

CS5200

Database Management

Indexing

Ken Baclawski
Spring 2017

Outline

- Mid-Term Exam
- Indexes
- Hash Tables
- B-Trees
- Assignment #6



Mid-Term Exam



Indexes

Storage Organization

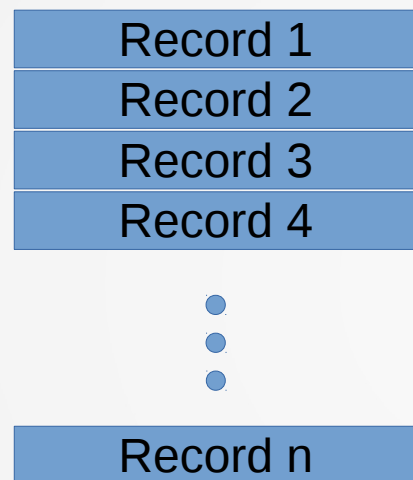
- The data in a database is stored in a persistent storage device
- All storage devices store data in large blocks, typically around 32K bytes
 - Reading and writing is *always* done in blocks
 - This is true both for hard drives and solid state drives
- Data stored on such a device should be organized to take advantage of this property

Index

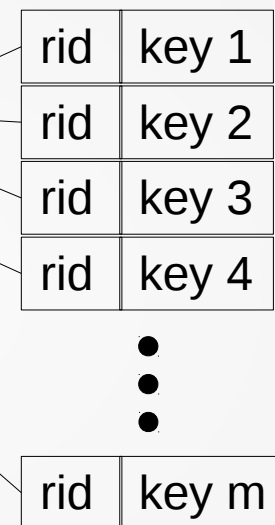
- An *index* is a data storage organization that allows rapid access to data given a *key*.
- There are two main kinds of index
 - Clustered index
 - Secondary index
- A clustered index for a table contains the records for the table
 - One can regard this index as *being* the table
- A secondary index for a table contains pointers to the records, not the records themselves

Indexes

Clustered Index



Secondary Index



In practice, the key for a clustered index is the primary key of the table. This is so commonly done that clustered indexes are also referred to as being *primary indexes*.

Types of Index

- Indexes (either clustered or secondary) are usually either hash tables or tree indexes
 - The main exceptions are the indexes for geographic data
- Tree indexes are almost always B-trees
 - The “B” in B-tree stands for “Bayer” who invented them

Specifying indexes in SQL

- The primary key has a clustered index
 - MySQL will always make a primary key into a clustered index
- Every unique key is enforced by a secondary index
- Additional secondary indexes can be specified for non-unique keys

Specifying indexes in SQL

```
create table Counselor(  
    id int not null,  
    primary key(id) using hash,  
    title varchar(200) not null  
);  
create table Client(  
    id int not null,  
    primary key(id) using hash,  
    name varchar(200) not null,  
    index ClientName(name) using btree  
);  
create table Meeting(  
    sees int references Client(id)  
        on delete cascade on update cascade,  
    meetsWith int references Counselor(id)  
        on delete cascade on update cascade,  
    primary key(sees, meetsWith) using btree  
);
```

The name of the primary index is "PRIMARY" which cannot be the name of any other index

The name of the Btree index on Client(name) is "ClientName"

Specifying indexes in SQL

```
create table MeetingDay(  
    sees int,  
    meetsWith int,  
    day enum('Monday','Tuesday','Wednesday',  
            'Thursday','Friday'),  
    foreign key(sees, meetsWith)  
        references Meeting(sees, meetsWith)  
        on delete cascade on update cascade,  
    primary key(sees, meetsWith, day) using btree  
);
```


Specifying indexes in SQL

```
create table Notes(  
  id int not null,  
  primary key(id) using hash,  
  sees int not null,  
  meetsWith int not null,  
  foreign key(sees, meetsWith)  
    references Meeting(sees, meetsWith)  
    on delete cascade on update cascade,  
  text varchar(65000) not null,  
  index NotesText(text(300)) using btree  
  date Date not null,  
  index NotesDate(date) using hash  
);
```

Only the first 300 characters of the text column are indexed.

The target of a foreign key must have an index.
Notes(date) is a target, so that column must have an index.

Specifying indexes in SQL

```
create table DateAnnotation(  
  id int not null,  
  primary key(id) using hash,  
  date Date not null,  
  foreign key(date) references Notes(date)   
    on delete no action on update no action,  
  text varchar(65000) not null  
);
```

The target of a foreign key must have an index. The target is Notes(date), so that column must have an index.

What is a key?

- Synonym for “index”
- The column or columns in an index
- The column or columns in the main unique index is the “primary key”
- A column or columns that must have values in the column or columns of another table is a “foreign key”
- A value being searched for is a “search key”
- A column or columns that uniquely determines a record in a table is a “superkey”
- A minimal set of columns that uniquely determines a record in a table is a “candidate key”



Hash Tables

Hash Table

- A sequence of disk blocks
 - The blocks are called *buckets*
- The hash value of the key is the address of the bucket, relative to the beginning of the hash table
- The bucket holds the records or (key, rid) pairs

| | |
|---|---------------|
| 0 | |
| 1 | The |
| 2 | |
| 3 | |
| 4 | over |
| 5 | quick, jumped |
| 6 | brown |
| 7 | fox |

```
hash("The") == 1
hash("quick") == 5
hash("brown") == 6
hash("fox") == 7
hash("jumped") == 5
hash("over") == 4
```


Hash Table

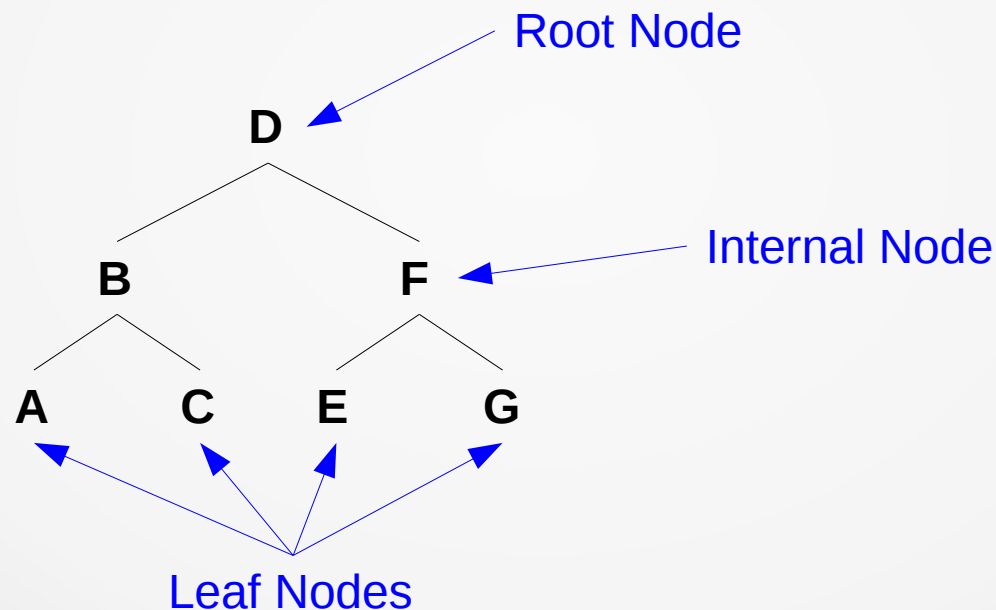
- When a hash table runs out of space, there are many techniques for expanding the hash table
 - The classical technique is to double the size of the hash table and rehash all elements into the new hash table
 - Expandable algorithms gradually increase the size rather than abruptly increasing it
- The classical technique is simpler and more efficient on average, but the hash table could be unavailable for a relatively long time during rehashing



B-Trees

Tree structures

A tree consists of a root node, internal nodes and leaf nodes



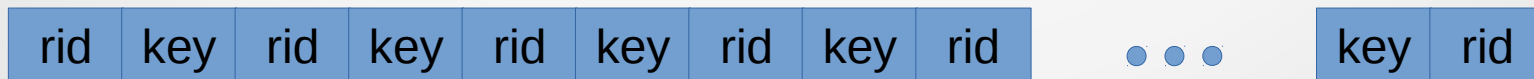
B-Tree versus B⁺-Tree

- The original B-tree proposed by Bayer had records in all of the nodes
- A later version, the B⁺-tree, has the records in the leaf nodes
- Main memory tree structures use B-trees
 - The usual name is “red-black tree” or “binary tree” but actually a B-tree of order 4
 - Dominates main memory tree structures
- External storage tree structures use B⁺-trees
 - Dominates external storage tree structures

B-Tree Node

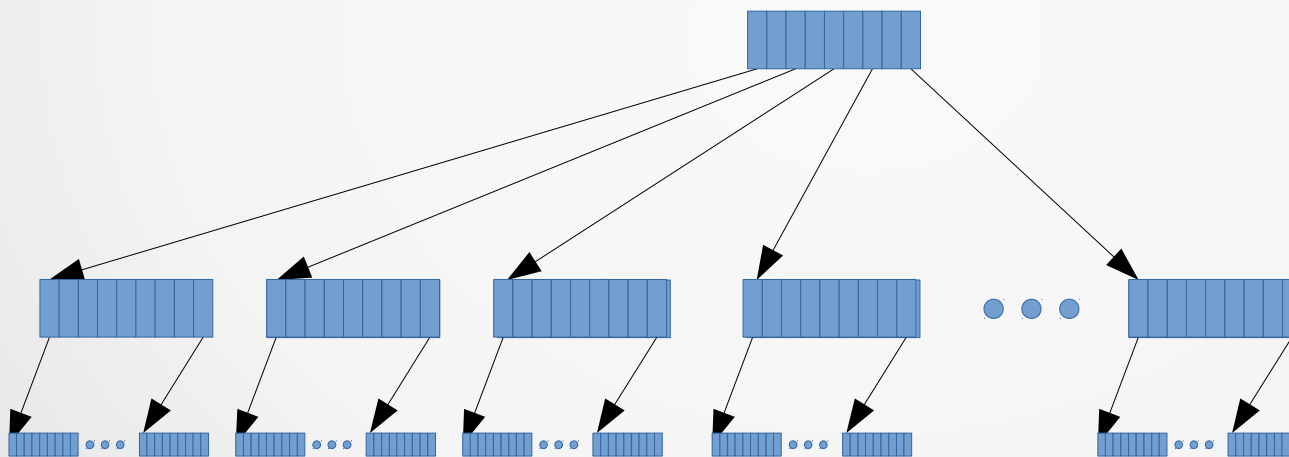
A B-tree node (root or internal node) has a sequence of either records or (key, address) pairs

- Like hash table buckets, a B-tree node uses a whole block
- Unlike hash table buckets, the entries in a B-tree node are in sorted order
- A single B-tree node can have several thousand addresses



B-Tree Structure

- The entire B-tree has a series of levels, starting with the root
- It is difficult to show the structure because of the enormous number of nodes at each level



This level can have thousands of nodes

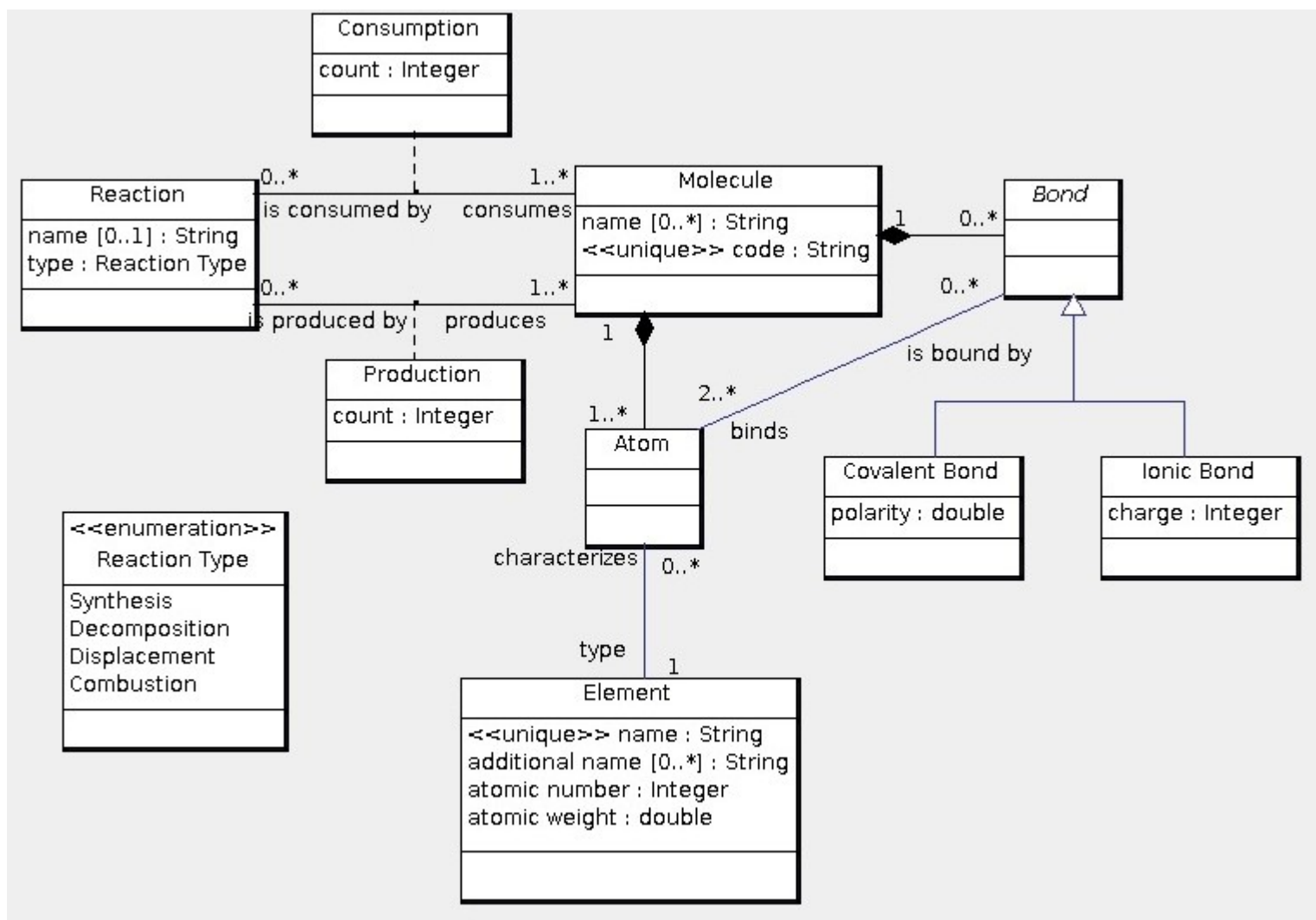
This level can have tens of millions of nodes

Multicolumn Indexes

- When an index has multiple columns, the index treats the key as a *single value*
 - The column values are essentially concatenated
 - In a tree index, the values are ordered lexicographically
- For example, in the Chemistry database, the Consumption table has two columns in its primary key

```
create table Consumption(  
  isConsumedBy int not null,  
  foreign key(isConsumedBy) references Reaction(id)  
    on update cascade on delete cascade,  
  consumes int not null,  
  foreign key(consumes) references Molecule(id)  
    on update cascade on delete cascade,  
  primary key(consumes, isConsumedBy) using btree,  
  count int not null  
);
```

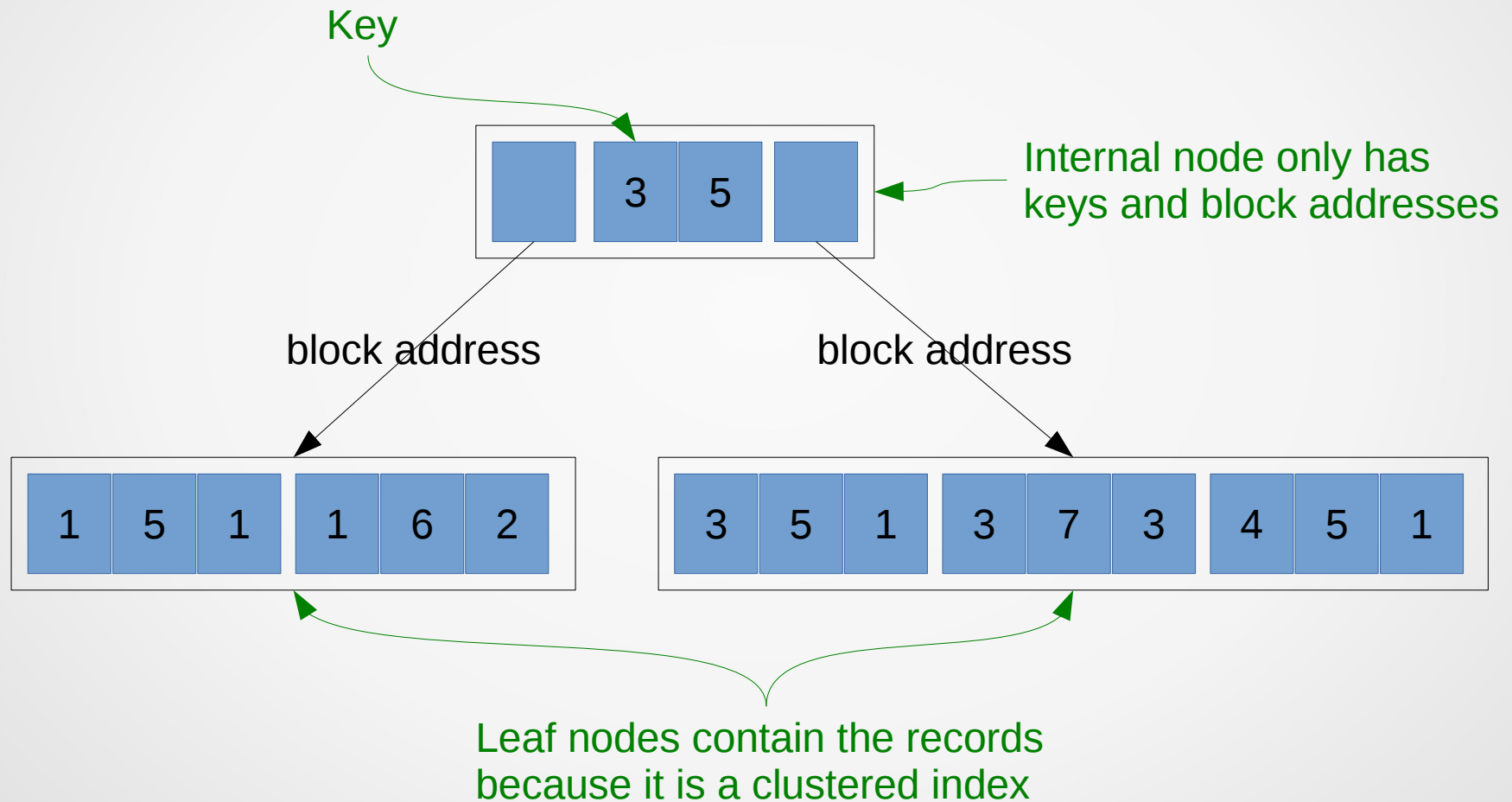
Chemistry database



Example Chemistry Data

| consumes | isConsumedBy | count |
|----------|--------------|-------|
| 1 | 5 | 1 |
| 1 | 6 | 2 |
| 3 | 5 | 1 |
| 3 | 7 | 3 |
| 4 | 8 | 1 |

Example B-Tree Primary Index



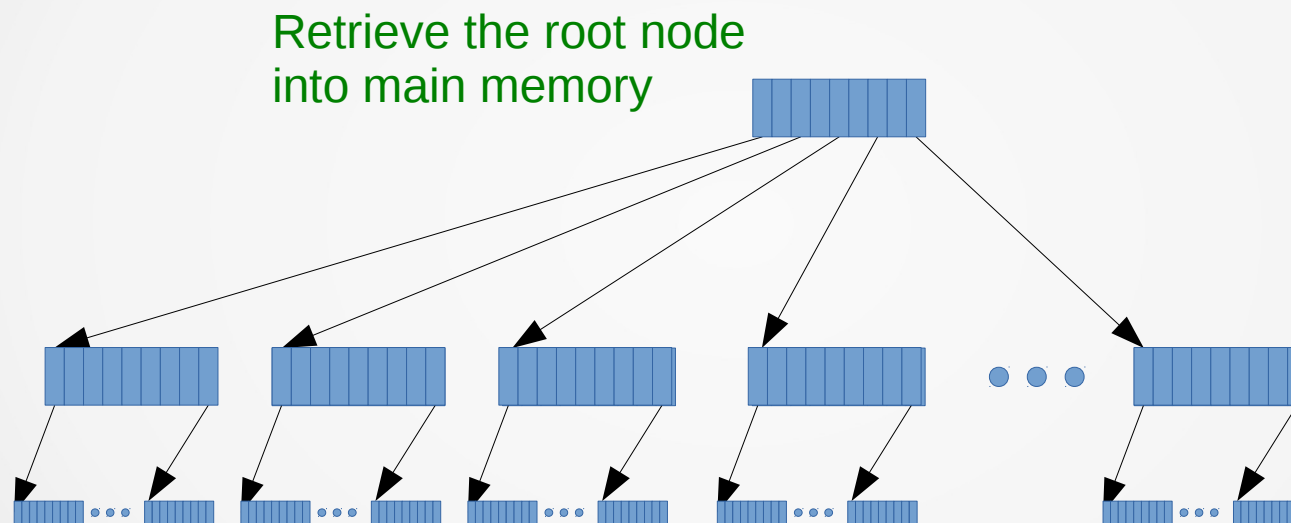
B-Tree Search Algorithm

The search algorithm starts at the root and proceeds down the tree

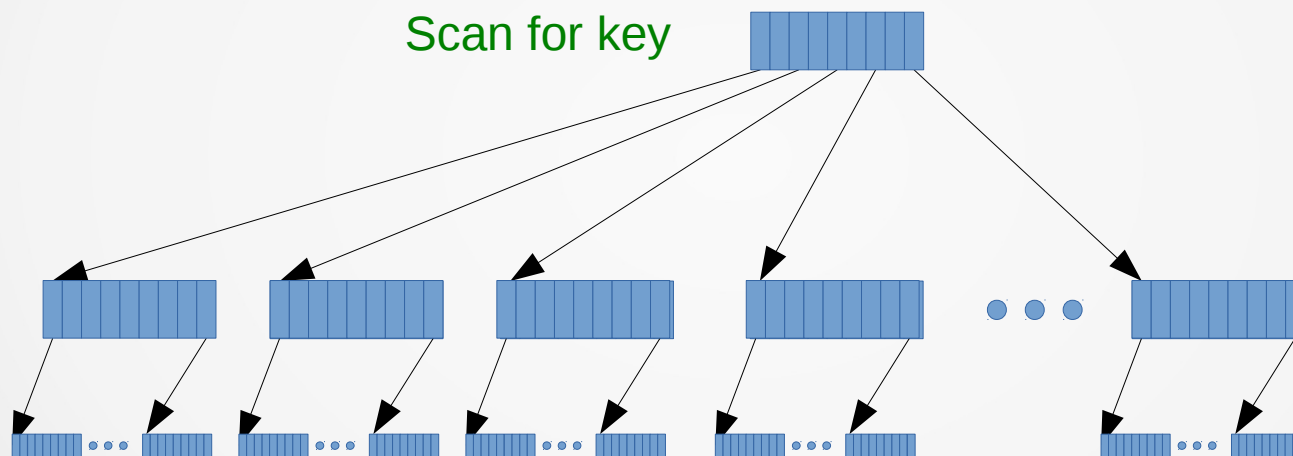
- At each level one must scan the node to find the address of the node on the next level to use
- Scanning a node is very quick compared to the time needed to copy a block from the storage device to memory

Because of the large number of addresses in a block, the number of levels that need to be scanned is reasonably small, although still larger than for a hash table

B-Tree Search Algorithm

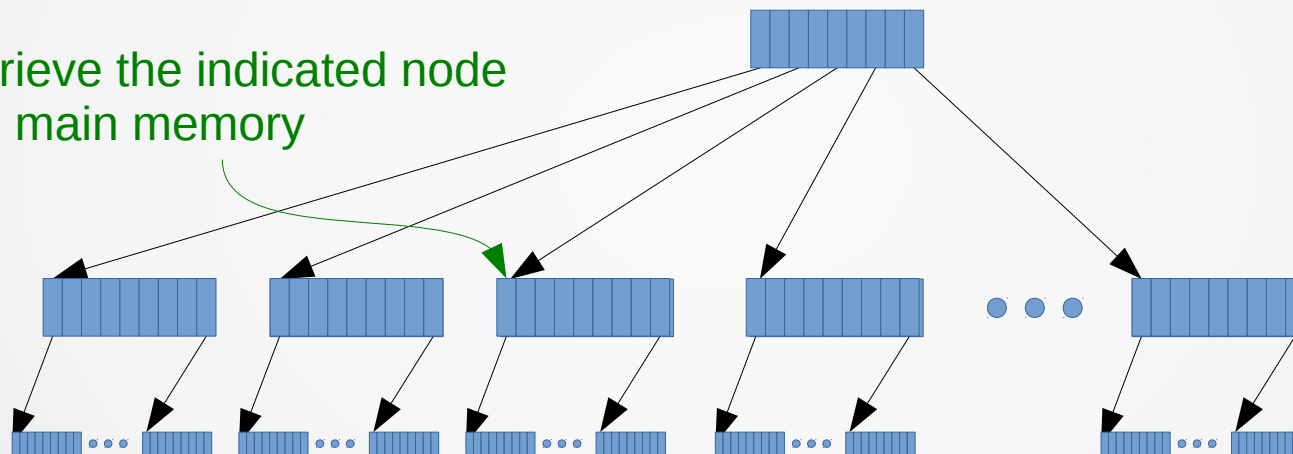


B-Tree Search Algorithm

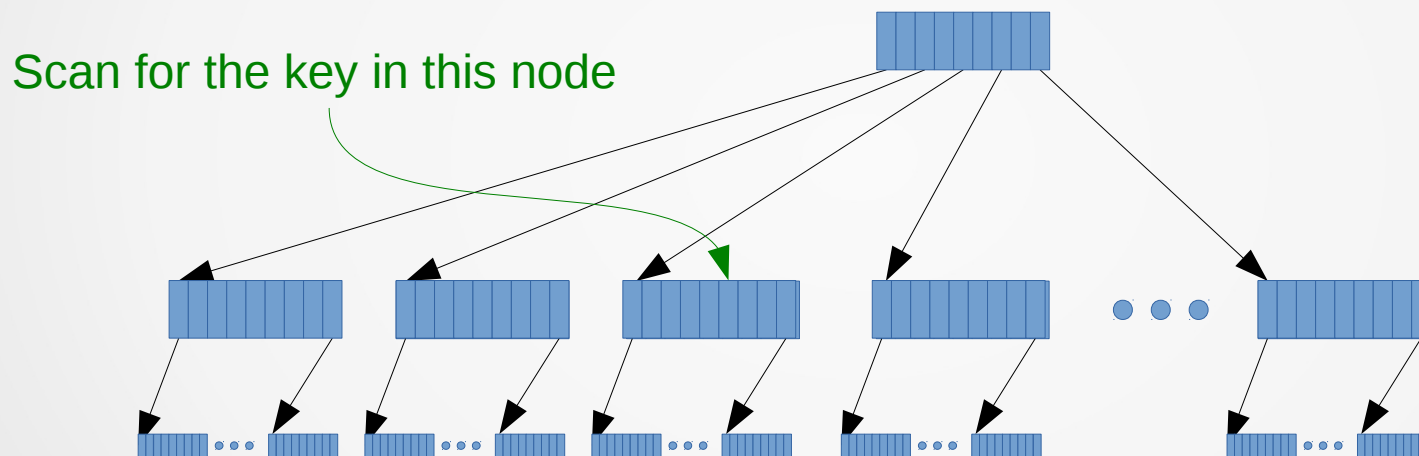


B-Tree Search Algorithm

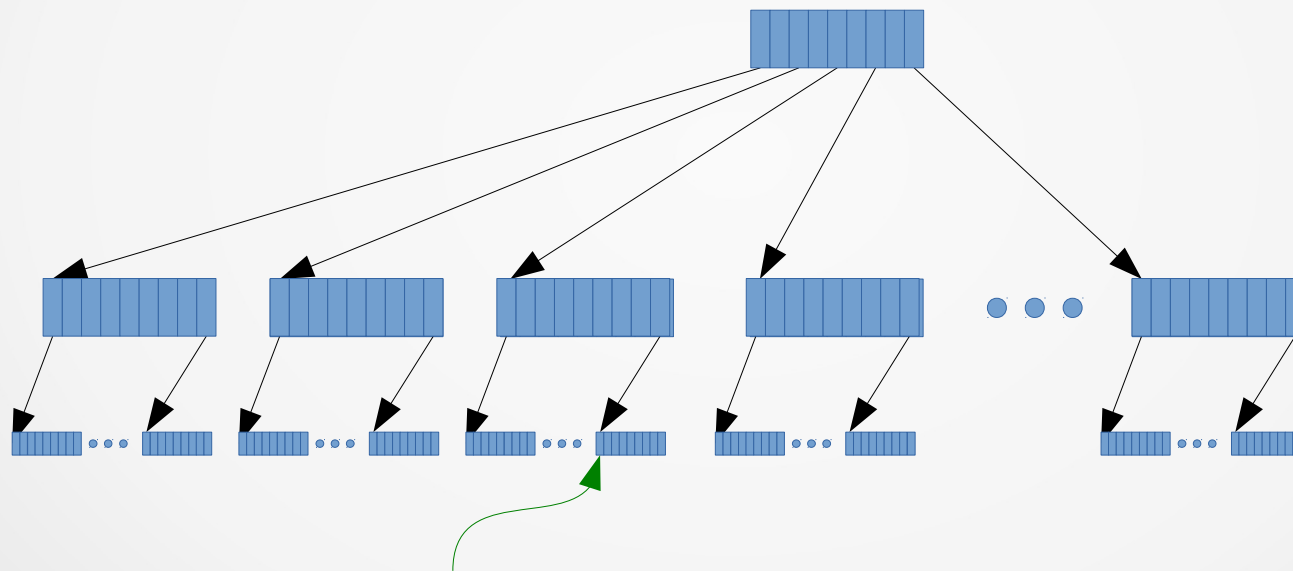
Retrieve the indicated node
into main memory



B-Tree Search Algorithm

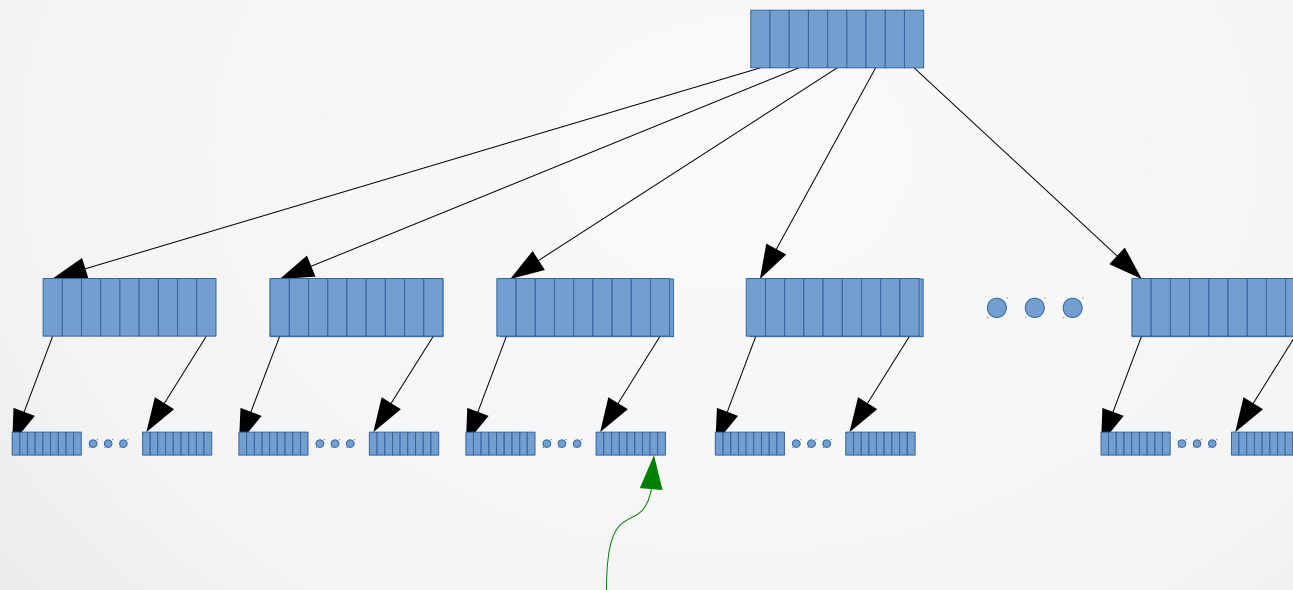


B-Tree Search Algorithm

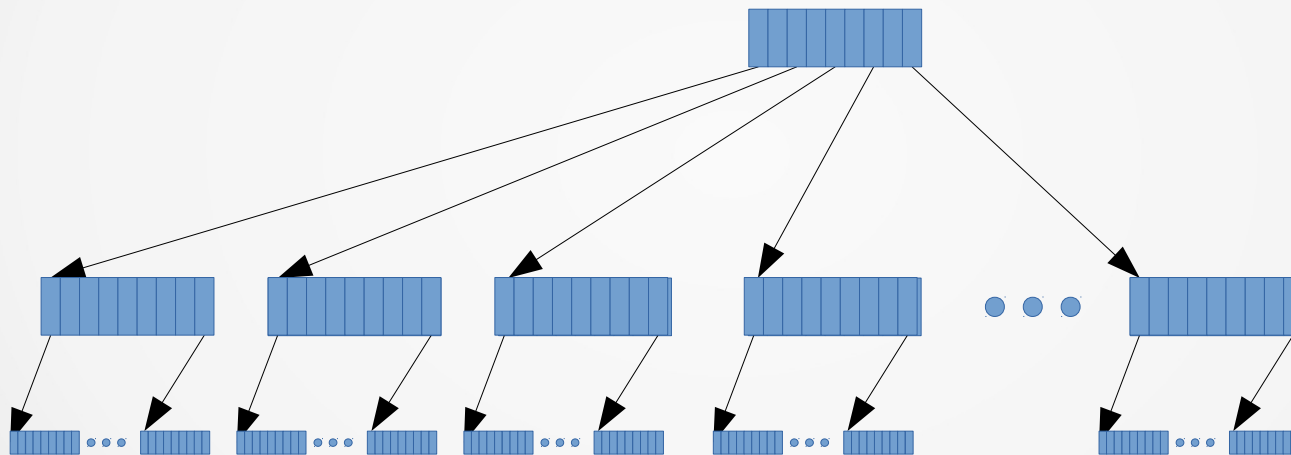


Retrieve the indicated node
into main memory

B-Tree Search Algorithm



B-Tree Search Algorithm



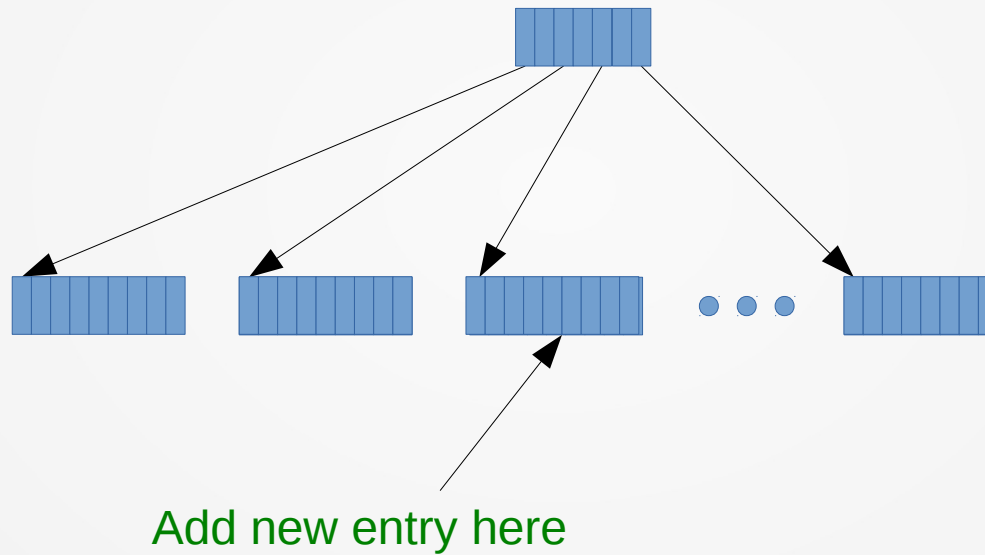
If the B-tree is clustered, then the record is in the leaf node
If the B-tree is secondary, then the leaf node has the address of the block that has the record

B-Tree Update

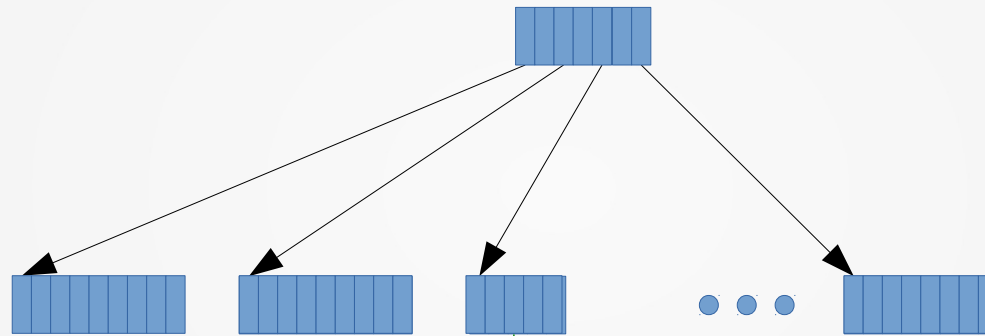
When a new entry is inserted into a node

- If there is space available, then it is added in the correct position
- If there is no space left
 - A new block is allocated on the storage device
 - The entries are distributed between the old node and the new node
 - A key (or partial key) that distinguishes the two nodes is inserted in the parent node
- The process continues recursively up the tree, if necessary
- If the root is reached and the root node is full, the original root node is split and a new root node is allocated

B-Tree Update



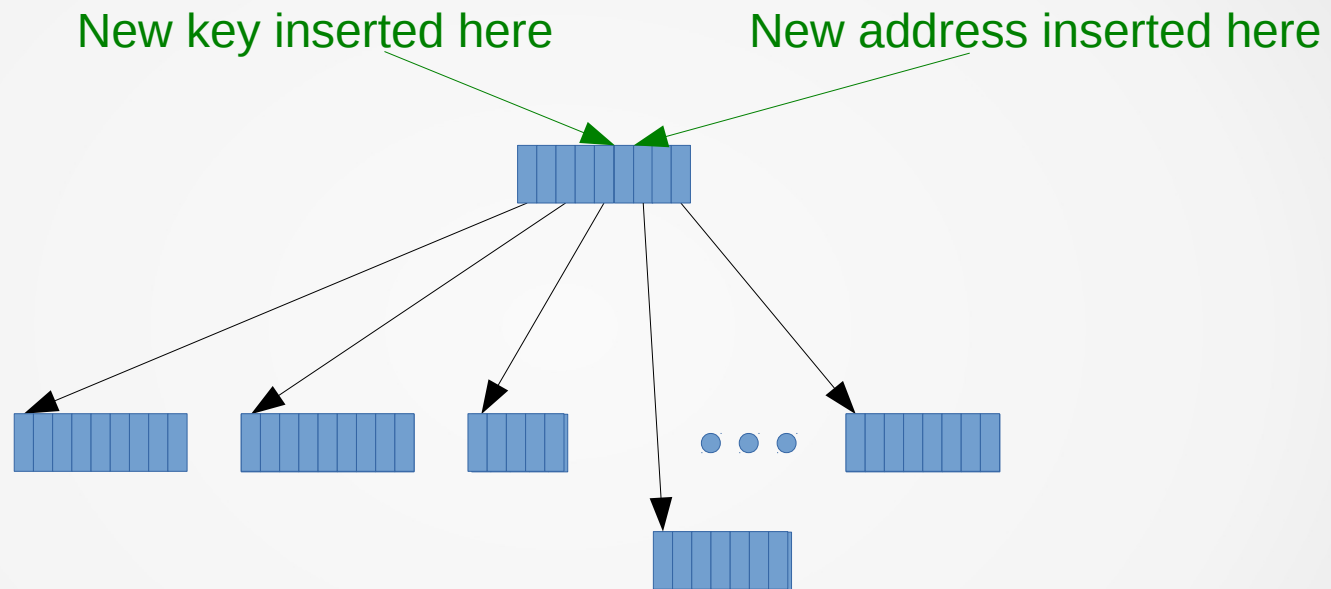
B-Tree Update



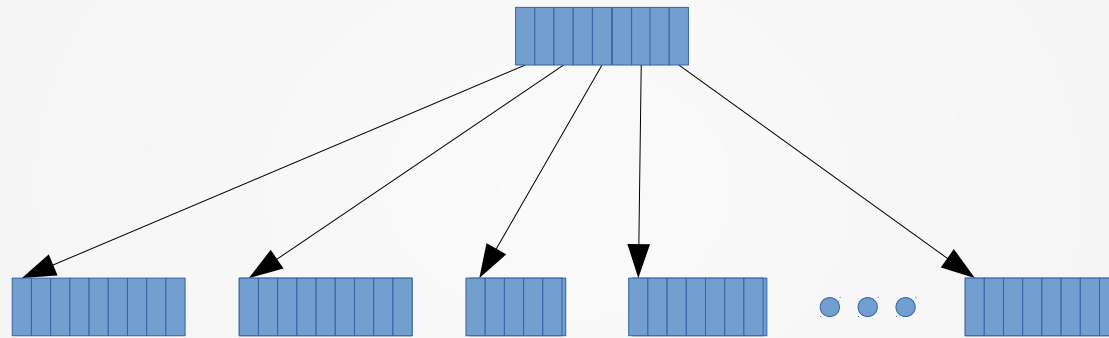
The original entries are distributed
between the old and new blocks

Newly allocated block

B-Tree Update



B-Tree Update



Updated B-tree

B-Tree Update

When an entry is deleted from a node

- In theory, one could try to merge nodes
- In practice, this is never done
- Eventually the B-tree must be reorganized, but this can be done without making the whole B-tree unavailable

Hash tables vs trees

- Requires a hash function
- Sensitive to the details of the hash function
- Keys are in random order
- Retrieval in constant time
- Lower storage overhead
- Requires a comparison function
- Not sensitive to the details of the comparison
- Keys are in sorted order
- Retrieval in logarithmic time
- Greater storage overhead

Performance Comparison

- Performance is dominated by the number of blocks that must be copied from the storage device
- A secondary index requires one more block access compared with a clustered index
 - A clustered index contains the records
 - A secondary index only contains addresses of records
- A hash index requires a single block access to retrieve a bucket
- A B-tree index requires a logarithmic number of block accesses but the base of the logarithm can be a thousand or more
 - Unless the number of records or keys is billions or more, the number of block accesses will be around 2 or 3.

Performance Comparison

| Search | Hash | Btree |
|-----------------|------------------|-----------------------|
| Clustered Index | 1 block access | 2 or 3 block accesses |
| Secondary Index | 2 block accesses | 3 or 4 block accesses |

| Update | Hash | Btree |
|-----------------|------------------|-----------------------|
| Clustered Index | 2 block accesses | 3 or 4 block accesses |
| Secondary Index | 3 block accesses | 4 or 5 block accesses |

Index Design Problem

```
create table Company(  
  id int primary key,  
  name varchar(500) not null,  
  product varchar(500)  
);  
create table Person(  
  id int primary key,  
  name varchar(200) not null unique,  
  worksFor int,  
  foreign key (worksFor) references Company(id)  
    on update cascade on delete no action  
);  
create table PersonEmail(  
  person int not null,  
  foreign key (person) references Person(id)  
    on update cascade on delete cascade,  
  email varchar(200) not null,  
  primary key(person, email)  
);
```

The most common query is to look up the name of the company that a person works for.

Write your solution on a piece of paper or on your laptop.



Assignment #6