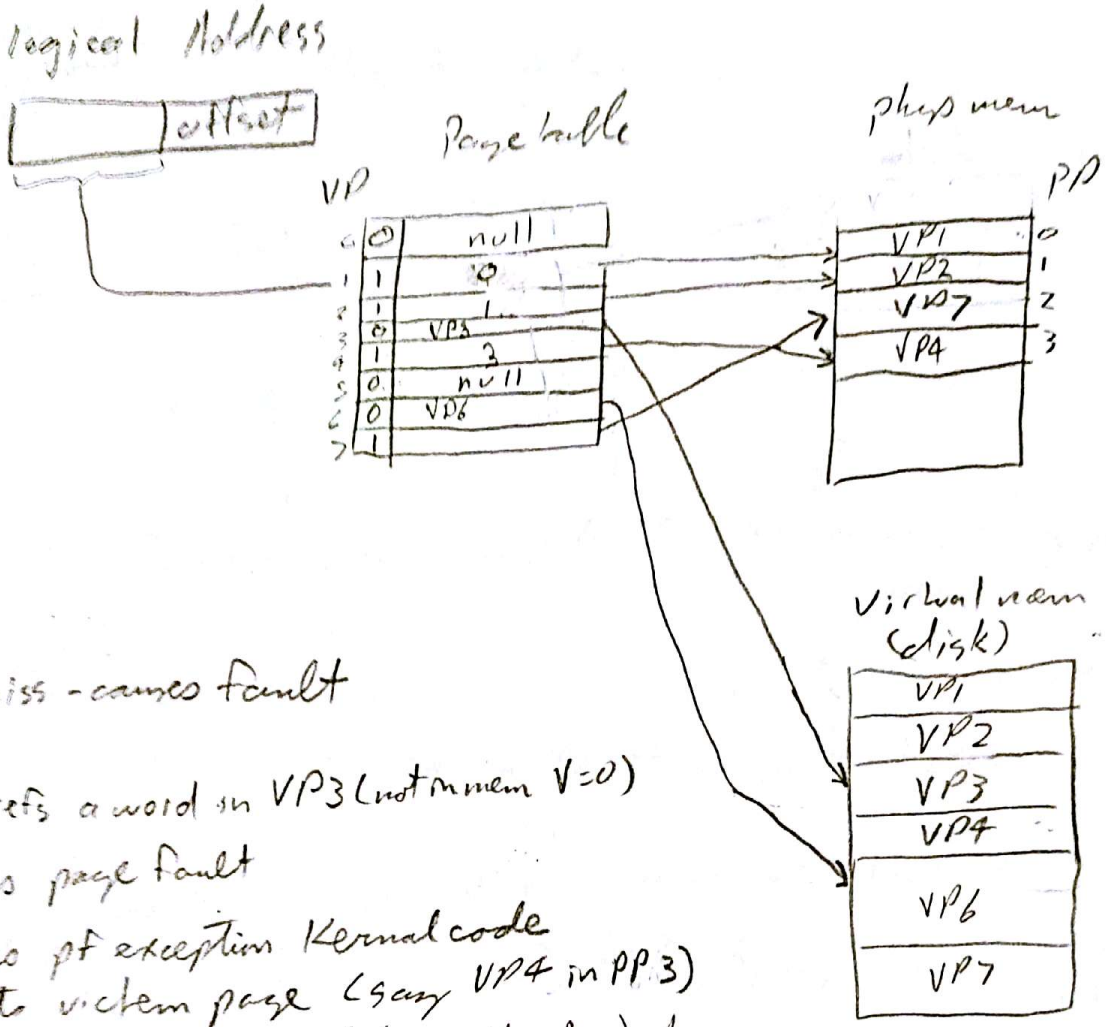
Page fault



mem miss - causes fault

CPU refs a word in VP3 (not in mem V=0)

triggers page fault

- invokes pf exception Kernal code
- selects victim page (say VP4 in PP 3)
- if VP4 has been modded (written to) it will be written to disk
- Kernal mods page table for VP4 to indicate VP4 not in mem
- Kernal copies VP3 from disk to PP4 in mem
- updates page table
- then returns
- restart faulting instruction
  - now VP3 is in main mem.
  - normal exec.

Page Replacement

Optimal throw out page that will not be used the longest time in the future.

FIFO oldest loaded

LRU page used longest ago.

Want = fewest page faults (disk access is slow)

test: run also on a particular string of mem references, compute # page faults

Frame access sequence (first page accessed is 2, then 3, then 2

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

Optimal 5 pages, 3 frames

	2	3	2	1	5	2	4	5	3	2	5	2
Frame	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
						↑			↑			
										no replace needed		

great, can't predict future

LRU how to imp? tag each page with Δt (lots overhead) scan through when replacing

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2

FIFO often bits heavily used throughout prog. (func calls) those will be paged in + out repeatedly.

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	~~4~~	2

ignores access patterns

LRU would be good but very expensive

compromise (find old page but not necessarily oldest)

Clock (LRU approx)

- ref bit/page
- mem ref hardware sets bit to 1
- page replace OS finds page with 0
- OS traverses pages clearing bits ever true
- FIFO + LRU - gives FIFO selection second chance

2	3	2	1	5	2	4	5	3	2	5	2
2^1	2^1	2^1	$\rightarrow 2^1$	5^1	5^1	5^1	$\rightarrow 5^1$	3^1	3^1	3^1	$\rightarrow 3^1$
$\Rightarrow$	3^1	3^1	$3^1 \rightarrow 3^0$	2^1	2^1	2^1	$\rightarrow 2^0$	$\rightarrow 2^1$	2^0	2^1	
	$\Rightarrow$	$\Rightarrow$	1^1	1^0	$\rightarrow 1^0$	4^1	4^1	4^0	4^0	5^1	5^1

= probs? yes what if a page is dirty (has been modded & needs to be written to disk)
= takes a long time, better to choose nondirty page

Mem Management

- less central (its cheap)
- virtual to physical is useful { gets rid of contiguous requirement }

- Large page sizes (better TLB coverage)
  smaller page tables
  bigger internal frag though

- Large 64 bit address space
  sparse address space (few blocks used)

-