

# Systems 3

## Memory Management

Marcel Waldvogel

Department of Computer and Information Science  
University of Konstanz

Winter 2019/2020

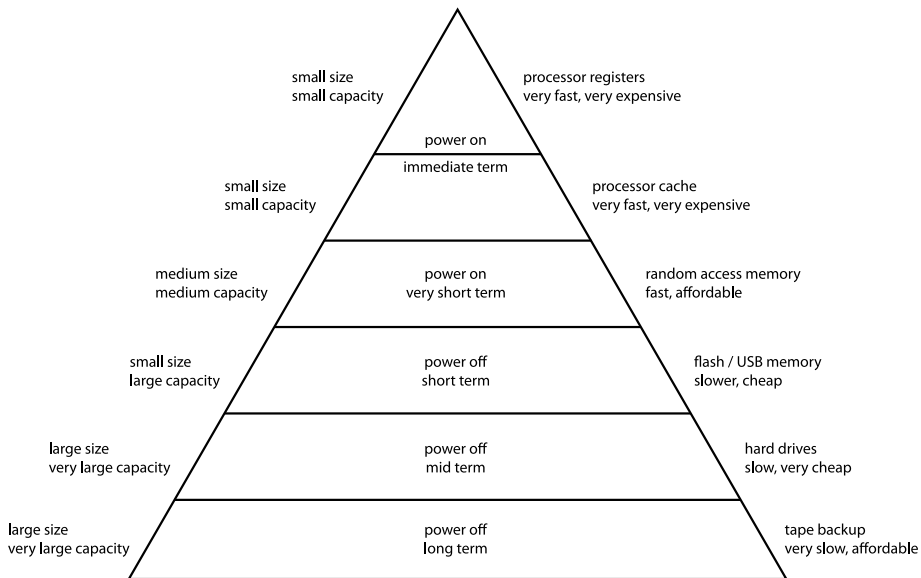
# Chapter Goals

- How is physical memory divided among processes?
- How is free physical memory allocated to processes?
- How can free memory be allocated in general?
- How can the memory management be simplified for both OS and application?

# Memory basics

- Which kinds of memory do you know?
- Why are there different types of memory?
- How does a program get executed?
- Why does a program need memory?

# Memory Hierarchy



# Basic Memory Management

## ■ Monoprogramming

User program	Operating system
--------------	------------------

## ■ Multiprogramming with Fixed Partitions

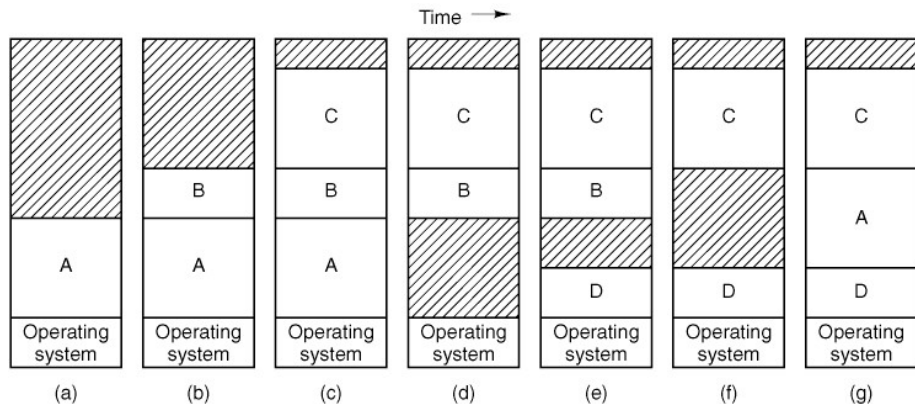
P1	P2	P3	OS
----	----	----	----

## ■ Relocation and Protection

# Swapping

- Why are presented solutions not sufficient?
- What is swapping?
- What is the difference between swapping and fixed partitions?

# Swapping



**Figure:** Memory allocation changes as processes come into memory and leave it. The shaded regions are unused memory. (Tannenbaum fig. 4-3)

# Memory Allocation Algorithms

Algorithm	Comment
First fit	Use first hole big enough
Next fit	Use next hole big enough
Best fit	Search list for smallest hole big enough
Worst fi	Search list for largest hole available
Quick fit	Separate lists of commonly requested sizes

(Dis-)advantages?



# OS Memory model: Motivation

- 1 Memory is not used uniformly (by processes)
- 2 Storage properties are not uniform

# Working set<sup>1</sup>

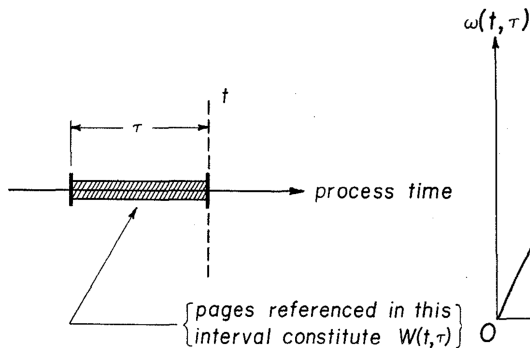


FIG. 2. Definition of  $W(t, \tau)$

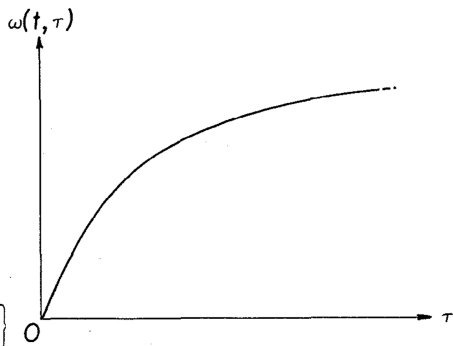


FIG. 3. Behavior of  $\omega(t, \tau)$

This locality results in a (slowly) growing working set. Memory areas, which have not been recently used are less likely to be accessed again soon and could be moved to slower memory.

<sup>1</sup>Denning, Peter J. (1968). "The working set model for program behavior". Communications of the ACM. 11(5):323–333

# Memory model too rigid

## Memory needs to be contiguous

- Need pre-determined size
- No space to grow
- No use shrinking

→ **virtual addresses** (process sees contiguous space, but OS has flexibility)

## Memory is blocked

- Not all memory is needed throughout the process lifetime (code, data)
- Need pre-determined size

→ **virtual memory** (process sees all data, but some is slower)

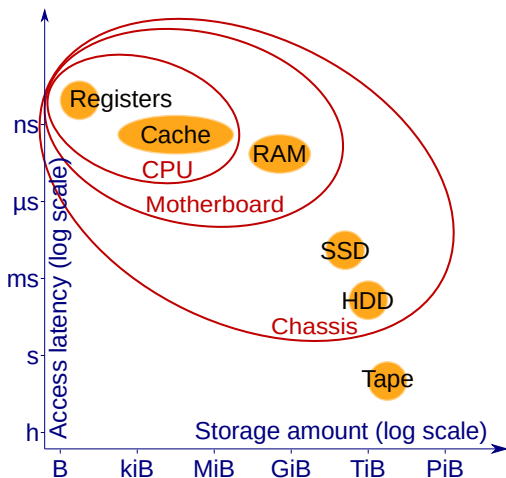
# Locality of Reference, speed/size/cost tradeoff

Data structures and program code (loops, shared functions, ...) frequently exhibit

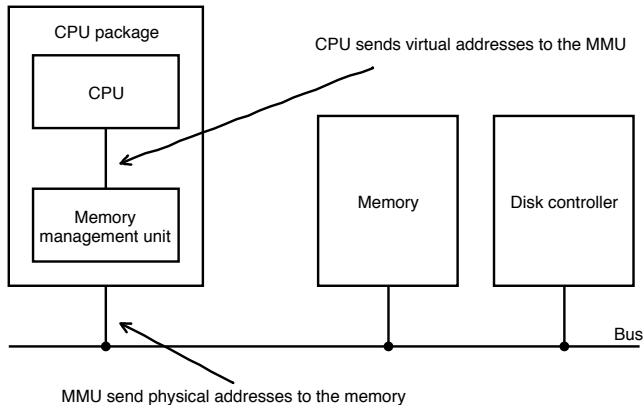
## Locality of Reference:

- Temporal locality  
(accessing the same address again soon) → cache
- Spatial locality  
(accessing nearby addresses) → cache lines, pages

Advantage of combining big//slow and small/fast memories in a **storage hierarchy**.

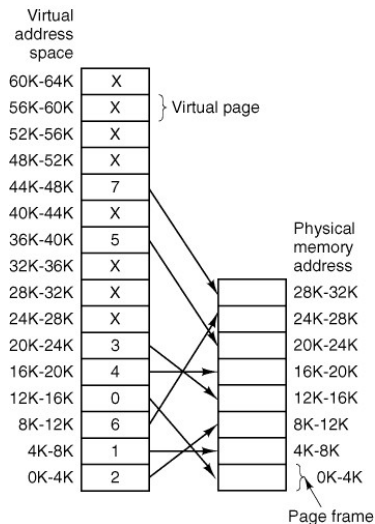


# Memory Management Unit



**Figure:** MMU as translation between logical addresses (i.e., as seen by the process) and physical addresses (as seen on the bus). Used to be a separate chip in ~1980s.

# MMU Translation



**Figure:** The relation between virtual addresses and physical memory addresses is given by the page table. (Tannenbaum fig. 4-8)

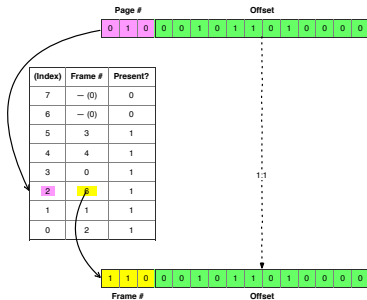
## Page vs. frame

**Page** Group of **logical** addresses

**Frame** Group of **physical** addresses

Both must be the **same size** (today typically 4 kiB), but often greatly **differ in number**.

# Address translation



# Page Tables

## Map virtual pages onto page frames

Main issues:

- 1 The page table can be extremely large.<sup>2</sup> *(How to shrink?)*
- 2 The mapping must be fast. *(How to make fast?)*
- 3 The storage in the MMU is limited. *(How to cache?)*

What data structures can help for (1) and (2)?

Two options

Multilevel Page Tables    Trees (actually, tries)

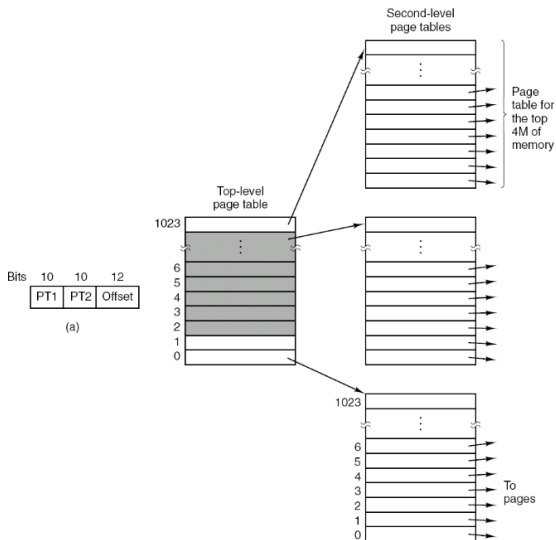
Inverted Page Tables    Hash tables

---

<sup>2</sup>64 bit address space has  $2^{52}$  pages @ 4 kiB (12 address bits).

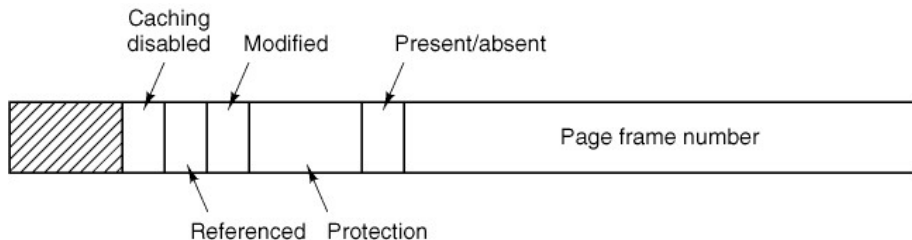


# Multilevel Page Tables



**Figure:** (a) A 32-bit address with two page table fields. (b) Two-level page tables. (Tannenbaum fig. 4-10)

# Page Table Entry



**Figure:** A typical page table entry<sup>3</sup>. (Tannenbaum fig. 4-11)

RW(X) for whom?

---

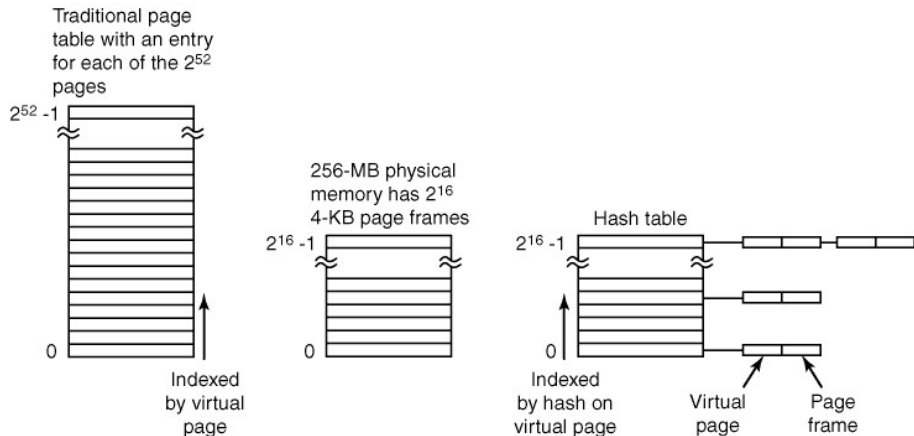
<sup>3</sup>modified bit = dirty bit

# Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	<i>RW</i> –	31
1	20	0	<i>R</i> – <i>X</i>	38
1	130	1	<i>RW</i> –	29
1	129	1	<i>RW</i> –	62
1	19	0	<i>R</i> – <i>X</i>	50
1	21	0	<i>R</i> – <i>X</i>	45
1	860	1	<i>RW</i> –	14
1	861	1	<i>RW</i> –	75

**Table:** A TLB to speed up paging

# Inverted Page Table



**Figure:** Comparison of a traditional page table with an inverted page table.  
(Tannenbaum fig. 4-13)