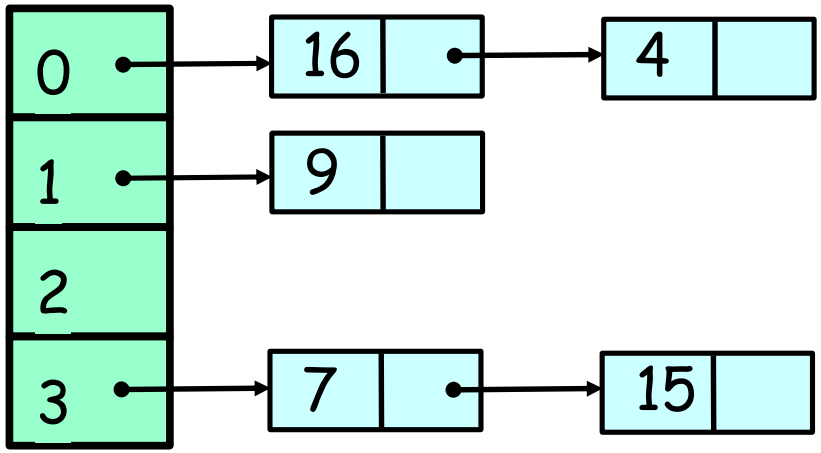
Multicore Computing
Lecture14 – Lock-Free Hashing

남 범 석

bnam@skku.edu

Background: Hashing



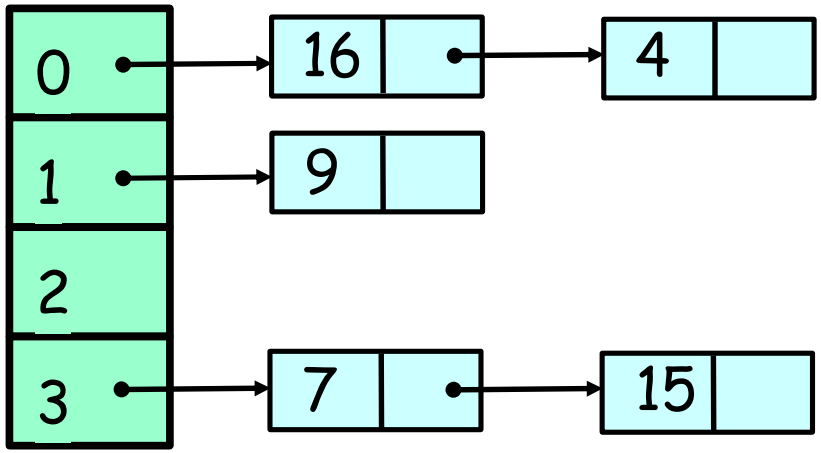
5 Items

h(k) = k mod 4

Lock-Free Hashing: No Brainer?

- Concurrent Hashing: Add(x), Remove(x), Contains(x)
  - Simple
  - Lock-free
- However, we don't know how to resize …

Background: Hashing

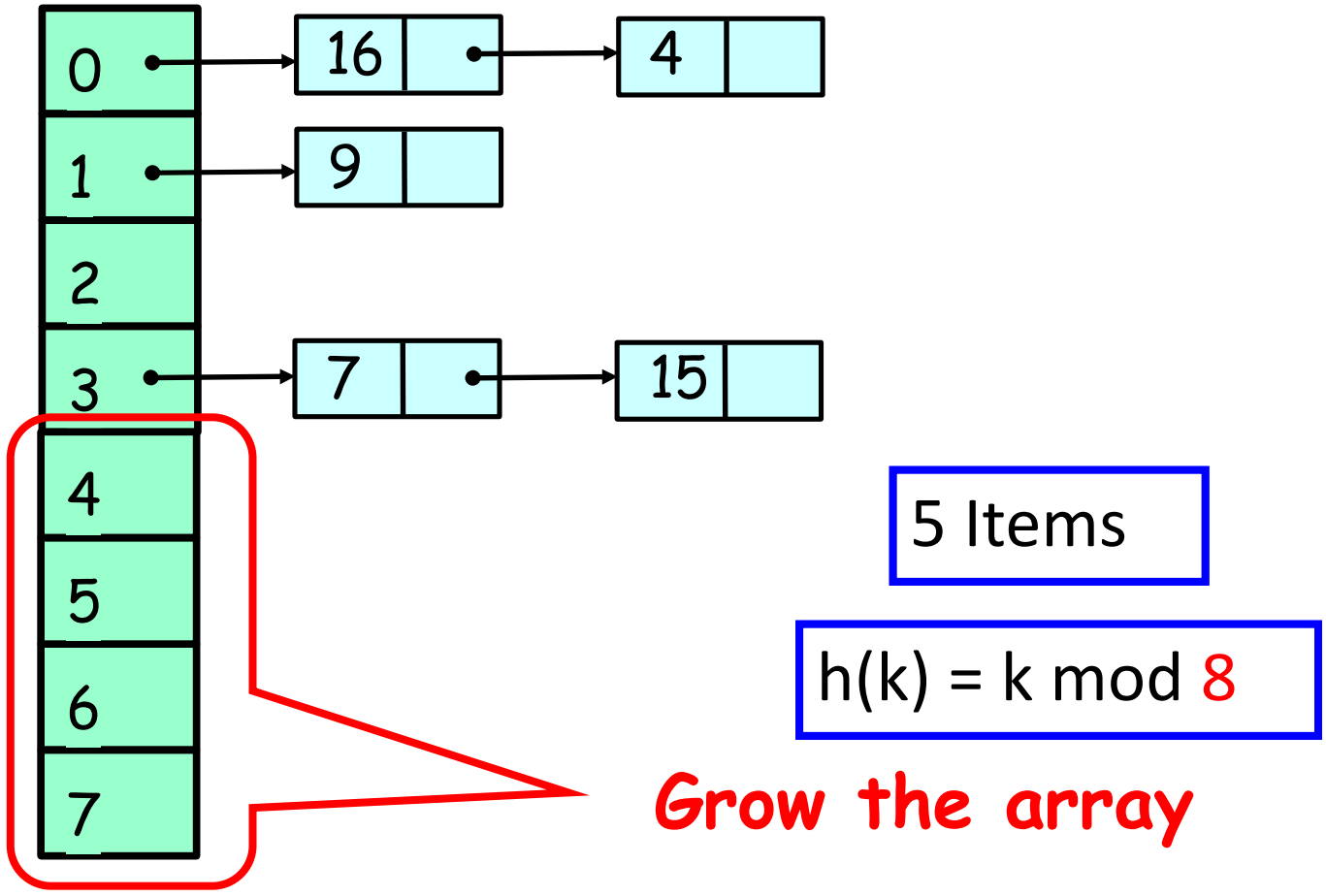


5 Items

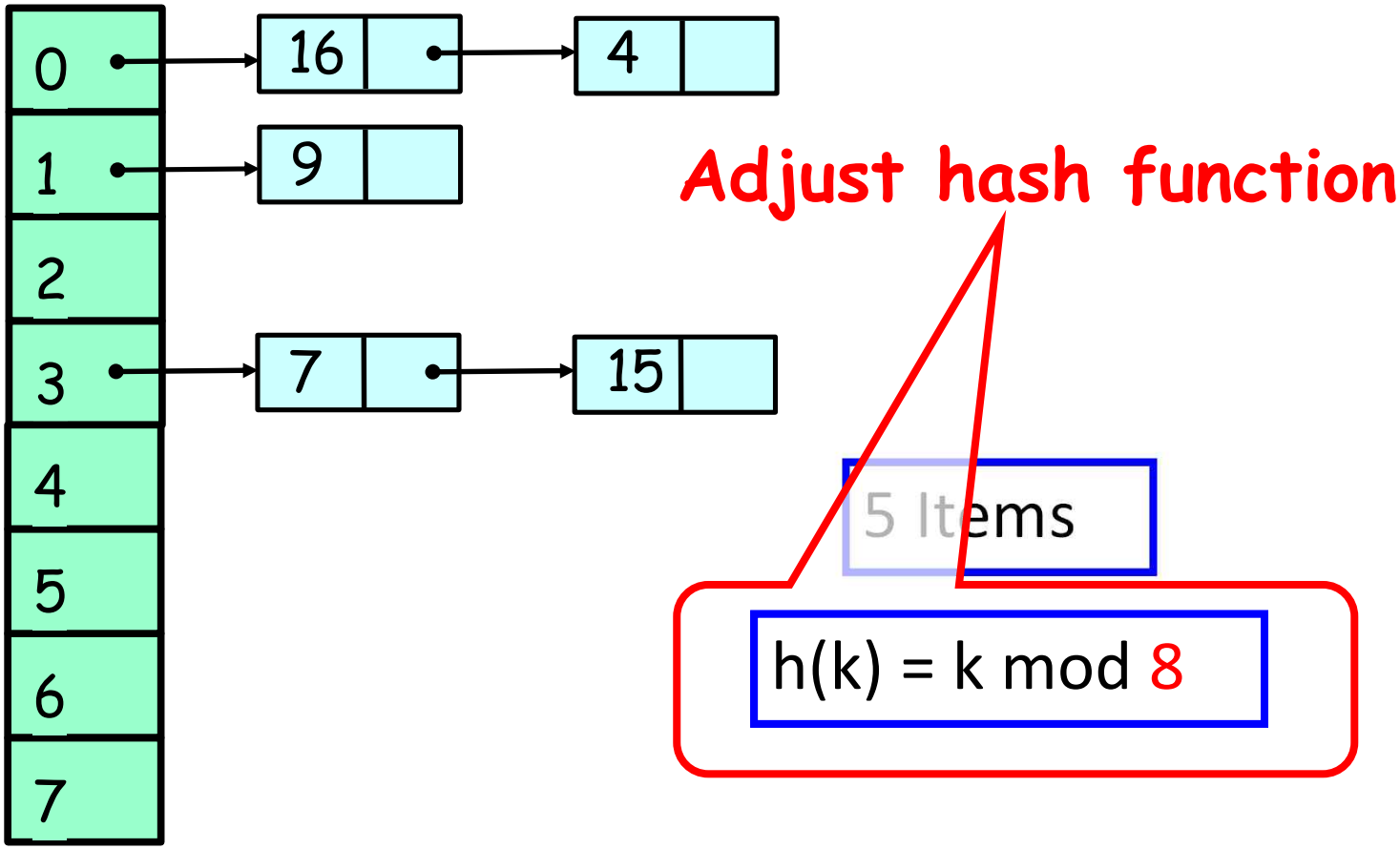
h(k) = k mod 4

Problem:
buckets getting too long

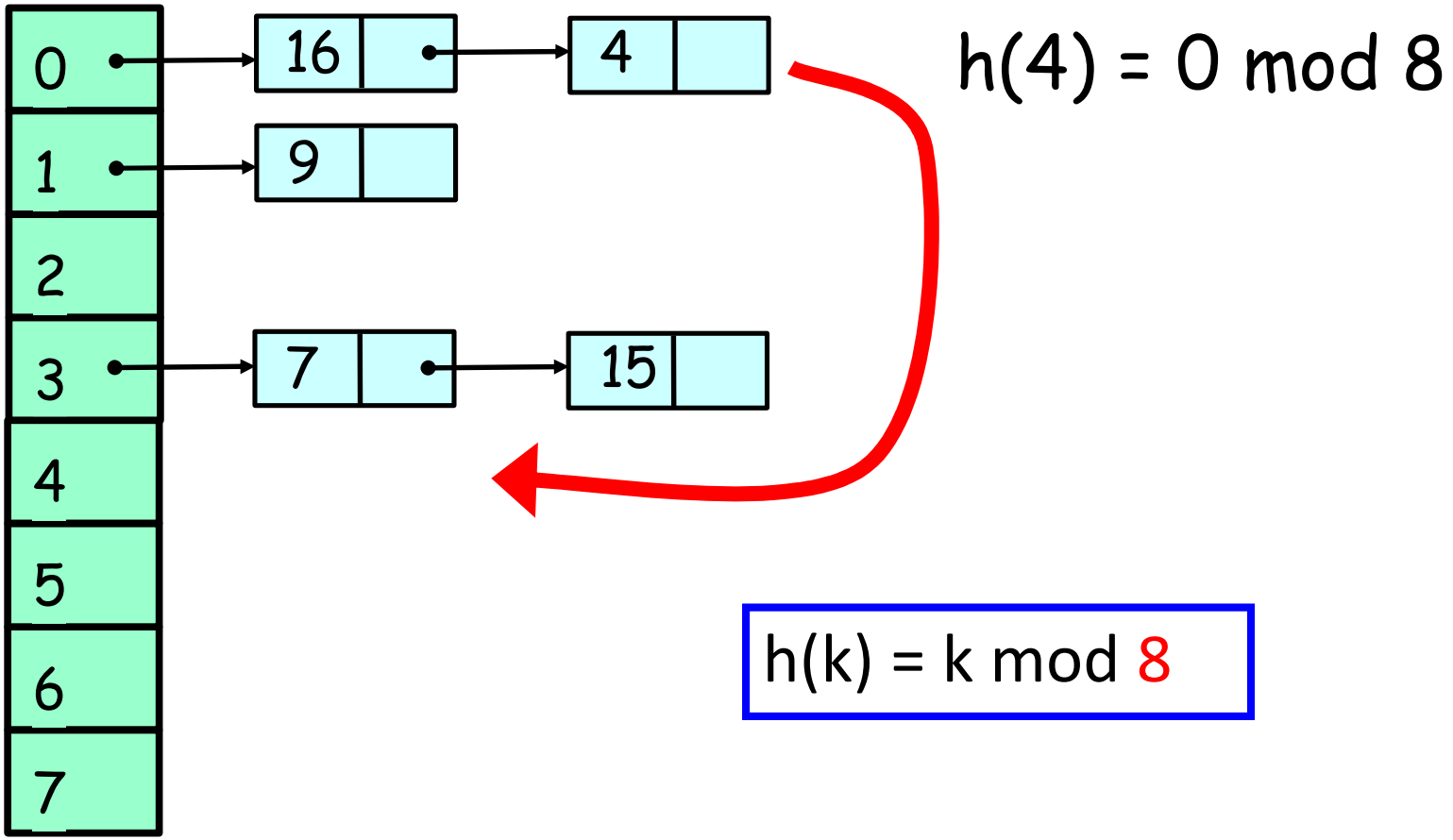
Background: Hashing



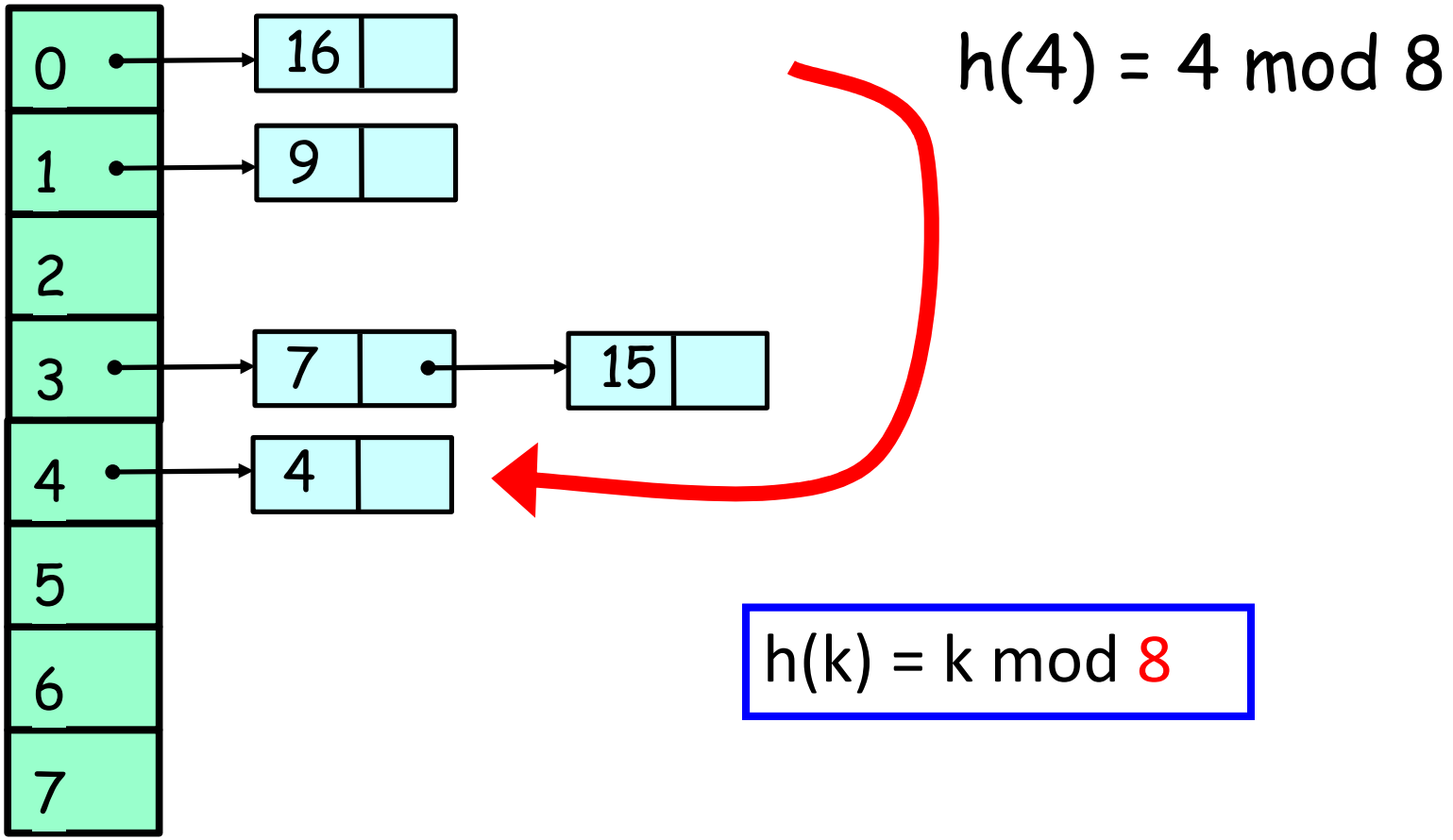
Background: Hashing



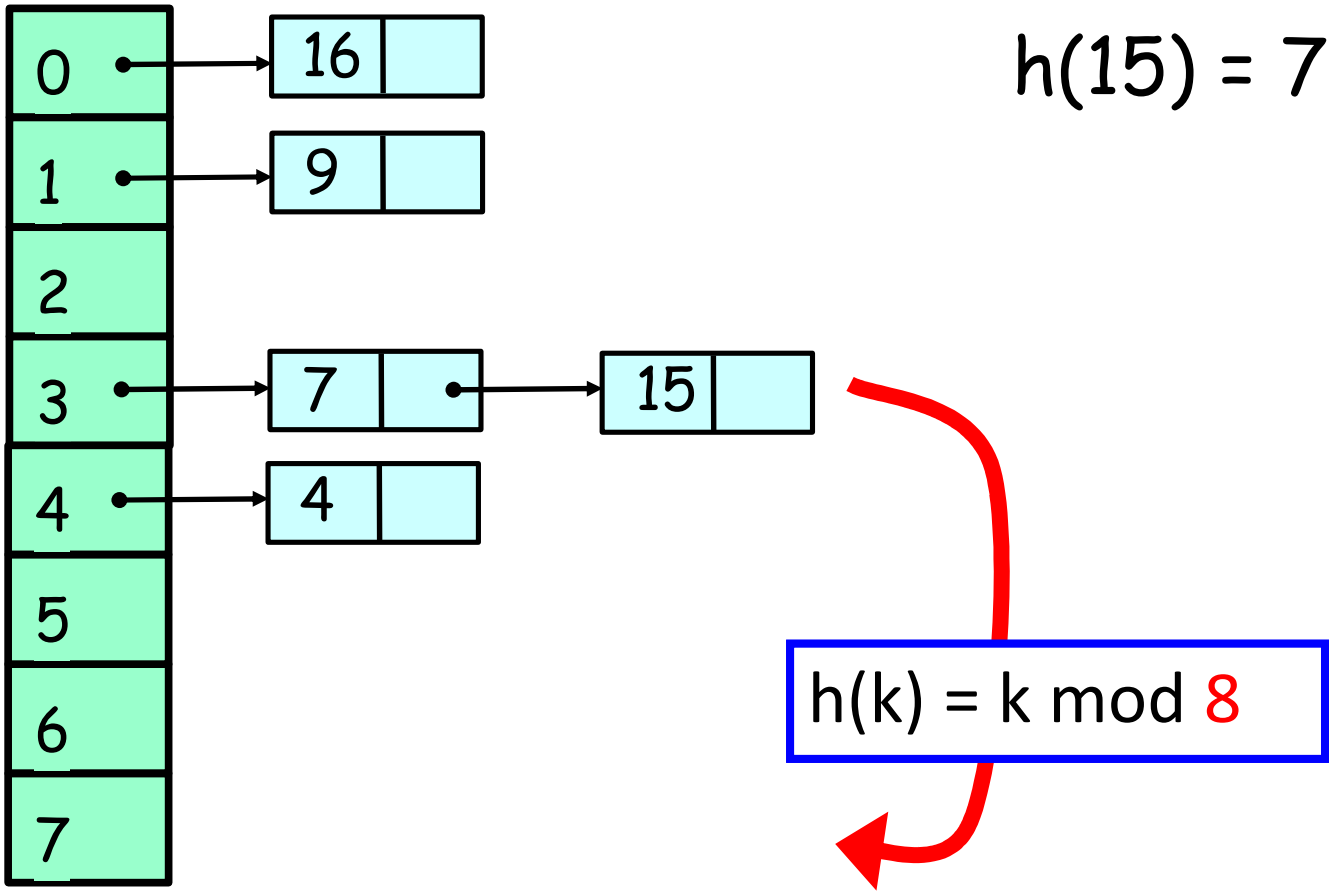
Background: Re-Hashing (Resizing)



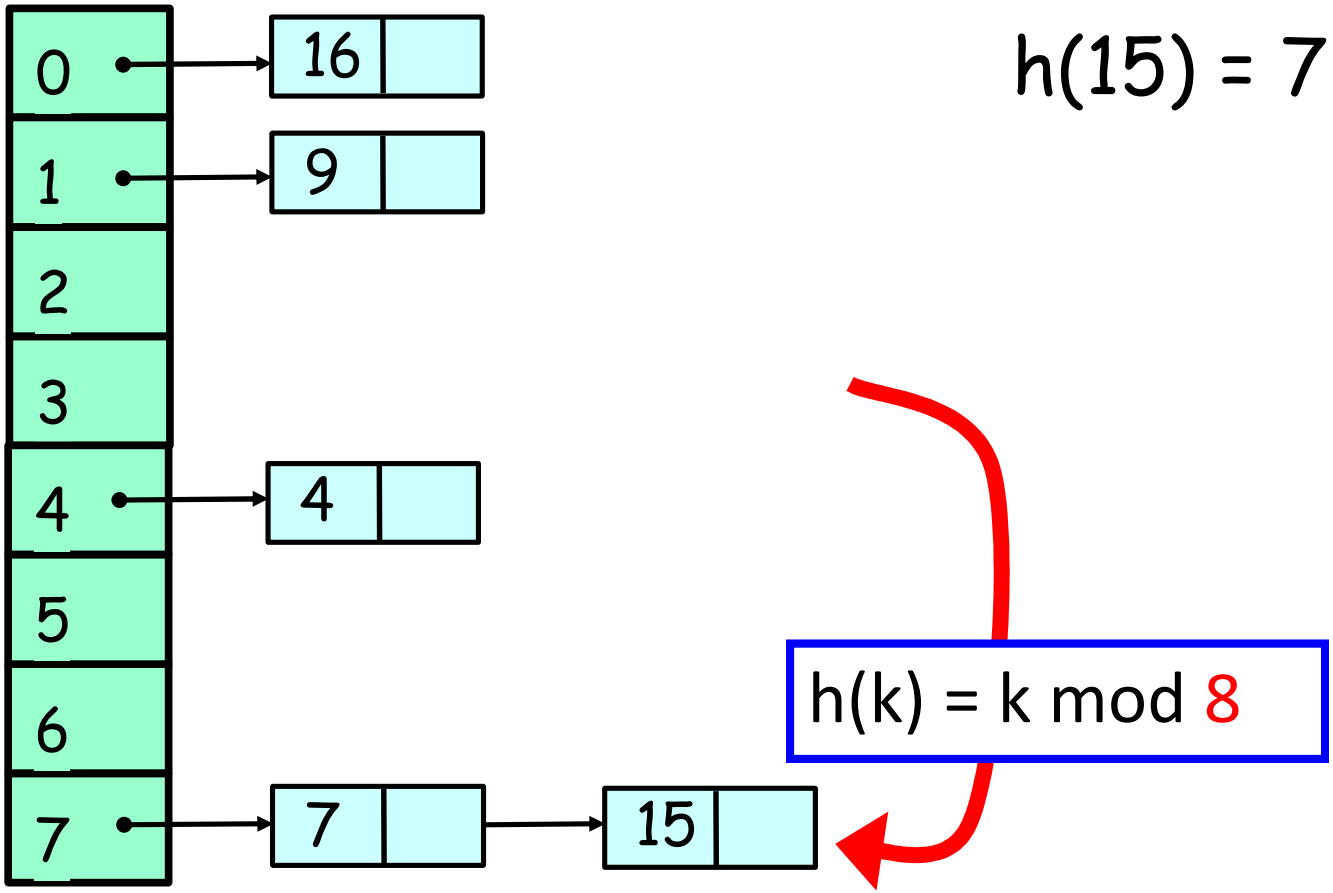
Background: Re-Hashing (Resizing)



Background: Re-Hashing (Resizing)



Background: Re-Hashing (Resizing)



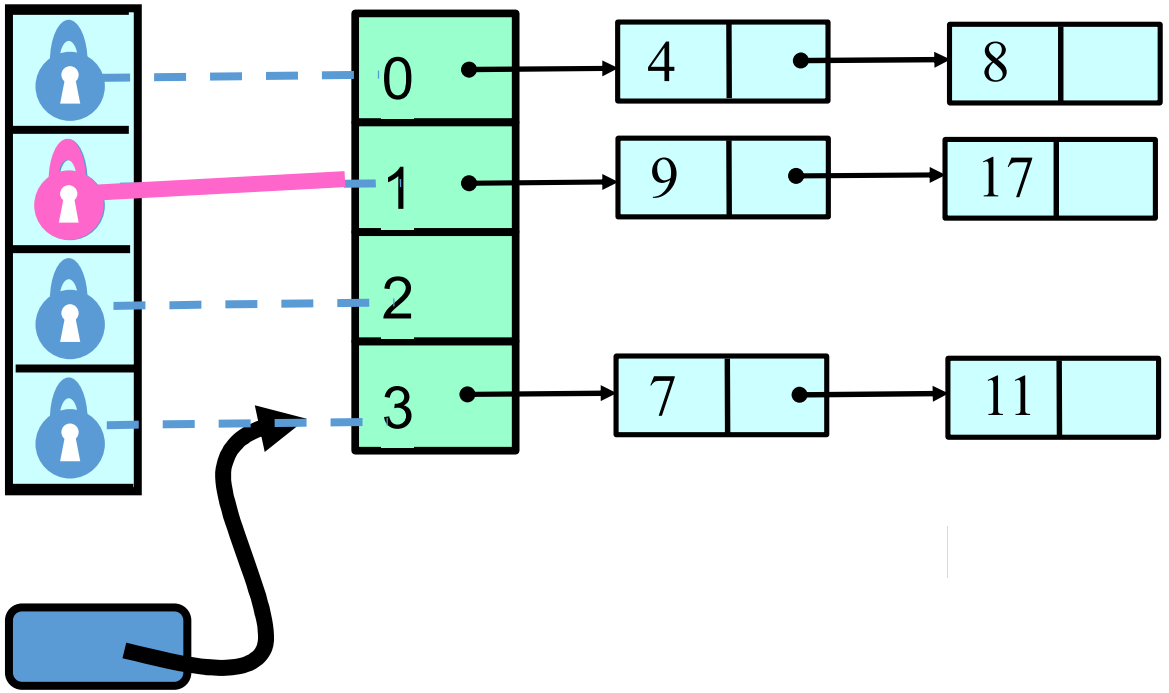
Is Resizing Necessary?

- Constant-time method calls require
  - Constant-length buckets
  - Table size proportional to set size
  - As set grows, must be able to resize

Coarse-Grained Locking

- Good parts
  - Simple
  - Hard to mess up
- Bad parts
  - Sequential bottleneck

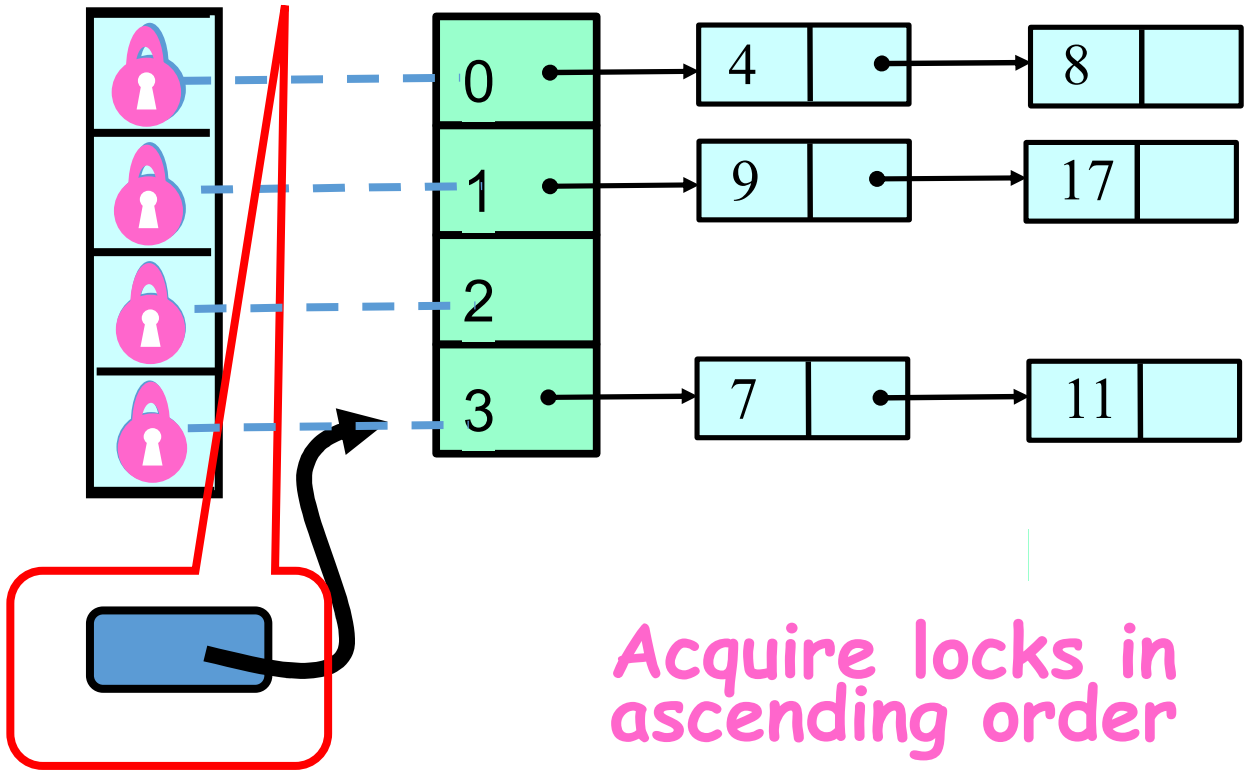
Fine-grained Locking



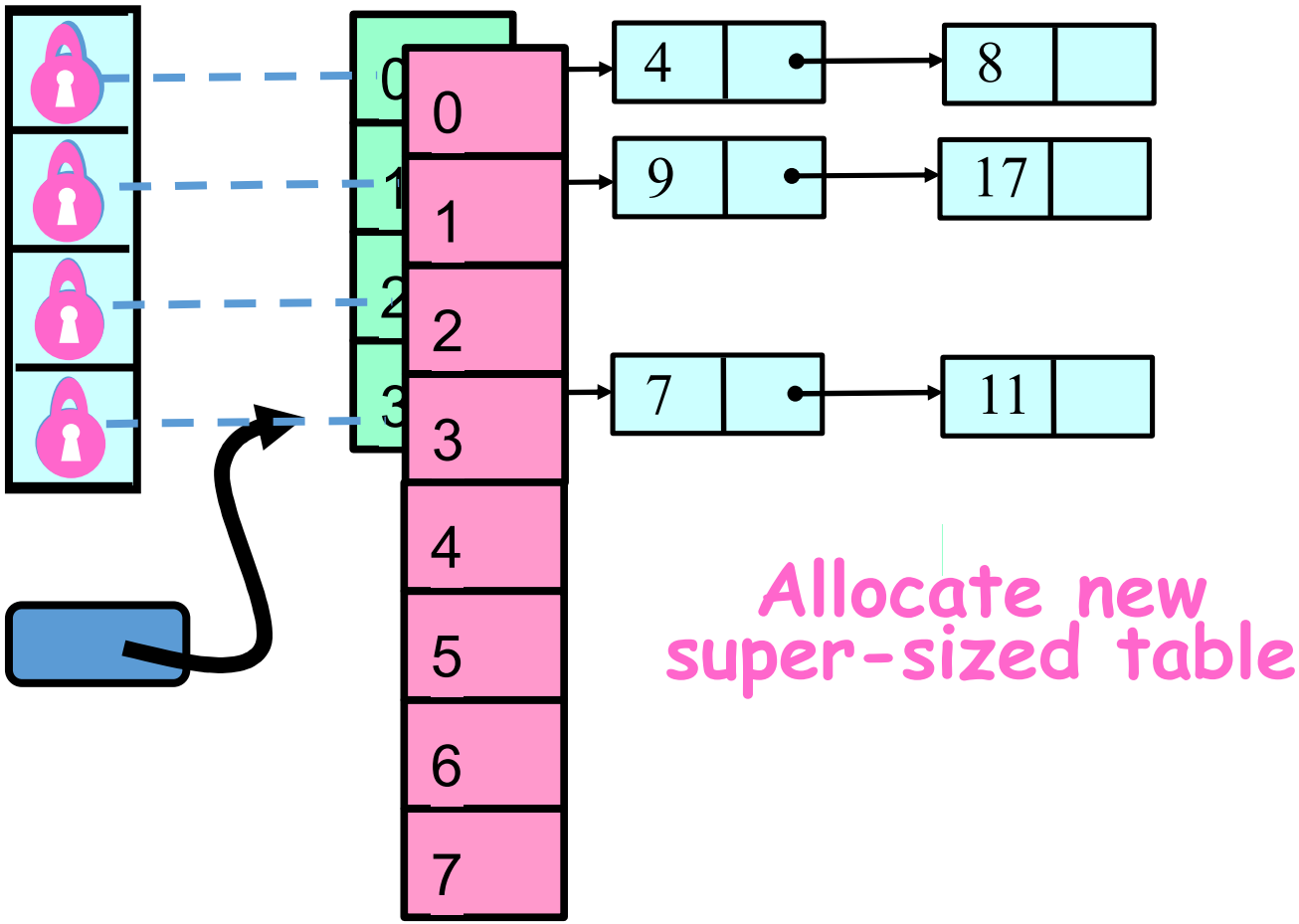
Each lock associated with one bucket

Fine-grained Locking: Resize This

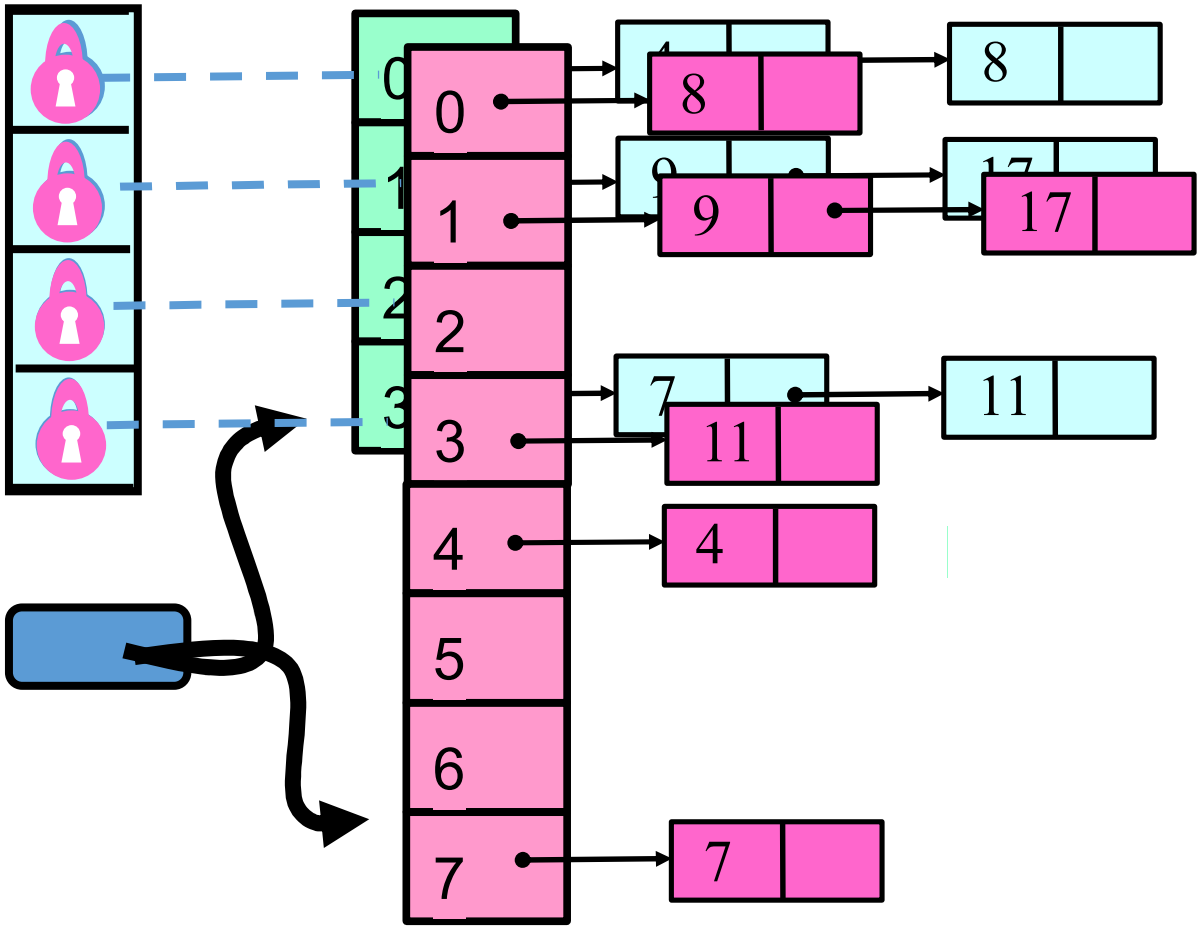
- Make sure table reference didn't change between resize decision and lock acquisition



Fine-grained Locking: Resize This

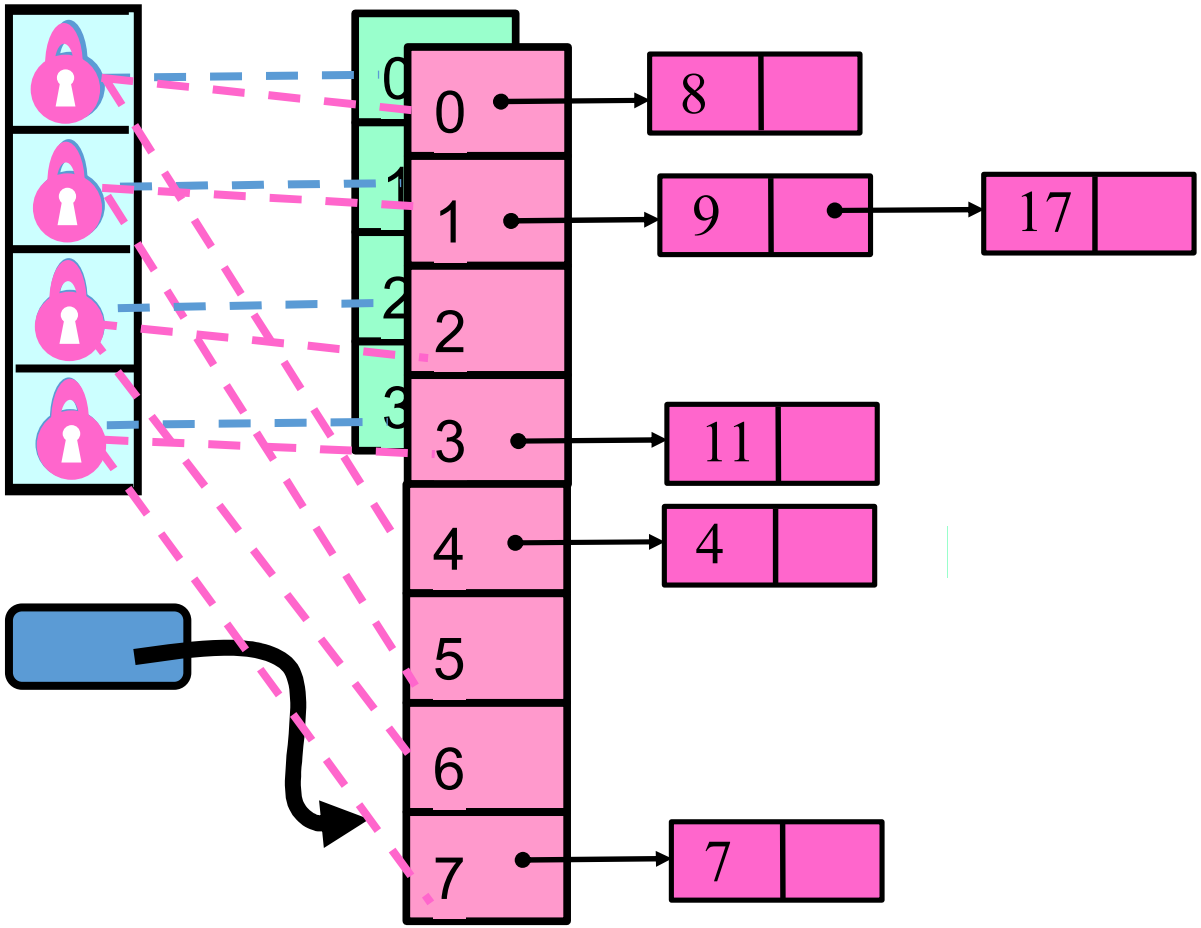


Fine-grained Locking: Resize This



Fine-grained Locking: Resize This

Striped Locks: each lock now associated with two buckets



Fine-grained Locking: Observations

- We grow the table, but not locks
  - Resizing lock array is tricky …
- We use sequential lists
  - Not LockFreeList lists
  - If we're locking anyway, why pay?

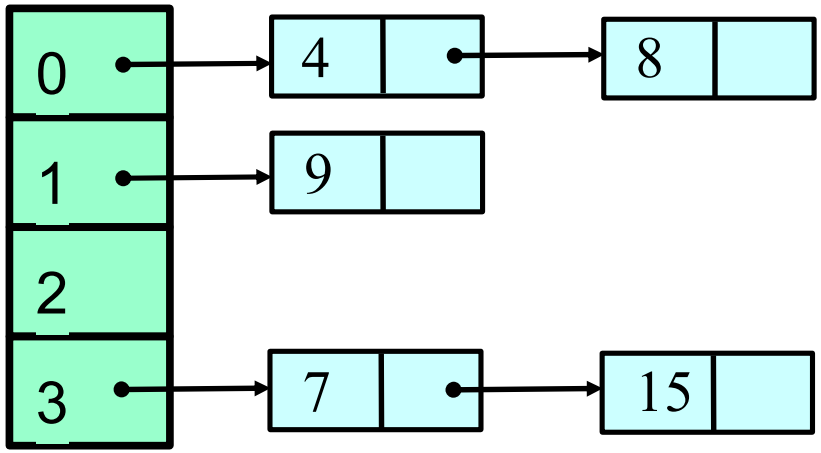
Fine-grained Locking:

- We can resize the table
- But not the locks
- Debatable whether method calls are constant-time in presence of contention …

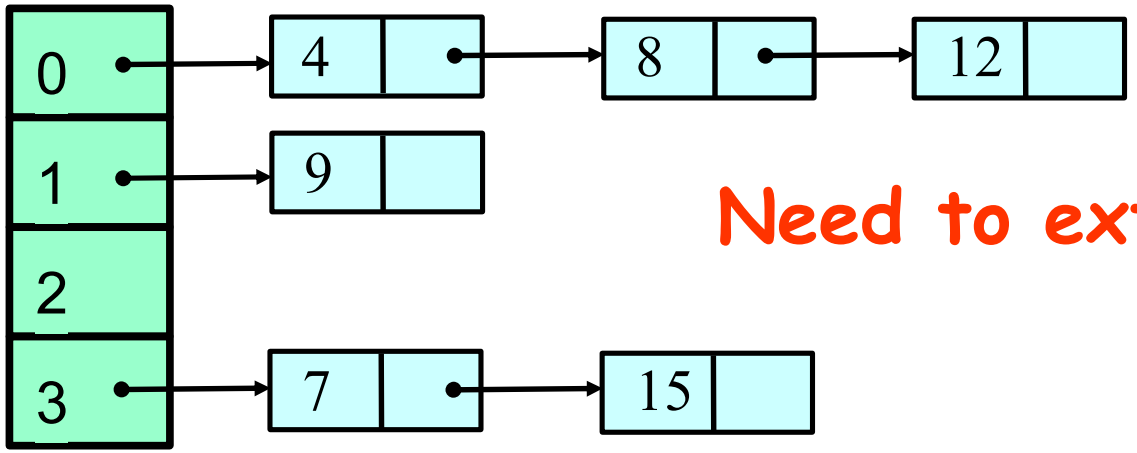
Stop The World Resizing

- Resizing stops all concurrent operations
- What about an incremental resize?
- Must avoid locking the table
- A lock-free table + incremental resizing?

Lock-Free Resizing Problem

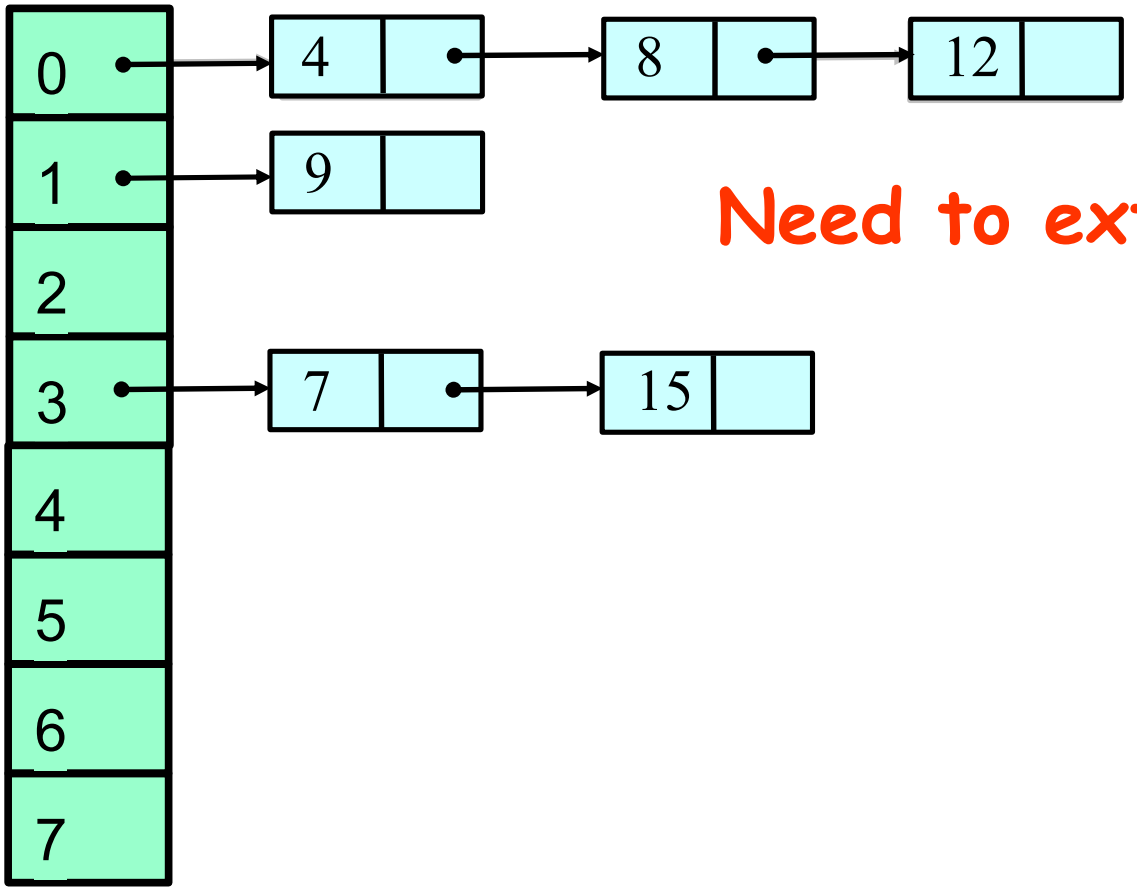


Lock-Free Resizing Problem



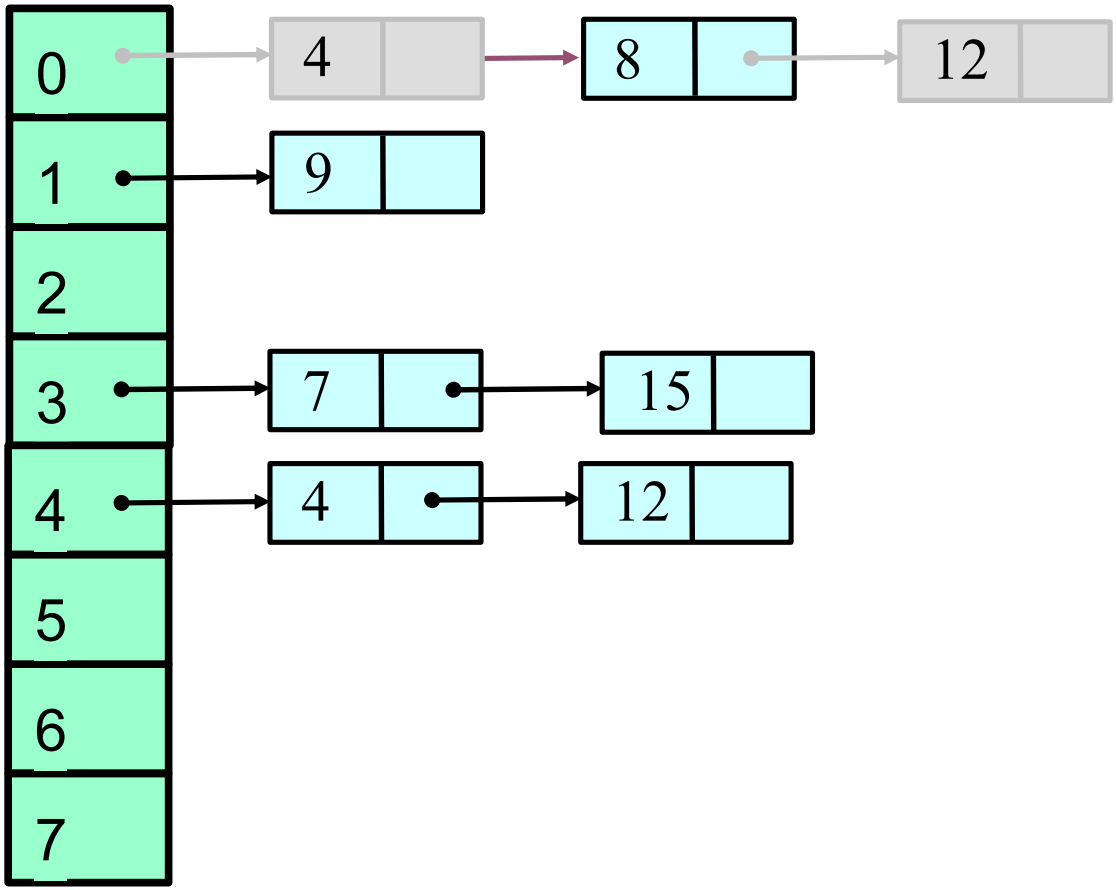
Need to extend table

Lock-Free Resizing Problem

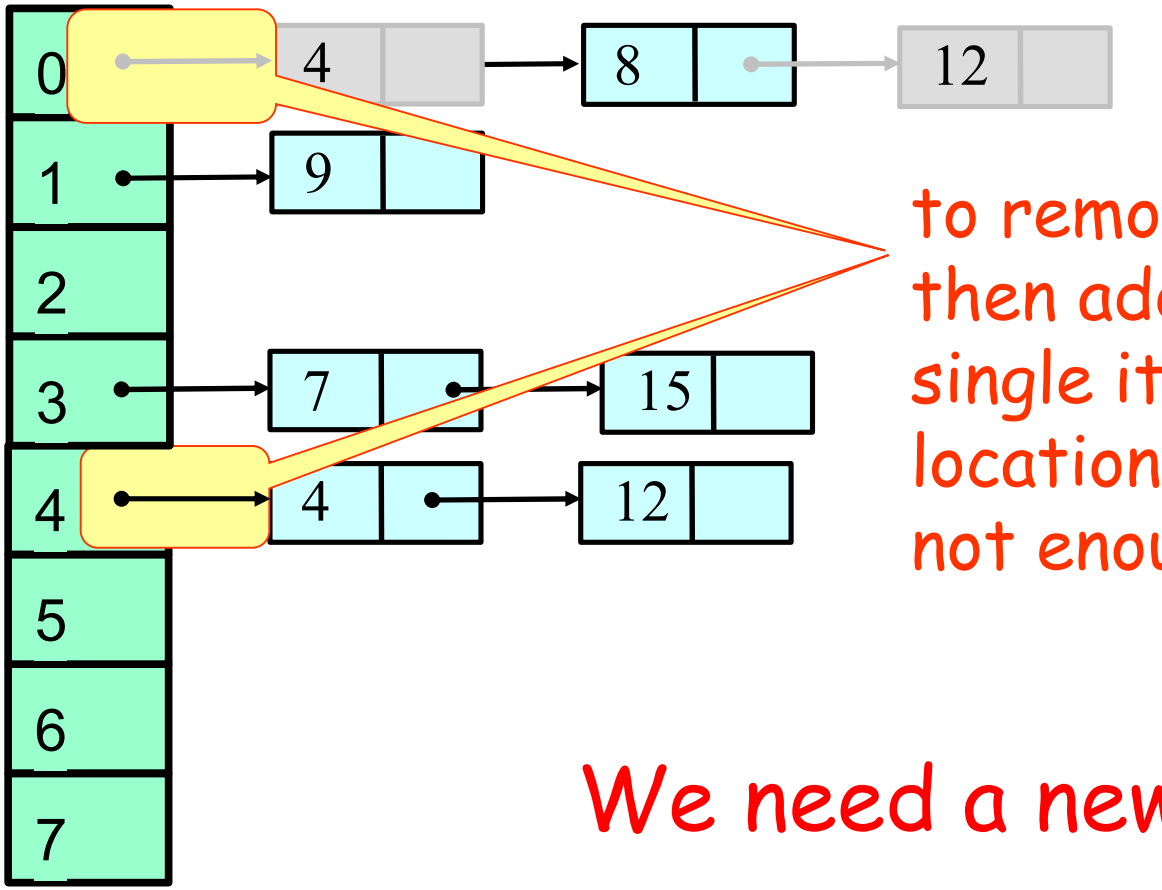


Need to extend table

Lock-Free Resizing Problem



Lock-Free Resizing Problem

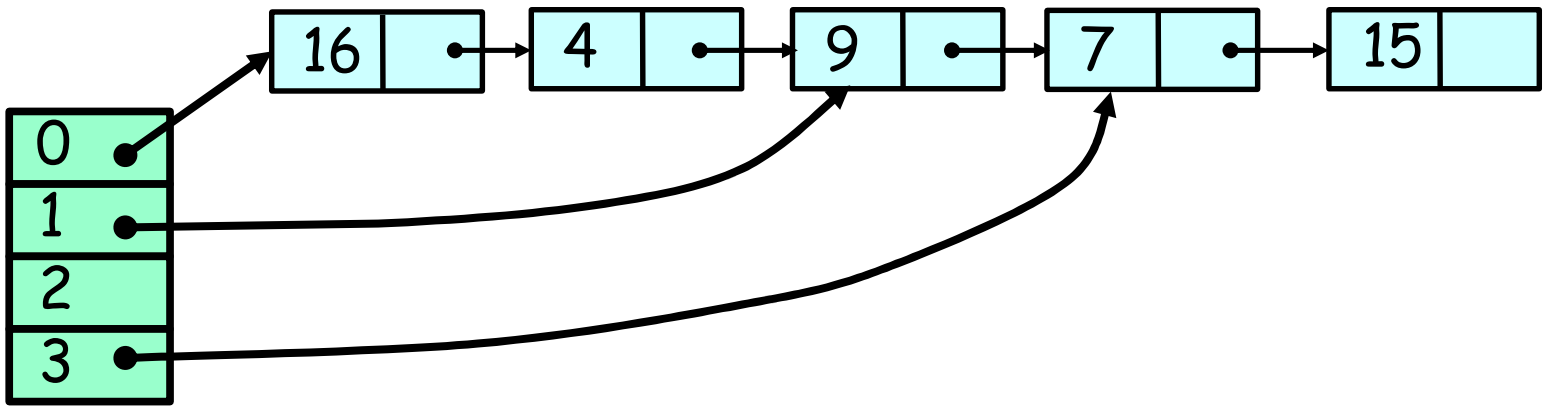


to remove and then add even a single item single location CAS not enough

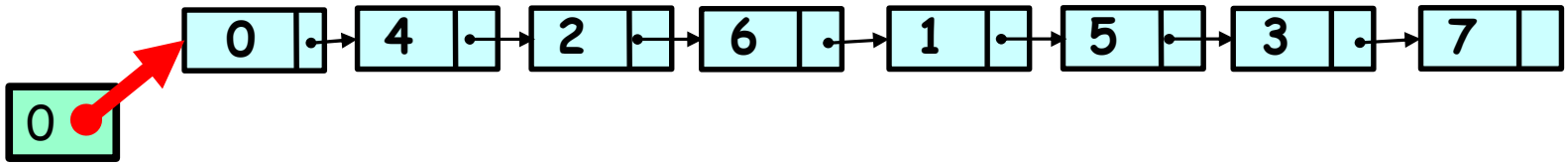
We need a new idea…

Don't move the items

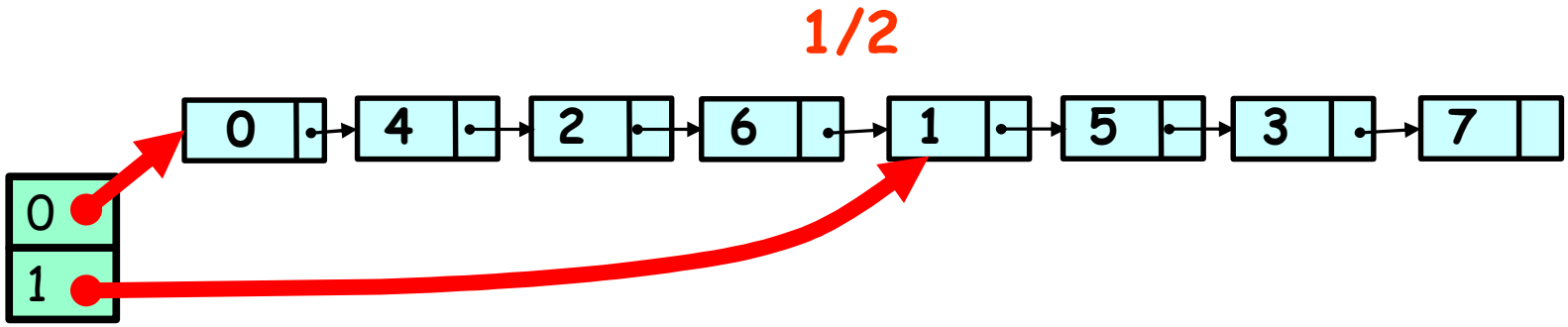
- Move the buckets instead
- Keep all items in a single lock-free list
- Buckets become "shortcut pointers" into the list



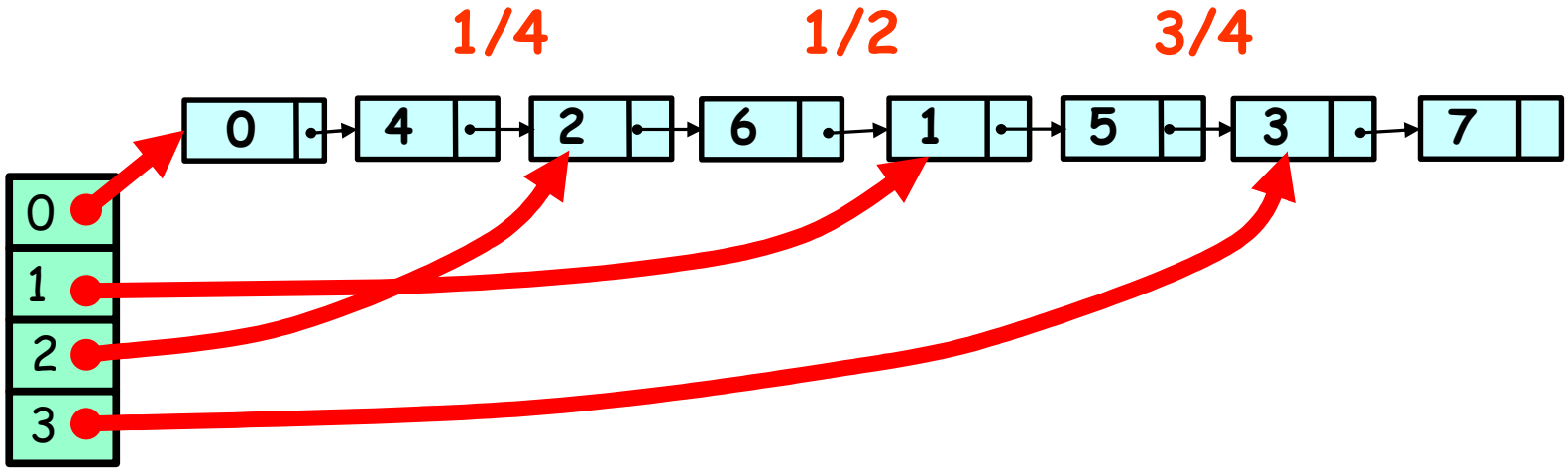
Recursive Split Ordering



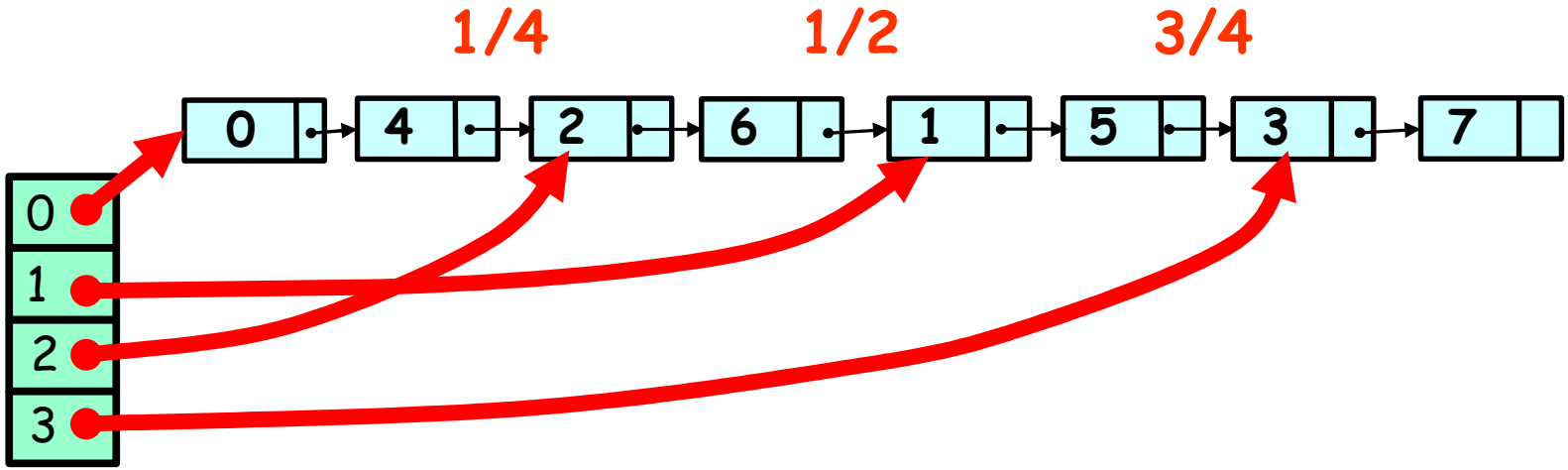
Recursive Split Ordering



Recursive Split Ordering

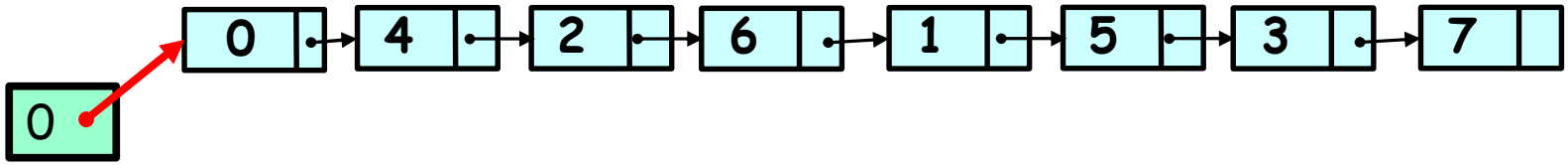


Recursive Split Ordering

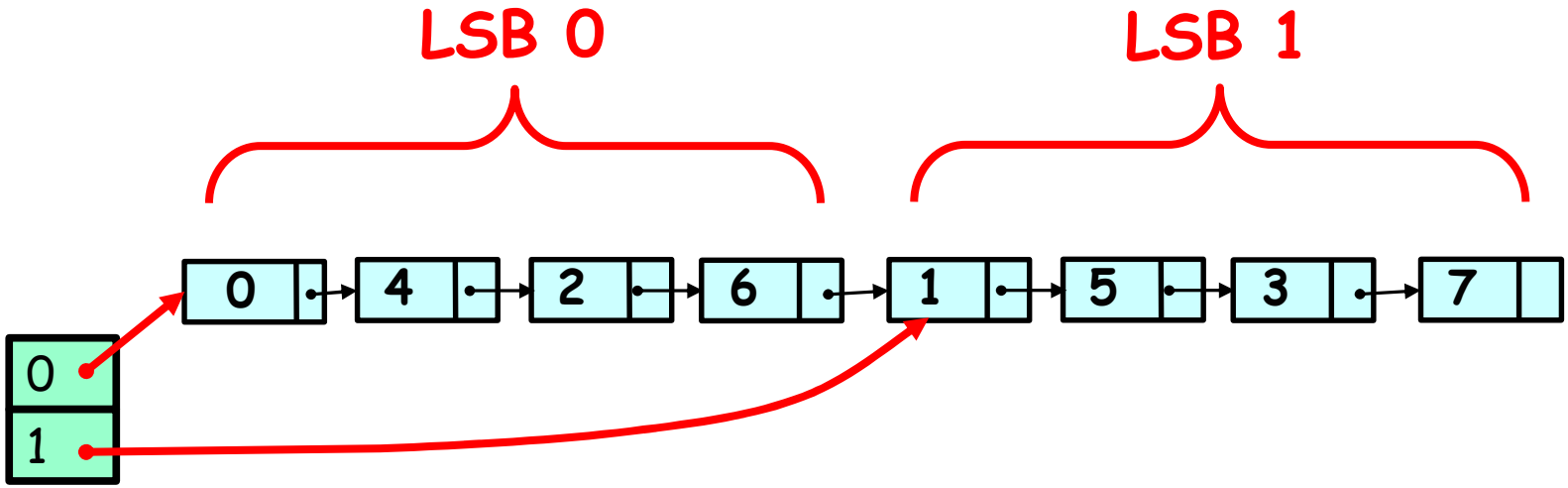


List entries sorted in order that allows recursive splitting. How?

Recursive Split Ordering

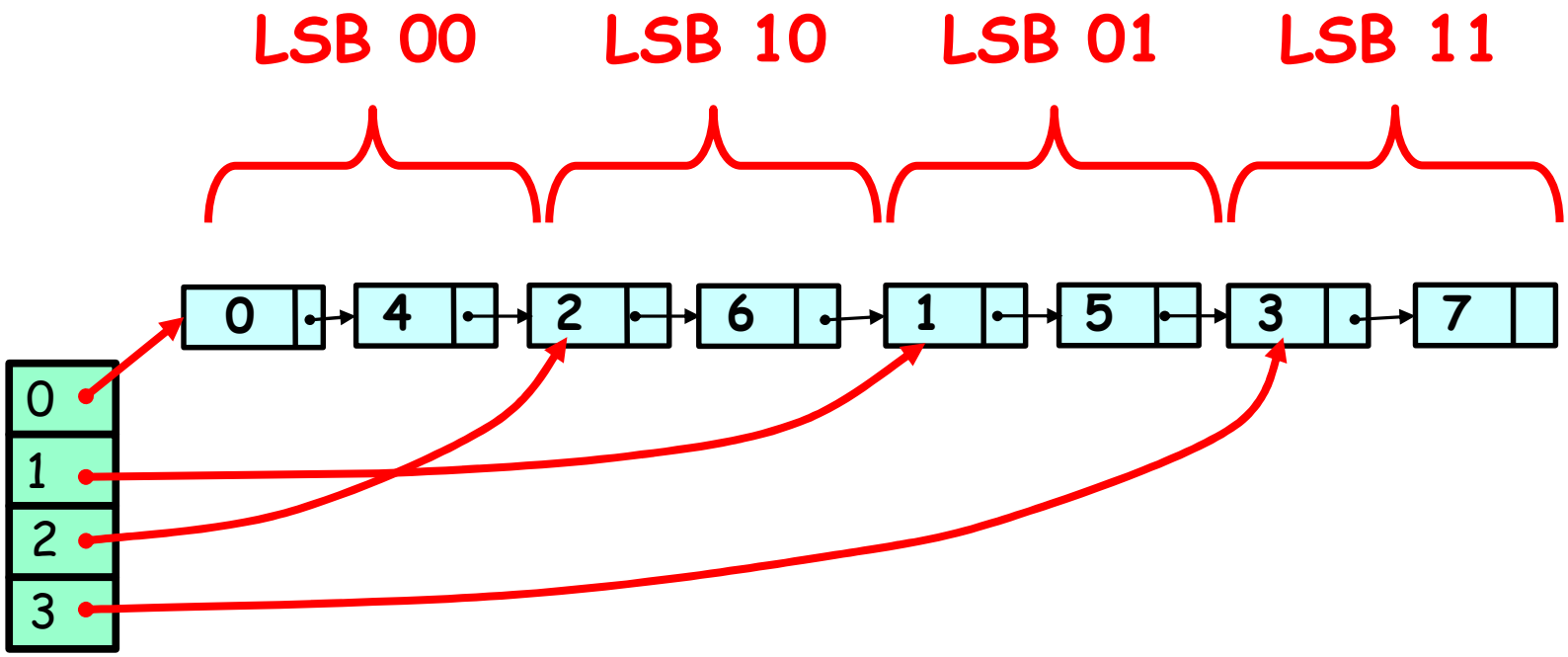


Recursive Split Ordering



LSB = Least significant Bit

Recursive Split Ordering



Split-Order

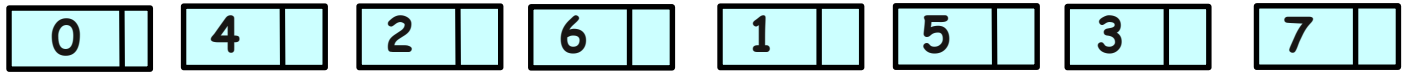
- If the table size is 2^i,
  - Bucket b contains keys k
    - k mod 2^i = b
  - bucket index consists of key's i LSBs

When Table Splits

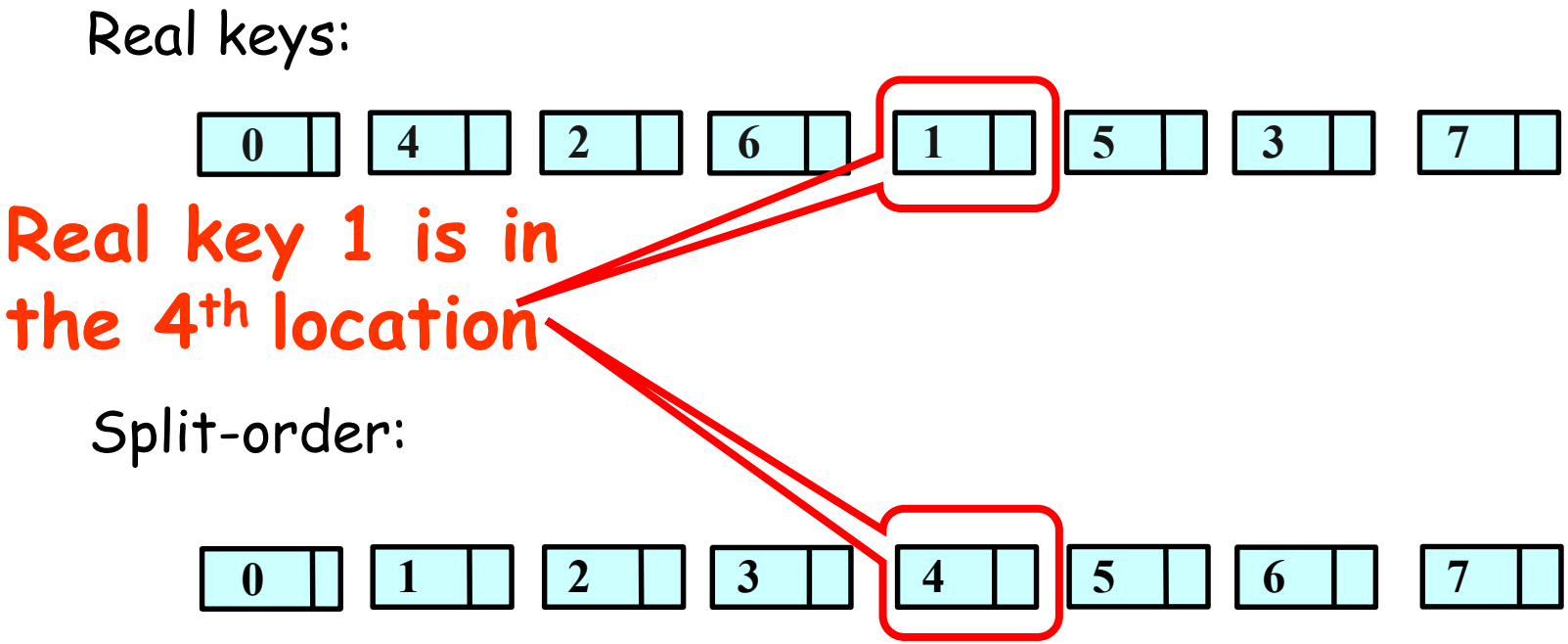
- Some keys stay
  - $b = k \bmod (2^i)$
- Some move
  - $b+2^i = k \bmod (2^i)$
- Determined by (i+1)st bit
  - Counting backwards
- Key must be accessible from both
  - Keys that will move must come later

A Bit of Magic

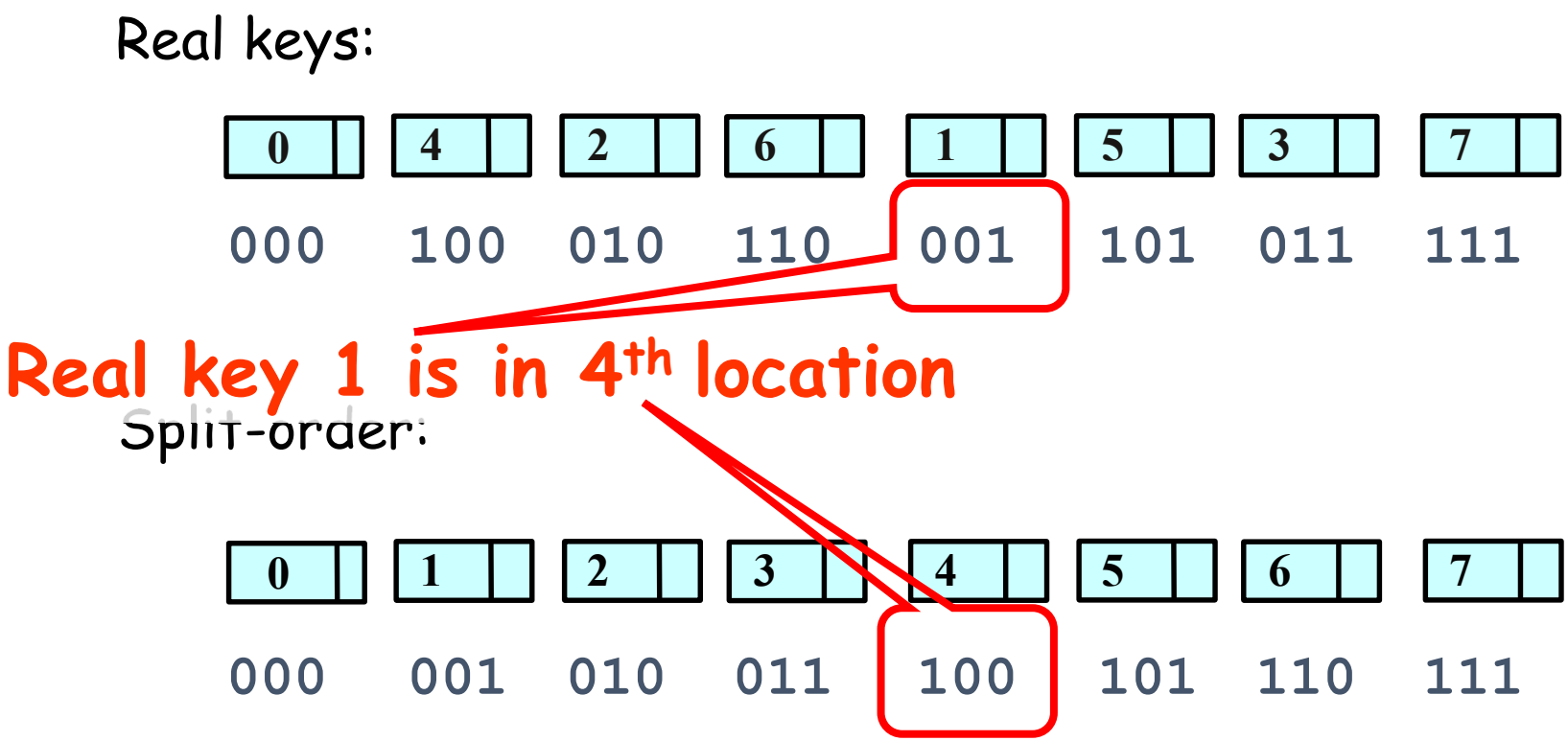
Real keys:



A Bit of Magic



A Bit of Magic



A Bit of Magic

Real keys:

000 100 010 110 001 101 011 111

Split-order:

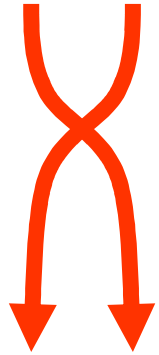
000 001 010 011 100 101 110 111

A Bit of Magic

Real keys:

000 100 010 110 001 101 011 111

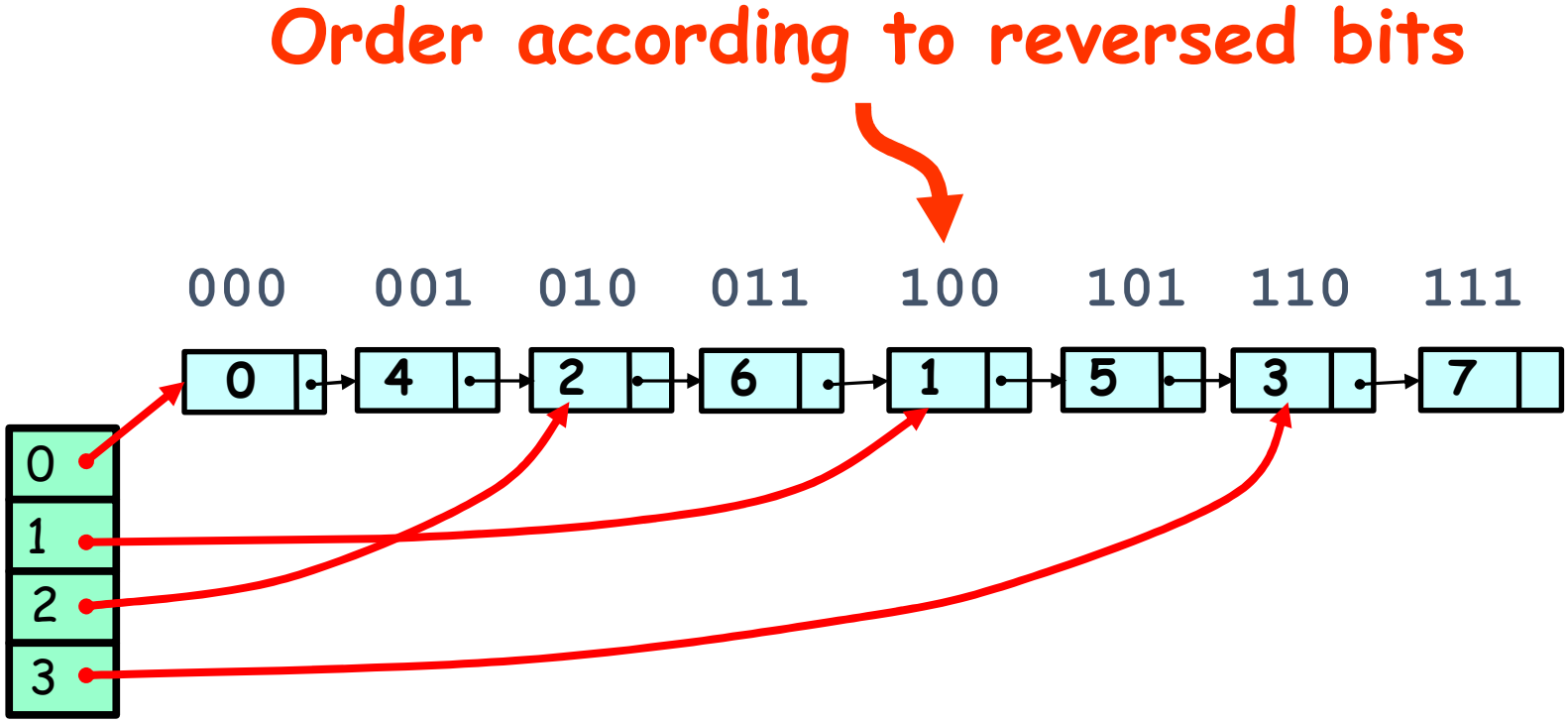
Split-order:



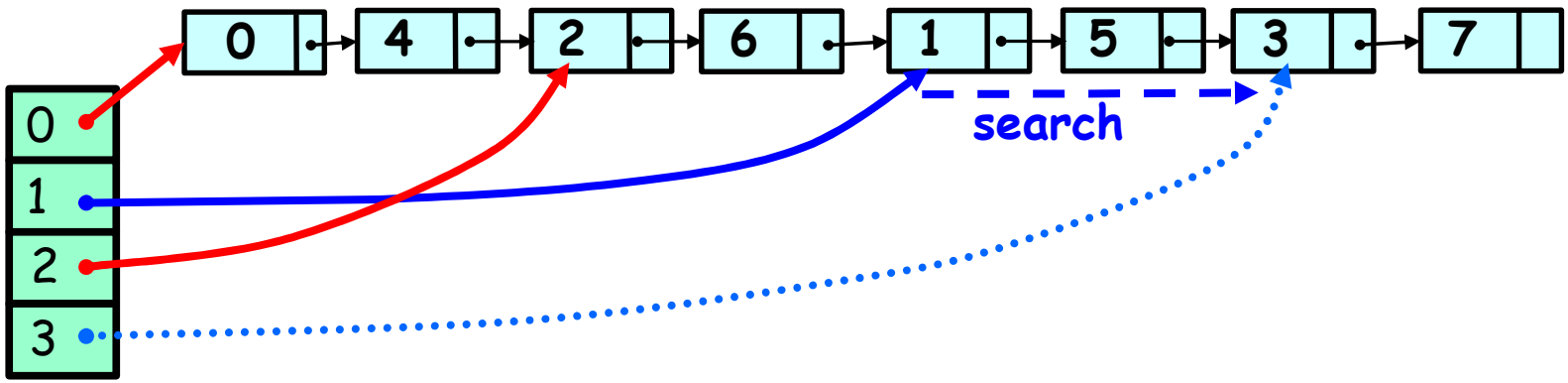
000 001 010 011 100 101 110 111

Just reverse the order of the key bits

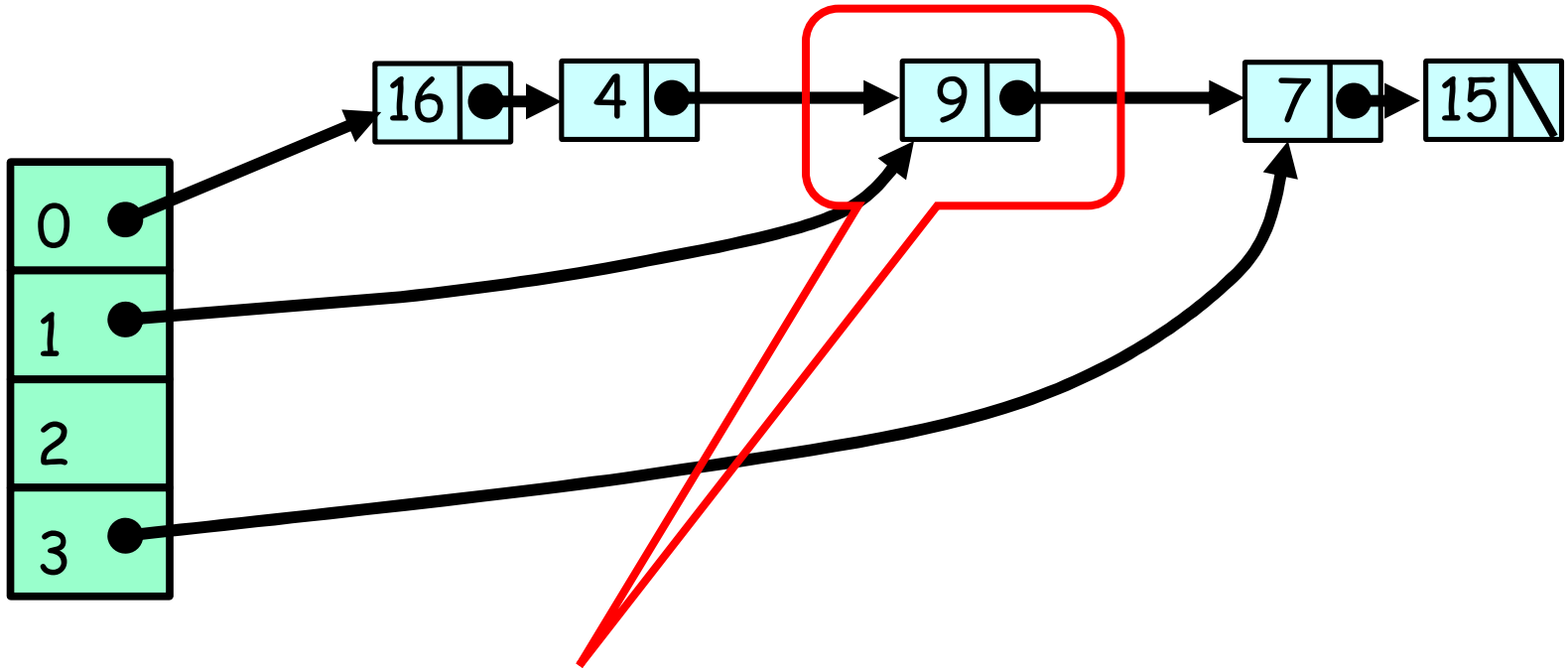
Split Ordered Hashing



Parent Always Provides a Short Cut

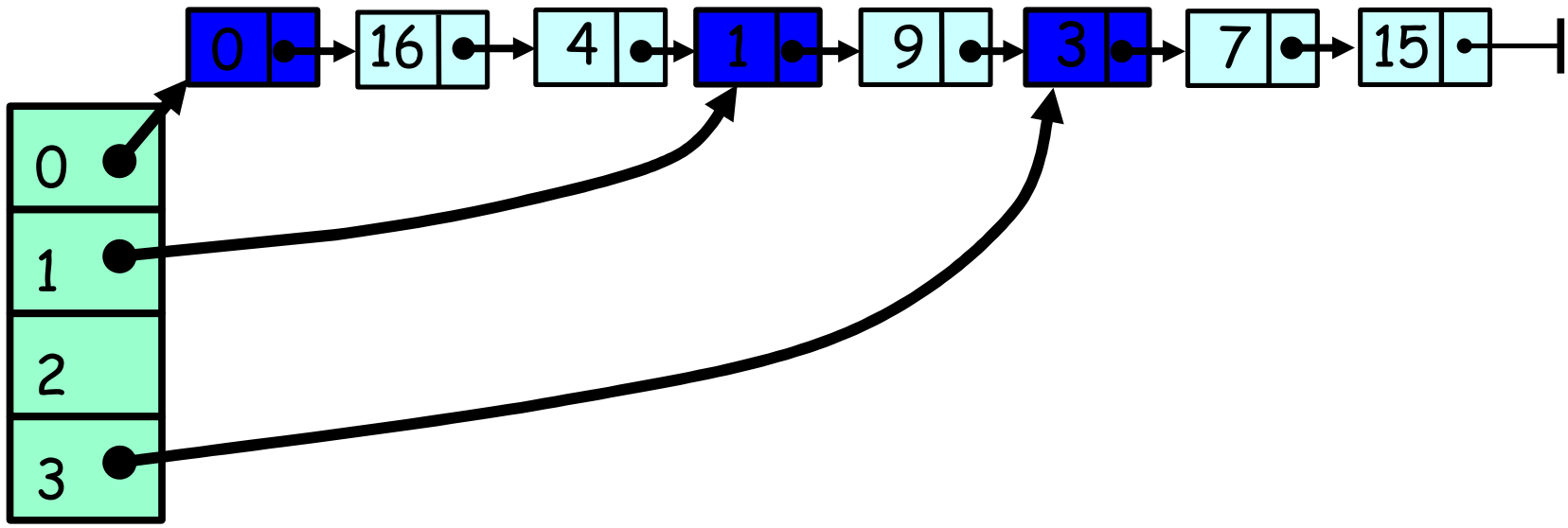


Sentinel Nodes



Problem: how to remove a node pointed by 2 sources using CAS

Sentinel Nodes



Solution: use a Sentinel node for each bucket

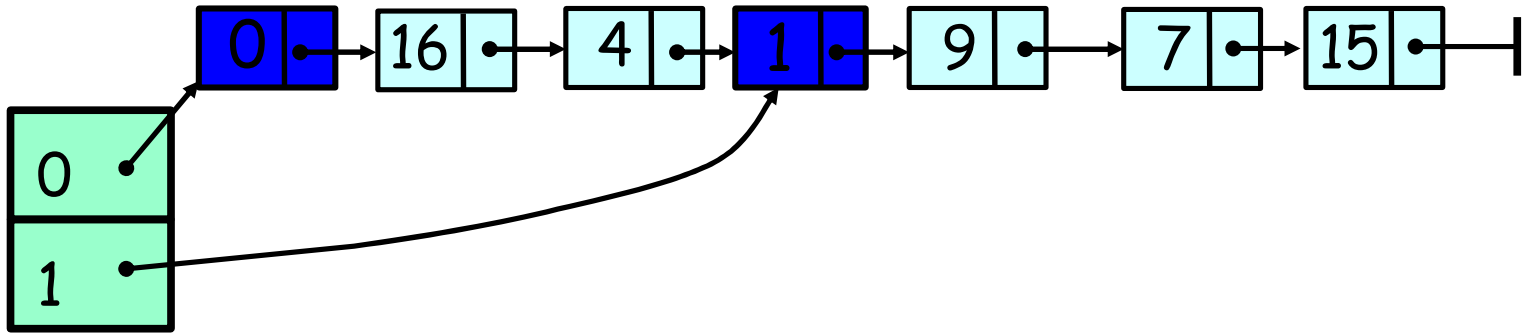
Sentinel vs Regular Keys

- Want sentinel key for i ordered
  - before all keys that hash to bucket i
  - after all keys that hash to bucket (i-1)

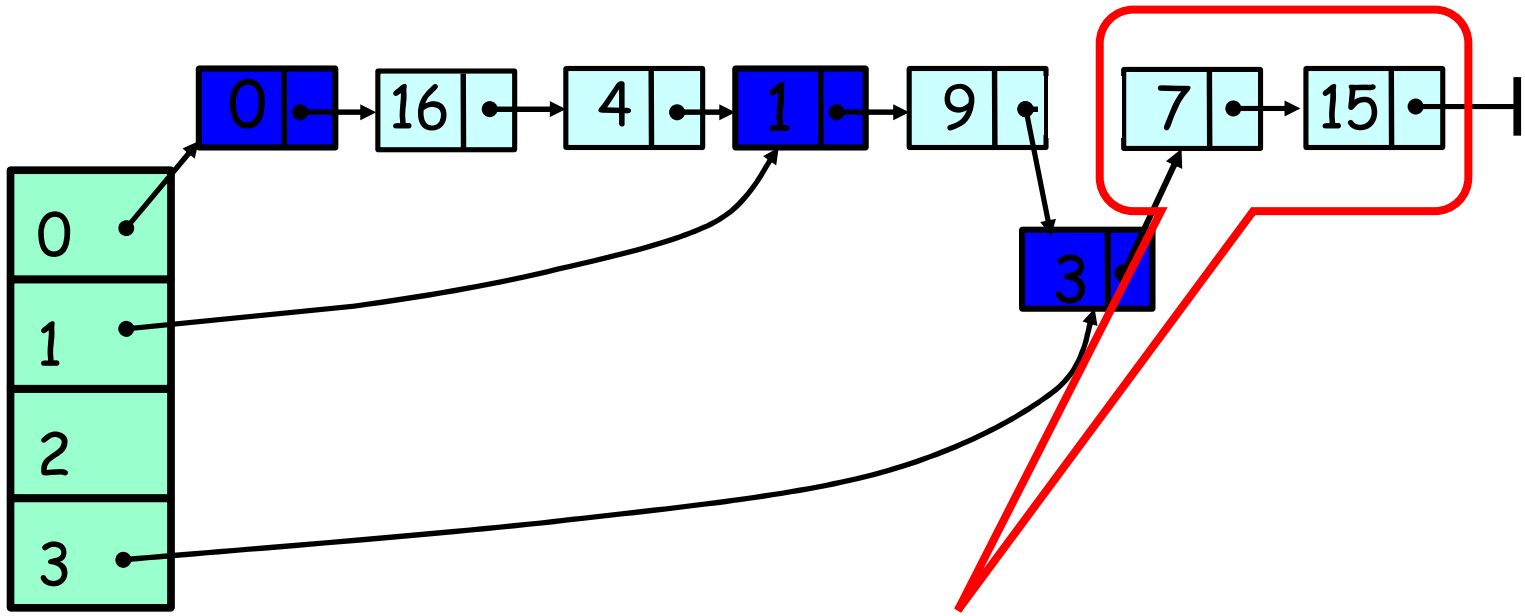
Splitting a Bucket

- We can now split a bucket
- In a lock-free manner
- Using two CAS() calls …

Initialization of Buckets

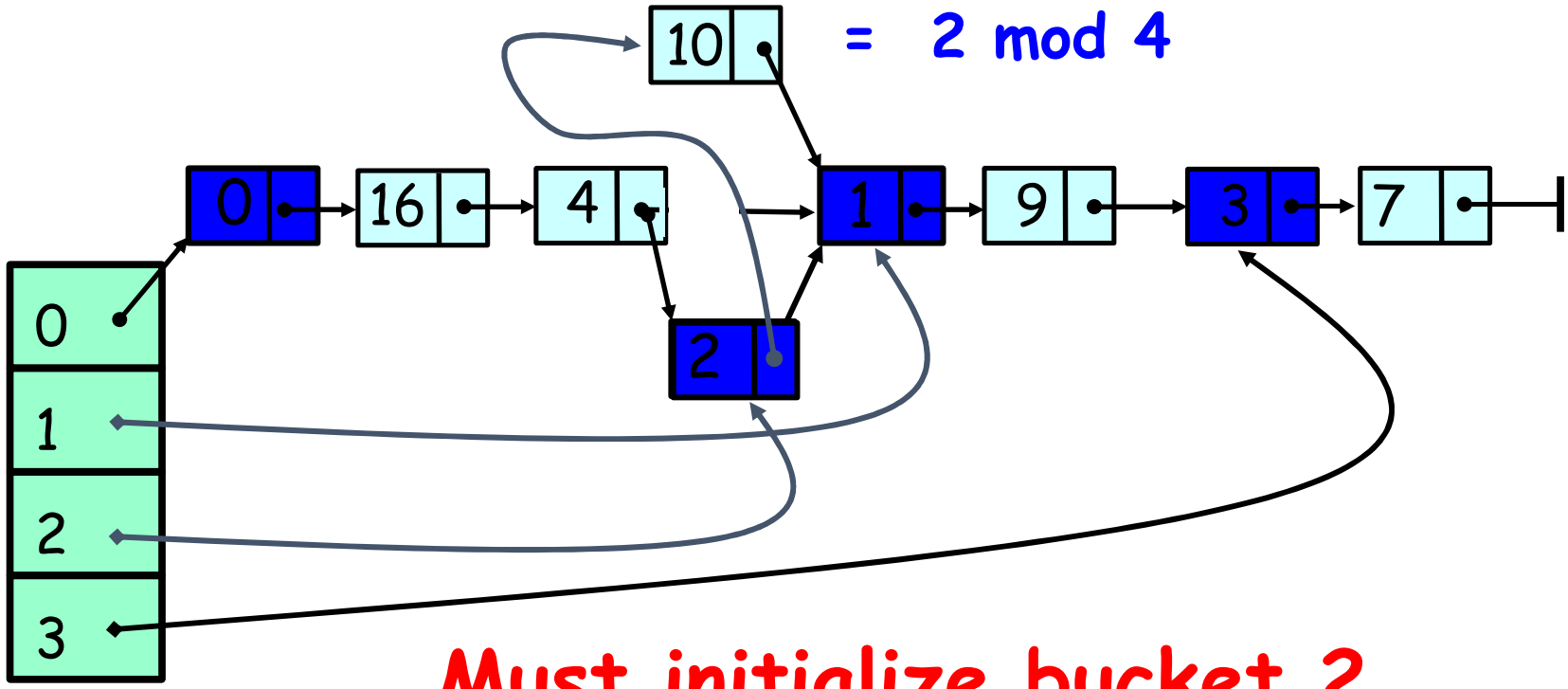


Initialization of Buckets



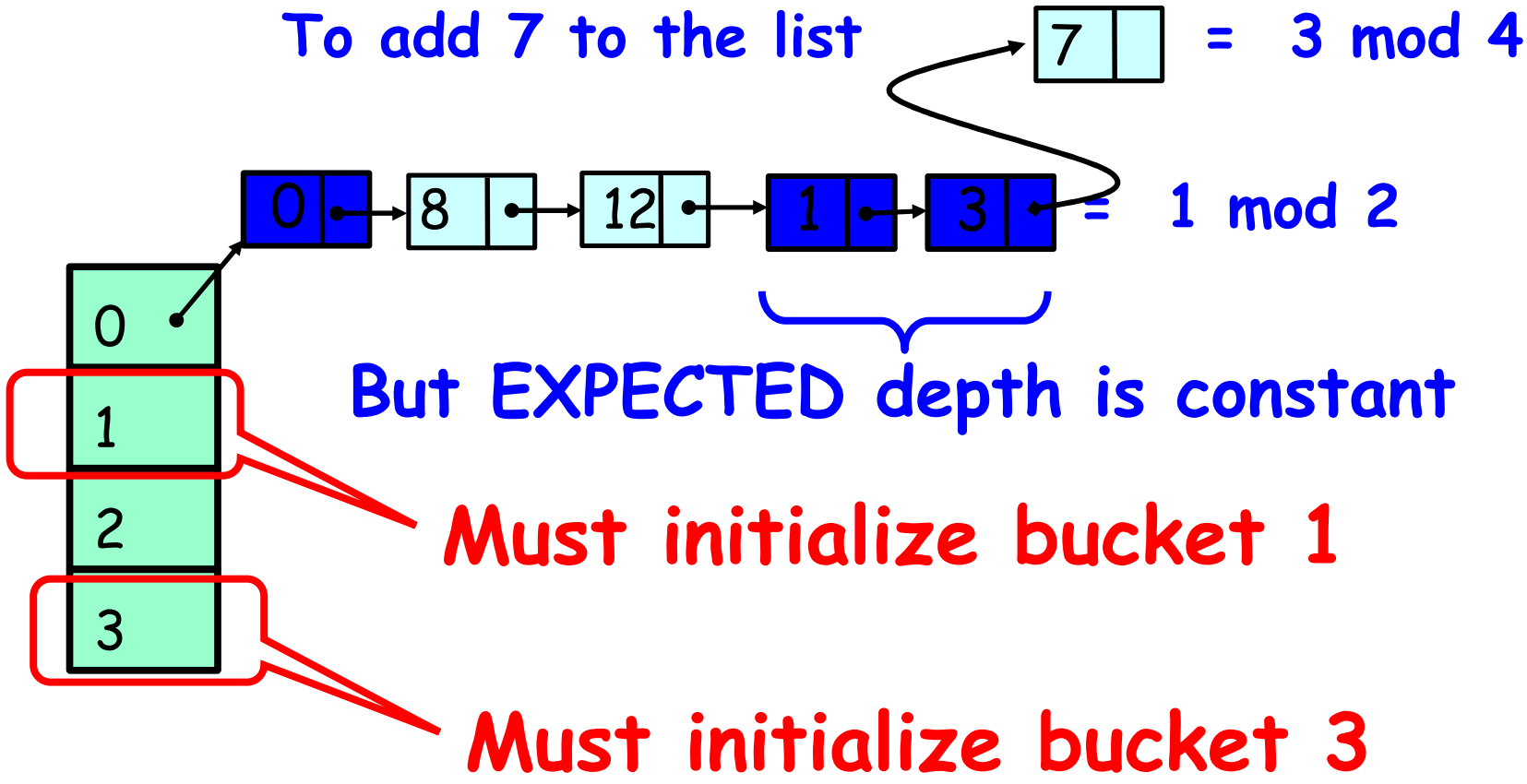
Need to initialize bucket 3 to split bucket 1

Adding 10



Must initialize bucket 2
Then can add 10

Recursive Initialization



Main List

- Lock-Free List from earlier class
- With some minor variations

Summary

- Concurrent resizing is tricky
- Lock-based
  - Fine-grained
  - Read/write locks
  - Optimistic
- Lock-free
  - Builds on lock-free list