Optimization: Better Cost Estimation

R&G Chapter 15

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

Recap

- 2 dimensions to search along for plans (or more)
  - What is the best access path? (π/σ equivs)
  - What is the best join order? ($\bowtie$ equivs)
- Consider the **cost** of each allowable plan.
- Understanding how each operator's output size relates to its input size makes it possible to accurately estimate the cost of a plan.
- Simplify the join order problem by exploring the cost of left-deep plans only.

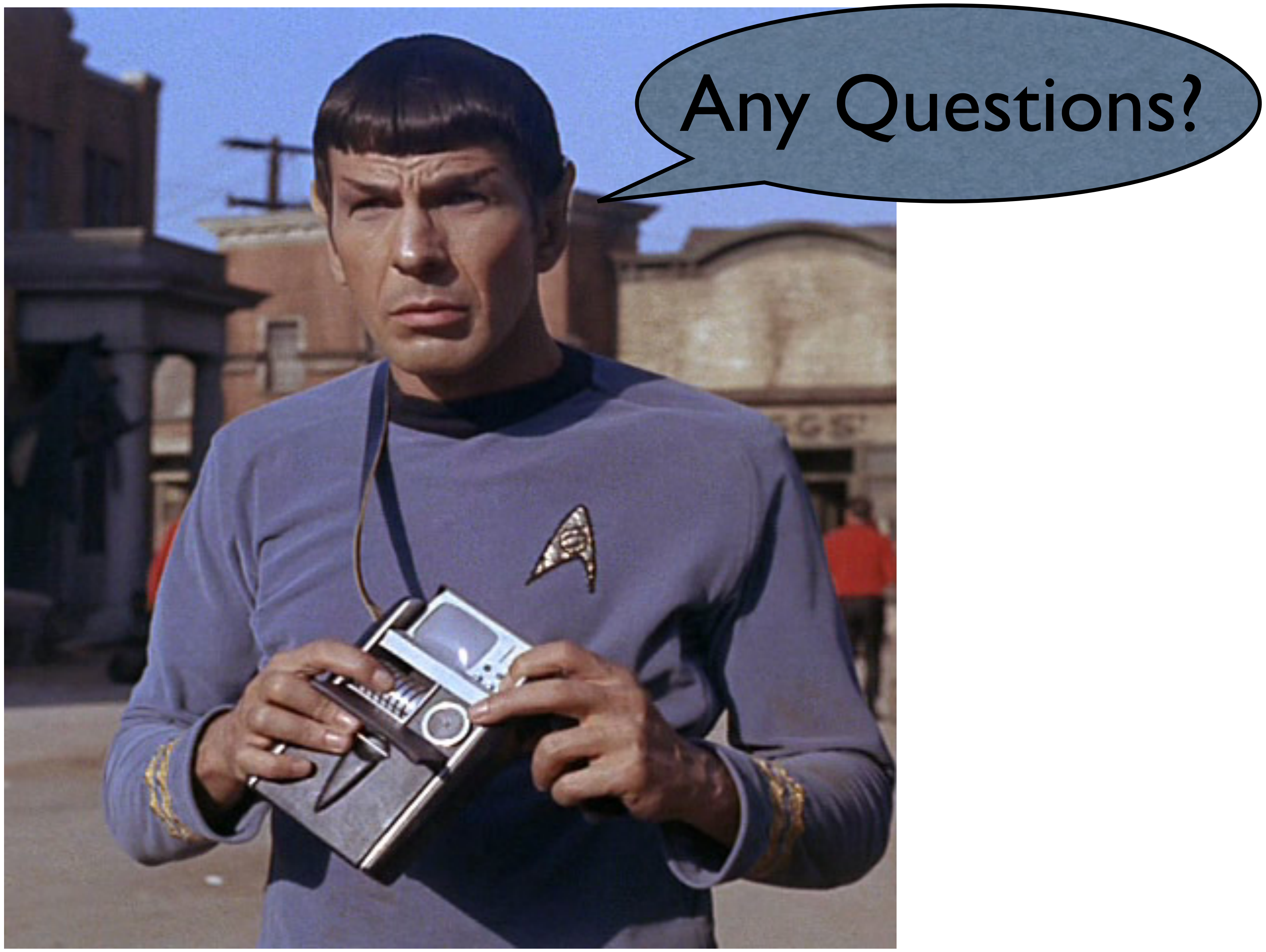
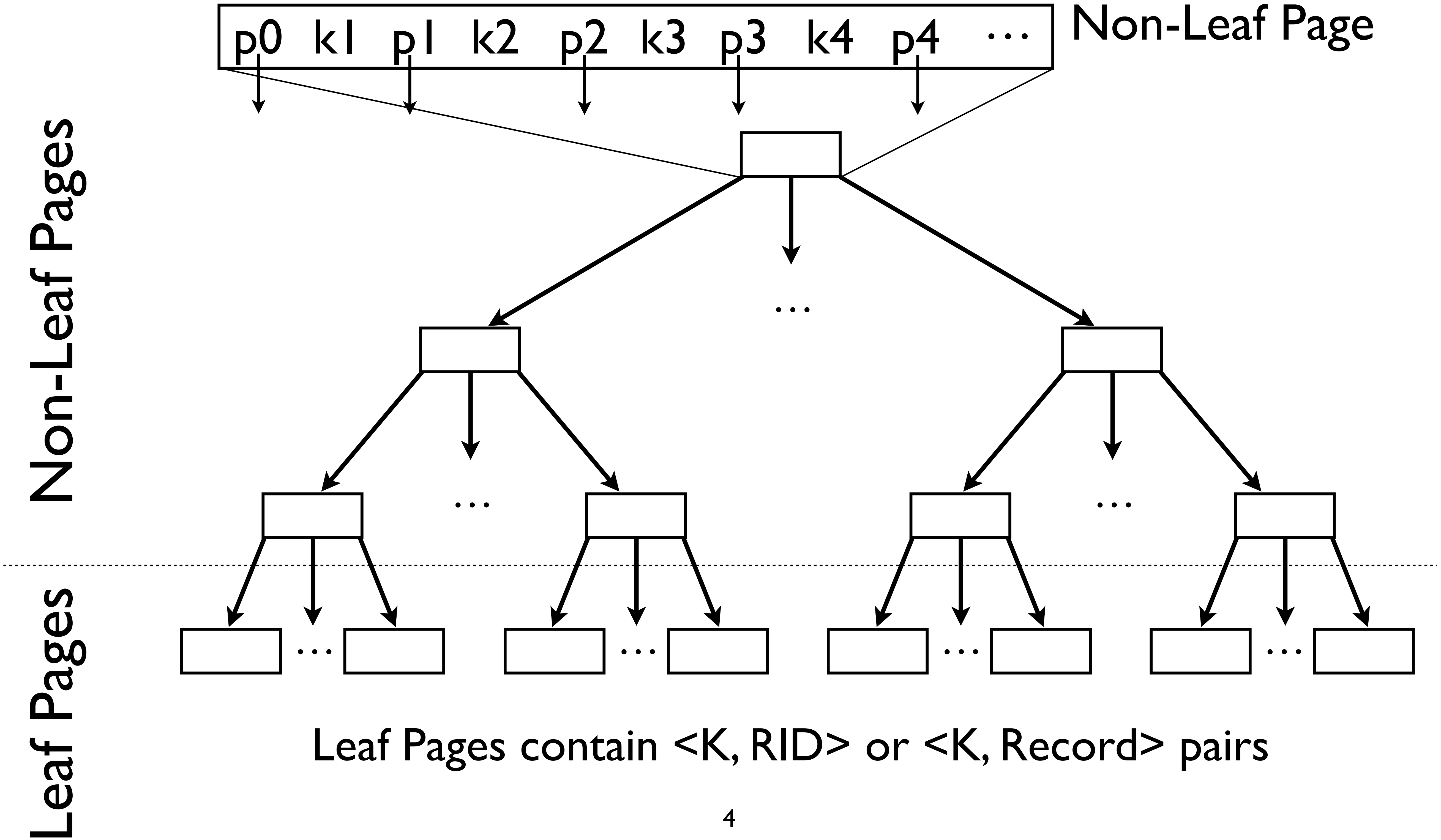


Image copyright: Paramount Pictures

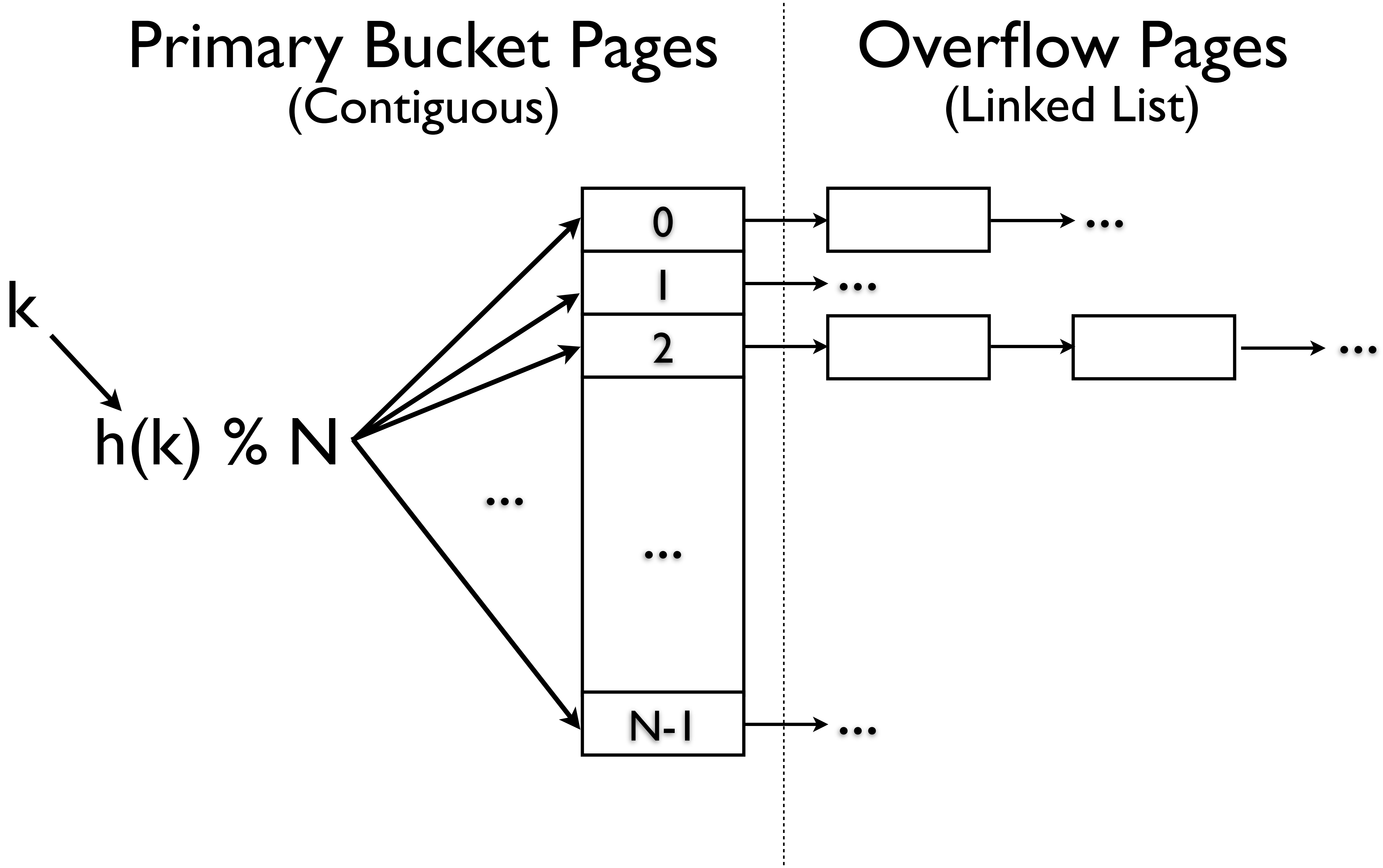
Recap: ISAM



Leaf Pages contain <K, RID> or <K, Record> pairs

ISAM: Index-structured access method

Recap: Static Hashing



Project 2: sql.Index

Data Stream Generator: `sql.test.TestDataStream`

(`implements Iterator<Datum[]>`)

Index Class Tests Your Indexes

```
index -hash hash.index
```

Build a hash index in ‘`hash.index`’

```
index -isam isam.index
```

Build an ISAM index in ‘`isam.index`’

Project 2: sql.Index

```
index -hash hash.index
```

Build a hash index in ‘`hash.index`’

```
index -hash hash.index -get 23
```

Use ‘`hash.index`’ to find `<23,...>`

Project 2: sql.Index

```
index -isam isam.index
```

Build an ISAM index in ‘`isam.index`’

```
index -isam isam.index -scan
```

Validate ISAM index scan

```
index -isam isam.index -scan
      -from 5 -to 23
```

Validate ISAM index scan over keys 5 to 23

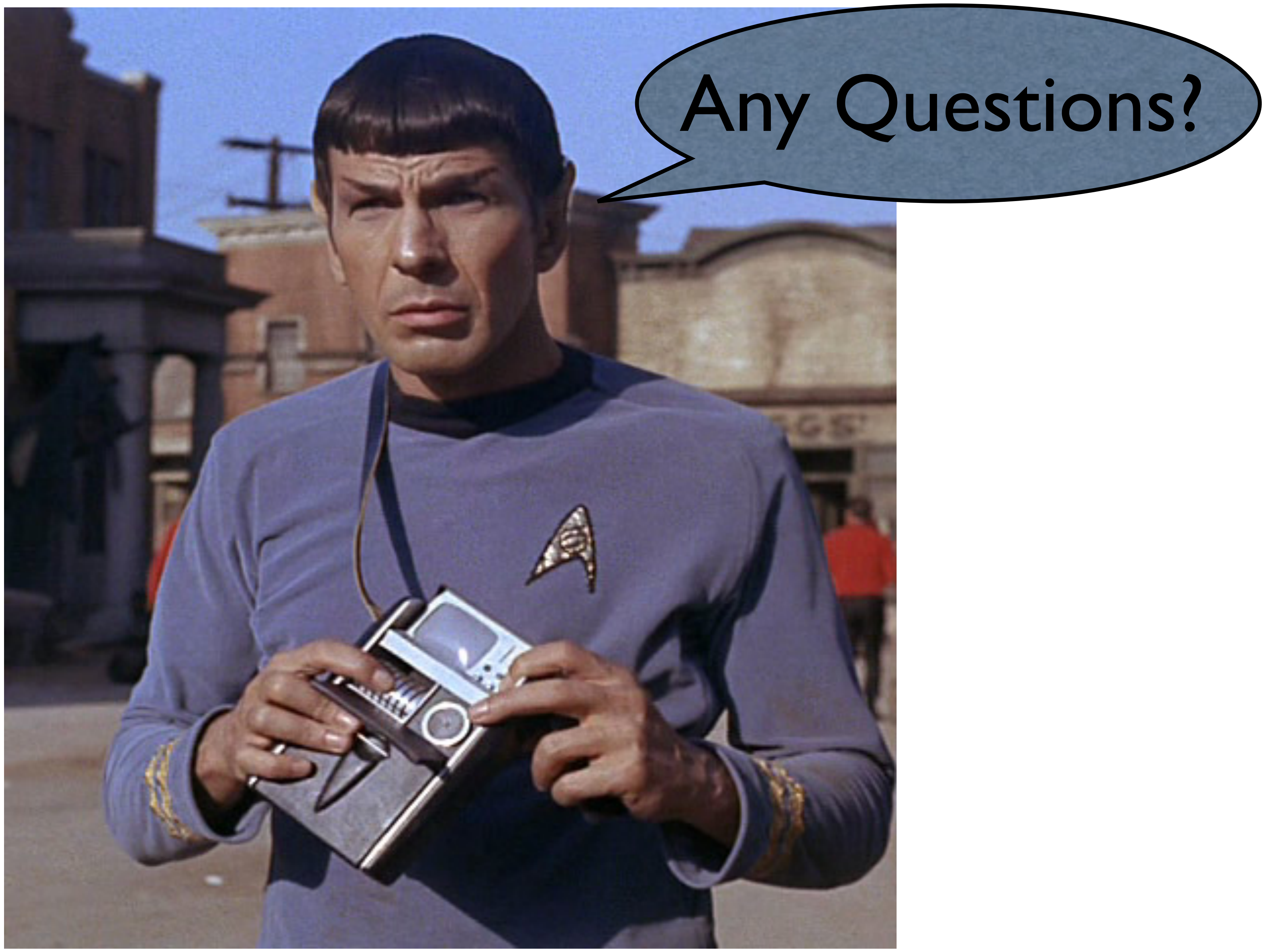
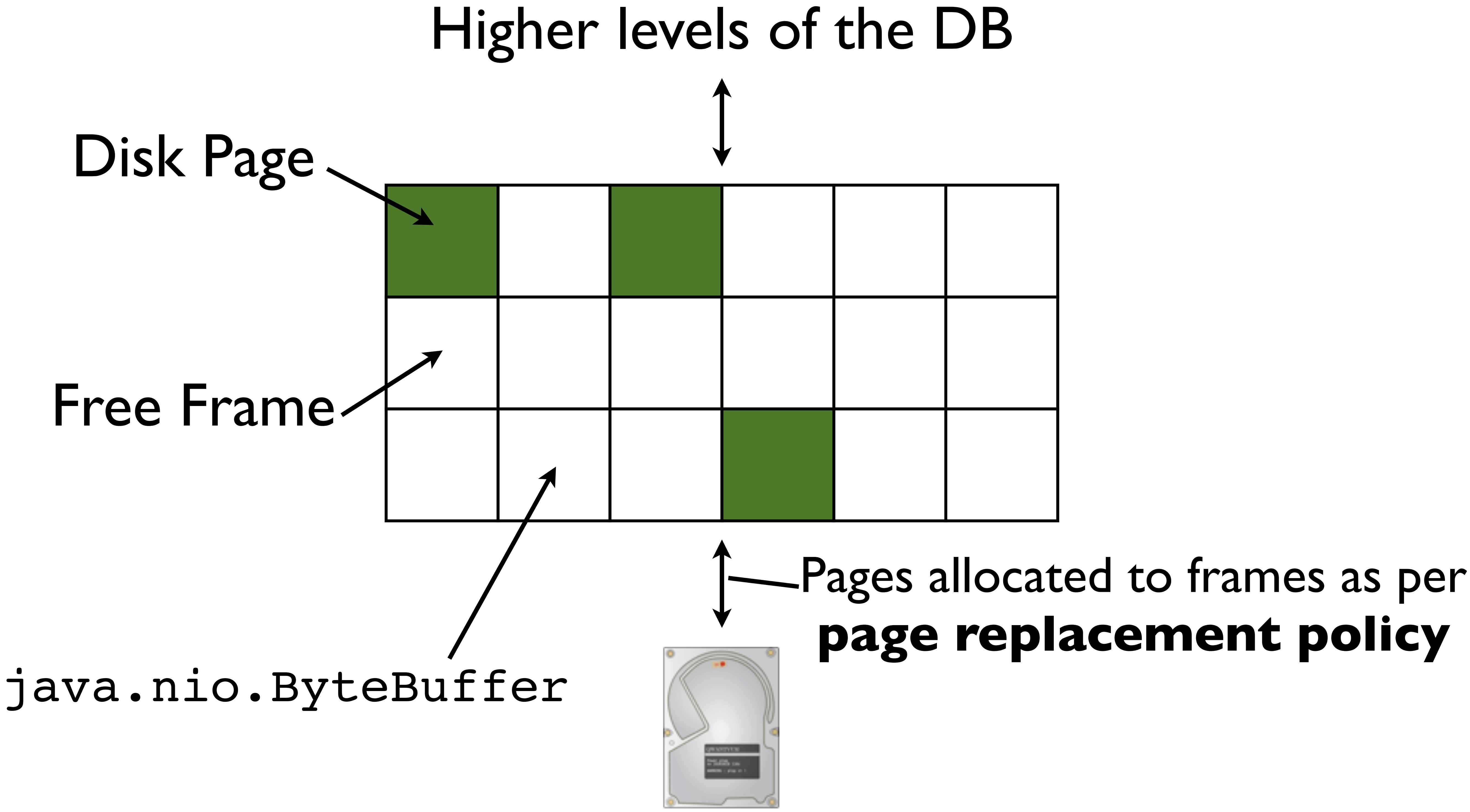


Image copyright: Paramount Pictures

sql.buffer.BufferManager



Higher levels of the DB

Disk Page

Free Frame

`java.nio.ByteBuffer`

Pages allocated to frames as per **page replacement policy**

`image credit: openclipart.org`

Pinned Pages

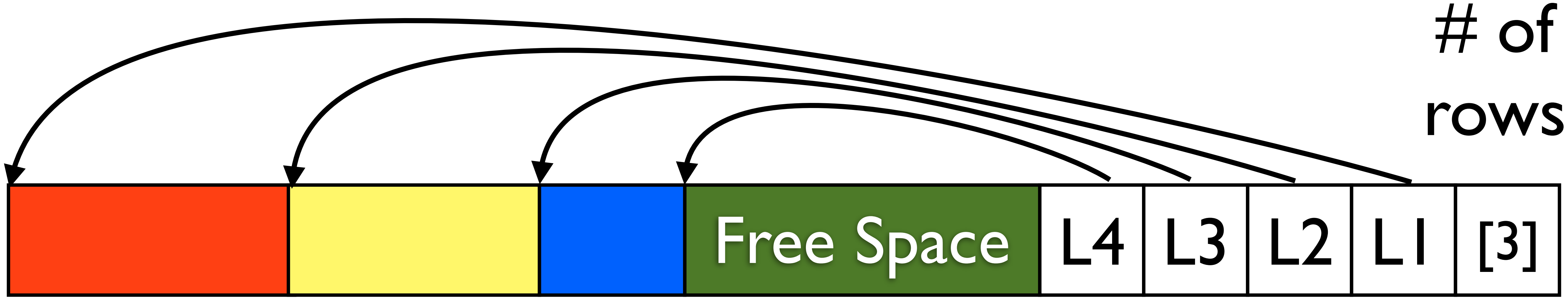
- Pinning a page indicates that it is being used.
- The requestor must unpin the page when done.
  - The requestor must also indicate whether the page has been modified (with a ‘dirty’ bit)
  - Dirty pages must be written to disk
- Pages may be requested multiple times
  - Use a pin count (reference count) to keep track.

sql.buffer.ManagedFile

- Interface to the Buffer Manager
- Request ManagedFiles through FileManager
- Maps file pages to buffer frames
  - Pages Data In and Out of Memory
- getBuffer(), pin(), unpin(), dirty(), flush()

sql.data.DatumBuffer

- Self-Describing Records
- Wraps around Java’s nio.ByteBuffer



Array of Field Offsets

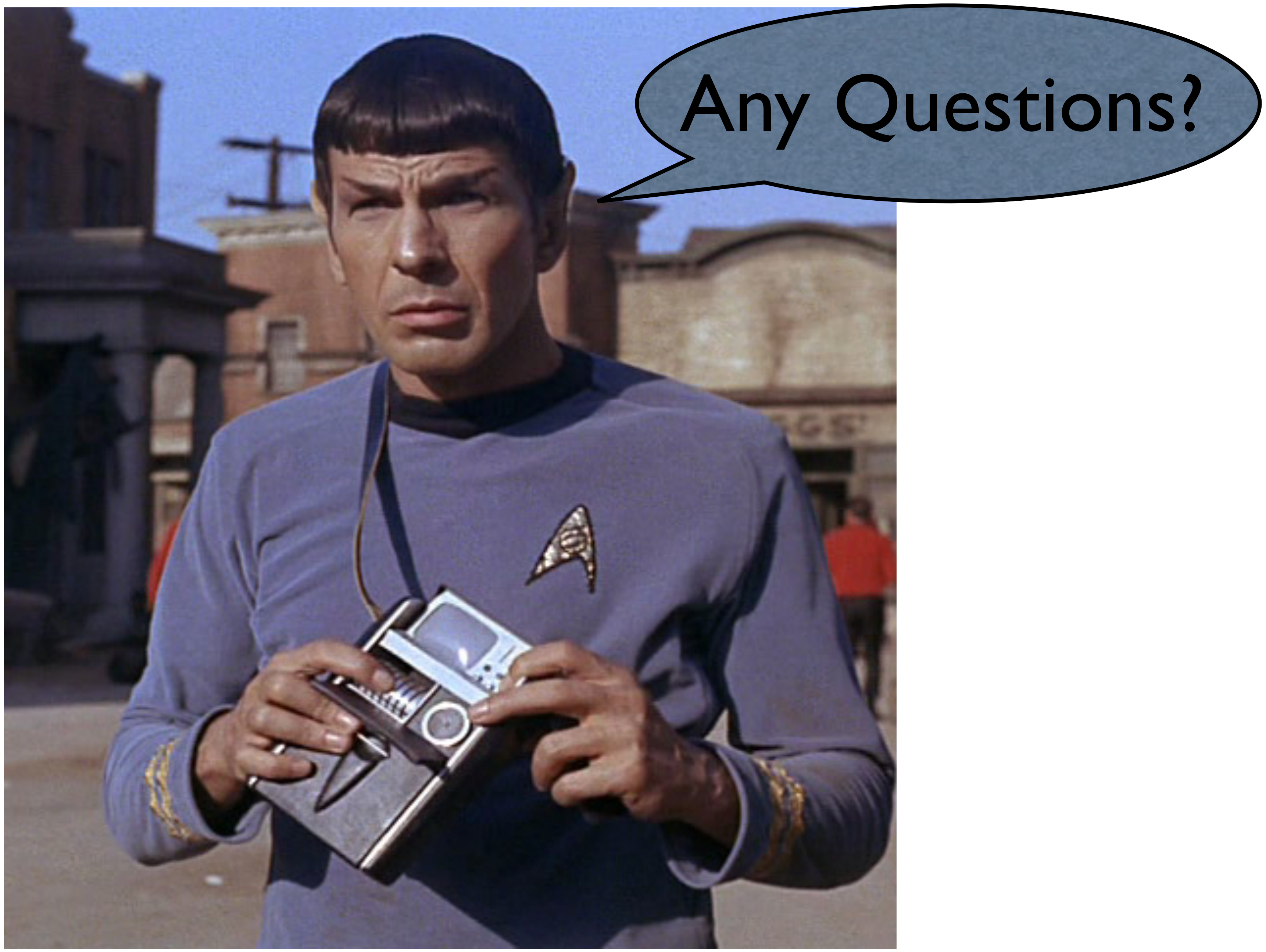


Image copyright: Paramount Pictures

Onwards!

Better Cost Estimation

- What information about data can we use to get better estimates of cost?
    - Data-based statistics
    - Schema-based properties

Histograms

Uniform Distributions are a strong assumption!

(data is often skewed)

Histograms

People

```
 Name          Age Rank
<"Alice",  21, 1  >
<"Bob",    20, 2  >
<"Carol",  21, 1  >
<"Dave",   19, 3  >
<"Eve",    20, 2  >
<"Fred",   20, 3  >
<"Gwen",   22, 1  >
<"Harry",  20, 3  >
```

```
SELECT Name
FROM People
WHERE Rank = 3
  AND Age = 20
```

vs

```
...
  AND Age = 19
```

$RF_{Age} = {}^{1}/_{nkeys} = {}^{1}/_{4}$

$RF_{Rank} = {}^{1}/_{nkeys} = {}^{3}/_{8}$

Age is best!

Histograms

People

```
 Name          Age Rank
<“Alice”,   21, 1  >
<“Bob”,     20, 2  >
<“Carol”,   21, 1  >
<“Dave”,    19, 3  >
<“Eve”,     20, 2  >
<“Fred”,    20, 3  >
<“Gwen”,    22, 1  >
<“Harry”,   20, 3  >
```

```
SELECT Name
FROM People
WHERE Rank = 3
  AND Age = 20
```

vs

```
...
  AND Age = 19
```

$RF_{Age-20} = {}^{1}/_{nkeys} = {}^{1}/_{2}$

$RF_{Rank} = {}^{1}/_{nkeys} = {}^{3}/_{8}$

Age is worst!

Histograms

People

```
 Name         Age Rank
<“Alice”,   21, 1  >
<“Bob”,     20, 2  >
<“Carol”,   21, 1  >
<“Dave”,    19, 3  >
<“Eve”,     20, 2  >
<“Fred”,    20, 3  >
<“Gwen”,    22, 1  >
<“Harry”,   20, 3  >
```

```
SELECT Name
FROM People
WHERE Rank = 3
  AND Age = 20
```

VS

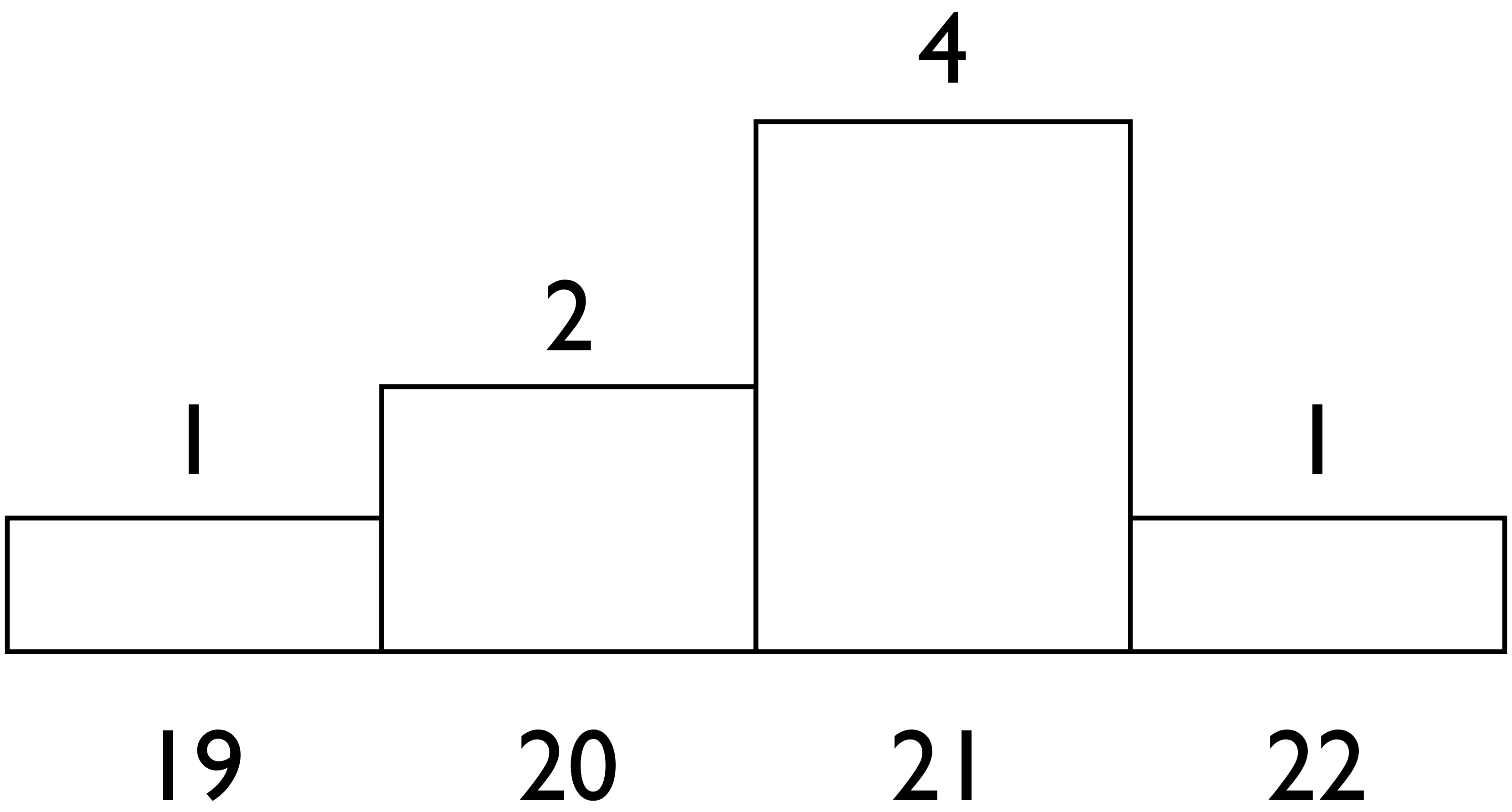
```
...
  AND Age = 19
```

$$RF_{Age\text{-}19} = {}^{1}/_{nkeys} = {}^{1}/_{8}$$

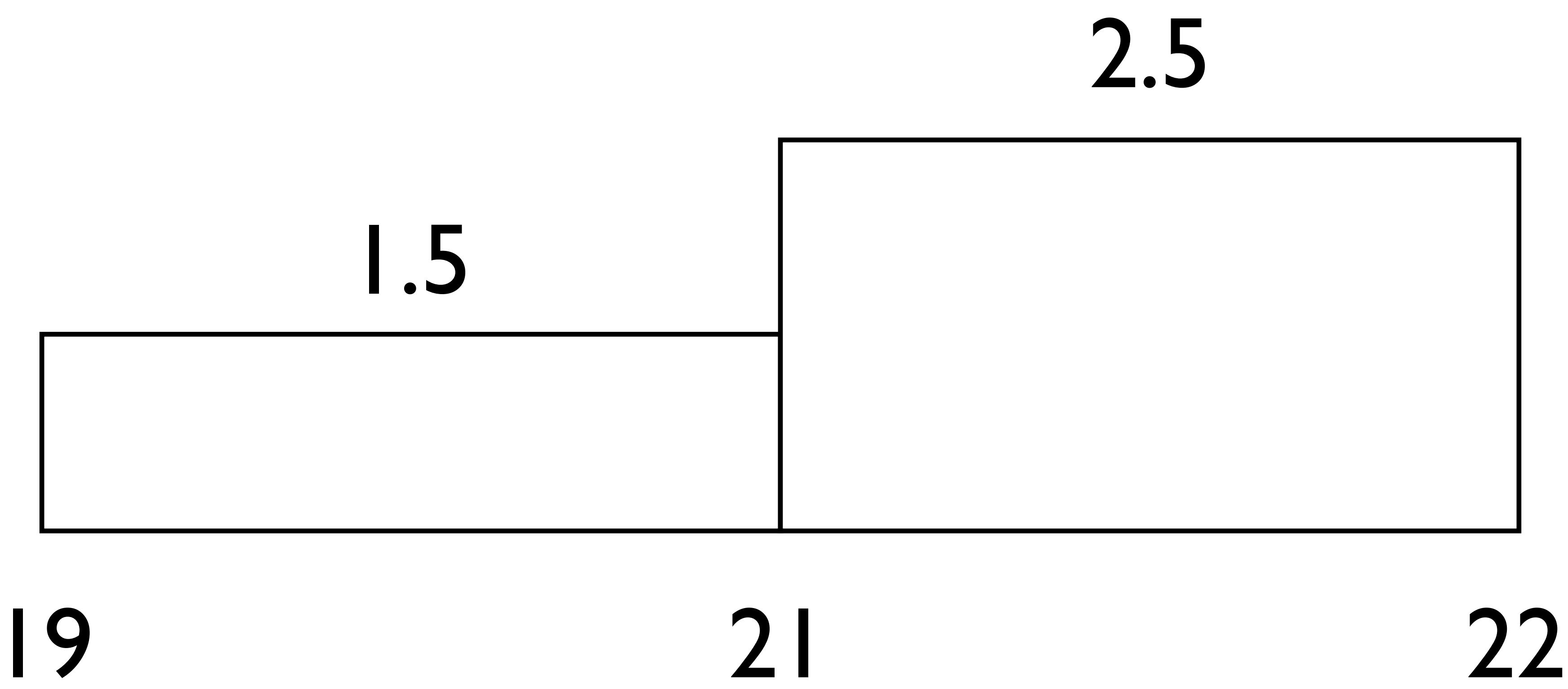
$$RF_{Rank} = {}^{1}/_{nkeys} = {}^{3}/_{8}$$

Age is best!

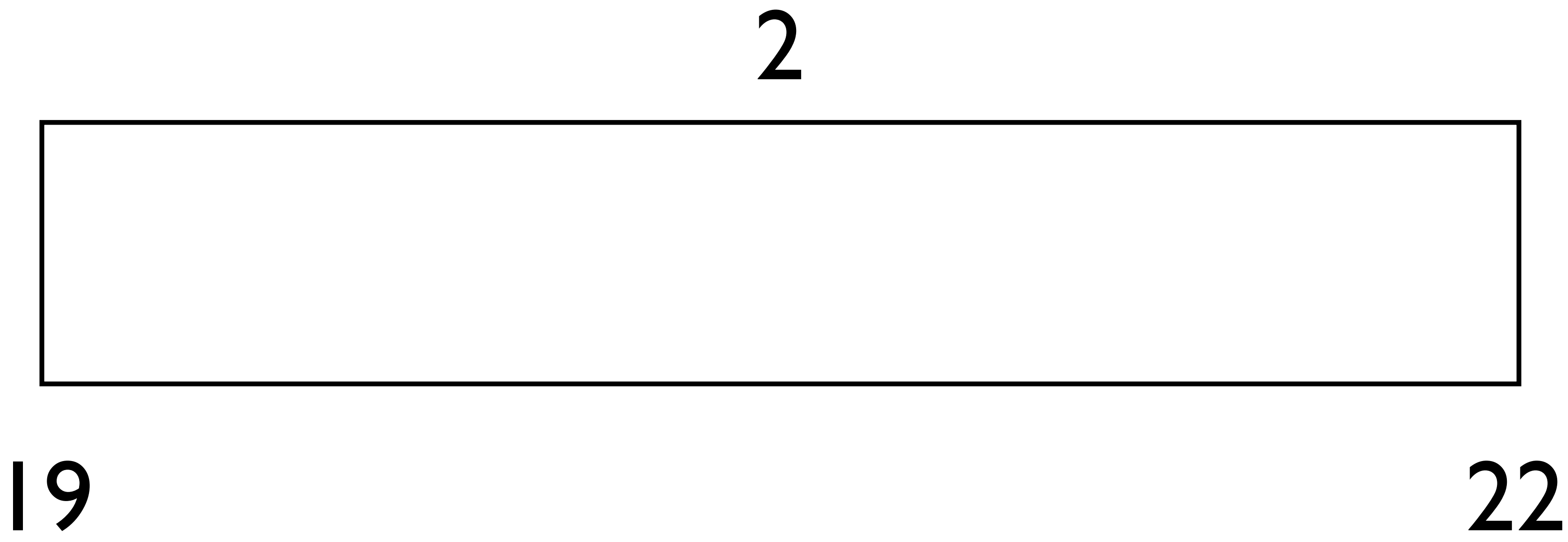
Histograms



Histograms



Histograms



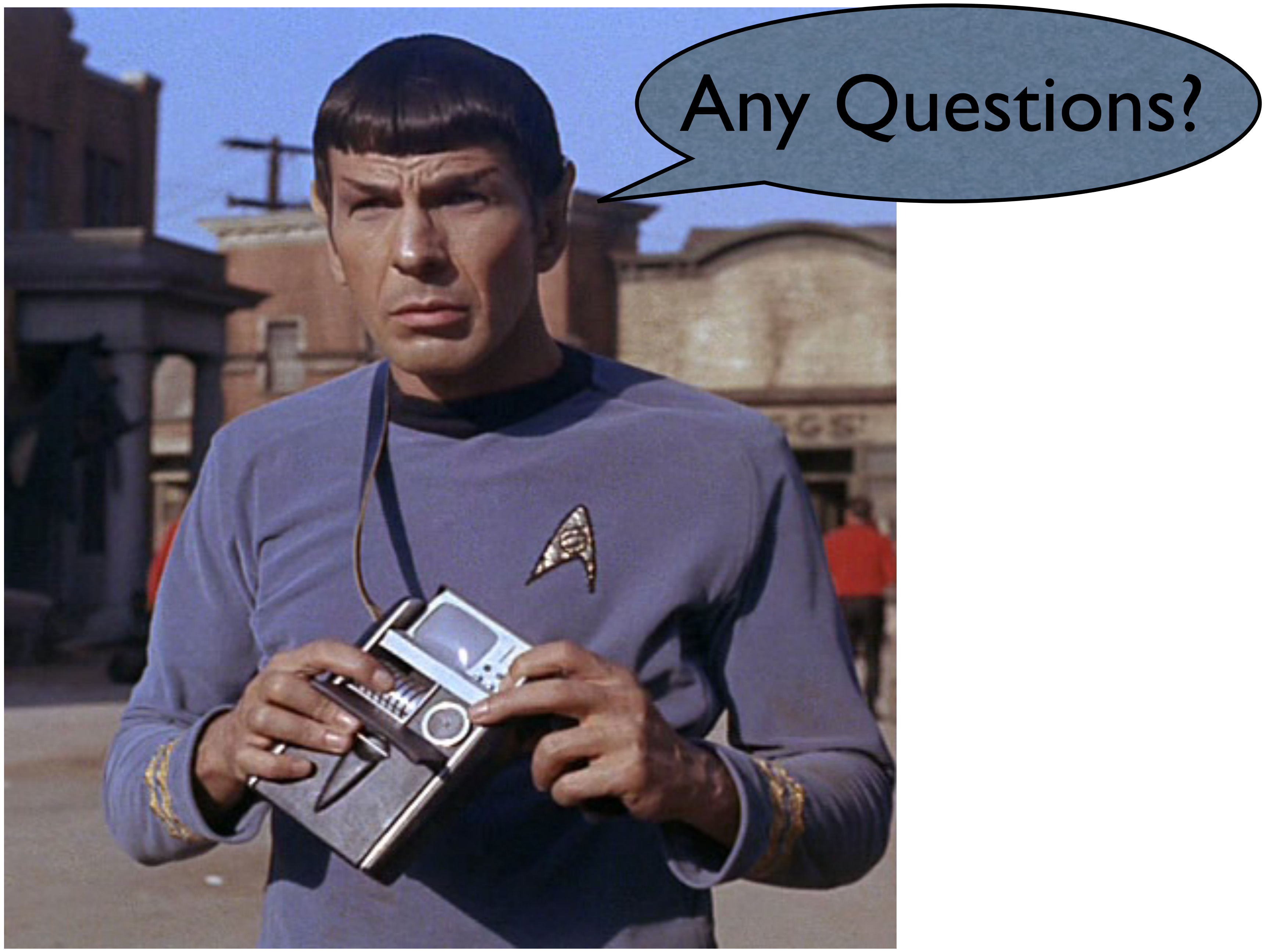
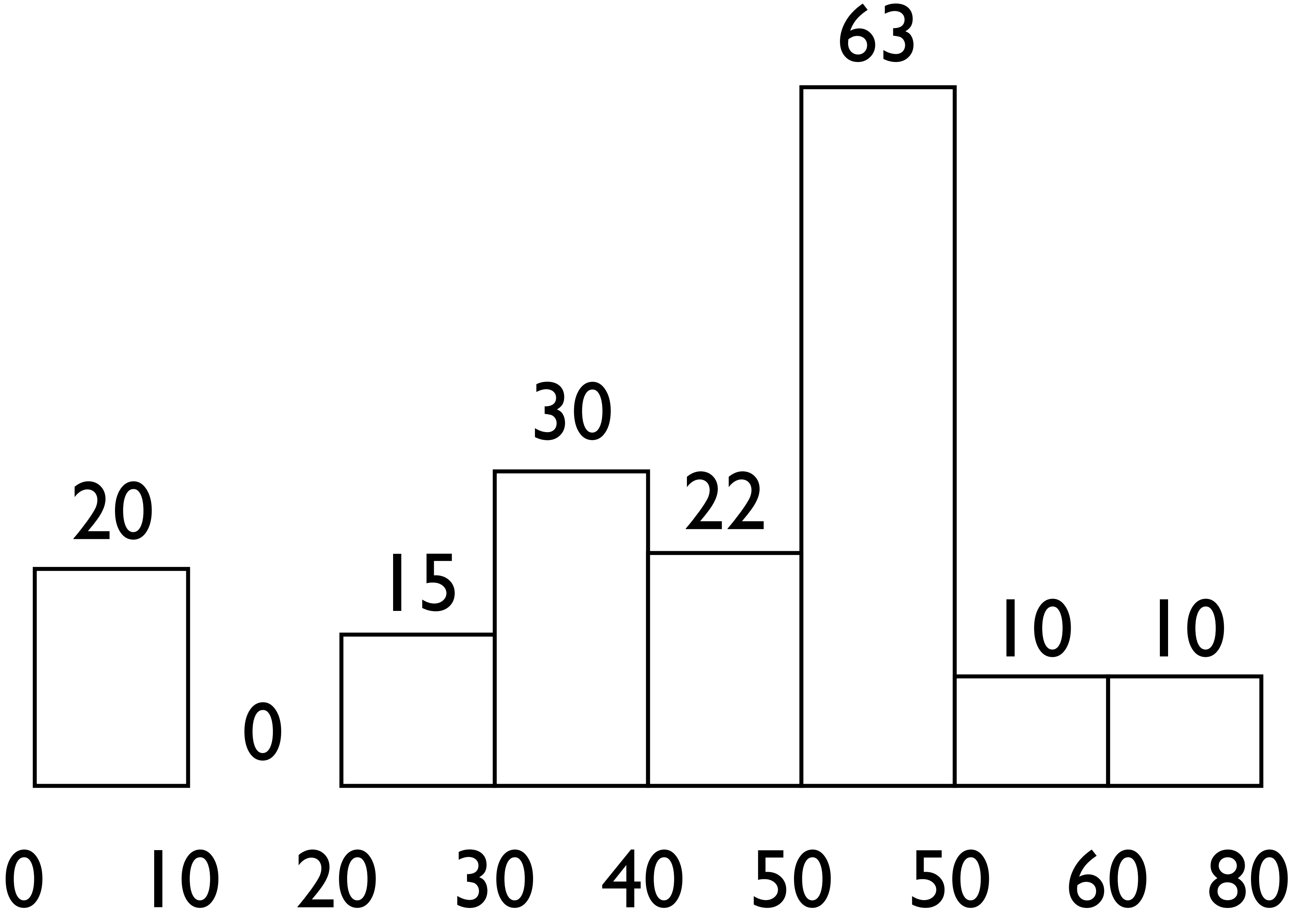


Image copyright: Paramount Pictures

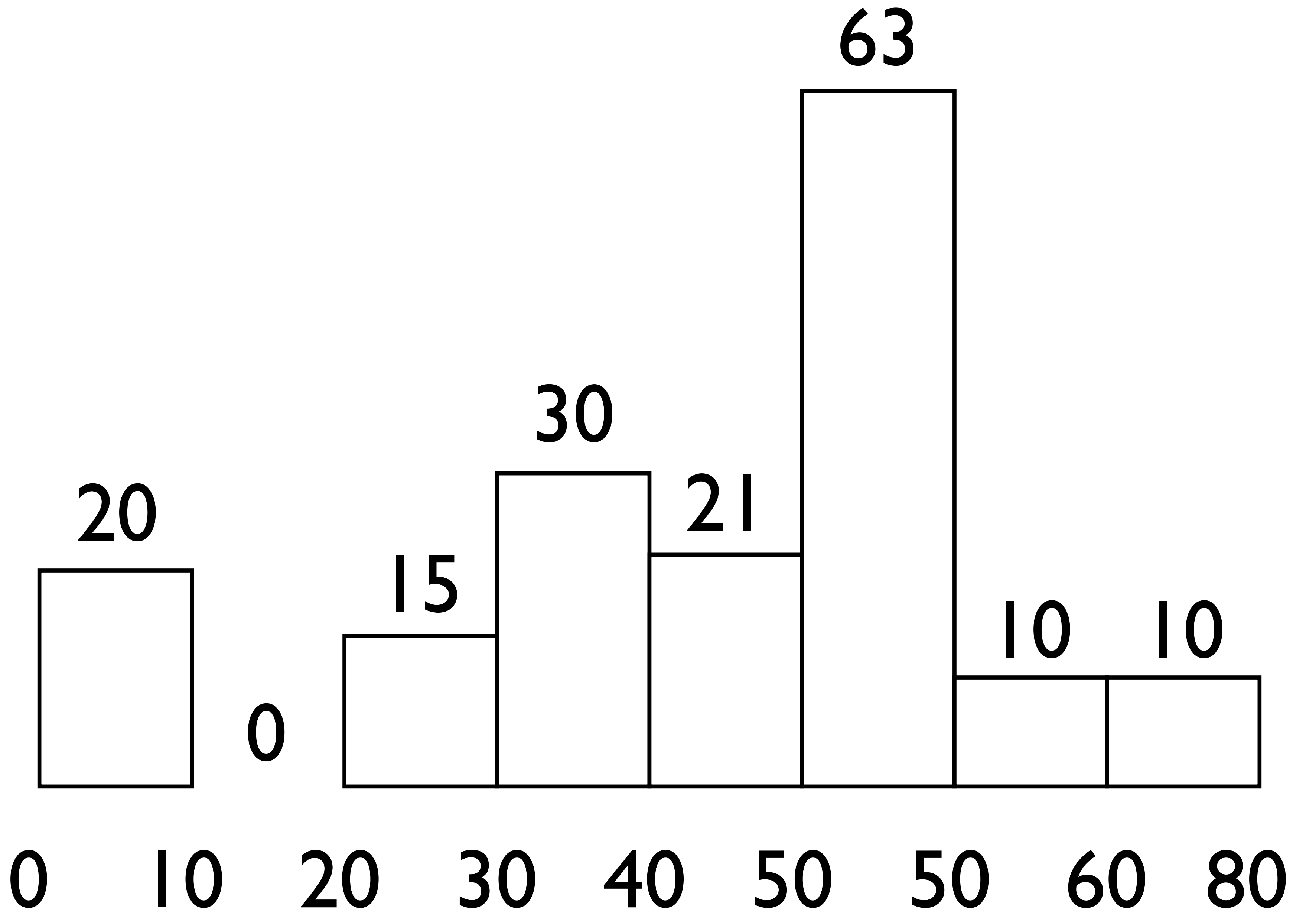
Histograms



```
SELECT … WHERE A = 33
```

Total number of keys: 20+15+30+22+63+10+10 = 170
(30/10)/170 = 3/170 = 3/180 = 0.0176 = 1.8%

Histograms



SELECT … WHERE A > 33

Total number of keys: 20+15+30+22+63+10+10 = 170
Keys matching predicate: ((40–33)/10)*30+21+63+10+10 = 21+104 = 125
125/170 = 0.735 = 74%

Any Questions?

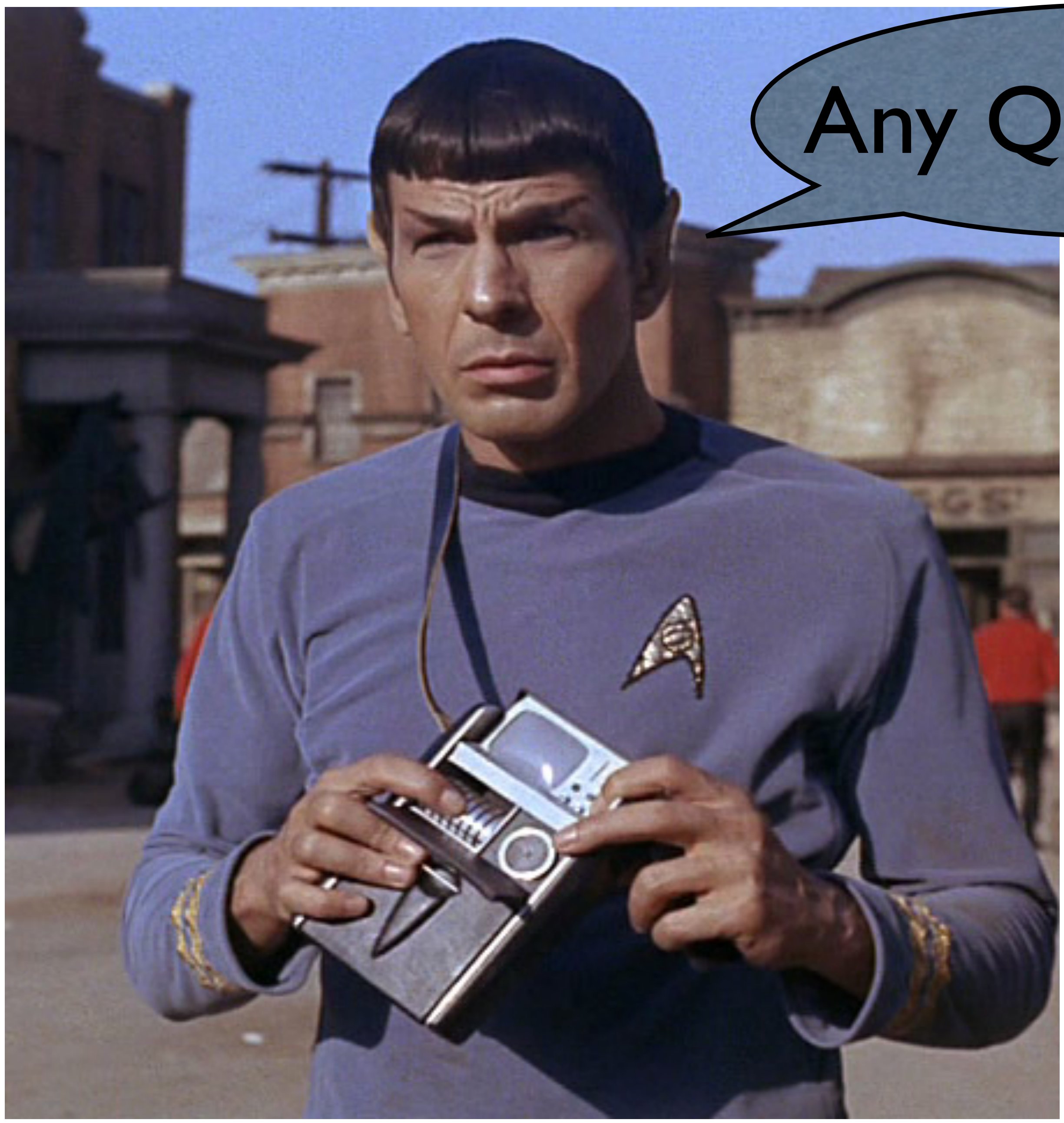


Image copyright: Paramount Pictures

Using Constraints

- A Key attribute has one distinct value per row (equality selects exactly one row)
- Foreign Key joins generate one row for each row in the **referencing** relation.
- Cascade relationship guarantees EXACTLY one row per reference.

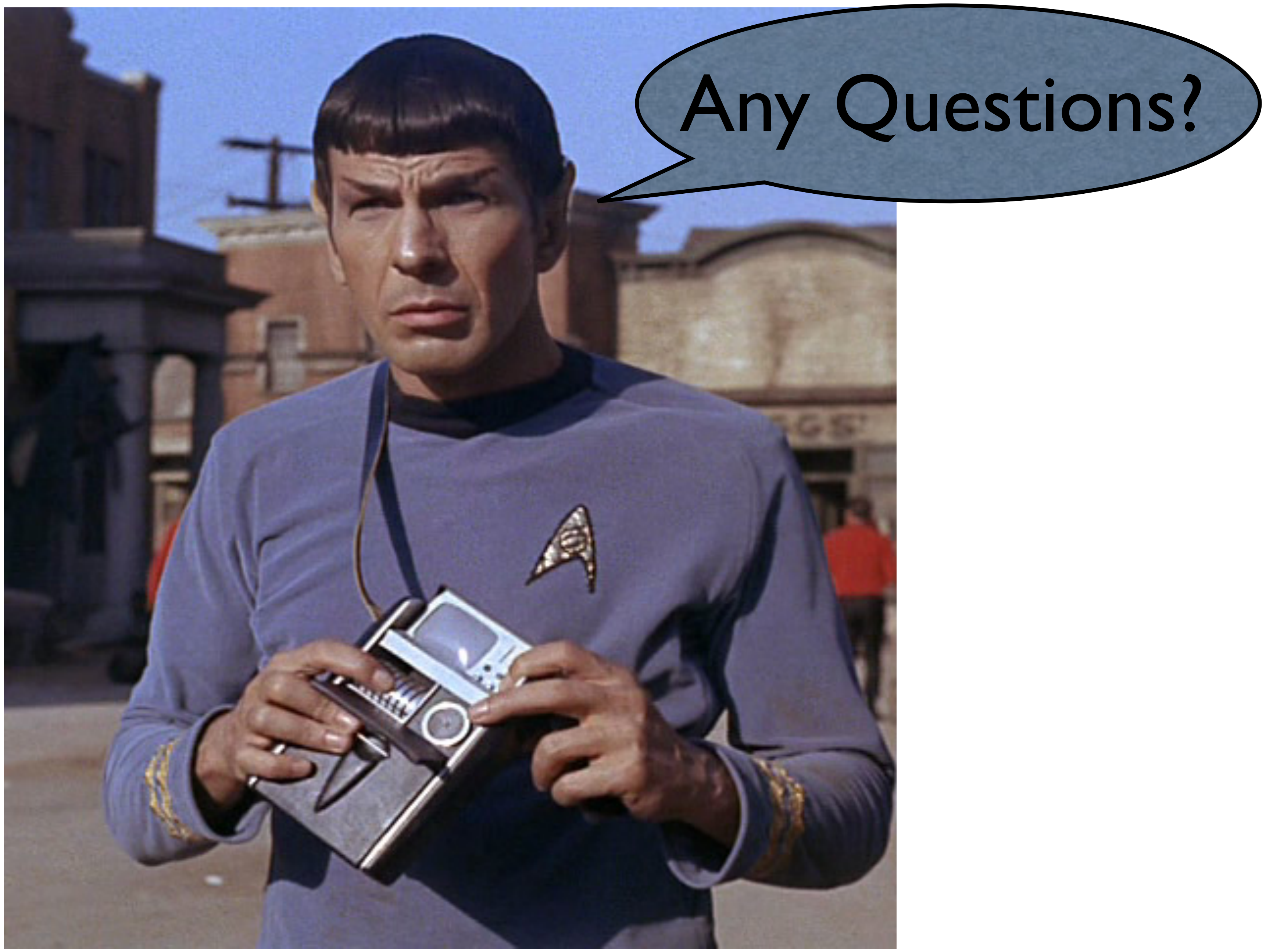


Image copyright: Paramount Pictures