CS 444/544 OS II
Lab Tutorial #4

Virtual Memory Management

Prof. Sibin Mohan | Spring 2022

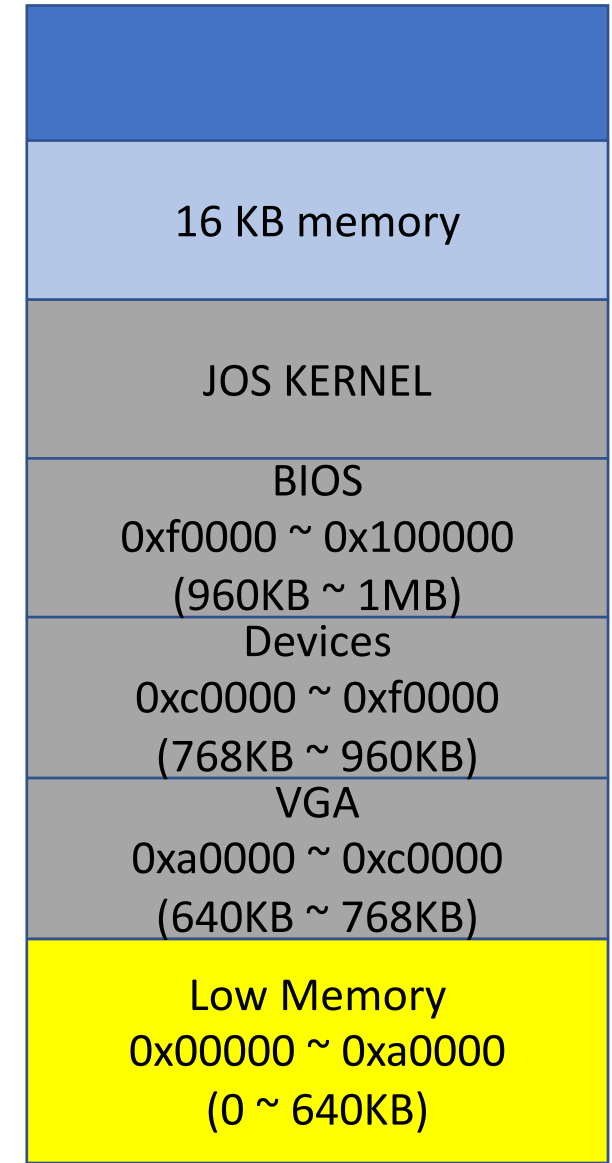
Adapted from content originally created by: Prof. Yeongjin Jang

Overview: Lab 2 Memory Management #1

- Manage Physical Memory
  - Use some part of it for peripheral device
  - Use some part of it for code/data
  - Maintain unused space and allocate as system requires more memory

Lab 2 Exercise 1 is for implementing this part!

```
boot_alloc()
mem_init()
page_init()
page_alloc()
page_free()
```

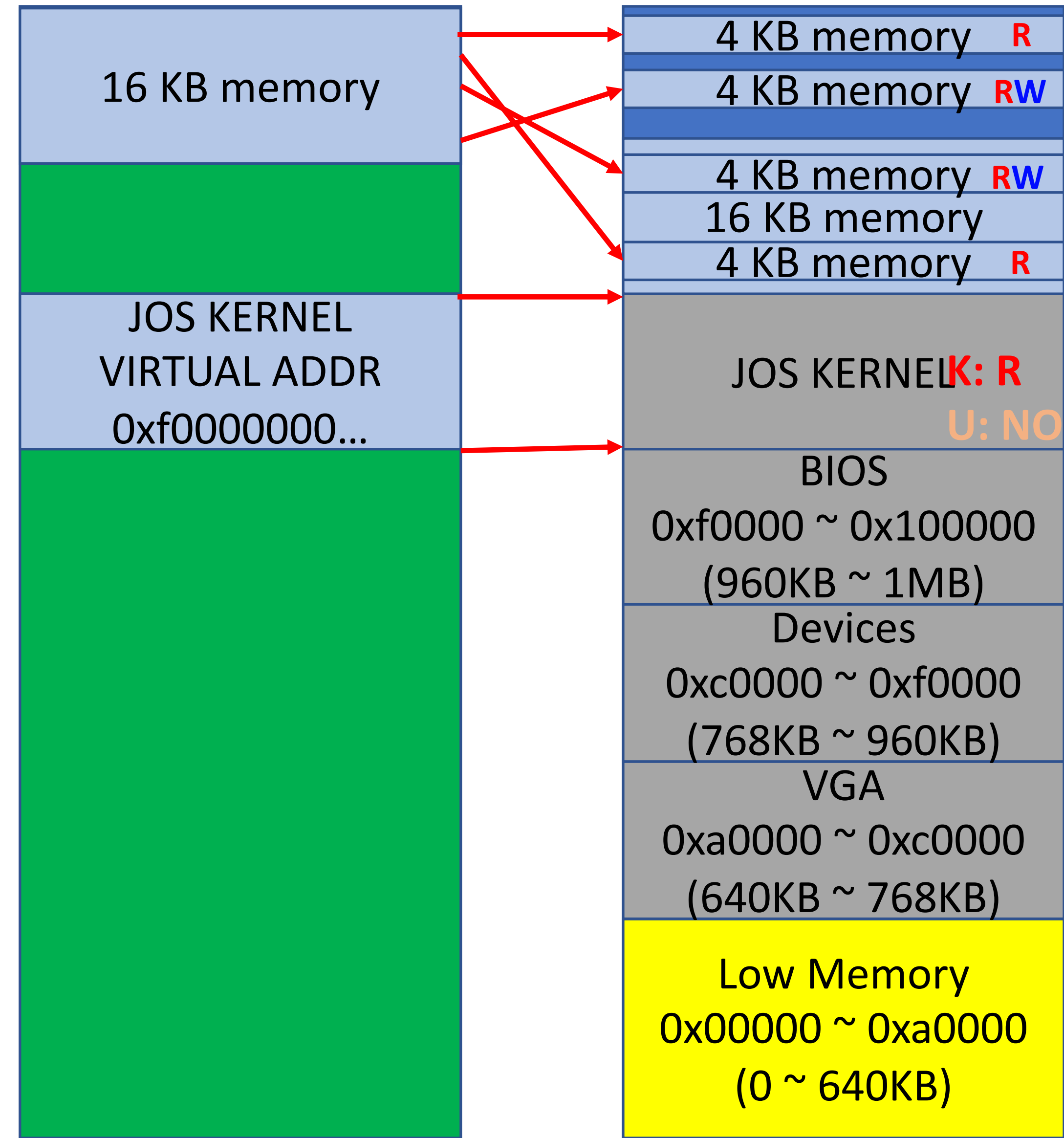


Overview: Lab 2 Memory Management #2

- Manage Virtual Memory
  - Virtual address mappings
  - Permission setup and access control

Lab 2 Exercise 4&5 is for implementing this part!

```
pgdir_walk()
boot_map_region()
page_lookup()
page_remove()
page_insert()
```



Physical address *vs* Virtual address

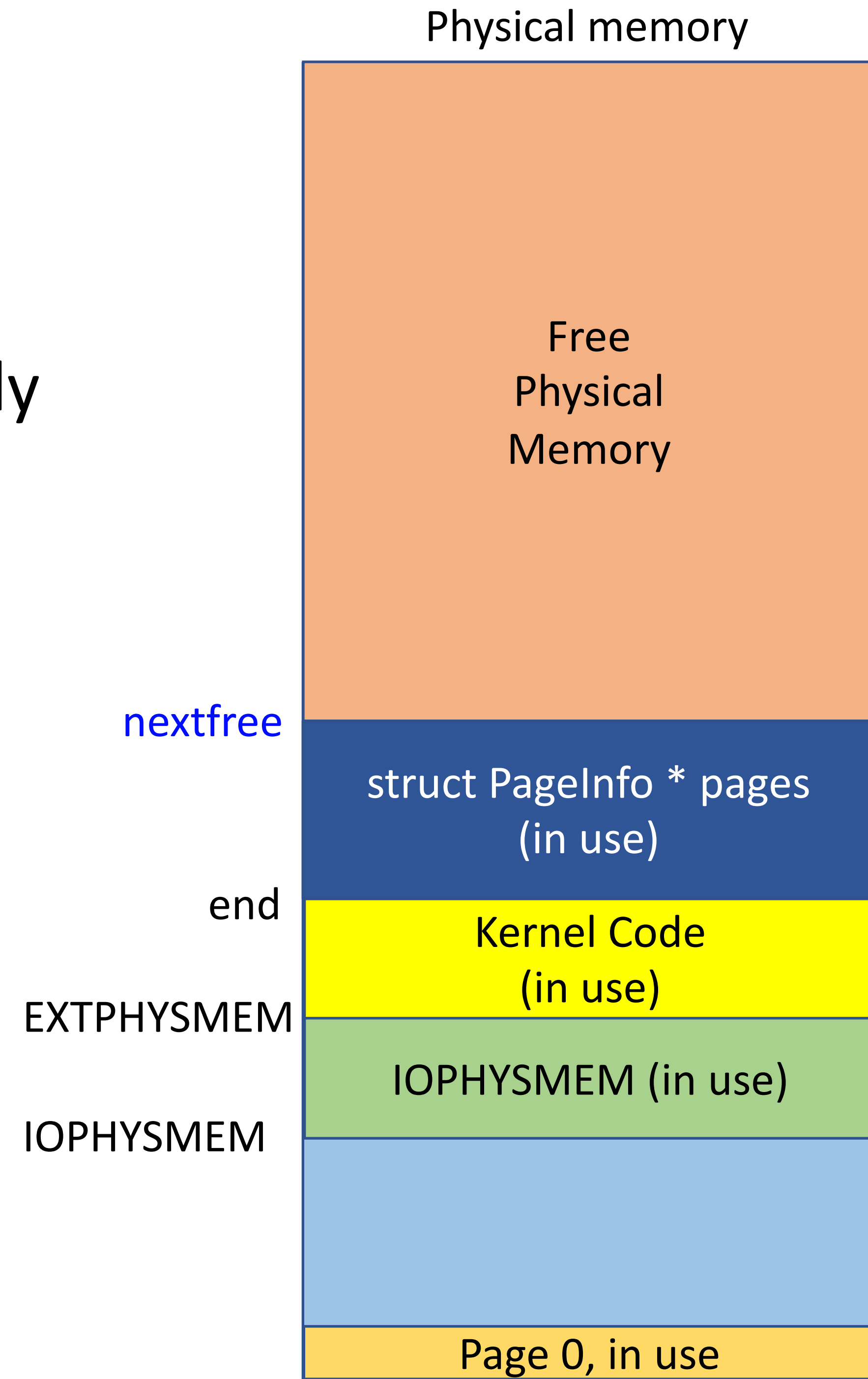
- All addresses that you can access from your C code is **virtual address**
- E.g., nextfree contains a virtual address
- Size of maximum physical memory: `npages * PGSIZE`
  - The limit is based on the **physical address**
  - How to get the **physical address**: `PADDR(virtual_address)`
- `if ((PADDR(nextfree) > npages * PGSIZE)`
  - `panic("out of memory");`

Don’t know which address a variable stores?

- Print it!
  - cprintf(“Address: %p”, addr);
- If the address is above 0xf0000000
  - I.e., 0xf0000000 ~ 0xffffffff
  - Then the address is a virtual address
- If the address is below 0xf0000000
  - E.g., 0x800000 or 0x102030
  - Then the address is a physical address

page_init()

- Take a look at the memory mapping thoroughly
- Mark all in-use pages as `pp_ref = 1`
- Link all pages with pp_ref = 0
  - Create a linked list with `page_free_list`



Others…

- page_alloc()
  - Get a free physical page from `page_free_list`
    - Manage head, etc. to keep the linked list correctly
  - Clear memory if the flag is with ALLOC_ZERO
    - i.e., `if (alloc_flags & ALLOC_ZERO == 1)`
    - Run `memset(addr, 0, PGSIZE)`
    - Must use kernel virtual address of the page to do this; `page2kva(pp)` will help you
- Page_free()
  - Do not adjust `pp_ref` here… it will be controlled by `page_decref()`
  - Just manage the `page_free_list`…

```
void
page_decref(struct PageInfo* pp)
{
    if (--pp->pp_ref == 0)
        page_free(pp);
}
```

Assertion Errors

```
444544 decimal is 1544200 octal!
Physical memory: 131072K available, base = 640K, extended = 130432K
check_page_free_list() succeeded!
kernel panic at kern/pmap.c:695: assertion failed: c[i] == 0
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
K>
```

- In kern/pmap.c, we have many function named check_...
  - These functions are there for running **sanity check** of your implementation
- What does it mean if an assertion fails?
  - An assertion failure means that your implementation is **incorrect**

Assertion Errors

- How can I debug this?
  - Understand the meaning and the context of the assertion
- It’s about ALLOC_ZERO check
  - check page_alloc()!

```
444544 decimal is 1544200 octal!
Physical memory: 131072K available, base = 640K, extended = 130432K
check_page_free_list() succeeded!
kernel panic at kern/pmap.c:695: assertion failed: c[i] == 0
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
K>
```

```
691     // get the virtual address of pp
692     c = page2kva(pp);
693     // check if all content of the page is 0x0 (not 0x1)
694     for (i = 0; i < PGSIZE; i++)
695         assert(c[i] == 0);
696     // then, ALLOC_ZERO is OK
```

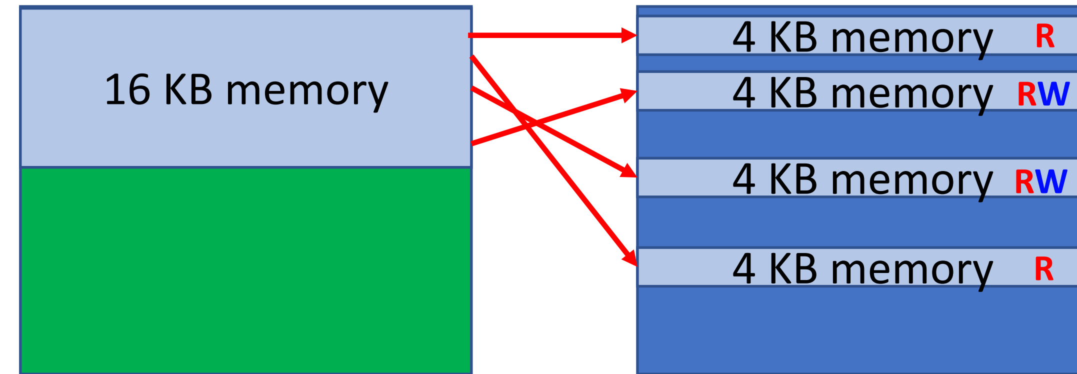
```
struct PageInfo *
page_alloc(int alloc_flags)
{
    // Fill this function in
    struct PageInfo *ret = page_free_list;
    if (ret == NULL) {
        return NULL;
    }
    page_free_list = page_free_list->pp_link;
    ret->pp_link = 0;
    ret->pp_ref = 0;

    /*
    if (alloc_flags & ALLOC_ZERO) {
        memset(page2kva(ret), 0, PGSIZE);
    }
    */

    return ret;
}
```

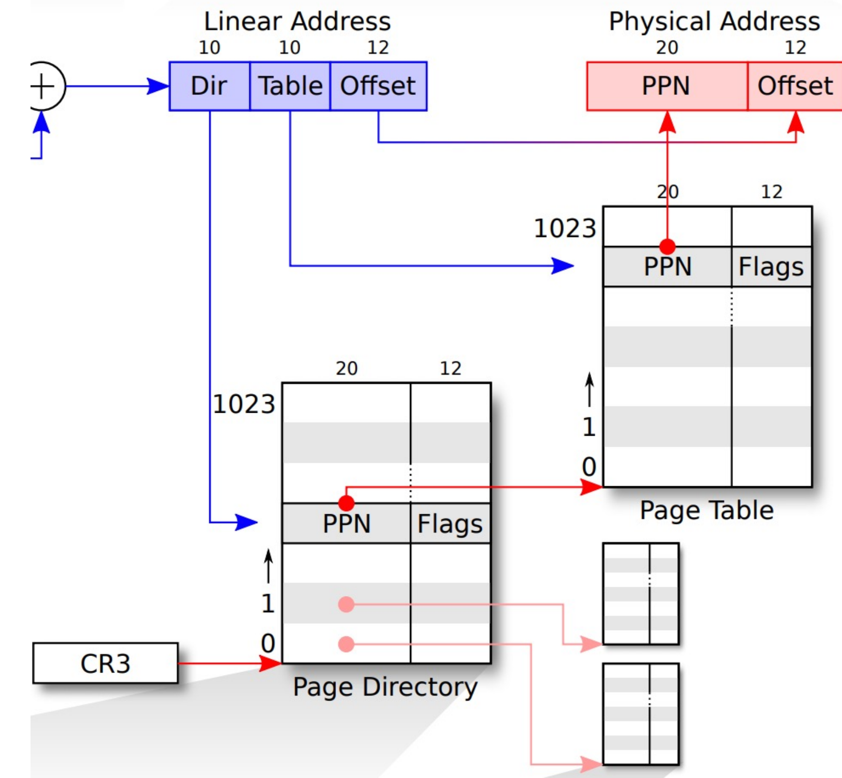
A detailed assertion description is available at here:

https://gist.github.com/blue9057/5efb9807a9e879c75ee3f1c7add93c08

Today’s Topic: Manage Virtual Memory

- Lab2, Exercise 4
  - pgdir_walk()
  - boot_map_region()
  - page_lookup()
  - page_remove()
  - page_insert()





pte_t * pgdir_walk(pgdir, va, create)

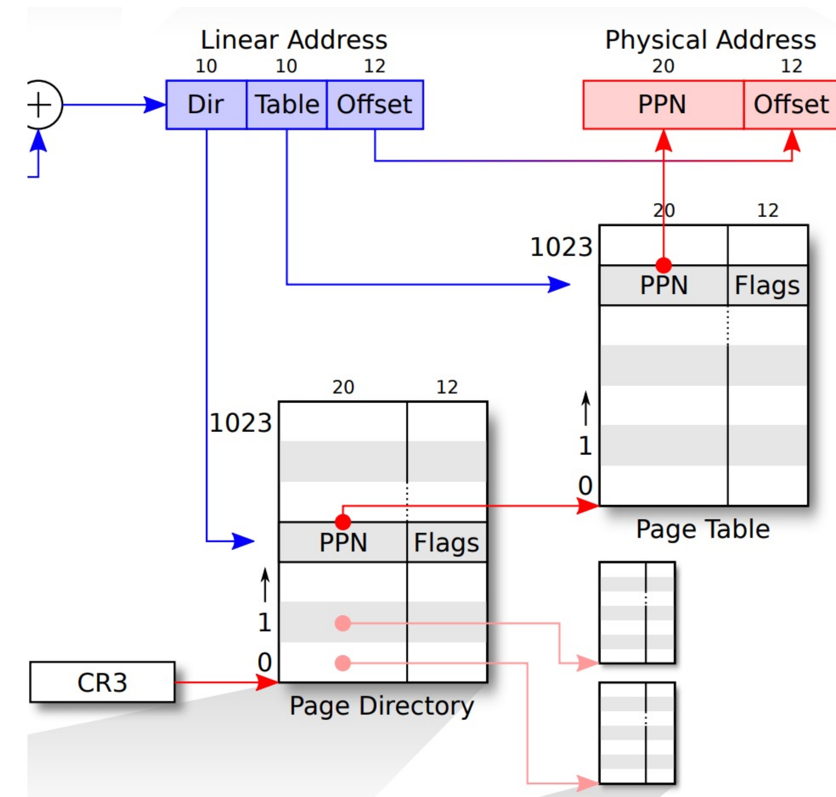
- Returns the address of page table entry pointed by va, from pgdir

- We have 2 level page table structure
  - `pde_t pde = pgdir[PDX(va)]`

  - Before accessing pte from pde, please check if pde is valid
    - `if (pde&PTE_P) { // valid }`

  - If valid, get the page table by:
    - `pde_t * page_table = (pde_t *)KADDR(PTE_ADDR(pde))`
    - **You can access only virtual addresses**

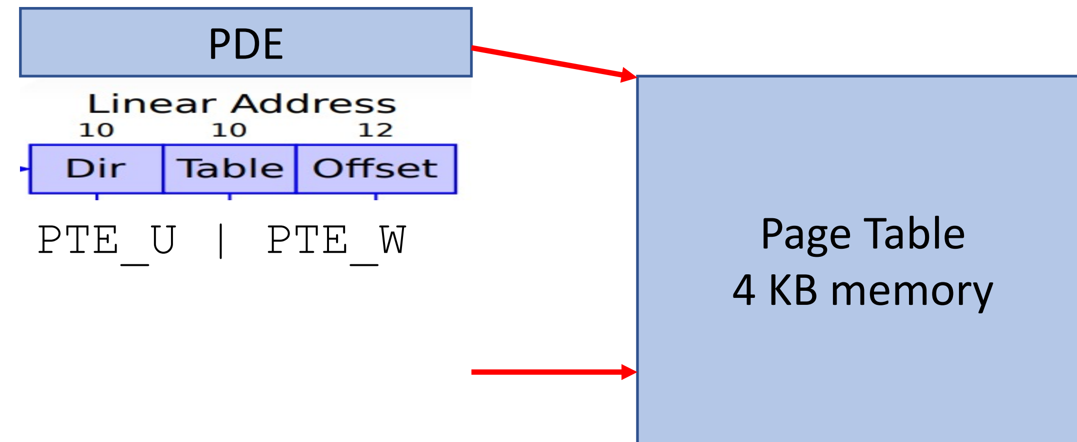
  - Return the address of the entry
    - `return &page_table[PTX(va)];`



Blue: Physical address
Red: Virtual address

pte_t * pgdir_walk(pgdir, va, create)

- Before accessing `pte` from `pde`, please check if `pde` is valid
  - `if (pde&PTE_P) { // valid } else { // invalid }`
- If `pde` invalid, create a new Page Table by:
  - call **pp_page_table** `= page_alloc();` to allocate a Page Table.
    - Do **pp_page_table**`->pp_ref += 1;`
  - Set pde accordingly
    - **pgdir**`[PDX(va)] =`
      - **page2pa(pp_page_table)** `| PTE_P | PTE_U | PTE_W`
  - Return the address of the entry
    - `return &`**page_table**`[PTX(va)];`



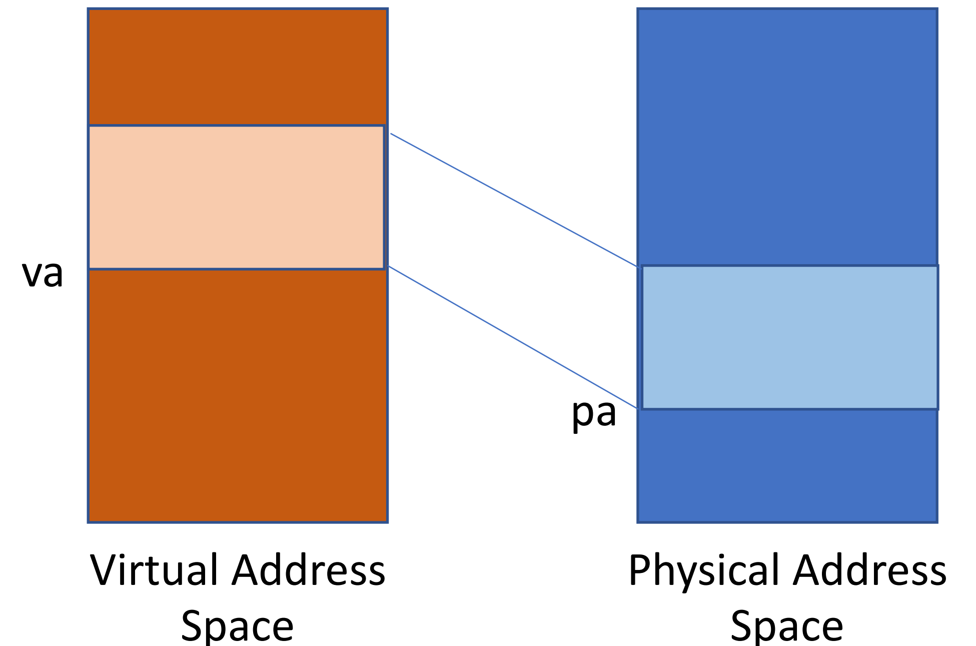
DO NOT GET CONFUSED about VA and PA

- All variables in C need to be
  - **VIRTUAL ADDRESS** to access them in C (0xf0000000 ~ 0xffffffff)

- All addresses stored into PDEs and PTEs are must be
  - **PHYSICAL ADDRESS** (0x00000000 ~ 0xefffffff)

boot_map_region(pgdir, va, size, pa, perm)

- Map multiple virtual pages to physical pages linearly
- Mapping a virtual address to a physical address can be done by
  - Setting the corresponding PTE!
- For a virtual address va, please do:
  - pte_t * **p_pte** = pgdir_walk(**pgdir**, **va**, 1);
  - ***p_pte** = **PTE_ADDR**(**pa**) | PTE_P | perm
- DO NOT increment pp_ref…



page_lookup(pgdir, va, pte_store)

- Get the address of page table entry that is corresponding to va and store that into pte_store
  - pte_store is used for returning a pointer
- This function returns two values
  - **struct PageInfo *** as the return value
  - **pte_t ***, storing it via pte_store

page_lookup(pgdir, va, pte_store)

- How?

- Returning the address of pte
  - `pte_t *p_pte = pgdir_walk(pgdir, va, 0);`
  - `if (pte_store != NULL) { *pte_store = p_pte; }`

- Returning Struct PageInfo *
  - `return pa2page(PTE_ADDR(*p_pte));`

page_remove(pgdir, va)

- Unmap the virtual address

- Set the corresponding page table entry to 0
  - `pte_t * p_pte;`
  - `Struct PageInfo *pp = page_lookup(pgdir, va, &p_pte);`

  - `*p_pte = 0; // if it was mapped`
  - `page_decref(pp);`
  - `tlb_invalidate(pgdir, va);`

page_insert(pgdir, pp, va, perm)

- Map a page pointed by pp to va, with permissions in perm

- VA to PA
  - What is PA of pp?
    - page2pa(pp);

- How to get the page table entry of a va?
  - pte_t *p_pte = pgdir_walk(pgdir, va, 1);

- Assign
  - *p_pte = PTE_ADDR(page2pa(pp)) | perm | PTE_P

page2pa(pp)
Physical addr

perm

pgdir_walk()
pte for the VA

page_insert(pgdir, pp, va, perm)

- If pte corresponds to va has already be mapped to a pa,
  - then unmap it!

- Think wisely to handle mapping to the same address
  - Existing pte maps va to pa
  - page_insert will do nothing for this

- READ:

```
Corner-case hint: Make sure to consider what happens when the same
pp is re-inserted at the same virtual address in the same pgdir.
However, try not to distinguish this case in your code, as this
frequently leads to subtle bugs; there's an elegant way to handle
everything in one code path.
```

Tips

- Whenever you update page table entry, please invalidate TLB of that VA
  - `tlb_invalidate(pgdir, va);`
- Do not get confused when to use VA and when to use PA
  - Don't know? Print it.
  - 0xf0123456 ← starting with f, this means **VA**
  - 0x00123456 ← not starting with f, this means **PA**
- pp_ref
  - Add if required, in `pgdir_walk()` and `page_insert()`
  - Call `page_decref()` in `page_remove()`

Tips

- Triple fault
  - Attach gdb, and continue -> GDB stops at the faluting instruction

- Assertion error
  - Utilize `backtrace` to locate the bug and use gdb to check variables, etc.
  - Take a look at functions starting with check_... to understand what grading code would like to check from your implementation
  - https://gist.github.com/blue9057/5efb9807a9e879c75ee3f1c7add93c08
- Avoiding using gdb?
  - Use lots of cprintf to check pointers, integers, variables, etc.
  - %p %x %d etc…