VIRTUAL MEMORY: CONCEPTS

CS 045

Computer Organization and Architecture

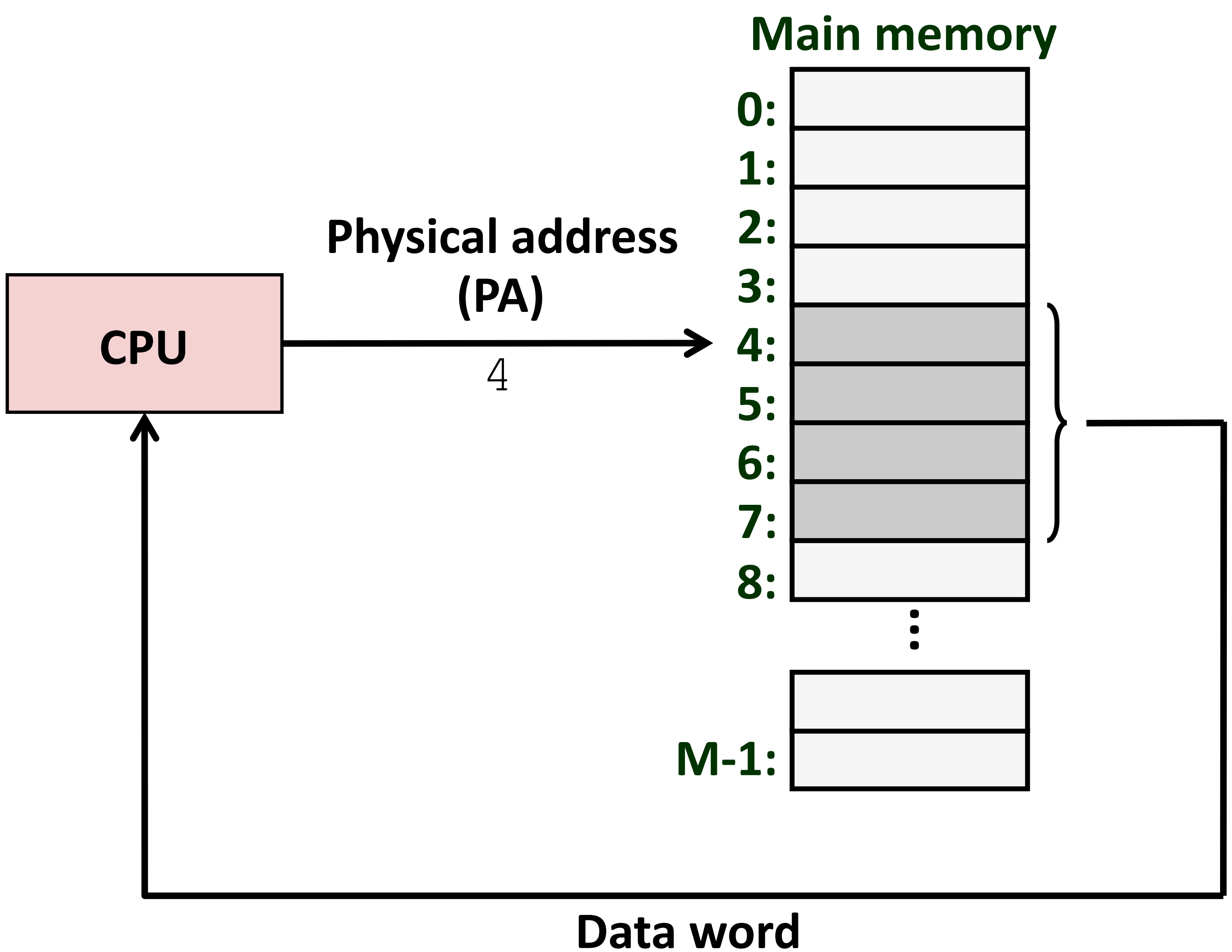
Prof. Donald J. Patterson

Adapted from Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

VIRTUAL MEMORY: CONCEPTS

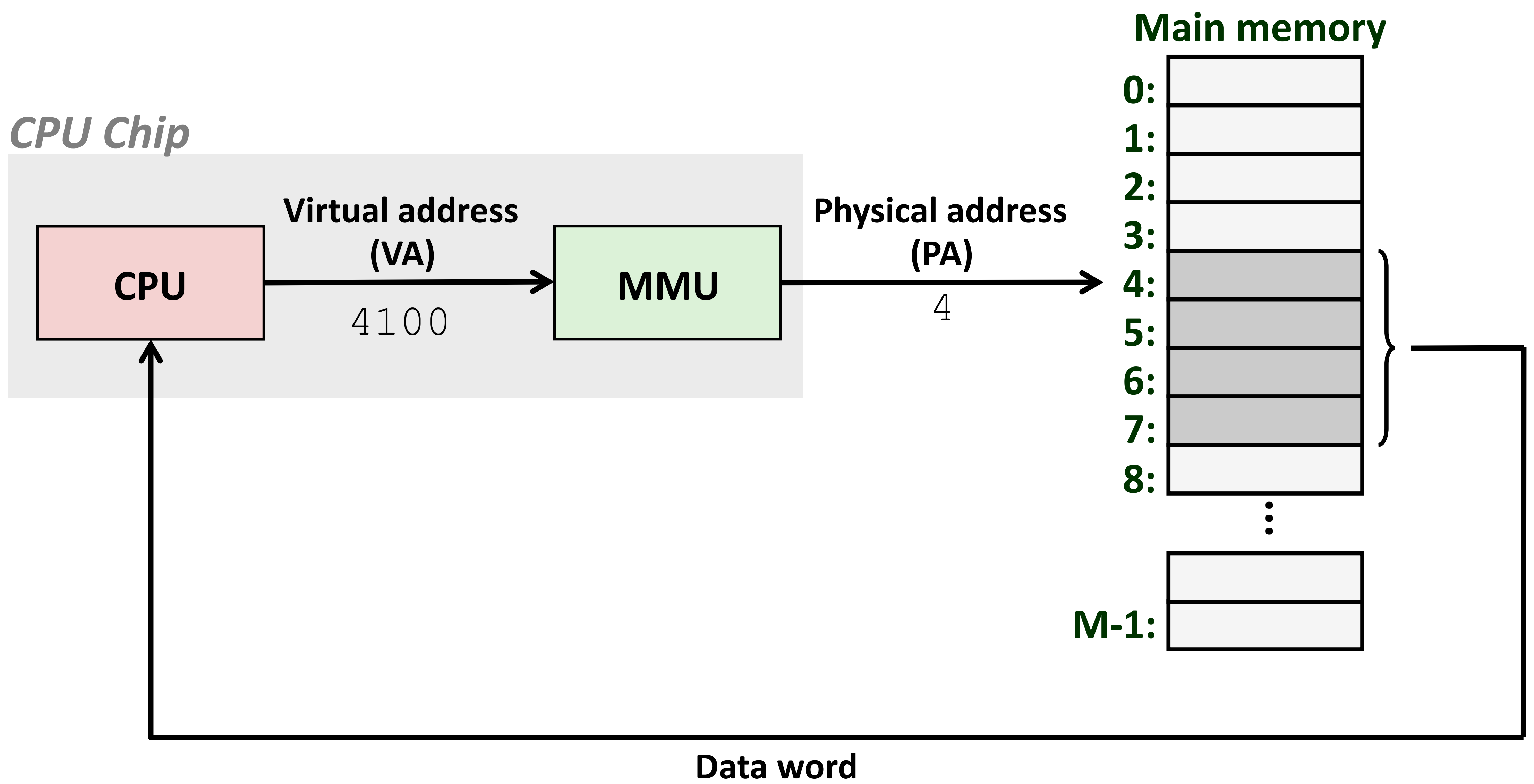
- ADDRESS SPACES
- VM AS A TOOL FOR CACHING
- VM AS A TOOL FOR MEMORY MANAGEMENT
- VM AS A TOOL FOR MEMORY PROTECTION
- ADDRESS TRANSLATION

A SYSTEM USING PHYSICAL ADDRESSING



- **Used in "simple" systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames**

A SYSTEM USING VIRTUAL ADDRESSING



- Used in all modern servers, laptops, and smart phones
- One of the great ideas in computer science

WHY VIRTUAL MEMORY (VM)?

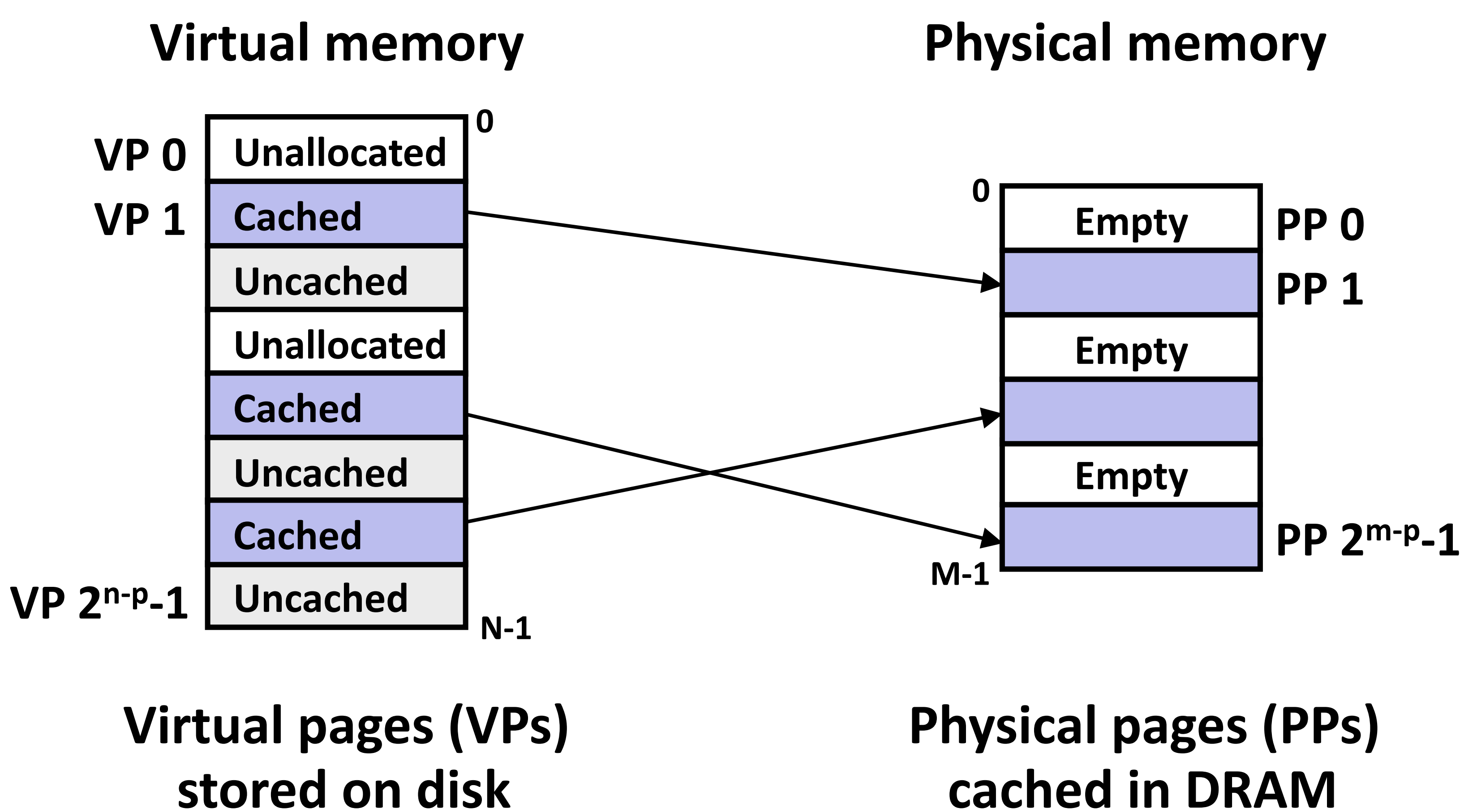
- **Uses main memory efficiently**
  - Use DRAM as a cache for parts of a virtual address space
- **Simplifies memory management**
  - Each process gets the same uniform linear address space
- **Isolates address spaces**
  - One process can't interfere with another's memory
  - User program cannot access privileged kernel information and code

VIRTUAL MEMORY: CONCEPTS

- ADDRESS SPACES
- VM AS A TOOL FOR CACHING
- VM AS A TOOL FOR MEMORY MANAGEMENT
- VM AS A TOOL FOR MEMORY PROTECTION
- ADDRESS TRANSLATION

VM AS A TOOL FOR CACHING

- **Conceptually, *virtual memory* is an array of N contiguous bytes stored on disk.**
- **The contents of the array on disk are cached in *physical memory* (*DRAM cache*)**
  - These cache blocks are called *pages* (size is $P = 2^p$ bytes)

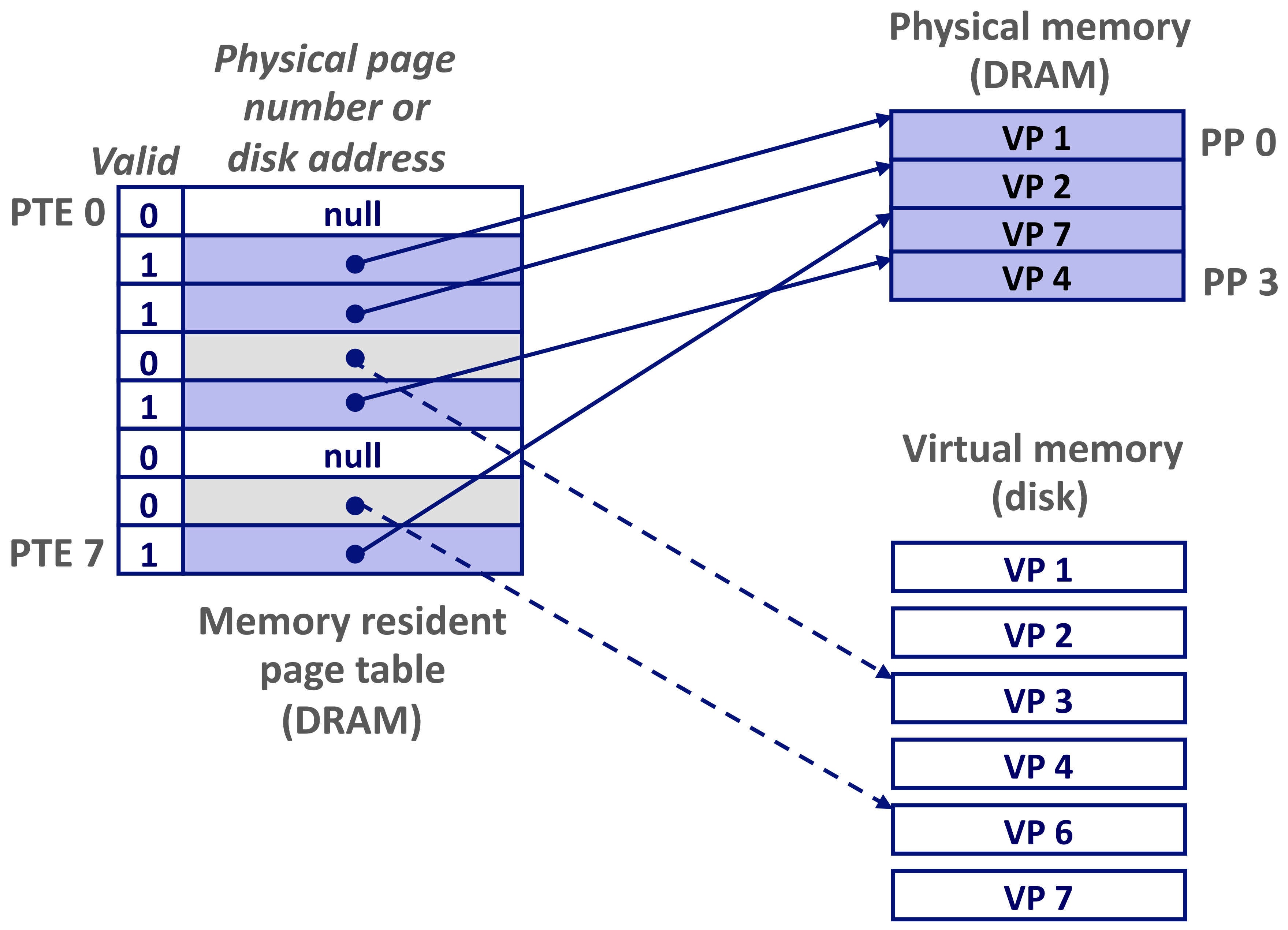


DRAM CACHE ORGANIZATION

- **DRAM cache organization driven by the enormous miss penalty**
  - DRAM is about 10x slower than SRAM
  - Disk is about 10,000x slower than DRAM
- **Consequences**
  - Large page (block) size:
    - typically 4 KB, sometimes 4 MB
  - Fully associative
    - Any VP can be placed in any PP
    - Requires a "large" mapping function – different from cache memories
  - Highly sophisticated, expensive replacement algorithms
    - Too complicated and open-ended to be implemented in hardware
  - Write-back rather than write-through

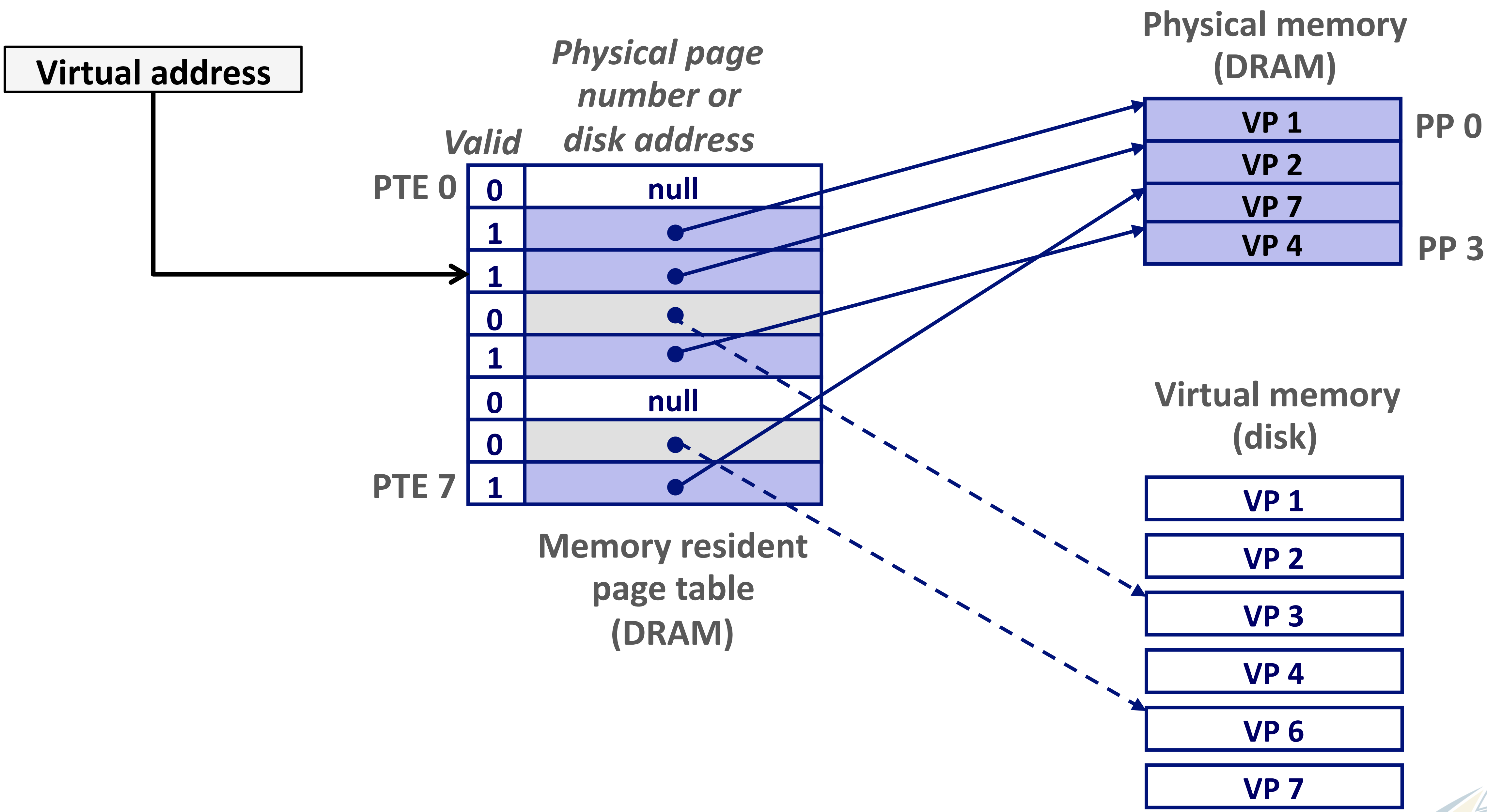
ENABLING DATA STRUCTURE: PAGE TABLE

- **A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages.**
  - Per-process kernel data structure in DRAM



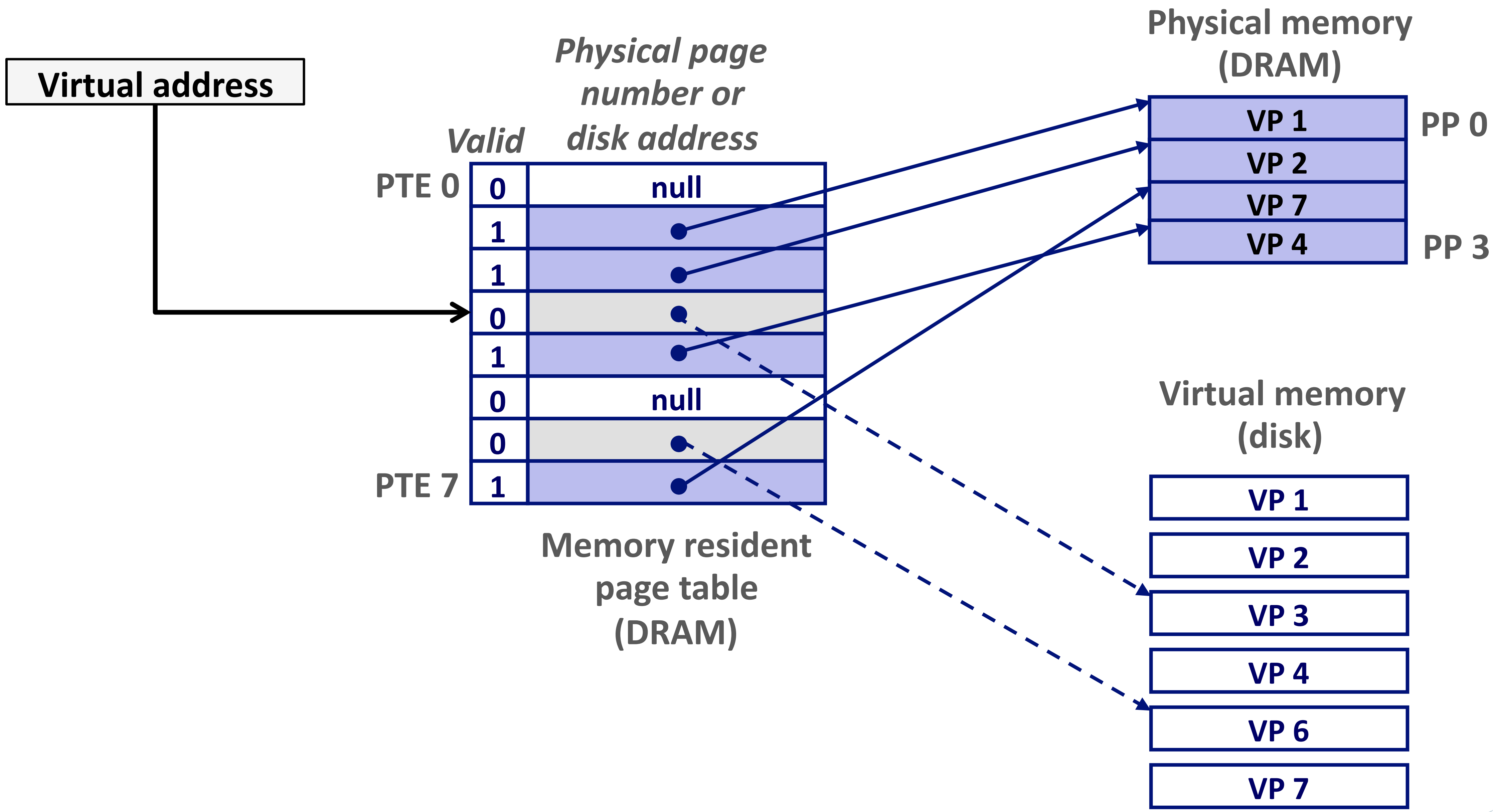
PAGE HIT

- ***Page hit:*** **reference to VM word that is in physical memory (DRAM cache hit)**



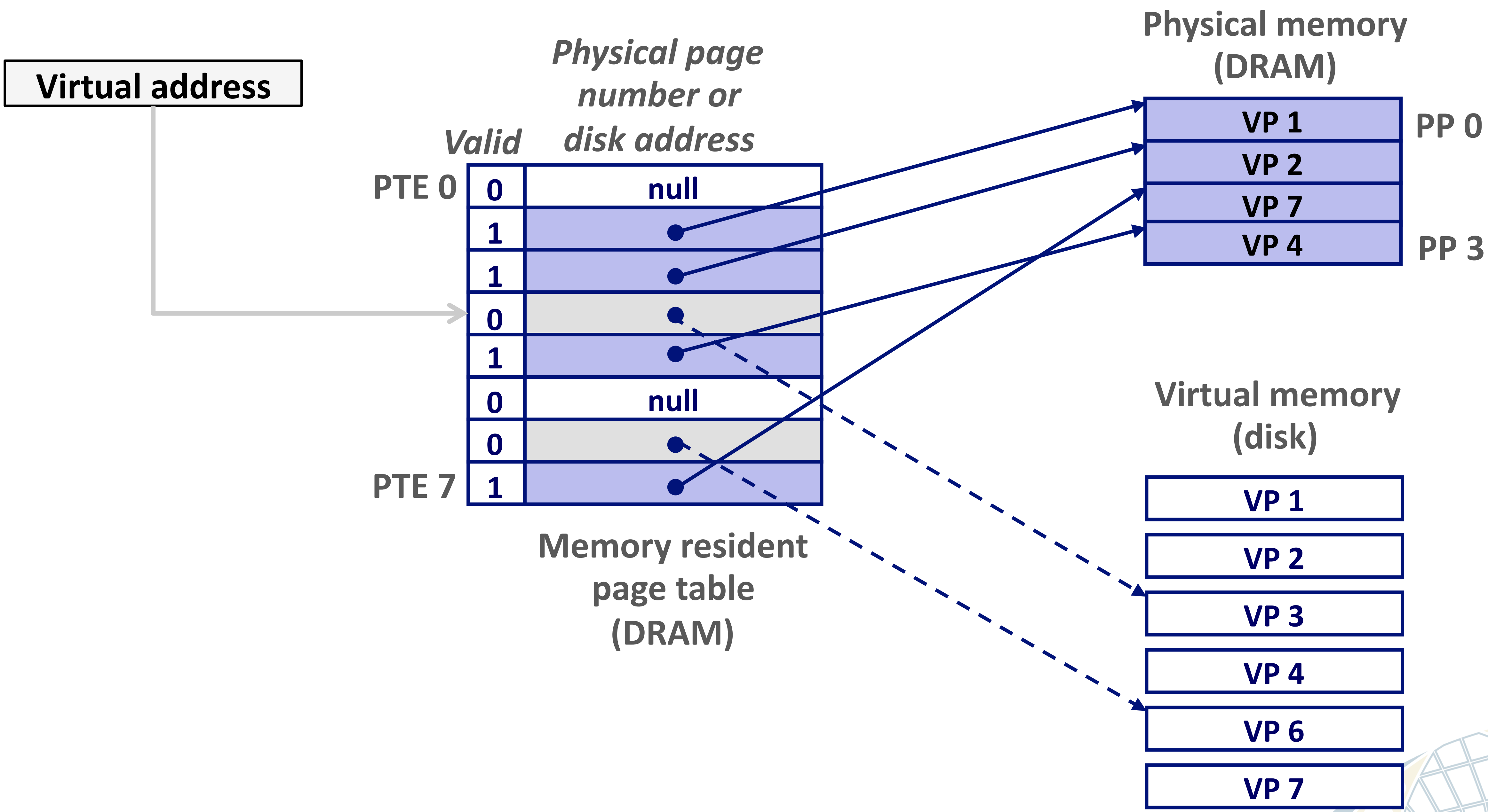
PAGE FAULT

- ***Page fault:*** **reference to VM word that is not in physical memory (DRAM cache miss)**



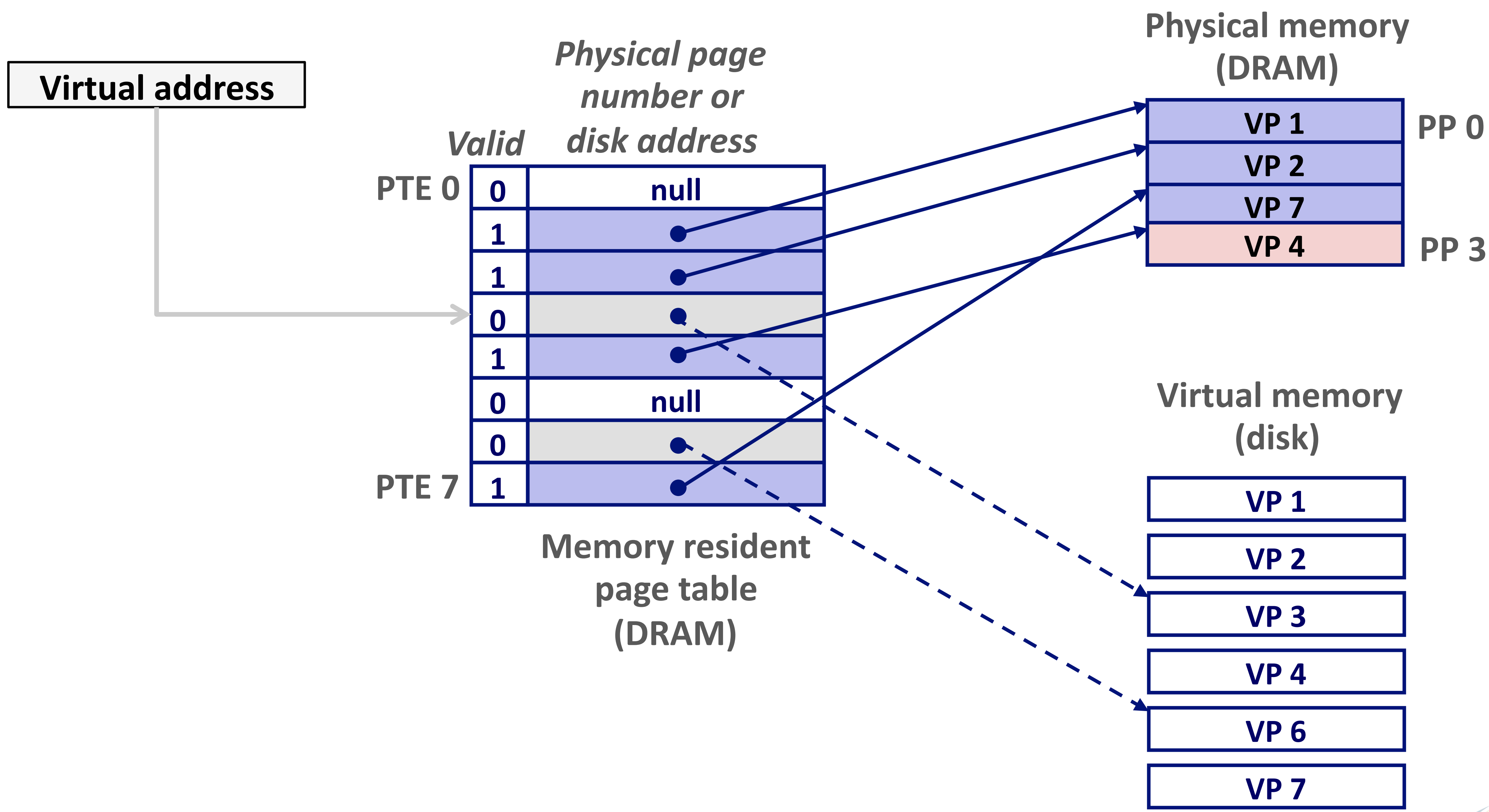
HANDLING PAGE FAULT

- Page miss causes page fault (an exception)



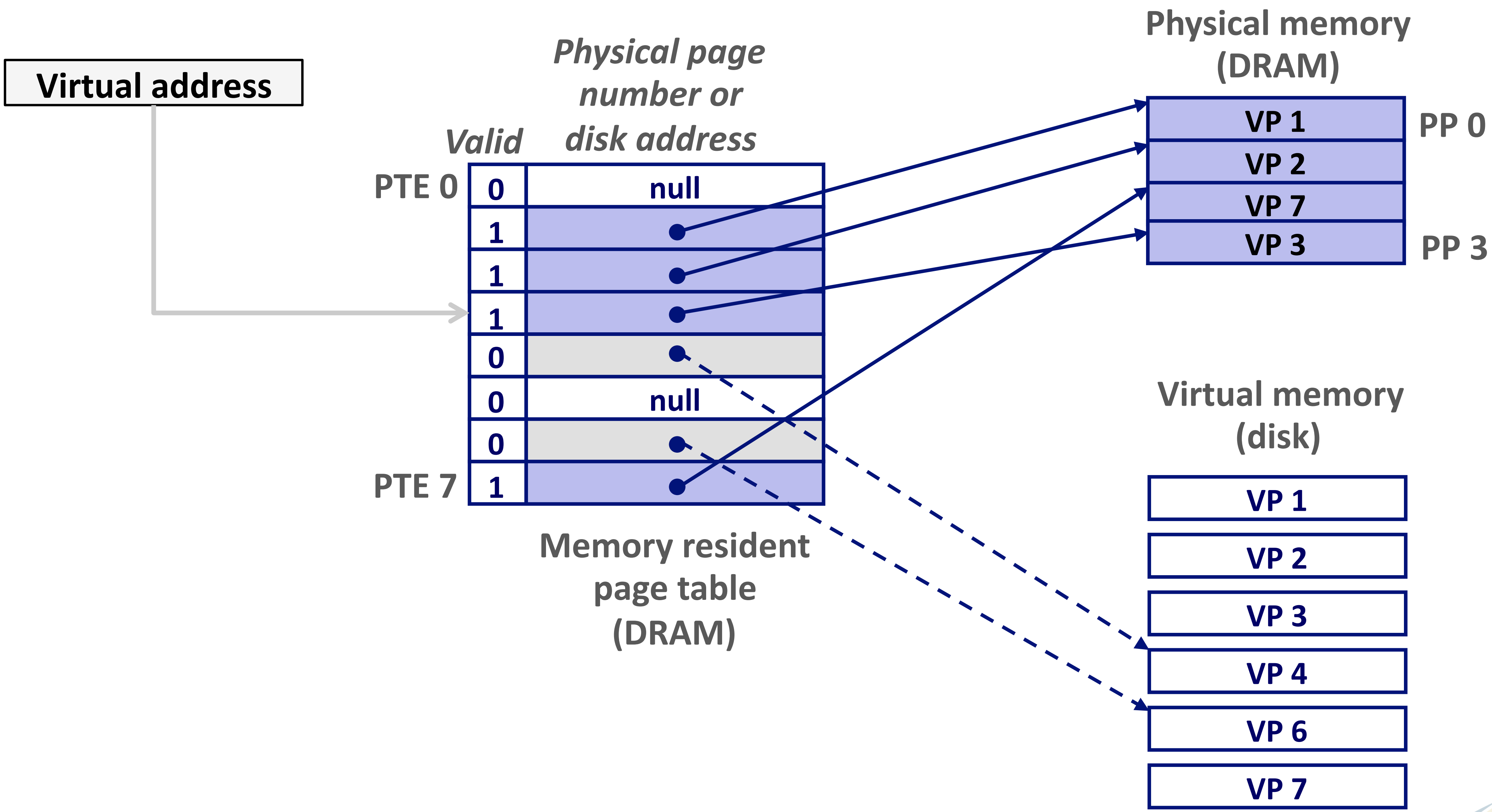
HANDLING PAGE FAULT

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)



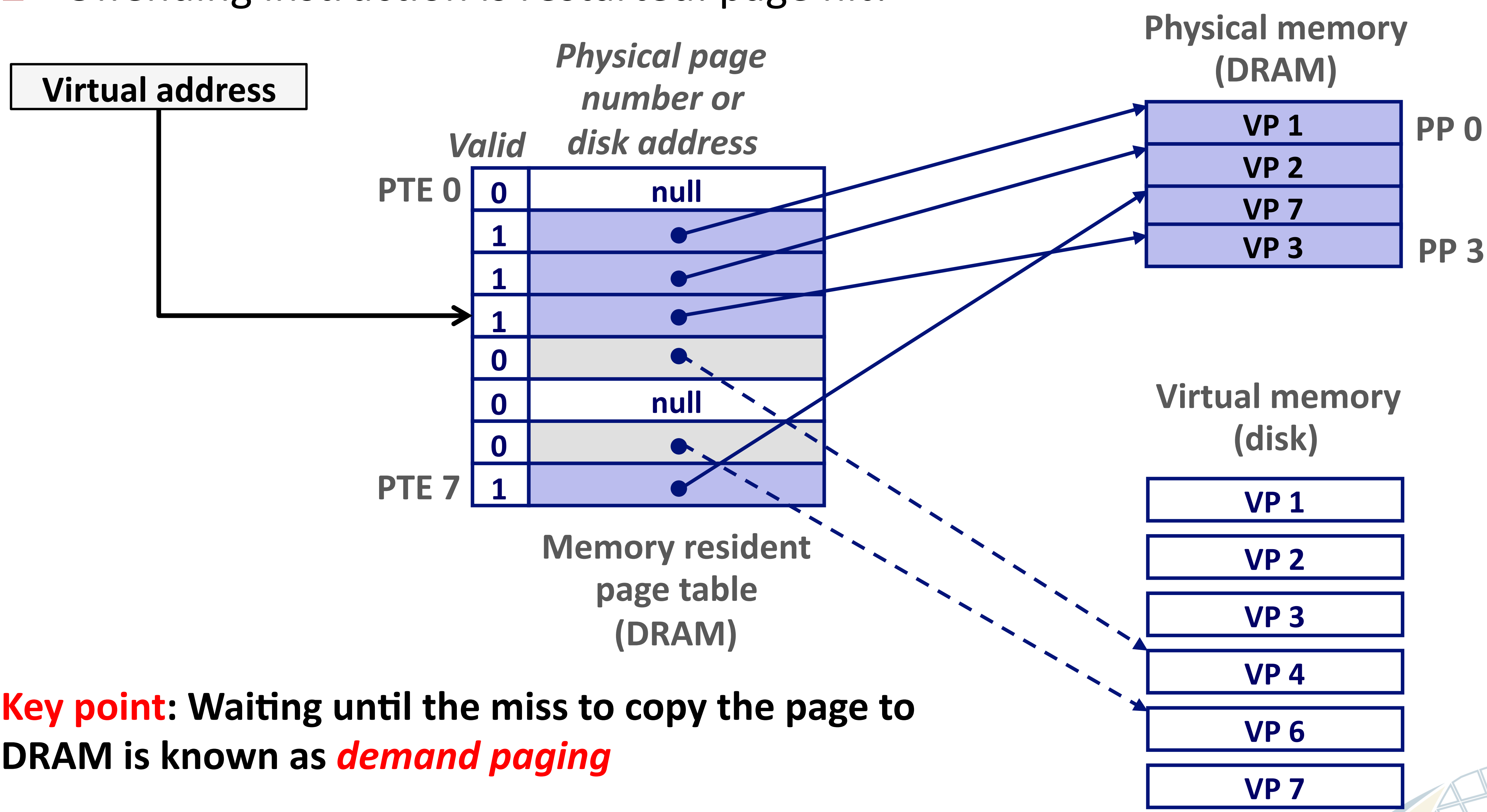
HANDLING PAGE FAULT

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)



HANDLING PAGE FAULT

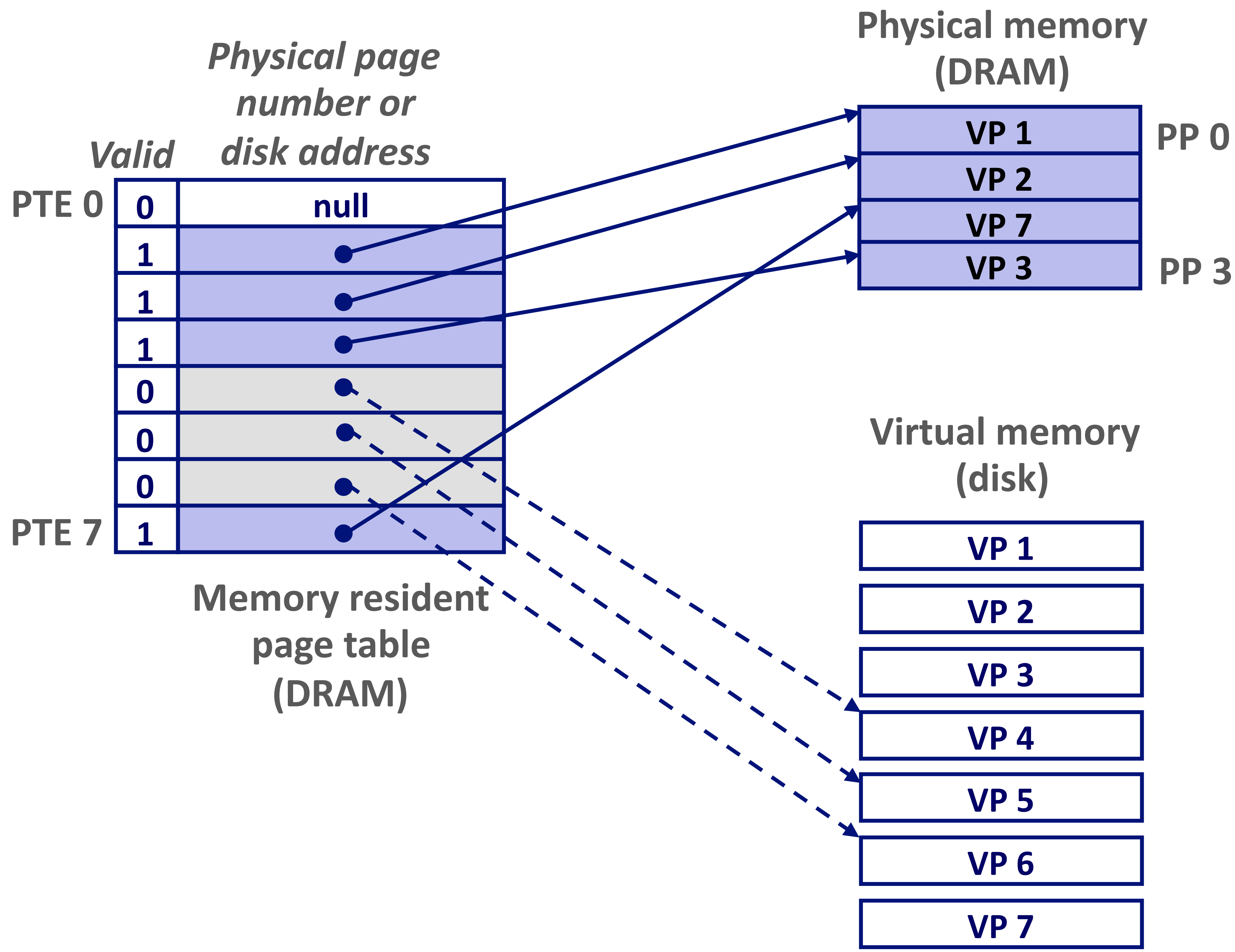
- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
- Offending instruction is restarted: page hit!



Key point: Waiting until the miss to copy the page to DRAM is known as *demand paging*

ALLOCATING PAGES

- **Allocating a new page (VP 5) of virtual memory.**



LOCALITY TO THE RESCUE AGAIN!

- Virtual memory seems terribly inefficient, but it works because of locality.
- At any point in time, programs tend to access a set of active virtual pages called the working set
  - Programs with better temporal locality will have smaller working sets

- If (working set size < main memory size)
  - Good performance for one process after compulsory misses

- If (SUM(working set sizes) > main memory size)
  - Thrashing: Performance meltdown where pages are swapped (copied) in and out continuously

WATCH VIRTUAL MEMORY IN TOP

```
top - 07:22:39 up 18 days, 11:12,  1 user,  load average: 0.00, 0.00, 0.00
Tasks: 1123 total,   1 running, 1122 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.1%us,  0.2%sy,  0.0%ni, 99.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  66057332k total,  3955752k used, 62101580k free,   326808k buffers
Swap: 16777208k total,        0k used, 16777208k free,  2227792k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  nFLT COMMAND
 3556 root      20   0  303m  14m 7688 S  0.0  0.0   0:38.90   93 httpd
 3884 gdm       20   0  360m  31m 9424 S  0.0  0.0   0:22.93   74 gnome-settings-
 3744 root      20   0  188m  35m 6168 S  0.0  0.1   0:46.69   71 Xorg
 3456 rocksdb   20   0  500m  11m 4696 S  0.0  0.0   0:03.79   64 mysqld
 3859 gdm       20   0  262m 7716 6104 S  0.0  0.0   0:00.20   60 gnome-session
 3895 gdm       20   0  371m  15m  10m S  0.0  0.0   0:25.67   21 gdm-simple-gree
 3880 gdm       20   0  117m 4432 3580 S  0.0  0.0   0:04.15   13 at-spi-registry
 3787 root      20   0 4019m 3464 2048 S  0.0  0.0   0:00.44   12 console-kit-dae
 3741 root      20   0  162m 3112 2564 S  0.0  0.0   0:00.00   10 gdm-simple-slav
    1 root      20   0 19364 1560 1236 S  0.0  0.0   0:01.48    9 init
 3894 gdm       20   0  276m 9644 7228 S  0.0  0.0   0:04.32    9 metacity
 2235 root      20   0  319m 2984 1140 S  0.0  0.0   0:27.75    5 rsyslogd
 2298 named     20   0 1884m  42m 2644 S  0.0  0.1   0:02.37    5 named
 2626 haldaemo  20   0 38316 4636 3332 S  0.0  0.0   0:09.47    4 hald
 3634 root      20   0  101m 4832 2484 S  0.0  0.0   0:04.85    4 tracker-server
 3886 gdm       20   0  352m 5080 2412 S  0.0  0.0   0:00.01    4 bonobo-activati
 3897 gdm       20   0  265m 8424 6700 S  0.0  0.0   0:13.59    4 gnome-power-man
 3893 gdm       20   0  131m 1900 1616 S  0.0  0.0   0:00.00    3 gvfsd
 3901 root      20   0 52400 3996 2972 S  0.0  0.0   0:00.14    3 polkitd
 3555 postfix   20   0 81536 3472 2540 S  0.0  0.0   0:01.43    2 qmgr
 3638 root      20   0  132m 2204 1892 S  0.0  0.0   0:00.03    2 gdm-binary
 3866 gdm       20   0  131m 4484 2212 S  0.0  0.0   0:04.97    2 gconfd-2
 3896 gdm       20   0  242m 7244 5736 S  0.0  0.0   0:00.01    2 polkit-gnome-au
 2627 root      20   0 20328 1176  976 S  0.0  0.0   0:00.00    1 hald-runner
 2666 haldaemo  20   0 17936 1028  888 S  0.0  0.0   0:00.00    1 hald-addon-acpi
 2668 root      20   0 22448 1092  932 S  0.0  0.0   0:00.00    1 hald-addon-inpu
```

WATCH VIRTUAL MEMORY IN TOP

n: **%MEM** -- Memory usage (RES)
A task's currently used share of available **physical** memory.

o: **VIRT** -- Virtual Image (kb)
The total amount of **virtual** memory used by the task. It includes all code, data and shared libraries plus pages that have been swapped out. (Note: you can define the STATSIZE=1 environment variable and the VIRT will be calculated from the /proc/#/state VmSize field.)

p: **SWAP** -- Swapped size (kb)
Per-process swap values are now taken from /proc/#/status VmSwap field.

q: **RES** -- Resident size (kb)
The non-swapped **physical** memory a task has used.

RES = CODE + DATA.

r: **CODE** -- Code size (kb)
The amount of **physical** memory devoted to executable code, also known as the 'text resident set' size or TRS.

s: **DATA** -- Data+Stack size (kb)
The amount of **physical** memory devoted to other than executable code, also known as the 'data resident set' size or DRS.

t: **SHR** -- Shared Mem size (kb)
The amount of **shared** memory used by a task. It simply reflects memory that could be potentially shared with other processes.

u: **nFLT** -- Page Fault count
The number of **major** page faults that have occurred for a task. A page fault occurs when a process attempts to read from or write to a virtual page that is not currently present in its address space. A major page fault is when disk access is involved in making that page available.

v: **nDRT** -- Dirty Pages count
The number of pages that have been modified since they were last written to disk. Dirty pages must be written to disk before the corresponding physical memory location can be used for some other virtual page.