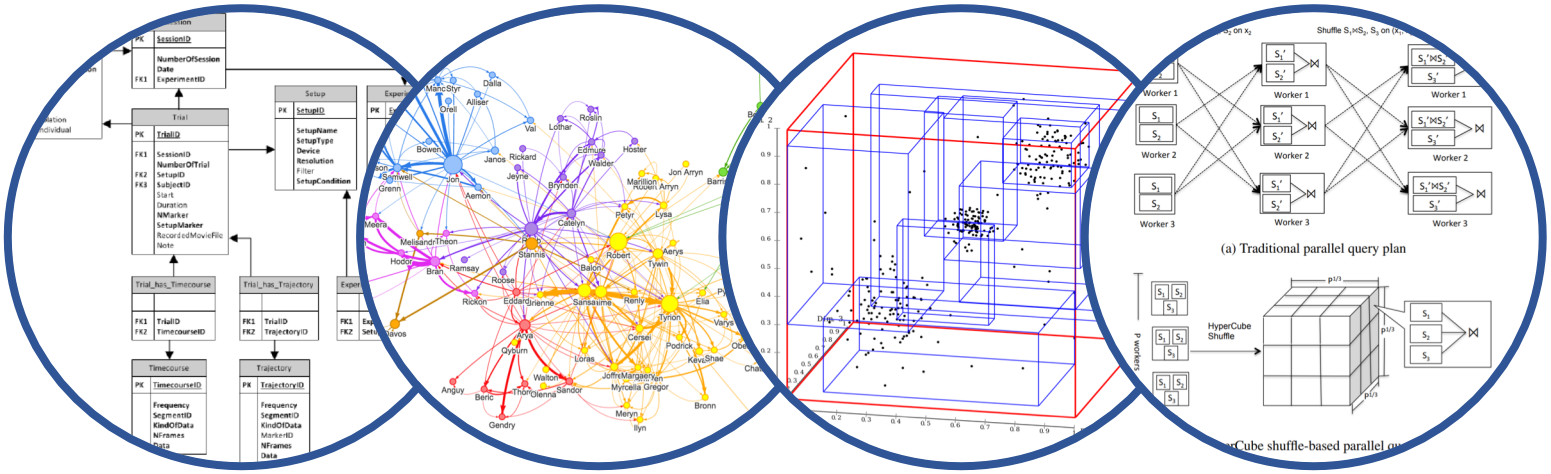


Introduction to Data Management

Query Cost Estimation

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Goals for Today

- Finish discussion on management ethics
  - Implicit disclosure
  - Passwords
  - Biased data
- Move to a short unit on RDBMS optimization

Implicit Disclosure

- FERPA allows institutions to disclose "directory information" without consent (institution policies can be stronger)
  - Name
  - Email
  - Photographs
  - Phone Number
- If users can derive sensitive information like grades, it violates FERPA

Implicit Disclosure

- "Hey, can you give me the directory information for students with a GPA of 3.5?"

Implicit Disclosure

- "Hey, can you give me the directory information for students with a GPA of 3.5?"

Reveals sensitive information by context

```
SELECT D.*
  FROM Directory AS D, Grades AS G
 WHERE D.id = G.id AND
       G.gpa = 3.5
```

Implicit Disclosure

Re-identification of Mass. Governor William Weld

Demonstrated by Latanya Sweeney, Ph.D. (MIT)

- Public voter data
  - Name
  - ZIP code
  - Sex
  - Birth date
  - …
- Anonymous insurance data
  - ZIP code
  - Sex
  - Birth date
  - Prescription
  - Diagnosis
  - …

Implicit Disclosure

Cambridge, MA Voter Data ($20)

Name	ZIP	Sex	Bday
…	…	…	…
W. Weld	12345	M	Feb 30
…	…	…	…

⋈

Anon. Insurance Data for Researchers

ZIP	Sex	Bday	MedInfo
…	…	…	…
12345	M	Feb 30	Afluenza
…	…	…	…

Implicit Disclosure

Cambridge, MA Voter Data ($20)

Name	ZIP	Sex	Bday
…	…	…	…
W. Weld	12345	M	Feb 30
…	…	…	…

⋈

Anon. Insurance Data for Researchers

ZIP	Sex	Bday	MedInfo
…	…	…	…
12345	M	Feb 30	Afluenza
…	…	…	…

6 matches on ZIP
to 3 matches on Sex
to 1 match on Bday

Implicit Disclosure

Cambridge, MA Voter Data ($20)

Name	ZIP	Sex	Bday
…	…	…	…
W. Weld	12345	M	Feb 30
…	…	…	…

⋈

Anon. Insurance Data for Researchers

ZIP	Sex	Bday	MedInfo
…	…	…	…
12345	M	Feb 30	Afluenza
…	…	…	…

Legal in 1997
Illegal since 2003 (HIPAA)

6 matches on ZIP
to 3 matches on Sex
to 1 match on Bday

Name	…	MedInfo
…	…	…
W. Weld	…	Afluenza
…	…	…

Storing Passwords

- Passwords are special
  - High potential for additional security compromises
  - Only operation that should be done is equality comparison

Storing Passwords

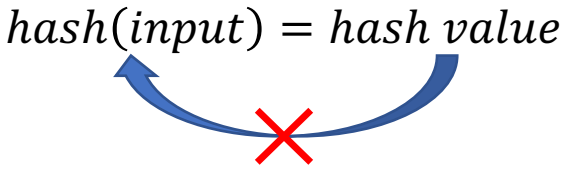
(bobtheninja246, password)

If you do this, Ted Codd will start rolling in his grave.

Username	Password
bobtheninja246	password
xXxDragonSlayerxXx	password
420_E-Sports_Masta	qwertyuiop

Storing Passwords

- Quick overview of hashing
  - Hash(input) → hash value
  - Hashing is deterministic
  - Ideally hashing is noninverible
  - Ideally hash values are uniformly spread out



Storing Passwords

Hash it!

(bobtheninja246, hash(password))

(bobtheninja246, FCgJFI9ryz)

Username	Hash
bobtheninja246	FCgJFI9ryz
xXxDragonSlayerxXx	FCgJFI9ryz
420_E-Sports_Masta	p8mel6usIF

Storing Passwords

Hash it!

(bobtheninja246, hash(password))

(bobtheninja246, FCgJFI9ryz)

Issues/pitfalls:

- Hashing functions have precomputed "rainbow tables"
- Some hashing functions are fast so brute forcing attacks can happen
- Patterns can occur for the same passwords

Username	Hash
bobtheninja246	FCgJFI9ryz
xXxDragonSlayerxXx	FCgJFI9ryz
420_E-Sports_Masta	p8mel6usIF

Storing Passwords

Salt it and hash it!

(bobtheninja246, slowhash(password * random salt), random salt)

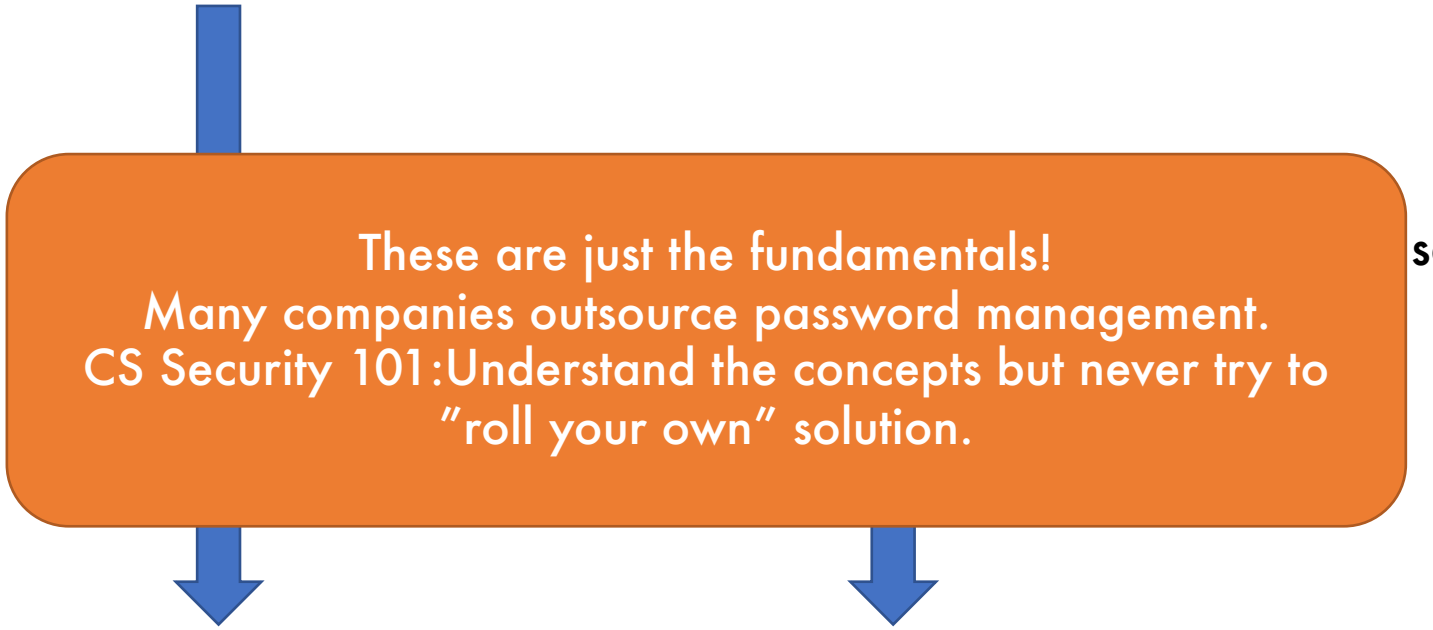
(bobtheninja246, slowhash(password * stored salt))

Username	Hash	Salt
bobtheninja246	HHxrd5o7Cn	WUKhhIFBLc
xXxDragonSlayerxXx	7rYFQIowpW	mq5rFL6JzF
420_E-Sports_Masta	cQF4DdSFfn	S8e0zpATNR

Storing Passwords

Salt it and hash it!

(bobtheninja246, slowhash(password * random salt), random salt)



salt))

Username	Hash	Salt
bobtheninja246	HHxrd5o7Cn	WUKhhIFBLc
xXxDragonSlayerxXx	7rYFQIowpW	mq5rFL6JzF
420_E-Sports_Masta	cQF4DdSFfn	S8e0zpATNR

Data Quality

- Quality is not only about cleanness
- Quality may also involve significance
  - Are certain groups large enough to draw meaningful aggregates?
  - If my data is a sample of a population, does it accurately depict that population?

Worlds Shortest Intro to Machine Learning

- Training data → Prediction program
  - Prediction program believes that the training data is representative of a population and covers all cases
  - If you never gave a hotdog recognizer examples of a hotdog, would you expect it to work?

Outline

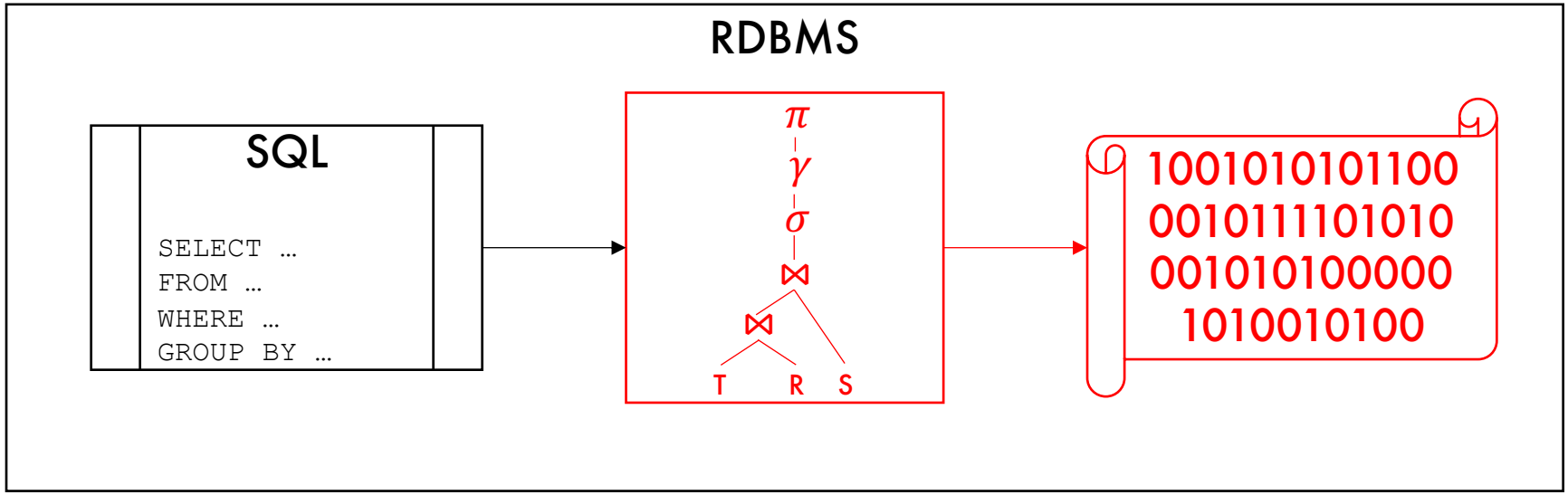
- Query execution
- Cost estimation ideas and assumptions
- Join algorithm analyses

Query Optimization

- So you wrote a SQL query…
  - SQL only tells the computer *what* you want
  - RDBMS needs to find a good way to actually do it

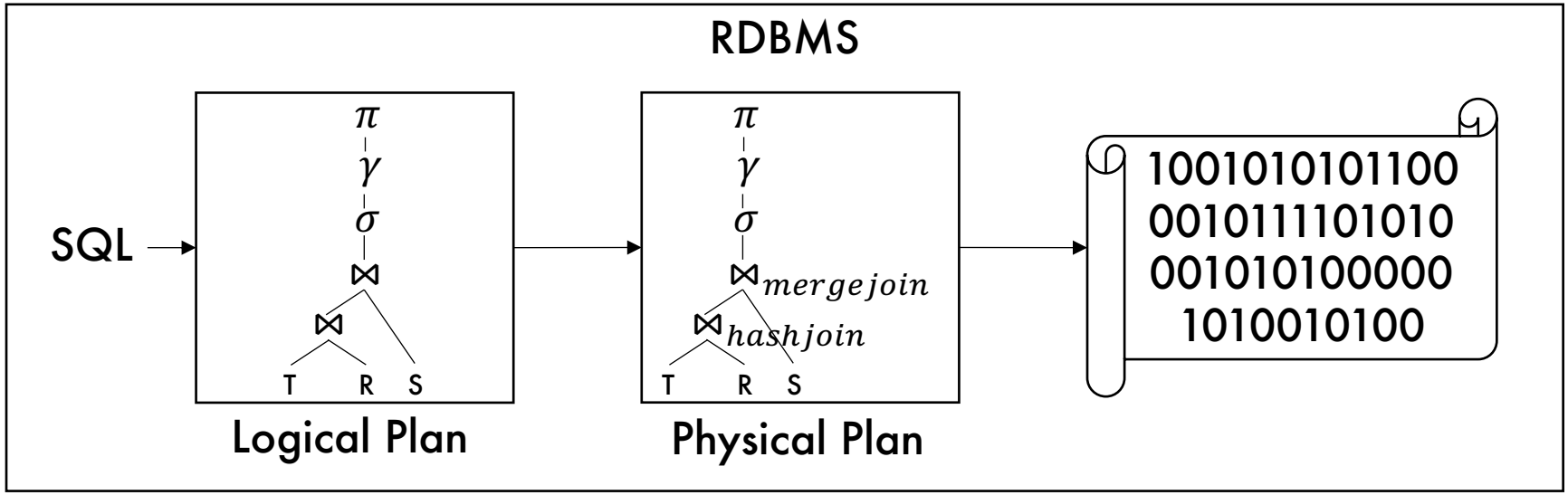
Logical vs Physical Plans

- SQL is translated into RA
- RA (logical plan) does not fully describe execution
- RA *with algorithms* (physical plan) is needed



Logical vs Physical Plans

- SQL is translated into RA
- RA (logical plan) does not fully describe execution
- RA *with algorithms* (physical plan) is needed



Disclaimer

- Cost estimation is an active research topic
- Equations and methods discussed in this class form a foundation of concepts, but usually cannot compare to a commercialized solution

Plan Enumeration

RDBMS optimize by selecting the estimated **least cost plan**

- SQL → RA
- RA → Set of equivalent RA
- Set of equivalent RA → Set of physical plans
- Set of physical plans → The least cost plan (run it)

Plan Enumeration

RDBMS

SQL

```
SELECT *
  FROM T, R, S
 WHERE …
```

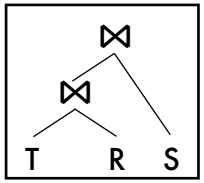
Plan Enumeration

RDBMS

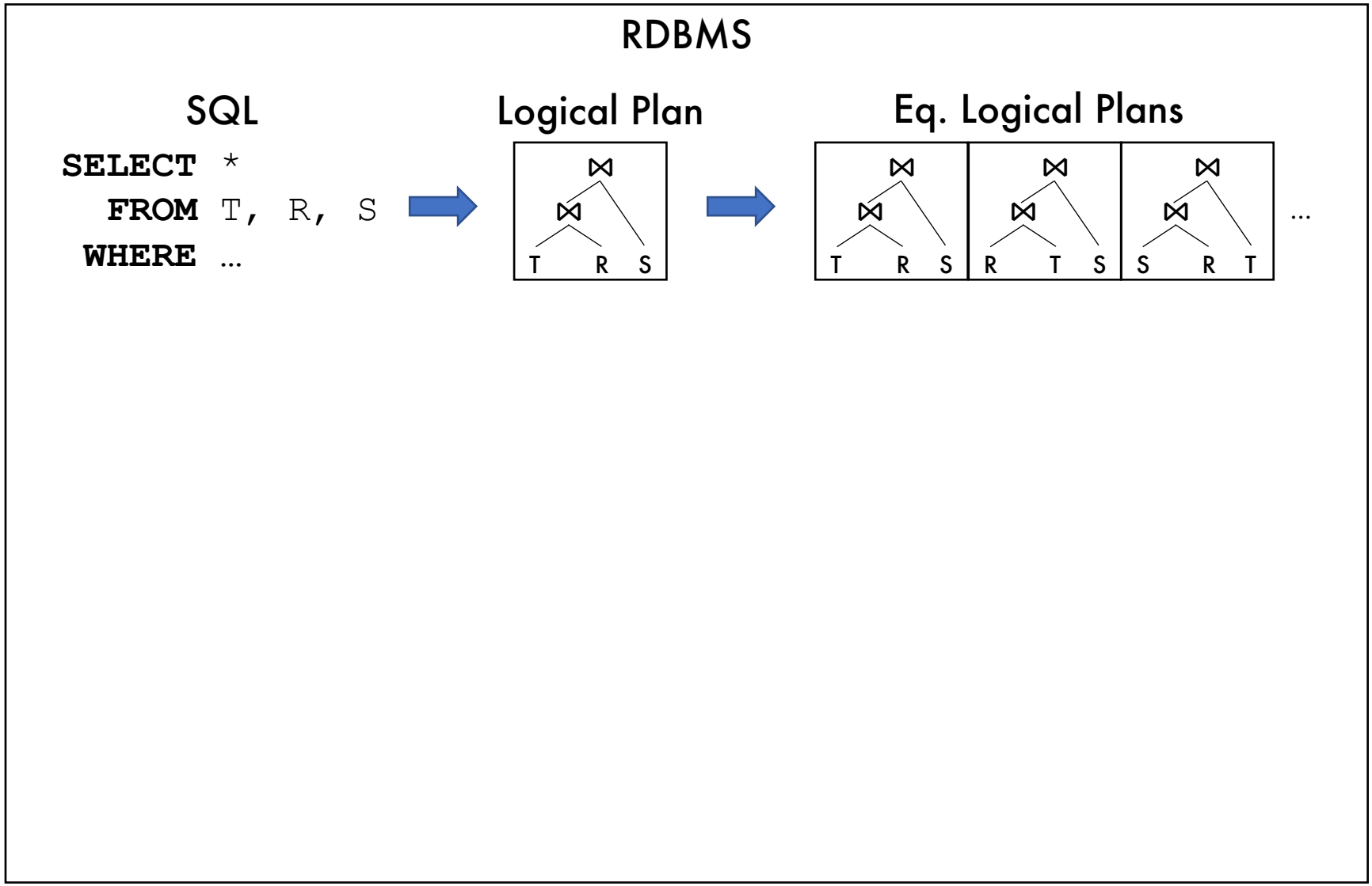
SQL

```
SELECT *
  FROM T, R, S
 WHERE ...
```

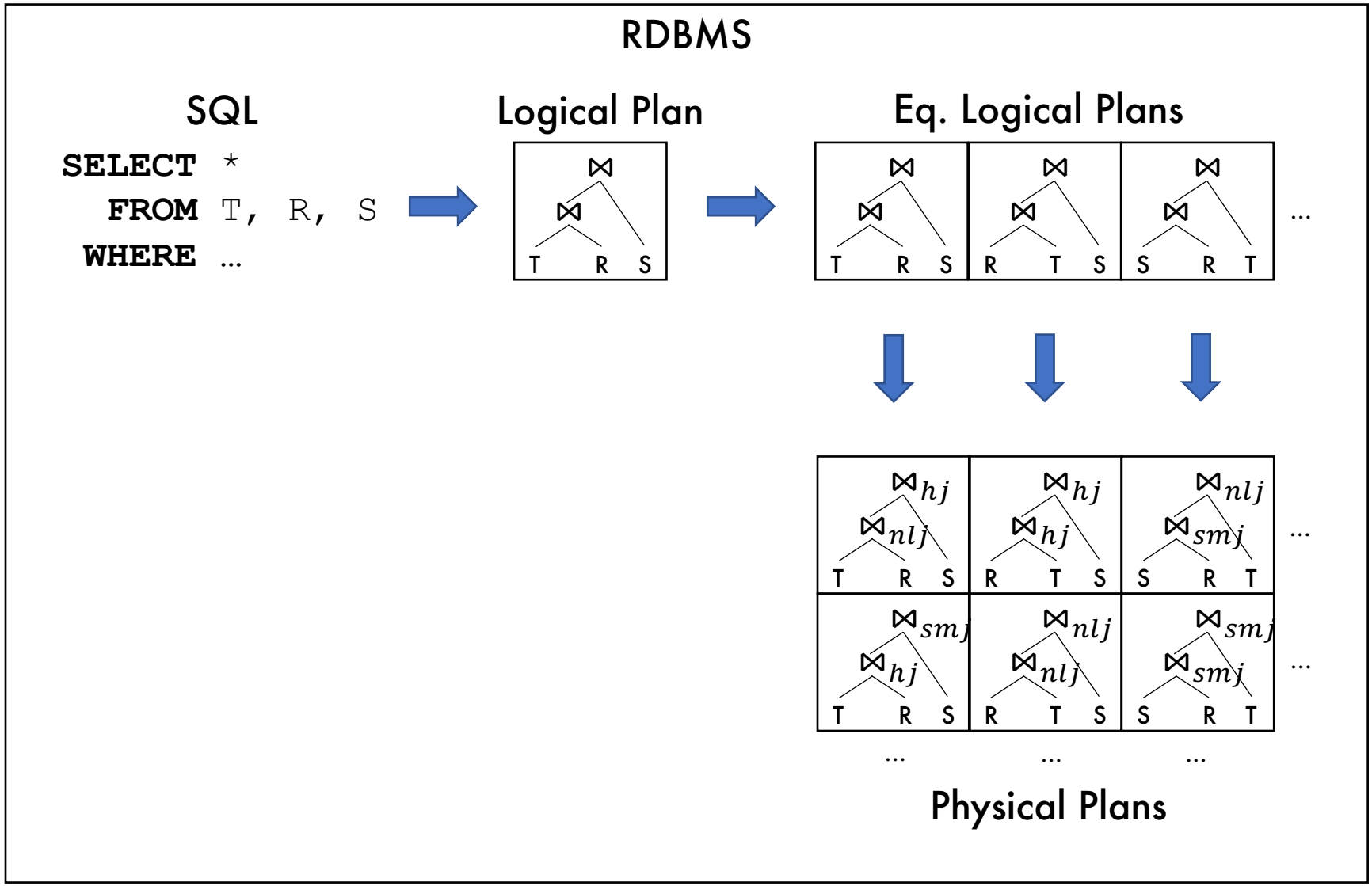
Logical Plan



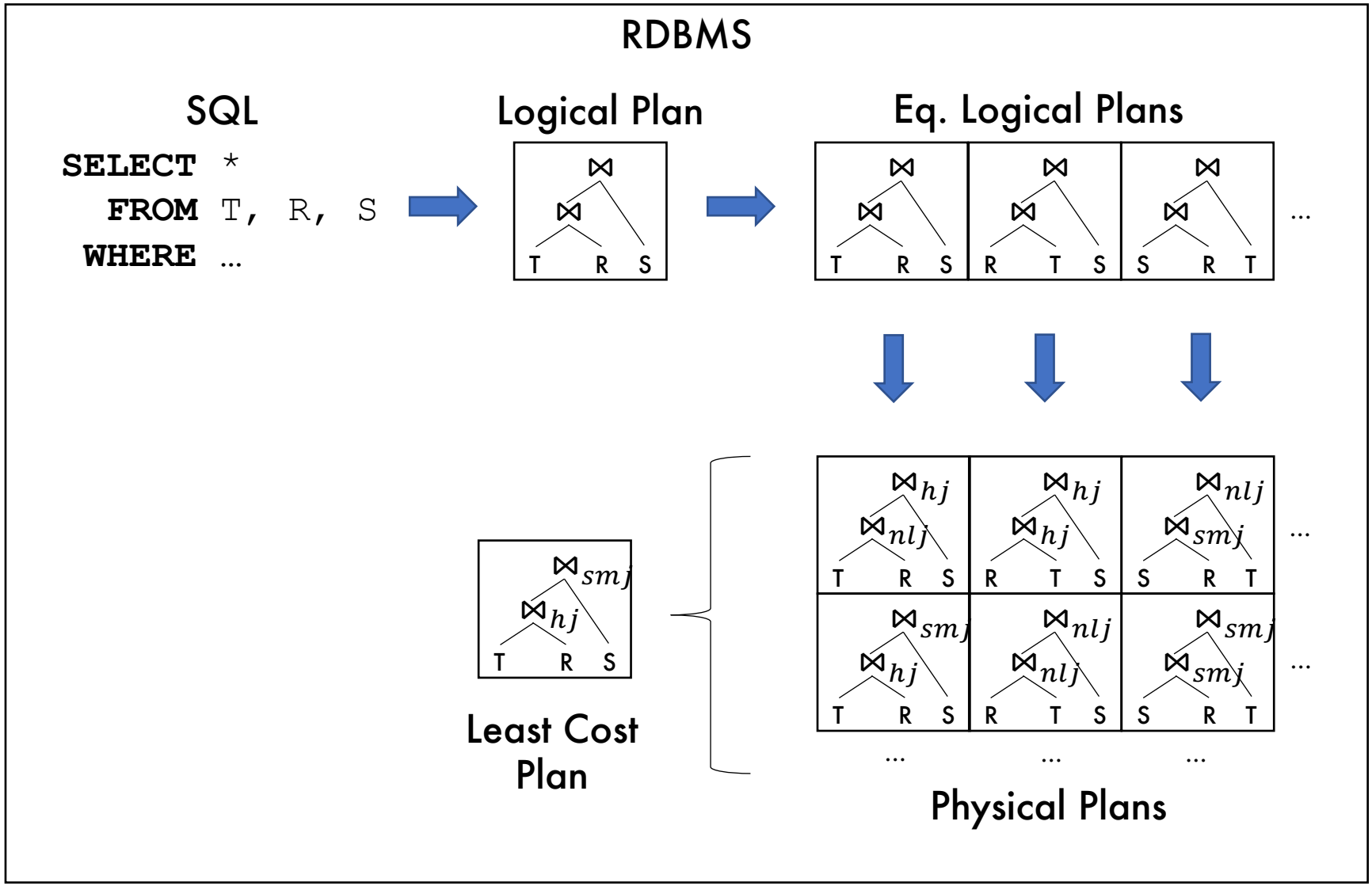
Plan Enumeration



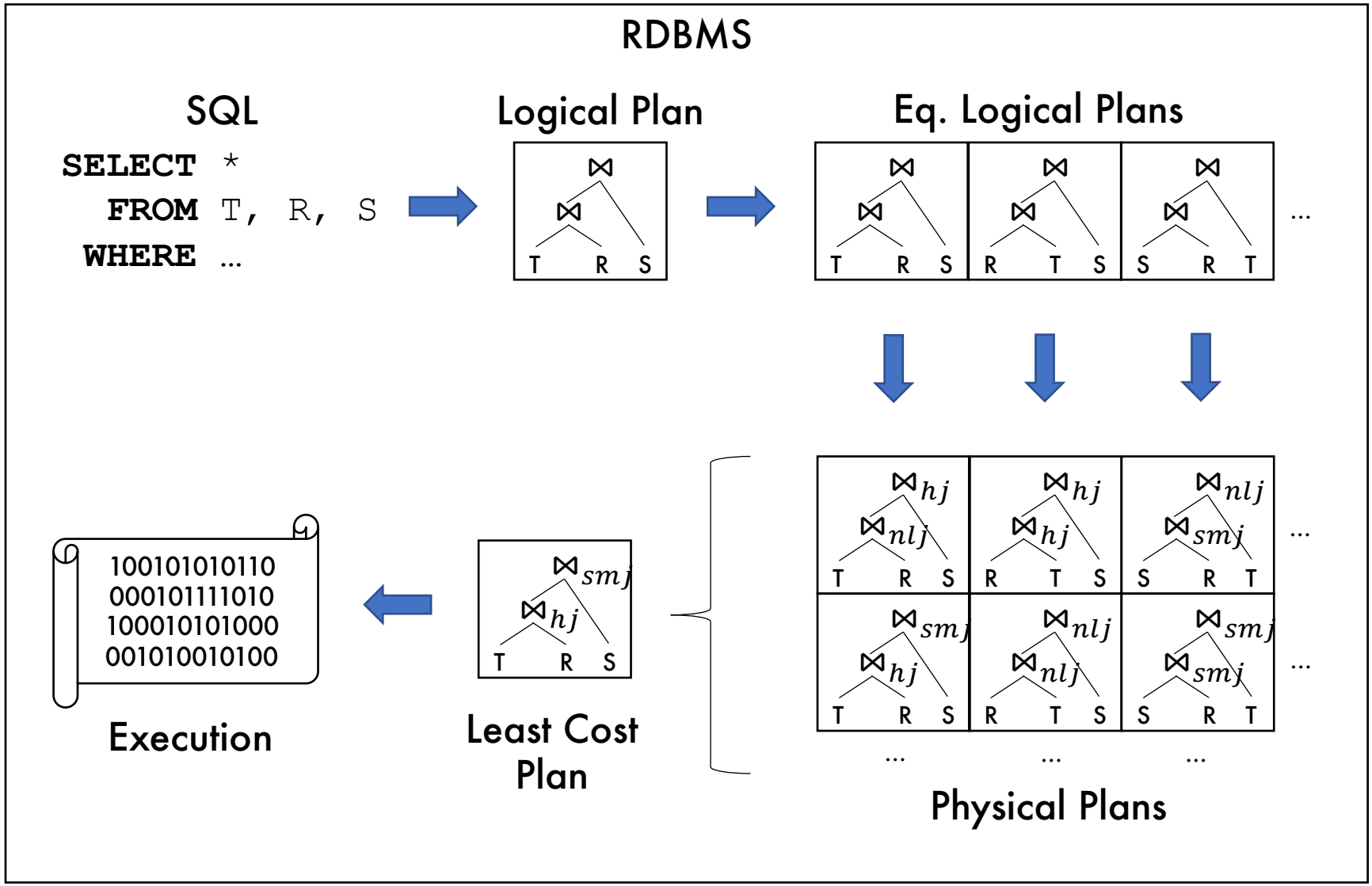
Plan Enumeration



Plan Enumeration



Plan Enumeration



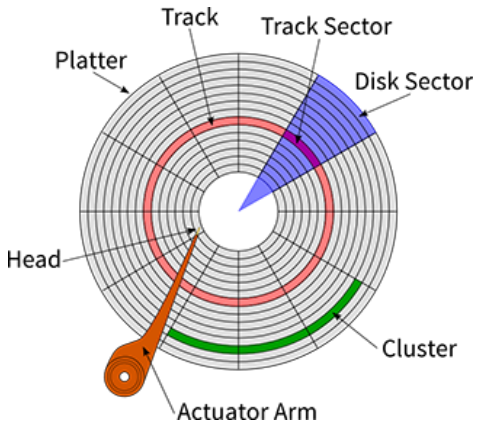
Assumptions

For this class we make a lot of assumptions

- **Disk-based storage**
  - HDD not SDD
- **Row-based storage**
  - Tuples are stored contiguously
- **IO cost** (reading from disk) only considered
  - Comprehensive cost estimation involves many factors
    - Network, disk, and CPU cost
    - Cache (main mem., L1 cache, L2 cache, disk cache, …)
  - Reading from disk is usually the biggest component
    - One IO access is ~100000x more expensive than one main memory access
- **Cold cache** (no data preloaded)

Disk Storage

- Mechanical hard drive
- Smallest unit of memory that can be read at once is a **block**
  - Usually 512B to 4kB
- DBMS will attempt to store table files in **contiguous chunks of memory** on disk
- Sequential disk reads are faster than random ones



Disk Storage

- Tables are stored as files
  - **Heap file** → Unsorted tuples (this lecture)
  - **Sequential file** → Sorted tuples (next lecture)
    - Attribute(s) sorted on is called a key because the database community is good at naming things

Making Cost Estimations

- RDBMS keeps statistics about our tables
  - **B(R)** = **# of blocks** in relation R
  - **T(R)** = **# of tuples** in relation R
  - **V(attr, R)** = **# of distinct values** of attr in R

- We only discuss **join algorithms** because they are usually the most expensive part of a query
- We only discuss nested-loop and single-pass join algorithms because cost equations get complex

Join Algorithm Summary

- Nested-Loop Join
  - Versatile
- Hash Join (single pass)
  - Fast
  - Needs at least one input to be small
- Sort-Merge Join (single pass)
  - Fast
  - Sorts data at the same time!
  - Needs both inputs to be small

Join Algorithm Summary

- **Nested-Loop Join**
  - Versatile
- Hash Join (single pass)
  - Fast
  - Needs at least one input to be small
- Sort-Merge Join (single pass)
  - Fast
  - Sorts data at the same time!
  - Needs both inputs to be small

Nested Loop Join Algorithm

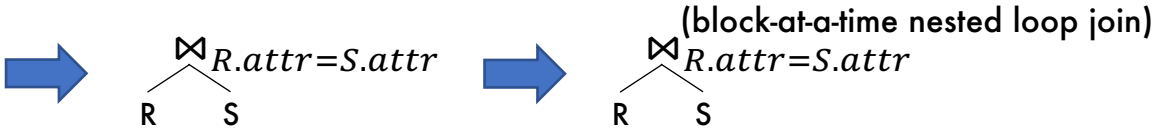
- Similar execution logic as nested-loop semantics

```
for each tuple t1 in R:
    for each tuple t2 in S:
        if t1 and t2 can join:
            output (t1,t2)
```

Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



To save time, we will read tuples from disk to memory in units of **blocks**. For fixed width tuples, each block will contain the same number of tuples.

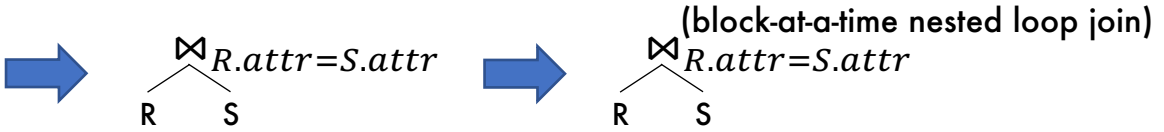
Block-at-a-time nested loop join:

```
for each block bR in R:
    for each block bS in S:
        for each tuple tR in bR:
            for each tuple tS in bS:
                if tR and tS can join:
                    output (tR,tS)
```

Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



To save time, we will read tuples from disk to memory in units of **blocks**. For fixed width tuples, each block will contain the same number of tuples.

Reads blocks from disk to memory

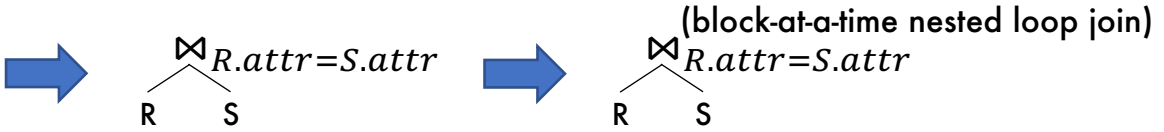
Block-at-a-time nested loop join:

```
for each block bR in R:
    for each block bS in S:
        for each tuple tR in bR:
            for each tuple tS in bS:
                if tR and tS can join:
                    output (tR,tS)
```

Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



To save time, we will read tuples from disk to memory in units of **blocks**. For fixed width tuples, each block will contain the same number of tuples.

Block-at-a-time nested loop join:

```
for each block bR in R:
    for each block bS in S:
        for each tuple tR in bR:
            for each tuple tS in bS:
                if tR and tS can join:
                    output (tR,tS)
```

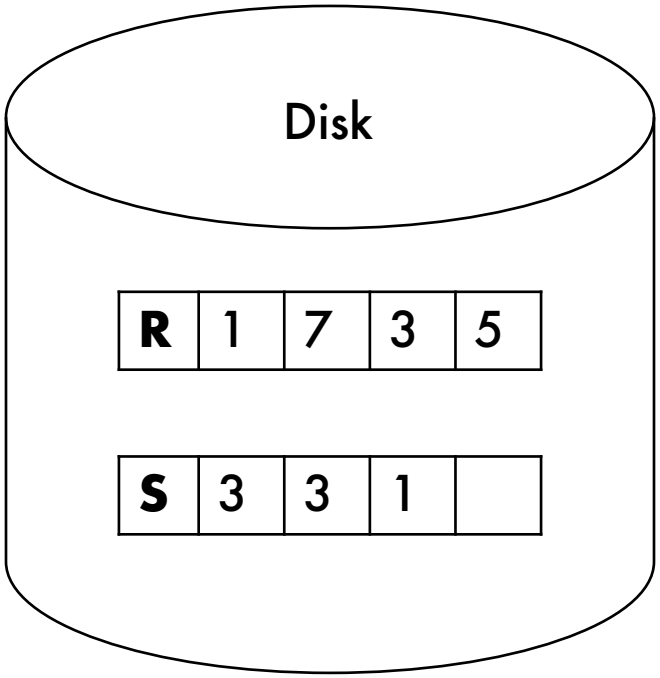
Blocks are joined in memory

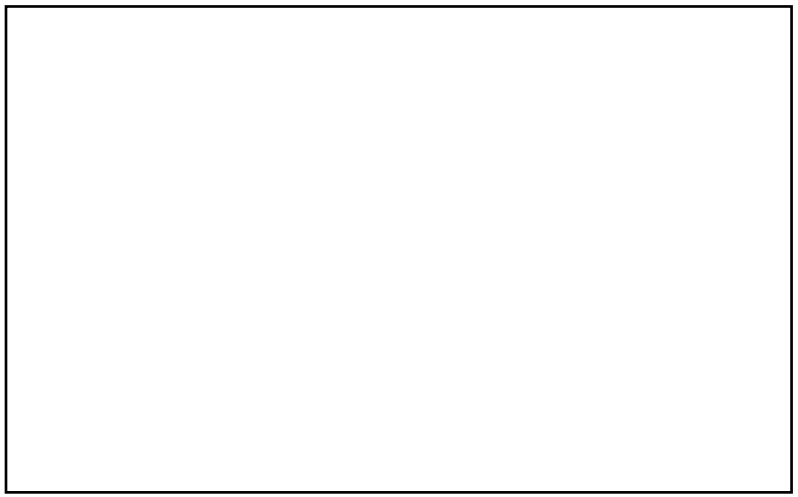
Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

x → A tuple where x is the join attribute value





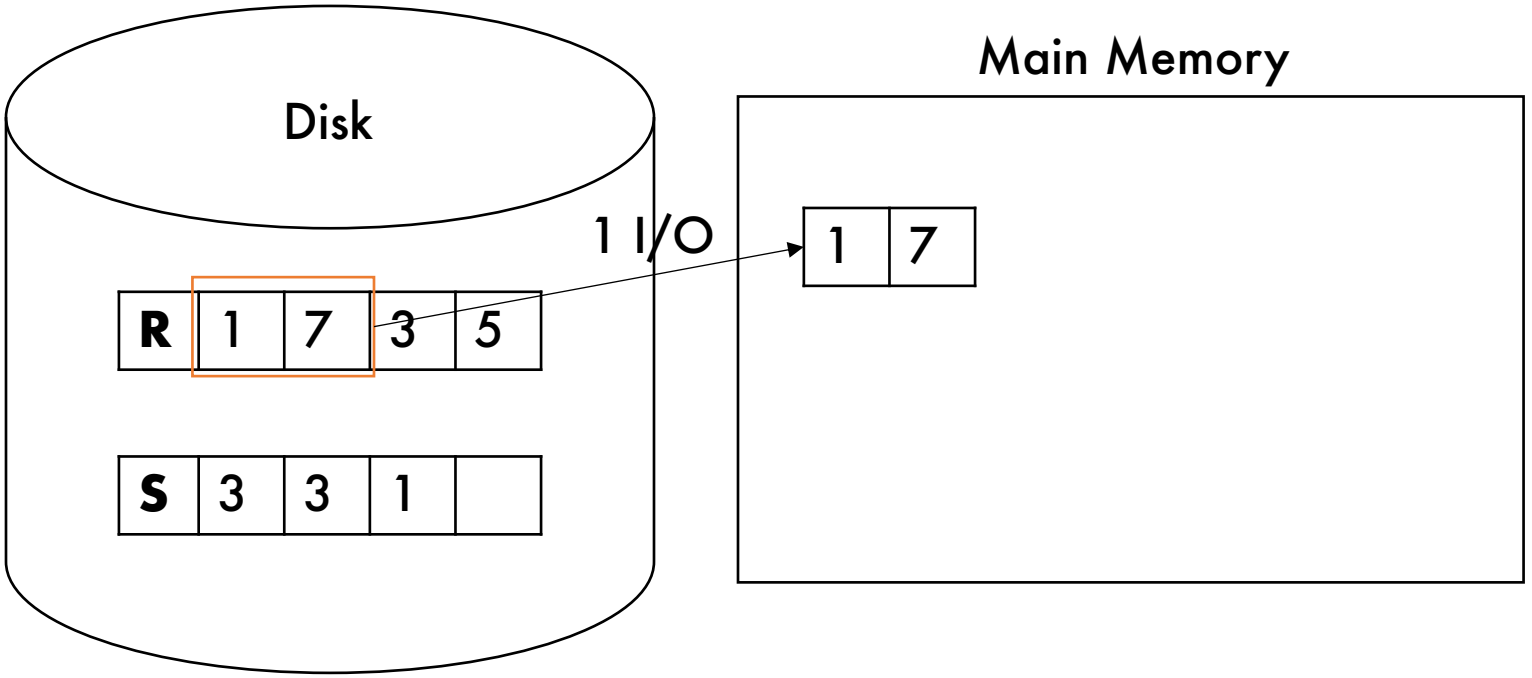
Assume block size = 2 tuples

Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

x → A tuple where x is the join attribute value



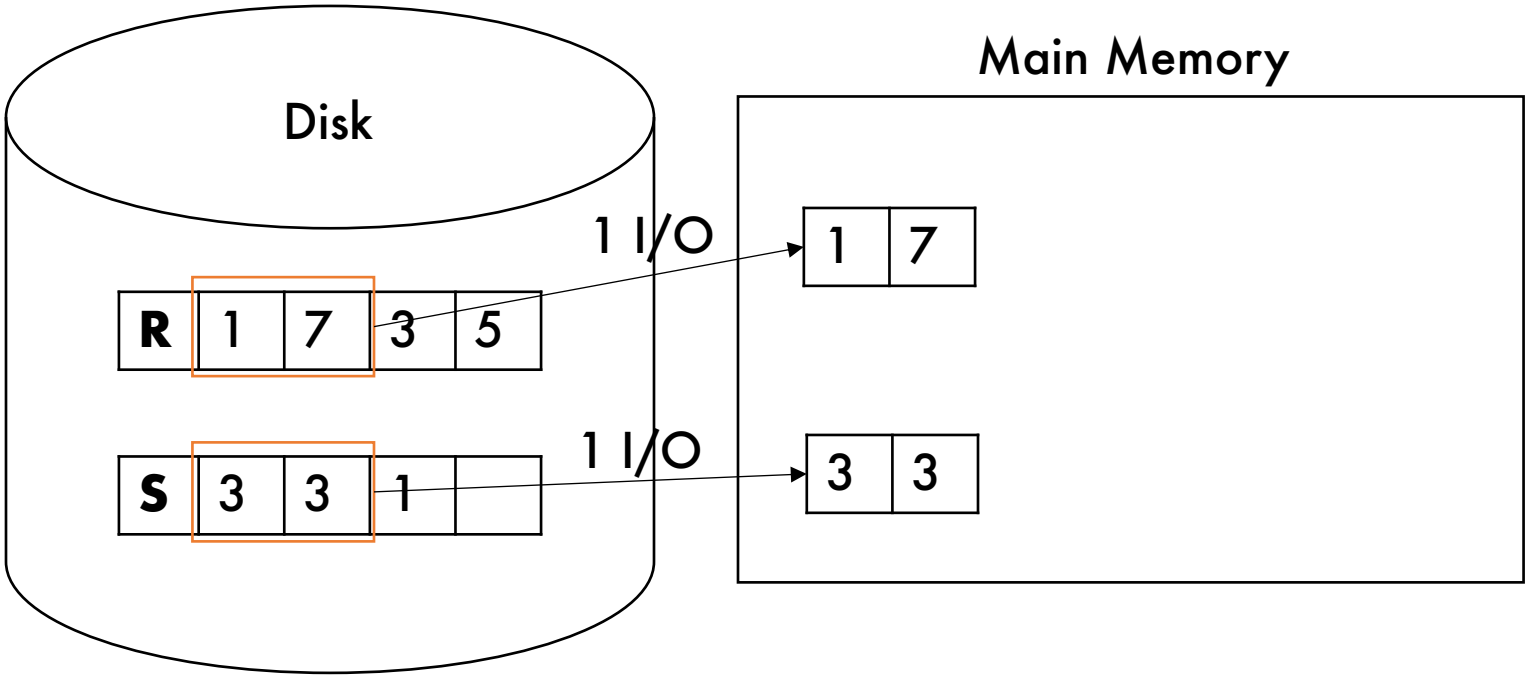
Assume block size = 2 tuples

Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

x → A tuple where x is the join attribute value

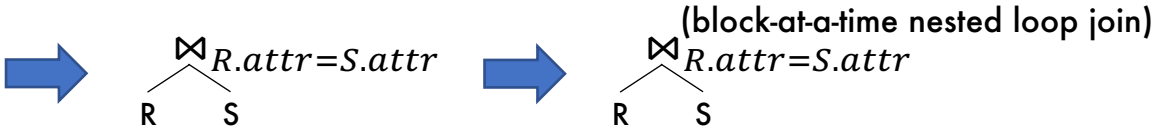


Assume block size = 2 tuples

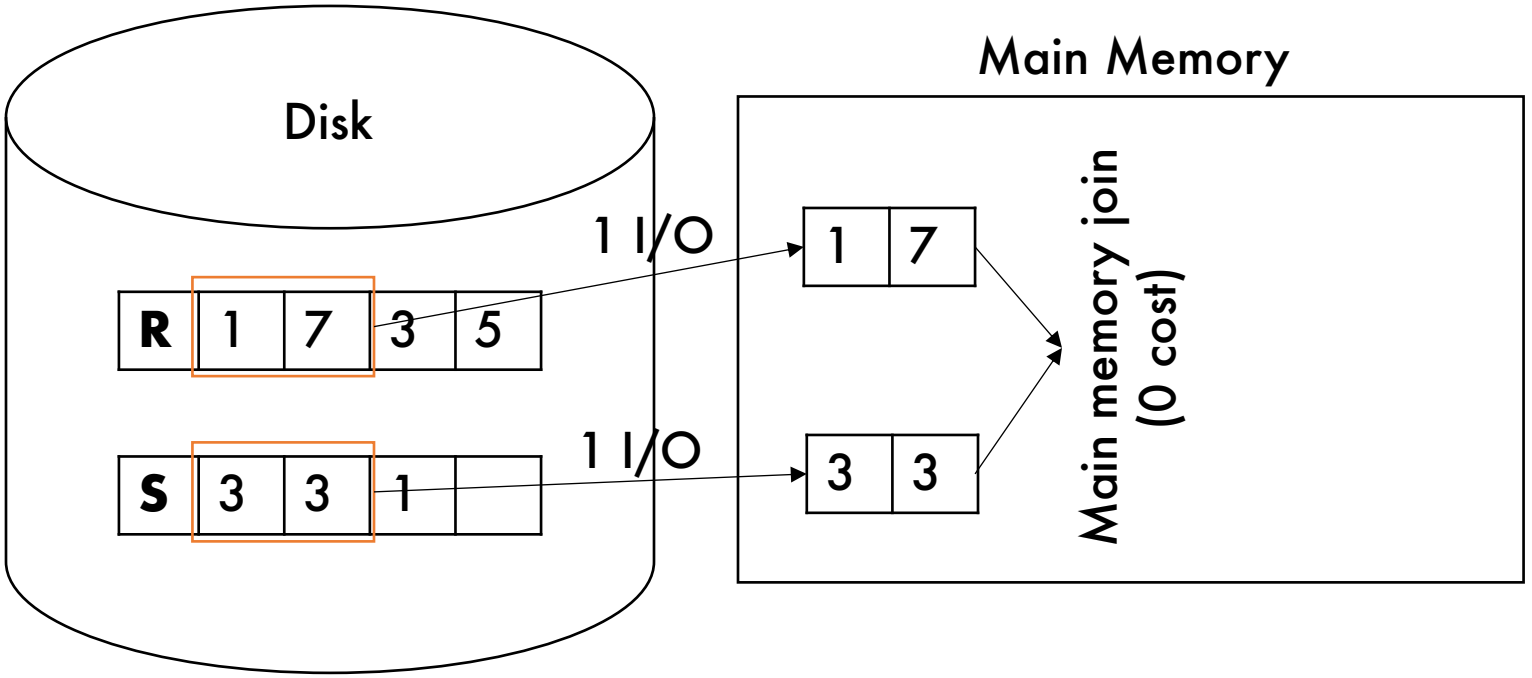
Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



x → A tuple where x is the join attribute value



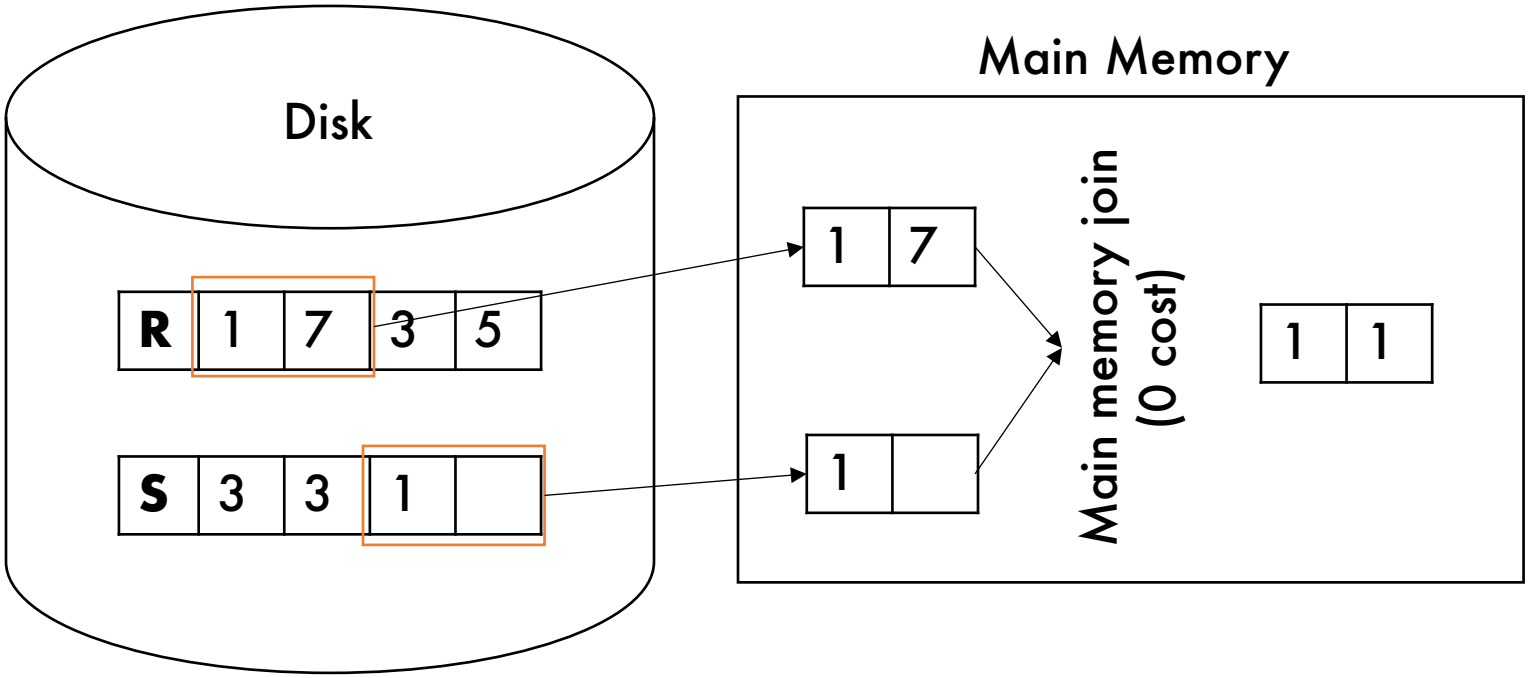
Assume block size = 2 tuples

Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

x → A tuple where x is the join attribute value



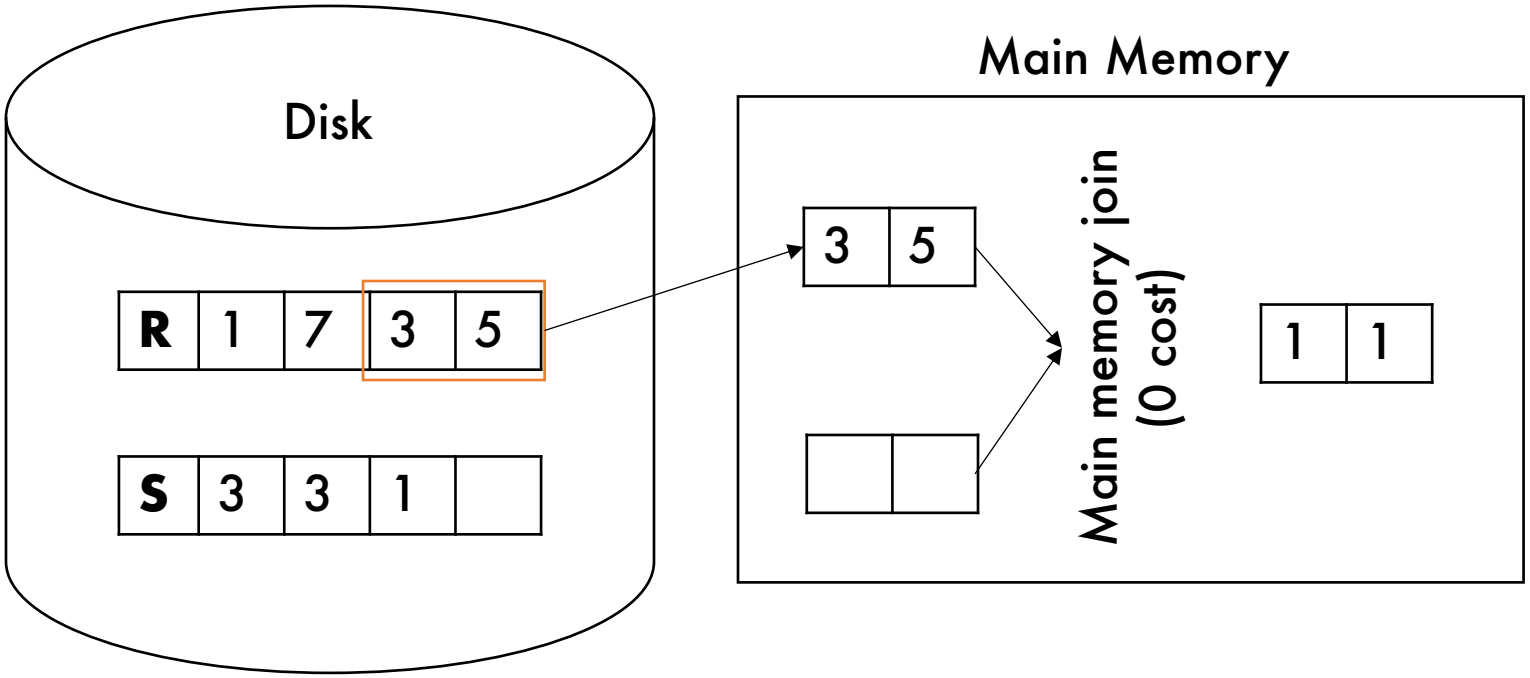
Assume block size = 2 tuples

Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

x → A tuple where x is the join attribute value

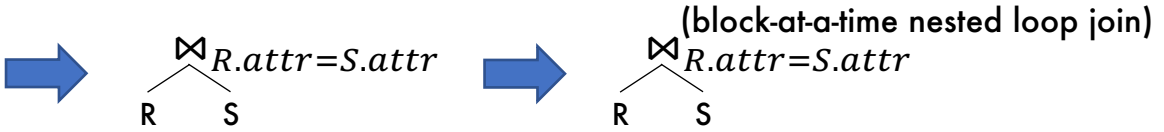


Assume block size = 2 tuples

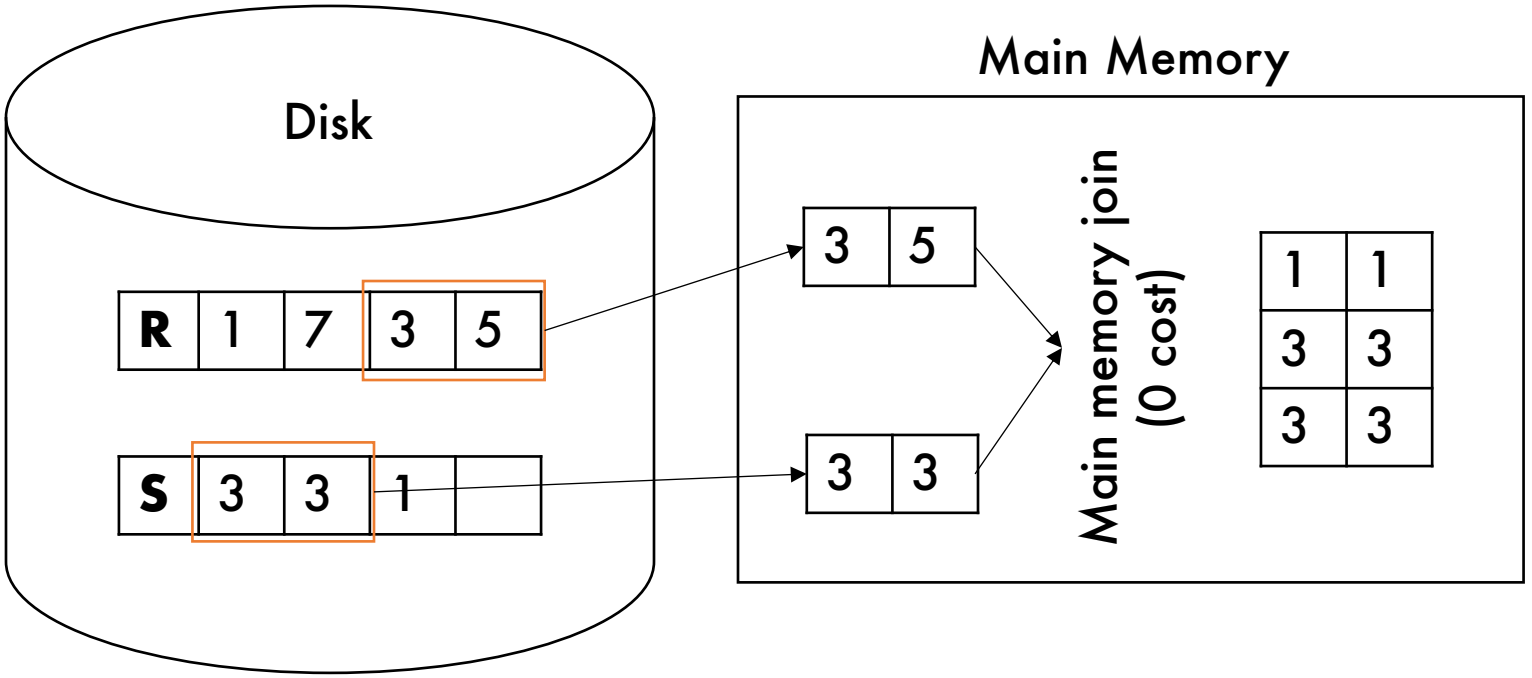
Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



x → A tuple where x is the join attribute value

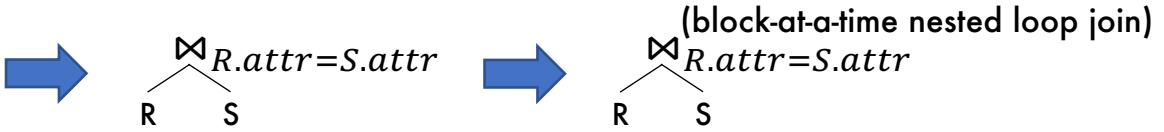


Assume block size = 2 tuples

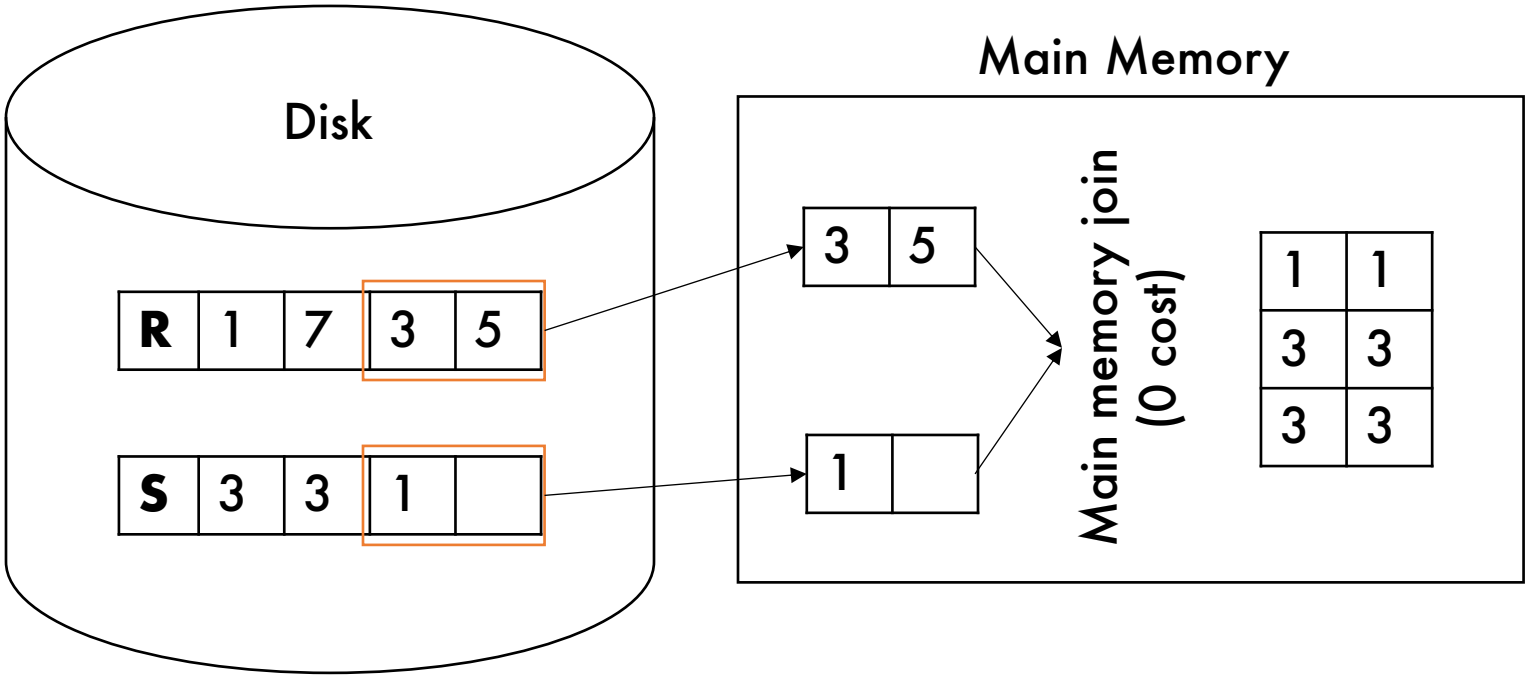
Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



x → A tuple where x is the join attribute value



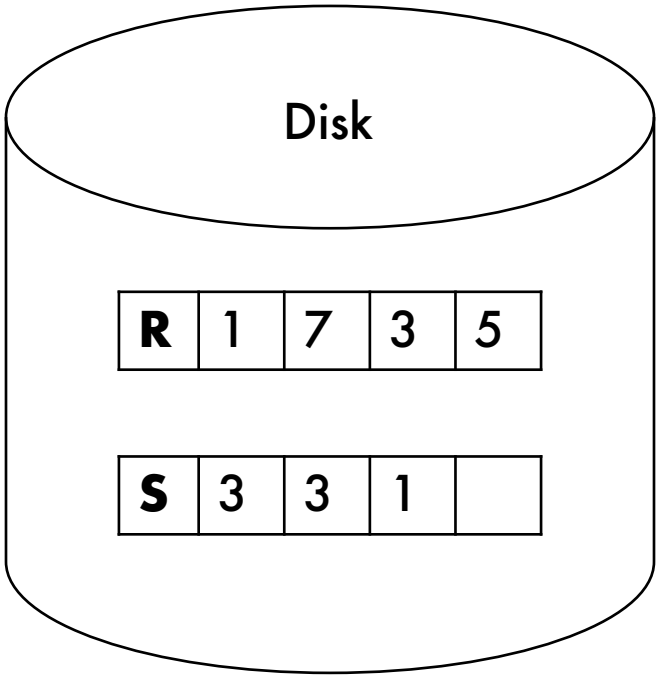
Assume block size = 2 tuples

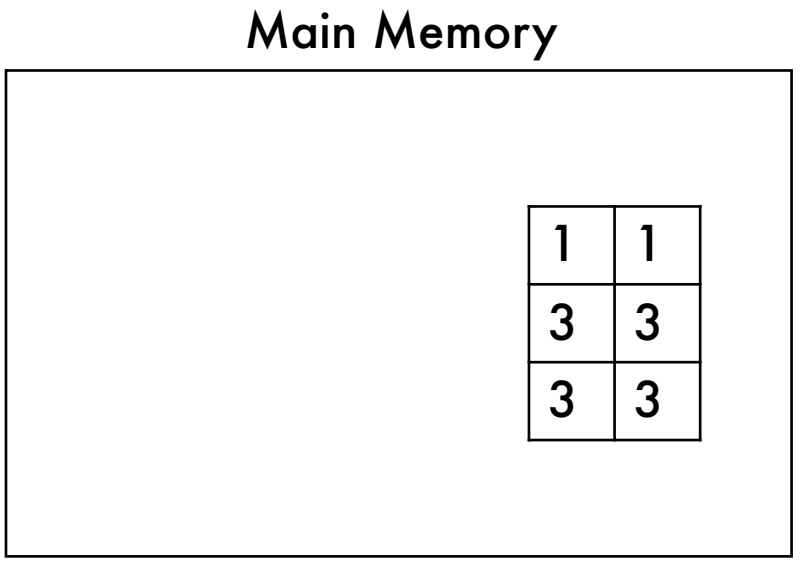
Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

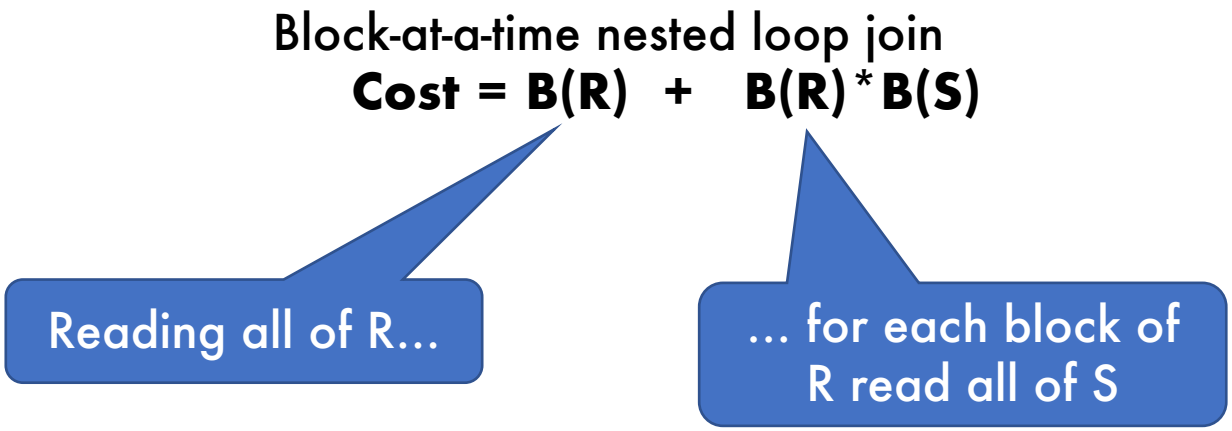
x → A tuple where x is the join attribute value





Assume block size = 2 tuples

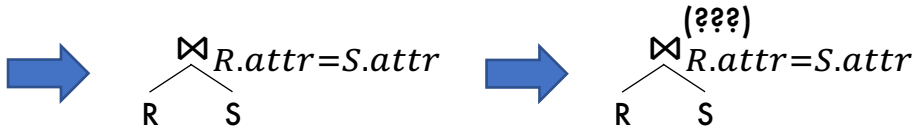
Nested Loop Join Algorithm



Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

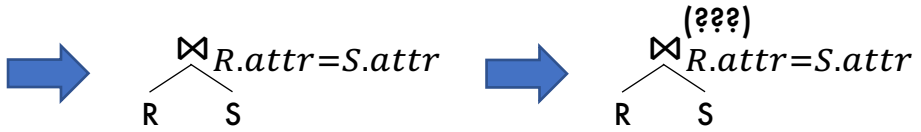


Can I do it faster?

Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



Can I do it faster?
Yes… if you're willing to use more memory

Algorithms 101:
Time complexity vs space complexity tradeoff

Optimized Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

Optimized Block-nested-loop join:

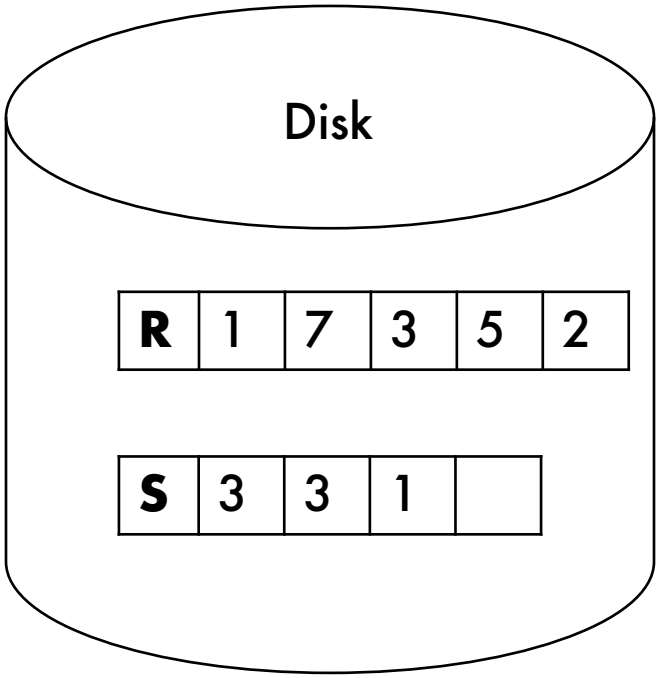
```
for each group of N blocks bR in R:
    for each block bS in S:
        for each tuple tR in bR:
            for each tuple tS in bS:
                if tR and tS can join:
                    output (tR,tS)
```

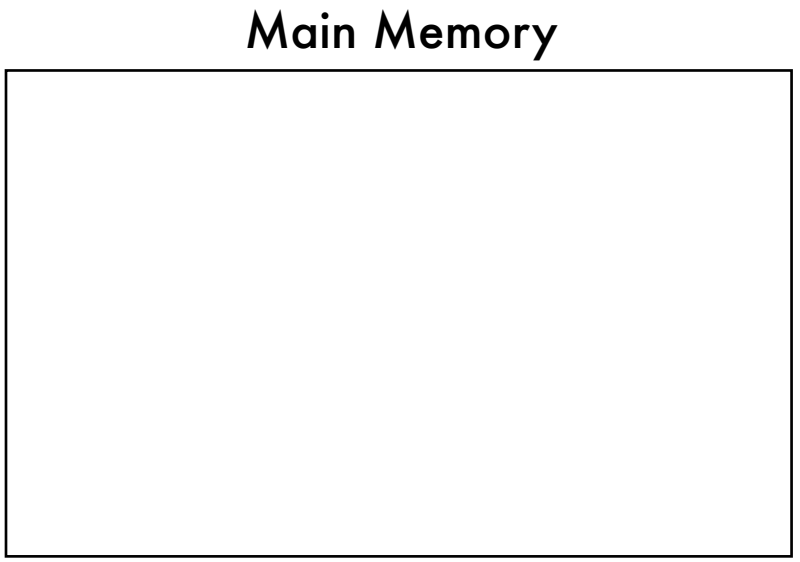
Optimized Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

N = 2 blocks at a time





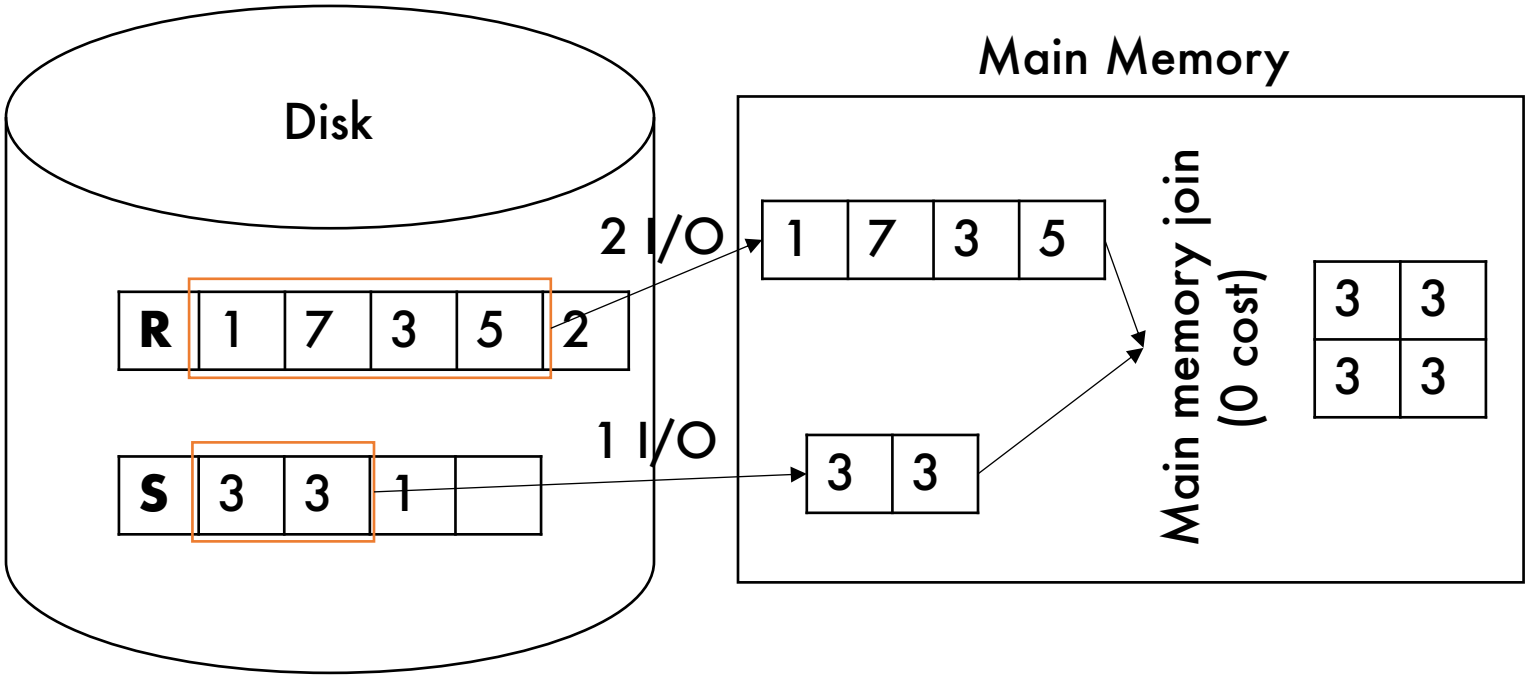
Assume block size = 2 tuples

Optimized Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

N = 2 blocks



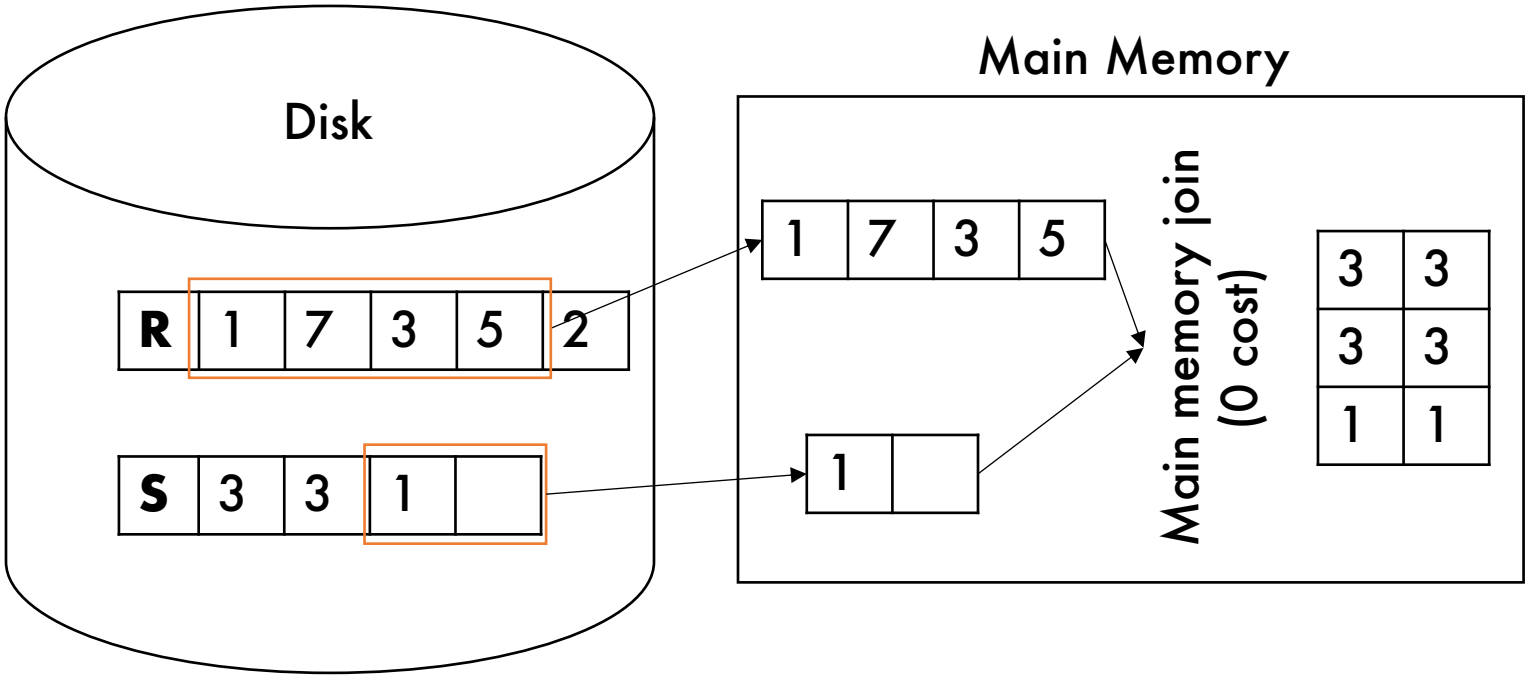
Assume block size = 2 tuples

Optimized Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

N = 2 blocks



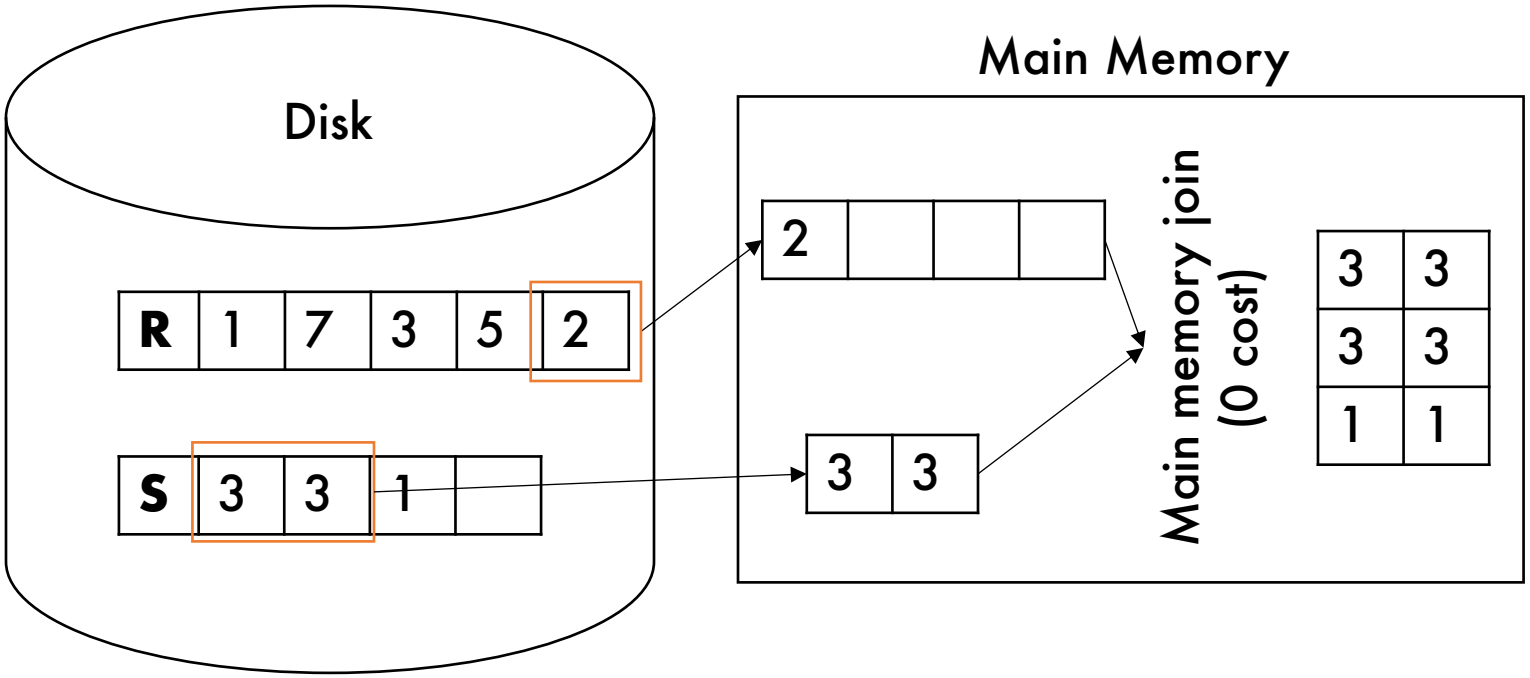
Assume block size = 2 tuples

Optimized Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

N = 2 blocks

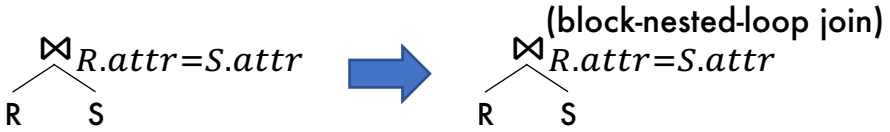


Assume block size = 2 tuples

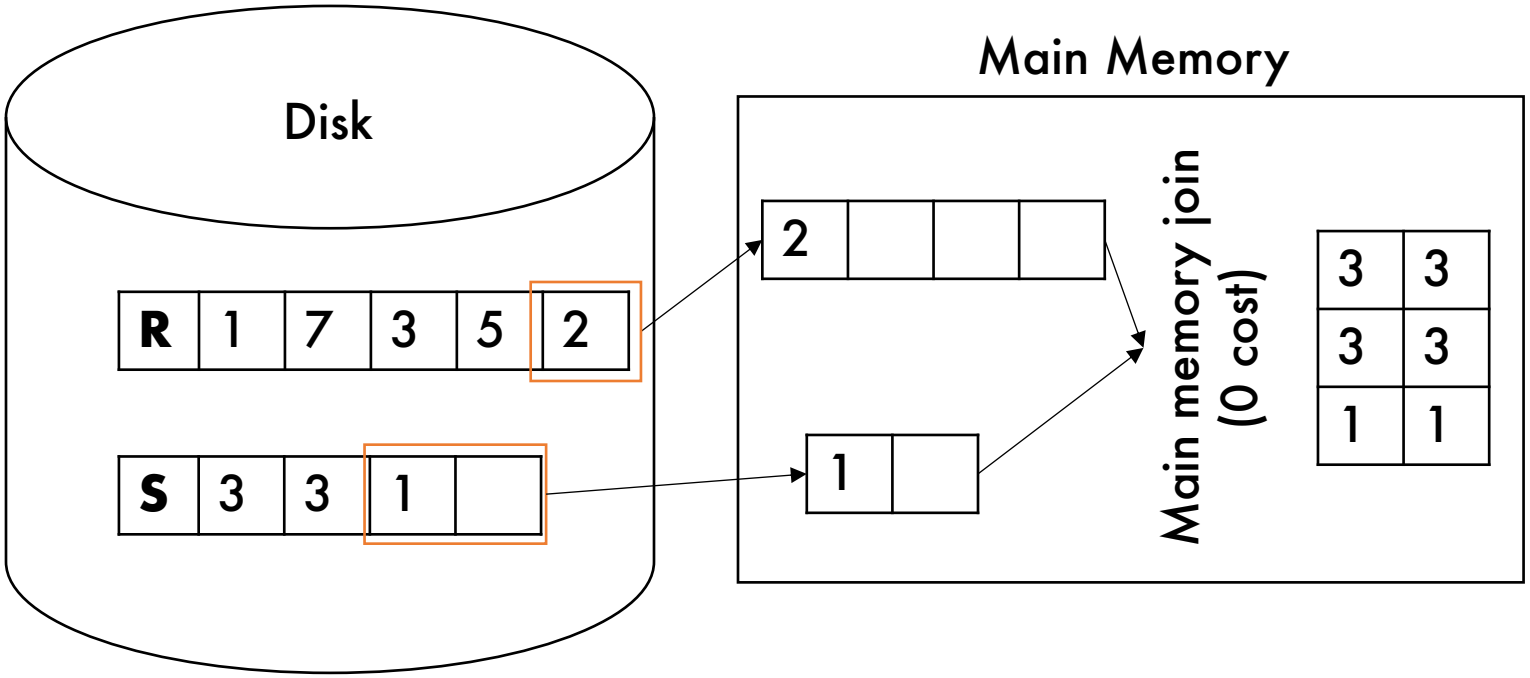
Optimized Nested Loop Join Algorithm

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

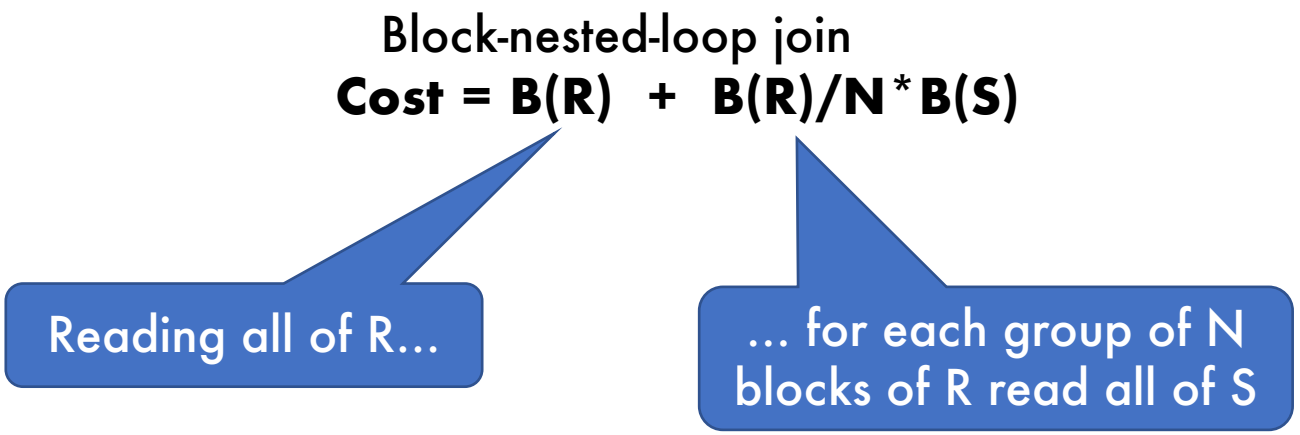


N = 2 blocks



Assume block size = 2 tuples

Optimized Nested Loop Join Algorithm



Join Algorithm Summary

- Nested-Loop Join
  - Versatile
  - Can speed up with more memory
- **Hash Join** (single pass)
  - Fast
  - Needs at least one input to be small
- Sort-Merge Join (single pass)
  - Fast
  - Sorts data at the same time!
  - Needs both inputs to be small

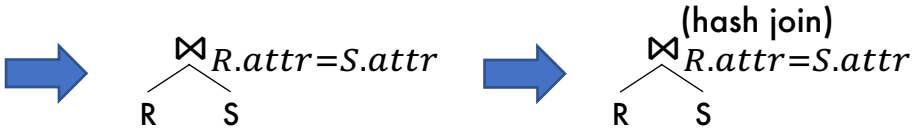
Hash Join

- Make a lookup/hash table from the smaller table
  - Smaller table has to be smaller than total main memory available (**B(R) < M or B(S) < M**)
- For each block of the larger table, join using the lookup/hash table

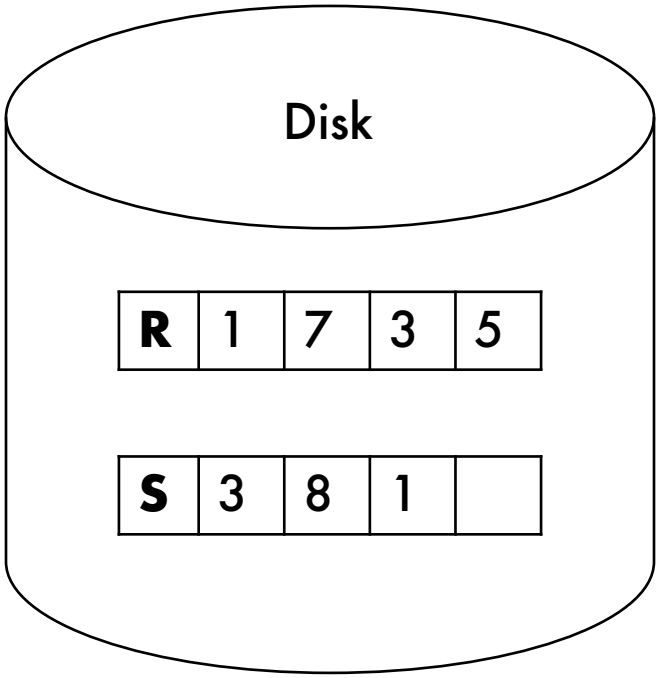
Hash Join

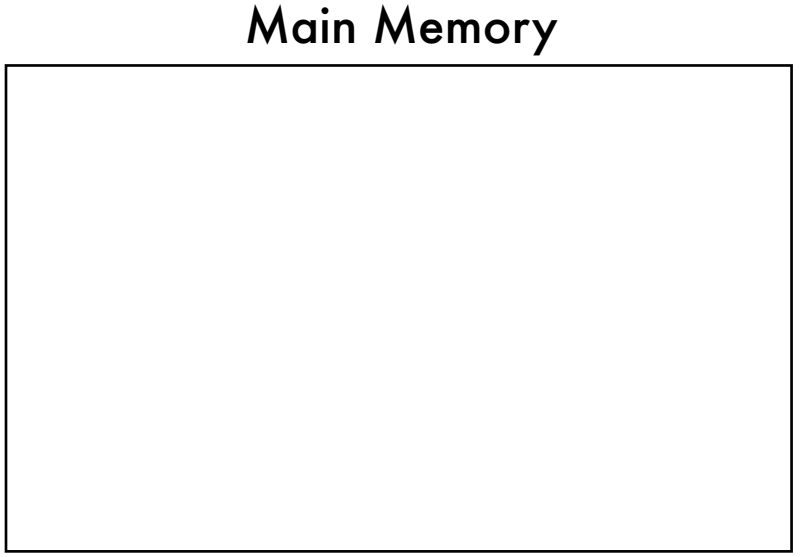
Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



M = 10 blocks, hash(x) = x mod 5



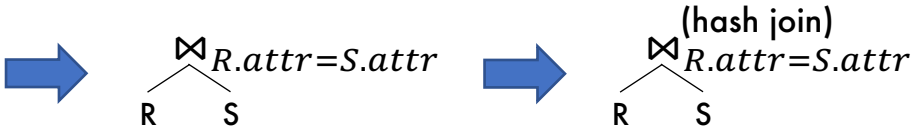


Assume block size = 2 tuples

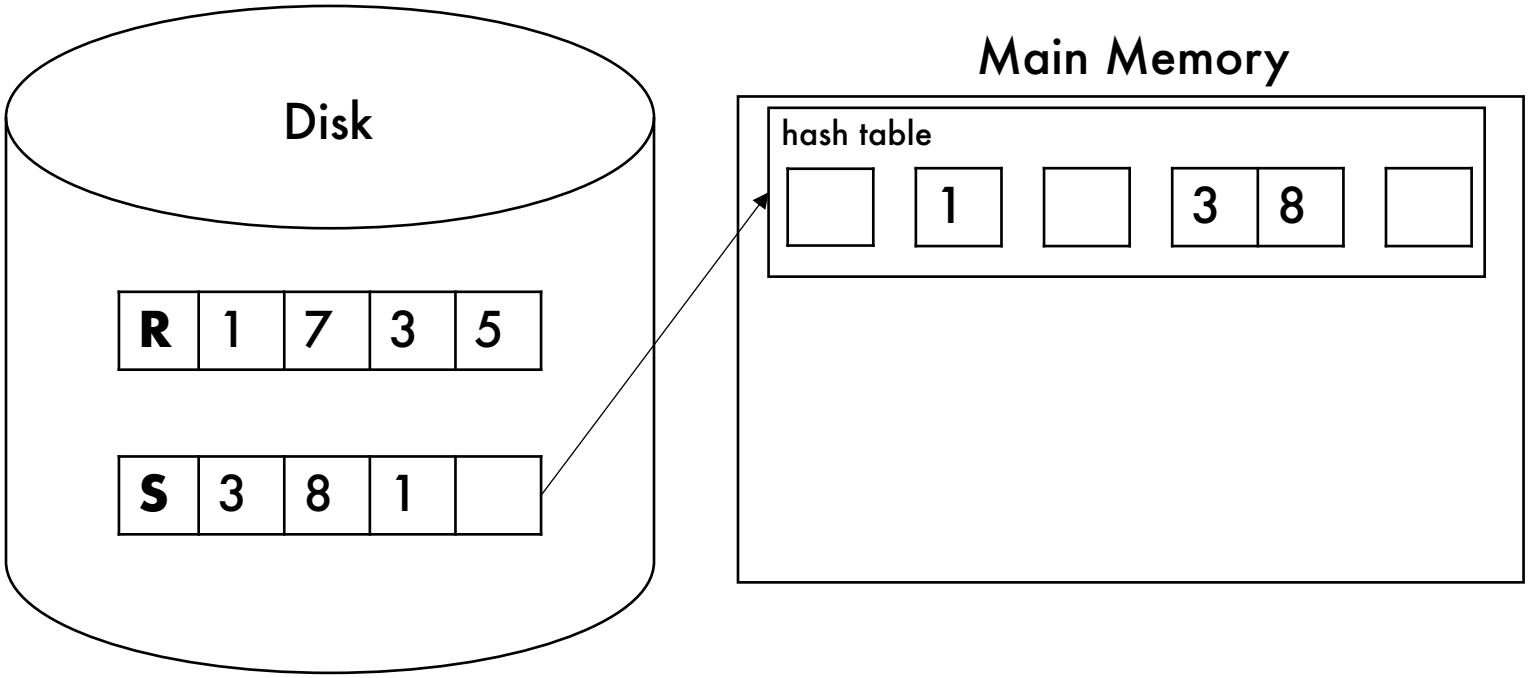
Hash Join

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



M = 10 blocks, hash(x) = x mod 5

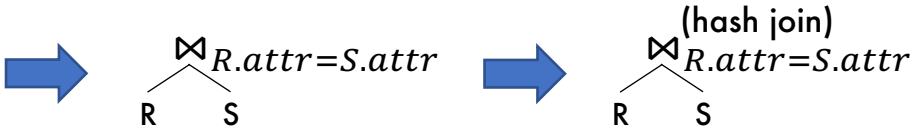


Assume block size = 2 tuples

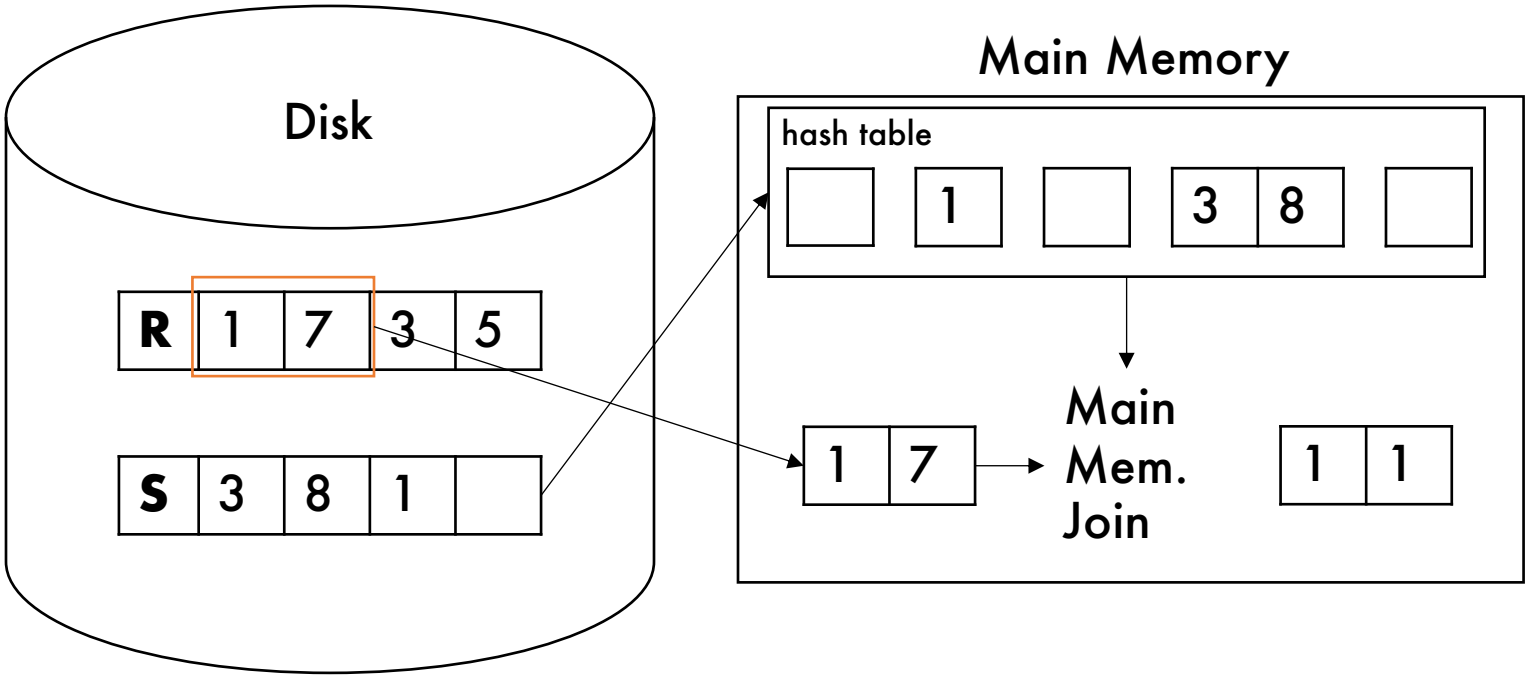
Hash Join

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



M = 10 blocks, hash(x) = x mod 5

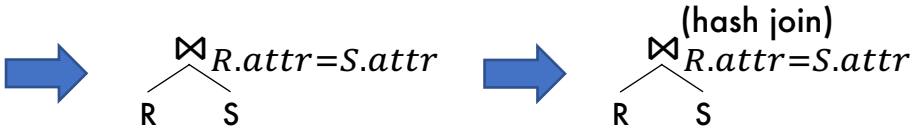


Assume block size = 2 tuples

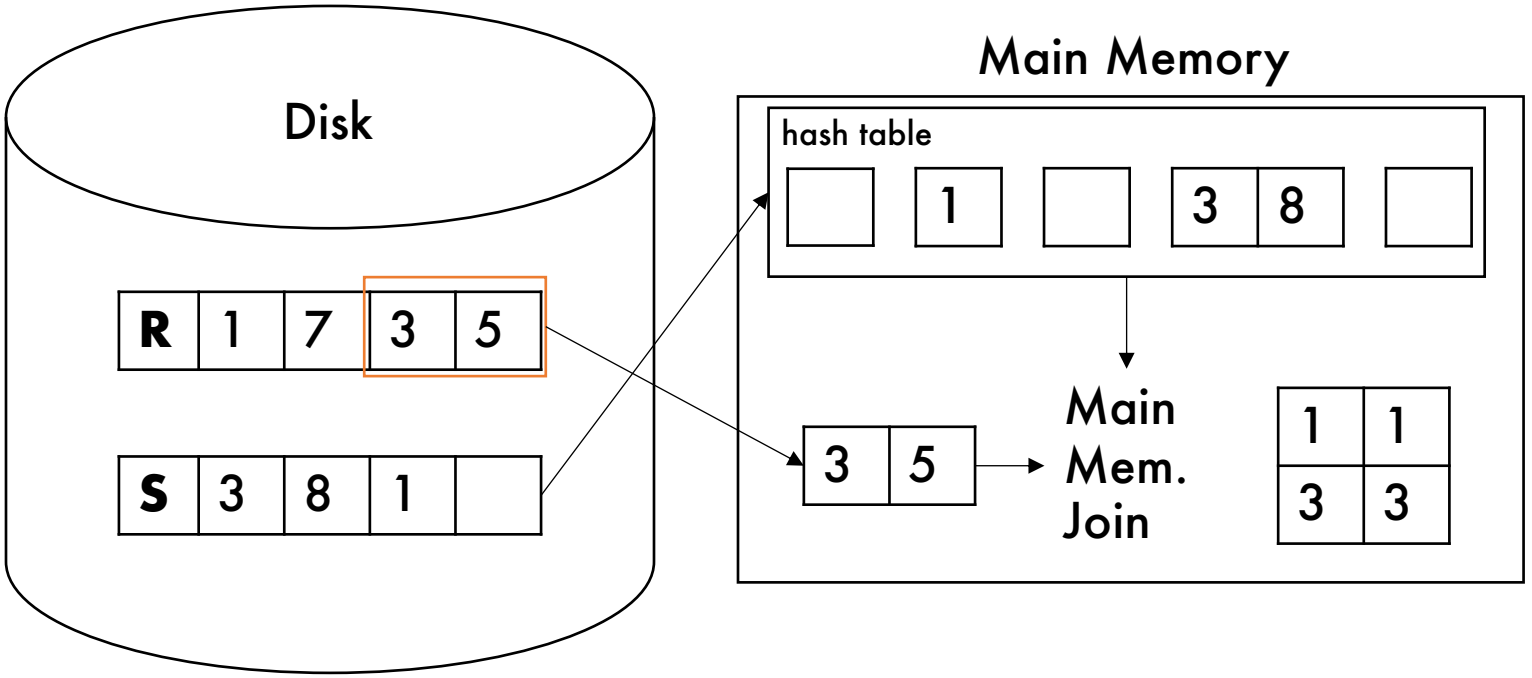
Hash Join

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

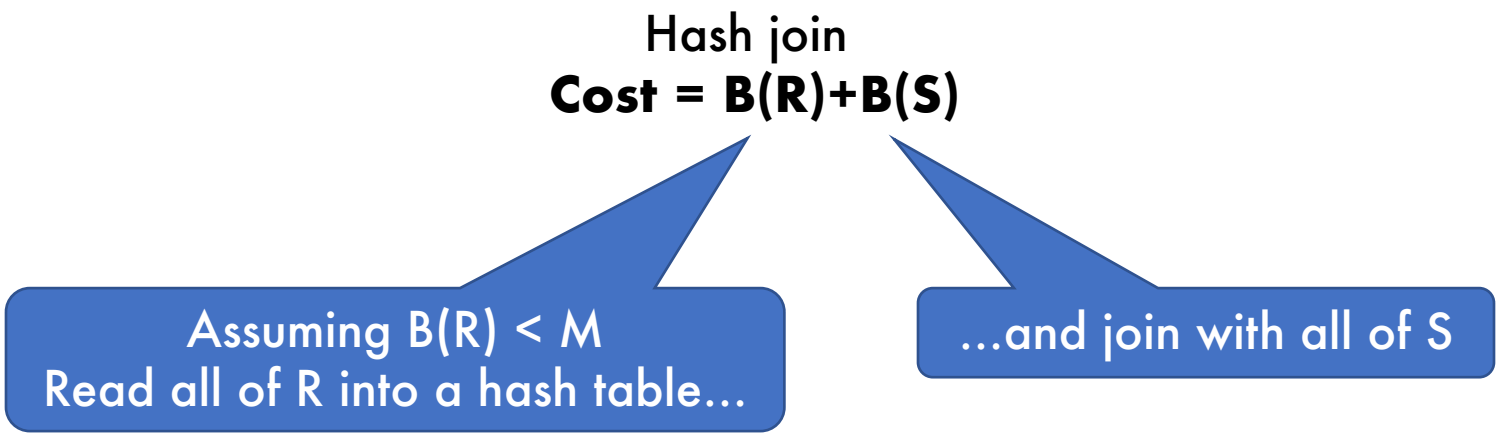


M = 10 blocks, hash(x) = x mod 5



Assume block size = 2 tuples

Hash Join


Hash join
Cost = B(R)+B(S)
Assuming B(R) < M
Read all of R into a hash table…
…and join with all of S

Hash Join

Hash join

Cost = B(R) + B(S)

Isn't this the same as block-nested-loop join where B(R)=N?

Cost = B(R) + B(R)/N * B(S)

Hash Join

Hash join

Cost = B(R) + B(S)

Isn't this the same as block-nested-loop join where B(R)=N?

Cost = B(R) + B(R)/N*B(S)

Yes!

It's the optimal "One Pass" algorithm

Join Algorithm Summary

- Nested-Loop Join
  - Versatile
- Hash Join (single pass)
  - Fast
  - Needs at least one input to be small
- **Sort-Merge Join** (single pass)
  - Fast
  - Sorts data at the same time!
  - Needs both inputs to be small

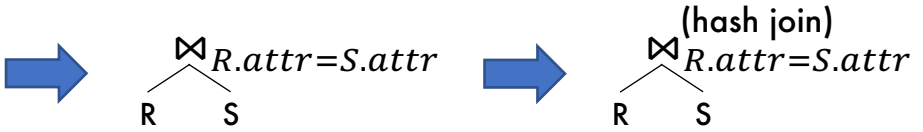
Sort-Merge Join

- Sort both tables into lists in memory
  - Since the sorted lists must contain all tuples, both tables together must fit in memory (**B(R)+B(S) < M**)
- Merge the lists in memory to join
  - Preserves order!

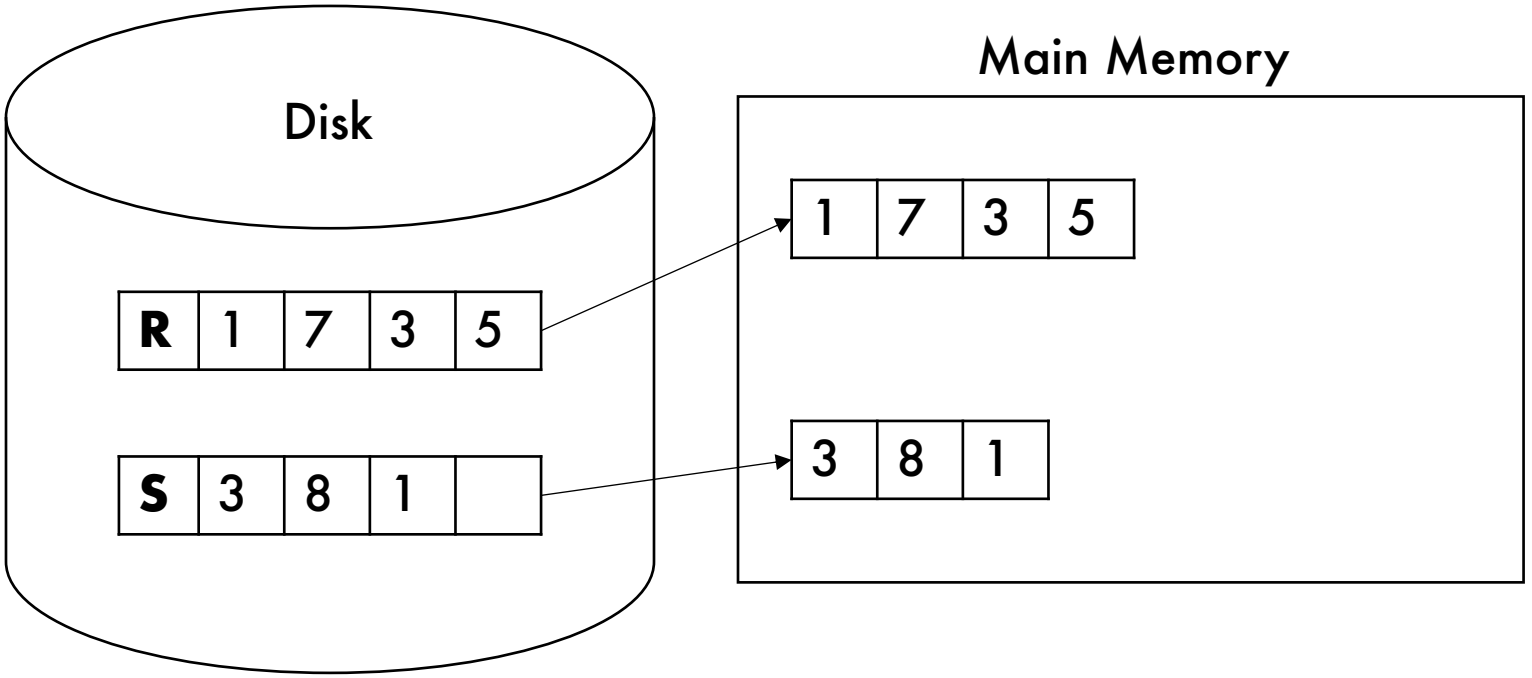
Sort-Merge Join

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



M = 10 blocks

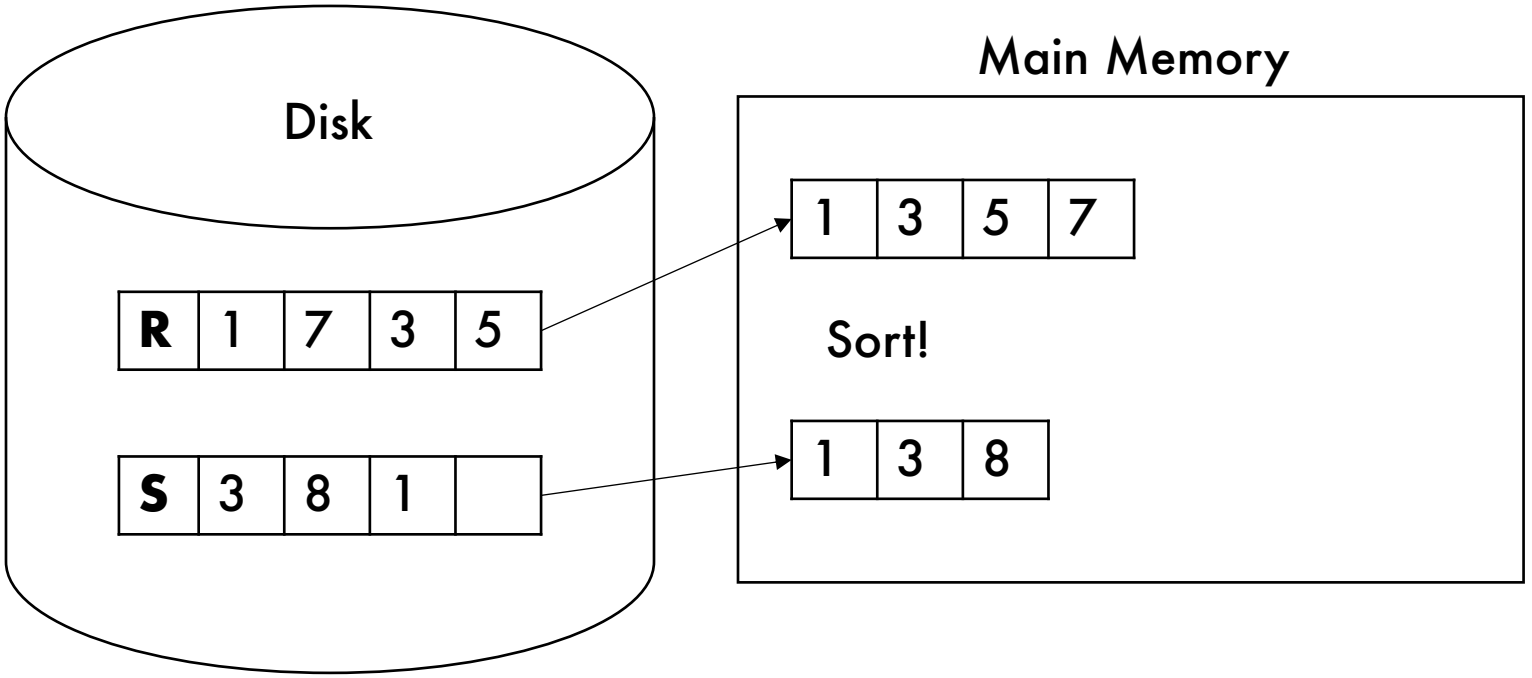


Sort-Merge Join

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```

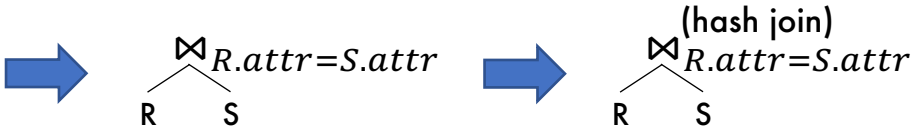
M = 10 blocks



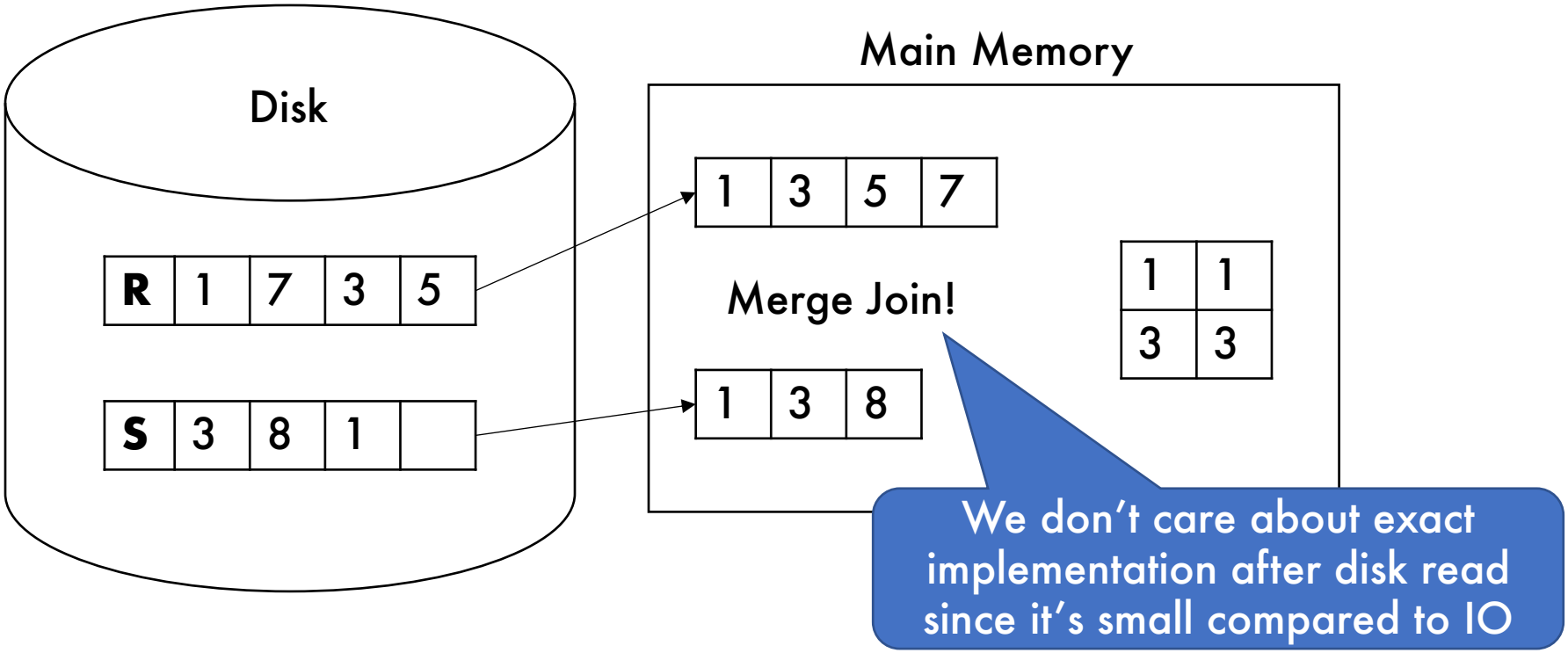
Sort-Merge Join

Example equijoin

```
SELECT *
  FROM R, S
 WHERE R.attr = S.attr
```



M = 10 blocks



Takeaways

- Nested-Loop Joins
  - Block-at-a-time → B(R) + B(R)*B(S)
  - Nested-block-loop → B(R) + B(R)/N*B(S)
- Hash Join and Sort-Merge Join → B(R) + B(S)

Next Time

- Comparing join algorithms to algorithms with index structures to help
- Plan pruning