بسم الله الرحمن الرحيم

٤٠

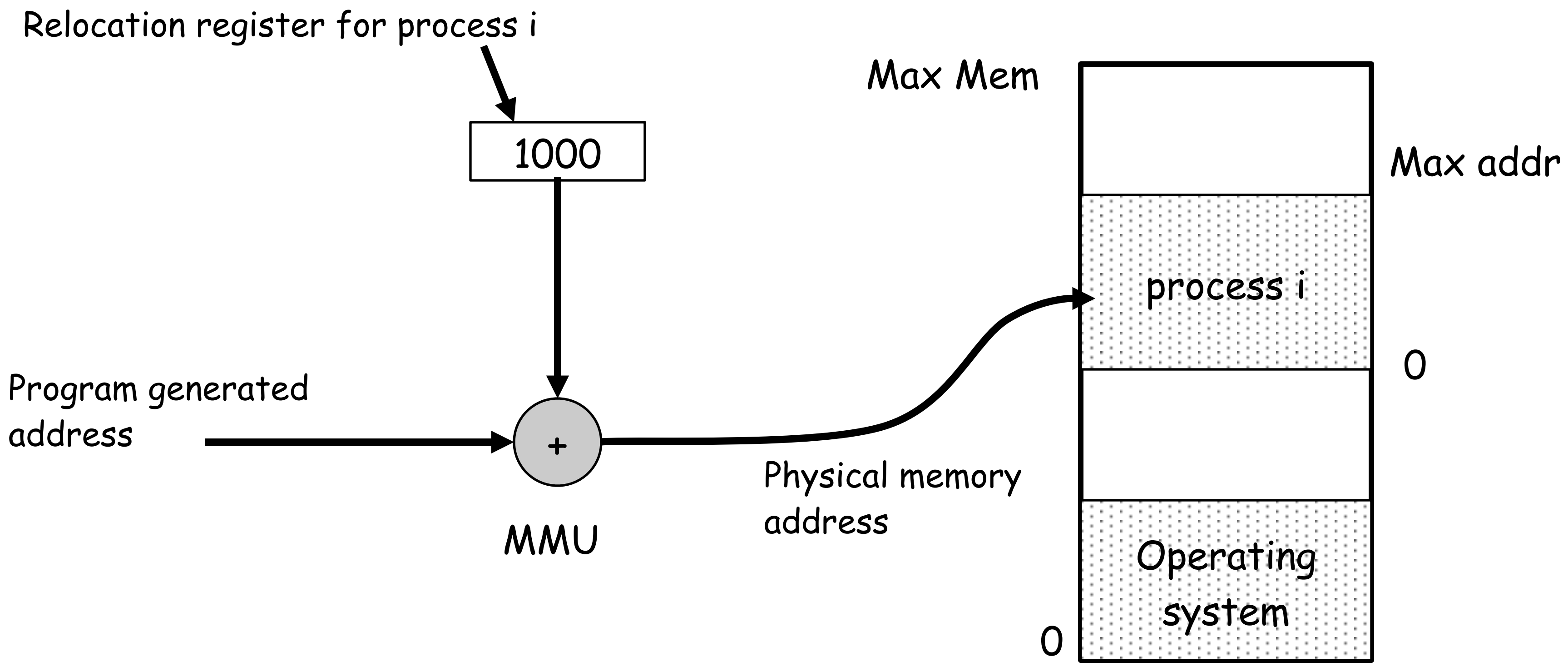
«سیستم عامل»

جلسه ۱۴: مدیریت حافظه

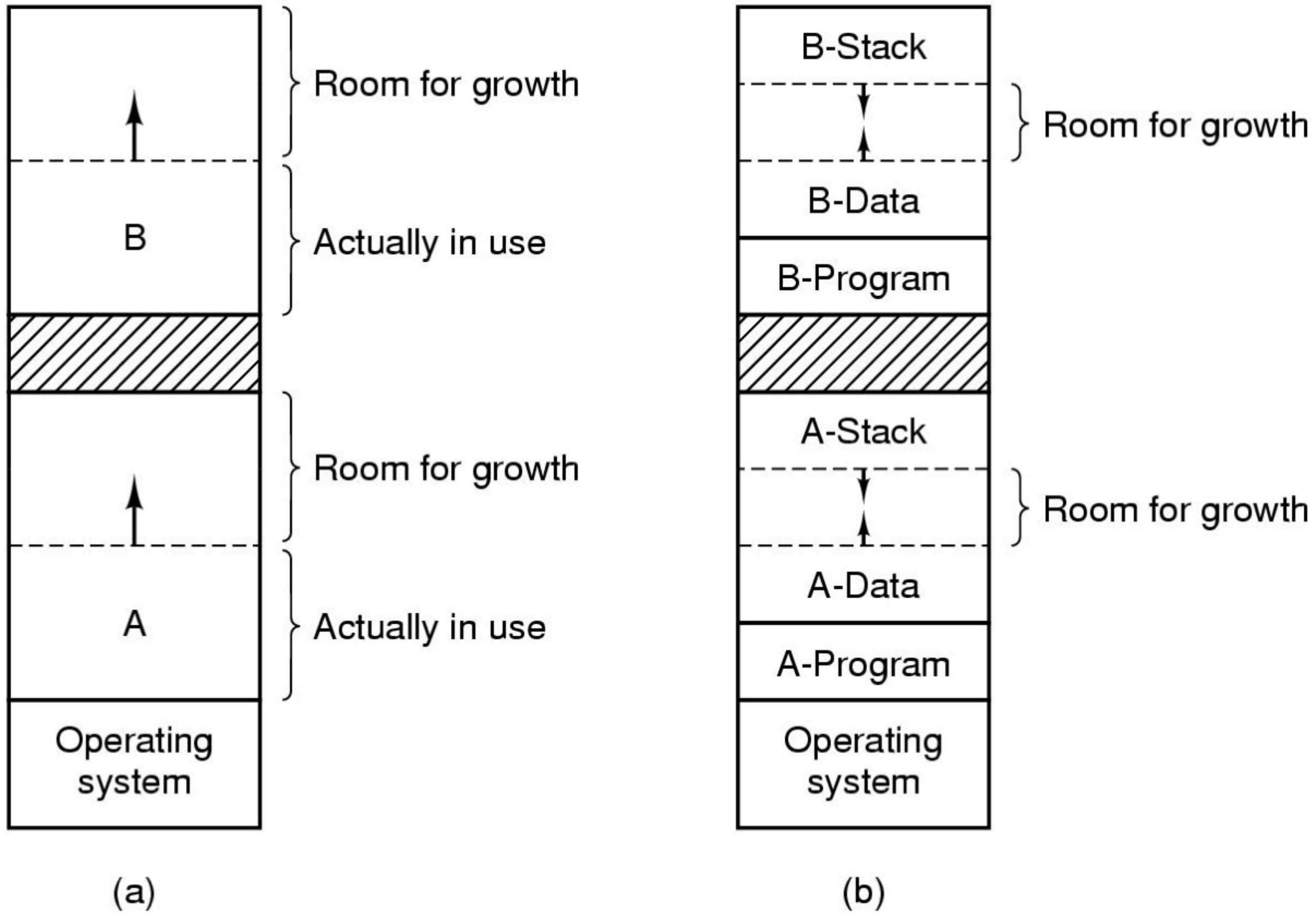
روش ۱: حافظه پیوسته

Dynamic relocation with a base register

- **Memory Management Unit (MMU) - dynamically converts logical addresses into physical address**
- **MMU contains base address register for running process**



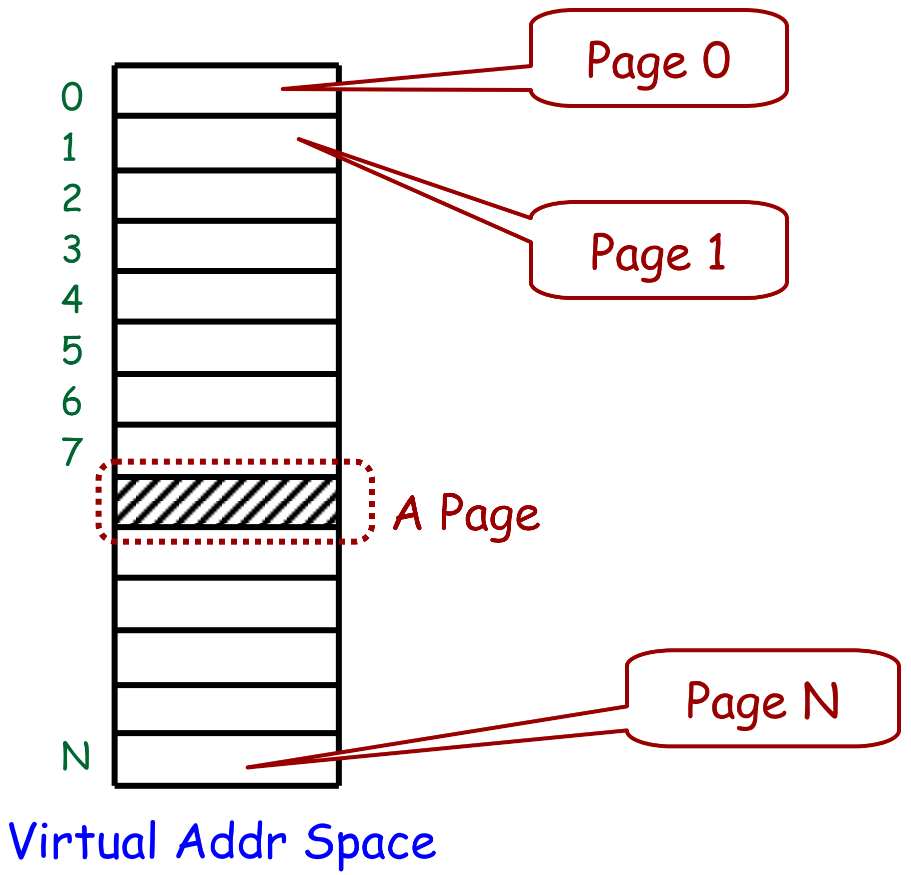
Allocating extra space within partitions



روش ۲: صفحه‌بندی

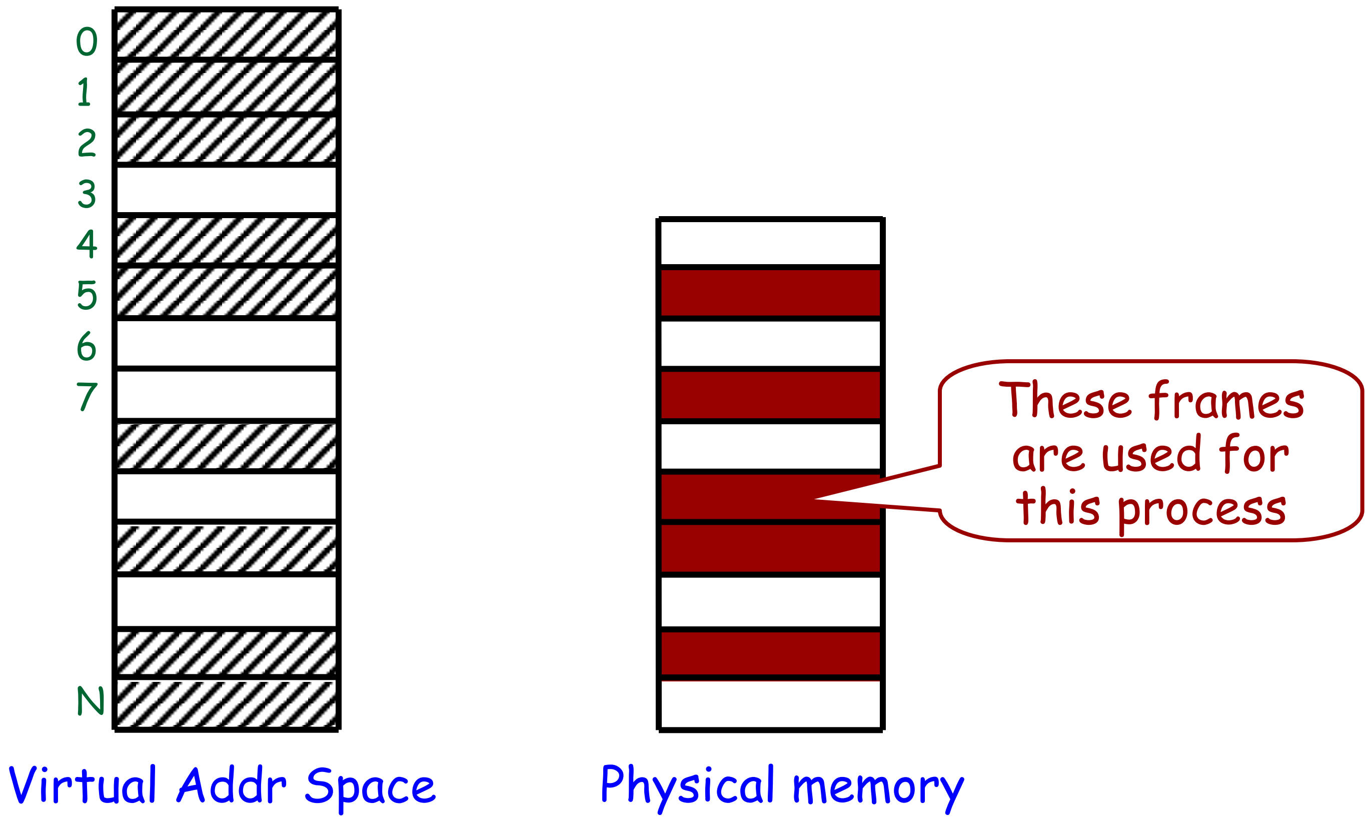
Virtual address spaces

- The address space is divided into "pages"
  - In BLITZ, the page size is 8K



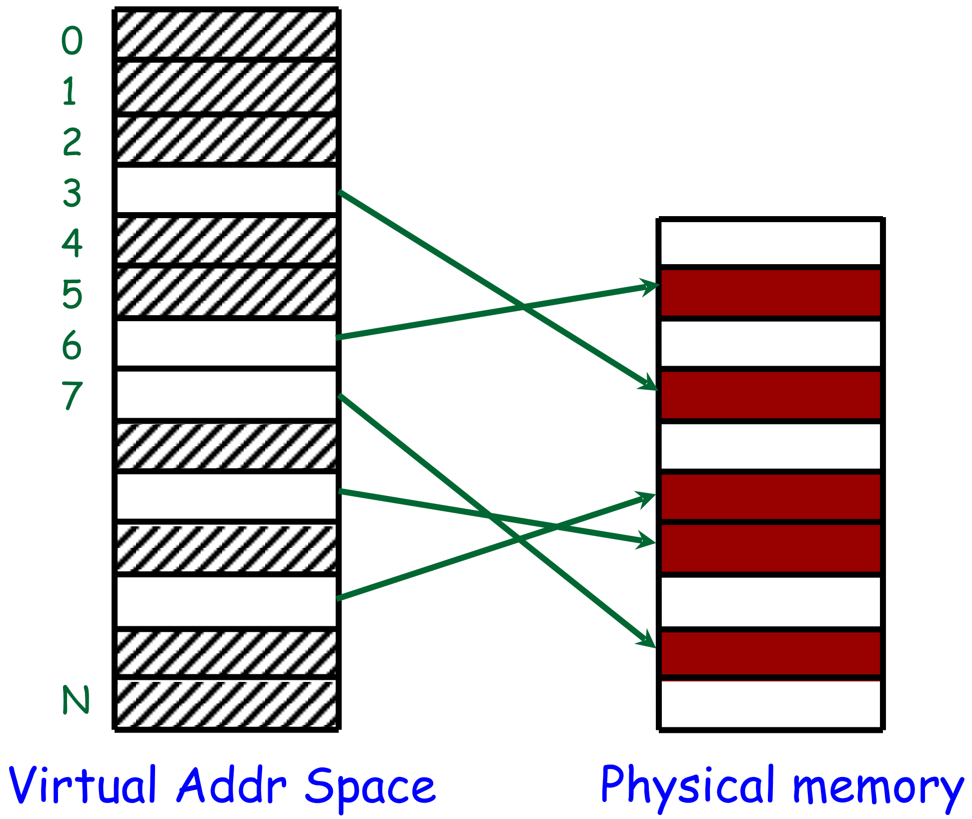
Virtual and physical address spaces

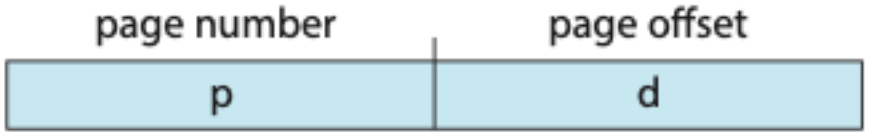
- Some frames are used to hold the pages of this process

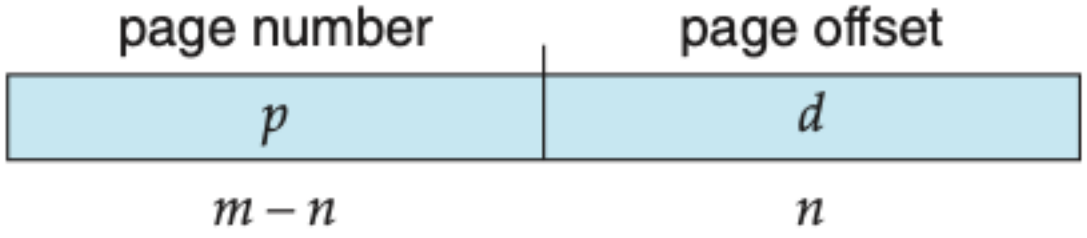


Page tables

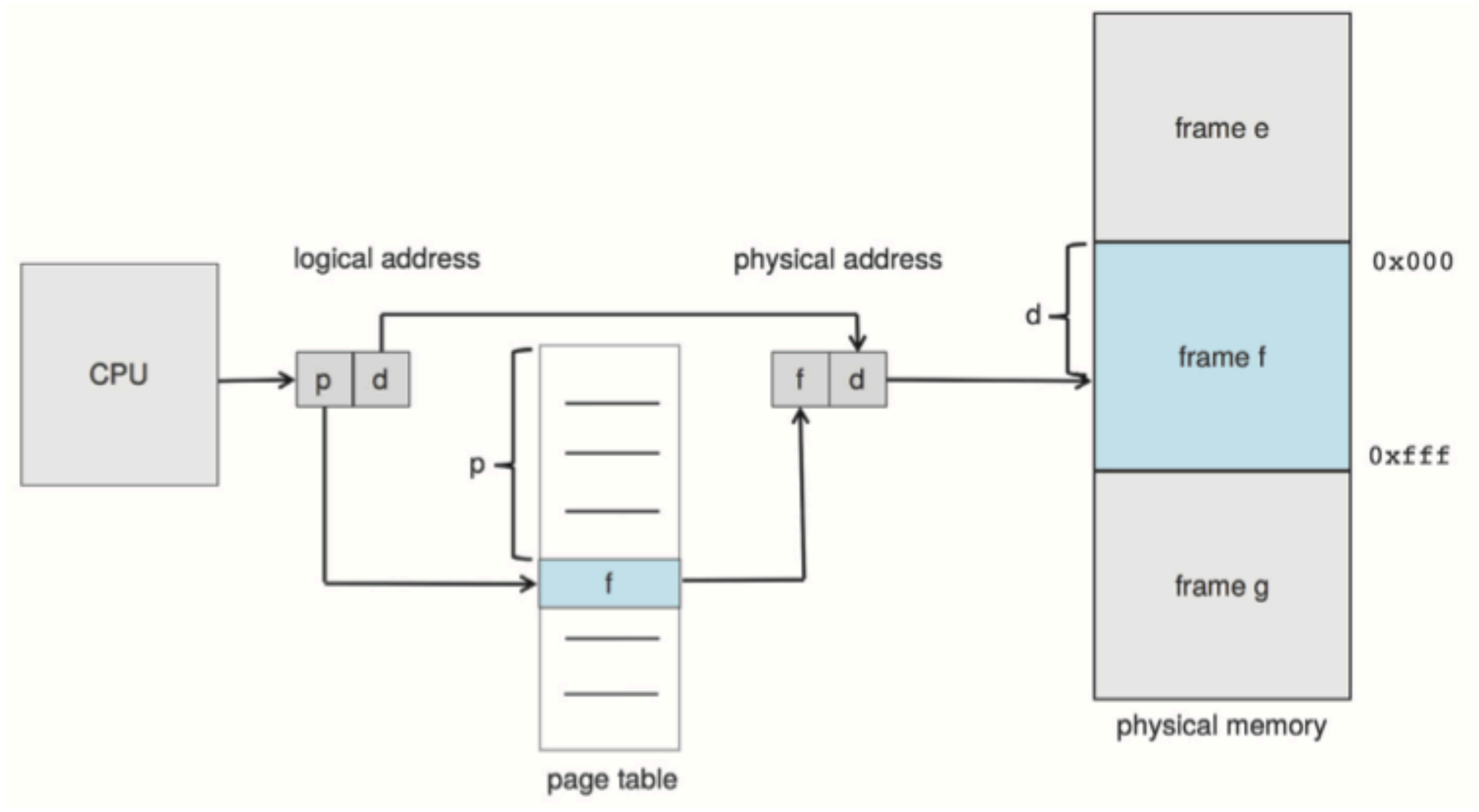
- Address mappings are stored in a page table in memory
- One page table entry per page...
  - Is this page in memory? If so, which frame is it in?




page number
page offset
p
d


page number
page offset
p
d
$m - n$
n

Logical Mapping



Address mappings and translation

- **Address mappings are stored in a page table in memory**
  - Typically one page table for each process

- **Address translation is done by hardware (ie the MMU)**

- **How does the MMU get the address mappings?**
  - Either the MMU holds the entire page table (too expensive)
    - **or it knows where it is in physical memory and goes there for every translation (too slow)**
  - Or the MMU holds a portion of the page table
    - **MMU caches page table entries**
    - **Cache is called a translation look-aside buffer (TLB)**
    - **… and knows how to deal with TLB misses**

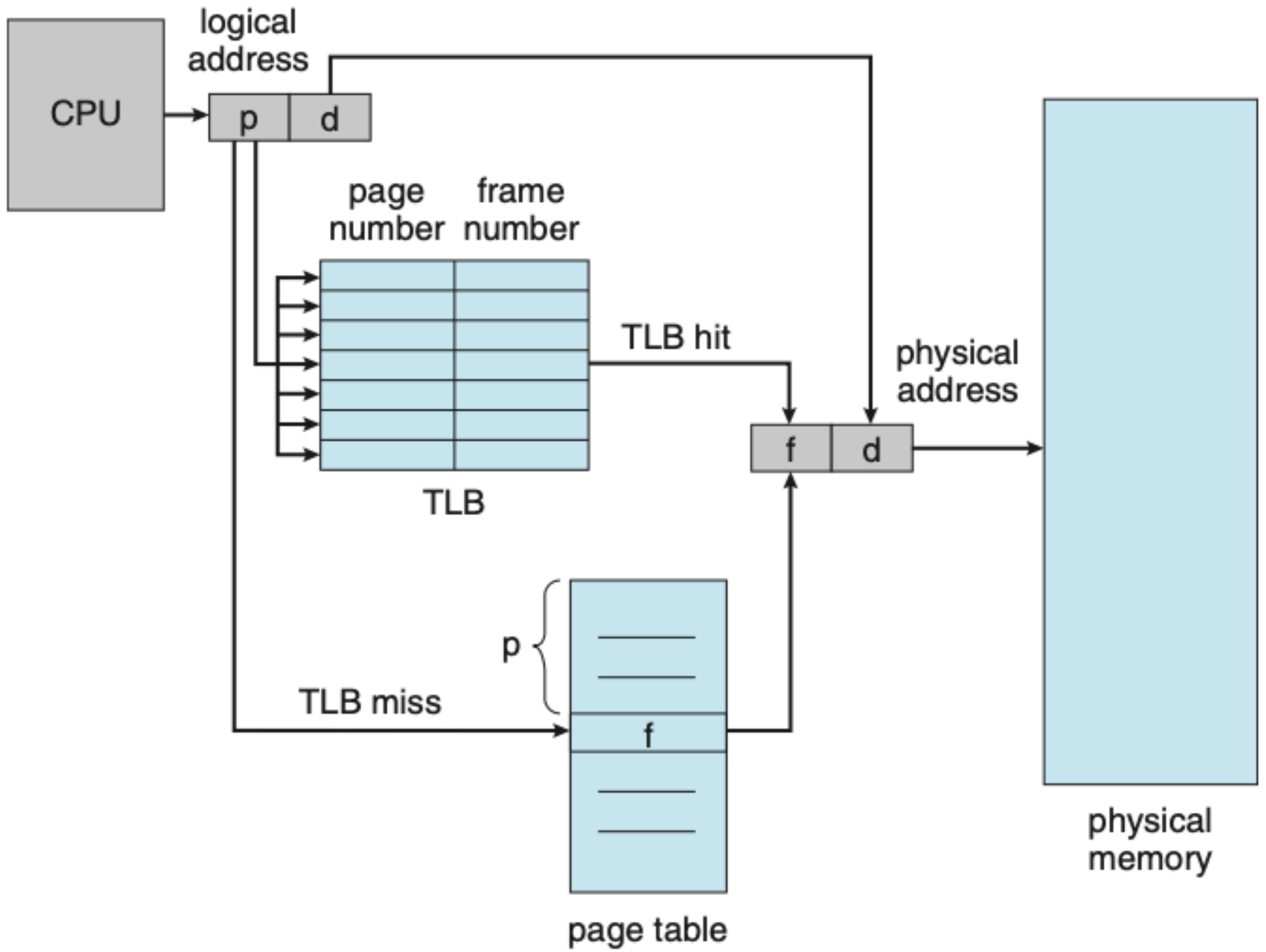


Figure 9.12 Paging hardware with TLB.

Address mappings and translation

- **What if the TLB needs a mapping it doesn't have?**

- **Software managed TLB**
  - it generates a TLB-miss fault which is handled by the operating system (like interrupt or trap handling)
  - The operating system looks in the page tables, gets the mapping from the right entry, and puts it in the TLB

- **Hardware managed TLB**
  - it looks in a pre-specified physical memory location for the appropriate entry in the page table
  - The hardware architecture defines where page tables must be stored in physical memory
    - **OS must load current process page table there on context switch!**

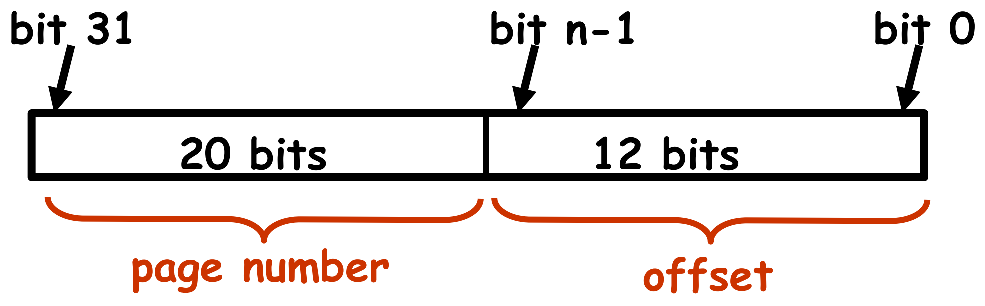
```
In real world #2
----------------------------------------------

From a user process point of view

    +----------------------+ <--- 0xBFFF FFFF (=3GB)
    | environment variable |
    |----------------------| <--- 0xBFFF FD0C
    | stacks (down grow)   |
    |          |           |
    |          v           |
    |----------------------|
    |                      |
    |     // free memory   |
    |                      |
    |                      |
    |----------------------| <--------------------
    |  myprogram.o         |                      |
    |----------------------|                      |
    |  mylib.o             |                      |
    |----------------------|           executable image
    |  myutil.o            |                      |
    |----------------------|                      |
    |  library code (libc) |                      |
    |----------------------| <--------------------
    |                      | 0x8000 0000 (=2GB)
    |                      |
    |  other memory        |
    |                      |
    +----------------------+
```

Virtual addresses

- Virtual memory addresses (what the process uses)
  - Page number plus byte offset in page
  - Low order n bits are the byte offset
  - Remaining high order bits are the page number



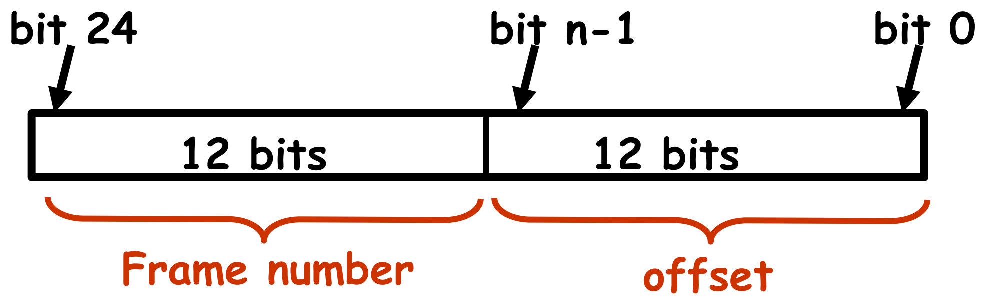
Example: 32 bit virtual address

Page size = 2^{12} = 4KB

Address space size = 2^{32} bytes = 4GB

Physical addresses

- Physical memory addresses (what memory uses)
  - Frame number plus byte offset in frame
  - Low order n bits are the byte offset
  - Remaining high order bits are the frame number



Example: 24 bit physical address

Frame size = 2^{12} = 4KB

Max physical memory size = 2^{24} bytes = 16MB

Address translation

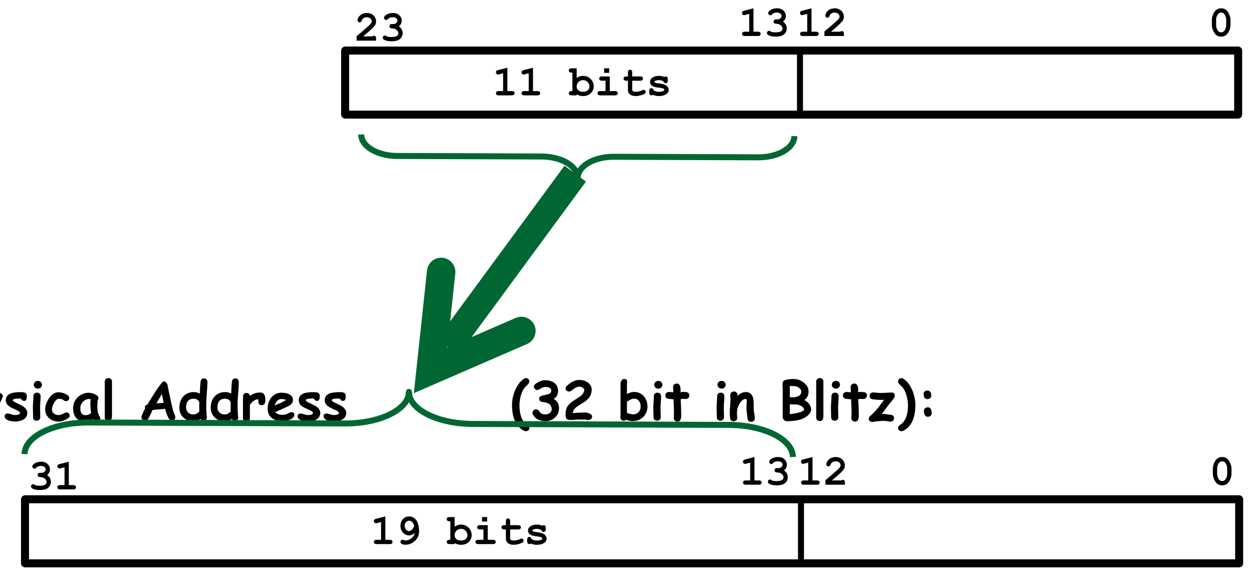
- **Complete set of address mappings for a process are stored in a page table in memory**
  - But accessing the table for every address translation is too expensive
  - So hardware support is used to map page numbers to frame numbers at full CPU speed
    - Memory management unit (MMU) has multiple registers for multiple pages and knows how to access page tables
    - Also called a translation look aside buffer (TLB)
    - Essentially a cache of page table entries

The BLITZ architecture

- The page table mapping:
  - Page --> Frame

- Virtual Address (24 bit in Blitz):

- Physical Address (32 bit in Blitz):



The BLITZ page table

- **An array of "page table entries"**
  - Kept in memory
- **2^{11} pages in a virtual address space**
  - ---> 2K entries in the table
- **Each entry is 4 bytes long**

19 bits	The Frame Number
1 bit	Valid Bit
1 bit	Writable Bit
1 bit	Dirty Bit
1 bit	Referenced Bit
9 bits	Unused (and available for OS algorithms)

The BLITZ page table

- **Two page table related registers in the CPU**
  - Page Table Base Register
  - Page Table Length Register

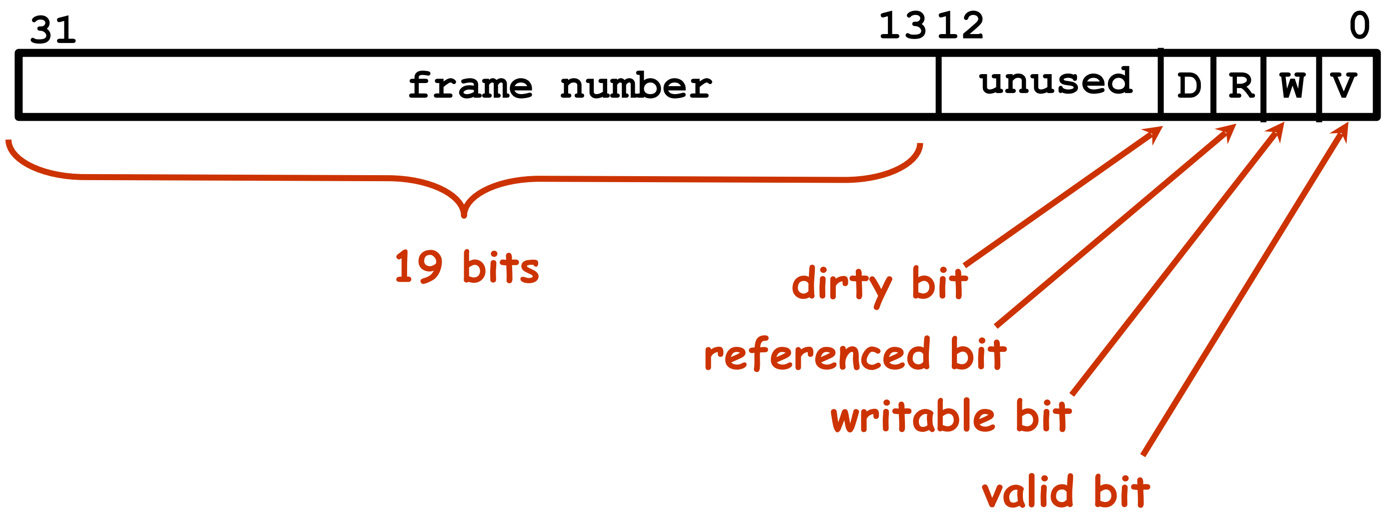
- **These define the page table for the "current" process**
  - Must be saved and restored on process context switch

- **Bits in the CPU "status register"**

  "System Mode"
  "Interrupts Enabled"
  "Paging Enabled"
  1 = Perform page table translation for every memory access
  0 = Do not do translation

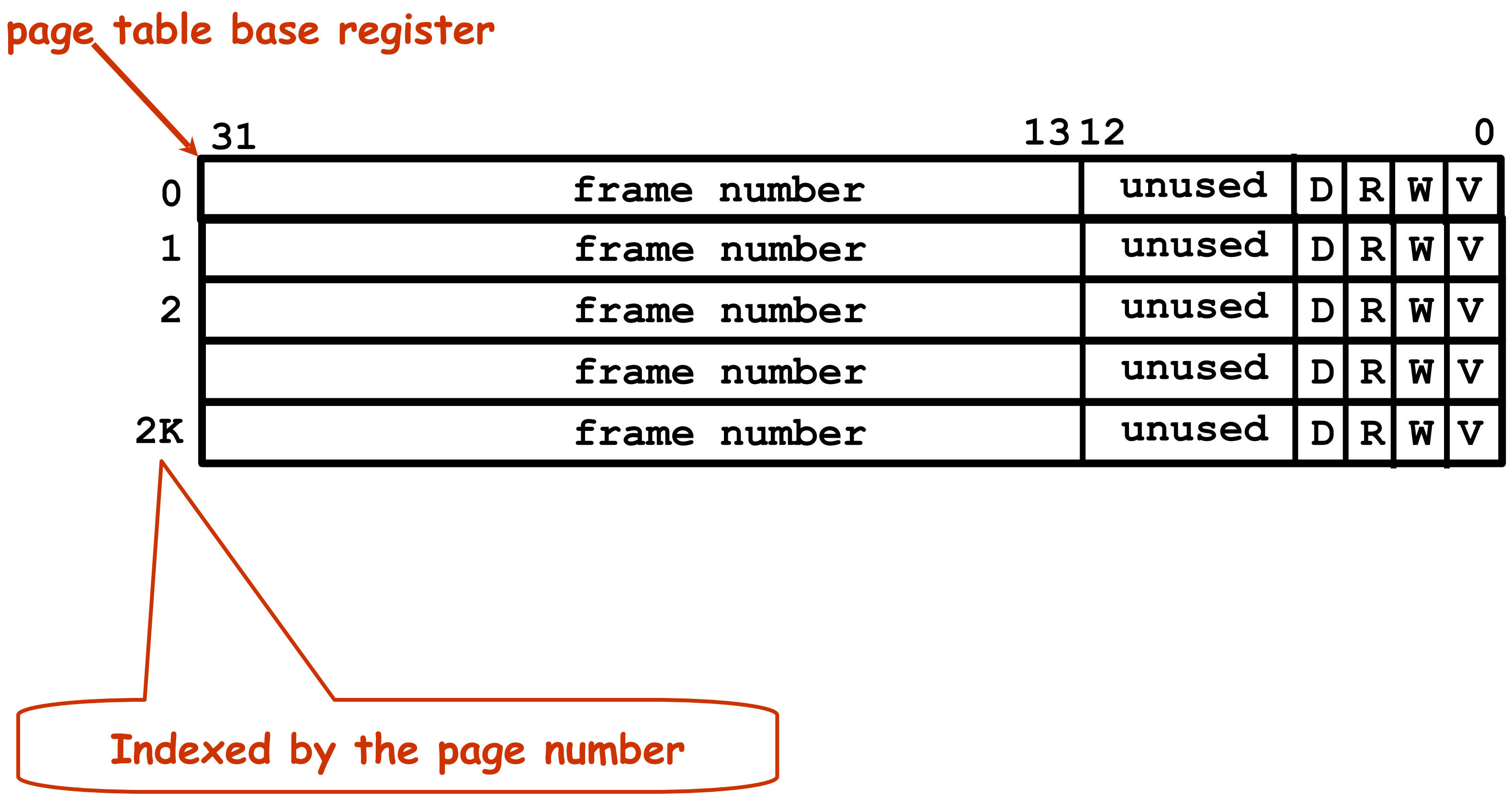
The BLITZ page table

- A page table entry



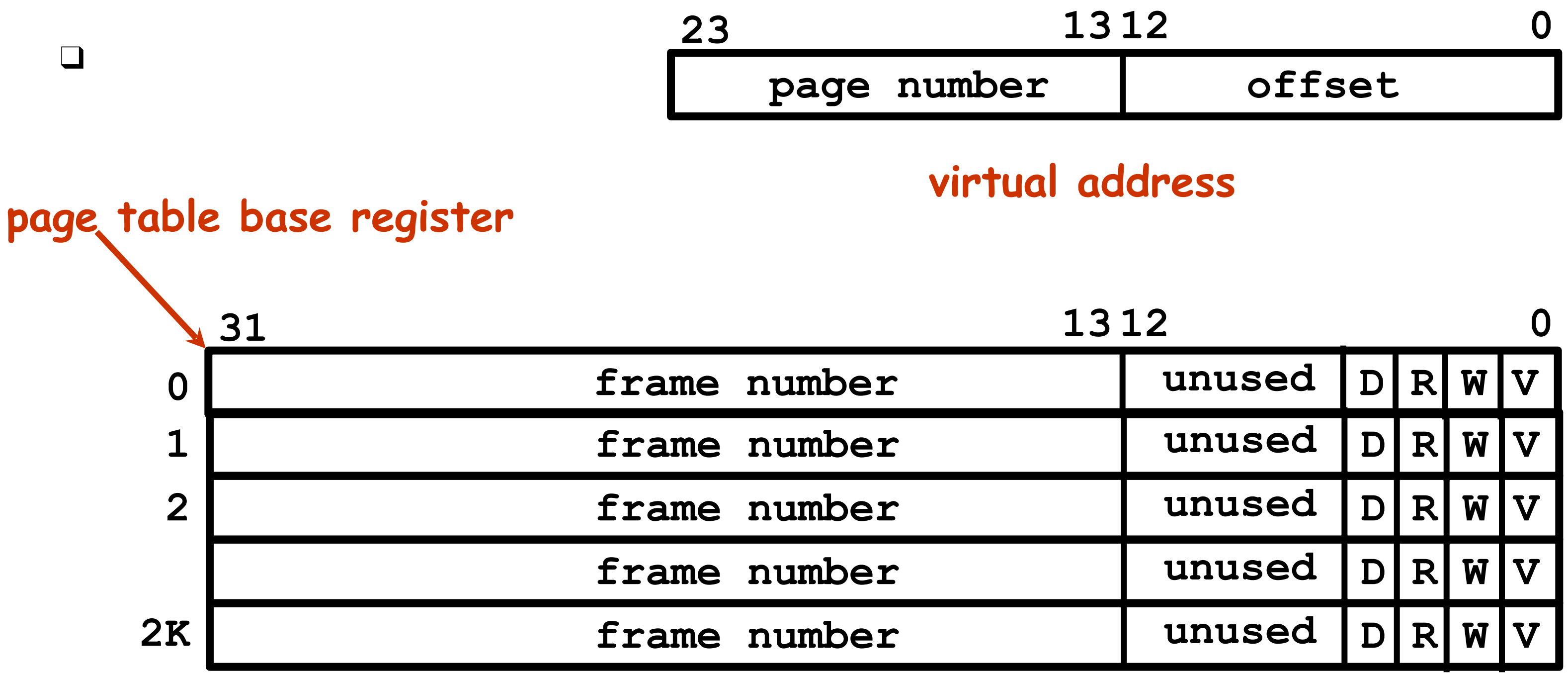
The BLITZ page table

- The full page table



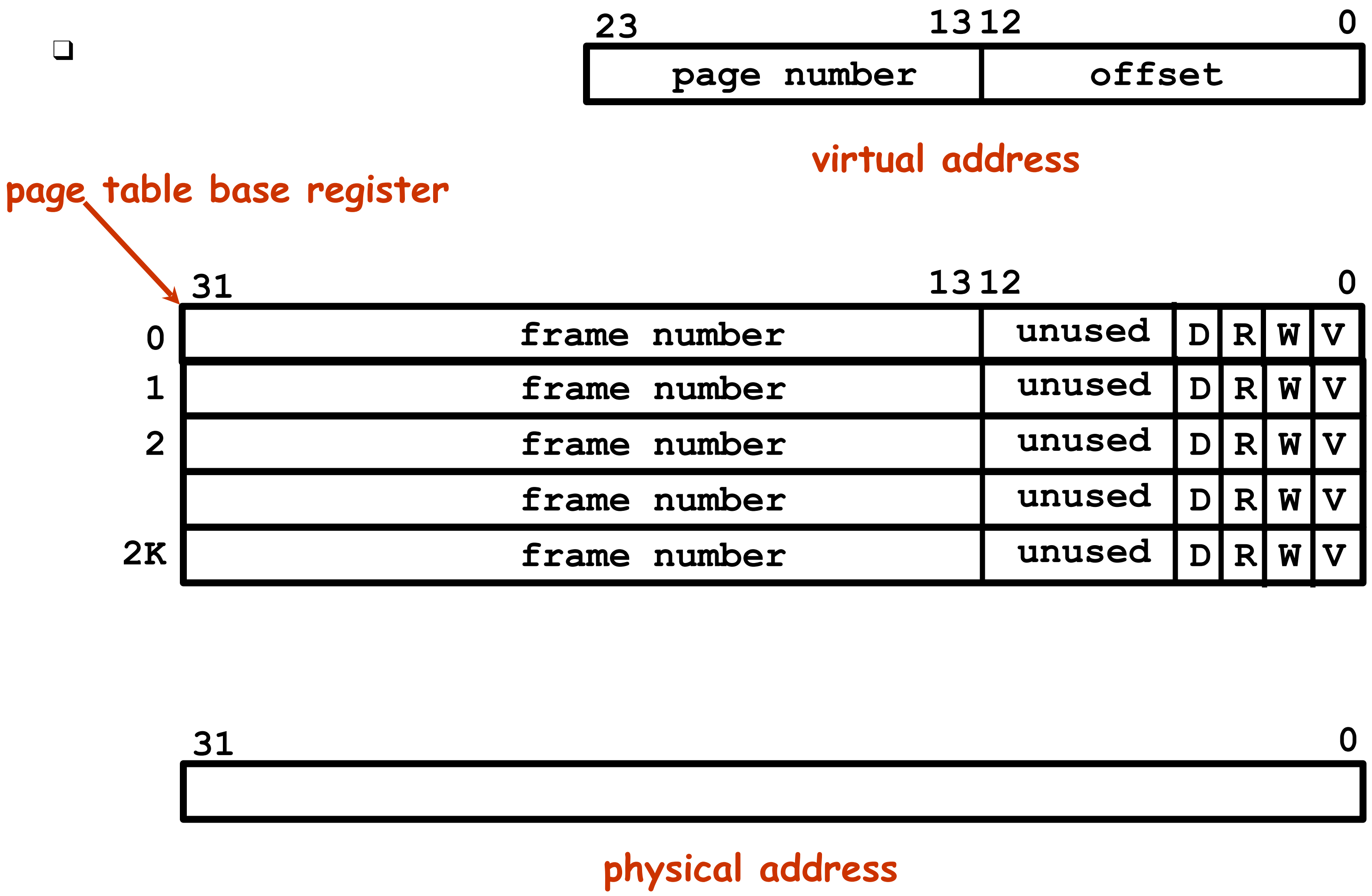
The BLITZ page table

❑



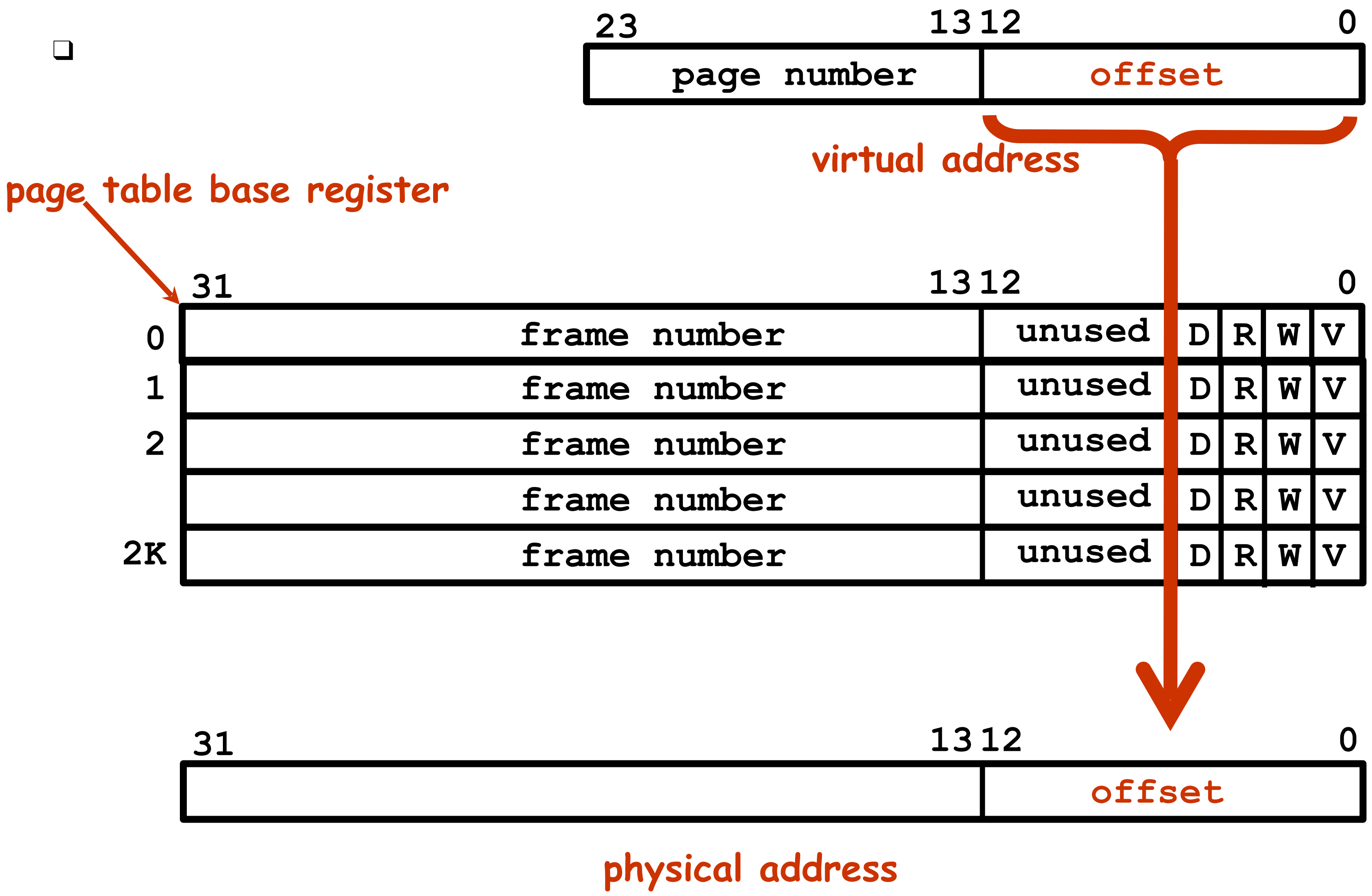
The BLITZ page table

-



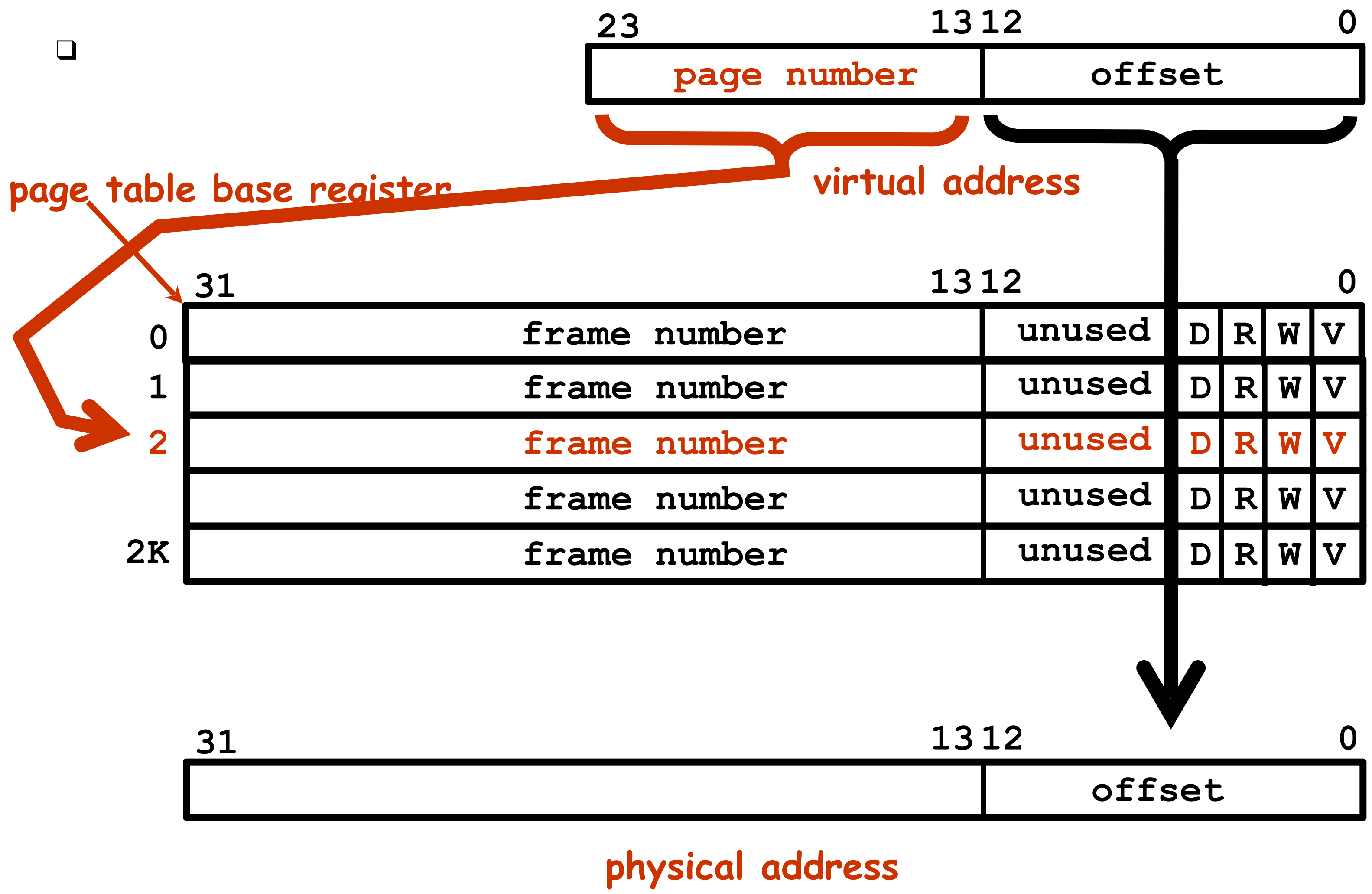
The BLITZ page table

-



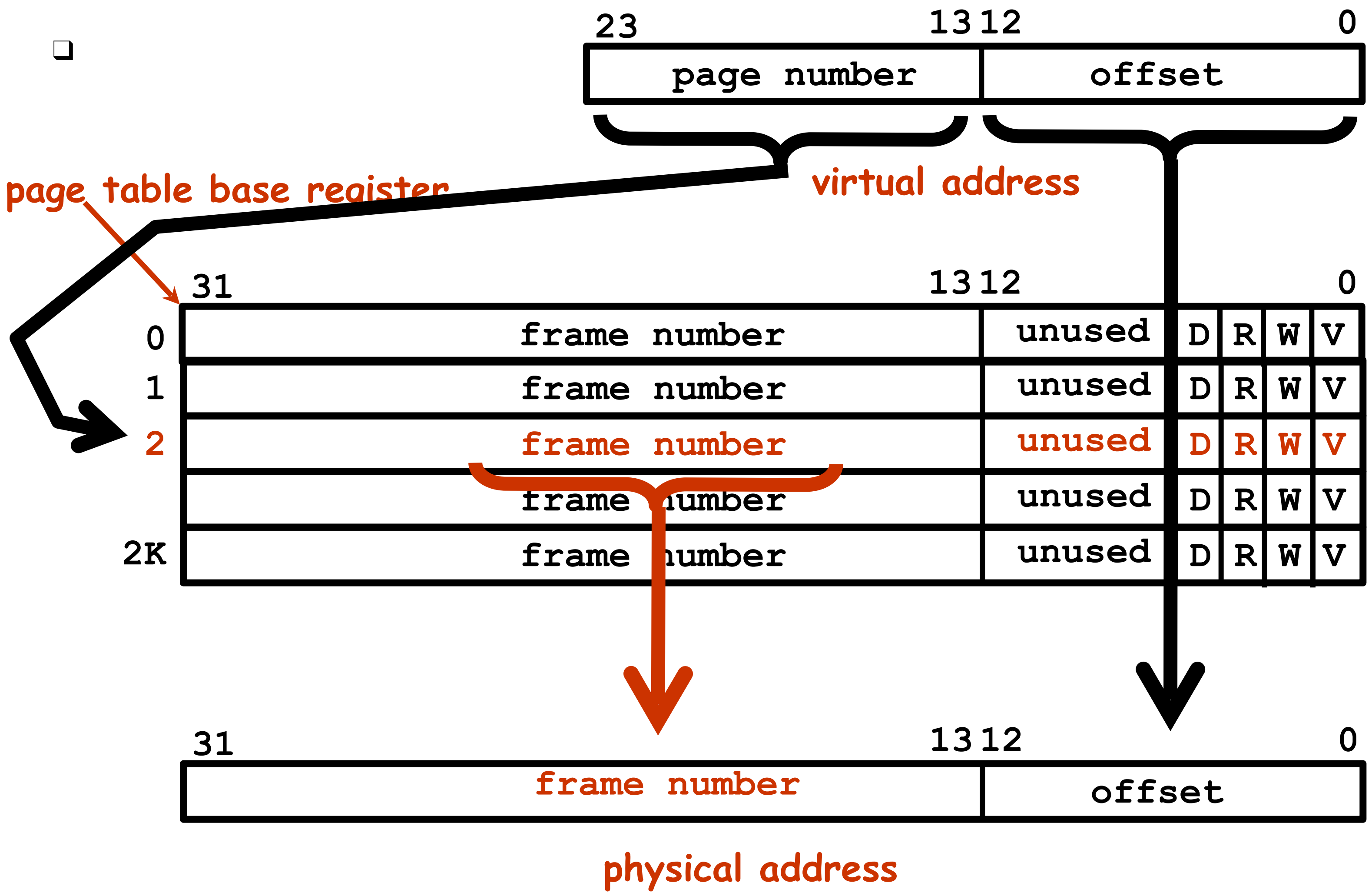
The BLITZ page table

❑



The BLITZ page table

❑



23
13
12
0
page number
offset
virtual address
page table base register
31
13
12
0
0
1
2
2K
frame number
unused
D
R
W
V
31
13
12
0
frame number
offset
physical address