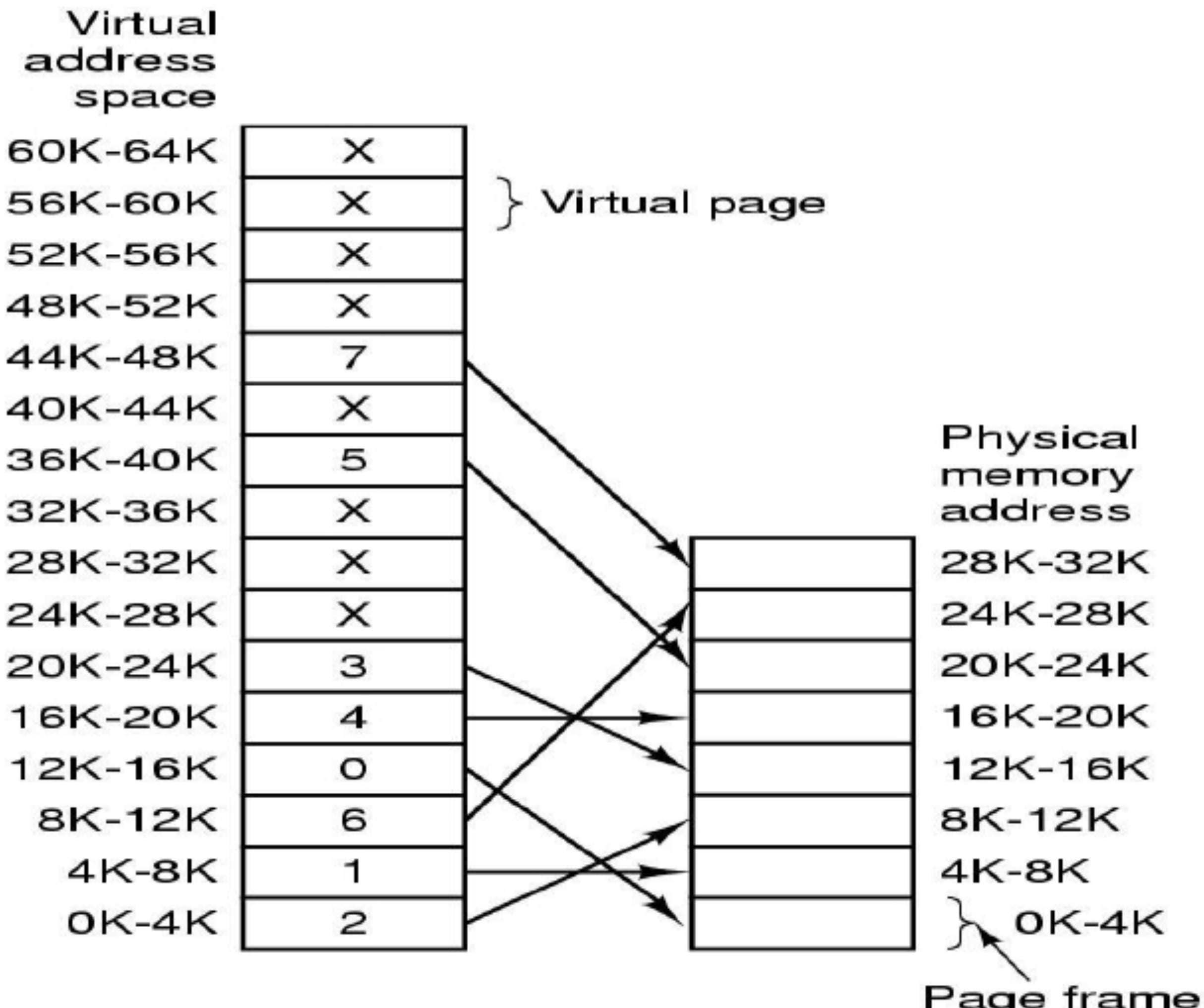
CS 492: Operating Systems

Page Replacement

Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

Virtual Addresses V.S. Physical Memory



16 pages

8 page frames

Paging and Swapping

- On heavily-loaded systems, memory can fill up
- To achieve good performance, must move "inactive" pages out to disk
- Two solutions:
  1. Swapping
     - Usually refers to moving the memory for an entire process out to disk
  2. Paging
     - Refers to moving individual pages out to disk (and back)

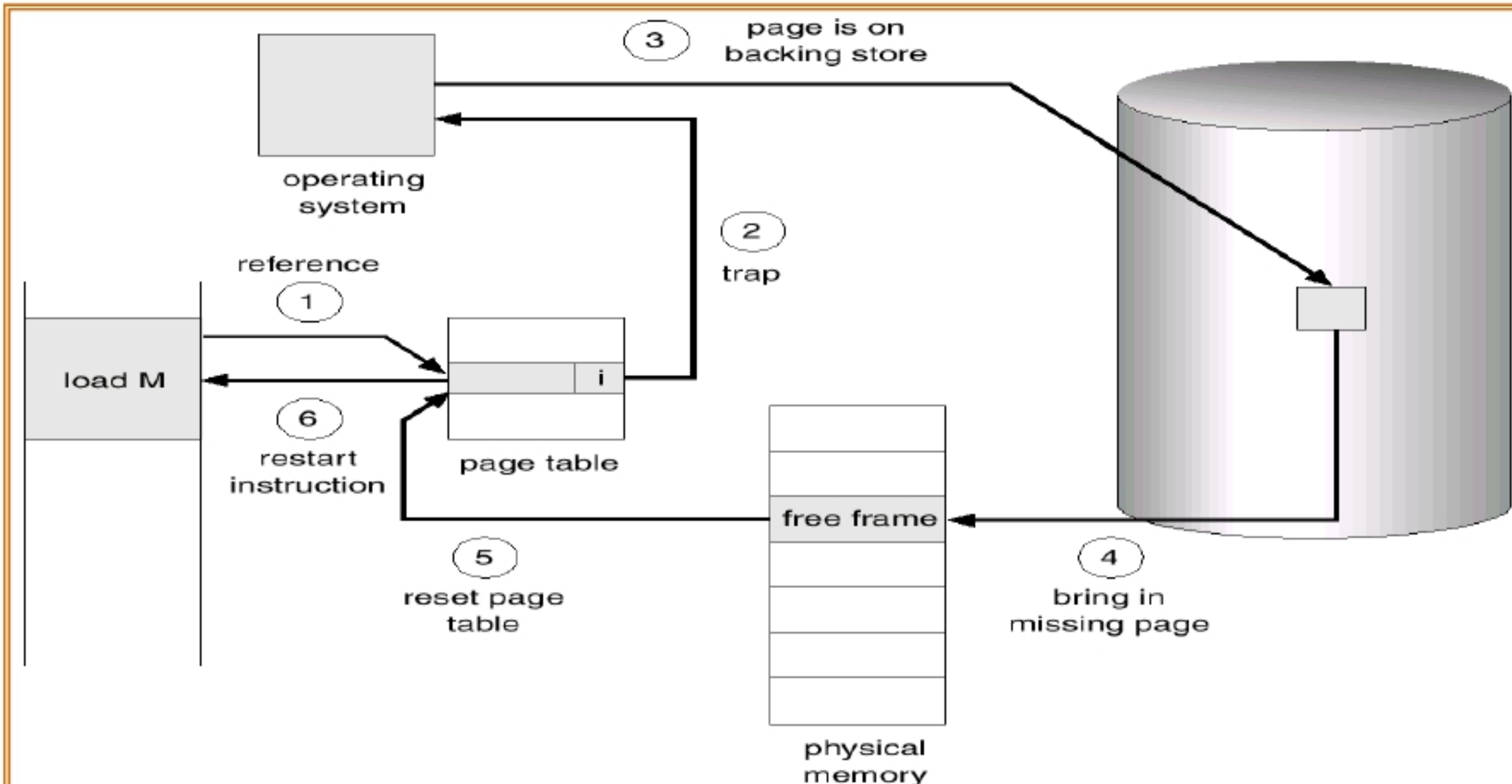
Outline

- How to move “inactive” pages out to disk to make space?

- Two solutions:
  1. Swapping
  2. Paging
     - Refers to moving individual pages out to disk (and back)

Page Fault

- If there is ever a reference to a page that is not in memory ⇒ **page fault**



Theoretical Foundation: Performance of Page Faults

- Page fault rate p: $0 \leq p \leq 1$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Effective Access Time (EAT)

$$
\begin{aligned}
\text{EAT} = (1 - p) \text{ x memory access} \\
+ p \text{ x (page fault overhead} \\
+ \text{swap page out overhead} \\
+ \text{swap page in overhead} \\
+ \text{restart overhead)}
\end{aligned}
$$

Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)

  = (1 – p x 200 + p x 8,000,000

  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault (i.e. p=1/1000), then

  EAT = 8.2 microseconds.

  This is a slowdown by a factor of 40 compared with memory access time (200 nanoseconds)!!!

Our goal: keep page fault as infrequent as possible!

What happens if there is no free frame in page table?

- Page replacement – find some page in memory and swap it out

- Page replacement forces choice
  - which page(s) must be removed

Page Replacement

- Question: which page to evict when memory is full and a new page is demanded?
- Goal: reduce the number of page faults

Page Replacement Algorithms

- Goal: **lowest** page-fault rate.
- *Input of algorithm*: a particular string of memory references (reference string)
  - Assume reference string in examples to follow is

    (page) 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- *Output of algorithm*: the number of page faults on that string.

Algorithm 1: Optimal (OPT) algorithm

- What's the best we can possibly do?
  - Assume OS knows the perfect knowledge of the future

- Algorithm: replace the page that will be used ***furthest*** in the future

- Estimate by …
  - logging page use on previous runs of process
  - although this is impractical

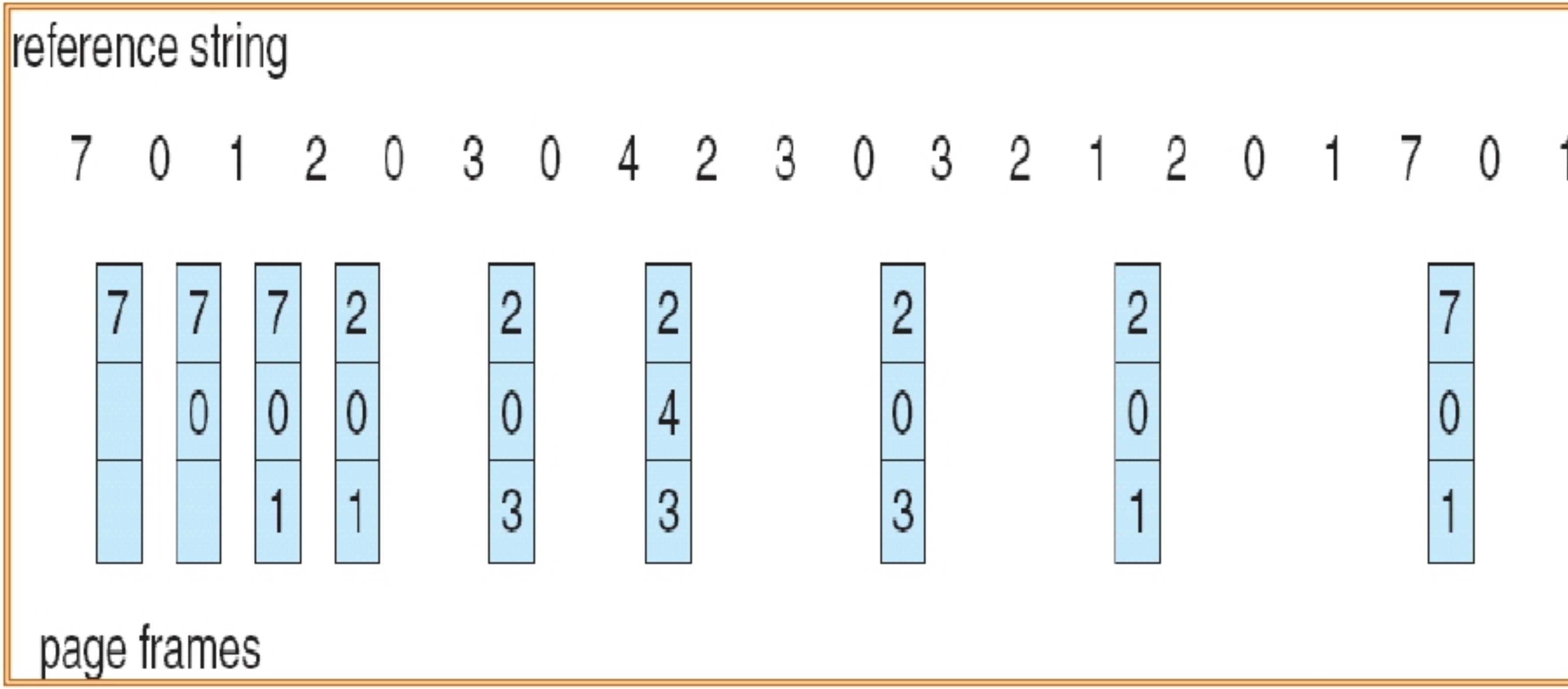
OPT Algorithm: Example

Assume there are three page frames in the memory. How many page faults will there be for the following memory reference string.?

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

OPT Algorithm: Example



9 page faults

Infeasibility of Optimal Algorithm

- Nice, but not achievable in real systems!
  - Only works if we know the whole sequence of page references!
  - Not realizable in practice (usually)

- However,
  - Can be approximated by running the program twice
    - Once to generate the reference trace
    - Once (or more) to apply the optimal algorithm