Anita's Super Awesome Recitation slides

15/18-213: Introduction to Computer Systems

Virtual Memory, 28 October 2013

Anita Zhang

Boring Stuff

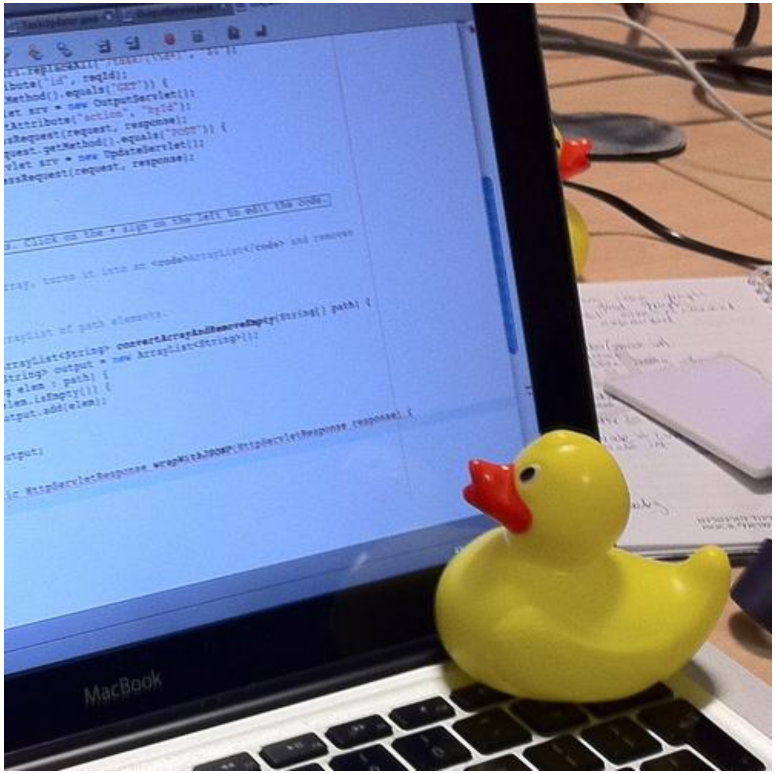
- Shell Lab due tomorrow Tuesday, 29 Oct 2013
  - Your first concurrency assignment!
  - If you like process level concurrency, take OS ☺
- Malloc Lab out the same time Shell Lab is due
  - My favorite lab!
  - Design and implement a memory allocator
  - Start early
- Pressing concerns?

Menu for Today

- Advice, not Motivation
- The Complete Picture
- Virtual Memory
- Address Translation
- Extra: C Primer

Rubber Duck Debugging

"To use this process, a programmer explains code to an inanimate object, such as a rubber duck, with the expectation that upon reaching a piece of incorrect code and trying to explain it, the programmer will notice the error."



The Complete Picture

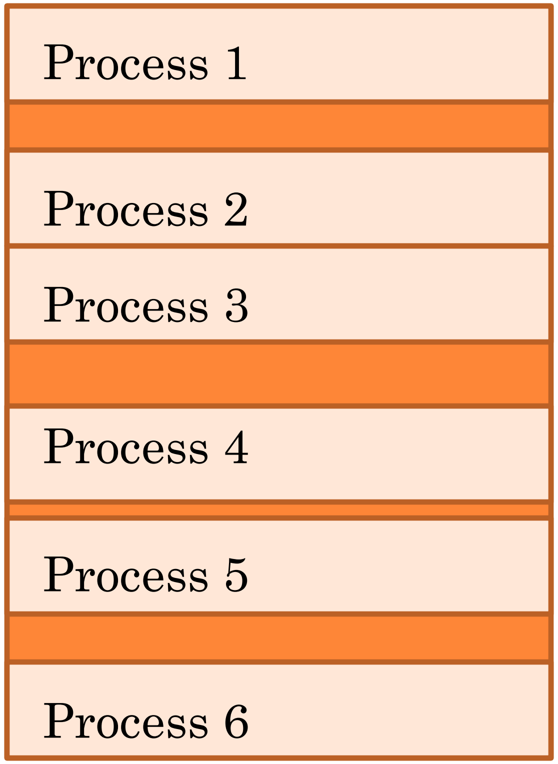
```
#include <stdio.h>

int main(int argc, char **argv) {
    int *val = NULL;
    *val = 0;
    printf("This is %d\n", *val);
    return 0;
}
```

VM: Problems with Direct Mapping

- Questions to ponder:
  - How can we grow processes safely?
  - What to do about fragmentation?
  - How can we make large contiguous chunks fit easier?

Direct Mapping Fragmentation



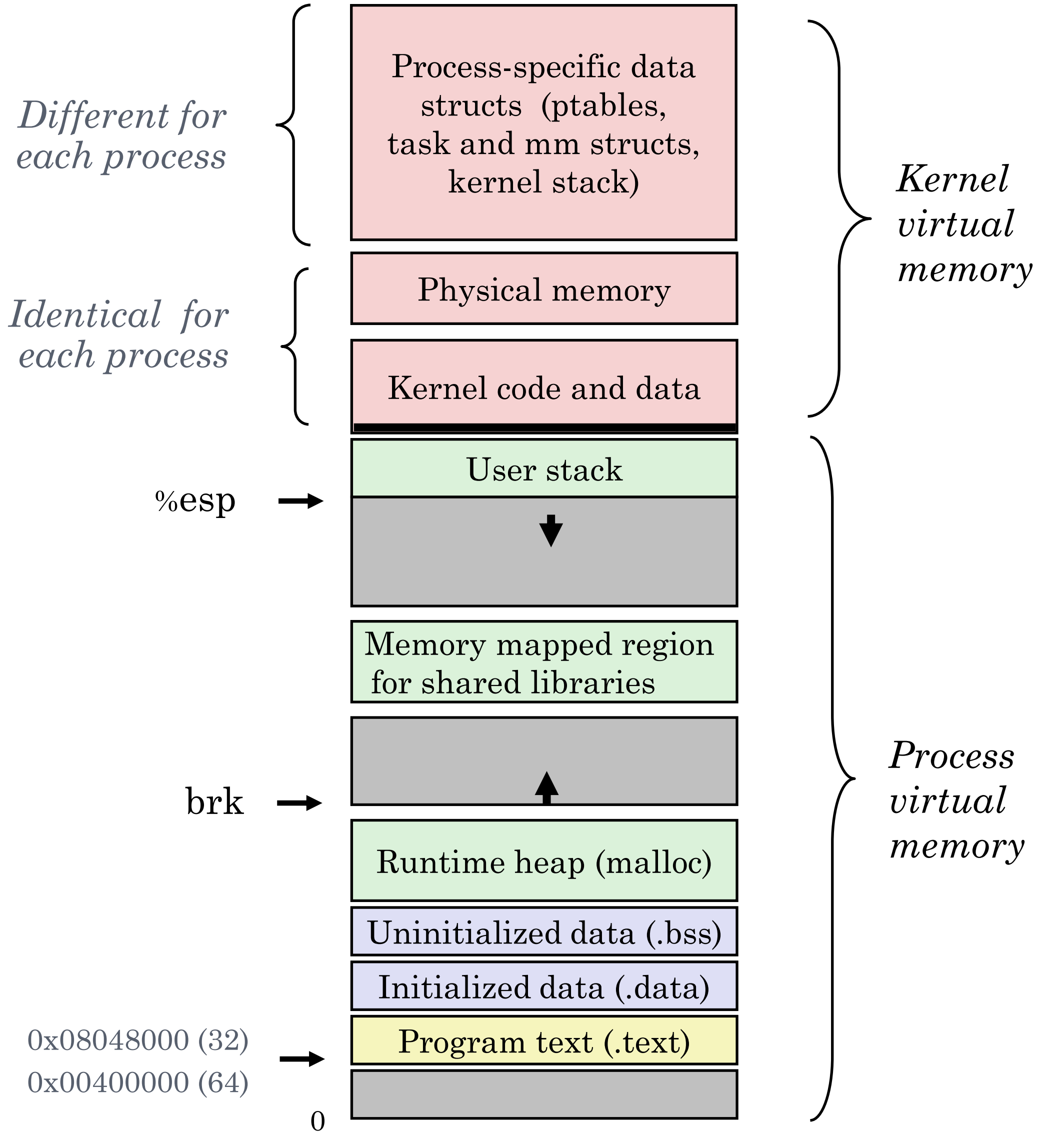
How do we Solve These Problems?

- We are scientists (and engineers)…
  - Insert a level of indirection

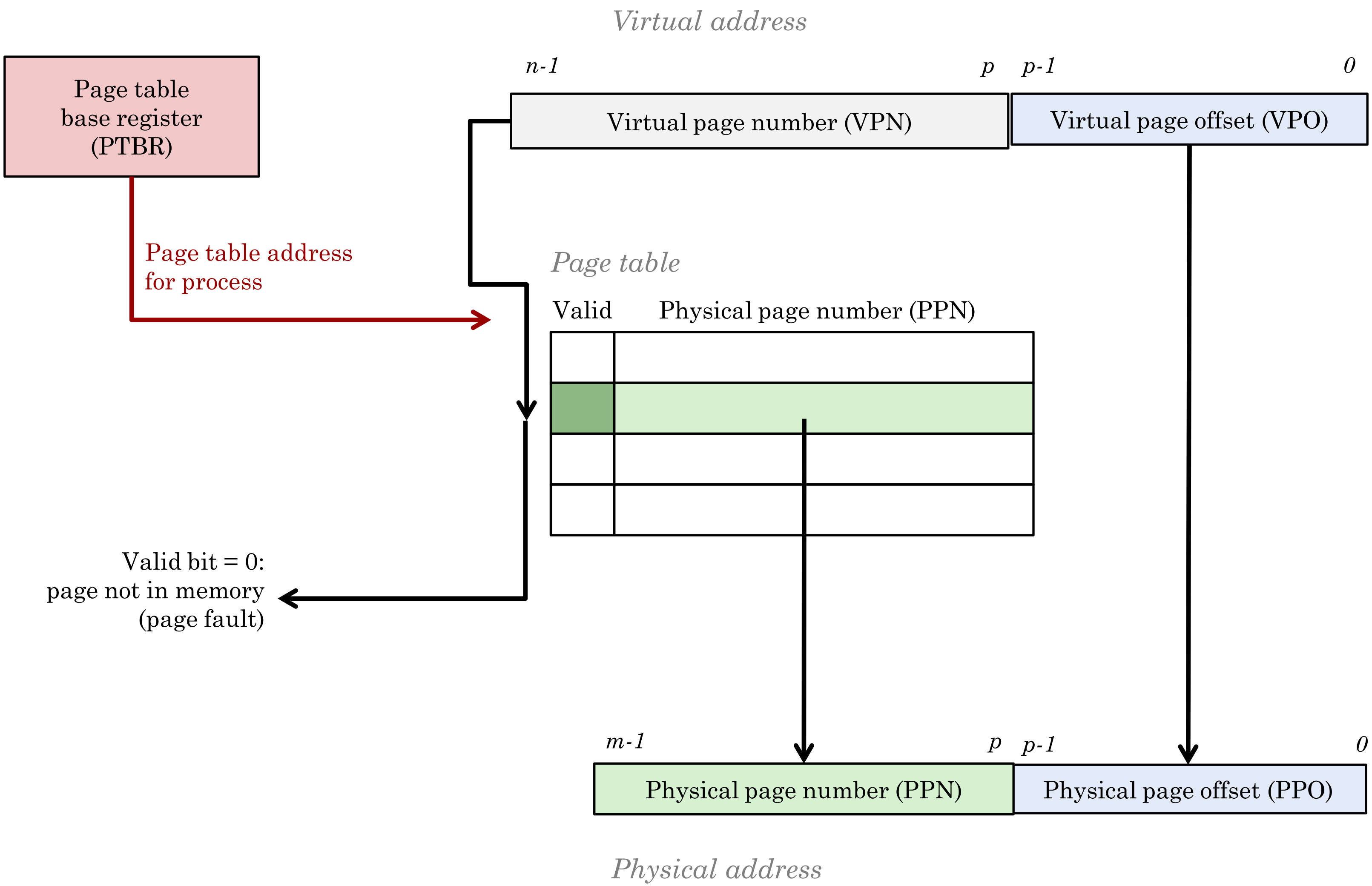
Virtual Memory

- ..Is the Best Thing Ever™
  - Demand paging
  - Memory Management
  - Protection
- Allows the illusion of infinite memory
  - Kernel manages page faults
- Each process gets its own virtual address space
  - Mapping is the heart of virtual memory

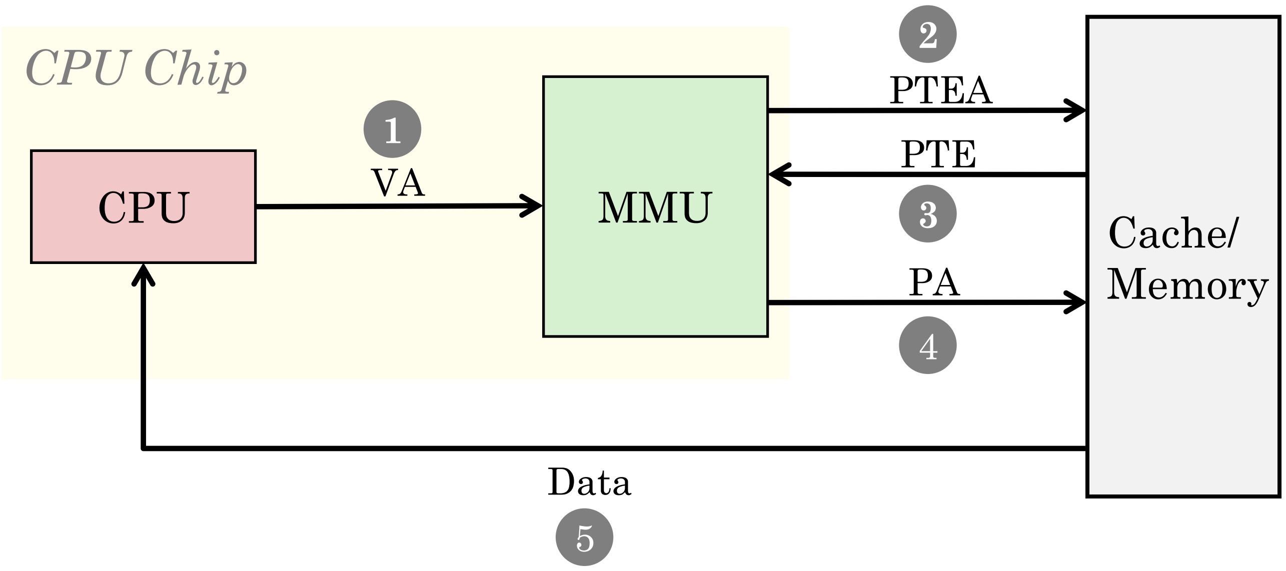
VM of a Linux Process



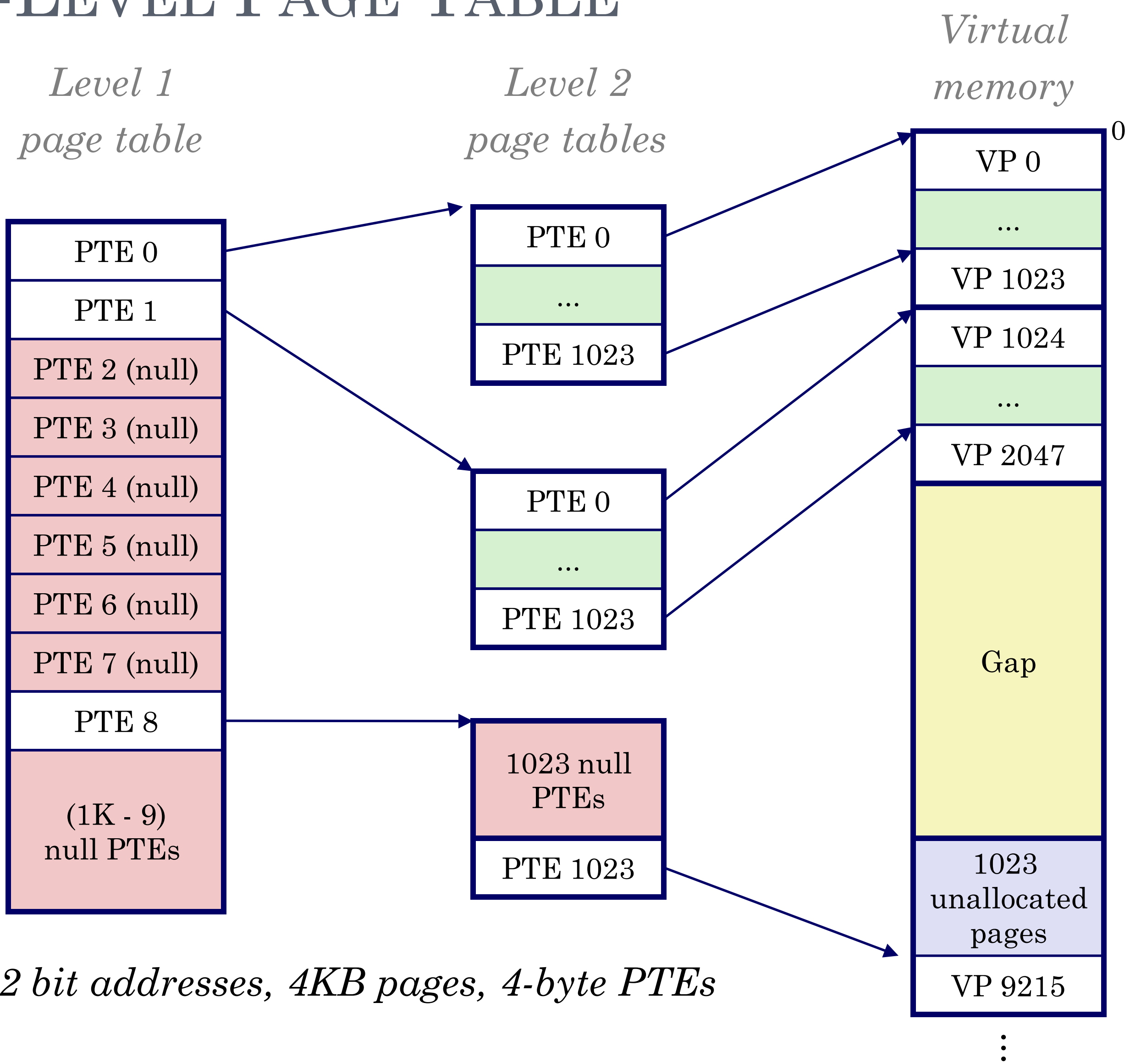
VM: Address Translations



Overview of a Hit

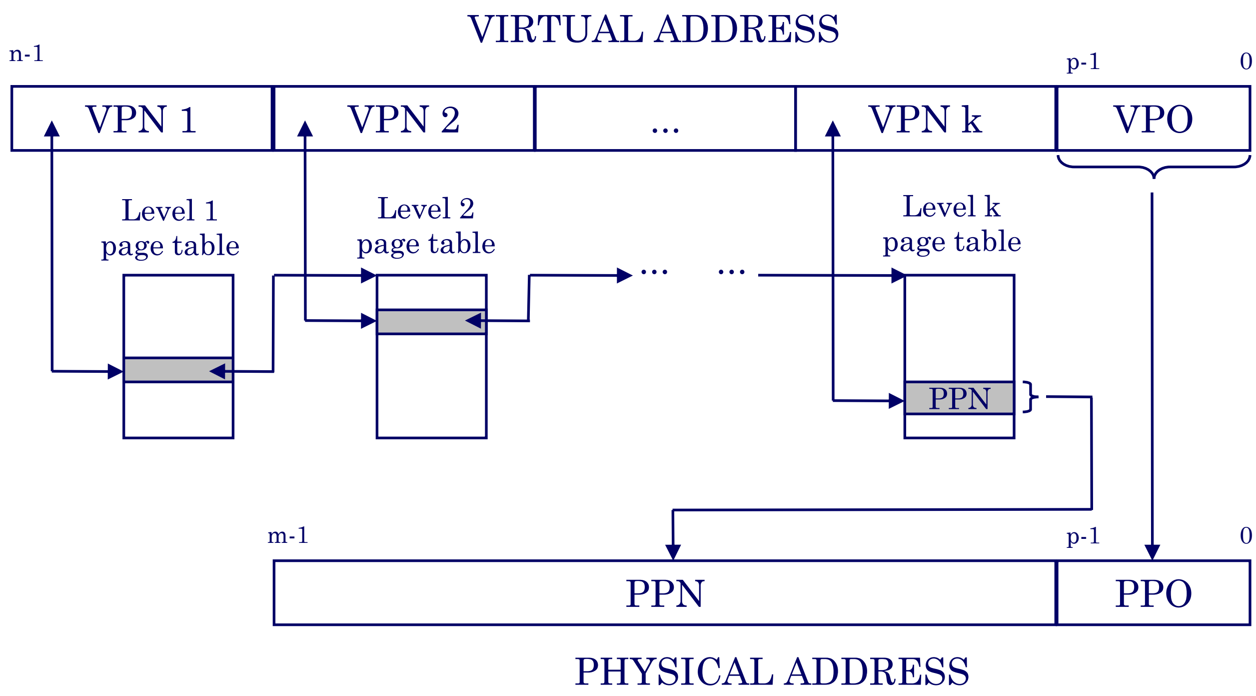


Two-Level Page Table



32 bit addresses, 4KB pages, 4-byte PTEs

Translating w/ a k-level Page Table



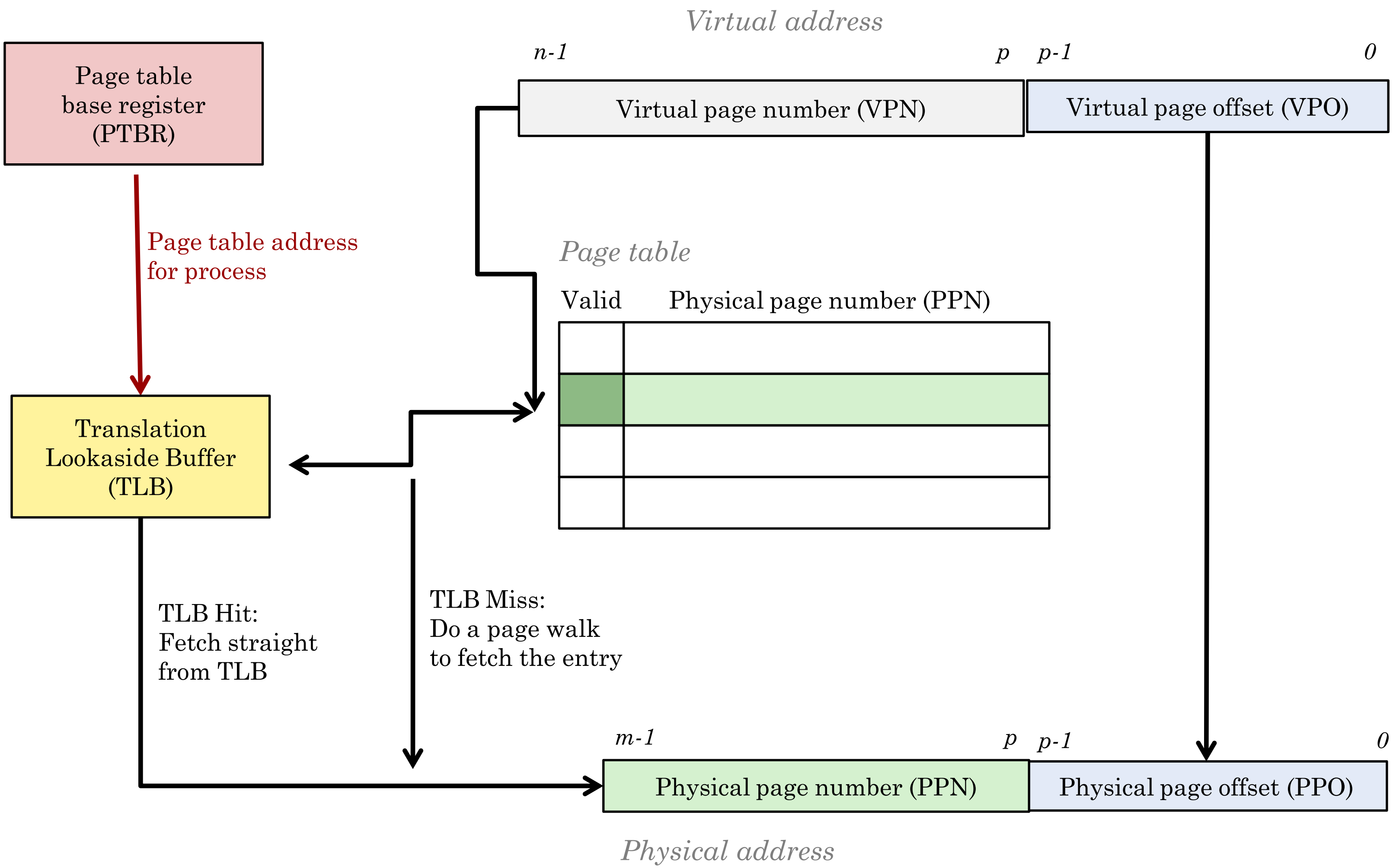
But Memory Accesses are Slow

- At least 2 memory accesses
  - Fetch page-table entry (PTE) from memory
  - Then fetch data from memory
- In x86, 3 memory accesses
  - Page directory, page table, physical memory
- In x86_64, 4 level page-mapping system
- What should we do?
  - Please don't say insert a level of "indirection"

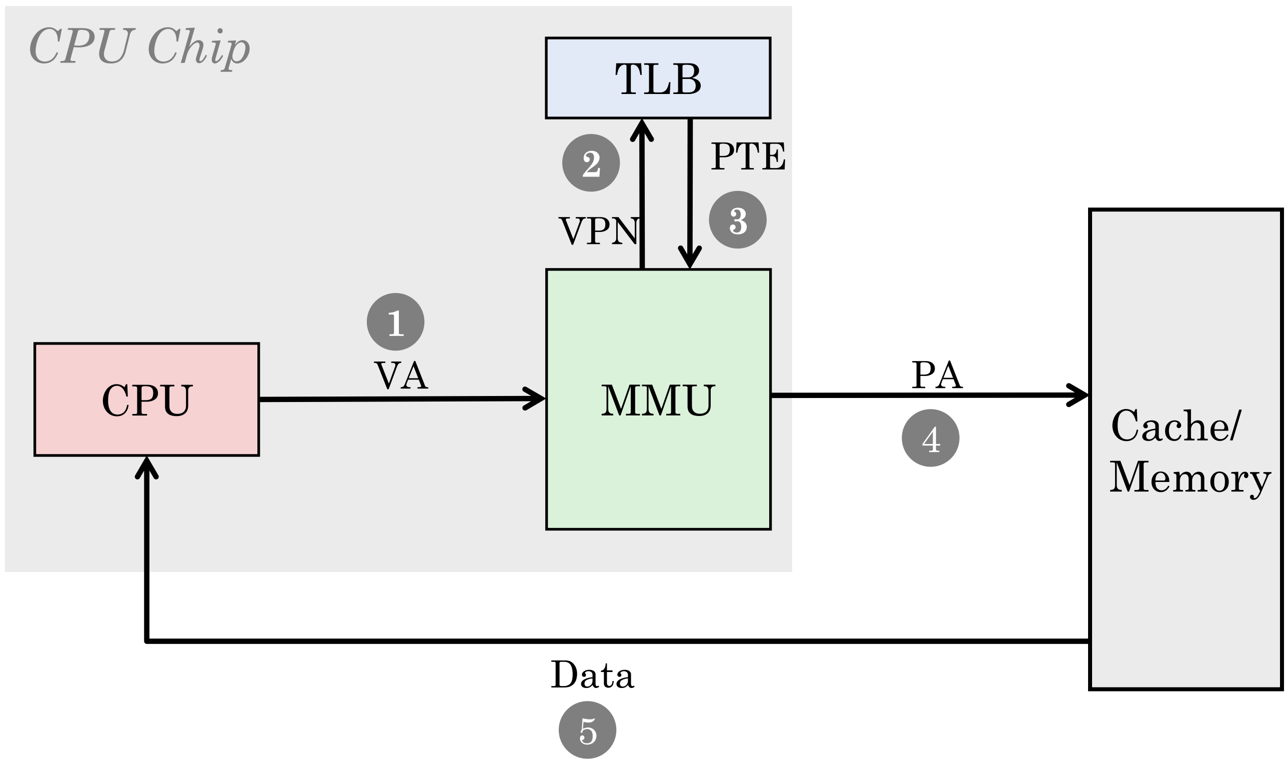
Translation Lookaside Buffer (TLB)

- Super fast hardware cache of PTEs
- Idea: Locality exists between memory accesses
  - Typically access nearby memory
  - Usually on the same page as current data
    - Arrays with loops
    - Program instructions

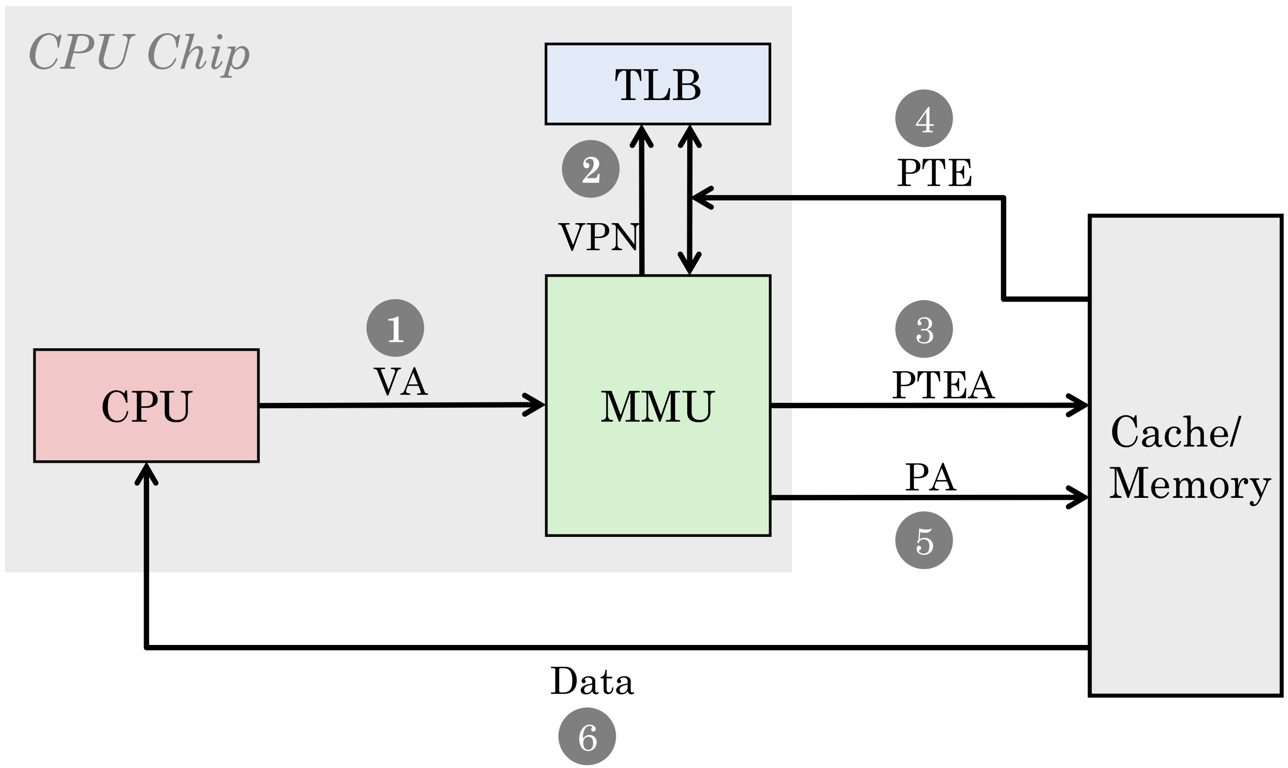
VM: Translations w/ TLB and Tables



Overview of a TLB Hit



Overview of a TLB Miss



Words and Symbols

- Basic Parameters
  - $N = 2^n$: Number of addresses in virtual address space
  - $M = 2^m$: Number of addresses in physical address space
  - $P = 2^p$: Page size (bytes)
- Components of the virtual address (VA)
  - TLBI: TLB index
  - TLBT: TLB tag
  - VPO: Virtual page offset
  - VPN: Virtual page number
- Components of a Physical Address
  - PPO: Physical page offset (same as VPO)
  - PPN: Physical page number
  - CO: Byte offset within cache line
  - CI: Cache index
  - CT: Cache tag

Extra Words

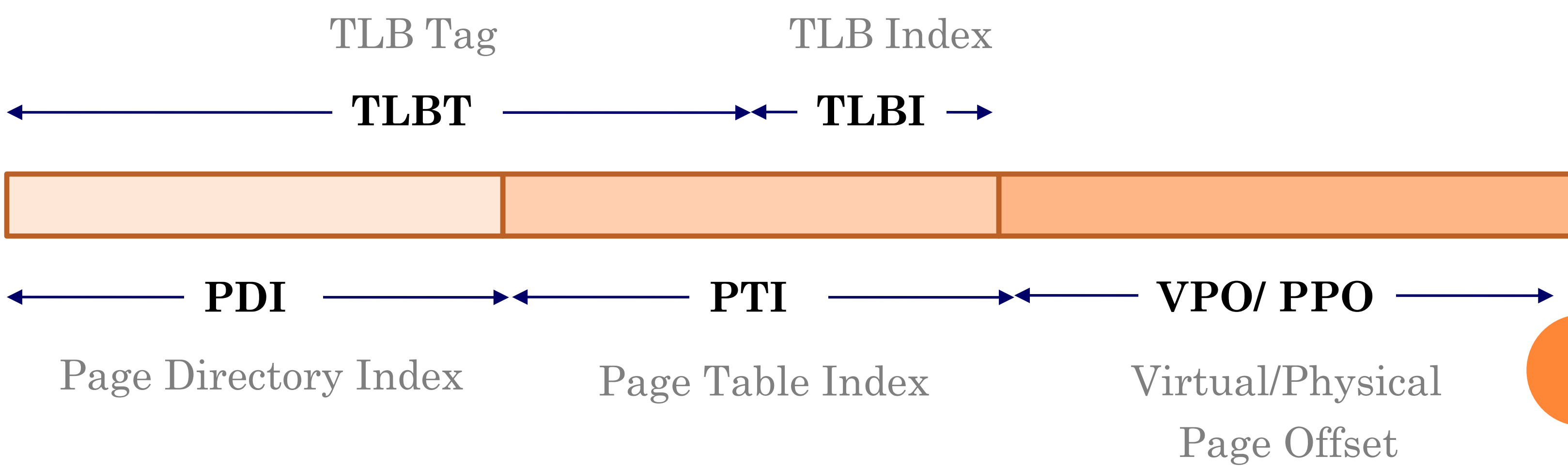
- PDE: Page Directory Entry
  - Entry in first level page table in a 2-level structure
- PTE: Page Table Entry
  - Entry in a page table
- PTI/PDI: Page table/ Page Directory index
  - The index into the page table for an entry
- Page Aligned
  - Address is a multiple of the page size
    - Page directory/ Page Table Base addresses are page aligned

Equations

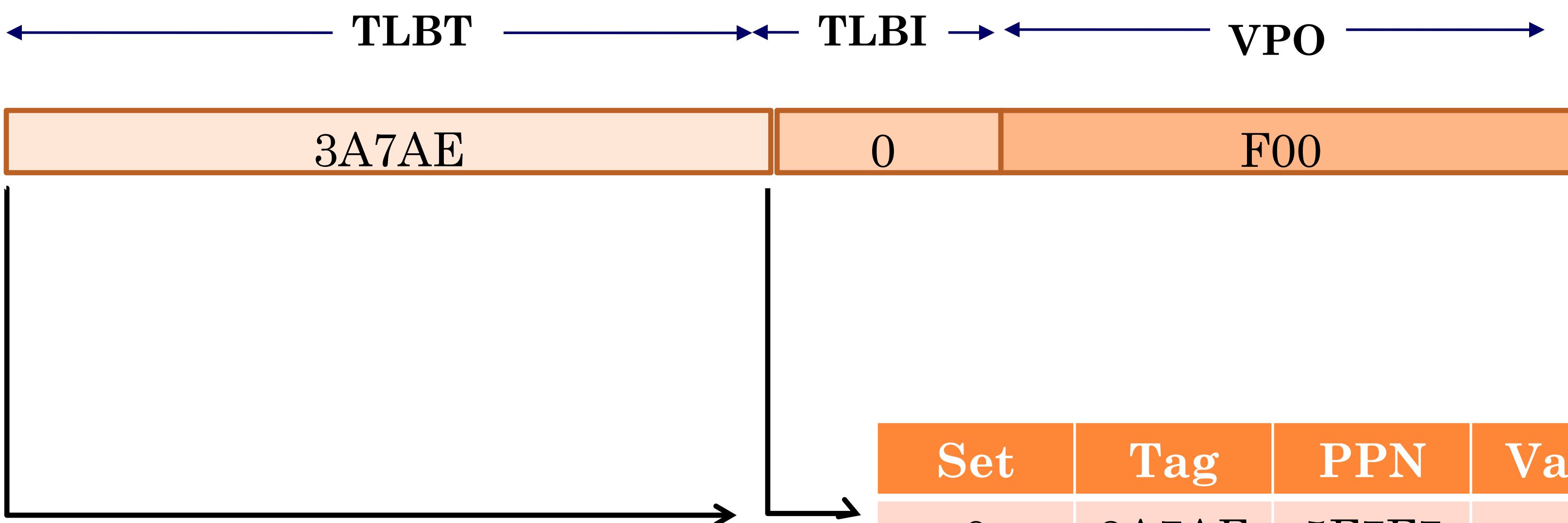
- # virtual address bits = log2(virtual address size)
- # physical address bits = log2(physical address size)
- # bits in VPO = log2(page size)
- # bits in VPN = (# VA bits) - log2(page size)
- Depending on the (problem) these two need to be aligned:
  - PDE Address = Base Address + sizeof(PDE)*(PDI)
  - PTE Address = PDE[PDI] + sizeof(PTE)*(PTI)

Tutorial: Virtual Address Translation

- Addressing
  - 32 bit virtual address
  - 32 bit physical address
  - Page size = 4 kb
- Paging
  - 10 bit page directory index
  - 10 bit page table index
  - 12 bit offset
- TLB
  - Direct Mapped
  - 4 entries



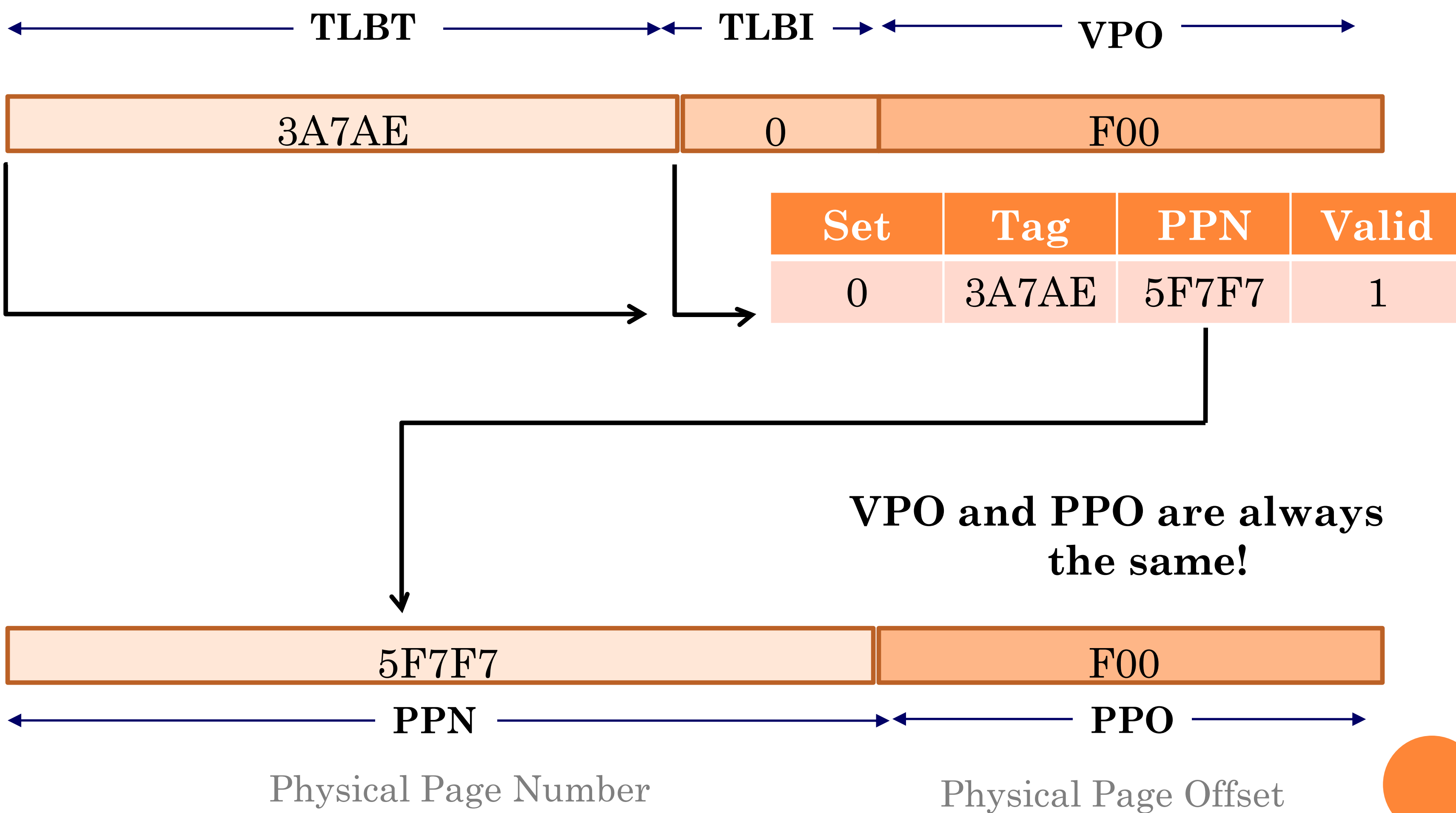
Tutorial: Address Translation Hit



Set	Tag	PPN	Valid
0	3A7AE	5F7F7	1
1			
2			
3			

- Always access TLB first

Tutorial: Address Translation Hit



Set	Tag	PPN	Valid
0	3A7AE	5F7F7	1

VPO and PPO are always the same!

TUTORIAL: ADDRESS TRANSLATION MISS

TLBT	TLBI	VPO
3B8AC	3	BEE

Set	Tag	PPN	Valid
0			
1			
2			
3	3B8AC	DEAD	0

- TLB Miss! Do page walk

Tutorial: Address Translation Miss

3B8	2B3	BEE
PDI	PTI	VPO

Page Directory Index	Page Table Base Address	Valid
..	..	..
0x3B8	0xFAFF8034	1
0x3B9	..	..
…	..	..

Tutorial: Address Translation Miss

3B8	2B3	BEE
PDI	**PTI**	**VPO**

PDI	PTA	Valid
..	..	..
0x3B8	0xFAFF8035	1
0x3B9	..	..
…	..	..

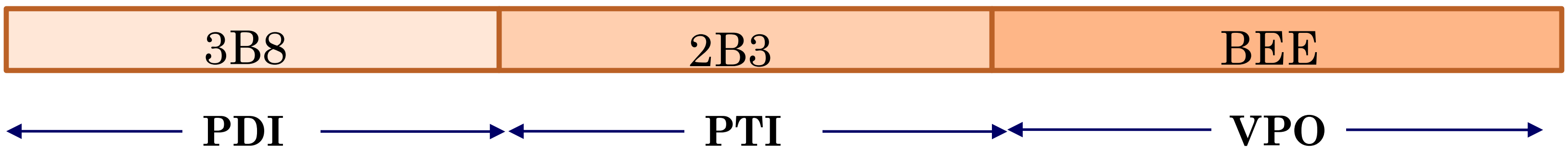
0xFAFF8000

PTI	PPN	Valid
..	..	..
…	…	…
…	..	..
…	..	..

0xFAFF9000

PTI	PPN	Valid
..	..	..
…	…	…
…	..	..
…	..	..

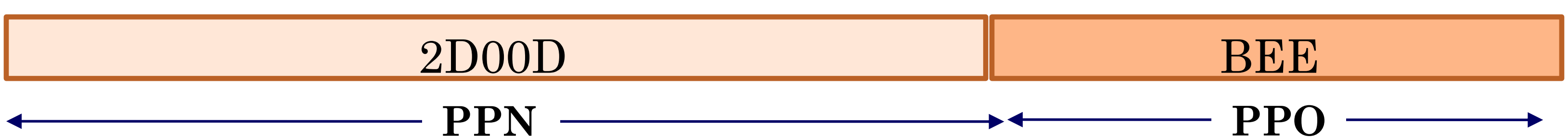
Tutorial: Address Translation Miss



Page Table Index	PPN	Valid
..	..	…
0x2B2	…	…
0x2B3	0x2D00D	1
…	…	…

TUTORIAL: ADDRESS TRANSLATION MISS

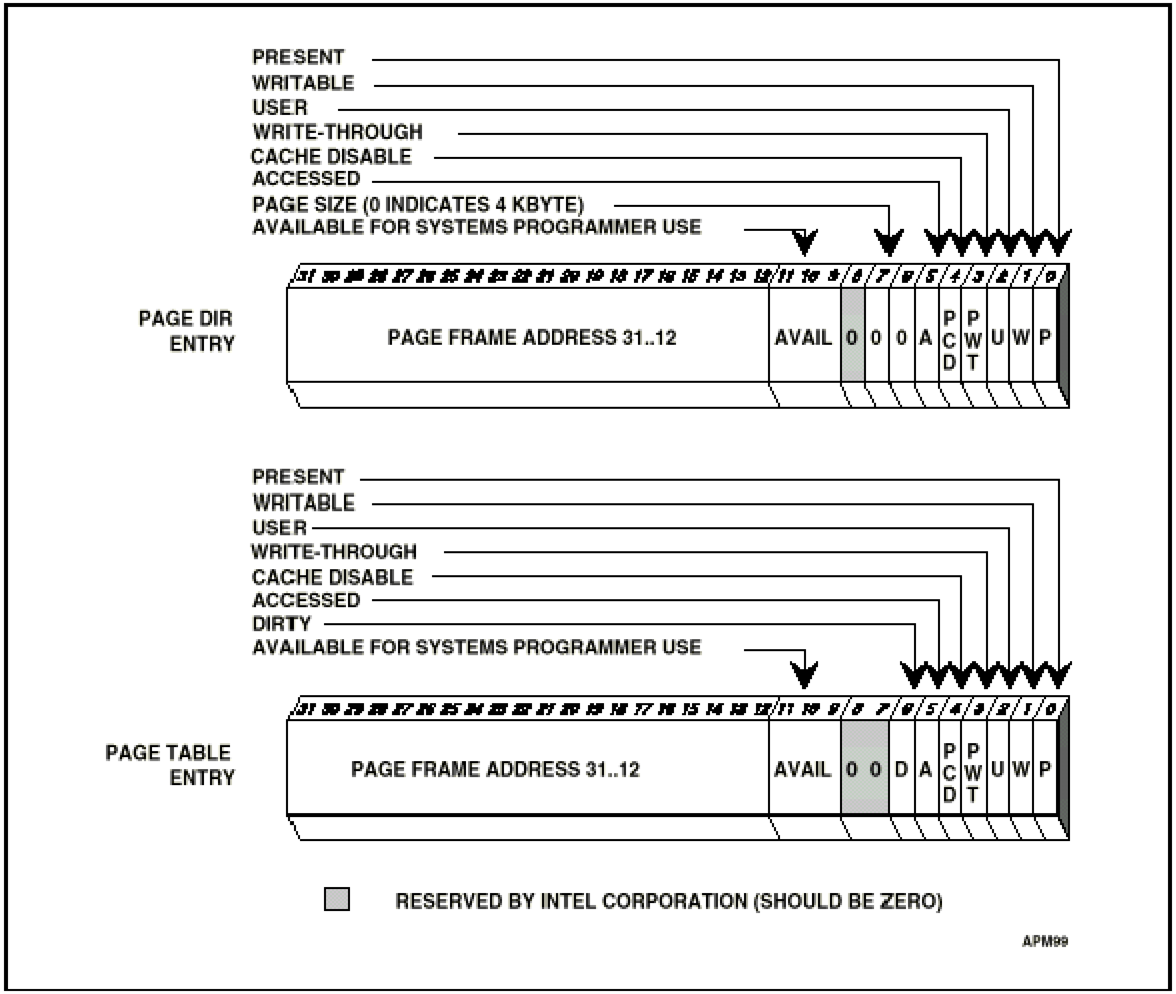
PTI	PPN	Valid
0x2B3	0x2D00D	1



Physical Page Number

Physical Page Offset

Every Bit Counts



Translation Macro Exercise

- 32 bit address: 10 bit VPN1, 10 bit VPN2, 12 bit VPO
- 4KB pages
- Define the following function like macros:
  - Page align

```
#define PAGE_ALIGN(v_addr) _______________________________
```

  - Gets VPN1/VPN2 as unsigned int from virtual address

```
#define VPN1(v_addr) _____________________________________
#define VPN2(v_addr) _____________________________________
```

  - Gets VPO as unsigned int from virtual address

```
#define VPO(v_addr) ______________________________________
```

  - Calculates the address of the page directory index

```
#define PDEA(pd_addr, v_addr) ____________________________
```

  - Calculate address of page table entry

```
#define PTEA(pd_addr, v_addr)_____________________________
```

  - Calculate physical address

```
#define PA(pd_addr, v_addr) ______________________________
```

Translation Macro Solution

- 32 bit address: 10 bit VPN1, 10 bit VPN2, 12 bit VPO
- 4KB pages
- Define the following function like macros:
  - Page align

```
#define PAGE_ALIGN(v_addr) ((unsigned int) v_addr & ~0xfff)
```

  - Gets VPN1/VPN2 as unsigned int from virtual address

```
#define VPN1(v_addr) ((unsigned int) (((v_addr)>>22)&0x3ff))
#define VPN2(v_addr) ((unsigned int) (((v_addr)>>12)&0x3ff))
```

  - Gets VPO as unsigned int from virtual address

```
#define VPO(v_addr) ((unsigned int) ((v_addr)&0xfff))
```

  - Calculates the address of the page directory index

```
#define PDEA(pd_addr, v_addr) (((void **)pd_addr)+VPN1(v_addr))
```

  - Calculate address of page table entry

```
#define PTEA(pd_addr, v_addr)
     (((void **)PAGE_ALIGN(*PDEA(pd_addr,v_addr)))+VPN2(v_addr))
```

  - Calculate physical address

```
#define PA(pd_addr, v_addr)
     (((PAGE_ALIGN(*PTEA(pd_addr,v_addr)))) | VPO(v_addr))
```

Extra Stuff

- For next week, or for your enjoyment

All the C!

- "Saving you from malloc misery…"
- Basics
- Useful C Stuff
- Debugging
- Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language, Second Edition,* Prentice Hall, 1988

C and Pointer Basics

- Statically allocated arrays:
  - `int prices[100];`
  - Get rid of magic numbers:
    - `int prices[NUMITEMS];`
- Dynamically allocated arrays:
  - `int *prices2 = (int *) malloc(sizeof(int) * var);`
- Which is valid:
  - `prices2 = prices;`
  - `prices = prices2;`
- The & operator:
  - `&prices[1]` is the same as `prices+1`
- Function Pointer:
  - `int (*fun)();`
  - Pointer to function returning int

Peeling the Onion (K&R p.101)

- `char **argv`
  - argv: pointer to a pointer to a char
- `int (*daytab)[13]`
  - daytab: pointer to array[13] of int
- `int *daytab[13]`
  - daytab: array[13] of pointer to int
- `char (*(*x())[])()`
  - x: function returning pointer to array[] of pointer to function returning char
- `char (*(*x[3])())[5]`
  - x: array[3] of pointer to function returning pointer to array[5] of char
- Takeaway
  - There is an algorithm to decode this (see K&R p. 101)
  - Always use parenthesis!!

Why Typedefs?

- For convenience and readable code
- Example:
  - ```
    typedef struct
    {
       int x;
       int y;
    } point;
    ```
- Function Pointer example:
  - `typedef int(*pt2Func)(int, int);`
  - `pt2Func` is a pointer to a function that takes 2 int arguments and returns an int
    ```

Macros are Cool

- C Preprocessor looks at macros during the preprocessing step of compilation
- Use `#define` to avoid magic numbers:
  - `#define TRIALS 100`
- Function like macros – short and heavily used code snippets
  - `#define GET_BYTE_ONE(x) ((x) & 0xff)`
  - `#define GET_BYTE_TWO(x) ( ( (x) >> 8) & 0xff)`
- Also look at inline functions (example prototype):
  - `inline int max(int a, int b)`
  - Requests compiler to insert assembly of max wherever a call to max is made
- Both useful for malloc lab

Debugging – Favorite Methods

- Using the `DEBUG` flag:
  - ```
    #define DEBUG

    #ifdef DEBUG
    #define dbg_printf(...) printf(__VA_ARGS__)
    #else
    #define dbg_printf(...)
    #endif
    ```
- Compiling (if you want to debug):
  - `gcc -DDEBUG foo.c -o foo`
- Using `assert`
  - `assert(posvar > 0);`
  - `man 3 assert`
- Compiling (if you want to turn off asserts):
  - `gcc -DNDEBUG foo.c -o foo`
    ```

Little Things

- Usage messages
  - Putting this in is a good habit – allows you to add features while keeping the user up to date
  - `man -h`
- `fopen/fclose`
  - Always error check!
- malloc()
  - Error check
  - Free everything you allocate
- Global variables
  - Namespace pollution
  - If you must, make them private:
    - `static int foo;`

Questions and References Slide

- Rubber Duck 1
- Rubber Duck Debugging on Wiki
- Florentijn Hofman's Duck
- Indirection on Wiki
- Pictures stolen from lecture slides
- Stole from 15-410 Virtual Memory Slides
  - Lectures reside here
  - BTW, Prof. Eckhardt is super cool
- IA32 Page Directory/Table Bits