departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Alternative Synchronization Strategies
— Lock Free Algorithms (1) —

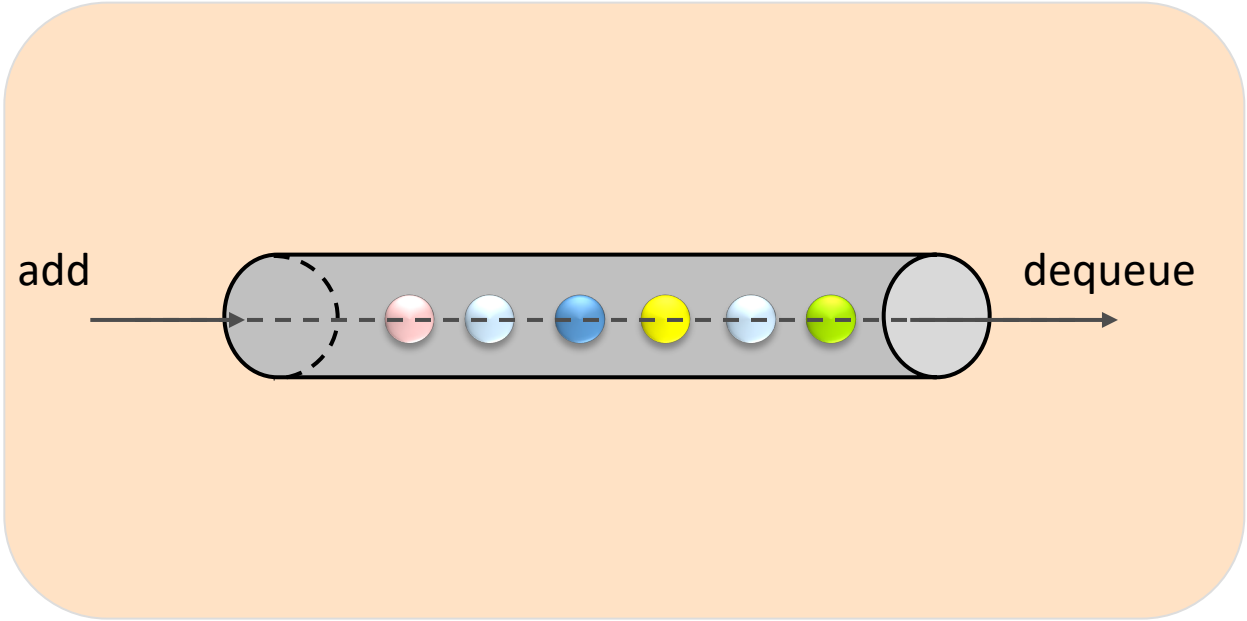
Concurrency and Parallelism — 2018-19

Master in Computer Science

(Mestrado Integrado em Eng. Informática)

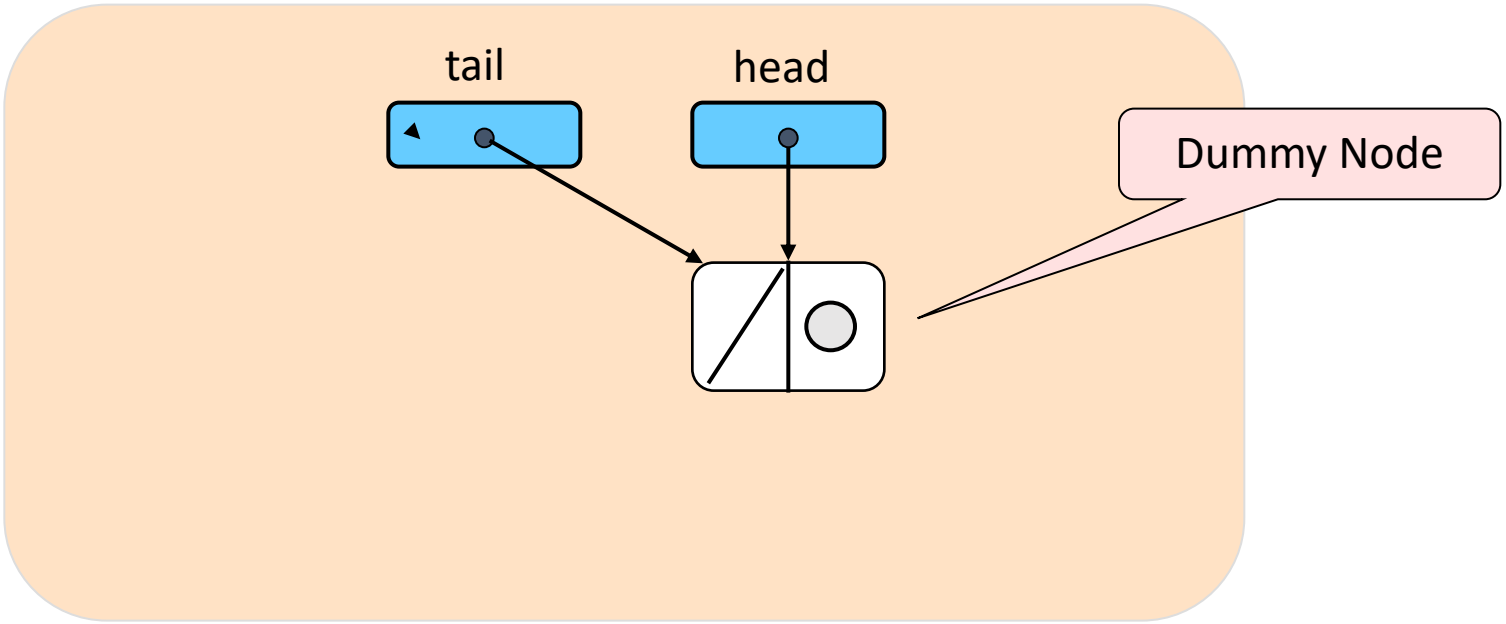
Joao Lourenço <joao.lourenco@fct.unl.pt>

Basics for a lock-free Queue

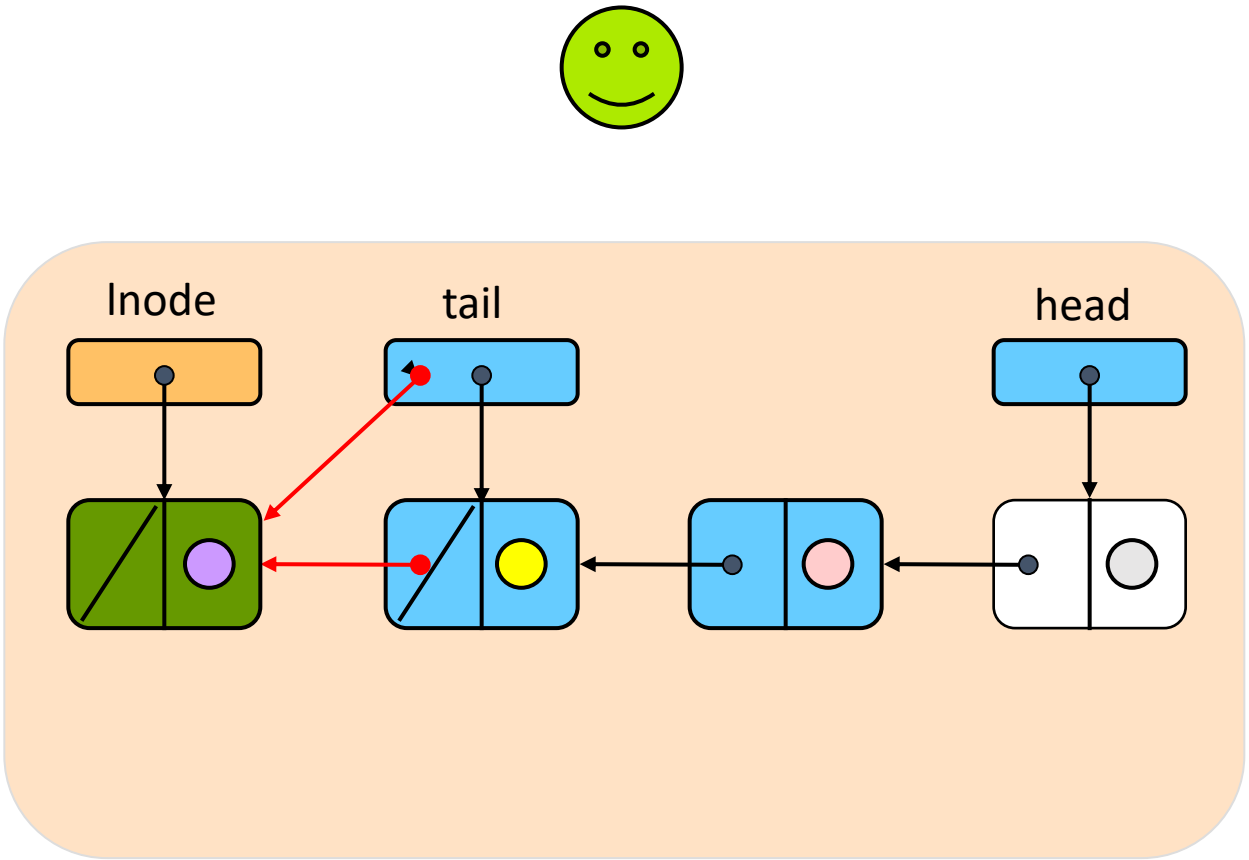


Basics for a lock-free Queue

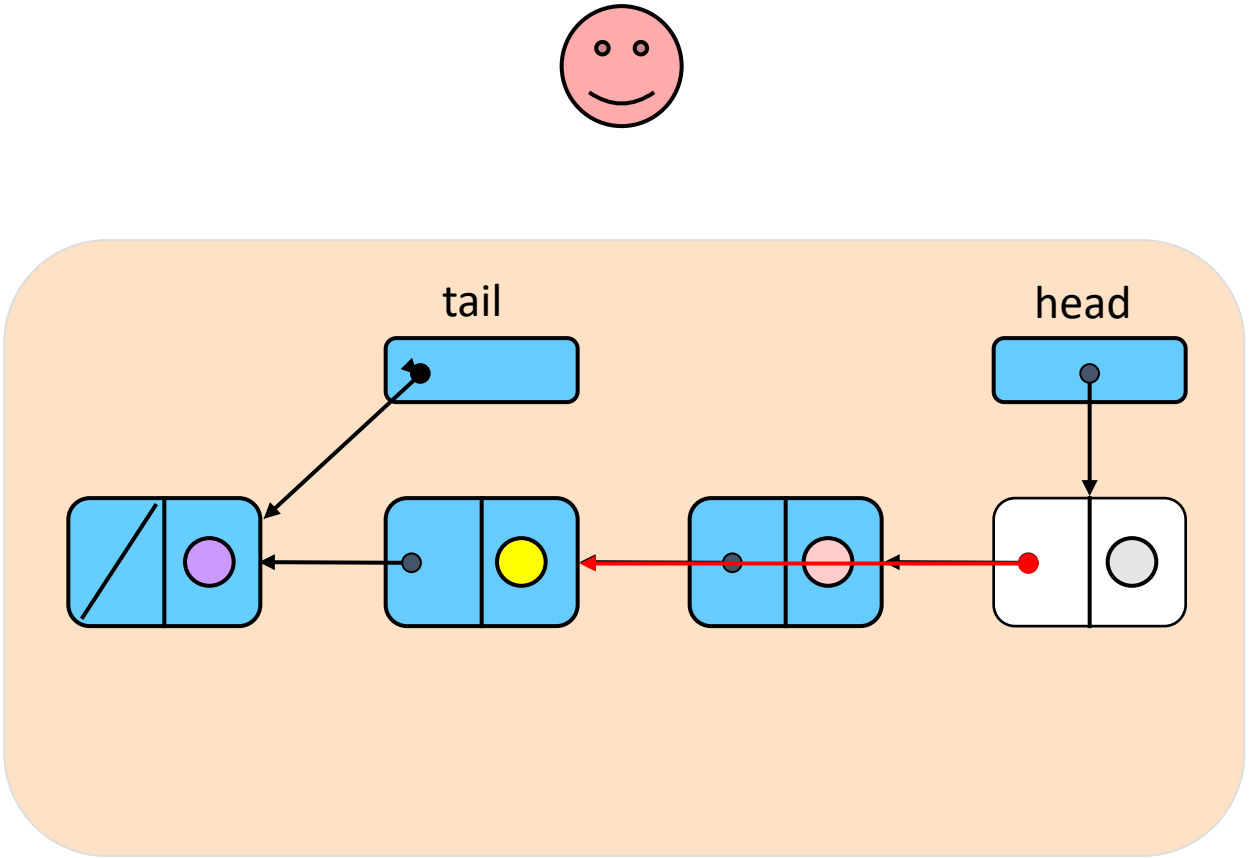
Empty queue



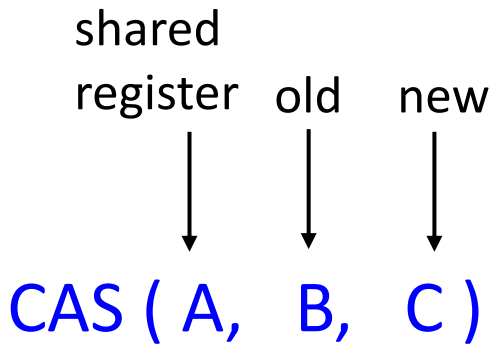
Enqueue



Dequeue



Compare & set(CAS)




```
if A=B then   A:=C;   return(true)
       else   return(false)
```

Supported by Sun, Intel, AMD, …

Reminder: Lock-Free Data Structures

- No matter what …

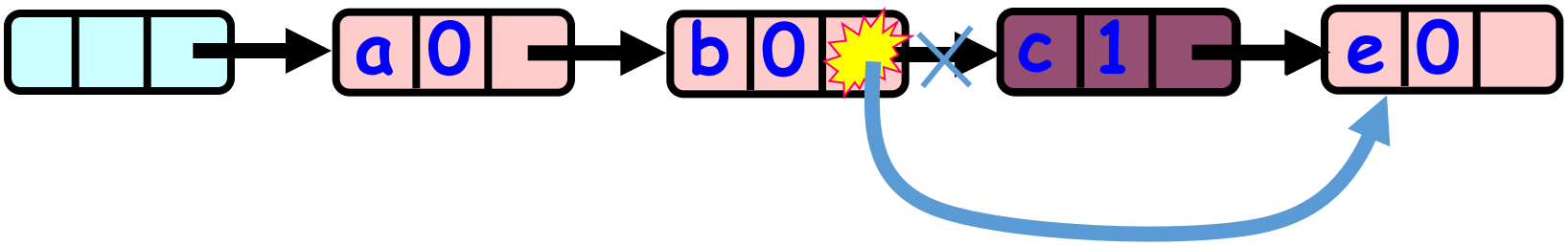
- Guarantees minimal progress in any execution
  - i.e., some thread will always complete a method call
- Even if others halt at malicious times
- Implies that implementation can't use locks

Lock-free Lists

- Next logical step is…
- Eliminate locking entirely
  - *contains()* wait-free
  - *add()* lock-free
  - *remove()* lock-free
- Use only *compareAndSet()*
- What could go wrong?

Remove Using CAS

Logical Removal =
Set Mark Bit

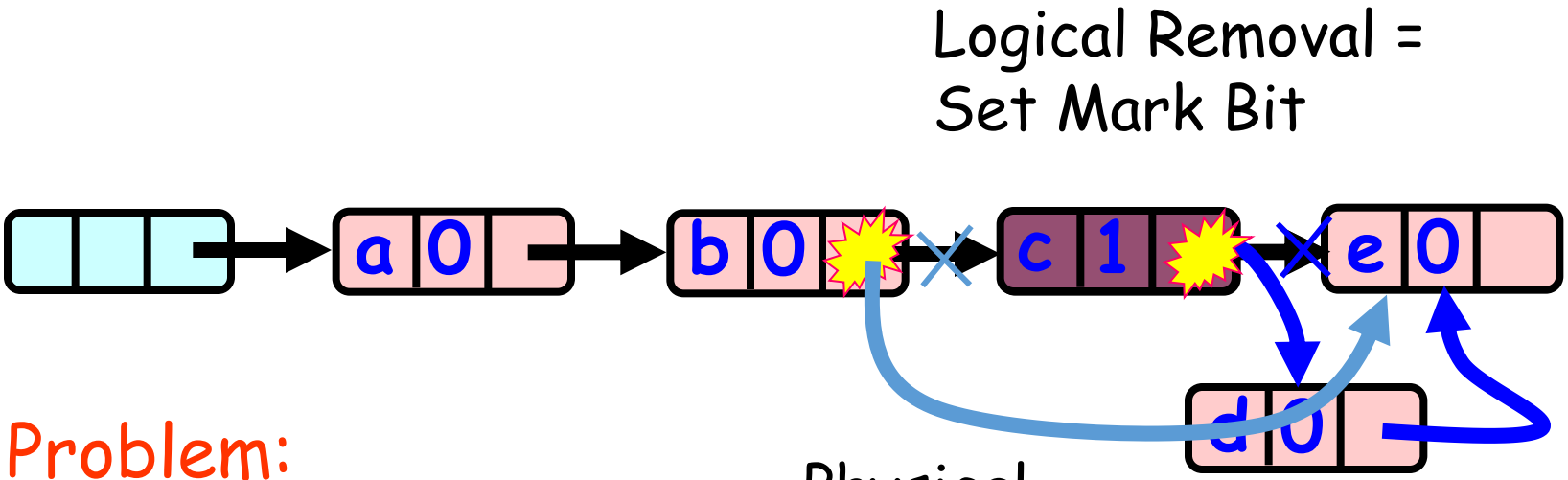


Use CAS to verify
if pointer is correct

Not enough!

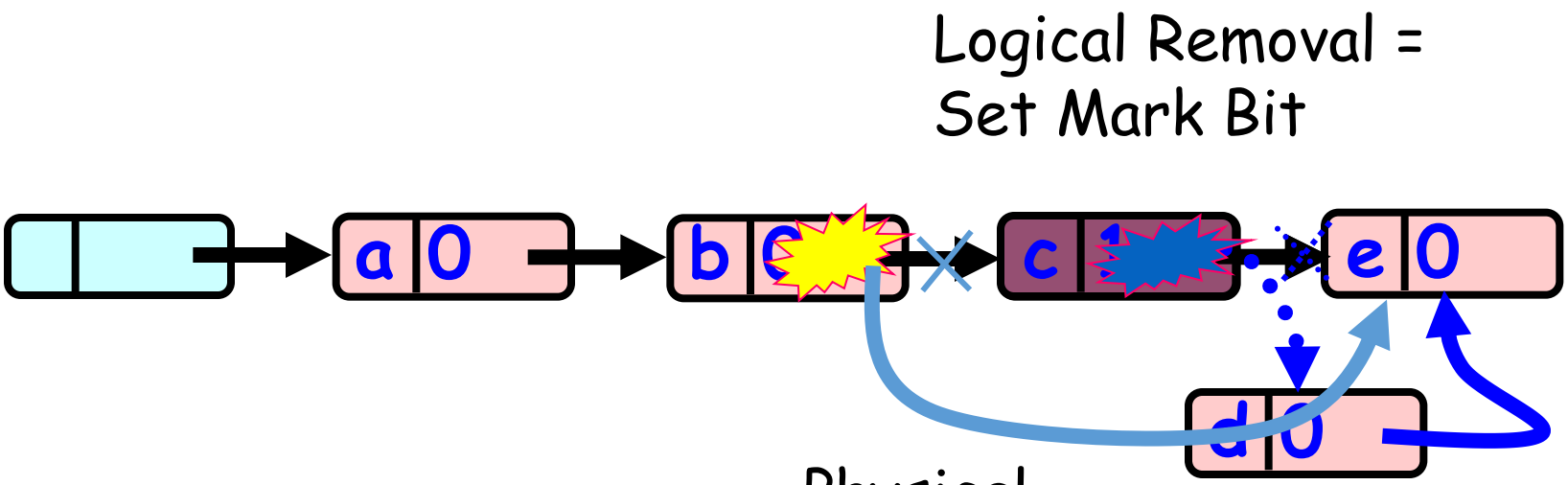
Physical
Removal
CAS pointer

Problem…



Problem:
'd' not added to list…
Must Prevent
manipulation of
removed node's pointer

The Solution: Combine Bit and Pointer



Mark-Bit and Pointer are CASed together (AtomicMarkableReference)

Solution

- Use *AtomicMarkableReference*
- Atomically
  - Swing reference and
  - Update flag
- Remove in two steps
  - Set mark bit in next field
  - Redirect predecessor's pointer

Marking a Node

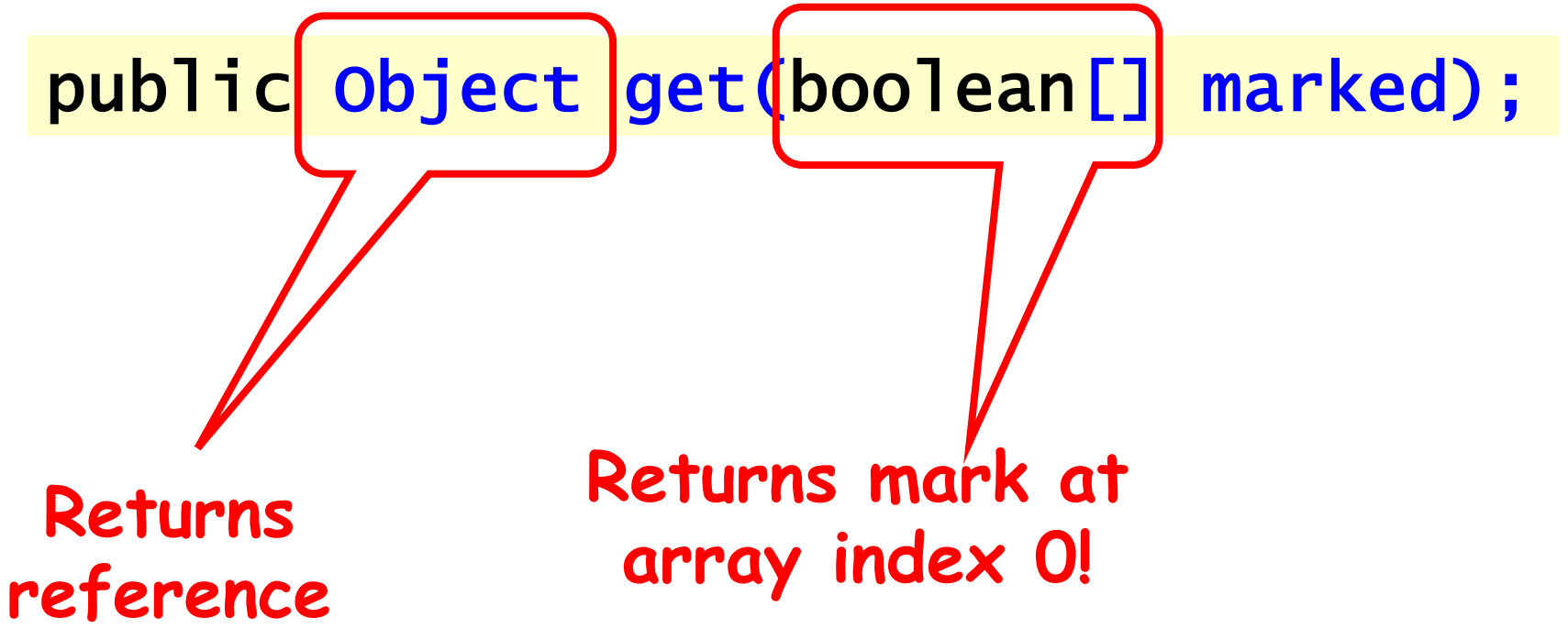
- AtomicMarkableReference class
  - Java.util.concurrent.atomic package

Extracting Reference & Mark

```
public Object get(boolean[] marked);
```

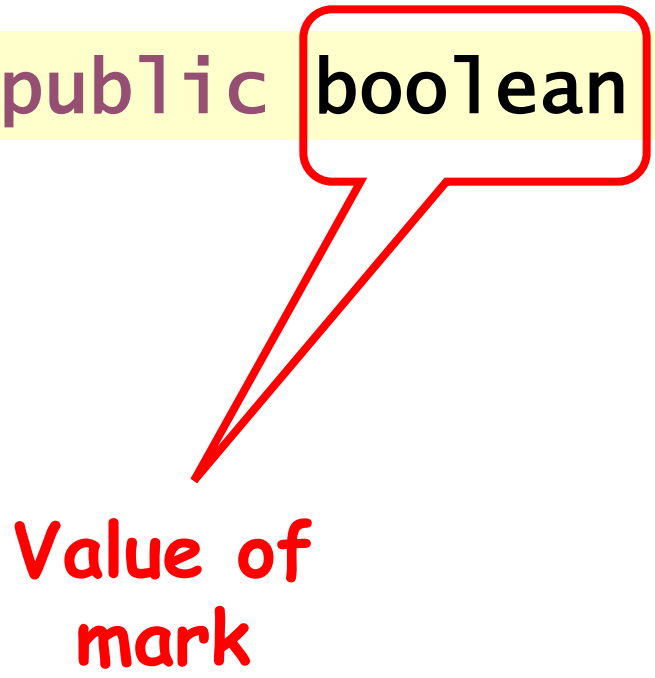
Extracting Reference & Mark

```
public Object get(boolean[] marked);
```



Extracting Reference Only

```
public boolean isMarked();
```



Changing State

```
public boolean compareAndSet(
  Object expectedRef,
  Object updateRef,
  boolean expectedMark,
  boolean updateMark);
```

Changing State

If this is the current reference …

```
public boolean compareAndSet(
  Object expectedRef,
  Object updateRef,
  boolean expectedMark,
  boolean updateMark);
```

And this is the current mark …

Changing State

…then change to this new reference …

```
public boolean compareAndSet(
  Object expectedRef,
  Object updateRef,
  boolean expectedMark,
  boolean updateMark);
```

… and this new mark

Changing State

```
public boolean attemptMark(
  Object expectedRef,
  boolean updateMark);
```

Changing State

```
public boolean attemptMark(
  Object expectedRef,
   boolean updateMark);
```

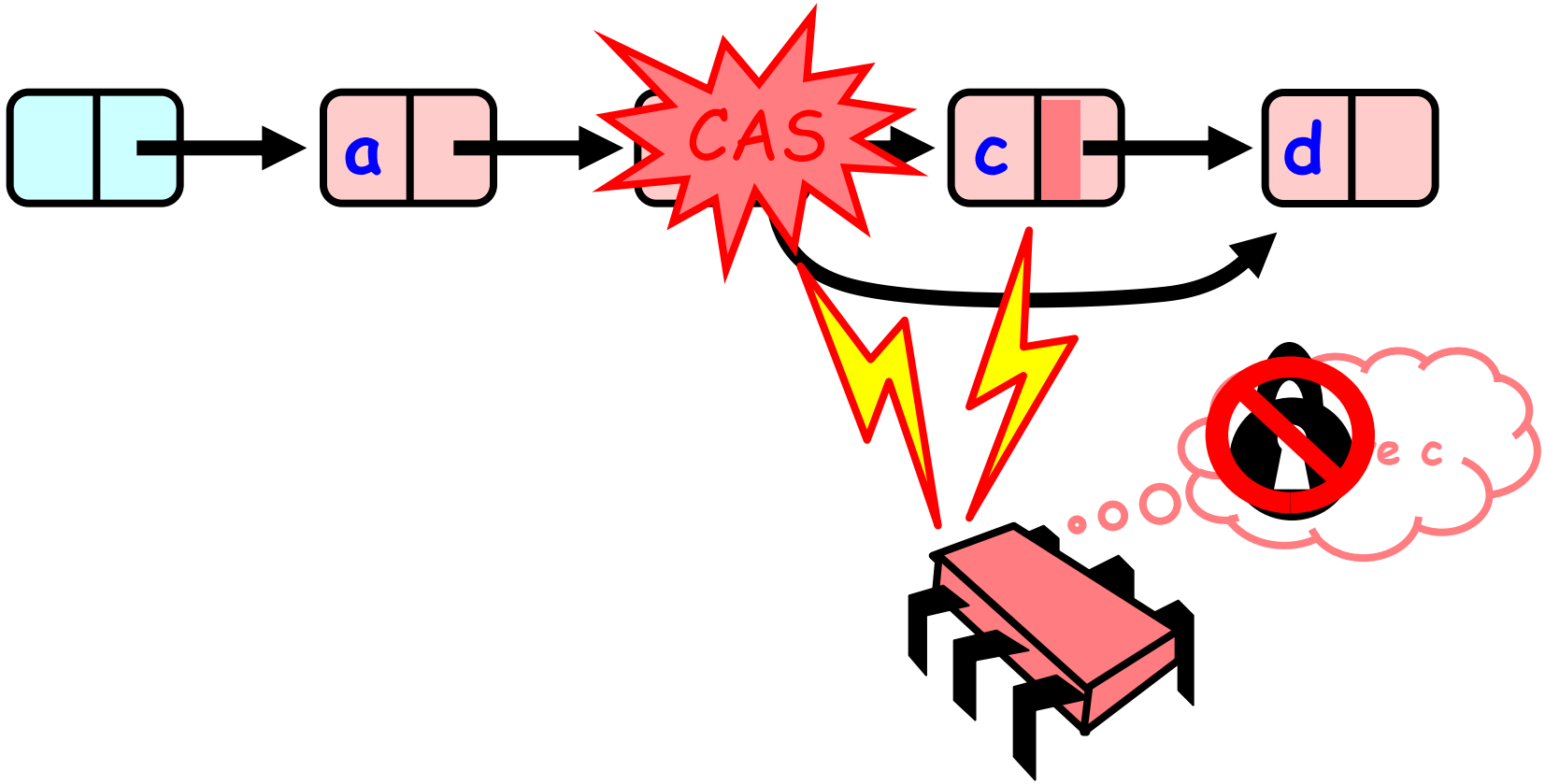

If this is the current reference …

Changing State

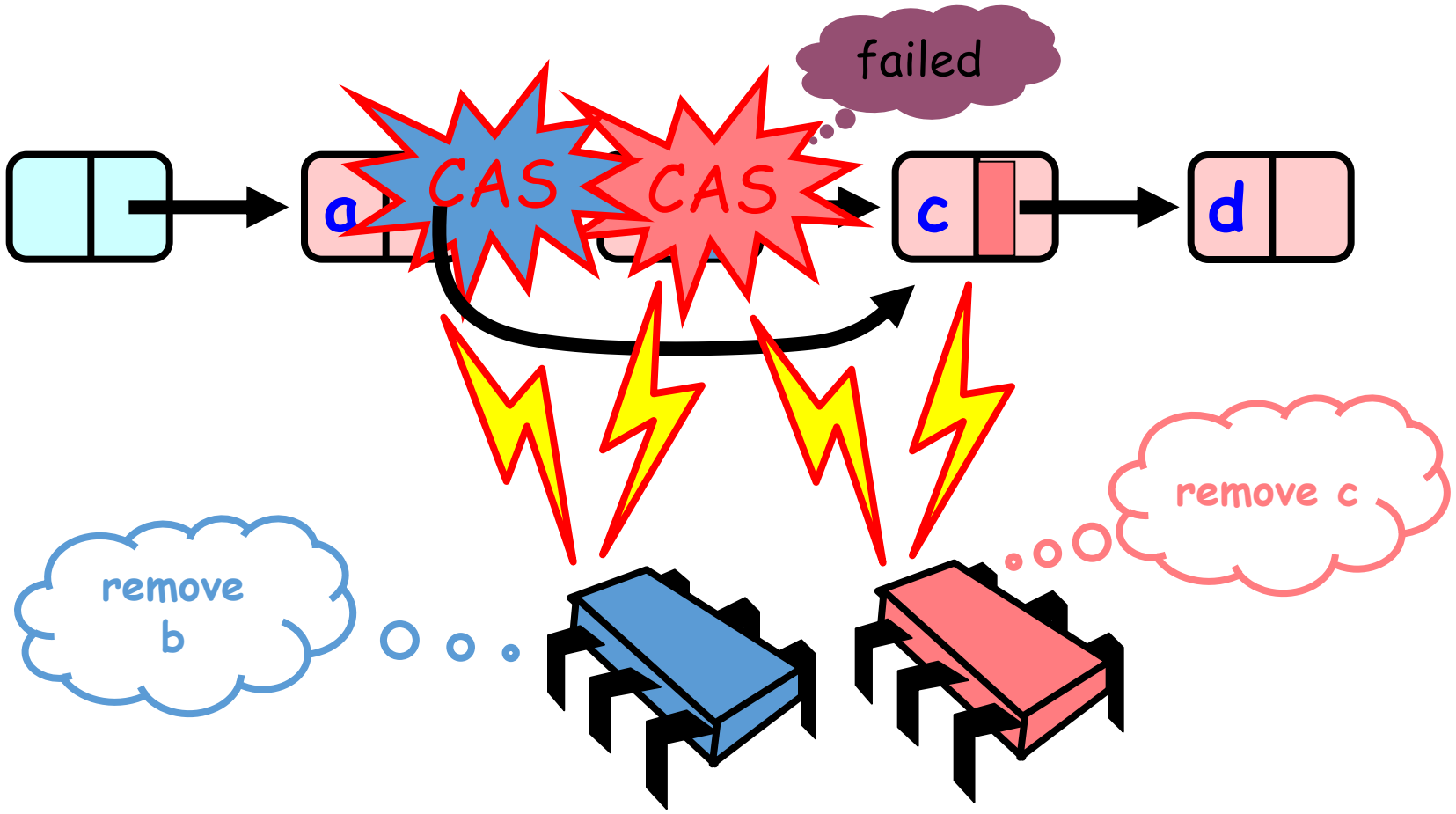
```
public boolean attemptMark(
   Object expectedRef,
   boolean updateMark);
```

.. then change to this new mark.

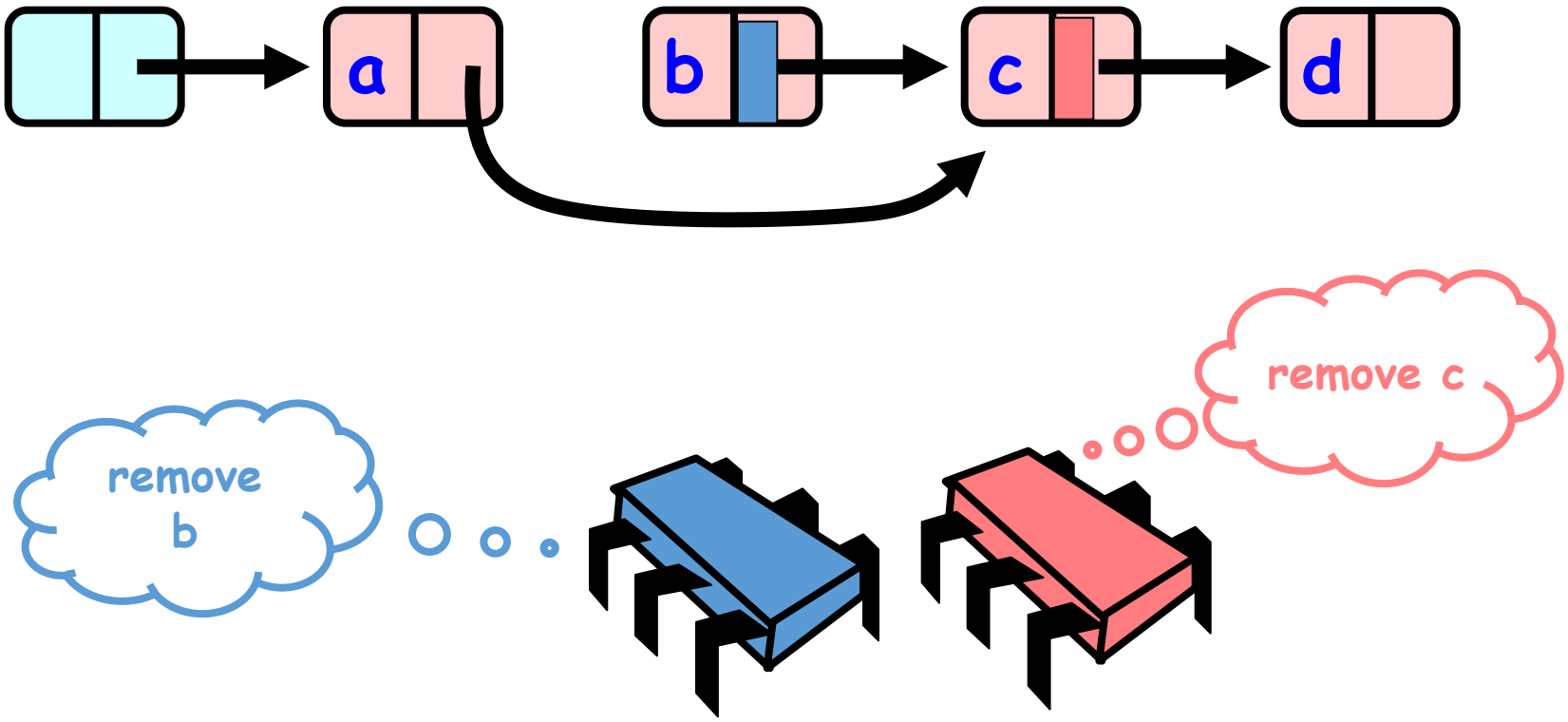
Removing a Node



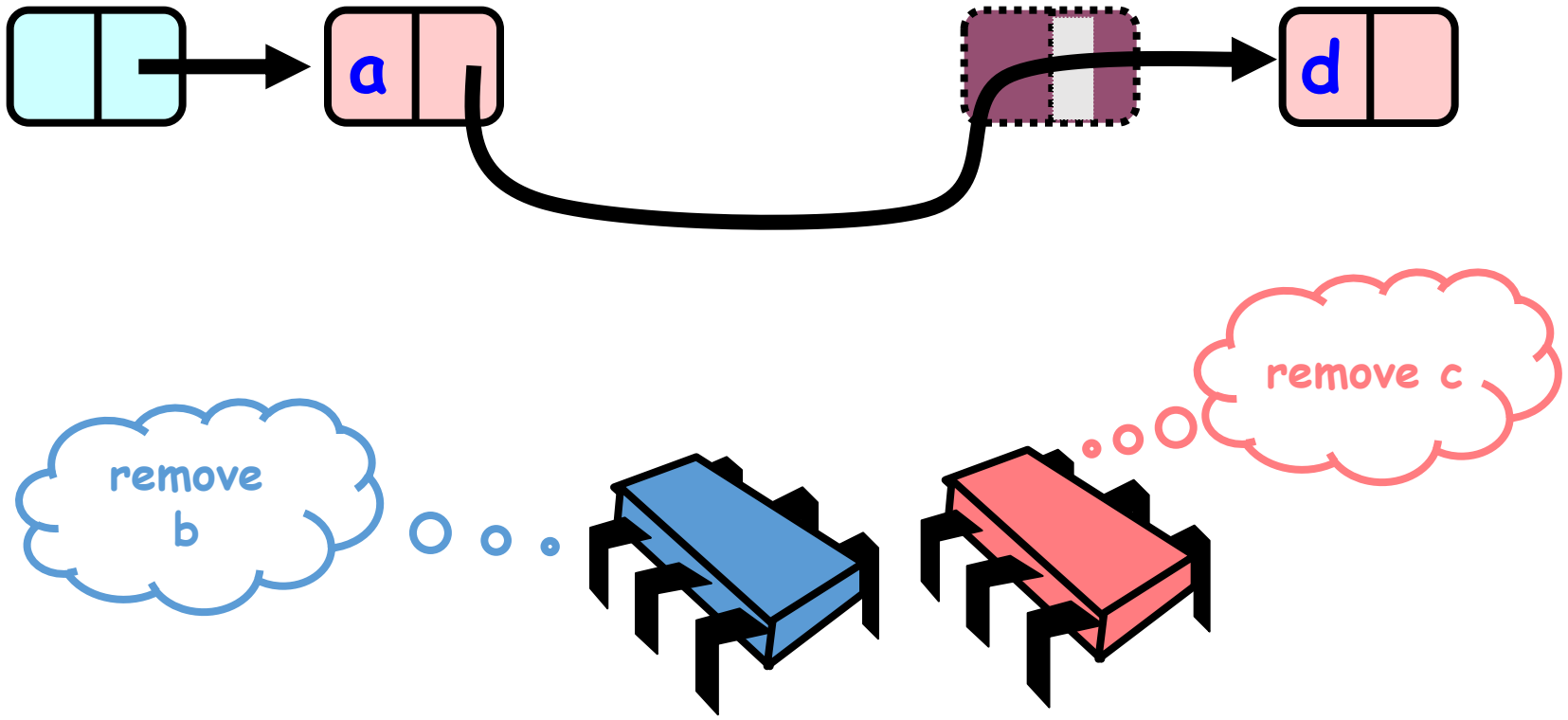
Removing a Node



Removing a Node



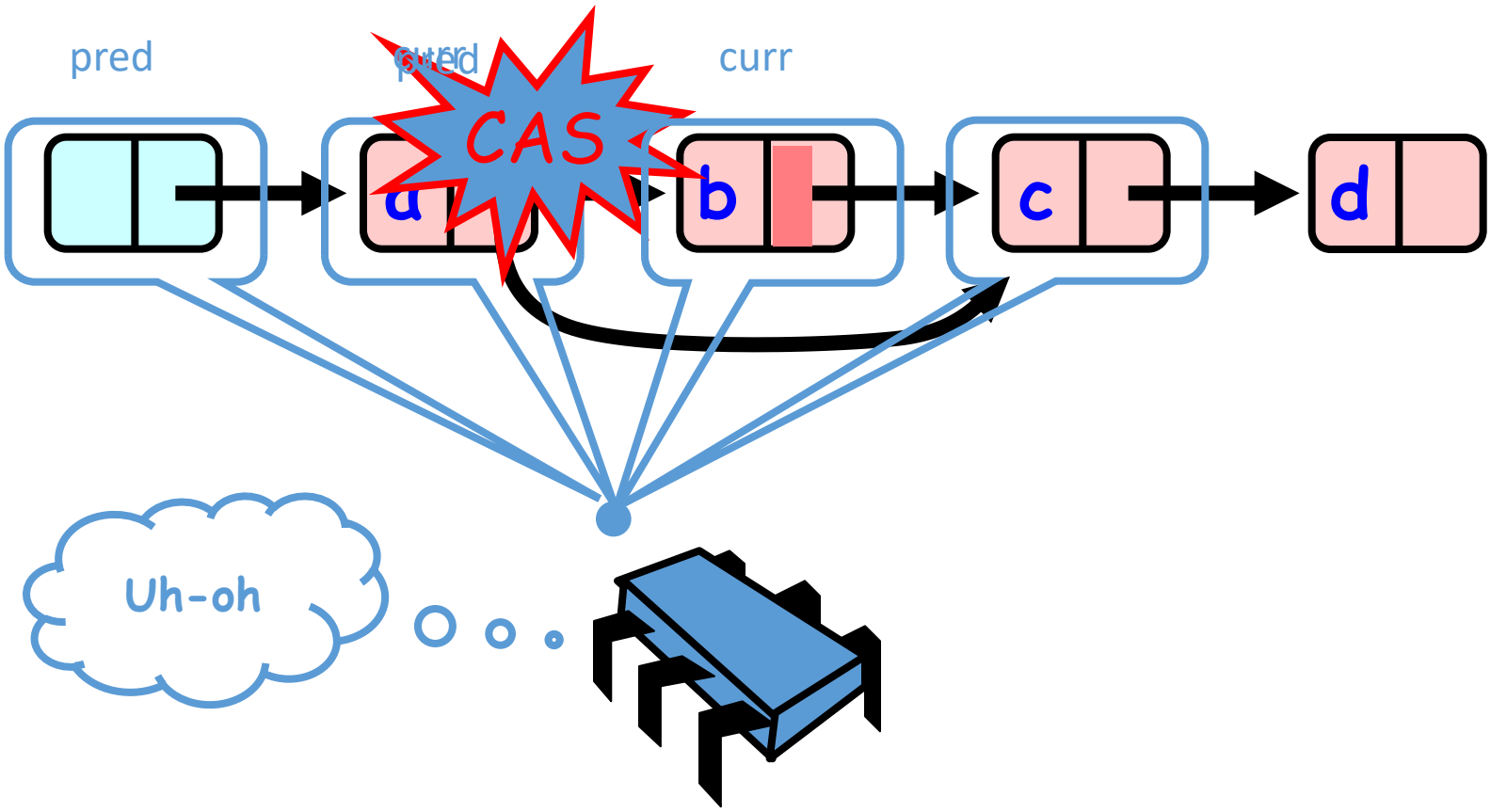
Removing a Node



Traversing the List

- Q: what do you do when you find a "logically" deleted node in your path?
- A: finish the job.
  - CAS the predecessor's next field
  - Proceed (repeat as needed)

Lock-Free Traversal (only Add and Remove)



The END