# Page Replacement

## Computer Operating Systems, Spring 2024

**Instructor:** Travis McGaha

**Head TAs:** Nate Hoaglund & Seungmin Han

**TAs:**

| | | | |
|---|---|---|---|
| Adam Gorka | Haoyun Qin | Kyrie Dowling | Ryoma Harris |
| Andy Jiang | Jeff Yang | Oliver Hendrych | Shyam Mehta |
| Charis Gao | Jerry Wang | Maxi Liu | Tom Holland |
| Daniel Da | Jinghao Zhang | Rohan Verma | Tina Kokoshvili |
| Emily Shen | Julius Snipes | Ryan Boyle | Zhiyan Lu |

# Poll Everywhere

pollev.com/tqm

- How is PennOS going?

# Administrivia

- PennOS
  - You have the first milestone, which should have been done last week
  - Everyone should have already contacted their group, and should get started working on it.
  - Milestone 1 is due this week
    - Between Tuesday the 9th and Friday the 12th
    - Need to meet with TA again to show significant progress
    - Have a plan (a REAL plan) for how to complete the rest
    - Autograder for pennfat is relased
      - You do not need to pass it for the milestone, but you should be able to showcase work you have done on it.
  - Full Thing due ~April 22nd

# Administrivia

- Check-in released: due at end of Friday
  - Another one will be released this week, due sometime next week

# Lecture Outline

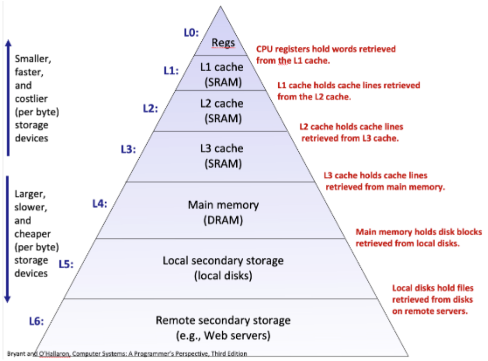
- **Page Replacement: High Level**
  - **FIFO**
  - **Reference Strings**
  - **Beladys**
- LRU
- Thrashing
- FIFO w/ Reference bit

# Page Replacement

- The operating system will sometimes have to evict a page from physical memory to make room for another page.

- If the evicted page is access again in the future, it will cause a page fault, and the Operating System will have to go to Disk to load the page into memory again

- Remember this? Disk access is very very slow (relatively speaking).
  - How can we minimize disk accesses?
  - How can we try to ensure the page we evict from memory is unlikely to be used again in the future?



# Reference String

- A reference string is a string representing a sequence of virtual page accesses. By a given process on some input.
  - E.g., `0 1 2 3 4 1 2 9 5 3 2 2 …`
  - Page 0 is accessed, then 1, then 2, then 3 …

- These strings are useful for reasoning about page replacement policies, and how they act on certain page access patterns

# FIFO Replacement

- One way to decide which pages can be evicted is to use FIFO (First in First Out)

- If a page needs to be evicted from physical memory, then the page that has been in memory the longest (since it was last brought into memory) can be evicted.

# FIFO Replacement

- If we have 4 frames, and the reference string:
  4 1 1 2 3 4 **5**
  - Red numbers indicate that accessing the page caused a page fault. Accessing 5 also causes 4 to be evicted from physical memory

| | Ref str: | 4 | 1 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|
| **Newest** | | 4 | 1 | 1 | 2 | 3 | 3 | 5 |
| | | | 4 | 4 | 1 | 2 | 2 | 3 |
| | | | | | 4 | 1 | 1 | 2 |
| **Oldest** | | | | | | 4 | 4 | 1 |

# Poll Everywhere

pollev.com/tqm

- Given the following reference string, how many page faults occur when using a FIFO algorithm
- 1 2 3 4 1 2 5 1 2 3 4 5

- Assume that
  - physical memory has three frames
  - we can ignore sharing those frames with other processes.
  - Physical Memory starts empty

- Part 2: If we didn't have to follow a strict policy, what is the "optimal" pages that could be evicted to minimize faults? How many less faults would we have?

# "optimal" replacement

- If you knew the exact sequence of page accesses in advance, you could optimize for smallest number of page faults

- Always replace the page that is furthest away from being used again in the future
  - How do we predict the future??????
  - You can't, but you can make a "best guess" (later in lecture)

- Optimal replacement is still a handy metric. Used for testing replacement algorithms, see how an algorithm compares to various "optimal" possibilities.

# Poll Everywhere

**pollev.com/tqm**

- Given the following reference string, how many page faults occur when using a FIFO algorithm
- 3 2 1 0 3 2 4 3 2 1 0 4

- Assume that
  - physical memory has **three** frames
  - we can ignore sharing those frames with other processes.
  - Physical Memory starts empty

- Part 2: What if we had **4** page frames, how many faults would we have?

# Bélády's anomaly

- Sometimes increasing the number of page frames results in an increase in the number of page faults ☹

- This behaviour is something that we want to avoid/minimize the possibility of.

- Stack based algorithms (Optimal, LIFO, LRU) avoid this issue

# Lecture Outline

- Page Replacement: High Level
  - FIFO
  - Reference Strings
  - Beladys
- **LRU**
- Thrashing
- FIFO w/ Reference bit

# LRU (Least Recently Used)

- If a page is used recently, it is likely to be used again in the near future
- Use past knowledge to predict the future
- Replace the page that has had the longest time since it was last used

| | **Ref str:** | **4** | **0** | **1** | **2** | **0** | **3** | **0** | **4** | **2** | **3** | **0** | **3** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Most recently used** | | 4 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 |
| | | | 4 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 |
| **LRU** | | | | 4 | 0 | 1 | 2 | 2 | 3 | 0 | 4 | 2 | 2 |
| **Victim** | | | | | 4 | | 1 | | 2 | 3 | 0 | 4 | |

# Poll Everywhere

pollev.com/tqm

- What if there are four frames instead of 3? How Many Page Faults?

| | Ref str: | 4 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Most recently used | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| LRU | | | | | | | | | | | | | |
| Victim | | | | | | | | | | | | | |

# LRU Implementation?

- To implement this properly, there are a couple possibilities
  - we would need to timestamp each memory access and keep a sorted list of these pages
    - High overhead, timestamps can be tricky to manage :/
  - Keep a counter that is incremented for each memory access Look through the table to find the lowest counter value on eviction
    - Looking through the table can be slow
    - How do you distinguish a process that has been accessed a lot in the past vs one accessed a little more recently?
  - Whenever a page is accessed find it in the stack of active pages and move it to the bottom

# LRU Approximation: Reference Bit & Clock

- It is expensive to do bookkeeping every time a page is accessed. Minimize the bookkeeping if possible

- When we access a page, we can update the reference bit for that PTE to show that it was accessed recently
  - This is done automatically by hardware, when accessing memory.
  - Setting a bit to 1 is much quicker than managing time stamps and re-organizing a stack

- We could check the reference bit at some clock interval to see if the page was used at all in the last interval period

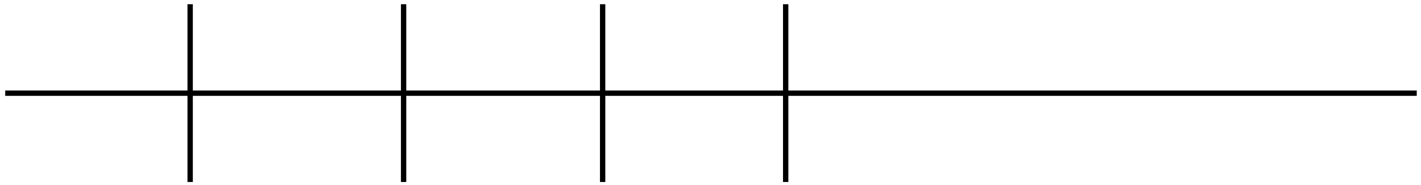
# LRU Approximation: Aging

- Each page gets an 8-bit counter.
- On clock interval and for every page:
  - shift the counter to the right by 1 bit
  - copy the reference bit into the MSB of the counter.
  - Reference bit in the PTE is reset to 0

- If we read the counter as an unsigned integer, then a larger value means the counter was accessed more recently

# Aging Illustration

- Timeline



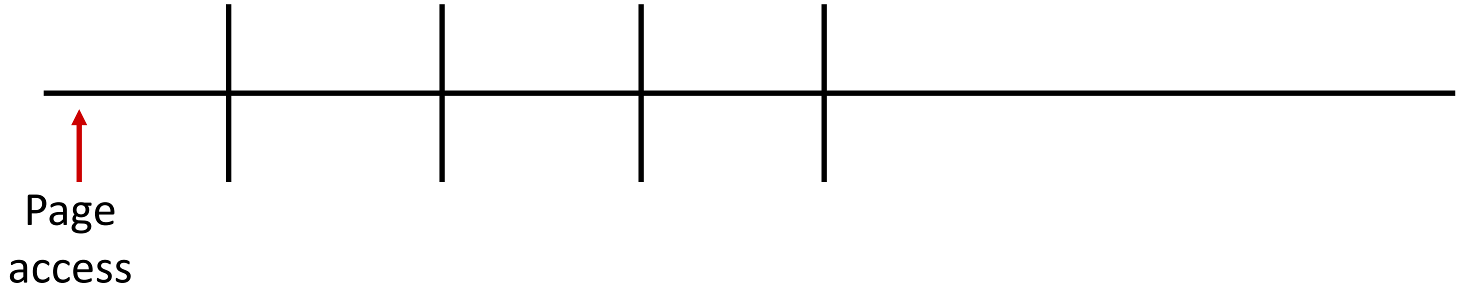
- Counter:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- Ref bit: 0

# Aging Illustration

- Timeline



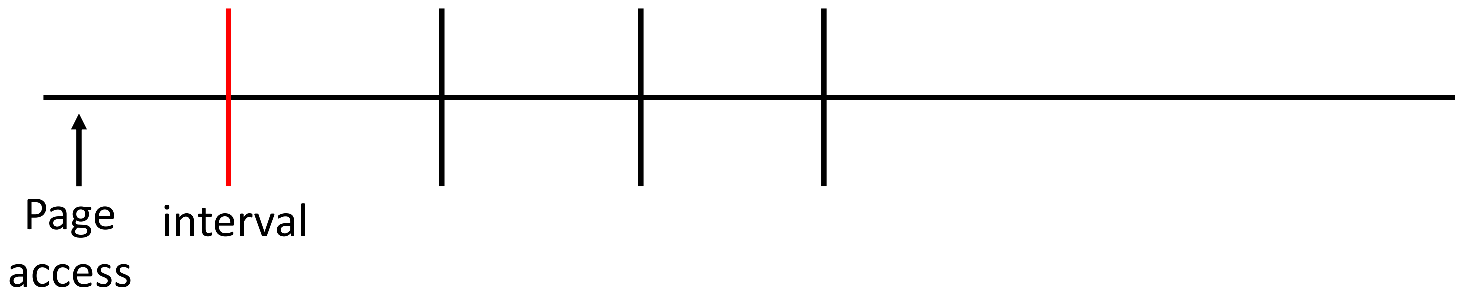

- Counter:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- Ref bit: 1

# Aging Illustration

- Timeline



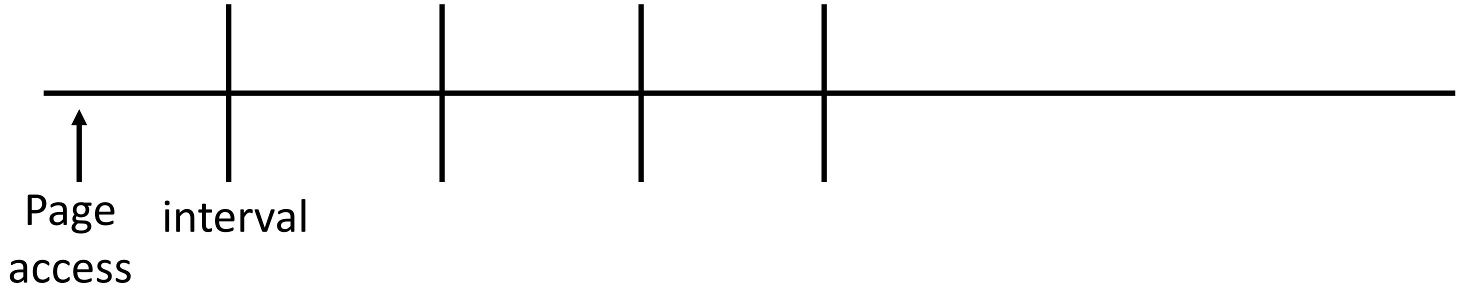

- Counter:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- Ref bit: 0

# Aging Illustration

- Timeline



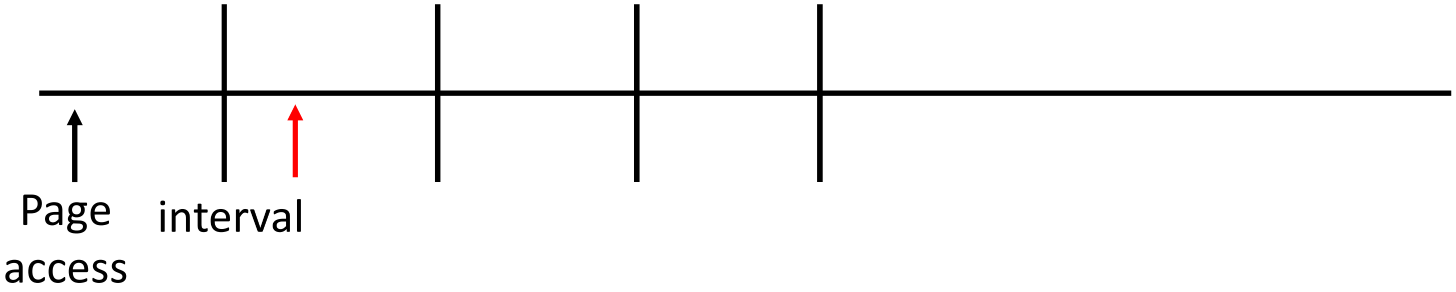

- Counter:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- Ref bit: 0

# Aging Illustration

- Timeline



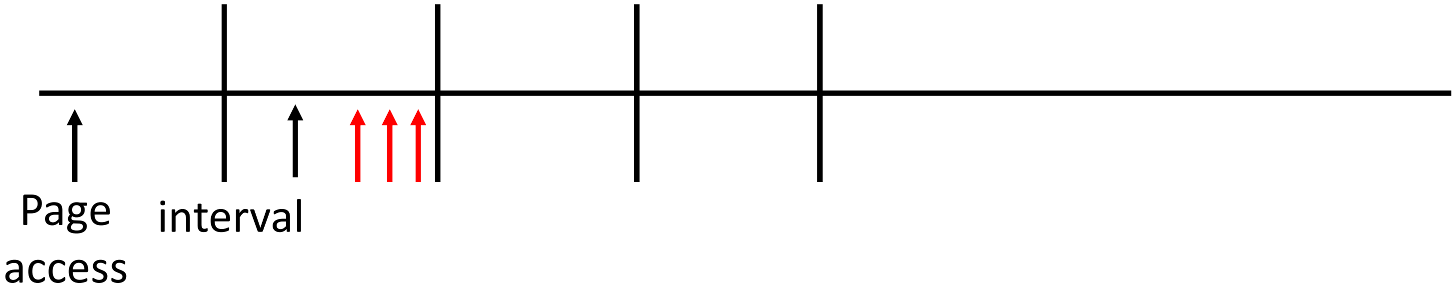

- Counter:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- Ref bit: 1

# Aging Illustration

- Timeline



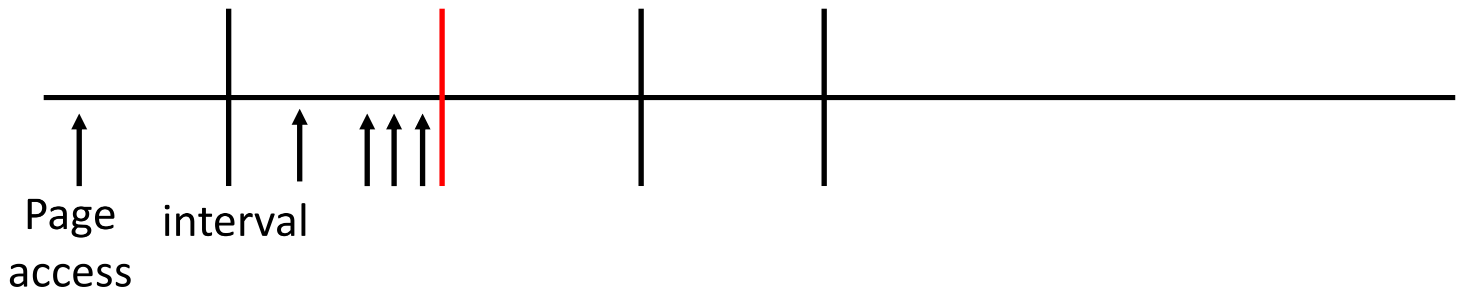

- Counter:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- Ref bit: 1

# Aging Illustration

- Timeline




- Counter:

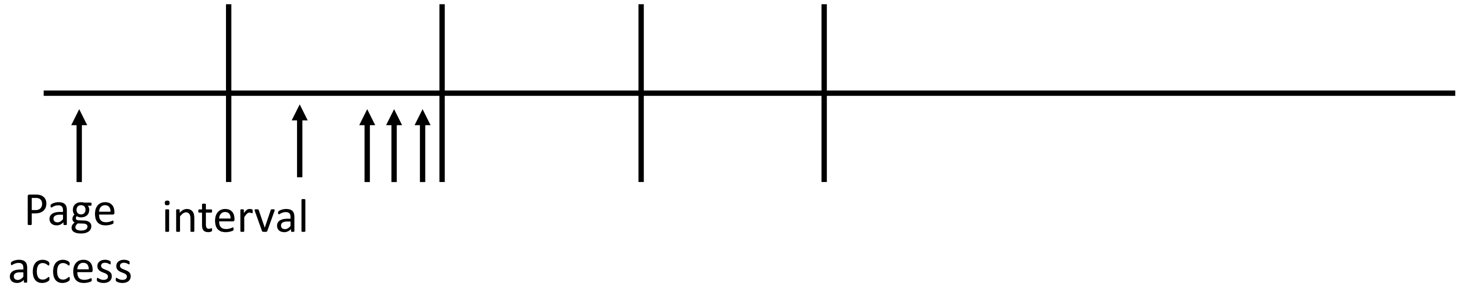
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

- Ref bit: 0

Same change to counter regardless of number of accesses in the interval, and when the accesses happened in the interval

# Aging Illustration

- Timeline



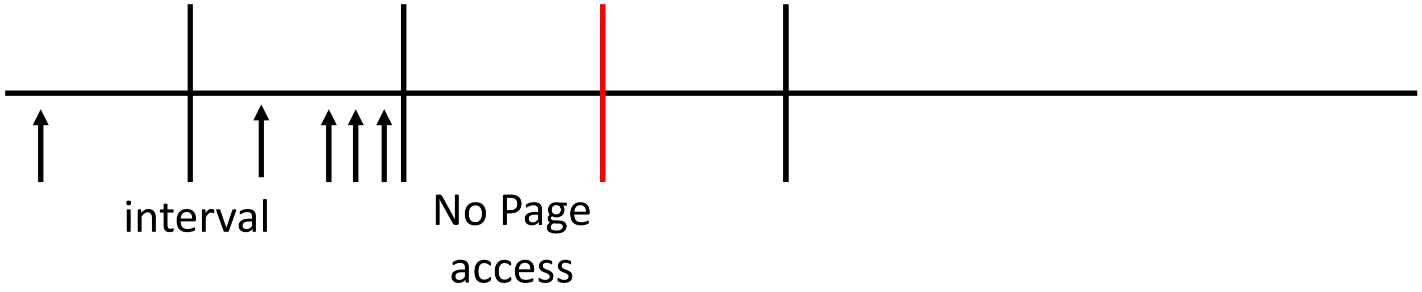

- Counter:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

- Ref bit: 0

# Aging Illustration

- Timeline




- Counter:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

- Ref bit: 0

# Aging: Analysis

- Analysis
  - Low overhead on clock tick and memory access
  - Still must search page table for entry to remove
  - Insufficient information to handle some ties
    - Only one bit information per clock cycle
    - Information past a certain clock cycle is lost

# Lecture Outline

- Page Replacement: High Level
  - FIFO
  - Reference Strings
  - Beladys
- LRU
- **Thrashing**
- FIFO w/ Reference bit

# Thrashing

- This is not specific to LRU, but it is easiest to demonstrate with LRU

- When the physical memory of a computer is overcommitted, causing almost constant page faults (which are slow)
  - Overcommitment most commonly happens when there are too many processes, and thus too much memory needed
  - Can also happen with a few processes, if the process needs too much memory

# Thrashing: LRU Example

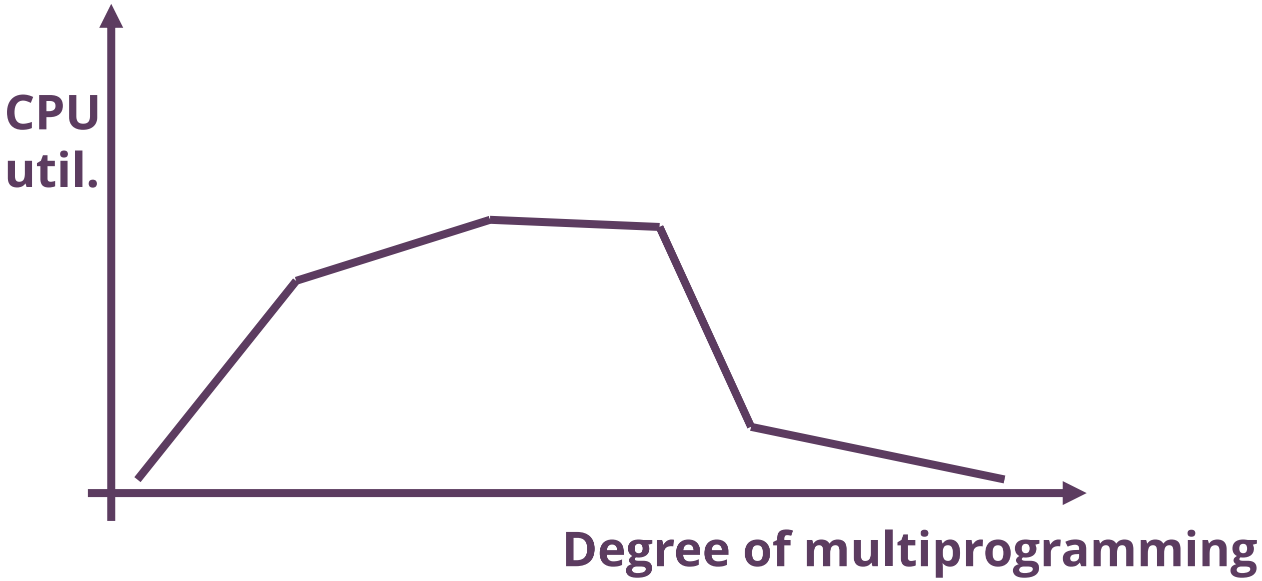
- Consider the following example with three page frames and LRU

| | Ref str: | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Most recently used | | 0 | 1 | 1 | 2 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| | | | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 |
| LRU | | | | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 |
| Victim | | | | | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 |

- Page fault on every memory access ☹

# Thrashing: Multiprogramming

- It is good to have more processes running, then we can have better utilization of CPU.
  - While one process waits on something, another can run
  - More on CPU Utilization later
- As we use more processes running at once, more memory is needed, can cause thrashing ☹



# Lecture Outline

- Page Replacement: High Level
  - FIFO
  - Reference Strings
  - Beladys
- LRU
- Thrashing
- **FIFO w/ Reference bit**

# FIFO Analysis

- Remember FIFO? The first page replacement algorithm we covered?
  - Evict the page that has been in physical memory the longest

- Analysis:
  - Low overhead. No need to do any work on each memory access, instead just need to do something when loading a new page into memory & evicting an existing page
  - Not the best at predicting which pages are used in the future ☹

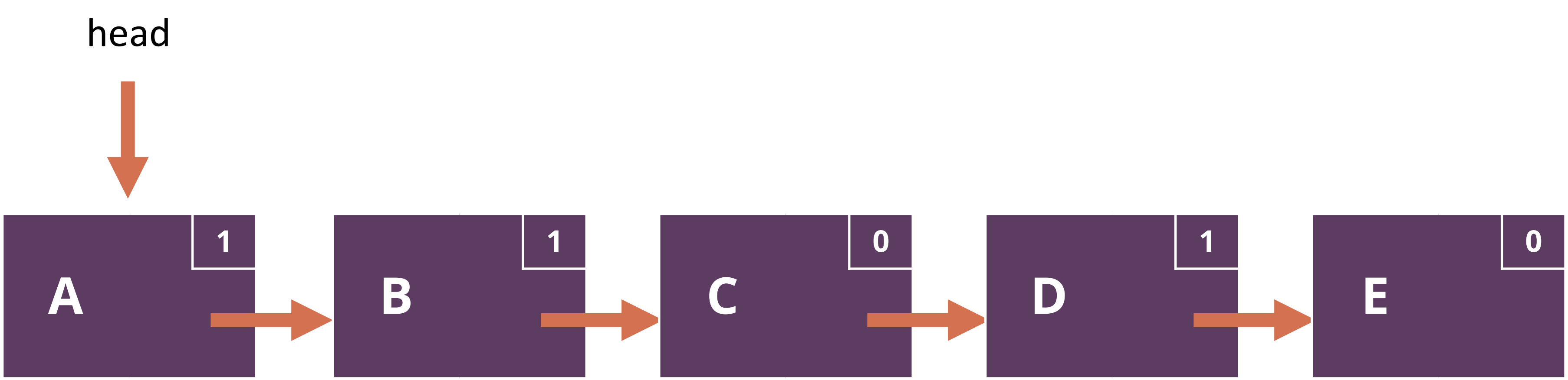
- Could we modify FIFO to better suit our needs?

# Second Chance

- Second chance algorithm is very similar to FIFO
  - Still have a FIFO queue
  - When we take the first page of the queue, instead of immediately evicting it, we instead check to see if the reference bit is 1 (was used in the last time interval)
  - If so, move it to the end of the queue
  - Repeat until we find a value that does not have the reference bit set (if all pages have reference bit as 1, then we eventually get back to the first page we looked at)

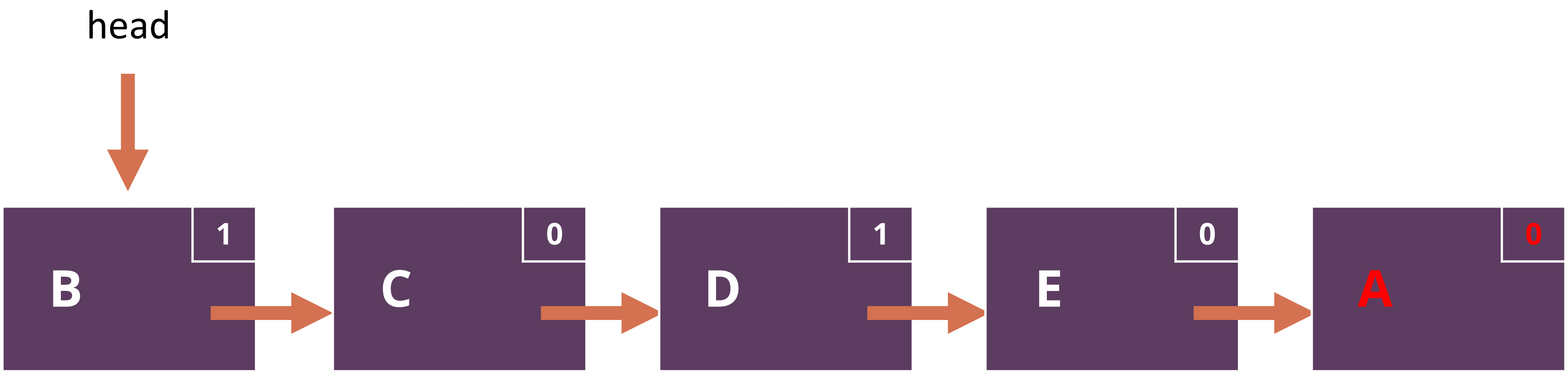
# Second Chance Example

- If we need to evict a page: start at the front
- Reference bit is 1, so set to 0 and move to end



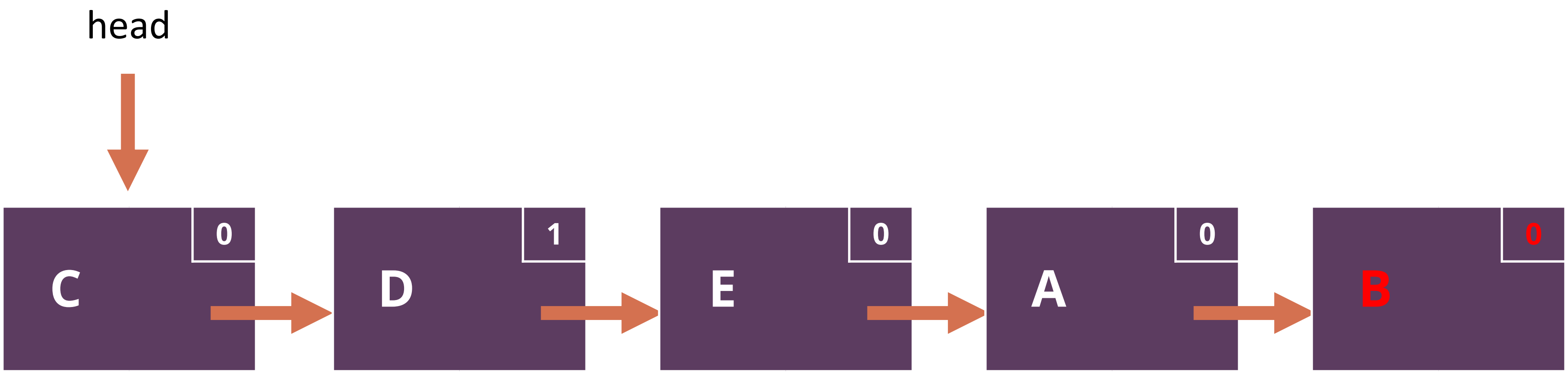
# Second Chance Example

- If we need to evict a page: start at the front
- Reference bit is 1, so move to end



# Second Chance Example

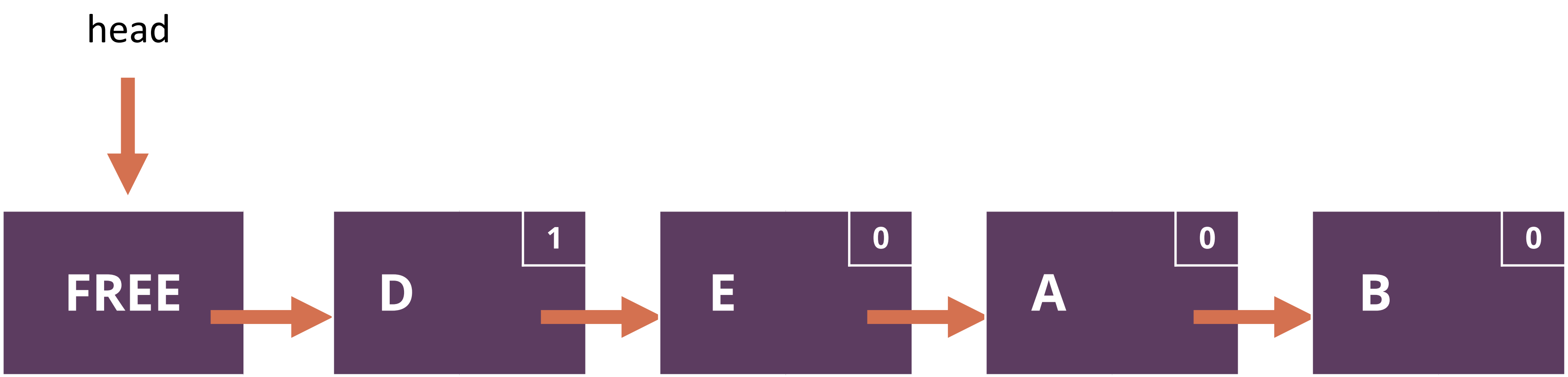
- If we need to evict a page: start at the front
- Reference bit is 1, so move to end



# Second Chance Example

- If we need to evict a page: start at the front

- Found a page with reference bit = 0, evict Page C!

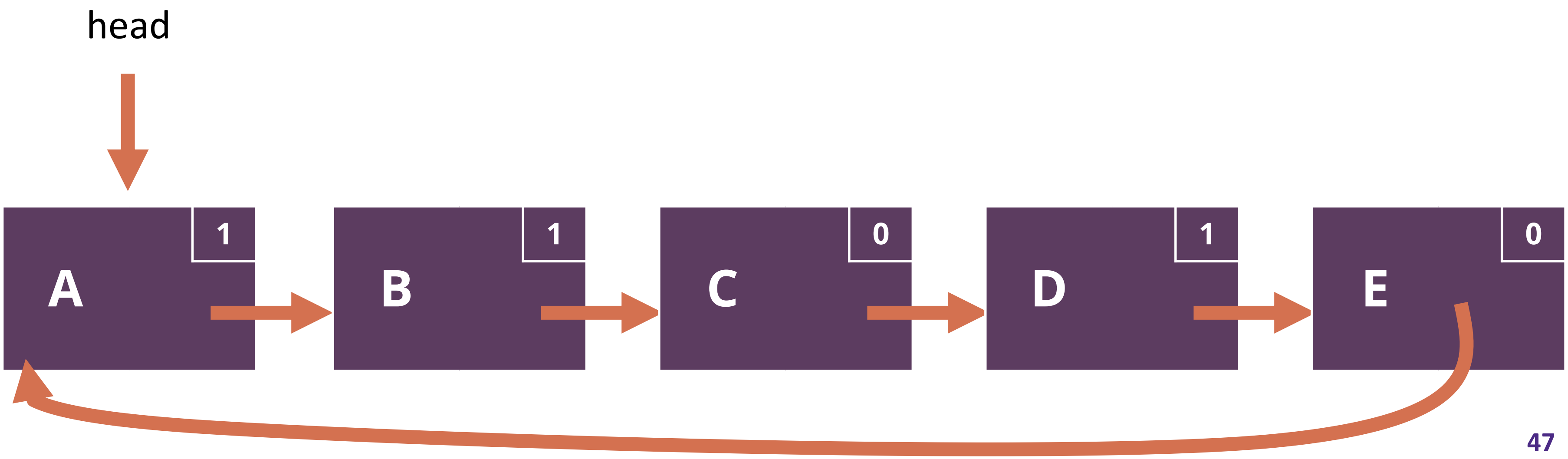


# Clock

- Optimization on the second chance algorithm
- Have the queue be circular, thus the cost to moving something to the "end" is minimal

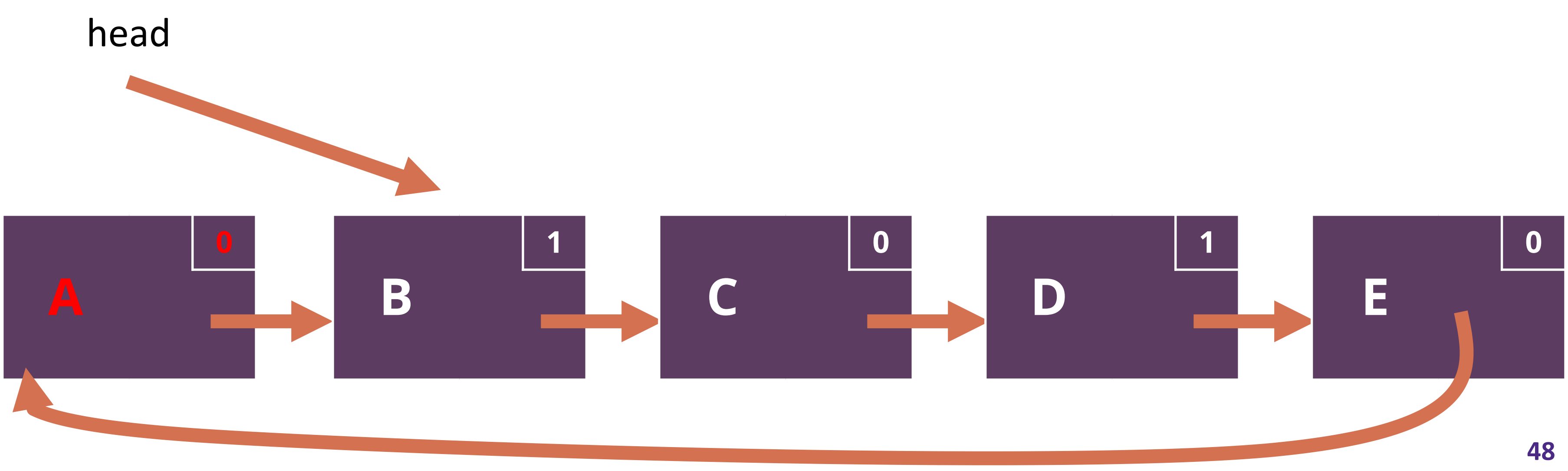
# Clock Example

- If we need to evict a page: start at the front
- Reference bit is 1, so set to 0 and move to end



# Clock Example

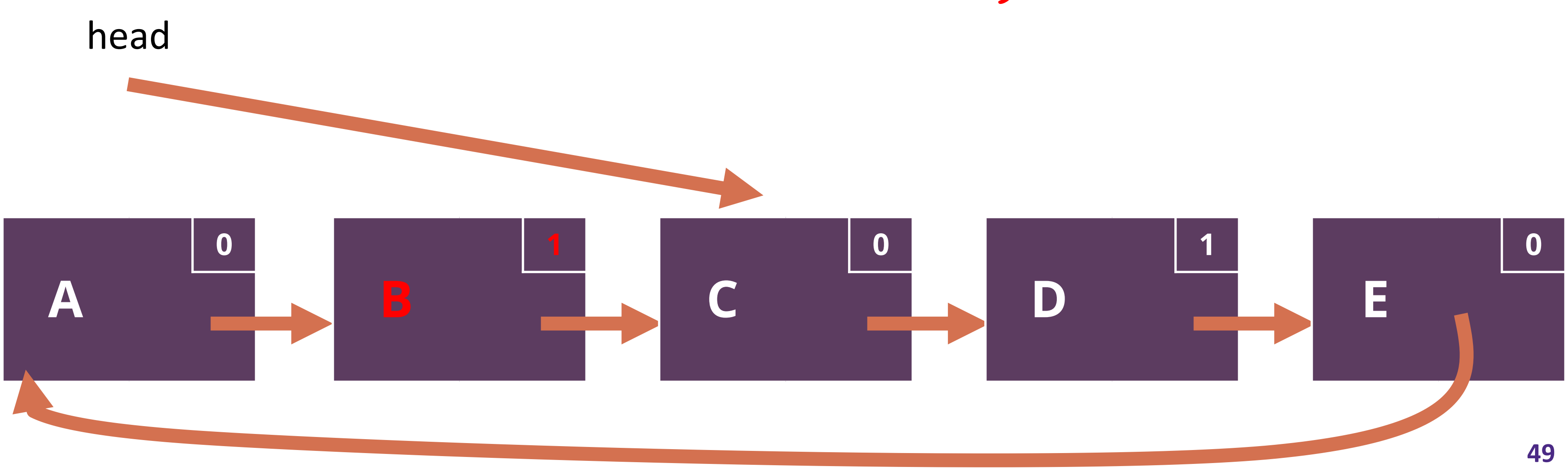
- If we need to evict a page: start at the front
- Reference bit is 1, so set to 0 and move to end



# Clock Example

- If we need to evict a page: start at the front
- Reference bit is 1, so set to 0 and move to end

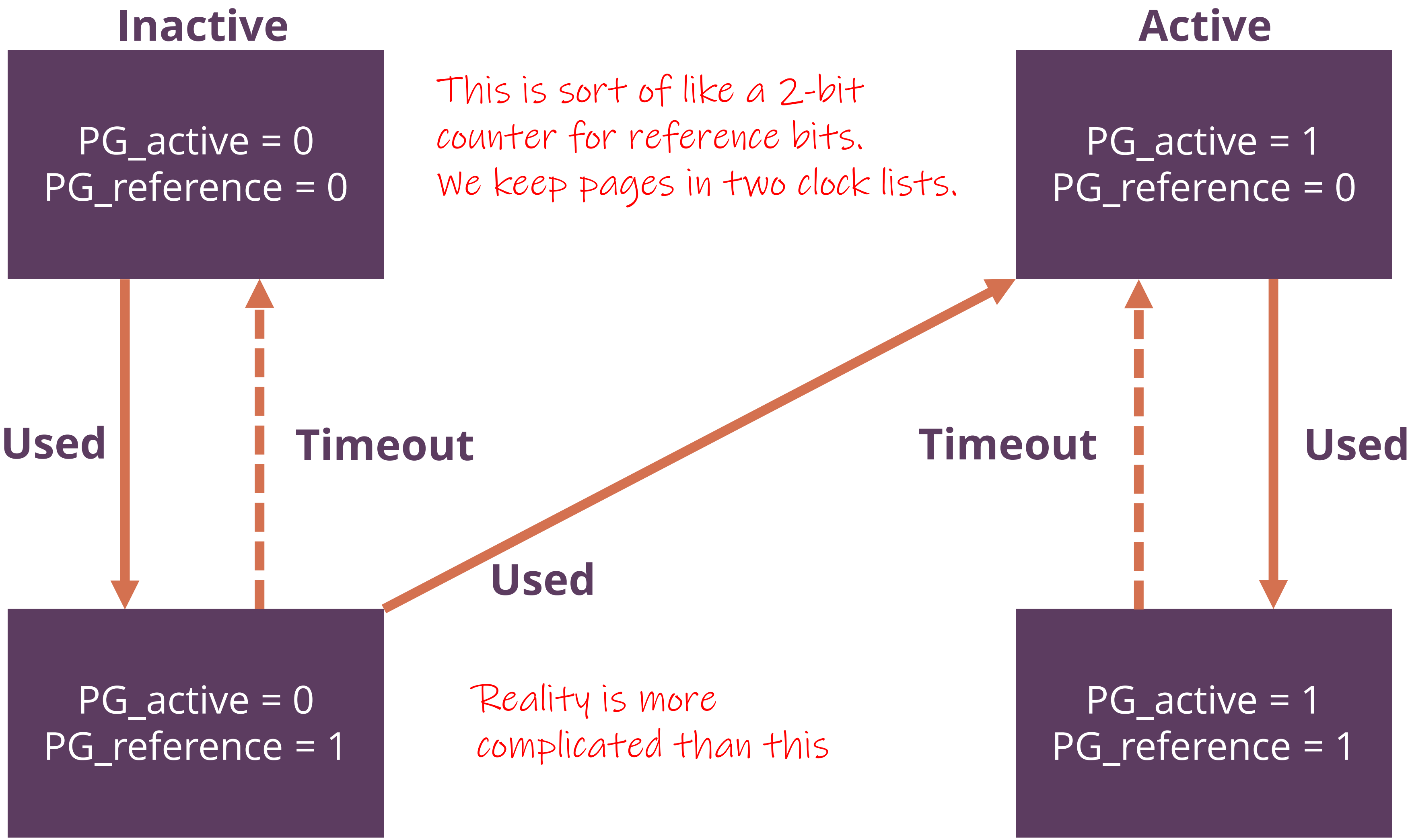
Can also be modified to prefer to evict clean pages instead of dirty pages



# Linux

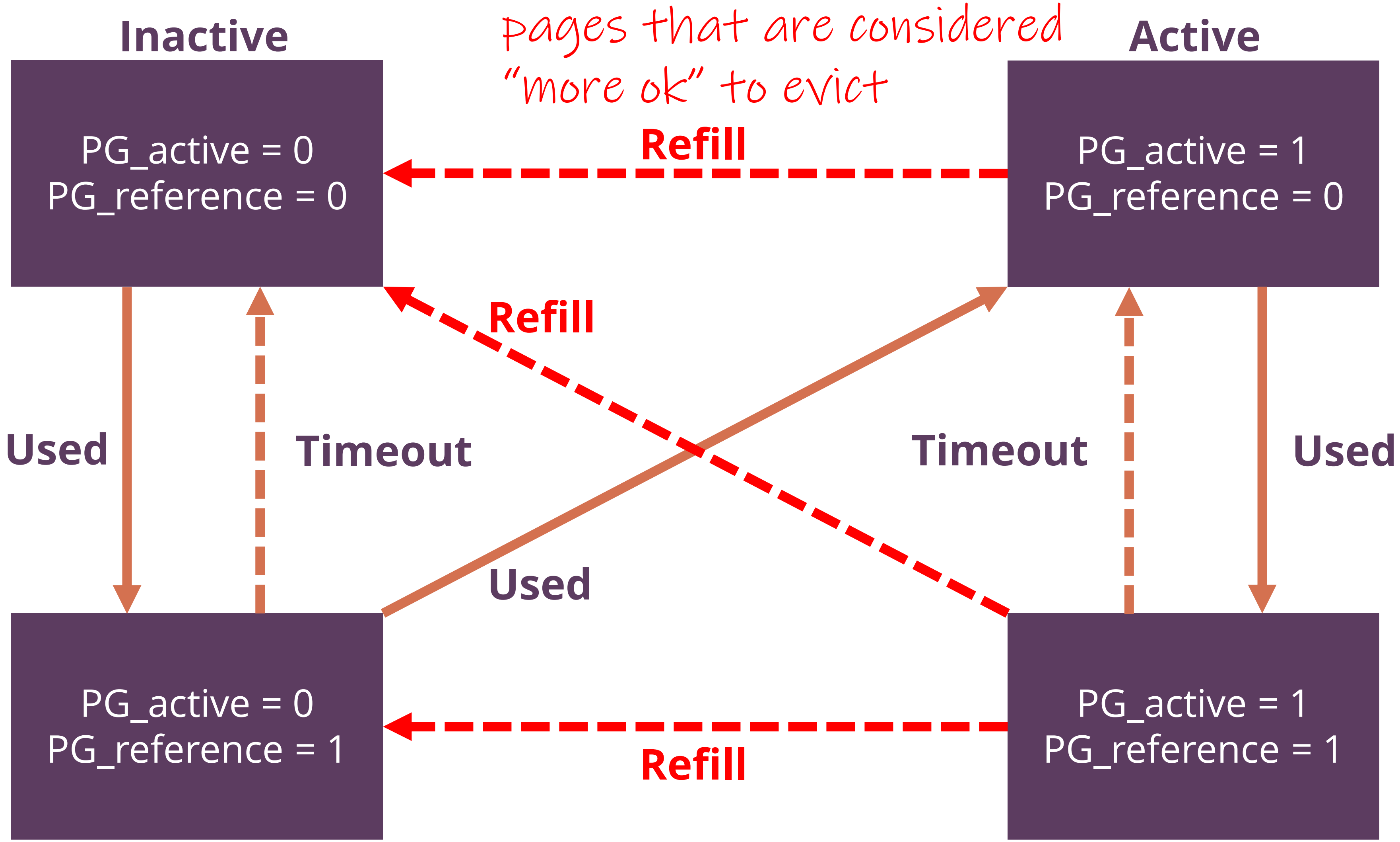
- Two Clock lists: Active and Inactive
  - Reclaim from inactive list first
  - If page has not been referenced recently, move to inactive list
  - If page is referenced:
    - Set reference flag to be true
    - Move to active list next time it is accessed
  - Two page accesses to be declared active
  - If second access does not happen, reference flag is reset periodically
- After two timeouts, move a page to inactive state

# Linux diagram



# Linux diagram

Linux will want to keep a good ratio of inactive to active, so that there are always some pages that are considered "more ok" to evict




Active should be ~2/3 of pages at most