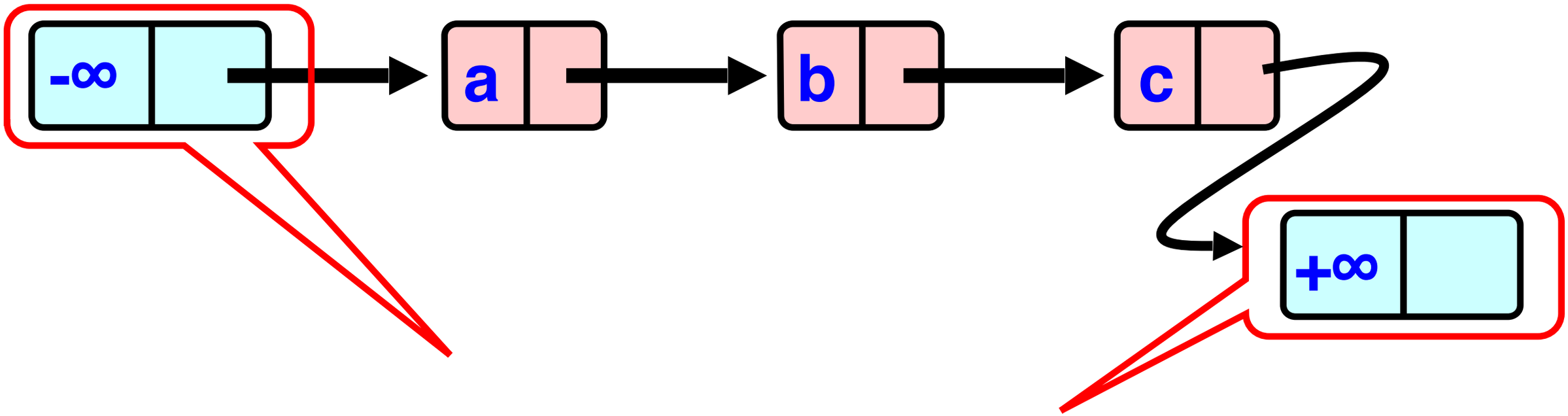
The List-Based Set



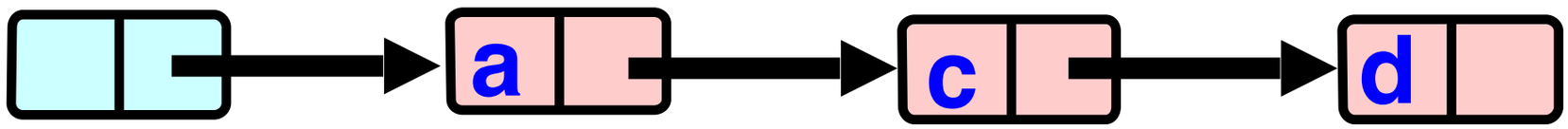
Sorted with Sentinel nodes
(min & max possible keys)

Invariants

- Sentinel nodes
  - tail reachable from head
- Sorted
- No duplicates

Sequential List Based Set

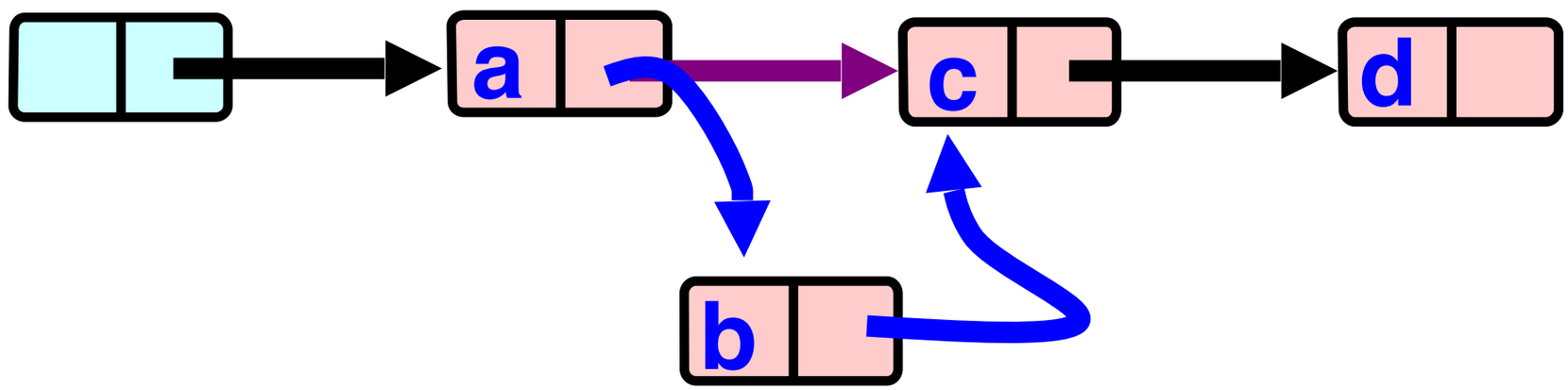
Add()



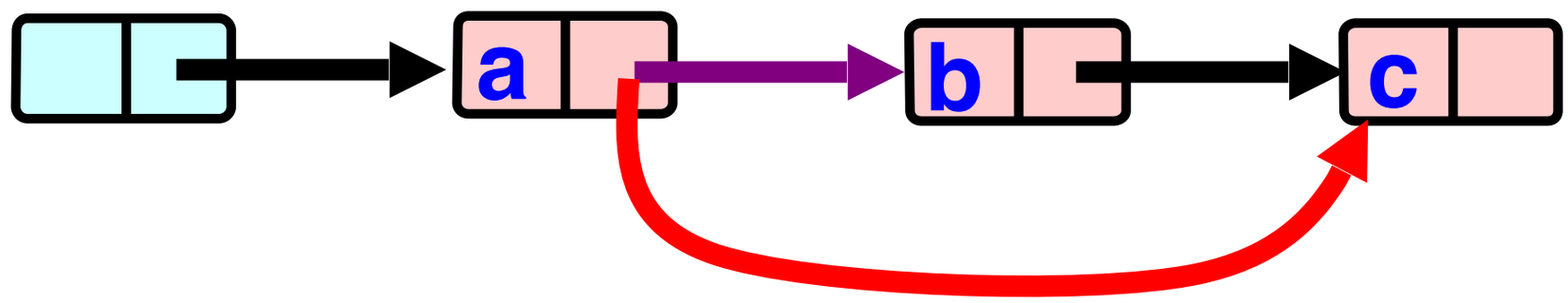
Remove()

Sequential List Based Set

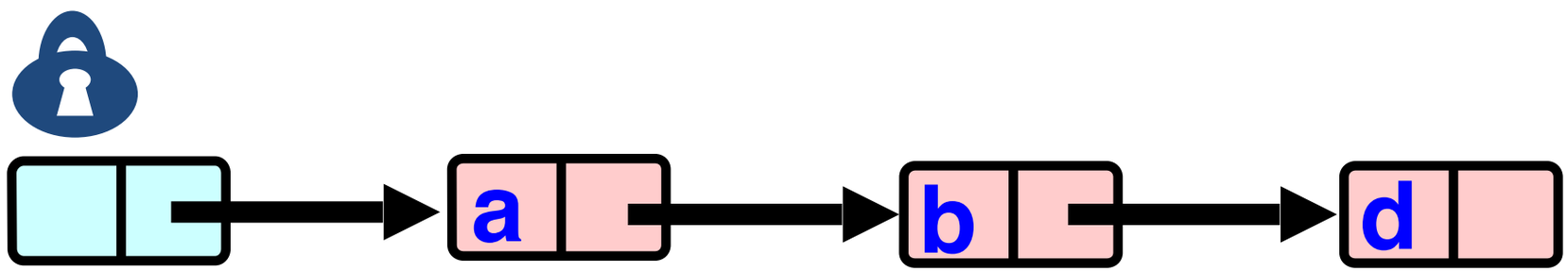
Add()



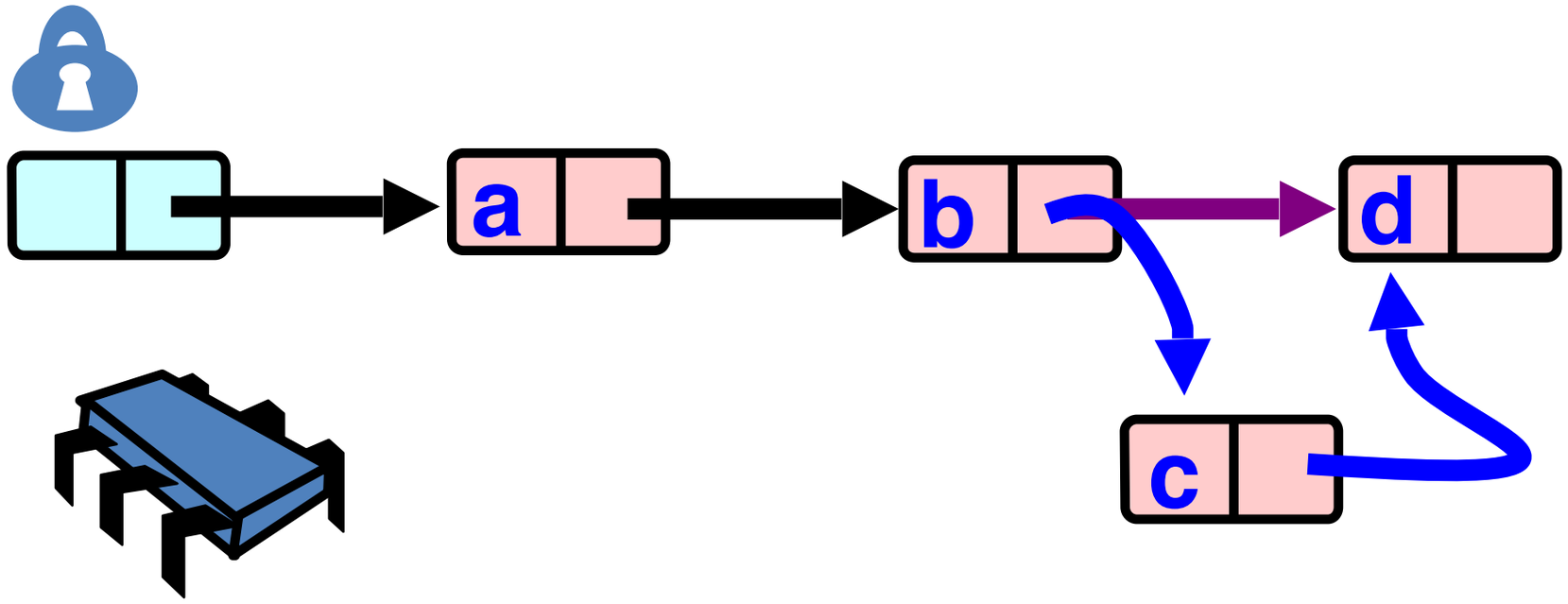
Remove()



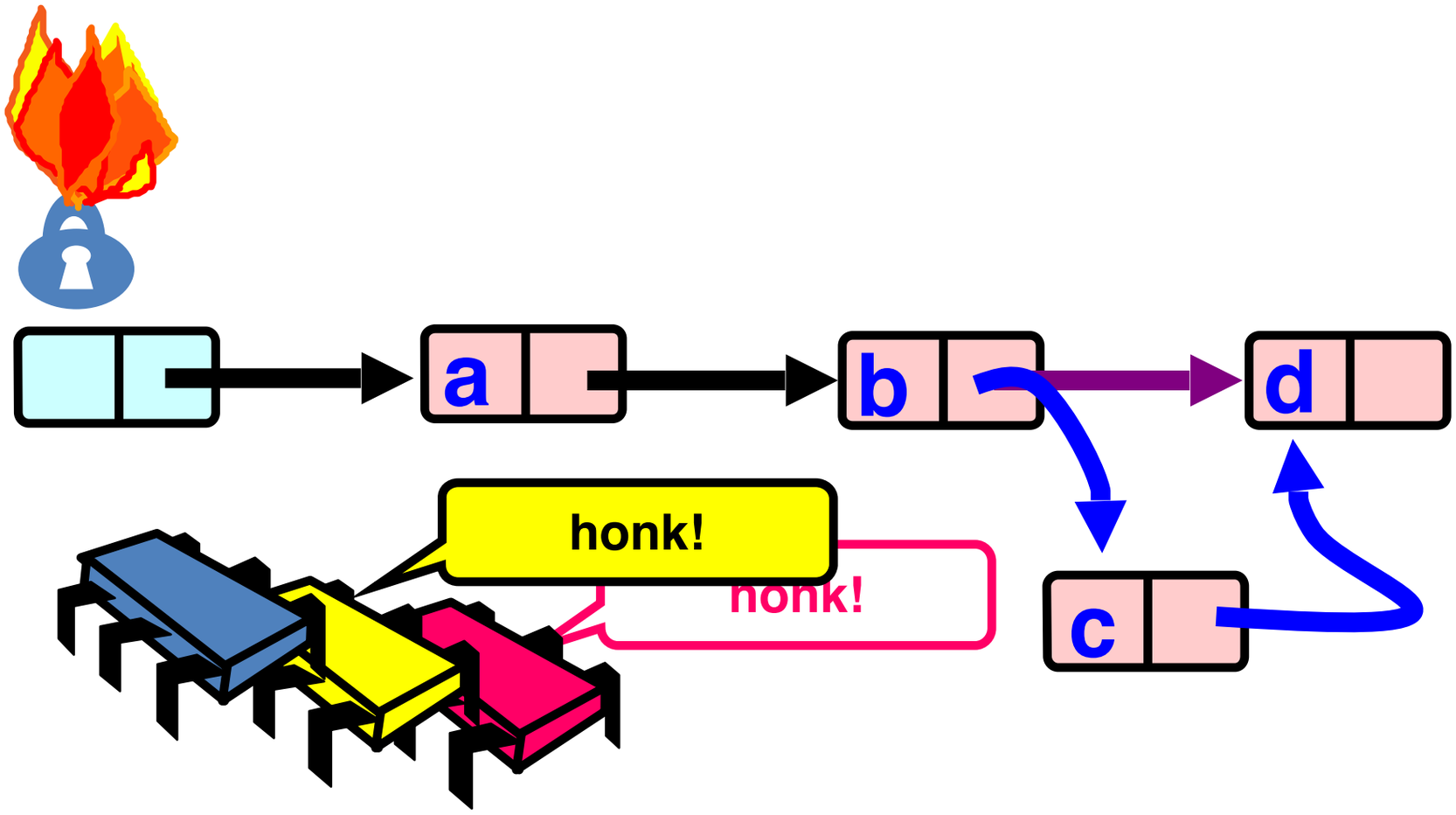
Coarse Grained Locking



Coarse Grained Locking



Coarse Grained Locking



Simple but hotspot + bottleneck

Coarse-Grained Locking

- Easy, same as synchronized methods
  - "One lock to rule them all …"

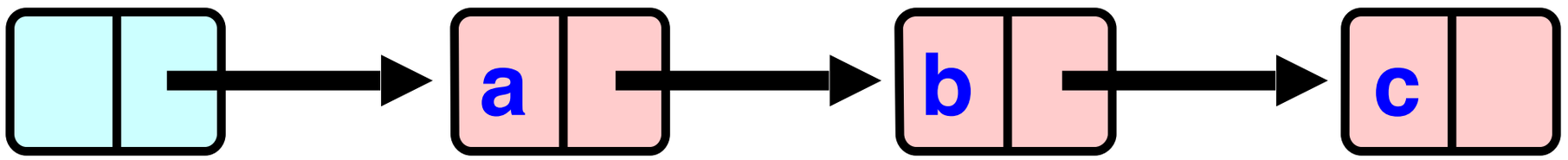
Coarse-Grained Locking

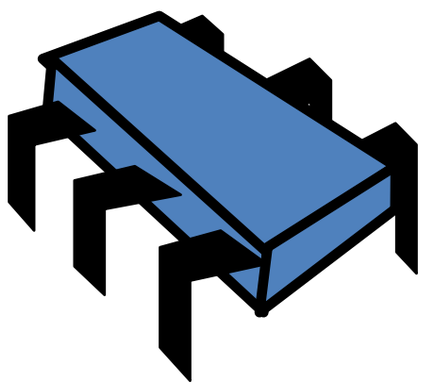
- Easy, same as synchronized methods
  - "One lock to rule them all …"
- Simple, clearly correct
  - Deserves respect!
- Works poorly with contention
  - Queue locks help
  - But bottleneck still an issue

Fine-grained Locking

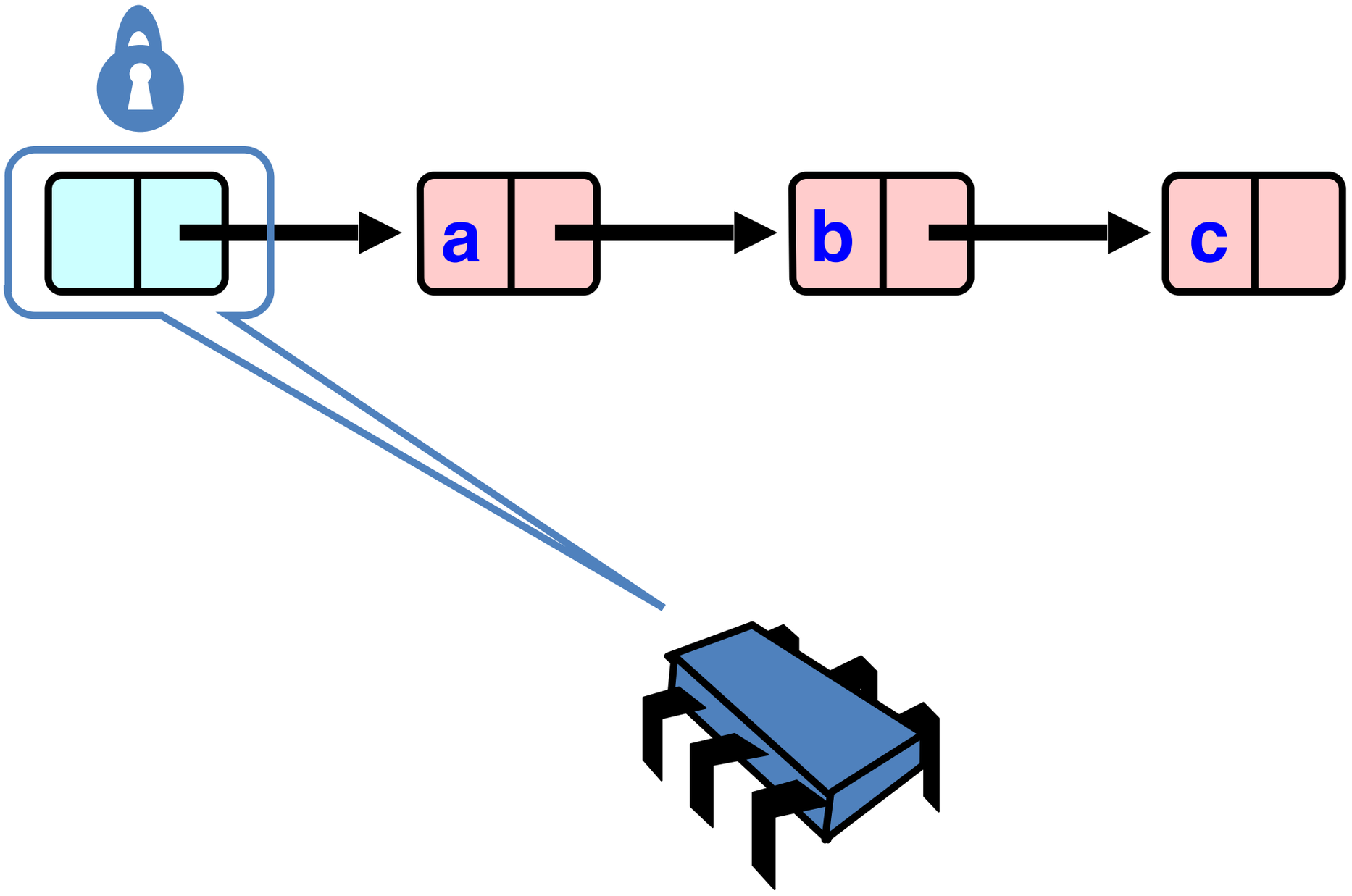
- Requires **careful** thought
- Split object into pieces
  - Each piece has own lock
  - Methods that work on disjoint pieces need not exclude each other

Hand-over-Hand locking

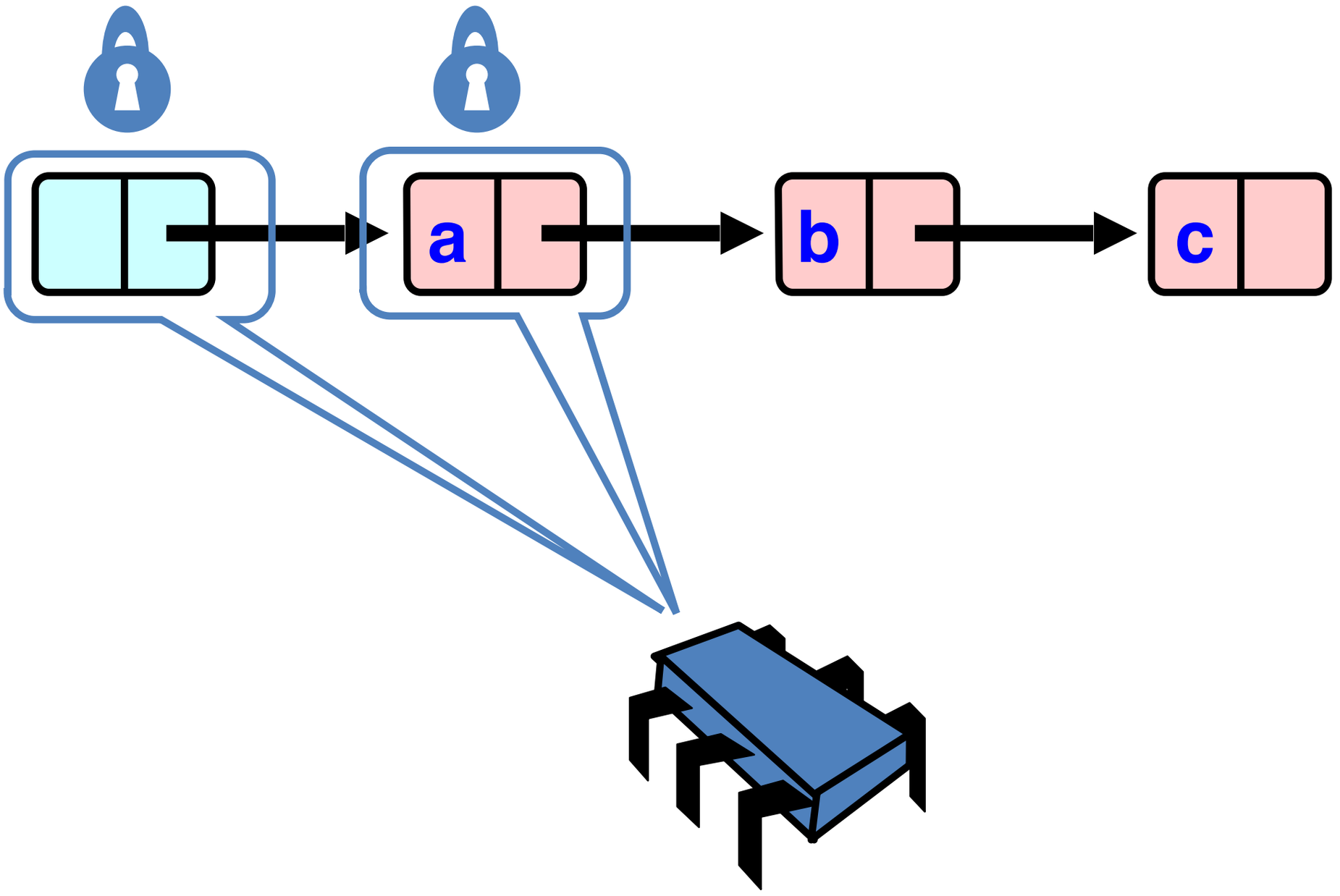




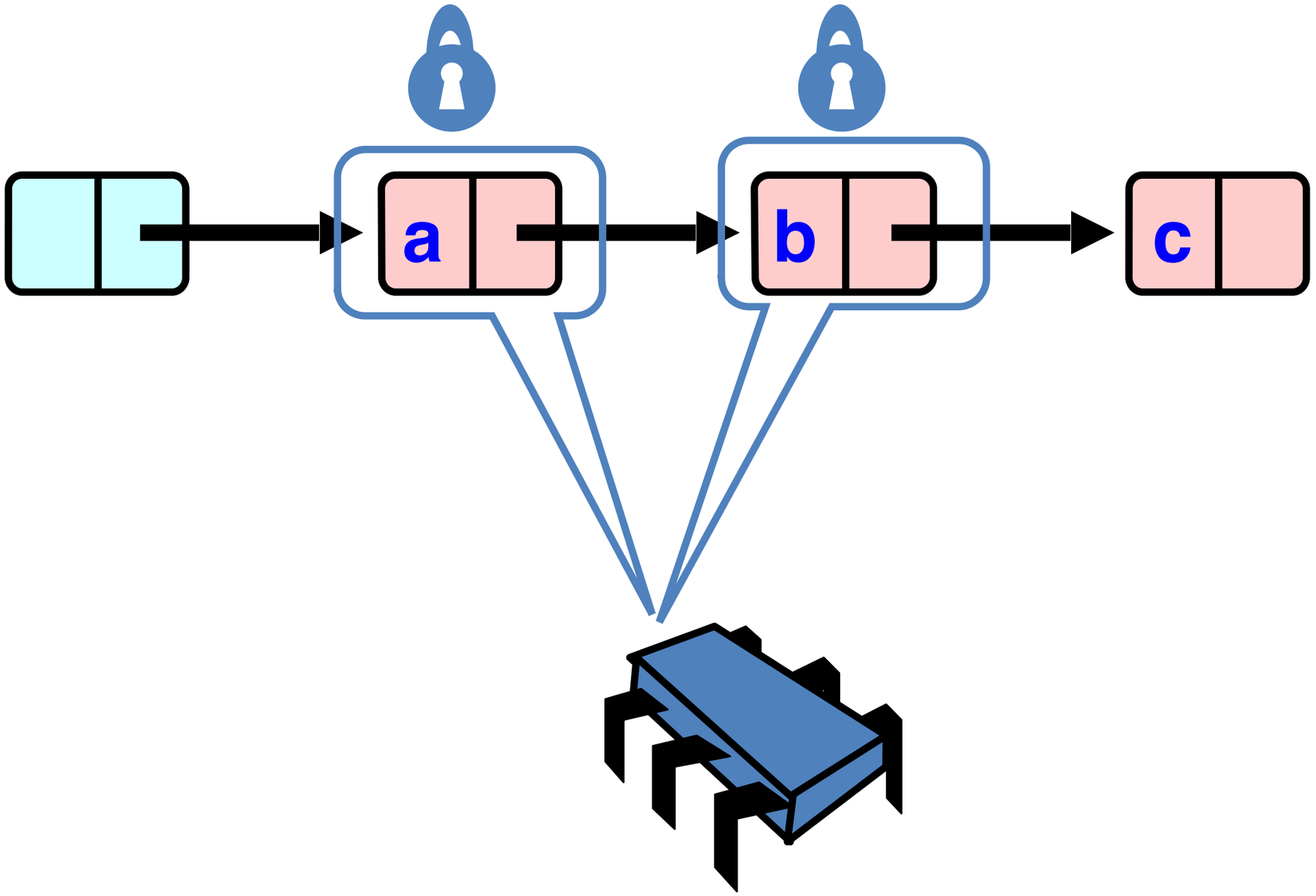
Hand-over-Hand locking



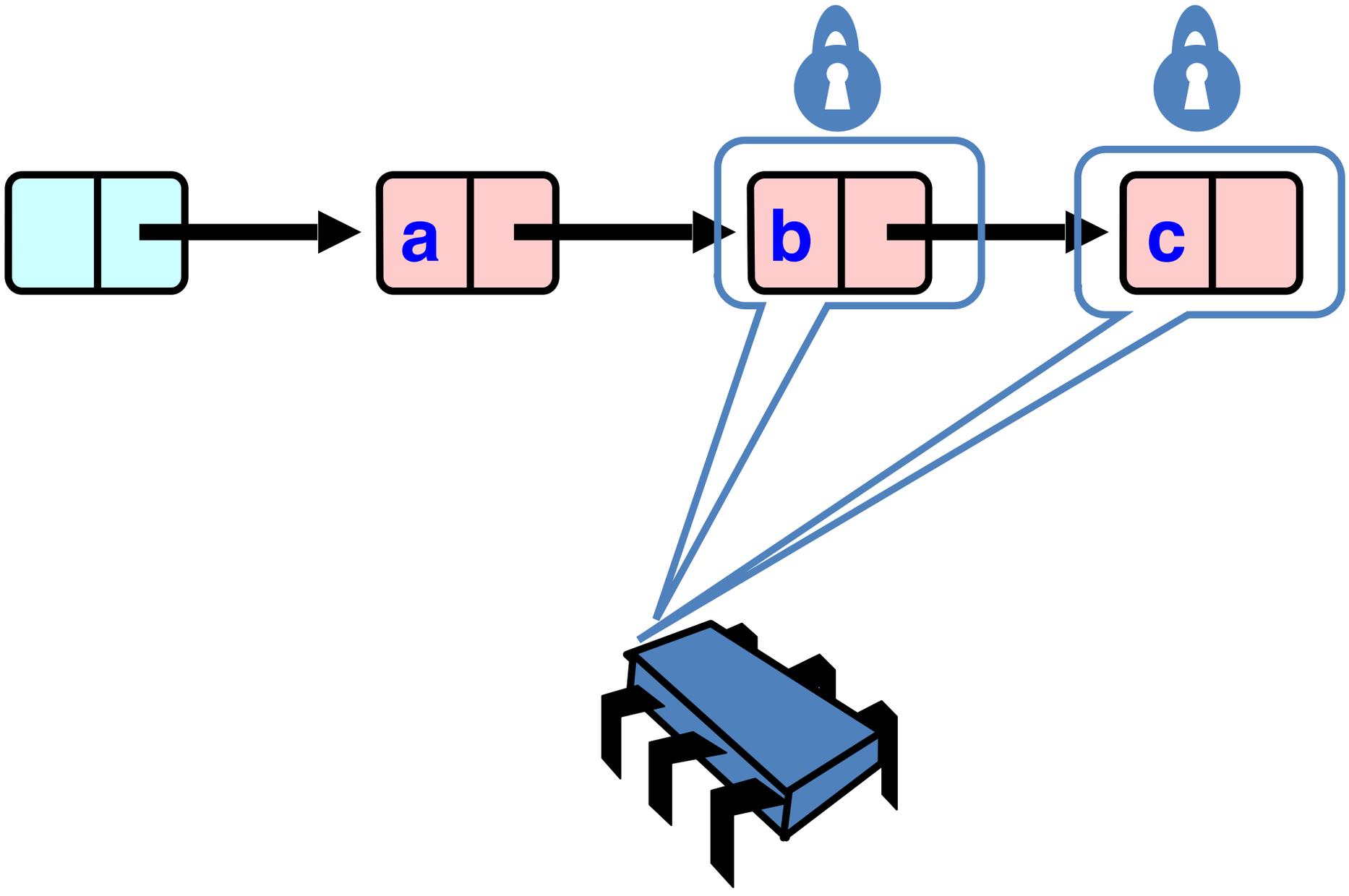
Hand-over-Hand locking



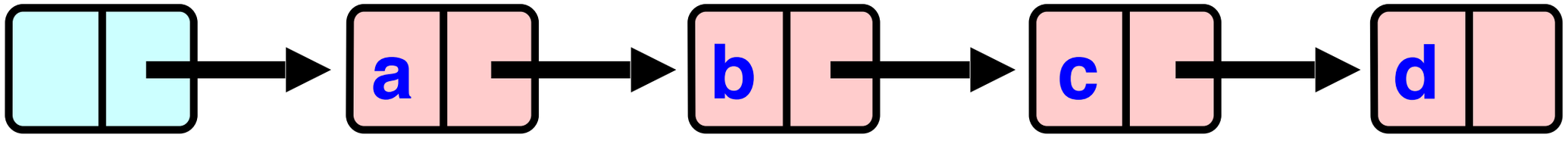
Hand-over-Hand locking

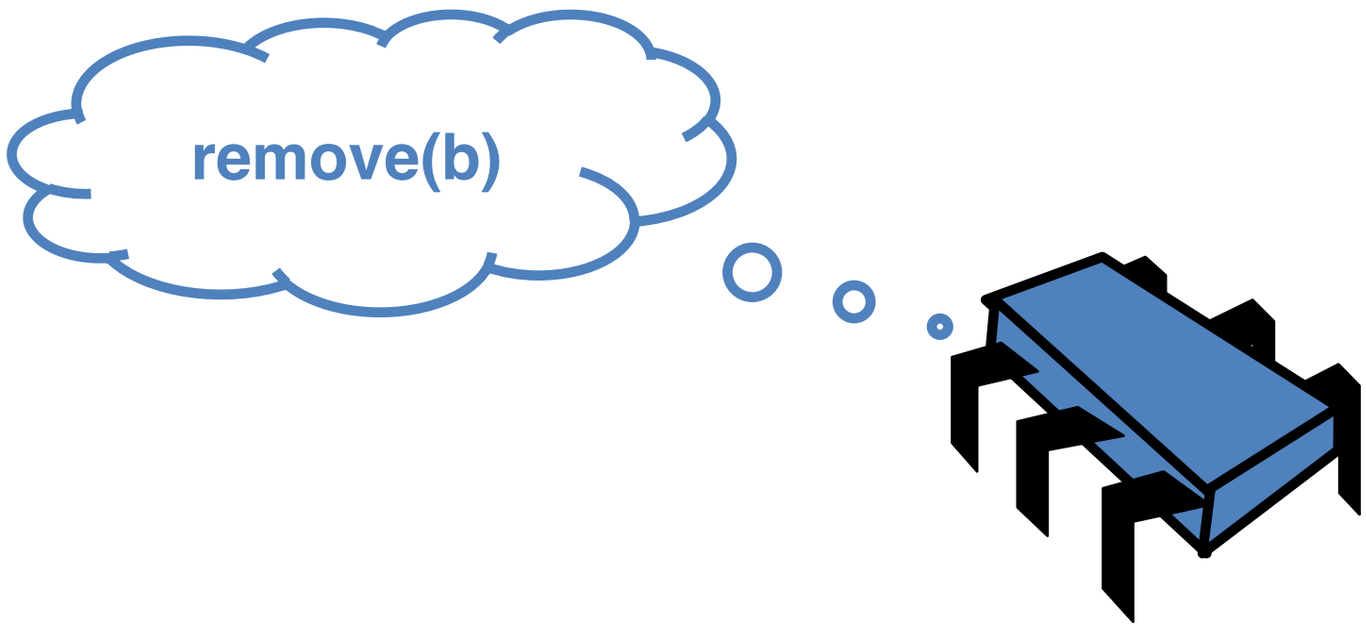


Hand-over-Hand locking

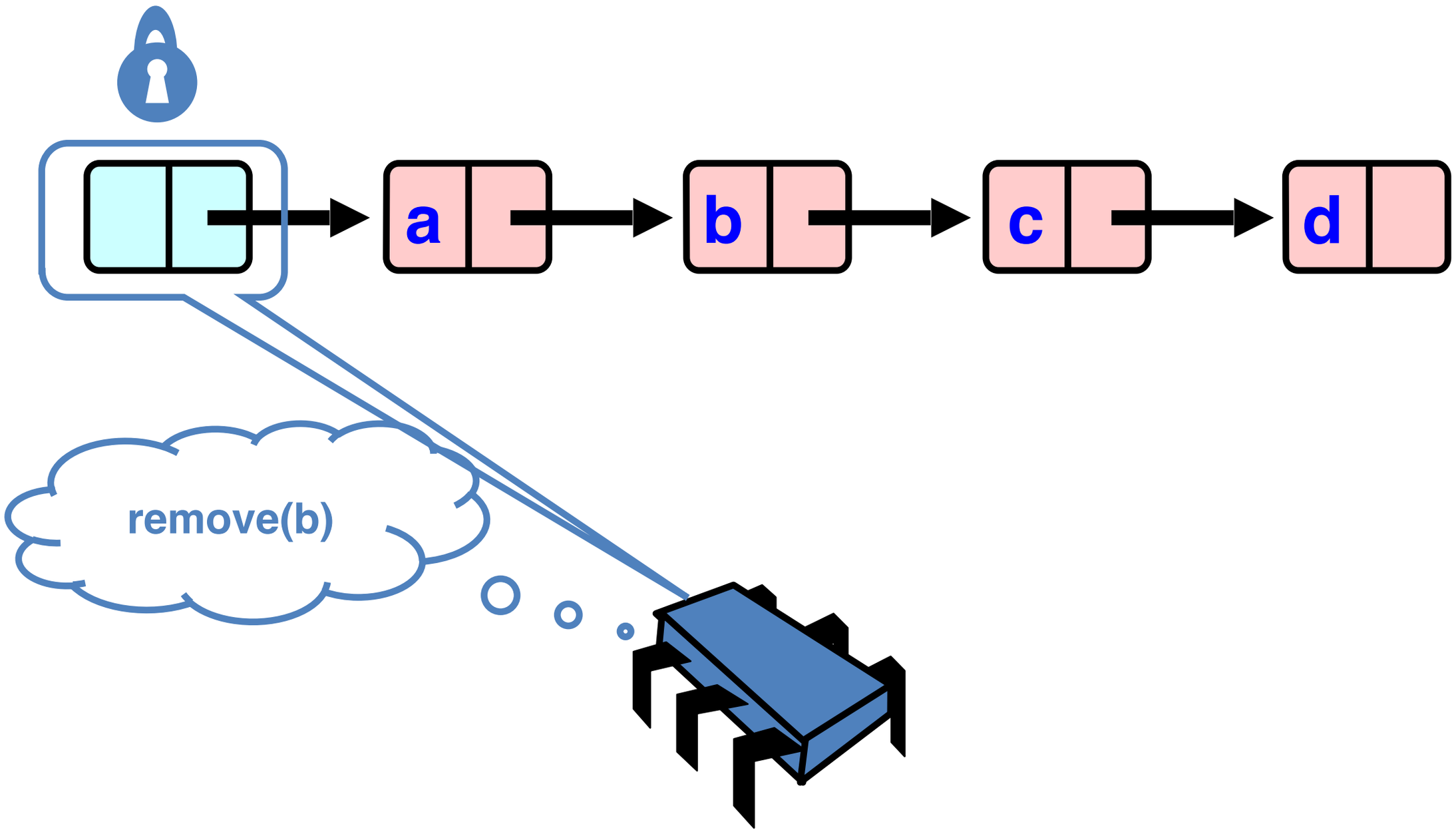


Removing a Node

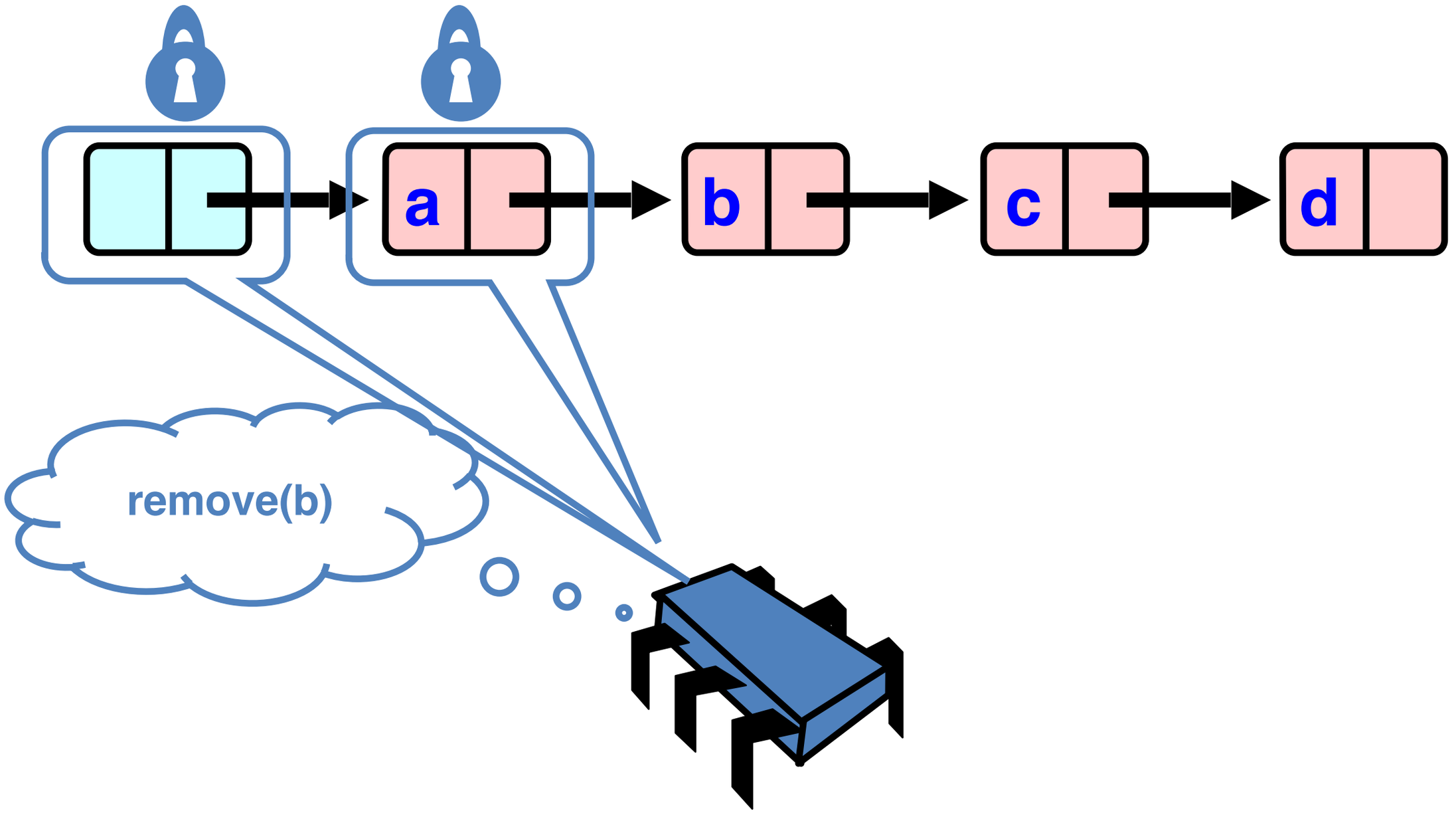




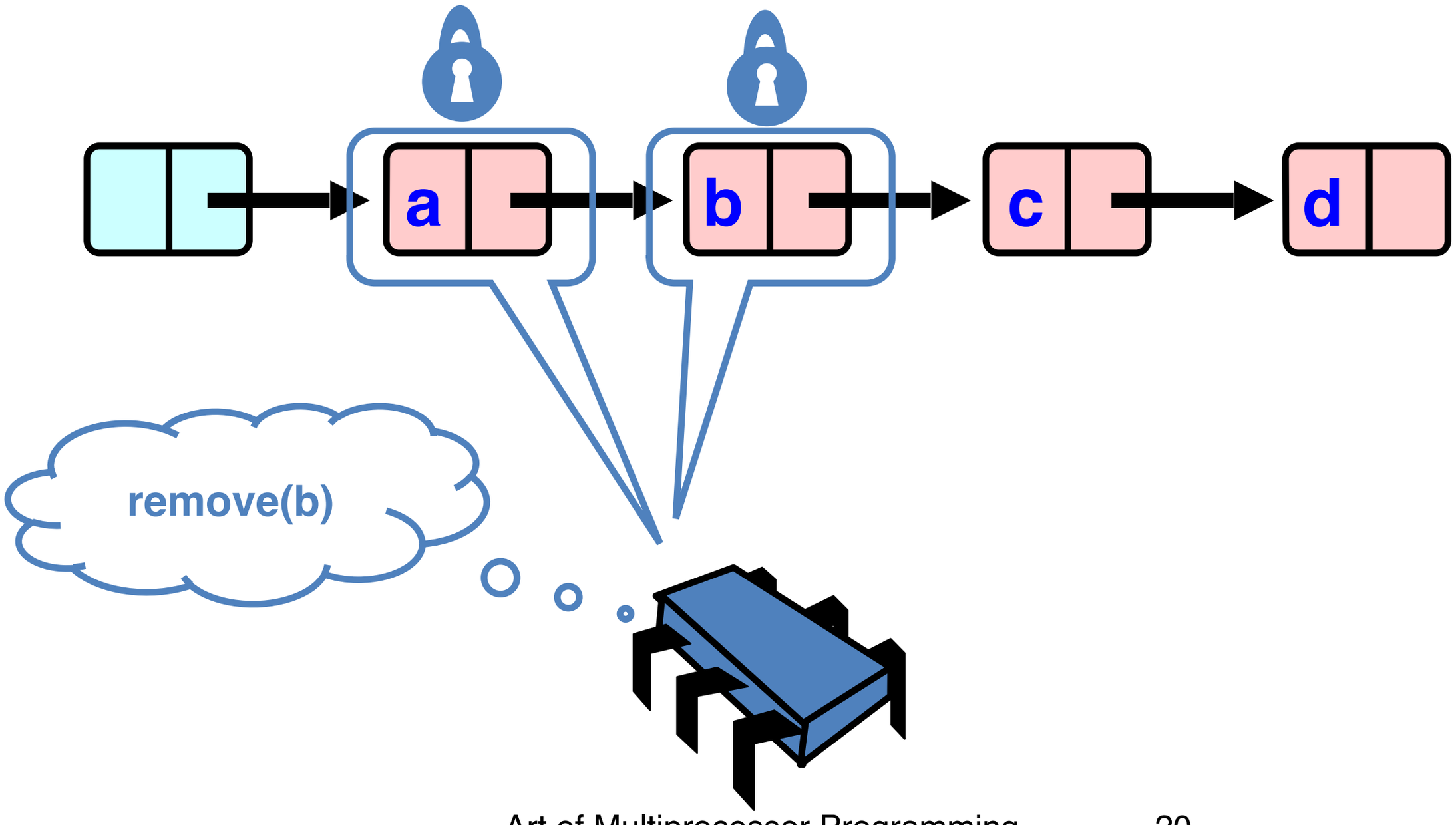
Removing a Node



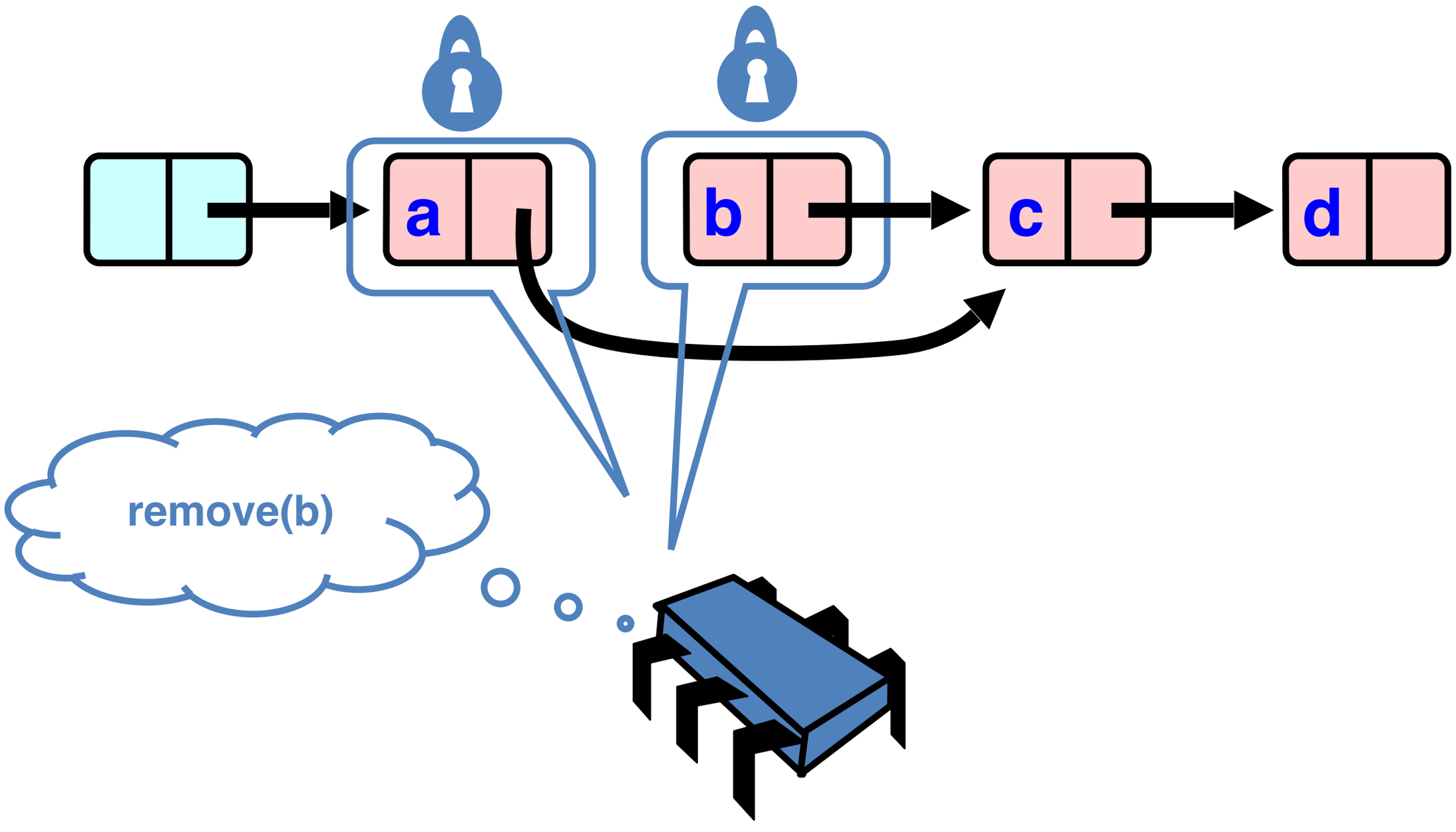
Removing a Node



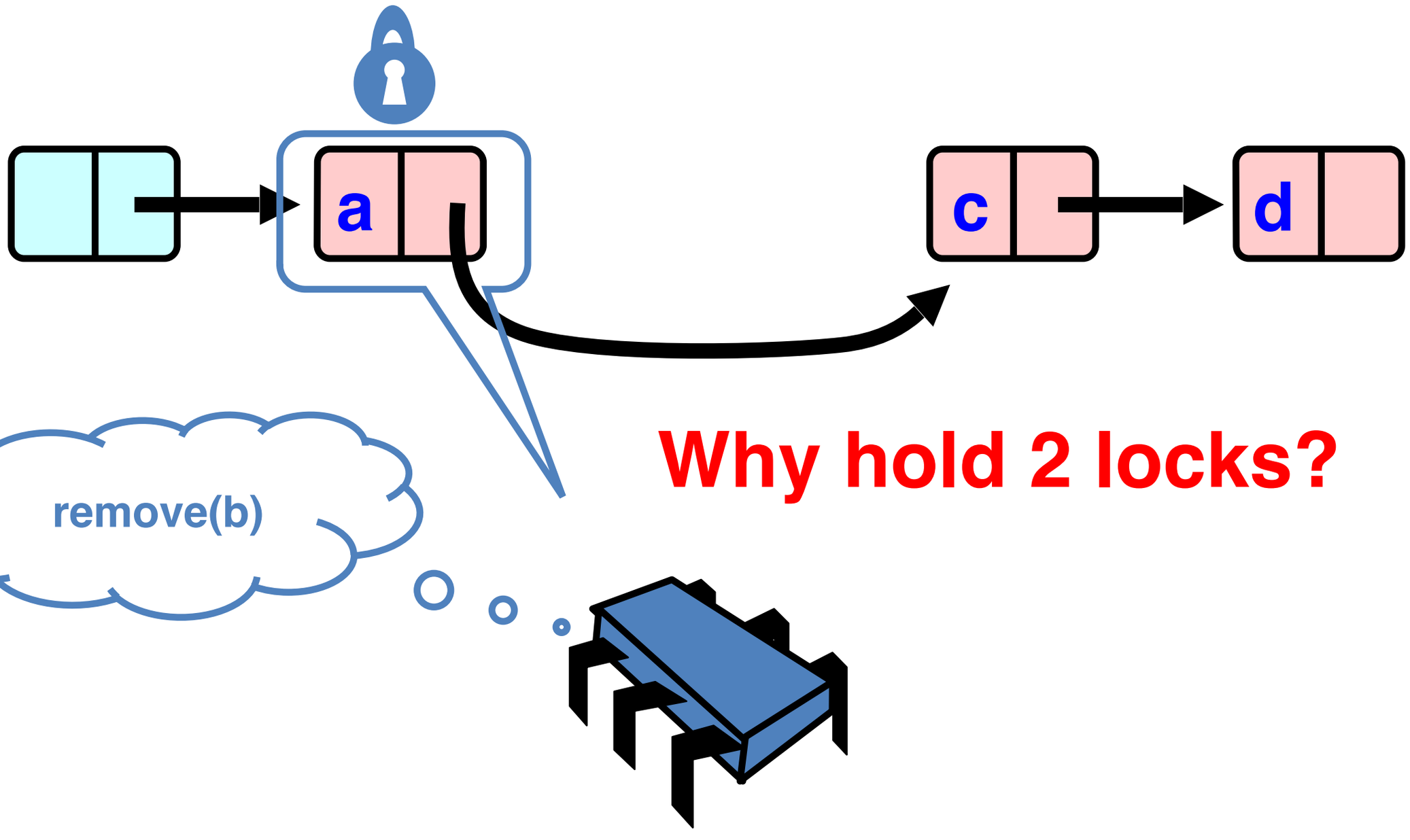
Removing a Node



Removing a Node

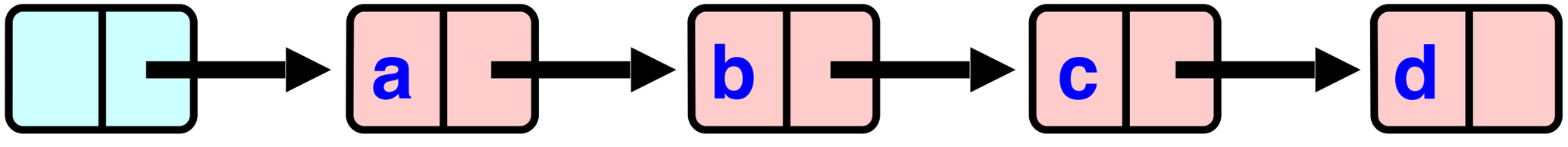


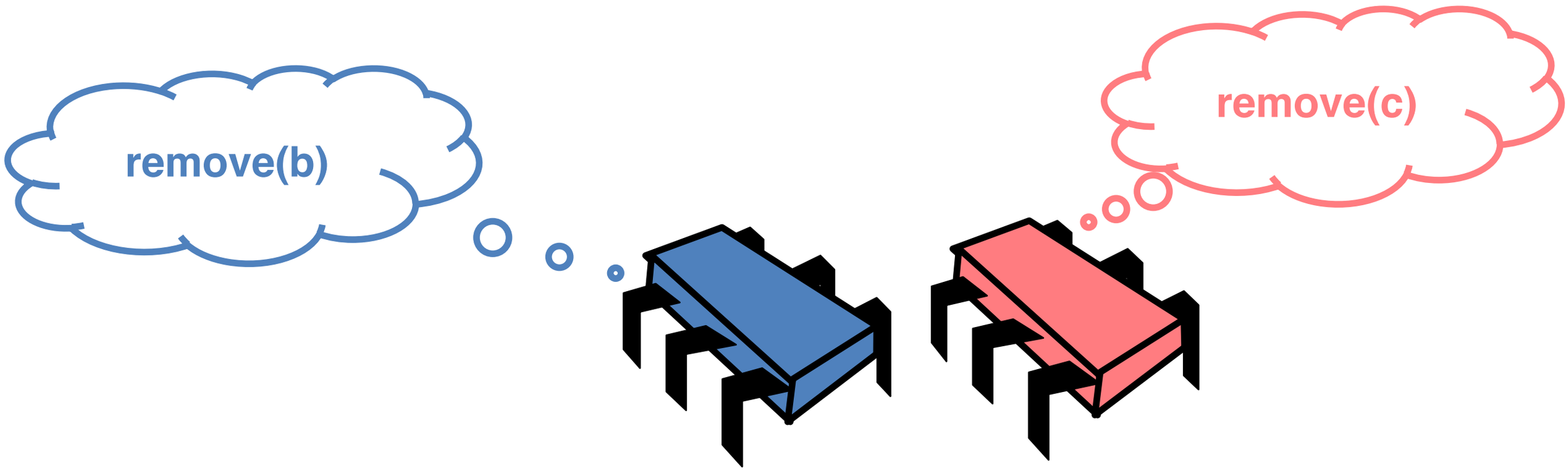
Removing a Node



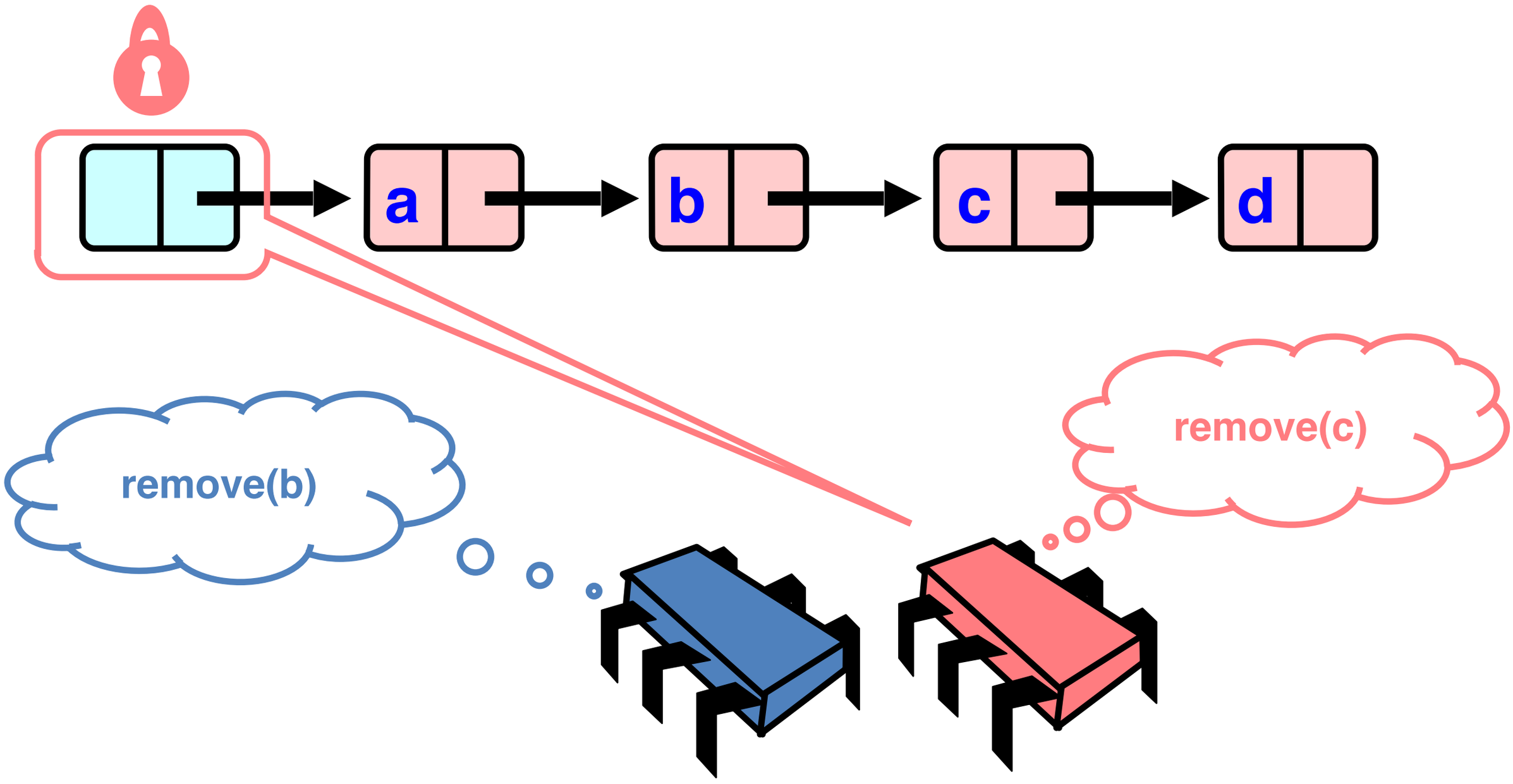
Why hold 2 locks?

Concurrent Removes

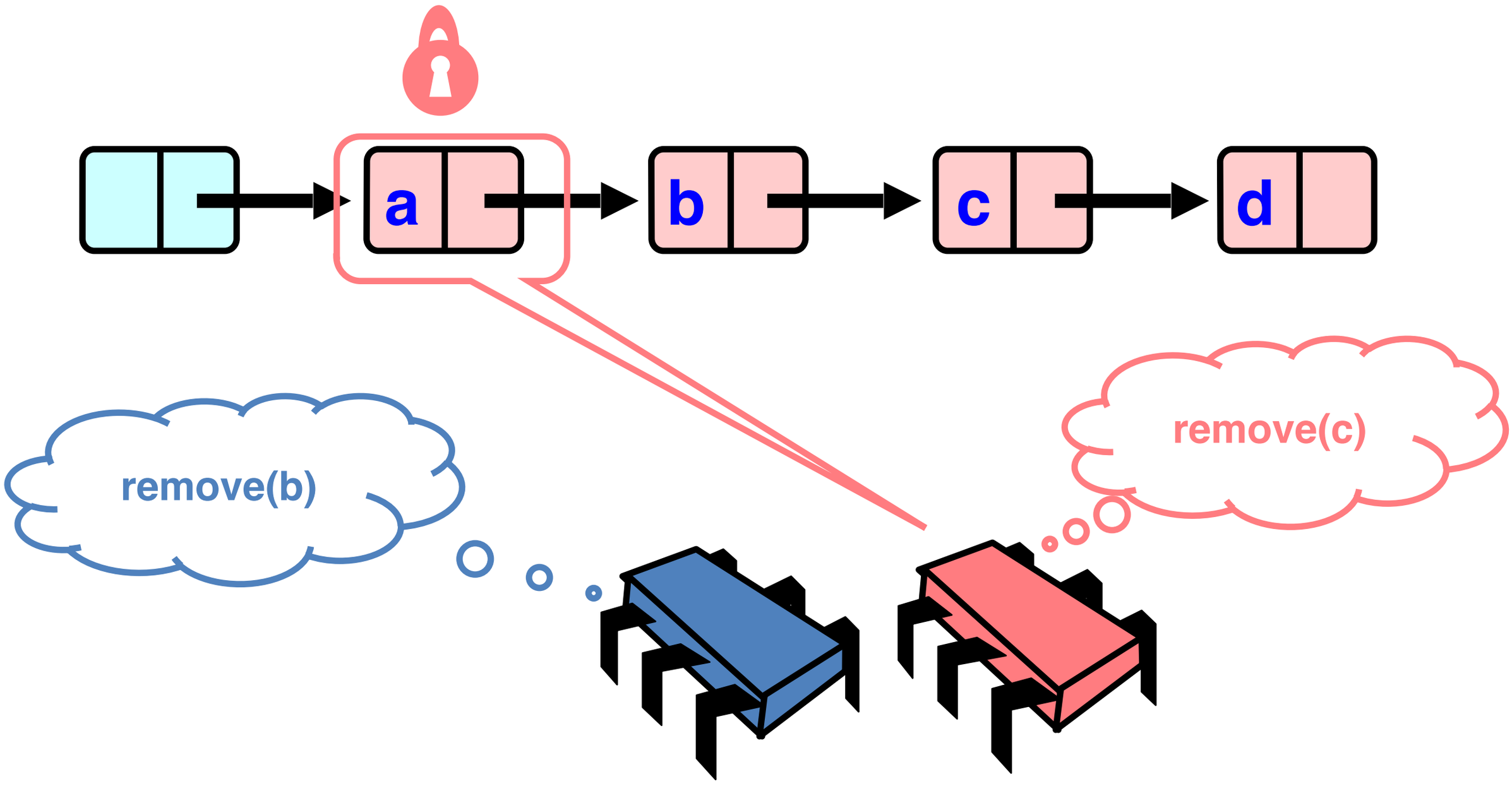




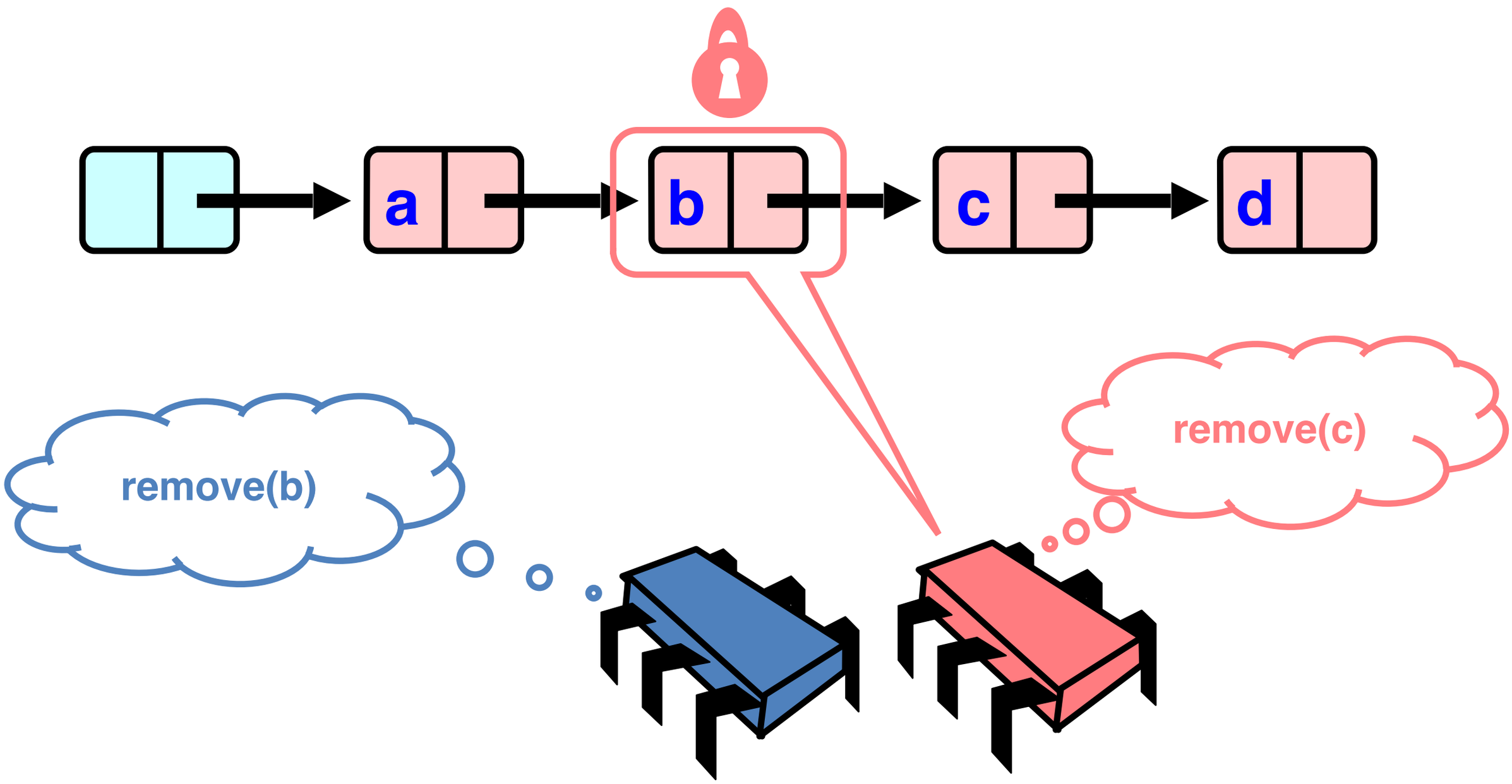
Concurrent Removes



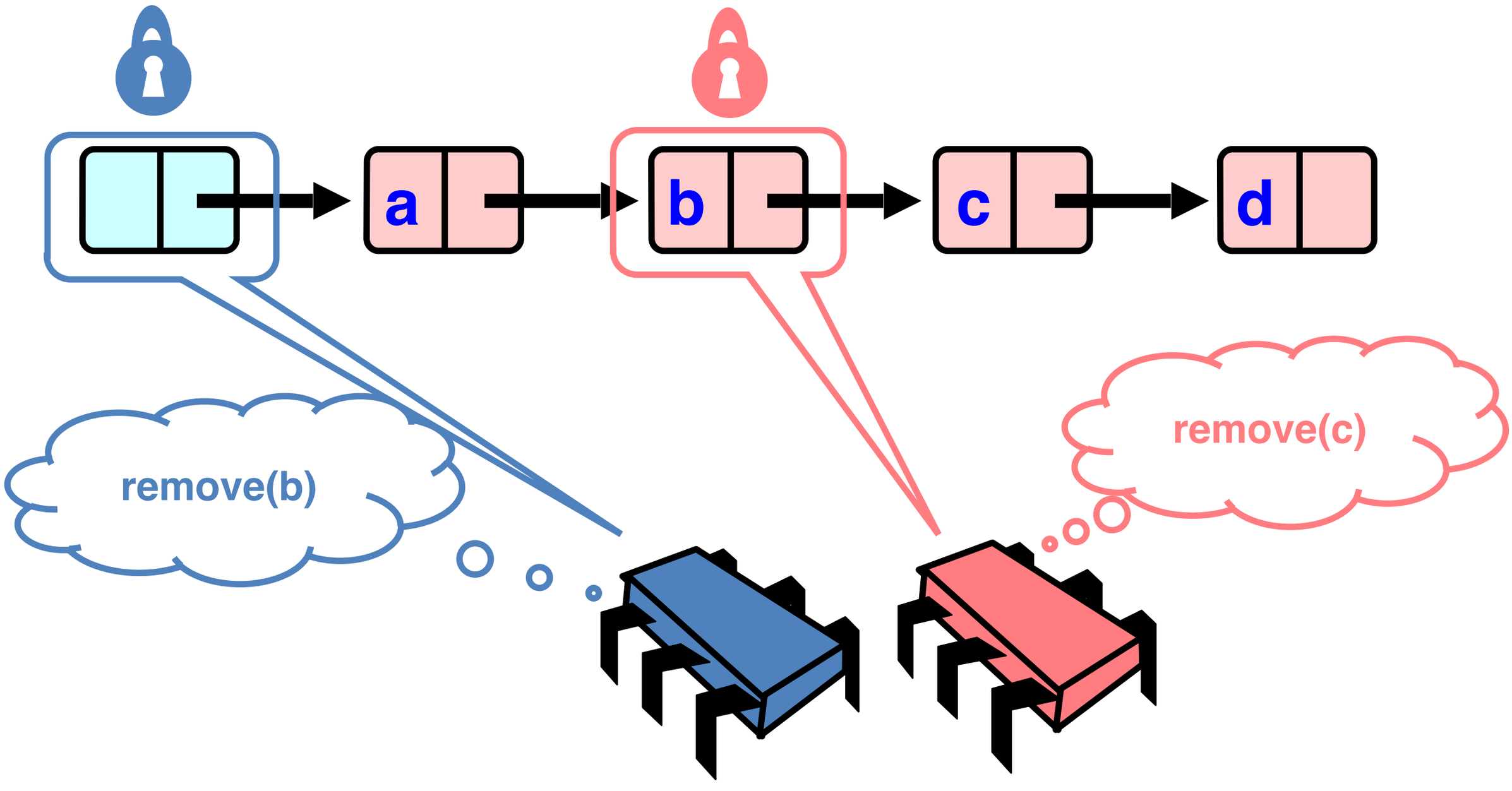
Concurrent Removes



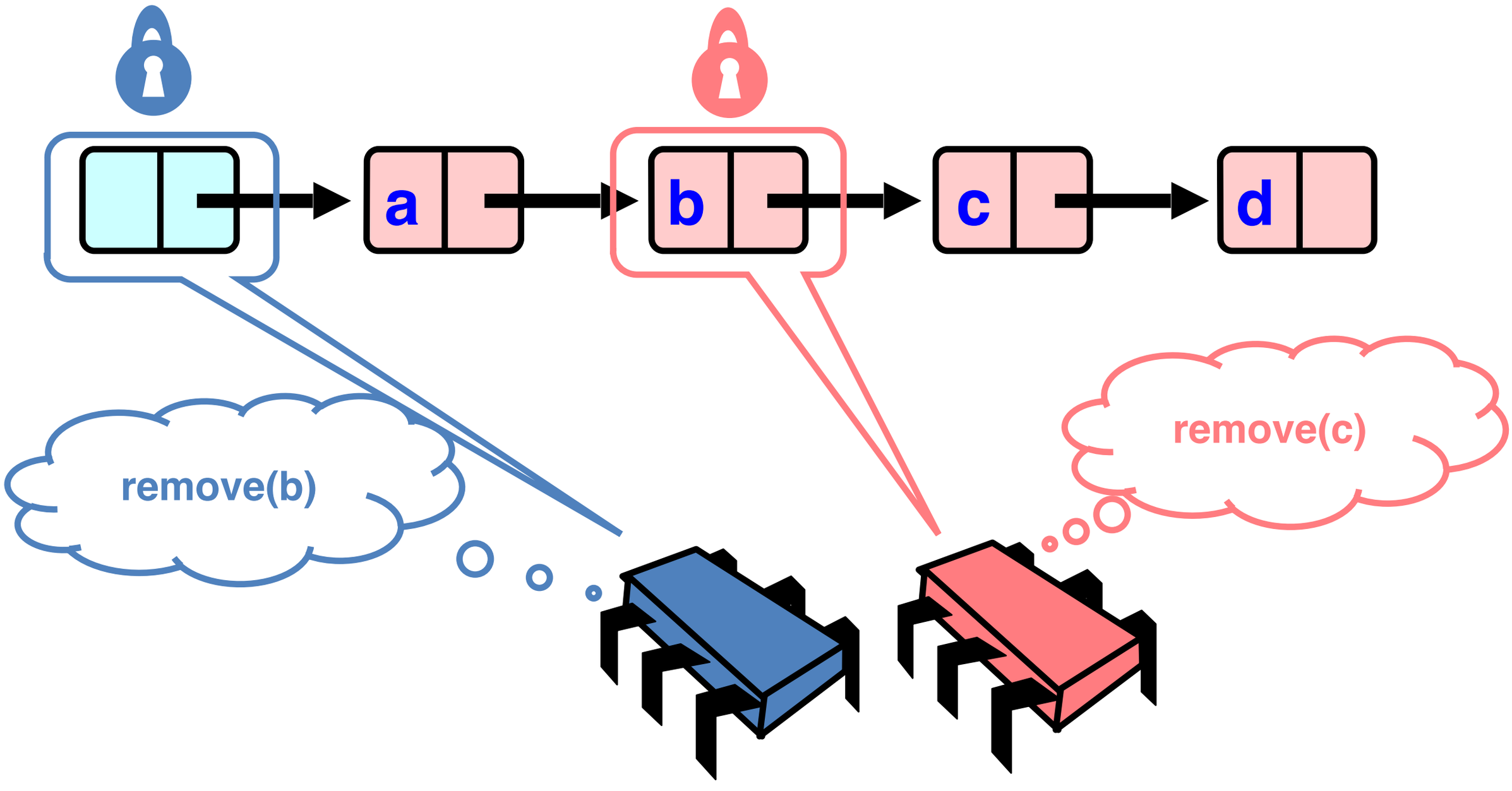
Concurrent Removes



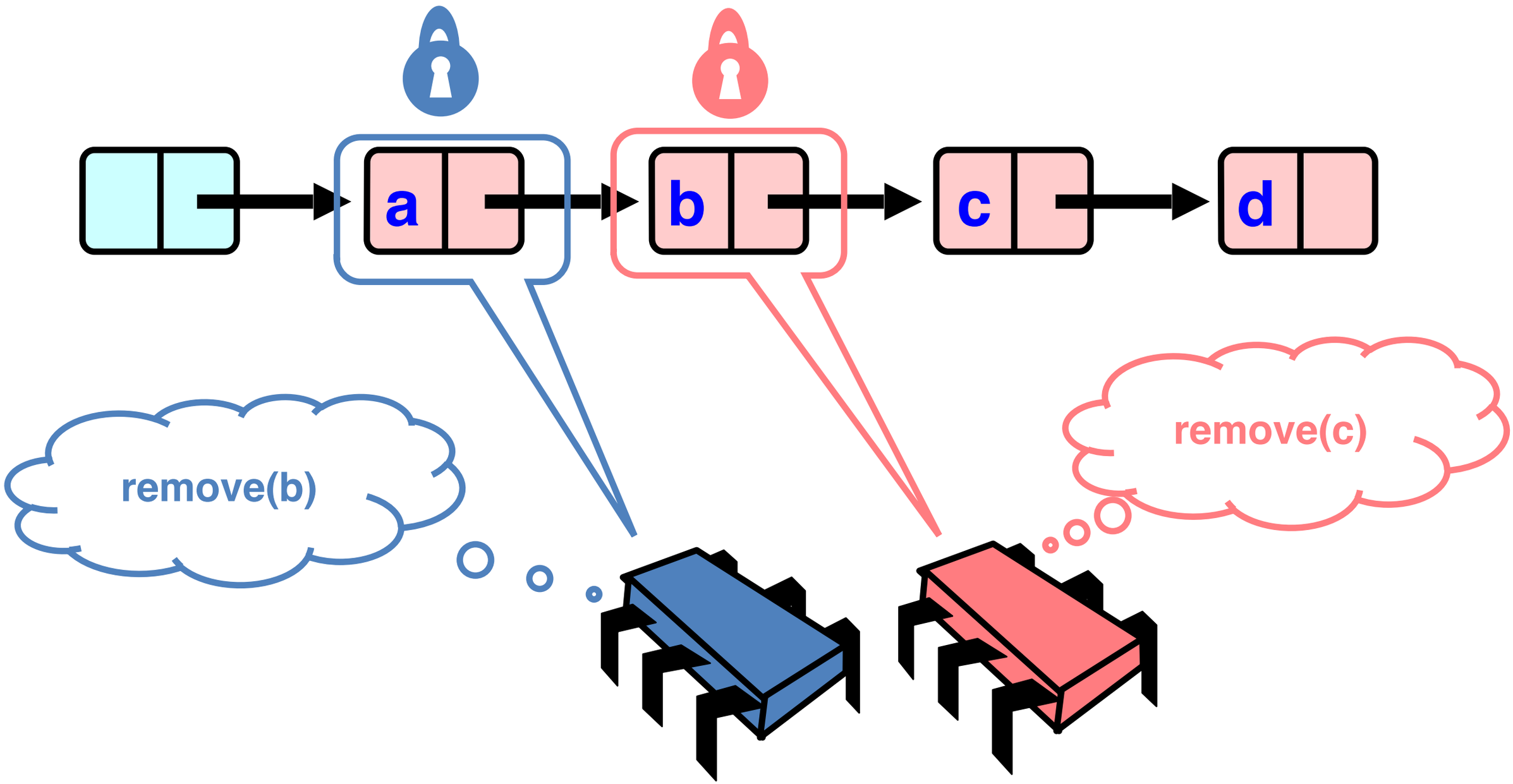
Concurrent Removes



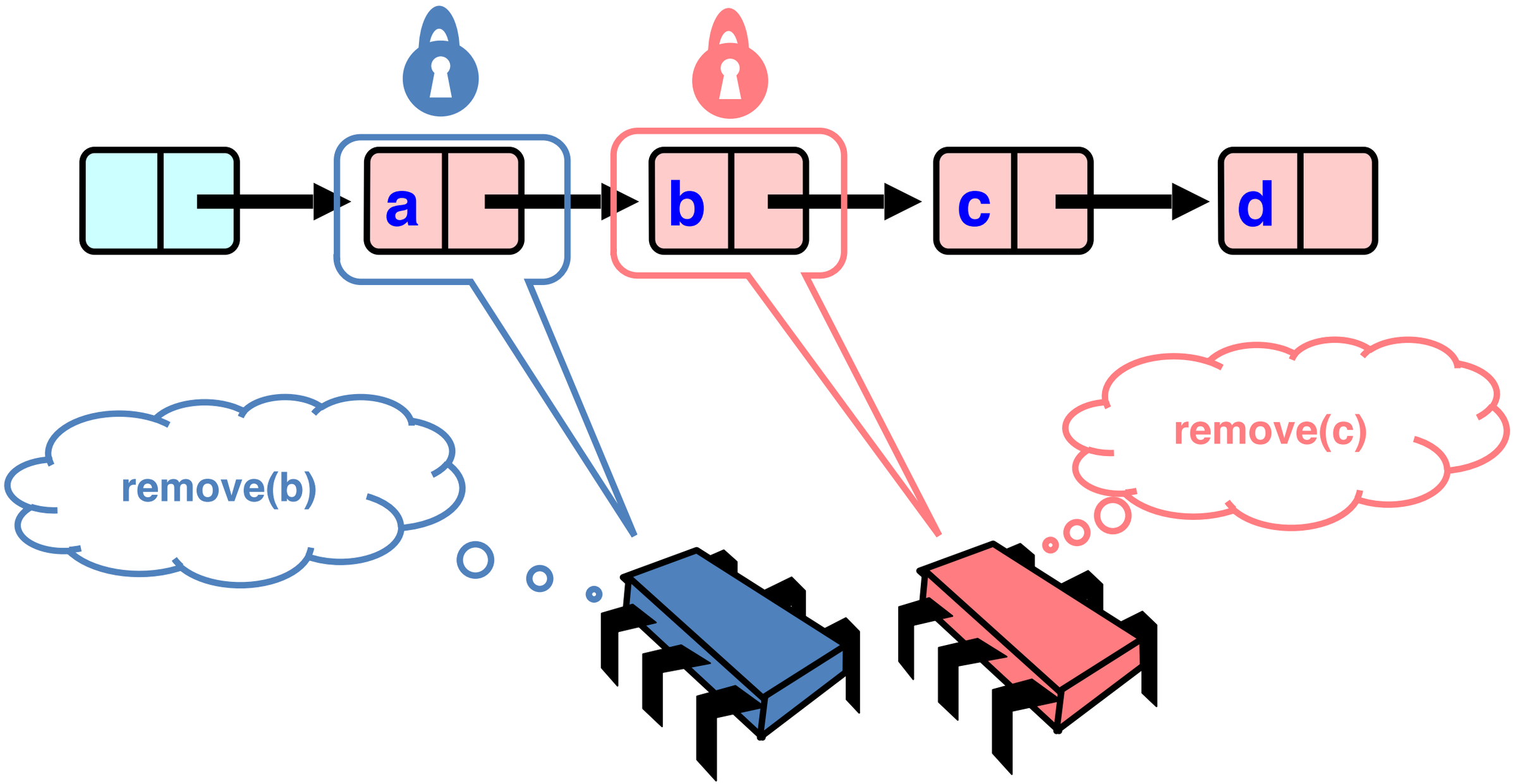
Concurrent Removes



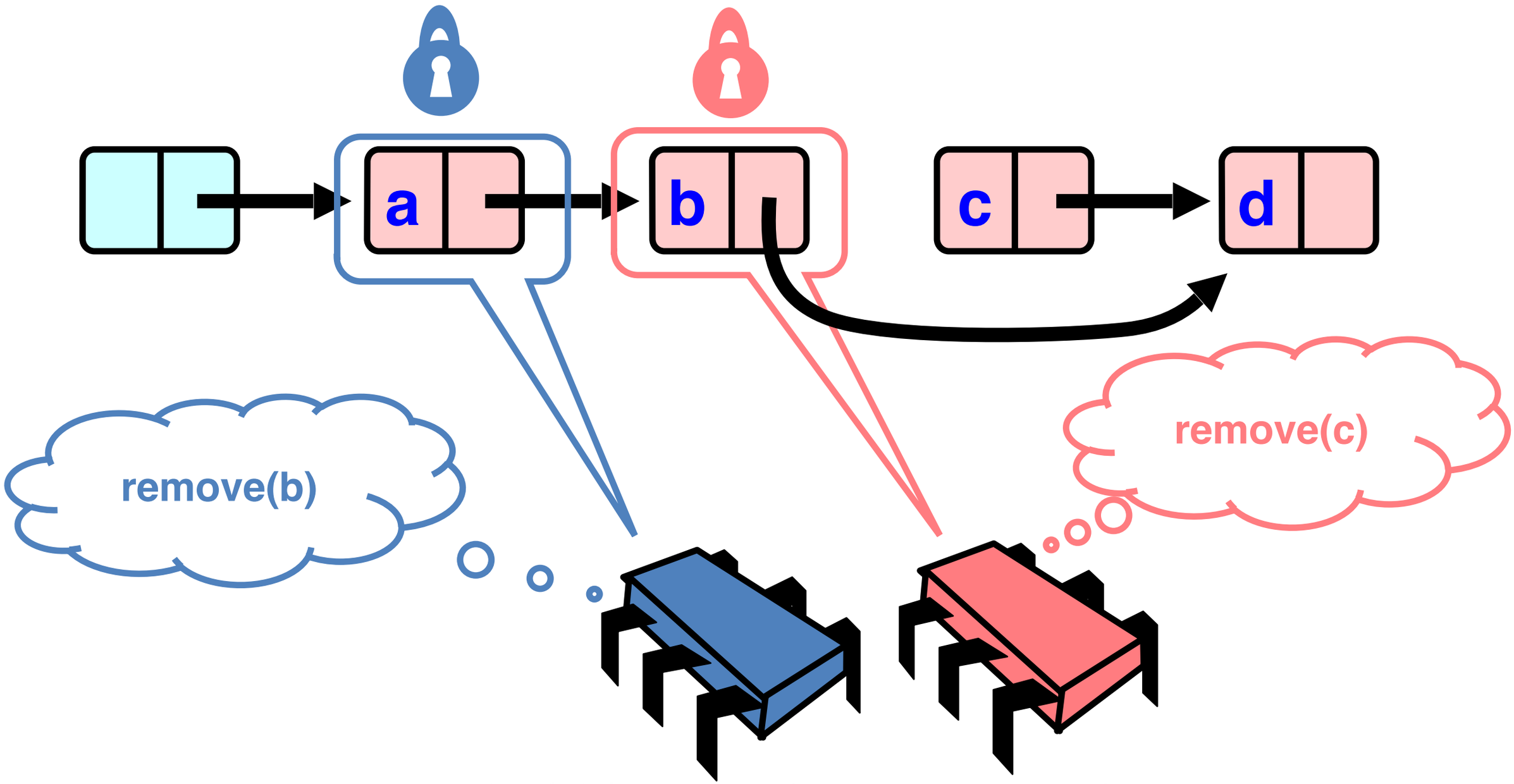
Concurrent Removes



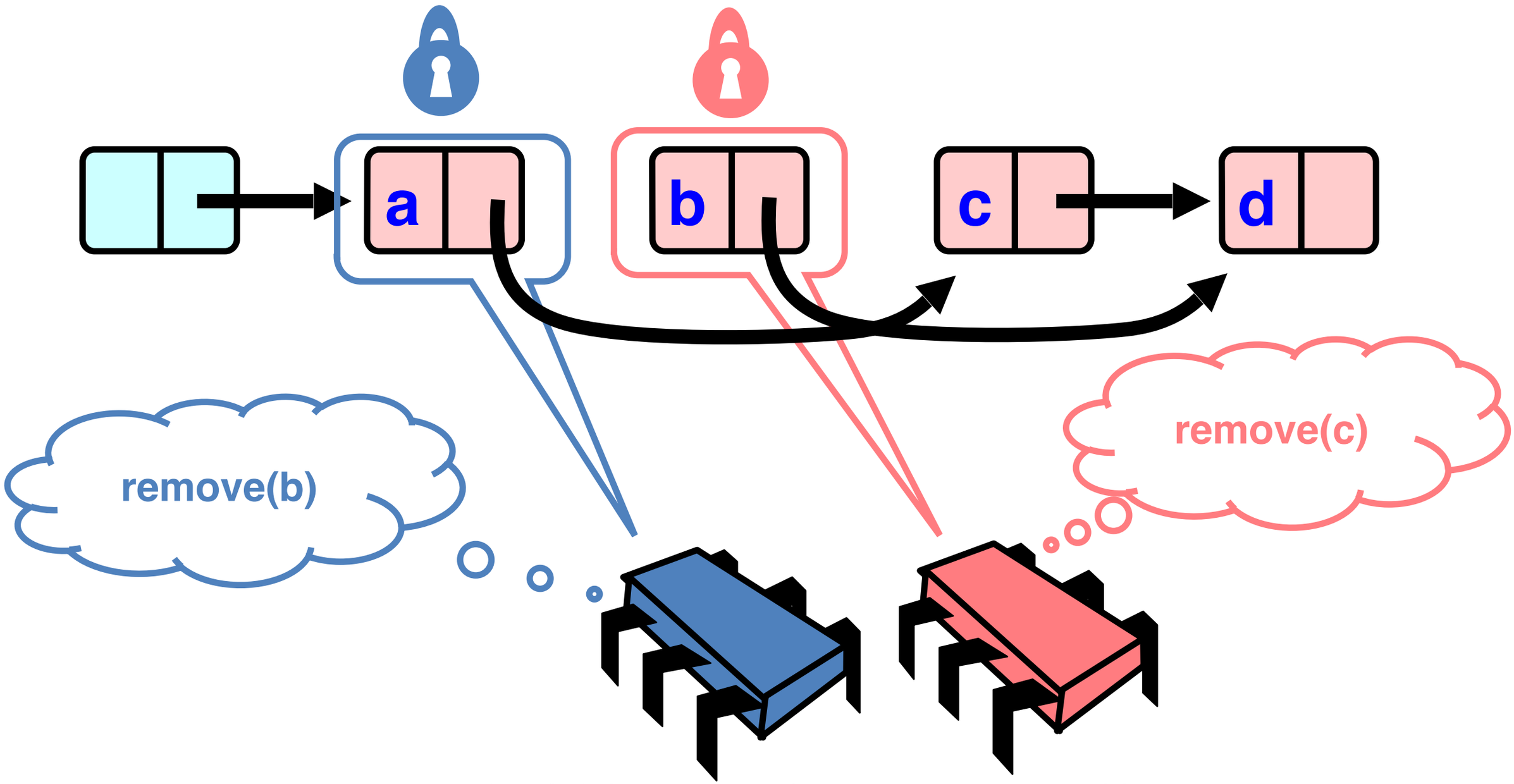
Concurrent Removes



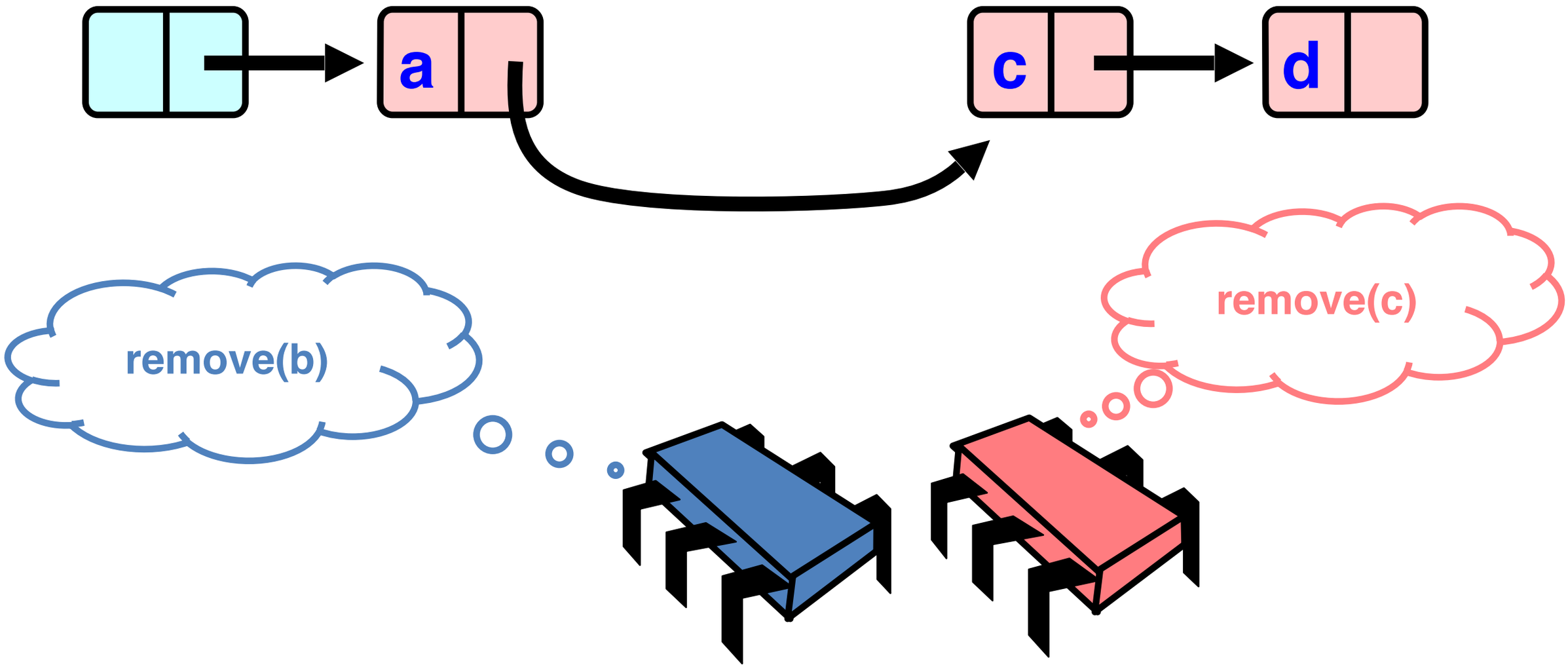
Concurrent Removes



Concurrent Removes

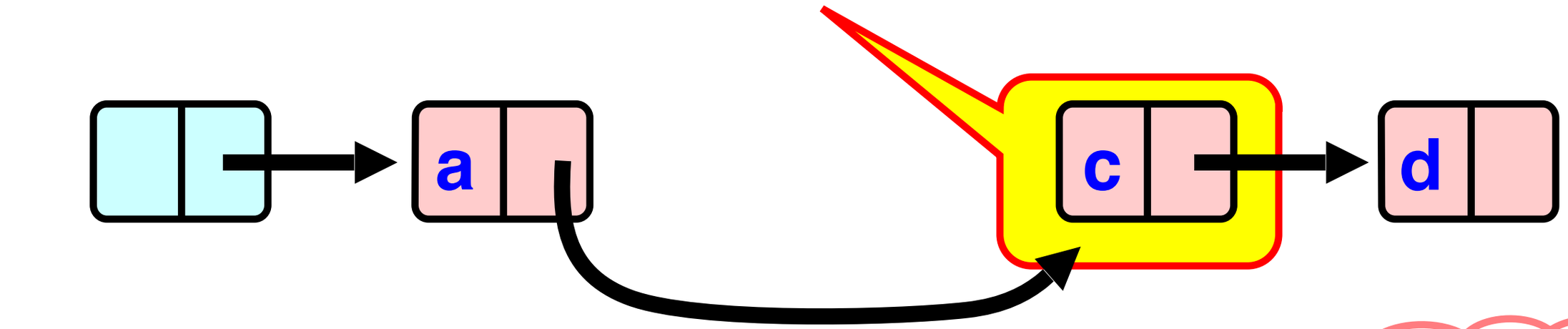


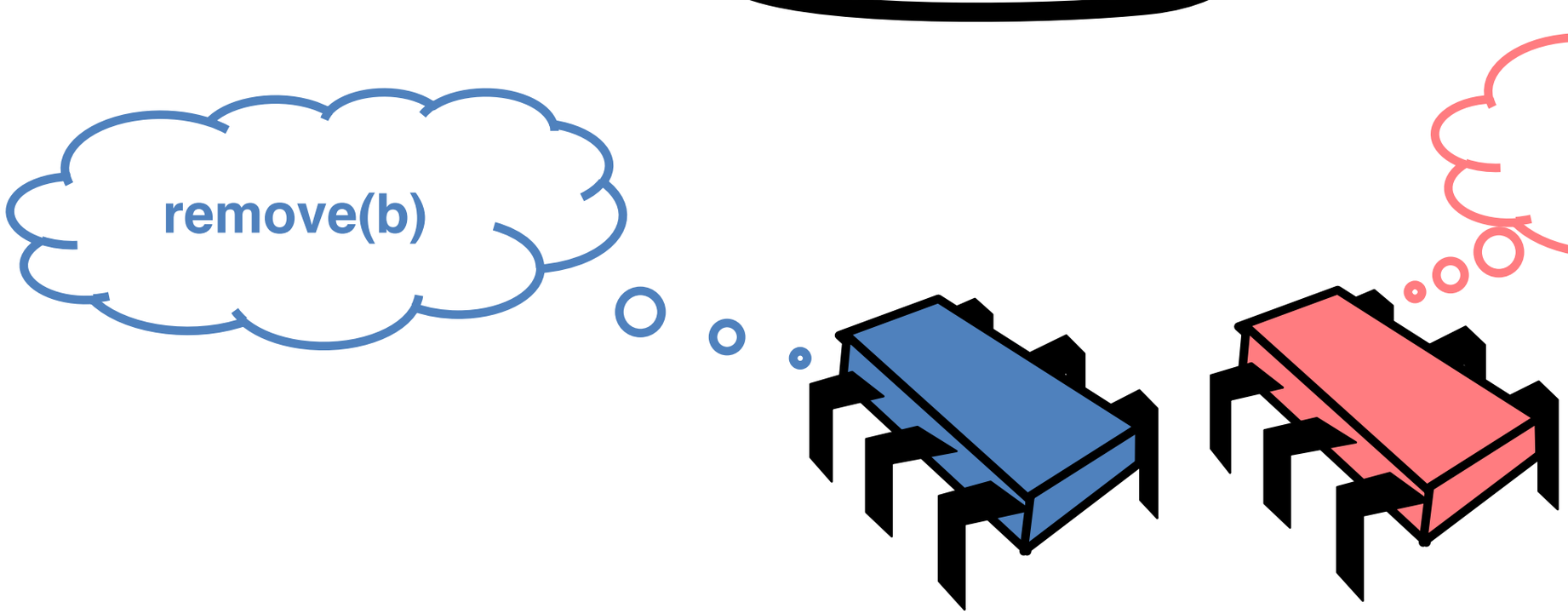
Uh, Oh



Uh, Oh

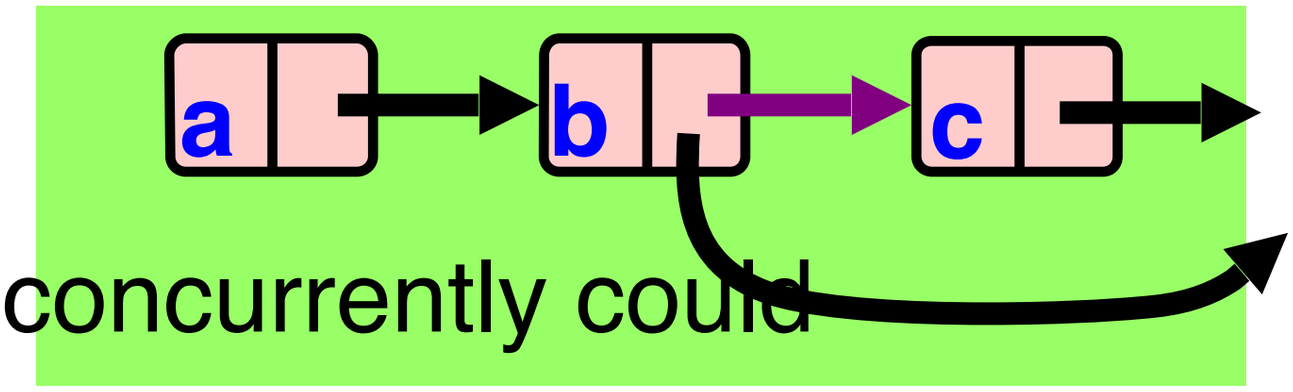
Bad news, c not removed

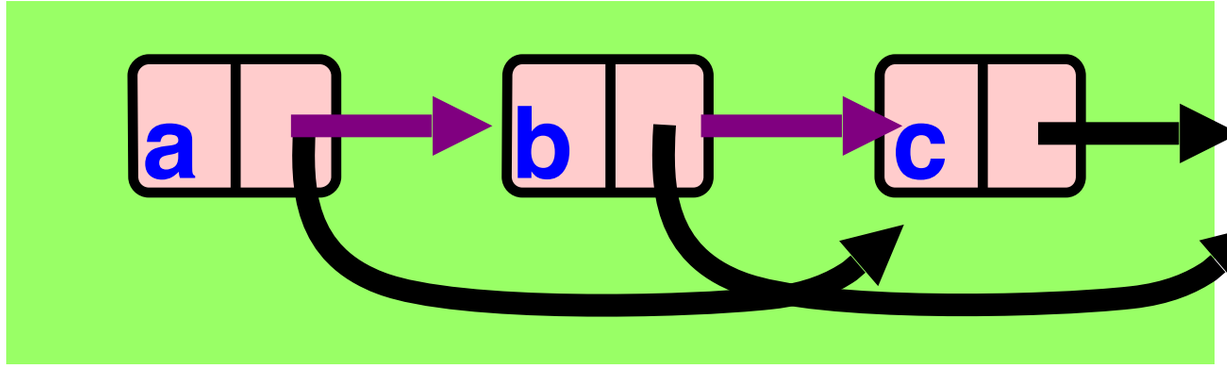




Problem

- To delete node c
  - Swing node b's next field to d
- Problem is,
  - Someone deleting b concurrently could direct a pointer to c

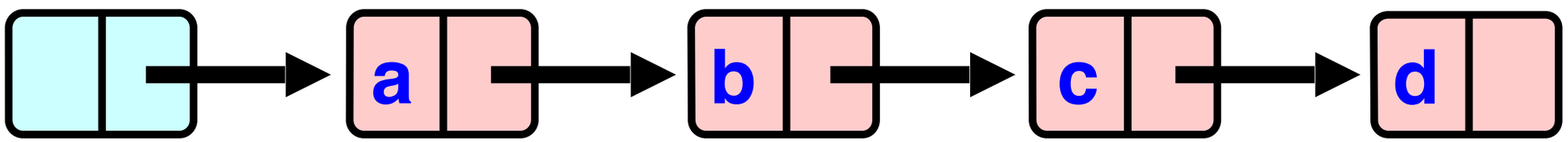


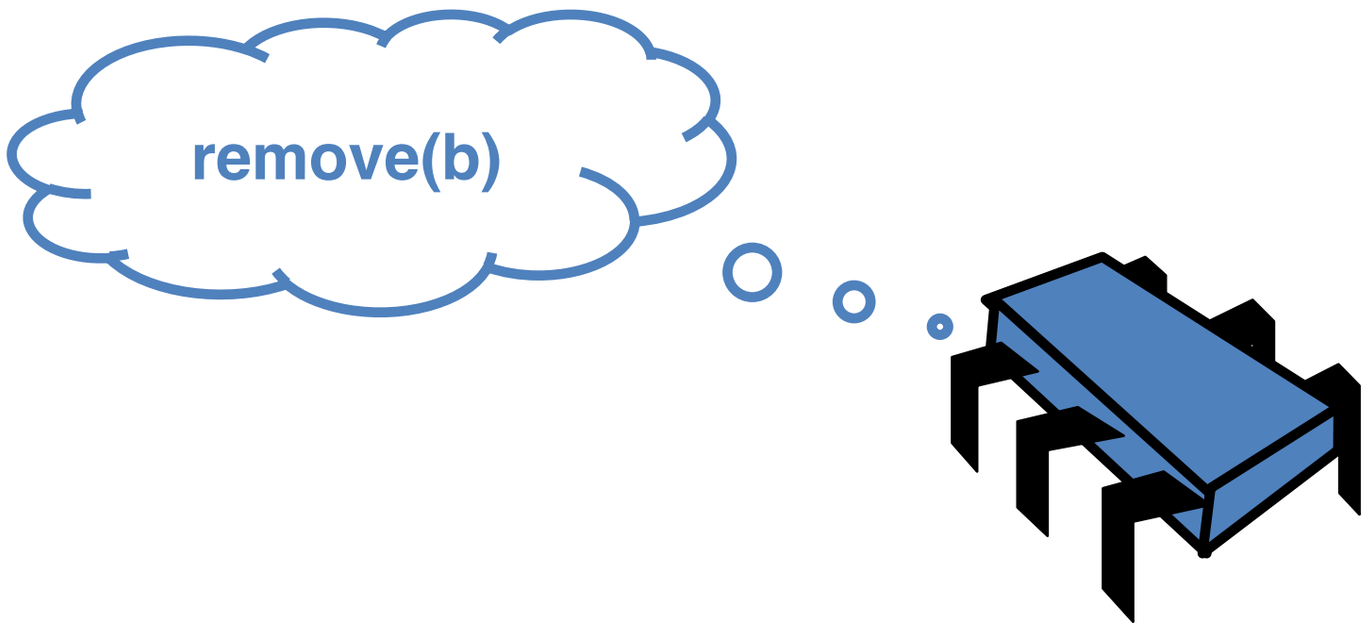


Insight

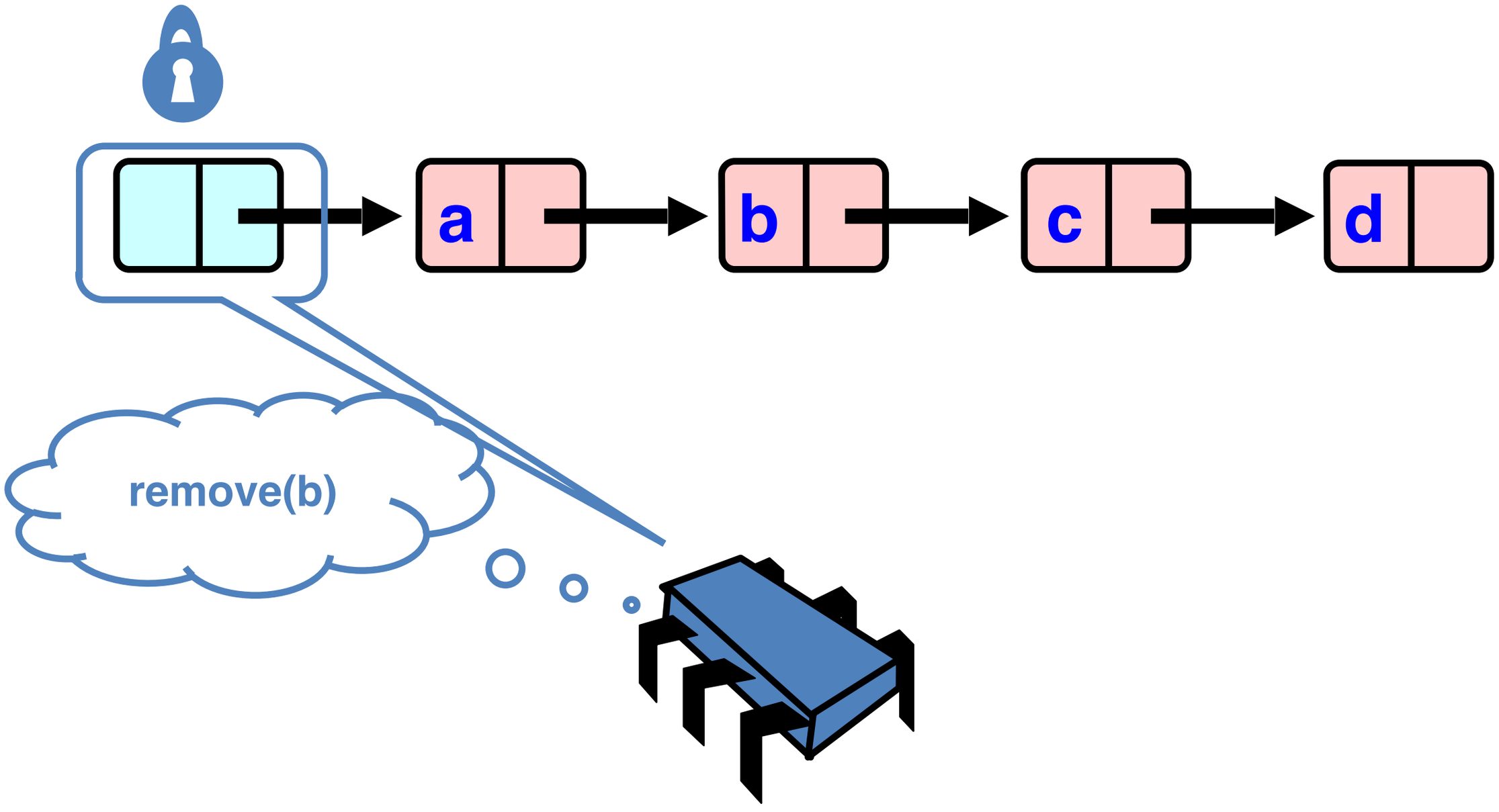
- If a node is locked
  - No one can delete node's *successor*
- If a thread locks
  - Node to be deleted
  - And its predecessor
  - Then it works

Hand-Over-Hand Again

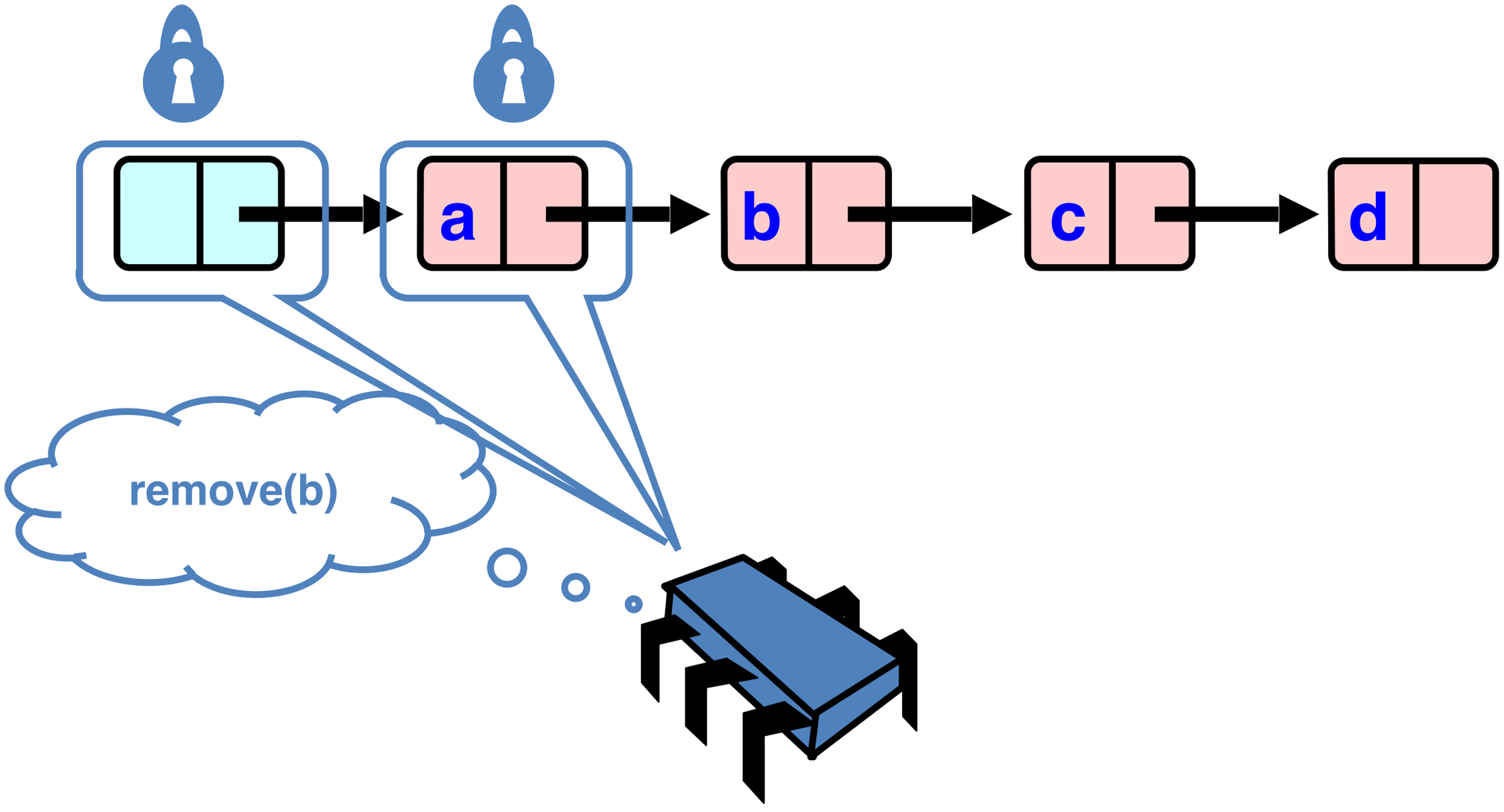




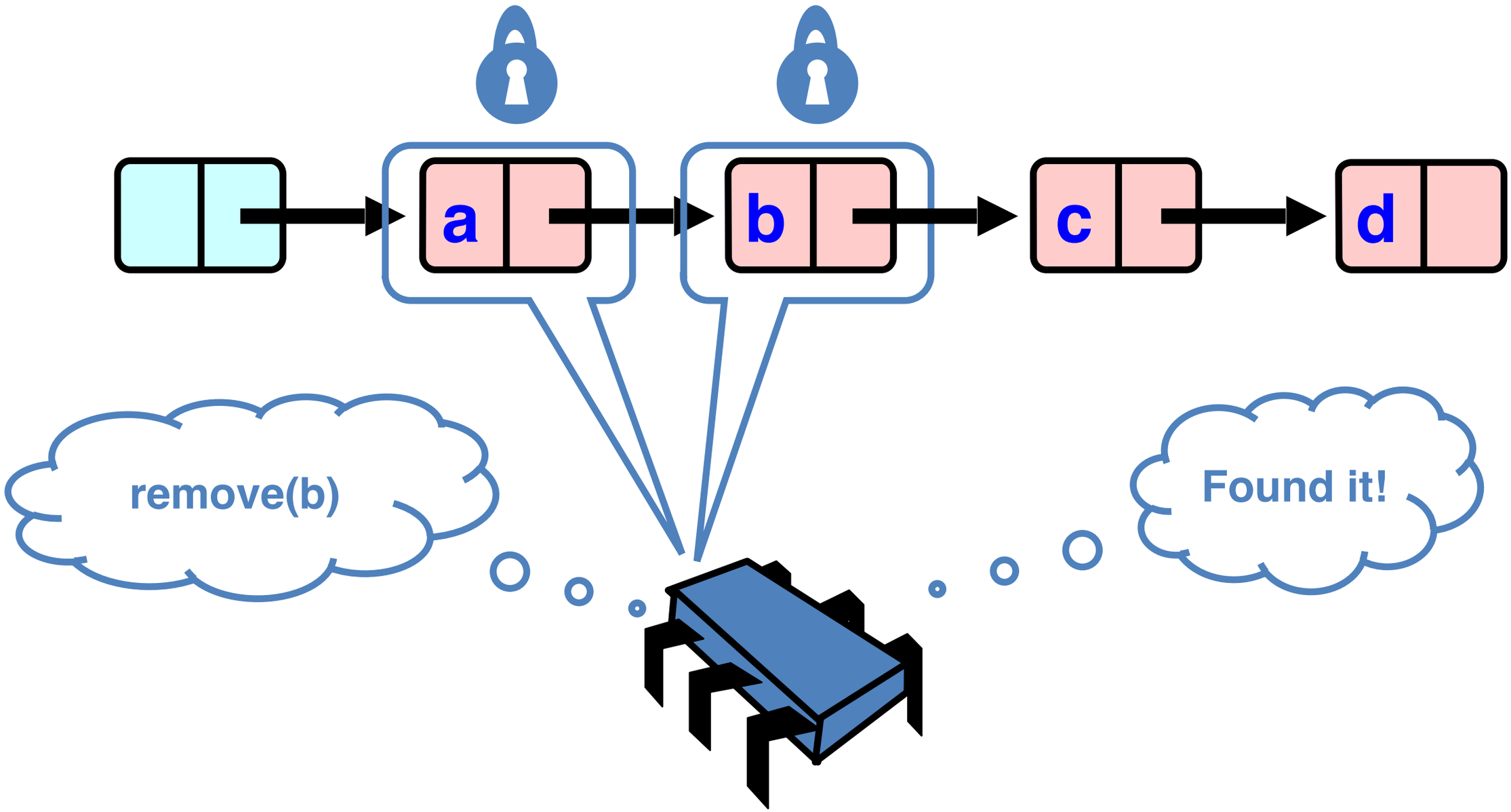
Hand-Over-Hand Again



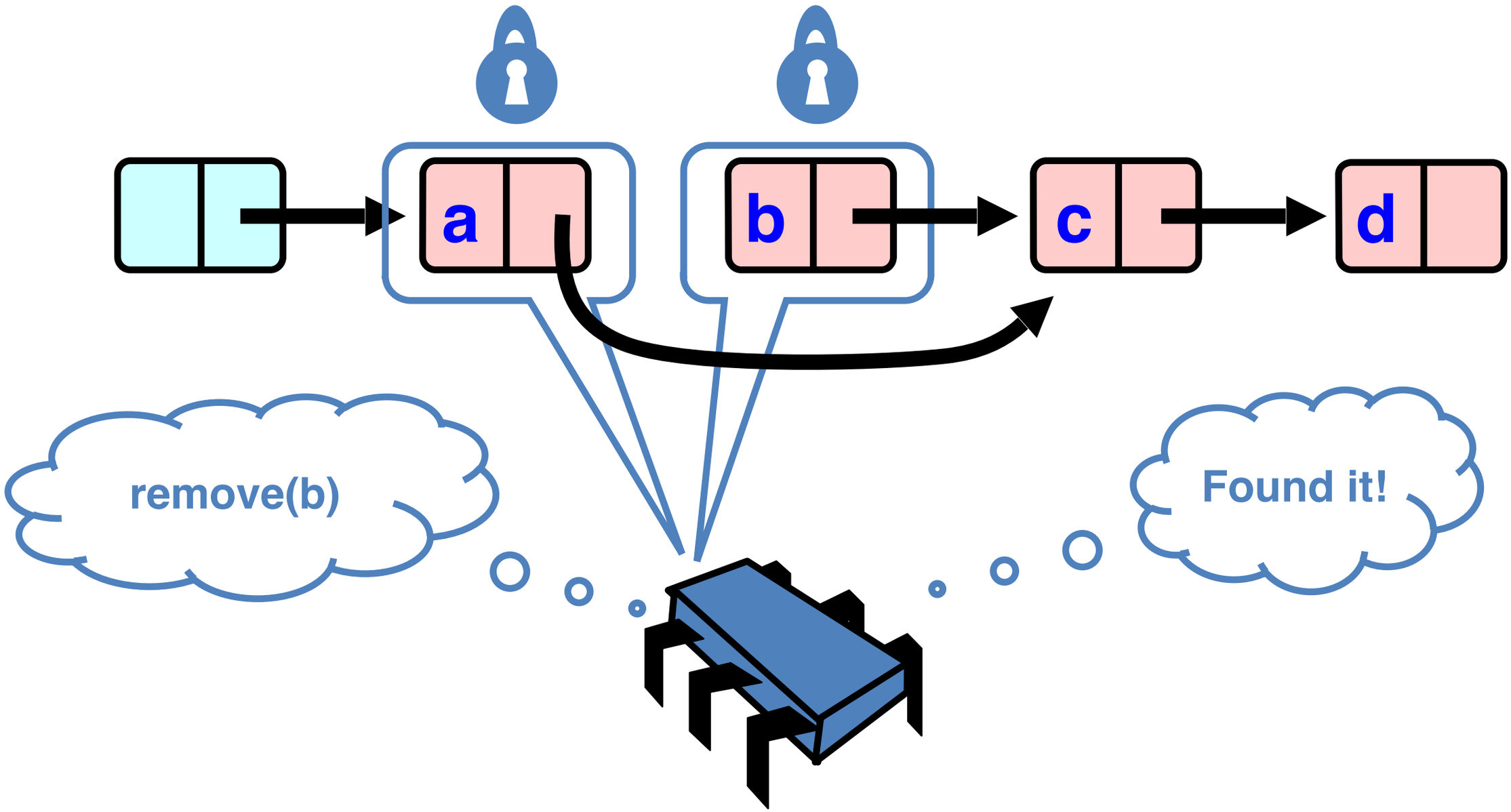
Hand-Over-Hand Again



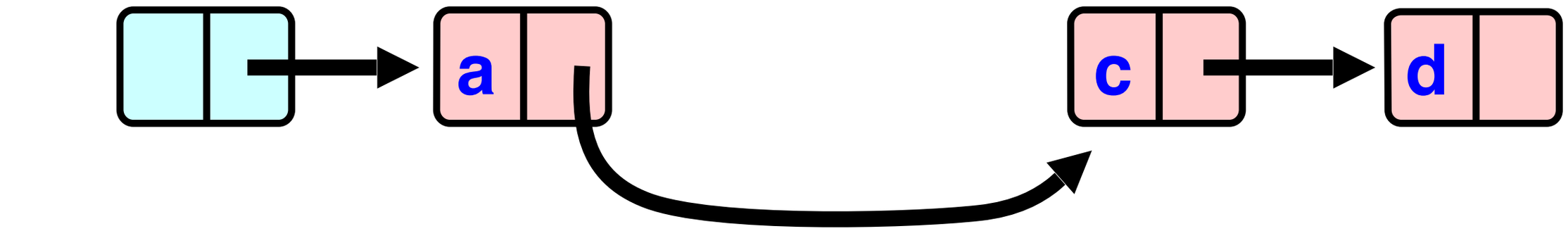
Hand-Over-Hand Again

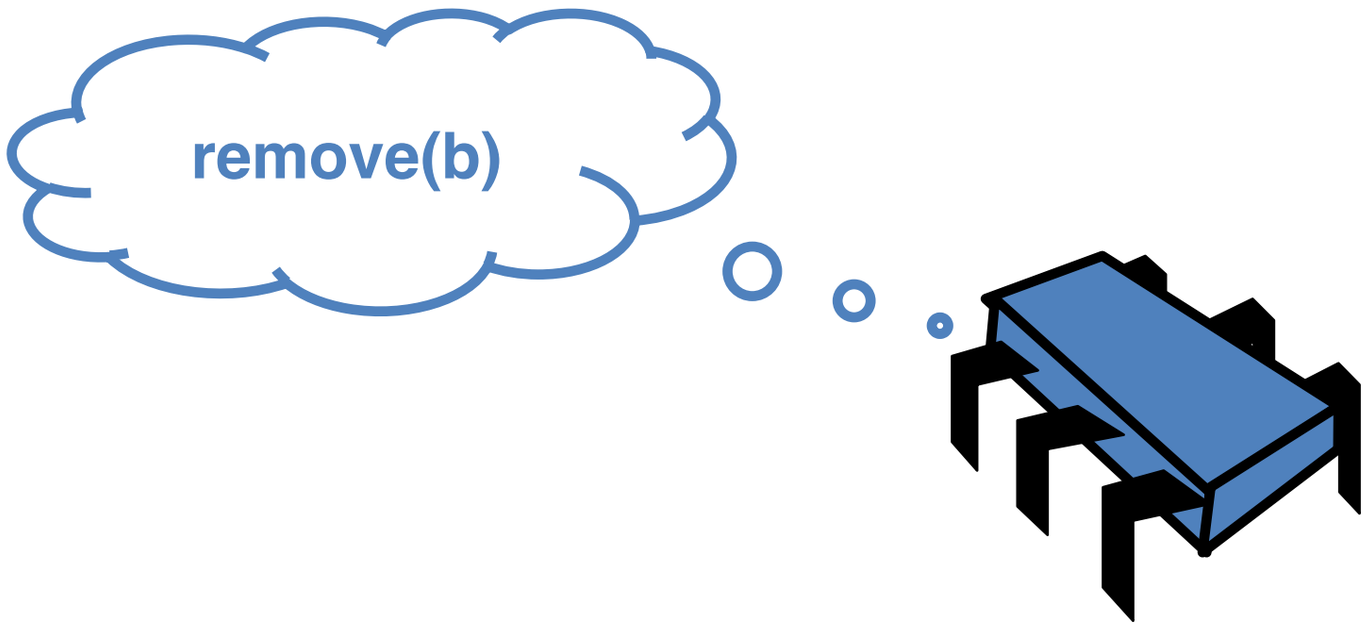


Hand-Over-Hand Again

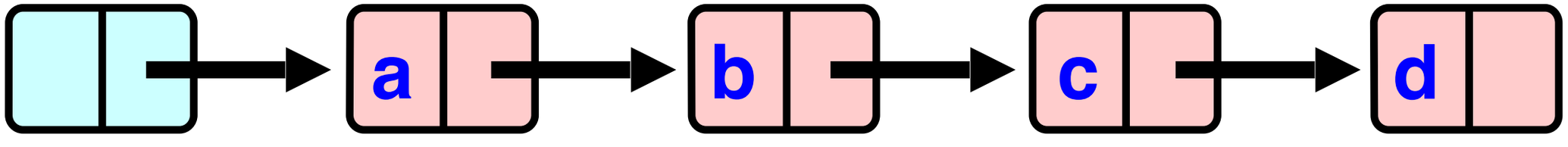


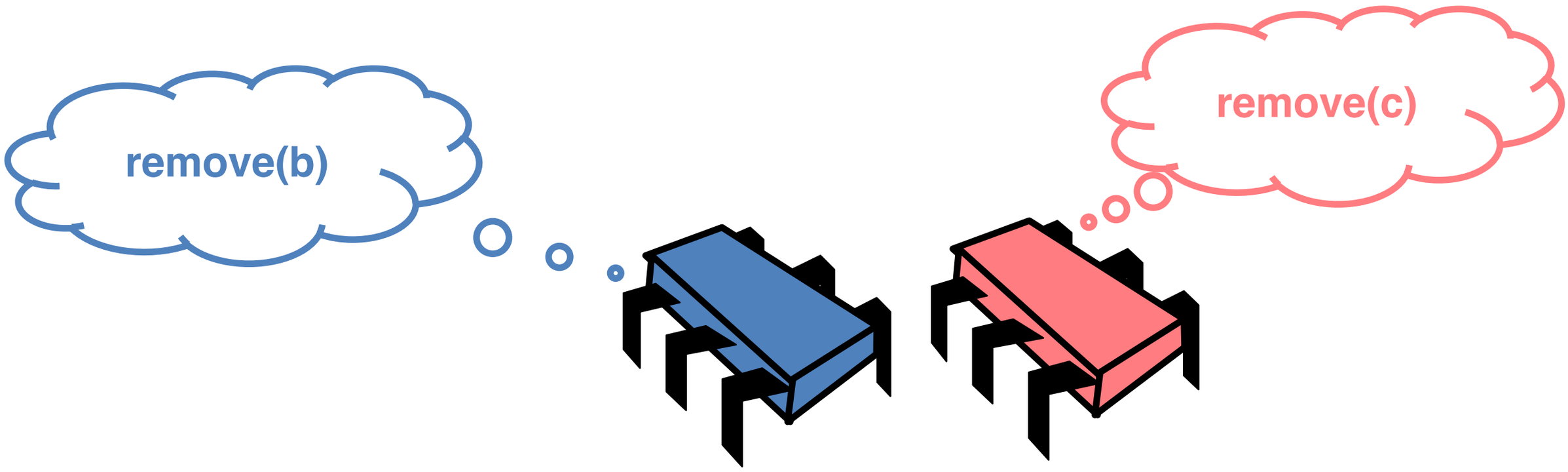
Hand-Over-Hand Again



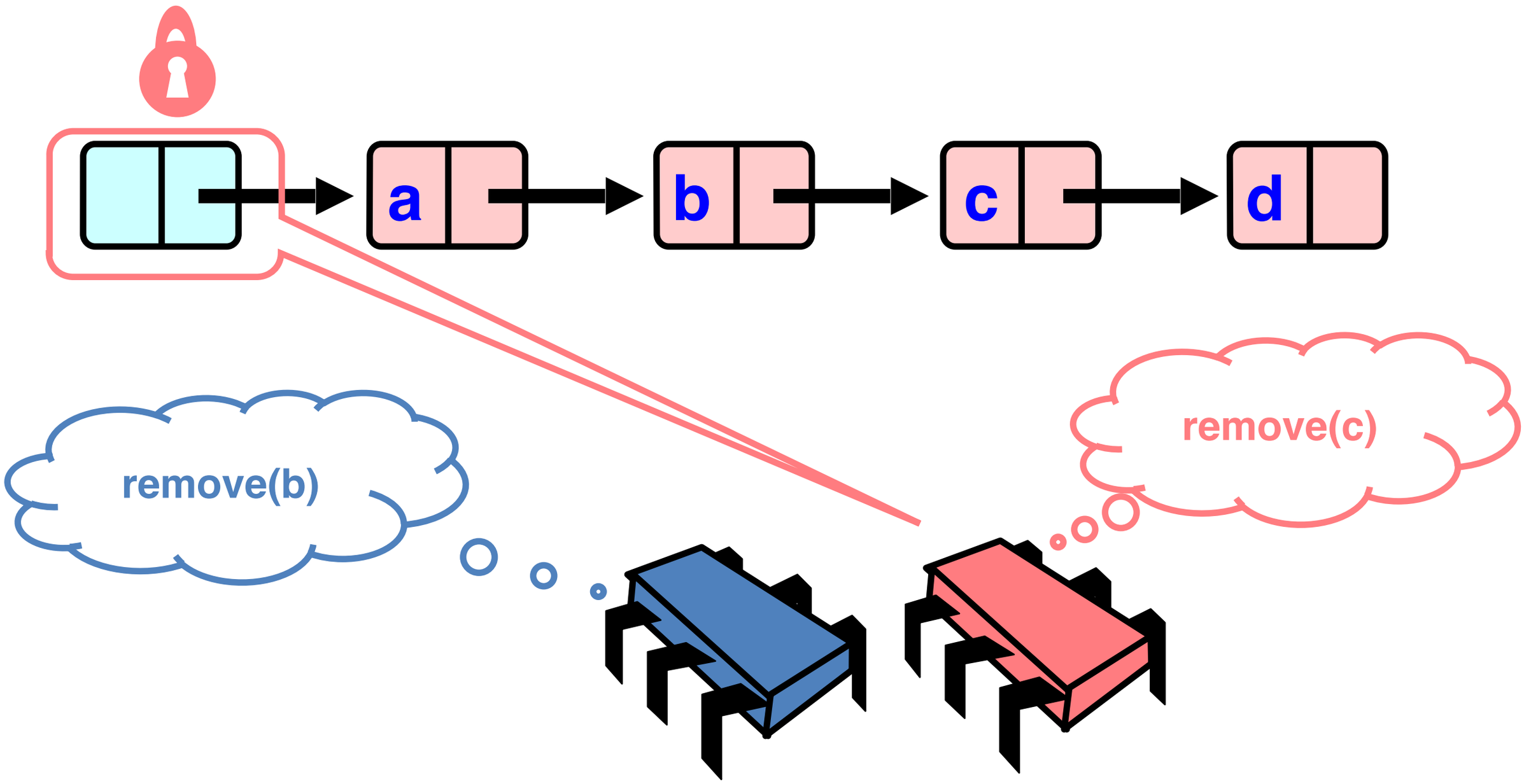


Removing a Node

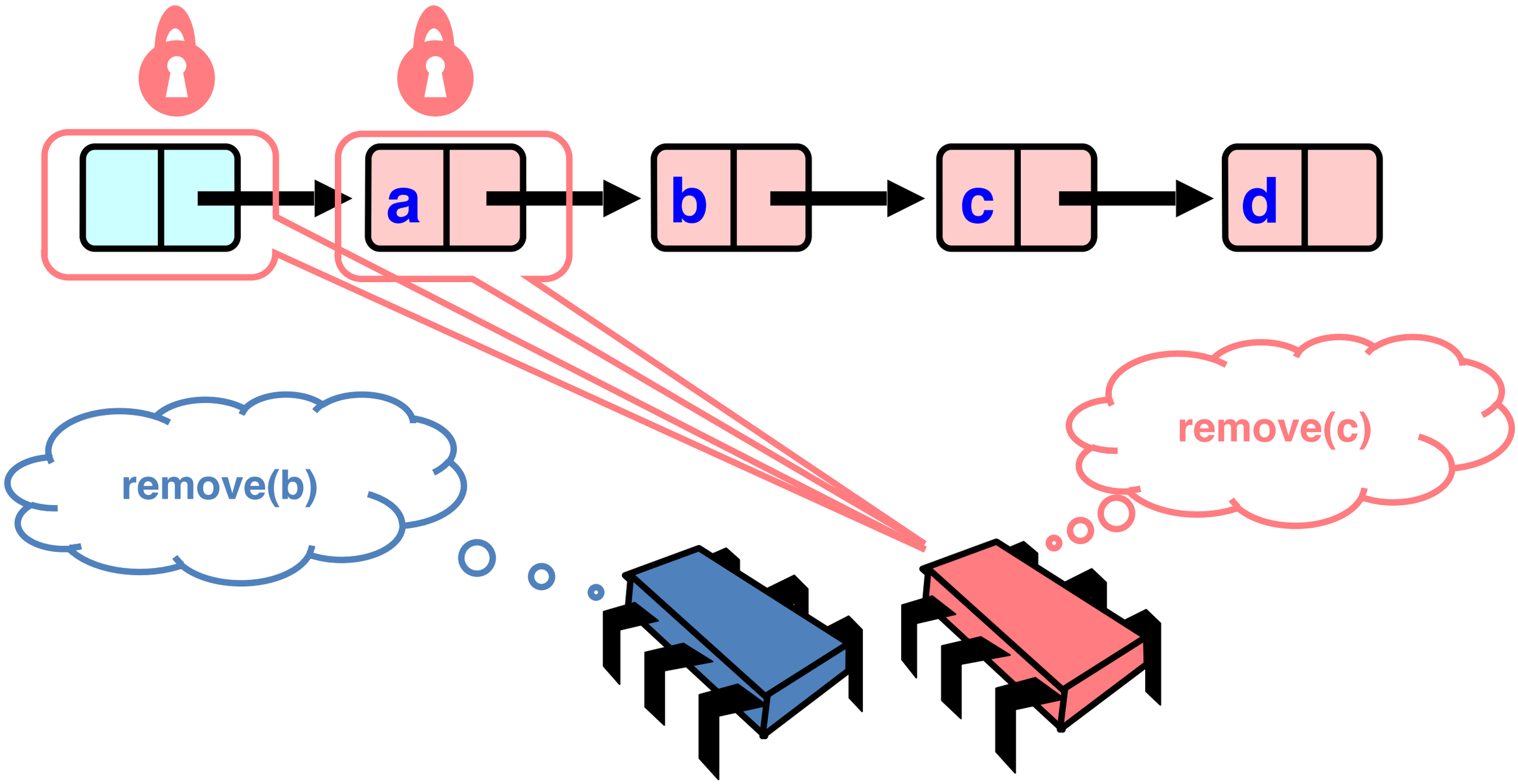




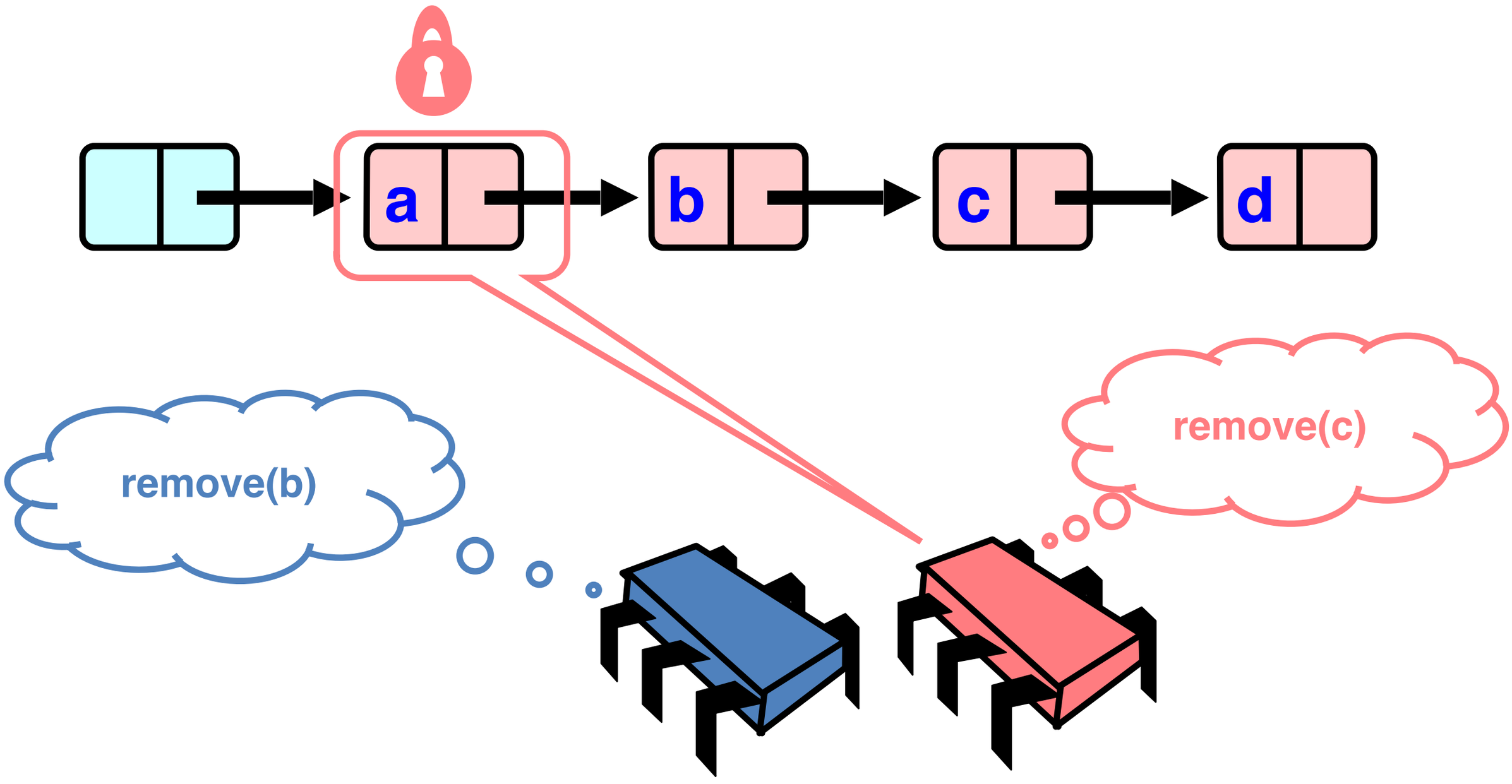
Removing a Node



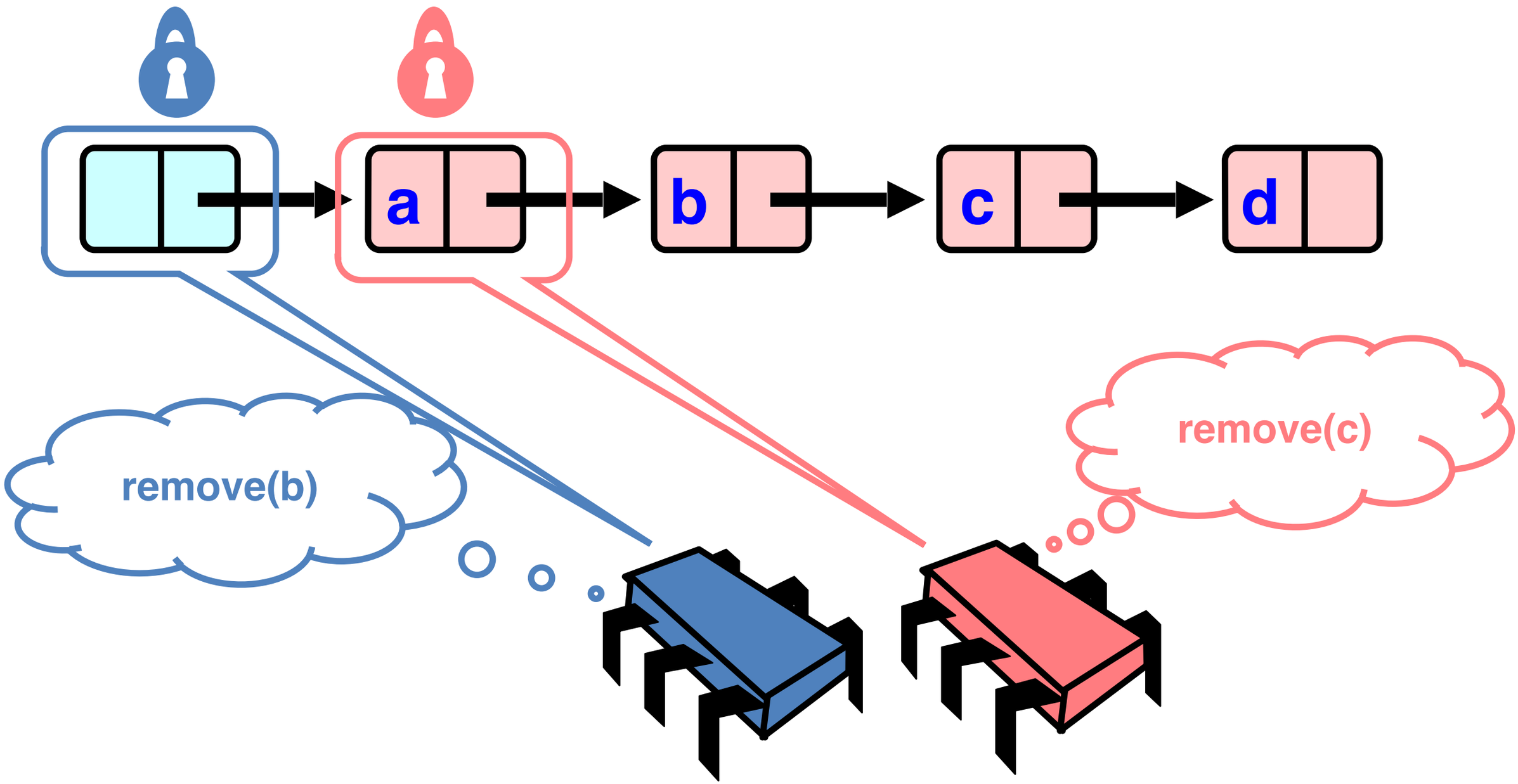
Removing a Node



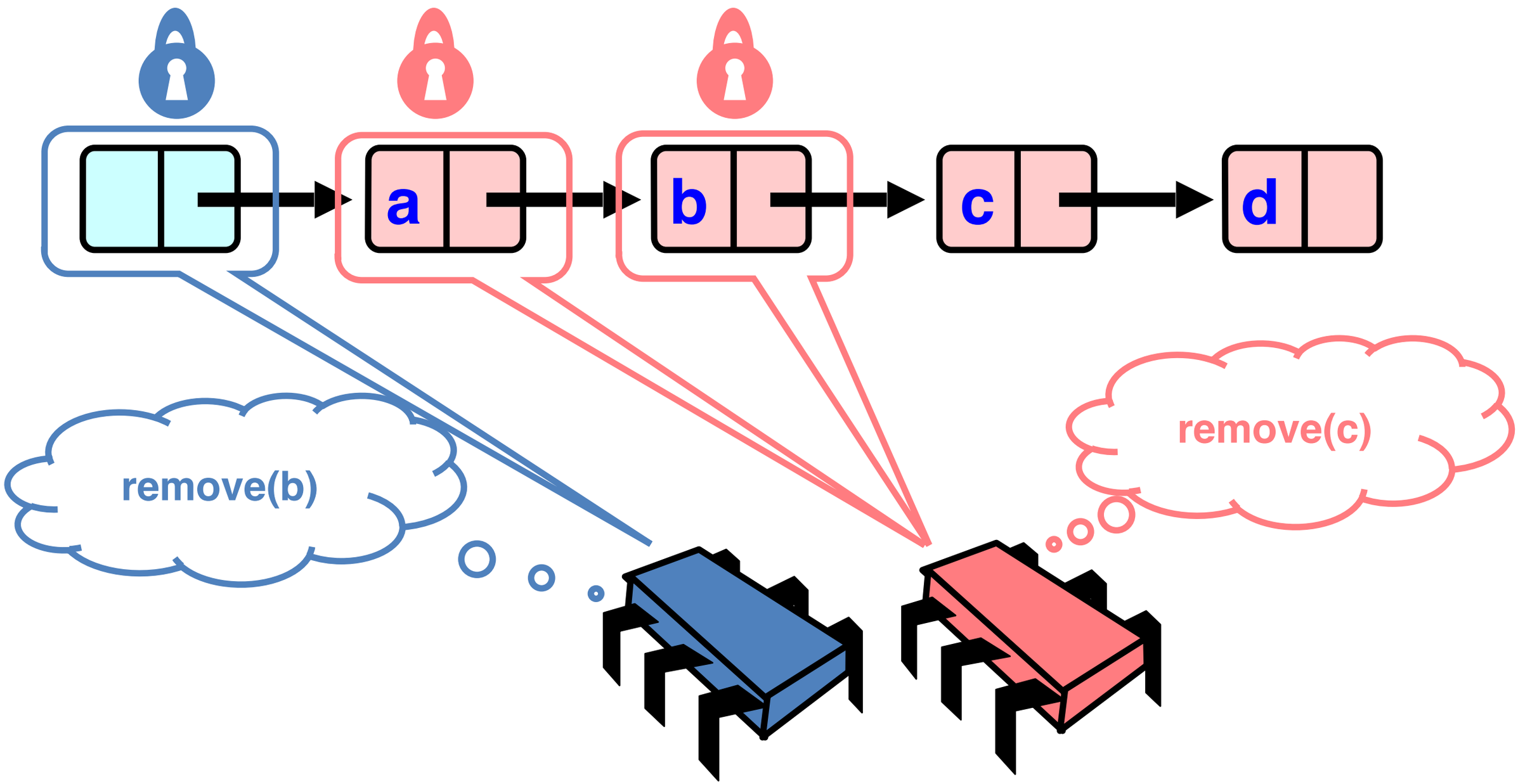
Removing a Node



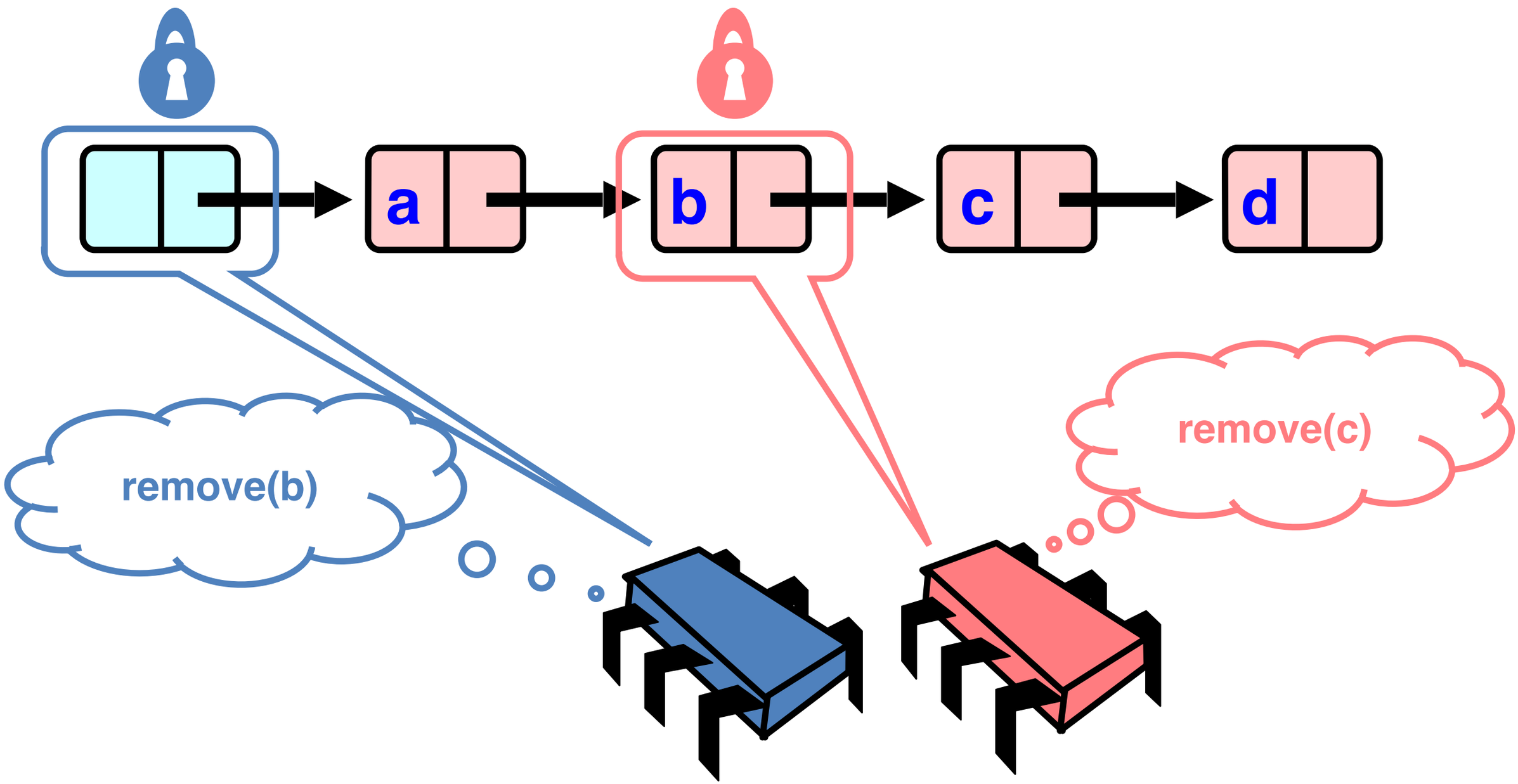
Removing a Node



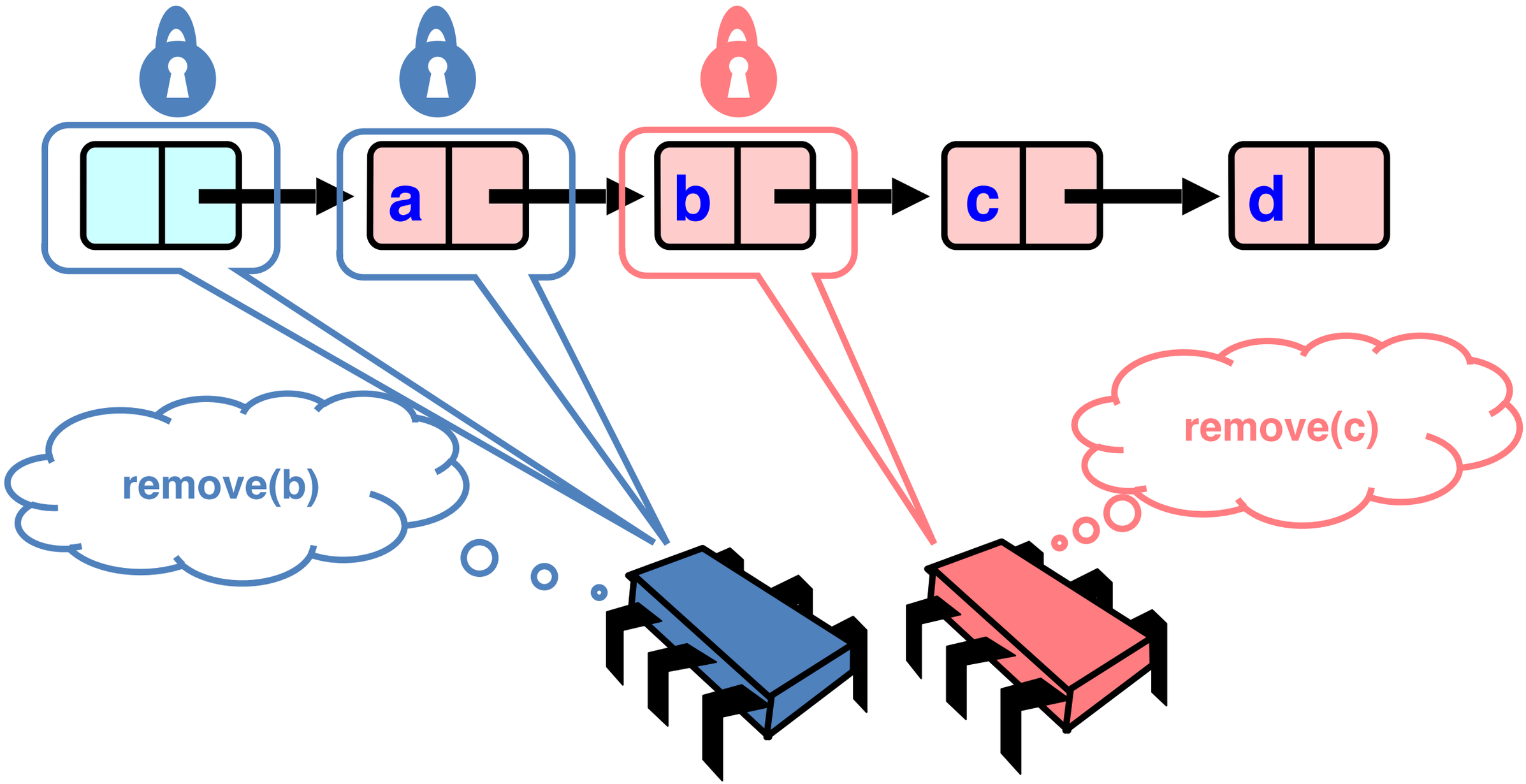
Removing a Node



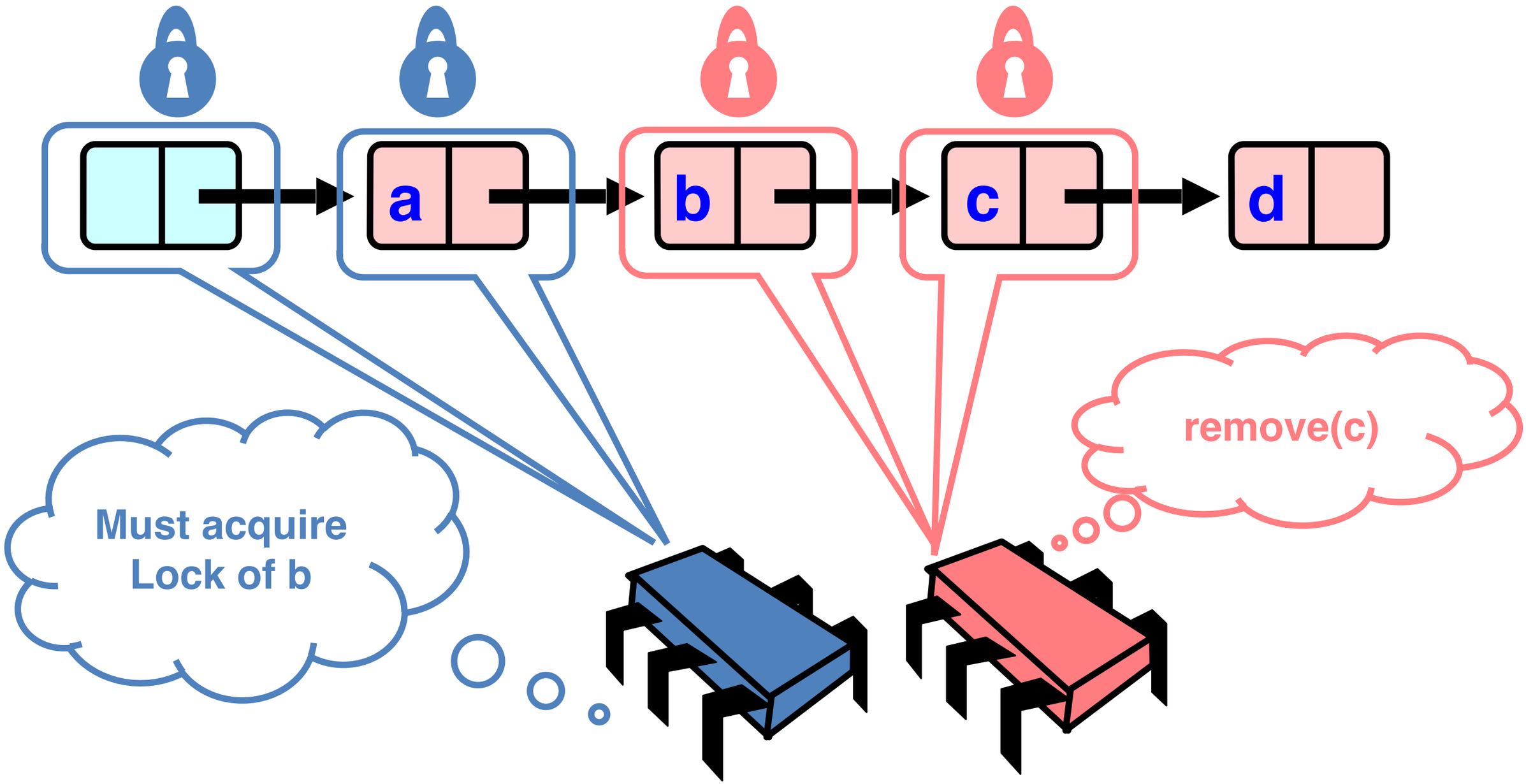
Removing a Node



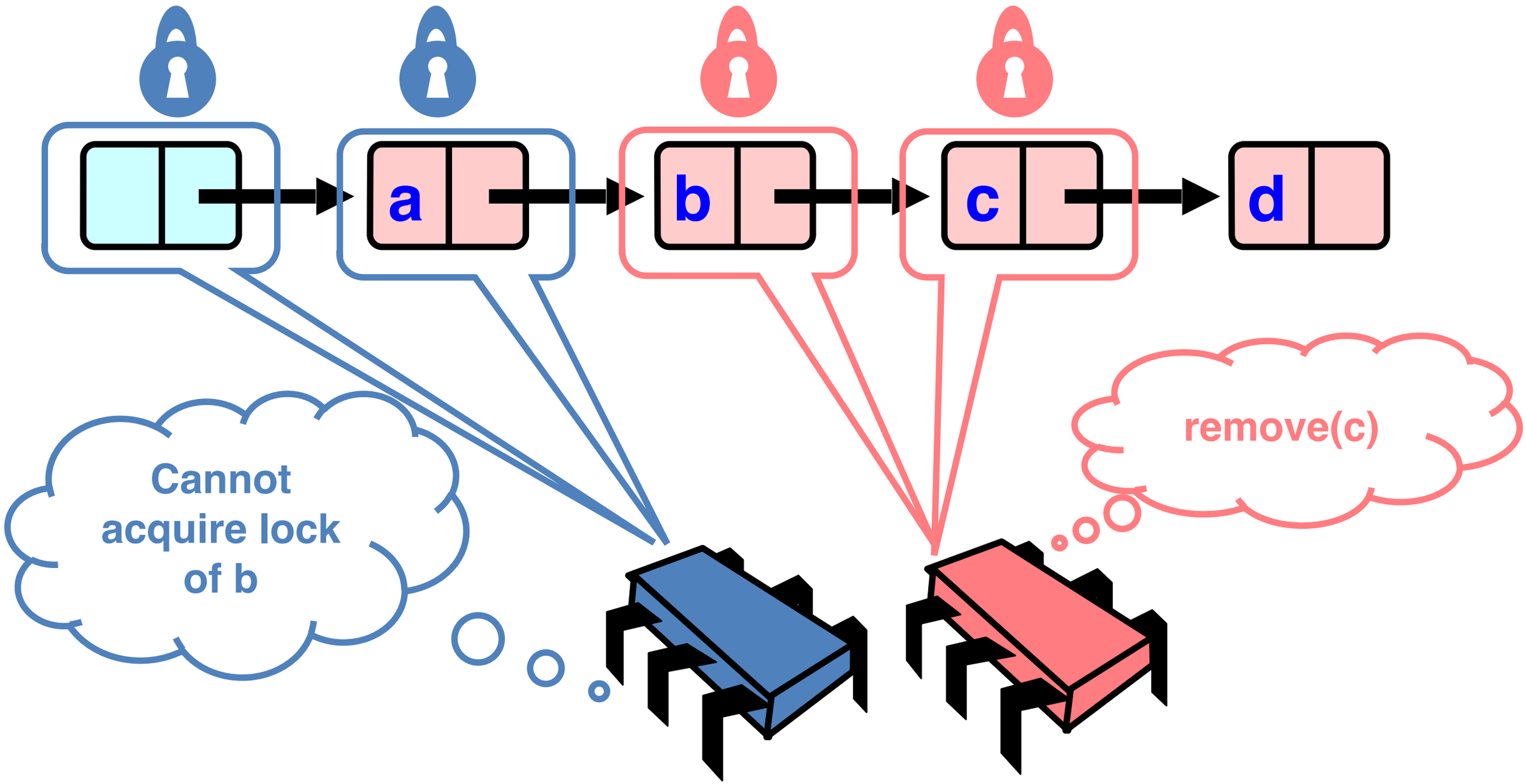
Removing a Node



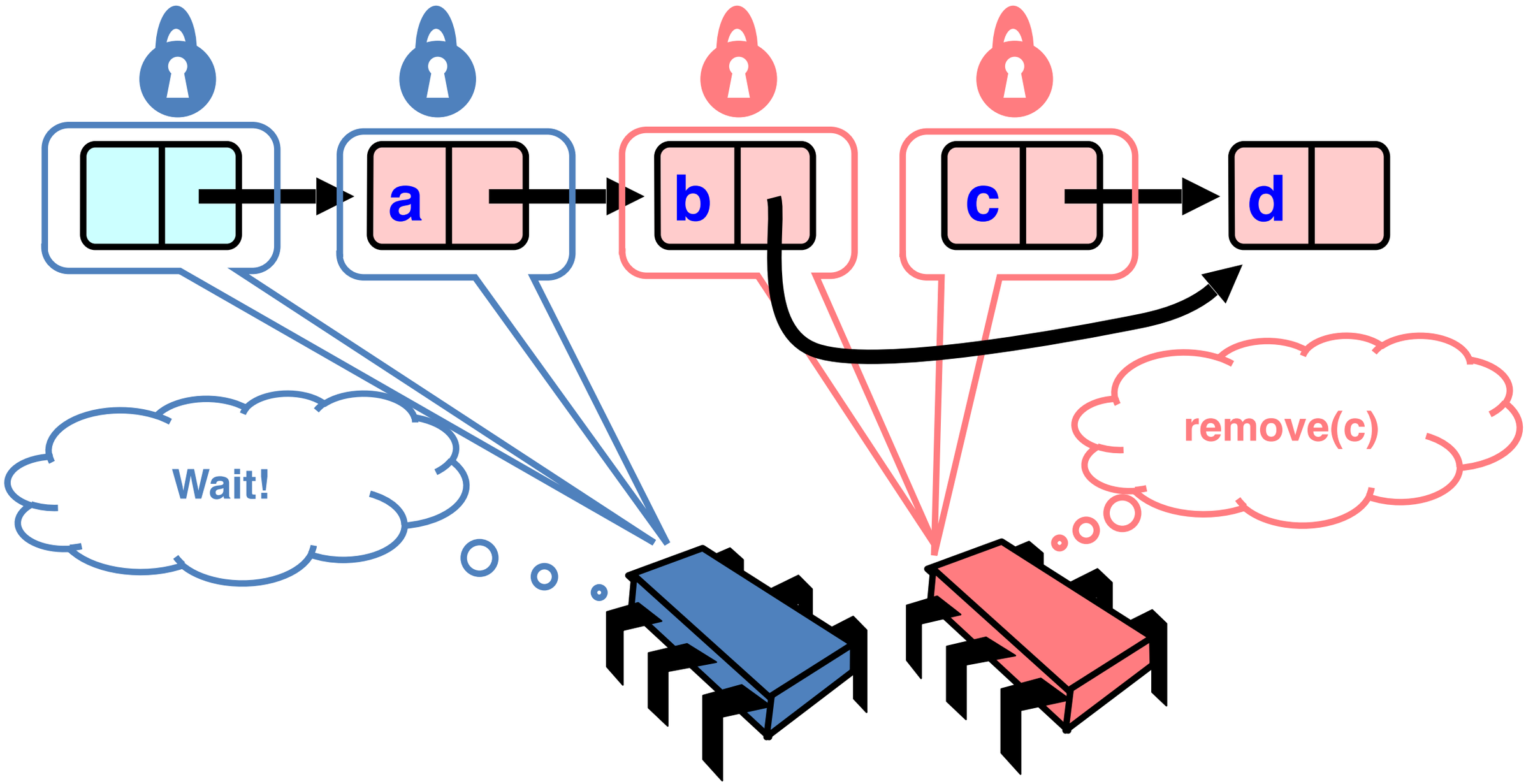
Removing a Node



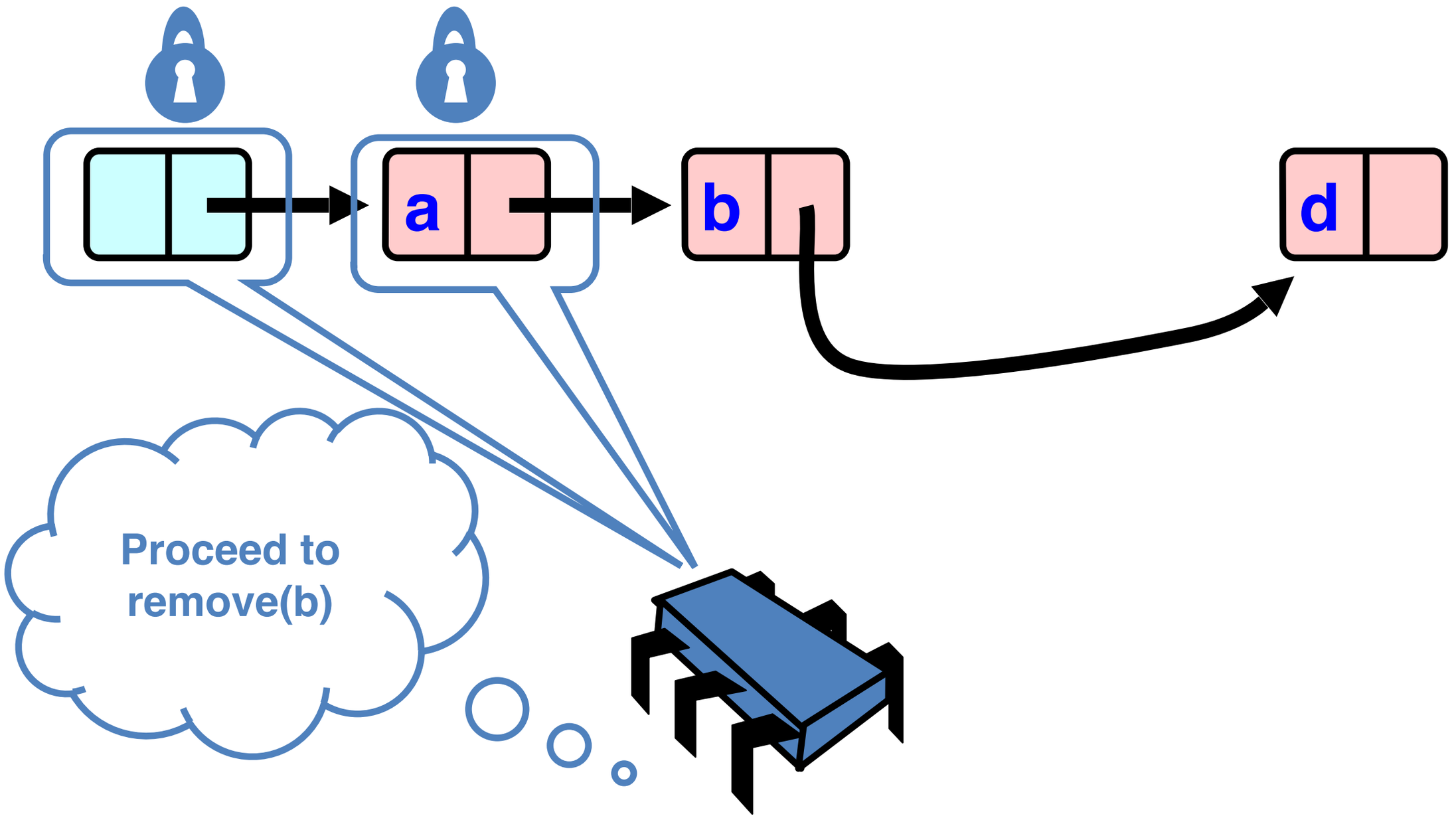
Removing a Node



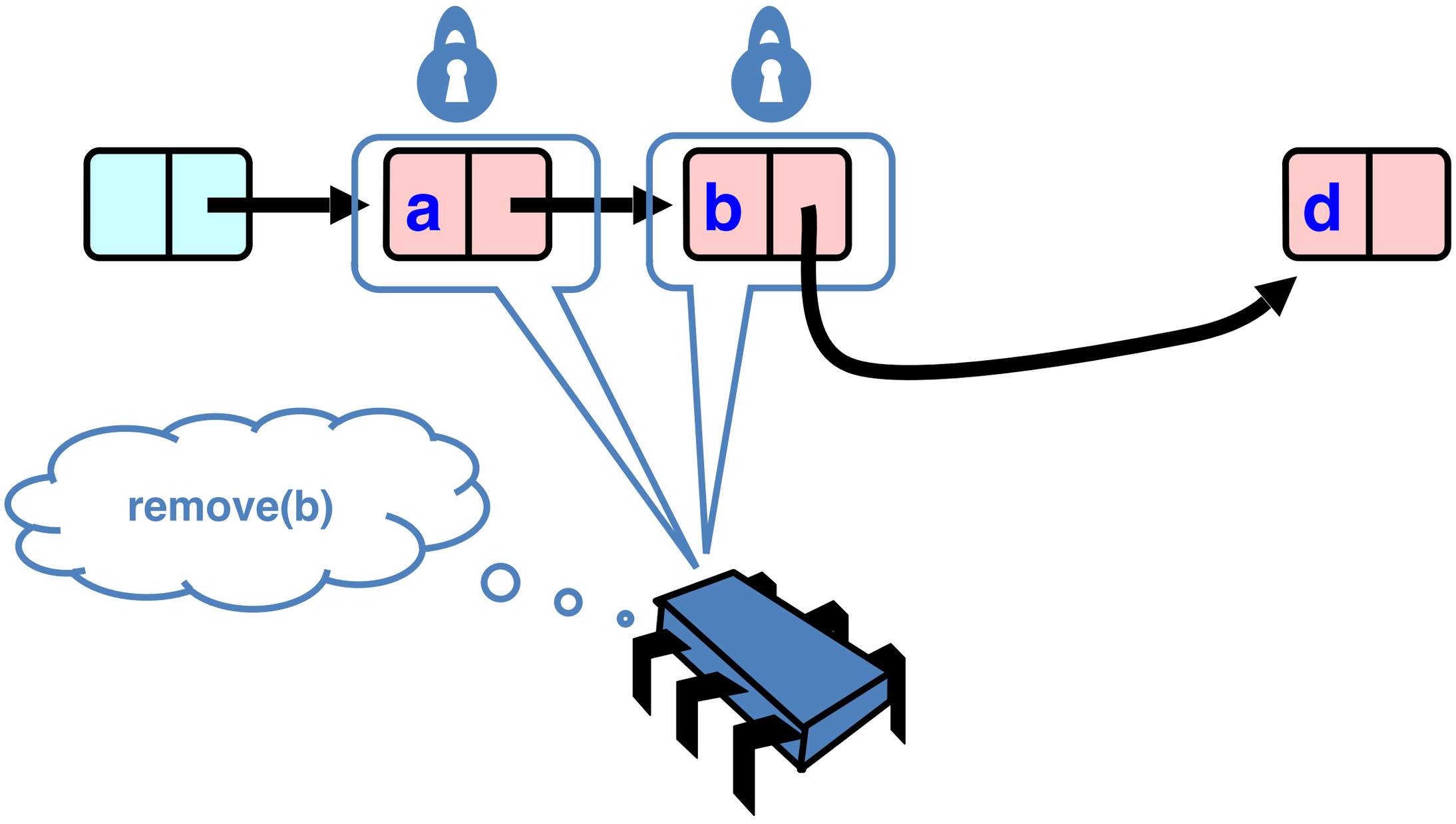
Removing a Node



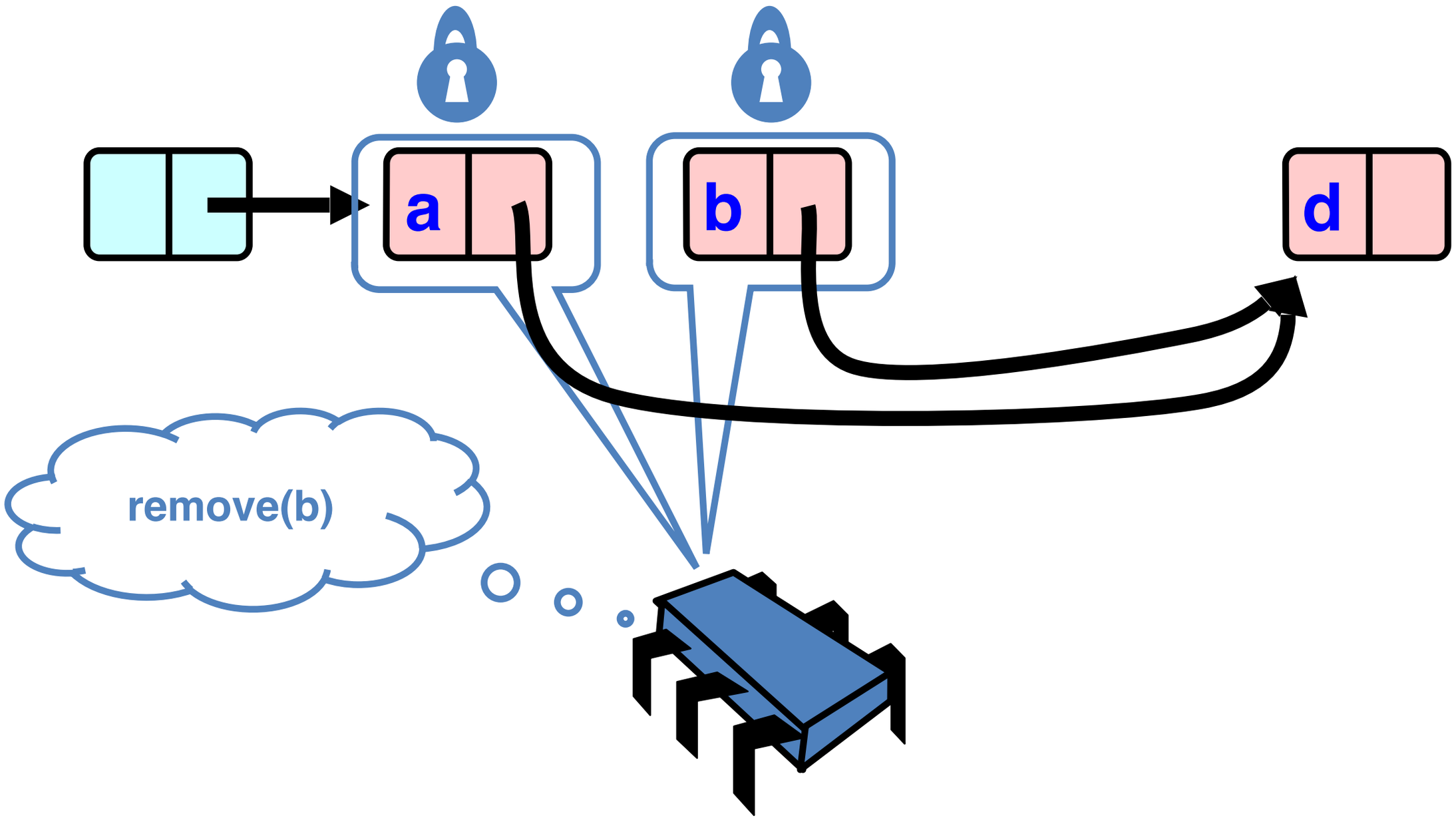
Removing a Node



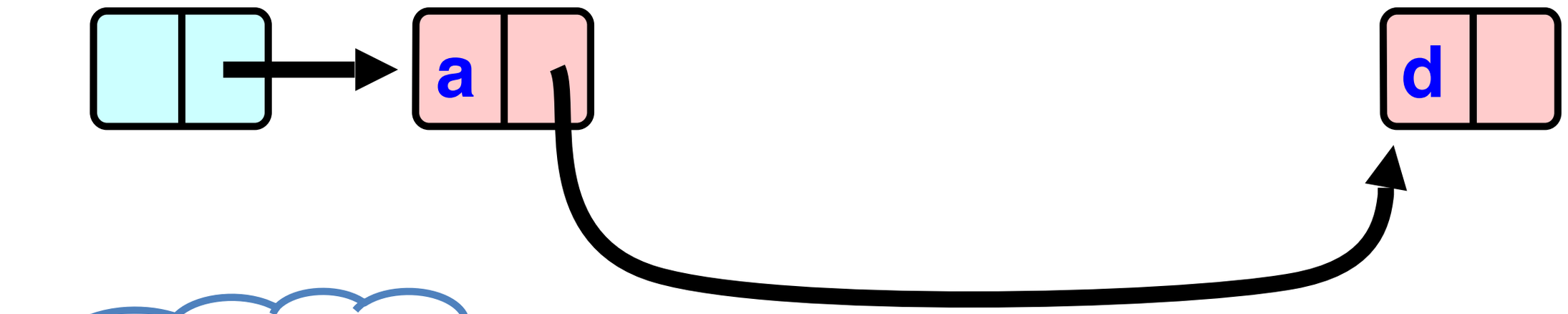
Removing a Node



Removing a Node



Removing a Node



Removing a Node