xv6©-rev10

(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)

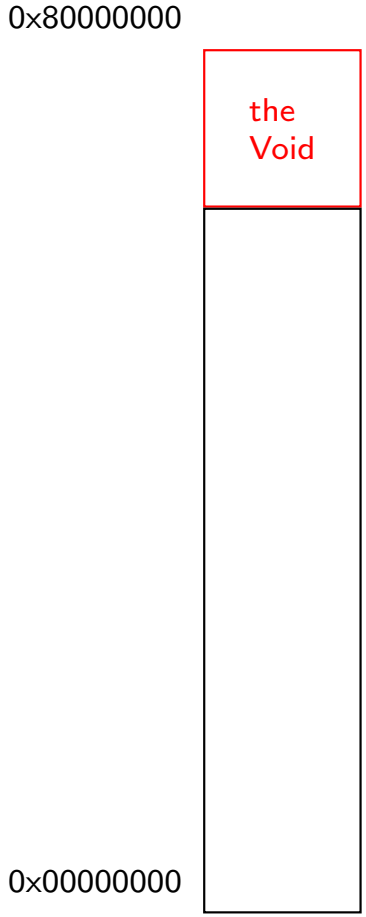
Paging

Carmi Merimovich

Tel-Aviv Academic College

September 19, 2017

xv6 Process address space

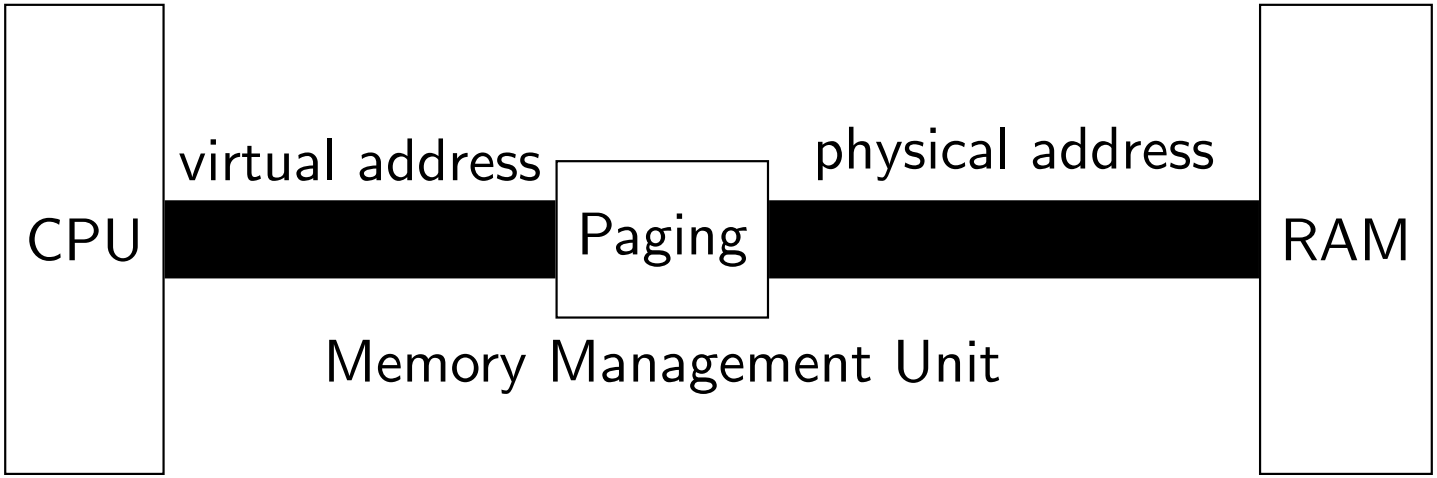


How can several processes have overlapping address spaces?

Naive Picture

Logically, this is the boot state.

More Realistic Picture

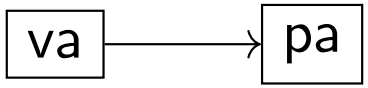


The paging unit is software enabled.

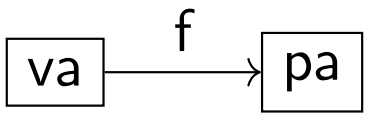
Solution?

- Assume the kernel controls the function the mmu implements.
- Process can occupy arbitrary physical addresses.
- The mmu will be configured to translate to this physical addresses.
- On context switch, the mmu will be reconfigured.

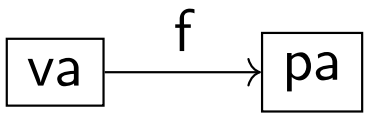
Address translation



Address translation

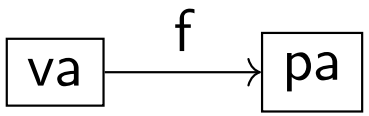


Address translation



$$\texttt{pa} = \texttt{f}(\texttt{va});$$

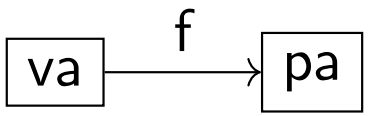
Address translation



$$\texttt{pa} = \texttt{f}(\texttt{va});$$

- Thus, the paging unit implements a function.
- The kernel must have the ability to control this function.
- What is a reasonable way to define a discrete function?

Address translation



$$\texttt{pa} = \texttt{f}(\texttt{va});$$

- Thus, the paging unit implements a function.
- The kernel must have the ability to control this function.
- What is a reasonable way to define a discrete function?

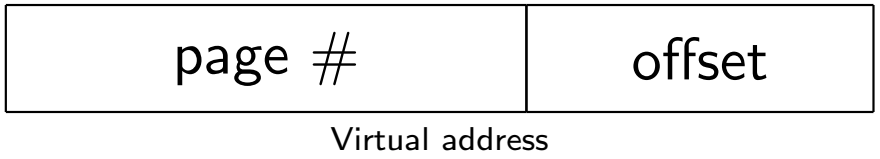
A VECTOR.

the VECTOR size

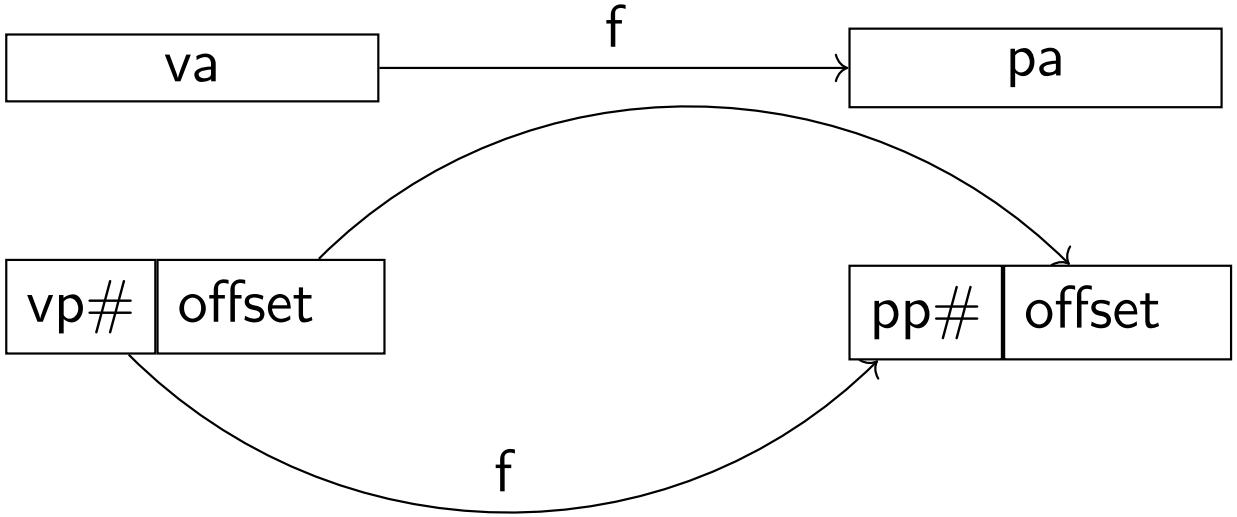
- Assume 32-bit **va** and 32-bit **pa**.
- We need to support (at least) illegal **va**'s.
- Vectors entries will be power of 2 bytes, hence 8B.
- 2^{32}-entries each of size 8B means vectors of size 32GB.

Pages!

- Having a vector entry for each virtual address is not feasable.
- We loose granularity by having a vector entry indexed by smaller field:



- The function works on the page #, and the offset is preserved

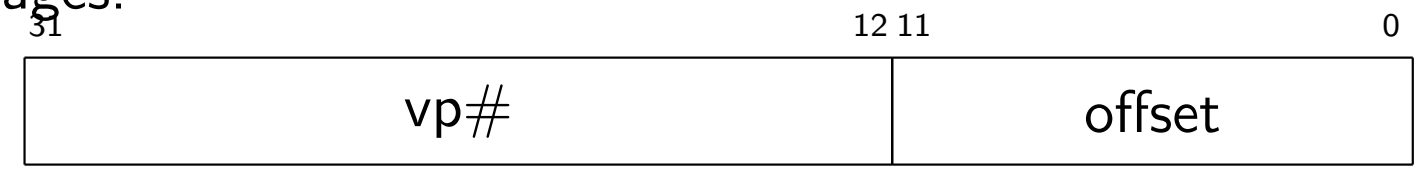


$$\texttt{pp\#} = \texttt{f}(\texttt{vp\#});$$

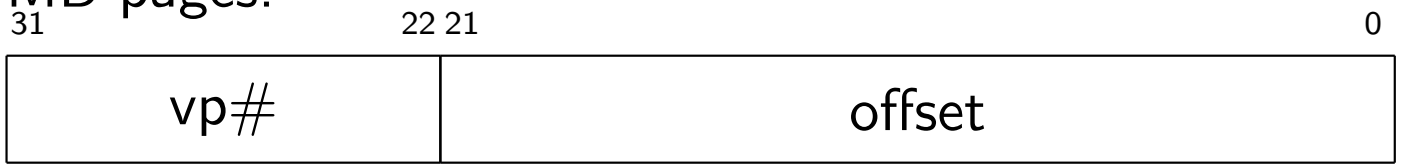
x86 pages

x86 can use in parallel pages of sizes 4KB and (elective) 4MB:

- 4KB pages:

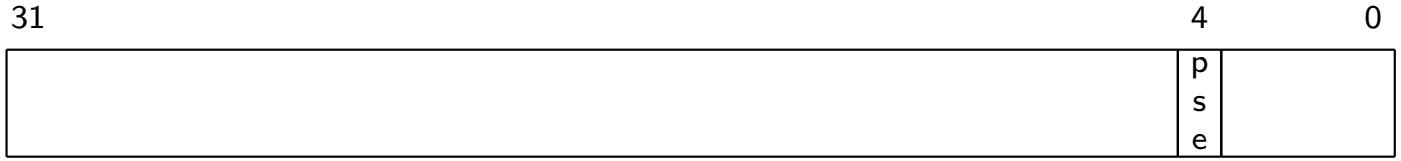


- (elective) 4MB pages:



This option should be enabled:

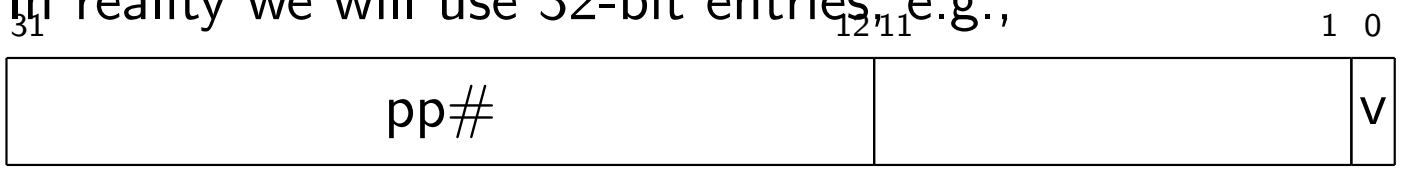
```
movl %cr4,%eax
orl  $CR4_PSE,%eax
movl %eax,%cr4
```



The cr4 register

x86 vector size

- Assume the mmu register **vecr** points to the translation vector.
- For 4KB pages we need vector of length 2^{20}.
- Each entry in the vector should be at least 20 (in fact 21) bits.
- In reality we will use 32-bit entries, e.g.,



- The size of this vector would be 4MB!
- The function $\texttt{pp\#} = \texttt{f}(\texttt{vp\#})$ would be given by

$$\texttt{pp\#} = (\texttt{vec}[\texttt{vp\#}] >> 12)\&\texttt{0xFFFFF};$$

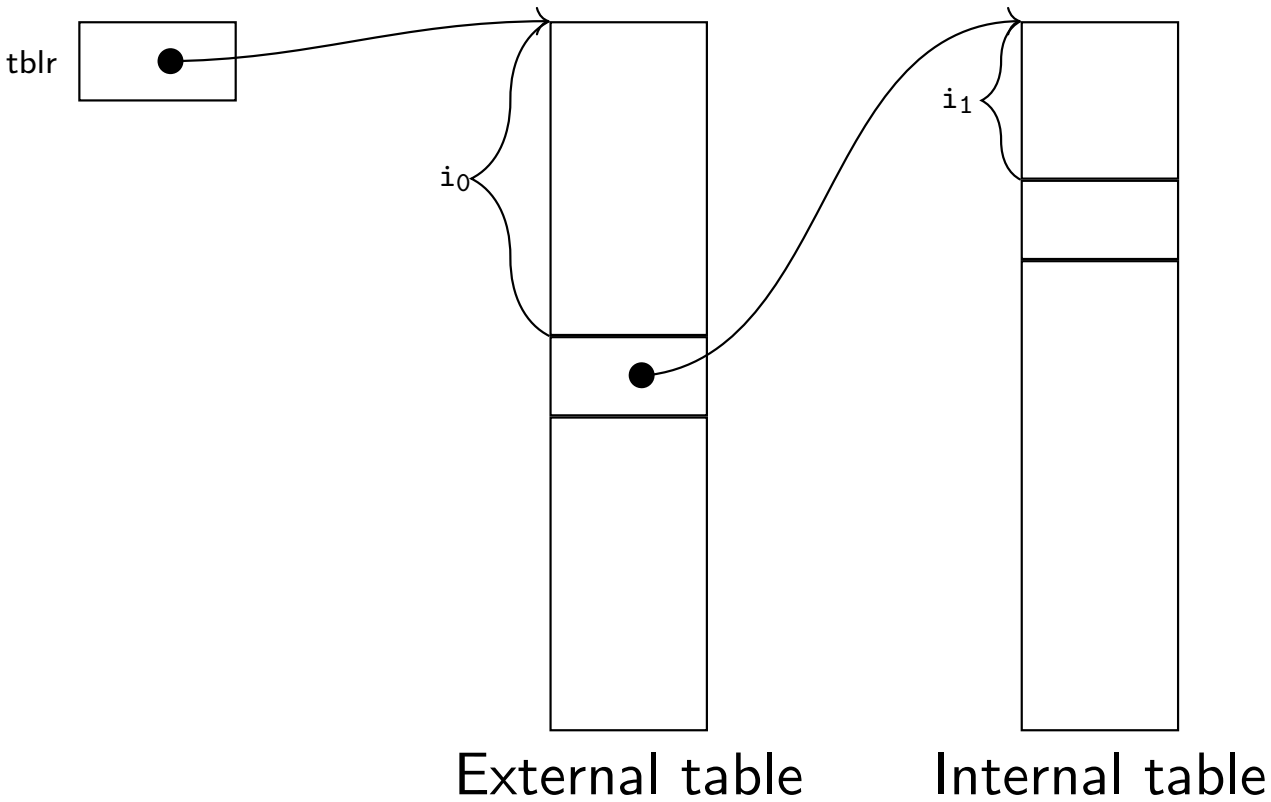
- **vecr** would have to contain physical address.
- 4MB of continuous physical memory is a bad idea.

2 dimensional Page table

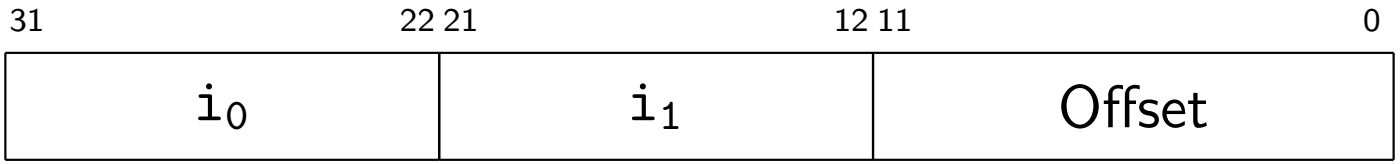
The mmu splits the vp# into two fields: i_0 and i_1:

i_0	i_1	Offset

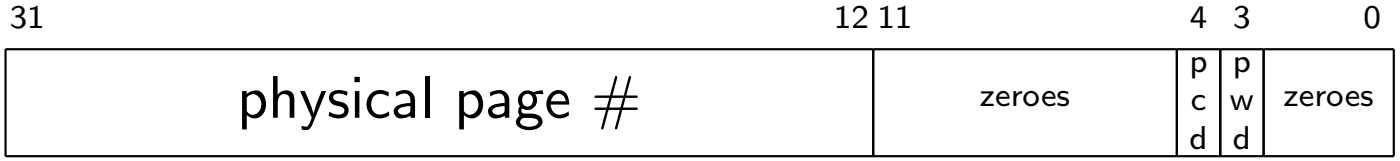
Virtual Address



x86 4KB pages, standard

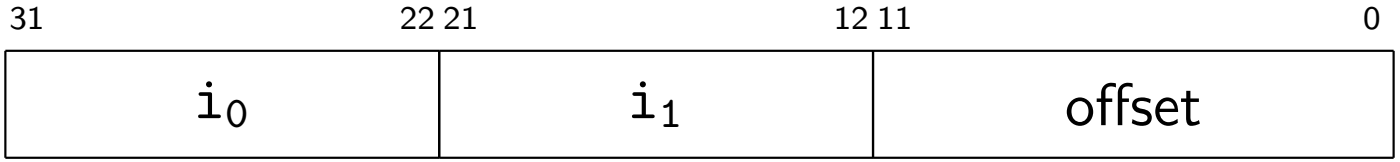


- External table size: $2^{10} \times 4\text{B} = 4\text{KB}$.
- Internal table size: $2^{10} \times 4\text{B} = 4\text{KB}$.
- **cr3** points to the external table:



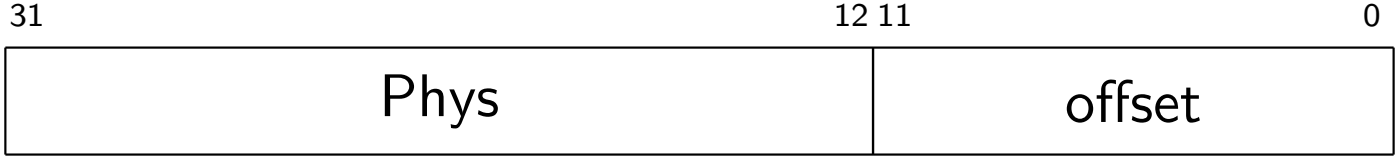
x86-32 two level translation

Ignoring permissions, validity, etc.



Virtual Address

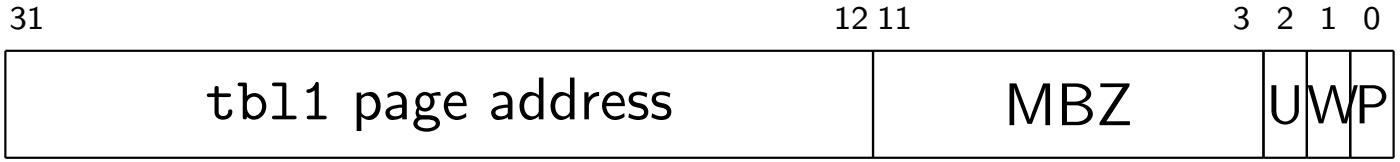
$$\texttt{tbl}_1 \leftarrow \textbf{cr3}[\texttt{i}_0];$$
$$\texttt{Phys} \leftarrow \texttt{tbl}_1[\texttt{i}_1];$$



Physical Address

x86 tbl0 entry structure

cr3 points to a 1024-entries vector. Each entry is of the form:



P one for page is Present, zero for generating page-fault.

U one for User mode can access, zero for User mode cannot access.

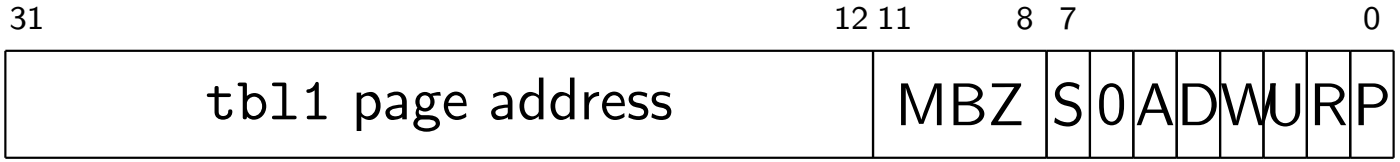
W one for R/W page, zero for Read-only page.

The above is in xv6 notation, i.e.,

```
#define PTE_P 1
#define PTW_W 2
#define PTE_U 4
```

real x86 tbl0 entry structure (elective)

cr3 points to a 1024-entries vector. Each entry is of the form:



- P one for page is Present, zero for generating page-fault.
- U one for User mode can access, zero for User mode cannot access.
- R one for R/W page, zero for Read-only page.
- S Zero for 4KB page Size, one for 4MB page Size.
- A one for page Accessed.
- D one for cache Disabled.
- W one for Write-through, zero for Write-back.

Kernel can always access a page, unless CR0.WP is in effect.
S will work only if CR4.PSE is set.

x86 tbl1 entry structure

cr3[i_0] points to a 1024-entries vector. Each entry is of the form:

P one for page is Present, zero for generating page-fault.

U one for User mode can access, zero for User mode cannot access.

W one for R/W page, zero for Read-only page.

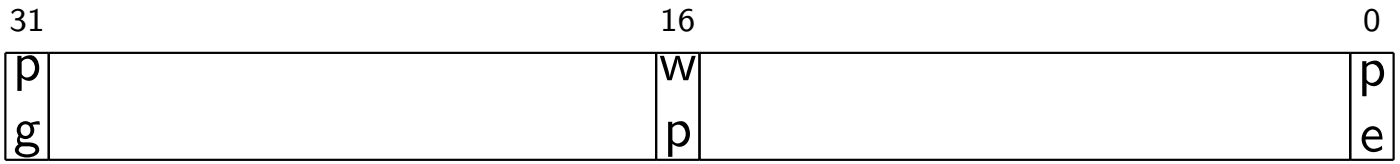
real x86 tbl1 entry structure (elective)

cr3[i_0] points to a 1024-entries vector. Each entry is of the form:

- P one for page is Present, zero for generating page-fault.
- U one for User mode can access, zero for User mode cannot access.
- R one for R/W page, zero for Read-only page.
- G one for Global, i.e., not remove from TLB when cr3 loaded.
- D page is Dirty, i.e., written to.
- A one for page Accessed.
- C one for Cache disabled.
- W one for Write-through, zero for Write-back.

G is in effect only if CR4.PGE is one.

x86: Enable paging (Elective)



The **cr0** register.

pe - 1 for protected mode (i.e., segmentation enabled).

wp - 1 for kernel faults on accessing read only pages.

pg - 1 for paging enabled.

xv6 code for enabling paging (pe must be set already!):

```
movl %cr0,%eax
orl  $CR0_PG|CR0_WP,%eax
movl %eax,%cr0
```