Indexing

- improving DB performance
- Reducing search space.

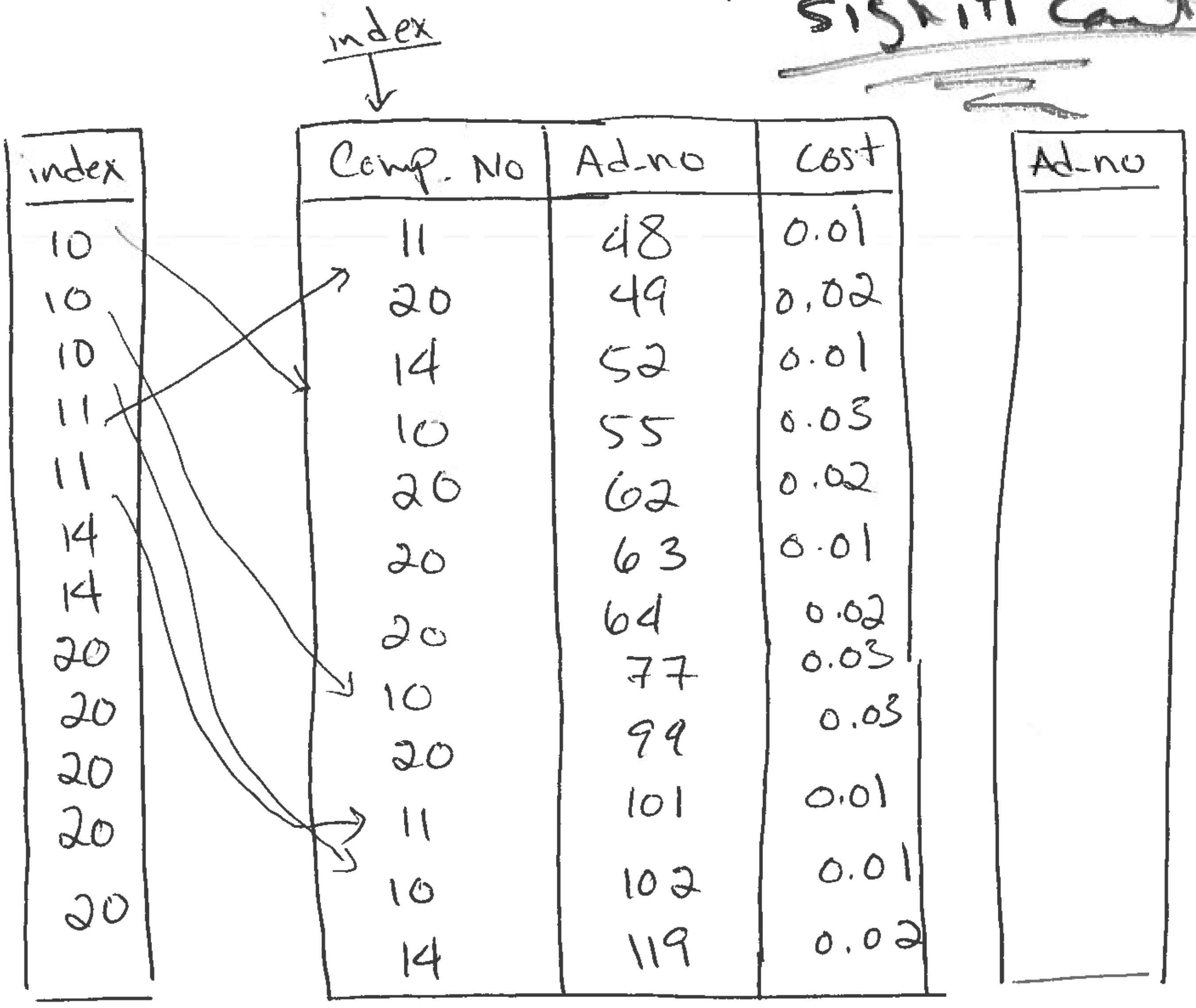
Example

→ text Book (1000 pages)
→ Find instances of a particular word
  i.e concentrate

* w/o index → takes a long time
  - start page 1
  - Read first word and every word (top to bottom)
  - ....
  - page 1000
  - Read first word and every word (top to bottom)

* w/ index → Jump directly to specific pages
  → Find c
    → Find co
      ---   (pge Nos)

Perf improvement using indexing is <u>significant</u>



index
10
10
10
11
11
14
14
20
20
20
20
20

Comp. No	Ad-no	cost
11	48	0.01
20	49	0.02
14	52	0.01
10	55	0.03
20	62	0.02
20	63	0.01
20	64	0.02
10	77	0.03
20	99	0.03
11	101	0.01
10	102	0.01
14	119	0.02

Ad-no

↑ separate index structure

index (comp-no, Ad-no)
composite

- Hash-based indexing
- Tree-based Indexing

Deciding when to index

① Read vs write
☺ ☹

② cardinality of attributes
~~runtime~~ Design-time

→ unique values
→ total observation
} cardinality

example → person DB
(1x10^6)
people

att → gender ⟨ M → index based on gender is not good
2 states ⟨ F

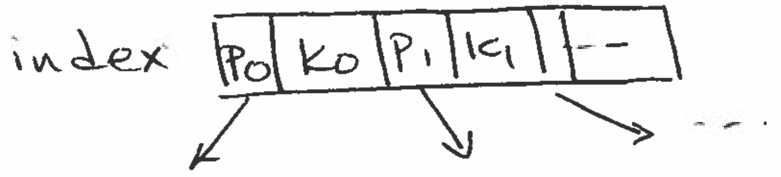
$$\text{Cardinality} = \frac{\text{total diff}}{\text{total}_{\text{records}}} = \frac{500k}{1 \times 10^{6}} = 0.5$$

③ search / Filter likelihood
→ how Frequently a Query is executed
↳ index

index → Data structure

→ organizes data records on disk

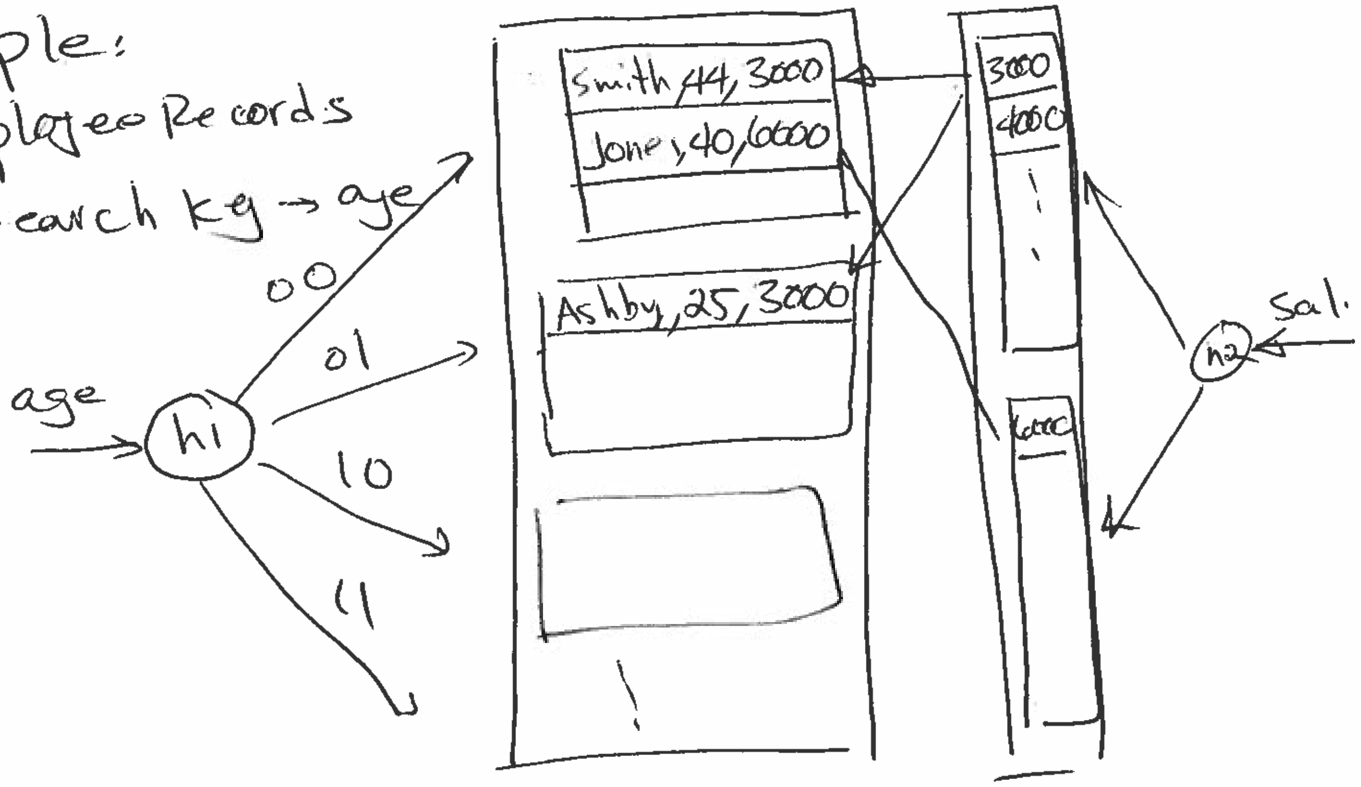
→ Fetching/retrieval is optimized



Hash-based indexing

Approach
— records grouped into buckets
Primary page — additional pages in a chain
— hash function: determines the bucket the record belongs.

Example:
Employee Records
search key → age

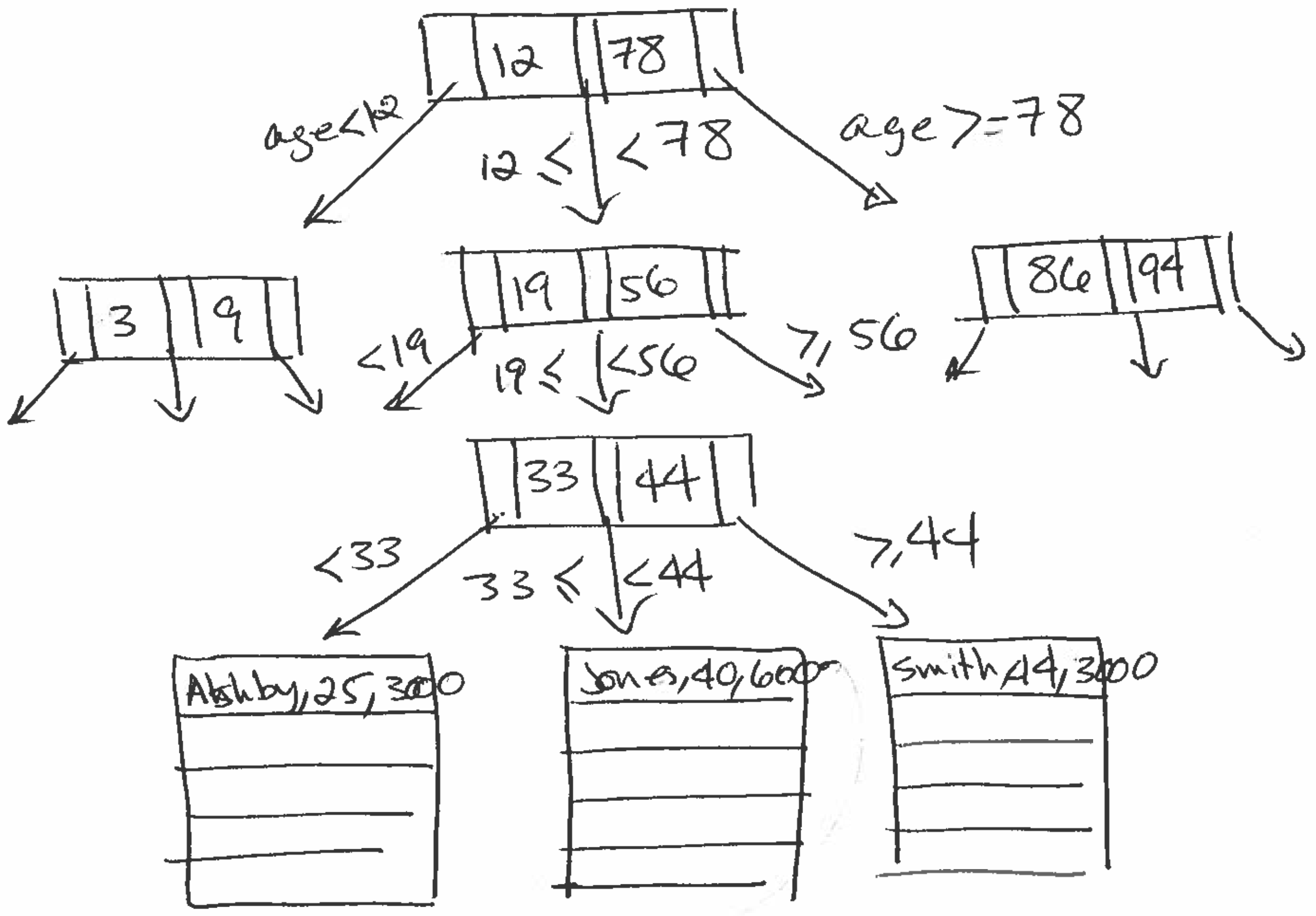


Tree-based Indexing

→ Organizes records in a tree like structure.

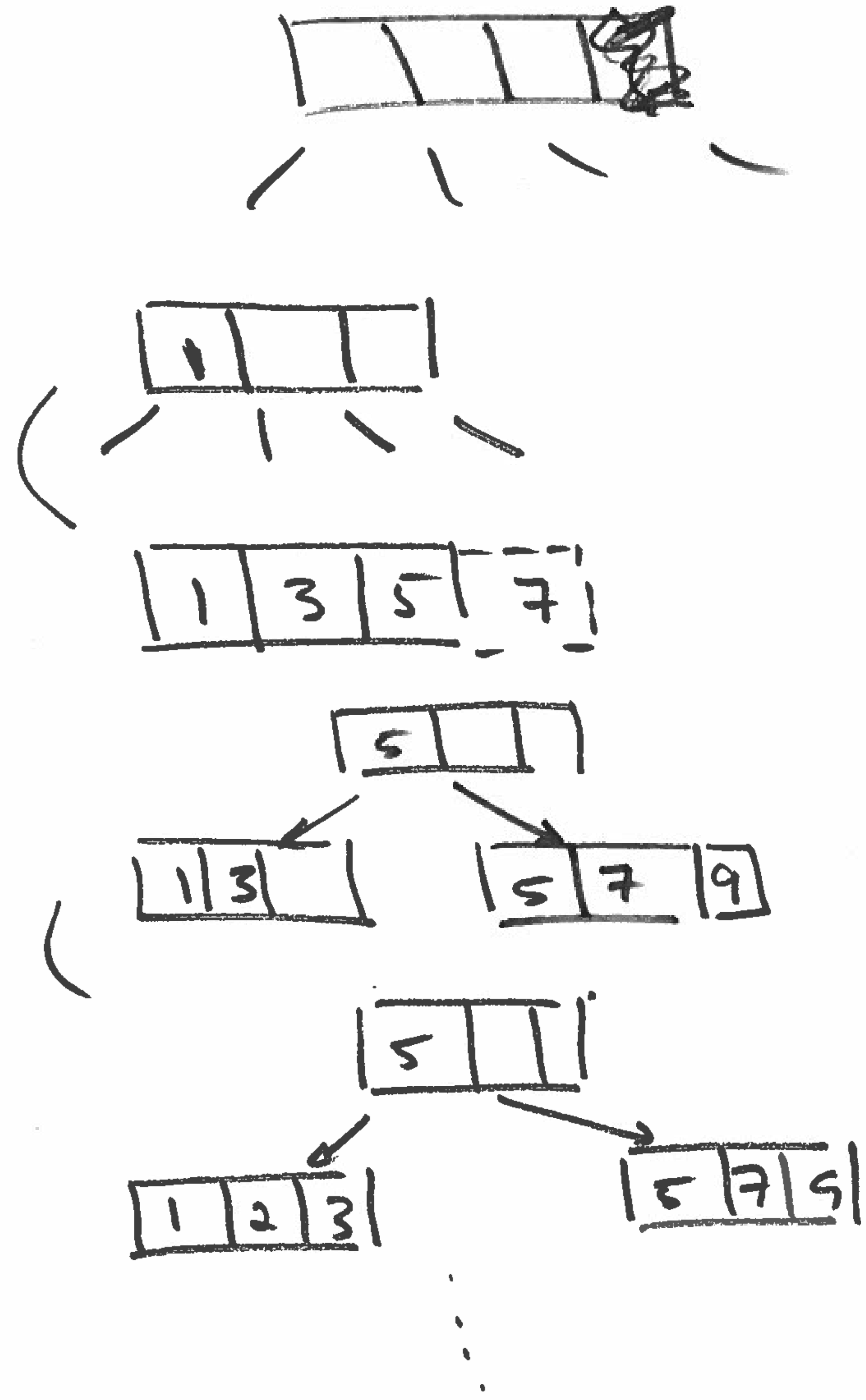
example

age



1, 3, 5, 7, 9, 2, 4, 6, 8, 10

→ tree node → 4 Pointers
3 keys


1
1 3 5 7
5
1 3
5 7 9
5
1 2 3
5 7 9