Operating Systems Design (CS 423)

Elsa L Gunter

2112 SC, UIUC

http://www.cs.illinois.edu/class/cs423/

Based on slides by Roy Campbell, Sam King, and Andrew S Tanenbaum

Illusions Provided by Address Apace

- Address independence
  - Same address in different processes not conflicting with each other
  - Eg Same address for stack
- Protection
  - One process cannot access the data of another
  - Secret data, protected code
- Virtual memory
  - 64 bit address space, memory many 4 G

Memory Allocation Provides?

- Which of these does Memory Allocation accomplish?
- Address independence?
- Protection?
- Virtual memory?

Memory Allocation Provides?

- Which of these does Memory Allocation accomplish?
- Address independence?
  - No
- Protection?
- Virtual memory?

Memory Allocation Provides?

- Which of these does Memory Allocation accomplish?
- Address independence?
  - No
- Protection?
  - No
- Virtual memory?

Memory Allocation Provides?

- Which of these does Memory Allocation accomplish?
- Address independence?
  - No
- Protection?
  - No
- Virtual memory?
  - Yes

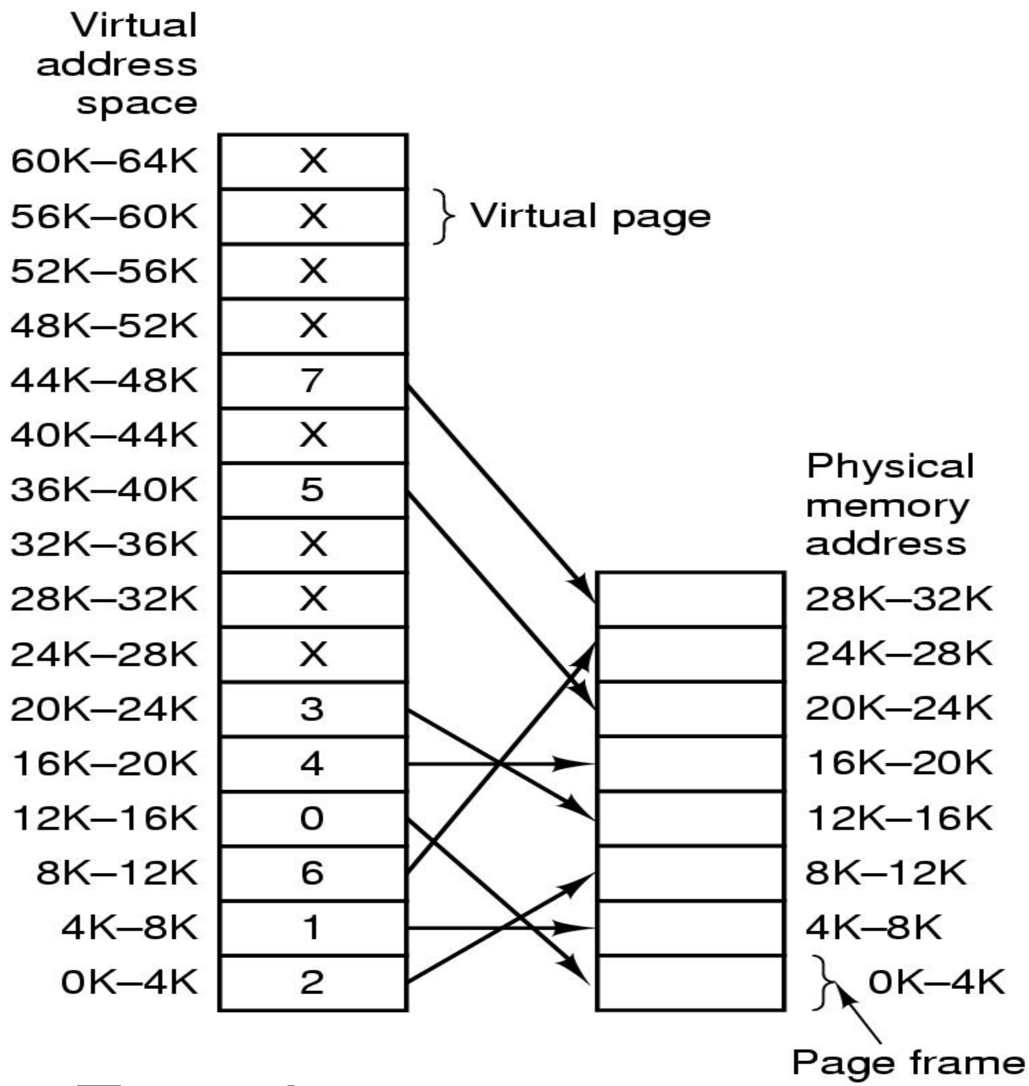
Memory Allocation Provides?

- Which of these does Memory Allocation accomplish?
- Address independence?
  - No
- Protection?
  - No
- Virtual memory?
  - Yes
- We need a new abstraction!

Paging

- Allocate physical memory in terms of fixed-size chunk
  - Fixed unit easier to allocate
  - Any free physical page (page frame) can store any virtual page
- Virtual address
  - Virtual page # (high bits of address)
  - Offset (low bits of address, e.g., bits 11-0 for 4k page)
- Each process has own pages, page table

Translation Table: Page Table



- Fig 3.9 from Tanenbaum

Translation Process

```
If(virtual page is invalid or non-
  resident or protected) {
   trap to OS fault handler
} else {
   physical page # =
   pageTable[virtpage#].physPageNum
}
```

Translation Process

- What must change on a context switch?

Translation Process

- What must change on a context switch?
  - Page table must be replaced

- Each virtual page can be in physical memory or swapped out to disk (called paged)

Resident Pages

- How does the processor know that a virtual page is not in memory?

Resident Pages

- How does the processor know that a virtual page is not in memory?
  - A bit in the page table entry

- Resident means virtual page is in memory

- NOT an error for program to access non-resident page

Valid Pages Accesses

- Pages can have different protections
  - Read, write, execute

- Valid means that a virtual page is legal for the program to access
  - E.g. page not part of the address space is invalid page
- IS an error to try to access an invalid page

Valid vs Resident

- Who makes a page resident / non-resident?
- Who makes a virtual page valid / invalid?
- Why would a process want one if its virtual pages to be invalid?

Valid vs Resident

- Who makes a page resident / non-resident?
  - OS
- Who makes a virtual page valid / invalid?
  - (user) program
- Why would a process want one of its virtual pages to be invalid?
  - Security, and general protection from self

Gen Structure of Page Table Entry

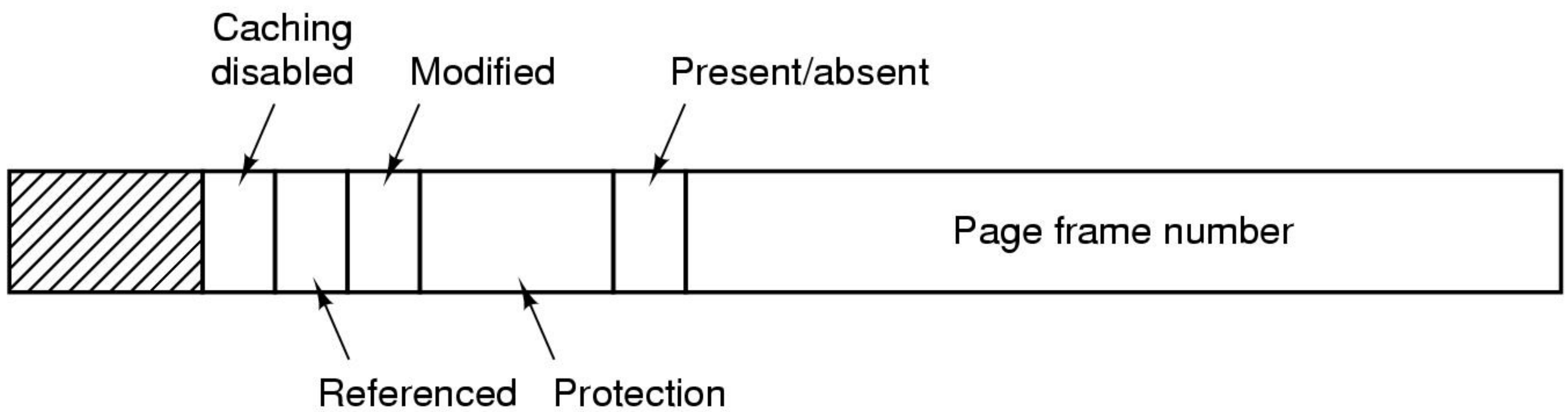


Figure 3-11. A typical page table entry.

Multi-level Translation

- Standard page table is a simple array
  - Might take huge amounts of memory for sparse address space (think 64 bit machines)
  - Multi-level translation changes this to a tree
  - Point: only some branches in memory
- Ex: two-level page table on 32 bit machine
  - Level 1 – virtual address bits 31-22 index
  - Level 2 – virtual address bits 21-12 index
  - Offset: bits 11-0 (4KB page)

Entry Layouts form Intel Manual

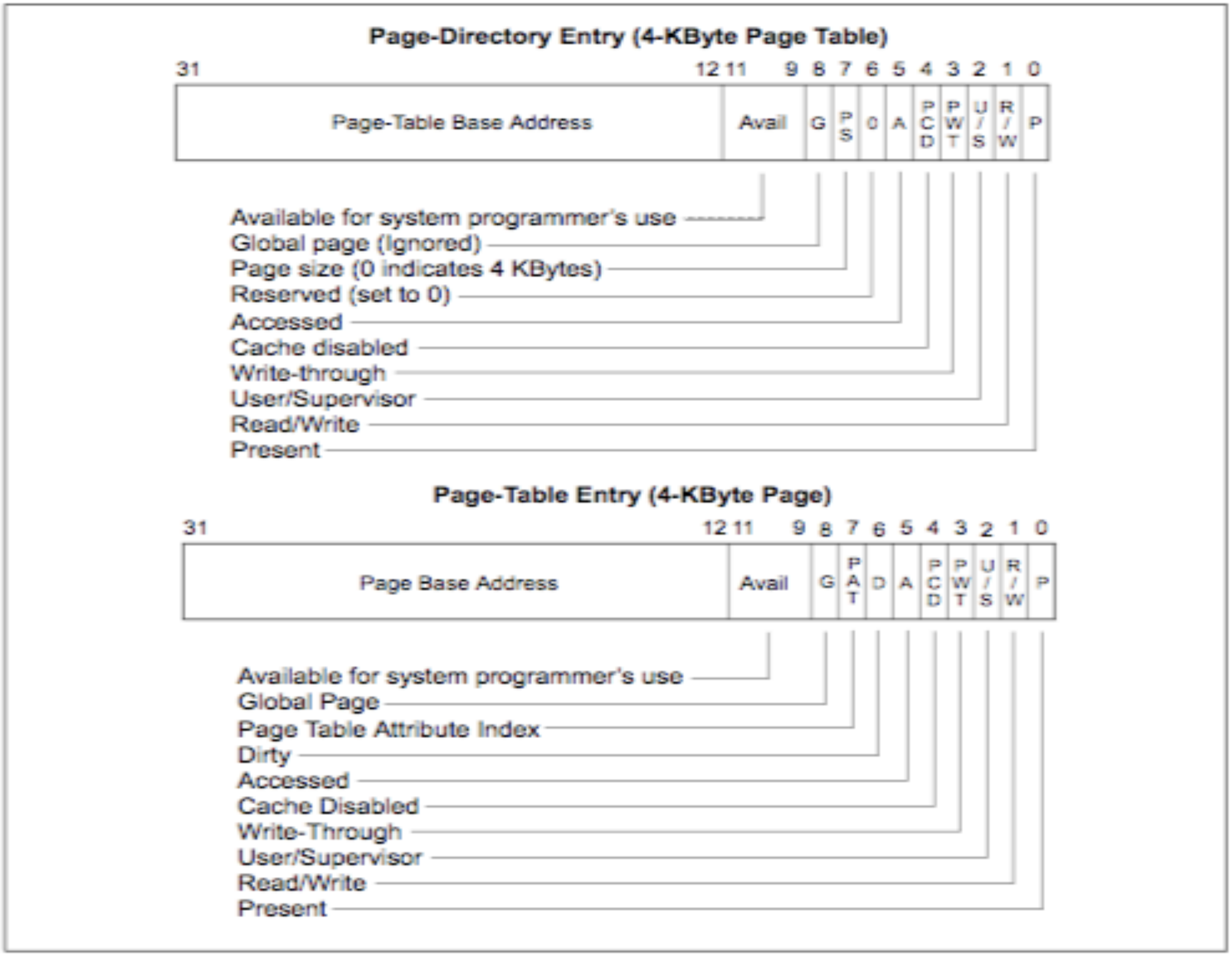
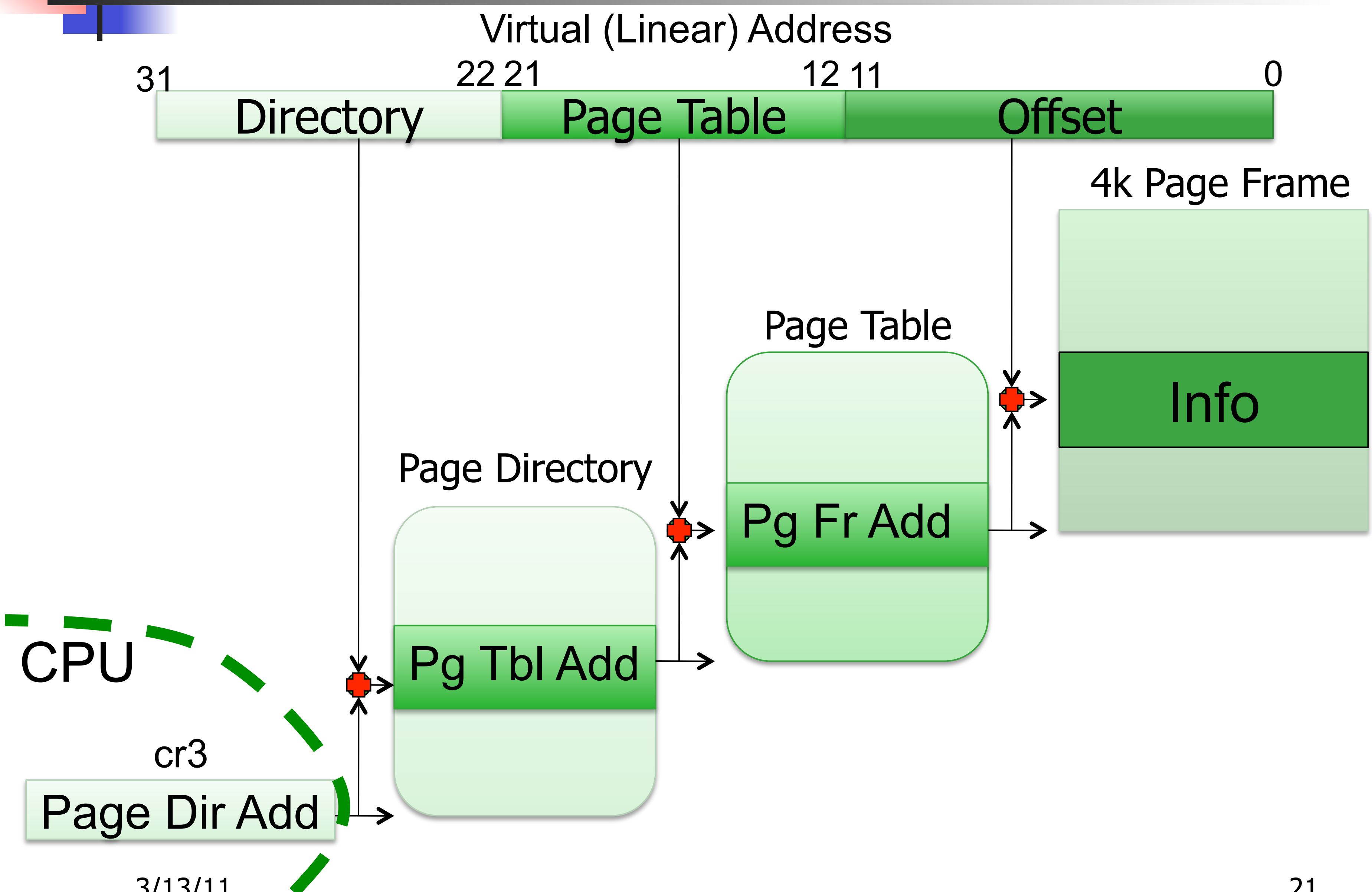
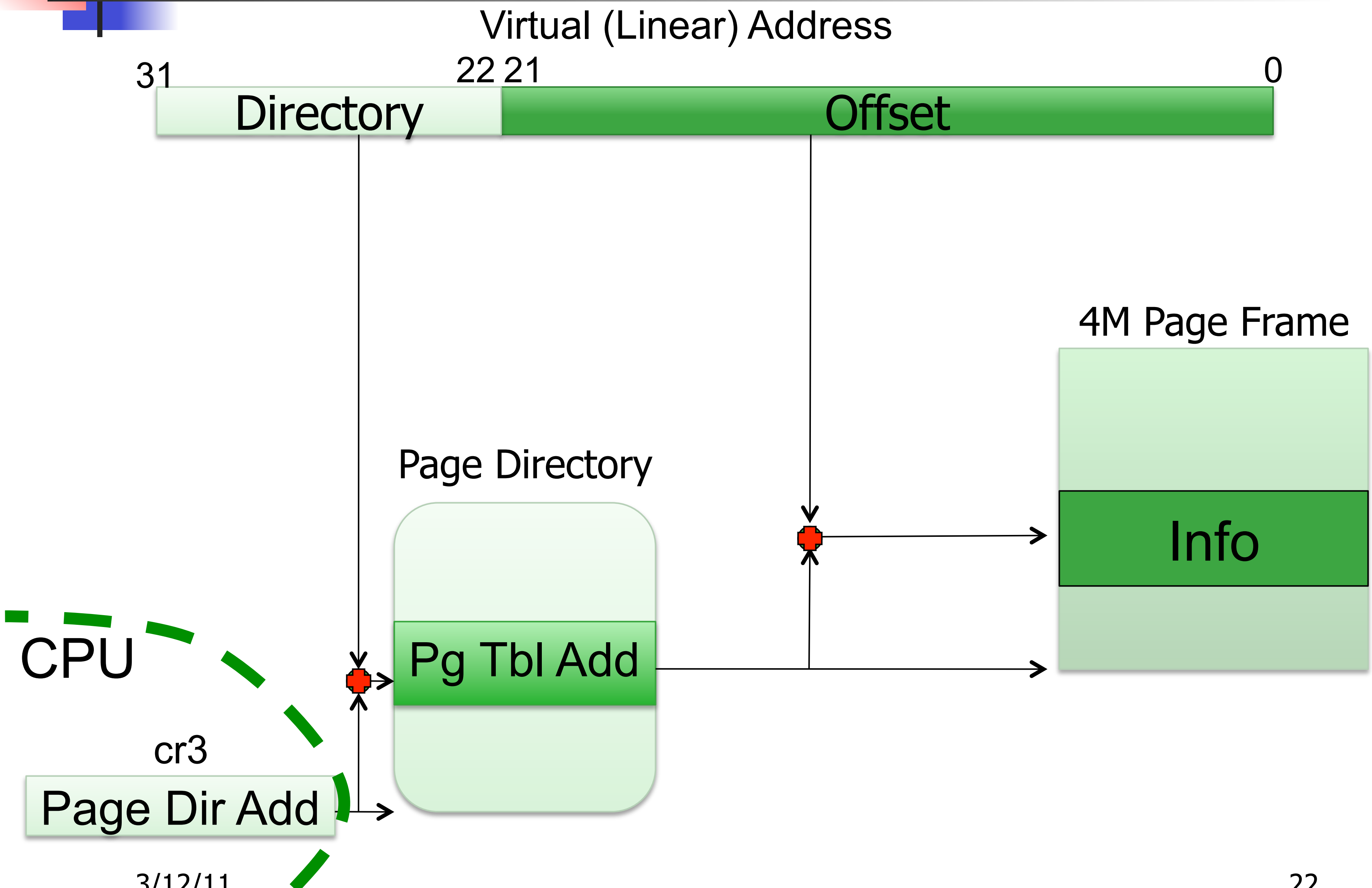


Figure 3-14. Format of Page-Directory and Page-Table Entries for 4-KByte Pages and 32-Bit Physical Addresses

Translating 32 bit Virtual Address



Translating 32 bit Virtual Address



Translating 64 bit Virtual Address