CS 471 Operating Systems

Yue Cheng

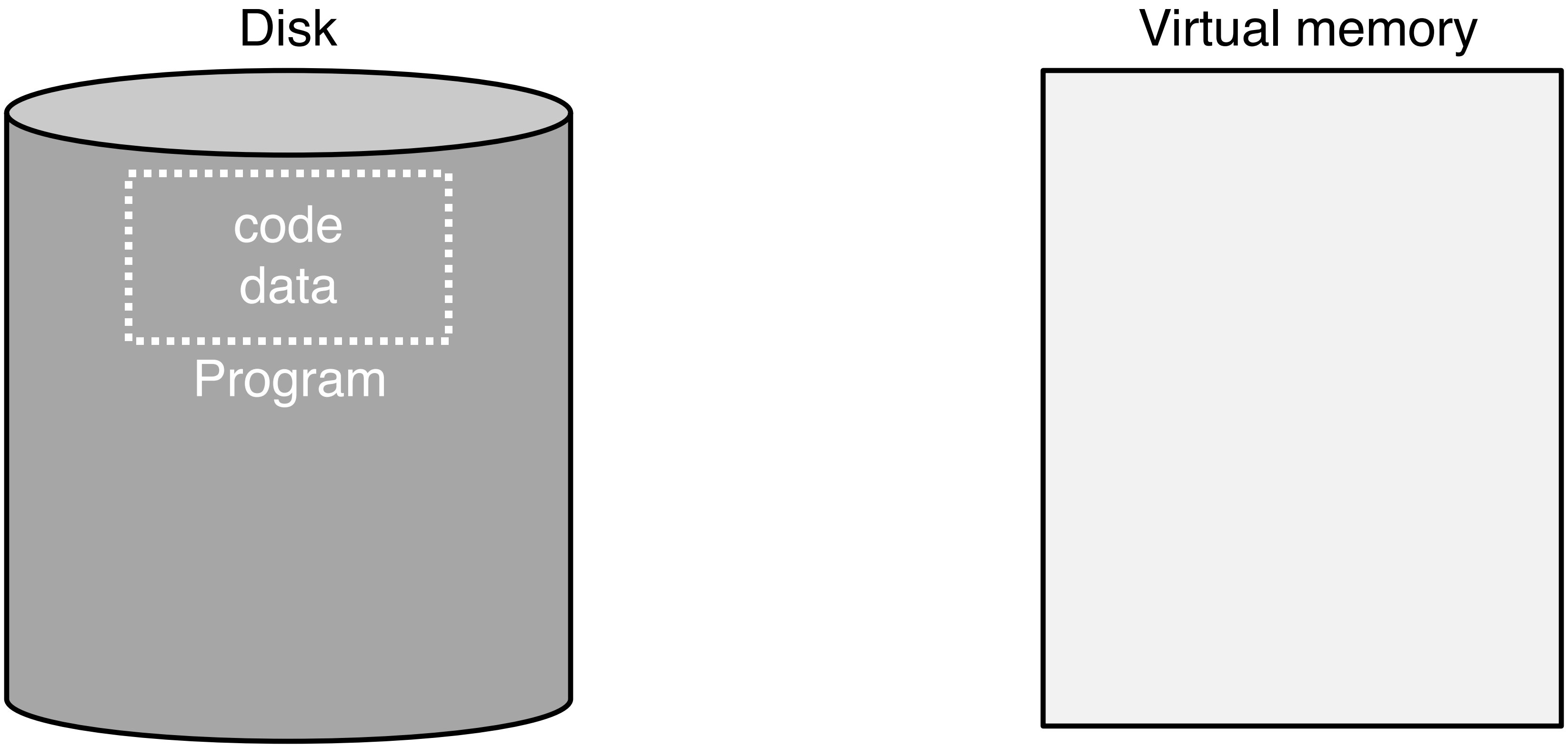
George Mason University

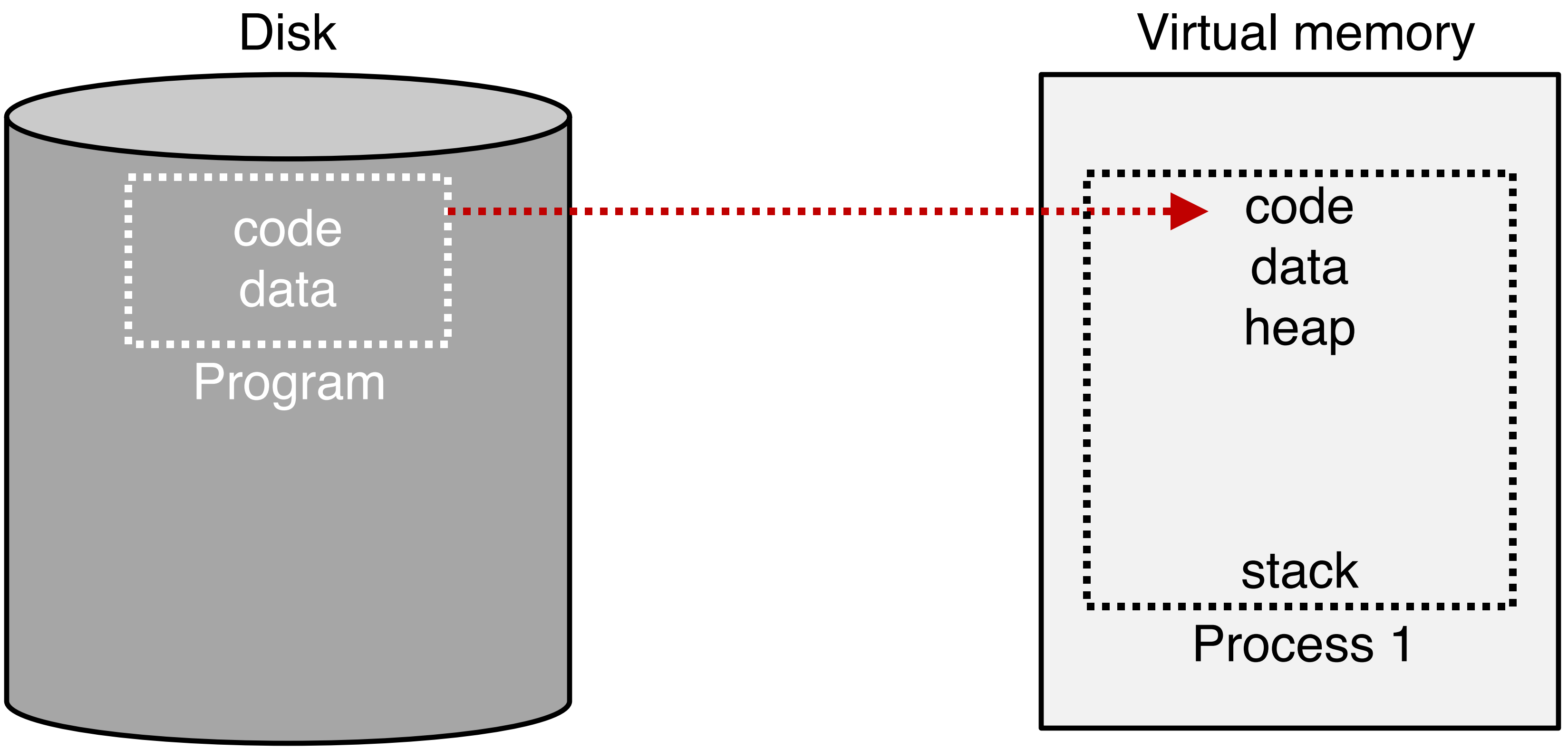
Fall 2019

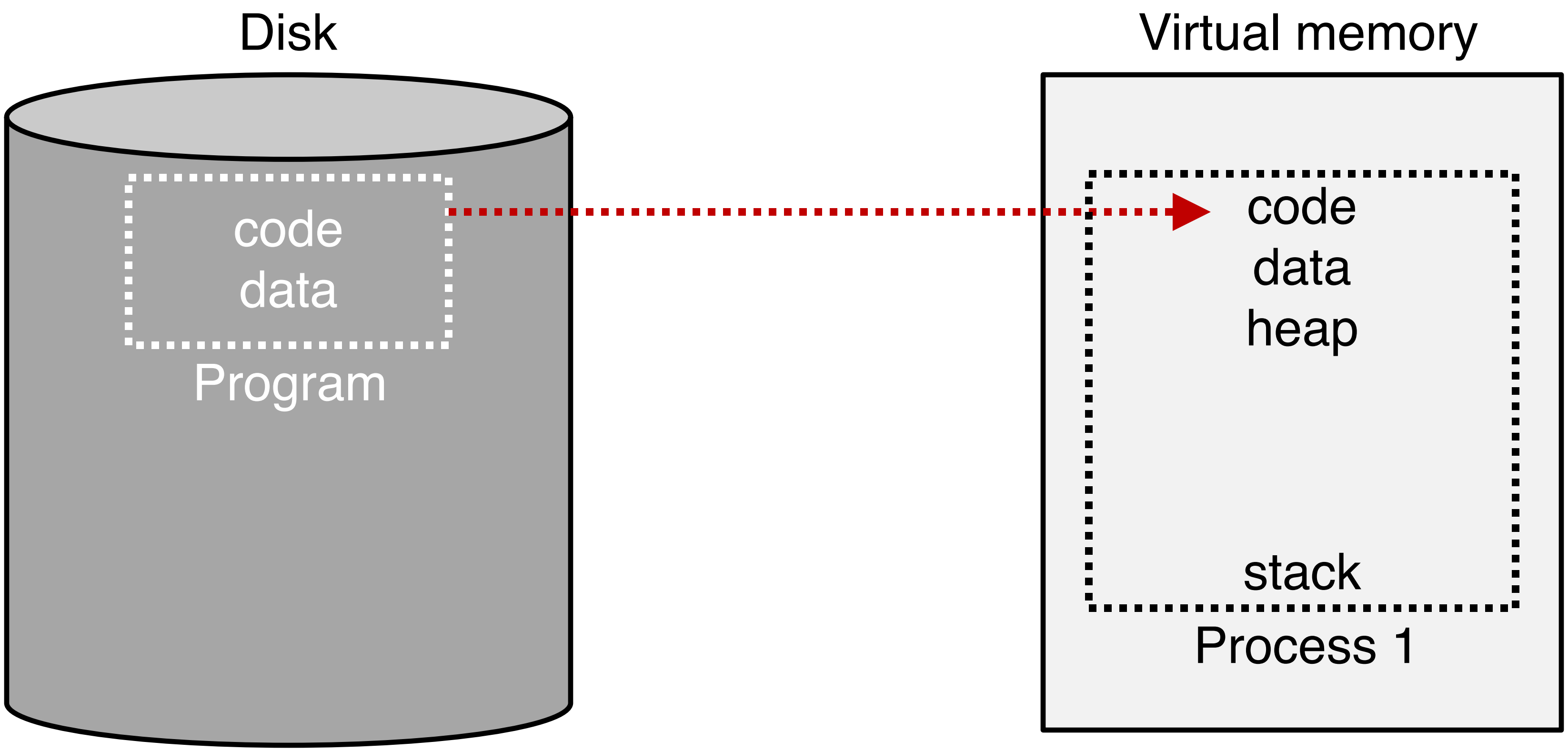
Announcement

- HW2 posted on BB after today's class
  - Due at 11:59pm Nov 1st (end of day next Friday)

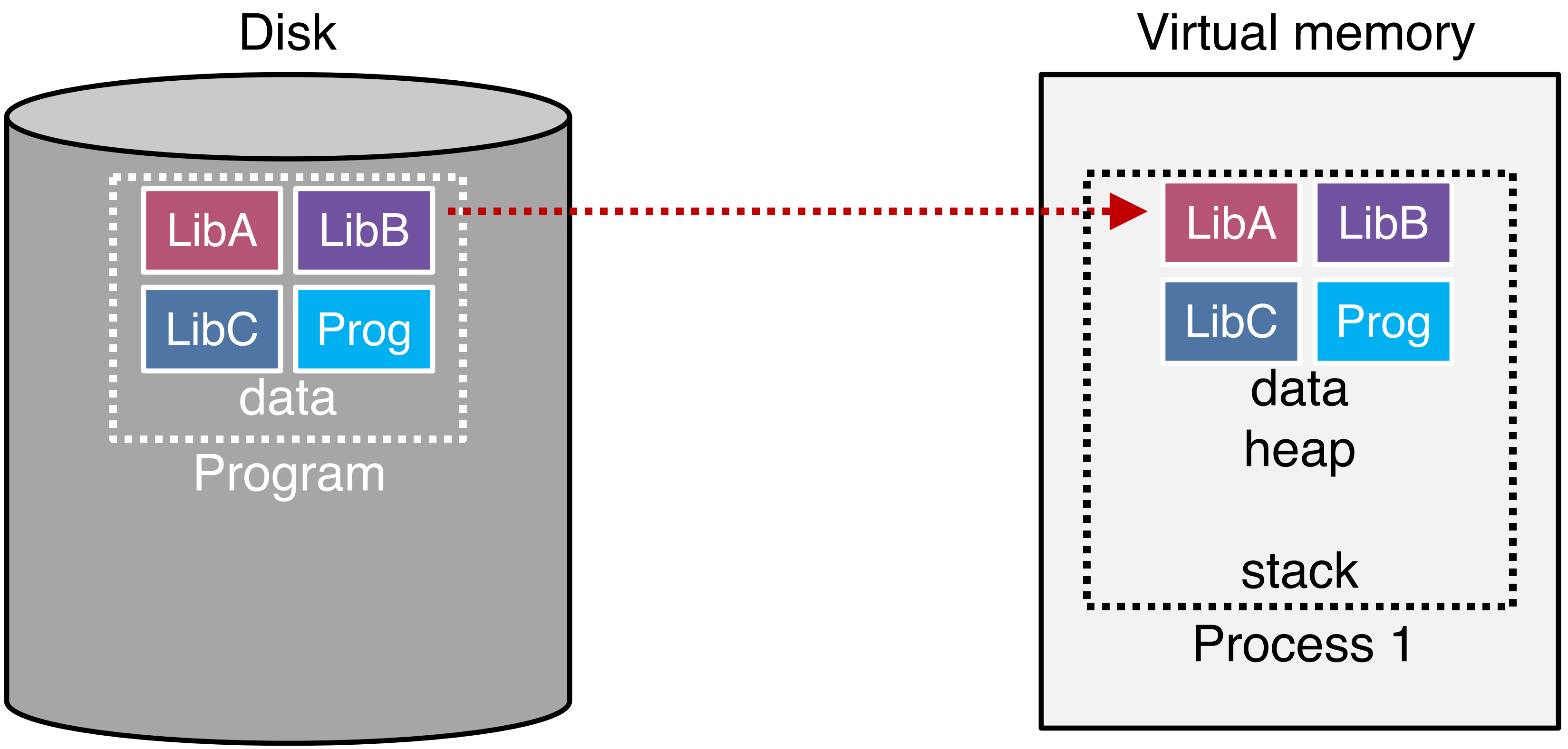
Swapping:
Beyond Physical Memory


Disk
code
data
Program
Virtual memory


Disk
code
data
Program
Virtual memory
code
data
heap
stack
Process 1

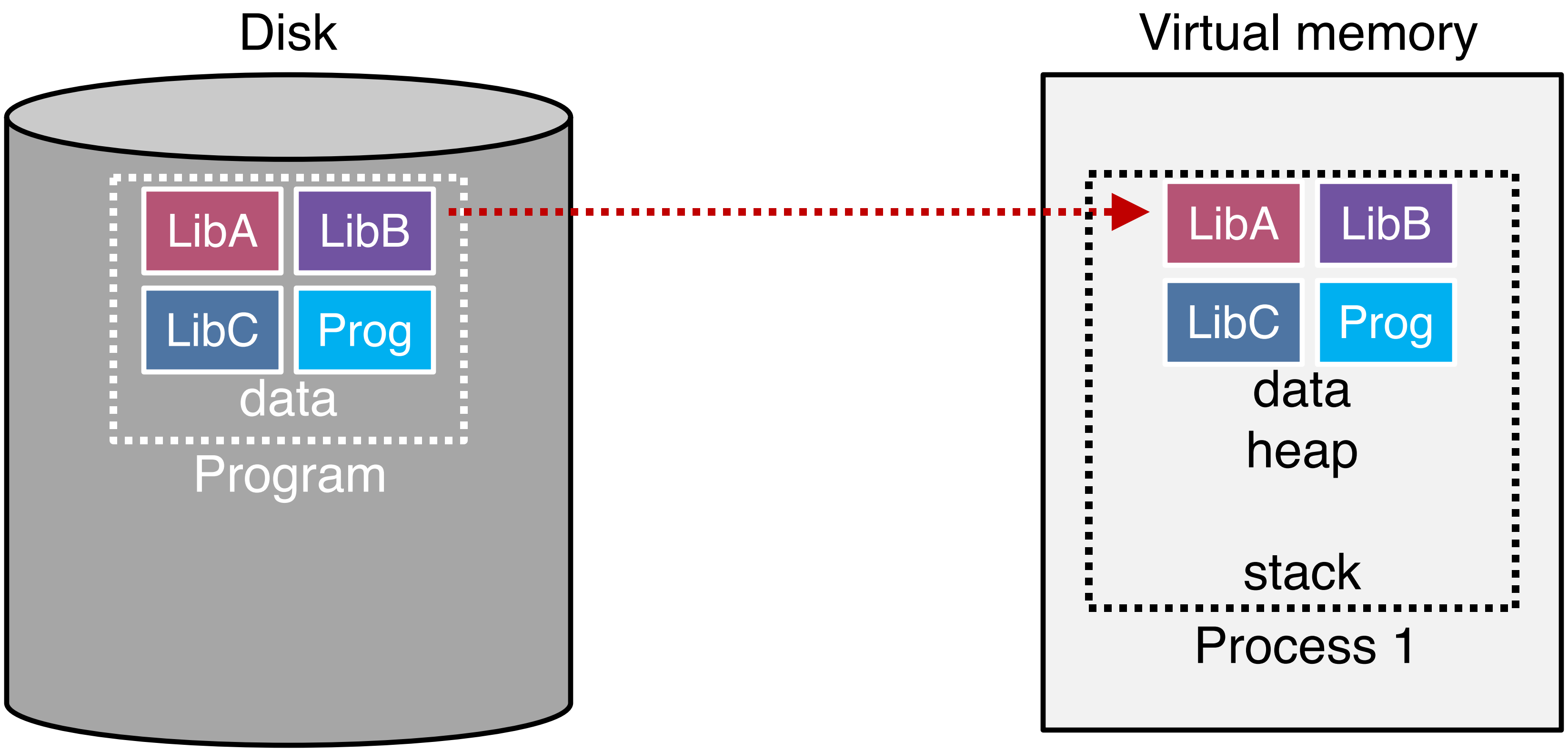


What's in code?

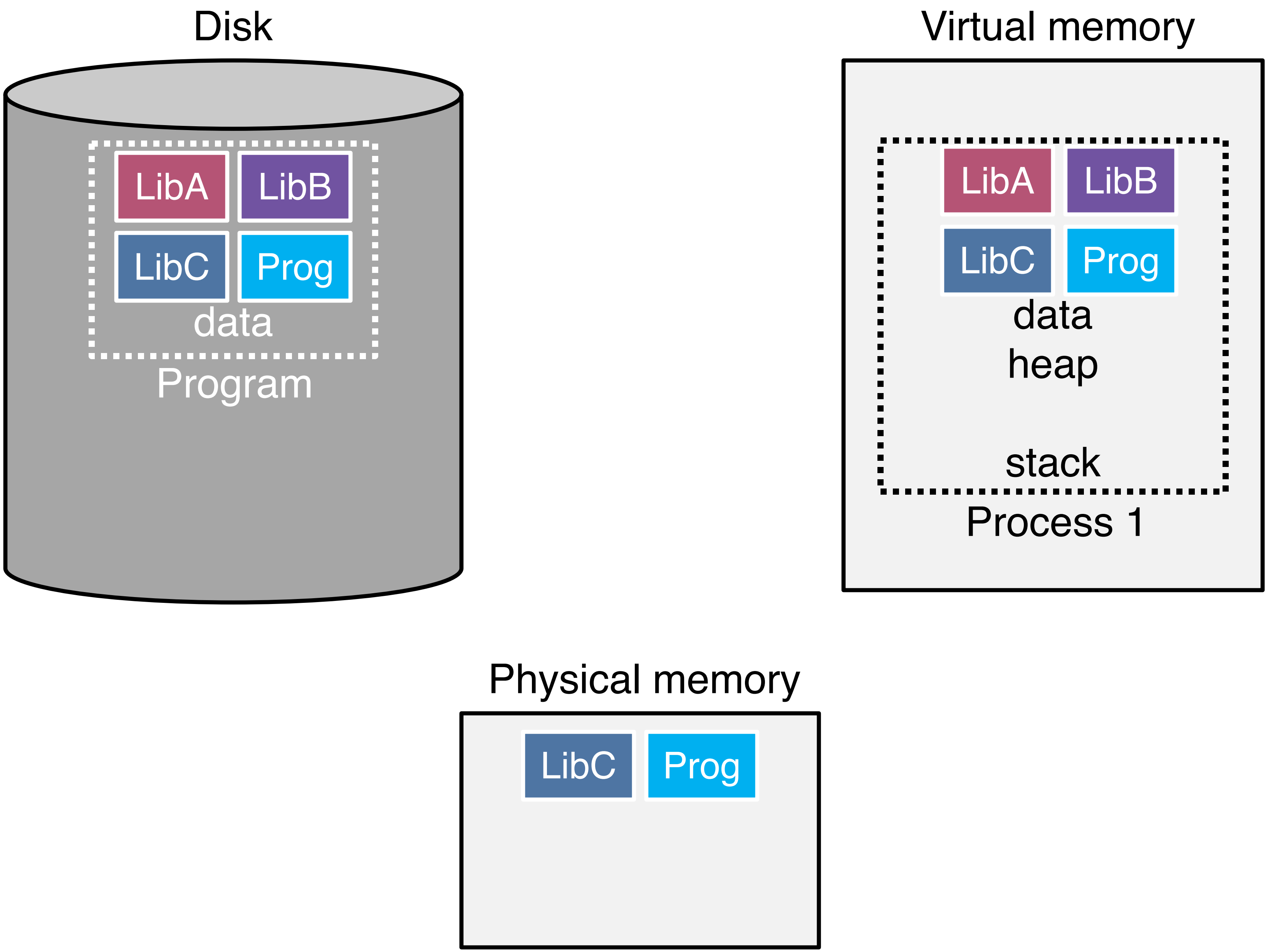


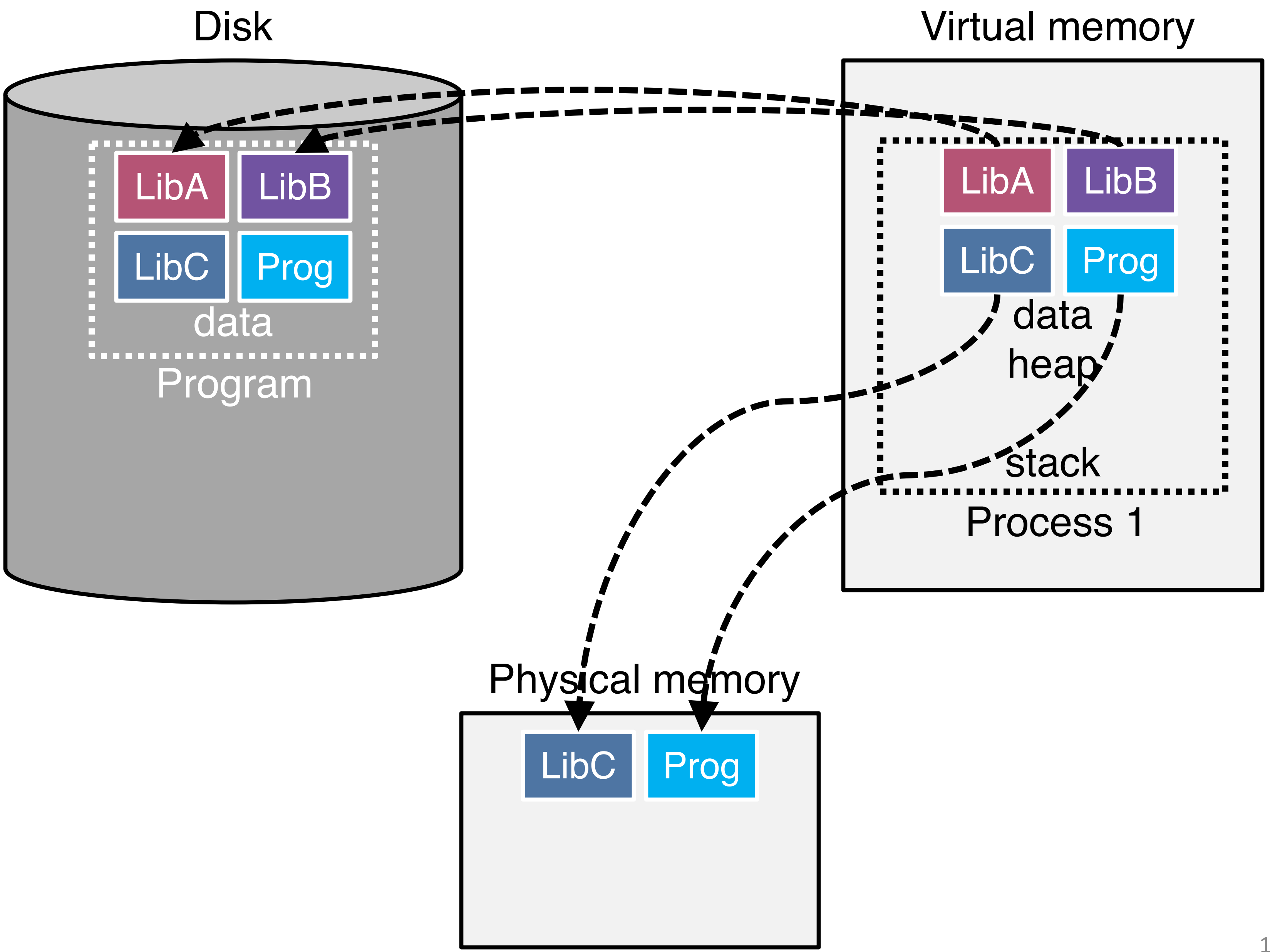
What's in code?

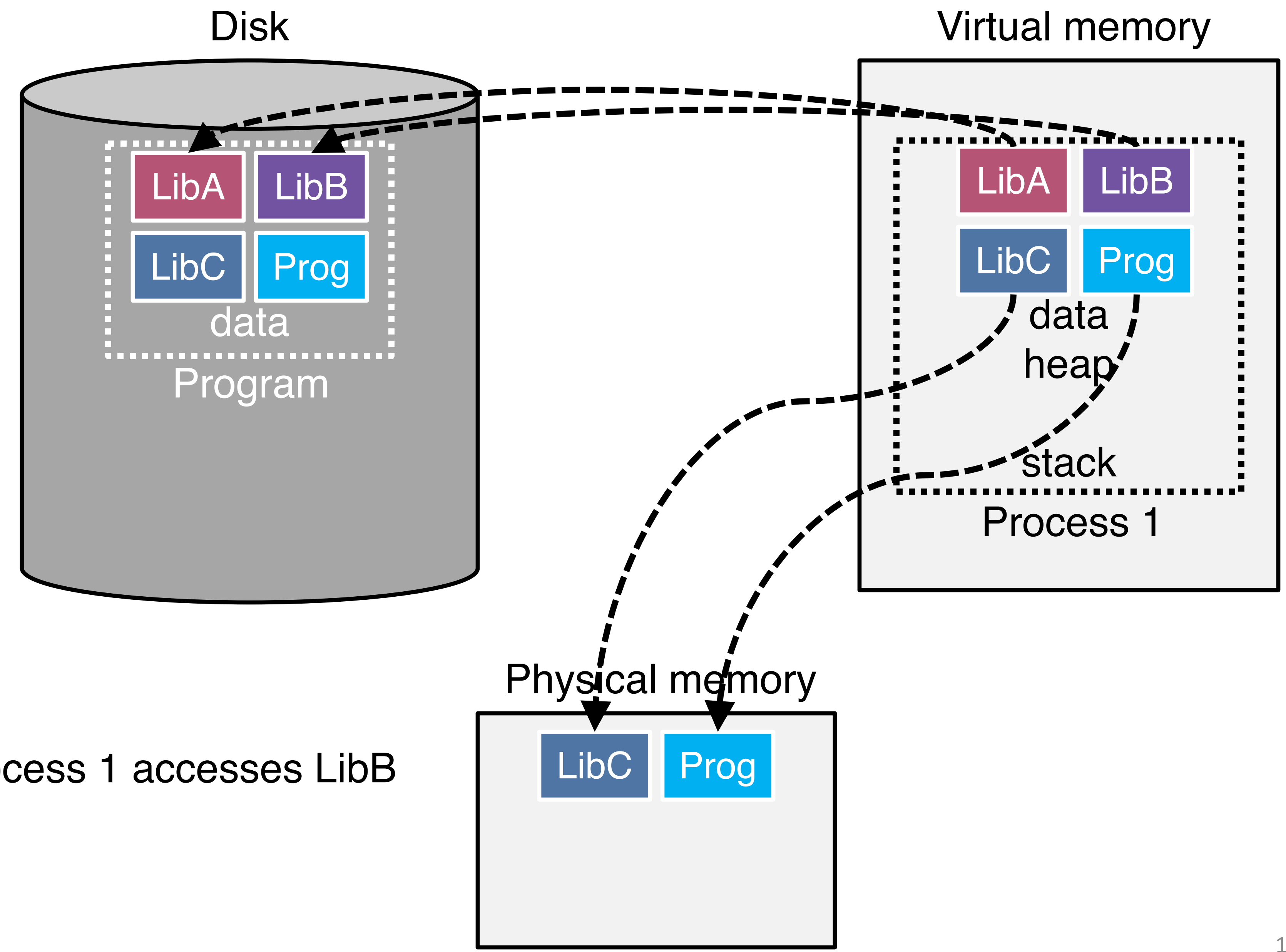
Many large libraries, some of which are rarely/never used



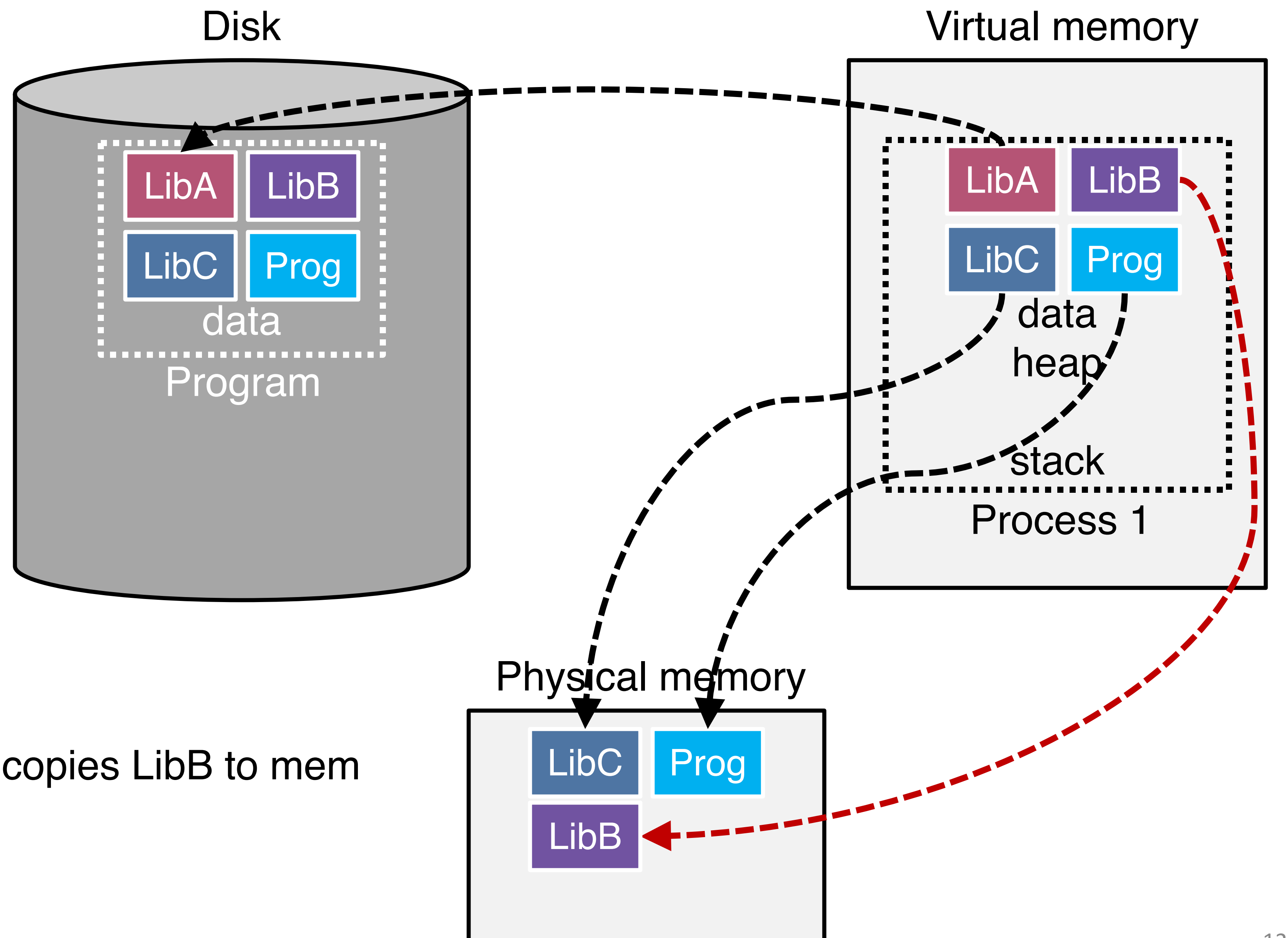
How to avoid wasting physical pages to back rarely used virtual pages?


Disk
LibA
LibB
LibC
Prog
data
Program
Virtual memory
LibA
LibB
LibC
Prog
data
heap
stack
Process 1
Physical memory
LibC
Prog

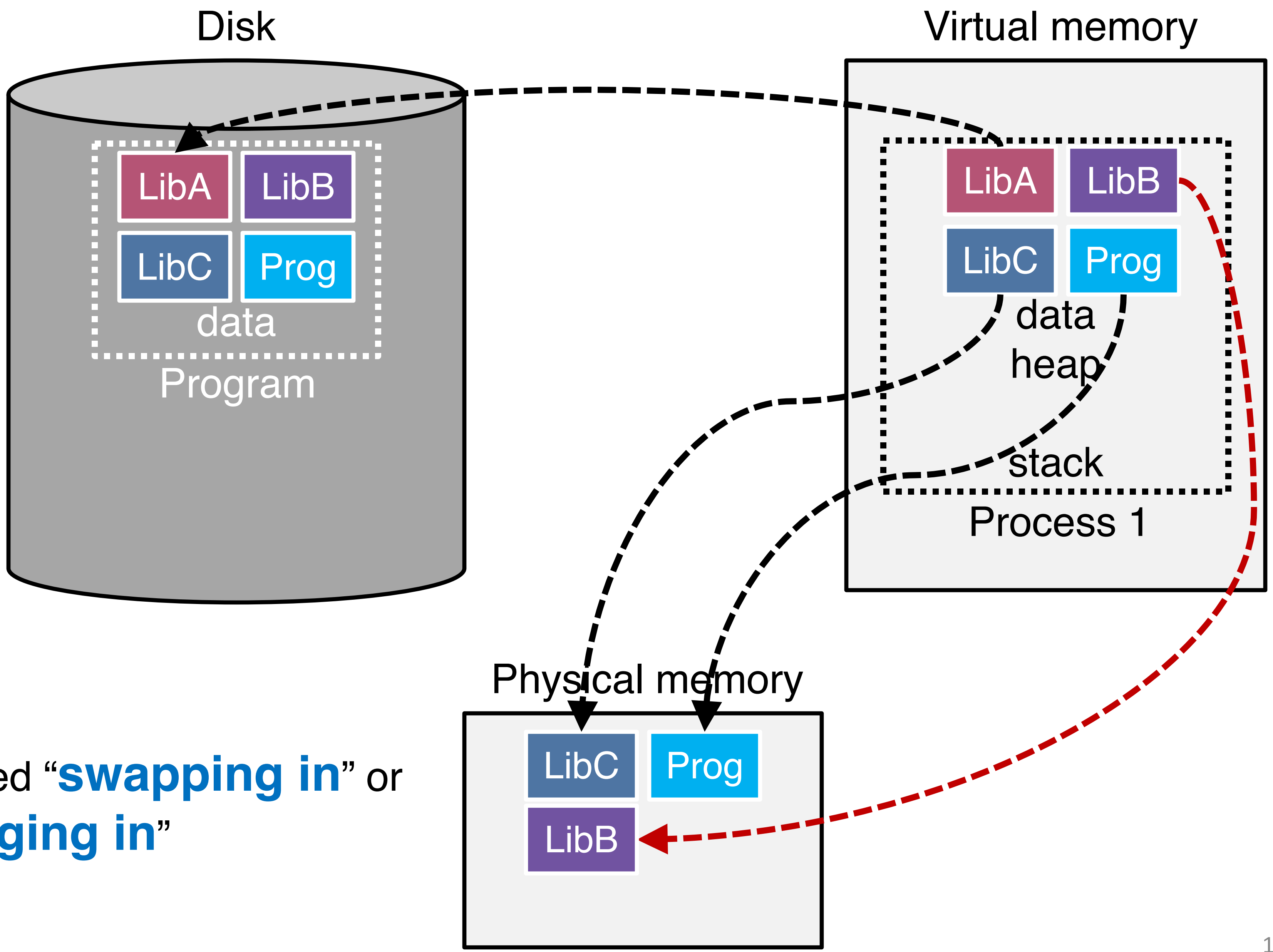

Disk
LibA
LibB
LibC
Prog
data
Program
Virtual memory
LibA
LibB
LibC
Prog
data
heap
stack
Process 1
Physical memory
LibC
Prog



Process 1 accesses LibB



OS copies LibB to mem



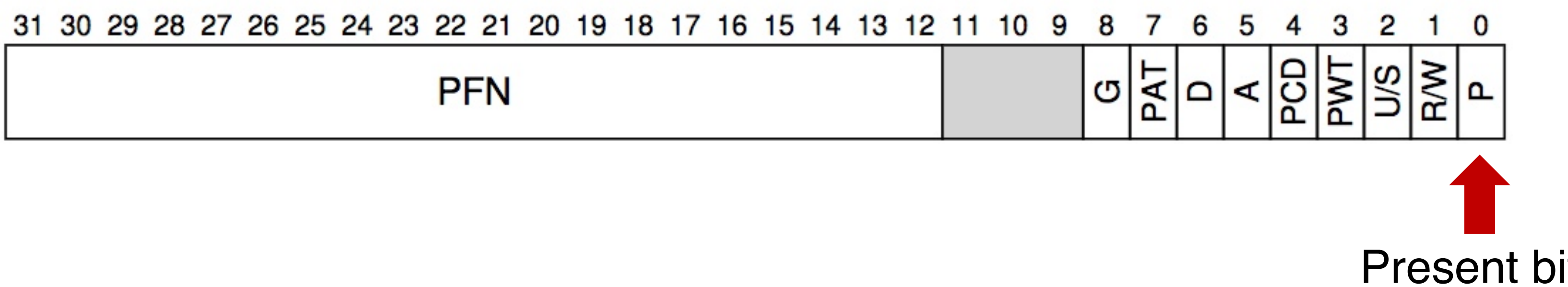
called “**swapping in**” or “**paging in**”

How to Know Where a Page Lives?

Present Bit

- With each PTE a present is associated
  - 1 ➔ in-memory, 0 ➔ out in disk

An 32-bit X86 page table entry (PTE)



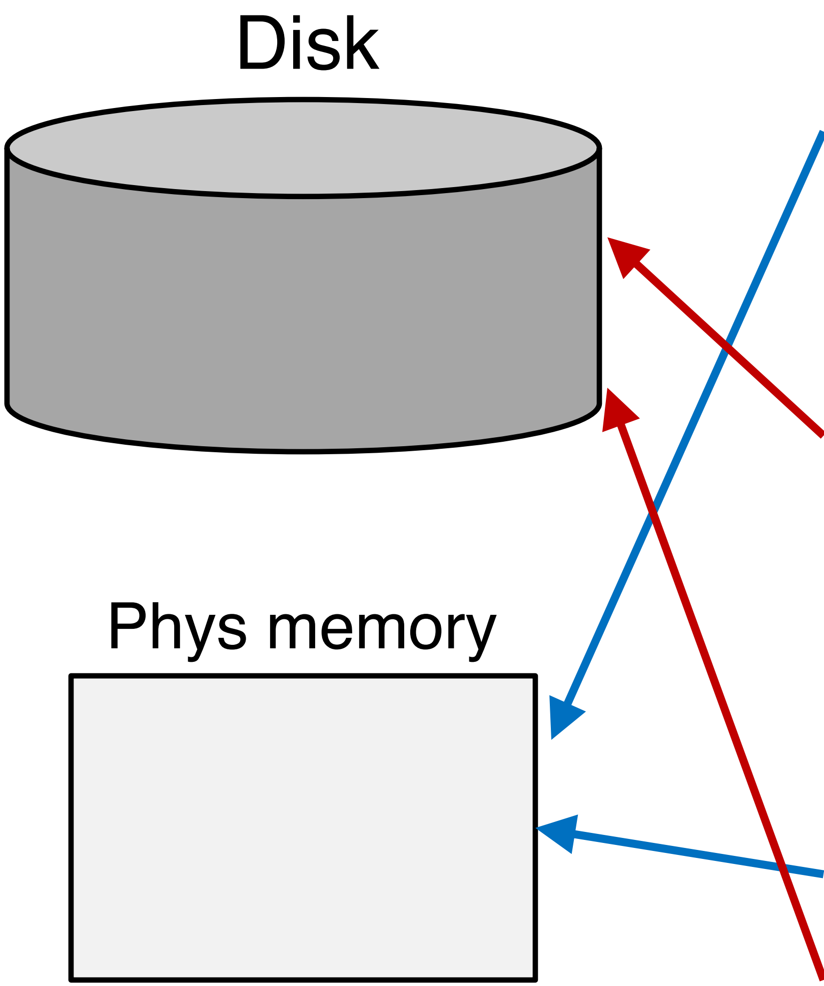
- During address translation, if present bit in PTE is 0 ➔ page fault

Present Bit

PFN	valid	prot	present
5	1	r-x	1
-	0	-	-
-	0	-	-
60	1	rw-	0
-	0	0	-
-	0	0	-
-	0	0	-
4	1	rw-	1
64	1	rw-	0

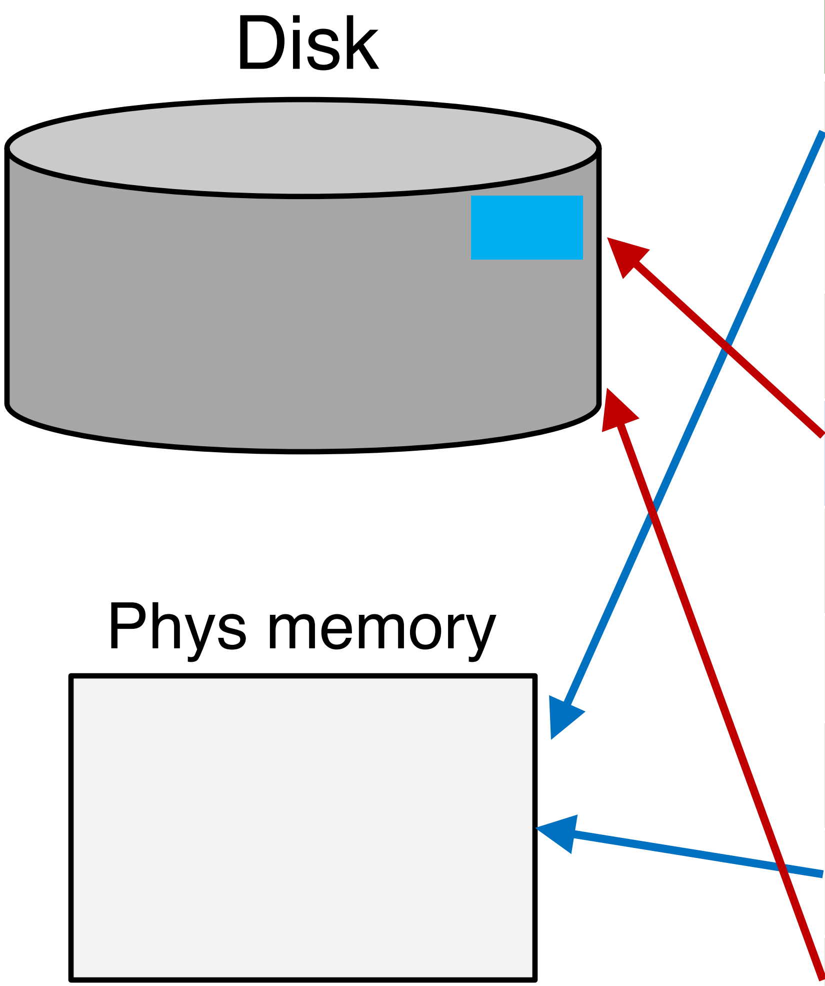
Page table

Present Bit



PFN	valid	prot	present
5	1	r-x	1
-	0	-	-
-	0	-	-
60	1	rw-	0
-	0	0	-
-	0	0	-
-	0	0	-
4	1	rw-	1
64	1	rw-	0

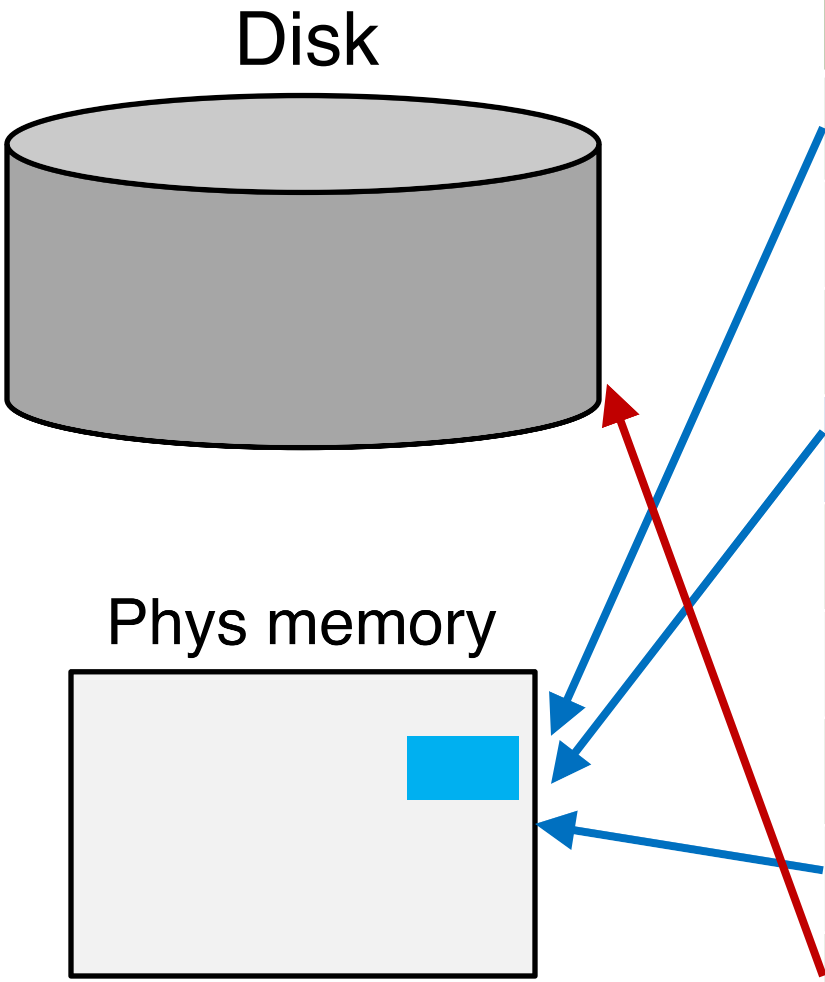
Present Bit



PFN	valid	prot	present
5	1	r-x	1
-	0	-	-
-	0	-	-
60	1	rw-	0
-	0	0	-
-	0	0	-
-	0	0	-
4	1	rw-	1
64	1	rw-	0

access

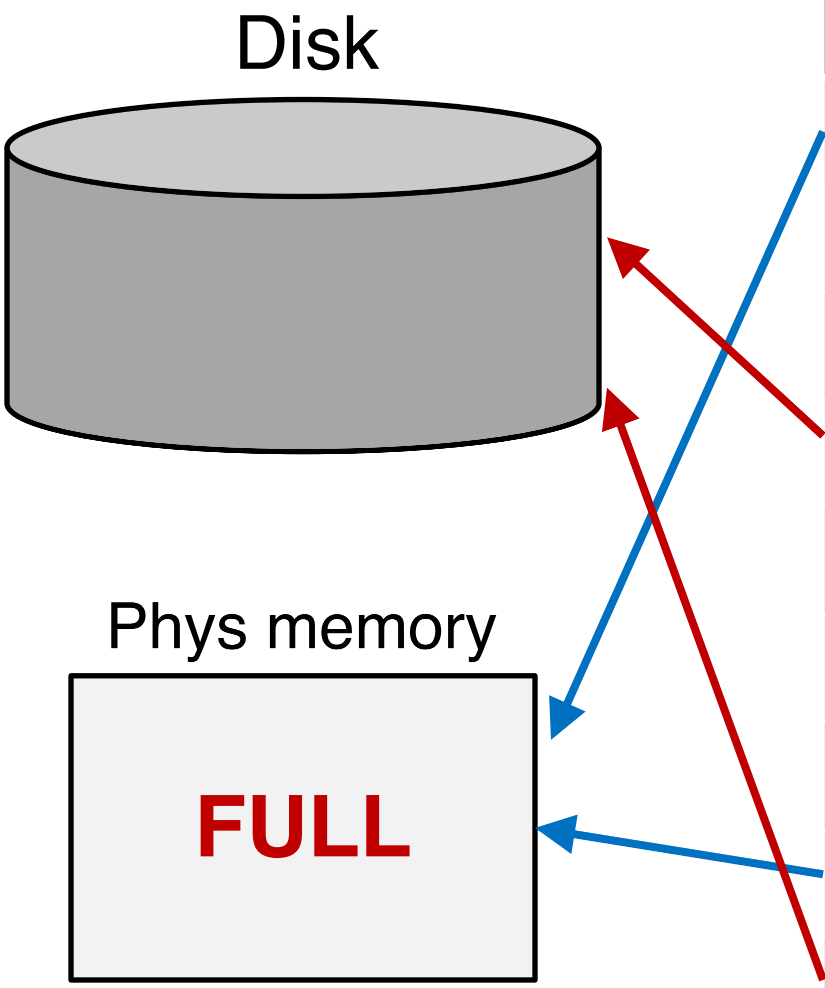
Present Bit



PFN	valid	prot	present
5	1	r-x	1
-	0	-	-
-	0	-	-
8	1	rw-	1 access
-	0	0	-
-	0	0	-
-	0	0	-
4	1	rw-	1
64	1	rw-	0

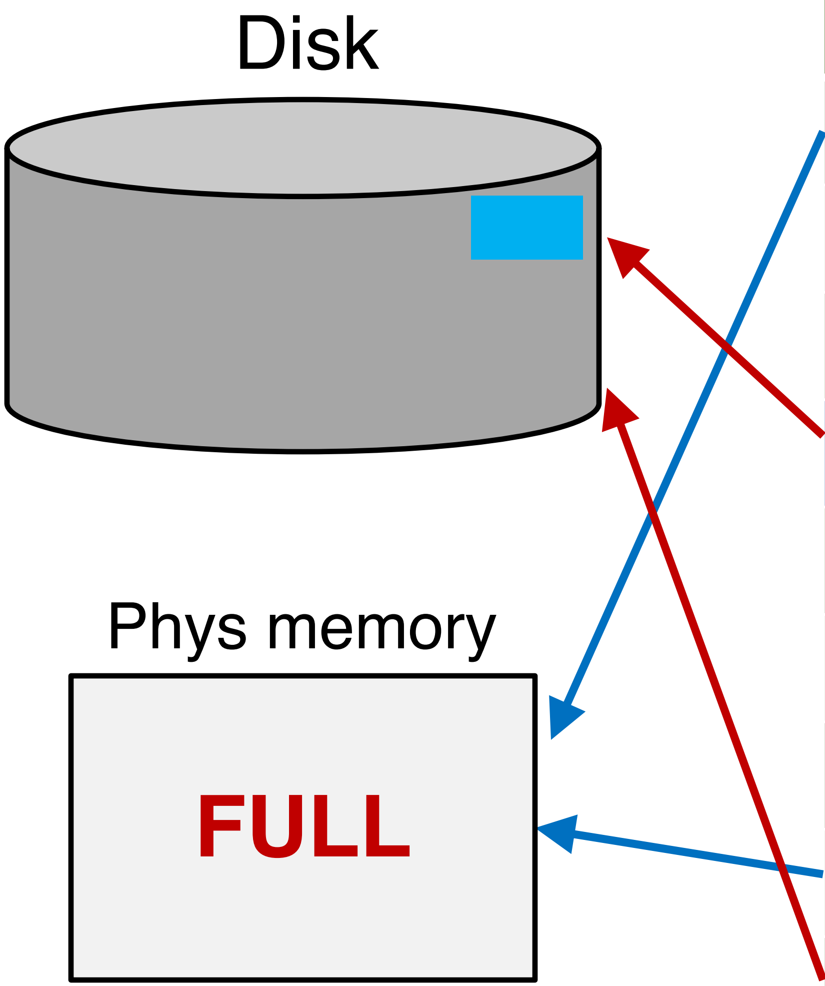
What if **NO** Memory is Left?

Present Bit



PFN	valid	prot	present
5	1	r-x	1
-	0	-	-
-	0	-	-
60	1	rw-	0
-	0	0	-
-	0	0	-
-	0	0	-
4	1	rw-	1
64	1	rw-	0

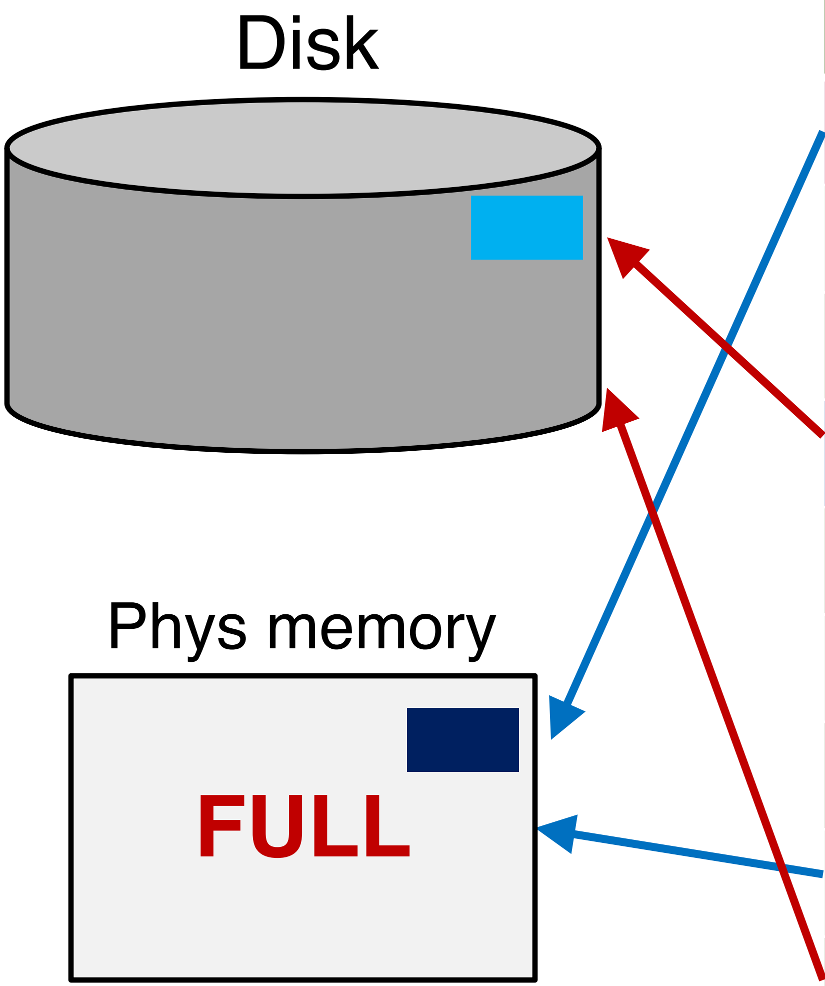
Present Bit



PFN	valid	prot	present
5	1	r-x	1
-	0	-	-
-	0	-	-
60	1	rw-	0
-	0	0	-
-	0	0	-
-	0	0	-
4	1	rw-	1
64	1	rw-	0

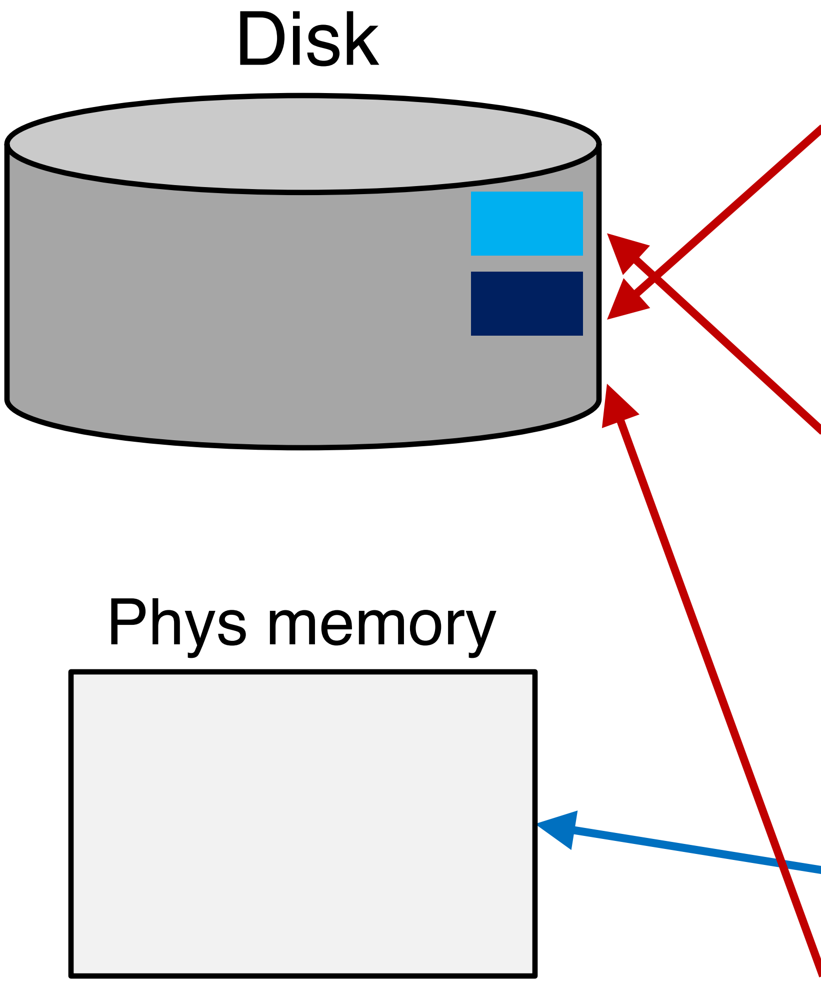
access

Present Bit



PFN	valid	prot	present	
5	1	r-x	1	evict
-	0	-	-	
-	0	-	-	
60	1	rw-	0	access
-	0	0	-	
-	0	0	-	
-	0	0	-	
4	1	rw-	1	
64	1	rw-	0	

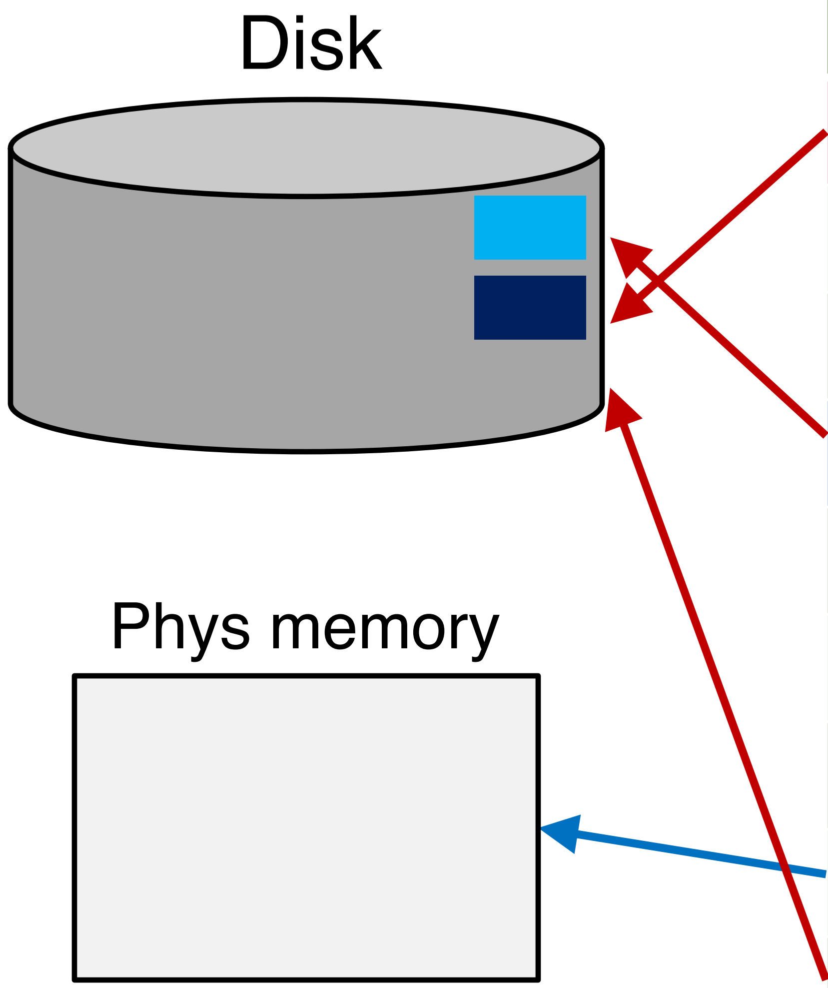
Present Bit



PFN	valid	prot	present	
63	1	r-x	0	evict
-	0	-	-	
-	0	-	-	
60	1	rw-	0	access
-	0	0	-	
-	0	0	-	
-	0	0	-	
4	1	rw-	1	
64	1	rw-	0	

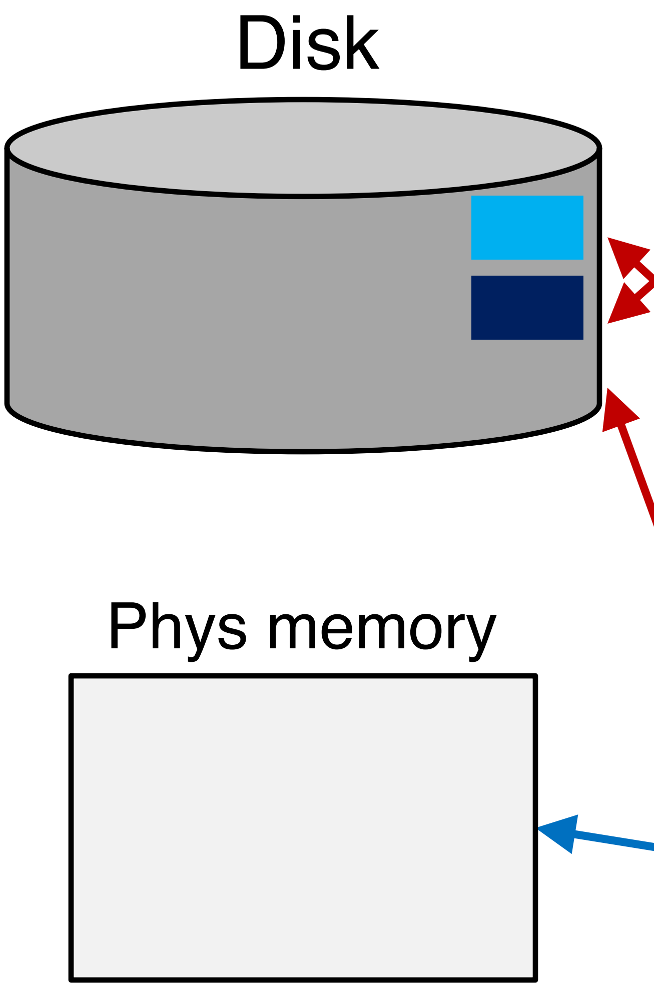
called "**swapping out**"
or "**paging out**"

Present Bit



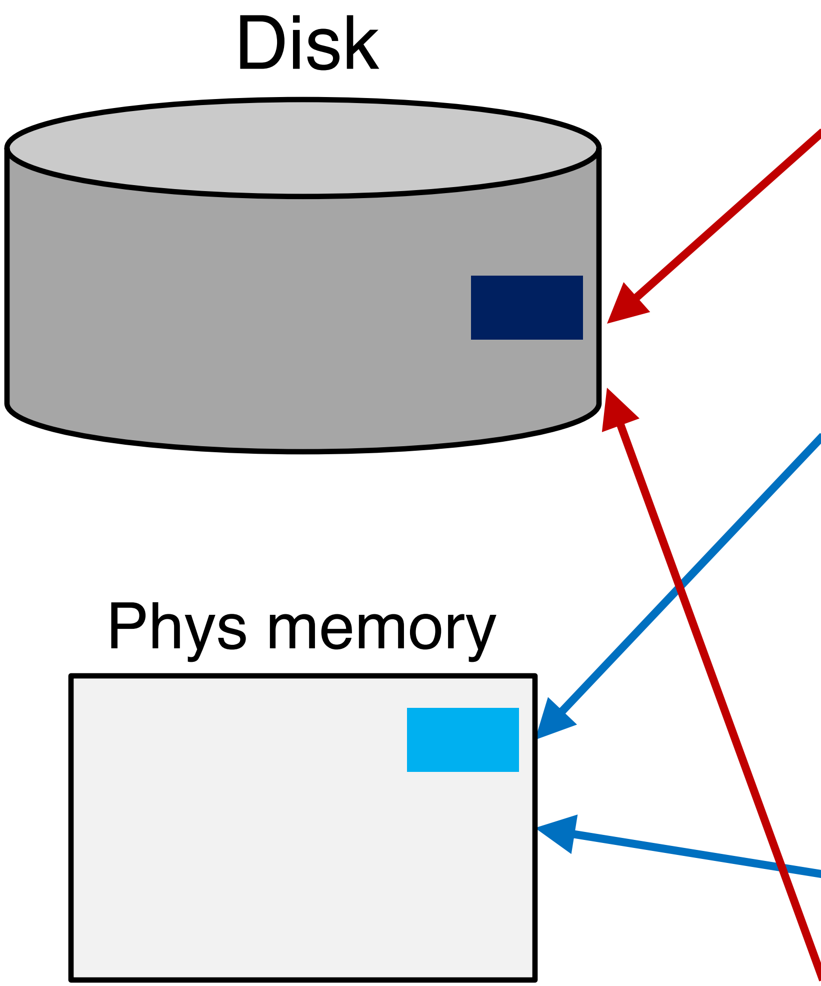
PFN	valid	prot	present	
63	1	r-x	0	**evict**
-	0	-	-	
-	0	-	-	
60	1	rw-	0	**access**
-	0	0	-	
-	0	0	-	
-	0	0	-	
4	1	rw-	1	
64	1	rw-	0	

Present Bit



PFN	valid	prot	present
63	1	r-x	0
-	0	-	-
-	0	-	-
60	1	rw-	0 access
-	0	0	-
-	0	0	-
-	0	0	-
4	1	rw-	1
64	1	rw-	0

Present Bit



PFN	valid	prot	present
63	1	r-x	0
-	0	-	-
-	0	-	-
5	1	rw-	1
-	0	0	-
-	0	0	-
-	0	0	-
4	1	rw-	1
64	1	rw-	0

access

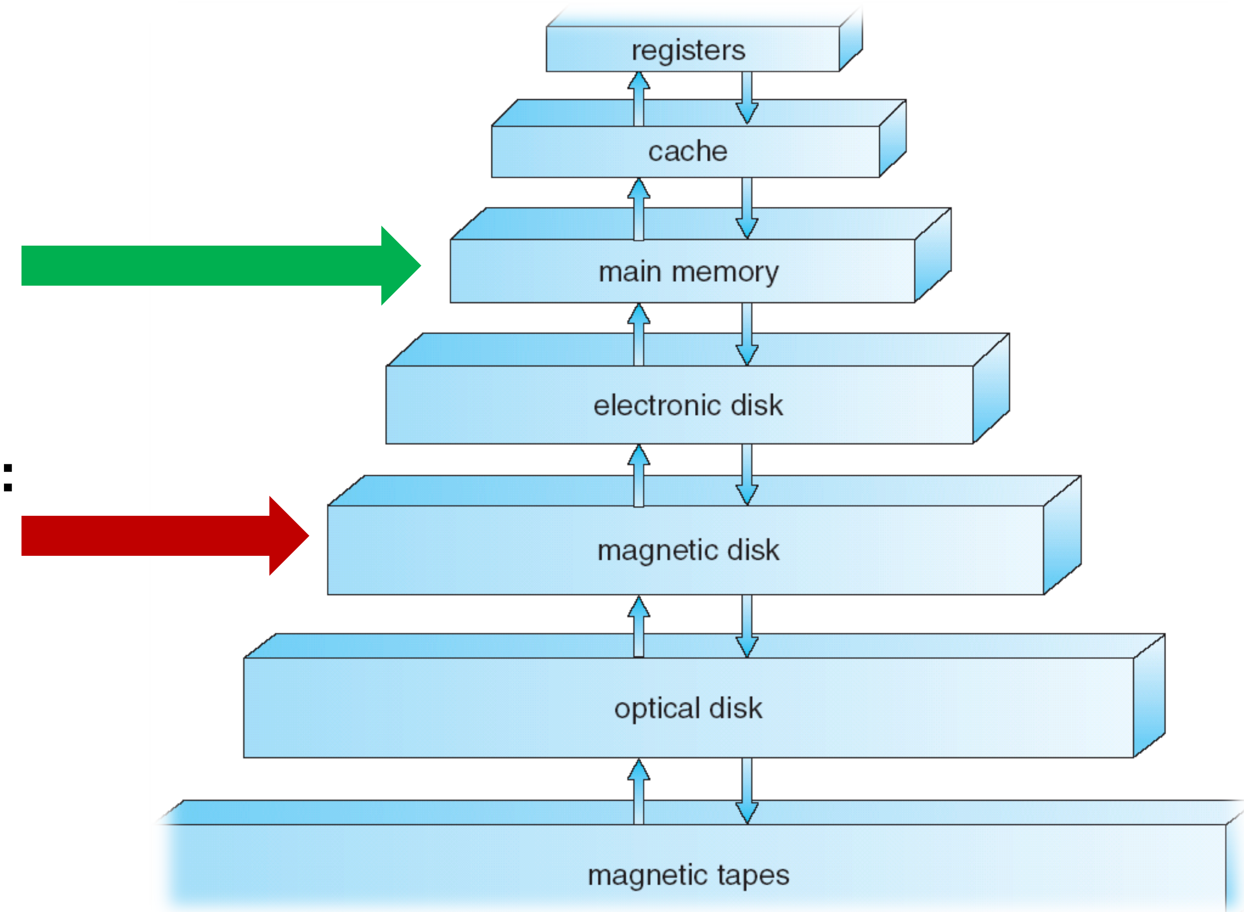
again, another "**swapping in**" or "**paging in**"

Why not Leave Page on Disk?

Storage Hierarchy

Main memory:
Smaller capacity
Faster accesses

Secondary storage:
Larger capacity
Way slower accesses



Why not Leave Page on Disk?

- Performance: Memory vs. Disk

- How long does it take to access a 4-byte `int` from main memory vs. disk?
  - DRAM: ~100ns
  - Disk: ~10ms

Beyond the Physical Memory

- Idea: use the disk space as an extension of main memory

- Two ways of interaction b/w memory and disk
  - Demand paging
  - Swapping

Demand Paging

- Bring a page into memory only when it is needed (demanded)
  - Less I/O needed
  - Less memory needed
  - Faster response
  - Support more processes/users

- Page is needed $\Rightarrow$ use the reference to page
  - If not in memory $\Rightarrow$ must bring from the disk

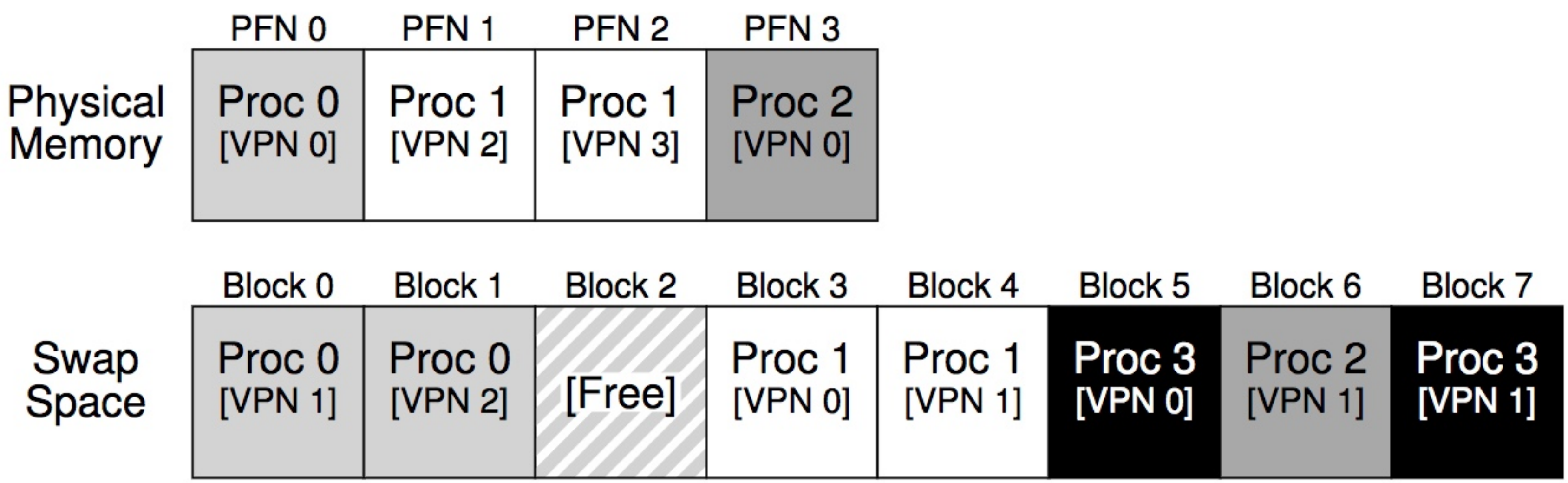
Swapping

- Swapping allows OS to support the illusion of a large virtual memory for multiprogramming
  - Multiple programs can run "**at once**"
  - Better utilization
  - Ease of use

- Demand paging vs. swapping
  - On demand vs. page replacement under memory pressure

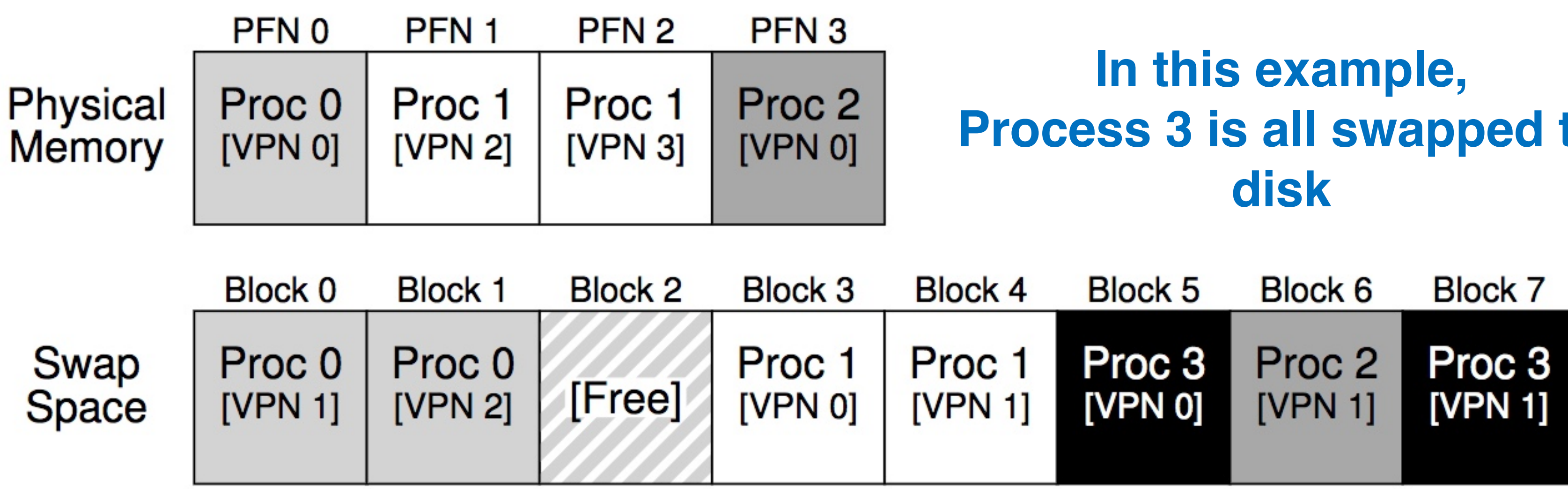
Swapping

- Swapping allows OS to support the illusion of a large virtual memory for multiprogramming
  - Multiple programs can run “**at once**”
  - Better utilization
  - Ease of use



Swap Space

- Part of disk space reserved for moving pages back and forth
  - Swap pages out of memory
  - Swap pages into memory from disk
- OS reads from and writes to the swap space at page-sized unit



In this example, Process 3 is all swapped to disk

Address Translation Steps

- Hardware: for each memory reference:

  Extract **VPN** from **VA**

  Check **TLB** for **VPN**

  TLB hit:

  - Build **PA** from **PFN** and offset
  - Fetch **PA** from memory

  TLB miss:

  - Fetch **PTE**
  - if (!valid): exception [segfault]
  - else if (!present): exception [page fault: page miss]
  - else: extract **PFN**, insert in **TLB**, retry

- Q: Which steps are expensive??

Address Translation Steps

- Hardware: for each memory reference:

(cheap) Extract **VPN** from **VA**
(cheap) Check **TLB** for **VPN**
TLB hit:
(cheap) Build **PA** from **PFN** and offset
(expensive) Fetch **PA** from memory
TLB miss:
(expensive) Fetch **PTE**
(expensive) if (!valid): exception [segfault]
(expensive) else if (!present): exception [page fault: page miss]
(cheap) else: extract **PFN**, insert in **TLB**, retry

- Q: Which steps are expensive??

Page Fault

- The act of accessing a page that is not in physical memory is called a **page fault**
- OS is invoked to service the page fault
  - Page fault handler
- Typically, **PTE** contains the page address on disk

Page-Fault Handler (OS)

```
PFN = FindFreePage()
if (PFN == -1)
        PFN = EvictPage()
DiskRead(PTE.DiskAddr, PFN)
PTE.present = 1
PTE.PFN = PFN
retry instruction
```

Page-Fault Handler (OS)

```
PFN = FindFreePage()
if (PFN == -1)
        PFN = EvictPage()
DiskRead(PTE.DiskAddr, PFN)
PTE.present = 1
PTE.PFN = PFN
retry instruction
```

Q: which steps are expensive?

Page-Fault Handler (OS)

```
(cheap)      PFN = FindFreePage()
(cheap)      if (PFN == -1)
(depends)        PFN = EvictPage()
(expensive)  DiskRead(PTE.DiskAddr, PFN)
(cheap)      PTE.present = 1
(cheap)      PTE.PFN = PFN
(cheap)      retry instruction
```

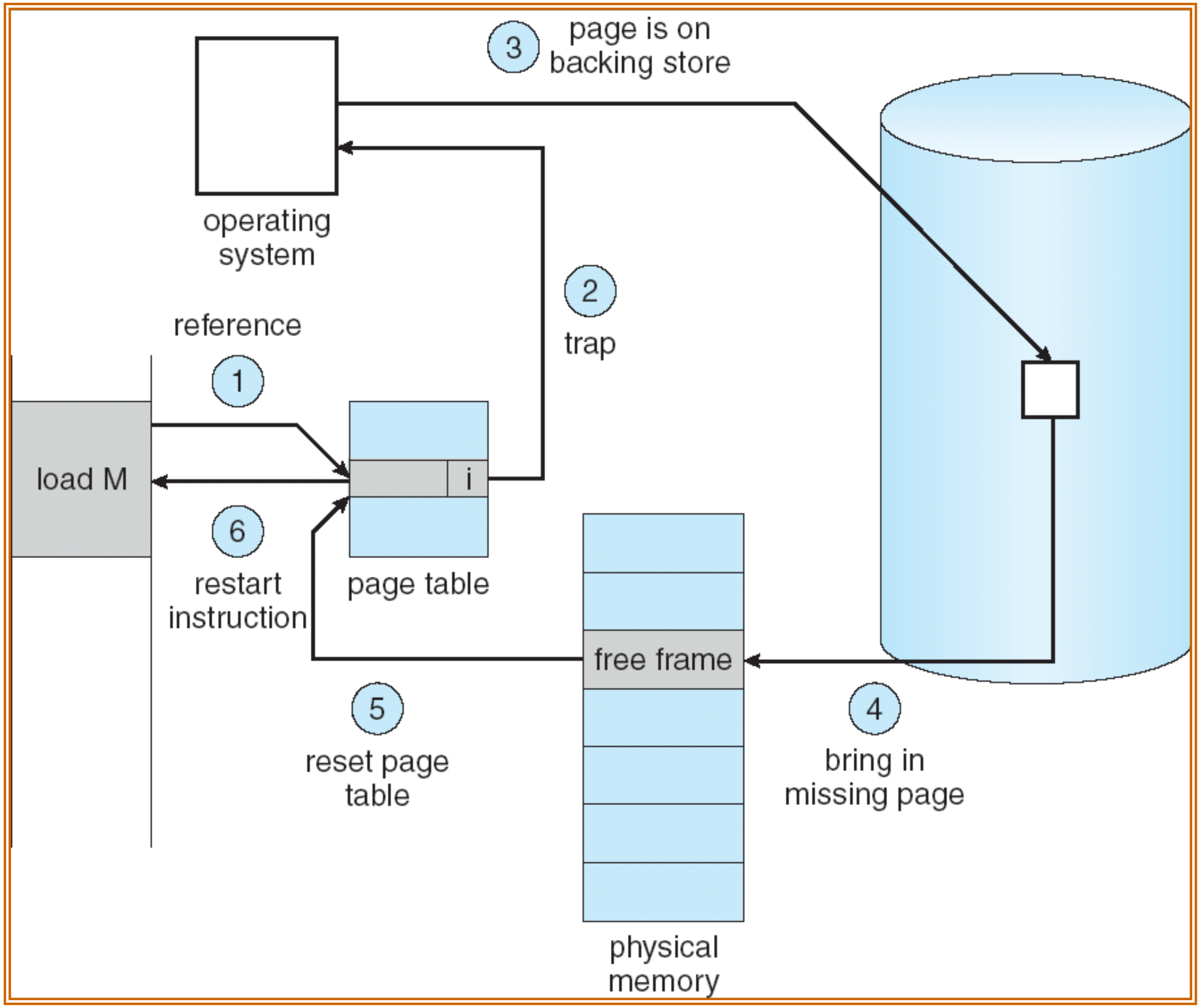
Q: which steps are expensive?

Page-Fault Handler (OS)

```
(cheap)      PFN = FindFreePage()
(cheap)      if (PFN == -1)
(depends)        PFN = EvictPage()
(expensive)  DiskRead(PTE.DiskAddr, PFN)
(cheap)      PTE.present = 1
(cheap)      PTE.PFN = PFN
(cheap)      retry instruction
```

What to evict?
What to read?

Major Steps of A Page Fault



Impact of Page Faults

- Each page fault affects the system performance negatively
  - The process experiencing the page fault will not be able to continue until the missing page is brought to the main memory
  - The process will be blocked (moved to the waiting state)
  - Dealing with the page fault involves disk I/O
    - Increased demand to the disk drive
    - Increased waiting time for process experiencing page fault

Memory as a Cache

- As we increase the degree of multiprogramming, over-allocation of memory becomes a problem
- What if we are unable to find a free frame at the time of the page fault?
- OS chooses to page out one or more pages to make room for new page(s) OS is about to bring in
  - The process to replace page(s) is called **page replacement policy**

Memory as a Cache

- OS keeps a small portion of memory free proactively
  - **High watermark** (HW) and **low watermark** (LW)

- When OS notices free memory is below LW (i.e., memory pressure)
  - A background thread (i.e., swap/page daemon) starts running to free memory
  - It evicts pages until there are **HW** pages available