Virtual Memory

Operating System Lab-in-a-Box

Previous

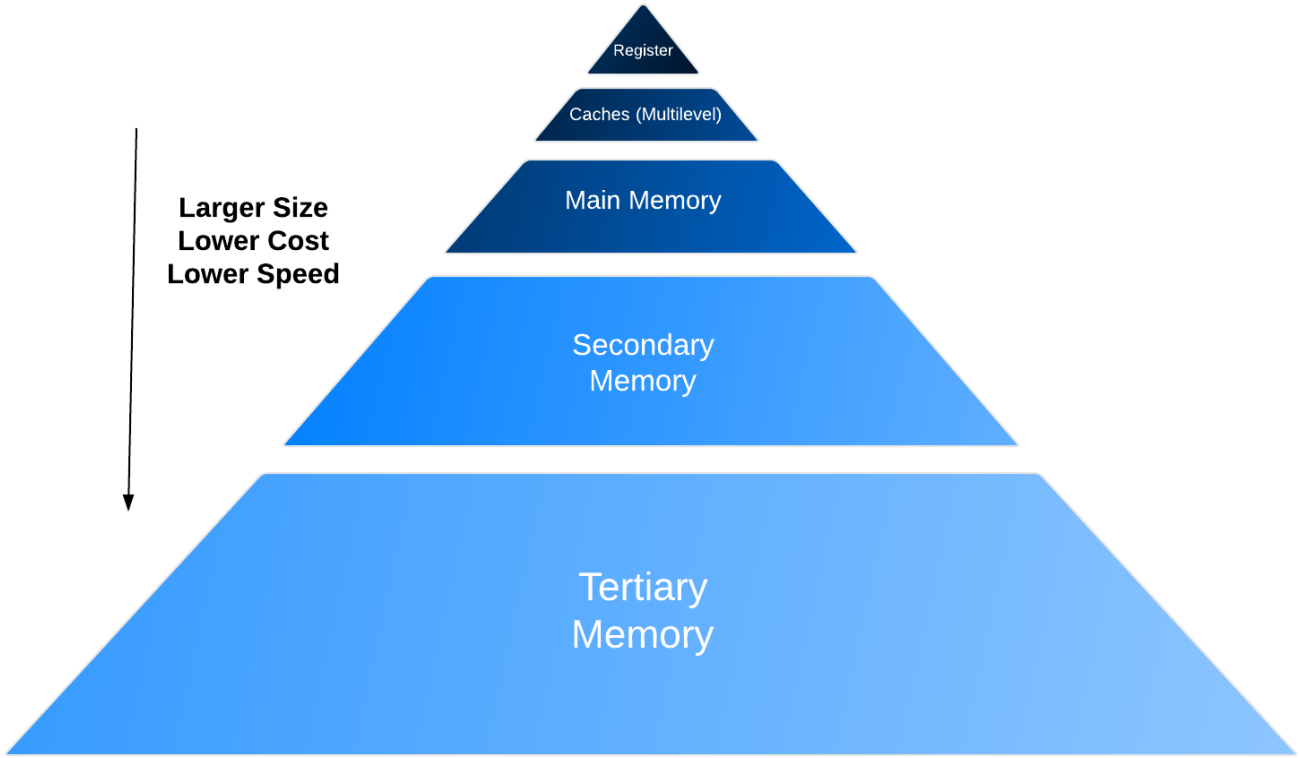
- Memory

Current Presentation

- Virtual Memory
- Paging and Handling Page Fault
- Fetching and replacing Pages
  - FIFO replacing
  - LRU Replacing
  - Clock like Replacing
  - Dynamic Locality and Working set
- Working Set Model
- Virtual Memory and multiprogramming

Memory Hierarchy

- Register: usually one CPU cycle to access
- Cache: CPU cache, TLB, Page cache
  - Static RAM
- Main Memory: the “memory” memory
  - Dynamic RAM
  - Volatile
- Secondary Memory: Hard disk
- Tertiary Memory: Tape libraries



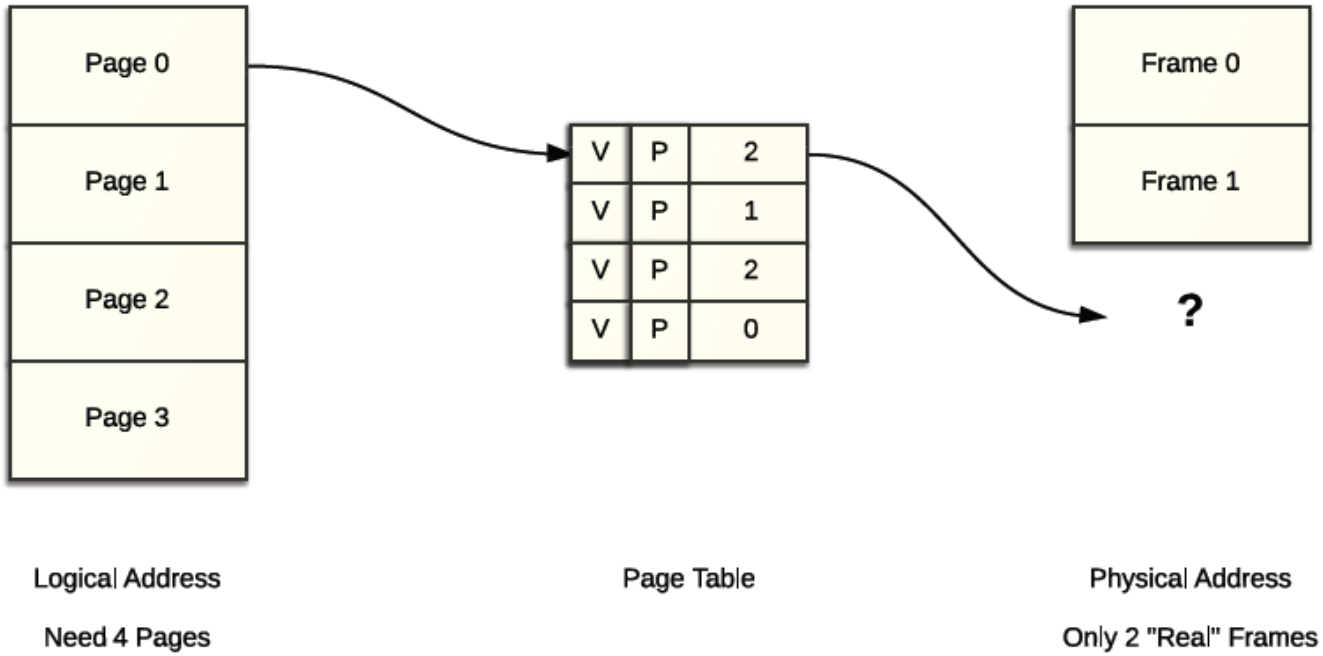
- Temporal locality
- Spatial locality

- Memory Hierarchy – to exploit the memory locality

arm

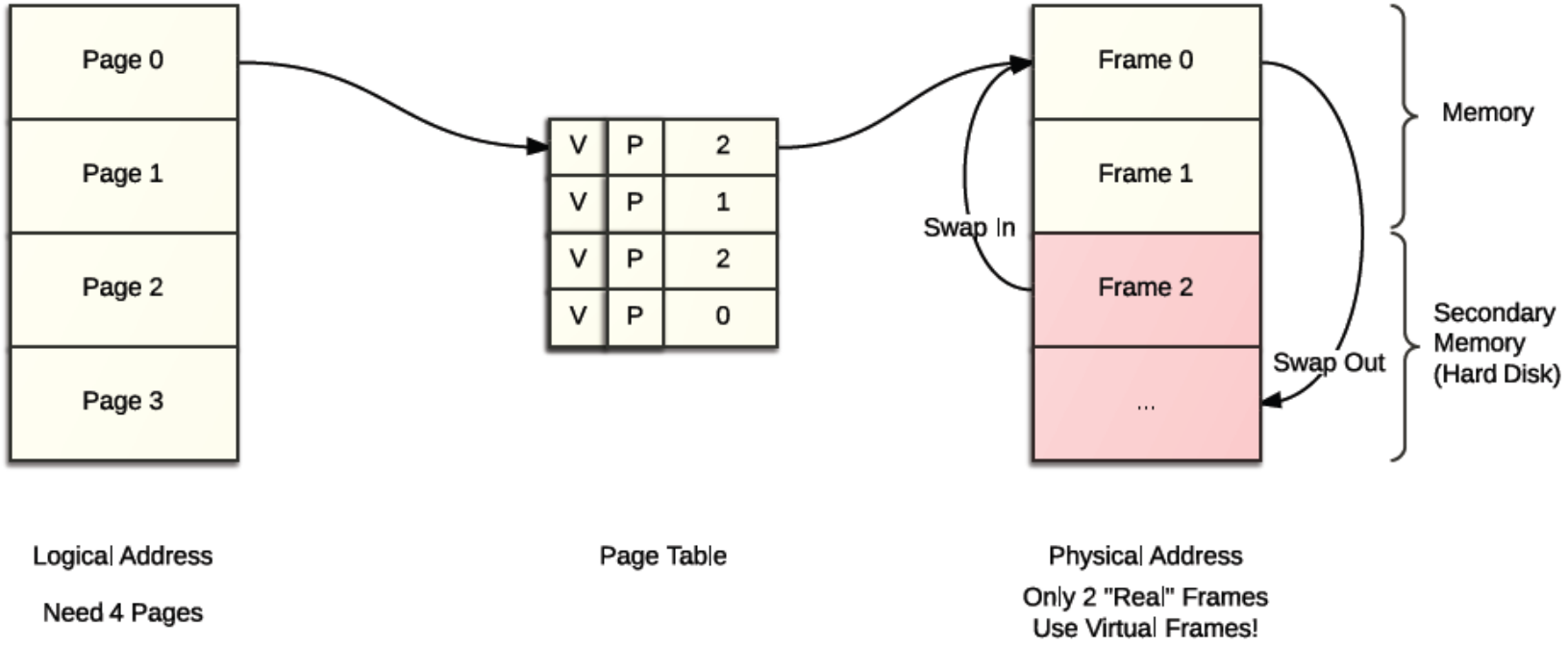
Pages>Frames?

- Programmers want more processes simultaneously and bigger memory for each
- What if the number of pages is larger than the number of frames in the memory?
- Locality:
  - A recently accessed page is more likely to be accessed again shortly (temporal)
  - Neighbour pages of the recently accessed page are more likely to be accessed again shortly (spatial)



Virtual Memory

- To exploit locality, use the main memory as a “cache”
- Provided that the page being currently executed is in memory, it is possible to push the rest to secondary memory (swap out) and get them back when needed (swap in)
- Physical address space now includes both physical memory and secondary memory
- OS now has to handle all the translations and other related overheads



Virtual Memory

- How it works with paging:
  - Initialize the page table by clearing the valid bits
  - If a page is requested and is in main memory then proceed as usual
  - If a page is requested and its entry in the page table is invalid
    - Usually dealt with in the fault(exception)-handler way, meaning this will causes a fault and the OS has to solve it before doing anything else
    - If it is on the disk, get the page from the disk otherwise allocate a new frame
    - What if no sufficient frames?
      - Also a fault
      - Invalidate some entries in the page table, swap the corresponding pages to the main memory, and free the frames for use
- Poses challenges to hardware(MMU, PT) design
  - Record control related information: modified(dirty bit)? In hard disk? Recently accessed?
  - Store them on PT or in the real frame?
  - Coherency problem

What Does (Virtual) Memory Management Do

- Handle the fault
- Fetch – Decide when to fetch a page, which page to fetch – is it a problem?
- Replacement – Which page(s) to swap out? From the same process or not?
- Resident Set – How many pages can a process have at most? How many of them can be resident in the main memory?

- Good designs:
  - In general, utilize the locality of the memory
  - Minimize page faults (avoid the trap by keeping more needed pages in the memory)
  - Minimize I/O activities (simply drop a clean page when needed to be swapped out)

Handling a page fault

- Block the current process
- 1. The frame is in the disk
  - OS requests a disk read to fetch the frame into main memory
  - Dispatch another process
  - Locate and fetch the page into a free frame, and when ready interrupt the OS
  - OS updates the page table
  - Unblock related process
- 2. If not in the disk/memory, gets harder, use the dirty bit that records if a frame is modified
  - Decide a frame to be swapped out
  - Invalid the entry in the page table
  - If dirty, write the frame back to the disk
  - If clean, drop the page (do nothing)
  - Allocate free frame for the new request
  - Update the page table
- Resume everything

Fetching and Prefetching Pages

- When?
  - When a page is needed
- Which?
  - The required page
- Not only on demand basis but can also prefetch some pages – prepaging
  - Predict which pages will be more likely to be referenced and bring them back in advance
  - Or fetch a group of highly relevant pages including the needed page when swapping in
  - Windows (prefetcher) and Linux have this feature
  - But still arguable if effective

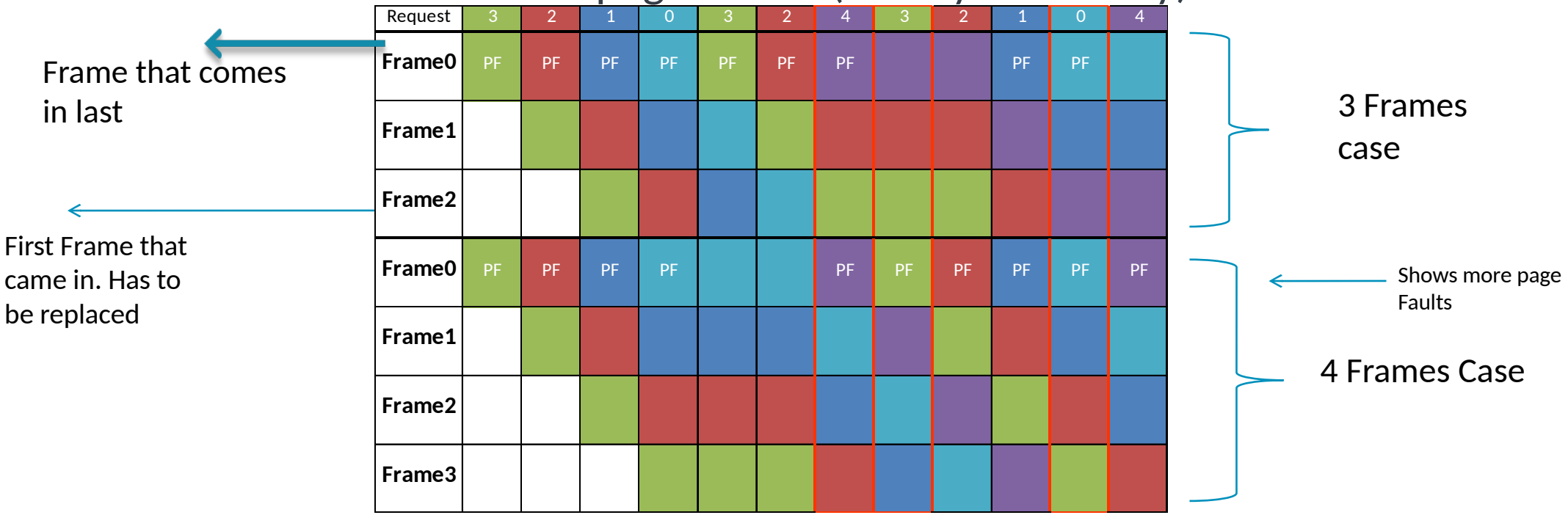
Policies for Replacing Pages

- Swap out a local page or a global page? Global can seemingly avoid page faults
- Swap out those with the longest time until their next access will be probably optimal.
  - Hard to predict and thus impractical
- Policies are usually evaluated and compared using the page faults/page accesses
- Generally, growing the number of frames will drop the page faults/page access towards zero asymptotically
- Except the Bélády's anomaly – page fault may increase if the replacing policy violate the inclusion property
- Inclusion property – if the algorithm maintains *m* pages in the *m* frame size memory, then the algorithm should include at least exactly the same *m* pages in any memory that has *n* frames where $n > m$.

- Some practical policies: random, FIFO, LRU, clock

FIFO Replacing

- Keep a first-in-first-out (FIFO) queue and swap out the oldest pages.
- Easy to implement and fair.
- But tends to swap out frequently used pages and violates the inclusion property.
- More frames could lead to more page faults! (Bélády's anomaly)



LRU Replacing

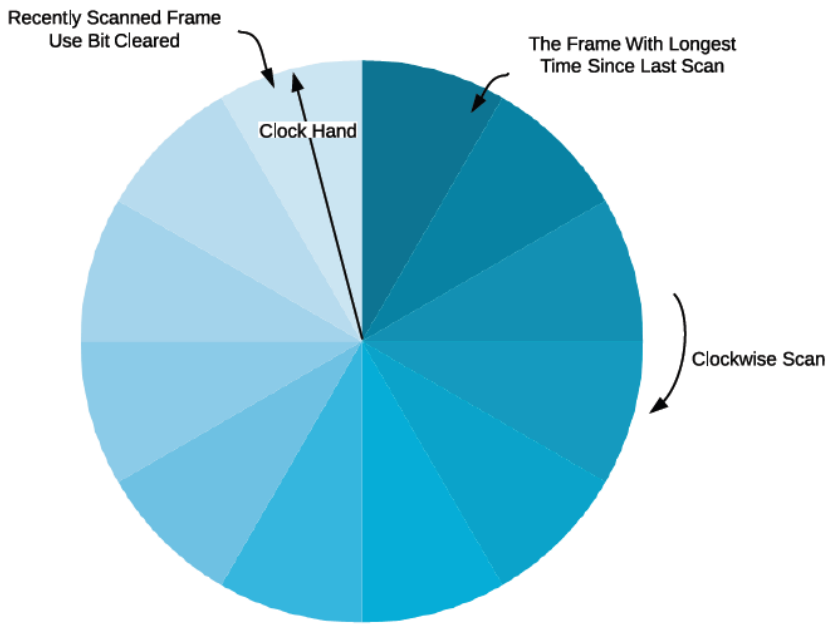
- Replace the Least Recently Used page (LRU). Temporal locality implies that it will be least likely to be reused soon.
- Maintains inclusion property. More frames will at least not increase page faults.
- Theoretically close to optimal.

Request	3	2	1	0	3	2	4	3	2	1	0	4
Frame0	PF	PF	PF	PF	PF	PF	PF			PF	PF	PF
Frame1												
Frame2												
Frame0	PF	PF	PF	PF			PF			PF	PF	PF
Frame1												
Frame2												
Frame3												

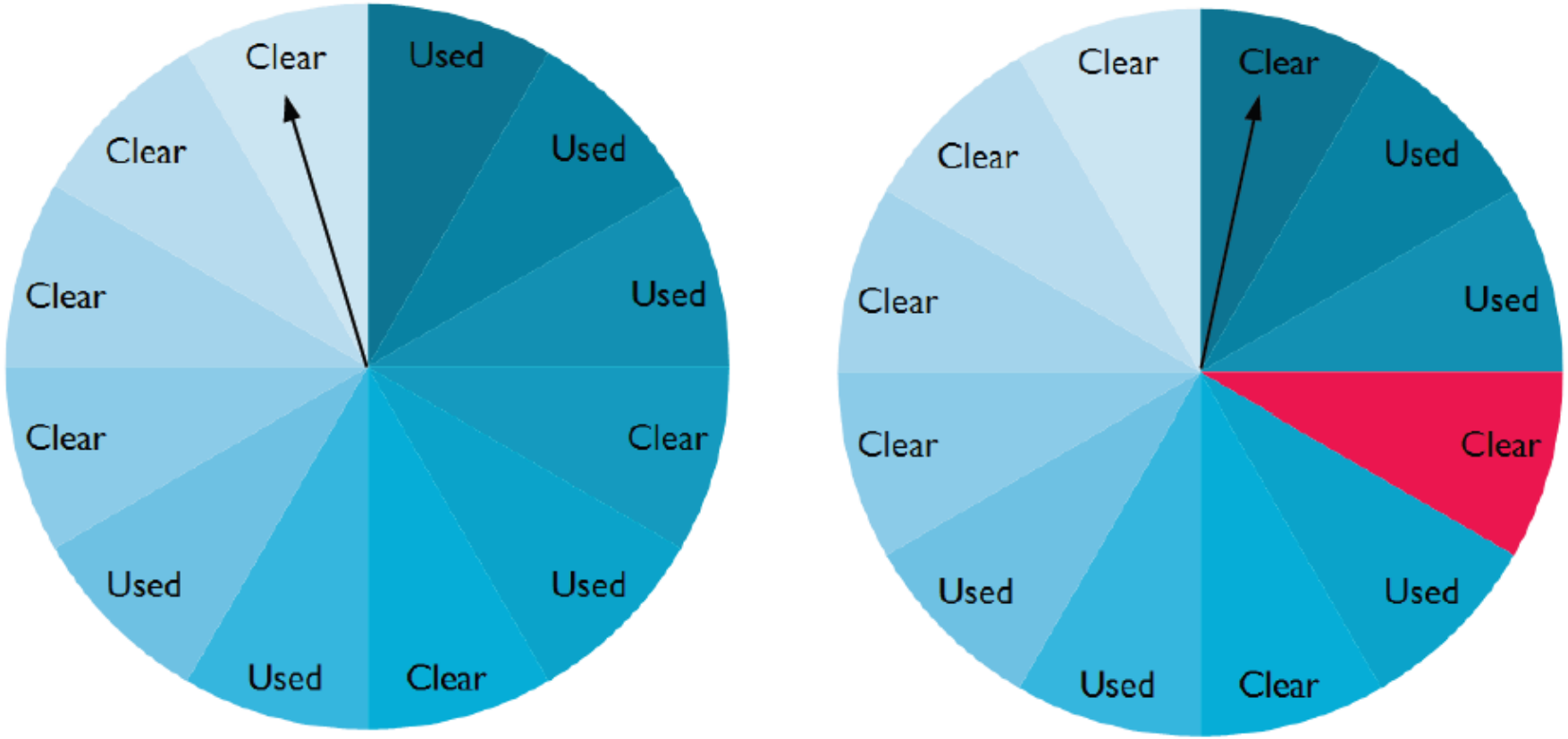
- Keeping track of the sequence, is however difficult.

Clock-like Replacing

- Approximated version of LRU. Practical and commonly used.
- Use the dirty bit again. Also an additional use bit. Whenever a page is referenced to, set the use bit in the page table entry.
- OS keeps a circular FIFO queue and a clock hand pointer.
- Each time a page is accessed, the use bit of corresponding page table entry will be set, clock hand pointer will scan the next frame and clear its use bit.
- So in terms of an individual entry, its use bit will be cleared periodically. If an entry is clear, then the page has not been used recently.
- When OS is looking for a page to be swapped out, clock hand pointer scans through the cycle until it finds the first:
  - clean page, do not have to swap out and just drop the page or
  - unused page, swap it out



Clock-like Replacing

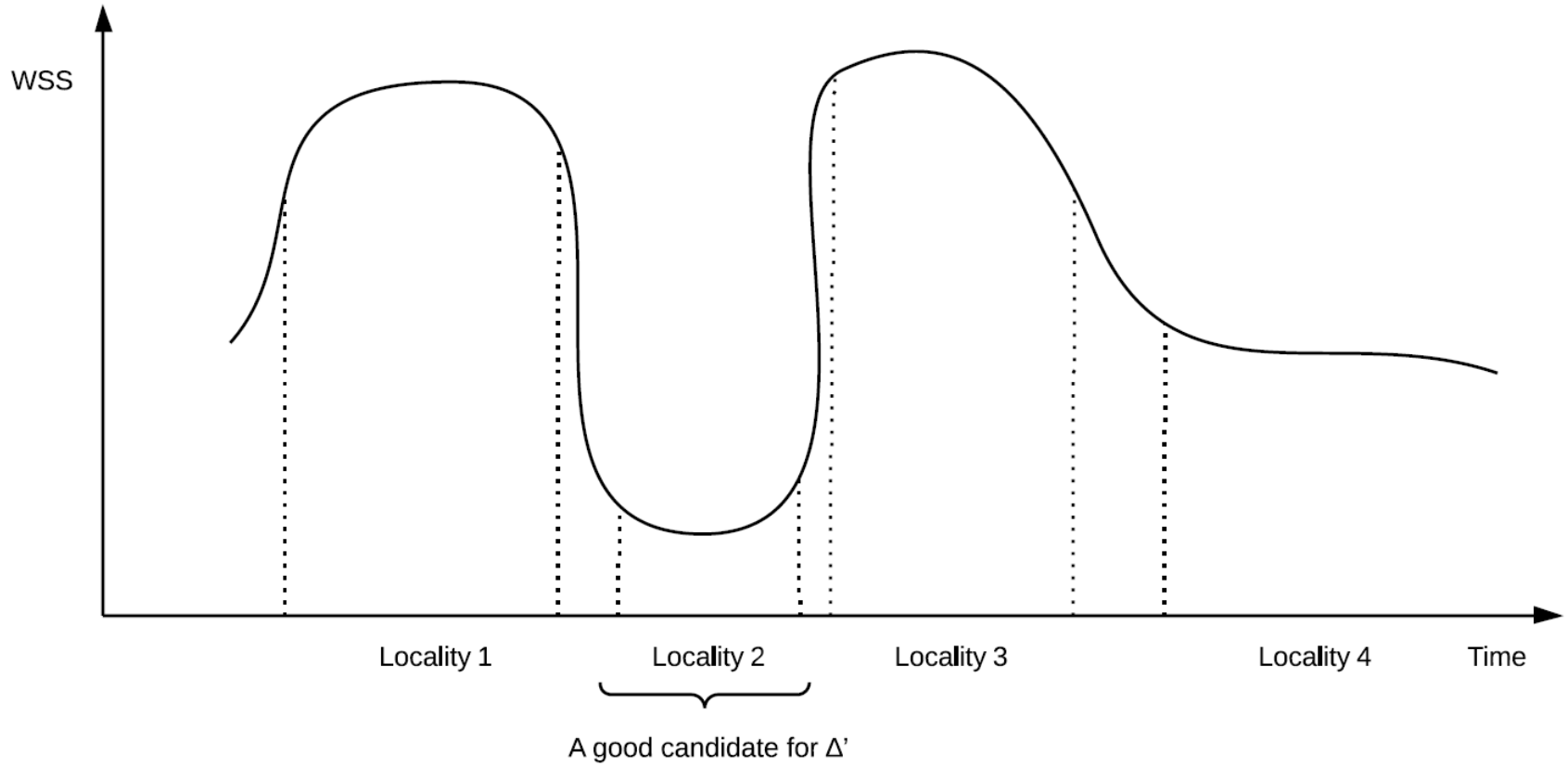


- An access to the page will result in a move of the clock hand pointer, if the next page is not the one requested, clear its use bit. (Frame 1)
- Page 4(red) will be selected if a free frame is needed as the pointer will scan through frame 2 and frame 3, finding them used.

Dynamic Locality And Working Set

- How many pages can a process own? This is a dynamic number.
- A process maintains locality and at the same time could be dynamically changing in the long run.
- To capture the dynamic locality, the concept of the “working set” is proposed.
  - *W(Δ, t)* – Set of pages accessed by a process over the time interval[t- Δ, t].
  - The working set size (WSS), generally grows with Δ.
  - However it is possible to identify a small enough Δ’ after which the WSS does not grow too much.
- Working set will remain a specific locality for a while and then migrates to another locality. Proper Δ’ helps neglect the effect of the interlocality transitions and capture the WSS of the most recent locality.

Dynamic Locality And Working Set



- A process may have several distinctive working modes.
- For each working mode, the WSS will reach a stable state and settle for a while.
- Anything between the stable states is the transitions.

Working Set Model

- Resident set, the pages in main memory, is a subset of the working set.
- An ideal situation is two sets are identical, i.e., entire working set is available. Thus no page faults.
- OS can manage to do that by:
  - Keep track of the working set of all processes
  - Allocate WSS frames to the process
  - Swap an entire process out of memory is getting tight, e.g., >Number of all frames
  - Remove stale pages if OS notice WSS is decreasing.
  - Only accept new/suspended processes with the condition that Number of all frames.

- Practical?
  - The price of recording WSS is high.

Working Set Model

- Accept some page faults
- Approximated solution:
  - Monitor page fault rate *r*
  - *r* = 1/(recent page fault time – last page fault time)
  - Set up an upper bound *U* and lower bound *L*
  - If *r* > *U*, add a page to WSS
  - If *r* < *L*, shrink WSS
  - Swap the process out if *r* > *U* and no free frames

Virtual Memory And Multiprogramming

- Conceptually increase the memory space
- Increase the degree of multiprogramming as sometimes only part of the program is necessary to be loaded
- As an extension of ordinary paging scheme, virtual memory better facilitates page protection and sharing

- However needs explicit support from both hardware and OS.

- User's mental model is not coherent with reality as OS conceals the low level details.

Next

- Cache