IBM Research Zurich Lab

Hacia un middleware de alto rendimiento más predecible

Luis Garcés-Erice

Advanced Messaging Technologies

Ideas principales

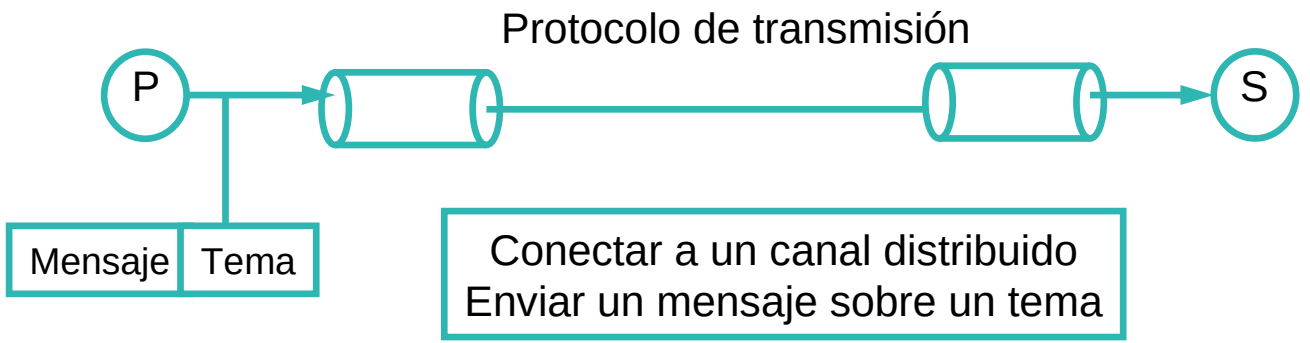
- Los sistemas de tiempo real tradicionales necesitan mucho control: actividades conocidas, planificador analítico
  - Control de admisión puede implementarse analíticamente
- Los complejos sistemas de software actuales tienen cada vez más necesidades de “tiempo real”
  - Proponemos un middleware que asigna recursos a actividades
  - Si modelamos el middleware podemos proveer control de admisión
- La complejidad de estos sistemas conlleva modelos que dependen de parámetros de tiempo de ejecución:
  - Imposibles de resolver de manera analítica
  - El sistema necesita ser calibrado para “aprender” estas dependencias

Agenda

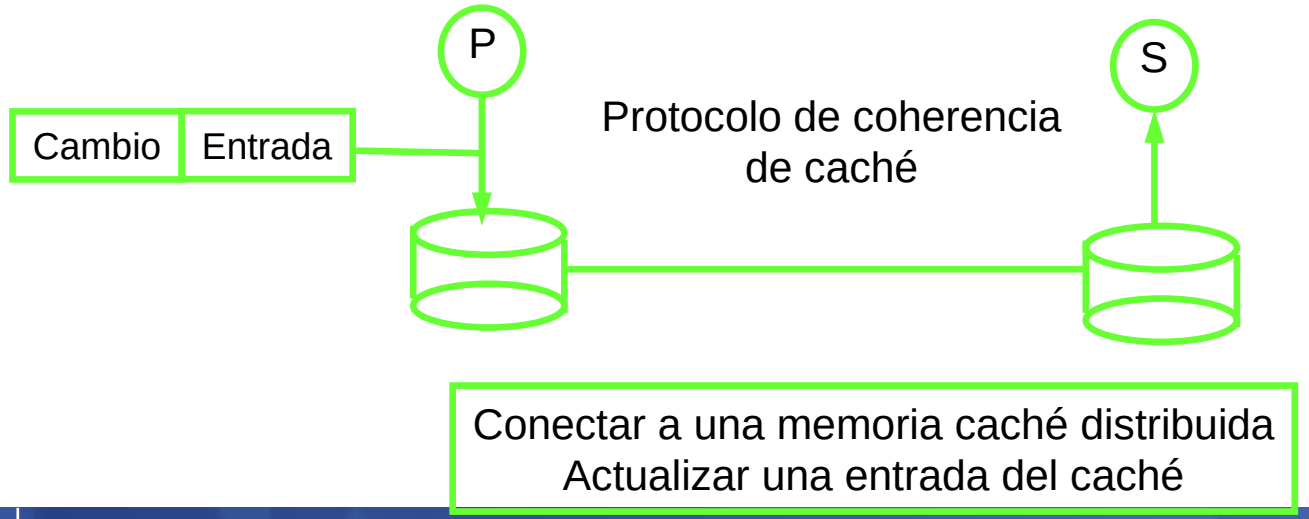
- Minúscula introducción al middleware pub/sub
- Arquitectura de Tempo: Planificador (scheduler)
- Análisis de rendimiento
- Predecir el comportamiento del middleware
- Resultados

Publish/Subscriber Middleware

Orientados a menajes (e.g. WS-Notification, CORBA)

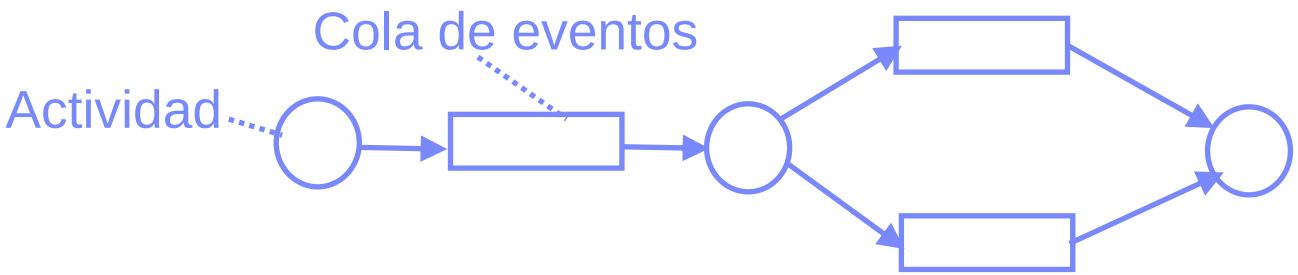


Orientados a datos (e.g Tuplespace, DDS)



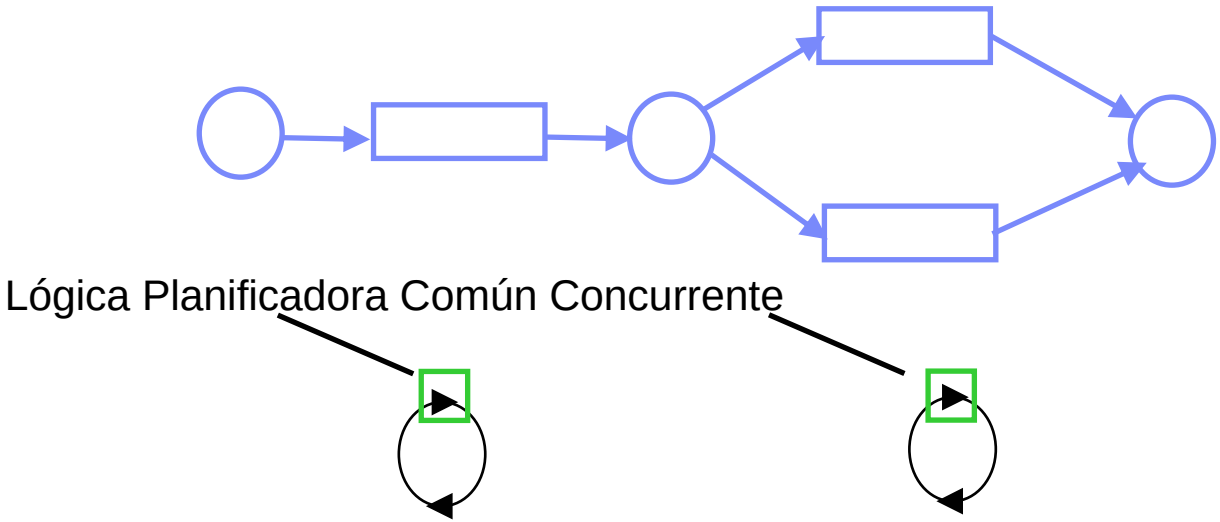
Motivación

El uso de un modelo asíncrono de paso de mensajes favorece la concurrencia



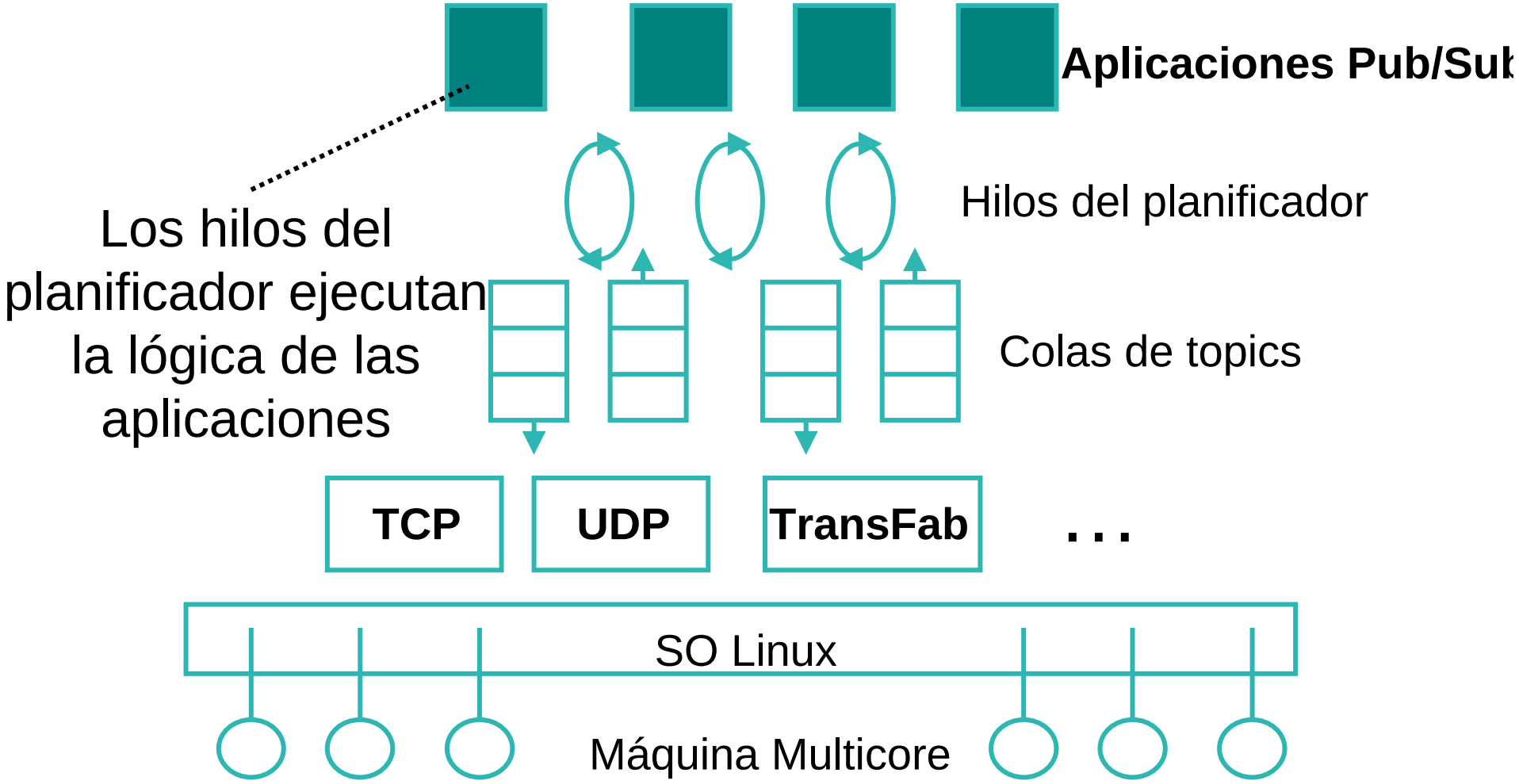
Motivación

Vamos a construir un planificador de eventos que ordena las actividades usando información sobre la importancia de las colas que sirven



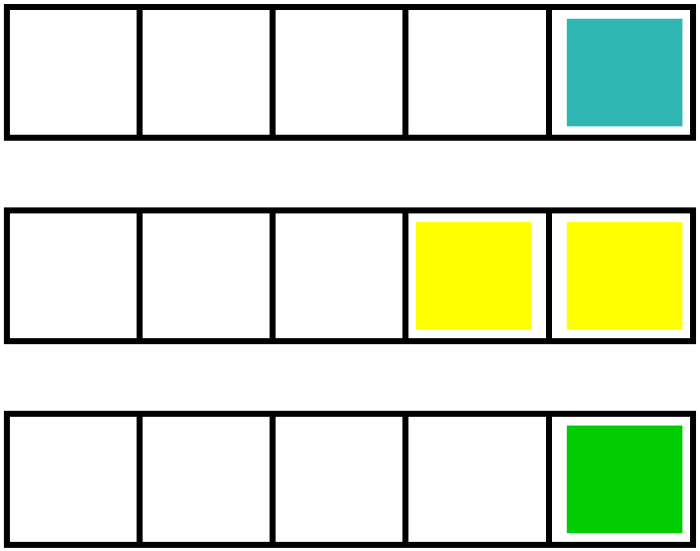
¡ No son hilos del planificador del SO !

Arquitectura de Tempo



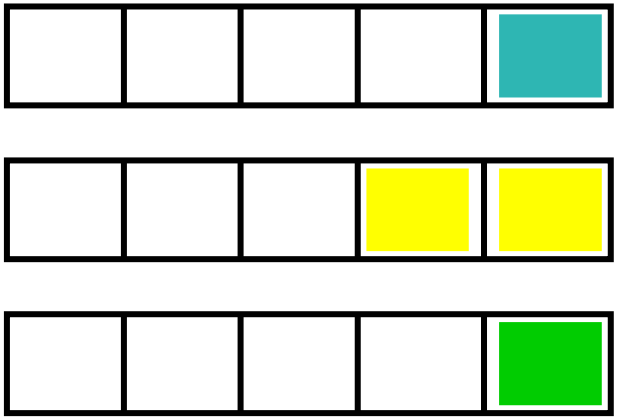
Cola de eventos de un tipo determinado (Topic)

El tipo lo determina la actividad, el proceso es FIFO, longitud limitada



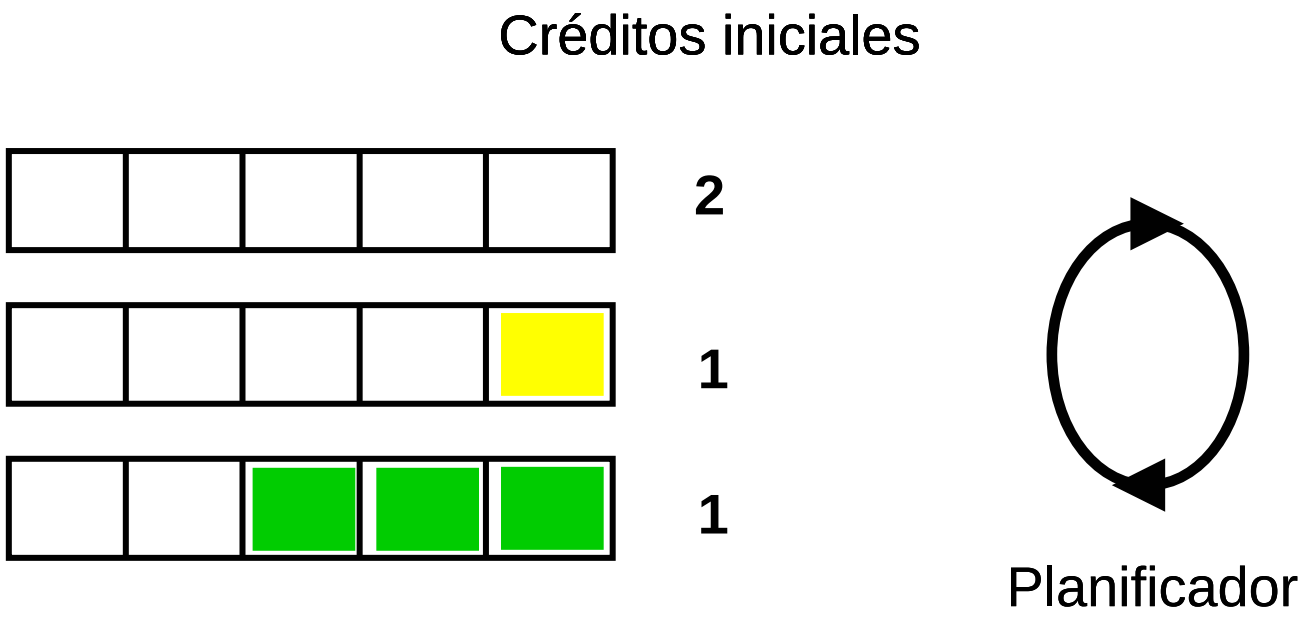
WFS implementado con créditos

Los créditos se asignan a cada cola, el planificador sirve la cola no-vacía con más créditos hasta que todas las colas con algún evento no tienen más créditos. Entonces los créditos se restauran

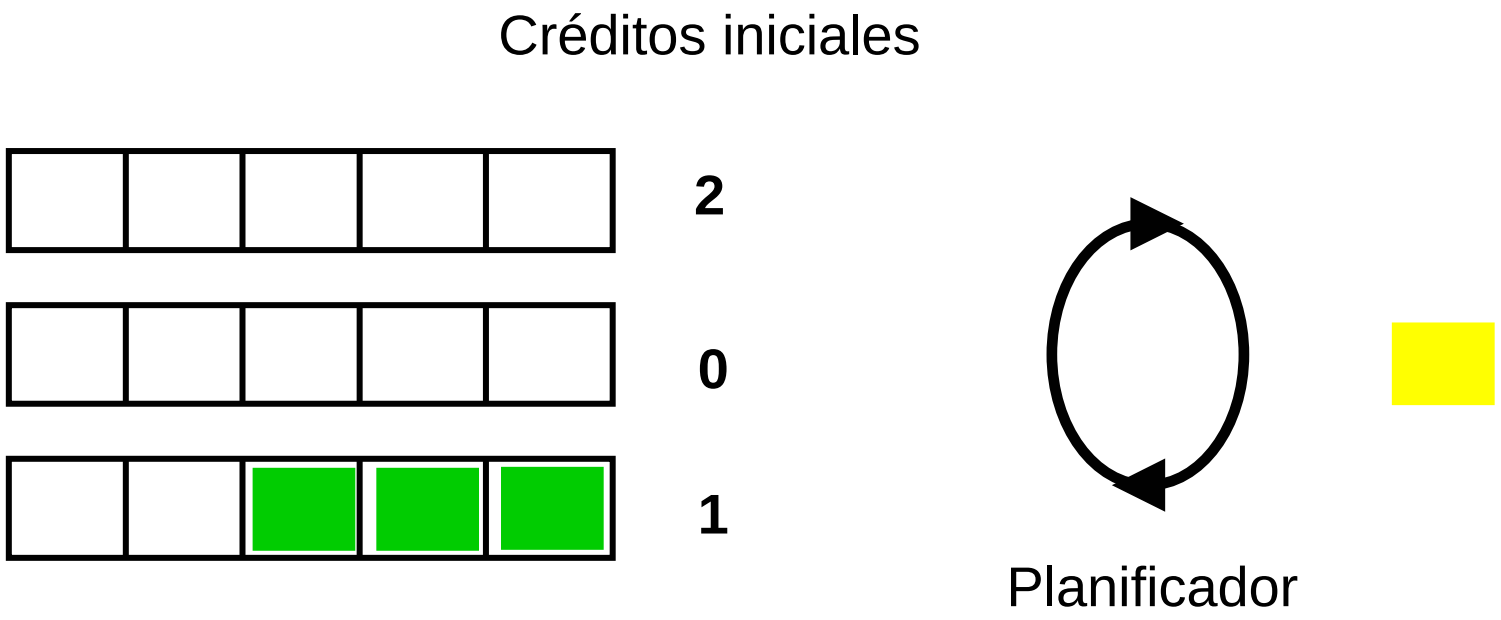


¡Work conserving!

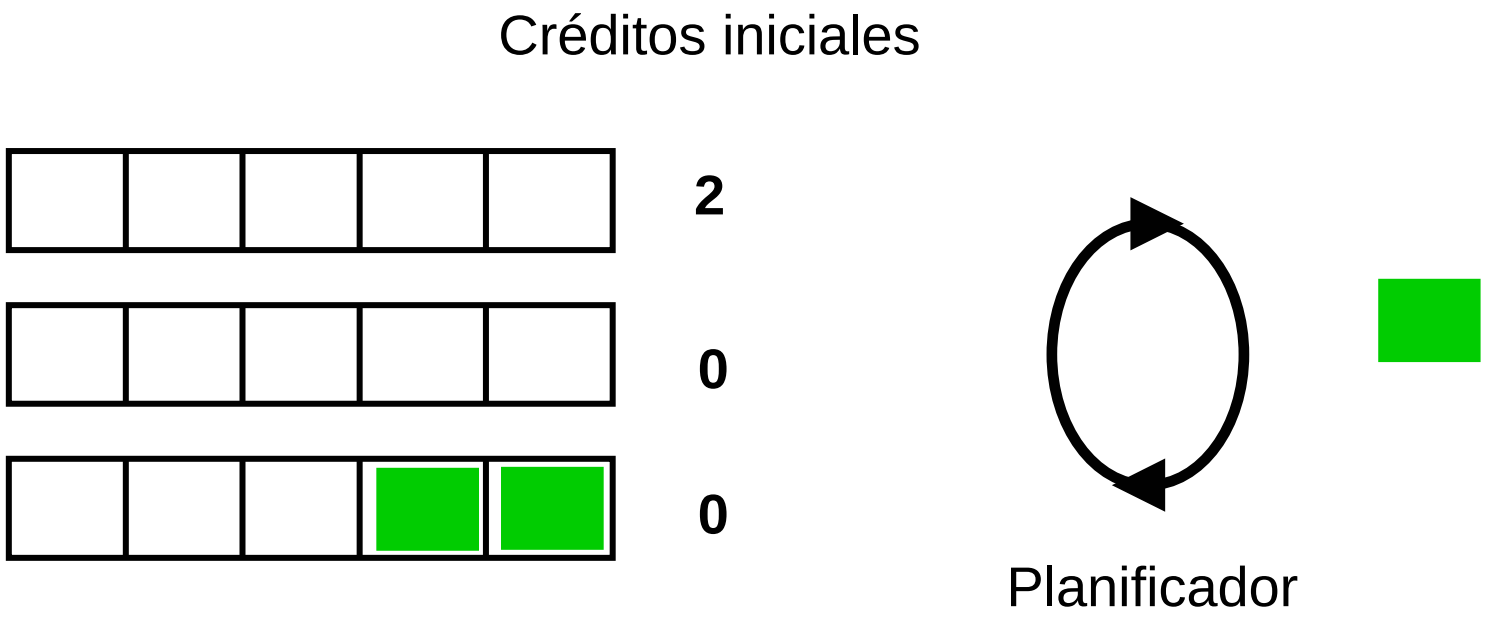
Planificador por créditos



Planificador por créditos



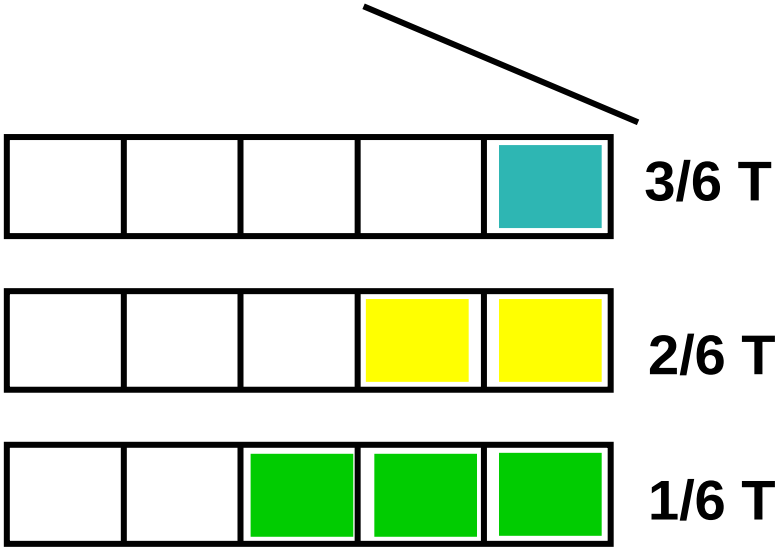
Planificador por créditos



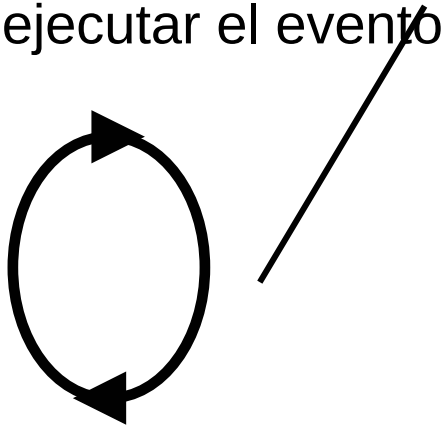
Ninguna cola con eventos tiene créditos → restaurar créditos …

Planificador de créditos de tiempo

Los créditos son una fracción de un periodo T



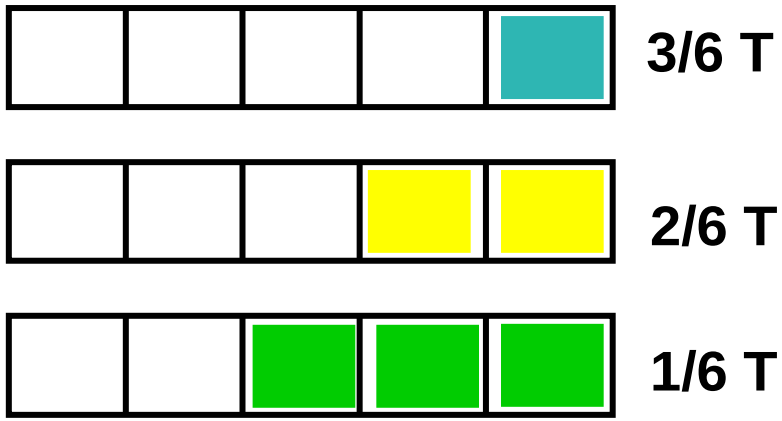
El planificador reduce el crédito según el tiempo empleado en ejecutar el evento

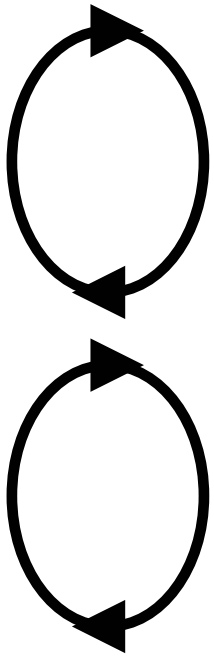


Si es *conforme*, una cola no puede estar desatendida más de T
Si es *estable*, un evento no puede permanecer en una cola más de T
La disconformidad o la inestabilidad se pueden detectar con facilidad

Planificador de créditos de tiempo multiprocessador

Los hilos del planificador deben respetar el plan con el mínimo de interferencia





Usar un método Lock-free para mantener el estado del planificador
Las colas con eventos son puestas en una Cola de Prioridad Lock-free

Código no bloqueante

Posible con instrucciones avanzadas del procesador
e.g. Compare and Swap

CAS x y z

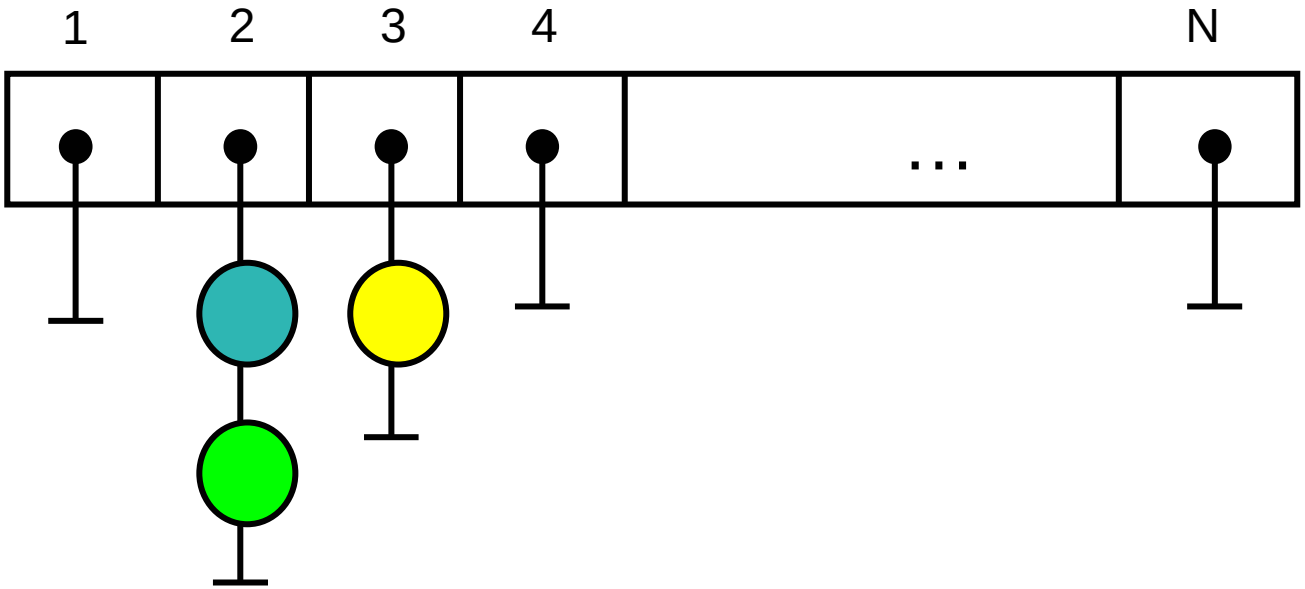
If x equals y
Then assign z to x
Else return false

```
AtomicIncrement(x)
  do {
    int y := x;
  } until ( CAS(x,y,y+1));
```

Nuevos algoritmos construidos sobre este elemento,
e.g. cola Michael-Scott FIFO

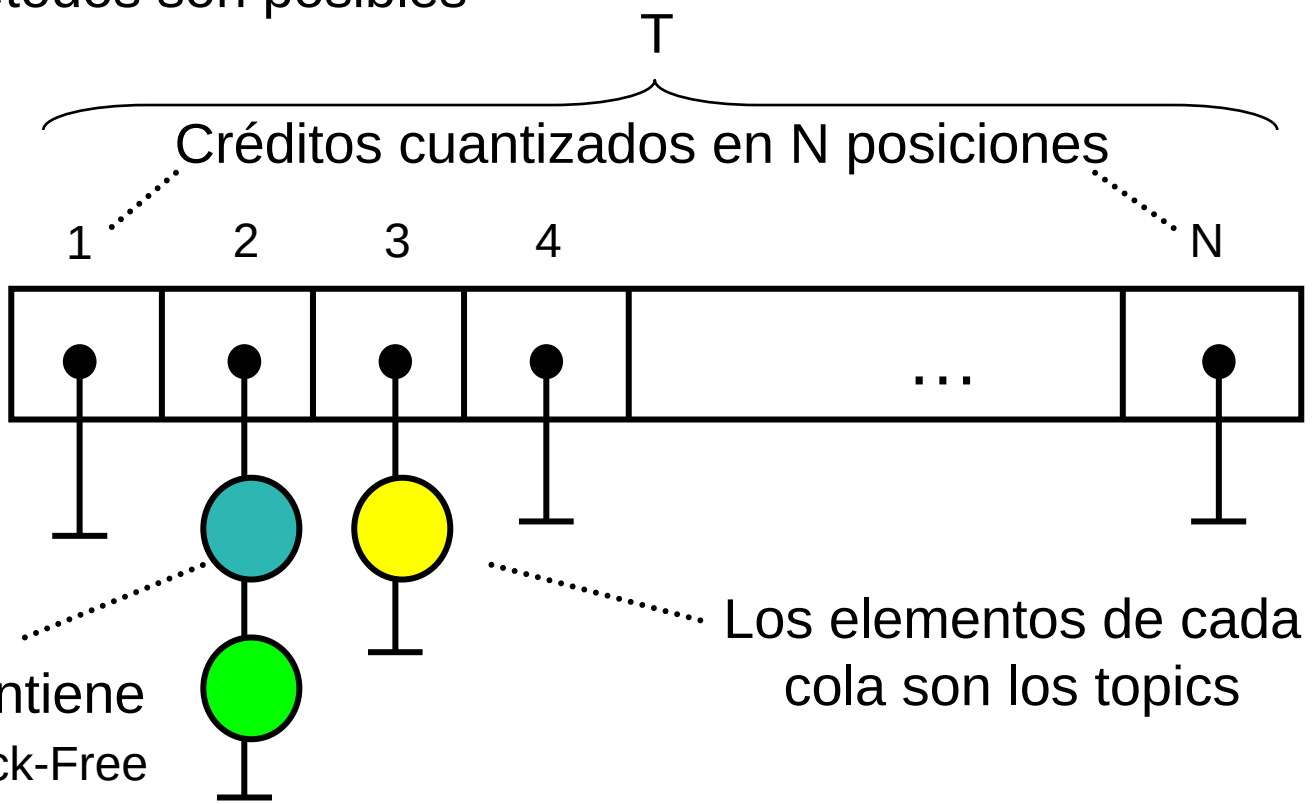
Cola de prioridad cuantizada lineal concurrente

El hilo del planificador “GET” el primer elemento (cola de topic) en una posición no vacía, procesa la cola asociada, actualiza los créditos y lo “PUT” en la posición correspondiente



Cola de prioridad cuantizada lineal concurrente

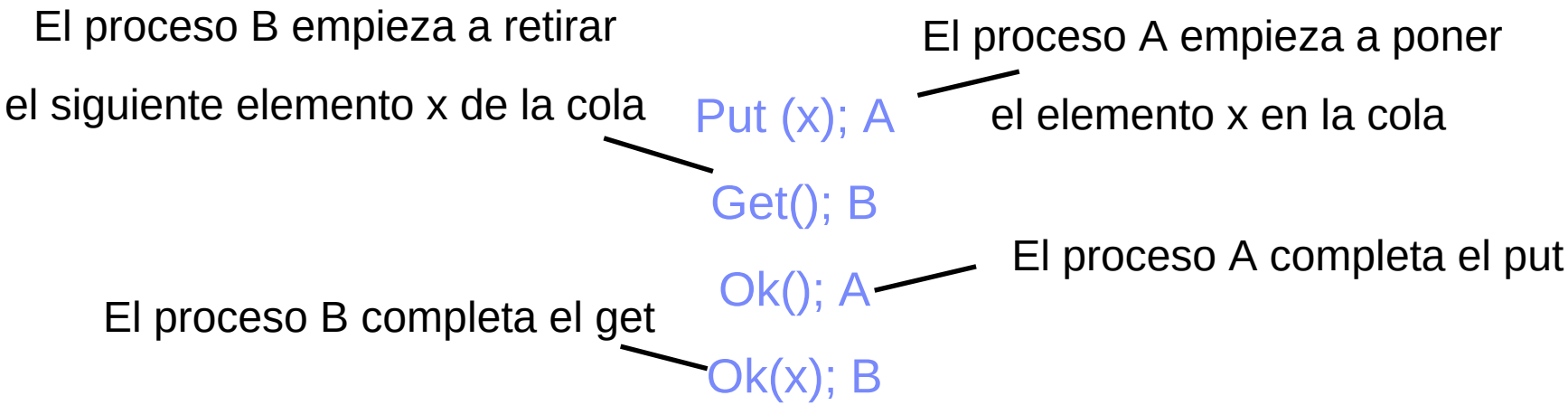
Diferentes métodos son posibles



¿Qué garantiza esta cola concurrente ?

Un invariante es una afirmación sobre todas las historias posibles de la cola

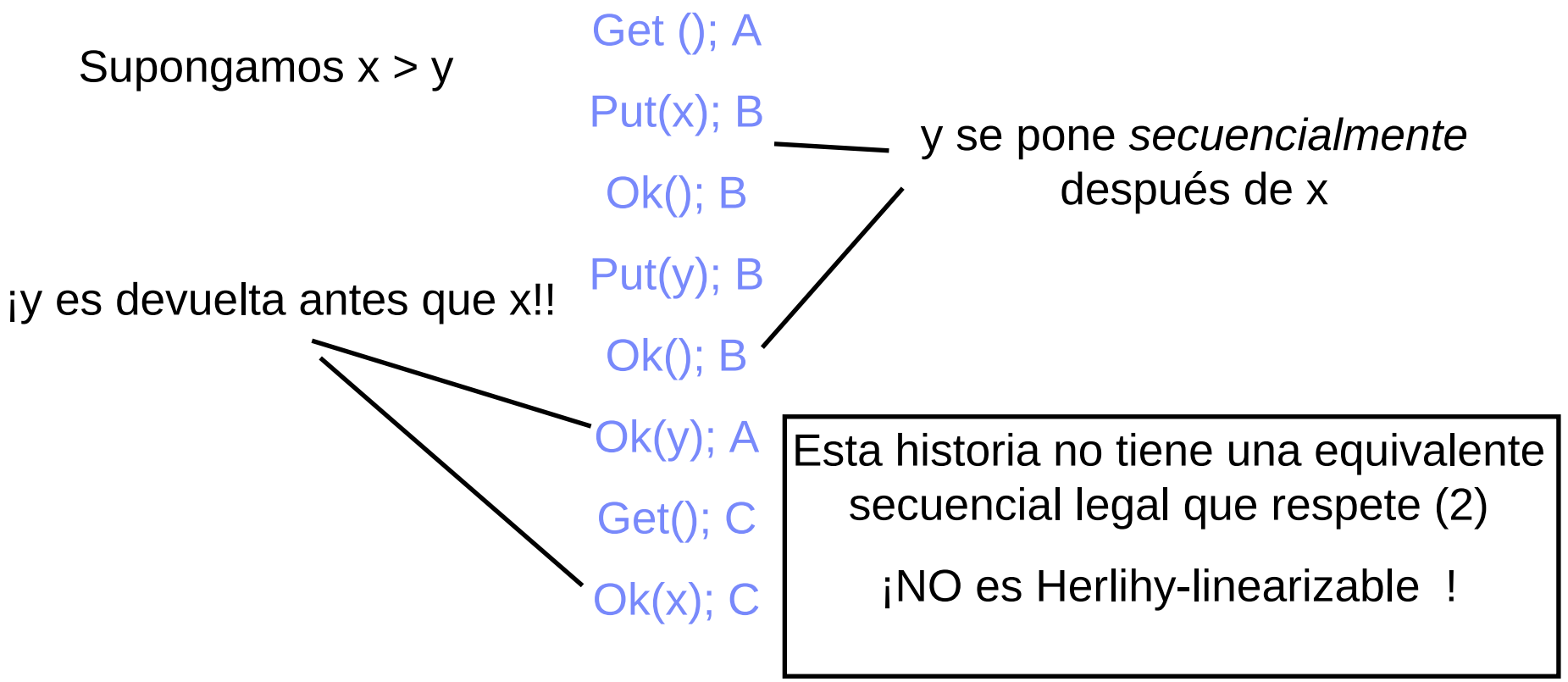
Una historia:



Condición de linearizabilidad Herlihy

(1) Todas las historias posibles de la estructura de datos tienen una secuencia legal equivalente

(2) El orden de las operaciones ejecutadas secuencialmente se respeta en esa historia



Cola de prioridad cuantizada lineal concurrente

Definimos

$$PUT ::= [e : Element, start : Time, end : Time]$$

$$GET ::= [p : PUT, start : Time, end : Time]$$

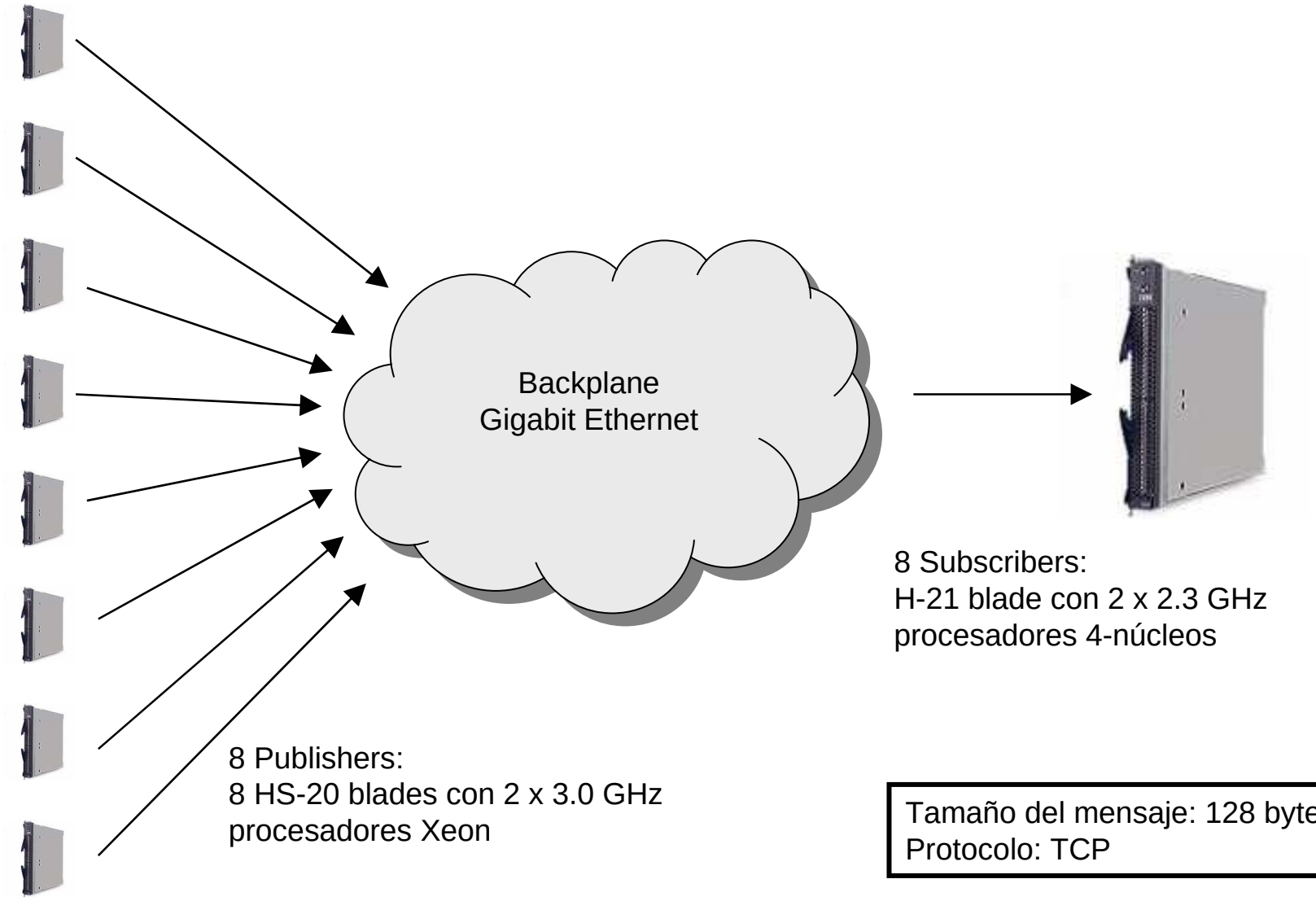
$$H \ : \ SEQ \ of \ GET, \forall i < j \ H[i].end \leq H[j].end$$

El invariante es

$$H[i].e.priority < H[j].e.priority => H[i].start < H[j].end$$

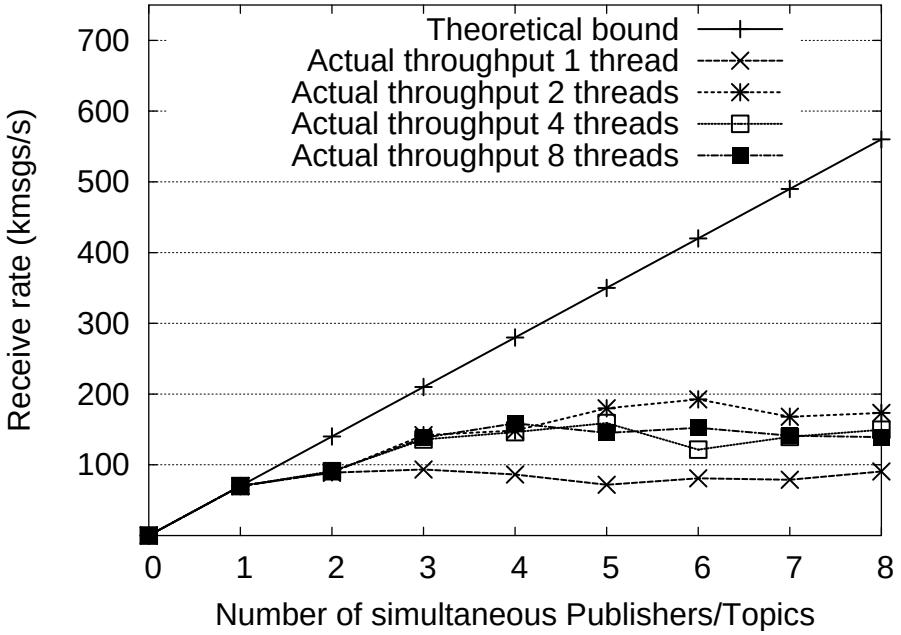
Esto es suficiente para el planificador de eventos …

Rendimiento: detalles del experimento

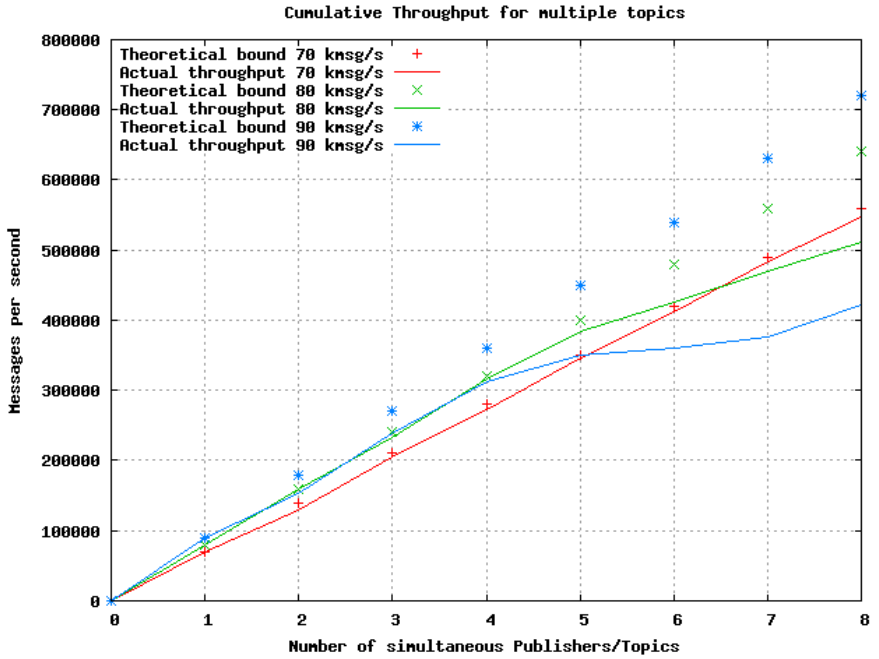


Escalabilidad con el número de procesadores

Cola secuencial (lock)



Cola cuantizada



Predecibilidad

- Necesidad de garantizar QoS: throughput y latencia
- Podemos asignar fracciones de la CPU a actividades
  - Supongamos que la latencia es “buena” si el throughput es el requerido
  - ¿Cómo hacemos corresponder una fracción de CPU a un throughput?
- ¿Cuán difícil es esto en un sistema complejo que funciona en una JVM en un SO genérico ?

Modelado de un sistema de eventos con QoS

Sean z_i, $i = 1 \ldots N$ las fracciones asignadas a N tareas y s_i el correspondiente tasa de envío en eventos por segundo evt/s.
La tasa de envío real r_i para el topic i esta dada por

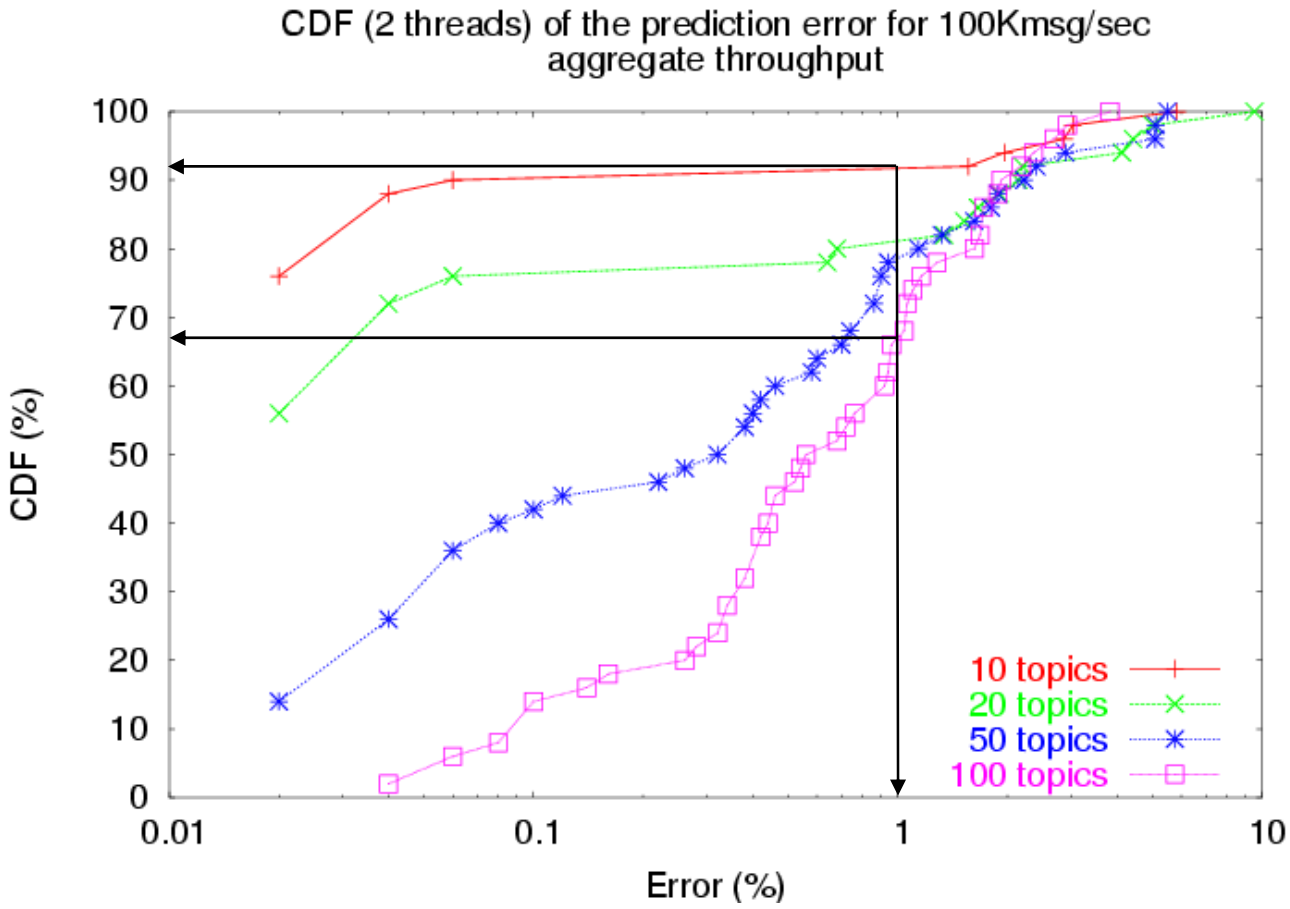
$$r_i = \min \left(s_i,\ z_i \frac{R^{max} - \sum_{x_j > x_i} r_j}{1 - \sum_{x_j > x_i} z_j} \right)$$

donde:

- R^{max} es la tasa total máxima realizable por la máquina.
- $x_k = z_k \left(\sum_{l=1}^{N} s_l / s_k \right)$ es la "exigencia" de una tarea.
- Weighted Max-Min fairness allocation.

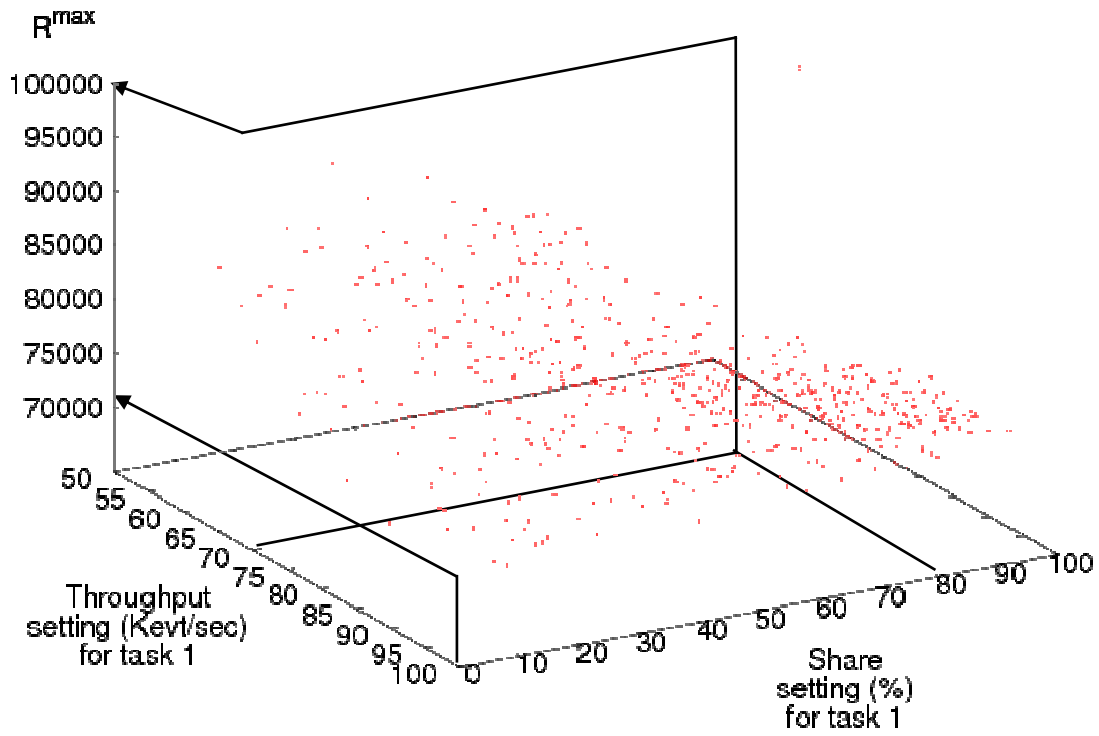
El modelo describe el sistema con precisión

- ¡Pero depende del parámetro R^{max} !



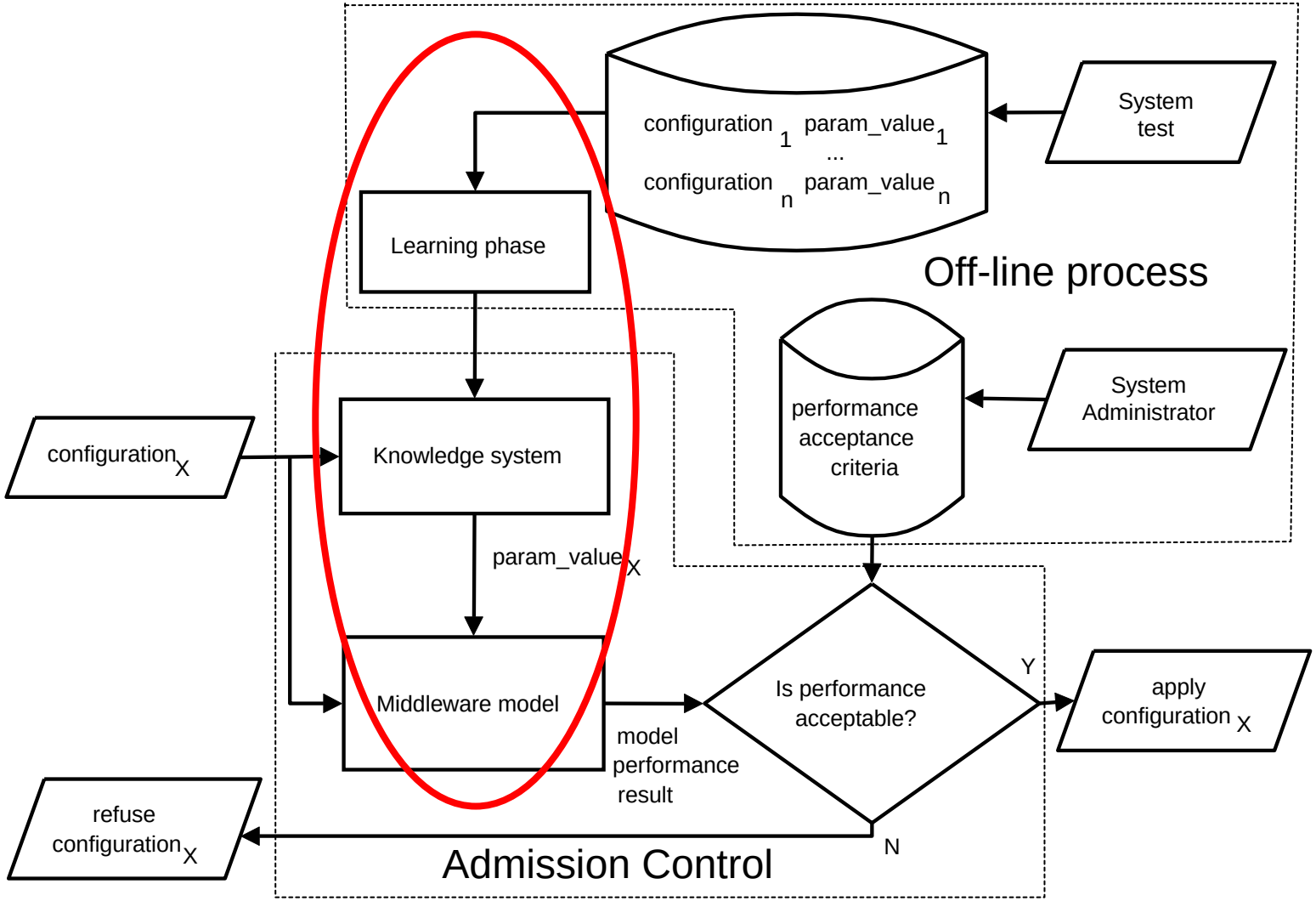
R^{max} depende de la configuración

- Un espacio de configuración simple con 2 actividades



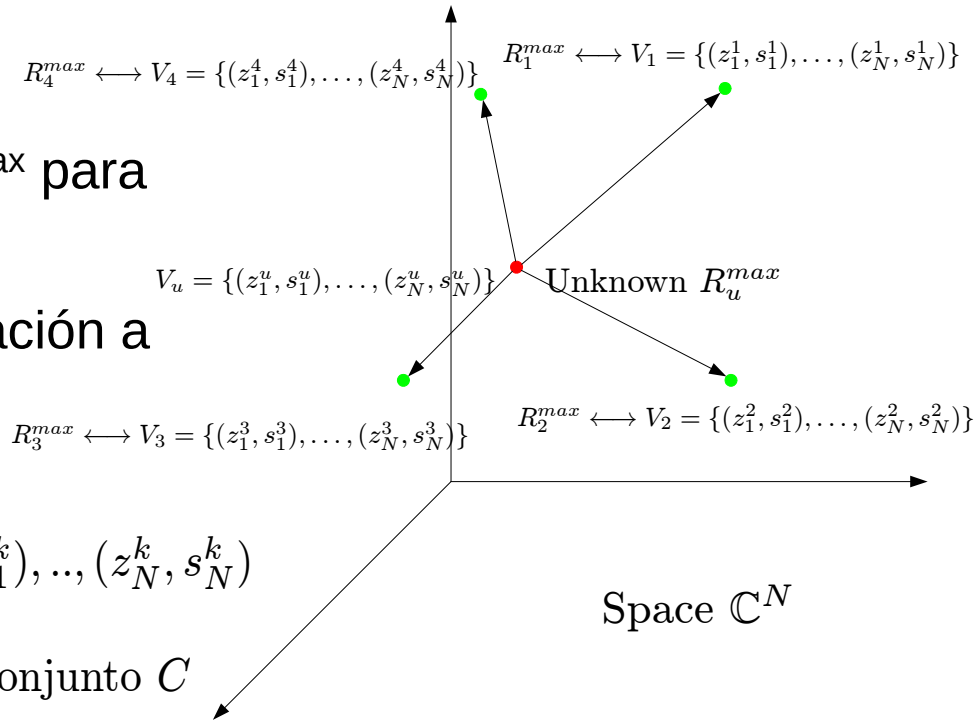
- Throughput total S: 100Kevt/sec
- $s_{task2} = S - s_{task1}$
- $z_{task2} = 100 - z_{task1}$
- $R^{max} \in \{70K, 100K\}$

Mecanismo de control de admisión



Estimación de R^{max}

- Calibrar el sistema obteniendo R^{max} para algunas configuraciones conocidas
- Extrapolar R^{max} para una configuración a partir de otras “cercanas”



Cada configuración es un vector $V_k = (z_1^k, s_1^k), .., (z_N^k, s_N^k)$

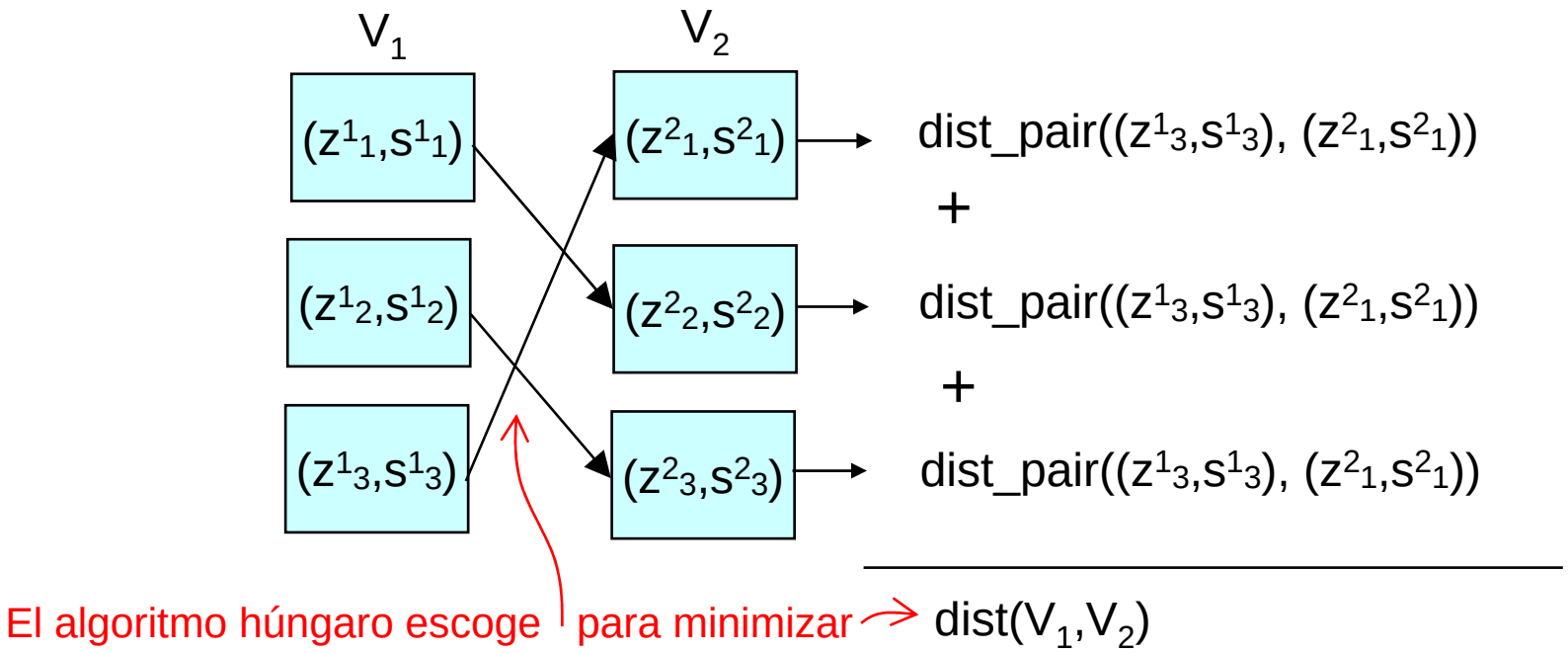
El resultado del proceso de calibrado es el conjunto C de pares de la forma $(V_k,\ R_k^{max})$

Para determinar el R_u^{max} de una configuración desconocida V_u usar la media ponderada de los puntos de C, donde los pesos son una función de ’parecido’.

$$R_u^{max} = \sum_{(V_i, R_i^{max}) \in C} weight(V_i, V_u, C) R_i^{max}$$

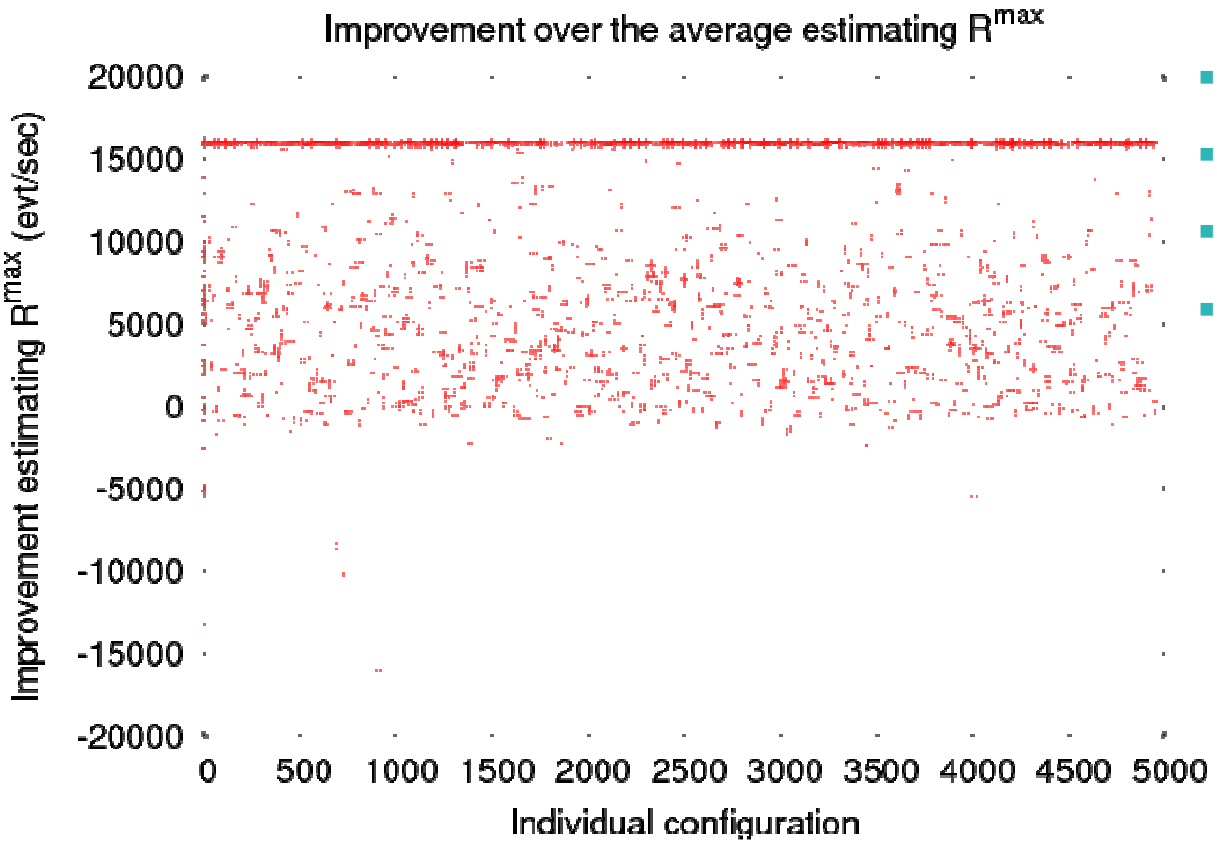
"Parecido" de las configuraciones

- Cada configuración *k* es un vector $V_k=\{(z^k_1,s^k_1),..,(z^k_N,s^k_N)\}$
- Para una configuración con 2 actividades, {(60,1000),(40,500)} es lo mismo que {(40,500),(60,1000)}: los elementos del vector no se pueden comparar siguiendo su posición



Estimador de R^{max}: espacio completo de 2 actividades

- nClosestPredictor: media de las R^{max} de las n configuraciones más cercanas
- Los puntos más cercanos son mejores estimadores que la media



- $z_1, z_2 \in \{0:1:100\}$
- $s_1 \in \{50K:1000:100K\}$
- $s_2 \in \{0K:1000:50K\}$
- Todas las 5,000 configuraciones $(z_1,s_1),(z_2,s_2)$ posibles:

  (100,100K),(0, 0K)

  (100, 99K),(0, 1K)

  …

  (0, 51K),(100, 49K)

  (0, 50K),(100, 50K)

Estimador de R^{max}: muestreo aleatorio del espacio de configuraciones

- 10 actividades, 1,000 configuraciones aleatorias (throughput total 100Kevt/sec)
- Cuantas más configuraciones testadas, mejor es la predicción
- Hasta cierto punto …

Árboles de decisión: C4.5 (I)

- La construcción de árboles de decisión a partir de objetos ya clasificados y sus atributos es en general NP-Completo
- ID3: algoritmo greedy
  - En cada nodo, se crean ramas basándose en el atributo que más reduce la entropía de los objetos clasificados bajo ese nodo hasta ahora
  - Nodo con entropía 0 se convierte en una hoja del árbol (clase)
- C4.5 incluye mejoras sobre ID3:
  - Evita sobre-ajuste del árbol podando ramas superfluas
  - Los atributos pueden tener rangos reales
- Para los experimentos usamos el C4.5 R8 en C de Quinlan

Árboles de decisión: C4.5 (II)

[Quinlan86]: Dado un conjunto de objetos C pertenecientes a dos clases P y N, con $|P| = p$ y $|N| = n$, la información dada por el árbol de decisión es

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Dado un atributo A con valores $A_1, A_2, \ldots, A_v$, divide C en $C_1, C_2, \ldots, C_v$, donde C_i contiene p_i objetos de la clase P y n_i objetos de clase N. La información esperada dada por el árbol dividido en A es

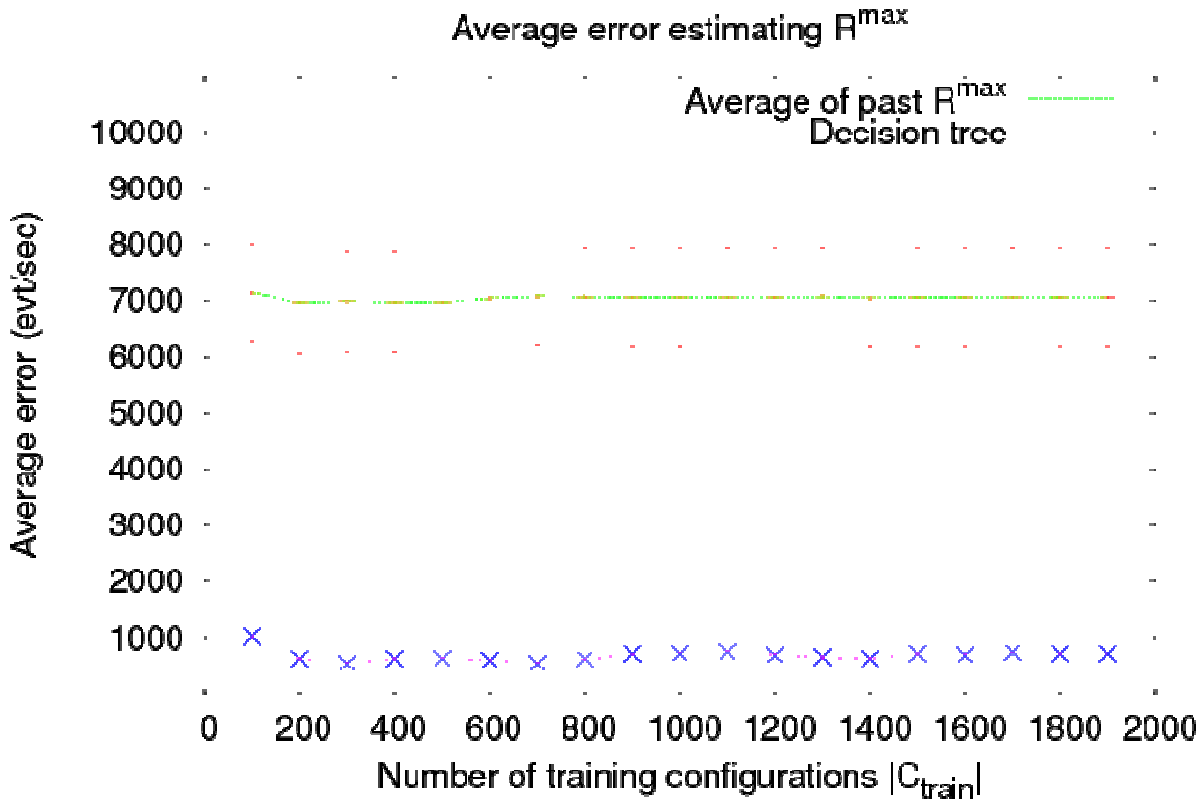
$$\sum_{i=1}^{v} \frac{p_i+n_i}{p+n} I(p_i,n_i)$$

Algoritmos de Machine learning: Árboles de decisión

- Los árboles de decisión son sistemas de propósito general basados en conocimiento
- Son útiles para expresar relaciones no lineales entre un grupo de objetos y otro de clases a través de los atributos de los objetos
- Los objetos son configuraciones $V = [(z_i^0, s_i^0) \dots (z_j^{t-1}, s_j^{t-1})]$ tales que $\forall p, q,\ if\ p < q \rightarrow s_i^p > s_j^q \ \vee \ s_i^p = s_j^q \ \wedge \ z_i^p > z_j^q$
- Las clases son rangos de valores posibles del parámetro del modelo

Estimador de R^{max} : Árbol de decisión

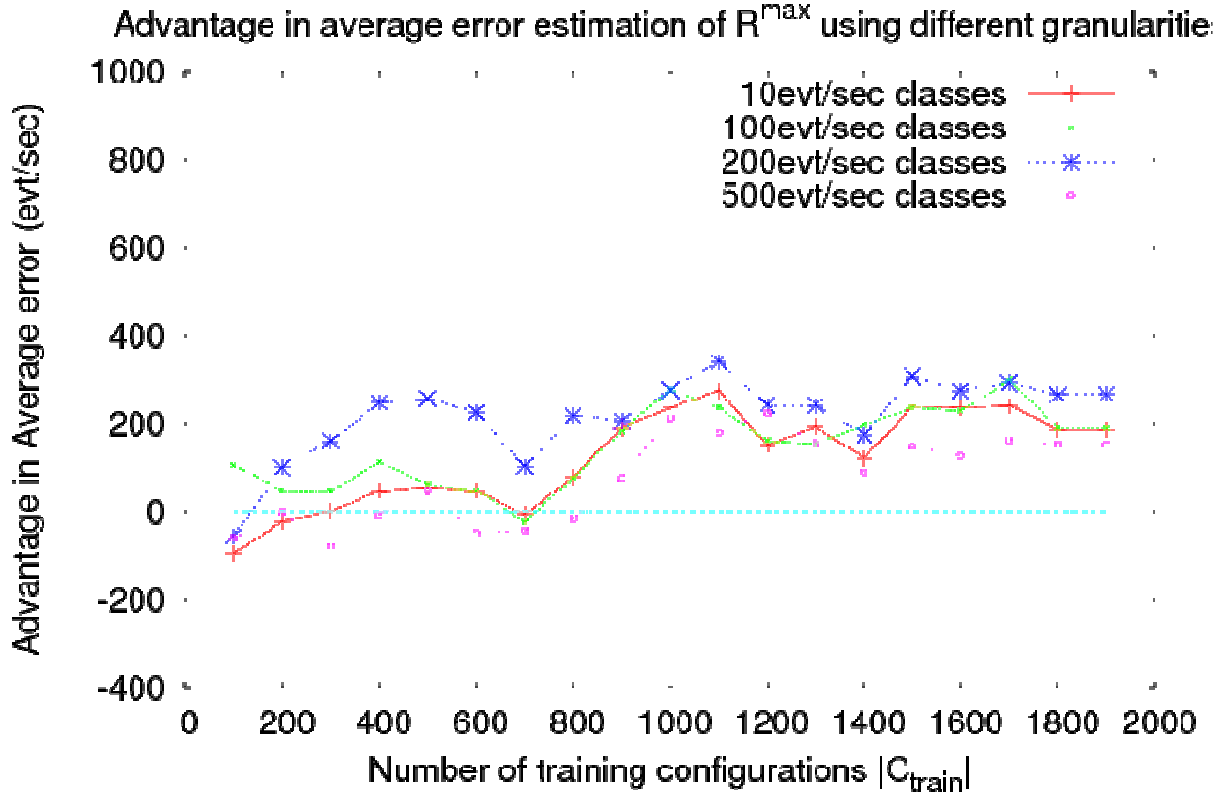
- Estimador Naif: la media de los R^{max} de las configuraciones anteriores
- ¡El error con el árbol de decisión es próximo al error de cuantización!



- 20 actividades
- 2,000 configuraciones
- Clases de 1,000 evt/sec
- Test con 100 configuraciones aleatorias.
- Entrenar con unos cientos de configuraciones es suficiente

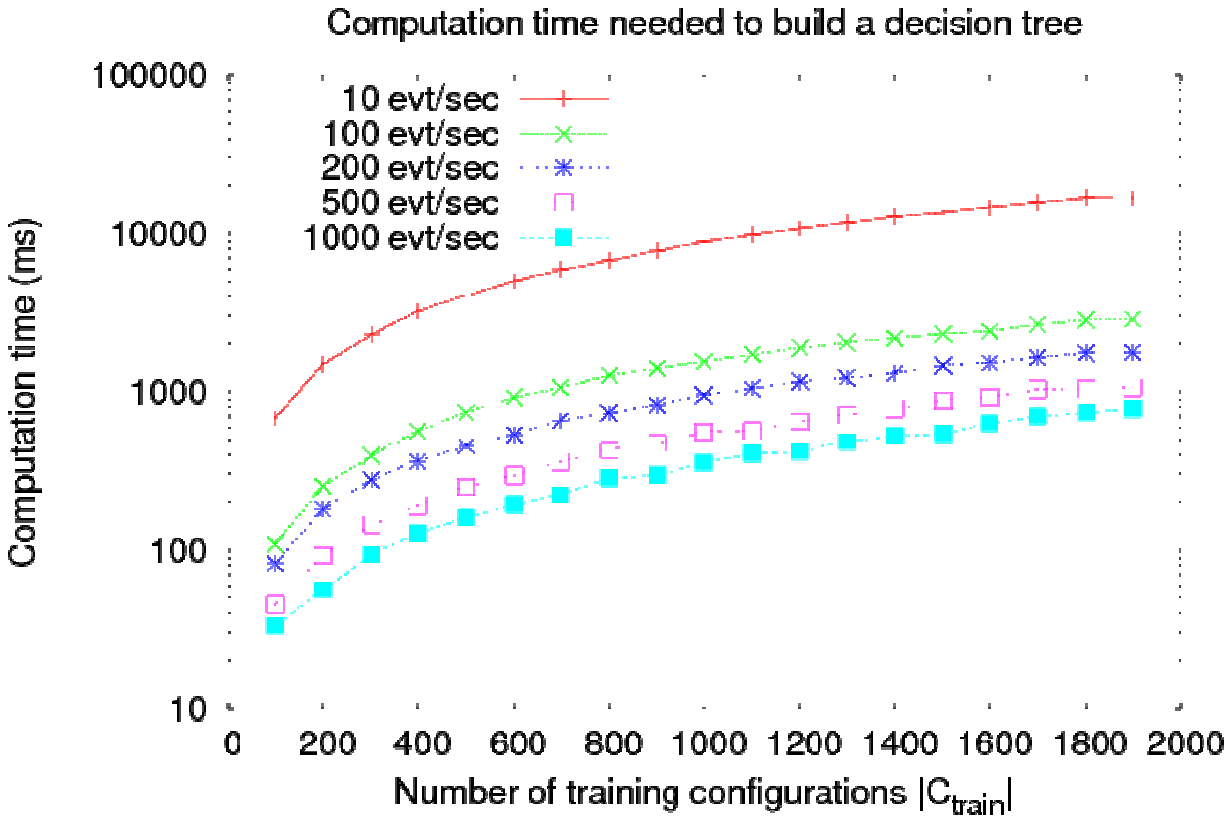
Granularidad de las clases

- La precisión puede mejorar hasta cierto punto
- Todas las clases mejoran por encima del error de cuantización



Costo de computación: construcción del árbol

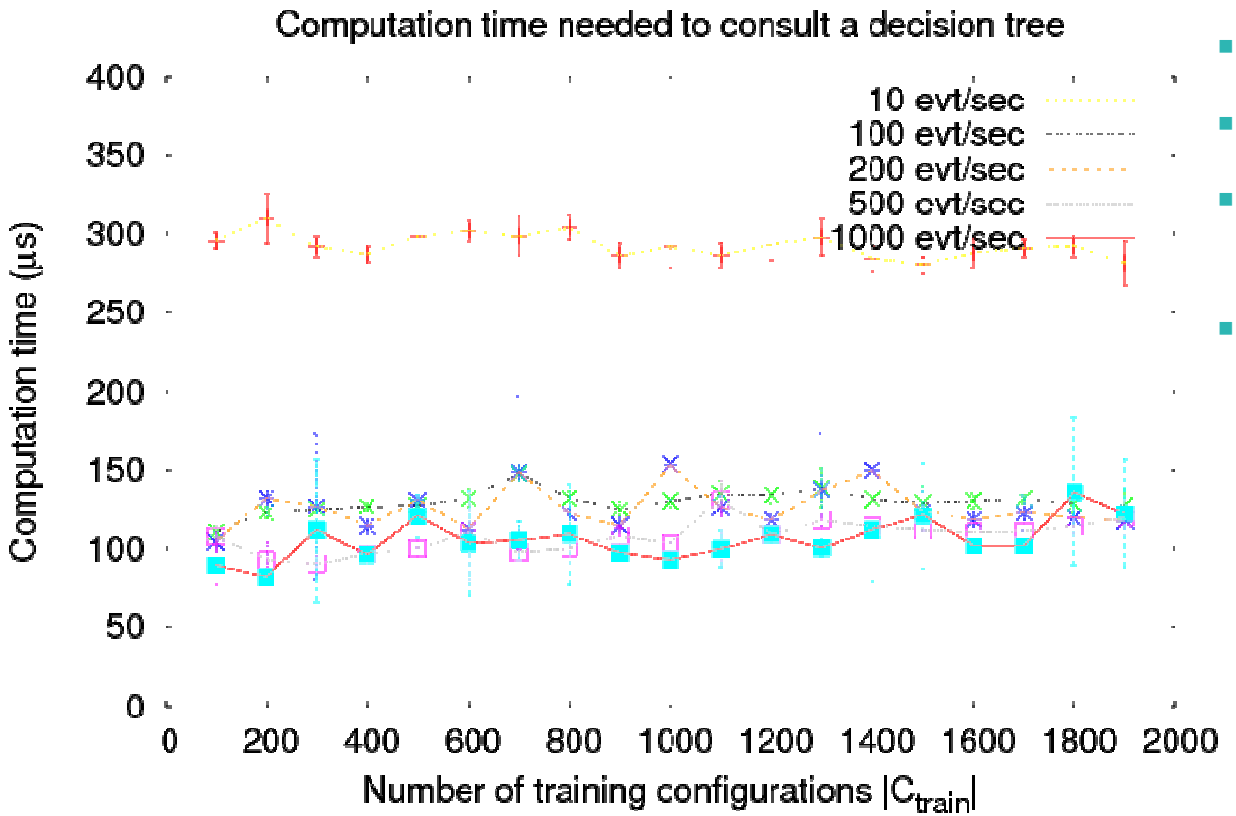
- Recomputación del árbol en tiempo real posible si el costo es bajo



- 20 actividades
- 2,000 configuraciones
- La clase de 10evt/sec requiere mucho trabajo sin aumentar la precisión
- Tiempo de cálculo sub-lineal con el número de configuraciones

Costo de computación: clasificación

- No interfiere con la operación normal del sistema



- 20 actividades
- 2,000 configuraciones
- La clase de 10evt/sec es la más costosa (gran número)
- El entrenamiento afecta sólo a la creación del árbol, no a la clasificación

Preguntas sin contestar

- ¿Cuantos puntos necesita calibrar un administrador para que el sistema sea capaz de predecir su rendimiento?
- ¿Podemos aplicar otras técnicas de ML, como el clustering?
- ¿Cómo afectaría al sistema un coste por evento no uniforme en las diferentes actividades?

Ideas principales (revisión)

- Los sistemas de tiempo real tradicionales necesitan mucho control: actividades conocidas, planificador analítico
  - Control de admisión puede implementarse analíticamente
- Los complejos sistemas de software actuales tienen cada vez más necesidades de “tiempo real”
  - Proponemos un middleware que asigna recursos a actividades
  - Si modelamos el middleware podemos proveer control de admisión
- La complejidad de estos sistemas conlleva modelos que dependen de parámetros de tiempo de ejecución:
  - Imposibles de resolver de manera analítica
  - El sistema necesita ser calibrado para “aprender” estas dependencias