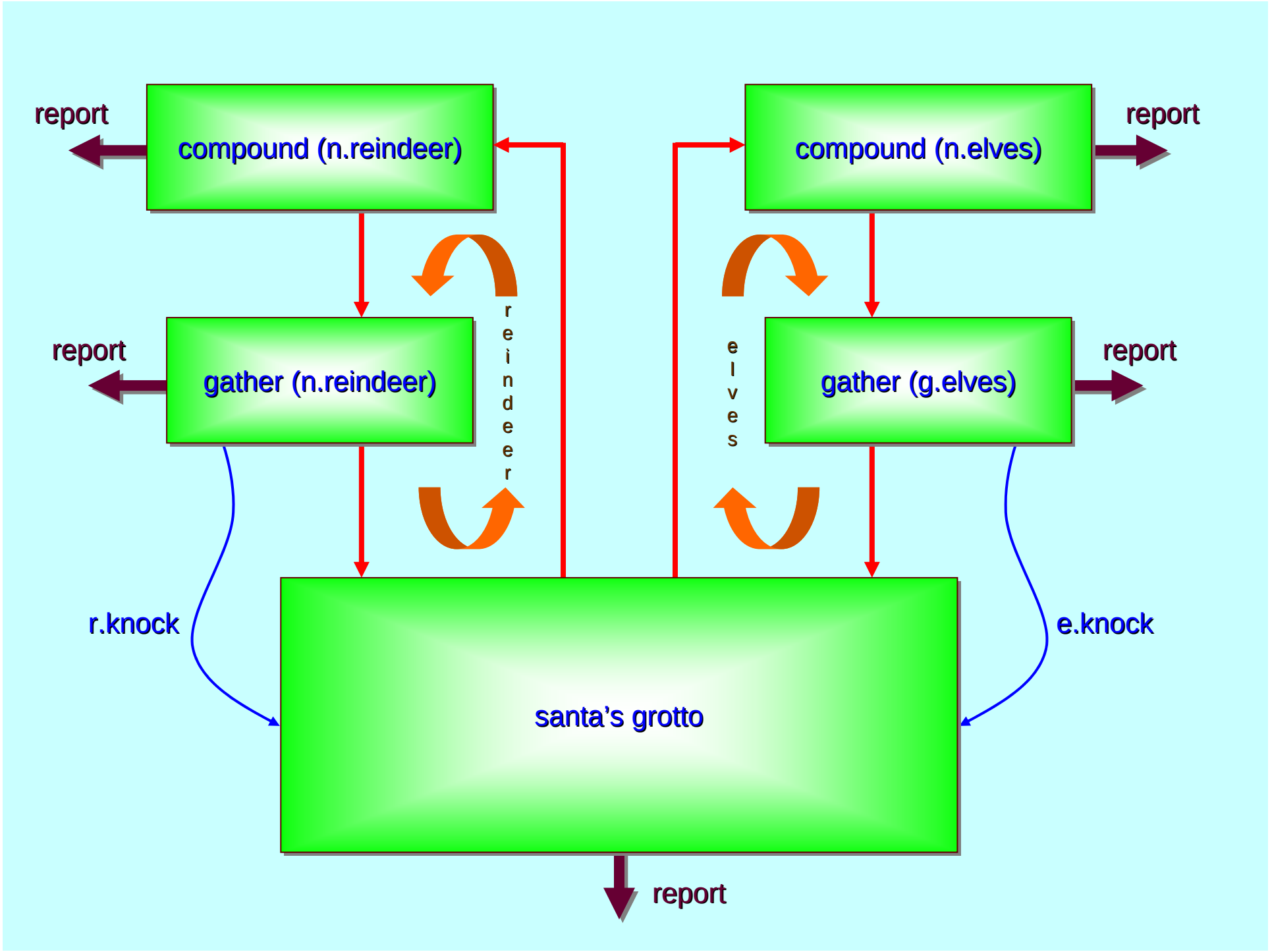


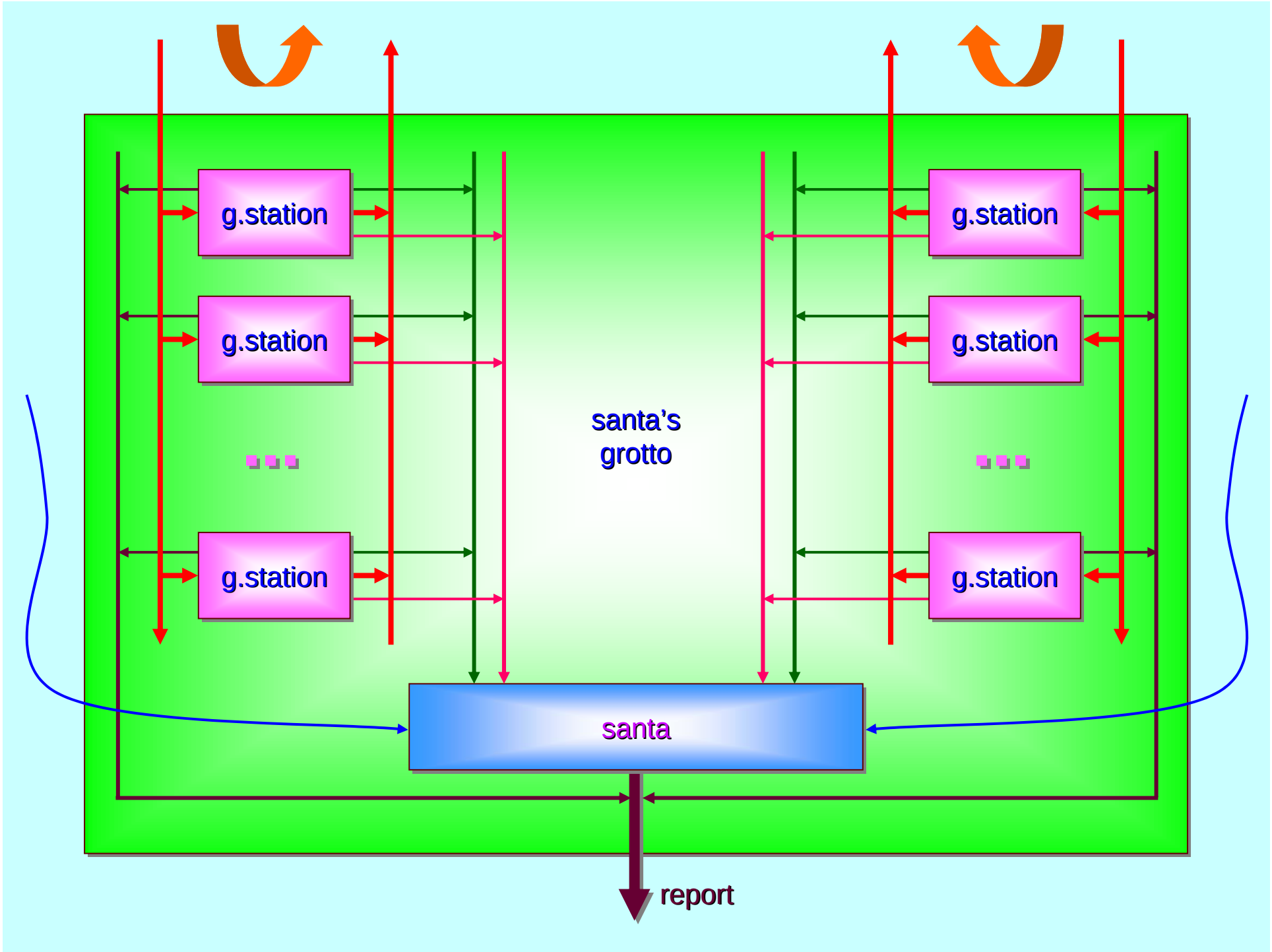

```
MOBILE PROC reindelf (CHAN AGENT.INITIALISE initialise?,
                      SHARED CHAN AGENT.MESSAGE report!,
                      ... ) IMPLEMENTS AGENT
  ...  local state declarations
  SEQ
    ...  in station compound (initialise local state)
    WHILE TRUE
      SEQ
        ...  in station compound
        SUSPEND       -- move to gathering place
        CLAIM report ! waiting; kind; id
        SUSPEND       -- move to santa's grotto
        ...  in santa's grotto
        SUSPEND       -- move to compound
:
```

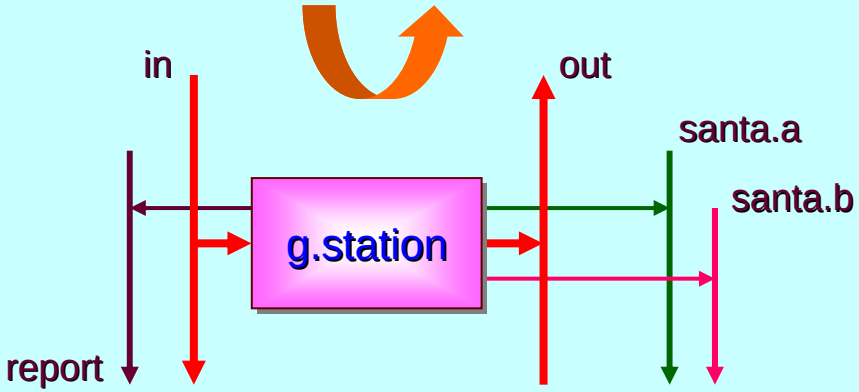


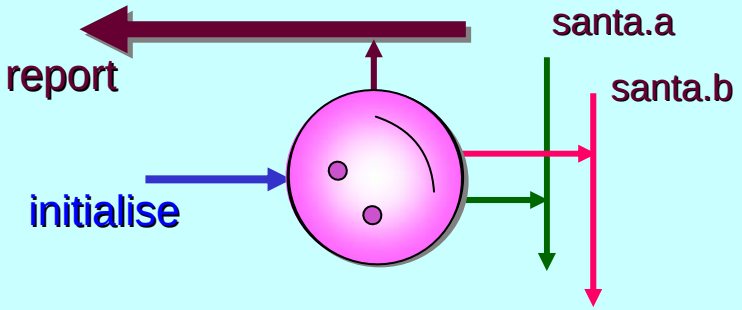



```
PROC gather (VAL INT n, CHAN MOBILE AGENT in?, out!,
             SHARED CHAN AGENT.MESSAGE report!,
             CHAN BOOL knock!)
  WHILE TRUE
    [n]MOBILE AGENT agent:
    SEQ
      SEQ i = 0 FOR n
        SEQ
          in ? agent[i]
          ...  plug in agent (let it make brief report)
      knock ! TRUE          -- knock on santa's door
      SEQ i = 0 FOR n
        out ! agent[i]
      knock ! TRUE          -- wait for door to slam
:
```


report
compound (n.reindeer)
compound (n.elves)
report
report
gather (n.reindeer)
reindeer
elves
gather (g.elves)
report
r.knock
e.knock
santa's grotto
report

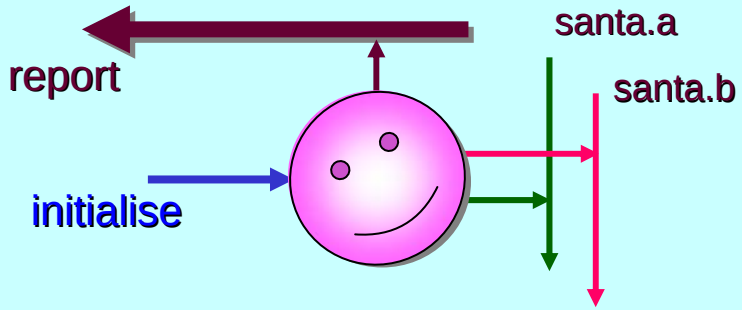

g.station
g.station
g.station
g.station
g.station
g.station
santa's
grotto
santa
report



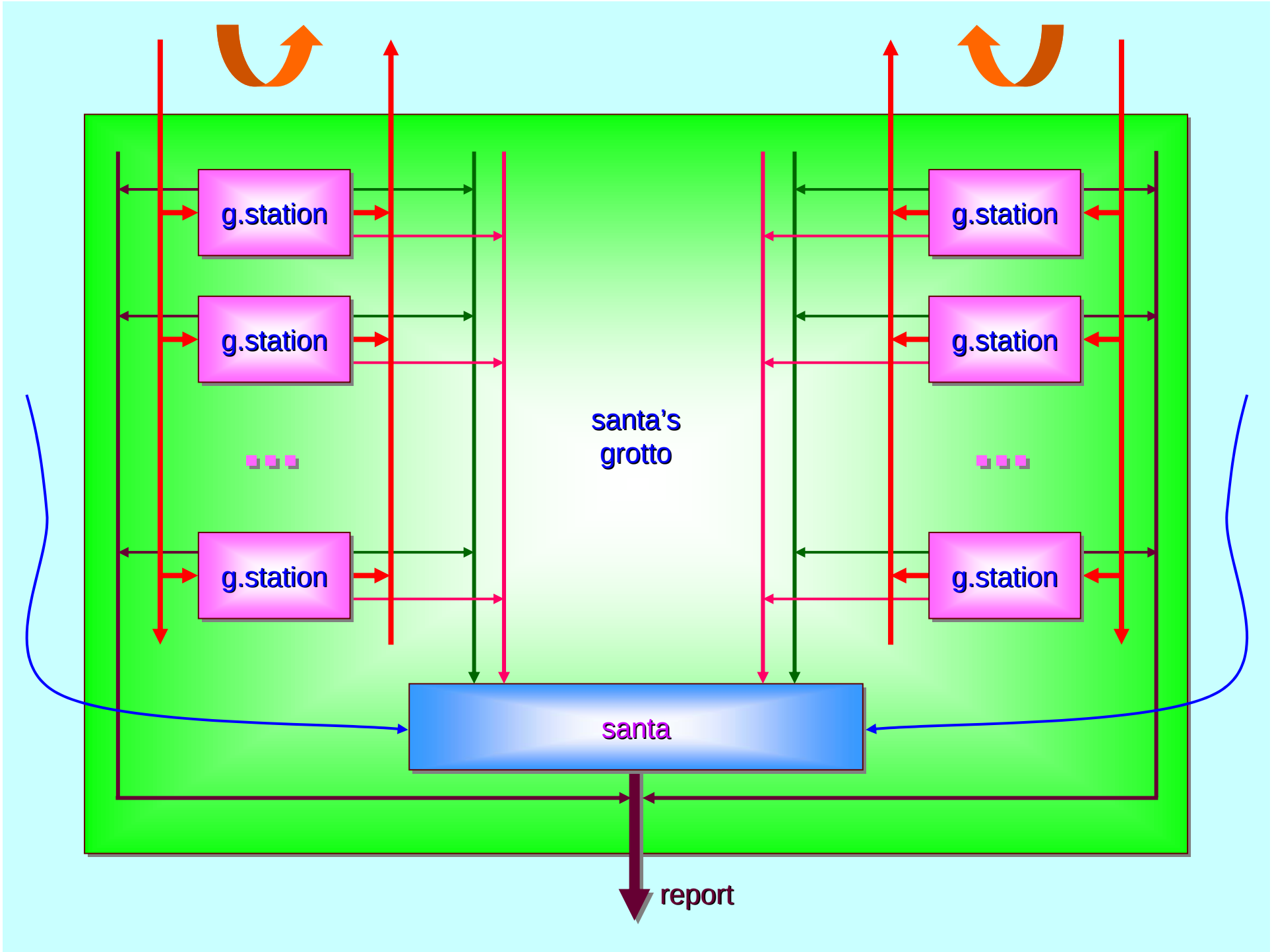



```
PROC grotto.station (SHARED CHAN MOBILE AGENT in?, out!,
                     SHARED CHAN AGENT.MESSAGE report!,
                     SHARED CHAN INT santa.a!, santa.b!)
  WHILE TRUE
    MOBILE AGENT agent:
    SEQ
      CLAIM in ? agent
      CHAN AGENT.INITIALISE dummy.init:
      agent (dummy.init?, report!, santa.a!, santa.b!)
      CLAIM out ! agent
:
```

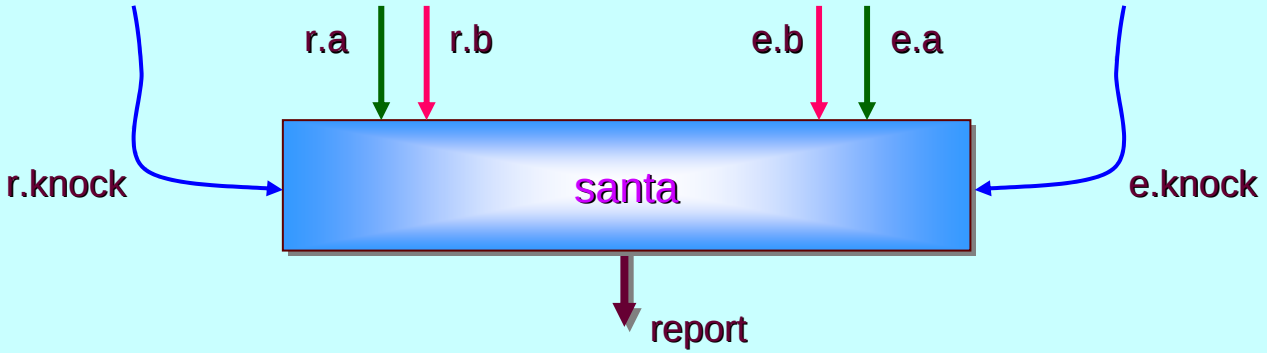
```
PROC TYPE AGENT IS (CHAN AGENT.INITIALISE initialise?,
                    SHARED CHAN AGENT.MESSAGE report!,
                    SHARED CHAN INT santa.a!, santa.b!):
```



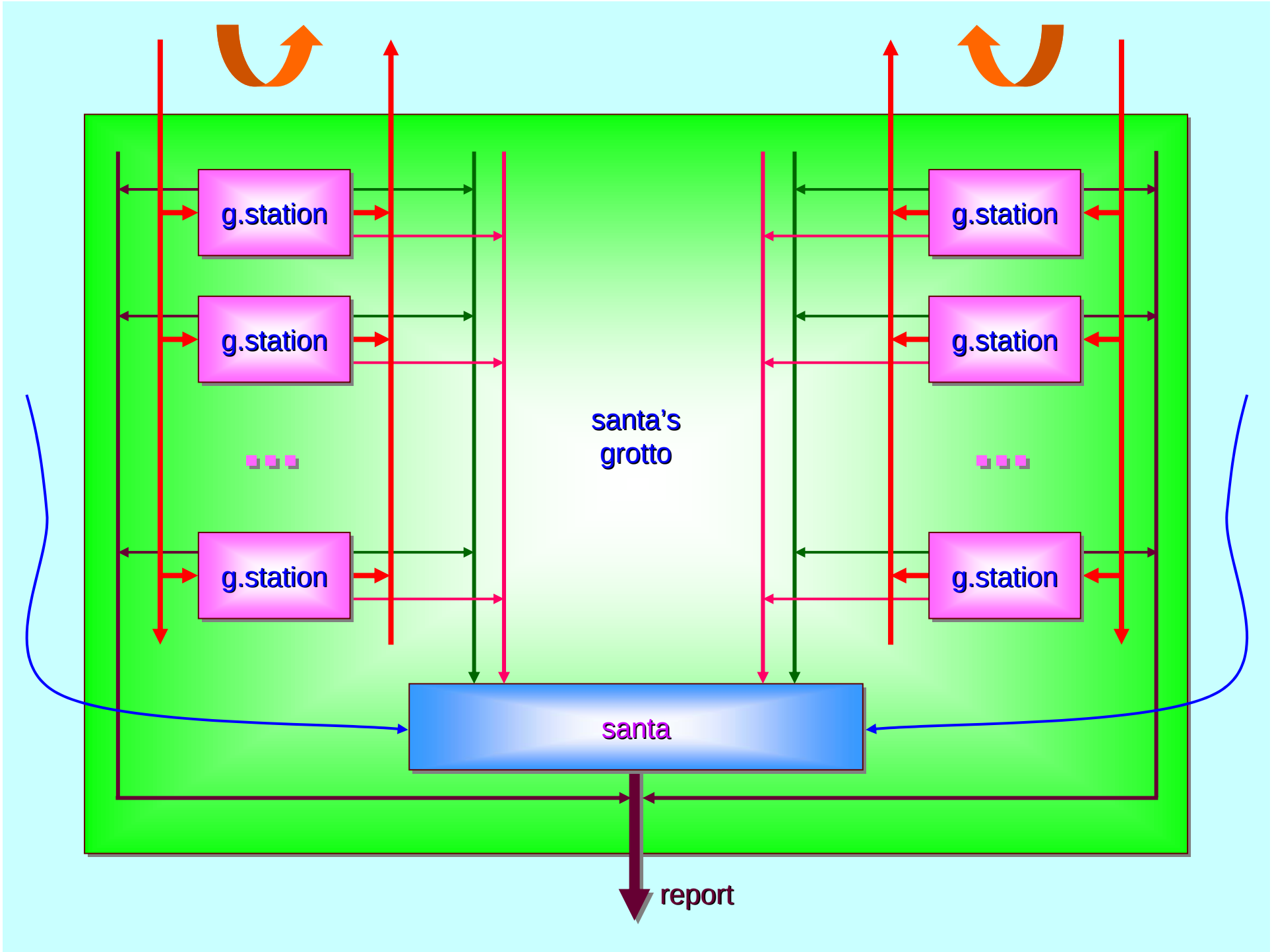

```
MOBILE PROC reindelf (CHAN AGENT.INITIALISE initialise?,
                      SHARED CHAN AGENT.MESSAGE report!,
                      SHARED CHAN INT santa.a!, santa.b!)
                                          IMPLEMENTS AGENT
  ...
:
```

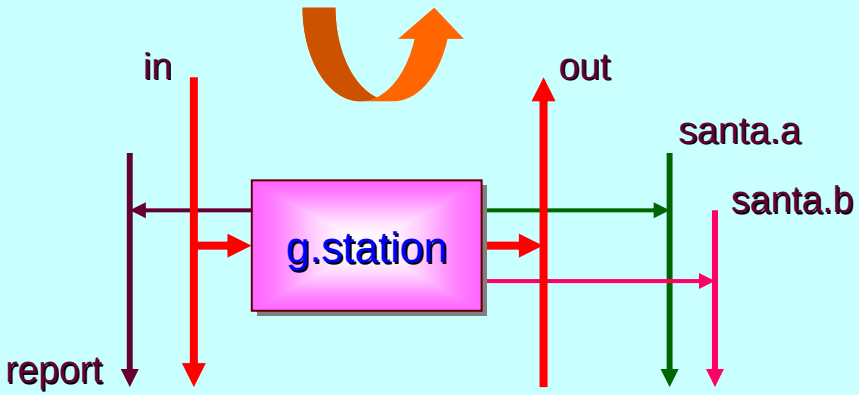

g.station
g.station
g.station
g.station
g.station
g.station
santa's
grotto
santa
report

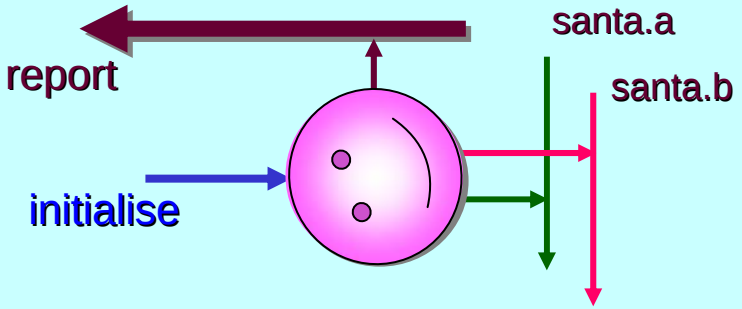
```
PROC santa (CHAN INT elf.a?, elf.b?,
            CHAN INT reindeer.a?, reindeer.b?,
            CHAN BOOL elf.knock?, reindeer.knock?,
            SHARED CHAN SANTA.MESSAGE report)
  WHILE TRUE
    BOOL any:
    PRI ALT
      reindeer.knock ? any
        SEQ
          CLAIM report ! agent.ready; REINDEER.KIND
          ...  engage with reindeer
      elf.knock ? any
        SEQ
          CLAIM report ! agent.ready; ELF.KIND
          ...  engage with elves
:
```



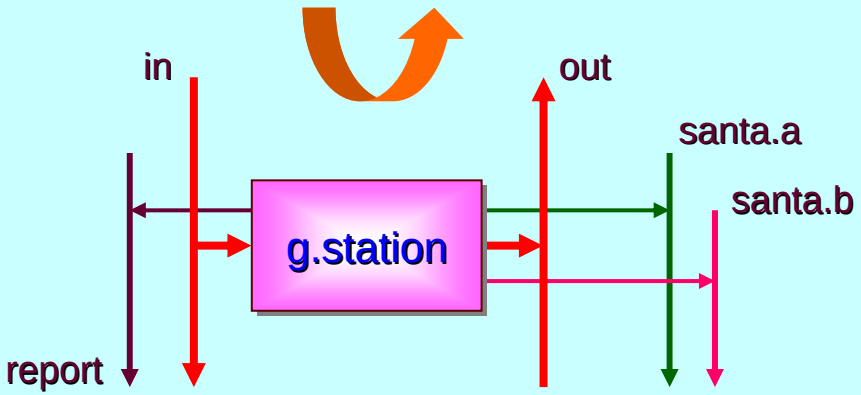
```
PROC engage (VAL INT group.size, kind,
             CHAN INT agent.a?, agent.b?,
             CHAN BOOL knock?,
             SHARED CHAN SANTA.MESSAGE report!)
  INT id:
  BOOL any:
  SEQ
    SEQ i = 0 FOR group.size
      SEQ
        agent.a ? id
        CLAIM report ! greet; kind; id
    knock ? any          -- slam the door
    SEQ i = 0 FOR group.size
      agent.b ? id
    CLAIM report ! engaged; kind
    ...  pause for a (random) while
    CLAIM report ! disengaged; kind
    SEQ i = 0 FOR group.size
      agent.a ? id
    SEQ i = 0 FOR group.size
      agent.b ?? id
        CLAIM report ! goodbye; kind, id
:
```

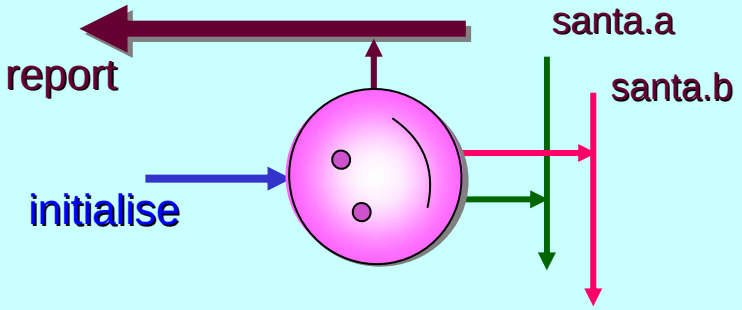

g.station
g.station
g.station
g.station
g.station
g.station
santa's
grotto
santa
report



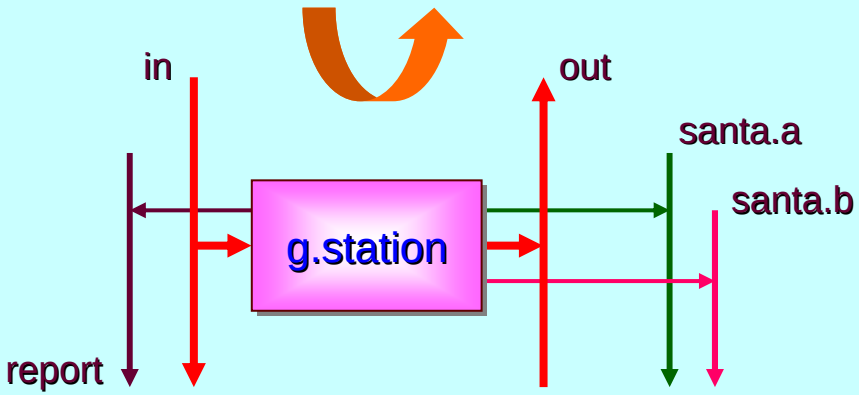


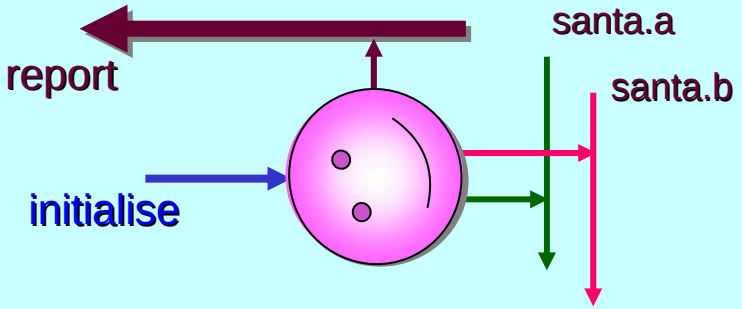

```
PROC grotto.station (SHARED CHAN MOBILE AGENT in?, out!,
                     SHARED CHAN AGENT.MESSAGE report!,
                     SHARED CHAN INT santa.a!, santa.b!)
  WHILE TRUE
    MOBILE AGENT agent:
    SEQ
      CLAIM in ? agent
      CHAN AGENT.INITIALISE dummy.init:
      agent (dummy.init?, report!, santa.a!, santa.b!)
      CLAIM out ! agent
:
```



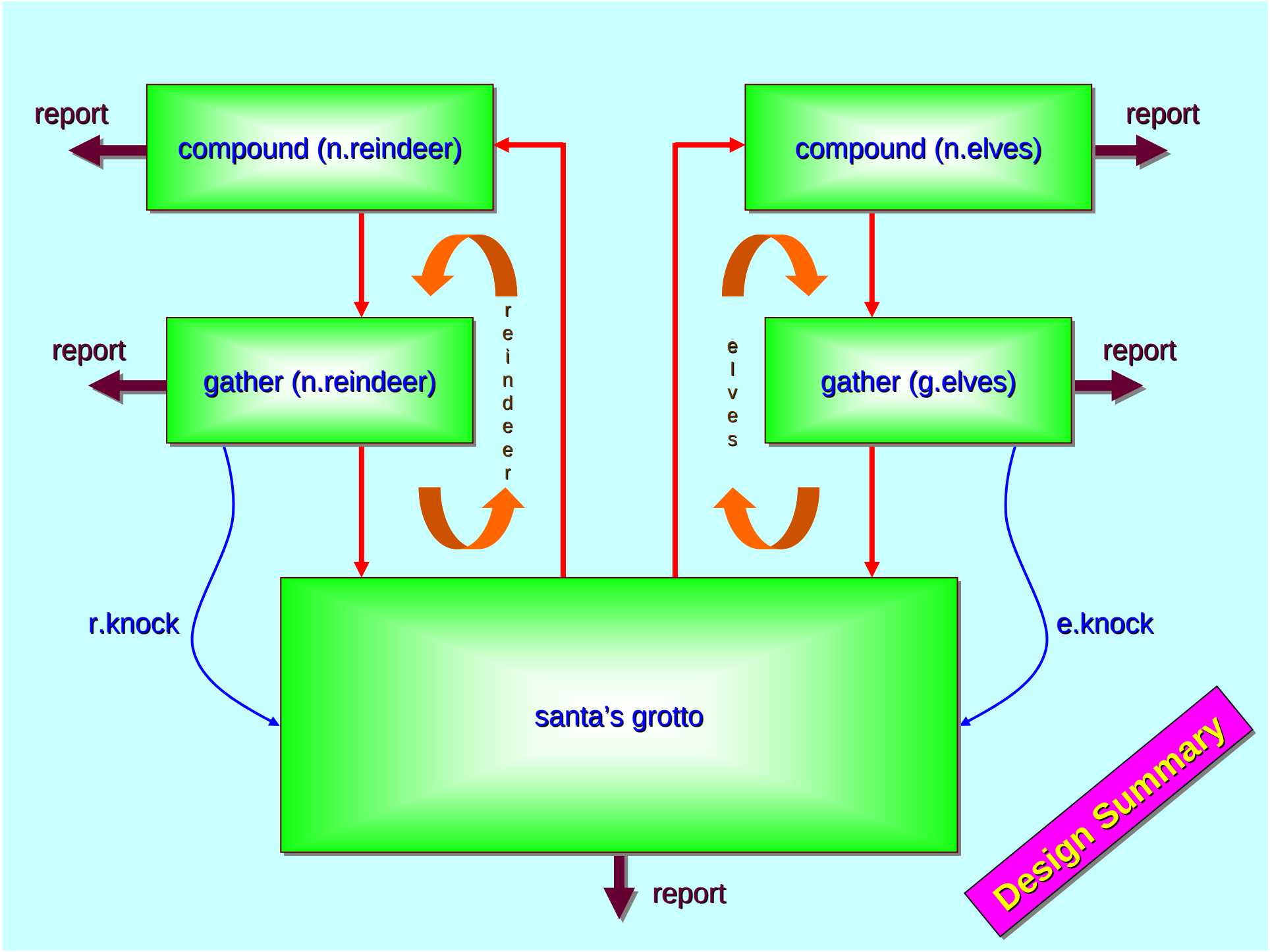


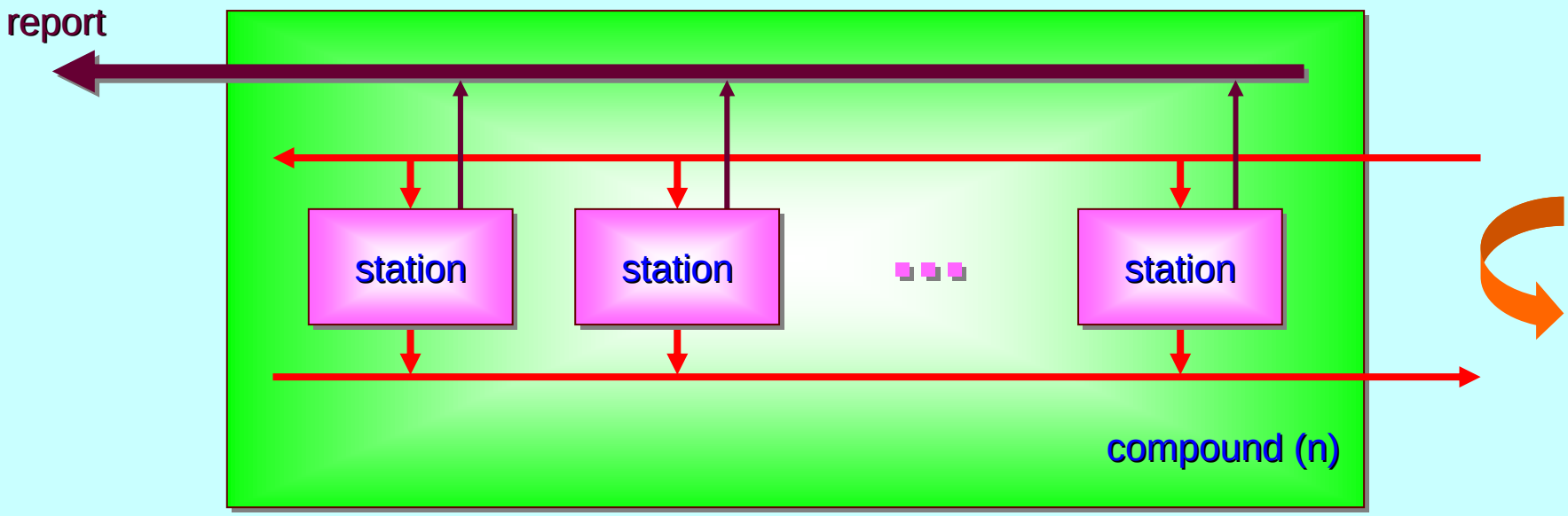

```
MOBILE PROC reindelf (CHAN AGENT.INITIALISE initialise?,
                      SHARED CHAN AGENT.MESSAGE report!,
                      SHARED CHAN INT santa.a!, santa.b!)
                                          IMPLEMENTS AGENT
  ...  local state declarations
  SEQ
    ...  in station compound (initialise local state)
    WHILE TRUE
      SEQ
        ...  in station compound
        SUSPEND       -- move to gathering place
        ...  in the gathering place
        SUSPEND       -- move to santa's grotto
        ...  in santa's grotto
        SUSPEND       -- move to compound
:
```



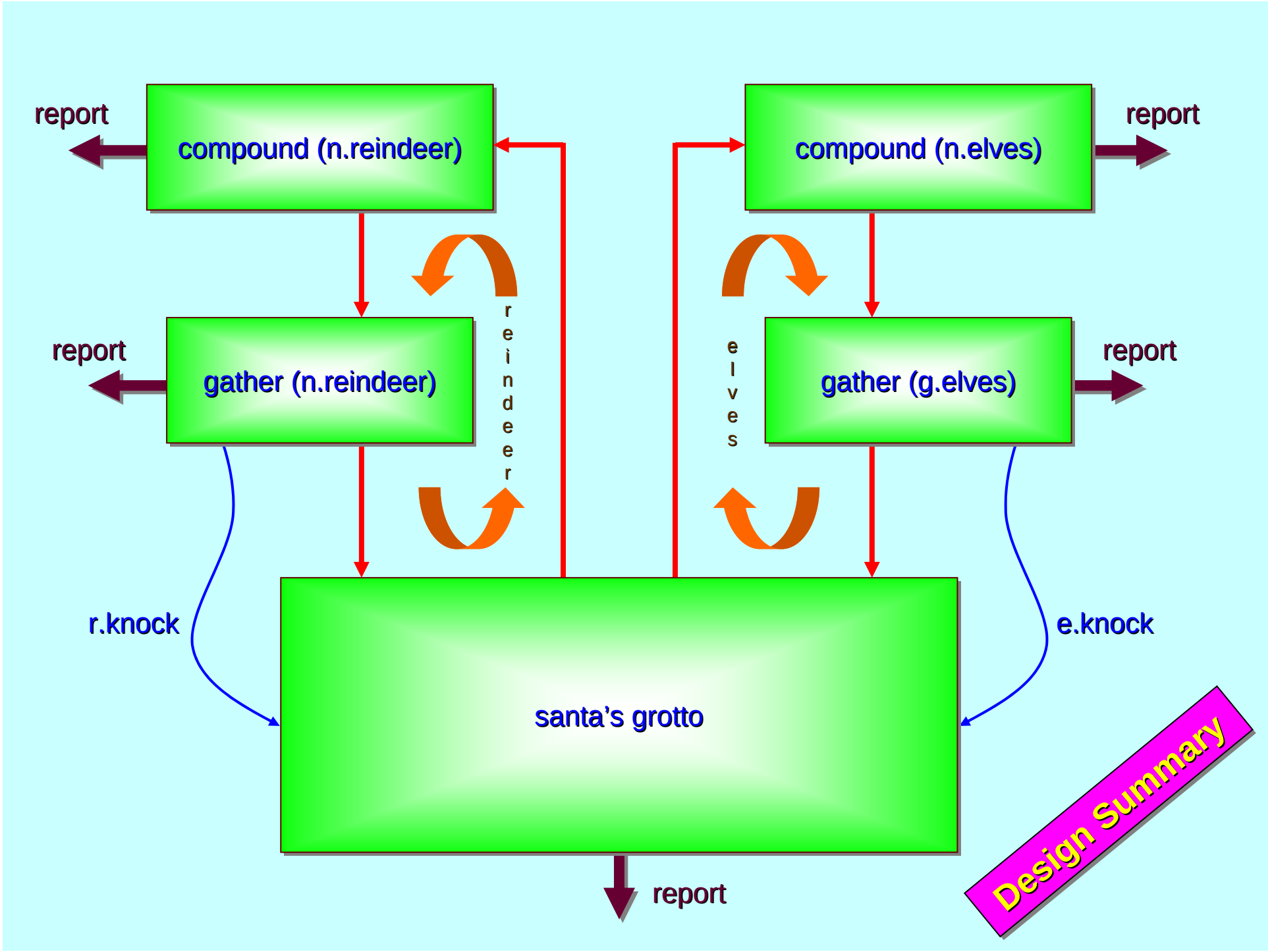


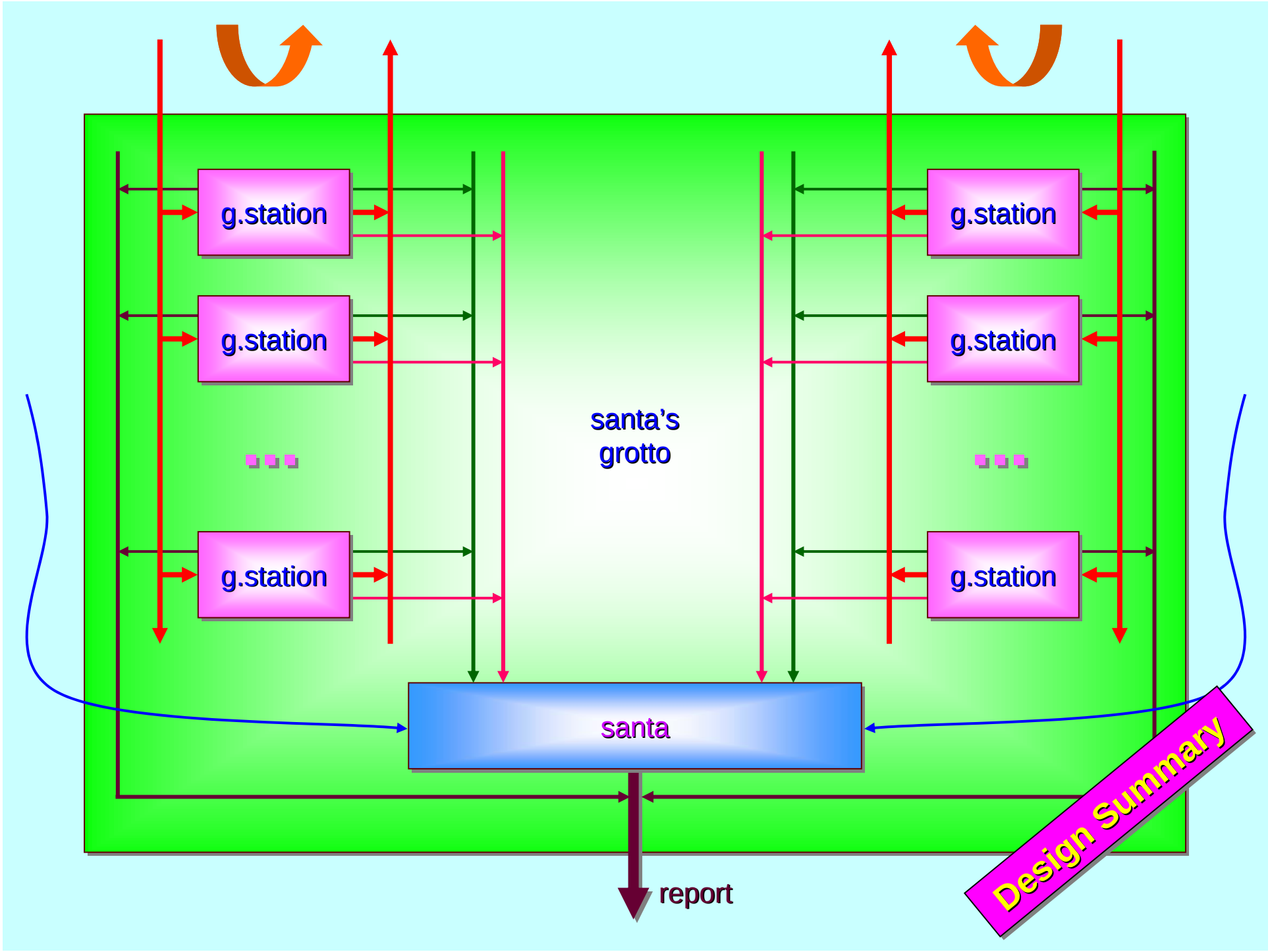

```
{{{  in santa's grotto
CLAIM santa.a ! id                  -- say hello to santa
CLAIM santa.b ! id                  -- sync with other agents
                                    -- and santa

CLAIM report ! busy; kind; id
CLAIM santa.a ! id                  -- wait for santa to finish
                                    -- working with me

CLAIM report ! done; kind; id
CLAIM santa.b ! id                  -- say goodbye to santa
}}}
```


report
compound (n.reindeer)
compound (n.elves)
report
report
gather (n.reindeer)
reindeer
elves
gather (g.elves)
report
r.knock
e.knock
santa's grotto
report
Design Summary

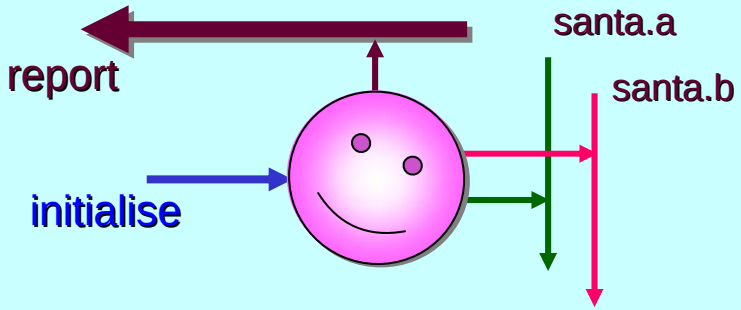


Design Summary


report
compound (n.reindeer)
compound (n.elves)
report
report
gather (n.reindeer)
reindeer
elves
gather (g.elves)
report
r.knock
e.knock
santa's grotto
report
Design Summary


g.station
g.station
g.station
g.station
g.station
g.station
santa's
grotto
santa
report
Design Summary

either a reindeer or an elf




```
MOBILE PROC reindelf (CHAN AGENT.INITIALISE initialise?,
                      SHARED CHAN AGENT.MESSAGE report!,
                      SHARED CHAN INT santa.a!, santa.b!)
                                          IMPLEMENTS AGENT
  ...  local state declarations
  SEQ
    ...  in station compound (initialise local state)
    WHILE TRUE
      SEQ
        ...  in station compound
        SUSPEND       -- move to gathering place
        ...  in the gathering place
        SUSPEND       -- move to santa's grotto
        ...  in santa's grotto
        SUSPEND       -- move to compound
:
```

Design Summary


Any Questions?