STOS - Pagination

Gabriel Laskar <gabriel@lse.epita.fr>

Vocabulary

- MMU
- Page
- Frame
- Address Space
- Logical Address
- Linear Address
- Physical Address

Virtual Memory?

- Separate Address Space (per-process)
- “Map” a virtual address to a physical address
- Fine grained allocation (Page granularity)
- Per-page permissions (R/W, U/S)

Module : pagination.ko

```
MODINFO {
    module_name("pagination"),
    module_init_once(pagination_init),
    module_type(M_PAGINATION | M_PAGE_ALLOCATOR),
    module_deps(M_INTERRUPTS | M_FRAME_ALLOCATOR)
};

EXPORT_SYMBOL(alloc_pages);
EXPORT_SYMBOL(map_pages);
EXPORT_SYMBOL(map_io);
EXPORT_SYMBOL(unmap_pages);
```

Pagination API (<kernel/page.h>)

```
void* alloc_pages(struct frame* pdbr, size_t n);

int map_pages(struct frame* pdbr, void* vaddr,
              struct frame* frames[],
              size_t n, int flags);

void unmap_pages(struct frame* pdbr, void* vaddr, int
n);

void* map_io(phys_t ioaddr, size_t len);
```

Code already in STOS

- Macros and Flags are defined inside `<kernel/arch/page.h>`
- Generic page flags are consumed by the api
  - `P_KERNEL`
  - `P_USER_RO`
  - `P_USER_RW`
- “`frame_allocator.ko`” is here to allocate physical frames.
- `<kernel/memory.h>` contains physical memory informations

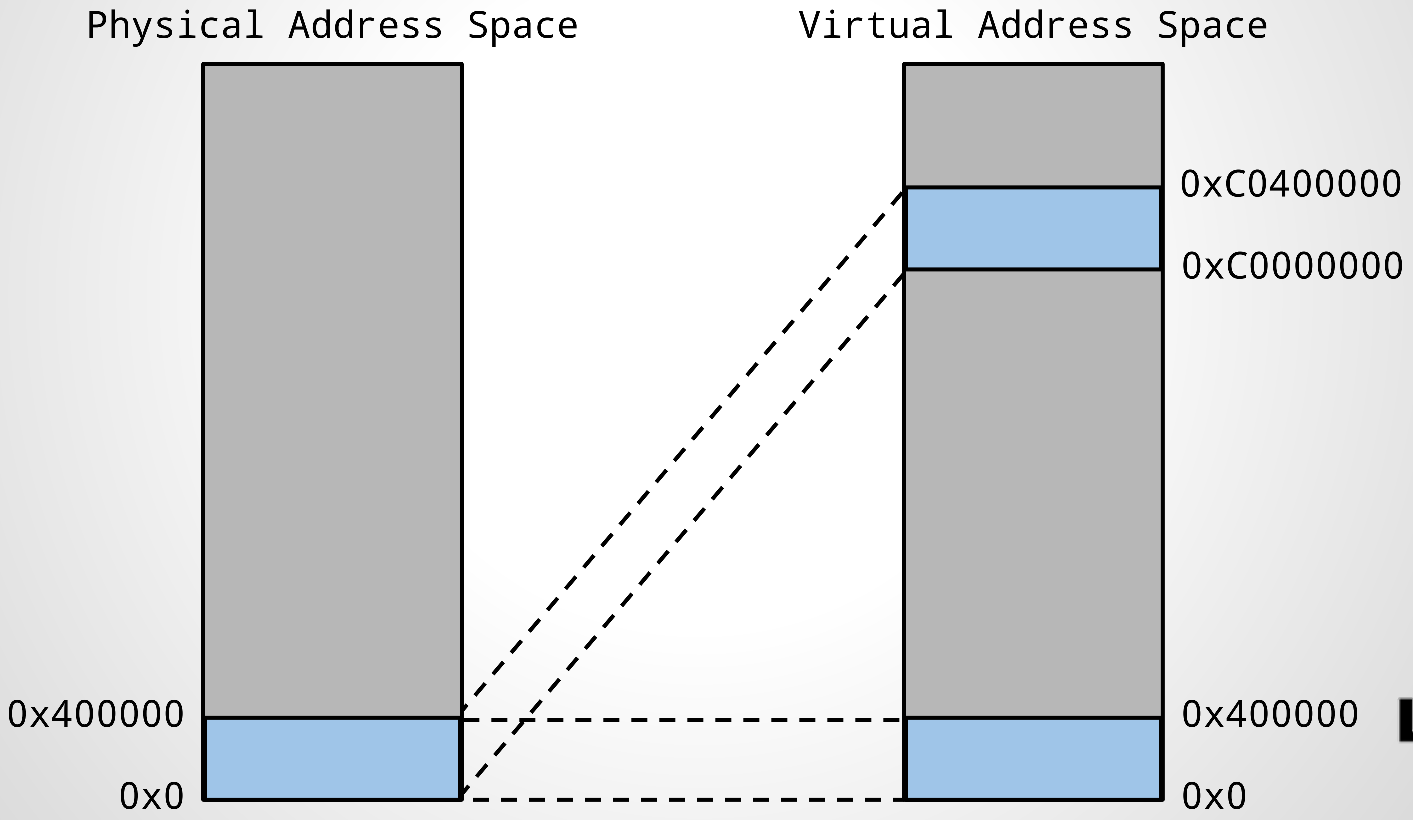
Frame Allocator (`<kernel/frame.h>`)

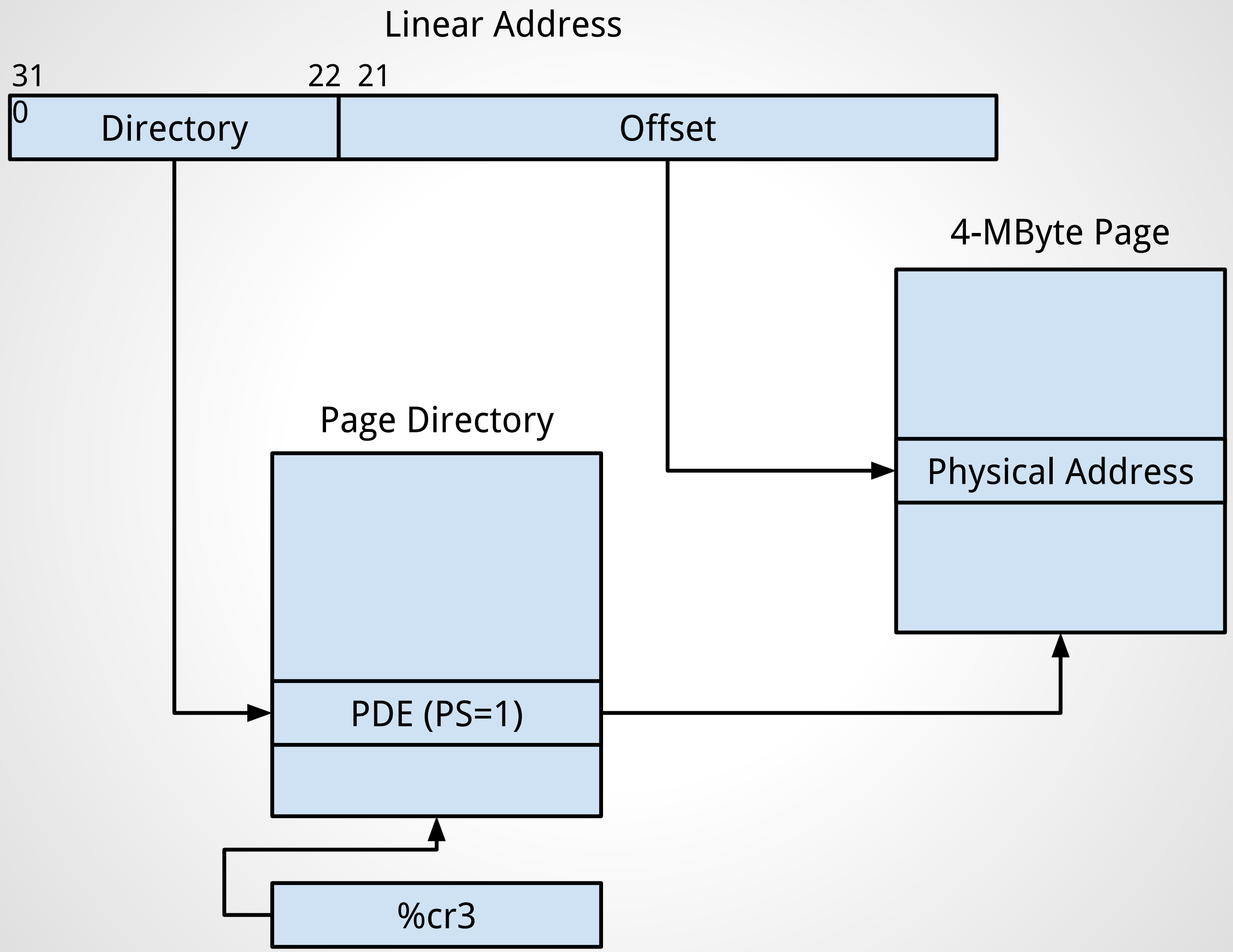
```c
struct frame* alloc_frame(void);
void free_frame(struct frame* frame);

int alloc_frames(u8 n, struct frame** frames);

static inline phys_t frame_to_phys(struct frame* f);
static inline struct frame* phys_to_frame(phys_t
addr);
```

More on frame allocator

- frames are stored in an array (from `framestart` through `frameend`)
- index of a frame is the frame number
- inside a frame, you must maintain the `vaddr`
- kernel maintain only a free list

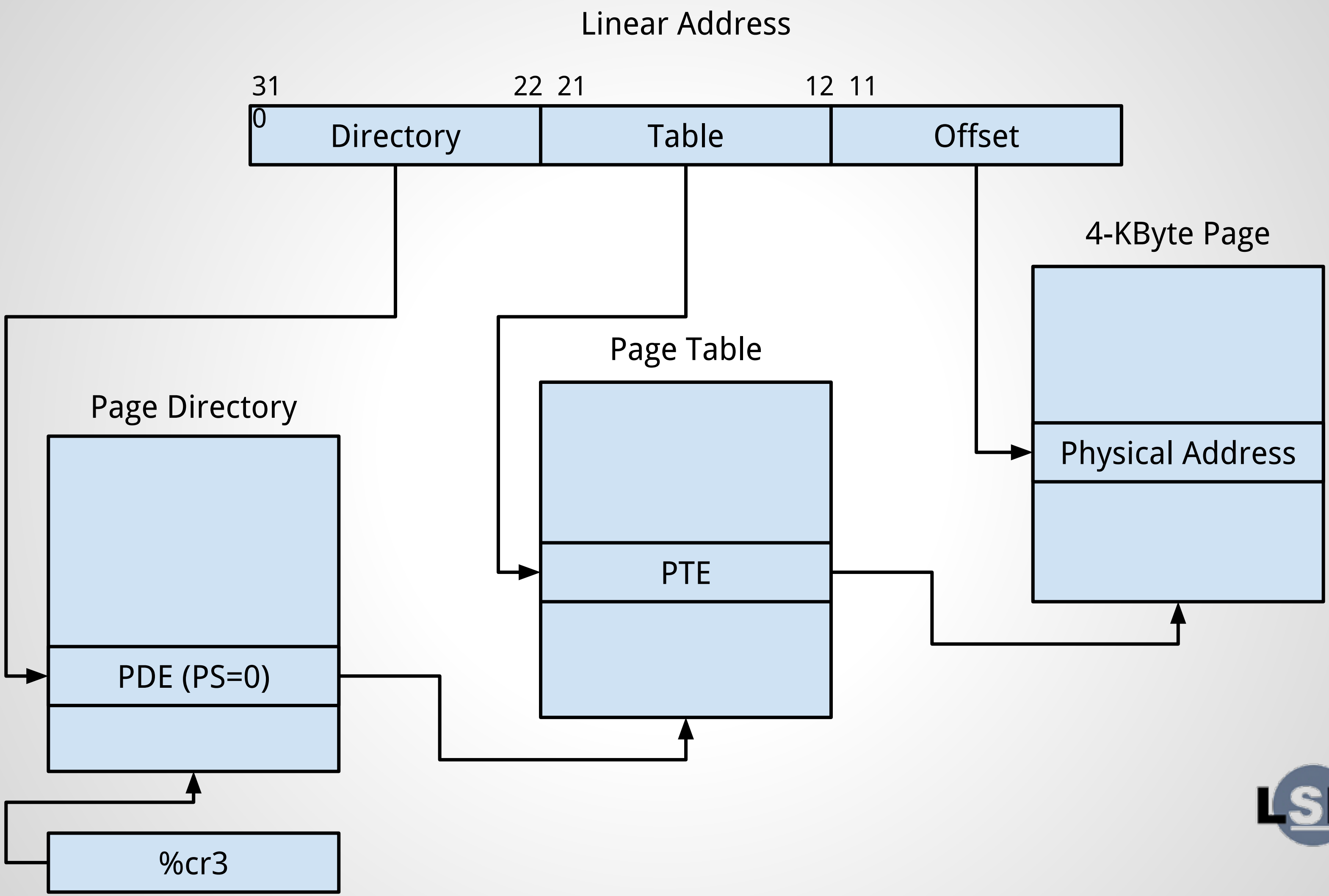
Reminder: Current paging state

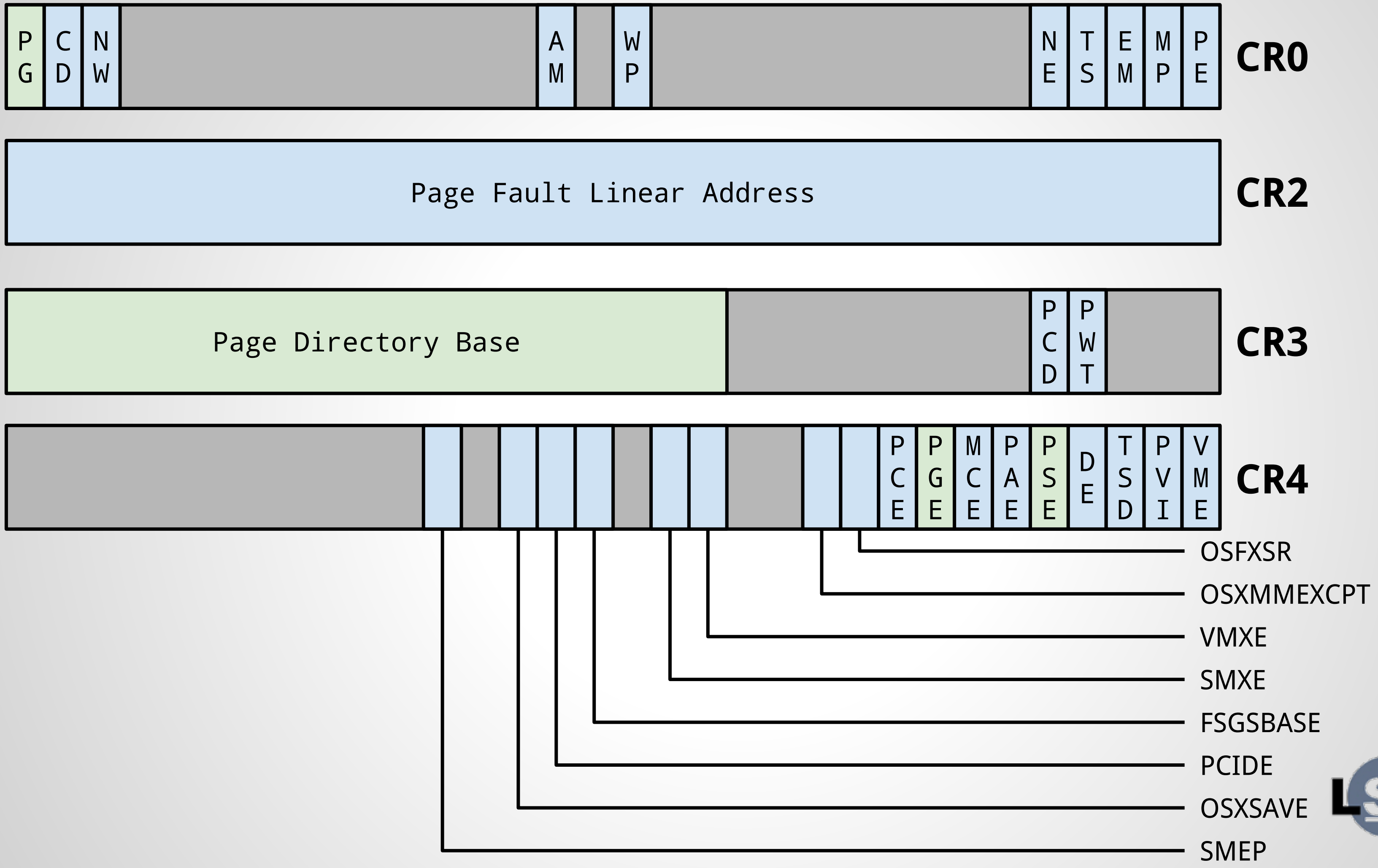



Linear Address
31
22
21
0
Directory
Offset
4-MByte Page
Physical Address
Page Directory
PDE (PS=1)
%cr3

Pagination on x86_32 (vol 3a, chapter 4)

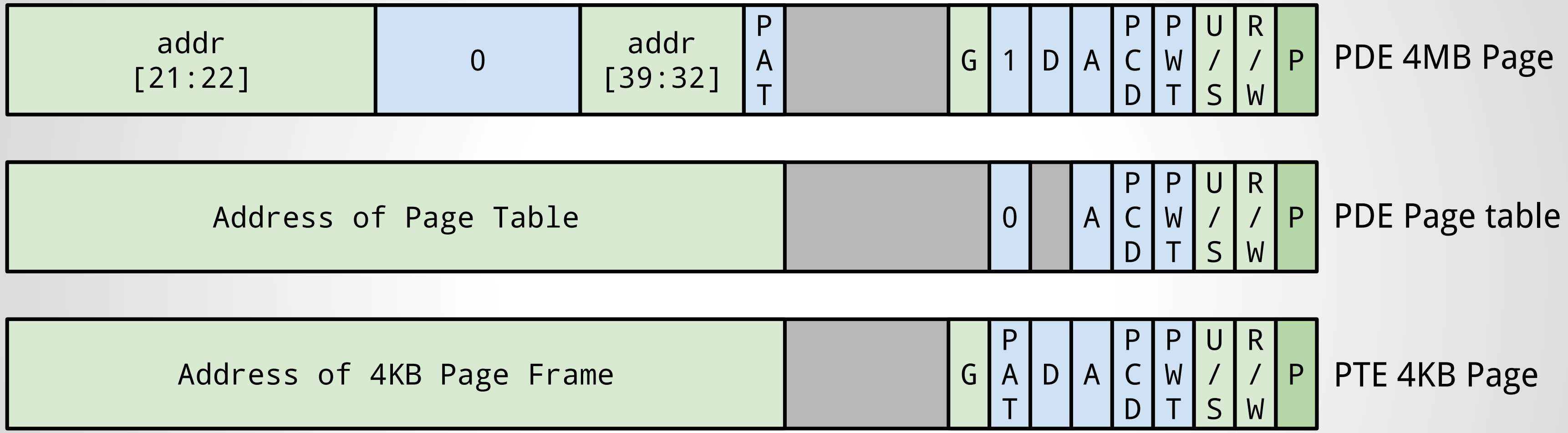
- 3 paging modes for x86
- multiple page sizes
- control registers used:
  - %cr0: activation (**PG**, WP)
  - %cr2: Page Fault Linear Address
  - %cr3: Page Directory Base
  - %cr4: extra features (PAE, **PSE**, **PGE**, PCIDE, SMEP)
  - IA32_EFER MSR: (LME, NXE)


Linear Address
31
0
22
21
12
11
Directory
Table
Offset
4-KByte Page
Page Table
Page Directory
Physical Address
PTE
PDE (PS=0)
%cr3



PG
CD
NW
AM
WP
NE
TS
EM
MP
PE
CR0
Page Fault Linear Address
CR2
Page Directory Base
PCD
PWT
CR3
PCE
PGE
MCE
PAE
PSE
DE
TSD
PVI
VME
CR4
OSFXSR
OSXMMEXCPT
VMXE
SMXE
FSGSBASE
PCIDE
OSXSAVE
SMEP

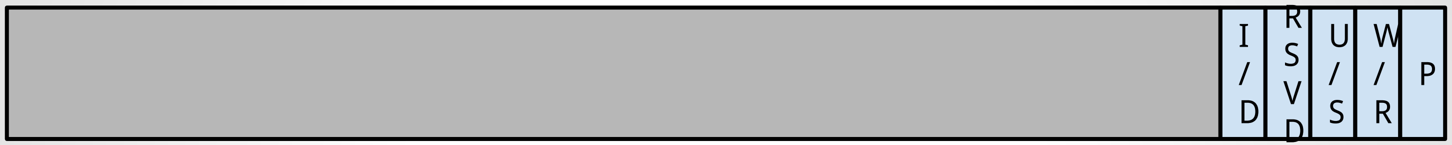
PDE and PTE



- R/W: Read/Write
- U/S: User/System
- PWT: Page Level write-through
- PCD: Page Level Cache disable

- A: Accessed
- D: Dirty
- G: Global (if `%cr4.pge = 1`)
- PAT: Reserved

Page Fault Handling



- Which address? Content of `%cr2`
- Error Code:
    - P: non-present (clear), page-level protection violation (set)
    - W/R: read (clear) or write (set) error
    - U/S: supervisor (clear) or user-mode (set)
    - RSVD: reserved bit violation (set)
    - I/D: data (clear) or instruction (set)

Translation Lookaside Buffer

- Page Directory Walk costs
- TLB cache the page walks
- TLB is automatically managed on x86
- Flush on :
  - mov %cr3
  - on change of some %cr registers
  - `invlpg addr`: instruction