Concurrency in Linear Hashing

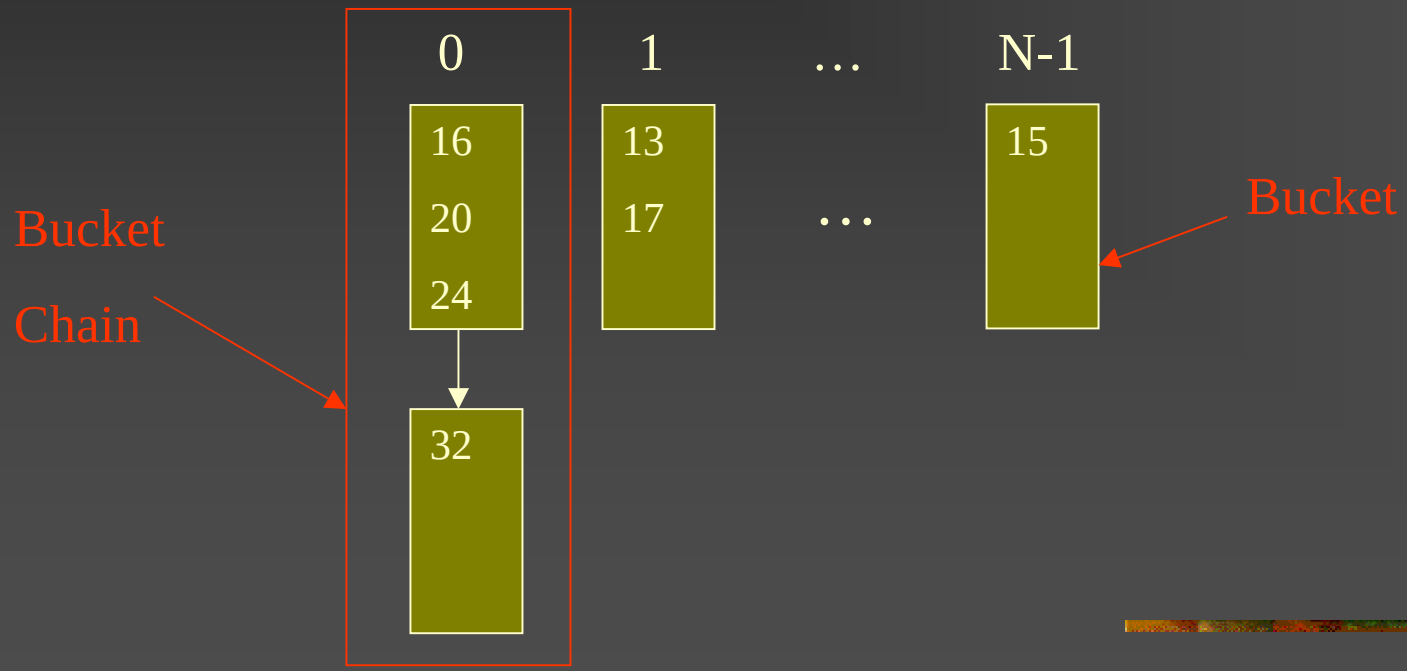
Huxia Shi

York University

Apr 05, 2009

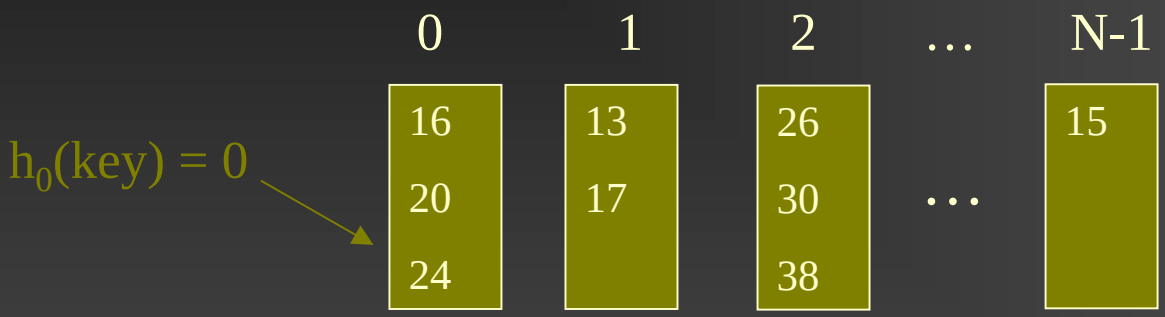
Linear Hashing

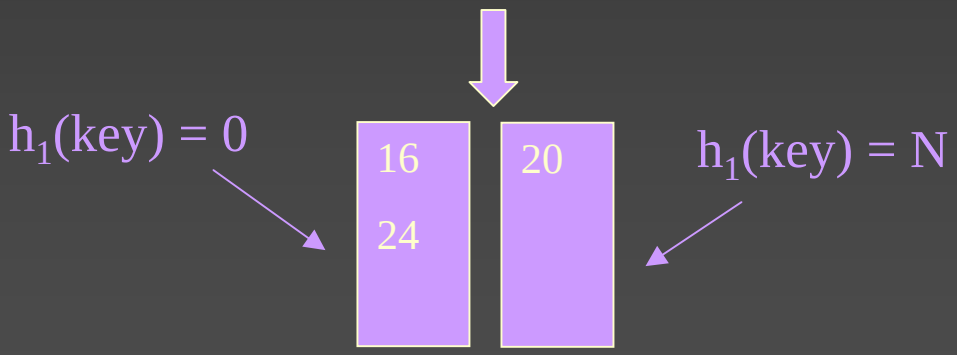
- A technique for dynamic hashing
- Invented by Witold Litwin,1980
- It can be used to save database indices



Linear Hashing

- Split



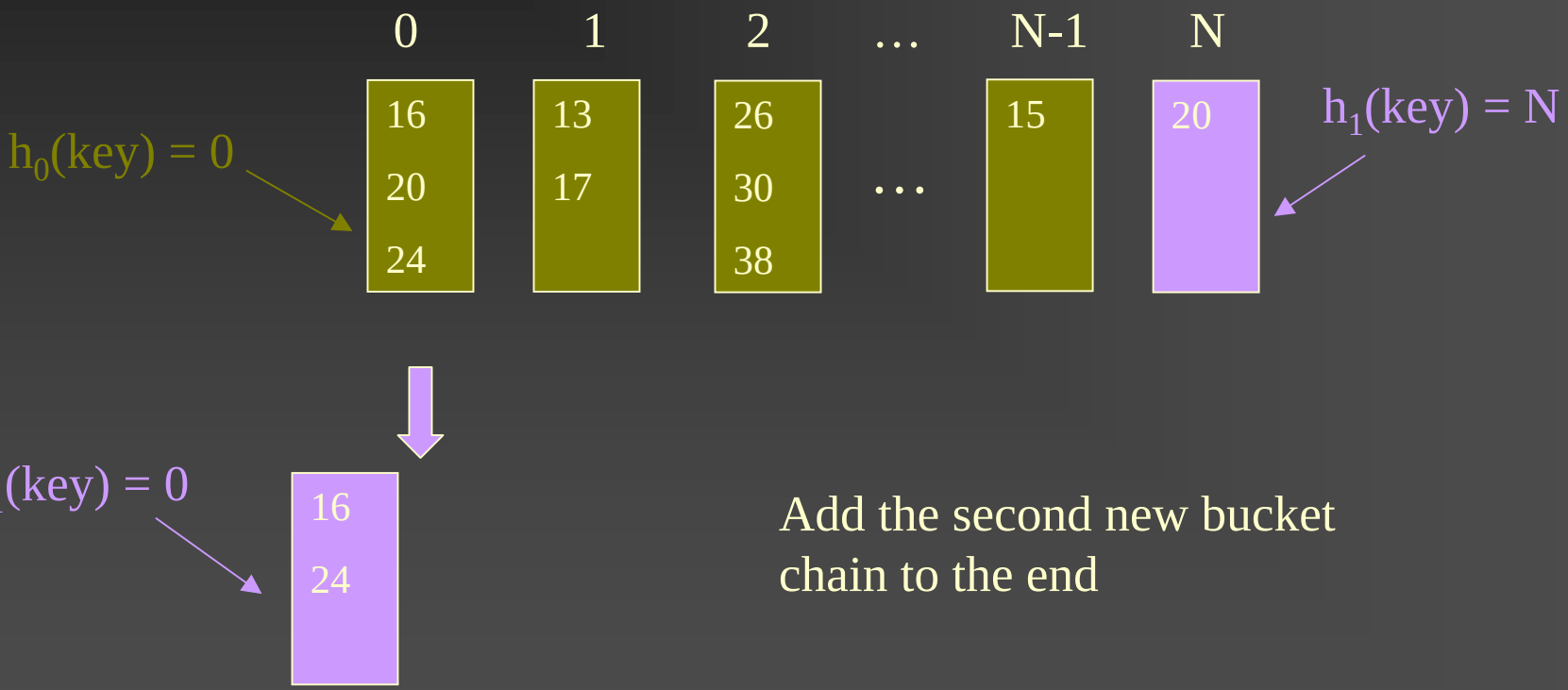


Choose new hash function h_1

Split the first bucket chain

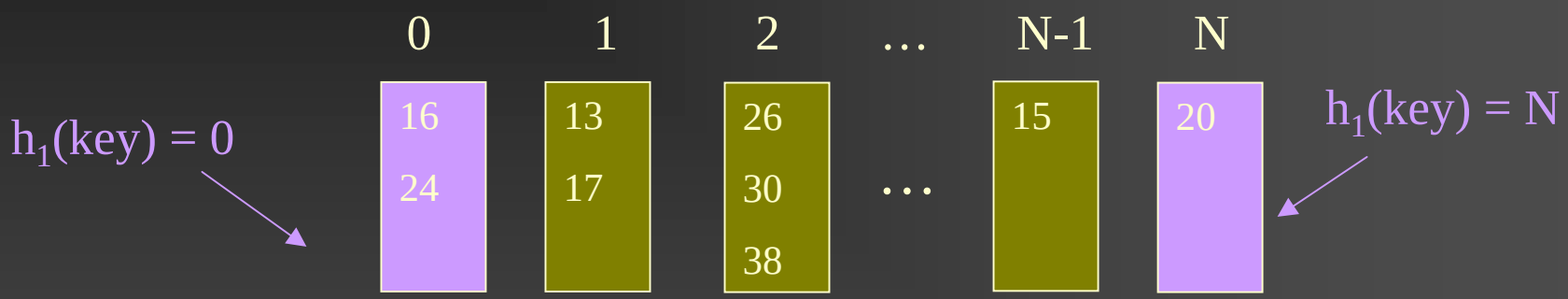
Linear Hashing

- Split



Linear Hashing

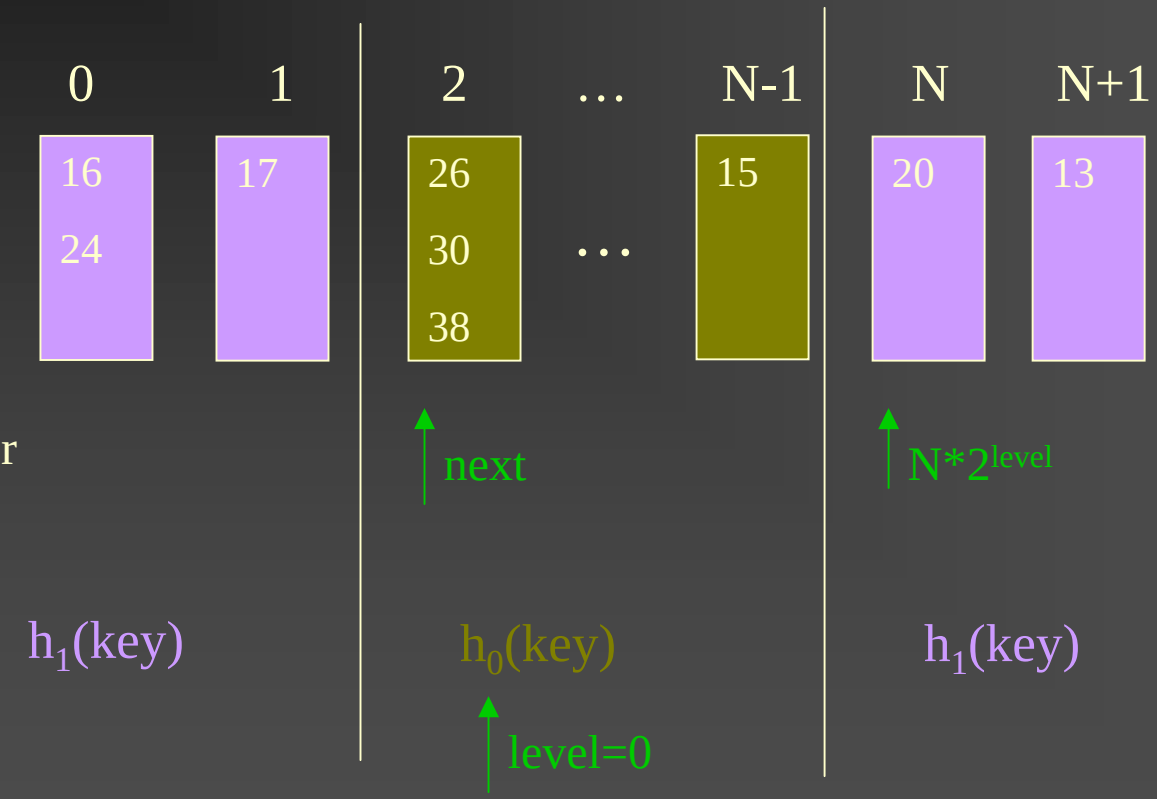
- Split



Replace the bucket chain to be splitted with the first new bucket chain

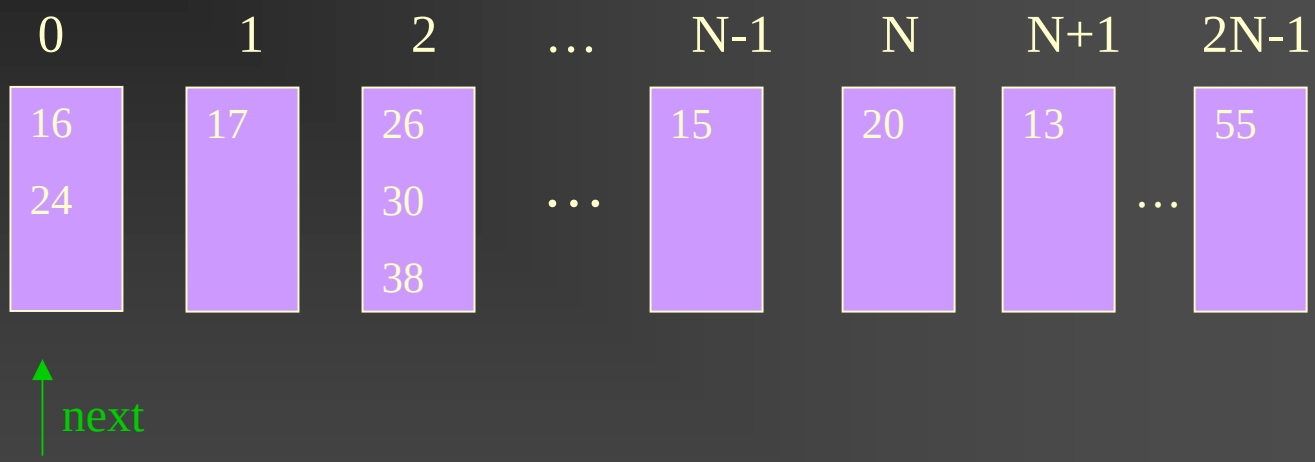
Linear Hashing

- Split



Linear Hashing

- Split



When next pointer moves to the upper bondary, it is set to 0

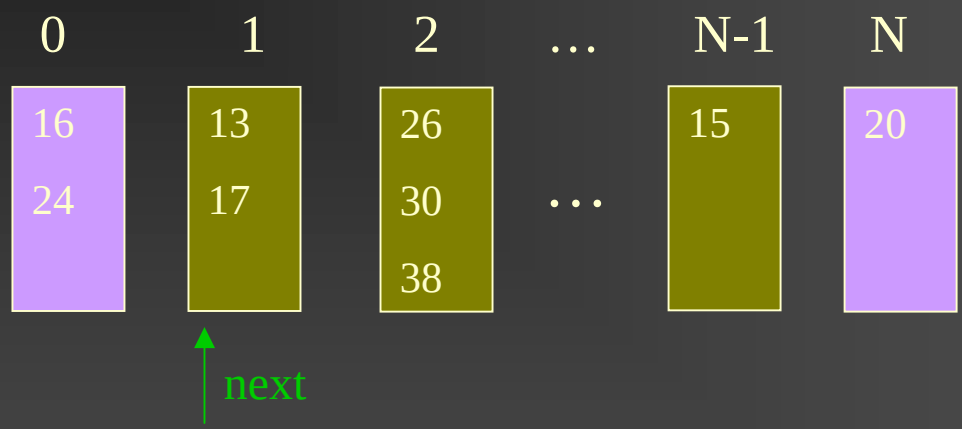
Then level value is increased

$h_1(key)$

level=1

Linear Hashing

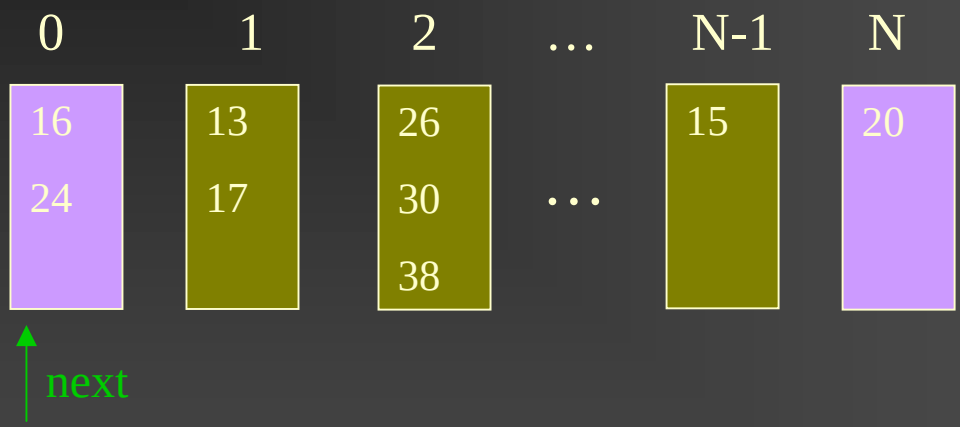
- Merge



Merge is opposite to Split

Linear Hashing

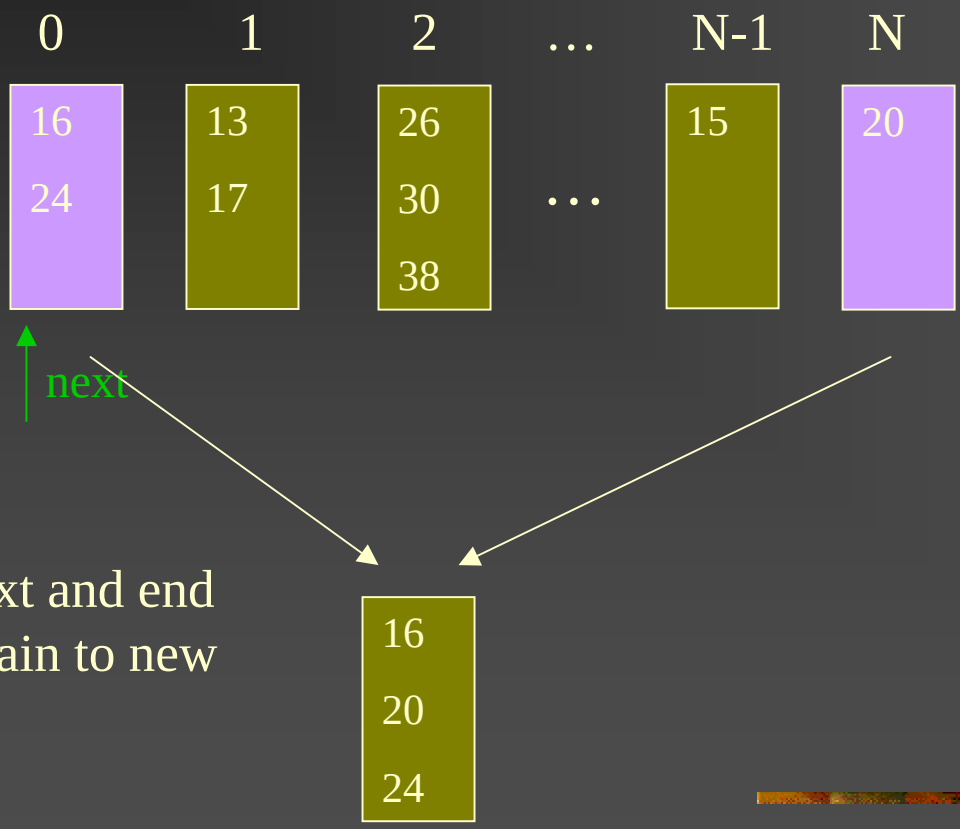
- Merge



Next pointer moves backwards

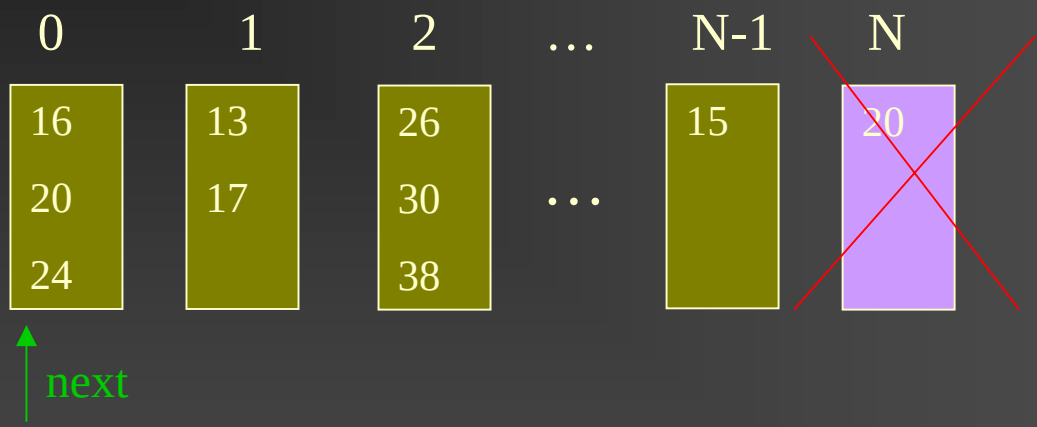
Linear Hashing

- Merge



Linear Hashing

- Merge



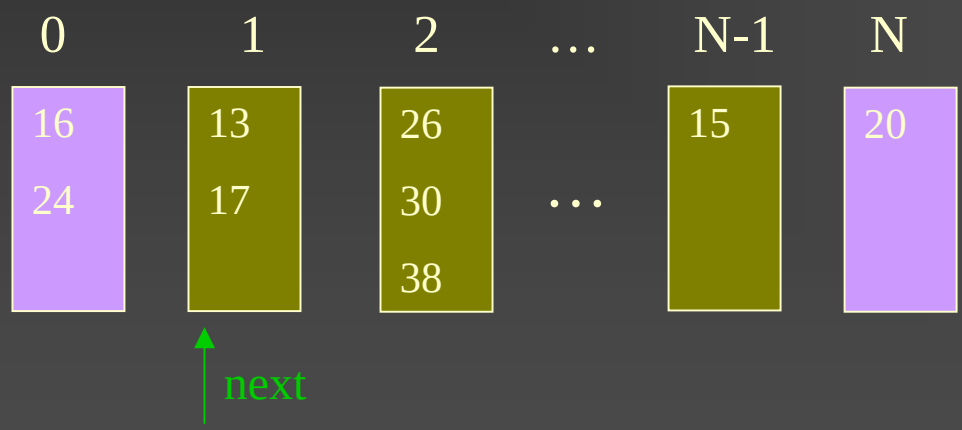
Replace the bucket chain pointed by next

Delete the last bucket chain

Current Operations

Process1 (find 13): hash(13) = 1

Process2 (split): split bucket chain 1



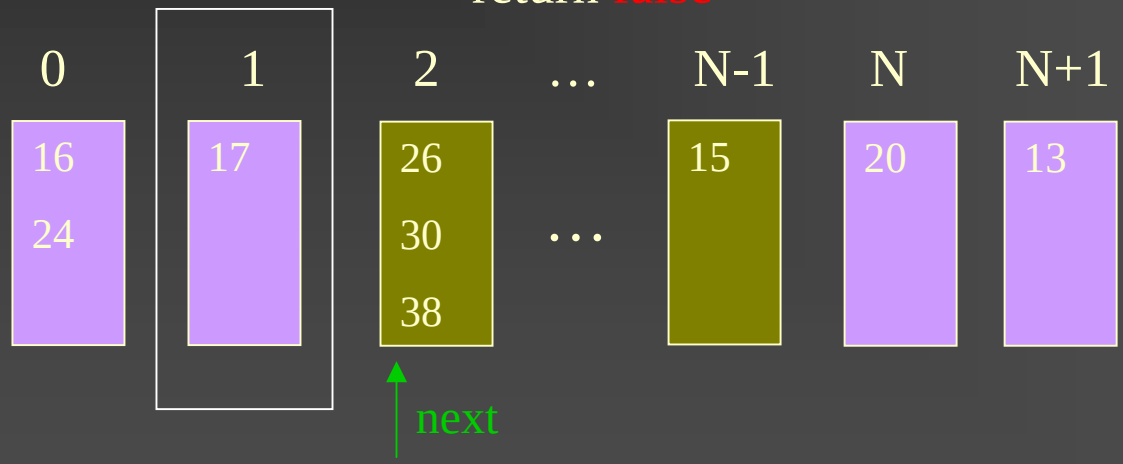
Current Operations

Process1 (find 13): hash(13) = 1

Process2 (split): split bucket chain 1

Process1 (find 13): check bucket chain 1

return false



My Paper

- Carla Schlatter Ellis. Concurrency in linear hashing. *ACM Transactions on Database Systems (TODS)*, 12(2): 195-217, June 1987.

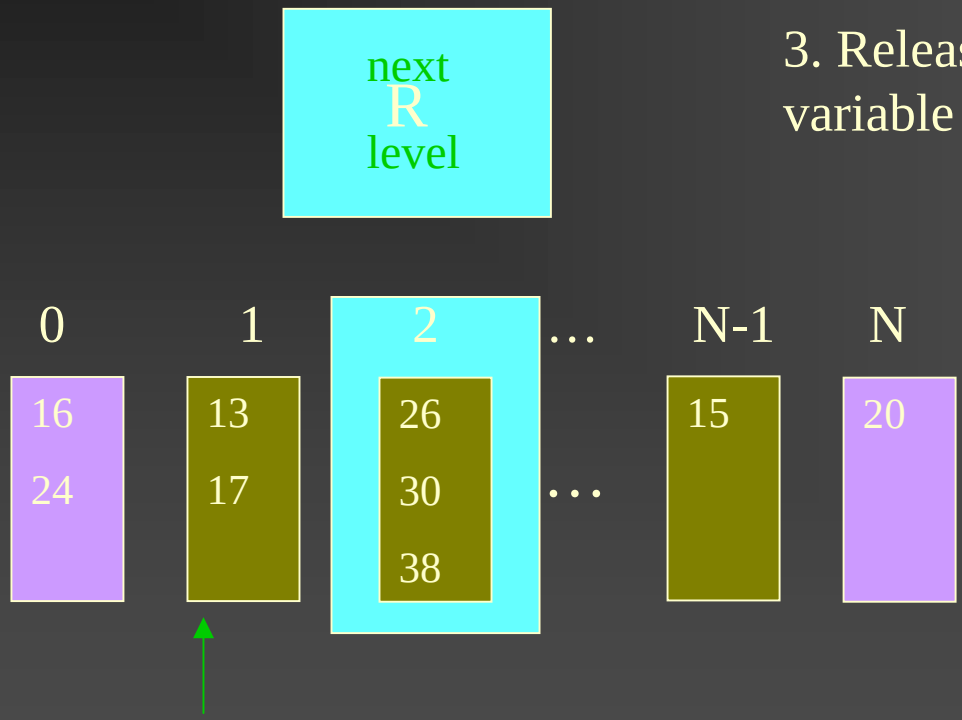
Main Idea

	Existing lock		
Lock Request	Read lock	Selective lock	Exclusive lock
Read lock	yes	yes	no
Selective lock	yes	no	no
Exclusive lock	no	no	no

R S E

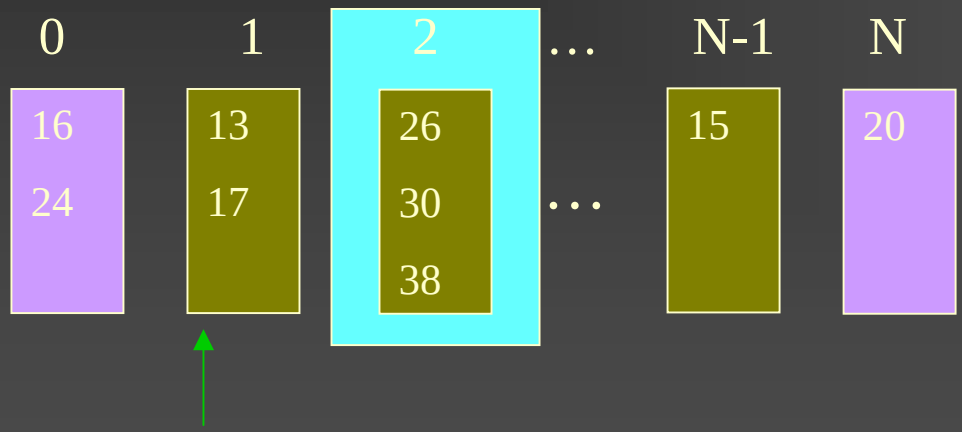
Find Operation

1. Put read lock on root variable: next and level
2. Put read lock on target bucket chain
3. Release lock on root variable



Find Operation

1. Put read lock on root variable: next and level
2. Put read lock on target bucket chain
3. Release lock on root variable
4. Read data in the locked bucket chain



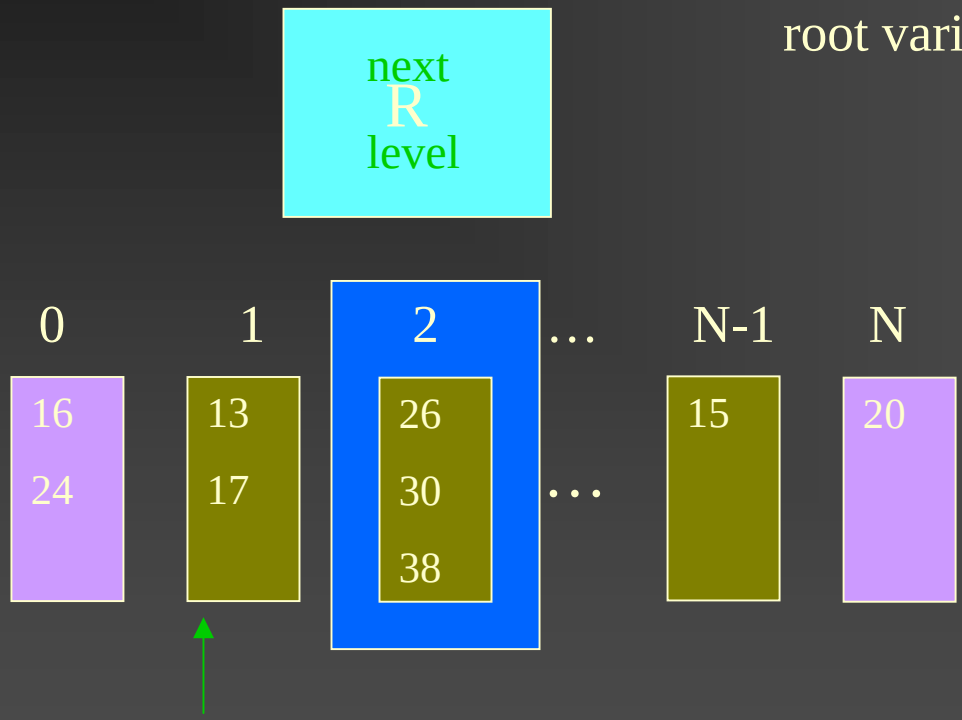
Find Operation

1. Put read lock on root variable: next and level
2. Put read lock on target bucket chain
3. Release lock on root variable
4. Read data in the locked bucket chain

5. Release read lock on bucket chain

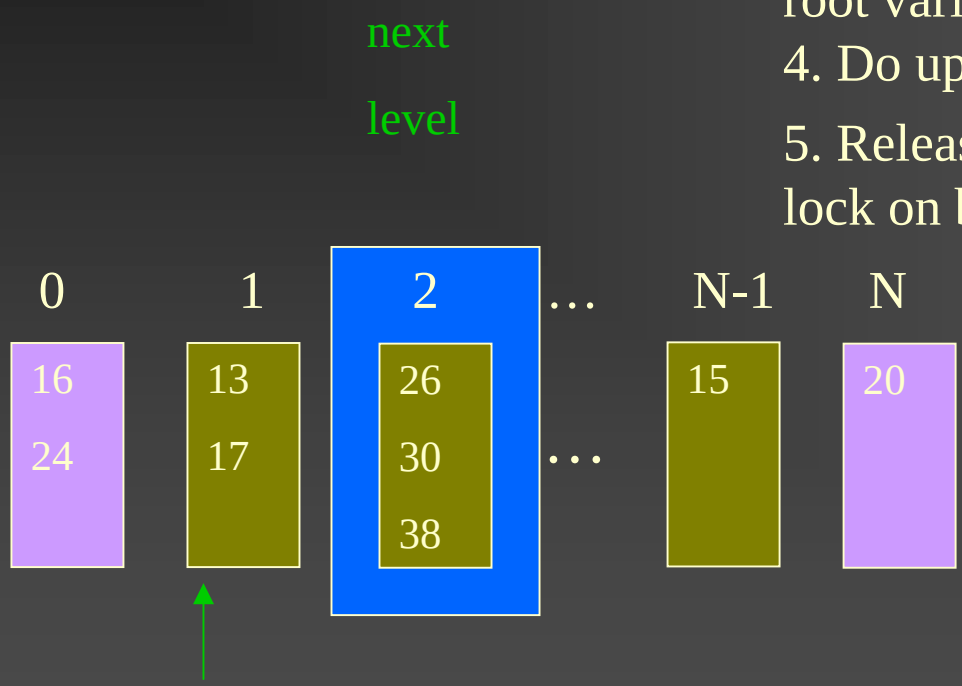
Insert and Delete Operation

1. Put read lock on root variable: next and level
2. Put selective lock on target bucket chain
3. Release read lock on root variable



Insert and Delete Operation

1. Put read lock on root variable: next and level
2. Put selective lock on target bucket chain
3. Release read lock on root variable
4. Do update
5. Release selective lock on bucket chain



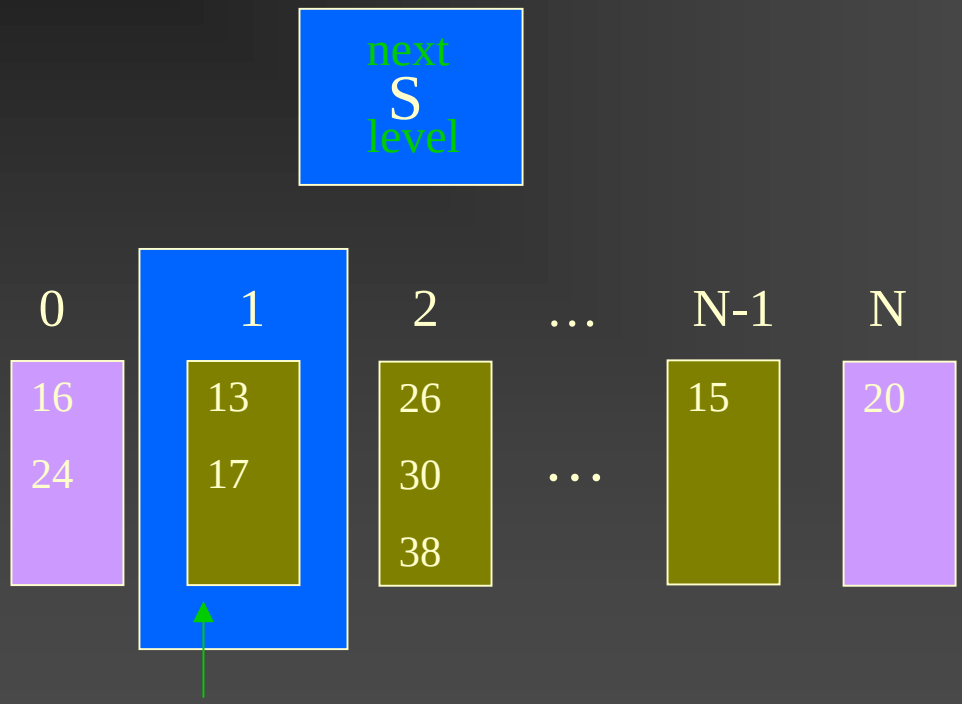
Insert and Delete Operation

1. Put read lock on root variable: next and level
2. Put selective lock on target bucket chain
3. Release read lock on root variable
4. Do update
5. Release selective lock on bucket chain

6. Invoke Split or Merge if required

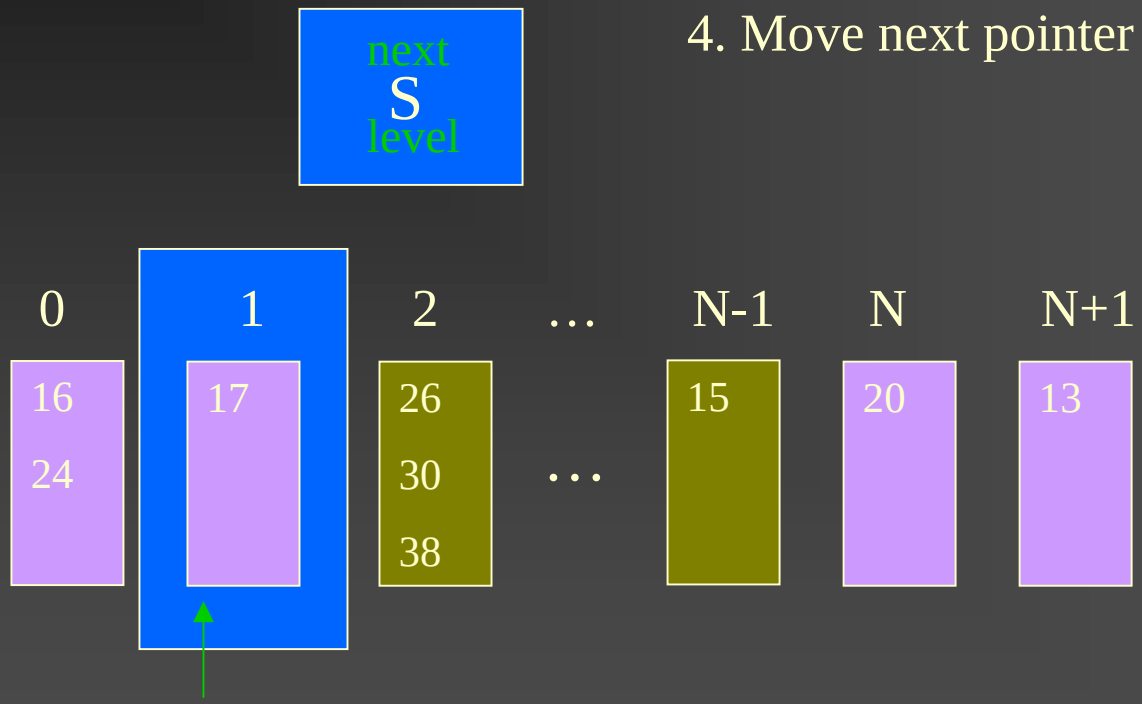
Split Operation

1. Put selective lock on root variable: next and level
2. Put selective lock on next bucket chain
3. Split next bucket chain



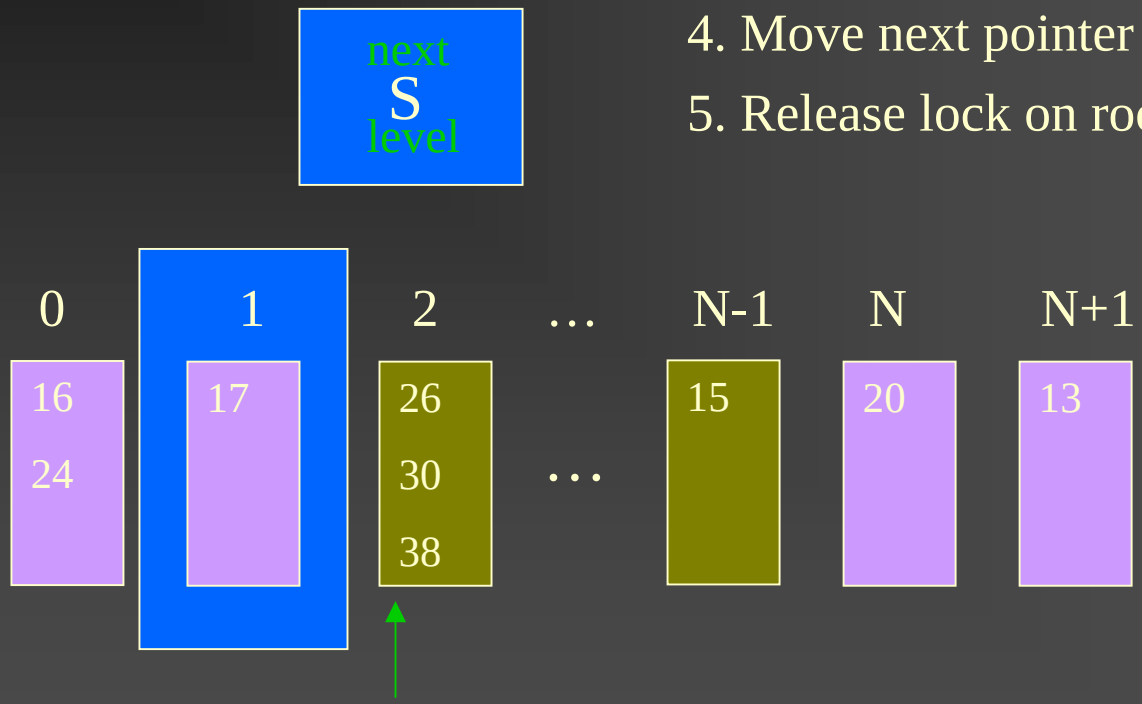
Split Operation

1. Put selective lock on root variable: next and level
2. Put selective lock on next bucket chain
3. Split next bucket chain
4. Move next pointer forwards



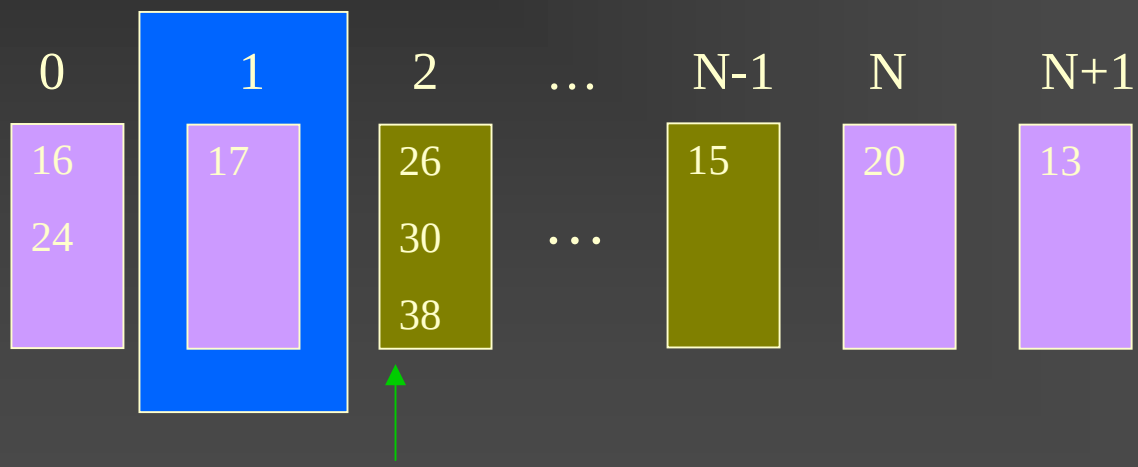
Split Operation

1. Put selective lock on root variable: next and level
2. Put selective lock on next bucket chain
3. Split next bucket chain
4. Move next pointer forwards
5. Release lock on root variable



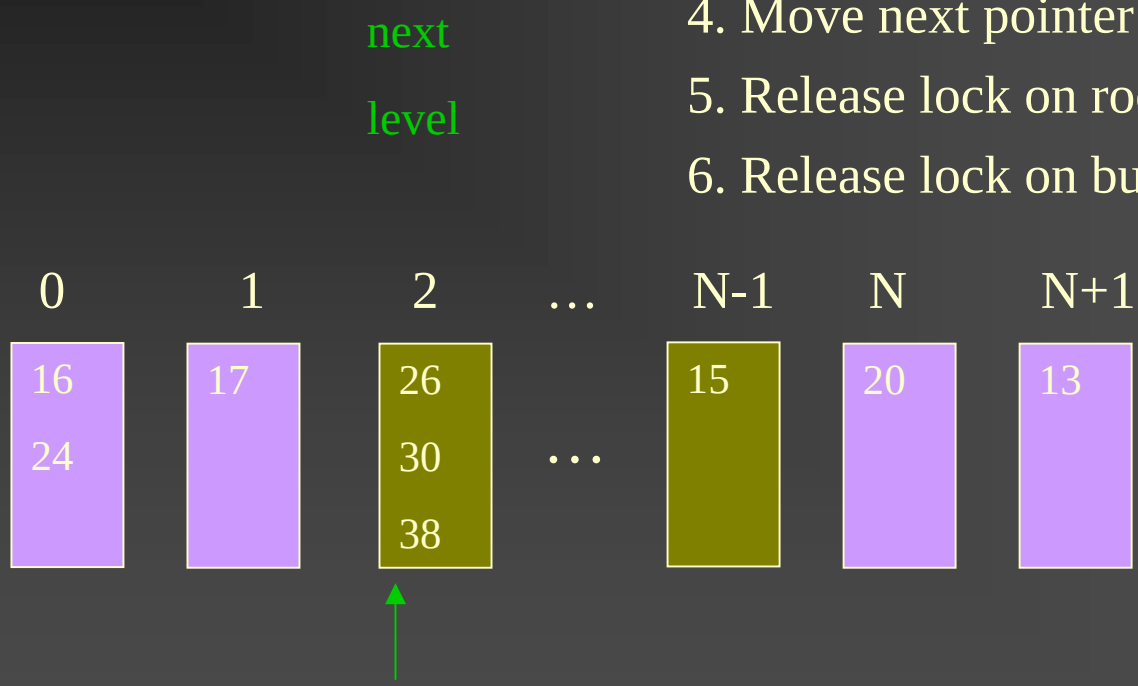
Split Operation

1. Put selective lock on root variable: next and level
2. Put selective lock on next bucket chain
3. Split next bucket chain
4. Move next pointer forwards
5. Release lock on root variable
6. Release lock on bucket chain



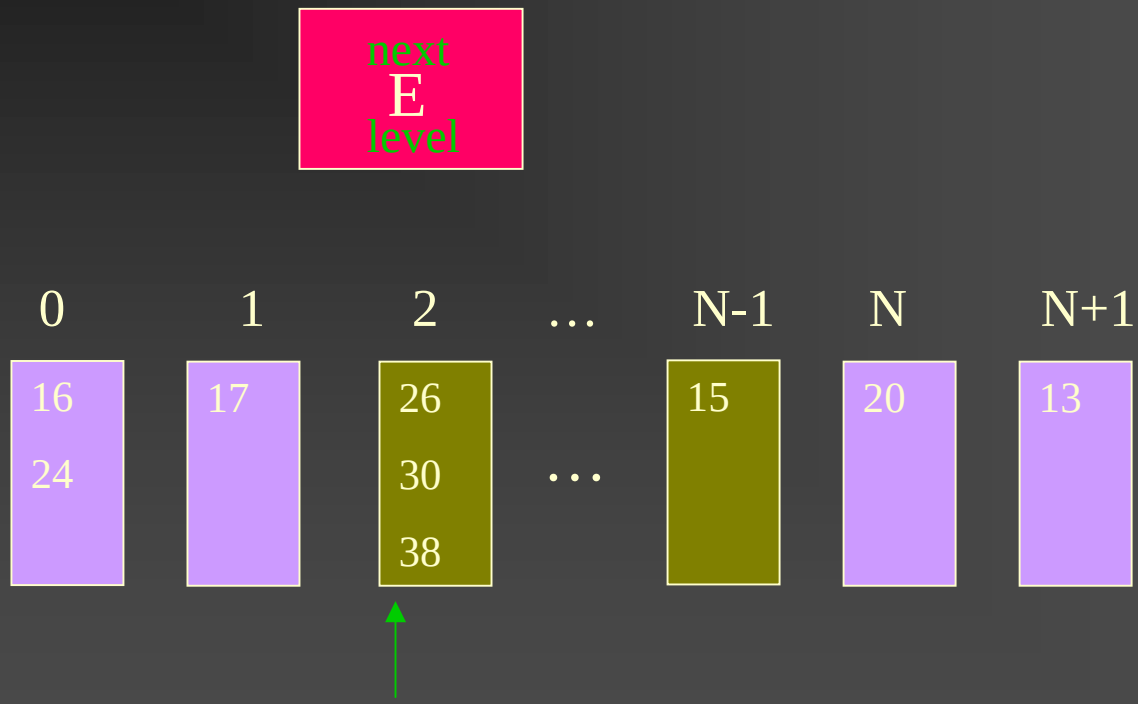
Split Operation

1. Put selective lock on root variable: next and level
2. Put selective lock on next bucket chain
3. Split next bucket chain
4. Move next pointer forwards
5. Release lock on root variable
6. Release lock on bucket chain



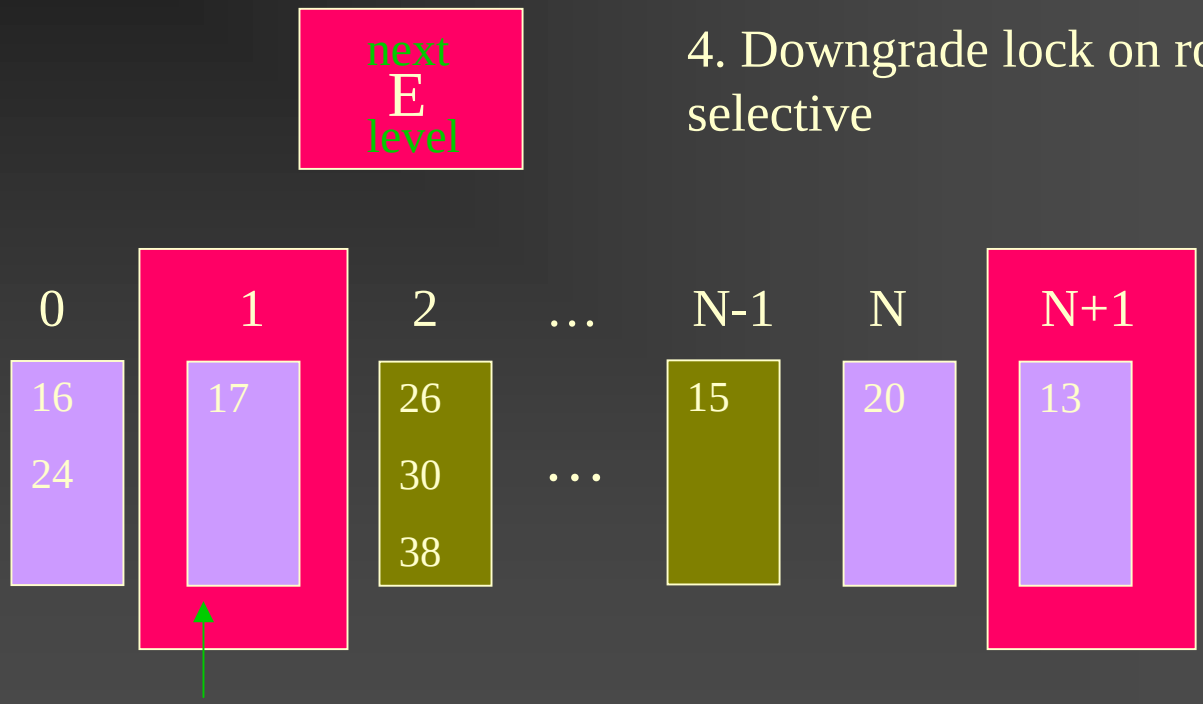
Merge Operation

1. Put exclusive lock on root variable: next and level
2. Move next pointer backwords



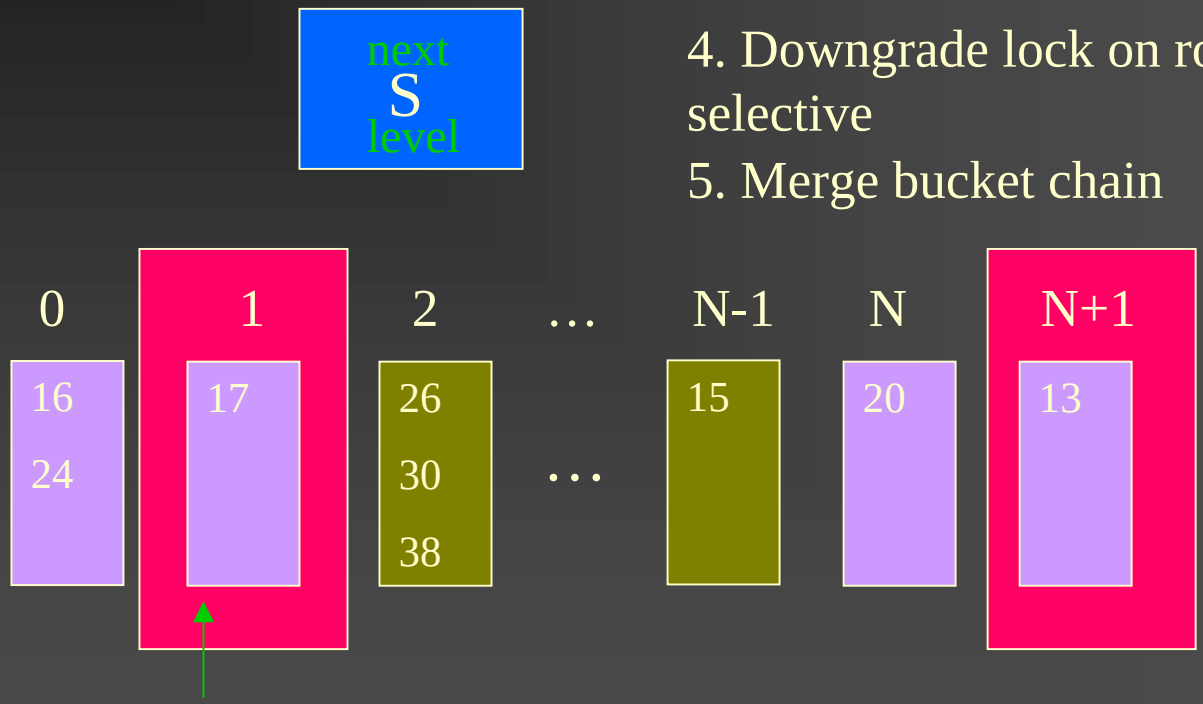
Merge Operation

1. Put exclusive lock on root variable: next and level
2. Move next pointer backwords
3. Put exclusive lock on next and end bucket chain
4. Downgrade lock on root to selective



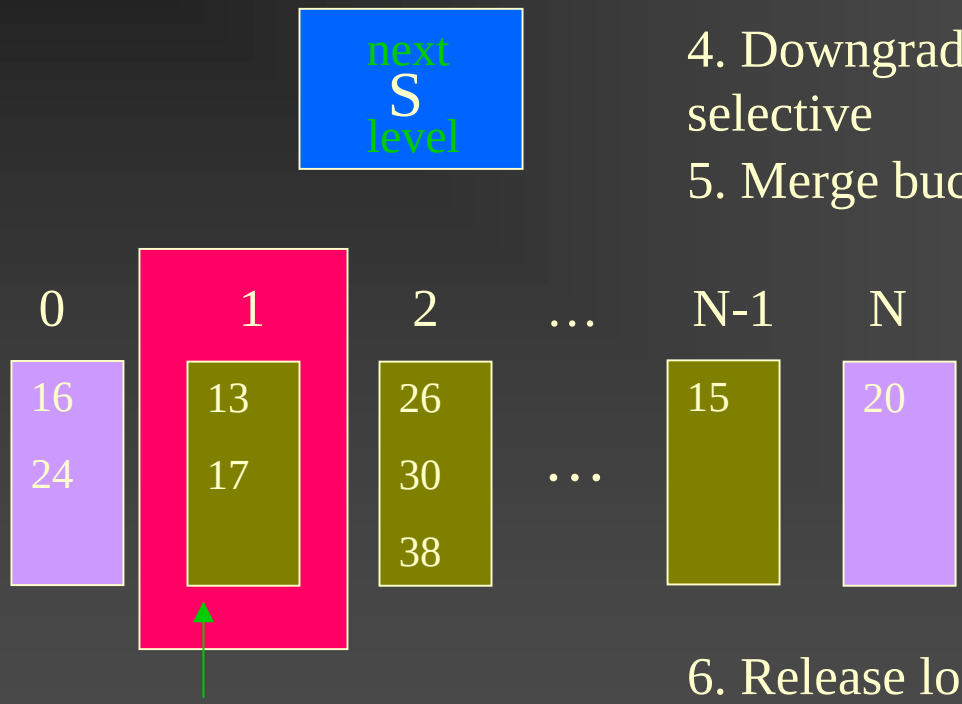
Merge Operation

1. Put exclusive lock on root variable: next and level
2. Move next pointer backwords
3. Put exclusive lock on next and end bucket chain
4. Downgrade lock on root to selective
5. Merge bucket chain



Merge Operation

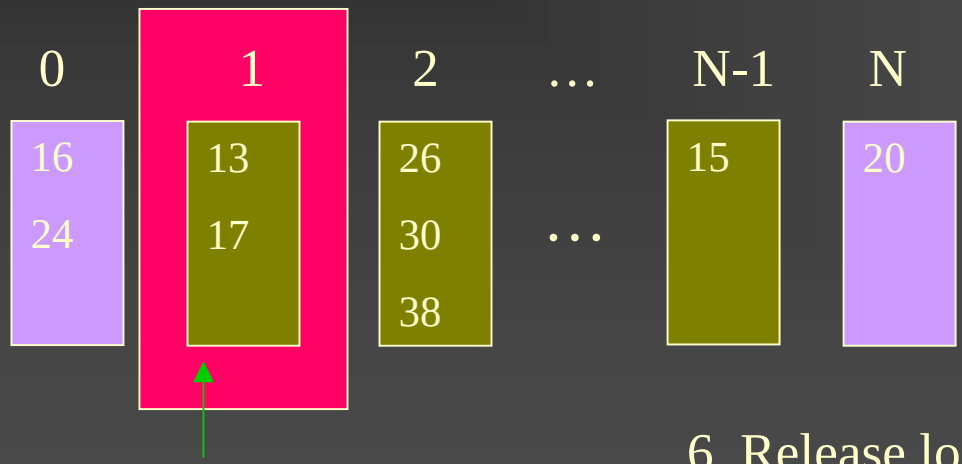
1. Put exclusive lock on root variable: next and level
2. Move next pointer backwords
3. Put exclusive lock on next and end bucket chain
4. Downgrade lock on root to selective
5. Merge bucket chain



6. Release lock on root variable

Merge Operation

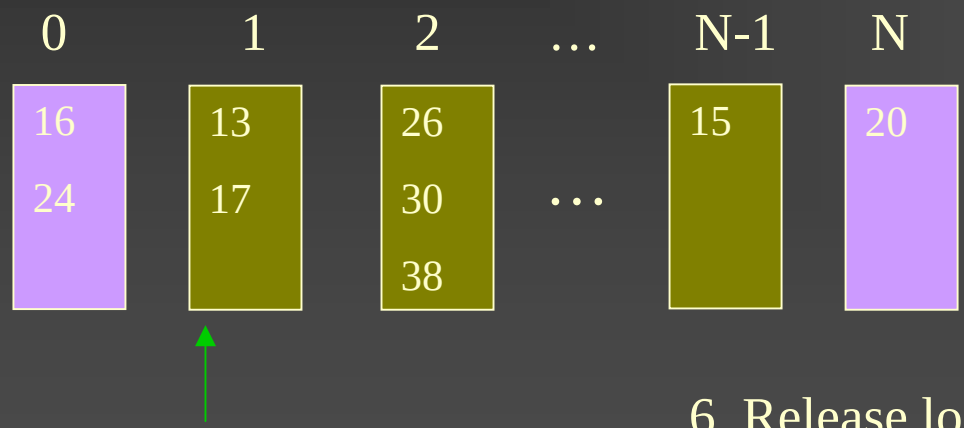
1. Put exclusive lock on root variable: next and level
2. Move next pointer backwords
3. Put exclusive lock on next and end bucket chain
4. Downgrade lock on root to selective
5. Merge bucket chain



6. Release lock on root variable
7. Release lock on bucket chain

Merge Operation

1. Put exclusive lock on root variable: next and level
2. Move next pointer backwords
3. Put exclusive lock on next and end bucket chain
4. Downgrade lock on root to selective
5. Merge bucket chain



6. Release lock on root variable
7. Release lock on bucket chain

Other problems

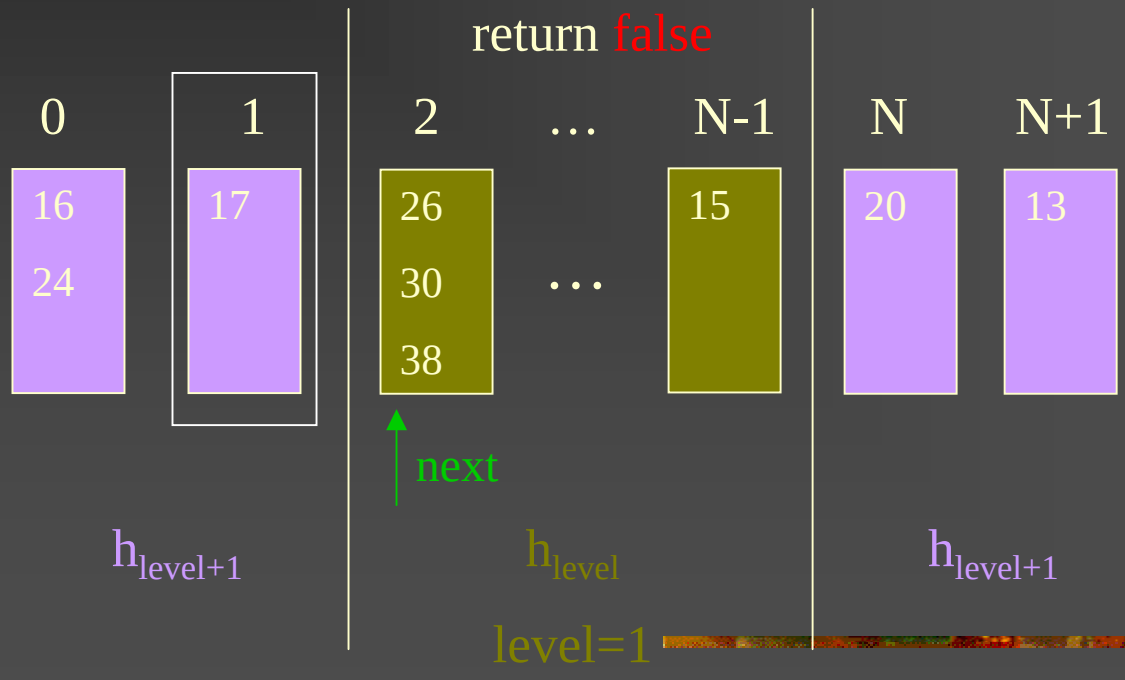
Process1 (find 13): $h_1(13) = 1$ (level =1)

Process2 (split): split bucket chain 1

Process1 (find 13): check bucket chain 1

Problem:

find and split can run simutaneously



Solution

Solution:

Add variable local level ll in bucket chain

Find process:

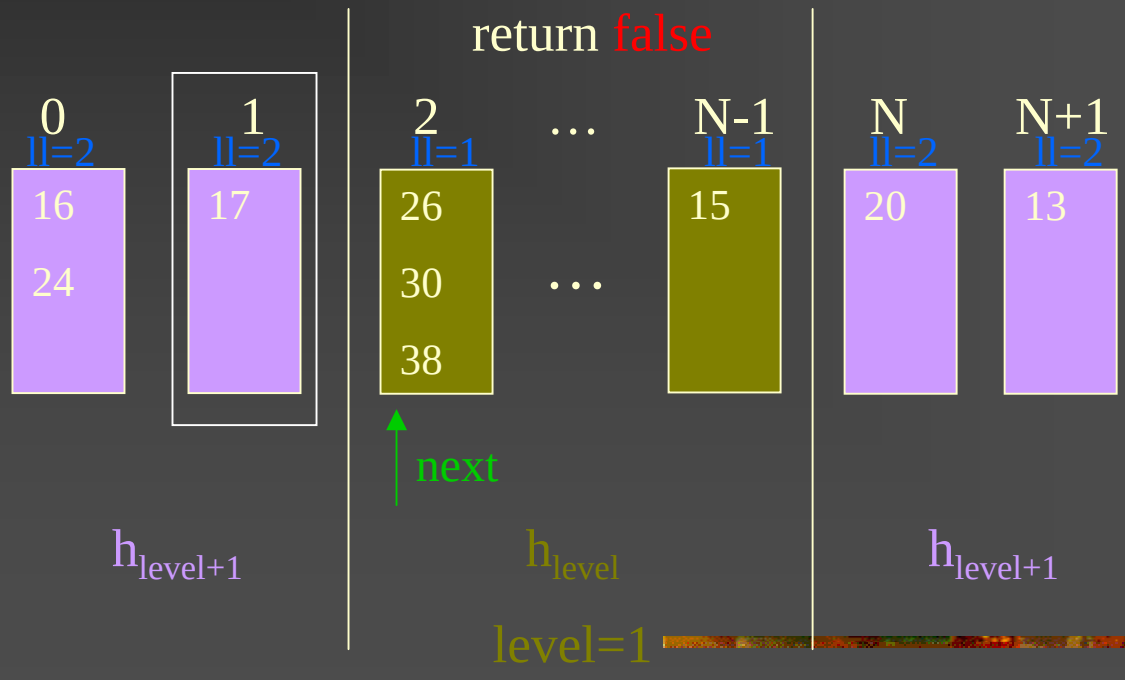
Compare ll with read level value

Try higher level if not matched

Process1 (find 13): $h_1(13) = 1$ (level =1)

Process2 (split): split bucket chain 1

Process1 (find 13): check bucket chain 1



Freedom from Deadlock

- Lock-coupling protocols
  - add lock on first element
  - add lock on next elment
  - release lock from first element
  - release lock from next element
- Locks are requested according to an ordering
  - root < any bucket chain
  - bucket chain (i) < bucket chain (j) if i < j

The End

Thanks

Q&A