Midterm Review

March 27, 2017

Overview

- Relational Algebra & Query Evaluation
- Relational Algebra Rewrites
- Index Design / Selection
- Physical Layouts

Relational Algebra & Query Evaluation

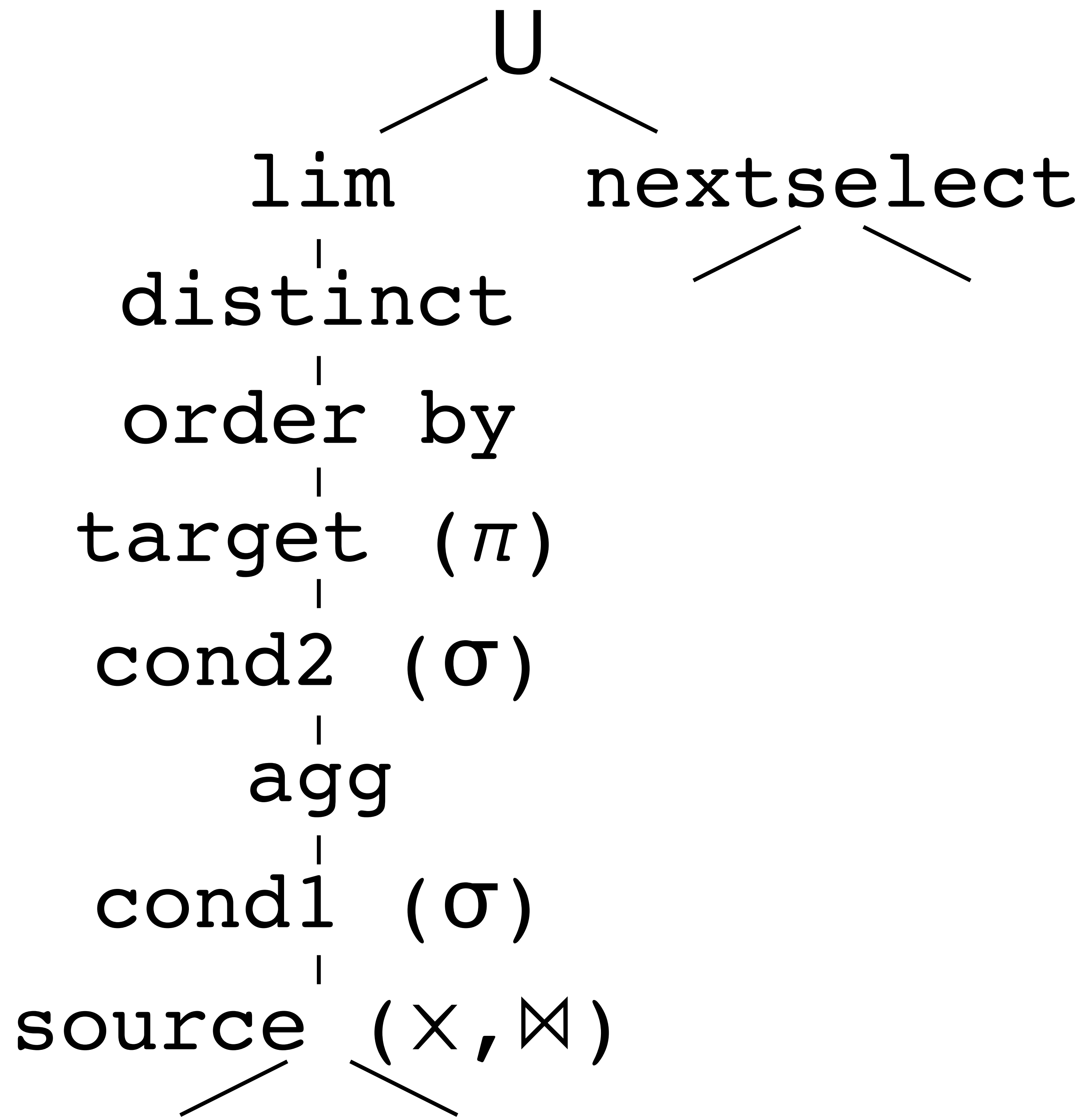
Relational Algebra

Operation	Sym	Meaning
Selection	σ	Select a subset of the input rows
Projection	π	Delete unwanted columns
Cross-product	x	Combine two relations
Set-difference	-	Tuples in Rel 1, but not Rel 2
Union	U	Tuples either in Rel 1 or in Rel 2

Also: Intersection, **Join**, Division, Renaming (Not essential, but very useful)

SQL to RA

```
SELECT [DISTINCT]
       target
FROM source
WHERE cond1
GROUP BY …
HAVING cond2
ORDER BY order
LIMIT lim
UNION nextselect
```



GetNext()

Relation

Read One Line from File

↓

Split Line into Fields

↓

Parse Field Types

↓

Return Tuple

What is the Working Set Size?

GetNext()

Projection (π)

Read One Tuple

↓

Compute Projected Attributes

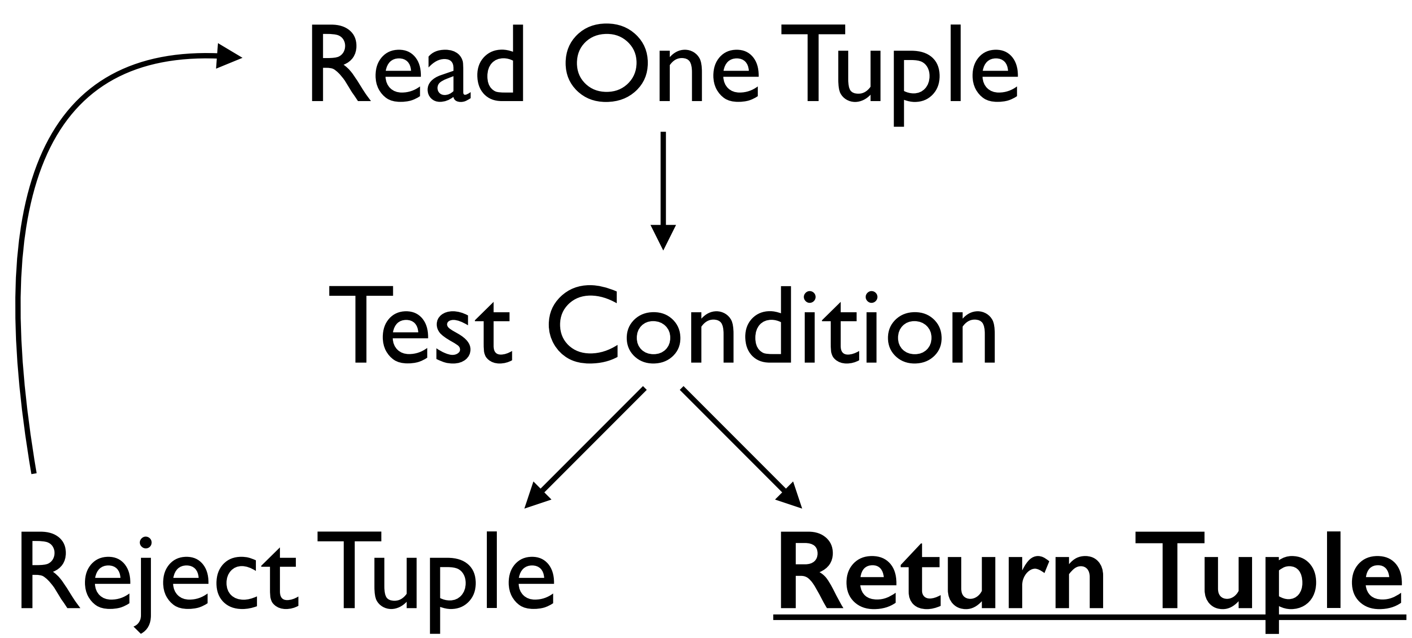
↓

Return Tuple

What is the Working Set Size?

GetNext()

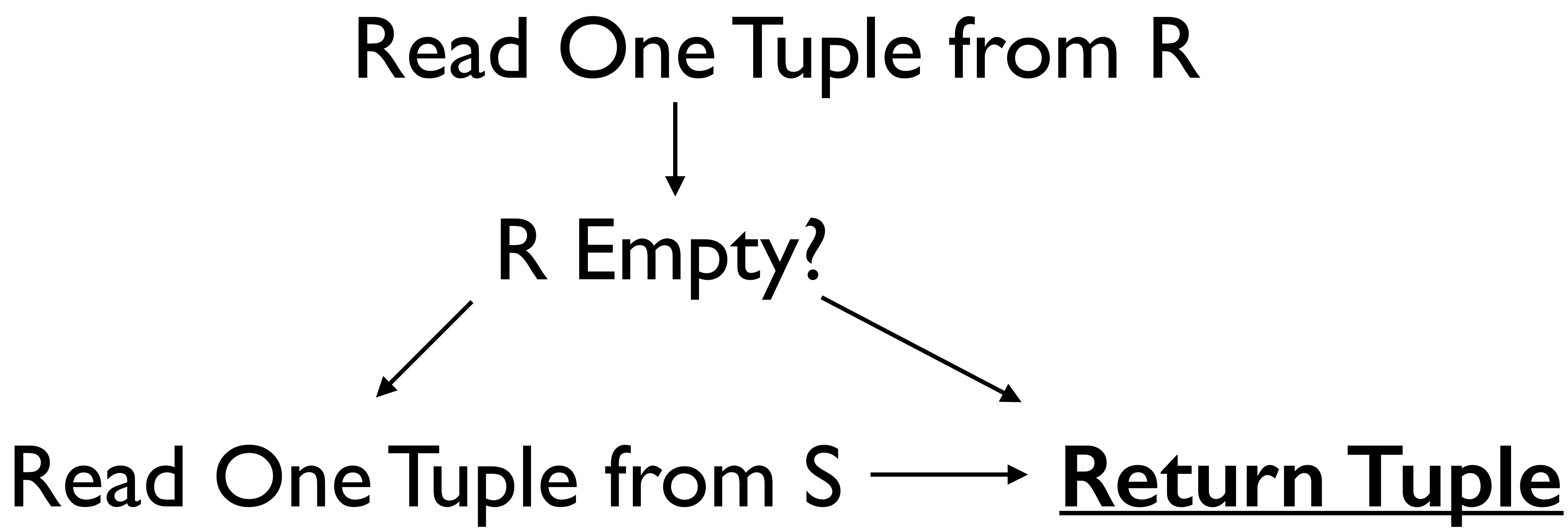
Selection (σ)



What is the Working Set Size?

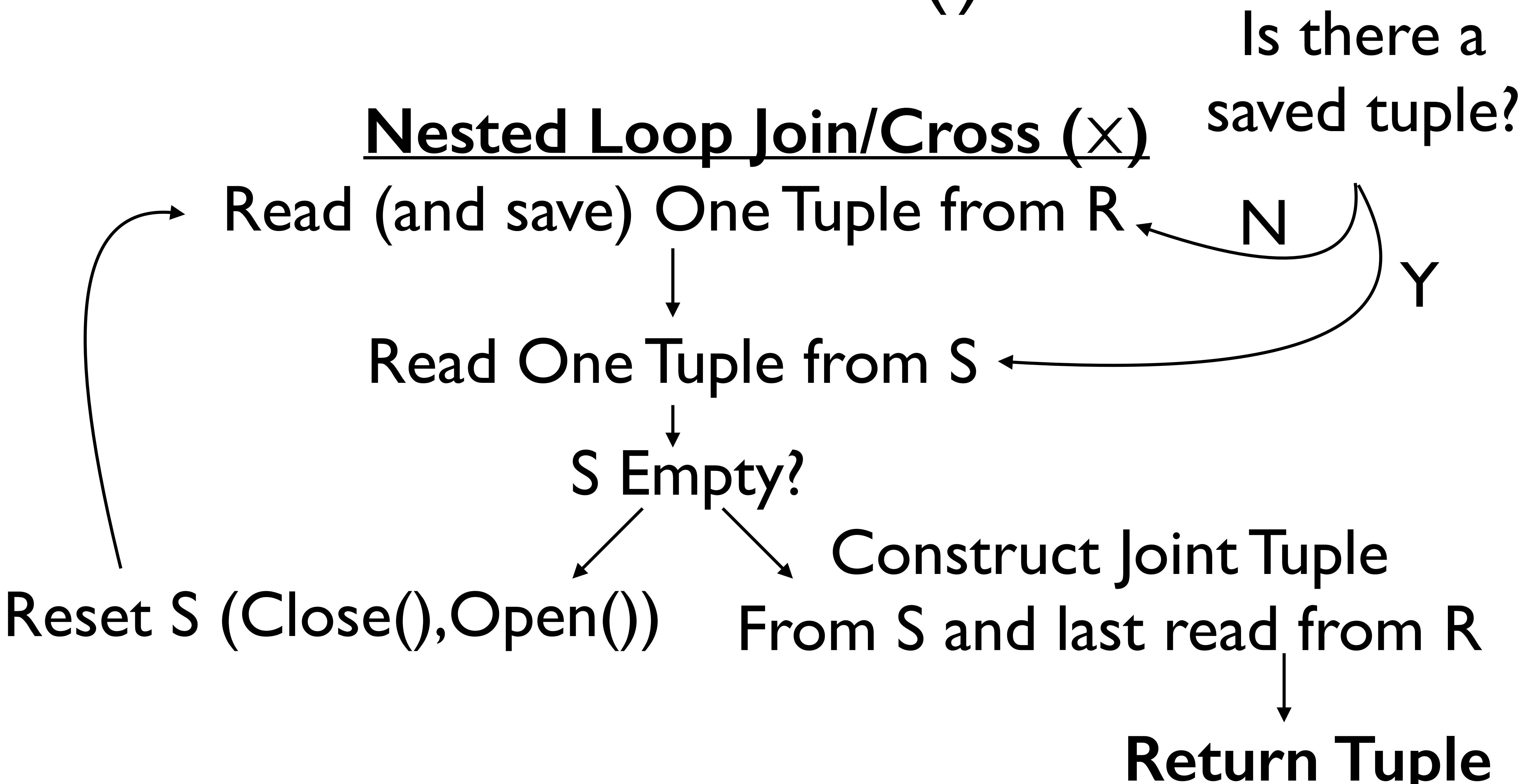
GetNext()

Union (U)



What is the Working Set Size?

GetNext()



What is the Working Set Size?
but…

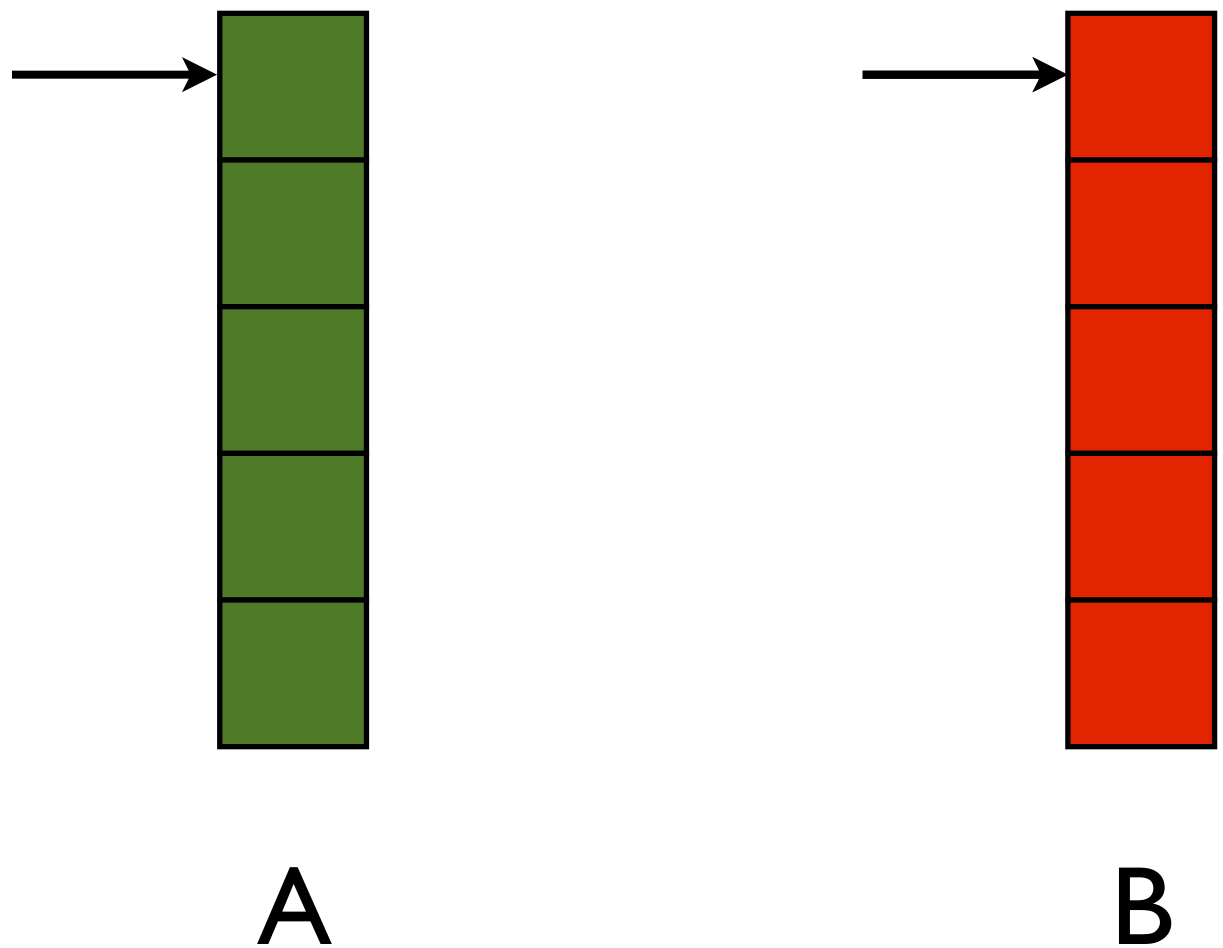
Memory Conscious Algorithms

- Join
  - NLJ has a small working set (but is slow)
- GB Aggregate
  - Working Set ~ # of Groups
- Sort
  - Working Set ~ Size of Relation

Implementing: Joins

Solution 1 (Nested-Loop)

For Each (a in A) { For Each (b in B) { emit (a, b); }}

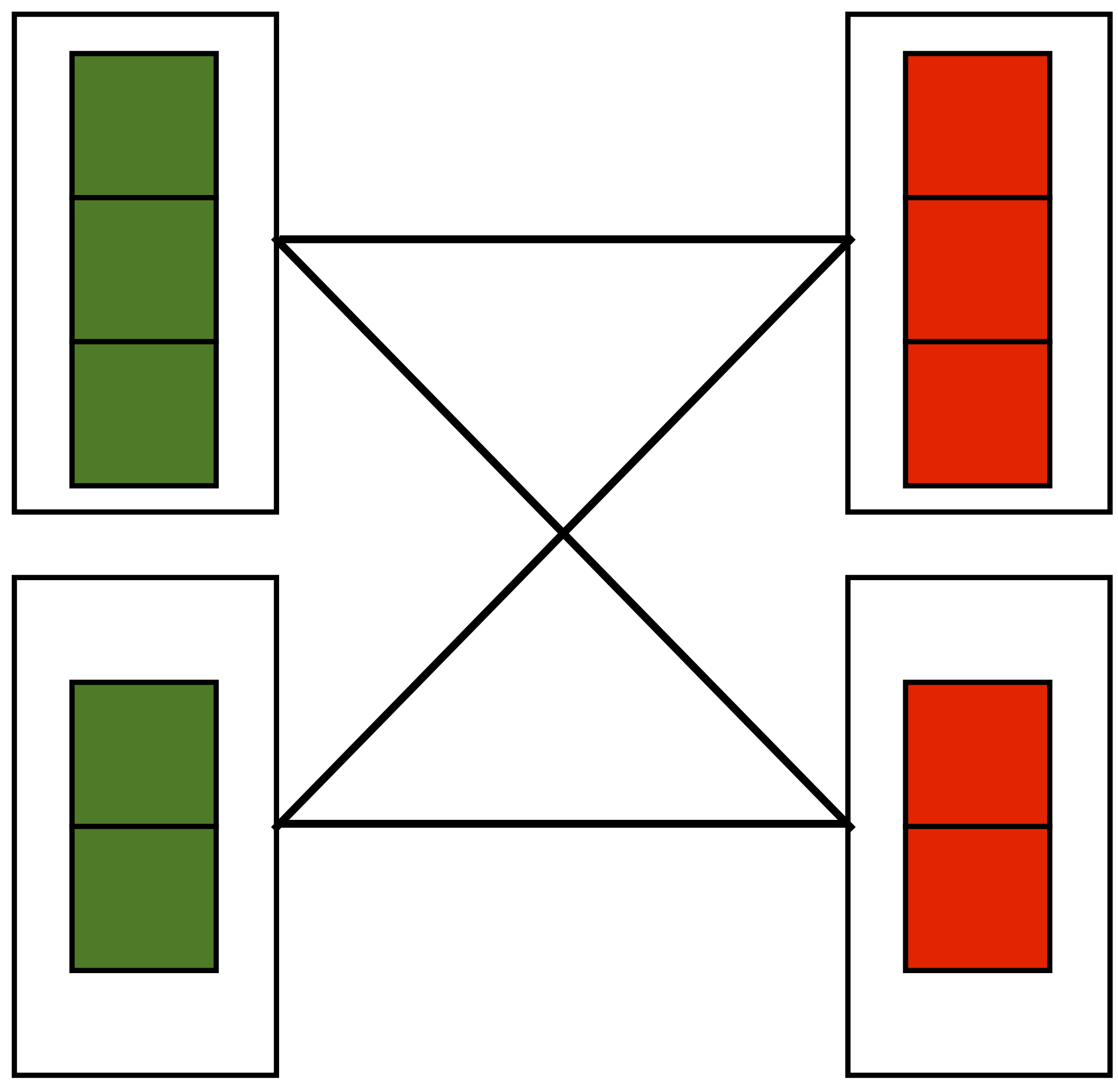


Implementing: Joins

Solution 2 (Block-Nested-Loop)

1) Partition into Blocks

2) NLJ on each pair of blocks



Implementing: Joins

Solution 3 (Index-Nested-Loop)

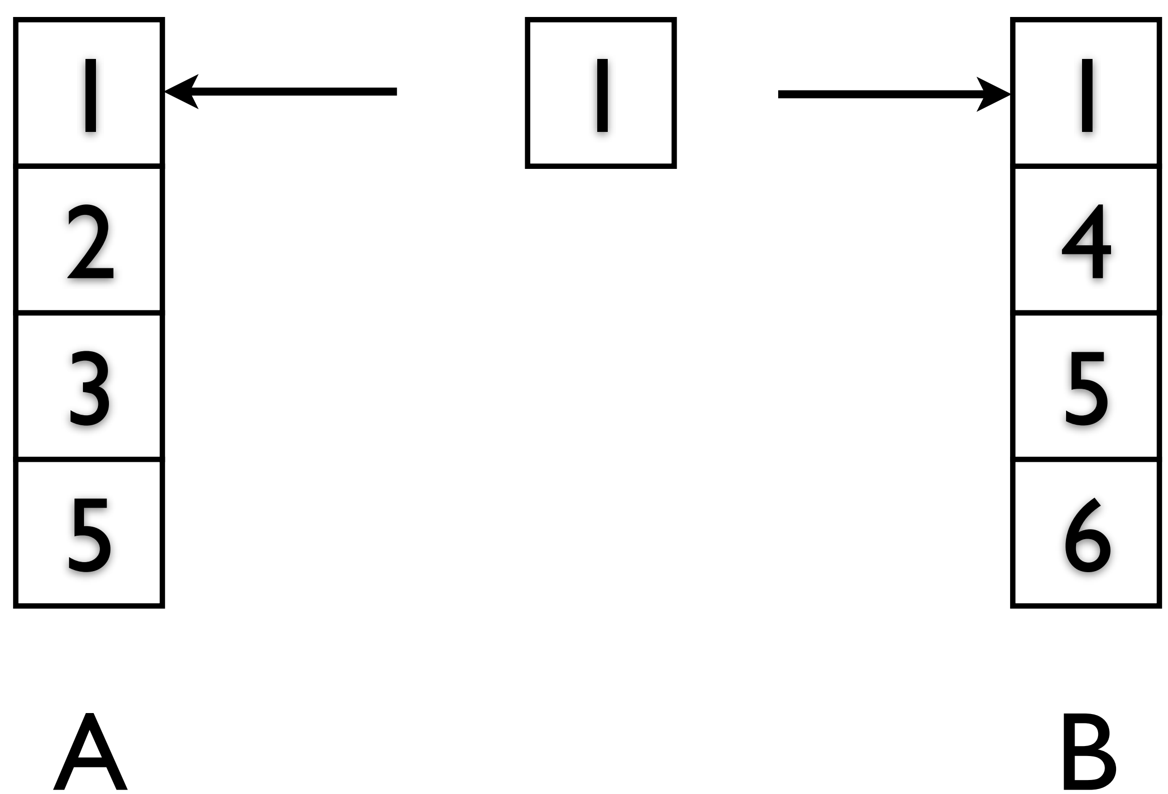
Like nested-loop, but use an index to make the inner loop much faster!

Implementing: Joins

Solution 4 (Sort-Merge Join)

Keep iterating on the set with the lowest value.

When you hit two that match, emit, then iterate both

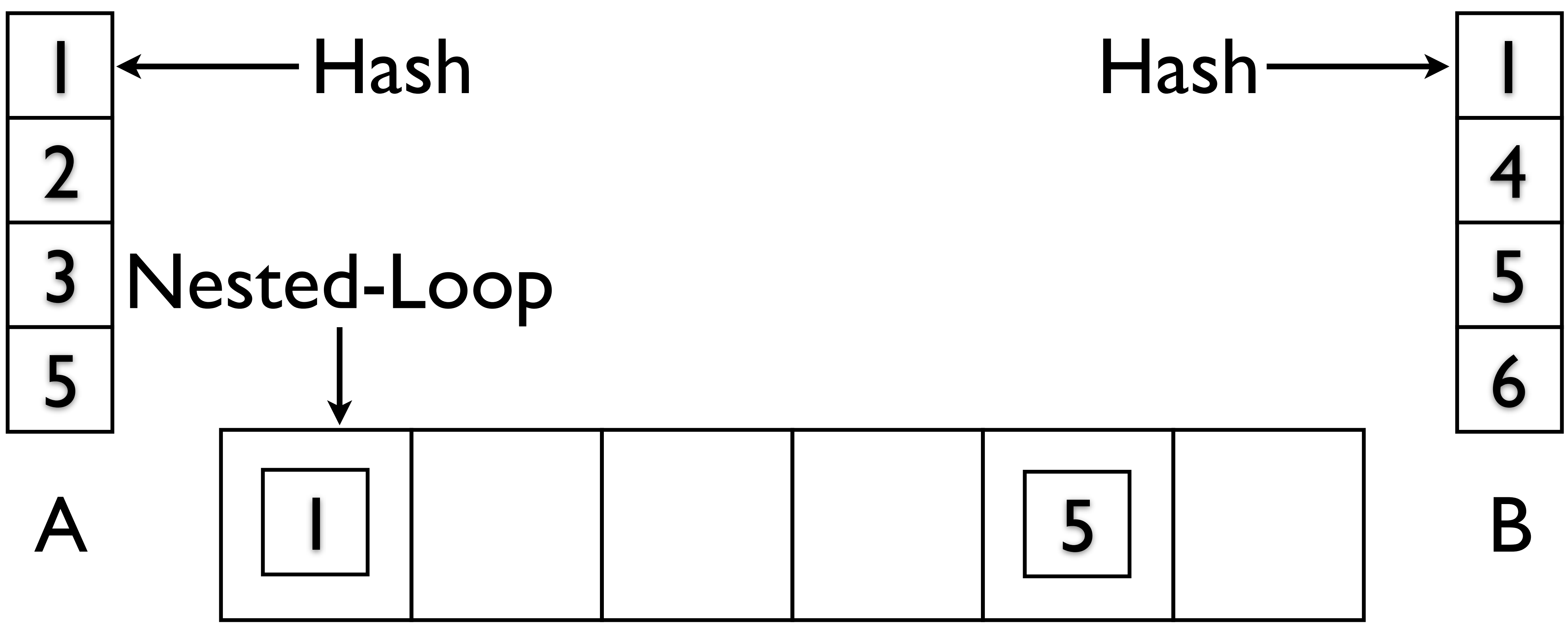


Implementing: Joins

Solution 5 (2-Pass Hash)

1) Build a hash table on both relations

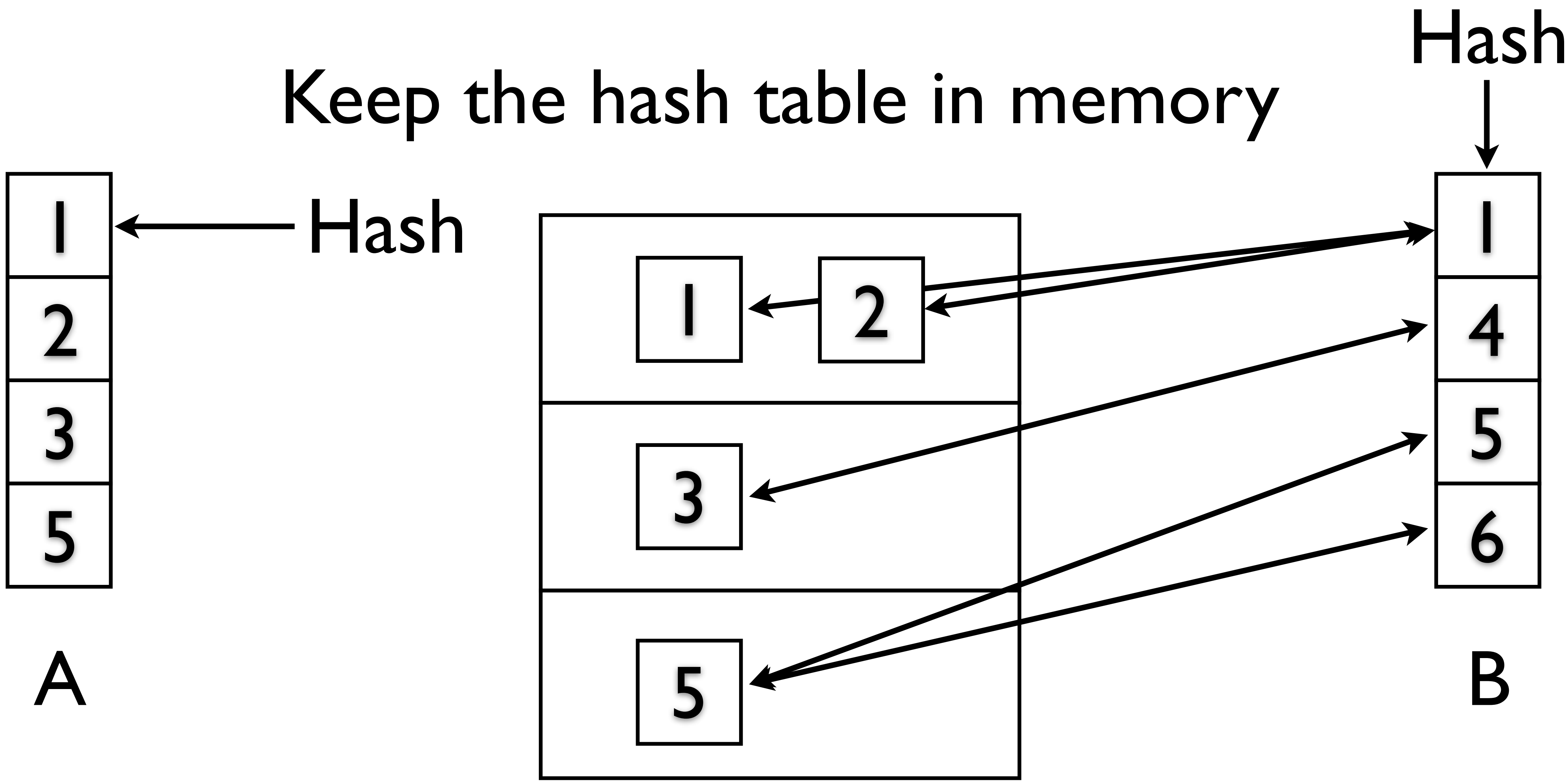
2) In-Memory Nested-Loop Join on each hash bucket



Implementing: Joins

Solution 6 (1-Pass Hash)

Keep the hash table in memory



(Essentially a more efficient nested loop join)

Relational Algebra Rewrites

RA Equivalencies

Selection

$$\sigma_{c_1 \wedge c_2}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(R))$$ (Decomposable)

$$\sigma_{c_1 \vee c_2}(R) \equiv \delta(\sigma_{c_1}(R) \cup \sigma_{c_2}(R))$$ (Decomposable)

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$ (Commutative)

Projection

$$\pi_a(R) \equiv \pi_a(\pi_{a \cup b}(R))$$ (Idempotent)

Cross Product (and Join)

$$R \times (S \times T) \equiv (R \times S) \times T$$ (Associative)

$$(R \times S) \equiv (S \times R)$$ (Commutative)

Try It: Show that $R \times (S \times T) \equiv T \times (R \times S)$

Selection and Projection

$$\pi_a(\sigma_c(R)) \equiv \sigma_c(\pi_a(R))$$

Selection <u>commutes</u> with Projection
(but only if attribute set **a** and condition **c** are *compatible*)

a must include all columns referenced by **c**

<u>Show that</u>

$$\pi_a(\sigma_c(R)) \equiv \pi_a(\sigma_c(\pi_{a \cup \texttt{cols}(c)}(R)))$$

When is this rewrite a good idea?

Join

$$\sigma_c(R \times S) \equiv R \bowtie_c S$$

Selection <u>combines</u> with Cross Product
to form a Join as per the definition of Join
(Note: This only helps if we have a join algorithm for conditions like **c**)

<u>Show that</u>

$$\sigma_{(R.B=S.B)\wedge(R.A>3)}(R \times S) \equiv \sigma_{(R.A>3)}(R \bowtie_{(R.B=S.B)} S)$$

When is this rewrite a good idea?

Selection and Cross Product

$$\sigma_c(R \times S) \equiv (\sigma_c(R) \times S)$$

Selection <u>commutes</u> with Cross Product
(but only if condition **c** references attributes of R exclusively)

<u>Show that</u>

$$\sigma_{(R.B=S.B)\wedge(R.A>3)}(R \times S) \equiv \sigma_{(R.A>3)}(R) \bowtie_{(R.B=S.B)} S$$

When is this rewrite a good idea?

Projection and Cross Product

$$\pi_a(R \times S) \equiv (\pi_{a_1}(R)) \times (\pi_{a_2}(S))$$

Projection <u>commutes</u> (distributes) over Cross Product
(where $\mathbf{a_1}$ and $\mathbf{a_2}$ are the attributes in $\mathbf{a}$ from R and S respectively)

<u>Show that</u>

$$\pi_a(R \bowtie_c S) \equiv (\pi_{a_1}(R)) \bowtie_c (\pi_{a_2}(S))$$

(under what condition)

How can we work around this limitation?

$$\pi_a((\pi_{a_1 \cup (\mathtt{cols}(c) \cap \mathtt{cols}(R))}(R)) \bowtie_c (\pi_{a_2 \cup (\mathtt{cols}(c) \cap \mathtt{cols}(S))}(S)))$$

When is this rewrite a good idea?

RA Equivalencies

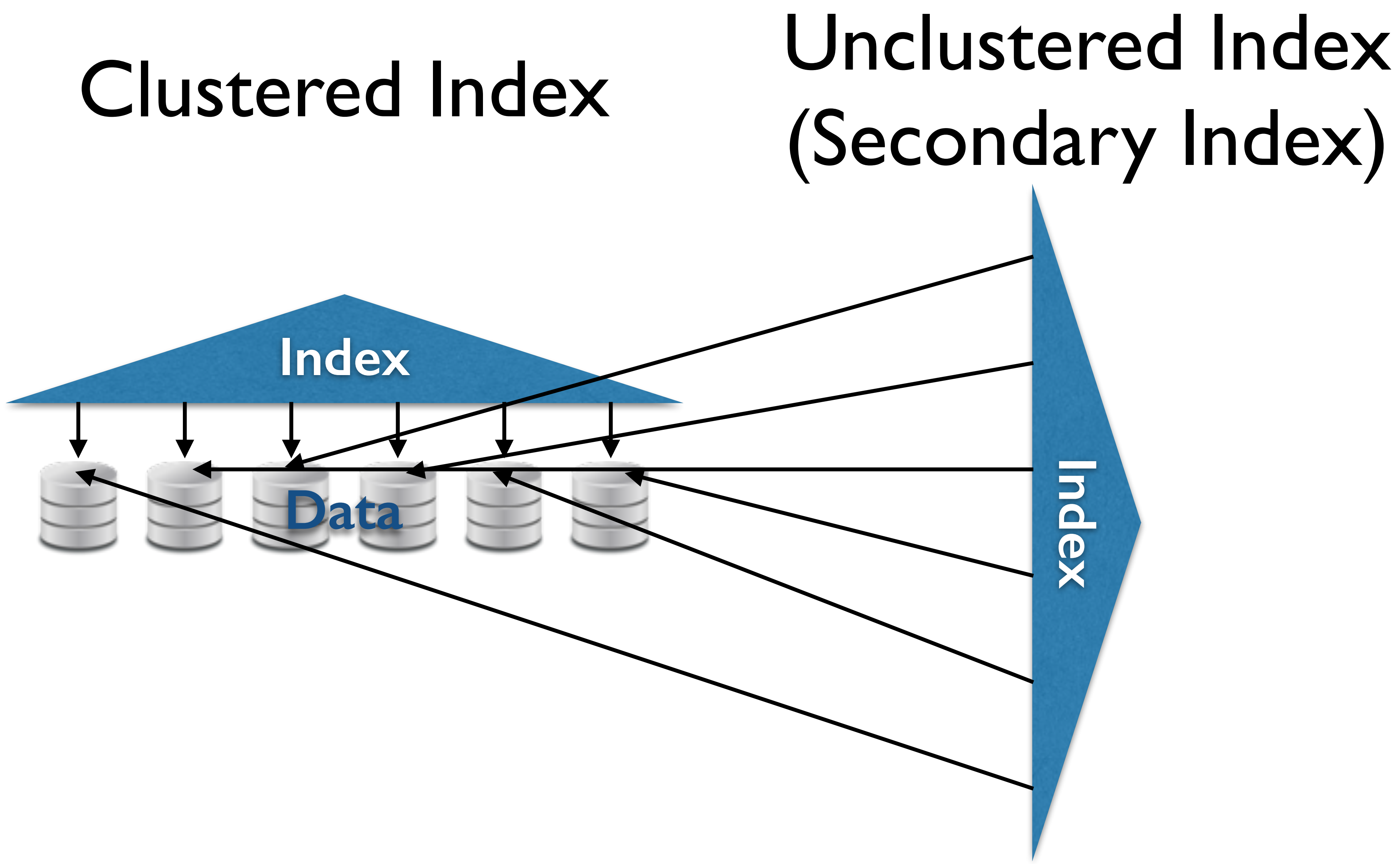
Union and Intersections are Commutative and Associative

Selection and Projection both commute with both Union and Intersection

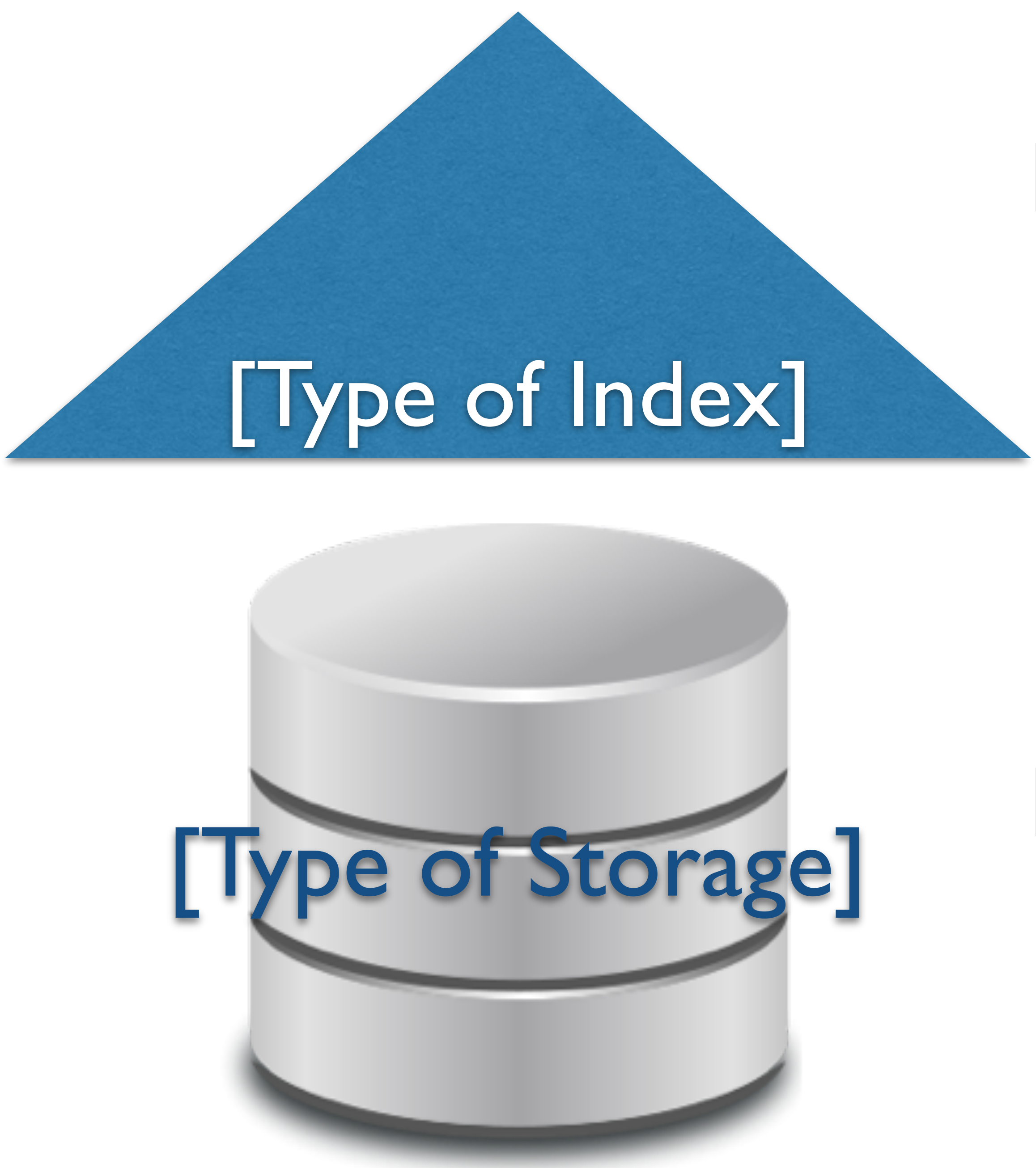
When is this rewrite a good idea?

Index Design / Selection

Indexes


Clustered Index
Unclustered Index
(Secondary Index)
Index
Data
Index

Indexes



How the Data is Organized

ISAM
B+Tree
Other Tree-Based
Hash Table
Other Hash-Based
Other…

How the Data is Laid Out

<u>In the Index</u>
Clustered

<u>Outside of the Index</u>
Sorted
Heap

Access Paths

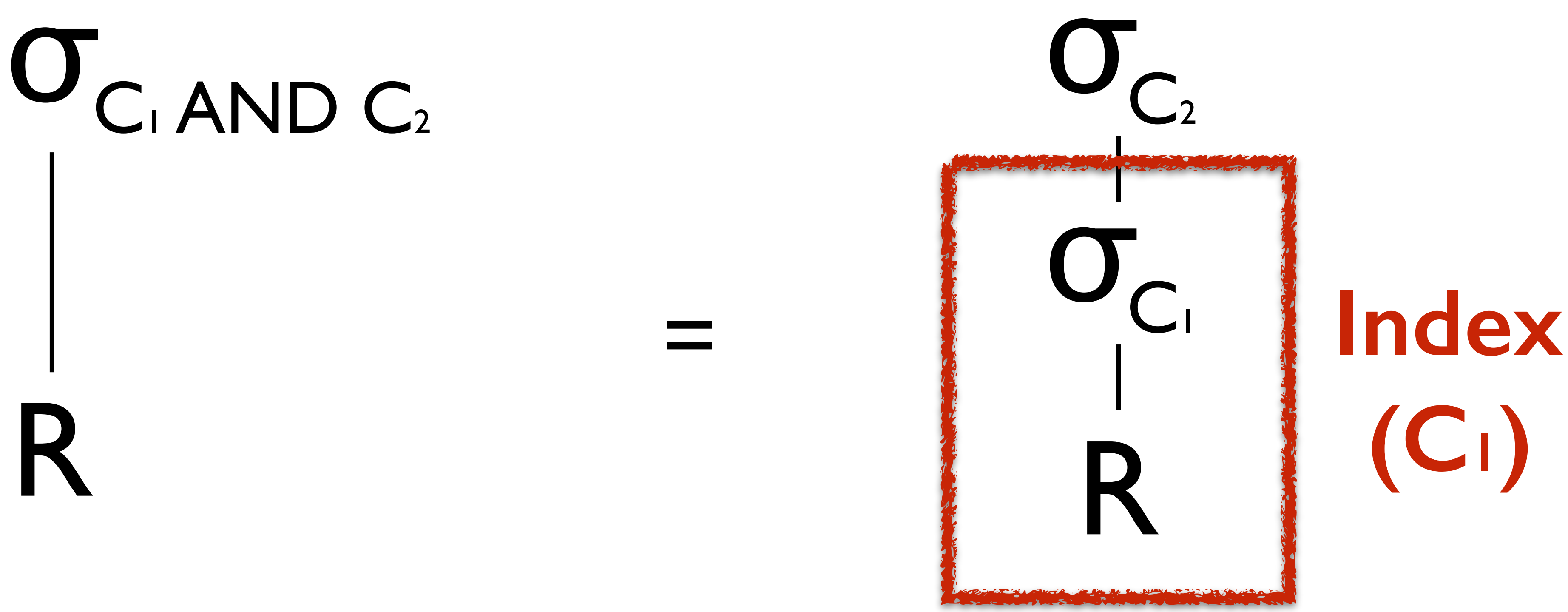
$$\sigma_{C_1 \text{ AND } C_2}$$

$|$

R

Can we "simplify" this condition

Access Paths



Access Paths

$\sigma_{A > 1 \text{ AND } A < 10}$

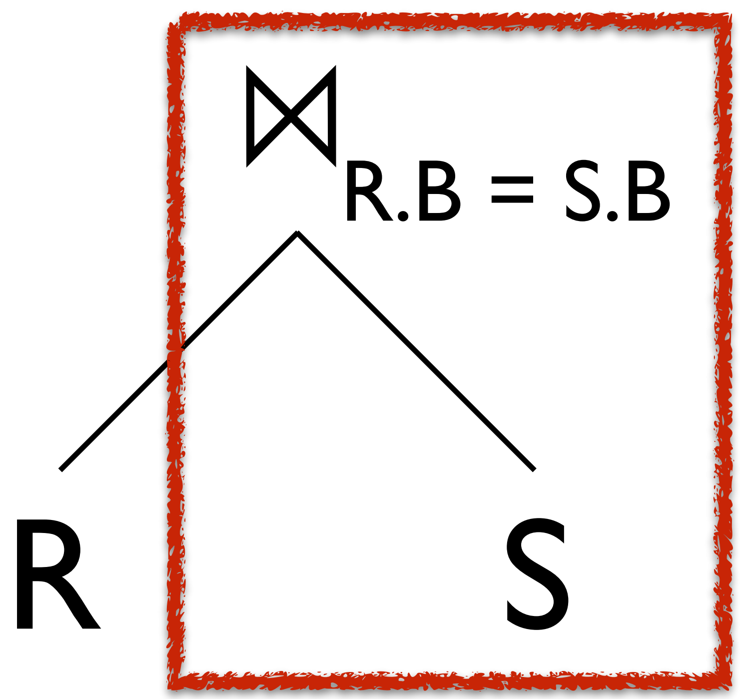
|

R

Index

$A \in (1,10)$

Access Paths

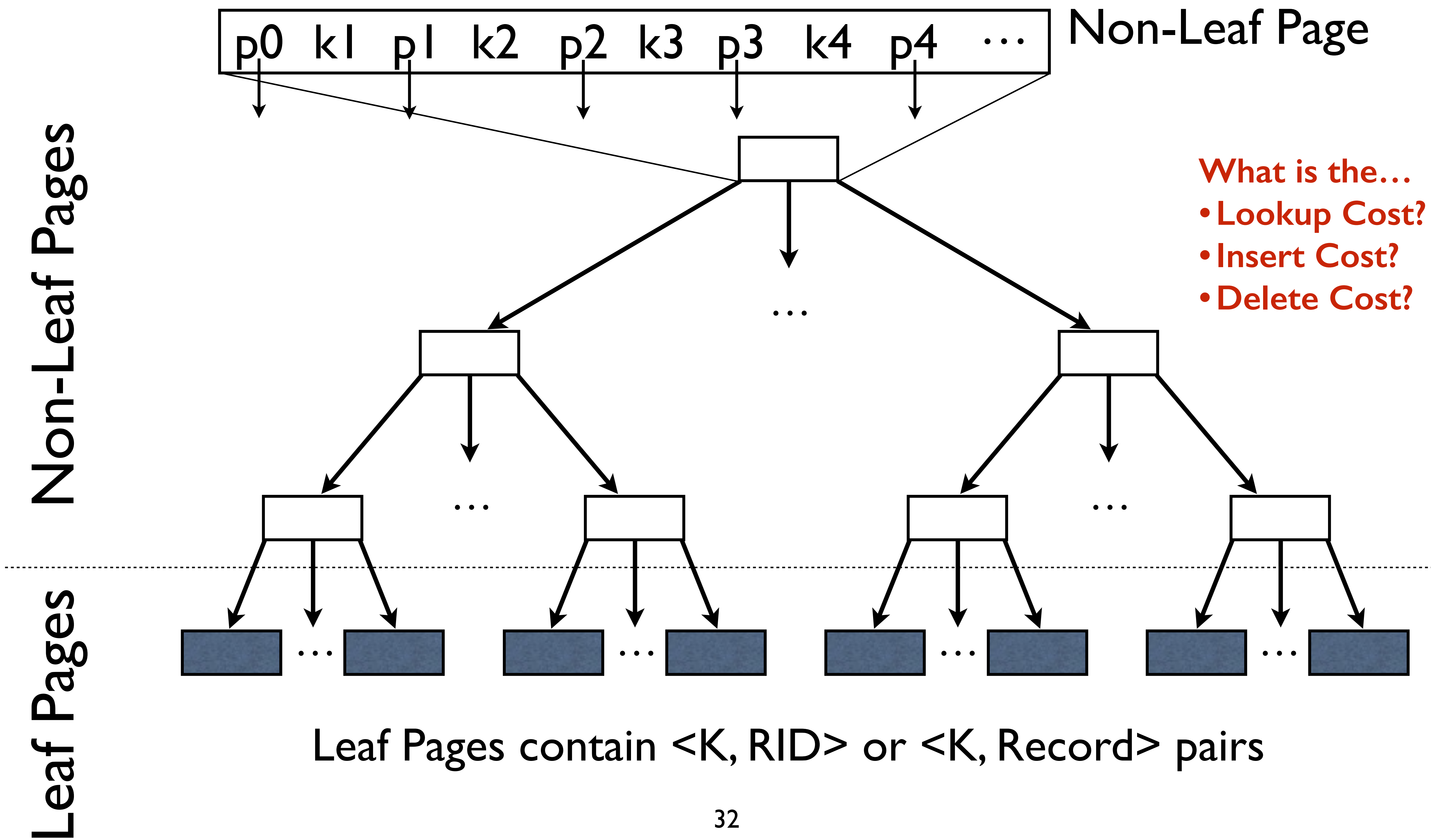


How could we compute this
If we had an index on S.B?

```
Foreach r in R
  Foreach s in
    IndexLookup(S, B=r.b)
    Emit(r × s)
```

What are the Working Set Size & IO Cost?

The ISAM Datastructure



What is the…

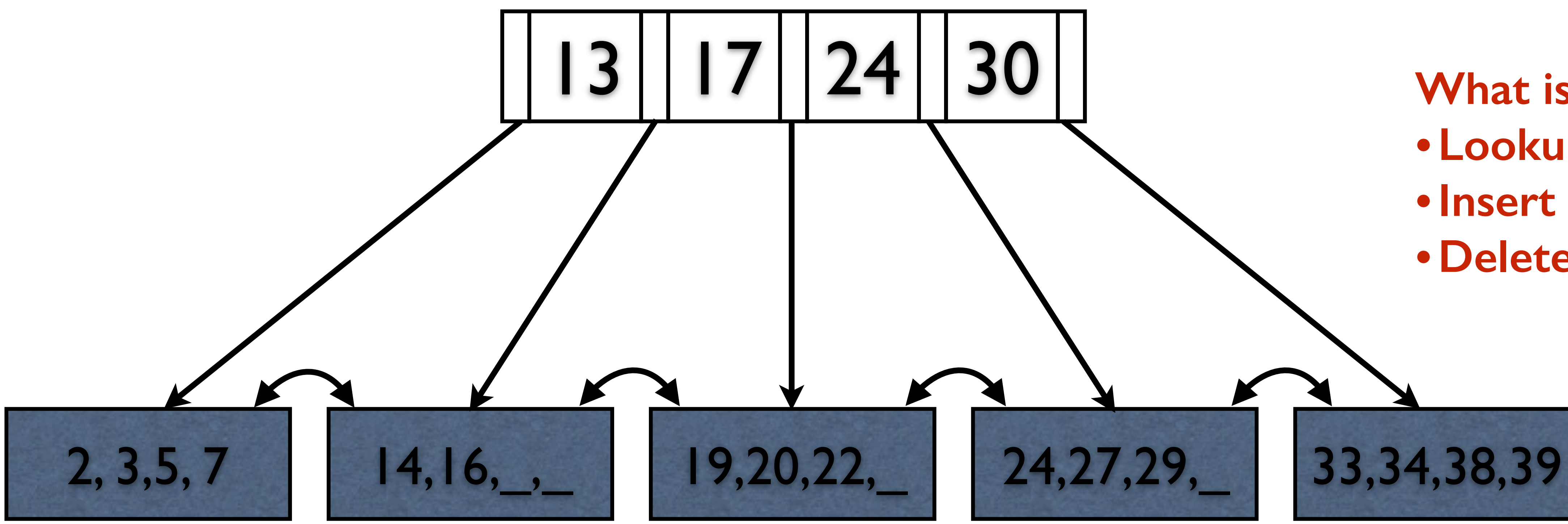
- Lookup Cost?
- Insert Cost?
- Delete Cost?

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

B+ Trees

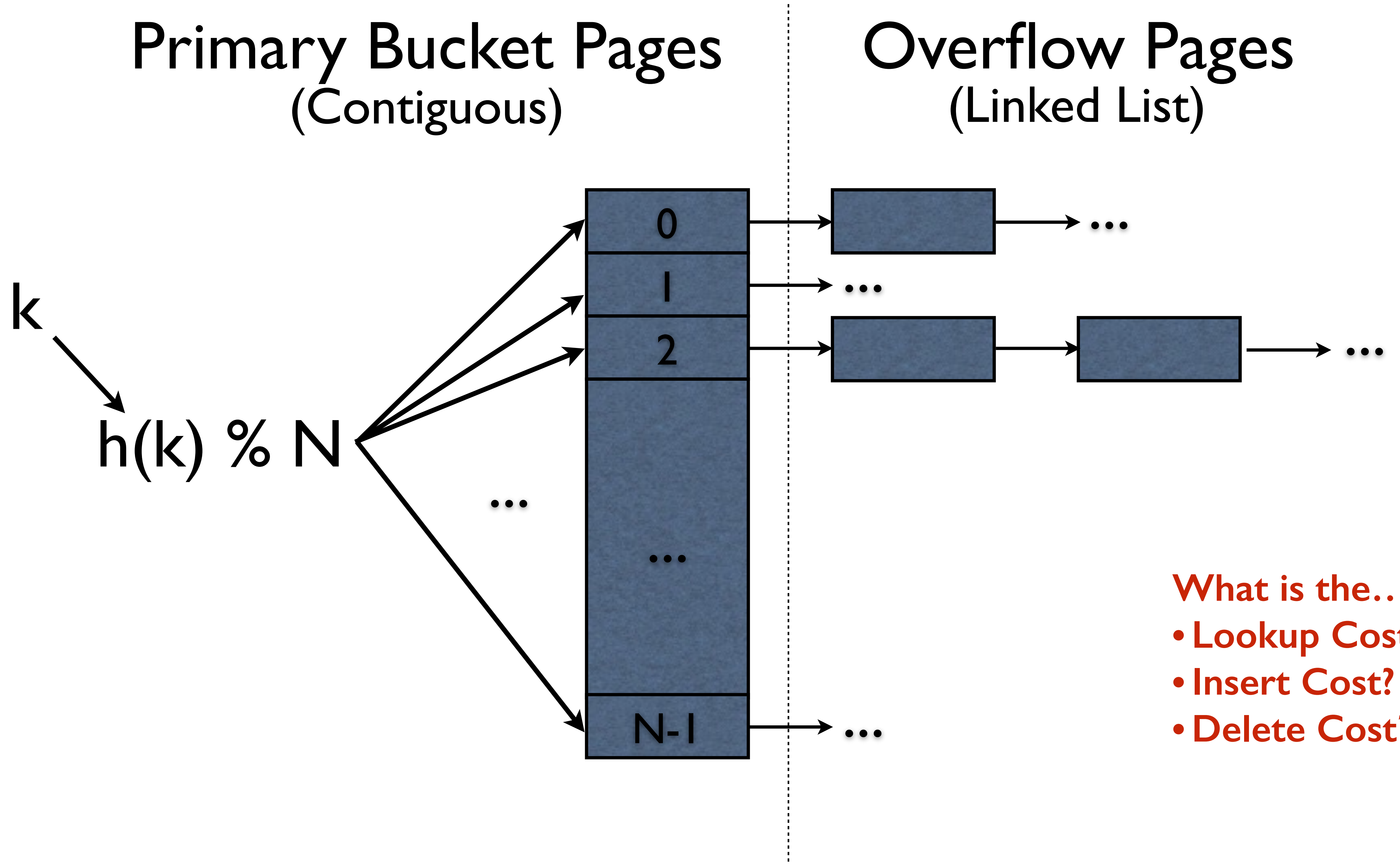
Search proceeds as in ISAM via key comparisons

Find 5. Find 15. Find [24, ∞)



What is the…
- Lookup Cost?
- Insert Cost?
- Delete Cost?

Static Hashing

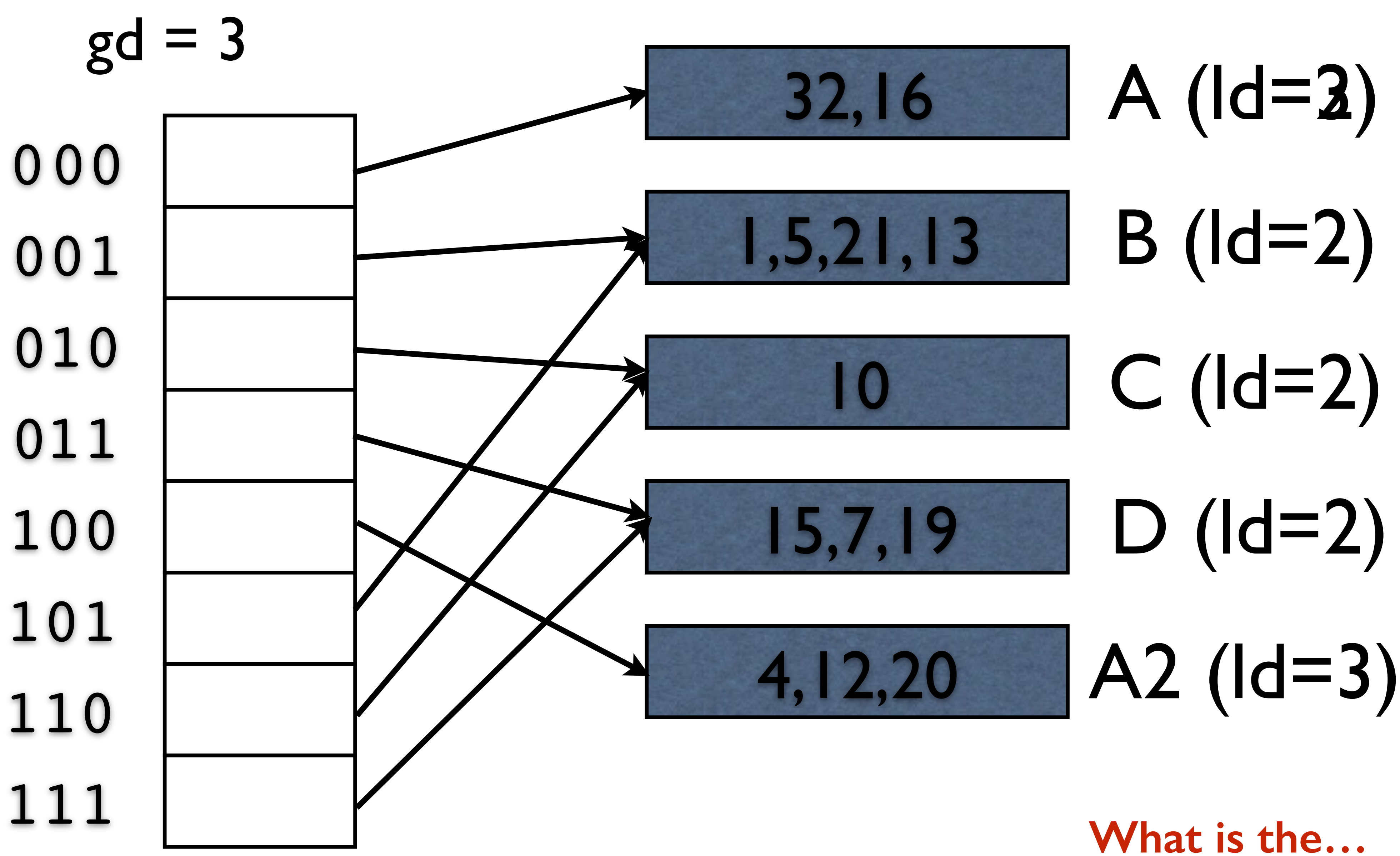


What is the…
- Lookup Cost?
- Insert Cost?
- Delete Cost?

Dynamic Hashing

- **Situation:** A bucket becomes full
  - Solution: Double the number of buckets!
  - Expensive! (N reads, 2N writes)
- **Idea:** Add one level of indirection
  - A directory of pointers to (noncontiguous) bucket pages.
  - Doubling just the directory is much cheaper.
  - Can we double only the directory?

Dynamic Hashing



Dir entries not being split point to the same bucket

What is the…

- Lookup Cost?
- Insert Cost?
- Delete Cost?

Which Index/Layout?

```
select
        sum(l_extendedprice*l_discount) as revenue
from
        lineitem
where
        l_shipdate >= date '[DATE]'
        and l_shipdate < date '[DATE]' + interval '1' year
        and l_discount between [DISCOUNT] - 0.01 and [DISCOUNT] + 0.01
        and l_quantity < [QUANTITY];
```

What features are interesting?

Which Index/Layout?

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

What features are interesting?

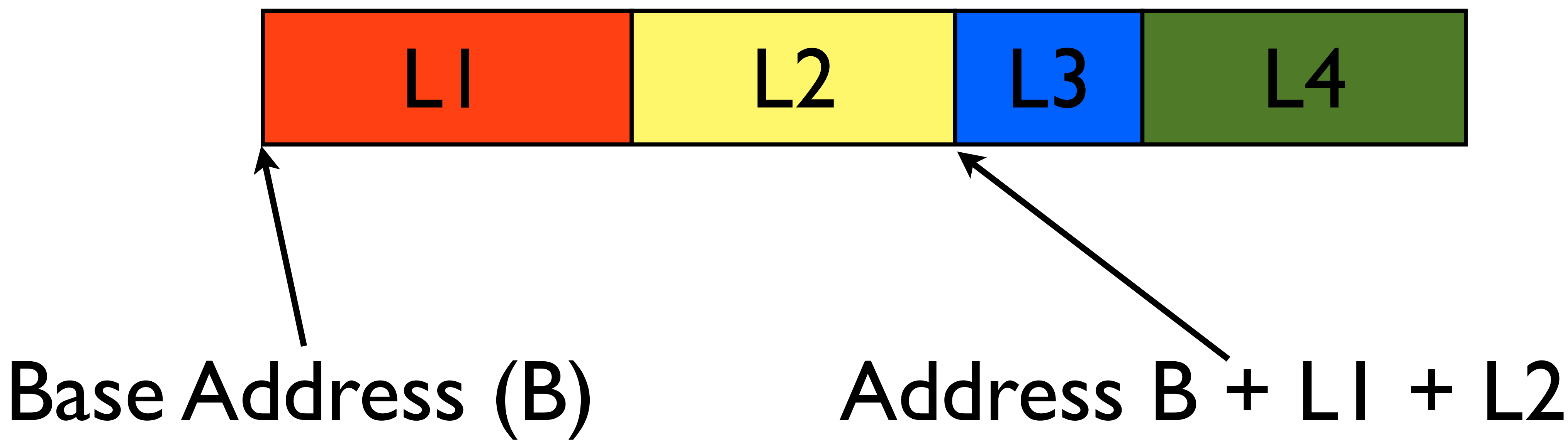
Physical Layouts

Data Organization

- How do we store data?
  - How are records represented on-disk? (Serialization)
  - How are records stored within a page?
  - How are pages organized in a file?
  - What other metadata do we need?
- Our solutions must also be persisted to disk.

Record (Tuple) Formats

- Fixed Length Records

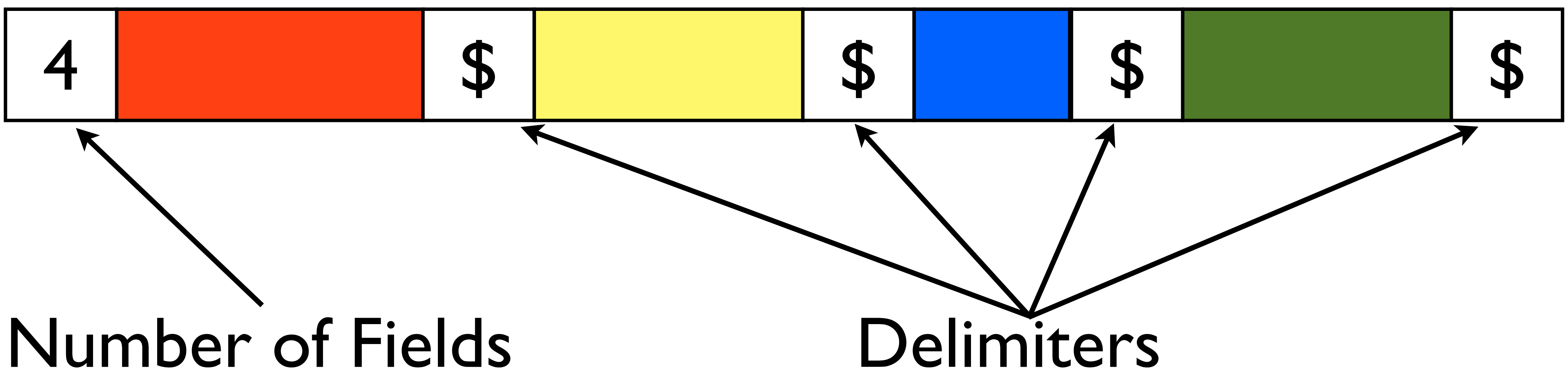


Record information stored in **System Catalog**

What are some advantages/disadvantages of storing records this way?

Record (Tuple) Formats

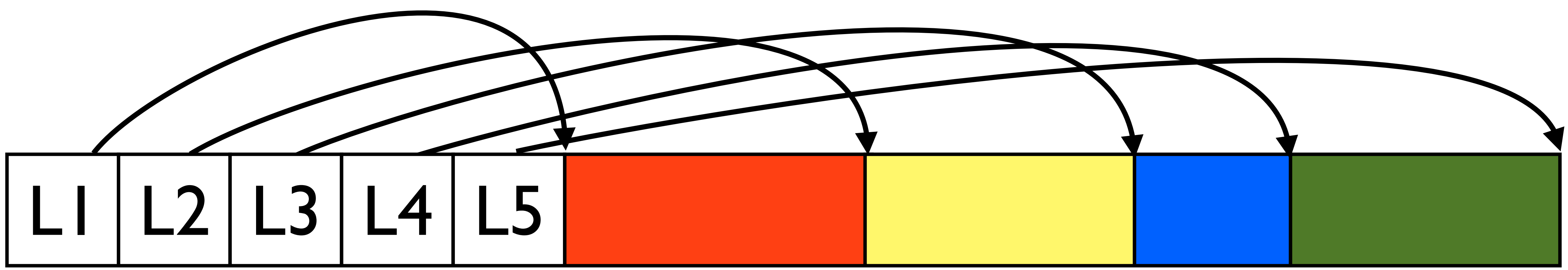
- Delimited Records



What are some advantages/disadvantages of storing records this way?

Record (Tuple) Formats

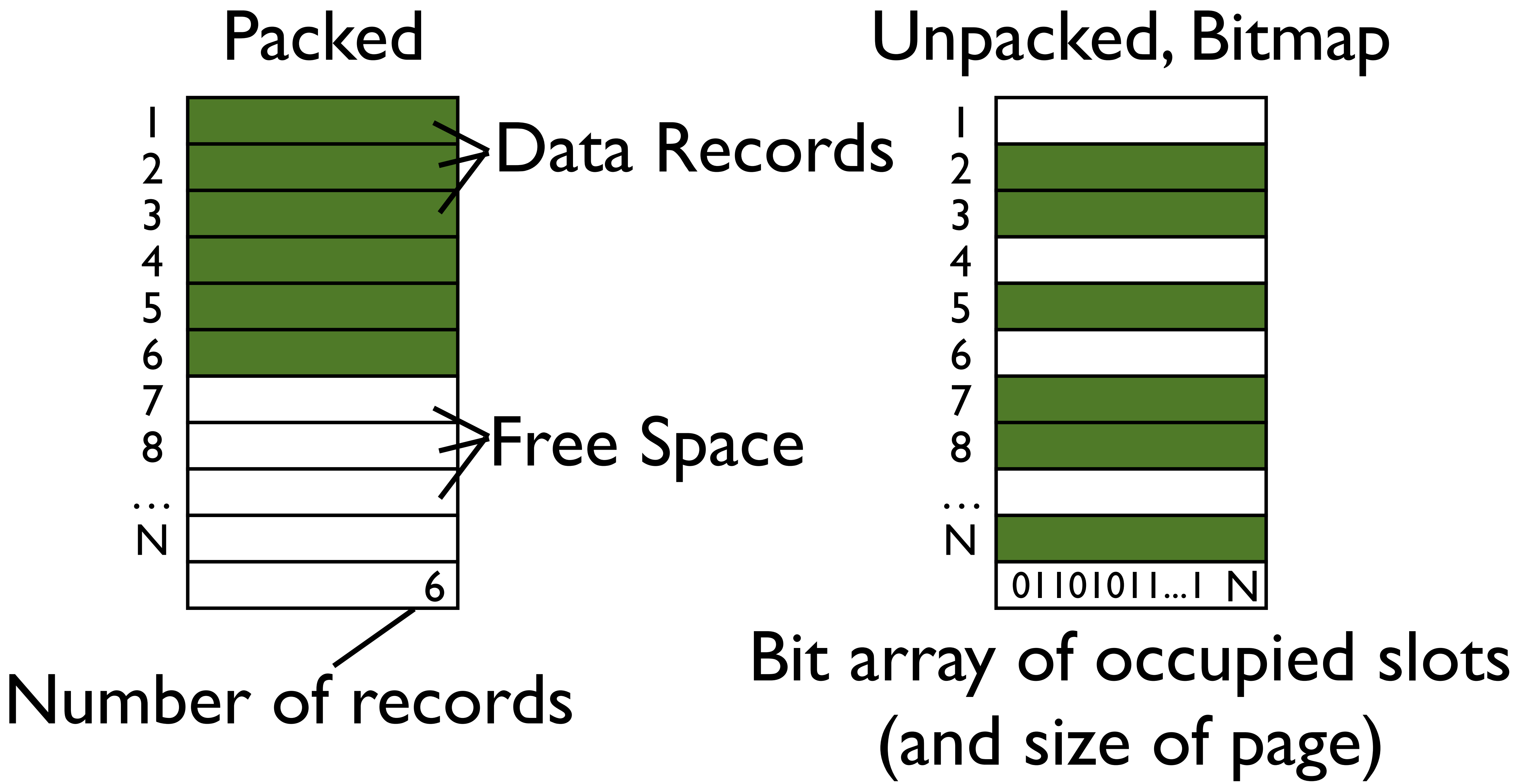
- Self-Describing Records



Array of Field Offsets

What are some advantages/disadvantages of storing records this way?

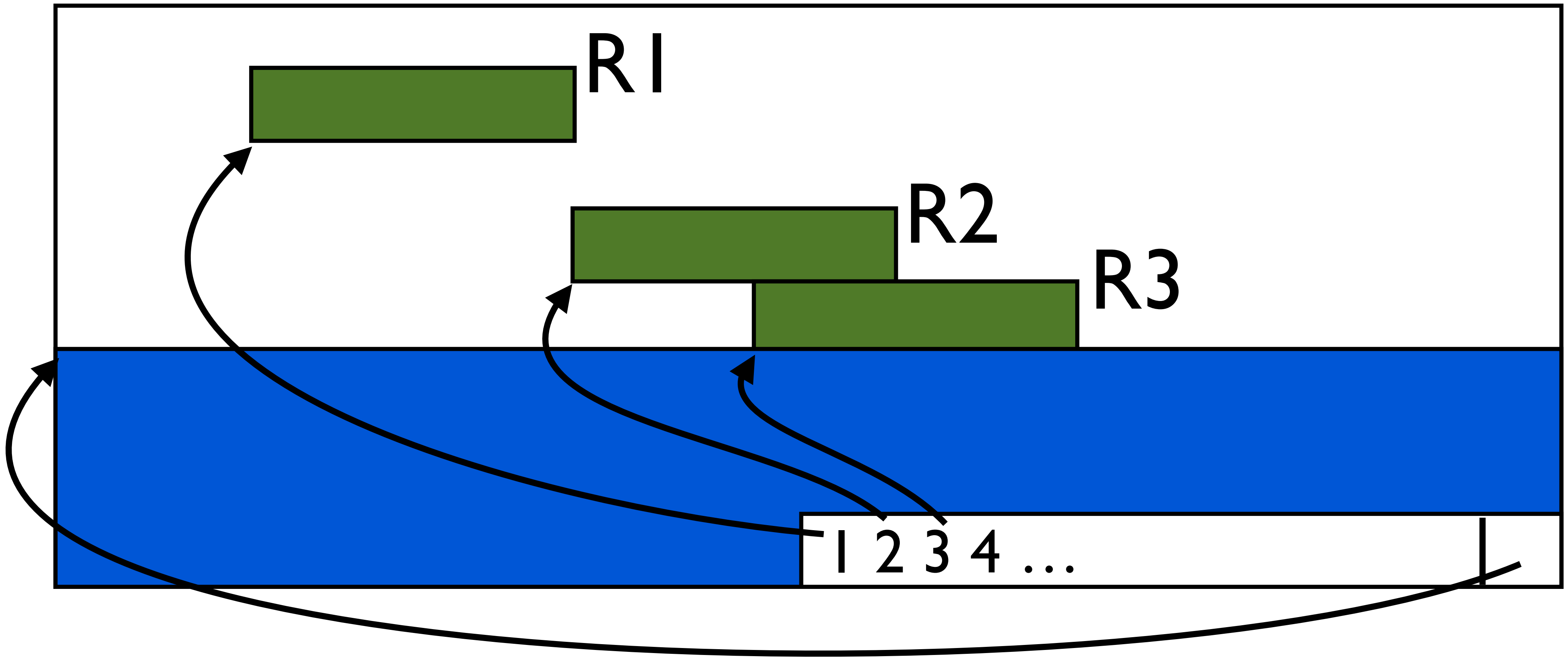
Page Formats



What are advantages/disadvantages of these formats?

Page Formats

Variable Size Records



What are advantages/disadvantages of this format?

Files of Records

IO is done at the Page/Block level

… but queries are done at the Record level

File: A collection of pages of records that must support:

Insert/Delete/Update a record
Read a record (using record ID)
Scan all records (possibly with some condition)

Unordered (Heap) Files

Store records in no particular order

Disk pages are allocated/freed as file grows and shrinks

Support for record level operations by:

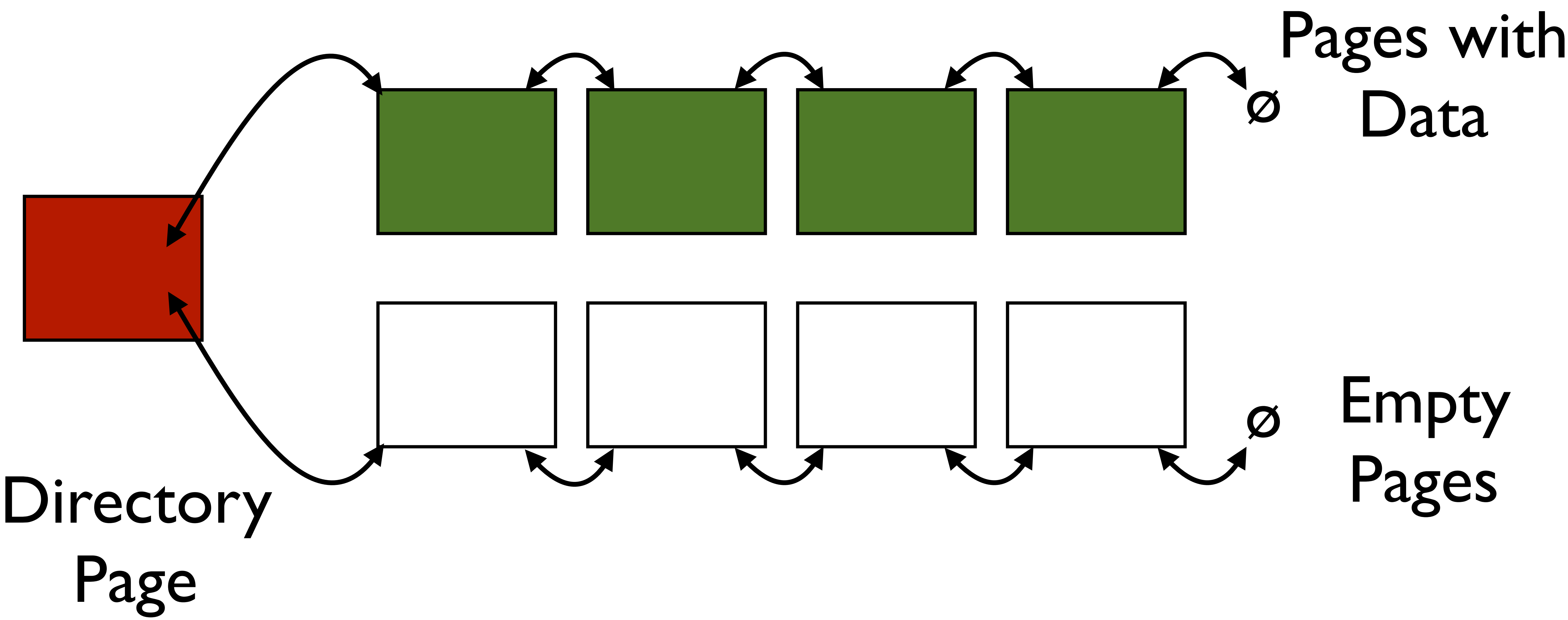
Keeping track of pages in the file

Keeping track of free space in each page

Keeping track of records on each page

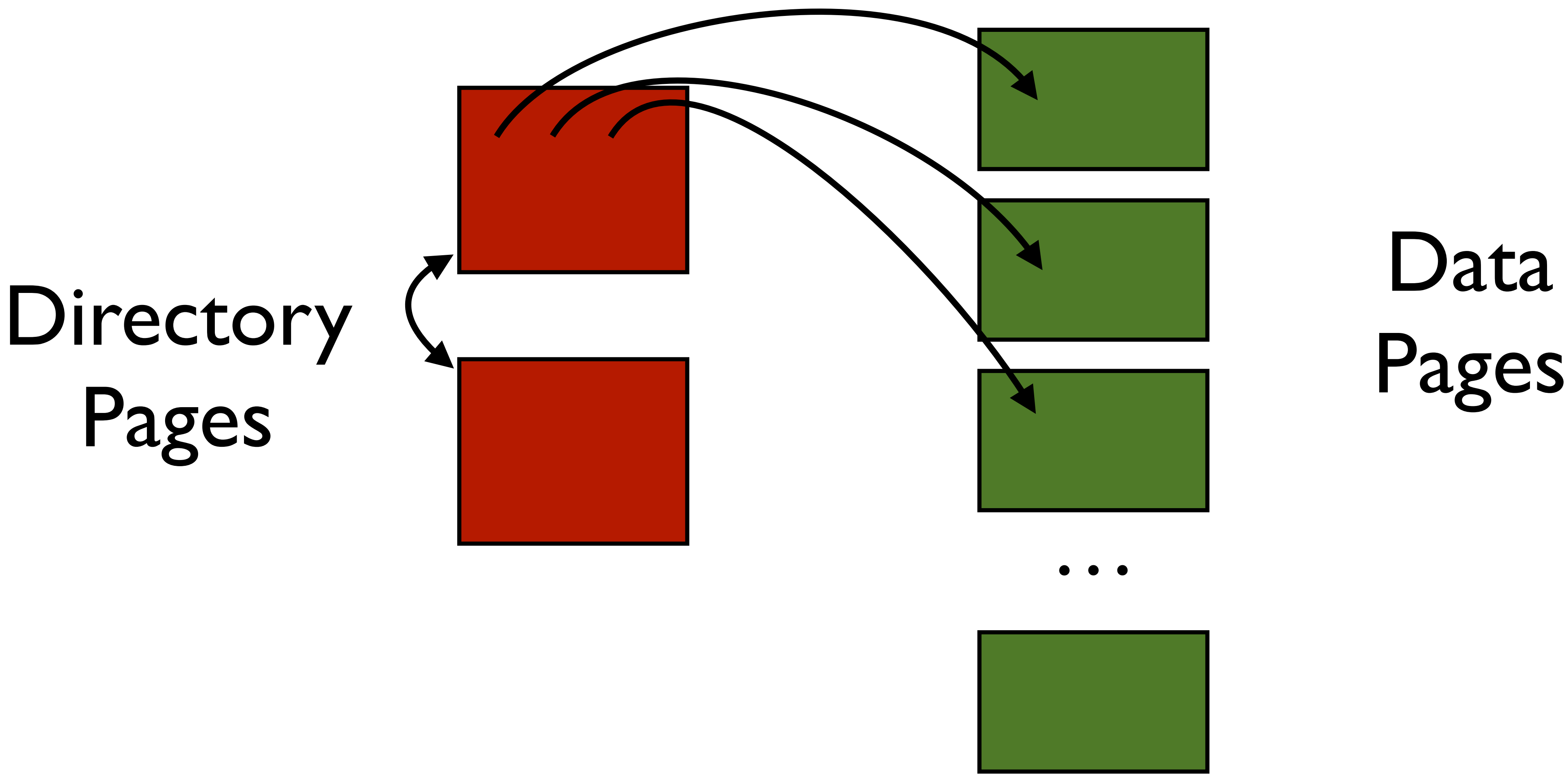
This data must be stored somewhere!

Unordered (Heap) Files

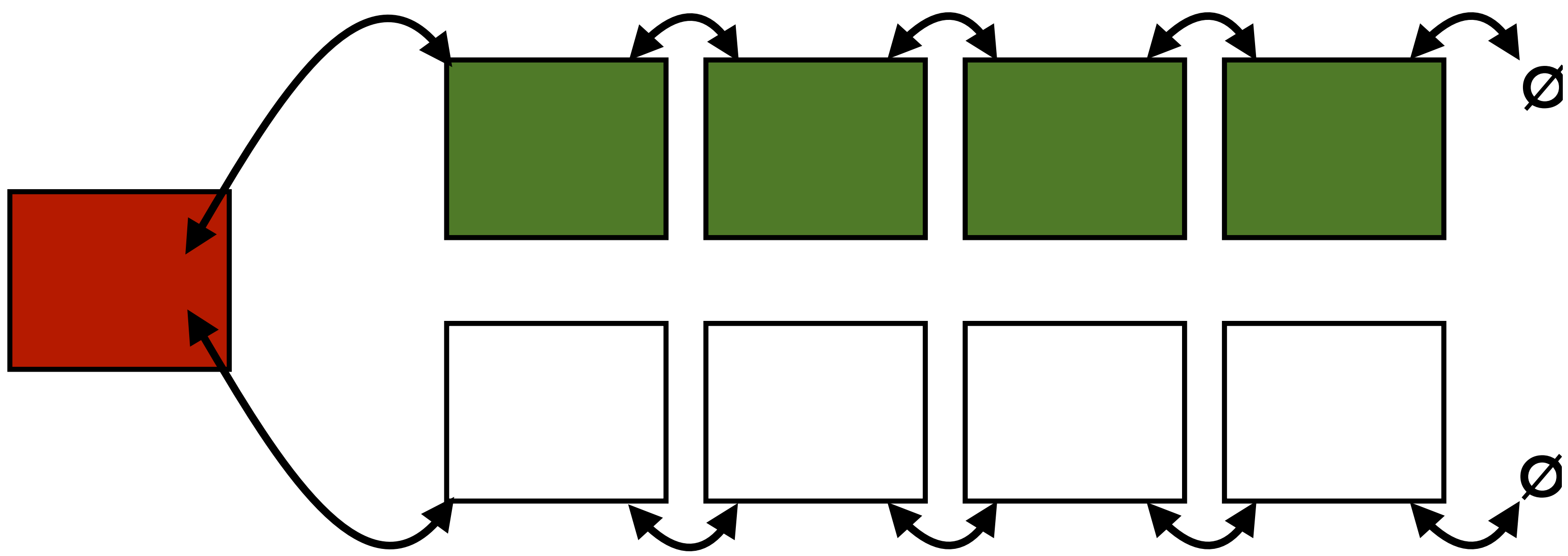


Each page contains 2 pointers plus data

Unordered (Heap) Files



Directories are a collection of pages (e.g., a linked list)
Directories point to all data pages
(entries can include # of free pages)



What are the advantages and disadvantages of each?