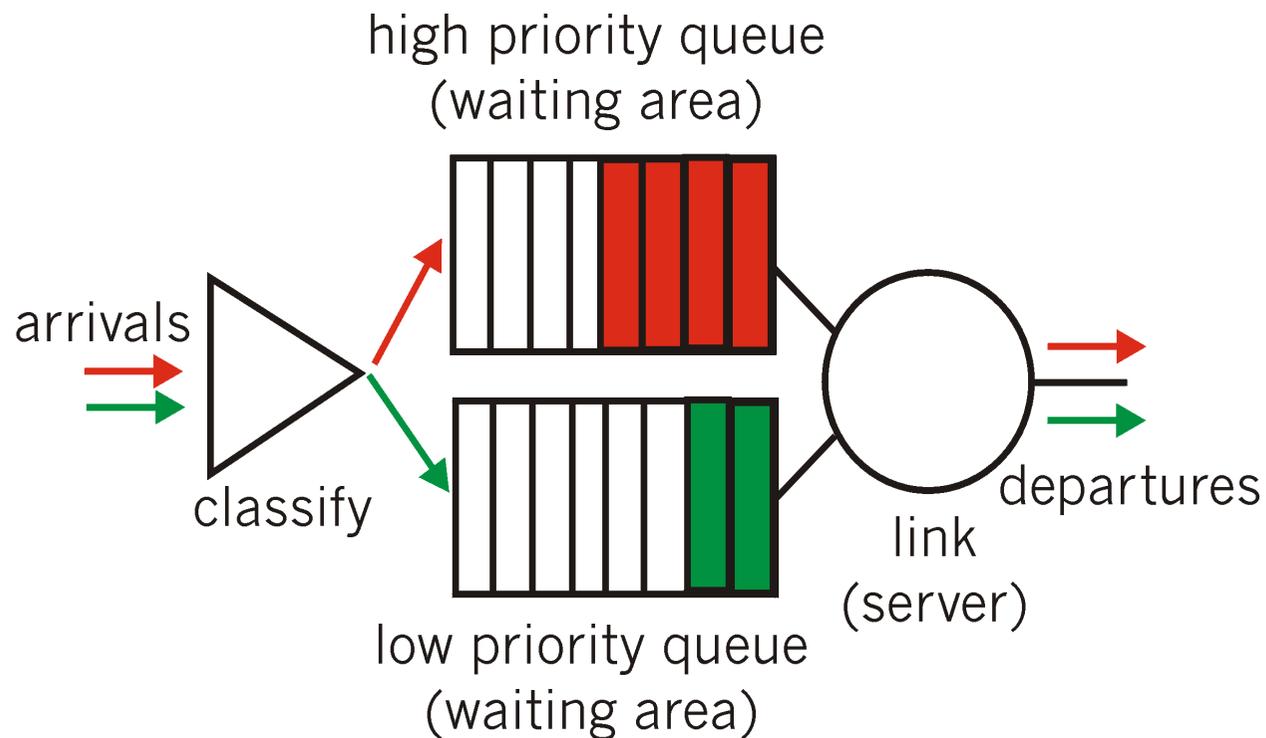


# Priority Queueing

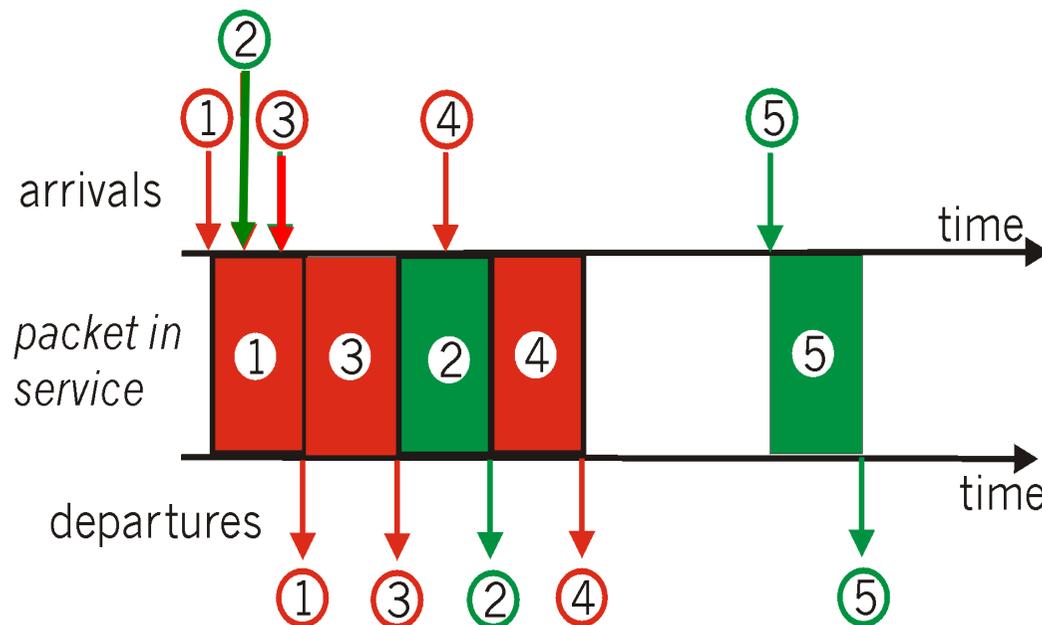
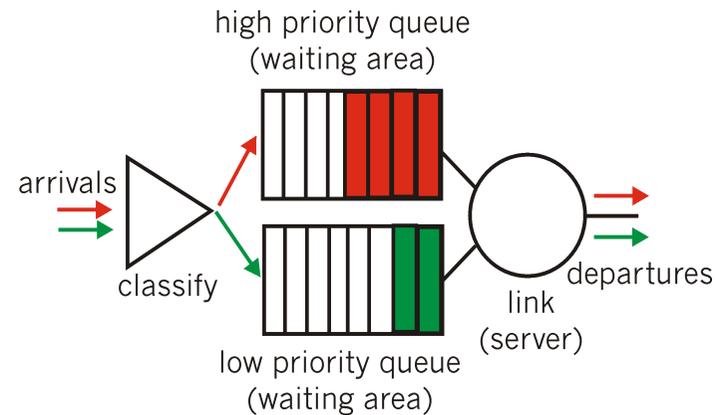
# Priority Queueing (PQ)

- Paquetes en cola de mayor prioridad se envían siempre antes que paquetes en colas de menor prioridad
- En cada cola FCFS



# Priority Queueing (PQ)

- Paquetes en cola de mayor prioridad se envían siempre antes que paquetes en colas de menor prioridad
- En cada cola FCFS



# Priority Queueing (PQ)

- Asegura que el tráfico importante reciba un servicio rápido
- Puede crear inanición (*starvation*), es decir, dejar fuera de servicio a tráfico menos prioritario
- Menor retardo en cola medio para un flujo a costa de mayor para otros
- *Multilevel priority with exhaustive service*: Los paquetes en una cola de menor prioridad no se envían hasta que todas las colas de mayor prioridad están vacías



# Priority Queueing

- El número de niveles de prioridad depende del número de clases de retardo a crear
- Son típicas al menos 3:
  - Prioridad alta: mensajes urgentes, por ejemplo protocolos de control de red
  - Prioridad media: servicio garantizado
  - Prioridad baja: best-effort
- Otra posibilidad:
  - Prioridad alta: voz
  - Prioridad media: vídeo
  - Prioridad baja: resto de datos
- Es vital un correcto control de admisión y policing para todo lo que no sea la clase más baja
- El reparto del BW entre las clases no es max-min fair
- Sencillo de implementar



IT'S  
NOT  
FAIR !!

# Reparto justo y protección

# Reparto justo y protección

- Flujos con requerimientos estrictos deben tenerlos garantizados independiente de esta “justicia”
- Reparto justo es importante para flujos best-effort
- Reparto decimos que es justo si satisface un *max-min fair share*



- Scheduling es normalmente una decisión local al nodo de conmutación pero la “justicia” para un flujo es un objetivo global
- (...)



# Reparto justo y protección

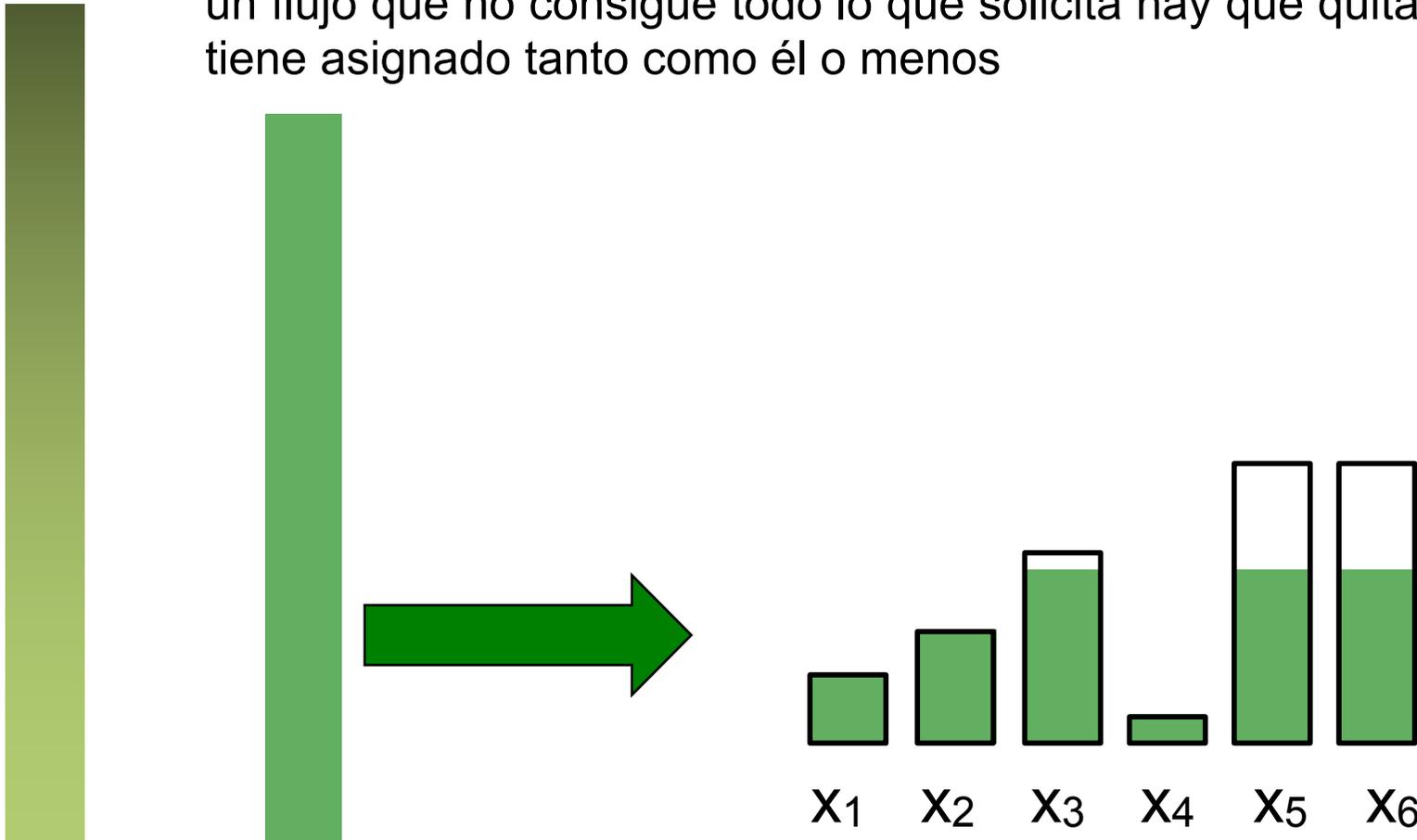
- Lograr justicia global con flujos cambiantes no es tan sencillo
- La protección implica que un flujo que envíe más que su asignación justa no afecte al resto
- Un planificador que haga un reparto justo ofrece protección
- La justicia es una característica del resultado del reparto, independientemente de cómo se llegue a él
- (...)



# Reparto justo (*max-min fair*)

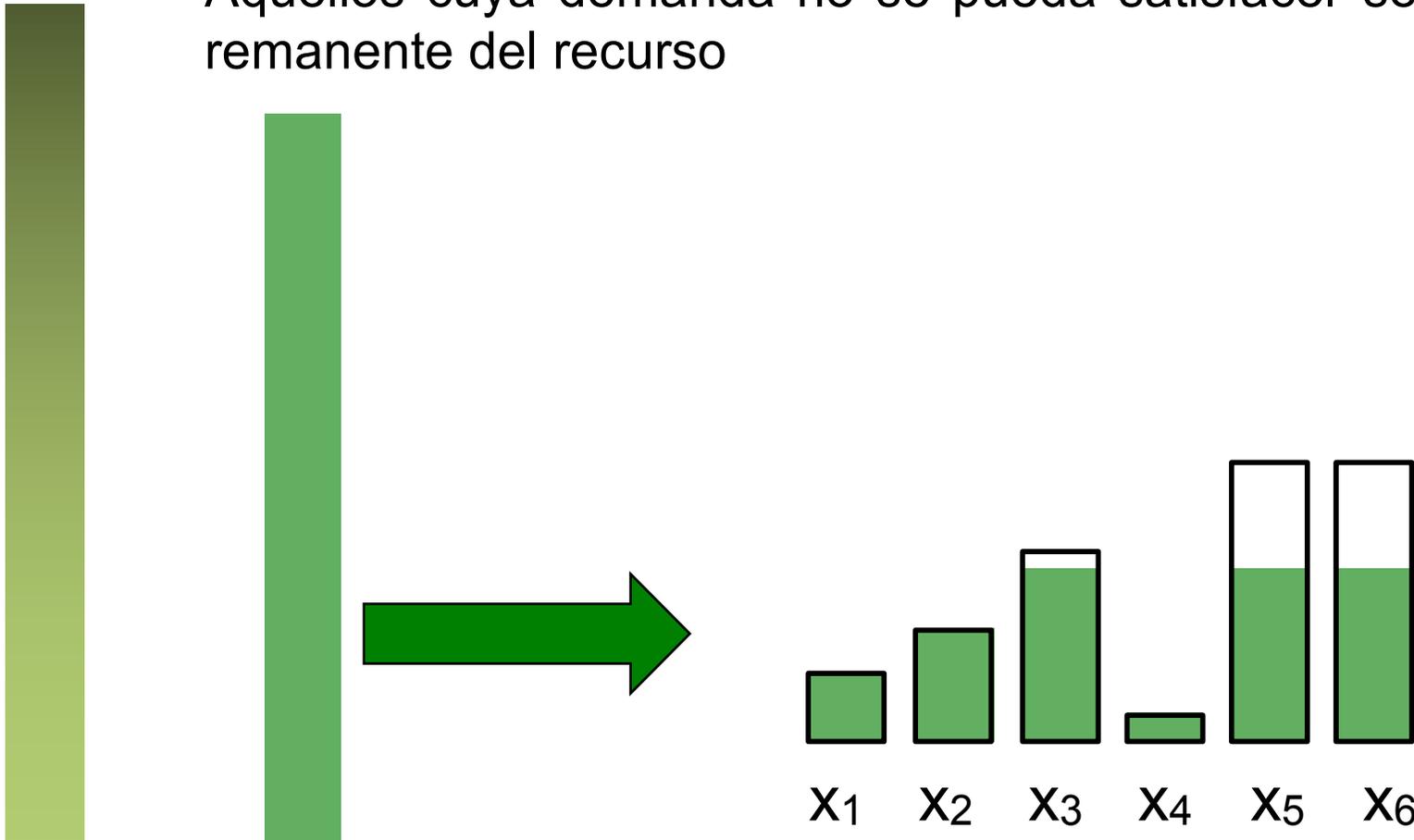
# Reparto justo (max-min fair)

- Para dividir recursos entre un conjunto de usuarios, todos con iguales “derechos” pero diferentes demandas
- De forma simple: a los que piden “poco” se les da lo que piden y lo que sobra se reparte entre los que piden “mucho”
- En una situación de reparto max-min-fair, para mejorar la asignación a un flujo que no consigue todo lo que solicita hay que quitarle a otro que tiene asignado tanto como él o menos



# Reparto justo (max-min fair)

- Todos reciben al menos su parte proporcional, salvo que hayan pedido menos
- Asignar recursos en orden creciente de demanda
- Ningún cliente recibe más de lo que solicita
- Aquellos cuya demanda no se pueda satisfacer se reparten el remanente del recurso

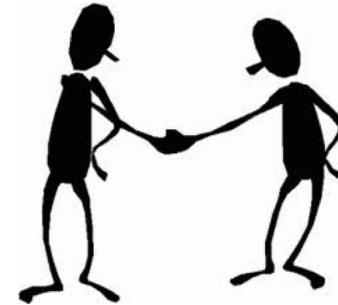


# Reparto justo (max-min fair)

- Flujos  $1, \dots, n$
- Demandas  $x_1, \dots, x_n$
- Demandas ordenadas  $x_1 \leq x_2 \leq \dots \leq x_n$
- Capacidad a repartir  $C$
- Inicialmente asignar  $C/n$  al flujo 1
- Si esto es más que lo que necesita ( $C/n > x_1$ ) lo que sobra se repartirá entre el resto
- Asignar al flujo 2:  $C/n$  más la parte que le corresponde de lo que sobró del flujo 1, es decir:

$$\frac{C}{n} + \frac{\frac{C}{n} - x_1}{n-1}$$

- Esto puede ser más que lo que el flujo 2 necesita, así que lo que sobra se puede repartir entre el resto
- *Al final todos tienen lo que han pedido o si no había suficiente para eso no tienen menos que cualquier otra fuente que ha pedido más*



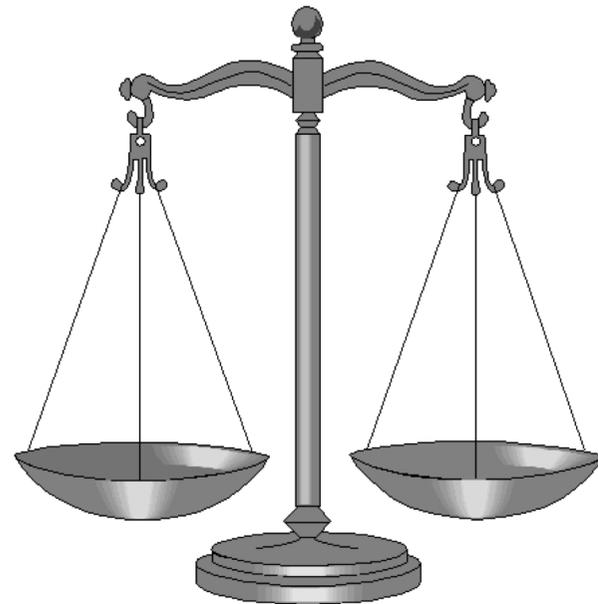
# Max-min Fair (Ejemplo)

- Recurso: 10
- Demandas: 2, 2.6, 4 y 5
- $10/4 = 2.5$ 
  - Demasiado para el primer cliente
  - Asignarle 2 y queda 0.5
- Ese 0.5 repartirlo entre los otros 3:
  - $0.5/3 = 0.16666\dots$
  - Asignaciones [2, 2.66, 2.66, 2.66]
  - Demasiado para el segundo cliente
  - Asignarle 2.6 y quedan 0.0666....
- Repartir ese 0.0666... entre los otros 2:
  - $(2.5+0.5/3-2.6)/2 = 0.03333\dots$
  - Asignaciones [2, 2.6, 2.7, 2.7]

# Processor Sharing

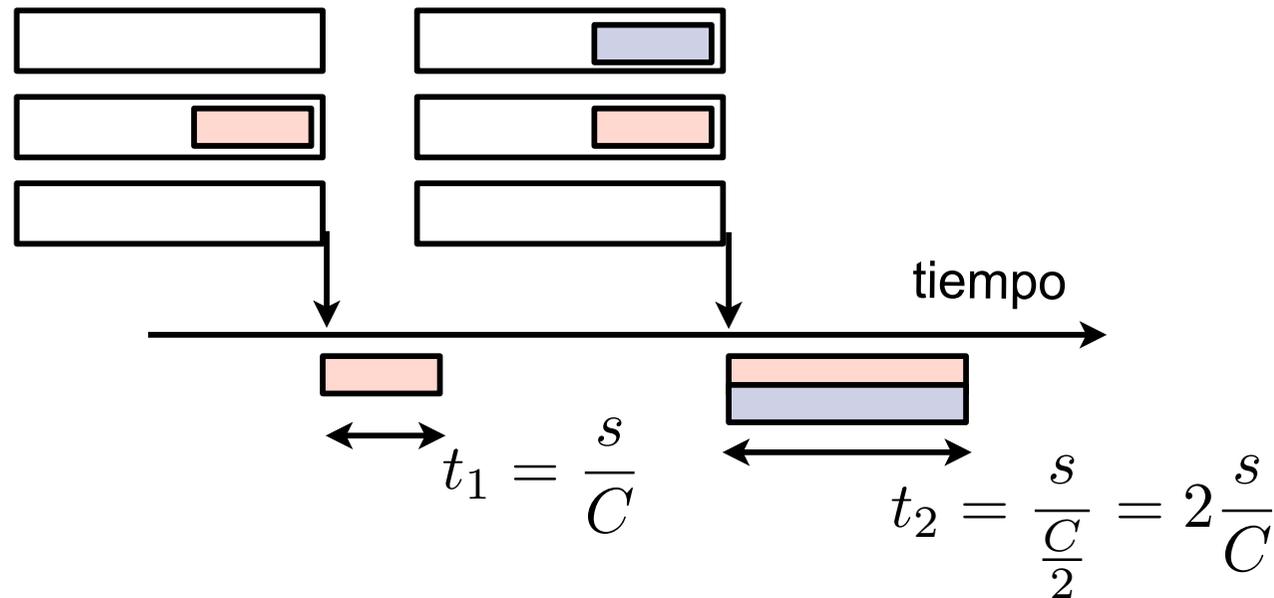
# PS

- Para best-effort querríamos un reparto *max-min fair*
- Esto se puede lograr con un scheduler llamado *Processor Sharing*
- Es un planificador *work-conserving*
- Sirve de forma simultanea todas las colas, repartiendo la capacidad
- O se puede decir que las sirve por turnos (round robin) pero sirviendo una cantidad infinitesimal de cada una
- (...)



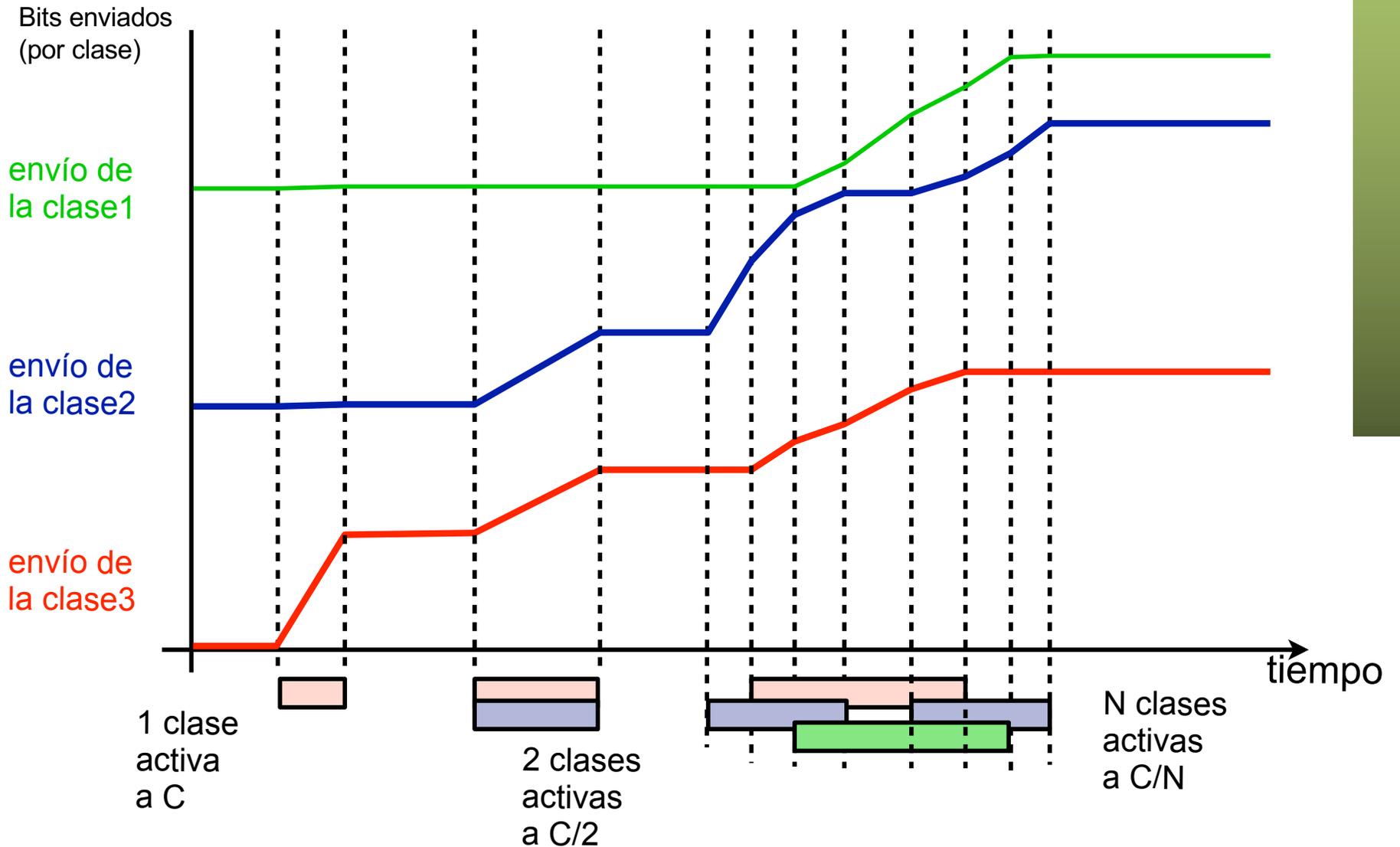
# PS

- Si una cola está vacía pasa a la siguiente, de forma que su tiempo se está repartiendo entre el resto (y de ahí el max-min)
- Aproximación de tráfico como un fluido
- Es un planificador ideal e imposible de implementar, aunque se puede aproximar



# Processor Sharing

## Ejemplo



# Round Robin

# Round Robin (RR)

- Opera en “turnos” (*rounds*), conservativo en trabajo
- En cada turno visita cada cola (en *round-robin*)
- En cada cola FCFS
- Se sirve un número de paquetes o paquetes durante un cierto tiempo fijo (la diferencia es cómo afectan sus tamaños)

