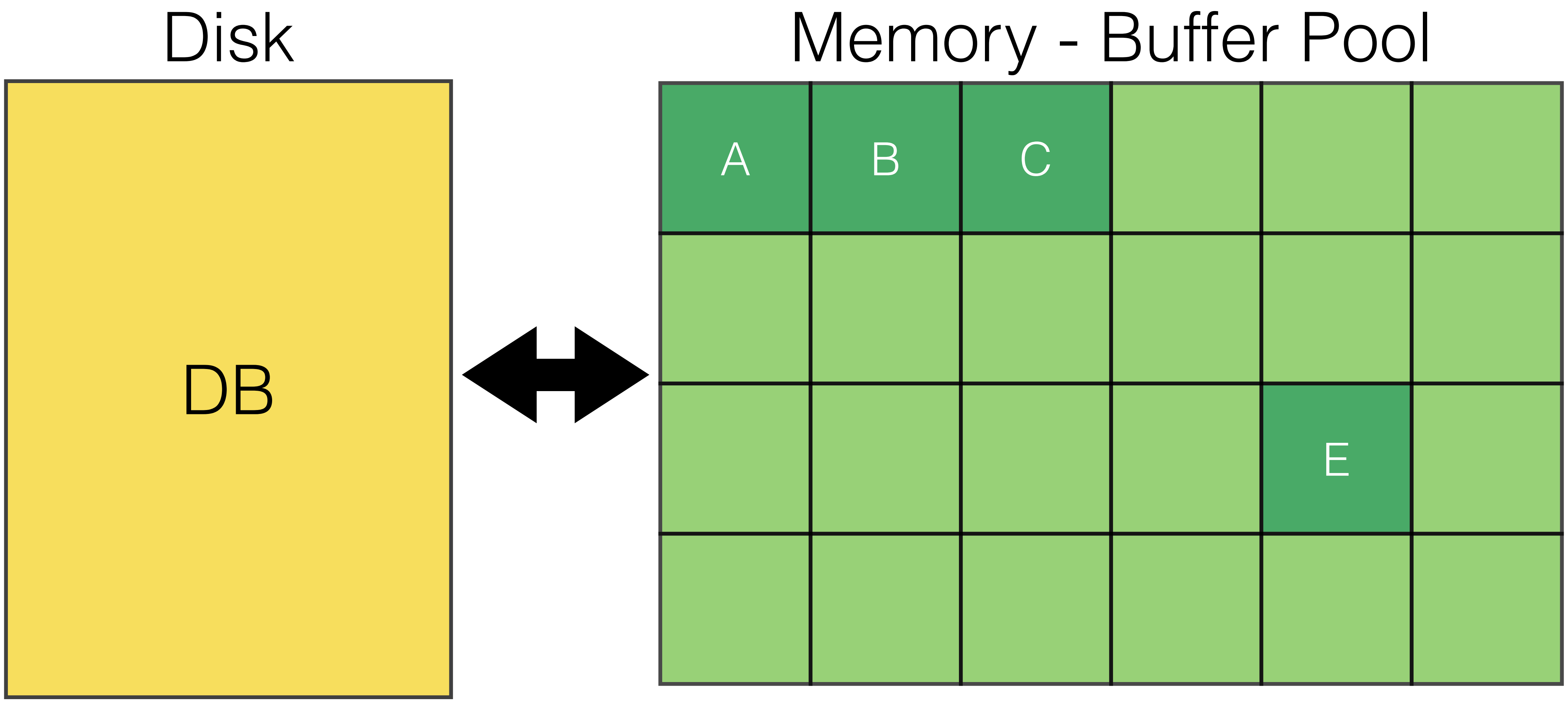
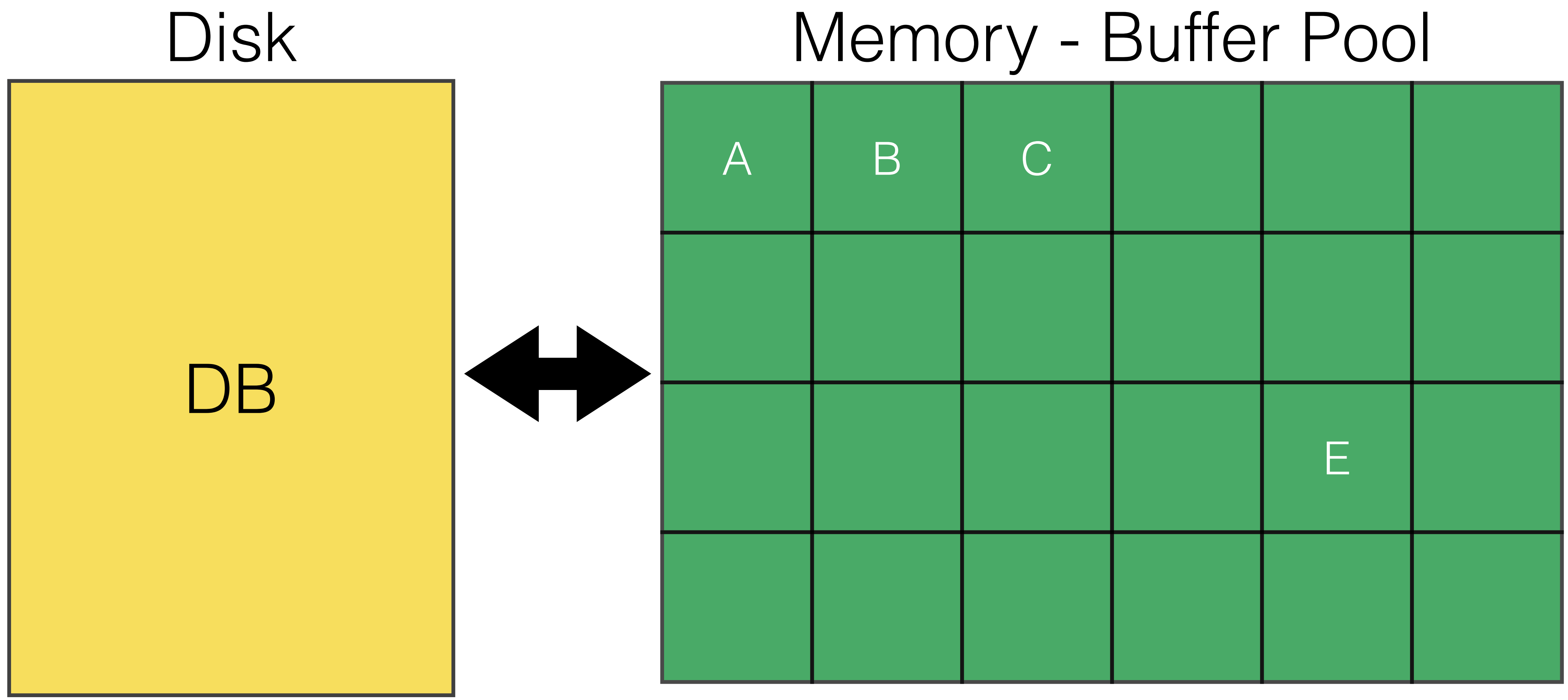
CS186 Discussion #4

(Buffer Management, Indexes, File Organization)

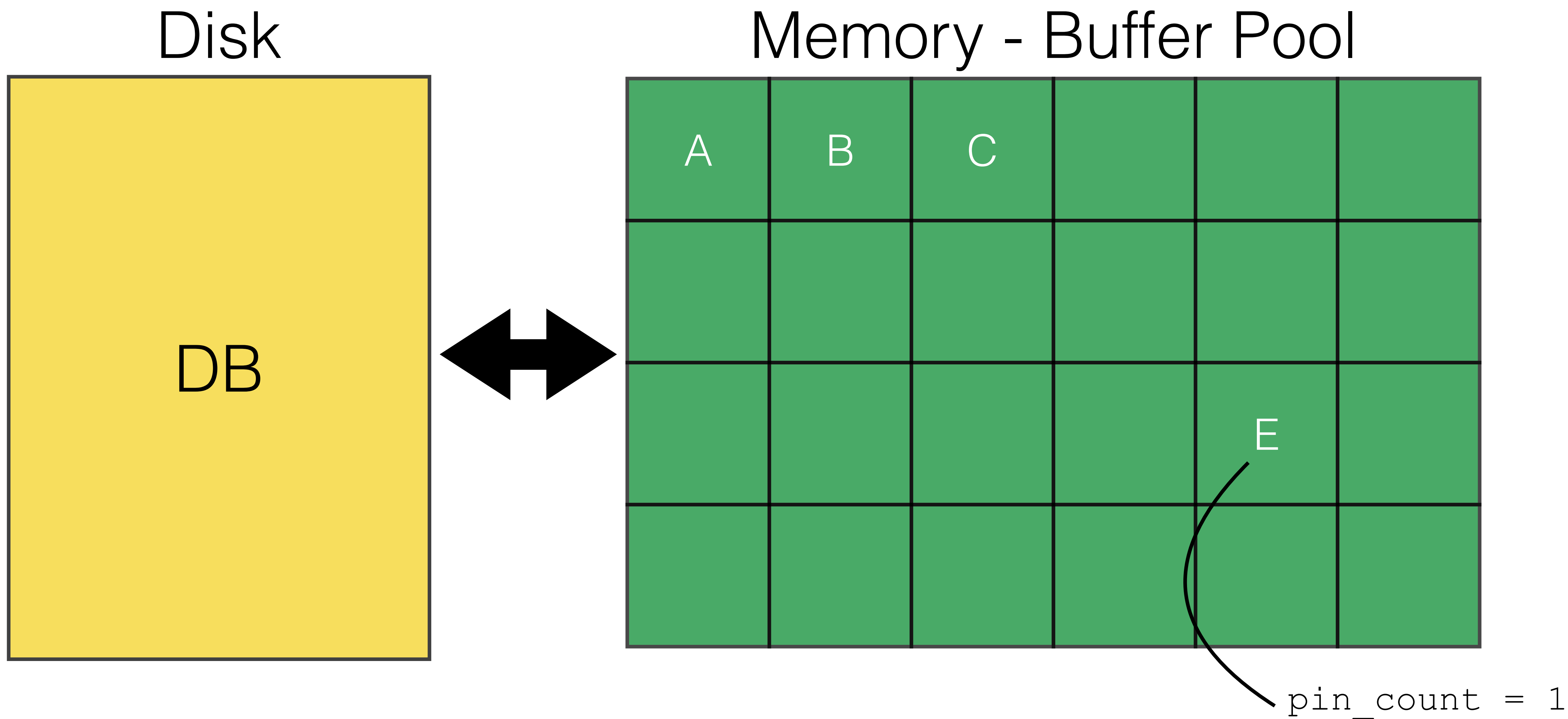
Buffer Management


Disk
DB
Memory - Buffer Pool
A
B
C
E

What happens when our buffer pool is full?
Which pages can we replace?



“Pin” a page (`pin_count++`) when page is requested. Only replace if `pin_count == 0`.



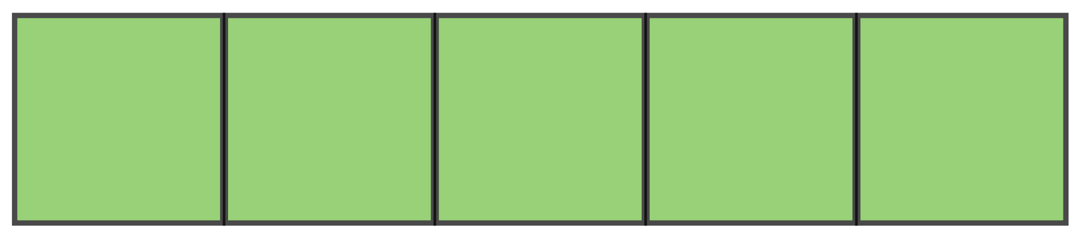
Buffer Replacement Policy

- Frame chosen for replacement using replacement policy (LRU, MRU, Clock, etc.)
- Policy can have a big impact on I/O's

Least Recently Used (LRU)

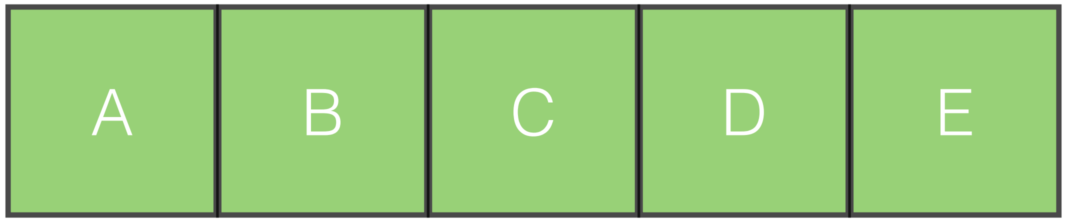
- Replace page that has been unused for the longest amount of time
  - Assumes pages used recently will be used again
- Must keep track of last time page was used/pinned
- Prone to sequential flooding
  - Reading all pages in a file multiple times
  - # buffer pages < # pages in file

LRU - Sequential Flooding



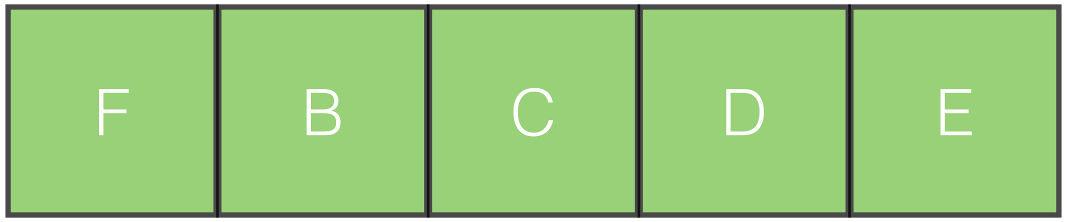
A, B, C, D, E, F, A, B, C, D

LRU - Sequential Flooding



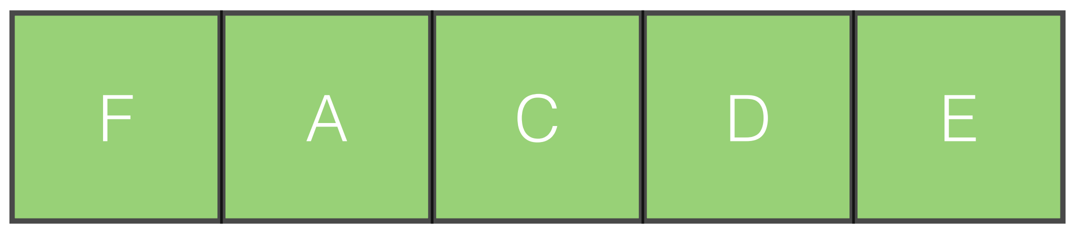
F, A, B, C, D

LRU - Sequential Flooding



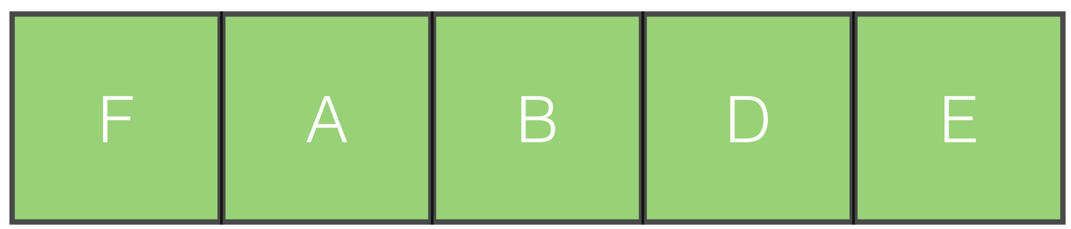
A, B, C, D

LRU - Sequential Flooding



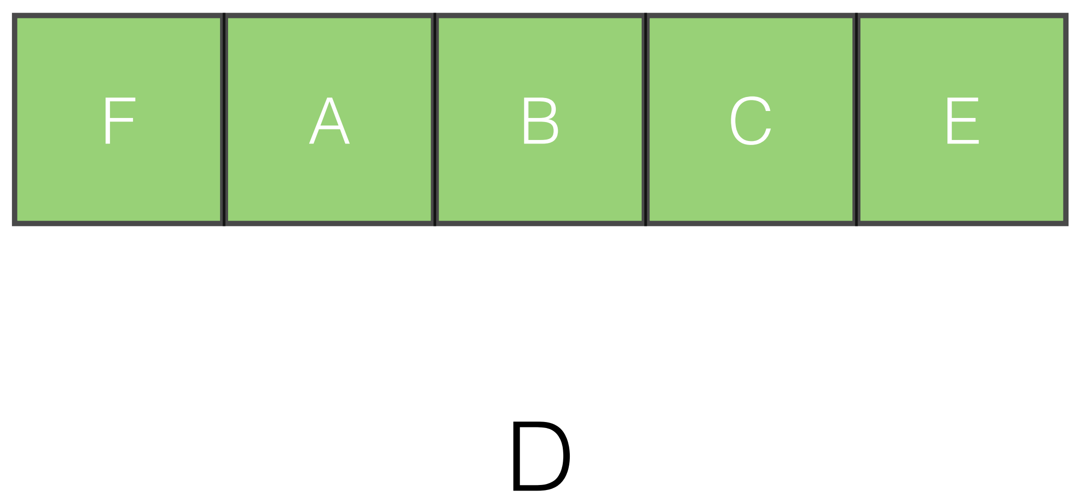
B, C, D

LRU - Sequential Flooding



C, D

LRU - Sequential Flooding



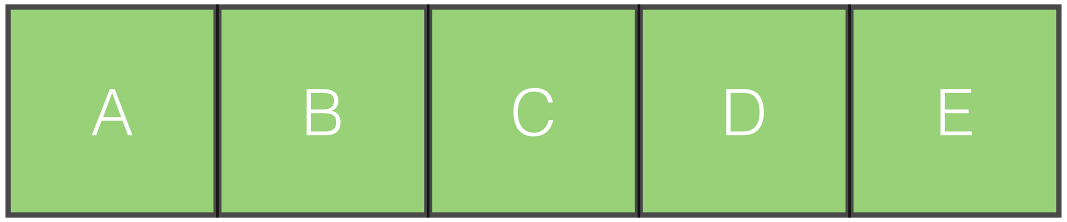
LRU - Sequential Flooding

Every page request results in a cache miss!

Most Recently Used (MRU)

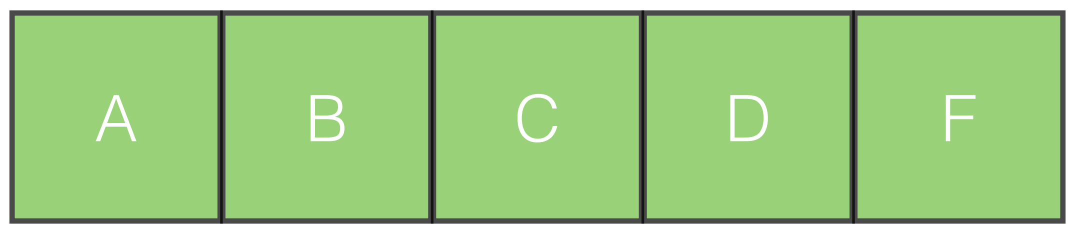
- Replace page that has just been used
- Fixes sequential flooding

MRU - Sequential Flooding



F, A, B, C, D

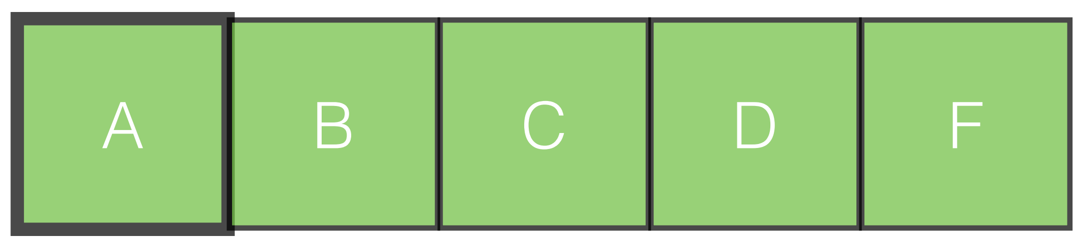
MRU - Sequential Flooding



A, B, C, D

MRU - Sequential Flooding

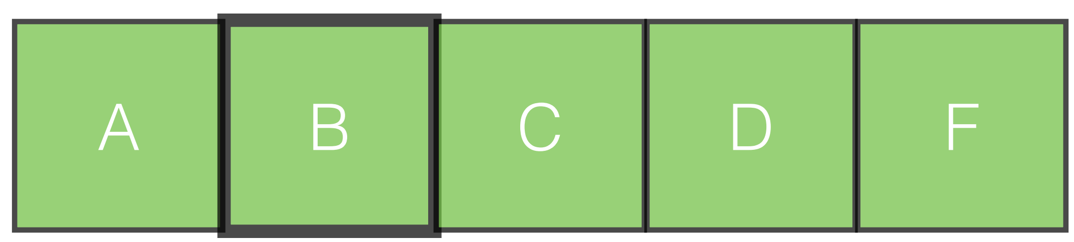
Cache hit!



B, C, D

MRU - Sequential Flooding

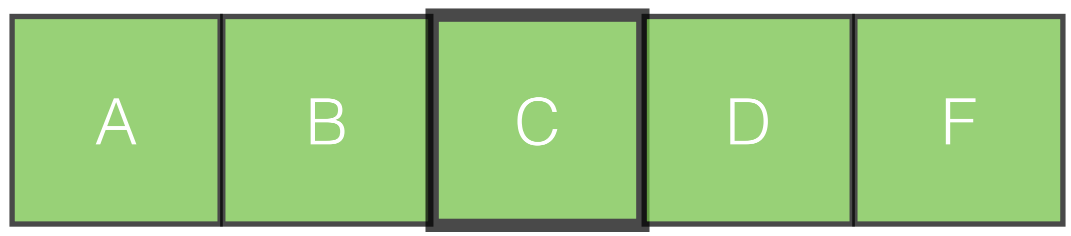
Cache hit!



C, D

MRU - Sequential Flooding

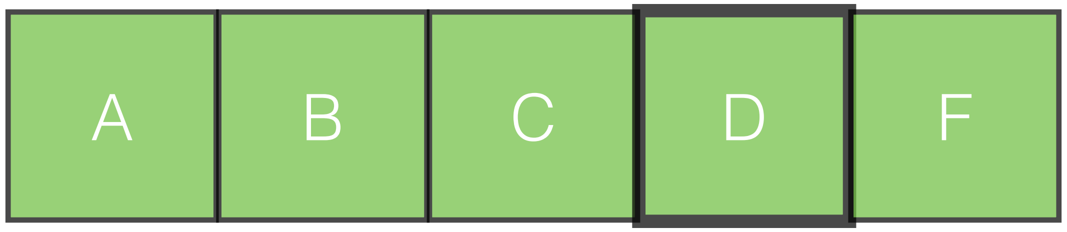
Cache hit!



D

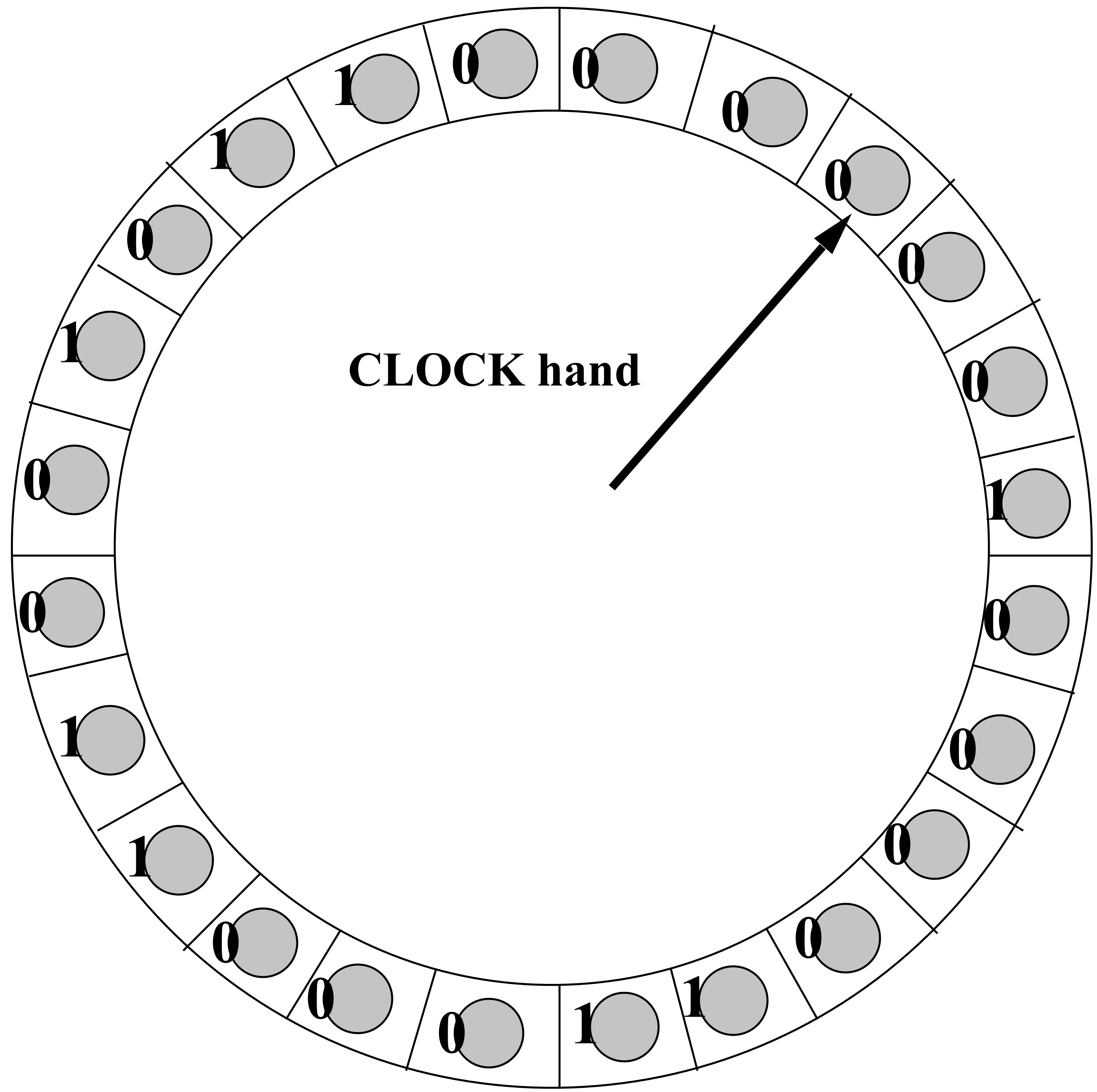
MRU - Sequential Flooding

Cache hit!



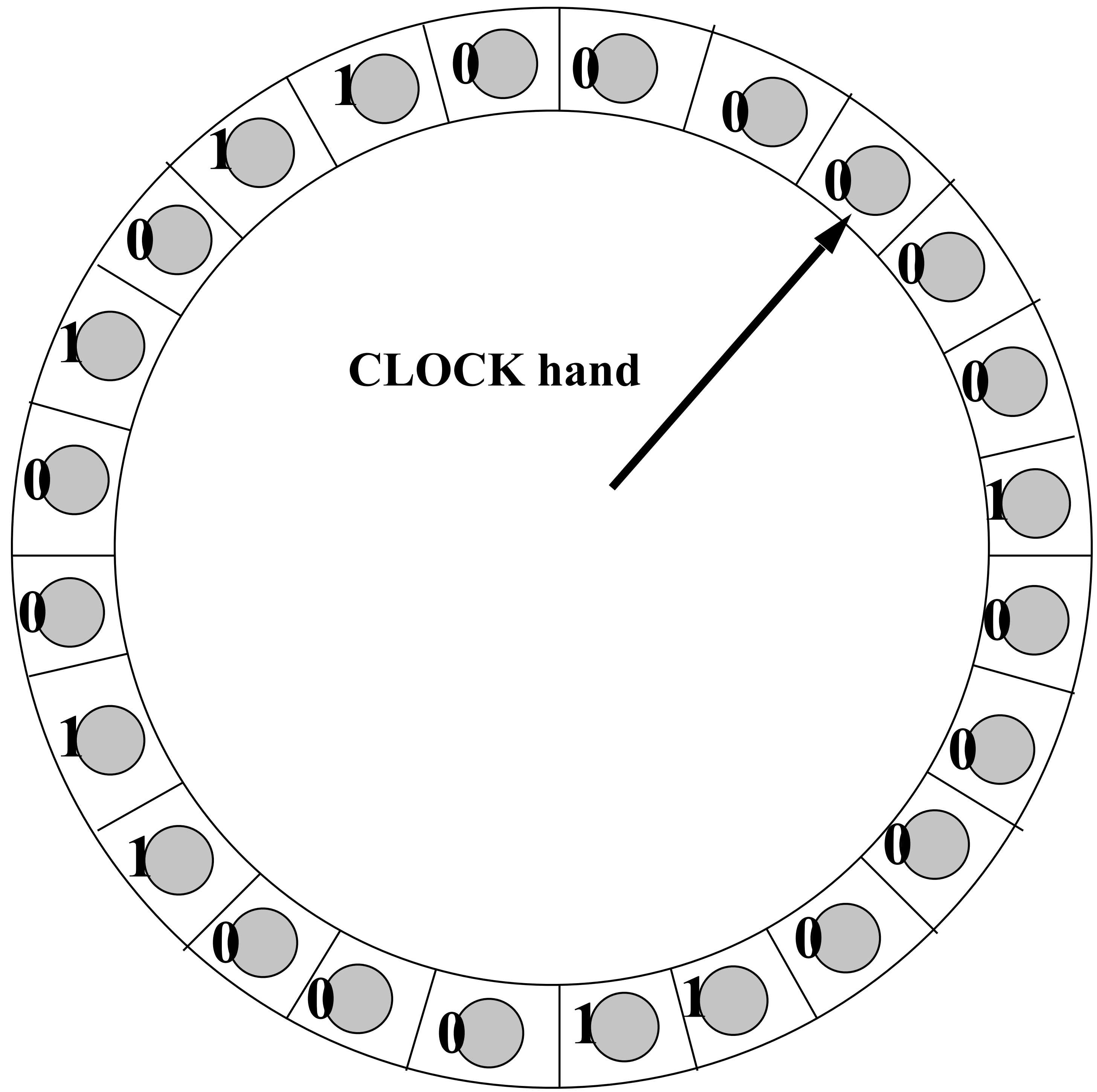
Clock Replacement

- All pages placed in a circular list.
- Each page has reference bit ("second-chance" bit) indicating if page has been accessed.



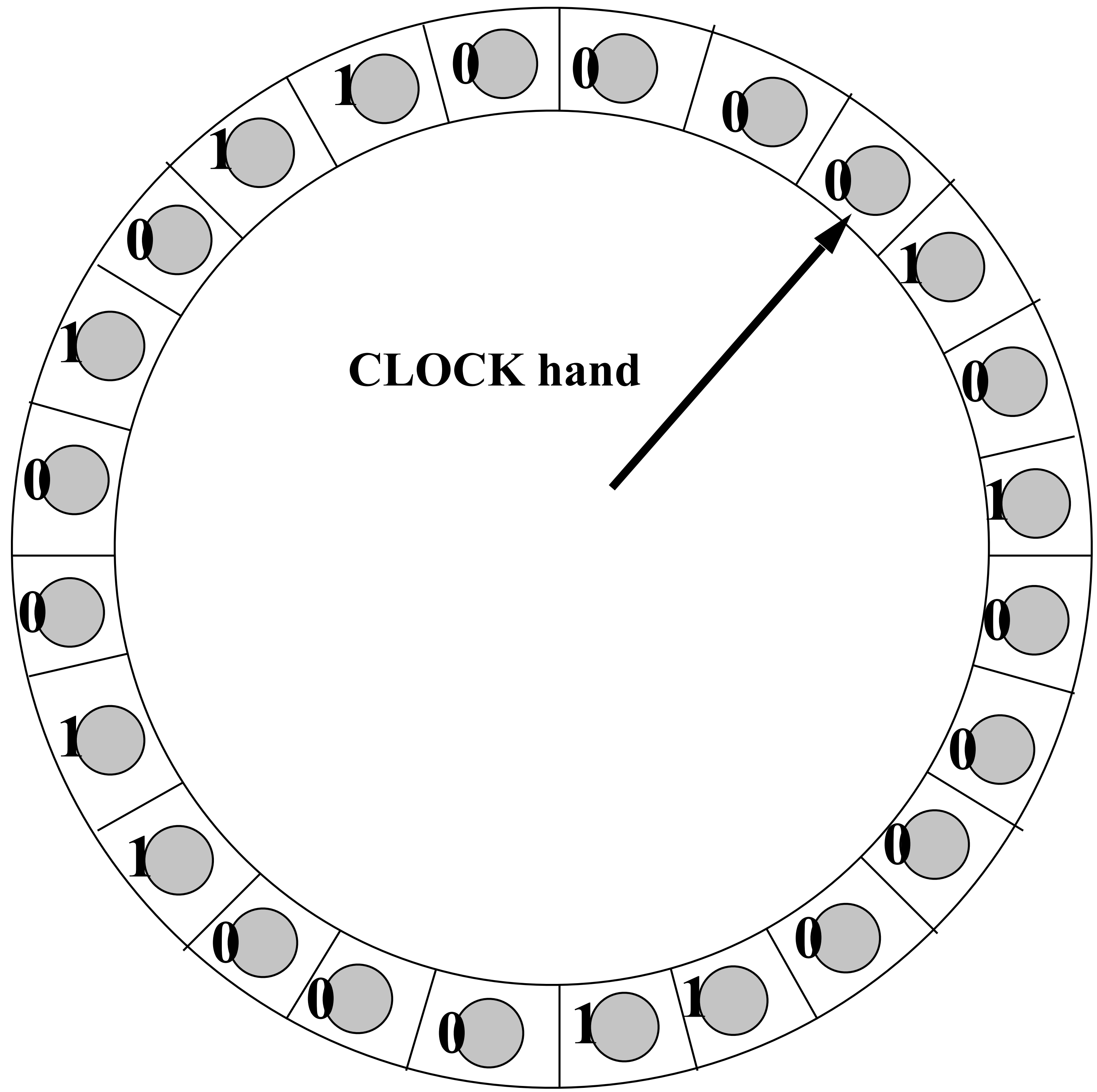
Clock Replacement

- On a HIT, set reference bit to 1.



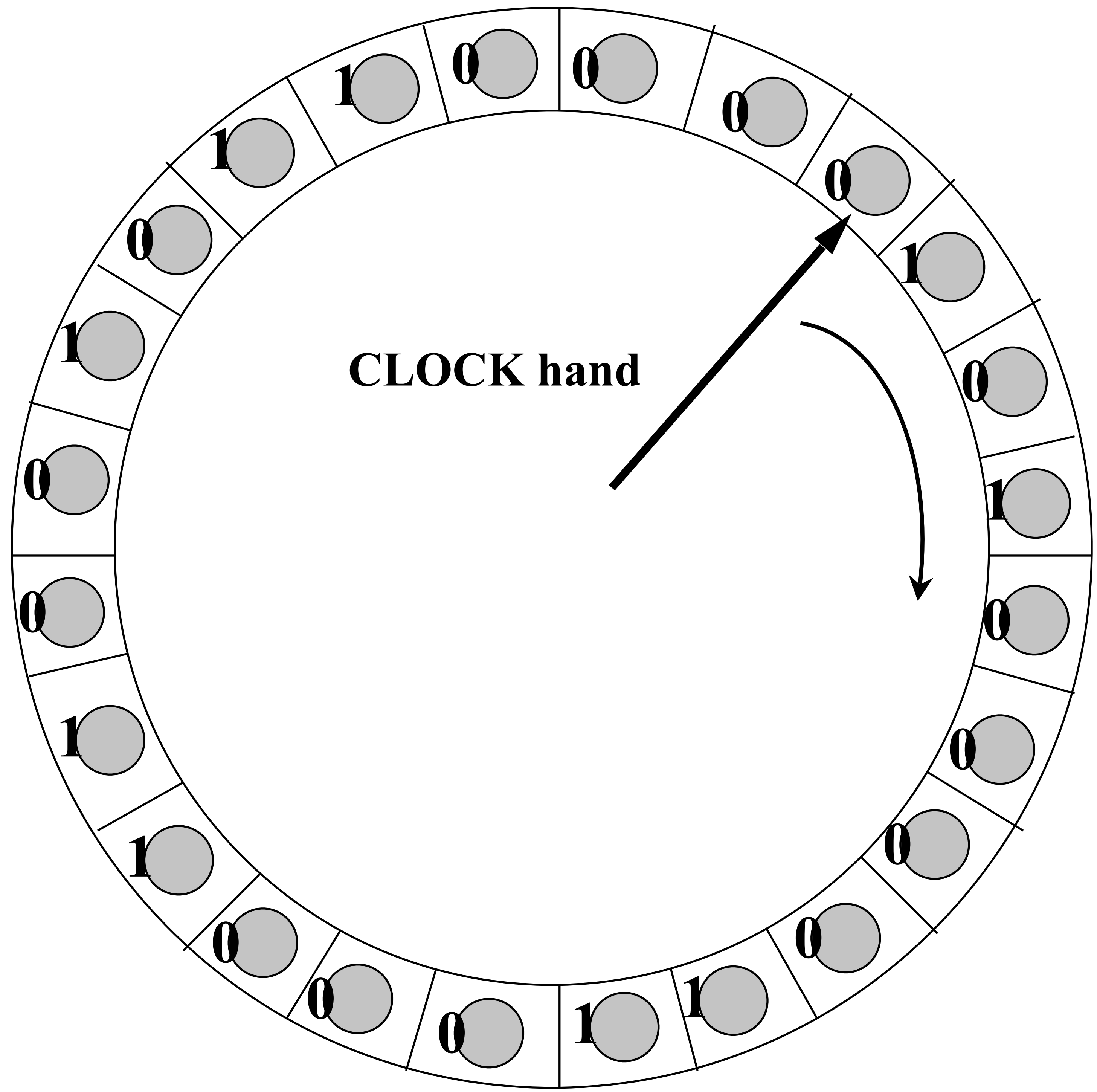
Clock Replacement

- On a HIT, set reference bit to 1.



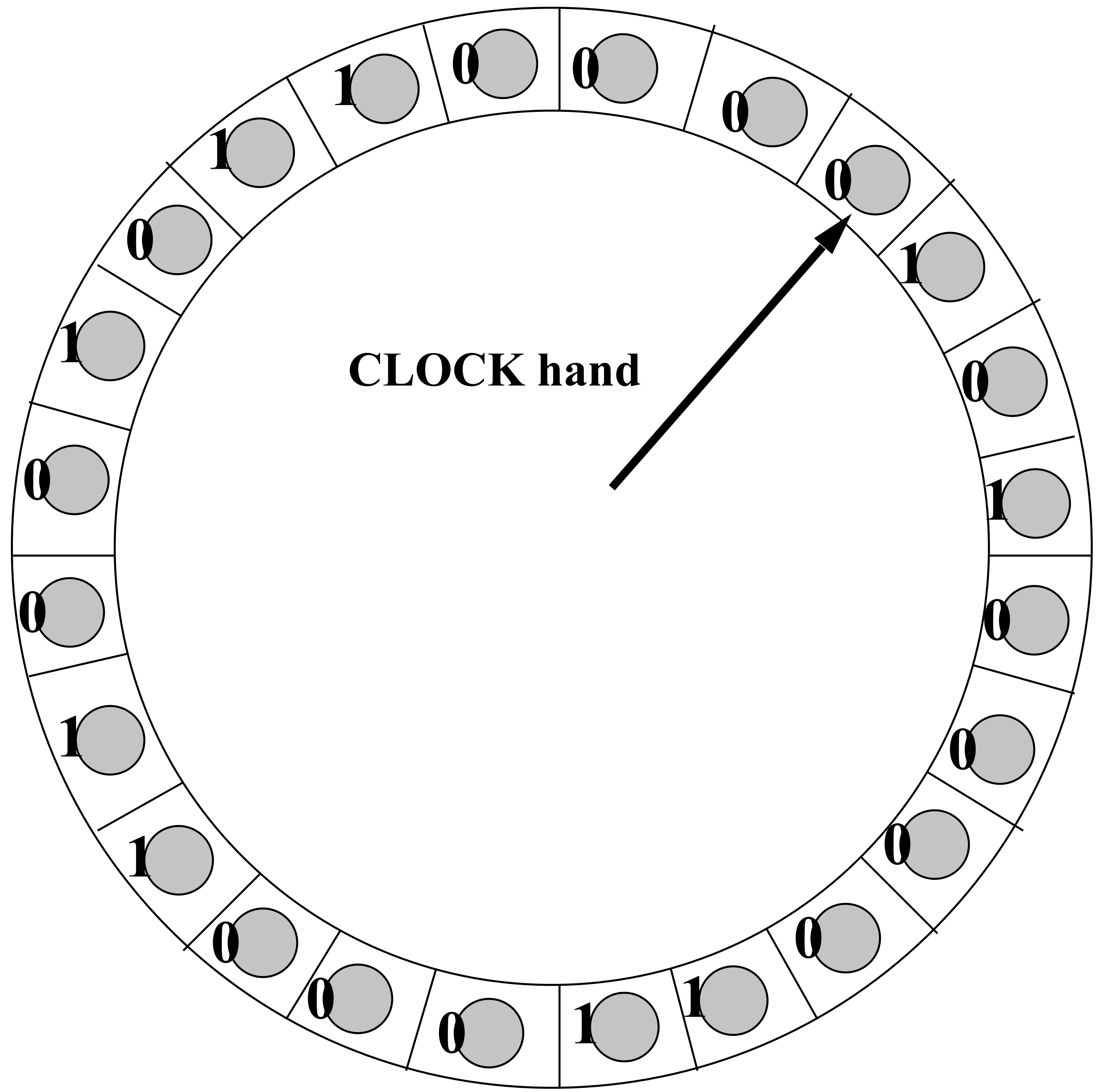
Clock Replacement

- On a MISS, move clock hand until reaches a page with “0” bit.
- Gives “1” bit pages a second chance and does not evict, but resets “1” to “0”.



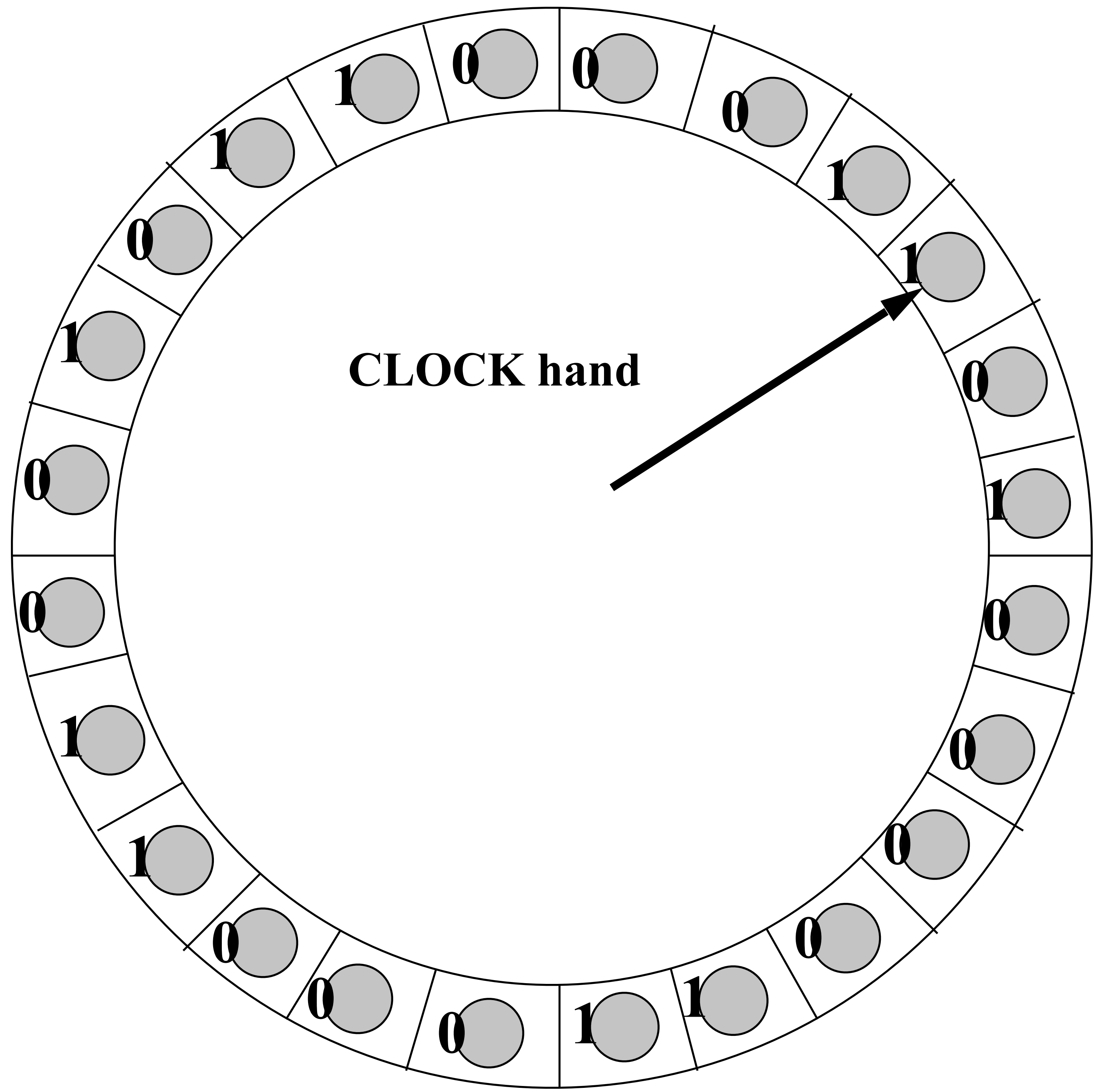
Clock Replacement

- 1 MISS



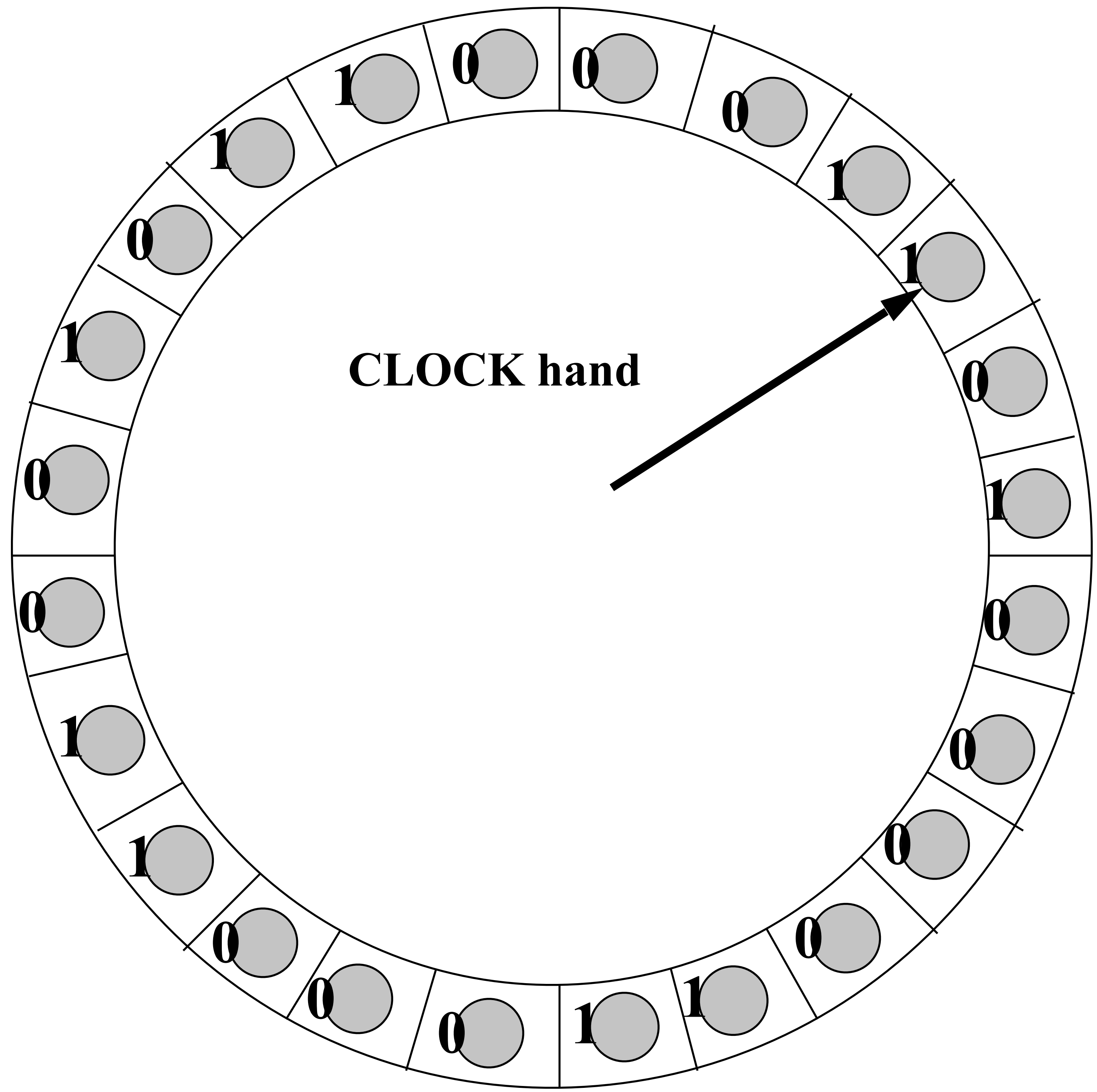
Clock Replacement

- 1 MISS



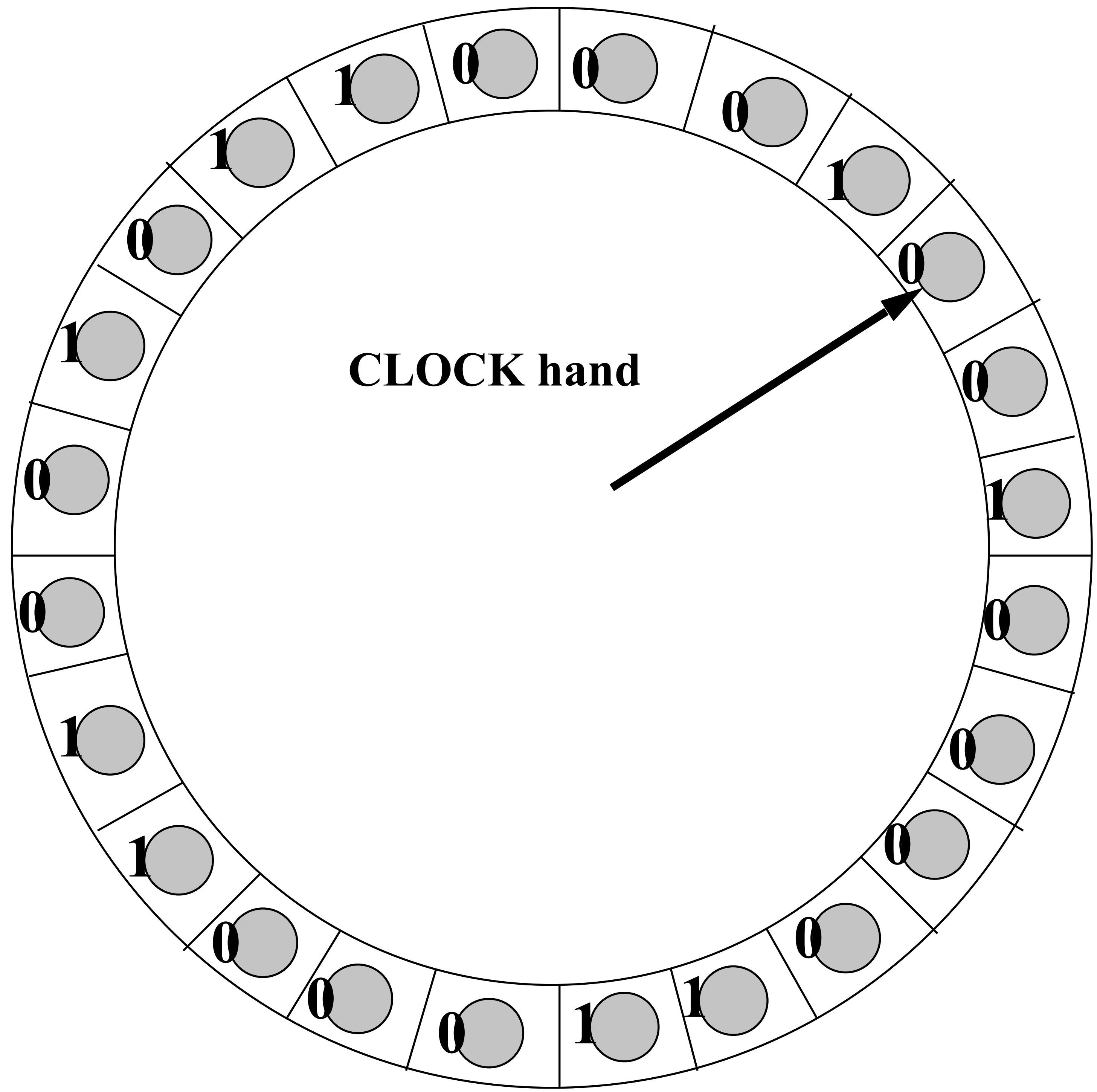
Clock Replacement

- Another MISS



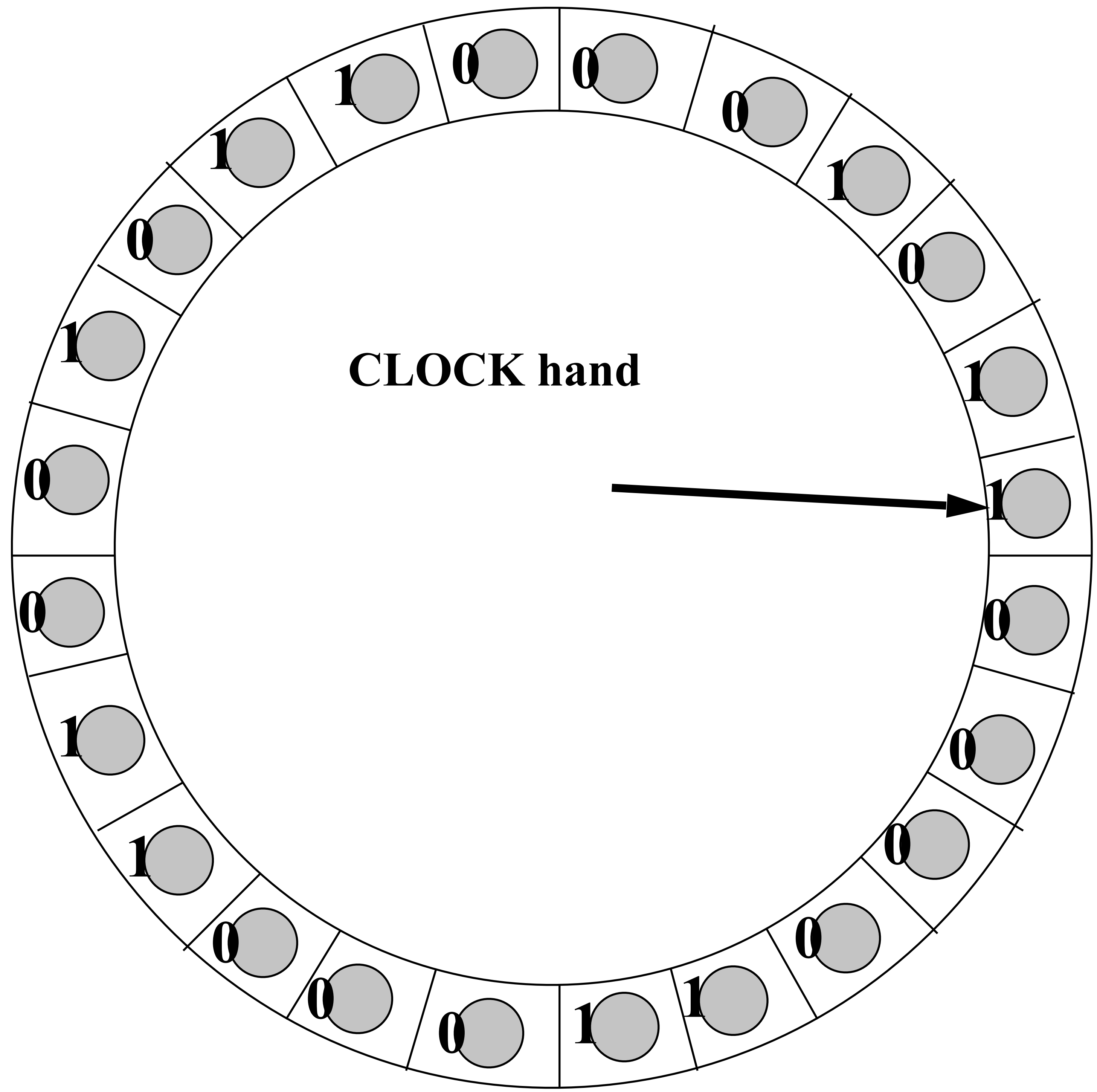
Clock Replacement

- Another MISS



Clock Replacement

- Another MISS



Worksheet #1

LRU

Access Pattern: A B C D A F A D G D G E D F

1	A				✓		✓							F	Hit Rate
2		B				F						E			6/14
3			C						G		✓				
4				D				✓		✓			✓		

MRU

Access Pattern: A B C D A F A D G D G E D F

1	A				✓	F	A								Hit Rate
2		B													2/14
3			C												
4				D				✓	G	D	G	E	D	F	

Clock Replacement

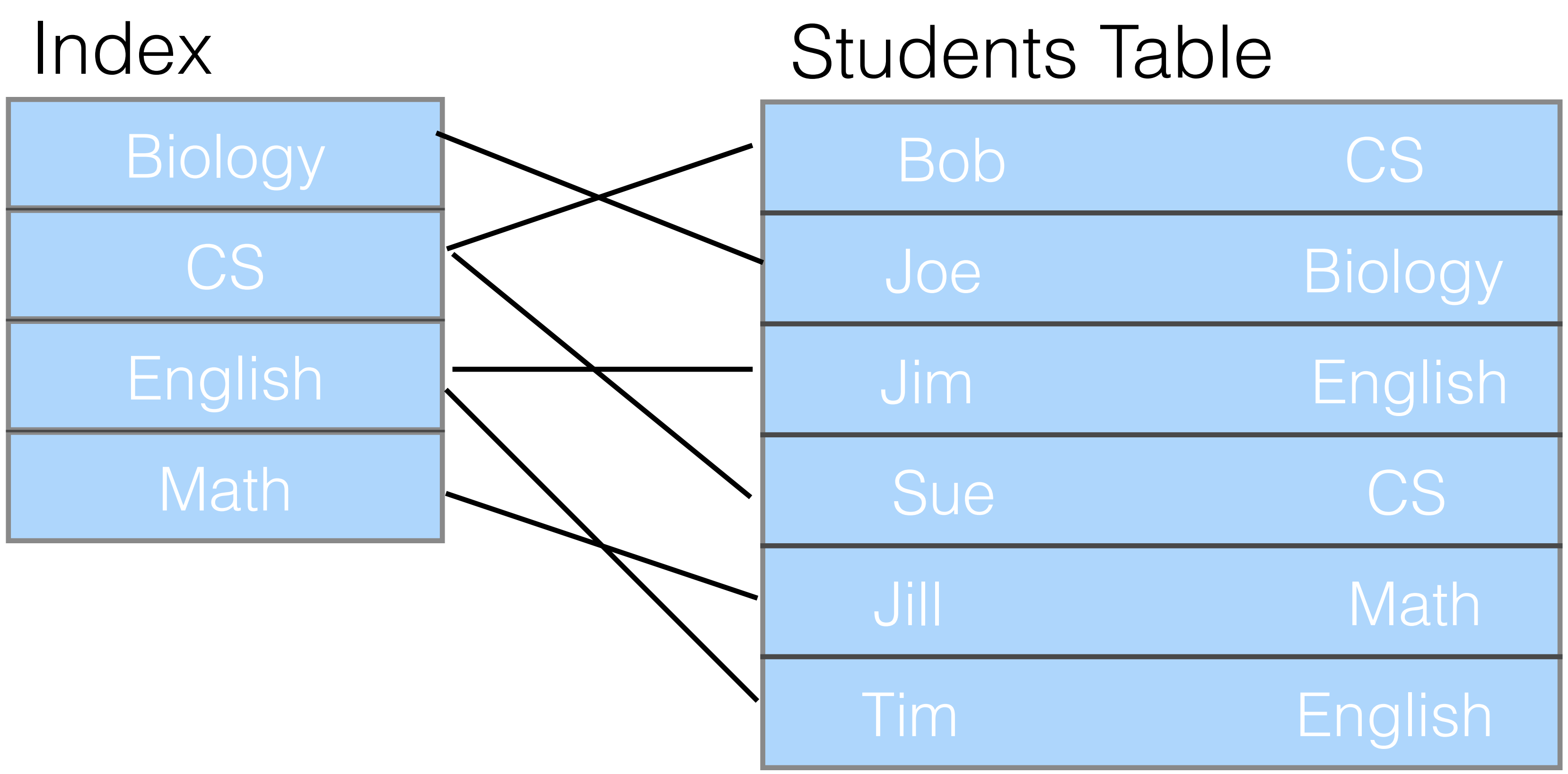
Access Pattern: A B C D A F A D G D G E D F

1	A			(1)	✓	F (1)	(1)	(1)	(1)	(1)	(1)	(0)	D(1)	(0)	Hit Rate
2		B		(1)		(0)	A (1)	(1)	(1)	(1)	(1)	(0)	(0)	F(1)	
3			C	(1)		(0)	(0)	(0)	G (1)	(1)	✓(1)	(0)	(0)	(0)	4/14
4				D(1)		(0)	(0)	✓(1)	(1)	✓(1)	(1)	E(1)	(0)	(0)	

Fill out the Index definitions on your worksheet.

Indexes

- Disk-based data structure for fast lookup by value (search key)
  - Find students in the CS department

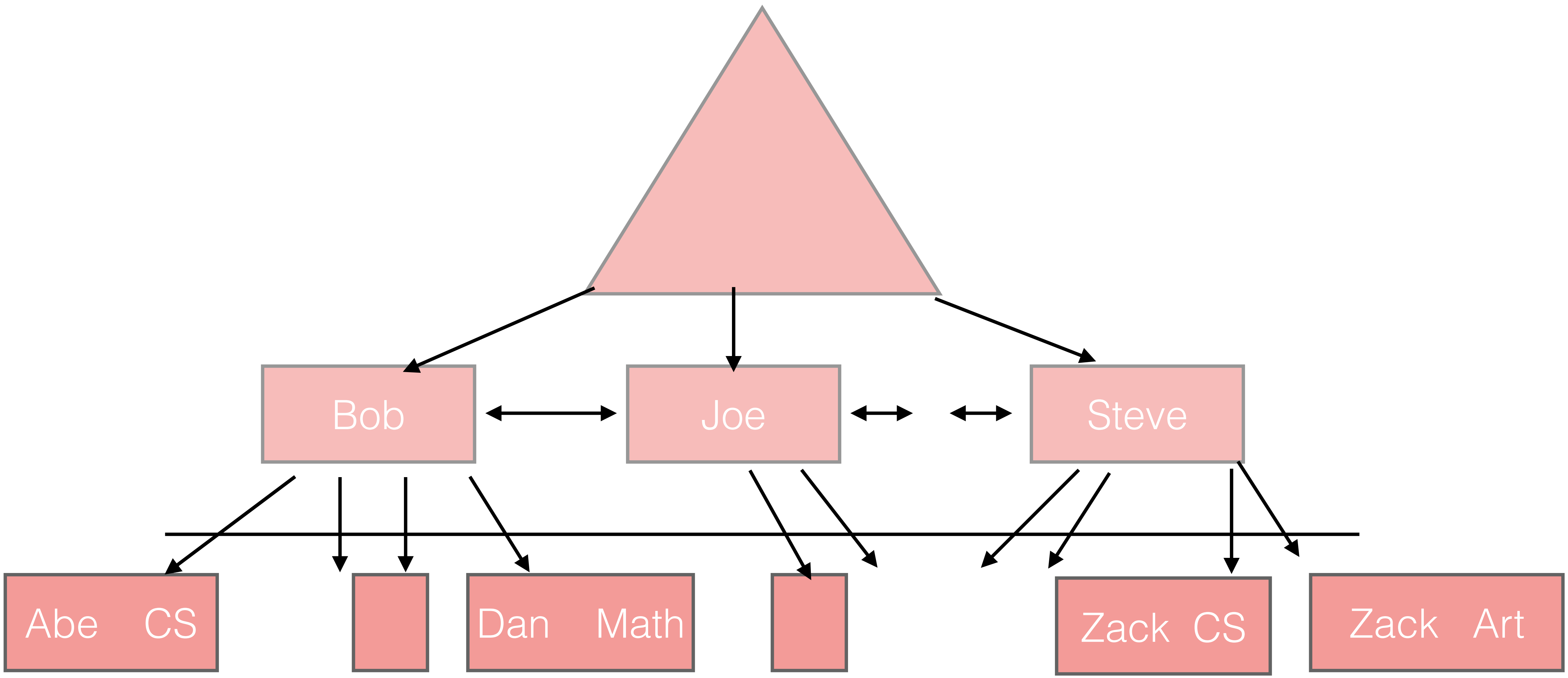


3 Ways to Store Entries in Index

- Alt 1: Actual data stored at index
  - Can have at most one index per table
- Alt 2: Store key and record ID of the matching record
- Alt 3: Store key and list of record IDs

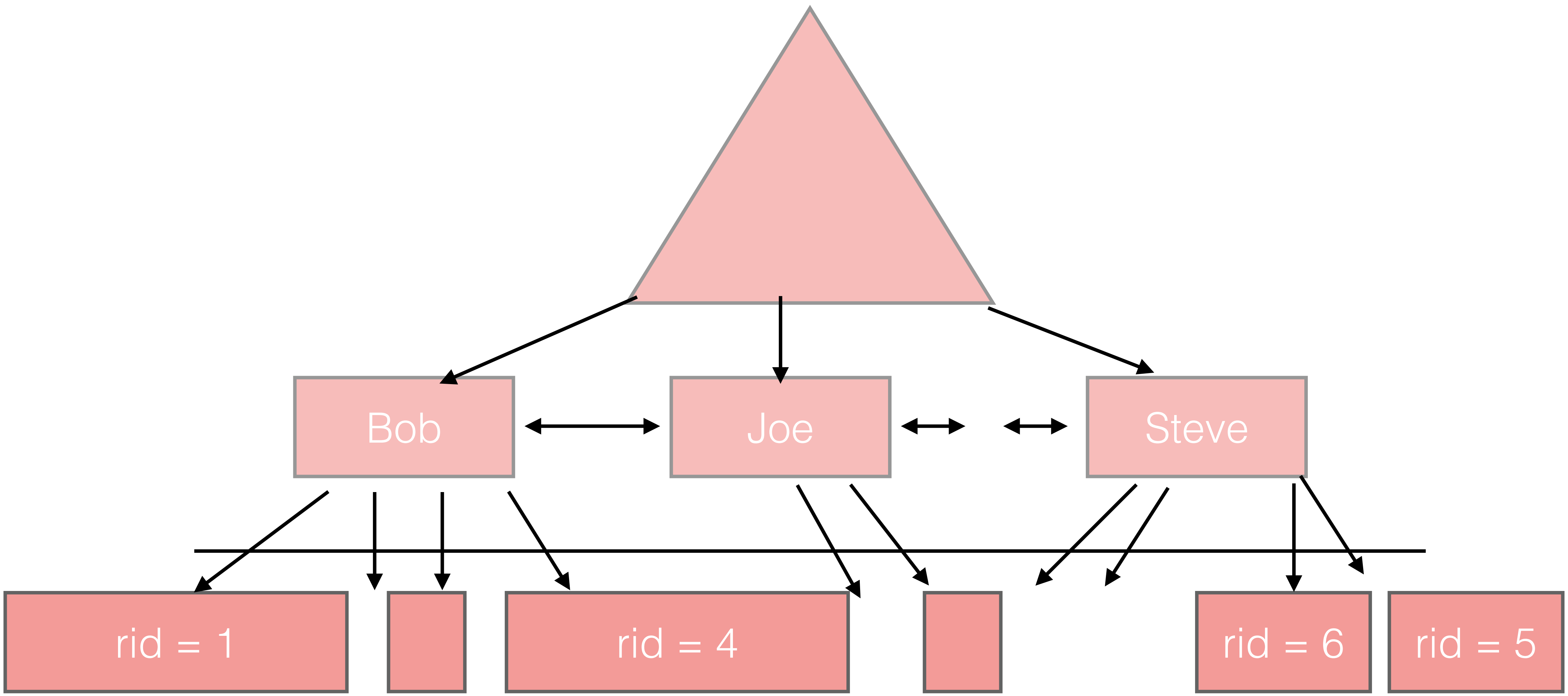
Alt 1

Actual data stored at the index



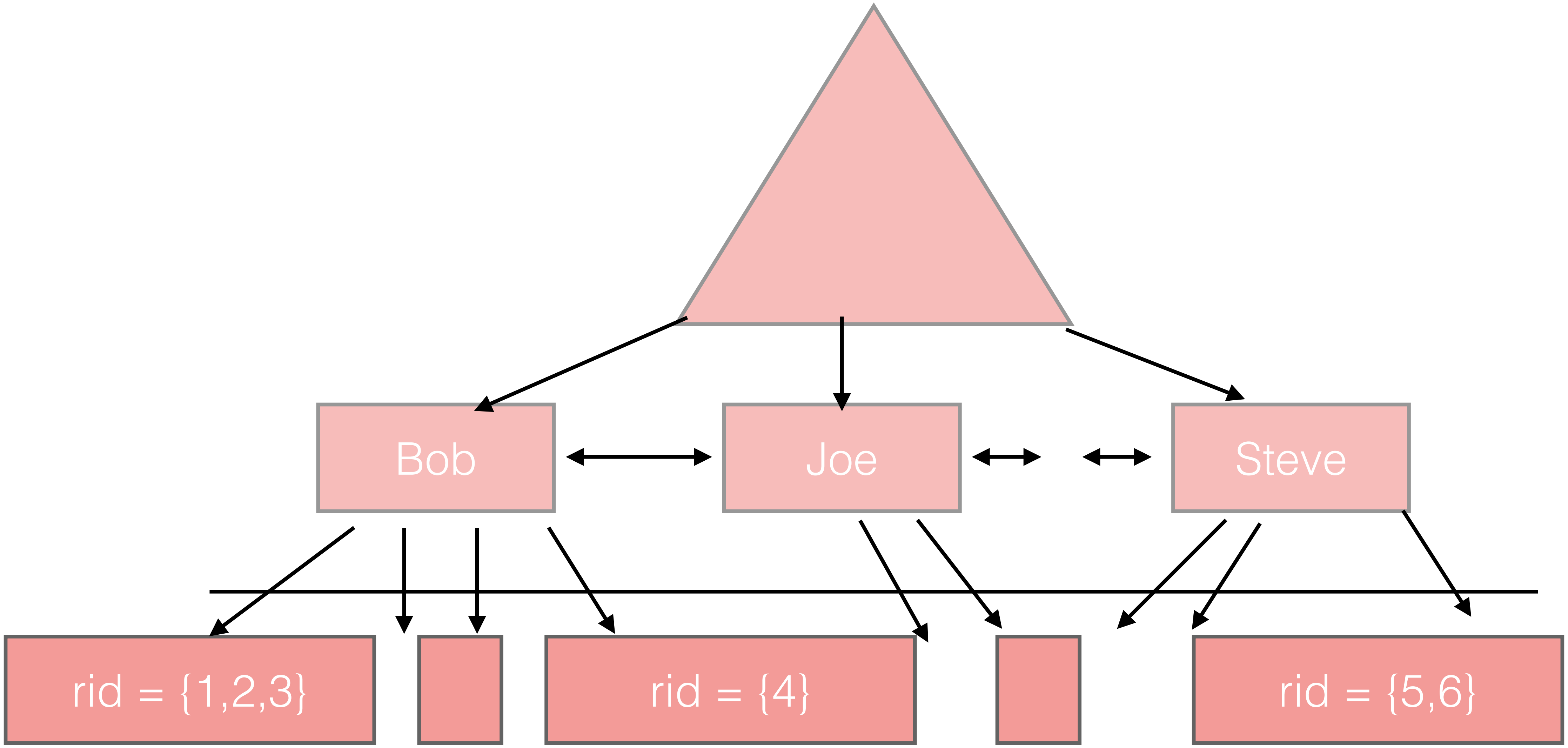
Alt 2

<key, record ID>



Alt 3

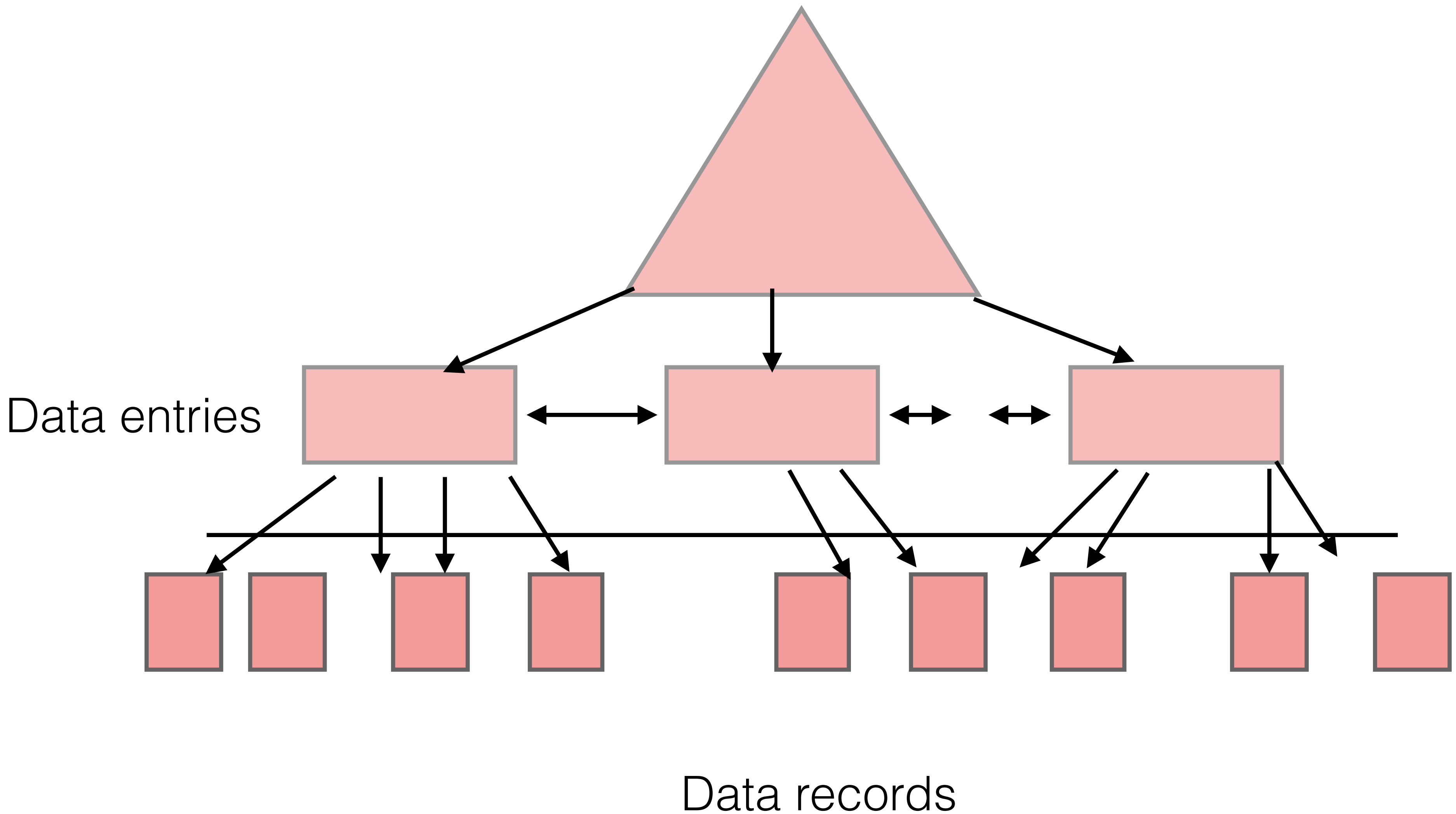
<key, list of matching record IDs>



Clustered vs Unclustered

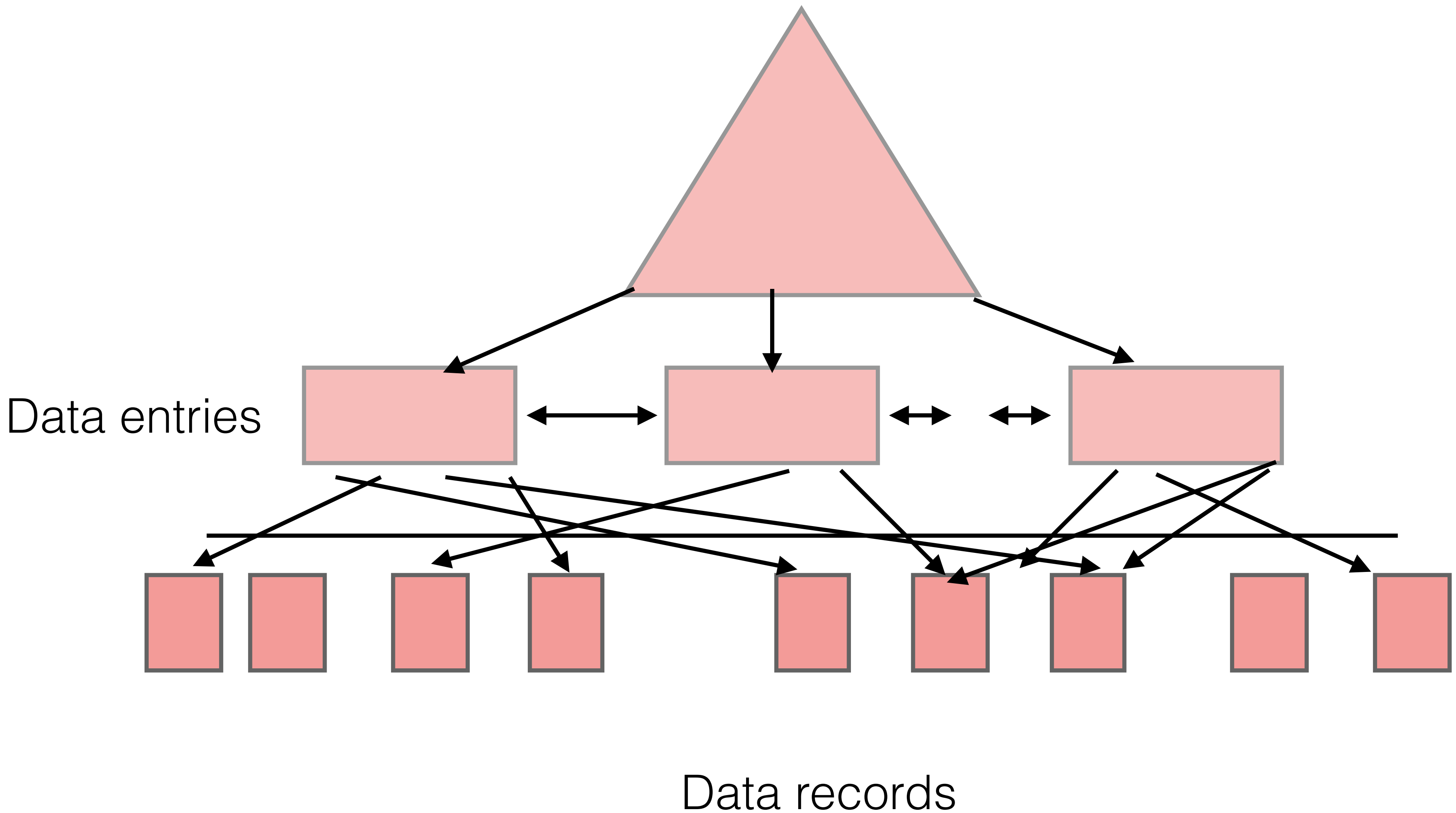
- Clustered - index data entries are stored in (approximate) order by value of search keys in data records
- Can be clustered on at most one key
- Alternative 1 is always clustered

Clustered vs Unclustered



Clustered

Clustered vs Unclustered



Unclustered

Worksheet #2

What are important factors in determining whether or not you should add an index to a table?

What are important factors in determining whether or not you should add an index to a table?

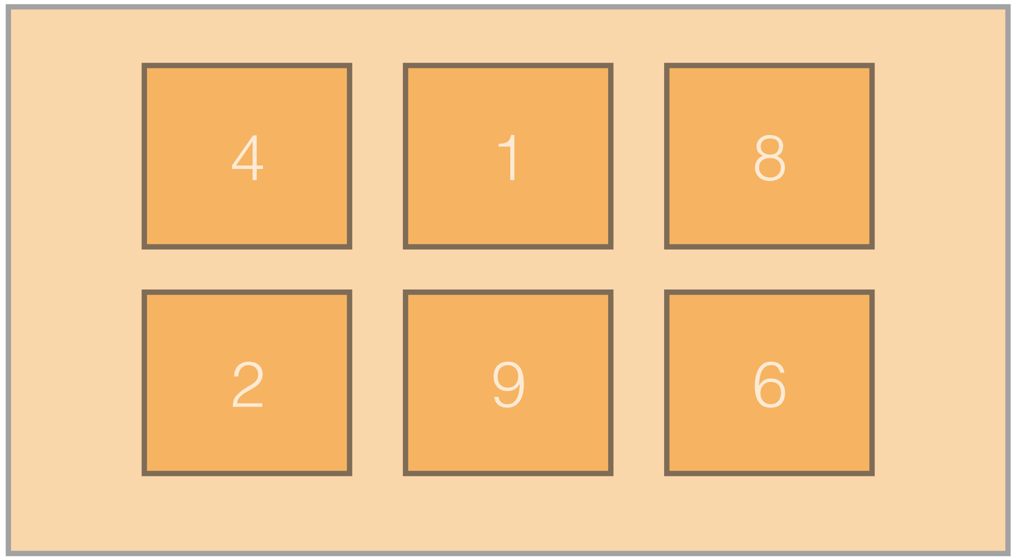
- Should know which field to cluster on (calculate I/Os based on typical queries that you'll need to run).

- Decide if you even want to cluster (high maintenance cost)

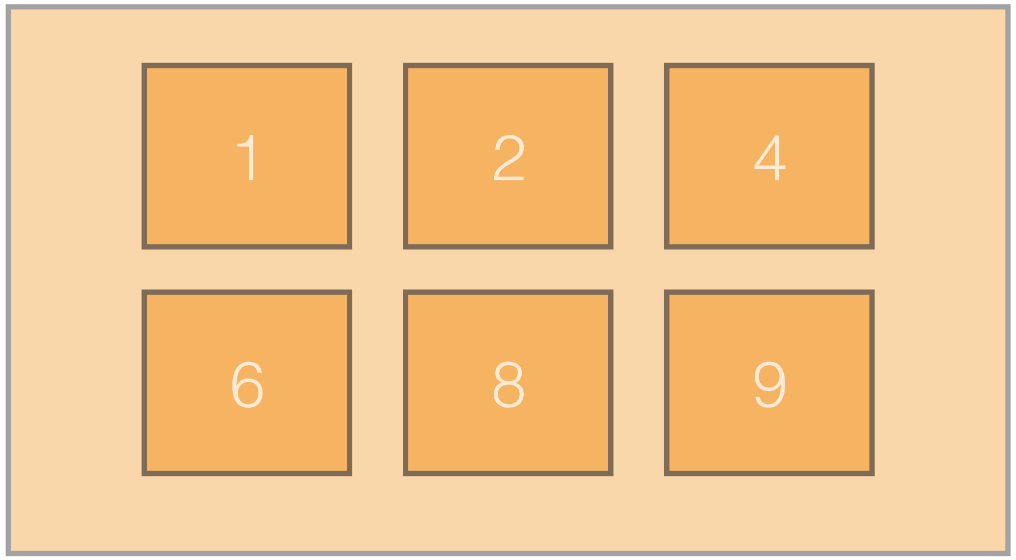
File Organization

File Organization

- Heap files: unordered set of records
- Sorted file: ordered set of records



Heap file



Sorted file

I/O Costs

Operation	Heap File	Sorted File
Scan all records	B	B
Equality Search	0.5B	log_2(B)
Range Search	B	log_2(B) + # pages matched
Insert	2	log_2(B)+ (B/2) *2
Delete	0.5B+1	log_2(B)+ (B/2) *2

Consider the table Enrolled(sid, course, grade) with 500 pages, 6,000 tuples and the query `SELECT * FROM Enrolled where sid > 4500`.
Assume SIDs are unique and range from 0 to 6000.

- How many I/Os would this query take if the table was stored in a heap file?

Consider the table Enrolled(sid, course, grade) with 500 pages, 6,000 tuples and the query `SELECT * FROM Enrolled where sid > 4500.`
Assume SIDs are unique and range from 0 to 6000.

- How many I/Os would this query take if the table was stored in a heap file?

B = 500

Consider the table Enrolled(sid, course, grade) with 500 pages, 6,000 tuples and the query `SELECT * FROM Enrolled where sid > 4500.`
Assume SIDs are unique and range from 0 to 6000.

- How many I/Os would this take if the table was stored in a sorted file sorted by grade?

Consider the table Enrolled(sid, course, grade) with 500 pages, 6,000 tuples and the query `SELECT * FROM Enrolled where sid > 4500.`
Assume SIDs are unique and range from 0 to 6000.

- How many I/Os would this take if the table was stored in a sorted file sorted by grade?

B = 500

Consider the table Enrolled(sid, course, grade) with 500 pages, 6,000 tuples and the query `SELECT * FROM Enrolled where sid > 4500.`
Assume SIDs are unique and range from 0 to 6000.

- How many I/Os would this take if the table was stored in a sorted file sorted by SID?

Consider the table Enrolled(sid, course, grade) with 500 pages, 6,000 tuples and the query `SELECT * FROM Enrolled where sid > 4500.`
Assume SIDs are unique and range from 0 to 6000.

- How many I/Os would this take if the table was stored in a sorted file sorted by SID?

log_2(500) + ¼ * 500

True or **False**? Given the table Students(sid, gpa, age), a hash index on gpa will significantly increase the performance of the following query:

```
SELECT * FROM Students WHERE age > 20;
```

True or **False**? Given the table Students(sid, gpa, age), a hash index on gpa will significantly increase the performance of the following query:

```
SELECT * FROM Students WHERE age > 20;
```

False

True or **False**? Given the table Students(sid char(20), gpa float, age integer), a clustered tree based index on gpa will increase the performance of the following query:

```
SELECT * FROM Students where age > 20
AND gpa > 3.5;
```

True or **False**? Given the table Students(sid char(20), gpa float, age integer), a clustered tree based index on gpa will increase the performance of the following query:

```
SELECT * FROM Students where age > 20
AND gpa > 3.5;
```

True