CS 186 Discussion 5

Joins and Relational Algebra

Logistics

- Homework 3
  - Notebook I due this Thursday
    - If stuck on join: review 2/23 lecture
  - Notebook II due next Friday
- Midterm
  - Midterm 1 next Tuesday
  - Review session this Saturday

Vitamin 3 - Q3

Assume that each page in our system can hold 64 KB (1 KB = 1024 bytes), integers are 32-bits wide, and bytes are 8-bits wide.

We add two variable-length fields to our table schema. Now our table looks like this:

```
CREATE TABLE Submissions (
  record_id integer UNIQUE,
  assignment_id integer,
  student_id integer,
  time_submitted integer,
  grade_received byte,

  comment text,
  regrade_request text,

  PRIMARY KEY(assignment_id, student_id)
);
```

We decide to use slotted pages to store the variable length records. Each page begins with a 24-byte header plus a slot directory. (Assume this header contains information such as the number of valid records in the page.) Each pointer inside the slot directory consumes 20 bits/record, while the record header storing field offsets is 32 bits wide.

Q3: What is the maximum number of records that can fit in our slotted pages?

Vitamin 3 - Q7

Suppose we have an alternative 2 unclustered index on (assignment_id, student_id) with a depth of 3 (one must traverse 3 index pages to reach any leaf page). Here's the schema:

```
CREATE TABLE Submissions (
  record_id integer UNIQUE,
  assignment_id integer,
  student_id integer,
  time_submitted integer,
  grade_received byte,

  comment text,
  regrade_request text,

  PRIMARY KEY(assignment_id, student_id)
);

CREATE INDEX SubmissionLookupIndex
ON Submissions (assignment_id, student_id);
```

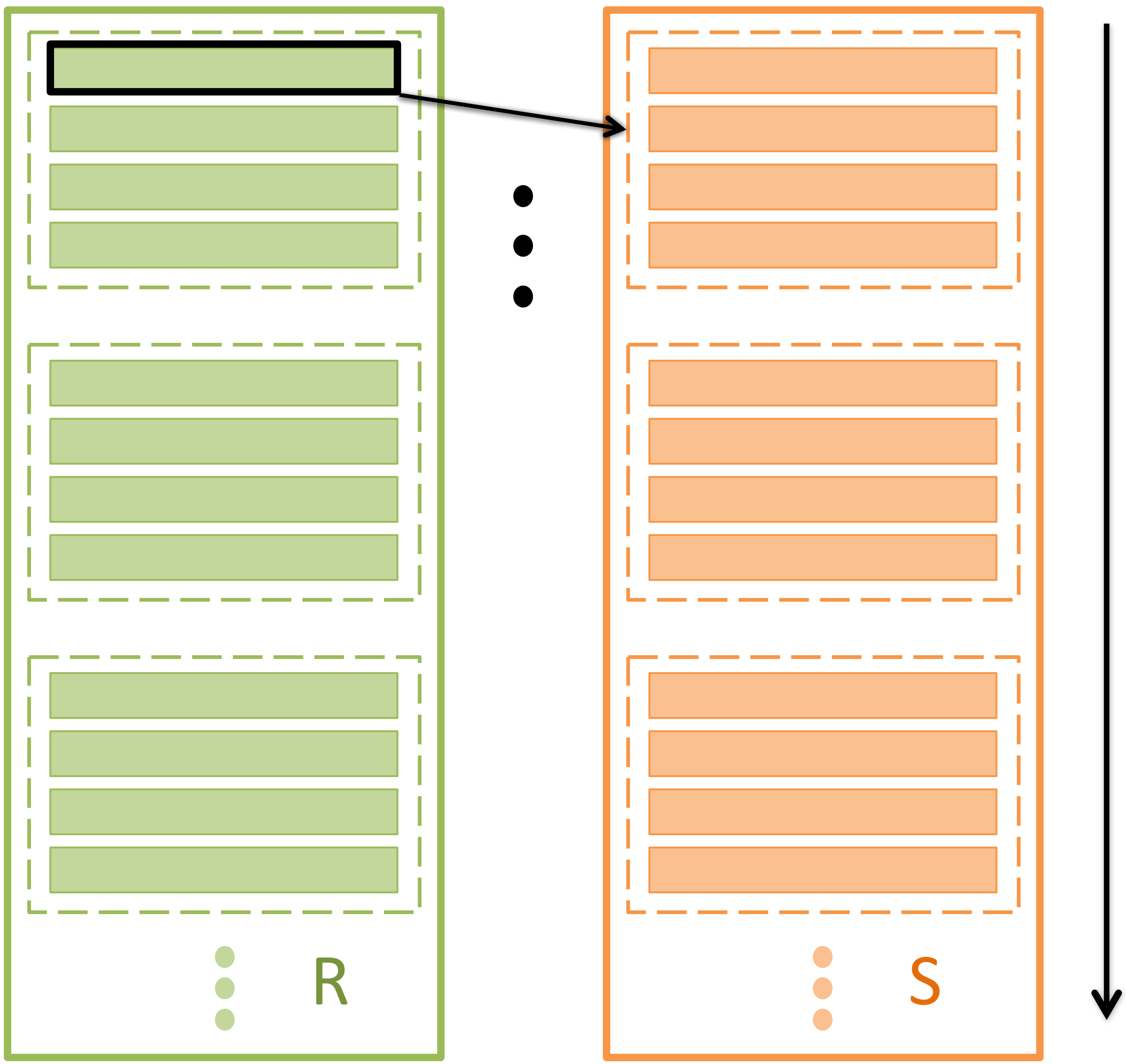
Assume the table takes up 12 MB on disk (1 MB = 1024 KB). (This includes extra space allocated for future insertions.)

Q7: In the best case, how many I/Os does it take to perform an equality search on grade_received?

Cost Notation

- $[R]$ = number of pages in Table R
- p_R = number of records per page of R
- $|R|$ = number of records in R
  - » (cardinality)
- Note: $|R| = p_R * [R]$

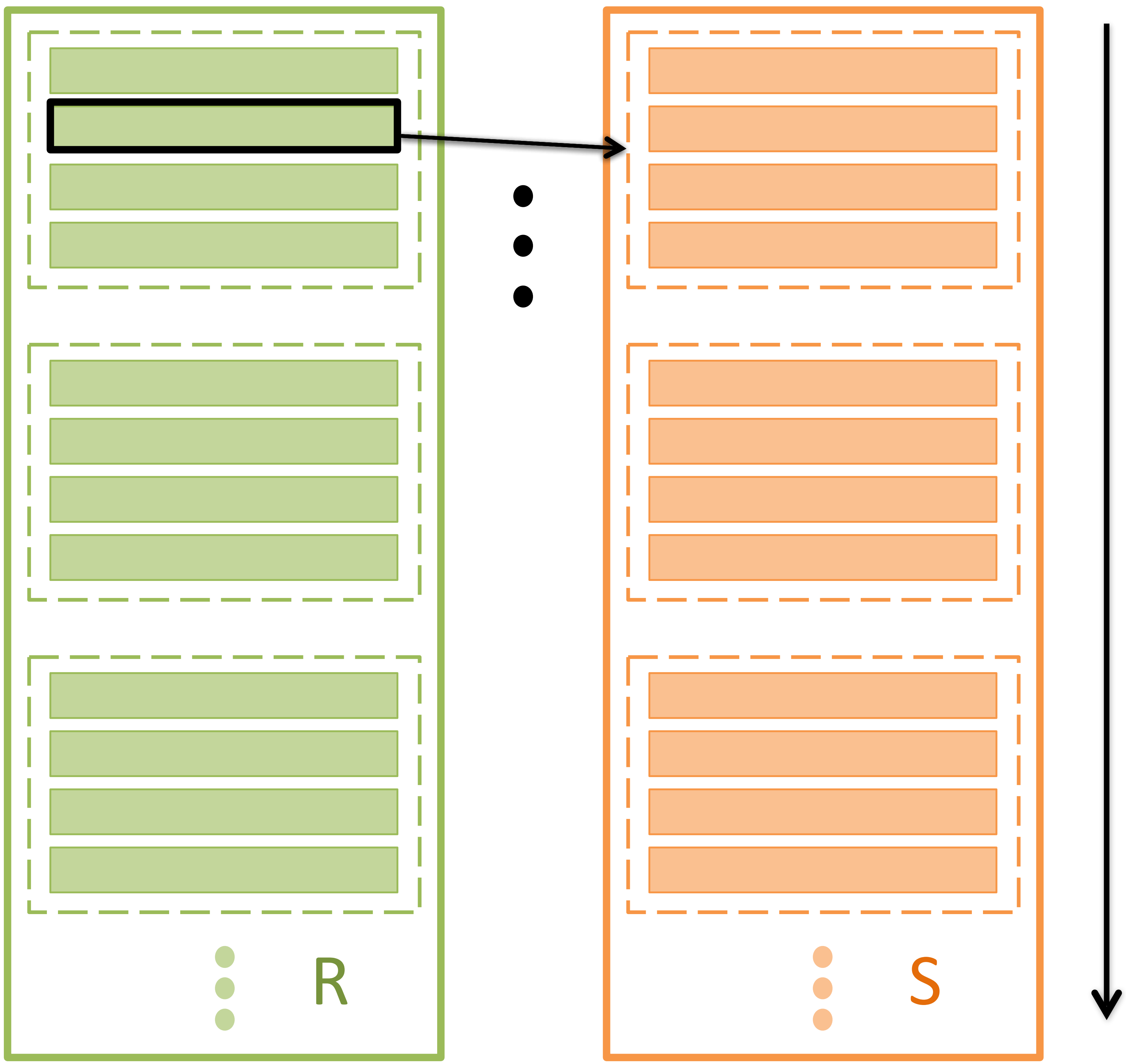
Simple Nested Loop Join



First iteration of outer loop…

Compare with all tuples in S
…and add matches to result

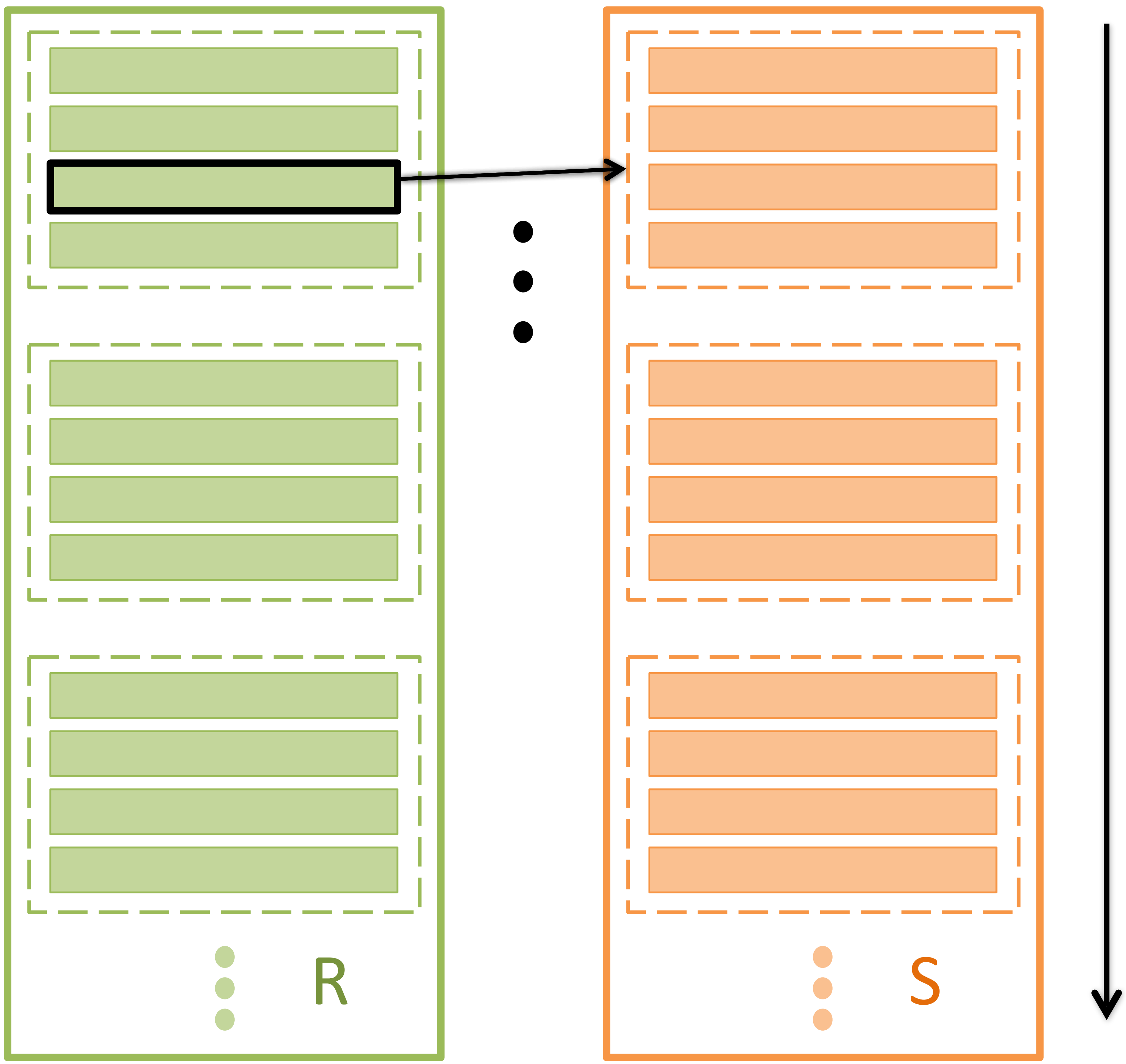
Simple Nested Loop Join



Second iteration of outer loop…

Compare with all tuples in S
…and add matches to result

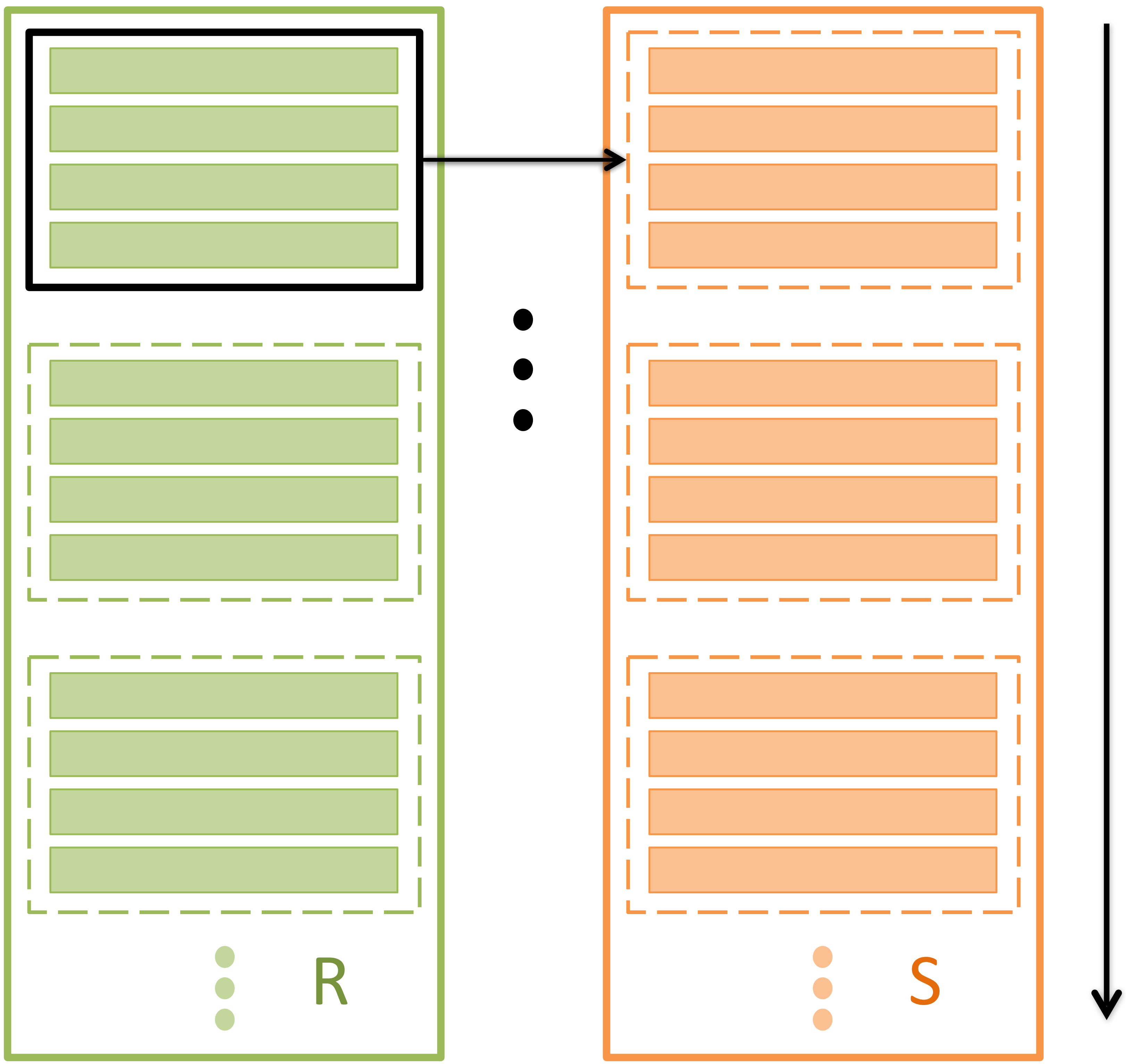
Simple Nested Loop Join



Cost =

$[R] + p_R[R][S]$

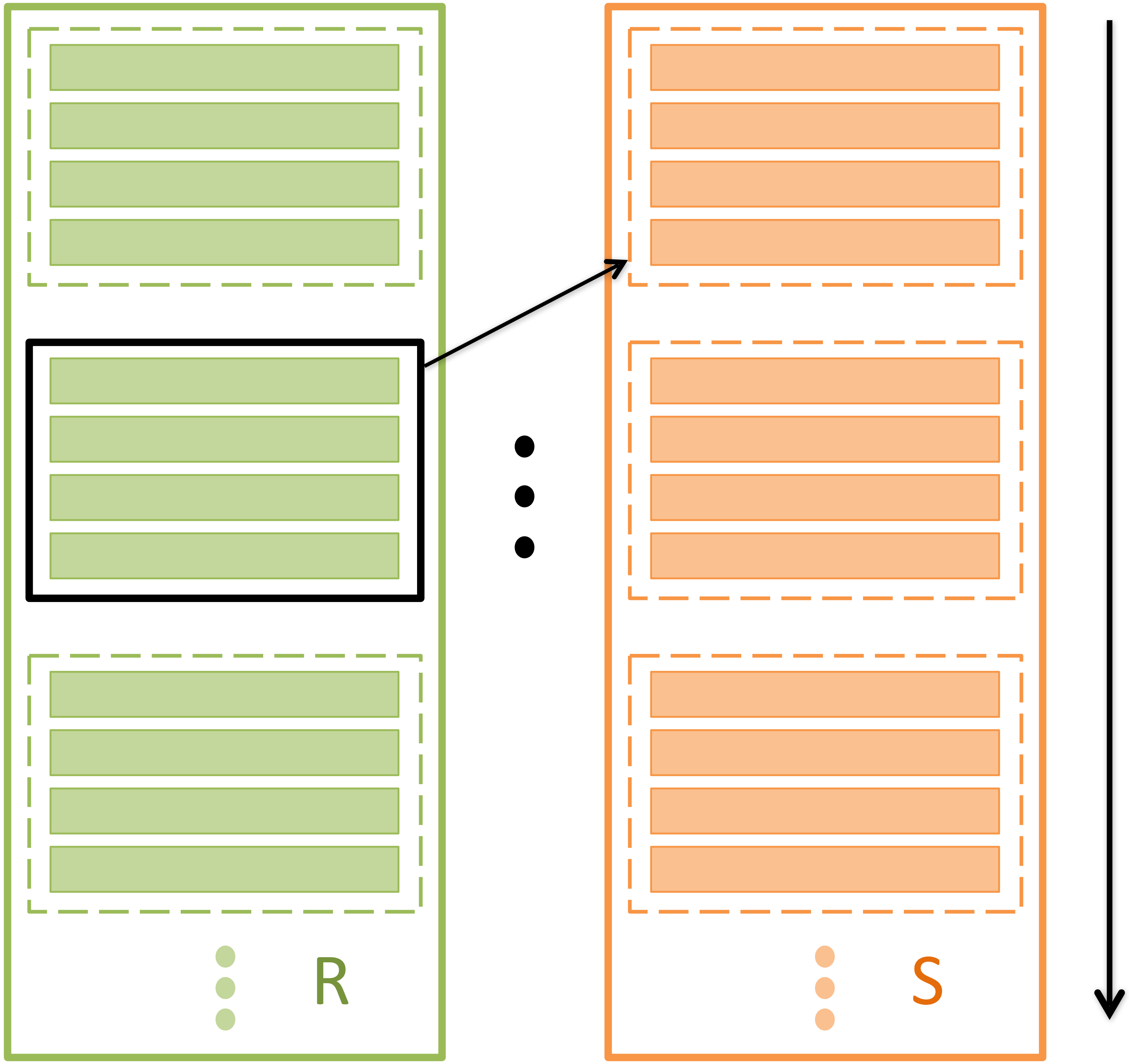
Page-Oriented Nested Loop Join



First iteration of outer loop…

Compare with all tuples in S
…and add matches to result

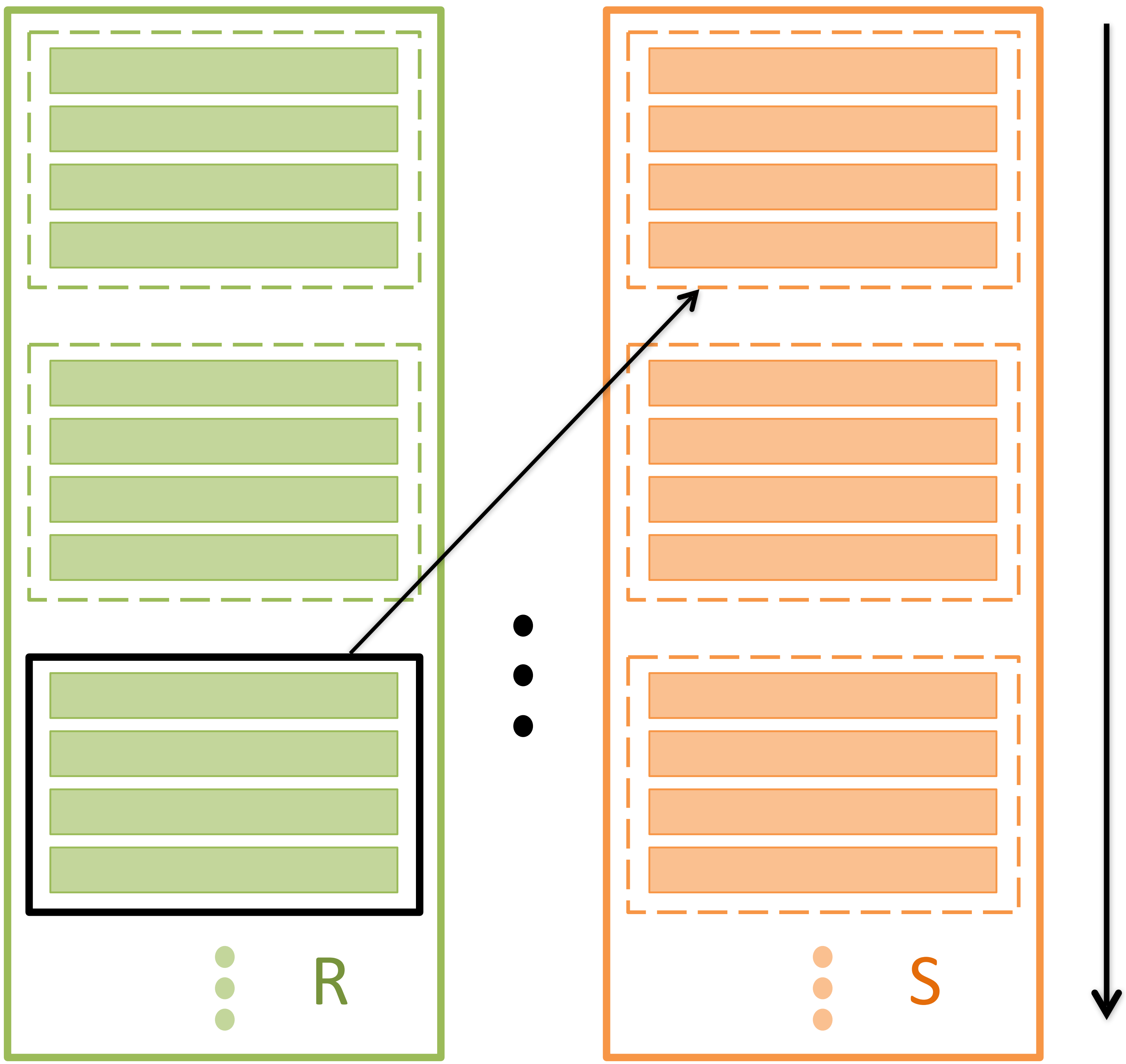
Page-Oriented Nested Loop Join



Second iteration of outer loop…

Compare with all tuples in S
…and add matches to result

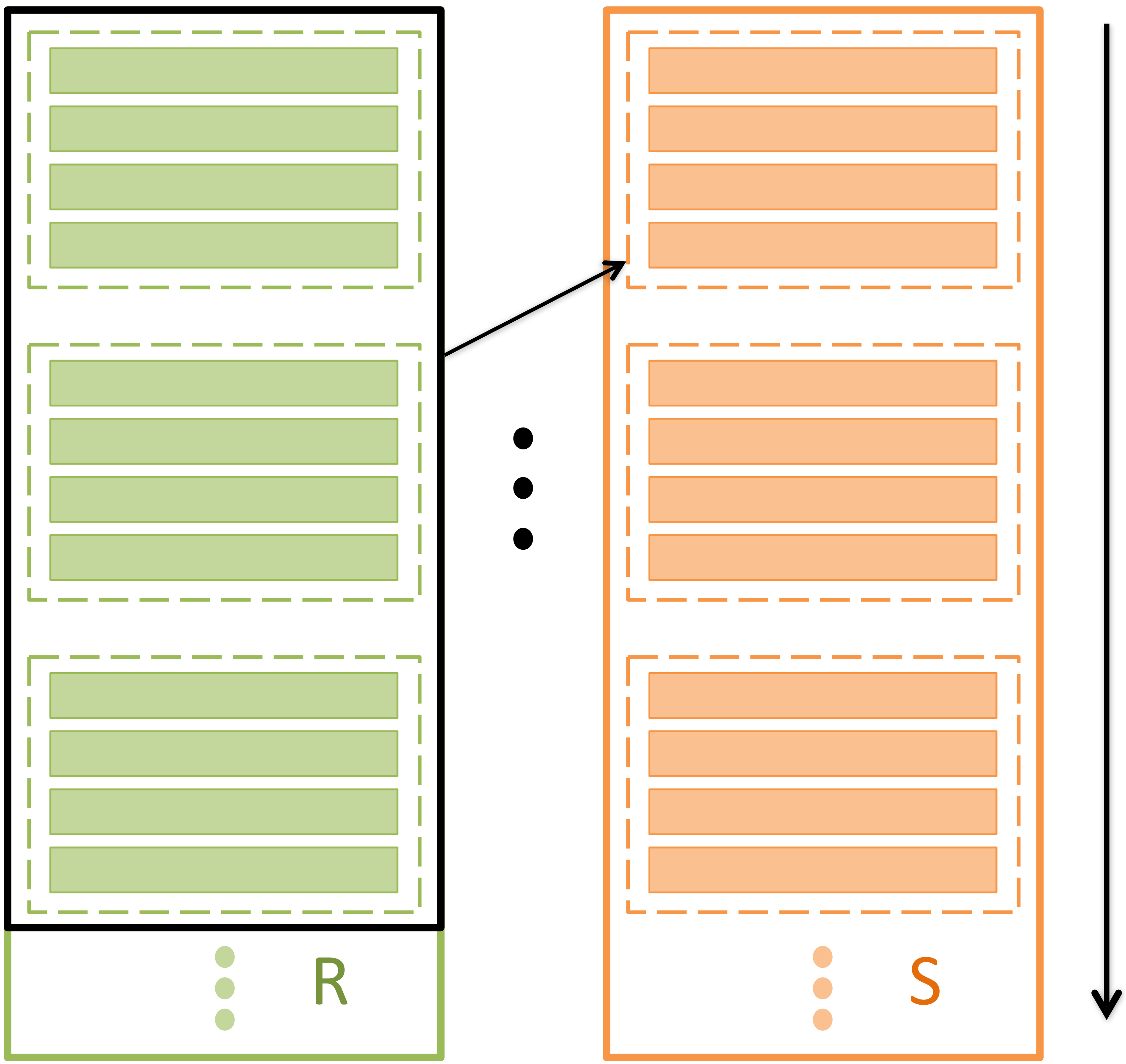
Page-Oriented Nested Loop Join



Cost =

$[R] + [R][S]$

Block Nested Loop Join



First iteration of outer loop…

B pages in memory!

Use B – 2 for R

Compare with all tuples in S

…and add matches to result

Cost of BNLJ?

$$[R] + (\#\text{ blocks in } R) * [S]$$

$$= [R] + \lceil [R] / _{chunksize} \rceil * [S]$$

$$\boxed{= [R] + \lceil [R] / _{(B-2)} \rceil [S]}$$

Sort-Merge Join

1. Sort R and S using external sorting:

   4[R] + 4[S]   (2 passes)

2. Scan sorted R and sorted S "in tandem" and output matches:

   [R] + [S]

Does this include final write costs?

Optimized Sort-Merge Join

1. Sort R and S using external sorting, but stop before the final pass:

   2[R] + 2[S]

2. Join on the final merge pass!

   [R] + [S] + [output]

Is [R] + [S] an upper bound on join cost?

Worksheet - 2c

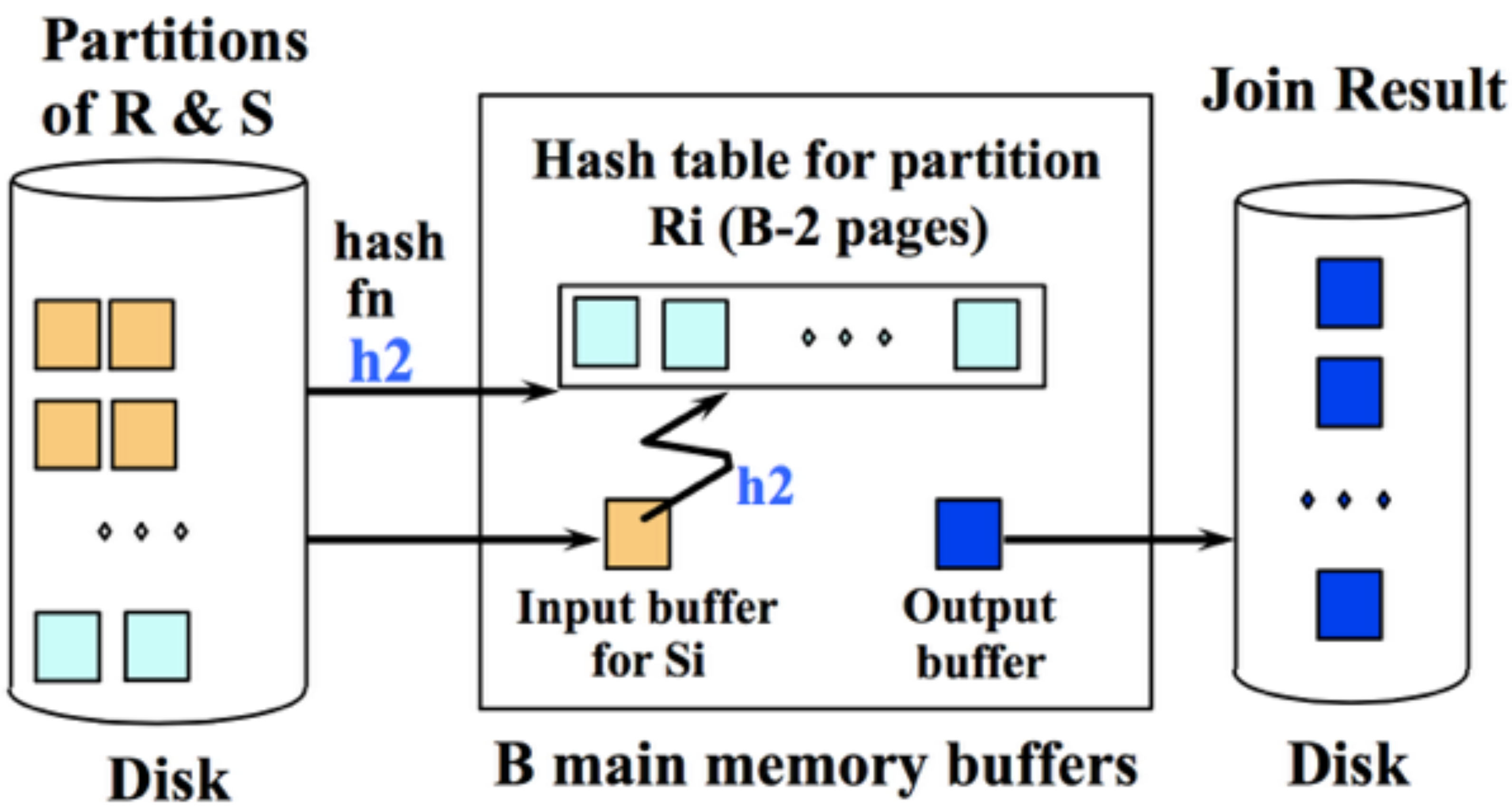
- Can we optimize?
  - B = 15, [R] = 100, [S] = 50

Worksheet - 2c

- Can we optimize?
  - B = 15, [R] = 100, [S] = 50
- After Pass 0:
  - R: $\lceil 100 / 15 \rceil$ = 7 runs
  - S: $\lceil 50 / 15 \rceil$ 4 runs
- Can we join now?

Hash Join

- Partition both tables! => 2[R] + 2[S]
- Build hash tables for R
- Then match ("probe") => [R] + [S]:



- Even better with hybrid hashing!

Join Costs Overview

- Block Nested Loop Join

$$[R] + {}^{[R][S]}/_{(B-2)}$$

- Sort-Merge Join / Hash Join

$$3[R] + 3[S]$$

- Index Nested Loop Join

$$[R] + p_R[R](\text{index lookup in } S)$$

Sort-Merge vs. Hash Join

Block Nested Loop Join

- Works for cross (Cartesian) products
- Works for non-equality predicates
- Scales nicely with buffer size!

Sort-Merge Join

- Good with sorted input/output
- Handles data skew + bad hashing (large partitions)
- Good with limited memory

Hash Join

- Good with hashed input/output
- # passes bounded by smaller relation! *Why?*
- Hybrid hashing

Relational Algebra

- Represent query execution plan with operators
  - More on query optimization soon

Basic Operators

- SELECTION (σ)
- PROJECTION (π)
- CROSS-PRODUCT (x)
- SET-DIFFERENCE (–)
- UNION (U)
- RENAME (ρ)

Basic Operators

- SELECTION (σ) — filter rows
- PROJECTION (π) — filter columns
- CROSS-PRODUCT (x)
- SET-DIFFERENCE (–)
- UNION (U)
- RENAME (ρ)

Compound Operators

- INTERSECTION ($\cap$)
- JOINs
  - NATURAL JOIN ($\bowtie$)
  - THETA JOIN ($\bowtie_\theta$)
    - θ: R.sid = S.sid

Relational Algebra Worksheet

```
Songs(song_id, song_name, album_id, weeks_in_top_40)

Artists(artist_id, artist_name, first_year_active)

Albums(album_id, album_name, artist_id, year_released, genre)
```

- 1. Find the name of the artists who have albums with a genre of either ‘pop’ or ‘rock’.

```
SELECT artist_name
FROM Artists, Albums
WHERE Artists.artist_id = Albums.artist_id
AND (genre = ‘pop’ OR genre = ‘rock’);
```

Relational Algebra Worksheet

```
Songs(song_id, song_name, album_id, weeks_in_top_40)

Artists(artist_id, artist_name, first_year_active)

Albums(album_id, album_name, artist_id, year_released, genre)
```

- 1. Find the name of the artists who have albums with a genre of either ‘pop’ or ‘rock’.

```
SELECT artist_name
FROM Artists, Albums
WHERE Artists.artist_id = Albums.artist_id
AND (genre = ‘pop’ OR genre = ‘rock’);
```

$$\pi_{artist\_name}((\sigma_{genre = \text{‘pop’} \vee genre = \text{‘rock’}} \text{Albums}) \bowtie \text{Artists})$$

Relational Algebra Worksheet

```
Songs(song_id, song_name, album_id, weeks_in_top_40)

Artists(artist_id, artist_name, first_year_active)

Albums(album_id, album_name, artist_id, year_released, genre)
```

- 2. Find the name of the artists who have albums of genres ‘pop’ AND ‘rock’.

Relational Algebra Worksheet

`Songs(song_id, song_name, album_id, weeks_in_top_40)`

`Artists(artist_id, artist_name, first_year_active)`

`Albums(album_id, album_name, artist_id, year_released, genre)`

- 2. Find the name of the artists who have albums of genres ‘pop’ AND ‘rock’.

$$\pi_{\text{artist\_name}}((\sigma_{\text{genre = ‘pop’}} \text{Albums}) \bowtie \text{Artists})$$
$$\cap$$
$$\pi_{\text{artist\_name}}((\sigma_{\text{genre = ‘rock’}} \text{Albums}) \bowtie \text{Artists})$$

Relational Algebra Worksheet

`Songs(song_id, song_name, album_id, weeks_in_top_40)`

`Artists(artist_id, artist_name, first_year_active)`

`Albums(album_id, album_name, artist_id, year_released, genre)`

- 3. Find the id of artists who have albums of genre ‘pop’ or have spent over 10 weeks in the top 40.

Relational Algebra Worksheet

```
Songs(song_id, song_name, album_id, weeks_in_top_40)

Artists(artist_id, artist_name, first_year_active)

Albums(album_id, album_name, artist_id, year_released, genre)
```

- 3. Find the id of artists who have albums of genre ‘pop’ or have spent over 10 weeks in the top 40.

$\pi_{artist\_id}((\sigma_{genre = 'pop'} Albums) \bowtie Artists)$

$\cup$

$\pi_{artist\_id}((\sigma_{weeks\_in\_top\_40 > 10} Songs) \bowtie Albums)$

Relational Algebra Worksheet

`Songs(song_id, song_name, album_id, weeks_in_top_40)`

`Artists(artist_id, artist_name, first_year_active)`

`Albums(album_id, album_name, artist_id, year_released, genre)`

- 4. Find the names of all artists who do not have any albums.

```
SELECT artist_name
FROM Artists
WHERE artist_id NOT IN
(SELECT artist_id
FROM Albums)
```

Relational Algebra Worksheet

`Songs(song_id, song_name, album_id, weeks_in_top_40)`

`Artists(artist_id, artist_name, first_year_active)`

`Albums(album_id, album_name, artist_id, year_released, genre)`

- 4. Find the names of all artists who do not have any albums.

$$\pi_{\text{artist\_name}}\,(\text{Artists} \bowtie (\pi_{\text{artist\_id}}\,\text{Artists} - \pi_{\text{artist\_id}}\,\text{Albums}))$$