Part 4: Beyond the limits of Physical Memory

The “RealWorld”™ -- Time to add swapping

An Implicit Assumption

Simple Paging: the entire address space of a proc fits into physical mem

The RealWorld™:

- 32-bit address space (4 GB)
- 64-bit address space (17 billion GB)

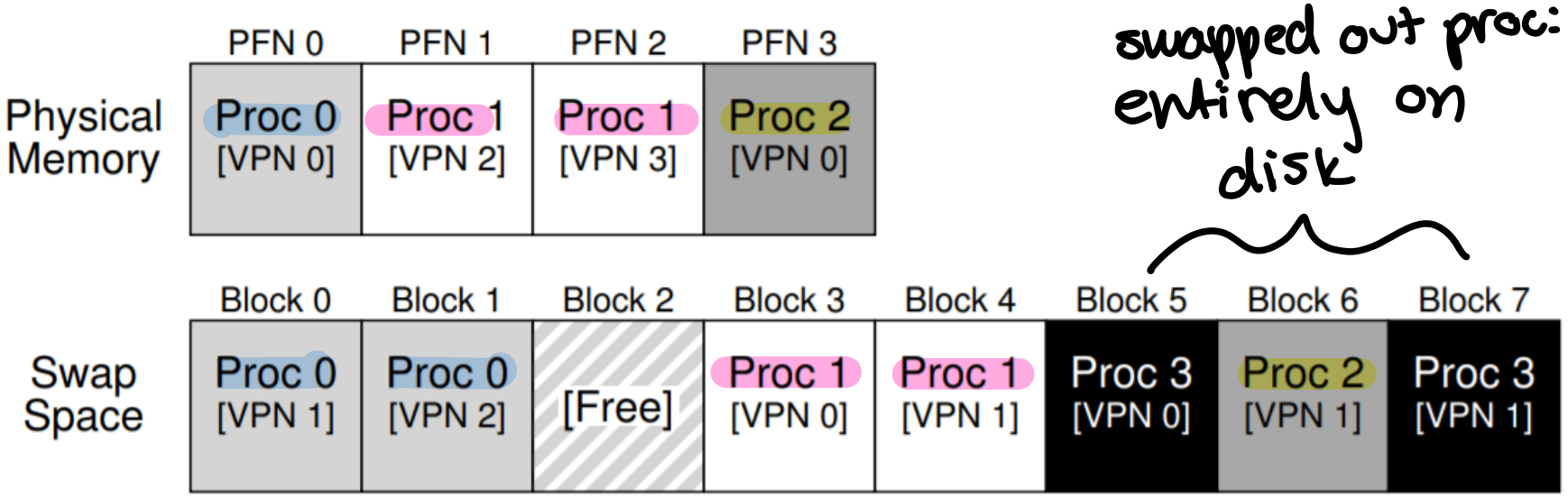
The Crux of the Problem

How do we go beyond the constraint of phys. mem?

→ use disk

Introduction

Swap Space: Reserved space on disk for swapped out pages



Address Translation

On TLB *miss*:

→ page table entry

1. Fetch the PTE

this is a perfectly legal address. not a user error. OS sees and handles.

2. if valid, check "present" bit
   a) if 1, all good (page is in RAM)
   b) if 0, no physical addr. (page has been swapped out)
      └ throws error

Page fault: the act of accessing a page not in phys mem

On Page Fault its an interupt

1. Set requesting process to blocked
2. if memory is full*, swap out a page from RAM to disk
   - which page? ⇒ replacement policy decides
3. issue I/O request to disk
4. update the page table entry (PTE)
5. set requesting process to Ready

Metric of Choice

$$\text{Hit Rate} = \frac{\#\text{hits}}{\#\text{hits} + \#\text{miss}}$$

- bigger is better metric

page fault

Replacement Policy #1: FIFO

First-in, First-out

1. Place pages into a queue when swapped in
2. Choose head of queue when you need to replace a page

doesn't care if its used often, does not take locality into account

Example: FIFO

cold Misses

Access	Hit/Miss?	Evict	State
0	M		0
1	M		01
2	M		012
0	H		012
1	H		012
3	M	0	123
0	M	1	230
3	H		230
1	M	2	301
2	M	3	012
1	H		012

|phys mem| = 3 pages and RAM starts empty

HR = #H / #H + #M

= 4/4+7 = 4/11

= 36%.

Just For Fun: Belady's Anomaly

#	H/M?	Evict	State
1	M		1
2	M		12
3	M		123
4	M	1	234
1	M	2	341
2	M	3	412
5	M	4	125
1	H		125
2	H		125
3	M	1	253
4	M	2	534
5	H		534

#	H/M?	Evict	State
1	M		1
2	M		12
3	M		123
4	M		1234
1	H		1234
2	H		1234
5	M	1	2345
1	M	2	3451
2	M	3	4512
3	M	4	5123
4	M	5	1234
5	M	1	2345

Replacement Policy #2: LRU

Least Recently used

→ replace the page which has not been referenced for the longest time

→ embraces locality, in general code has lots of locality

Example: LRU

Access	Hit/Miss?	Evict	State
0	M		0
1	M		01
2	M		012
0	H		120
1	H		201
3	M	2	013
0	H		130
3	H		103
1	H		031
2	M	0	312
1	H		321

|pm|=3 pages
starts empty

$\frac{6}{6+5} = 54.5\%$

LRU has high overhead
← state changes every-time

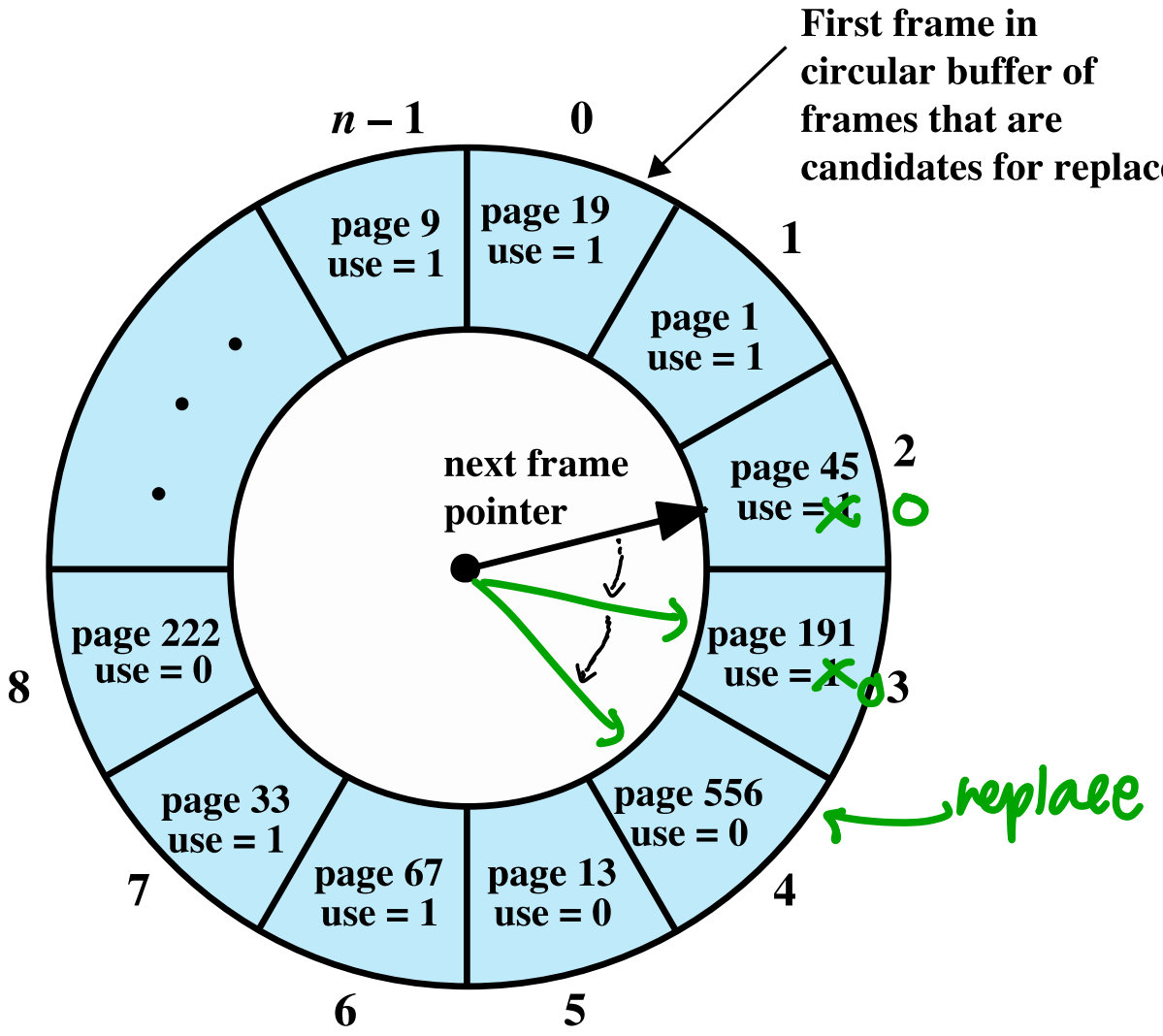
Replacement Policy #3: CLOCK

Approximates LRU

1. add a "use" bit to every page
2. Whenever a page is referenced set use = 1
3. OS uses the "clock" algorithm on page replacement
   - all pages are stored as a circular list
   - a "clock" hand points to some page
   - while (p.use == 1):
     p.use = 0
     p += 1

     } replace p at the end

CLOCK Visual



(a) State of buffer just prior to a page replacement

"Replacement Policy #4": OPT

Optimal

1. assume we have perfect knowledge of the future
2. replace the page which is accessed furthest in the future

OPT Example

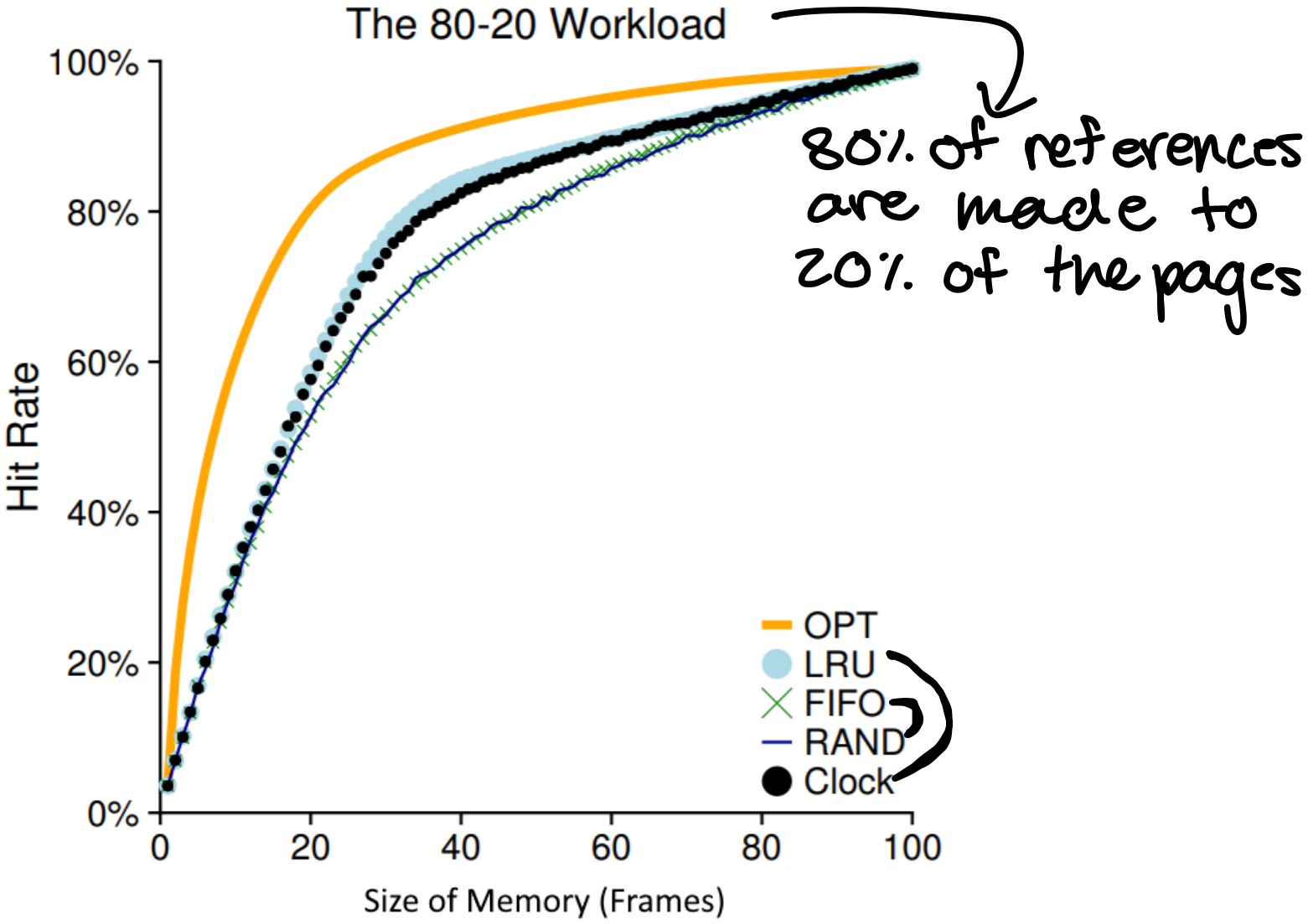
Access	Hit/Miss?	Evict	State
0	M		0
1	M		0 1
2	M		0 1 2
0	H		0 1 2
1	H		0 1 2
3	M	2	0 1 3
0	H		0 1 3
3	H		0 1 3
1	H		0 1 3
2	M	3	0 1 2
1	H		0 1 2

this doesn't matter

|PM| = 3 pages

$$\frac{6}{6+5} = 54.5\%$$

Replacement Comparison



Part 5: Some Loose Ends

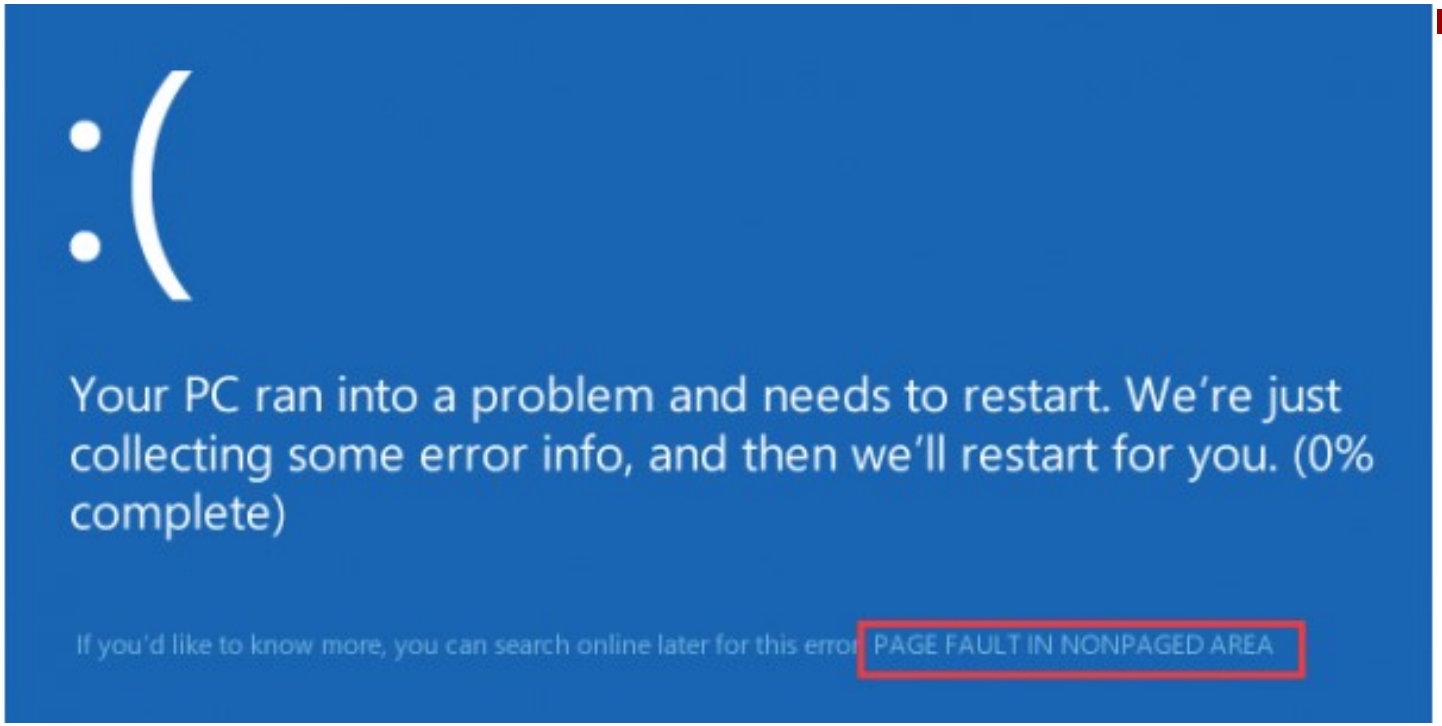
Terminology

Thrashing: When the system spends majority time handling page faults rather than doing useful work

Frame Locking: a locked frame cannot be removed from RAM.

→ typically used for the kernel

PAGE_FAULT_IN_NONPAGED_AREA



Non-Paged Area: locked frames

Resident Set Management

Resident Set: Portion of the proc's addr space in RAM

- set of pages w/ present = 1
- typically limited by the OS
- fixed or dynamic

Replacement Scope:

- Global: all unlocked pages
- Local: replace pages in proc's resident set

Fetch Policies

Demand Paging: only swap in a page on page fault

→ lots of cold misses

Prepaging: pre-fetch pages in some way

→ fix cold misses