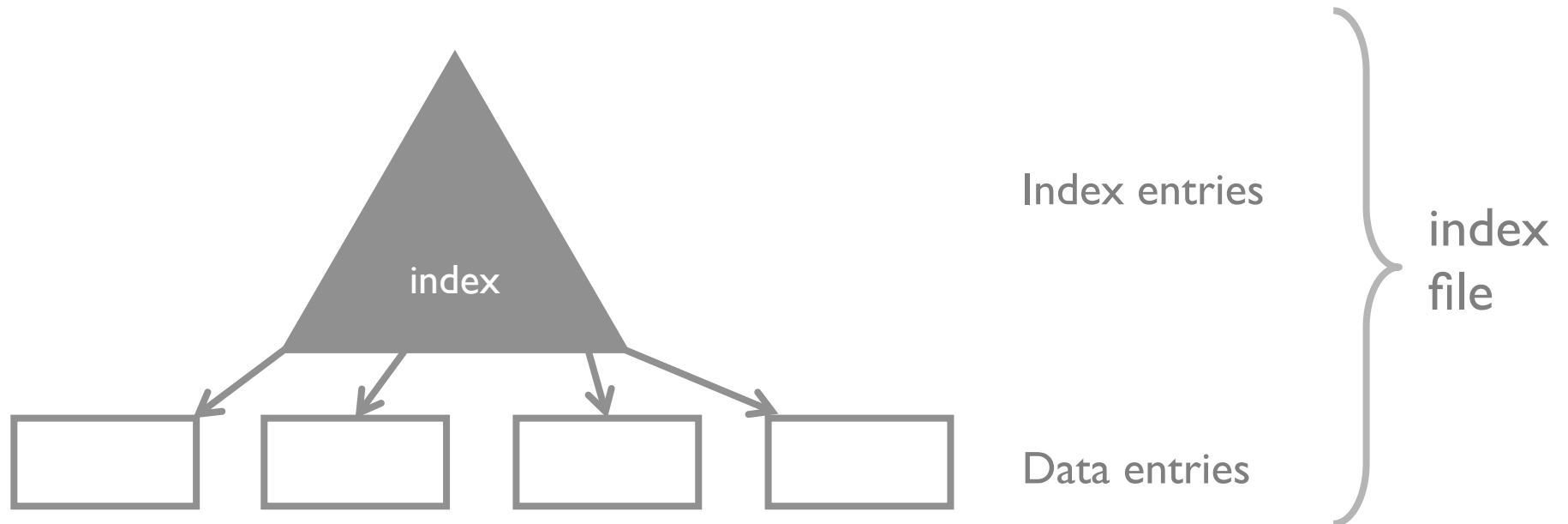


L20

Indexing Continued

High level index structure



What is a data entry?

actual data record

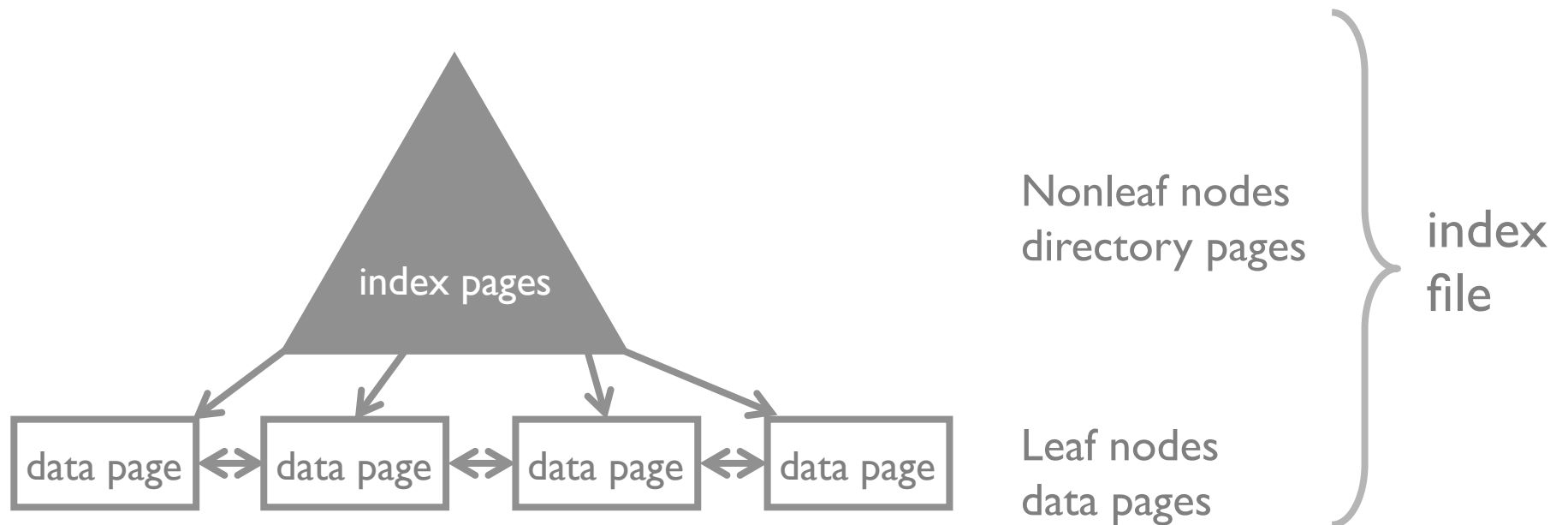
<search key value, rid>

Tradeoffs

directly access tuple.

compact, fixed size entries

B+ Tree Index



Node = Page

Equality and range queries

Self balancing

Leaf nodes are connected

Disk optimized

B+ Tree on (age)

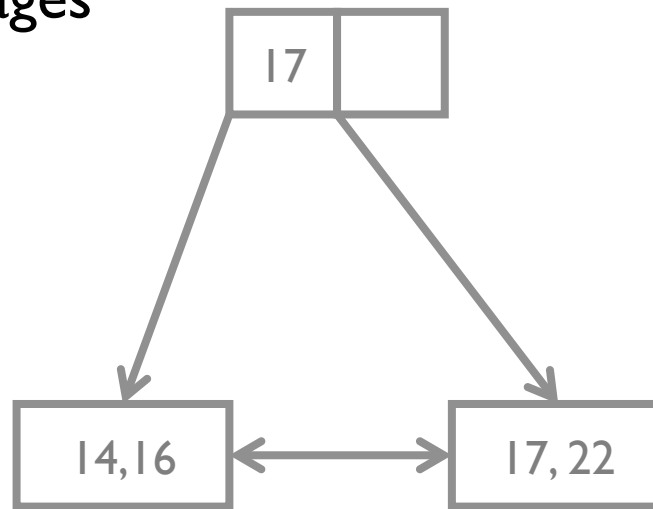
Non-leaf directory pages

m index entries

m+1 pointers

Leaf Data pages

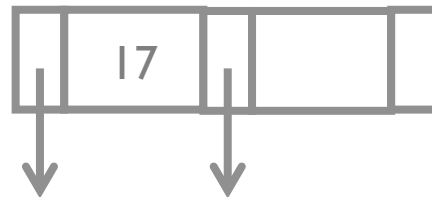
data entries/tuples



index & data
page contents
are in order

Query: `SELECT * WHERE age= 14`

directory page



Index Only Queries: B+ Tree on (age)

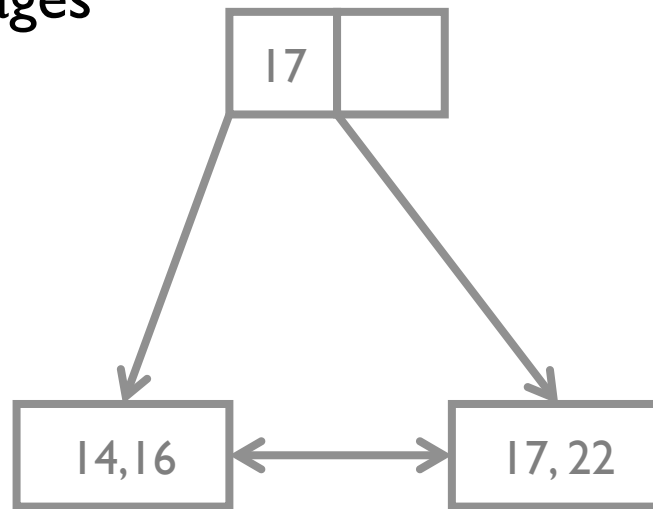
Non-leaf directory pages

m index entries

m+1 pointers

Leaf Data pages

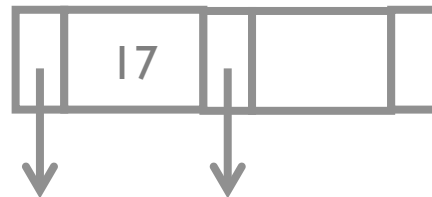
data entries/tuples



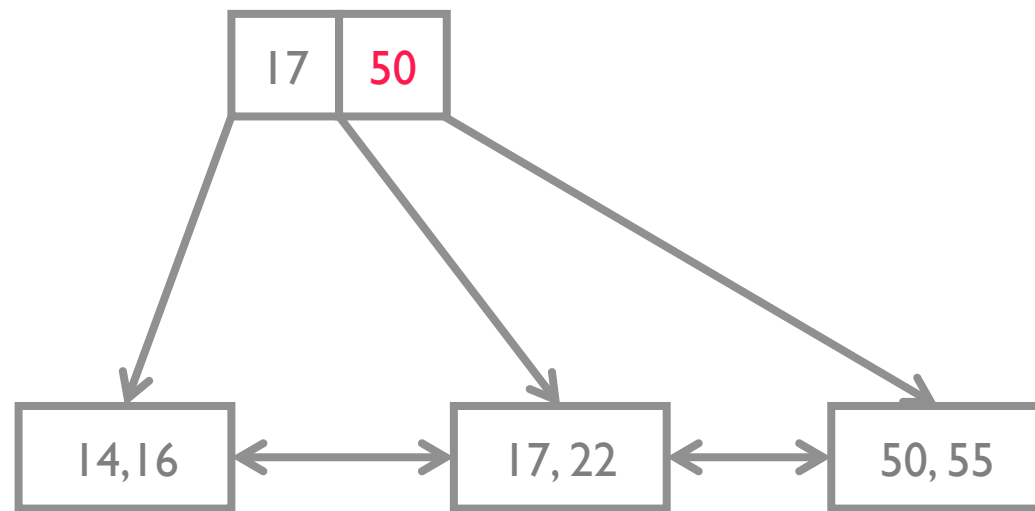
index & data
page contents
are in order

Query: **SELECT age WHERE age = 14**
(index only!)

directory page



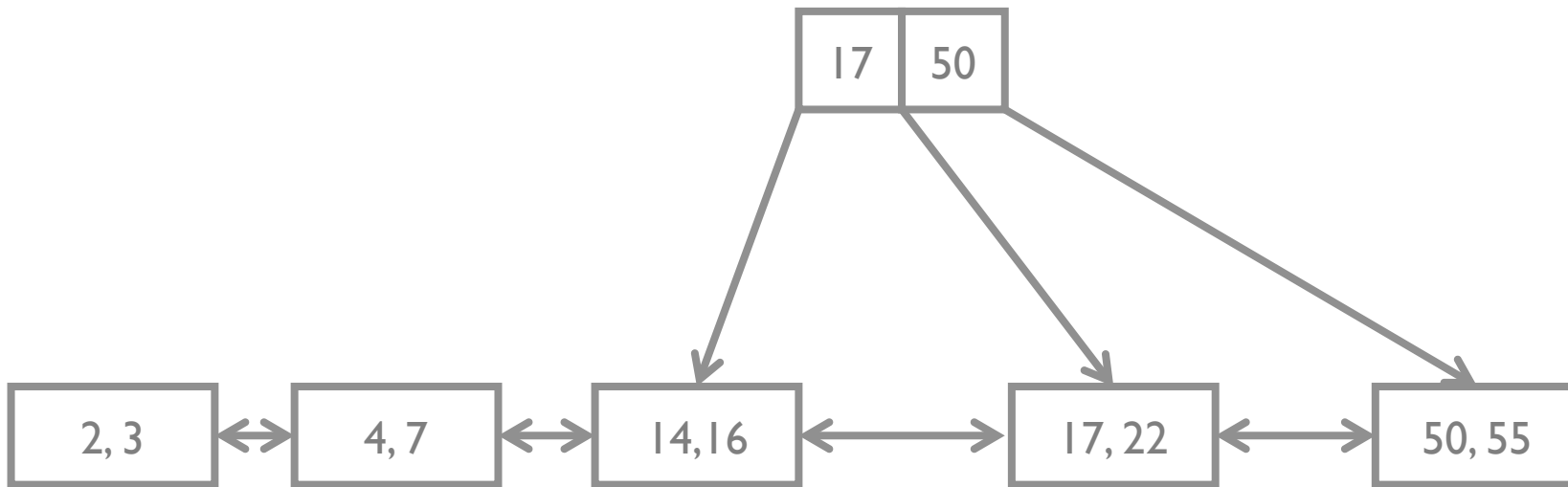
B+ Tree on (age)



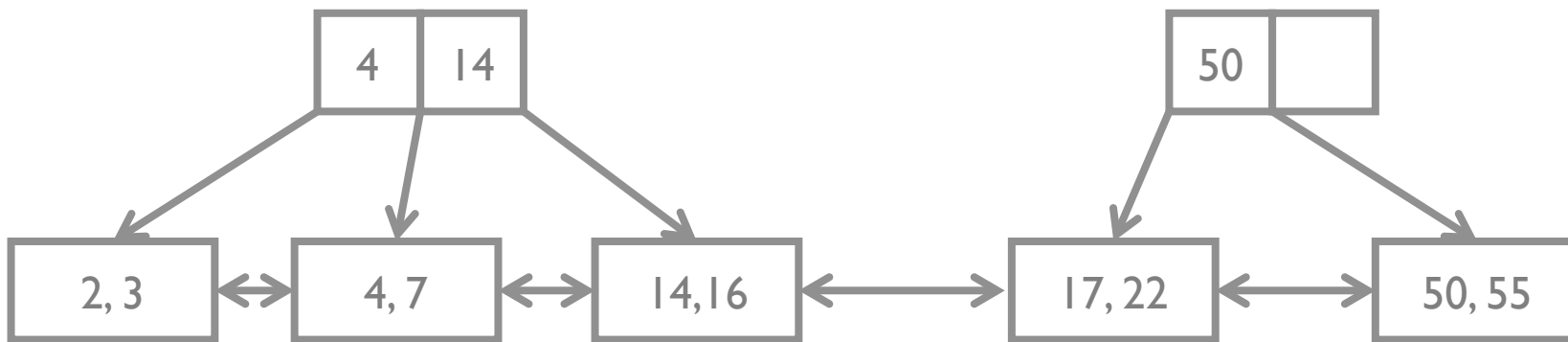
Query: `SELECT * WHERE age = 50`



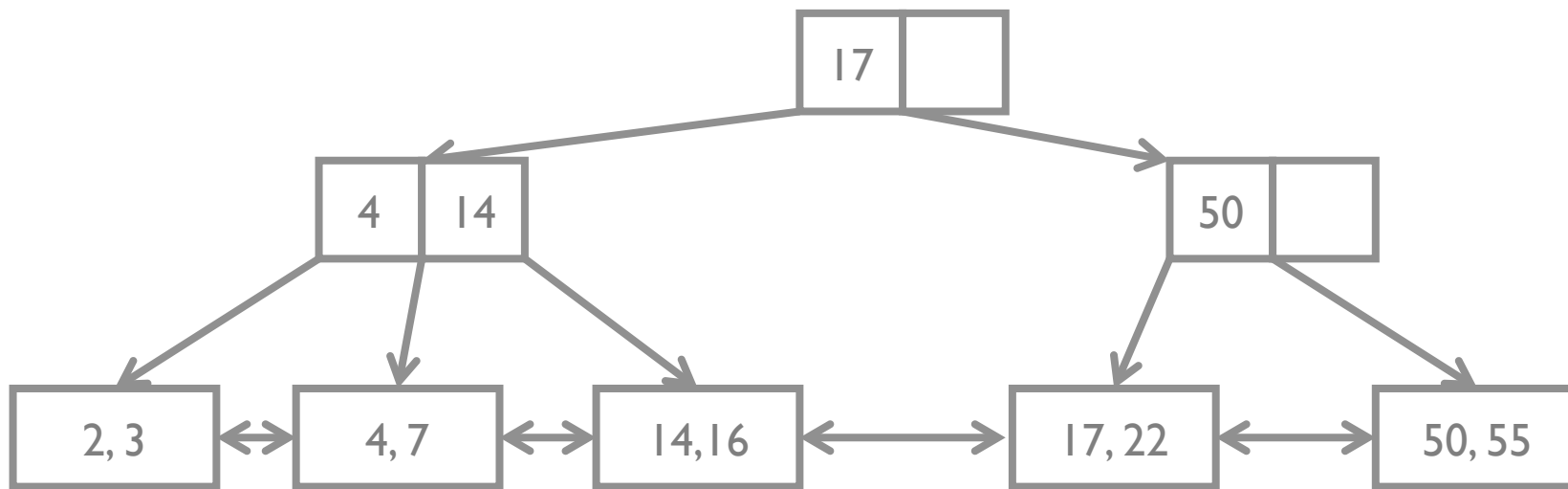
B+ Tree on (age)



B+ Tree on (age)

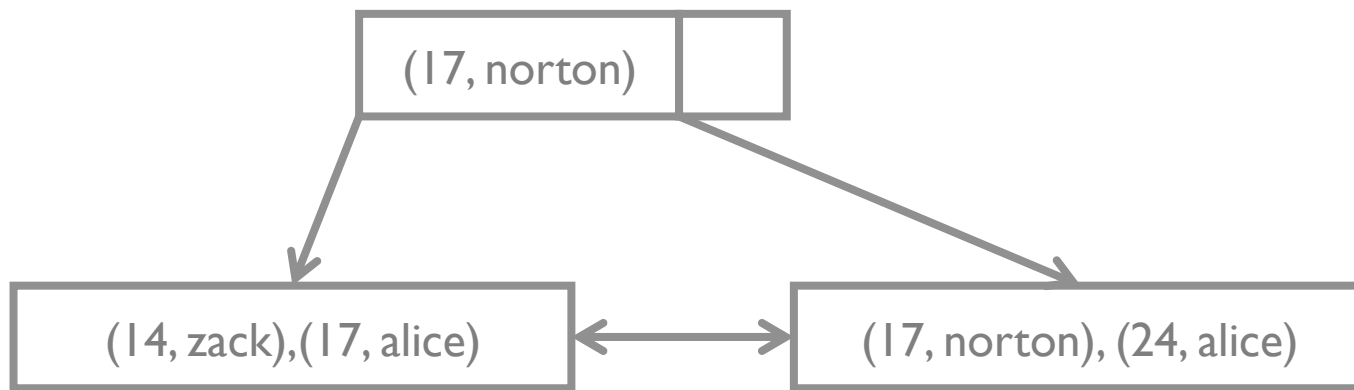


B+ Tree on (age)



Query: `SELECT * WHERE age > 15`

B+ Tree on (age, name)



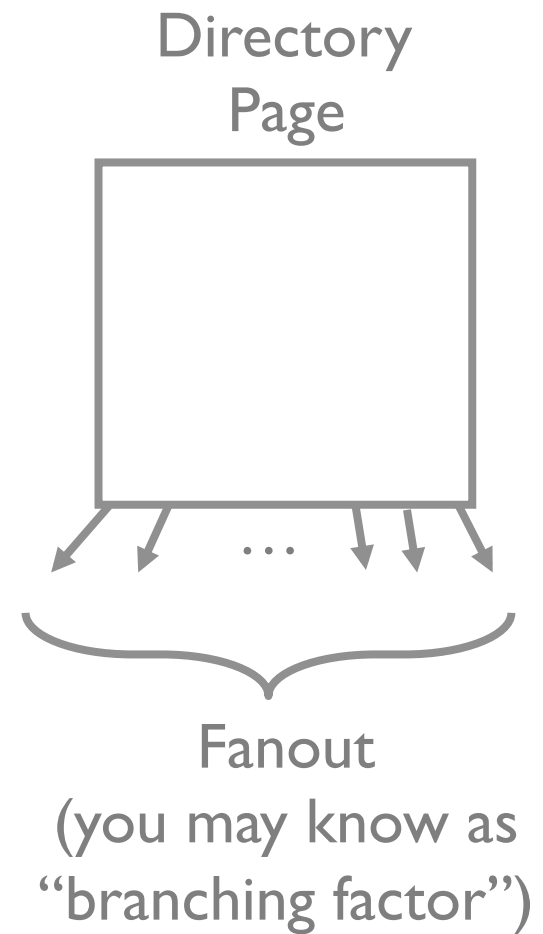
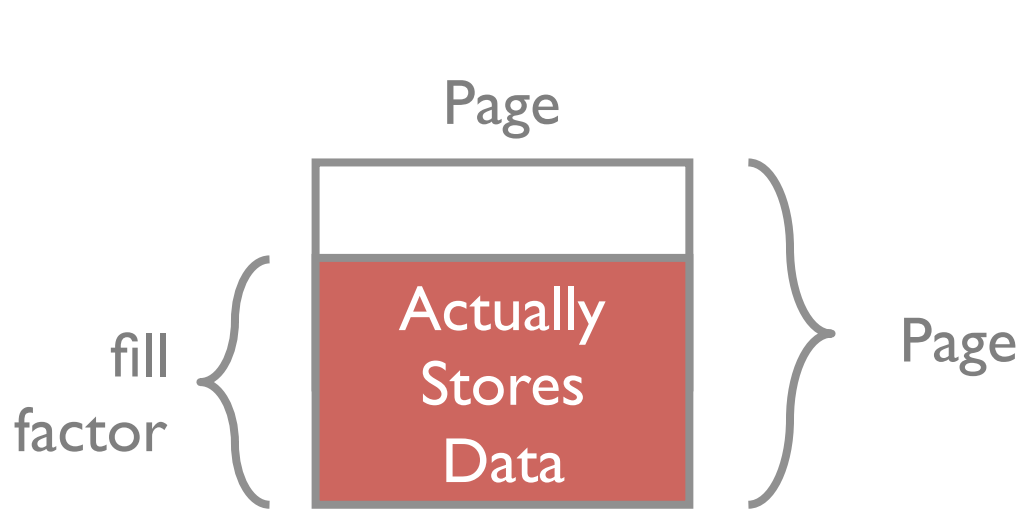
How do the following queries use the index on (age, name)?

SELECT age WHERE age = 14

SELECT * WHERE age < 18 AND name < 'monica'

SELECT age WHERE name = 'bobby'

Terminology



Some numbers (8kb pages)

How many levels?

fill-factor: ~66%

~300 entries per directory page

height 2: $300^3 \sim 27$ Million entries

height 3: $300^4 \sim 8.1$ Billion entries

Top levels often in memory

height 2 only 300 pages ~2.4MB

height 3 only 90k pages ~750MB

Hash Index on age

Hash function

$$h(v) = v \% 3$$

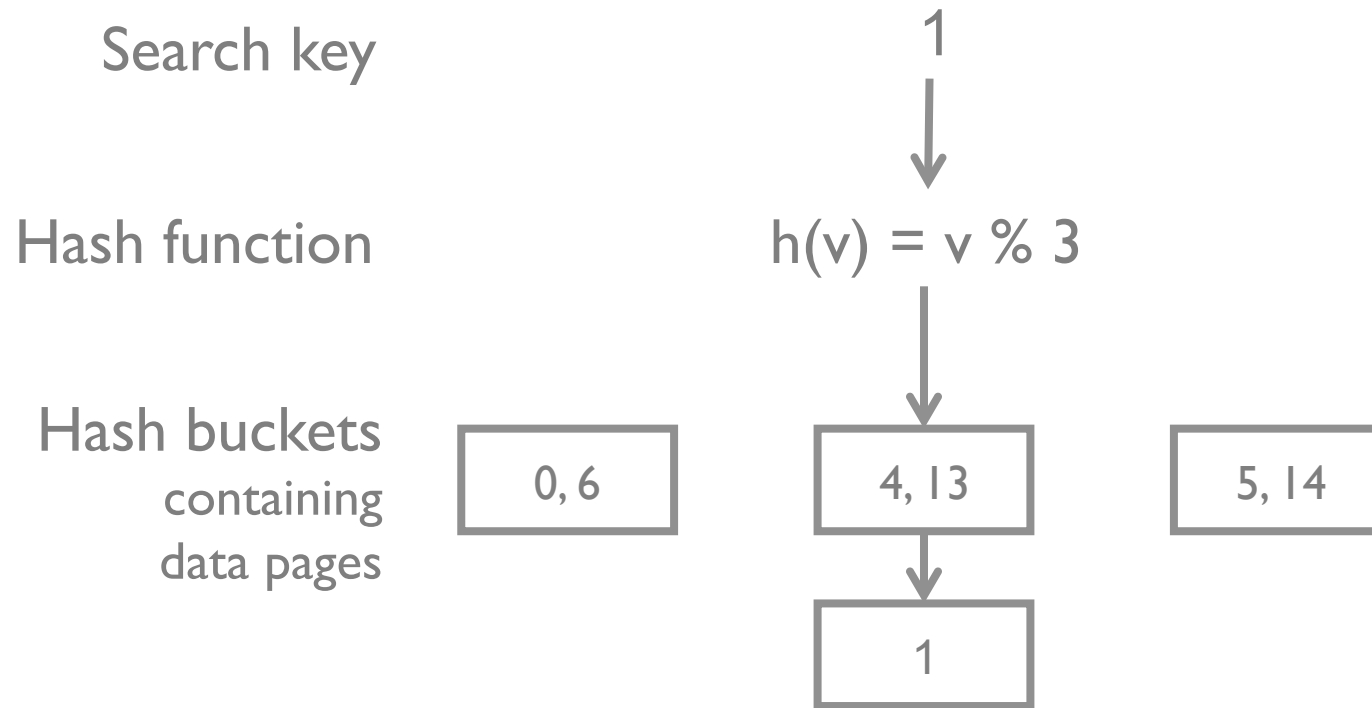
Hash buckets
containing
data pages

0, 6

4, 13

5, 14

INSERT Hash Index on age



INSERT Hash Index on age

Search key

11

Hash function

$$h(v) = v \% 3$$

Hash buckets
containing
data pages

0, 6

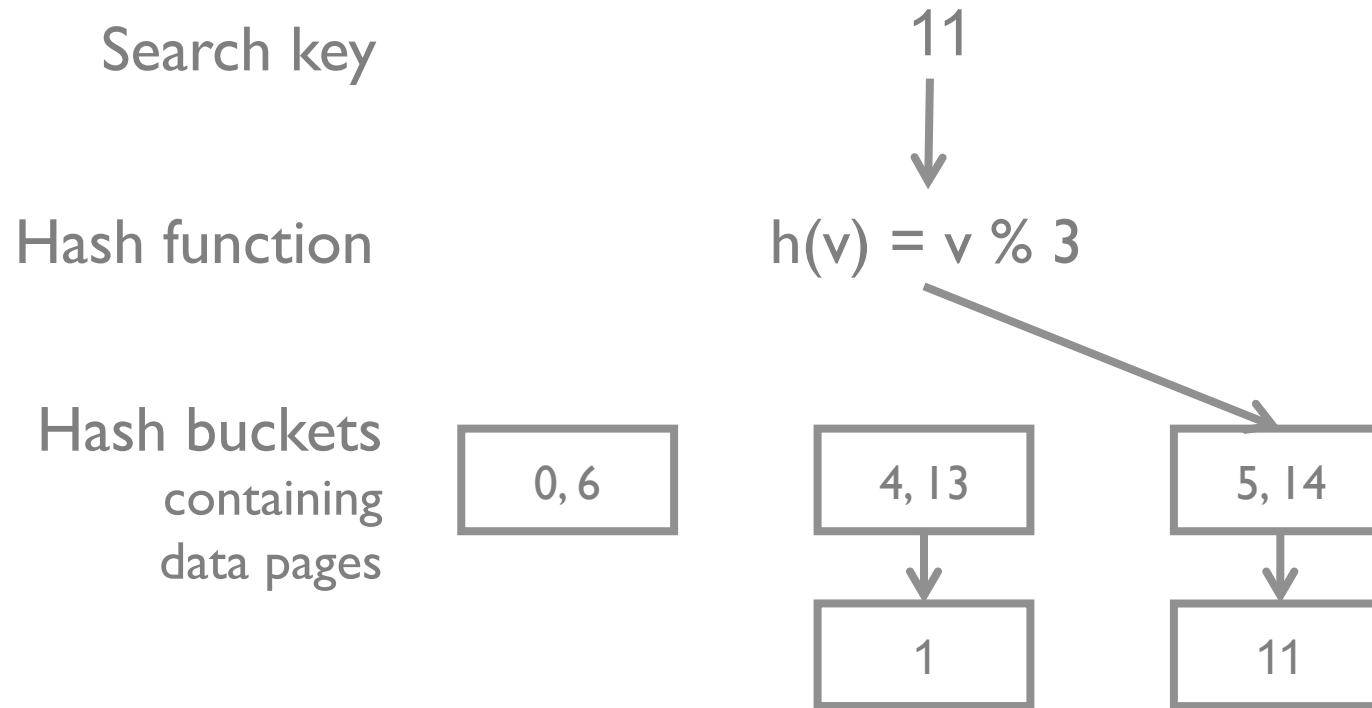
4, 13

5, 14

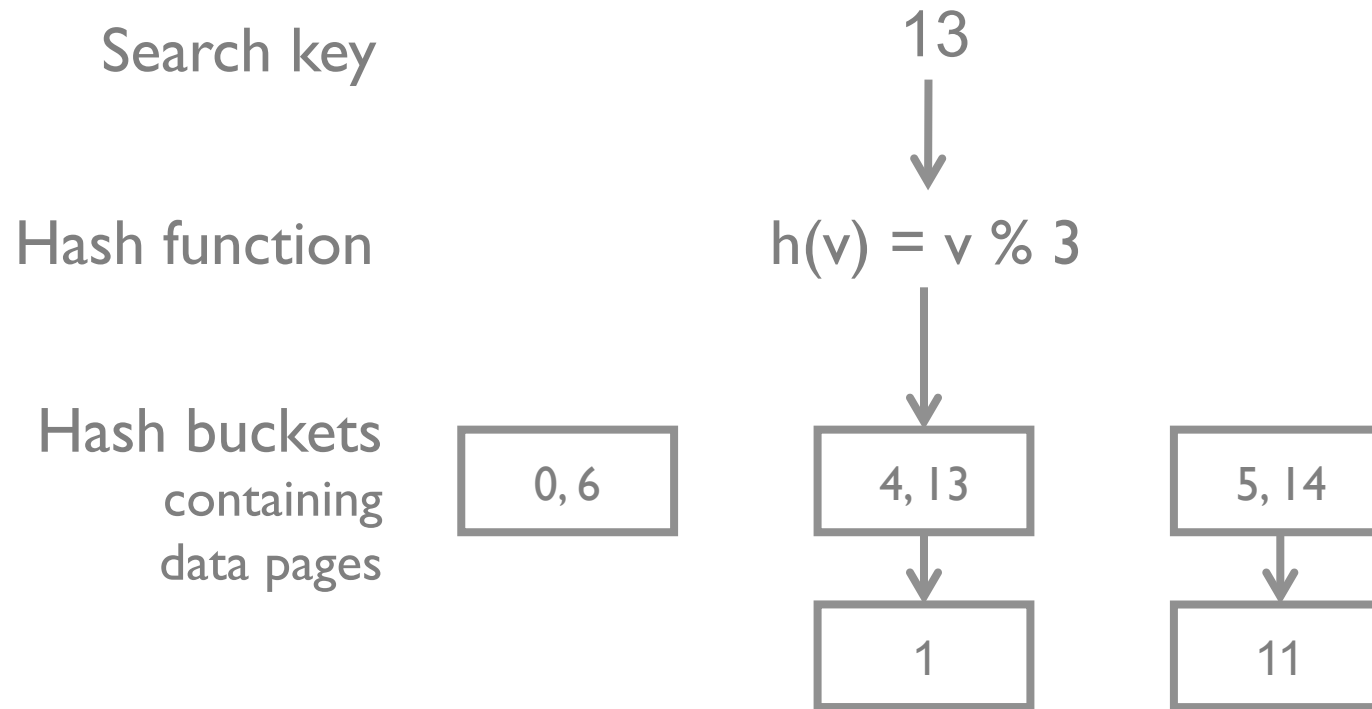
1



INSERT Hash Index on age



SEARCH Hash Index on age



Good for equality selections

Index = data pages + overflow data pages

Hash function $h(v)$ takes as input the *search key*

Costs

Three file types

Heap, B+ Tree, Hash

Operations we care about

Scan all data `SELECT * FROM R`

Equality `SELECT * FROM R WHERE x = I`

Range `SELECT * FROM R WHERE 10 < x and x < 50`

Insert record

Delete record

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything				
Equality				
Range				
Insert				
Delete				

- B** # *data* pages
- D** time to read/write page
- M** # pages in range query

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything	BD			
Equality	0.5BD (avg)			
Range	BD			
Insert	2D			
Delete	Search + D			

Heap File

equality on a key. How many results?

- B** # data pages
- D** time to read/write page
- M** # pages in range query

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything	BD	BD		
Equality	0.5BD	$D(\log_2 B)$		
Range	BD	$D(\log_2 B + M)$		
Insert	2D	Search + BD		
Delete	Search + D	Search + BD		

Heap File

equality on a key. How many results?

Sorted File

files compacted after deletion

B # data pages

D time to read/write page

M # pages in range query

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything	BD	BD	1.25BD	
Equality	0.5BD	$D(\log_2 B)$	$D(\log_{80} B + 1)$	
Range	BD	$D(\log_2 B + M)$	$D(\log_{80} B + M)$	
Insert	2D	Search + BD	$D(\log_{80} B + 2)$	
Delete	Search + D	Search + BD	$D(\log_{80} B + 2)$	

Heap File

equality on a key. How many results?

Sorted File

files compacted after deletion

B+ Tree

100 entries/directory page

80% fill factor

B # data pages

D time to read/write page

M # pages in range query

	Heap File	Sorted Heap	B+ Tree	Hash
Scan everything	BD	BD	1.25BD	1.25BD
Equality	0.5BD	$D(\log_2 B)$	$D(\log_{80} B + 1)$	D
Range	BD	$D(\log_2 B + M)$	$D(\log_{80} B + M)$	1.25BD
Insert	2D	Search + BD	$D(\log_{80} B + 2)$	2D
Delete	Search + D	Search + BD	$D(\log_{80} B + 2)$	2D

Heap File

equality on a key. How many results?

Sorted File

files compacted after deletion

B+ Tree

100 entries/directory page

80% fill factor

Hash index

no overflow

80% fill factor

B # data pages

D time to read/write page

M # pages in range query

How to pick?

Depends on your queries (workload)

Which relations?

Which attributes?

Which types of predicates ($=$, $<$, $>$)

Selectivity

Insert/delete/update queries? how many?

Naïve Algorithm

```
get query workload
group queries by type
for each query type in order of importance
    calculate best cost using current indexes
    if new index IDX will further reduce cost
        create IDX
```

Why not create every index?

- updates are slower: upkeep costs
- takes up space

High level guidelines

Check the WHERE clauses

- attributes in WHERE are search/index keys

- equality predicate → hash index

- range predicate → tree index

Multi-attribute search keys supported

- order of attributes matters for range queries

- may enable queries that don't look at data pages (*index-only*)

Summary

Design depends on economics, access cost ratios

Disk still dominant wrt cost/capacity ratio

Many physical layouts for files

- same APIs, difference performance

- remember physical independence

Indexes

- Structures to speed up read queries

- Multiple indexes possible

- Decision depends on workload

Things to Know

- How a hard drive works and its major performance characteristics
- The storage hierarchy and rough performance differences between RAM, SSD, Hard drives
- What files, pages, and records are, and how they are different than the UNIX model
- Heap File data structure
- B+ tree and Hash indexes
- Performance characteristics of different file organizations

L20

Query Execution & Optimization

Steps for a New Application

Requirements

what are you going to build?

Conceptual Database Design

pen-and-pencil description

Logical Design

formal database schema

Schema Refinement:

fix potential problems, normalization

Physical Database Design

optimize for speed/storage

Optimization

App/Security Design

prevent security problems

Recall

Relational algebra

equivalence: multiple stmts for same query
some statements (much) faster than others

Which is faster?

- a. $\sigma_{v=1}(R \times T)$
- b. $\sigma_{v=1}(R) \times T$

$|R| = |T| = 10$ pages. 100? 1M?

unique values in R = 1. 100? 1M?  selectivity!

Overview of Query Optimization

SQL → query plan

How plans are executed

Some implementations of operators

Cost estimation of a plan

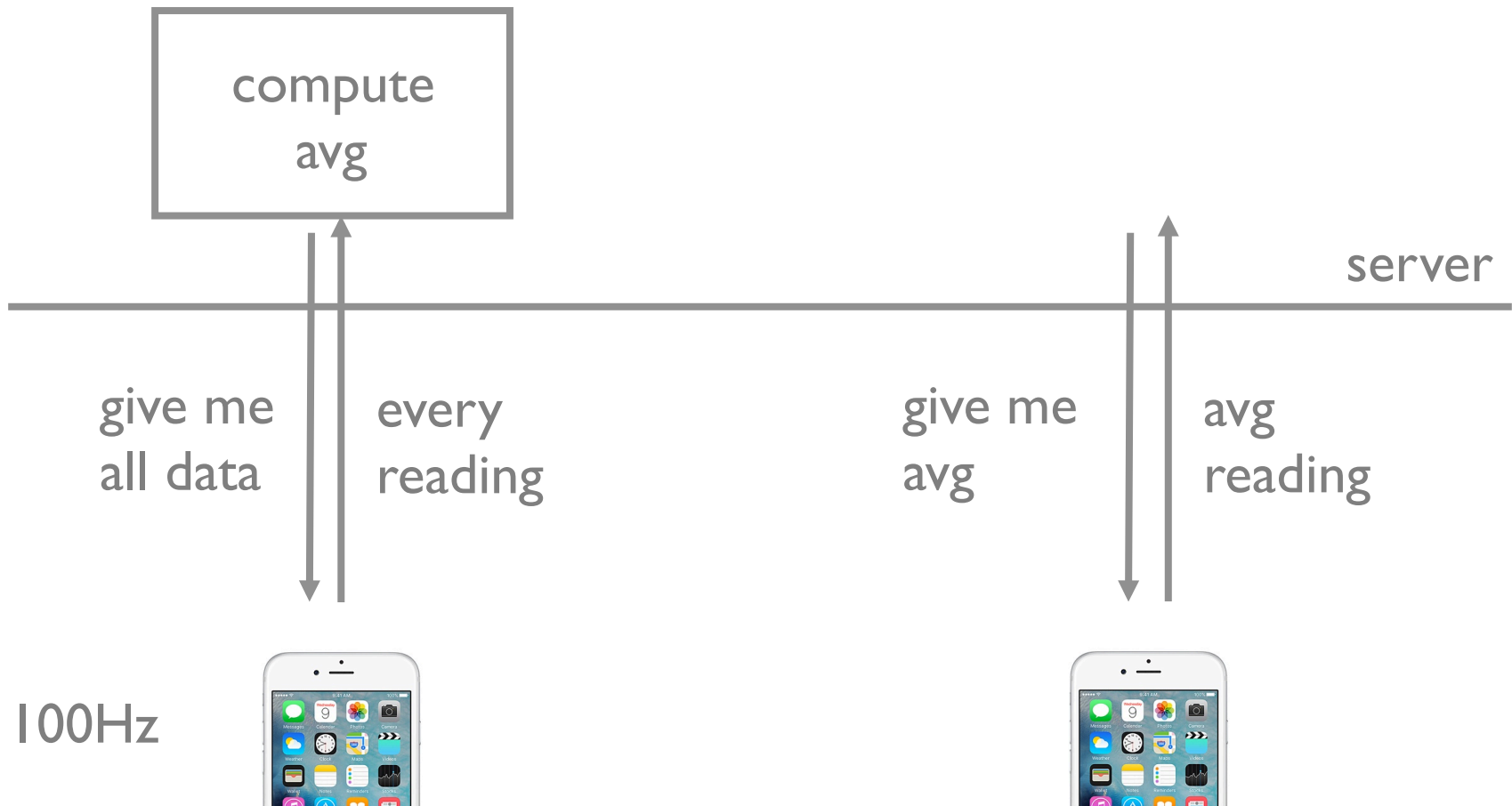
Selectivity

System R dynamic programming

All ideas from System R's “Selinger Optimizer” 1979

iPhones as a database

“avg acceleration over the past hour”



SQL \rightarrow Query Plan

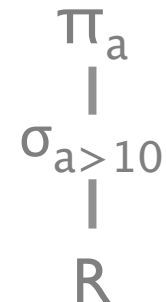
SELECT a FROM R

$\pi_a(R)$



SELECT a FROM R
WHERE a > 10

$\pi_a(\sigma_{a>10}(R))$



SELECT a
FROM R JOIN S
ON R.b = S.b

$\pi_a(\bowtie_b(R, S))$

