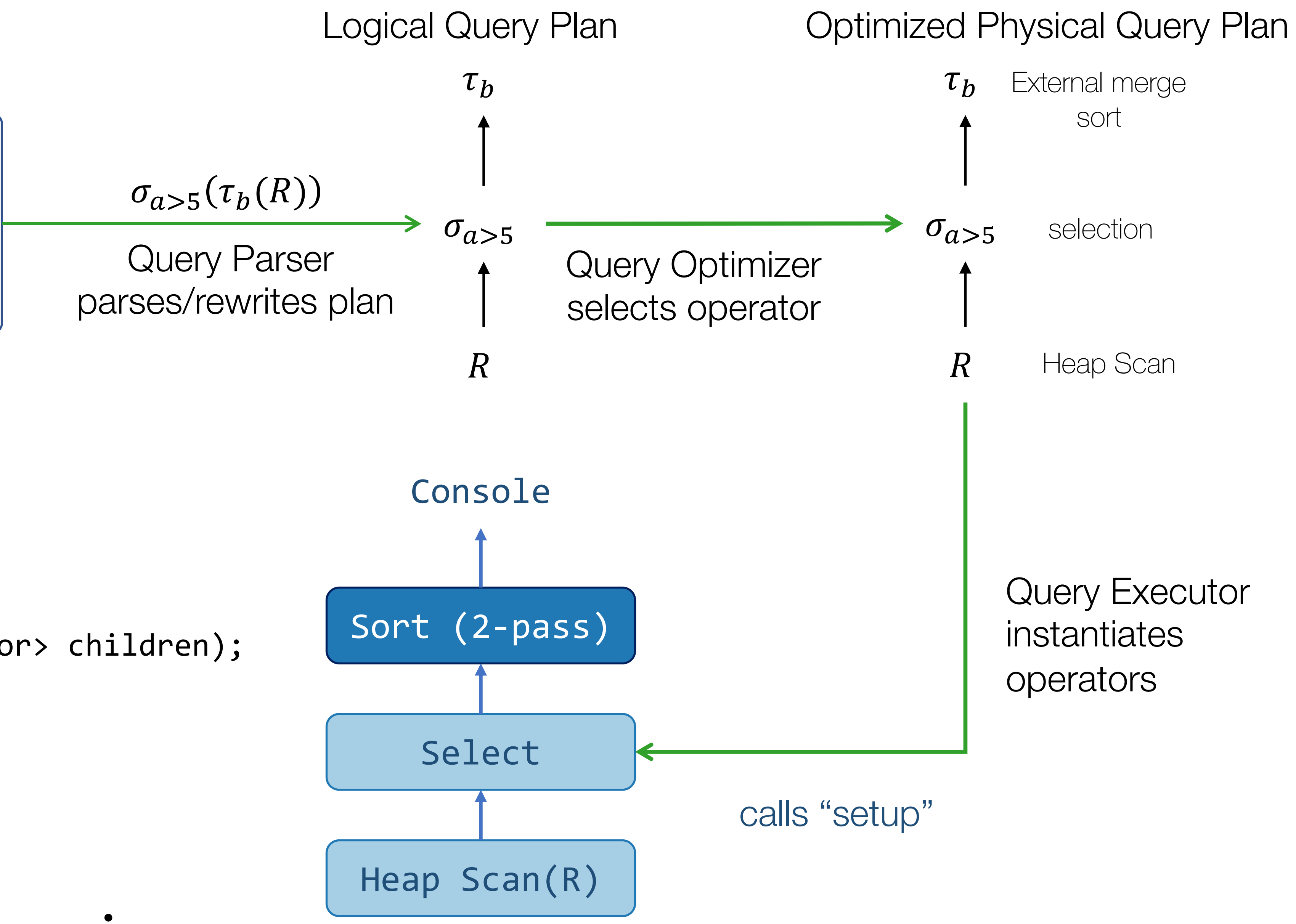
Query Processing

SQL Query

```sql
select * from R
where a > 5
order by b;
```



Each operator is a subclass of an *iterator*

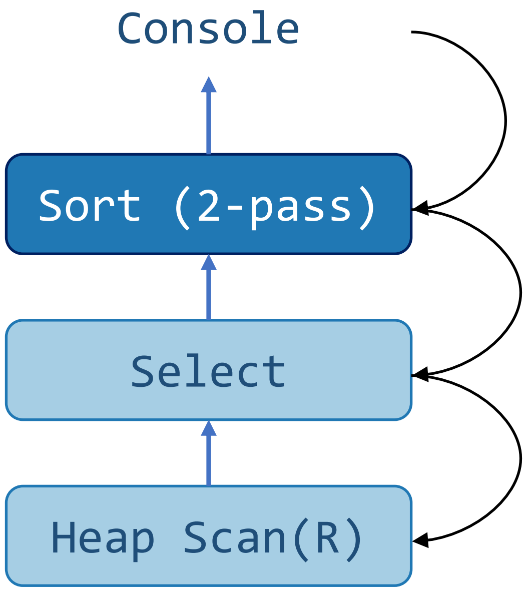
```
abstract class iterator
  void setup(List<Iterator> children);
  void init(args);
  tuple next();
  void close();
}
```

Query Processing

Query Processing – Iterator Model

Each operator is a subclass of an *iterator*

```
abstract class iterator
  void setup(List<Iterator> children);
  void init(args);
  tuple next();
  void close();
}
```




```
for each child in children:
  child.init()
```

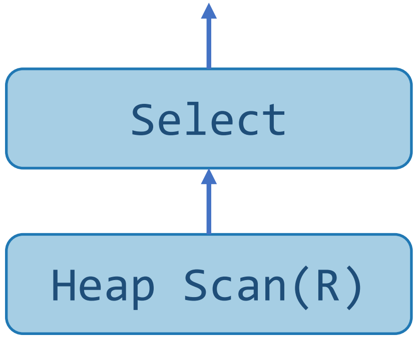
`child.init()`

`child.init()`

`child.next()` Blocking

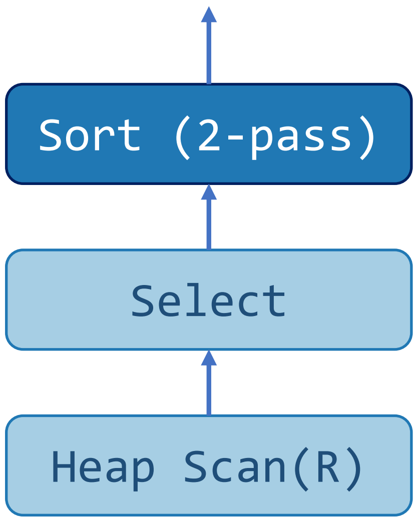
`child.next()` Streaming

`child.next()` Streaming




```
new(function predicate): //constructor given predicate
    p = predicate; //{return tuple.getValue("a")>5}
init():
    child.init(); //Initializes the heap scan op
    current = null;
next():
    while (current != EOF && !p(current)){
        current = child.next();
    }
    return current;
```

```
init(): //the op was setup to access the heap file
    current_page = file.getPage(0);
    current_slot = current_page.getSlot(0);
next():
    if(current_page == null) return EOF;
    current = current_slot.getTuple();
    current_slot.next();
    if(current_slot == null):
        current_page.next();
        if(current_page != null):
            current_slot = current_page.getSlot(0);
    return current;
```




```
init():
    child.init();
    repeatedly call child.next() to generate sorted
          runs on disk until EOF

    open each sorted run / load into input buffer

next():
    output = min tuple across all input buffers (remove
        min tuple)

    if no tuples remain:
        return EOF;

    if min tuple was last one in its buffer:
        fetch next page from that run into buffer;

    return output;
```

The Join Operator

What is a Join?

$$R \bowtie_{\sigma} S = \sigma(R \times S)$$

Cartesian Product ×

$R \times S$

Each row in R is paired with each row in S to produce nm rows.

Loans (R has n rows)

sid	eid	date	duration	…
72	981	3/8/20	2 weeks	
76	786	3/18/21	2 days	

Students (S has m rows)

sid	name	major	…
72	Ibn Sina	Bio	
73	Plato	Phil	
76	Al Khawarizmi	CS	

Loans × Students

sid	eid	date	duration	sid	name	major	…
72	981	3/8/20	2 weeks	72	Ibn Sina	Bio	
72	981	3/8/20	2 weeks	73	Plato	Phil	
72	981	3/8/20	2 weeks	76	Al Khawarizmi	CS	
76	786	3/18/21	2 days	72	Ibn Sina	Bio	
76	786	3/18/21	2 days	73	Plato	Phil	
76	786	3/18/21	2 days	76	Al Khawarizmi	CS	

Join ⋈

$R \bowtie_{R.\mathrm{sid}=S.\mathrm{sid}} S$

Each row in R is matched to a row in S that satisfies the join condition.

Loans (R has n rows)

sid	eid	date	duration	…
72	981	3/8/20	2 weeks	
76	786	3/18/21	2 days	

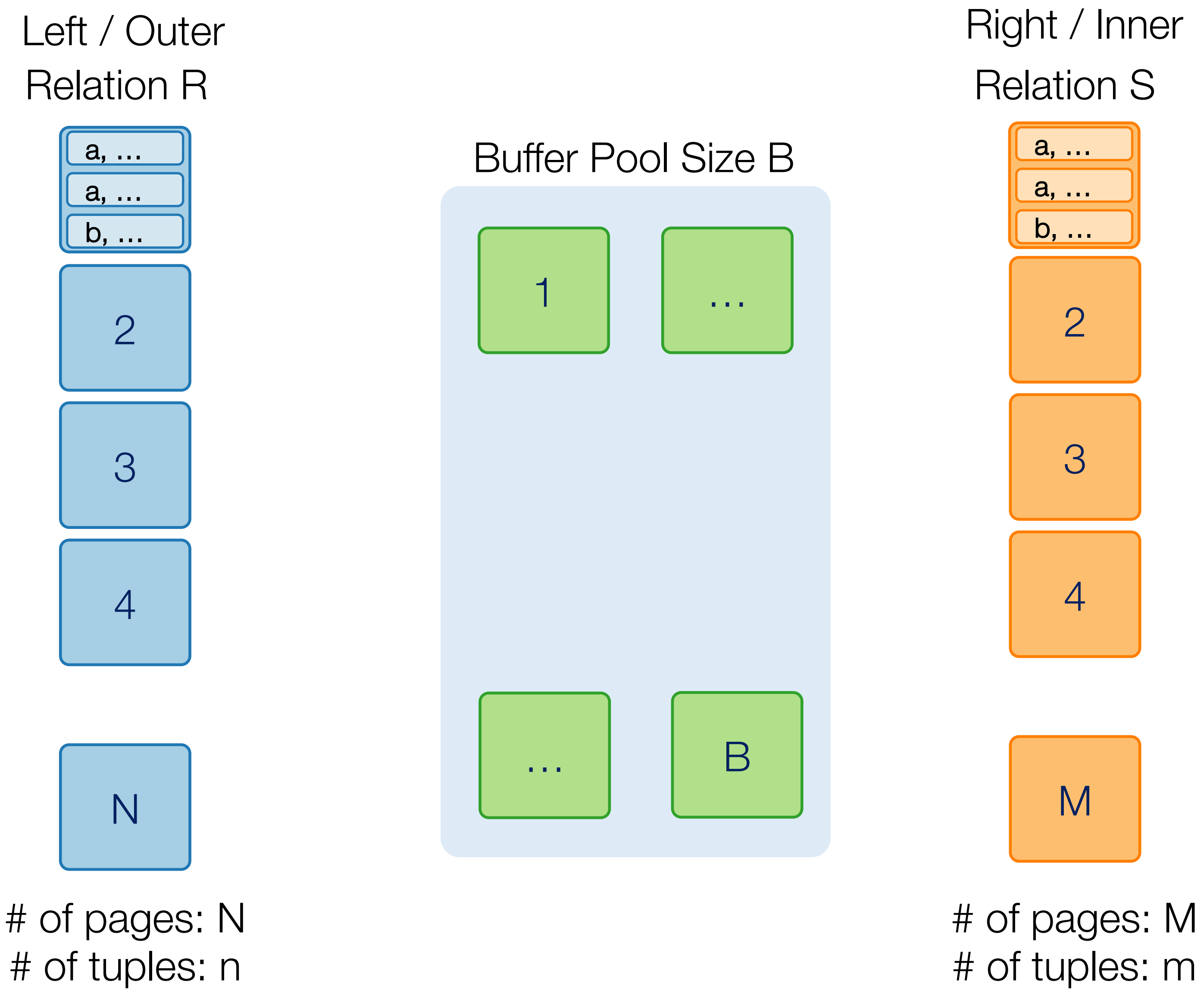
Students (S has m rows)

sid	name	major	…
72	Ibn Sina	Bio	
73	Plato	Phil	
76	Al Khawarizmi	CS	

$$\text{Loans} \bowtie \text{Students} \equiv \sigma_{\mathrm{sid}=\mathrm{sid}}(\text{Loans} \times \text{Students})$$

sid	eid	date	duration	sid	name	major	…
72	981	3/8/20	2 weeks	72	Ibn Sina	Bio	
76	786	3/18/21	2 days	76	Al Khawarizmi	CS	
~~72~~	~~981~~	~~3/8/20~~	~~2 weeks~~	~~76~~	~~Al Khawarizmi~~	~~CS~~	
~~76~~	~~786~~	~~3/18/21~~	~~2 days~~	~~72~~	~~Ibn Sina~~	~~Bio~~	
~~76~~	~~786~~	~~3/18/21~~	~~2 days~~	~~73~~	~~Plato~~	~~Phil~~	
76	786	3/18/21	2 days	76	Al Khawarizmi	CS	

The Join Analysis Set Up



Each table is broken down into pages.

Each table has a join key attribute and a value that could be a record id, or a tuple, etc. We only show the join keys.

When computing join costs, ***we will ignore the output costs:***

(a) For now, we don't know how many tuples will join

(b) Across all implementations, the output cost is the same

The Equipment Loans Application

R – Loans
(*sid*, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

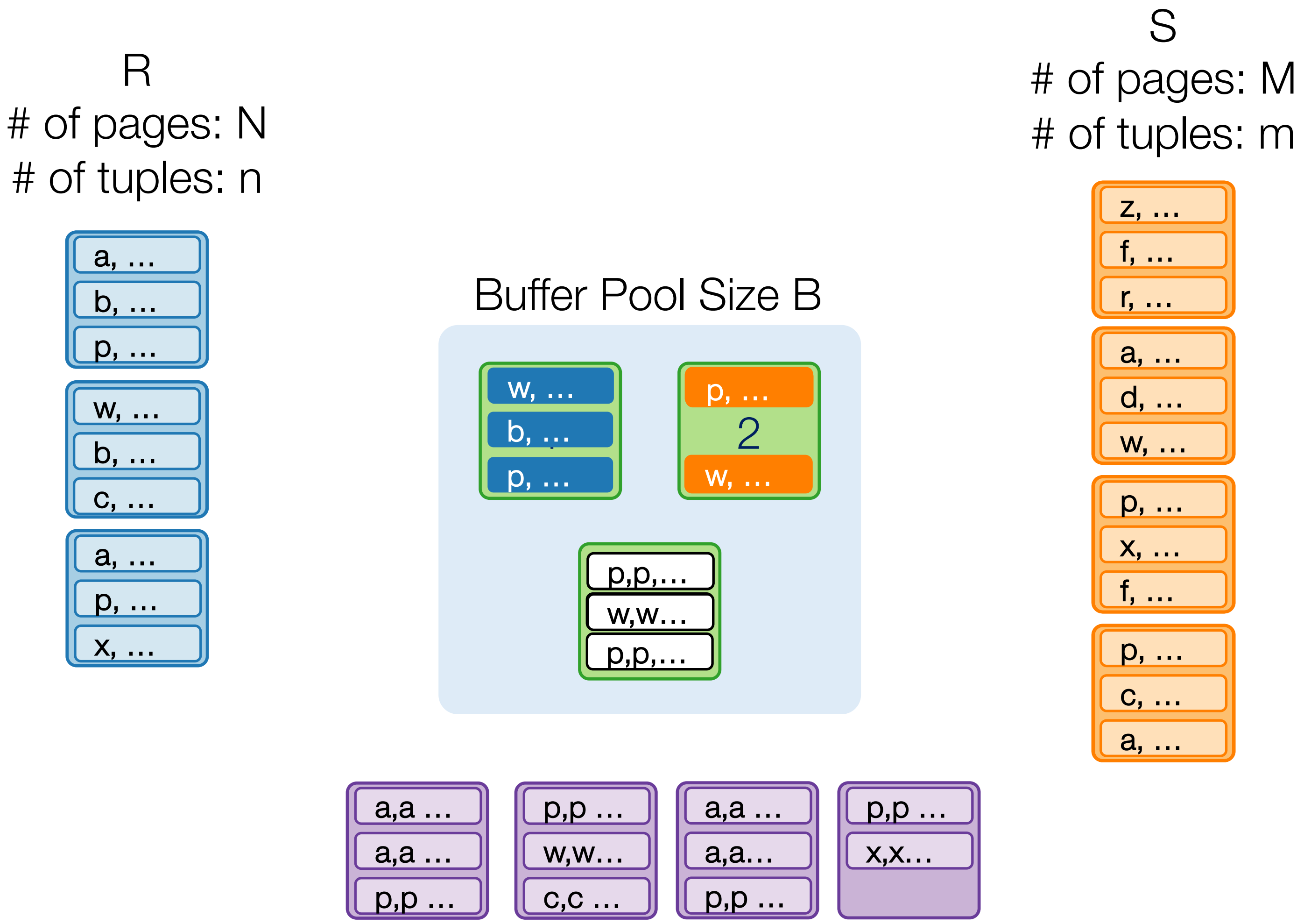
S – Students
(<u>*sid*</u>, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

B buffer size
$B \leq 102$

Algorithm	Naïve NLJ	Page NLJ	Block NLJ	Index NLJ	…
Cost (inner: S)					
Cost (inner: R)					
Effect of Buffer Size					

Nested Loops Join

Naïve Nested Loops Join




```
for each tuple r in R:
  for each tuple s in S:
    if(key(r) == key(s)):
      emit(r, s)
```

Cost Analysis – Naïve NLJ

R – Loans
(*sid*, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

S – Students
(*sid*, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

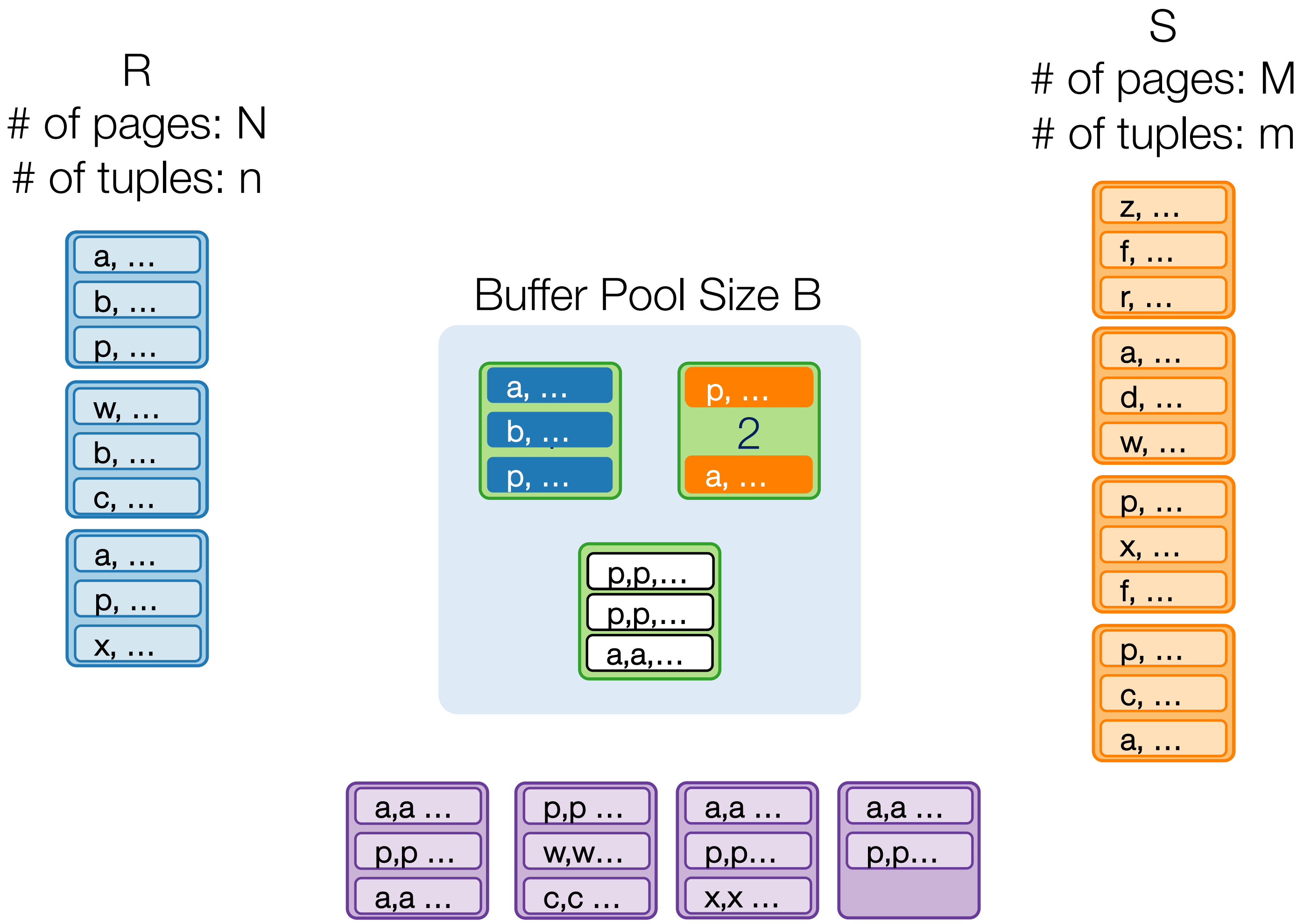
B buffer size
$B \leq 102$

Algorithm	Naïve NLJ	Page NLJ	Block NLJ	Index NLJ
Cost (inner: S)	$N + nM$ $500 + 40{,}000 \times 1000 = 40{,}000{,}500$			
Cost (inner: R)	$M + mN$ $1000 + 100{,}000 \times 500 = 50{,}001{,}000$			
Effect of Buffer Size	$B > 500 \rightarrow N + M = 1500$			

Does it matter what the inner relation is?
Yes! We want the larger relation inside.

Poor implementation
We could match multiple tuples at a time for pages that are loaded

Page Nested Loops Join




```
for each page pr in R:
  for each page ps in S:
    for each tuple r in pr:
      for each tuple s in ps:
        if(key(r) == key(s)):
          emit(r, s)
```

Cost Analysis – Page NLJ

R – Loans
(*sid*, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

S – Students
(*sid*, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

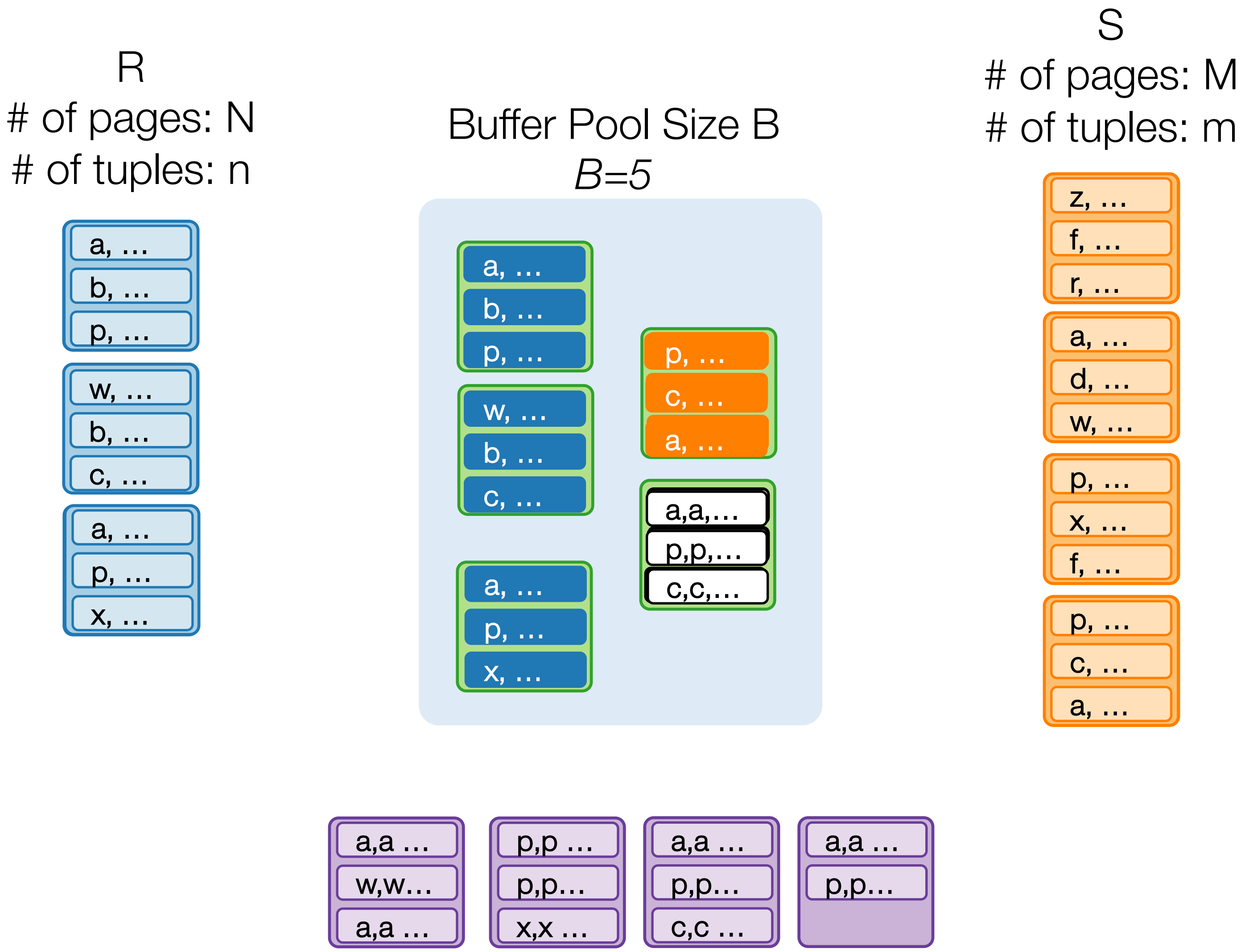
B buffer size
$B \leq 102$

Algorithm	Naïve NLJ	Page NLJ	Block NLJ	Index NLJ
Cost (inner: S)	$N + nM$ $40{,}000{,}500$	$N + NM$ $500{,}500$		
Cost (inner: R)	$M + mN$ $50{,}001{,}000$	$M + MN$ $501{,}000$		
Effect of Buffer Size	$B > 500 \rightarrow N + M = 1500$			

Better implementation but…

We could match multiple tuples at a time for multiple pages of the outer relation that are loaded

Block Nested Loops Join




```
block := [B-2] pages
for each block Br in R:
  for each page ps in S:
    for each tuple r in Br:
      for each tuple s in ps:
        if(key(r) == key(s)):
          emit(r, s)
```

Cost Analysis – Block NLJ

R – Loans
(<u>sid</u>, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

S – Students
(<u>sid</u>, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

B buffer size
$B \leq 102$

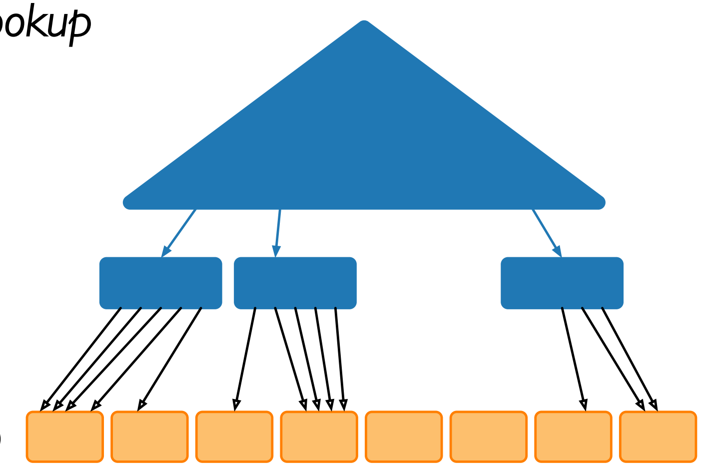
Algorithm	Naïve NLJ	Page NLJ	Block NLJ	Index NLJ
Cost (inner: S)	$N + nM$	$N + NM$	$N + \left\lceil \frac{N}{B-2} \right\rceil M$	
(B = 102)	$40{,}000{,}500$	$500{,}500$	$500 + \left\lceil \frac{500}{100} \right\rceil 1000 = 5500$	
Cost (inner: R)	$M + mN$	$M + NM$	$M + \left\lceil \frac{M}{B-2} \right\rceil N$	
(B = 102)	$50{,}001{,}000$	$501{,}000$	$1000 + \left\lceil \frac{1000}{100} \right\rceil 500 = 6000$	
Effect of Buffer Size	$B > 500 \rightarrow N + M = 1500$			

Even better implementation but …
What if we have an index on the inner relation?

Index Nested Loops Join

```
for each tuple r in R:
  s = lookup(key(r), index(S))
    if(s):
     emit(r, s)
```

Cost of a Lookup



Index traversal

+ Find page in file by RID

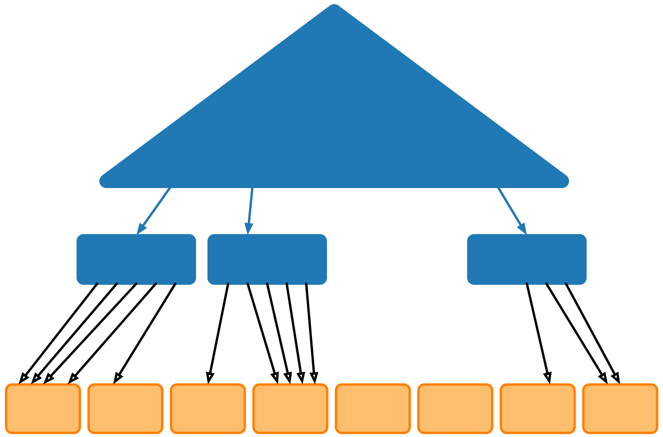
Height of a tree (2-4)

+ 1

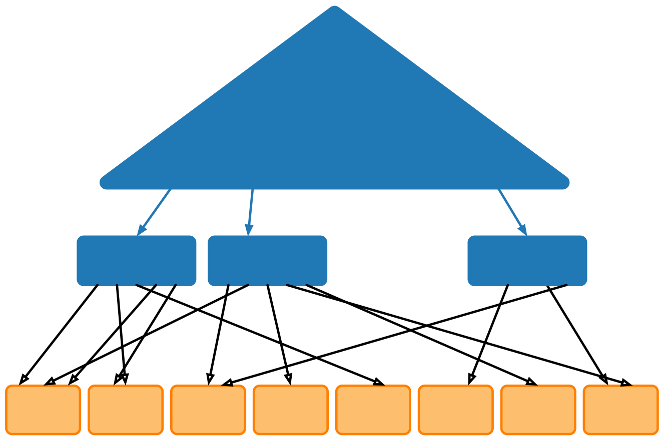
Multiple matches

Index traversal + # of matching pages

Clustered Index



Unclustered Index



Index traversal + # of matching tuples

Cost Analysis – Index NLJ

R – Loans
(*sid*, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

S – Students
(*sid*, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

B buffer size
$B \leq 102$

Algorithm	Naïve NLJ	Page NLJ	Block NLJ	Index NLJ
Cost (inner: S)	$N + nM$	$N + NM$	$N + \left\lceil \frac{N}{B-2} \right\rceil M$	$N + n(k)$
(B = 102)	$40{,}000{,}500$	$500{,}500$	5500	$500 + 40{,}000(2+1)$ $= 120{,}500$
Cost (inner: R)	$N + nM$	$N + NM$	$N + \left\lceil \frac{N}{B-2} \right\rceil M$	No index on R by sid
(B = 102)	$50{,}001{,}000$	$501{,}000$	6000	
Effect of Buffer Size	$B > 500 \rightarrow N + M$			

Compute the full cross product: Quadratic!	Linear

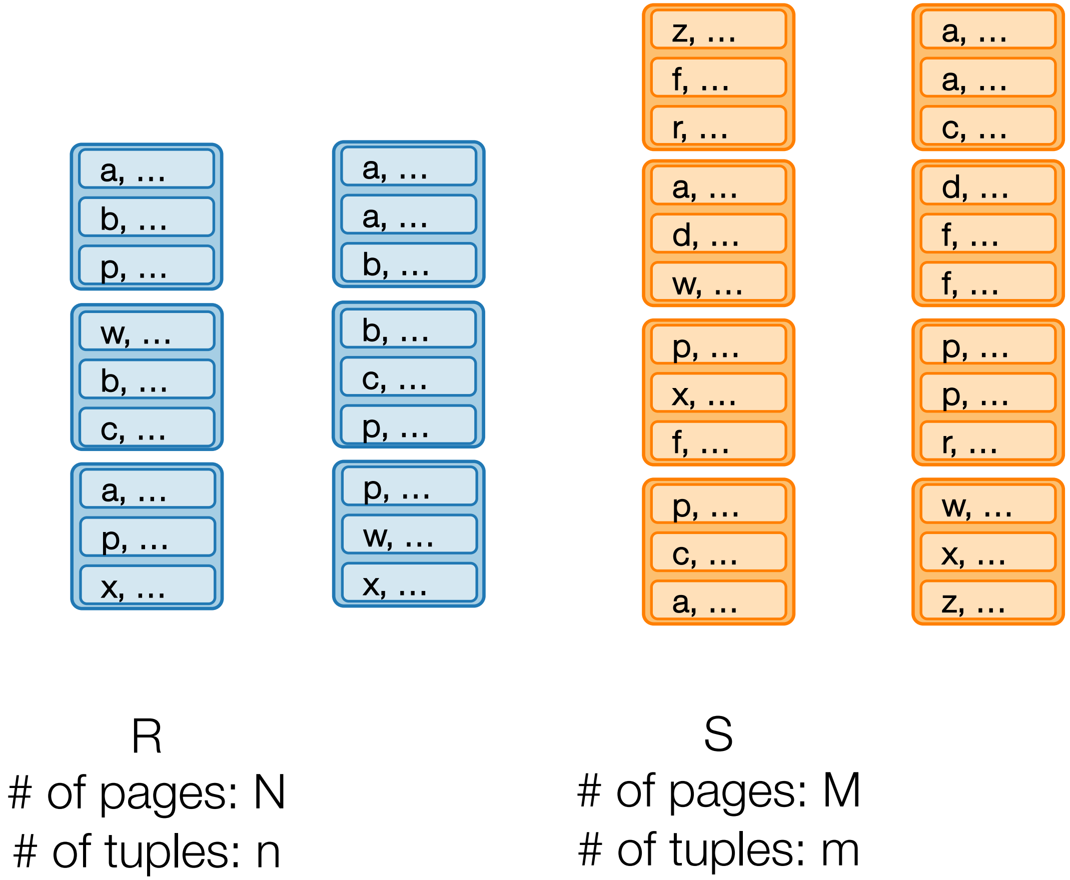
INLJ uses structure to overcome the need to check all other tuples

When does an index help?

- $n \ll MN$
- *Small number of lookups*
- *Small buffer*

Sort-Merge Join

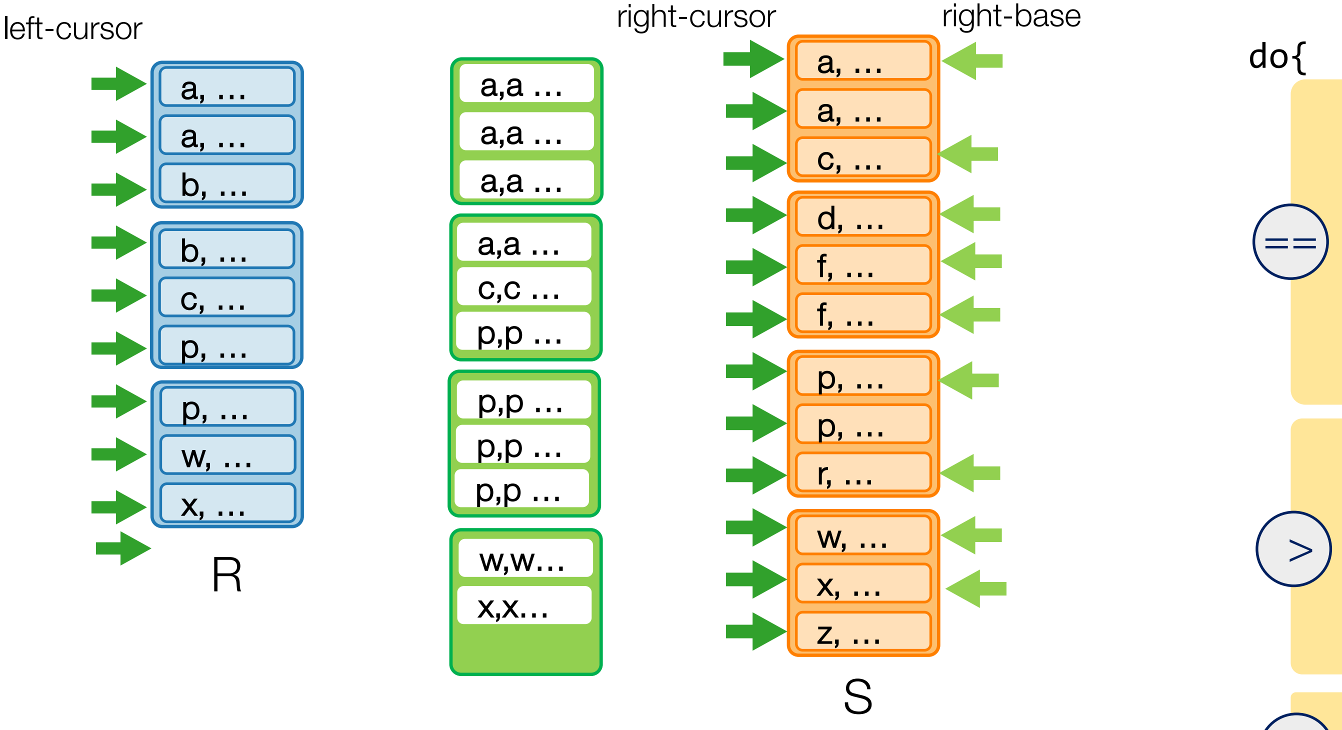
Sort-Merge Join



1. Sort R, S on join key using *external merge sort*

2. Scan sorted files and "merge"

Sort-Merge Join – Merge Pass




```
do{
    key(left-cursor) == key(right-base):
      right-cursor = right-base
      do{
         emit(r, s)
         advance(right-cursor)
      } while(key(left-cursor) == key(right-cursor))
      advance left-cursor

    key(left-cursor) > key(right-base):
      if(right-cursor > right-base):
         right-base = right-cursor
      else
         advance(right-base, right-cursor)

    key(left-cursor) < key(right-base):
      advance(left-cursor)
}
while(left-cursor != EOF)
```

Cost Analysis – Merge Join

R – Loans
(*sid*, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

S – Students
(*sid*, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

B buffer size
$B \leq 102$

Assuming files are sorted

Best Case Cost: $N + M$

- Single scan advancing cursors left and right!
- Equality Join with no duplicates!

Linear!

Worst Case Cost: nM

- Effectively a nested loop join
- Single duplicate key on both sides === a cross product! (very unlikely)

Quadratic!

Cost Analysis – SMJ

R – Loans
(*sid*, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

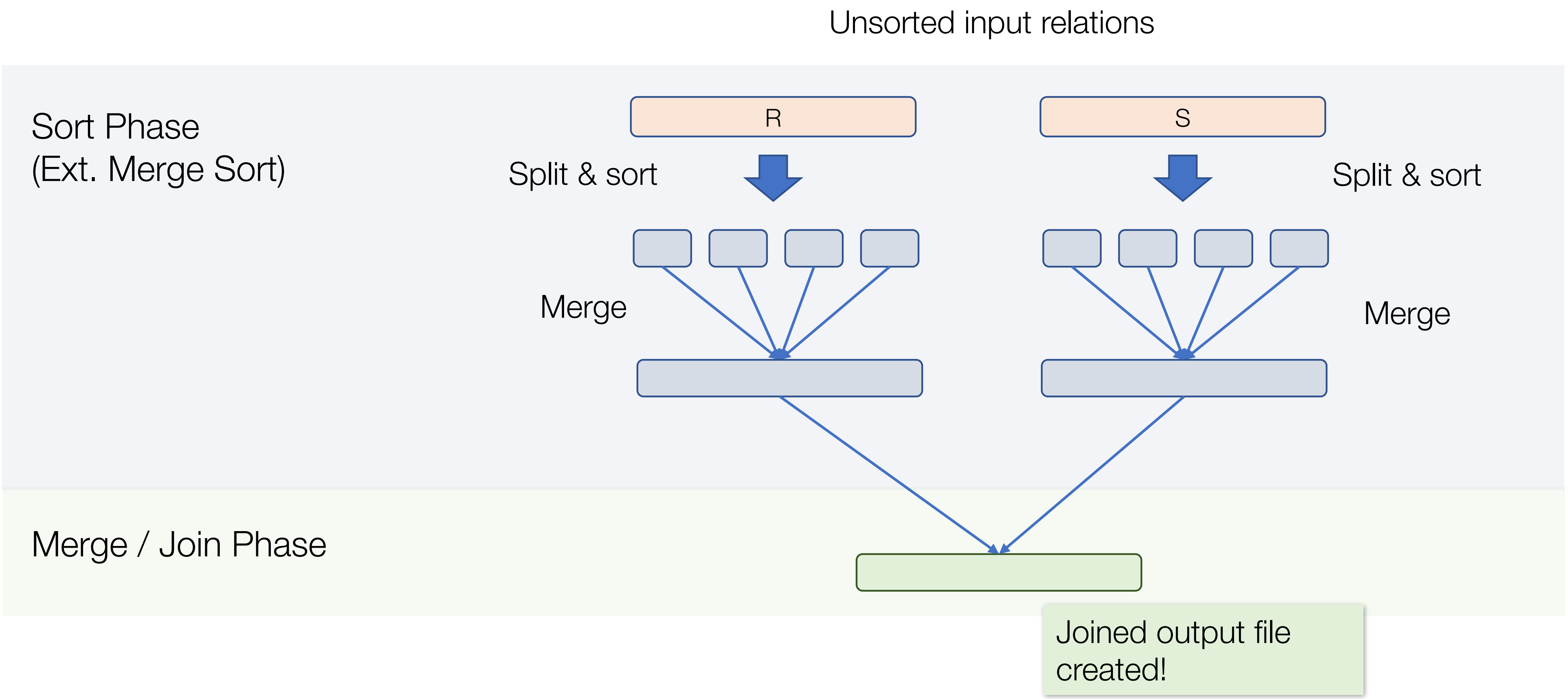
S – Students
(*sid*, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

B buffer size
$B \leq 102$

Algorithm	Block NLJ	Sort Merge Join
Cost	$N + \left\lceil \frac{N}{B-2} \right\rceil M$	Sort + Merge $(4N + 4M) + (N + M)$
$B = 102 \geq \max(\sqrt{N}, \sqrt{M})$	5500	7500
$B = 35 \geq \max(\sqrt{N}, \sqrt{M})$	16500	7500

But we can do better if we can integrate merge join within the sort pass!

Sort-Merge Join (No Refinement)



Slide Credit: https://thodrek.github.io/cs564-fall17/lectures/lecture-17/Lecture_17_Joins.pdf

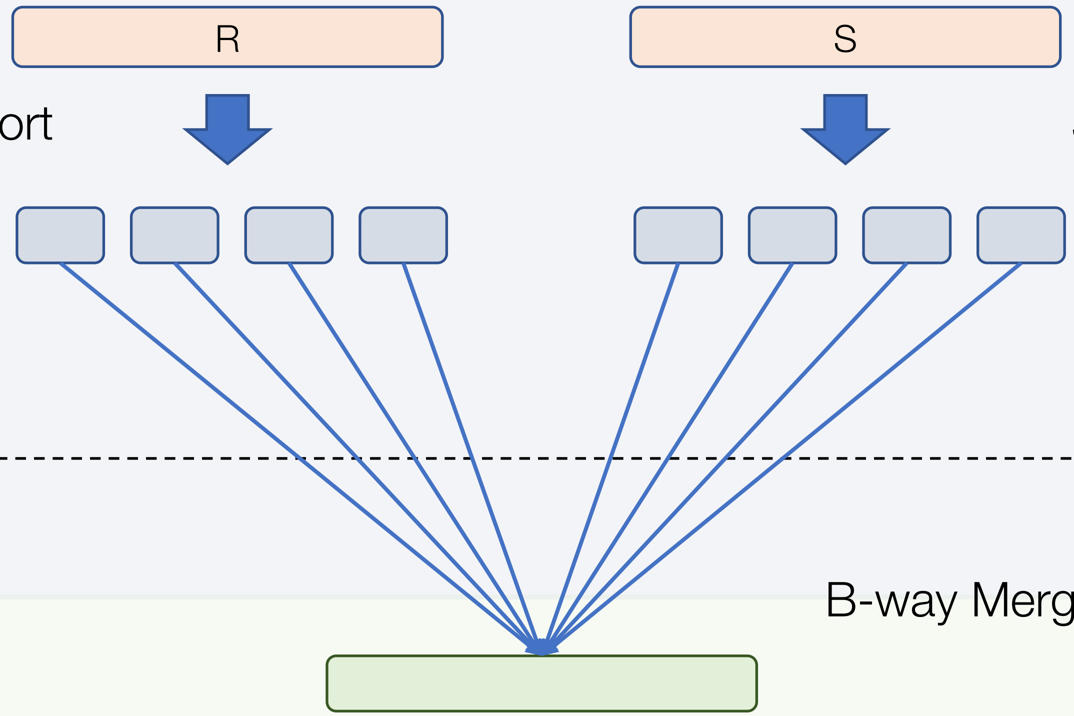
Sort-Merge Join (With Refinement)

Unsorted input relations

Given ***B*** buffer pages

Sort Phase
(Ext. Merge Sort)

Split & sort



Split & sort

<= ***B-1*** total sorted runs

B-way Merge / Join

Merge / Join Phase

Joined output file created!

How big is B for this optimization? $B \geq \sqrt{N} + \sqrt{M}$

Slide Credit: https://thodrek.github.io/cs564-fall17/lectures/lecture-17/Lecture_17_Joins.pdf

R – Loans
(*sid*, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

S – Students
(*sid*, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

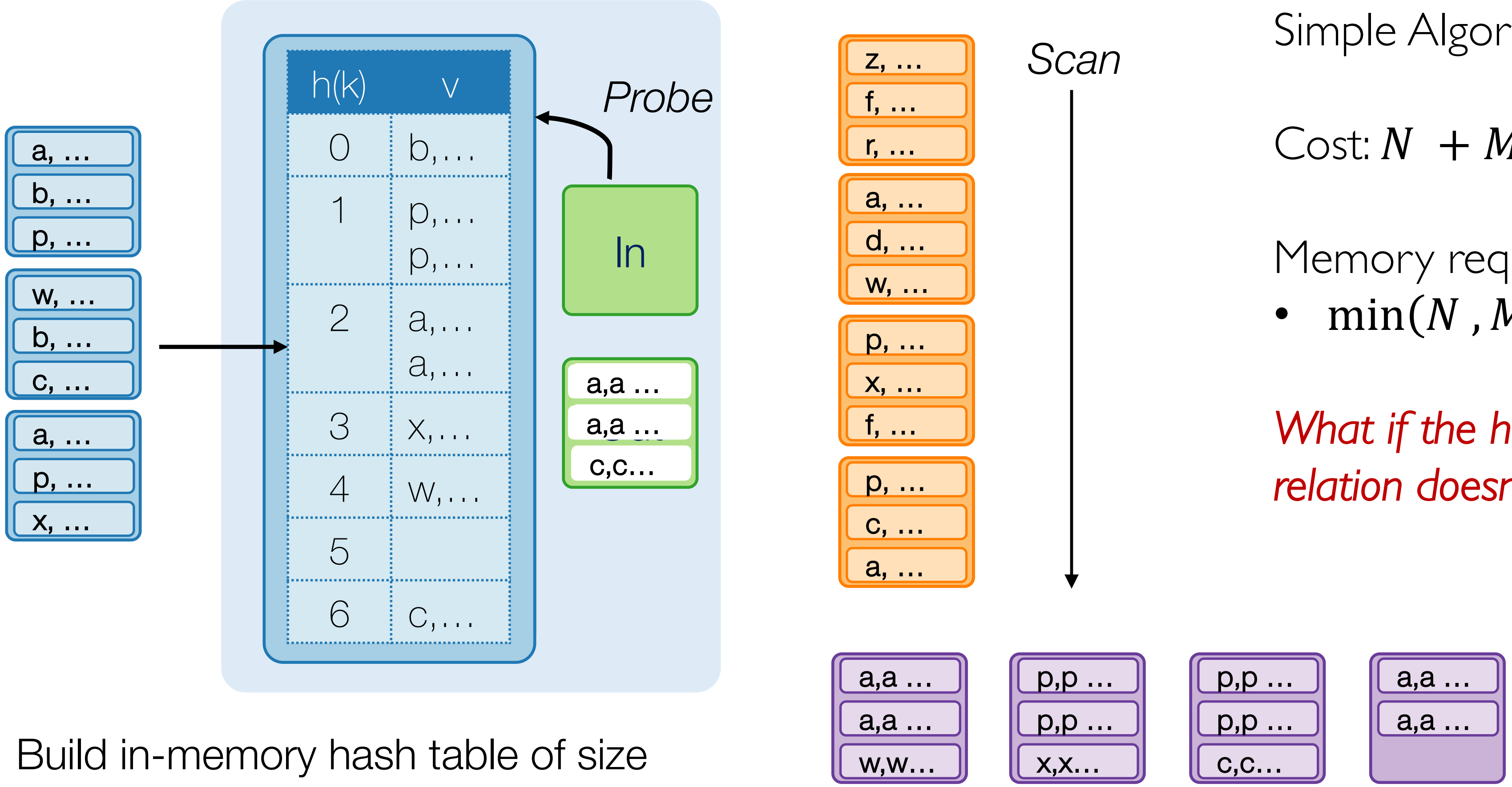
B buffer size
$B \leq 102$

Algorithm	Block NLJ	Sort Merge Join	SMJ - Refined
Cost	$N + \left\lceil \frac{N}{B-2} \right\rceil M$	Sort + Merge $(4N + 4M) + (N + M)$	First Sort Pass + Merge Pass (exclude output) $3N + 3M$
$B = 102 \geq \sqrt{N} + \sqrt{M}$	5500	7500	4500
$B = 55 \geq \sqrt{N} + \sqrt{M}$	10500	7500	4500

Cost Analysis – SMJ

Grace Hash Join

Naïve Hash Join



Build in-memory hash table of size (B-2) using hashing function *h(k)*

Simple Algorithm

Cost: $N + M$

Memory requirement

- $\min(N, M) < B - 2$

What if the hash table of the smaller relation doesn't fit?

Divide / Partition Phase

- Use a partitioning hash function to divide each table into $B - 1$ uniform partitions

Conquer Phase

For each partition P_i

- Build an in-memory hash table using the smaller partition(R): $P_i(R)$
- Hash table has to fit in $B - 2$ buffers
- Scan each page from S's partition $P_i(S)$ and probe the in-memory hash-table

Grace Hash Join

Cost and Memory Analysis of Grace Hash Join

Cost

Partitioning Phase + Matching Phase

$$(2N + 2M) + (N + M) = 3(N + M)$$

Memory Requirement

R is the smaller relation

Partitioning Phase divides R into (B-1) partitions of size $\frac{N}{B-1}$

Matching Phase requires each partition to be:

- $\frac{N}{B-1} < B - 2$
- $N < (B-2)(B-1)$
- $B \geq \sqrt{N}$

The probing relation S can be quite big, there are no restrictions on the size of its partitions!

R – Loans
(*sid*, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

S – Students
(*sid*, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

B buffer size
$B \leq 102$

Algorithm	Block NLJ	SMJ - Refined	Grace Hash Join
Cost	$N + \left\lceil \frac{N}{B-2} \right\rceil M$	$3N + 3M$	$3N + 3M$
$B = 102 \geq \sqrt{N} + \sqrt{M}$	5500	4500	4500
$B = 55 \geq \sqrt{N} + \sqrt{M}$	10500	4500	4500

Cost Analysis – Grace Hash Join

Cost Analysis – Grace Hash Join

R – Loans
(*sid*, eid, date, duration, …)
$N = 500$
$n = 40{,}000$
Tuples per page: 80

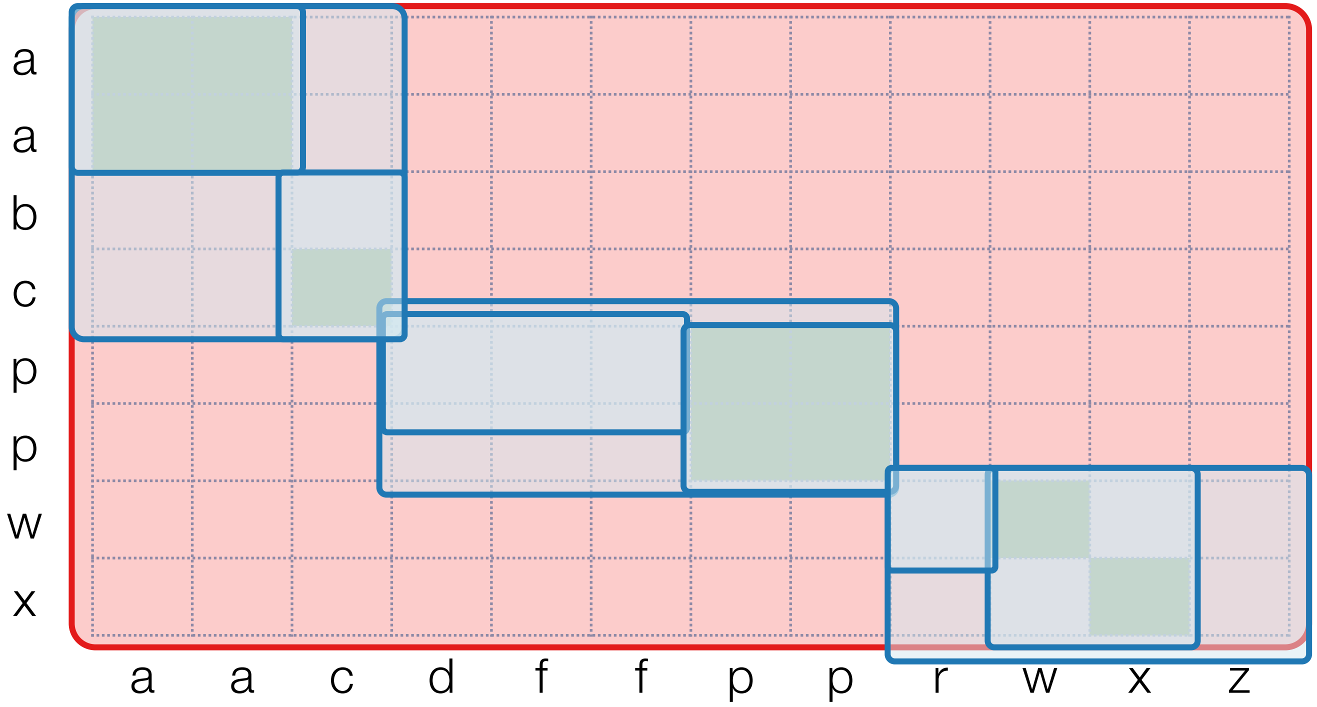
S – Students
(*sid*, name, major, …)
$M = 1000$
$m = 100{,}000$
Tuples per page: 100

B buffer size
$B \leq 102$

Algorithm	Block NLJ	SMJ - Refined	Grace Hash Join
Cost	$N + \left\lceil \frac{N}{B-2} \right\rceil M$	$3N + 3M$	$3N + 3M$
$B = 102 \geq \sqrt{N} + \sqrt{M}$	5500	4500	4500
$B = 55 \geq \sqrt{N} + \sqrt{M}$	10500	4500	4500
$B = 25 \geq \sqrt{N}$	22500	Needs more passes	4500

Summary

The Grace Hash Join partitioning breaks down the grid into smaller grids for further matching



The SMJ uses order to avoid searching the whole grid and establishing search boundaries

BNLJ doesn't take advantage of structure – we explore the whole grid for matches!

Visual Comparison – BNLJ, SMJ vs. GHJ

Key Takeaways

Nested Loops Join — Works for arbitrary join conditions

Index Nested Loops Join — If you have an index, equi-join and a small number of lookups! $n < NM$

Sort-Merge/Hash Join
- Linear IO complexity
- No index required
- Hash is better if one of the relations is much smaller
- Sort-Merge is better if order is required or if the relations are already sorted (perhaps from a previous join).

A typical DBMS implements all of these and uses a query optimizer to select the best join for a given query plan!