

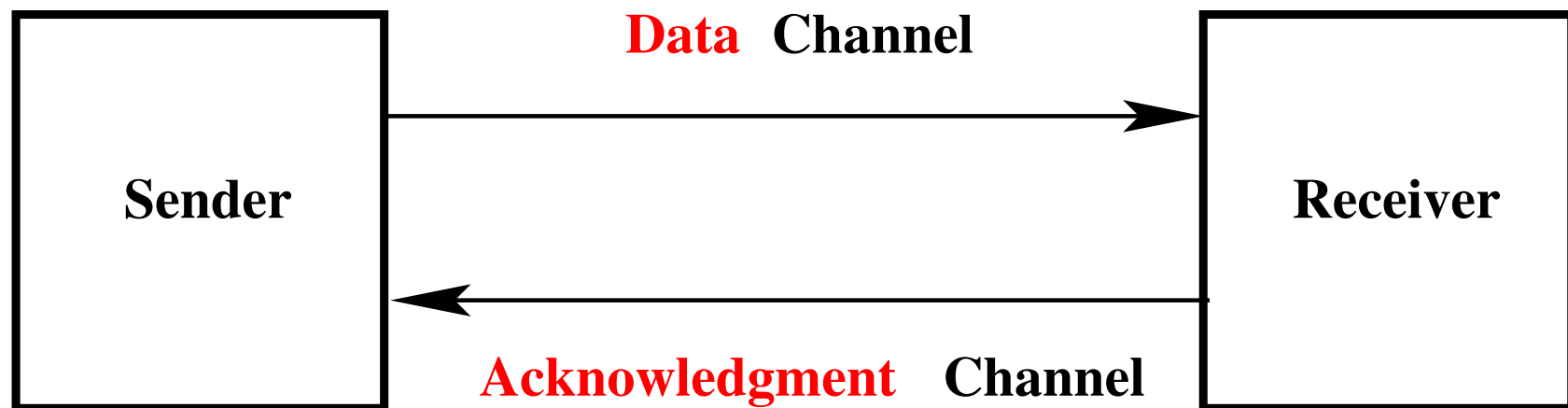
# 4. The Bounded Re-transmission Protocol

Jean-Raymond Abrial

March 2008

- The Bounded Re-transmission Protocol is a **file transfer protocol**
- This is a problem dealing with **fault tolerance**
- We suppose that the transfer channels are **unreliable**
- We present classical solutions to handle that problem: **timers**.
- We would like to see how we can **formalize such timers**

- A **sequential file** is transmitted from a **Sender** to a **Receiver**
- The file is transmitted **piece by piece** through a **Data Channel**
- After receiving some data, the Receiver sends an **acknowledgment**
- After receiving it, the Sender sends the **next piece of data**, etc.



- **Messages can be lost** in the Data or Acknowledgment channels

The goal of the BRP is to **totally** or **partially** transfer a certain non-empty original sequential file from one site to another.

FUN\_1

A **total transfer** means that the transmitted file is a **copy** of the original one.

FUN\_2

A **partial transfer** means that the transmitted file is a **genuine prefix** of the original one.

FUN\_3

- Messages can be lost in the Data or Acknowledgment channels
- The Sender starts a timer before sending a piece of data
- The timer wakes up the Sender after a delay  $dl$
- This occurs if the Sender has not received an acknowledgment in the meantime

- $dl$  is guaranteed to be **greater than twice the transmission time**
- When waken up, the Sender is then **sure** that the **data** or the **acknowledgment** has been **lost**
- When waken up, the Sender **re-transmits the previous data**
- The Sender sends an **alternating bit** together with a **new data**
- This ensures that the Receiver **does not confuse** (?) a **new data** with a **retransmitted** one.

- The Sender can re-transmit the same data **at most  $MAX + 1$  times**
- After this, the Sender **decides to abort**
- **How does the Receiver know** that the Sender aborted?

- Each time the Receiver receives a **new** piece of data, it **starts a timer**
- The timer **wakes up** the Receiver after a delay  $(MAX + 1) \times dl$
- This occurs if the Sender **has not received a new data** in the meantime.
- After this delay, the Receiver is certain that **the Sender has aborted**
- Then the **Receiver aborts** too.



- At the end of the protocol, we might be in one of the **three situations**:
  - (1) The file **has been transmitted** entirely and the Sender **has received** the last acknowledgment
  - (2) The file **has been transmitted** entirely but the Sender **has not received** the last acknowledgment
  - (3) The file **has not been transmitted** entirely

Each site may end up in any of the **two situations**:

- either **it believes** that the protocol has **terminated successfully**,
- or **it believes** that the protocol has **aborted**

FUN\_4

When the **Sender** believes that the protocol has **terminated successfully** then the **Receiver** believes so too.

FUN\_5

However, it is possible for the **Sender** to believe that the protocol has **aborted** while the **Receiver** believes that it has **terminated successfully**.

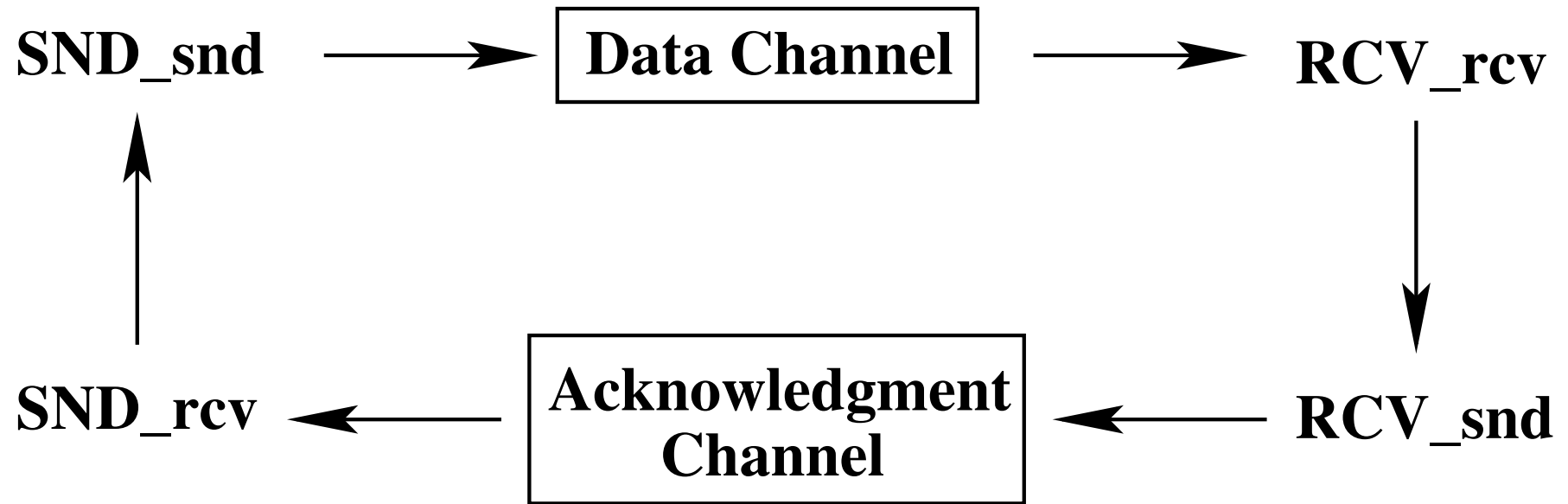
FUN\_6

When the **Receiver** believes that the protocol has **terminated successfully**, this is because the original file has been **entirely copied** on the Receiver's site.

FUN\_7

When the **Receiver** believes that the protocol has **aborted**, this is because the original file has **not been copied entirely** on the Receiver's site.

FUN\_8



SND\_snd

**when**

SND\_snd is waken up

**then**

Acquire data from Sender's file;

Store acquired data on Data Channel;

Store Sender's bit on Data Channel;

Start Sender's timer;

Activate Data Channel;

**end**

```
RCV_rcv
  when
    Data Channel interrupt occurs
  then
    Acquire Sender's bit from Data Channel;
    if Sender's bit = Receiver's bit then
      Acquire Data from Data Channel;
      Store data on Receiver's file;
      Modify Receiver's bit;
      if data is not the last one then
        Start Receiver's timer;
      end
    end
  end
  Reset Data Channel Interrupt;
  Wake up RCV_snd;
end
```

```
RCV_snd  
  when  
    RCV_snd is waken up  
  then  
    Activate Acknowledgment Channel;  
  end
```

```
SND_rcv
  when
    Acknowledgment Channel interrupt occurs
  then
    Remove Data from Sender's file;
    Reset retry counter;
    Modify Sender's bit;
    Wake up event SND_snd;
    Reset Acknowledgment Channel interrupt;
    if Sender's file is not empty then
      Wake up event SND_snd
    end
  end
end
```



```
SND_timer
  when
    Sender's timer interrupt occurs
  then
    if retry counter is equal to MAX+1 then
      Abort protocol on Sender's site;
    else
      Increment retry counter;
      Wake up event SND_snd;
    end
  end
```

RCV\_timer

**when**

Receiver's timer interrupt occurs

**then**

Abort protocol on Receiver's site

**end**

- 
- Quite often, protocols are "specified" by such pseudo-codes
  - In fact, such a pseudo-code raises a number of questions:
    - Are we sure that this description is correct?
    - Are we sure that this protocol terminates?
    - What kinds of properties should this protocol maintain?
  - Hence the formal development which is presented now

- (1) FUN\_1, FUN\_2, FUN\_3: **partial transmission** of the file **in one shot**.
- (2) FUN\_4 to FUN\_8: each participant has **access to the other**
- (3) Introducing **unreliable channels** and **timers**.
- (4) **Optimize** protocol

The goal of the BRP is to **totally** or **partially** transfer a certain non-empty original sequential file from one site to another.

FUN\_1

A **total transfer** means that the transmitted file is a **copy** of the original one.

FUN\_2

A **partial transfer** means that the transmitted file is a **genuine prefix** of the original one.

FUN\_3

Each site may end up in any of the **two situations**:

- either **it believes** that the protocol has **terminated successfully**,
- or **it believes** that the protocol has **aborted**

FUN\_4

When the **Sender** believes that the protocol has **terminated successfully** then the **Receiver** believes so too.

FUN\_5

However, it is possible for the **Sender** to believe that the protocol has **aborted** while the **Receiver** believes that it has **terminated successfully**.

FUN\_6

When the **Receiver** believes that the protocol has **terminated successfully**, this is because the original file has been **entirely copied** on the Receiver's site.

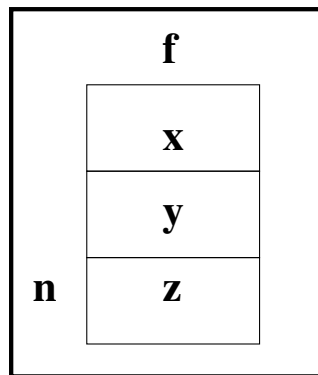
FUN\_7

When the **Receiver** believes that the protocol has **aborted**, this is because the original file has **not been copied entirely** on the Receiver's site.

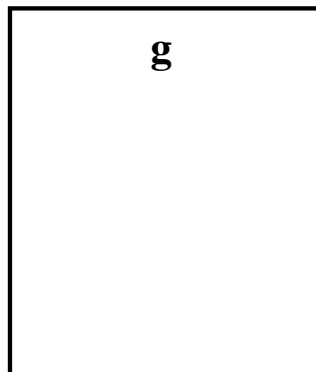
FUN\_8

## INITIAL SITUATION

**SENDER**

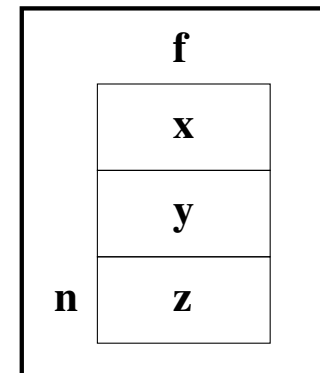


**RECEIVER**

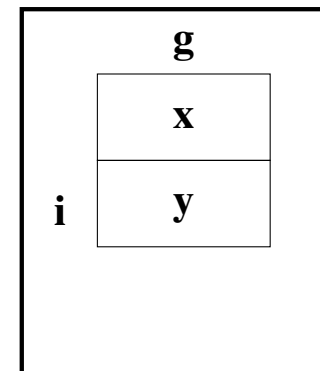


## FINAL SITUATION

**SENDER**



**RECEIVER**





- Set  $D$  denotes the objects in the files
- Constant  $n$  denotes the **size** of the **non-empty** file
- Constant  $f$  denotes the **original file**.

**set:**  $D$

**constants:**  $n$   
 $f$

**axm0\_1:**  $0 < n$

**axm0\_2:**  $f \in 1 .. n \rightarrow D$

- Variable  $i$  denotes the **size** of file  $g$
- Variable  $g$  denotes the **transmitted file**.

**variables:**  $i$   
 $g$

**inv0\_1:**  $i \in 0 .. n$

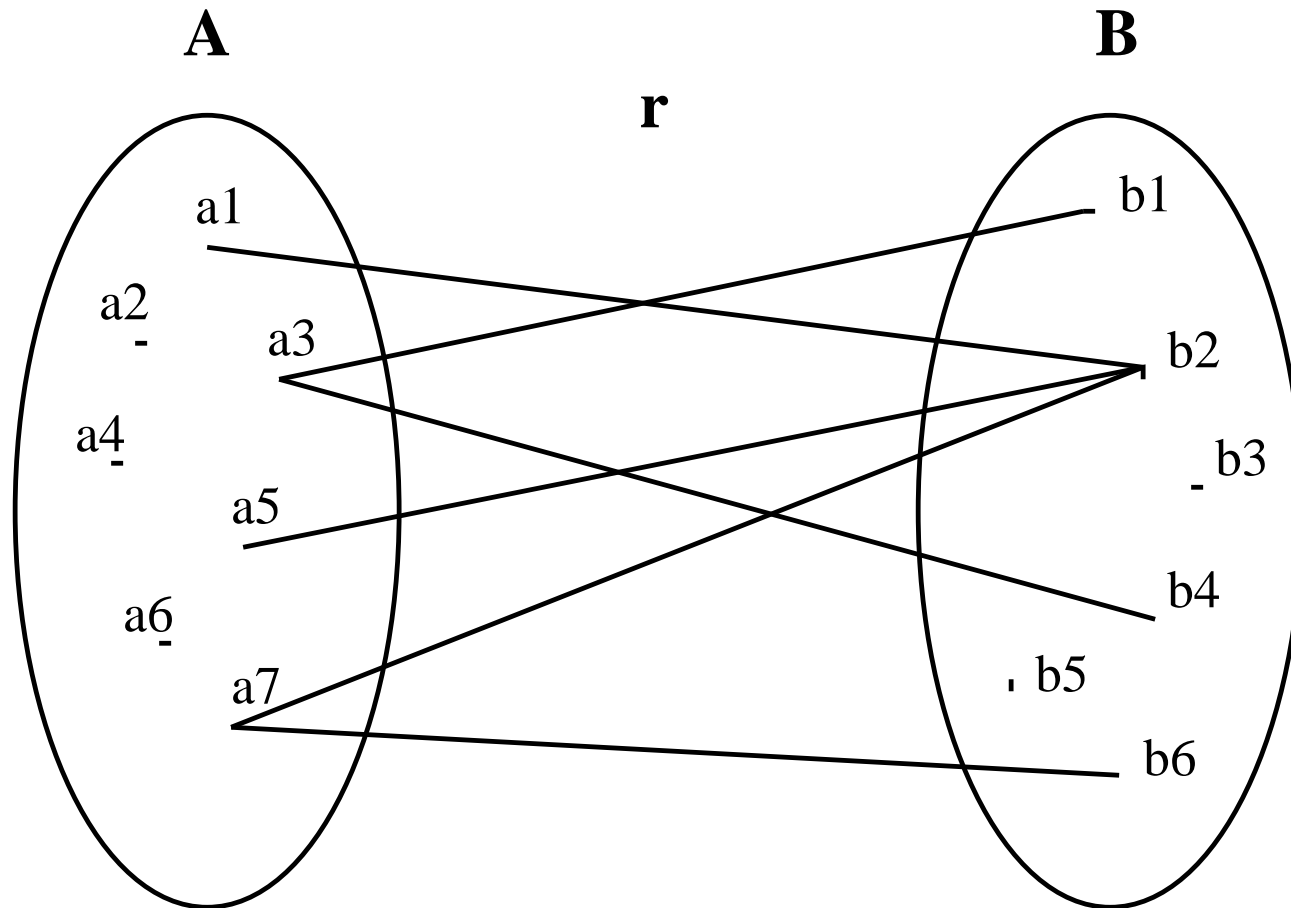
**inv0\_2:**  $g \in 1 .. i \rightarrow D$

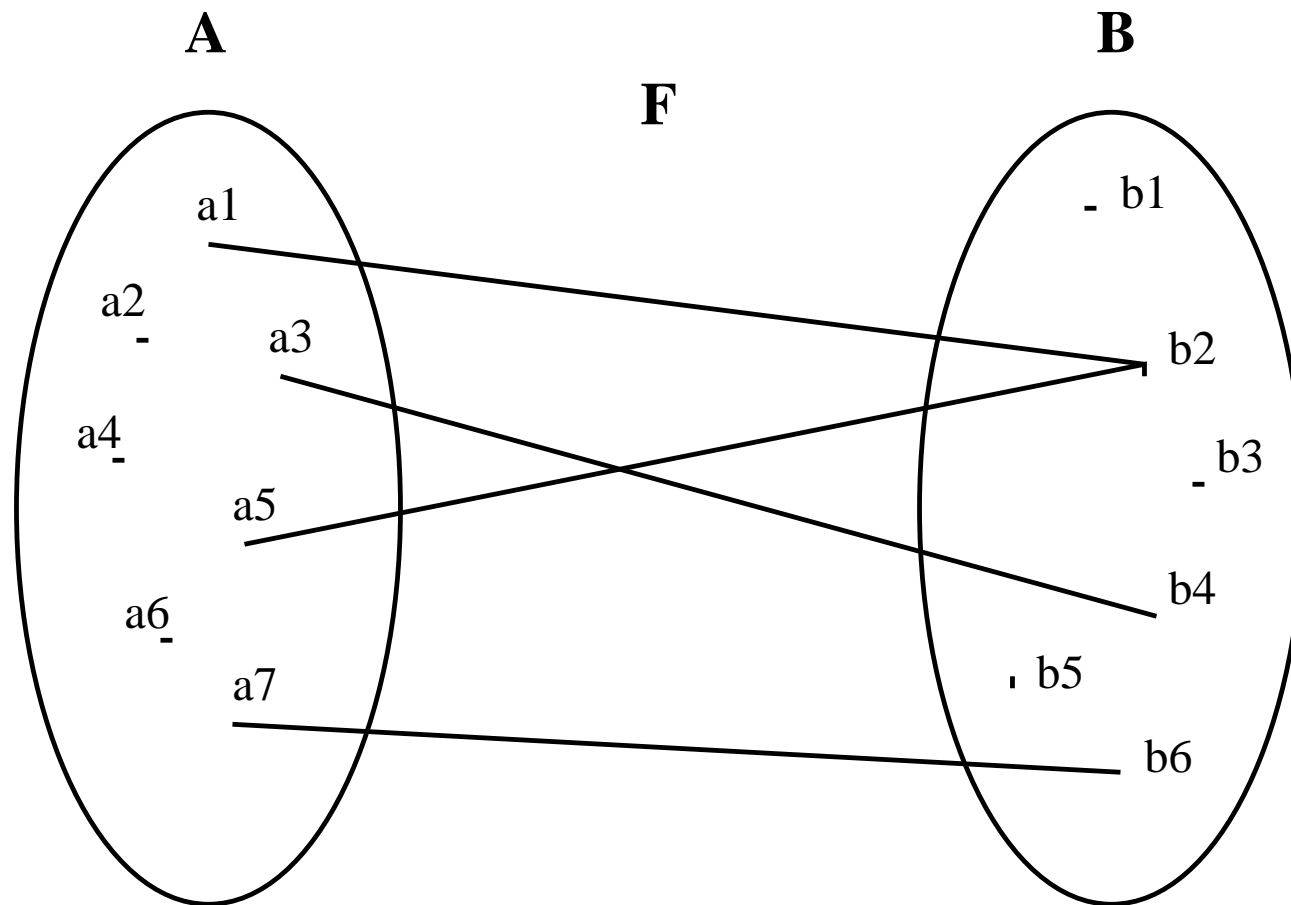
---

$x \in S$	set membership operator
$\mathbb{N}$	set of natural numbers: $\{0, 1, 2, 3, \dots\}$
$a .. b$	interval from $a$ to $b$ : $\{a, a + 1, \dots, b\}$ (empty when $b < a$ )
$a \mapsto b$	pair constructing operator
$S \times T$	Cartesian product operator
$S \subseteq T$	set inclusion operator
$\mathbb{P}(S)$	power set operator

---

$S \leftrightarrow T$	set of binary relations from $S$ to $T$
$S \rightarrow T$	set of total functions from $S$ to $T$
$S \twoheadrightarrow T$	set of partial functions from $S$ to $T$
$\text{dom}(r)$	domain of a relation $r$
$\text{ran}(r)$	range of a relation $r$

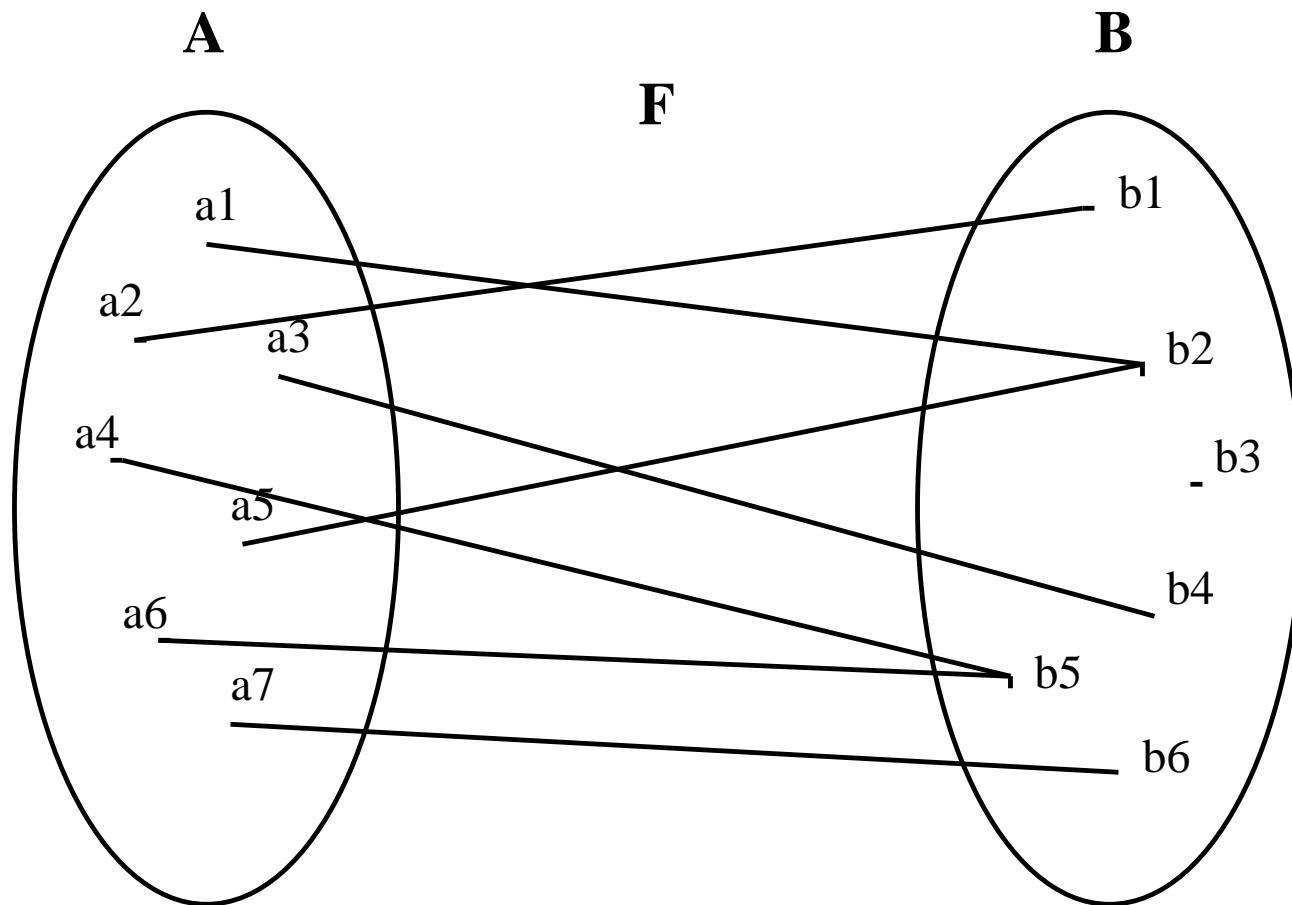




$$F = \{a_1 \mapsto b_2, a_3 \mapsto b_4, a_5 \mapsto b_2, a_7 \mapsto b_6\}$$

$$\text{dom}(F) = \{a_1, a_3, a_5, a_7\}$$

$$\text{ran}(F) = \{b_2, b_4, b_6\}$$



$$\text{dom}(F) = A$$

- Event **brp** describes the situation at the **end of the protocol**
- It only says that the file might be **partially transmitted**
- It is made of a **non-deterministic assignment**

init

$i := 0$

$g := \emptyset$

brp

$i, g :| i' \in 0 .. n \wedge$   
 $g' = (1 .. i') \triangleleft f$

- Operator **:|** is to be read: "**become such that ...**"



brp

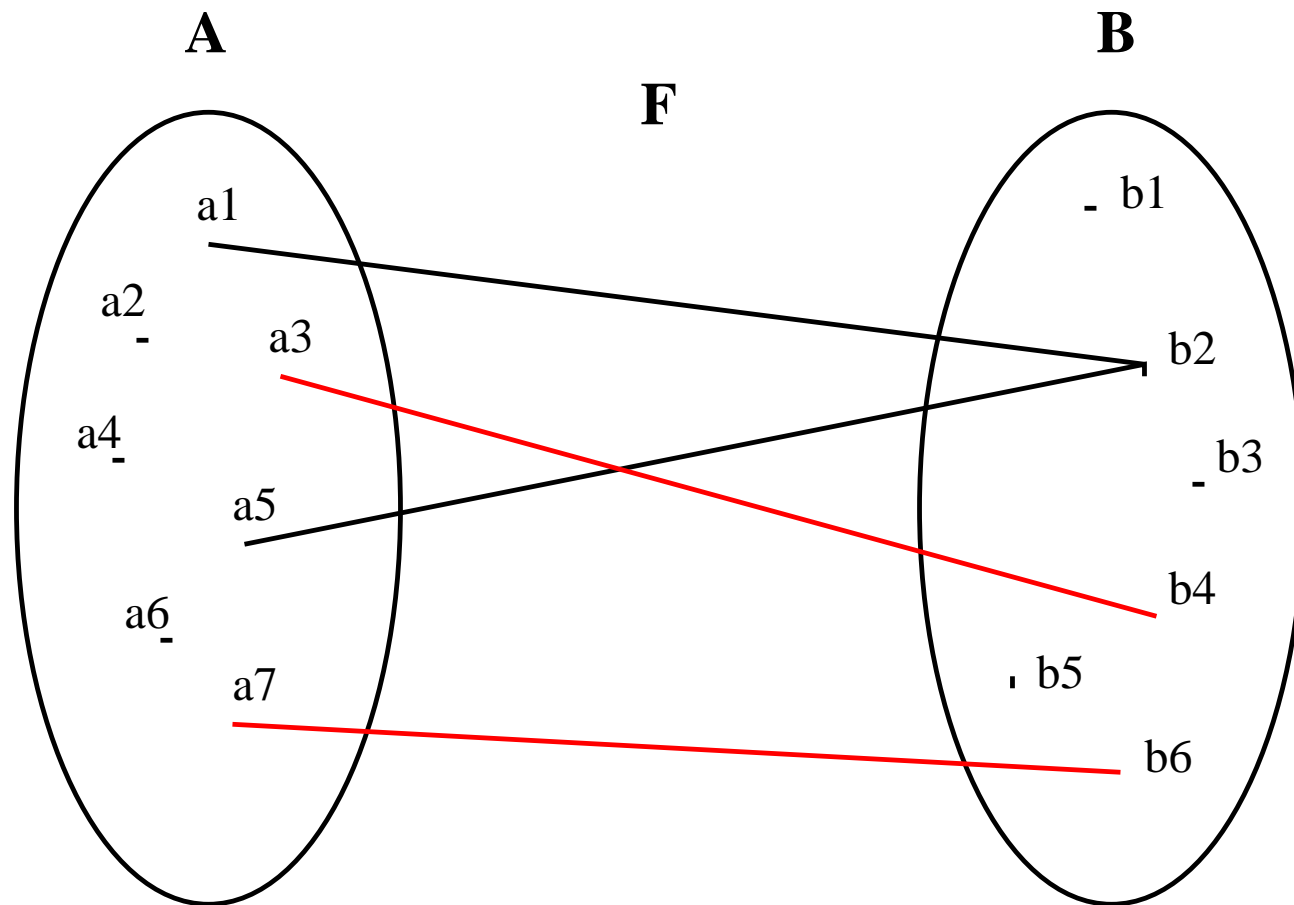
$$i, g : | \left( \begin{array}{l} i' \in 0 .. n \\ g' = (1 .. i') \triangleleft f \end{array} \right)$$

$i$  and  $g$  are assigned **any values  $i'$  and  $g'$**  such that the following holds:

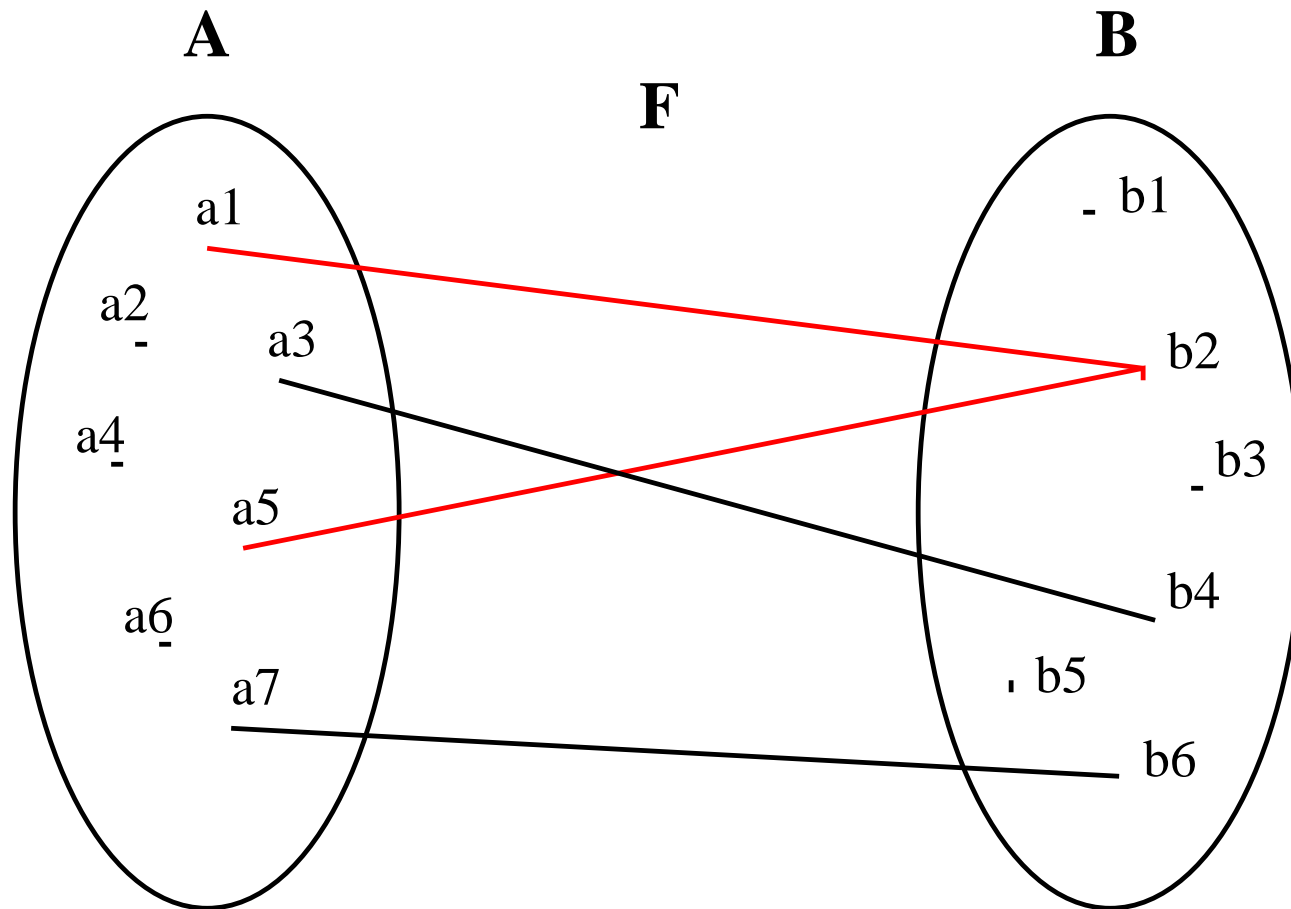
$$i' \in 0 .. n \wedge g' = (1 .. i') \triangleleft f$$

---

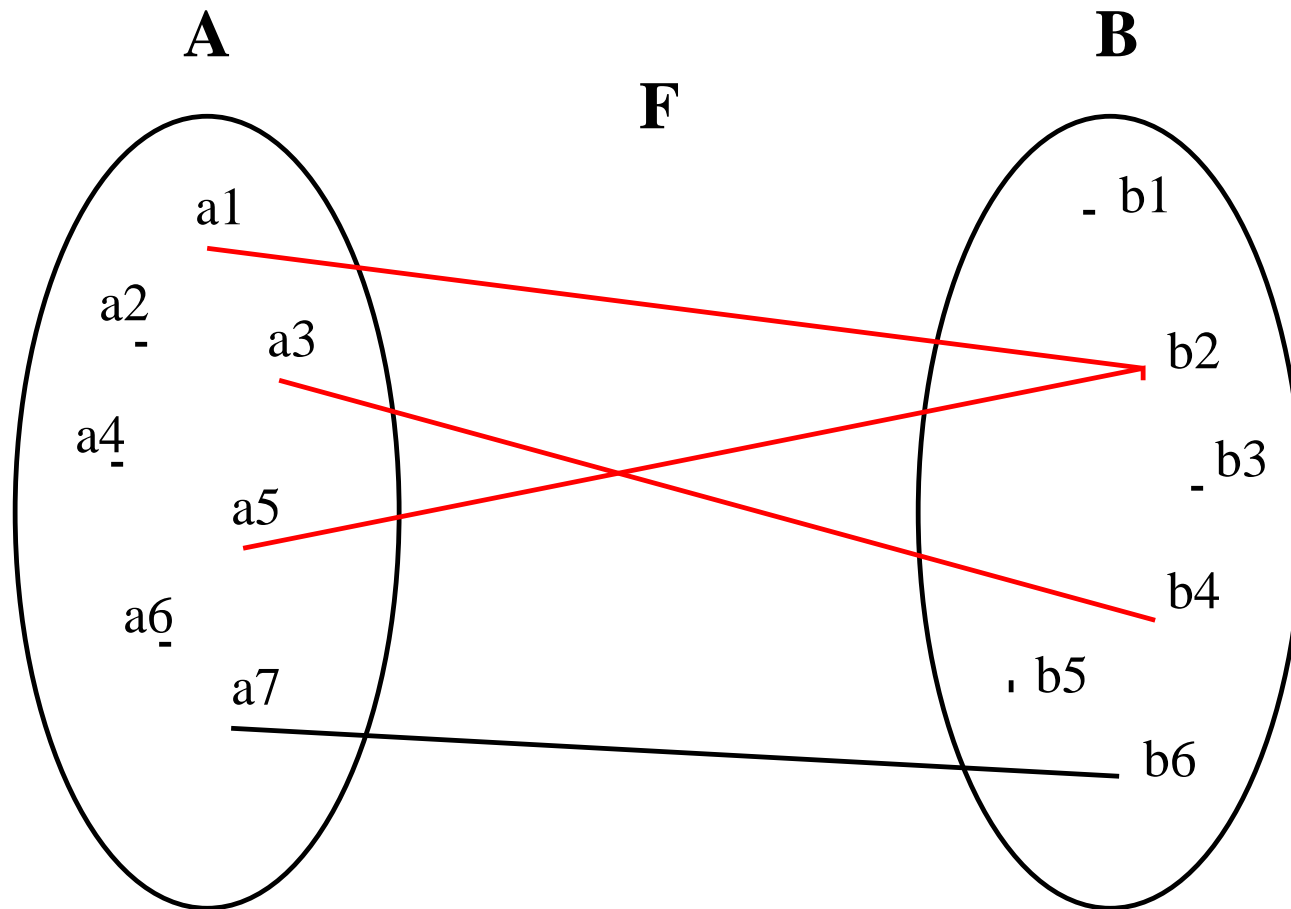
$s \triangleleft r$	domain restriction operator
$s \triangleleft r$	domain subtraction operator
$r \triangleright t$	range restriction operator
$r \triangleright t$	range subtraction operator



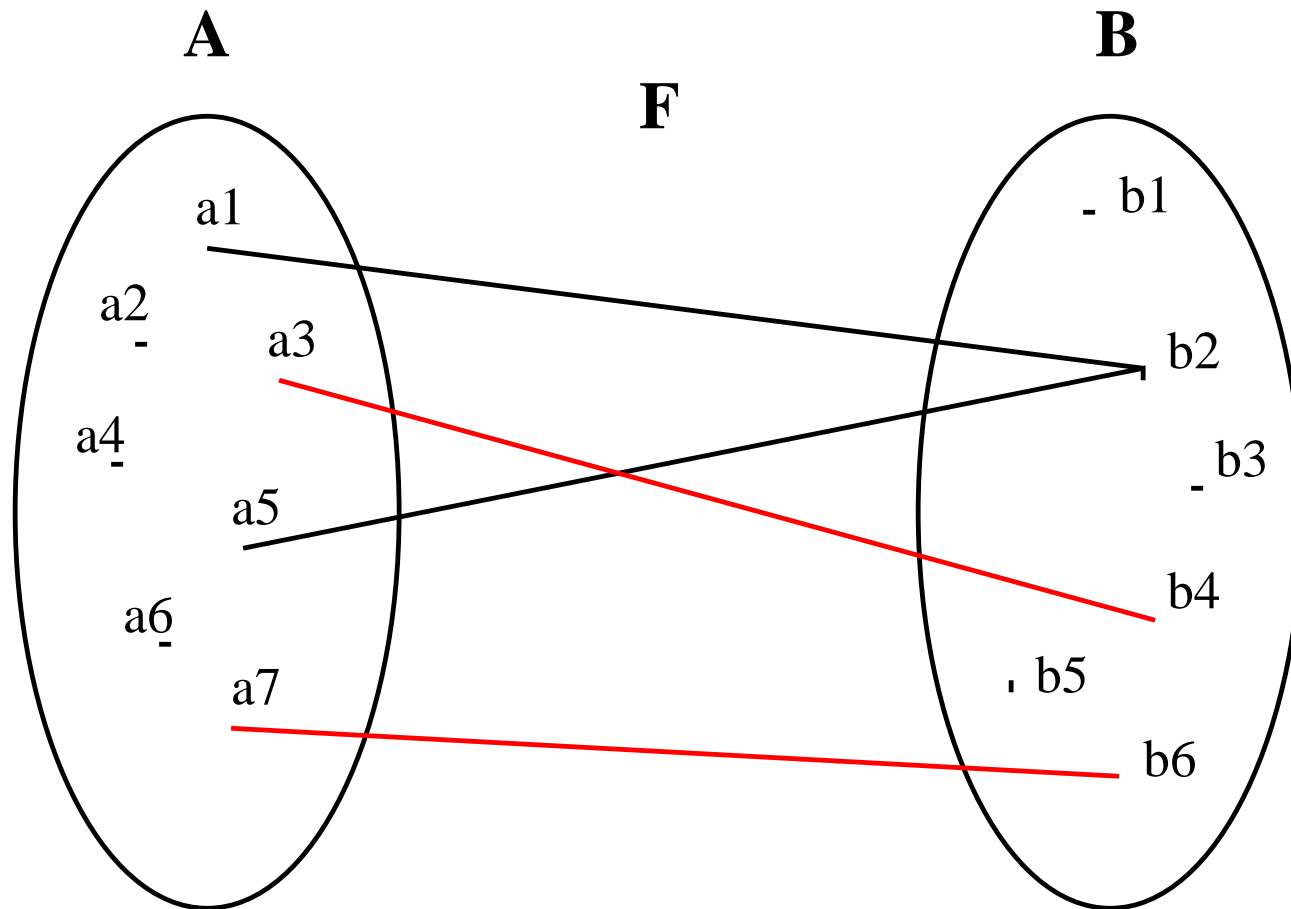
$$\{a_3, a_7\} \triangleleft F$$



$$\{a_3, a_7\} \triangleleft F$$



$$F \triangleright \{b2, b4\}$$

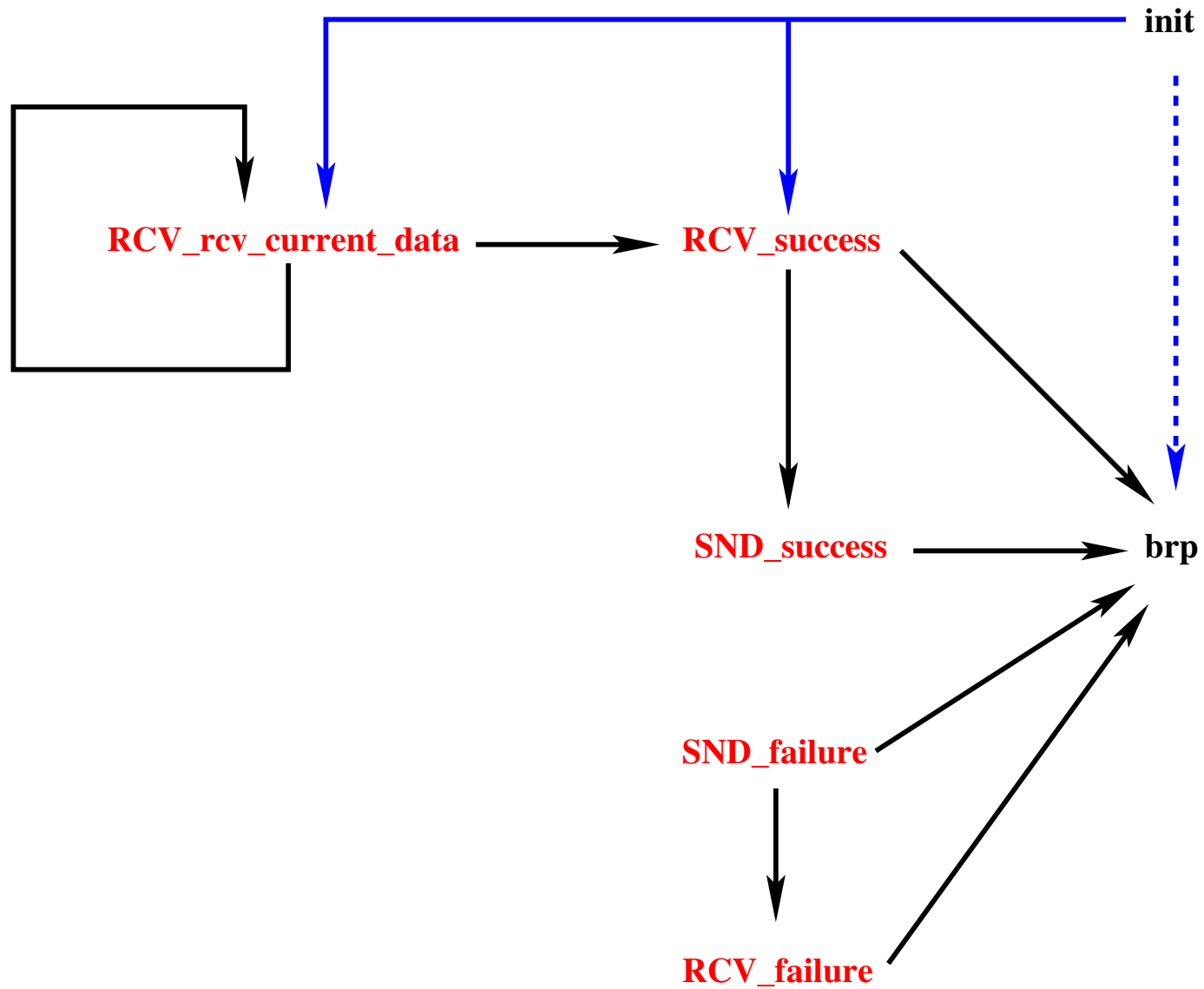


$$F \triangleright \{b_2\}$$

**init**



**brp**





- They allow to **observe** the (future) system with a **finer time grain**
- Analogies with a **microscope** or a **parachute**
- They **refine** the (implicit) event **doing nothing** (skip)
- They must **not take control for ever** (exhibiting a variant)

**set:** *STATUS*

**constants:** ...  
*working*  
*success*  
*failure*

**axm1\_1:** *STATUS* = {*working*, *success*, *failure*}

**axm1\_2:** *working*  $\neq$  *success*

**axm1\_3:** *working*  $\neq$  *failure*

**axm1\_4:** *success*  $\neq$  *failure*

- Variables  $i$  and  $g$  are replaced by variables  $r$  and  $h$
- Variable  $r$  denotes the size of the transmitted file
- Variable  $h$  denotes the transmitted file
- Variables  $s\_st$  and  $r\_st$  denote the status of the participants (Sender and Receiver respectively).

<b>variables:</b>	$r$
	$h$
	$s\_st$
	$r\_st$

- Variables  $h$  is a prefix of constant  $f$  (invariant **inv1\_1** and **inv1\_2**)

$$\mathbf{inv1\_1:} \quad r \in 0 .. n$$

$$\mathbf{inv1\_2:} \quad h = (1 .. r) \triangleleft f$$

- The **typing** of variables  $s\_st$  and  $r\_st$  is **implicit** (FUN\_4)
- Requirements FUN\_7 and FUN\_8 (Receiver's belief is true) is taken care invariant **inv1\_3**
- Requirements FUN\_5 and FUN\_6 (Sender's status) are taken care by invariant **inv\_4**

$$\mathbf{inv1\_3:} \quad r\_st = success \Leftrightarrow r = n$$

$$\mathbf{inv1\_4:} \quad s\_st = success \Rightarrow r\_st = success$$

- Initialisation

init

$r := 0$

$h := \emptyset$

$r\_st := working$

$s\_st := working$

- Event **(concrete\_)brp** now **does nothing**
- We give **witnesses** for the abstract after values  $i'$  and  $g'$

$$\text{(abstract\_)}\text{brp}$$
$$i, g : | \left( \begin{array}{l} i' \in 0 .. n \\ g' = (1 .. i') \triangleleft a \end{array} \right)$$
$$\text{(concrete\_)}\text{brp}$$

**when**  
 $r \neq \text{working}$   
 $s \neq \text{working}$

**with**  
 $i' : i' = r$   
 $g' : g' = h$

**then**  
skip

**end**

```
RCV_rcv_current_data
  when
     $r\_st = working$ 
     $r + 1 < n$ 
  then
     $r := r + 1$ 
     $h := h \cup \{r + 1 \mapsto f(r + 1)\}$ 
  end
```

- This event is "cheating" (accessing constant  $f$ )
- This new event maintains invariant **inv1\_3** and it refines skip

**inv1\_3:**  $r\_st = success \Leftrightarrow r = n$



RCV\_success

**when**

$r\_st = working$

$r + 1 = n$

**then**

$r\_st := success$

$r := r + 1$

$h := h \cup \{n \mapsto f(n)\}$

**end**

RCV\_failure

**when**

$r\_st = working$

$s\_st = failure$

**then**

$r\_st := failure$

**end**

- These new events are **cheating** (accessing  $f$  and  $s\_st$ )
- These new events **maintain inv1\_3 and inv1\_4** and they refine **skip**

**inv1\_3:**  $r\_st = success \Leftrightarrow r = n$

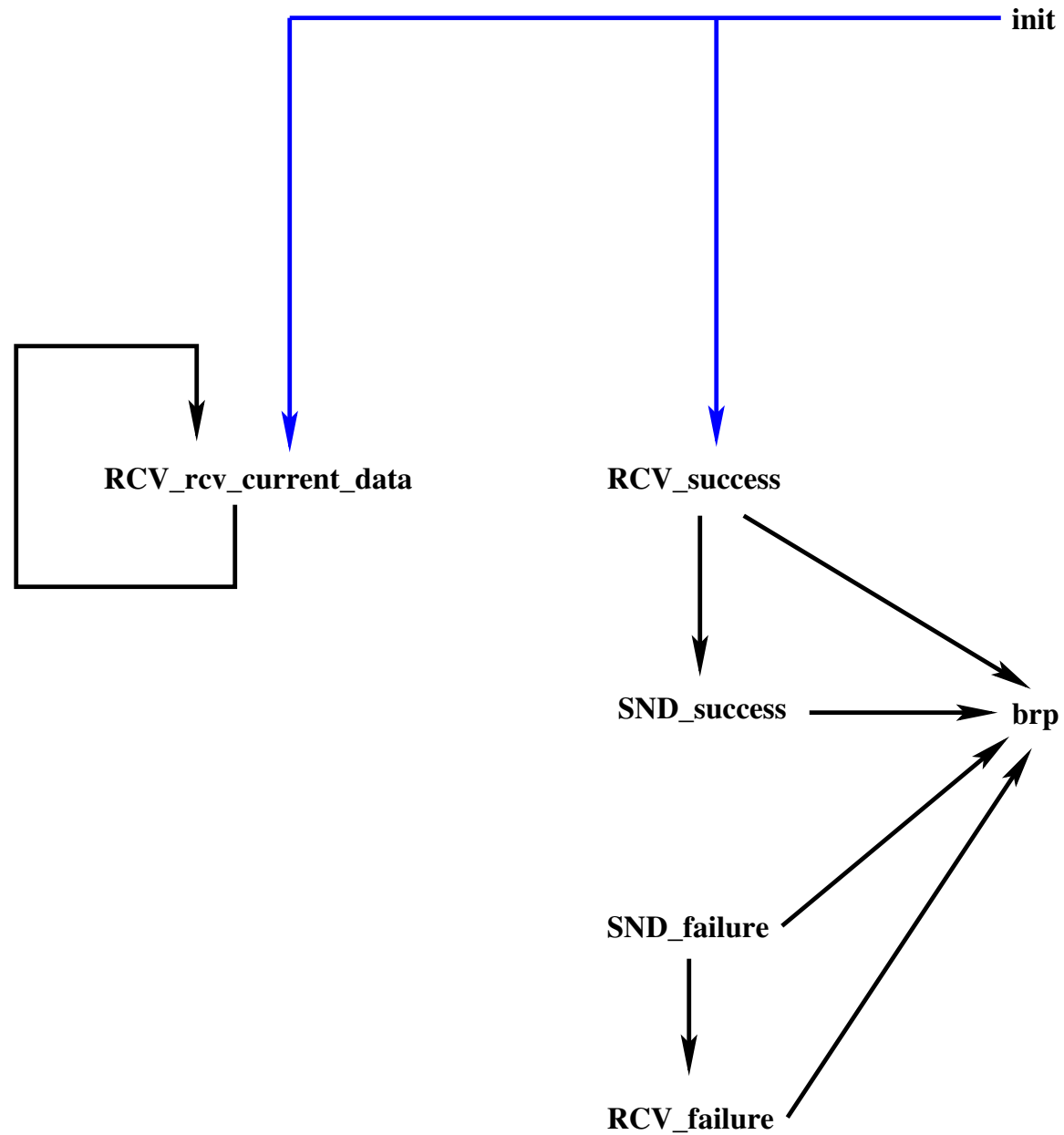
**inv1\_4:**  $s\_st = success \Rightarrow r\_st = success$

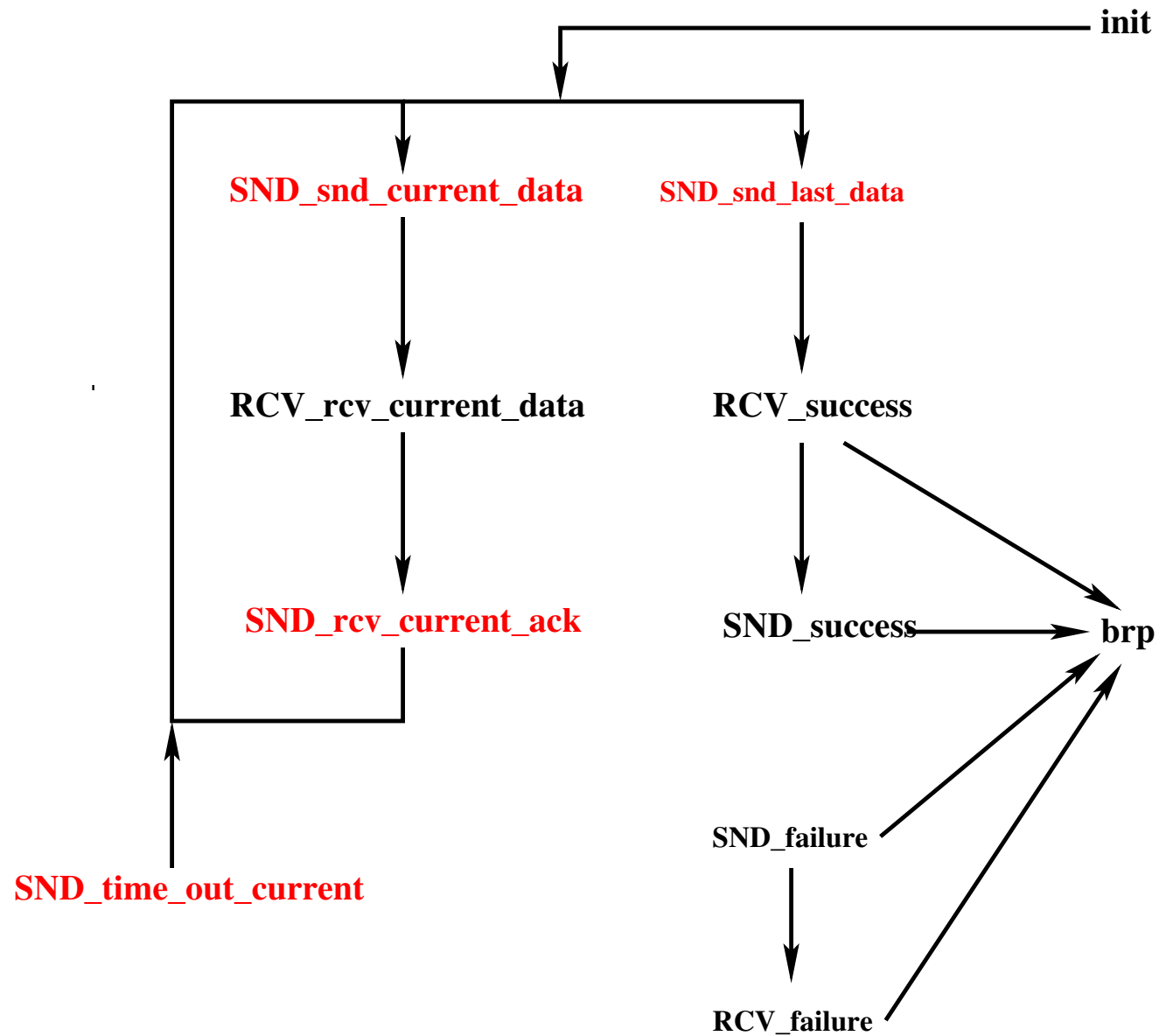
```
SND_success
  when
    s_st = working
    r_st = success
  then
    s_st := success
  end
```

```
SND_failure
  when
    s_st = working
  then
    s_st := failure
  end
```

- Event SND\_success is **cheating** (accessing *r\_st*)
- Event SND\_success **maintains invariant inv1\_4**

**inv1\_4:**  $s\_st = success \Rightarrow r\_st = success$





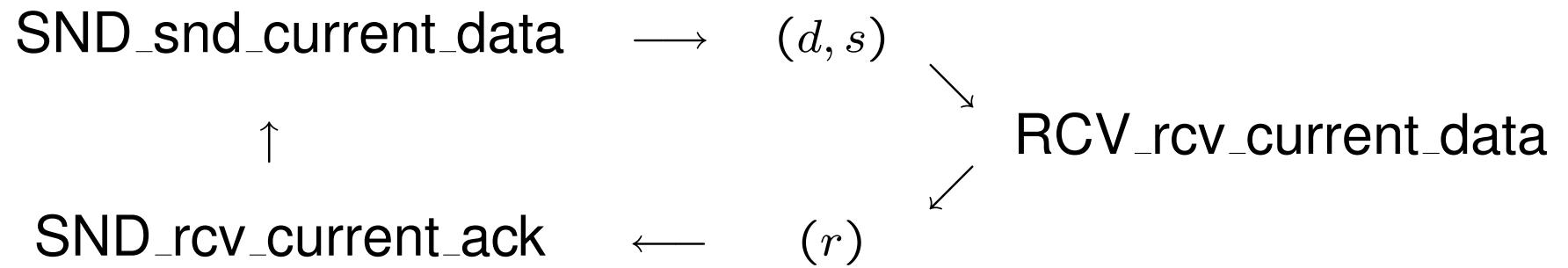
- Variable  $s$  is the Sender **pointer** sent to the Receiver
- Variable  $d$  is the **data sent** to the Receiver
- Variable  $w$  is the Sender **activation bit**
- When  $w$  is **TRUE** it means the Sender has **just received the acknowledgement**
- When  $w$  is **FALSE** it means the Sender has **sent the information to the Receiver**

**variables:**     $\dots$   
                   $w$   
                   $s$   
                   $d$

**inv2\_1:**     $s \in 0 .. n - 1$

**inv2\_2:**     $r \in s .. s + 1$

**inv2\_3:**     $w = \text{FALSE} \Rightarrow d = f(s + 1)$



init

$r := 0$

$h := \emptyset$

$r\_st := working$

$s\_st := working$

$s := 0$

$d \in D$

$w := \text{TRUE}$

brp

**when**

$r\_st \neq working$

$s\_st \neq working$

**then**

skip

**end**

- **New Events**: the Sender prepares **data  $d$**  and **pointer  $s$**  to be sent

```
SND_snd_current_data
when
   $s\_st = working$ 
   $w = \text{TRUE}$ 
   $s + 1 < n$ 
then
   $d := f(s + 1)$ 
   $w := \text{FALSE}$ 
end
```

```
SND_snd_last_data
when
   $s\_st = working$ 
   $w = \text{TRUE}$ 
   $s + 1 = n$ 
then
   $d := f(s + 1)$ 
   $w := \text{FALSE}$ 
end
```

- These events clearly **refine skip** and maintain invariant **inv2\_3**

**inv2\_3:**  $w = \text{FALSE} \Rightarrow d = f(s + 1)$



- The Receiver receives data  $d$  and pointer  $s$ . It sends pointer  $r$ .

RCV\_rcv\_current\_data

**when**

$r\_st = working$

$w = FALSE$

$r = s$

$r + 1 < n$

**then**

$r := r + 1$

$h := h \cup \{r + 1 \mapsto d\}$

**end**

RCV\_success

**when**

$r\_st = working$

$w = FALSE$

$r = s$

$r + 1 = n$

**then**

$r\_st := success$

$r := r + 1$

$h := h \cup \{r + 1 \mapsto d\}$

**end**

- The Receiver **still cheats**: it accesses constant  $n$

```

(abstract-)RCV_rcv_current_data
when
   $r\_st = working$ 
   $r + 1 < n$ 
then
   $r := r + 1$ 
   $h := h \cup \{r + 1 \mapsto f(r + 1)\}$ 
end

```

```

(concrete-)RCV_rcv_current_data
when
   $r\_st = working$ 
   $w = \mathbf{FALSE}$ 
   $r = s$ 
   $r + 1 < n$ 
then
   $r := r + 1$ 
   $h := h \cup \{r + 1 \mapsto d\}$ 
end

```

- Observe **guard strengthening**
- This invariant helps proving **event refinement**

**inv2\_3:**  $w = \mathbf{FALSE} \Rightarrow d = f(s + 1)$

```

(abstract-)RCV_success
when
   $r\_st = working$ 
   $r + 1 = n$ 
then
   $r\_st := success$ 
   $r := r + 1$ 
   $h := h \cup \{n \mapsto f(n)\}$ 
end

```

```

(concrete-)RCV_success
when
   $r\_st = working$ 
   $w = \mathbf{FALSE}$ 
   $r = s$ 
   $r + 1 = n$ 
then
   $r\_st := success$ 
   $r := r + 1$ 
   $h := h \cup \{r + 1 \mapsto d\}$ 
end

```

- Observe **guard strengthening**
- This invariant helps proving **event refinement**

**inv2\_3:**  $w = \mathbf{FALSE} \Rightarrow d = f(s + 1)$

- The first event is **new**. It clearly **refines skip**
- The activation bit is set to **TRUE** (activating `SND_snd_current_data`)
- The Sender receives acknowledgment (pointer  $r$ )

```
SND_rcv_current_ack
when
     $s\_st = working$ 
     $w = FALSE$ 
     $s + 1 < n$ 
     $r = s + 1$ 
then
     $w := TRUE$ 
     $s := s + 1$ 
end
```

```
SND_success
when
     $s\_st = working$ 
     $w = FALSE$ 
     $s + 1 = n$ 
     $r = s + 1$ 
then
     $s\_st := success$ 
end
```

```
(abstract-)SND_success
when
  s_st = working
  r_st = success
then
  s_st := success
end
```

```
(concrete-)SND_success
when
  s_st = working
  w = FALSE
  s + 1 = n
  r = s + 1
then
  s_st := success
end
```

- The presence of **inv1\_3** ensures that the **guard is strengthened**

```
inv1_3: r_st = success  $\Leftrightarrow$  r = n
```

- This new events will receive a full explanation in the next refinement

```
SND_time_out_current
  when
    s_st = working
    w = FALSE
  then
    w := TRUE
  end
```

- **At most** one activation bit is **TRUE** at a time

**variables:**    ...  
                   $db$   
                   $ab$   
                   $v$

**inv3\_1:**  $w = \text{TRUE} \Rightarrow db = \text{FALSE}$

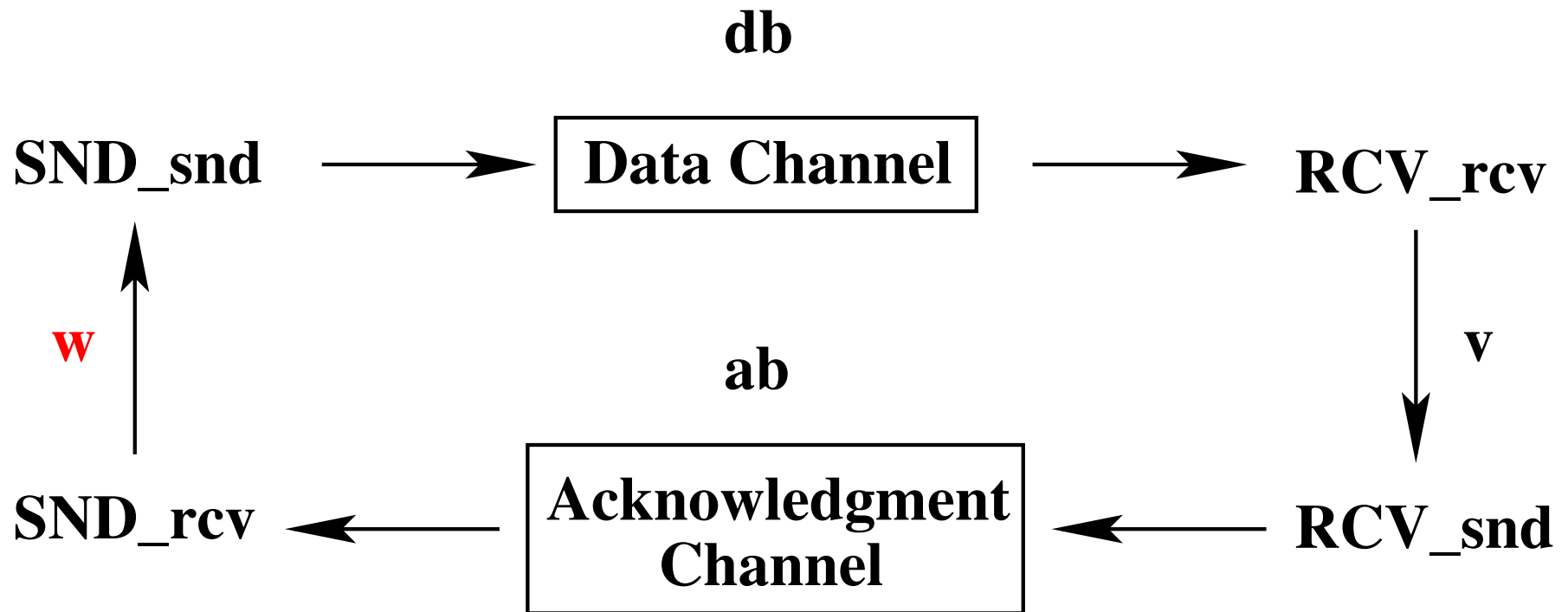
**inv3\_2:**  $w = \text{TRUE} \Rightarrow ab = \text{FALSE}$

**inv3\_3:**  $w = \text{TRUE} \Rightarrow v = \text{FALSE}$

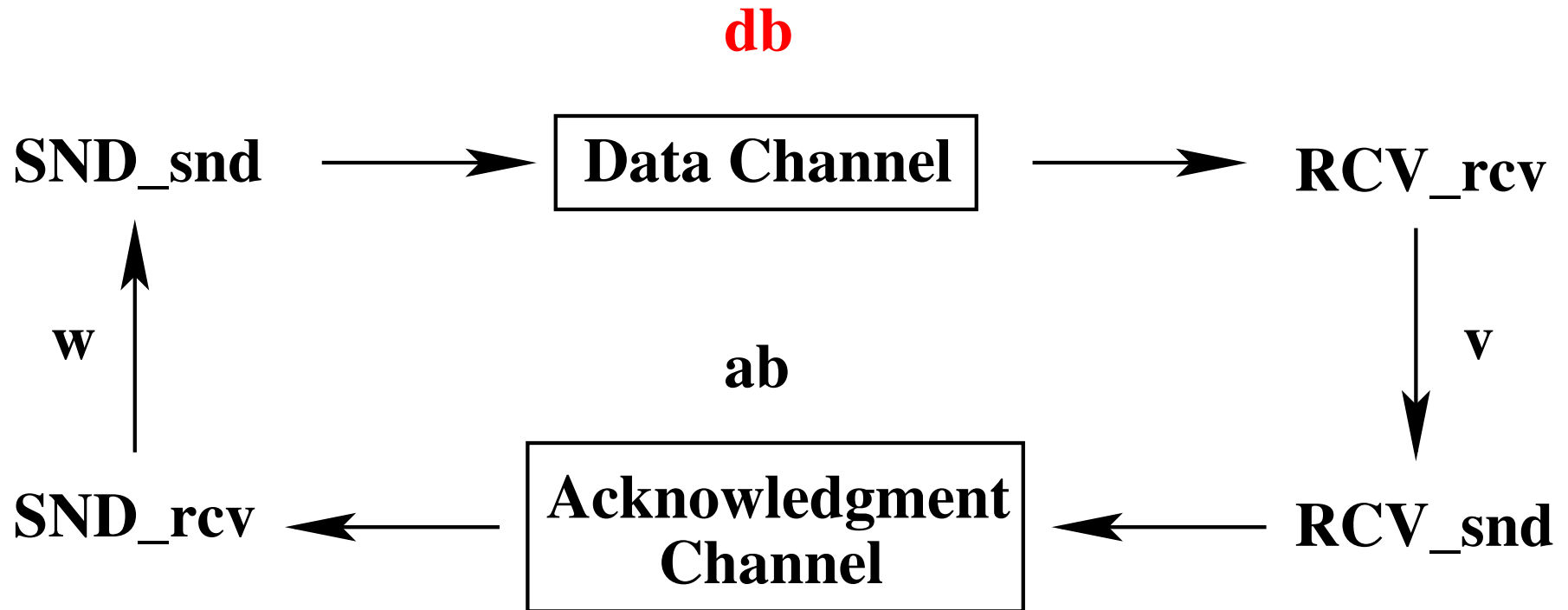
**inv3\_4:**  $db = \text{TRUE} \Rightarrow ab = \text{FALSE}$

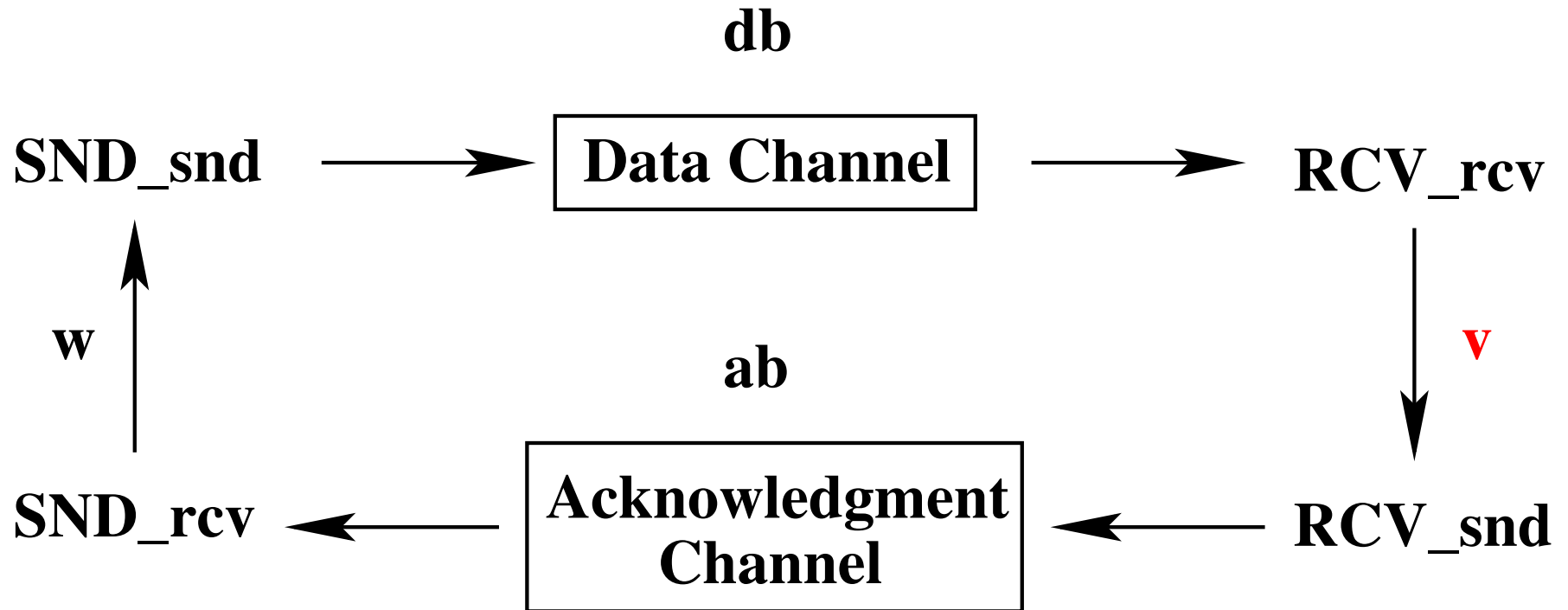
**inv3\_5:**  $db = \text{TRUE} \Rightarrow v = \text{FALSE}$

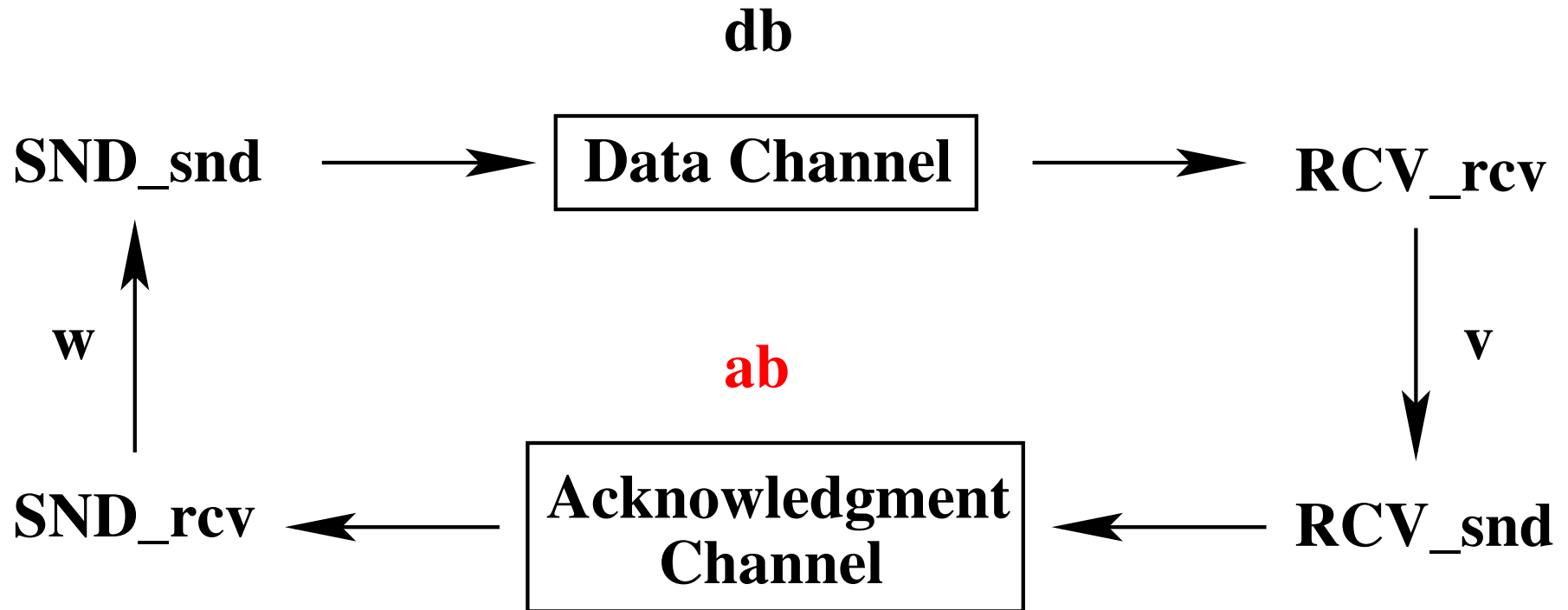
**inv3\_6:**  $ab = \text{TRUE} \Rightarrow v = \text{FALSE}$

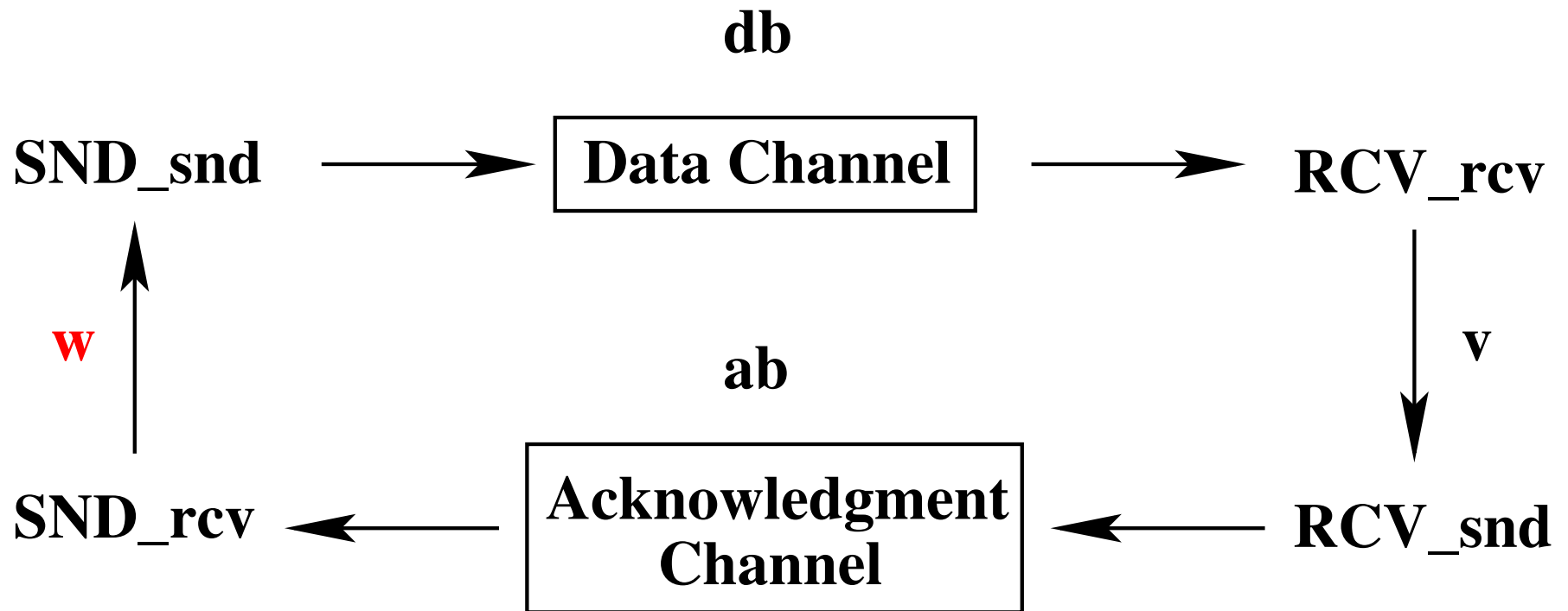


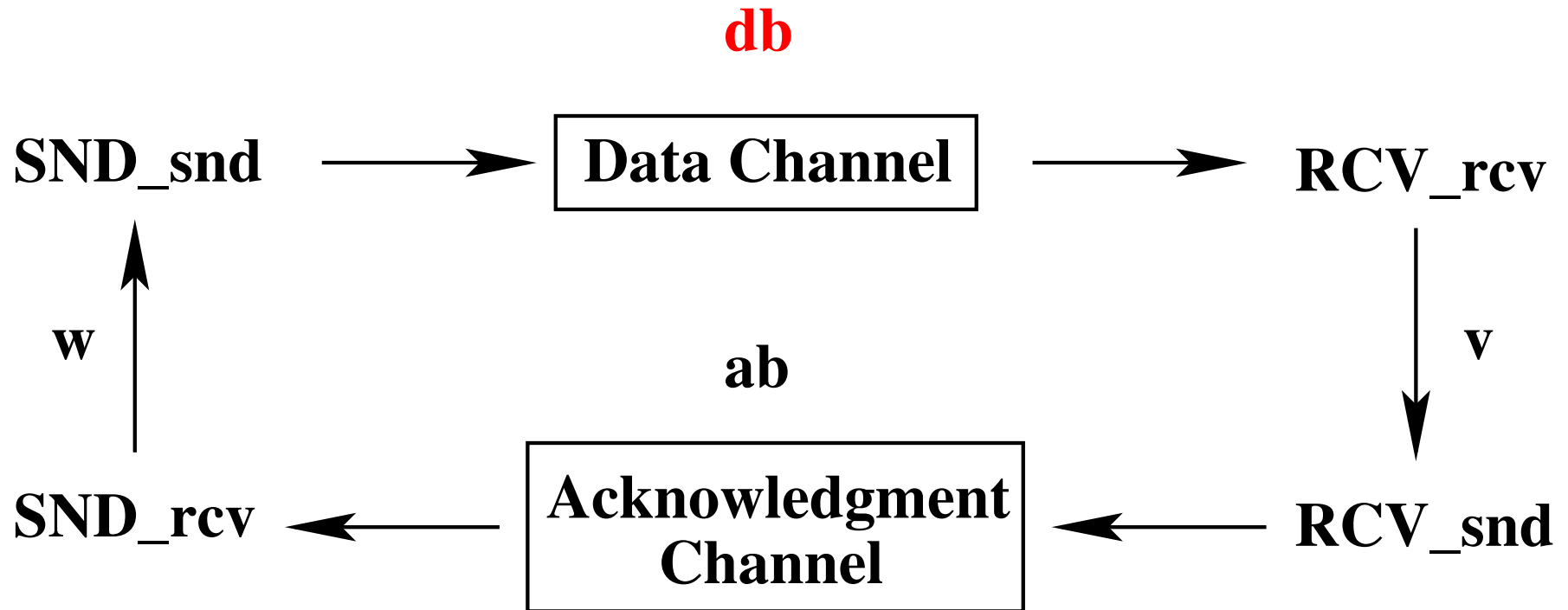


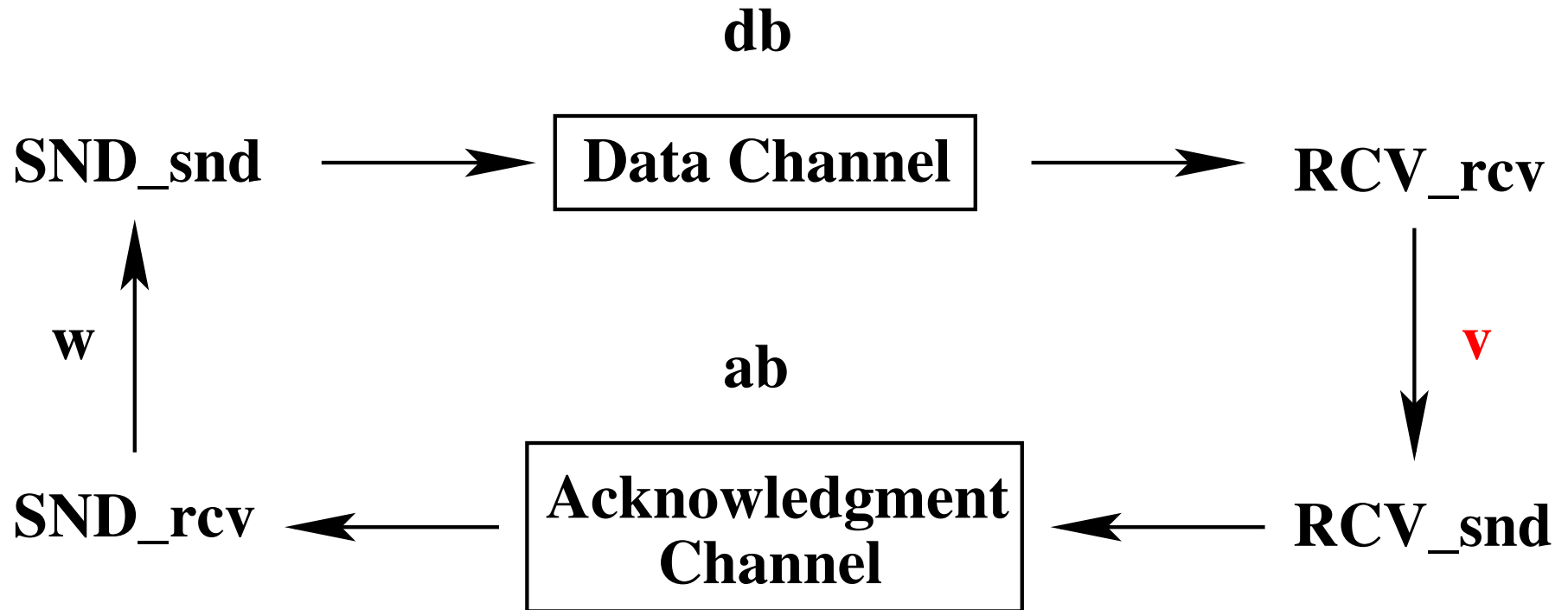


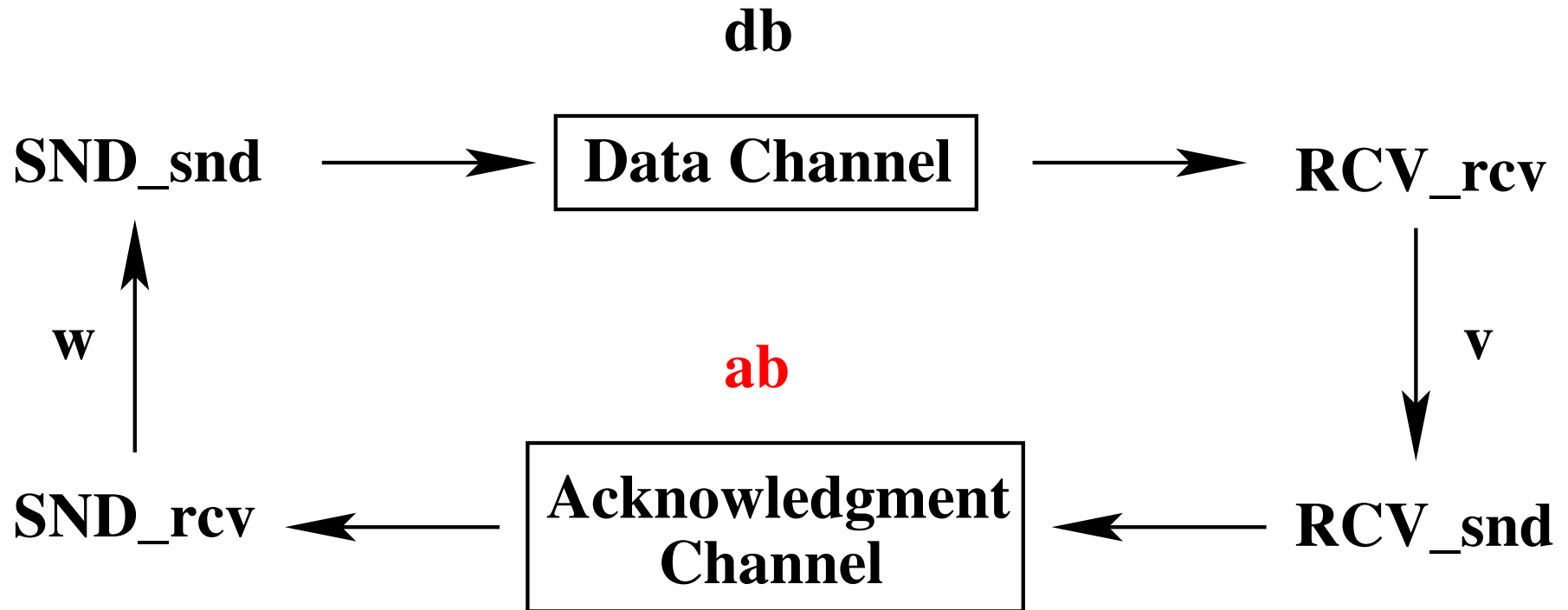












- These invariants define the **last data indicator**

**variables:**     $\dots$   
                   $l$

**inv3\_7:**  $db = \text{TRUE} \wedge r = s \wedge l = \text{FALSE} \Rightarrow r + 1 < n$

**inv3\_8:**  $db = \text{TRUE} \wedge r = s \wedge l = \text{TRUE} \Rightarrow r + 1 = n$

- This bit is sent by the Sender to the Receiver
- When equal to **TRUE**, this bit indicates that the sent item is the last one



- Constant  $MAX$  denotes the maximum number of retries
- The sender fails iff the retry counter  $c$  exceeds  $MAX$  (**inv3\_10**)

**constants:** ...  
 $MAX$

**axm3\_1:**  $MAX \in \mathbb{N}$

**variables:** ...  
 $c$

**inv3\_9:**  $c \in 0 .. MAX + 1$

**inv3\_10:**  $c = MAX + 1 \Leftrightarrow s\_st = failure$

init

$r := 0$

$h := \emptyset$

$r\_st := working$

$s\_st := working$

$s := 0$

$d \in D$

$w := \text{TRUE}$

$db := \text{FALSE}$

$ab := \text{FALSE}$

$v := \text{FALSE}$

$l := \text{FALSE}$

$c := 0$

brp

**when**

$r \neq working$

$s \neq working$

**then**

skip

**end**

SND\_snd\_current\_data

**when**

$s\_st = working$

$w = \text{TRUE}$

$s + 1 < n$

**then**

$d := f(s + 1)$

$w := \text{FALSE}$

$db := \text{TRUE}$

$l := \text{FALSE}$

**end**

SND\_snd\_last\_data

**when**

$s\_st = working$

$w = \text{TRUE}$

$s + 1 = n$

**then**

$d := f(s + 1)$

$w := \text{FALSE}$

$db := \text{TRUE}$

$l := \text{TRUE}$

**end**

- Daemons are **breaking the channels**

```
DMN_data_channel
when
    db = TRUE
then
    db = FALSE
end
```

```
DMN_ack_channel
when
    ab = TRUE
then
    ab = FALSE
end
```

- A **failure** is characterized by **all activation bits being FALSE**

SND\_time\_out\_current

**when**

*s\_st = working*

*w = FALSE*

*ab = FALSE*

*db = FALSE*

*v = FALSE*

*c < MAX*

**then**

*w := TRUE*

*c := c + 1*

**end**

SND\_failure

**when**

*s\_st = working*

*w = FALSE*

*ab = FALSE*

*db = FALSE*

*v = FALSE*

*c = MAX*

**then**

*s\_st := failure*

*c := c + 1*

**end**

- Sender aborts after  $MAX + 1$  tries

Rcv\_rcv\_current\_data

**when**

$r\_st = working$

$db = TRUE$

$r = s$

$l = FALSE$

**then**

$r := r + 1$

$h := h \cup \{r + 1 \mapsto d\}$

$db := FALSE$

$v := TRUE$

**end**

Rcv\_success

**when**

$r\_st = working$

$db = TRUE$

$r = s$

$l = TRUE$

**then**

$r\_st := success$

$r := r + 1$

$h := h \cup \{r + 1 \mapsto d\}$

$db := FALSE$

$v := TRUE$

**end**

Reminder:  $l$  is the last data indicator

```

(abstract-)RCV_rcv_current_data
when
   $r\_st = working$ 
   $w = FALSE$ 
   $r = s$ 
   $r + 1 < n$ 
then
   $r := r + 1$ 
   $h := h \cup \{r + 1 \mapsto d\}$ 
end

```

```

(concrete-)RCV_rcv_current_data
when
   $r\_st = working$ 
   $db = TRUE$ 
   $r = s$ 
   $l = FALSE$ 
then
   $r := r + 1$ 
   $h := h \cup \{r + 1 \mapsto d\}$ 
   $db := FALSE$ 
   $v := TRUE$ 
end

```

**inv3\_1'**:  $db = TRUE \Rightarrow w = FALSE$

**inv3\_7**:  $db = TRUE \wedge r = s \wedge l = FALSE \Rightarrow r + 1 < n$

```

(abstract-)RCV_success
when
  r_st = working
  w = FALSE
  r = s
  r + 1 = n
then
  r := r + 1
  h := h ∪ {r + 1 ↦ d}
end

```

```

(concrete-)RCV_success
when
  r_st = working
  db = TRUE
  r = s
  l = TRUE
then
  r_st := success
  r := r + 1
  h := h ∪ {r + 1 ↦ d}
  db := FALSE
  v := TRUE
end

```

**inv3\_1'**:  $db = \text{TRUE} \Rightarrow w = \text{FALSE}$

**inv3\_8**:  $db = \text{TRUE} \wedge r = s \wedge l = \text{TRUE} \Rightarrow r + 1 = n$



```
RCV_rcv_retry
  when
    db = TRUE
    r ≠ s
  then
    db := FALSE
    v := TRUE
  end
```

```
RCV_snd_ack
  when
    v = TRUE
  then
    v := FALSE
    ab := TRUE
  end
```

```
RCV_failure
  when
    r_st = working
    c = MAX + 1
  then
    r_st := failure
  end
```

```
SND_rcv_current_ack
when
  s_st = working
  ab = TRUE
   $s + 1 < n$ 
then
  w := TRUE
  s := s + 1
  c := 0
  ab := FALSE
end
```

```
SND_success
when
  s_st = working
  ab = TRUE
   $s + 1 = n$ 
then
  s_st := success
  c := 0
  ab := FALSE
end
```

```
(abstract-)SND_rcv_current_ack
when
  s_st = working
  w = FALSE
   $s + 1 < n$ 
  r = s + 1
then
  w := TRUE
  s := s + 1
end
```

```
(concrete-)SND_rcv_current_ack
when
  s_st = working
  ab = TRUE
   $s + 1 < n$ 
then
  w := TRUE
  s := s + 1
  c := 0
  ab := FALSE
end
```

**inv3\_2'**:  $ab = \text{TRUE} \Rightarrow w = \text{FALSE}$

- In order to prove guard strengthening we need invariant **inv3\_11**

**inv3\_11**:  $ab = \text{TRUE} \Rightarrow r = s + 1$

**inv3\_12**:  $v = \text{TRUE} \Rightarrow r = s + 1$

- Invariant **inv3\_12** is needed to prove **inv3\_11**

```
(abstract-)SND_success
when
  s_st = working
  w = FALSE
  s + 1 = n
  r = s + 1
then
  s_st := success
end
```

```
(concrete-)SND_success
when
  s_st = working
  ab = TRUE
  s + 1 = n
then
  s_st := success
  c := 0
  ab := FALSE
end
```

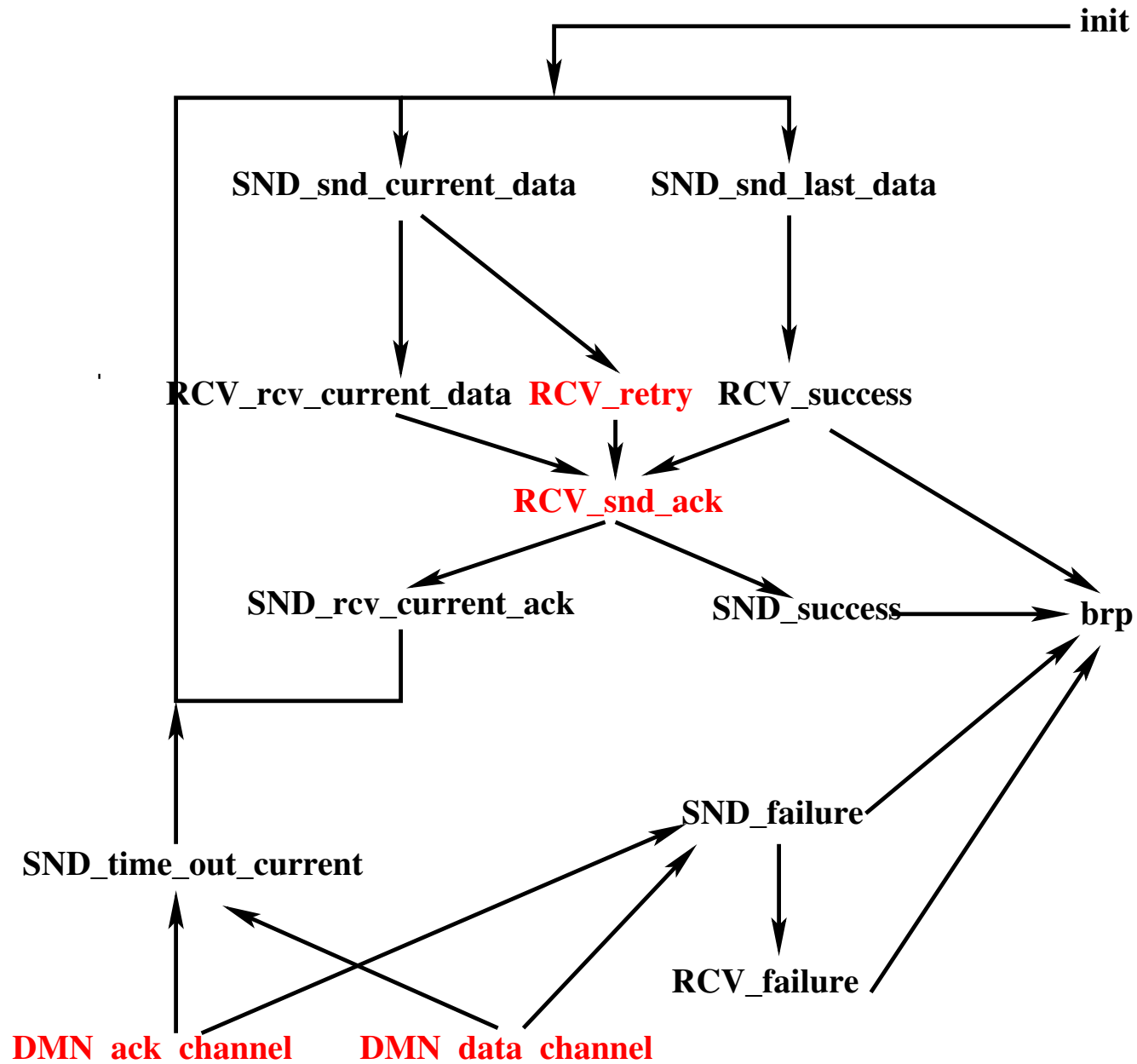
**inv3\_2'**:  $ab = \text{TRUE} \Rightarrow w = \text{FALSE}$

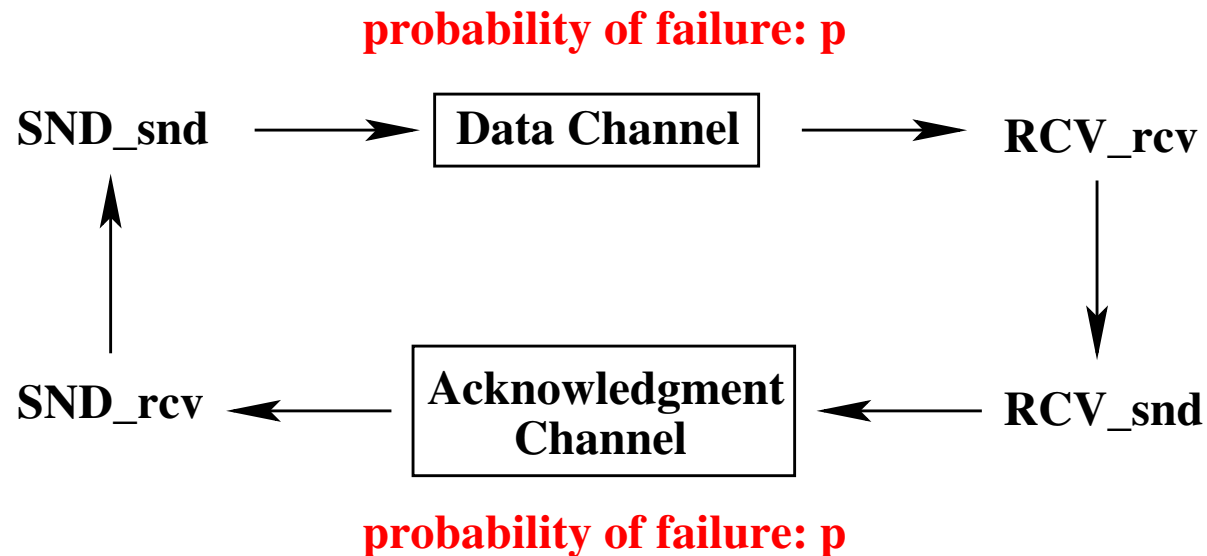
- In order to prove guard strengthening we need invariant **inv3\_11**

**inv3\_11**:  $ab = \text{TRUE} \Rightarrow r = s + 1$

**inv3\_12**:  $v = \text{TRUE} \Rightarrow r = s + 1$

- Invariant **inv3\_12** is needed to prove **inv3\_11**





- We would like to compute the probability of success
- It is a function of:
  - $p$ : probability of failure for one channel
  - $n$ : size of the file
  - $MAX + 1$ : number of re-tries

Failure on one channel

$p$

Failure on one channel  $p$

Success on one channel  $1 - p$



Failure on one channel	$p$
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$

Failure on one channel	$p$
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Fails on one try	$1 - (1 - p)^2$

---

Failure on one channel	$p$
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Fails on one try	$1 - (1 - p)^2$
Fails on $MAX + 1$ tries	$(1 - (1 - p)^2)^{MAX+1}$

---

Failure on one channel	$p$
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Fails on one try	$1 - (1 - p)^2$
Fails on $MAX + 1$ tries	$(1 - (1 - p)^2)^{MAX+1}$
Succ. on $MAX + 1$ tries	$1 - (1 - (1 - p)^2)^{MAX+1}$

---

Failure on one channel	$p$
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Fails on one try	$1 - (1 - p)^2$
Fails on $MAX + 1$ tries	$(1 - (1 - p)^2)^{MAX+1}$
Succ. on $MAX + 1$ tries	$1 - (1 - (1 - p)^2)^{MAX+1}$
Success for n data	$(1 - (1 - (1 - p)^2)^{MAX+1})^n$

---

Failure on one channel	$p$
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Fails on one try	$1 - (1 - p)^2$
Fails on $MAX + 1$ tries	$(1 - (1 - p)^2)^{MAX+1}$
Succ. on $MAX + 1$ tries	$1 - (1 - (1 - p)^2)^{MAX+1}$
Success for $n$ data	$(1 - (1 - (1 - p)^2)^{MAX+1})^n$

$$p = .1$$

$$MAX = 5$$

$$n = 100$$

$$.995$$