Scheduling (2)

Area de Ingeniería Telemática
http://www.tlm.unavarra.es

Redes
4º Ingeniería Informática

Temario

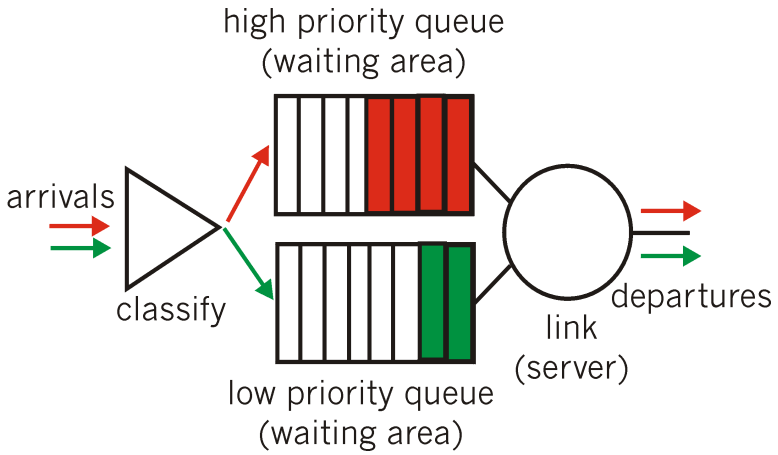
1. Introducción a las redes
2. Encaminamiento
3. Transporte extremo a extremo
4. **Arquitectura de conmutadores de paquetes**
5. Tecnologías para redes de área local
6. Tecnologías para redes de área extensa y última milla
7. Conmutación de circuitos

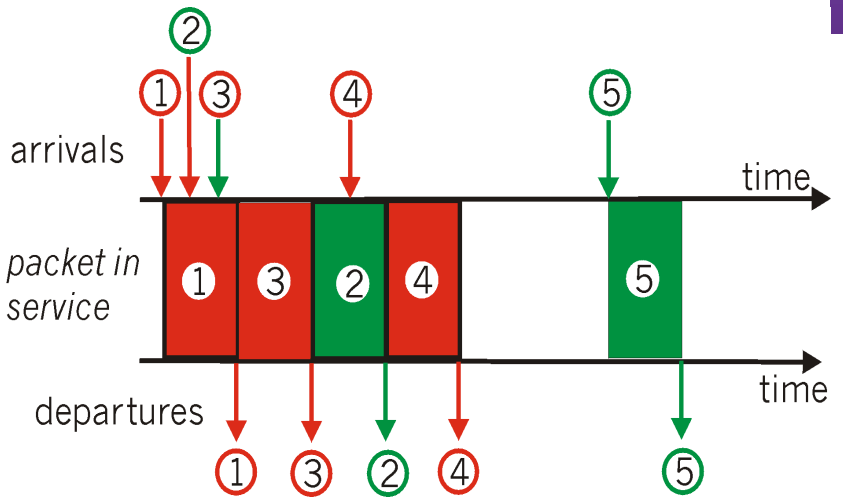
Objetivos

- Conocer las características y el funcionamiento de los planificadores más habituales
- Saber que se pueden calcular cotas a parámetros de red estadísticos que afectan al tráfico

Priority Queueing (PQ)

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





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:
  - Prioriad alta: voz
  - Prioridad media: vídeo
  - Prioridad baja: resto de datos

- Un flujo de alta prioridad puede “ahogar” a otros de prioridad más baja
- Es vital un correcto control de admisión y policing para todo lo que no sea la clase más baja
- Sencillo de implementar
- El reparto del BW entre las clases no es max-min fair

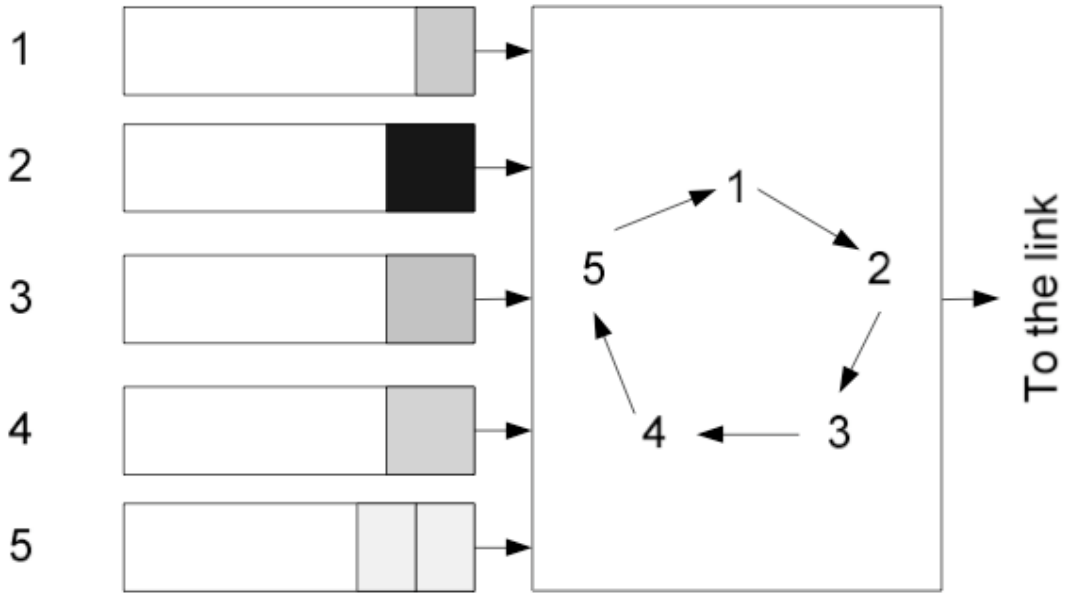
Priority Queueing

Ejemplo

- Se conoce el tamaño de la ráfaga más larga que puede llegar de un flujo (b_i)
- Para el flujo de prioridad mayor i = 1 el b_1 debe ser inferior al retardo de peor caso que se busque
- Para el flujo de prioridad i = 2 el retardo máximo es b_1+b_2
- El flujo de prioridad i = k sufre un retardo de caso peor de $\sum_{i=1}^{k} b_i$

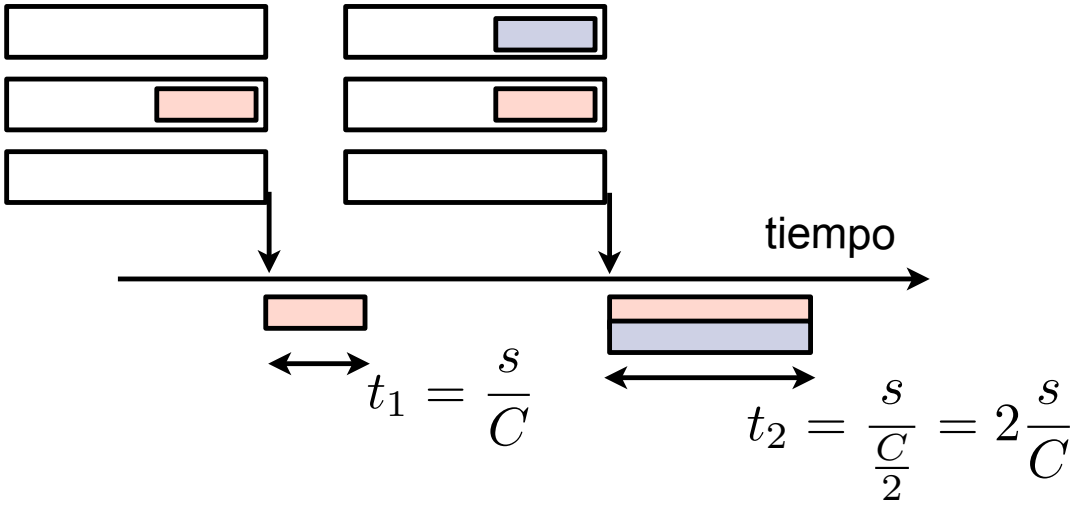
Round Robin (RR)

- Opera en “turnos” (*rounds*)
- 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)



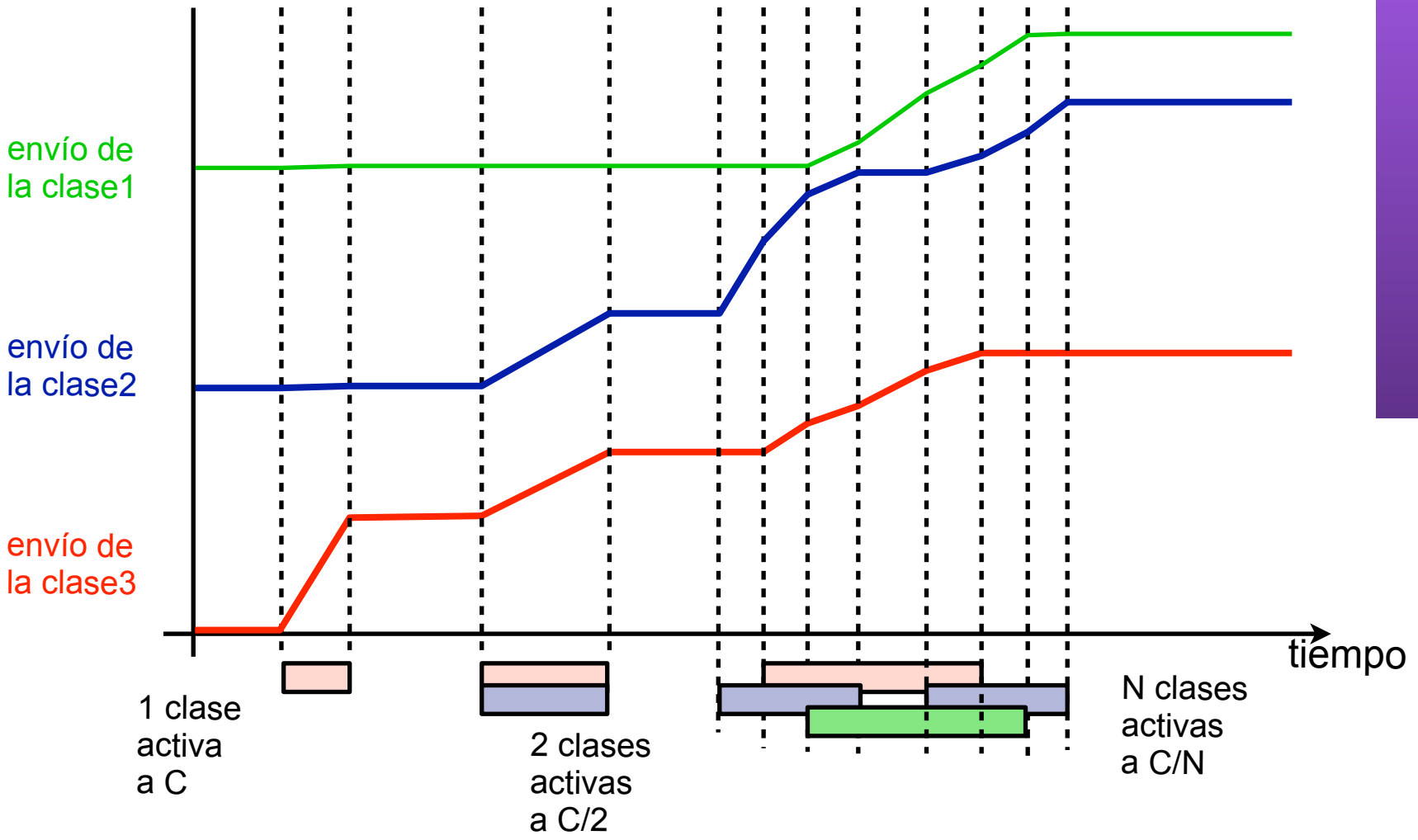
PS

- 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
- 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

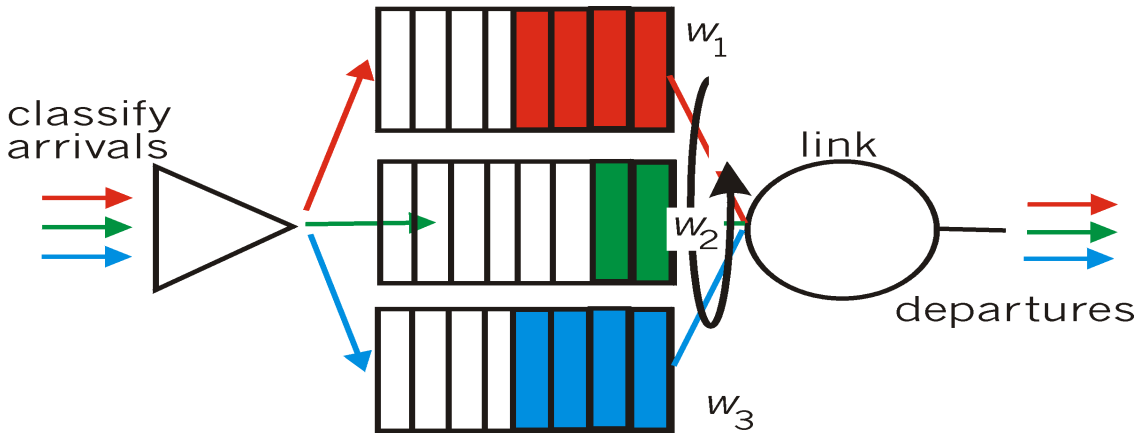
Ejemplo



GPS

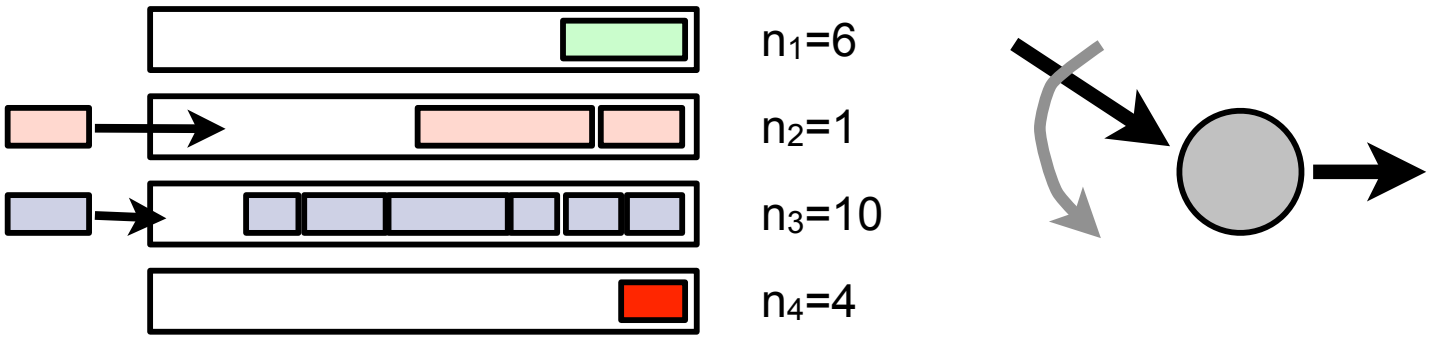
- Se puede asociar un peso $\phi(i)$ a cada cola y entonces la cantidad de servicio es proporcional al mismo
- Ofrece *weighted max-min fairness* y lo llamamos *Generalized Processor Sharing (GPS)*
- En cualquier caso, en la realidad no podemos servir fluidos sino que servimos paquetes así que solo podremos aproximarlo
- Round Robin es una aproximación a PS

$$c_i = C\frac{\phi(i)}{\sum \phi(i)}$$



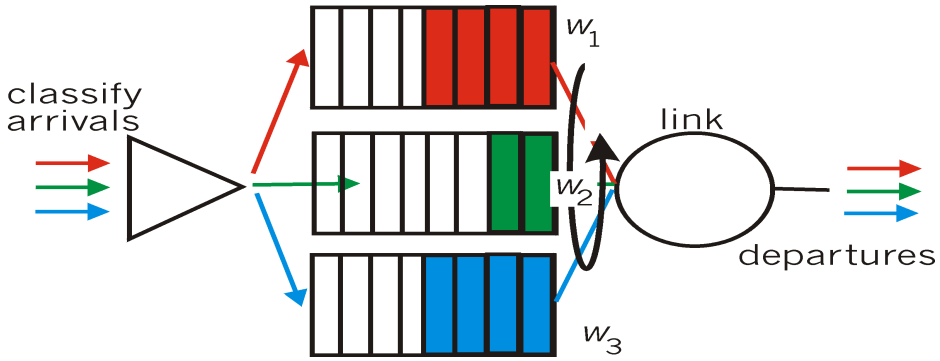
Weighted Round Robin (WRR)

- Opera por “turnos”
- Se normaliza el peso por el tamaño medio de paquete en la clase $\frac{\phi(i)}{s_i}$
- Normaliza el resultado para que sean enteros
- En cada turno visita cada cola (en RR) y sirve tantos paquetes como su peso normalizado
- Ejemplo:
  - Pesos: 0.03, 0.05, 1 y 0.5
  - Tamaños medios: 50, 500, 1000 y 1200 bytes
  - Renormalizados según tamaños medios: 0.0006, 0.0001, 0.001 y 0.0004
  - Renormalizados a enteros: 6, 1, 10, 4



Weighted Round Robin (WRR)

- Necesita saber el tamaño medio de paquete de cada clase
- Más sencillo si los paquetes son de tamaño constante
- Es justo solo por encima de la escala de la duración del turno
- Ejemplo:
  - Enlace a 45Mbps
  - 500 flujos con peso 1 y 500 flujos con peso 10
  - Supongamos que todos los flujos tiene tráfico
  - Todos los paquetes de tamaño constante, 53 bytes
  - Un turno requiere enviar: 500 x 1 + 500 x 10 = 5500 paquetes
  - 5500 paquetes a 45Mbps requieren 51.82ms
  - Por debajo de una escala de 50ms unos flujos reciben más que otros



Deficit Round Robin (DRR)

- Permite hacer un RR con pesos sin conocer tamaños medios de paquetes
- Veamos primero versión **sin pesos**
- Cada clase mantiene un contador de déficit inicializado a 0
- En cada turno se añade **q** (el *quantum*) al contador de cada clase si tiene paquetes por servir, si no se resetea
- El planificador visita cada clase y sirve el primer paquete de la cola si su tamaño es menor que su contador de déficit
- y decrementa el contador en el tamaño del paquete
- Ejemplo:
  - q = 1000 bytes
  - Tres clases A, B y C con paquetes de 1500, 800 y 1200 bytes
  - Turno 1: Clase A contador a 1000, clase B se sirve paquete y el contador se queda en 200, clase C contador a 1000
  - Turno 2: Clase A se sirve paquete y contador a 500, clase B se resetea pues no tiene paquetes (para que no acumule), clase C se sirve paquete y contador a 800

Deficit Round Robin (DRR)

- En la versión con pesos el *quantum* es el peso de cada clase
- El *quantum* debería ser al menos del tamaño máximo de paquete para servir alguno en todos los turnos
- Es sencillo de implementar

WFQ

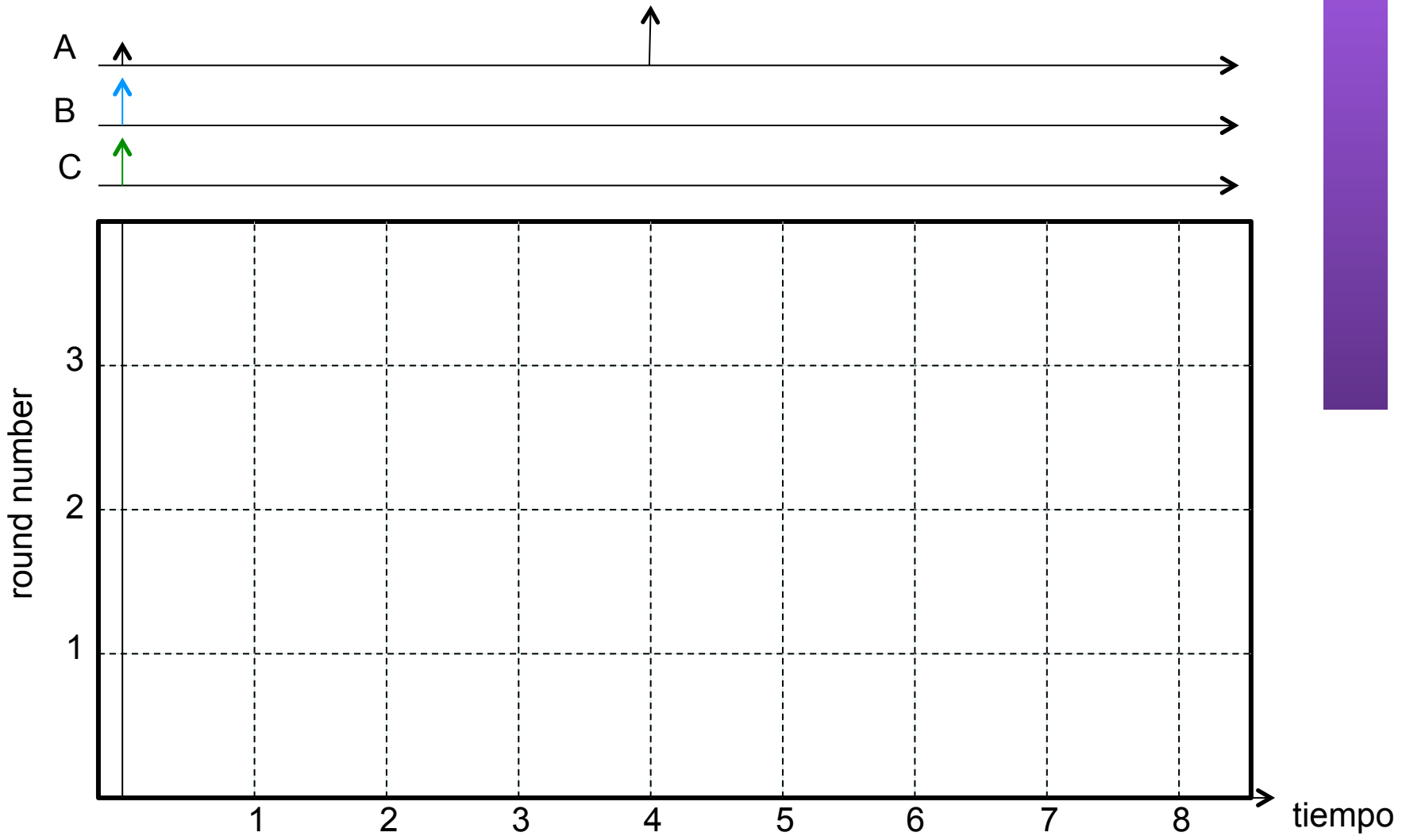
- *Weighted Fair Queueing*
- Aproximación de GPS (*Generalized Packet Sharing*) para el caso de paquetes
- Equivalente a PGPS (*Packet-by-packet Generalized Processor Sharing)*
- No requiere conocer el tamaño medio de paquete
- Emplea un reloj virtual
- Calcula el final virtual en que se enviaría cada paquete en el caso ideal GPS
- Se envían en orden de tiempo final virtual
- Más complejo de implementar
- Puede ofrecer *worst-case bounds*

WFQ

- Se simulan “turnos”
- Supongamos que no hay pesos
- Supongamos que GPS no sirve fluido perfecto sino bit-a-bit
- El número de turno (*round number*) es el número de turnos bit-a-bit que se han completado en un instante
- Cuantos más flujos activos simultáneos hay, más despacio se incrementa el turno con el tiempo pues en un turno hay que enviar un bit de cada uno de ellos
- En realidad podemos ignorar el servir bit-a-bit si definimos el round number como un valor que crece a una velocidad inversamente proporcional al núero de flujos activos
- El *finish number F(i,k,t)* del paquete **k** del flujo **i** que llega en **t** es:
  - Si el flujo está inactivo: el *round number* actual + el tamaño en bits
  - Si el flujo está activo: máx[F(i,k-1,t), round_number] + tamaño
- Si un paquete llega a una cola llena se descartan paquetes en orden decreciente de finish number hasta que quepa
- Una vez calculado el finish number de un paquete no hay que recalcularlo ante nuevas llegadas

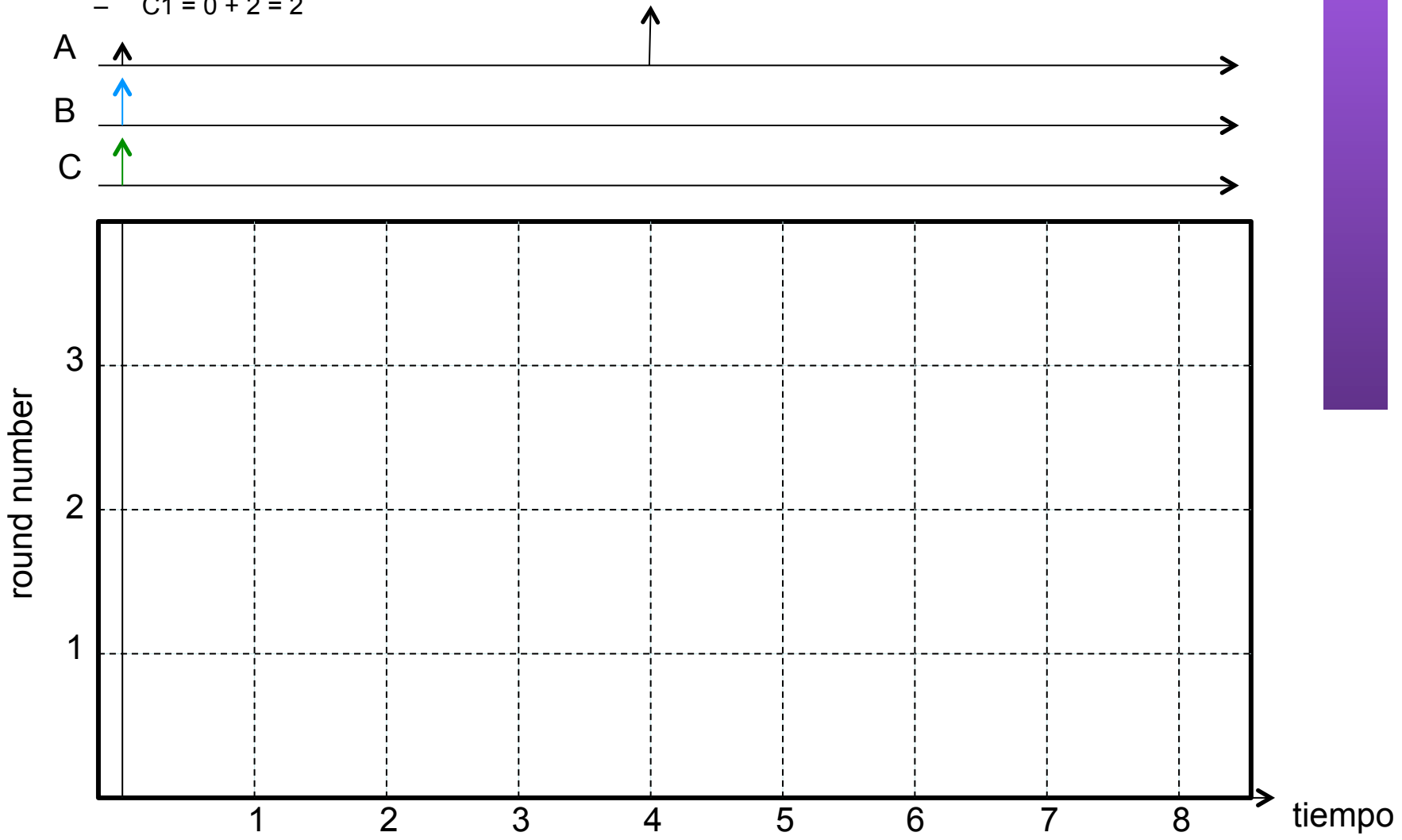
WFQ (Ejemplo)

- Enlace a 1 unidad/s
- Llegadas de tamaños 1, 2 y 2 unidades en t=0 y de tamaño 2 unidades en t=4



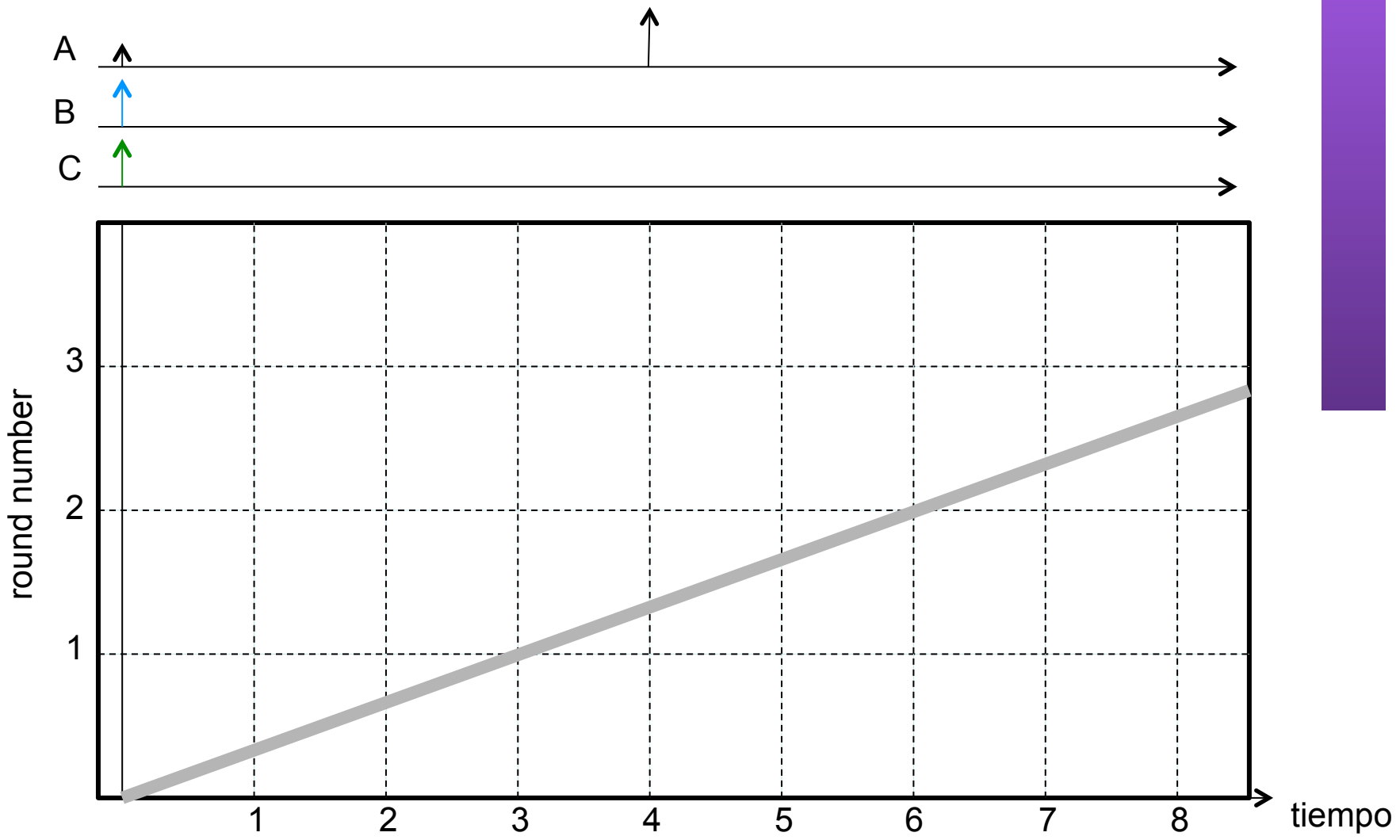
WFQ (Ejemplo)

- Finish numbers:
  - A1 = 0 + 1 = 1
  - B1 = 0 + 2 = 2
  - C1 = 0 + 2 = 2



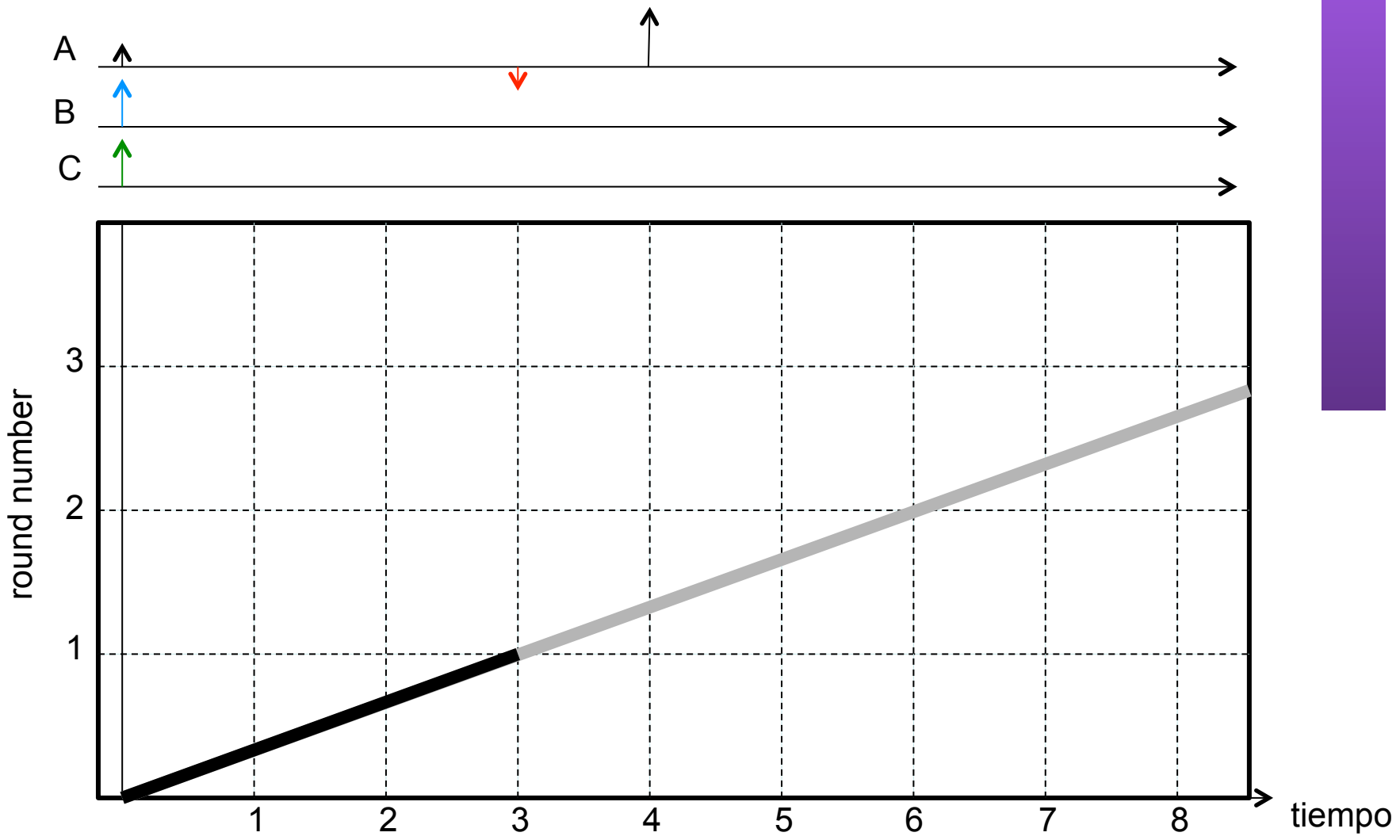
WFQ (Ejemplo)

- Hay 3 flujos a enviar simultáneamente
- El round number se incrementa a C/3 = 1/3 por unidad de tiempo



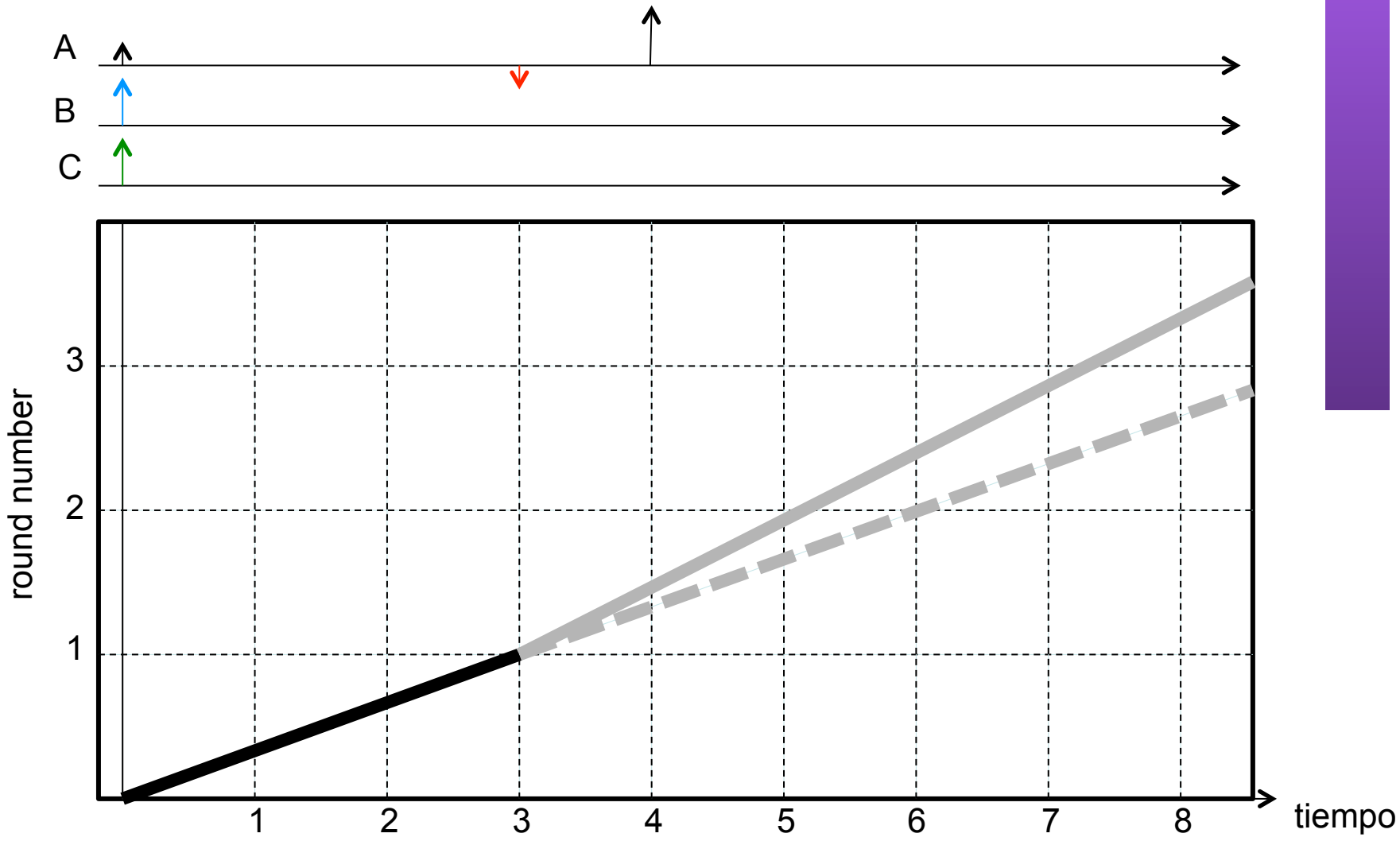
WFQ (Ejemplo)

- En el instante t=3 se han servido 3 bits, eso es uno por flujo y por lo tanto termina el round 1 y termina de enviarse A1



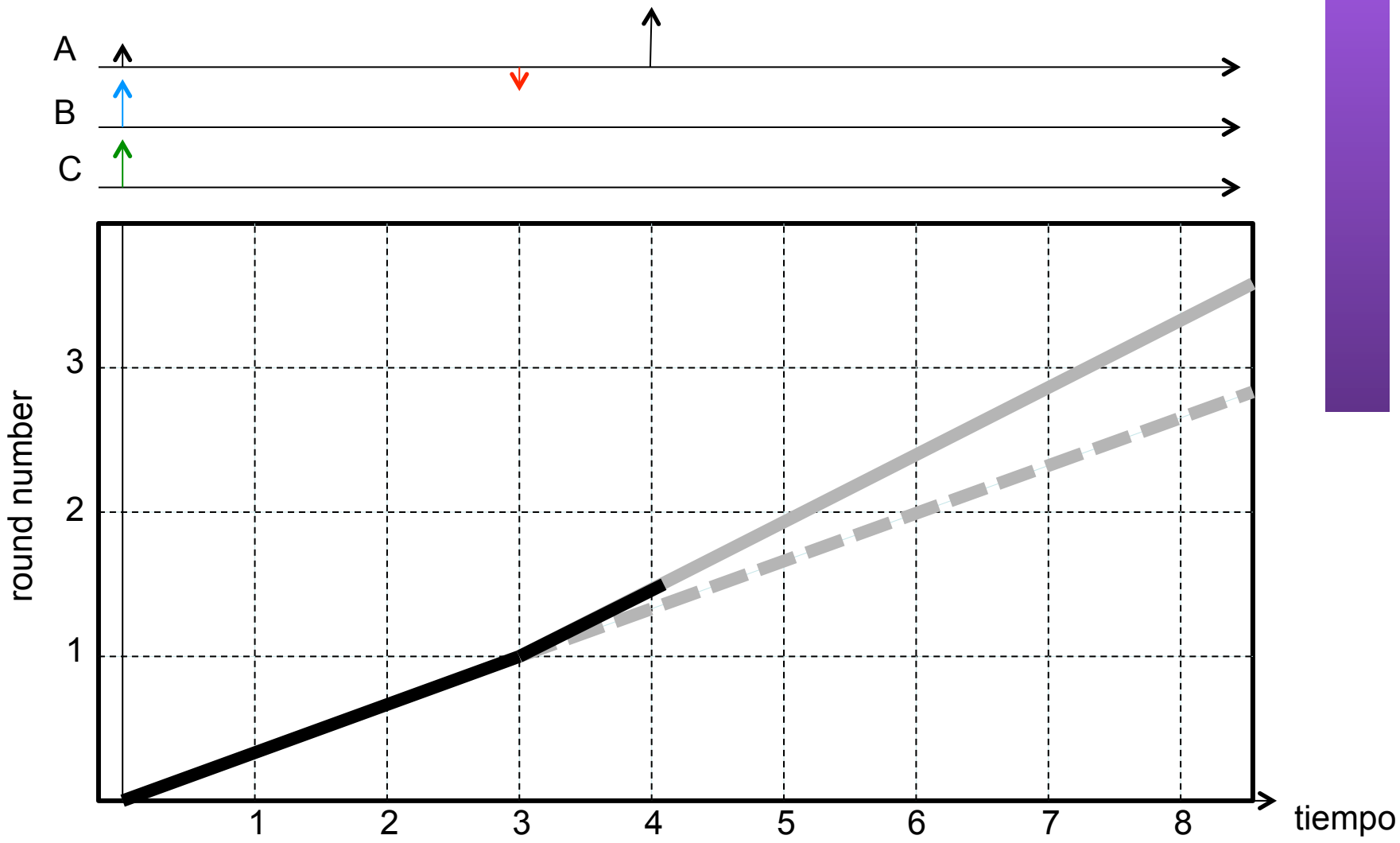
WFQ (Ejemplo)

- A partir de ahí se siguen sirviendo B1 y C1 con finish number = 2
- Al haber dos flujos activos crece el round number a 1/2



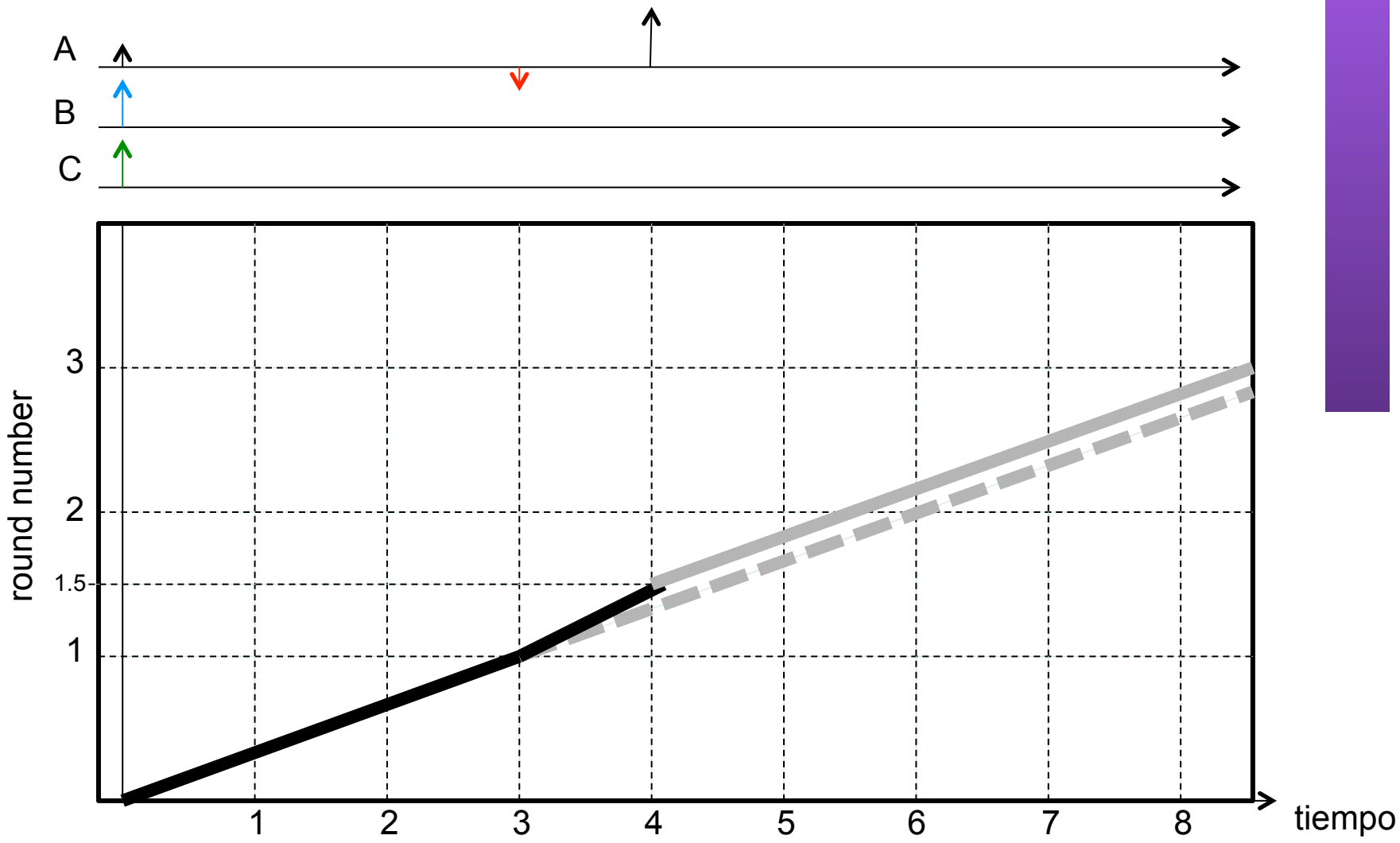
WFQ (Ejemplo)

- B1 y B2 terminarían de enviarse al alcanzar round number = 2 (t = 5) pero llega antes A2



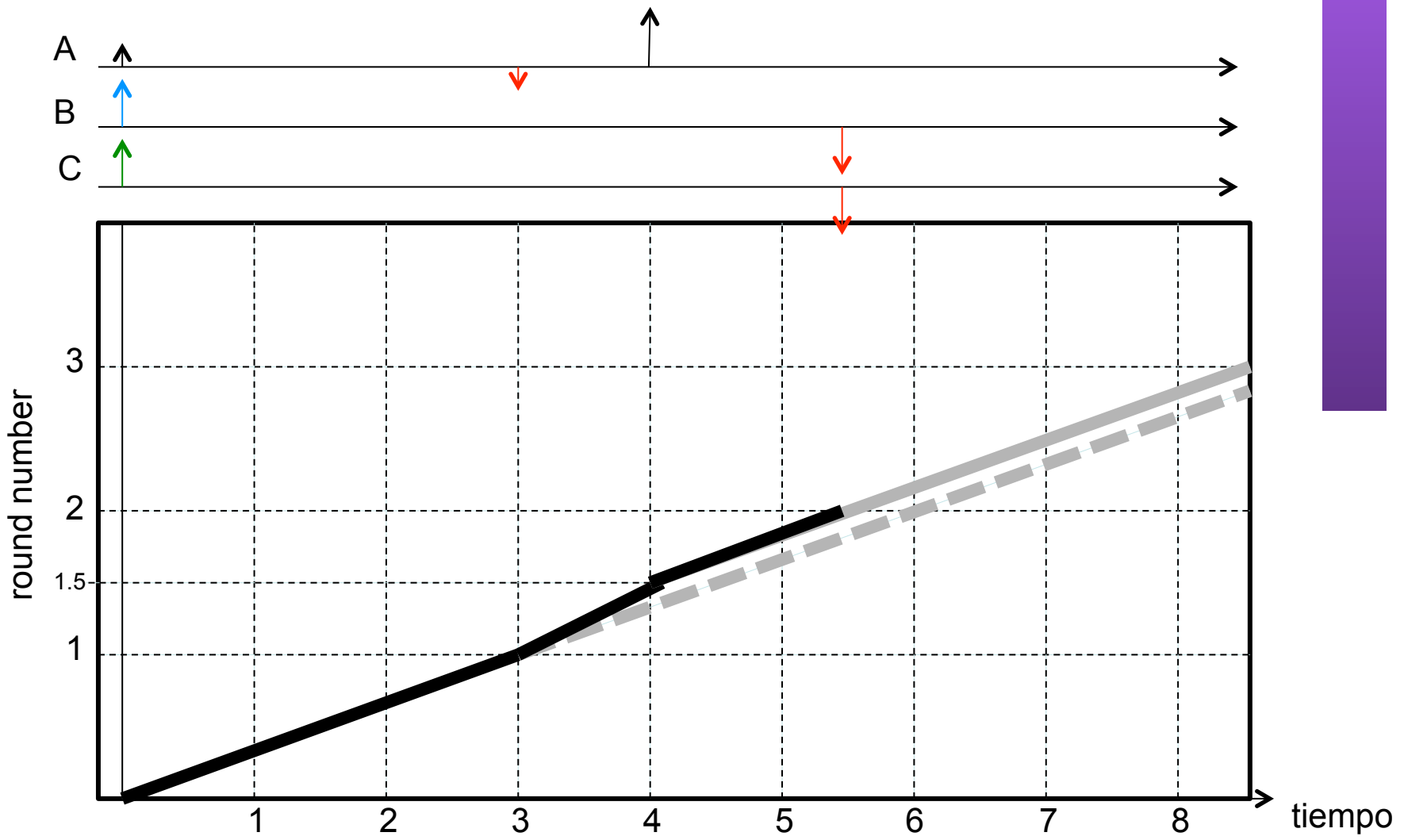
WFQ (Ejemplo)

- Finish number de A2 es 1.5 + 2 = 3.5
- A partir de t=4 vuelve a haber 3 flujos simultáneos



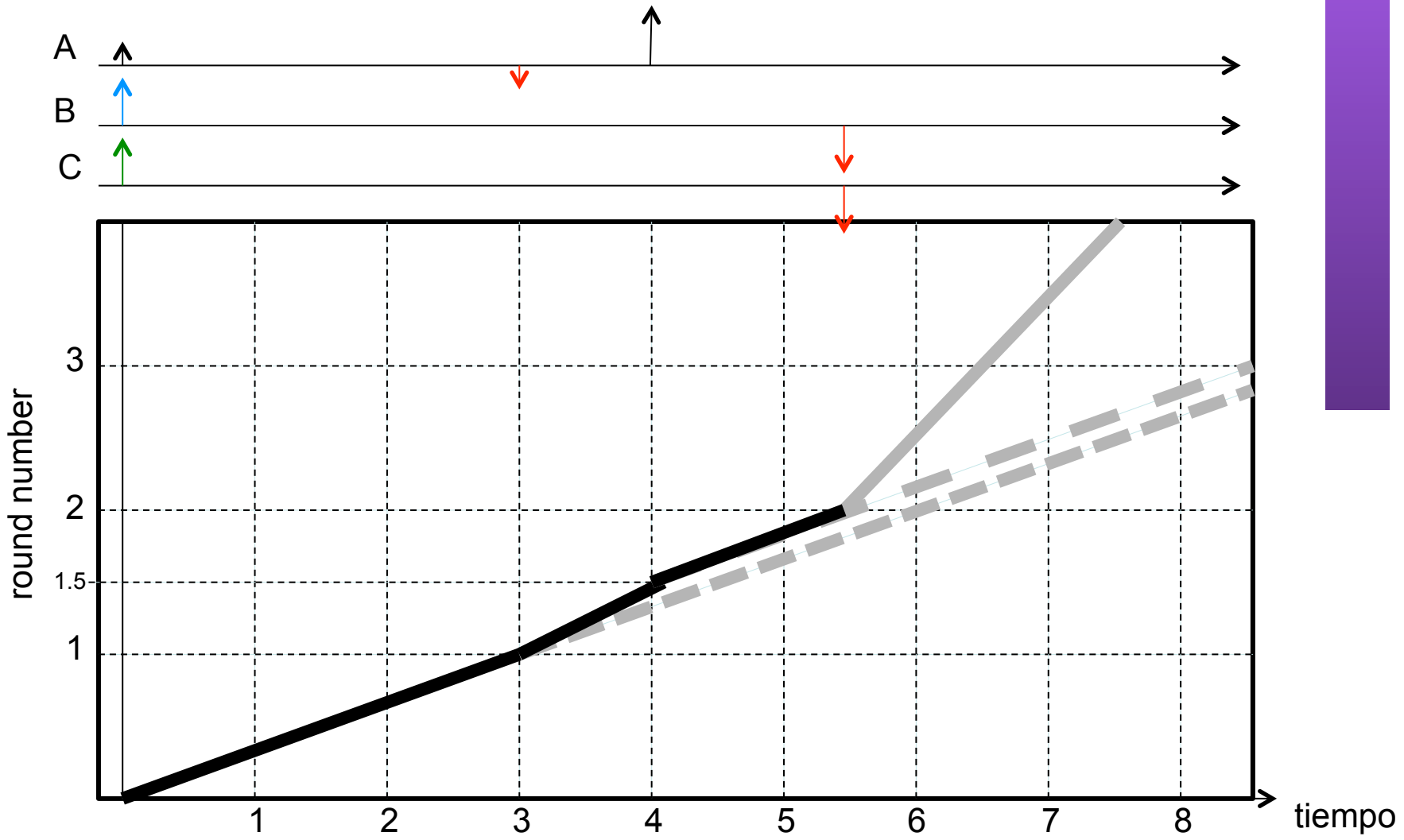
WFQ (Ejemplo)

- Se alcanza el round number 2 en t = 4 + 0.5/(1/3) = 5.5
- Entonces se completaría el envío GPS de B1 y C1



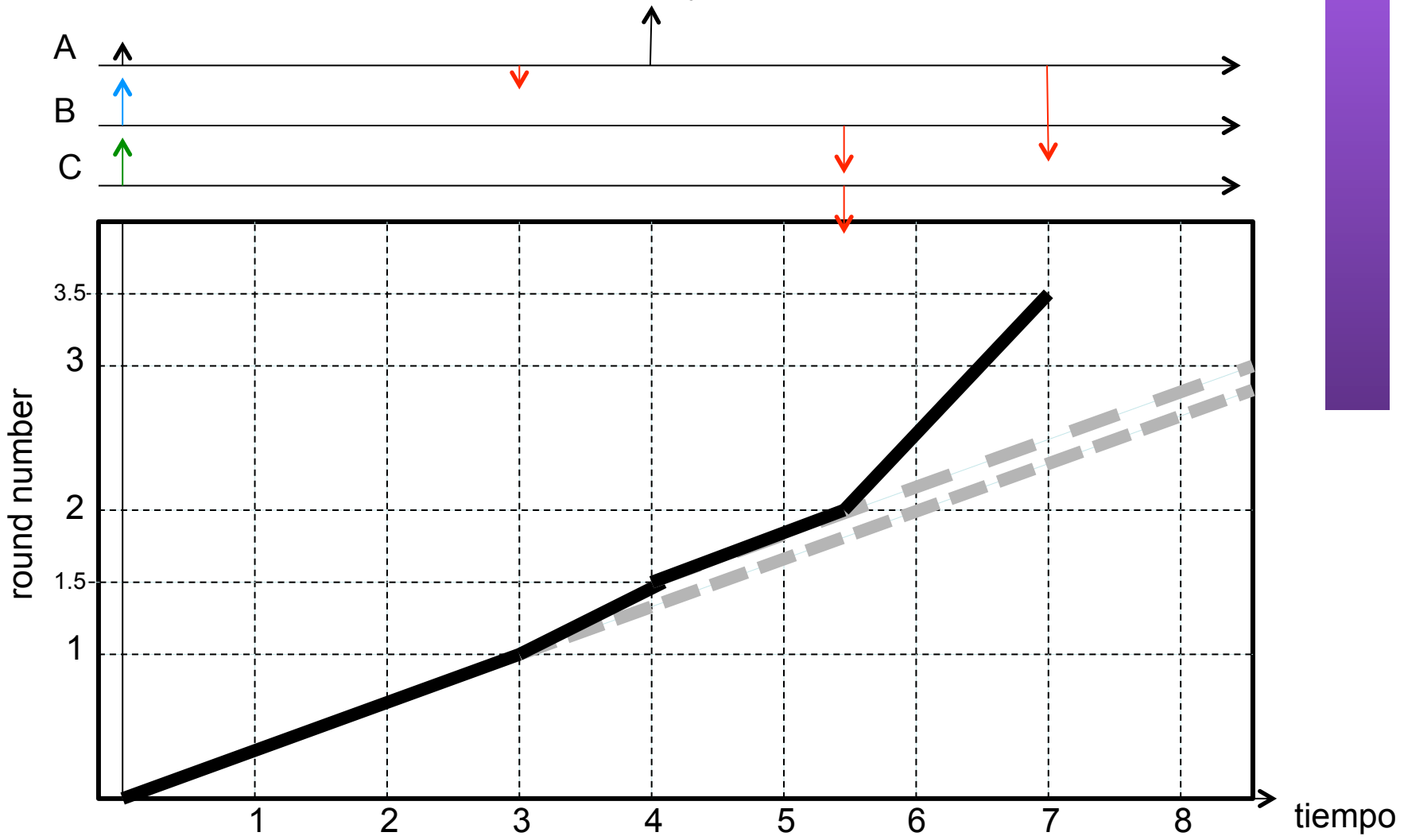
WFQ (Ejemplo)

- Queda solo una fuente activa luego ahora se avanza a 1 round por unidad de tiempo



WFQ (Ejemplo)

- Queda solo una fuente activa luego ahora se avanza a 1 round por unidad de tiempo
- En t = 7 se alcanza el round number 3.5 y termina de enviarse A2



WFQ

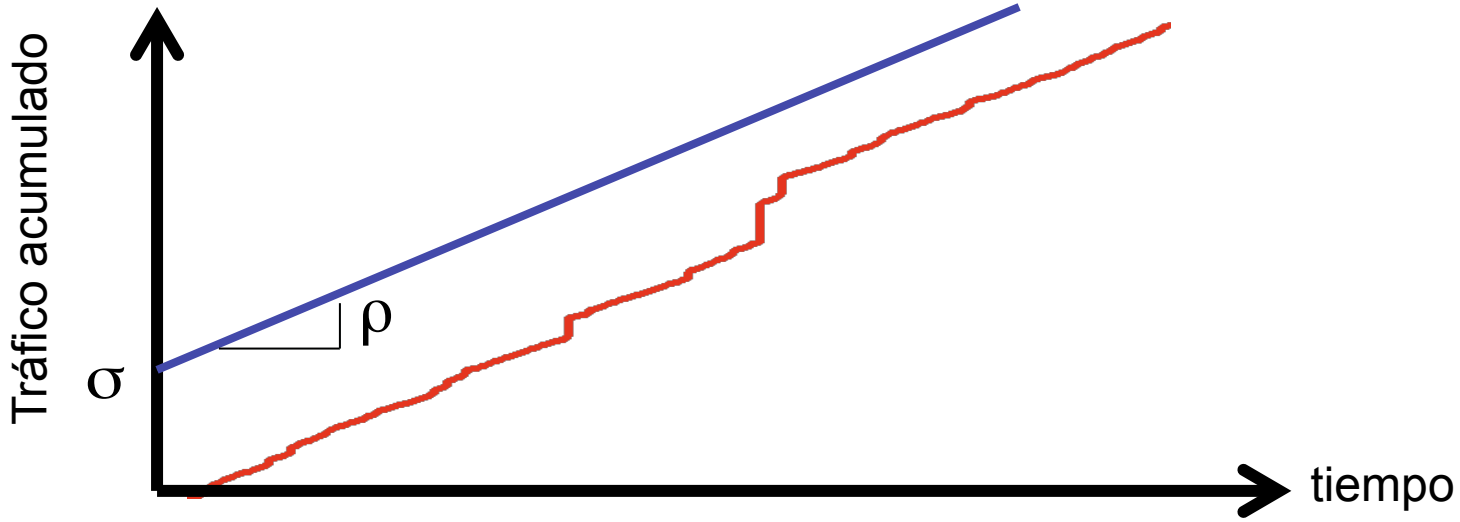
- Calcular el round number es complejo
- Hay que hacerlo para cada paquete que llega y por cada uno que se envía
- En el caso con pesos a la hora de calcular el finish number:
  - Si flujo inactivo: el *round number* actual + tamaño / peso
  - Si flujo activo: máx[F(i,k-1,t), round_number] + tamaño / peso
- y el round number se incrementa con el inverso de la suma de los pesos
- Existen variantes para simplificar este cálculo:
  - Self-Clocked Fair Queuing (SCFQ)
  - Start-Time Fair Queuing

Cotas (*bounds*) en WFQ

- WFQ garantiza reparto weighted max-min fair
- Eso quiere decir que cada flujo recibe una asignación proporcional a su peso

$$c_i = C\frac{\phi(i)}{\sum \phi(i)}$$

- Además pone una cota al retardo máximo
- Supongamos un flujo con una restricción "sigma-ro" (σ, ρ) :
  - En un intervalo t llegan como mucho $\sigma + \rho t$ bits
  - Es la salida de un *token bucket*
  - También llamado *Linear Bounded Arrival Process* (LBAP)



Cotas (*bounds*) en WFQ

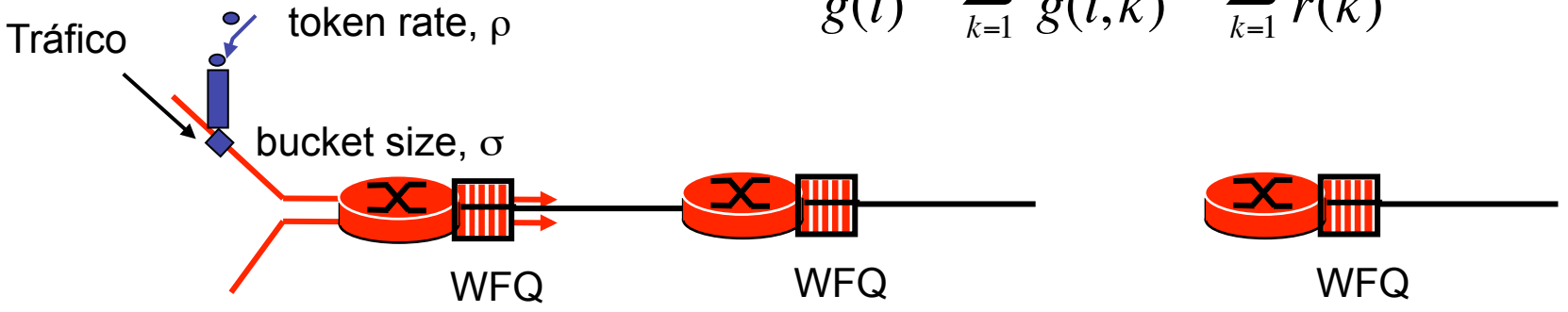
- Un flujo *i* con restricción (σ(i), ρ(i)) (el resto puede no estar conformado)
- Camino con *K* saltos (todos WFQ)
- Se le ha asignado una tasa g(i,k) en cada uno:

$$g(i,k) = r(k)\frac{\phi(i,k)}{\sum_j \phi(j,k)}$$

r(k) link rate en enlace k

- g(i) es el mínimo de g(i,k) y g(i) ≥ ρ(i)
- $P_{max}(i)$ es el mayor tamaño de paquete del flujo *i* y P_{max} en la red
- Entonces el retardo extremo a extremo debido a encolado y transmisión en el peor caso es:

$$D^*(i) \le \frac{\sigma(i)}{g(i)} + \sum_{k=1}^{K-1}\frac{P_{\max}(i)}{g(i,k)} + \sum_{k=1}^{K}\frac{P_{\max}}{r(k)}$$



Cotas (*bounds*) en WFQ

- Podemos garantizar un retardo máximo extremo a extremo
- Planificadores WFQ en todo el camino
- Requiere que el flujo esté conformado por un leaky bucket
- No se impone restricciones al resto de flujos en la red
- Solo hay que seleccionar los valores adecuados de reserva de BW en los enlaces
- **Ejemplo**

$$D^*(i) \le \frac{\sigma(i)}{g(i)} + \sum_{k=1}^{K-1} \frac{P_{\max}(i)}{g(i,k)} + \sum_{k=1}^{K} \frac{P_{\max}}{r(k)}$$

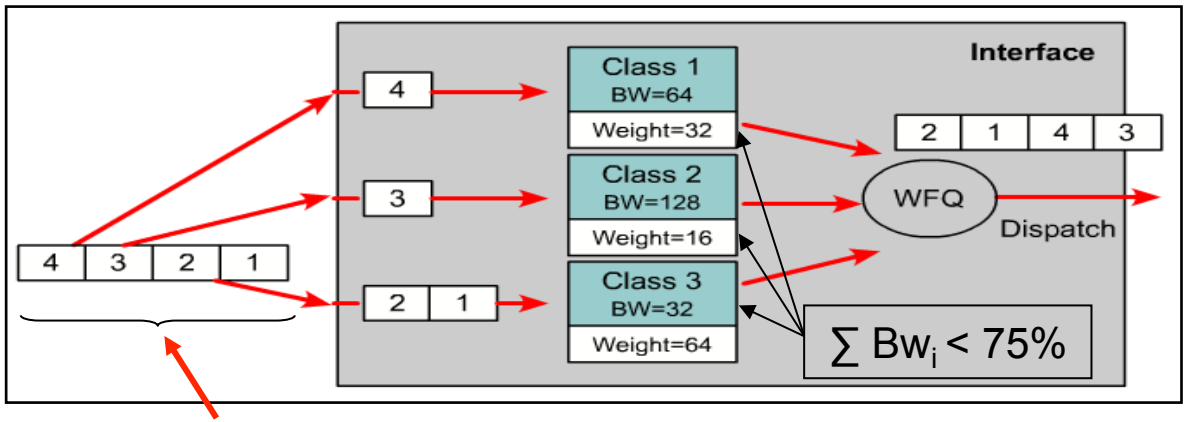
  - Flujo LBAP con parámetros (16 KBytes, 150 Kbps)
  - K = 10 saltos, todos a 45 Mbps, retardo de propagación total de 30ms
  - Máximo tamaño de paquete de 8 KBytes
  - Queremos un retardo extremo-a-extremo máximo de 100 ms
  - Entonces retardo máximo de 100 - 30 = 70 ms

$$\frac{16*1024*8}{g} + 9*\frac{8*1024*8}{g} + 10*\frac{8*1024*8}{45*10^6} \le 70*10^{-3}$$

  - g ≥ 13.004 Mbps !!
  - El segundo término contribuye en 45.4 ms y el tercero en 14.5 ms
  - El término σ/ρ solo contribuye en torno a 10.07 ms
  - Al haber paquetes grandes tienen un gran efecto en el retardo de caso peor

Típica implementación de WFQ

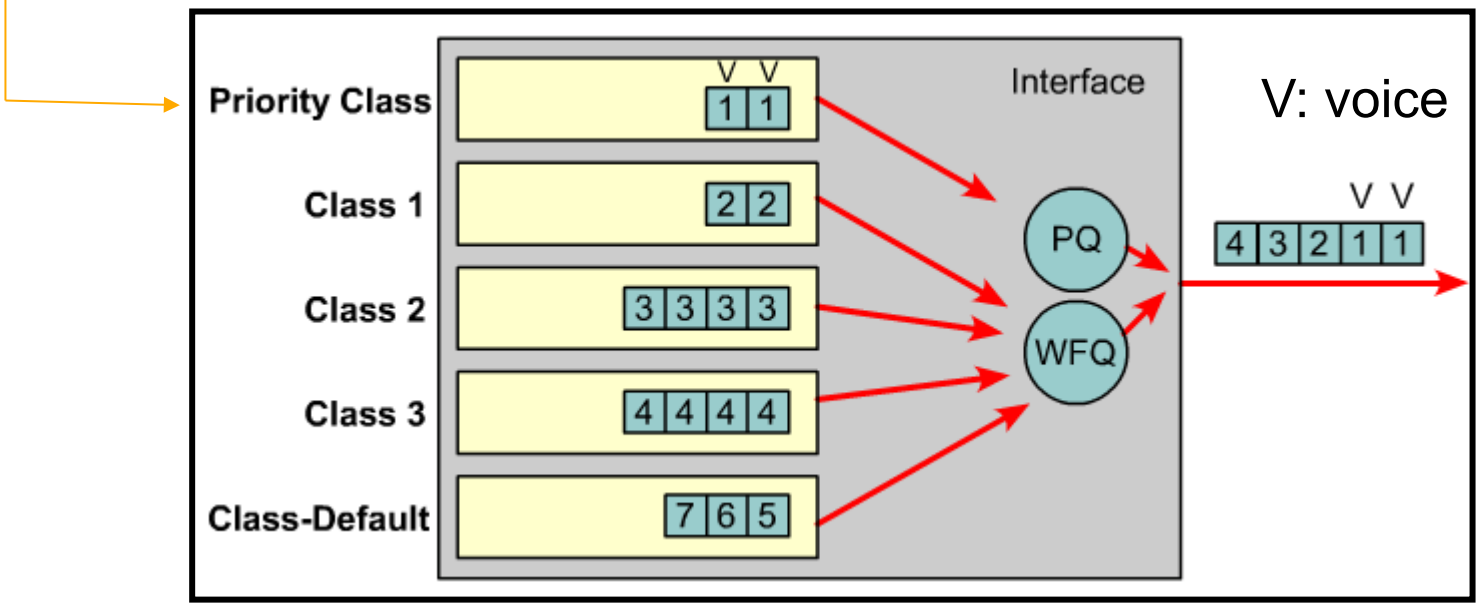
- Cada flujo es una "conversación" reconocida por info. de layer 3 (direcciones IP, precedencia) y de layer 4 (puertos)
- Pesos en función de los bits de precedencia de los paquetes
- No requiere configuración
- No escala (una cola por conversación)
- CBWFQ
  - *Class Based WFQ*
  - Especificar los filtros (clases) que determinan los paquetes que van a cada cola (una por clase, no por flujo)
  - Especificar peso para cada cola



Low Latency Queueing (LLQ)

- Añade una PQ (*Priority Queue*) a CBWFQ = PQ-CBWFQ = LLQ
- Recomendable para tráfico multimedia (VoIP): bajo retardo y jitter.
- Se puede configurar junto al resto de colas CBWFQ como una cola más asociada a una clase determinada.

LLQ se comporta como una *Priority Queue.*



Resumen

- Priority Queueing
- Round Robin
- Weighted Round Robin
- Weighted Fair Queueing
- Worst case bounds