

Marcel Fernandez

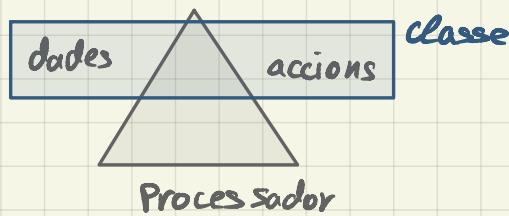
marcel @entel.upc.edu

Nota 25% lab (2 exams)

75% | 15% Parcial
| (màxim)
| 60% Final

Introducció al Java

"Ingredients Software"



Class PilaEnteres {

```
    Proteïda int[] espai;
    Proteïda int size; → num d'elements
    // Crear           (representació al espai)
    ATRIBUT
```

```
public void push (int valor) {
    espai [size] = valor;
    size = size + 1;
}
```

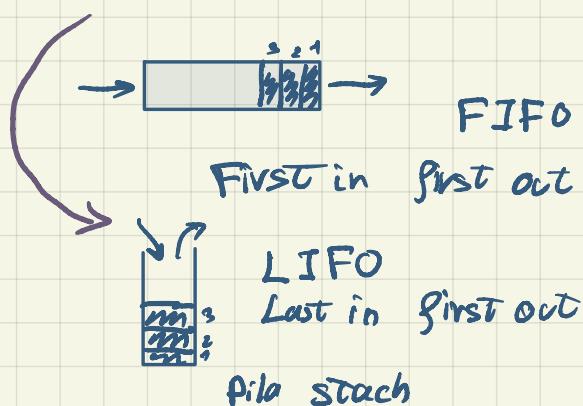
```
public int top() {
    return espai [size - 1]
```

→ retorna l'últim enter (representació d'una operació)
que s'ha posat per temps, rutina
MÈTODE

```
public void pop() {
    size = size - 1;
```

```
public boolean empty () {
    return size == 0
```

Exemple: Pila "infinita" d'enters



accions:

- push
- Pop ← treure
- top ← consultar
- empty

Especificació

Precondició:	Cu no buid
Post condició:	No



Exemple La classe de números complexes

```
public class complex {
    protected double _real, _imag
```

```
public double real () {
    return _real;
```

```
public double imag ()
    return _imag;
```

```
Public double mod () {
    return Math.sqrt (_real*_real + _imag*_imag)
    ↑ perque a la class Math, l'objecte es static
```

```
Public Complex suma (Complex altre) {
    Complex resultat = new Complex (); ← c=a.suma (b)
    resultat._real = this._real + altre._real; ← amb això ja no
    resultat._imag = this._imag + altre._imag; ← Compila però sort
    return resultat; null pointer exception
```

Tipus expandits i tipus referència

Arrays en Java

SÓN TIPUS REFERÈNCIA

```
int [] v; //Declaració
```

```
v=new int [4] //construcció
```

Exemple matríg de 5 files de complex
10 columnes

```
Complex [][] m;
```

```
m = new Complex [5] [10];
```

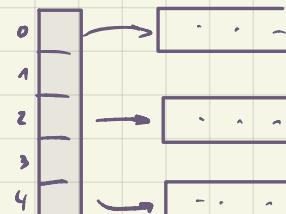
```
for (int i=0; i<5; i++) {
```

```
    m[i] = new Complex [10];
```

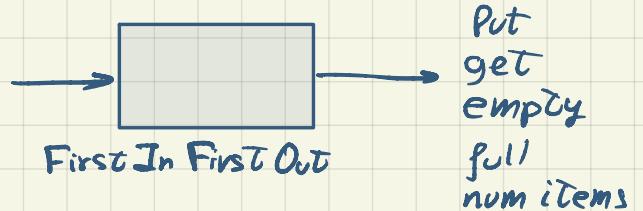
```
    for (int j=0; j<10; j++)
```

```
        m[i][j] = new Complex();
```

$m \square \rightarrow$



- Crear una classe
a partir d'una altra
- Afegir mètodes i/o atributs
 - Definir mètodes
 - Implementar classes abstractes



Herència

```
public abstract class Queue {
    // Precondició: cua no plena
    // Postcondició: cua no buida
    public abstract void put (Object e);
    // Precondició: cua no buida
    // Postcondició: cua no plena

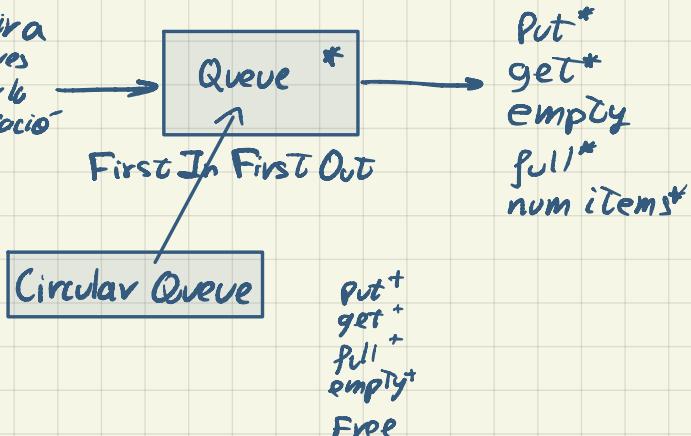
    public abstract Object get();

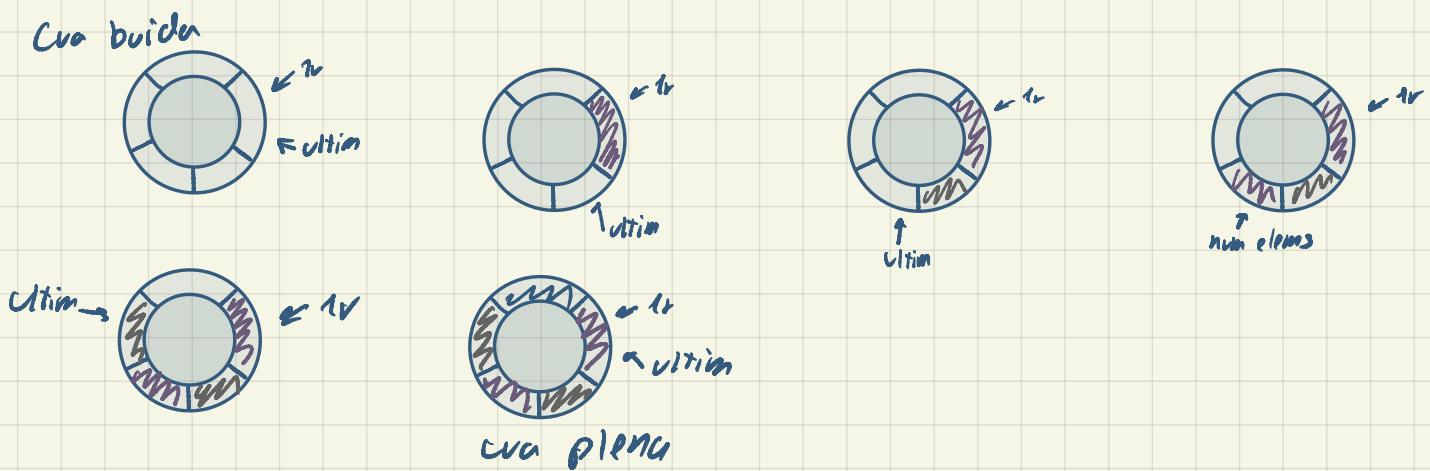
    public abstract int numElems();

    public abstract boolean full();

    public boolean empty();
}
```

Les classes abstractes
Permeten accedir a
dades per les seves
propietats i no per la
seva implementació



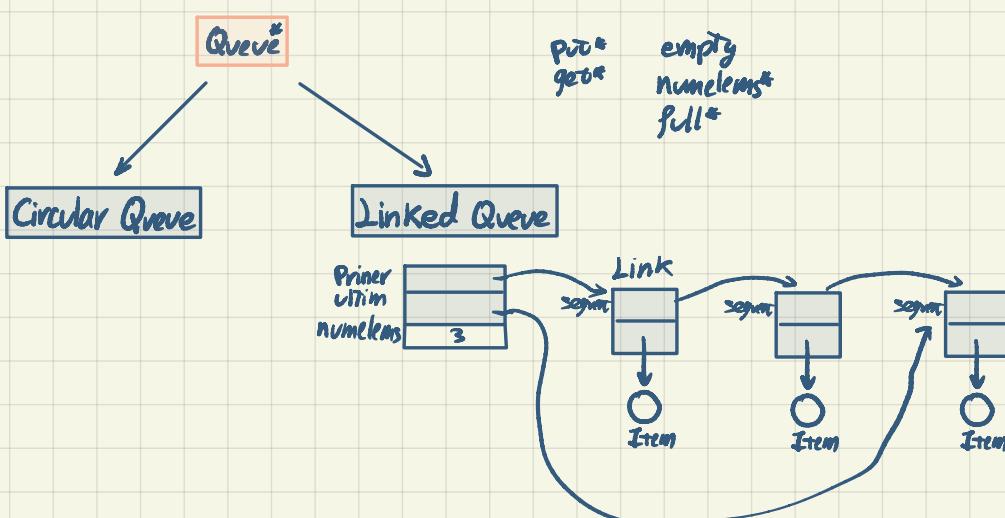


```

public class CircularQueue extends Queue;
protected Object [] espacio;
protected int capacidad, primer, ultimo, _numelems;

public void put (Object e)
    espacio [ultimo] = e;
    ultimo = (ultimo + 1) % capacidad;
    _numelems = _numelems++;

public Object get ()
    Object res = espacio [primer];
    primer = (primer + 1) % capacidad;
    _numelems = _numelems - 1;
  
```



```

class Link {
    Link segment;
    Object item
}

Public Link (Link s, Object i)
{
    Segment = s;
    Item = i;
}
  
```

```

Public void enlazar (Link e)
{
    segment = e;
}
  
```

protected class linkedQueue extends Queue {

protected Link primer, ultim;
protected int _numElems;

Public int numElems () {
return _numElems;
}

Public boolean full () {
return false;

Public void put (Object e) {
Link tmp = new Link (null, e);
if (primer == null)
primer = tmp;
else {
ultim.enllaçar (tmp);
}
ultim = tmp;
_numElems = _numElems + 1;
}

Public Object get () {
Object resultat = primer.item;
Primer = primer.seguent;
if (primer == null)
ultim = null;
_numElems = _numElems - 1;
return resultat;

Circular

cursor de ir a ultim
↑ int

linked Queue

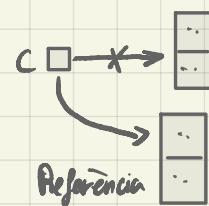
has.next → Seguent no null
un link fa de cursor

Tipus expandits i Tipus referència

int i, j int < short
 long
 boolean
 char } expandits

Complex C;
C=new Complex();

classe
anys



En tipus referència el cicle de vida dels objectes es independent del cicle de les variables

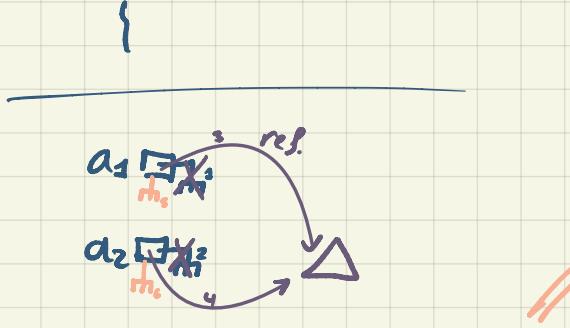
cicle vida var
...;
int i;
{

cicle vida objecte
inici amb un new
fi ...

```

1   A a1;
2   A a2;
3   a1 = new A();
4   a2 = a1;
5   a1 = null;
6   a2 = null;

```



Exercici: Fer un programa que simuli una cursa de 100 m lliures amb 2 corredors

```
public class cursa {
```

```
    public static void main (String[] args[])
        Corredor C1 = new Corredor("C1")
        Corredor C2 = new Corredor ("C2")
```

```
        C2.correr();
        C1.correr();
    }
```

```
public class corredor {}
```

```
    protected String nom;
```

```
    public Corredor (String nm) {
```

```
        nom=nm;
```

```
    public void correr () {
```

```
        for (int i=0; i<=100; i++) {
```

```
            sleep(Math.random());
```

```
            System.out.println (nom + " -> " + i + "m")
```

```
}
```

```
{
```

FL D'EXECUCIÓ THREAD

public class cursa {

```
    public static void main (String[] args[])
        Corredor C1 = new Corredor ("C1")
        Corredor C2 = new Corredor ("C2")
```

```
C2.run(); start();
C1.run(); start();
```

← Arranca el thread
i executa el run

C2.join
C1.join
print ("cursa acabada")

} }

public class corredor extends Thread {}

protected String nom;

public Corredor (String nm) {

{ nom=nm;

private void correr () {

for (int i=0; i<=100; i++) {

sleep (Math.random());

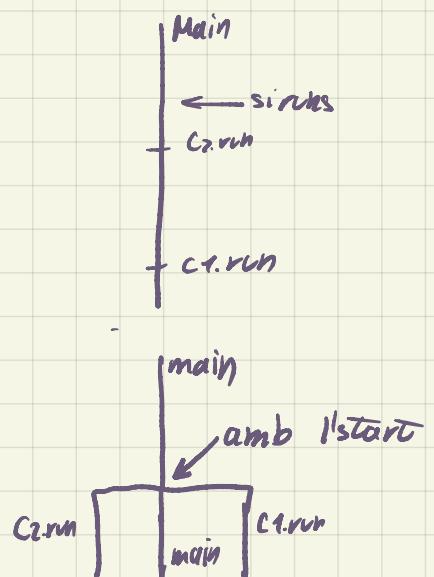
System.out.println (nom + " -> " + i + "m")

Obligatori
(el que fa el thread) { }

→ public void run() {

correr();

}



Mètodes del thread

run()

start()

sleep (temps)

join() ← pq statut

Static void →

Sleep (temps)

join() ← pq statut

Si ara volguessim que corredor heredes d'una altra classe persona per exemple,
no es podria fer, java no permet herencia multiple. Podriem utilitzar:

public interface Runnable {

public void run();

{

s'ha d'implementar
obligatoriament

(Quan era de thread no feia falta)

```
public class cursa {
```

```
    public static void main (String[] args[])
```

```
        Corredor C1 = new Corredor ("C1")
```

```
        Corredor C2 = new Corredor ("C2")
```

```
        Thread T1 = new Thread (C1)
```

```
        Thread T2 = new Thread (C2)
```

```
        T1.start ()
```

```
        T2.start ()
```

```
        C1.join ()
```

```
        T2.join ()
```

```
        System.out.println ("cursa acabada")
```

```
{ }
```

```
public class corredor extends Persona implements Runnable {
```

```
    public Corredor (String nm) {
```

```
        super (nm);
```

```
}
```

```
    public void correr () {
```

```
        for (int i=0; i<=100; i++) {
```

```
            Thread.sleep (Math.random());
```

```
            System.out.println (nm + " -> " + i + "m")
```

```
}
```

```
{ }
```

Thread A

a₁

a₂

a₃

Thread B

b₁

b₂

b₃

Traces d'execució

a₁

a₂

a₃

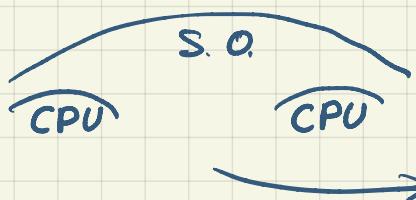
b₁

a₁

a₂

Paralelisme

NO HI HA DEBUGGING



Si només tenguessim un CPU, tendriem Concurrència

Thread A

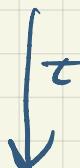
$$v = v + 1;$$

$$\begin{aligned} a_1, & a_x = v; \\ a_2, & \text{Inc } a_x; \\ a_3, & v = a_x; \end{aligned}$$

Thread B

$$v = v + 1$$

$$\begin{aligned} b_1, & b_x = v; \\ b_2, & \text{Inc } b_x; \\ b_3, & v = b_x; \end{aligned}$$



$v?$ → Problema pot valer 1 o més segons que s'executi primer.

traça execució → $b_1 \rightarrow b_2 \rightarrow a_1 \rightarrow b_3 \rightarrow a_2 \rightarrow a_3 \rightarrow v = 1$

Exercici: Una aplicació té 2 threads, un que observa esdeveniments i un altre que informa sobre el número d'esdeveniments que han passat

El codi no és
Correcte sintàcticament

Thread Observador {

```
→ int compt=0;
While (true){
    // esper esdeveniment
    → compt=compt+1;
}
```

Zona Crítica COMPARTIDA

Thread Informador {

```
→ int compt=0;
While (true){
    System.out.println("han passat "+compt+" esdeveniments")
    → compt=0;
}
```

Estructura general d'un Thread accedint a zona crítica

Thread[i] { // 1 < i < n

While (true){

Protocol entrada
a zona crítica

zona crítica

Protocol de sortida

Primitives
d'exclusió
mútua

1) La zona crítica s'ha d'executar en EXCLUSIÓ MÚTUA

(només un thread pot accedir a c. alhora)

2) Si un thread no és dins z.c. no pot impedir que un altre hi entri

3) Si un thread vol accedir a z.c. eventualment ho aconseguirà (JUSTÍCIA)

• Primera "solució" al problema d'EM.

boolean ocupat;
ocupat=false;

Thread 1

```
while (true)
    while (ocupat == true){}
    ocupat = true
ZONA CRÍTICA
    ocupat = false
```

Bucle espera activa

Thread 2

```
while (true)
    while (ocupat == true){}
    ocupat = true
ZONA CRÍTICA
    ocupat = false
```

OBRIM PARENTESI

Tenim un problema similar pq ocupat també és una var. compartida

• Segona "solució" al problema d'EM

boolean dins1, dins2;
dins1 = false;
dins2 = false;

Thread 1

```
while (true)
    dins1 = true;
    while (dins2 == true){}
    ZONA CRÍTICA
    dins1 = false;
```

Thread 2

```
while (true)
    dins2 = true;
    while (dins1 == true){}
    ZONA CRÍTICA
    dins2 = false;
```

Un thread modifica la seva var.
Però només pot consultar la var. de l'altre Thread

Si els dos es posen a True alhora entren als bucles i no en surten ENLLAVAMENT

• Tercera "solució" al problema d'EM

boolean dins1, dins2;
int ultim; //ultim thread intenta entrar
dins1 = false;
dins2 = false;

Thread 1

```
while (true)
    dins1 = true;
    entrarzc
        ultim = 1;
        while (dins2 == true && ultim == 1){}
        ZONA CRÍTICA
    salirzc
        dins1 = false;
```

Thread 2

```
while (true)
    dins2 = true;
    ultim = 2;
    while (dins1 == true && ultim == 2){}
    ZONA CRÍTICA
    dins2 = false;
```

Exercici: Programar classe amb dos mètodes incl(), decl() que incrementen

1. i disminueixen el comptador en una unitat

```
public class Comptador {  
    protected int valor;  
  
    public Comptador (int inicial)  
        valor = inicial;  
  
    public void incl()  
        compt = compt + 1;  
  
    public void decl()  
        compt = compt - 1;  
}
```

2. Modificar el comptador perque pugui ser utilitzat
per varis threads alhora

```
public class Comptador {  
    protected Mutex em;  
    protected int valor;  
  
    public Comptador (int inicial)  
        valor = inicial;  
  
    public void incl()  
        entrar ZCC();  
        valor = valor + 1;  
        emsortir ZCC();  
    }  
  
    public void decl()  
        entrar ZCC();  
        valor = valor - 1;  
        emsortir ZCC();  
    }  
}
```

3. Modificar el comptador perque sempre sigui positiu
Si val 0, i un thread crida decl() s'espera fins que thread
crida incl() i llavors decrementa

```

public class Comptador {
    Protected Mutex em;
    protected int valor;

    Public Comptador (int inicial)
        valor = inicial;

    public void inc(){
        ementrar ZCC();
        valor = valor + 1;
        emsortir ZCC();
    }
}

```

```

public void dec(){
    ementrar ZCC();
    while (valor == 0){
        emsortir ZCC();
        ementrar ZCC();
    }
    valor = valor - 1;
    emsortir ZCC();
}

```

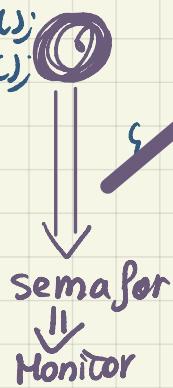
~~public void dec()~~

```

while (valor == 0){
    ementrar ZCC();
    valor = valor - 1;
    emsortir ZCC();
}

```

No pq entra més d'un fil al while i després pot ser que es deurem.



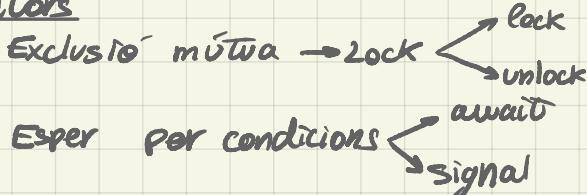
TANCA PARENTESI

Monitor: Construcció que encapsula una estructura de dades on els mètodes s'executen en exclusió mútua

Per l'espera per condicions es defineixen VARIABLES DE CONDICIO (sales d'espera)

- await (): Atura el Thread que l'invoca de manera incondicional i deixa el monitor lliure
- Signal (): Si hi ha threads aturats en desperta un Si no, no fa res.

Monitors



```

public class ComptadorPositiv {
    public int valor;
    protected Lock mon;
    protected Condition positiv;

```

```

    public ComptadorPositiv() {
        mon = new ReentrantLock();
        positiv = mon.newCondition();
    }
}

```

```

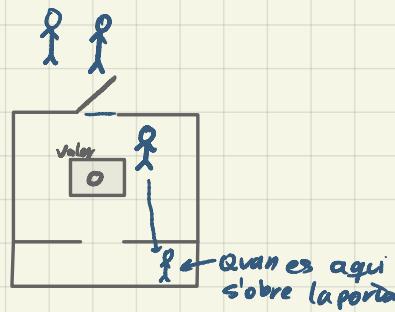
    public void inc() {
        mon.lock();
        valor = valor + 1;
        positiv.signal();
        mon.unlock();
    }
}

```

```

    public void dec() {
        mon.lock();
        while(valor == 0)
            positiv.await();
        valor = valor - 1;
        mon.unlock();
    }
}

```



SIGNAL es un anís, no una ordre

Problema Productor-Consumidor

```

public class Buffer {
    protected Queue q;
    protected Lock mon;
    protected Condition noPlena, noBuida;
}

```

```

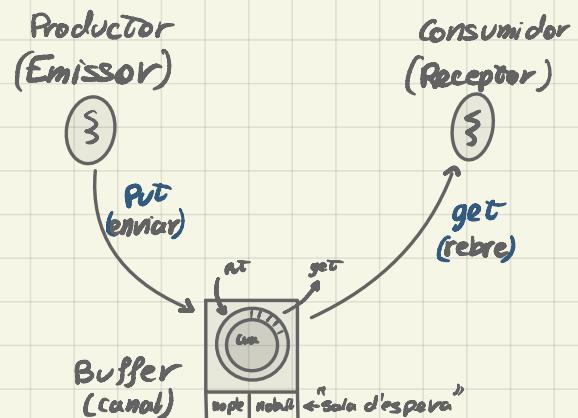
    public void put(Object e) {
        mon.lock();
        while(q.full())
            noPlena.wait();
        // Aserció: cua no plena
        q.put(e);
        // Aserció: cua no buida
        noBuida.signal();
        mon.unlock();
    }
}

```

```

    public Object get() {
        mon.lock();
        while(q.empty())
            noBuida.wait();
        // Aserció: cua no buida
        Object res = q.get();
        // Aserció: cua no plena
        noPlena.signal();
        mon.unlock();
        return res;
    }
}

```



```

try {
    mon.lock();
    ;
    return;
} finally {
    mon.unlock();
}

```

Public class Productor implements Runnable {

Protected Buffer b;

Public Productor (buffer b) {

b = buff;

}

Public void run () {

For (int i = 0; i < q; i++)

b.put

}

public class Execucio {

public static void main (String [] args) {

Buffer b = new Buffer

Productor p1 = new Productor (b);

Productor p2 = new Productor (b);

Exercici

BarreraTres. Implementa un monitor BarreraTres amb un únic mètode atura().
Hi ha 3 threads que creiden atura() i esperen, el tercer desperta als altres dos i continuen tots 3 (i així de manera ciclica).

Els 2 primers threads que creiden atura() s'esperen, el tercer desperta als altres dos i continuen tots 3 (i així de manera ciclica).
Hi ha 3 threads

public class BarreraTres {

Private int cont;

Private lock mon;

Private Condition tres;

mil;

Public void atura () {

mon.lock();

cont = cont + 1;

If (cont < 3)

Tres.await();

else

Tres.signal();

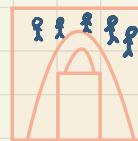
Tres.signal(); mil.SignalAll();

Cont = 0;

} mon.unlock();

Exercici 3: fer la barrera amb while

Exercici 1:



Els jugadors volen accedir a la línia de tir lliures on només pot haver com a molt 3. Un cop un jugador és a la línia pot tirar si ningú està tirant.

Per cada cistella:

Individual

Jugador 3: 3 punts

Punts Total 16

comú

Cada jugador acaba quan ha fet 5 cistelles

```
public class Entrenamiento {  
    public void accederLinia() {...}  
    public void iniciar() {...}  
    public void tirar() {...}  
    public void sortirLinia() {...}
```

Public class entrenamiento implements thread

```
private int jugadoresdins, puntosTotal, numJugadores;  
private boolean ningutira;  
private ReentrantLock mon;  
private Condition vultirar, vullentrar, cincPuntos;  
private int[] puntos;
```

```
public void accederLinia(){  
    mon.lock;  
    try{  
        while(jugadoresdins >= 3)  
            vullentrar.await();  
        jugadoresdins++;  
        numJugadores++;  
    } finally  
    {  
        mon.unlock  
    }
```

```
public void iniciar(){  
    mon.lock;  
    try{  
        while(!ningutira)  
            vultirar.await();  
        ningutira = false;  
    } finally  
    {  
        mon.unlock  
    }
```

```
public void fitir() {  
    mon.lock;  
    try {
```

```
        finally  
        mon.unlock
```

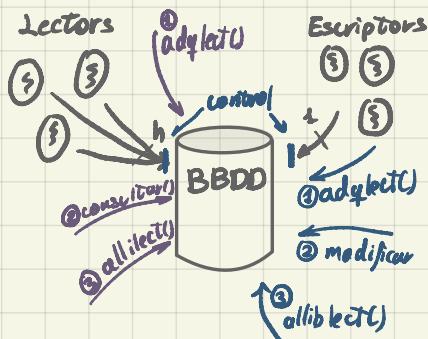
```
}
```

Problema lectors i escriptors

- Threads lectors i escriptors comparteixen una base de dades

- Hi pot haver vari's lectors alhora

- Per altra banda, cada escriptor té accés exclusiu.



```

public ControlLecEsc {
    protected lock mon;
    protected Condicion pvlEsc, pvllec;
    protected int numlec, lectesp, escEsp;
    protected boolean escript, despertantLectors;
}
    
```

```

public class lector implements Runnable {
    protected ControlLecEsc control;
}
    
```

constructor()

```

public void run() {
    while (true) {
        control.adglect();
        // llegir
        control.alliLect();
        // fer algo.
    }
}
    
```

```

public void adgEsc() {
    mon.lock();
    while (numLec > 0 || escript || despertantLectors) {
        escEsp = escEsp + 1;
        pvlEsc . wait();
        escEsp = escEsp - 1;
        escript = true;
        mon.unlock();
    }
}
    
```

```

public void adglect() {
    mon.lock();
    while (escript || escEsp > 0) {
        lectesp = lectesp + 1;
        pvllec . wait();
        lectesp = lectesp - 1;
        numLec = numLec + 1;
        mon.unlock();
    }
}
    
```

```

public void aliEsc() {
    mon.lock();
    escript = false;
    if (lectesp > 0) {
        pvllec . signal();
        despertantLectors = true;
    } else if (escEsp > 0) {
        pvlEsc . signal();
    }
    mon.unlock();
}
    
```

```

public void alliLec() {
    mon.lock();
    numLec = numLec - 1;
    if (numLec == 0) {
        pvlEsc . signal();
    }
    mon.unlock();
}
    
```

*while((escrit||escEsp>0)&!despertantLectors)
*, if(lectesp==0)
despertantLectors=false;

És injust per escriptors
És injust per lectors

Intercanviador

Els vol implementar un monitor amb un únic mètode

Object intercanvi (Object e)

Els primer thread que crida intercanvi s'espera a que arribi un segon thread i s'intercanvien l'objecte que passa per paràmetre

a) Nomes dos threads

Class intercanviador

```
private lock mon;
private condition sondos;
private int compt;
private Object aux;
```

Object Intercanvi (Object e) {

mon.lock

try {

Object res;

1 2

compt = compt + 1;

if (compt == 1)

aux = e

sondos.await;

res = aux;

compt --;

{ else }

res = aux;

aux = e;

Segon arribat. signal;

compt --;

return res;

{ finally { mon.unlock(); }}

b) N>>2

Class intercanviador

```
private lock mon;
private condition sondos, intercanviArabat;
private int compt = 0; ← node fils intercanviant
private Object aux;
```

Object Intercanvi (Object e) ↴

mon. lock

try ↴

```
Object res;
while (compt==2)
    intercanviAcabat.await;

Compt=Compt+1;
if (compt==1)
    aux=e
    sondos.await;
    res=aux;
    Compt=0;
    intercanviAcabat.SignalAll;
{
else}
    res=tmp;
    tmp=e;
    Segon arribat. signal;
    Compt=2;
}
return res;
{ finally ↴ mon.unlock} ↴
```

↓ entendre pq esto malament
class intercanviador

```
private lock mon;
private Condition sondos, intercanviAcabat;
private int Compt=0; ← node fils intercanviant
private Object aux, res1, res2;
```

Object Intercanvi (Object e) ↴

mon. lock

try ↴

```
while (compt==2)
    intercanviAcabat.await;

Compt=Compt+1;
if (compt==1)
    aux=e
    sondos.await;
    res1=aux;
    Compt=0;
    intercanviAcabat.SignalAll;
    return res1;
}
else {
    res2=tmp;
    tmp=e;
    Segon arribat. signal;
    return res2;
}
{ finally ↴ mon.unlock} ↴
```

```
public class Estalvis {
```

```
protected lock mon;
protected Condition Suficient;
protected int balance;
```

```
public Estalvis()
```

```
balance = initial;
mon = new ReentrantLock();
Suficient = new mon.Condition();
```

```
{
```

```
public void depositar(int quantitat) {
```

```
mon.lock();
balance = balance + quantitat;
Suficient.signalAll();
mon.unlock()
```

```
{
```

```
public void extreure(int quantitat) {
```

```
mon.lock();
while (balance < quantitat)
    Suficient.await();
```

```
balance = balance - quantitat
mon.unlock();
```

```
{
```

```
public class Estalvis {
```

```
    protected lock mon;
    protected LinkedQueue<Conditions> PoderTreure;
    protected int balance;
    protected LinkedQueue<Integer> aextreure;
```

```
    public Estalvis()
```

```
        balance = inicial;
        mon = new ReentrantLock();
        sufficient = new mon.Condition();
```

```
{
```

```
    public void depositar(int quantitat) {
```

```
        mon.lock();
```

```
        balance = balance + quantitat
```

```
        if (!PoderTreure.isEmpty() && balance >= aextreure.peekFirst())
            PoderTreure.peekFirst().Signal();
```

```
        mon.unlock()
    }
```

```
    public void extreure(int quantitat) {
```

```
        mon.lock();
```

```
        if (balance < quantitat || !PoderTreure.empty()) {
```

```
            Condition tmp = mon.newCondition();
```

```
            PoderTreure.put(tmp);
```

```
            aExtreure.put(quantitat);
```

```
            tmp.await();
```

```
            balance = balance - quantitat;
```

```
            aExtreure.get();
```

```
            PoderTreure.get();
```

```
        } else
```

```
            balance = balance - quantitat;
```

Despertar en →
cadenas

```
        if (!PoderTreure.isEmpty() && balance >= aextreure.peekFirst())
            PoderTreure.peekFirst().Signal();
```

```
}
```

One Lane Bridge → Pont de Sentit Únic

Public class Pont

- a) Ferho
- b) Solució justa

Public void entrar (boolean sentit) {

}

Public void sortir () {

}

Public class cotxe implements Runnable {

Protected boolean sentitMev;

Protected Pont pont;

Public class cotxe(Pont p, boolean sentit)

pont=p;
sentitMev=sentit;

{

Public void run () {

for (~ ~) {

Pont. entrar (sentitMev);

Thread.sleep (m);

Pont. sortir ();

}

4

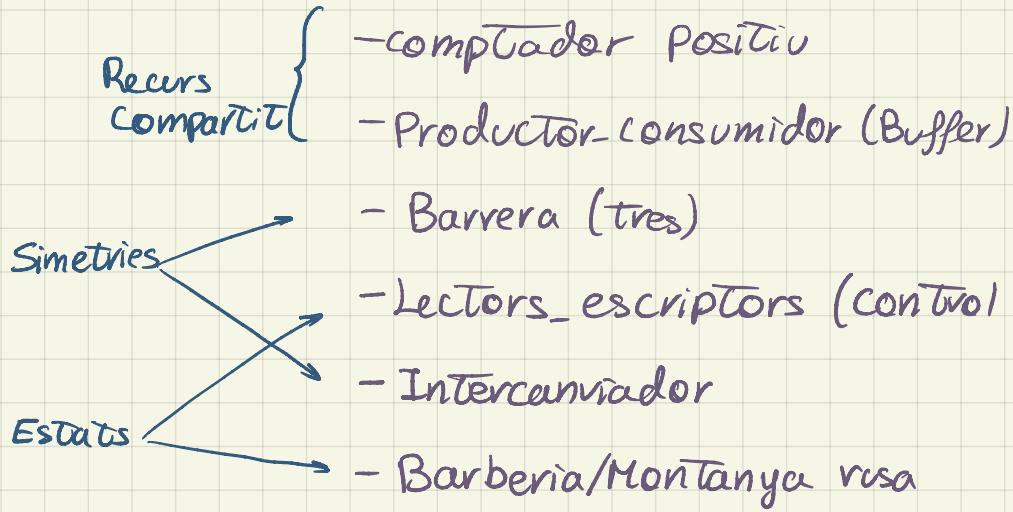
Public class Pont

```
Private Lock mon;
Private boolean sentitPont;
Private Condition potpasar;
Private int numCotxes;
```

NO ESTA
ACCESAT

```
Public void entrar (boolean sentit) {
    mon.lock();
    while (sentitPont != sentit)
        potpasar.wait();
    numCotxes++;
    mon.unlock();
}
```

```
Public void sortir() {
    mon.lock();
    numCotxes--;
    if (numCotxes == 0) {
        potpasar.signalAll();
        if (sentitPont)
            sentitPont = false;
        else
            sentitPont = true;
    }
    mon.unlock()
}
```



public class MontanyaRusa {

```
protected final int _poderPujar, _PoderCircular;
protected int estat = -PoderPujar;
protected condition poderPujar, PoderCircular;
protected int capacitat, numPassatgers;
```

```
public void ferVolta() { ... }
public void arrencar() { ... }
public void Parar() { ... }
```

Passatgers

```
while (true) {
    // socialitzar
    mon. ferVolta();
```

```
public void ferVolta() {
    mon. lock();
    while (estat != PoderPujar)
        poderPujar. await();
```

```
    numPassatgers++;
    if (numPassatgers == capacitat) {
        estat = -PoderCircular;
        PoderCircular. signalAll();
```

```
    while (estat != PoderBaixar)
        poderBaixar. await();
```

```
    numPassatgers--;
    if (numPassatgers == 0) {
        estat = -PoderPujar;
        PoderPujar. signalAll();
```

↓
mon. unlock

Vago'

```
while (true) {
    mon. arrencar();
    // fer volta
    mon. parar();
```

Public void arrancar ()

```
mon. lock();
while (estat != PoderCircular)
    PoderCircular. await();
```

```
mon. unlock();
```

Public void pujar ()

```
mon. lock();
estat = -PoderBaixar;
PoderBaixar. signalAll();
mon. unlock();
```

Public class Comptable implements Runnable {

 Oracle mon;

```
    public void run() {
        while (true) {
            int op1 = ...
            int op2 = ...
            int resultat = mon.suma(op1, op2);
        }
    }
```

Public class Mag implements Runnable {

 Oracle mon;

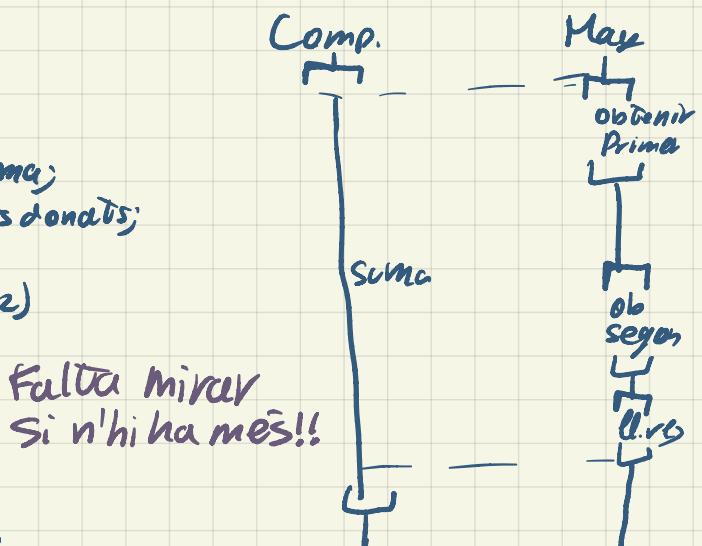
```
    public void run() {
        while (true) {
            int op1 = mon.obtenirPrimer();
            int op2 = mon.obtenirSegon();
            int resultat = op1 + op2;
            mon.llivarResultat(resultat);
        }
    }
```

Public class Oracle

```
Private lock mon;
Private int op1, op2, res;
private Condition valors, persuma;
private boolean sumafeta, valorsdonats;
```

```
Public int Suma(int op1, int op2)
{
    mon.lock();
    op1 = op1;
    op2 = op2;
    valorsdonats = true;
    While (!sumafeta)
        persuma.await();
    persuma.signal();
    valors.signal();
    return res;
    mon.unlock();
}
```

```
Public int ObtenirPrimer()
{
    mon.lock();
    While (!valorsdonats)
        valors.await();
    return op1;
}
```



```
Public int obtenirSegon()
{
    return op2;
}
```

```
Public void llivarResultat(int resultat)
{
    res = resultat;
    sumafeta.signal();
}
```

```
Public class Oracle {
    protected lock mon;
    protected int estat;
    protected final int
        PoderSumar=0
        -mag =1
        -resultado=2
        =3
```

```
protected Condition poderSumar, magia;
protected int op1, op2;
```

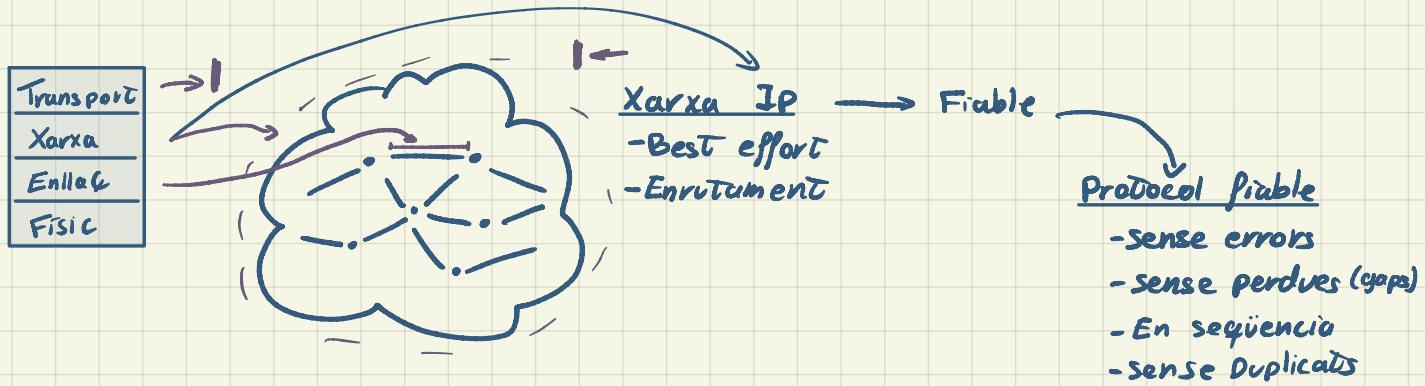
```
Public int Suma(int op1, int op2) {
    Try{mon.lock();
        while (estat!=PoderSumar)
            PoderSumar.await();
    }
    This.op1=op1;
    This.op2=op2;
    estat=_mag;
    magia.signal();
    while (estat != _resultado)
        Sumafeta.await();
    estat=PoderSumar;
    PoderSumar.signal();
    return resultado;
    {finally}
        mon.unlock();
    }
```

```
Public int obtenerPrimer()
    mon.lock()
    While (estat!=_mag)
        magia.await();
    estat=_calculant;
    mon.unlock();
    return op1;
    }
```

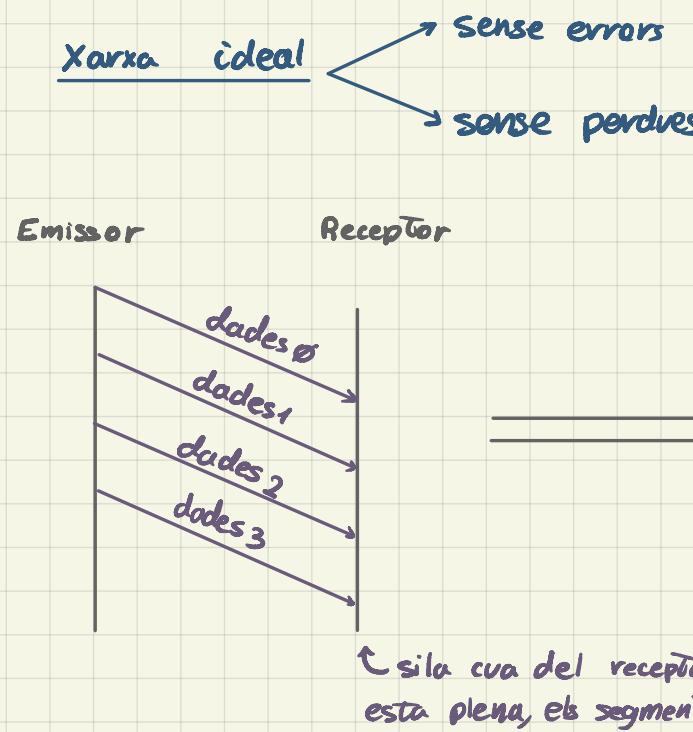
```
Public int obtenerSegundo()
    return op2;
{
```

```
Public void llivarResultado(int resultado)
    mon.lock();
    resultado=res;
    estat=_resultado
    Sumafeta.signal();
    mon.unlock();
    }
```

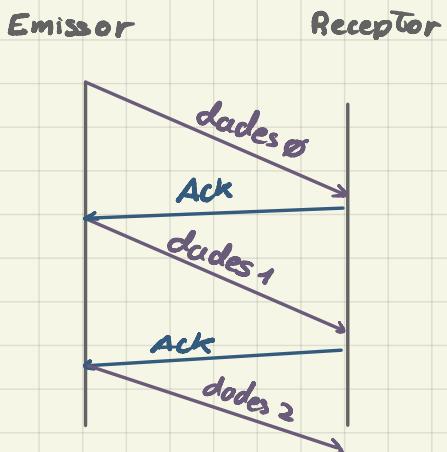
Tema 3 → Protocol



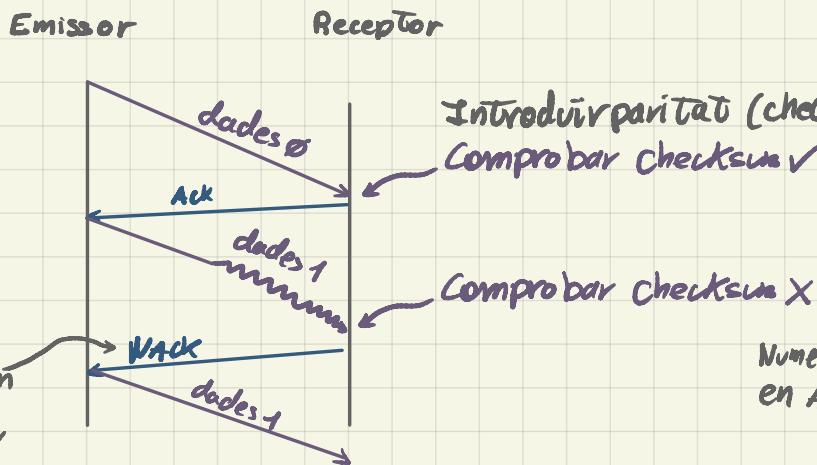
Protocol: defineix el format i l'ordre dels missatges que s'intercanviuen entre dues entitats en una comunicació; així com les accions a realitzar en rebre/enviar aquests missatges.



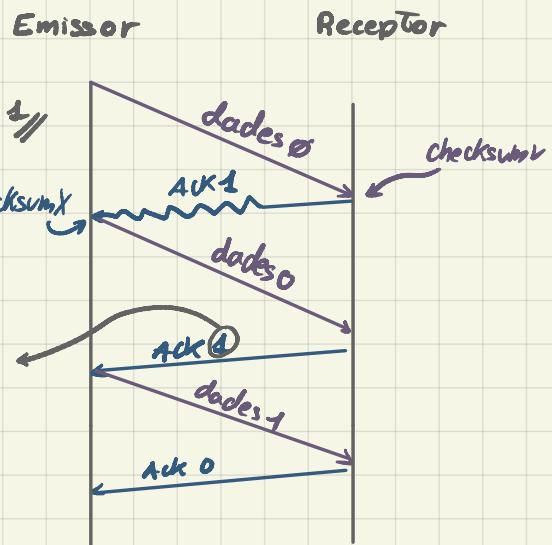
1) Control de Flux



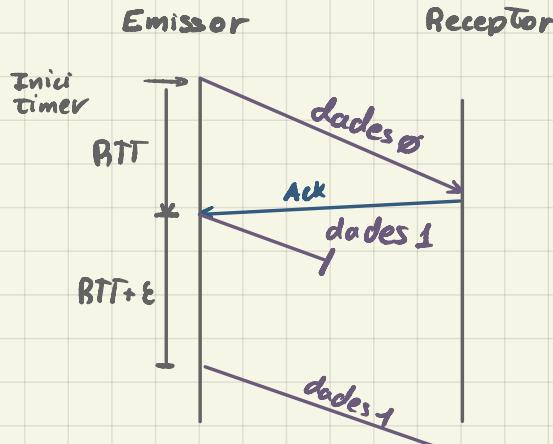
2) Xarxa amb errors



Xarxa amb errors



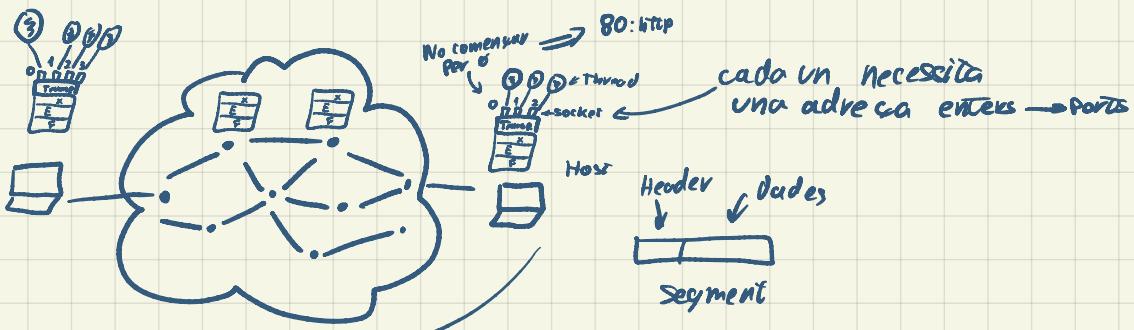
3) Xarxa amb errors i perdudes



Esperant un temps → Timer suficient

- mínim RTT
- màxim RTT+E

S'les dades arriben quan ja s'ha fet el següent ACK, aquestes s'eliminen



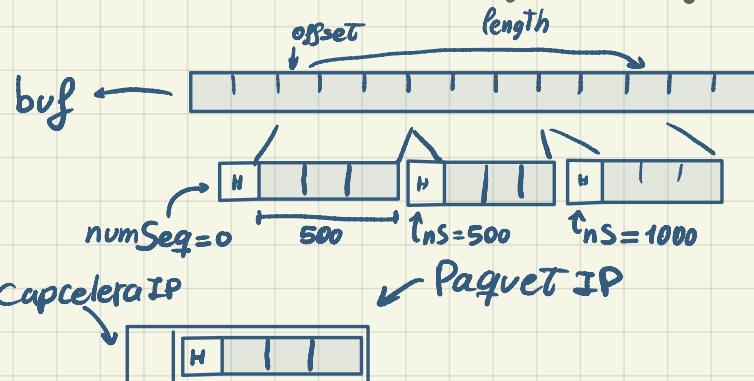
Extrem entrega Host-host (Ip) → No fiable

↓
entrega Thread-Thread
No fiable (UDP User Datagram Protocol)
Fiable (TCP Transmission Control Protocol)

* Senyal UDP



Public void sendData(byte[] buf, int offset, int length)



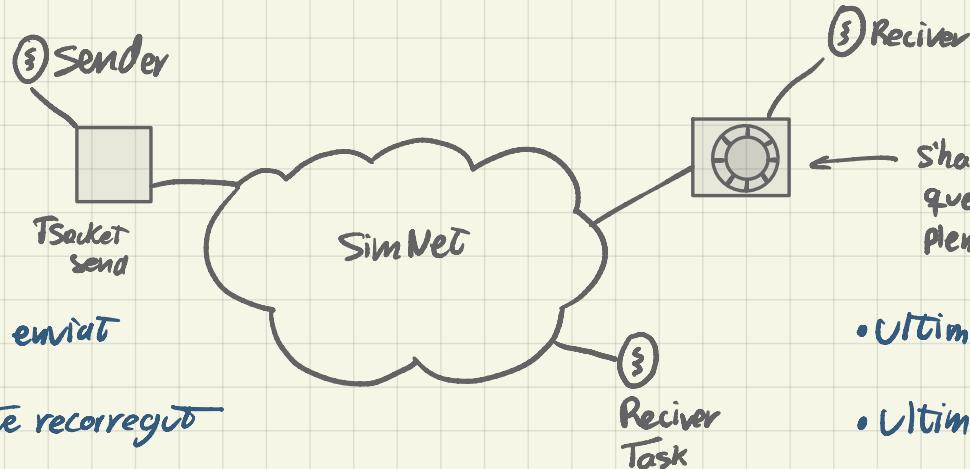
TCP(Fiable)



num del segment
1 byte del flux de bytes

Reconeixement acumulatiu
Num seq Ack = num bytes rebuts fins al moment

Conexió desconexió



- últim byte rebut
v
- últim byte lleyit (entregat al receptor)

$$rcv\ window = \text{Capacitat} - (\text{últim Byte Rebut} - \text{últim byte lleyit})$$

(espai lliure)

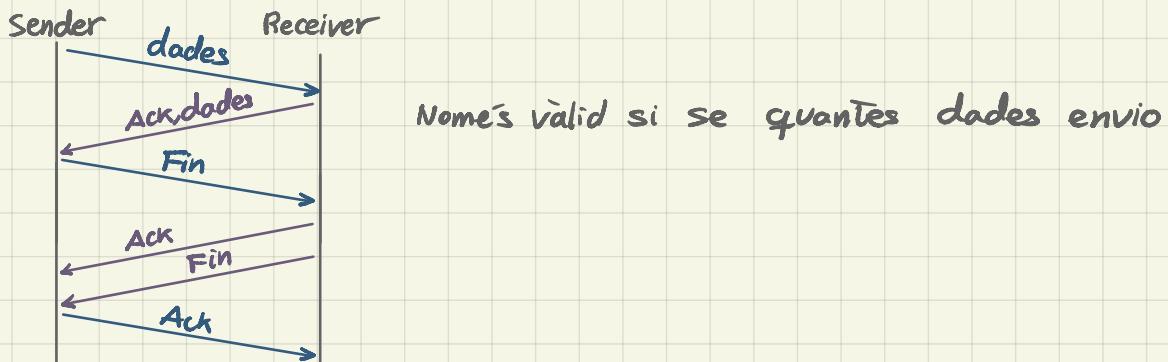
Bytes allunyats

Fines tra zero → s'arregla enviant bytes d'un en un.

Conexió

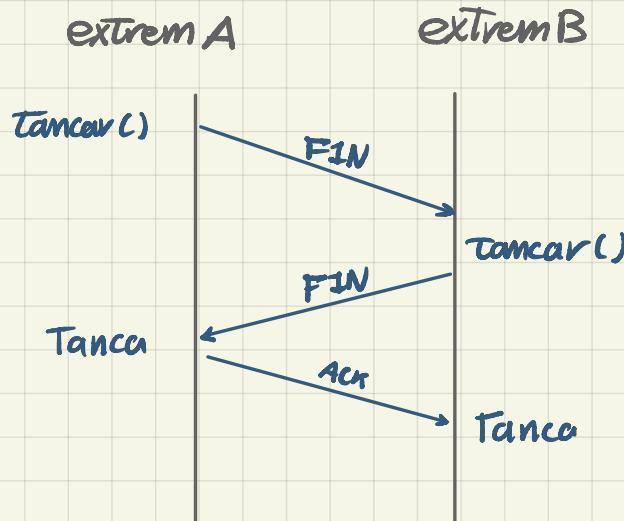


Desconexió

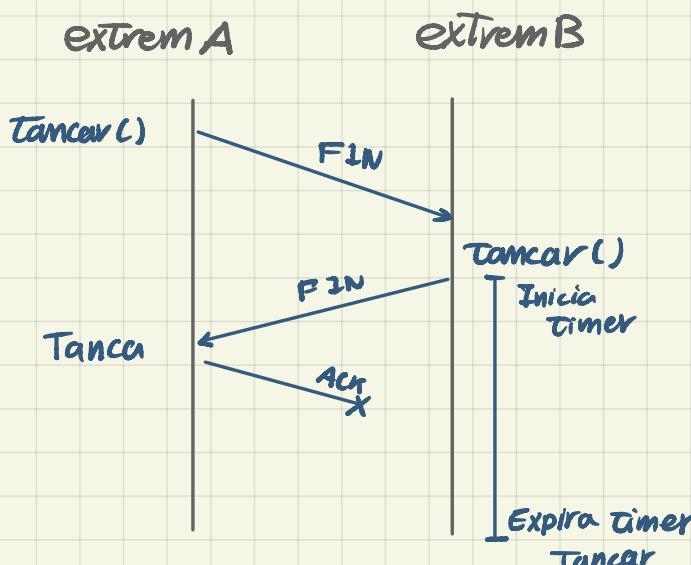


Protocol desconnexió

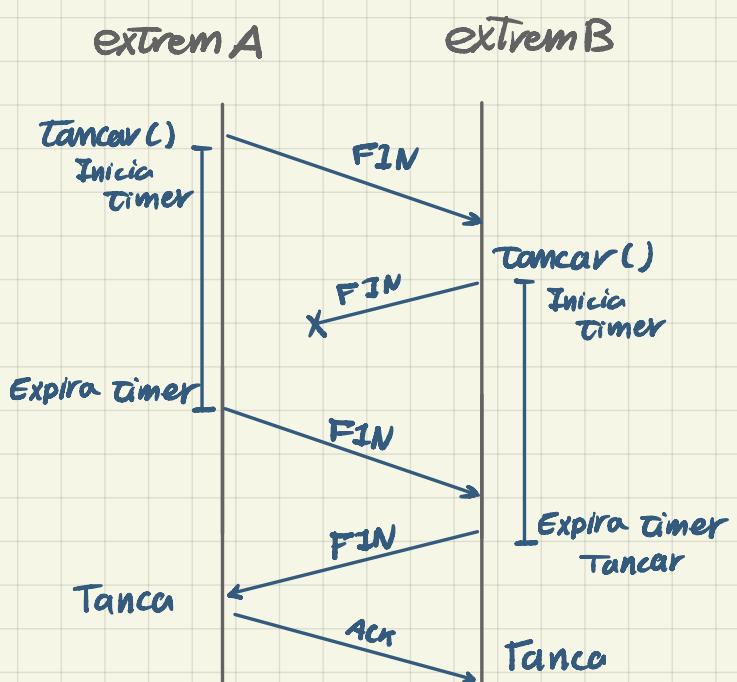
S'ha d'intentar que els dos extrems tenguin notificació del tancament



Cas 1: es perd l'últim ACK

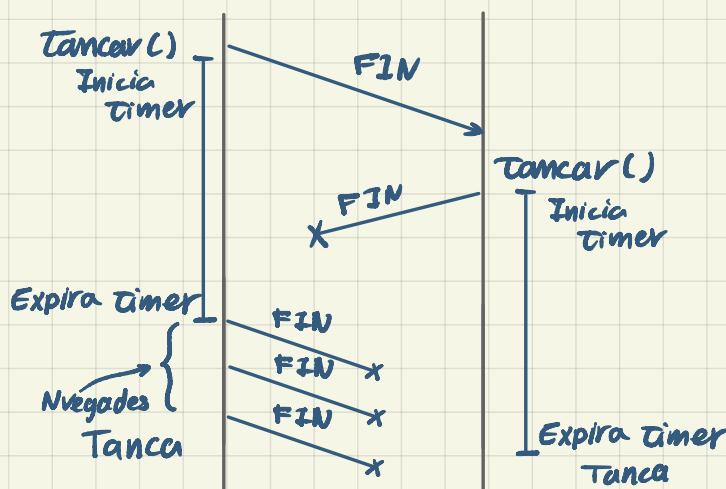


Cas 2: es perd el segon FIN

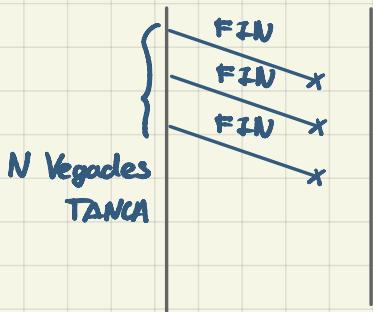


Curs 3: es perd el segon FIN i la resta

extrem A extrem B



Queda la conexió nus oberta
Si es perd el tr FIN N vegades



Connexió desconexió 1

El timer pot ser exponential backoff cada cop que acaba es duplica la durada.

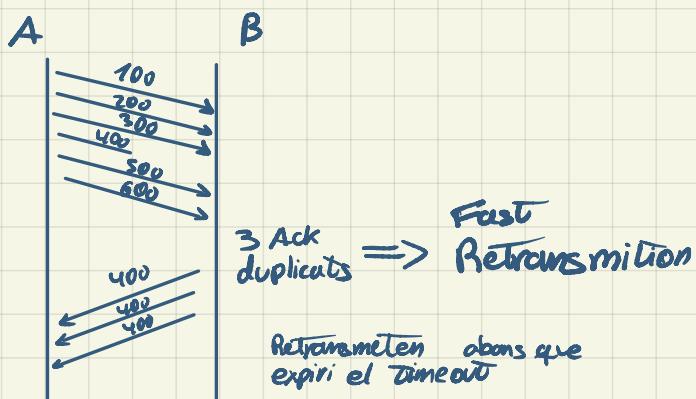
CONTROL DE CONGESTIÓ

Event

- Arribada d'un segment amb num seqüència esperat i tots els segments reconeguts
- Arribada d'un segment amb num seqüència esperat i tenim segment pendent de reconeixer
- deteccions GAP → Arriba un segment amb num seqüència mes gran es perdi.
- Arribada d'un segment que completa el gap (potser parcialment)

Acció Receptor TCP

- Retrasar ACK si espera 500ms a l'arribada d'un altre segment, si no arriba enviar ACK
- Envio ACK IMMEDIATAMENT reconeixent els dos següents
- Envio ACK IMMEDIATAMENT amb num seq esperat
- Envio ACK IMMEDIATAMENT



Control de flux



control de congestió



"Llei Conservació de segments no injectar segment xarxa fins estar segurs que un ha sortit"

Control de Congestió

1) Com s'adona l'emissor TCP que hi ha congestió?

- Expira el timer (Més greu) / actuar diferent
- ACK's duplicats

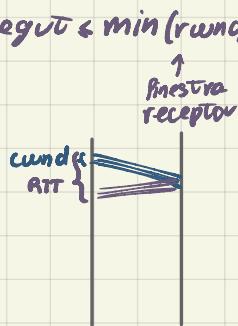
2) Com limita l'emissor l'enviament de dades a la xarxa?

- L'emissor controla una variable **cwnd** finestra de congestió

Suposem $rwnd \gg cwnd$ $\text{ultimo Byte Enviat} - \text{ultimo Byte Reconegut} < \min(rwnd, cwnd)$

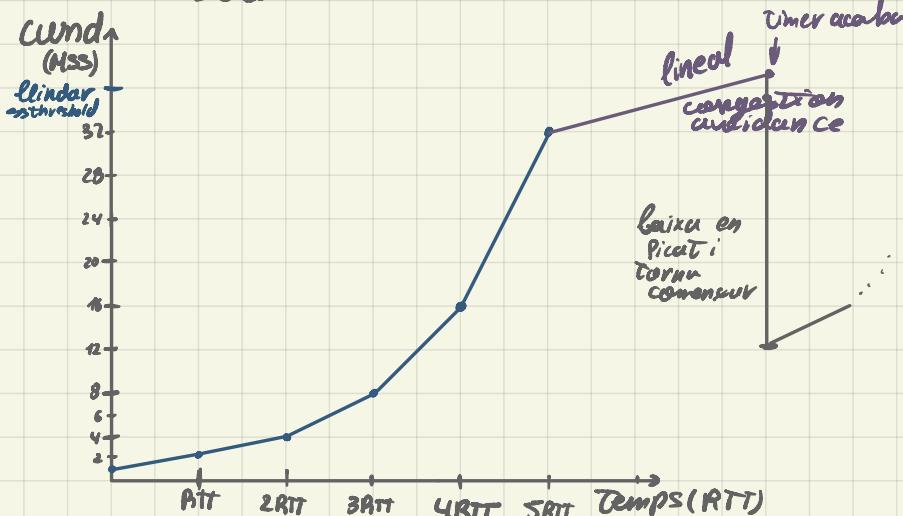
Quina és la Velocitat de Transmissió de l'emissor?

$$\text{Velocitat de Transmissió de l'emissor TCP} = \frac{cwnd}{RTT} \text{ B/s}$$

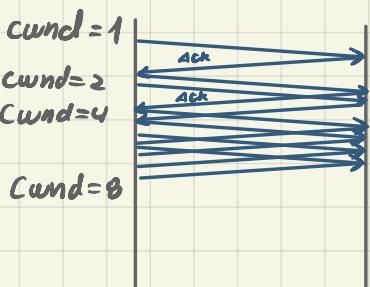


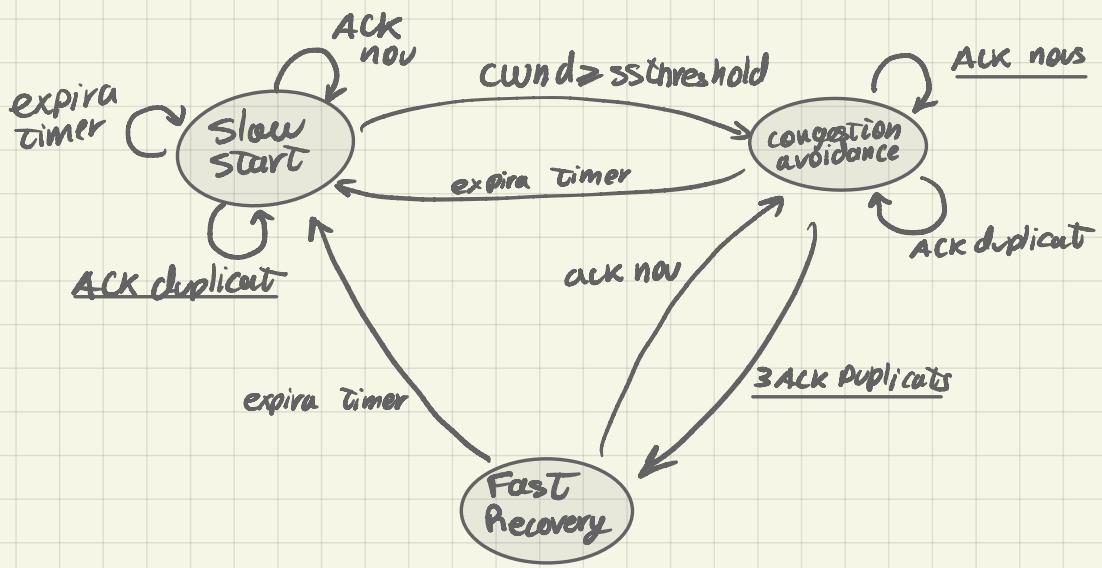
3) Quin algoritme es fa servir per canviar la tasa d'enviament en funció de la congestió?

slow start

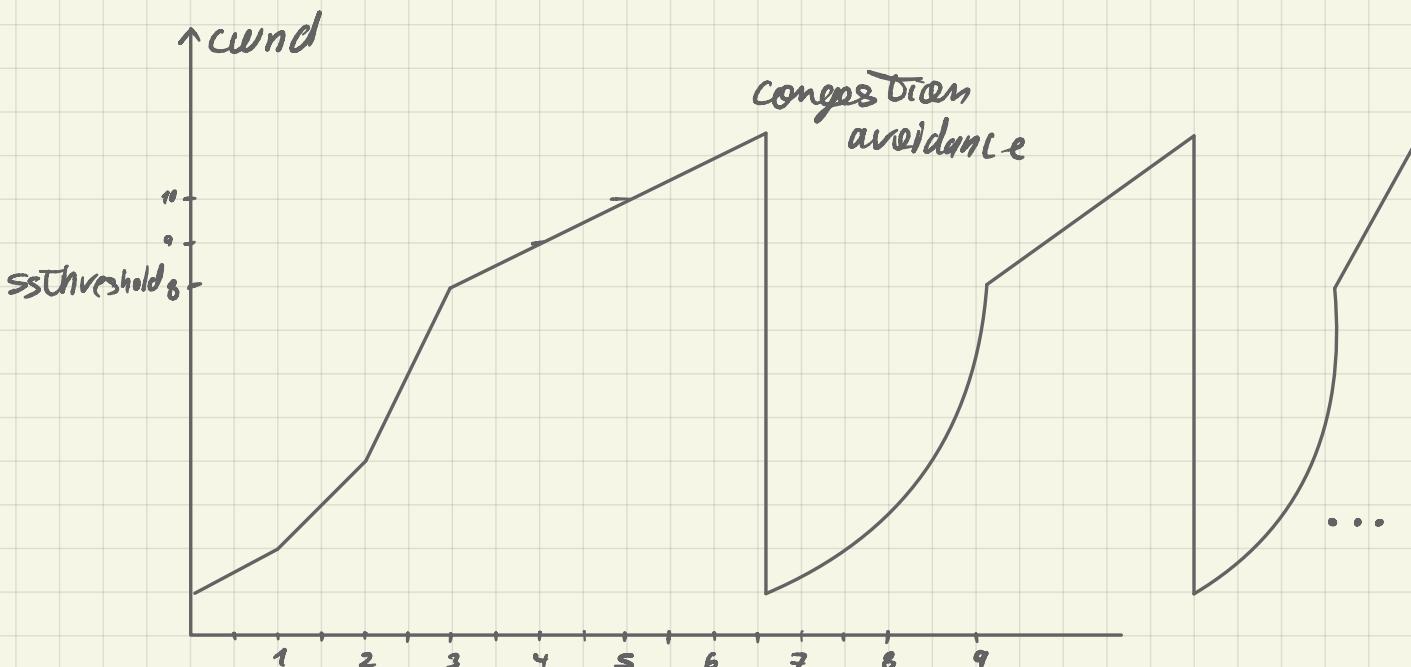
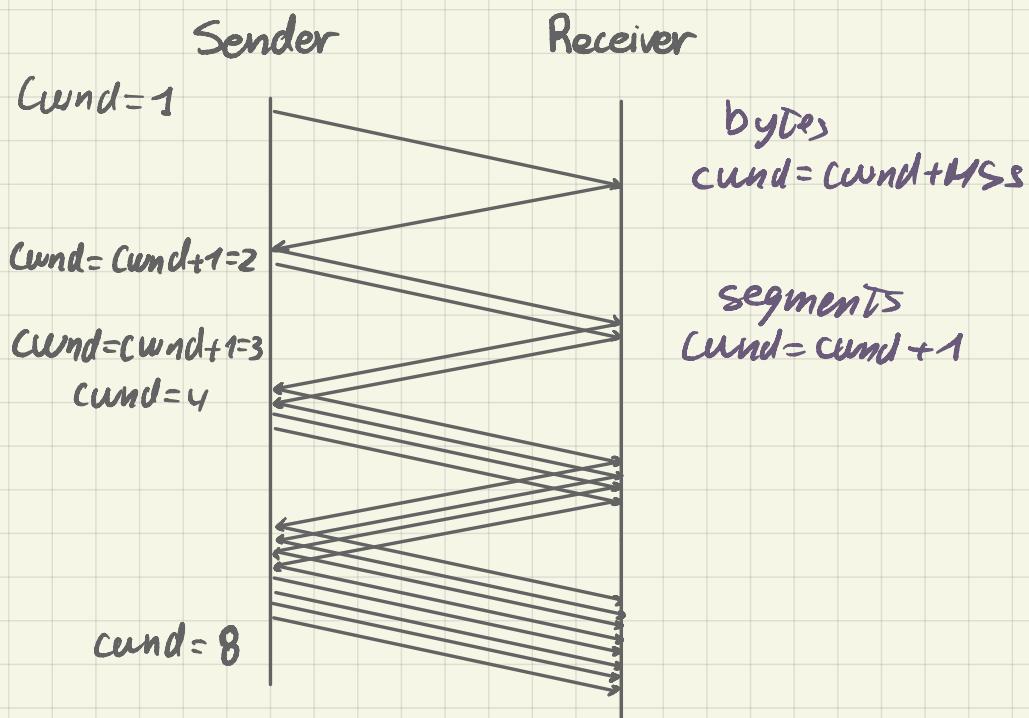


Percorreg. ACK
 $cwnd = cwnd + 1 \text{ MSB}$





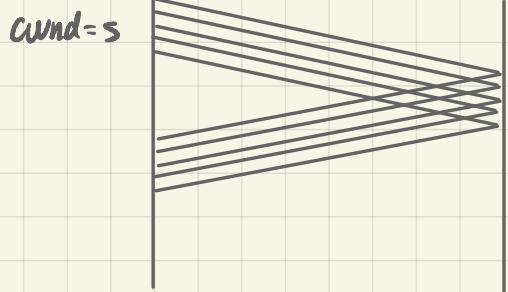
Control de congestión



Congestion Avoidance

Sender

Receiver



Per cada ACK

$$cwnd = cwnd + \frac{1}{cwnd}$$

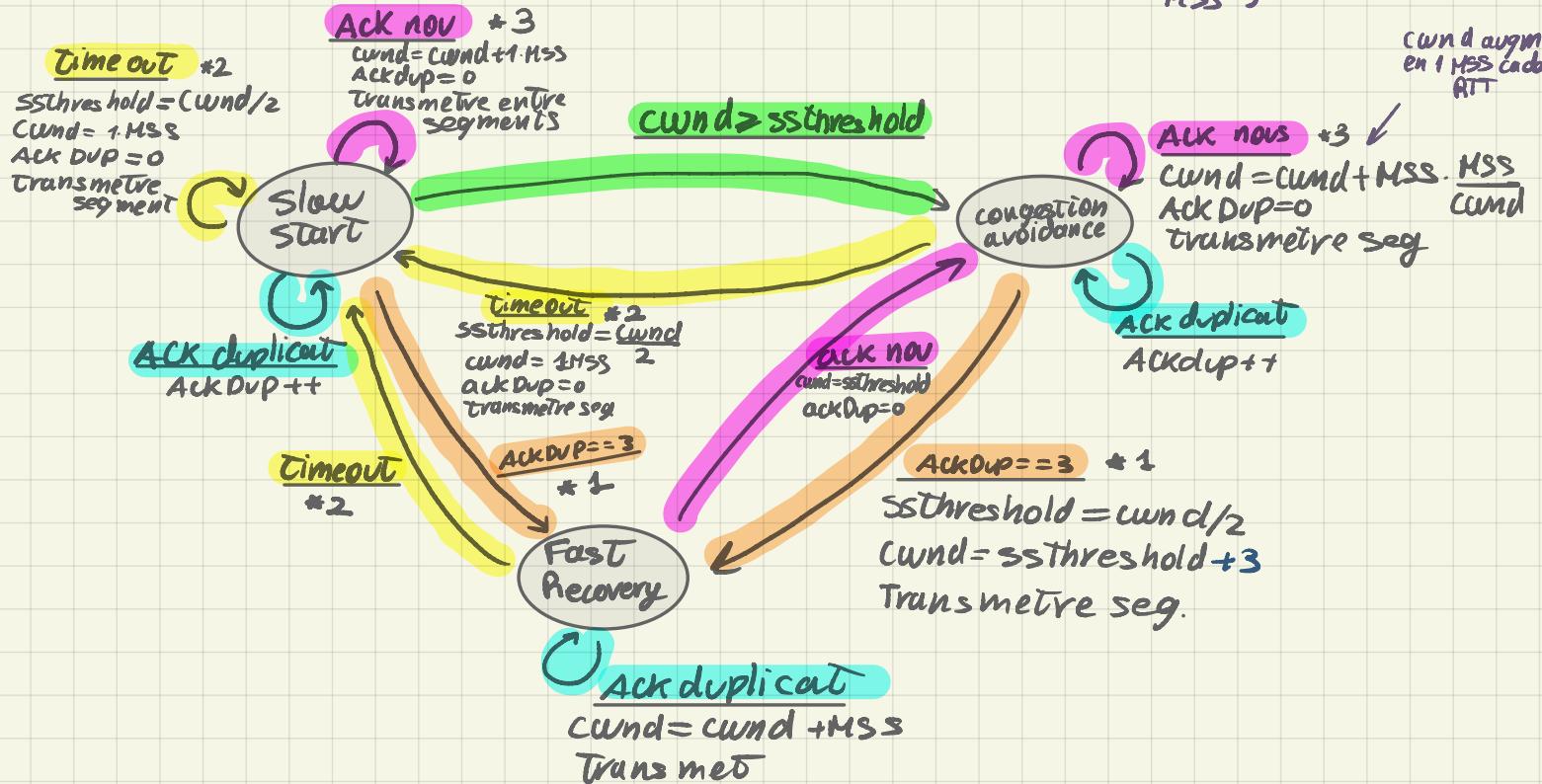
Segments

Bytes

$$cwnd = cwnd + MSS$$

$\frac{1}{cwnd} \times MSS$ } Segments que se envían en RTT

cwnd aumenta en 1 MSS cada RTT



Productor consumidor distribuit

class Productor {
 Buffer buf;

```
    public productor (Buffer b) {
        buf = b;
    }
```

```
    public void run() {
        while(true) {
            //generar elemento
            buf.out(e);
        }
    }
}
```

class Buffer representant {
 JAVA

```
    Protected Socket s;
    Protected BufferedReader entradaTxt;
    Protected PrintWriter sortidaTxt;
```

```
    Public void BufferRepresentant () {
        s=new Socket (const. adreçadestí, const. portdestí);
        sortidaTxt=new PrintWriter (s.getOutputStream ());
    }
```

Entrada TxT= new BufferedReader (new InputStreamReader (s.getInputStream ()));

Public void put (object e) {

}

public object get () {

}

