CS 61:
Database Systems

Data persistence, file organization, indexing

Adapted from Silberschatz, Korth, and Sundarshan unless otherwise noted
Augmented by Claude Opus

Big picture: find a needle in a big data haystack quickly

Find data on Morris Park Bake Shop in large database quickly…

But Morris Park Bake Shop is just one entry in large data set

What would we do in CS10 to quickly find data?

Can’t we just use a hash table?

As we saw in CS10, hash tables provide expected O(1) performance
Why not just use a hash table?

Some in-memory database like Redis do store data in hash tables

They typically persist data to disk periodically

Problems

- Persistence: data ends when program ends
- Durability: if server crashes, data is gone
- Scale: can not exceed RAM
- Concurrent access: multiple users cannot safely access it concurrently

We will solve these problems by storing data on disk
We will soon call writing to disk “durability”

Agenda

1. Data persistence
2. Database file organization
3. Indexing

What does persistence mean?

What does persistence mean?

- Data outlives process that created it
- Without persistence, data is just a cache

Key question: If I insert and then pull the plug, will my data be lost?

Where did my data go?

SELECT @@datadir; -- shows where files are kept
On my Mac: /usr/local/mysql/data/

From terminal
$ ls -la /usr/local/mysql/data

```
...
drwxr-x---   34 _mysql  _mysql      1088 Apr 30 17:22 #innodb_redo
drwxr-x---   12 _mysql  _mysql       384 Mar 12 11:54 #innodb_temp
...
drwxr-x---    8 _mysql  _mysql       256 Dec  4 13:44 mysql
-rw-r-----    1 _mysql  _mysql  28311552 May 15 11:39 mysql.ibd
-rw-r-----    1 _mysql  _mysql       131 Dec  4 13:44 mysql_upgrade_history
-rw-r-----    1 _mysql  _mysql    115903 May  6 19:01 mysqld.local.err
-rw-r-----    1 _mysql  _mysql         4 Mar 12 11:54 mysqld.local.pid
-rw-r--r--    1 root    _mysql        48 Dec  4 13:44 mysqld.my
drwxr-x---   12 _mysql  _mysql       384 May  4 08:25 nyc_data
...
-rw-r-----    1 _mysql  _mysql  50331648 May 15 11:24 undo_001
-rw-r-----    1 _mysql  _mysql  33554432 May 15 11:41 undo_002
```

Temp and redo logs

Redo log: recovers committed data after a crash

Undo log: rolls back transactions

Directory for each schema

Undo logs

Where did my data go?

SELECT @@datadir; -- shows where files are kept
On my Mac: /usr/local/mysql/data/

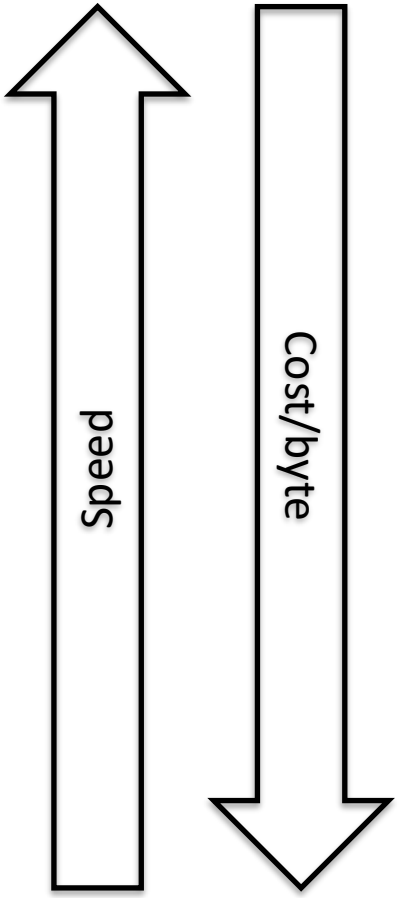
From terminal
$ ls -la /usr/local/mysql/data/nyc_data

```
drwxr-x---  12 _mysql  _mysql       384 May  4 08:25 .
drwxr-x---  48 _mysql  _mysql      1536 May 10 11:52 ..
-rw-r-----   1 _mysql  _mysql    114688 Apr 29 18:04 actions.ibd
-rw-r-----   1 _mysql  _mysql    114688 Apr 29 18:04 cuisine.ibd
-rw-r-----   1 _mysql  _mysql    131072 May  5 08:56 employees.ibd
-rw-r-----   1 _mysql  _mysql  16777216 Apr 29 18:04 inspections.ibd
-rw-r-----   1 _mysql  _mysql    114688 Apr 29 18:04 inspectiontypes.ibd
-rw-r-----   1 _mysql  _mysql  17825792 Apr 29 18:05 inspectionviolations.ibd
-rw-r-----   1 _mysql  _mysql 130023424 Apr 29 18:04 restaurant_inspections.ibd
-rw-r-----   1 _mysql  _mysql  13631488 May  5 11:38 restaurants.ibd
-rw-r-----   1 _mysql  _mysql    114688 May  6 19:01 users.ibd
-rw-r-----   1 _mysql  _mysql    147456 Apr 29 18:05 violationcodes.ibd
```

One file for each table

Each file is stored on disk

There are many different types of data storage used by databases



CPU Cache

Main memory

Flash memory

Magnetic disk

Tape

CPU cache is fast, but small and volatile

CPU cache

- Fast but expensive
- Holds small amount of data (normally megabytes of recently used data)
- Volatile (loose if power fails)

Caches

Cache	Size	Access time
Register	< 1 KB	< 1 ns
L1	64 KB	1 ns
L2/L3	32 MB	10 ns

CPU Cache

Typical access time
1-10 ns

Main memory

Flash memory

Magnetic disk

Tape

Main memory is larger, but still small and volatile

Main memory

- Larger than CPU cache (gigabytes)
- Volatile (loose if power fails)
- Some databases can be entirely contained in memory
- Use CS10 data structures to access data quickly if stored in memory

CPU Cache

Typical access time 1-10 ns

Main memory

50-100 ns

Flash memory

Magnetic disk

Tape

Flash (SSDs) are larger and non-volatile, but much slower than main memory

Flash memory

- Solid State Drives (SSDs)
- Often larger than main memory (up to terabytes)
- Non-volatile (do *not* loose if power fails)
- Faster data access than magnetic disks
- Read data in blocks (pages) of about 4 KB

CPU Cache — Typical access time 1-10 ns

Main memory — 50-100 ns — Volatile

Flash memory — 24 - 100 µs — Non-volatile

Magnetic disk

Tape

Magnetic disk have been the mainstay of data storage for decades

Magnetic disk

- Mainstay of data storage
- Large (up to dozens of terabytes)
- Made up of (perhaps many) spinning platters
- Slower than SSDs
  - Seek track/ sector
  - Rotational latency
- Read data in blocks (page) of roughly 8 KB

Limit disk access when possible

"If RAM access is like grabbing a book from your shelf, Disk access is like waiting for Amazon Prime"
Speed ratio: 100,000:1

CPU Cache — Typical access time 1-10 ns

Main memory — 50-100 ns — Volatile

Flash memory — 25 - 100 µs — Non-volatile

Magnetic disk — 5 - 10 ms

Tape

Tape is large, but slow; mainly used for back ups

Tape

- Very large capacity (terabytes)
- Normally used for off-line backups
- Do you need back up if replicate database?
- YES!!!!
- A rouge process that writes garbage writes it to all replicas!

CPU Cache — Typical access time 1-10 ns

Main memory — 50-100 ns

Volatile

Non-volatile

Flash memory — 25 - 100 µs

Magnetic disk — 5 - 10 ms

Tape — Seconds to minutes

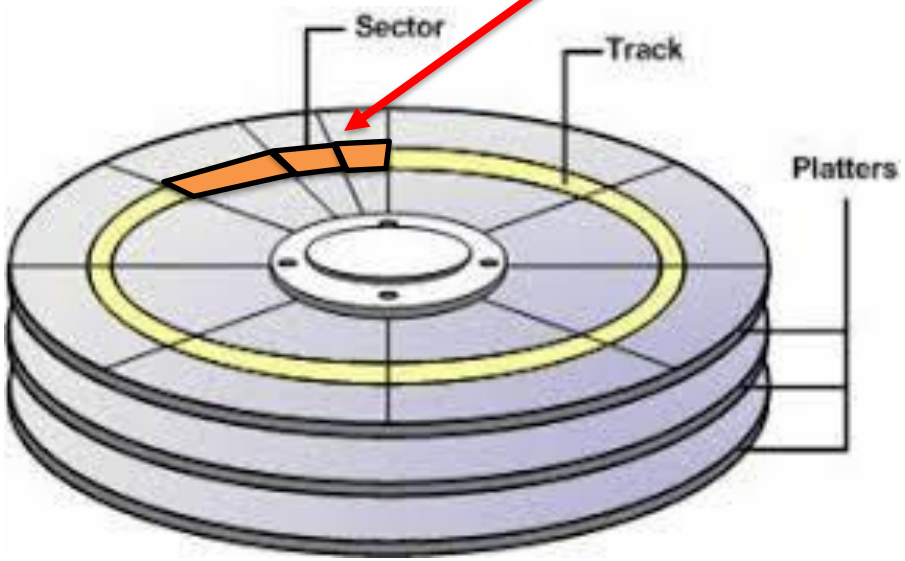
Most online data is stored on Hard Disk Drive (HDD) or Solid-State Disk (SSD)

Data storage on magnetic disk

Time to read/write:

- **Seek time to track**
- **Rotational delay for sector**
- **Transfer speed**
- **Controller overhead**

1 disk block
4KB in size



RAID – Redundant Array of Independent Disks stores data across multiple disks

- Striping to increase throughput (RAID 0)
- Mirroring to reduce failures (RAID 1)
- Striping and Mirroring (RAID 10)

Hard drive read/write

- A hard disk often has multiple spinning *platters*, each with a read/write head
- Each platter has several concentric *tracks*
- Each track is divided into multiple *sectors*
- The read/write head can address data on at a track/sector location
- To read or write, move arm to correct track, and wait for sector to spin underneath head
- Disk itself cannot address smaller amounts of data than one sector (normally 512 bytes)

Operating system addresses blocks of data

- A block spans several sectors
- OS cannot address smaller than block
- Normally around 4 KB today (8 sectors)
- Files written across multiple blocks

Most online data is stored on Hard Disk Drive (HDD) or Solid-State Disk (SSD)

SSDs use Flash Memory

- Data stored in a grid of memory cells, each a transistor that traps electrons
- Cells organized into:
  - Pages: read/write unit, typically 4-16 KB
  - Blocks: groups of pages typically 128 KB-4 MB
- No moving parts — access is purely electrical
  - 4 to 100 X faster than HDD
  - More expensive

Important quirk:

- Flash cells must be erased before being rewritten
- Erase happens at the *block* level, not the page level
- Creates complexity (write amplification, garbage collection) that the SSD controller hides from the OS
- A controller chip manages reads, writes, wear leveling, and garbage collection

Reading: No need to wait for disk to spin into correct position means low latency (around 0.02 ms, 100X faster than HDD)

Durability: No moving parts makes them resistant to shock, vibrations

Power: no need to spin disk, uses less poer

Wear: finite number of write cycles (managed through wear leveling)

DAS vs NAS vs SAN: The classic comparison

Feature	Direct Attached Storage (DAS)
Description	Drives directly attached to one server
Access type	Block
Shared?	One server only
Performance	Fast but limited to one server
Cost	Low
Typical use	Small servers

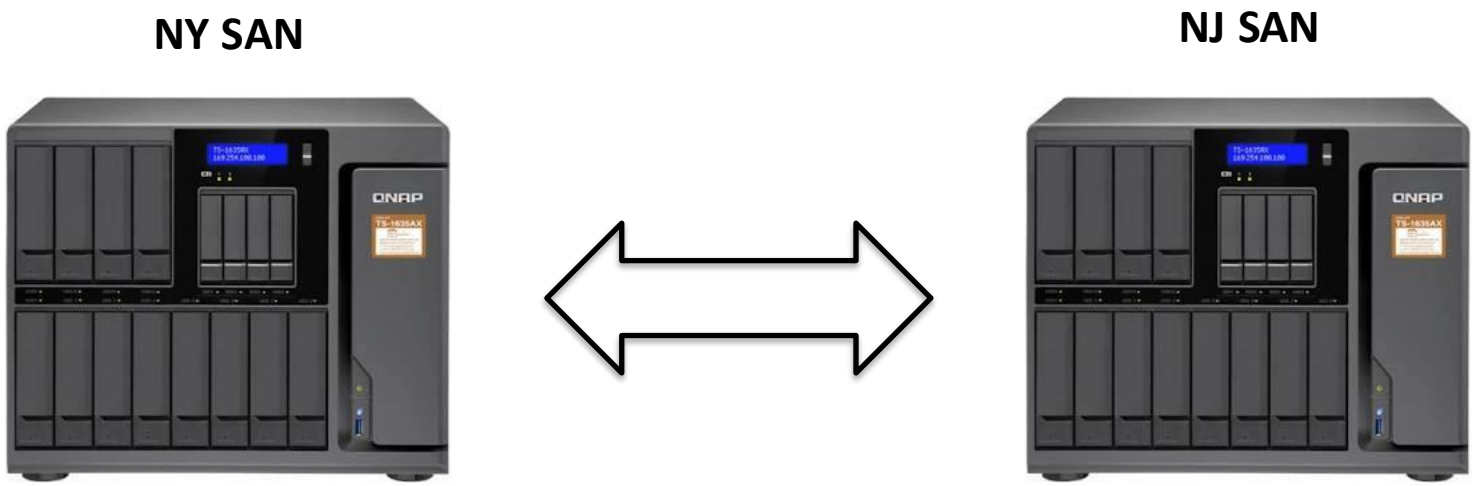
DAS vs NAS vs SAN: The classic comparison

Feature	Direct Attached Storage (DAS)	Network Attached Storage (NAS)
Description	Drives directly attached to one server	Serves files over network and is relatively inexpensive
Access type	Block	File
Shared?	One server only	Multiple clients
Performance	Fast but limited to one server	Moderate
Cost	Low	Moderate
Typical use	Small servers	File shares, home directories

DAS vs NAS vs SAN: The classic comparison

Feature	Direct Attached Storage (DAS)	Network Attached Storage (NAS)	Storage Area Network (SAN)
Description	Drives directly attached to one server	Serves files over network and is relatively inexpensive	Complex, high-speed network of multiple storage devices
Access type	Block	File	Block
Shared?	One server only	Multiple clients	Multiple clients
Performance	Fast but limited to one server	Moderate	Very fast, low latency
Cost	Low	Moderate	High
Typical use	Small servers	File shares, home directories	Databases, VMs, enterprise apps

Disks are often grouped into Storage Area Networks (SANs)



Storage Area Network (SAN)

- Provides block-addressable high-performance non-volatile storage
- SANs are often connected to other SANs
- SAN software replicates data across SAN devices to eliminate single point of failure (e.g., fire in data center)
- Not JBOD, often use striping and mirroring within one SAN

SANs pros and cons

Pros

- **Consolidation**: One storage pool serves many servers
- **Utilization**: Allocate space where needed instead of stranding it on each server
- **Performance**: Optimized for high-throughput, low-latency block I/O
- **Centralized management**: Snapshots, replication, encryption all in one place
- **Disaster recovery**: Replicate entire system to a remote SAN
- **Server independence**: Replace a server without migrating data

Cons

- **Cost**: Fiber Channel SANs are expensive (switches, HBAs, arrays, licensing)
- **Complexity**: Requires specialized expertise (SAN admins are a job role)
- **Single point of failure risk**: If misconfigured, the whole SAN can take down many servers

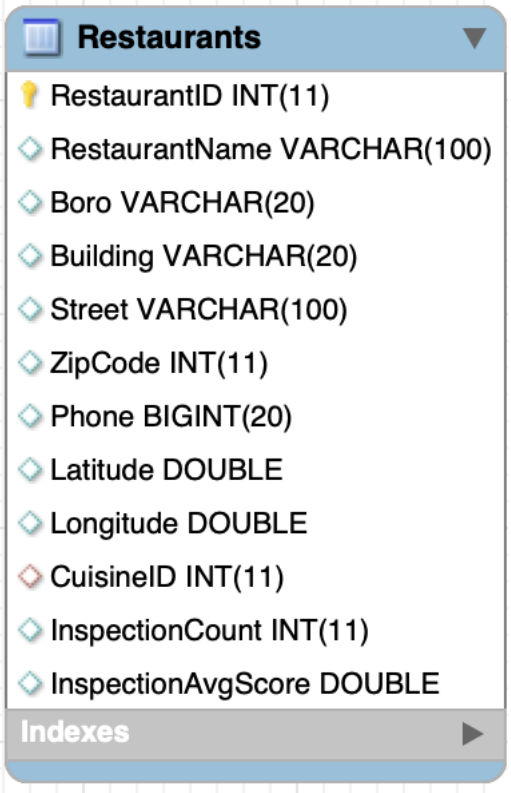
Modern Trends

- **All-flash arrays** have largely replaced spinning disks for performance tiers
- **Cloud storage** (EBS in AWS, Persistent Disk in GCP) is essentially "SAN-as-a-service" — block storage attached to VMs over a network

Agenda

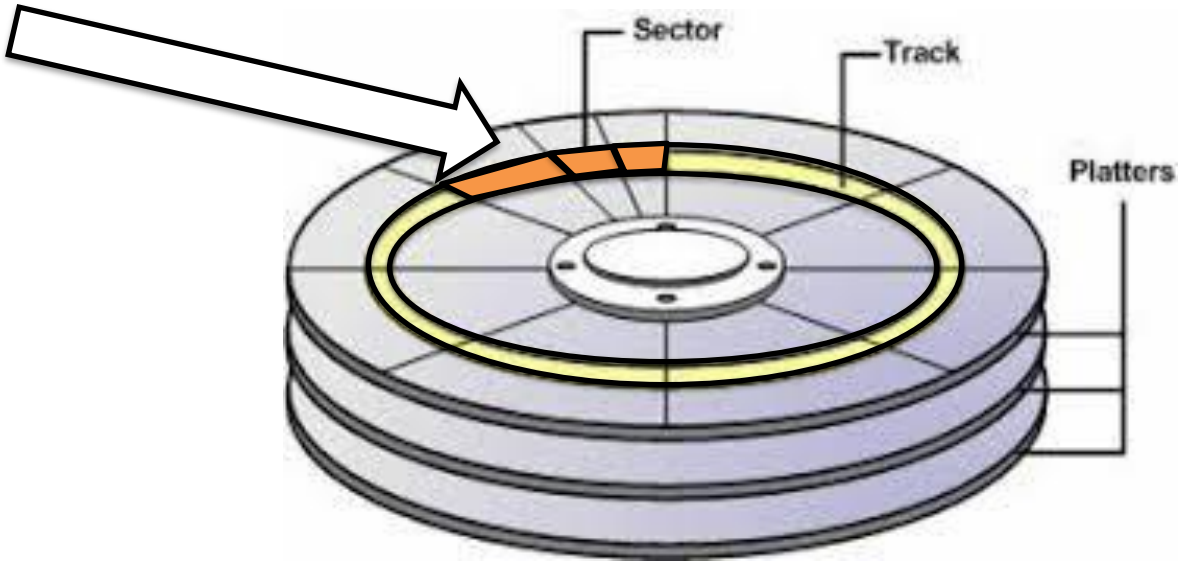
1. Data persistence
2. Database file organization
3. Indexing

Databases persist data onto disk in files that normally span multiple blocks



Two approaches

- **Fixed length rows**
- **Variable length rows**



- Each relation generally stored in one file
- Each file is a sequence of table rows made up of attributes mapped onto disk blocks roughly 4KB in size (16 KB for MySQL)
- One row assumed to be smaller than block size (book gives solution if not)

Fixed length rows are easy to implement, but can waste disk space

Simplified fixed length Restaurant records

ID INT	Name CHAR(100)	Boro CHAR(20)	AvgGrade DOUBLE
30075445	Morris Park Bake Shop	Bronx	10.6
30075445	Wendy's	Brooklyn	19.8
30191841	DJ Reynolds Pub and Restaurant	Manhattan	10.8

4 byte integers

100 bytes for Name

Other bytes not used if Restaurant name < 100 characters long

20 bytes for Boro

Other bytes not used if Boro name < 20 characters long

8 byte double

Row *size* = 132 bytes, find record *i* at *file start + i*132* bytes (use *fseek* in C or *seek* in Python)

Most likely some bytes wasted (Restaurant name is probably not 100 characters long)

Two other problems:

1. Unless block size is exactly 132 bytes, records may cross disk block boundaries
   - Use *block size/row size* bytes of each block, discard the remainder
2. Hard to delete records
   - Could move all records up (costly!)
   - Mark record as deleted and keep pointers to next free space

Deleting rows can be tricky, could copy records to fill hold, but inefficient!

Simplified fixed length Restaurant records

ID INT	Name CHAR(100)	Boro CHAR(20)	AvgGrade DOUBLE
30075445	Morris Park Bake Shop	Bronx	10.6
Deleted			
30191841	DJ Reynolds Pub and Restaurant	Manhattan	10.8
40356018	Riviera Caterers	Brooklyn	11.1

Could fill gap but might involve many copies if record near start of file and many entries
This would be slow!!!

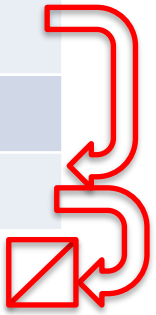
Row *size* = 132 bytes, find record *i* at *file start + i*132* bytes
Most likely some bytes wasted (Restaurant name is probably not 100 characters long)
Two other problems:

1. Unless block size is exactly 132 bytes, records will cross disk block boundaries
   - Use *block size/row size* bytes of each block, discard the remainder
2. Hard to delete records
   - Could move all records up (costly!)
   - Mark record as deleted and keep pointers to next free space

A better way to handle deletes is to keep a list of free spaces

Simplified fixed length Restaurant records

ID INT	Name CHAR(100)	Boro CHAR(20)	AvgGrade DOUBLE
30075445	Morris Park Bake Shop	Bronx	10.6
Deleted			
30191841	DJ Reynolds Pub and Restaurant	Manhattan	10.8
Deleted			



Normally more inserts than deletes

Keep list of free spaces, insert record into space on free list on next insert

We know the row will fit because all rows are the same sizse

Row *size* = 132 bytes, find record *i* at *file start + i*132* bytes

Most likely some bytes wasted (Restaurant name is probably not 100 characters long)

Two other problems:

1. Unless block size is exactly 132 bytes, records will cross disk block boundaries
   - Use *block size/row size* bytes of each block, discard the remainder
2. Hard to delete records
   - Could move all records up (costly!)
   - Mark record as deleted and keep pointers to next free space

Inserting new row, add onto end if no entries in free list

Simplified fixed length Restaurant records

ID INT	Name CHAR(100)	Boro CHAR(20)	AvgGrade DOUBLE
30075445	Morris Park Bake Shop	Bronx	10.6
30075445	**Wendy's**	**Brooklyn**	**19.8**
30191841	DJ Reynolds Pub and Restaurant	Manhattan	10.8
Deleted			

Insert new record into free space or add to end of file
Wendy's inserted at free space

Row *size* = 132 bytes, find record *i* at *file start + i*132* bytes
Most likely some bytes wasted (Restaurant name is probably not 100 characters long)
Two other problems:

1. Unless block size is exactly 132 bytes, records will cross disk block boundaries
   - Store *block size/row size* records in each block, do not use the remainder
2. Hard to delete records
   - Could move all records up (costly!)
   - Mark record as deleted and keep pointers to next free space

Inserting new row, add onto end if no entries in free list

Simplified fixed length Restaurant records

ID INT	Name CHAR(100)	Boro CHAR(20)	AvgGrade DOUBLE
30075445	Morris Park Bake Shop	Bronx	10.6
30075445	Wendy's	Brooklyn	19.8
30191841	DJ Reynolds Pub and Restaurant	Manhattan	10.8
40356018	**Riviera Caterers**	**Brooklyn**	**11.1**

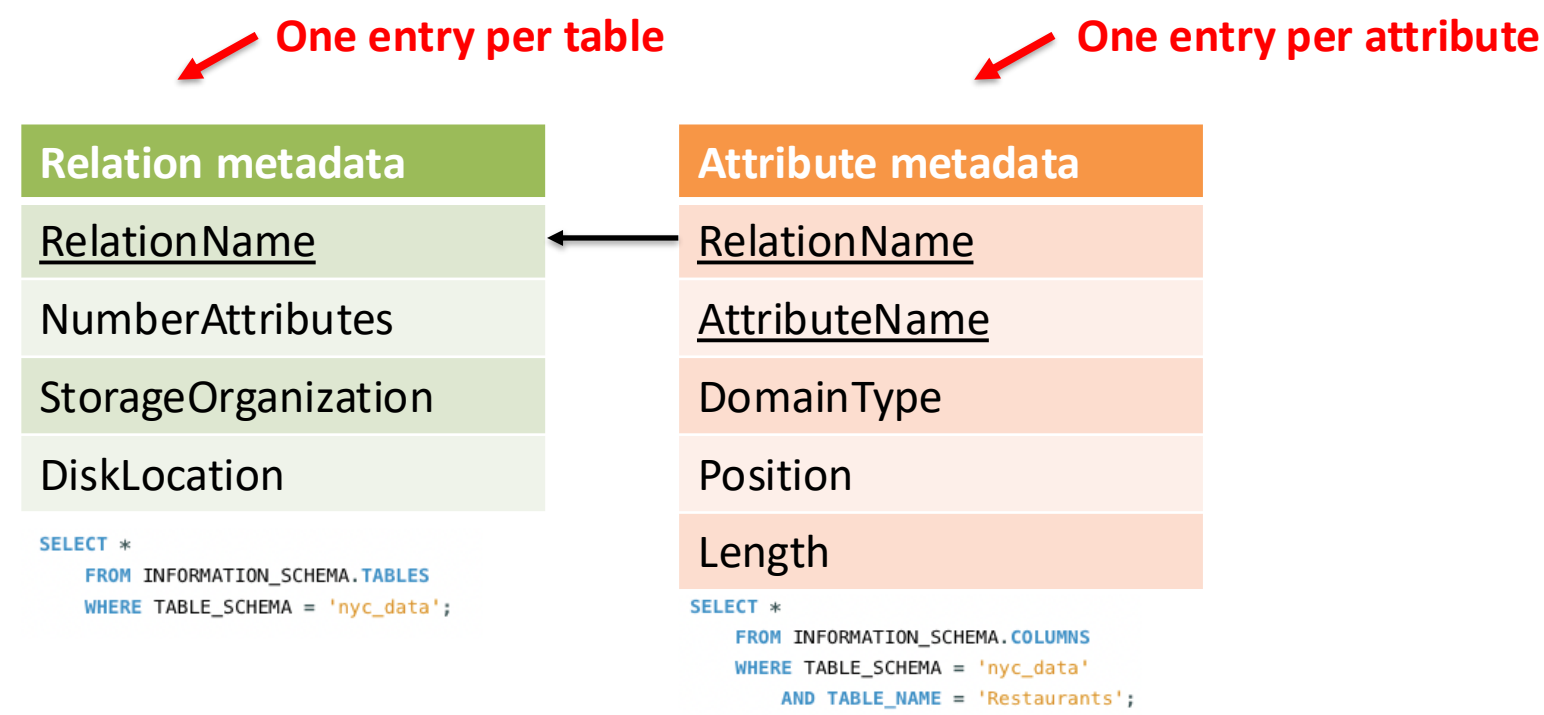
Insert new record into free space or add to end of file
Riviera Caterers instead at end

Row *size* = 132 bytes, find record *i* at *file start + i*132* bytes
Most likely some bytes wasted (Restaurant name is probably not 100 characters long)
Two other problems:

1. Unless block size is exactly 132 bytes, records will cross disk block boundaries
   - Store *block size/row size* records in each block, do not use the remainder
2. Hard to delete records
   - Could move all records up (costly!)
   - Mark record as deleted and keep pointers to next free space

Database keeps track of file layouts in data dictionary (system catalog)

Data dictionary



Data dictionary tracks relations and attributes

Relation metadata table has entry for each table

Attribute metadata uses RelationName in PK and has entry for each attribute

- Lists domain type for each attribute (e.g., INT, VARCHAR, DOUBLE)
- Position in record layout on disk
- Length of attribute

You can see this data by querying the INFROMATION_SCHEMA table

Variable length records

MySQL tracks some other attributes in addition to our somewhat simplified model shown in the previous slide

```
1 •  USE nyc_data;
2 •  SELECT *
3        FROM INFORMATION_SCHEMA.COLUMNS
4        WHERE TABLE_SCHEMA = 'nyc_data'
5            AND TABLE_NAME = 'Restaurants';
```

00% 14:1

Result Grid Filter Rows: Search Export:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	CHA
def	nyc_data	Restaurants	Boro	3	NULL	YES	varchar	20	80
def	nyc_data	Restaurants	Building	4	NULL	YES	varchar	20	80
def	nyc_data	Restaurants	CuisineID	12	NULL	YES	int	NULL	NULL
def	nyc_data	Restaurants	InspectionAvg	10	NULL	YES	float	NULL	NULL
def	nyc_data	Restaurants	InspectionCount	11	NULL	YES	float	NULL	NULL
def	nyc_data	Restaurants	Latitude	8	NULL	YES	double	NULL	NULL
def	nyc_data	Restaurants	Longitude	9	NULL	YES	double	NULL	NULL
def	nyc_data	Restaurants	Phone	7	NULL	YES	bigint	NULL	NULL
def	nyc_data	Restaurants	RestaurantID	1	NULL	NO	bigint	NULL	NULL
def	nyc_data	Restaurants	RestaurantName	2	NULL	YES	varchar	100	400
def	nyc_data	Restaurants	Street	5	NULL	YES	varchar	100	400
def	nyc_data	Restaurants	ZipCode	6	NULL	YES	int	NULL	NULL

Variable length records are more complicated to implement but save space

Variable length records

NULL	ID	NamePtr	BoroPtr	AvgScore	Name data	Boro data
0 3	4 7	8 11	12 15	16 23	24 42	43 47
0000	30075445	24, 19	43, 5	10.6	Morris Park Bake Shop	Bronx

To track NULLs, records have a byte array where each bit represents one attribute

- Set bit to 1 to indicate the value is NULL
  - e.g., Boro is in 3rd position, set 3rd bit to 1 if Boro is NULL
- I did not show this field in the fixed length record, but often used there too

Some domain types are always of fixed length such as INT and DOUBLE

Variable length records

INT fixed length 4 bytes

DOUBLE fixed length 8 bytes

NULL	ID	NamePtr	BoroPtr	AvgScore	Name data	Boro data
0 3	4 7	8 11	12 15	16 23	24 42	43 47
0000	30075445	24, 19	43, 5	10.6	Morris Park Bake Shop	Bronx

Look up attribute types in data dictionary, get order and length for each attribute

Attribute metadata
RelationName
AttributeName
DomainType
Position
MaxLength

Relation Name	Attribute Name	Domain Type	Position	Max Length
T_1	ID	INT	1	4
T_1	Name	VARCHAR	2	100
T_1	Boro	VARCHAR	3	20
T_1	AvgScore	DOUBLE	4	8

Use start and length pointers to track variable length attributes such as VARCHAR

Variable length records

Variable length have start and length integers
Start and end are fixed length, 2 bytes each

NULL	ID	NamePtr	BoroPtr	AvgScore	Name data	Boro data
0 3	4 7	8 11	12 15	16 23	24 44	45 49
0000	30075445	24, 21	45, 5	10.6	Morris Park Bake Shop	Bronx

Look up attribute types in data dictionary, get order and length for each attribute

Attribute metadata
RelationName
AttributeName
DomainType
Position
MaxLength

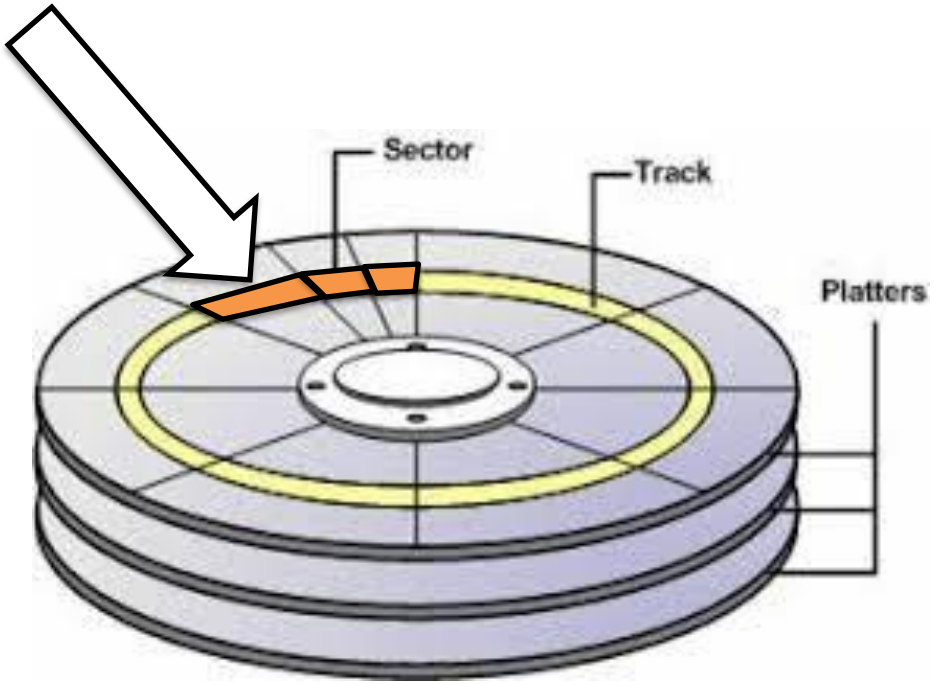
Relation Name	Attribute Name	Domain Type	Position	Max Length
T_1	ID	INT	1	4
T_1	**Name**	**VARCHAR**	**2**	**100**
T_1	**Boro**	**VARCHAR**	**3**	**20**
T_1	AvgScore	DOUBLE	4	8

Name starts at byte 24 and is 21 characters long
Boro starts at byte 45 and is 5 characters long

Multiple records are typically stored in one disk block

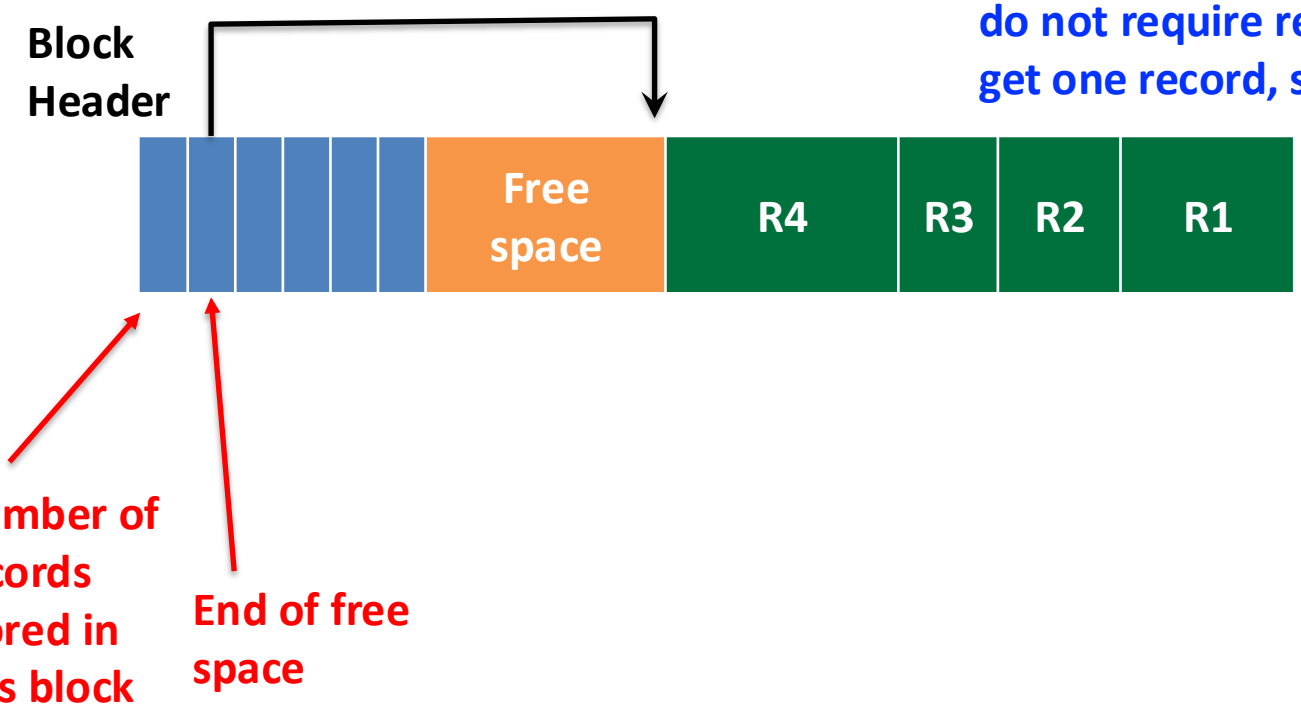
RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30075445	MORRIS PARK BAKE SHOP	Bronx	1007	MORRIS PARK AVE	10462	7188924968
30112340	WENDY'S	Brooklyn	469	FLATBUSH AVENUE	11225	7182875005
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40356018	RIVIERA CATERERS	Brooklyn	2780	STILLWELL AVENUE	11224	7183723031
40356151	BRUNOS ON THE BOULEVARD	Queens	8825	ASTORIA BOULEVARD	11369	7183350505
40356483	WILKEN'S FINE FOOD	Brooklyn	7114	AVENUE U	11234	7184443838
40356731	TASTE THE TROPICS ICE CREAM	Brooklyn	1839	NOSTRAND AVENUE	11226	7188560821
40357217	WILD ASIA	Bronx	2300	SOUTHERN BOULEVA…	10460	7182207846

If disk block is 4KB and each row is roughly 500 bytes, then there are around 8 records (4,000/500) per disk block



Data stored in blocks on disk so that records do not span multiple blocks

Disk block organization



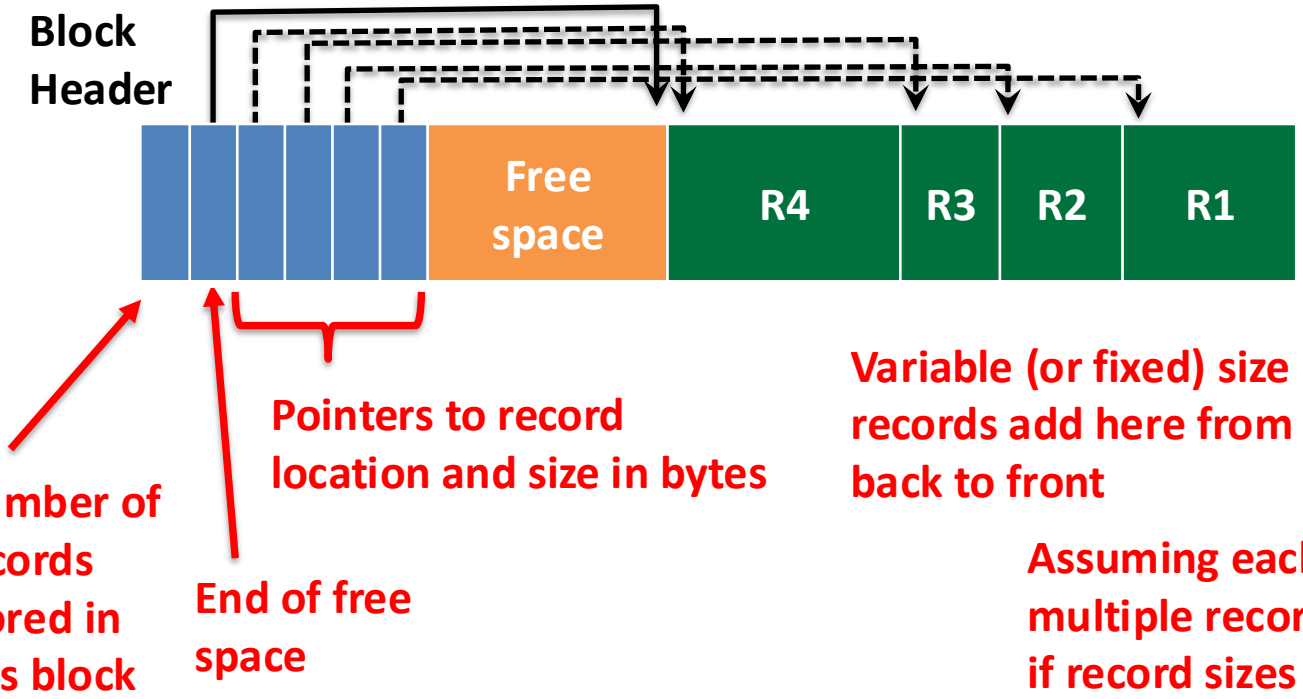
Remember reading from disk is slow, do not require reading two blocks to get one record, so do not span blocks!

Storing records in a disk block

- Header gives number of records stored in block
- Records added from back to front
- Free space in between header and records
- New records added to end of free space

Data stored in blocks on disk so that records do not span multiple blocks

Disk block organization



Storing records in a disk block

- Header gives number of records stored in block
- Records added from back to front
- Free space in between header and records
- New records added to end of free space

This format is sometimes called a ***slotted page***

Agenda

1. Data persistence
2. Database file organization
3. Indexing

Without indices, database must do a full table scan to find rows meeting criteria

```
2 •  SELECT * FROM Restaurants
3    WHERE Boro = 'Manhattan';
4
5
```

100% 1:6

Must look over entire table for restaurants in Manhattan
Called a *table scan*
Slow if table is large! 10 million rows = 10 million comparisons
Disk I/O is 1000X (or more) slower than memory access!
If query by boro a lot, would prefer fast way to find by boro

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744

Notice results are sorted by RestaurantID, why?
MySQL automatically creates a *clustered index* for PK (concatenate for composite key PKs)
Clustered index means rows are sorted on disk by PK, secondary indices are not
MySQL uses physical ordering based on clustered index to return rows

Each table has at least one index; normally based on Primary Key

```
2 •   SELECT * FROM Restaurants
3     WHERE Boro = 'Manhattan';
4
5
```

100% 1:6

Indexes commonly use a B+ tree

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744

There can be only one (and only one) clustered index per table
All tables have at least one index
If a primary key is not declared, MySQL uses the first UNIQUE index
If no UNIQUE index, MySQL creates a synthetic column with RowID

Add new rows to table, update index with location of each new row

```
2 •   SELECT * FROM Restaurants
3     WHERE Boro = 'Manhattan';
4
5
```

100% 1:6

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744

If new rows are inserted into table:

- **Add at space indicated by free space list or end of file**
- **Update all indexes to show where new row is located (indices cause increased overhead)**
- **Database will move records to keep clustered index sorted by index when not busy**

Add deleted rows to free list and update index for each row removed

```
2 •   SELECT * FROM Restaurants
3     WHERE Boro = 'Manhattan';
4
5
```

100% 1:6

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

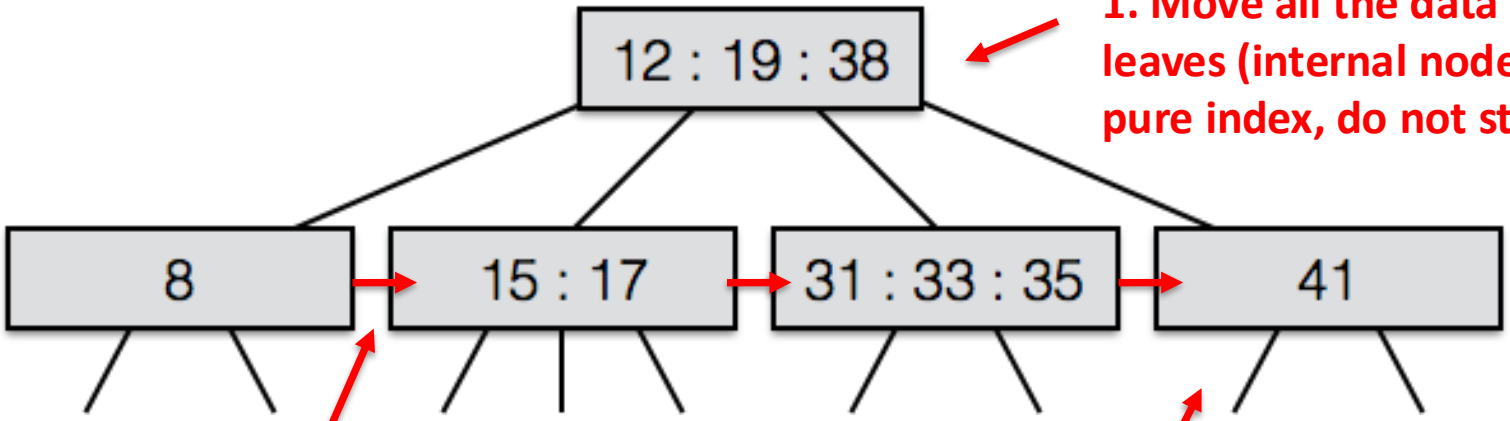
RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744

If rows are deleted:

- **Add to row to free space list**
- **Update all indices by removing entry (indices cause increased overhead)**
- **Database will move records to keep clustered index sorted by index when not busy**

Indices speed up reads, but slow down writes

Indexes use B+ trees (like 2-3-4 trees from CS10 with three main changes)



1. Move all the data down to the leaves (internal nodes become a pure index, do not store data)

2. Link leaves together

3. Make every node much larger
Holds hundreds of keys, not 3

2-3-4 tree review:

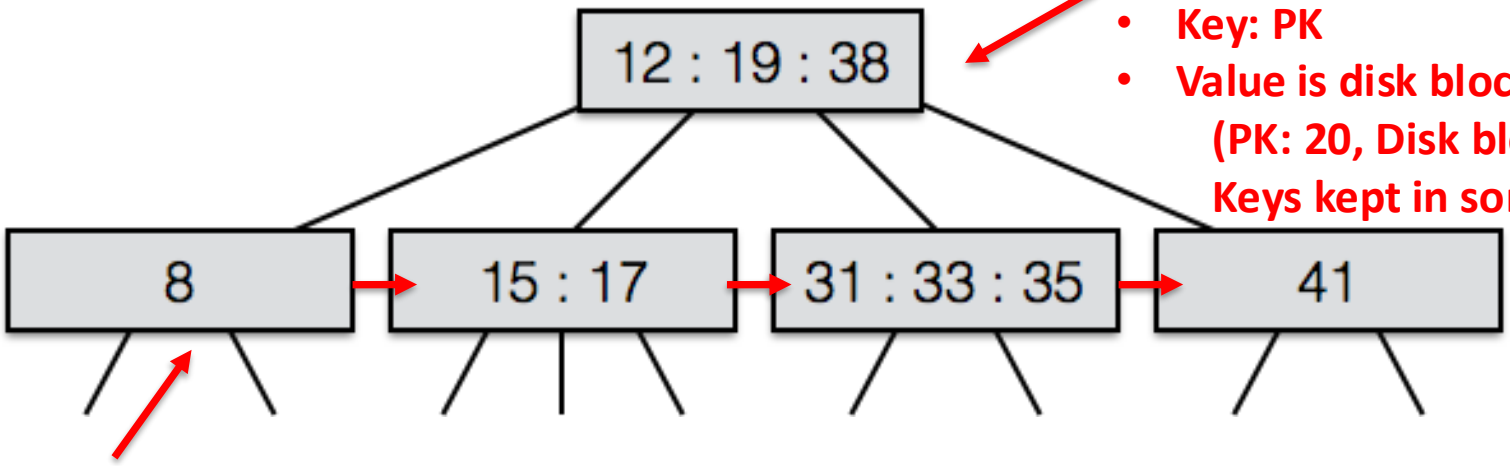
- Insert key and value (only key shown)
- Store up to three keys per node
- Keys in sorted order in nodes
- Split and promote when node is full
- All leaves on the same level

B+ trees work similar, but with three main changes

Each node is exactly the size of one disk block
Why?
We can only read disk in blocks, so might as well get a lot of keys per read!

Due to large key size in nodes, this tree will not be tall (tree is shallow)

Non-leaf nodes navigate to leaf nodes (which hold table row data in slotted page)



Non-leaf nodes:

- **Key: PK**
- **Value is disk block of slotted page**
  (PK: 20, Disk block: 1047)
  Keys kept in sorted order

Leaf nodes:

- **Hold table row data in slotted page**
- **Keys in sorted order**
- **Leaves chained together into a linked list**

$\text{Log}_{\lceil m/2 \rceil} n \leq \text{height} \leq \log_m n$

Where:

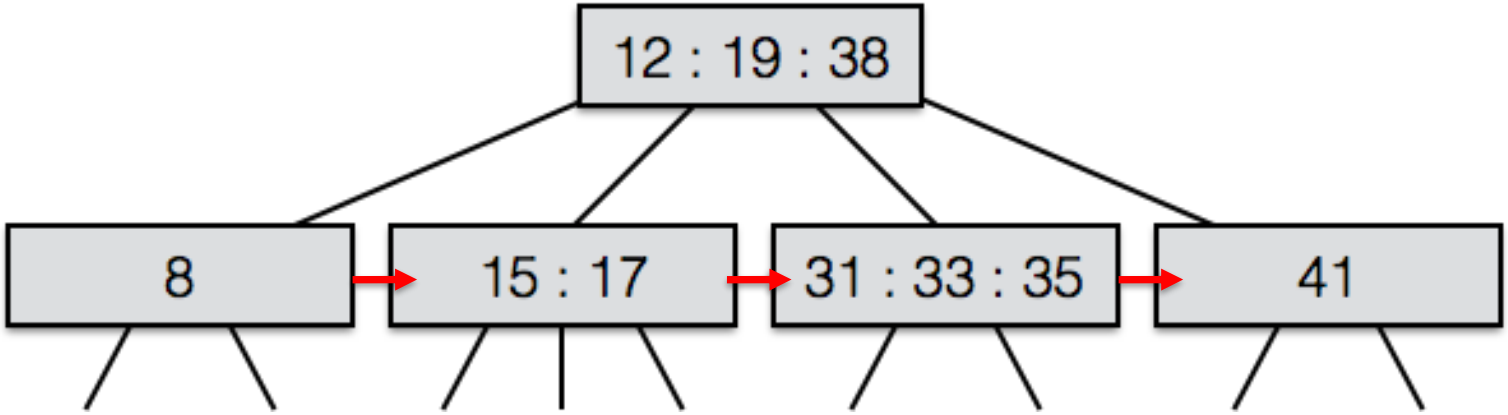
m = children/node

n = nodes in B+ tree

In practice:

- **m is normally 100 or more**
- **Height is usually around 3 or 4, even for large tables**

Find row data using Primary Key by traversing B+ tree



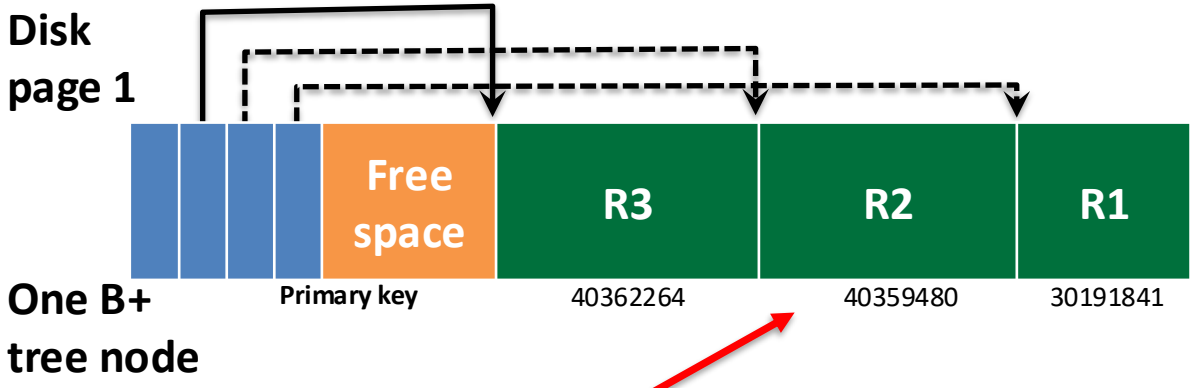
Query looking for specific Primary Key

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID = 30075445;
```

To find row quickly by PK:

- **Start at root**
- **Traverse tree looking for PK**
- **Each key has a value that is the disk block of child node**
- **Read child node from disk if not already in memory**
- **Find edge to follow using binary search in node (remember, nodes now have many keys, not use 3 like 2-3-4 trees!)**

Leaves use the slotted format to store multiple table rows



Leaf nodes are a block that stores tables rows in a slotted fashion

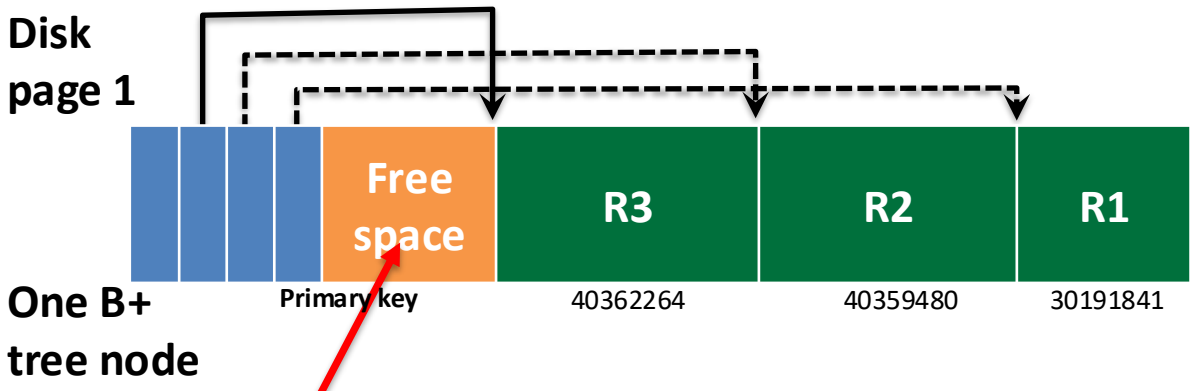
Size of each node is one disk page

Data sorted by primary key

First 3 rows

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

Fill leaf node until run out of space



Leaf nodes are a block that stores tables rows in a slotted fashion

Size of each node is one disk page

Data sorted by primary key

Next insert does not fit

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715		Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298		Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

Split and promote like a 2-3-4 tree

Internal node

- **No data**
- **Just pointers to disk blocks based on key**

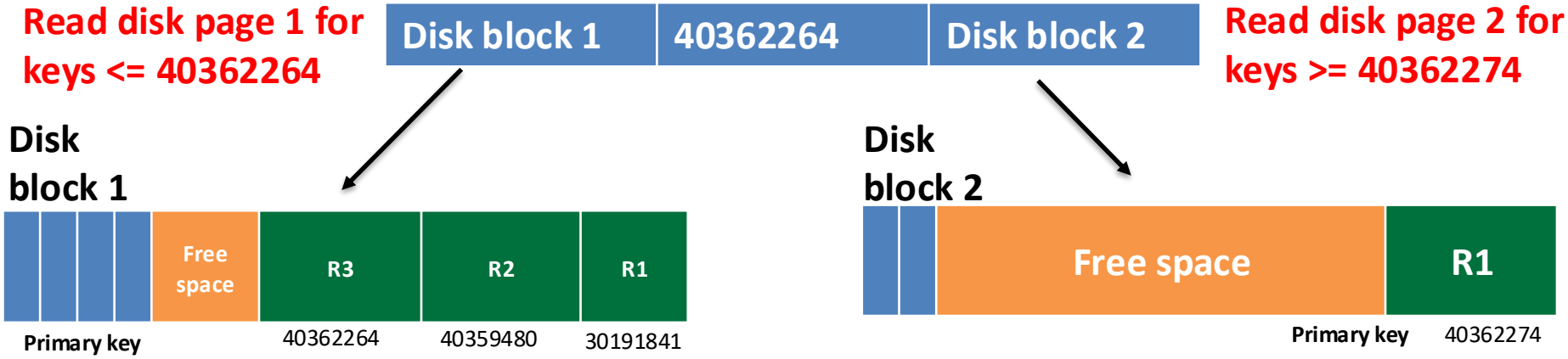
Split and promote like 2-3-4 tree

Disk block 1

Disk block 2

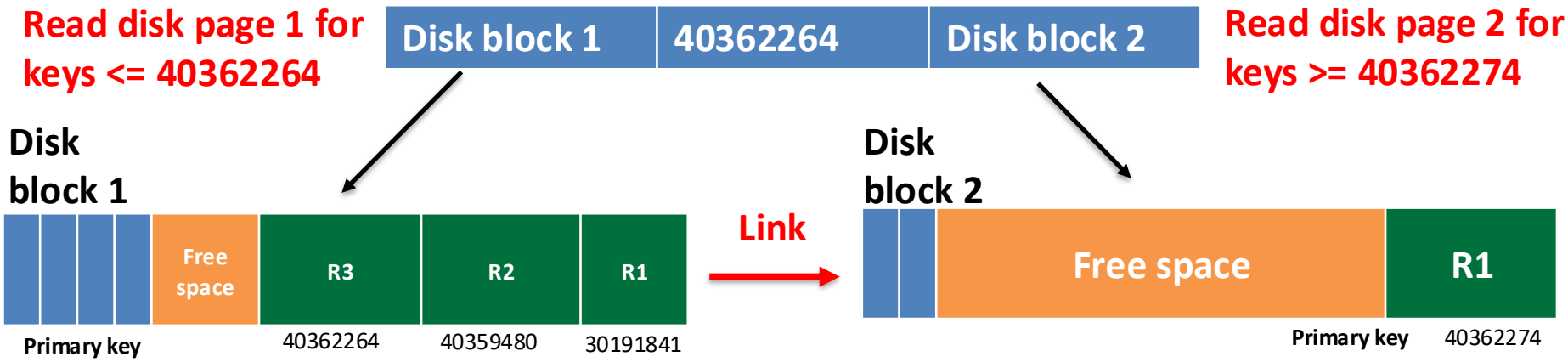
RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

Split and promote like a 2-3-4 tree



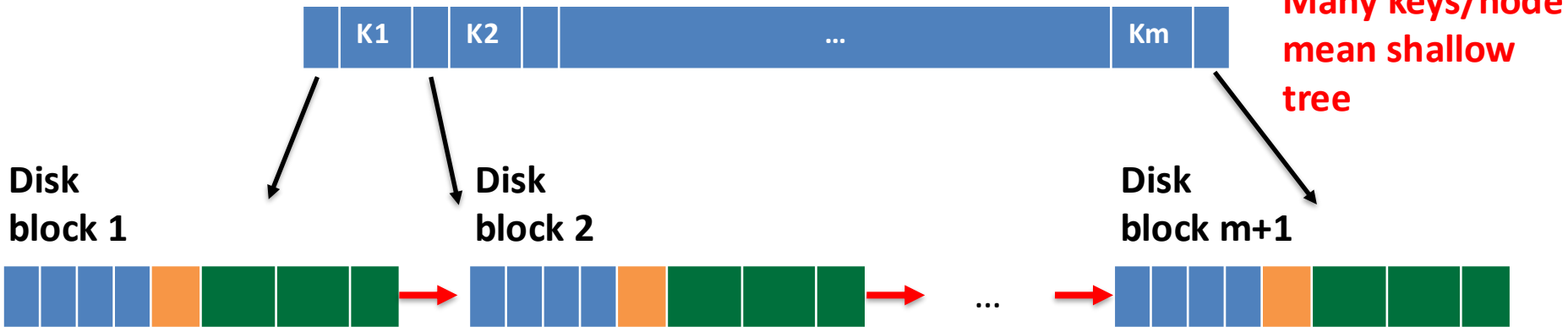
RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

Link leaf nodes



RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

Continue to split and promote as rows are added to the table



RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude
30191841	DJ REYNOLDS PUB AND RESTAU…	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262064
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570	40.72574362
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132	40.70590688
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820	40.75181634
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080	40.77528111
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121	40.75751173
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200	40.77807357
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882	NULL
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137127
40364347	METROPOLITAN CLUB	Manhattan	1	EAST 60 STREET	10022	2128387400	40.76479552
40364362	21 CLUB	Manhattan	21	WEST 52 STREET	10019	2125827200	40.76040784

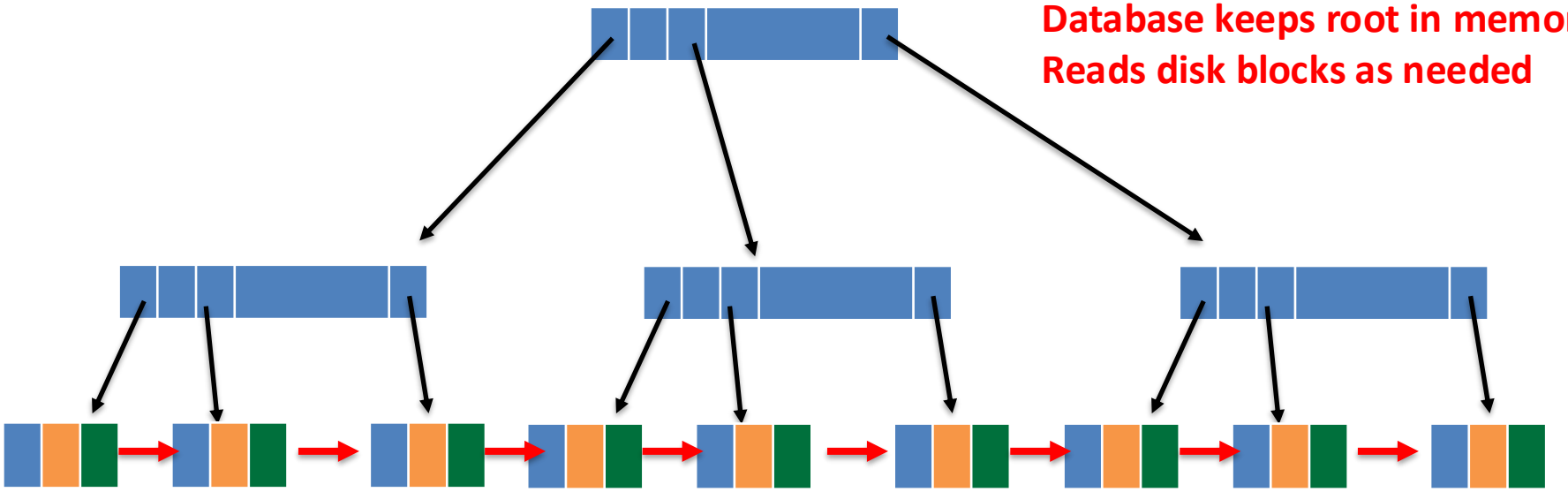
Each internal node can hold many keys

Block size >= (M * Key size) + ((M+1) * Pointer size) + Page header

If:
Block size: 16 KB (MySQL uses this value)
Page header: 96 bytes (leaves 16096 bytes for data)
Index Key: BIGINT (8 bytes)
Disk Pointer: 8 bytes

M: $\frac{16096\ bytes}{16\ bytes/entry} \cong 1000$ keys

Even tables with a large number of rows produce shallow B+ trees



Each internal node holds up to 1000 keys
Most tables are only 3 or 4 layers deep

Data base IS the B+ tree organized by PK (leaves hold row data)
Nodes are physically stored on disk in sorted order by Primary Key
Must update B+ tree index as nodes added, deleted, updated to keep order

There are several types of searches a database performs

Search types:

Equality (WHERE ID = 5)

Range query (WHERE ID BETWEEN 5 AND 10)

Sorted retrieval (ORDER BY ID)

Equality queries are fast!

Creating B+ tree index

Equality query ID = 40359480

1) Start at root (binary search to find edge)

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID = 40359480;
```

Equality queries are fast!

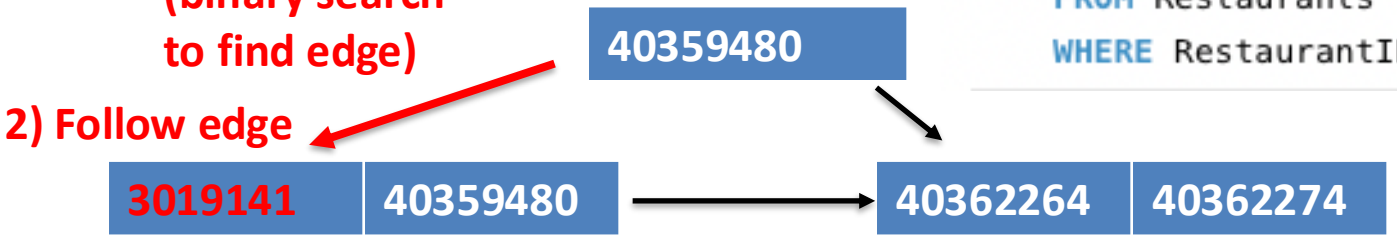
Creating B+ tree index

Equality query ID = 40359480

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID = 40359480;
```

1) Start at root (binary search to find edge)

2) Follow edge

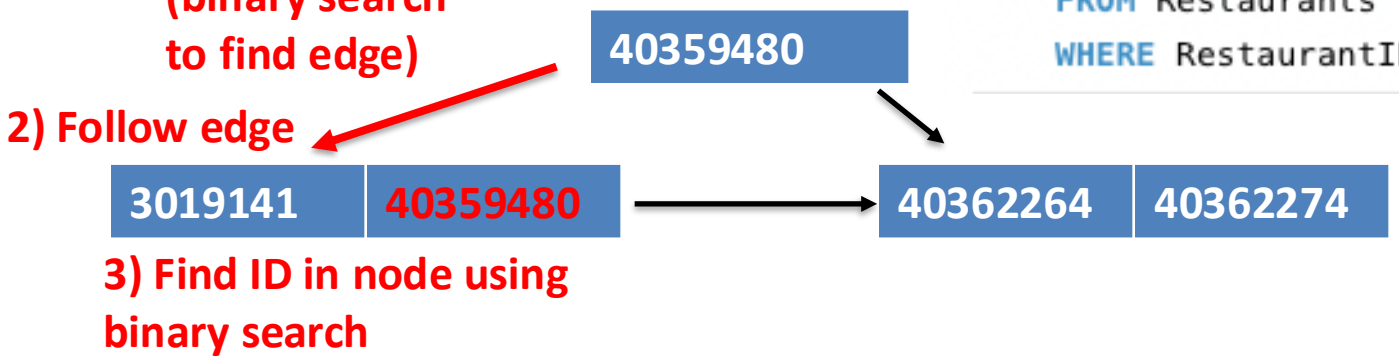


Equality queries are fast!

Creating B+ tree index

Equality query ID = 40359480

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID = 40359480;
```



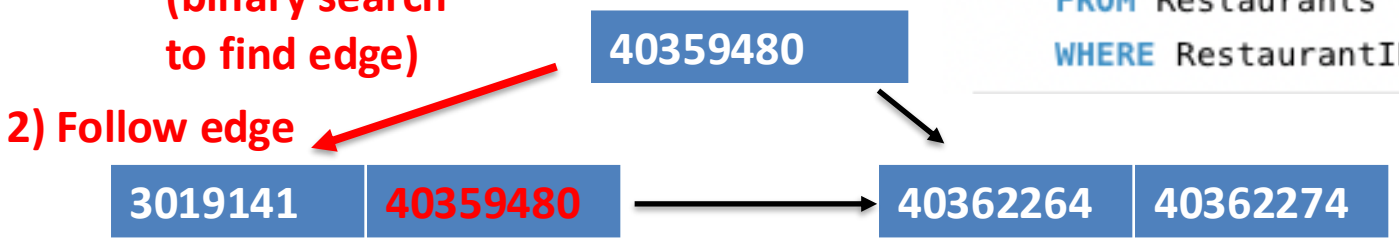
Equality queries are fast!

Creating B+ tree index

Equality query ID = 40359480

1) Start at root (binary search to find edge)

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID = 40359480;
```



RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	InspectionAvg	Inspection
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691423	-73.96958057845	9	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

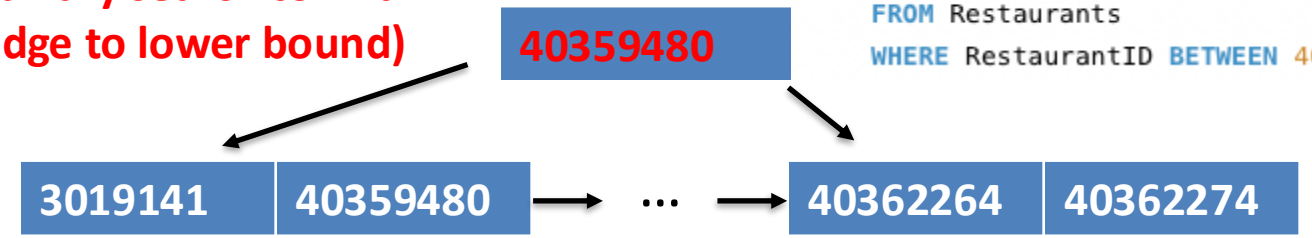
Range queries are also fast!

Creating B+ tree index

1) Start at root (binary search to find edge to lower bound)

Range: ID between 40359480 AND 40362264

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID BETWEEN 40359480 AND 40362264;
```



Range queries are also fast!

Creating B+ tree index

Range: ID between 40359480 AND 40362264

1) Start at root (binary search to find edge to lower bound)

2) Follow edge

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID BETWEEN 40359480 AND 40362264;
```

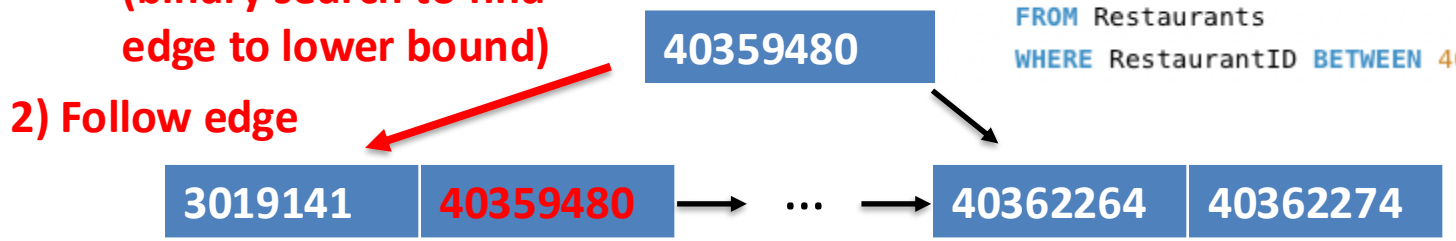
Range queries are also fast!

Creating B+ tree index

Range: ID between 40359480 AND 40362264

1) Start at root (binary search to find edge to lower bound)

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID BETWEEN 40359480 AND 40362264;
```



3) Find lower bound in node

4) ID has row data

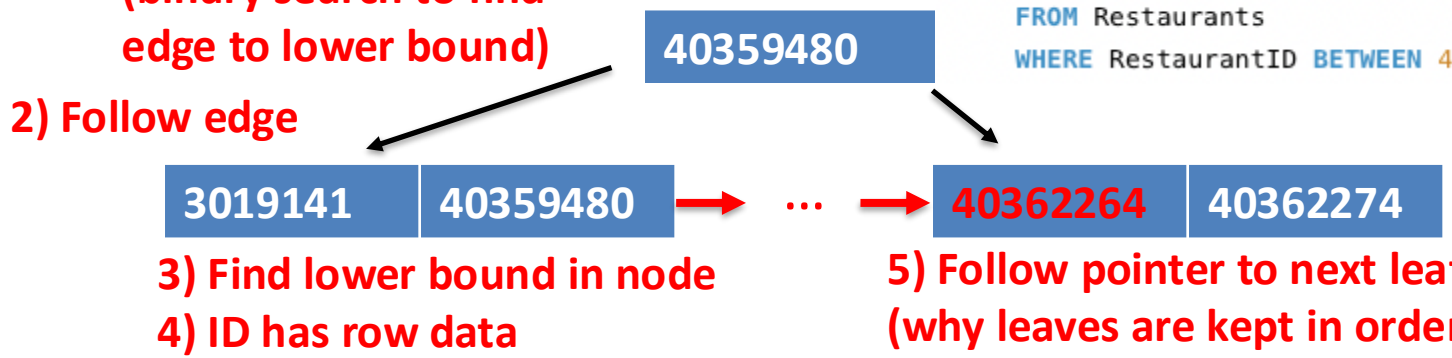
Range queries are also fast!

Creating B+ tree index

Range: ID between 40359480 AND 40362264

1) Start at root
(binary search to find
edge to lower bound)

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID BETWEEN 40359480 AND 40362264;
```



Range queries are also fast!

Creating B+ tree index

1) Start at root (binary search to find edge to lower bound)

2) Follow edge

Range: ID between 40359480 AND 40362264

```
SELECT *
    FROM Restaurants
    WHERE RestaurantID BETWEEN 40359480 AND 40362264;
```

Restau...	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	InspectionAvg	InspectionCou
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691423	-73.96958057845	9	4
40359705	NATHAN'S FAMOUS	Brooklyn	1310	SURF AVENUE	11224	7183332202	40.57553686942	-73.98165226411	40.625	16
40360045	SEUDA FOODS	Brooklyn	705	KINGS HIGHWAY	11223	7183751500	40.60618692143	-73.9654664459	10.75	8
40361618	SAL'S DELI	Queens	129-08	20 AVENUE	11356	7186619498	40.78167383607	-73.83941597683	11.5714	7
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262063589	-73.96770961573	12.5455	11
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

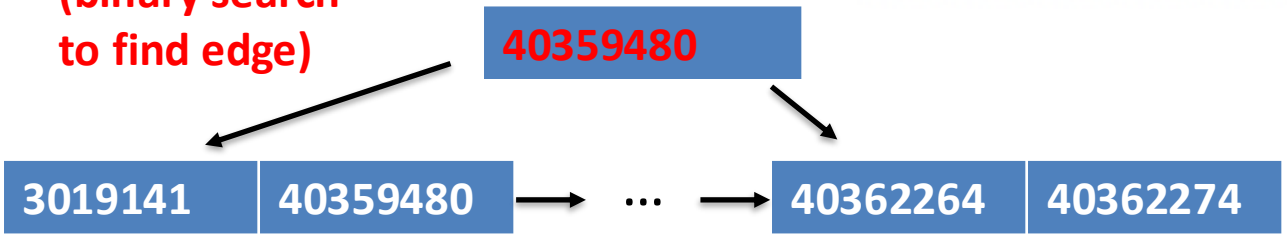
Sorted order query

Creating B+ tree index

ORDER BY ID

```
SELECT *
    FROM Restaurants
    ORDER BY RestaurantID;
```

1) Start at root (binary search to find edge)



Sorted order query

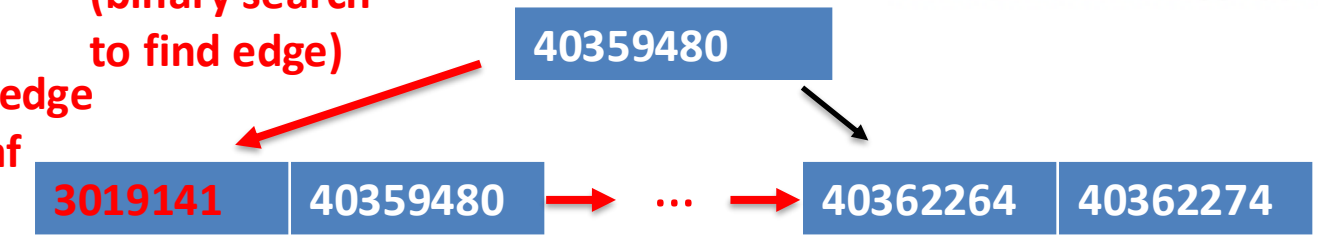
Creating B+ tree index

ORDER BY ID

```
SELECT *
    FROM Restaurants
    ORDER BY RestaurantID;
```

1) Start at root (binary search to find edge)

2) Follow edge to first leaf



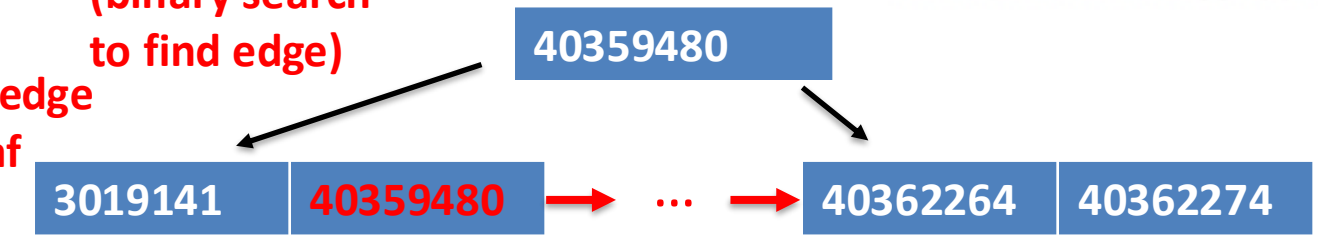
Sorted order query

Creating B+ tree index

ORDER BY ID

```
SELECT *
    FROM Restaurants
    ORDER BY RestaurantID;
```

1) Start at root (binary search to find edge)



2) Follow edge to first leaf

3) Get IDs in order in node

4) Follow pointer to next leaf (leaves kept in order)

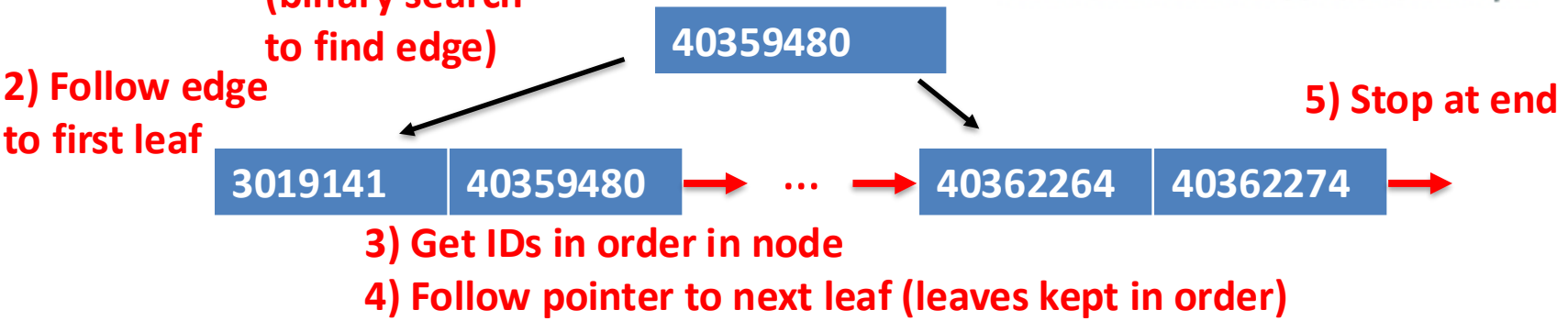
Sorted order query

Creating B+ tree index

ORDER BY ID

```
SELECT *
    FROM Restaurants
    ORDER BY RestaurantID;
```

1) Start at root (binary search to find edge)



Restau...	RestaurantName	Boro	Building	Street	ZipCode	Phone	Latitude	Longitude	InspectionAvg	In
30075445	MORRIS PARK BAKE SHOP	Bronx	1007	MORRIS PARK AVENUE	10462	7188924968	40.84823122453	-73.85597188993	19.6842	19
30191841	D.J. REYNOLDS	Manhattan	351	WEST 57 STREET	10019	2122452912	40.76732571155	-73.98431042362	18.4	10
40356018	RIVIERA CATERERS	Brooklyn	2780	STILLWELL AVENUE	11224	7183723031	40.57989566331	-73.98208665586	10	2
40356483	WILKEN'S FINE FOOD	Brooklyn	7114	AVENUE U	11234	7184443838	40.62011159093	-73.9069894897	23.65	20
40356731	TASTE THE TROPICS ICE CREAM	Brooklyn	1839	NOSTRAND AVENUE	11226	7188560821	40.64079501746	-73.94848798153	11.3333	9
40357217	ASIA PLAZA CAFÉ	Bronx	2300	SOUTHERN BOULEVARD	10460	7187411426	40.850536752	-73.88245508146	12	4
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900	40.76854691423	-73.96958057845	9	4
40359705	NATHAN'S FAMOUS	Brooklyn	1310	SURF AVENUE	11224	7183332202	40.57553686942	-73.98165226411	40.625	16
40360045	SEUDA FOODS	Brooklyn	705	KINGS HIGHWAY	11223	7183751500	40.60618692143	-73.9654664459	10.75	8
40361618	SAL'S DELI	Queens	129-08	20 AVENUE	11356	7186619498	40.78167383607	-73.83941597683	11.5714	7
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030	40.79262063589	-73.96770961573	12.5455	11
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STREET	10012	2129952570	40.72574361744	-73.99747810759	10.2	5
40362432	HO MEI	Queens	103-05	37 AVENUE	11368	7187796903	40.75330591158	-73.86429537578	18.7778	9
40362869	SHASHEMENE INT'L RESTAURANT	Brooklyn	195	EAST 56 STREET	11203	3474300871	40.65198322279	-73.92457494415	11.7143	7
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342	40.75618542308	-73.99056473379	18.9091	11
40363427	BAGELS N BUNS	Staten Isl...	2491	VICTORY BOULEVARD	10314	7187611900	40.61026786383	-74.14640121029	23.5714	14
40363834	CARVEL	Staten Isl...	1111	HYLAN BOULEVARD	10305	7188167807	40.59855313075	-74.07964600818	15.75	8
40363920	NEW GOLDEN BILLION	Brooklyn	976	RUTLAND ROAD	11212	7187357707	40.66214059458	-73.92718021322	6	4
40364179	MISS MAIME'S SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744	40.80137126967	-73.96015994361	20.4	5
40364220	KOSHER BAGEL HOLE	Brooklyn	1423	AVENUE J	11230	7182584150	40.62511112873	-73.96203433937	23	10

Secondary indexes speed up finding data on non-PK attributes

```
2 •   SELECT * FROM Restaurants
3     WHERE Boro = 'Manhattan';
4
5
```

100% 1:6

Side note: it is more common to say secondary *indexes* than secondary *indices* in the database world

Result Grid | Filter Rows: Search | Edit: | Export/Import: | Fetch rows:

RestaurantID	RestaurantName	Boro	Building	Street	ZipCode	Phone
30191841	DJ REYNOLDS PUB AND RESTAURANT	Manhattan	351	WEST 57 STREET	10019	2122452912
40359480	1 EAST 66TH STREET KITCHEN	Manhattan	1	EAST 66 STREET	10065	2128793900
40362264	P & S DELI GROCERY	Manhattan	730	COLUMBUS AVENUE	10025	2129323030
40362274	ANGELIKA FILM CENTER	Manhattan	18	WEST HOUSTON STR…	10012	2129952570
40362715	THE COUNTRY CAFE	Manhattan	60	WALL STREET	10005	3474279132
40363298	CAFE METRO	Manhattan	625	8 AVENUE	10018	2127149342
40363426	LEXLER DELI	Manhattan	405	LEXINGTON AVENUE	10174	2126870820
40363630	LORENZO & MARIA'S KITCHEN	Manhattan	1418	THIRD AVENUE	10028	2127941080
40363685	BERKELEY	Manhattan	437	MADISON AVENUE	10022	2128328121
40363945	DOMINO'S	Manhattan	148	WEST 72 STREET	10023	2125010200
40364149	AUNTIE ANNE'S PRETZELS	Manhattan	0	34 STREET	NULL	2122396882
40364179	SPOONBREAD TOO	Manhattan	364	WEST 110 STREET	10025	2128656744

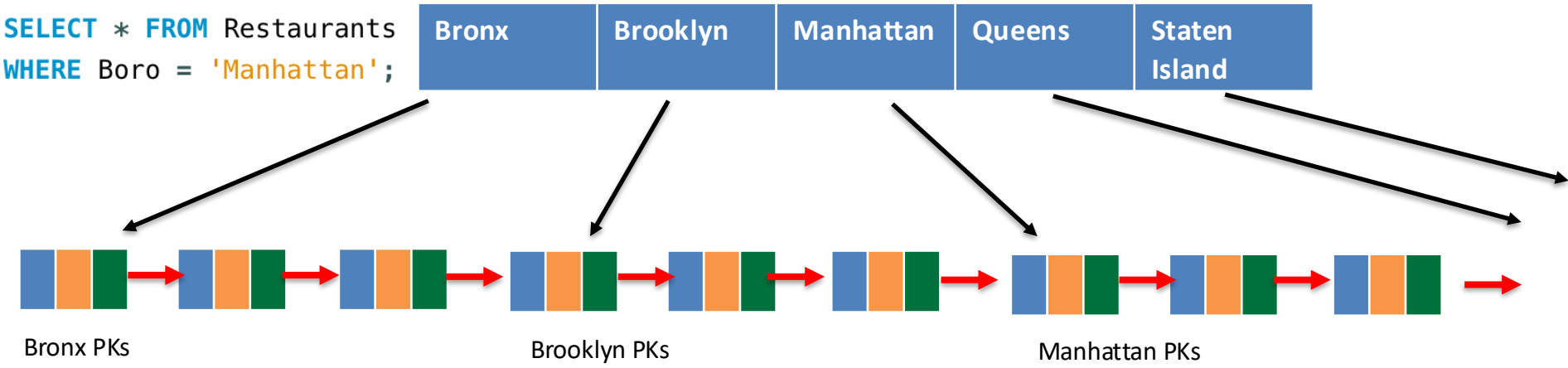
Secondary index: separate B+ tree built from values from a non-PK attribute of a table
Keep B+ tree of values in column
Why not put an index on every column?
Trades space and write performance for read performance (indexes must be updated!)

Secondary indexes give PK disk block, look up data in PK index

```
CREATE INDEX idx_boro ON Restaurants(Boro);
```

Builds second B+ tree, keyed by Boro

```
SELECT * FROM Restaurants
WHERE Boro = 'Manhattan';
```



Second B+ built on Boro values
To find Manhattan restaurants
Start at root
Follow edge to Manhattan
Leaves give PK of Manhattan restaurants
(not table row data like clustered index does)
Now go to clustered B+ tree and search by PK

If index on boro did not exist, MySQL would have to search the whole table to find Manhattan restaurants

Now it can quickly find PKs of Manhattan restaurants

Get data by traversing primary index by PK

Create indices on tables using the CREATE INDEX command

CREATE INDEX idx_boro **ON** Restaurants(Boro);

- If index is secondary index (non-clustered), a second B+ tree gives primary key of each attribute value
- Use attribute value + primary key to get unique value (‘Queens’ + PK)
- Now have pointers to rows with all unique values (for each Boro here)
- Result is that we can find all restaurants in each Boro (say Queens) without scanning the entire table
- Cardinality of the index is the number of unique items (5 boros here)
- MySQL creates a separate BTREE for each index on table
- Index downsides:
  - More storage space (not practical to put index on each attribute)
  - Must keep indices up to date on insert, update, delete operations
  - Database reorders clustered indices when not busy