University of Pittsburgh

Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

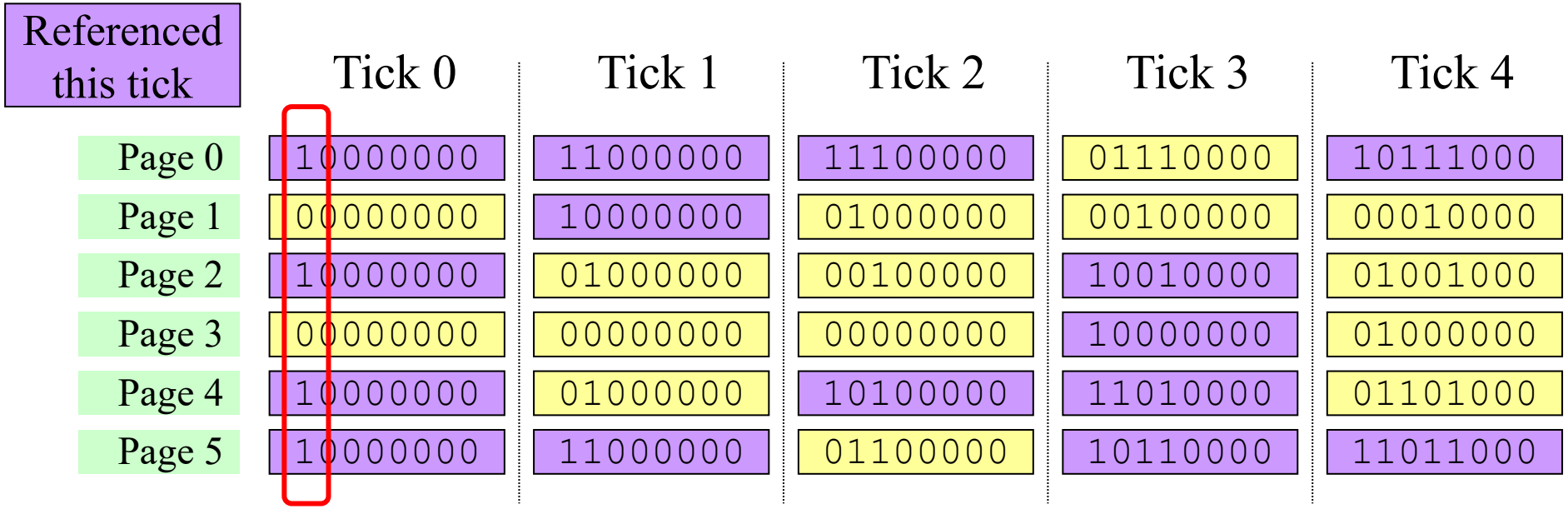
- Upcoming deadlines
  - Quiz 2 and Homework 8 due this Friday
    - 2 extra attempts for HW 8
  - Lab 3 is due on Tuesday 3/28 at 11:59 pm
  - Project 3 is due Friday 4/7 at 11:59 pm

Previous Lecture …

- Page replacement algorithms
  - OPT, NRU, FIFO, Second Chance, CLOCK, LRU, NFU

Aging replacement algorithm

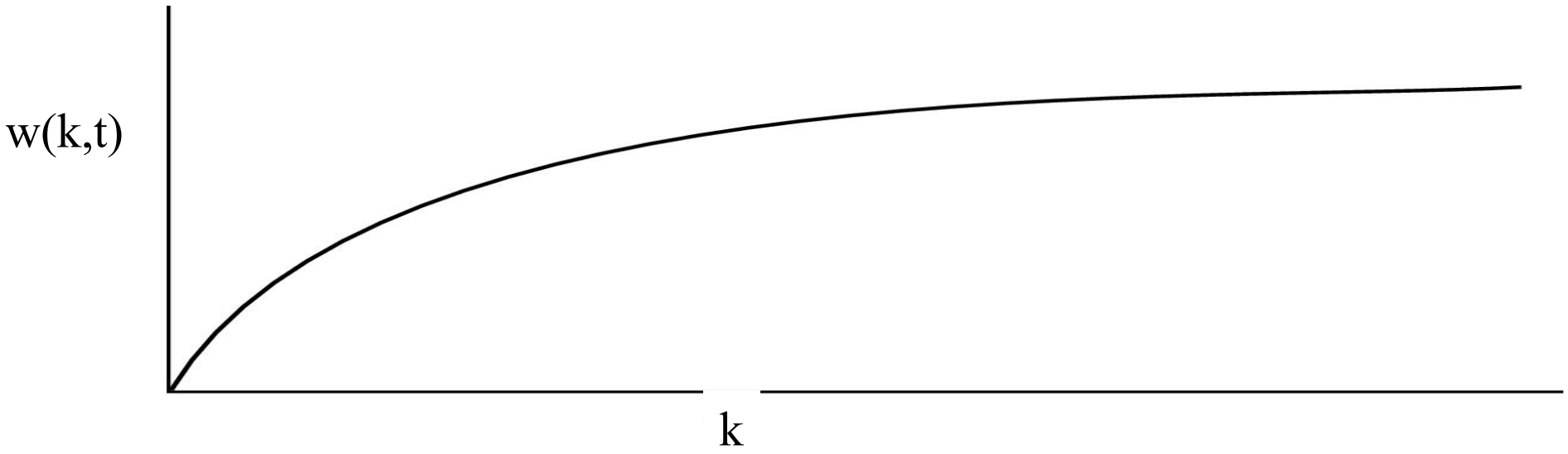
- Reduce counter values over time
  - Divide by two every clock cycle (use right shift)
  - More weight given to more recent references!
- Select page to be evicted by finding the lowest counter value
- Algorithm is:
  - Every clock tick, shift all counters right by 1 bit
  - On reference, set leftmost bit of a counter (can be done by copying the reference bit to the counter at the clock tick)



Working set

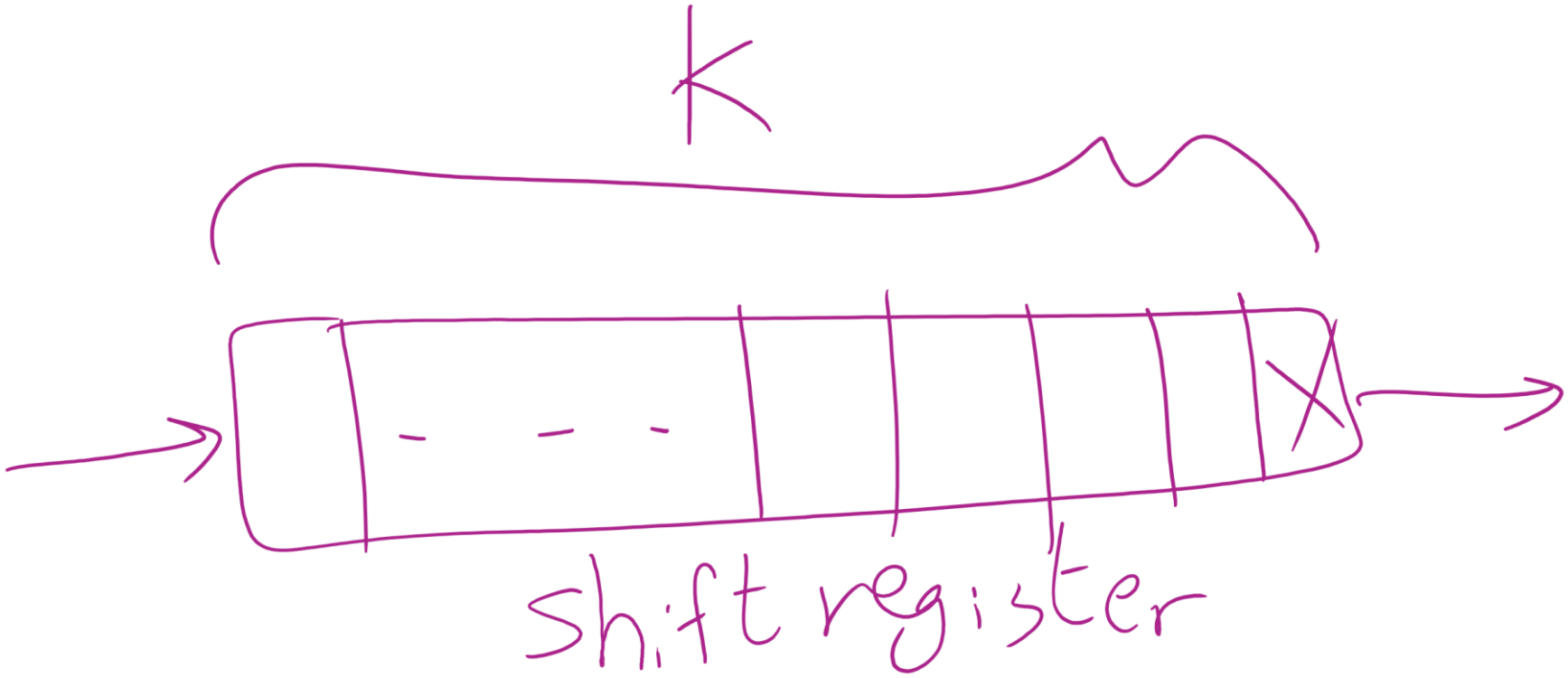
- *Demand paging*: bring a page into memory when it's requested by the process
- How many pages are needed?
  - Could be all of them, but not likely
  - Instead, processes reference a small set of pages at any given time—*locality of reference*
  - Set of pages can be different for different processes or even different times in the running of a single process
- Set of pages used by a process in a given interval of time is called the *working set*
  - If entire working set is in memory, no page faults!
  - If insufficient space for working set, thrashing may occur
  - Goal: keep most of working set in memory to minimize the number of page faults suffered by a process

How big is the working set?

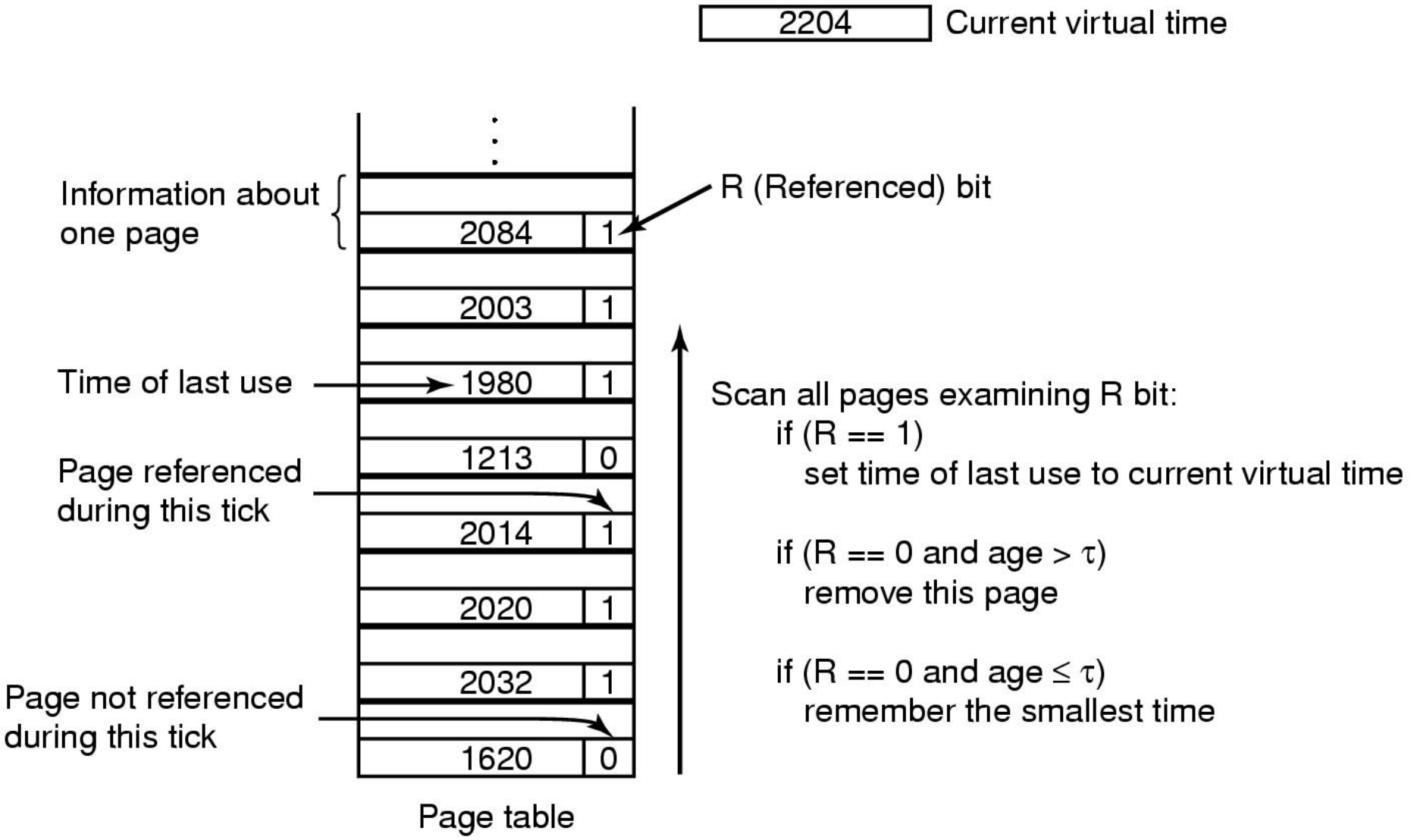


- Working set is the set of pages used by the *k* most recent memory references
- w(k,t) is the size of the working set at time *t*
- Working set may change over time
  - Size of working set can change over time as well…

Keeping track of the Working Set



Working set page replacement algorithm



Summary

- Page replacement algorithms

Algorithm	Comment
OPT (Optimal)	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Crude
FIFO (First-In, First Out)	Might throw out useful pages
Second chance	Big improvement over FIFO
Clock	Better implementation of second chance
LRU (Least Recently Used)	Excellent, but hard to implement exactly
NFU (Not Frequently Used)	Poor approximation to LRU
Aging	Good approximation to LRU, efficient to implement
Working Set	Somewhat expensive to implement
WSClock	Implementable version of Working Set

Algorithm Simulation

- How to simulate page replacement algorithms
  - FIFO/Clock
  - LRU, OPT

How is modeling done?

- Generate a list of references
  - Artificial (made up)
  - Trace a real workload (set of processes)
- Use an array (or other structure) to track the pages in physical memory at any given time
  - May keep other information per page to help simulate the algorithm (modification time, time when paged in, etc.)
- Run through references, applying the replacement algorithm
- Example: FIFO replacement on reference string 0 1 2 3 0 1 4 0 1 2 3 4
  - Page replacements highlighted in yellow

Page referenced	0	1	2	3	0	1	4	0	1	2	3	4
Youngest page	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
Oldest page			0	1	2	3	0	0	0	1	4	4

Interactive Simulation Tool

- https://sim-50.github.io/cs-tools/

FIFO with 3 frames

FIFO Example 1

FIFO with 4 frames

Belady's anomaly

- Reduce the number of page faults by supplying more memory
  - Use previous reference string and FIFO algorithm
  - Add another page to physical memory (total 4 pages)
- More page faults (10 vs. 9), not fewer!
  - This is called *Belady's anomaly*
  - Adding more pages shouldn't result in worse performance!
- Motivated the study of paging algorithms

CLOCK Simulation

Modeling more replacement algorithms

- Paging system characterized by:
  - Reference string of executing process
  - Page replacement algorithm
  - Number of page frames available in physical memory (*m*)
- Model this by keeping track of all *n* pages referenced in array *M*
  - Top part of *M* has *m* pages in memory
  - Bottom part of *M* has *n*-*m* pages stored on disk
- Page replacement occurs when page moves from top to bottom
  - Top and bottom parts may be rearranged without causing movement between memory and disk

Example: LRU

- Model LRU replacement with
  - 8 unique references in the reference string
  - 4 pages of physical memory
- Array state over time shown below
- LRU treats list of pages like a stack

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	5	5
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Stack algorithms

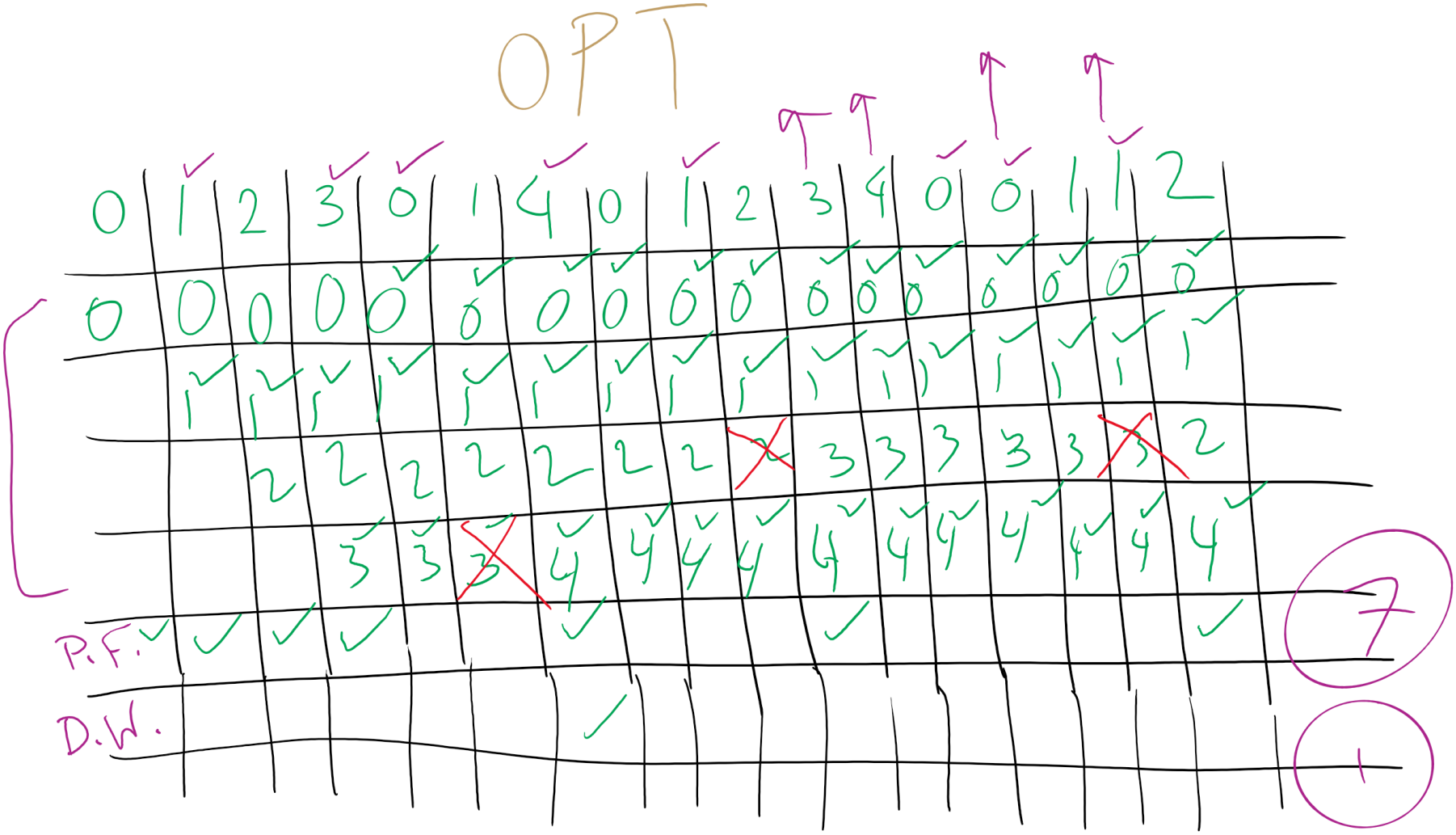
- LRU is an example of a stack algorithm
- For stack algorithms
  - Any page in memory with *m* physical pages is also in memory with *m*+1 physical pages
  - Increasing memory size is guaranteed to reduce (or at least not increase) the number of page faults
- Stack algorithms do not suffer from Belady's anomaly
- *Distance* of a reference == position of the page in the stack before the reference was made
  - Distance is ∞ if no reference had been made before
  - Distance depends on reference string and paging algorithm: might be different for LRU and optimal (both stack algorithms)

Predicting page fault rates using distance

- Distance can be used to predict page fault rates
- Make a single pass over the reference string to generate the distance string on-the-fly
- Keep an array of counts
  - Entry *j* counts the number of times distance *j* occurs in the distance string
- The number of page faults for a memory of size *m* is the sum of the counts for *j*>*m*
  - This can be done in a single pass!
  - Makes for fast simulations of page replacement algorithms
- This is why virtual memory theorists like stack algorithms!

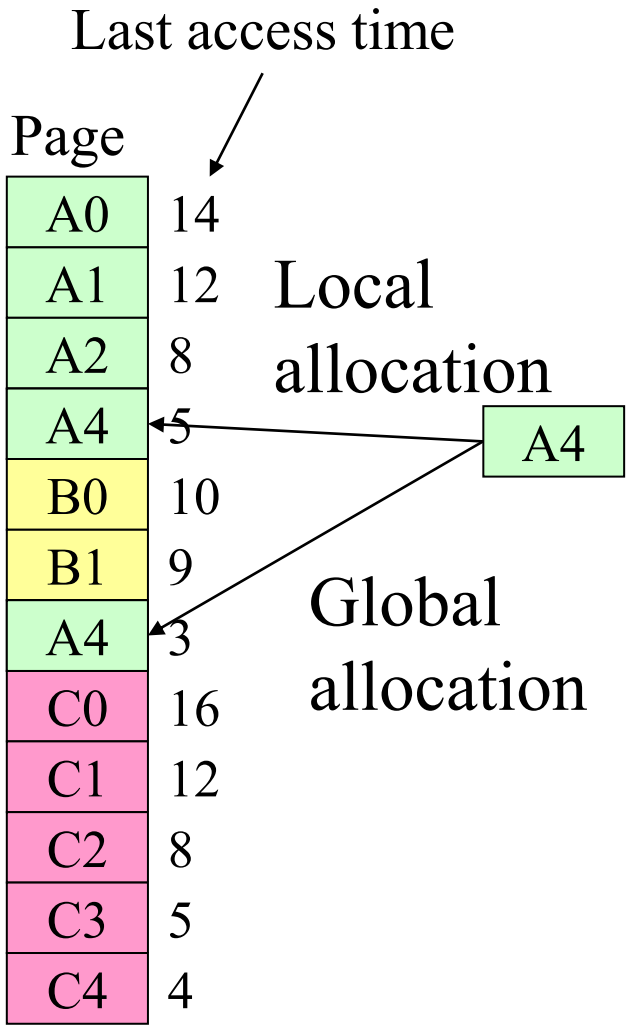
LRU

OPT



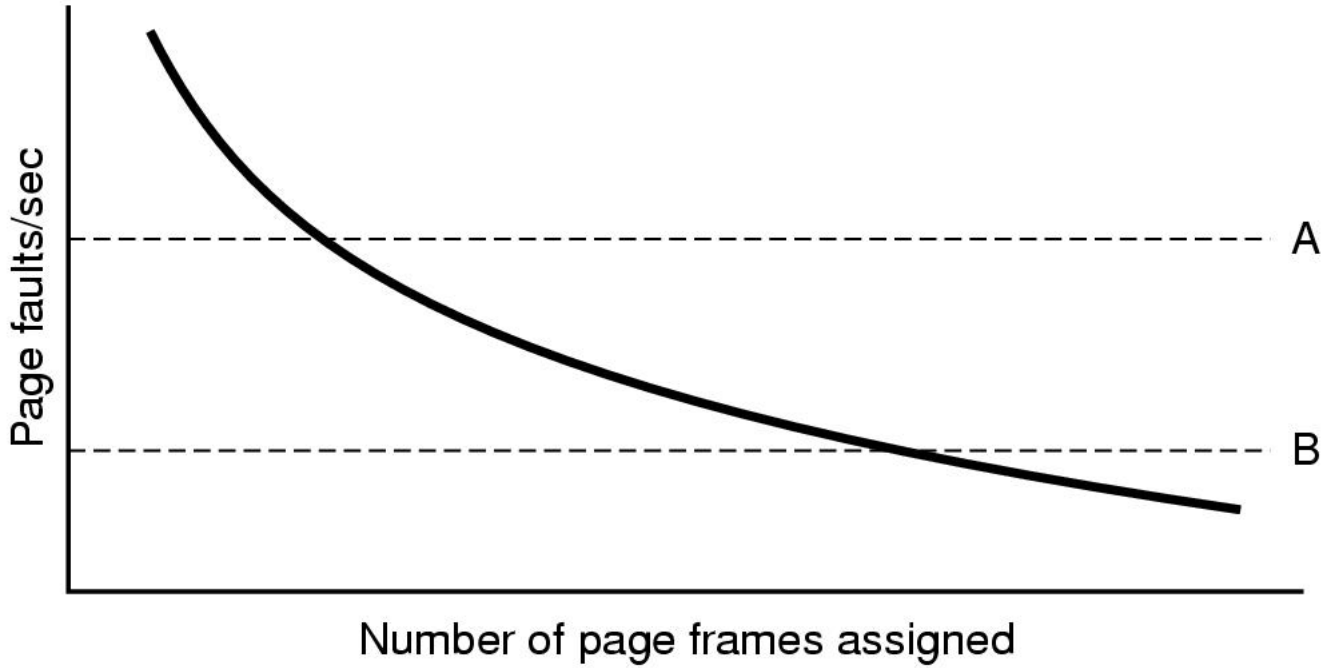
Local vs. global allocation policies

- What is the pool of pages eligible to be replaced?
  - Pages belonging to the process needing a new page
  - All pages in the system
- Local allocation: replace a page from this process
  - May be more “fair”: penalize processes that replace many pages
  - Can lead to poor performance: some processes need more pages than others
- Global allocation: replace a page from *any* process



Page fault rate vs. allocated frames

- Local allocation may be more “fair”
  - Don’t penalize other processes for high page fault rate
- Global allocation is better for overall system performance
  - Take page frames from processes that don’t need them as much
  - Reduce the overall page fault rate (even though rate for a single process may go up)



Control overall page fault rate

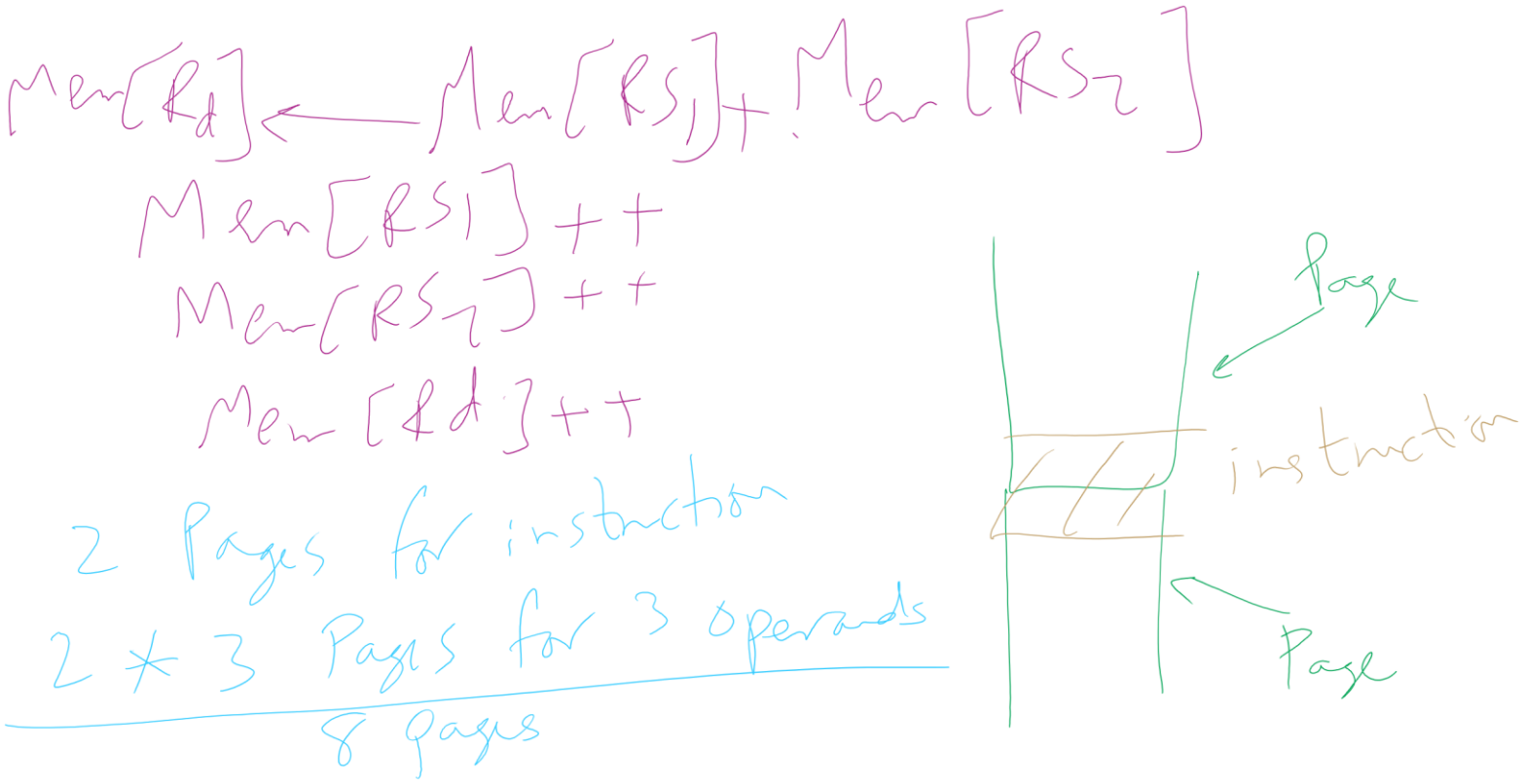
- Despite good designs, system may still thrash
- Most (or all) processes have high page fault rate
  - Some processes need more memory, …
  - but no processes need less memory (and could give some up)
- Problem: no way to reduce page fault rate
- Solution :
  Reduce number of processes competing for memory
  - Swap one or more to disk, divide up pages they held
  - Reconsider degree of multiprogramming

Backing up an instruction

- Problem: page fault happens in the middle of instruction execution
  - Some changes may have already happened
  - Others may be waiting for VM to be fixed
- Solution: undo all of the changes made by the instruction
  - Restart instruction from the beginning
  - This is easier on some architectures than others
- Example: LW R1, 12(R2)
  - Page fault in fetching instruction: nothing to undo
  - Page fault in getting value at 12(R2): restart instruction

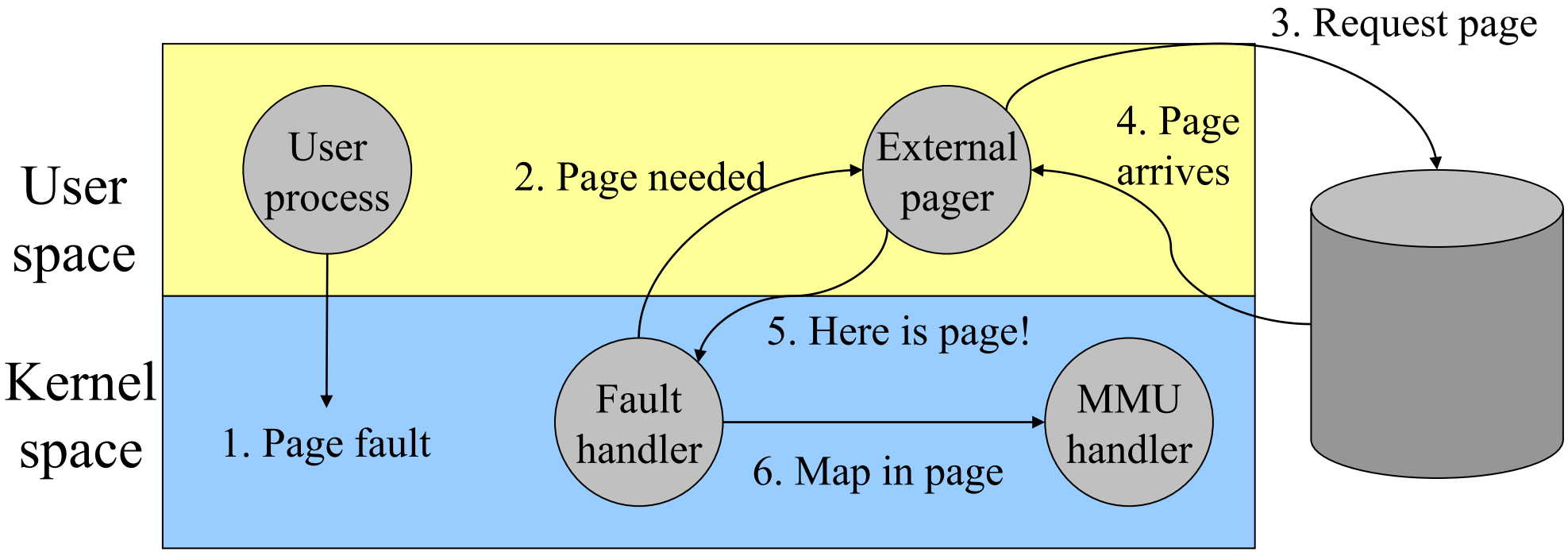
Minimum memory allocation to a process

- Example: ADD (Rd)+,(Rs1)+,(Rs2)+
  - Page fault in writing to (Rd): may have to undo an awful lot…

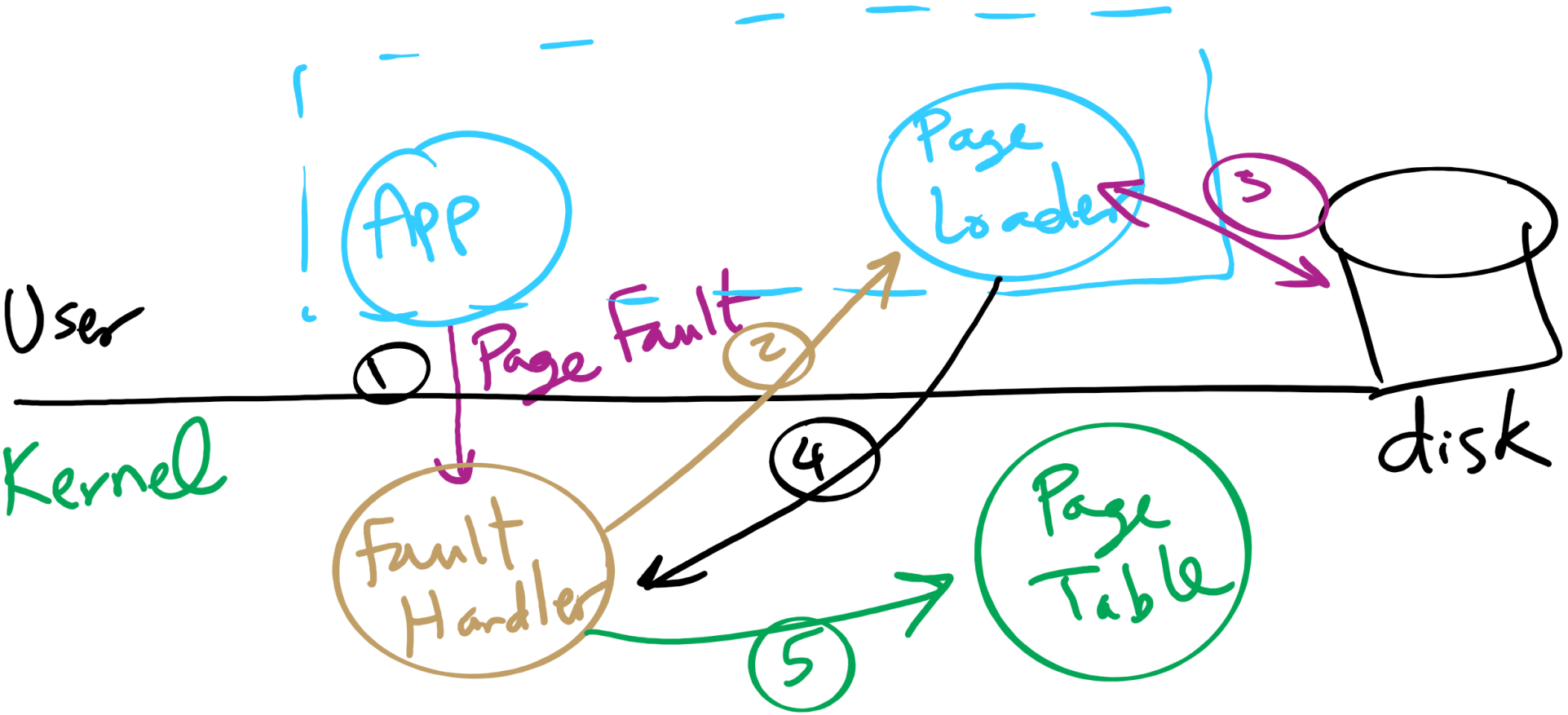


Separating policy and mechanism

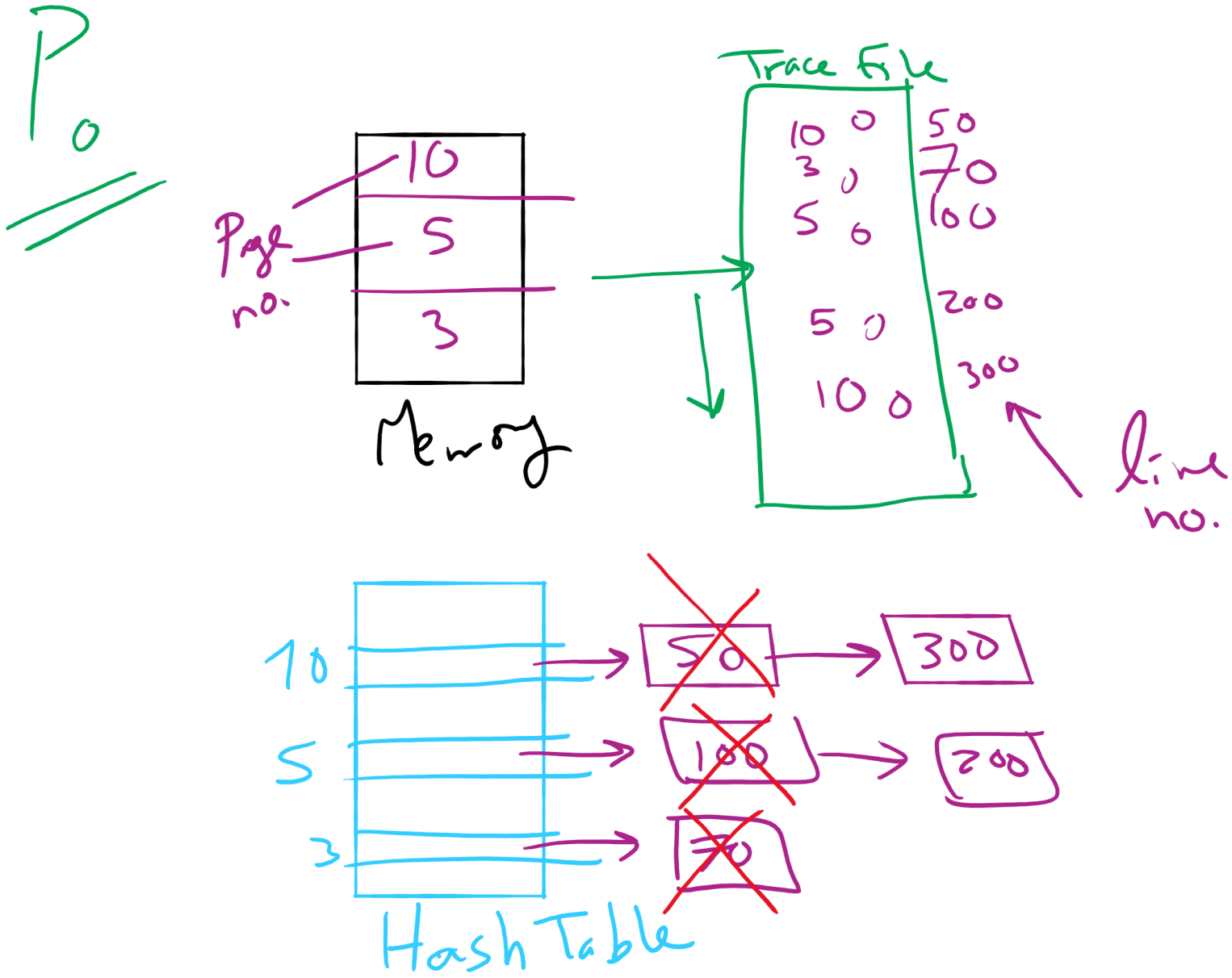
- Mechanism for page replacement has to be in kernel
  - Modifying page tables
  - Reading and writing page table entries
- Policy for deciding which pages to replace could be in user space
  - More flexibility



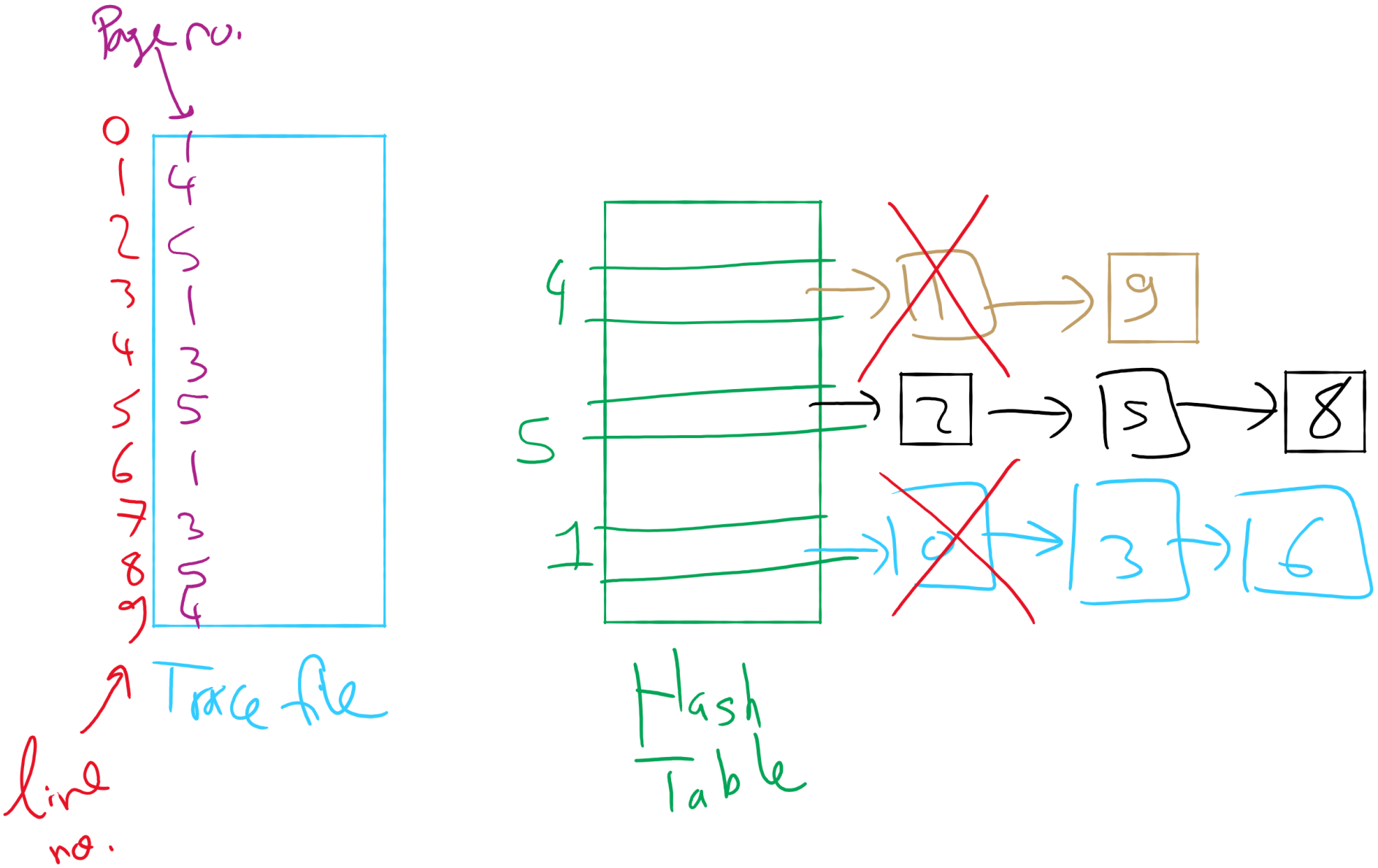
Separating Policy and Mechanism for Page Replacement



Project 3: OPT Simulation

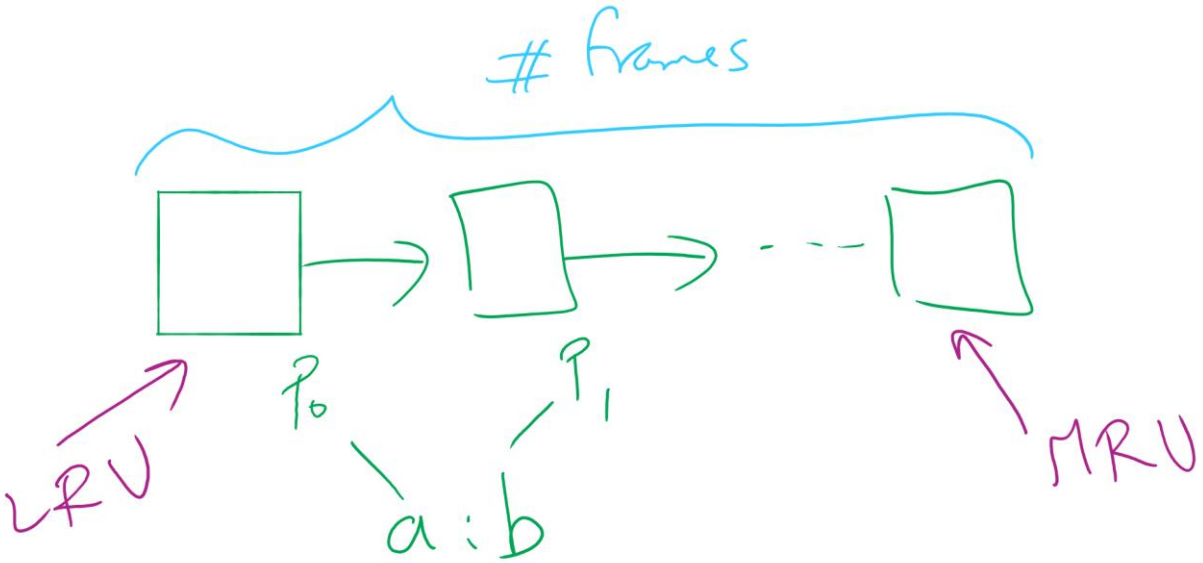


OPT Implementation Example



Project 3: LRU and miscellaneous hints

gunzip 1.trace.gz



$n \div (a+b) == 0$

$Frames_0 = \left(\frac{n}{a+b}\right) * a$

$Frames_1 = n - Frames_0$

Project 3: Writeup hints