CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 – DB Applications

Faloutsos & Pavlo
Lecture#11 (R&G ch. 11)
Hashing

CMU SCS

Outline

- (static) hashing
- extendible hashing
- linear hashing
- Hashing vs B-trees

CMU SCS

(Static) Hashing

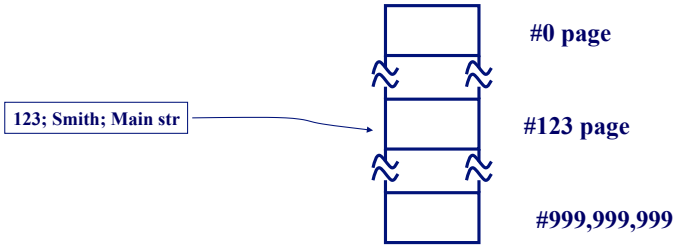
Problem: "*find EMP record with ssn=123*"

What if disk space was free, and time was at premium?

CMU SCS

Hashing

A: Brilliant idea: key-to-address transformation:

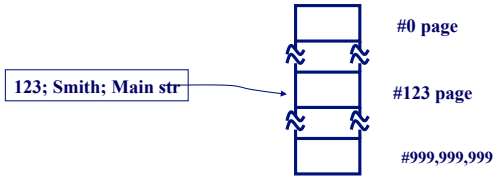


CMU SCS

Hashing

Since space is NOT free:

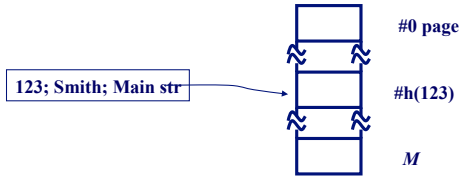
- use *M*, instead of 999,999,999 slots
- hash function: *h(key) = slot-id*



CMU SCS

Hashing

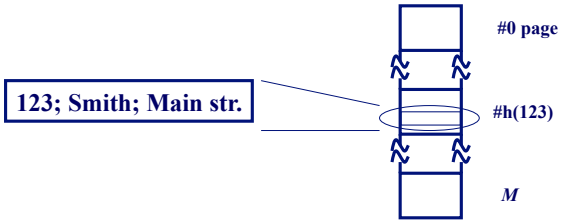
Typically: each hash bucket is a page, holding many records:



CMU SCS

Hashing

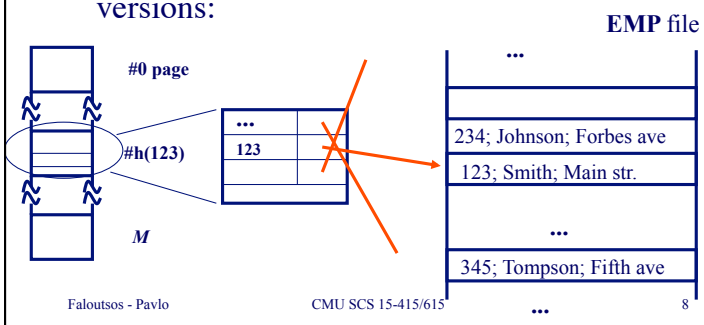
Notice: could have **clustering**, or non-clustering versions:



CMU SCS

Hashing

Notice: could have clustering, or **non-clustering** versions:



CMU SCS

Indexing- overview

- hashing
  - hashing functions
  - size of hash table
  - collision resolution
- extendible hashing
- Hashing vs B-trees

CMU SCS

Design decisions

1) formula *h()* for hashing function
2) size of hash table *M*
3) collision resolution method

CMU SCS

Design decisions - functions

- Goal: uniform spread of keys over hash buckets
- Popular choices:
  - Division hashing
  - Multiplication hashing

CMU SCS

Division hashing

$$h(x) = (a*x+b) \mod M$$

- eg., h(ssn) = (ssn) mod 1,000
  - gives the last three digits of ssn
- M: size of hash table - choose a prime number, defensively (why?)

CMU SCS

Division hashing

- eg., *M*=2; hash on driver-license number (dln), where last digit is 'gender' (0/1 = M/F)
- in an army unit with predominantly male soldiers
- Thus: avoid cases where *M* and keys have common divisors - prime *M* guards against that!

CMU SCS

Multiplication hashing

h(x) = [fractional-part-of (x * φ)] * M

- ***φ***: golden ratio (0.618... = (sqrt(5)-1)/2)
- in general, we need an irrational number
- advantage: *M* need not be a prime number
- but ***φ*** must be irrational

CMU SCS

Other hashing functions

- quadratic hashing (bad)
- ...

CMU SCS

Other hashing functions

- quadratic hashing (bad)
- ...
- conclusion: use division hashing

CMU SCS

Design decisions

1) formula *h()* for hashing function
2) size of hash table *M*
3) collision resolution method

CMU SCS

Size of hash table

- eg., 50,000 employees, 10 employee-records / page
- Q: *M*=?? pages/buckets/slots

CMU SCS

Size of hash table

- eg., 50,000 employees, 10 employees/page
- Q: *M*=?? pages/buckets/slots
- A: utilization ~ 90% and
  - *M*: prime number

Eg., in our case: *M*= closest prime to 50,000/10 / 0.9 = 5,555

CMU SCS

Design decisions

1) formula *h()* for hashing function
2) size of hash table *M*
3) collision resolution method

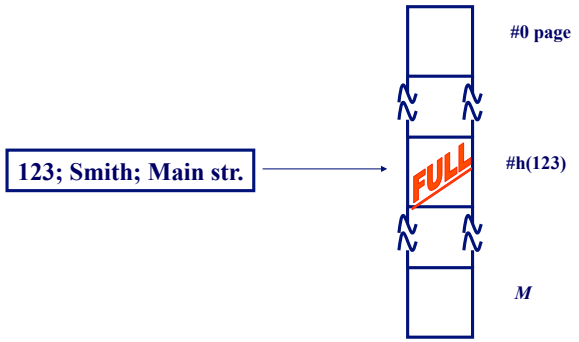
CMU SCS

Collision resolution

- Q: what is a 'collision'?
- A: ??

CMU SCS

Collision resolution



CMU SCS

Collision resolution

- Q: what is a 'collision'?
- A: ??
- Q: why worry about collisions/overflows? (recall that buckets are ~90% full)
- A: 'birthday paradox'

CMU SCS

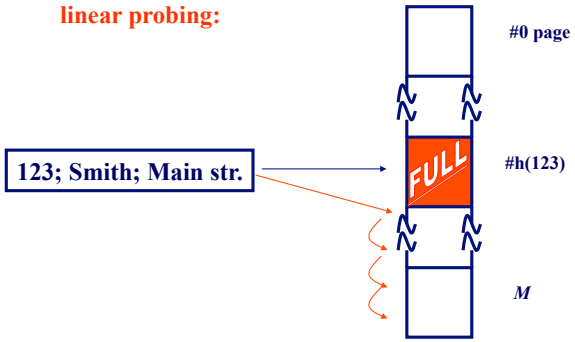
Collision resolution

- open addressing
  - linear probing (ie., put to next slot/bucket)
  - re-hashing
- separate chaining (ie., put links to overflow pages)

CMU SCS

Collision resolution

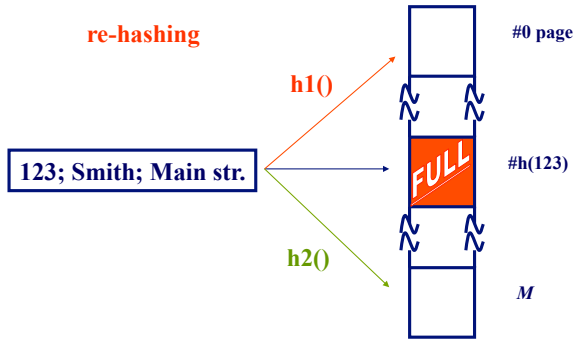
linear probing:



CMU SCS

Collision resolution

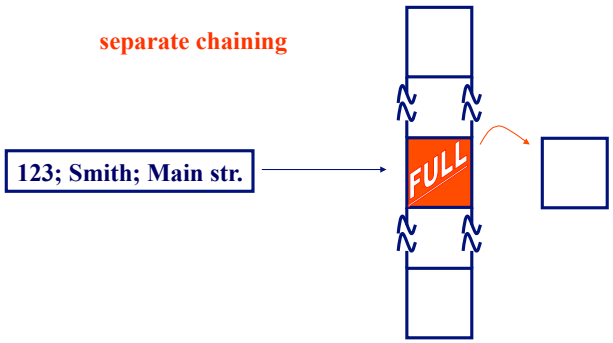
re-hashing



CMU SCS

Collision resolution

separate chaining



CMU SCS

Design decisions - conclusions

- function: division hashing
  - *h(x) = (a*x+b) mod M*
- size *M*: ~90% util.; prime number.
- collision resolution: separate chaining
  - easier to implement (deletions!);
  - no danger of becoming full

CMU SCS

Outline

- (static) hashing
- extendible hashing
- linear hashing
- Hashing vs B-trees

CMU SCS

Problem with static hashing

- problem: overflow?
- problem: underflow? (underutilization)

CMU SCS

Solution: Dynamic/extendible hashing

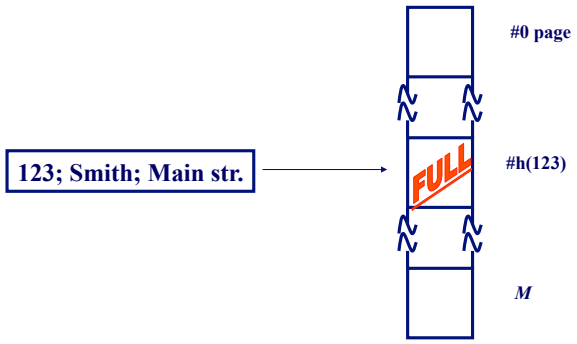
- idea: shrink / expand hash table on demand..
- ..dynamic hashing

Details: how to grow gracefully, on overflow?

Many solutions - One of them: 'extendible hashing' [Fagin et al]

CMU SCS

Extendible hashing



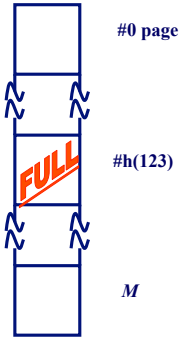
CMU SCS

Extendible hashing

solution: don't overflow – instead:

SPLIT the bucket in two

123; Smith; Main str.



CMU SCS

Extendible hashing

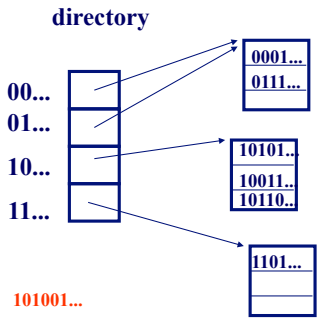
in detail:

- keep a directory, with ptrs to hash-buckets
- Q: how to divide contents of bucket in two?
- A: hash each key into a very long bit string; keep only as many bits as needed

Eventually:

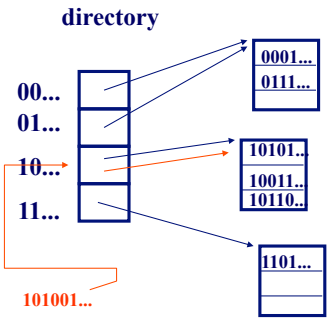
CMU SCS

Extendible hashing



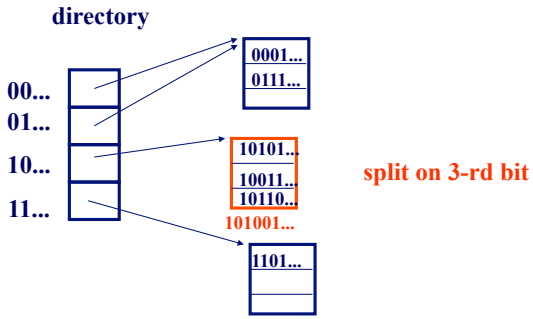
CMU SCS

Extendible hashing



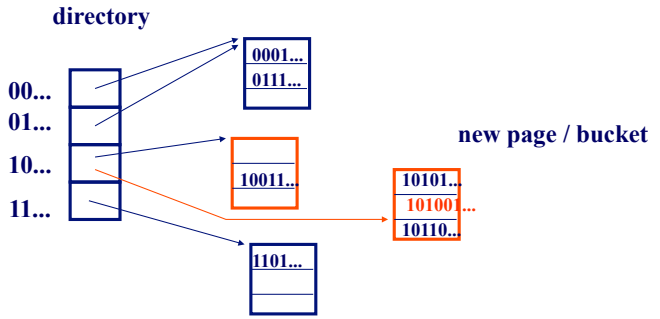
CMU SCS

Extendible hashing



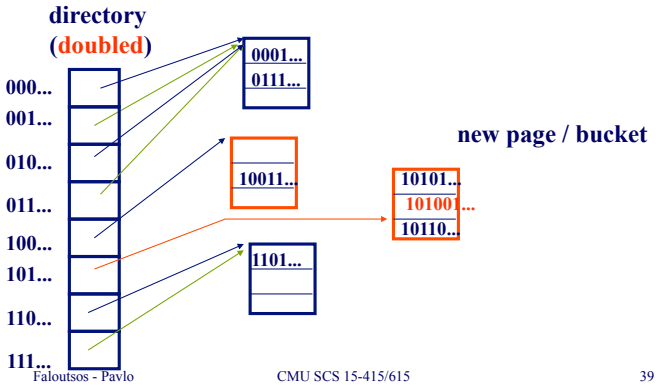
CMU SCS

Extendible hashing



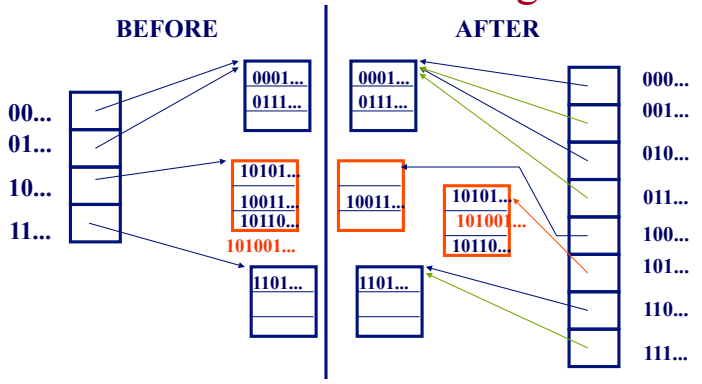
CMU SCS

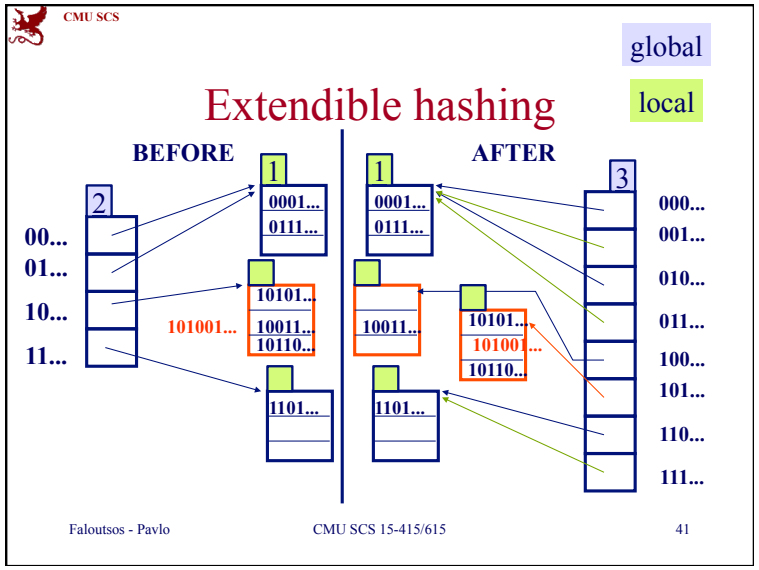
Extendible hashing

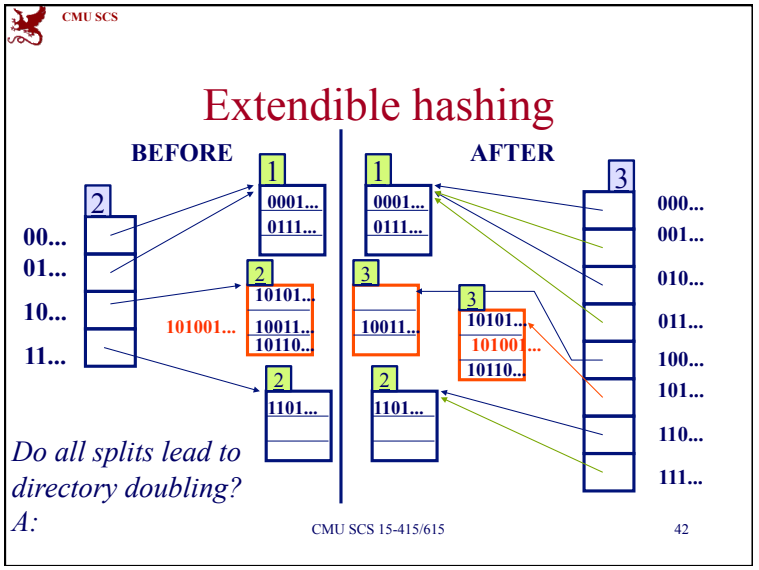


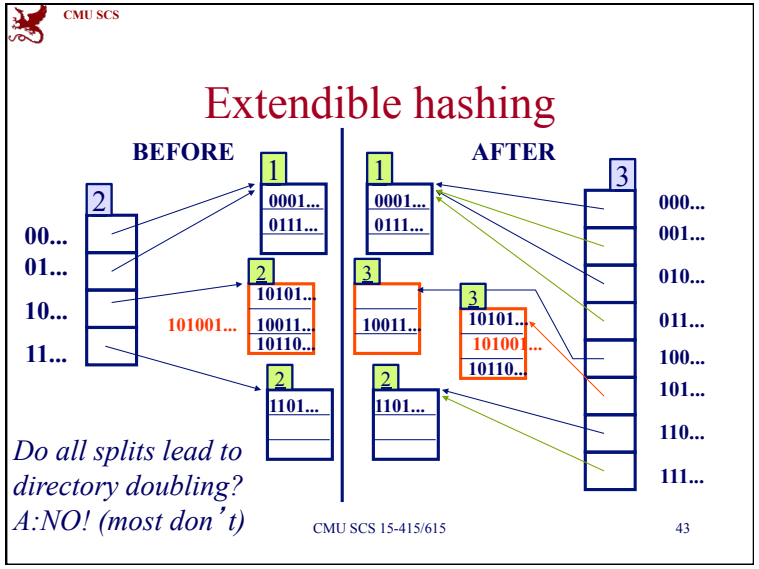
CMU SCS

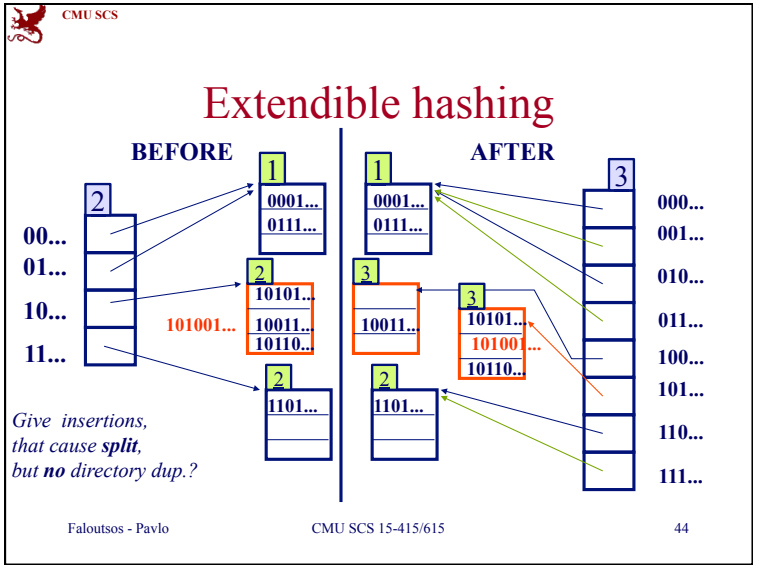
Extendible hashing




CMU SCS
global
local
Extendible hashing
BEFORE
AFTER
00...
01...
10...
11...
0001...
0111...
10101...
10011...
10110...
101001...
1101...
000...
001...
010...
011...
100...
101...
110...
111...
Faloutsos - Pavlo
CMU SCS 15-415/615
41

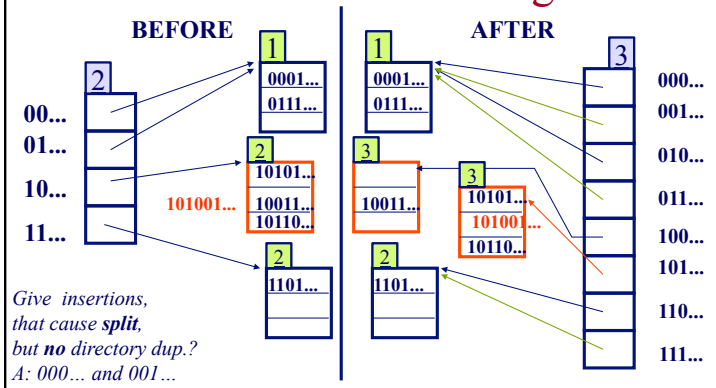

CMU SCS
Extendible hashing
BEFORE
AFTER
00...
01...
10...
11...
0001...
0111...
10101...
10011...
10110...
101001...
1101...
000...
001...
010...
011...
100...
101...
110...
111...
Do all splits lead to directory doubling?
A:
CMU SCS 15-415/615
42


CMU SCS
Extendible hashing
BEFORE
AFTER
00...
01...
10...
11...
0001...
0111...
10101...
10011...
10110...
101001...
1101...
000...
001...
010...
011...
100...
101...
110...
111...
Do all splits lead to directory doubling?
A:NO! (most don't)
CMU SCS 15-415/615
43


CMU SCS
Extendible hashing
BEFORE
AFTER
00...
01...
10...
11...
0001...
0111...
10101...
10011...
10110...
101001...
1101...
000...
001...
010...
011...
100...
101...
110...
111...
Give insertions, that cause split, but no directory dup.?
Faloutsos - Pavlo
CMU SCS 15-415/615
44

CMU SCS

Extendible hashing



*Give insertions, that cause **split**, but **no** directory dup.? A: 000… and 001…*

CMU SCS

Extendible hashing

- Summary: directory doubles on demand
- or halves, on shrinking files
- needs 'local' and 'global' depth

CMU SCS

Outline

- (static) hashing
- extendible hashing
- linear hashing
- Hashing vs B-trees

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- deletion

CMU SCS

Linear hashing

Motivation: ext. hashing needs directory etc etc; which doubles (ouch!)

Q: can we do something simpler, with smoother growth?

CMU SCS

Linear hashing

Motivation: ext. hashing needs directory etc etc; which doubles (ouch!)

Q: can we do something simpler, with smoother growth?

A: split buckets from left to right, **regardless** of which one overflowed ('crazy' , but it works well!) - Eg.:

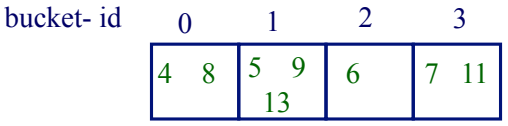
CMU SCS

Linear hashing

Initially: *h(x) = x mod N* (N=4 here)

Assume capacity: 3 records / bucket

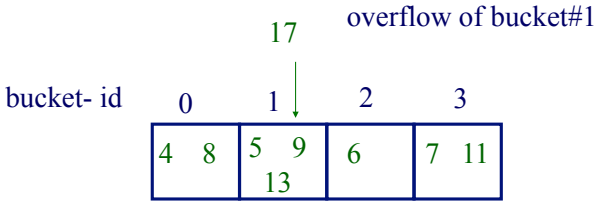
Insert key '17'



CMU SCS

Linear hashing

Initially: *h(x) = x mod N* (N=4 here)



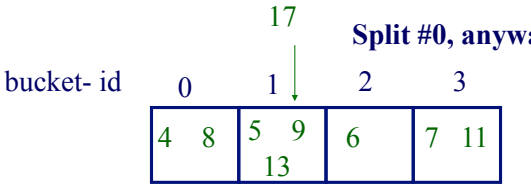
CMU SCS

Linear hashing

Initially: *h(x) = x mod N* (N=4 here)

overflow of bucket#1

Split #0, anyway!!!



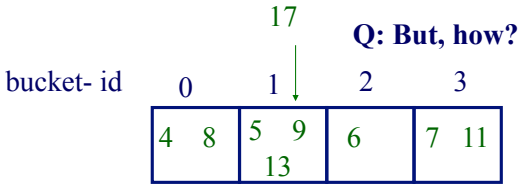
CMU SCS

Linear hashing

Initially: *h(x) = x mod N* (N=4 here)

Split #0, anyway!!!

Q: But, how?

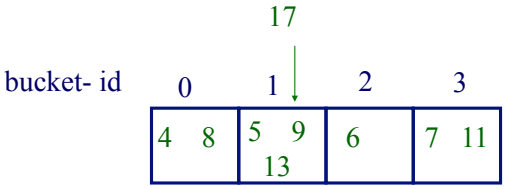


CMU SCS

Linear hashing

A: use two h.f.: *h0(x) = x mod N*

*h1(x) = x mod (2*N)*

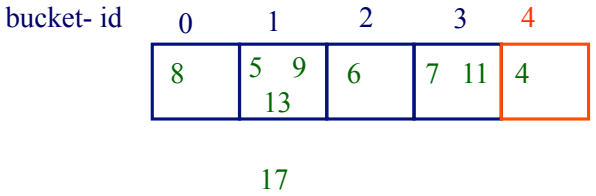


CMU SCS

Linear hashing - after split:

A: use two h.f.: *h0(x) = x mod N*

*h1(x) = x mod (2*N)*

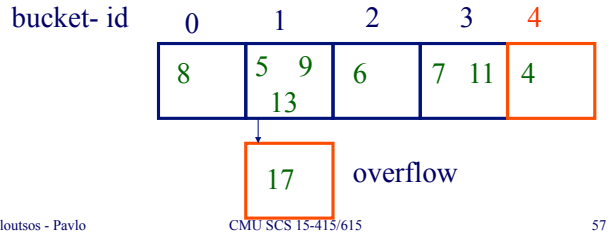


CMU SCS

Linear hashing - after split:

A: use two h.f.: $h0(x) = x \bmod N$

$h1(x) = x \bmod (2*N)$

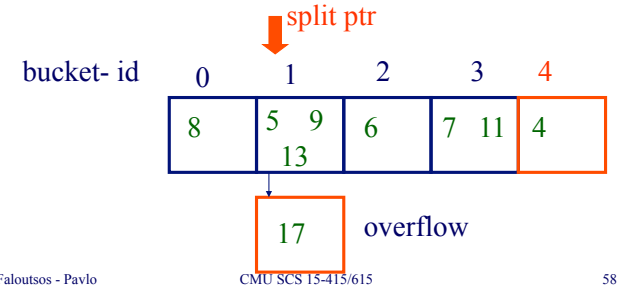


CMU SCS

Linear hashing - after split:

A: use two h.f.: $h0(x) = x \bmod N$

$h1(x) = x \bmod (2*N)$



CMU SCS

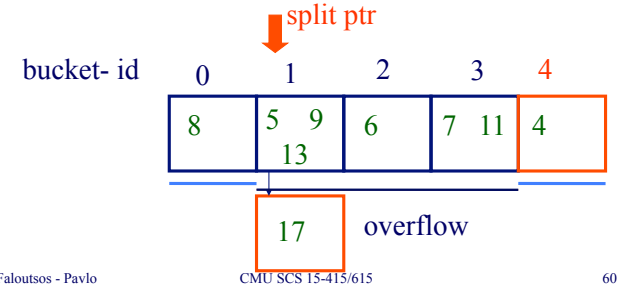
Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- deletion

CMU SCS

Linear hashing - searching?

- $h0(x) = x \bmod N$ *(for the un-split buckets)*
- $h1(x) = x \bmod (2*N)$ *(for the splitted ones)*

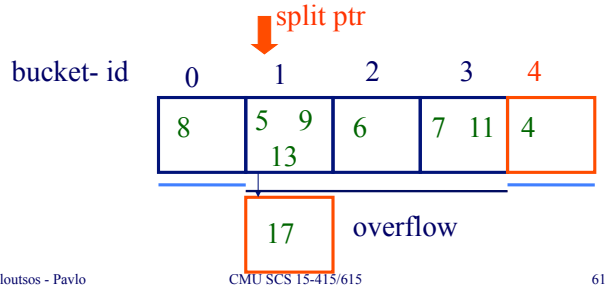


CMU SCS

Linear hashing - searching?

Q1: find key ‘6’ ? Q2: find key ‘4’ ?

Q3: key ‘8’ ?



CMU SCS

Linear hashing - searching?

Algo to find key ‘k’ :

compute *b= h0(k);* *// original slot*

if *b<split-ptr* *// has been split*

compute *b=h1(k)*

search bucket *b*

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- deletion

CMU SCS

Linear hashing - insertion?

Algo: insert key ‘*k*’

- compute appropriate bucket ‘*b*’
- if the **overflow criterion** is true
  - split the bucket of ‘split-ptr’
  - split-ptr ++ (*)

CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!

Q: suggestions?

CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!

Q: suggestions?

A1: space utilization >= u-max

CMU SCS

Linear hashing - insertion?

notice: overflow criterion is up to us!!

Q: suggestions?

A1: space utilization >= u-max

A2: avg length of ovf chains > max-len

A3: ....

CMU SCS

Linear hashing - insertion?

Algo: insert key '*k*'

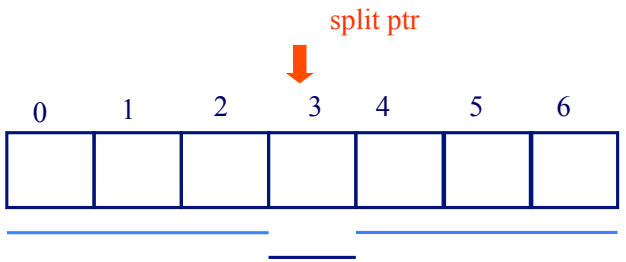
- compute appropriate bucket '*b*'
- if the **overflow criterion** is true
  - split the bucket of 'split-ptr'
  - split-ptr ++ **(*)**

what if we reach the right edge??

CMU SCS

Linear hashing - split now?

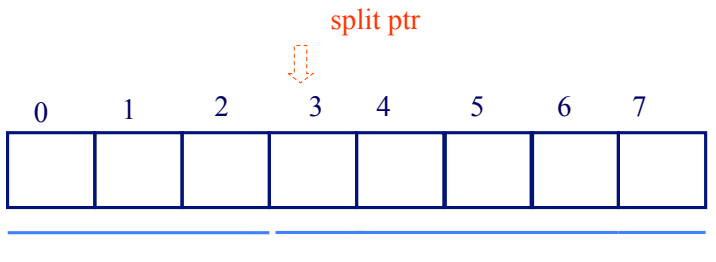
h0(x) = x mod N (for the un-split buckets)
*h1(x) = x mod (2*N) for the splitted ones)*



CMU SCS

Linear hashing - split now?

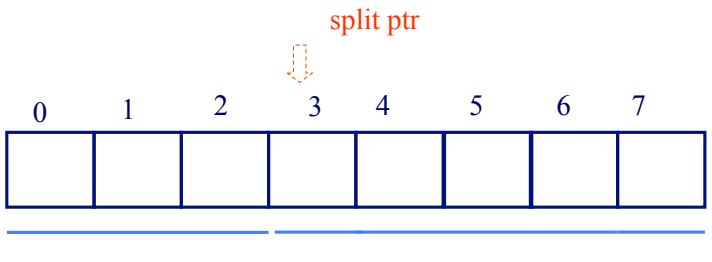
h0(x) = x mod N (for the un-split buckets)
*h1(x) = x mod (2*N) (for the splitted ones)*



CMU SCS

Linear hashing - split now?

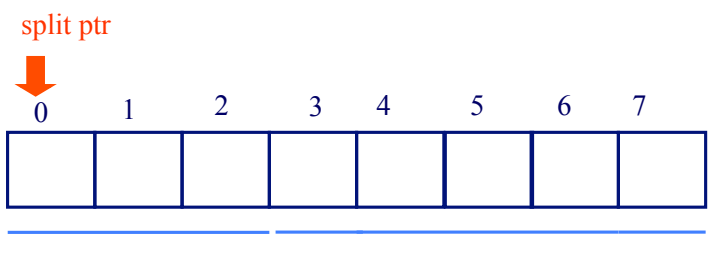
~~*h0(x) = x mod N (for the un-split buckets)*~~
*h1(x) = x mod (2*N) (for the splitted ones)*



CMU SCS

Linear hashing - split now?

~~*h0(x) = x mod N (for the un-split buckets)*~~
*h1(x) = x mod (2*N) (for the splitted ones)*

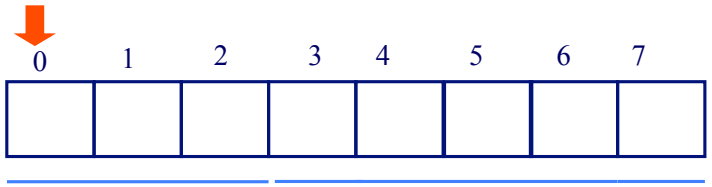


CMU SCS

Linear hashing - split now?

this state is called **'full expansion'**

split ptr



CMU SCS

Linear hashing - observations

In general, at any point of time, we have at **most two** h.f. active, of the form:

- $h_n(x) = x \bmod (N * 2^n)$
- $h_{n+1}(x) = x \bmod (N * 2^{n+1})$

(after a full expansion, we have only one h.f.)

CMU SCS

Linear hashing - overview

- Motivation
- main idea
- search algo
- insertion/split algo
- deletion

CMU SCS

Linear hashing - deletion?

- reverse of insertion:

CMU SCS

Linear hashing - deletion?

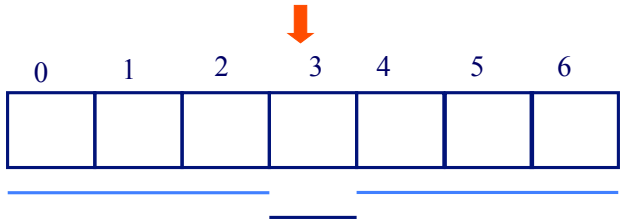
- reverse of insertion:
- if the underflow criterion is met
  - contract!

CMU SCS

Linear hashing - how to contract?

h0(x) = mod N *(for the un-split buckets)*
*h1(x) = mod (2*N)* *(for the splitted ones)*

split ptr

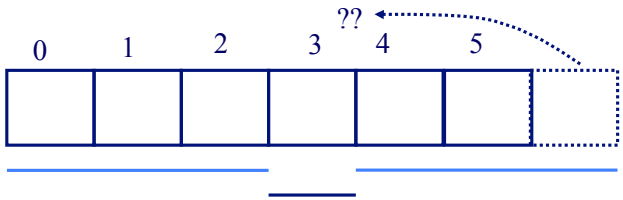


CMU SCS

Linear hashing - how to contract?

h0(x) = mod N *(for the un-split buckets)*
*h1(x) = mod (2*N)* *(for the splitted ones)*

split ptr

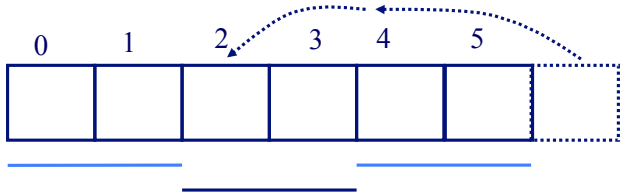


CMU SCS

Linear hashing - how to contract?

h0(x) = mod N *(for the un-split buckets)*
*h1(x) = mod (2*N)* *(for the splitted ones)*

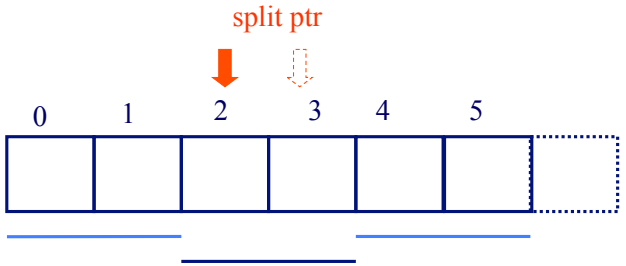
split ptr



CMU SCS

Linear hashing - how to contract?

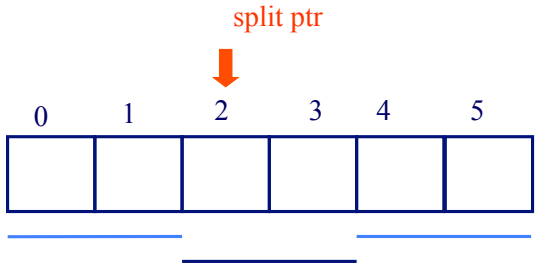
h0(x) = mod N *(for the un-split buckets)*
*h1(x) = mod (2*N)* *(for the splitted ones)*



CMU SCS

Linear hashing - how to contract?

h0(x) = mod N *(for the un-split buckets)*
*h1(x) = mod (2*N)* *(for the splitted ones)*



CMU SCS

Outline

- (static) hashing
- extendible hashing
- linear hashing
- Hashing vs B-trees

CMU SCS

Hashing - pros?

CMU SCS

Hashing - pros?

- Speed,
  - on exact match queries
  - on the average

CMU SCS

B(+)-trees - pros?

CMU SCS

B(+)-trees - pros?

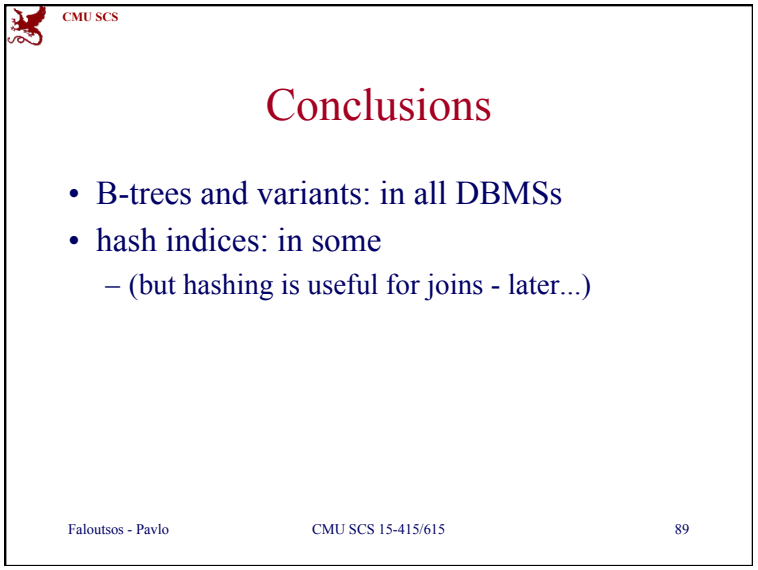
- Speed on search:
  - exact match queries, worst case
  - range queries
  - nearest-neighbor queries
- Speed on insertion + deletion
- smooth growing and shrinking (no re-org)

CMU SCS

B(+)-trees vs Hashing

- Speed on search:
  - exact match queries, worst case
  - range queries
  - nearest-neighbor queries
- Speed on insertion + deletion
- smooth growing and shrinking (no re-org)

- Speed,
  - on exact match queries
  - on the average


CMU SCS
Conclusions
• B-trees and variants: in all DBMSs
• hash indices: in some
– (but hashing is useful for joins - later...)
Faloutsos - Pavlo CMU SCS 15-415/615 89