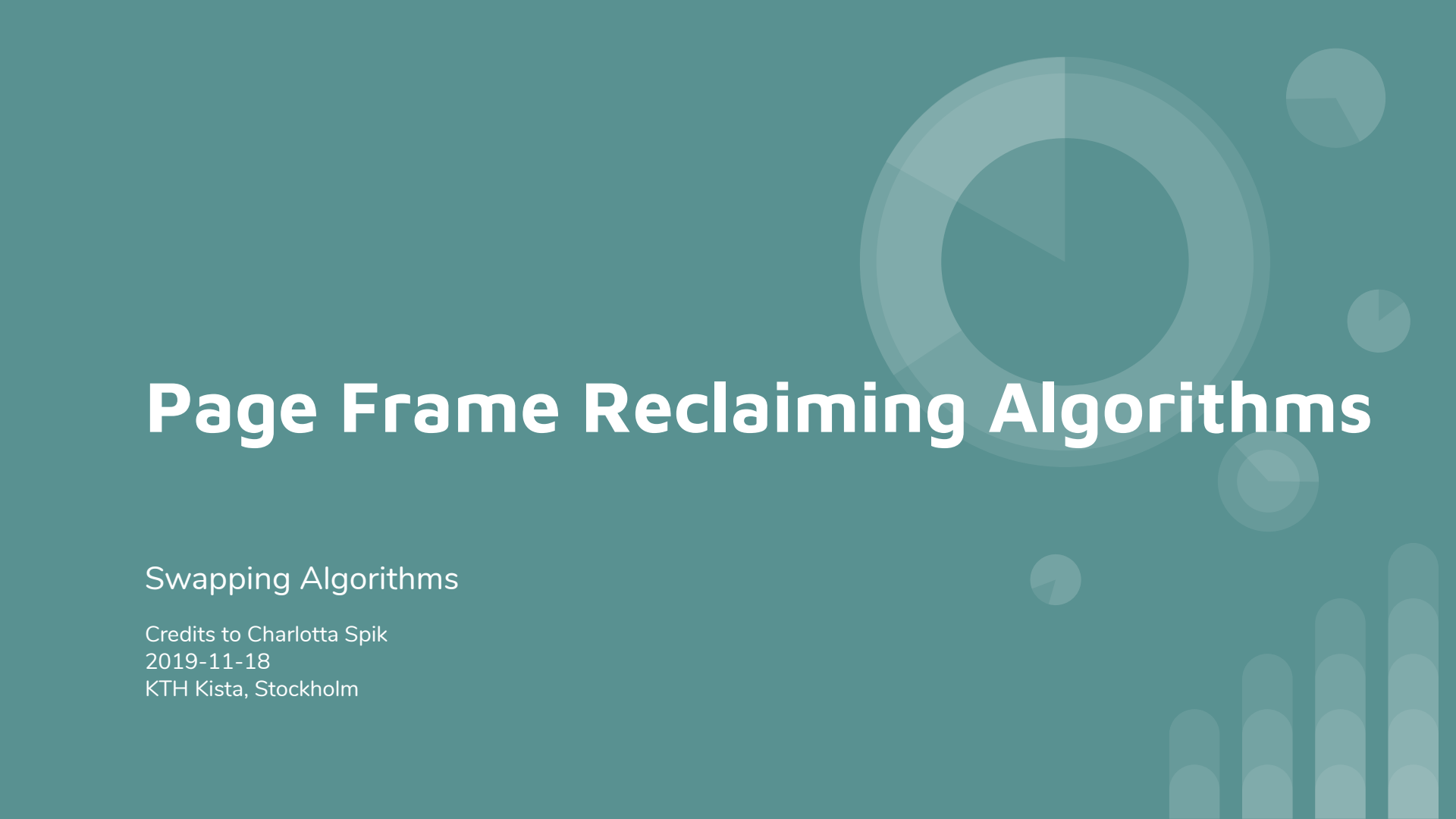


Page Frame Reclaiming Algorithms



Swapping Algorithms

Credits to Charlotta Spik
2019-11-18
KTH Kista, Stockholm





Swapping (1)

- What happens if we run out of physical memory (RAM)?
- We need to move some pages from physical memory to disk
 - Swap space – space on disk reserved for moving pages back and forth.
- This is called swapping pages between physical memory and disk
- Allows the OS to support the illusion of a large virtual memory for multiple concurrently-running processes
- If physical memory is full, a page in physical memory needs to be evicted when we access a page that is on disk



Swapping (2)

- Accessing a page that is on disk is slower than accessing a page from physical memory
 - Disk is slower than physical memory
 - The swapping process takes time
- Page hit
 - The page we want to access is already in physical memory, no need for swapping
- Page miss
 - The page we want to access resides on disk, we need to swap it for another page in physical memory
- We want as many page hits as possible as this requires less work



How to Choose Which Page to Evict?

- We do not want to evict a page that will soon be used again
 - Then we would have to place it in memory soon again
- We want to evict a page that will not be used for long
- There are different policies to decide what page to evict



Random Policy

- Evict a random page
- Pros:
 - Simple!
- Cons:
 - Luck based - can lead to us evicting a page that will be used soon again in the future => wasted time



Optimal Policy

- Replaces the page that will be accessed furthest in the future
- Leads to the fewest number of misses overall
- Problem: We do not know which pages will be accessed in the future!
 - Therefore, this policy cannot be used on practice
 - But we can simulate it and use the result to compare other policies' performance



FIFO Policy

- Pages are placed in a queue when they enter the system
- When replacement occurs, the page at the tail of the queue is evicted
 - This is the page that first entered the system, the “first-in” page
- Pros:
 - Easy to implement
- Cons:
 - Bad hit-rate as it cannot determine the importance of blocks
 - Bad performance with looping-sequential workload



Least Recently Used (LRU)

- Use history to predict the future
- If a program has accessed a page in the near past, it is likely to access it again in the near future
- LRU replaces the least recently used page
 - I.e. the page that hasn't been used for the longest time
- Pros:
 - Better hit-rate as it is likely to keep “hot pages” in memory
- Cons:
 - Complicated to implement
 - Computational performance heavy
 - Bad performance with looping-sequential workload



Clock Algorithm

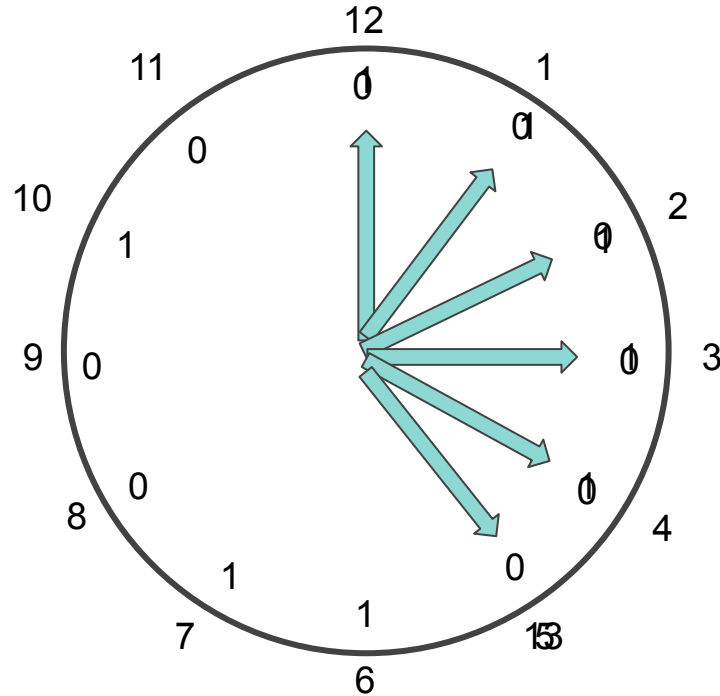
- Approximation of LRU
- Arranges the pages in a circular structure
- Uses one bit to determine if the page has been accessed since the last iteration of the circle
- Kick out the first page that has not been accessed since last iteration
- Performs almost as well as LRU, but with better computational performance and less complicated implementation



Clock Algorithm

We need to swap in
page 13!

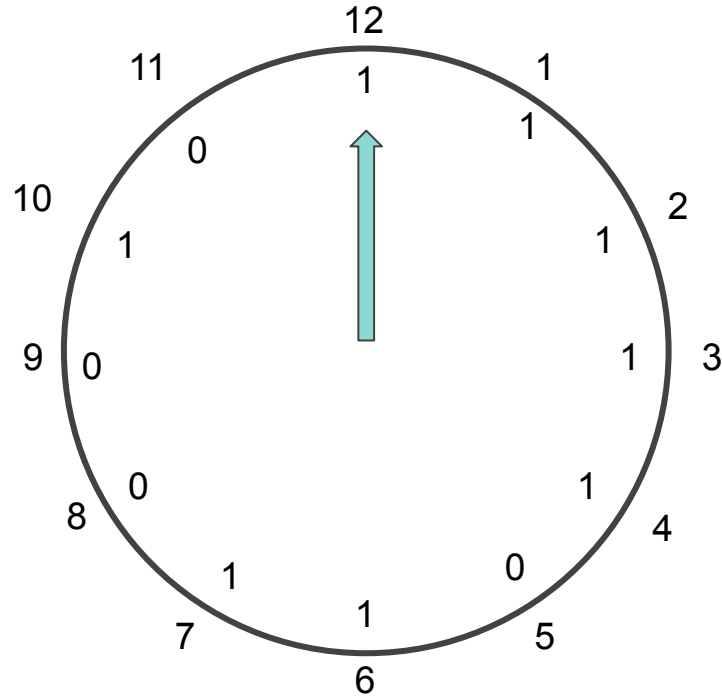
Found page to evict!





Clock Algorithm

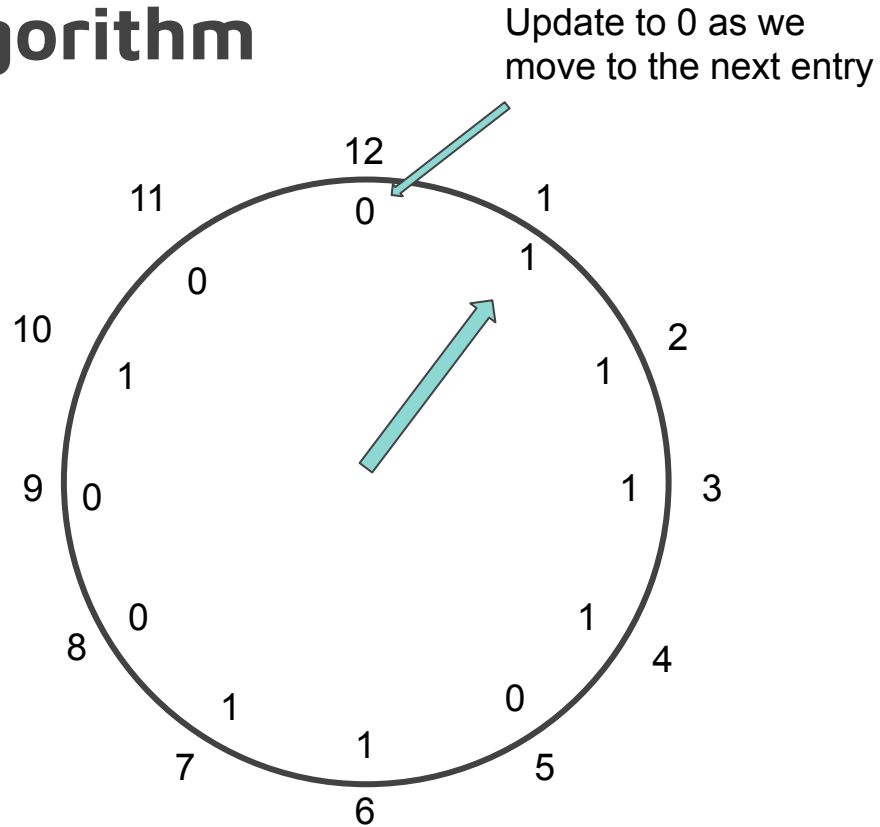
We need to swap in
page 13!





Clock Algorithm

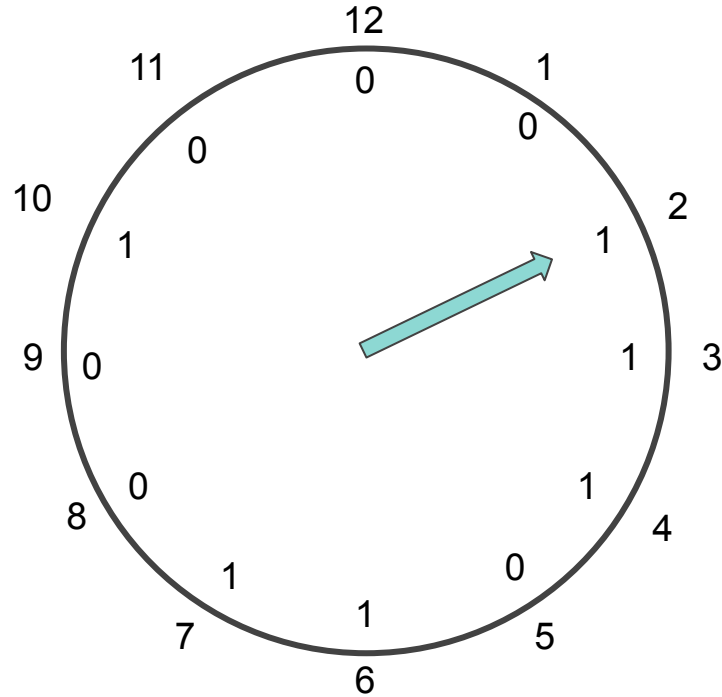
We need to swap in
page 13!





Clock Algorithm

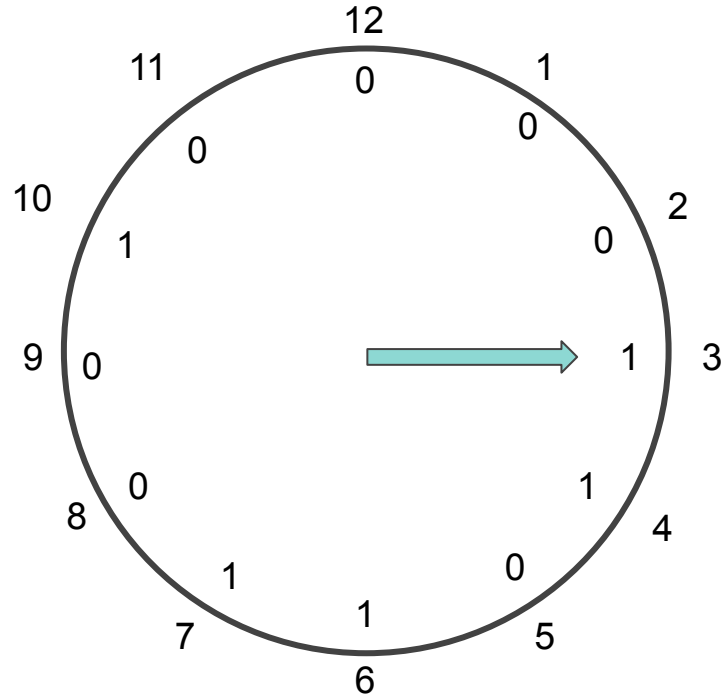
We need to swap in
page 13!





Clock Algorithm

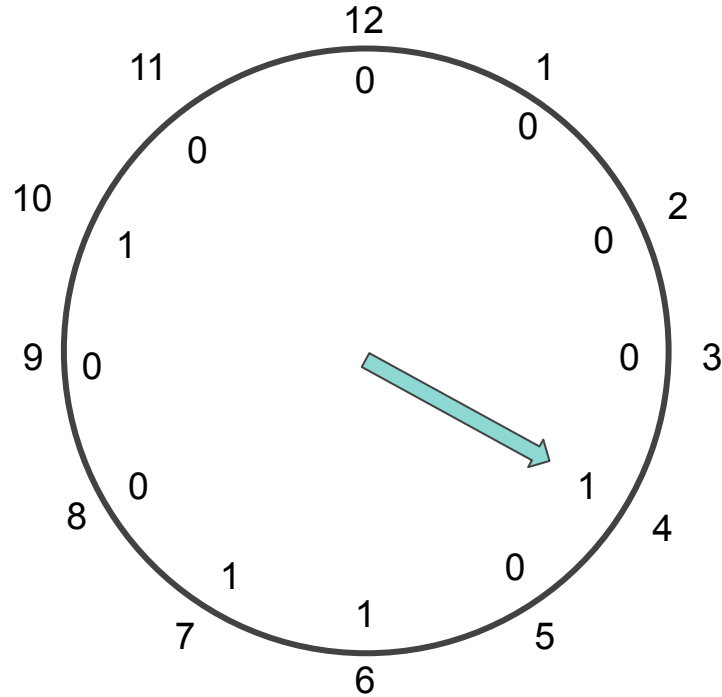
We need to swap in
page 13!





Clock Algorithm

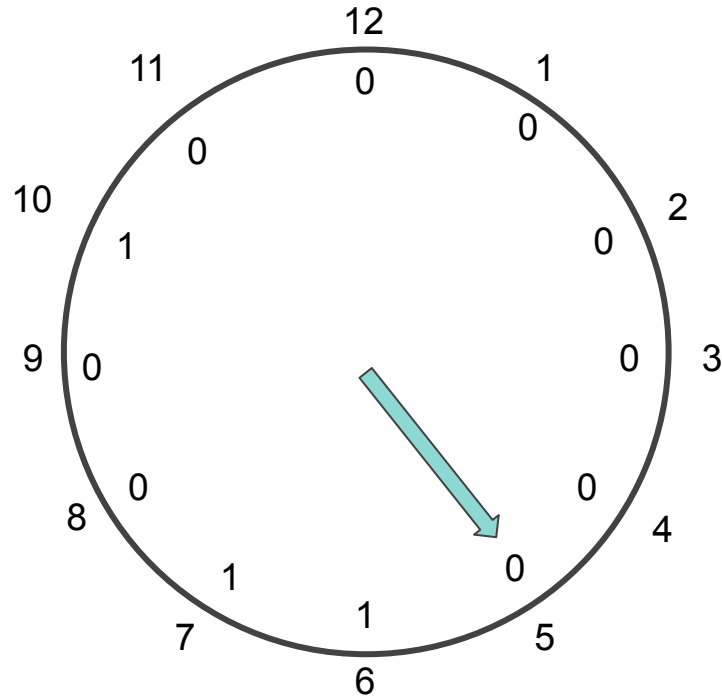
We need to swap in
page 13!





Clock Algorithm

We need to swap in
page 13!

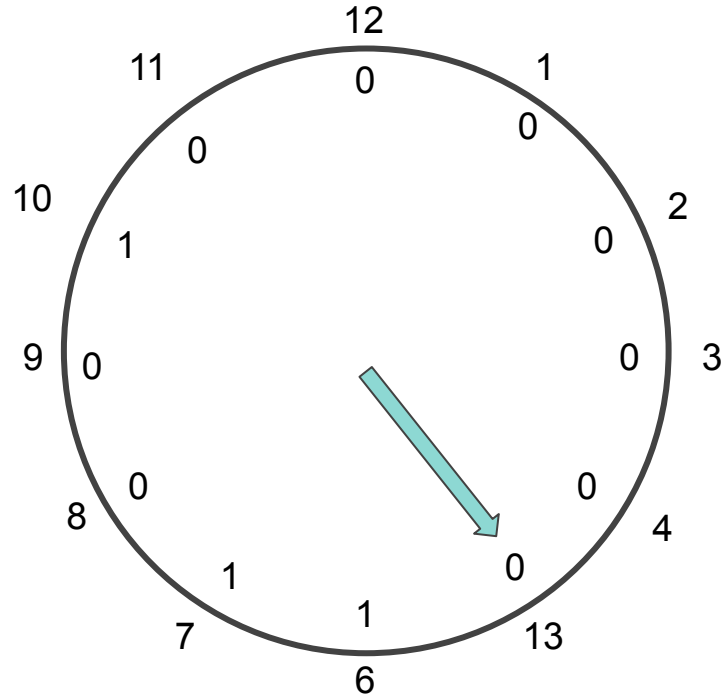




Clock Algorithm

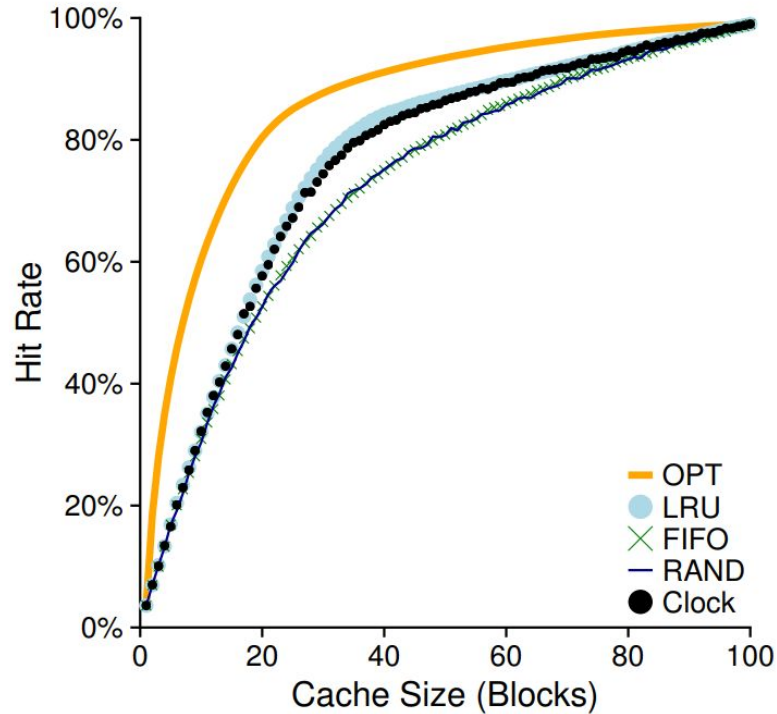
We need to swap in
page 13!

Found page to evict!





Performance Comparison



Exam Questions



Exam Question 1

Below is an extract from a program that implements Least Recently Used (**LRU**). The code shows why LRU is expensive to implement and why one probably instead chooses to approximate this strategy. What is the code doing and when is it executed?

```
if (entry->present == 1) {  
  
    if (entry->next != NULL) {  
  
        if (first == entry) {  
            first = entry->next;  
        } else {  
            entry->prev->next = entry->next;  
        }  
        entry->next->prev = entry->prev;  
  
        entry->prev = last;  
        entry->next = NULL;  
  
        last->next = entry;  
        last = entry;  
    }  
} else {
```

Below is an extract from a program that implements Least Recently Used (**LRU**). The code shows why LRU is expensive to implement and why one probably instead chooses to approximate this strategy. What is the code doing and when is it executed?

```
if (entry->present == 1) {  
    if (entry->next != NULL) {  
        if (first == entry) {  
            first = entry->next;  
        } else {  
            entry->prev->next = entry->next;  
        }  
        entry->next->prev = entry->prev;  
  
        entry->prev = last;  
        entry->next = NULL;  
  
        last->next = entry;  
        last = entry;  
    }  
} else {  
    :  
}
```

Answer: The code unlinks an entry and places it last in a list that should be updated with the least used pages first. This operation must be done every time a page is referenced.

Exam Question 2

When implementing the ***clock algorithm*** it is sufficient to have the list single linked i.e. there is no need for a double linked list. Why can we manage with a single linked list?

Exam Question 2

When implementing the ***clock algorithm*** it is sufficient to have the list single linked i.e. there is no need for a double linked list. Why can we manage with a single linked list?

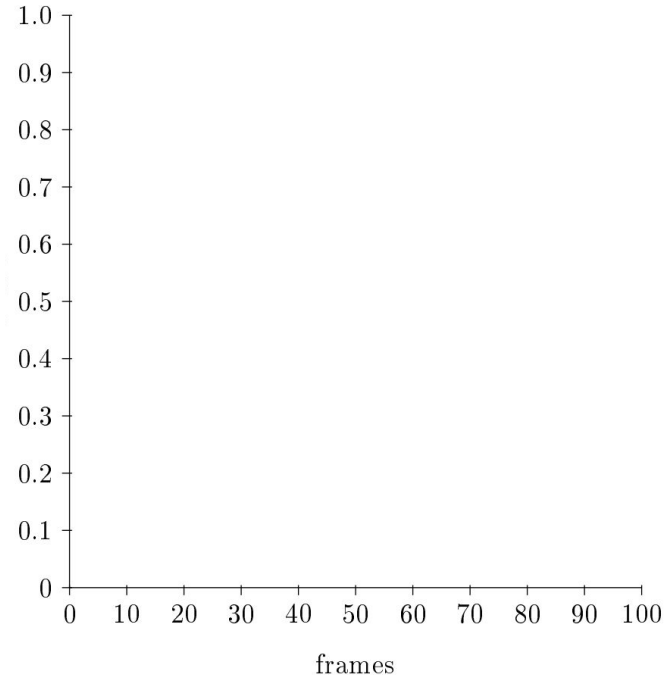
Answer: We never remove an element at random but always from the current position of the dial. We can easily keep track of its position and its immediate predecessor.

Exam Question 3

In an experiment we have a virtual memory of 100 pages and simulate a memory of up to 100 frames. The experiment simulate a sequence of memory operations with temporal locality

In the diagram below you should plot justiable graphs for the following three strategies:

- **RND**: *evict the page by random*
- **OPT**: *evict the page that won't be used for the longest time*
- **LRU**: *evict the page that was recently used*

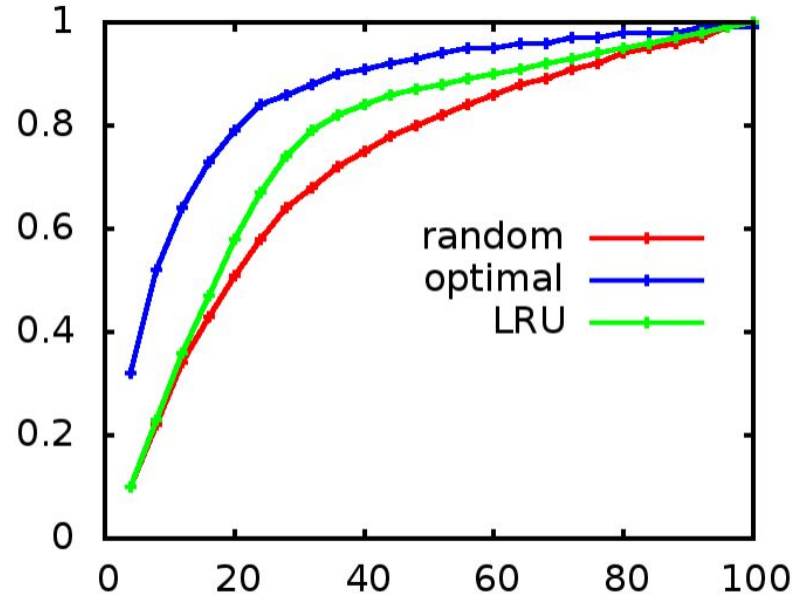


Exam Question 3

In an experiment we have a virtual memory of 100 pages and simulate a memory of up to 100 frames. The experiment simulate a sequence of memory operations with temporal locality

In the diagram below you should plot justiable graphs for the following three strategies:

- **RND**: *evict the page by random*
- **OPT**: *evict the page that won't be used for the longest time*
- **LRU**: *evict the page that was recently used*

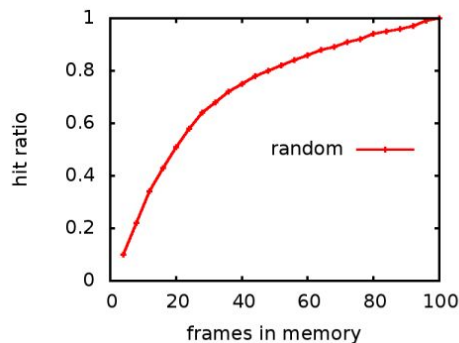


Exam Question 4

3.2 random not that wrong [2 points]

When benchmarking an implementation for swapping one found that a completely random algorithm worked better than expected. In the graph below you can see the ratio of page hits and page references when the size of the memory is changed. The process that runs uses 100 pages and the different runs shows how the hit ratio varies with the size of the memory.

One would expect 50% hit ratio when the memory is half of what the process needs, but the hit ratio is better than this. What could a plausible explanation to the observed behavior be?

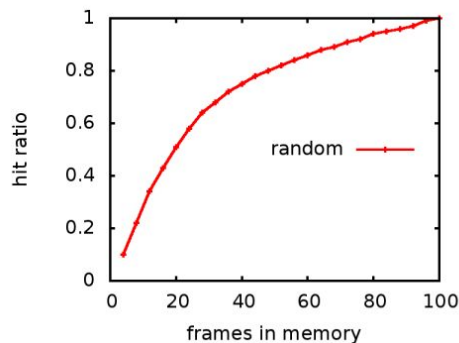


Exam Question 4

3.2 random not that wrong [2 points]

When benchmarking an implementation for swapping one found that a completely random algorithm worked better than expected. In the graph below you can see the ratio of page hits and page references when the size of the memory is changed. The process that runs uses 100 pages and the different runs shows how the hit ratio varies with the size of the memory.

One would expect 50% hit ratio when the memory is half of what the process needs, but the hit ratio is better than this. What could a plausible explanation to the observed behavior be?



Answer: A plausible explanation is that the process does not reference the pages randomly. It could be the case that some pages are referenced more often or that pages are often referenced repeatedly in a sequence (time locality).

Exam Question 5

4.1 approximate what? [2 points]

The clock algorithm, that is used to swap pages from memory, is described as an approximation. What is it that it tries to approximate? Describe a scenario when it does not do the right choice because of the approximation.

Exam Question 5

4.1 approximate what? [2 points]

The clock algorithm, that is used to swap pages from memory, is described as an approximation. What is it that it tries to approximate? Describe a scenario when it does not do the right choice because of the approximation.

Answer: It approximates LRU (*least recently used*). It only notes that a page has been used since the last turn, not how often or when it was accessed. If it is time to swap a page and all pages are marked as being used, the algorithm will clear the markers one by one and then swap the first page. It could be that this page was the page that was access most recently and thus should not be swapped.