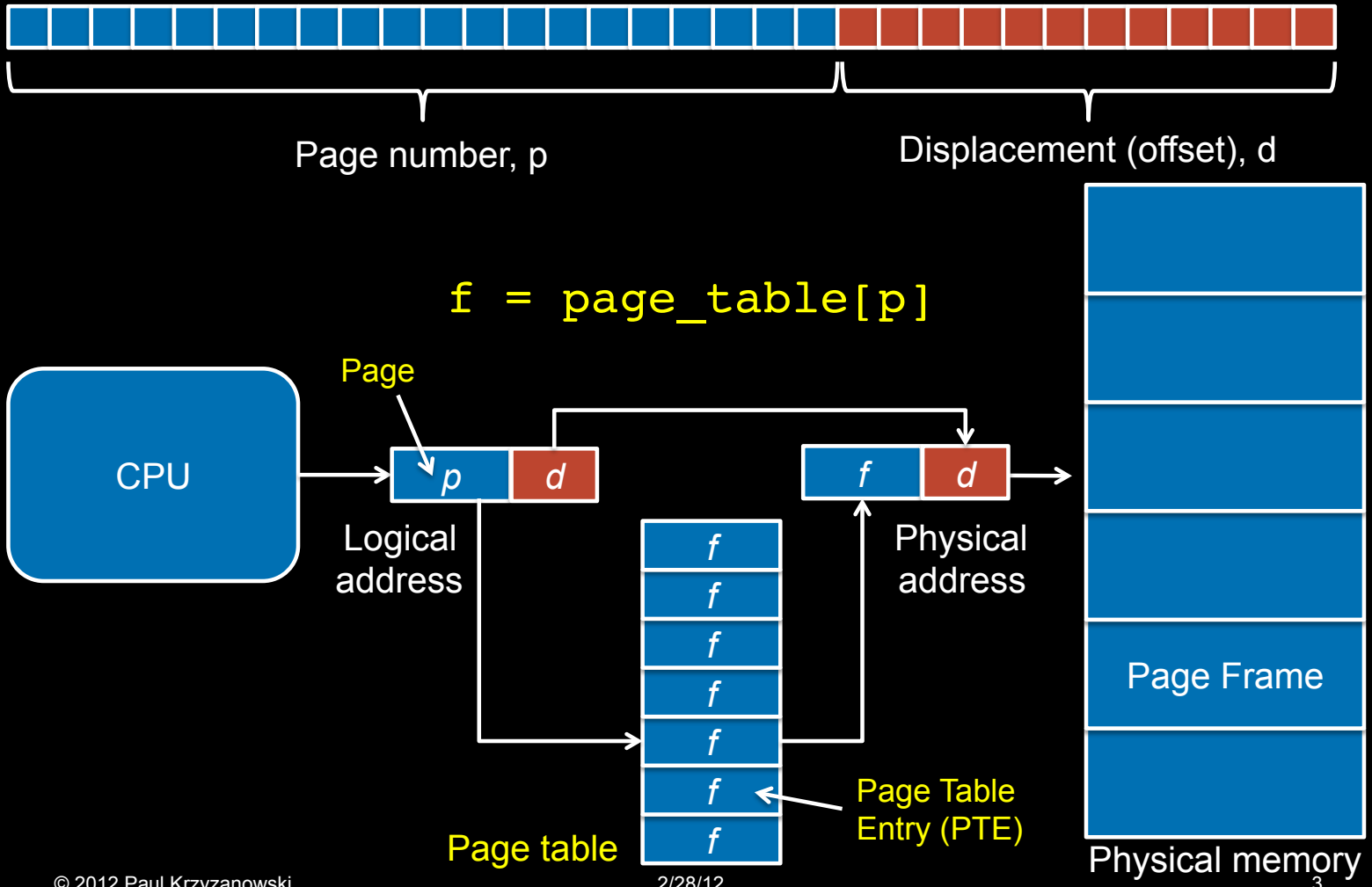
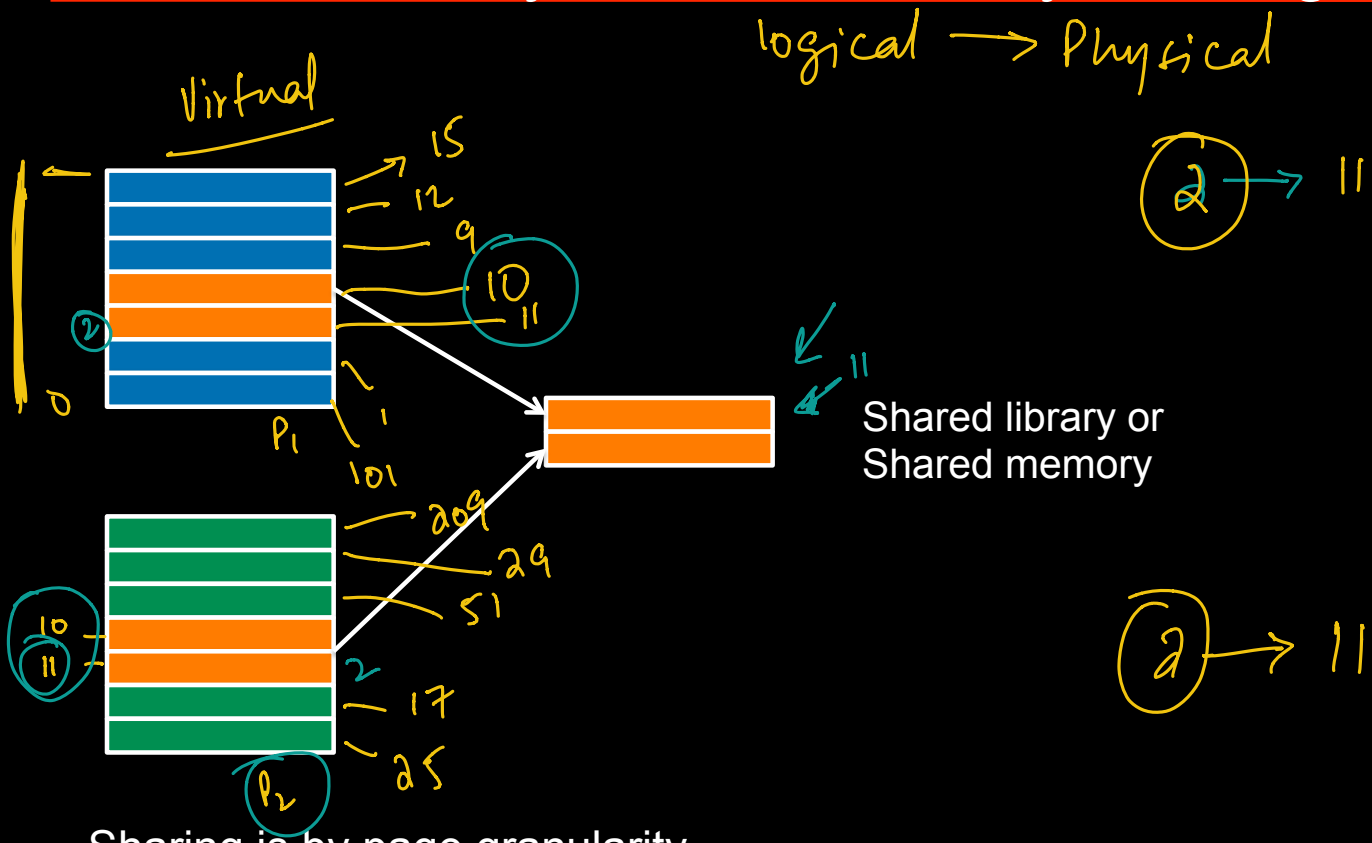
Operating Systems Design
10. Memory Management: Paging

Paul Krzyzanowski
pxk@cs.rutgers.edu

Page translation

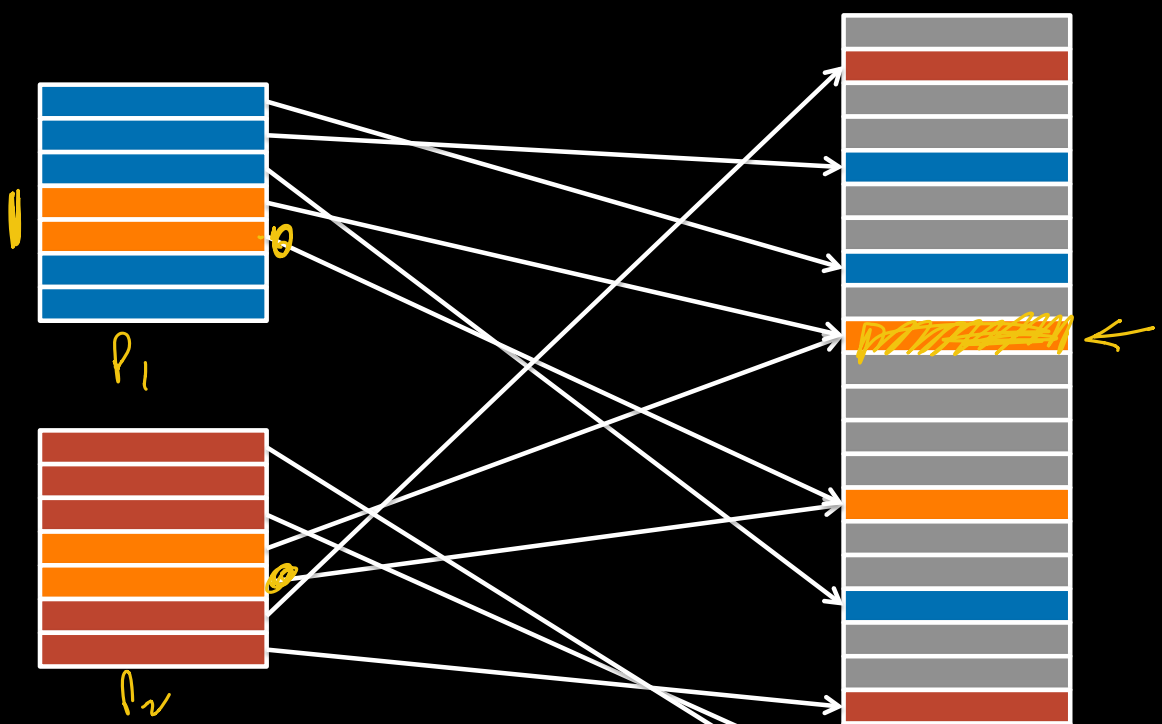


Virtual memory makes memory sharing easy



Sharing is by page granularity.

Virtual memory makes memory sharing easy

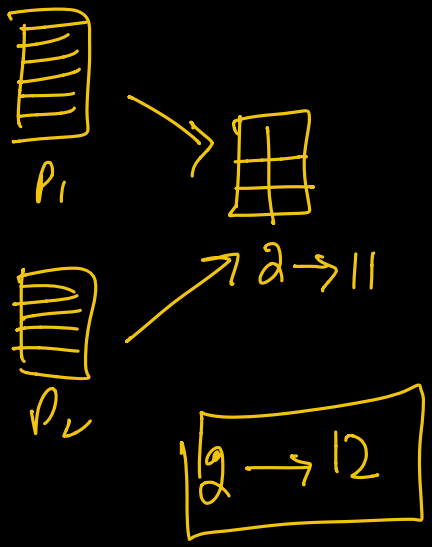


Sharing is by page granularity.

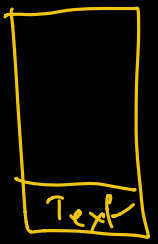
Keep reference counts!

Copy on write

- Share until a page gets modified
- Example: fork()
  - Set all pages to read-only
  - Trap on write
  - If legitimate write
    - Allocate a new page and copy contents



Q: Do we keep the whole process image in RAM?



Demand Paging

Executing a program

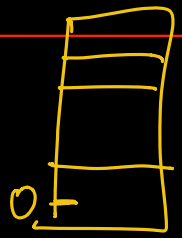
- Allocate memory + stack and load the entire program from memory (including linked libraries)
- Then execute it

Executing a program

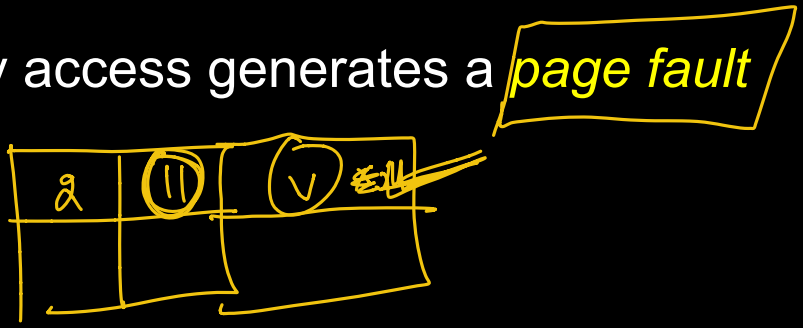
- Allocate memory + stack and load the entire program from memory (including linked libraries)
- Then execute it

We don’t need to do this!

Demand Paging

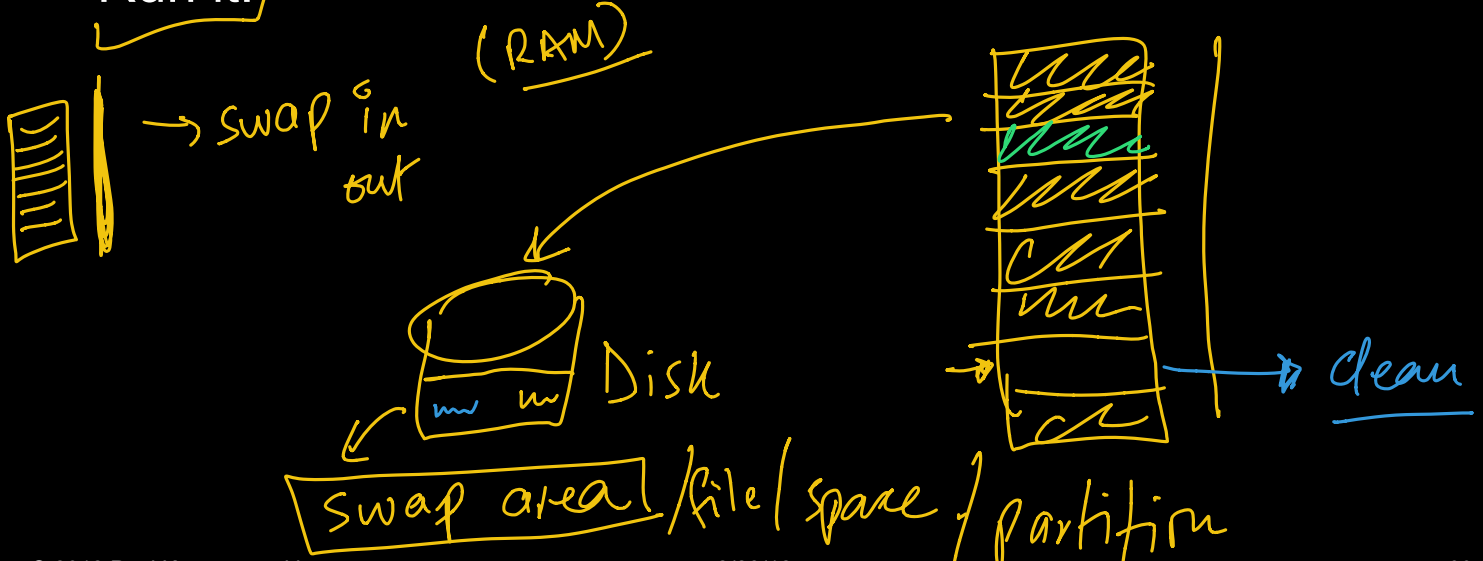


- Load pages into memory only as needed
  - On first access
  - Pages that are never used never get loaded
- Use valid/invalid bit in page table entry
  - Valid: the page is in memory ("valid" mapping)
  - Invalid: out of bounds access or page is not in memory
    - ~~Have to check the process' memory map in the PCB to find out~~
- Invalid memory access generates a *page fault*



Demand Paging: At Process Start

- Open executable file
- Set up memory map (stack & text/data/bss)
  - But don't load anything!
- Load first page & allocate initial stack page
- Run it!



Demand Paging: Page Fault Handling

- Soon the process will access an address without a valid page
  - OS gets a page fault from the MMU
- What happens?
  - Kernel searches a tree structure of memory allocations for the process to see if the faulting address is valid
    - If not valid, send a SEGV signal to the process
  - Is the type of access valid for the page?
    - Send a signal if not
  - We have a valid page but it's not in memory
  - Bring page in, mark as valid. ← restart instruction

Page Replacement

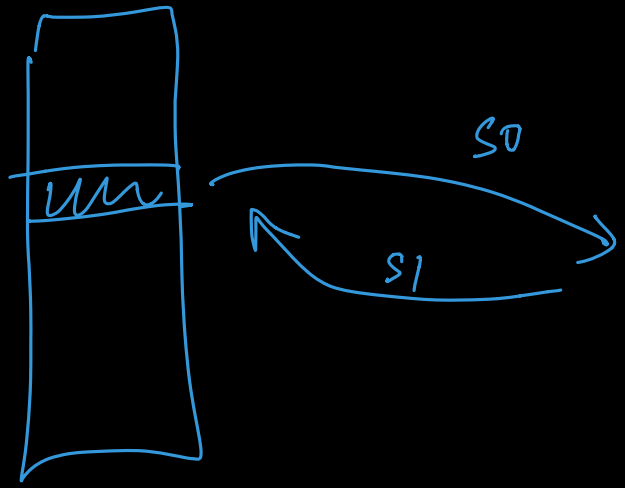
- A process can run without having all of its memory allocated
  - It's allocated on demand
- If the
  {address space used by all processes + OS} ≤ physical memory
  then we're ok
- Otherwise:
  - Make room: discard or store a page onto the disk
  - If the page came from a file & was not modified
    - Discard … we can always get it
  - If the page is dirty, it must be saved in a swap file
  - Swap file: a file (or disk partition) that holds excess pages

Cost

- Handle page fault exception: ~ 400 usec
- Disk seek & read: ~ 10 msec
- Memory access: ~ 100 ns
- Page fault degrades performance by around 100,000!!
- Avoid page faults!
  - If we want < 10% degradation of performance, we can have just one page fault per 1,000,000 memory accesses

Page replacement

- We need a good replacement policy for good performance



FIFO Replacement

- First In, First Out
- Good
  - May get rid of initialization code or other code that's no longer used
- Bad
  - May get rid of a page holding frequently used global variables

Least Recently Used (LRU)

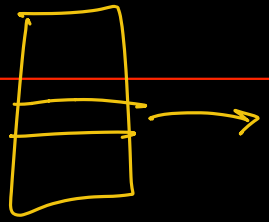
- Timestamp a page when it is accessed
- When we need to remove a page, search for the one with the oldest timestamp

- Nice algorithm but…
  - Timestamping is a pain – we can't do it with the MMU!

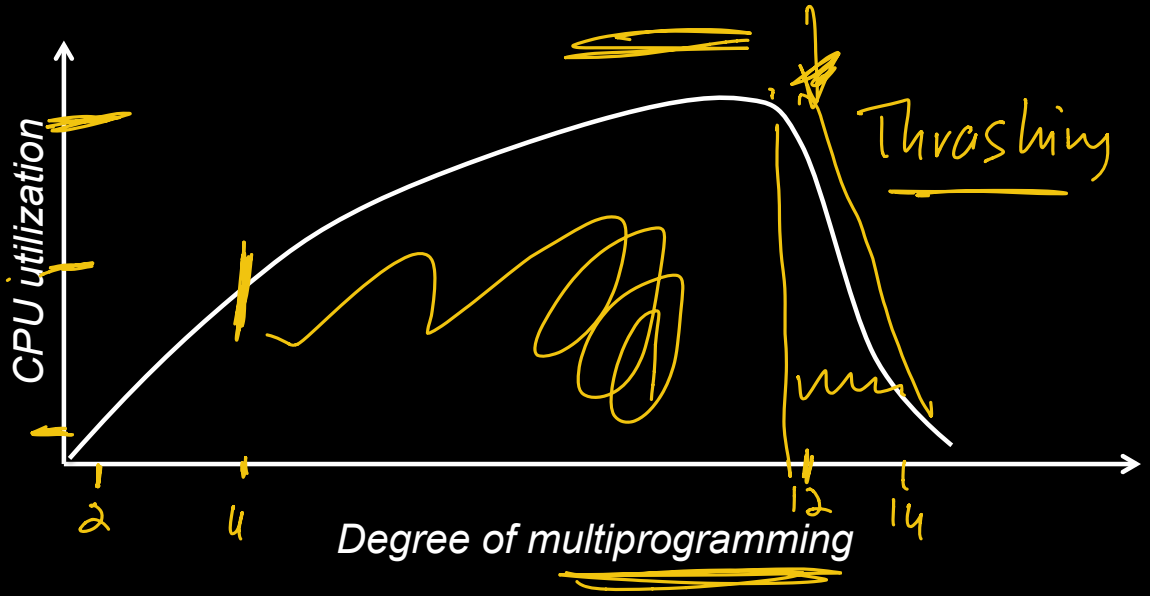
Not Frequently Used Replacement

- Approximate LRU
- Each PTE has a reference bit
- Keep a counter for each page frame
- At each clock interrupt:
  - Add the reference bit of each frame to its counter
  - Clear reference bit
- To evict a page, choose the frame with the lowest counter
- Problem
  - No sense of time: a page that was used a lot a long time ago may still have a high count
  - Updating counters is expensive

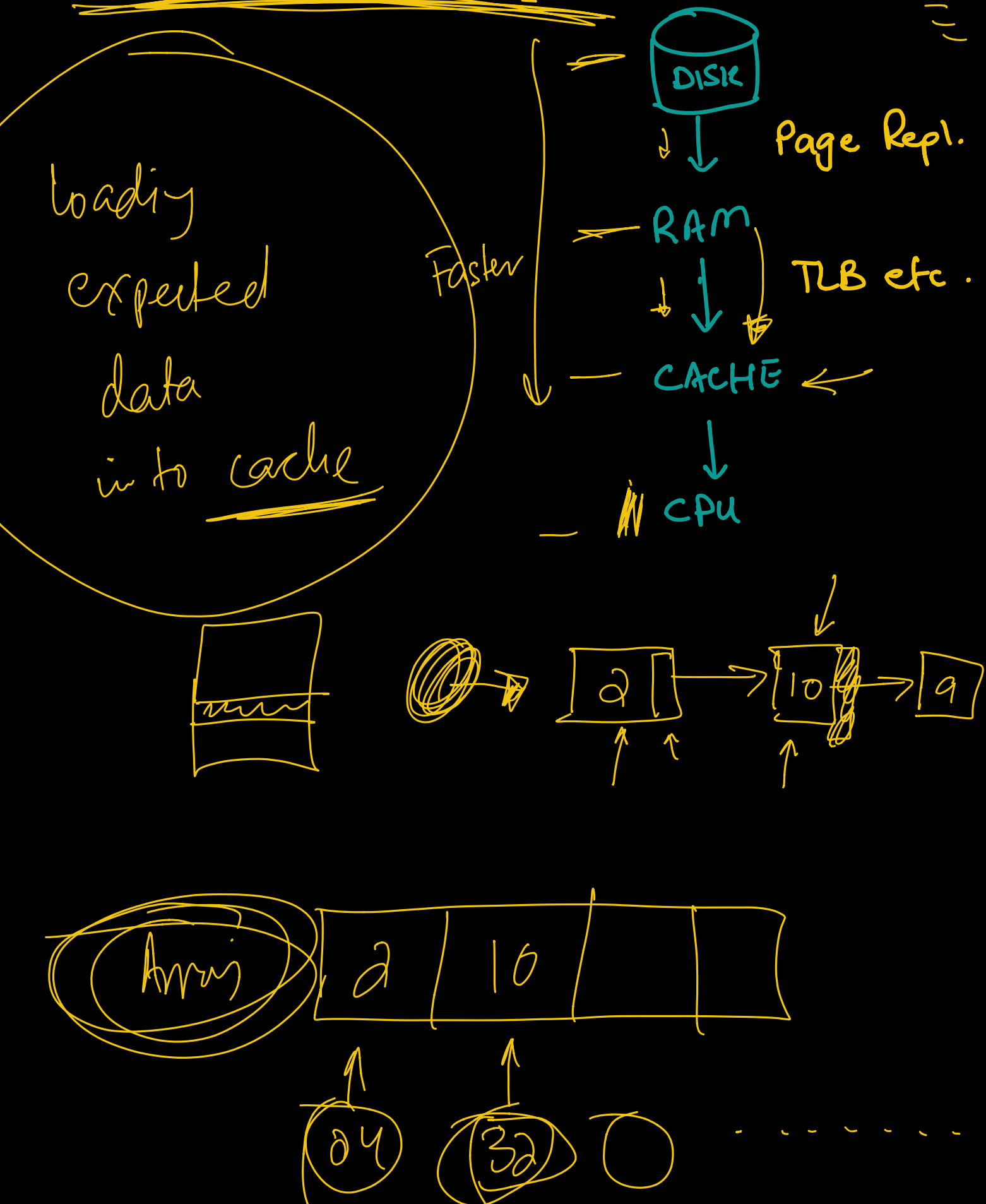
Thrashing



- Locality
  - Process migrates from one locality (working set) to another
- Thrashing
  - Occurs when sum of all working sets > total memory



Cache Coherence



The End