CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Copy on Write

When parent forks child process, child gets copy of entire address space of parent

This is simple and convenient, but inefficient

Particularly wasteful if child is going to exec immediately

With paging, memory "copying" part of fork can be done extremely quickly

Page table of child is copy of page table of parent

All accesses from either process point to same frame, and so to same place in memory

Downside to this?

These are *processes*, not *threads*

They should not actually share memory

Changes to memory by one process should not affect memory of other

To get around this, OS marks every entry of page table in both processes as **copy-on-write**

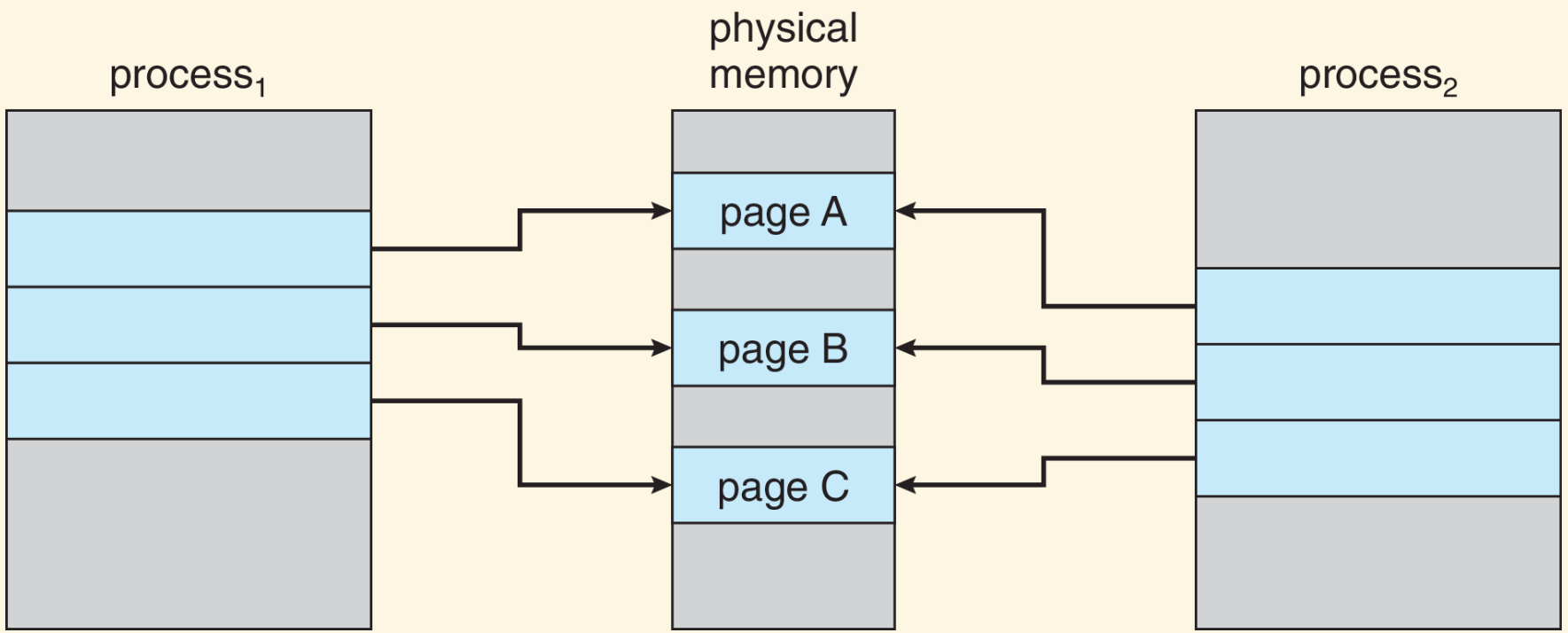
No need for wasteful copy of everything

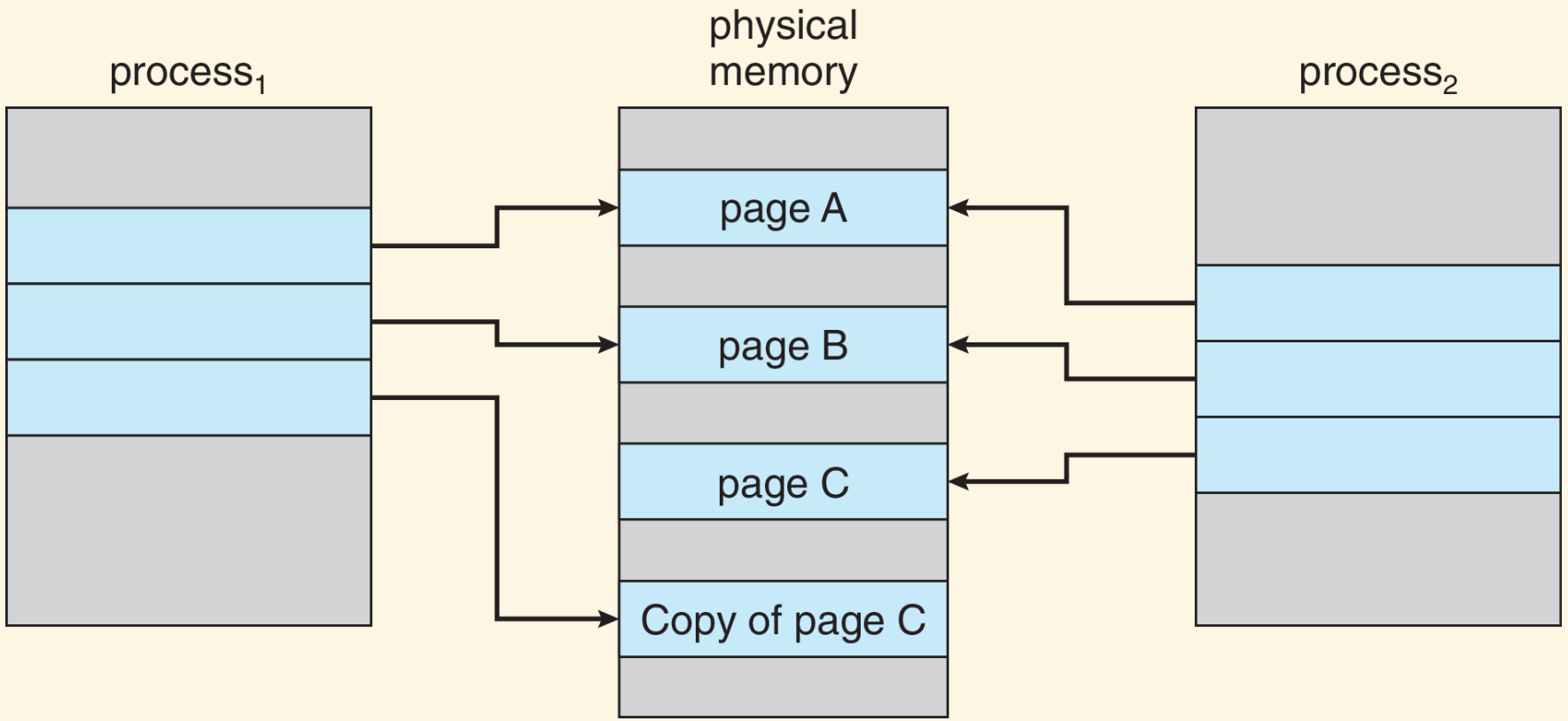
However, when memory is written to (and therefore may differ between processes), following procedure is enacted

Copy-on-write

1. New frame is allocated to writing process
2. Memory from "shared" frame copied into new frame
3. Page table for writing process updated and memory written

Note that this applies only to frame that is overwritten -- all other frames still "shared"


process1
physical memory
process2
page A
page B
page C


process1
physical memory
process2
page A
page B
page C
Copy of page C

Much more efficient than naive copy because

- No pages copied unless absolutely necessary
- Only affected pages are copied, not entire memory

OS keeps pool of free frames around to satisfy copy-on-write requests and other instances of process memory growing or changing

However, OS must ensure privacy of process memory even after process terminates

Frames in pool still hold data from previous processes

Frame allocation uses **zero-fill-on-demand**

Whenever frame is allocated to process, memory is filled with zeros to avoid leaking information

"On demand" because, similar to copy-on-write, zero-fill is performed only when actually needed -- when frame is allocated