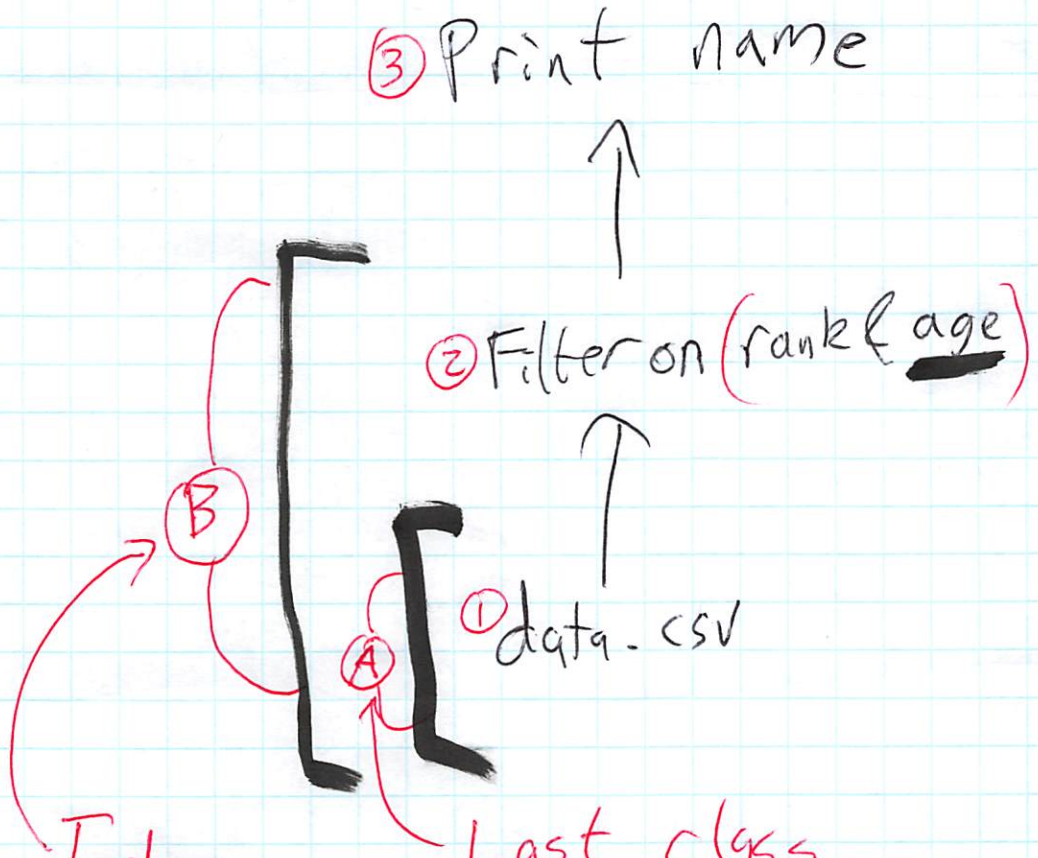
```
$> ls . /lib
          /build
          /data
          /data/[filename].csv
          :
$> echo "SELECT * FROM filename;" |
     java -cp build:jsqlparser dubstep.Main -

red | 23 | 59.265 | pickup
blue | 47 | 63.921 | dropoff
     :
```

Expected output for checkpoint 0

Workflow for Script

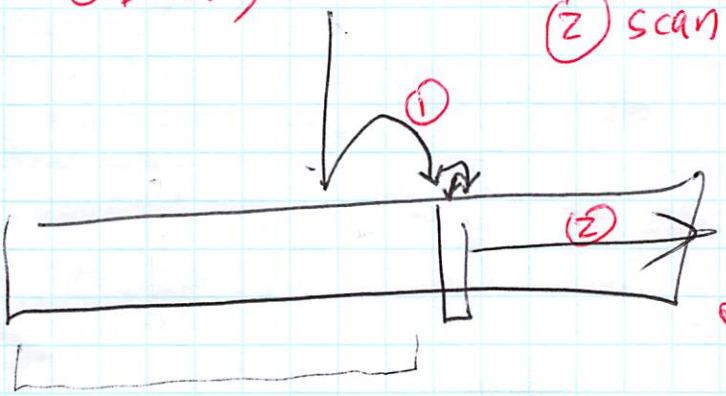


How do we make filter(age > 25) fast?

Idea: Sort data by age!

① binary search to find age = 25

② scan + print all later records



Question

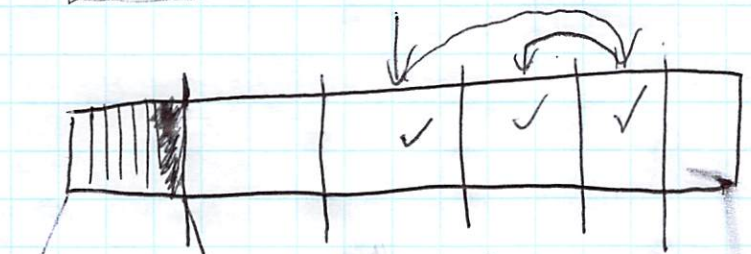
← Big blob of bits

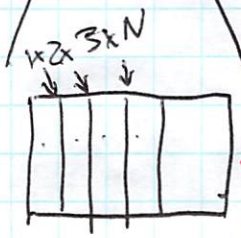
● where does the middle record start?

↳ Use fixed length records

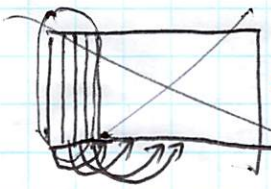
↳ Create fixed size "chunks" of records

↳ still need to do binary search within a chunk





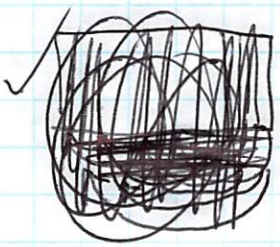
← Option1: fixed size records



← Option2: Header w directory of record positions

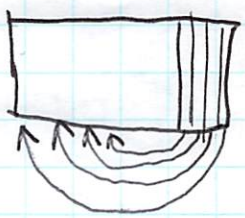
(not generally used)

Bonus Q: Why



~~Option3~~: Footer w/ directory of record positions

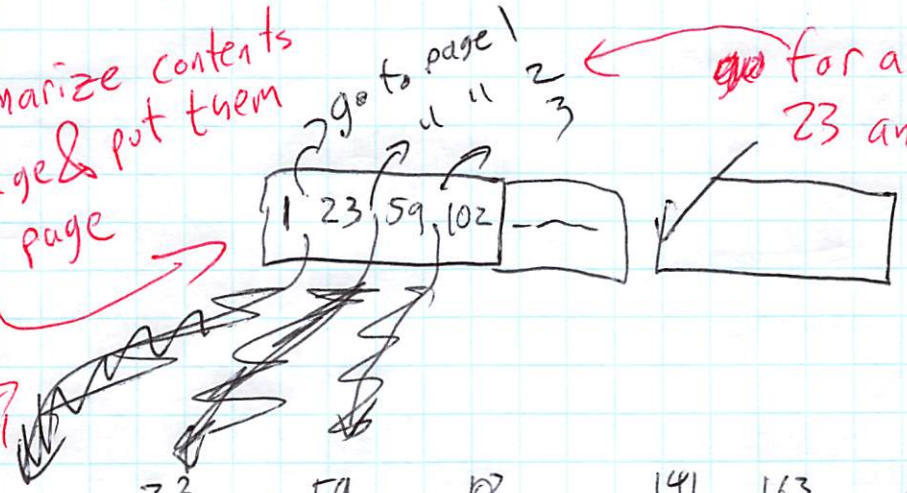
(actually used... why?)



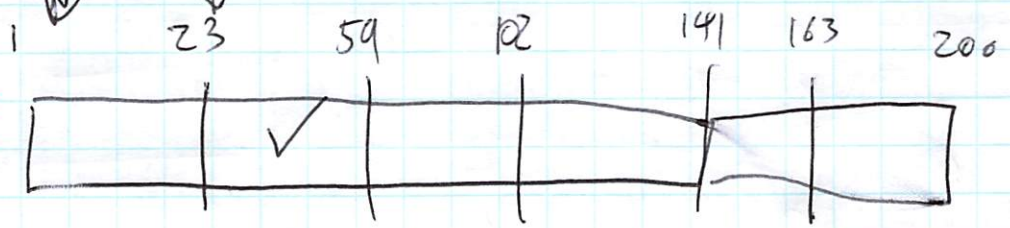
Free Bonus: Chunk = disk page (~4kb)
= cache line (64b)

Problem: Binary search → log N pages loaded

Idea: Summarize contents of each page & put them all on one page

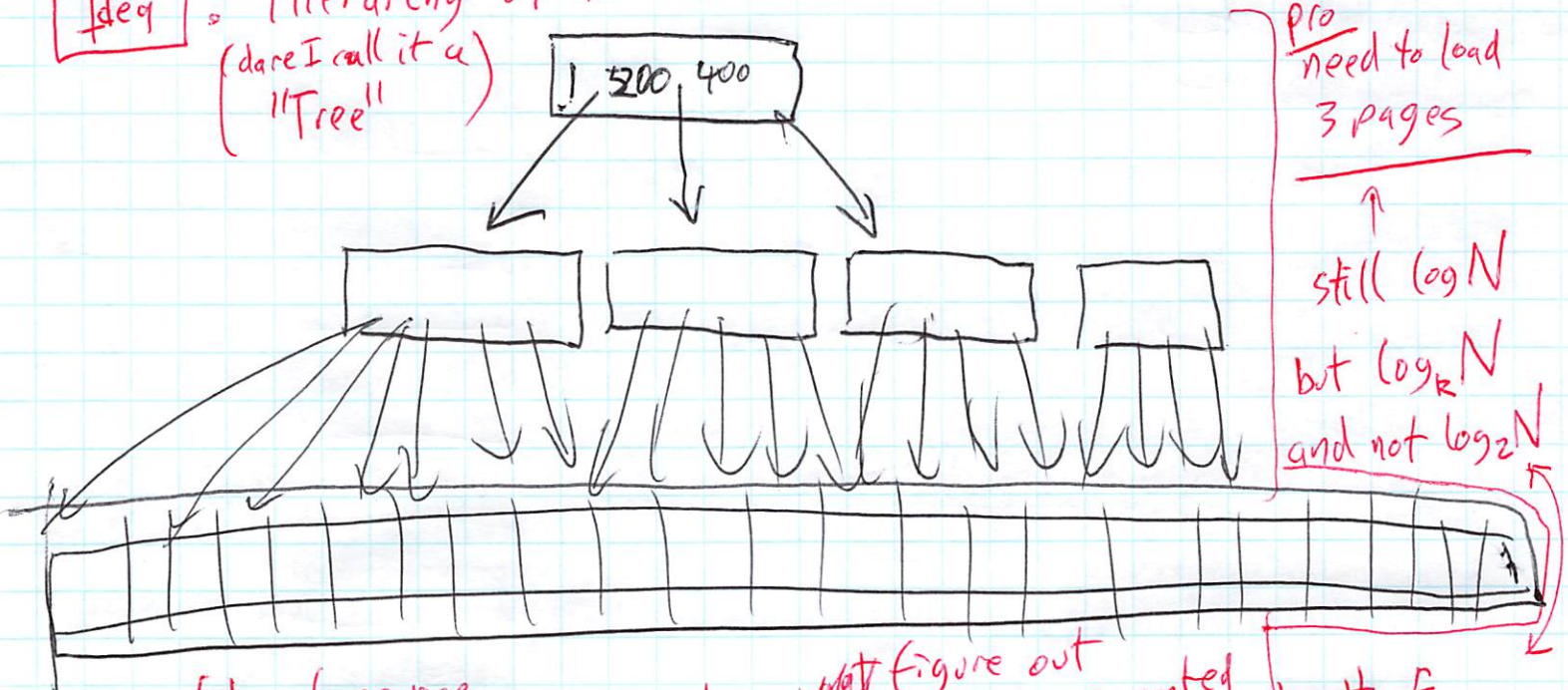


pro: only need to load 2 pages = #2



problem: what if you need more than one page of summaries?

Idea: Hierarchy of summaries → Summarize the summaries (dare I call it a "Tree")



pro need to load 3 pages

↑

still log N but $\log_k N$ and not $\log_2 N$

k = # of page refs stored on one summary page

In the language of DBs

Index: Summaries to help you figure out where your data lives

ISAM Index: The "Tree Index" we just invented

Index page: A "summary" page or "chunk"

Data page: A page or "chunk" of records

Generalizations to other filtering conditions

age = X

↳ binary search

age > X (age ≥ x)

↳ binary search for page 1

↳ scan remaining

age < X

↳ scan up to last record

X < age < Y