

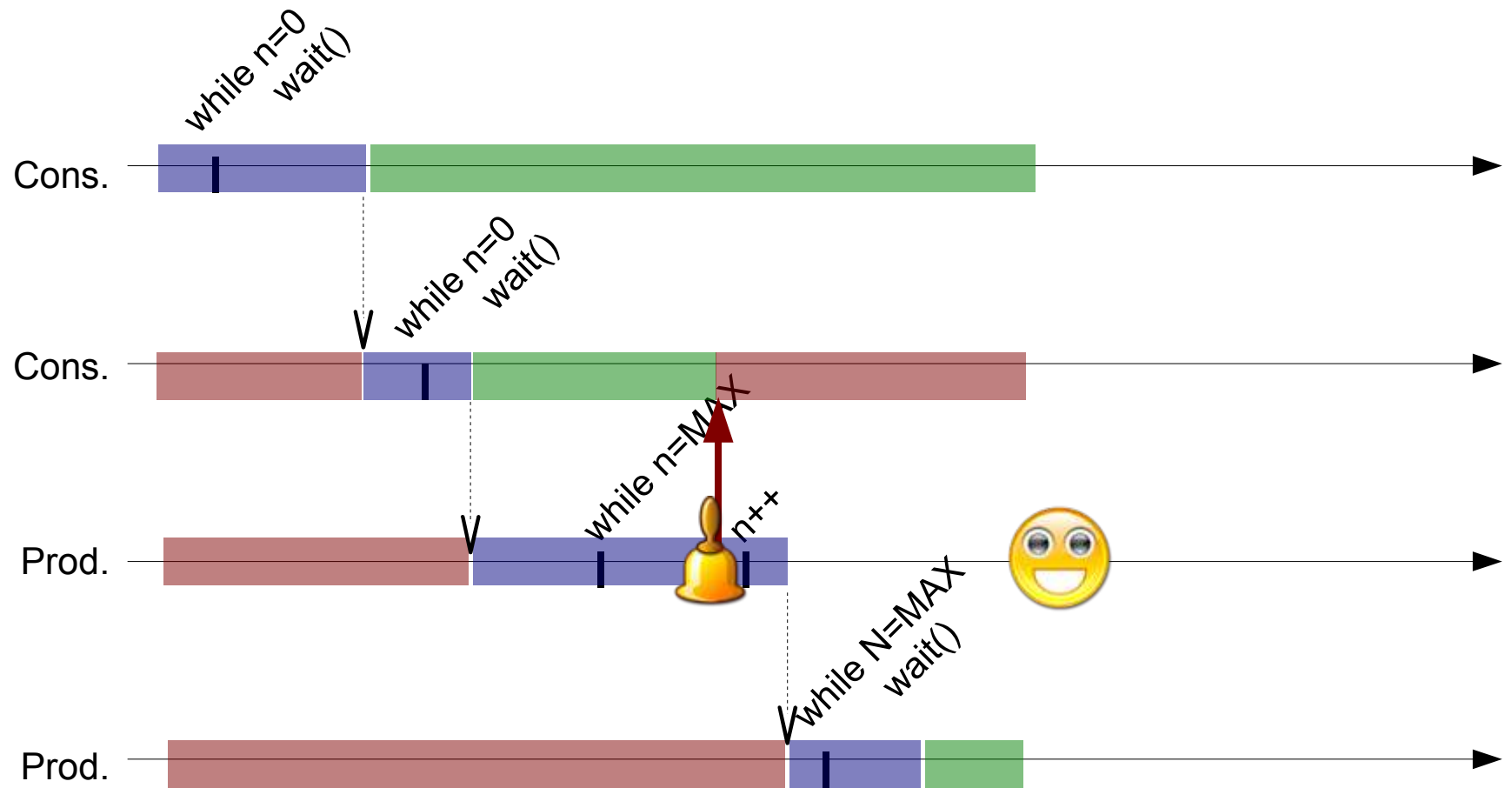
notify() vs notifyAll()


- Será possível usar notify() em vez de notifyAll() na implementação de *bounded buffer* com apenas um variável de condição?
- Intuitivamente a resposta é sim:
 - O *buffer* não pode estar simultaneamente vazio e cheio!!
 - Não podem portanto estar simultaneamente bloqueados consumidores e produtores
 - O notify() acorda sempre a actividade certa

Exemplo: cenário


- Considere um *buffer* com tamanho máximo 1 e inicialmente vazio
- Considere a chegada em simultâneo de 2 consumidores e 2 produtores
- Pode ser construído um caso semelhante com *buffers* de qualquer tamanho aumentando também o número de *threads*


Exemplo





 executando na secção critica

 bloqueado no mutex

 bloqueado na variável de condição


 terminação com sucesso

 invocação de notify()

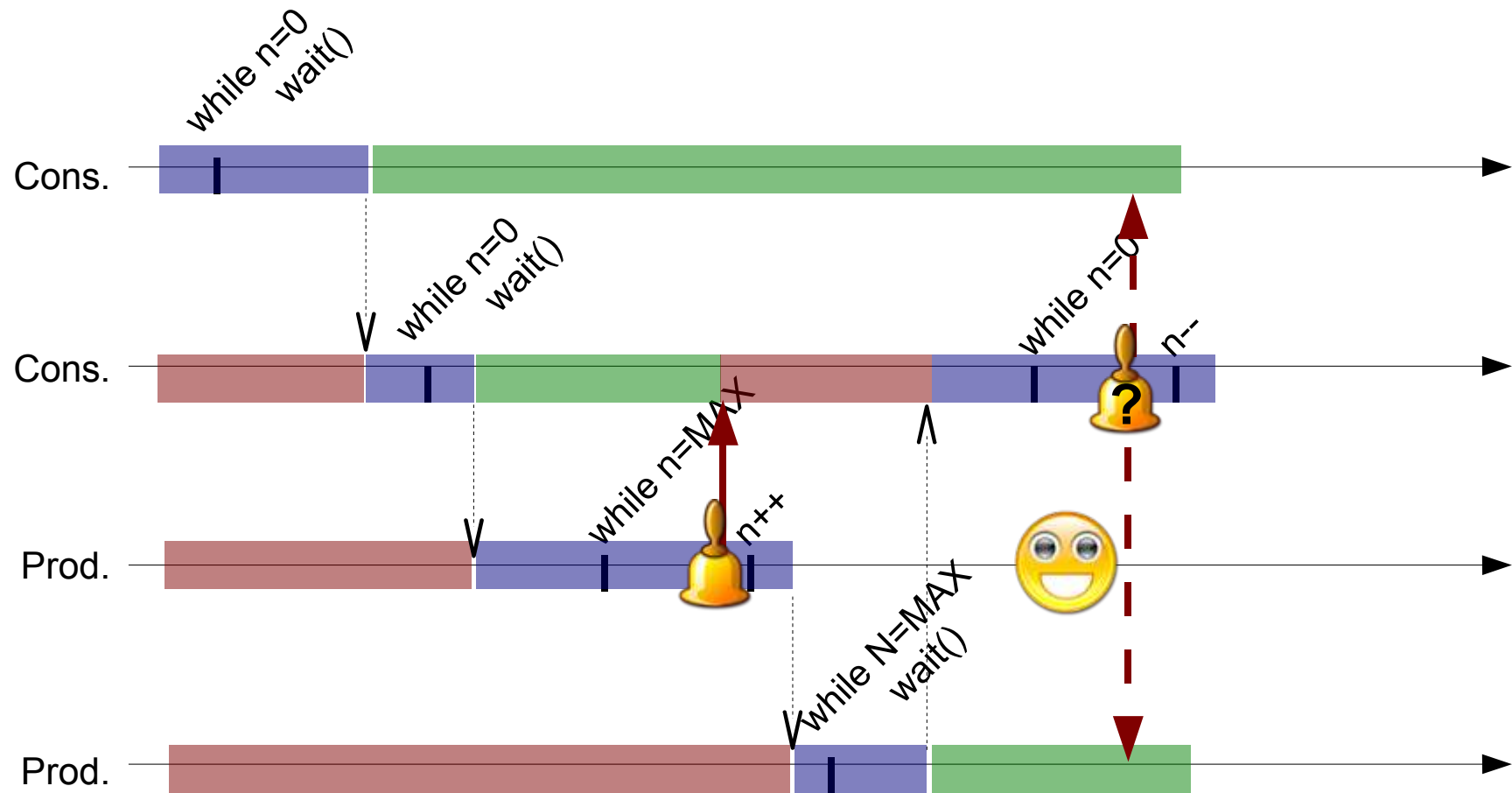
 bloqueado em deadlock


 thread

 resultado do notify()


 resultado da libertação do mutex


Exemplo





 executando na secção critica

 bloqueado no mutex

 bloqueado na variável de condição


 terminação com sucesso

 invocação de notify()

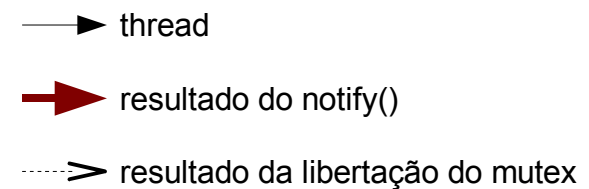
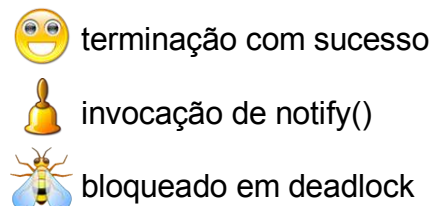
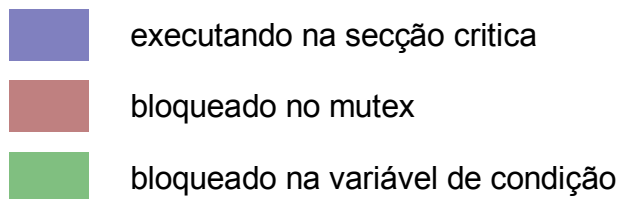
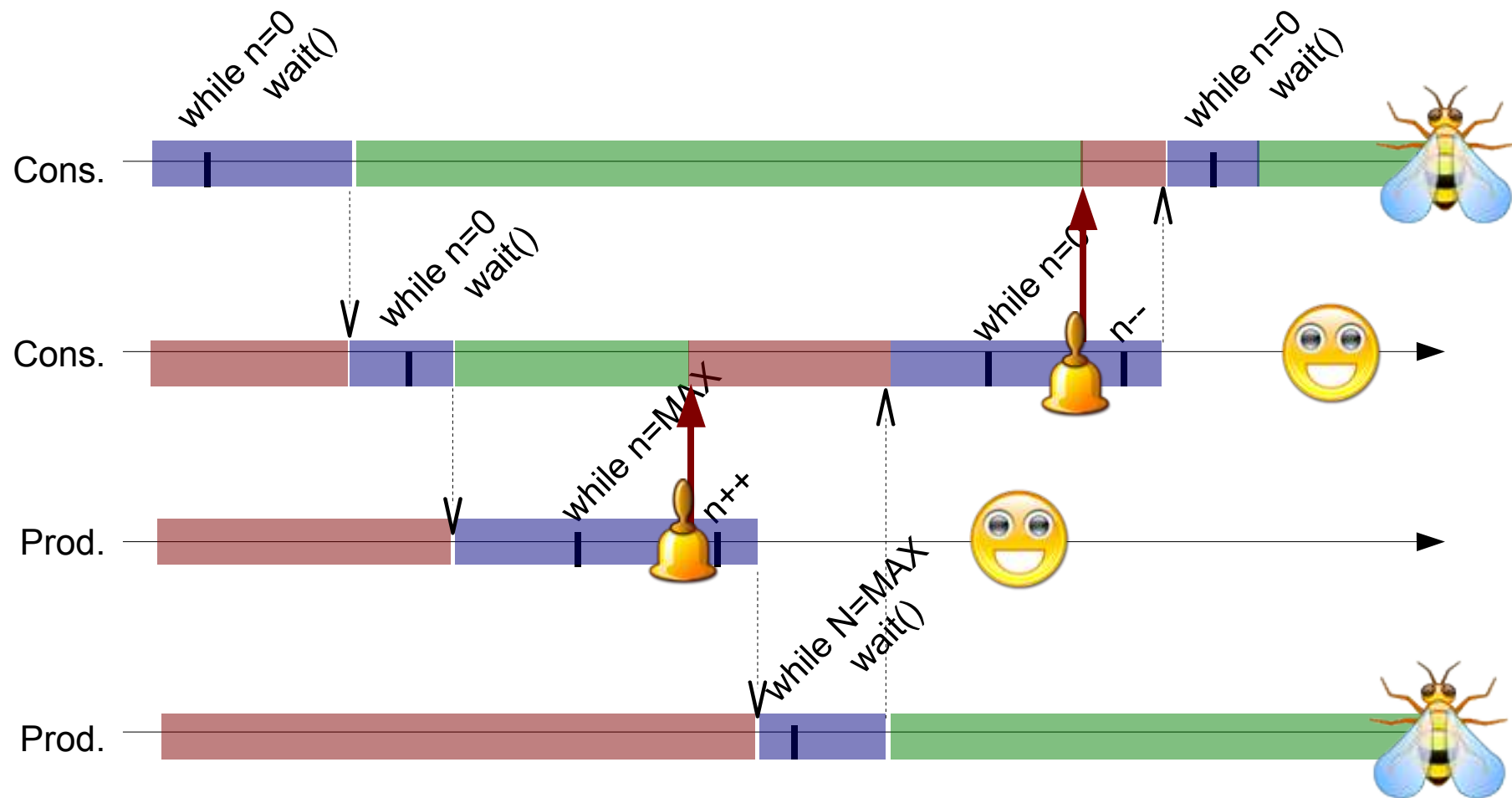
 bloqueado em deadlock

 thread

 resultado do notify()

 resultado da libertação do mutex

Exemplo: Deadlock!



Solução: notifyAll()

- Utilização de notifyAll() sempre que a variável de condição é usada com diferentes condições:
 - Neste caso, o código da condição é diferente
 - Em alguns casos, o código é o mesmo mas depende de parâmetros diferentes
- Impacto no desempenho quando existe contenção (i.e. uma lista de espera longa na condição)

Solução: j.u.c.locks

- Utilização das primitivas de sincronização em `java.util.concurrent.locks` que permitem múltiplas variáveis de condição associadas ao mesmo mutex
- Programação dificultada:
 - Declaração explícita de mutex e condições
 - Necessidade de emparelhar `lock()` e `unlock()`
 - Cuidado especial com exceções (cláusula `finally`)

Monitores vs j.u.c.locks

- Declaração implícita do mutex
- `synchronized ... {`
 ...
}
- Declaração implícita da variável de condição
- `wait();`
- `notify();`
- `notifyAll();`
- `Lock l=new ReentrantLock();`
- `l.lock();`
 ...
 `l.unlock();`
- `Condition c=l.newCondition();`
- `c.await();`
- `c.signal();`
- `c.signalAll();`

j.u.c.locks: Excepções

- Utilização correcta (da documentação Java):

```
Lock l = ...;  
l.lock();  
try {  
    // access the resource protected by this lock  
} finally {  
    l.unlock();  
}
```