Prof. Ricardo Inácio Álvares e Silva

Sistemas Operacionais

Revisão de Hardware

Lorem Ipsum Dolor

Escopo da aula

- Processadores
  - Tecnologias envolvidas
  - Assembly MIPS básico
- Memórias
  - Hierarquia
  - Diversos tipos de memórias
- Dispositivos E/S
  - Exemplos
  - Drivers

Processadores

- Funcionamento
  - Lê palavra da memória – Decodifica instrução
  - Executa

- Registradores
  - Propósito geral
  - Contador de programa
  - Ponteiro da pilha
  - PSW (Program Status Word)

Processador básico

Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Valor armazenado	i0	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10	i11	i12	i13

Memória Principal

- CISC - Complex Instruction Set Computer
  - Feito para programadores humanos, em assembly
  - Muitas funcionalidades complicam o projeto da CPU

- RISC - Reduced Instruction Set Computer
  - Feito para programação de alto nível, linguagens compiladas
  - Projeto mais simples possibilitou novas técnicas
    - Pipeline
      - "Uma instrução por ciclo"
    - Superscalar
      - Vários pipelines em um único núcleo

Execução sem Pipeline

Tempos	T1	T2	T3	T4	T5	T6	T7	T8
Próxima	i1	-	-	-	i2	-	-	-
Decodific	-	i1	-	-	-	i2	-	-
Execução	-	-	i1	-	-	-	i2	-
Armazen	-	-	-	i1	-	-	-	i2

Execução com Pipeline

Tempos	T1	T2	T3	T4	T5	T6	T7	T8
Próxima	i1	i2	i3	i4	i5	i6	i7	i8
Decodific	-	i1	i2	i3	i4	i5	i6	i7
Execução	-	-	i1	i2	i3	i4	i5	i6
Armazen	-	-	-	i1	i2	i3	i4	i5

- Modos de operação
  - Controlado por registrado especial - PSW
  - Usuário
    - Não pode acessar toda a memória
    - Nem utilizar todas as instruções
  - Supervisor (ou Kernel)
    - Acesso completo
    - Destinado somente ao Sistema Operacional

- Lei de Moore
  - Tamanho se reduz pela metade a cada 1 ano e meio
  - Desempenho é melhorado à medida que sobra espaço

- Multithreading (Hyperthreading)
  - Múltiplas threads por núcleo

- Multicore
  - Um mesmo chip possui vários processadores

- Sistema operacional precisa diferenciar entre Multithreading e Multicore
  - Afeta o desempenho, por causa da memória cache e processos

Linguagem montadora

- O circuitos eletrônicos do computador só entendem variações no estado de correntes elétricas
  - Mais utilizada é variação binária: `ausente - presente`
- Para facilitar compreensão humana, é expressa em números
  - Binária: 0 (ausente) - 1 (presente)
- Instruções para o processador são variações em uma combinação de correntes
  - Ex: `ausente - presente - ausente - ausente`, ou 0100

- **Linguagem de máquina** são as possíveis combinações de estados de correntes que fazem o processador operar
  - é expressa pelos valores números dos estados das correntes combinadas
- Exemplos (fictícios):
  - 0010 -> soma registrador A com registrador B
  - 1000 -> PC recebe o valor de A, ou seja, um desvio

- Os primeiros computadores eram programados em linguagem de máquina
  - A dificuldade tornava contraproducente
  - Processadores mais complexos eram impossíveis de serem operados
- A solução foi elaborar uma linguagem mnemônica (procure no google), chamada de **linguagem montadora**
- Ao invés do programador chamar 0100 para somar, ele indica `add A, B` e um programa traduz automaticamente

- Por estar intimamente ligada a linguagem de máquina, cada processador tem sua própria linguagem montadora
- Na nossa disciplina, vamos considerar as instruções do MIPS
  - Primeiro processador RISC e com pipelines
  - Utilizado em supercomputadores e roteadores de redes

Registradores do MIPS

Nome	Descrição
$zero	Registrador que sempre contém o valor **zero**
$v0 - $v1	Utilizados para retornar **valores** de rotinas (funções e métodos)
$a0 - $a3	Utilizados para passar **argumentos** para funções
$t0 - $t9	Registradores **temporários**, que podem ser apagados em alguma rotina chamada
$s0 - $s7	Registradores **salvos**, nunca são apagados por uma rotina chamada
$sp	(**Stack pointer**), aponta para o topo da pilha na memória
$ra	(**Return address**), armazena o valor de retorno de uma subrotina para o PC da rotina que a chamou
$pc	(**Program counter**), armazena o valor da próxima instrução a ser buscada na memória

- Estes registradores são denominados de propósito geral, e podem ser utilizados por qualquer uma das instruções
  - Lógicas
  - Aritméticas
  - Leitura/Escrita de memória
  - Desvios na linha de execução

Principais instruções

- *lw rd, src*
  - Load Word - carrega uma palavra de 32 bits da memória
- Exemplos:

```
# Valores constantes e strings do programa
.data
CARTEIRA:
   .word 300
SALARIO:
   .word 1000

# Instruções do programa vão abaixo de .text
.text
   lw $t0, CARTEIRA
   lw $t1, SALARIO
```

Pricipais instruções

- *li rd, src*
  - Load Immediate - carrega um valor expresso direto na instrução
- Exemplo:

```
# Valores constantes e strings do programa
.data
CARTEIRA:
   .word 300

# Instruções do programa vão abaixo de .text
.text
   lw $t0, CARTEIRA
   li $t1, 1000          # Salario
```

Principais instruções

- *add rd, rs, rt*
  - Add - soma *rs* com *rt* e guarda em *rd*
- Exemplo:

```
# Valores constantes e strings do programa
.data
CARTEIRA:
   .word 300

# Instruções do programa vão abaixo de .text
.text
   lw $t0, CARTEIRA
   li $t1, 1000          # Salario
   add $t2, $t0, $t1
```

Principais instruções

- *sub rd, rs, rt*
  - Subtract - subtrai *rt* de *rs* e guarda em *rd*

```
# Valores constantes e strings do programa
.data
CARTEIRA:
   .word 300

# Instruções do programa vão abaixo de .text
.text
   lw $t0, CARTEIRA
   li $t1, 1000          # Salario
   add $s0, $t0, $t1
   li $t0, 447           # Valor dos impostos
   sub $s0, $s0, $t0
```

Principais instruções

- *sw rs, dst*
  - Store Word - armazena valor de *rs* na memória

```
# Valores constantes e strings do programa
.data
CARTEIRA:
    .word 300

# Instruções do programa vão abaixo de .text
.text
    lw $t0, CARTEIRA
    li $t1, 1000          # Salario
    add $s0, $t0, $t1
    li $t0, 447           # Valor dos impostos
    sub $s0, $s0, $t0
    sw $s0, CARTEIRA
```

Principais instruções

- *j target*
  - Jump - registrador *$pc* recebe valor de *target*

```
# Valores constantes e strings do programa
.data
CARTEIRA:
    .word 300

# Instruções do programa vão abaixo de .text
.text
COMPUTA_SALARIO:
    lw $t0, CARTEIRA
    li $t1, 1000          # Salario
    add $s0, $t0, $t1
    li $t0, 447           # Valor dos impostos
    sub $s0, $s0, $t0
    sw $s0, CARTEIRA
    j COMPUTA_SALARIO
```

Principais instruções

- *bne rs, rt, target*
  - Branch if Not Equal - *$pc* recebe valor de target se *rs != rt*

```
.data
CARTEIRA:
    .word 300

.text
    li $a0, 12              # Meses de cálculo
COMPUTA_SALARIO:
    lw $t0, CARTEIRA
    li $t1, 1000            # Salario
    add $s0, $t0, $t1
    li $t0, 447             # Valor dos impostos
    sub $s0, $s0, $t0
    sw $s0, CARTEIRA

    addi $a0, $a0, -1
    bne $a0, $zero, COMPUTA_SALARIO
```

Principais instruções

- Outras instruções de instruções de desvio:
  - *beq rs, rt, target* - Branch if Equal
  - *blt rs, rt, target* - Branch if Less Than
  - *ble rs, rt, target* - Branch if Less or Equal
  - *bgt rs, rt, target* - Branch if Greater Than
  - *bge rs, rt, target* - Branch if Greater or Equal

Principais instruções

- Chamando uma rotina:
  - *jal target*
    - Jump And Link
      - armazena o valor de *$pc* em *$ra*
      - copia o valor de *target* para *$pc*

- Retornando de uma rotina:
  - *jr rt*
    - Jump to Register - *$pc* recebe o valor de *rt*
    - Para retornar, *jr $ra*

```
.data
CARTEIRA:
    .word 300

.text
    lw $a0, CARTEIRA
    li $a1, 12              # Meses de cálculo
    li $a2, 1000            # Valor do salário
    li $a3, 437             # Imposto mensal bruto
    jal COMPUTA_SALARIO # Chama a rotina do salario
    sw $v0, CARTEIRA

# Nossa rotina de salários
COMPUTA_SALARIO:
    add $a0, $a0, $a2   # Soma salario com carteira
    sub $a0, $a0, $a3   # Subtrai os impostos
    addi $a1, $a1, -1   # decrementa um mês
    bne $a1, $zero, COMPUTA_SALARIO
    add $v0, $a0, $zero # Copia o resultado para retorno
    jr $ra              # Retorna após o "jal"
```

Chamadas ao sistema

- Suponha que queiramos escrever o resultado do salário na tela
- Quem faz isso é o sistema operacional, que tem uma rotina específica para essa funcionalidade
- Bastaria o processador chamar a rotina
- Porém, rotinas do SO são restritas ao kernel (superusuário), inacessíveis ao nosso programa
- Para chamá-la, utilizamos uma instrução que passa o controle para o SO, ao mesmo tempo que indica o que gostaríamos que ele fizesse (ele não é obrigado a obedecer)

- Esse tipo de rotina é conhecida como **chamadas ao sistema** (*system calls*)
- Elas são invocadas através de instruções que causam uma **interrupção** (*interruption*) no processador
- Cada sistema operacional responde às interrupções e chamadas ao sistema de maneira diferente
  - Imagine um programa em linguagem de máquina do *Core i5,* com chamadas ao sistema do *Linux*
  - Rodará em um computador com o mesmo *Core i5* e *Windows*? Por quê?

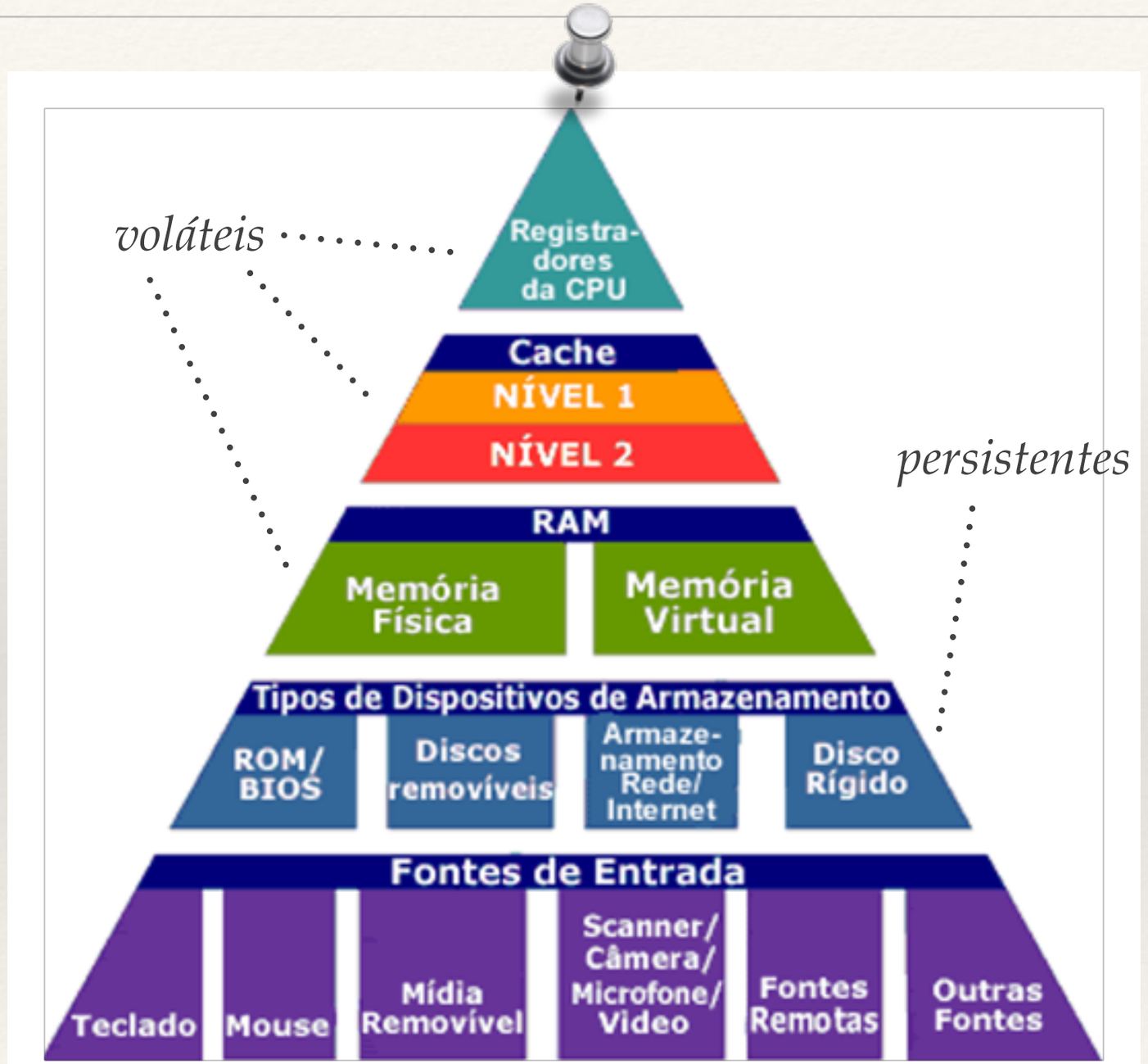
- No MIPS, a instrução de interrupção é:
  - *syscall*
    - não possui nenhum parâmetro direto na instrução
    - Cada SO utiliza sua própria forma para descobrir porque foi feito aquele *syscall* pelo programa do usuário
- No Sistema Operacional do MARS (nosso *simulador)*
  - *$v0* é usado para indicar qual serviço o usuário deseja
  - cada serviço utiliza alguns registradores para receber parâmetros, têm que consultar no manual

```
.data
CARTEIRA:
    .word 300
STR_SALARIO:
    .asciiz "O salario calculado foi de: "

.text
    lw $a0, CARTEIRA
    li $a1, 12              # Meses de cálculo
    li $a2, 1000            # Valor do salário
    li $a3, 437             # Imposto mensal bruto
    jal COMPUTA_SALARIO # Chama a rotina do salario
    sw $v0, CARTEIRA

    li $v0, 4               # Nº serviço escrita de string
    la $a0, STR_SALARIO # Endereço da frase a escrever
    syscall

    li $v0, 1               # Serviço de escrita de inteiros
    lw $a0, CARTEIRA
    syscall

    li $v0, 10              # Serviço de fim de programa
    syscall

# Nossa rotina de salários
COMPUTA_SALARIO:
    add $a0, $a0, $a2   # Soma salario com carteira
    sub $a0, $a0, $a3   # Subtrai os impostos
    addi $a1, $a1, -1   # decrementa um mês
    bne $a1, $zero, COMPUTA_SALARIO
    add $v0, $a0, $zero # Copia o resultado para retorno
    jr $ra              # Retorna após o "jal"
```

Memória

desempenho

Preço



disponibilidade

- Registradores
  - Dentro do processador
  - Pouquíssima disponibilidade - entre 16 a 64 palavras
  - Acesso direto para o programador
- Cache
  - Até 3 níveis: L1, L2, L3
  - Acesso varia
    - Controle automático pelo processador
    - Ajuda do Sistema Operacional
  - Erro e acerto de cache

- Memória principal
  - Maior quantidade
  - Desempenho médio (10x inferior)
  - Volátil
  - Acesso aleatório
    - Com memória virtual, acesso intermediado por hardware e SO

- ROM, EEPROM, Flash e CMOS
  - Persistentes
  - Algumas tem limitações em rescritas

Discos

- Ainda mais barata
  - Desempenho pior (100x em relação a RAM)
  - Dados persistem
- Componentes
  - Braços, trilhas e cilindros
  - Acessos aleatórios possíveis, porém prejudicam desempenho

- Memória virtual
  - Estende e protege a memória principal
  - Depende da MMU (Memory Management Unit)
- Implementação Hardware - Software (SO)
- Melhora o desempenho em ambientes multiprogramados

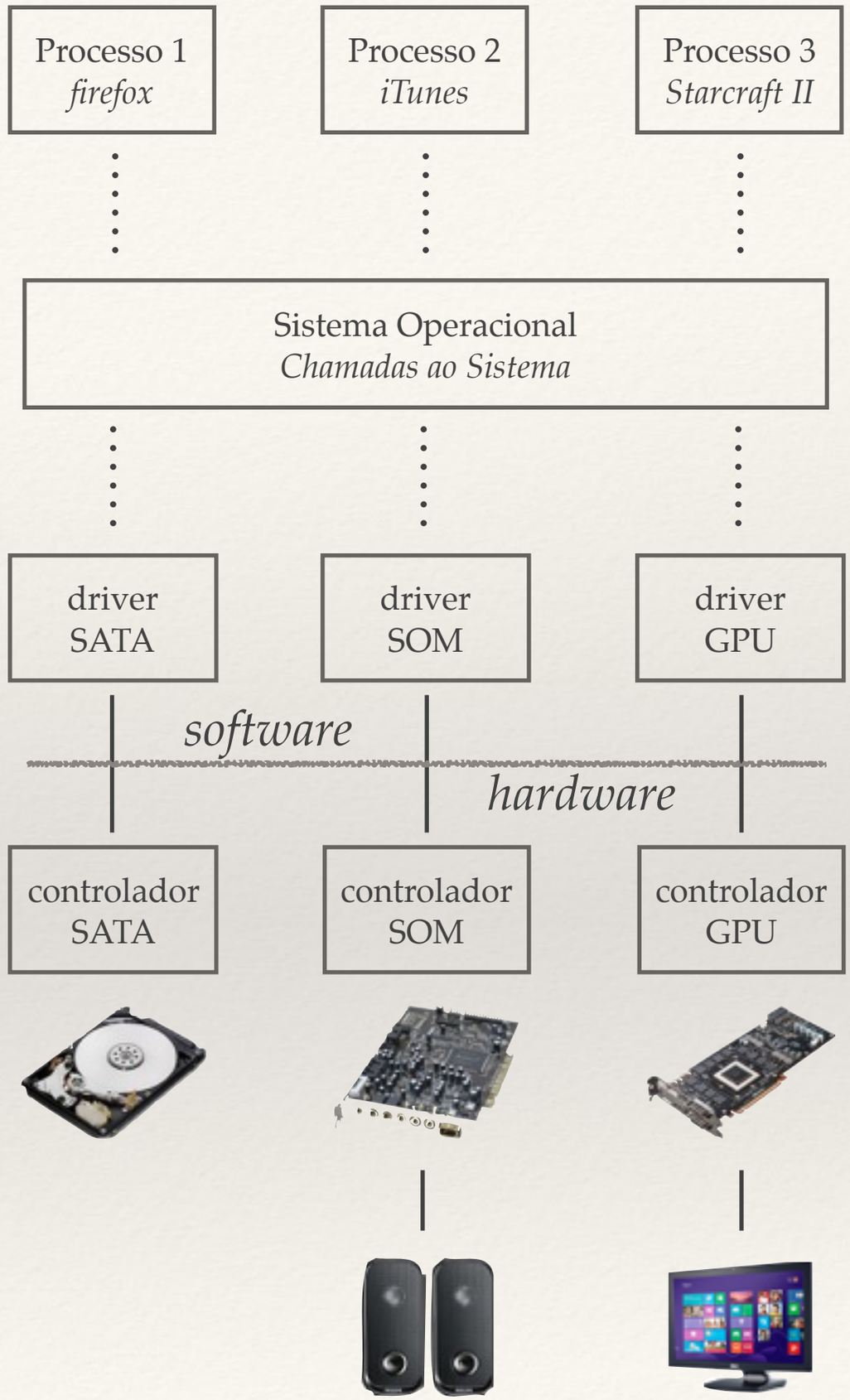
Fitas magnéticas

- Mais barata e durável
- Uma gravação dura de minutos a horas
- Acesso sequencial
- Ideal para backups, ainda é utilizada em servidores

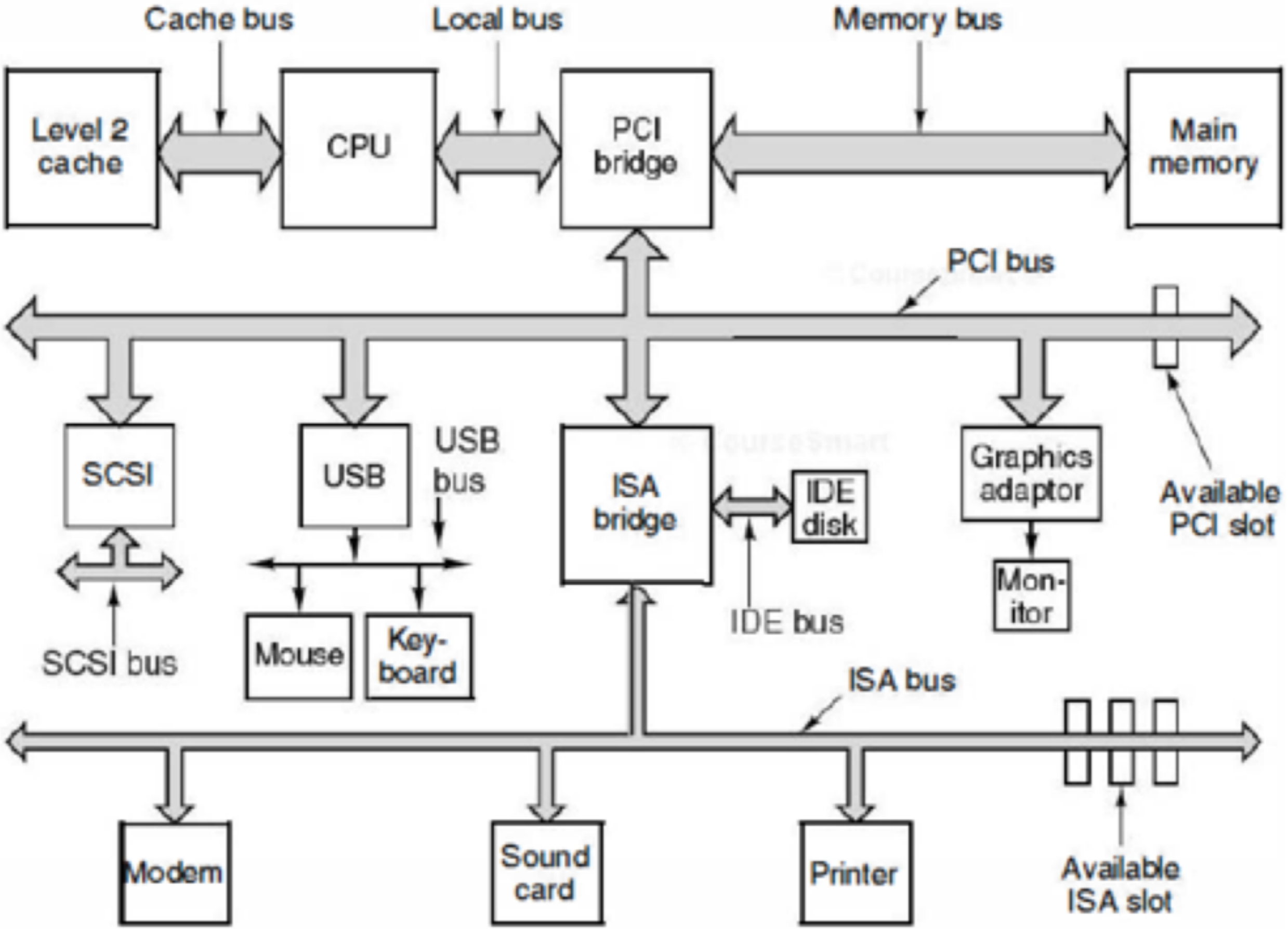
- `tarballs` - serializa diversos arquivos para gravar em fita

Dispositivos E/S

- Possuem controlador próprio
  - Operam o dispositivo
  - Exemplos:
    - IDE
    - SCSI
    - Rede
    - Vídeo



- SO comunica com os controladores
  - Driver de dispositivo
  - Registradores dos controladores
  - Endereço E/S
    - Instruções IN e OUT
    - Mapeamento no espaço de memória
- Métodos
  - Pooling
  - Interrupção
  - DMA (Direct Memory Access)


Cache bus
Local bus
Memory bus
Level 2 cache
CPU
PCI bridge
Main memory
PCI bus
SCSI
USB
USB bus
ISA bridge
IDE disk
Graphics adaptor
Available PCI slot
SCSI bus
Mouse
Key-board
IDE bus
Mon-itor
ISA bus
Modem
Sound card
Printer
Available ISA slot