- سیستمی را در نظر بگیرید که یک ارسال کننده s و یک دریافت کننده R دارد. این دو جز از طریق دو کانال c, d با یکدیگر ارتباط دارند. ظرفیت هر دو کانال نیز نامتناهی است. هدف طراحی یک پروتکل ارتباطی است که مطمن شود پیام فرستاده شده از طرف s حتما به R رسیده است. پیام‌ها یکی یکی ارسال می‌شوند. ارسال کننده زمانی پیام جدید را ارسال می‌کند که مطمن باشد پیام قبلی تحویل داده شده است.

sender can have a timer and the timer can have a timeout which will prevent our sender to send all the time.

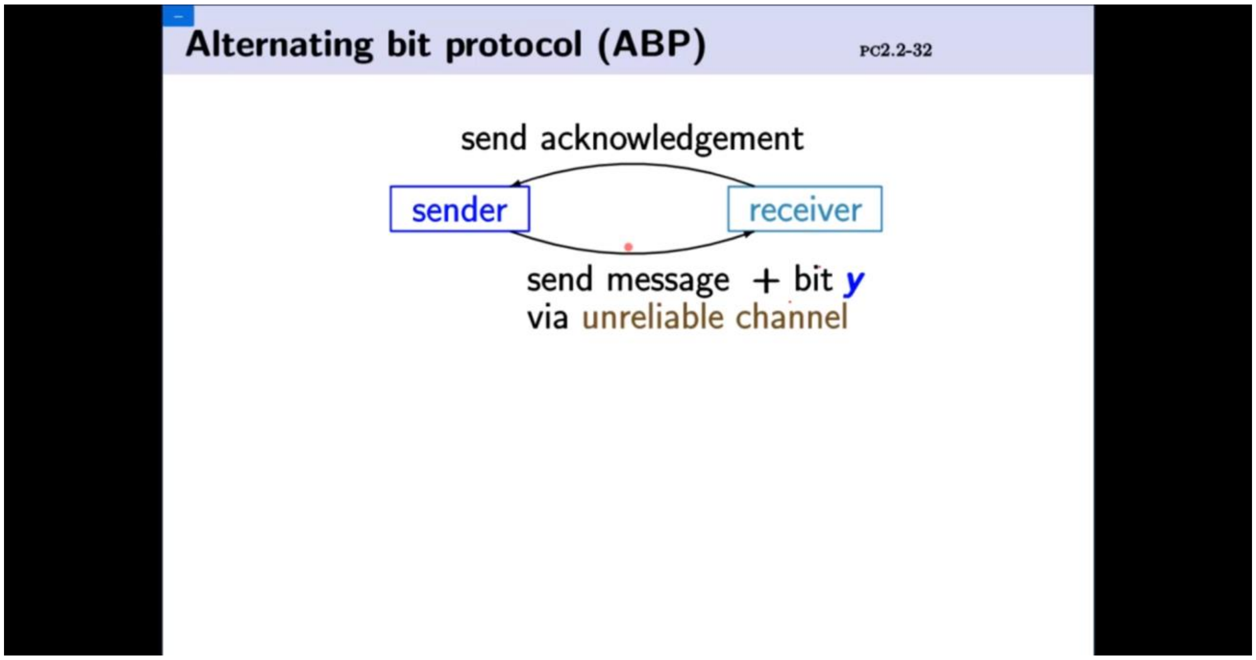
We have a y flag as a control flag which sender will ask our receiver if it can receive a message and to see if the channel is free

Sync , async , model checker , program graph , gcl?

Timer is defined in sender side

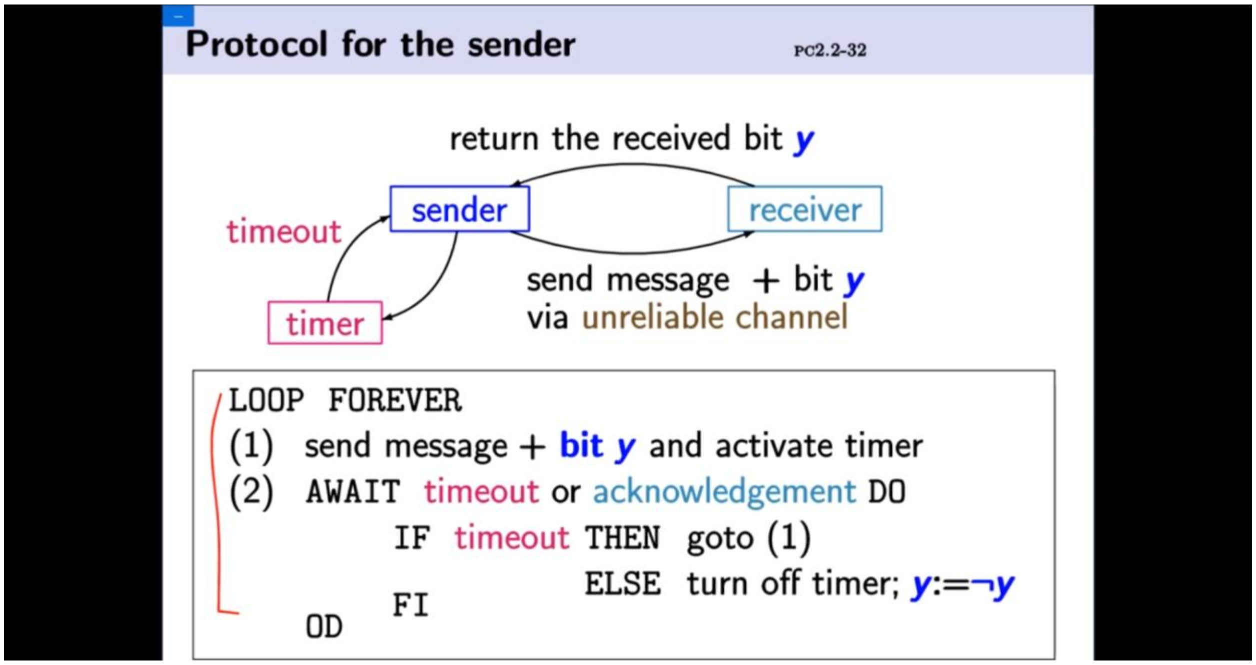
If our channel is dedicated we don’t need a control bit

When we send message + control bit receiver will acknowledge it to say that it has received the message.



Alternating bit protocol (ABP)

Gcl is :



Protocol for the sender

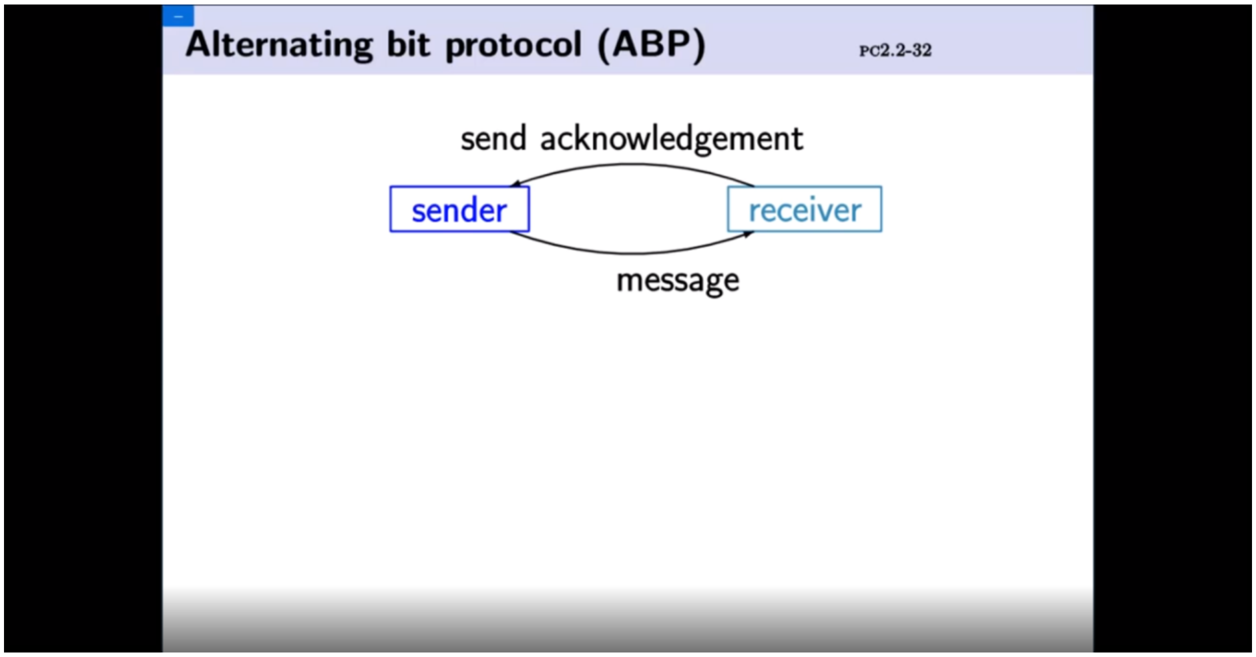
```
LOOP FOREVER
(1)  send message + bit y and activate timer
(2)  AWAIT timeout or acknowledgement DO
         IF timeout THEN goto (1)
                    ELSE turn off timer; $y := \neg y$
         FI
     OD
```

We have 2 types of transform

1. Sync : the moment that sender sends , receiver will answer
2. Async : the moment that sender sends, receiver will answer later this means that we have a wait in between

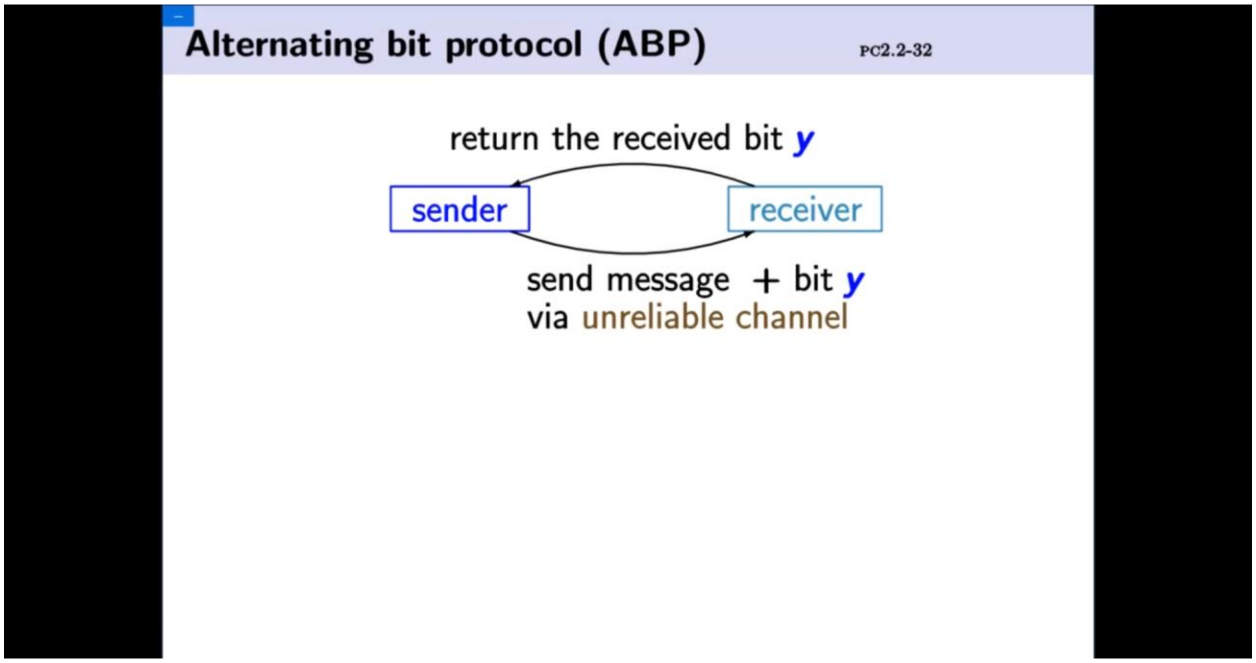
ABP : means we want to get and send data

We have a dedicated channel and when we send message receiver will send an acknowledgement back


Alternating bit protocol (ABP)

Message might be sent to receiver or may not so we use bit y and append it to message to realize if the receiver got the message.

We have a timer and if we haven't got the message in the specified timeout we will try again and send message again



Alternating bit protocol (ABP)

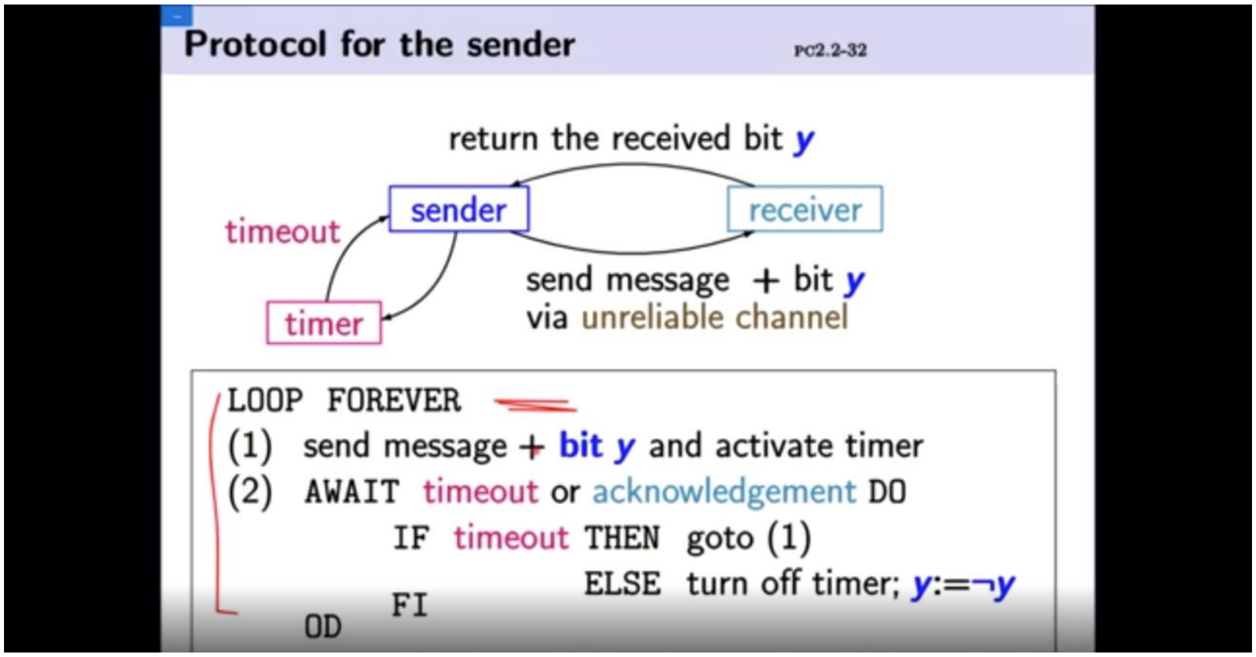
PC2.2-32

If we had timeout but the message was not sent or wasn’t sent correctly, send it again

And if the message was received and an ack will be sent to the sender to say that the message was received

DO will be closed with OD

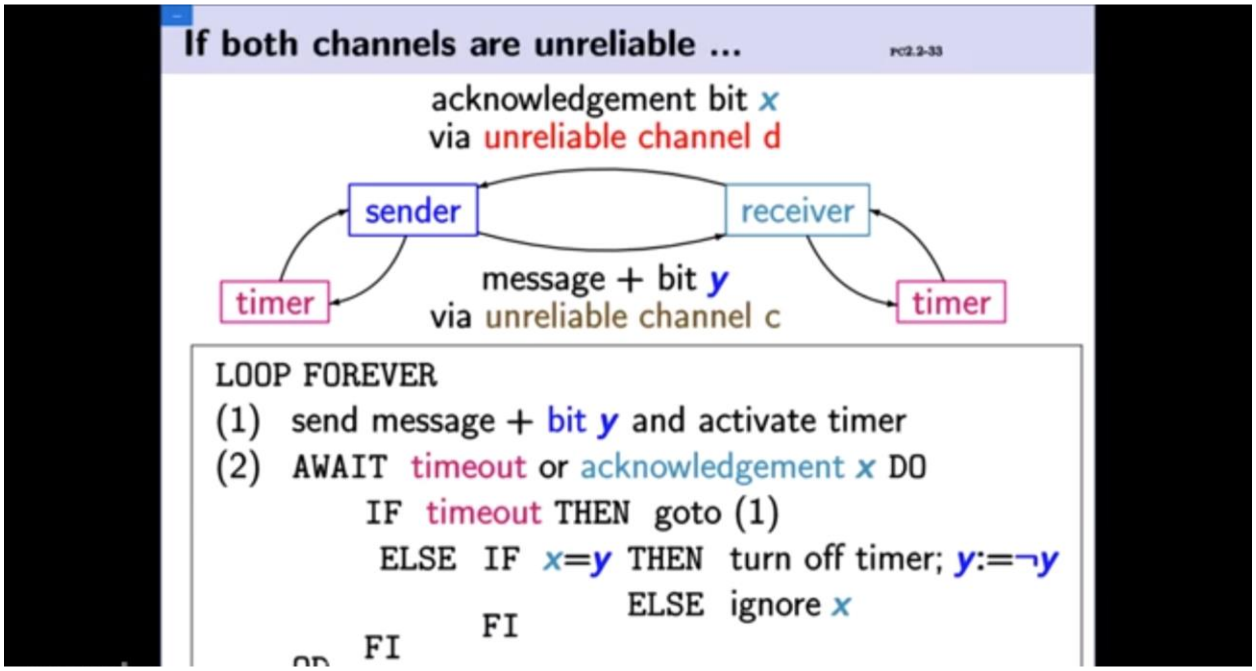
We have a protocol in sender side and another protocol in receiver side



Protocol for the sender

```
LOOP FOREVER
(1)  send message + bit y and activate timer
(2)  AWAIT timeout or acknowledgement DO
           IF timeout THEN goto (1)
                      ELSE turn off timer; y:=¬y
           FI
     OD
```

We have timer in both sides because both of our channels are unreliable so we use timer in both ends



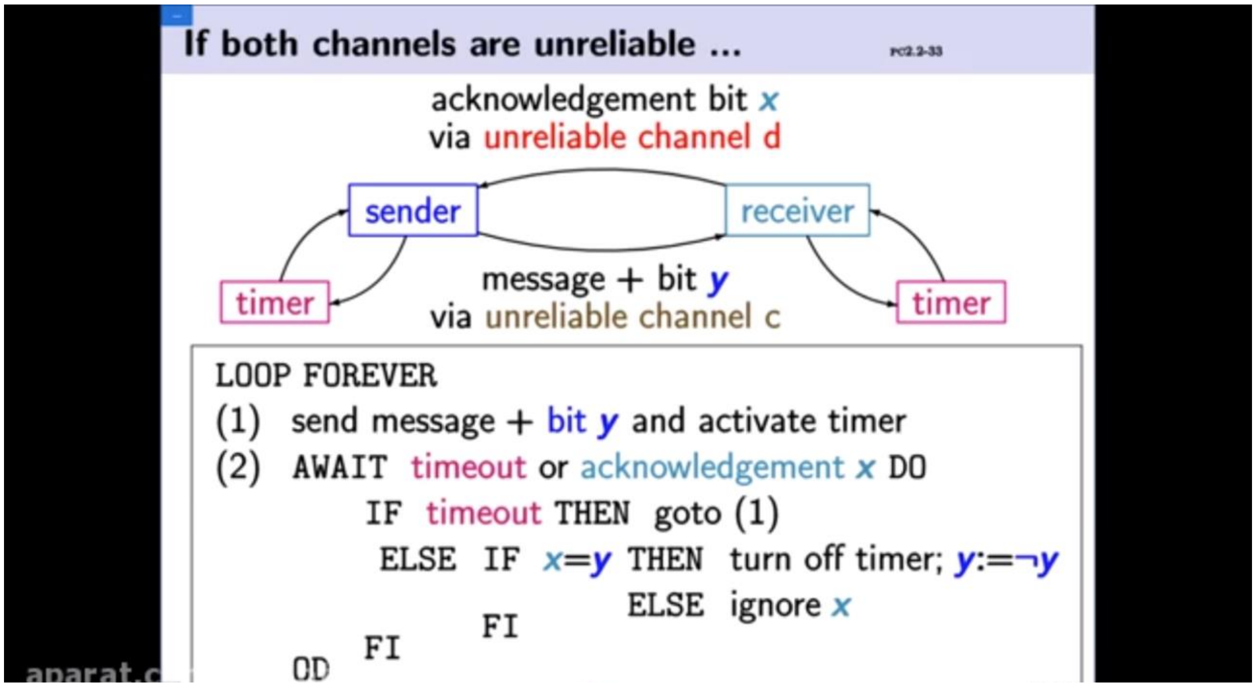
If both channels are unreliable ...

```
LOOP FOREVER
(1)  send message + bit y and activate timer
(2)  AWAIT timeout or acknowledgement x DO
           IF timeout THEN goto (1)
            ELSE IF x=y THEN turn off timer; y:=¬y
                           ELSE ignore x
                 FI
           FI
     OD
```

Acknowledgement for sender is bit y and acknowledgement for receiver is bit x

If x = y then we will turn the timer off means both sender and receiver got ack alright

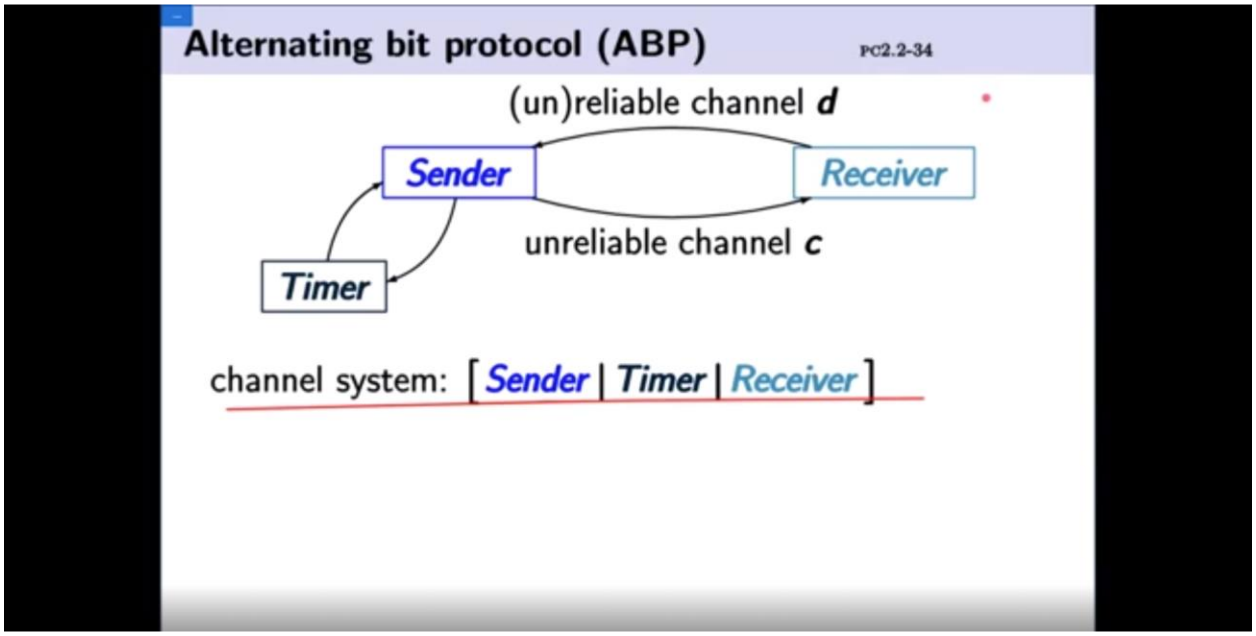
Below images is like above the only difference is that x = y then set the timer off



If both channels are unreliable ...

```
LOOP FOREVER
(1)  send message + bit y and activate timer
(2)  AWAIT timeout or acknowledgement x DO
         IF timeout THEN goto (1)
          ELSE IF x=y THEN turn off timer; y:=¬y
                      ELSE ignore x
               FI
         FI
     OD
```

System relies on receiver , timer , sender



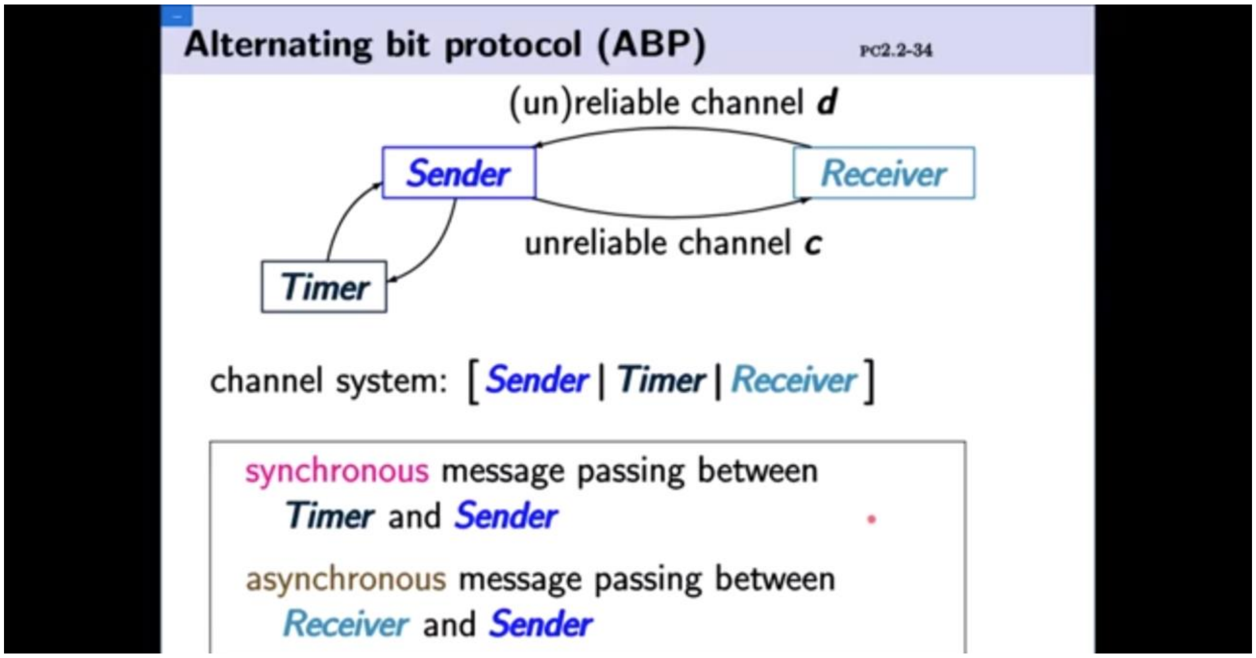
Alternating bit protocol (ABP)

PC2.2-34

channel system: [*Sender* | *Timer* | *Receiver*]

We have timer In sync so we don’t have conflict and we have it in TCP and is connection oriented

Why don’t we need timer in async : because we don’t care if receiver has gotten the message or not and we have it in UDP



Alternating bit protocol (ABP)

PC2.2-34

channel system: [*Sender* | *Timer* | *Receiver*]

synchronous message passing between *Timer* and *Sender*

asynchronous message passing between *Receiver* and *Sender*

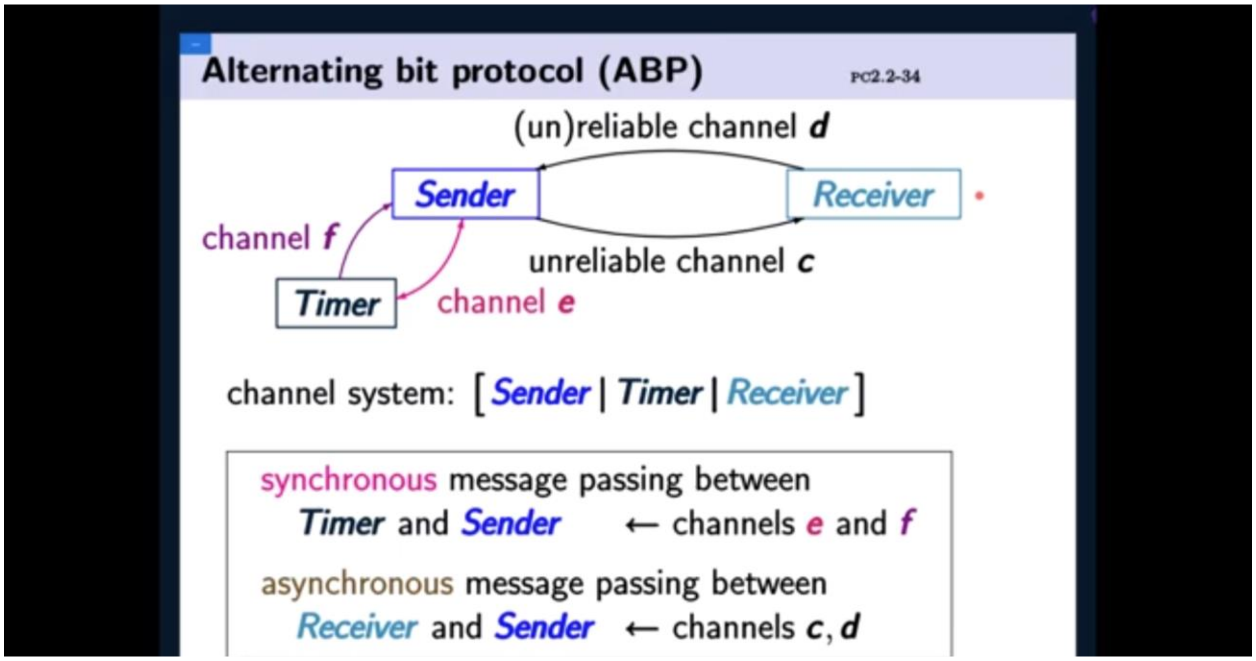
In async channels are not busy but in sync channels are busy

In timer we have location graph which can be on or off

Timer has to do 2 things:

Timer will get variable from sender to see it should go on with on or off mode

And should get ack so it would go off after



Alternating bit protocol (ABP)

channel system: $[\,Sender \mid Timer \mid Receiver\,]$

synchronous message passing between Timer and Sender ← channels e and f

asynchronous message passing between Receiver and Sender ← channels c, d

! = send

? = receive

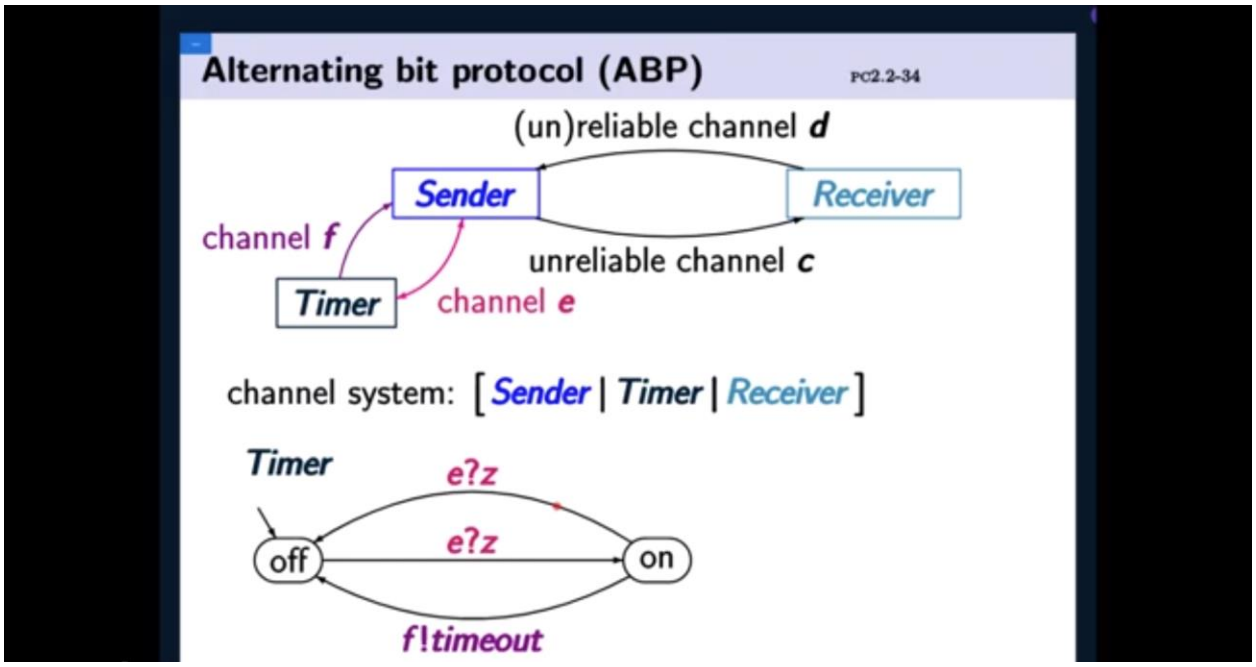
We have 3 actions below the most imp one is timeout

E?z means I will get z from e channel

E!z means I will put z in e channel

F!timeout = put timeout in f channel

Middle flash means e gets z and activate timeout

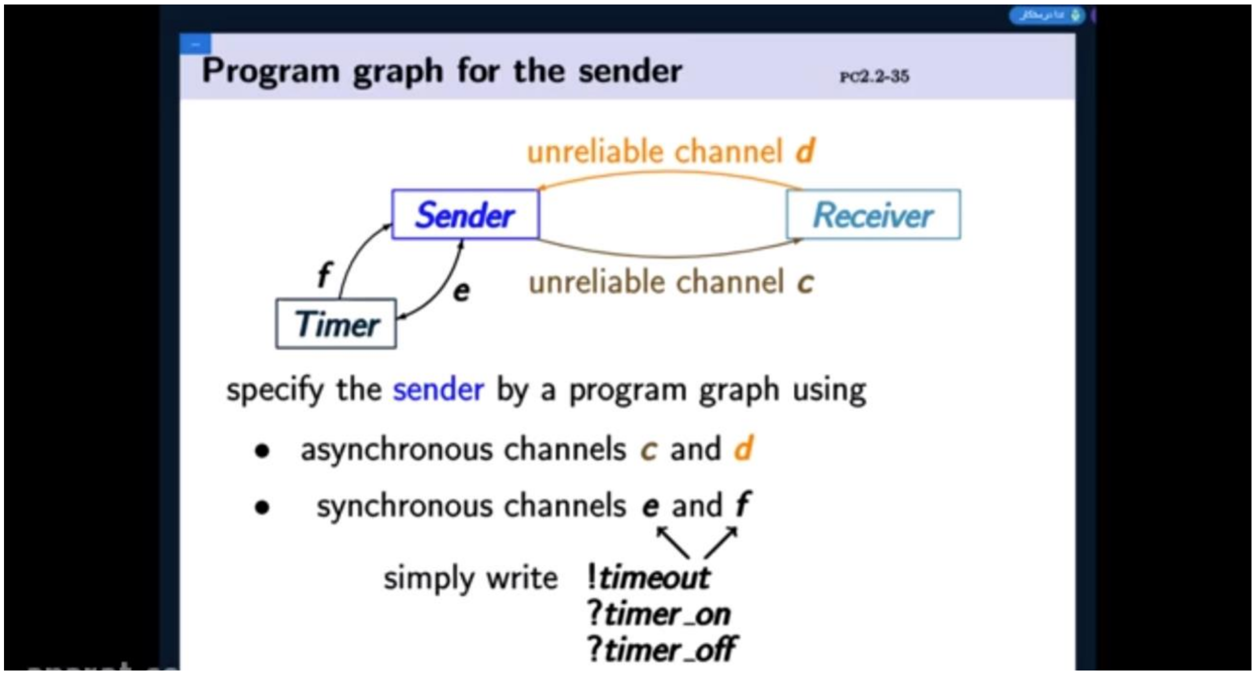


Alternating bit protocol (ABP)

PC2.2-34

channel system: [Sender | Timer | Receiver]

What is on and off ? means when we send message to receiver we set the timer on till we get timeout or ack and send again



Program graph for the sender

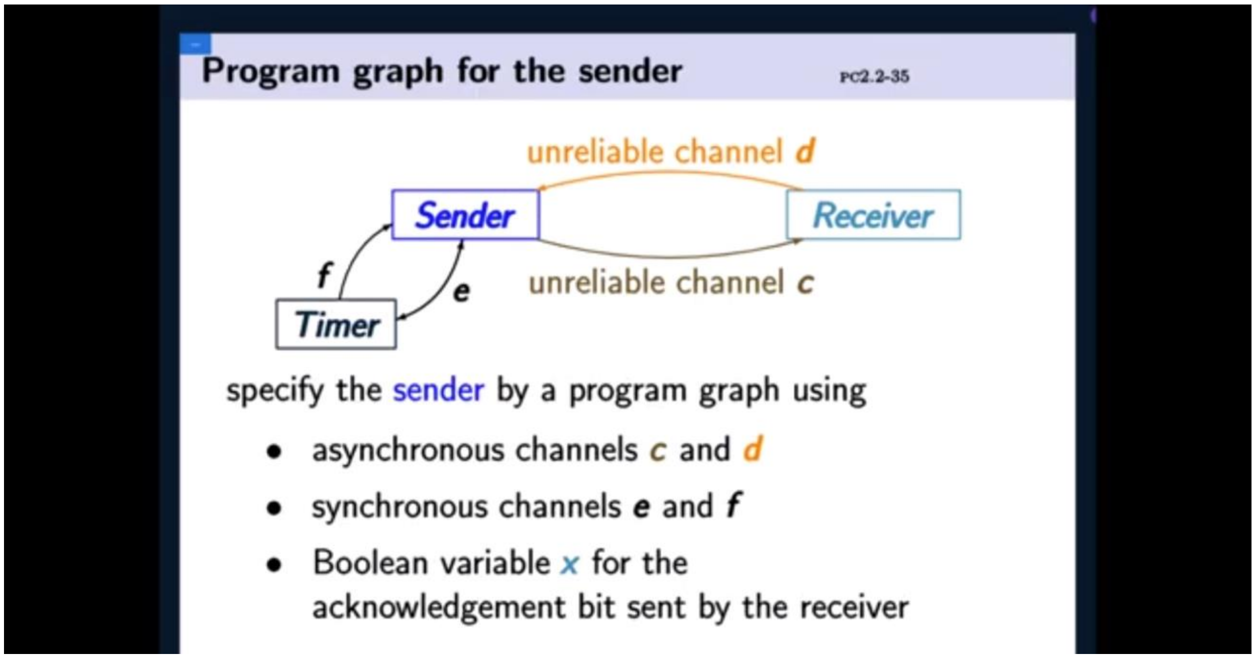
PC2.2-35

specify the sender by a program graph using

- asynchronous channels *c* and *d*
- synchronous channels *e* and *f*

simply write !*timeout*
?*timer_on*
?*timer_off*

When we get sth from channel = that sth will get deleted from channel



Program graph for the sender

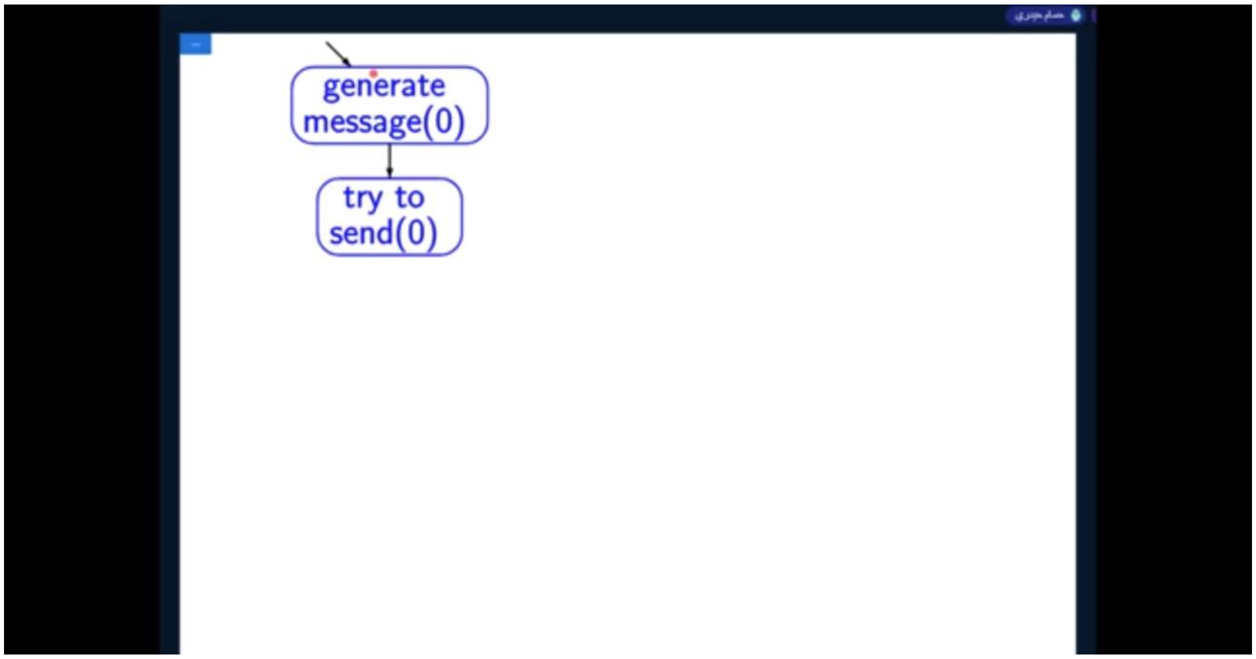
PC2.2-35

specify the sender by a program graph using

- asynchronous channels *c* and *d*
- synchronous channels *e* and *f*
- Boolean variable *x* for the acknowledgement bit sent by the receiver

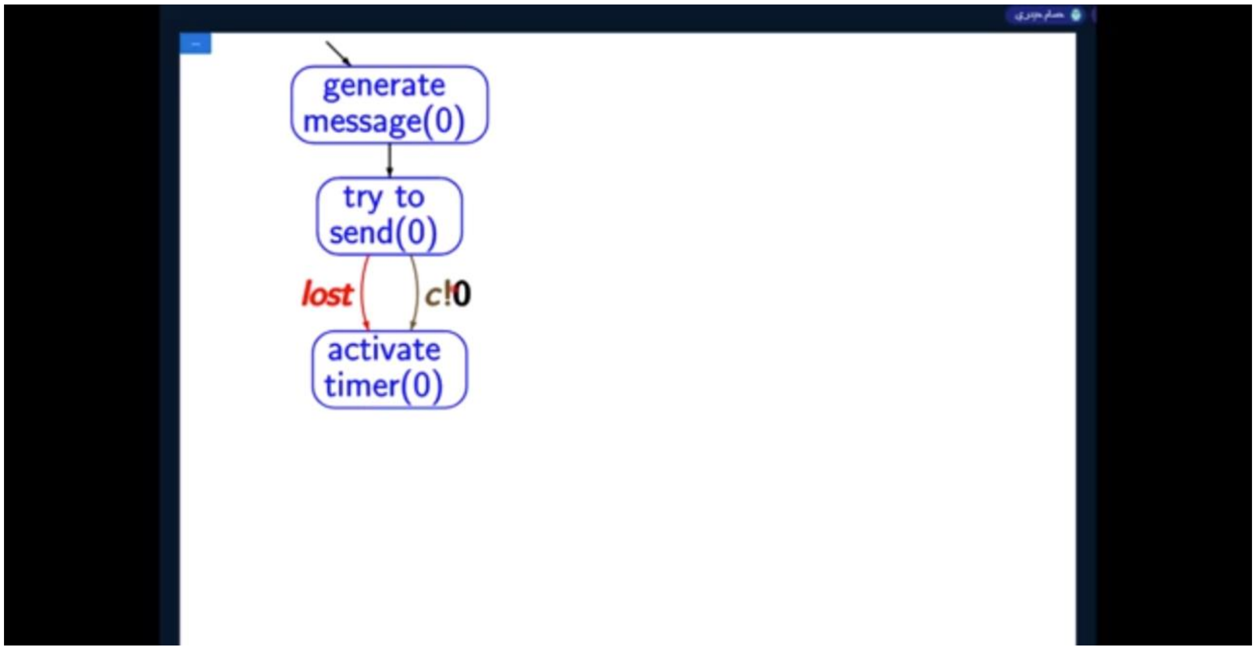
Program graph for sender

The first loc that we start on is generate message(0) which will generate the message then we should try to send it



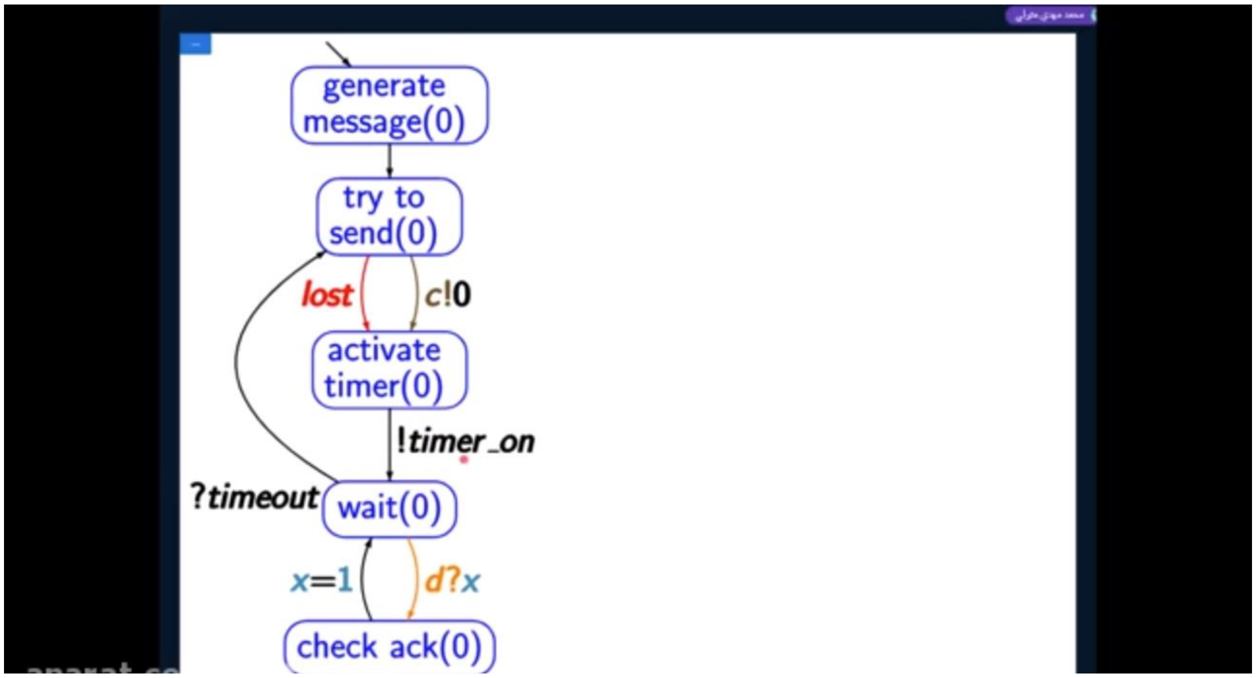
We have 2 actions either We send 0 to channel c or we lose it

In either ways the timer will be on

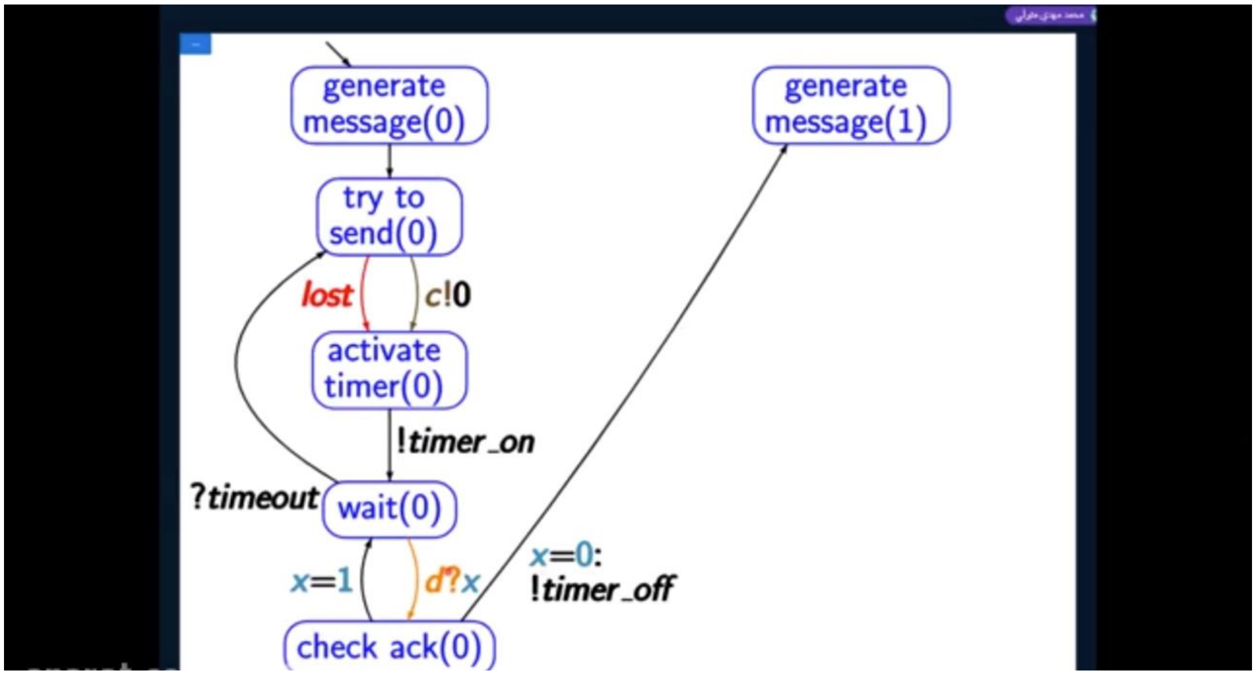


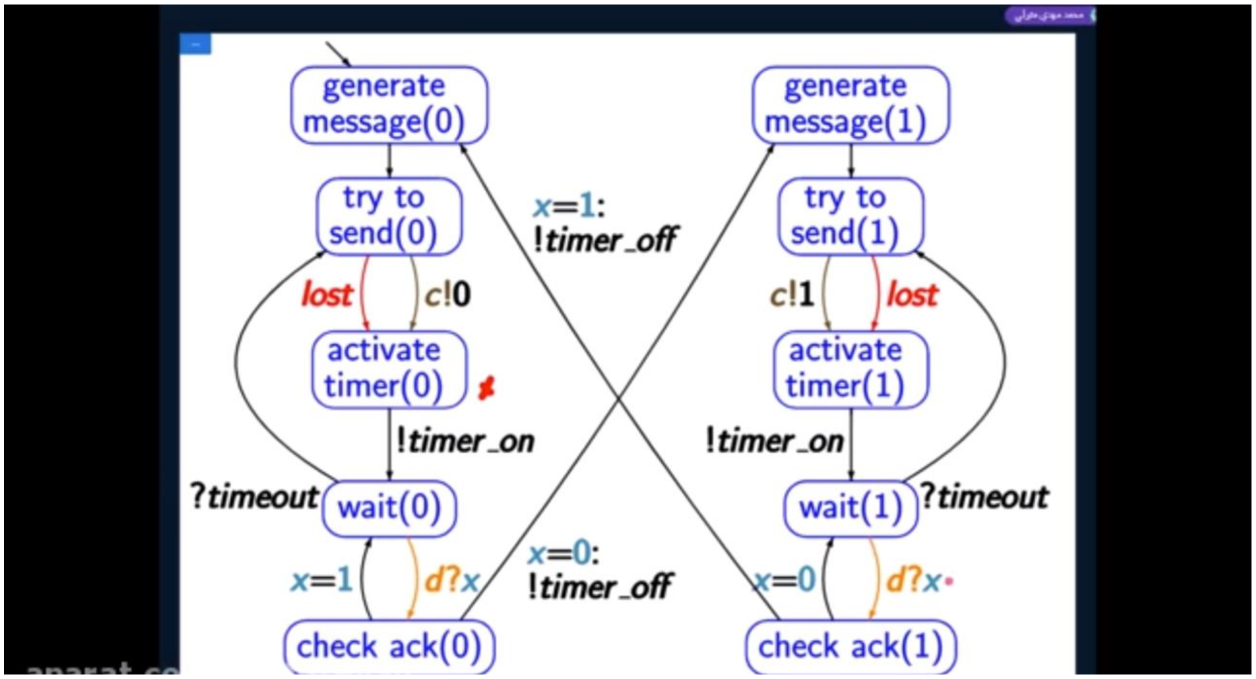
Channels are async between sender and receiver

In wait(0) we can have timeout then we ll go to try to send(0) stage



Then we check ack again and we have two modes again then we have the same graph again




generate message(0)
try to send(0)
lost
c!0
activate timer(0)
!timer_on
?timeout
wait(0)
x=1
d?x
check ack(0)
x=1: !timer_off
x=0: !timer_off
generate message(1)
try to send(1)
c!1
lost
activate timer(1)
!timer_on
wait(1)
?timeout
x=0
d?x
check ack(1)