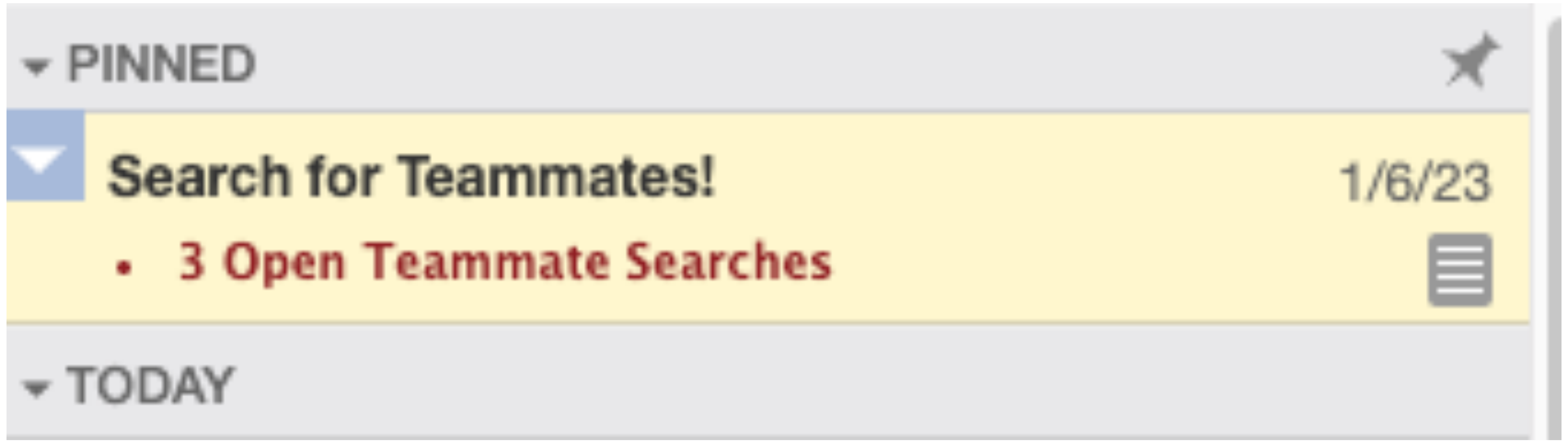
Tutorial

Table & Buffer management

Database System Technology - Niv Dayan

If you don’t have a group



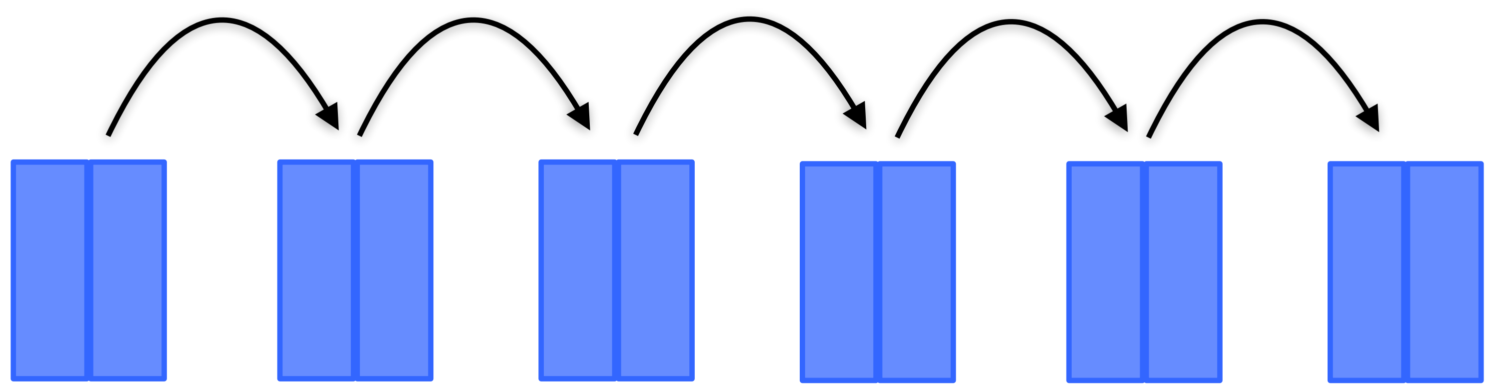
Each group should have 3 members

Question 1

Consider a table allocated as a linked list of database pages.

What scan throughout (in MB/s) would you expect on disk? How about an SSD?

How can we better structure this table to improve throughput?



Question 1

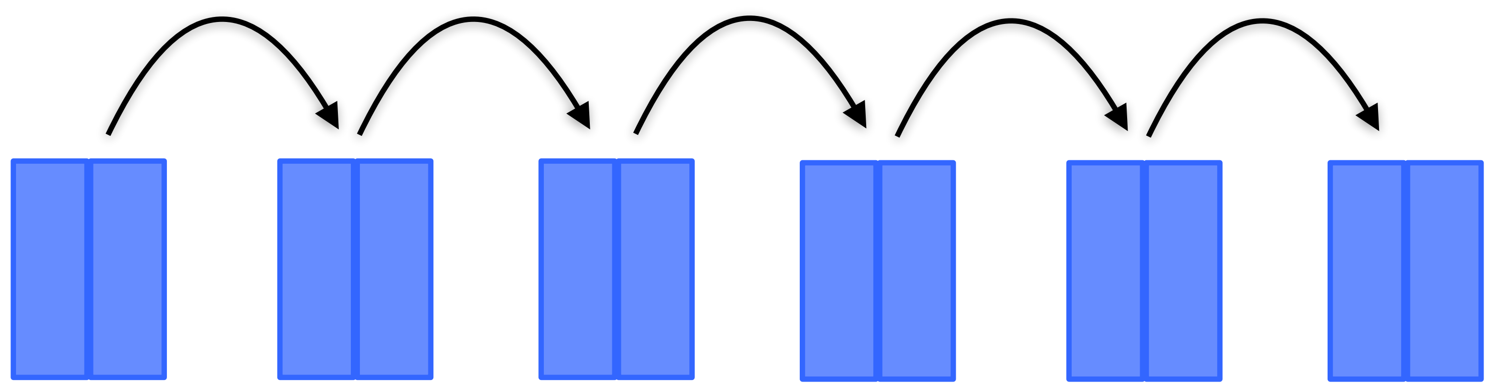
Consider a table allocated as a linked list of database pages.

What scan throughout (in MB/s) would you expect on disk? How about an SSD?

A disk I/O takes 10ms to read 4KB pages. Throughput: 0.4 MB/s.

An SSD I/O takes 100us to read 4KB pages. Throughput: 40 MB/s.

How can we better structure this table to improve throughput?



Question 1

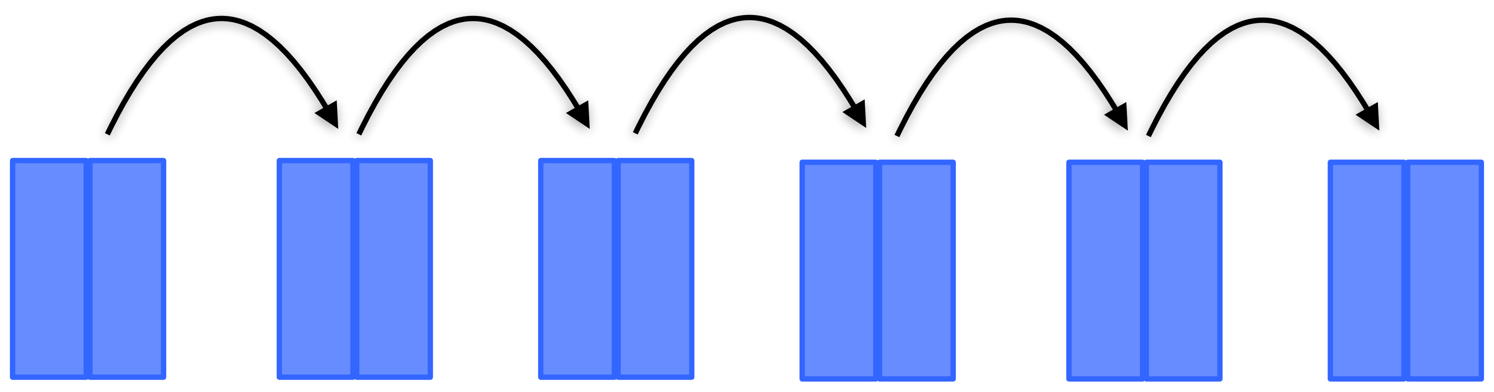
Consider a table allocated as a linked list of database pages.

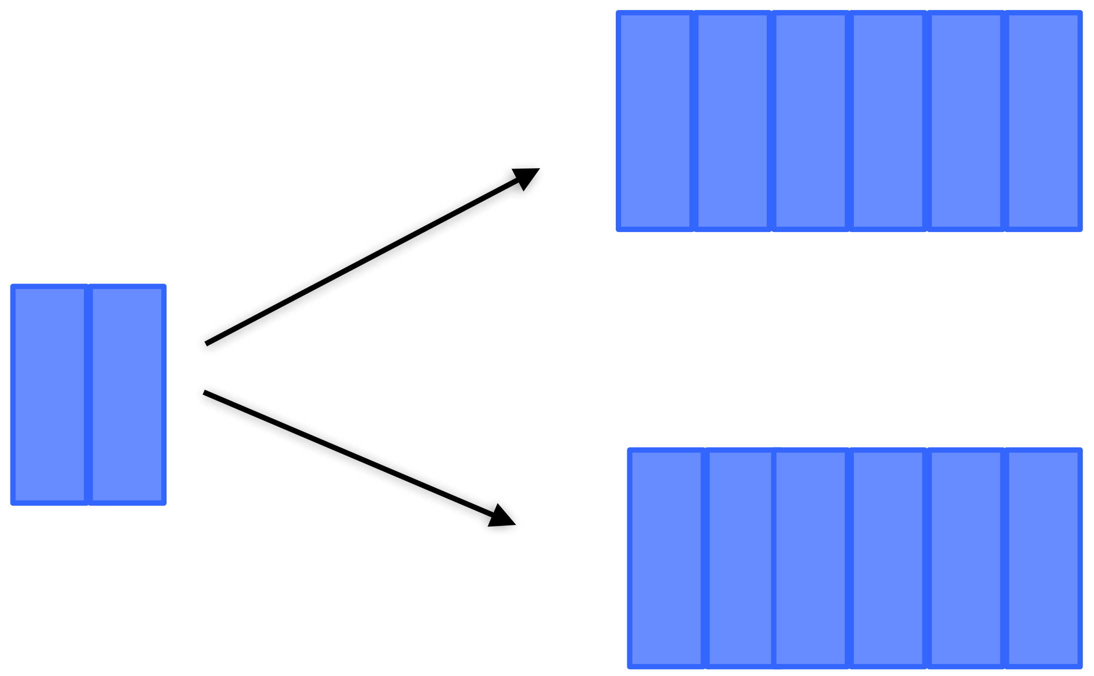
What scan throughout (in MB/s) would you expect on disk? How about an SSD?

A disk I/O takes 10ms to read 4KB pages. Throughput: 0.4 MB/s.

An SSD I/O takes 100us to read 4KB pages. Throughput: 40 MB/s.

How can we better structure this table to improve throughput?





Using a directory of extents

Question 2

Consider a database that's subject to uniformly random reads.

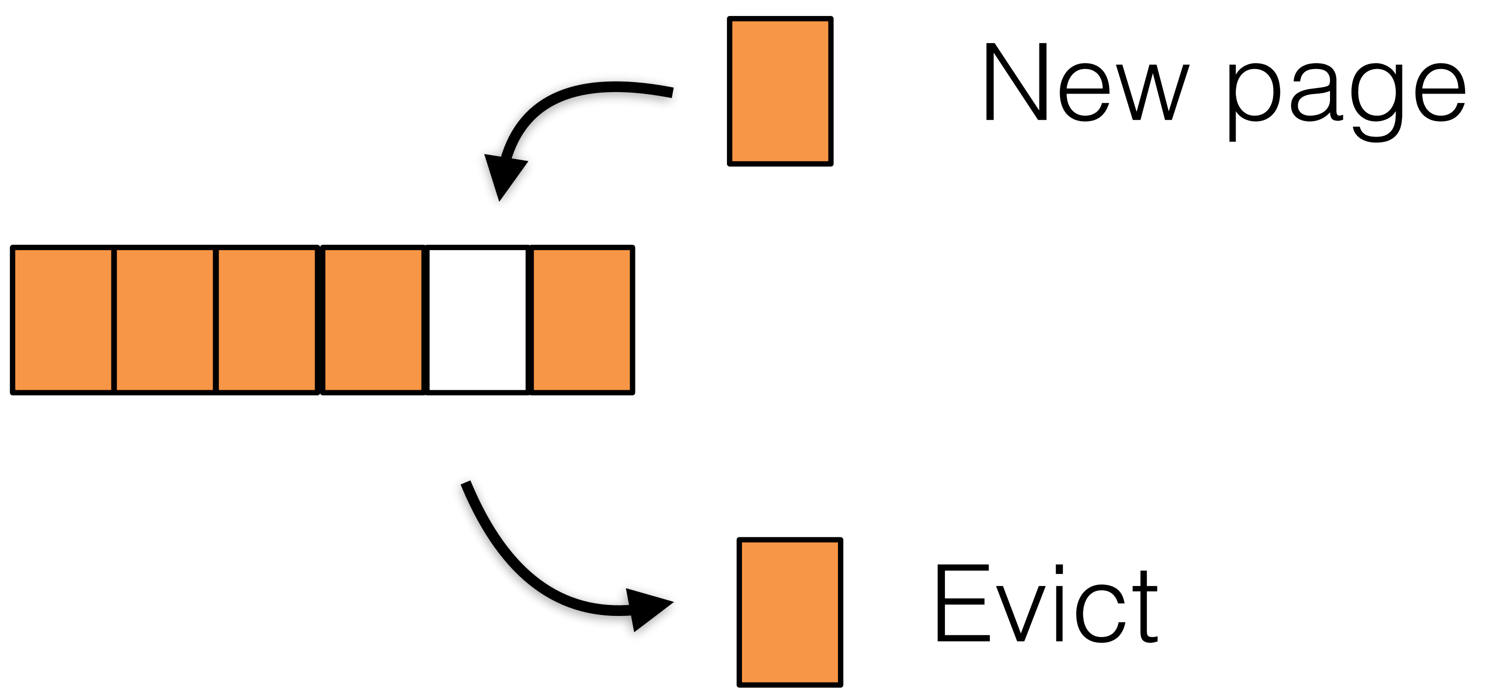
What's the best buffer management strategy for this case and why?

Would you also use this strategy if the reads are heavily skewed? Why or why not?

Question 2

Consider a database that's subject to uniformly random reads.

What's the best buffer management strategy for this case and why?



Random eviction.

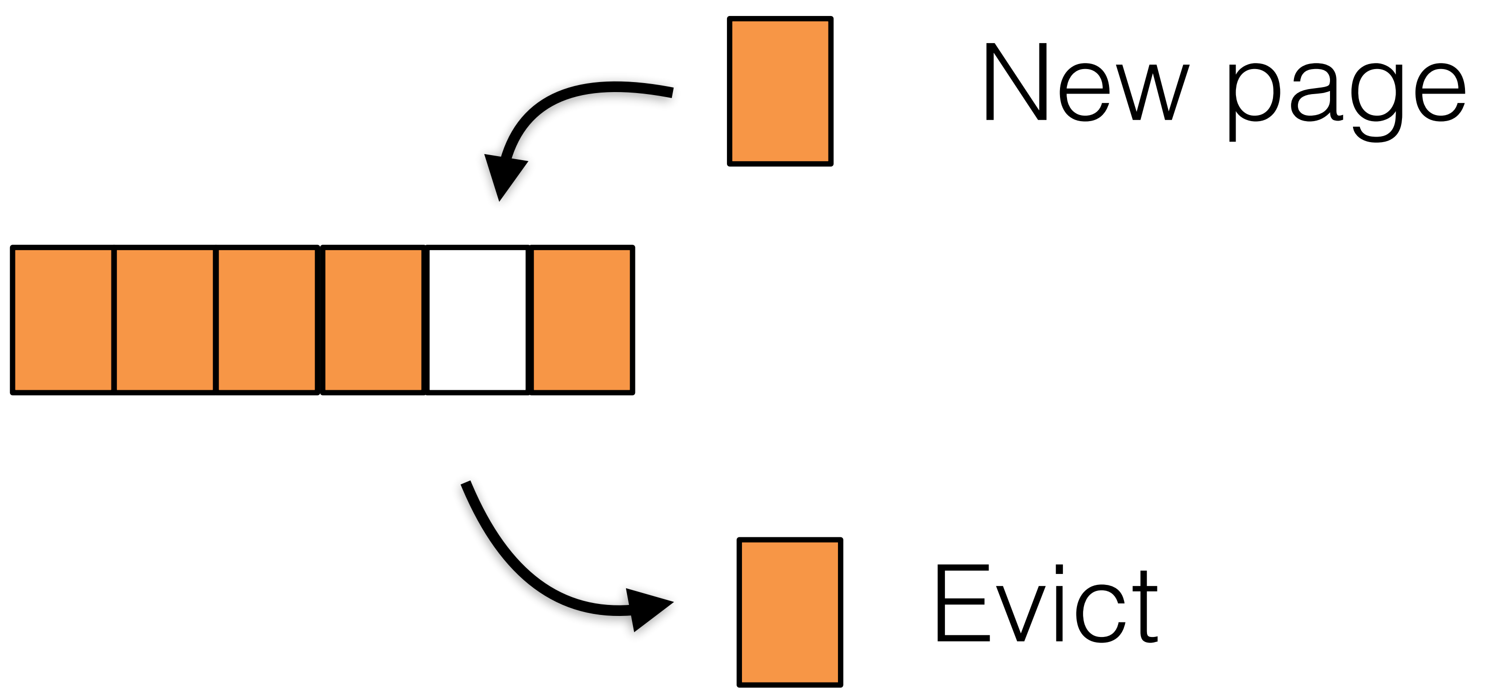
(1) Allows 100% hash table utilization.
(2) Fastest since hitting reads incur no collisions
(3) There is no metadata so we
can set the hash table to be slightly larger.

Would you also use this strategy if the reads are heavily skewed? Why or why not?

Question 2

Consider a database that's subject to uniformly random reads.

What's the best buffer management strategy for this case and why?



Random eviction.

(1) Allows 100% hash table utilization.
(2) Fastest since hitting reads incur no collisions
(3) There is no metadata so we
can set the hash table to be slightly larger.

Would you also use this strategy if the reads are heavily skewed? Why or why not?

No because it might evict hot pages. Clock or LRU are better for this.

Question 3

Give pros and cons for managing buffer pool at the unit of entries rather than pages.

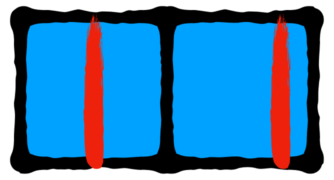
Pros:

Cons:

Question 3

Give pros and cons for managing buffer pool at the unit of entries rather than pages.

Pros: (1) Some hot entries may exist on otherwise cold pages. Buffering entries allows us to fit more of the hot working set into memory. This is great for reads.

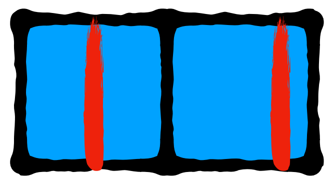


Cons:

Question 3

Give pros and cons for managing buffer pool at the unit of entries rather than pages.

Pros: (1) Some hot entries may exist on otherwise cold pages. Buffering entries allows us to fit more of the hot working set into memory. This is great for reads.



Cons:

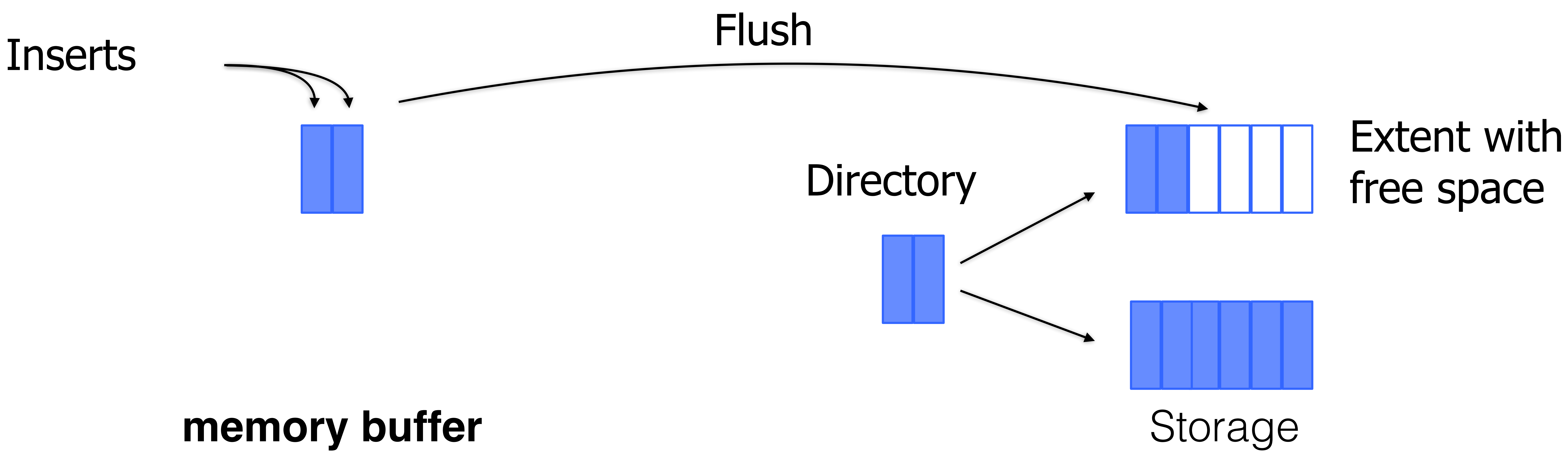
(1) Evicting a dirty entry requires reading its page first and then rewriting it

(2) Heavier metadata overheads (e.g., to track which page each entry belongs to, etc)

(3) Inefficient for short scans as we lose locality

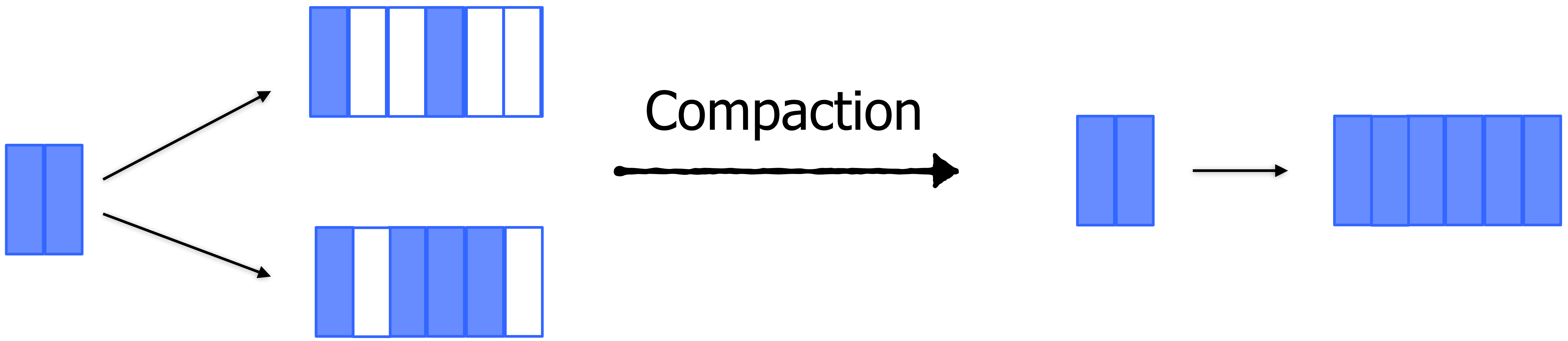
Question 4

Recall how we can buffer insertions in memory until a page fills up and flush



Question 4

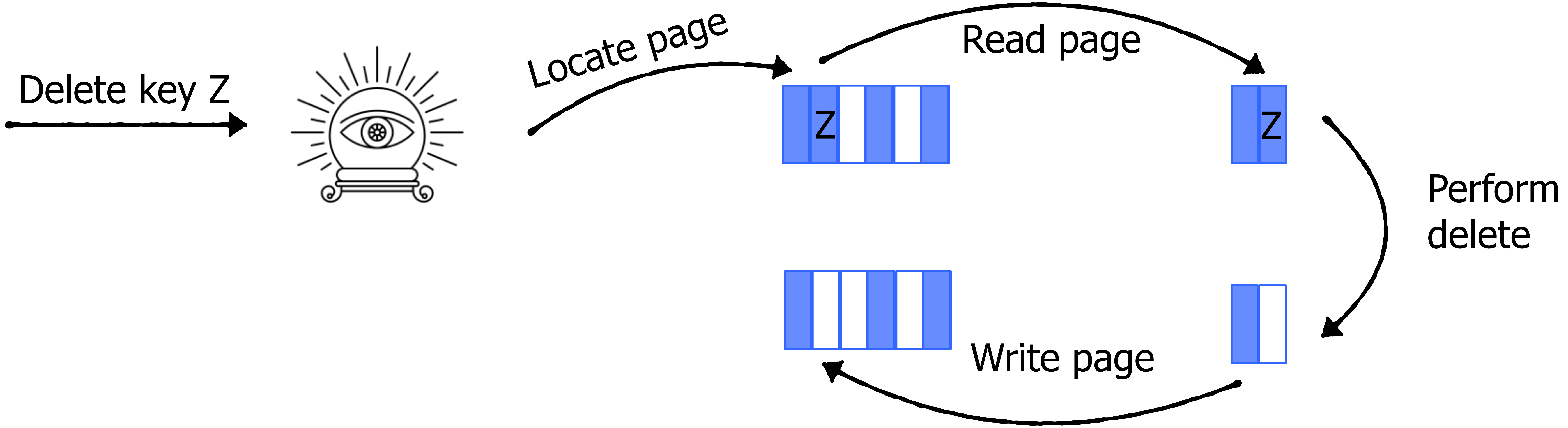
Consider a workload with only deletes and insertions. The deletes create “holes”. Whenever there are X holes, we compact the whole table to be stored contiguously again.



Question 4

Consider a workload with only deletes and insertions. The deletes create “holes”. Whenever there are X holes, we compact the whole table to be stored contiguously again.

All data entries in the table are fixed-length. We have on oracle that tells us on which page the entry we want to delete is, thus obviating the need to scan the table to find it. (We’ll learn how to implement this oracle using an index next week).



Question 4

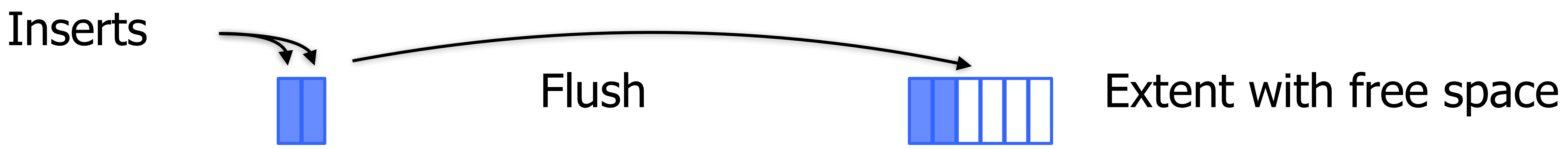
Consider a workload with only deletes and insertions. The deletes create “holes”. Whenever there are X holes, we compact the whole table to be stored contiguously again.

All data entries in the table are fixed-length. We have on oracle that tells us on which page the entry we want to delete is, thus obviating the need to scan the table to find it. (We’ll learn how to implement this oracle using an index next week).

Analyze the amortized worst-case cost of insertions and deletes using big O notations in terms of N, B and X. Let N denote the number of valid (non-deleted) entries. Assume X is fixed and that N>X.

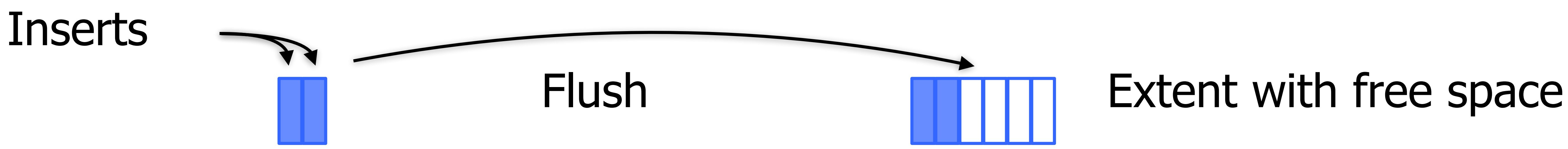
Question 4

Insertion cost is O(1/B) as each write I/O flushes B entries to storage

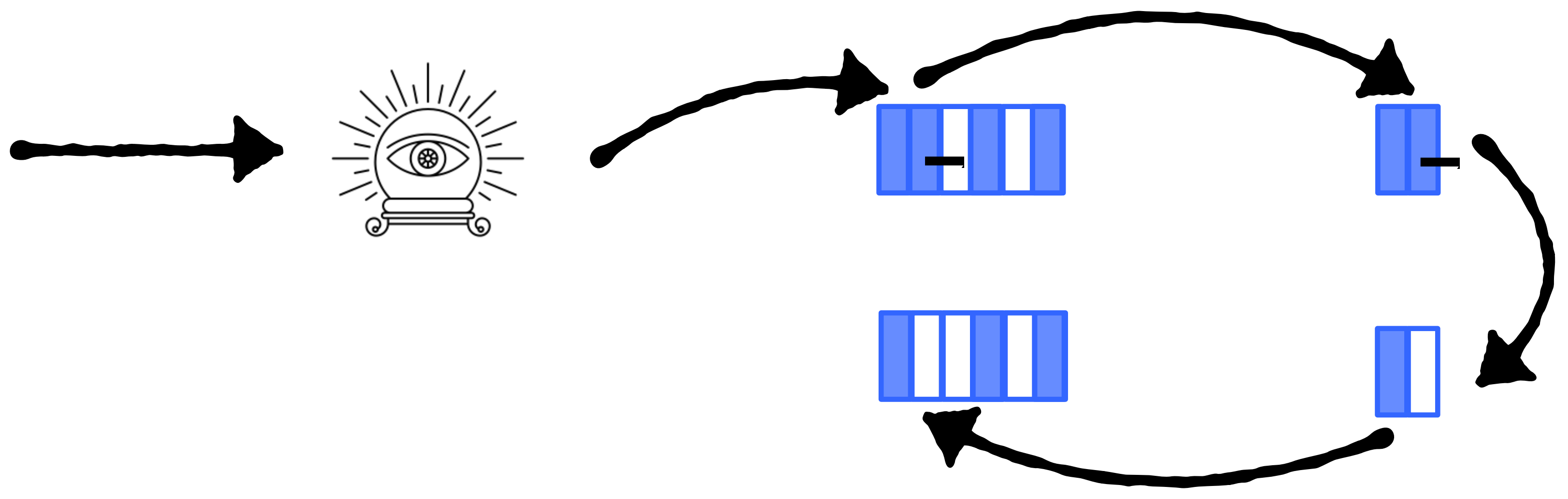


Question 4

Insertion cost is O(1/B) as each write I/O flushes B entries to storage

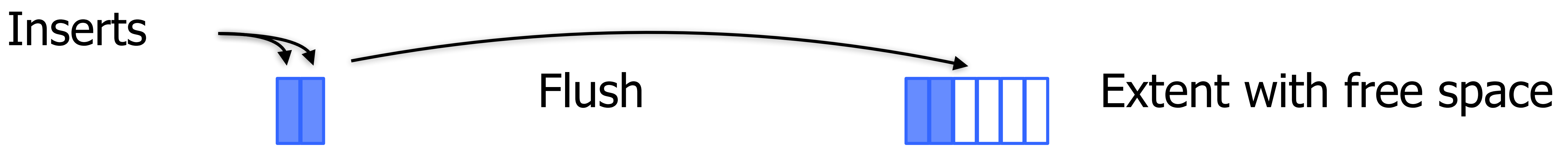


A delete entails 1 read/write I/O to create a hole in a given page.

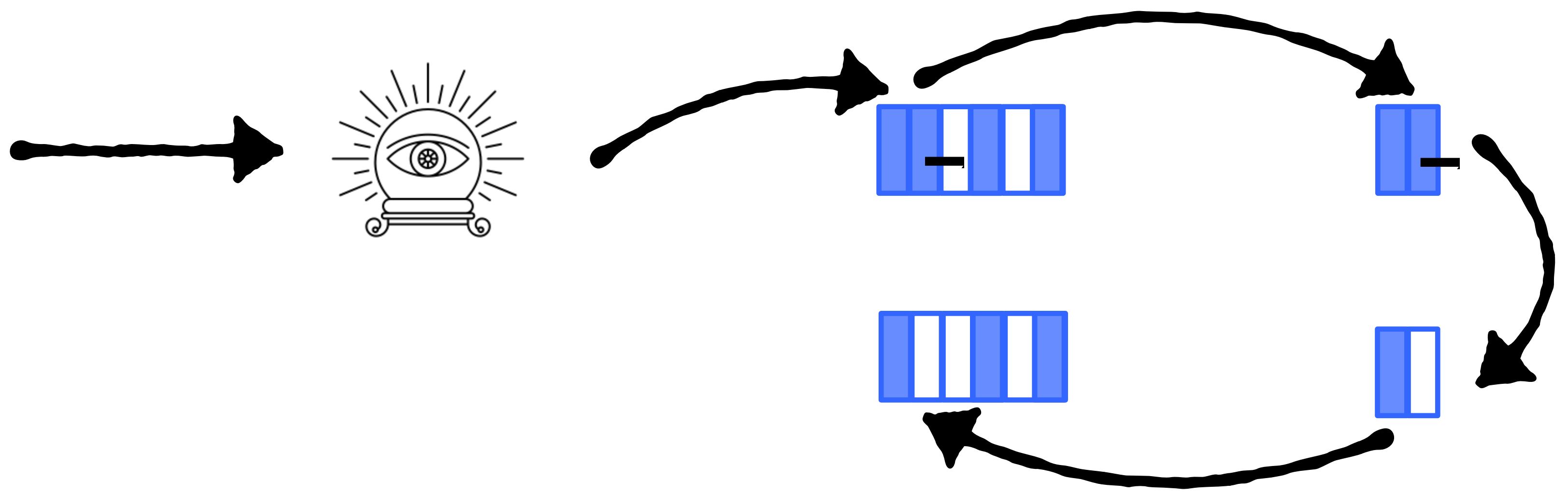


Question 4

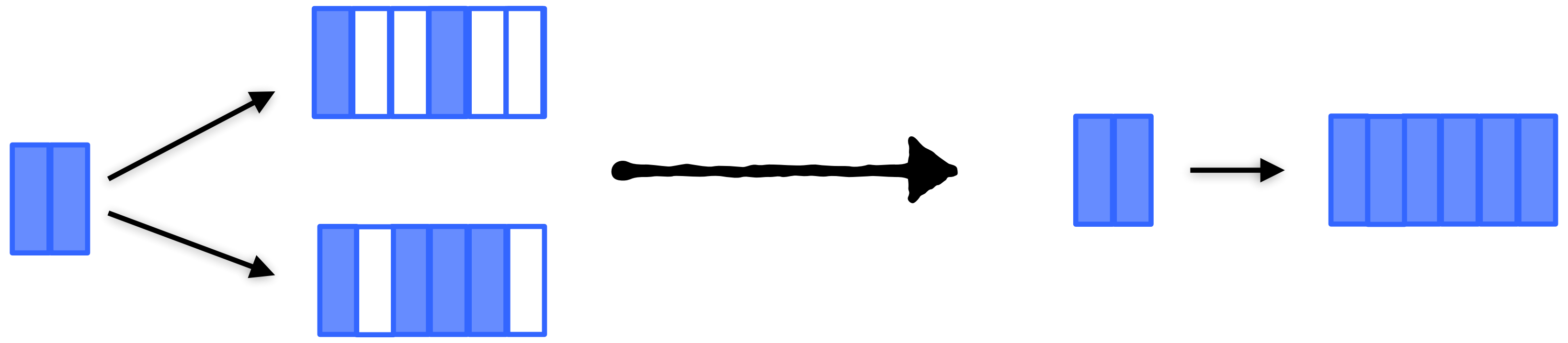
Insertion cost is O(1/B) as each write I/O flushes B entries to storage



A delete entails 1 read/write I/O to create a hole in a given page.

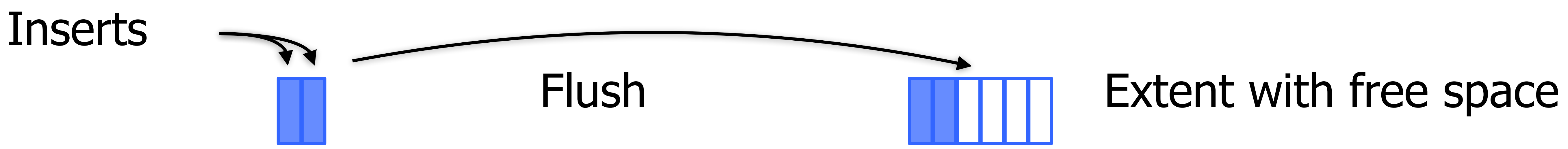


For every X deletes, we must read & write ≈(N+X)/B pages. This costs O((N+X)/(B*X)) I/Os per delete.

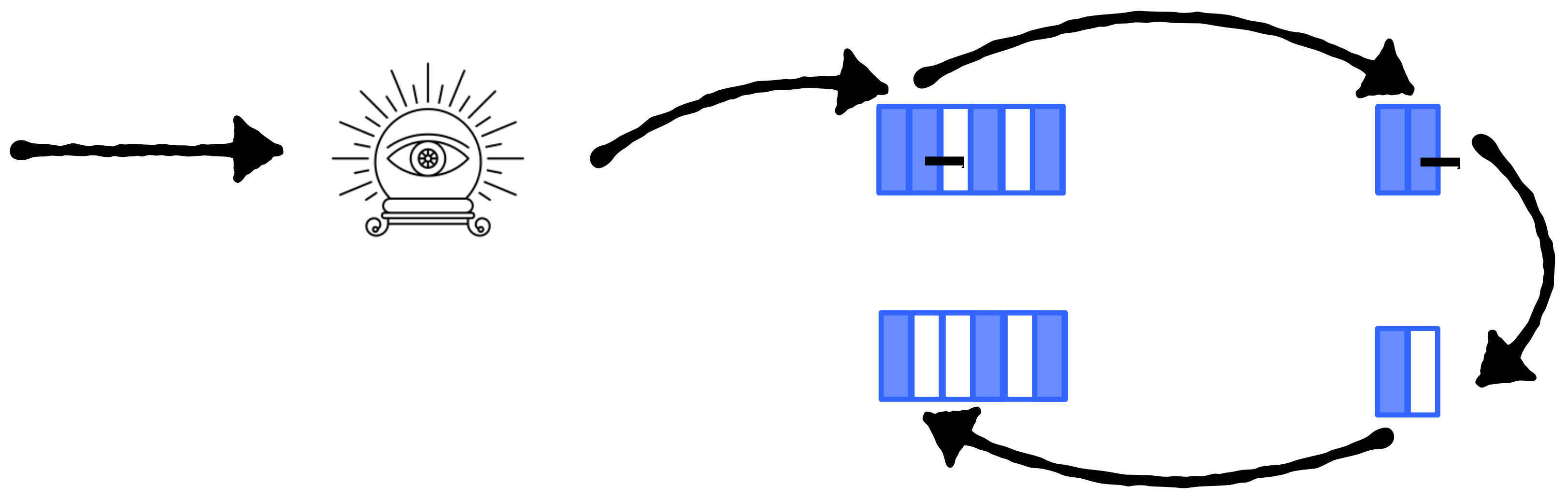


Question 4

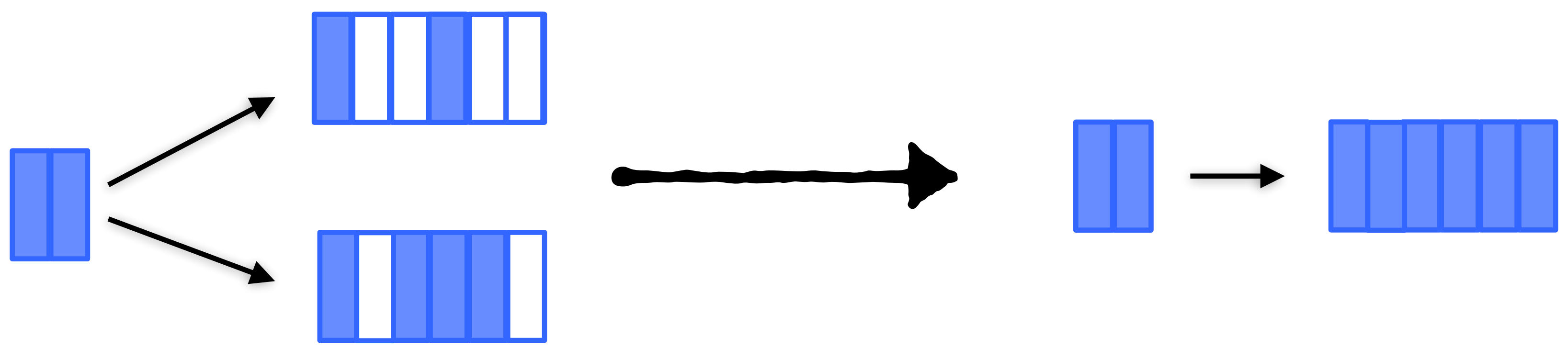
Insertion cost is O(1/B) as each write I/O flushes B entries to storage



A delete entails 1 read/write I/O to create a hole in a given page.



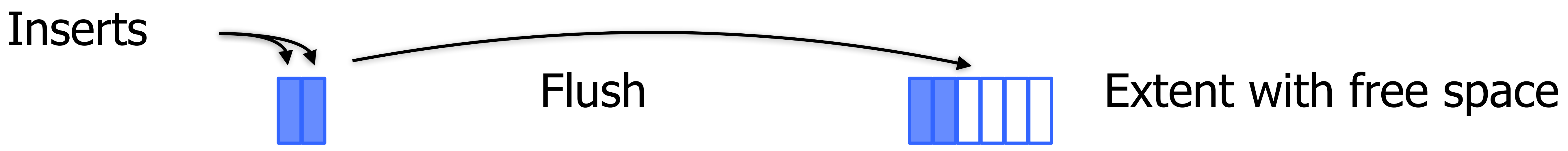
For every X deletes, we must read & write ≈(N+X)/B pages. This costs O((N+X)/(B*X)) I/Os per delete.



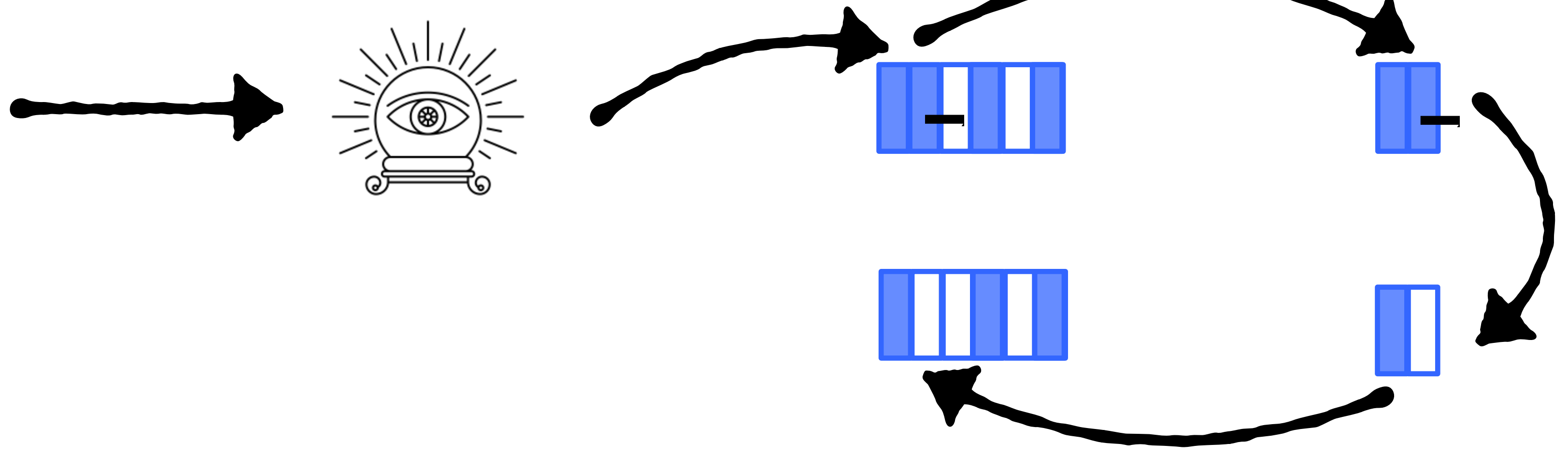
Total delete cost is O(1+(N+X)/(B*X)), or more simply O(1+N/(B*X)) assuming N>X

Question 4

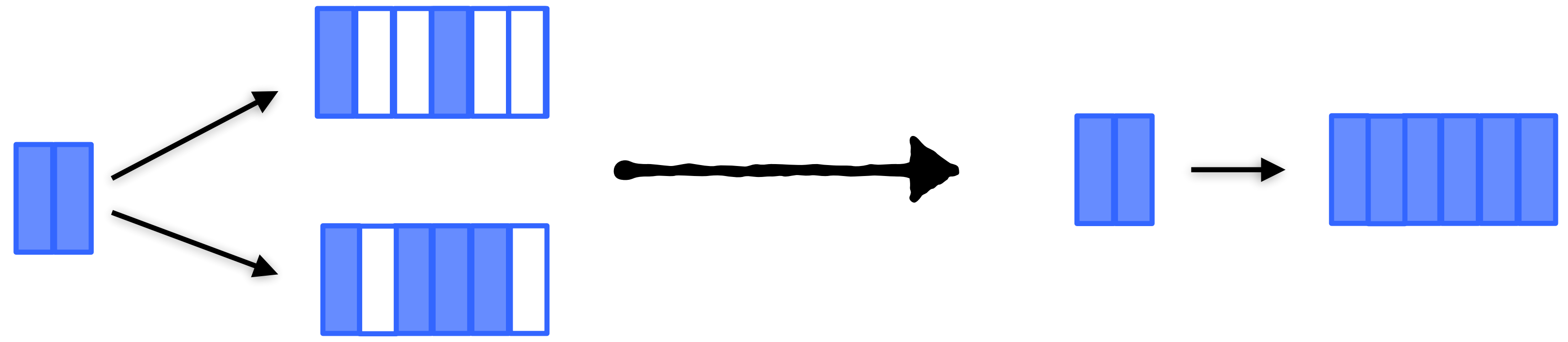
Insertion cost is $O(1/B)$ as each write I/O flushes B entries to storage



A delete entails 1 read/write I/O to create a hole in a given page.



For every X deletes, we must read & write $\approx(N+X)/B$ pages. This costs $O((N+X)/(B*X))$ I/Os per delete.



Total delete cost is $O(1+(N+X)/(B*X))$, or more simply $O(1+N/(B*X))$ assuming $N>X$

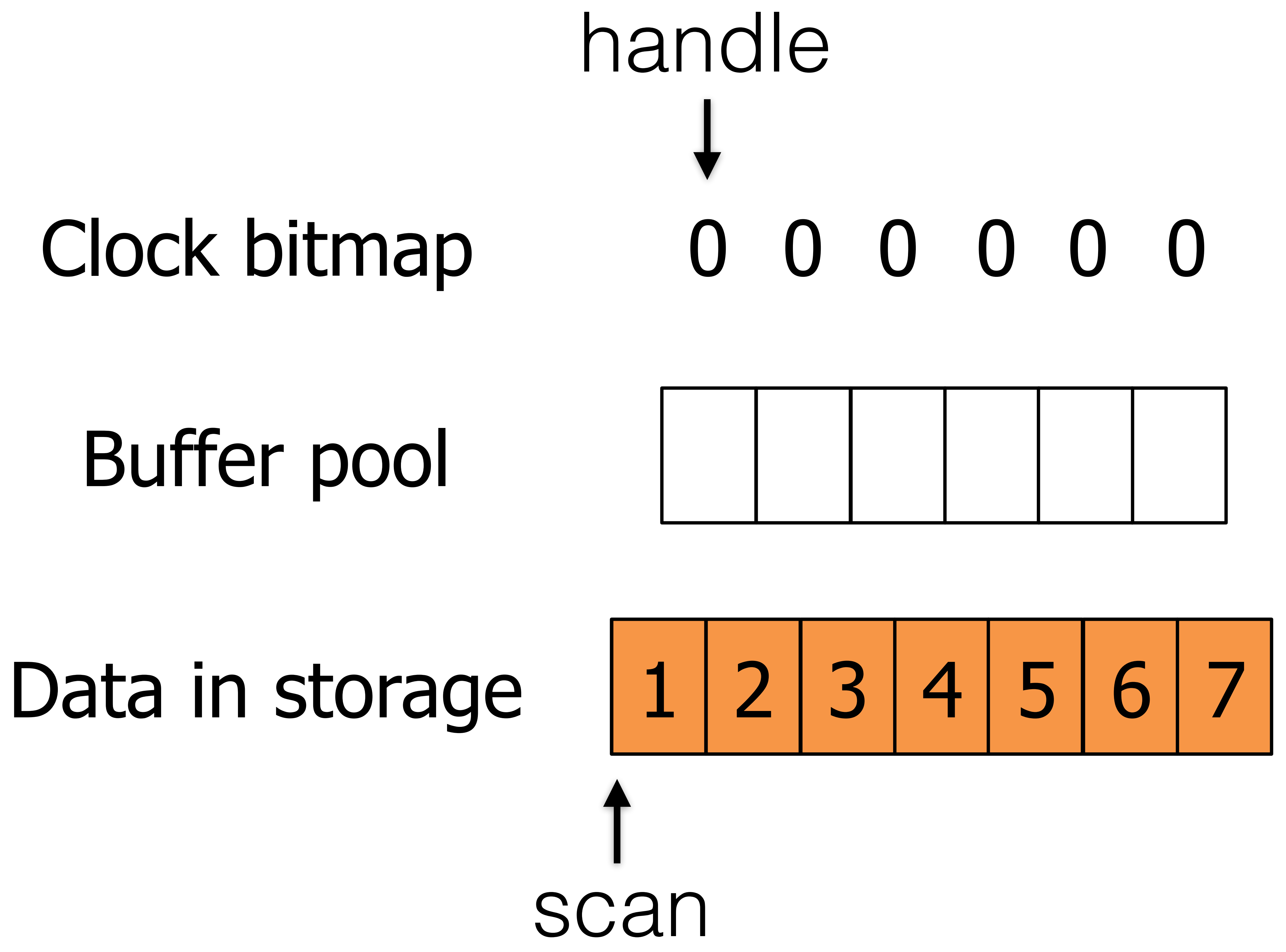
Side note: without the oracle, we would need to scan the table to locate the entry to be deleted at a cost of $O(N/B)$ per delete.

Question 5

What happens to LRU or Clock in the presence of a sequential read that’s larger than the buffer size? How can we address this problem?

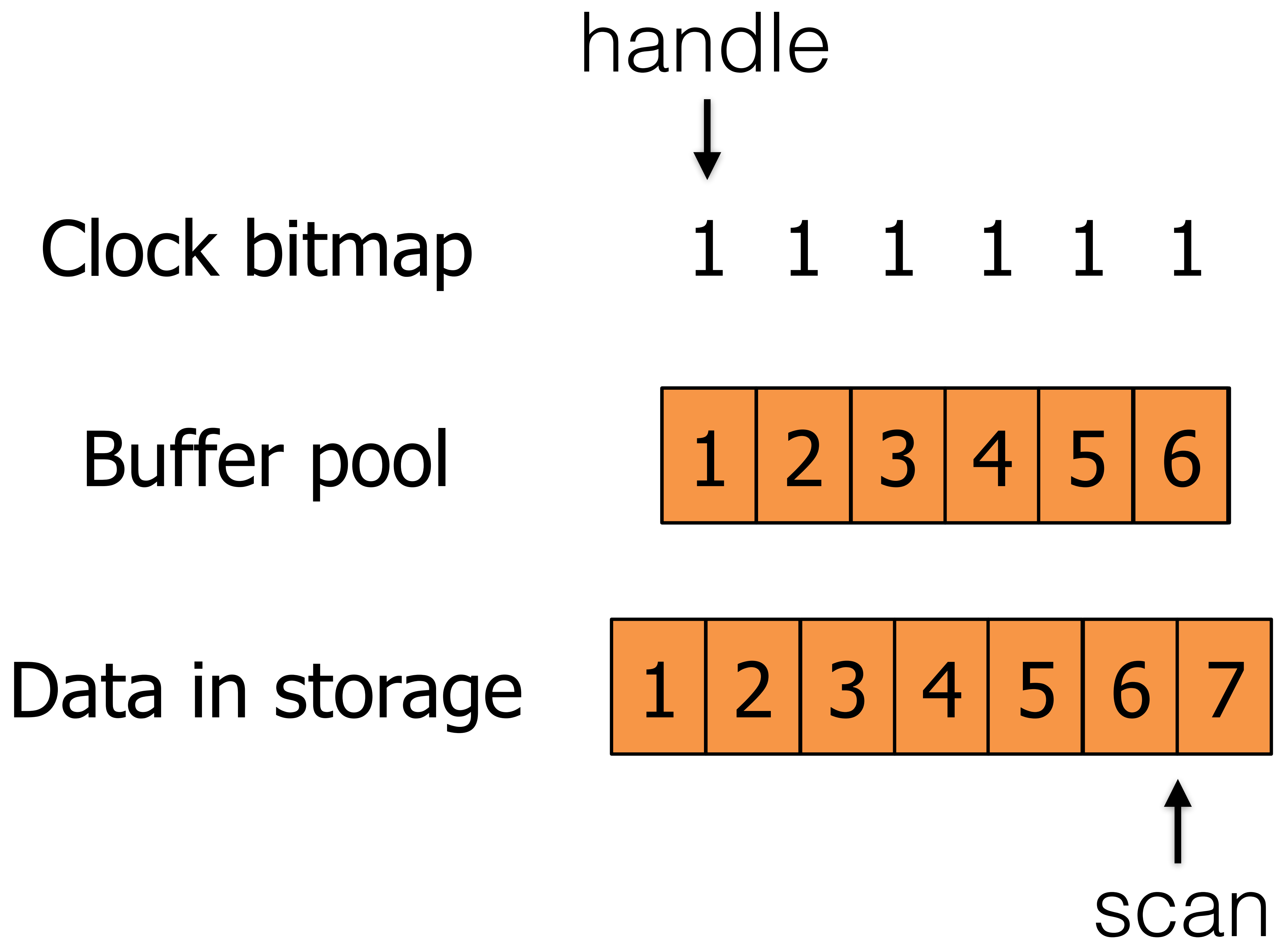
Question 5 - **Sequential Flooding of Clock**

Suppose the DB is repeatedly scanning data larger than the buffer pool



Question 5 - Sequential Flooding of Clock

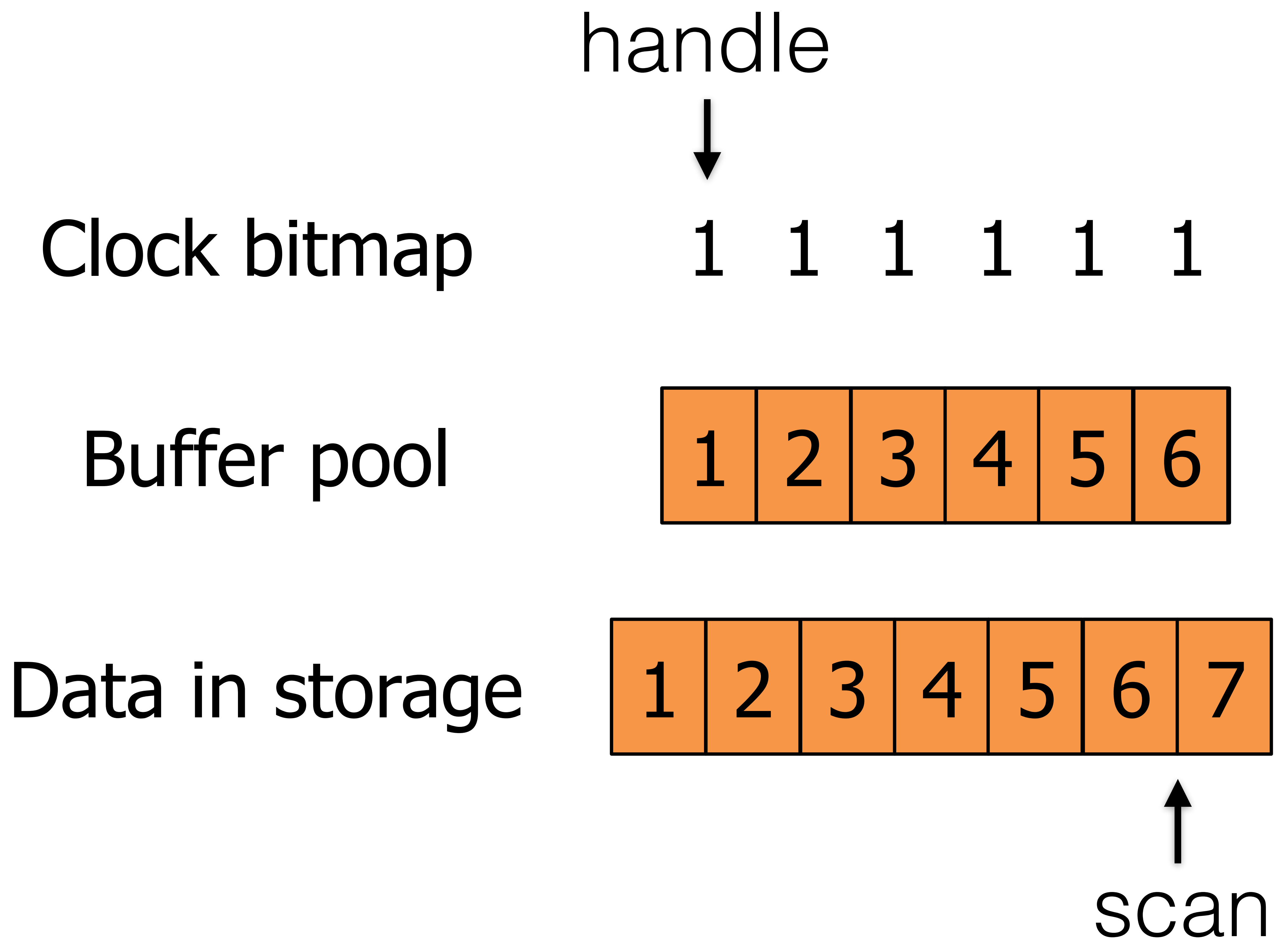
Suppose the DB is repeatedly scanning data larger than the buffer pool



Question 5 - Sequential Flooding of Clock

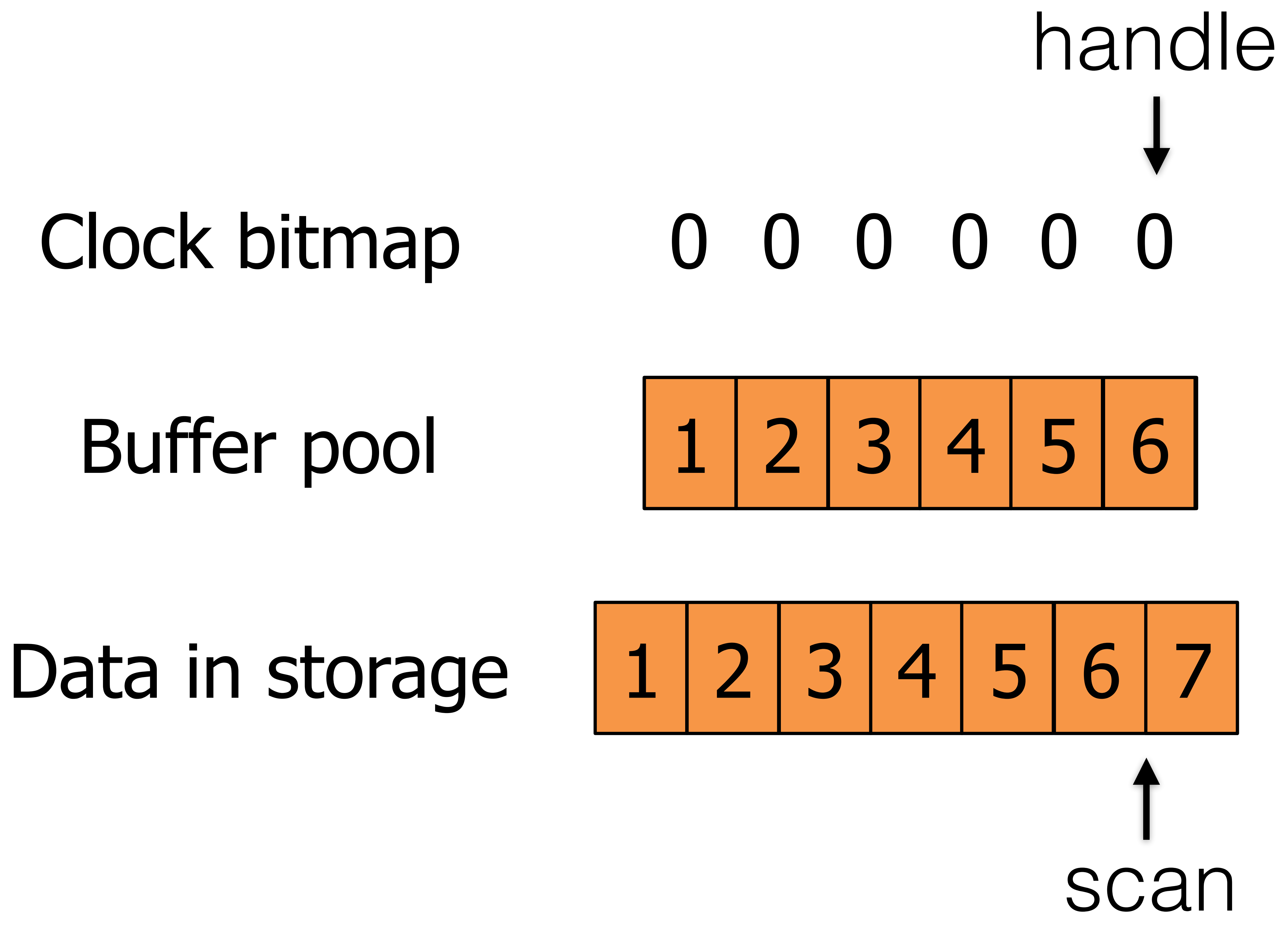
Suppose the DB is repeatedly scanning data larger than the buffer pool

Note the Simplification. Elements would really be randomly mapped in the buffer pool due to hashing.



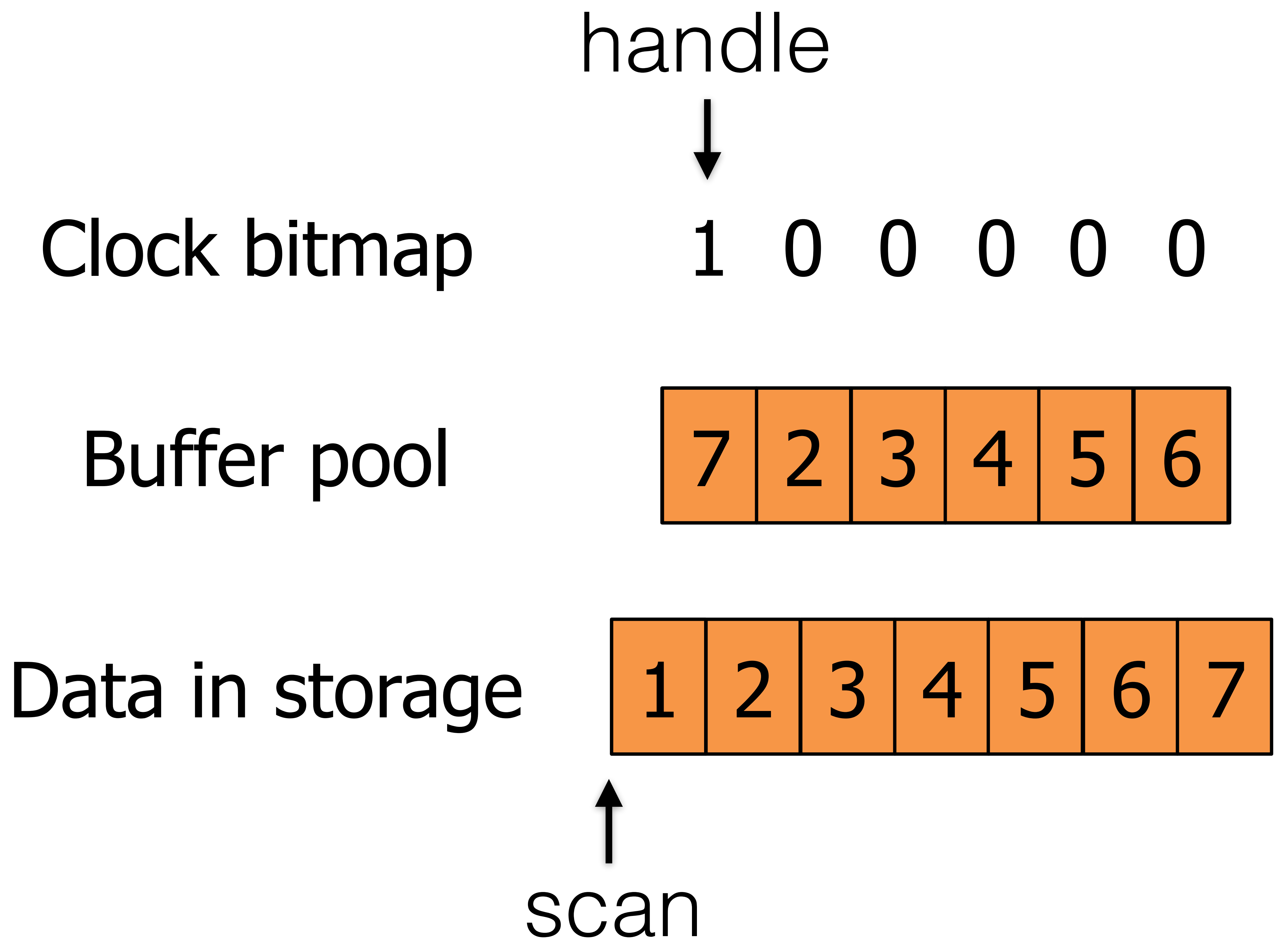
Question 5 - Sequential Flooding of Clock

Suppose the DB is repeatedly scanning data larger than the buffer pool



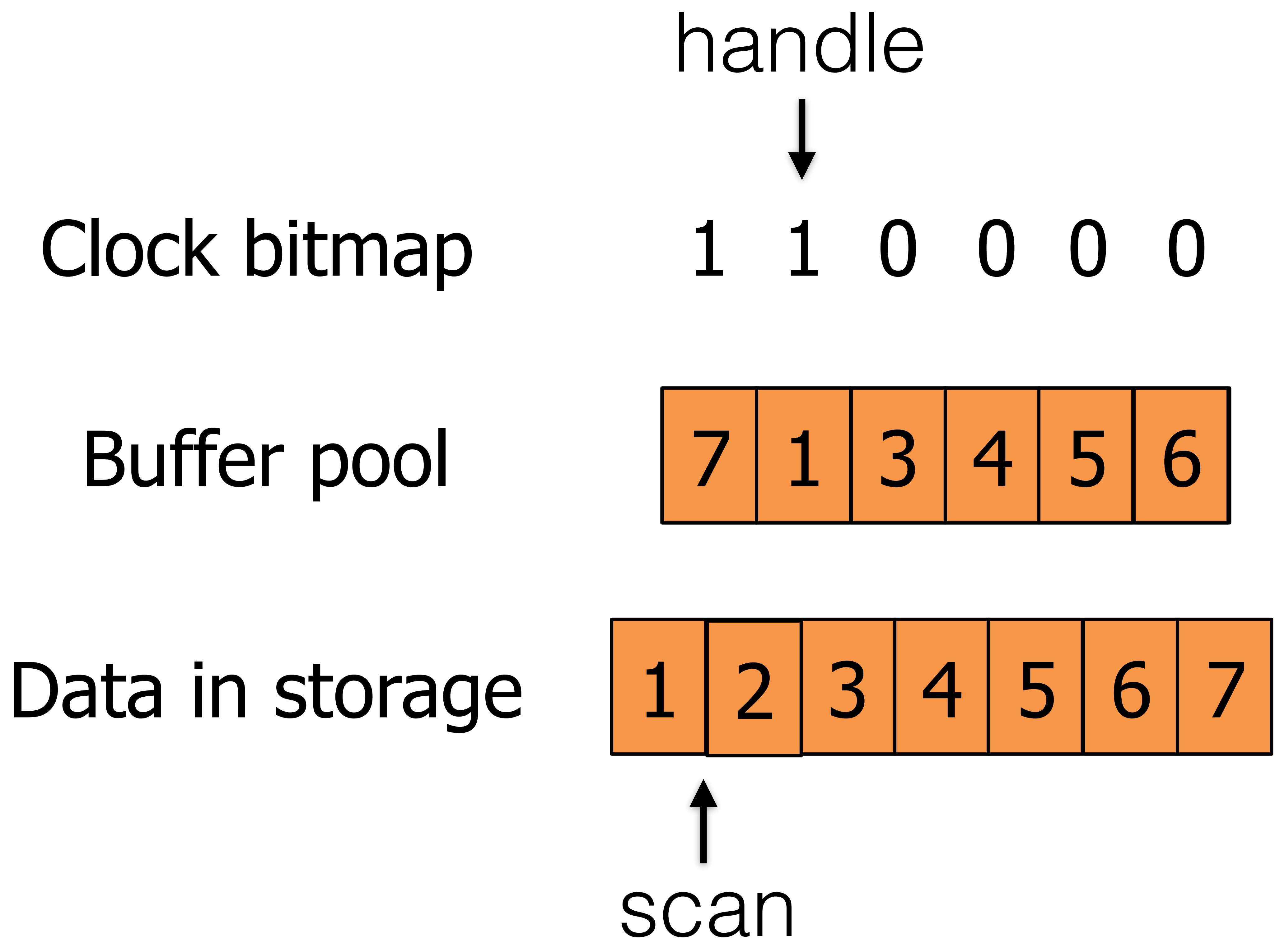
Question 5 - Sequential Flooding of Clock

Suppose the DB is repeatedly scanning data larger than the buffer pool



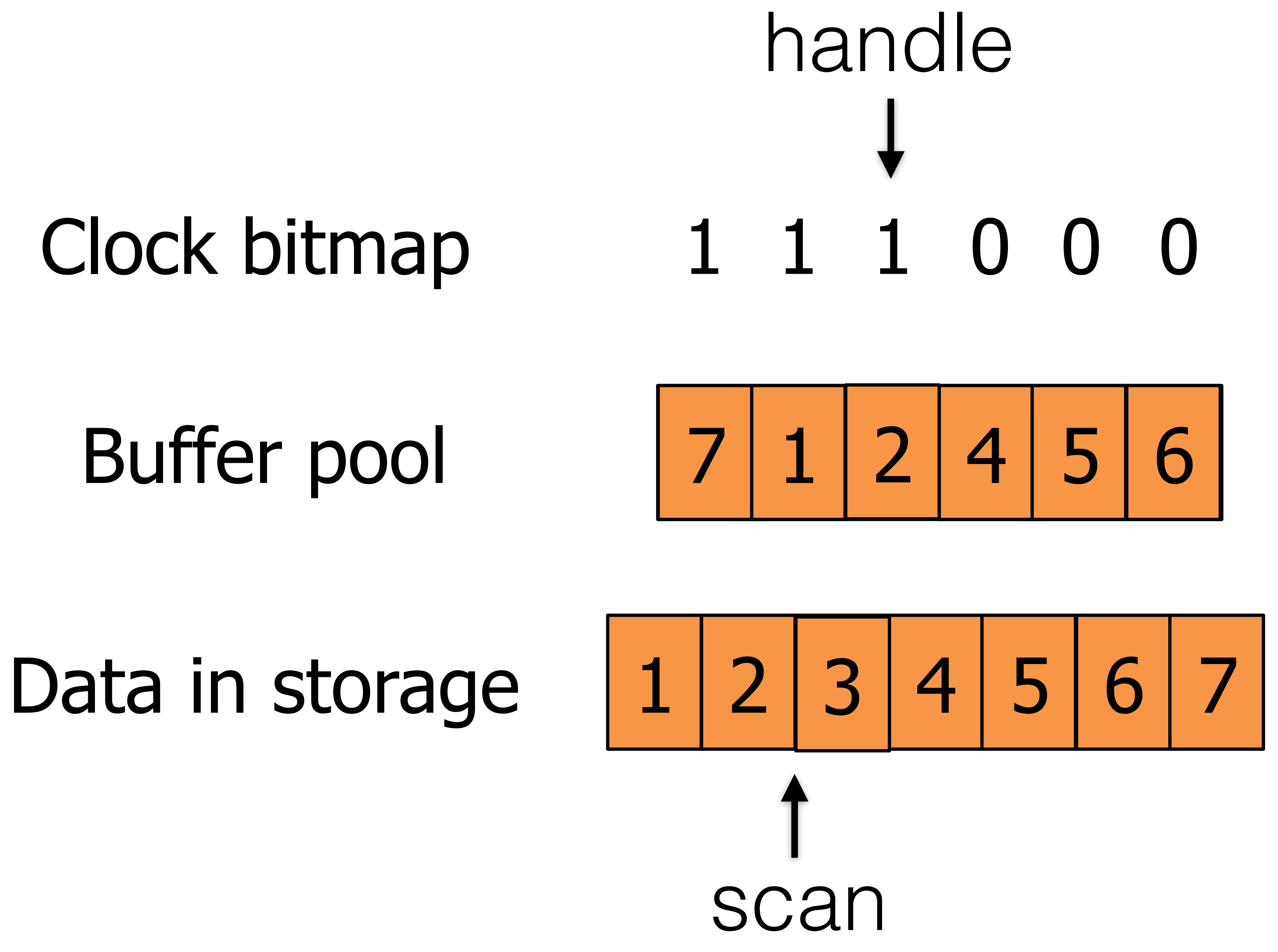
Question 5 - Sequential Flooding of Clock

Suppose the DB is repeatedly scanning data larger than the buffer pool



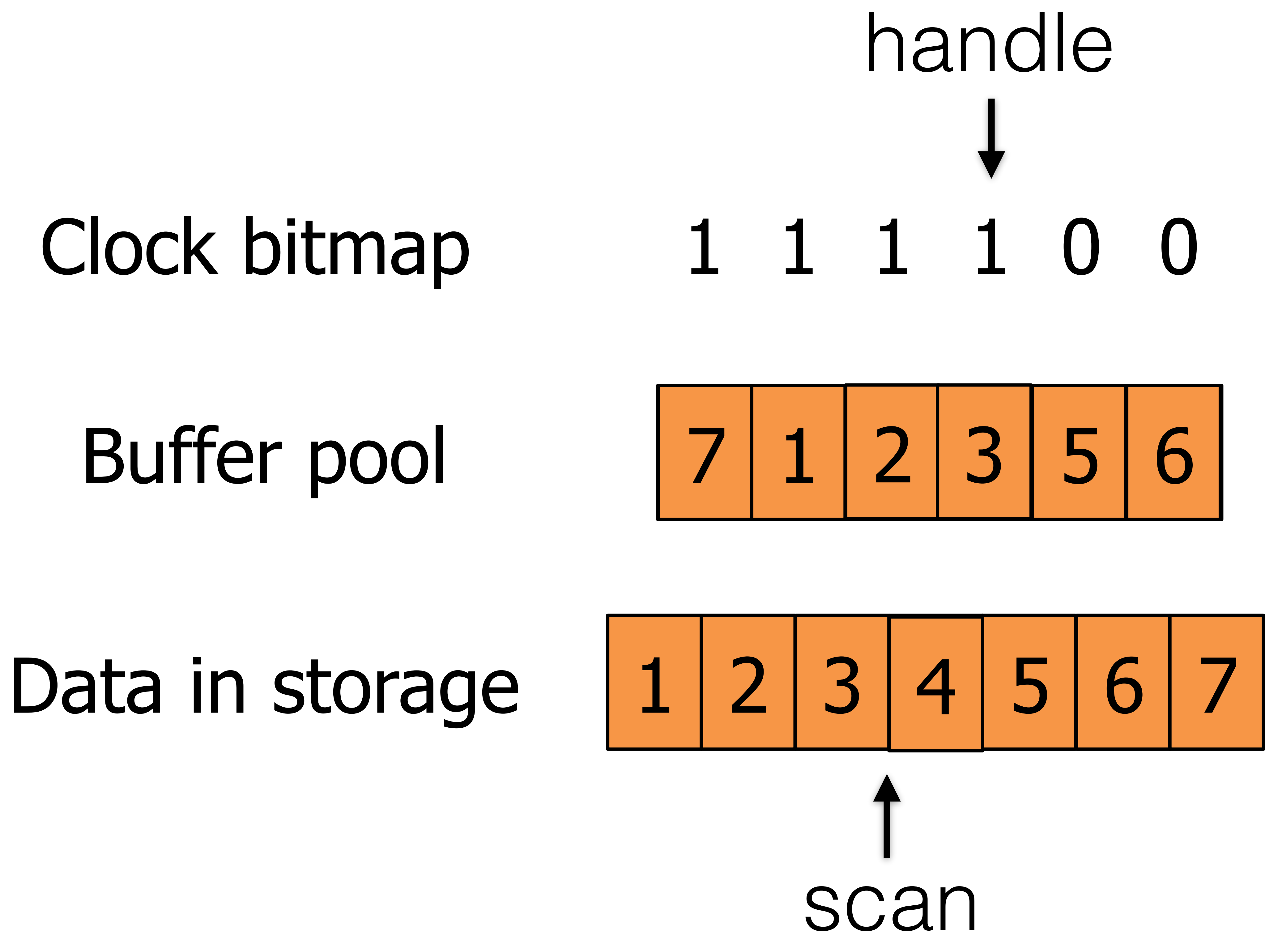
Question 5 - Sequential Flooding of Clock

Suppose the DB is repeatedly scanning data larger than the buffer pool



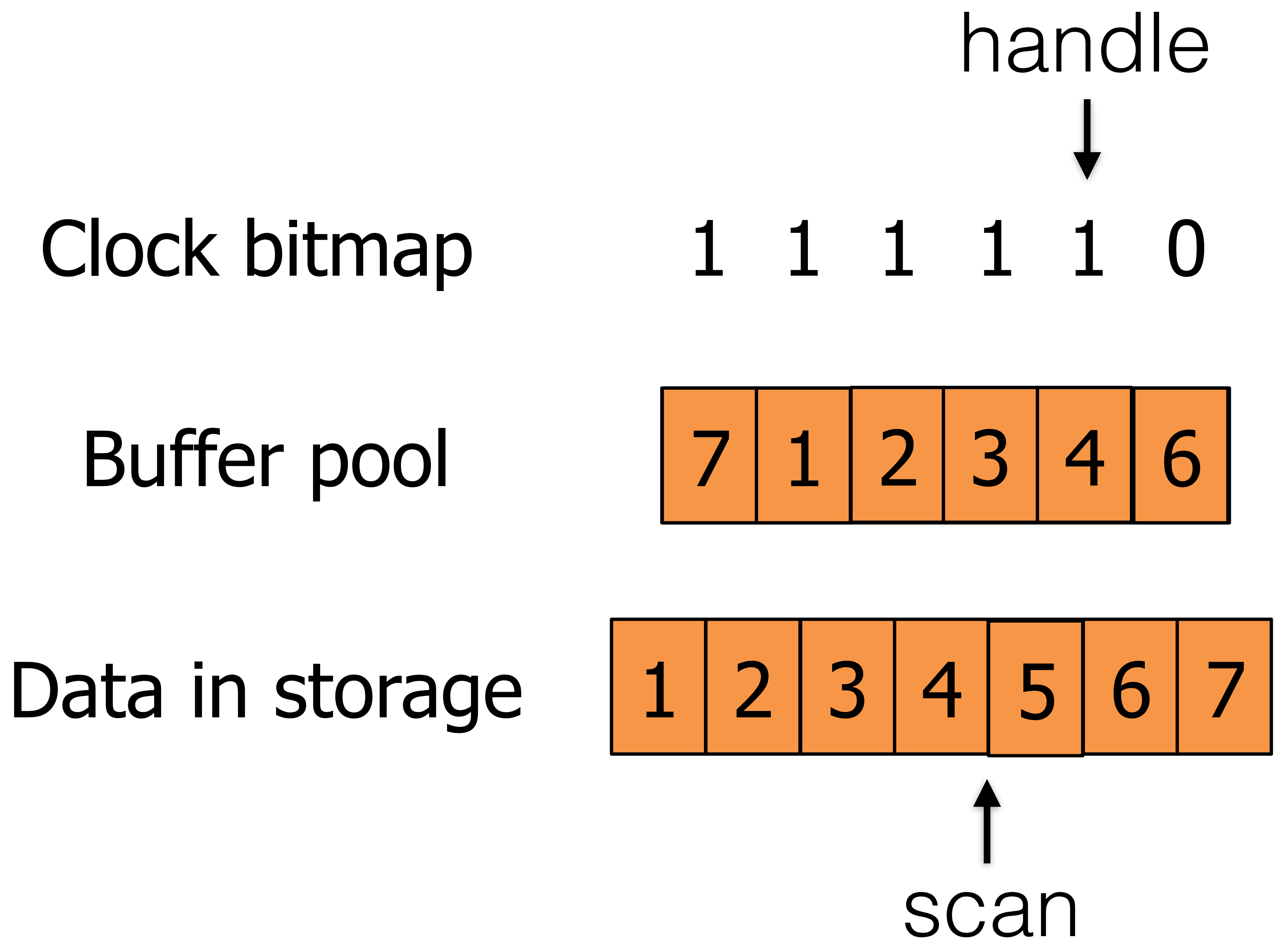
Question 5 - Sequential Flooding of Clock

Suppose the DB is repeatedly scanning data larger than the buffer pool



Question 5 - Sequential Flooding of Clock

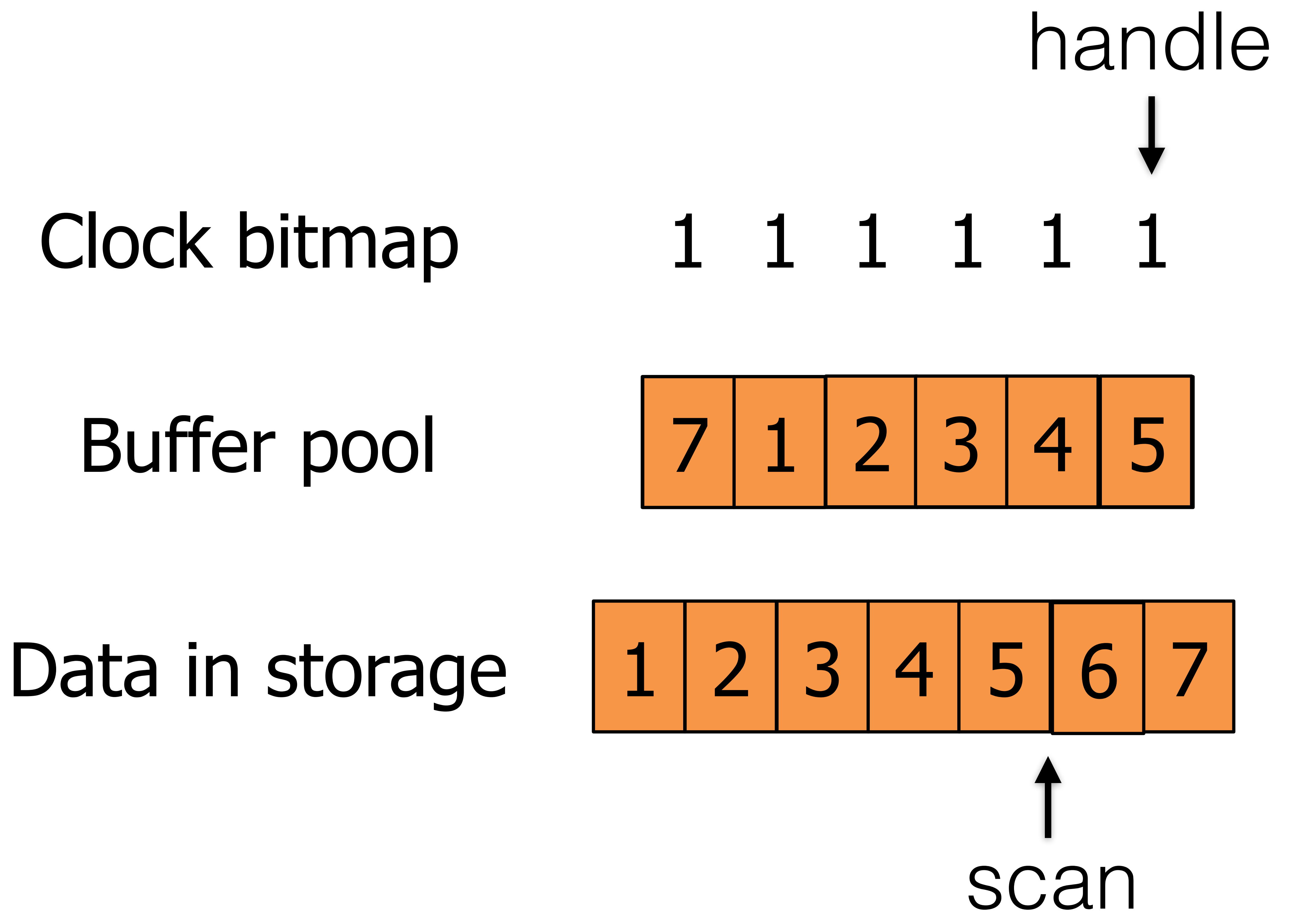
Suppose the DB is repeatedly scanning data larger than the buffer pool



Question 5 - Sequential Flooding of Clock

Suppose the DB is repeatedly scanning data larger than the buffer pool

Clock constantly evicts the page we need to read next!



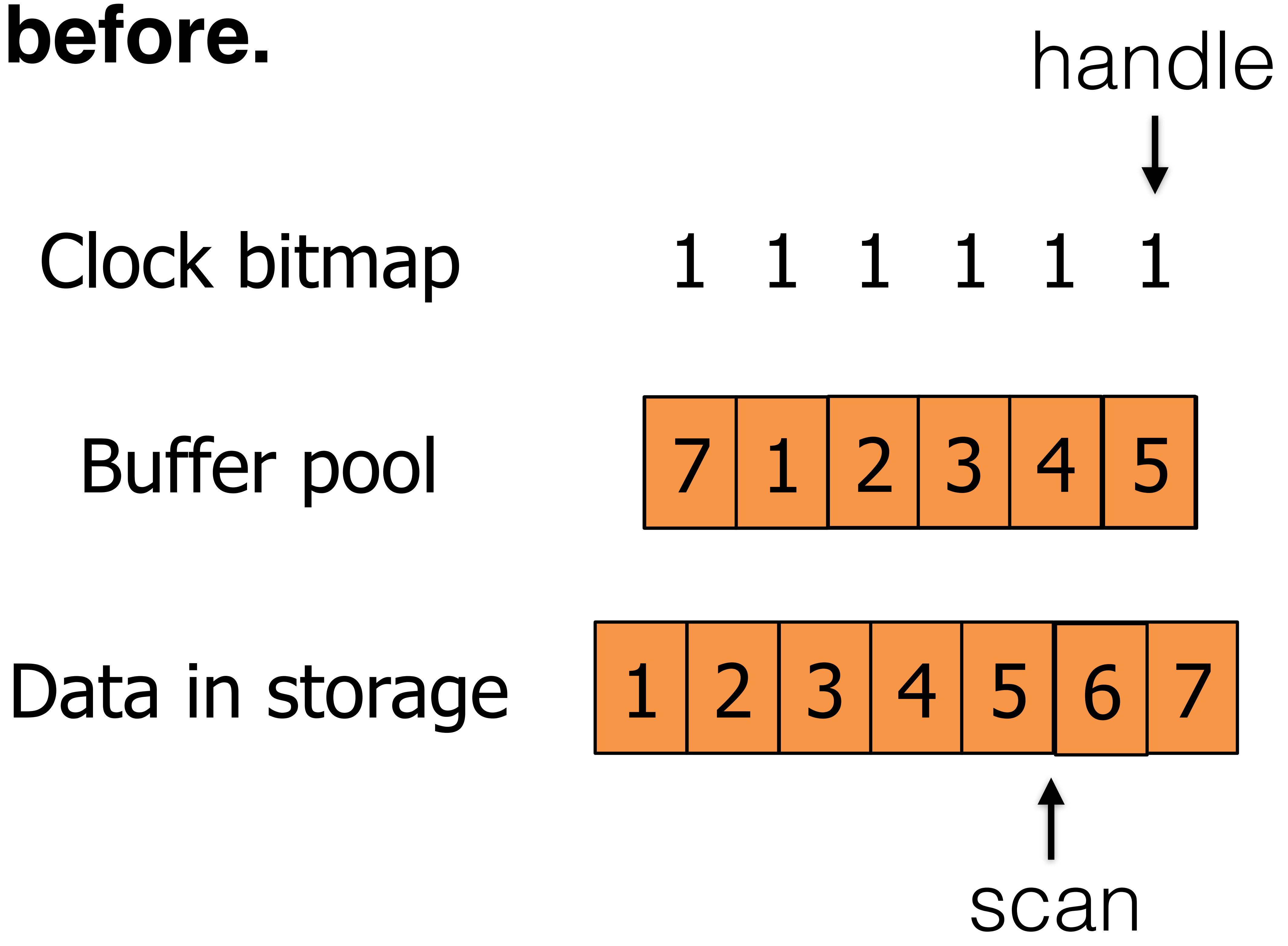
Question 5 - Sequential Flooding of Clock

Suppose the DB is repeatedly scanning data larger than the buffer pool

Clock constantly evicts the page we need to read next!

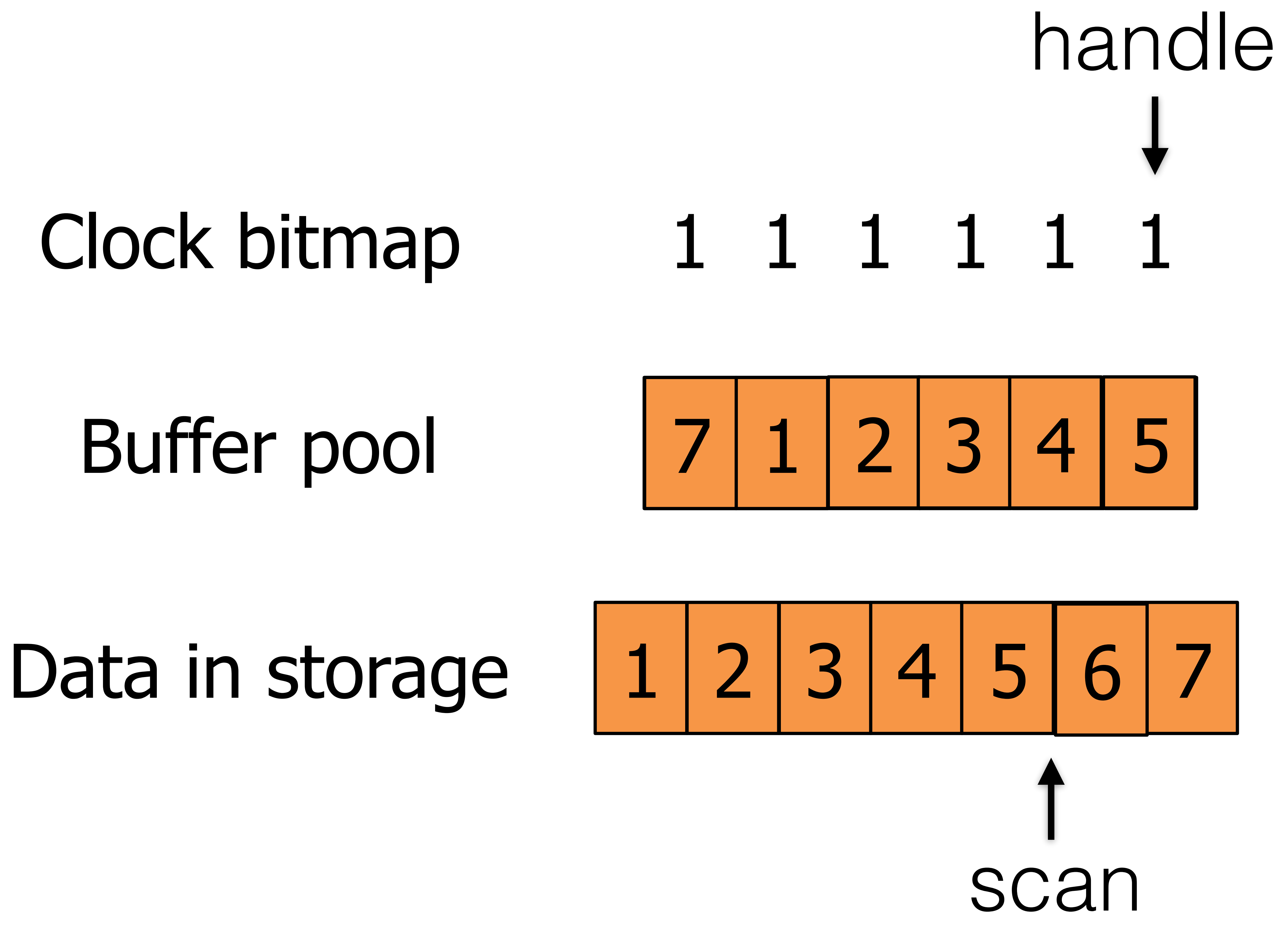
Also a problem for LRU from before.

Solution?



Question 5 - Sequential Flooding of Clock

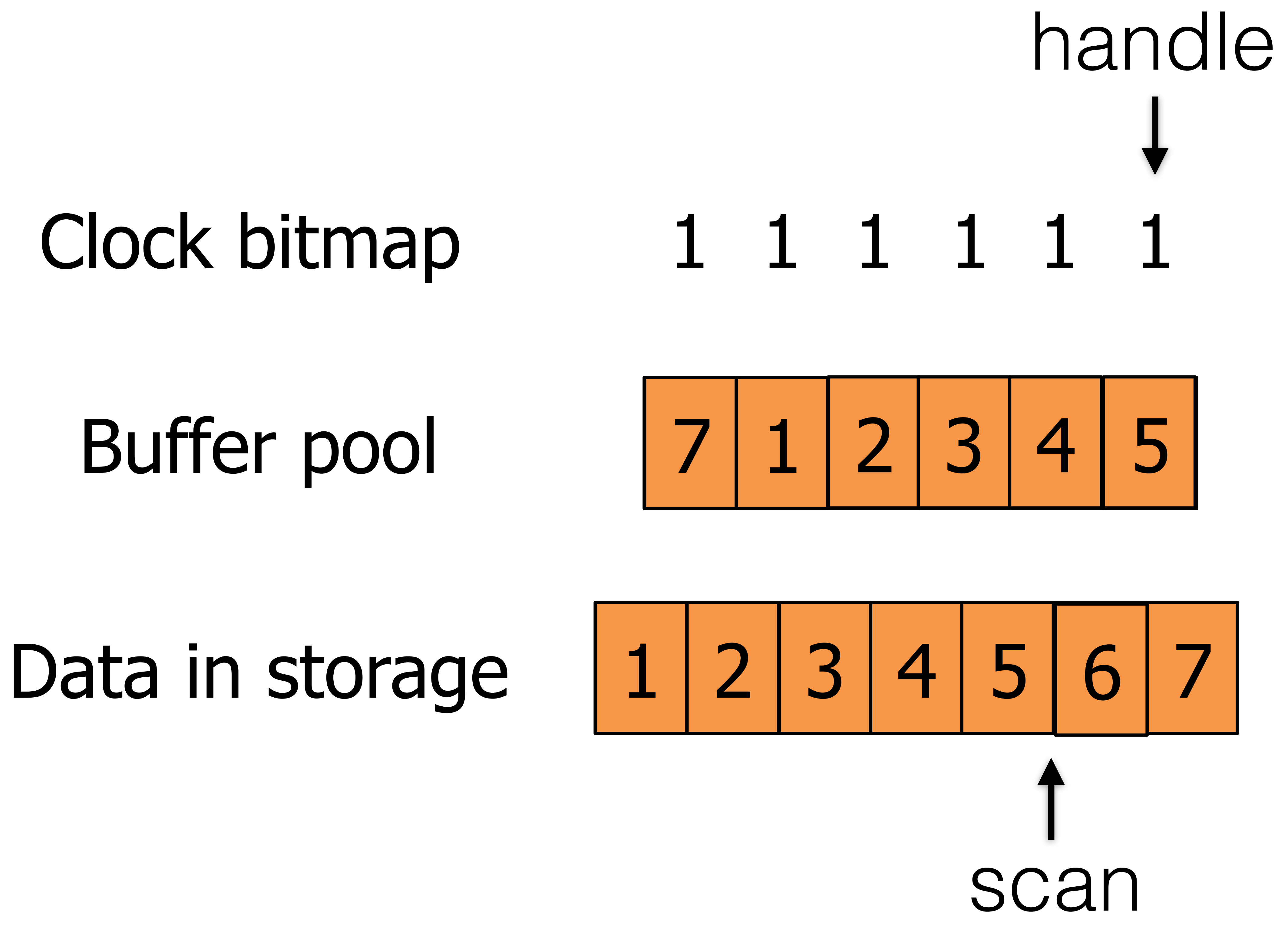
Using a most recently used (MRU) policy? :)



Question 5 - Sequential Flooding of Clock

Using a most recently used (MRU) policy? :) **No.**

We can avoid putting sequentially scanned data in the buffer pool



And now: office hours. Next week, we’re starting indexing.