Column Stores

Supplemental Reading
(papers posted on Piazza)

Announcements

- Project 2 regrade incoming by Monday

Moore's Law

- Processor speeds double every ~2 years.
    - … but hard disks haven't kept pace.
    - Disks are bigger, but not much faster.
- Memory hasn't kept pace much better.
- **Consequence**: CPU is cheap, IO is expensive.
    - Can we exploit this in the DB?

Reducing IO

- Store the data sorted!
  - … can only sort on one column.
- Store the data compressed!
  - … makes it hard to access individual fields.
- Don’t scan all the data
  - … need to read entire rows.

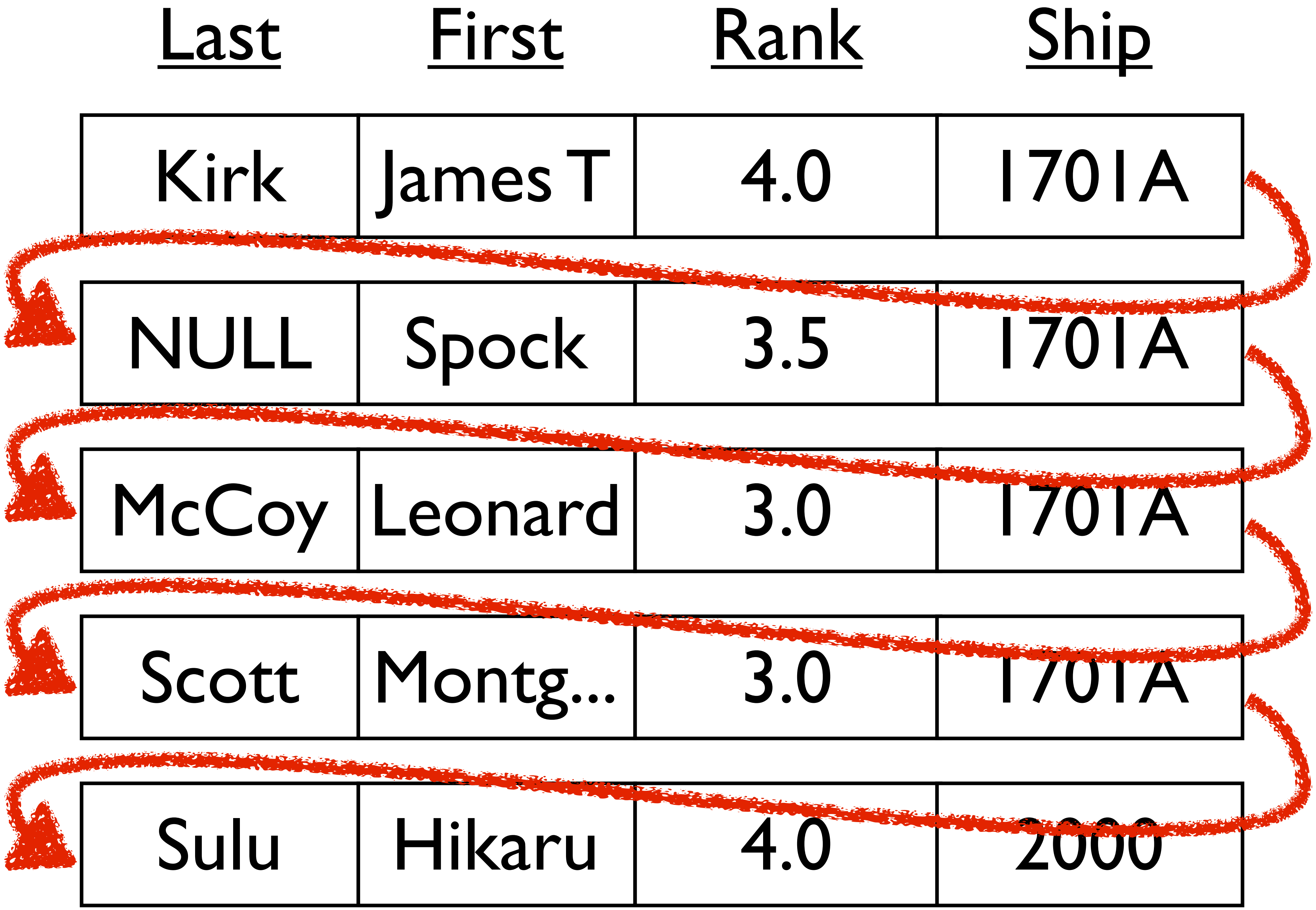
Row vs Column Stores

Last	First	Rank	Ship
Kirk	James T	4.0	1701A
NULL	Spock	3.5	1701A
McCoy	Leonard	3.0	1701A
Scott	Montg...	3.0	1701A
Sulu	Hikaru	4.0	2000

Row vs Column Stores

Last	First	Rank	Ship
Kirk	James T	4.0	1701A
NULL	Spock	3.5	1701A
McCoy	Leonard	3.0	1701A
Scott	Montg...	3.0	1701A
Sulu	Hikaru	4.0	2000

Row vs Column Stores



Last	First	Rank	Ship
Kirk	James T	4.0	1701A
NULL	Spock	3.5	1701A
McCoy	Leonard	3.0	1701A
Scott	Montg...	3.0	1701A
Sulu	Hikaru	4.0	2000

Row vs Column Stores

	Last		First		Rank		Ship
1:	Kirk	1:	James T	1:	4.0	1:	1701A
2:	NULL	2:	Spock	2:	3.5	2:	1701A
3:	McCoy	3:	Leonard	3:	3.0	3:	1701A
4:	Scott	4:	Montg...	4:	3.0	4:	1701A
5:	Sulu	5:	Hikaru	5:	4.0	5:	2000

Store Each Column Separately (with a ‘rowid’)

Row vs Column Stores

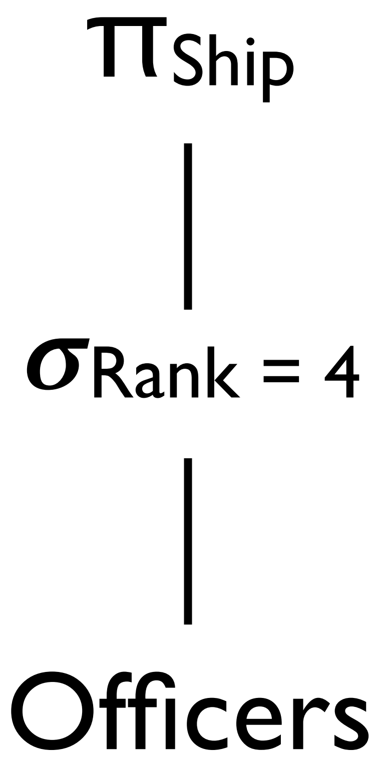
	Last		First		Rank		Ship
1:	Kirk	5:	Hikaru	1:	4.0	1:	1701A
3:	McCoy	1:	James T	5:	4.0	2:	1701A
4:	Scott	3:	Leonard	2:	3.5	3:	1701A
5:	Sulu	4:	Montg...	3:	3.0	4:	1701A
2:	NULL	2:	Spock	4:	3.0	5:	2000

Keep Columns Sorted on Value and/or RowID

Querying a Column Store

Which Ships Have a Captain (Rank 4.0)?

```
SELECT O.Ship
FROM Officers O
WHERE O.Rank = 4
```



π_{Ship}

$\sigma_{\text{Rank = 4}}$

Officers

Querying a Column Store

Selection Can Be Applied Looking Only At Rank Data

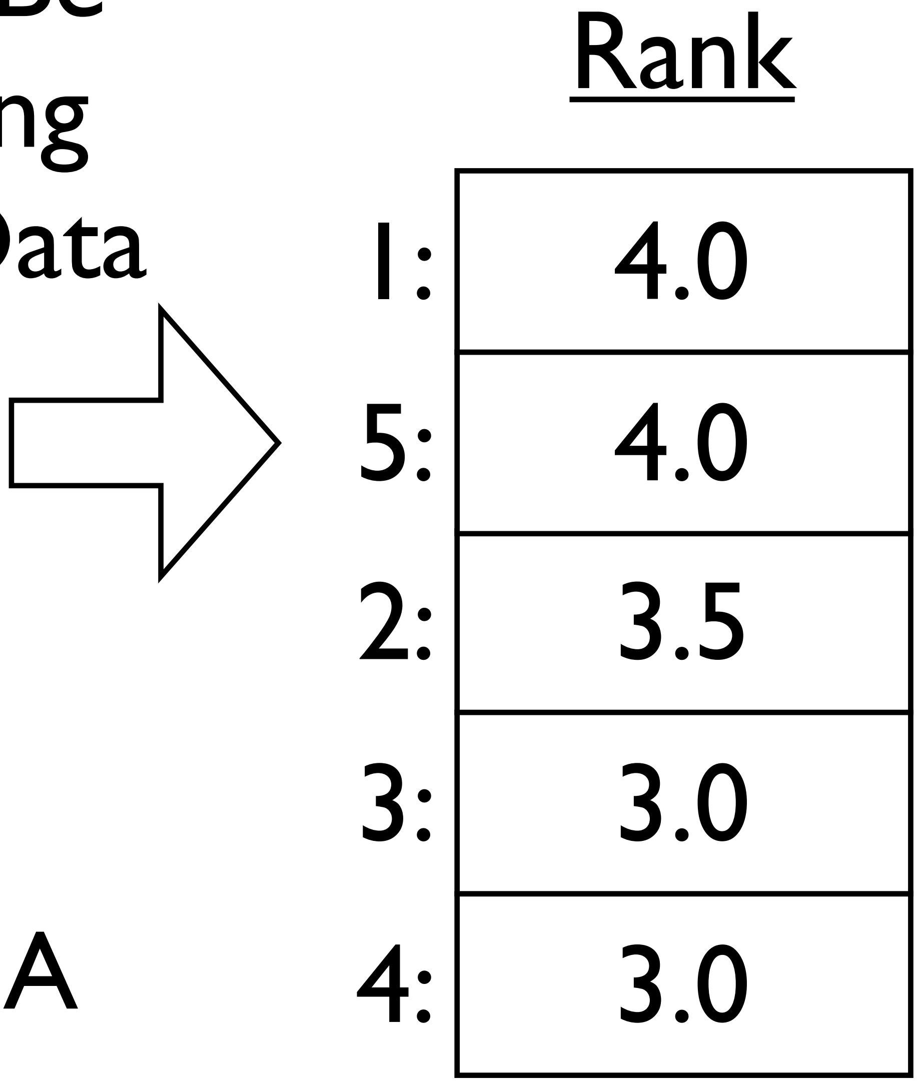
$\sigma_{\text{Rank} = 4}$

	Rank
1:	4.0
5:	4.0
2:	3.5
3:	3.0
4:	3.0

Querying a Column Store

Selection Can Be Applied Looking Only At Rank Data

$\sigma_{\text{Rank = 4}}$



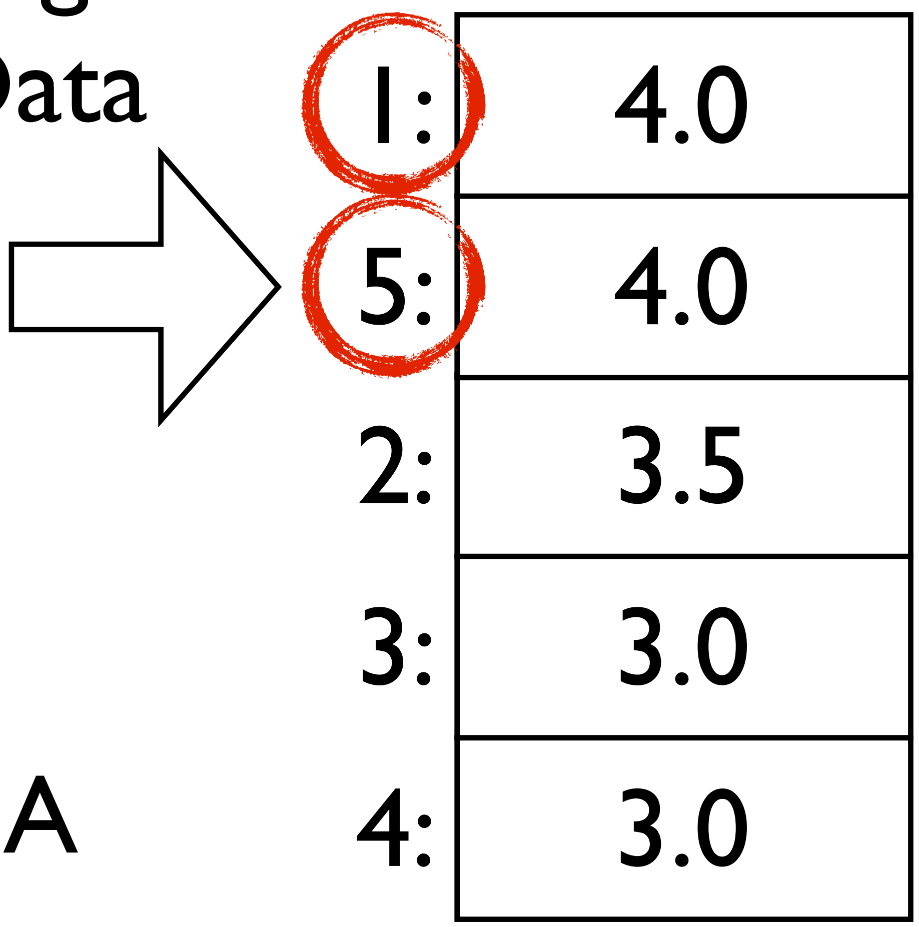
	Rank
1:	4.0
5:	4.0
2:	3.5
3:	3.0
4:	3.0

Selection Is Just A Binary Search Over Sorted Data

Querying a Column Store

Selection Can Be Applied Looking Only At Rank Data

$\sigma_{Rank = 4}$



	Rank
1:	4.0
5:	4.0
2:	3.5
3:	3.0
4:	3.0

Tuples in Answer Represented by RowIDs

Selection Is Just A Binary Search Over Sorted Data

Querying a Column Store

π_{Ship}

Answer Set: 1, 5

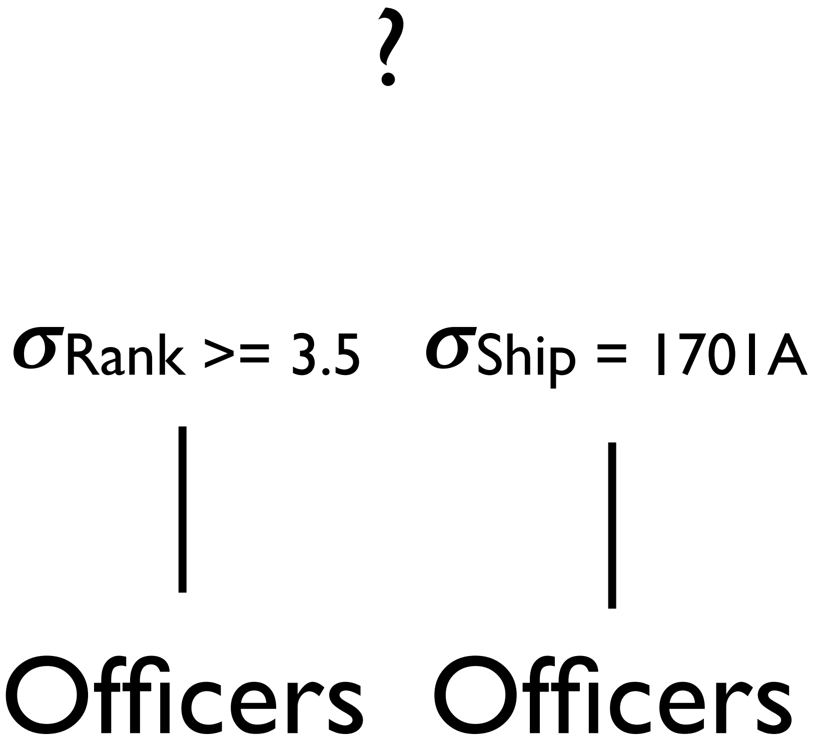
RowIDs Are Only Dereferenced When Producing Output For The User

	Ship
1:	1701A
2:	1701A
3:	1701A
4:	1701A
5:	2000

Querying a Column Store

Which officers on the Enterprise (1701A) are ranked commander (3.5) or higher?

```
SELECT O.First
FROM Officers O
WHERE O.Ship = ‘1701A’
  AND O.Rank >= 3.5
```



?

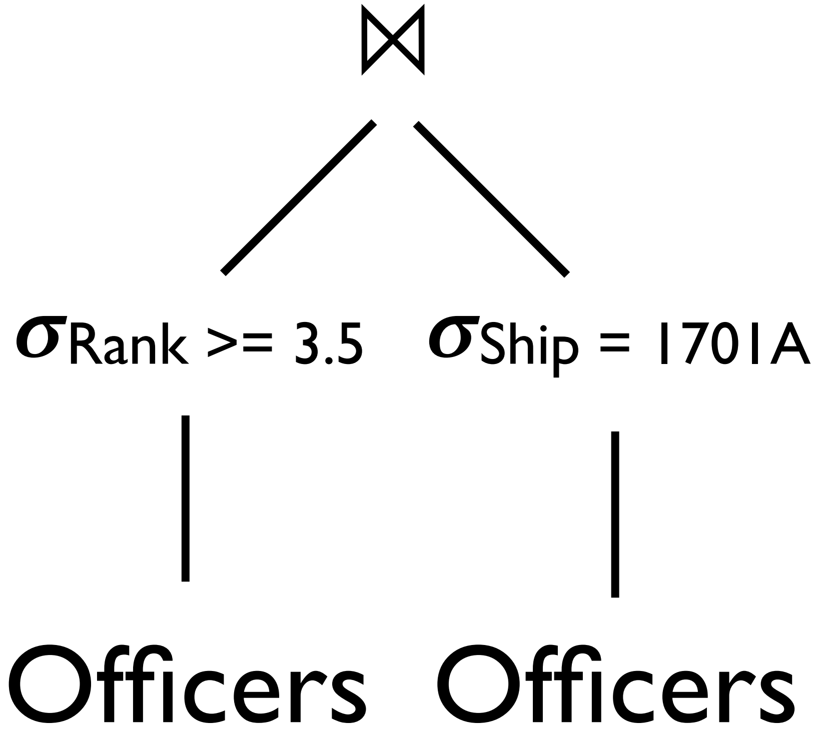
$\sigma_{\text{Rank} >= 3.5}$ $\sigma_{\text{Ship} = 1701\text{A}}$

Officers Officers

Querying a Column Store

Which officers on the Enterprise (1701A) are ranked commander (3.5) or higher?

```
SELECT O.First
FROM Officers O
WHERE O.Ship = '1701A'
  AND O.Rank >= 3.5
```



$\bowtie$

$\sigma_{\text{Rank >= 3.5}}$ $\sigma_{\text{Ship = 1701A}}$

Officers Officers

Querying a Column Store

{1,5,2} ∩ {1,2,3,4}

	Rank
1:	4.0
5:	4.0
2:	3.5
3:	3.0
4:	3.0

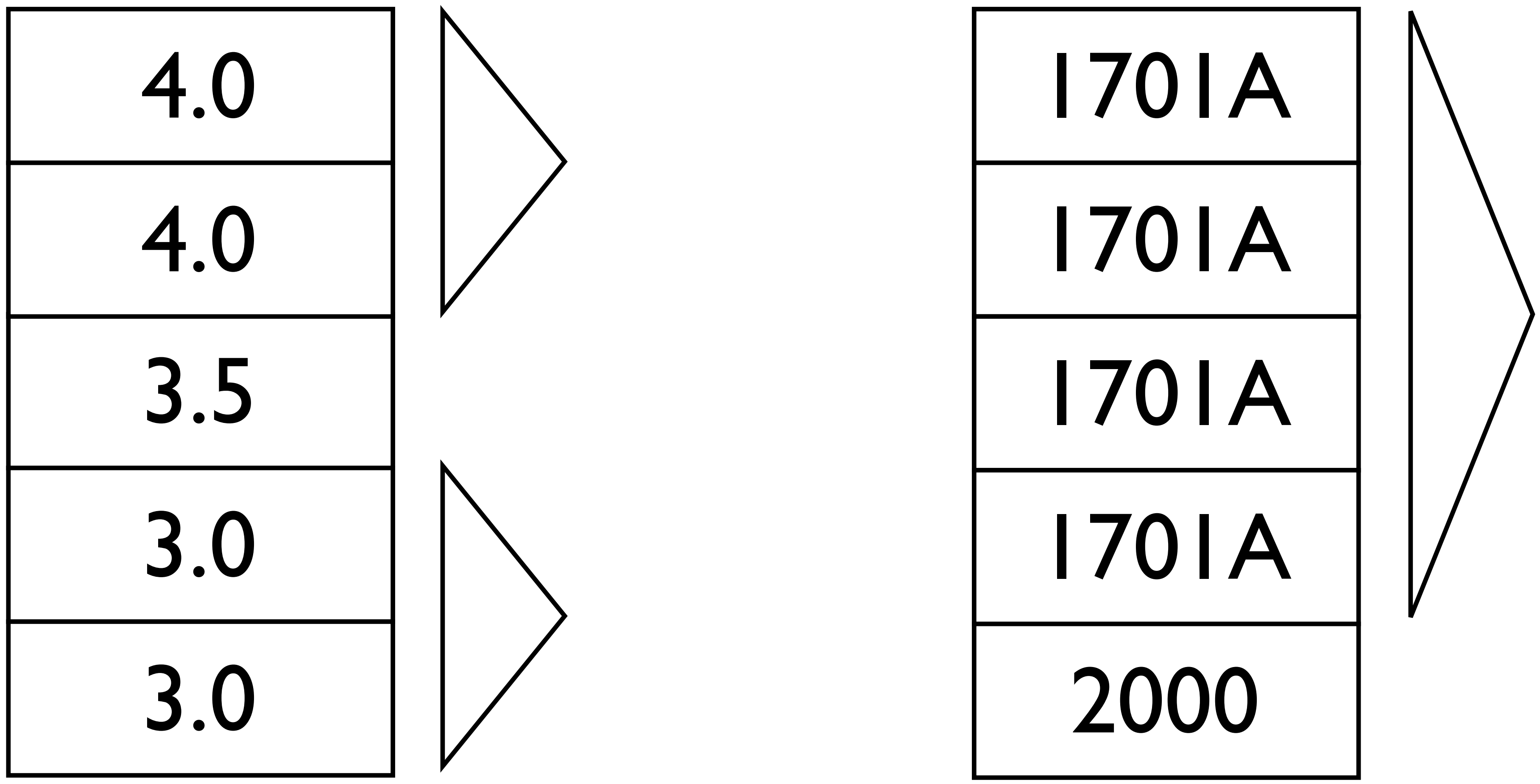
	Ship
1:	1701A
2:	1701A
3:	1701A
4:	1701A
5:	2000

Compression

	Last		First		Rank		Ship
1:	Kirk	5:	Hikaru	1:	4.0	1:	1701A
3:	McCoy	1:	James T	5:	4.0	2:	1701A
4:	Scott	3:	Leonard	2:	3.5	3:	1701A
5:	Sulu	4:	Montg...	3:	3.0	4:	1701A
2:	NULL	2:	Spock	4:	3.0	5:	2000

Compression

“Run Length” Encoding



Rank	Ship
4.0	1701A
4.0	1701A
3.5	1701A
3.0	1701A
3.0	2000

Compression

“Run Length” Encoding

Rank

4.0	x2
3.5	x1
3.0	x2

Ship

1701A	x4
2000	x1

Compression

Not Quite “Run Length” Encoding

Rank

4.0	:{1,5}
3.5	:{2}
3.0	:{3,4}

Ship

1701A	:{1,2,3,4}
2000	:{5}

Column Stores

- Split relations into one sub-relation per column, each with schema <RowID,Value>
- Columnar representation is more IO efficient.
    - …doesn't need to read entire rows.
    - …can organize each column differently.
    - …can easily compress data.

Column Stores

- Split relations into one sub-relation per column, each with schema <RowID, Value>
- Columnar representation is less CPU efficient.
  - …reconstructing rows requires joins.
  - …compressed data must be decompressed.
- Extremely useful if entire rows needed infrequently.
  - …big data analytics, computational advertising

Database “Cracking”

- **ETL**: Extract Transform <u>Load</u>
- Load step is incredibly expensive
  - Data must be sorted, indexed.
  - High <u>upfront</u> cost.
- Can we amortize this cost over queries?

Database “Cracking”

Data Loaded
Unsorted,
Unindexed

43
15
16
19
65
10
84
23

```
SELECT ...
FROM ...
WHERE v > 65
```

Database “Cracking”

Perform One Step Of Quick-Sort Using Predicate as Pivot

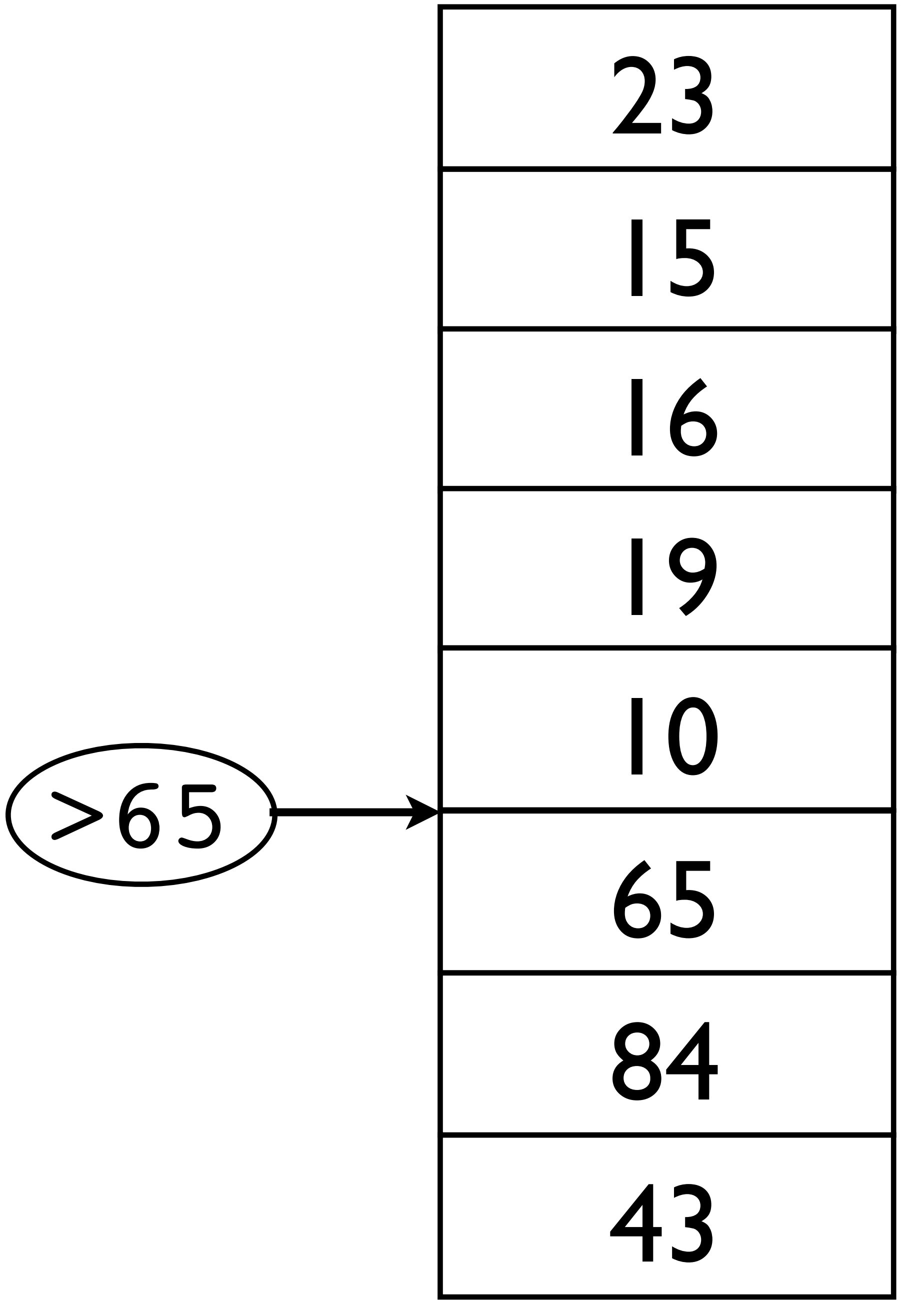
```
SELECT ...
FROM ...
WHERE v > 65
```

23
15
16
19
10
65
84
43

Database “Cracking”

Perform <u>One</u> Step Of Quick-Sort Using Predicate as Pivot

```
SELECT …
FROM …
WHERE v > 65
```



23

15

16

19

10

>65

65

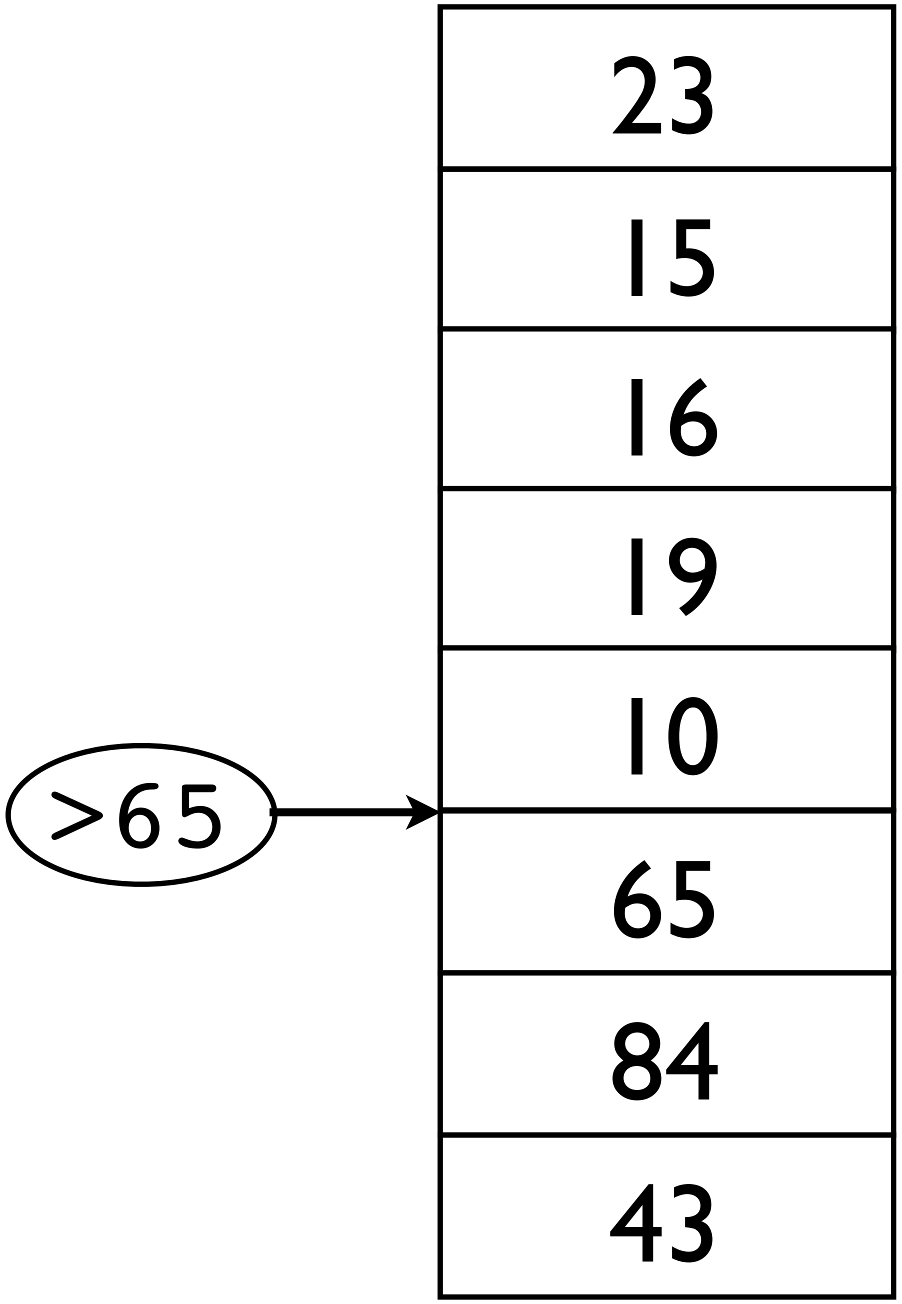
84

43

Database "Cracking"

Perform One Step Of Quick-Sort Using Predicate as Pivot

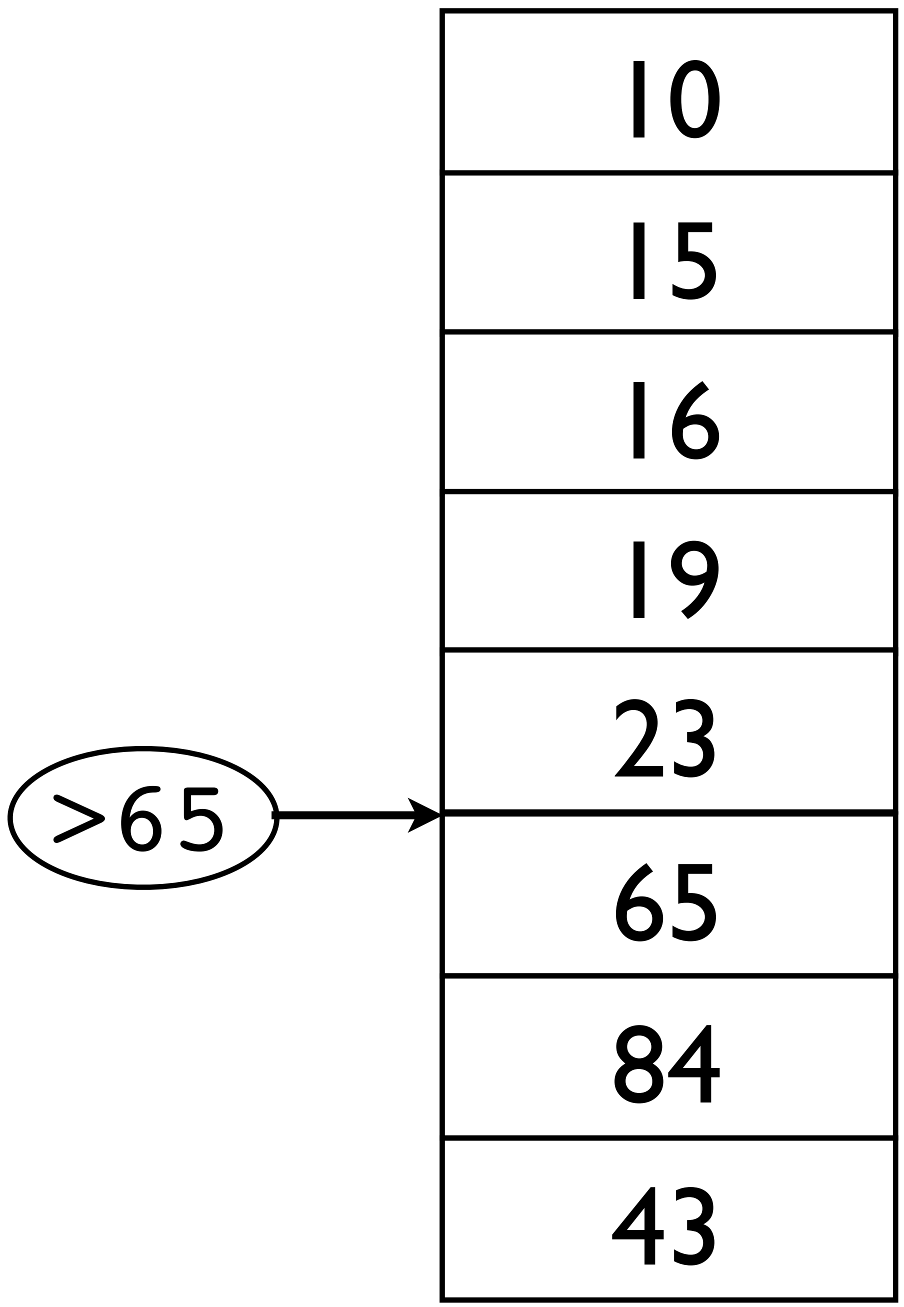
```
SELECT …
FROM …
WHERE v < 17
```



Database "Cracking"

Next Query Only Needs
To Scan ~1/2 of the Table

```
SELECT …
FROM …
WHERE v < 17
```

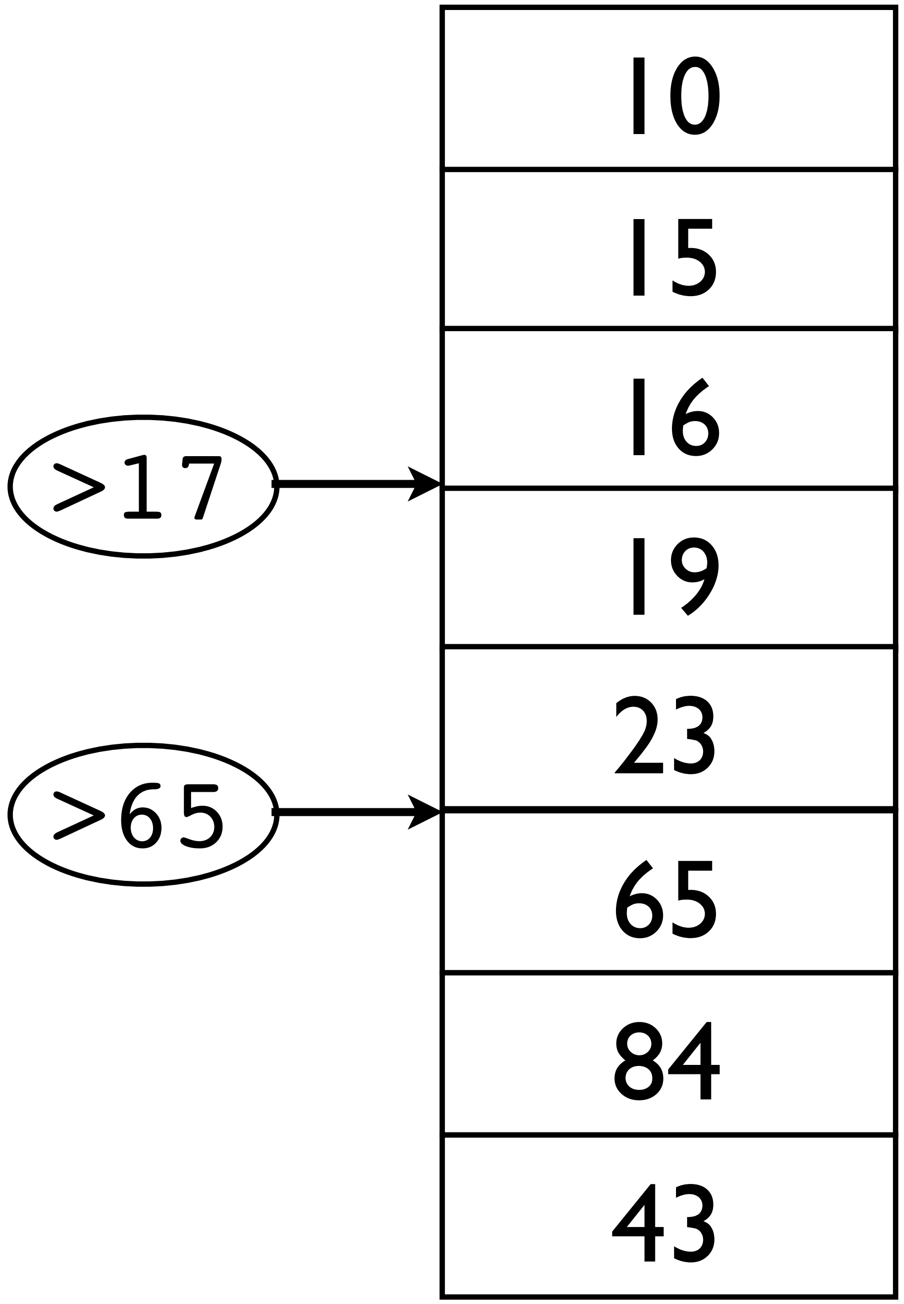


10
15
16
19
23
>65
65
84
43

Database “Cracking”

Next Query Only Needs
To Scan ~1/2 of the Table

```
SELECT …
FROM …
WHERE v < 17
```

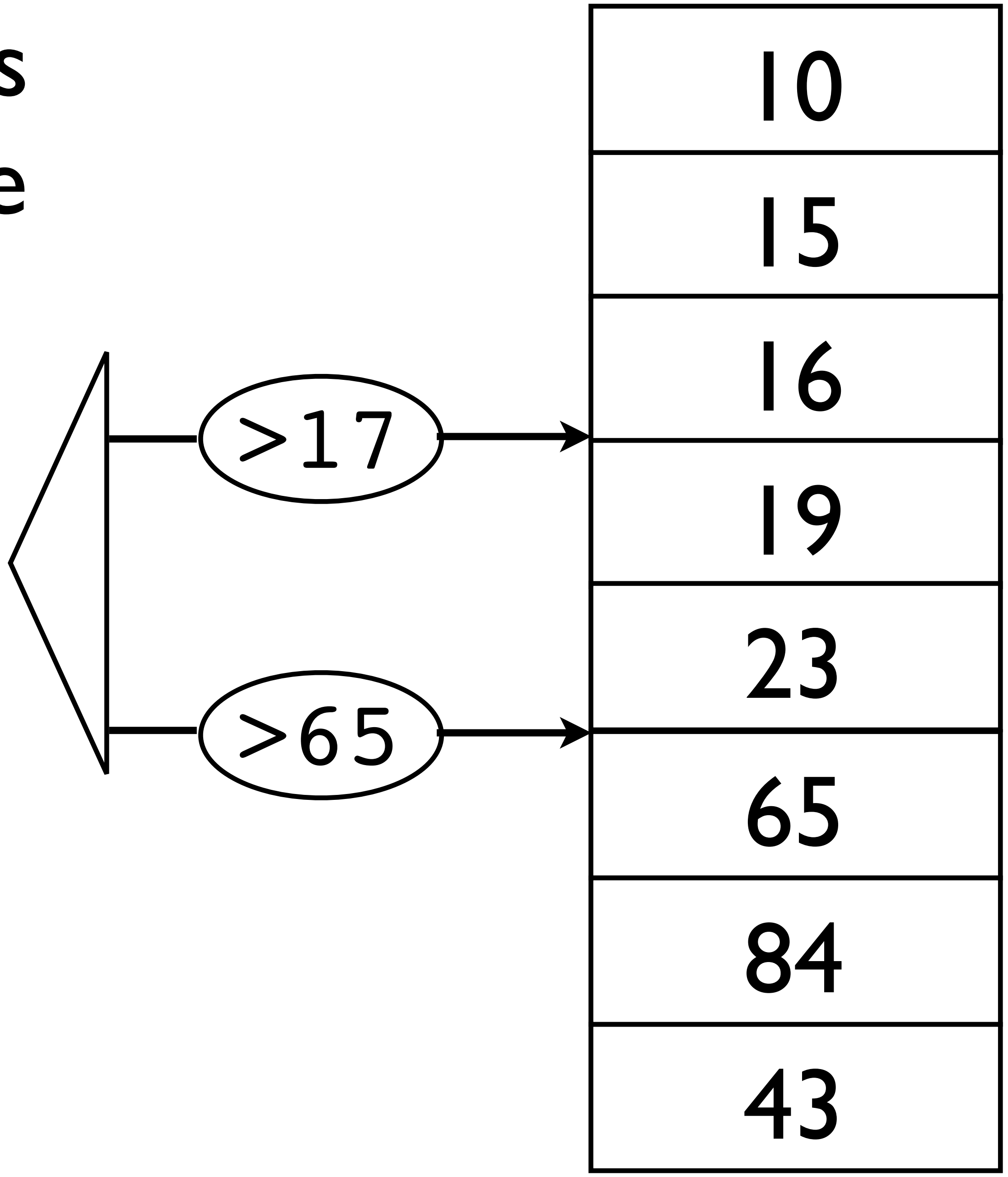


Database “Cracking”

Next Query Only Needs
To Scan ~1/2 of the Table

Start Building An
Index Over Pointers

```
SELECT ...
FROM ...
WHERE v < 17
```



10
15
16
19
23
65
84
43

Database “Cracking”

- Start with unsorted data
  - First query scans entire table and partitions
  - Second query scans only relevant partition(s).
- Optimization: First query copies before scan.
  - Keep original copy sorted in RowID order.
- Table gets progressively more sorted.
- How do we handle insertions?