Virtual Memory

Daniel Bosk[1]

Department of Information and Communication Systems (ICS),
Mid Sweden University, Sundsvall.

vmem.tex 279 2018-11-29 08:26:00Z jimahl

Overview

1. Virtual Memory
   - Huge logical but small physical
2. Demand Paging
   - What is demand paging?
   - Demand paging
3. Page Replacement
   - Need for page replacement
   - Page-replacement algorithms

Literature

This lecture will give an overview of memory management. I.e. it gives an overview of Chapter 9 “Virtual Memory” in [SGG13a; SGG13b], or Chapter 9 “Virtual-Memory Management” in [SGG09].

Overview

Huge logical but small physical

- What if we want to run a program which is larger than the physical memory?
- When we run a program we don't necessarily use all of it, some parts of the code, e.g. error handling, isn't always used.
- Then why load the entire program into memory? Why not load only the pages of a program used by a process?
- This is the idea of virtual memory and demand paging.

Huge logical but small physical

- We've already separated the logical address space from the physical address space by paging and segmentation.
- Now we separate it further by virtual memory.
- Virtual memory means we pretend as if we have a huge memory, the virtual memory.

Huge logical but small physical

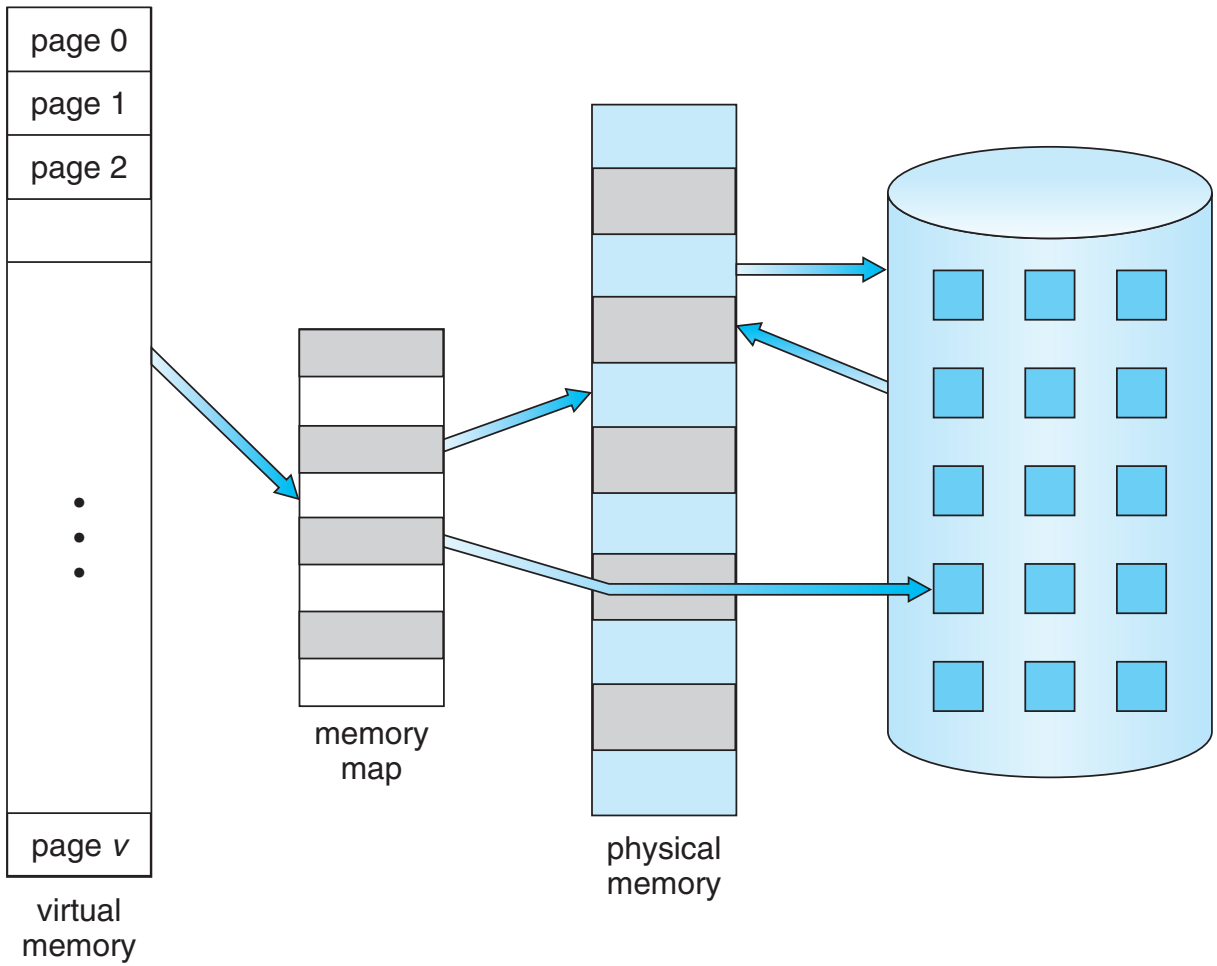

Figure: An illustration of the idea of a larger virtual memory than physical memory. Image: [SGG13b].

Huge logical but small physical

- Actually, with this approach *each* process can have a huge virtual memory, seeming to the process that it has all available memory to itself.

Huge logical but small physical

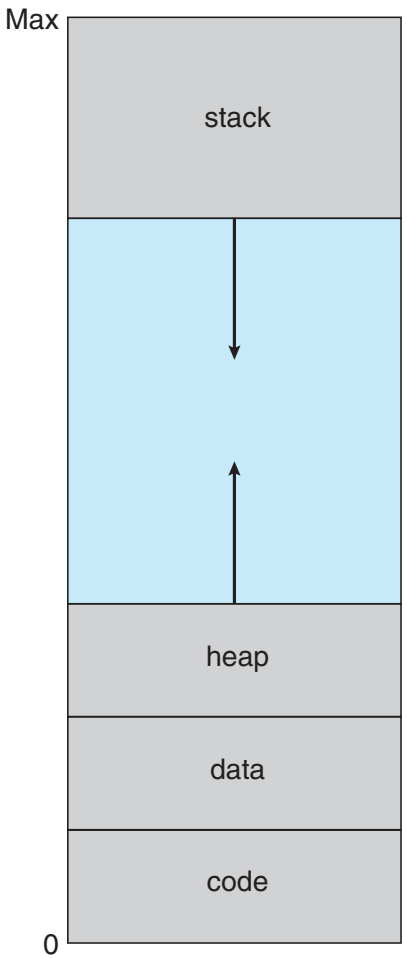

Figure: A process's virtual address space. Image: [SGG13b].

Huge logical but small physical

- With this huge gap in the middle we can fit a huge stack and an enormous heap.
- We can also use the space for other purposes.
- We can map in various shared pages there.

Huge logical but small physical

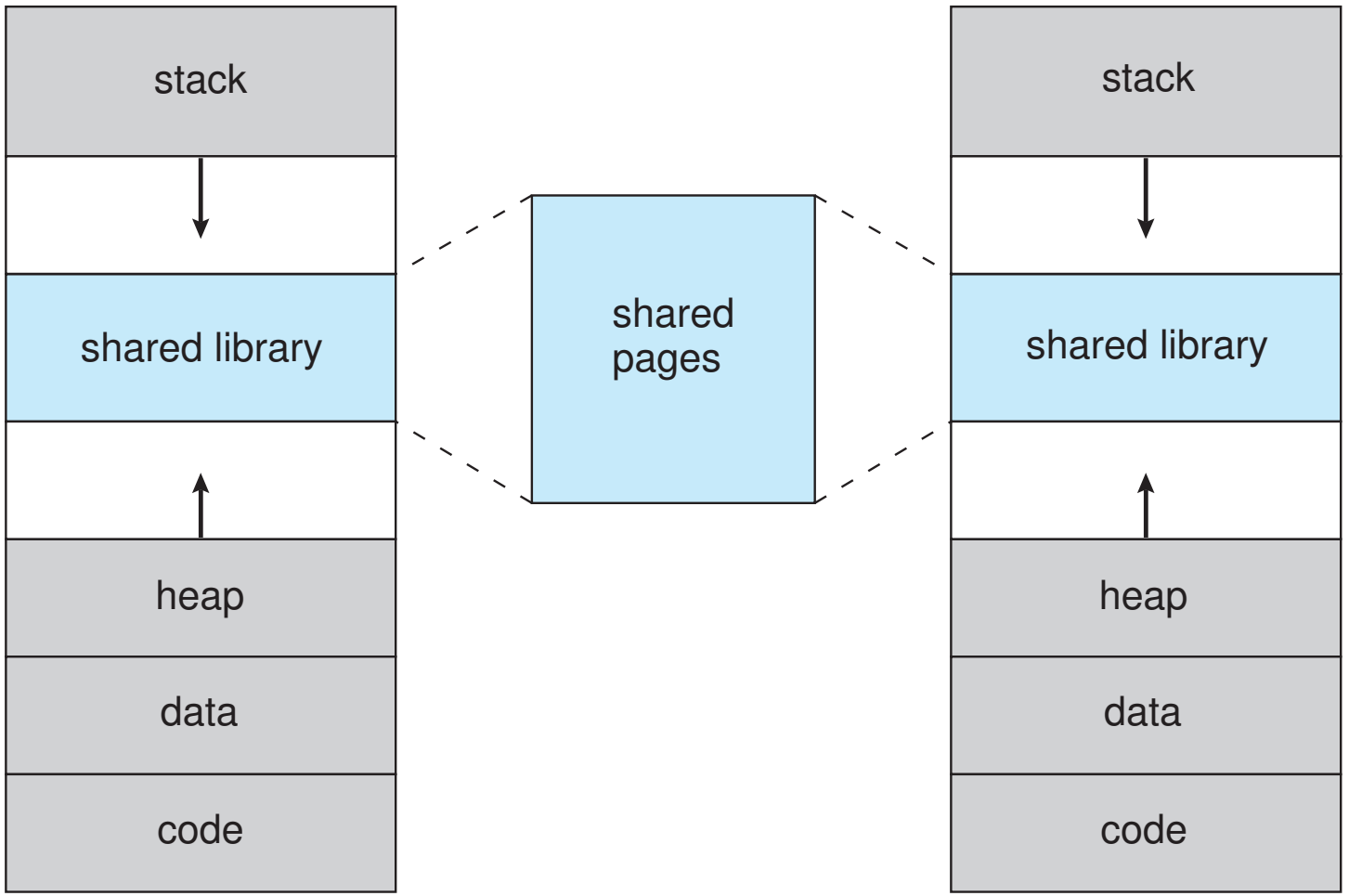

Figure: Shared library using virtual memory. Image: [SGG13b].

Overview

What is demand paging?

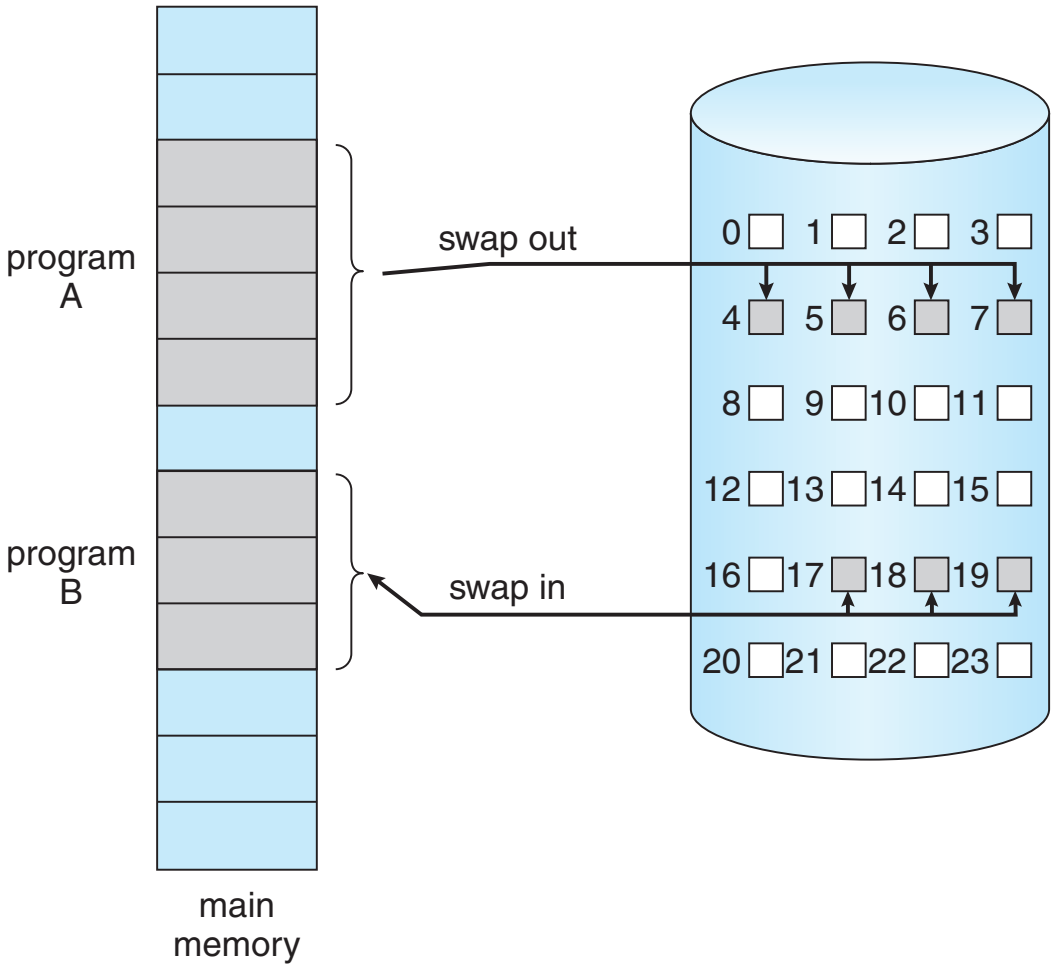

Figure: The principle of swapping as seen before. Image: [SGG13b].

What is demand paging?

- A swapper works with entire processes.
- What if we "swapped" in and out only parts of the programs?
- This is what a lazy swapper does.
- As we now see a process as a sequence of pages, we use a *pager* to page in and out pages of the process.
- For this kind of pager we need some form of hardware support to detect when a page is needed but not in memory, we use the valid–invalid bits from before.

Demand paging

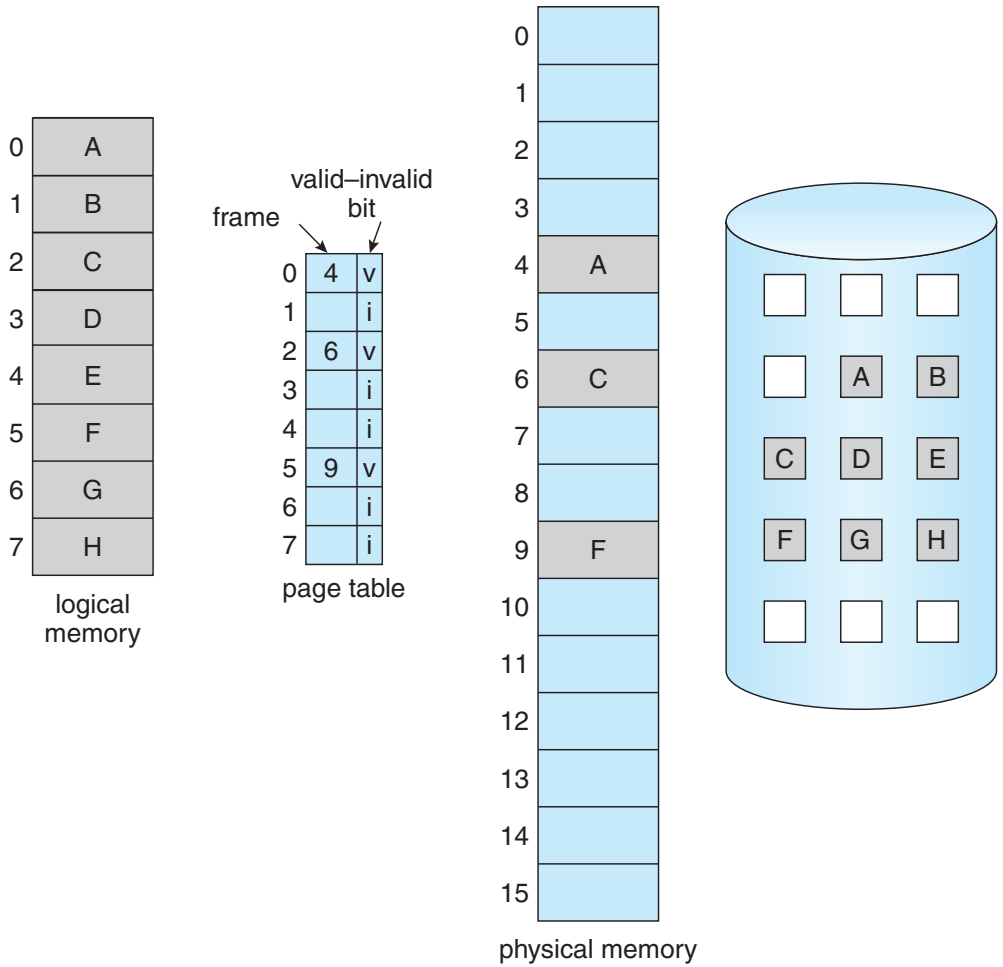

Figure: Page table with some pages in memory and some on disk. Image: [SGG13b].

Demand paging

- As long as pages are allocated to frames and are in memory everything works as before.
- What happens when the process refers to a page not in memory?
- Then the memory unit traps to the OS, it generates a page fault.

Demand paging

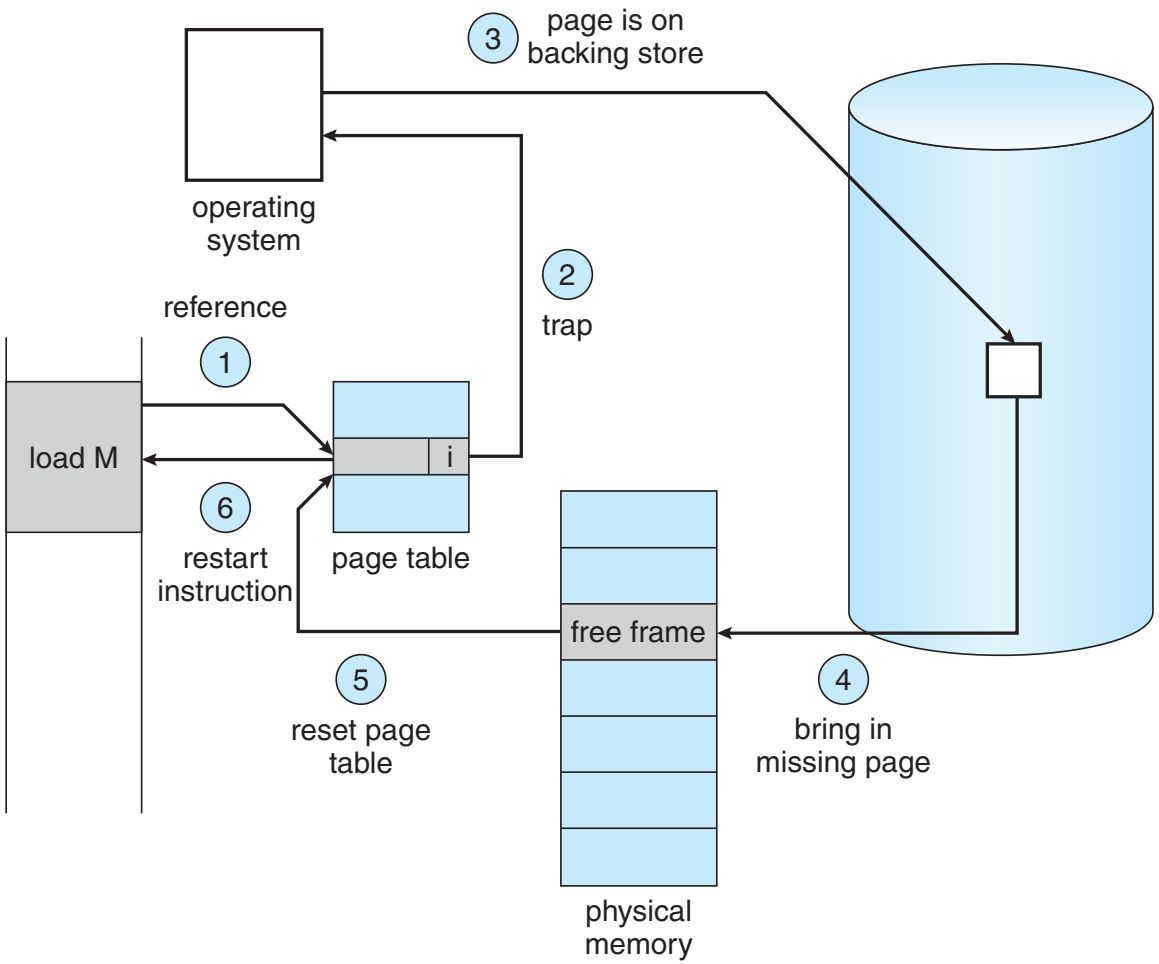

Figure: Procedure when a page fault occurs. Image: [SGG13b].

Demand paging

- Pure demand paging means we never bring a page into memory until it's needed (by generating a page fault).
- The hardware requirements for demand paging are mostly the same as for paging (with valid–invalid bits) and swapping.
- However, it also needs the ability to restart any instruction in the CPU after a page fault.

Demand paging

- How is the performance of demand paging?
- Usually the memory access time is around 10 to 200 nanoseconds.
- With no page faults we keep the same performance as usual.
- We compute the effective access time as we did before for the TLB.

Demand paging

Effective access time

The effective access time t is

$$t = (1 - p) \times t_m + p \times t_p,$$

where p is the probability of a page fault, t_m is the memory access time and $t_p > t_m$ is the page-fault time.

Demand paging

Example

We have $t_m = 200\ \text{ns}$ and $t_p = 8\ \text{ms} = 8000000\ \text{ns}$. We want $t < 220\ \text{ns}$, i.e. less than 10 percent loss.

$$
\begin{aligned}
t &> (1-p) \times t_m + p \times t_p && (1)\\
220 &> (1-p) \times 200 + p \times 8000000 && (2)\\
220 &> 200 - p \times 200 + p \times 8000000 && (3)\\
220 &> 200 + p \times (8000000 - 200) && (4)\\
220 &> 200 + 7999800 \times p && (5)\\
20 &> 7999800 \times p && (6)\\
\frac{20}{7999800} \approx 0.0000025 &> p && (7)
\end{aligned}
$$

Overview

Need for page replacement

- Need to select a victim frame.
- Different page-replacement algorithms for this.
- Some pages are more expensive to replace.
- If they're not changed we can use the copy on disk already.
- If they're changed we need to write the new version to disk again.

Need for page replacement

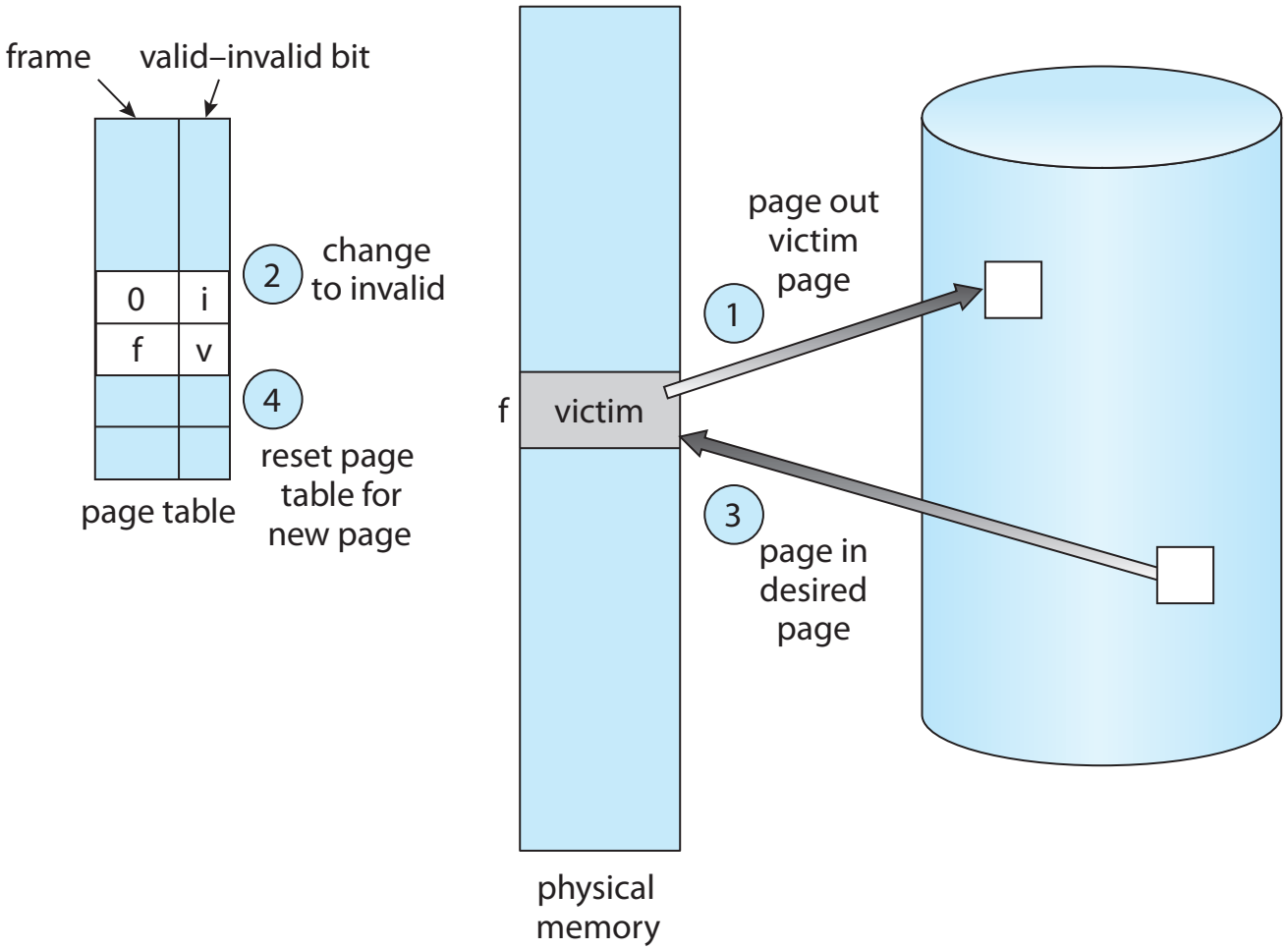

Figure: A page replacement happening. Image: [SGG13b].

Page-replacement algorithms

- FIFO page replacement (first in, first out)
- Optimal page replacment
- LRU page replacement (least recently used)
- Second chance
- Enhanced second chance

Page-replacement algorithms

- We can use a page-reference string to test an algorithm through a simulator.
- For some of these algorithms there is a simulator which is included in the lab (only non-campus students).
- The simulator available is written in Python.
- For campus students the simulator is available for testing purposes.

Referenser I

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. 8th ed. International Student Version. Hoboken, N.J.: John Wiley & Sons Inc, 2009.

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. 9th ed. International Student Version. Hoboken, N.J.: John Wiley & Sons Inc, 2013.

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. 9th ed. Hoboken, N.J.: John Wiley & Sons Inc, 2013.

Mittuniversitetet
MID SWEDEN UNIVERSITY