[537] Beyond Physical Memory

Chapters 21-22
Tyler Harter
9/29/14

Problem 1: PT Size

page directory

PFN	valid
0101	1
1111	1
-	0
0110	1

PFN: 0101

PFN	valid
0000	1
1010	1
-	0
-	0

PFN: 0110

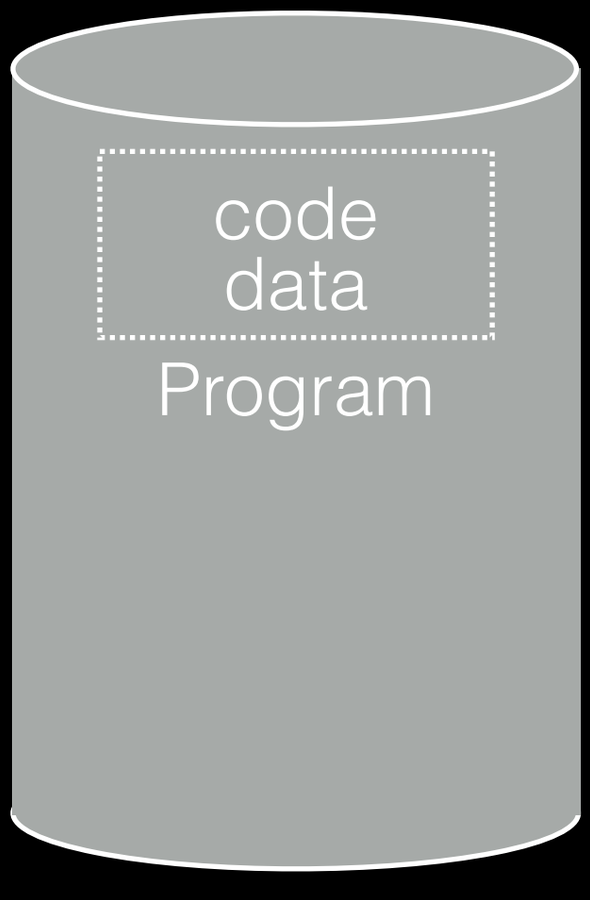
PFN	valid
1001	1
-	0
-	0
-	0

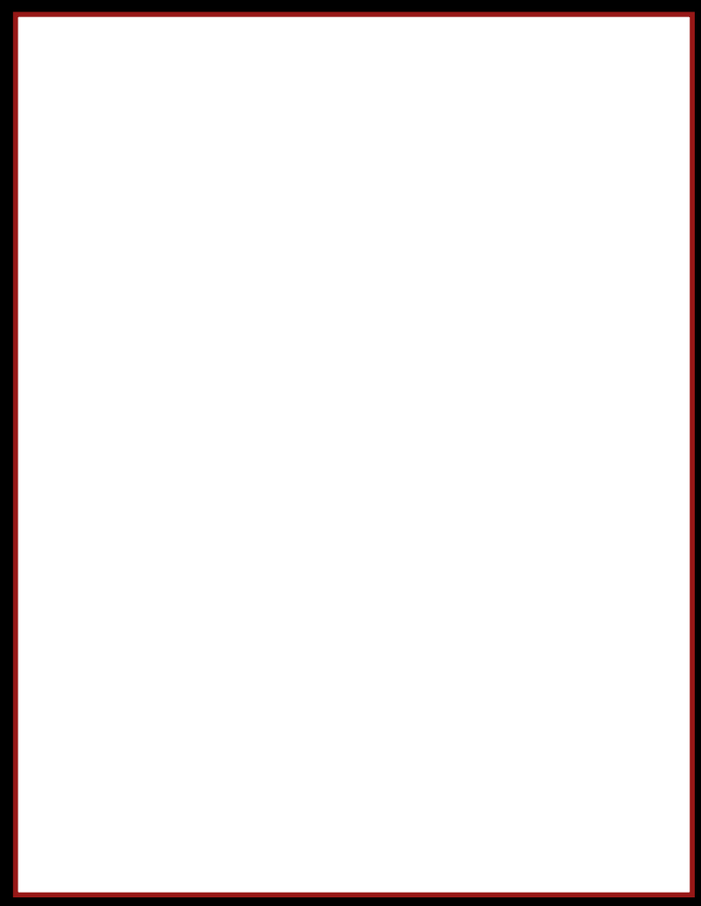
PFN:1111

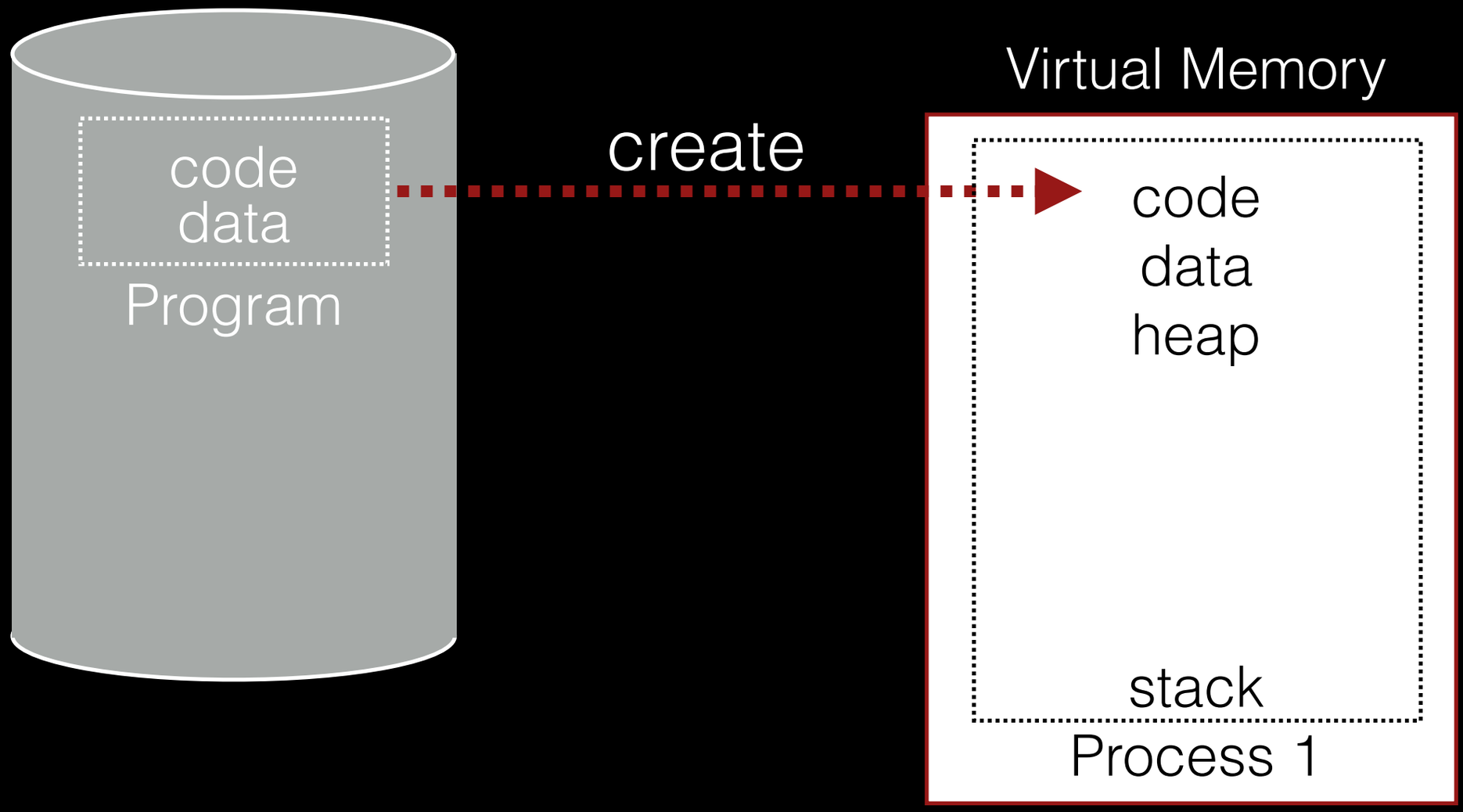
PFN	valid
-	0
-	0
-	0
1011	1

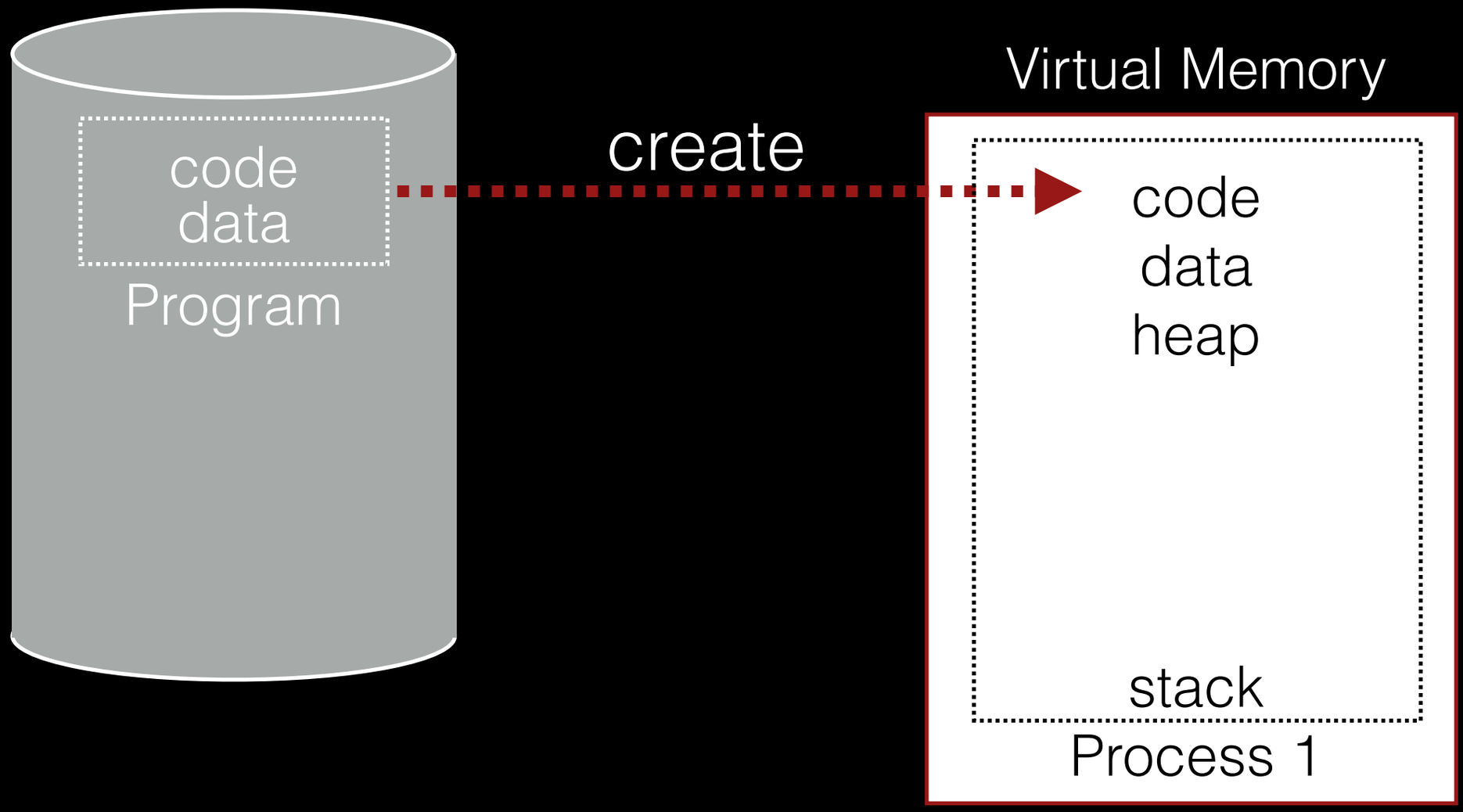
Problem 2
(worksheet)

assume 12-bit
virtual addrs

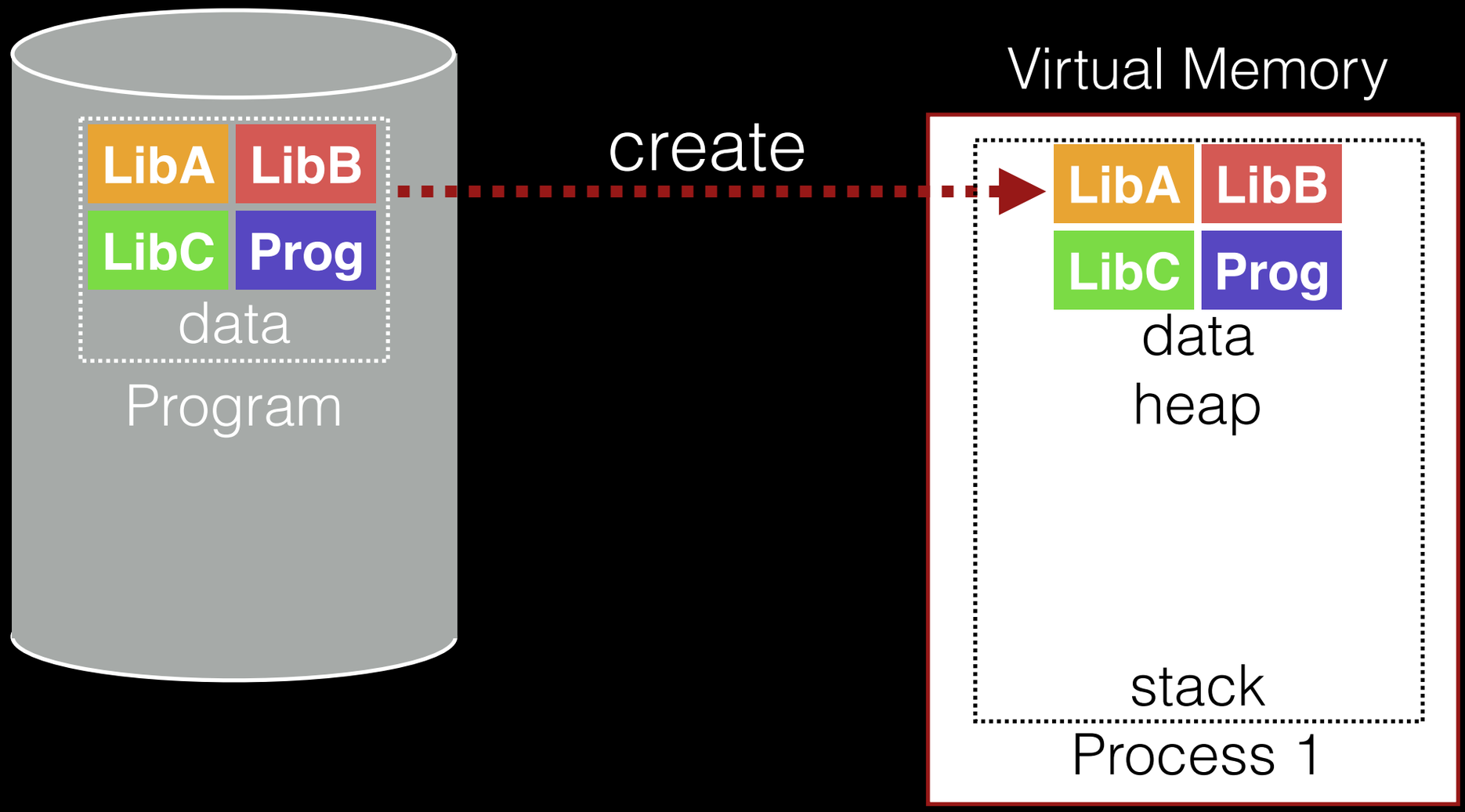

code
data
Program


Virtual Memory


code
data
Program
create
Virtual Memory
code
data
heap
stack
Process 1

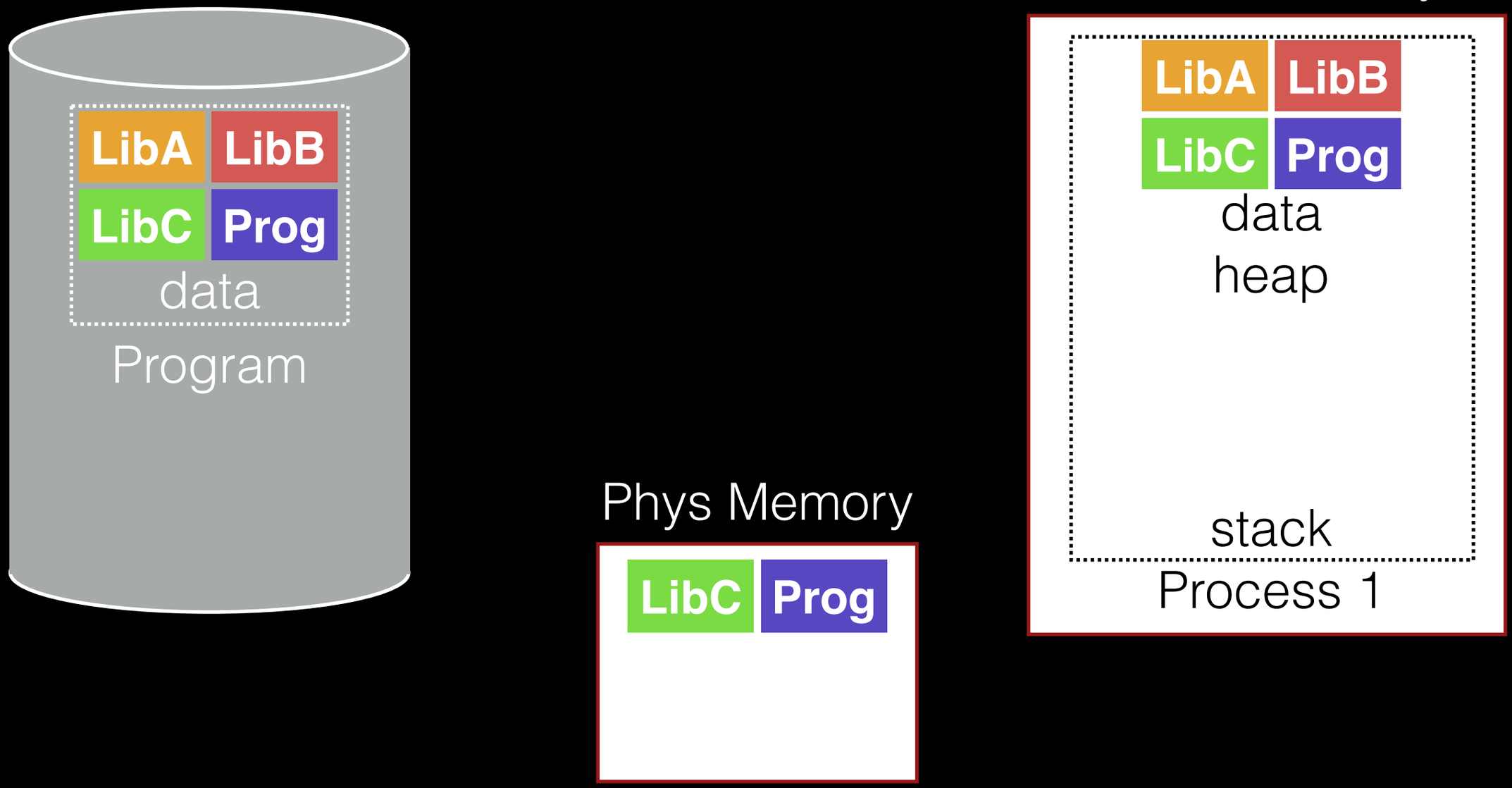


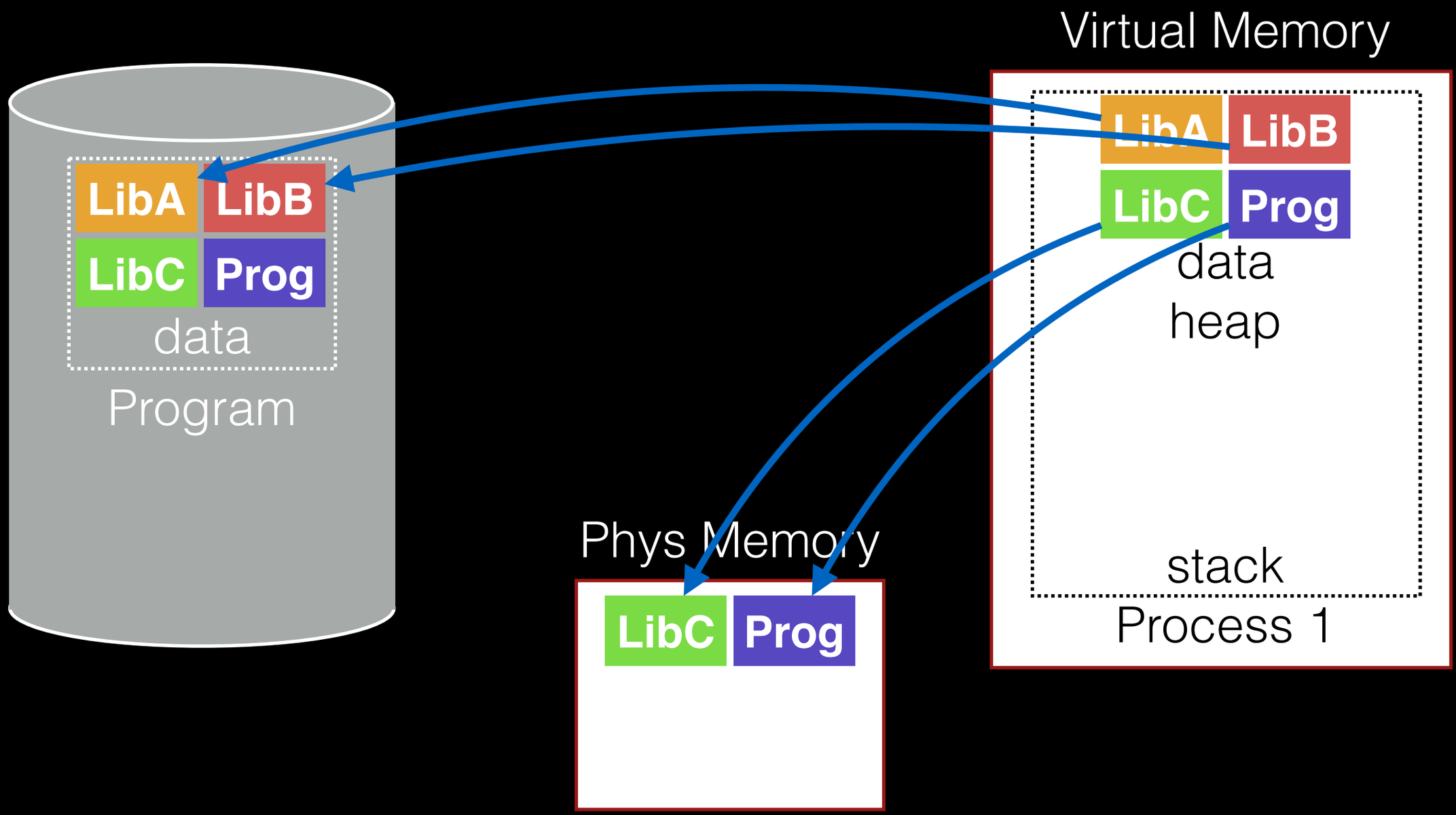
what's in code?

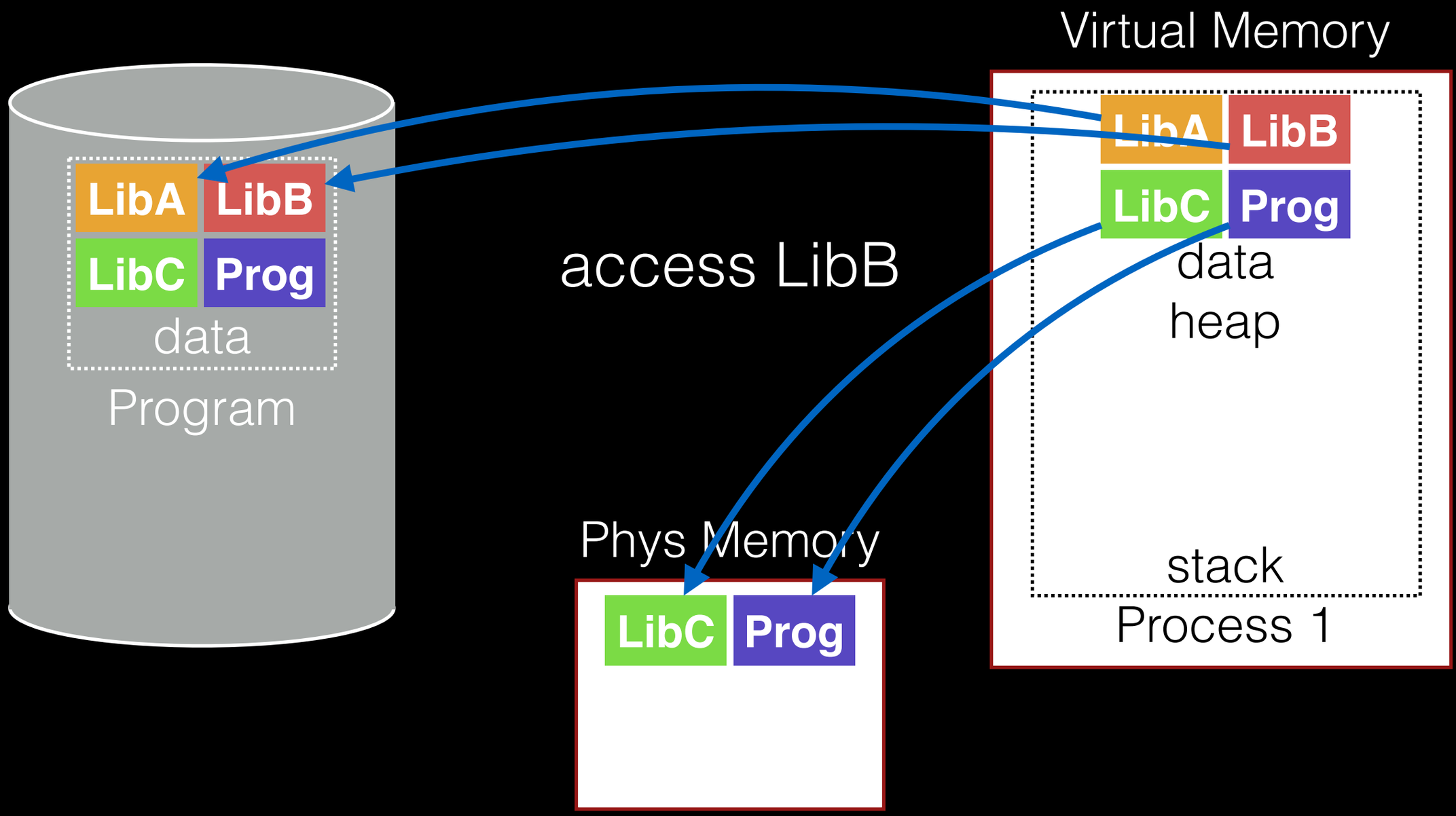


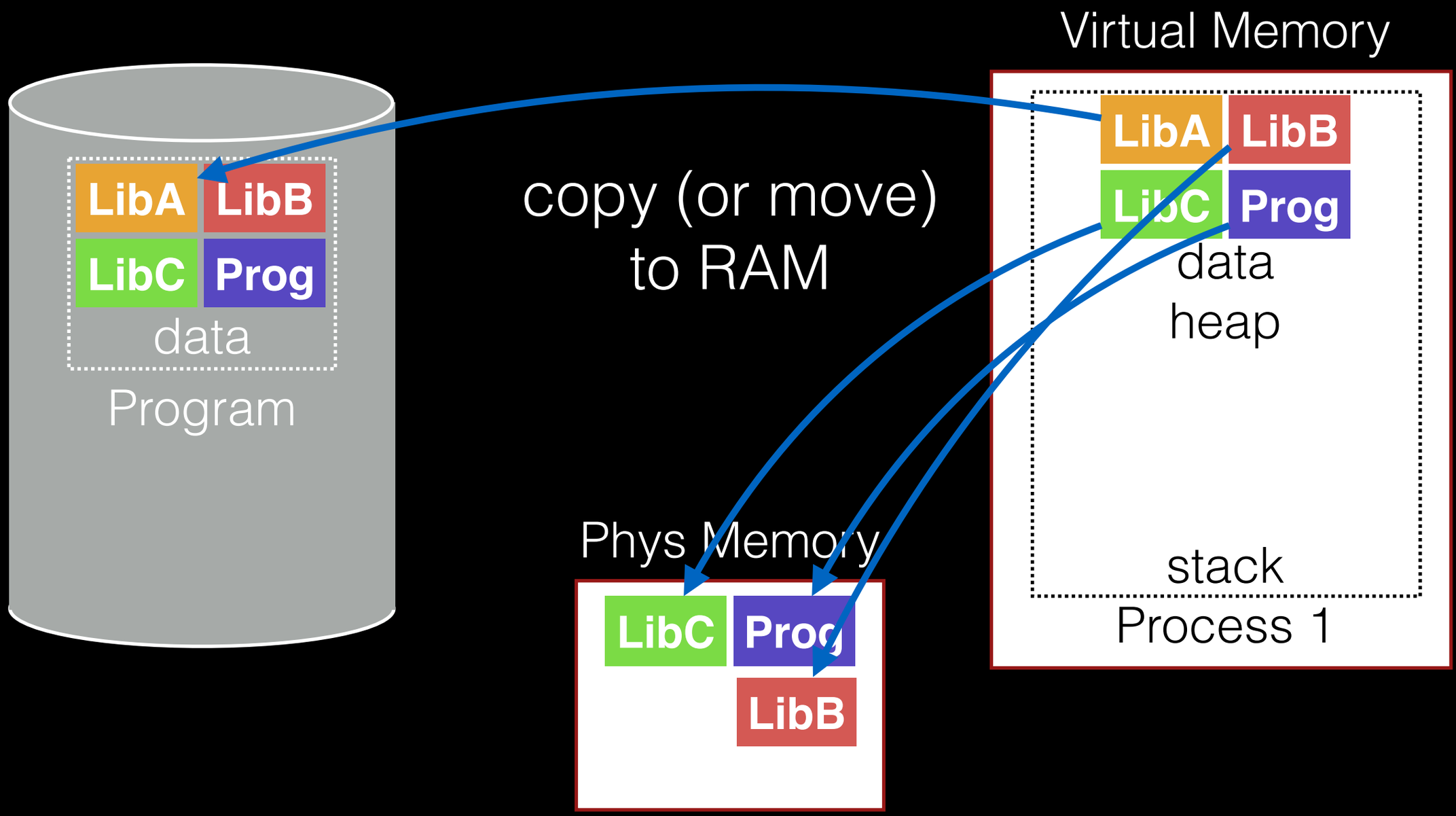
many large libraries, some
of which are rarely/never used

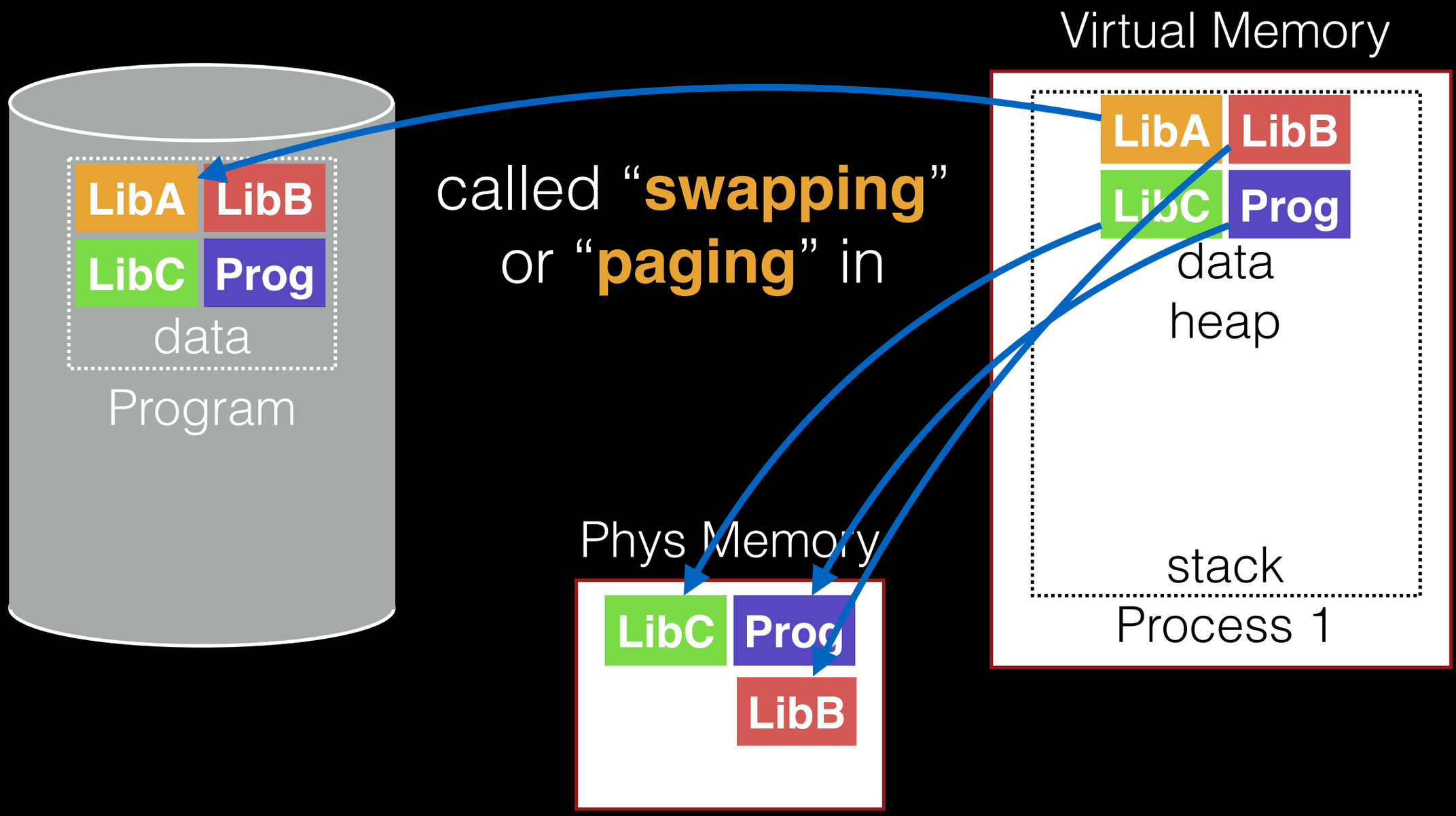
How to avoid wasting physical pages to back rarely used virtual pages?


LibA
LibB
LibC
Prog
data
Program
Phys Memory
LibC
Prog
Virtual Memory
LibA
LibB
LibC
Prog
data
heap
stack
Process 1


Virtual Memory
LibA
LibB
LibC
Prog
data
heap
stack
Process 1
Phys Memory
LibC
Prog
LibA
LibB
LibC
Prog
data
Program


Virtual Memory
LibA
LibB
LibC
Prog
data
heap
stack
Process 1
access LibB
Phys Memory
LibC
Prog
LibA
LibB
LibC
Prog
data
Program


Virtual Memory
LibA
LibB
LibC
Prog
data
heap
stack
Process 1
copy (or move)
to RAM
Phys Memory
LibC
Prog
LibB
LibA
LibB
LibC
Prog
data
Program


Virtual Memory
LibA
LibB
LibC
Prog
data
heap
stack
Process 1
called “swapping”
or “paging” in
Phys Memory
LibC
Prog
LibB
Program
LibA
LibB
LibC
Prog
data

How to know where a page lives?

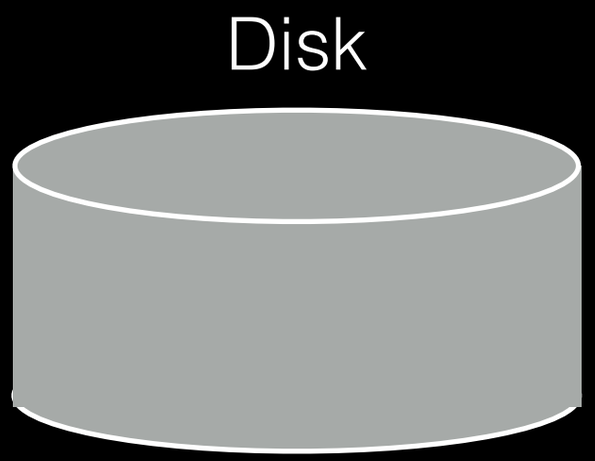
Present Bit

PFN	valid	prot
10	1	r-x
-	0	-
23	1	rw-
-	0	-
-	0	-
-	0	-
-	0	-
-	0	-
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

Present Bit

PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28	1	rw-	0
4	1	rw-	1

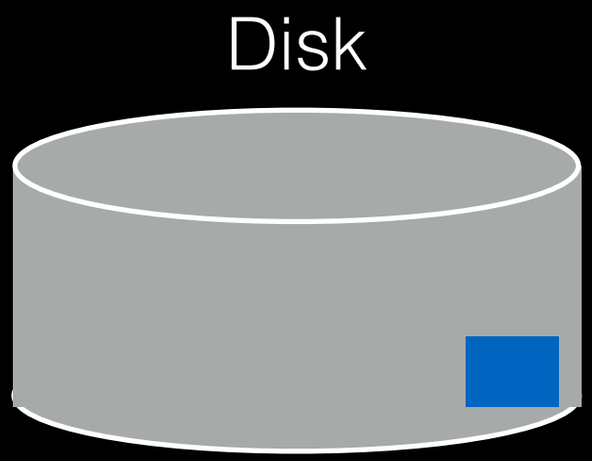
Present Bit

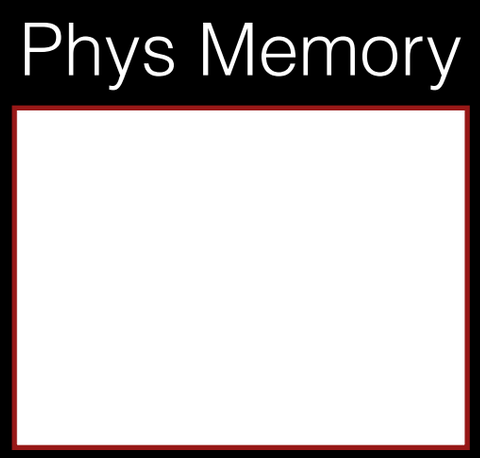


Phys Memory

PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28	1	rw-	0
4	1	rw-	1

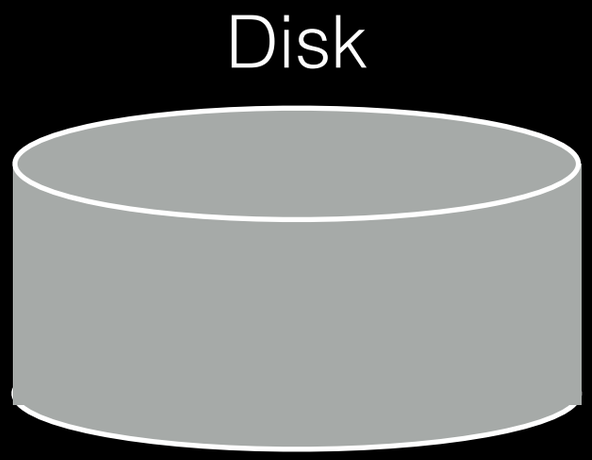
Present Bit

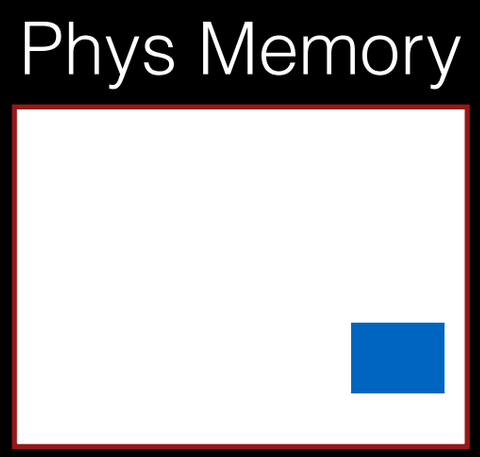




PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28	1	rw-	0 ← access
4	1	rw-	1

Present Bit

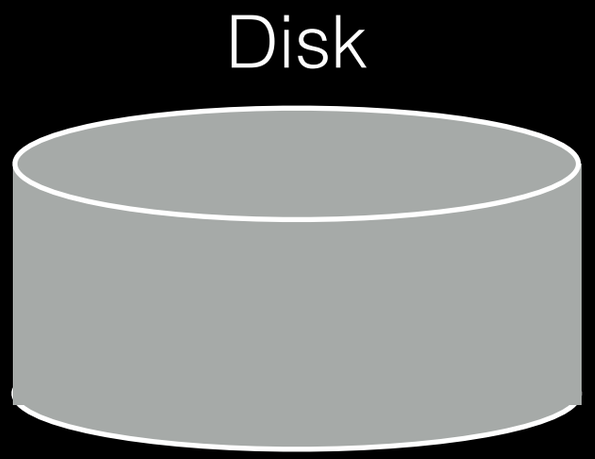




PFN	valid	prot	present	
10	1	r-x	1	
-	0	-	-	
23	1	rw-	0	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
15	1	rw-	1	← access
4	1	rw-	1	

What if no RAM is left?

Present Bit

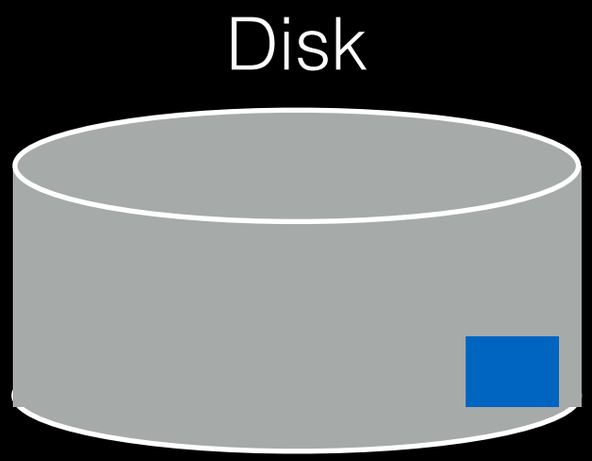


Phys Memory

FULL

PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28	1	rw-	0
4	1	rw-	1

Present Bit

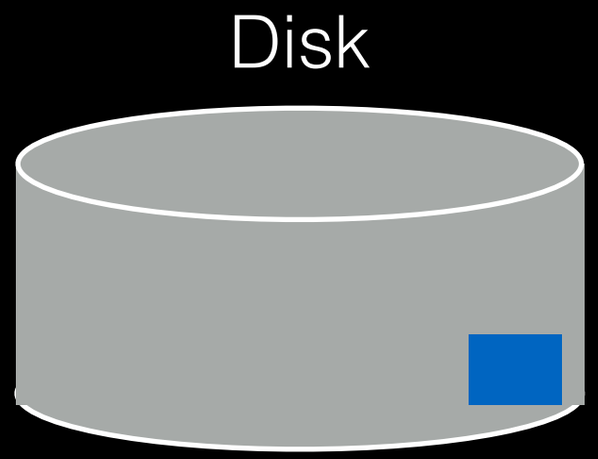


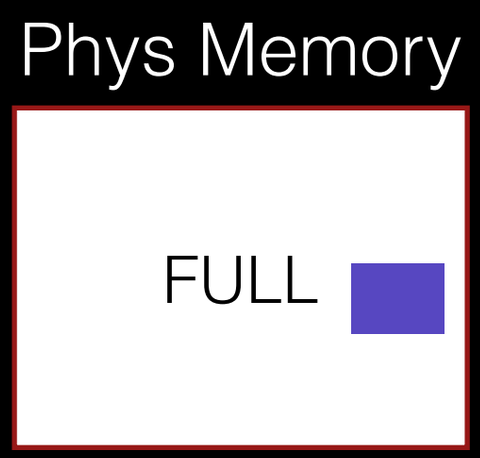
Phys Memory

FULL

PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28	1	rw-	0 ← access
4	1	rw-	1

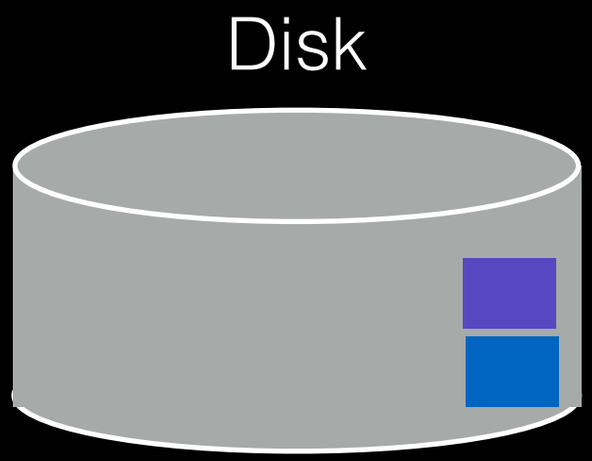
Present Bit





PFN	valid	prot	present	
10	1	r-x	1	← evict
-	0	-	-	
23	1	rw-	0	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
28	1	rw-	0	← access
4	1	rw-	1	

Present Bit

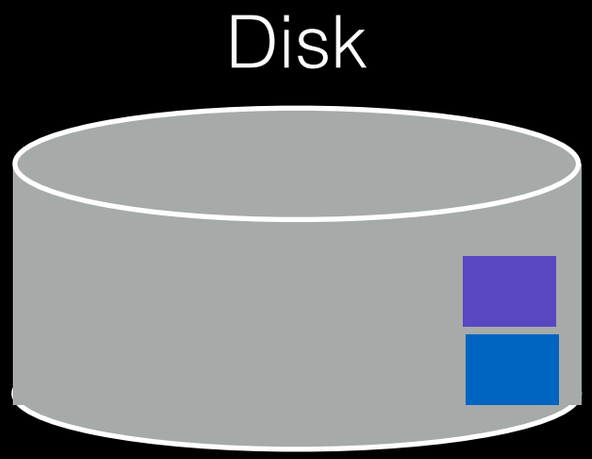


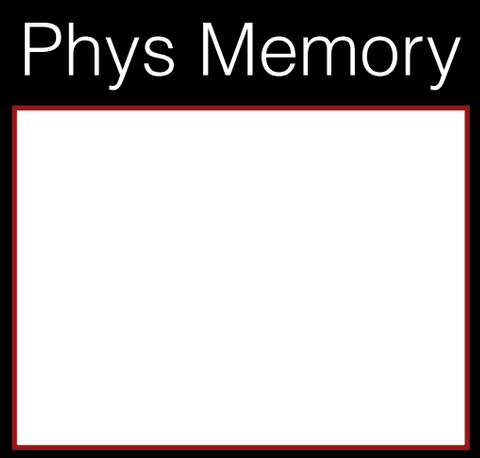
Phys Memory

PFN	valid	prot	present	
25	1	r-x	0	← evict
-	0	-	-	
23	1	rw-	0	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
28	1	rw-	0	← access
4	1	rw-	1	

Present Bit

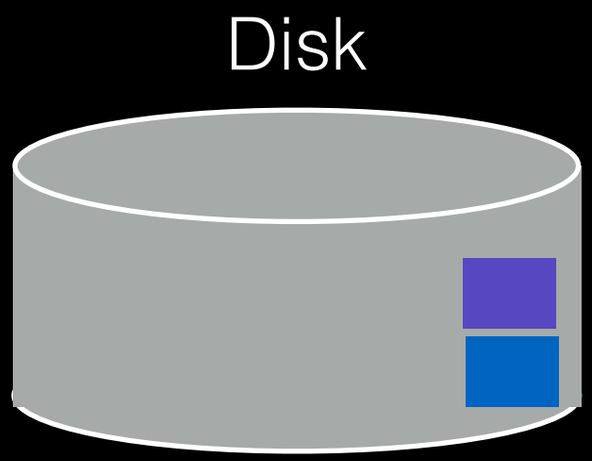
called "**swapping**" or "**paging**" out



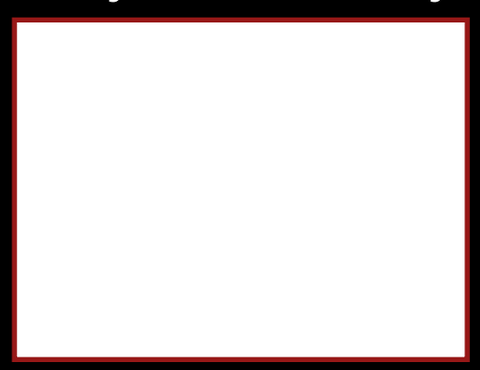


PFN	valid	prot	present	
25	1	r-x	0	← evict
-	0	-	-	
23	1	rw-	0	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
28	1	rw-	0	← access
4	1	rw-	1	

Present Bit

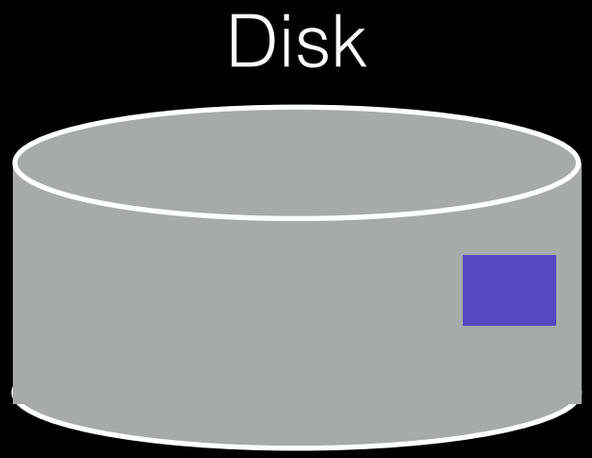


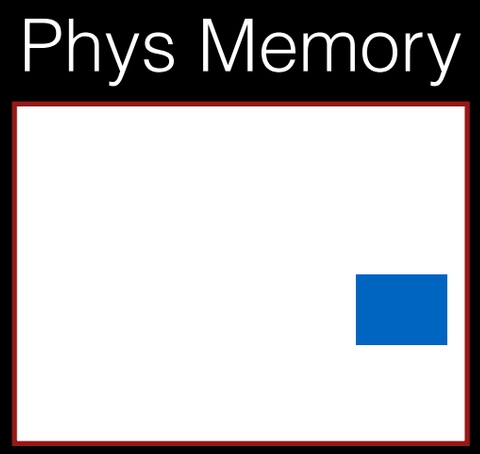
Phys Memory



PFN	valid	prot	present	
25	1	r-x	0	
-	0	-	-	
23	1	rw-	0	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
28	1	rw-	0	← access
4	1	rw-	1	

Present Bit





PFN	valid	prot	present	
25	1	r-x	0	
-	0	-	-	
23	1	rw-	0	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
-	0	-	-	
10	1	rw-	1	← access
4	1	rw-	1	

Why not leave page on disk?

Performance: RAM vs. Disk

How long does it take to access a 4-byte `int`?

RAM: 5ns to 40ns per `int` (depending on TLB hit)

Disk: 15ms per `int`

Performance: RAM vs. Disk

How long does it take to access a 4-byte `int`?

RAM: 5ns to 40ns per `int` (depending on TLB hit)

Disk: 15ms per `int`
- because of high fixed costs
- reading 4KB of ints: 15us per `int`
- reading many megabytes of ints: 30ns per `int`

Average Memory Access Time (AMAT)

$Hit_{\%}$ = portion of accesses that go straight to RAM
$Miss_{\%}$ = portion of accesses that go to disk first
T_m = time for memory access
T_d = time for disk access

$$AMAT = (Hit_{\%} * T_m) + (Miss_{\%} * T_d)$$

Average Memory Access Time (AMAT)

$Hit_{\%}$ = portion of accesses that go straight to RAM
$Miss_{\%}$ = portion of accesses that go to disk first
T_m = time for memory access
T_d = time for disk access

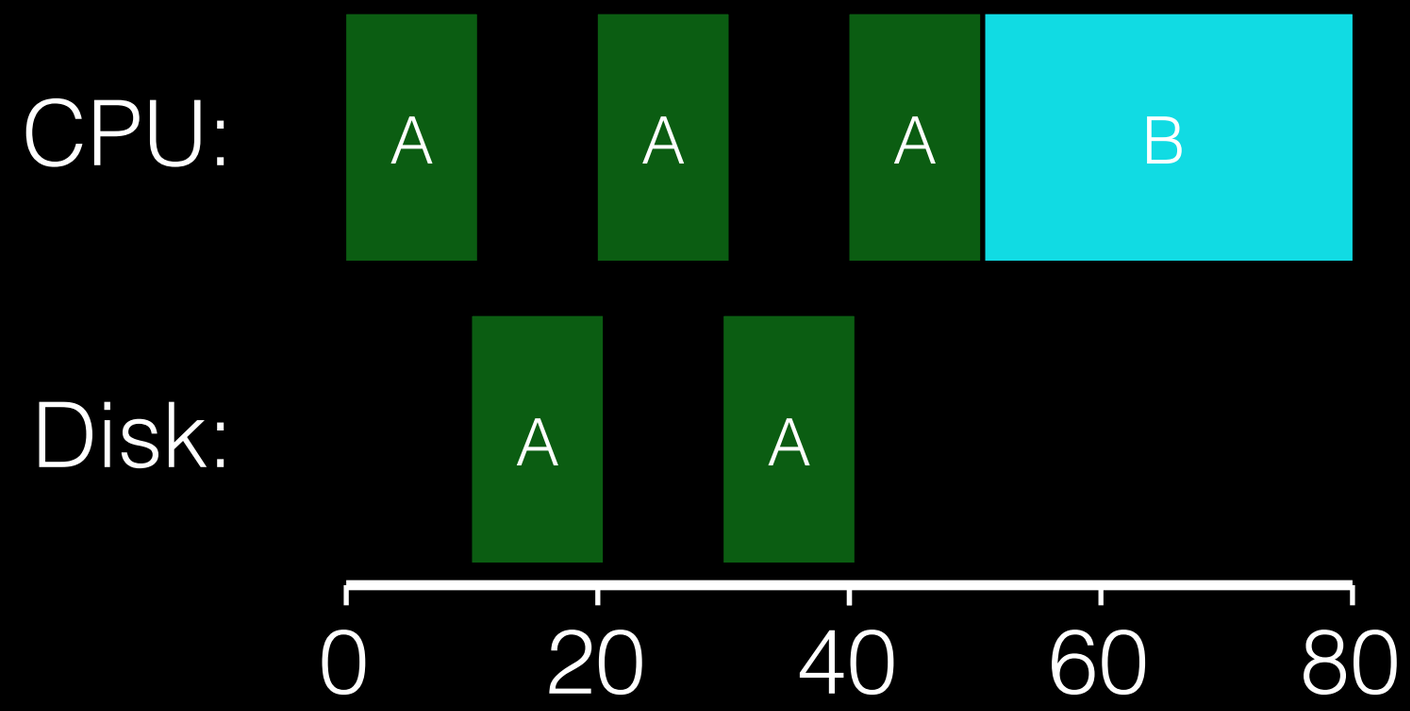
$AMAT = (Hit_{\%} * T_m) + (Miss_{\%} * T_d)$

Problem 3

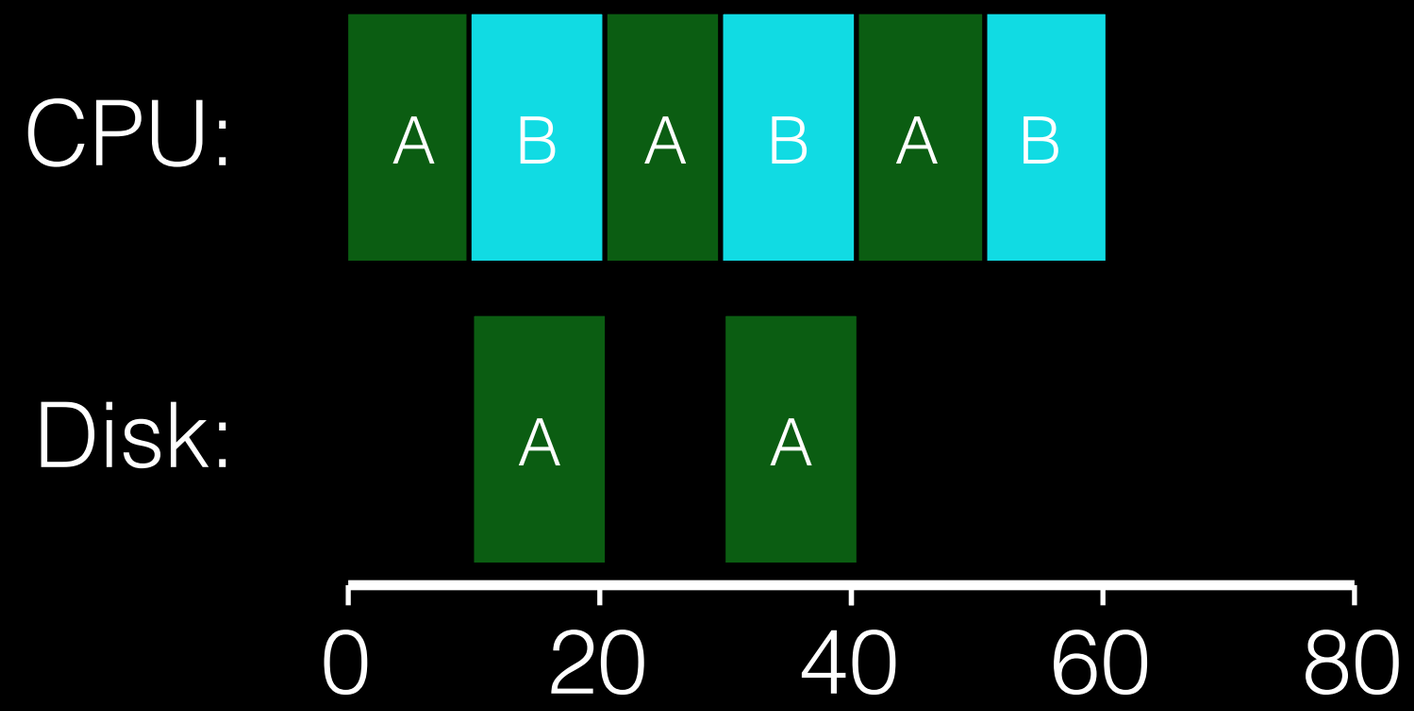
Who should do swapping?

H/W or **OS**?

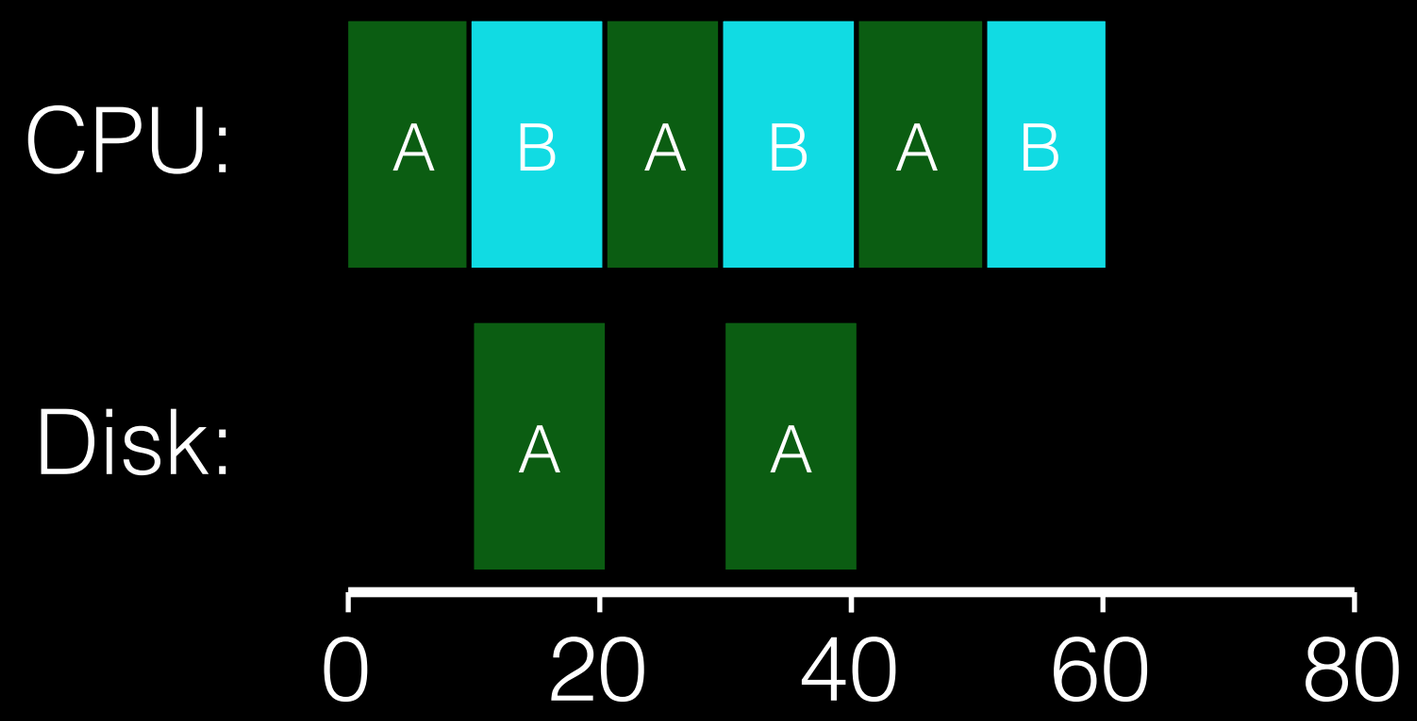
H/W



OS



OS



context switch to other process while swapping in

Translation Steps

H/W: for each mem 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, or page miss]
else: extract **PFN**, insert in **TLB**, retry

Translation Steps

H/W: for each mem 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, or page miss]
    else: extract **PFN**, insert in **TLB**, retry

which steps are expensive?

Translation Steps

H/W: for each mem 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, or page miss]
(cheap) *else*: extract **PFN**, insert in **TLB**, retry

which steps are expensive?

Translation Steps

H/W: for each mem 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, or page miss]
(cheap) *else*: extract **PFN**, insert in **TLB**, retry

which steps are expensive?

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
```

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

which steps are expensive?

Page-Fault Handler (OS)

(cheap)	**PFN** = FindFreePage()	
(cheap)	if (**PFN** == -1)	
(depends)	**PFN** = EvictPage()	what to evict?
(expensive)	DiskRead(**PTE**.DiskAddr, **PFN**)	what to read?
(cheap)	**PTE**.present = 1	(policy)
(cheap)	**PTE**.**PFN** = **PFN**	
(cheap)	retry instruction	

Caching Policy

Workload: series of loads/stores to virtual pages

Cache: chooses what to prefetch/evict

Metric: hit rate, AMAT

Cache “algebra”, given 2 variables, find the 3rd:

f(**W**, **C**) = **M**

Cache

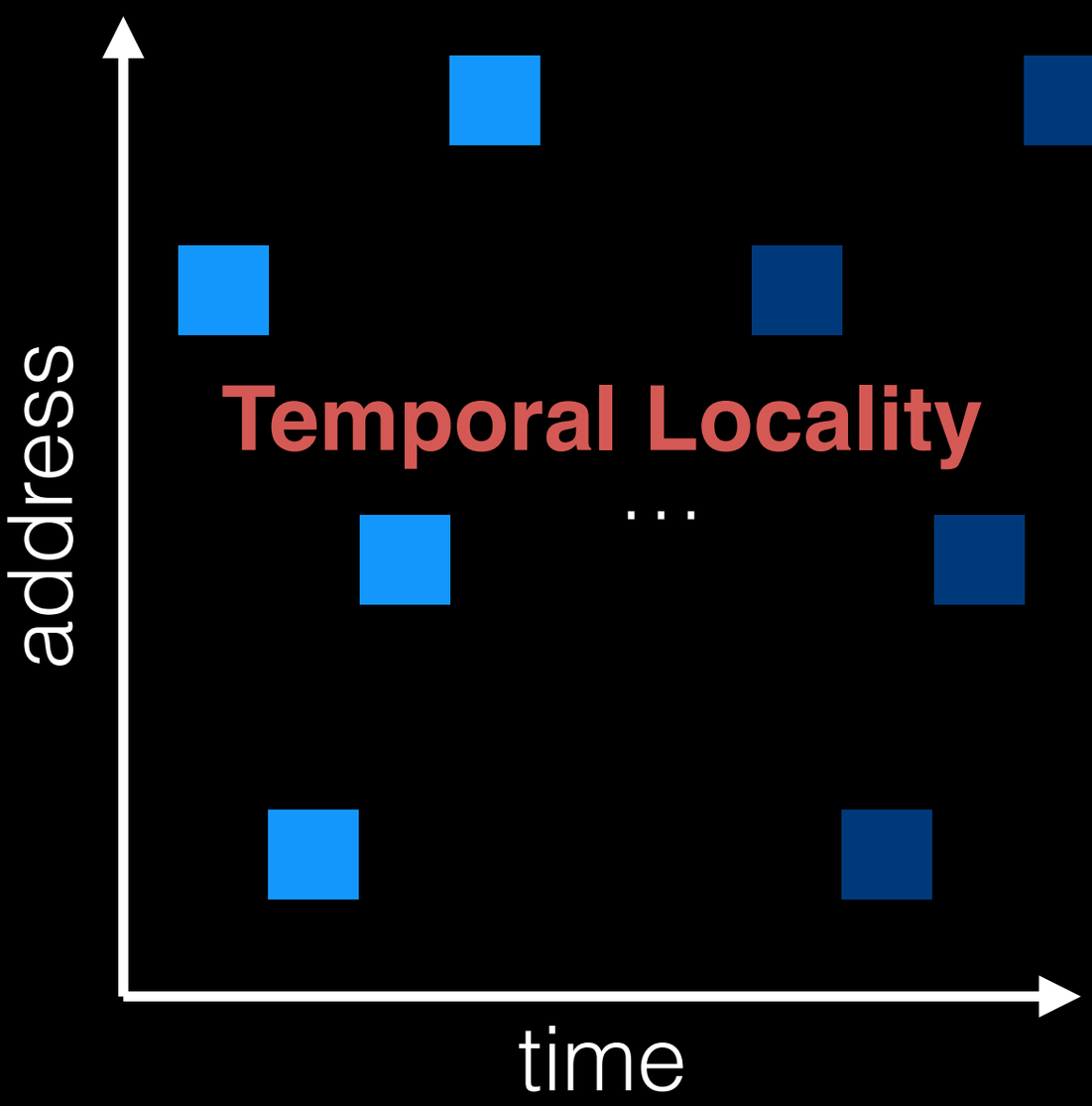
Upon access, we must load the desired page.

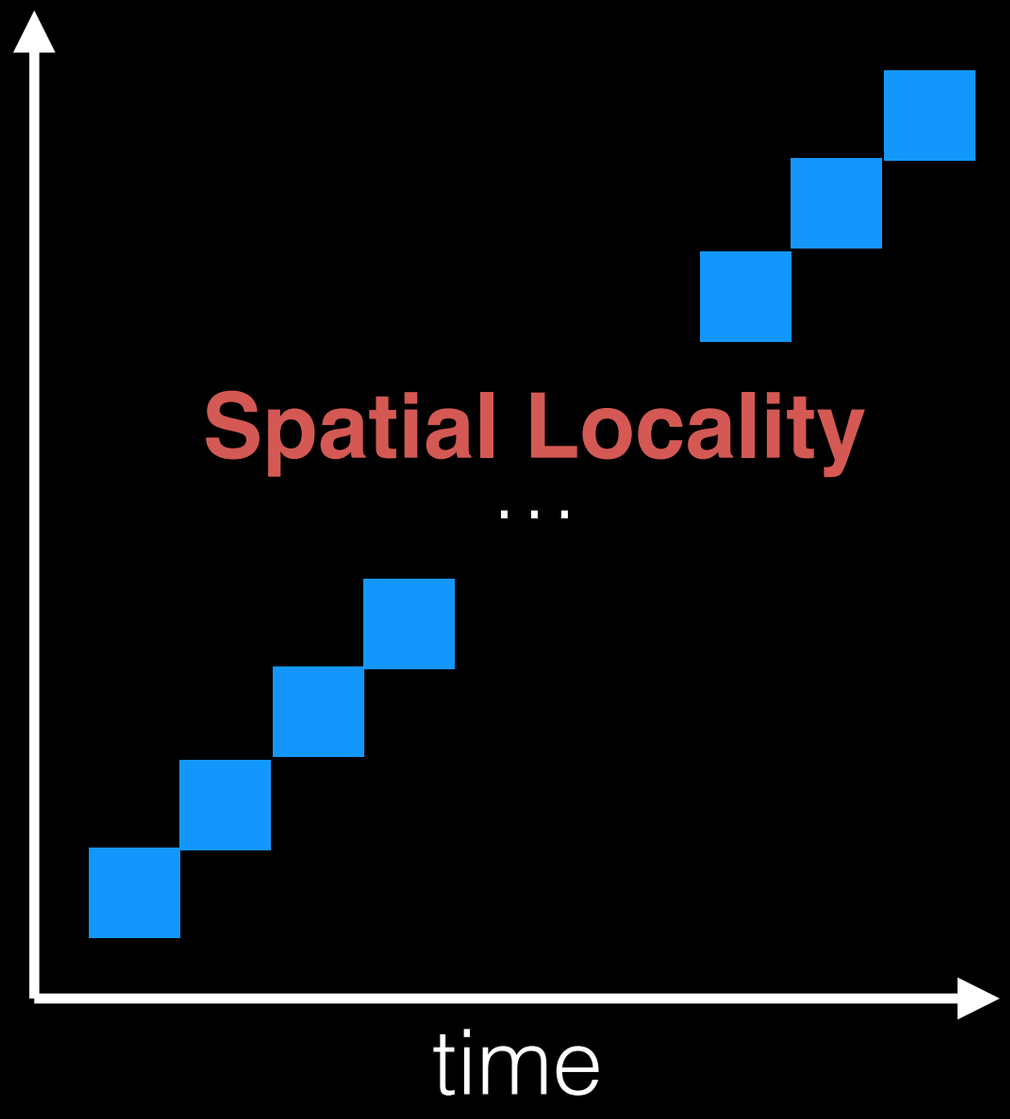
Do we **prefetch** other adjacent pages?
(remember disks have high fixed costs)

Prefetching more means we will have to **evict** more.

What to **evict**?

Workload: is prefetching good?





Cache

Upon access, we must load the desired page.

Do we **prefetch** other adjacent pages?
(remember disks have high fixed costs)

Prefetching more means we will have to **evict** more.

What to **evict**?

Cache

Upon access, we must load the desired page.

Do we **prefetch** other adjacent pages?
(remember disks have high fixed costs)

Prefetching more means we will have to **evict** more.

What to **evict**? [today's focus]

Replacement Policy

Want to maximize hit rate?
Optimal strategy: evict pages to be accessed furthest in future

Example Workload: 1,2,3,4,1,2,3,4,3,2,1

Problem 4: optimal algorithm

Access	Hit	State (after)
1		
2		
3		
4		
1		
2		
3		
4		
3		
2		
1		

assume
cache size 3

Problem 4: optimal algorithm

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	???
1		
2		
3		
4		
3		
2		
1		

assume
cache size 3

Problem 4: optimal algorithm

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,4
1		
2		
3		
4		
3		
2		
1		

assume
cache size 3

Problem 4: optimal algorithm

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,4
1		
2		
3		
4		
3		
2		
1		

Worksheet

assume
cache size 3

Problem 4: optimal algorithm

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,4
1	yes	1,2,4
2	yes	1,2,4
3	no	2,3,4
4	yes	2,3,4
3	yes	2,3,4
2	yes	2,3,4
1	no	…

assume
cache size 3

Problem 4: optimal algorithm

Worksheet:
hit rate?

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,4
1	yes	1,2,4
2	yes	1,2,4
3	no	2,3,4
4	yes	2,3,4
3	yes	2,3,4
2	yes	2,3,4
1	no	…

assume
cache size 3

Problem 4: optimal algorithm

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,4
1	yes	1,2,4
2	yes	1,2,4
3	no	2,3,4
4	yes	2,3,4
3	yes	2,3,4
2	yes	2,3,4
1	no	…

no fair!
compulsory miss

assume
cache size 3

Problem 4: optimal algorithm

no fair!
compulsory miss

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,4
1	yes	1,2,4
2	yes	1,2,4
3	no	2,3,4
4	yes	2,3,4
3	yes	2,3,4
2	yes	2,3,4
1	no	…

Worksheet:
hit rate modulo compulsory?

assume cache size 3

FIFO

Items are evicted in the order they are inserted

Same example…

Problem 5: FIFO

Access	Hit	State (after)
1		
2		
3		
4		
1		
2		
3		
4		
3		
2		
1		

assume
cache size 3

Problem 5: FIFO

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	???
1		
2		
3		
4		
3		
2		
1		

assume
cache size 3

Problem 5: FIFO

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	2,3,4
1		
2		
3		
4		
3		
2		
1		

Worksheet

assume
cache size 3

Problem 5: FIFO

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	2,3,4
1	no	3,4,1
2	no	4,1,2
3	no	1,2,3
4	no	2,3,4
3	yes	2,3,4
2	yes	2,3,4
1	no	3,4,1

assume
cache size 3

Problem 6: more FIFO

Access	Hit	State (after)
1		
2		
3		
4		
1		
2		
5		
1		
2		
3		
4		
5		

Worksheet
(a) cache size 3
(b) cache size 4

(a) size 3

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	2,3,4
1	no	3,4,1
2	no	4,1,2
5	no	1,2,5
1	yes	1,2,5
2	yes	1,2,5
3	no	2,5,3
4	no	5,3,4
5	yes	5,3,4

(b) size 4

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,3,4
1	yes	1,2,3,4
2	yes	1,2,3,4
5	no	2,3,4,5
1	no	3,4,5,1
2	no	4,5,1,2
3	no	5,1,2,3
4	no	1,2,3,4
5	no	2,3,4,5

(a) size 3

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	2,3,4
1	no	3,4,1
2	no	4,1,2
5	no	1,2,5
1	yes	1,2,5
2	yes	1,2,5
3	no	2,5,3
4	no	5,3,4
5	yes	5,3,4

(b) size 4

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,3,4
1	yes	1,2,3,4
2	yes	1,2,3,4
5	no	2,3,4,5
1	no	3,4,5,1
2	no	4,5,1,2
3	no	5,1,2,3
4	no	1,2,3,4
5	no	2,3,4,5

Belady's Anomaly

(a) size 3

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	2,3,4
1	no	3,4,1
2	no	4,1,2
5	no	1,2,5
1	yes	1,2,5
2	yes	1,2,5
3	no	2,5,3
4	no	5,3,4
5	yes	5,3,4

(b) size 4

Access	Hit	State (after)
1	no	1
2	no	1,2
3	no	1,2,3
4	no	1,2,3,4
1	yes	1,2,3,4
2	yes	1,2,3,4
5	no	2,3,4,5
1	no	3,4,5,1
2	no	4,5,1,2
3	no	5,1,2,3
4	no	1,2,3,4
5	no	2,3,4,5

LRU, MRU

LRU: evict least-recently used
 - consider history

MRU: evict most-recently used

LRU, MRU

Count hits for four combos:

Policy: LRU or MRU (both size 3)

Workloads:

1,2,3,4,3,4,3,4

AND

1,2,3,4,1,2,3,4

LRU, MRU

Count hits for four combos:

Policy: LRU or MRU (both size 3)

Workloads:

1,2,3,4,3,4,3,4

AND

1,2,3,4,1,2,3,4

worksheet!

(problem 7)

Discuss

Can Belady's anomaly happen with LRU?

Discuss

Can Belady's anomaly happen with LRU?

Stack property: smaller cache always subset of bigger

Discuss

Can Belady's anomaly happen with LRU?

Stack property: smaller cache always subset of bigger

Does optimal have stack property?

LRU Hardware Support

What is needed?

LRU Hardware Support

What is needed?

Timestamps. Why can't OS alone track this?

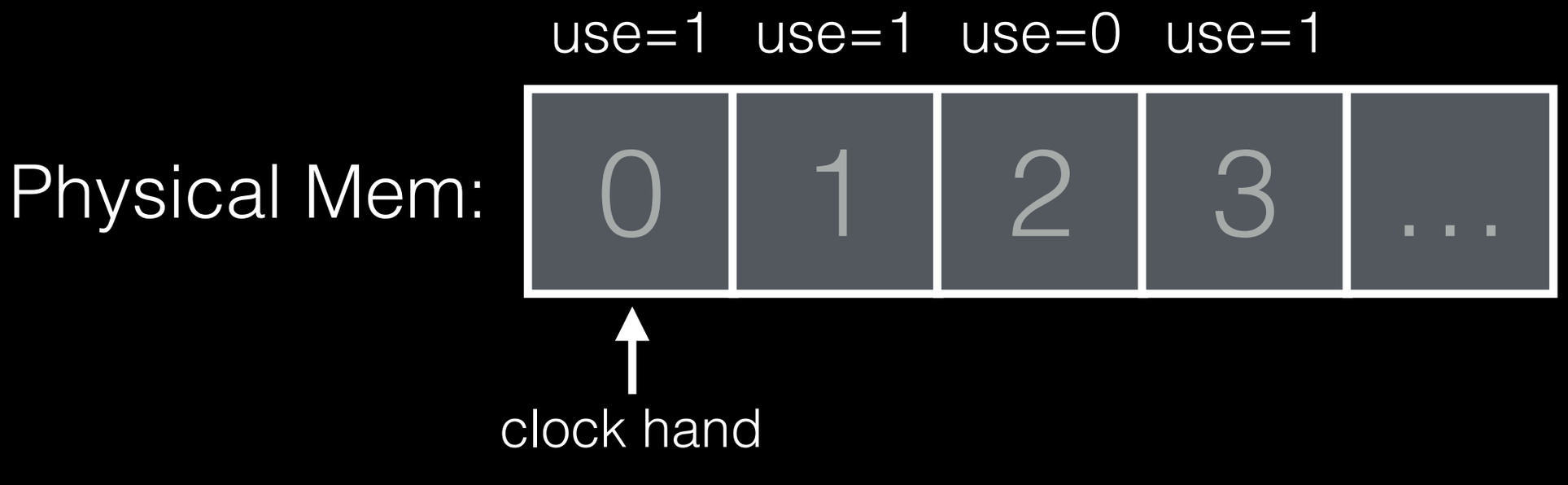
LRU Hardware Support

What is needed?

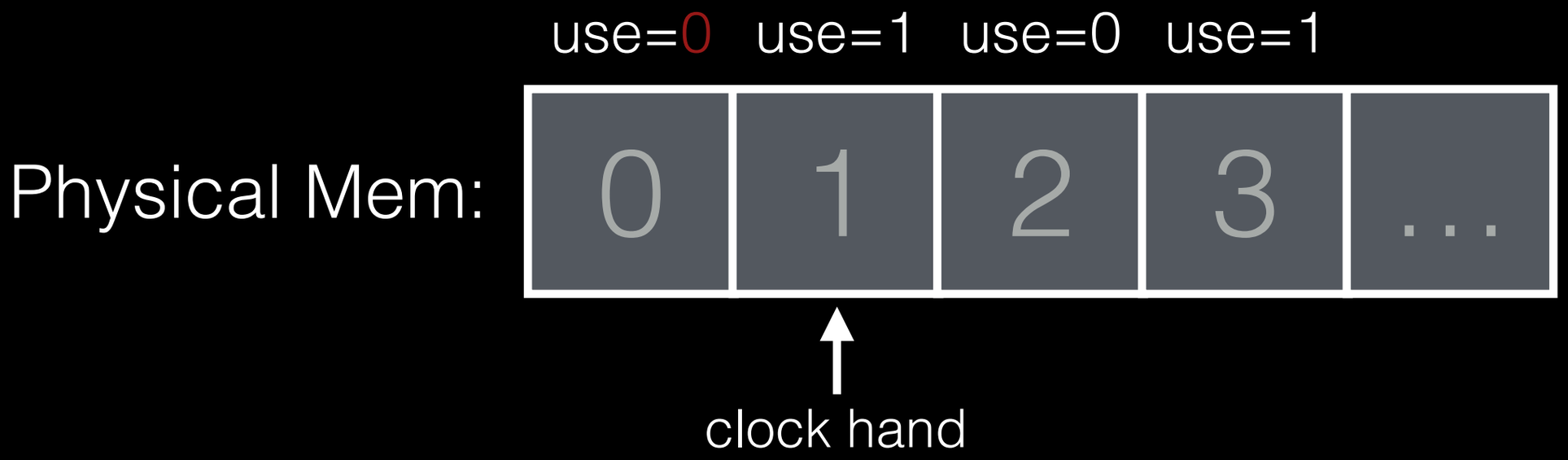
Timestamps. Why can't OS alone track this?

Cheap approximation: reference (or use) bits.
- set upon access, cleared by OS
- useful for clock algorithm

Clock: Look For a Page



Clock: Look For a Page



Clock: Look For a Page

Clock: Look For a Page

evict **page 2** because it has not been recently used

Clock: Look For a Page

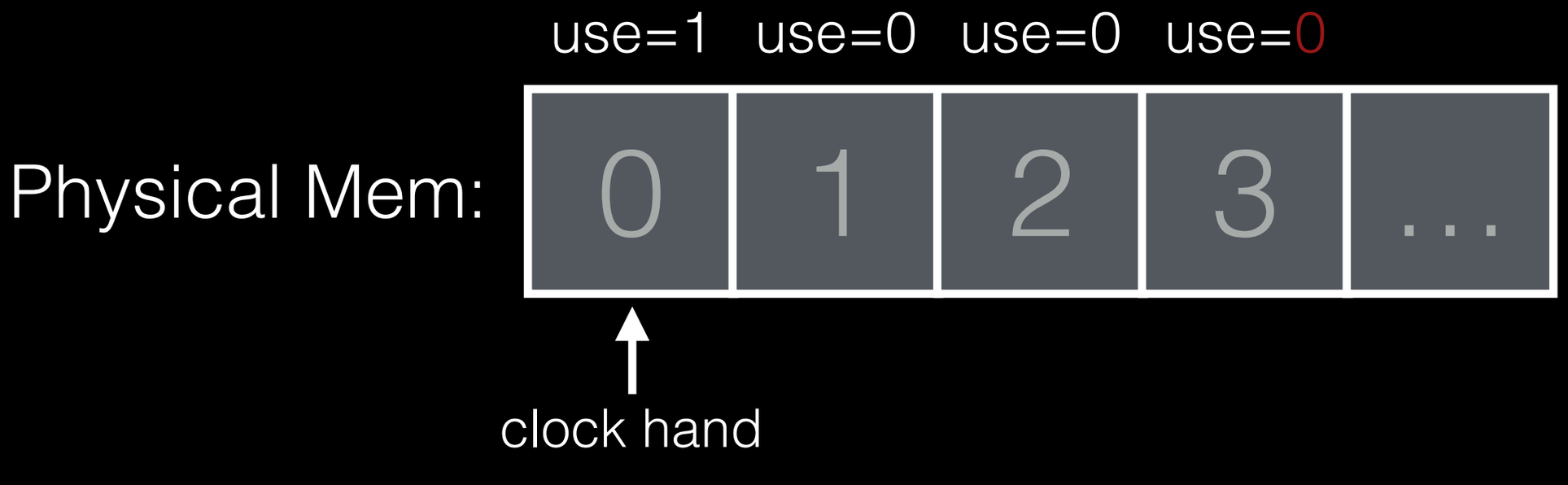
Clock: Look For a Page

page 0 is accessed

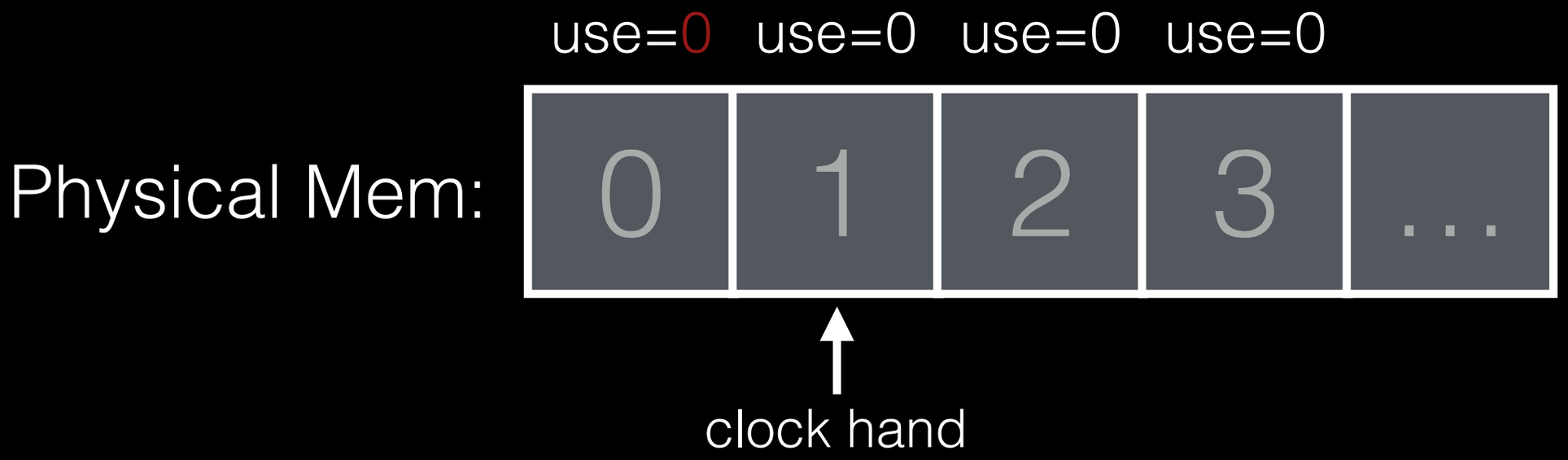
Clock: Look For a Page

Clock: Look For a Page

Clock: Look For a Page

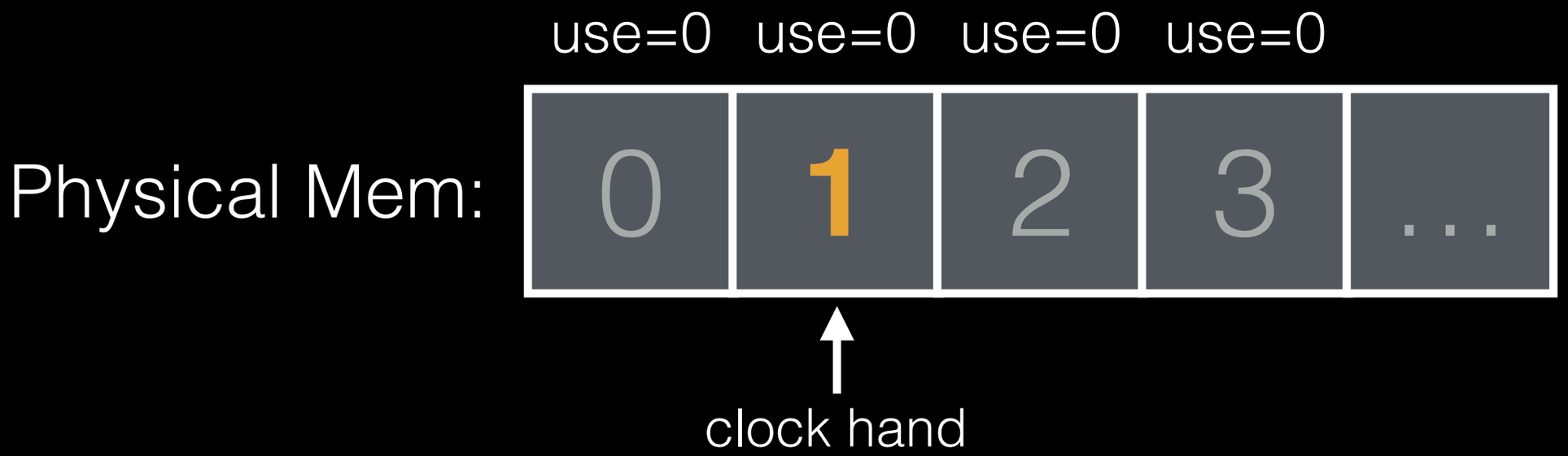


Clock: Look For a Page



Clock: Look For a Page

evict **page 1** because it has not been recently used



Other factors

Assume page is both in RAM **and** on disk

Do we have to write to disk for eviction?

Other factors

Assume page is both in RAM **and** on disk

Do we have to write to disk for eviction?
- not if page is clean
- track with dirty bit

Thrashing

A machine is **thrashing** when there is not enough RAM, and we constantly swap in/out pages

Solutions?

Thrashing

A machine is **thrashing** when there is not enough RAM, and we constantly swap in/out pages

Solutions?

- admission control (like scheduler project)
- buy more memory
- Linux out-of-memory killer!

Summary

Virtual memory abstraction:
- big address space
- possible to fill address space, even with small RAM!

Many policy decisions:
- what to prefetch
- what to evict
- how much to evict

Announcements

One easy piece done!

Shell due Friday.

Lab office hours now.