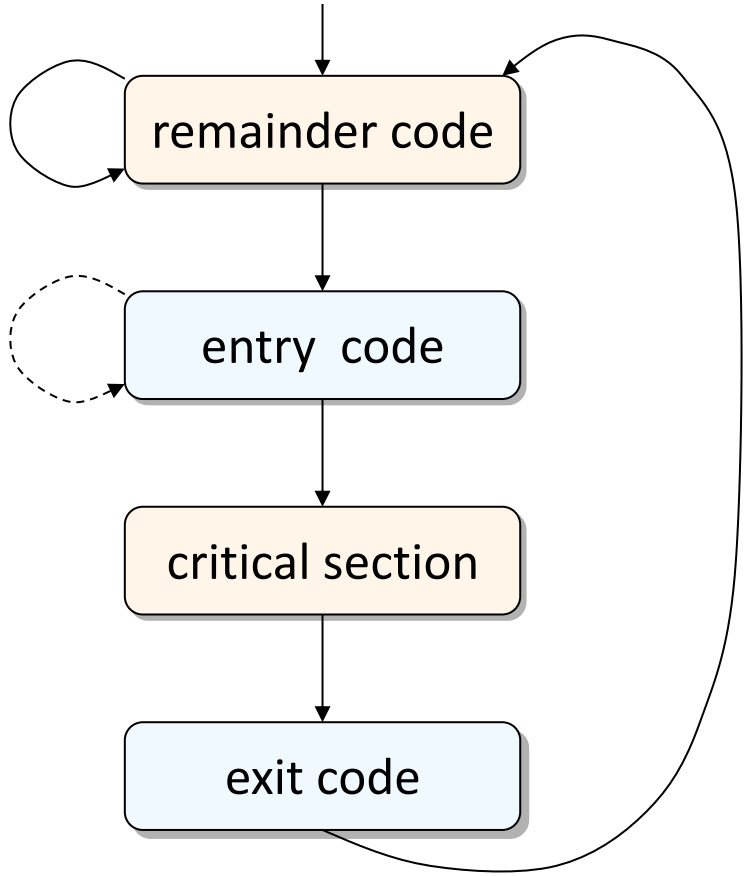
Mutual exclusion algorithms with constant RMR complexity and wait-free exit code

Rotem Dvir

Joint work with Gadi Taubenfeld

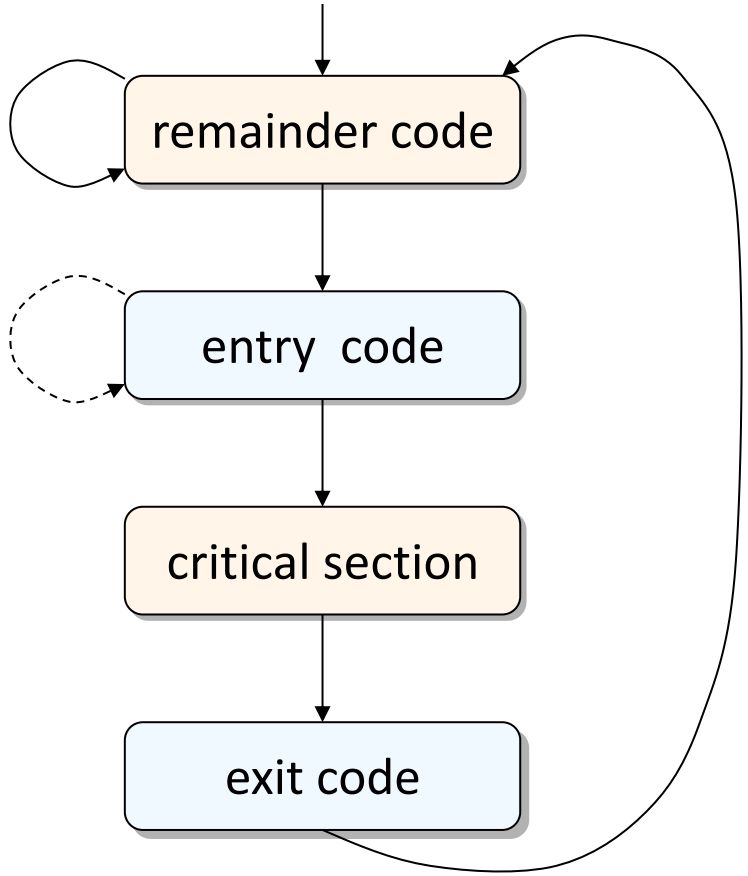
Mutual Exclusion

The Problem: design the entry and exit code in a way that guarantees that the mutual exclusion and deadlock-freedom properties are satisfied.



Mutual Exclusion

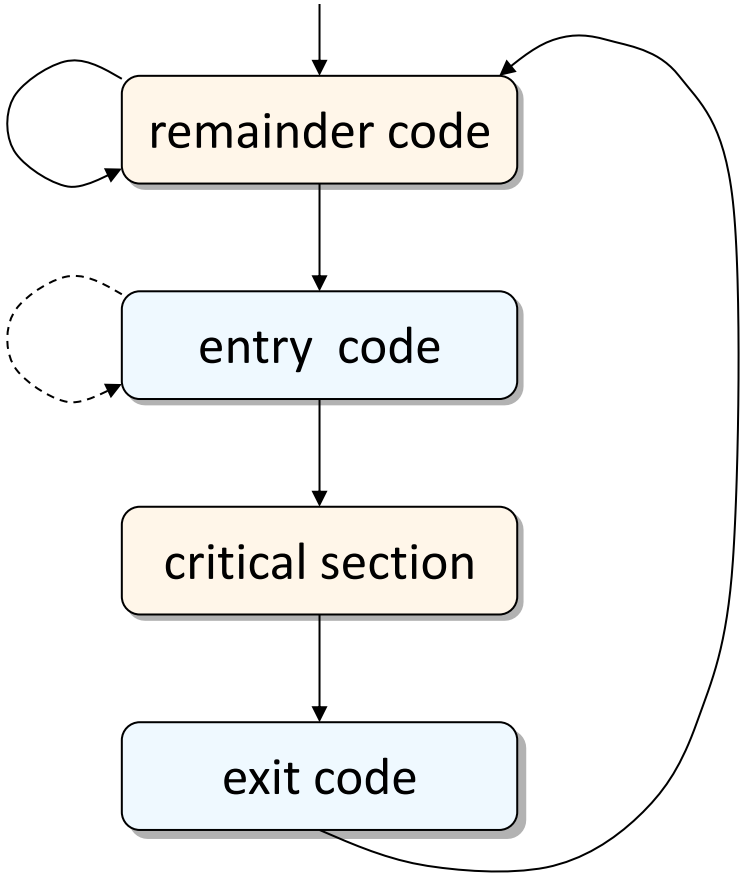
- Mutual Exclusion: No two processes are in their critical sections at the same time.



Mutual Exclusion

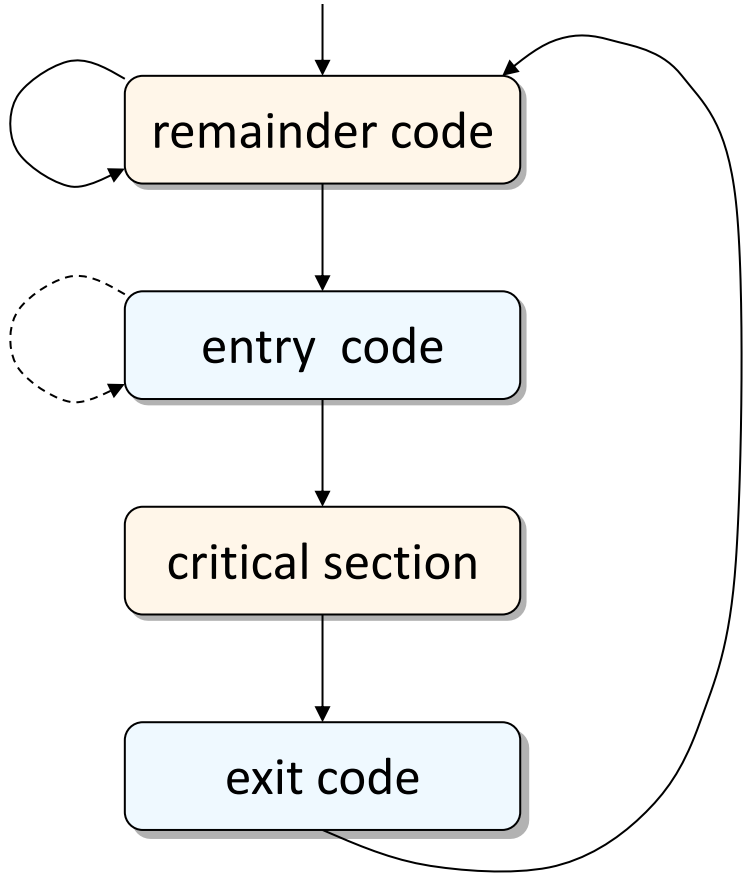
- Mutual Exclusion: No two processes are in their critical sections at the same time.

- Deadlock-Freedom: If a process is trying to enter its critical section, then some process, not necessarily the same one, eventually enters its critical section.



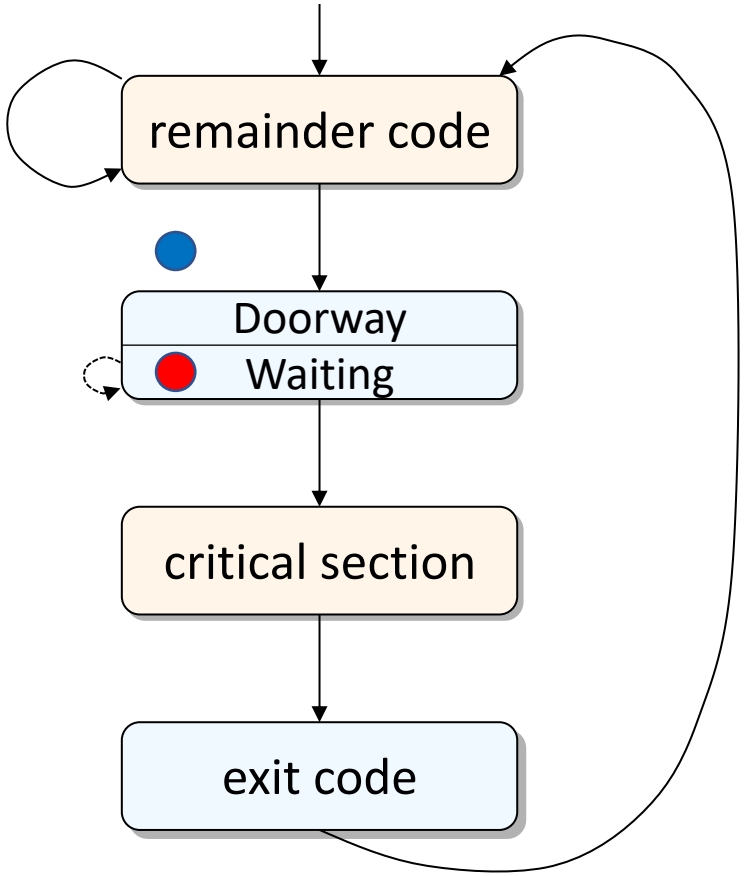
Mutual Exclusion

- Mutual Exclusion: No two processes are in their critical sections at the same time.
- Deadlock-Freedom: If a process is trying to enter its critical section, then some process, not necessarily the same one, eventually enters its critical section.
- Starvation-Freedom: If a process is trying to enter its critical section, then this process must eventually enter its critical section.



FIFO

FIFO (First-in-first-out): A beginning process cannot execute its critical section before a waiting process executes its critical section.



Memory References

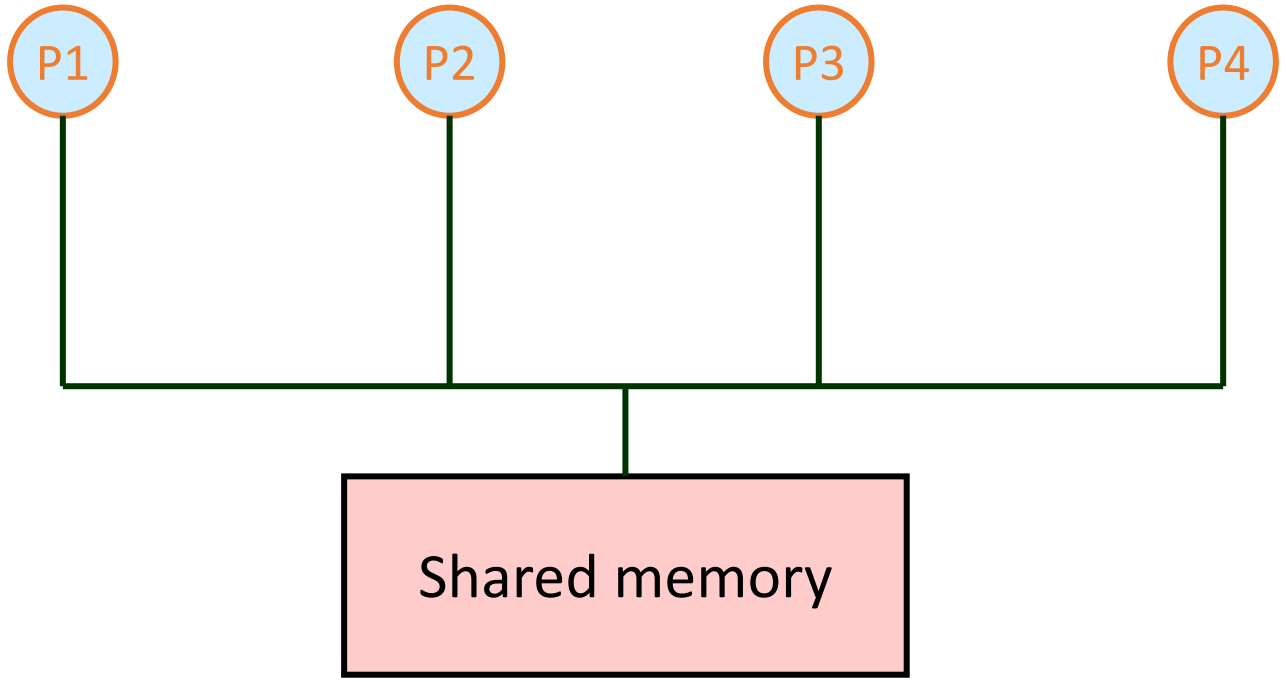
- Remote reference: slow
- Local reference: fast

Memory References

- Remote reference: slow
- Local reference: fast

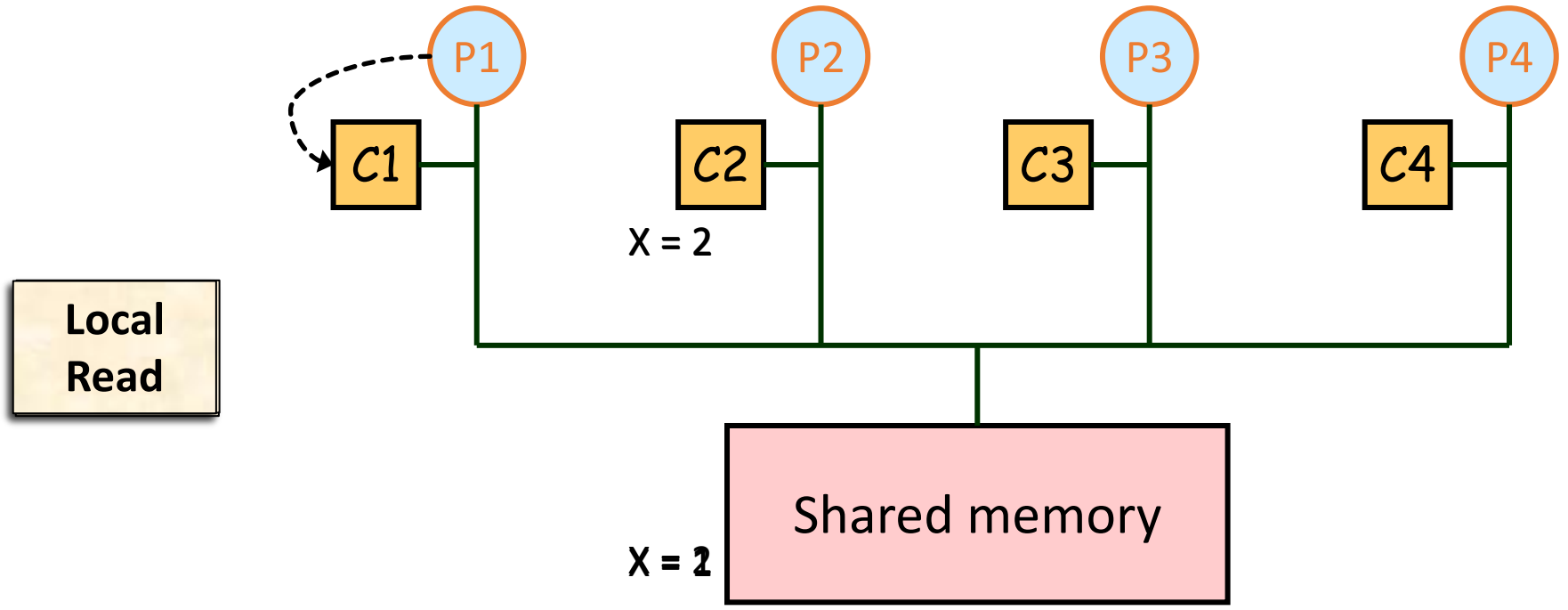
Depends on the memory model

Memory Models: Simple Shared Memory



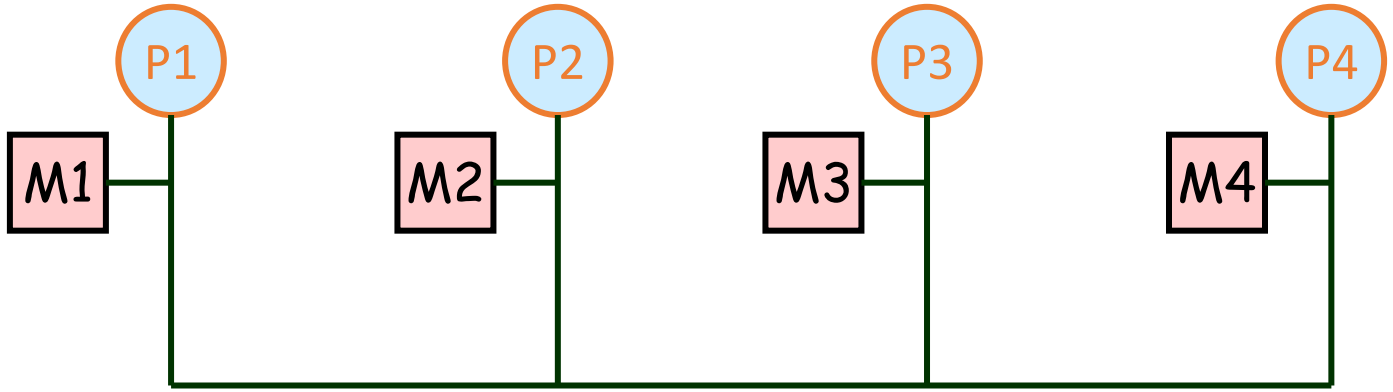
All memory accesses are remote

Memory Models: Coherent Caching



Local and remote memory accesses

Memory Models: Distributed Shared Memory



Local and remote memory accesses

Remote Memory Reference Complexity

RMR Complexity: The maximum number of remote memory references that a process may need to perform in its entry and exit sections.

Algorithms Properties

Algorithms Properties

Wait-free exit codes

Algorithms Properties

- ✓ Wait-free exit codes
- ✓ Satisfy FIFO fairness

Algorithms Properties

- ✓ Wait-free exit codes
- ✓ Satisfy FIFO fairness
- ✓ Constant RMR complexity in CC and DSM models

Algorithms Properties

- ✓ Wait-free exit codes
- ✓ Satisfy FIFO fairness
- ✓ Constant RMR complexity in CC and DSM models
- ✓ Not assumed that the number of processes is priori known

Algorithms Properties

- ✓ Wait-free exit codes
- ✓ Satisfy FIFO fairness
- ✓ Constant RMR complexity in CC and DSM models
- ✓ Not assumed that the number of processes is priori known
- ✓ Use only O(n) shared memory locations

Algorithms Properties

- ✓ Wait-free exit codes
- ✓ Satisfy FIFO fairness
- ✓ Constant RMR complexity in CC and DSM models
- ✓ Not assumed that the number of processes is priori known
- ✓ Use only O(n) shared memory locations
- ✓ Make no assumptions on what and how memory is allocated

Algorithms Properties

- ✓ Wait-free exit codes
- ✓ Satisfy FIFO fairness
- ✓ Constant RMR complexity in CC and DSM models
- ✓ Not assumed that the number of processes is priori known
- ✓ Use only O(n) shared memory locations
- ✓ Make no assumptions on what and how memory is allocated
- ✓ Instruction set: Read/Write, Fetch-And-Store, Compare-And-Swap

MCS Algorithm Properties


Wait-free exit code


Satisfy FIFO fairness


Constant RMR complexity in CC and DSM models


Not assumed that the number of processes is priori known

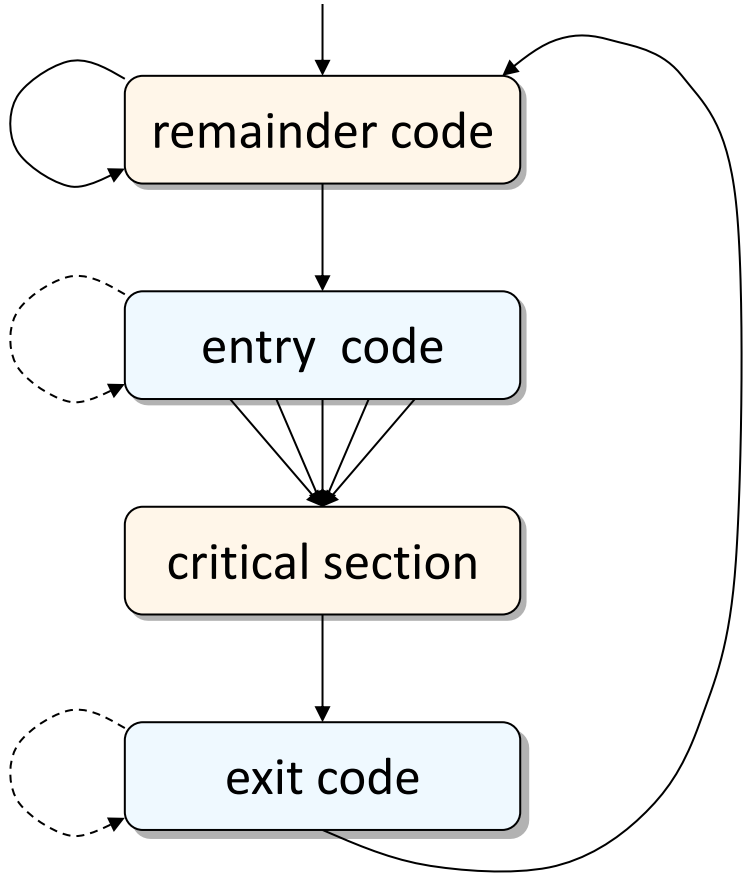

Use only O(n) shared memory locations


Make no assumptions on what and how memory is allocated


Instruction set: Read/Write, Fetch-And-Store, Compare-And-Swap

Wait-Free Vs. Not Wait-Free

- Exiting process may need to wait for entering process
- Particularly problematic in high contention scenarios

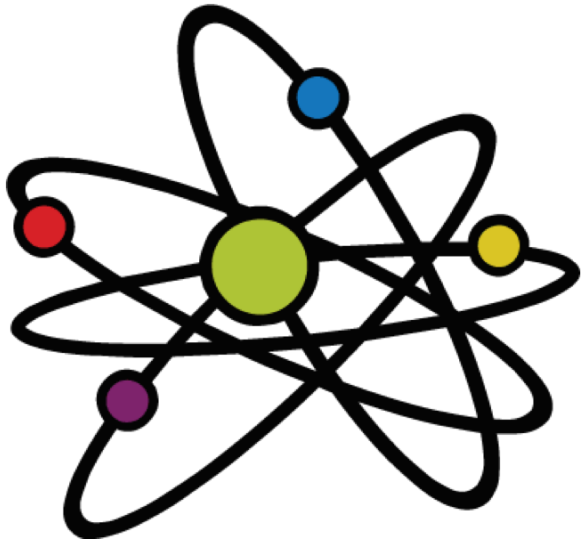


Space Complexity When Using L Locks

- First algorithm: O(Ln) shared memory locations
- Second algorithm: O(L+n) shared memory locations

Computational Model: Atomic Instruction Set

- Atomic read/write
- Compare-And-Swap (Destination, New, Old)
- Fetch-And-Store (Destination, Value)



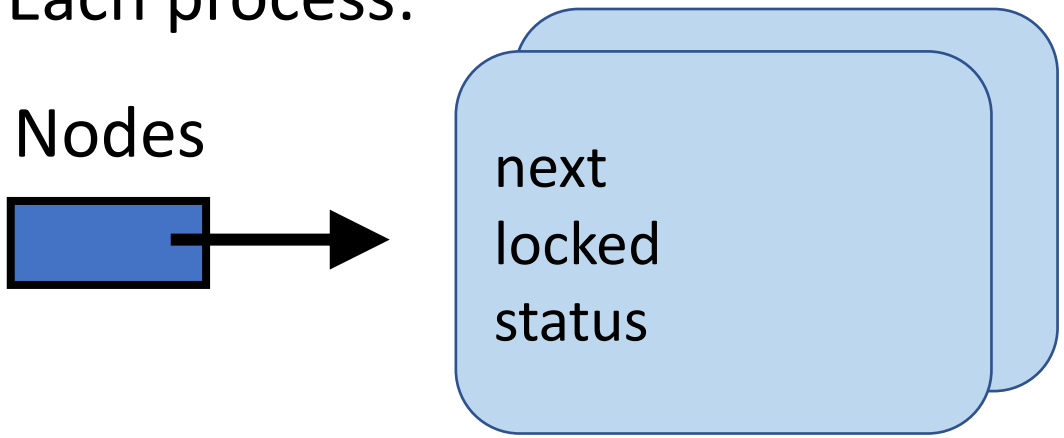
The First Algorithm

Initially

Tail

Each process:

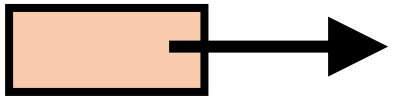
Nodes



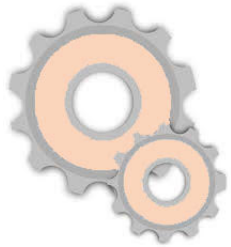
Process p Enters

Tail

p's node

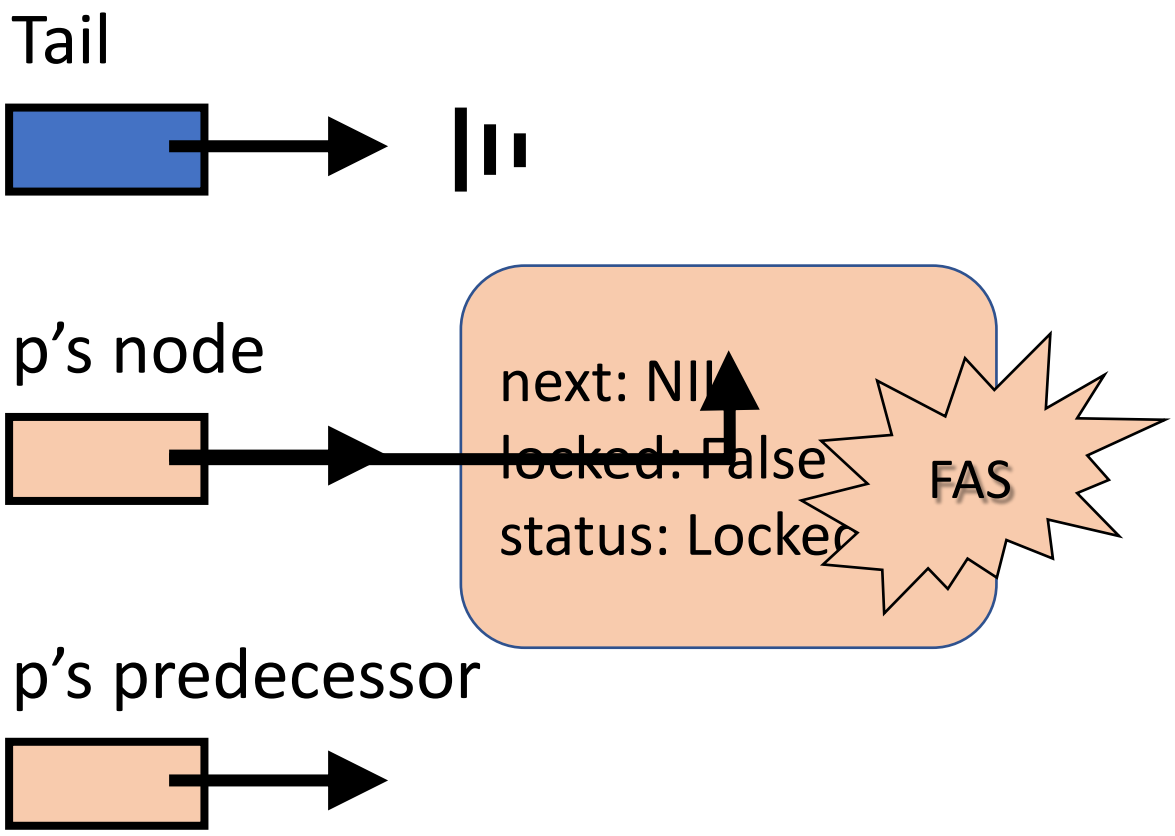


next: NIL
locked: False
status: Locked



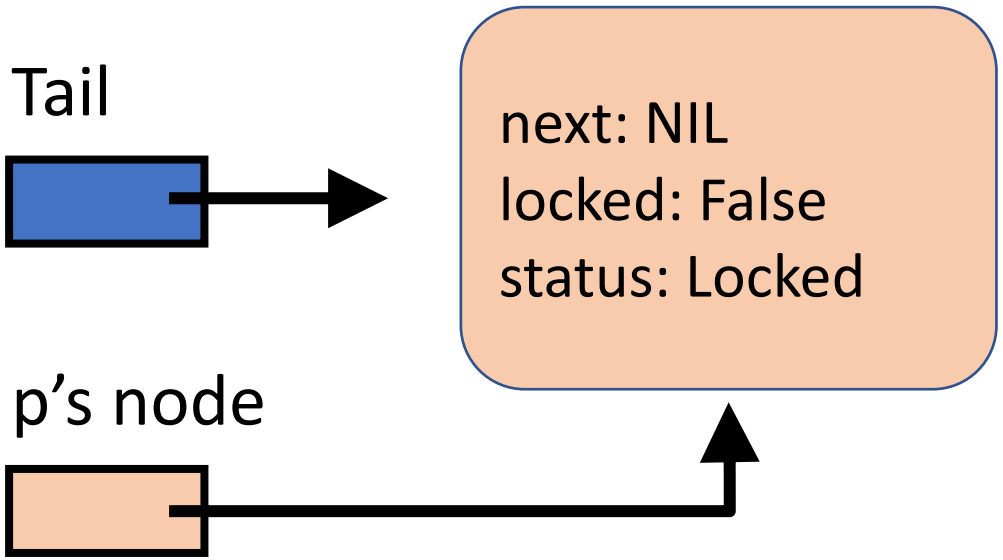
Initializing

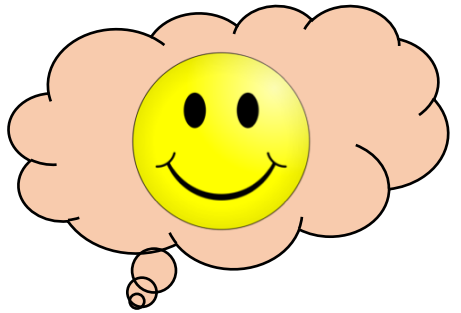
Process p Enters



Thread joins the queue

Process p Enters

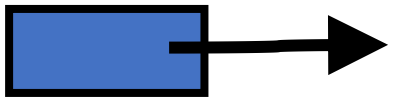




Continues to critical section

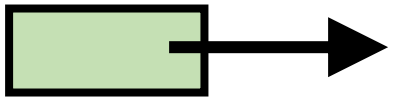
Process q Enters

Tail



next: NIL
locked: False
status: Locked

q’s node

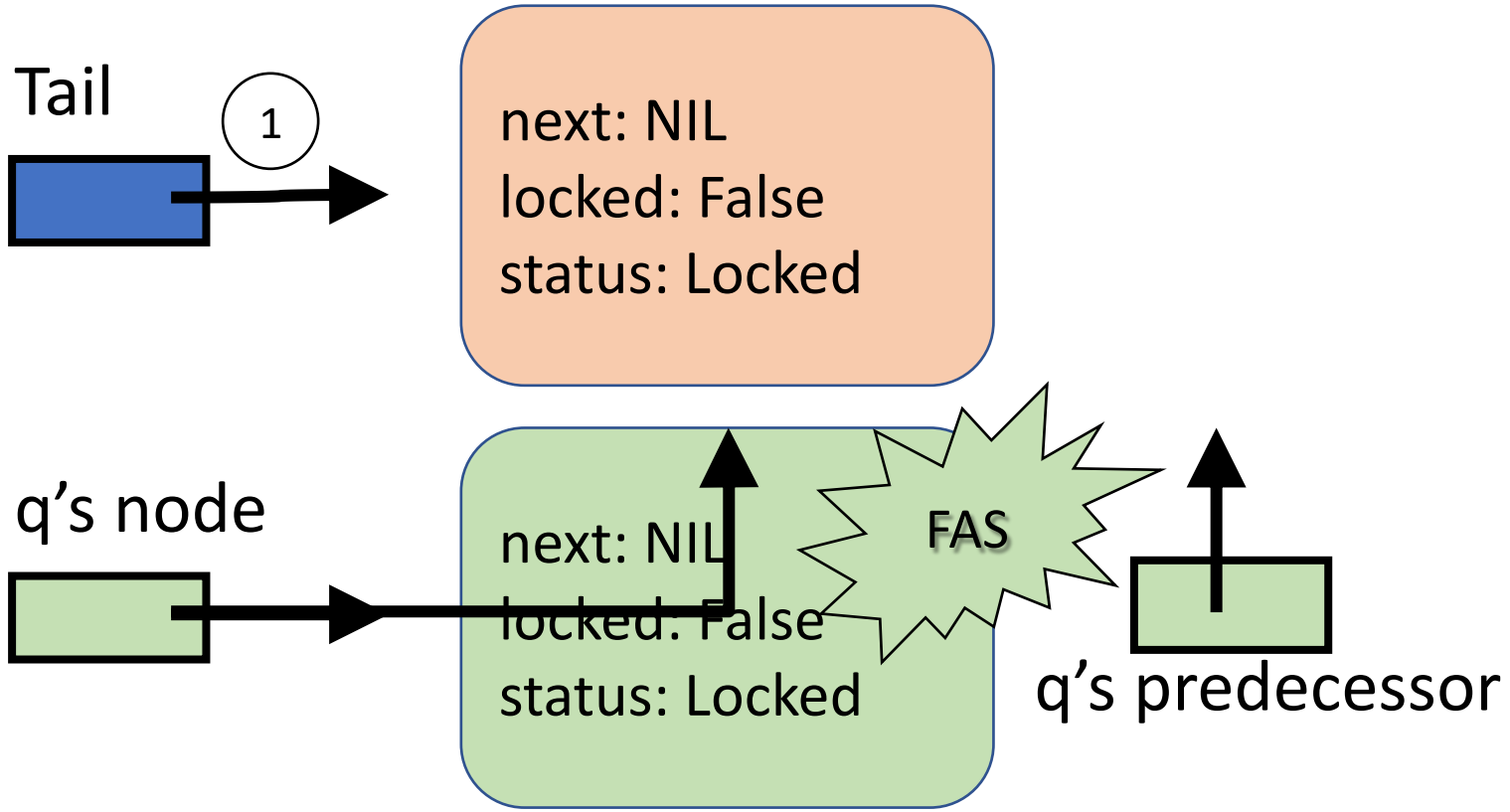


next: NIL
locked: False
status: Locked

In critical section

Initializing

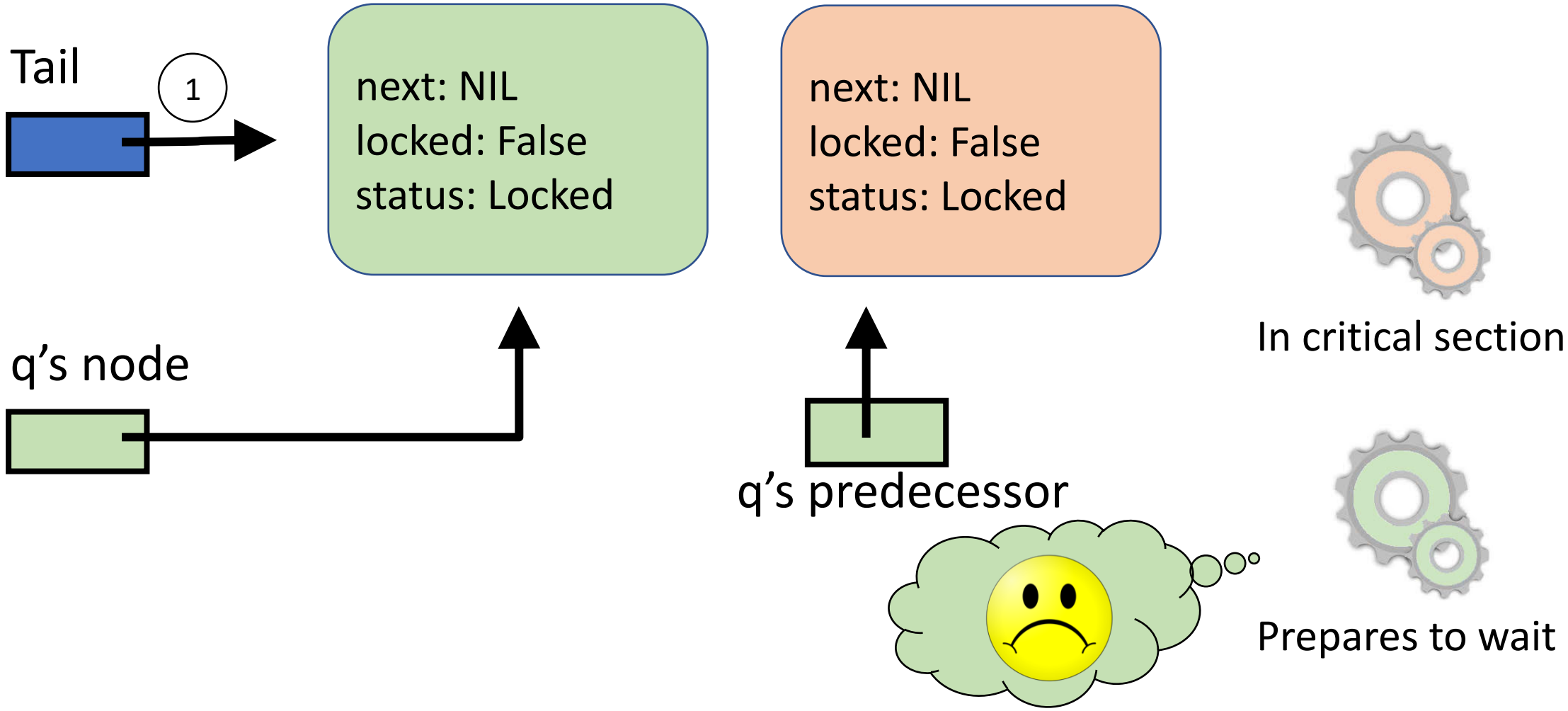
Process q Enters



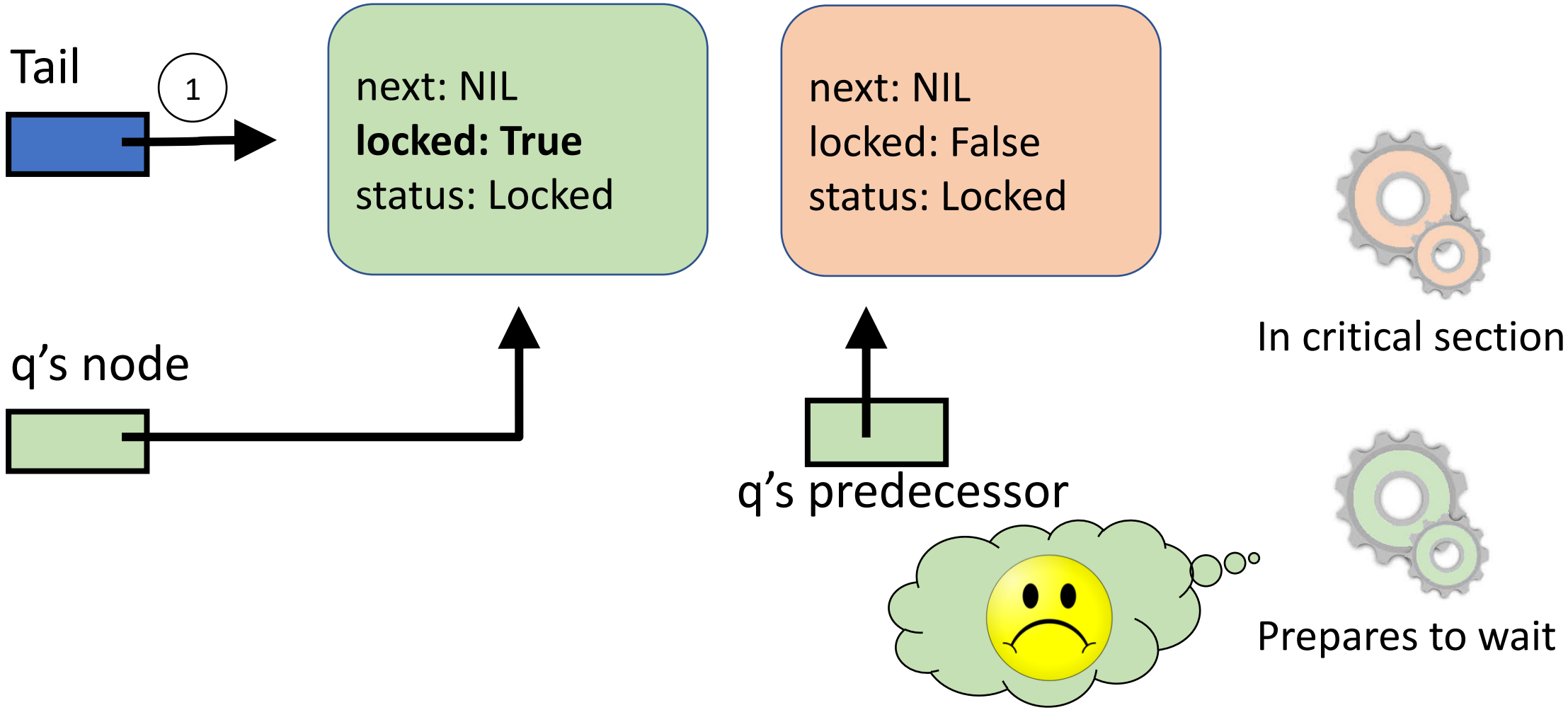
In critical section

Threads into the queue

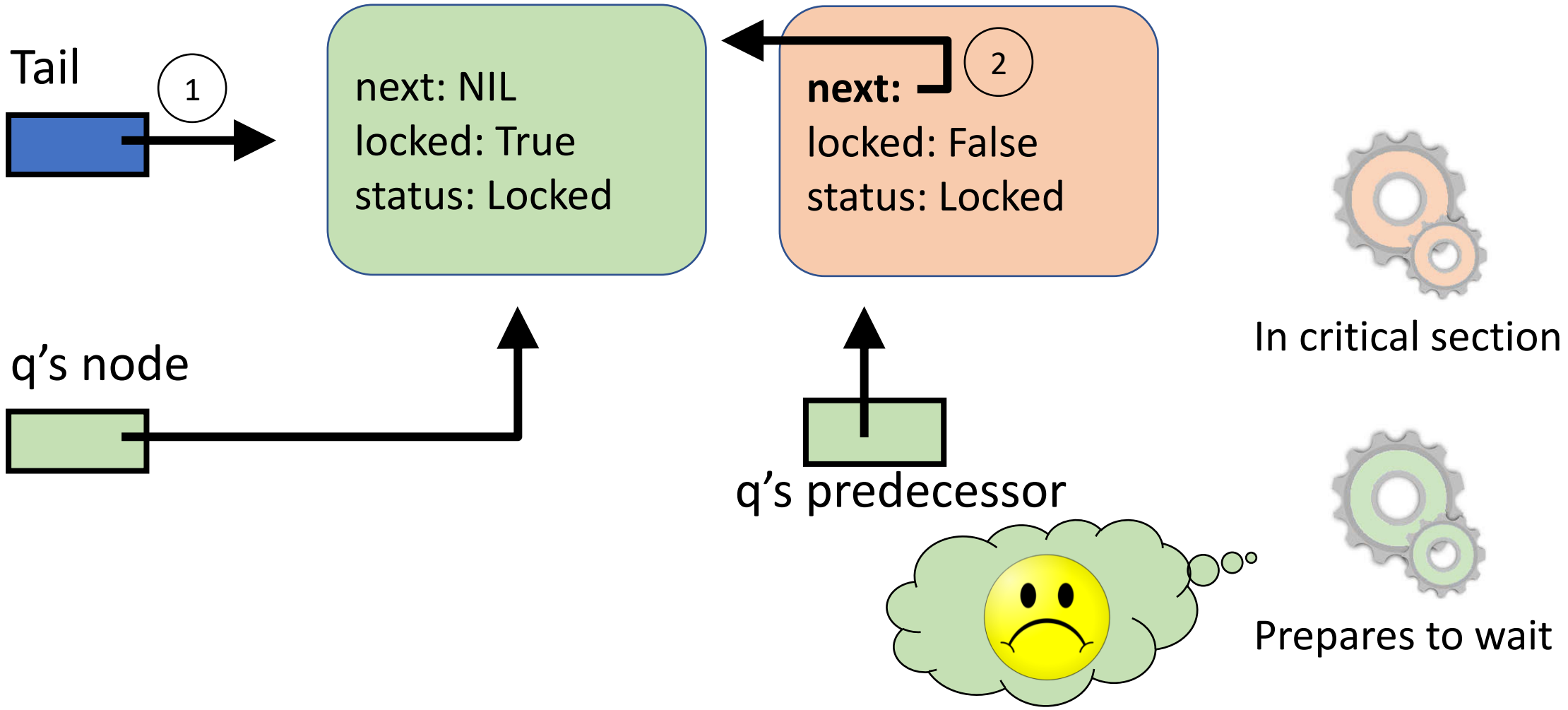
Process q Enters



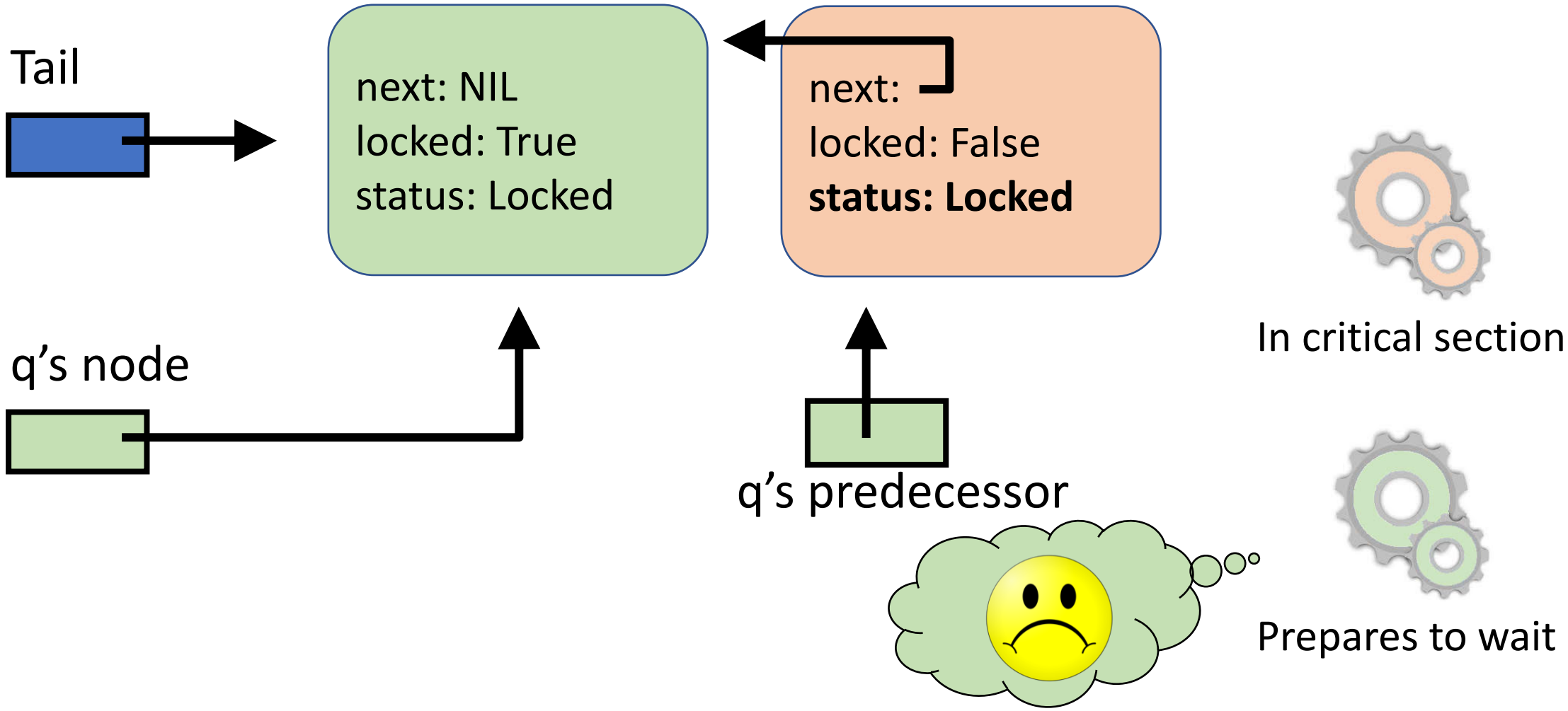
Process q Enters



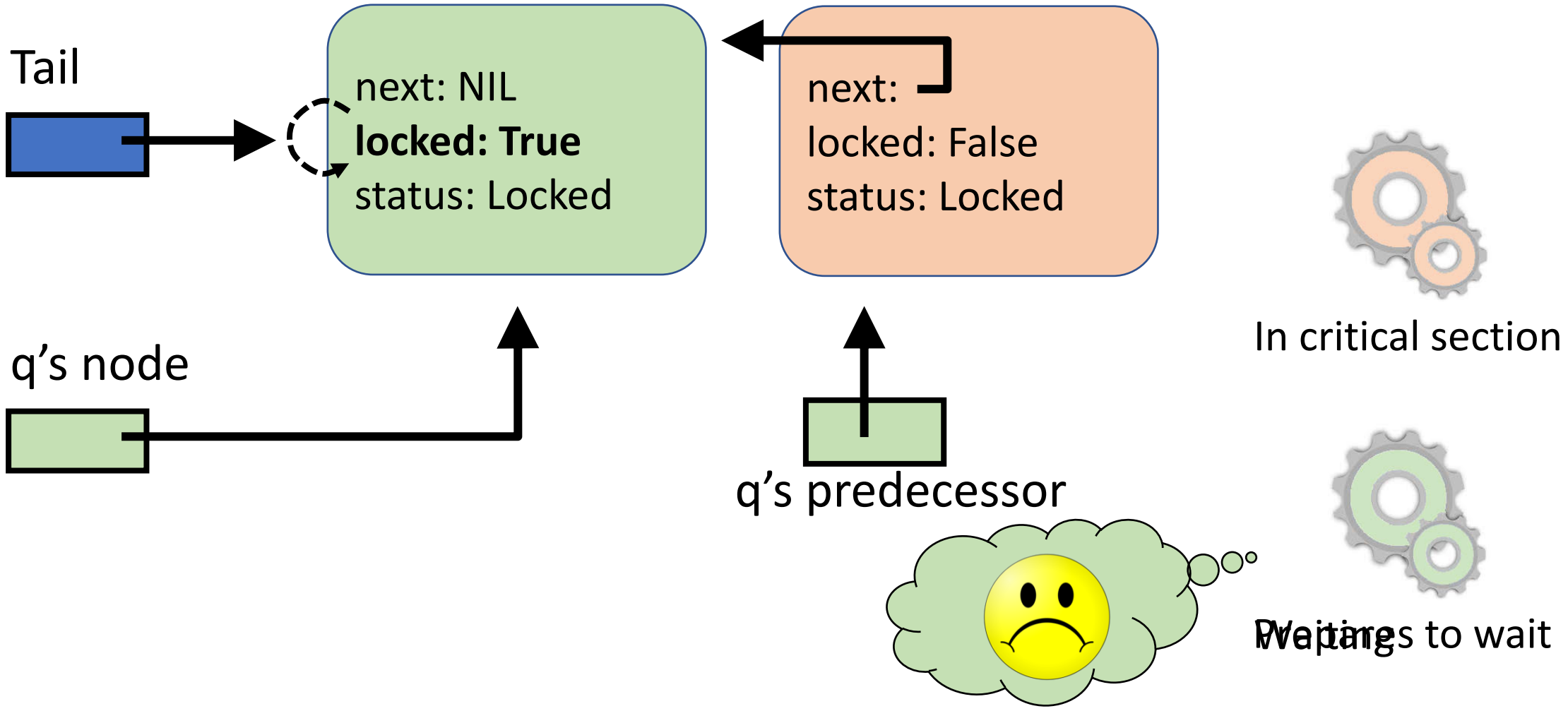
Process q Enters



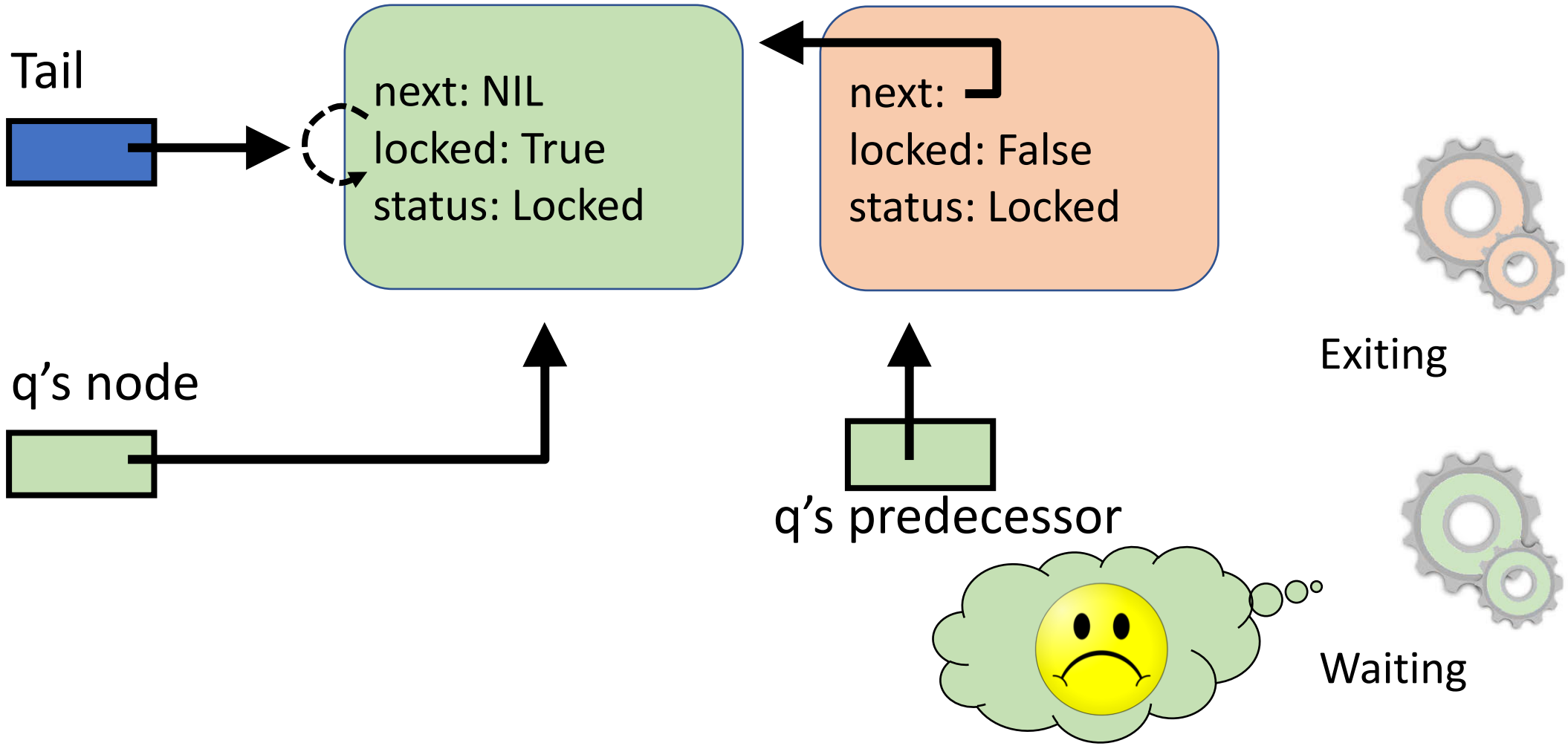
Process q Enters



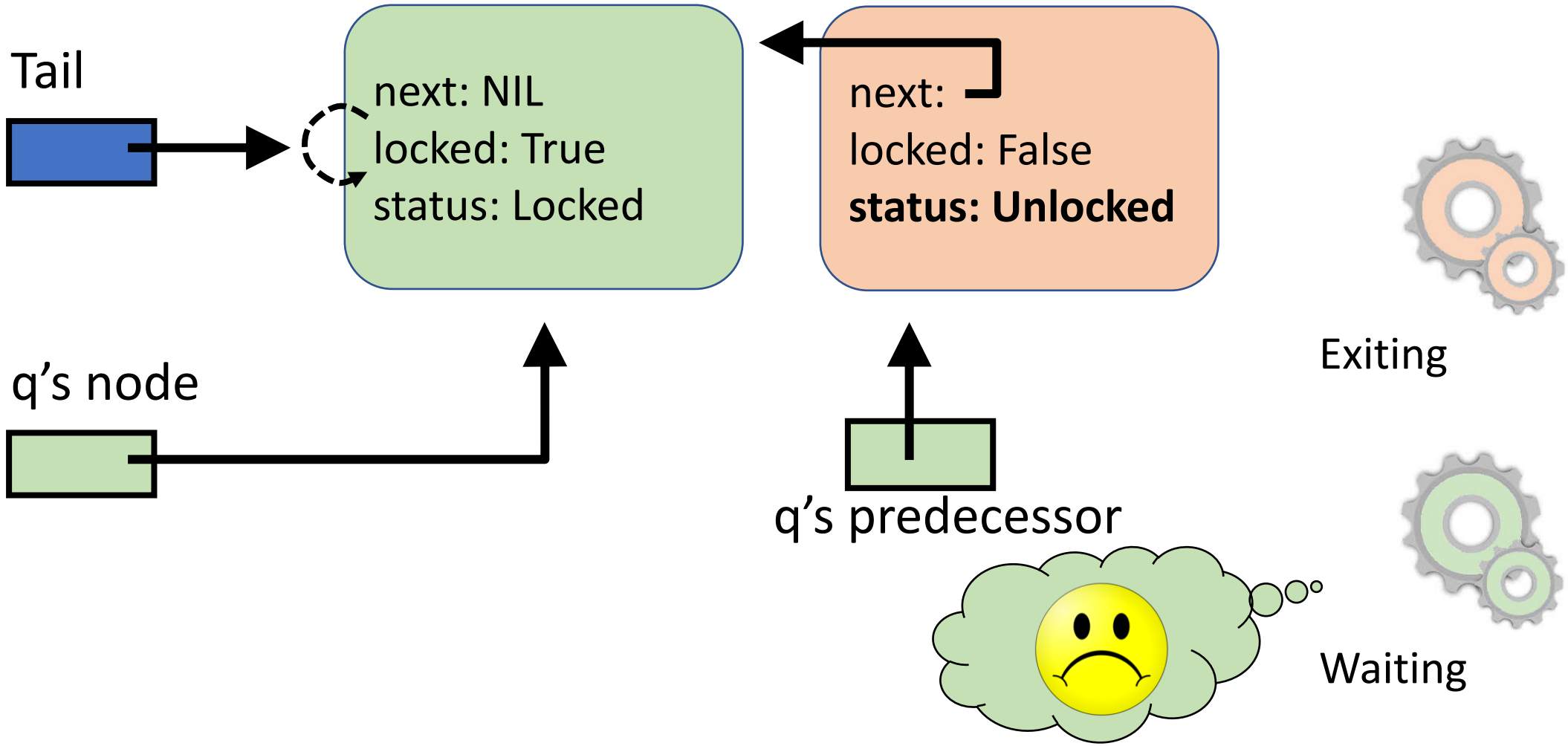
Process q Enters



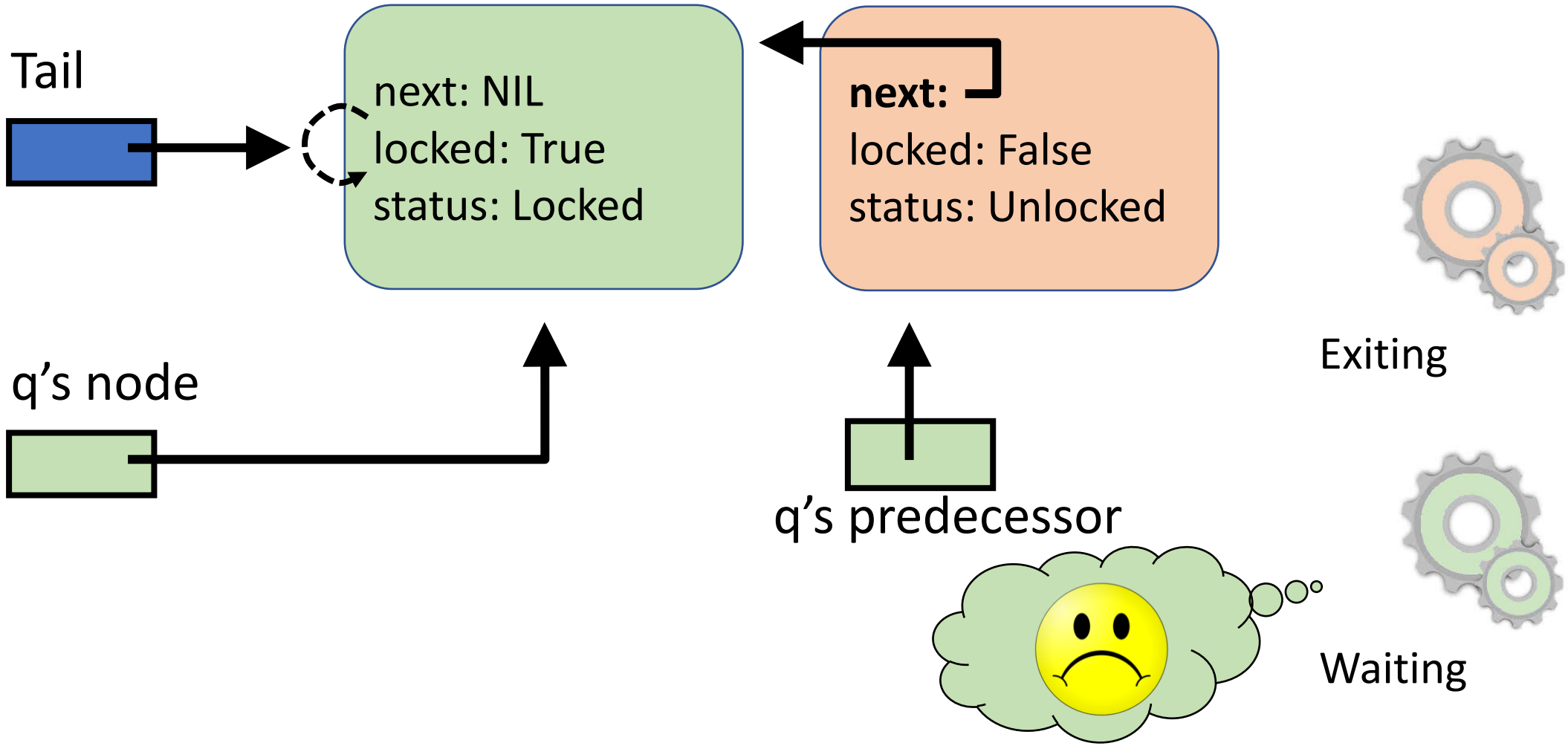
Process p Exits



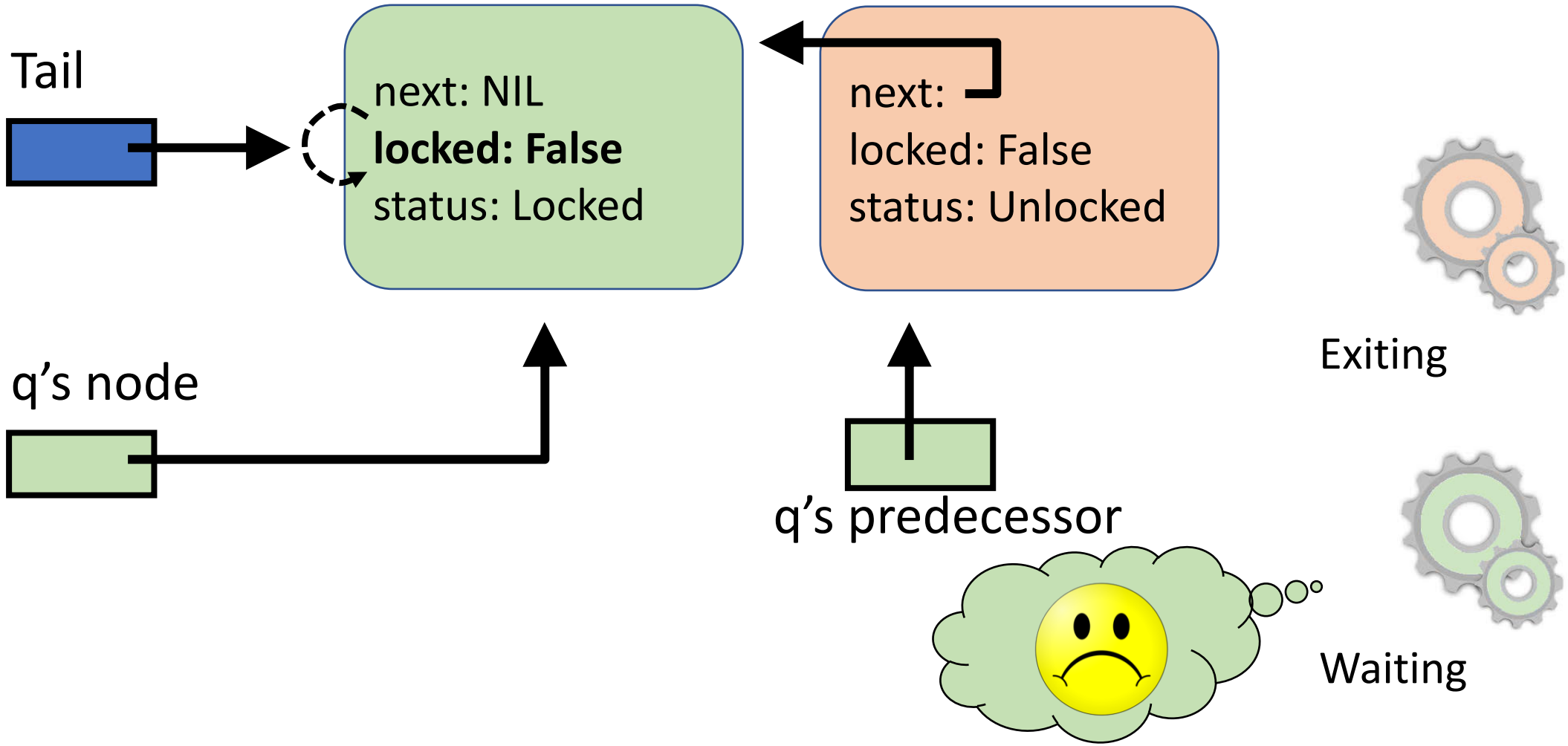
Process p Exits



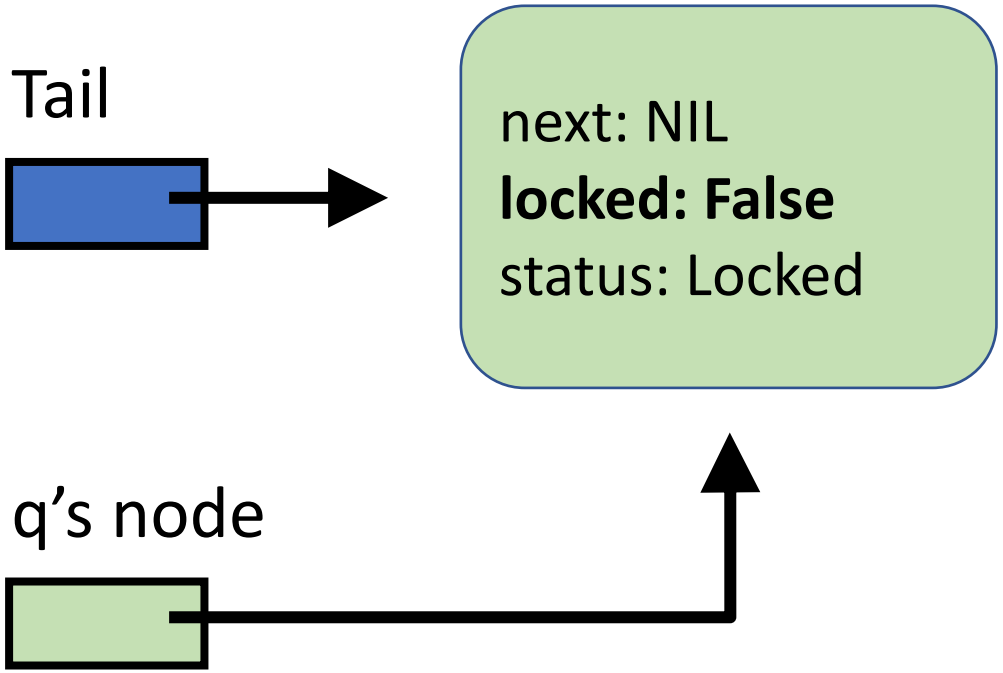
Process p Exits

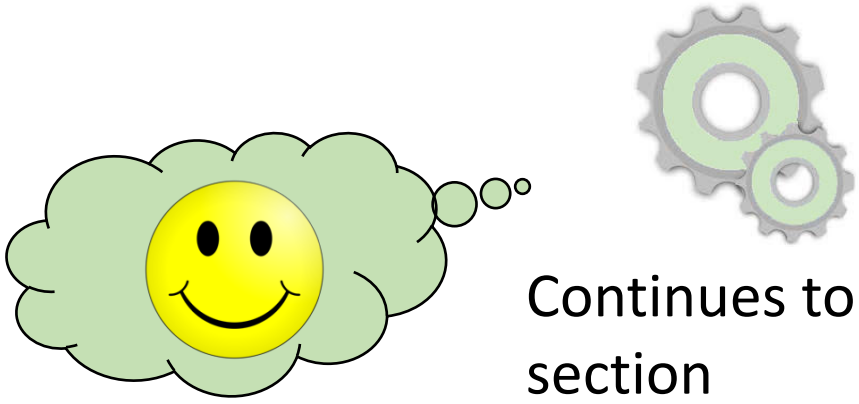


Process p Exits



Process q Enters

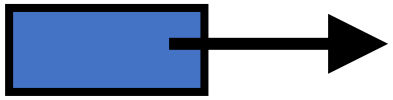




Continues to critical section

Process q In Critical Section

Tail

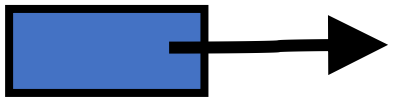


next: NIL
locked: False
status: Locked

In critical section

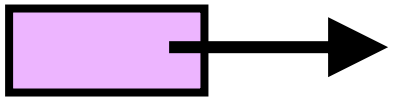
Process r Enters

Tail



next: NIL
locked: False
status: Locked

r’s node

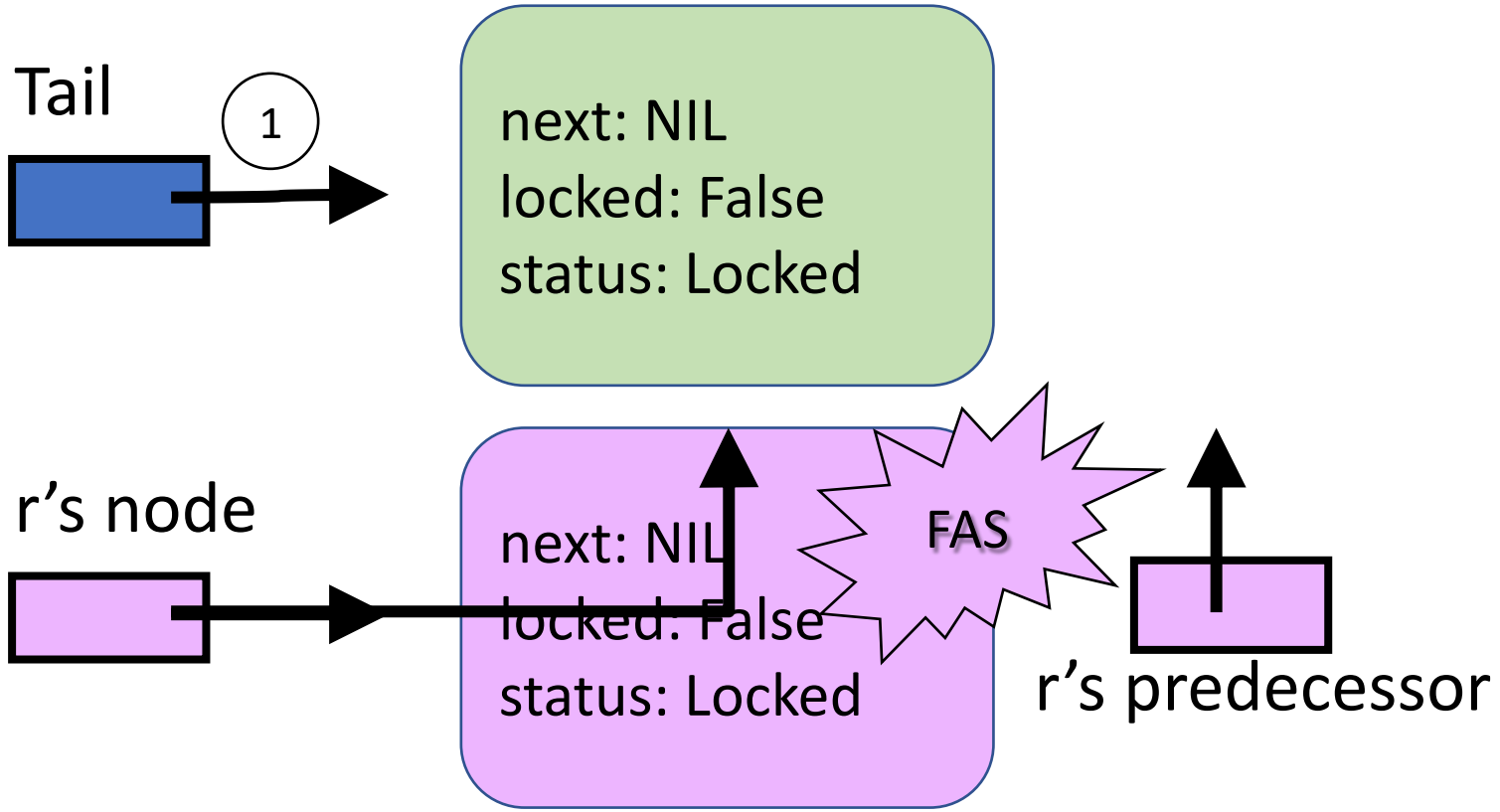


next: NIL
locked: False
status: Locked

In critical section

Initializing

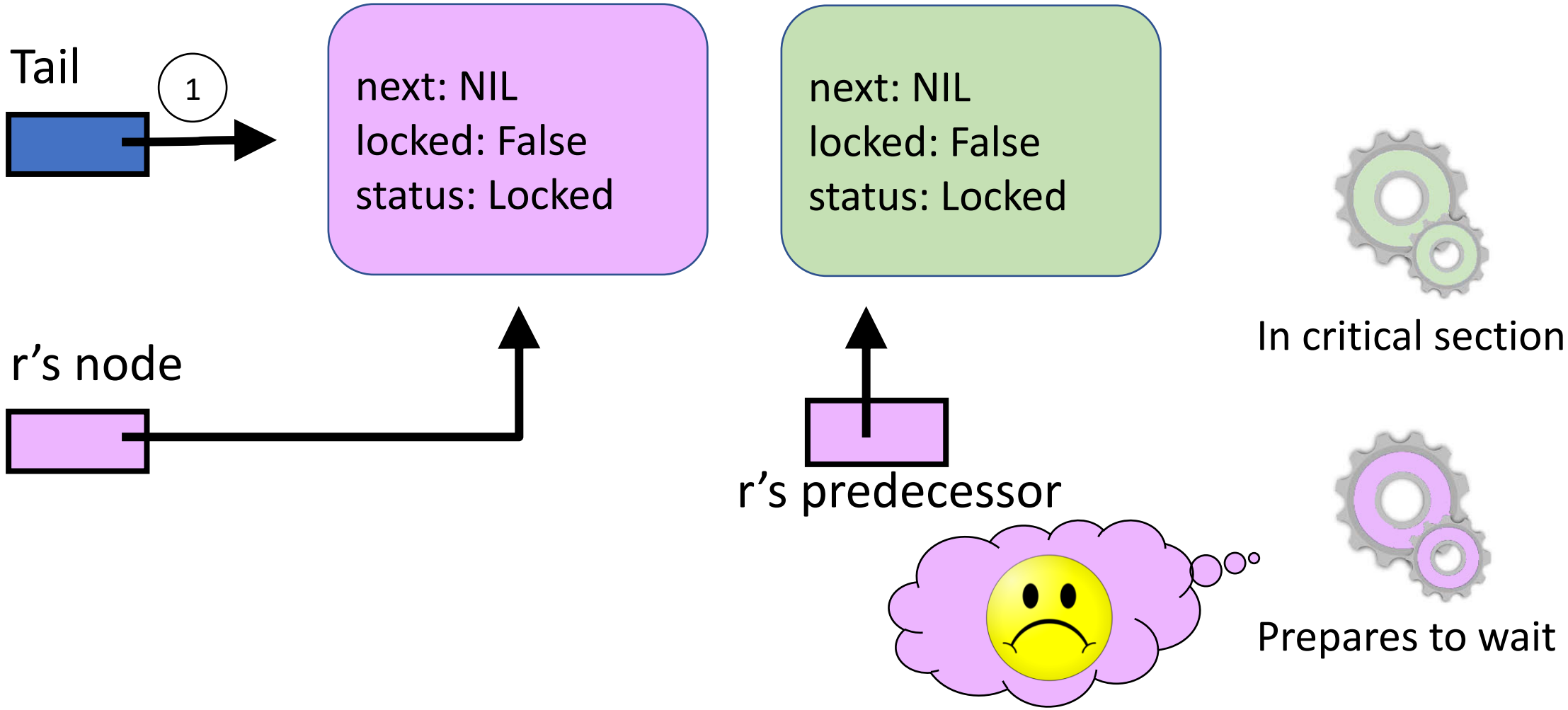
Process r Enters



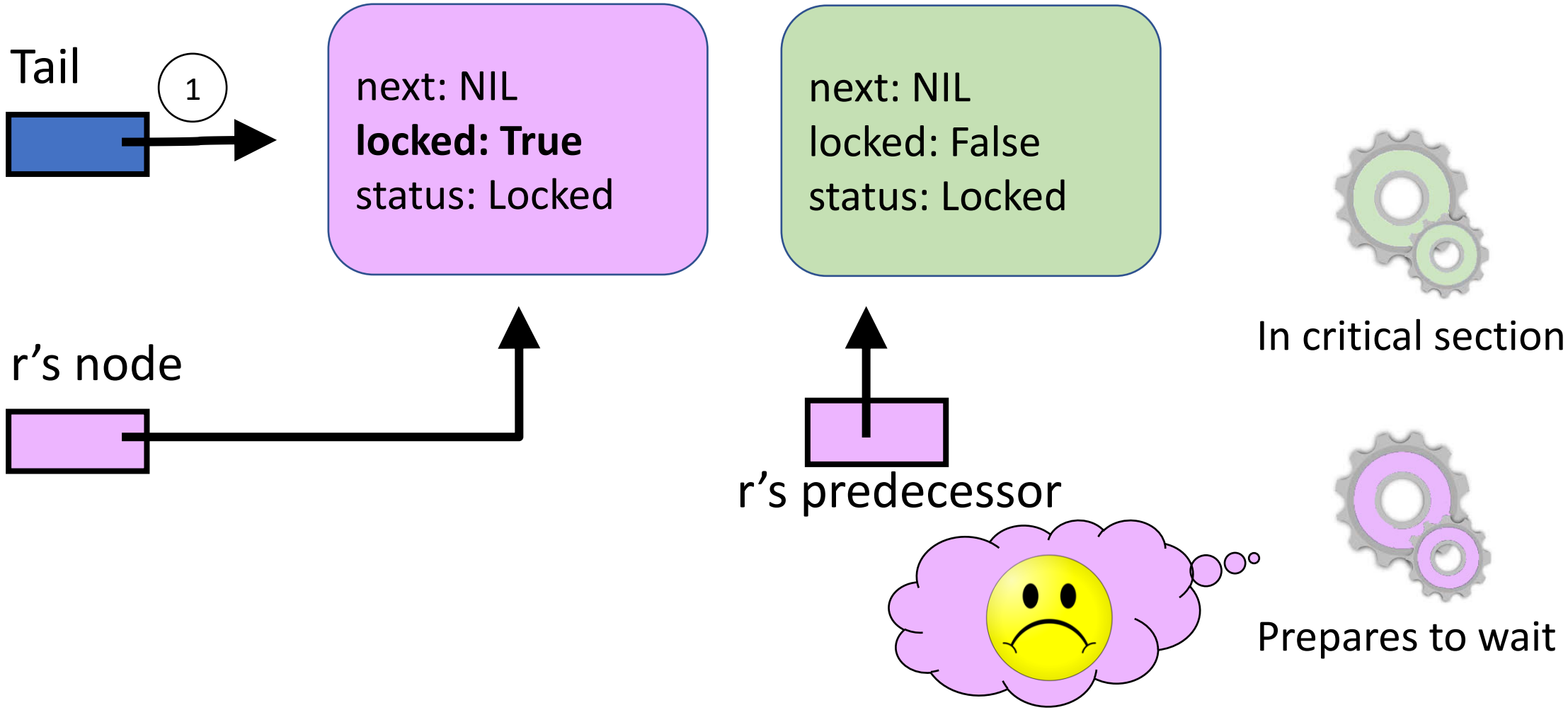
In critical section

Threads into the queue

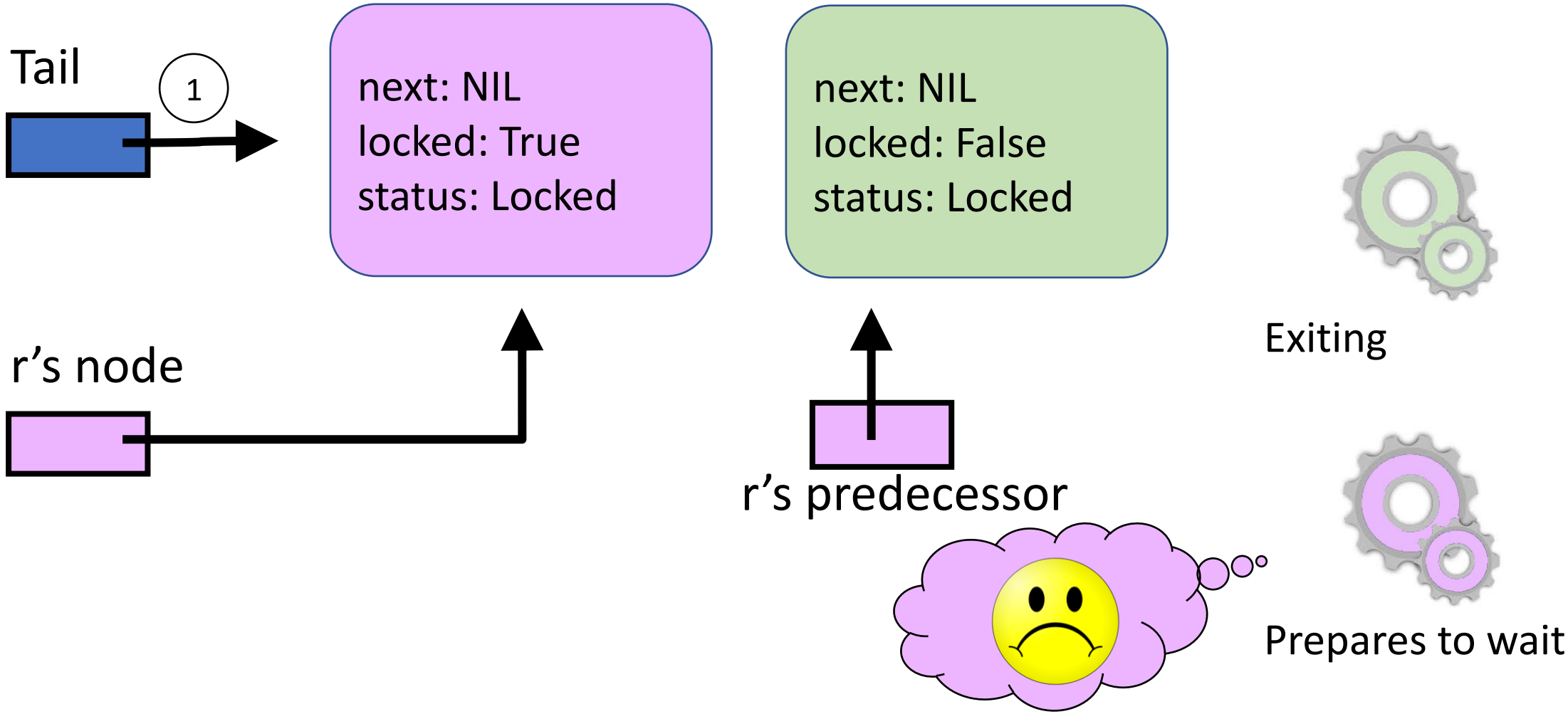
Process r Enters



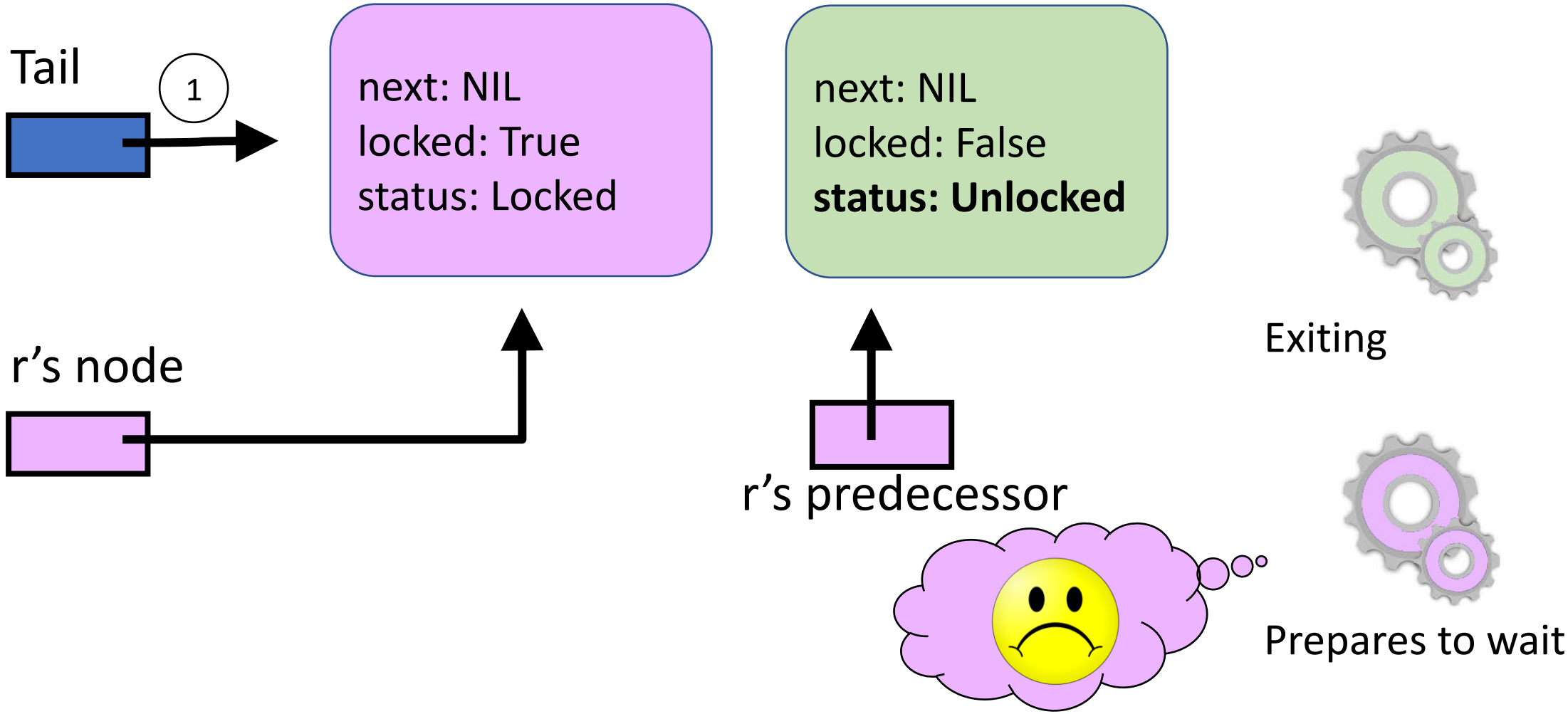
Process r Enters



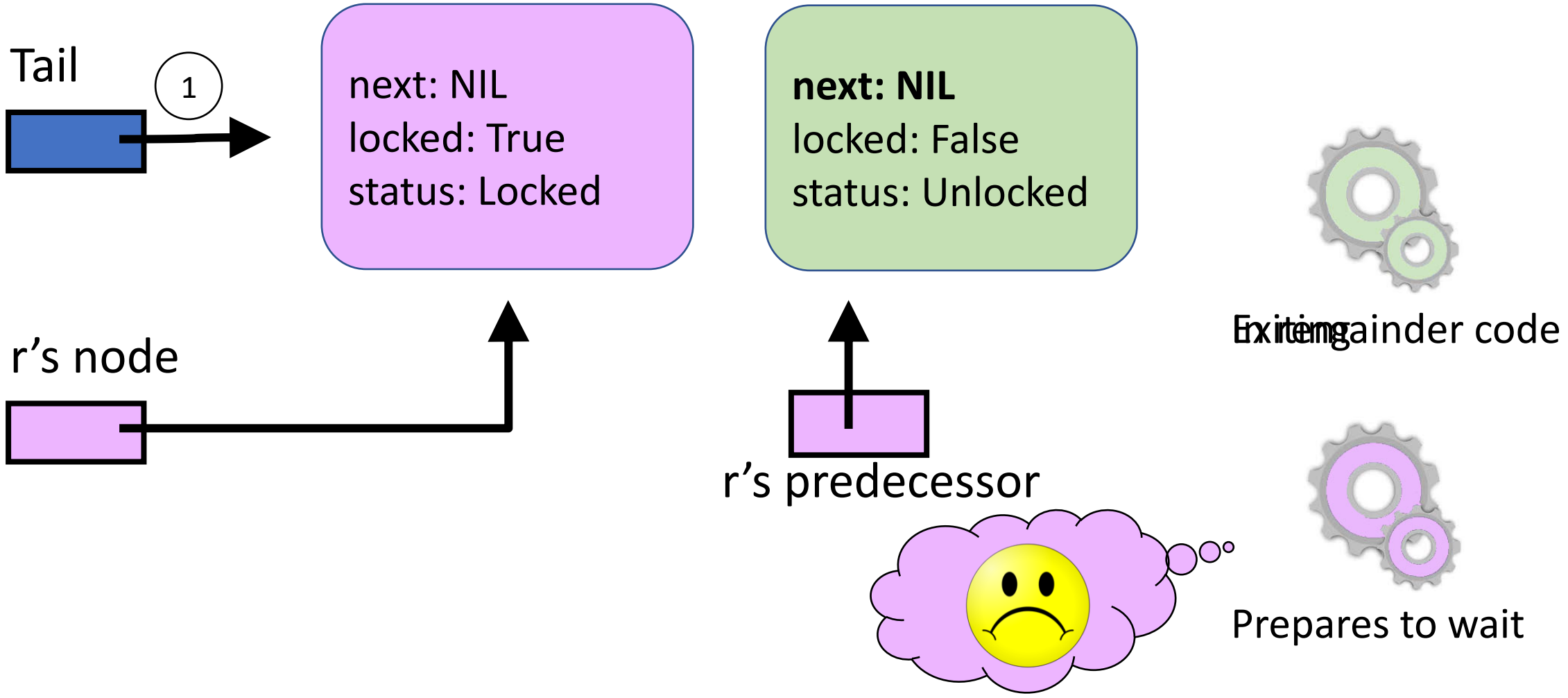
Process q Exits



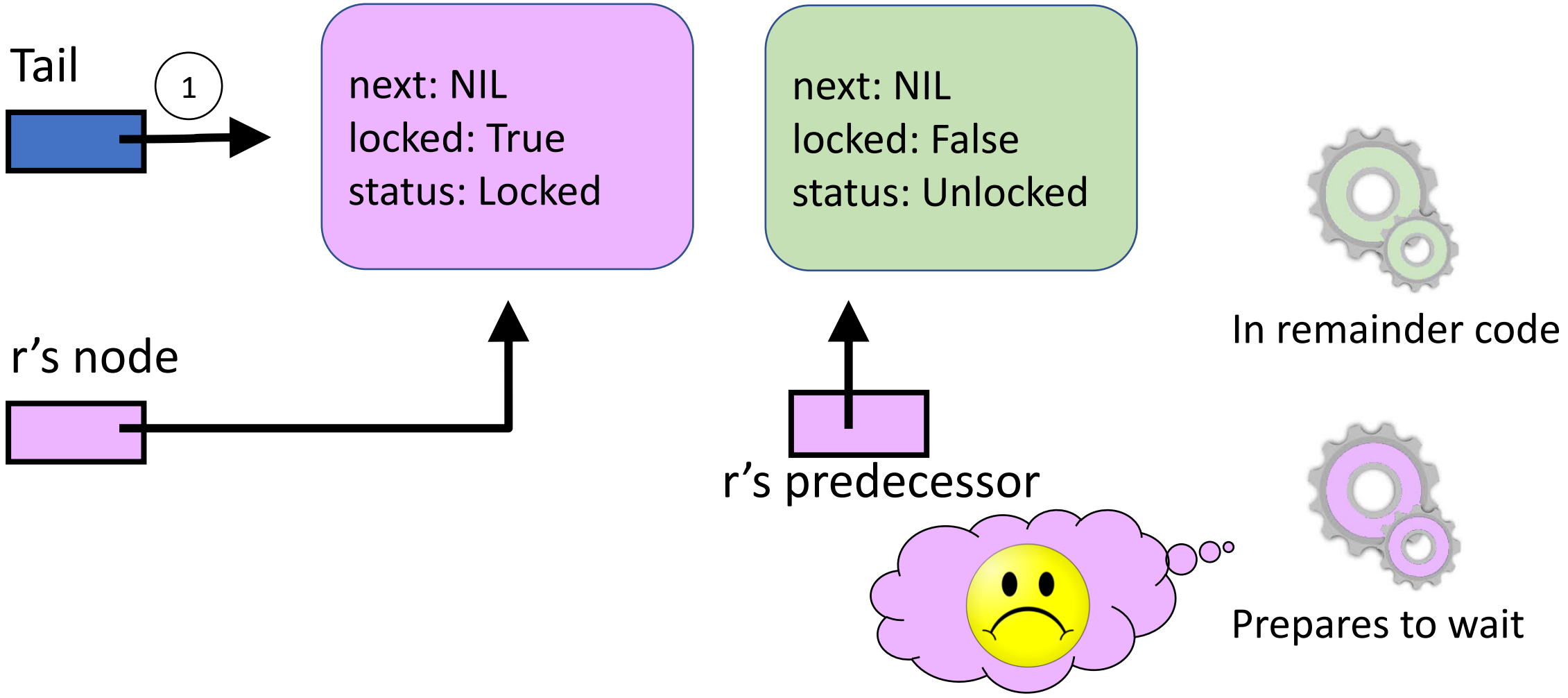
Process q Exits



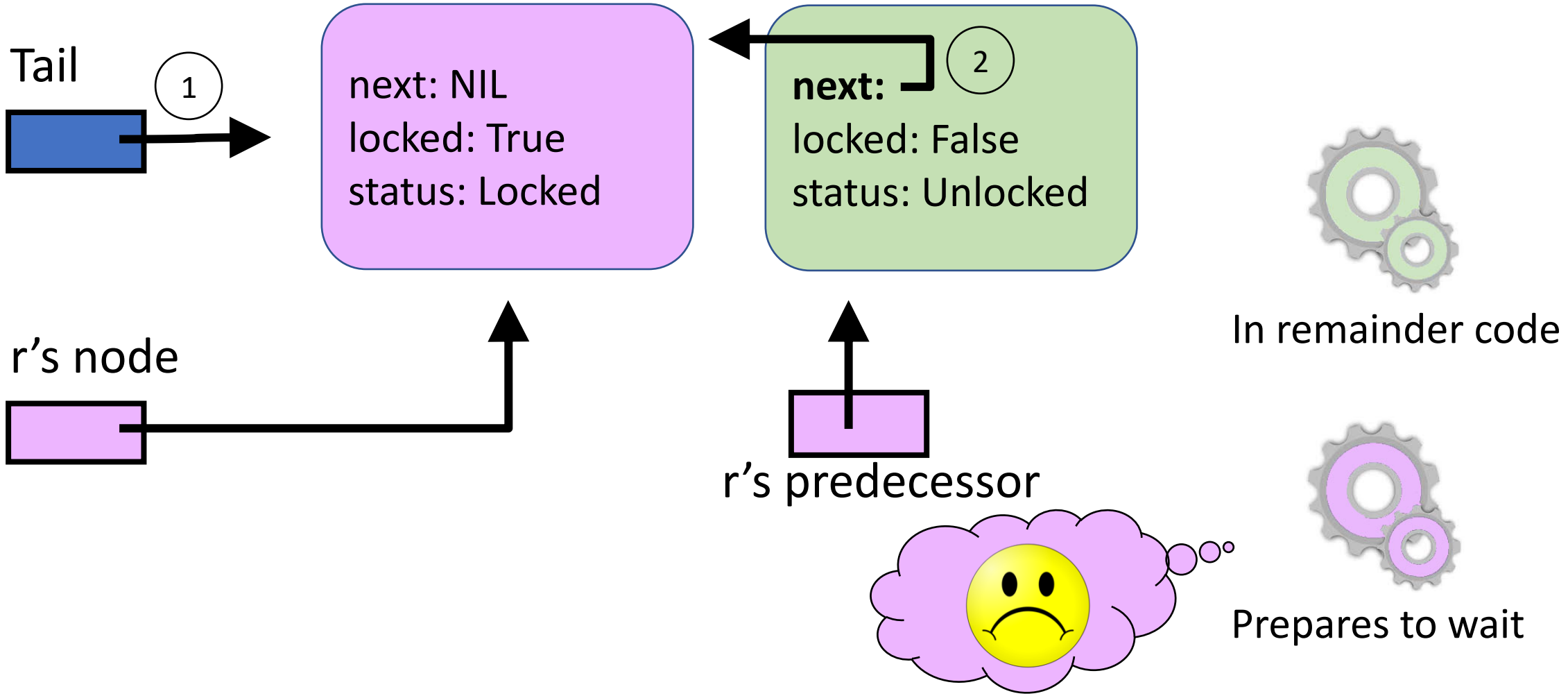
Process q Exits



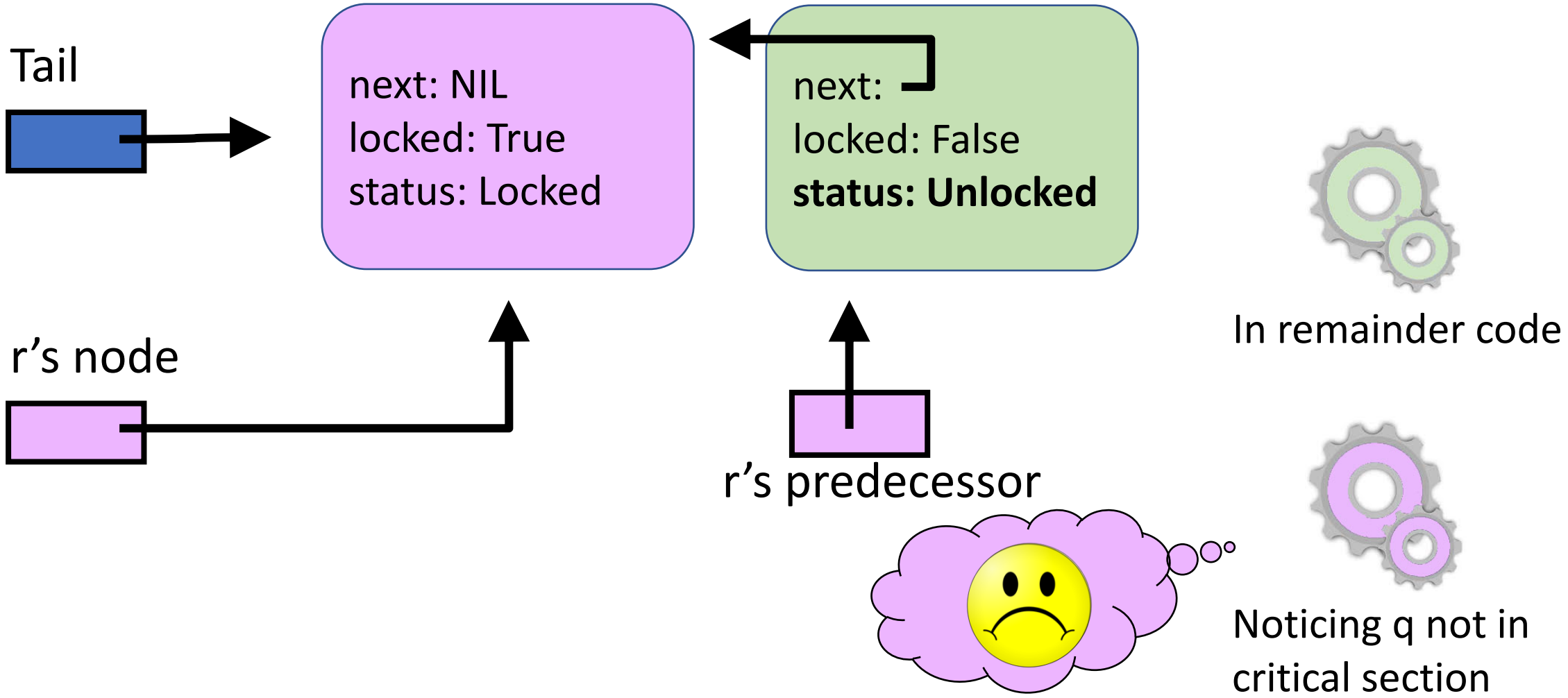
Process r Enters



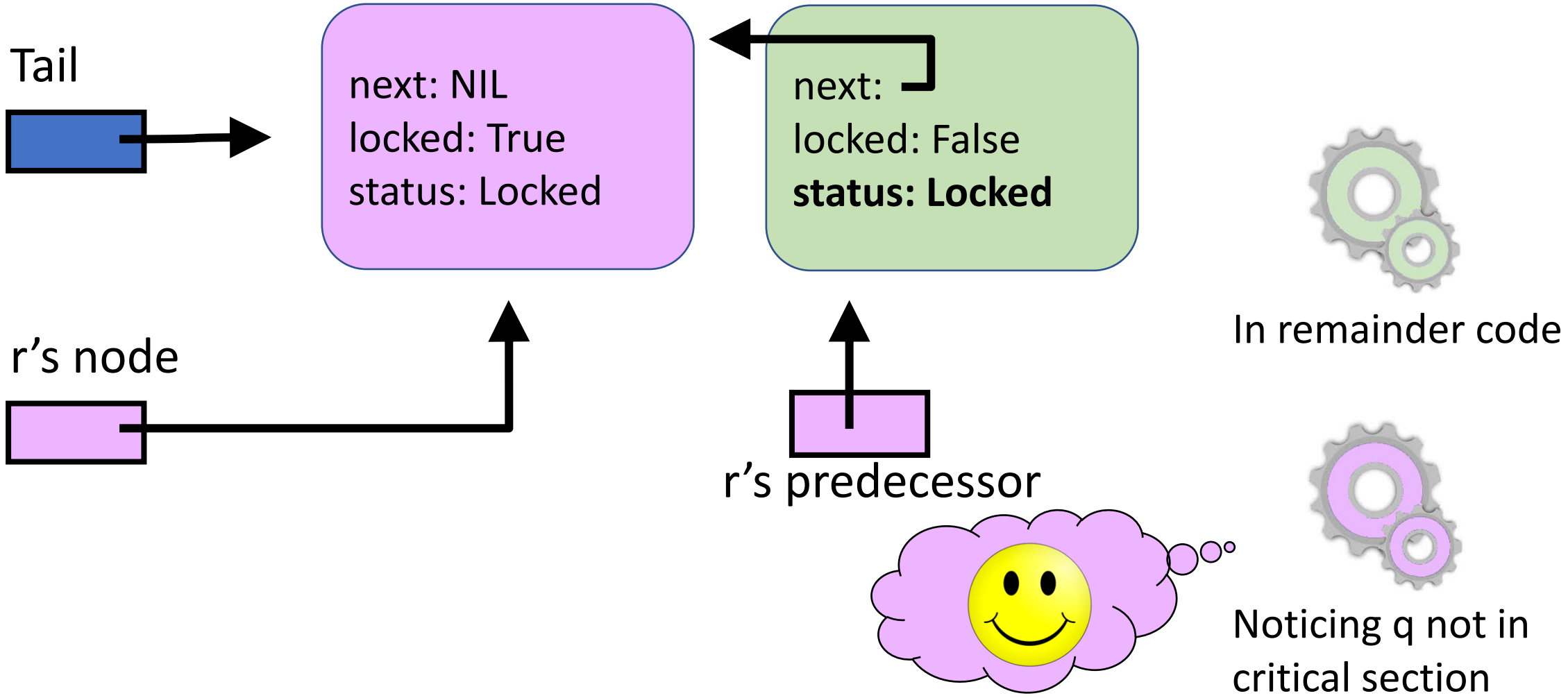
Process r Enters



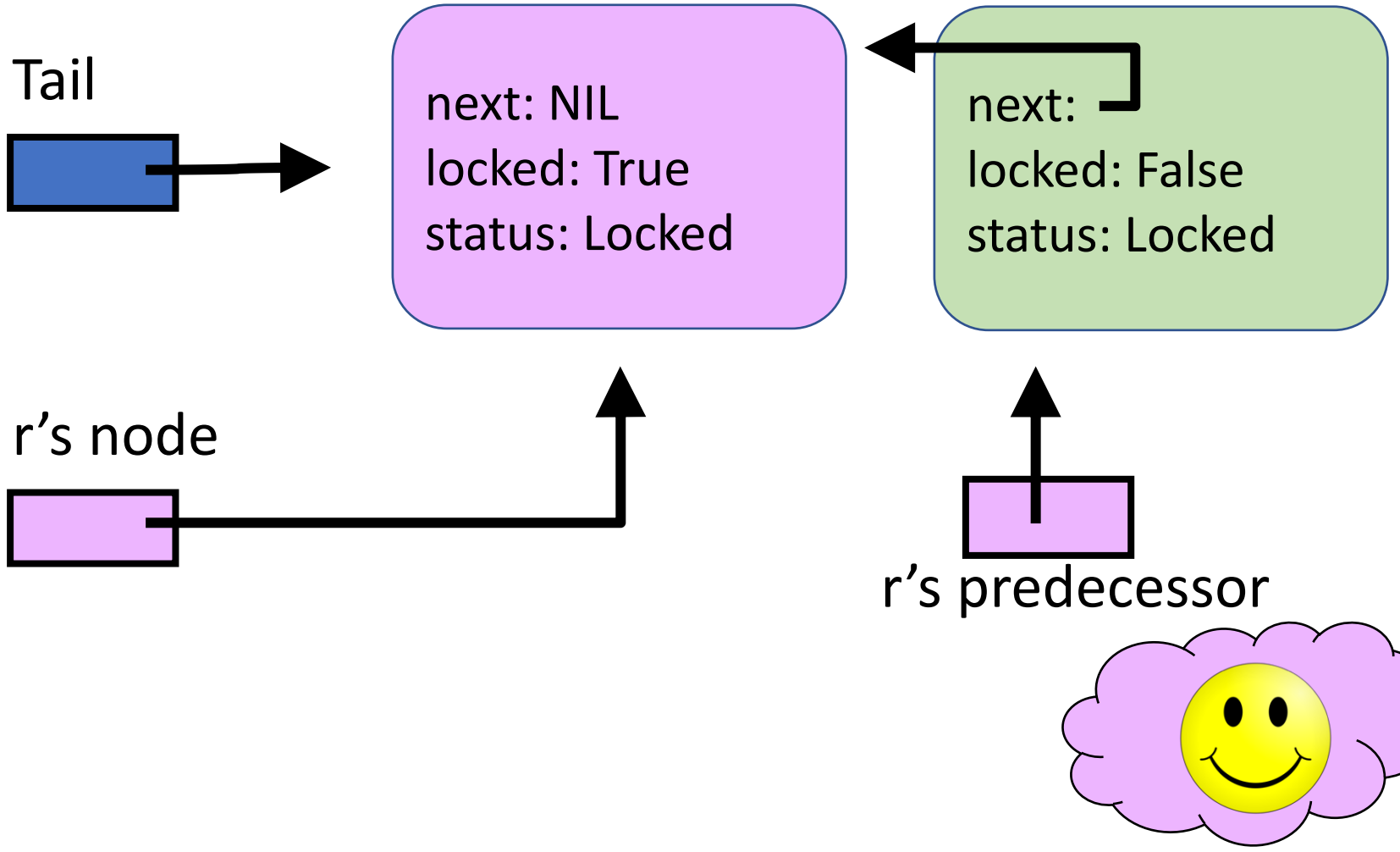
Process r Enters



Process r Enters



Process r Enters



In remainder code

Continues to critical section

Summary

- ✔ Wait-free exit codes
- ✔ Satisfy FIFO fairness
- ✔ Constant RMR complexity in CC and DSM models
- ✔ Not assumed that the number of processes is priori known
- ✔ Use only O(n) shared memory locations
- ✔ Make no assumptions on what and how memory is allocated
- ✔ Instruction set: Read/Write, Fetch-And-Store, Compare-And-Swap

Open Questions

Algorithms extensions:

- Readers-writers lock
- Group mutual exclusion
- Abortable
- Recoverable

Thank you!

Questions?