UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 3380 Spring 2024

Database Systems

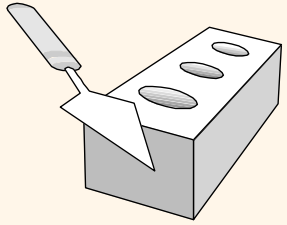
M & W 4:00 to 5:30 PM

Prof. Victoria Hilford

PLEASE TURN your webcam ON (must have)

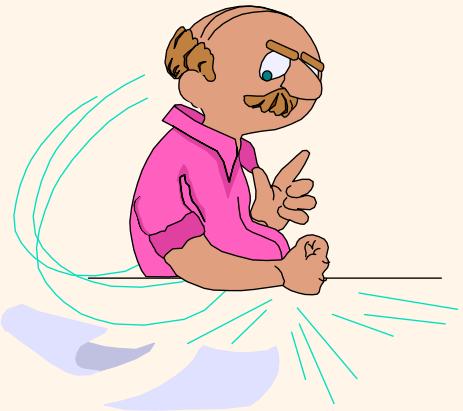
NO CHATTING during LECTURE

VH, unhide Section 16: SET 3 STORAGE IV – HASH INDEX



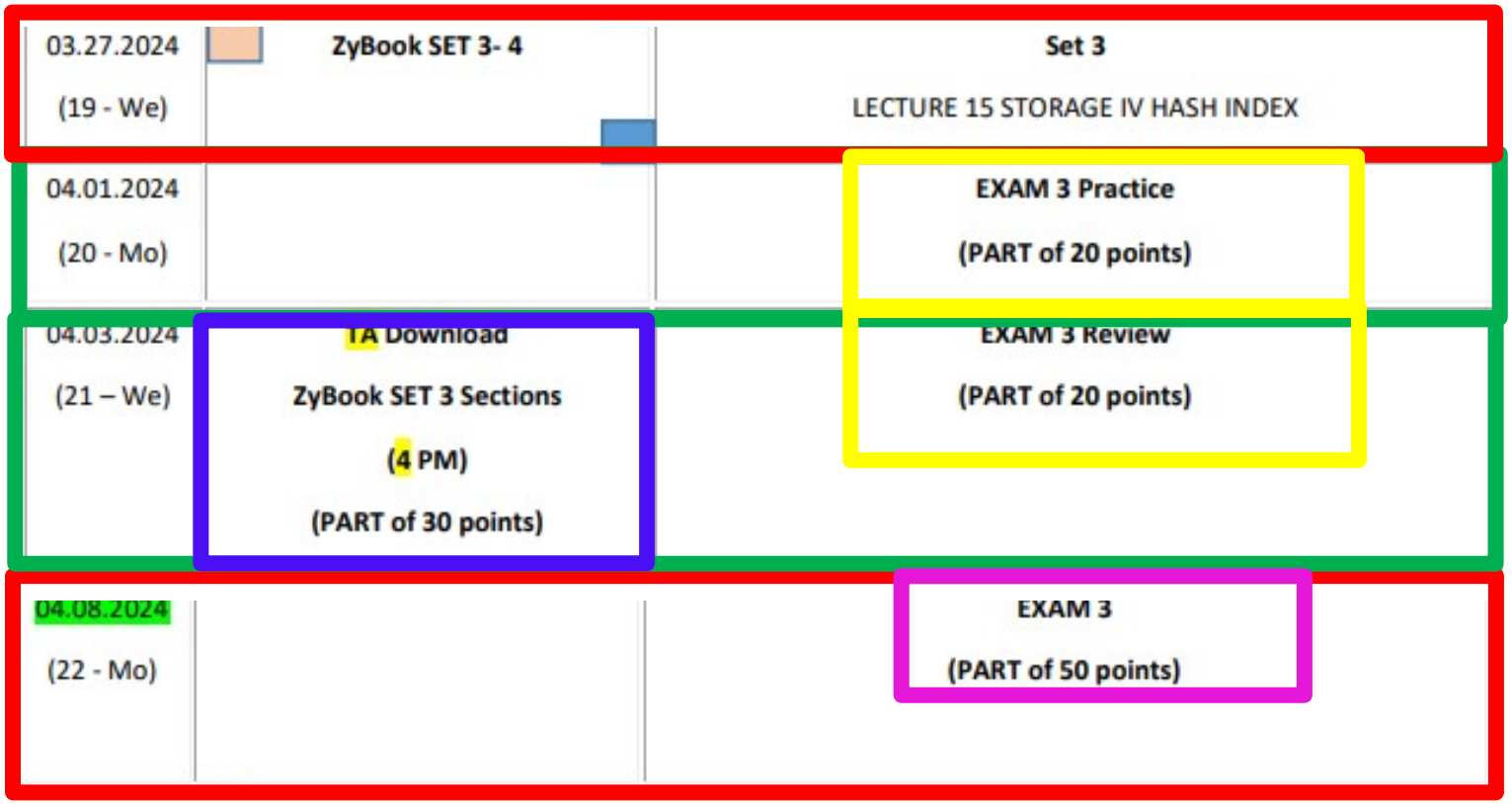
COSC 3380

4 to 5:30



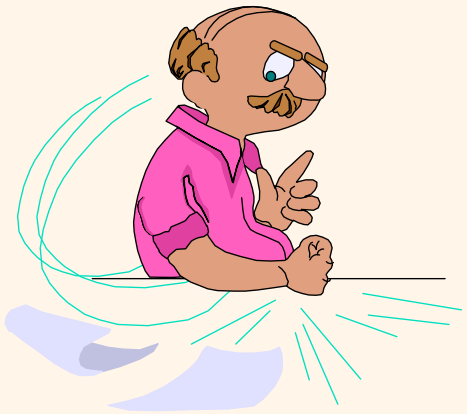
PLEASE
LOG IN
CANVAS

Please close all other windows.


03.27.2024
(19 - We)
ZyBook SET 3- 4
Set 3
LECTURE 15 STORAGE IV HASH INDEX
04.01.2024
(20 - Mo)
EXAM 3 Practice
(PART of 20 points)
04.03.2024
(21 – We)
TA Download
ZyBook SET 3 Sections
(4 PM)
(PART of 30 points)
EXAM 3 Review
(PART of 20 points)
04.08.2024
(22 - Mo)
EXAM 3
(PART of 50 points)

3

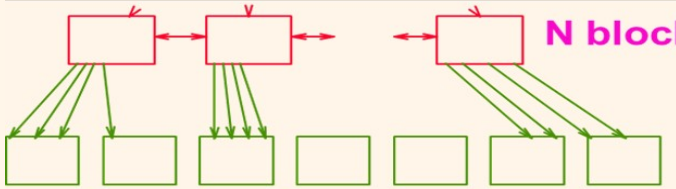
COSC 3380



Class 19

03.27.2024 (19 - We)	ZyBook SET 3- 4	Set 3 LECTURE 15 STORAGE IV HASH INDEX

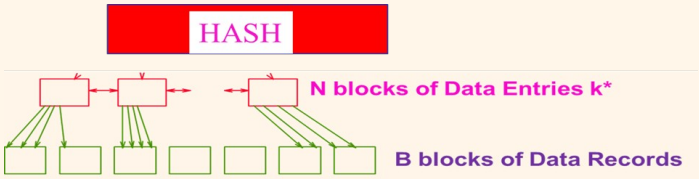
HASH



From 4:00 to 4:07 PM – 5 minutes.

03.27.2024 (19 - We)	ZyBook SET 3- 4	Set 3 LECTURE 15 STORAGE IV HASH INDEX

CLASS PARTICIPATION 20 points — 20% of Total



HASH INDEX

Class 19 BEGIN PARTICIPATION
Not available until Mar 27 at 4:00pm | Due Mar 27 at 4:07pm | 100 pts

VH, publish

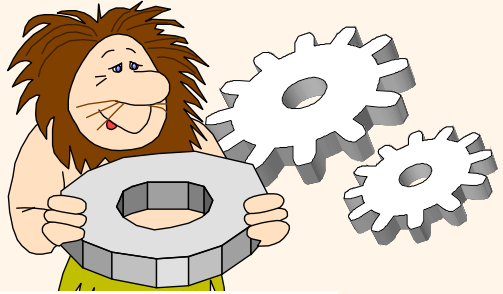
This is a synchronous online class.

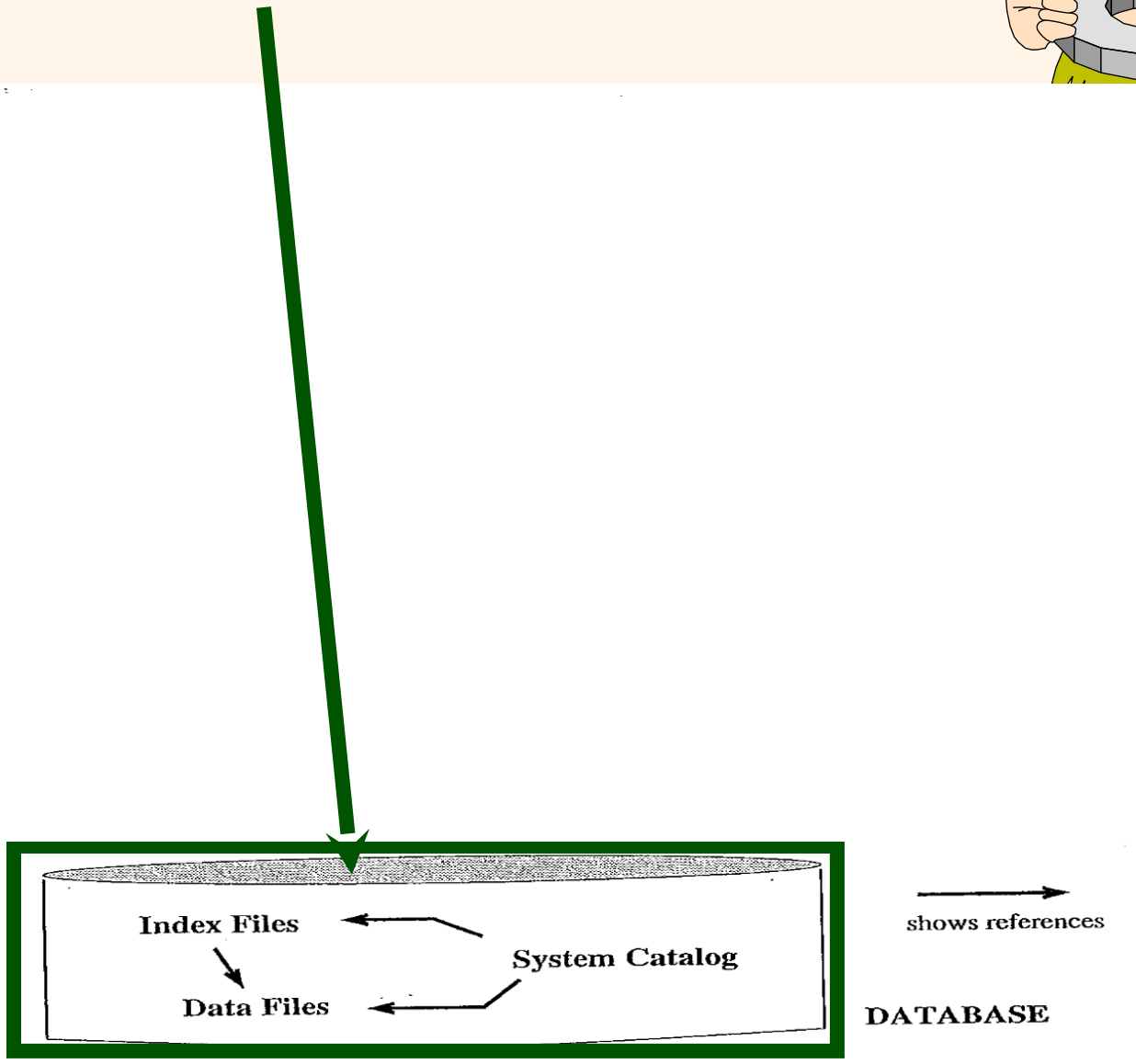
Attendance is required.

Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

A DBMS





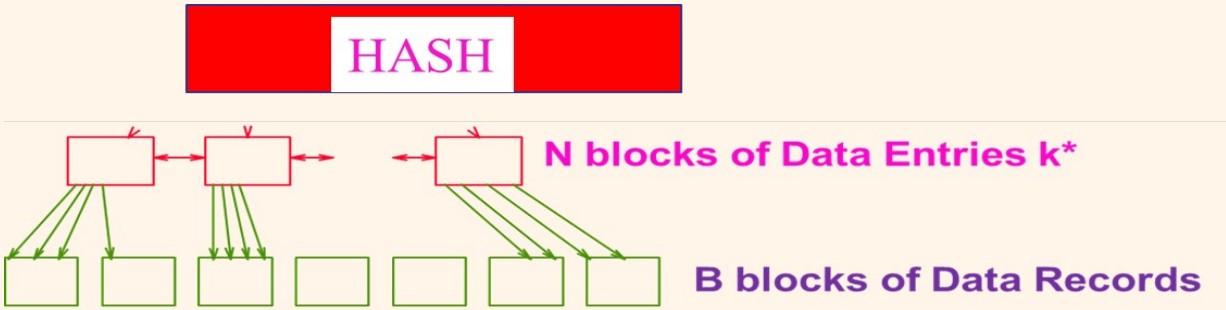
Tables
Indexes

From 4:07 to 5:00 PM – 53 minutes.

COSC 3380

Lecture 15

Hash Based Indexes



h + 1 + 1

Leaf (Data Entry k*) blocks; 1 Table Block

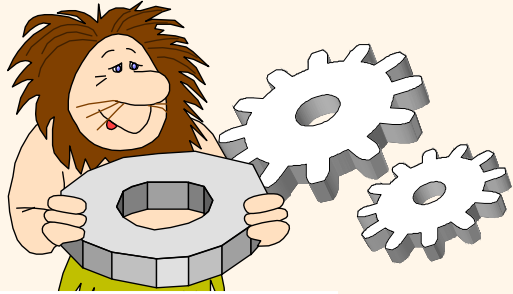
h + 1 + 1+ 1 + 1

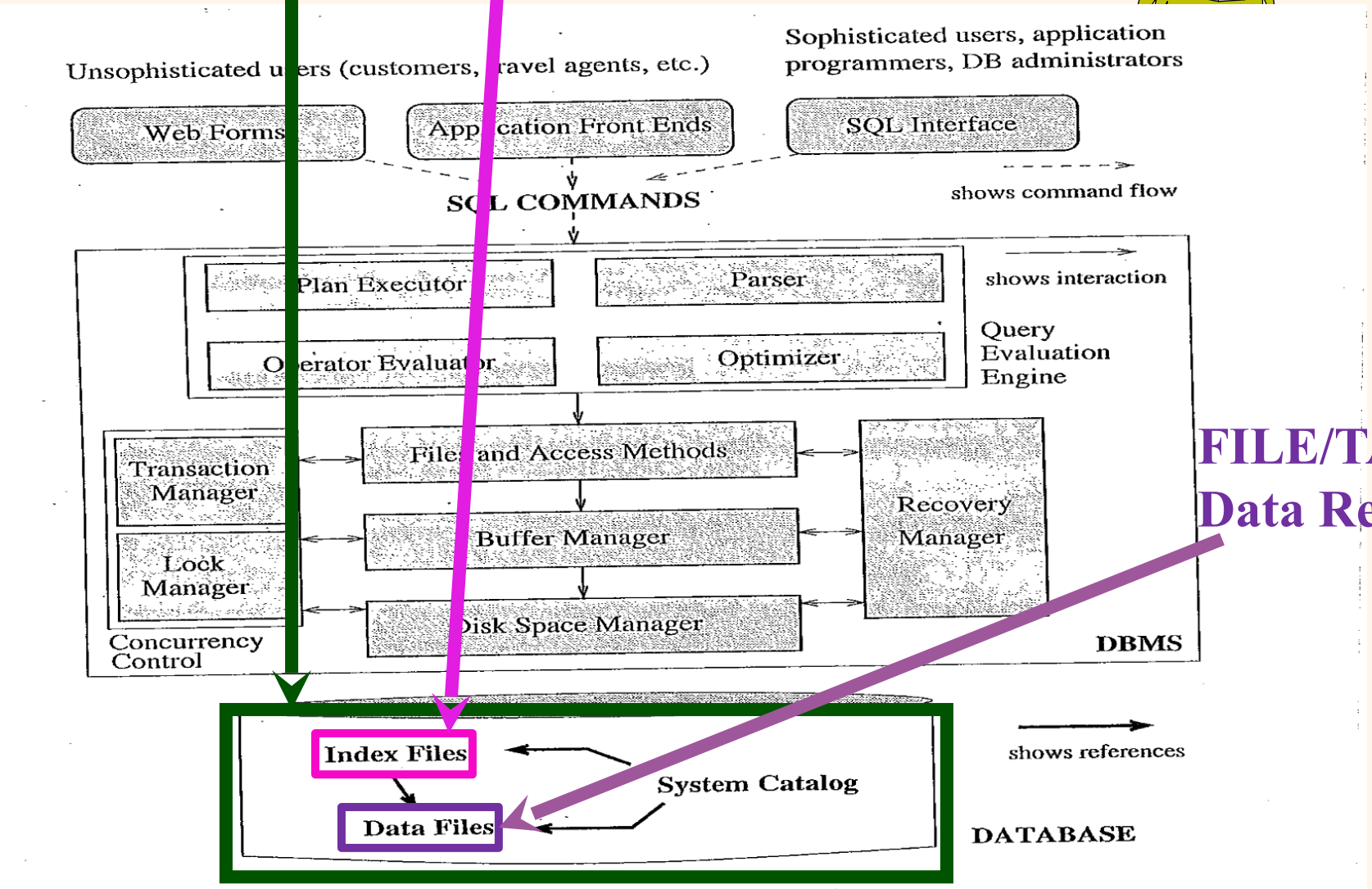
2 (read and write back) Leaf (Data Entry k*) blocks; 2 (read and write back) Table Blocks

A DBMS

Index File

Data Entries k*



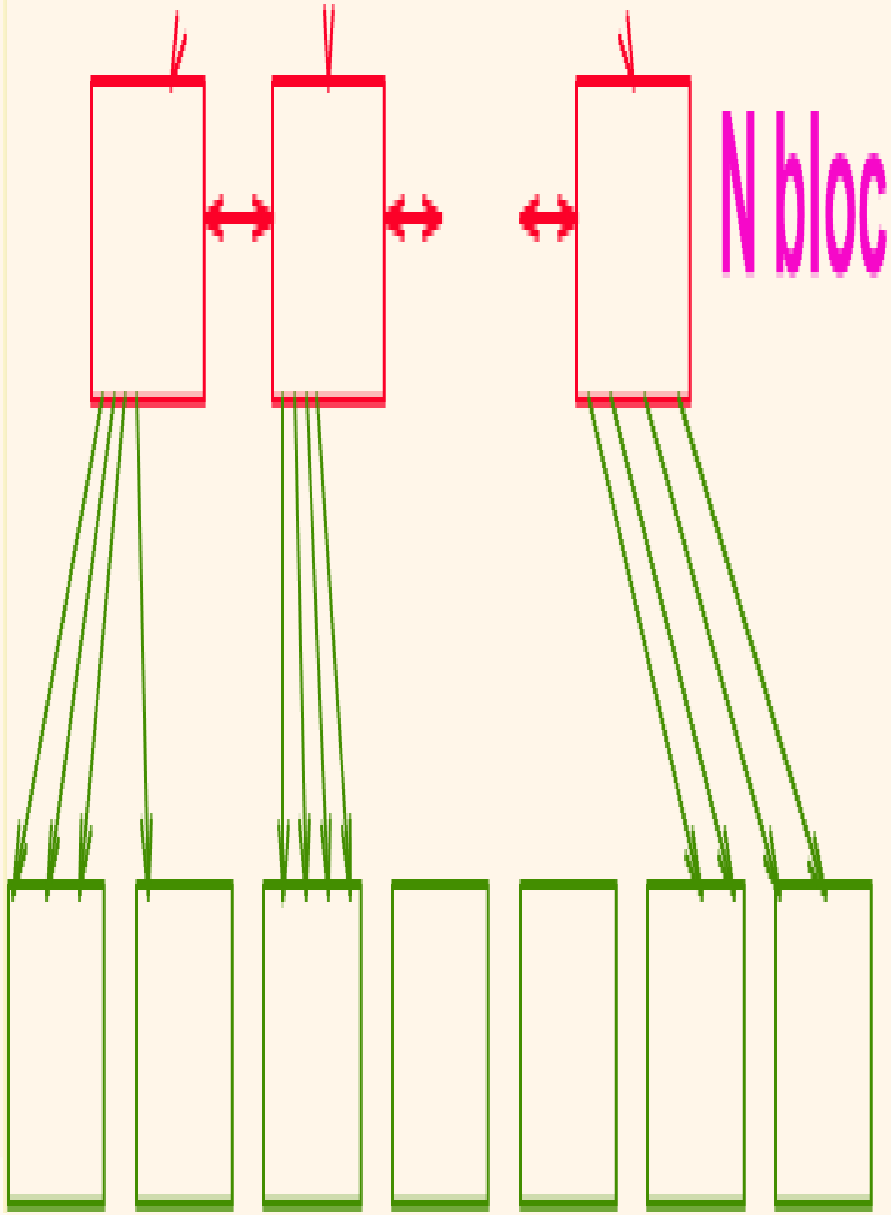


FILE/TABLE

Data Records

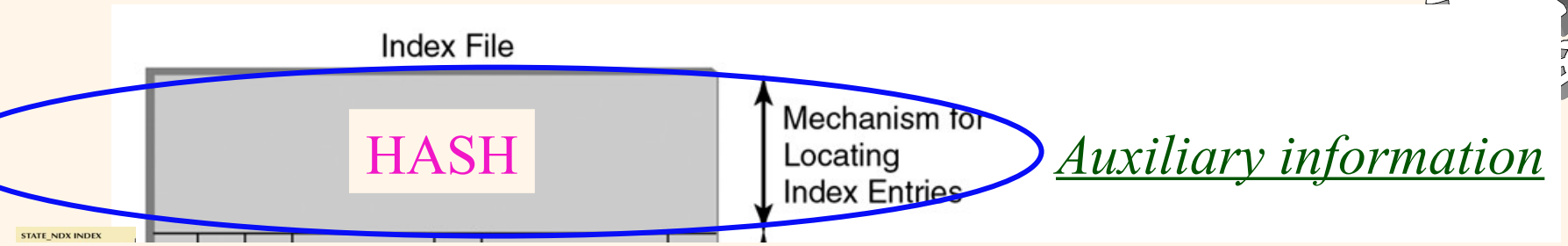
Indexes

HASH

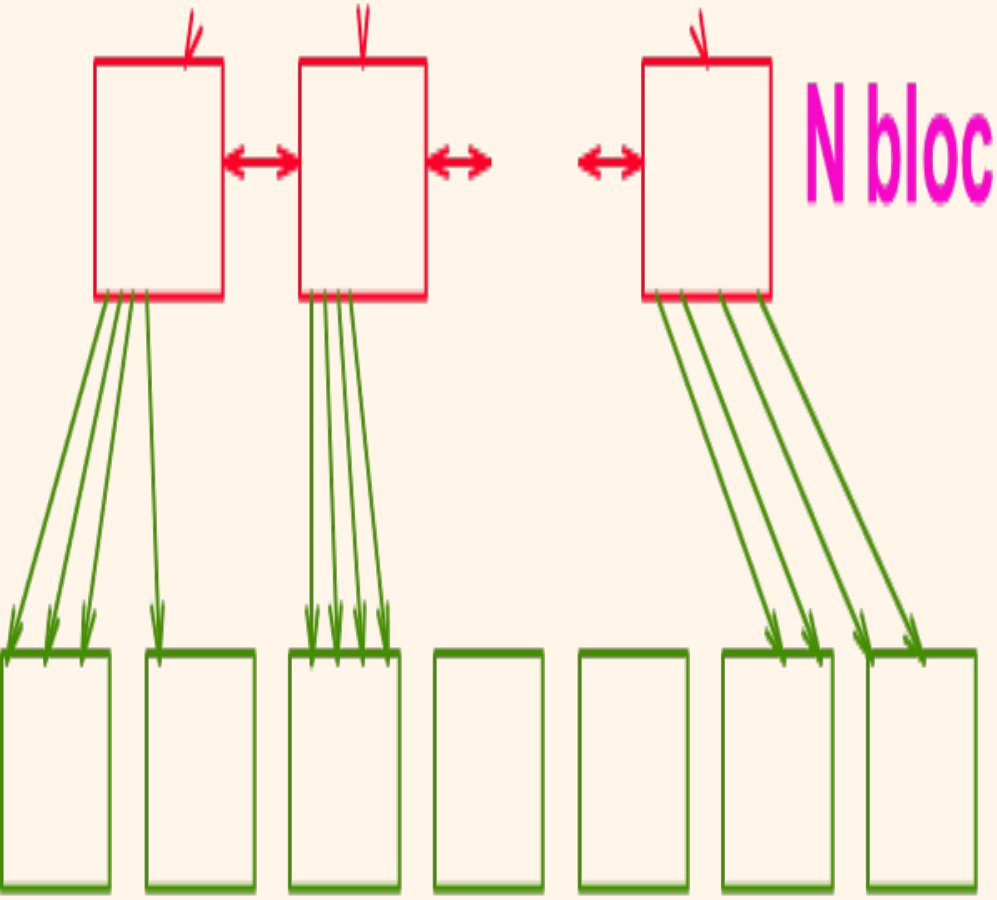


N blocks of Data Entries k*

B blocks of Data Records



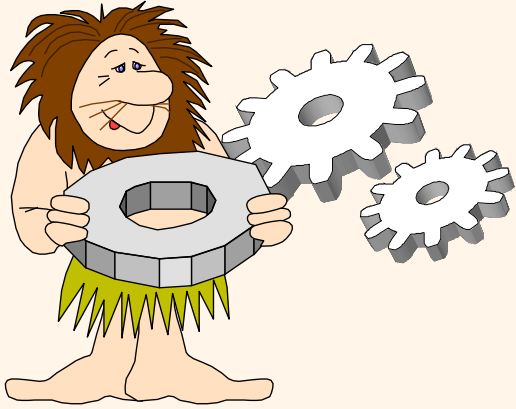
Auxiliary information

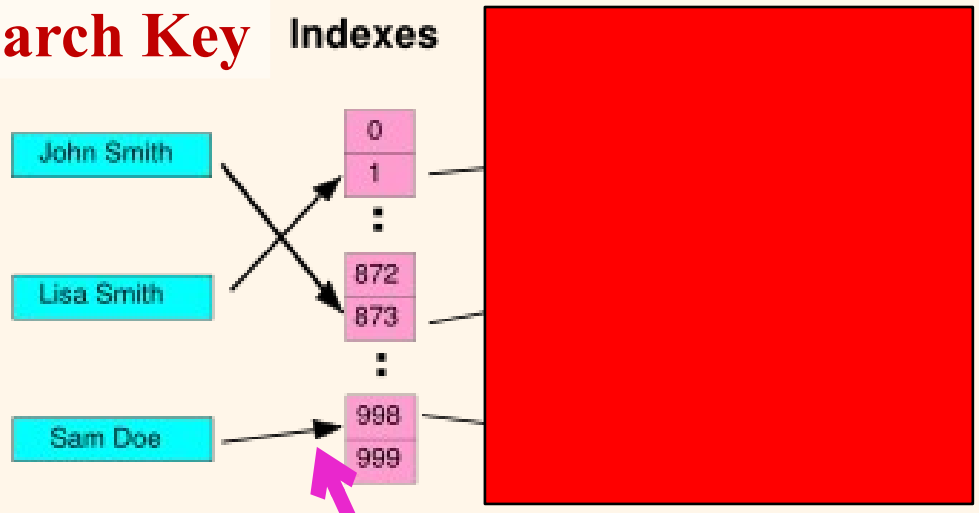


N blocks of Data Entries k*

B blocks of Data Records

Hash Table *(Hash Map)*



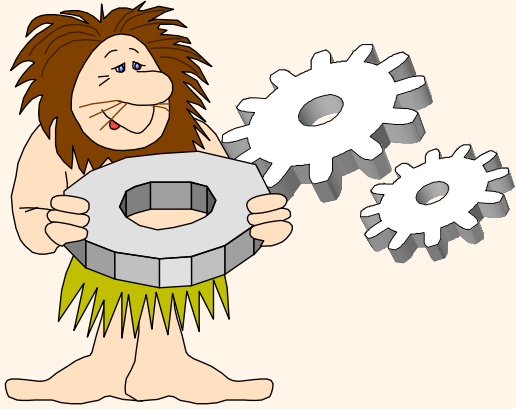


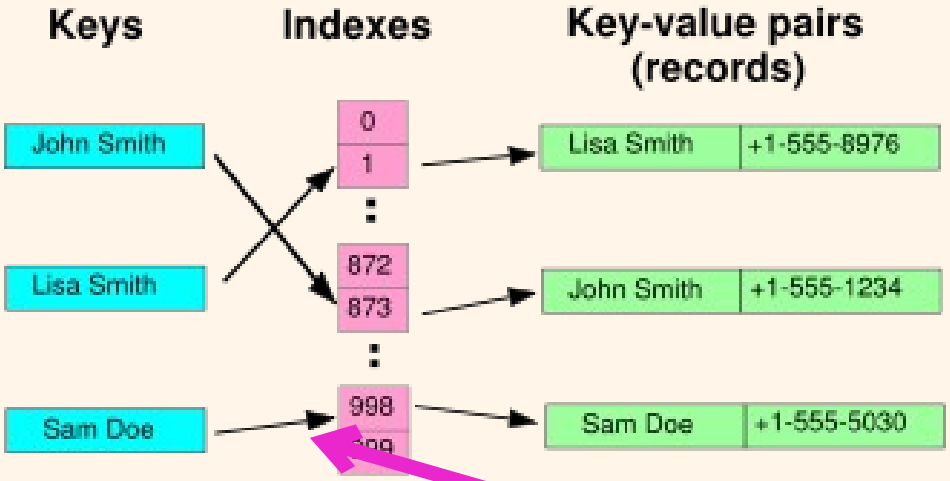
A small phone book as a ***Hash Table (***Array***).***

A ***Hash Table*** associates **keys** with **values**. Given a **key** (e.g., a *person's name*), find the corresponding **value** (e.g. that *person's telephone number*).

Transform the **key** using a ***hash function*** into a **hash**, a number that is used as an **index** in an ***Hash Table*** *(*Array) to locate the desired location ("**bucket**") where the **values** should be

Hash Table *(Hash Map)*



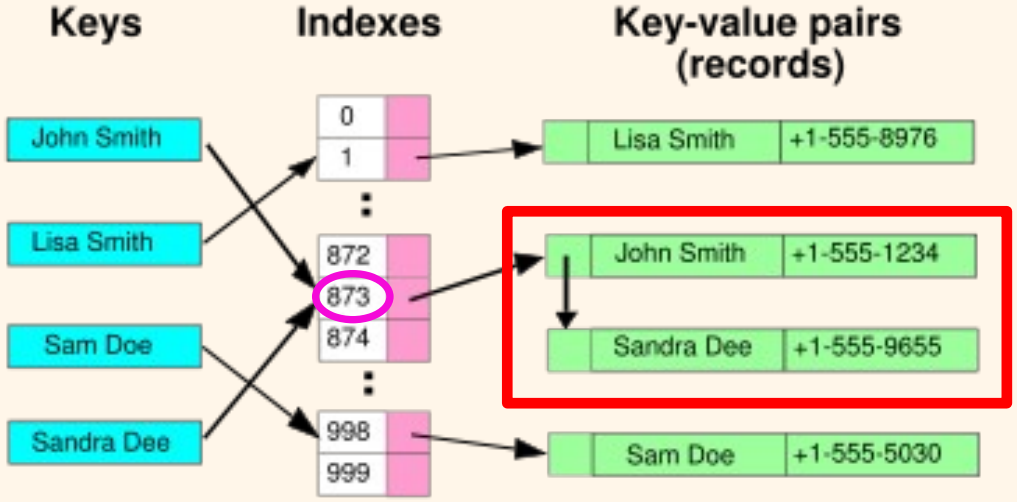


A small phone book as a ***Hash Table (***array***)****.*

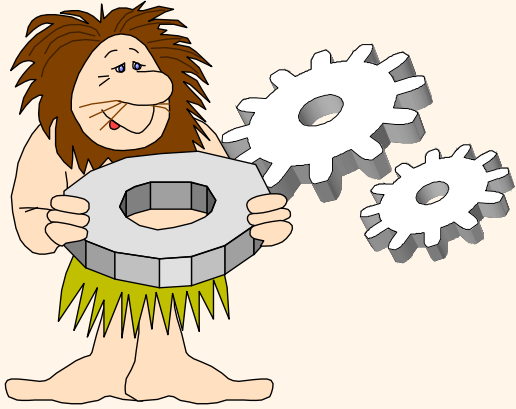
Collision: when two **keys** **hash** to the same ***Hash Table*** *(*Array) **index**, an alternate location must be determined because two **keys** cannot be stored in the same ***Hash Table*** *(*Array) location (same **index**).

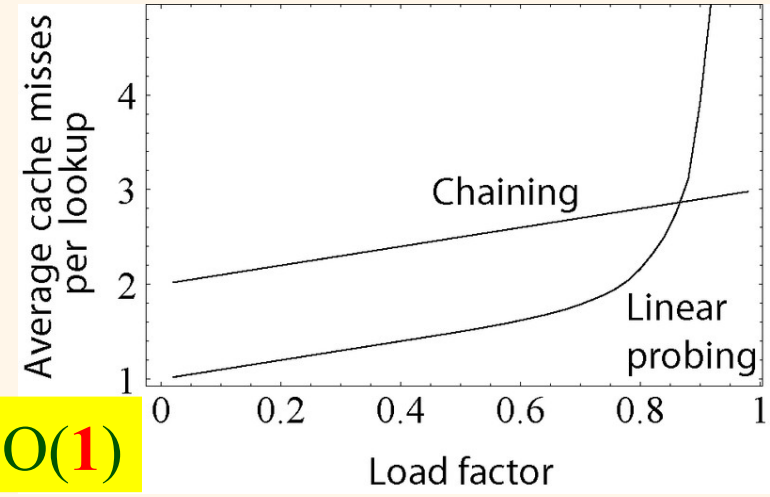
Collision resolution: **Separate Chaining** & **Open**

Separate Chaining

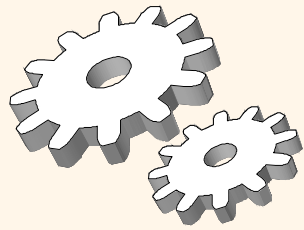


Bucket (Linked)





O(1)



Hash-Based* Indexes** are best for ***equality *selections*. ***Cannot*** support **range** searches. O(1)

Static and ***Dynamic*** **Hashing Index** techniques exist;

- trade-offs similar to **ISAM** vs. **B+ Trees Index**.

Hash table

In a ***hash table***, rows are assigned to buckets. A ***bucket*** is a block or group of blocks containing rows. Initially, each bucket has one block. As a table grows, some buckets eventually fill up with rows, and the database allocates additional blocks. New blocks are linked to the initial block, and the bucket becomes a chain of linked blocks.

The bucket containing each row is determined by a hash function and a hash key. The ***hash key*** is a column or group of columns, usually the primary key. The ***hash function*** computes the bucket containing the row from the hash key.

Hash functions are designed to scramble row locations and evenly distribute rows across blocks. The ***modulo function*** is a simple hash function with four steps:

1. Convert the hash key by interpreting the key's bits as an integer value.
2. Divide the integer by the number of buckets.
3. Interpret the division remainder as the bucket number.
4. Convert the bucket number to the physical address of the block containing the row.

As tables grow, a fixed hash function allocates more rows to each bucket, creating deep buckets consisting of long chains of linked blocks. Deep buckets are inefficient since a query may read several blocks to access a single row. To avoid deep buckets, databases may use dynamic hash functions. A ***dynamic hash function*** automatically allocates more blocks to the table, creates additional buckets, and distributes rows across all buckets. With more buckets, fewer rows are assigned to each bucket and, on average, buckets contain fewer linked blocks.

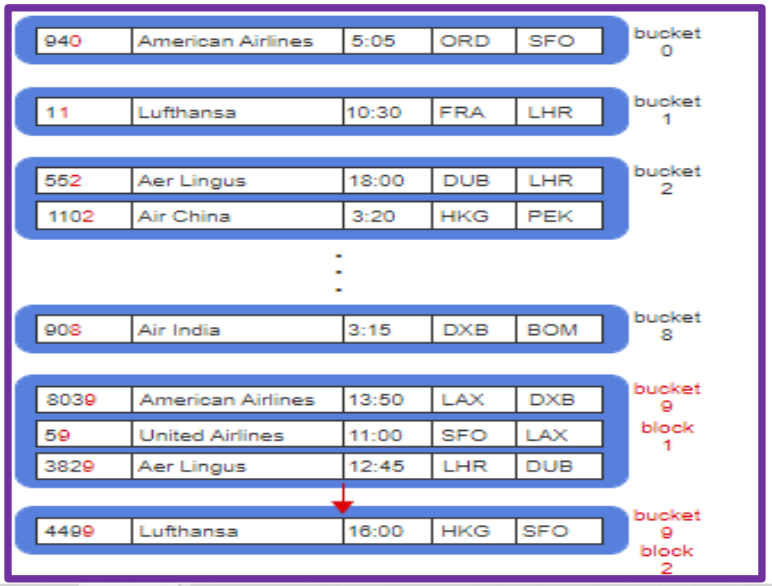
Hash tables are optimal for inserts and deletes of individual rows, since row location is quickly determined from the hash key. For the same reason, hash tables are optimal for selecting a single row when the hash key value is specified in the WHERE clause. Hash tables are slow on queries that select many rows with a range of values, since rows are randomly distributed across many blocks.

PARTICIPATION ACTIVITY 13.2.5: Assigning rows to buckets with a modulo function.

Flight

Flight Number	AirlineName	Depart Time	Depart Airport	Arrive Airport
11	Lufthansa	10:30	FRA	LHR
552	Aer Lingus	18:00	DUB	LHR
908	Air India	3:15	DXB	BOM
940	American Airlines	5:05	ORD	SFO
1102	Air China	3:20	HKG	PEK
8039	American Airlines	13:50	LAX	DXB
59	United Airlines	11:00	SFO	LAX
3829	Aer Lingus	12:45	LHR	DUB
4499	Lufthansa	16:00	HKG	SFO

hash key



Hash indexes

The multi-level index is the most commonly used index type. Several additional index types are used less often but supported by many databases:

- Hash index
- Bitmap index
- Logical index
- Function index

Hash indexes

The multi-level index is the most commonly used index type. Several additional index types are used less often but supported by many databases:

- Hash index
- Bitmap index
- Logical index
- Function index

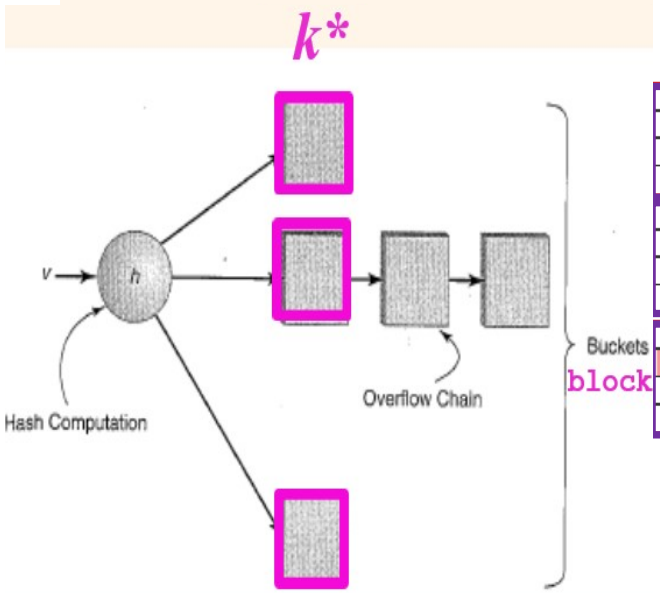
In a ***hash index***, index entries are assigned to buckets. A ***bucket*** is a block or group of blocks containing index entries. Initially, each bucket has one block. As an index grows, some buckets eventually fill up, and additional blocks are allocated and linked to the initial block.

The bucket containing each index entry is determined by a ***hash function***, which computes a bucket number from the value of the indexed column. To locate a row containing a column value, the database:

1. Applies the hash function to the column value to compute a bucket number.
2. Reads the index blocks for the bucket number.
3. Finds the index entry for the column value and reads the table block pointer.
4. Reads the table block containing the row.

A hash index is similar to a hash table, described in another section. However, a hash index stores **index entries** in each bucket, while a hash table stores **table rows** in each bucket.

PARTICIPA
ACTIVITY



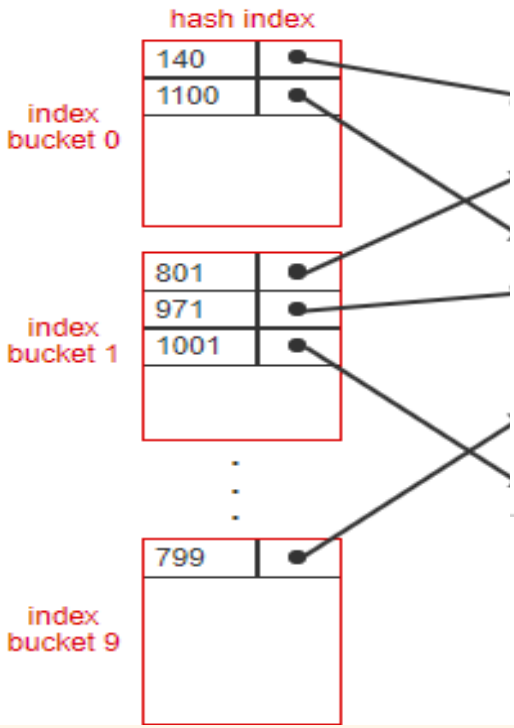
table

666	United Airlines	6:00	ATL	Airbus 321
140	Aer Lingus	10:30	DUB	Boeing 737
803	Air China	3:20	DME	Boeing 777
801	Air India	3:15	DXB	Boeing 747
552	Lufthansa	18:00	FRA	MD 111
1100	Air China	3:20	HKG	Boeing 787
971	Air India	19:35	LAX	Boeing 747
702	American Airlines	5:05	ORD	Airbus 323
1154	American Airlines	13:50	ORD	MD 111
799	British Airways	13:50	PHL	Airbus 323
1222	Delta Airlines	8:25	SEA	Boeing 747
1001	Southwest Airlines	5:05	SJC	Airbus 323

blocks

N blocks of Data Entries k*

B blocks of Data Records



B blocks of Data Records

N blocks of Data Entries k*

Terminology

Sometimes the term hash index is used to mean a hash key, but the two terms are different. A ***hash index*** *is an index that is structured using a hash function. A* ***hash key*** *is a column that determines the physical location of rows in a hash table.*

PARTICIPATION ACTIVITY 16.1.2: Hash indexes.

1) A hash table can have a hash index.

- (•) True
- ( ) False

Correct

Hash tables cannot have a primary or clustering index because rows of a hash table are not stored in sort order. However, hash tables can have secondary indexes. The secondary indexes can be any structure, including hash.

?????

TA time (Jordan)
(CA 16.1.1 Step 1 – Hash Index)

CHALLENGE ACTIVITY | 16.1.1: Other indexes.

Hash Index

Given that the hash index is on FlightNumber, and the hash function is modulo 10, select the row(s) with a hash index in bucket 3.

Ending in 3

table

☐	581	United Airlines	6:00	ATL	Airbus 321
☐	696	Aer Lingus	10:30	DUB	Boeing 737
☐	126	Air China	3:20	DME	Boeing 777
☐	183	Air India	3:15	DXB	Boeing 747
☐	165	Lufthansa	18:00	FRA	MD 111
☐	783	Air China	3:20	HKG	Boeing 787
☐	721	Air India	19:35	LAX	Boeing 747
☐	924	American Airlines	5:05	ORD	Airbus 323
☐	936	American Airlines	13:50	ORD	MD 111
☐	686	British Airways	13:50	PHL	Airbus 323
☐	714	Delta Airlines	8:25	SEA	Boeing 747
☐	791	Southwest Airlines	5:05	SJC	Airbus 323

183

783

1

VH

?????

COSC 3380 23578 M & W 4 to 5:30 PM LECTURE (in TEAMS from 3:50 to 5:15 PM)

35:42

Pop out | Chat | People | Raise | View | Rooms | Apps | More | Camera | Mic | Share | Leave

Yu, Jordan T

View all

Participants

Invite someone or dial a number

Share invite

Waiting in lobby (1)

VM Martinez, Valerio

In this meeting (111) Mute all

Hilford, Victoria
Organizer

RA Adhikari, Rohit

MA Ahmed, Mohamed A

AA Akram, Ali

BA Akukwe, Benetta O

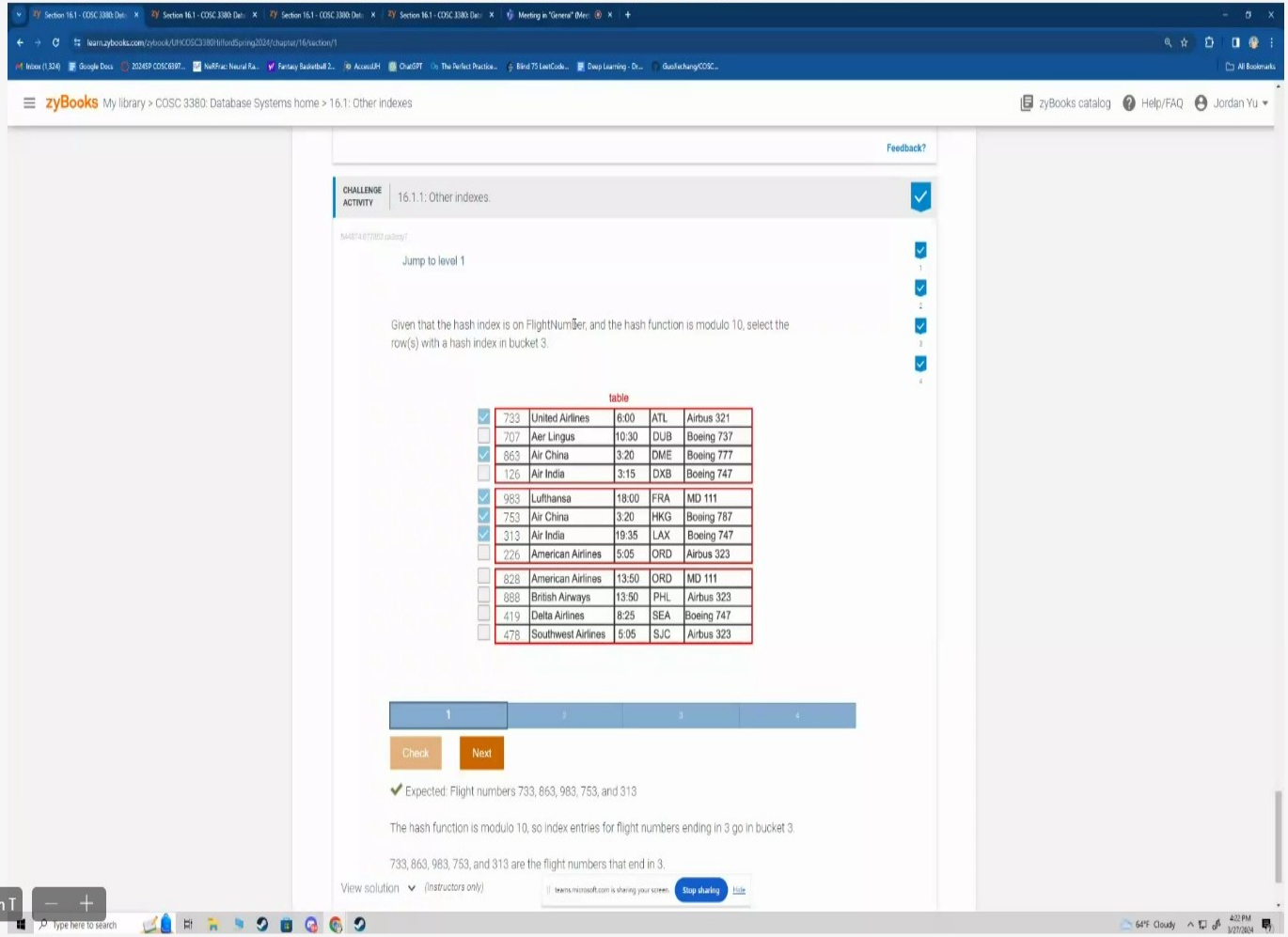
SA Alsayed, Sami H

SA Altaf, Sameer

SA Alvarez, Stephanie

OA Anayor-Achu, Ogochukwu E

HA Avci, Hatice Kubra

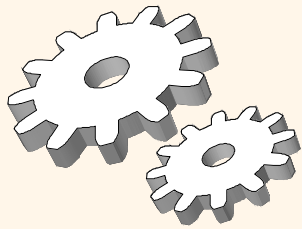


Yu, Jordan T

TA, Jordan (A – L).

TA, Alvaro (M – Z).

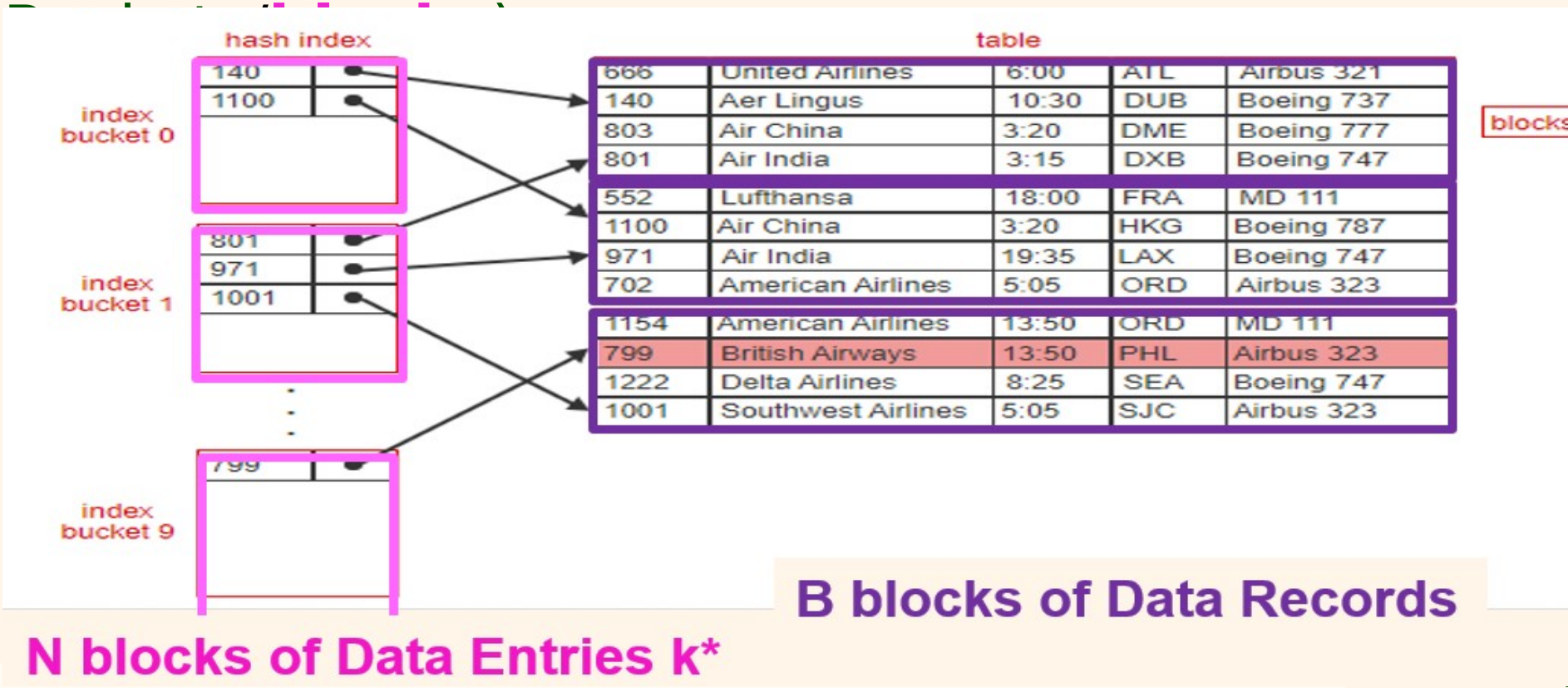
Please compare CANVAS vs. TEAMS Attendance.
Print screens of students in CANVAS but not in the TEAMS meeting.
(3.27.2024 Attendance X missing LastName.docx)



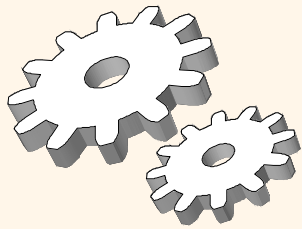
Static Hashing Indexes

Static Hashing Indexes

- # primary **blocks** f***ixed***, allocated sequentially, never de-allocated; **overflow blocks if needed**.
- **h(*key*) mod N** = Bucket/**Page** to which data entry ***k**** with key *key* belongs. (**N** = # of



Static Hashing Indexes



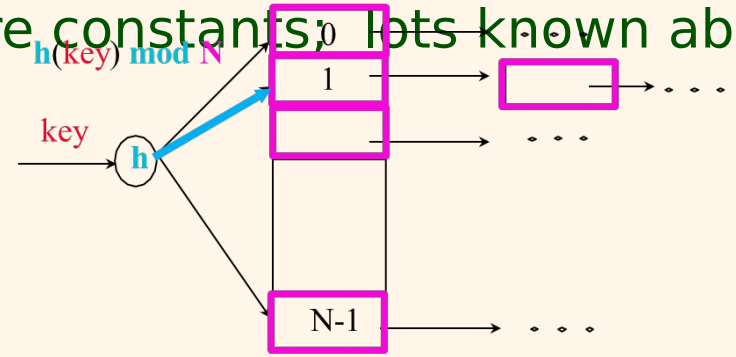
Buckets/blocks contain *Data Entrie*

Data Entry k*
k*blocks

Hash function **h** works on *search* ***key*** field of **Data Record *r*.**

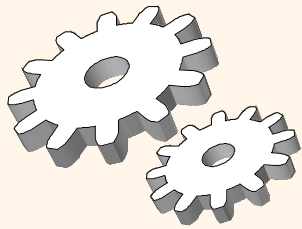
Must distribute values over range 0 ... N-1.

- **h**(***key***) = (a * ***key*** + b) usually works well.
- a and b are constants; lots known abo w to tune h.



~~O(1)~~

Long overflow **Chains** can develop and degrade performance



Dynamic Extendable Hashing Indexes

Dynamic Extendable Hashing Indexes

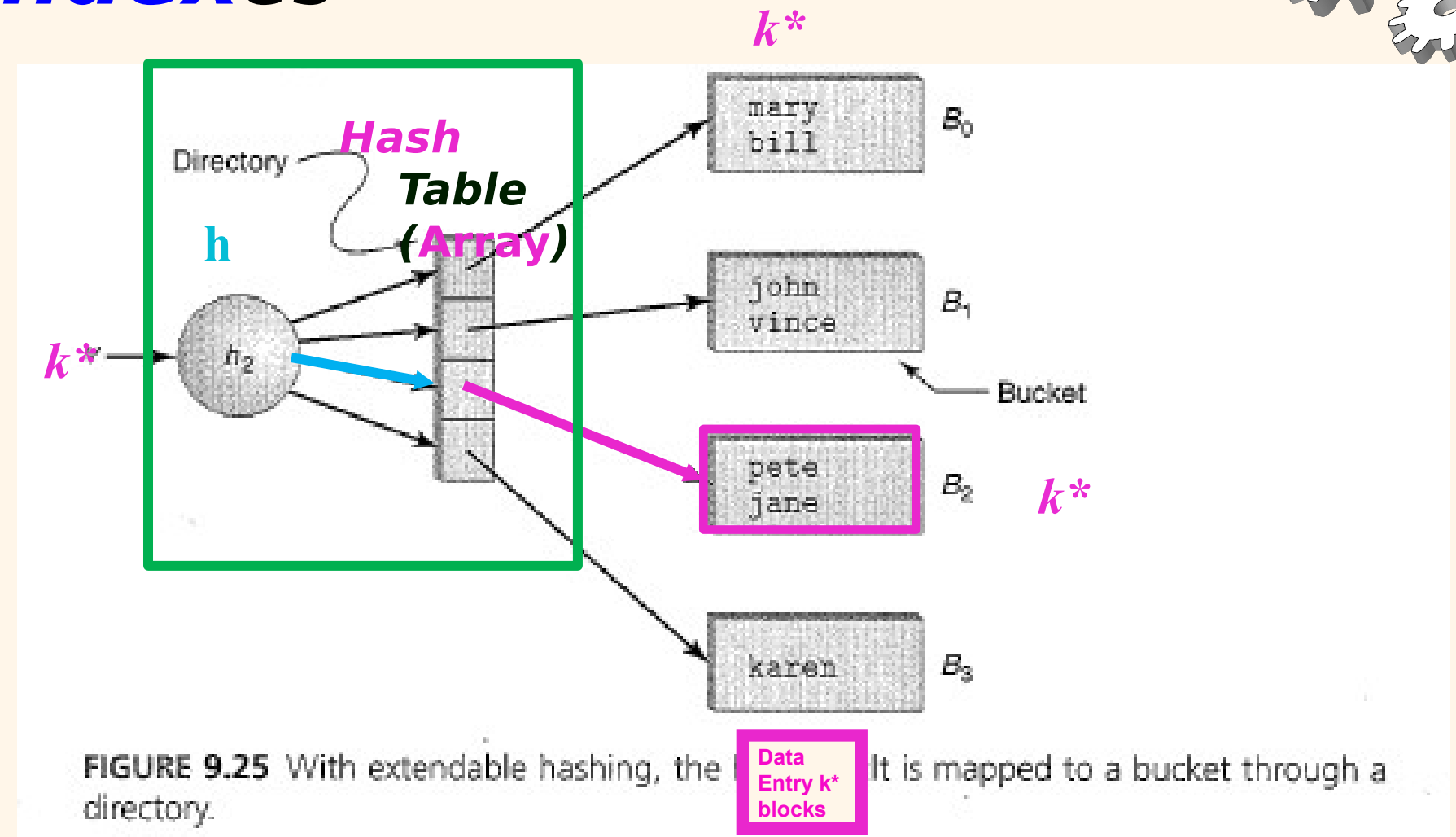
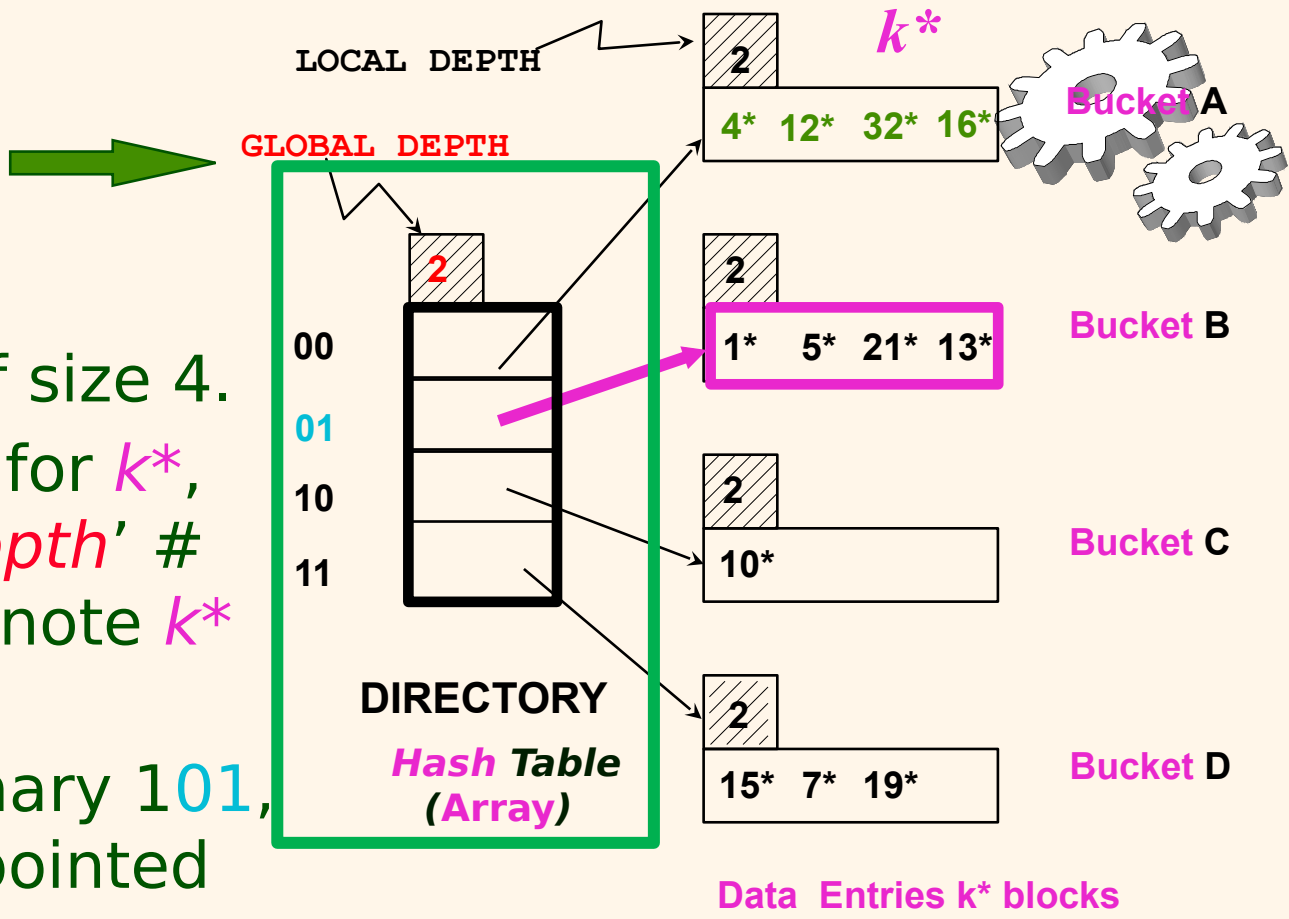


FIGURE 9.25 With extendable hashing, the [illegible] It is mapped to a bucket through a directory.

Example



DIRECTORY is Array of size 4.

To **Search Bucket** for k^*, take last `*global depth*' # bits of **h**(k^*); we denote k^* by **h**(k^*).

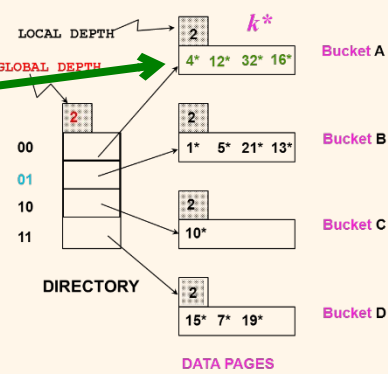
- If **h**(k^*) = 5 = binary 101, it is in **Bucket** pointed to by 01.

sert k^*: If **Bucket** is full, *split* it (*allocate* ***new Page***, *re-distr*

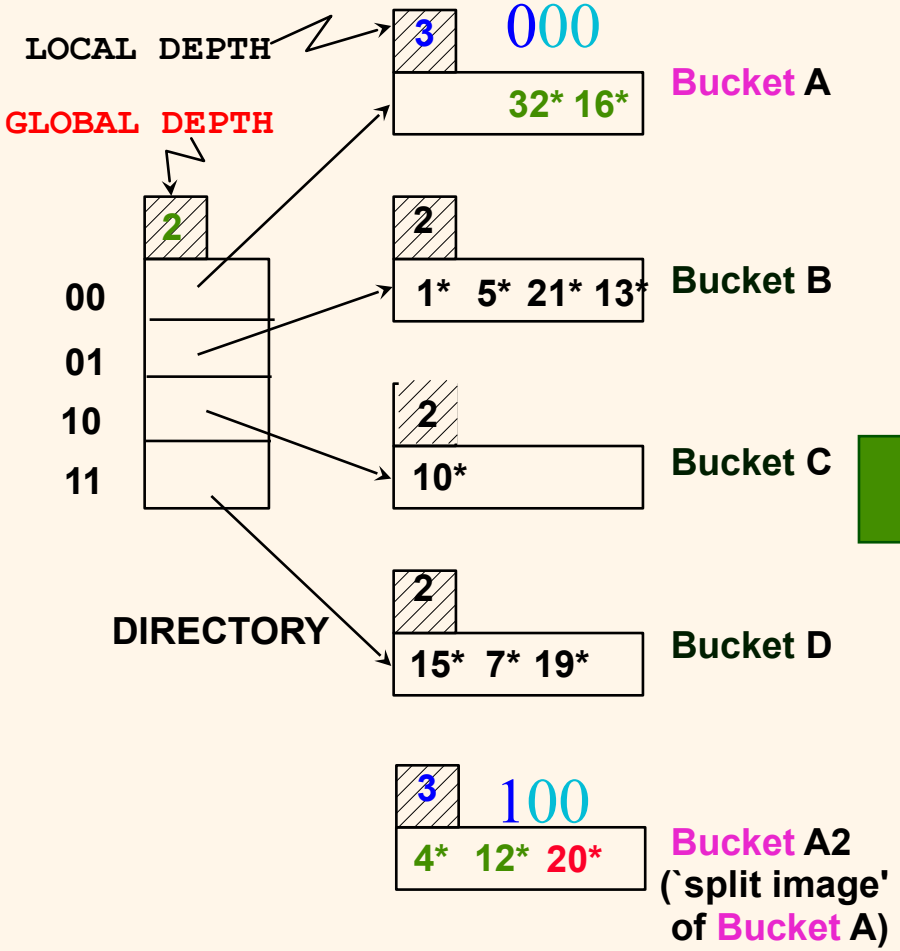
- *If necessary*, double the **DIRECTORY**. (As we will see, **splitting a Bucket** does not always require doubling; we can tell by comparing **GLOBAL DEPTH** with **LOCAL DEPTH** for the

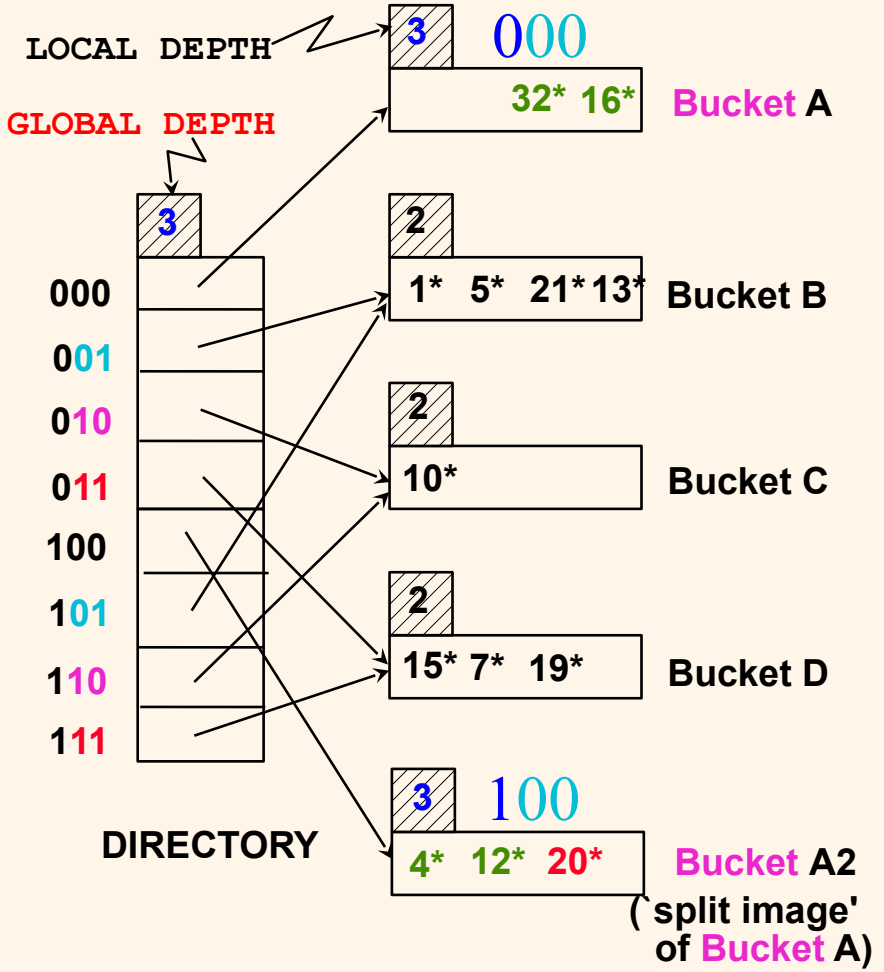
Insert $h(k^*)=20$ (Causes Doubling)

(10100)



Insert: If **Bucket** is full, *split* it (*allocate new Page, re-distribute*).





Dynamic Extendable Hashing Indexes

Situation: **Bucket** (primary **Page**) becomes full.

Why not re-organize **Index** by *doubling* # of **Buckets**?

D per **Page**

- Reading and writing all **blocks** is expensive!

- *Idea*: Use *Directory* ***Hash*** ***Table*** *of pointers (**pid**) to **Buckets**/**blocks**, doubling the Directory, **splitting just the Bucket that overflowed**!*

- *Directory* ***Hash*** ***Table*** much smaller than **File**. so doubling it is muc a
  Entries k* is split.

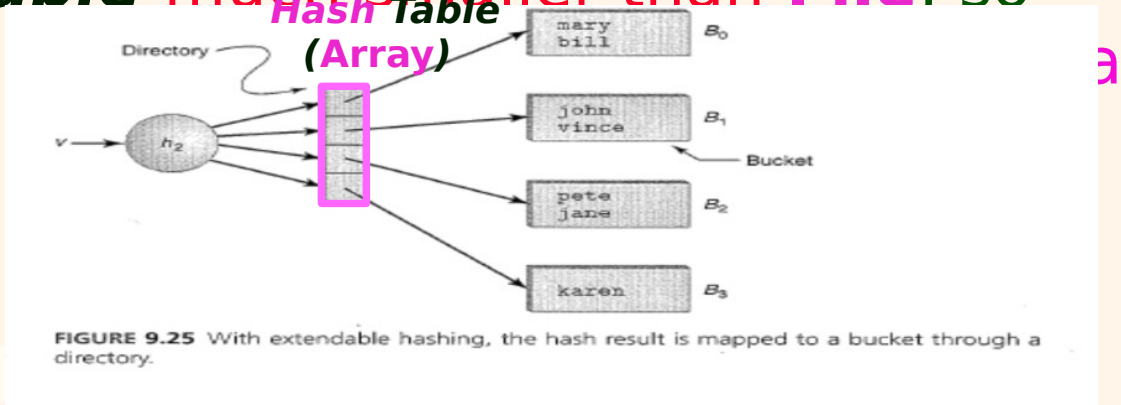
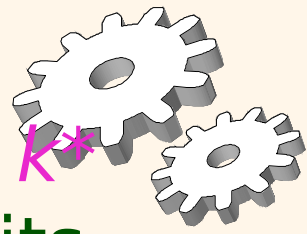


FIGURE 9.25 With extendable hashing, the hash result is mapped to a bucket through a directory.

No overflow Page!

Points to Note

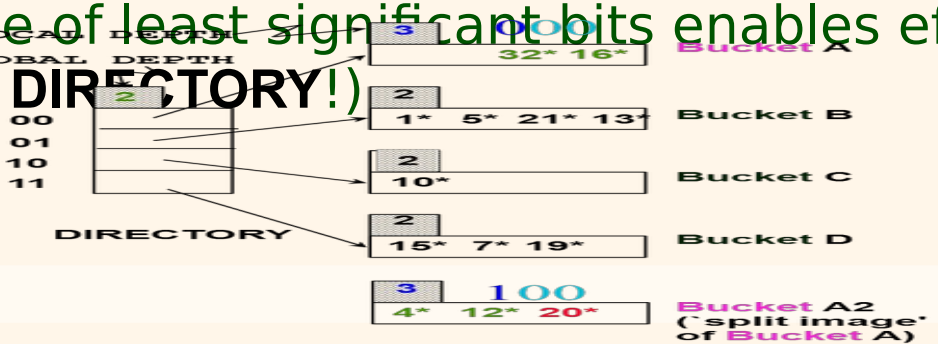


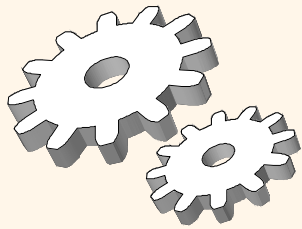
20 = binary 10100. Last **2** bits (00) tell us *k** belongs in **Bucket** A or **Bucket** A2. Last **3** bits needed to tell which.

- **GLOBAL DEPTH** *of* **DIRECTORY**: Max # of bits needed to tell which **Bucket** an Data Entry k* belongs to.
- **LOCAL DEPTH** *of a* ***Bucket***: # of bits used to determine if an Data Entry k* belongs to this **Bucket**.

When does **Bucket** split cause **DIRECTORY** **doubling**

- Before **Insert**, **LOCAL DEPTH** of **Bucket** = **GLOBAL DEPTH**. **Insert** causes **LOCAL DEPTH** to become > **GLOBAL DEPTH**; **DIRECTORY** is doubled by *copying it over* and `fixing' pointer to split image **Page**. (Use of least significant bits enables efficient doubling via copying of **DIRECTORY**!)





Dynamic Linear Hashing Indexes

Dynamic Linear Hashing Indexes

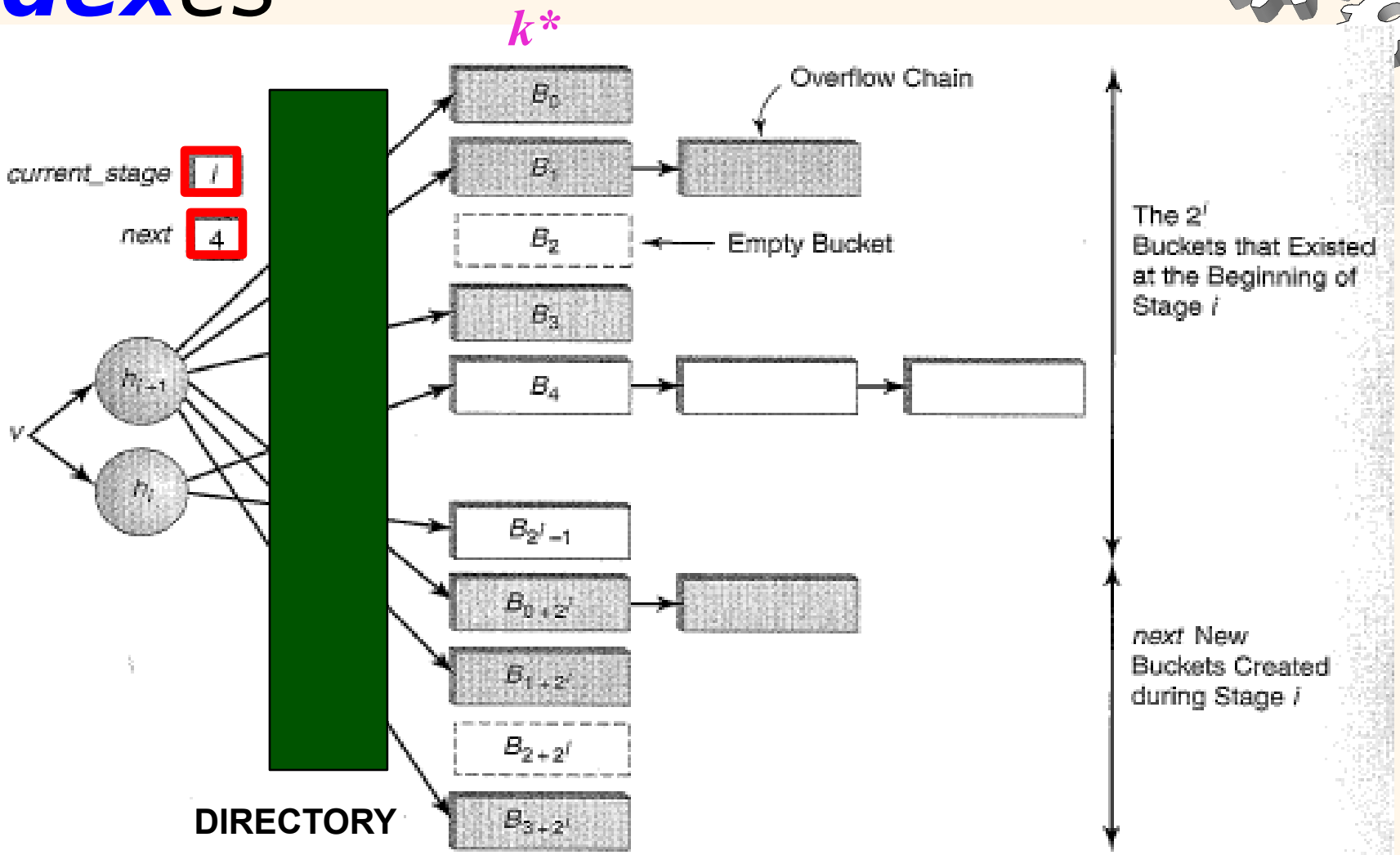


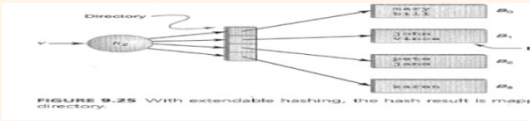
FIGURE 9.28 Linear hashing sp … secutively. The shaded buckets are accessed through h_{i+1}.

What is missing???

What is place of DIRECTORY???

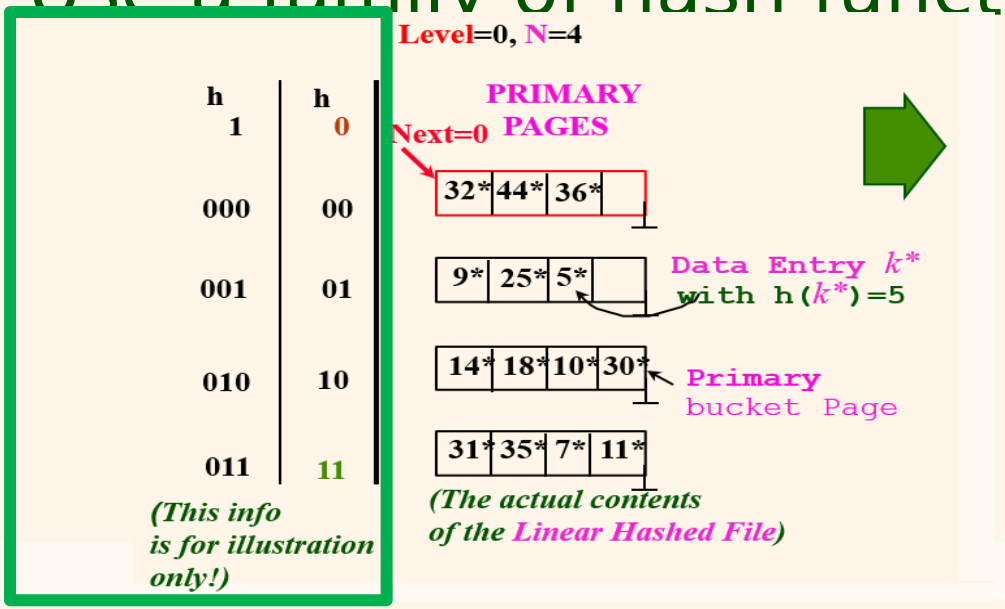
Dynamic Linear Hashing Indexes

This is another ***Dynamic*** **Hashing** scheme, an alternative to ***Dynamic*** **Extendable Hashing Index**es.



LH handles the problem of long overflow chains *without using a* **DIRECTORY**, and handles duplicates.

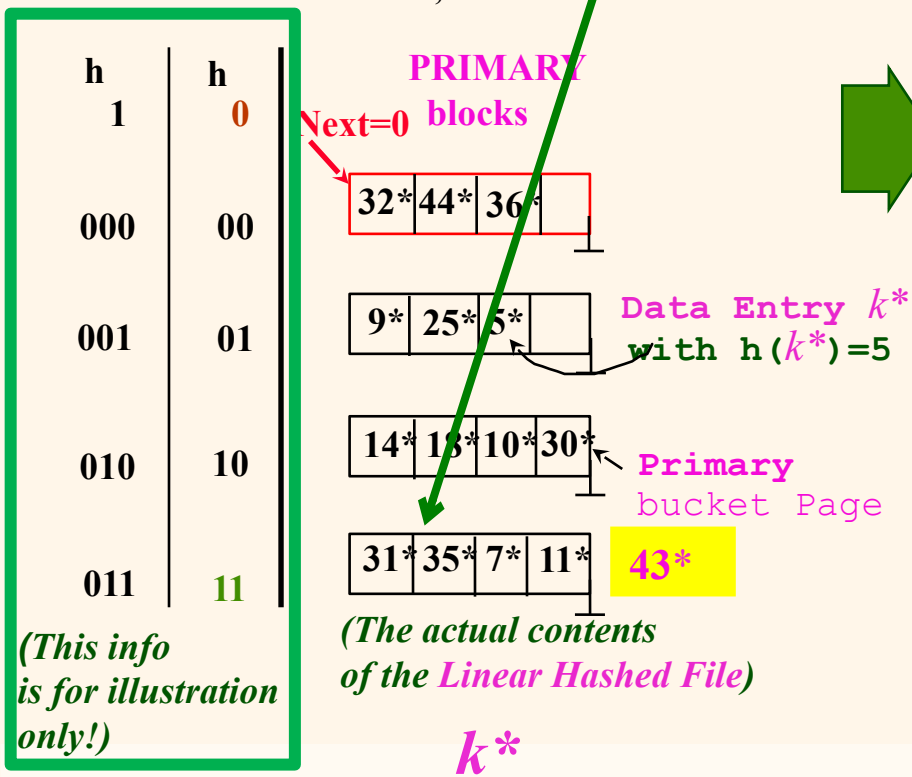
Idea: Use a family of hash functions $\mathbf{h}_0$, $\mathbf{h}_1$, $\mathbf{h}_2$, ...



Example of Dynamic Linear Hashing Indexes (Insert 43*) (101011)

- Insert: Find Bucket by applying h_{Level} / $h_{Level+1}$:
  - If Bucket to insert into is full:
    - Add overflow Page and insert data entry k*.
    - Split *Next* Bucket and increment *Next*.

- On split, $h_{Level+1}$ is used to **re-distribute Data Entries k***.



000

100

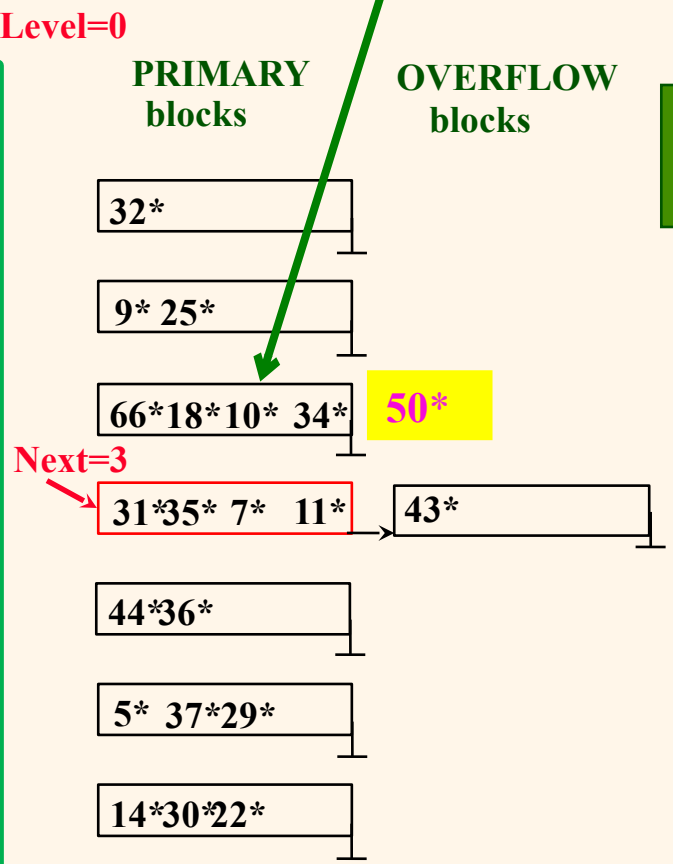
Example: End of a Rour

Insert 50* (110010)

- **Insert**: Find **Bucket** by applying h_{Level} / $h_{Level+1}$:
  - If **Bucket** to insert into is full:
    - Add **overflow Page** and insert data entry k*.
    - Split *Next* **Bucket** and increment *Next*.

Level=0

h_1	h_0
000	00
001	01
010	10
011	11
100	00
101	01
110	10

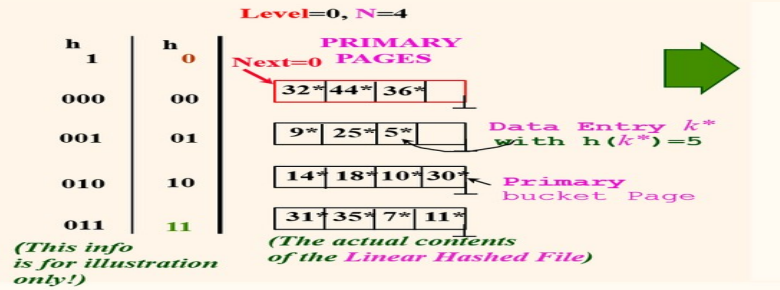


011

111

Dynamic Linear Hashing Indexes

DIRECTORY avoided in **LH** by using overflow **blocks**, and choosing **Bucket**/**Page** to split round-robin.

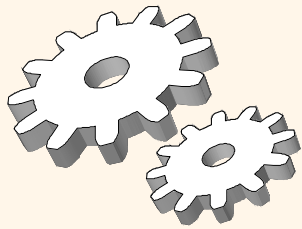


- Splitting proceeds in `Rounds'. `Round' ends when all N_R initial (for round *R*) **Buckets** are split. **Buckets** 0 to *Next-1* (***next***) have been split; *Next* to N_R yet to be split.

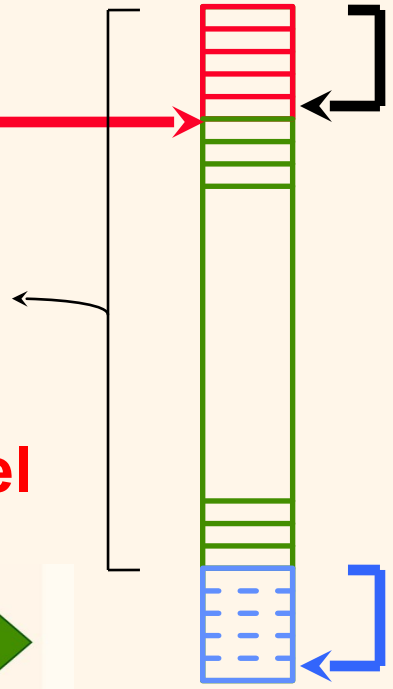
- Current `Round' number is *Level*. (**current_stage**)

- **Search k*:** To find **Bucket** for Data Entry *k**, find $h_{Level}(k^*)$:
  - If $h_{Level}(k^*)$ in range `*Next* to N_R', *k** belongs here.

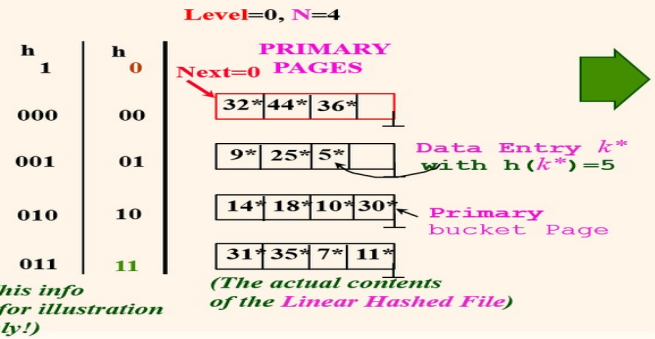
Overview of LH File Indexes



- In the middle of a `Round'.

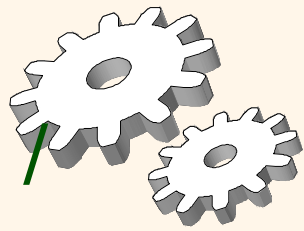


Buckets split in this round:
If h Level (search key value) is in this range, must use h Level+1 (search key value) to decide if entry is in `split image' Bucket.



`split image' Buckets: created (through splitting of other Buckets) in this round

Dynamic Linear Hashing Indexes



- **Insert k*:** Find **Bucket** by applying h_{Level} / $h_{Level+1}$:
  - If **Bucket** to insert into is full:
    - Add **overflow Page** and insert Data Entry k*.
    - Split ***Next* Bucket** and increment ***Next***.

- Since **Buckets** are split **round-robin**, long overflow chains don't develop!

- Doubling of **DIRECTORY** in **Extendable Hashing** is similar; switching of hash fun
how the # of bits examined is

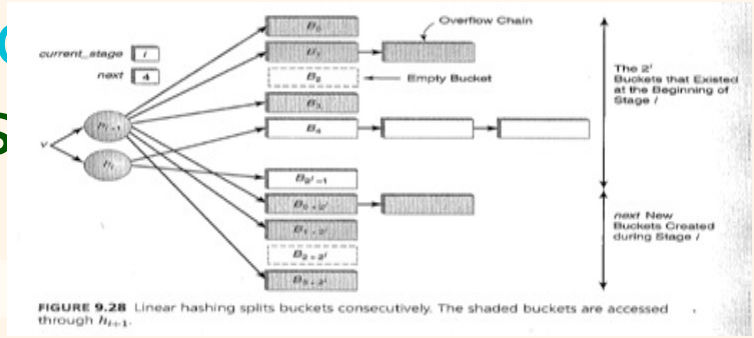


FIGURE 9.28 Linear hashing splits buckets consecutively. The shaded buckets are accessed through h_{i+1}.

LH Described as a Variant of EH

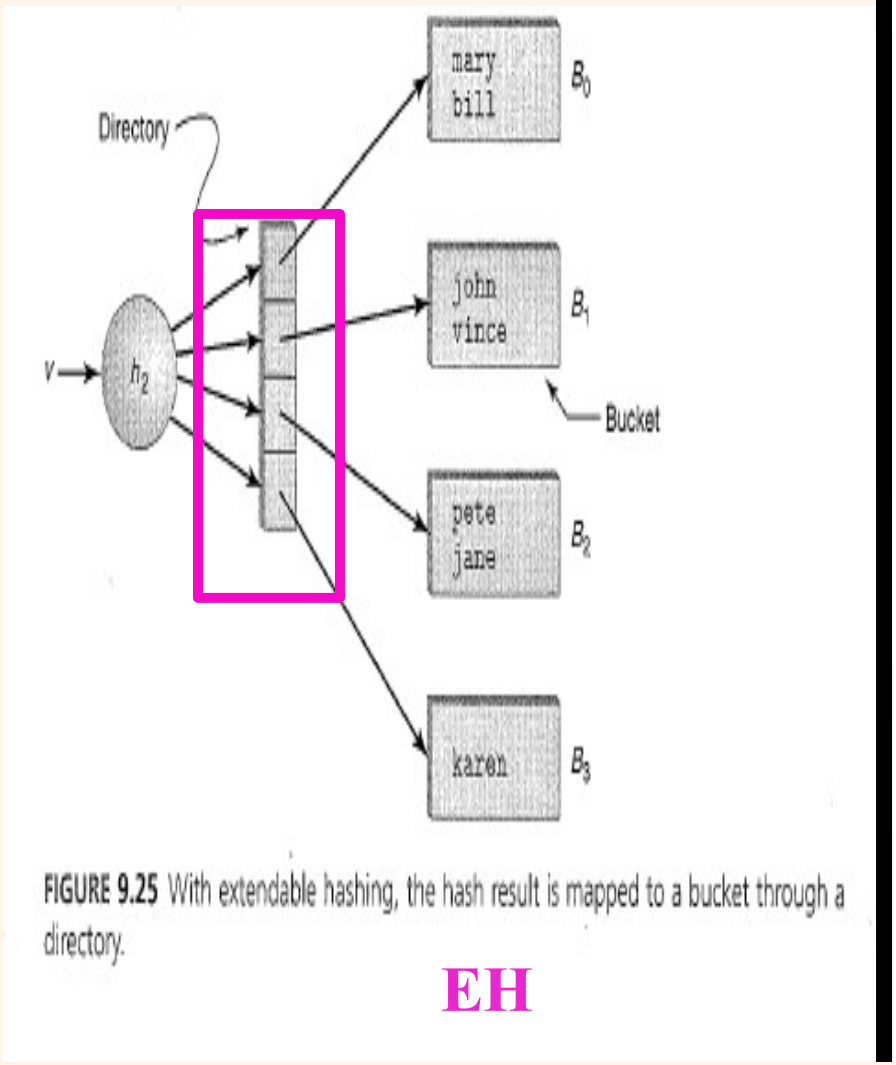


FIGURE 9.25 With extendable hashing, the hash result is mapped to a bucket through a directory.

EH

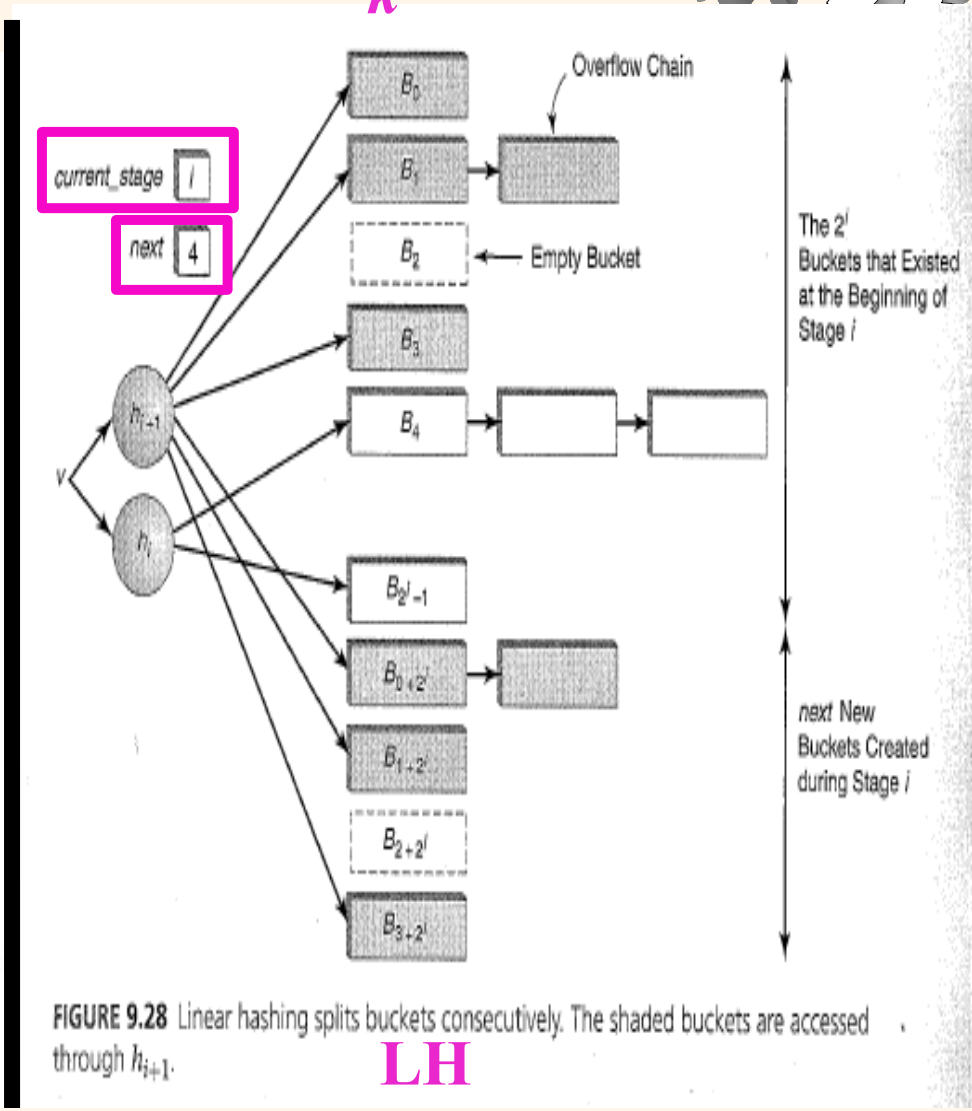
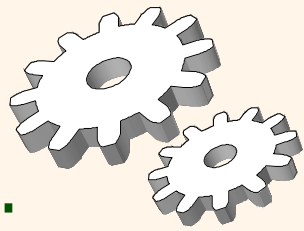


FIGURE 9.28 Linear hashing splits buckets consecutively. The shaded buckets are accessed through h_{i+1}.

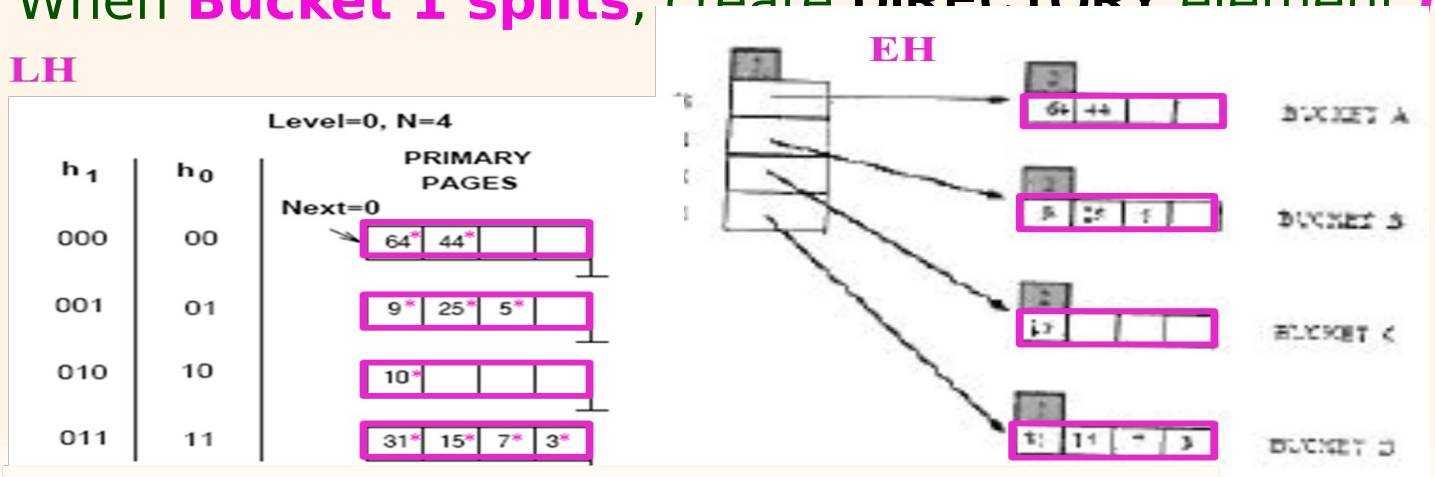
LH

LH Described as a Variant of EH

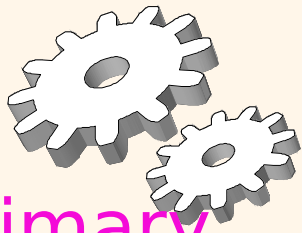


The two schemes are actually quite similar:

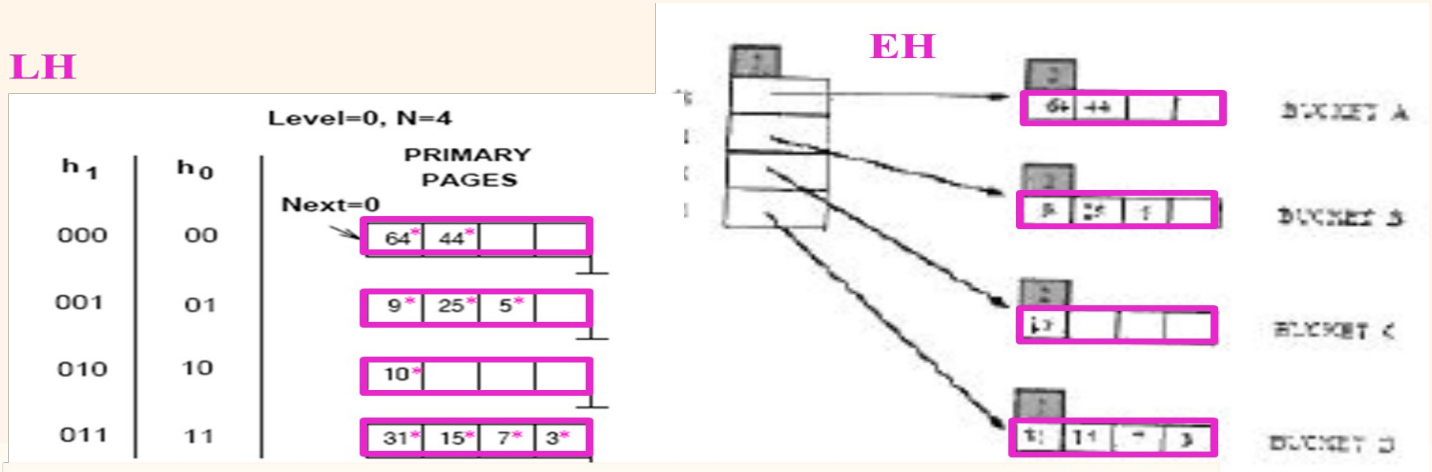
- Begin with an **EH Index** where **DIRECTORY** has ***N*** elements.
- Use overflow **blocks**, split **Buckets** round-robin.
- First split is at **Bucket 0**. (Imagine **DIRECTORY** being doubled at this point.) But elements <1,*N*+1>, <2,*N*+2>, ... are the same. So, need only create **DIRECTORY** element *N*, which differs from 0, now.
  - When **Bucket 1 splits**, create **DIRECTORY** element ***N*+1**, etc.

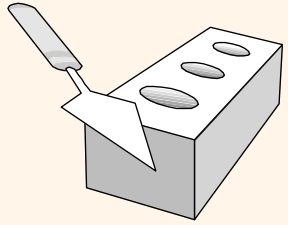


LH Described as a Variant of EH



- So, DIRECTORY can double gradually. Also, primary Bucket blocks are created in order. If they are *allocated* in sequence too (so that finding i'th is easy), **we actually don't need a DIRECTORY**! Voila **LH**.





16. SET 3 - 4:STORAGE IV HASH INDEX 0% 0%

16.1 Other indexes Hidden 0% 0%

Hash indexes

The multi-level index is the most commonly used index type. Several additional index types are used less often but supported by many databases:

- Hash index
- Bitmap index
- Logical index
- Function index

Bitmap indexes

A ***bitmap index*** is a grid of bits:

- Each index row corresponds to a unique table row. If the table's primary key is an integer, the index row number might be the primary key value. Alternatively, the index row number might be an internal table row number, maintained by the database.
- Each index column corresponds to a distinct table value. Ex: If the index is on AirportCode, each index column corresponds to a different three-letter airport code. The mapping of index column numbers to table values is computed with a function or stored in an internal 'lookup' table.

Bitmap indexes contain ones and zeros. 'One' indicates that the table row corresponding to the index row number contains the table value corresponding to the index column number. 'Zero' indicates the row does not contain the value.

To locate rows containing a table value, the database:

1. Determines the index column corresponding to the table value.
2. Reads the index column and finds index rows that are set to 'one'.
3. Determines table rows corresponding to the index rows.
4. Determines pointers to blocks containing the table rows.
5. Reads the blocks containing the table rows.

An efficient bitmap index has two characteristics:

- The database can quickly determine the block containing a table row from the index row number (steps 3 and 4). Ex: The index row number is the hash key for a hash table. The block is determined by applying the hash function to the row number. Ex: The index row number is the table primary key. The block is determined with a primary index.
- The indexed column contains relatively few distinct values, typically tens or hundreds. If the indexed column contains thousands of distinct values, the bitmap index is large and inefficient.

Bitmap indexes with the above characteristics enable fast reads. Ex: A table with 10 million rows has a bitmap index on a column with 100 distinct values. The index contains 125 million bytes (10 million rows × 100 column values × 1 bit per index entry / 8 bits per byte) and can easily be retained in memory.

PARTICIPATION ACTIVITY | 16.1.3: Bitmap index.

bitmap index

	ATL	DUB	LAX	MSN	ORD	SEA	SJC
0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0
4	0	0	0	1	0	0	0
5	1	0	0	0	0	0	0
6	0	0	0	0	0	1	0
7	0	0	0	1	0	0	0
8	0	0	0	0	1	0	0
9	1	0	0	0	0	0	0
10	0	0	1	0	0	0	0
11	0	0	0	0	0	0	1

table

				blocks
0	666	United Airlines	6:00	ATL	Airbus 321
1	140	Aer Lingus	10:30	DUB	Boeing 737
2	803	Air China	3:20	LAX	Boeing 777
3	801	Air India	3:15	MSN	Boeing 747
4	552	Lufthansa	18:00	MSN	MD 111
5	1100	Air China	3:20	ATL	Boeing 787
6	971	Air India	19:35	SEA	Boeing 747
7	702	American Airlines	5:05	MSN	Airbus 323
8	1154	American Airlines	13:50	ORD	MD 111
9	799	British Airways	13:50	ATL	Airbus 323
10	1222	Delta Airlines	8:25	LAX	Boeing 747
11	1001	Southwest Airlines	5:05	SJC	Airbus 323

PARTICIPATION ACTIVITY 16.1.3: Bitmap index.

bitmap index

	ATL	DUB	LAX	MSN	ORD	SEA	SJC
0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0
4	0	0	0	1	0	0	0
5	1	0	0	0	0	0	0
6	0	0	0	0	0	1	0
7	0	0	0	1	0	0	0
8	0	0	0	0	1	0	0
9	1	0	0	0	0	0	0
10	0	0	1	0	0	0	0
11	0	0	0	0	0	0	1

table

blocks

0	686	United Airlines	6:00	ATL	Airbus 321
1	140	Aer Lingus	10:30	DUB	Boeing 737
2	803	Air China	3:20	LAX	Boeing 777
3	801	Air India	3:15	MSN	Boeing 747
4	552	Lufthansa	18:00	MSN	MD 111
5	1100	Air China	3:20	ATL	Boeing 787
6	971	Air India	19:35	SEA	Boeing 747
7	702	American Airlines	5:05	MSN	Airbus 323
8	1154	American Airlines	13:50	ORD	MD 111
9	799	British Airways	13:50	ATL	Airbus 323
10	1222	Delta Airlines	8:25	LAX	Boeing 747
11	1001	Southwest Airlines	5:05	SJC	Airbus 323

PARTICIPATION ACTIVITY 16.1.4: Bitmap indexes.

1) Refer to the bitmap index in the above animation. The three 1's in the MSN column means:

- ○ The value MSN appears in three table blocks.
- ◉ The value MSN appears in three table rows.
- ○ The value MSN appears in three buckets.

Correct

Each 1 in the MSN column indicates MSN appears in a distinct row. Since the MSN column contains three 1's, the value MSN appears in three rows.

?????

TA time (Jordan)
(CA 16.1.1 Step 2 – Bitmap Index)

CHALLENGE ACTIVITY | 16.1.1: Other indexes.

Bitmap Index

bitmap index

	LAX	DUB	MSN	SEA	ORD	ATL	SJC
0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	1
2	0	0	0	0	0	1	0
3	0	1	0	0	0	0	0
4	0	0	0	1	0	0	0
5	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0
8	0	0	1	0	0	0	0
9	0	1	1	0	0	0	0
10	1	0	0	0	0	0	1
11	0	0	0	0	0	1	0

row 3

row 9

Table row number(s) containing DUB: Ex: 3, 9 Enter as comma-separated list.

2

VH

?????

COSC 3380 23578 M & W 4 to 5:30 PM LECTURE (in TEAMS from 3:50 to 5:15 PM)

01:00:26

Pop out | Chat | People 114 | Raise | View | Rooms | Apps | More | Camera | Mic | Share | Leave

Yu, Jordan T

View all

zyBooks My library > COSC 3380: Database Systems home > 16.1: Other indexes

zyBooks catalog | Help/FAQ | Jordan Yu

Feedback?

CHALLENGE ACTIVITY 16.1.1: Other indexes

Jump to level 1

bitmap index

	ORD	MSN	SEA	ATL	DUB	SJC	LAX
0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0
2	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	1	0	1	1	0	1
6	0	0	0	0	0	0	0
7	0	0	0	0	1	0	0
8	1	0	1	1	0	0	0
9	1	0	0	0	0	0	0
10	0	0	0	0	1	0	0
11	0	0	0	0	0	0	0

Table row number(s) containing LAX: 2, 5 Enter as comma-separated list.

Check Next

Expected: 2, 5

In the LAX column, each 1 indicates the table row number containing LAX.

LAX has a 1 in row 2, so 2 is a table row number containing LAX. Same for 5.

View solution (Instructors only)

Yu, Jordan T

Participants

Invite someone or dial a number

Share invite

In this meeting (114) Mute all

- Hilford, Victoria — Organizer
- RA Adhikari, Rohit
- MA Ahmed, Mohamed A
- AA Akram, Ali
- BA Akukwe, Benetta O
- SA Alsayed, Sami H
- SA Altaf, Sameer
- SA Alvarez, Stephanie
- OA Anayor-Achu, Ogochukwu E
- HA Avci, Hatice Kubra
- RA Aysola, Riya
- AB Bahl, Anish

Hash indexes

The multi-level index is the most commonly used index type. Several additional index types are used less often but supported by many databases:

- Hash index
- Bitmap index
- Logical index
- Function index

Logical indexes

A single- or multi-level index normally contains pointers to table blocks and is called a ***physical index***.

A ***logical index*** is a single- or multi-level index in which pointers to table blocks are replaced with primary key values. Logical indexes are always secondary indexes and require a separate primary index on the same table. To locate a row containing a column value, the database:

1. Looks up the column value in the logical index to find the primary key value.
2. Looks up the primary key value in the primary index to find the table block pointer.
3. Reads the table block containing the row.

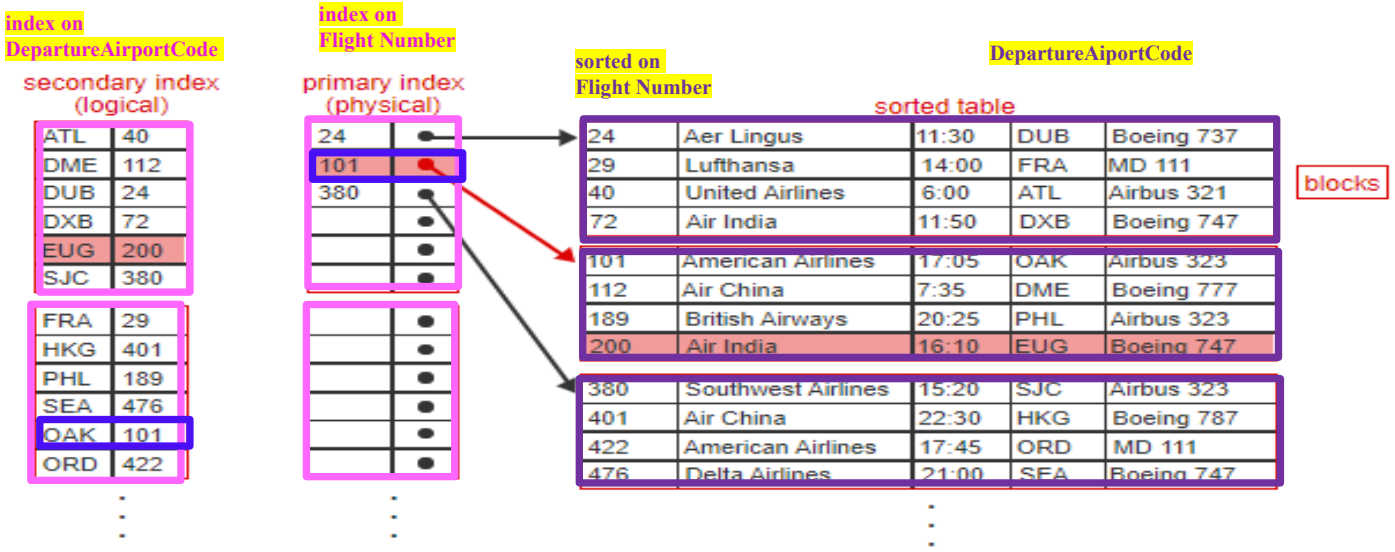
Logical indexes change only when primary key values are updated, which occurs infrequently. Physical indexes change whenever a row moves to a new block, which occurs in several ways:

- **A row is inserted into a full block**. To create space for the new row, the block splits and some rows move to a new block.
- **A clustering column is updated**. When a clustering column is updated, the row may move to a new block to maintain sort order.
- **The table is reorganized**. Occasionally, a database administrator may physically reorganize a table to recover deleted space or order blocks contiguously on magnetic disk.

If a table has several indexes, the time required to update physical indexes is significant, and logical indexes are more efficient.

On read queries, a logical index requires an additional read of the primary index and is slower than a physical index. However, the primary index is often retained in memory, mitigating the cost of the additional read.

PARTICIPATION ACTIVITY | 16.1.5: Logical index.



Logical indexes

A single- or multi-level index normally contains pointers to table blocks and is called a ***physical index***.

A ***logical index*** is a single- or multi-level index in which pointers to table blocks are replaced with primary key values. Logical indexes are always secondary indexes and require a separate primary index on the same table. To locate a row containing a column value, the database:

1. Looks up the column value in the logical index to find the primary key value.
2. Looks up the primary key value in the primary index to find the table block pointer.
3. Reads the table block containing the row.

Logical indexes change only when primary key values are updated, which occurs infrequently. Physical indexes change whenever a row moves to a new block, which occurs in several ways:

- **A row is inserted into a full block**. To create space for the new row, the block splits and some rows move to a new block.
- **A clustering column is updated**. When a clustering column is updated, the row may move to a new block to maintain sort order.
- **The table is reorganized**. Occasionally, a database administrator may physically reorganize a table to recover deleted space or order blocks contiguously on magnetic disk.

If a table has several indexes, the time required to update physical indexes is significant, and logical indexes are more efficient.

On read queries, a logical index requires an additional read of the primary index and is slower than a physical index. However, the primary index is often retained in memory, mitigating the cost of the additional read.

PARTICIPATION ACTIVITY 16.1.6: Logical indexes.

	An index on a unique sort column.
	An index on a non-sort column.
Logical index	An index with primary key values rather than block pointers. A logical index contains primary index values rather than table block pointers. Usually the primary index is on the primary key of the table, so logical indexes usually contain primary key values.
	An index with table block pointers.

?????

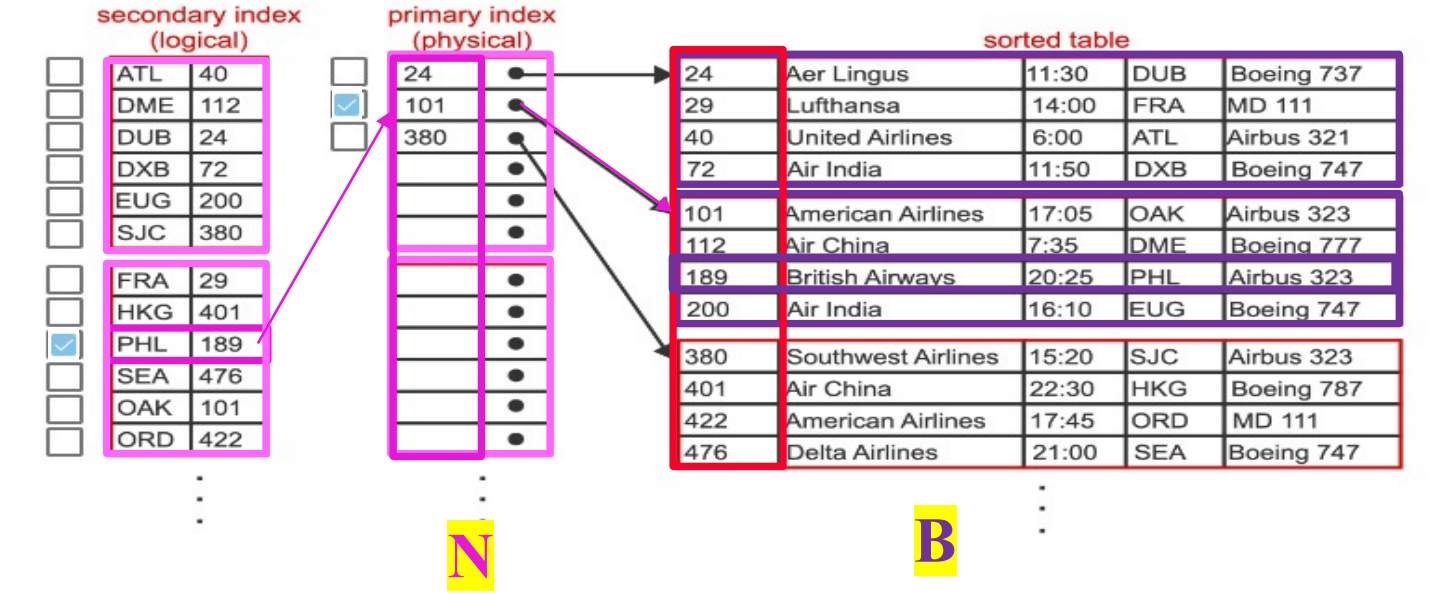
TA time (Jordan)

(CA 16.1.1 Step 3 – Logical Index)

CHALLENGE ACTIVITY | 16.1.1: Other indexes.

Logical Index

The values in the primary index and the table are sorted on flight number. Select the indexes to find the row containing PHL.



N

B

3

VH

?????

COSC 3380 23578 M & W 4 to 5:30 PM LECTURE (in TEAMS from 3:50 to 5:15 PM)

01:07:22

Pop out | Chat | People 113 | Raise | View | Rooms | Apps | More | Camera | Mic | Share | Leave

Yu, Jordan T

View all

zyBooks My library > COSC 3380: Database Systems home > 16.1: Other indexes

zyBooks catalog | Help/FAQ | Jordan Yu

CHALLENGE ACTIVITY 16.1.1: Other indexes.

Jump to level 1

The values in the primary index and the table are sorted on flight number. Select the indexes to find the row containing DME.

secondary index (logical)

ATL	40
DME	112
DUB	24
DXB	72
EUG	200
SJC	380
FRA	29
HKG	401
PHL	189
SEA	476
OAK	101
ORD	422

primary index (physical)

24	●
101	●
380	●

sorted table

24	Aer Lingus	11:30	DUB	Boeing 737
29	Lufthansa	14:00	FRA	MD 111
40	United Airlines	6:00	ATL	Airbus 321
72	Air India	11:50	DXB	Boeing 747
101	American Airlines	17:05	OAK	Airbus 323
112	Air China	7:35	DME	Boeing 777
189	British Airways	20:25	PHL	Airbus 323
200	Air India	16:10	EUG	Boeing 747
380	Southwest Airlines	15:20	SJC	Airbus 323
401	Air China	22:30	HKG	Boeing 787
422	American Airlines	17:45	ORD	MD 111
476	Delta Airlines	21:00	SEA	Boeing 747

Check | Next

✔ Expected: Secondary index 112, primary index 101

The values in both the primary index and the table are sorted on flight number.

First, the database searches the secondary index to find the primary key for DME, which is 112.
Next, the database searches the primary index for the largest key that is ≤ primary key, which is 101.
The table block pointer points to the table block that contains the row.

View solution (Instructors only)

teams.microsoft.com is sharing your screen. Stop sharing | Hide

Yu, Jordan T

65°F Cloudy

Participants

Invite someone or dial a number

Share invite

In this meeting (113) | Mute all

- Hilford, Victoria — Organizer
- RA Adhikari, Rohit
- MA Ahmed, Mohamed A
- AA Akram, Ali
- BA Akukwe, Benetta O
- SA Alsayed, Sami H
- SA Alvarez, Stephanie
- OA Anayor-Achu, Ogochukwu E
- HA Avci, Hatice Kubra
- RA Aysola, Riya
- AB Bahl, Anish
- SB Banza, Sean Paolo B

Hash indexes

The multi-level index is the most commonly used index type. Several additional index types are used less often but supported by many databases:

- Hash index
- Bitmap index
- Logical index
- Function index

Function indexes

In some cases, values specified in a WHERE clause may be in a different format or units than values stored in the column. Ex:

- The WHERE clause specifies values in upper case, but the column contains mixed upper and lower case characters.
- The WHERE clause specifies values as percentages, from 0 to 100, but the column contains values from 0 to 1.

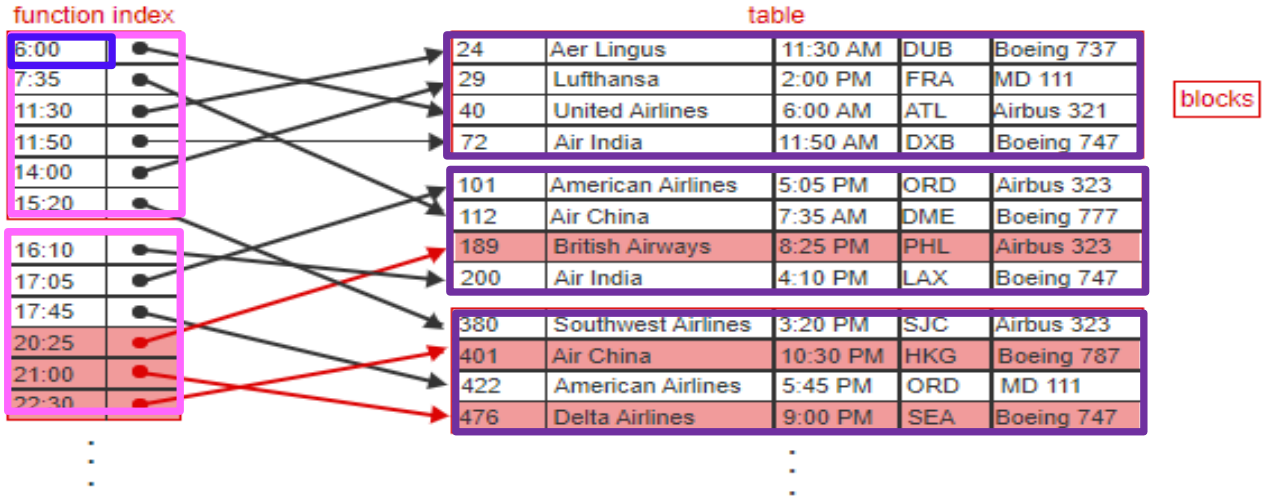
In the above examples, index entries do not match values in the WHERE clause, so the database cannot use the index to execute the query.

To address this problem, some databases support function indexes. In a ***function index***, the database designer specifies a function on the column value. Index entries contain the result of the function applied to column values, rather than the column values.

Ex: Column values are stored as decimal numbers between 0 and 1, but users specify percentages as integers between 0 and 100 in queries. The database designer specifies a function index that multiplies column values by 100 and converts the result to an integer. The index contains integers between 0 and 100, so the database can use the function index to process queries.

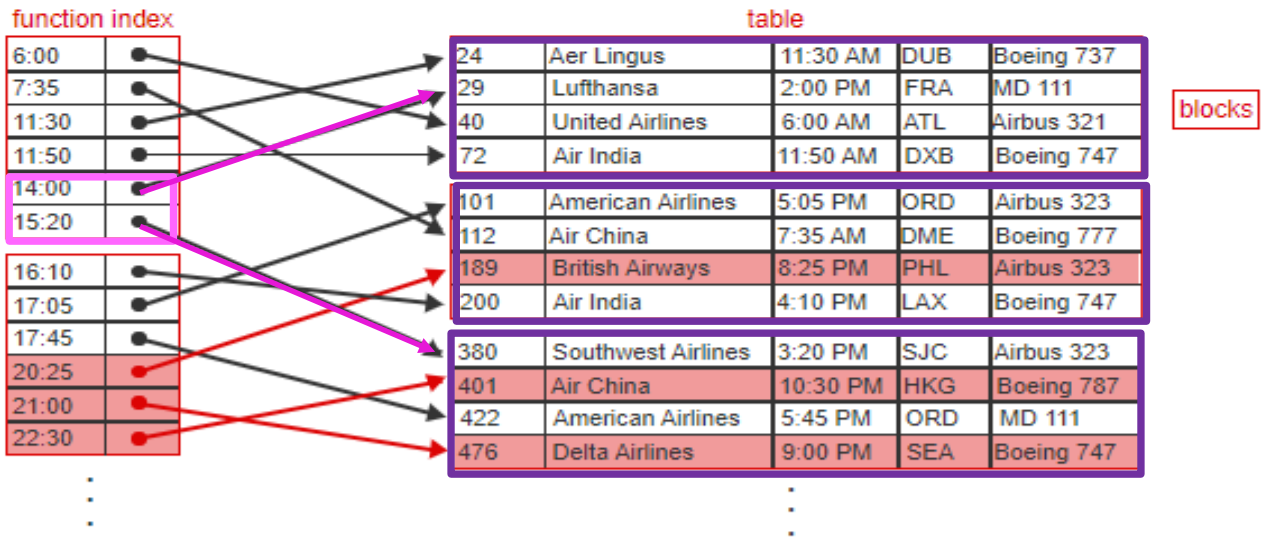
In principle, functions can be used with any index type, including single-level, multi-level, hash, bitmap, and logical indexes. In practice, support varies by database.

PARTICIPATION ACTIVITY 16.1.7: Function index.




```
SELECT FlightNumber, AirlineName
FROM Flight
WHERE DepartureTime > "20:00"
```

PARTICIPATION ACTIVITY | 16.1.7: Function index.




```
SELECT FlightNumber, AirlineName
FROM Flight
WHERE DepartureTime > "20:00"
```

PARTICIPATION ACTIVITY | 16.1.8: Function indexes.

1) Refer to the animation above. Which table blocks does the following query read?

```
SELECT FlightNumber, AirlineName
FROM Flight
WHERE DepartureTime > "12:00" AND DepartureTime < "16:00";
```

- ○ Block 0
- ○ Block 1
- ◉ Blocks 0 and 2

Correct

The query selects departure times of 14:00 and 15:20 from the index. 14:00 refers to flight 29 in block 0, and 15:20 refers to flight 380 in block 2.

?????

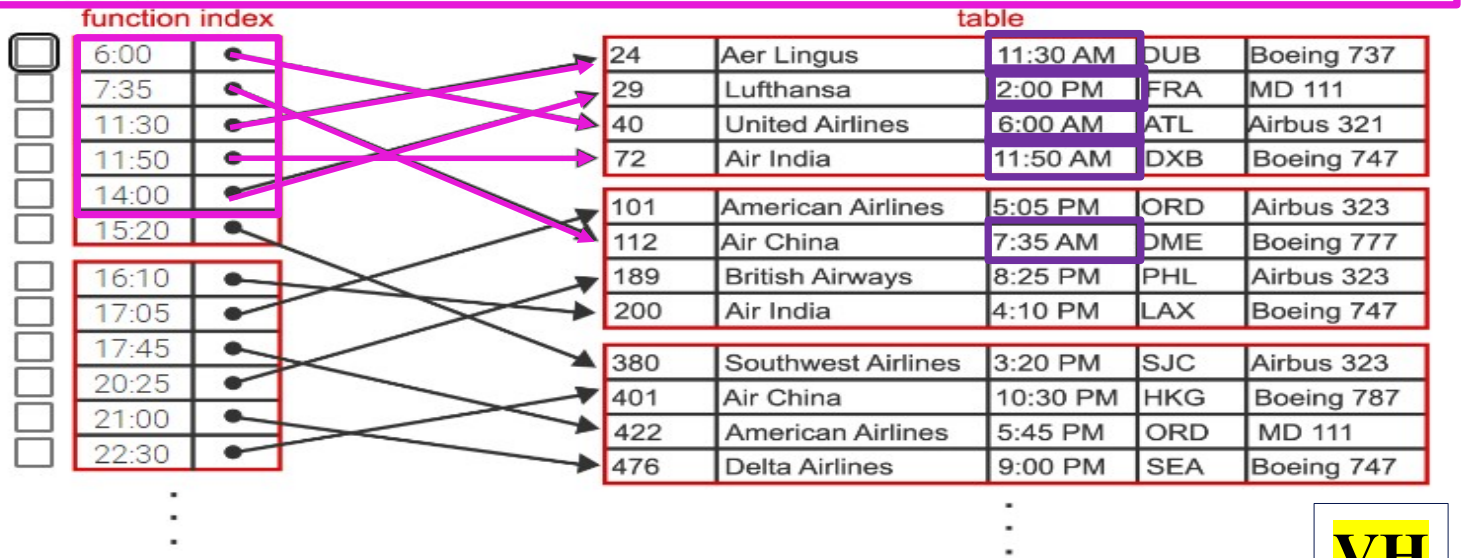
TA time (Jordan)
(CA 16.1.1 Step 4 – Function Index)

CHALLENGE ACTIVITY | 16.1.1: Other indexes.

Function Index

The table contains times in AM/PM format. The function index converts AM/PM format to 24-hour format.

Select the indexes that match the expression: DepartureTime < "15:00", given that the expression specifies time in 24-hour format.



VH

?????

4

COSC 3380 23578 M & W 4 to 5:30 PM LECTURE (in TEAMS from 3:50 to 5:15 PM)

01:14:30

Pop out | Chat | People 113 | Raise | View | Rooms | Apps | More | Camera | Mic | Share | Leave

Yu, Jordan T

View all

Participants

Invite someone or dial a number

Share invite

In this meeting (113) Mute all

- Hilford, Victoria — Organizer
- RA Adhikari, Rohit
- MA Ahmed, Mohamed A
- AA Akram, Ali
- BA Akukwe, Benetta O
- SA Alsayed, Sami H
- SA Alvarez, Stephanie
- OA Anayor-Achu, Ogochukwu E
- HA Avci, Hatice Kubra
- RA Aysola, Riya
- AB Bahl, Anish
- SB Banza, Sean Paolo B

zyBooks My library > COSC 3380: Database Systems home > 16.1: Other indexes

zyBooks catalog | Help/FAQ | Jordan Yu

Feedback?

CHALLENGE ACTIVITY 16.1.1: Other indexes.

Jump to level 1

The table contains times in AM/PM format. The function index converts AM/PM format to 24-hour format.

Select the indexes that match the expression: DepartureTime > '17:30', given that the expression specifies time in 24-hour format.

function index

6:00	
7:35	
11:30	
11:50	
14:00	
15:20	
16:10	
17:05	
17:45	✓
20:25	✓
21:00	✓
22:30	✓

table

24	Aer Lingus	11:30 AM	DUB	Boeing 737
29	Lufthansa	2:00 PM	FRA	MD 111
40	United Airlines	6:00 AM	ATL	Airbus 321
72	Air India	11:50 AM	DXB	Boeing 747
101	American Airlines	5:05 PM	ORD	Airbus 323
112	Air China	7:35 AM	DME	Boeing 777
189	British Airways	8:25 PM	PHL	Airbus 323
200	Air India	4:10 PM	LAX	Boeing 747
380	Southwest Airlines	3:20 PM	SJC	Airbus 323
401	Air China	10:30 PM	HKG	Boeing 787
422	American Airlines	5:45 PM	ORD	MD 111
476	Delta Airlines	9:00 PM	SEA	Boeing 747

Check | Next | Done. Click any level to practice more. Completion is preserved.

✓ Expected: 17:45, 20:25, 21:00, and 22:30

The departure times 17:45, 20:25, 21:00, and 22:30 match the expression DepartureTime > '17:30'.

View solution (Instructors only)

Stop sharing | Hide

Feedback?

Yu, Jordan T

Hash Index Practice Questions

16. SET 3 - 4:STORAGE IV HASH INDEX

Hash index entries are assigned to _____.

a. values

b. clusters

c. buckets

d. blocks

????

16. SET 3 - 4:STORAGE IV HASH INDEX

What are two characteristics of an efficient bitmap index?

a. Each bucket initially has one block. Any additional blocks are allocated and linked to the initial block.

b. The WHERE clause may specify any values in upper case. The column used in a WHERE clause may contain mixed upper and lower case characters.

c. The database can quickly determine the block containing a table row from the index row number. Any indexed column contains relatively few distinct values.

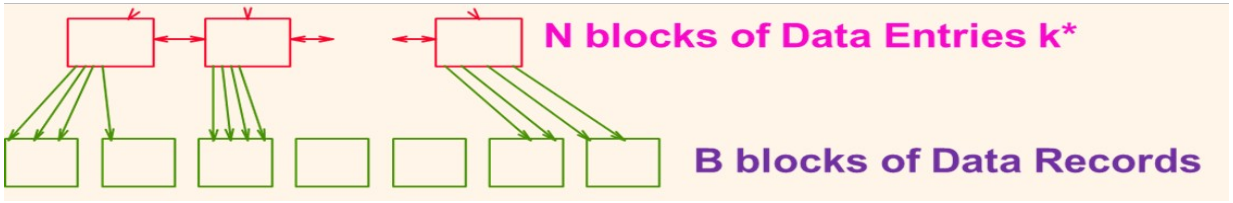
d. Indexes change only when primary key values are updated. Physical indexes change whenever a row moves to a new block.

????

At 5:00 PM .

03.27.2024 (19 - We)	ZyBook SET 3- 4	Set 3 LECTURE 15 STORAGE IV HASH INDEX

HASH



VH work on
SET 3 – 16

Next

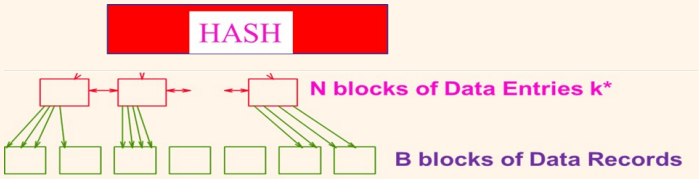
04.01.2024 (20 - Mo)			EXAM 3 Practice (PART of 20 points)	

☐	12. SET 3	Empty		
☐	13. SET 3 - 1:STORAGE I	Hidden	0%	0%
☐	14. SET 3: 2:STORAGE II	Hidden		0%
☐	15. SET 3 - 3:STORAGE III B TREE INDEX	Hidden	0%	0%
☐	16. SET 3 - 4:STORAGE IV HASH INDEX	Hidden	0%	0%

From 5:05 to 5:15 PM – 5 minutes.

03.27.2024 (19 - We)	ZyBook SET 3- 4	Set 3 LECTURE 15 STORAGE IV HASH INDEX

CLASS PARTICIPATION 20 points — 20% of Total



HASH INDEX

Class 19 END PARTICIPATION
Not available until Mar 27 at 5:05pm | Due Mar 27 at 5:15pm | 100 pts

VH, publish

This is a synchronous online class.

Attendance is required.

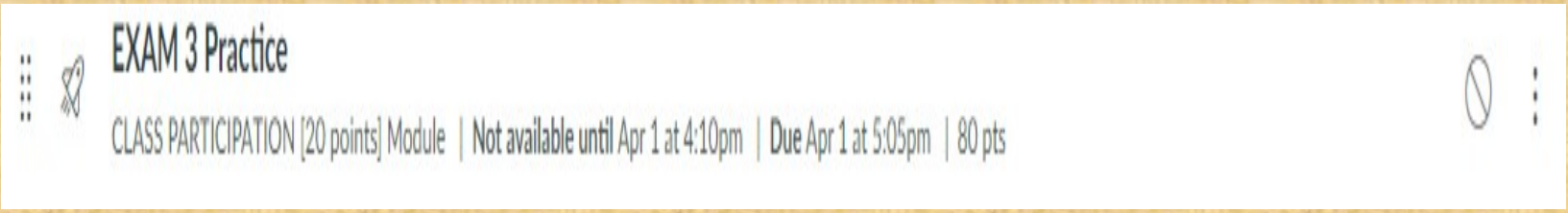
Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

At 5:15 PM.

End Class 19

VH, Download Attendance Report
Rename it:
3.27.2024 Attendance Report FINAL TEAMS



VH, upload Class 19 to CANVAS.