



# MAC422 Sistemas Operacionais

Prof. Marcel P. Jackowski

Aula #1

**Conceitos, Componentes e Arquiteturas de S.O.**

# Meu primeiro computador: CP-400

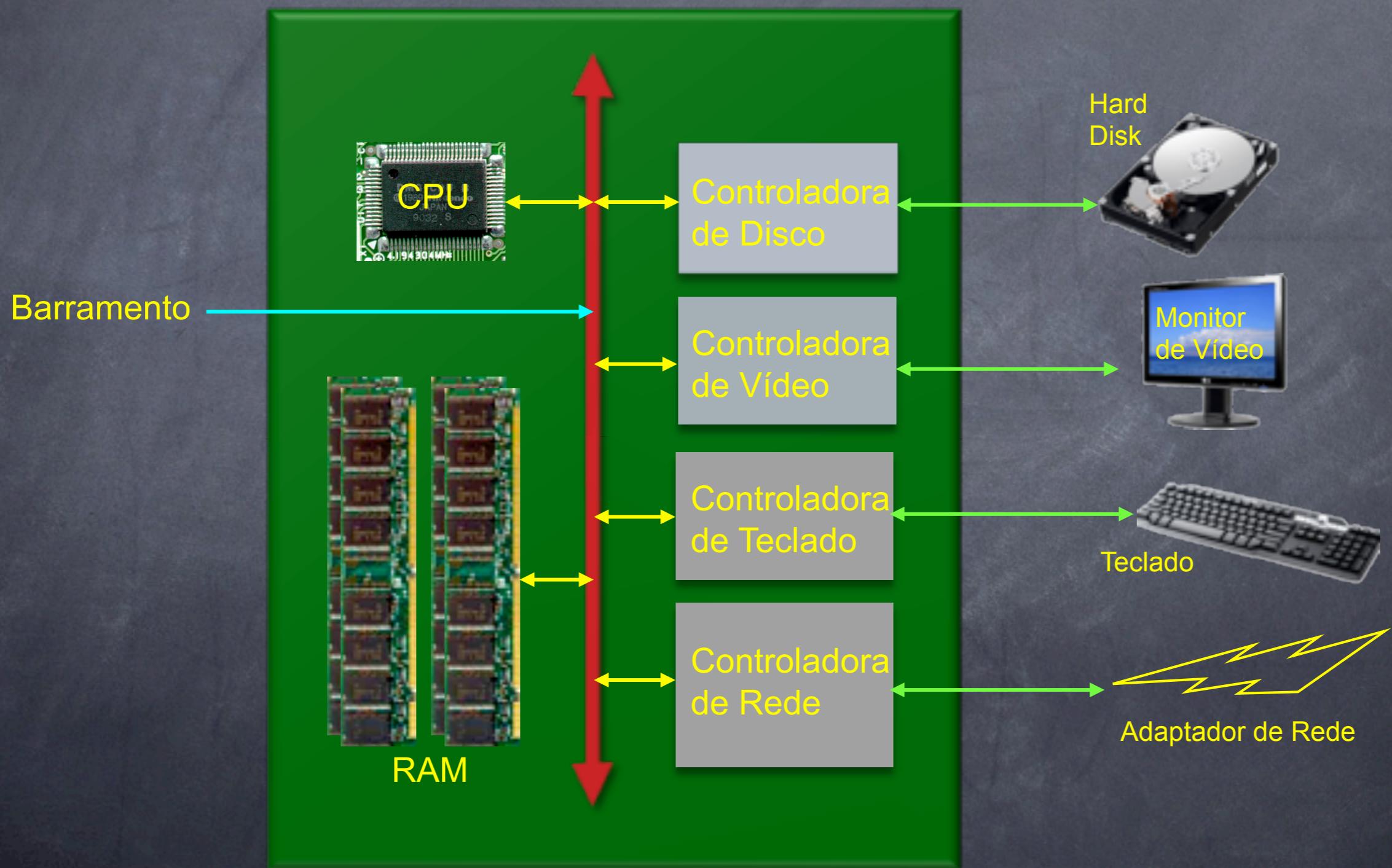
- Processador Motorola 6809E, de 895 KHz
- ROM de 16K.
- 16 ou 64 K de memória RAM
- Modo texto com 32 x 16 caracteres
- Gráficos de baixa resolução, com 64 x 32 pixels
- Gráficos de média e alta resolução (128 x 96, 128 x 192 e 256 x 192)
- Suporte à impressora, gravador de fita cassete e unidade de disco flexível
- Porta de expansão para conectar cartuchos com programas, jogos, etc.
- E/S: serial, cassete, joystick analógico, porta de expansão.
- Cadê o SO ?



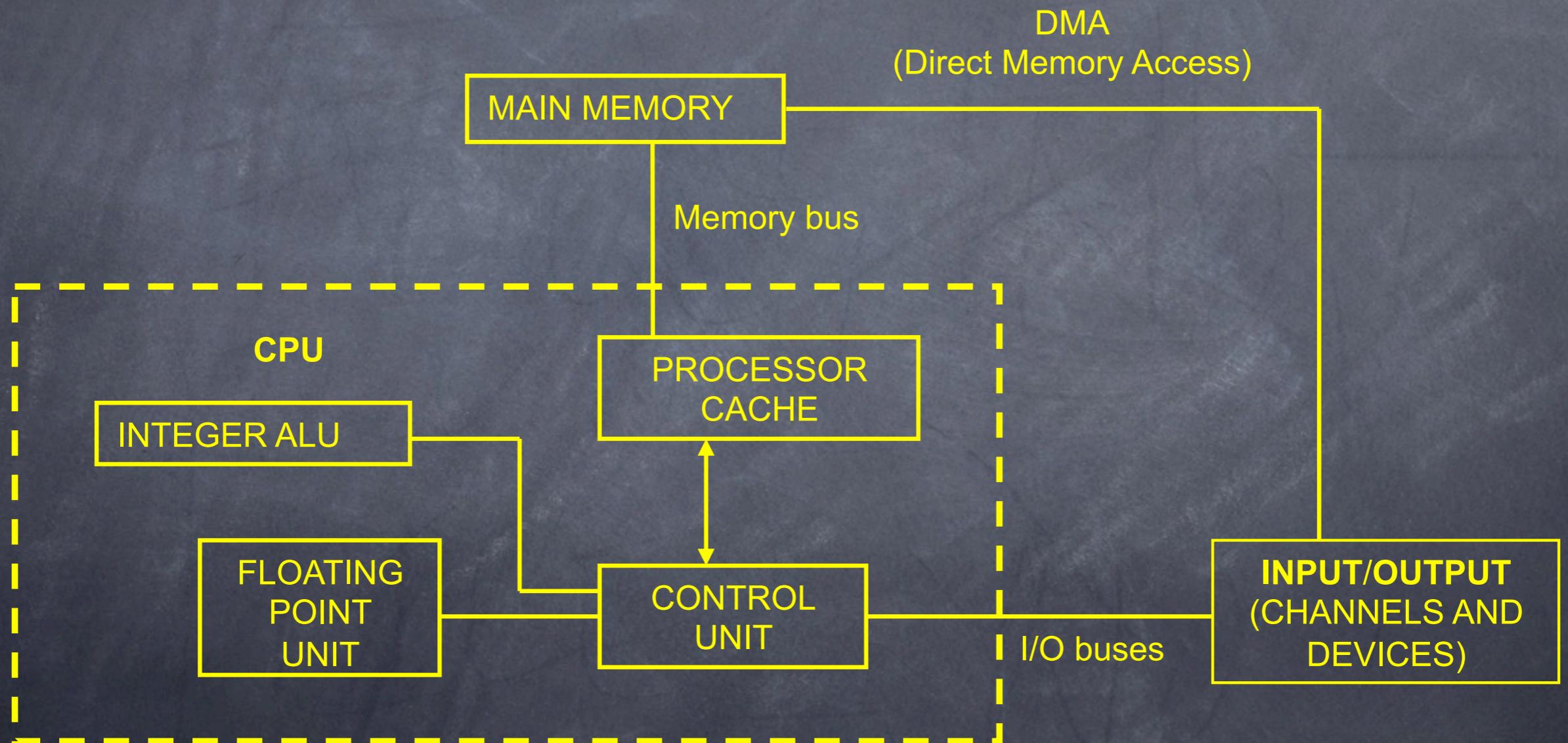
# Classificação

- Computadores pessoais (“**desktops**” e “**notebooks**”)
- Servidores (vários usuários)
- Sistemas embarcados (“**embedded**”)
  - Automóveis, aviões, eletrodomésticos, etc.
- Sistemas móveis
  - PDAs
  - Celulares e smartphones
- O que todos eles têm em comum ?

# Organização principal



# Componentes principais



# Tipos de registradores

- Registradores de instruções
- Contador de programa (PC)
- Registradores de dados (ex. acumulador)
- Registradores de endereços
  - Base, índice
- Registradores de interrupções
- Registradores de estado (indicadores de última comparação, overflow, divisão por zero, etc).
- Registrador de clock.

# CPUs

- Historicamente os processadores eram constituídos de relays, válvulas e transistores.
- Hoje em dia, são integrados em um único circuito integrado (“**chip**”).
- Processadores atuais são dotados de vários núcleos (“**multicore**”):
  - Máquinas multiprocessadas;
  - Atualmente com 2 a 10 núcleos;
  - Em rápida evolução: como aproveitar tais recursos ?

# Início da era dos PCs no Brasil

- Processador: Zilog Z-80A (3,58 MHz)
- Memória ROM: 16 KBytes
- Memória RAM: 16 ou 48 KBytes
- Teclado: tipo "chiclete" com 40 teclas
- Programação: Assembly e BASIC
- Expansão: Interface padrão RS-232
- Joystick: Entrada de 9 pinos padrão Atari 2600
- Texto: 24 linhas x 32 colunas
- Gráfico: 192 x 256 pixels



A Microdigital lança no Brasil o micro pessoal de maior sucesso no mundo.

A partir de agora a história dos micros pessoais vai ser contada em duas partes: antes e depois do TK 90X.

O TK 90X é, simplesmente, o único micro pessoal lançado no Brasil que merece a classificação de "software machine", um caso raro de micro que pela sua facilidade de uso, grandes recursos e preço acessível recebeu a atenção dos criadores de programas e pernéricos em todo o mundo.

E aqui o TK 90X já sai com mais de 100 programas, enquanto outros estão em fase final de desenvolvimento para lhe dar mais opções para trabalhar, aprender ou se divertir que com qualquer outro micro.

O TK 90X tem duas versões de memória (de 16 ou 48 K), imagem de alta resolução gráfica com 8 cores, carregamento rápido de programas (controlável pelo próprio monitor), som pela TV, letras maiúsculas e minúsculas e ainda uma exclusividade: acentuação em português.

Faça o seu programa: peça já uma demonstração do novo TK 90X.

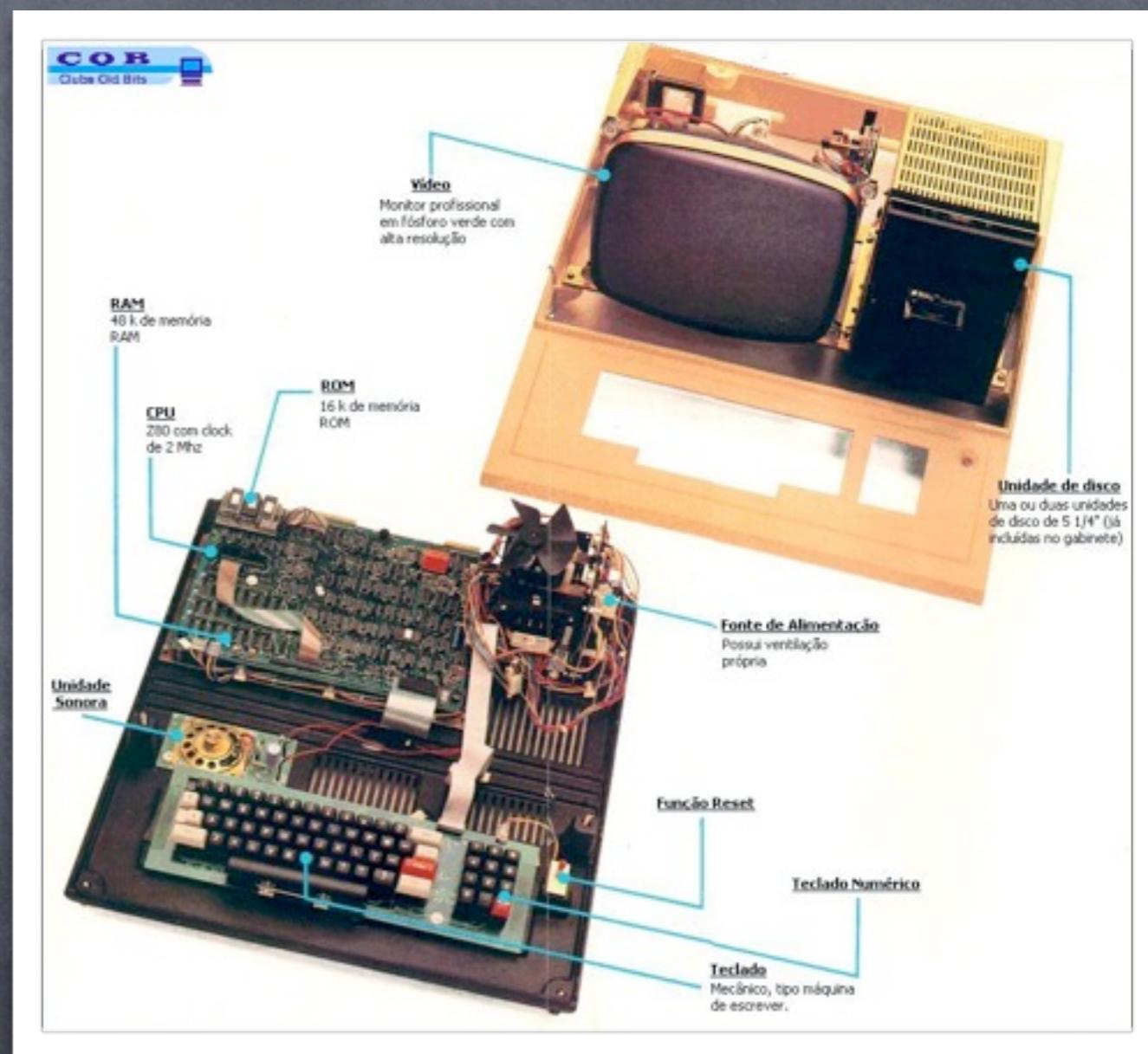
Preço de lançamento:  
16 K - D\$ 1.499,950 • 48 K - D\$ 1.749,950

**MICRODIGITAL**

## Chegou o micro cheio de programas

TK90X

# Início da era dos PCs no Brasil



# Início da era dos PCs no Brasil

- 1 Processador: Zilog Z80A a 3.58 MHz
- 2 ROM: 32 KB
- 3 BIOS (16 KB)
- 4 MSX BASIC V1.0 (16 KB)
- 5 RAM: 8 KB mínimo, a maioria das máquinas tinham 32K ou 64K
- 6 Processador de vídeo: Texas Instruments TMS9918
- 7 RAM de vídeo: 16 KB
- 8 Modos de texto: 40×24 and 32×24
- 9 Resolução: 256×192 (16 cores)
- 10 Chip de áudio: General Instrument AY-3-8910 (PSG)



# Início da era dos PCs no Brasil

- Processador: WDC 65C02 (1MHz)
- Memória ROM: 16KBytes
- Memória RAM: 64KBytes (máximo de 1MByte)
- Teclado: tipo profissional com 77 teclas no padrão QWERTY com teclado numérico reduzido
- SO: Apple DOS 3.3, PRODOS, CP/M (com cartão Z-80)
- Programação: BASIC, Pascal, COBOL, FORTH etc
- Portas de expansão: sete slots para interfaces padrão [[Apple II]] mais um slot auxiliar para expansão de memória
- Joystick: tipo analógico através de entrada padrão DIP, de 16 pinos
- Texto: 24 linhas x 40 colunas (24 linhas x 80 colunas com placa de expansão TK WORKS)
- Gráfico: 192 x 280 (192 x 560 com placa de expansão TK WORKS) pixels
- Portas de cassete: uma entrada e uma saída para gravador de fita cassete



TK 3000 (Apple IIe)

# Algumas características

- PCs antigos tinham poucos recursos comparados com computadores atuais
- Não possuíam um sistema operacional
  - Programa **monitor** em ROM
- Pouco software disponível
  - BASIC
  - Talvez um montador ("assembler").

# Monitor

- Tarefas típicas de um programa monitor:
  - Imprimir caracteres (vídeo)
  - Ler caracteres do teclado
  - Armazenar parte (ou toda) memória em cassete ou disco flexível
  - Restaurar memória do cassete ou disco
  - Enviar caracter à impressora

# Uma chamada ao Monitor

1. Carrega caracter dentro do registrador **EH**
2. Carrega número da função requisitada dentro do registrador **CH**
3. Causa interrupção de software **10h**
4. [Monitor imprime caracter]
5. Carrega status no registrador '**AH**'
6. Retorna ao programa chamador.

# CP/M

- Gary Kildall
- Digital Research Inc.



<http://www.cpm.z80.de/>

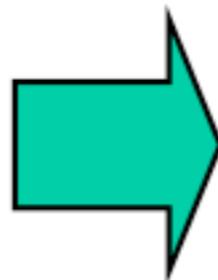
# A crise do software

- Mesmo com a mesma CPU
  - Cada monitor era único
  - Editores e compiladores eram únicos
- Produzir software não era coisa fácil
  - Era necessário mercado, que não época ainda não existia
- Solução: padronizar acesso ao hardware através da BIOS.
  - Surgimento dos primeiros S.O.s

# O que é um SO ?

- “A program that acts as an intermediary between a user of a computer and the computer hardware.”

“stuff between”



# Duas perspectivas

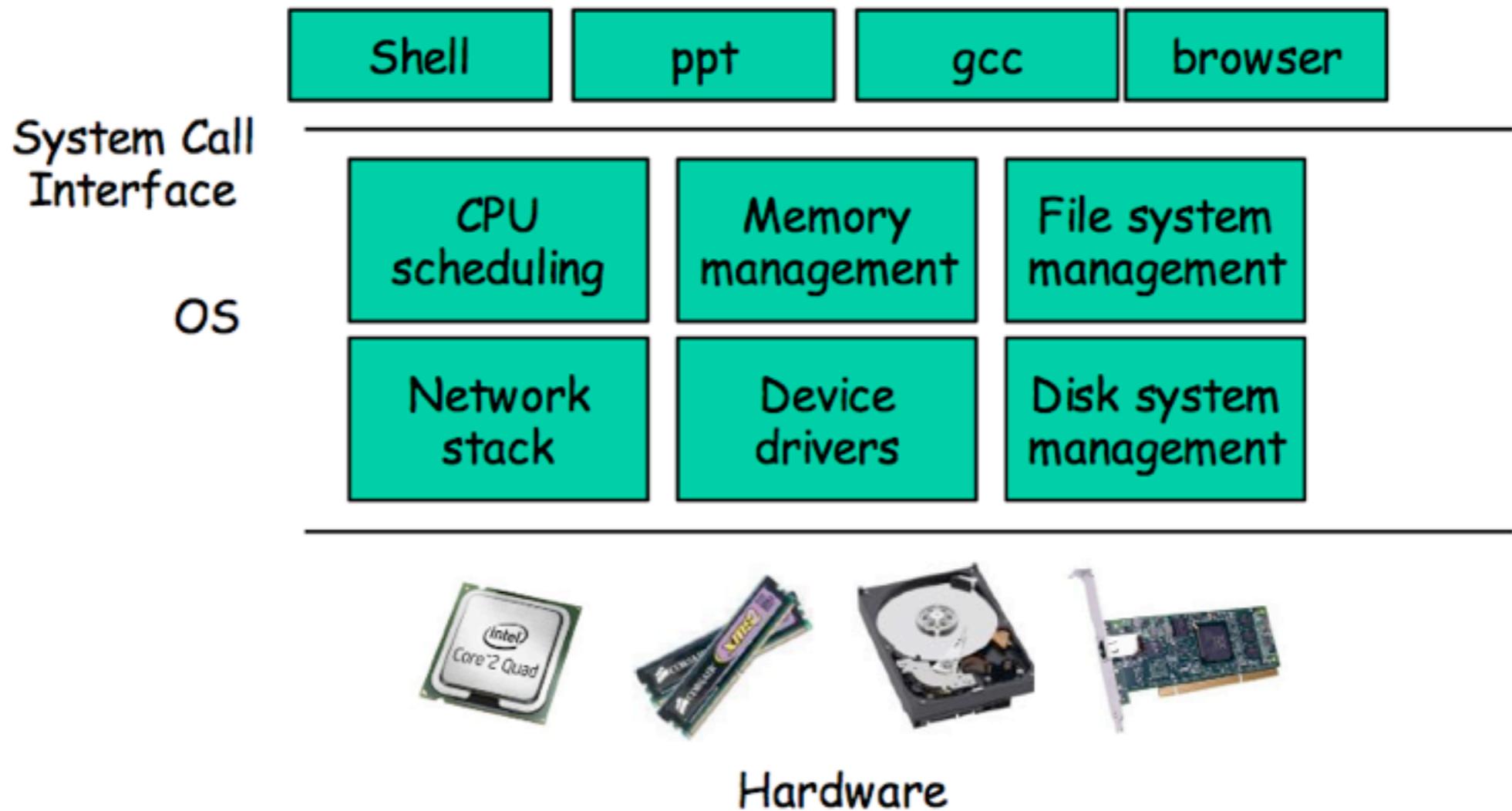
- Top-down perspective: **hardware abstraction layer**, turn hardware into something that applications can use
  
- Bottom-up perspective: **resource manager/coordinator**, manage your computer's resources

# SO: Abstração do hardware

- ❑ “standard library” “OS as virtual machine”
  - E.g. `printf("hello world")`, shows up on screen
  - App issue **system calls** to use OS abstractions
- ❑ Why good?
  - Ease of use: higher level, easier to program
  - Reusability: provide common functionality for reuse
    - E.g. each app doesn't have to write a graphics driver
  - Portability / Uniformity: stable, consistent interface, different OS/ver/hw look same
    - E.g. scsi/ide/flash disks
- ❑ Why hard?
  - What are the **right** abstractions ?

# SO: Coordenador de recursos

- Computer has resources, OS must manage.
  - Resource = CPU, Memory, disk, device, bandwidth, ...



# SO: Coordenador de recursos

## ❑ Why good?

- **Sharing/Multiplexing:** more than 1 app/user to use resource
- **Protection:** protect apps from each other, OS from app
  - Who gets what when
- **Performance:** efficient/fair access to resources

## ❑ Why hard? Mechanisms vs policies

- **Mechanism:** how to do things
- **Policy:** what will be done
- **Ideal:** general mechanisms, flexible policies
  - Difficult to design right

# Abstração do SO: processo

- Running program, stream of running instructions + process state
  - A key OS abstraction: the applications you use are built of processes
    - Shell, powerpoint, gcc, browser, ...
- Easy to use
  - Processes are protected from each other
    - process = address space
  - Hide details of CPU, when&where to run

# System calls relacionadas

- **int fork (void)**
  - Create a copy of the invoking process
  - Return process ID of new process in "parent"
  - Return 0 in "child"
- **int execv (const char\* prog, const char\* argv[])**
  - Replace current process with a new one
  - prog: program to run
  - argv: arguments to pass to main()
- **int wait (int \*status)**
  - wait for a child to exit

# Shell simples

```
// parse user-typed command line into command and args  
...  
  
// execute the command  
switch(pid = fork ()) {  
    case -1: perror ("fork"); break;  
    case 0: // child  
        execv (command, args, 0); break;  
    default: // parent  
        wait (0); break; // wait for child to terminate  
}
```

# Abstração de SO: arquivo

- Array of bytes, persistent across reboot
  - Nice, clean way to read and write data
  - Hide the details of disk devices (hard disk, CDROM, flash ...)
- Related abstraction: **directory**, collection of file entries

# System calls relacionadas

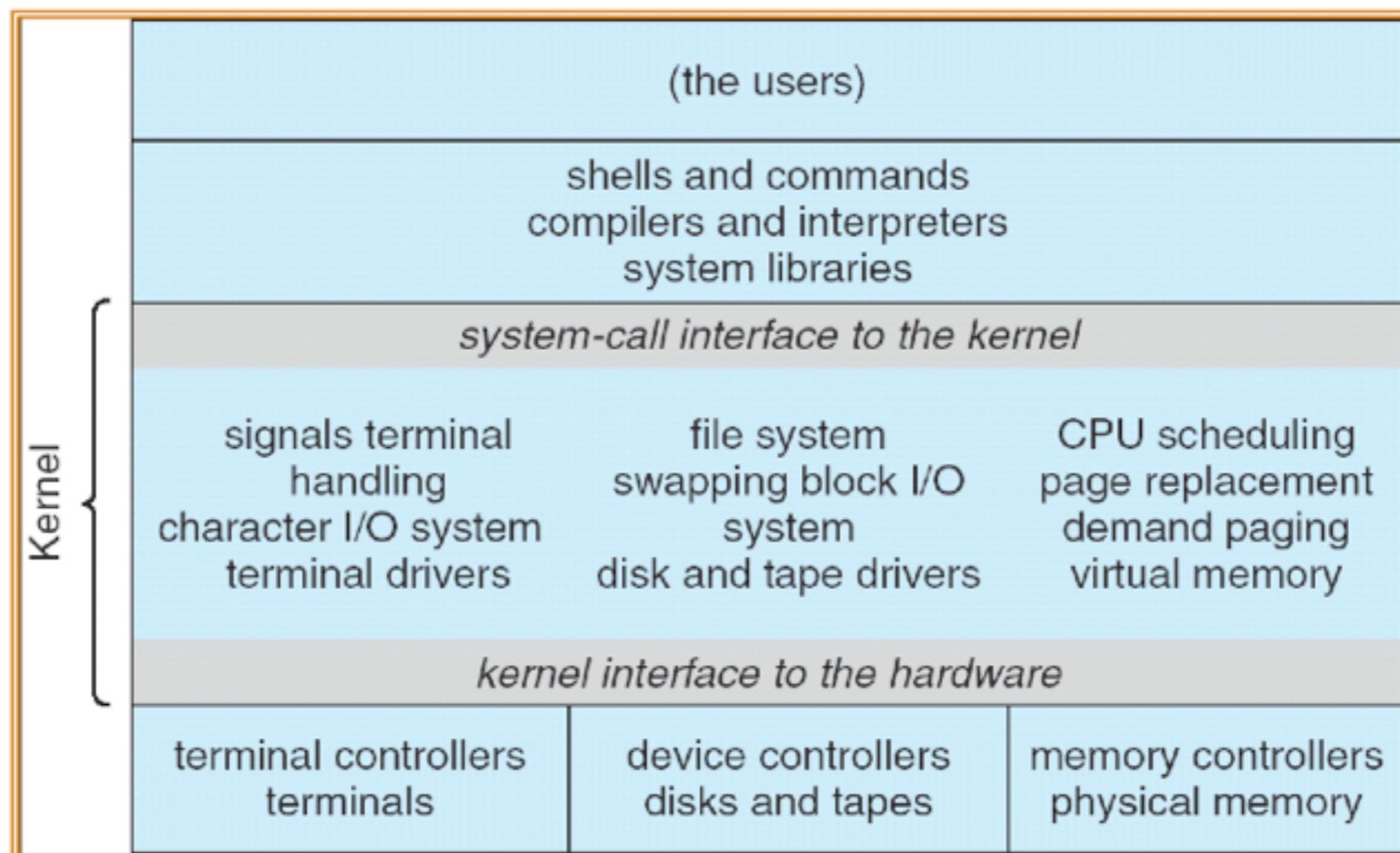
- ❑ `int open(const char *path, int flags, int mode)`
  - Opens a file and returns an integer called a **file descriptor** to use in other file system calls
  - Default file descriptors
    - 0 = `stdin`, 1 = `stdout`, 2 = `stderr`
- ❑ `int write(int fd, const char* buf, size_t sz)`
  - Writes `sz` bytes of data in `buf` to `fd` at current file offset
  - Advance file offset by `sz`
- ❑ `int close(int fd)`
- ❑ `int dup2 (int oldfd, int newfd)`
  - makes `newfd` an exact copy of `oldfd`
  - closes `newfd` if it was open
  - two file descriptors will share same offset

# Estrutura de um SO

- OS structure: what goes into the **kernel**?
  - Kernel: most interesting part of OS
    - Privileged; can do everything → must be careful
    - Manages other parts of OS
- Different structures lead to different
  - Performance, functionality, ease of use, security, reliability, portability, extensibility, cost, ...
- Tradeoffs depend on technology and workload

# Monolítico

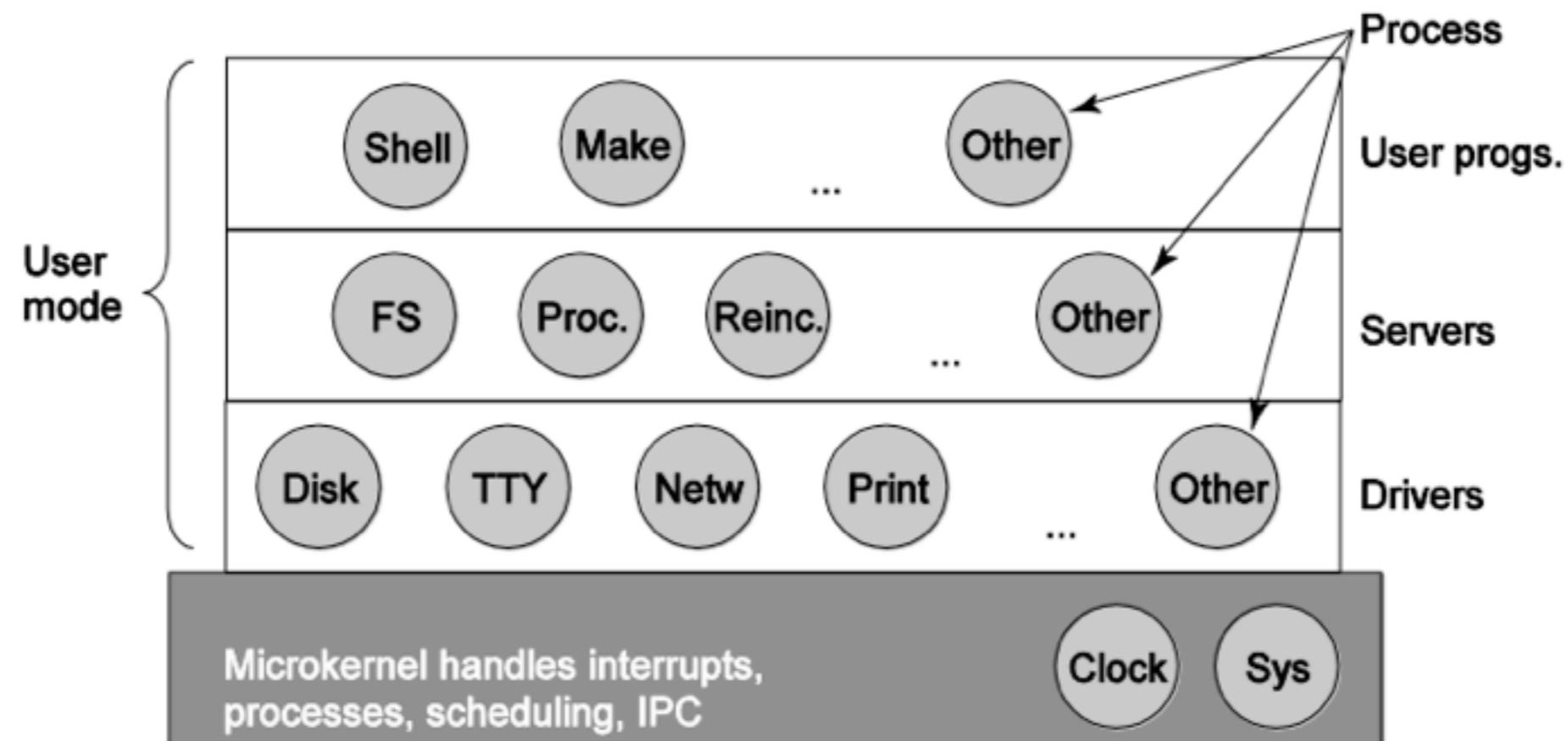
## □ Most traditional functionality in kernel



Unix System Architecture

# Microkernel

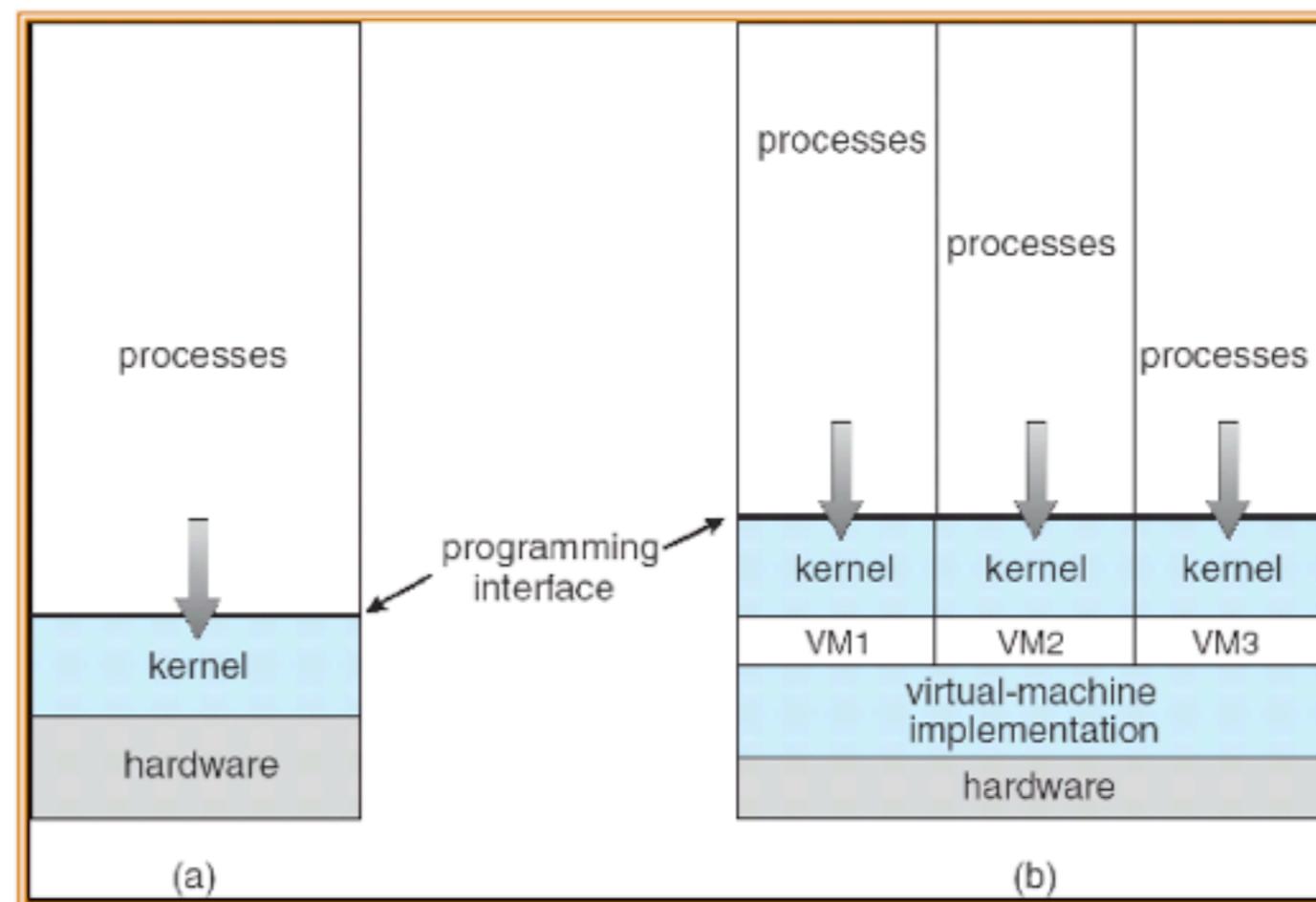
- Move functionality out of kernel



Minix 3 System Architecture

# Máquina virtual

- Export a fake hardware interface so that multiple OS can run on top



Non-virtual Machine

Virtual Machine

# Evolução de SO

- Many outside factors affect OS
- User needs + technology changes → OS must evolve
  - New/better abstractions to users
  - New/better algorithms to implement abstractions
  - New/better low-level implementations (hw change)
- Current OS: evolution of these things

# Tendência

- Hardware: cheaper and cheaper
- Computers/user: increases
- Timeline
  - 70s: mainframe, 1 / organization
  - 80s: minicomputer, 1 / group
  - 90s: PC, 1 / user

# Anos 70: mainframe

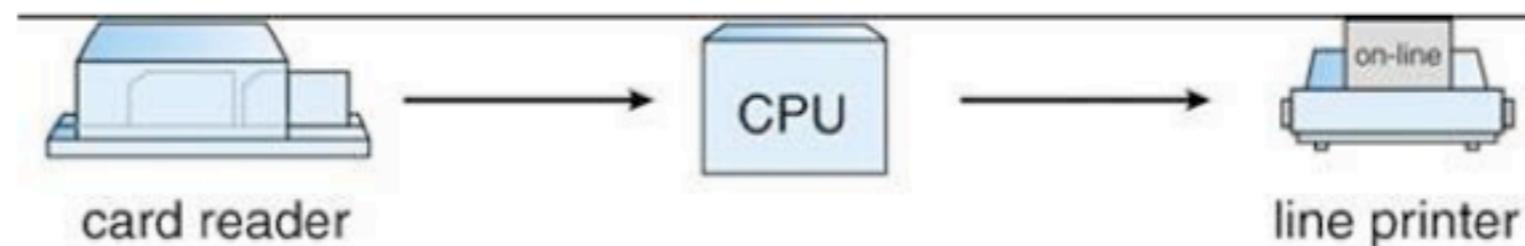
- **Hardware:**
  - Huge, \$\$\$, slow
  - IO: punch card, line printer
- **OS**
  - simple library of device drivers (no resource coordination)
  - Human = OS: single programmer/operator programs, runs, debugs
  - One job at a time
- **Problem:** poor performance (utilization / throughput)  
Machine \$\$\$, but idle most of the time because  
programmer slow

# Processamento em lote ("batch")

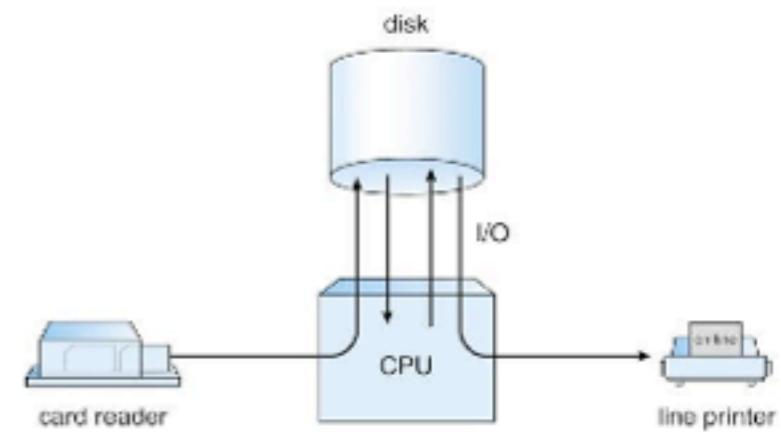
- Batch: submit group of jobs together to machine
  - Operator collects, **orders**, runs (resource coordinator)
- Why good? can better optimize given more jobs
  - Cover setup overhead
  - Operator quite skilled at using machine
  - Machine busy more (programmers debugging offline)
- Why bad?
  - Must wait for results for long time
- Result: utilization increases, interactivity drops

# “Spooling”

- **Problem:** slow I/O ties up fast CPU
  - Input → Compute → Output
  - Slow punch card reader and line printer

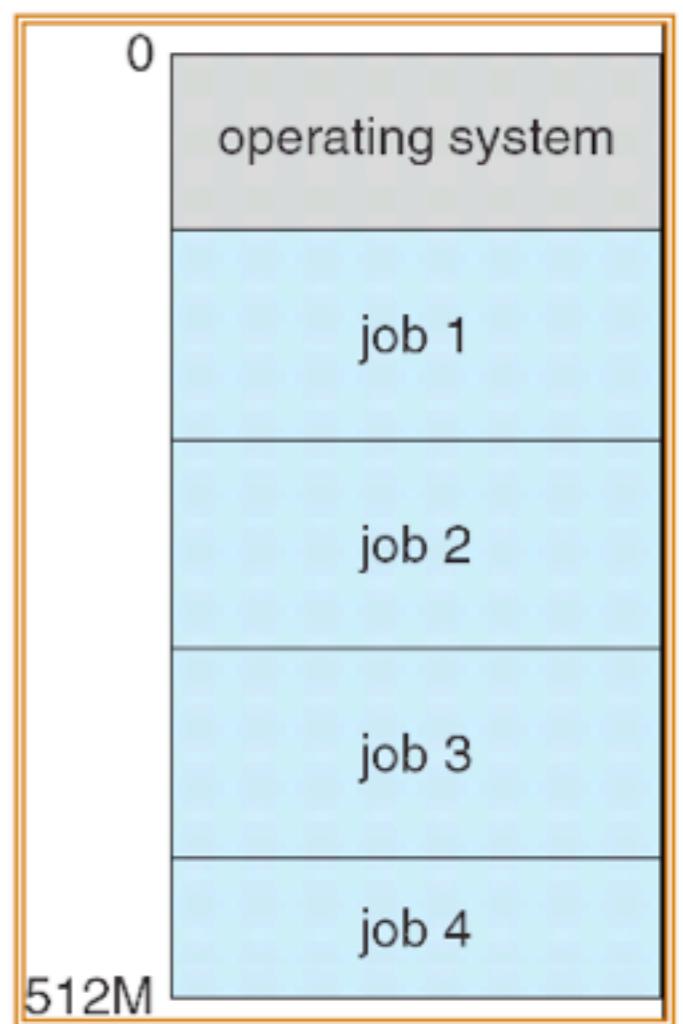


- **Idea:** overlap one job's IO with other jobs' compute
- **OS functionality**
  - buffering, DMA, interrupts
- **Good:** better utilization/throughput
- **Bad:** still not interactive



# Multiprogramação

- Spooling → multiple jobs
- Multiprogramming
  - keep multiple jobs in memory, OS chooses which to run
  - When job waits for I/O, switch
- OS functionality
  - job scheduling, mechanism/policies
  - Memory management/protection
- Good: better throughput
- Bad: still not interactive



# Anos 80: minicomputador

- Hardware gets cheaper. 1 / group
- Need better interactivity, short response time
- Concept: timesharing
  - Fast switch between jobs to give impression of dedicated machine
- OS functionality:
  - More complex scheduling, memory management
  - Concurrency control, synchronization
- Good: immediate feedback to users

# Anos 90: PC

- ❑ Even cheaper. 1 / user
- ❑ Goal: easy of use, more responsive
- ❑ Do not need a lot of stuff
- ❑ Example: DOS
  - No time-sharing, multiprogramming, protection, VM
  - One job at a time
  - OS is subroutine again
- ❑ Users + Hardware → OS functionality

# Tendências atuais

- Large
  - Users want more features
  - More devices
  - Parallel hardware
  - Result: large system, millions of lines of code
- Reliability, Security
  - Few errors in code, can recover from failures
  - At odds with previous trend
- Small: e.g. handheld device
  - New user interface
  - Energy: battery life
  - One job at a time. OS is subroutine again

# Tarefa de casa

- Leitura “Sistemas Operacionais Modernos”
  - Capítulo 1
- EP #1
- xv6 e QEMU