Abstract Model Checking and CEGAR

Abstraction

- Abstraction of states → abstract states
- A mapping $\alpha : S \rightarrow A$
- Usually, *A* is much simpler than *S*
  - Typically, *A* is a finite set while *S* is infinite
- Example: alternating bit protocol
  - State is a tuple of
    - Sender's state
    - Receiver's state
    - Channels' states --- infinite ← apply abstraction

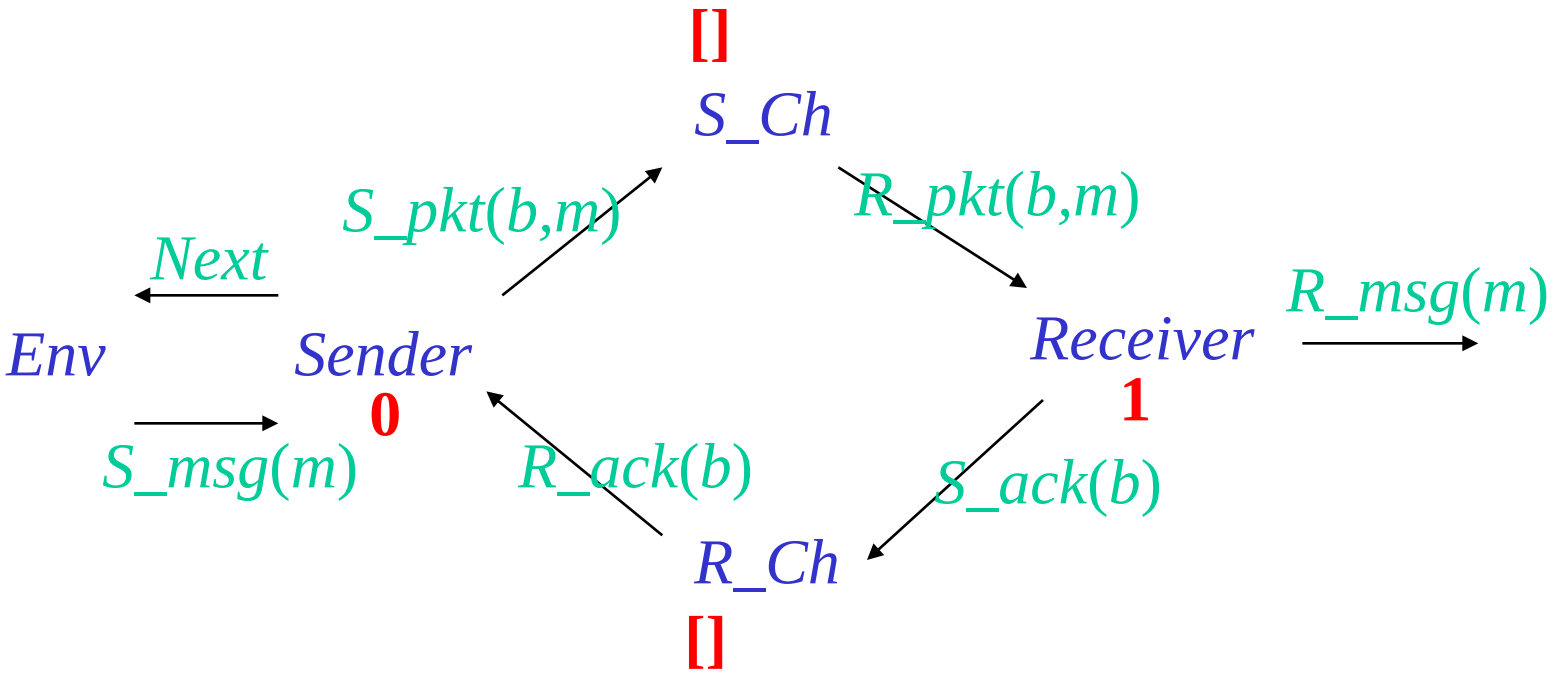
Various Kinds of Abstraction

- Abstraction of data (abstract interpretation)
  - Sign abstraction
    - Numbers → "**-**", **0**, "**+**"
    - "**+**"+"**+**" = "**+**", "**+**"+**0**="**+**", "**+**"+"**-**"=**T** (unknown)
  - Abstraction of lists/multisets → sets
    - [a,b,c,a,b,a,a,c] → {a,b,c}
  - Boolean abstraction (cf. SLAM)
  - Constant abstraction
- Abstraction of processes
  - Processes with identical state are mapped to one
    - A kind of multisets→sets

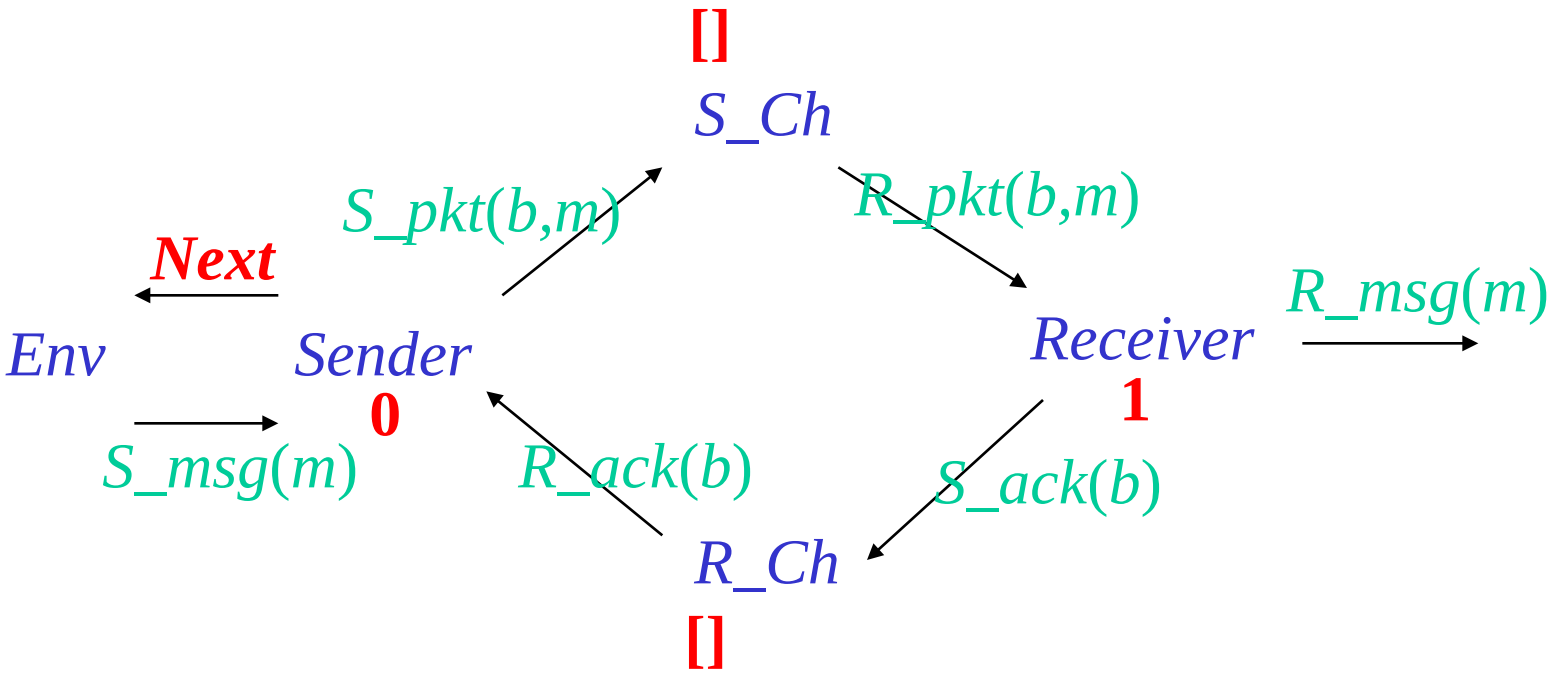
Alternating Bit Protocol

- *Sender*
- *Receiver*
- *S_Ch*: channel of message packets
  - Each packet is a pair of a message and a header
  - Modeled as a queue of packets
  - Messages may be duplicated or lost
- *R_Ch*: channel of acknowledgements
  - Ditto

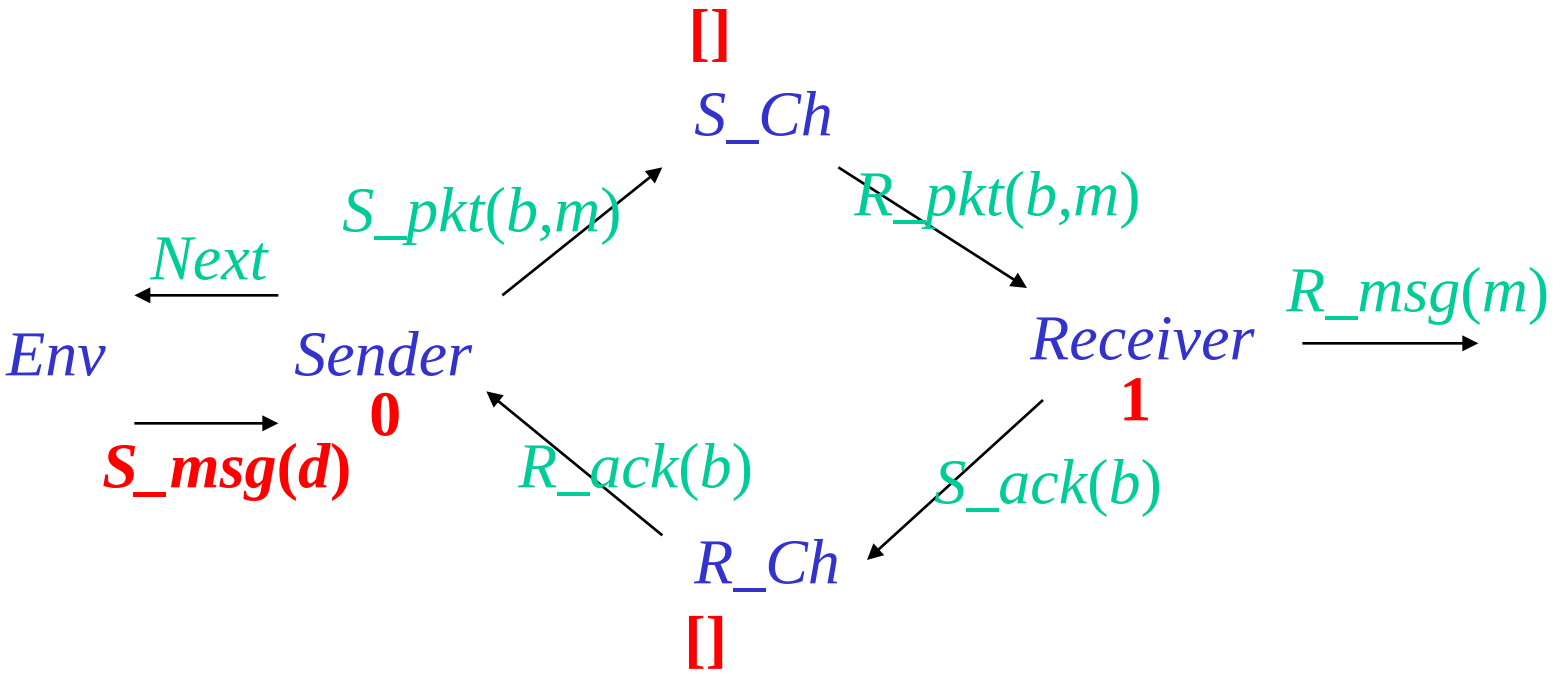
Alternating Bit Protocol



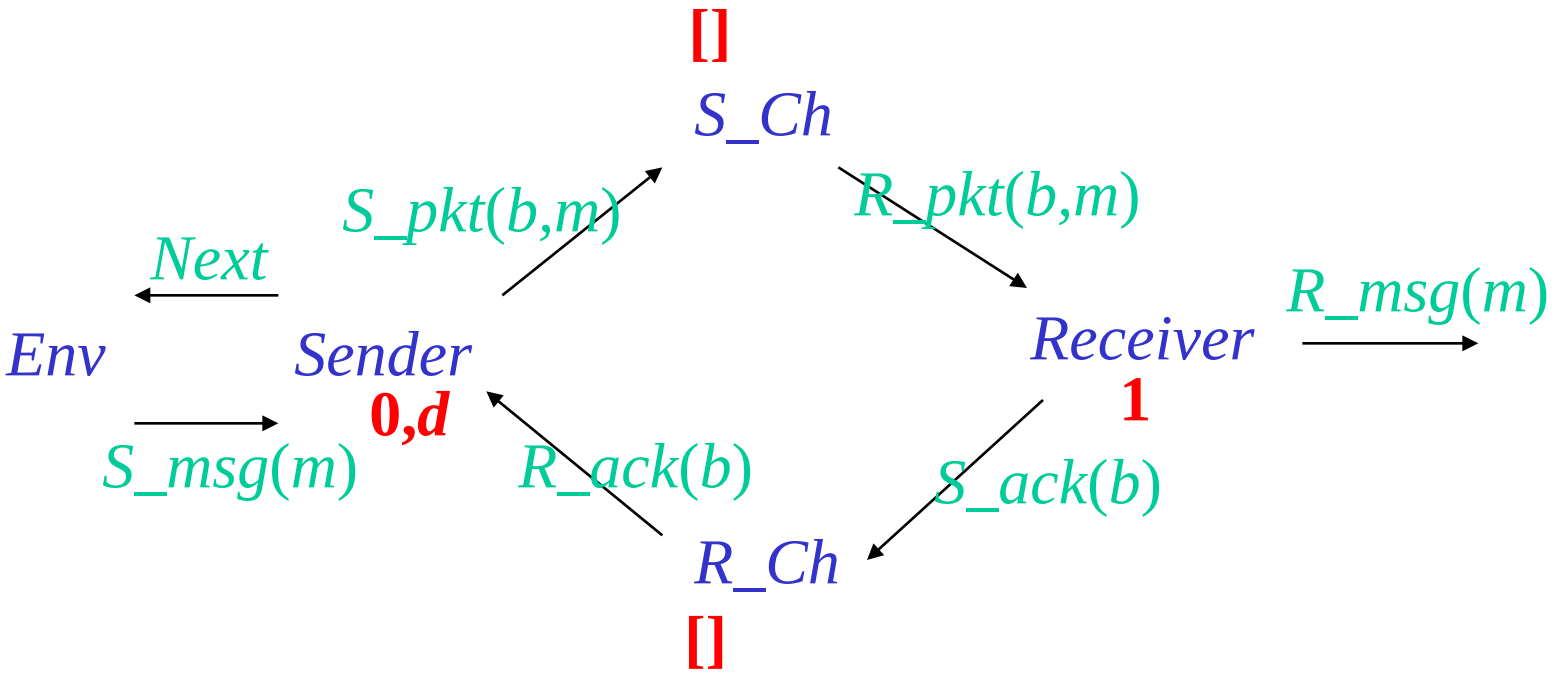
Alternating Bit Protocol



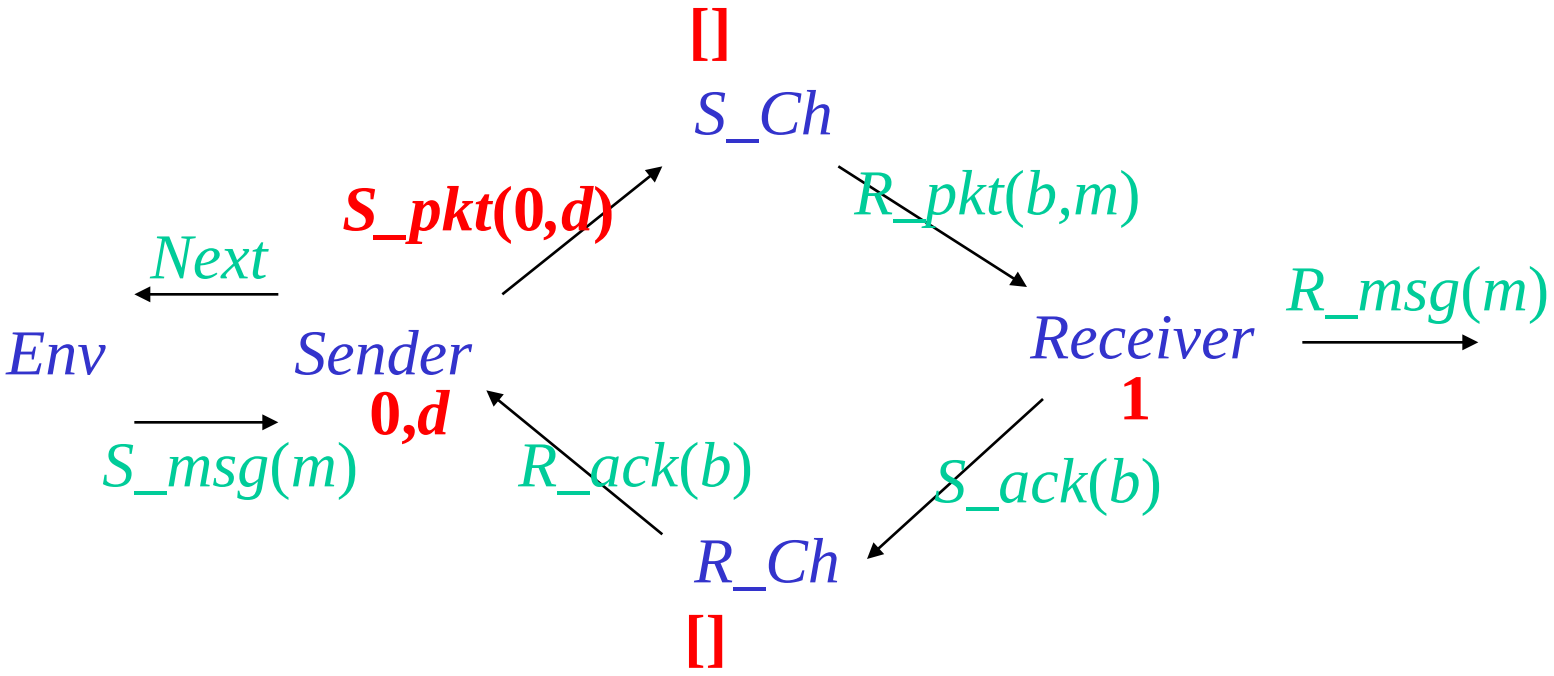
Alternating Bit Protocol



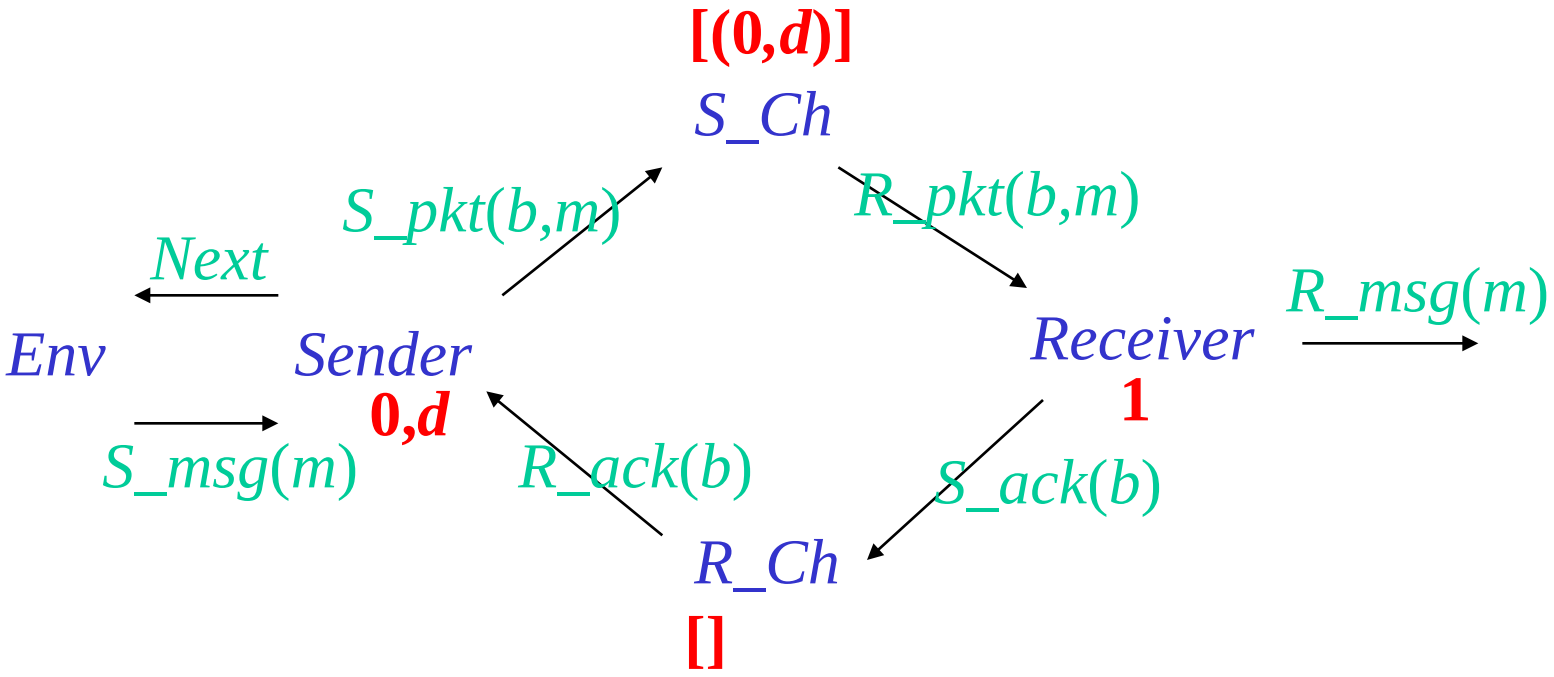
Alternating Bit Protocol



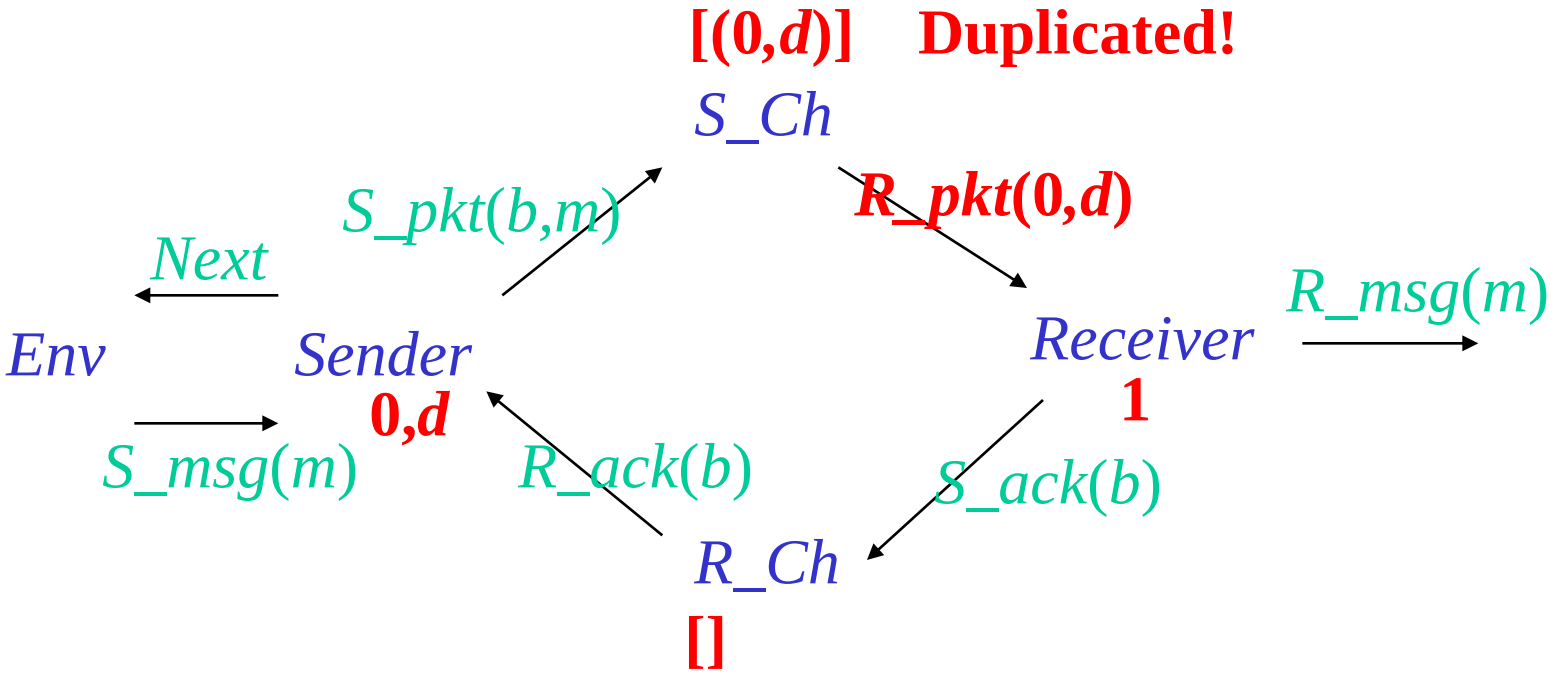
Alternating Bit Protocol



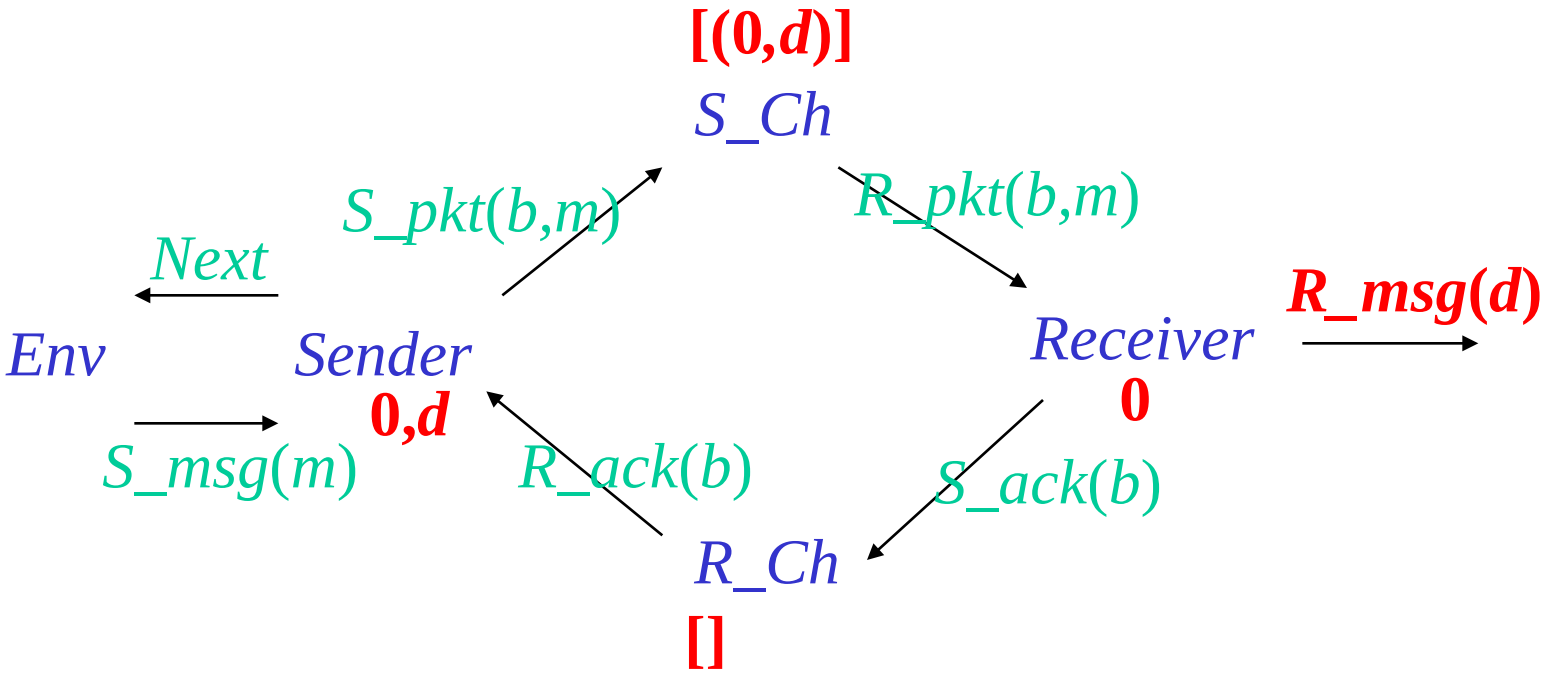
Alternating Bit Protocol



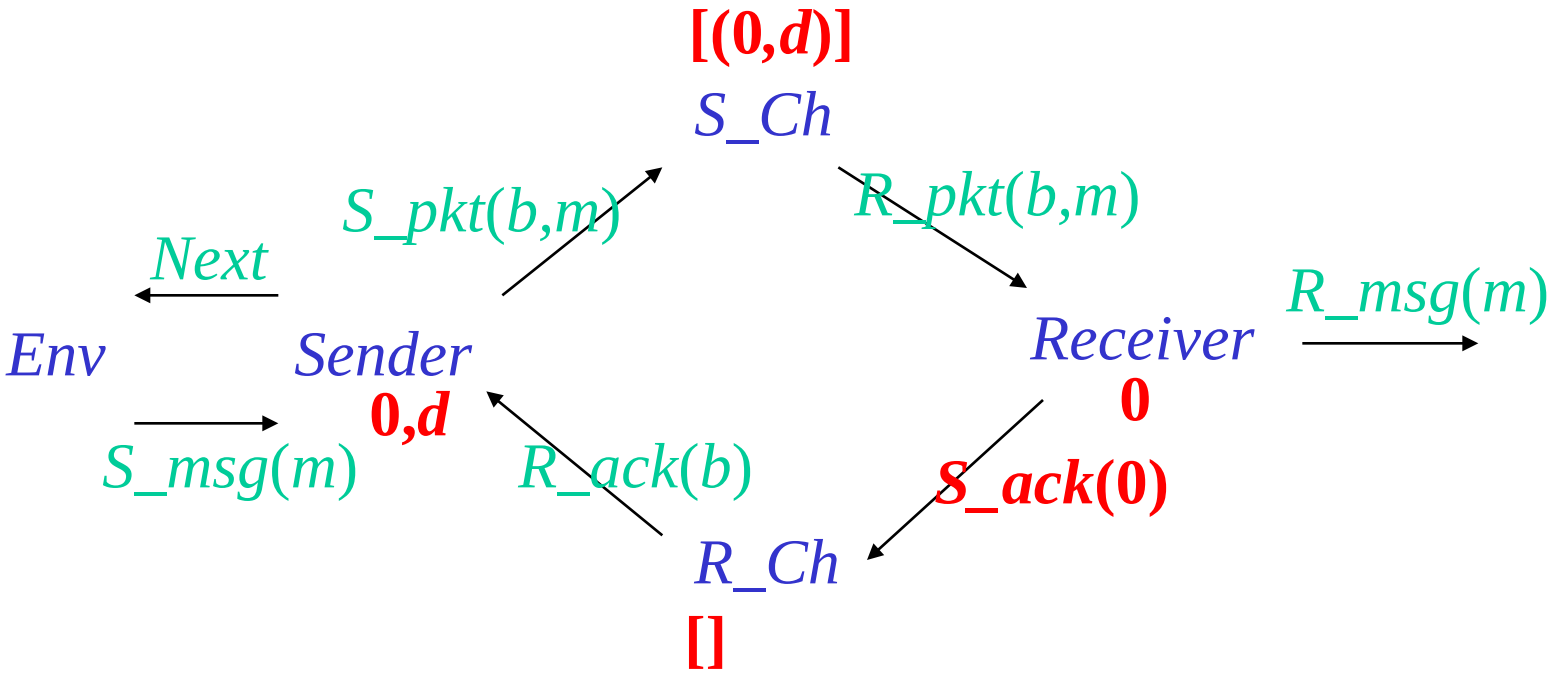
Alternating Bit Protocol



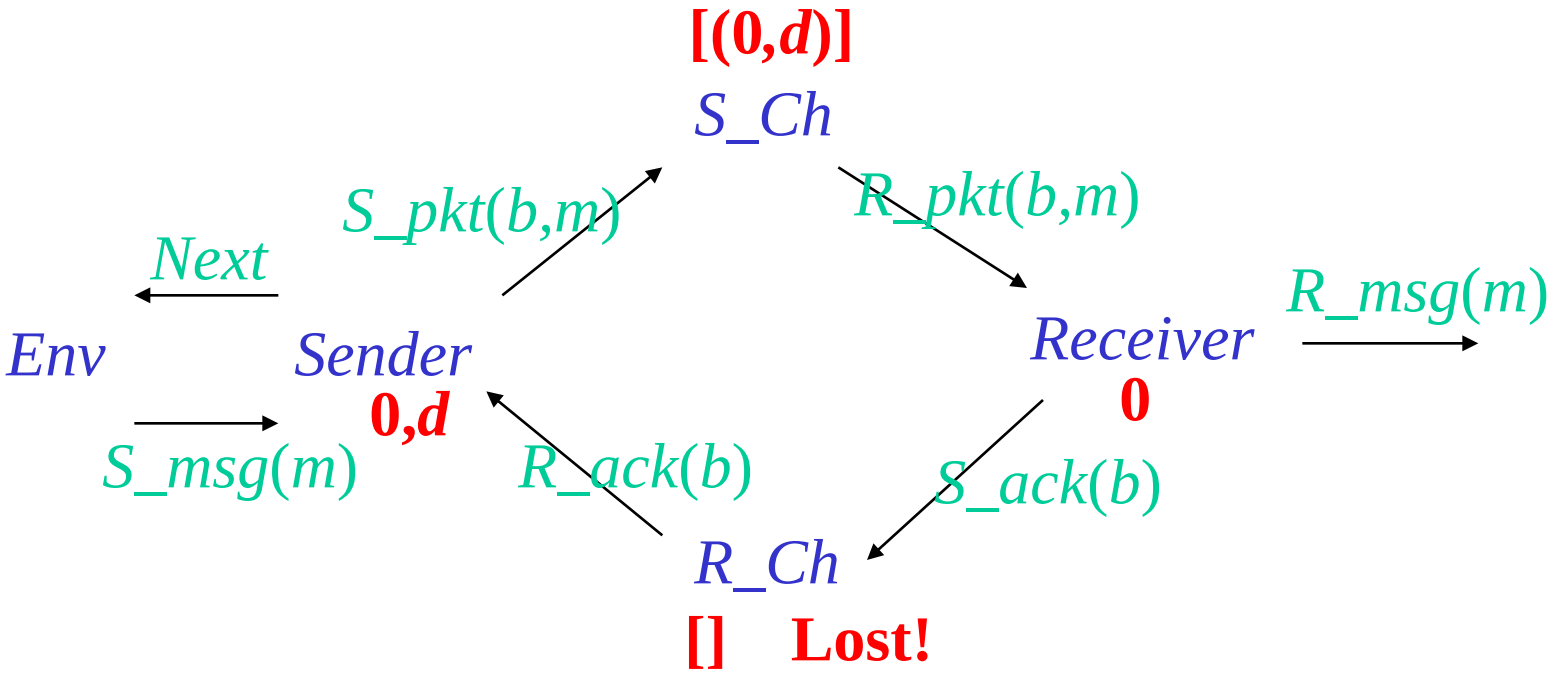
Alternating Bit Protocol



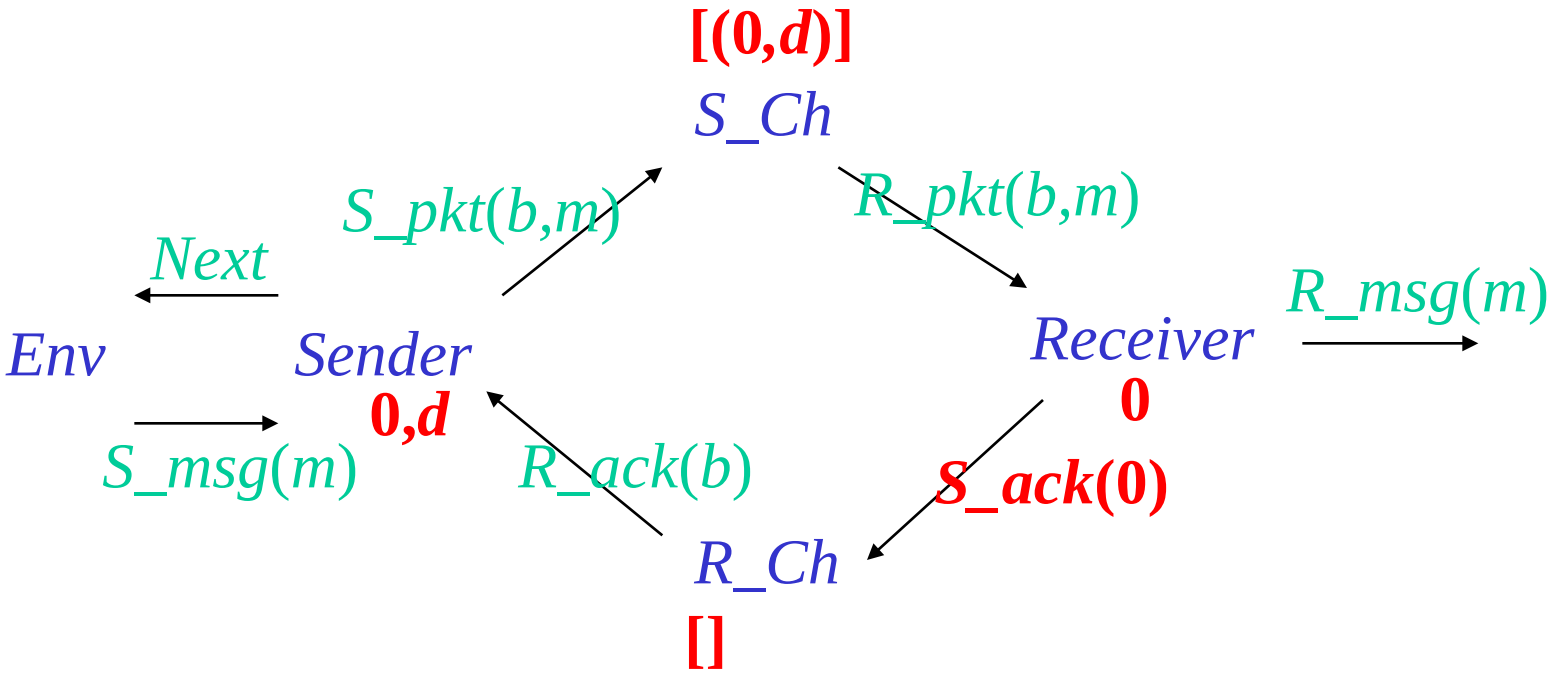
Alternating Bit Protocol



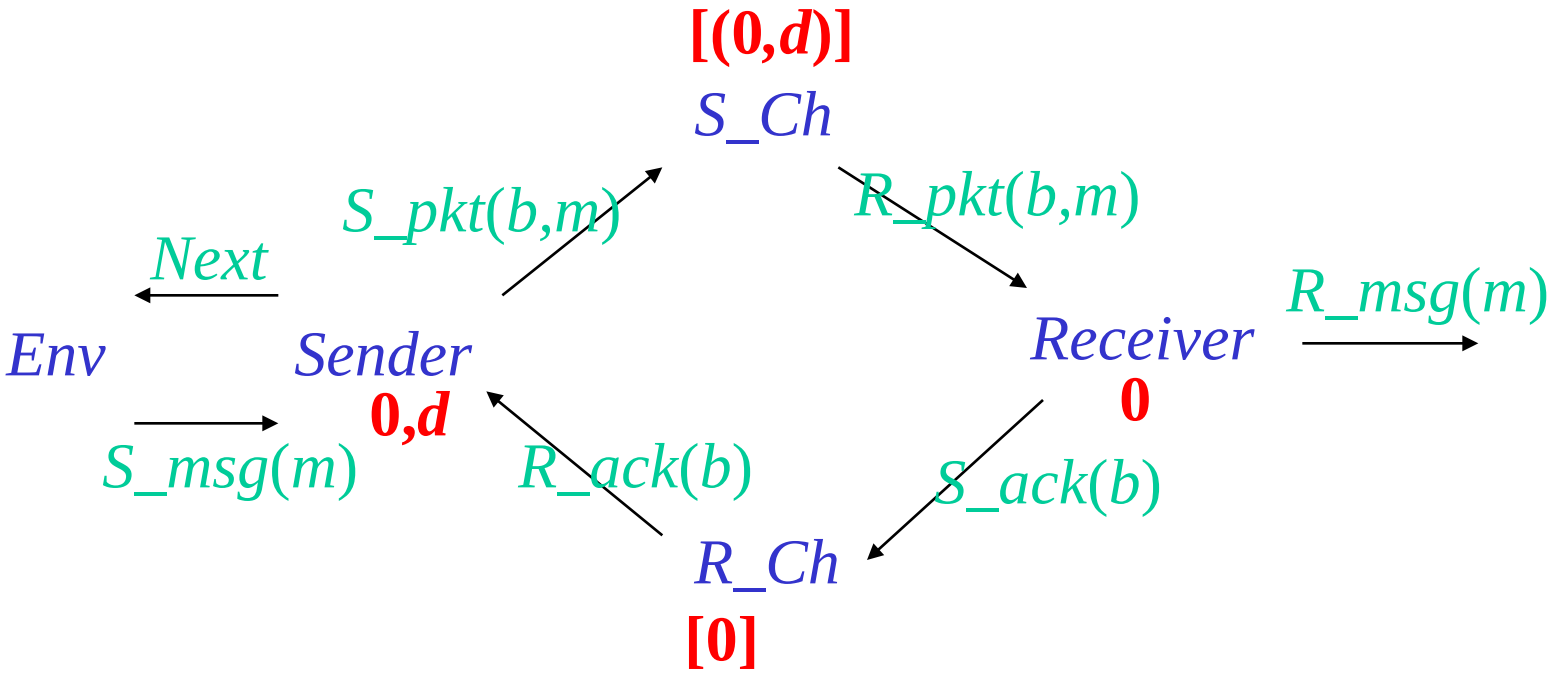
Alternating Bit Protocol



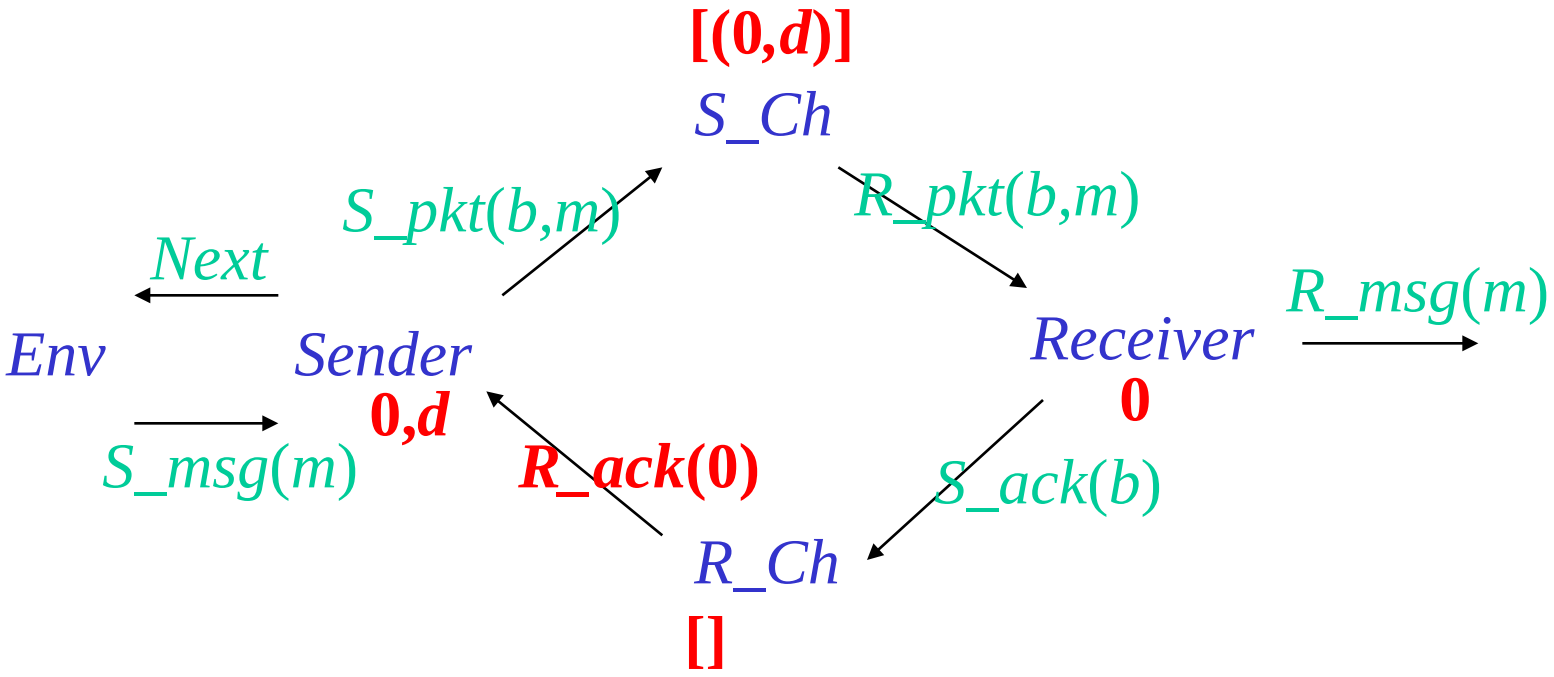
Alternating Bit Protocol



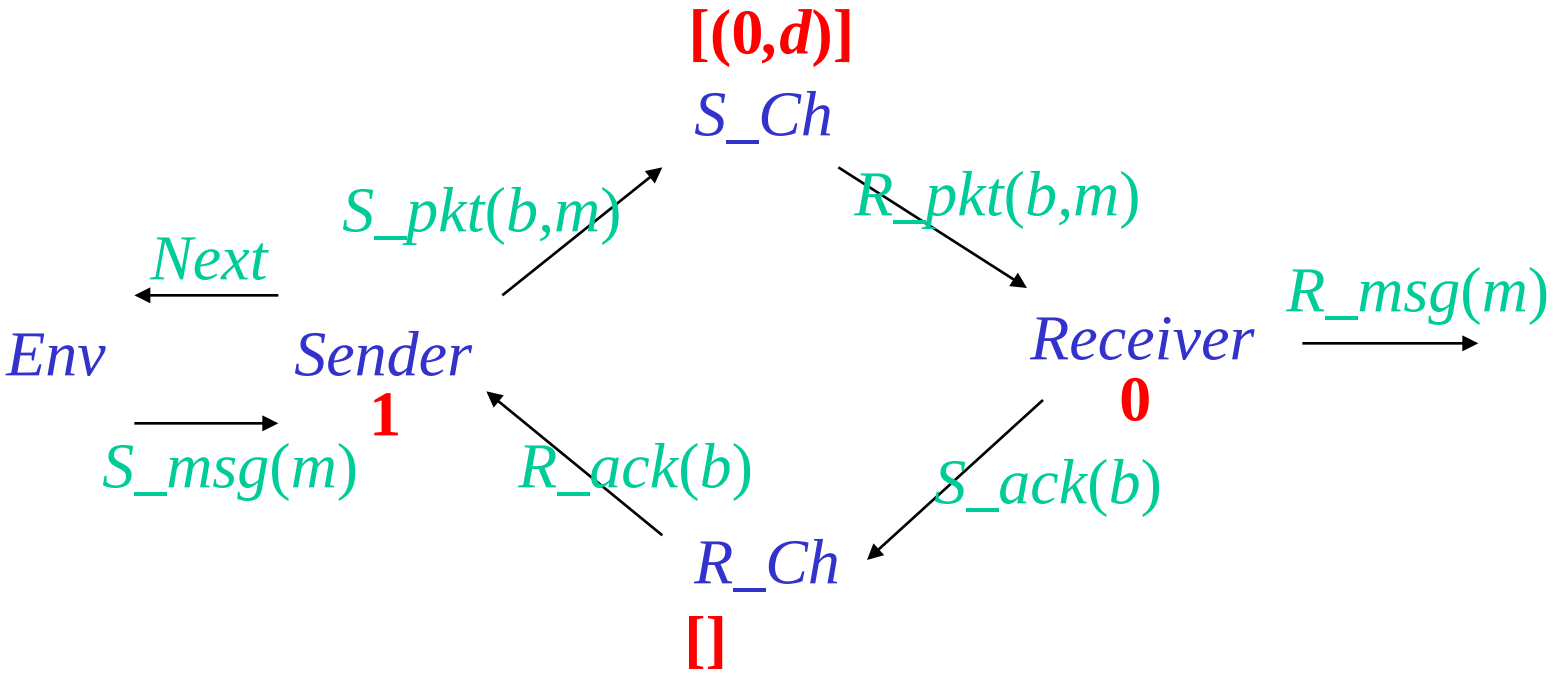
Alternating Bit Protocol



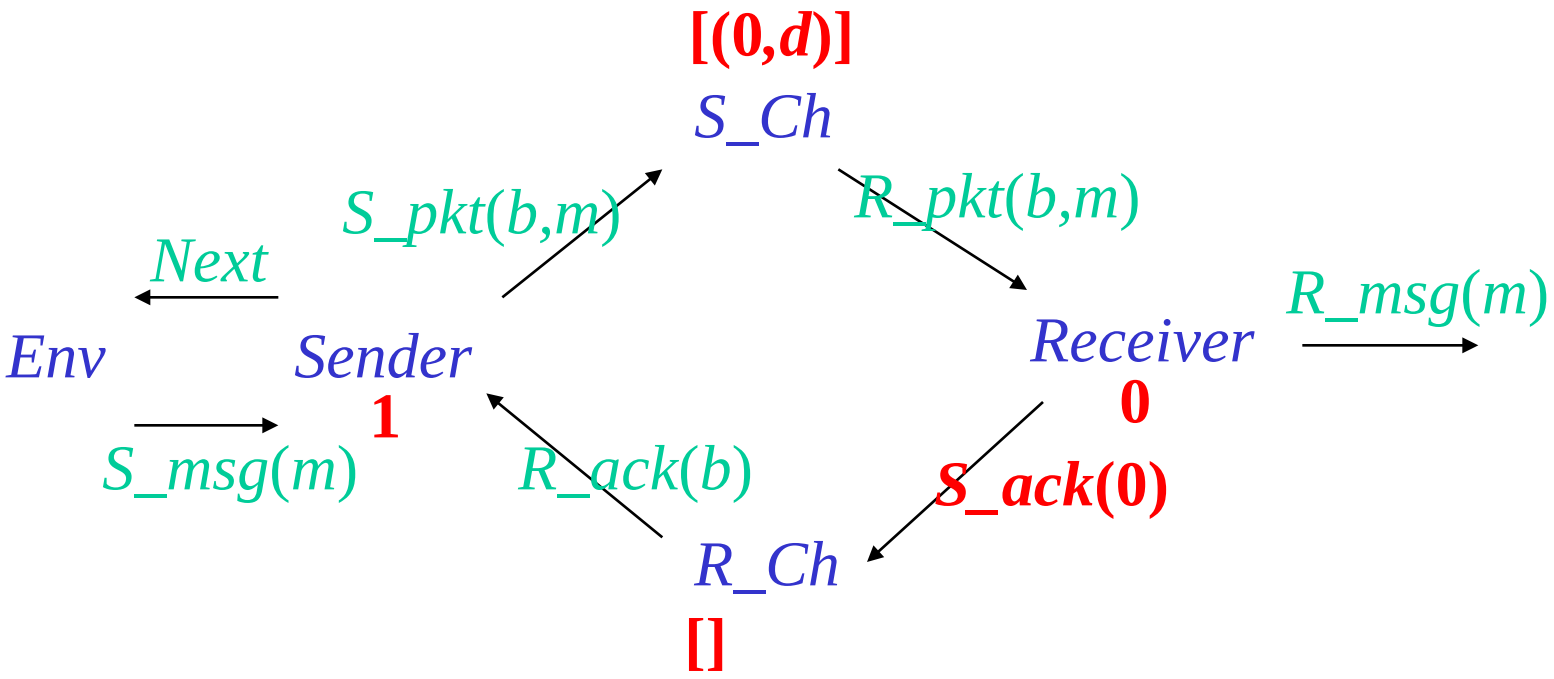
Alternating Bit Protocol



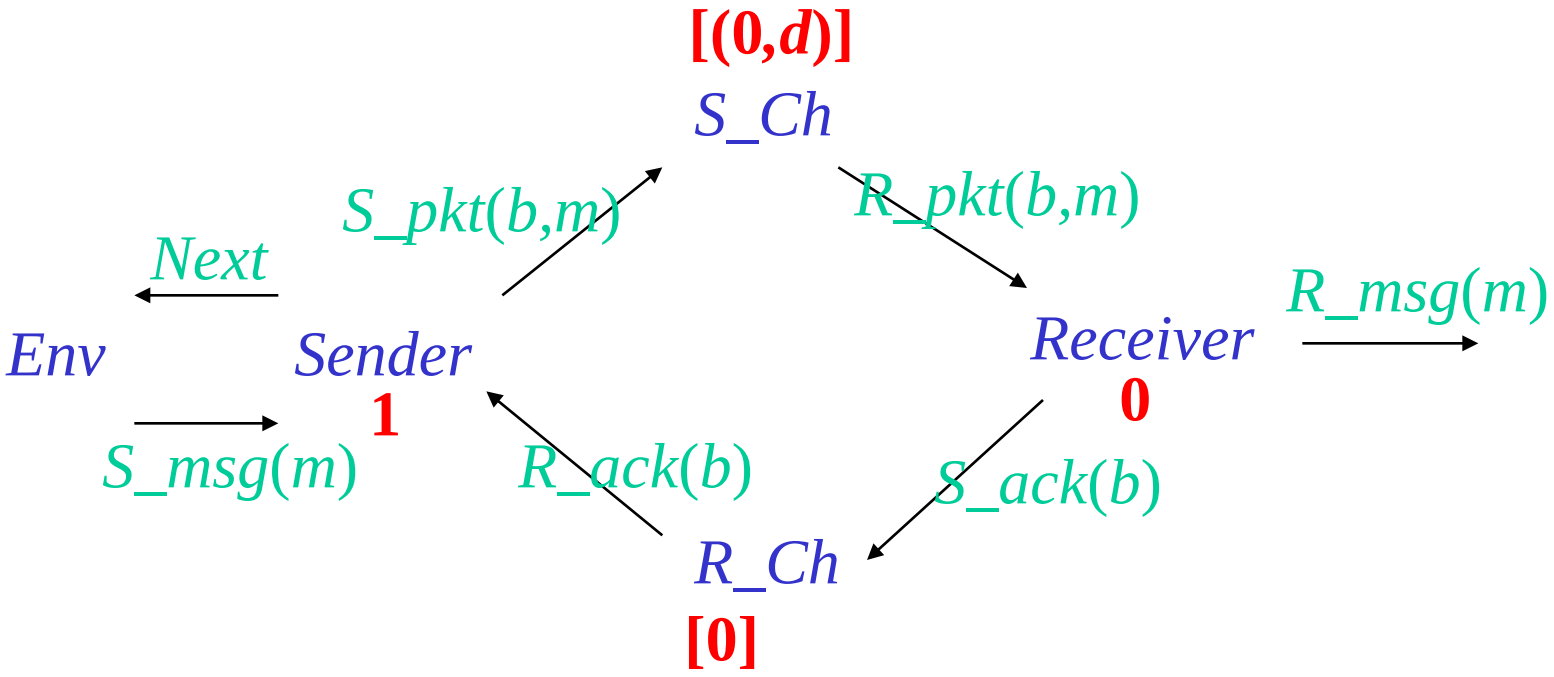
Alternating Bit Protocol



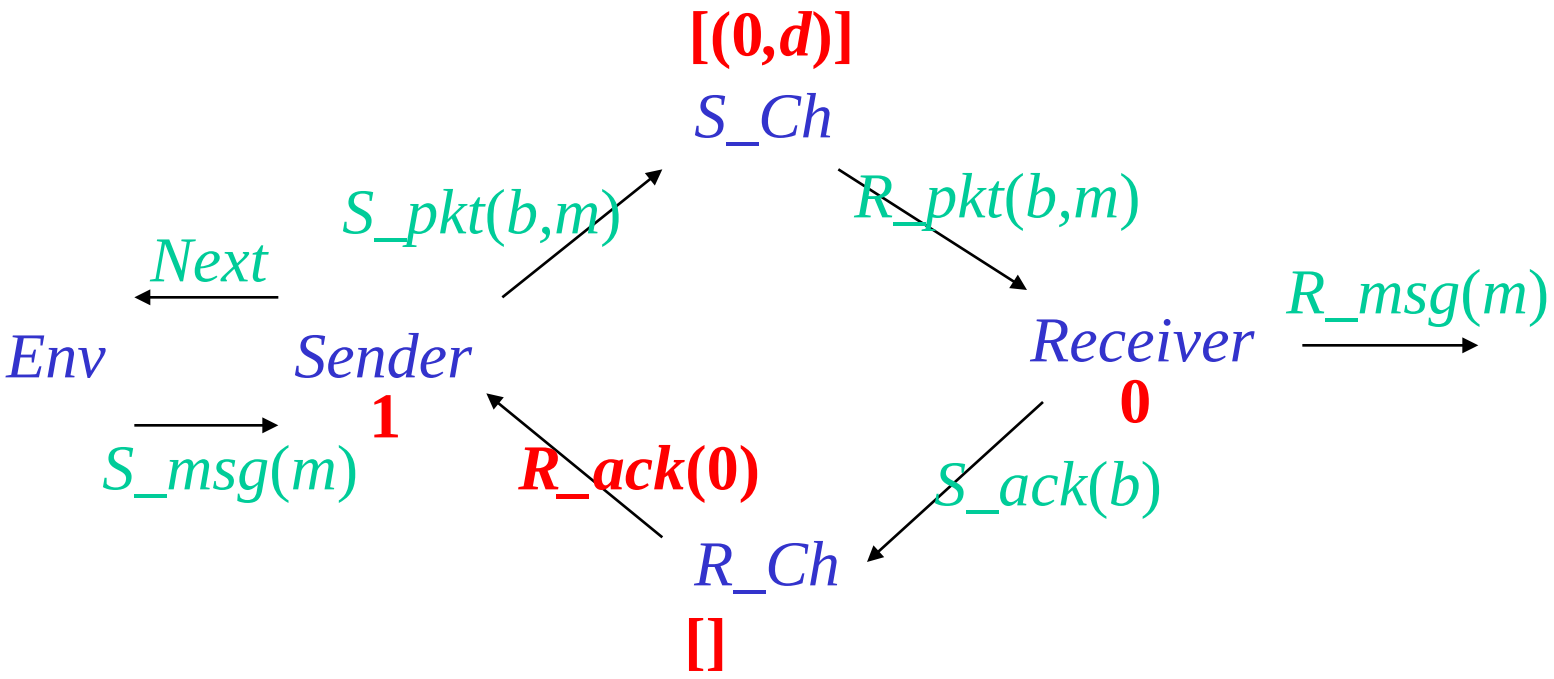
Alternating Bit Protocol



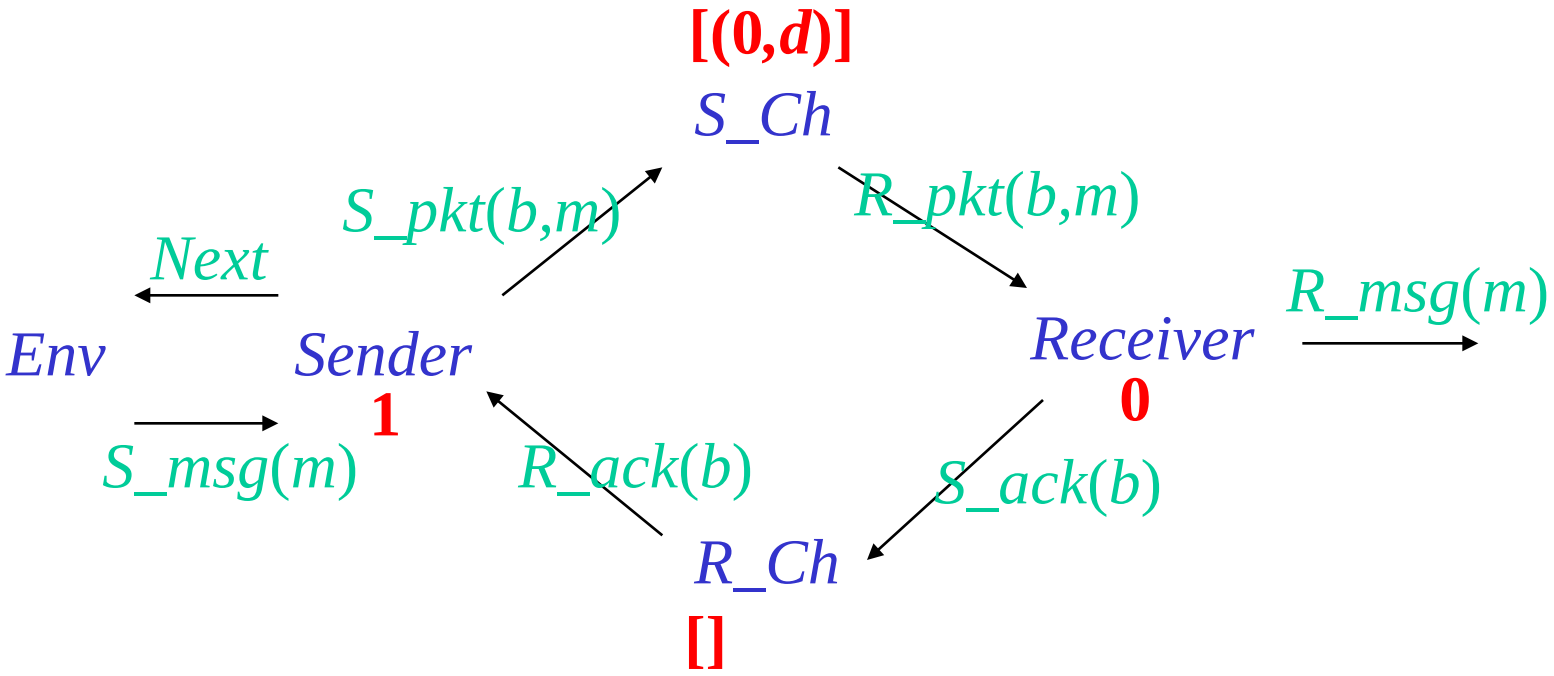
Alternating Bit Protocol



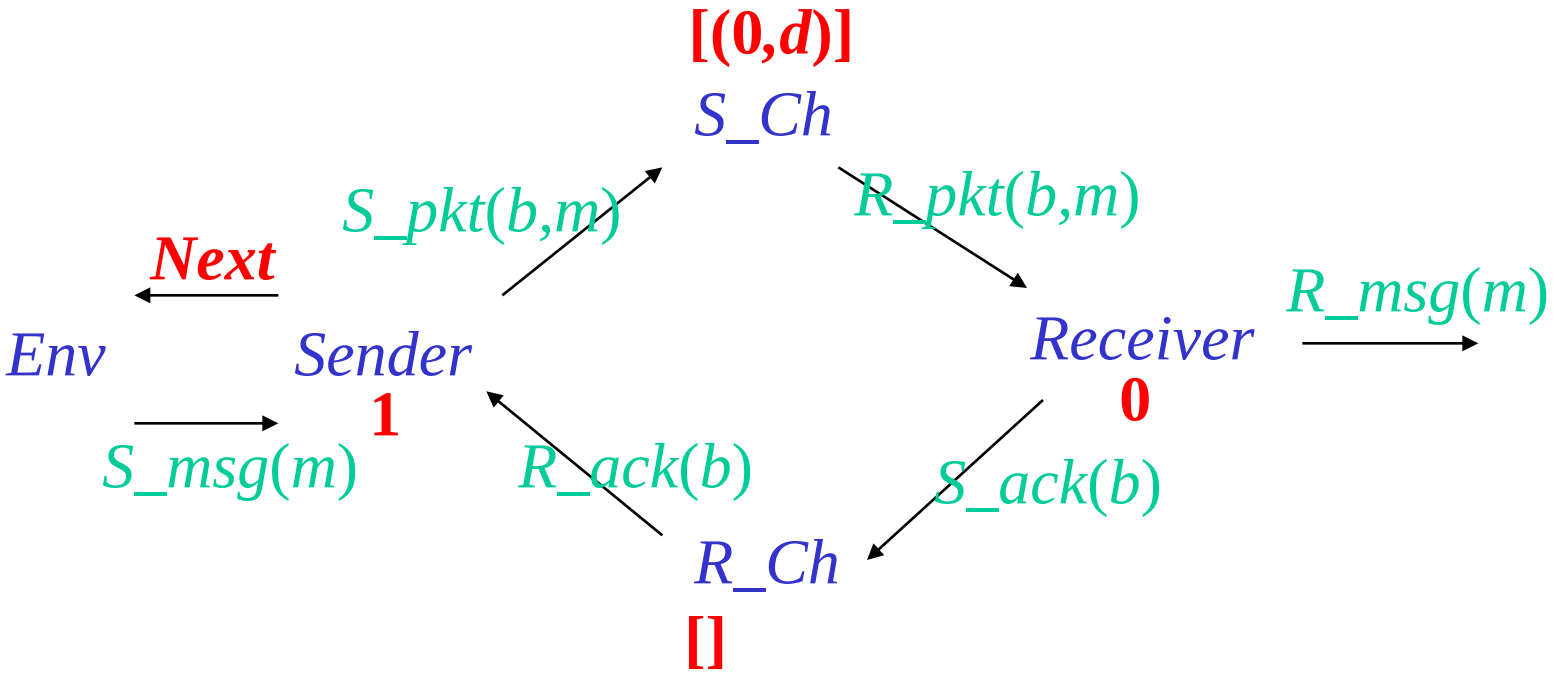
Alternating Bit Protocol



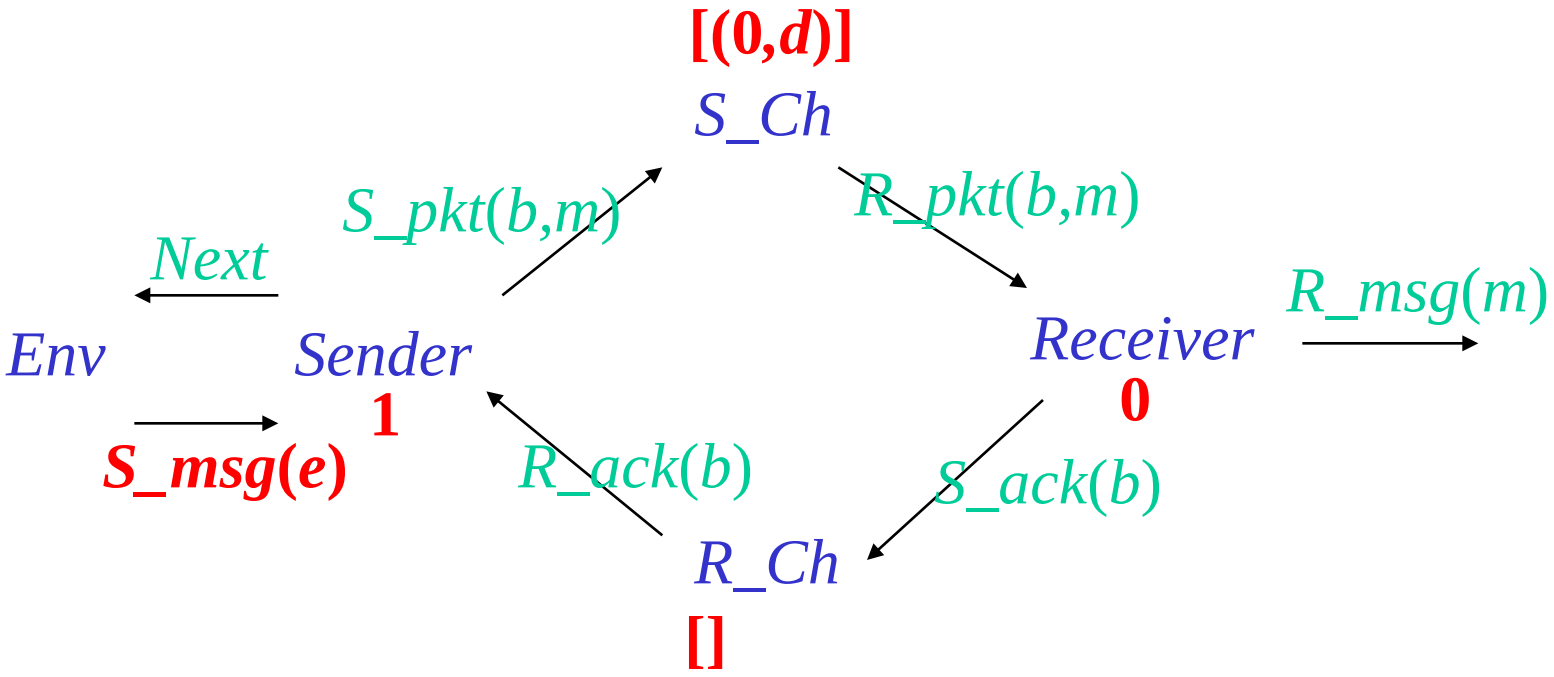
Alternating Bit Protocol



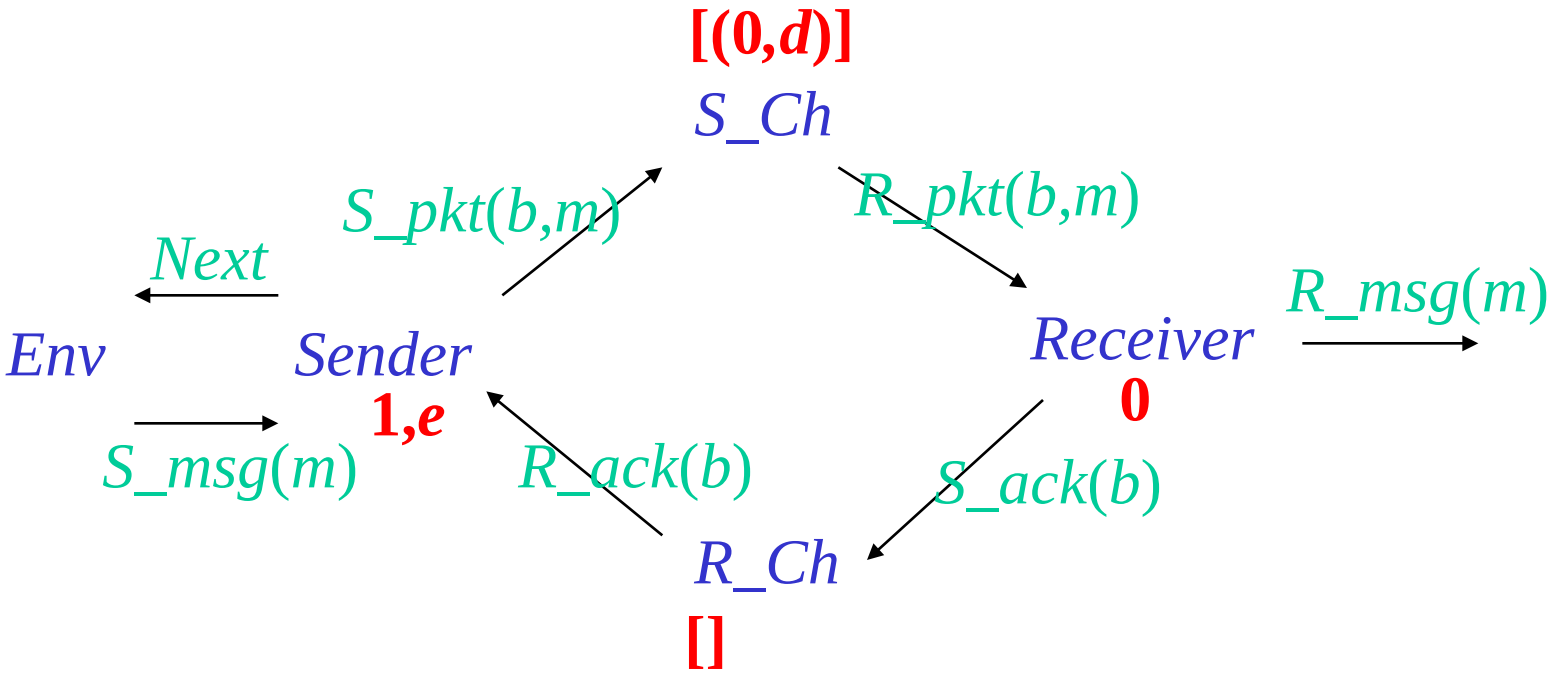
Alternating Bit Protocol



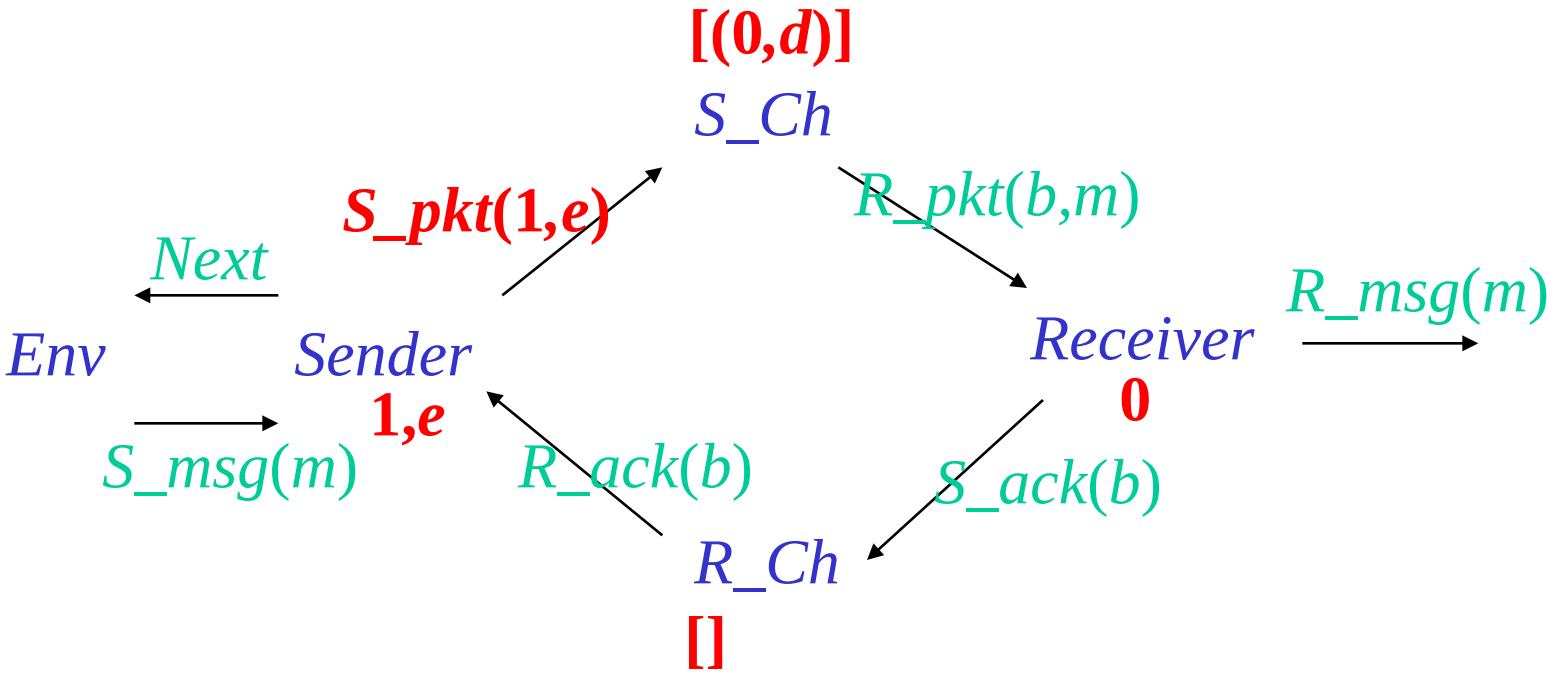
Alternating Bit Protocol



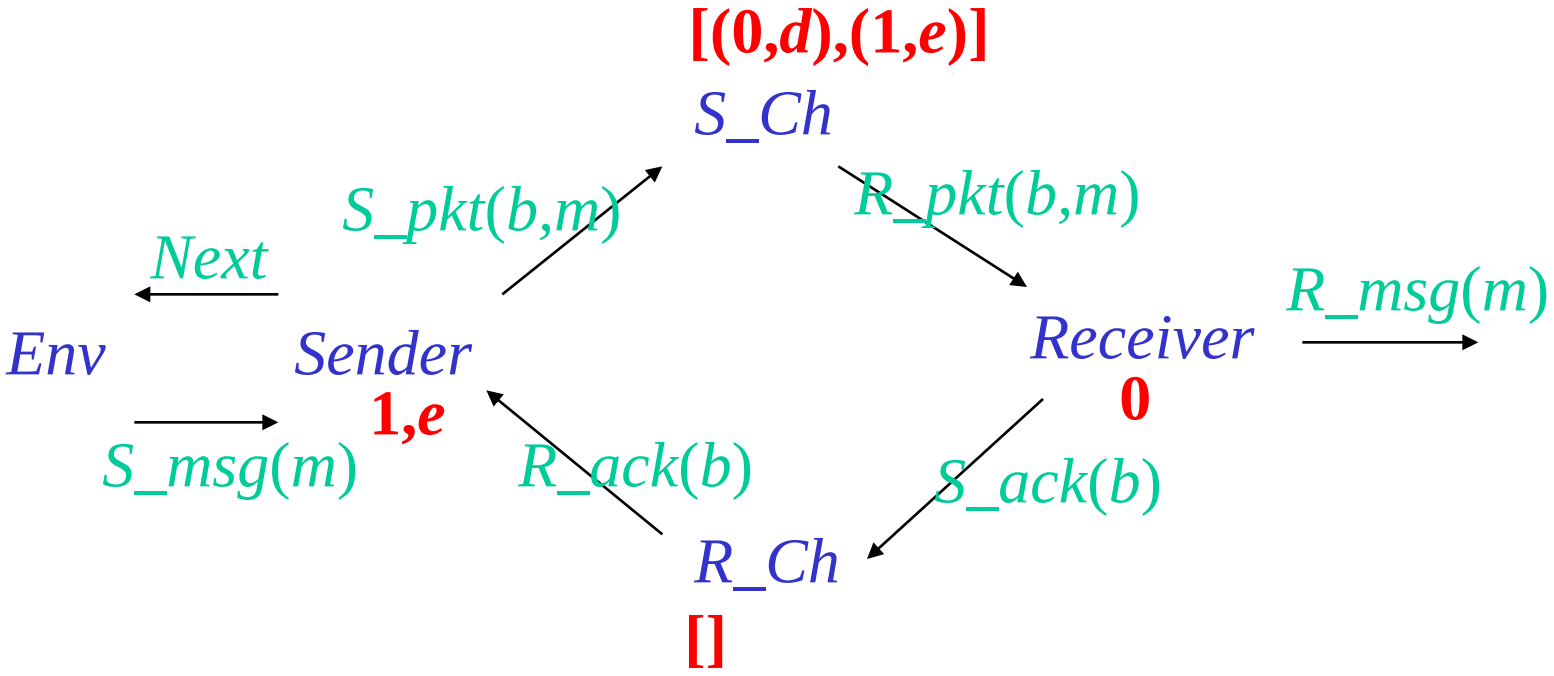
Alternating Bit Protocol



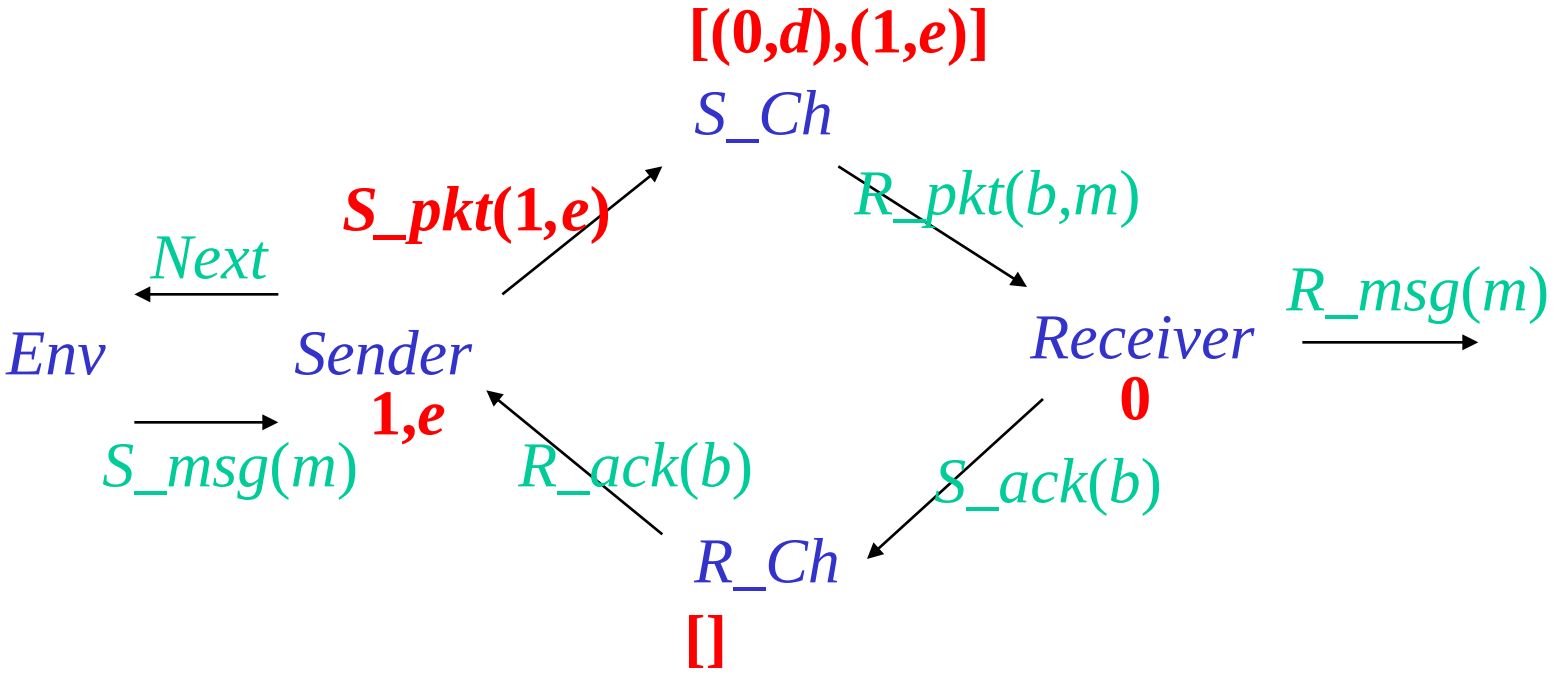
Alternating Bit Protocol



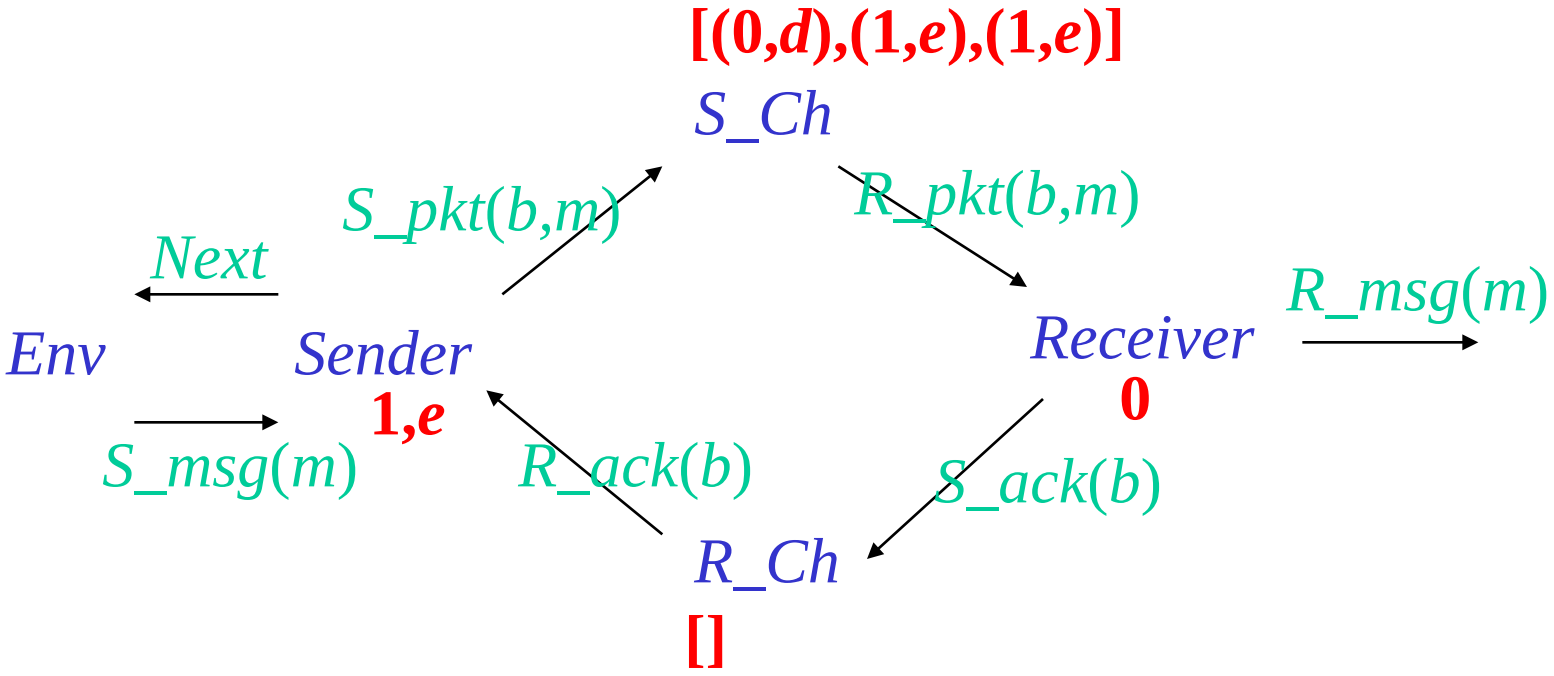
Alternating Bit Protocol



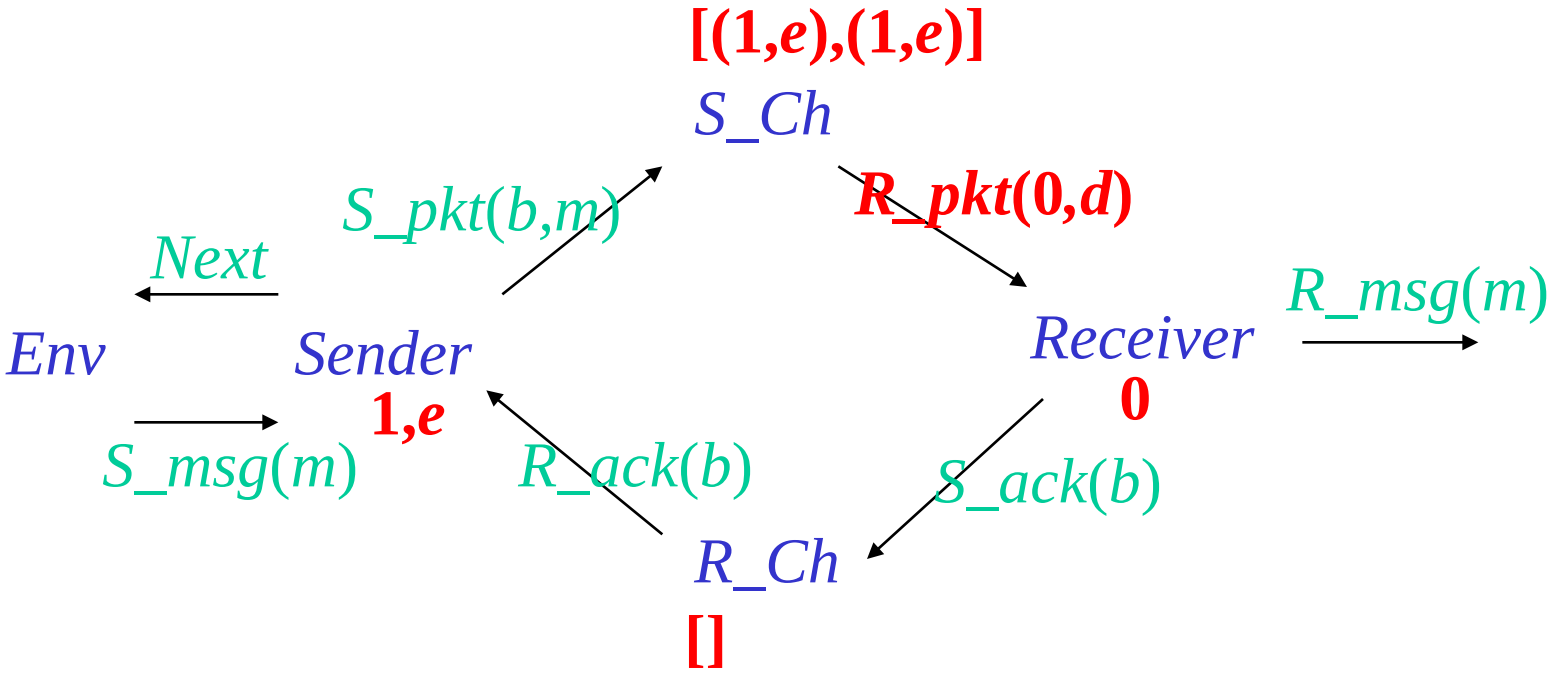
Alternating Bit Protocol



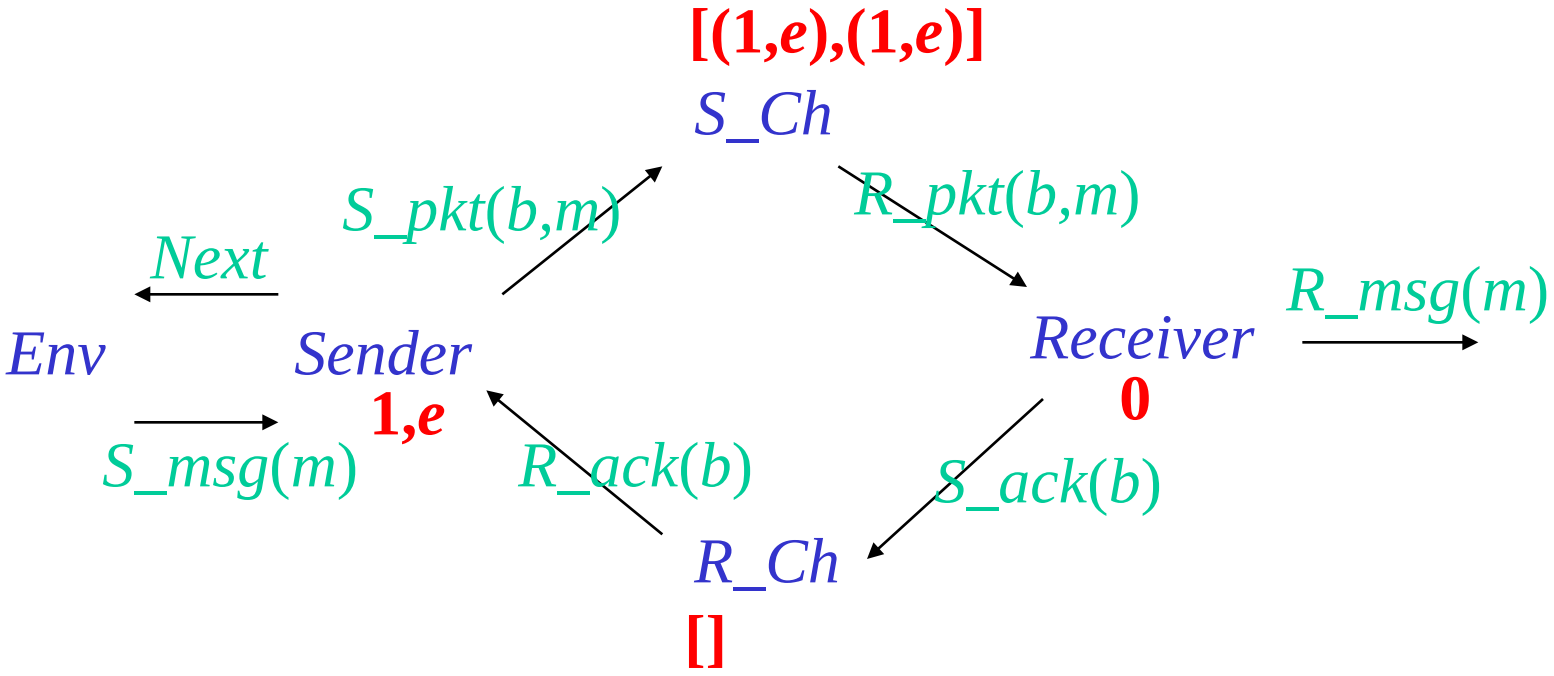
Alternating Bit Protocol



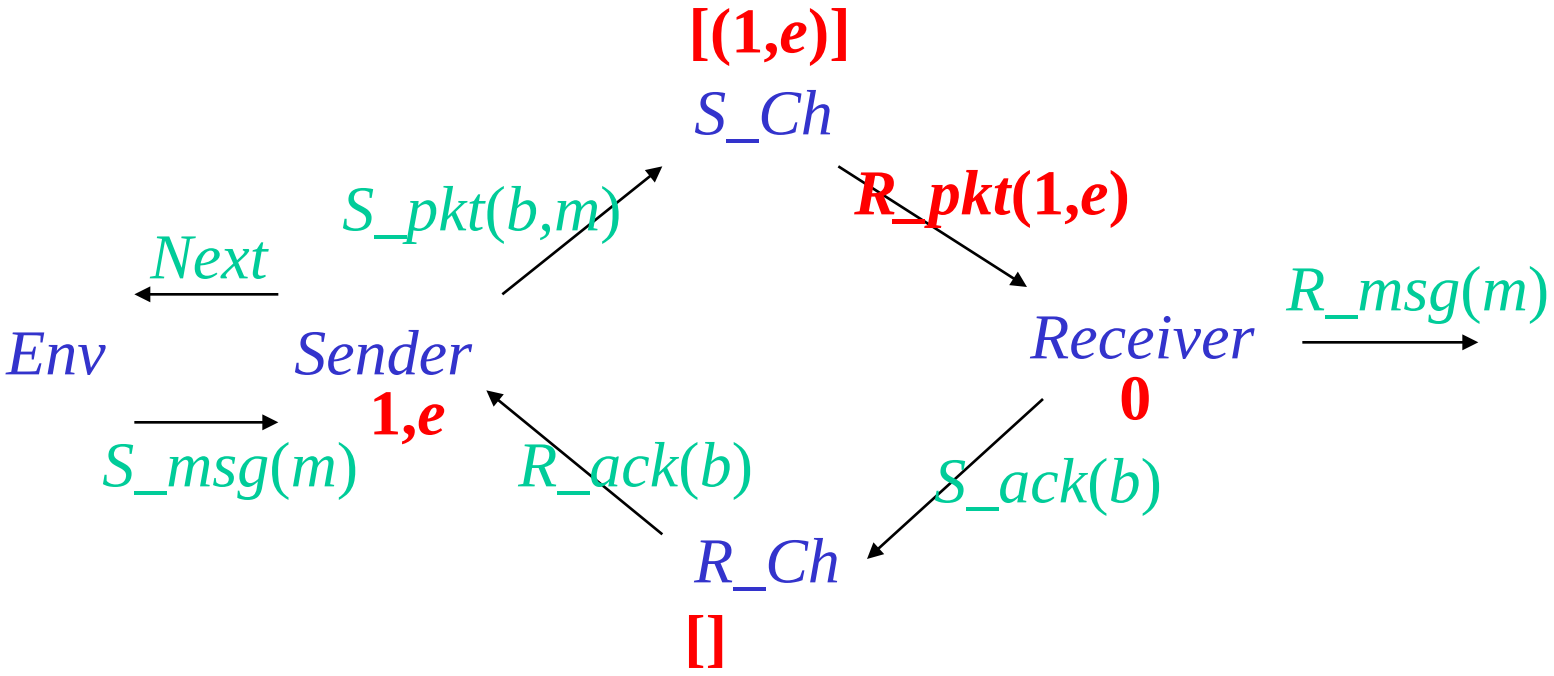
Alternating Bit Protocol



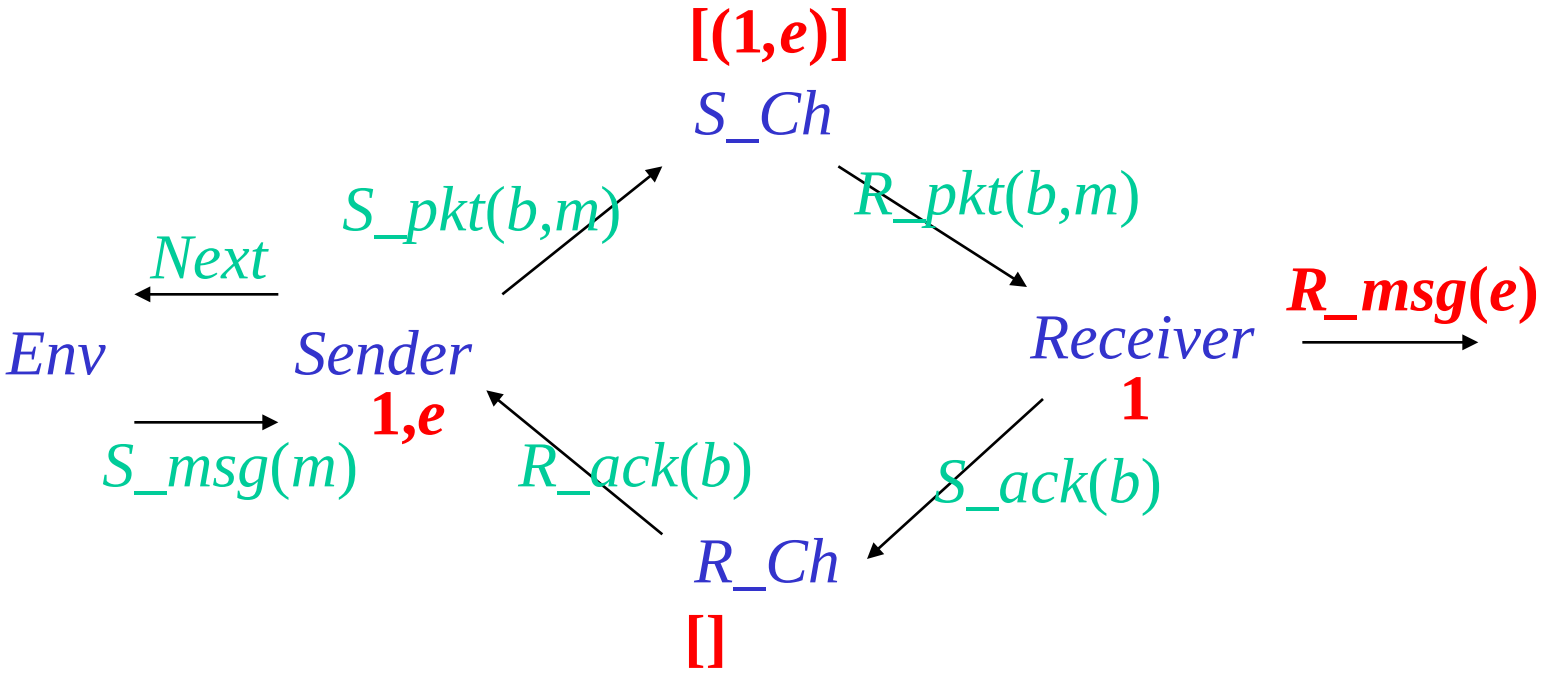
Alternating Bit Protocol



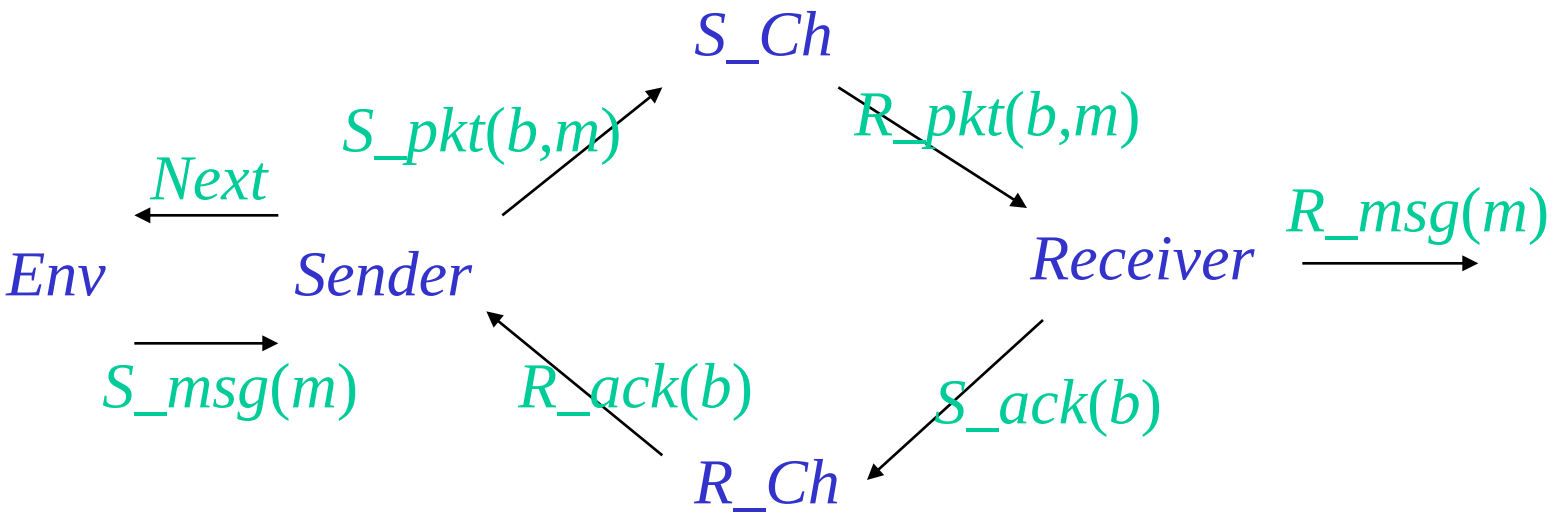
Alternating Bit Protocol



Alternating Bit Protocol



Alternating Bit Protocol

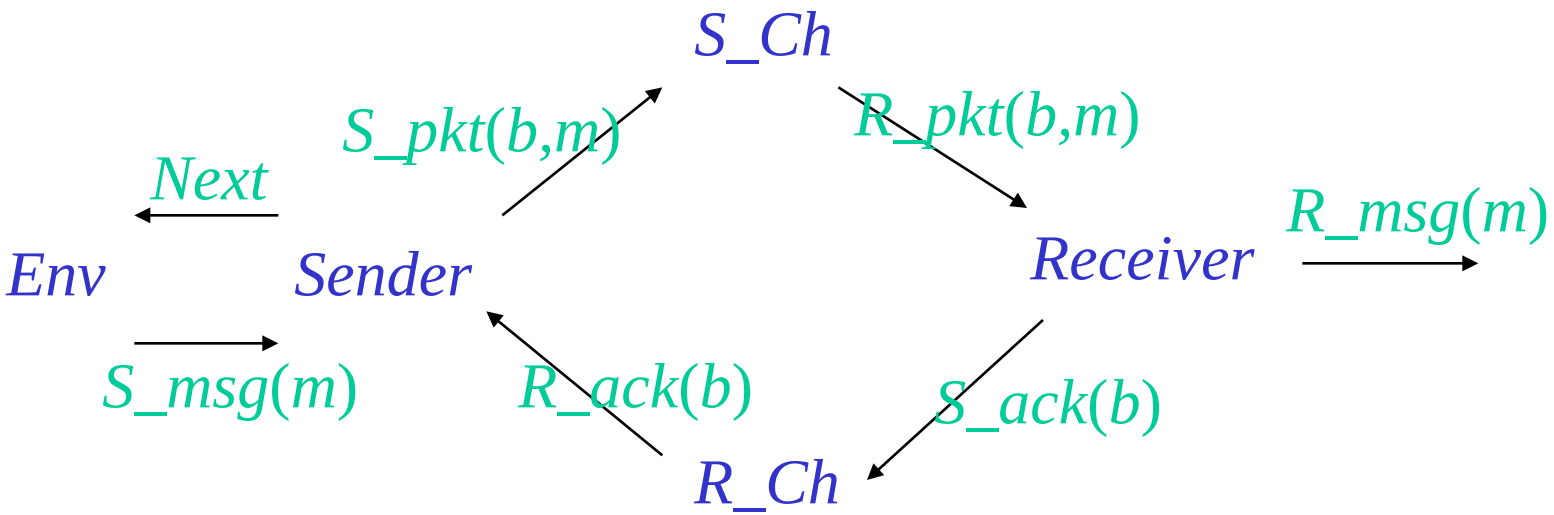


Sender state: *message* : *None* or *Some*(*m*)

header : 0 or 1

trans.:
- if *message*=*None* then
  send *Next* ; get *S_msg*(*m*) ; *message* := *Some*(*m*)
- if *message*=*Some*(*m*) then send *S_pkt*(*header*,*m*)
- when *R_ack*(*b*) is received
  if *b*=*header* then flip *header* ; *message* := *None*

Alternating Bit Protocol

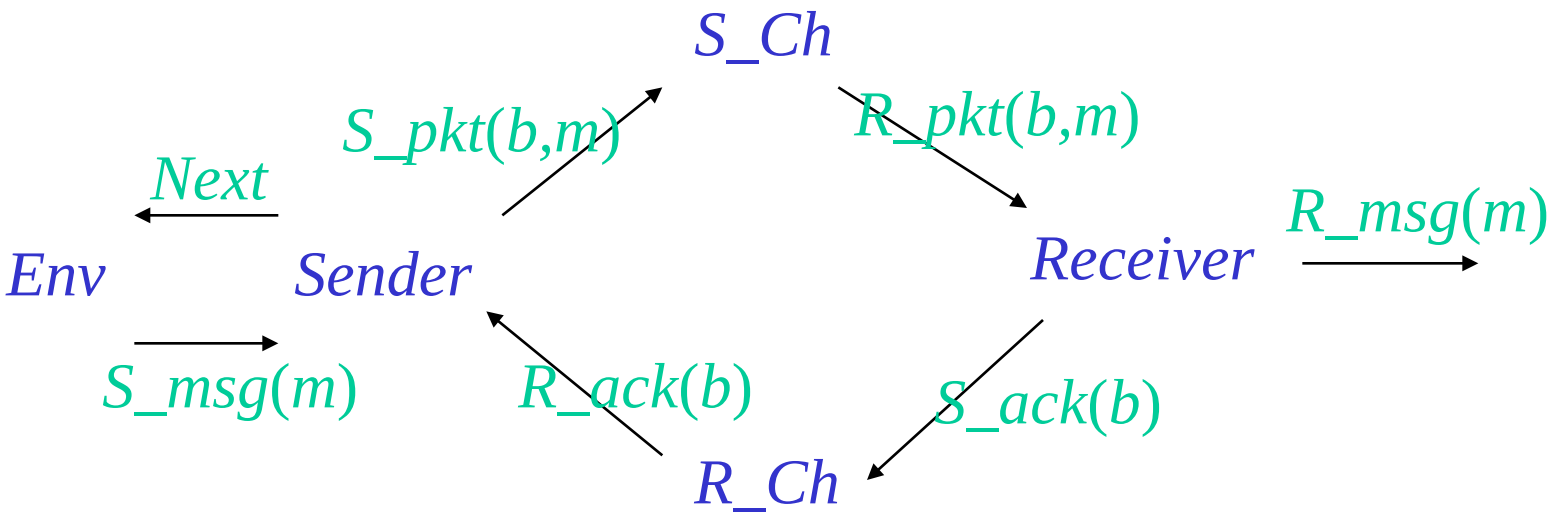


Receiver

state: *header* : 0 or 1

trans.: • when *R_pkt*(*b*,*m*) is received
if *b*≠*header* then
send *R_msg*(*m*) ; flip *header*
• send *S_ack*(*header*)

Alternating Bit Protocol



S_Ch state: queue of packets

trans.:
- when *bm*(=*S_pkt*(*b*,*m*)) is received
  enqueue *bm* or ignore *bm* (*bm* is lost)
- if the queue is not empty
  let *bm*(=…) be the first packet ; send *R_pkt*(*b*,*m*)
  dequeue *bm* or do not dequeue *bm*
  (*bm* is duplicated)

Concrete States

- Queues may grow unlimitedly
  - An element of a queue may be duplicated indefinitely
- The concrete state space is obviously infinite

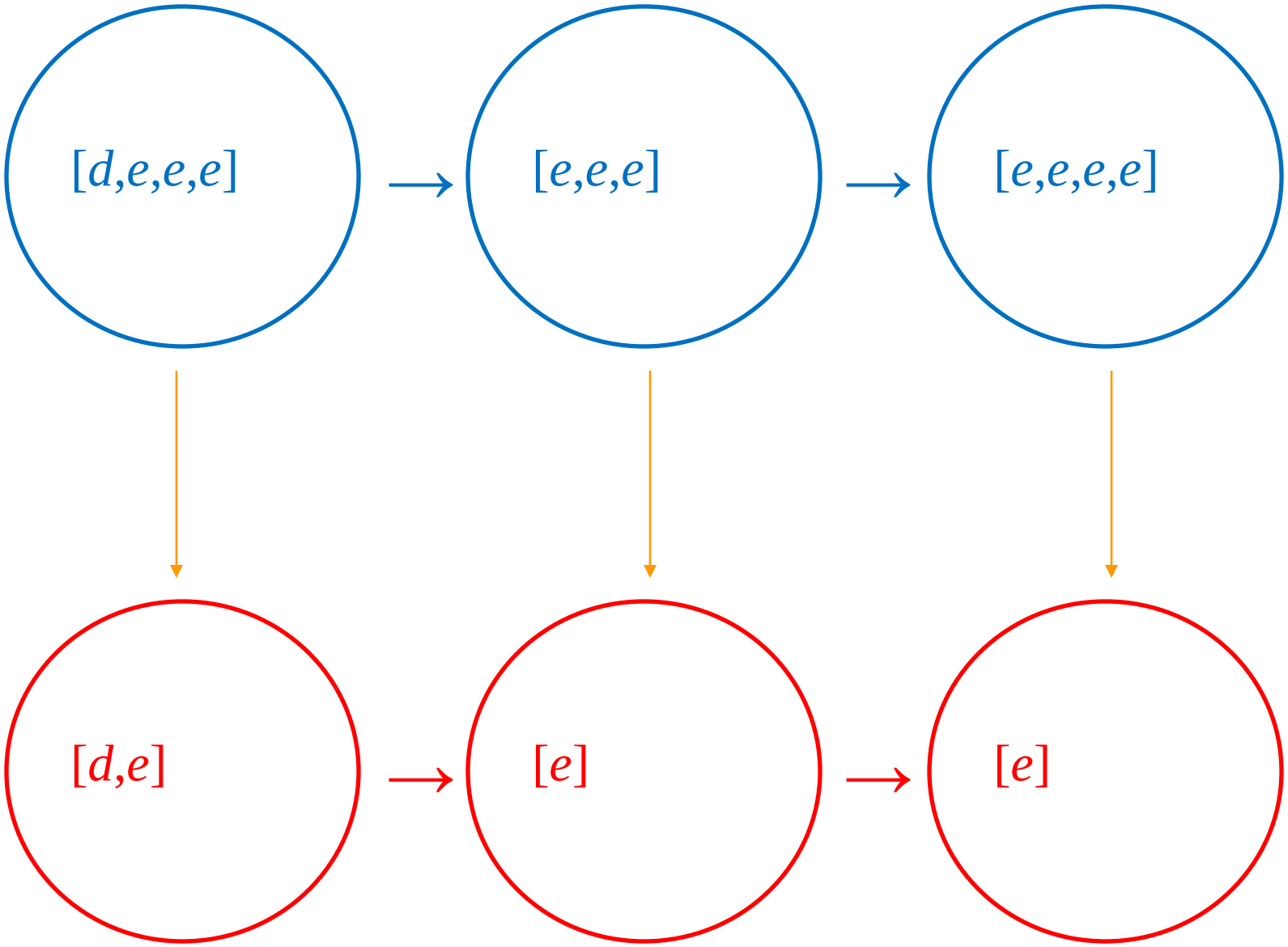
Abstraction of Channels

- Reducing queues by removing duplicated elements in a queue
  - For example, [*d*,*d*,*d*,*d*,*e*,*e*,*e*,*f*] is reduced to [*d*,*e*,*f*]
- Accordingly, transitions of channels are abstracted

Abstract S_Ch

state: reduced queue of packets

trans.:
- when *bm*(=*S_pkt*(*b*,*m*)) is received
  enqueue *bm* ; reduce or ignore *bm* (*bm* is lost)
- if the queue is not empty
  let *bm*(=…) be the first packet ; send *R_pkt*(*b*,*m*)
  dequeue *bm* or do not dequeue *bm*
  (*bm* is duplicated)


[d,e,e,e]
[e,e,e]
[e,e,e,e]
[d,e]
[e]
[e]

Abstract States

- Queues in channels are reduced
- Even though the abstract state space is still infinite in this example,
- Only a finite number of abstract states are reachable from an initial state
  - The length of a reduced queue in a channel is at most 2

Simulation

- If t is an abstraction of s and $s \rightarrow s'$ then there exists an abstraction t' of s' such that $t \rightarrow t'$ (if $s \rightarrow s'$ then $\alpha(s) \rightarrow \alpha(s')$)

Concrete path

$$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n$$

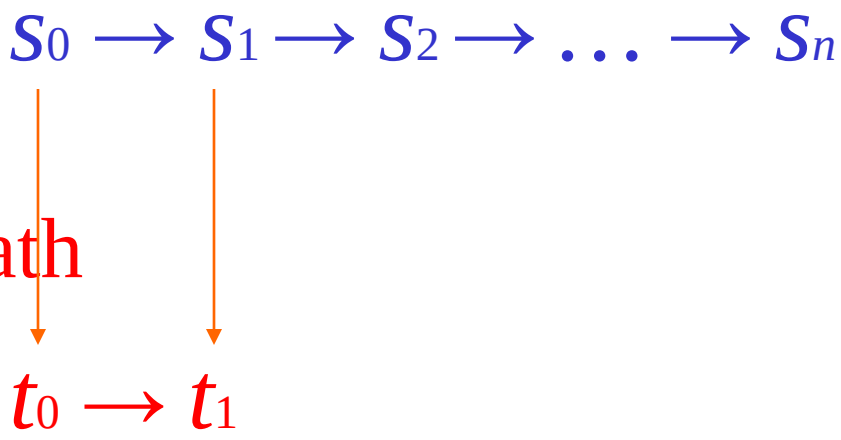
Abstract path

t_0

Simulation

- If t is an abstraction of s and $s \rightarrow s'$ then there exists an abstraction t' of s' such that $t \rightarrow t'$ (if $s \rightarrow s'$ then $\alpha(s) \rightarrow \alpha(s')$)

Concrete path



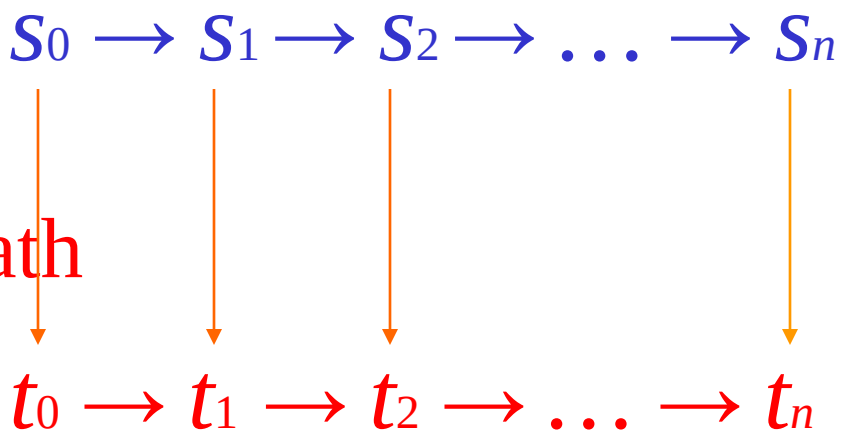
Abstract path

Simulation

- If t is an abstraction of s and $s \rightarrow s'$ then there exists an abstraction t' of s' such that $t \rightarrow t'$ (if $s \rightarrow s'$ then $\alpha(s) \rightarrow \alpha(s')$)

Concrete path

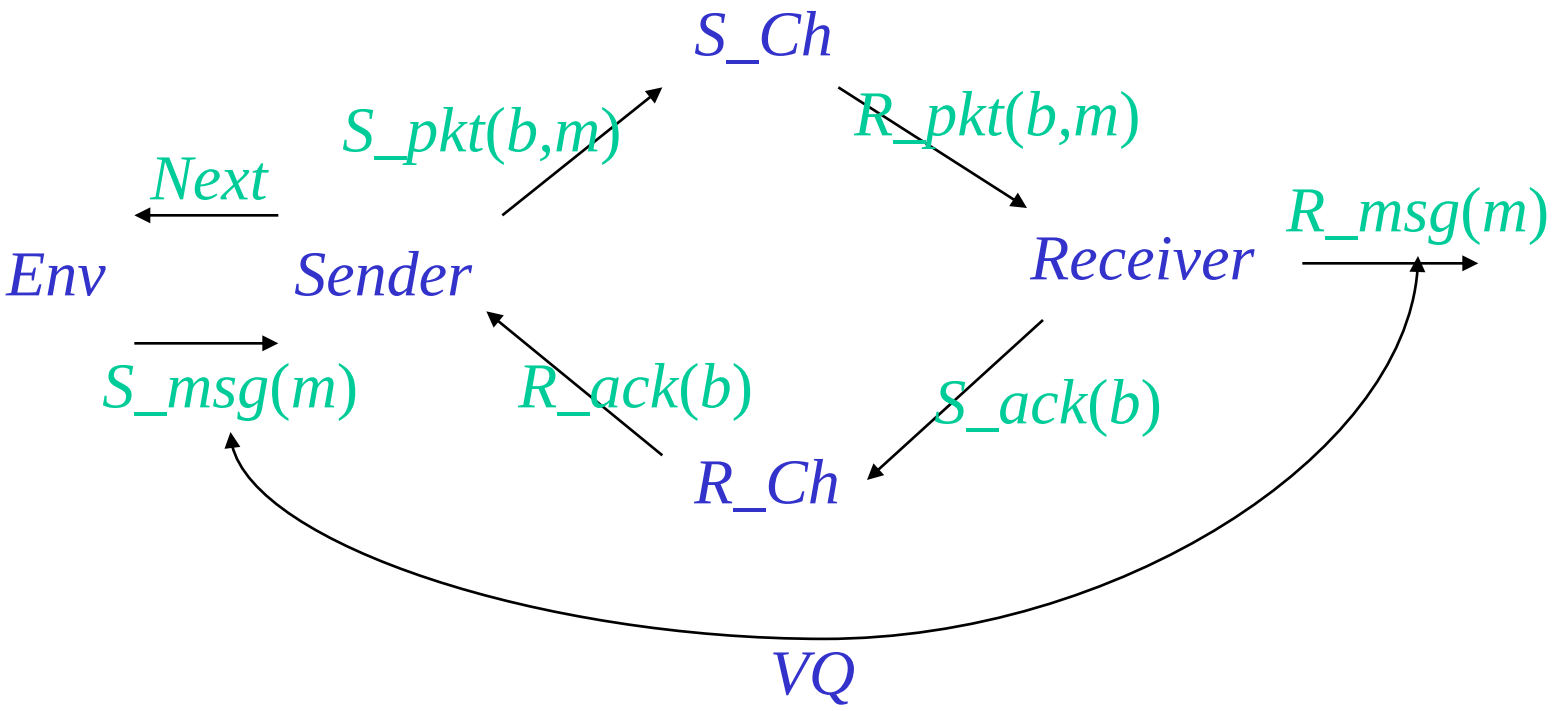
Abstract path



Abstract Model Checking

- If a property holds in any abstract path, then the corresponding property holds in any concrete path
- Example
  - After a transition by $S\_msg(m)$, if there exists a transition by $R\_msg(m')$, then $m=m'$

Alternating Bit Protocol



After a transition by *S_msg*(*m*), if there exists a transition by *R_msg*(*m′*), then *m*=*m′*

(This property is expressed by a virtual queue (*VQ*) that stores *S_msg*(*m*) and compares it with *R_msg*(*m′*))

Microsoft Static Driver Verifier (SDV)

- Hostile model of the driver's execution environment
  - *Harness code* simulates the operating system initializing and invoking the device driver
  - *Stub code* provides the semantics for the kernel APIs
- SLAM Tookit
  - CEGAR
- API usage rules (properties)
  - About 60, described in SLIC

SLAM Toolkit

- Safety verification of system software
  - Target: Device drivers for Windows with well defined interface
- Three phases
  - C2BP: tool for translating (abstracting) C programs to Boolean programs, using predicates in specifications (API usage rules)
  - BEBOP: model checker for Boolean programs
  - NEWTON: tool that analyzes error paths produced by the model checker, and discovers predicates for refining Boolean programs

CEGAR

- Counter
- Example-
- Guided
- Abstraction
- Refinement

```
state {
   enum { Unlocked=0, Locked=1 }
     state = Unlocked;
}


KeAcquireSpinLock.return {
  if (state == Locked)
    abort;
  else
    state = Locked;
}

KeReleaseSpinLock.return {
  if (state == Unlocked)
    abort;
  else
    state = Unlocked;
}
```

SLIC spec. Specification Language for Interface Checking

```
enum { Unlocked=0, Locked=1 }
  state = Unlocked;

void slic_abort() {
  SLIC_ERROR: ;
}

void KeAcquireSpinLock_return {
  if (state == Locked)
    slic_abort();
  else
    state = Locked;
}

void KeReleaseSpinLock_return {
  if (state == Unlocked)
    slic_abort();
  else
    state = Unlocked;
}
```

C program obtained by compiling SLIC spec.

```
void example() {
 do {
  KeAcquireSpinLock();

  nPacketsOld = nPackets;
  req = devExt->WLHV;
  if(req && req->status) {
    devExt->WLHV = req->Next;
    KeReleaseSpinLock();

    irp = req->irp;
    if(req->status > 0){
      irp->IoS.Status = SUCCESS;
      irp->IoS.Info = req->Status;
    } else {
      irp->IoS.Status = FAIL;
      irp->IoS.Info = req->Status;
    }
    SmartDevFreeBlock(req);
    IoCompleteRequest(irp);
    nPackets++;
  }
 } while(nPackets!=nPacketsOld);
 KeReleaseSpinLock();

}
```

Code for a device driver

```
void example() {
 do {
  KeAcquireSpinLock();
A:KeAcquireSpinLock_return();
  nPacketsOld = nPackets;
  req = devExt->WLHV;
  if(req && req->status) {
    devExt->WLHV = req->Next;
    KeReleaseSpinLock();
B:  KeReleaseSpinLock_return();
    irp = req->irp;
    if(req->status > 0){
      irp->IoS.Status = SUCCESS;
      irp->IoS.Info = req->Status;
    } else {
      irp->IoS.Status = FAIL;
      irp->IoS.Info = req->Status;
    }
    SmartDevFreeBlock(req);
    IoCompleteRequest(irp);
    nPackets++;
  }
 } while(nPackets!=nPacketsOld);
 KeReleaseSpinLock();
C:KeReleaseSpinLock_return();
}
```

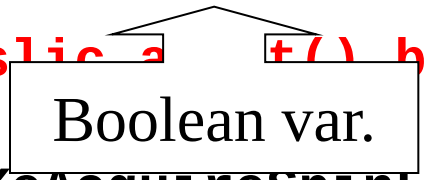
Device driver code with inserted checks for specification

```
decl {state==Locked}, {state==Unlocked};

void slic_a    t() begin SLIC    OR: skip; end

void KeAcquireSpinLock_return
begin
  if ({state==Locked})
    slic_abort();
  else
    {state==Locked},{state==Unlocked} := T,F;
begin

void KeReleaseSpinLock_return
begin
  if ({state==Unlocked})
    slic_abort();
  else
    {state==Locked},{state==Unlocked} := F,T;
end
```



Boolean program obtained from SLIC spec.

```
decl bL, bU;

void slic_abort() begin SLIC_ERROR: skip; end

void KeAcquireSpinLock_return
begin
  if (bL)
    slic_abort();
  else
    bL,bU := T,F;
begin

void KeReleaseSpinLock_return
begin
  if (bU)
    slic_abort();
  else
    bL,bU := F,T;
end
```

Boolean program obtained from SLIC spec.

```
void example() begin
 do {
  skip();
A:KeAcquireSpinLock_return();
  skip;
  skip;
  if(*) then
    skip;
    skip();
B:  KeReleaseSpinLock_return();
    skip;
    if(*) then
      skip;
      skip;
    else
      skip;
      skip;
    fi
    skip;
    skip;
    skip;
  }
 } while(*);
 skip();
C:KeReleaseSpinLock_return();
end
```

Boolean program obtained from device driver code

***** : undetermined

Model Checking

- Find a path that reaches "**SLIC_ERROR**"
- In this case, an error path

  A, **A**, **SLIC_ERROR**

  is found
- Verify if the error path is valid with respect to the original C program
  - Verification condition generation（VCGen）
- In this case, it is not valid because the predicate

  nPackets==nPacketsOld

  is both true and false in the path

Re-abstraction

- Use the predicate **`nPackets==nPacketsOld`**
- The statement **`nPacketsOld = nPackets`** makes the predicate true
- The statement **`nPackets++`** makes the predicate false if it is now true (detected by the theorem prover)

```
bool choose(pos, neg)
begin
     if (pos) then return T; elsif (neg) then return F;
  elsif (*)   then return T; else             return F; fi
end
```

```
void example() begin
 do {
  skip();
A:KeAcquireSpinLock_return();
  b := T;
  skip;
  if(*) then
    skip;
    skip();
B:  KeReleaseSpinLock_return();
    skip;
    if(*) then
      skip;
      skip;
    else
      skip;
      skip;
    fi
    skip;
    skip;
    b := choose(F,b);
  }
 } while(!b);
 skip();
C:KeReleaseSpinLock_return();
end
```

Boolean program obtained by re-abstraction (refinement)

b:

nPackets==nPacketsOld

Model Checking Again

- In this case, there is no error path that reaches "**`SLIC_ERROR`**"
- Loop invariant :

$$(\text{state} = \text{Locked} \wedge \text{nPackets} = \text{nPacketsOld})$$

$$\vee\ (\text{state} = \text{Unlocked} \wedge \text{nPackets} \neq \text{nPacketsOld})$$

References

– T. Ball, and S. K. Rajamani. Automatically Validating Temporal Safety Properties of Interfaces, *SPIN*, LNCS2057, 2001.

– T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. K. Rajamani, and A. Ustuner. Thorough static analysis of device drivers, *EuroSys*, 2006.

– T. Ball, E. Bounimova, R. Kumar, V. Levin. SLAM2: Static Driver Verification with Under 4% False Alarms, *FMCAD*, 2010.

SLAM

- SLAM2
- The Static Driver Verifier Research Platform
- http://research.microsoft.com/en-us/projects/slam/
- Related project
  - http://mtc.epfl.ch/software-tools/blast/index-epfl.php
    - Lazy abstraction