

# Introduction

## Virtual memory

The main memory is used as disc cache

- Some part of the disc is used as a level in the memory hierarchy
- Reduces disc latency
- Principle of locality

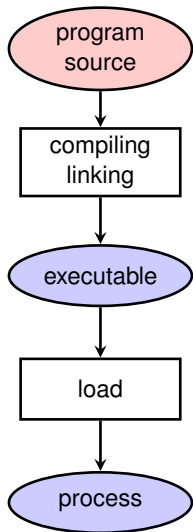
## Advantages

- Increases the capacity of the memory system
- Increases the functionality of the memory system

## Limitations

- Increases the complexity of the memory system
- Requires OS support

# Addresses



## Symbolic addresses

String of characters

- names of variables

## Relocatable addresses

Address space owned by the process

- virtual addresses

## Absolute addresses

Absolute position in physical memory

- reach memory devices

# Objectives

- O1 Enlarge the capacity of the memory system
- O2 Provide memory protection support
- O3 Share memory
- O4 Simplify development and load tools

# O1: Enlarge the capacity of the memory system

## Use the main memory as disc cache

- Run programs larger than the capacity of the main memory
- Run several programs simultaneously

## Disc area

- Windows  $\Rightarrow$  page file
- Unix/Linux  $\Rightarrow$  swap partition

## Virtual addresses

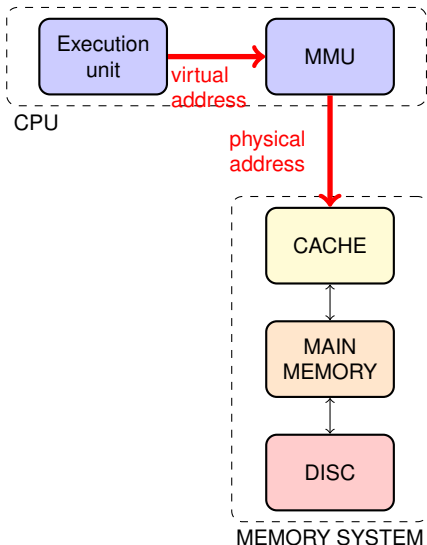
### Two address types

- Virtual: Used by programs
- Physical: Used by the memory system

Translation?

### Memory Management Unit

- Hardware
- Translation tables



# Address space

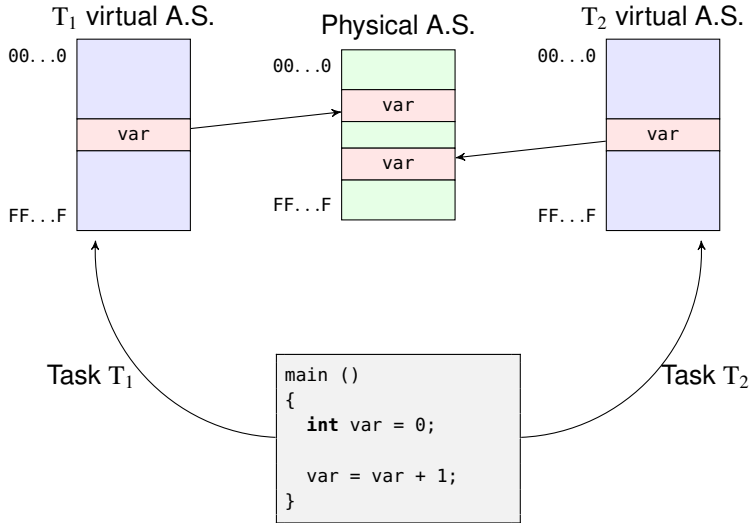
## Virtual address space

- Set of all virtual addresses
- Fixed set → depends on the CPU addressing capacity
- One complete space per task

## Physical address space

- Set of all physical addresses
- Variable set → depends on the physical memory installed in the computer
- One space per computer

# Example

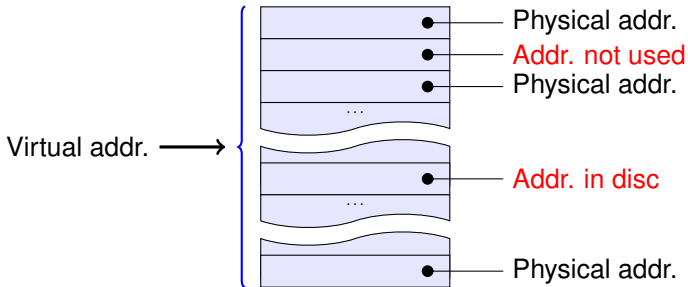


# Simplified translation

## Translation table

One per task

- Virtual address  $\Rightarrow$  Physical address / Not used / In disc





## Cache-like operation

- ① The execution unit generates a virtual address
- ② The MMU translates this address
  - ✓ Hit  $\Rightarrow$  the data item is in main memory
  - ✗ Miss  $\Rightarrow$  exception
    - The OS determines that the data item is in the disc
      - 1.- A block is copied from the disc to the main memory
      - 2.- The translation table is updated
      - 3.- The access is repeated
    - The OS determines that the data item is **not** in the disc

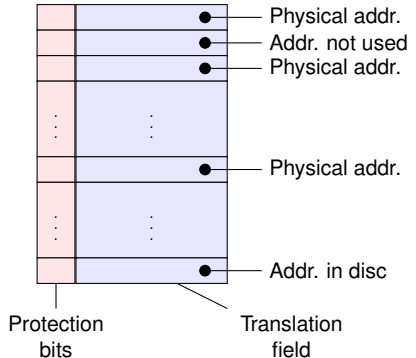
## O2: Memory protection

### Twofold

- Protect the OS against tasks
- Protect tasks among themselves

### Solution

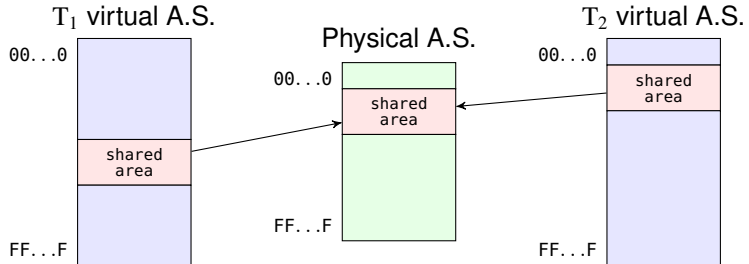
- One translation table per task
- Protection bits



## O3: Memory sharing

### Between two or more tasks

- Task communication
- Code sharing



## O4: Development and load tools

### The virtual memory facilitates

- Compiling and linking programs
- Loading programs

### Independent virtual address spaces

- Locate data and code structures
- All addresses available
- Do not worry about other tasks

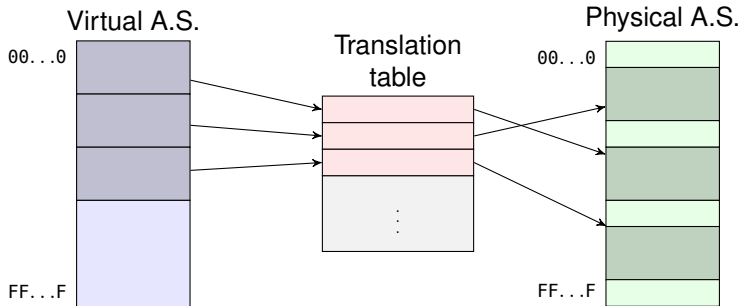
### Translate virtual address $\Rightarrow$ physical address

- Locate tasks in any area of the main memory
- Track free areas

# The translation table

Translate virtual address  $\Rightarrow$  physical address

- Do not translate one address at a time (**cost**)
- Do it in blocks



# Virtual memory strategies

## Segmentation

- ✓ Variable-size block  $\Rightarrow$  Efficient distribution
  - Address = segment + offset
- ✓ Very large address space
- ✗ External fragmentation
- ✗ Visible to the programmer

## Paging

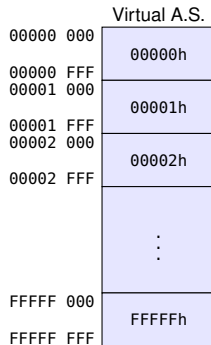
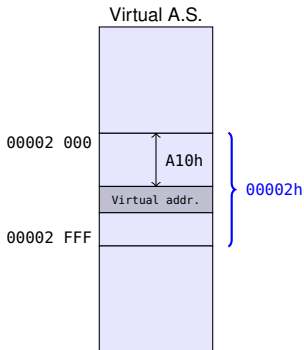
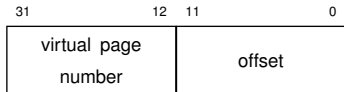
- Fixed-size blocks (pages)
- Address = one register
- ✗ Shorter address space
- ✓ No external fragmentation; a bit of internal fragmentation
- ✓ Transparent to the programmer

# Paging

## Virtual address

- Virtual page number
- Offset

00002A10

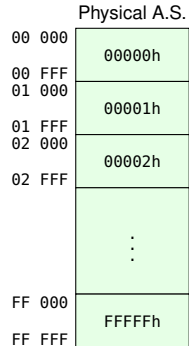
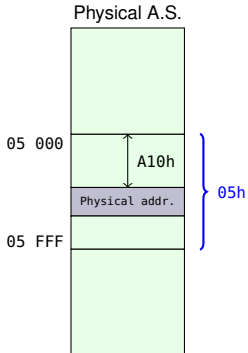
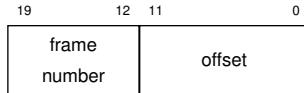


# Paging

## Physical address

- Frame number
- Offset

5F A10





# Page table

## Translation

Virtual page  $\Rightarrow$  Physical page (frame)

- The offset remains the same

## Characteristics

- One per task
- Managed by the OS
- Pointed by the page table register
- As many entries as virtual pages

# Page table

## Fields

- **Presence bit**
  - ✓ Page in memory (accessible)
  - ✗ Page in disc or not valid
- **Page frame/Disc location**: Depends on the presence bit
  - ✓ Page frame
  - ✗ Disc location or page not valid
- **Protection bits**
  - Page permissions
- **State bits**: Facilitate management  $\approx$  cache
  - *dirty*  $\Rightarrow$  shows whether or not the page has been modified
  - *accessed*  $\Rightarrow$  shows whether the page has been accessed recently

# Page table example

## 32-bit virtual address

- Page: 20 bits  $\Rightarrow 2^{20}$  pages
- Offset: 12 bits  $\Rightarrow$  4-KiWord page

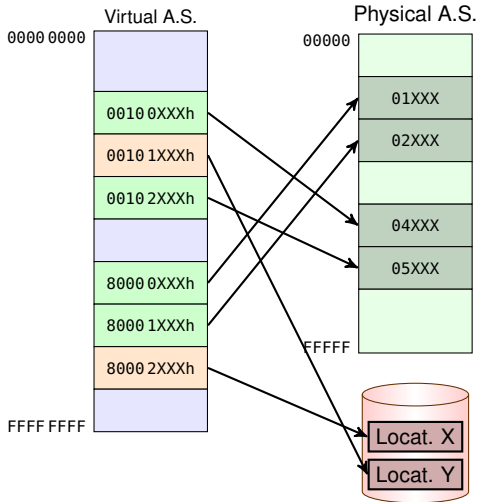
## 20-bit physical address

- Frame: 8 bits  $\Rightarrow 2^8$  frames
- Offset: 12 bits  $\Rightarrow$  4-KiWord page

1-byte word  $\Rightarrow$  4-KiB page

# Page table example

| Page  | Pres. | Frame/Location |
|-------|-------|----------------|
| 00000 | No    | Not valid      |
| 00001 | No    | Not valid      |
| ...   | ...   | ...            |
| 00100 | Yes   | 04             |
| 00101 | No    | Location Y     |
| 00102 | Yes   | 05             |
| 00103 | No    | Not valid      |
| ...   | ...   | ...            |
| 80000 | Yes   | 01             |
| 80001 | Yes   | 02             |
| 80002 | No    | Location X     |
| 80003 | No    | Not valid      |
| ...   | ...   | ...            |
| FFFFE | No    | Not valid      |
| FFFFF | No    | Not valid      |



# Translation example

Virtual addr.                      Physical addr.

80001 5AB                       $\xrightarrow{\text{translate}}$                       02                      5AB

| Page  | Pres. | Frame/Location |
|-------|-------|----------------|
| 00000 | No    | Not valid      |
| 00001 | No    | Not valid      |
| ...   | ...   | ...            |
| 00100 | Yes   | 04             |
| 00101 | No    | Location Y     |
| 00102 | Yes   | 05             |
| 00103 | No    | Not valid      |
| ...   | ...   | ...            |
| 80000 | Yes   | 01             |
| 80001 | Yes   | 02             |
| 80002 | No    | Location X     |
| 80003 | No    | Not valid      |
| ...   | ...   | ...            |
| FFFFE | No    | Not valid      |
| FFFFF | No    | Not valid      |

# Tasks in memory

32-bit computer with 4-KiB pages

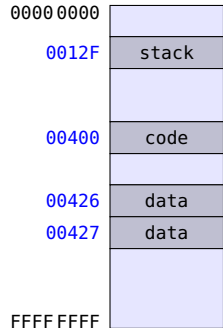
```
float var[1500];

main ()
{
    int i;

    for (i = 1; i < 1500; i++)
        var[i] = 0;
}
```

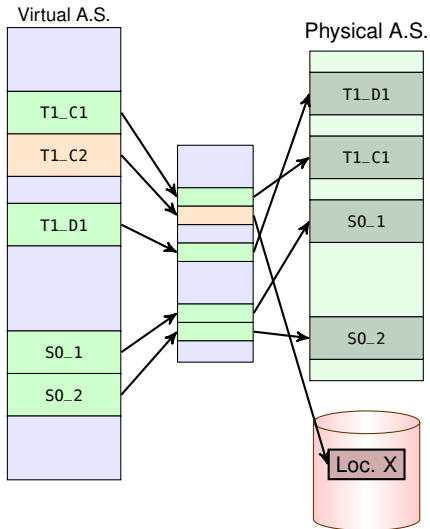
Compiler  
+  
Linker

Virtual A.S.



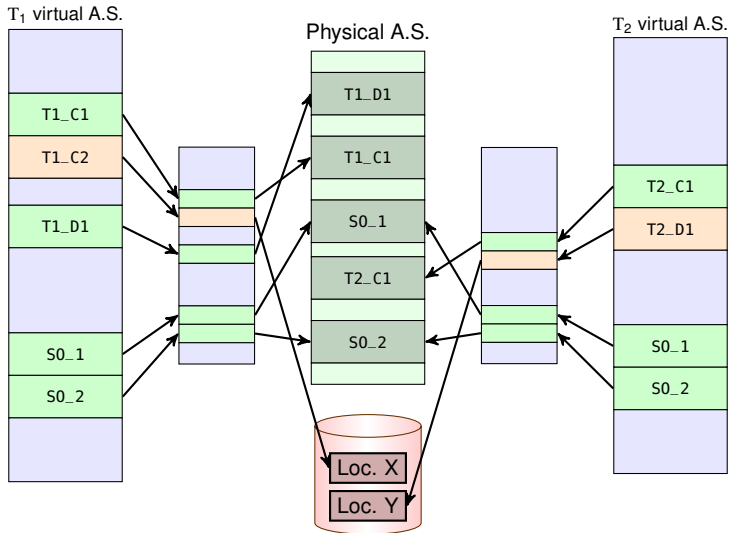
## Adding the OS

- It is located in the virtual A.S. of each task
- Resources in the page table
- The page table translates task and OS addresses



## Adding the OS (2 tasks)

Tasks share OS memory





# Memory protection

## Independence among task address spaces

- Each task has its own page table
- Overlaps in main memory are avoided

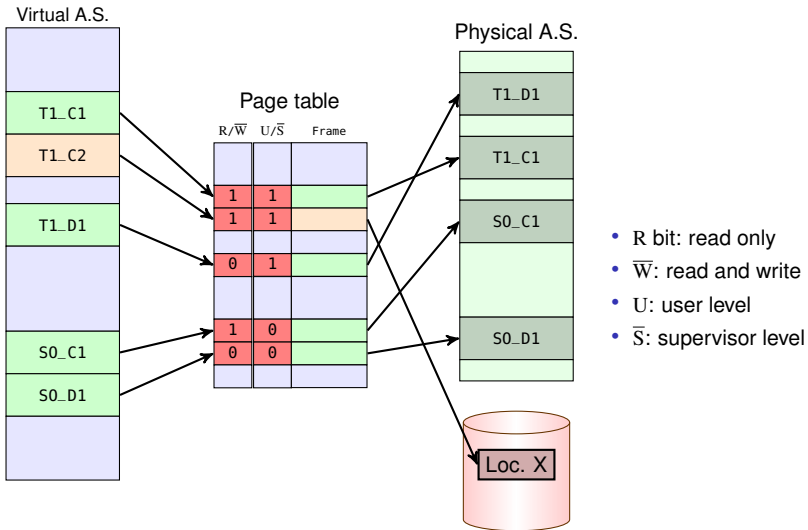
## Access type

- Additional bits in each entry of the page table for a page to be accessed

## Privilege level

- Each page has a privilege level assigned

# Protection bits



# Protection example

## Windows 32 bits

Virtual memory divided in two ranges

- 0000 0000h - 7FFF FFFFh (2 GiB): task accessible
- 8000 0000h - FFFF FFFFh (2 GiB): OS only accessible

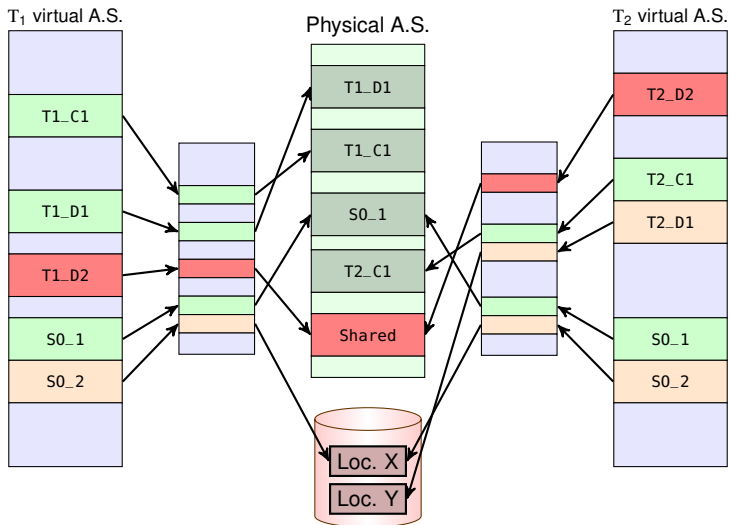
## Linux 32 bits

Virtual memory divided in two ranges

- 0000 0000h - BFFF FFFFh (3 GiB): task accessible
- C000 0000h - FFFF FFFFh (1 GiB): OS only accessible


# Memory sharing

Shared memory is requested to the OS



# Memory sharing

## Problems



Cached shared memory area?

Real caches operate with virtual addresses



Data item modified in cache?

Page *dirty* bit

Coherence among page tables

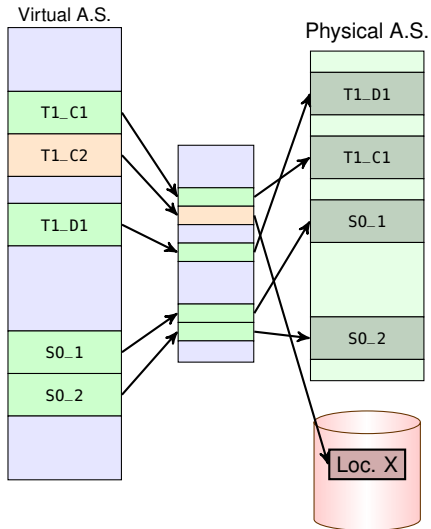
# Performance issue

## Double memory access

- 1 Page table entry
  - Address translation
- 2 Data item

## Solution

TLB (Translation lookaside buffer)



- Principle of locality of references
  - Access to a few virtual pages
  - Few entries of the page table are accessed
- A cache can be used
- More used entries are stored in the TLB
  - Specific cache
  - Inside the CPU

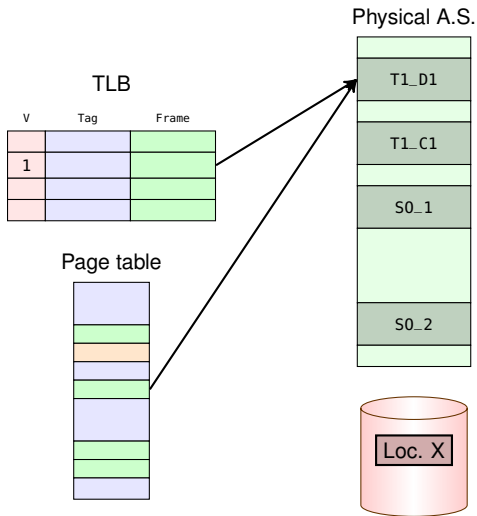
# TLB

## Without TLB

- 1 Page table
- 2 Data item

## With TLB

- 1 TLB (valid bit)
- 2 Data item





## Characteristics

- Cache memory
- Stores more accessed entries of the page table
- Valid and access bits; tag
- Fully associative cache
- Replacement and writing strategies
- Managed by hardware (IA32) / software (MIPS)

## Operation in a task

Look for entries in the page table in the TLB

- ✓ Hit  $\Rightarrow$  Frame number is obtained very fast
- ✗ Miss  $\Rightarrow$  Update the TLB entry

## Issue while switching context

- Task switching  $\Rightarrow$  Switch the page table
- Several TLB entries are invalidated  $\Rightarrow$  TLB misses
  - Performance penalty

## Solution

Extra bits for each TLB entry

- Task identification
- Avoid invalidating the whole TLB
- OS table page entries marked as global

Why is the TLB necessary if there is a cache?

- Cache memory can store entries of the page table
- Unified caches are more efficient

## Advantages

Simultaneous access to cache and TLB

- Address translation and access

# Combining TLB and cache

## Virtual address translation

- 1 The TLB is accessed with the virtual page number
- 2 Hit  $\Rightarrow$  The frame number is obtained
  - The address is translated
- 3 Physical address is interpreted according to the cache
- 4 The data item is fetched from the cache

## Problem

It is not possible to access concurrently the TLB and the cache

- Segmented access
- Modern computers: Caches indexed with virtual addresses
  - Coherence in shared pages

# Page fault

## When?

The virtual address is not in physical memory

- Presence bit reset

## What happens?

Page miss (page fault) exception

- A handler of the OS is executed

## The OS determines the fault

- Address in a page with storage not assigned
- Address in a page in disc

# Page fault

## Non-recoverable fault

Page with storage not assigned

## Reading access

00103 42Ch



Exception

| Page. | Pres. | Frame/Location |
|-------|-------|----------------|
| 00000 | No    | Not valid      |
| 00001 | No    | Not valid      |
| ...   | ...   | ...            |
| 00100 | Yes   | 04             |
| 00101 | No    | Location Y     |
| 00102 | Yes   | 05             |
| 00103 | No    | Not valid      |
| ...   | ...   | ...            |
| 80000 | Yes   | 01             |
| 80001 | Yes   | 02             |
| 80002 | No    | Location X     |
| 80003 | No    | Not valid      |
| ...   | ...   | ...            |
| FFFFE | No    | Not valid      |
| FFFFF | No    | Not valid      |

# Page fault

## Recoverable fault

Page in disc

## Reading access

00101190h



Exception

The OS must decide:

- 1 Where to load the page
  - Replacement
- 2 Update the replaced page
  - *dirty* bit
- 3 Load the page and update the page table
- 4 Execute the instruction again

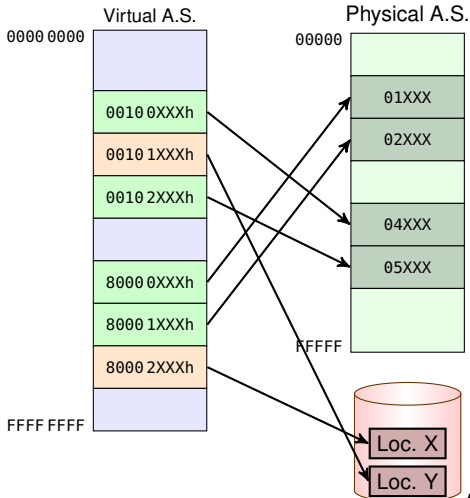
| Page. | Pres. | Frame/Location |
|-------|-------|----------------|
| 00000 | No    | Not valid      |
| 00001 | No    | Not valid      |
| ...   | ...   | ...            |
| 00100 | Yes   | 04             |
| 00101 | No    | Location Y     |
| 00102 | Yes   | 05             |
| 00103 | No    | Not valid      |
| ...   | ...   | ...            |
| 80000 | Yes   | 01             |
| 80001 | Yes   | 02             |
| 80002 | No    | Location X     |
| 80003 | No    | Not valid      |
| ...   | ...   | ...            |
| FFFFE | No    | Not valid      |
| FFFFF | No    | Not valid      |

# Page fault

## Reading access

00101190h  $\Rightarrow$  replace page 02. *Working set*

| Page. | Pres. | Dirty | Frame/Location |
|-------|-------|-------|----------------|
| 00000 | No    | -     | Not valid      |
| 00001 | No    | -     | Not valid      |
| ...   | ...   | ...   | ...            |
| 00100 | Yes   | No    | 04             |
| 00101 | No    | -     | Location Y     |
| 00102 | Yes   | Yes   | 05             |
| 00103 | No    | -     | Not valid      |
| ...   | ...   | ...   | ...            |
| 80000 | Yes   | Yes   | 01             |
| 80001 | Yes   | Yes   | 02             |
| 80002 | No    | -     | Location X     |
| 80003 | No    | -     | Not valid      |
| ...   | ...   | ...   | ...            |
| FFFFE | No    | -     | Not valid      |
| FFFFF | No    | -     | Not valid      |



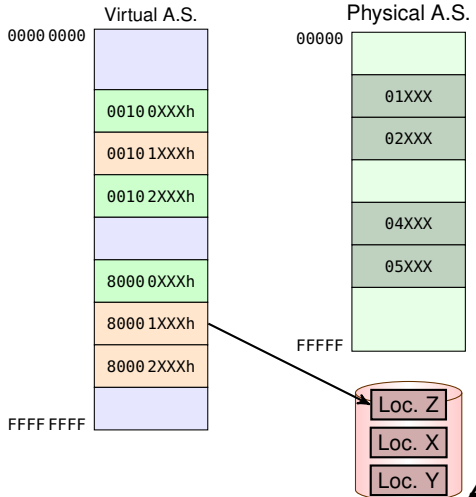


# Page fault

## Reading access

00101 190h  $\Rightarrow$  replace page 02. *Working set*

| Page. | Pres. | Dirty | Frame/Location |
|-------|-------|-------|----------------|
| 00000 | No    | -     | Not valid      |
| 00001 | No    | -     | Not valid      |
| ...   | ...   | ...   | ...            |
| 00100 | Yes   | No    | 04             |
| 00101 | No    | -     | Location Y     |
| 00102 | Yes   | Yes   | 05             |
| 00103 | No    | -     | Not valid      |
| ...   | ...   | ...   | ...            |
| 80000 | Yes   | Yes   | 01             |
| 80001 | No    | -     | Location Z     |
| 80002 | No    | -     | Location X     |
| 80003 | No    | -     | Not valid      |
| ...   | ...   | ...   | ...            |
| FFFFE | No    | -     | Not valid      |
| FFFFF | No    | -     | Not valid      |

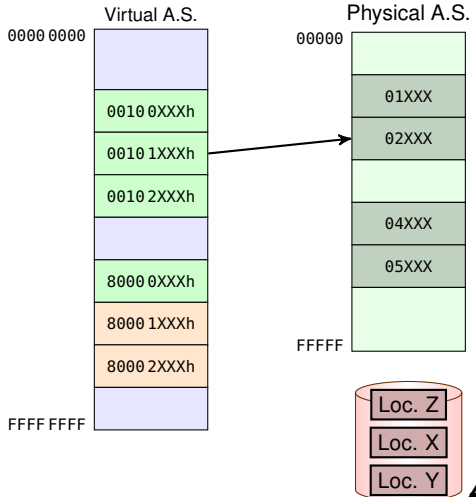


# Page fault

## Reading access

00101190h  $\Rightarrow$  replace page 02. *Working set*


| Page. | Pres. | Dirty | Frame/Location |
|-------|-------|-------|----------------|
| 00000 | No    | -     | Not valid      |
| 00001 | No    | -     | Not valid      |
| ...   | ...   | ...   | ...            |
| 00100 | Yes   | No    | 04             |
| 00101 | Yes   | No    | 02             |
| 00102 | Yes   | Yes   | 05             |
| 00103 | No    | -     | Not valid      |
| ...   | ...   | ...   | ...            |
| 80000 | Yes   | Yes   | 01             |
| 80001 | No    | -     | Location Z     |
| 80002 | No    | -     | Location X     |
| 80003 | No    | -     | Not valid      |
| ...   | ...   | ...   | ...            |
| FFFE  | No    | -     | Not valid      |
| FFFF  | No    | -     | Not valid      |



# Page fault

## TLB fault

- Exception  $\Rightarrow$  OS handler
- The entry is searched in the page table
  - Translation to physical address
- The entry is copied to the TLB
- All occupied?  $\Rightarrow$  Replacement strategy



Subsequent page fault?

# One-level page table

## Problem

Size of the page table

Virtual address: 32 bits

- Virtual page: 20 bits
- Offset: 12 bits

Page table

$$2^{20} \times 4 = 4 \text{ MiB}$$

| Page. | Pres. | Frame/Location |
|-------|-------|----------------|
| 00000 | No    | Not valid      |
| 00001 | No    | Not valid      |
| ...   | ...   | ...            |
| 00100 | Yes   | 04             |
| 00101 | No    | Location Y     |
| 00102 | Yes   | 05             |
| 00103 | No    | Not valid      |
| ...   | ...   | ...            |
| 80000 | Yes   | 01             |
| 80001 | Yes   | 02             |
| 80002 | No    | Location X     |
| 80003 | No    | Not valid      |
| ...   | ...   | ...            |
| FFFFE | No    | Not valid      |
| FFFFF | No    | Not valid      |

4 bytes

$2^{20}$

# IA-32 paging

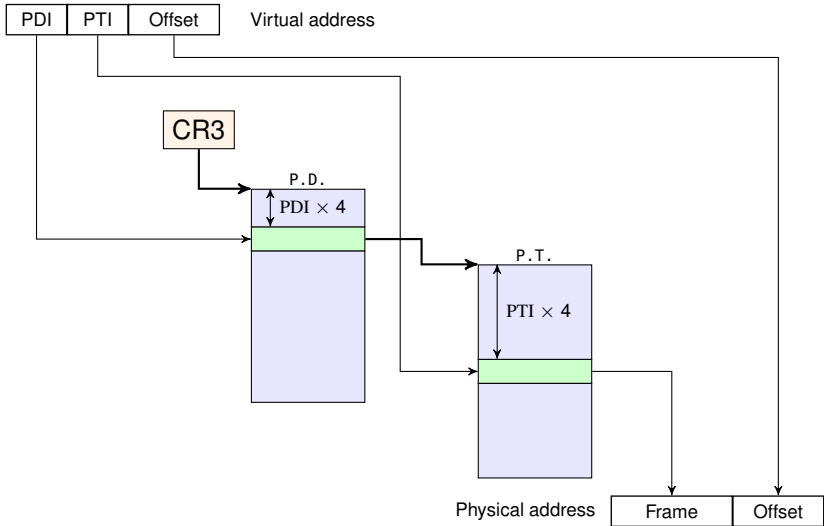
## Page table

- 1 Page directory (PD)
  - Pointed by the page table register (CR3)
- 2 Page table (PT)
  - Pointed by entries of the page directory

## Virtual address

- Virtual page: 20 bits
  - Page directory index (PDI): 10 bits
  - Page table index (PTI): 10 bits
- Offset: 12 bits

# IA-32 paging



# Two-level page table

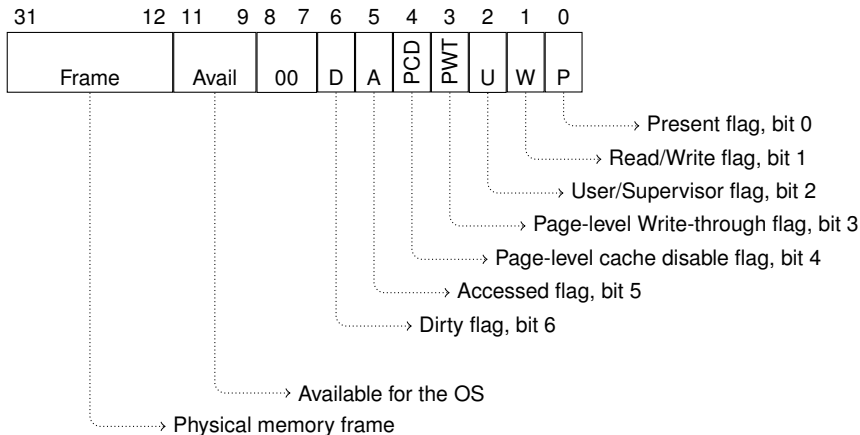
## Size

- PD  $\Rightarrow 1024 \times 4 = 4 \text{ KiB}$
- PT  $\Rightarrow 1024 \times 4 = 4 \text{ KiB}$
- Each page directory entry  $\Rightarrow 1024 \times 4 \text{ KiB}$

## Advantages

- PDE points to a PT of 1024 entries
- PDE covers 4 MiB of physical memory
- Entries in PD not used are marked as not valid
  - Memory savings  $\Rightarrow$  PT is not created
- PT can be paged to disc
- PD always in memory

# Page table entry



If  $P = 0 \Rightarrow$  bits 1 to 31 are available for the OS



# IA-32 paging

## Operation

- ✓ Present flag set in PDE  $\Rightarrow$  Page table frame
  - ✓ Present flag set in PTE  $\Rightarrow$  Page frame
  - ✗ Present flag reset in PTE  $\Rightarrow$  Exception
    - Page file
    - Page not valid
- ✗ Present flag reset
  - Two page faults can be chained

# IA-32 paging

## Multi-tasking operating system

- OS entries in the page table of each task
- Physical memory is shared

## Problem

- Many page tables with repeated entries
- Complexity when changing an OS page

## Solution

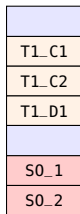
Establish local and global page tables

# Global page table

## Efficiency

- Local tables for tasks
- Global tables for OS
- Repeated entries in the PT are avoided

T<sub>1</sub> virtual A.S.



T<sub>2</sub> virtual A.S.

