Demand Paging and Page Replacement Algorithms

ECE 469, Feb 05

Aravind Machiry

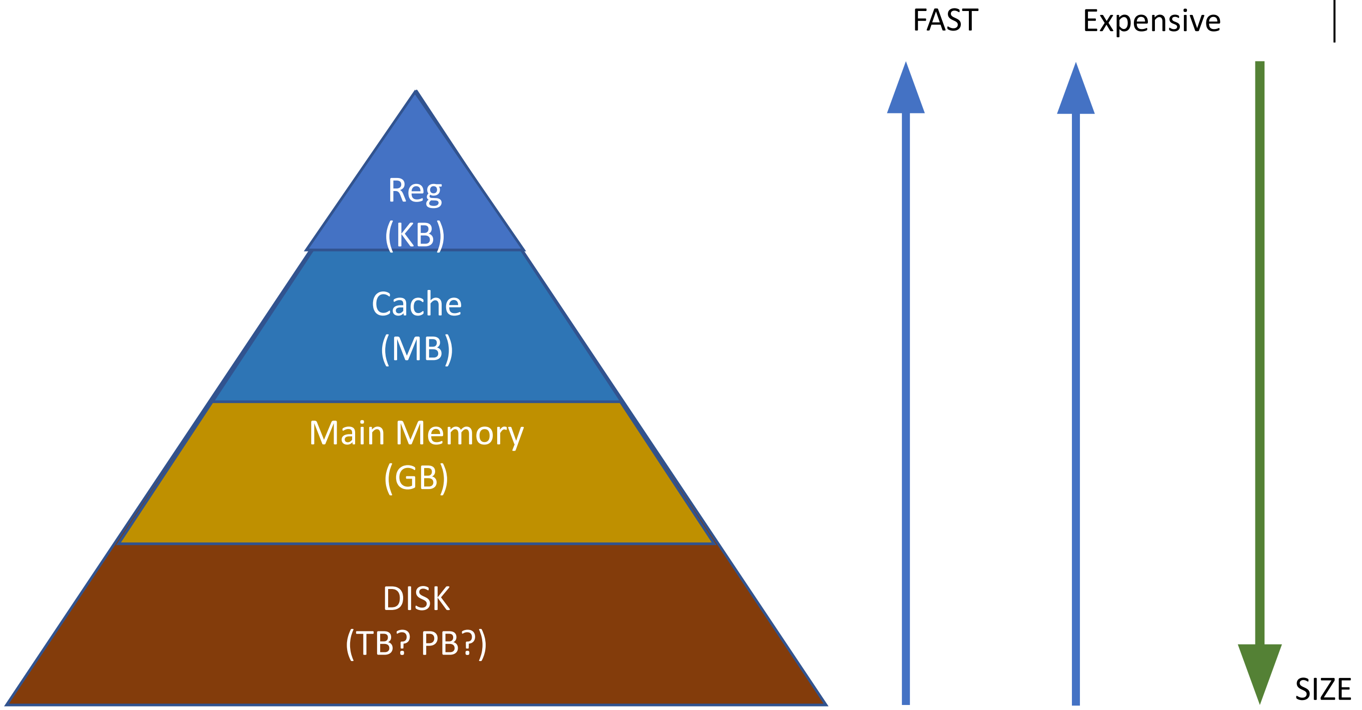
Handling low memory

- Suppose you have 8GB of main memory

- Can you run a program that its program size is 16GB?
  - Yes, you can load them part by part
  - This is because we do not use all of data at the same time

- Can your OS do this execution seamlessly to your application?

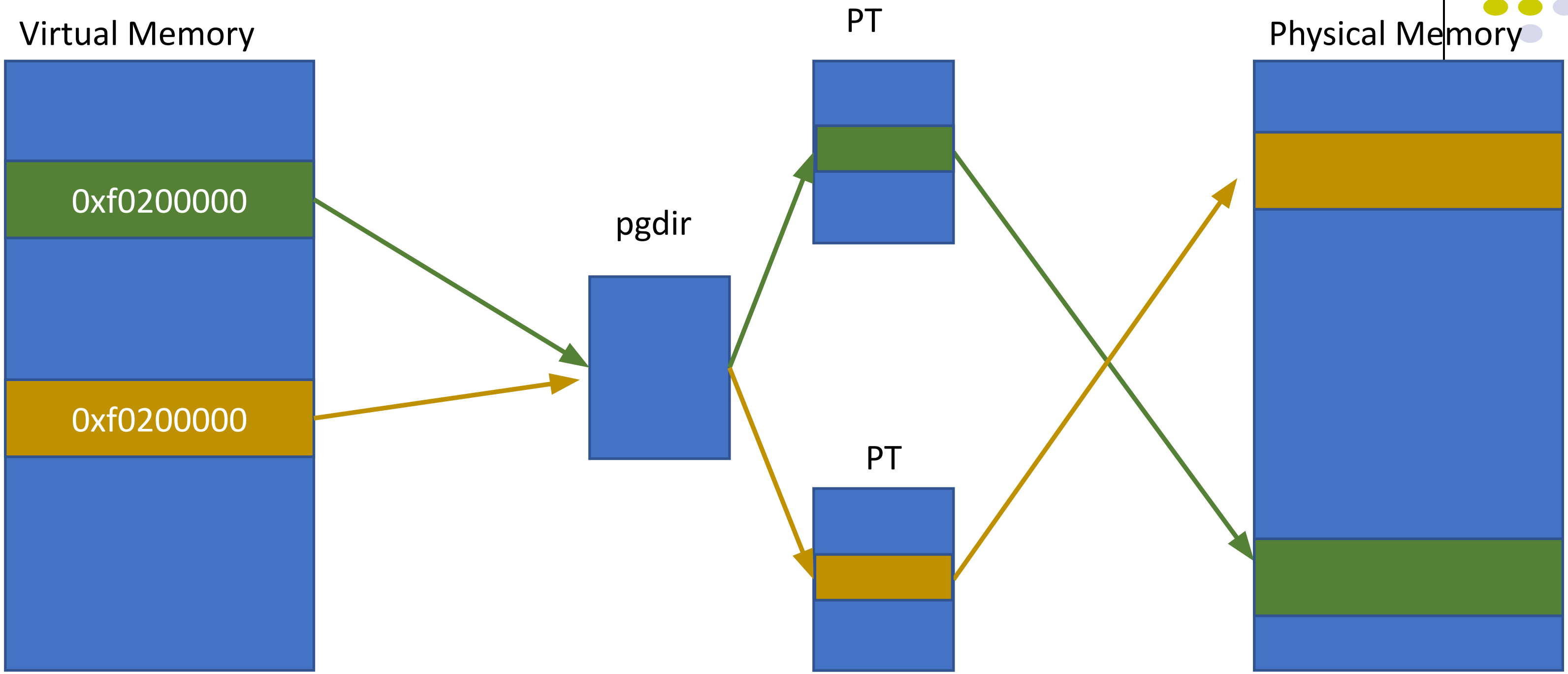
Memory Hierarchy



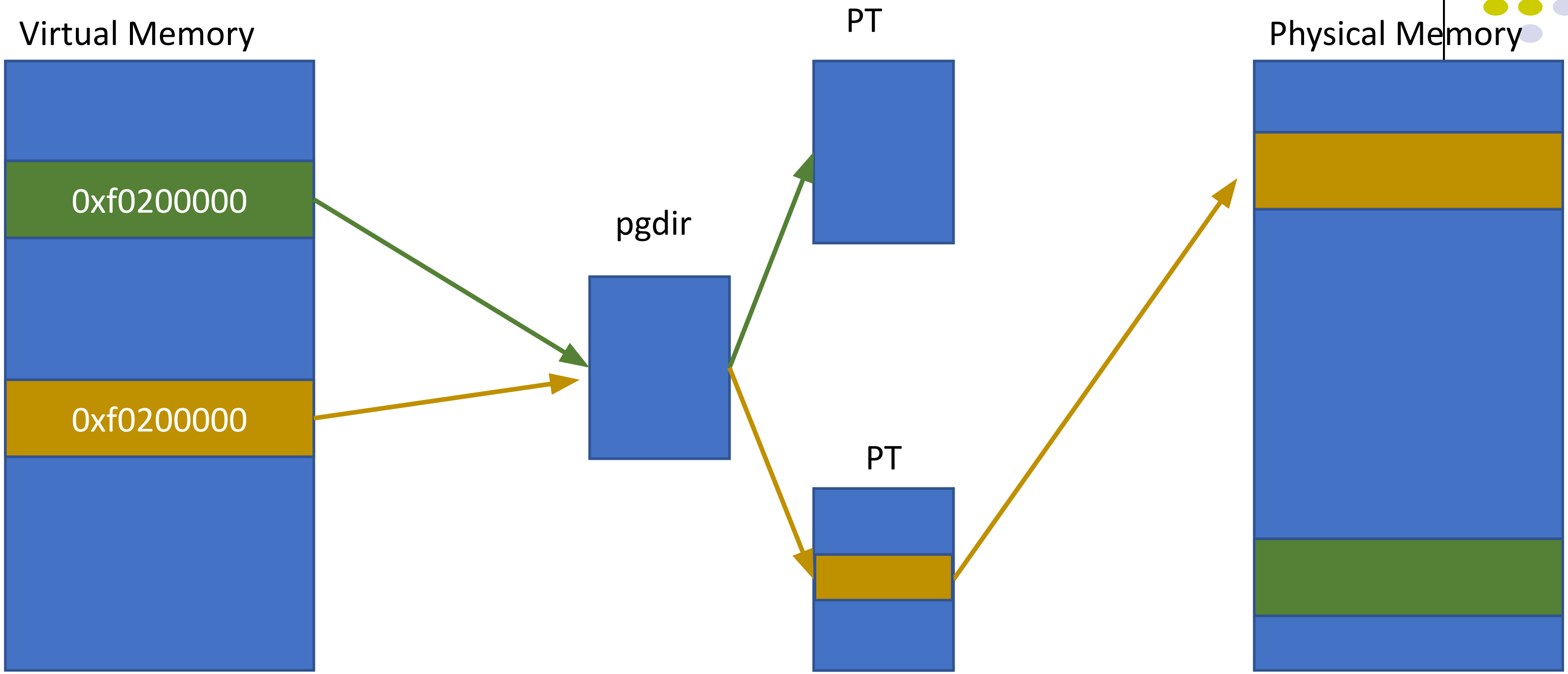
Memory Swapping

- Use disk as backing store under memory pressure

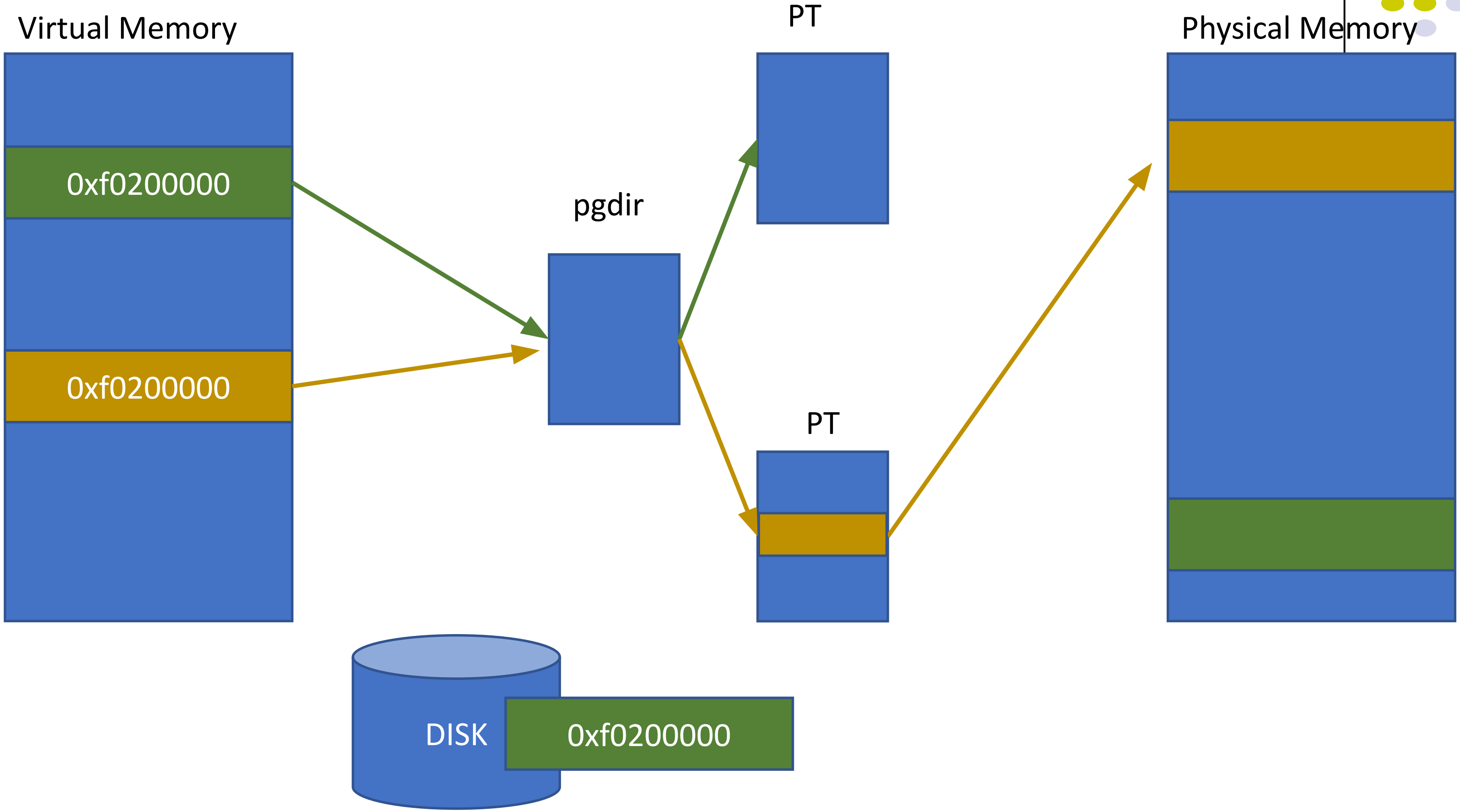
Memory Swapping



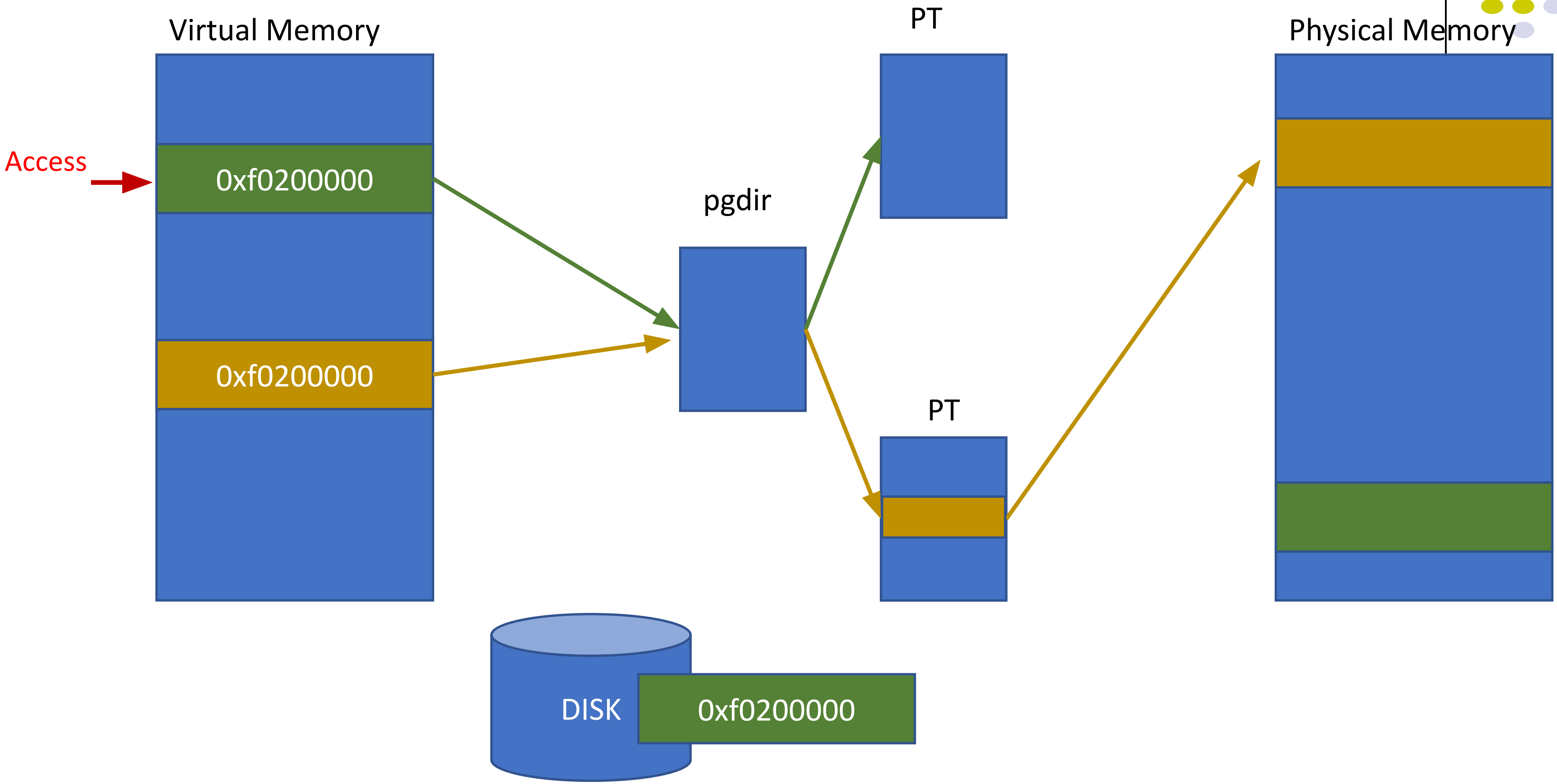
Memory Swapping - Removing a page



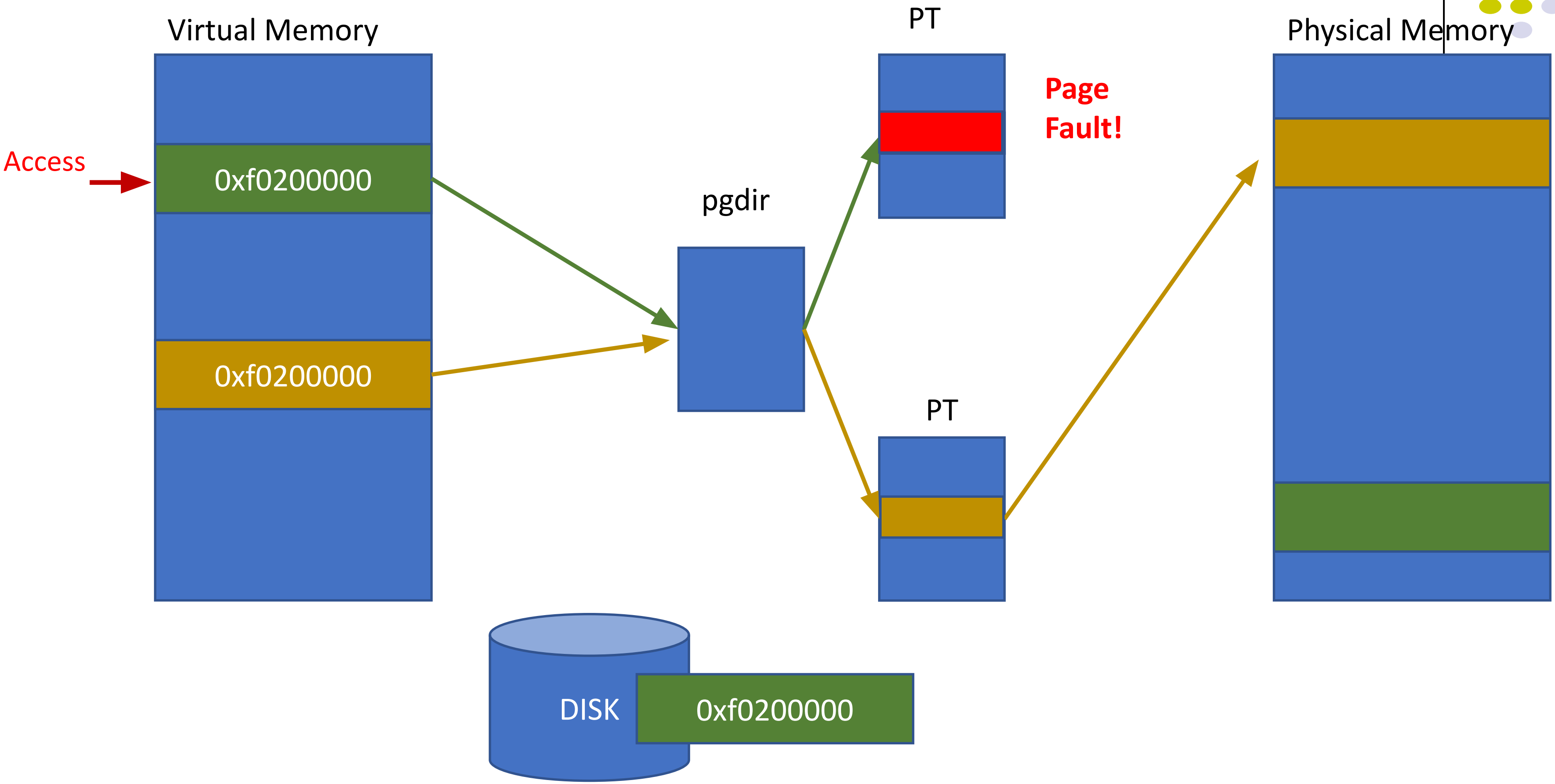
Memory Swapping - Removing a page



Memory Swapping - Removing a page



Memory Swapping - Removing a page



Swapping - Transparently load page from disk

- Page fault handler

Swapping - Transparently load page from disk

- Page fault handler
  - Read CR2 (get address, `0xf0200000`)

Swapping - Transparently load page from disk

- Page fault handler
  - Read CR2 (get address, `0xf0200000`)
  - Read error code

Swapping - Transparently load page from disk

- Page fault handler
  - Read CR2 (get address, `0xf0200000`)
  - Read error code
- If error code says that the fault is caused by non-present page and

Swapping - Transparently load page from disk

- Page fault handler
  - Read CR2 (get address, `0xf0200000`)
  - Read error code
- If error code says that the fault is caused by non-present page and
- The faulting page of the current process is stored in the disk

Swapping - Transparently load page from disk

- Page fault handler
  - Read CR2 (get address, `0xf0200000`)
  - Read error code
- If error code says that the fault is caused by non-present page and
- The faulting page of the current process is stored in the disk
  - Lookup disk if it swapped put 0xf0200000 of this environment (process)

Swapping - Transparently load page from disk

- Page fault handler
    - Read CR2 (get address, `0xf0200000`)
    - Read error code
- If error code says that the fault is caused by non-present page and
- The faulting page of the current process is stored in the disk
    - Lookup disk if it swapped put 0xf0200000 of this environment (process)
        - This must be per process because virtual address is per-process resource

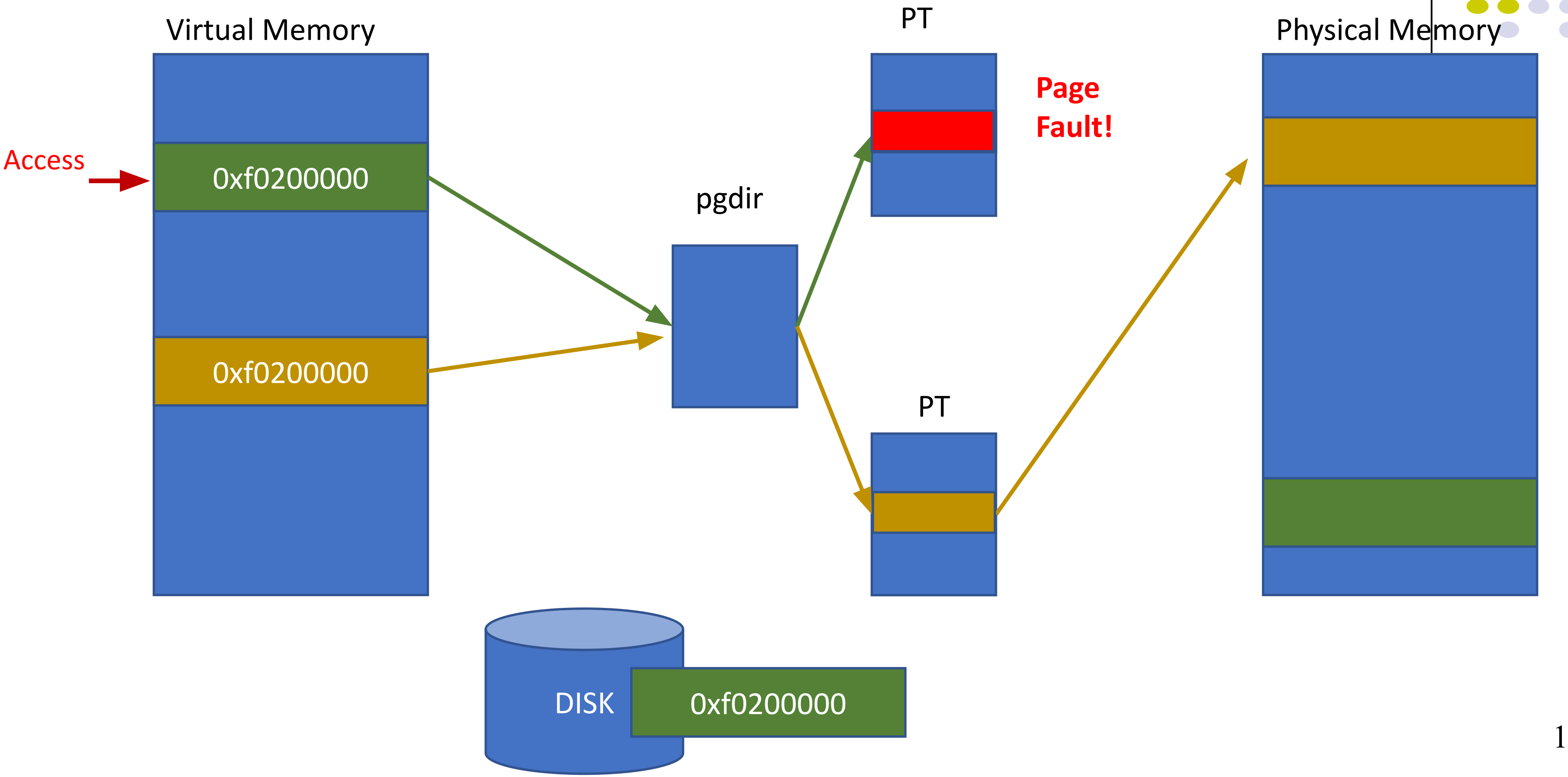
Swapping - Transparently load page from disk

- Page fault handler
  - Read CR2 (get address, `0xf0200000`)
  - Read error code
- If error code says that the fault is caused by non-present page and
- The faulting page of the current process is stored in the disk
  - Lookup disk if it swapped put 0xf0200000 of this environment (process)
    - This must be per process because virtual address is per-process resource
- Load that page into physical memory

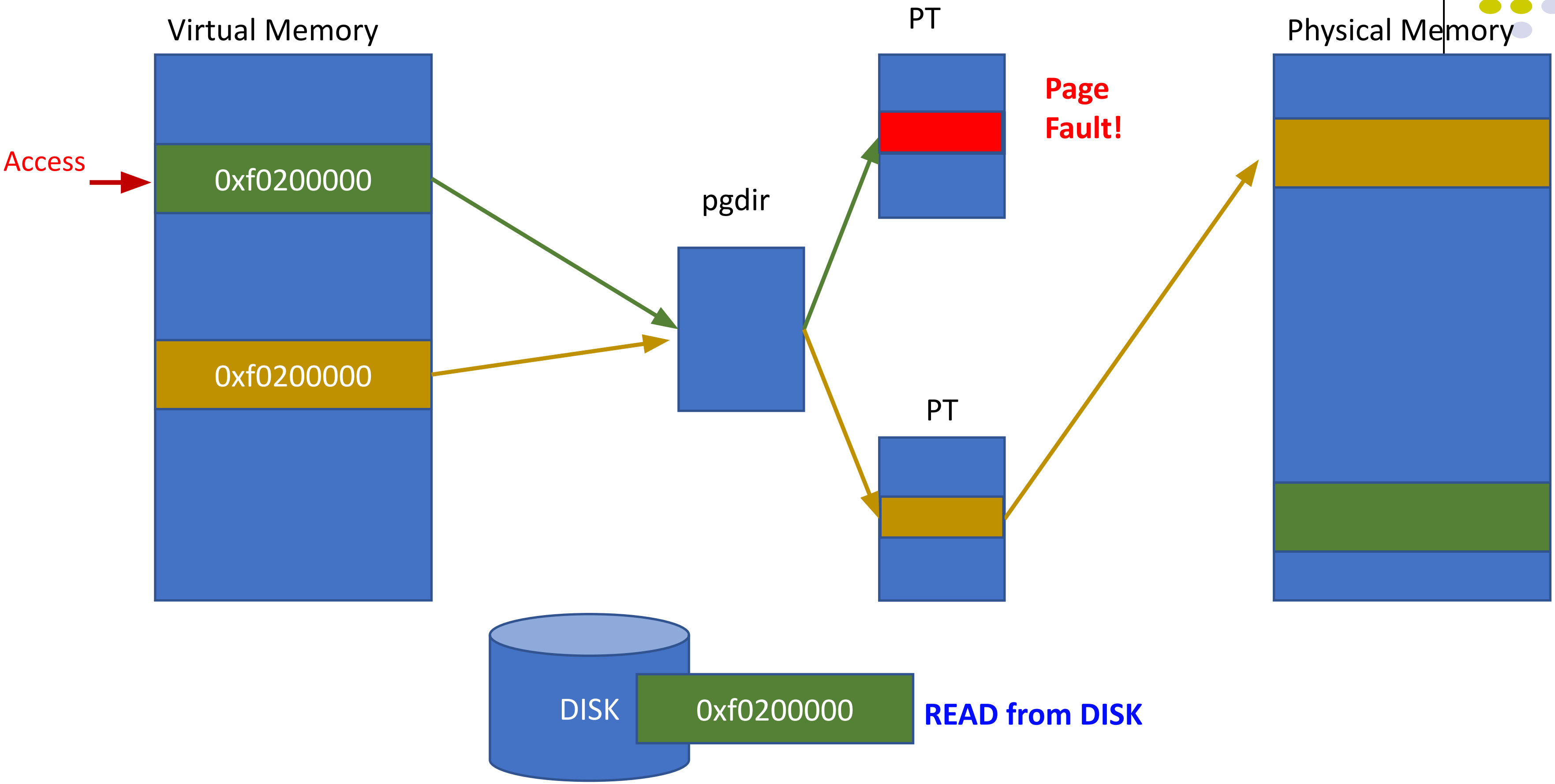
Swapping - Transparently load page from disk

- Page fault handler
  - Read CR2 (get address, `0xf0200000`)
  - Read error code
- If error code says that the fault is caused by non-present page and
- The faulting page of the current process is stored in the disk
  - Lookup disk if it swapped put 0xf0200000 of this environment (process)
    - This must be per process because virtual address is per-process resource
- Load that page into physical memory
- Map it and then continue!

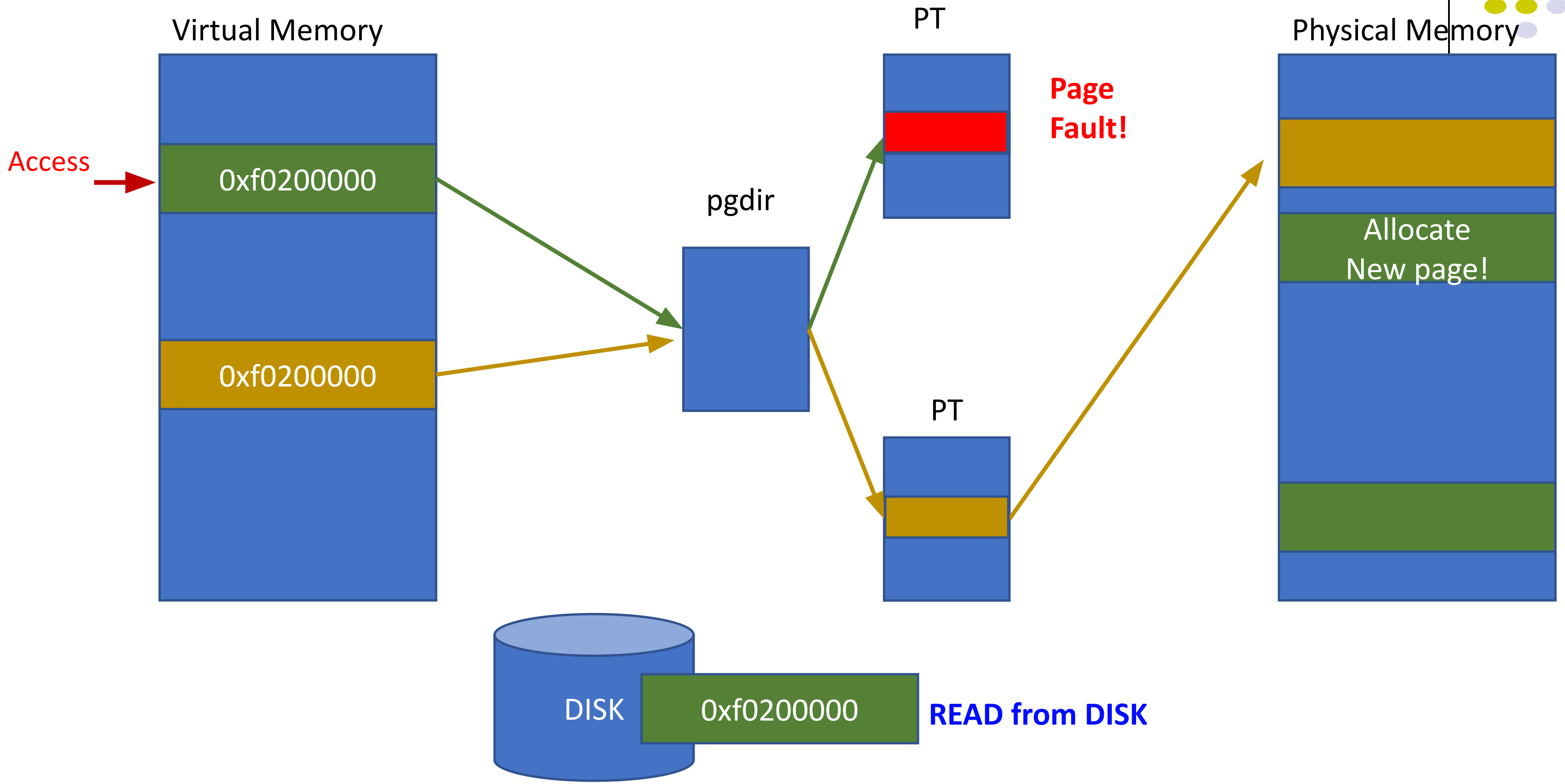
Swapping - Transparently load page from disk



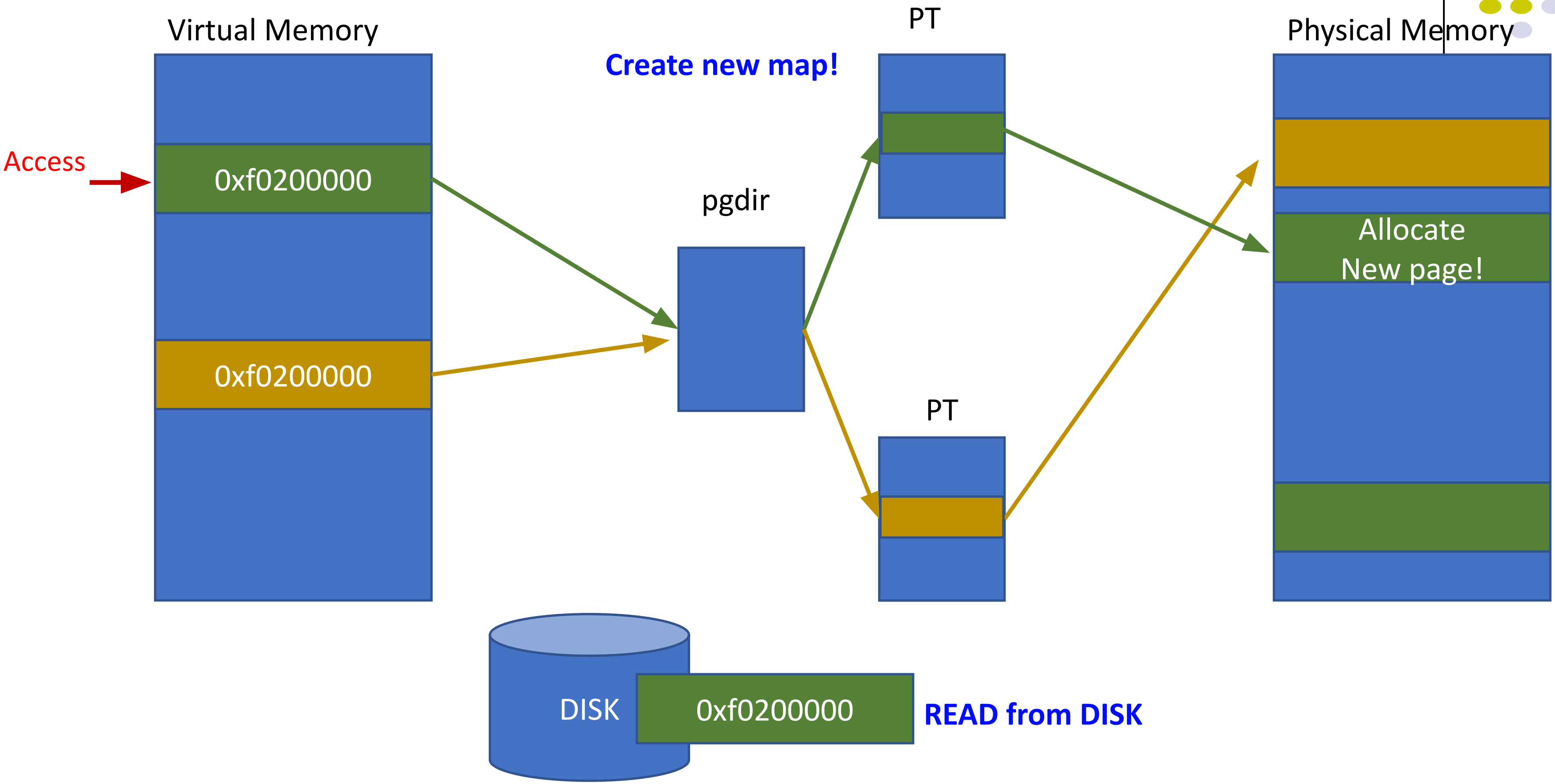
Swapping - Transparently load page from disk



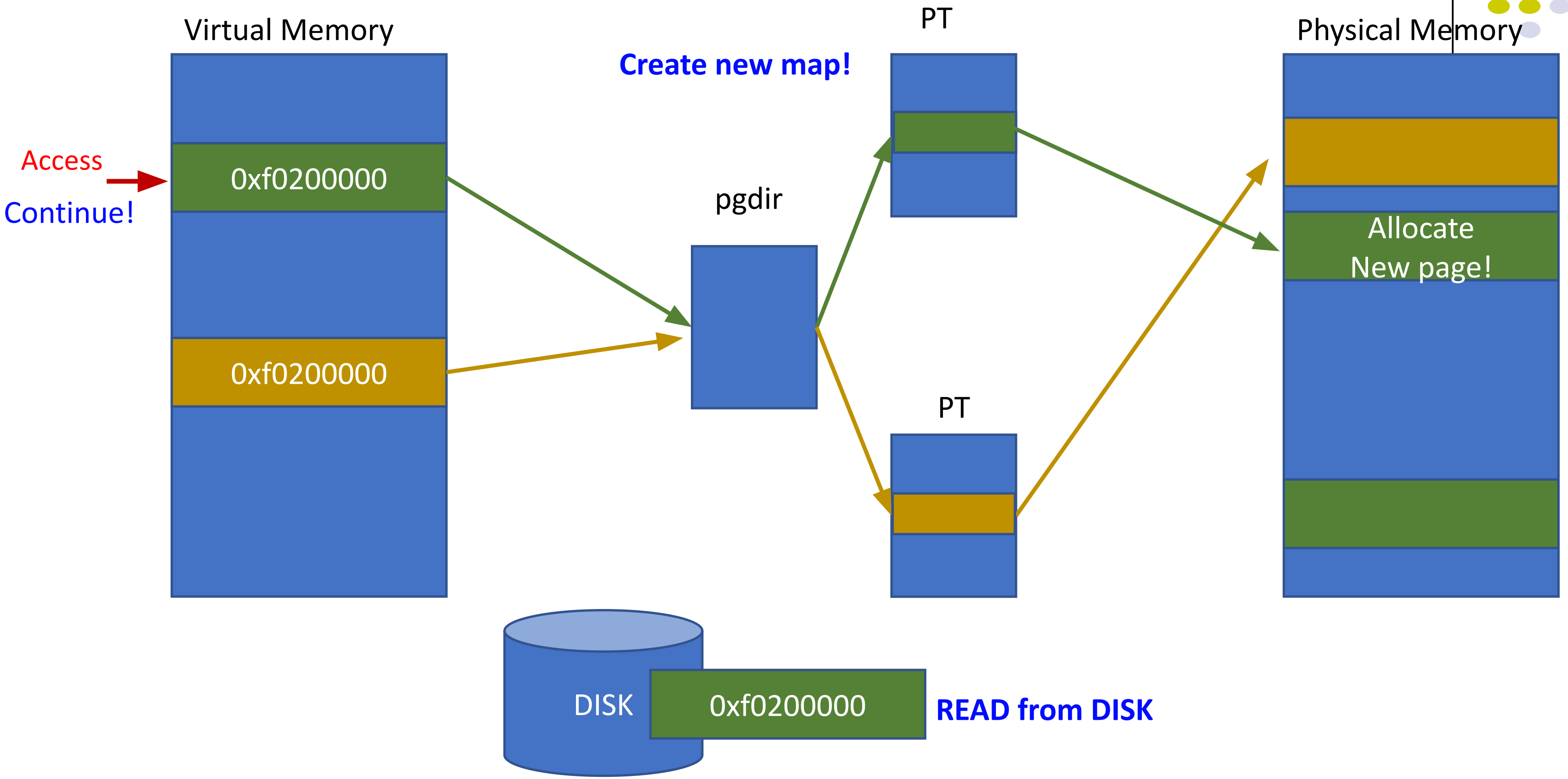
Swapping - Transparently load page from disk



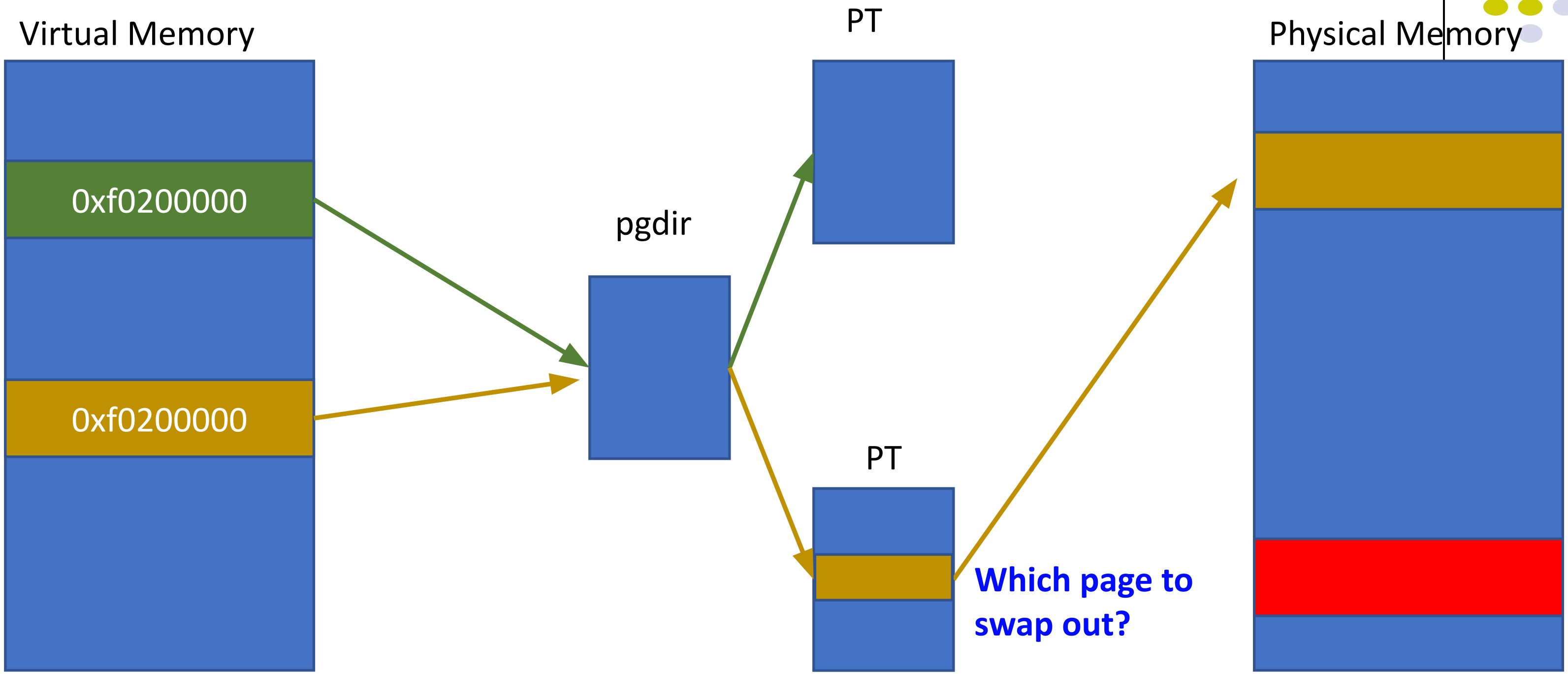
Swapping - Transparently load page from disk



Swapping - Transparently load page from disk



Selecting Page to Swap out!



Which page to swap out?

System at Full Memory Capacity

- Expect to run with all phy. pages in use
- Every “page-in” requires an eviction

System at Full Memory Capacity

- Expect to run with all phy. pages in use
- Every “page-in” requires an eviction
- Goal of page replacement
  - Maximize hit rate -> kick out the page that’s least useful

System at Full Memory Capacity

- Expect to run with all phy. pages in use
- Every “page-in” requires an eviction
- Goal of page replacement
  - Maximize hit rate -> kick out the page that’s least useful
    - Challenge: how do we determine utility?
      - Kick out pages that aren’t likely to be used again

System at Full Memory Capacity

- Expect to run with all phy. pages in use
- Every “page-in” requires an eviction
- Goal of page replacement
  - Maximize hit rate -> kick out the page that’s least useful
    - Challenge: how do we determine utility?
      - Kick out pages that aren’t likely to be used again
- Page replacement is a difficult policy problem

Finding Least Useful Page is Hard

- Don’t know future!

Finding Least Useful Page is Hard

- Temporal Locality:
  - Past behavior is a good indication of future behavior! (e.g. LRU)

- Perfect (past) reference stream hard to get
  - Every memory access would need bookkeeping
  - Is this feasible (in software? In hardware?)

Finding Least Useful Page is Hard

- Temporal Locality:
    - Past behavior is a good indication of future behavior! (e.g. LRU)
- Perfect (past) reference stream hard to get
    - Every memory access would need bookkeeping
    - Is this feasible (in software? In hardware?)
- Minimize overhead
    - If no memory pressure, ideally no bookkeeping
    - In other words, make the common case fast (page hit)

Finding Least Useful Page is Hard

- → Get imperfect information, while guaranteeing foreground perf
  - What is minimum hardware support that need to added?

Definitions
(or Jargons asked during interviews)

- **Pressure** – the demand for some resource (often used when demand exceeds supply)

  ex: the system experienced memory pressure
- **Eviction** – throwing something out

  ex: cache lines and memory pages got evicted
- **Pollution** – bringing in useless pages/lines

  ex: this strategy causes high cache pollution

Definitions

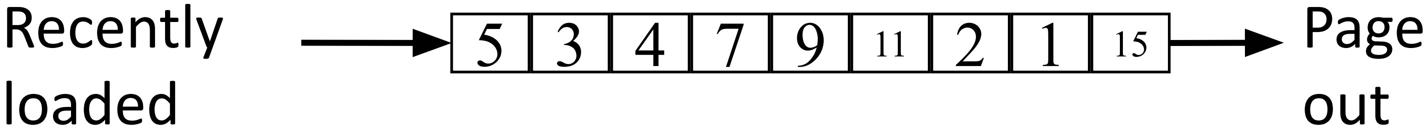
- **Thrashing** – extremely high rate of moving things in and out (usually unnecessarily)

- **Locality** – re-use – it makes the world go rounds!
- **Temporal Locality** – re-use in time
- **Spatial Locality** – re-use of close by locations

Performance metric for Page Replacement algorithms

- Give a sequence of memory accesses, minimize the # of page faults
  - Similar to cache miss rate
  - What about hit latency and miss latency?

First In First Out (FIFO)



Recently loaded

5	3	4	7	9	11	2	1	15

Page out

- Algorithm
  - Throw out the oldest page
- Pros
  - Low-overhead implementation
- Cons
  - No frequency/no recency □ may replace the heavily used pages

First In First Out (FIFO)

- For a given set of page references, what happens when we increase the physical memory?

First In First Out (FIFO)

- For a given set of page references, what happens when we increase the physical memory?
  - Expected: Number of page faults decreases.
- Are your sure!?

Belady’s anomaly

Belady's anomaly: Laszlo Belady states that it is possible to have more page faults when increasing the number of page frames.

Previously, it was believed that an increase in the number of page frames would always provide the same number or fewer page faults.

Example

Page Requests

321032432104

Example (Page Faults in Red)

Page Requests – 3 frames

Total Page Faults: 9

	3	2	1	0	3	2	4	3	2	1	0	4
Frame 1	3	3	3	0	0	0	4	4	4	4	4	4
Frame 2		2	2	2	3	3	3	3	3	1	1	1
Frame 3			1	1	1	2	2	2	2	2	0	0

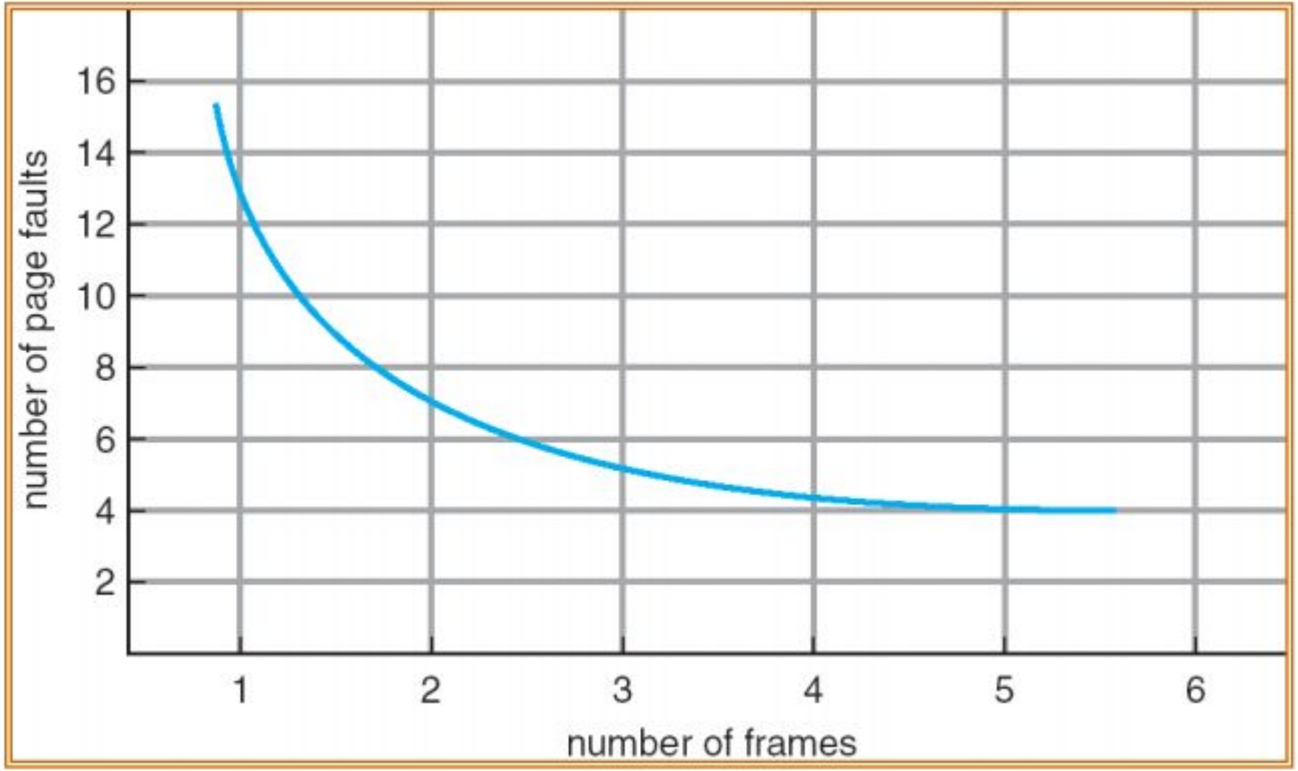
Example (Page Faults in Red)

Page Requests – 4 frames

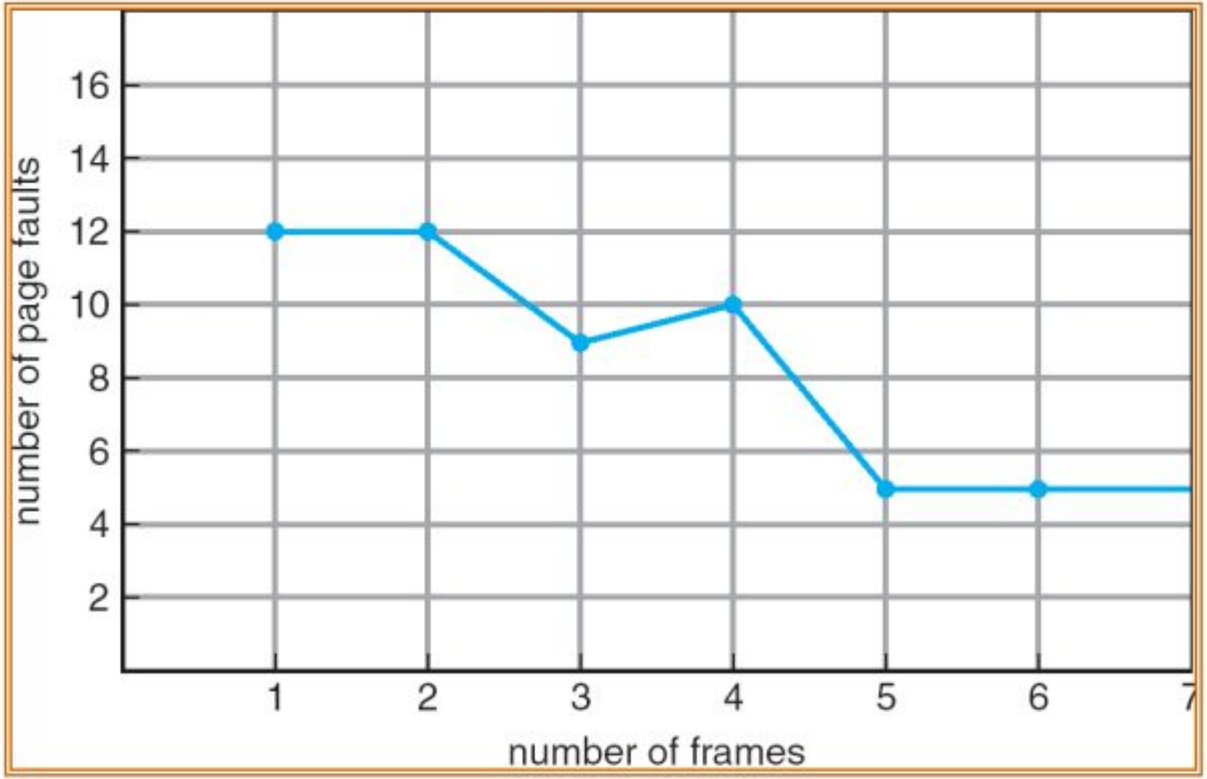
Total Page Faults: 10

	3	2	1	0	3	2	4	3	2	1	0	4
Frame 1	🟥 3	3	3	3	3	3	🟥 4	4	4	4	🟥 0	0
Frame 2		🟥 2	2	2	2	2	2	🟥 3	3	3	3	🟥 4
Frame 3			🟥 1	1	1	1	1	1	🟥 2	2	2	2
Frame 4				🟥 0	0	0	0	0	0	🟥 1	1	1

Ideal curve of # of page faults v.s. # of physical pages



FIFO illustrating Belady's anomaly



Optimal or MIN

- Algorithm (also called Belady’s Algorithm)
  - Replace the page that won’t be used for the **longest** time
- Pros
  - Minimal page faults (can you prove it?)
  - Used as an **off-line** algorithm for perf. analysis
- Cons
  - **No on-line** implementation
- What was the CPU scheduling algorithm of similar nature?

Predicting Future based on Past

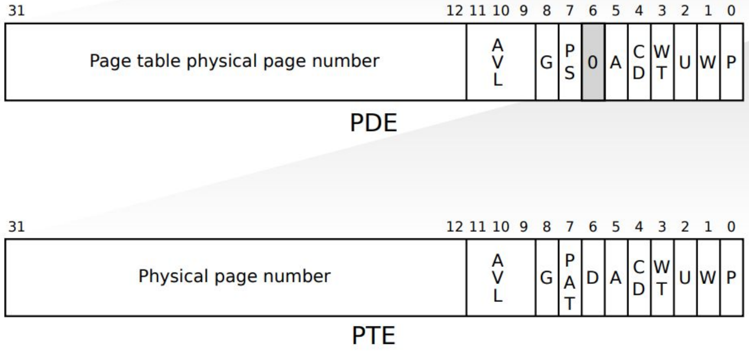
- "Principle of locality"
  - Recency:
    - Page recently used are likely to be used again in the near future
  - Frequency:
    - Pages frequently used (recently) are likely to be used frequently again in the near future
- Is this temporal or spatial locality?

How to record locality?

- Software Solution!?

How to record locality?

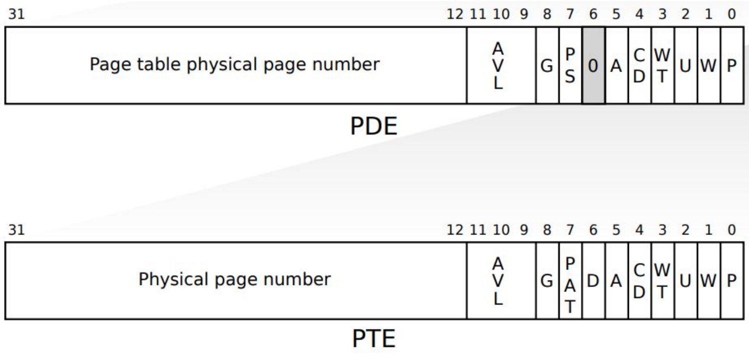
- Can hardware give any hints?



P	Present
W	Writable
U	User
WT	1=Write-through, 0=Write-back
CD	Cache disabled
A	Accessed
D	Dirty
PS	Page size (0=4KB, 1=4MB)
PAT	Page table attribute index
G	Global page
AVL	Available for system use

How to record locality?

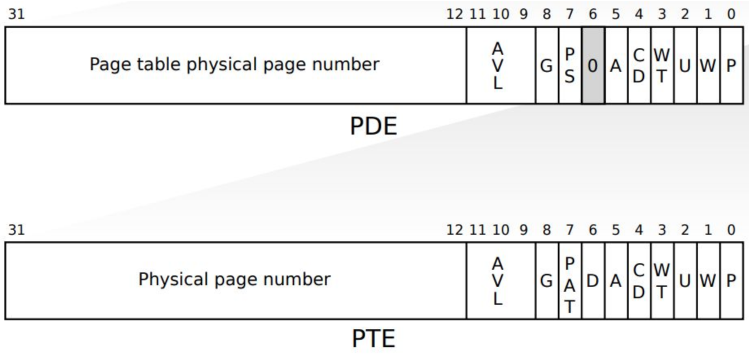
- Can hardware give any hints?



P	Present
W	Writable
U	User
WT	1=Write-through, 0=Write-back
CD	Cache disabled
A	Accessed
D	Dirty
PS	Page size (0=4KB, 1=4MB)
PAT	Page table attribute index
G	Global page
AVL	Available for system use

How to record locality?

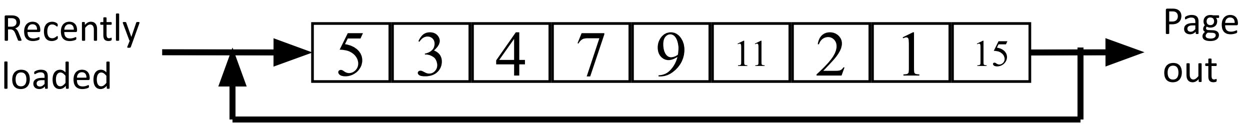
- Can hardware give any hints?



P	Present
W	Writable
U	User
WT	1=Write-through, 0=Write-back
CD	Cache disabled
A	Accessed
D	Dirty
PS	Page size (0=4KB, 1=4MB)
PAT	Page table attribute index
G	Global page
AVL	Available for system use

Accessed or Reference bit: A hardware bit that is set whenever the page is referenced (read or written)

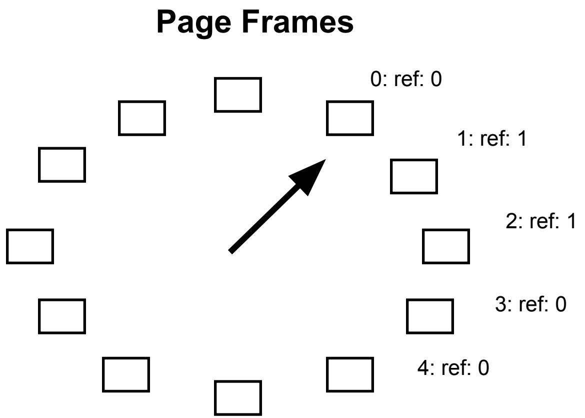
FIFO with Second Chance



If reference bit is 1

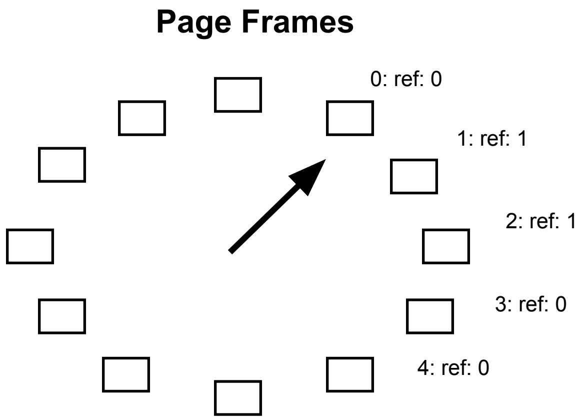
- Algorithm
  - Check the reference-bit of the oldest page (first in)
  - If it is 0, then replace it
  - If it is 1, clear the referent-bit, put it to the end of the list, and continue searching
- Pros
  - Fast
  - Frequency □ do not replace a heavily used page
- Cons
  - The worst case may take a long time

Clock: a simple FIFO with 2nd chance



- FIFO clock algorithm
  - Maintain the list of page frames
  - Hand points to the oldest page
  - On a page fault, follow the hand to inspect pages
- Second chance
  - If the reference bit is 1, set it to 0 and advance the hand
  - If the reference bit is 0, use it for replacement
- What is the difference between Clock and the previous one?
  - Mechanism vs. policy?

Clock: a simple FIFO with 2nd chance



- What happens if all reference bits are 1?
- What does it suggest if observing clock hand is sweeping very fast?
- What does it suggest if clock hand is sweeping very slow?

Least Recently Used (LRU)

- Algorithm
  - Replace page that hasn’t been used for the longest time

- Advantage: with locality, LRU approximates Optimal

Implementing LRU: software

- A doubly linked list of pages
- Every time page is referenced, move it to the front of the list
- **Page replacement**: remove the page from back of list
  - Avoid scanning of all pages
- **Problem**: too expensive
  - Requires 6 pointer updates for each page reference info
  - High contention on multiprocessor

Least Recently Used (LRU)

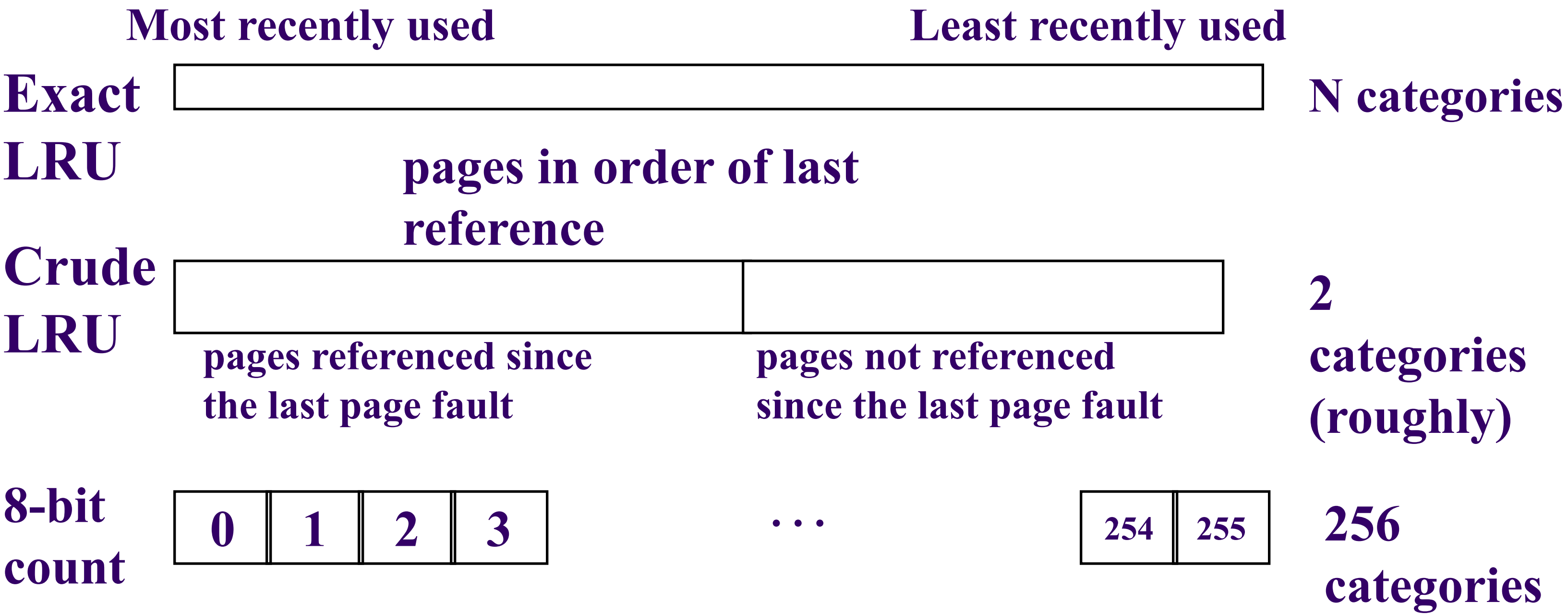
- What hardware mechanisms are required to implement LRU?

Implementing LRU: hardware/software

- A timestamp for each page
- Every time page is referenced, save system clock into the timestamp of the page
- Page replacement: scan through pages to find the one with the oldest clock

- Problem: have to search all pages/counters!

Approximate LRU



Most recently used

Least recently used

Exact LRU

N categories

pages in order of last reference

Crude LRU

2 categories (roughly)

pages referenced since the last page fault

pages not referenced since the last page fault

8-bit count

0 1 2 3 … 254 255

256 categories

Keep 8-bit counter for each page in a table in memory

Approximate LRU

Initial

00000000
00000000
00000000
00000000

Initial

Page Table

Ref	Frame #
0	3
0	2
0	0
0	1

Approximate LRU

Initial	Interval 1
00000000	00000000
00000000	00000000
00000000	10000000
00000000	00000000

Page Fault Victim?

Interval 1

Page Table

Ref	Frame #
0	3
1	2
0	0
0	1

⇒

Ref	Frame #
0	3
0	2
0	0
0	1

Approximate LRU

Initial	Interval 1	Interval 2
00000000	00000000	00000000
00000000	00000000	10000000
00000000	10000000	11000000
00000000	00000000	00000000

Page Fault Victim?

Interval 2

Page Table

Ref	Frame #
0	3
1	2
0	0
1	1

⇒

Ref	Frame #
0	3
0	2
0	0
0	1

Approximate LRU

Initial	Interval 1	Interval 2	Interval 3
00000000	00000000	00000000	10000000
00000000	00000000	10000000	01000000
00000000	10000000	11000000	11100000
00000000	00000000	00000000	00000000

Page Fault Victim?

Interval 3

Page Table

Ref	Frame #
0	3
1	2
1	0
0	1

Ref	Frame #
0	3
0	2
0	0
0	1

Approximate LRU

Initial	Interval 1	Interval 2	Interval 3	Interval 4
00000000	00000000	00000000	10000000	01000000
00000000	00000000	10000000	01000000	10100000
00000000	10000000	11000000	11100000	01110000
00000000	00000000	00000000	00000000	10000000

Page Fault Victim?

Interval 4

Page Table

Ref	Frame #
1	3
0	2
0	0
1	1

Ref	Frame #
0	3
0	2
0	0
0	1

Approximate LRU

Initial	Interval 1	Interval 2	Interval 3	Interval 4	Interval 5
00000000	00000000	00000000	10000000	01000000	10100000
00000000	00000000	10000000	01000000	10100000	01010000
00000000	10000000	11000000	11100000	01110000	01110000
00000000	00000000	00000000	00000000	10000000	01000000

```
Page Fault Victim?
```

Interval 5

Page Table

Ref	Frame #
0	3
0	2
1	0
0	1

⇒

Ref	Frame #
0	3
0	2
0	0
0	1

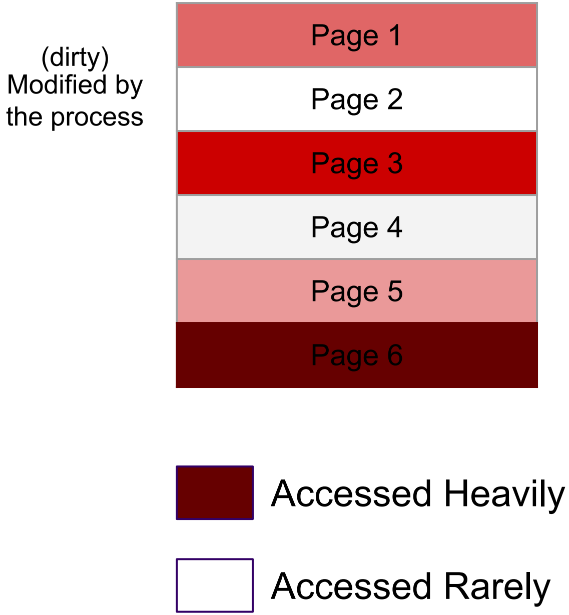
Approximate LRU

Initial	Interval 1	Interval 2	Interval 3	Interval 4	Interval 5
00000000	00000000	00000000	10000000	01000000	10100000
00000000	00000000	10000000	01000000	10100000	01010000
00000000	10000000	11000000	11100000	01110000	01110000
00000000	00000000	00000000	00000000	10000000	01000000

- Algorithm
  - At regular interval, OS shifts reference bits (in PTE) into counters (and clear reference bits)
  - Replacement: Pick the page with the “smallest counter”
- How many bits are enough?
  - In practice 8 bits are quite good
- Pros: **Require one reference bit, small counter/page**
- Cons: **Require looking at many counters (or sorting)**

Which page to evict? (Victim page)

Heat map of the page usages.



(dirty)
Modified by
the process

Page 1

Page 2

Page 3

Page 4

Page 5

Page 6

🟥 Accessed Heavily

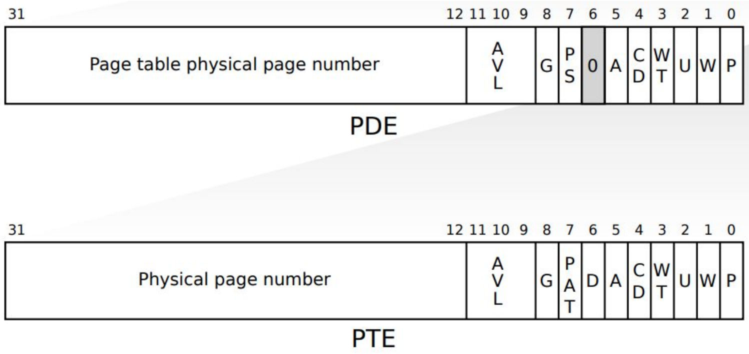
□ Accessed Rarely

We have focused on miss rate. What about miss latency?

- Key observation: it is cheaper to pick a “clean” page over a “dirty” page
  - Clean page does not need to be swapped to disk

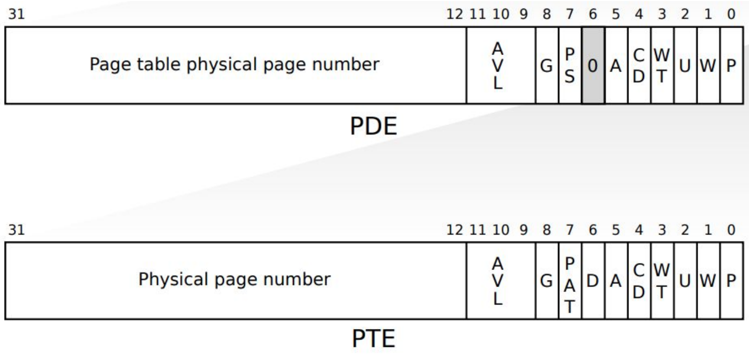
- Challenge:
  - How to get this info?

Let's look back at PTE entries!



P	Present
W	Writable
U	User
WT	1=Write-through, 0=Write-back
CD	Cache disabled
A	Accessed
D	Dirty
PS	Page size (0=4KB, 1=4MB)
PAT	Page table attribute index
G	Global page
AVL	Available for system use

Dirty Bit - Modified bit



P	Present
W	Writable
U	User
WT	1=Write-through, 0=Write-back
CD	Cache disabled
A	Accessed
D	Dirty
PS	Page size (0=4KB, 1=4MB)
PAT	Page table attribute index
G	Global page
AVL	Available for system use

Enhanced FIFO with 2nd Chance

Same as the basic FIFO with 2nd chance, except that it considers both (reference bit, modified bit)

Ref, Mod	Needed Soon?	Replacement Cost?	Preference
0, 0	Unlikely	Low (Drop the page)	😍
0, 1	Unlikely	High (Write to disk)	😄
1, 0	Likely	Low (Drop the page)	😊
1, 1	Likely	High (Write to disk)	😟

Enhanced FIFO with 2nd Chance

- On page fault, follow hand to inspect pages:
  - Round 1:
    - If bits are (0,0), take it
    - if bits are (0,1), record 1st instance
    - Clear ref bit for (1,0) and (1,1), if (0,1) not found yet
  - At end of round 1, if (0,1) was found, take it
  - If round 1 does not succeed, try 1 more round

Summary: Page Replacement Algorithms

- Optimal

- FIFO
- Random

- Approximate LRU (NRU)

- FIFO with 2nd chance
- Clock: a simple FIFO with 2nd chance
- Enhanced FIFO with 2nd chance