K22 - Operating Systems: Design Principles and Internals

Fall 2025 @dit

Vaggelis Atlidakis

Lecture 18

References: Similar OS courses @Columbia, @Stanford, @UC San Diego, @Brown, @di (previous years); and textbooks: Operating Systems: Three Easy Pieces, Operating Systems: Principles and Practice, Operating System Concepts, Linux Kernel Development, Understanding the Linux Kernel

Overview

- We'll start from hardware and follow a question-oriented approach
  - ~~Intro [Q: What is an OS?]~~
  - ~~Events [Q: When does the OS run?]~~
  - ~~Runtime [Q: How does a program look like in memory?]~~
  - ~~Processes [Q: What is a process?]~~
  - ~~IPC [Q: How do processes communicate?]~~
  - ~~Threads [Q: What is a thread?]~~
  - ~~Synchronization [Q: What goes wrong w/o synchronization?]~~
  - ~~Time Management [Q: What is scheduling?]~~
  - **Memory Management [Q: What is virtual memory?]**
  - Files [Q: What is a file descriptor?]
  - Storage Management [Q: How do we allocate disk space to files?]

* Basic (H/W & S/W)
* Abstractions
* Primitives
* **Mechanisms**

Overview

- **Memory management**
  - **Q1**: What is virtual memory?
  - **Q2**: Why is virtual memory necessary?
  - **Q3**: How is virtual memory implemented?

  ...

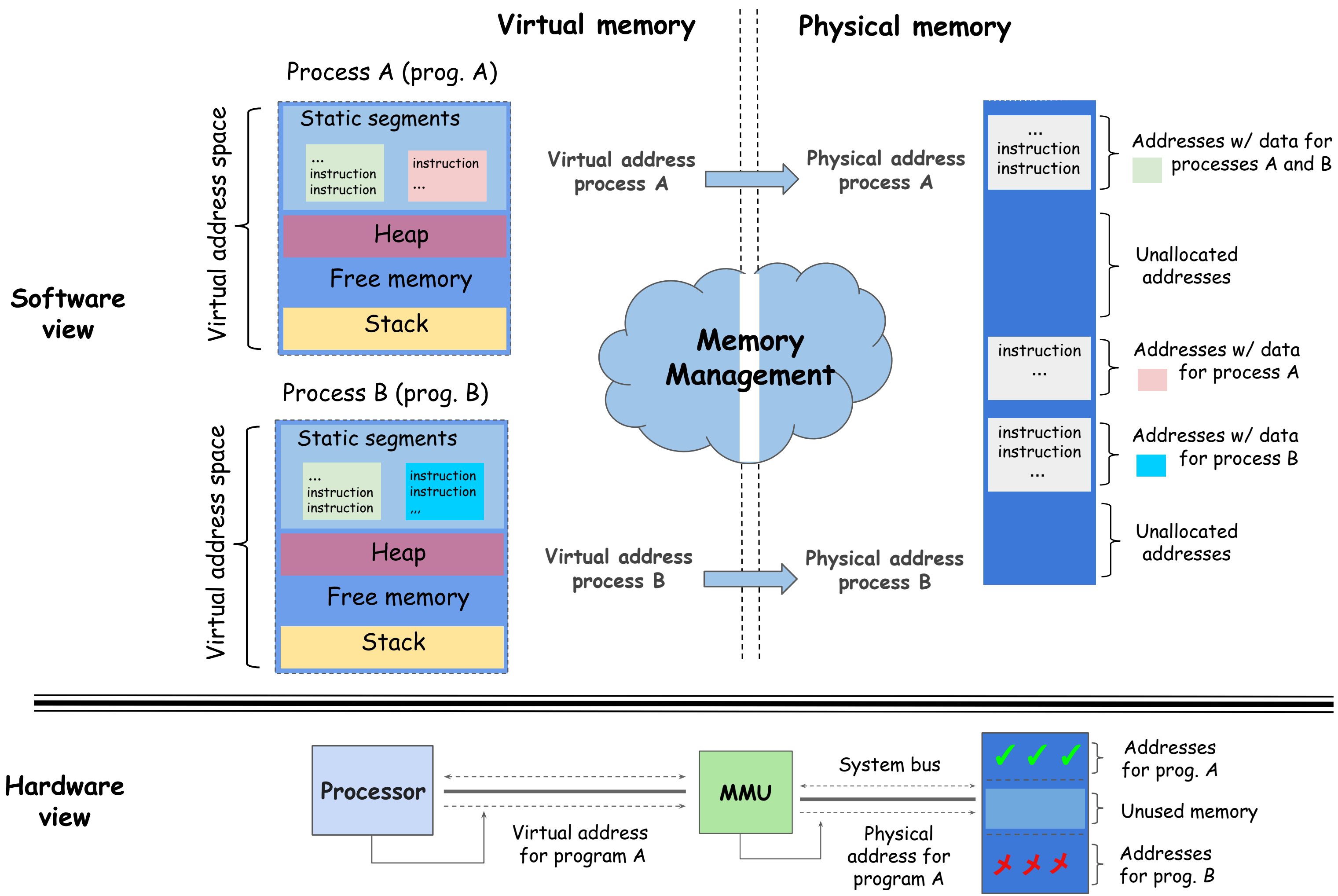
What is virtual memory?

>> A layer of abstraction

- Translates virtual addresses into physical addresses

- **Virtual addr.**: the language processes talk to the processor about memory
- **Physical addr.**: the language actual contents of memory are accessed

- The way programmers imagine programs interact with memory is an illusion on top of the virtual memory abstraction layer



Virtual memory
Physical memory
Software view
Process A (prog. A)
Virtual address space
Static segments
...
instruction
instruction
instruction
...
Heap
Free memory
Stack
Process B (prog. B)
Virtual address space
Static segments
...
instruction
instruction
instruction
instruction
,,,
Heap
Free memory
Stack
Virtual address process A
Physical address process A
Memory Management
Virtual address process B
Physical address process B
...
instruction
instruction
Addresses w/ data for processes A and B
Unallocated addresses
instruction
...
Addresses w/ data for process A
instruction
instruction
...
Addresses w/ data for process B
Unallocated addresses
Hardware view
Processor
Virtual address for program A
MMU
System bus
Physical address for program A
Addresses for prog. A
Unused memory
Addresses for prog. B

What is virtual memory? (demo)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

int main(int argc, char **argv) {
   int i;
   pid_t pid = getpid();

   i = atoi(argv[1]);

  printf("[PID: %d]; &i: %p; i: %d\n", pid, &i, i)
  printf("[PID: %d]; Sleep for %d sec\n", pid, i);
  sleep(i);
  printf("[PID: %d]; I am done now\n", pid, i);
  return 0;
}
```

```
➜ git:(master) ✗ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
0

➜ vm git:(master) ✗ ./hello_vm 10 &
[1] 186394
[PID: 186394]; &i: 0xffffffff210; i: 10
[PID: 186394]: Seep for 10 sec

➜ vm git:(master) ✗ ./hello_vm 3 &
[2] 186432
[PID: 186432]; &i: 0xffffffff210; i: 3
[PID: 186432]: Sleep for 3 sec

➜ vm git:(master) ✗ [PID: 186432]; I am done now
[2]  + 186432 done       ./hello_vm 3

➜ vm git:(master) ✗ [PID: 186394]; I am done now
[1]  + 186394 done       ./hello_vm 10
```

Same address different value, both programs running...

Why need virtual memory?

- **Fault isolation:** Anything that goes wrong in the address space of one process does not affect any other process
- **Illusion of continuous memory:** Programs are written, compiled, assembled, linked, and loaded assuming access to **continuous** memory
- **Frugal use of resources**
  - >> Demand paging: Postpone allocating physical memory until necessary
  - >> Copy-On-Write: One physical replica of common readed-only code
- **Performant use of resources**
  - >> Programs have a memory footprint as much as the size of storage and run at speed close to the speed of CPU caches (how?...)
  - >> **CoW**: Laziness -> defering Vs. **Locality**: Proactivity -> prefetching

How is virtual memory implemented?

"We've rewritten the VM several times in the last ten years, and I expect it will be changed several more times in the next few years. Within five years, we'll almost certainly have to make the current three-level page tables be four levels, etc."

—Linus Torvald, 2001.

How is virtual memory implemented?

> Usually considered the most complex kernel subsystem

- It's simple if you view it as an index

Ask the right questions

Q1: What are the entries of the index?

Q2: How are the entries of the index used?

Q3: How are the entries of the index allocated?

Q4: How are the entries of the index replaced?

How is virtual memory implemented?

> Usually considered the most complex kernel subsystem

- It's simple if you view it as an index

Ask the right questions

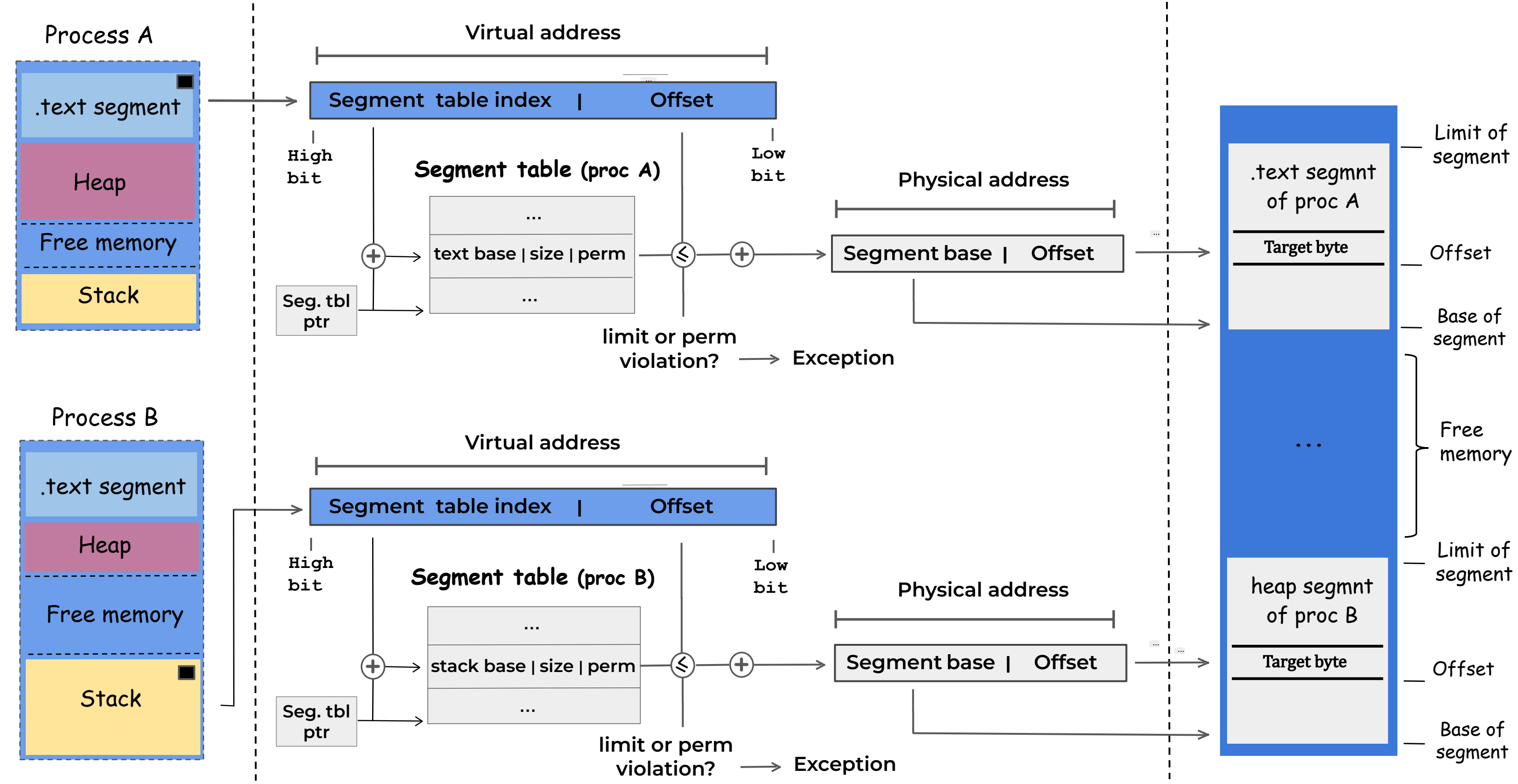
$\boxed{\text{Q1: What are the entries of the index?}}$

Q2: How are the entries of the index used?

Q3: How are the entries of the index allocated?

Q4: How are the entries of the index replaced?

Mechanism: Segmentation



Mechanism: Segmentation

› Assume 32-bit virtual and physical address space
w/ 8-bits for the segment table index (segment selector) and 24-bits for offset

Segment table (process A)

Seg No	Base	Size	Perm
0	**0x4f00000**	**0xffffff**	**rw-**
1	0x0100000	0x1fffff	r-x
2	0x3000000	0x9fffff	r--

<u>Q1</u>: How many segments can a process have?

- 2^8 = 64 segments

<u>Q2</u>: How large can a segment be in bytes?

- 2^24 bytes = 16*(2^20) = 16 MiB

<u>Q3</u>: What's the physical address for virtual address?

- 0x00001030 = [0000 0000] [0000 0000] [0001 0000] [0011 0000]
= [0100 11111] [0000 0000] [0001 0000] [0011 0000]
= 0x4f 00 10 30 => 00 10 30 < 9f ff ff ? OK
= 0x4f001030

Mechanism: Segmentation

› Assume 32-bit virtual and physical address space
w/ 8-bits for the segment table index (segment selector) and 24-bits for offset

Segment table (process A)

Seg No	Base	Size	Perm
0	0x4f00000	0xffffff	rw-
1	0x0100000	0x1fffff	r-x
2	**0x3000000**	0x9fffff	r--

Q1: How many segments can a process have?

- 2^8 = 64 segments

Q2: How large can a segment be in bytes?

- 2^24 bytes = 16*(2^20) = 16 MiB

Q3: What's the physical address for virtual address?

- 0x02a20357 = [0000 0010] [1010 0010] [0000 0011] [0101 1111]
= [0011 0000] [1010 0010] [0000 0011] [0101 1111]
= 0x30 a2 03 5f => a2 03 5f < 9f ff ff ? NOT_OK

Mechanism: Segmentation

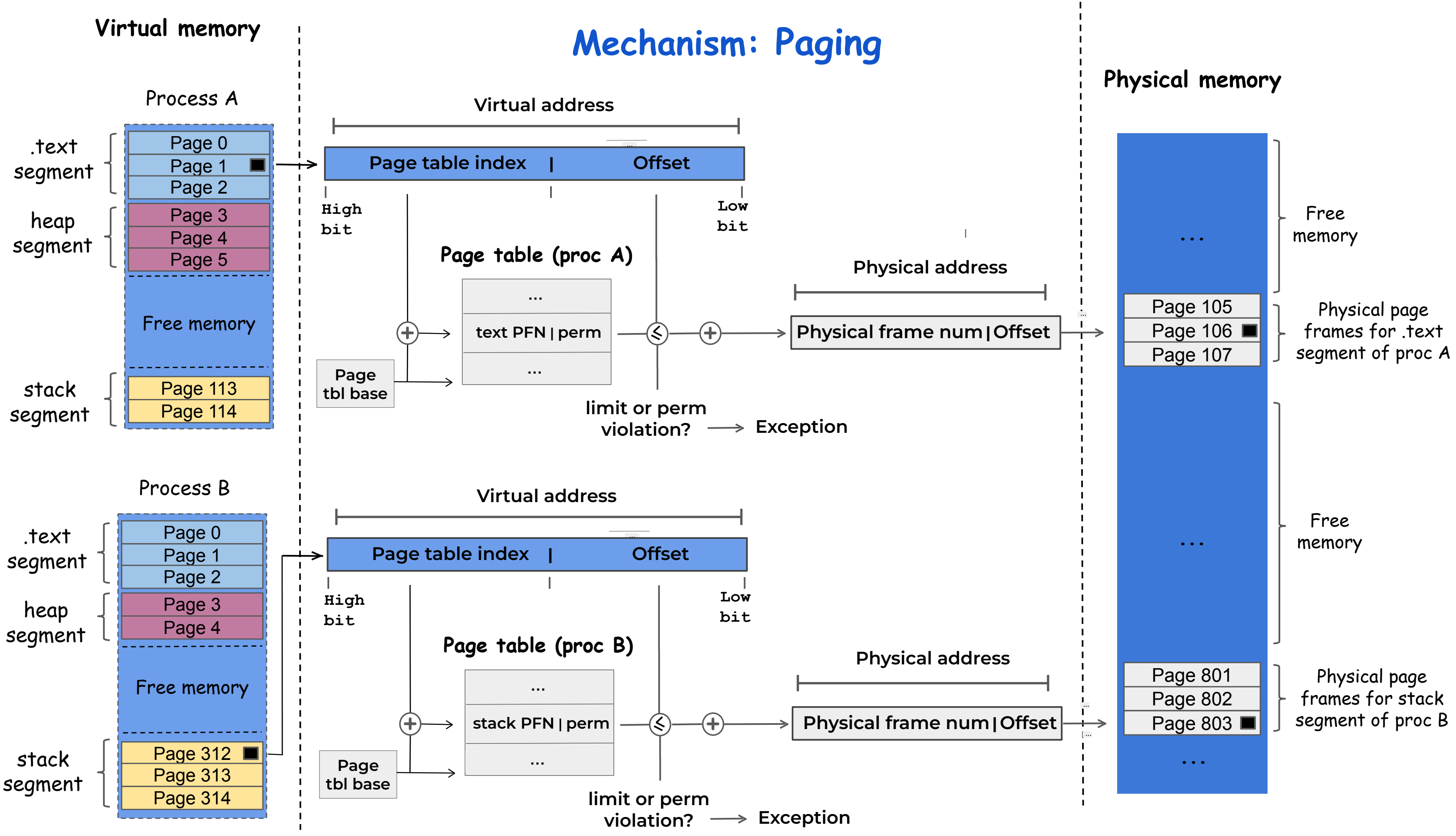
Limitations of Segmentation

> Fragmentation: Inability to use available memory

>> Internal fragmentation: Unused portions internally on each segment

>> External fragmentation: Free segments not usable due to unfit sizes

Mechanism: Paging



Mechanism: Paging

› Assume 32-bit virtual and physical address space, 20-bits for page table index and 4KB pages (i.e., 12 bits for offset)

Page table (process A)

Index	PFN	Perm
0	302	rw-
1	**106**	**r-x**
2	19	r--

Q: What's the physical address for virtual address?

- 0x00000400 = [0000 0000] [0000 0000] [0000 0100] [0000 0000]
  = [0000 0000] [0001 0010] [1110 0100] [0000 0000]
  = 0x00 12 e4 00 = 0x0012e400

- 0x00001402 = [0000 0000] [0000 0000] [0001 0100] [0000 0010]
  = [0000 0000] [0000 0110] [1010 0100] [0000 0010]
  = 0x00 06 a4 02 = 0x0006a402

Mechanism: Paging

Limitations of segmentation

> Fragmentation: Inability to use available memory

>> Internal fragmentation: Unused portions internally on each segment

>> External fragmentation: Free segments not usable due to unfit sizes

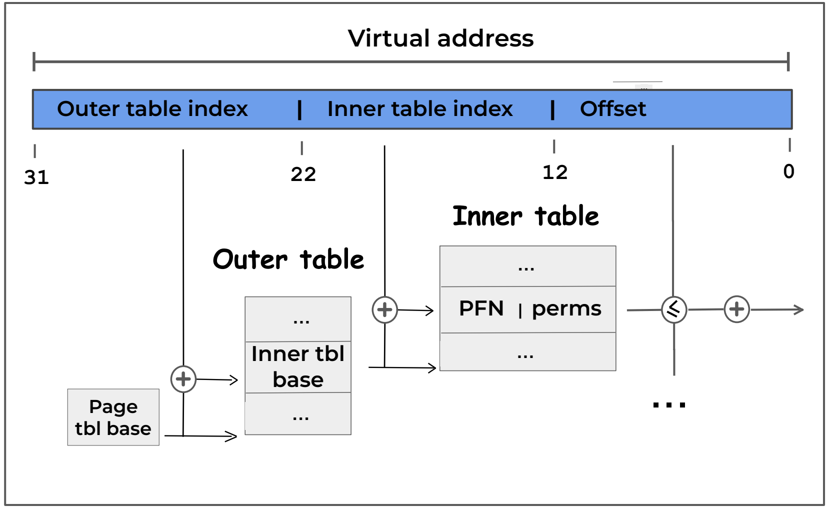
Limitations of single-level paging

> Size of page table: E.g., with 32-bit virtual addresses, need a 4MB flat array per process to map a sparse 4GB address space

> Internal fragmentation: Still an issue, but not as prominent

Mechanism: 2-level hierarchical paging

› Assume 32-bit virtual and physical address space, where the 12 lower bits are for the offset, the 10 middle bits are the index for the inner table, and the 10 higher bits are the index for the outer table



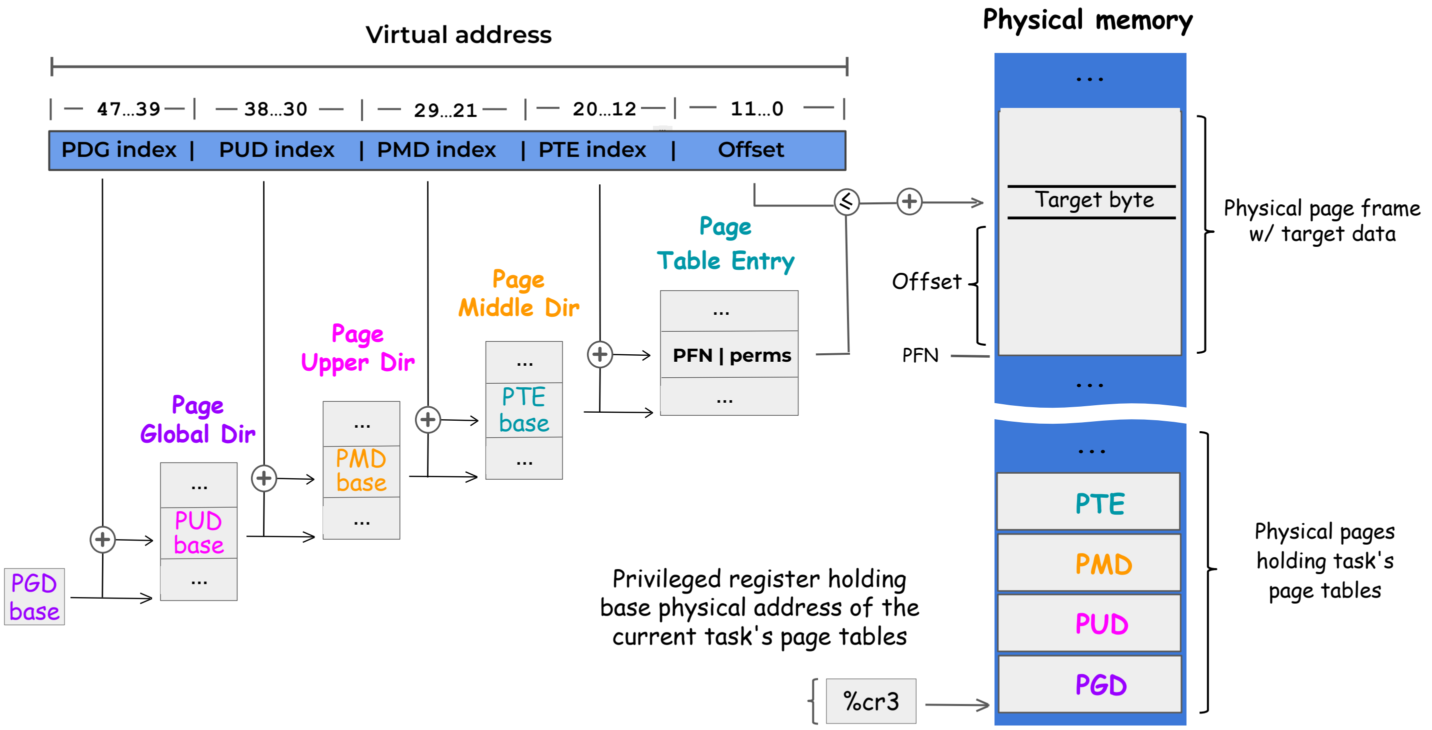
Q: Size of "map-able" address space?

› **With two pages** (one for each of the outer and inner table): 1 * 1024 PTEs * 4KB = 4MiB

› With each additional page in the inner table (i.e., 1024 PTEs), 4 additional MiB can be mapped

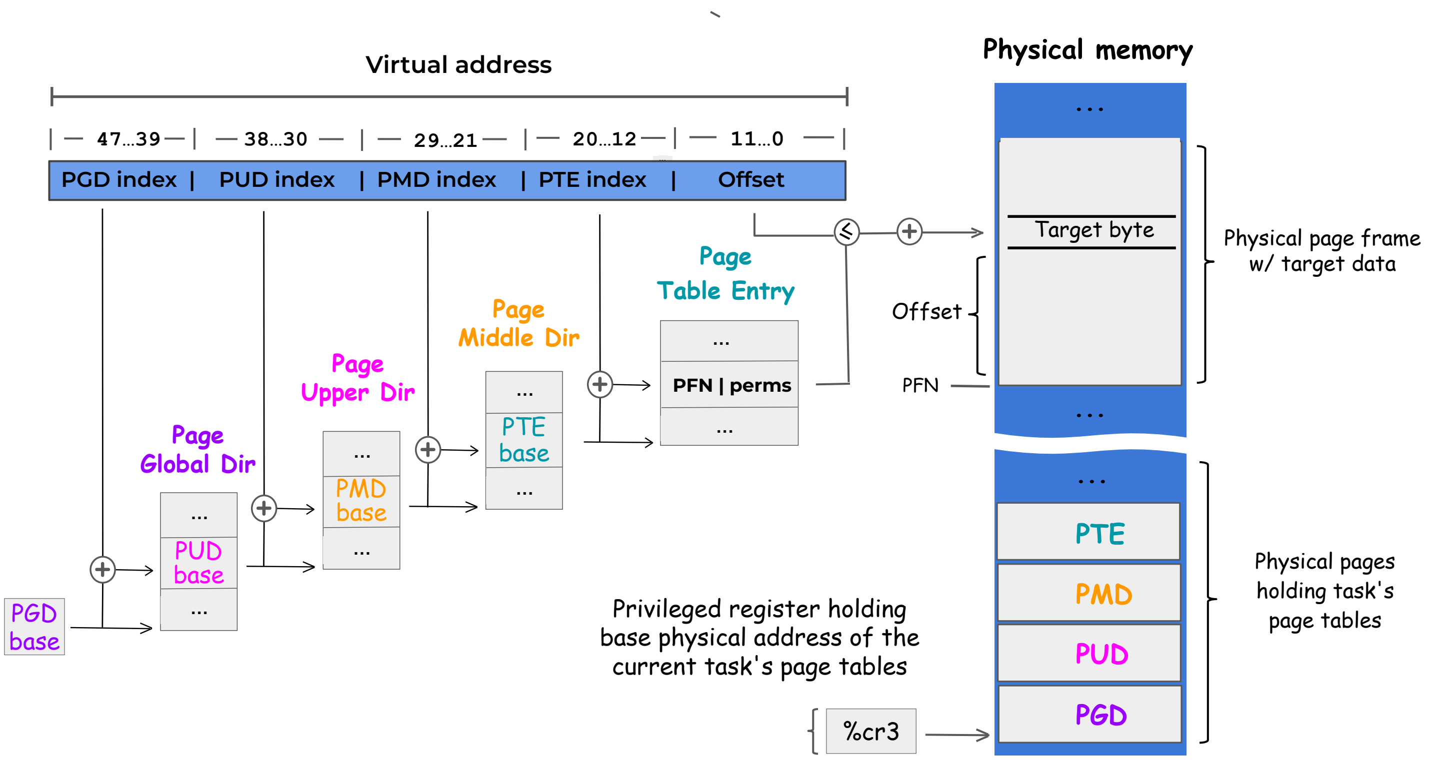
There is another positive side-effect...hang on...

Mechanism: Multi-level paging



> See relevant linux definitions: here

Mechanism: Multi-level paging



> What is the reach of one page of PTE/PMD/PUD/PGD entries? **Why care?**

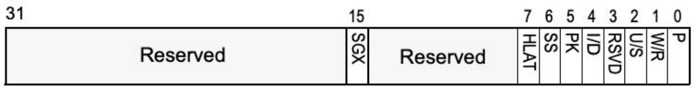
x86 PTE format that maps a 4KB page

63	62 … 59	58 … 52	51 … M[1]	M-1 … 12	11	10 … 9	8	7	6	5	4	3	2	1	0	
XD	Prot. Key	Ignored	Rsvd.	Address of 4KB page frame	R	Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	1	**PTE: 4KB page**
Ignored															0	**PTE: not present**

0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 5.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 5.8)

> See Intel Software Developer's Manual

Error codes on x86 Page fault (PGD @%cr3; faulting addr. @%cr2)



This will be placed to the stack by hardware

P	0	The fault was caused by a non-present page.
	1	The fault was caused by a page-level protection violation.
W/R	0	The access causing the fault was a read.
	1	The access causing the fault was a write.
U/S	0	A supervisor-mode access caused the fault.
	1	A user-mode access caused the fault.
RSVD	0	The fault was not caused by reserved bit violation.
	1	The fault was caused by a reserved bit set to 1 in some paging-structure entry.
I/D	0	The fault was not caused by an instruction fetch.
	1	The fault was caused by an instruction fetch.
PK	0	The fault was not caused by protection keys.
	1	There was a protection-key violation.
SS	0	The fault was not caused by a shadow-stack access.
	1	The fault was caused by a shadow-stack access.
HLAT	0	The fault occurred during ordinary paging or due to access rights.
	1	The fault occurred during HLAT paging.
SGX	0	The fault is not related to SGX.
	1	The fault resulted from violation of SGX-specific access-control requirements.

> See Intel Software Developer's Manual (p. 3213)

Checking the address space size of your machine

```
➜  ~ cat /proc/$$/maps | head
aaaaab8d0000-aaaaab9a3000 r-xp 00000000 08:02 2375001
aaaaab9b3000-aaaaab9b5000 r--p 000d3000 08:02 2375001
aaaaab9b5000-aaaaab9bb000 rw-p 000d5000 08:02 2375001
aaaaab9bb000-aaaaab9cf000 rw-p 00000000 00:00 0
aaaaac024000-aaaaac40b000 rw-p 00000000 00:00 0
ffffb82b0000-ffffb8530000 r--s 00000000 08:02 2890570
ffffb8530000-ffffb853f000 r-xp 00000000 08:02 2495184
ffffb853f000-ffffb854e000 ---p 0000f000 08:02 2495184
ffffb854e000-ffffb854f000 r--p 0000e000 08:02 2495184
ffffb854f000-ffffb8550000 rw-p 0000f000 08:02 2495184
```

- Why am I seeing 12 hexadecimal digits?

Checking the address space size of your machine

```
➜  ~ cat /proc/$$/maps | head
aaaaab8d0000-aaaaab9a3000 r-xp 00000000 08:02 2375001
aaaaab9b3000-aaaaab9b5000 r--p 000d3000 08:02 2375001
aaaaab9b5000-aaaaab9bb000 rw-p 000d5000 08:02 2375001
aaaaab9bb000-aaaaab9cf000 rw-p 00000000 00:00 0
aaaaac024000-aaaaac40b000 rw-p 00000000 00:00 0
ffffb82b0000-ffffb8530000 r--s 00000000 08:02 2890570
ffffb8530000-ffffb853f000 r-xp 00000000 08:02 2495184
ffffb853f000-ffffb854e000 ---p 0000f000 08:02 2495184
ffffb854e000-ffffb854f000 r--p 0000e000 08:02 2495184
ffffb854f000-ffffb8550000 rw-p 0000f000 08:02 2495184
➜  ~ cat /boot/config-`uname -r` | grep "VA_BITS\|PA_BITS"
# CONFIG_ARM64_VA_BITS_39 is not set
CONFIG_ARM64_VA_BITS_48=y
CONFIG_ARM64_VA_BITS=48
CONFIG_ARM64_PA_BITS_48=y
CONFIG_ARM64_PA_BITS=48
```

How is virtual memory implemented?

> Usually considered the most complex kernel subsystem

- It's simple if you view it as an index

Ask the right questions

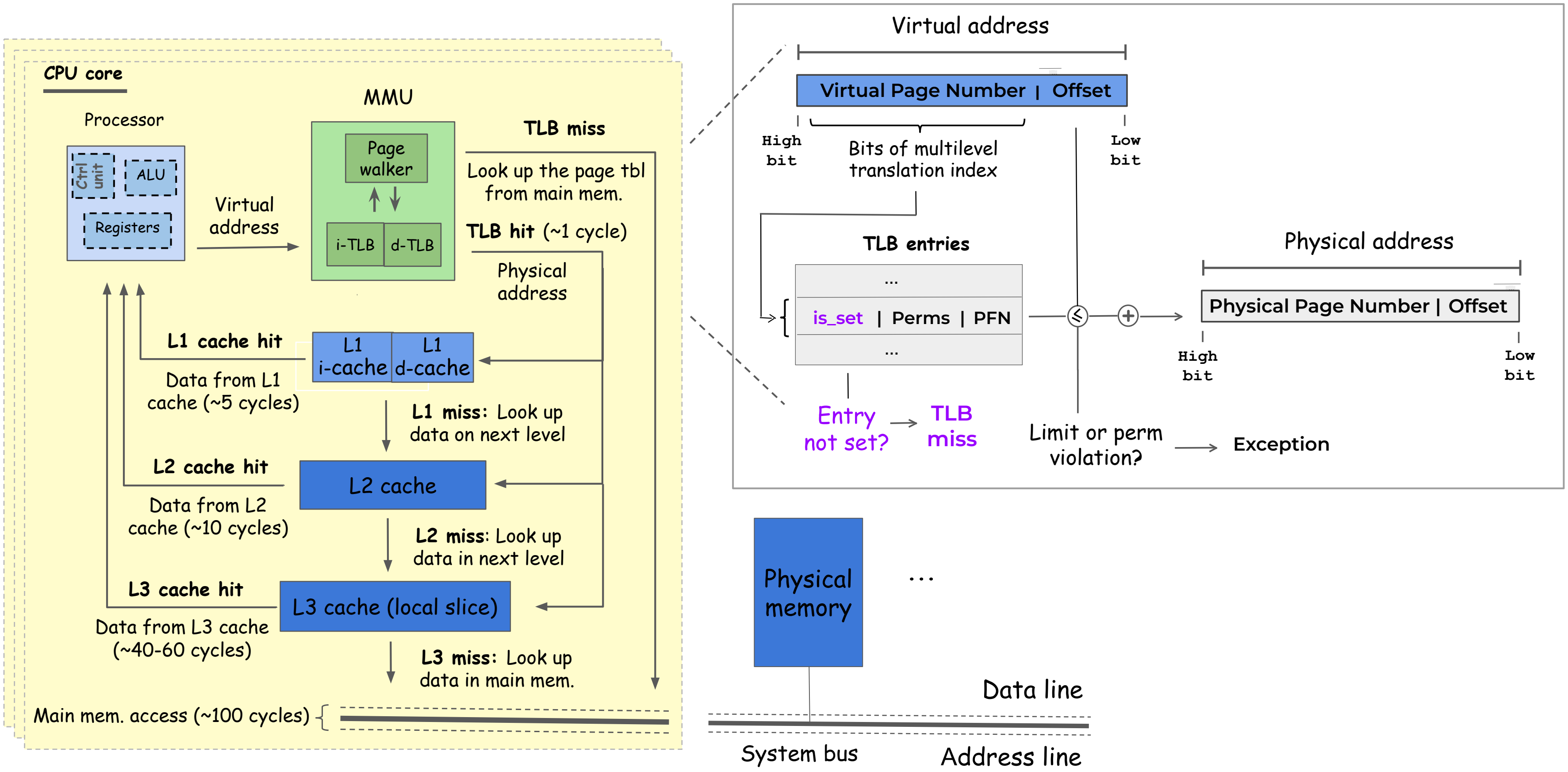
~~Q1: What are the entries of the index?~~

~~Q2: How are the entries of the index used?~~

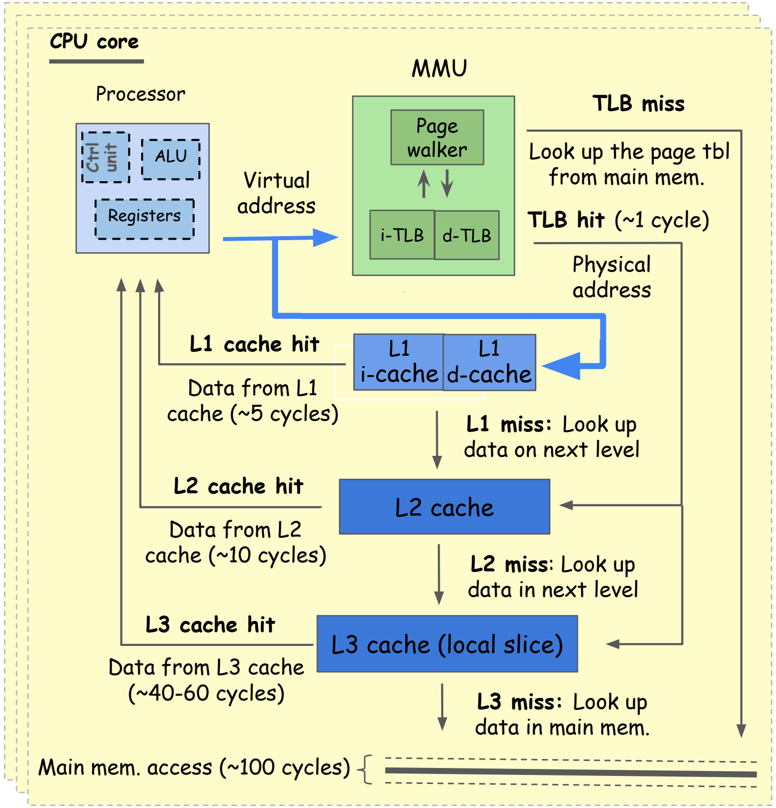
~~Q3: How are the entries of the index allocated?~~

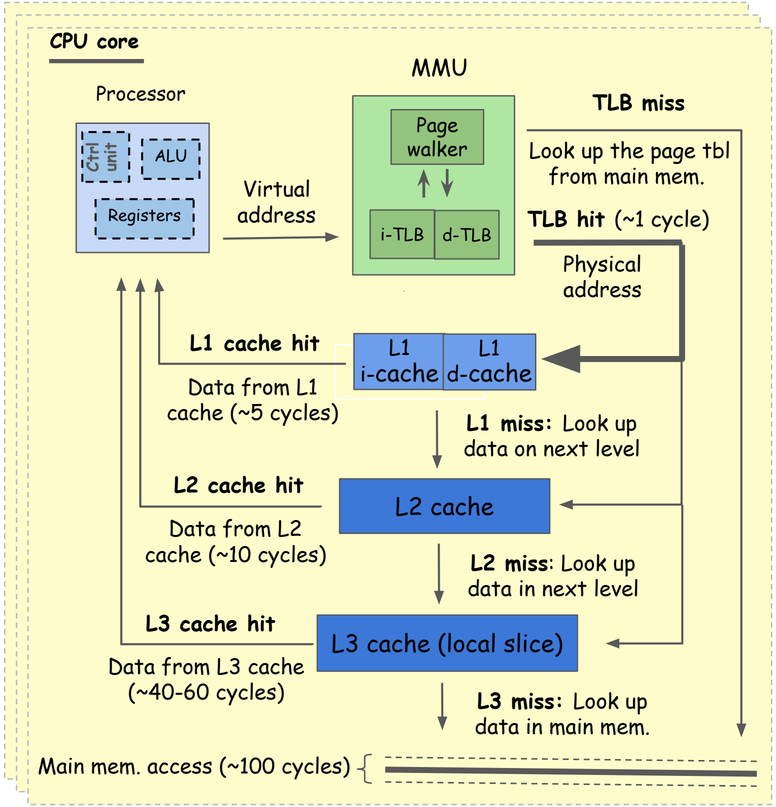
Q4: How are the entries of the index replaced?

Translation Lookaside Buffer (TLB)

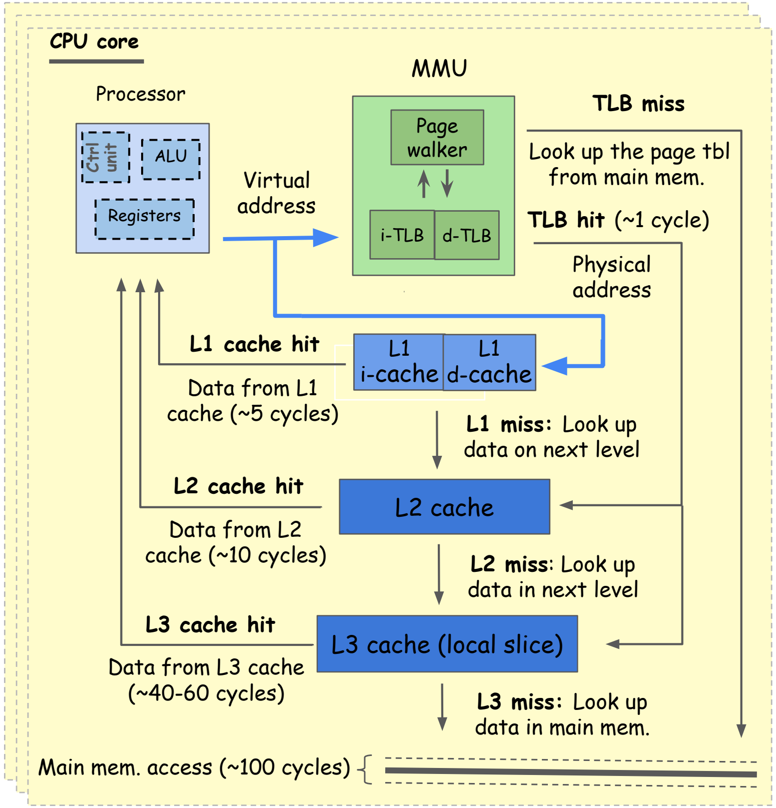


L1 cache: Why not virtually indexed <u>and</u> tagged?





L1 cache: Why not virtually indexed <u>and</u> tagged?



Homonym problem: After a context switch a proc can access leftover cache contents not mapped to its virtual address space (data leakage)

>> Flush caches on ctx switch? Slow

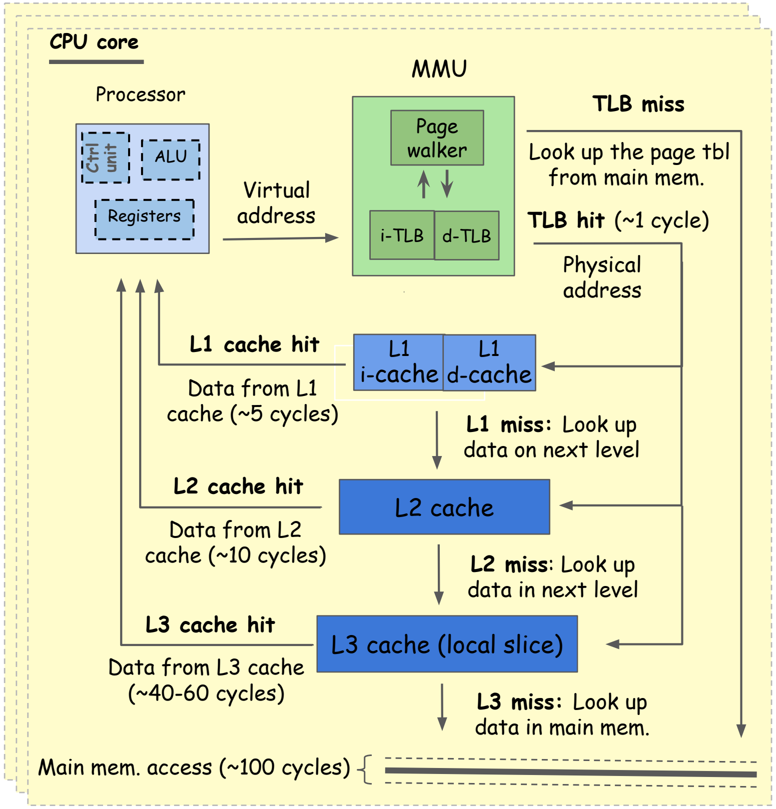
>> Use Addr. Space Ids in cache tags? Wasteful

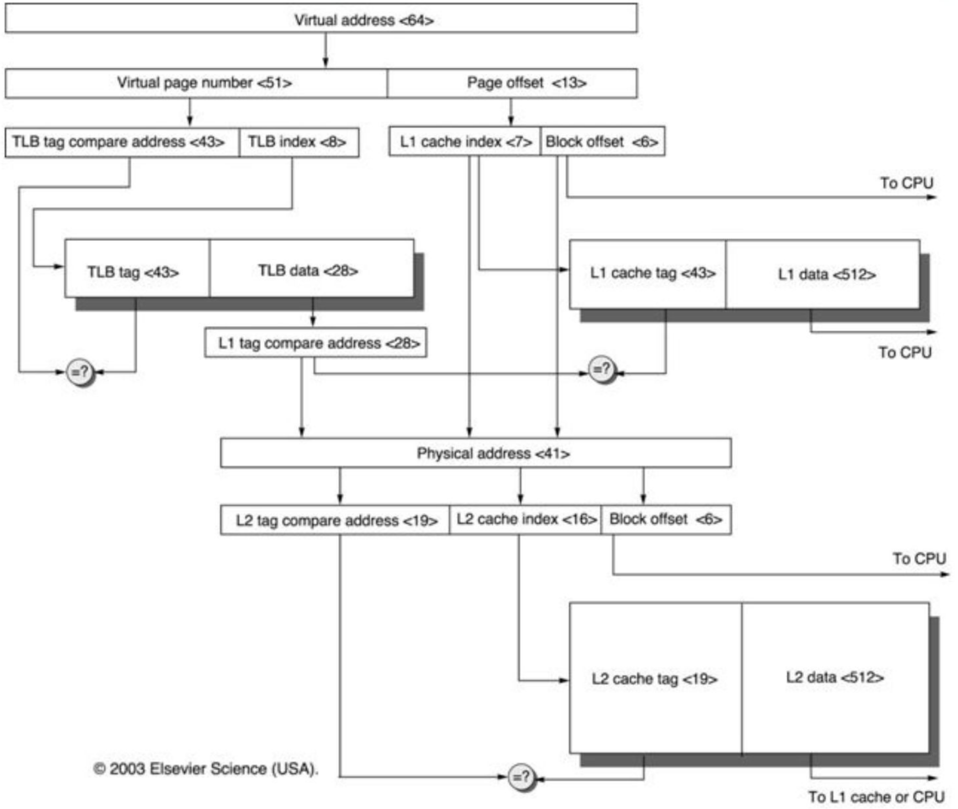
Synonym problem (a.k.a aliasing): Different virtual addresses reference the same data => OS decision => Invisible to hardware's cache coherency protocols...

>> Page coloring (beyond scope)

>> Use VIPT or PIPT caches

L1 cache: Virtually-indexed / Physically-tagged





How is virtual memory implemented?

> Usually considered the most complex kernel subsystem

- It's simple if you view it as an index

Ask the right questions

~~Q1: What are the entries of the index?~~

~~Q2: How are the entries of the index used?~~

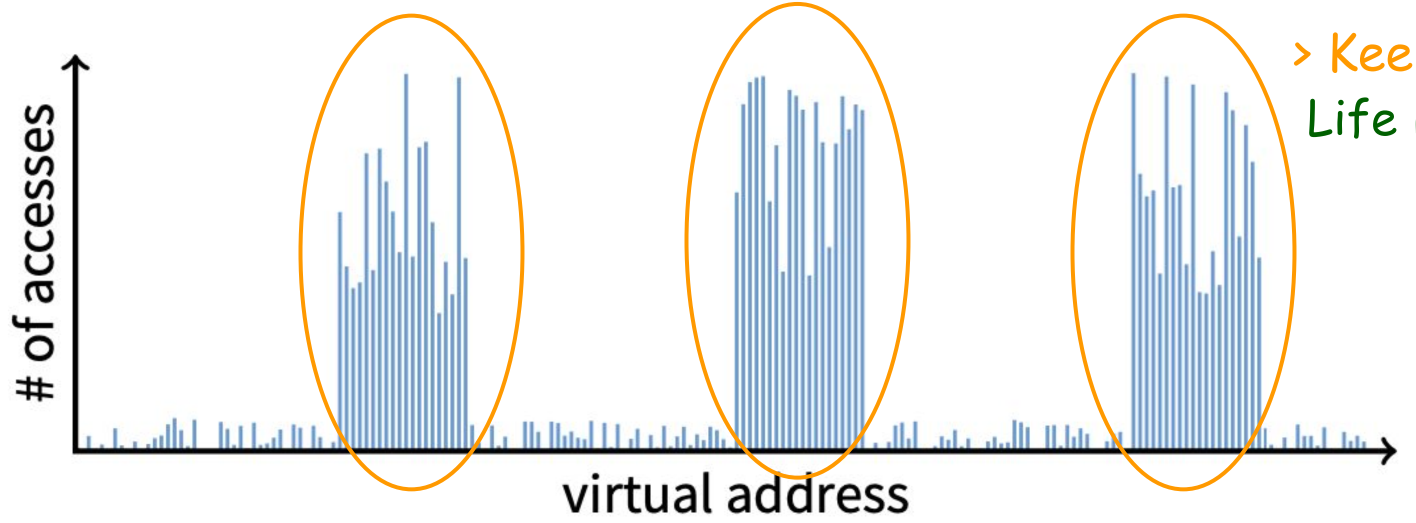
~~**Q3: How are the entries of the index allocated?**~~

~~Q4: How are the entries of the index replaced?~~

The working set model for program behavior

> A process can be in main memory iff all the pages it is currently using can be in main memory, by P. Denning (1968)

Rule of thumb: 20% of memory gets 80% of accesses

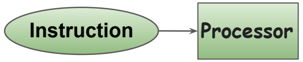


> Keep these "hot"?

Life is good...

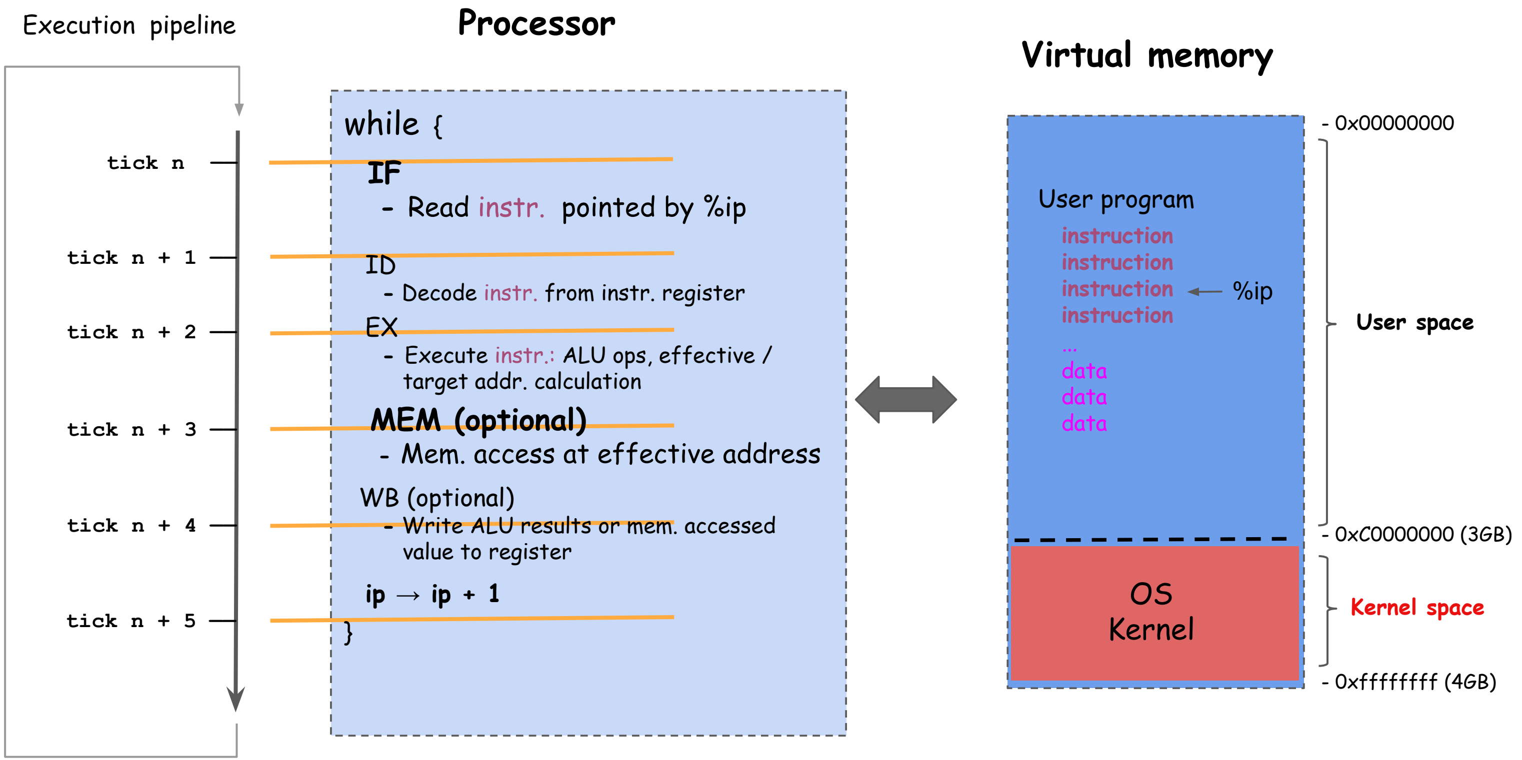
Demand paging via page fault handling

Demand paging via page fault handling

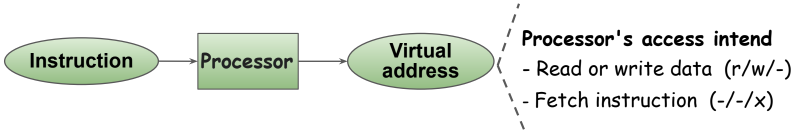


Demand paging via page fault handling

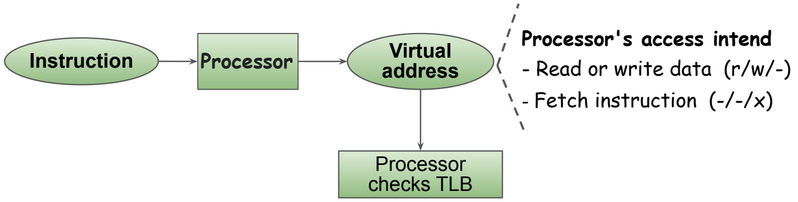
When may the processor access memory?



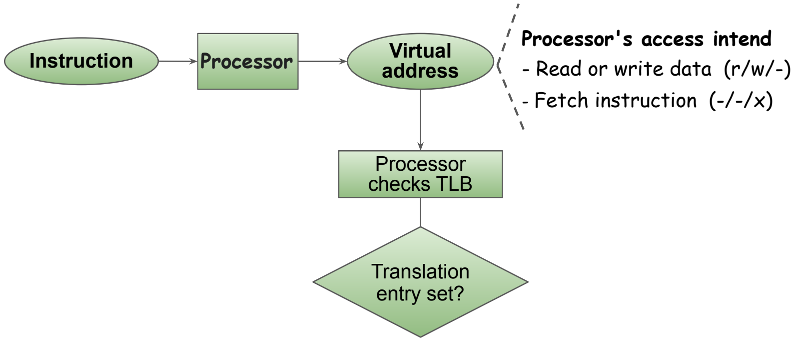
Demand paging via page fault handling



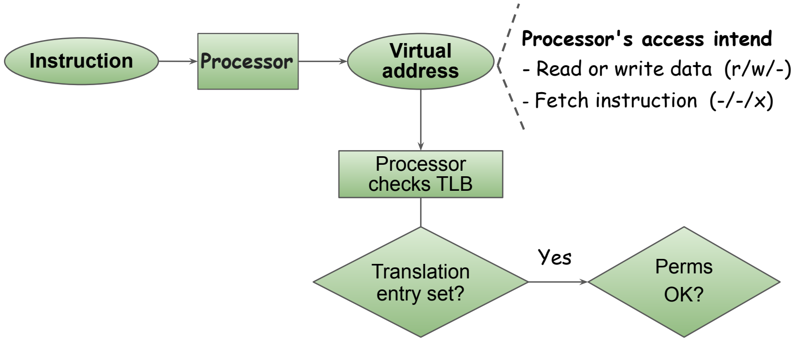
Demand paging via page fault handling



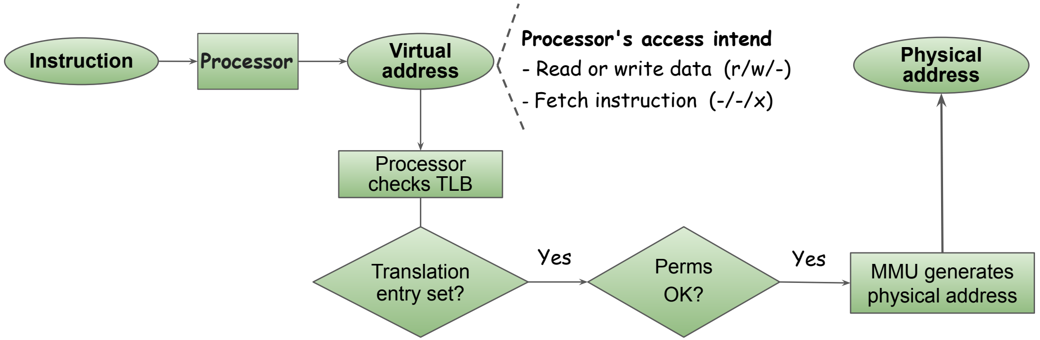
Demand paging via page fault handling



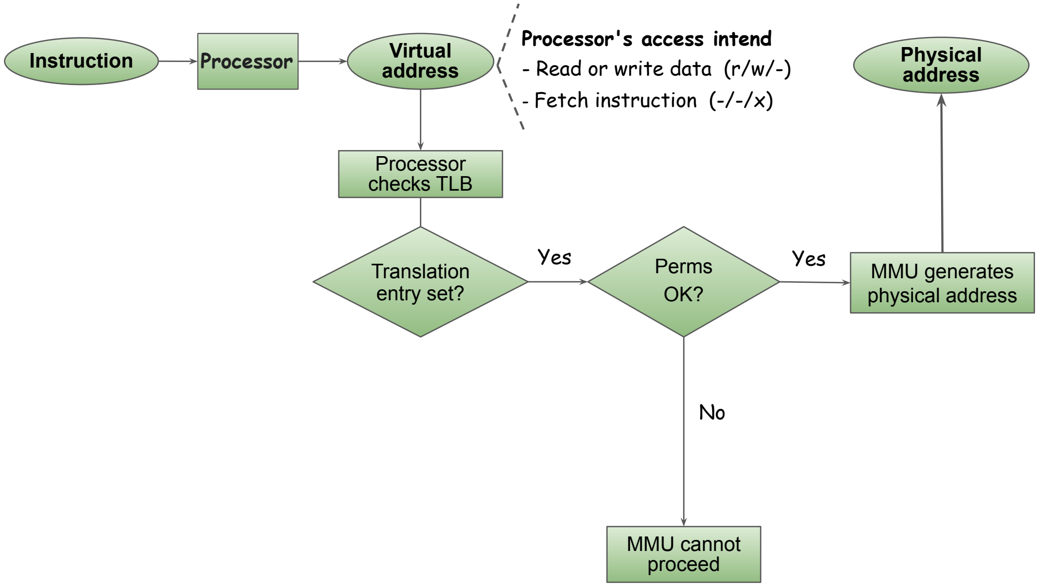
Demand paging via page fault handling



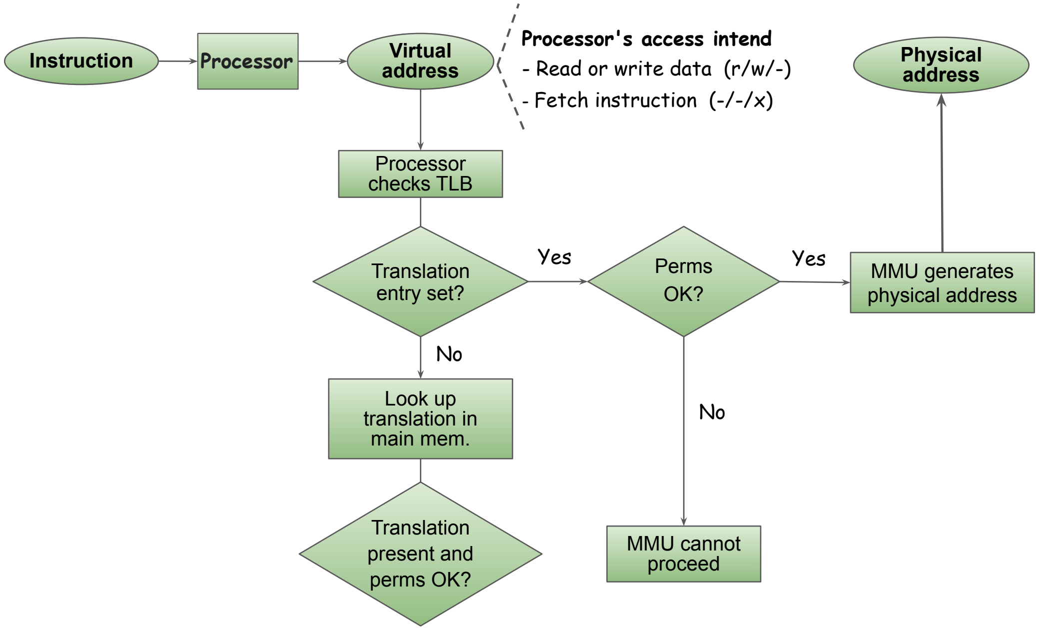
Demand paging via page fault handling



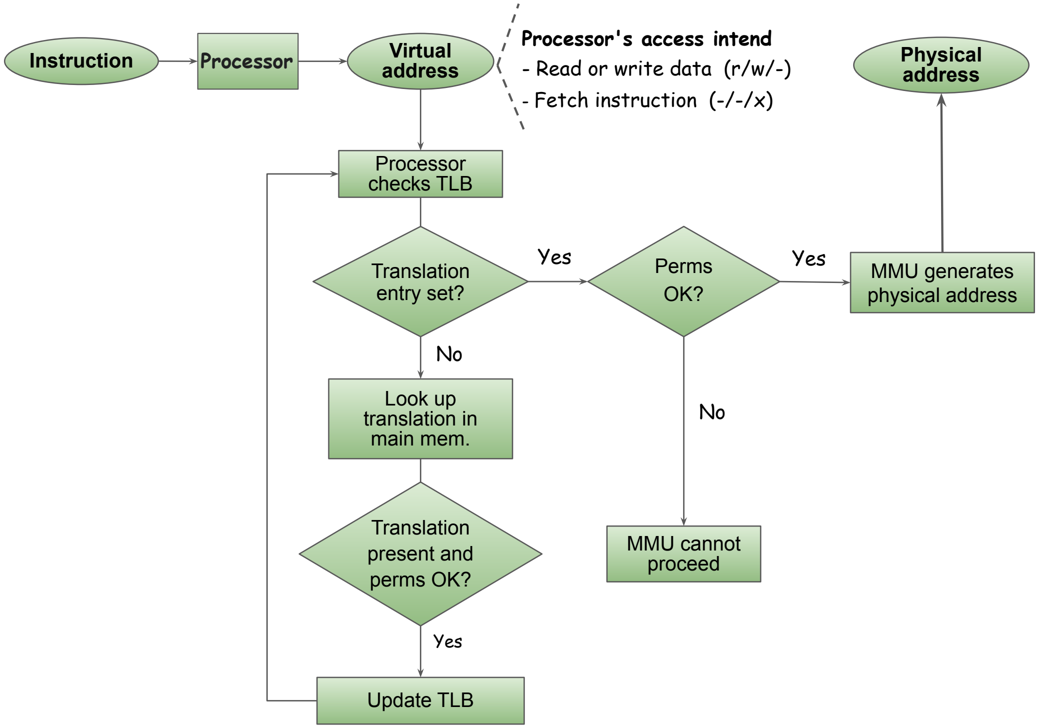
Demand paging via page fault handling



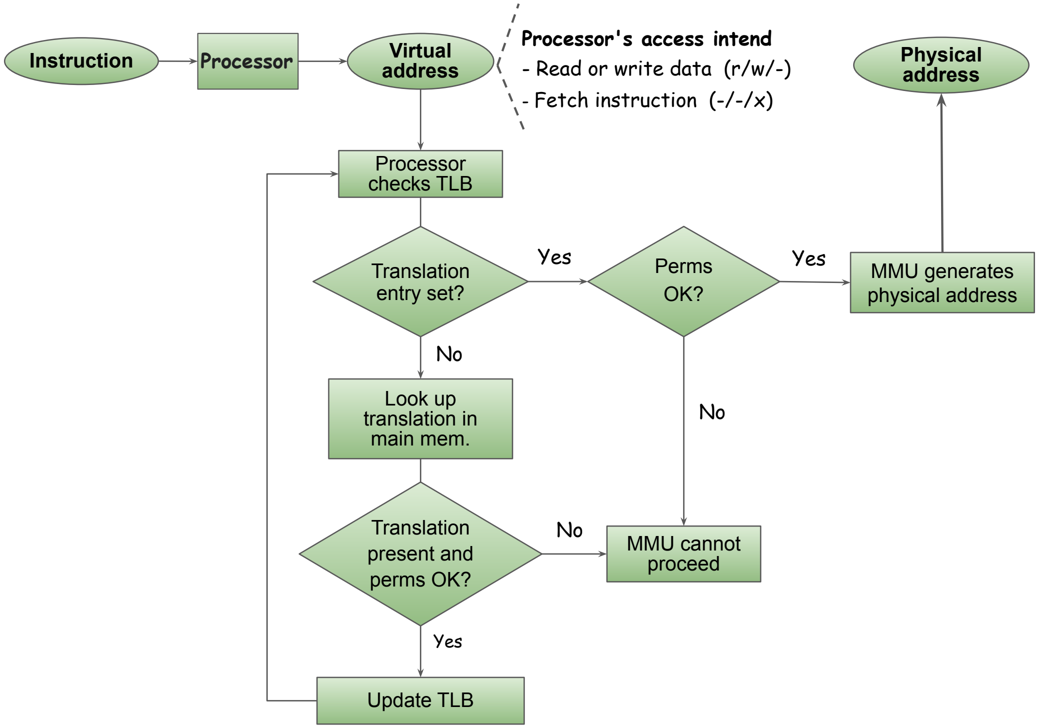
Demand paging via page fault handling



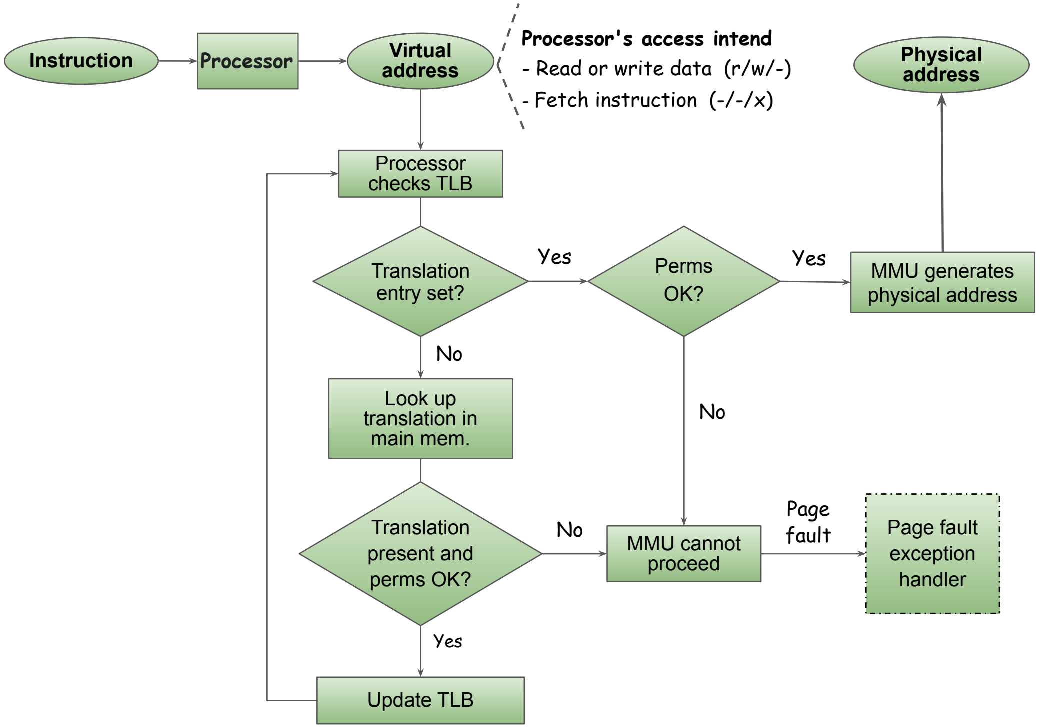
Demand paging via page fault handling



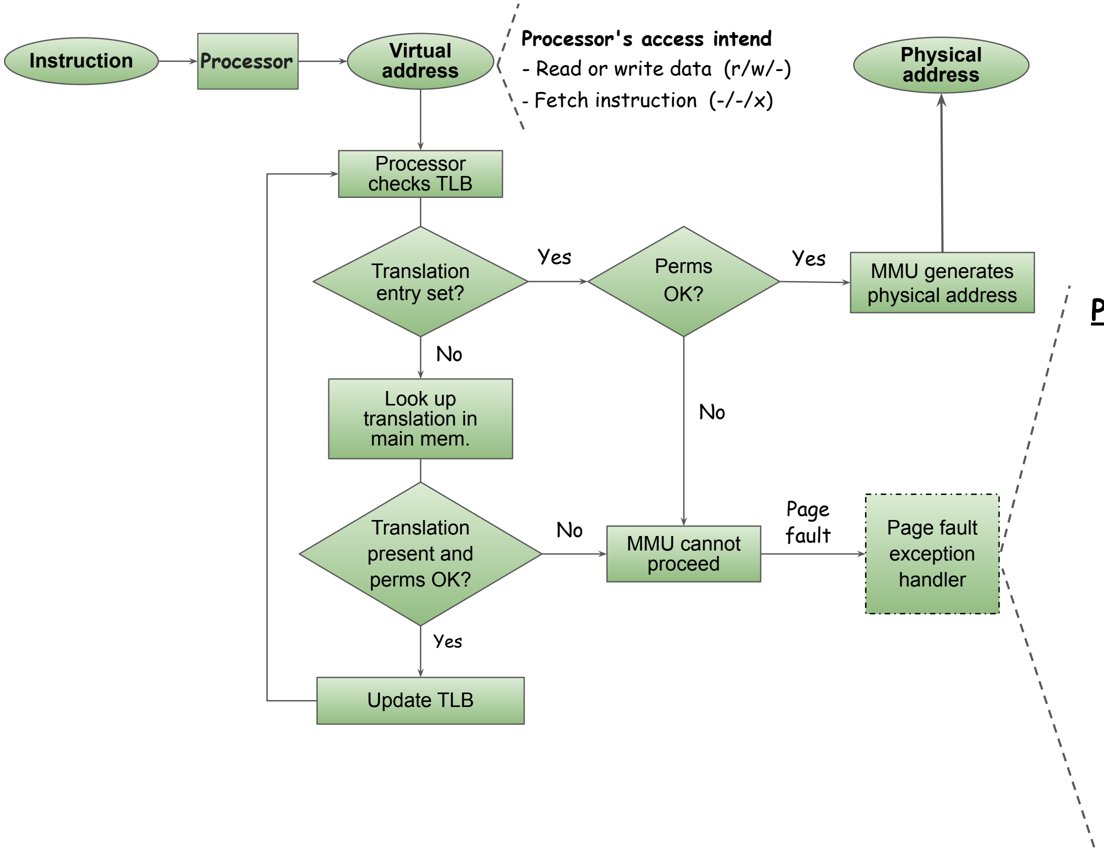
Demand paging via page fault handling



Demand paging via page fault handling

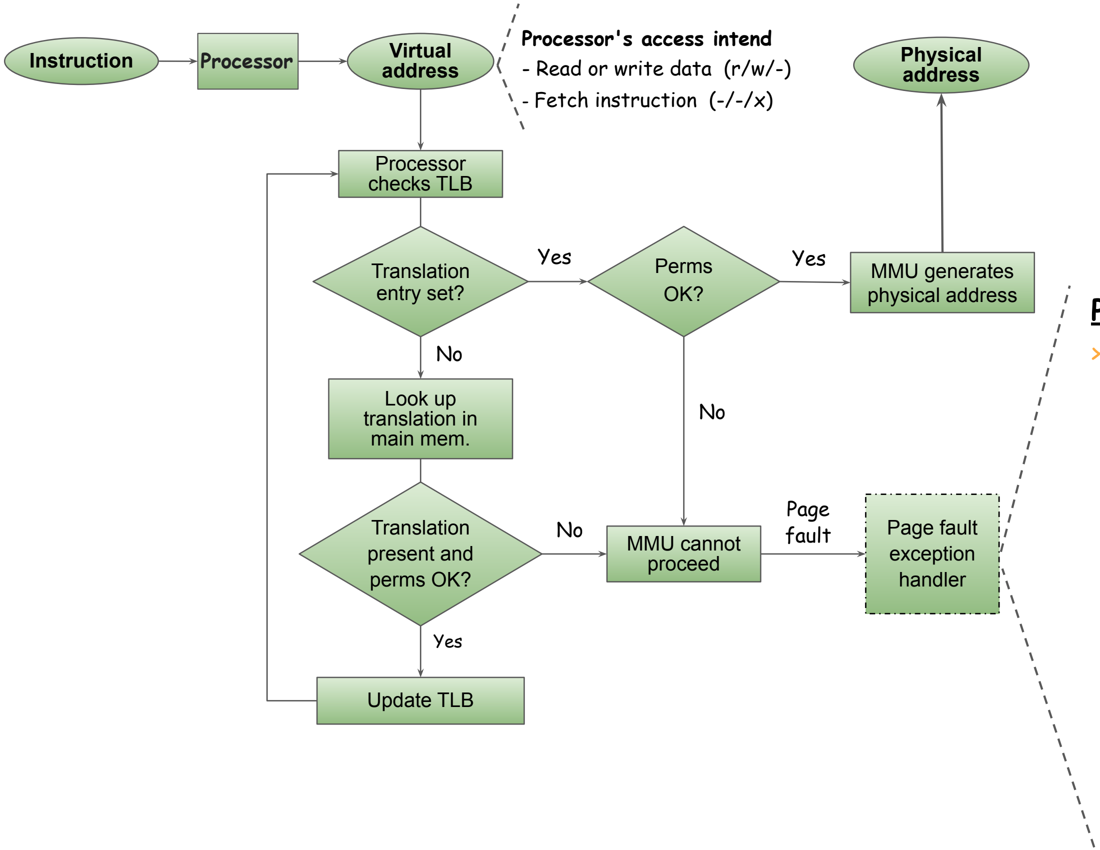


Demand paging via page fault handling



Page fault handling (see here and here)

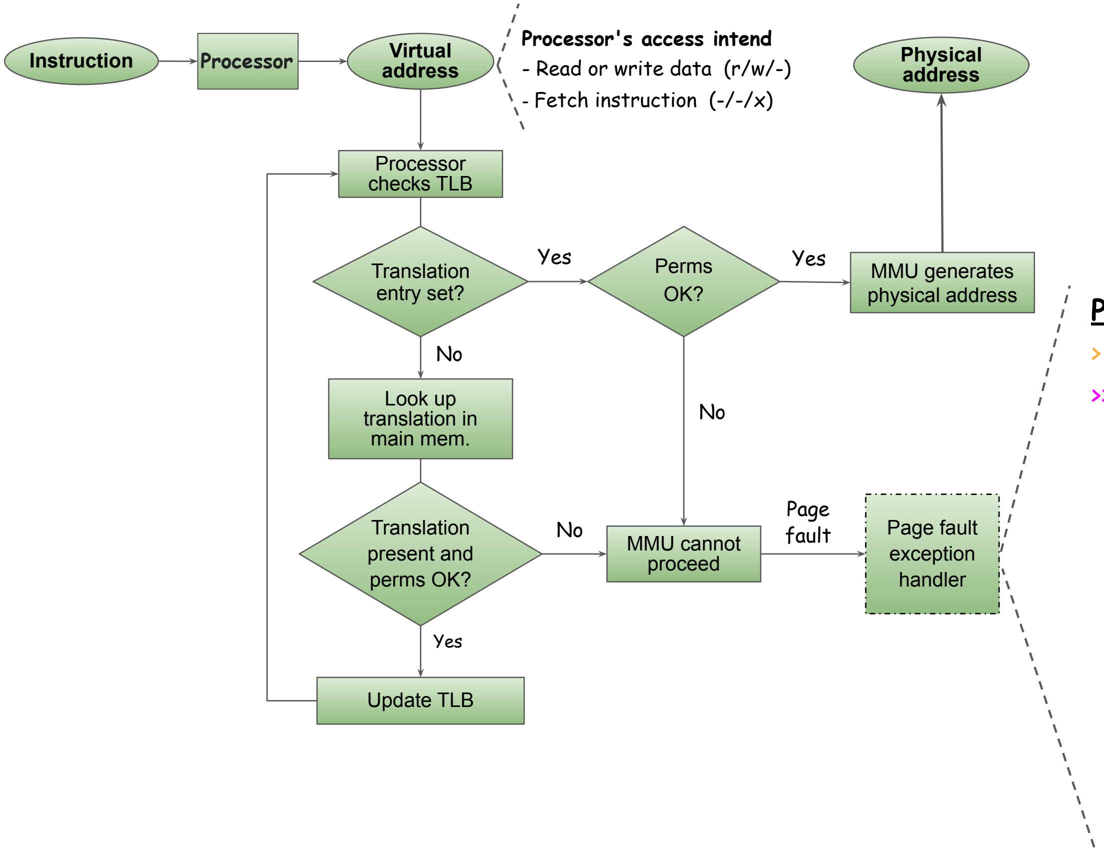
Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

Demand paging via page fault handling

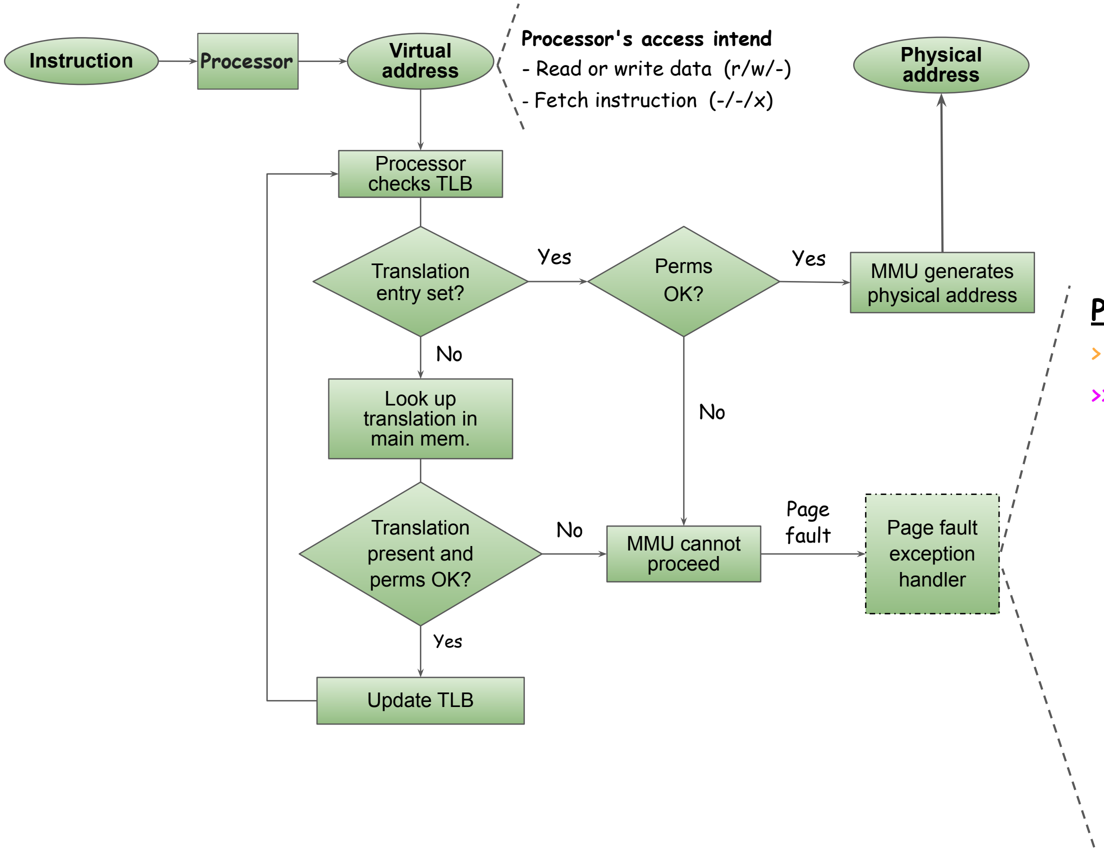


Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

Demand paging via page fault handling



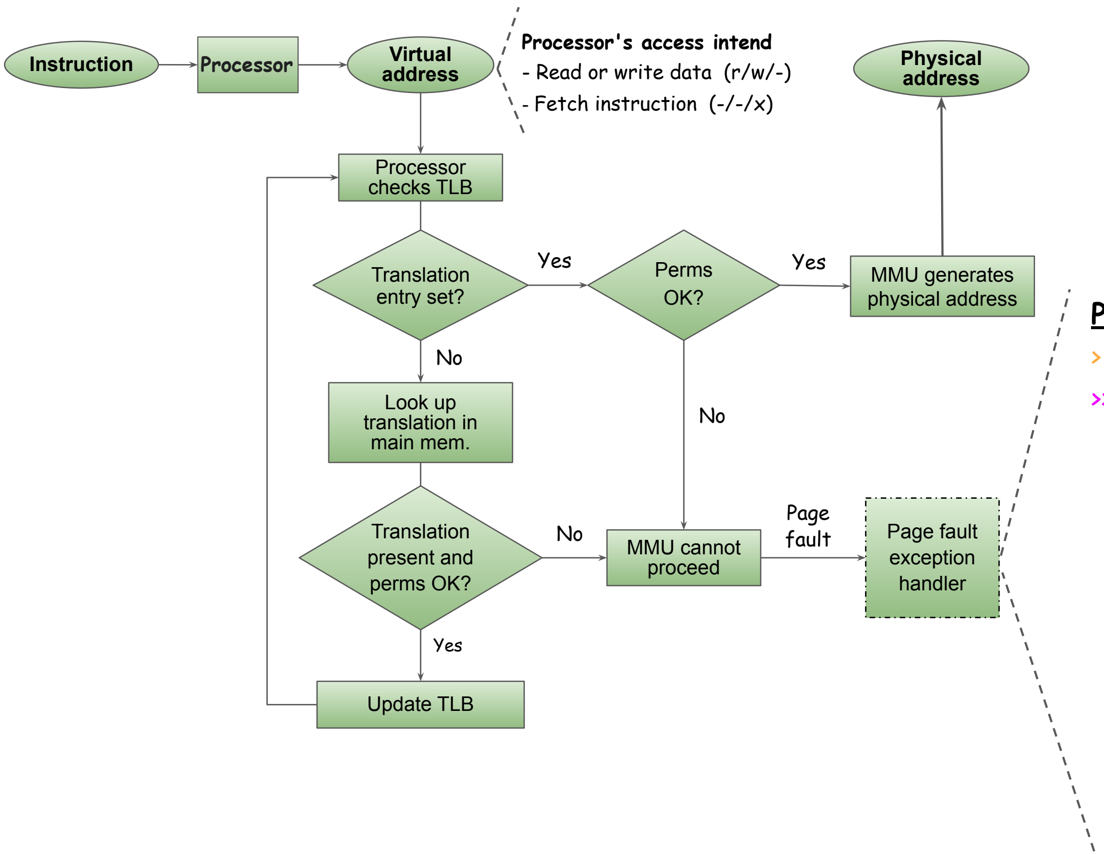
Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

Demand paging via page fault handling



Page fault handling (see here and here)

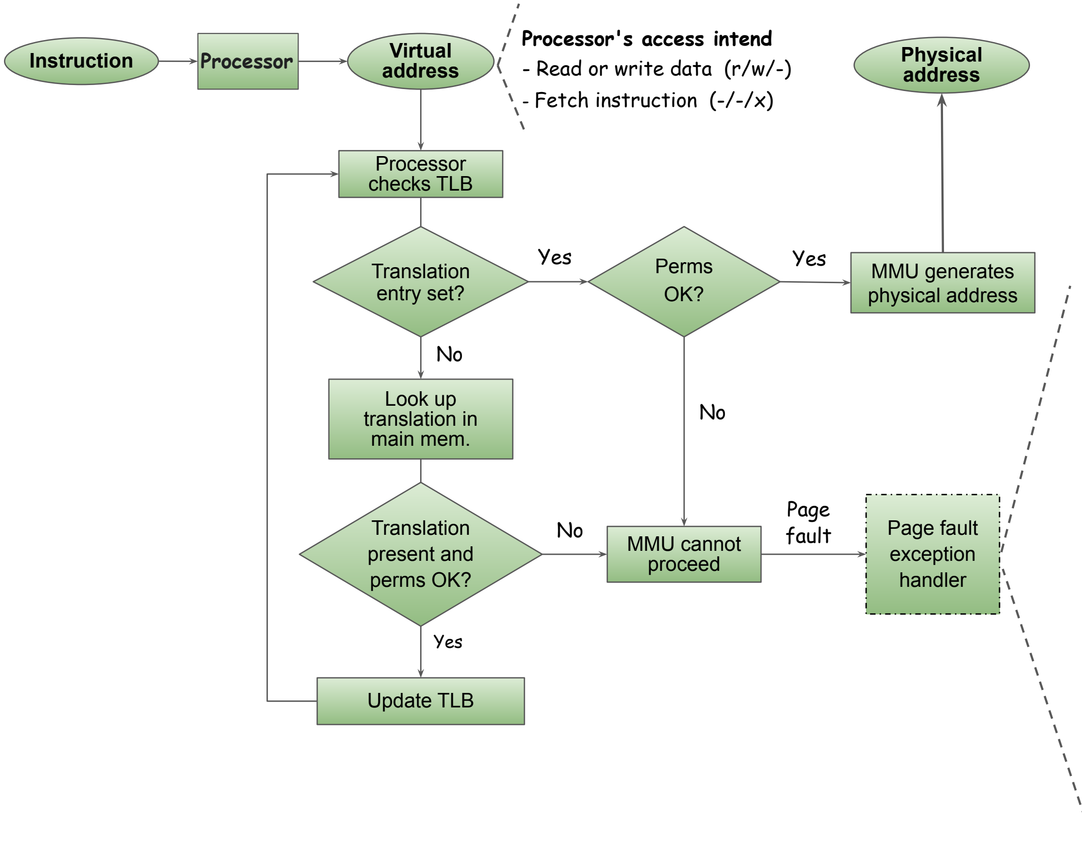
> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

Demand paging via page fault handling



Page fault handling (see here and here)

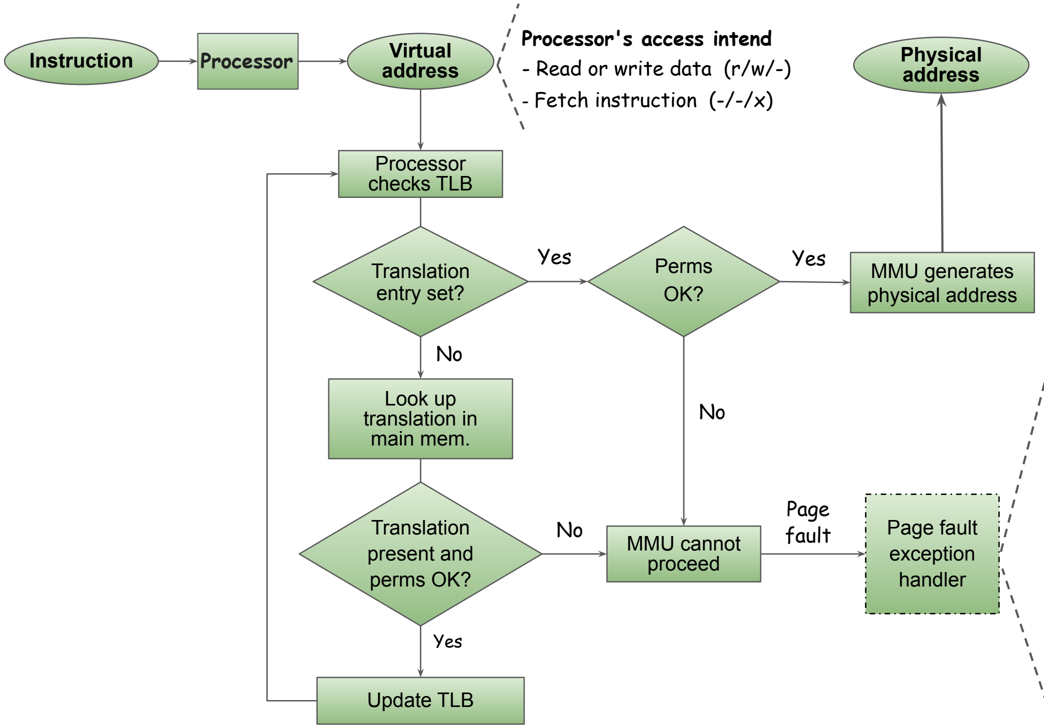
\> Access **not** permitted (see here) => SIGSEGV

\>> Translation entry not set (see here)

\>>> Not file-backed VMA (see here)
do_anonymous_page=> Minor p.f

\>>> File-backed VMA (see here)

Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

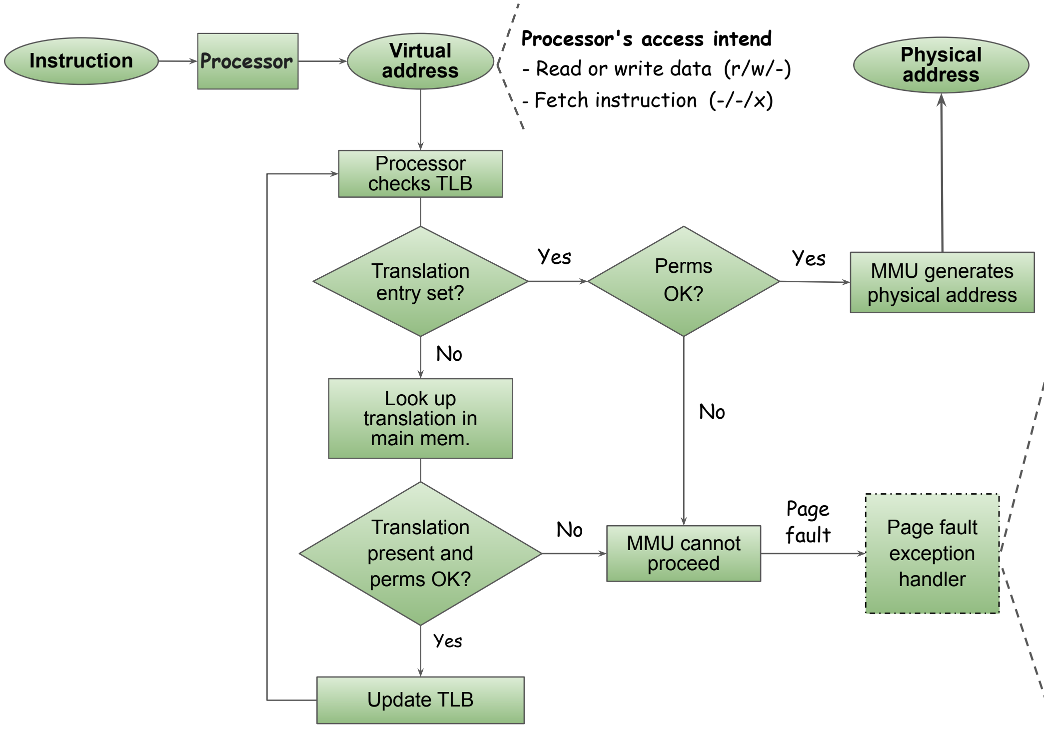
>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

>>> File-backed VMA (see here)

do_cow_fault => Minor p.f

Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

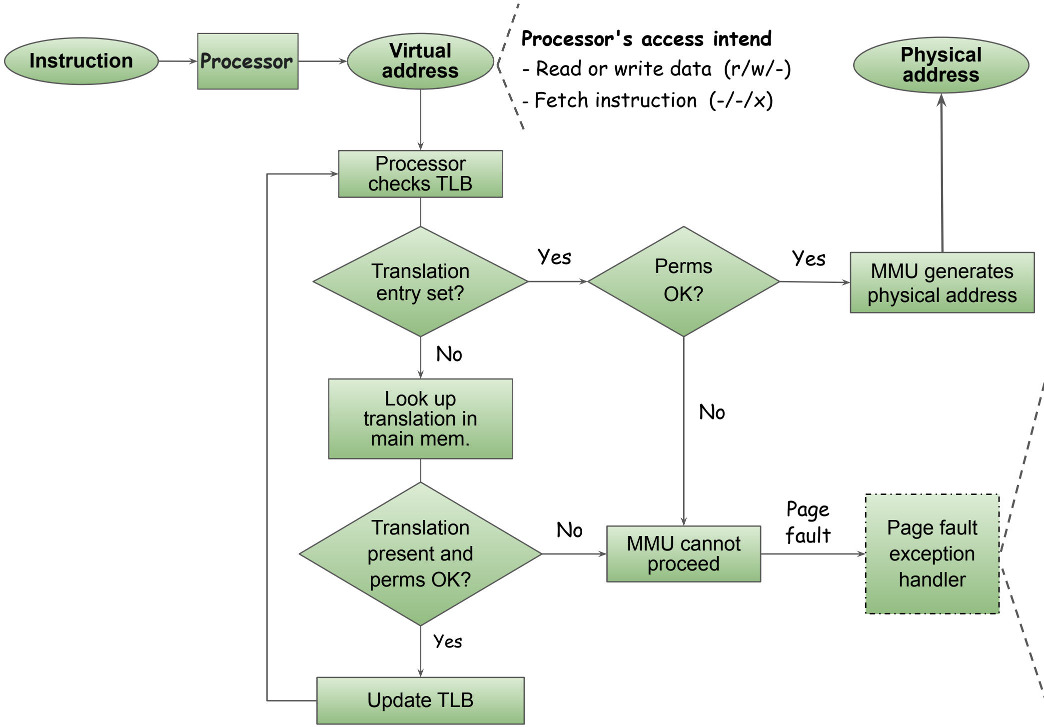
do_anonymous_page=> Minor p.f

>>> File-backed VMA (see here)

do_cow_fault => Minor p.f

do_read_fault => Minor or Major p.f.

Demand paging via page fault handling



<u>Page fault handling</u> (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

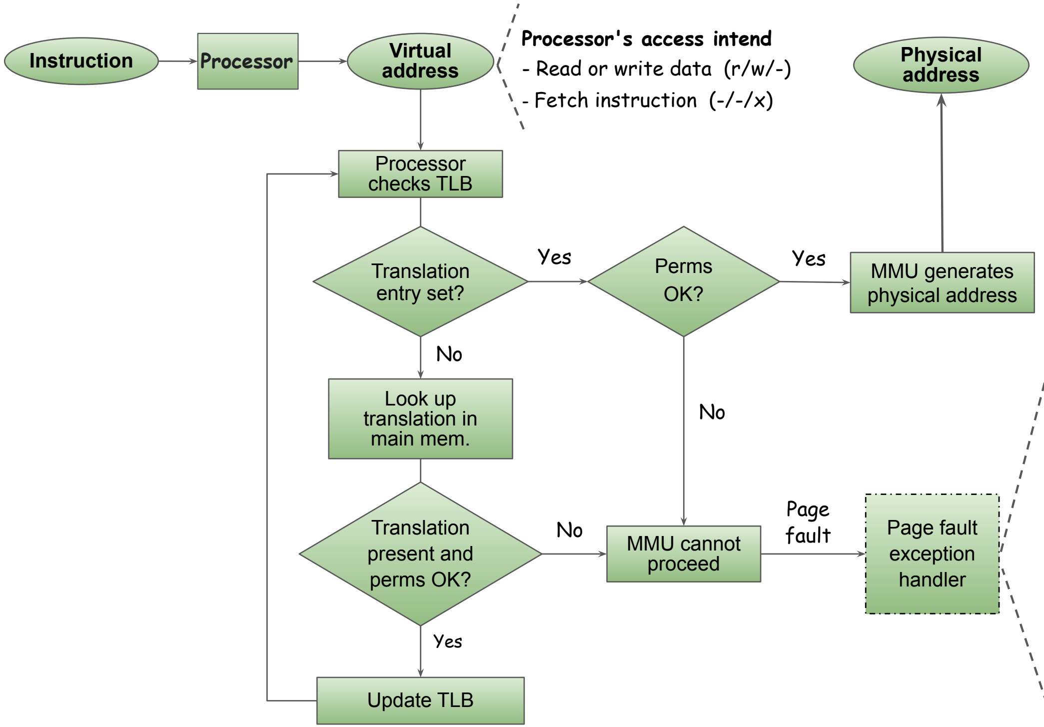
>>> File-backed VMA (see here)

do_cow_fault => Minor p.f

do_read_fault => Minor or Major p.f.

do_shared_fault => Minor or Major p.f.

Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

>>> File-backed VMA (see here)

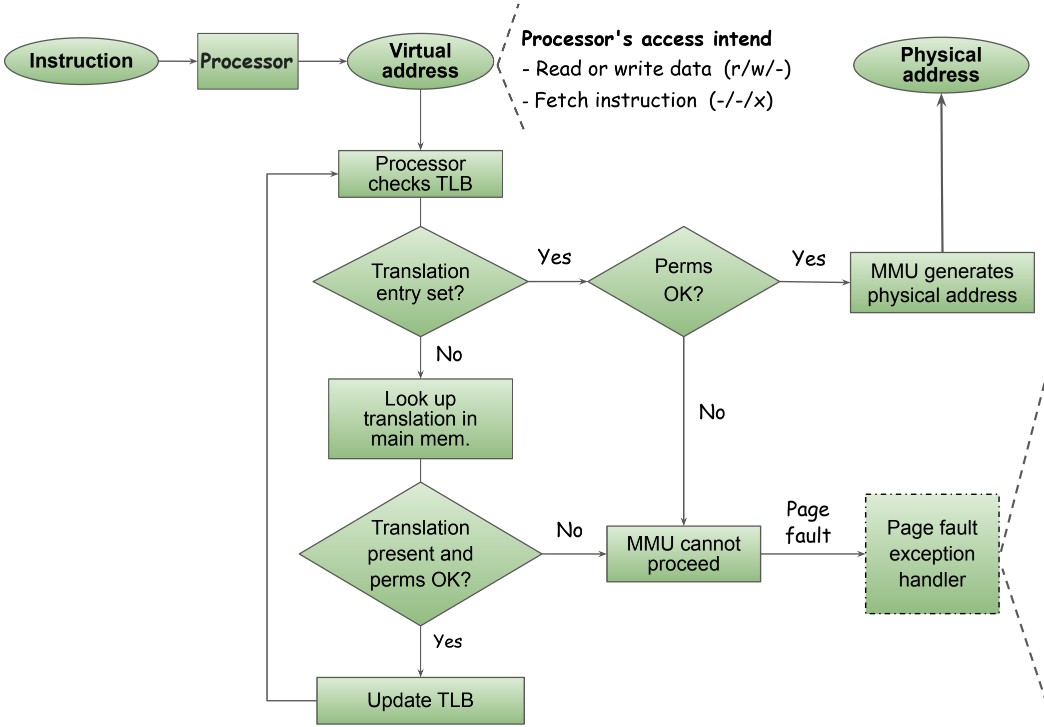
do_cow_fault => Minor p.f

do_read_fault => Minor or Major p.f.

do_shared_fault => Minor or Major p.f.

>> Translation entry **set** (see here)

Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

>>> File-backed VMA (see here)

do_cow_fault => Minor p.f

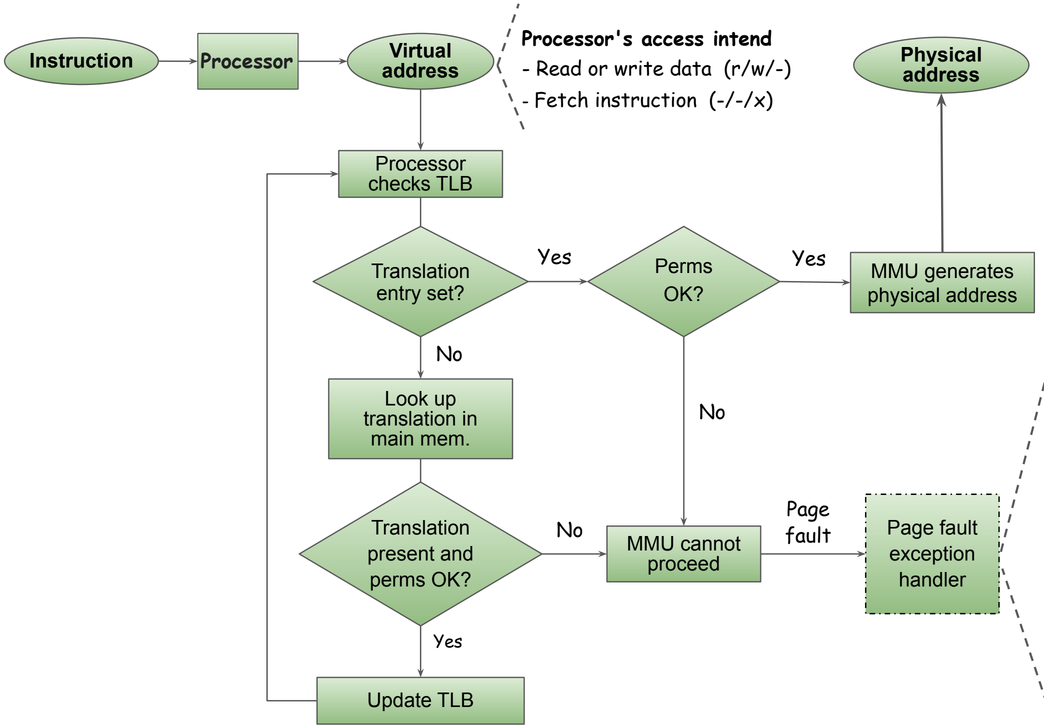
do_read_fault => Minor or Major p.f.

do_shared_fault => Minor or Major p.f.

>> Translation entry **set** (see here)

>>> "present" bit off (see here)

Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

>>> File-backed VMA (see here)

do_cow_fault => Minor p.f

do_read_fault => Minor or Major p.f.

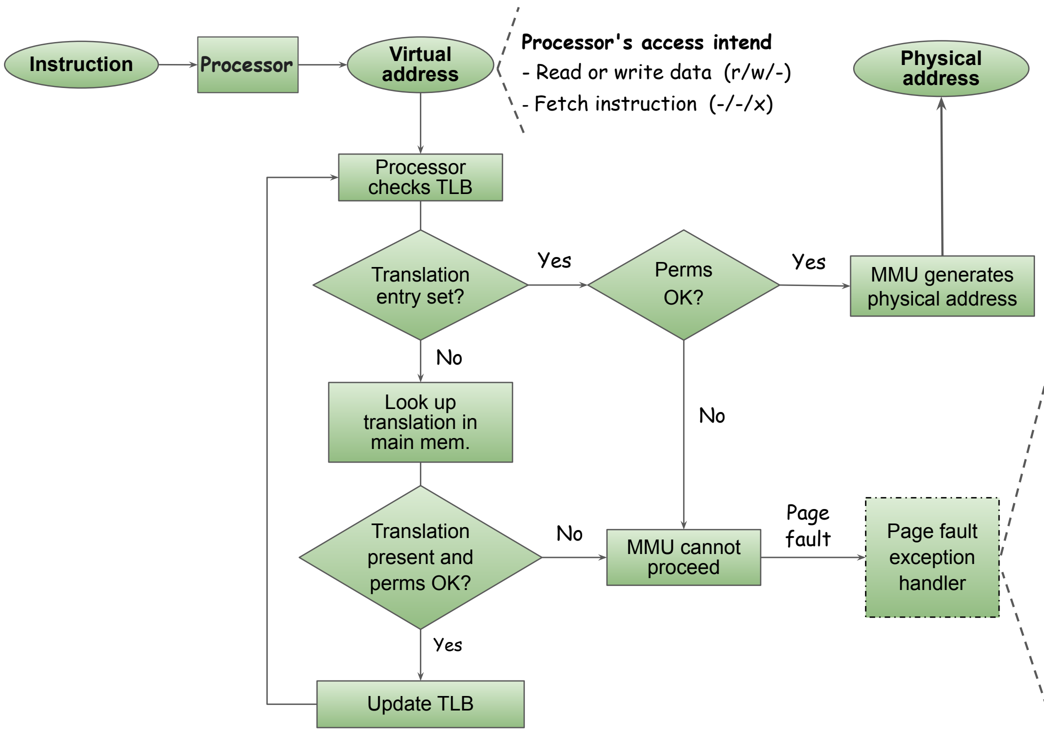
do_shared_fault => Minor or Major p.f.

>> Translation entry **set** (see here)

>>> "present" bit off (see here)

do_swap_page => Major p.f.

Demand paging via page fault handling



<u>Page fault handling</u> (see <u>here</u> and <u>here</u>)

> Access **not** permitted (see <u>here</u>) => SIGSEGV

>> Translation entry not set (see <u>here</u>)

>>> Not file-backed VMA (see <u>here</u>)

do_anonymous_page=> Minor p.f

>>> File-backed VMA (see <u>here</u>)

do_cow_fault => Minor p.f

do_read_fault => Minor or Major p.f.

do_shared_fault => Minor or Major p.f.

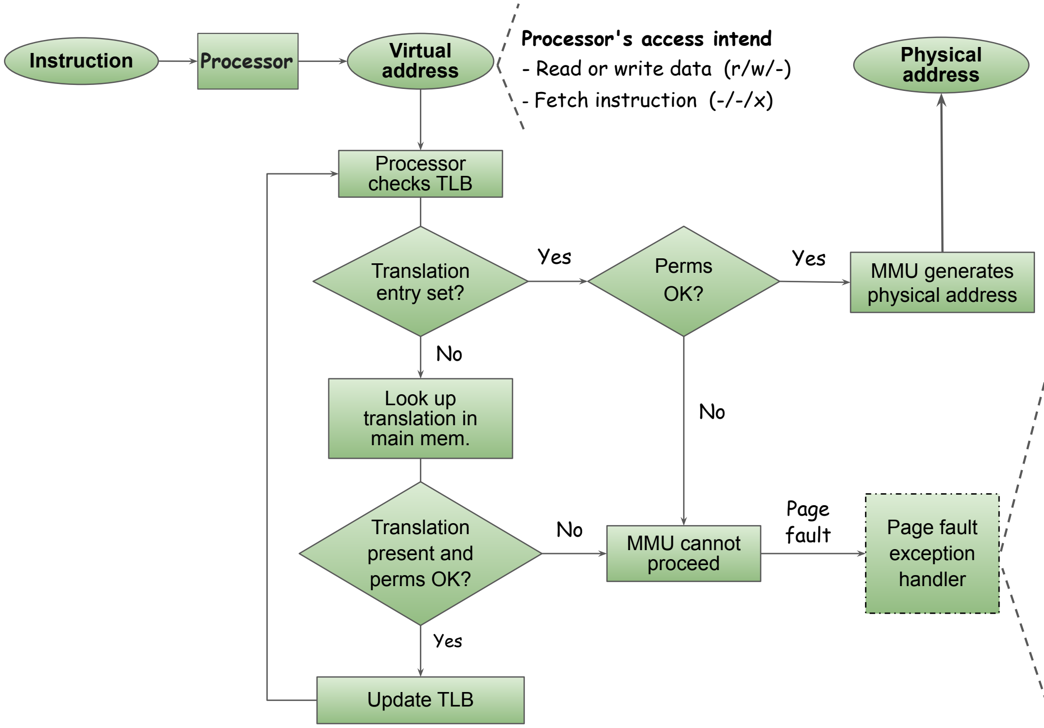
>> Translation entry **set** (see <u>here</u>)

>>> "present" bit off (see <u>here</u>)

do_swap_page => Major p.f.

>>> "write" bit off (see <u>here</u>)

Demand paging via page fault handling



<u>Page fault handling</u> (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

>>> File-backed VMA (see here)

do_cow_fault => Minor p.f

do_read_fault => Minor or Major p.f.

do_shared_fault => Minor or Major p.f.

>> Translation entry **set** (see here)

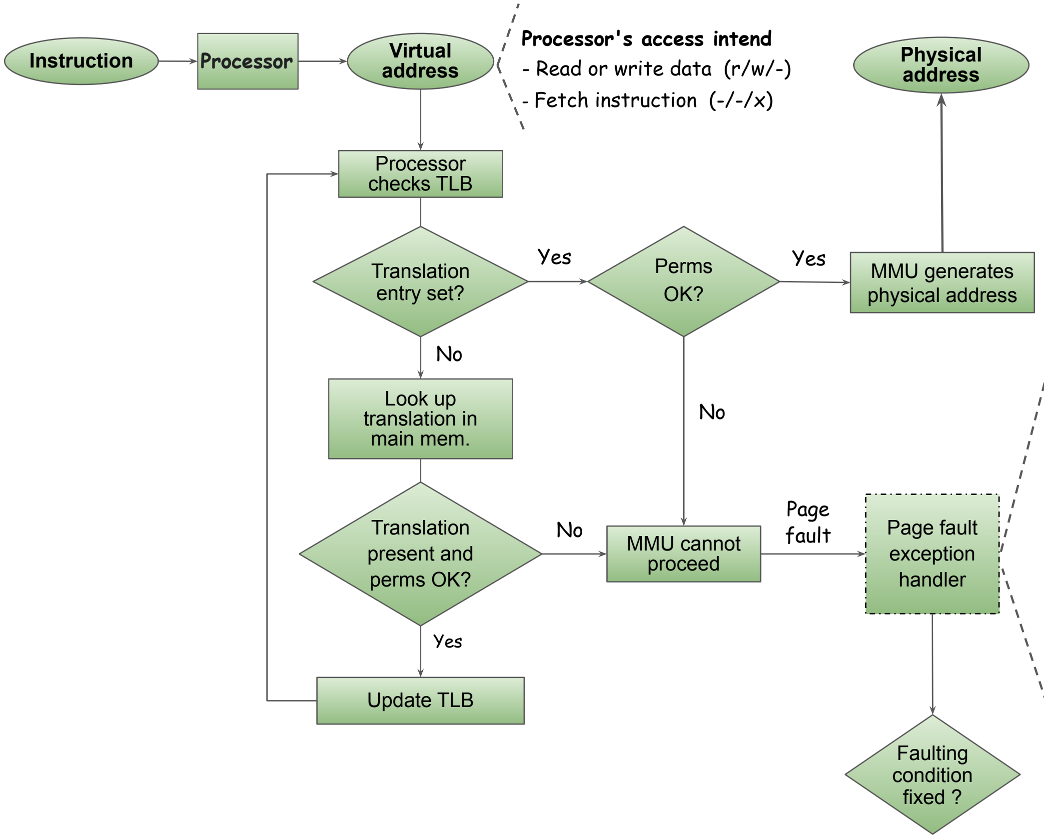
>>> "present" bit off (see here)

do_swap_page => Major p.f.

>>> "write" bit off (see here)

do_wp_page => Minor p.f.

Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

>>> File-backed VMA (see here)

do_cow_fault => Minor p.f

do_read_fault => Minor or Major p.f.

do_shared_fault => Minor or Major p.f.

>> Translation entry **set** (see here)

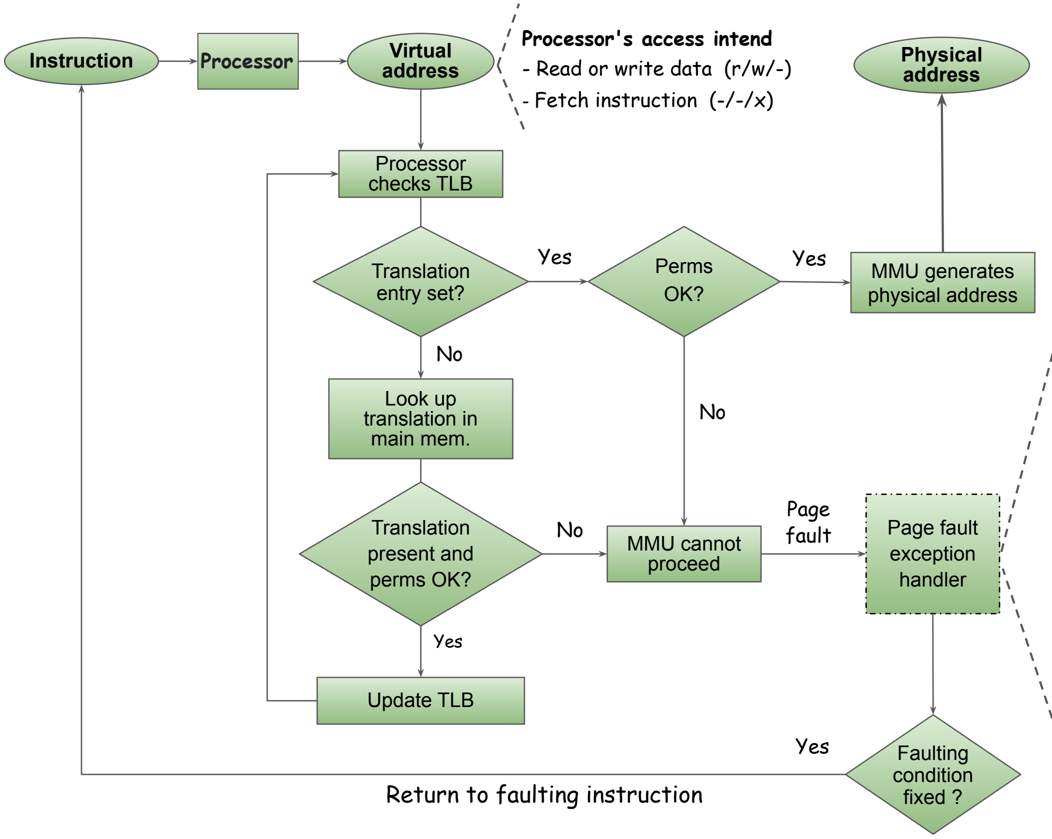
>>> "present" bit off (see here)

do_swap_page => Major p.f.

>>> "write" bit off (see here)

do_wp_page => Minor p.f.

Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)
do_anonymous_page=> Minor p.f

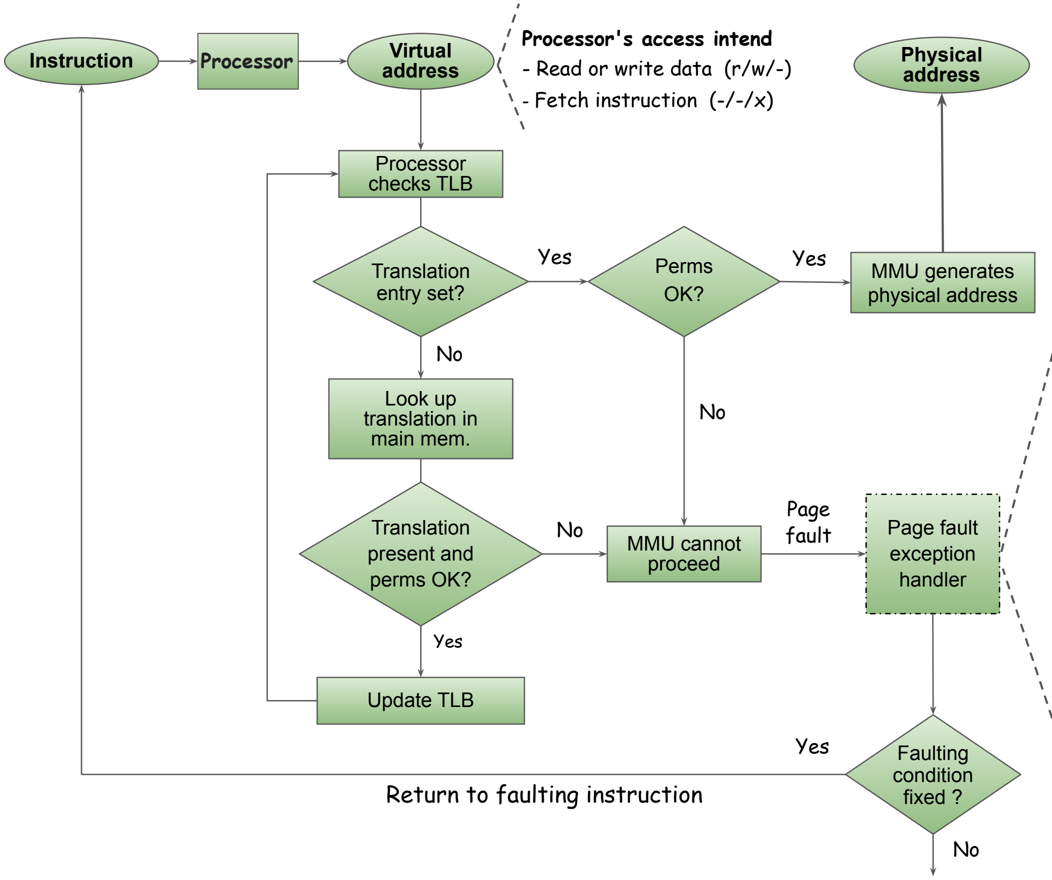
>>> File-backed VMA (see here)
do_cow_fault => Minor p.f
do_read_fault => Minor or Major p.f.
do_shared_fault => Minor or Major p.f.

>> Translation entry **set** (see here)

>>> "present" bit off (see here)
do_swap_page => Major p.f.

>>> "write" bit off (see here)
do_wp_page => Minor p.f.

Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

>>> File-backed VMA (see here)

do_cow_fault => Minor p.f

do_read_fault => Minor or Major p.f.

do_shared_fault => Minor or Major p.f.

>> Translation entry **set** (see here)

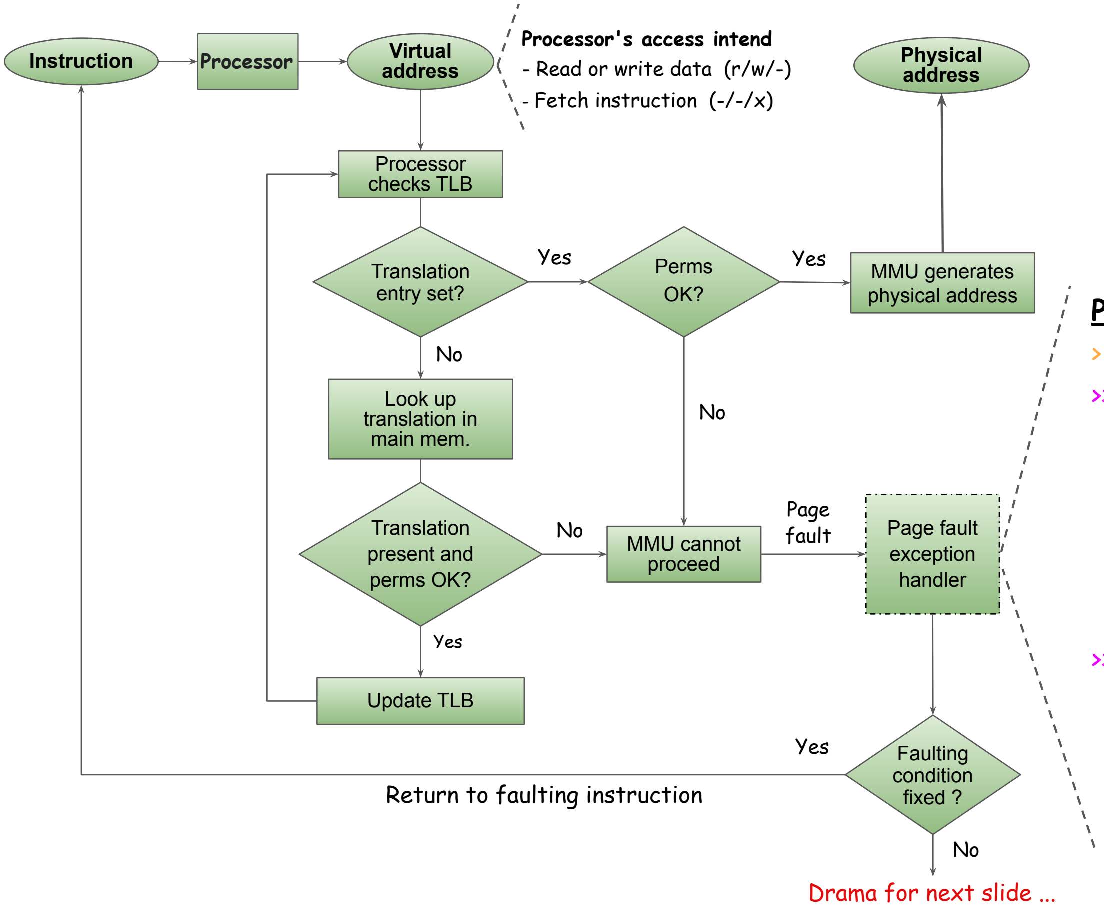
>>> "present" bit off (see here)

do_swap_page => Major p.f.

>>> "write" bit off (see here)

do_wp_page => Minor p.f.

Demand paging via page fault handling



Page fault handling (see here and here)

> Access **not** permitted (see here) => SIGSEGV

>> Translation entry not set (see here)

>>> Not file-backed VMA (see here)

do_anonymous_page=> Minor p.f

>>> File-backed VMA (see here)

do_cow_fault => Minor p.f

do_read_fault => Minor or Major p.f.

do_shared_fault => Minor or Major p.f.

>> Translation entry **set** (see here)

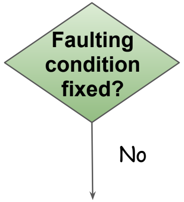
>>> "present" bit off (see here)

do_swap_page => Major p.f.

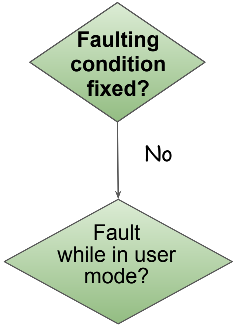
>>> "write" bit off (see here)

do_wp_page => Minor p.f.

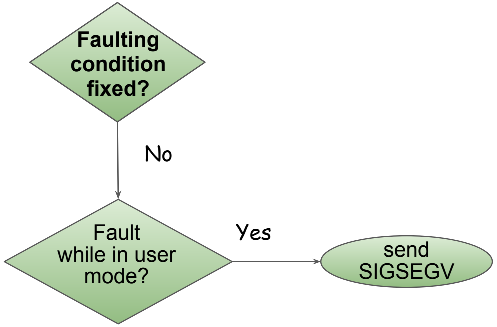
Page fault handling (cont'ed)



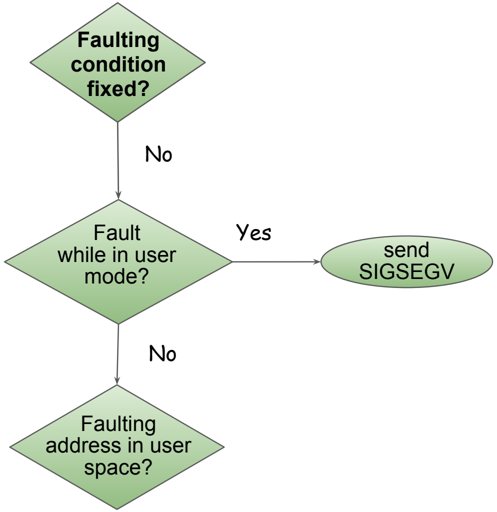
Page fault handling (cont'ed)



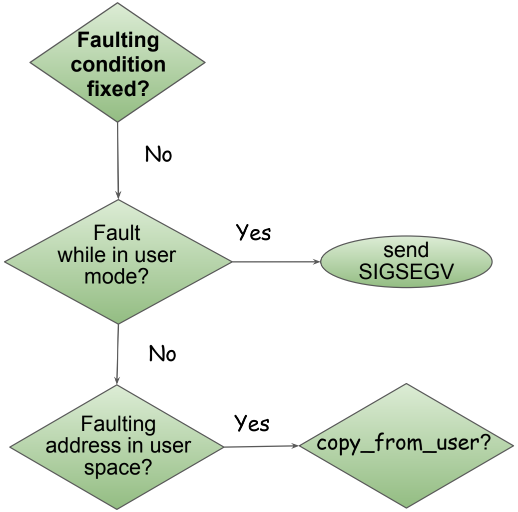
Page fault handling (cont'ed)



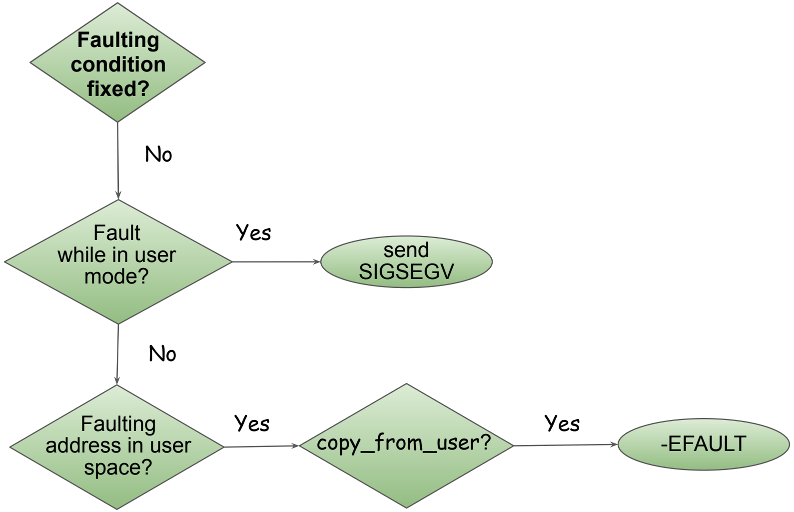
Page fault handling (cont'ed)



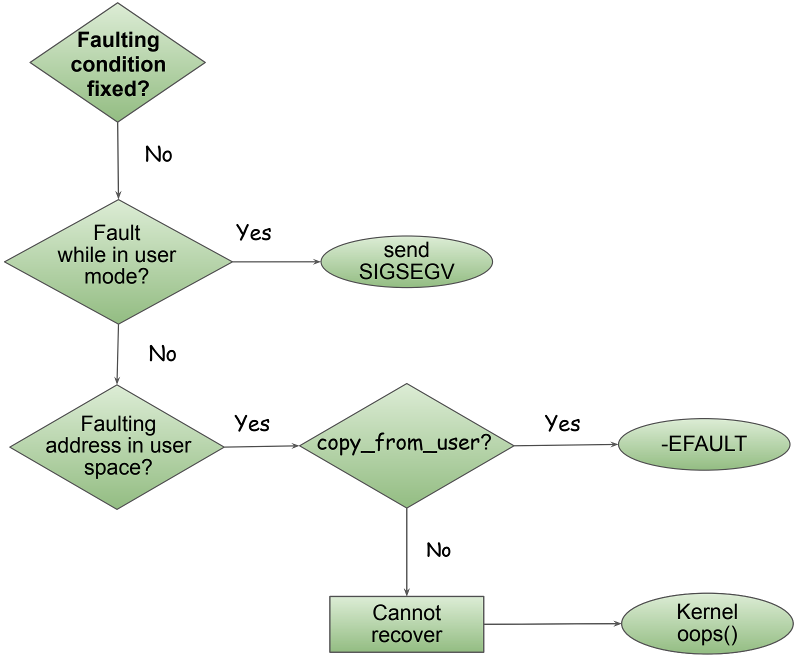
Page fault handling (cont'ed)



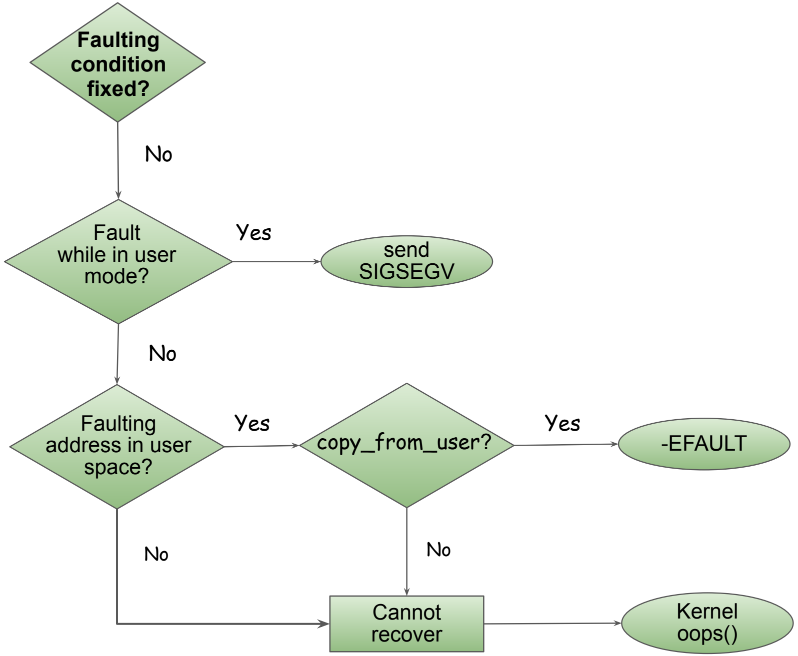
Page fault handling (cont'ed)



Page fault handling (cont'ed)



Page fault handling (cont'ed)



Page fault demo: read->write

```c
int main(int argc, char **argv) {
   int vma_size = 2 * 4096; // size of 2 pages in Linux
   char a, *buffer = mmap(NULL, vma_size, PROT_READ | PROT_WRITE,
                          MAP_PRIVATE | MAP_ANONYMOUS,
                          -1, 0);
  for (int i = 0; i < vma_size; i += 4096) {
      start_time = clock_gettime_ns();
      a = buffer[i];
      end_time = clock_gettime_ns();
      printf("page-%d: Time elapsed: %lu ns (1st read)\n",
             i / 4096, end_time - start_time);
      start_time = clock_gettime_ns();
      a = buffer[i];
      end_time = clock_gettime_ns();
      printf("page-%d: Time elapsed: %lu ns (2nd read)\n",
             i / 4096, end_time - start_time);
  }
  for (int i = 0; i < vma_size; i += 4096) {
      start_time = clock_gettime_ns();
      buffer[i] = 'A';
      end_time = clock_gettime_ns();
      printf("page-%d: Time elapsed: %lu ns (1st write)\n",
             i / 4096, end_time - start_time);
      start_time =clock_gettime_ns();
      buffer[i] = 'A';
      end_time = clock_gettime_ns();
      printf("page-%d: Time elapsed: %lu ns (2nd write)\n",
             i / 4096, end_time - start_time);
   }
}
```

Page fault demo: read->write

```c
int main(int argc, char **argv) {
  int vma_size = 2 * 4096; // size of 2 pages in Linux
  char a, *buffer = mmap(NULL, vma_size, PROT_READ | PROT_WRITE,
                         MAP_PRIVATE | MAP_ANONYMOUS,
                         -1, 0);
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st read)\n",
           i / 4096, end_time - start_time);
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd read)\n",
           i / 4096, end_time - start_time);
 }
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st write)\n",
           i / 4096, end_time - start_time);
    start_time =clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd write)\n",
           i / 4096, end_time - start_time);
  }
}
```

```
➜ git:(master) ✗ ./demo
(1) minor page faults: 87, major page faults: 0
```

Page fault demo: read->write

```
int main(int argc, char **argv) {
   int vma_size = 2 * 4096; // size of 2 pages in Linux
   char a, *buffer = mmap(NULL, vma_size, PROT_READ | PROT_WRITE,
                          MAP_PRIVATE | MAP_ANONYMOUS,
                          -1, 0);
  for (int i = 0; i < vma_size; i += 4096) {
      start_time = clock_gettime_ns();
      a = buffer[i];
      end_time = clock_gettime_ns();
      printf("page-%d: Time elapsed: %lu ns (1st read)\n",
             i / 4096, end_time - start_time);
      start_time = clock_gettime_ns();
      a = buffer[i];
      end_time = clock_gettime_ns();
      printf("page-%d: Time elapsed: %lu ns (2nd read)\n",
             i / 4096, end_time - start_time);
  }
  for (int i = 0; i < vma_size; i += 4096) {
      start_time = clock_gettime_ns();
      buffer[i] = 'A';
      end_time = clock_gettime_ns();
      printf("page-%d: Time elapsed: %lu ns (1st write)\n",
             i / 4096, end_time - start_time);
      start_time =clock_gettime_ns();
      buffer[i] = 'A';
      end_time = clock_gettime_ns();
      printf("page-%d: Time elapsed: %lu ns (2nd write)\n",
             i / 4096, end_time - start_time);
   }
}
```

```
➜ git:(master) ✗  ./demo
(1) minor page faults: 87, major page faults: 0
page-0: Time elapsed: 2633 nanoseconds (1st read)
page-0: Time elapsed: 78 nanoseconds (2nd read)
page-1: Time elapsed: 1956 nanoseconds (1st read)
page-1: Time elapsed: 78 nanoseconds (2nd read)
(2) minor page faults: 89, major page faults: 0
```

Read pg faults

Page fault demo: read->write

```c
int main(int argc, char **argv) {
  int vma_size = 2 * 4096; // size of 2 pages in Linux
  char a, *buffer = mmap(NULL, vma_size, PROT_READ | PROT_WRITE,
                         MAP_PRIVATE | MAP_ANONYMOUS,
                         -1, 0);
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st read)\n",
           i / 4096, end_time - start_time);
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd read)\n",
           i / 4096, end_time - start_time);
 }
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st write)\n",
           i / 4096, end_time - start_time);
    start_time =clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd write)\n",
           i / 4096, end_time - start_time);
  }
}
```

```
➜ git:(master) ✗ ./demo
(1) minor page faults: 87, major page faults: 0
page-0: Time elapsed: 2633 nanoseconds (1st read)
page-0: Time elapsed: 78 nanoseconds (2nd read)
page-1: Time elapsed: 1956 nanoseconds (1st read)
page-1: Time elapsed: 78 nanoseconds (2nd read)
(2) minor page faults: 89, major page faults: 0
page-0: Time elapsed: 4131 nanoseconds (1st write)
page-0: Time elapsed: 113 nanoseconds (2nd write)
page-1: Time elapsed: 3694 nanoseconds (1st write)
page-1: Time elapsed: 58 nanoseconds (2nd write)
(3) minor page faults: 91, major page faults: 0
```

Read pg faults

Write pg faults

Page fault demo: write->read

```c
int main(int argc, char **argv) {
  int vma_size = 2 * 4096; // size of 2 pages in Linux
  char a, *buffer = mmap(NULL, vma_size, PROT_READ | PROT_WRITE,
                         MAP_PRIVATE | MAP_ANONYMOUS,
                         -1, 0);
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st write)\n",
           i / 4096, end_time - start_time);
    start_time = clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd write)\n",
           i / 4096, end_time - start_time);
 }
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st read)\n",
           i / 4096, end_time - start_time);
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd read)\n",
           i / 4096, end_time - start_time);
 }
}
```

```
➜ git:(master) ✗ ./demo
(1) minor page faults: 87, major page faults: 0
page-0: Time elapsed: 2633 nanoseconds (1st read)
page-0: Time elapsed: 78 nanoseconds (2nd read)
page-1: Time elapsed: 1956 nanoseconds (1st read)
page-1: Time elapsed: 78 nanoseconds (2nd read)
(2) minor page faults: 89, major page faults: 0
page-0: Time elapsed: 4131 nanoseconds (1st write)
page-0: Time elapsed: 113 nanoseconds (2nd write)
page-1: Time elapsed: 3694 nanoseconds (1st write)
page-1: Time elapsed: 58 nanoseconds (2nd write)
(3) minor page faults: 91, major page faults: 0
➜ git:(master) ✗ ./write_read_page_faults
(1) minor page faults: 88, major page faults: 0
```

Read pg faults

Write pg faults

Page fault demo: write->read

```c
int main(int argc, char **argv) {
  int vma_size = 2 * 4096; // size of 2 pages in Linux
  char a, *buffer = mmap(NULL, vma_size, PROT_READ | PROT_WRITE,
                         MAP_PRIVATE | MAP_ANONYMOUS,
                         -1, 0);
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st write)\n",
           i / 4096, end_time - start_time);
    start_time = clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd write)\n",
           i / 4096, end_time - start_time);
 }
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st read)\n",
           i / 4096, end_time - start_time);
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd read)\n",
           i / 4096, end_time - start_time);
 }
}
```

```
➜ git:(master) ✗ ./demo
(1) minor page faults: 87, major page faults: 0
page-0: Time elapsed: 2633 nanoseconds (1st read)
page-0: Time elapsed: 78 nanoseconds (2nd read)
page-1: Time elapsed: 1956 nanoseconds (1st read)
page-1: Time elapsed: 78 nanoseconds (2nd read)
(2) minor page faults: 89, major page faults: 0
page-0: Time elapsed: 4131 nanoseconds (1st write)
page-0: Time elapsed: 113 nanoseconds (2nd write)
page-1: Time elapsed: 3694 nanoseconds (1st write)
page-1: Time elapsed: 58 nanoseconds (2nd write)
(3) minor page faults: 91, major page faults: 0
➜ git:(master) ✗ ./write_read_page_faults
(1) minor page faults: 88, major page faults: 0
page-0: Time elapsed: 5868 nanoseconds (1st write)
page-0: Time elapsed: 115 nanoseconds (2nd write)
page-1: Time elapsed: 5487 nanoseconds (1st write)
page-1: Time elapsed: 48 nanoseconds (2nd write)
(2) minor page faults: 90, major page faults: 0
```

Read pg faults

Write pg faults

Write pg faults

Page fault demo: write->read

```c
int main(int argc, char **argv) {
  int vma_size = 2 * 4096; // size of 2 pages in Linux
  char a, *buffer = mmap(NULL, vma_size, PROT_READ | PROT_WRITE,
                         MAP_PRIVATE | MAP_ANONYMOUS,
                         -1, 0);
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st write)\n",
           i / 4096, end_time - start_time);
    start_time = clock_gettime_ns();
    buffer[i] = 'A';
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd write)\n",
           i / 4096, end_time - start_time);
 }
 for (int i = 0; i < vma_size; i += 4096) {
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (1st read)\n",
           i / 4096, end_time - start_time);
    start_time = clock_gettime_ns();
    a = buffer[i];
    end_time = clock_gettime_ns();
    printf("page-%d: Time elapsed: %lu ns (2nd read)\n",
           i / 4096, end_time - start_time);
 }
}
```

```
➜ git:(master) ✗ ./demo
(1) minor page faults: 87, major page faults: 0
page-0: Time elapsed: 2633 nanoseconds (1st read)
page-0: Time elapsed: 78 nanoseconds (2nd read)
page-1: Time elapsed: 1956 nanoseconds (1st read)
page-1: Time elapsed: 78 nanoseconds (2nd read)
(2) minor page faults: 89, major page faults: 0
page-0: Time elapsed: 4131 nanoseconds (1st write)
page-0: Time elapsed: 113 nanoseconds (2nd write)
page-1: Time elapsed: 3694 nanoseconds (1st write)
page-1: Time elapsed: 58 nanoseconds (2nd write)
(3) minor page faults: 91, major page faults: 0
➜ git:(master) ✗ ./write_read_page_faults
(1) minor page faults: 88, major page faults: 0
page-0: Time elapsed: 5868 nanoseconds (1st write)
page-0: Time elapsed: 115 nanoseconds (2nd write)
page-1: Time elapsed: 5487 nanoseconds (1st write)
page-1: Time elapsed: 48 nanoseconds (2nd write)
(2) minor page faults: 90, major page faults: 0
page-0: Time elapsed: 90 nanoseconds (1st read)
page-0: Time elapsed: 92 nanoseconds (2nd read)
page-1: Time elapsed: 59 nanoseconds (1st read)
page-1: Time elapsed: 46 nanoseconds (2nd read)

(3) minor page faults: 90, major page faults: 0
```

Read pg faults

Write pg faults

Write pg faults

Read pg faults

Page fault demo: strcpy

```c
int main(int argc, char **argv) {
  // mmap one page (4096 in Linux)
  char *buffer = mmap(NULL, 4096,
                      PROT_READ | PROT_WRITE,
                      MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   start_time = clock_gettime_ns();
   strcpy(buffer, "Hello, mmap!");
   end_time = clock_gettime_ns();
   printf("1. Time elapsed: %lu ns\n", end_time - start_time);
   start_time = clock_gettime_ns();
   strcpy(buffer, "Hello, mmap!");
   end_time = clock_gettime_ns();
   printf("2. Time elapsed: %lu ns\n", end_time - start_time);
   if (mprotect(buffer, page_size, PROT_READ)) {
      perror("mproctect");
      return 1;
   }
   buffer[0] = 'C';
}
```

```
➜ git:(master) ✗ ./demo
1. Time elapsed: 3083 nanoseconds
2. Time elapsed: 167 nanoseconds
[1]    80193 segmentation fault  ./test
```

Calculating paze size by observing page fault latency

```c
#define NUM_PAGES 2
#define STEP_SIZE 512

void foo(char *ptr, int end_byte) {
    start_time = clock_gettime_ns();
    *ptr = 'A';
    end_time = clock_gettime_ns();
    printf("%d; Time elapsed: %lu ns\n", end_byte,
           (end_time - start_time));
}

int main(int argc, char **argv) {
    long page_size = sysconf(_SC_PAGE_SIZE);
    char *buf = mmap(NULL, NUM_PAGES*page_size,
                     PROT_READ | PROT_WRITE,
                     MAP_PRIVATE | MAP_ANONYMOUS,  -1, 0);

    int end_byte = STEP_SIZE;

    while (end_byte < NUM_PAGES * page_size) {
        foo(buf+end_byte, end_byte);
        end_byte += STEP_SIZE;
    }

    munmap(buf, NUM_PAGES*page_size);
    return 0;
}
```

```
➜ git:(master) ✗  ./demo

foo: 0; time elapsed: 2250 nanoseconds
foo: 512; time elapsed: 125 nanoseconds
foo: 1024; time elapsed: 125 nanoseconds
foo: 1536; time elapsed: 125 nanoseconds
foo: 2048; time elapsed: 125 nanoseconds
foo: 2560; time elapsed: 167 nanoseconds
foo: 3072; time elapsed: 167 nanoseconds
foo: 3584; time elapsed: 125 nanoseconds
foo: 4096; time elapsed: 3625 nanoseconds
foo: 4608; time elapsed: 125 nanoseconds
foo: 5120; time elapsed: 167 nanoseconds
foo: 5632; time elapsed: 83 nanoseconds
foo: 6144; time elapsed: 83 nanoseconds
foo: 6656; time elapsed: 125 nanoseconds
foo: 7168; time elapsed: 125 nanoseconds
foo: 7680; time elapsed: 334 nanoseconds
```

How is virtual memory implemented?

> Usually considered the most complex kernel subsystem

- It's simple if you view it as an index

Ask the right questions

~~Q1: What are the entries of the index?~~

~~Q2: How are the entries of the index used?~~

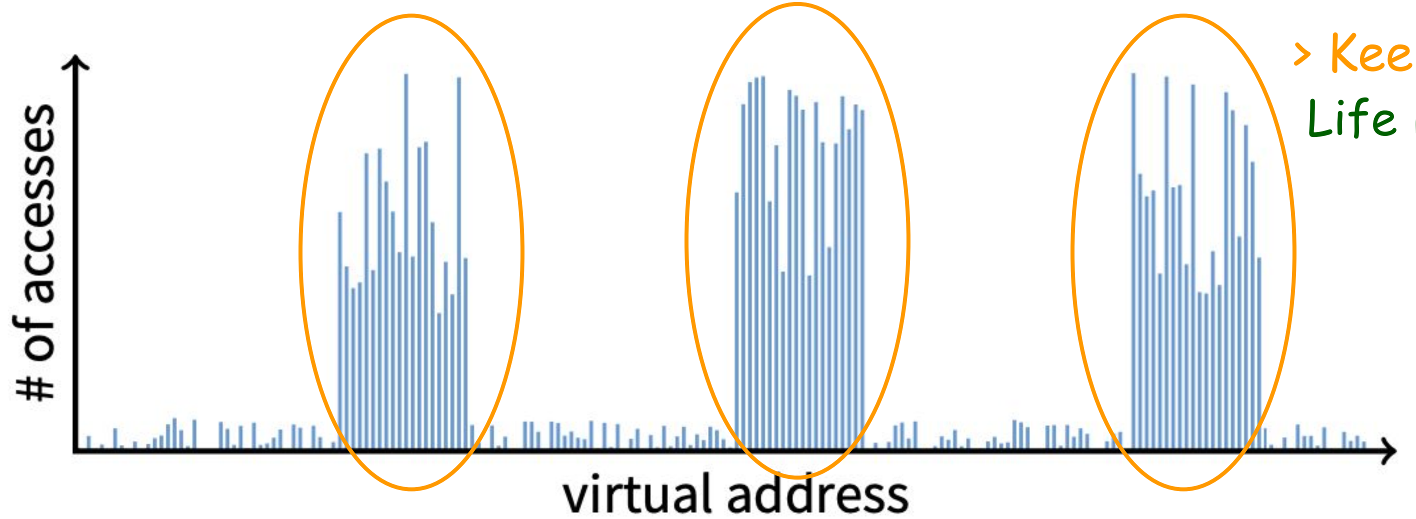
~~Q3: How are the entries of the index allocated?~~

Q4: How are the entries of the index replaced?

The working set model for program behavior

> A process can be in main memory iff all the pages that it is currently using can be in main memory, by P. Denning (1968)

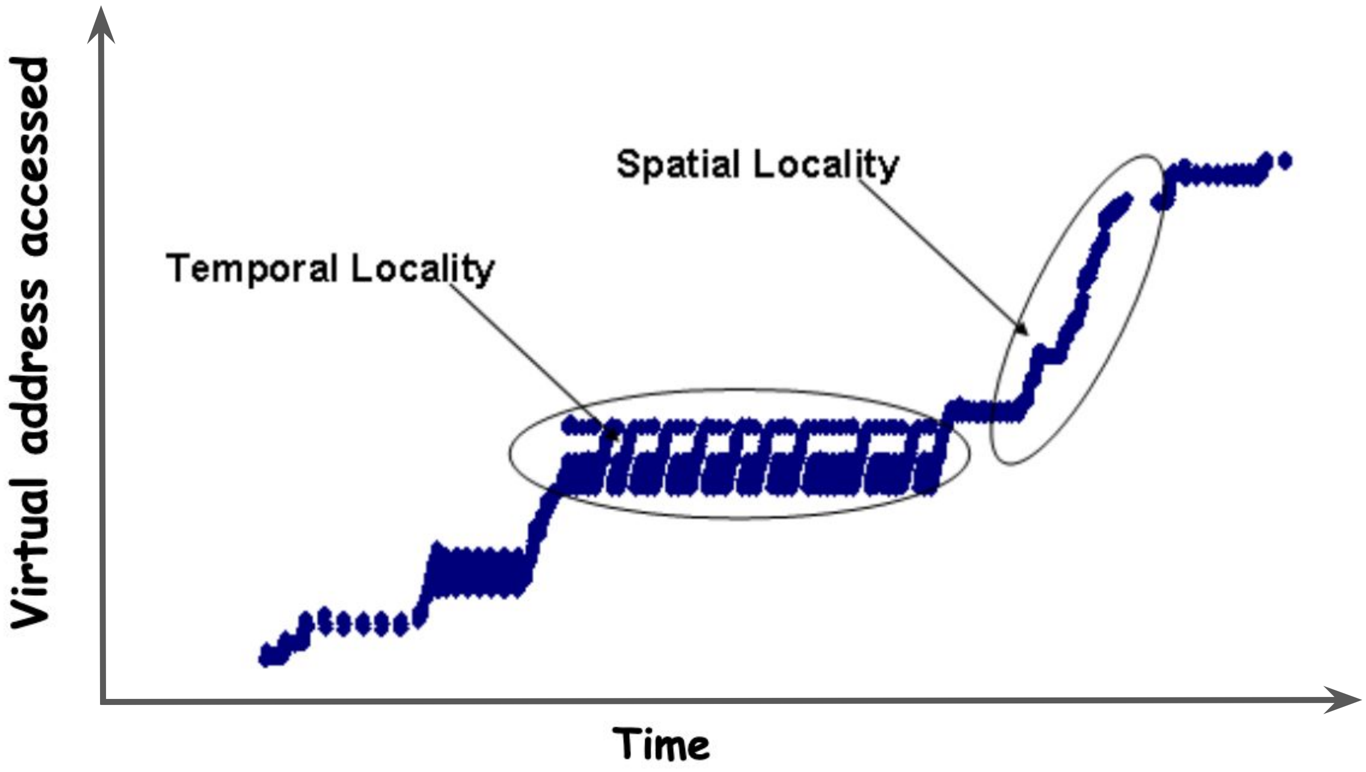
Rule of thumb: 20% of memory gets 80% of accesses



> Keep these "hot"?

Life is good…

Temporal and spatial locality



Replacement policies

> When all physical memory is in use, the OS must reclaim some physical page frames in order to serve new translations

Replacement policies

> When all physical memory is in use, the OS must reclaim some physical page frames in order to serve new translations

Which translations are "good candidates" to be evicted?

Replacement policies

> When all physical memory is in use, the OS must reclaim some physical page frames in order to serve new translations

Which translations are "good candidates" to be evicted?

> Converse of locality property: The Last Recently Used (LRU) page is the least to be used soon ("best eviction candidate")

Replacement policies

> When all physical memory is in use, the OS must reclaim some physical page frames in order to serve new translations

Which translations are "good candidates" to be evicted?

> Converse of locality property: The Last Recently Used (LRU) page is the least to be used soon ("best eviction candidate")

> Approximate LRU (a.k.a. CLOCK algorithm)

Replacement policies

> When all physical memory is in use, the OS must reclaim some physical page frames in order to serve new translations

Which translations are "good candidates" to be evicted?

> Converse of locality property: The Last Recently Used (LRU) page is the least to be used soon ("best eviction candidate")

> Approximate LRU (a.k.a. CLOCK algorithm)

- LRU is expensive to implement in software => Hardware support?

Replacement policies

> When all physical memory is in use, the OS must reclaim some physical page frames in order to serve new translations

Which translations are "good candidates" to be evicted?

> Converse of locality property: The Last Recently Used (LRU) page is the least to be used soon ("best eviction candidate")

> Approximate LRU (a.k.a. CLOCK algorithm)

- LRU is expensive to implement in software => Hardware support?
- Remember PTE "referenced" bit?

Replacement policies

> When all physical memory is in use, the OS must reclaim some physical page frames in order to serve new translations

Which translations are "good candidates" to be evicted?

> Converse of locality property: The Last Recently Used (LRU) page is the least to be used soon ("best eviction candidate")

> Approximate LRU (a.k.a. CLOCK algorithm)
- LRU is expensive to implement in software => Hardware support?
- Remember PTE "referenced" bit? The OS periodically swipes through PTEs and resets "accessed" bits

Replacement policies

> When all physical memory is in use, the OS must reclaim some physical page frames in order to serve new translations

Which translations are "good candidates" to be evicted?

> Converse of locality property: The Last Recently Used (LRU) page is the least to be used soon ("best eviction candidate")

> Approximate LRU (a.k.a. CLOCK algorithm)

- LRU is expensive to implement in software => Hardware support?
- Remember PTE "referenced" bit? The OS periodically swipes through PTEs and resets "accessed" bits
- Translation entries w/ "accessed" bits off

Replacement policies

> When all physical memory is in use, the OS must reclaim some physical page frames in order to serve new translations

Which translations are "good candidates" to be evicted?

> Converse of locality property: The Last Recently Used (LRU) page is the least to be used soon ("best eviction candidate")

> Approximate LRU (a.k.a. CLOCK algorithm)

- LRU is expensive to implement in software => Hardware support?
- Remember PTE "referenced" bit? The OS periodically swipes through PTEs and resets "accessed" bits
- Translation entries w/ "accessed" bits off => Not recently accessed

Replacement policies

- Not all pages are equal: Reclaiming decision is made based on the the type of the occupied pages

Replacement policies

- **Not all pages are equal:** Reclaiming decision is made based on the the type of the occupied pages
  - **File-backed pages:** Dropped from the kernel's page cache **first** since they can be reloaded later from the corresponding file

Replacement policies

> Not all pages are equal: Reclaiming decision is made based on the the type of the occupied pages

- File-backed pages: Dropped from the kernel's page cache **first** since they can be reloaded later from the corresponding file
- Anonymous pages w/o file backing: Must be written to the swap space in order to be available for reloading; **dropped next**

Replacement policies

- Not all pages are equal: Reclaiming decision is made based on the the type of the occupied pages
  - File-backed pages: Dropped from the kernel's page cache **first** since they can be reloaded later from the corresponding file
  - Anonymous pages w/o file backing: Must be written to the swap space in order to be available for reloading; **dropped next**
  - File-system-related kernel caches: **Dropped last** (next chapter)

Replacement policies

> Not all pages are equal: Reclaiming decision is made based on the the type of the occupied pages

- File-backed pages: Dropped from the kernel's page cache **first** since they can be reloaded later from the corresponding file
- Anonymous pages w/o file backing: Must be written to the swap space in order to be available for reloading; **dropped next**
- File-system-related kernel caches: **Dropped last** (next chapter)
- Last resort: Out-Of-Memory (OOM) killer terminates the process

Thrashing (or, "when nothing works")

Thrashing (or, "when nothing works")

> I/O devices at 100% utilization but CPU utilization drops to 0%

- Processes remain blocked waiting for pages to be fetched in

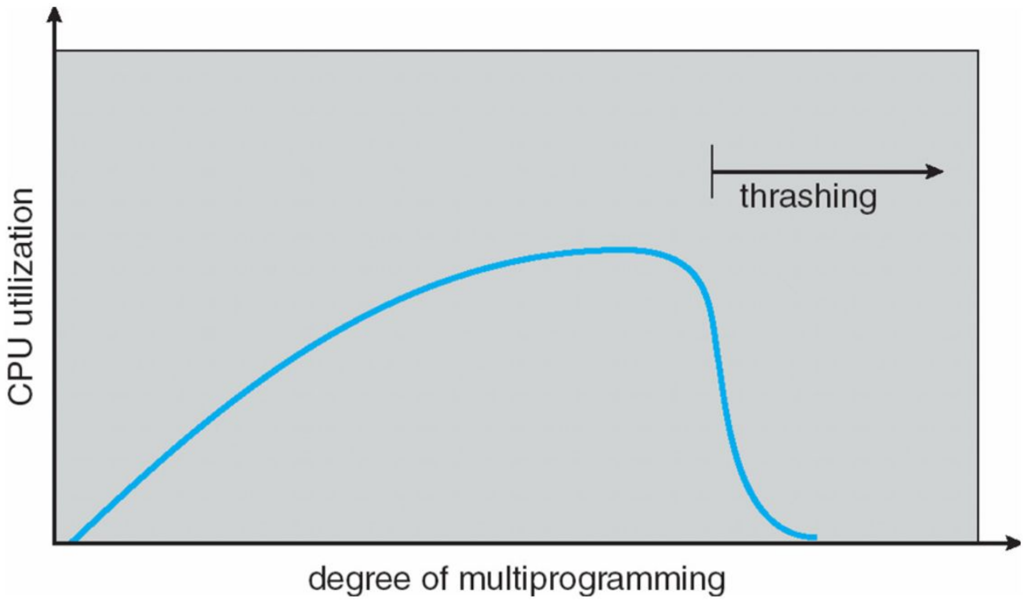
Thrashing (or, "when nothing works")

> I/O devices at 100% utilization but CPU utilization drops to 0%

- Processes remain blocked waiting for pages to be fetched in
- The system does no useful work

Thrashing (or, "when nothing works")

- I/O devices at 100% utilization but CPU utilization drops to 0%
  - Processes remain blocked waiting for pages to be fetched in
  - The system does no useful work



Thrashing (or, "when nothing works")

> Reasons for thrashing

- Memory accesses w/o temporal locality => Past $\neq$ Future

Thrashing (or, "when nothing works")

> Reasons for thrashing

- Memory accesses w/o temporal locality => Past $\neq$ Future
- Processes' hot memory does not fit in maim memory

Thrashing (or, "when nothing works")

> Reasons for thrashing
  - Memory accesses w/o temporal locality => Past $\neq$ Future
  - Processes' hot memory does not fit in maim memory

> What we ordered?

Thrashing (or, "when nothing works")

- Reasons for thrashing
  - Memory accesses w/o temporal locality => Past $\neq$ Future
  - Processes' hot memory does not fit in maim memory
- What we ordered? Memory with the size of disk and the speed of CPU caches

Thrashing (or, "when nothing works")

> Reasons for thrashing
  - Memory accesses w/o temporal locality => Past $\neq$ Future
  - Processes' hot memory does not fit in maim memory

> What we ordered? Memory with the size of disk and the speed of CPU caches

> What we got?

Thrashing (or, "when nothing works")

> Reasons for thrashing

- Memory accesses w/o temporal locality => Past $\neq$ Future
- Processes' hot memory does not fit in maim memory

> What we ordered? Memory with the size of disk and the speed of CPU caches

> What we got? Memory with the access time of the disk :-(

POSIX Memory management syscalls (cont'ed)

void *mmap (void *addr, ...)

> Creates a new mapping in the virtual address space of the calling process

- addr: If NULL, the kernel chooses the address at which to create the mapping
- On success, mmap(...) returns a pointer to the mapped area

POSIX Memory management syscalls (cont'ed)

void *mmap (void *addr, ...)

> Creates a new mapping in the virtual address space of the calling process

- addr: If NULL, the kernel chooses the address at which to create the mapping
- On success, mmap(...) returns a pointer to the mapped area

int munmap (void *addr, size_t length)

> Deletes the mappings for the specified address range

- addr: Start of the address range
- length: Size of the address range
- On success, munmap(...) returns 0

POSIX Memory management syscalls (cont'ed)

void *mmap (void *addr, ...)

> Creates a new mapping in the virtual address space of the calling process

- addr: If NULL, the kernel chooses the address at which to create the mapping
- On success, mmap(...) returns a pointer to the mapped area

int munmap (void *addr, size_t length)

> Deletes the mappings for the specified address range

- addr: Start of the address range
- length: Size of the address range
- On success, munmap(...) returns 0

int mprotect (void *addr, size_t len, int prot)

> Updates the protections for page(s) in range [addr, addr+len) to "prot"

- addr: Must be aligned to a page boundary
- On success, mprotect(...) returns 0

POSIX Memory management syscalls (cont'ed)

int msync (void *addr, ...):

> Flushes to disk changes made to the in-kernel copy of a file that was mapped into memory using mmap

- Note: Without use of msync(...) there is no guarantee that changes made to the in-kernel copy of a file will be written back to disk before munmap(...) is called
- On success, msync(...) returns 0

POSIX Memory management syscalls (cont'ed)

int msync (void *addr, ...):

> Flushes to disk changes made to the in-kernel copy of a file that was mapped into memory using mmap

- Note: Without use of msync(...) there is no guarantee that changes made to the in-kernel copy of a file will be written back to disk before munmap(...) is called

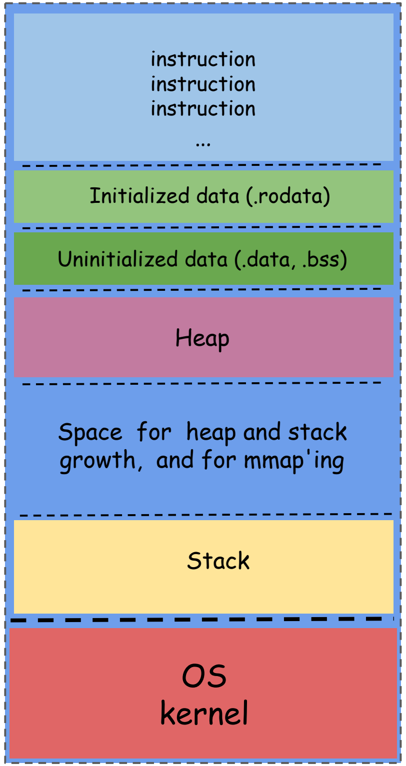
- On success, msync(...) returns 0

void *mlock/unlock (void *addr, ...)

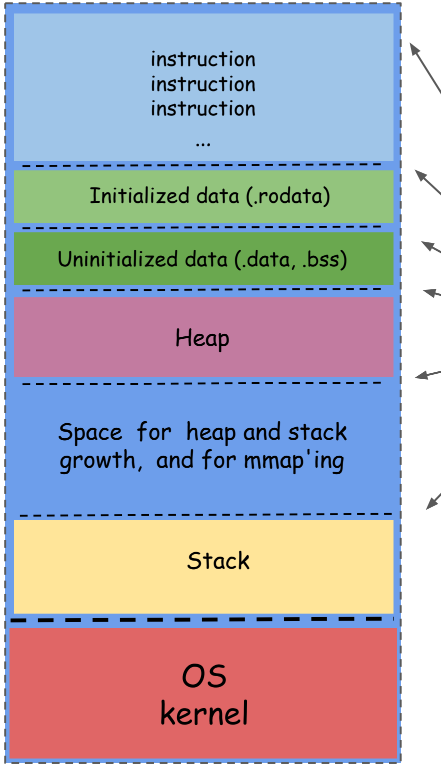
> Locks/unlocks part or all of the calling process's virtual address space into main mem. preventing that mem. from being paged out to the swap area

- On success, mlock(...)/unlock(...) returns 0

Putting it all together: (1/3) Per-process index of VAS segments

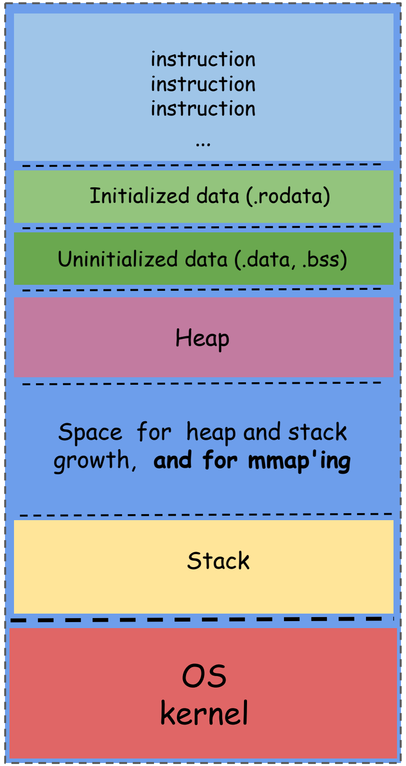


Putting it all together: (1/3) Per-process index of VAS segments




```
struct task_struct {
    ...
    char comm[TASK_COMM_LEN];
    struct mm_struct *active_mm;
    struct mm_struct *mm;
    struct mm_struct {
        ...
        pgd_t *pgd;
        unsigned long start_code, end_code;
        unsigned long start_data, end_data;
        unsigned long start_brk, brk;
        unsigned long start_stack;
        struct maple_tree mm_mt;
        struct vm_area_struct {
            unsigned long vm_start;
            unsigned long vm_end;
            pgprot_t vm_page_prot;
            vm_flags_t vm_flags;
            ...
      }
```

Putting it all together: (1/3) Per-process index of VAS segments

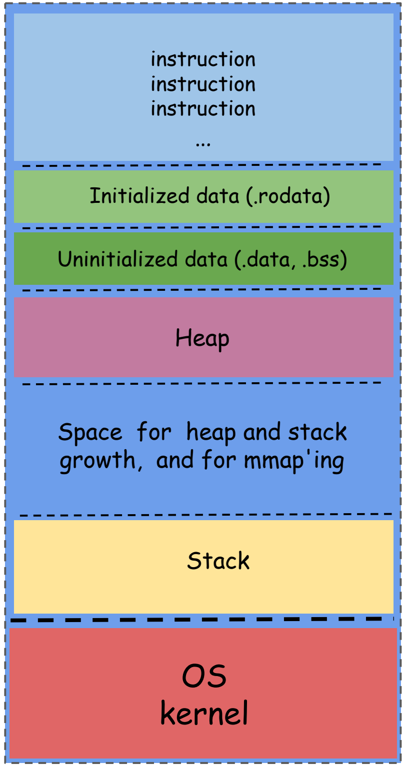



```
struct task_struct {
    ...
    char comm[TASK_COMM_LEN];
    struct mm_struct *active_mm;
    struct mm_struct *mm;
    struct mm_struct {
        ...
        pgd_t *pgd;
        unsigned long start_code, end_code;
        unsigned long start_data, end_data;
        unsigned long start_brk, brk;
        unsigned long start_stack;
        struct maple_tree mm_mt;
        struct vm_area_struct {
            unsigned long vm_start;
            unsigned long vm_end;
            pgprot_t vm_page_prot;
            vm_flags_t vm_flags;
            ...
    }
```

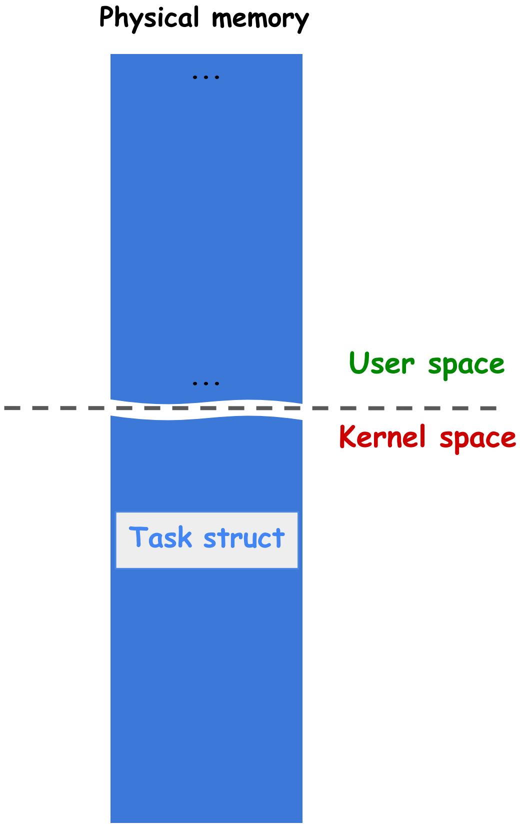
> Mapple tree: Read [here]

```
➜ git:(master) ✗  cat /proc/12453/maps
aaaadcda0000-aaaadcda1000 r-xp  ... /foo
aaaadcdb0000-aaaadcdb1000 r--p  ... /foo
aaaadcdb1000-aaaadcdb2000 rw-p ... /foo
...
aaaaed20d000-aaaaed22e000 rw-p ... [heap]
ffffbe610000-ffffbe798000 r-xp ... libc.so
ffffbe798000-ffffbe7a7000 ---p ... libc.so
...
fffff6461000-fffff6482000 rw-p ... [stack]
```

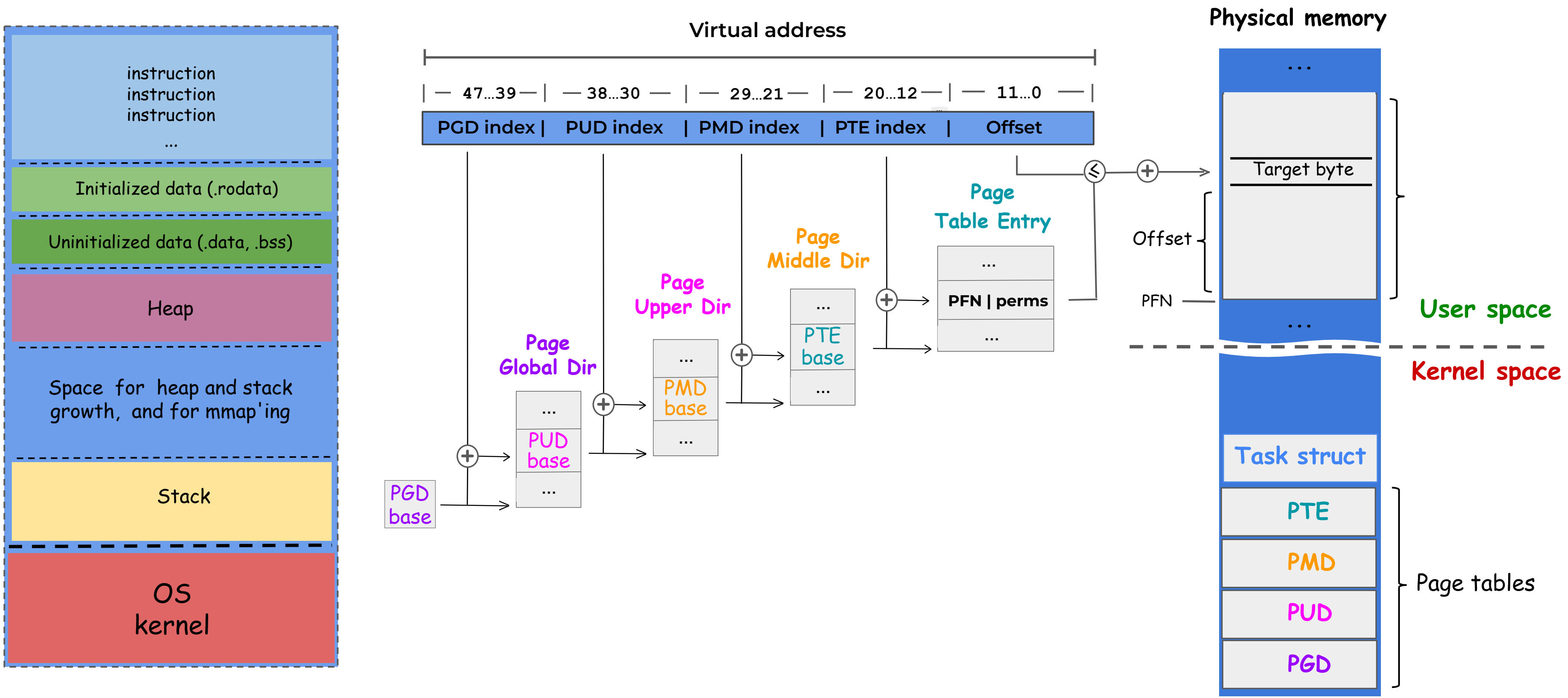
Putting it all together: (1/3) Per-process index of VAS segments



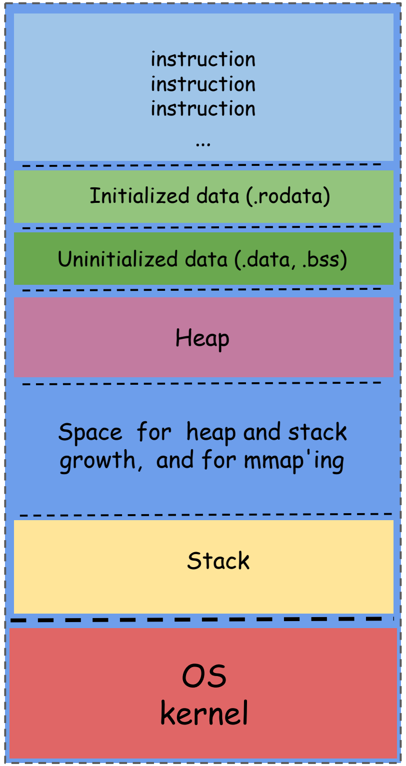

```
struct task_struct {
  ...
  char comm[TASK_COMM_LEN];
  struct mm_struct *active_mm;
  struct mm_struct *mm;
  struct mm_struct {
    ...
    pgd_t *pgd;
    unsigned long start_code, end_code;
    unsigned long start_data, end_data;
    unsigned long start_brk, brk;
    unsigned long start_stack;
    struct maple_tree mm_mt;
    struct vm_area_struct {
      unsigned long vm_start;
      unsigned long vm_end;
      pgprot_t vm_page_prot;
      vm_flags_t vm_flags;
      ...
  }
```



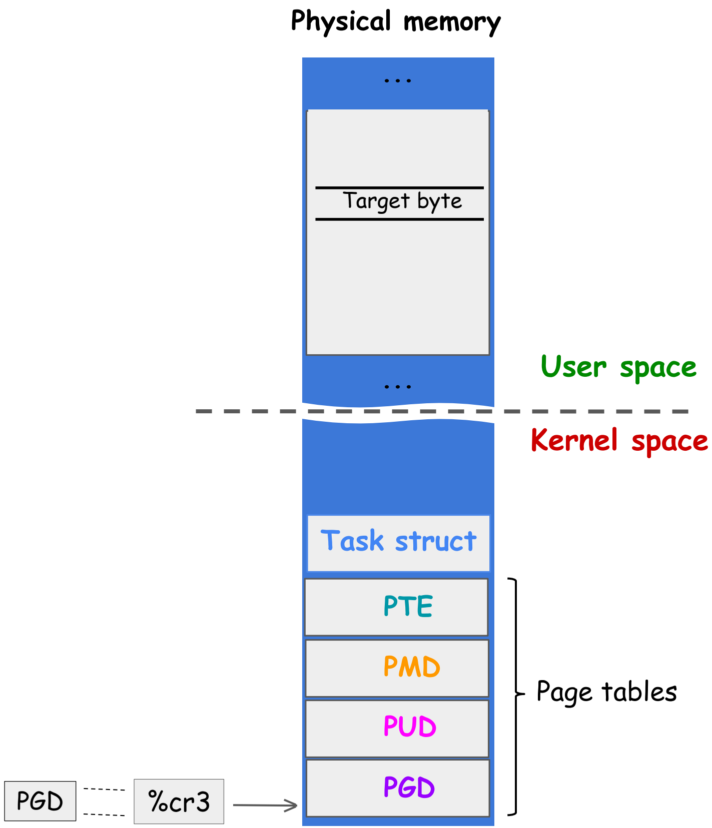
Putting it all together: (2/3) Per-process page table



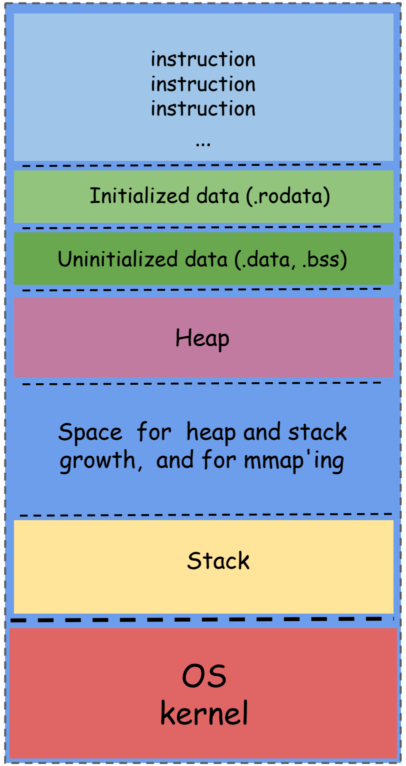
Putting it all together: (2/3) Per-process page table



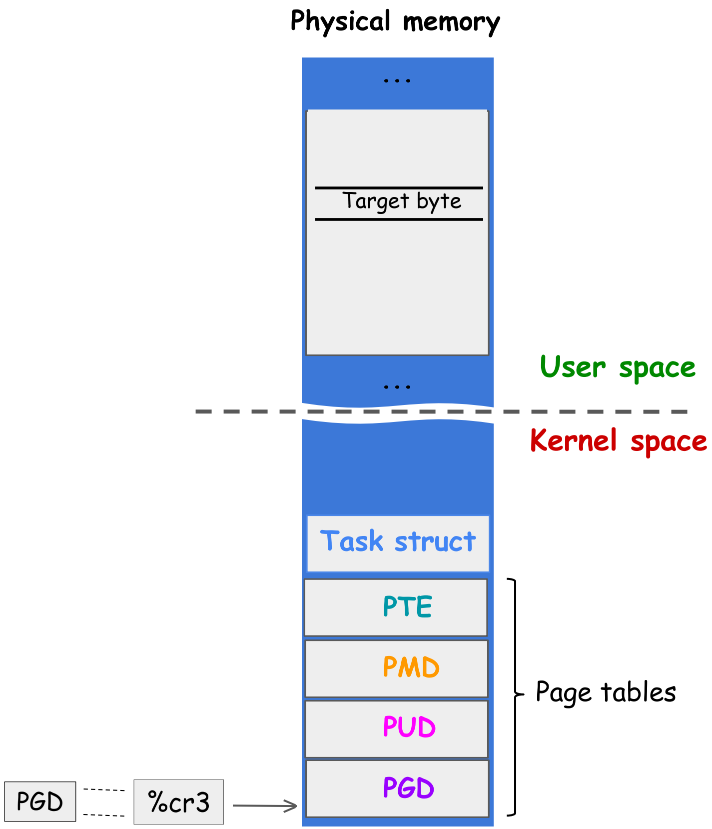

```
struct task_struct {
    ...
    char comm[TASK_COMM_LEN];
    struct mm_struct *active_mm;
    struct mm_struct *mm;
    struct mm_struct {
        ...
        pgd_t *pgd;
        unsigned long start_code, end_code;
        unsigned long start_data, end_data;
        unsigned long start_brk, brk;
        unsigned long start_stack;
        struct maple_tree mm_mt;
        struct vm_area_struct {
            unsigned long vm_start;
            unsigned long vm_end;
            pgprot_t vm_page_prot;
            vm_flags_t vm_flags;
            ...
        }
```



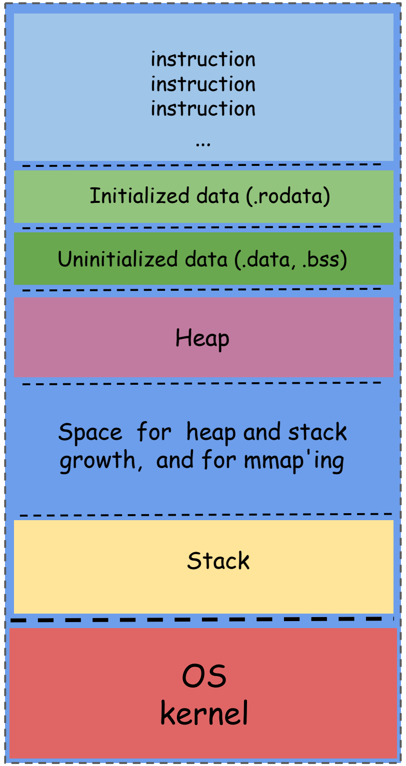
Putting it all together: (3/3) System-wide page frame allocators



>Buddy memory allocator

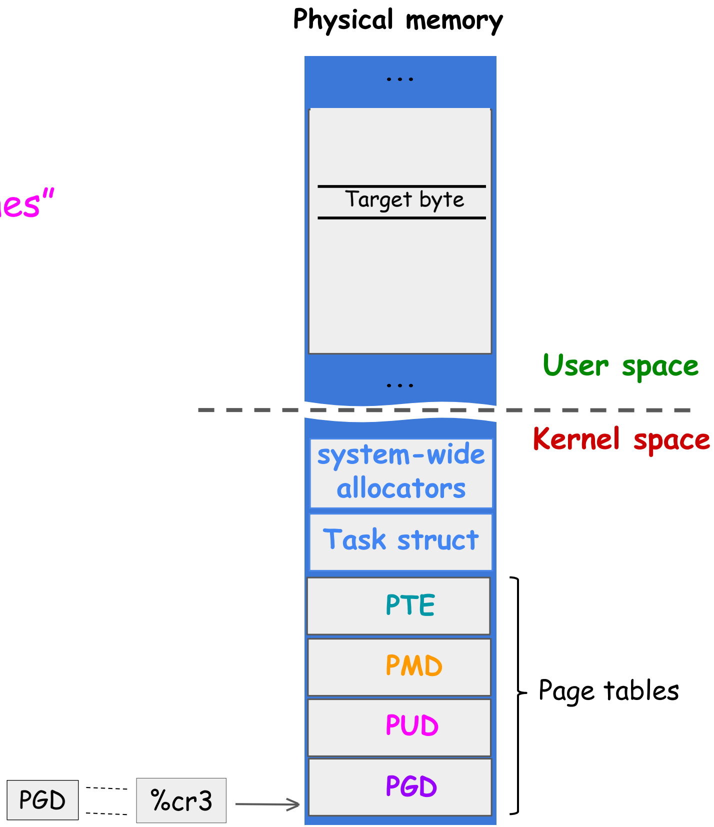


Putting it all together: (3/3) System-wide page frame allocators

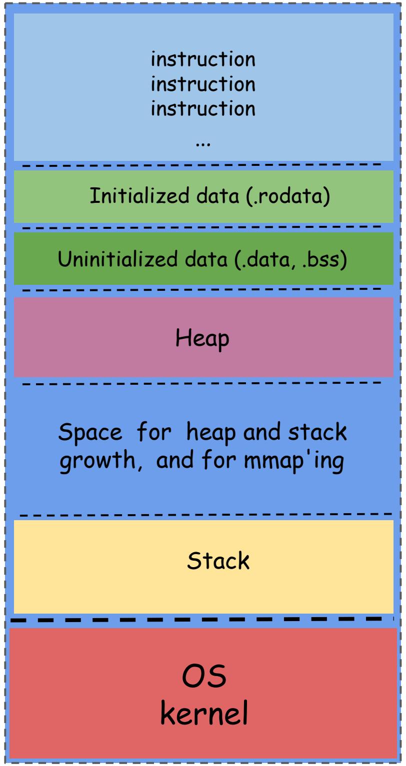


> Buddy memory allocator

>> Physical memory is divided into "zones"



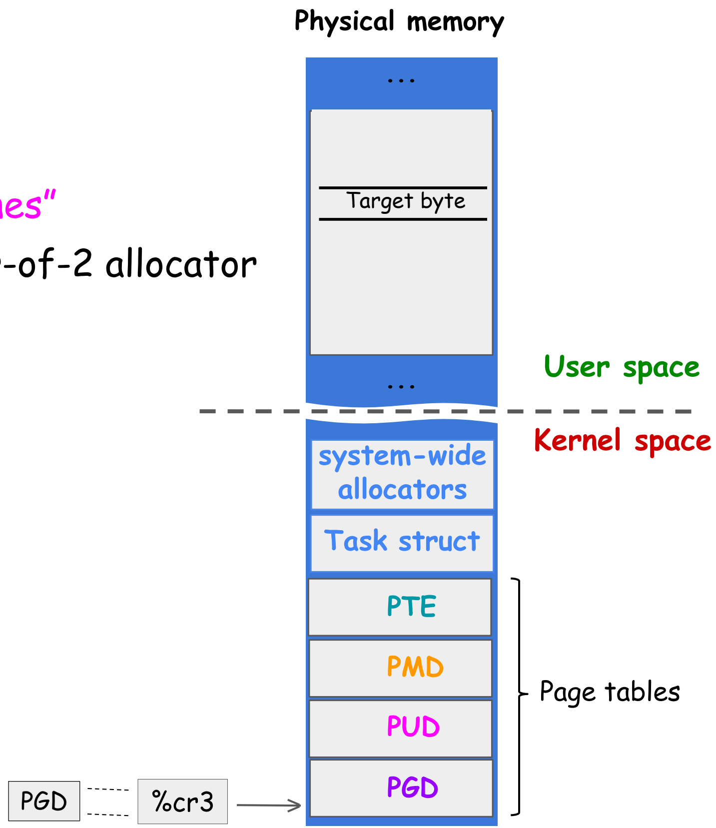
Putting it all together: (3/3) System-wide page frame allocators



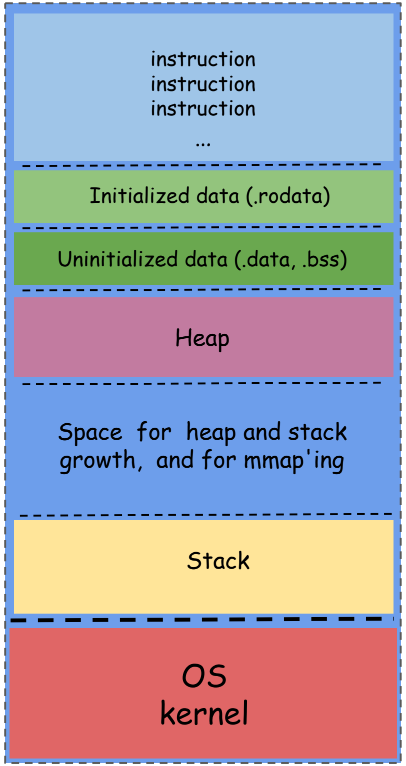
>Buddy memory allocator

>> Physical memory is divided into "zones"

- Buddy allocator: Range-based, power-of-2 allocator



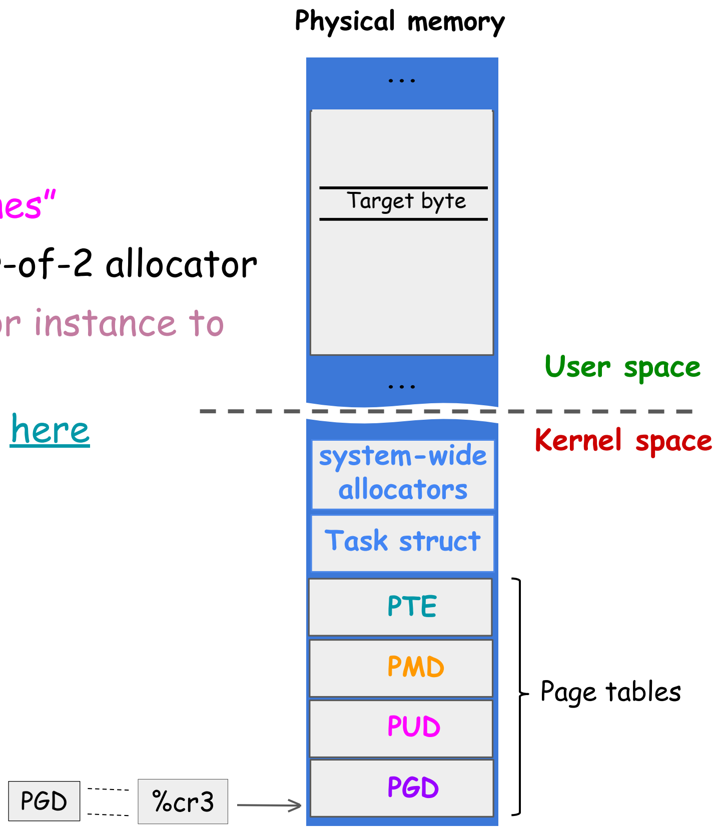
Putting it all together: (3/3) System-wide page frame allocators



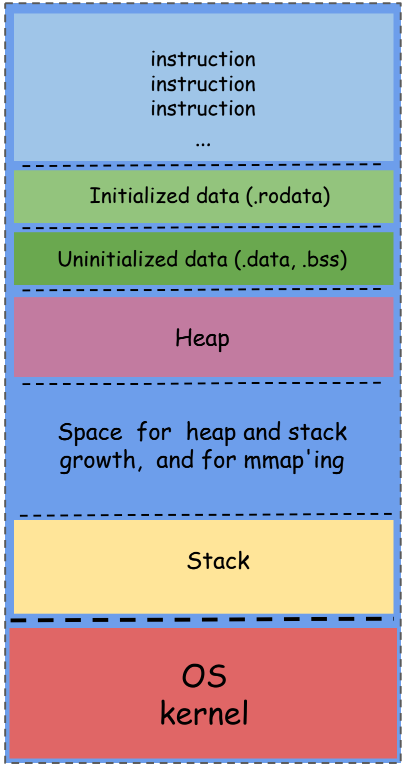
>Buddy memory allocator

>> Physical memory is divided into "zones"

- Buddy allocator: Range-based, power-of-2 allocator
- Each zone has its own buddy allocator instance to manage free physical pages
- Core of the zoned buddy allocator is here



Putting it all together: (3/3) System-wide page frame allocators



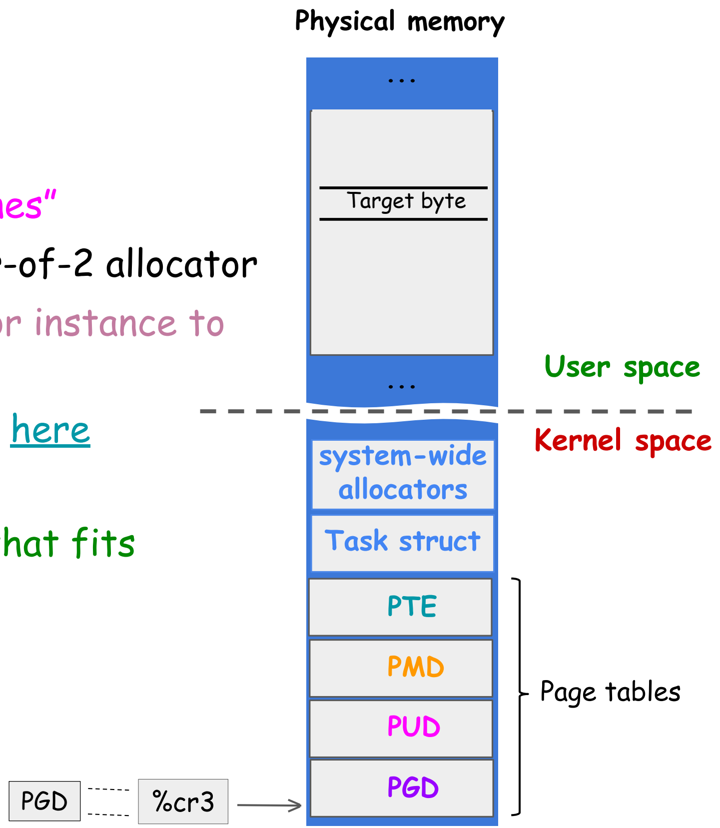
>Buddy memory allocator

>> Physical memory is divided into "zones"

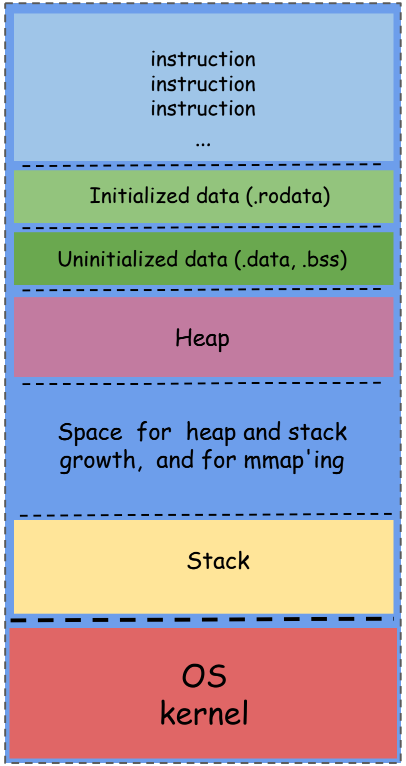
- Buddy allocator: Range-based, power-of-2 allocator
- Each zone has its own buddy allocator instance to manage free physical pages
- Core of the zoned buddy allocator is [here]

>> Request comes in

- Allocator finds the smallest block that fits



Putting it all together: (3/3) System-wide page frame allocators



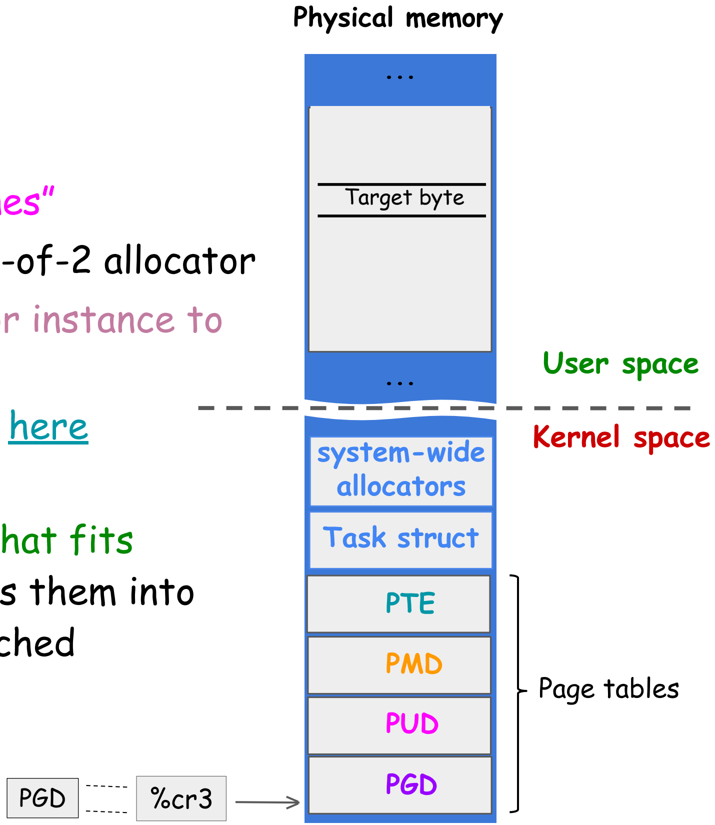
>Buddy memory allocator

>> Physical memory is divided into "zones"

- Buddy allocator: Range-based, power-of-2 allocator
- Each zone has its own buddy allocator instance to manage free physical pages
- Core of the zoned buddy allocator is here

>> Request comes in

- Allocator finds the smallest block that fits
- Only larger blocks available? Splits them into "buddies" until the right size is reached



Intercepting mem. accesses w/ mprotect and signals

```
static void handler(int sig, siginfo_t *si, void *unused) {
    printf("Got SIGSEGV\n");
    printf("Someone touched address: 0x%lx\n");
     exit(1); // Remove this line and see what happens...
}

int main(int argc, char **argv) {
   struct sigaction sa;
   sa.sa_flags = SA_SIGINFO;
   sa.sa_sigaction = handler;
   sigemptyset(&sa.sa_mask);
   sigaction(SIGSEGV, &sa, NULL);
   char *buf = mmap(NULL, 4096,  PROT_READ | PROT_WRITE,
                        MAP_PRIVATE | MAP_ANONYMOUS,  -1, 0);
   printf("1. Going to touch\n");
   buf[123] = 'a';
   if ( mprotect(buf, 4096, PROT_NONE)) {
      perror("mprotect");
      return 1;
   }
   show_self_proc_maps();
   printf("2. Going to touch\n");
   buf[123] = 'a';
}
```

```
➜  git:(master) ✗  ./test
1. Going to touch
```

Intercepting mem. accesses w/ mprotect and signals

```c
static void handler(int sig, siginfo_t *si, void *unused) {
    printf("Got SIGSEGV\n");
    printf("Someone touched address: 0x%lx\n");
    exit(1); // Remove this line and see what happens...
}

int main(int argc, char **argv) {
   struct sigaction sa;
   sa.sa_flags = SA_SIGINFO;
   sa.sa_sigaction = handler;
   sigemptyset(&sa.sa_mask);
   sigaction(SIGSEGV, &sa, NULL);
   char *buf = mmap(NULL, 4096,  PROT_READ | PROT_WRITE,
                          MAP_PRIVATE | MAP_ANONYMOUS,  -1, 0);
   printf("1. Going to touch\n");
   buf[123] = 'a';
   if ( mprotect(buf, 4096, PROT_NONE)) {
      perror("mprotect");
      return 1;
   }
   show_self_proc_maps();
   printf("2. Going to touch\n");
   buf[123] = 'a';
}
```

```
➜  git:(master) ✗  ./test
1. Going to touch
aaaab6840000-aaaab6842000 r-xp
aaaab6851000-aaaab6852000 r--p
aaaab6852000-aaaab6853000 rw-p
aaaaeafda000-aaaaeaffb000   rw-p   [heap]
ffff88510000-ffff88698000 r-xp    libc.so.6
ffff88698000-ffff886a7000 ---p    libc.so.6
ffff886a7000-ffff886ab000 r--p    libc.so.6
ffff886ab000-ffff886ad000 rw-p    libc.so.6
ffff886ad000-ffff886b9000 rw-p
ffff886da000-ffff88705000 r-xp    ld-linux-aarch64.so.1
ffff8870e000-ffff8870f000 ---p
ffff8870f000-ffff88711000 rw-p
ffff88711000-ffff88713000 r--p    [vvar]
ffff88713000-ffff88714000 r-xp    [vdso]
ffff88714000-ffff88716000 r--p    ld-linux-aarch64.so.1
ffff88716000-ffff88718000 rw-p    ld-linux-aarch64.so.1
fffff5a11000-fffff5a32000  rw-p    [stack]
```

Intercepting mem. accesses w/ mprotect and signals

```c
static void handler(int sig, siginfo_t *si, void *unused) {
    printf("Got SIGSEGV\n");
    printf("Someone touched address: 0x%lx\n");
     exit(1); // Remove this line and see what happens...
}

int main(int argc, char **argv) {
   struct sigaction sa;
   sa.sa_flags = SA_SIGINFO;
   sa.sa_sigaction = handler;
   sigemptyset(&sa.sa_mask);
   sigaction(SIGSEGV, &sa, NULL);
   char *buf = mmap(NULL, 4096,  PROT_READ | PROT_WRITE,
                    MAP_PRIVATE | MAP_ANONYMOUS,  -1, 0);
   printf("1. Going to touch\n");
   buf[123] = 'a';
   if ( mprotect(buf, 4096, PROT_NONE)) {
      perror("mprotect");
      return 1;
   }
   show_self_proc_maps();
   printf("2. Going to touch\n");
   buf[123] = 'a';
}
```

```
➜ git:(master) ✗  ./test
1. Going to touch
aaaab6840000-aaaab6842000 r-xp
aaaab6851000-aaaab6852000 r--p
aaaab6852000-aaaab6853000 rw-p
aaaaeafda000-aaaaeaffb000   rw-p   [heap]
ffff88510000-ffff88698000 r-xp    libc.so.6
ffff88698000-ffff886a7000 ---p    libc.so.6
ffff886a7000-ffff886ab000 r--p    libc.so.6
ffff886ab000-ffff886ad000 rw-p    libc.so.6
ffff886ad000-ffff886b9000 rw-p
ffff886da000-ffff88705000 r-xp    ld-linux-aarch64.so.1
ffff8870e000-ffff8870f000 ---p
ffff8870f000-ffff88711000 rw-p
ffff88711000-ffff88713000 r--p    [vvar]
ffff88713000-ffff88714000 r-xp    [vdso]
ffff88714000-ffff88716000 r--p    ld-linux-aarch64.so.1
ffff88716000-ffff88718000 rw-p    ld-linux-aarch64.so.1
fffff5a11000-fffff5a32000  rw-p    [stack]
2. Going to touch
Got SIGSEGV
```

Intercepting mem. accesses w/ mprotect and signals

```c
static void handler(int sig, siginfo_t *si, void *unused) {
    printf("Got SIGSEGV\n");
    printf("Someone touched address: 0x%lx\n");
    exit(1); // Remove this line and see what happens...
}

int main(int argc, char **argv) {
    struct sigaction sa;
    sa.sa_flags = SA_SIGINFO;
    sa.sa_sigaction = handler;
    sigemptyset(&sa.sa_mask);
    sigaction(SIGSEGV, &sa, NULL);
    char *buf = mmap(NULL, 4096,  PROT_READ | PROT_WRITE,
                     MAP_PRIVATE | MAP_ANONYMOUS,  -1, 0);
    printf("1. Going to touch\n");
    buf[123] = 'a';
    if ( mprotect(buf, 4096, PROT_NONE)) {
        perror("mprotect");
        return 1;
    }
    show_self_proc_maps();
    printf("2. Going to touch\n");
    buf[123] = 'a';
}
```

```
➜  git:(master) ✗  ./test
1. Going to touch
aaaab6840000-aaaab6842000 r-xp
aaaab6851000-aaaab6852000 r--p
aaaab6852000-aaaab6853000 rw-p
aaaaeafda000-aaaaeaffb000   rw-p   [heap]
ffff88510000-ffff88698000 r-xp    libc.so.6
ffff88698000-ffff886a7000 ---p    libc.so.6
ffff886a7000-ffff886ab000 r--p    libc.so.6
ffff886ab000-ffff886ad000 rw-p    libc.so.6
ffff886ad000-ffff886b9000 rw-p
ffff886da000-ffff88705000 r-xp    ld-linux-aarch64.so.1
ffff8870e000-ffff8870f000  ---p
ffff8870f000-ffff88711000 rw-p
ffff88711000-ffff88713000 r--p    [vvar]
ffff88713000-ffff88714000 r-xp    [vdso]
ffff88714000-ffff88716000 r--p    ld-linux-aarch64.so.1
ffff88716000-ffff88718000 rw-p    ld-linux-aarch64.so.1
fffff5a11000-fffff5a32000  rw-p    [stack]
2. Going to touch
Got SIGSEGV
Someone touched address: 0xffff8870e07b
```

Tracking PTE invalidations with mprotect

```c
void foo(char *buf) {
   start_time = clock_gettime_ns();
   *buf = 'a';
   end_time = clock_gettime_ns();
   printf("time elapsed: %lu nanoseconds \n");
}
int main(int argc, char **argv) {
   char *buf = mmap(NULL, page_size, PROT_READ | PROT_WRITE,
                        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i += page_size/4)
       foo(buf+i);

   char *buf = mmap(NULL, page_size, PROT_READ | PROT_WRITE,
                        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
       foo(buf+i);
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }
   char *buf = mmap(NULL, page_size, PROT_WRITE,
                        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
      foo(buf+i);
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }
   char *buf = mmap(NULL, page_size, PROT_WRITE,
                        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
      foo(buf+i);
      mprotect(buf, page_size, PROT_READ );
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
}}
```

```
➜ git:(master) ✗ ./test
time elapsed: 1417 nanoseconds
time elapsed: 125 nanoseconds
time elapsed: 334 nanoseconds
time elapsed: 125 nanoseconds
------------------------------------
```

Tracking PTE invalidations with mprotect

```
void foo(char *buf) {
   start_time = clock_gettime_ns();
   *buf = 'a';
   end_time = clock_gettime_ns();
   printf("time elapsed: %lu nanoseconds \n");
}
int main(int argc, char **argv) {
   char *buf = mmap(NULL, page_size, PROT_READ | PROT_WRITE,
                       MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i += page_size/4)
       foo(buf+i);

   char *buf = mmap(NULL, page_size, PROT_READ | PROT_WRITE,
                       MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
       foo(buf+i);
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }
   char *buf = mmap(NULL, page_size, PROT_WRITE,
                       MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
      foo(buf+i);
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }
   char *buf = mmap(NULL, page_size, PROT_WRITE,
                       MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
      foo(buf+i);
      mprotect(buf, page_size, PROT_READ );
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }}
```

```
➜ git:(master) ✗ ./test
time elapsed: 1417 nanoseconds
time elapsed: 125 nanoseconds
time elapsed: 334 nanoseconds
time elapsed: 125 nanoseconds
---------------------------------

time elapsed: 3084 nanoseconds
time elapsed: 42 nanoseconds
time elapsed: 125 nanoseconds
time elapsed: 84 nanoseconds
--------------------------------
```

Tracking PTE invalidations with mprotect

```c
void foo(char *buf) {
   start_time = clock_gettime_ns();
   *buf = 'a';
   end_time = clock_gettime_ns();
   printf("time elapsed: %lu nanoseconds \n");
}
int main(int argc, char **argv) {
   char *buf = mmap(NULL, page_size, PROT_READ | PROT_WRITE,
                    MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i += page_size/4)
       foo(buf+i);

   char *buf = mmap(NULL, page_size, PROT_READ | PROT_WRITE,
                    MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
       foo(buf+i);
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }
   char *buf = mmap(NULL, page_size, PROT_WRITE,
                    MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
      foo(buf+i);
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }
   char *buf = mmap(NULL, page_size, PROT_WRITE,
                    MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
      foo(buf+i);
      mprotect(buf, page_size, PROT_READ );
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
}}
```

```
➜  git:(master) ✗  ./test
time elapsed: 1417 nanoseconds
time elapsed: 125 nanoseconds
time elapsed: 334 nanoseconds
time elapsed: 125 nanoseconds
---------------------------------
time elapsed: 3084 nanoseconds
time elapsed: 42 nanoseconds
time elapsed: 125 nanoseconds
time elapsed: 84 nanoseconds
--------------------------------
time elapsed: 2334 nanoseconds
time elapsed: 3459 nanoseconds
time elapsed: 125 nanoseconds
time elapsed: 125 nanoseconds
---------------------------------
```

Tracking PTE invalidations with mprotect

```c
void foo(char *buf) {
   start_time = clock_gettime_ns();
   *buf = 'a';
   end_time = clock_gettime_ns();
   printf("time elapsed: %lu nanoseconds \n");
}
int main(int argc, char **argv) {
   char *buf = mmap(NULL, page_size, PROT_READ | PROT_WRITE,
                        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i += page_size/4)
       foo(buf+i);

   char *buf = mmap(NULL, page_size, PROT_READ | PROT_WRITE,
                        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
       foo(buf+i);
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }
   char *buf = mmap(NULL, page_size, PROT_WRITE,
                        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
      foo(buf+i);
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }
   char *buf = mmap(NULL, page_size, PROT_WRITE,
                        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   for (int i = 0; i < page_size; i+=page_size/4) {
      foo(buf+i);
      mprotect(buf, page_size, PROT_READ );
      mprotect(buf, page_size, PROT_WRITE | PROT_READ );
   }}
```

```
➜ git:(master) ✗  ./test
time elapsed: 1417 nanoseconds
time elapsed: 125 nanoseconds
time elapsed: 334 nanoseconds
time elapsed: 125 nanoseconds
---------------------------------
time elapsed: 3084 nanoseconds
time elapsed: 42 nanoseconds
time elapsed: 125 nanoseconds
time elapsed: 84 nanoseconds
---------------------------------
time elapsed: 2334 nanoseconds
time elapsed: 3459 nanoseconds
time elapsed: 125 nanoseconds
time elapsed: 125 nanoseconds
---------------------------------
time elapsed: 1459 nanoseconds
time elapsed: 2000 nanoseconds
time elapsed: 2125 nanoseconds
time elapsed: 5167 nanoseconds
---------------------------------
```

Measuring row- vs column-major mem. access time

```
int main(int argc, char **argv) {
    int *buf  =  malloc(10000*sizeof(int *));
   for (int i = 0; i < 1000; i++)
        buf[i]  =  malloc(10000*sizeof(int));

   for (int i = 0; i < 1000; i++)
     for (int j = 0; i < 1000; j++)
         buf[i][j] = i + j;

   start = clock_gettime_ns();
   for (int i = 0; i < 1000; i++)
     for (int j = 0; i < 1000; j++)
         buf[i][j]  = i + j;  // access row-major
   end = clock_gettime_ns();
   printf("%.1f ns per access\n", (end - start) / e8 );

   start_time = clock_gettime_ns();
   for (int i = 0; i < 1000; i++)
     for (int j = 0; i < 1000; j++)
         buf[j][i]  = i + j;  // access column-major
   end_time = clock_gettime_ns();
   printf("%.1f ns per access\n", (end - start) / e8 );
```

```
➜  git:(master) ✗  ./test
0.7 nanoseconds per write access
```

Measuring row- vs column-major mem. access time

```c
int main(int argc, char **argv) {
   int *buf = malloc(10000*sizeof(int *));
   for (int i = 0; i < 1000; i++)
       buf[i] = malloc(10000*sizeof(int));

   for (int i = 0; i < 1000; i++)
     for (int j = 0; i < 1000; j++)
         buf[i][j] = i + j;

   start = clock_gettime_ns();
   for (int i = 0; i < 1000; i++)
     for (int j = 0; i < 1000; j++)
         buf[i][j] = i + j;  // access row-major
   end = clock_gettime_ns();
   printf("%.1f ns per access\n", (end - start) / e8 );

   start_time = clock_gettime_ns();
   for (int i = 0; i < 1000; i++)
     for (int j = 0; i < 1000; j++)
         buf[j][i] = i + j;  // access column-major
   end_time = clock_gettime_ns();
   printf("%.1f ns per access\n", (end - start) / e8 );
```

```
➜  git:(master) ✗  ./test
0.7 nanoseconds per write access
4.5 nanoseconds per write access
```