Gerência de Recursos

Sérgio Campos

Gerência de Recursos

Quais recursos ?

- Quaisquer recursos compartilhados.
  - rede;
  - disco;
  - memória;
  - ...

Acesso exclusivo e não interrompíveis.

Gerência de Recursos

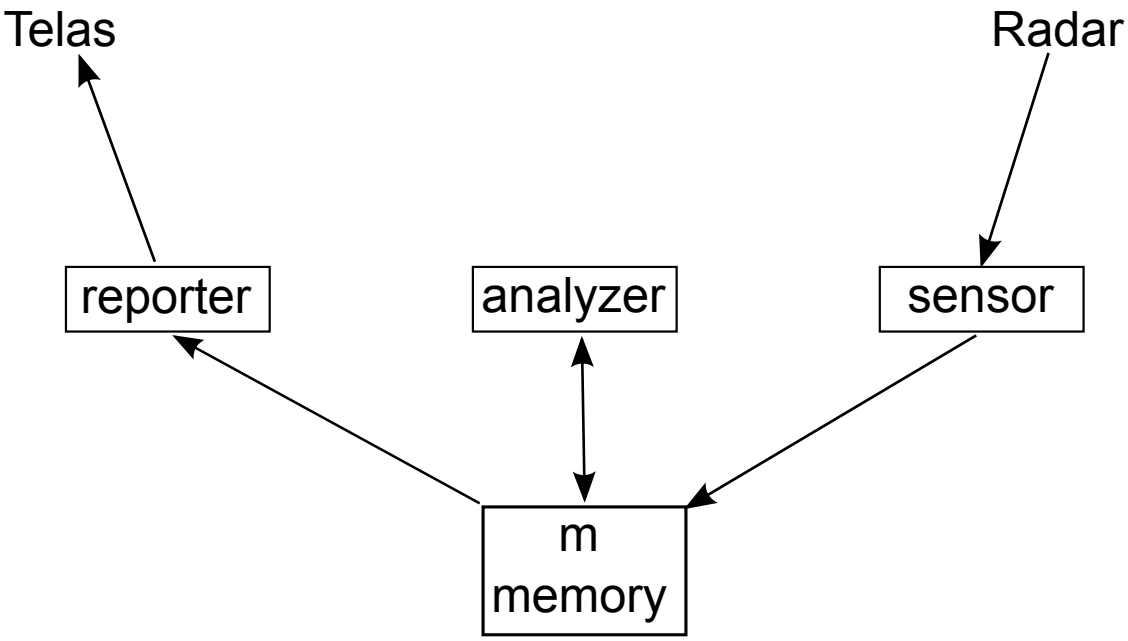
Uso:

```
while (true) {
   ...;
   lock(R1);
   use(R1);
   unlock(R1);
};
```

Porque compartilhamento de recursos causa problemas em tempo real ?

Priority Inversion

Considere um sistema de controle de tráfego aéreo:

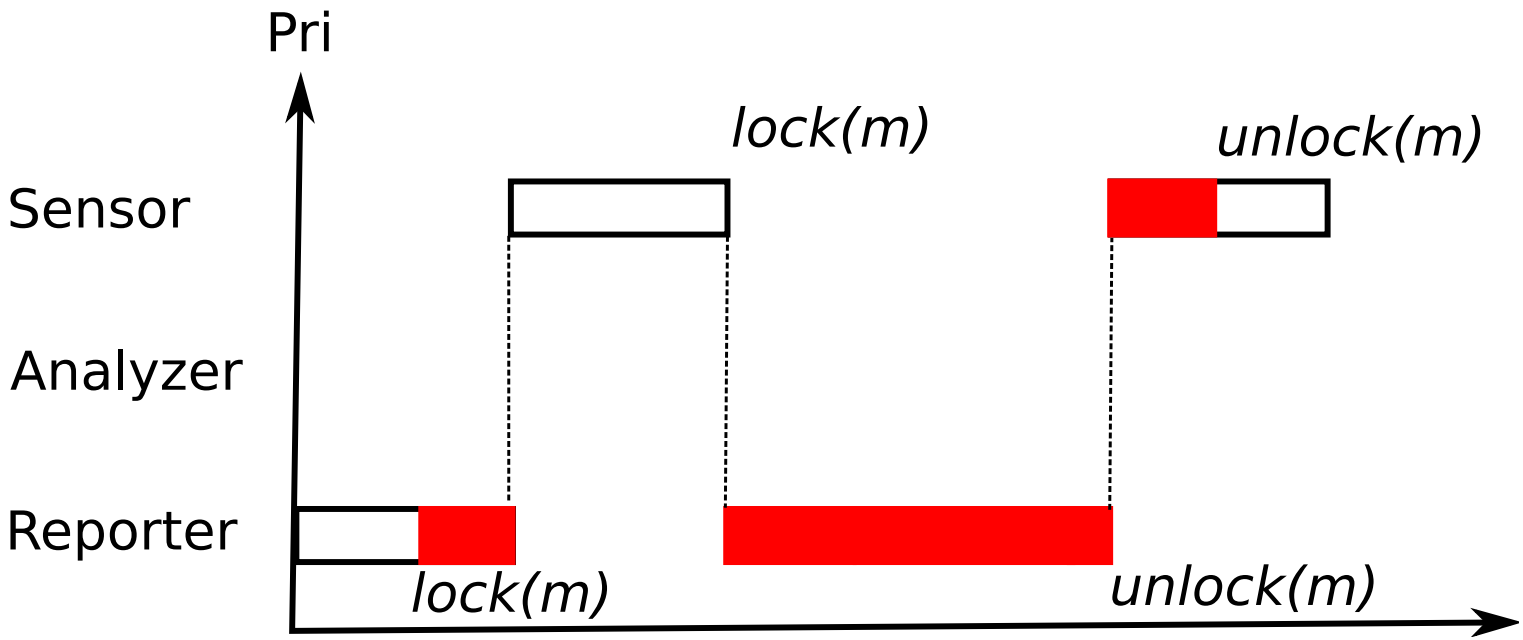


Os processos se comunicam através de memória compartilhada;
Ordem de prioridades:

- Sensor (mais importante);
- Analisador;
- Repórter;

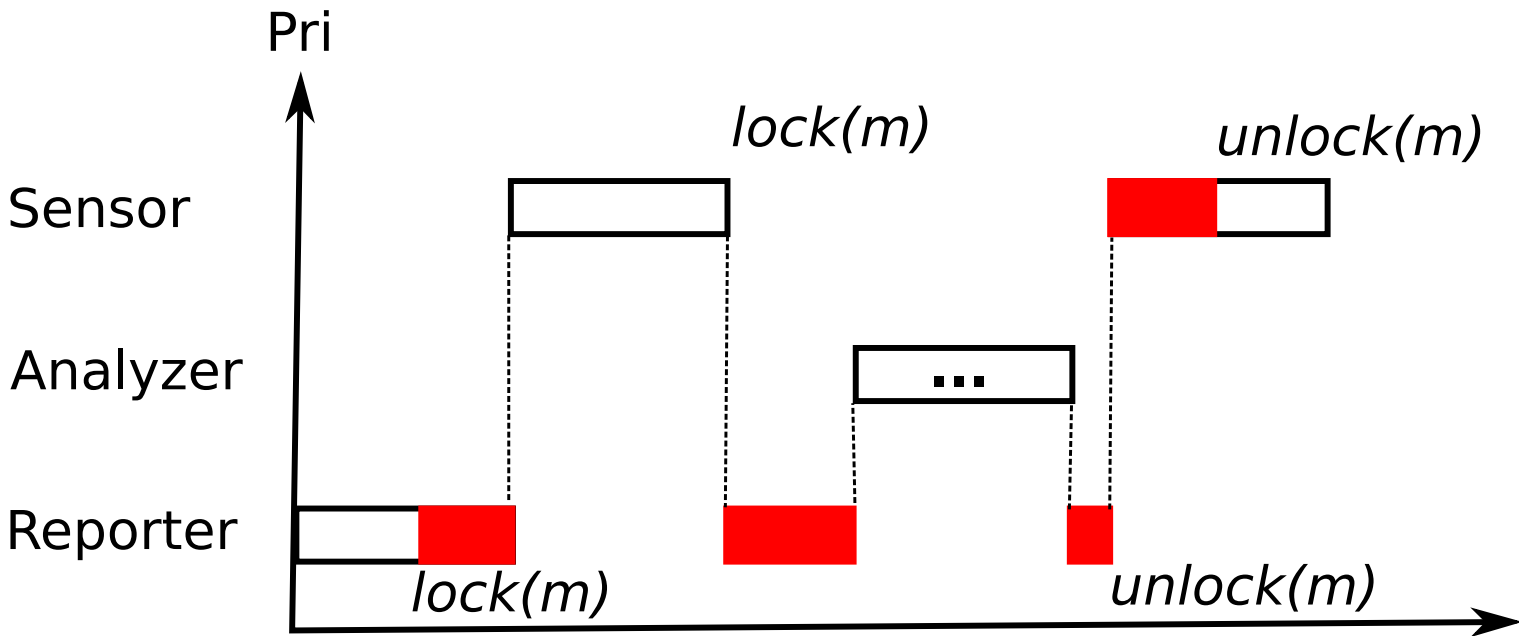
Inversão de Prioridades

O sensor não pode ser bloqueado nunca!
Mas os eventos abaixo levam à inversão de prioridades:



Inversão de Prioridades

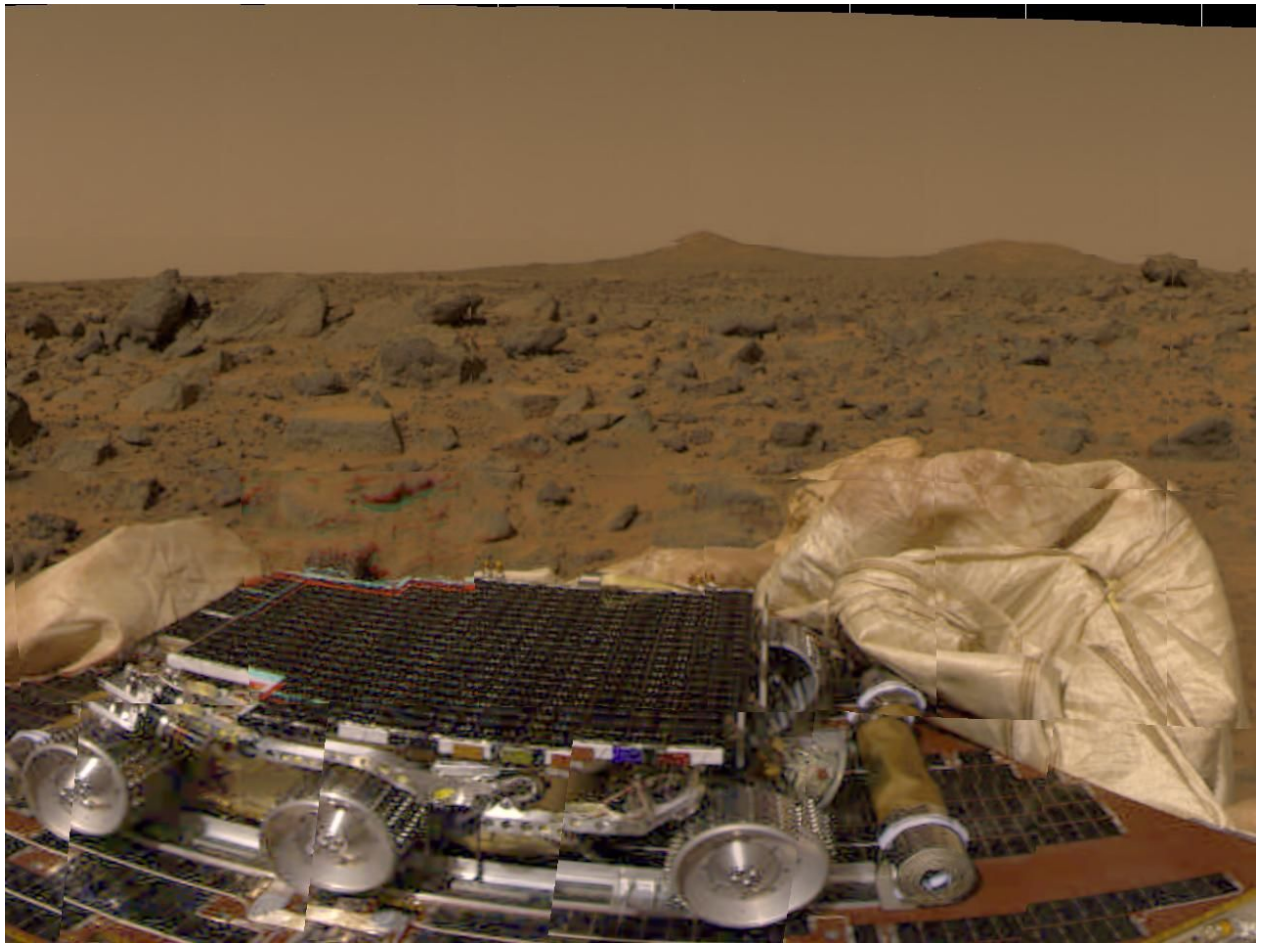
Mas e se o analisador decidir executar ?



O analisador não vê o sensor, e assim bloqueia o reporter *indefinidamente*!

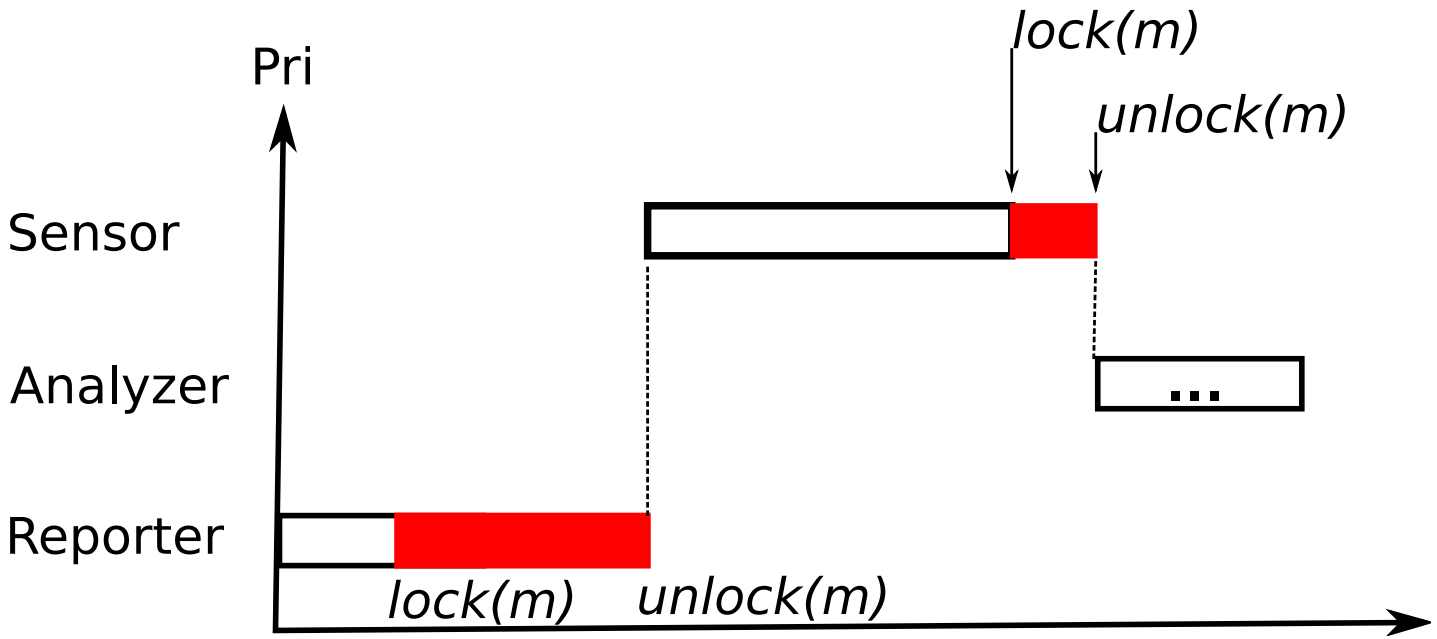
Inversão de Prioridades

Aconteceu! Em Marte!!!
NASA Pathfinder, 1997:



Solução: mutex não interrompível

Todas as regiões críticas executam em prioridade máxima.



- Simples;
- Mas ineficiente: jobs podem ser bloqueados por qualquer job menos prioritário, mesmo que não haja conflito.

Herança de Prioridades Básica

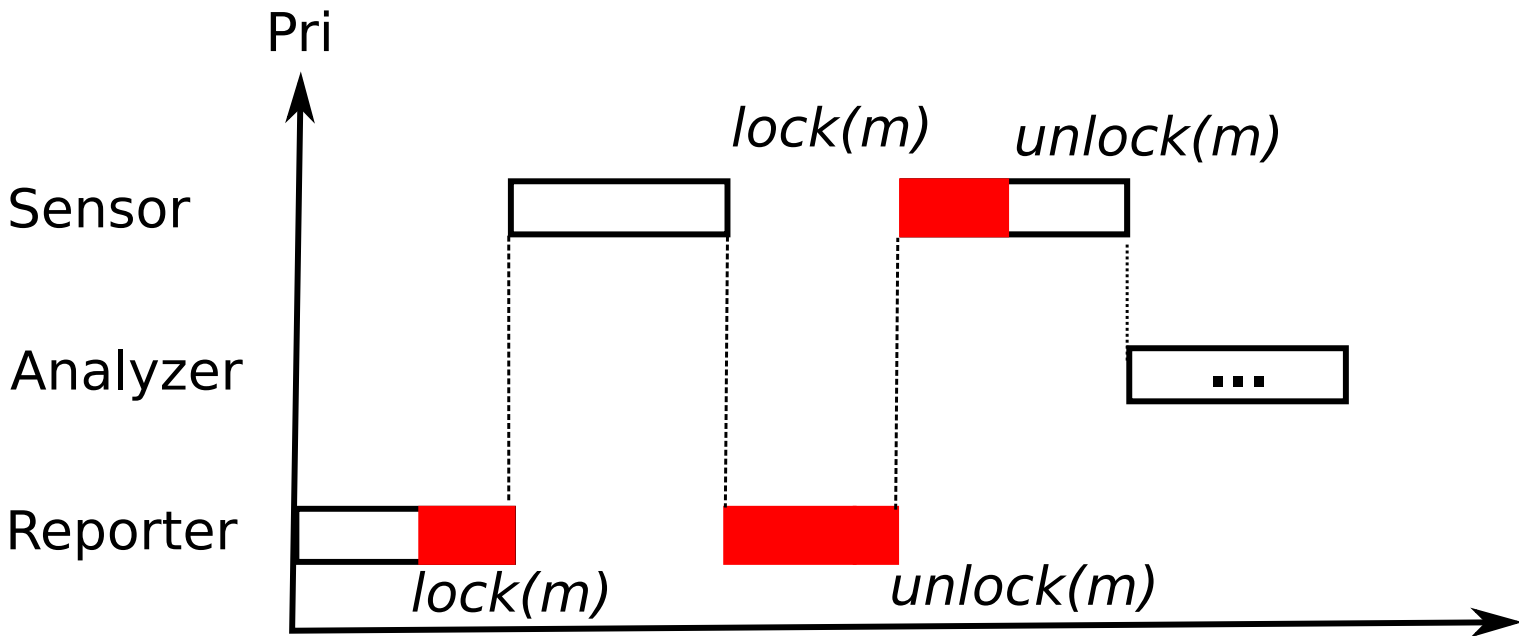
Jobs podem mudar de prioridade:

- Prioridade atribuída: original;
- Prioridade atual: pode ser maior.

Protocolo:

- **Escalonamento**: a execução é feita na prioridade atribuída.
- **Alocação de recursos**:
  - `lock(m)` aloca `m` se estiver livre;
  - `lock(m)` bloqueia se `m` estiver ocupado.
- **Herança de prioridades**: Se `lock(m)` bloquear o job J_h (pri atual h) e o job J_l (pri atual l) estiver de posse de m:
  - Se $h > l$ então a prioridade atual de J_l passa a ser h;
  - Quando `m` for liberado, J_l volta a ter prioridade l.

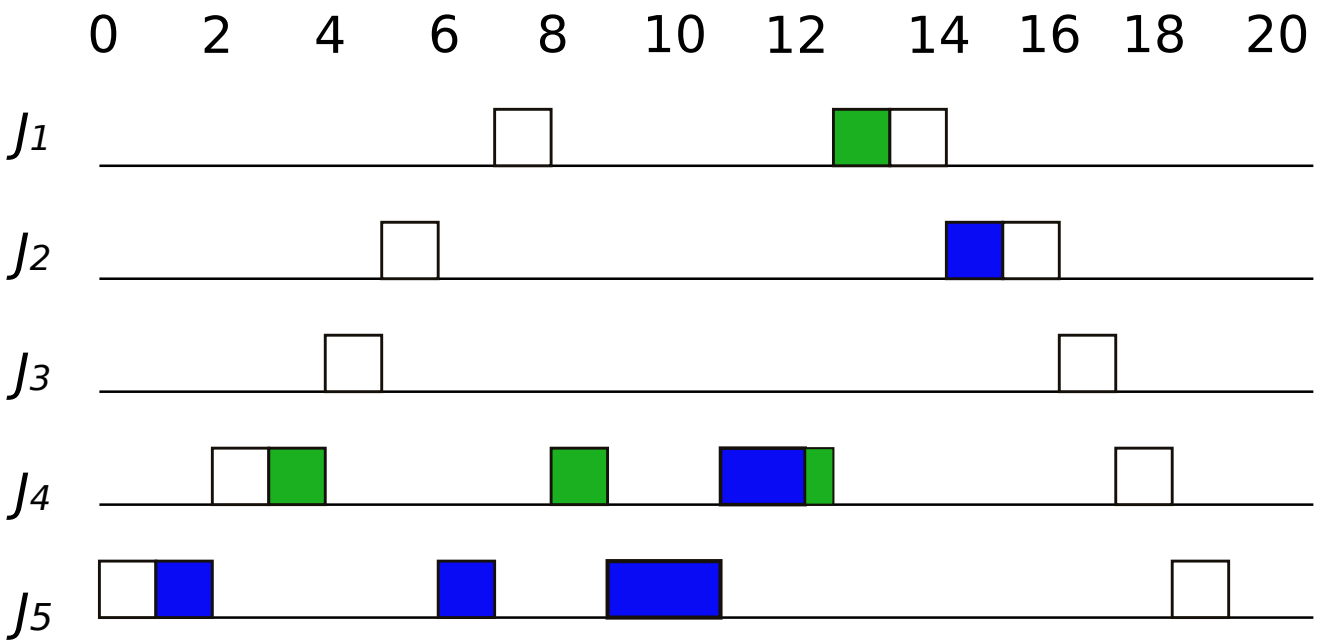
Herança de Prioridades Básica



Herança básica:

- Não impede deadlocks;
- Não acaba com inversão de prioridades, somente com a ilimitada;
- Um job com k recursos pode ser interrompido k vezes.

Priority Inheritance Protocol



Protocolo de Priority-Ceiling

Jobs também podem mudar de prioridades.

- Prioridades atribuídas são estáticas;
- Todas as necessidades de recursos são conhecidas.

O teto de prioridades $\Pi(R_i)$ de um recurso é:

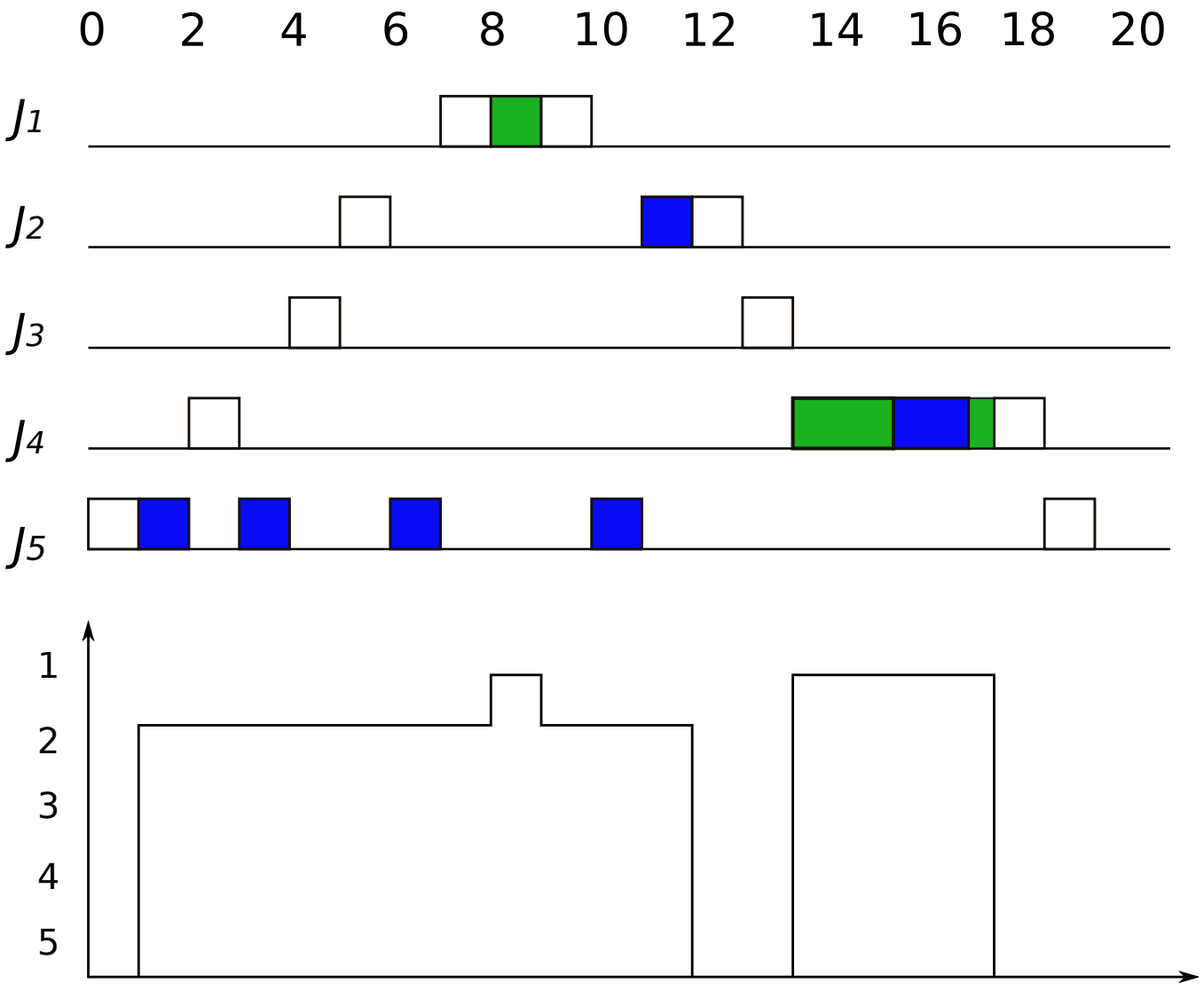
- A maior prioridade dentre todos os jobs que podem requisitar o recurso.
- O teto de prioridades do sistema $\Pi(t)$ é:
  - O maior $\Pi(R_i)$ dentre todos os R_i em uso.

Protocolo de Priority-Ceiling

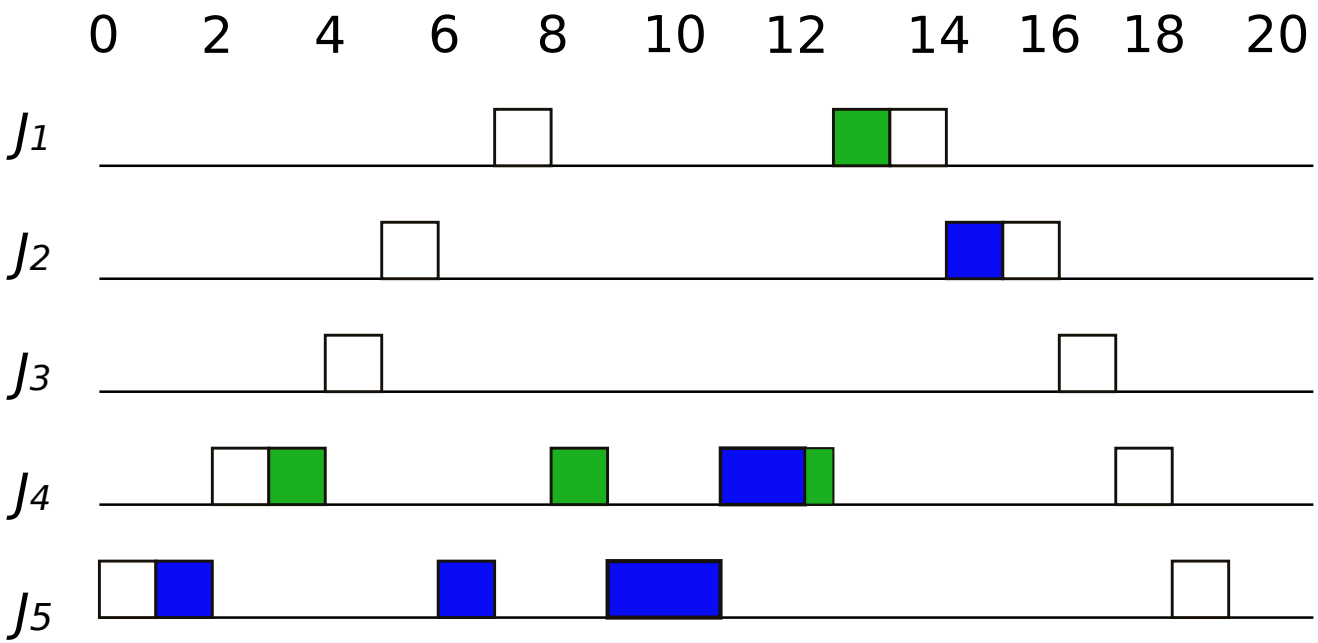
Protocolo:

- **Escalonamento**: a execução é feita na prioridade atribuída.
- **Alocação de recursos**:
  - `lock(m)` bloqueia se `m` estiver ocupado.
  - prioridade atual $>$ $\Pi(t)$, `lock(m)` ok; senão
  - job atual usa o recurso com prioridade $\Pi(t)$, ok.
  - senão, bloqueia.
- **Herança de prioridades**: Se `lock(m)` bloquear o job J_h (pri atual h) mesmo que indiretamente e o job J_l (pri atual l) tiver posse de `m`:
  - Se $h > l$ então a prioridade atual de J_l passa a ser h;
  - Quando `m` for liberado, J_l volta a ter prioridade l.

Priority Ceiling Protocol



Priority Inheritance Protocol



Protocolo de Priority-Ceiling

Este protocolo garante ausência de deadlocks!

- O teto de prioridades de um recurso é sua importância;
- A importância de quem está executando somente cresce:
  - Porque um novo pedido vem que alguém mais importante;
  - Ou porque o novo pedido vem de quem já é importante.
- Em algum instante quem executa é tão importante que ninguém o interrompe mais.

Mas o protocolo bloqueia jobs mesmo quando seus recursos estão livres. Isto não os atrasa ?

- O bloqueio máximo é de uma seção crítica somente (melhor que o anterior);
- Porque uma vez que o teto de prioridades sobe jobs menos importantes não tomam recursos mesmo que pudessem.