SO: Escalonamento - Parte 2

Sistemas Operacionais
2017-1
Flavio Figueiredo (http://flaviovdf.github.io)

Prioridades

Escalonamento com Prioridades

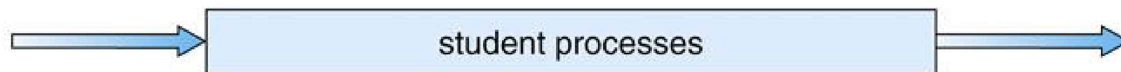
- Associar uma prioridade para cada processo
  - No Unix, valores menores implicam em maior prioridade

- Processos de maior prioridade são alocados primeiro

- *[Aula Passada] Shortest job first*
  - Pode ser visto como um escalonamento com prioridades
  - A prioridade é o predição do quantum

Escalonamento Multinível

highest priority



lowest priority

Quais são os problemas de escalonar com prioridades?

Problemas de usar Prioridades

- Continuamos com problemas de *starvation*

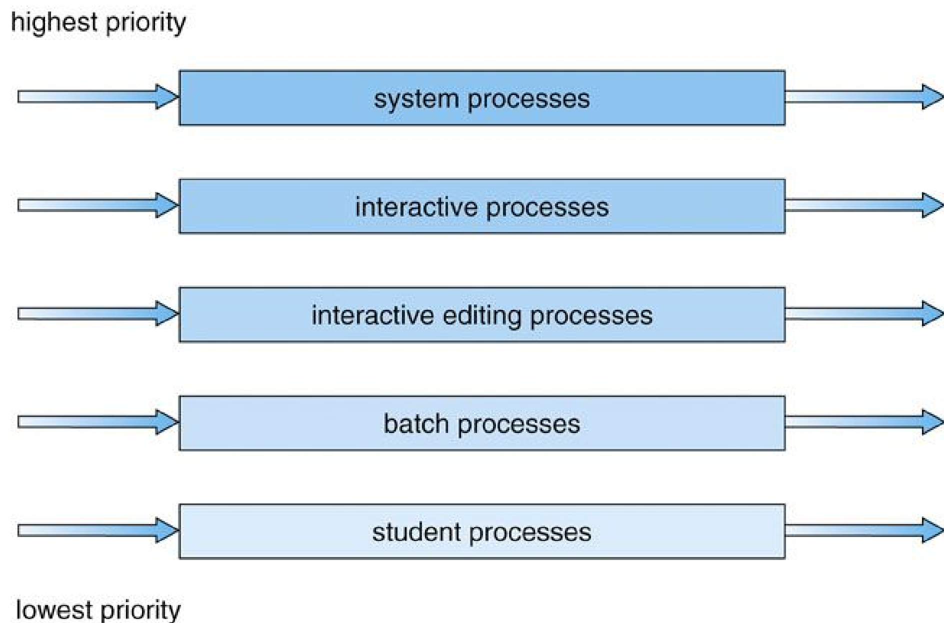
- Processo de baixa prioridade pode nunca ser executado

- Solução (Aging):
  - Aumentar a prioridade com o tempo de espera
  - Eventualmente o processo vai executar
  - Sem garantias de quando

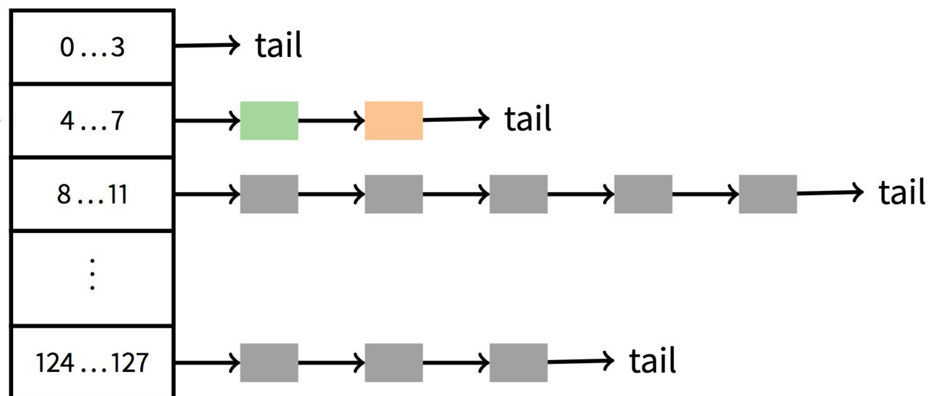
Algumas Formas de Implementar

- Processos de prioridade baixa só executam após as filas superiores liberarem
  - Batch Jobs apenas quando as 3 filas superiores estiverem vazia

- Fatiamento de tempo entre filas
  - Algumas filas com 70% de cpu
  - outras com 10%
  - ...
  - Melhor no quesito de starvation



Feedback Queues (Retroalimentação)



- Processos do kernel na primeira fila não vazia (atualizado de tempo em tempos)
  - Diferente de sempre ter maior prioridade (exemplo, processo laranja acima)
- [Aging] Mudar processos de prioridade de acordo com necessidade de CPU
  - [Contra-Intuitivo] **Mais prioridade** para processos com **menor** uso de CPU
  - Round Robin em cada fila
- [Foco] Processos interativos acabam com maior prioridade. Qual o motivo?

[Side-Note] Processos Interativos

- Pouco uso de CPU

- Porém vários picos

- Browser:
  - Esperando cliques do usuário
  - Pegando dados da rede
  - Casos acima causa system calls e interrupções

- Em comparação. Batch:
  - Sempre preciso de 100% de CPU
  - Experimentos de doutorado

Feedback Queues Resumido com Poucas Regras

- **Rule 1:** If Priority(A) > Priority(B), A runs (B doesn't).
- **Rule 2:** If Priority(A) = Priority(B), A & B run in RR.
- **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue).
- **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- **Rule 5:** After some time period S, move all the jobs in the system to the topmost queue.

Como fazer a mudança de prioridade?

- Solaris
  - Conjunto de regras
  - [Exemplos de Regras]
    - Após 1 segundos na fila de prioridade 5 mudar para fila de prioridade 6
    - Renovar prioridades a cada 60 segundos

- FreeBSD
  - Equações que são atualizadas dinamicamente. A cada timer interrupt
  - Consideram o uso de CPU passado e uma estimativa da caga (*load*) do sistema

http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-mlfq.pdf

FreeBSD 4.4

- Processos em execução aumenta o p_estcpu. Note o aumento com p_nice nicess level. (2x / (2x+1) sempre é menor do que 1)

$$\texttt{p_estcpu} \leftarrow \left(\frac{2 \cdot \text{load}}{2 \cdot \text{load} + 1}\right) \texttt{p_estcpu} + \texttt{p_nice}$$

- Processos dormindo reduzem o p_est_cpu (na verdade aumentar o p_slptime)

$$\texttt{p_estcpu} \leftarrow \left(\frac{2 \cdot \textbf{load}}{2 \cdot \textbf{load} + 1}\right)^{\texttt{p_slptime}} \times \texttt{p_estcpu}$$

- max(*p_usrpri/4, 127) ->* determina a fila *prioridade*. Lembre-se que maior é pior

$$\texttt{p_usrpri} \leftarrow 50 + \left(\frac{\texttt{p_estcpu}}{4}\right) + 2 \cdot \texttt{p_nice}$$

Maioria dos SOs usam alguma forma de prioridades

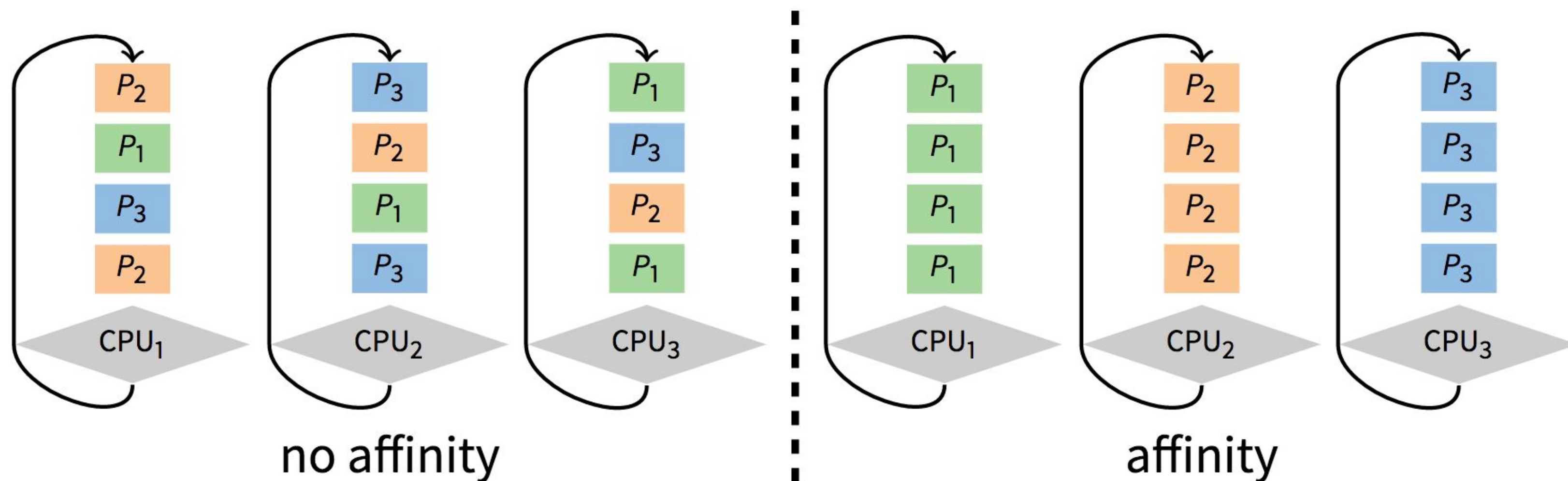
Multi Processamento

Sistemas com Multiprocessadores

- Multiprocessamento simétrico (SMP)
    - Todas as CPU tem acesso às estruturas do kernel
    - Problemas de contenção
    - Maioria dos SOs

- Multiprocessamento assimétrico (AMP)
    - Apenas um processador tem acesso às estruturas do kernel
    - Mais simples
    - Primeiras versões do Linux com multiprocessamento
    - Pouco utilizado hoje em dia

Afinidade



- [Lembre-se] Trocas de contexto afetam o tempo de execução
- Re-popular caches pode desperdiçar bastante tempo
- Afinidade com processador ajuda
- **sched_setaffinity** no Linux

Afinidade

- Pode ser setado com syscall
- [Melhor Ainda] Um bom escalonador mantém afinidade com CPU

- *Gang Scheduling*
  - Assumindo um Quantum fixo
  - Todas as CPUs escalonam no mesmo tempo
  - Mantém as mesmas tarefas no curto prazo

- Escalonadores do Linux tentam sempre manter processos nas mesmas cores
  - O(1) Scheduler (outdated)
  - CFS Scheduler

Balanceamento de Carga

- Processadores ficam ociosos com o tempo

- [Opção 1] Roubar a tarefa de outro processador
  - Nenhum processador nunca fica ocioso
  - Afeta a afinidade

- [Opção 2] Rebalancear a carga de tempos em tempos
  - Processadores ficam ociosos até a carga ser balanceada
  - Pode ser melhor para afinidade (sem garantias)

Escalonamento Real Time

- Alguns processos com prazos bem definidos

- Soft Real Time
  - Streaming de músicas ou vídeos
  - Queremos manter uma certa qualidade
  - Um pouco de erro não faz mal
  - Pode ser tratado com uma fila FIFO de prioridade especial (+CPU)

- Hard Real Time
  - Sistemas especialistas
  - Casos de erros implicam impactos na vida real (quedas de avião)
  - Políticas específicas de escalonamento

Exemplos

XV6

- Round Robin Simples

- Os processadores usam a mesma fila de processos

- Multiprocessamento simétrico

- Um efeito de Gang Scheduling é esperado
  - Quantum fixo
  - Ao terminar de executar P1 no core 0 (por exemplo), não devo roubar processos de outros cores (estão RUNNING). Assim se mantém uma certa afinidade com o processador

- Um laço nos processos para escolher o próximo
  - O(n)

Linux O(1) Scheduler

- Default até a versão 2.6 do Linux

- Garantia de que todo o escalonamento era feito em O(1)
  - Não depende do número de processos na fila

- 140 níveis de prioridade
  - 140 filas
  - Cada fila é identificada por 1 bit em um array de bits
  - Achar o primeiro bit setado é rápido
  - Setar prioridade também é bastante rápido
  - O(140) com uma constante baixa (bit operations)

Linux O(1) Scheduler

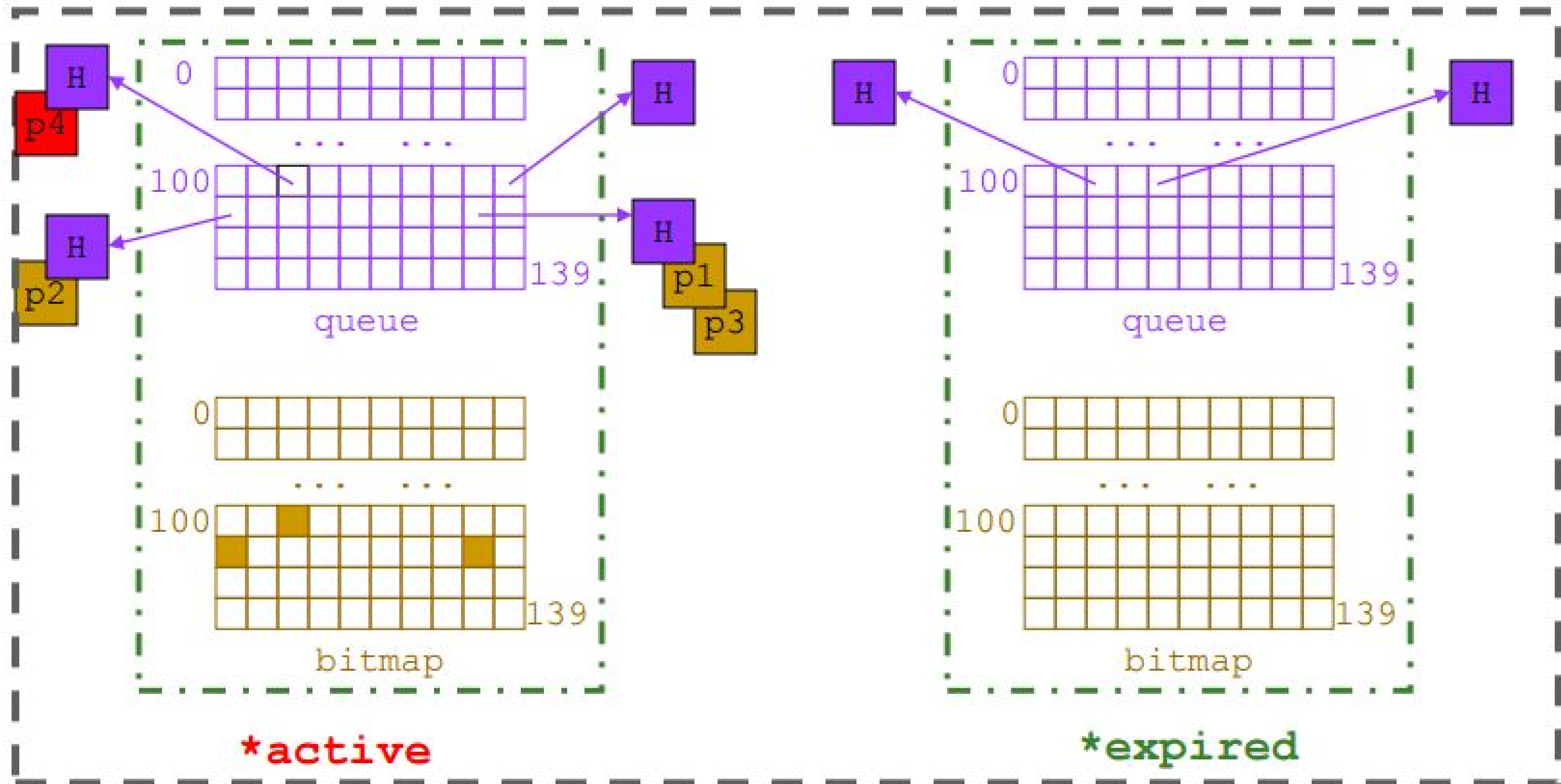
- O Linux na verdade mantém 3 grupos de execução

- Um grupo para tarefas normais
  - O(1) Scheduler ou CFS (mais a frente)

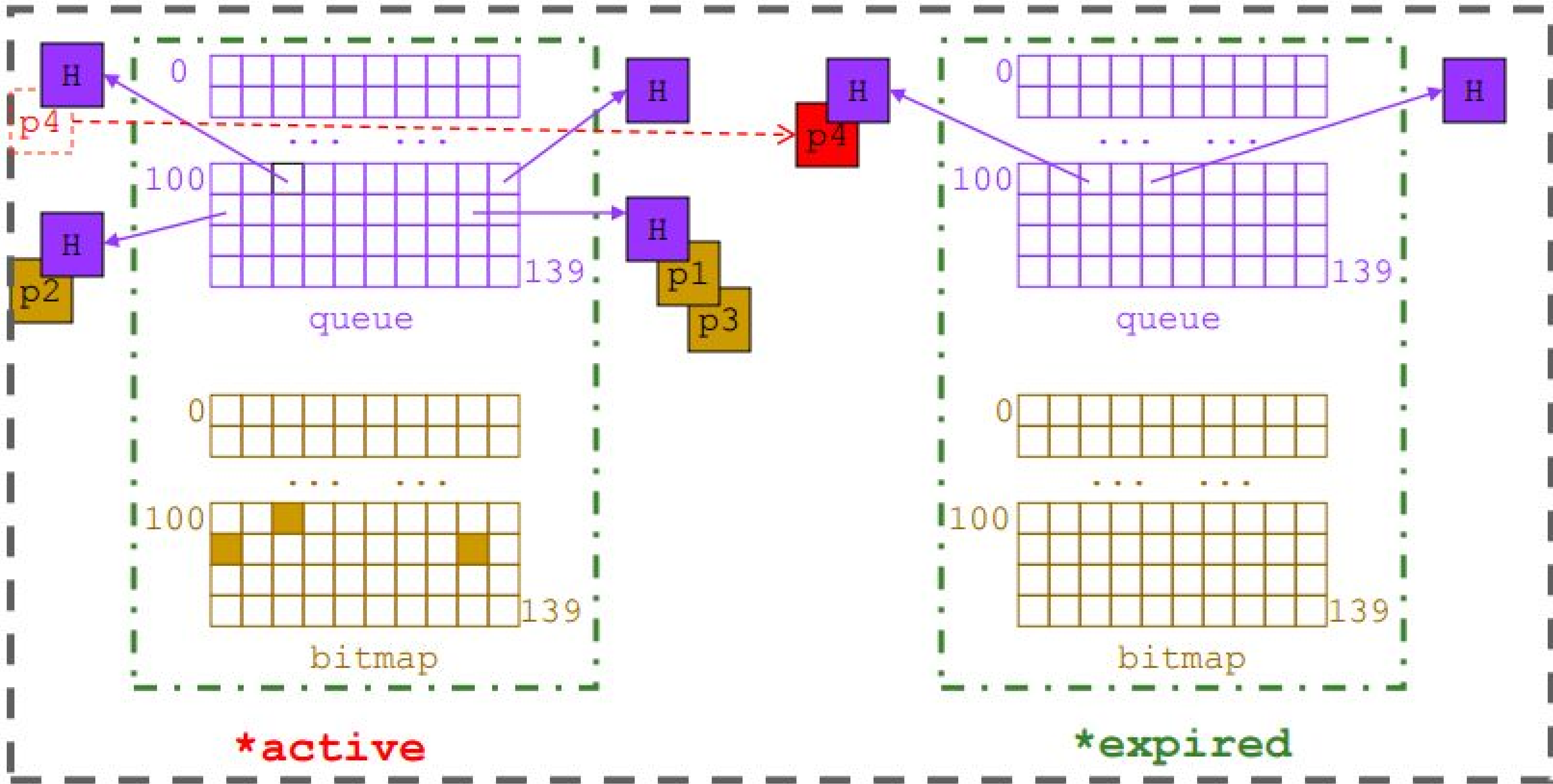
- Dois grupos para tarefas *real time (soft real time)*
  - Tarefas real time podem ser configuradas como FIFO
  - Round Robin

Escalonador O(1) no Linux

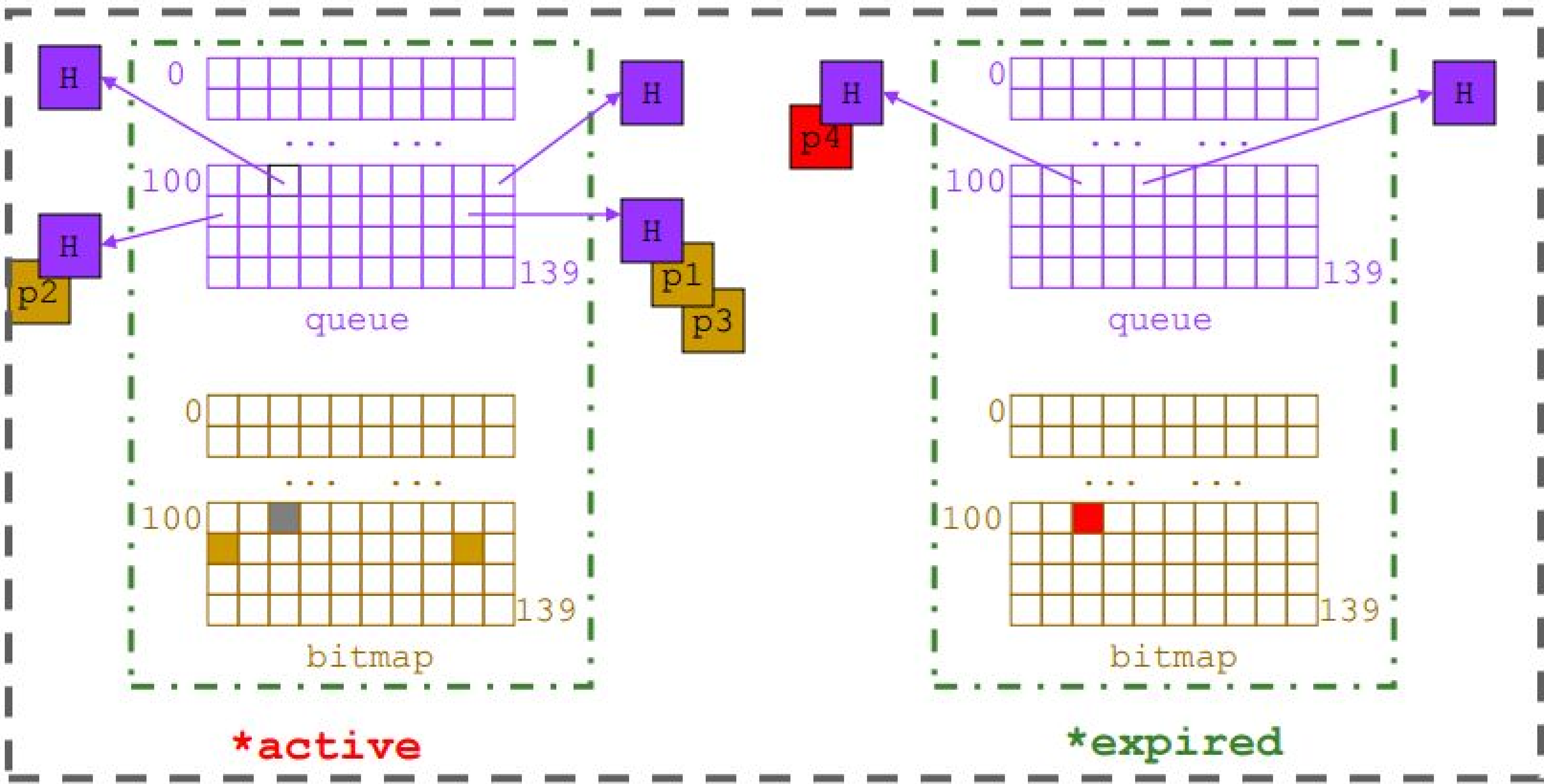
P4 tem maior prioridade.
H == head pointer



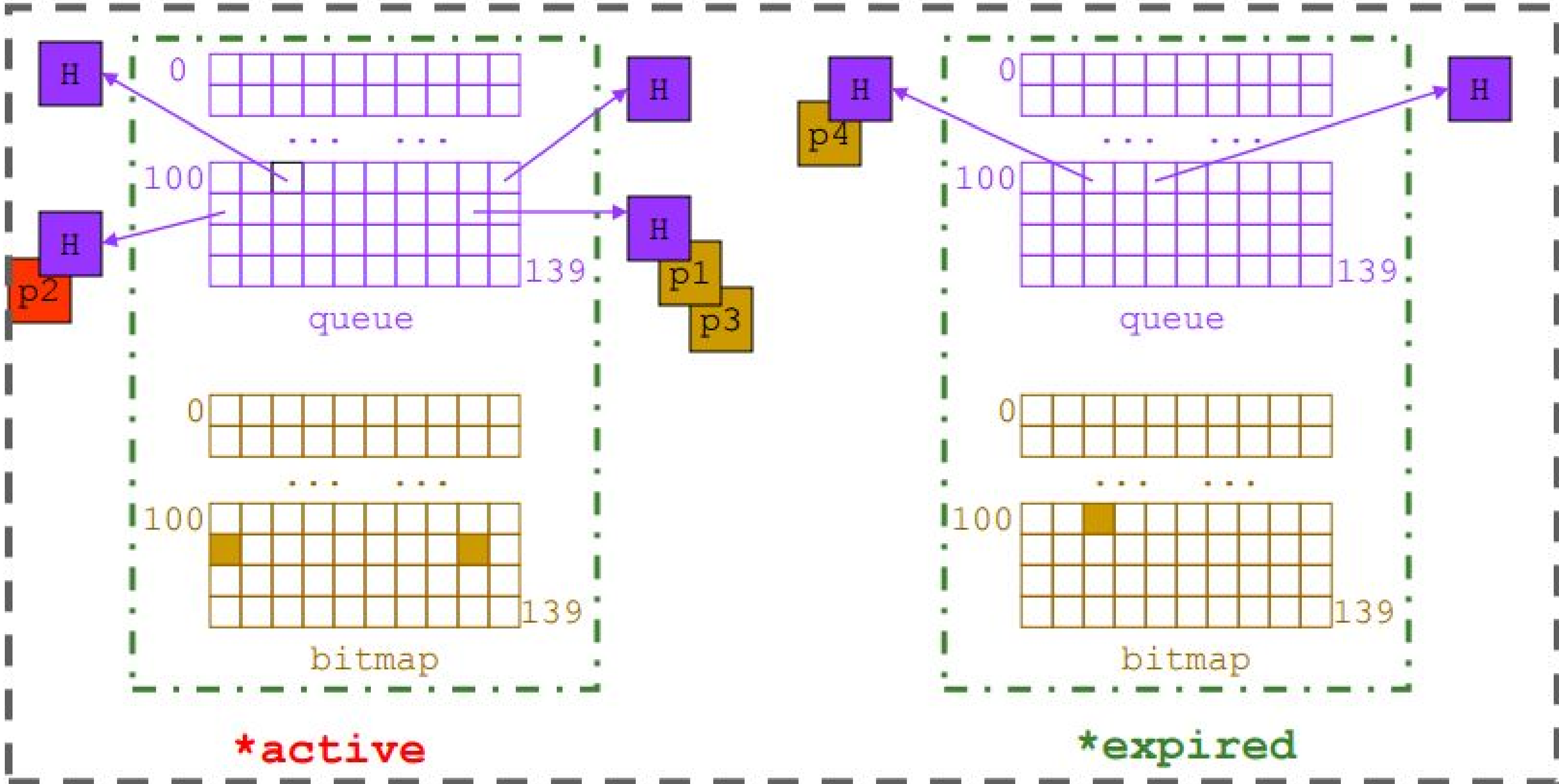
Escalonador O(1) no Linux



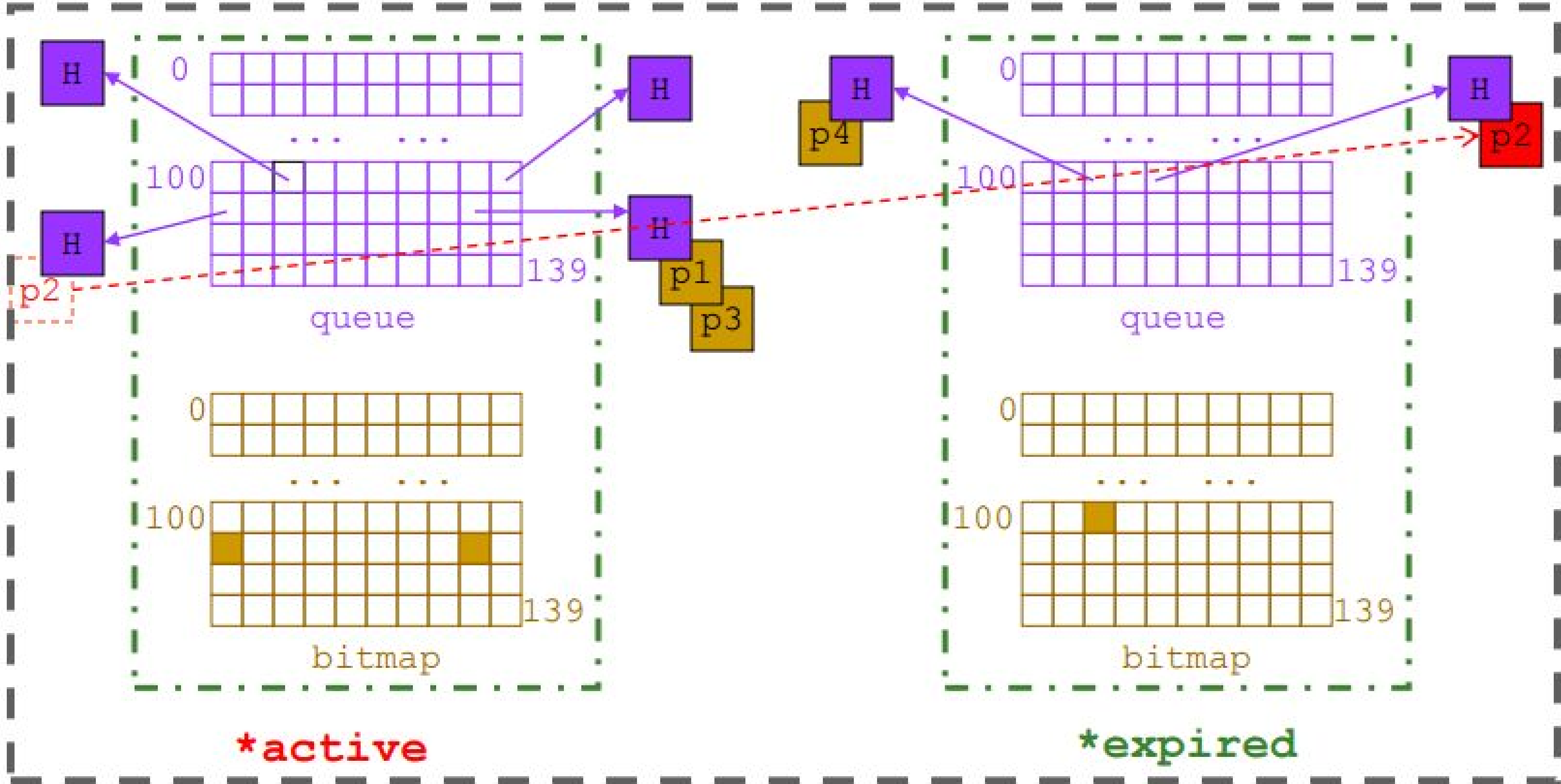
Escalonador O(1) no Linux



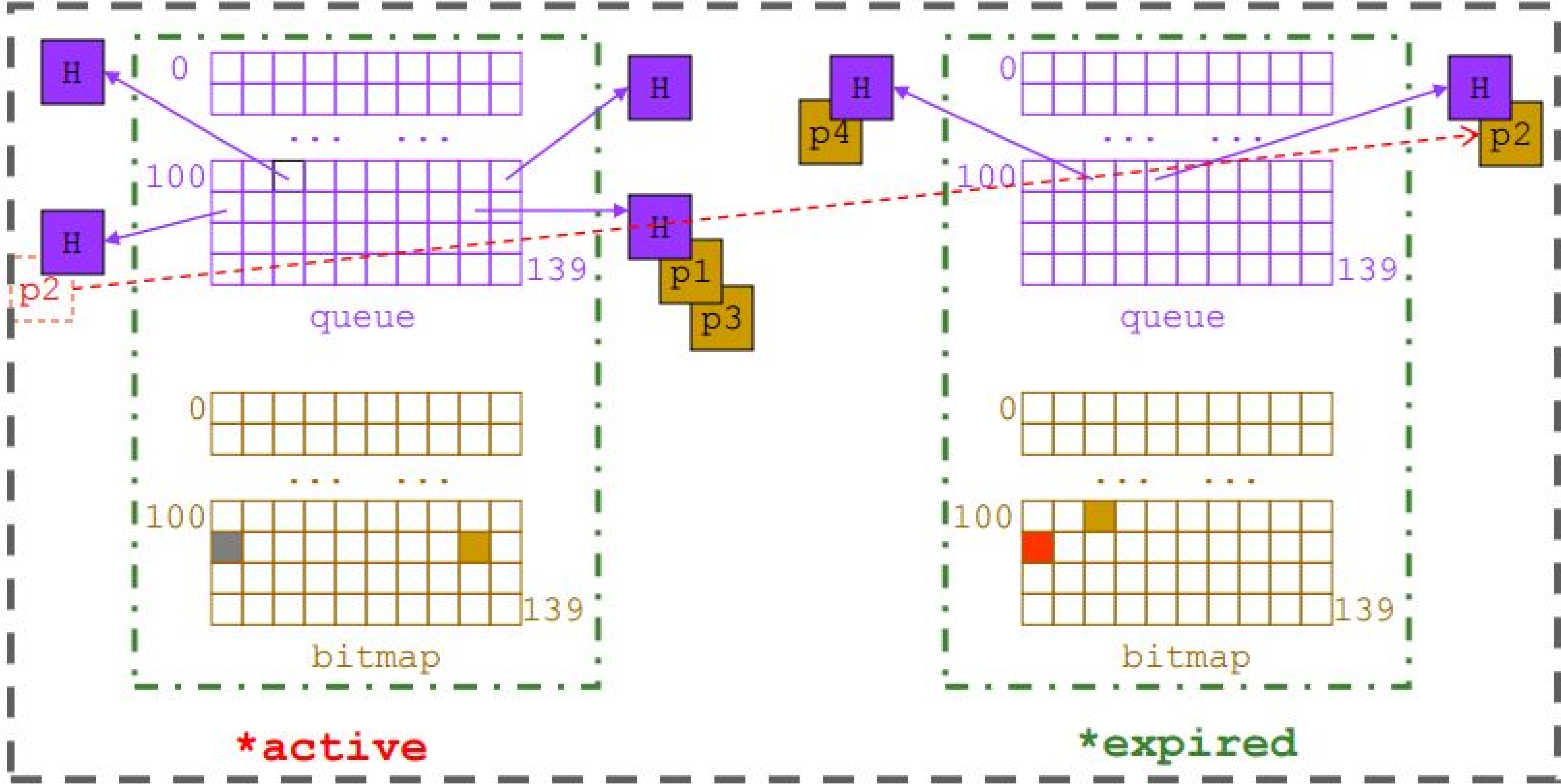
Escalonador O(1) no Linux



Escalonador O(1) no Linux



Escalonador O(1) no Linux

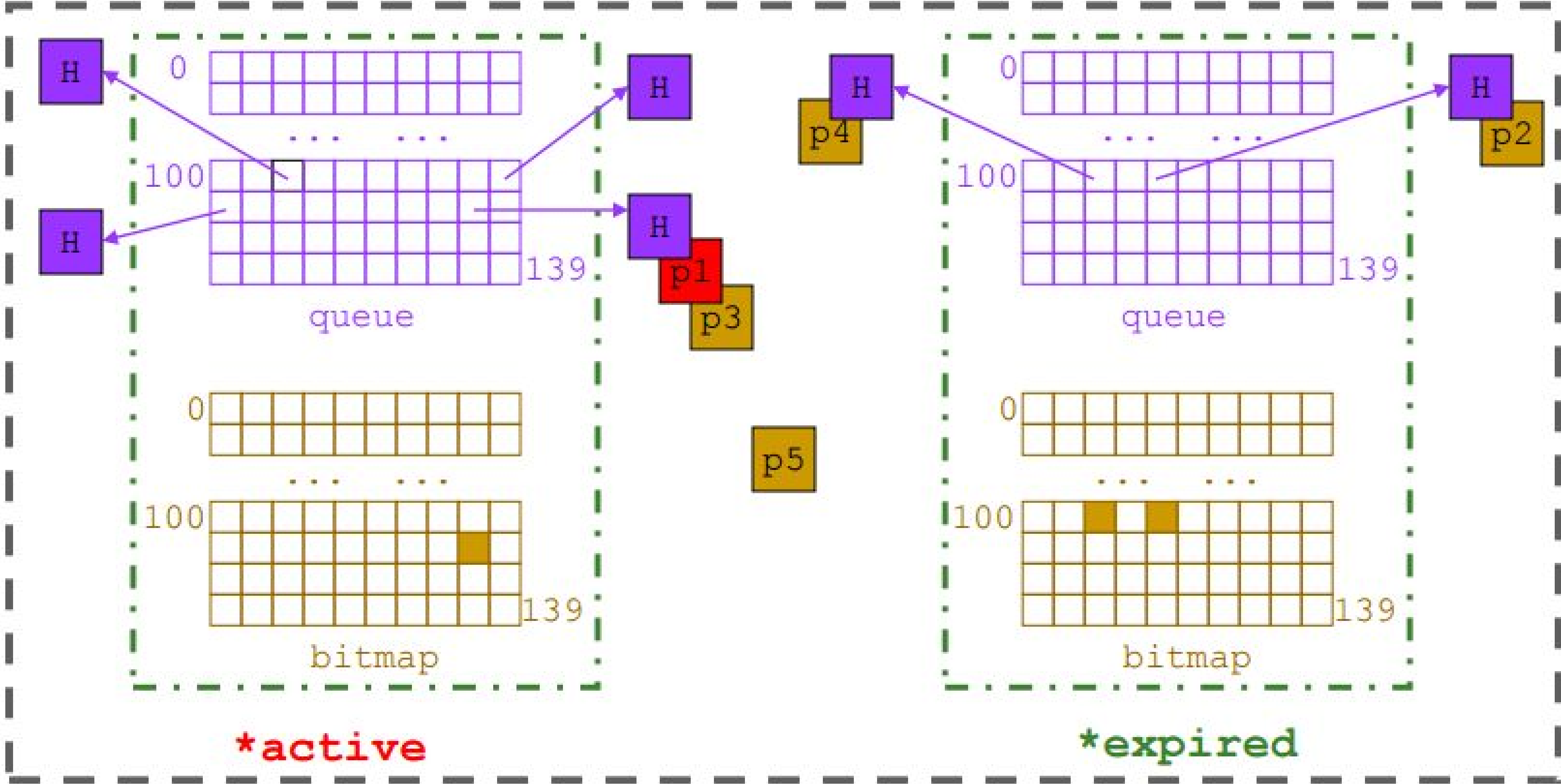


Escalonador O(1) no Linux

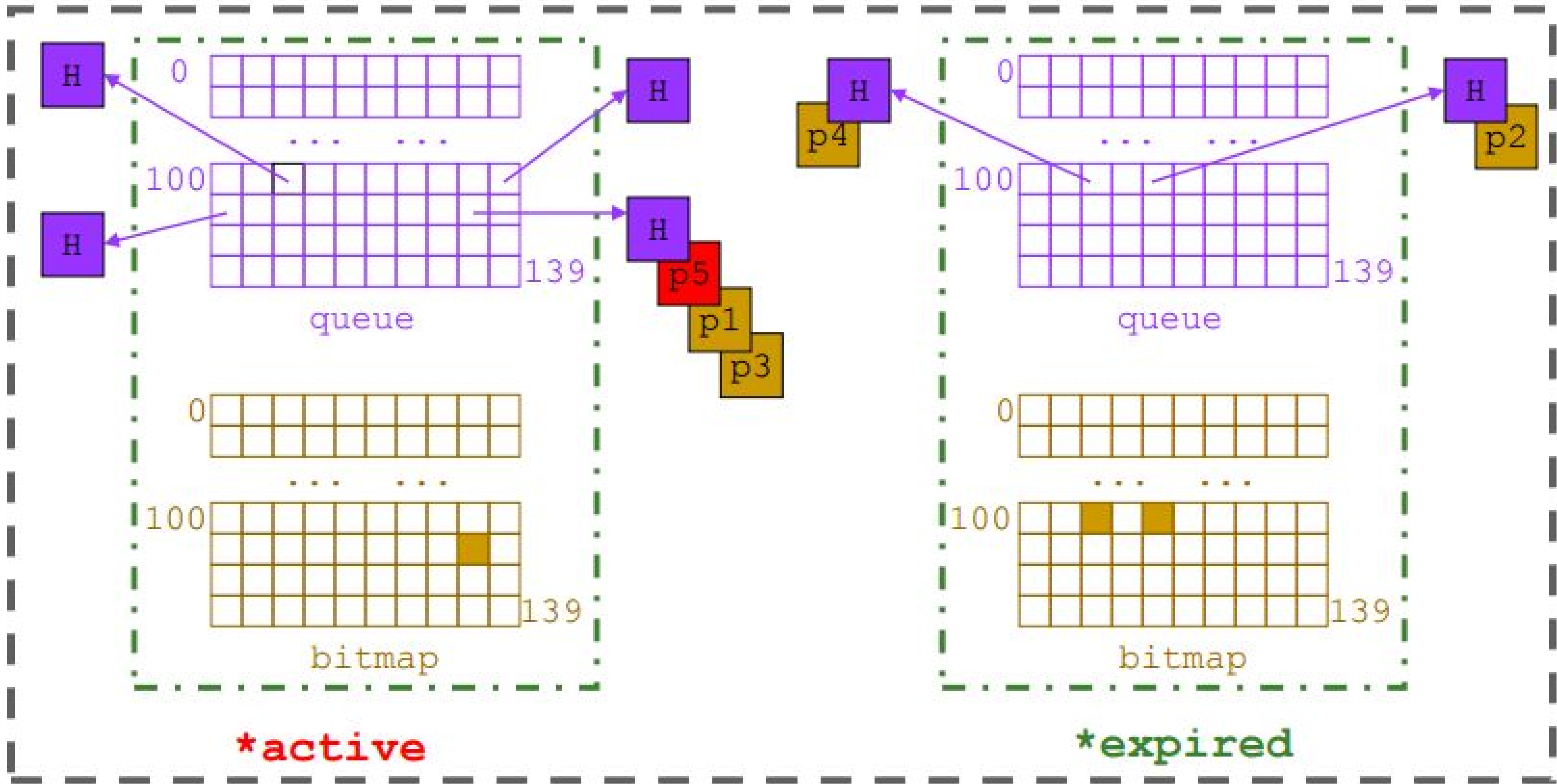
Mudança de prioridade.
Depende do tempo de execução

Escalonador O(1) no Linux

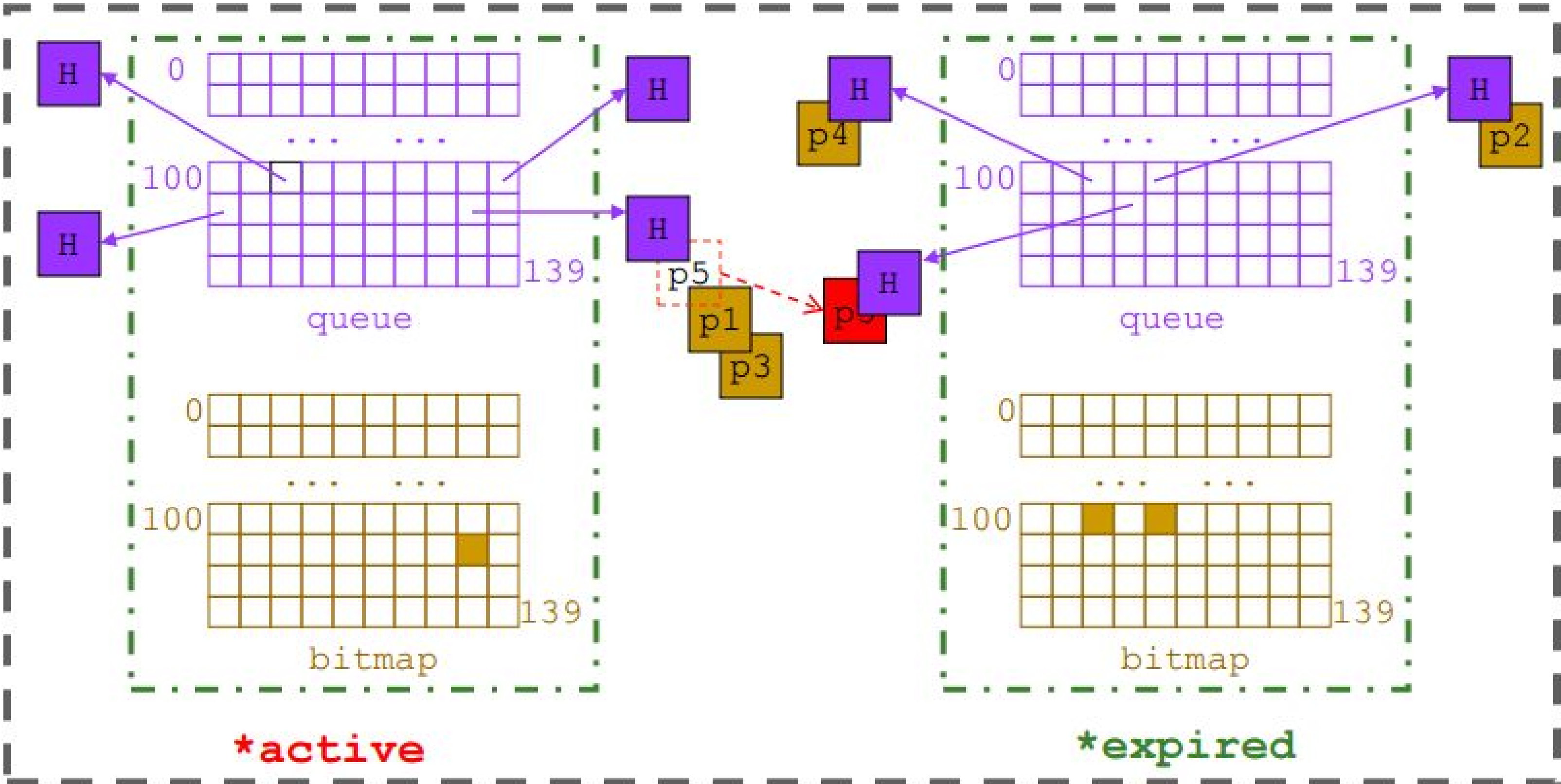
P1 fez um fork/exec criando P5



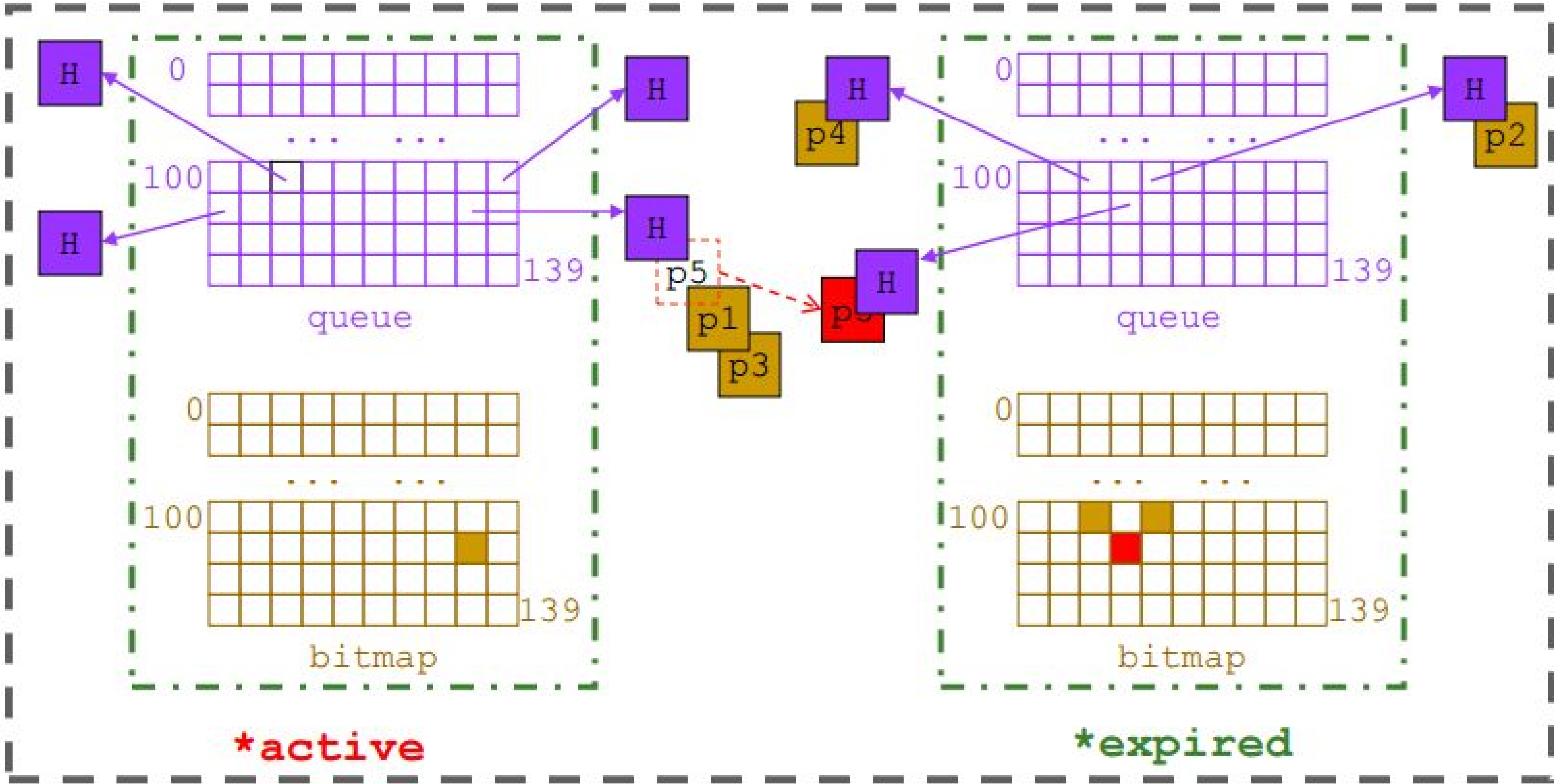
Escalonador O(1) no Linux



Escalonador O(1) no Linux

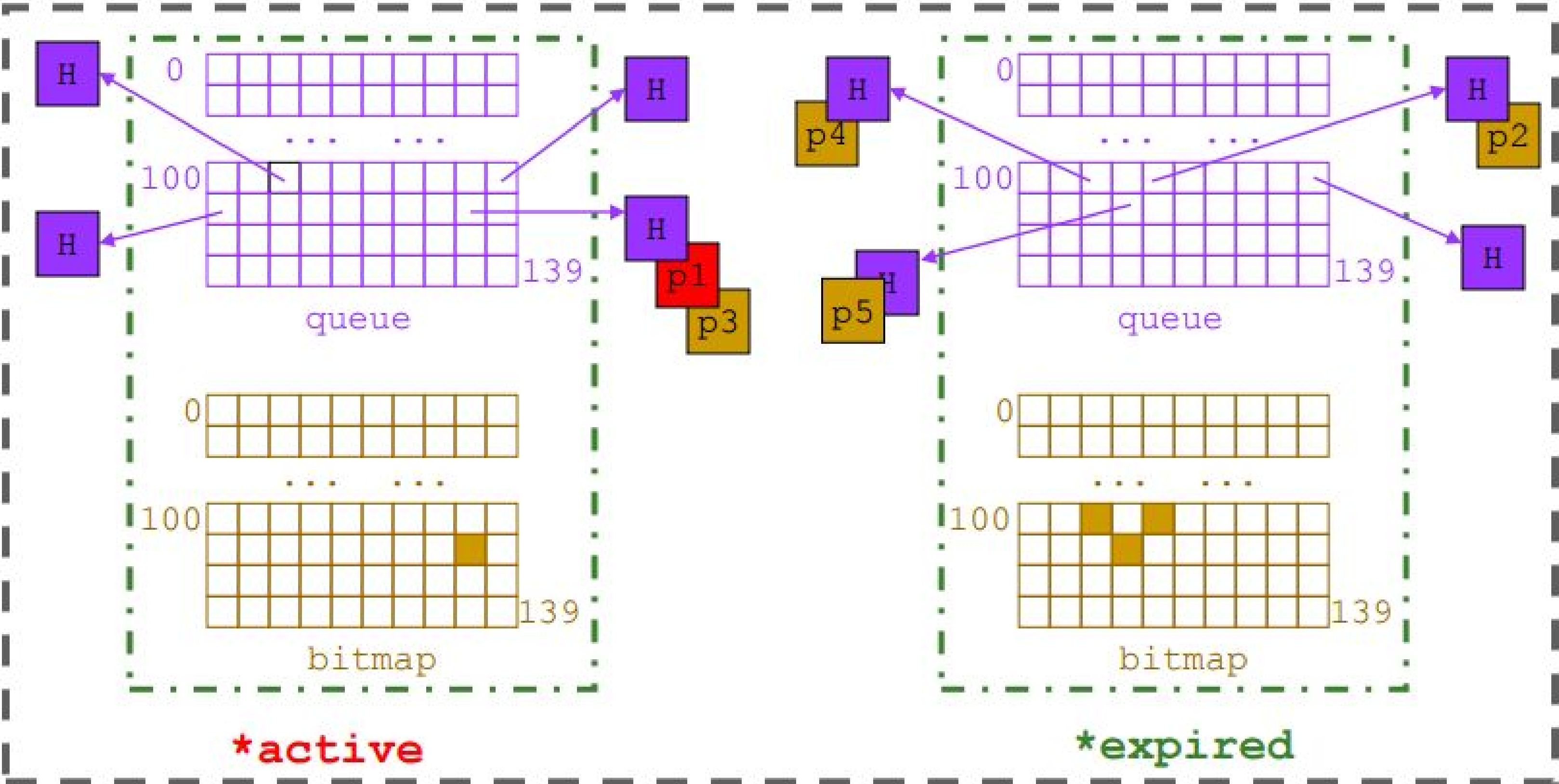


Escalonador O(1) no Linux

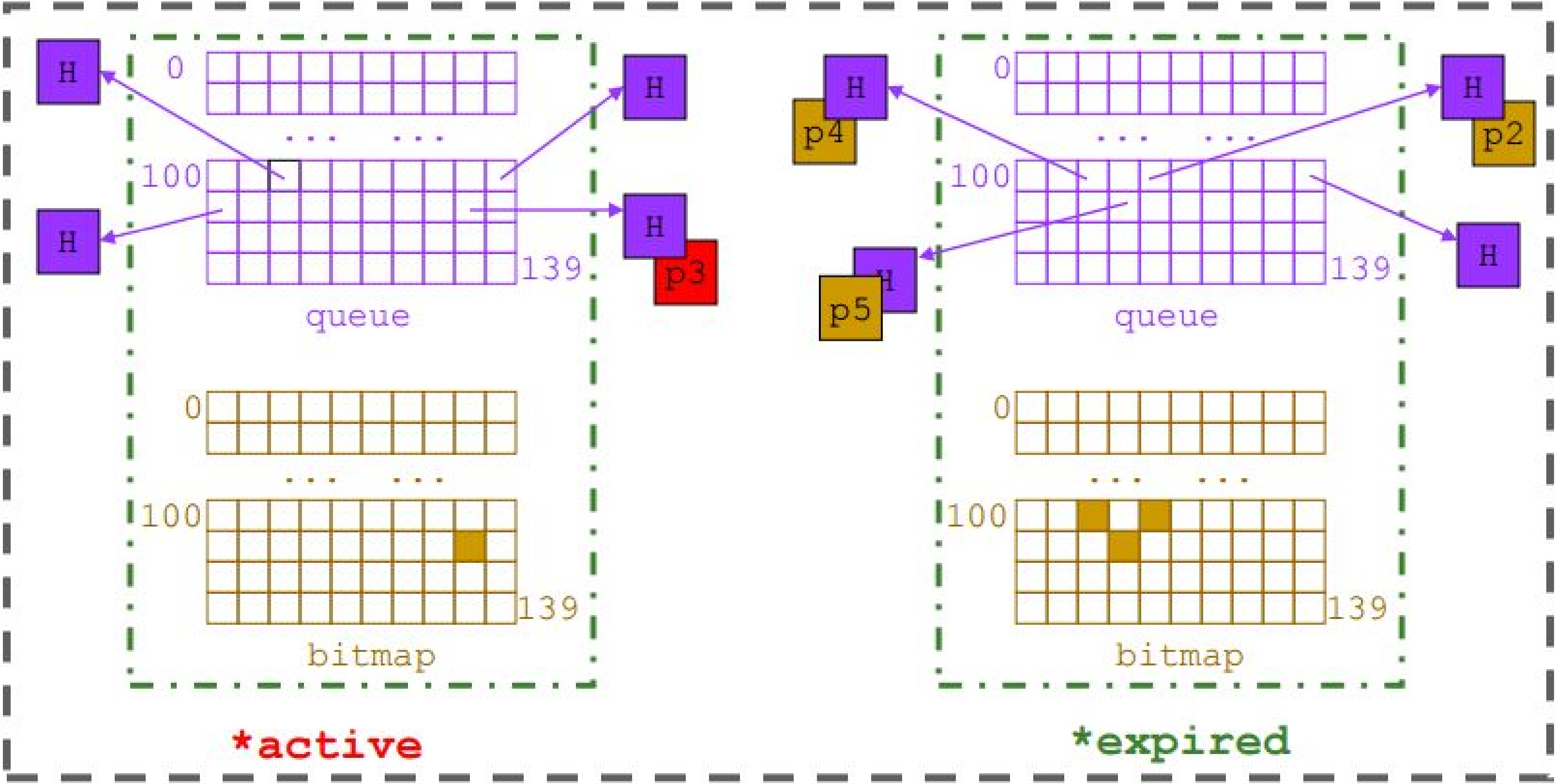


Escalonador O(1) no Linux

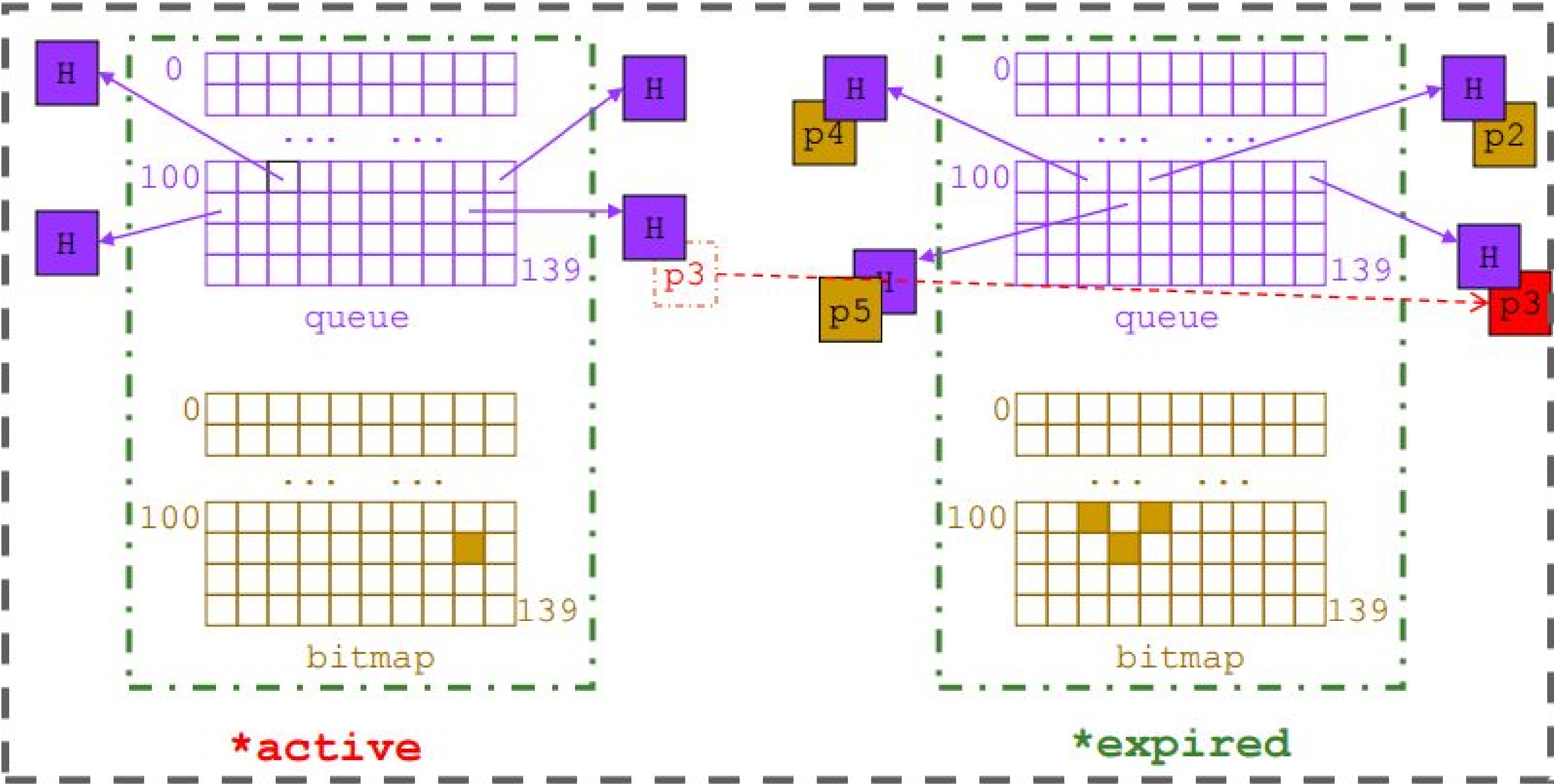
P1 finalizou



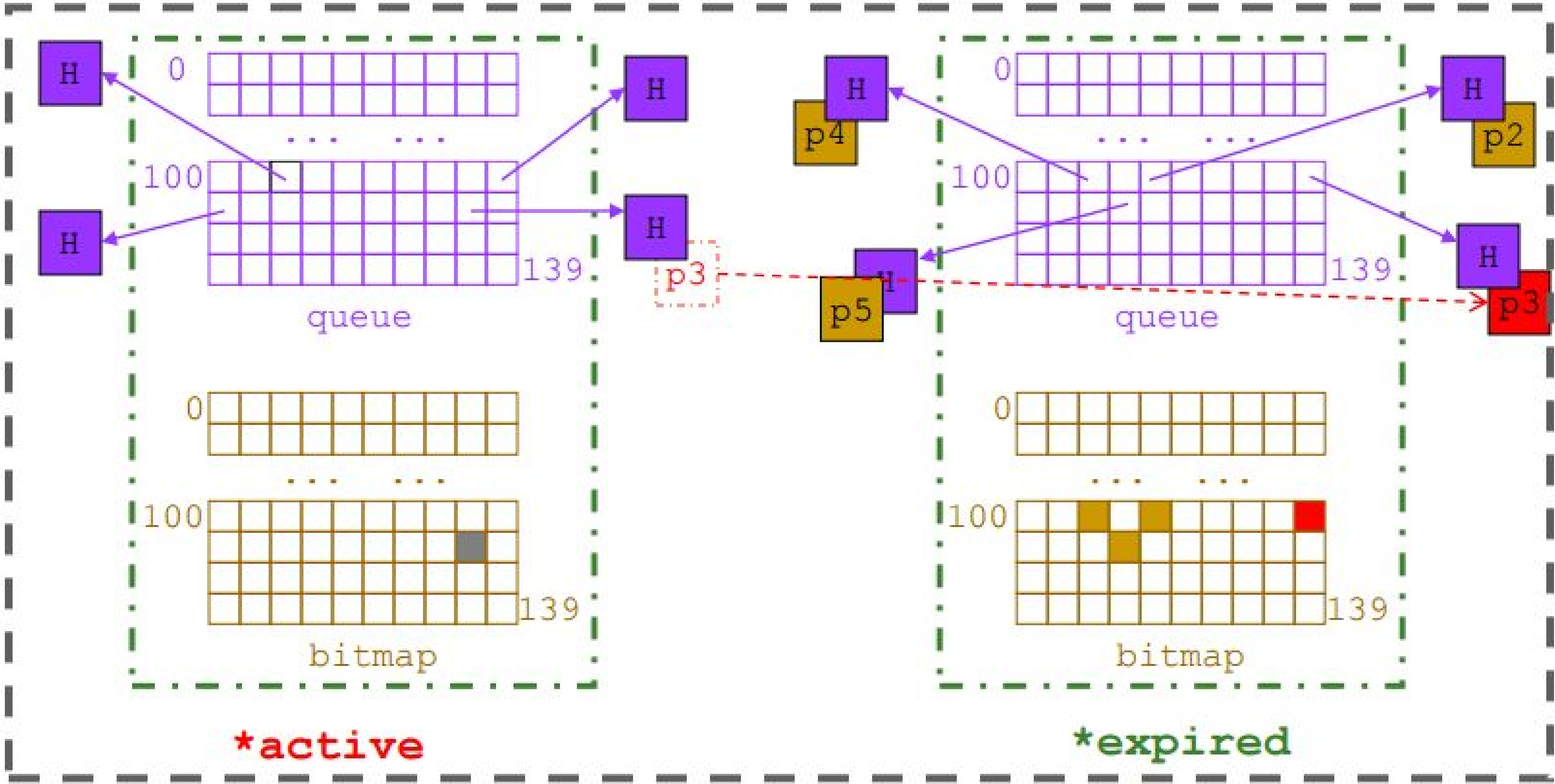
Escalonador O(1) no Linux



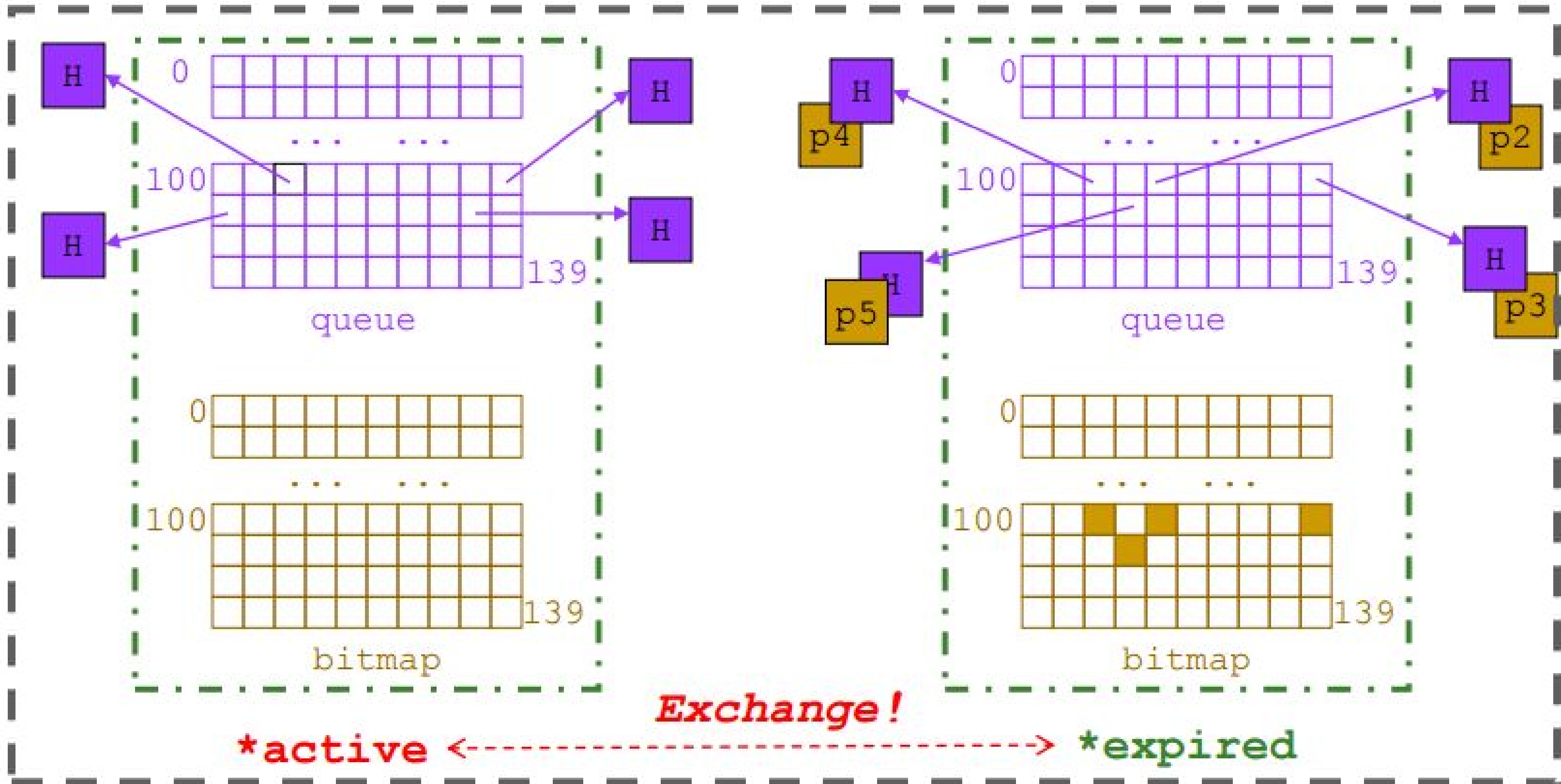
Escalonador O(1) no Linux



Escalonador O(1) no Linux



Escalonador O(1) no Linux



Não utilizamos mais o O(1) scheduler. Quais são os motivos?

Linux Completely Fair Scheduler (CFS)

- Mais simples de manter
  - Código mais simples do que O(1)

- Mais justo
  - Cálculos de prioridade no O(1) estavam levando para situações de degradação em tarefas interativas. Recebiam menos CPU

- Ainda mantemos os 3 grupos de execução:
  - 1 Normal
    - CFS
  - 2 Real Time
    - Round Robin
    - FIFO

Virtual Runtime

- A prioridade de uma tarefa depende do seu virtual runtime
- "In practice, the virtual runtime of a task is its actual runtime normalized to the total number of running tasks."
  - https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git/tree/Documentation/scheduler/sched-design-CFS.txt
- Computado quando a tarefa executa e dorme
  - Running e Waiting
- Descontos/Ganhos dependendo do niceness
  - Pequenos ajustes de prioridade
  - Comando nice no Linux

Árvore Balanceada

https://www.ibm.com/developerworks/library/l-completely-fair-scheduler/

27

Nodes represent
sched_entity(s)
indexed by their
virtual runtime

19
34
7
25
31
65
NIL
NIL
NIL
NIL
2
49
98
NIL
NIL
NIL
NIL
NIL
NIL

Virtual runtime

Most need of CPU

Least need of CPU

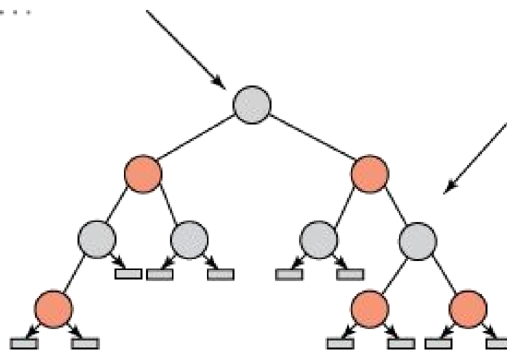
Árvore Balanceada

https://www.ibm.com/developerworks/library/l-completely-fair-scheduler/

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
};
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
};
```

```
struct ofs_rq {
   ...
   struct rb_root tasks_timeline;
   ...
};
```



```
struct rb_node {
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
```

Qual o custo de selecionar a tarefa com menor vruntime? Como isto se compara ao O(1)?