Introduction to Data Structure (Data Management) Lecture 14

(Ch. 15.{1,3,4.6,6}, 16.4-5)

Felipe P. Vista IV

Reminder

- Everybody, make sure that your name in ZOOM is in the following format:
  - University ID Num Name (no “( )“)
  - Ex: 202054321 Juan Dela Cruz
  -
  - Not changing your name to this format
    - you might be marked Absent

    * → absent?

- Query Optimization Basics
- Cost of reading from disk
- Cost of single RA operators
- Cost of query plans

INTRO TO DATA STRUCTURE

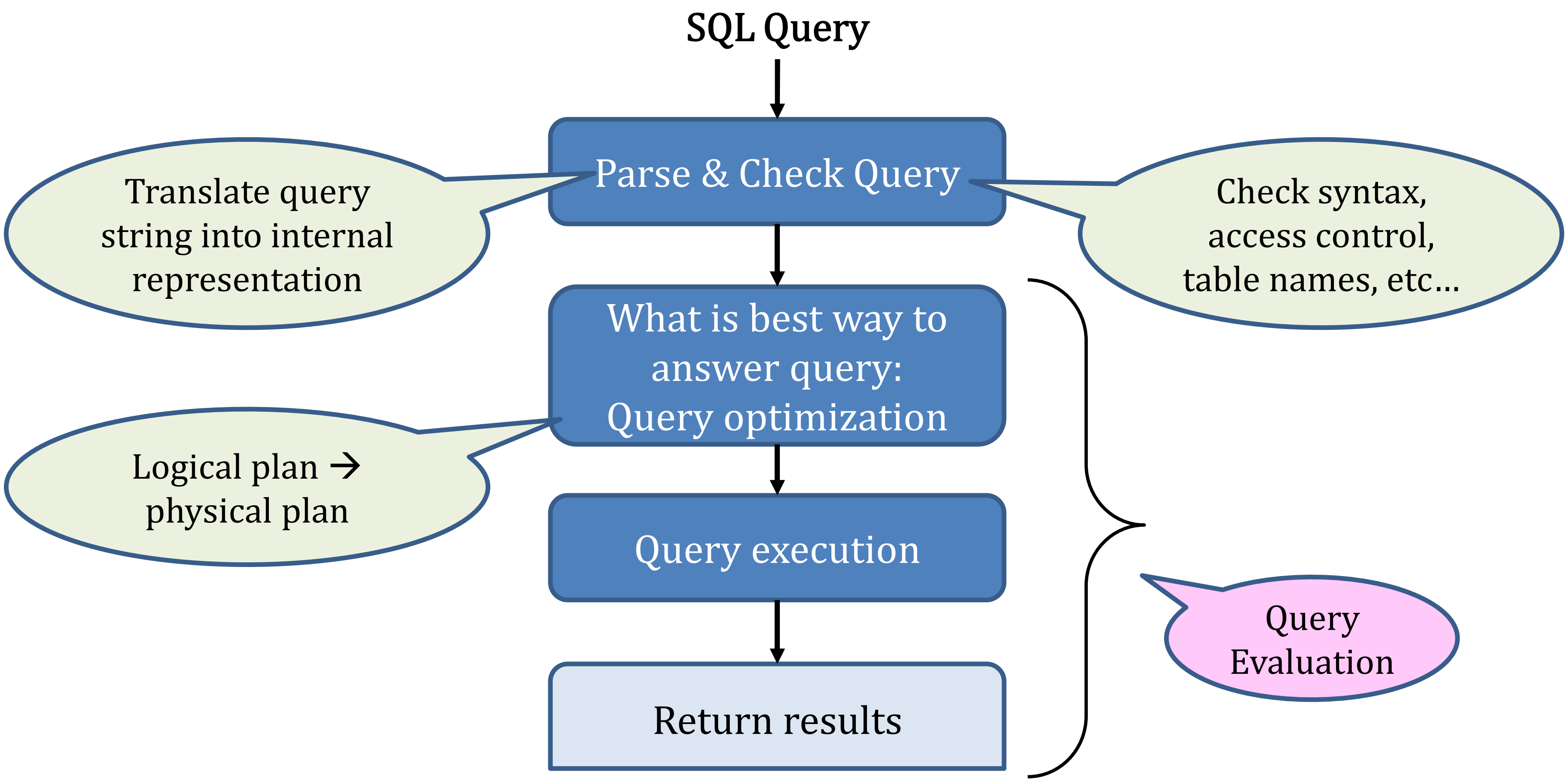
Query Optimization Basics

Motivation

- To understand performance, we need to understand a bit about how DBMS works:
  - the database application is too slow… why?
  - one of the queries is very slow… why?

- Under our direct control: index choice
  - understand how that affects query performance

Recap: Query Evaluation Steps



Query Optimizer Overview

- **Input**: Parsed & checked SQL
- **Output**: A good physical query plan
- **Basic query optimization algorithm**:
  - Enumerate alternative plans (logical and physical)
  - Compute estimated cost of each plan
    - Compute number of I/Os
    - Optionally take into account other resources
  - Choose plan with lowest cost
  - This is called cost-based optimization

Query Optimizer Overview

- There are exponentially many query plans
  - exponential in the size of the query
  - simple SFW with 3 joins does not have too many
- Optimizer will consider many, many of them
- Worth substantial cost to **avoid bad plans**

INTRO TO DATA STRUCTURE

Cost of Reading Data from Disk

Cost Parameters (Statistics)

- Cost = Disk I/O + CPU + Network I/O
  - focus on Disk I/O

* Why use Disk I/O ?
- We assume that it takes longer to get data from the disk that to do anything useful with it once the data is in the main memory.
- We assume that the arguments of any operator are found on disk, but the result of the operator is left in main memory.

Cost Parameters (Statistics)

- Cost = Disk I/O + CPU + Network I/O
  - focus on Disk I/O

* Why use Disk I/O ?
- We assume that it takes longer to get data from the disk that to do anything useful with it once the data is in the main memory.
- We assume that the arguments of any operator are found on disk, but the result of the operator is left in main memory.

- Parameters:
  - $B(R)$ = # of blocks (i.e., pages) for relation R
  - $T(R)$ = # of tuples in relation R
  - $V(R, A)$ = # of distinct values of attribute(column) A
    - When A is a key, $V(R, A) = T(R)$
    - When A is not a key, $V(R, A)$ can be anything $< T(R)$

S+N.

Cost Parameters (Statistics)

- Cost = Disk I/O + CPU + Network I/O
  - focus on Disk I/O

* Why use Disk I/O ?
- We assume that it takes longer to get data from the disk that to do anything useful with it once the data is in the main memory.
- We assume that the arguments of any operator are found on disk, but the result of the operator is left in main memory.

- Parameters:
  - B(R) = # of blocks (i.e., pages) for relation R
  - T(R) = # of tuples in relation R
  - V(R, A) = # of distinct values of attribute(column) A
    - When A is a key, V(R, A) = T(R)
    - When A is not a key, V(R, A) can be anything < T(R)
- Where do these values come from?
  - DBMS collects statistics about data on disk

Selectivity Factors for Conditions

- $A = c$ /* $\sigma_{A=c}(R)$ */
  - Selectivity = $1/V(R, A)$

- $A < c$ /* $\sigma_{A<c}(R)$*/
  - Selectivity = $(c - Low(R, A))/(High(R, A) - Low(R, A))$

- $c1 < A < c2$ /* $\sigma_{c1<A<c2}(R)$*/
  - Selectivity = $(c2 – c1)/(High(R, A) - Low(R, A))$

* Selectivity factor - defined as the ratio of output to input tuples
- quality of a filter in its ability to reduce the number of rows that will need to be examined and ultimately returned
- ratio between the number of values in a column, the COUNT, and the number of values that are distinct or unique

Example: Selectivity of $\sigma_{A=c}(R)$

```
T(R) = 100,000
V(R, A) = 20
```

How many records are returned by $\sigma_{A=c}(R)$= ?

Answer:

Number of records returned =

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Example: Selectivity of $\sigma_{A=c}(R)$

```
T(R) = 100,000
V(R, A) = 20
```

How many records are returned by $\sigma_{A=c}(R)$ = ?

Answer: $X * T(R)$, where X = selectivity…

… $X = 1/V(R,A) = 1/20$

$\approx \frac{T(R)}{V(R,A)}$

Number of records returned = 100,000/20 = 5,000

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Cost of Index-based Selection

- Sequential scan for relation R costs B(R)
- Index-based selection
  - Estimate selectivity factor **X** (see previous slide)
  - Clustered index: X*B(R) ; X = 1 · B(R) = BR
  - Unclustered index: X*T(R)

Note: we are ignoring I/O cost for index pages

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R,A)=T(R), V(R,A) < T(R))*

Example: Cost of $\sigma_{A=c}(R)$

- Example

```
B(R)    = 2000
T(R)    = 100,000
V(R, A) = 20
```

Cost of $\sigma_{A=c}(R)$=?

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Example: Cost of $\sigma_{A=c}(R)$

- Example

```
B(R)    = 2000
T(R)    = 100,000
V(R, A) = 20
```

Cost of $\sigma_{A=c}(R)$=?

- Table scan: B(R) = 2,000 I/Os

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Example: Cost of $\sigma_{A=c}(R)$

- Example

B(R) = 2000
T(R) = 100,000
V(R, A) = 20

Cost of $\sigma_{A=c}(R)$=?

$x = \frac{1}{V(R, A)}$

- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R)/V(R, A) = 100 I/Os
  - If index is unclustered: T(R)/V(R, A) = 5,000 I/Os

500

Lesson: Don't build unclustered indexes when V(R, A) is small !

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

INTRO TO DATA STRUCTURE

Cost of Executing Operators

(With focus on Joins)

Outline

- Join operator algorithms
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)

- Note about readings:
  - In class, we discuss only algorithms for joins
  - Other operators are easier: read the book

Join Algorithms

- HASH join
- NESTED LOOP join
- SORT-MERGE join

Hash Join

- Hash join: R ⋈ S
  - Scan R, build buckets in main memory
  - Then scan S and join
  - Cost: B(R) + B(S)

- One-pass algorithm when B(R) ≤ M (memory size)
  - more disk access also when B(R) > M

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Hash Join Example

```
Patient(pid, name, address)
Insurance(pid, provider, policy_nb)
Patient ⋈ Insurance
```

Patient

1	'Divan'	'Haga'
2	'Matt'	'Seongchon'
3	'Khan'	'Ajung'
4	'Soheil'	'Haga'

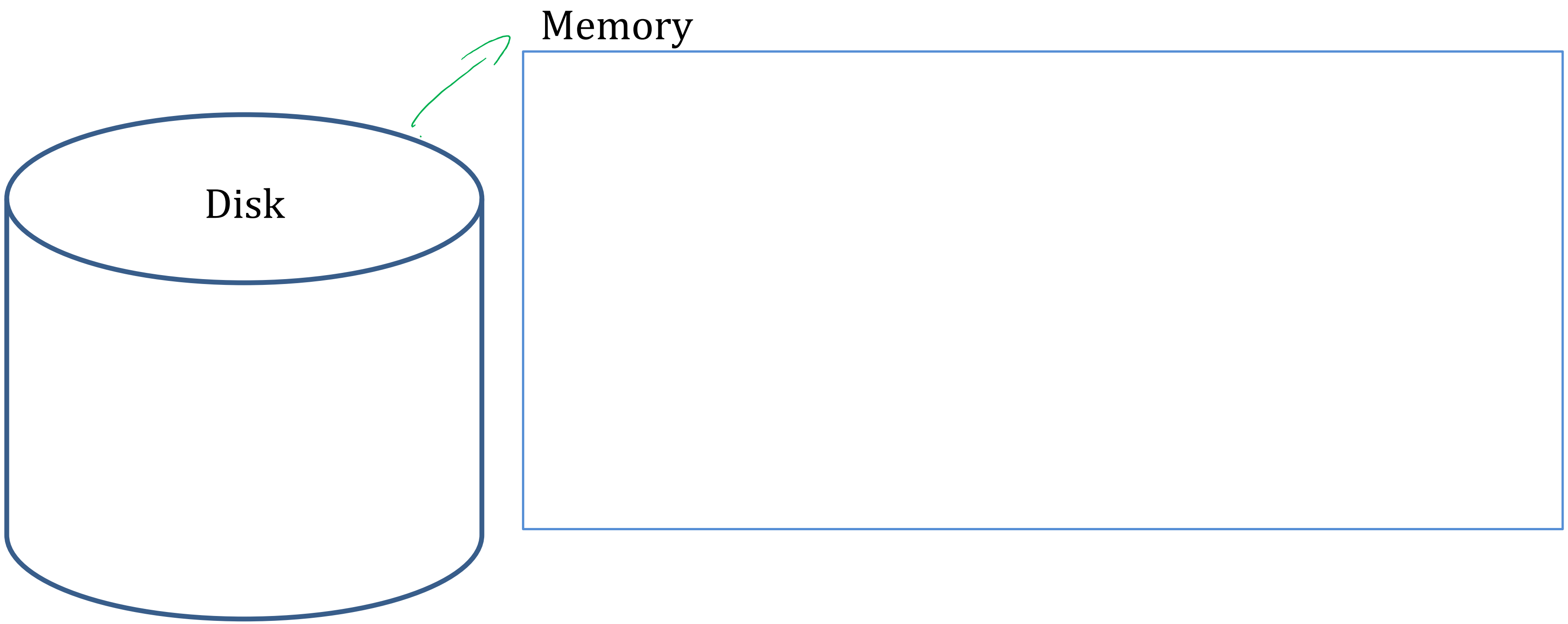
Insurance

2	'Blue'	987
4	'Prem'	654
4	'Prem'	243
3	'Group'	589

Two tuples per page

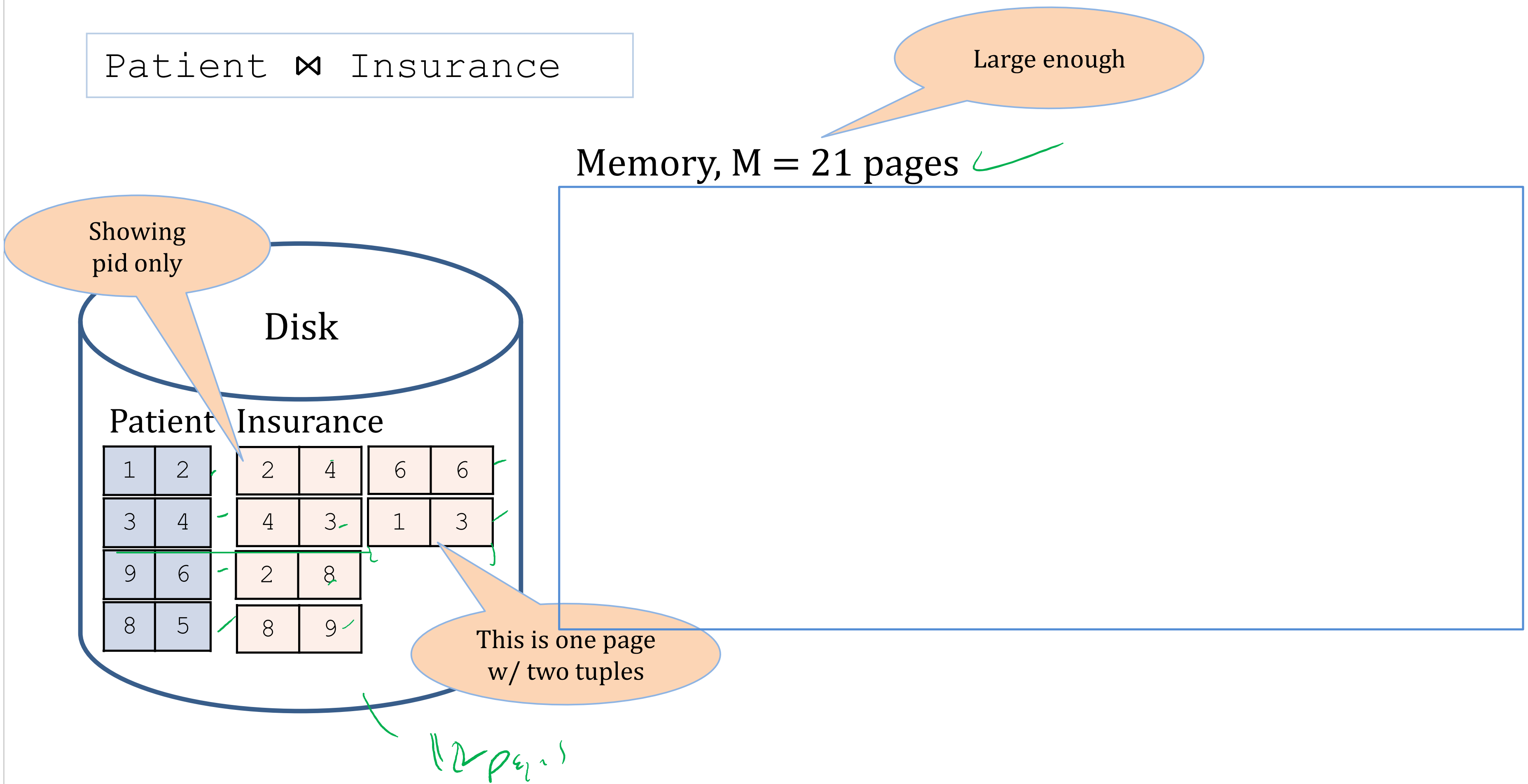
Hash Join Example

Patient ⋈ Insurance



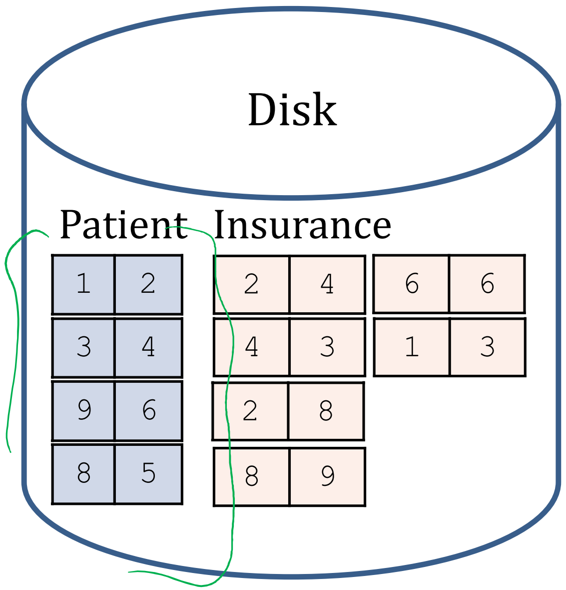
Hash Join Example

`Patient ⋈ Insurance`



Hash Join Example

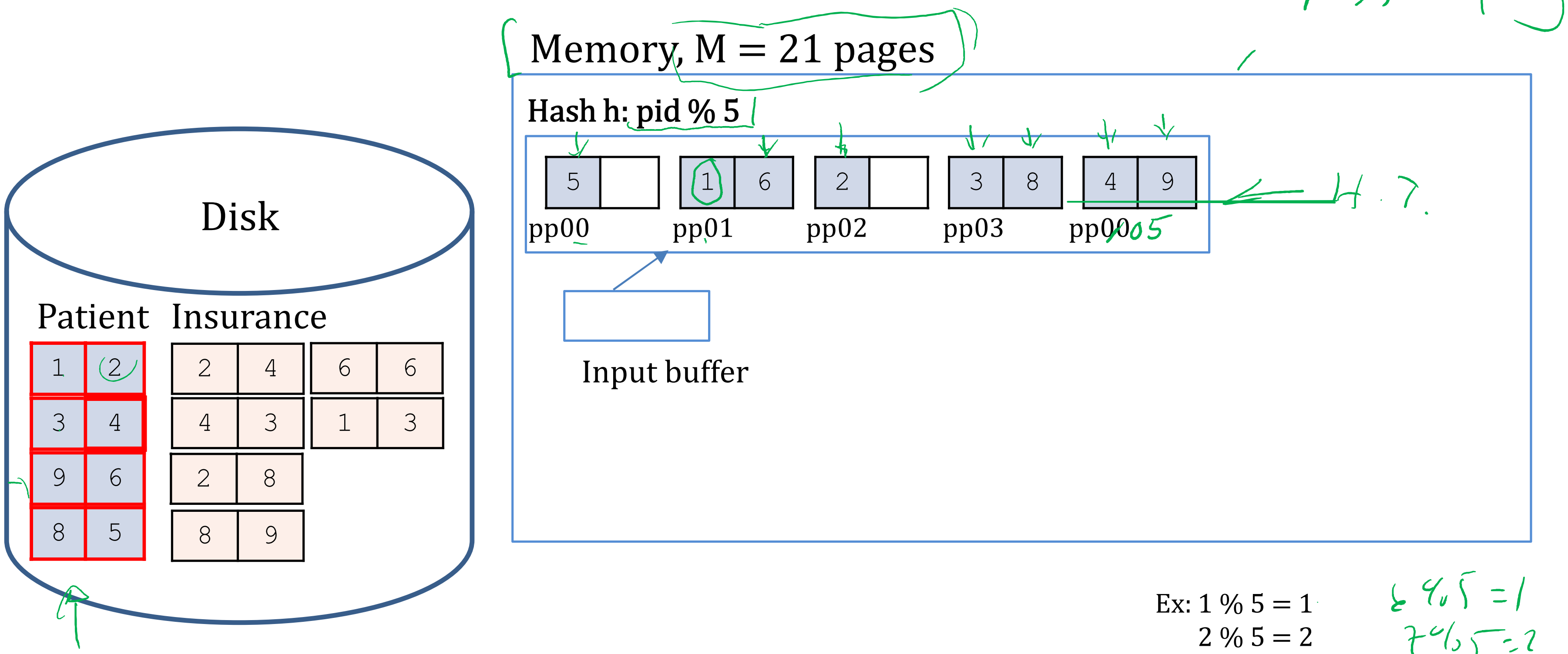
Step 1: Scan Patient and build hash in table in memory



Memory, M = 21 pages

Hash Join Example

Step 1: Scan Patient and build hash in table in memory

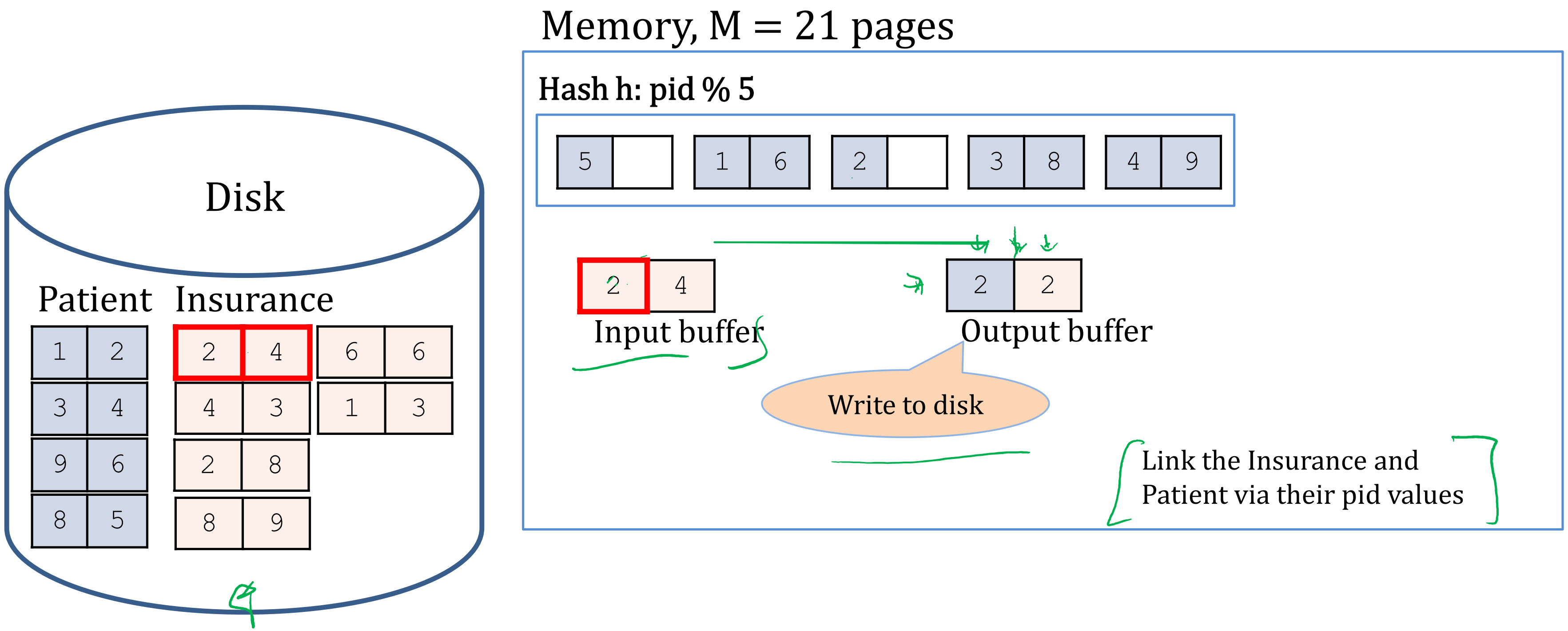


Ex: 1 % 5 = 1
2 % 5 = 2
3 % 5 = 3
5 % 5 = 0

6 % 5 = 1
7 % 5 = 2

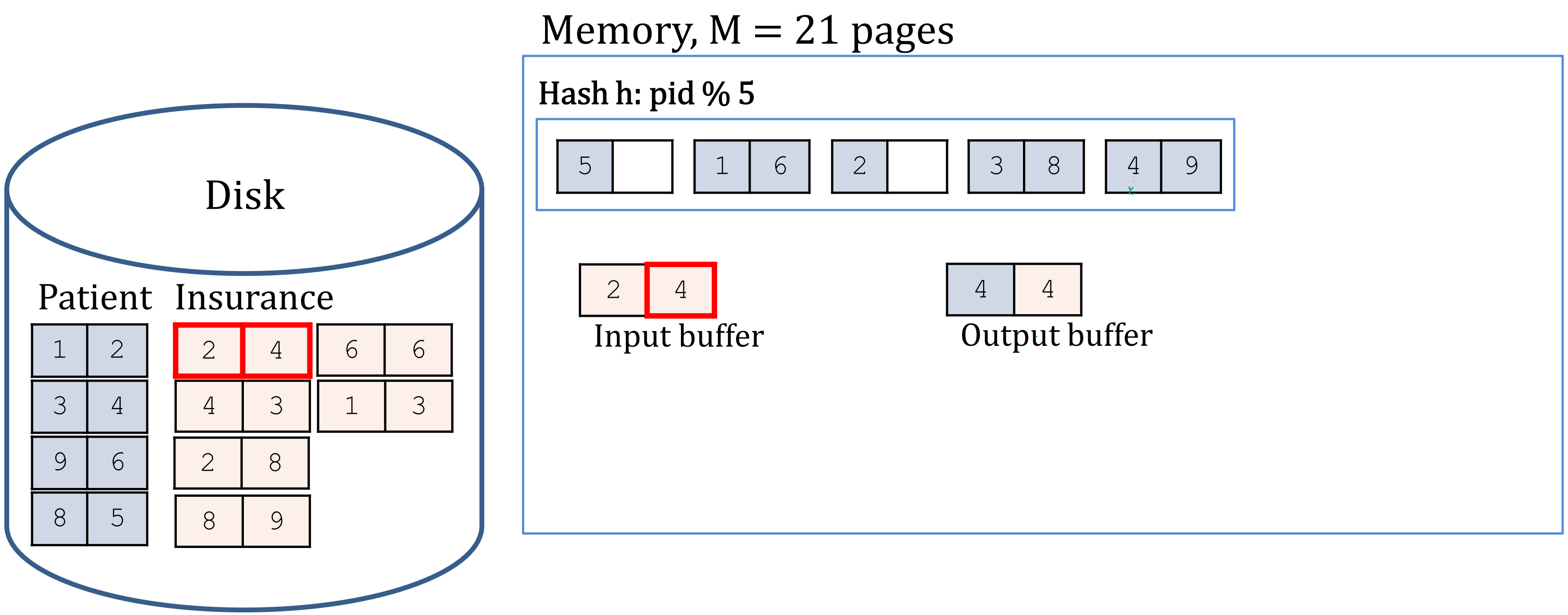
Hash Join Example

Step 2: Scan Insurance and probe into hash table



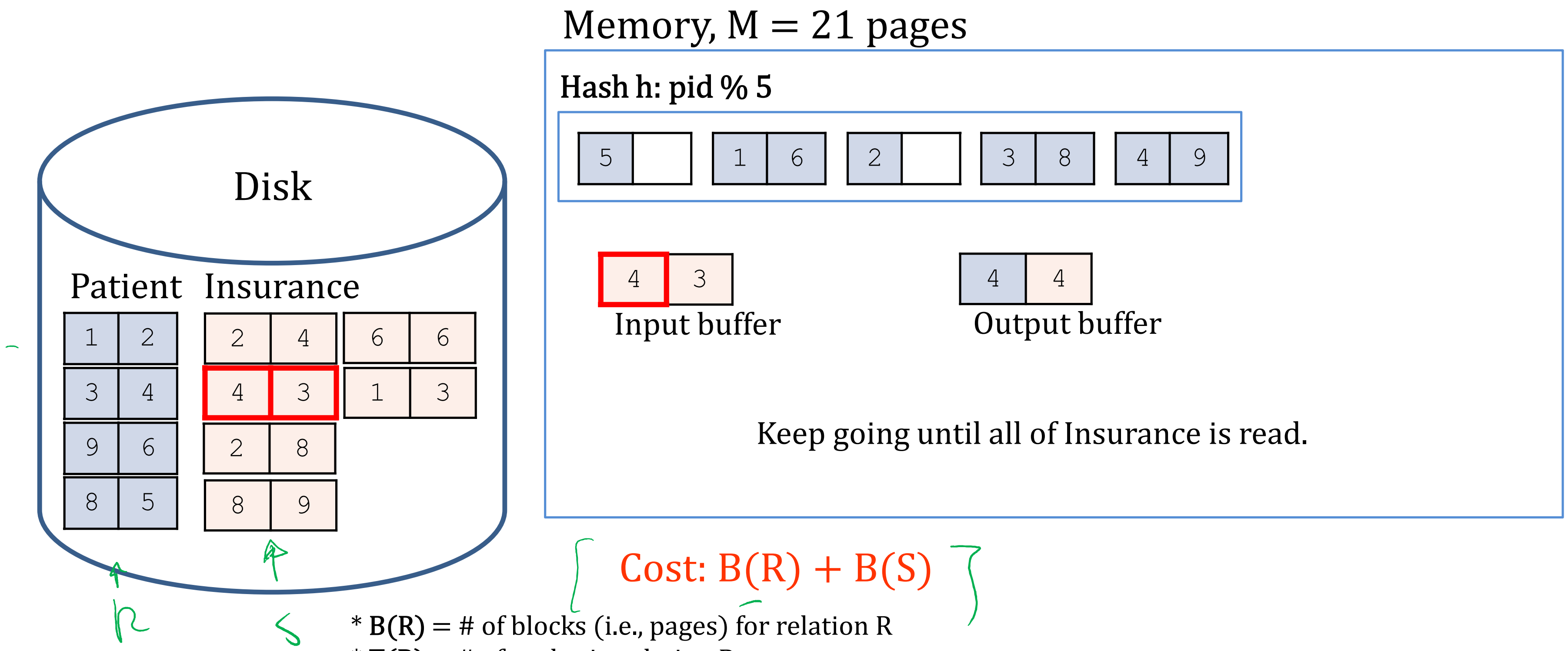
Hash Join Example

Step 2: Scan Insurance and probe into hash table



Hash Join Example

Step 2: Scan Insurance and probe into hash table



Cost: B(R) + B(S)

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$
- R is the outer relation, S is the inner relation

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

What is the Cost?

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$
- R is the outer relation, S is the inner relation

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

R1 $S_1 - S_n$ R2 $S_i - S_n$ R n $S_1 - S_n$

What is the Cost?

- Cost: $B(R) + T(R)B(S)$
- Multiple-pass because S is read many times

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R,A)=T(R), V(R,A) < T(R))*

Block-at-a-time Refinement

$\boxed{21:}$

```
for each block of tuples r in R do
  for each block of tuples s in S do
    for all pairs of tuples t1 in r, t2 in s
      if t1 and t2 join then output (t1, t2)
```

$C = B(R) + T(S)\,B(S)$

What is the Cost?

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Block-at-a-time Refinement

```
for each block of tuples r in R do
  for each block of tuples s in S do
    for all pairs of tuples t1 in r, t2 in s
      if t1 and t2 join then output (t1, t2)
```

What is the Cost?

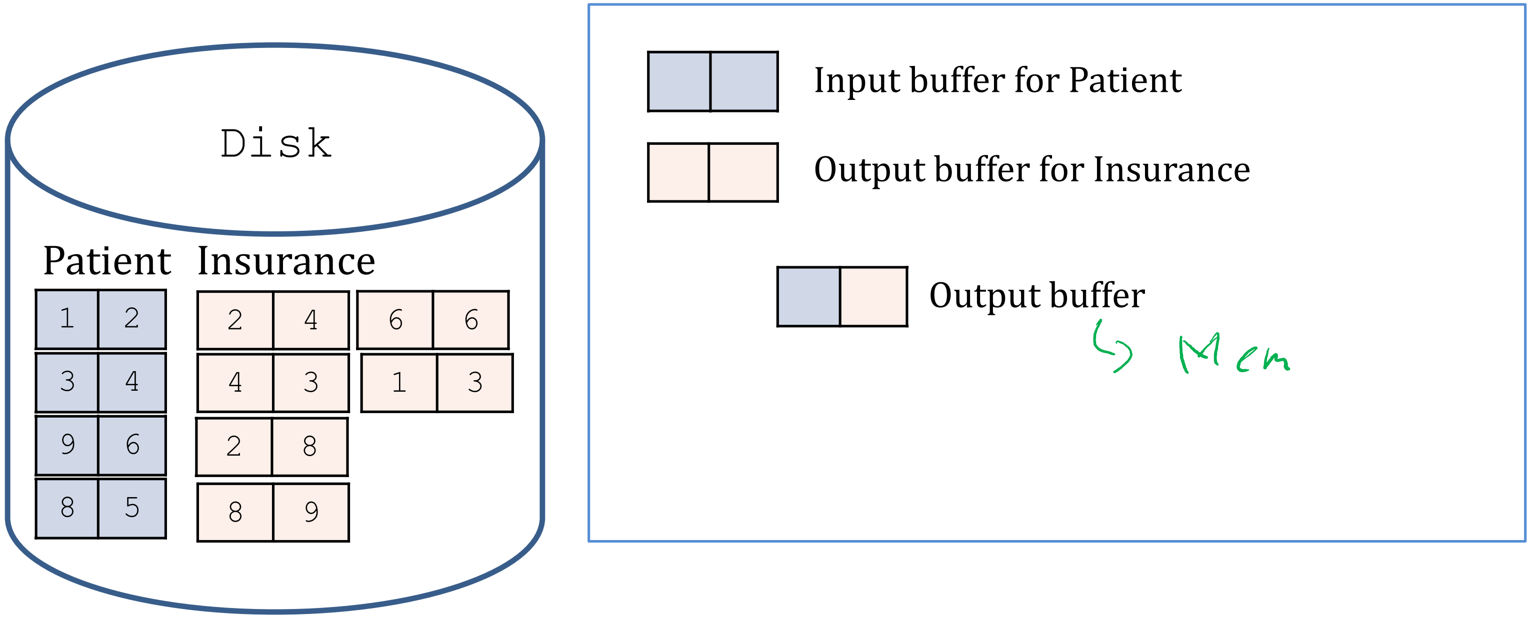
$B(R) \ll T(R)$

$R(R) + T(R)\,B(S)$

- Cost: B(R) + B(R)B(S)
  - Cost: B(R) + T(R)B(S) -> Nested Loop Join

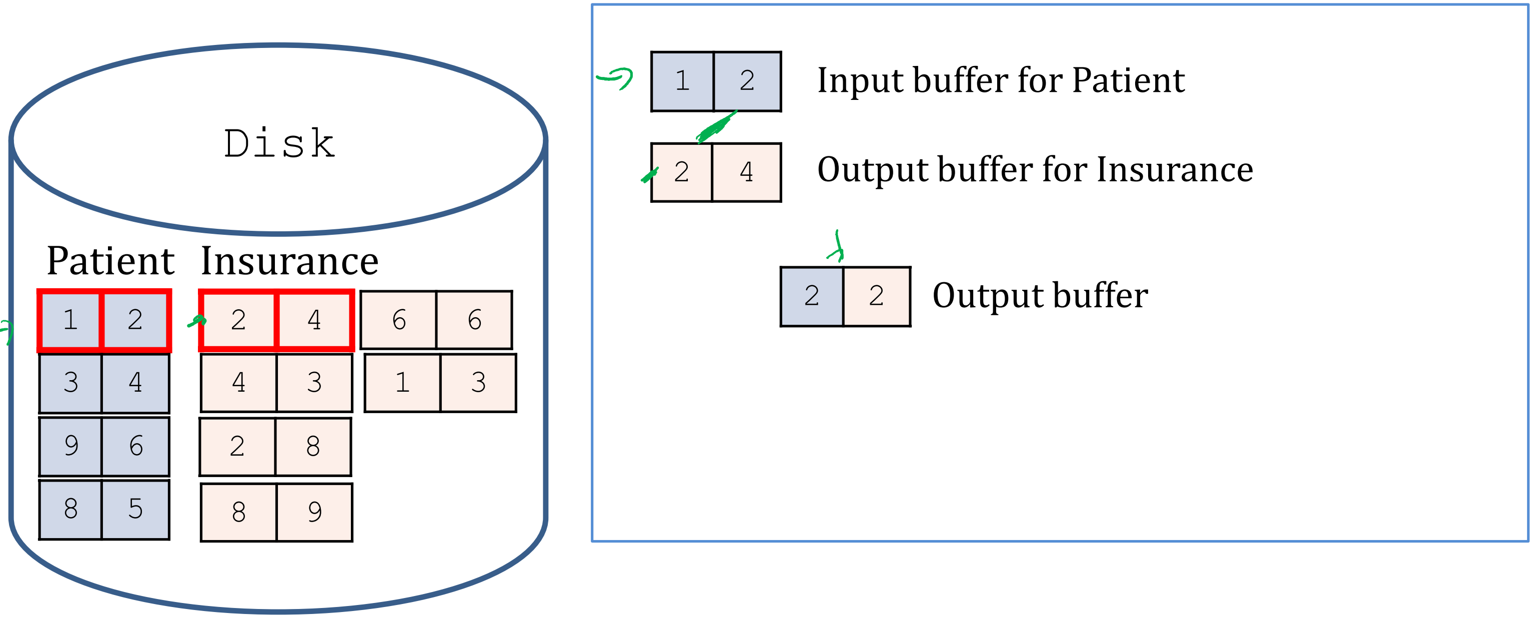
* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R,A)=T(R), V(R,A) < T(R))*

Block-at-a-time Refinement



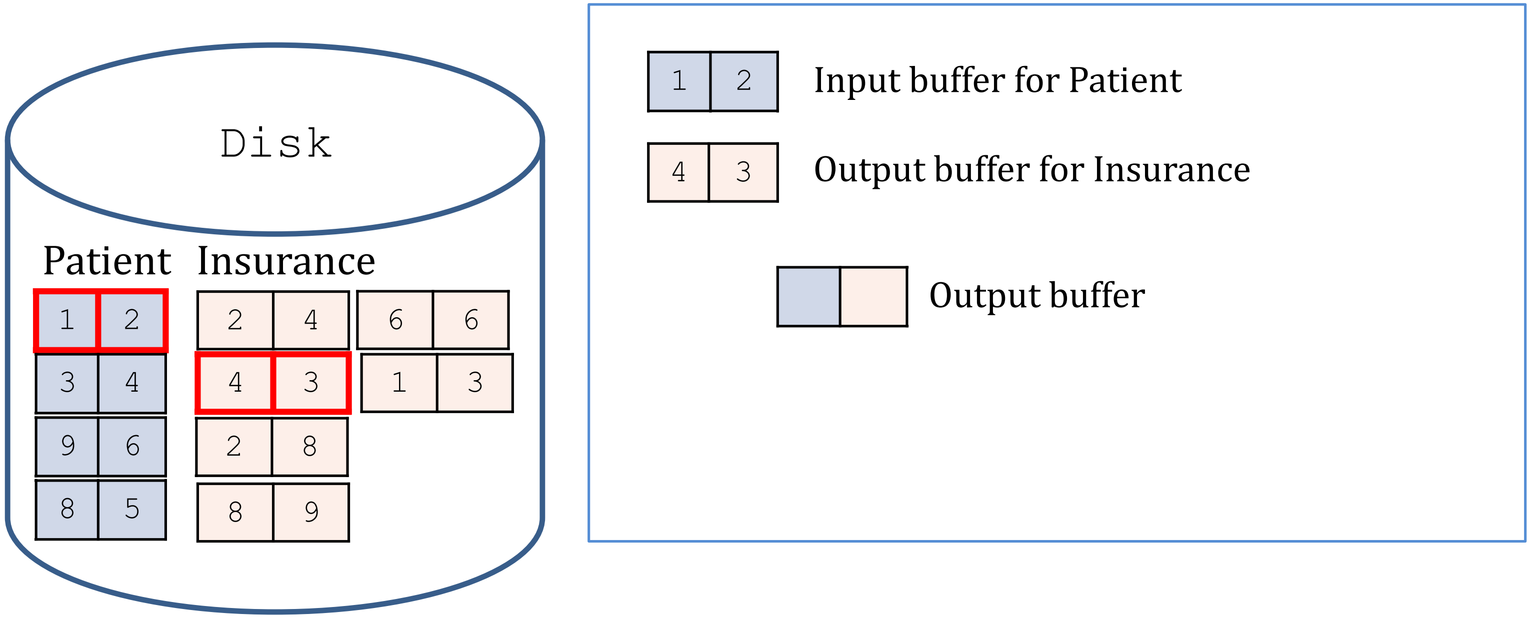
* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Block-at-a-time Refinement

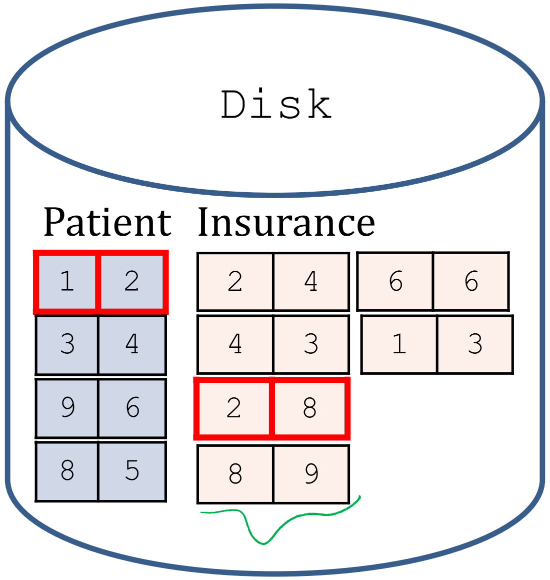


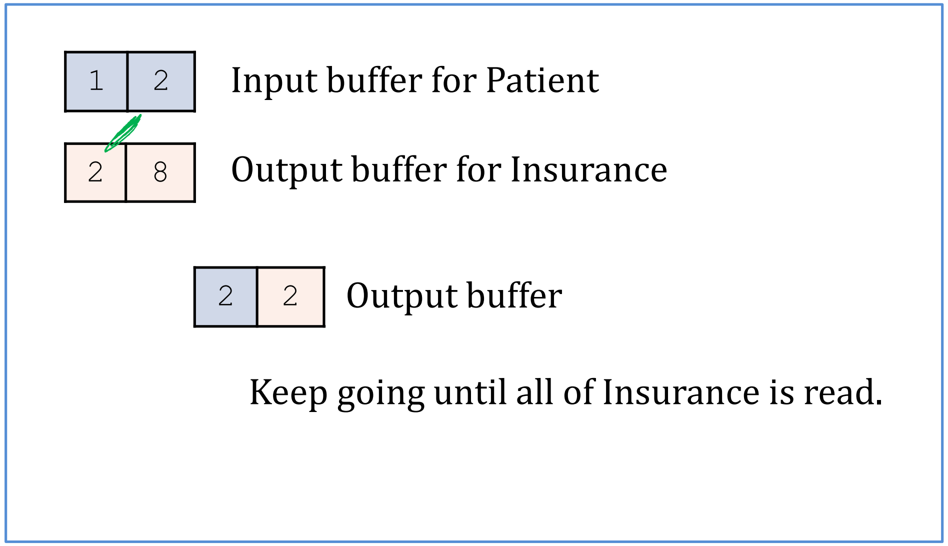
* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Block-at-a-time Refinement



Block-at-a-time Refinement





P → 1|2

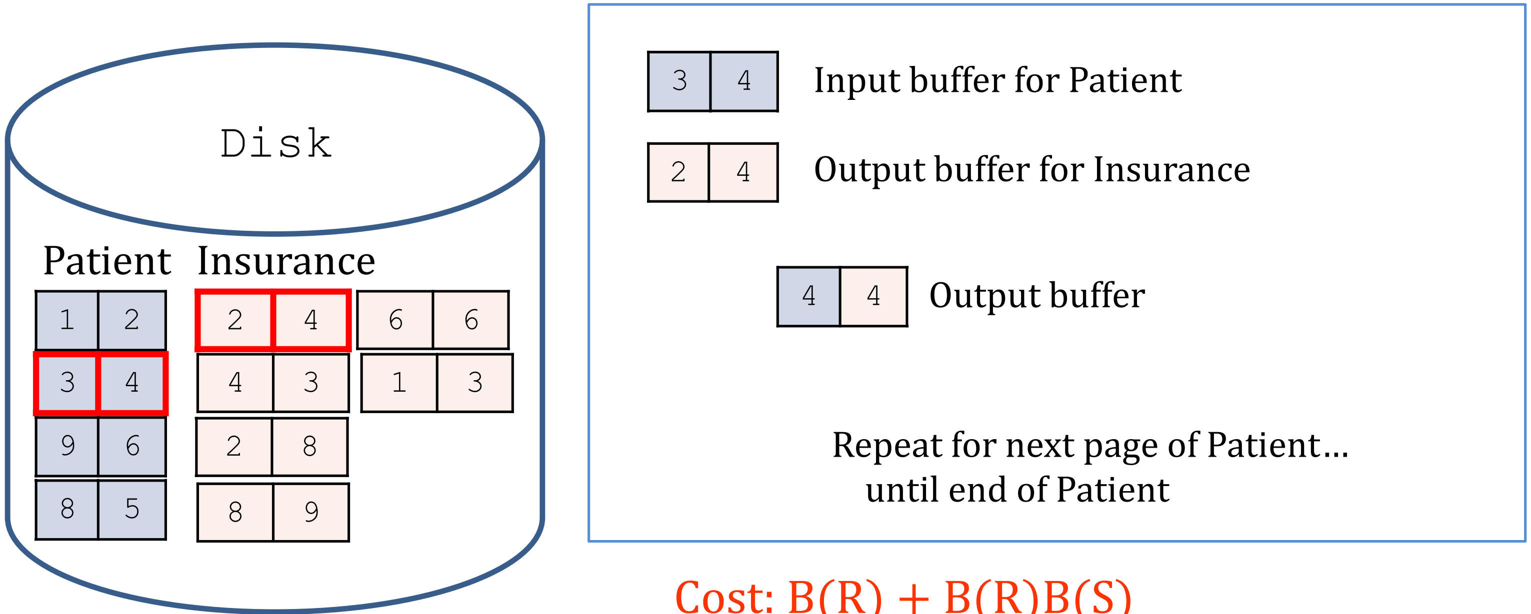
I → 8|9

O.B.

1 | 2

1 | 3

Block-at-a-time Refinement



Cost: B(R) + B(R)B(S)

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Block-Nested-Loop Refinement

```
for each (group) of M-1 pages, r in R do
 for each page of tuples s in S do
  for all pairs of tuples t1 in r, t2 in s
   if t1 and t2 join then output (t1, t2)
```

What is the Cost?

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Block-Nested-Loop Refinement

```
for each group of M-1 pages r in R do
 for each page of tuples s in S do
  for all pairs of tuples t1 in r, t2 in s
   if t1 and t2 join then output (t1, t2)
```

What is the Cost?

- Cost: B(R) + B(R)B(S)/(M-1)
  - Cost: B(R) + B(R)B(S) -> Block-at-a-time Refinement
  - Cost: B(R) + T(R)B(S) -> Nested Loop Join

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

Sort-Merge Join

- Sort-merge join: $R \bowtie S$
  - Scan R and sort in main memory
  - Scan S and sort in main memory
  - Merge R and S

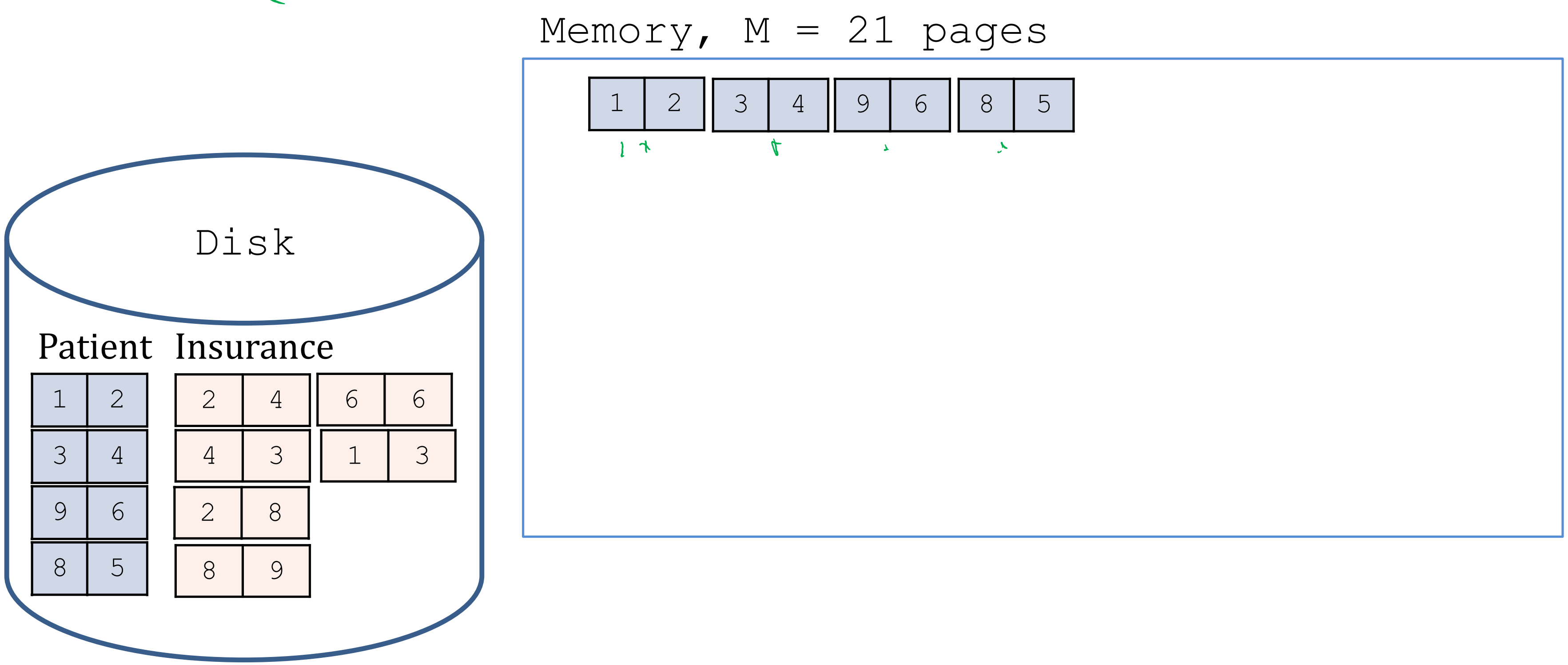
- Cost: B(R) + B(S)
  - One pass algorithm when B(S) + B(R) <= M
  - Typically, this is NOT a one pass algorithm

20 20 ~~4~~ 16

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

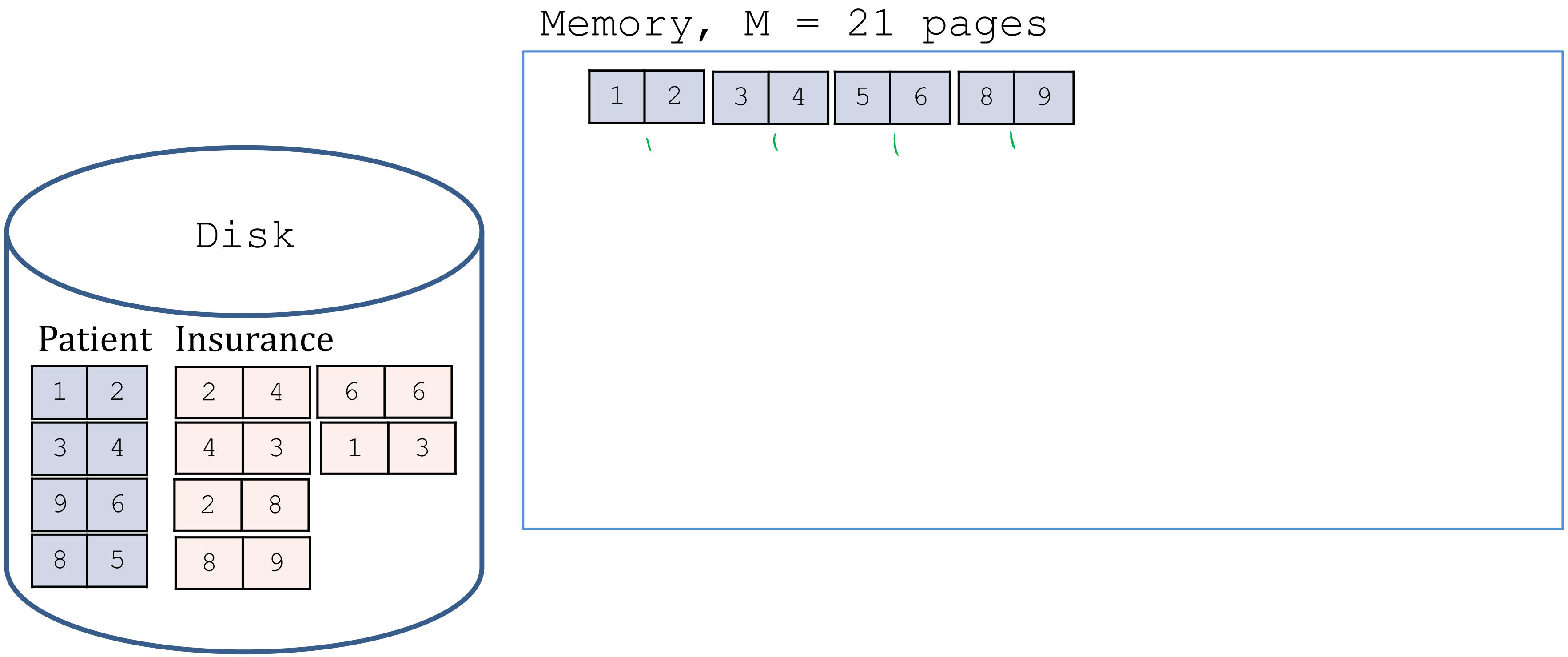
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory
(12)



Sort-Merge Join Example

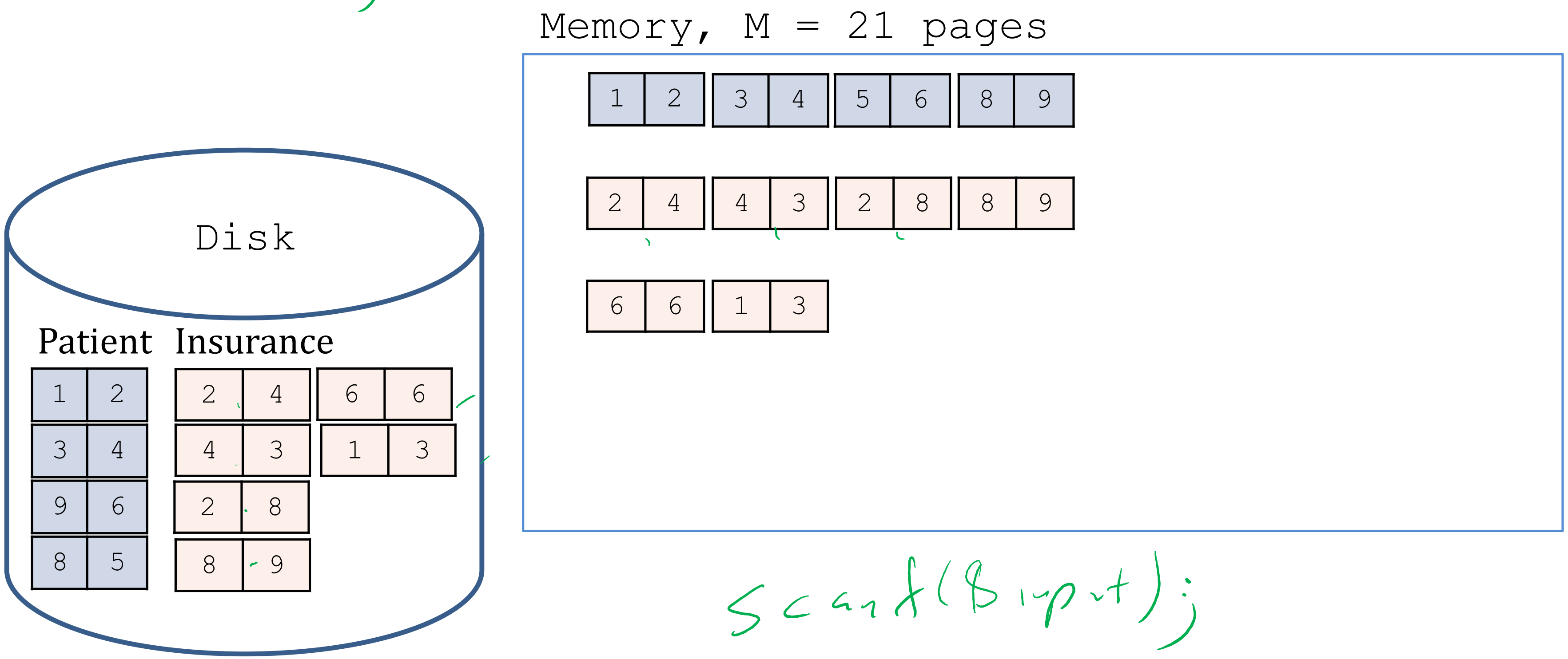
Step 1: Scan Patient and sort in memory



Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory

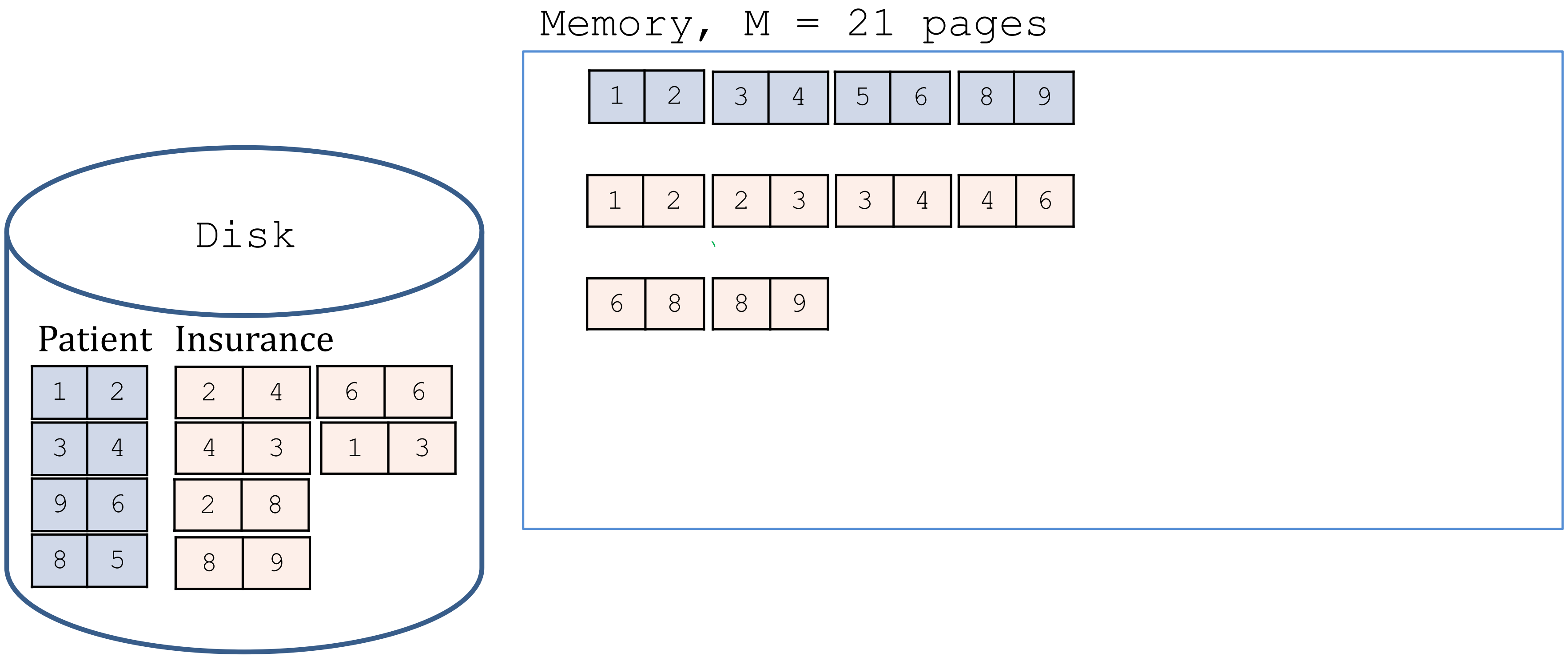
(s)



scanf(&input);

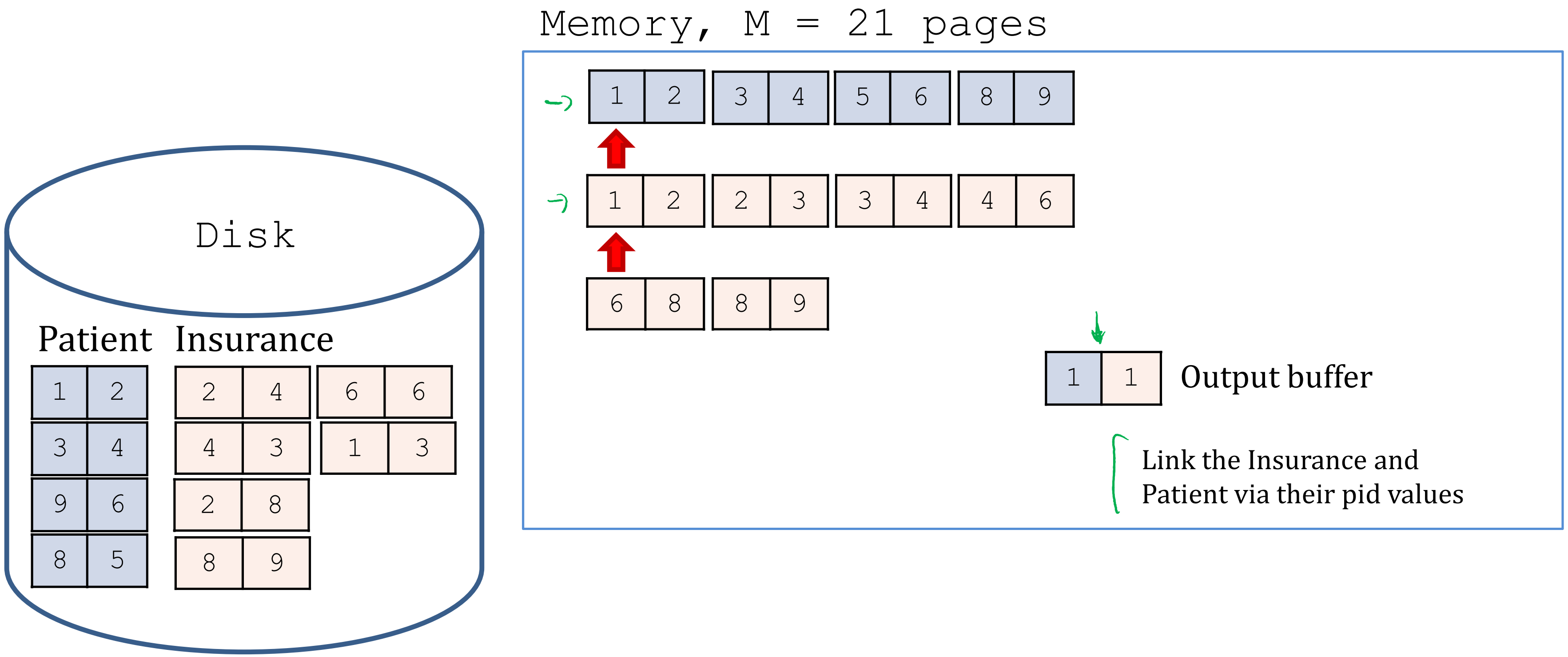
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory



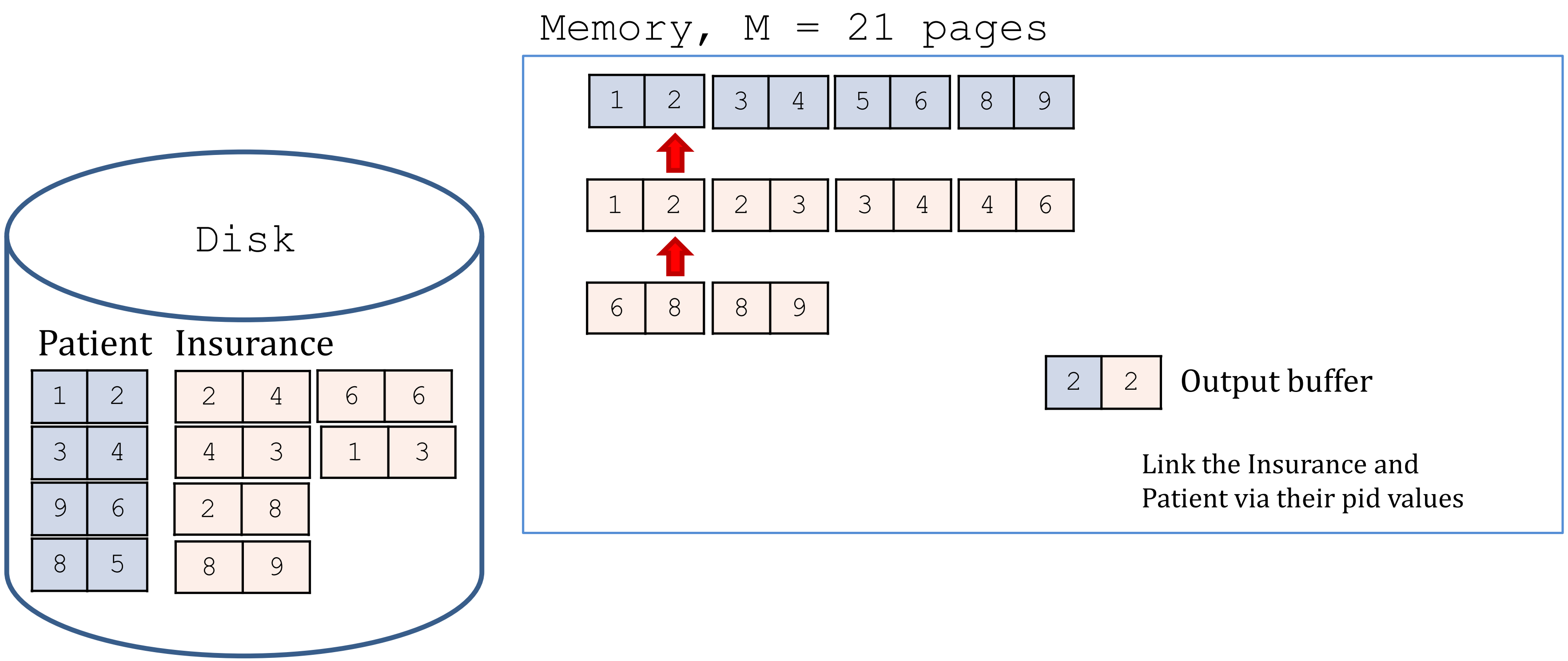
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Sort-Merge Join Example

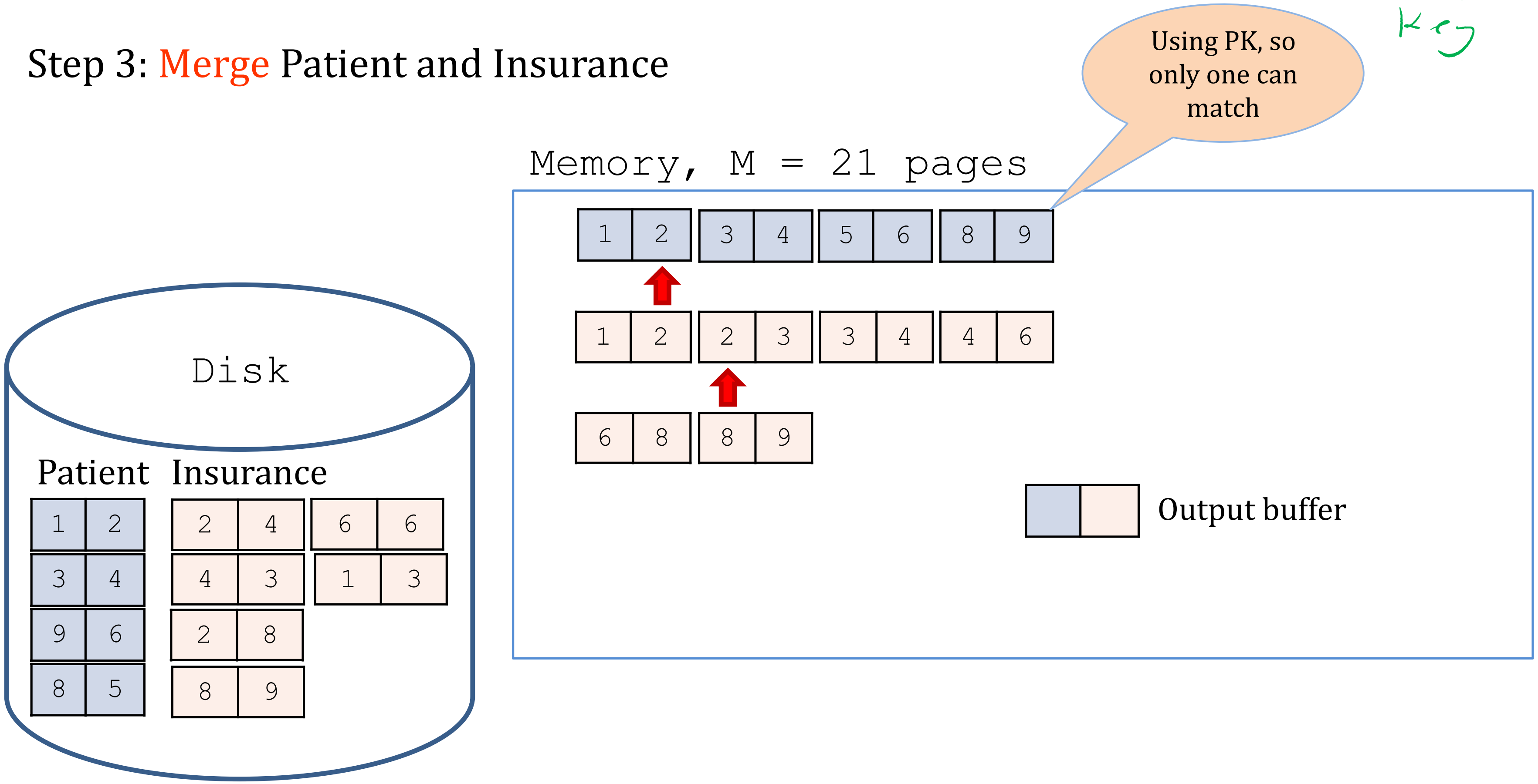
Step 3: Merge Patient and Insurance



Sort-Merge Join Example

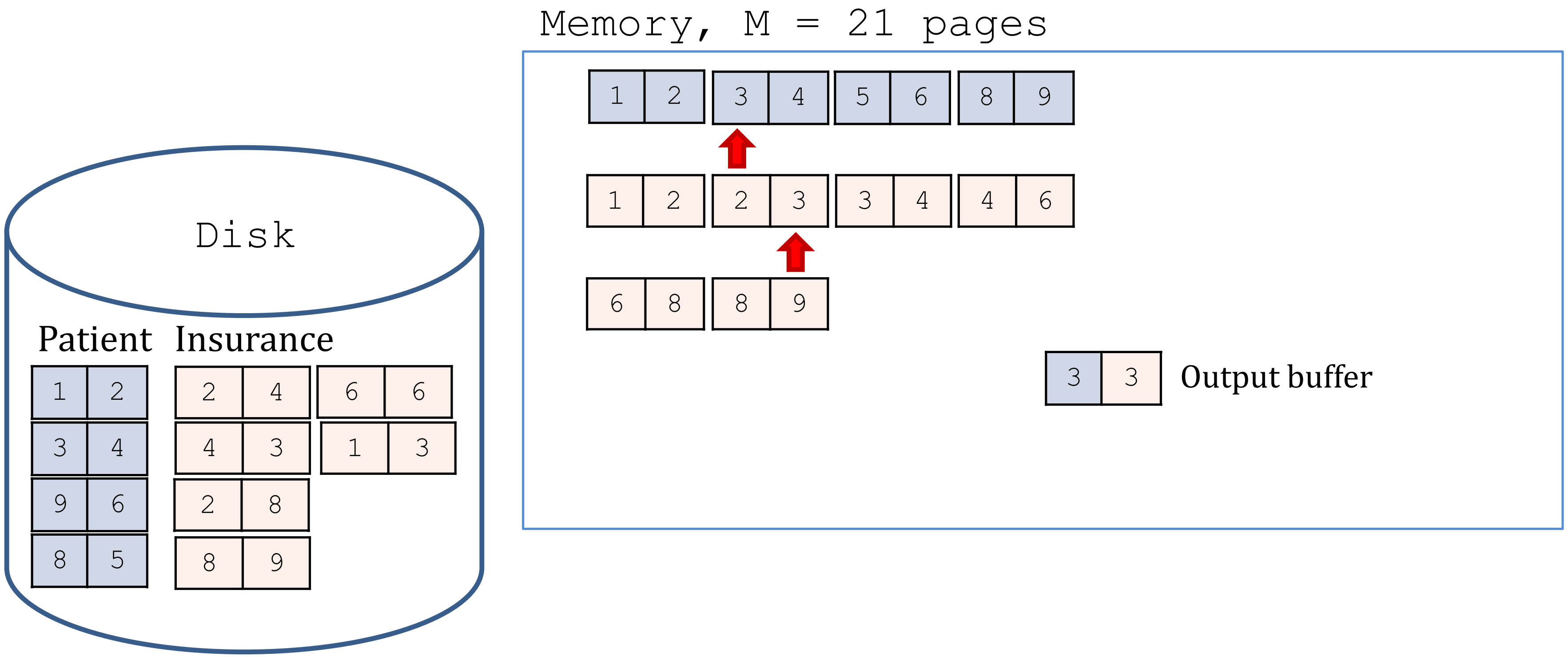
Primary key

Step 3: Merge Patient and Insurance



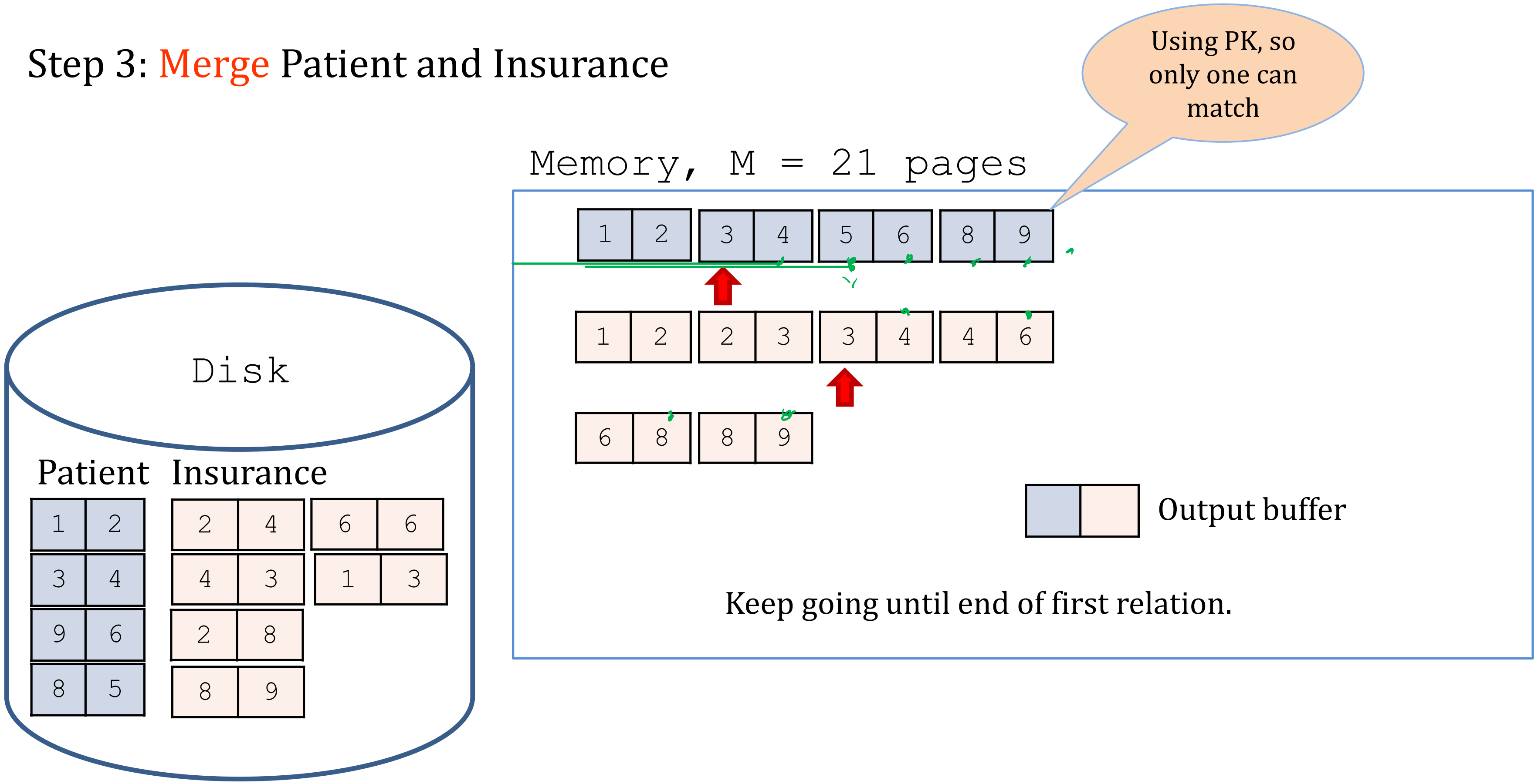
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Index Nested Loop Join

- R ⋈ S
  - Assume S has an index on the join attribute S pid ⋈R pid
  - (loop)
    Iterate over R, for each tuple, fetch corresponding tuple(s) from S

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R,A)=T(R), V(R,A) < T(R))*

Index Nested Loop Join

- R ⋈ S
  - Assume S has an index on the join attribute
  - Iterate over R, for each tuple, fetch corresponding tuple(s) from S

T(S) = 50,000 T(R) = 100,000
B(S) = 100 B(R) = 200

- Cost:
  - If index on S is clustered: B(R) + T(R)B(S)/V(S, A) 50K
  - If index on S is unclustered: B(R) + T(R)T(S)/V(S, A) 100

* B(R) = # of blocks (i.e., pages) for relation R
* T(R) = # of tuples in relation R
* V(R, A) = # of distinct values of attribute(column) A; *IF (A is a key?, V(R, A)=T(R), V(R, A) < T(R))*

INTRO TO DATA STRUCTURE

Cost of Query Plans

Physical Query Plan 1

(On the fly) π_{sname}

(On the fly) $\sigma_{city = \text{'Jeonju'} \wedge sprov = \text{'Capiz'} \wedge pno = 2}$

(Nested loop) $\bowtie_{sid = sid}$

Supplier (File scan)   Supply (File scan)

~~Select sname~~
From Suppliers, Supply
Where city = 'Jeonju' AND
sprov = 'Capiz' AND
pno = 2 AND
s.pid = u.pid;

Physical Query Plan 1

Disk I/O's

```
T(Supplier) = 1000     B(Supplier) = 100     V(Supplier,scity) = 20     M = 11
T(Supply) = 10,000     B(Supply) = 100       V(Supplier,sprov) = 10
                                             V(Supply,pno) = 2,500
```

(On the fly) π_{sname}

(On the fly) $\sigma_{\text{city} = \text{'Jeonju'} \wedge \text{sprov} = \text{'Capiz'} \wedge \text{pno} = 2}$

(Nested loop) $\bowtie_{\text{sid} = \text{sid}}$

Supplier (File scan) | Supply (File scan)

Selection and project on-the-fly
→ No additional cost

Total cost of plan is thus the cost of join:
= B(Supplier)+B(Supplier)*B(Supply)
= 100 + 100 * 100
= 10,100 I/Os

B(Sup) + B(Sup) B(Sp)

Physical Query Plan 2

(On the fly) π_{sname}

(Sort-merge join) $\bowtie_{\text{sid} = \text{sid}}$

(Scan & write to T_1) $\sigma_{\text{city} = \text{'Jeonju'} \wedge \text{sprov} = \text{'Capiz'}}$

(Scan & write to T_2) $\sigma_{\text{pno} = 2}$

Supplier (File scan)

Supply (File scan)

Physical Query Plan 2

```
T(Supplier) = 1000      B(Supplier) = 100     V(Supplier,scity) = 20    M = 11
T(Supply) = 10,000      B(Supply) = 100       V(Supplier,sprov) = 10
                                              V(Supply,pno) = 2,500
```

(On the fly) — (d) π_{sname} = 0

(Sort-merge join) — (c) $\bowtie_{sid = sid}$ = ?

(Scan & write to T_1) — (a) $\sigma_{city = \text{'Jeonju'} \wedge sprov = \text{'Capiz'}}$ = 0

(Scan & write to T_2) — (b) $\sigma_{pno = 2}$ = 0

Supplier (File scan)

Supply (File scan)

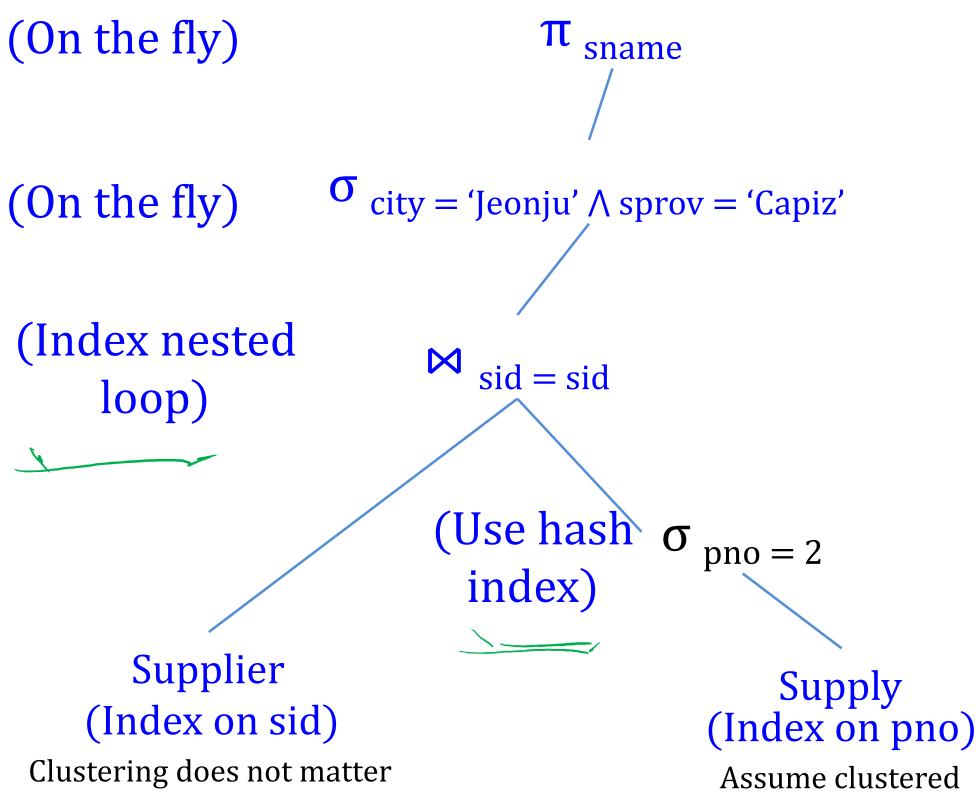
~~Selection~~ σ and project π on-the-fly X
→ No additional cost

Total cost of plan is thus the cost of join:
= 100 + 100 * (1/20) * (1/10) (a)
+ 100 + 100 * (1/2500) (b)
+ 2 (c)
+ 0 (d)
≈ **204 I/Os**

a) = B(Supp) + B(Supp) * $\frac{1}{20}$ * $\frac{1}{10}$ — V(Supp, scity), V(Supp, sprov)

~~b) = B(S)~~ + B(S) + V(S, pno)
100 + 100 + $\frac{1}{2500}$

Physical Query Plan 3



Physical Query Plan 3

```
T(Supplier) = 1000      B(Supplier) = 100      V(Supplier,scity) = 20     M = 11
T(Supply) = 10,000      B(Supply) = 100        V(Supplier,sprov) = 10
                                               V(Supply,pno) = 2,500
```

(On the fly) π_{sname} (d)

(On the fly) $\sigma_{\text{city = 'Jeonju'} \wedge \text{sprov = 'Capiz'}}$ (c)

(Index nested loop) $\bowtie_{\text{sid = sid}}$ (b)

(Use hash index) $\sigma_{\text{pno = 2}}$ (a)

Supplier
(Index on sid)
Clustering does not matter

Supply
(Index on pno)
Assume clustered

Selection and project on-the-fly
→ No additional cost

Total Cost
= 1 (a)
+ 4 (b)
+ 0 (c)
+ 0 (d)
≈ 5 I/Os

Thank you.