

O Núcleo do Sistema de Operação

(1)

José C. Cunha, DI/FCT/UNL

Referências:

Modern Operating Systems, A. Tanenbaum
Sistemas Operativos, J. Alves Marques et al.
Sistemas de Exploração, J. Cardoso e Cunha

O Sistema de Operação visto como...

● Gestor de Recursos

- Gere e permite a partilha do *hardware* pelos processos
- Gere a execução dos programas e as suas interacções
- Garante a protecção e a consistência no acesso aos recursos (processamento, memória, comunicação)

● Máquina Virtual

- Oferece abstracções de mais fácil utilização
- Oferece operações de mais "alto nível" que suportam o resto do sistema e que são chamadas pelos programas utilizadores

Principais grupos de operações

- Controlo da execução de programas

- Iniciar, gerir a memória, detectar erros, terminar, etc..

Os *processos e os threads*

- Concorrência entre processos

- Os problemas de escalonamento e de sincronização
- Múltiplas aplicações e utilizadores

- Operações de entrada/saída de dados (ou I/O) e arquivo

- Comunicar com os periféricos...
- O acesso à informação em disco

Os *canais de comunicação e os ficheiros*

- Comunicação entre processos concorrentes

- A transmissão de informação (mensagens)
- O acesso à informação partilhada e a sincronização

SO: abordagem tradicional

Application
Programs

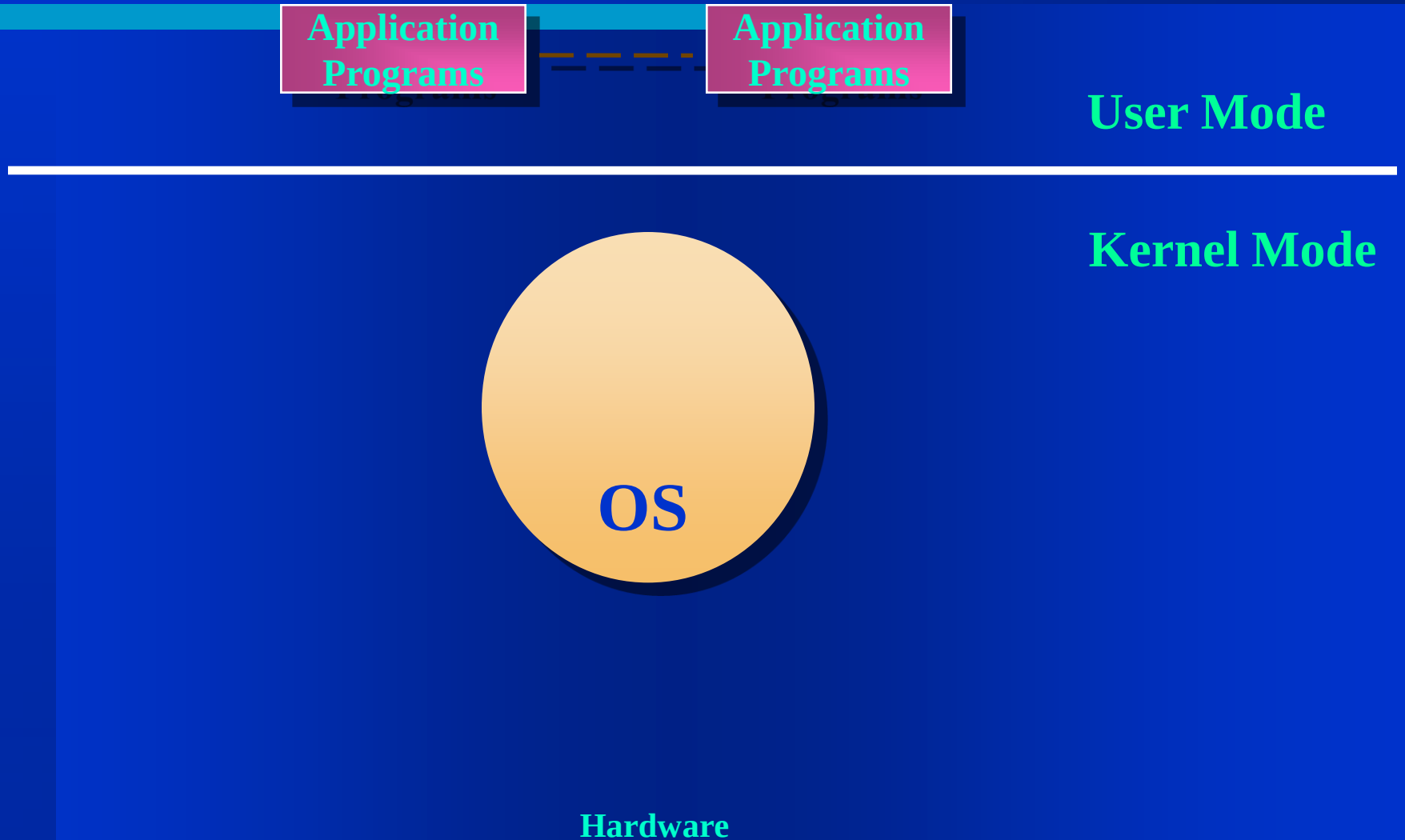
Application
Programs

User Mode

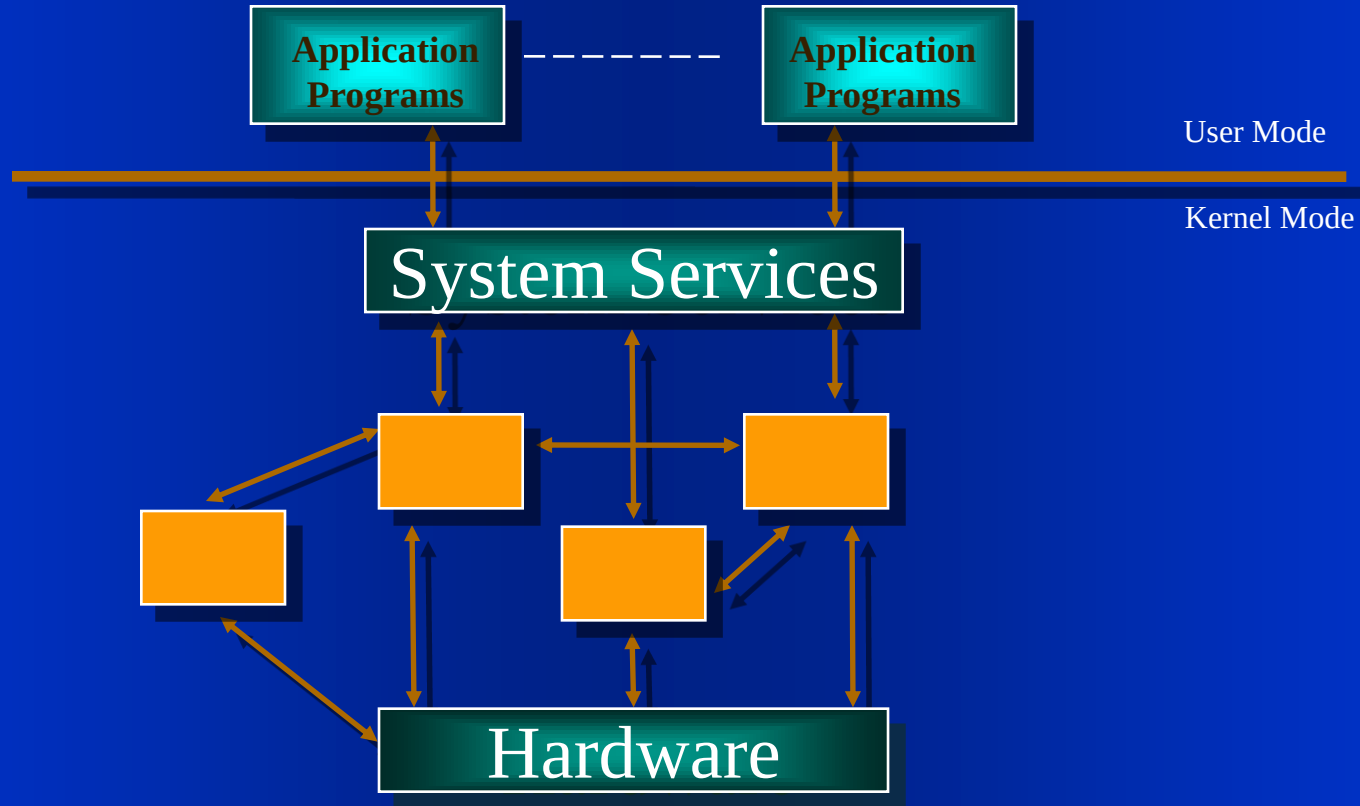
Kernel Mode

OS

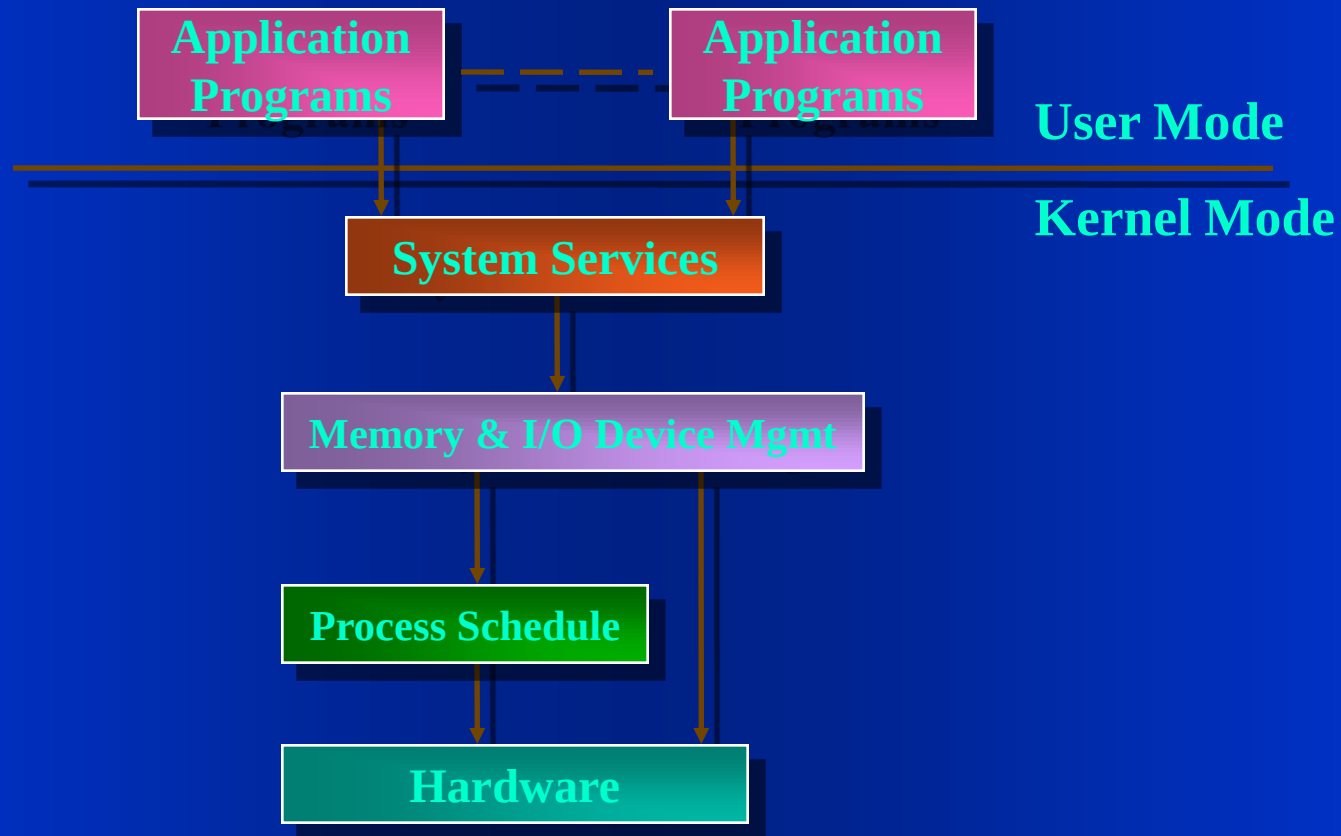
Hardware



SO

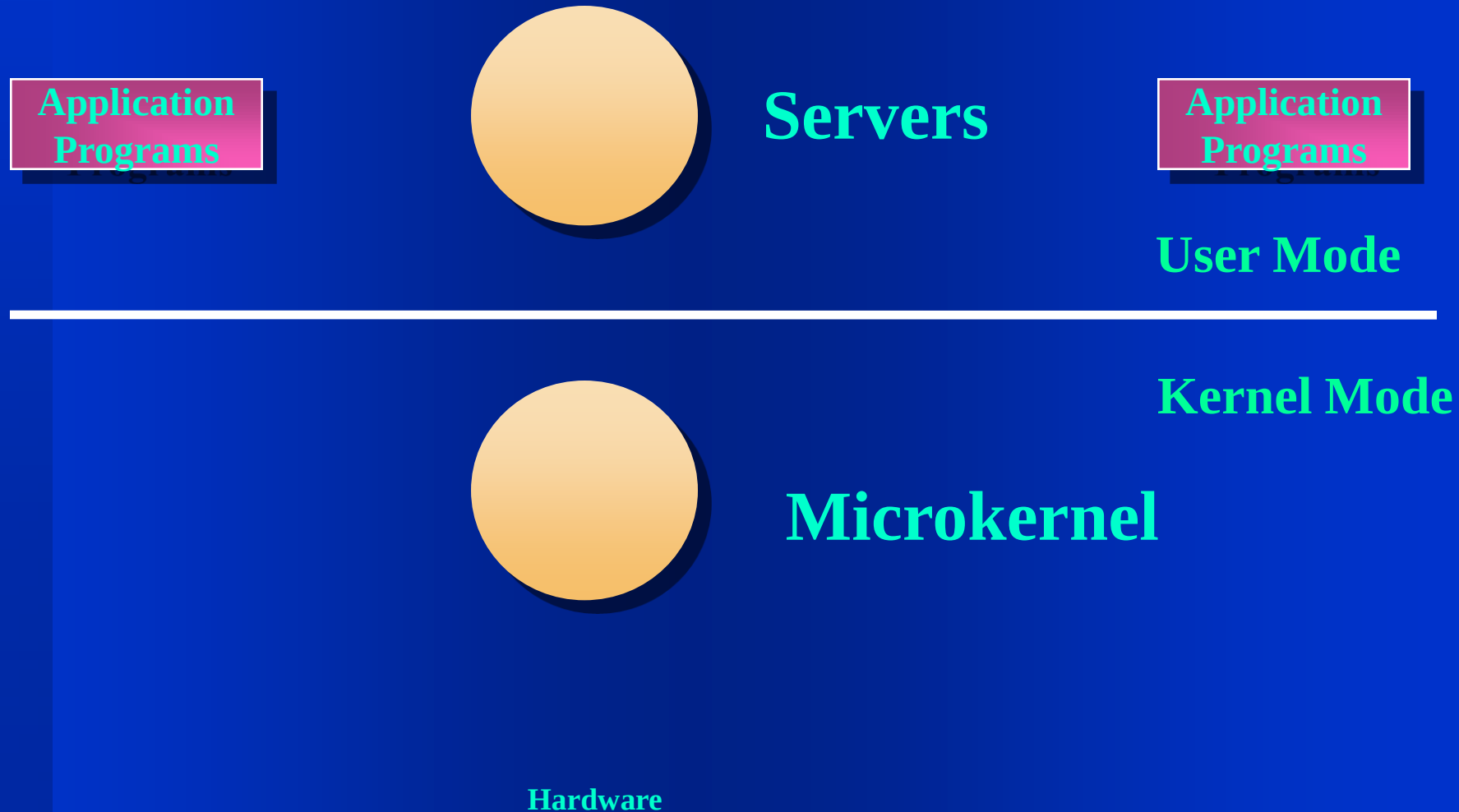


SO organizado em camadas

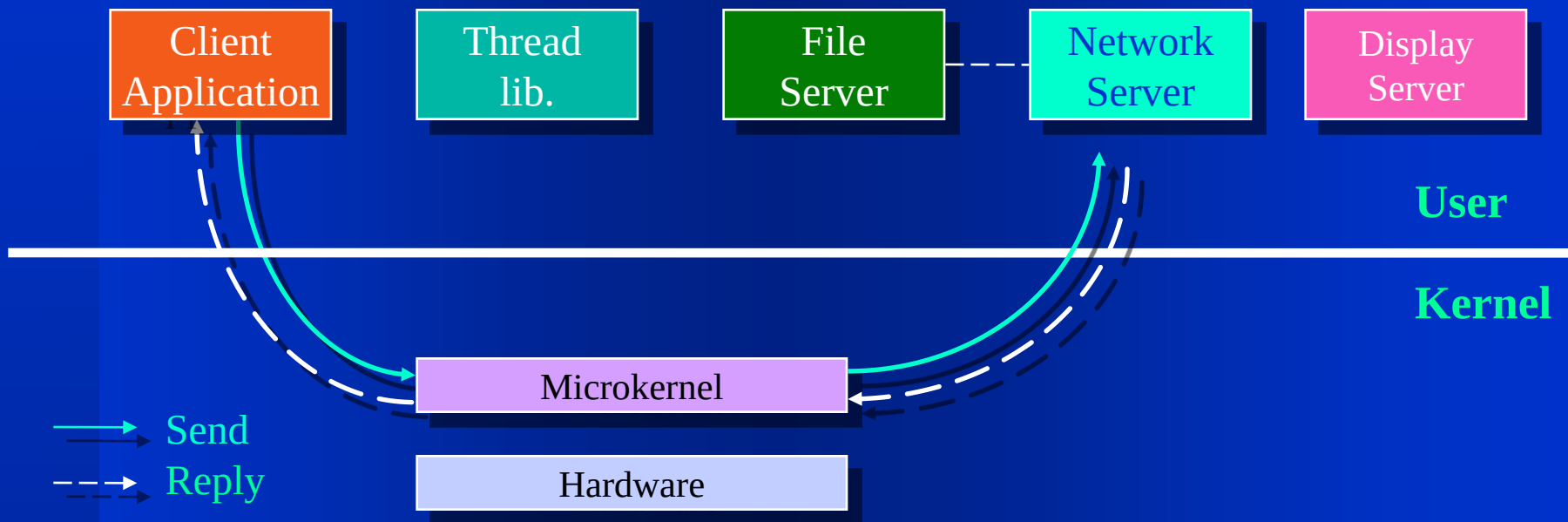


Cada camada acede uma interface de mais baixo nível

Tendência para SO mais flexíveis



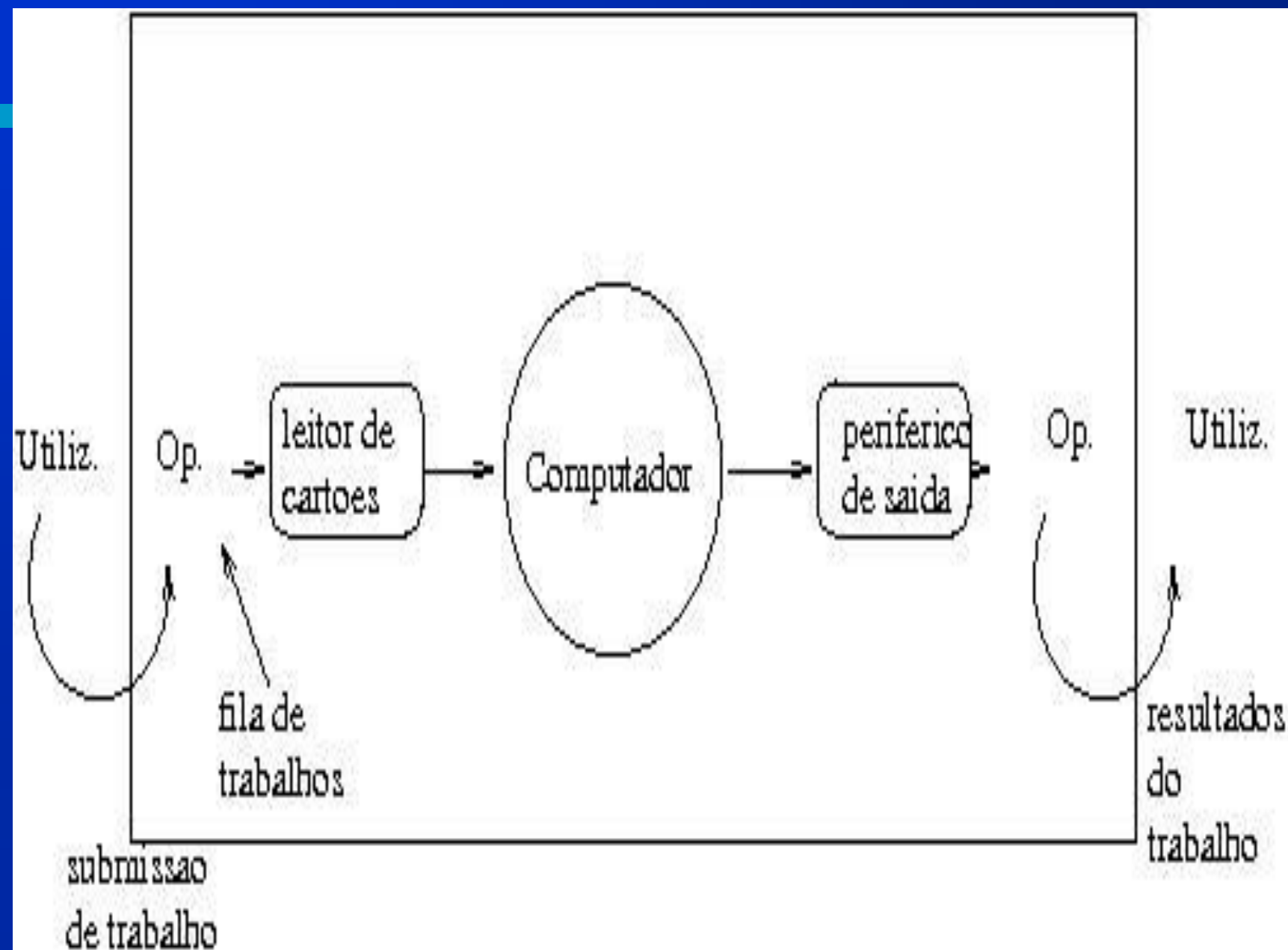
SO = núcleo mínimo + serviços



- Tiny OS kernel providing basic primitives (process, memory, communication)
- Traditional services become subsystems
- OS = Microkernel + User Subsystems

Perspectiva do Utilizador...

1 - Processamento não interactivo, em lotes (*Batch*)



Processamento em lotes (*batch*)

- - define-se a noção de Trabalho (*Job*): uma sequência de passos:
carregar, compilar, ligar, executar
pedida por um utilizador autenticado

- O Utilizador trabalha *off-line*, em modo não-interactivo, submetendo pedidos de trabalhos (programas e dados)
- Cada trabalho é descrito numa linguagem de descrição (*Job Description / Job Control Language*)

Processamento em *Batch*



IBM BlueGene/L

- **top500.org** (Department of Energy-EUA)
 - 32768 cpu
 - 70720 Gflops



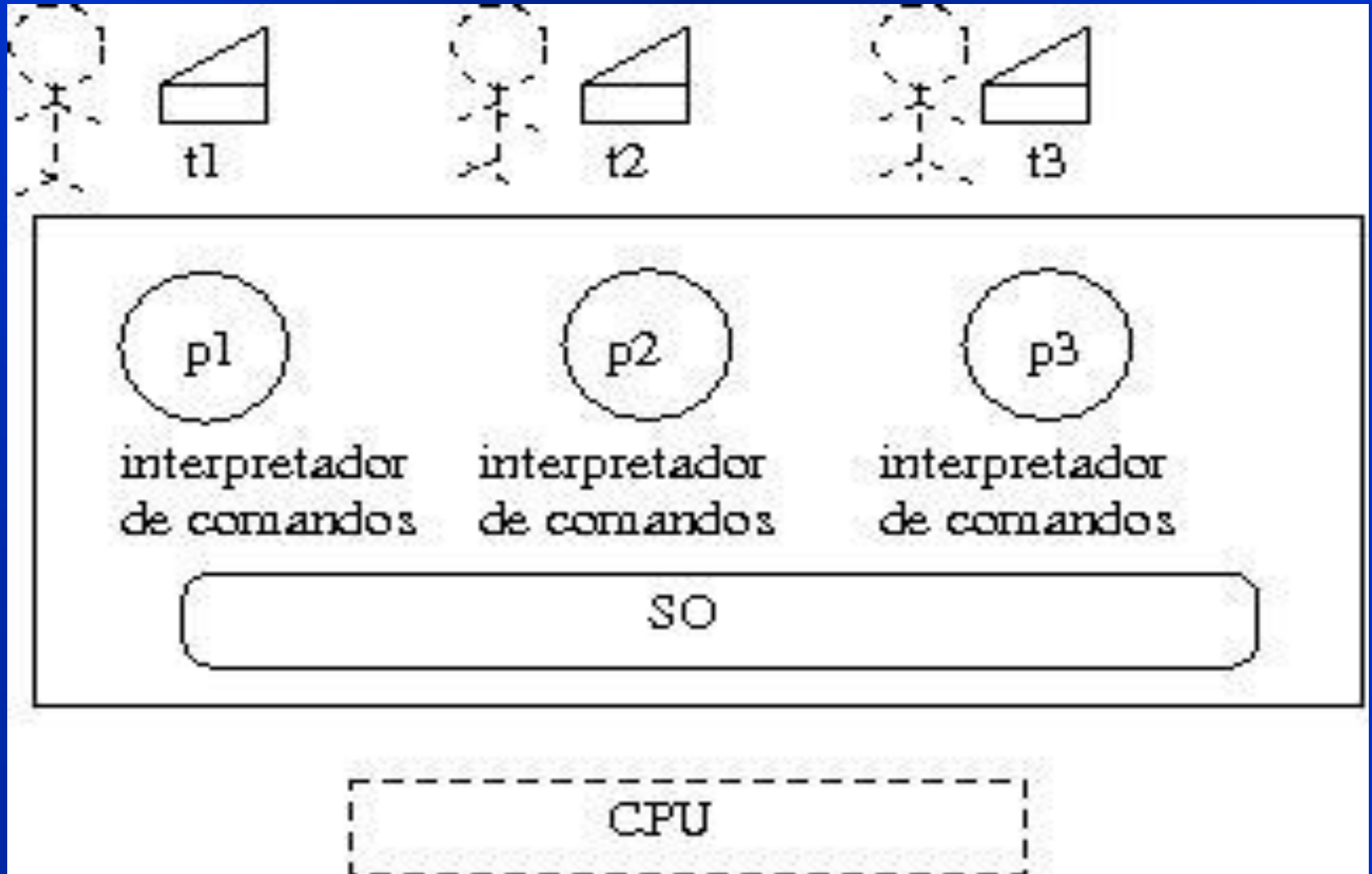
Columbia/SGI Altix supercluster

- **top500.org (NASA)**

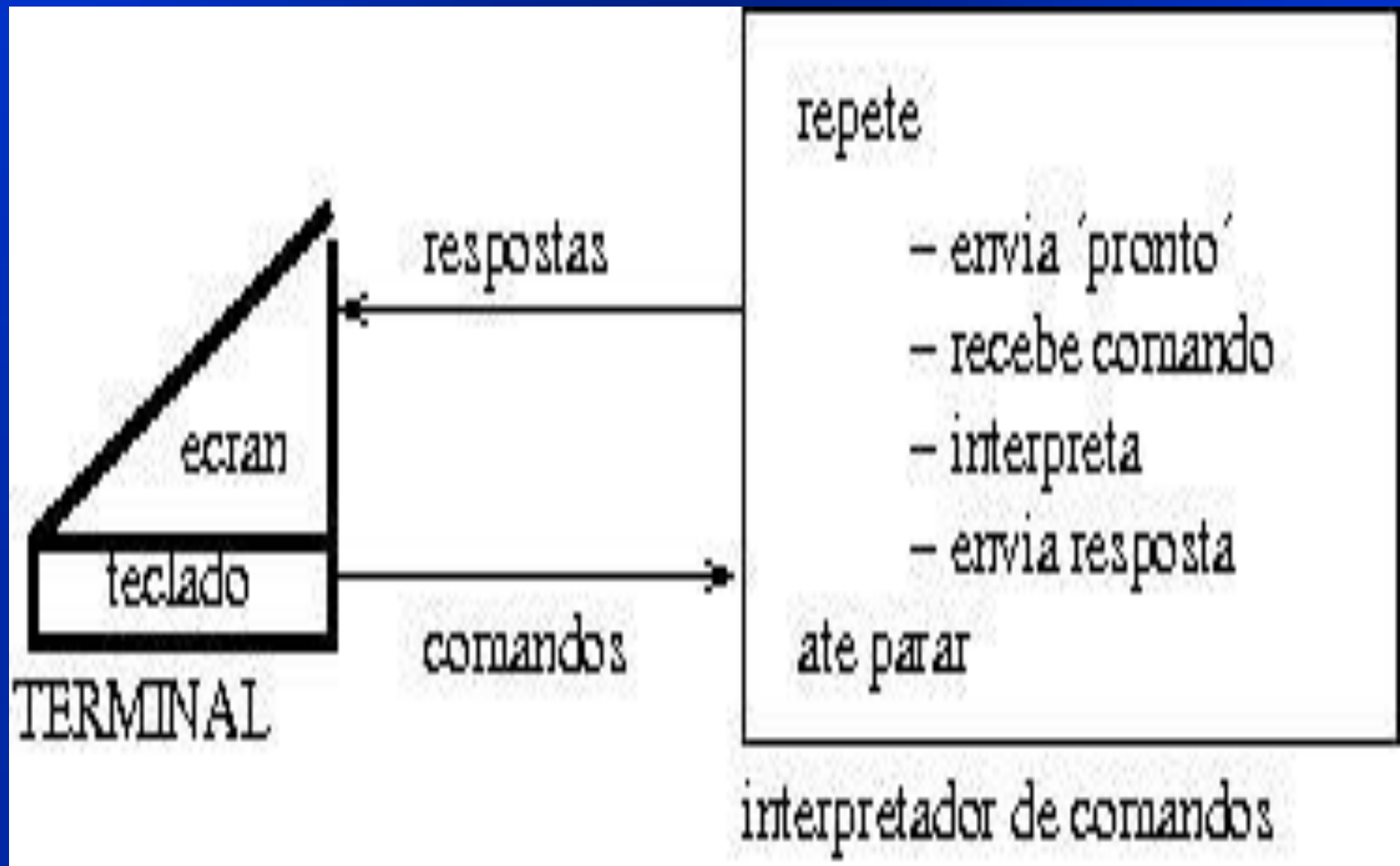
- 10240 cpu
- 51870 GFlops
- 20 TeraBytes RAM
- 440 TeraBytes discos



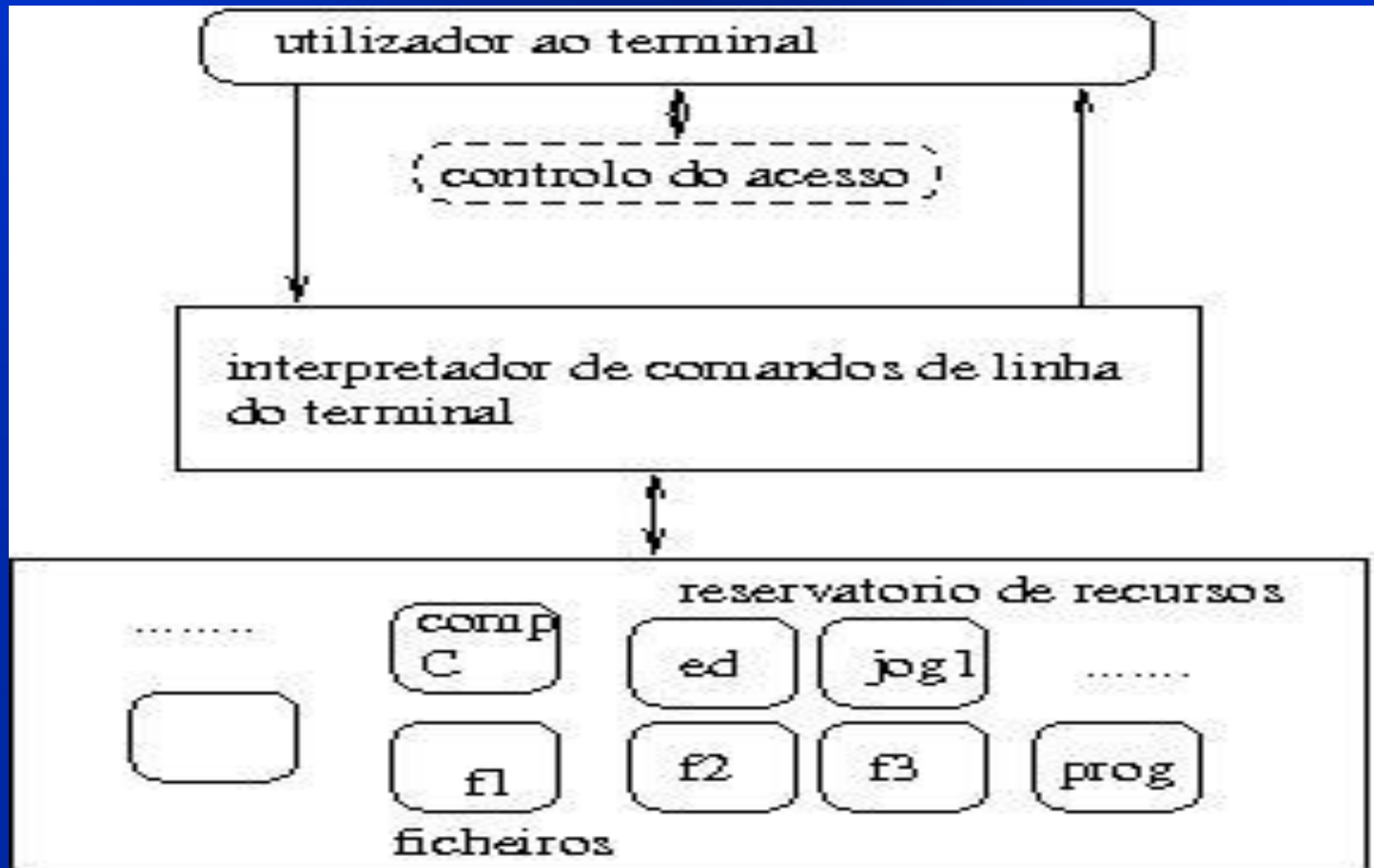
2 - Sistemas interactivos



Interpretador de comandos



Máquina virtual vista p/ utilizador



3 - Sistemas de controlo de processos em Tempo Real

Soft: capazes de adquirir e guardar dados de entrada ao ritmo a que chegam, sem os perder, mas com os processamentos feitos *post-mortem*

Baseados em Deadlines: tempos limite, prazos definidos pela aplicação, para produzir os resultados desejados

Hard: aquisição em tempo real e reacção também, exigindo processamento e resposta imediatos



Interfaces de acesso ao SO

- **Interpretador de comandos: *shell* ou menus**
 - Pedidos através do terminal
- **Chamadas de funções de bibliotecas suportando linguagens específicas (eg C):**
 - Tratam os tipos de dados específicos (eg strings)
 - Algumas recorrem às chamadas ao SO
- **Chamadas ao SO:**
 - Invocadas por instruções máquina ‘ ‘ Chamada ao Supervisor ‘ ‘

Instruções da máquina virtual

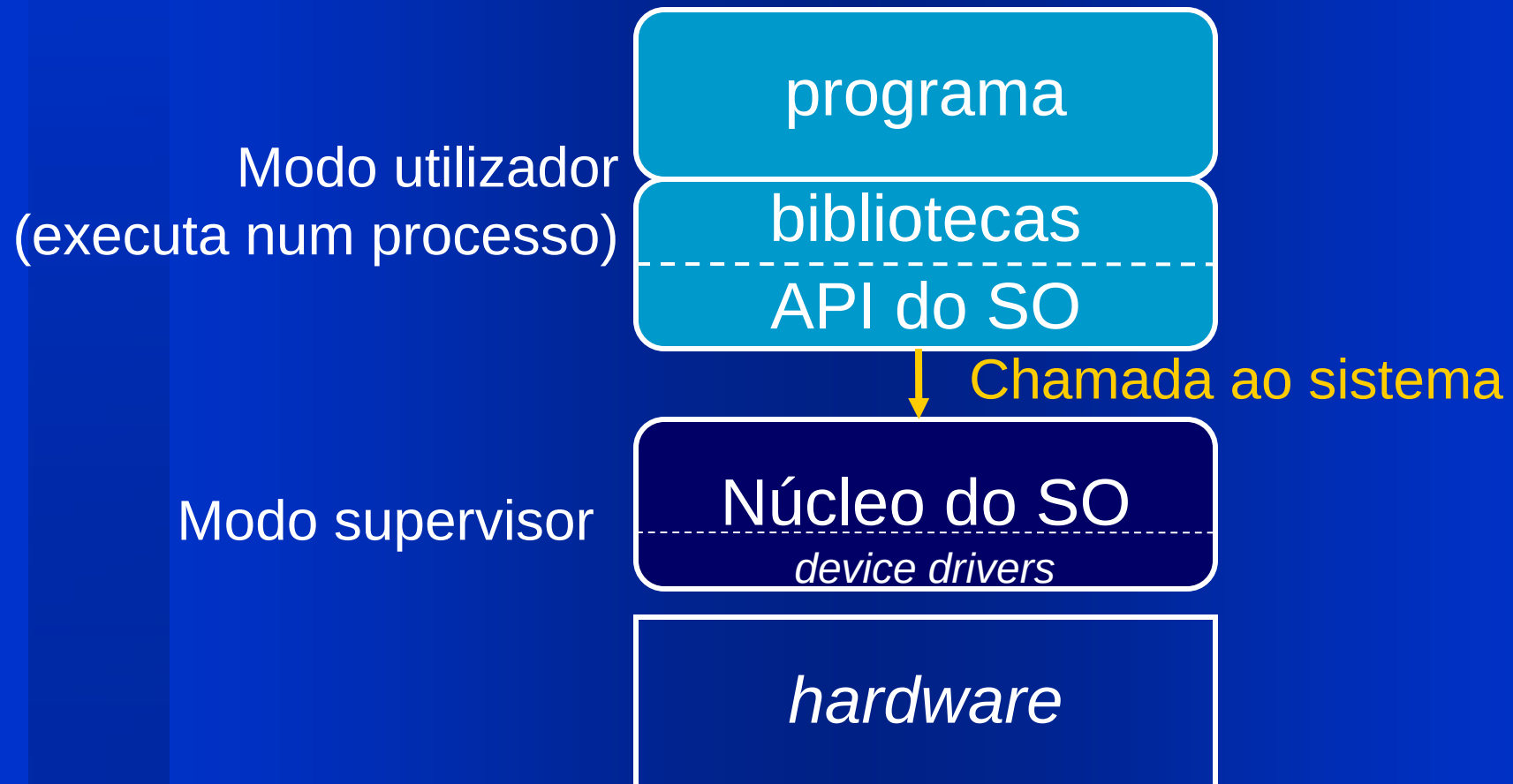
- Um programa em execução tem acesso a:
 - Instruções da máquina hardware
 - Chamadas a funções de biblioteca
 - Chamadas ao SO
- Para além disso, o SO suporta, automaticamente, o Ambiente de Execução do programa, de modo a simular uma máquina virtual dedicada

Classes de chamadas ao SO

- Pedir a execução de novos programas
- Controlar a execução de programas
- Acesso a ficheiros e directorias
- Comunicação e sincronização de processos concorrentes
- Obter informação de estado do sistema

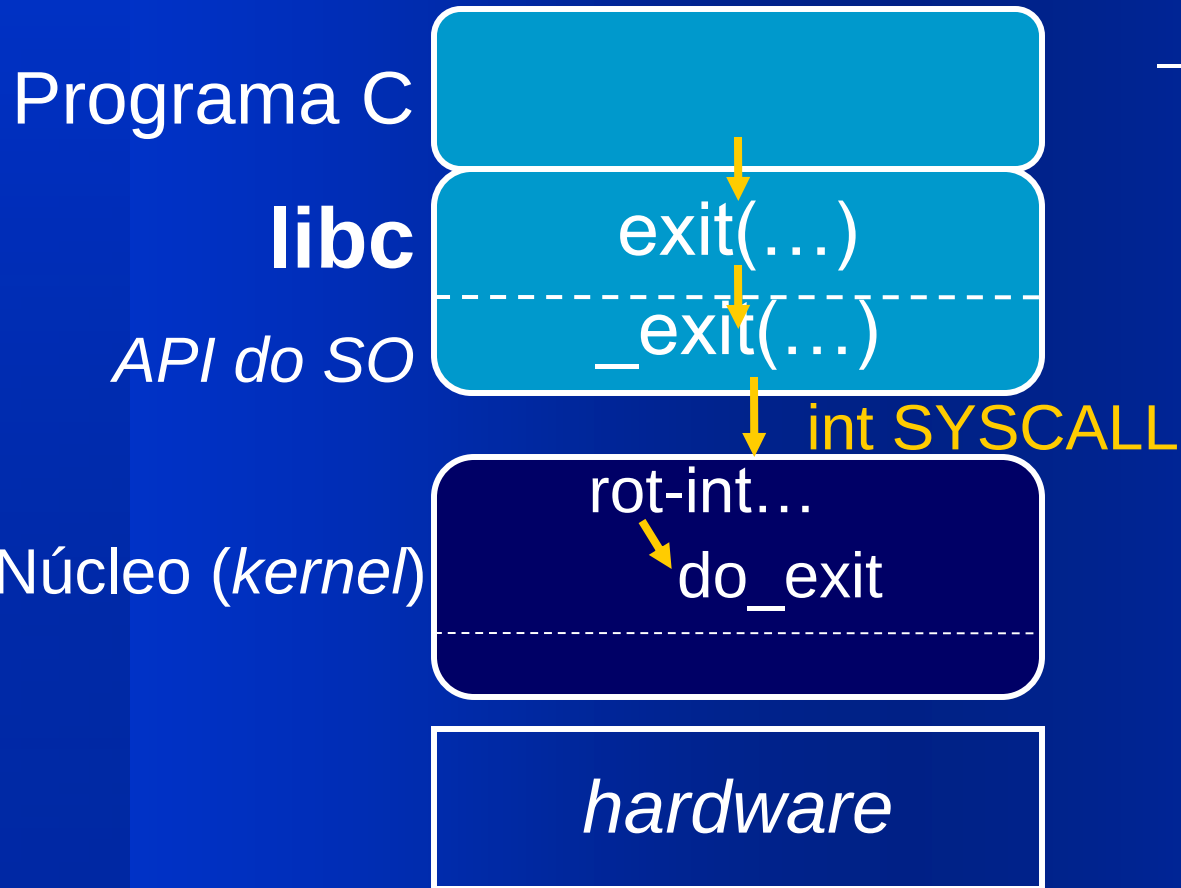
Hierarquia de níveis

- Visão de uma aplicação



Chamada ao sistema `_exit`

```
void _exit(int status);
```



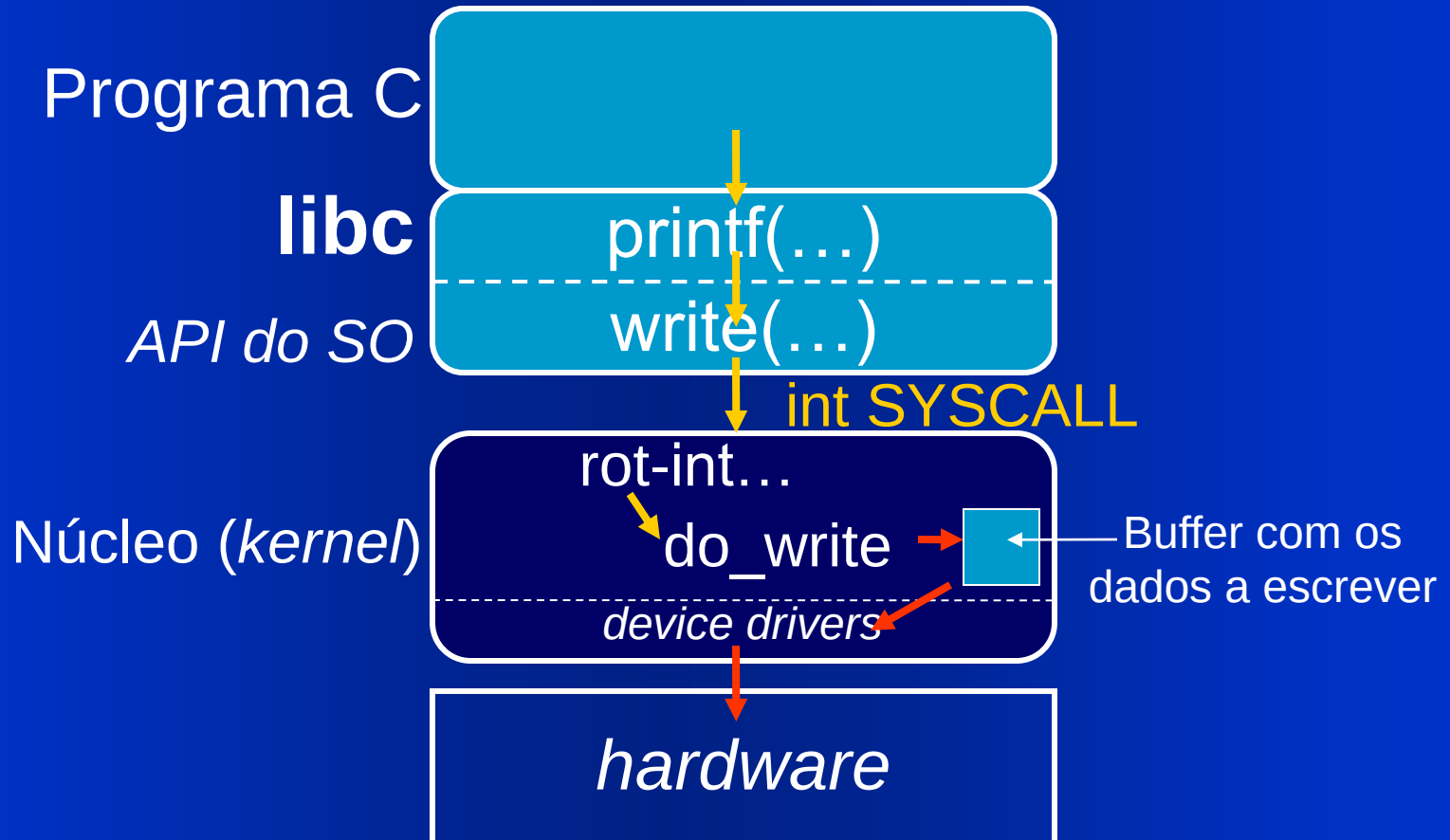
Código possível

```
_exit: ...  
mov eax,EXIT_SCALL  
mov ebx,[ebp+4]  
int SYSCALL  
...
```

No caso do linux/x86:
EXIT_SCALL=1
SYSCALL=0x80

Chamadas ao sistema (I/O)

- Exemplo de saída de dados





a1

SEQUENCIAL

a2

tempo — ordenacao de eventos

a1

CONCORRENTE

a2

tempo — ordenacao de eventos

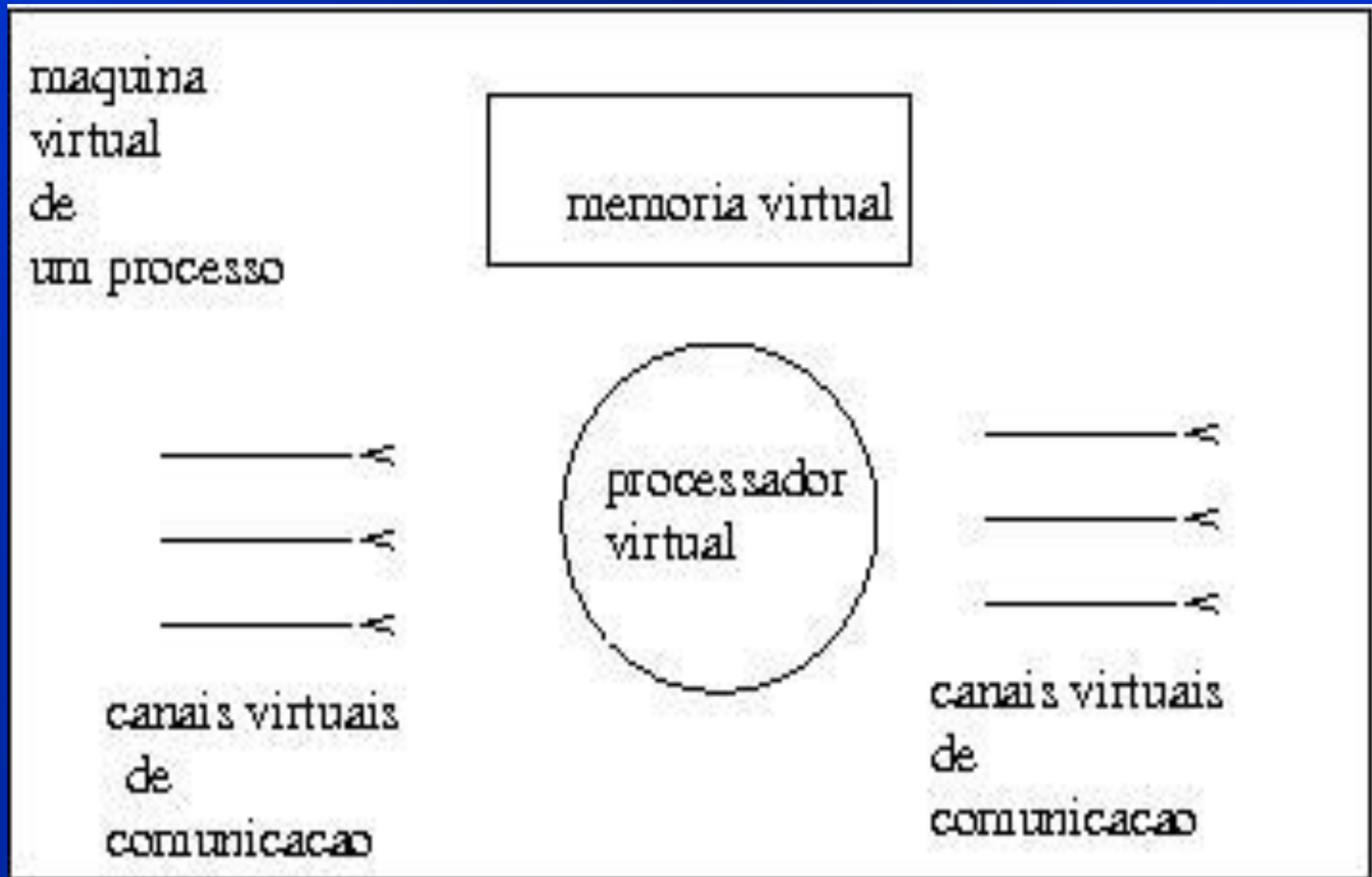
SO baseados em processos

- Nem todas as acções do SO exigem as interrupções desligadas
- Maior descentralização das acções do SO
- **Processos:**
 - entidades assíncronas interrompíveis
 - partilham o CPU e passam por estados executando, bloqueado, pronto
- Processo para acções de 'device driver'
- Processo para execução de cada programa

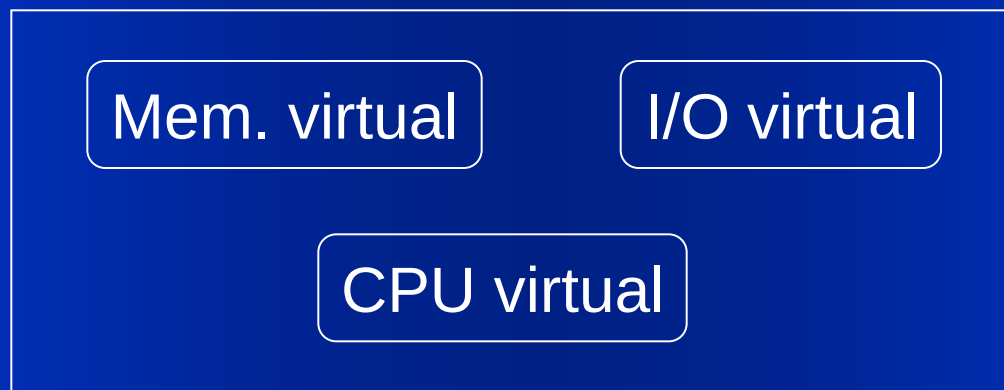
Controlo de Processos

- Ambiente de execução dos programas:
 - instruções da máquina *hardware*
 - chamadas a funções de biblioteca
 - chamadas ao sistema de operação
- Suporte duma Máquina Virtual dedicada a cada processo:
 - Processador Virtual
 - Memória Virtual
 - Canais Virtuais de Comunicação

Máquina virtual do processo



Máquina virtual de um processo

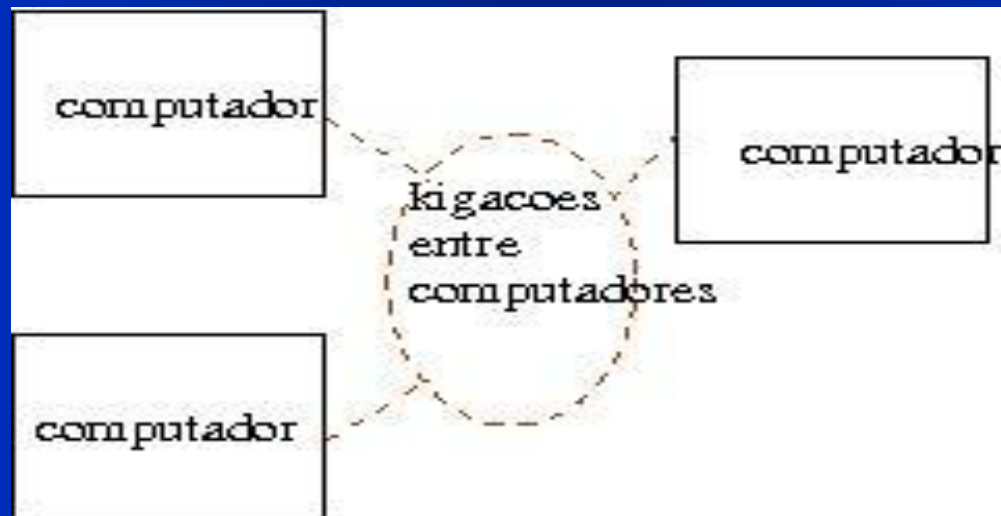
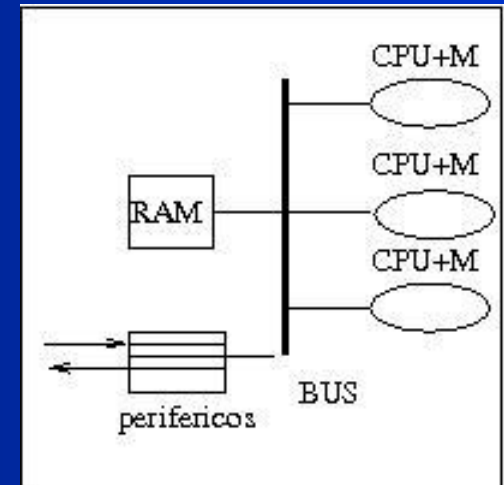
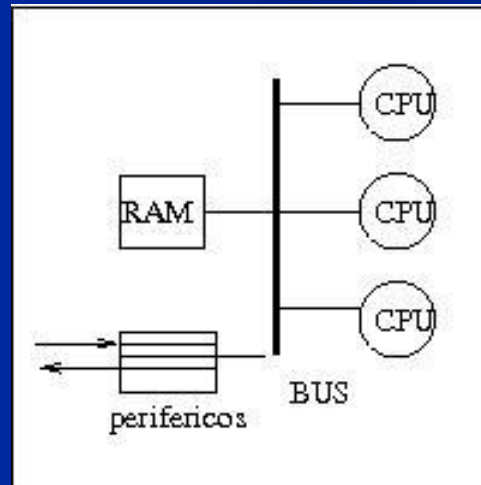
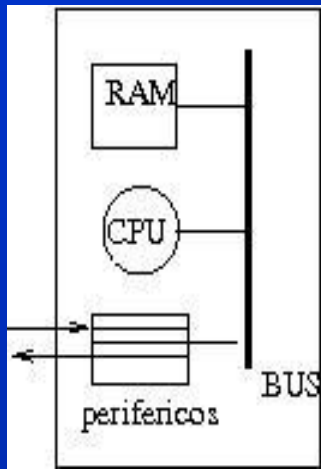


- O SO gere e projecta na máquina real
 - Memória virtual → mapa de memória
 - CPU virtual → CPU real / *time-sharing*
 - I/O virtual → canais de I/O → ficheiros e periféricos

Processador virtual → real

- Que processador real (CPU) vai executar as instruções de cada processo?
 - A) Se só 1 CPU → multiprogramação
 - B) Se múltiplos CPU → execução paralela: escolhem-se os CPU livres para executar os processos que forem sendo pedidos...
- Quando todos os CPU ocupados → passar a multiprogramar cada um deles

1- Processador virtual → real



Memória virtual num computador

Significa suportar um espaço de endereços virtuais de tamanho independente do da memória central

Endereços virtuais: 64 bits $\rightarrow 2^{64}$ bytes

Endereços reais bus: 32 bits $\rightarrow 2^{32}$ bytes

Com base na capacidade agregada de memória central + secundária (discos) num mesmo computador

Registadores do CPU

sob controlo do programa

Memória cache

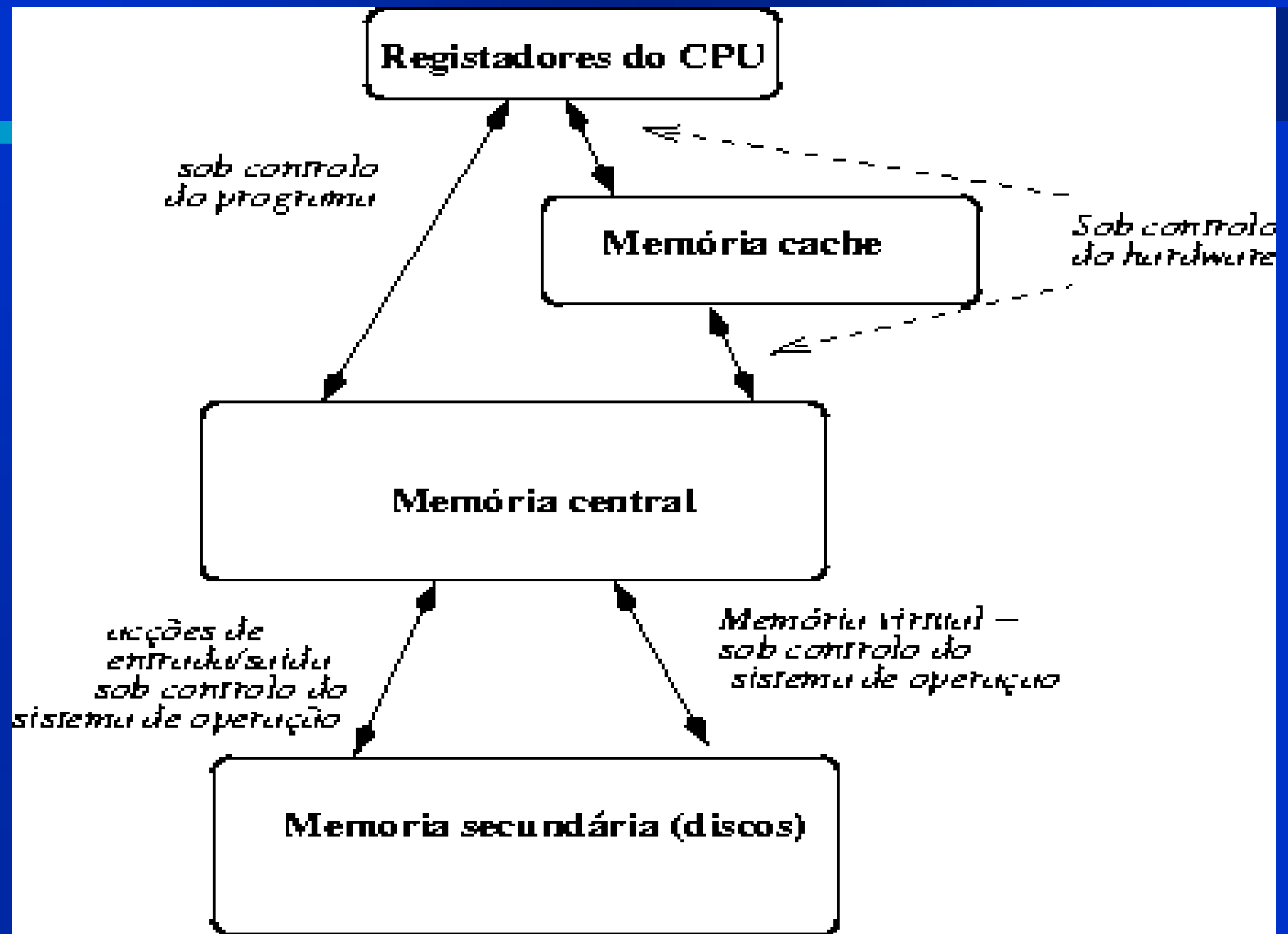
Sob controlo da hardware

Memória central

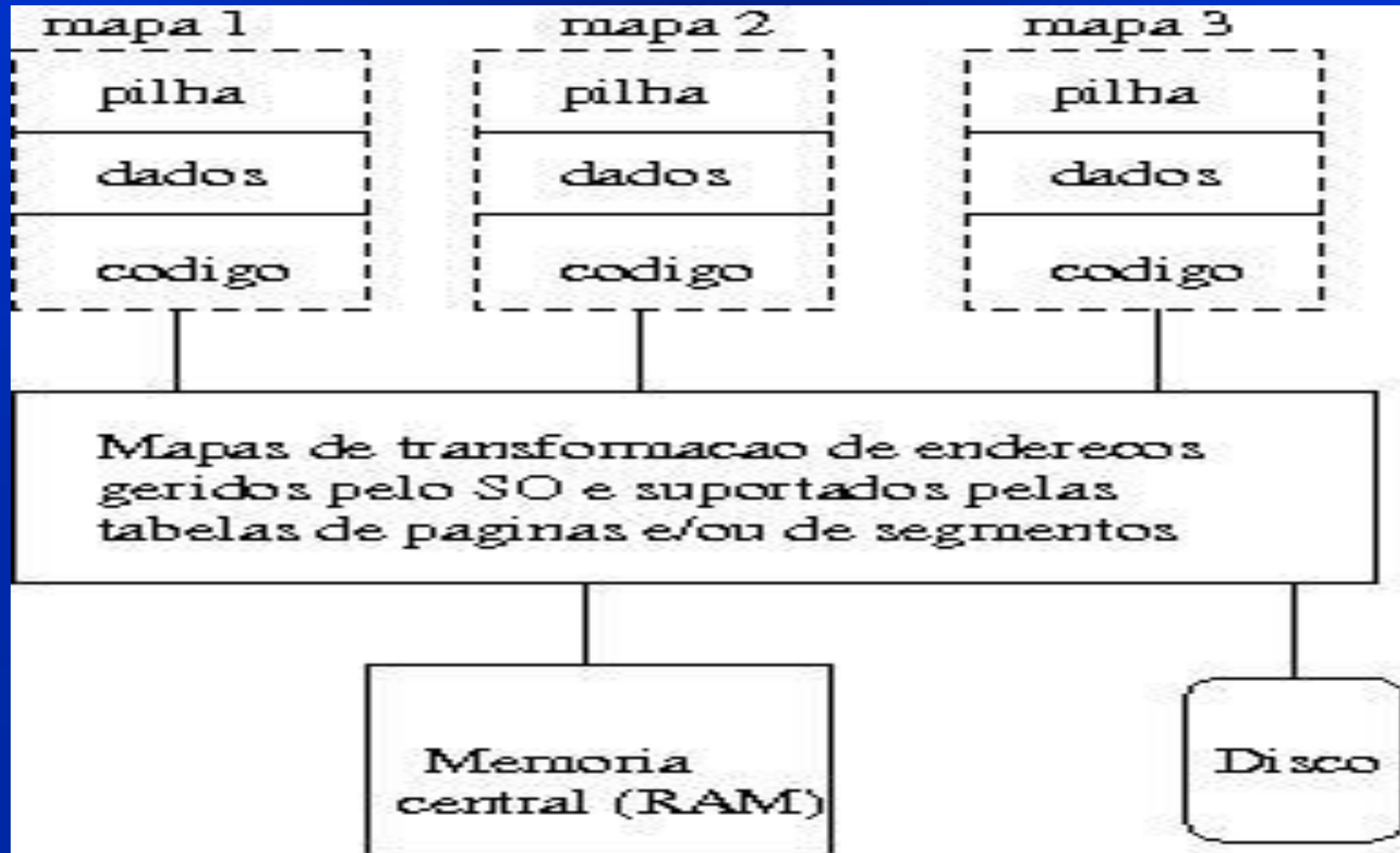
*ações de entrada/saída
sob controlo do
sistema de operação*

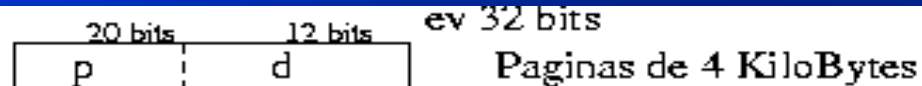
*Memória virtual –
sob controlo do
sistema de operação*

Memória secundária (discos)



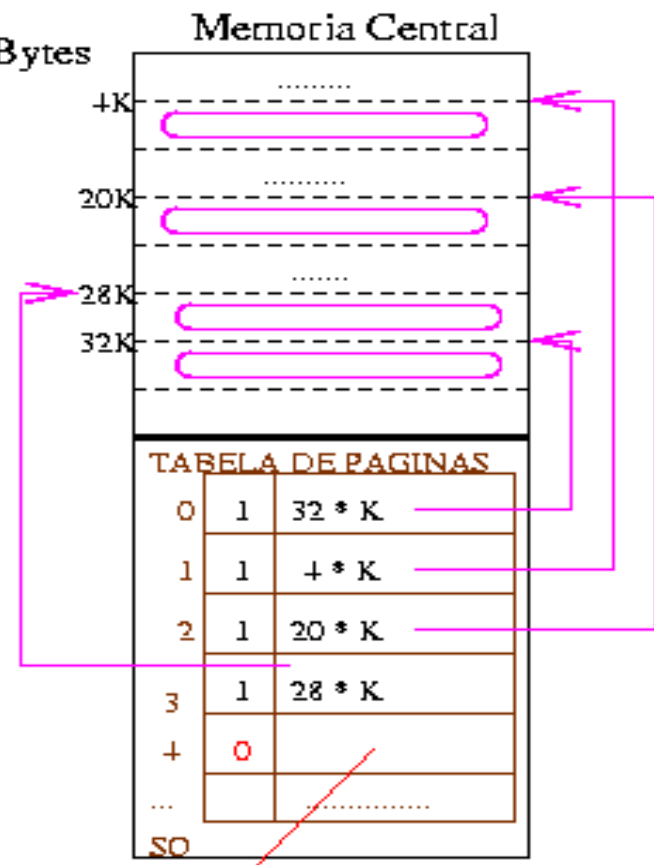
2- Memória virtual → real





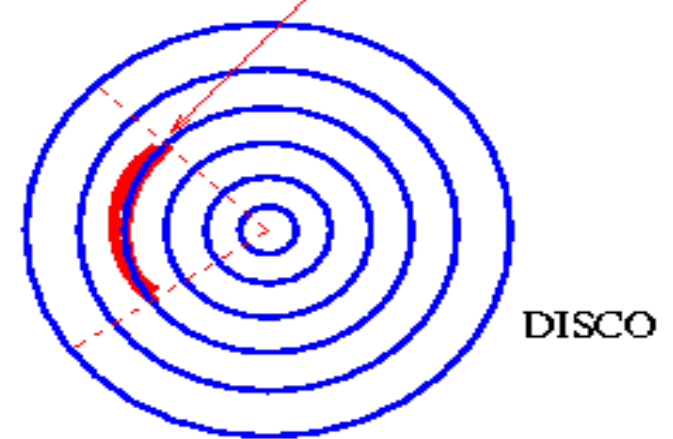
(TLB) dentro do CPU

→		
→	0	32 * K
→		
→	1	4 * K
→		
→	3	28 * K
→		



Sequencia de referencias:

- (p = 0, d= 1)
- (p = 1, d=2)
- (p = 3, d=1)
- (p, = 4, d=1)



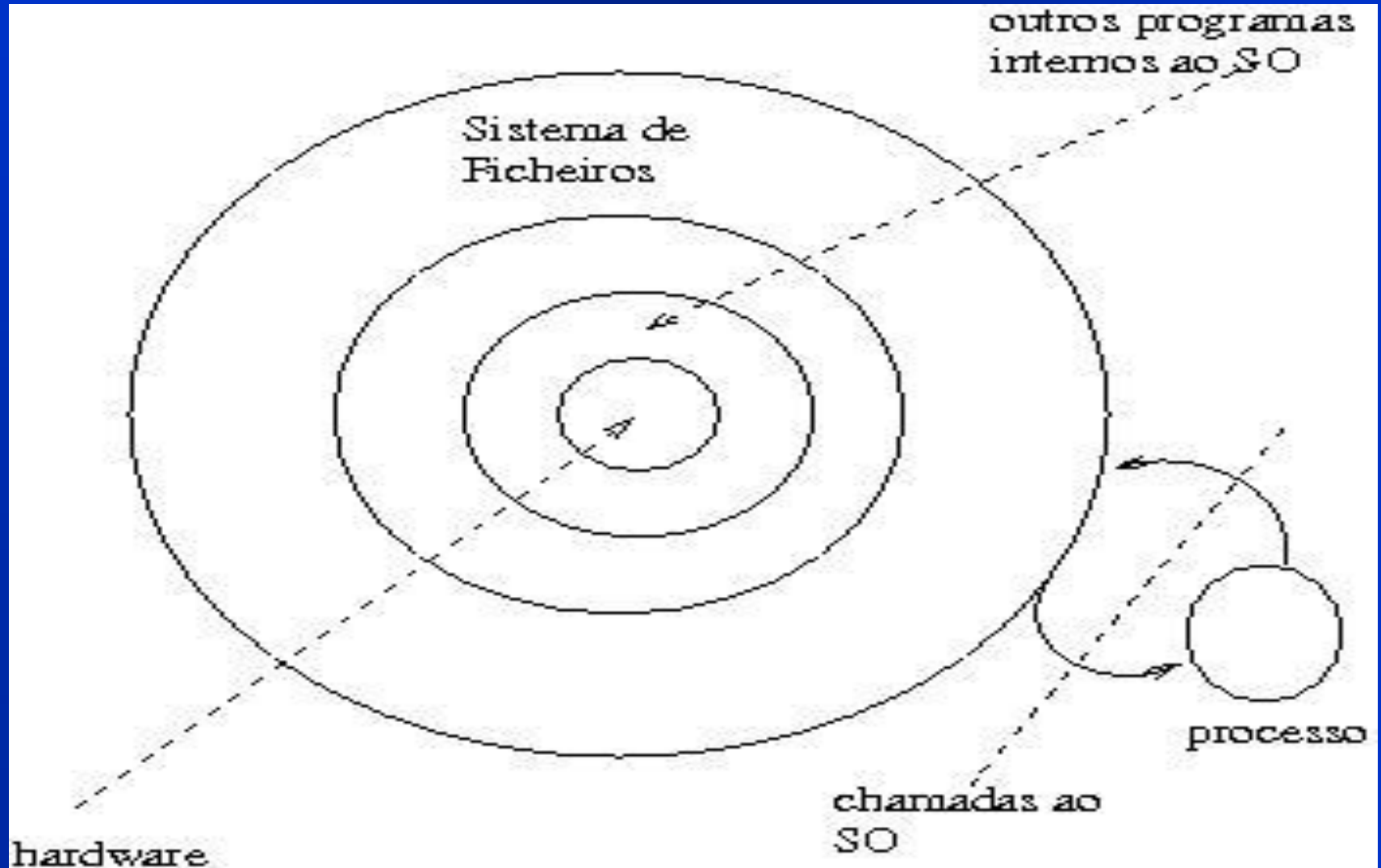
3- Canais virtuais de I/O

- Uniformizar a interacção do processo com o exterior, seja qual for o tipo de dispositivo
- Suportar periféricos virtuais (eg discos e impressoras)
- Esconder, ao processo, a natureza física das operações de entrada e saída e a gestão dos *buffers*

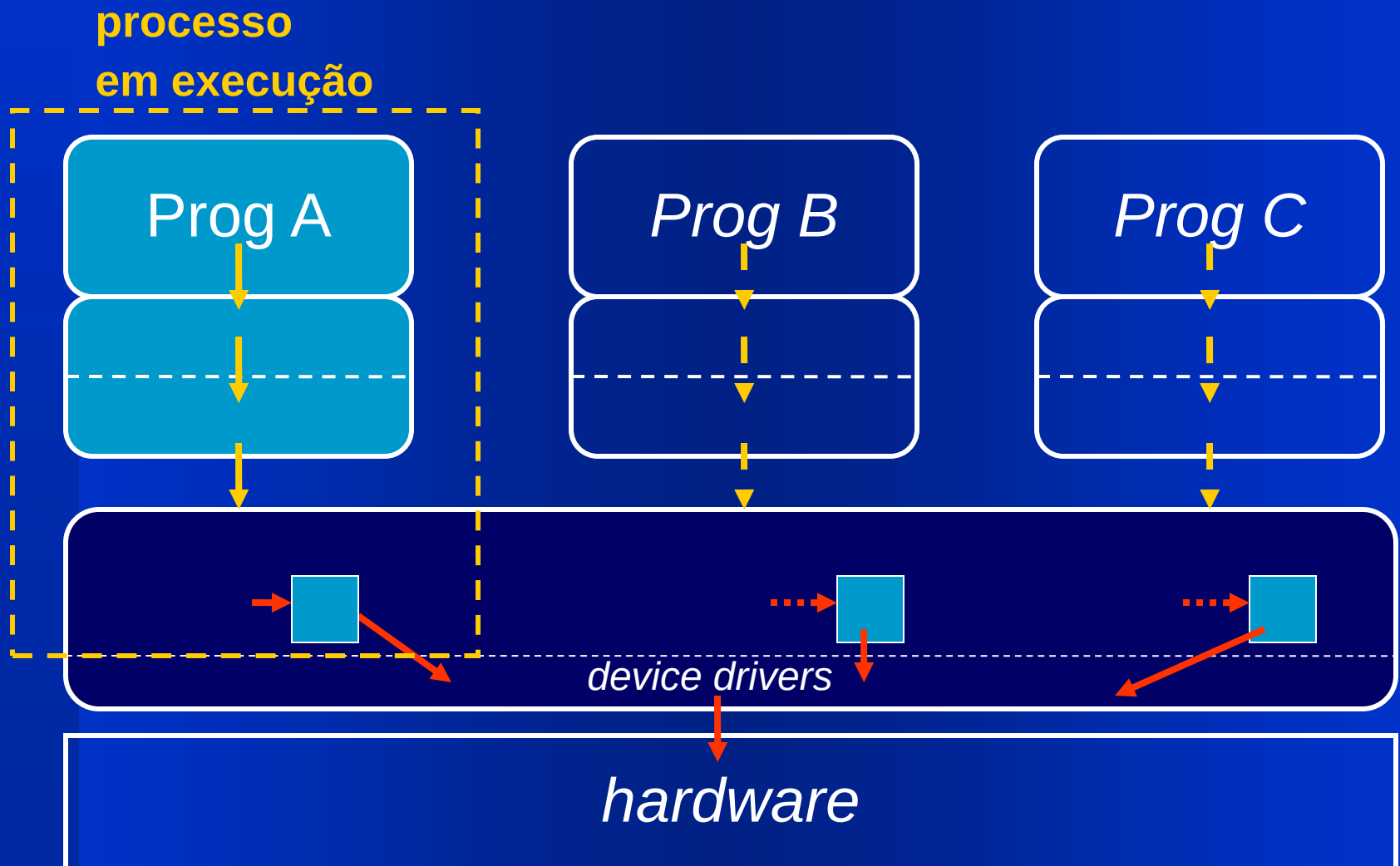
Sistema de ficheiros

- **Organização lógica de ficheiros:**
 - Em hierarquias de directorias
 - Protecção no acesso
 - O ficheiro, como uma estrutura lógica
- **Operações uniformes de I/O:** inicializar, ler, escrever, fechar, criar, destruir
- **Canais virtuais de I/O:** ligar canais a diferentes dispositivos, dinamicamente
- **Gestão de buffers de I/O e dos suportes físicos dos ficheiros**

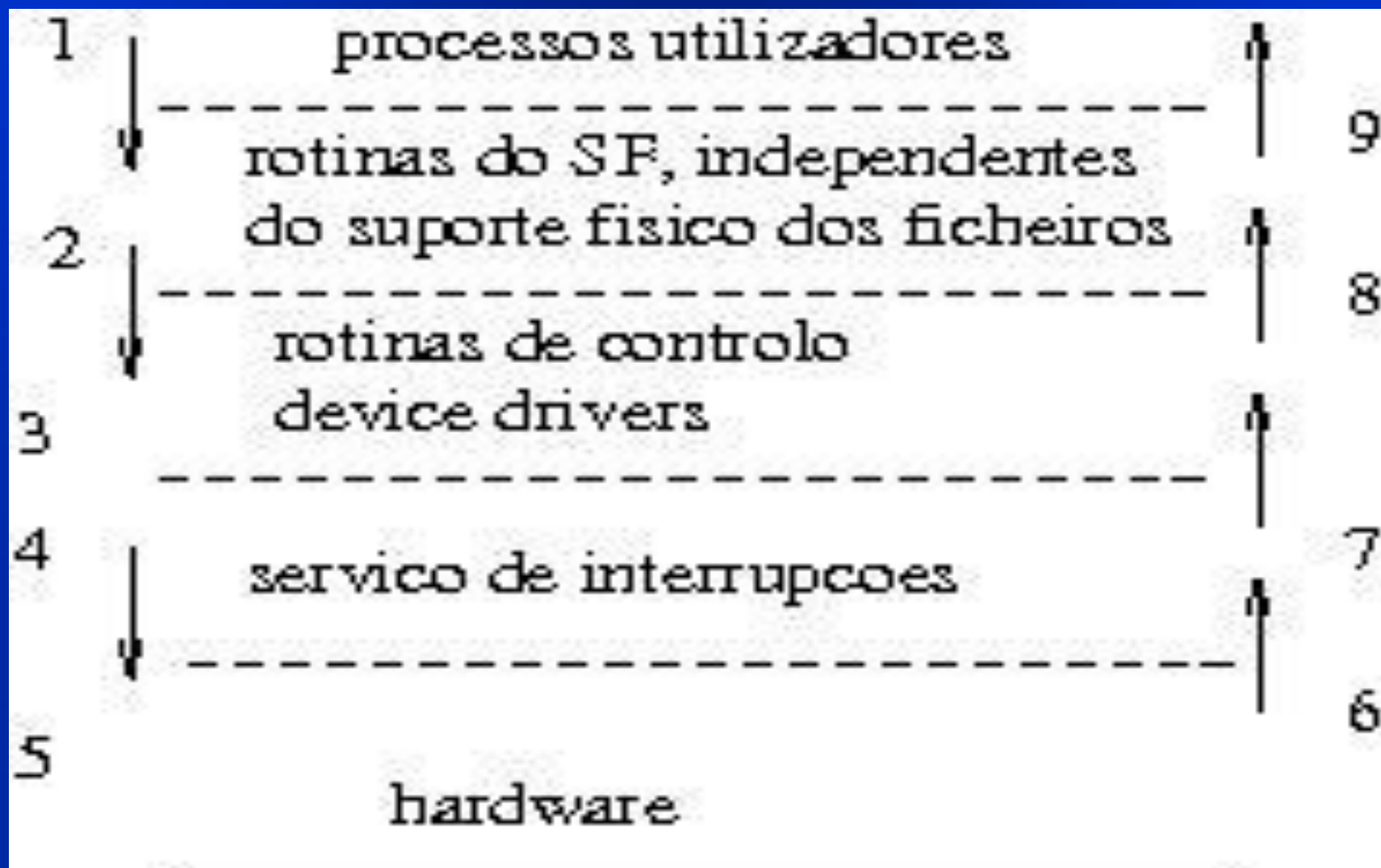
Sistema de ficheiros



Cada processo, seus canais de I/O



Hierarquia de camadas





O Núcleo do SO

- Oferece funções (Chamadas ao SO) para:
 - Processos fazerem pedidos ao SO
- Para operações sobre os Recursos:
 - Processadores, Processos/Threads
 - Memórias,
 - Ficheiros,
 - Entrada/Saídas
 - Comunicação e sincronização

Funções do Núcleo

- **Gestão de Processos:**
 - controlo da execução concorrente e da multiprogramação, da comunicação e da sincronização
- **Gestão de Memória:** central e secundária
- **Sistema de Ficheiros:** acesso, organização em directorias e arquivo
- **Controlo do I/O:** acesso aos ficheiros, comunicação do programa com o exterior e gestão das operações de I/O

Escalonamento de Jobs

Processos

**Camadas do
Núcleo**

GestãoProcessador

Arquitectura Hardware

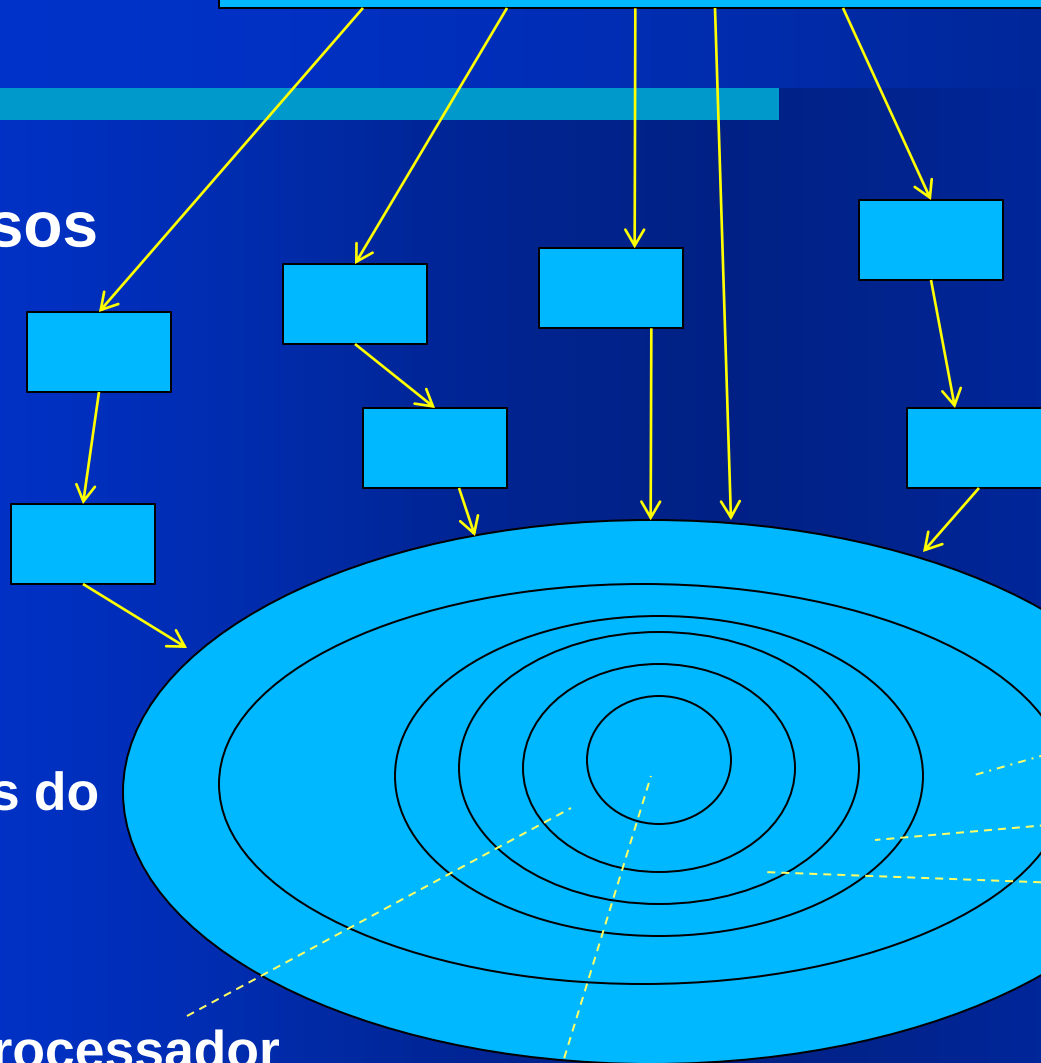
Interface Ch.SO

SistemaFicheiros

GestãoProcessos

GestãoMemória

**GestaoPeriféricos
(nível inferior)**



Gestão do processador:

afecção do CPU / despacho de processos

mecanismos de multiprogramação

processamento de interrupções

Gestão dos periféricos (nível inferior):

Rotinas de controlo dos device drivers/gestão nível físico

Gestão de memória:

afectar e libertar memória, gerir memória central/auxiliar

Gestão de processos:

criação, controlo, comunicação e sincronização

Sistema de ficheiros:

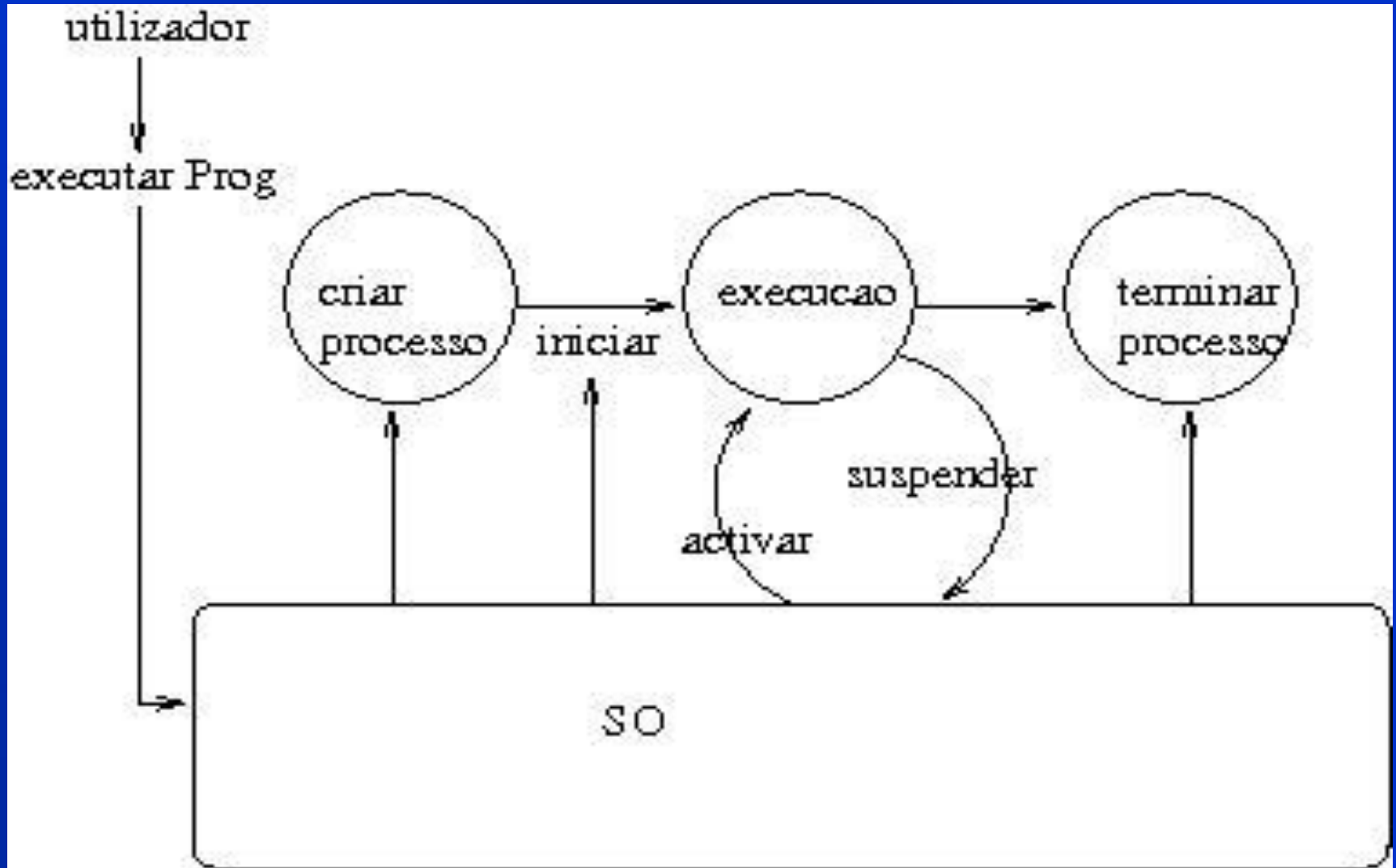
gestão a nível simbólico e operações das chamadas ao SO

Interface das chamadas ao SO



Gestão dos Processos

Ciclo de vida de um processo



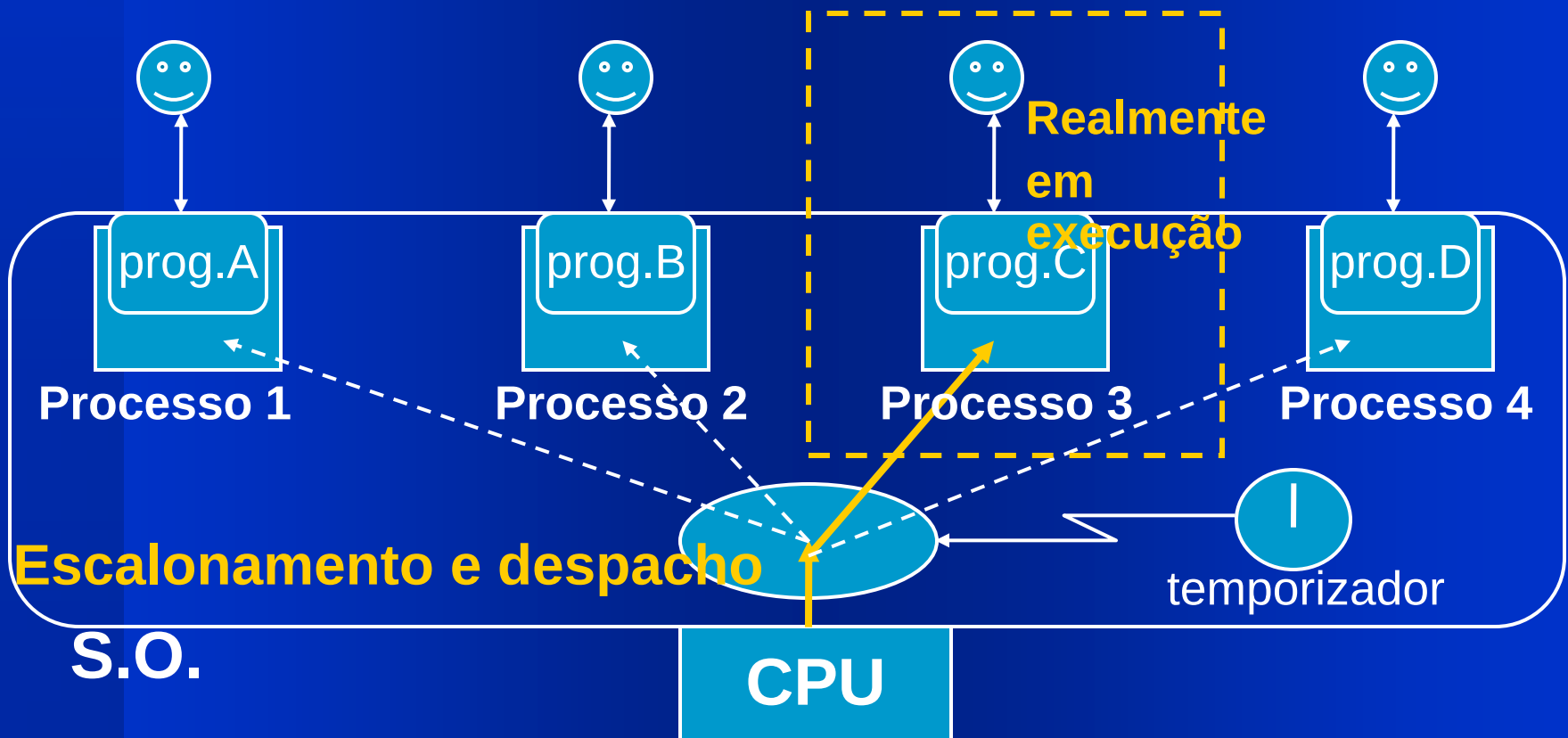
Concorrência através de multiprogramação

O SO controla a execução de múltiplos programas, alternando de uns para outros, em momentos oportunos:

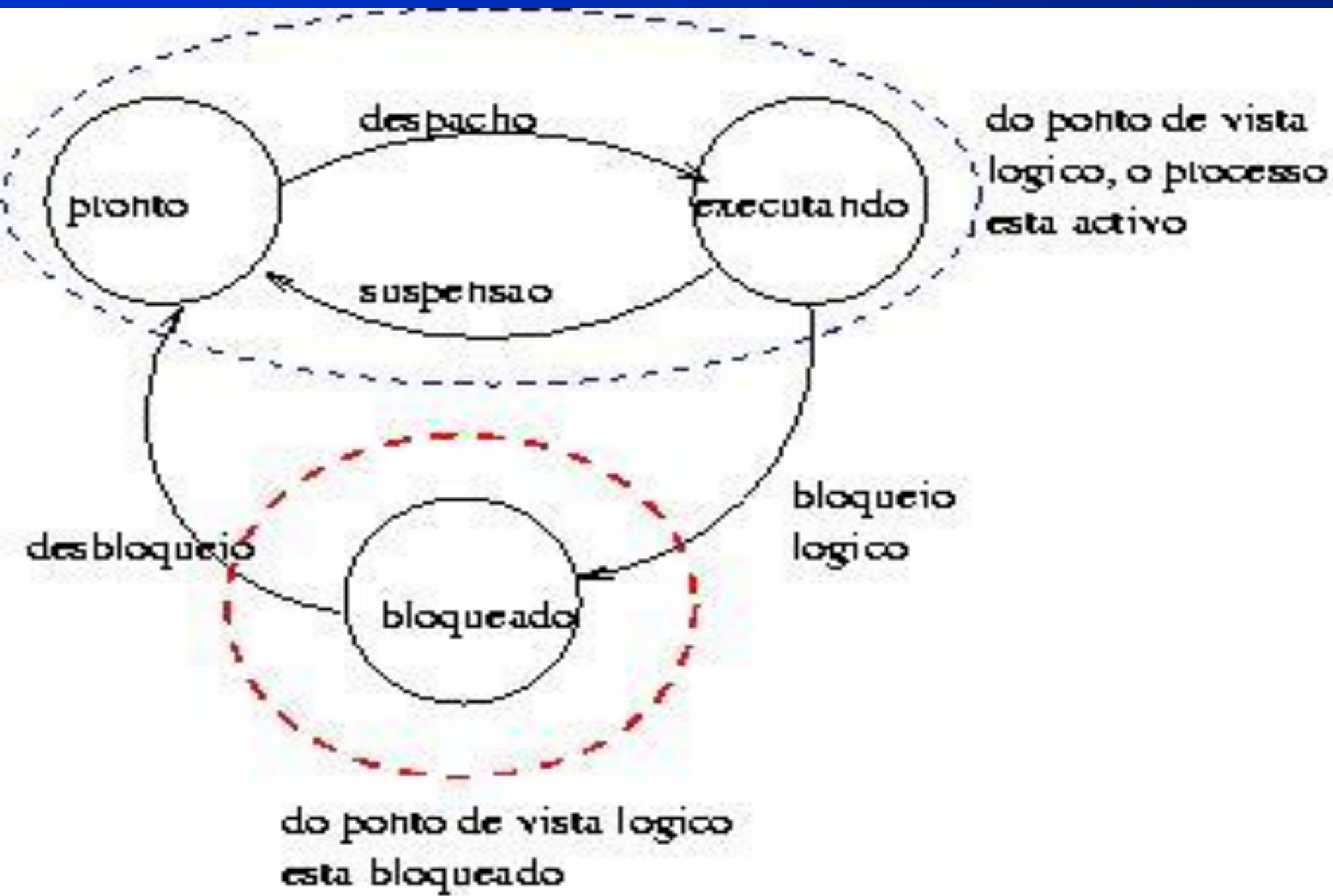
- quando um processo se bloqueia em READ por um buffer estar vazio
- quando um processo faz WAIT por 1 filho que ainda não terminou
- quando um processo expira o tempo de CPU a que tinha direito (TIME-SLICE)
- etc.

Multiprogramação

- Exemplo: 4 processos **concorrem** por 1 CPU
- Se um processo espera por I/O ou termina, o SO atribui o CPU a outro processo



Estados de um processo no SO



Dimensões da gestão de processos

Necessário, em geral:

1 - manter informação sobre o estado dos CPUs, dos trabalhos admitidos ao sistema e dos processos sob multiprogramação

2 - estratégias de escalonamento (scheduling) a nível dos trabalhos (jobs) e dos processos

3 - mecanismos de afectação / desafectação dos CPUS – despacho e comutação de processos

Comutação de processos

Sempre que muda o estado de algum processo, verifica se deve continuar o processo corrente ou não

Multiprogramação

- Cada programa executa num contexto de um processo, definido por:
 - Imagem em memória (código, dados e pilha)
 - Estado do CPU (valores dos registadores)
 - Estado das interfaces de I/O
- O SO muda de contexto quando necessário
 - Suporta um ambiente de execução virtual para cada processo

Um Processo no SO

Contexto de execução de um programa:

- Programa executável**
- Zonas de dados e pilha**
- Zona para salvaguarda de estado do CPU
(PC, SP, outros registadores)**
- Informação sobre canais abertos para
ficheiros e outra informação de controlo**

Guardada em Descritores internos do SO
(Process Id: PID -- aponta para descritor)

Executar um programa → Processo

- **Pedir a criação de um processo:**
 - o SO verifica todas as permissões e...
 - cria as estruturas para gerir o processo
 - inicia todos os seus atributos
 - inicia o mapa de memória com o programa (p. ex. a partir de um ficheiro executável)
 - inicia o CPU, transferindo o controlo para o processo criado (PC, SP,...)

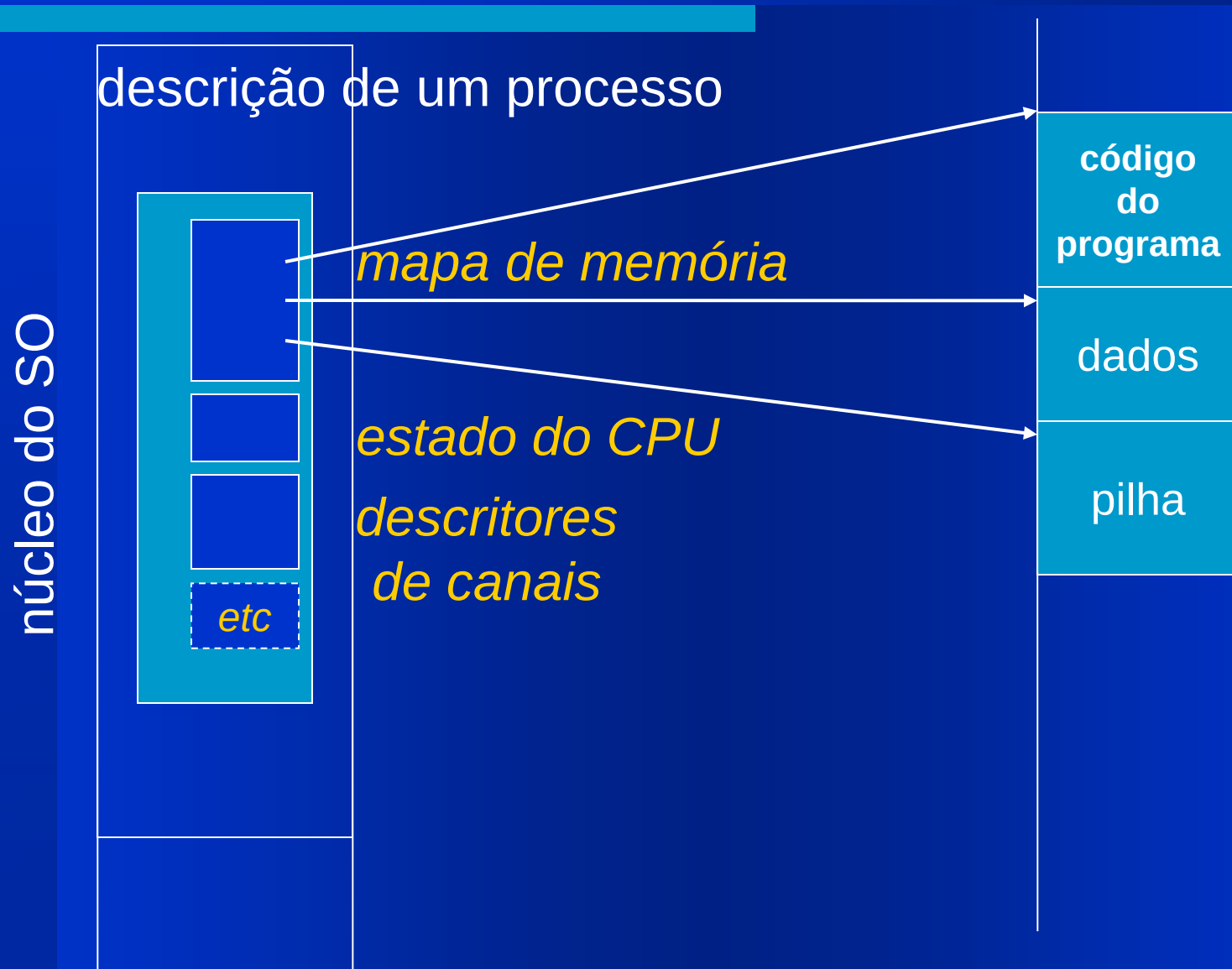


***CriarProcesso(ficheiro, canais-iniciais,
atributos, ambiente,...etc)***

Criação de um processo

- Define:
 - O mapa de memória (código, dados, pilha)
 - Os valores iniciais de variáveis do ambiente (args. do programa, directoria corrente inicial, etc.)
 - Os valores iniciais dos registadores do CPU
 - Os canais iniciais de entrada/saída
 - Uma entrada numa Tabela de Processos, interna ao SO, com o Descritor do Processo

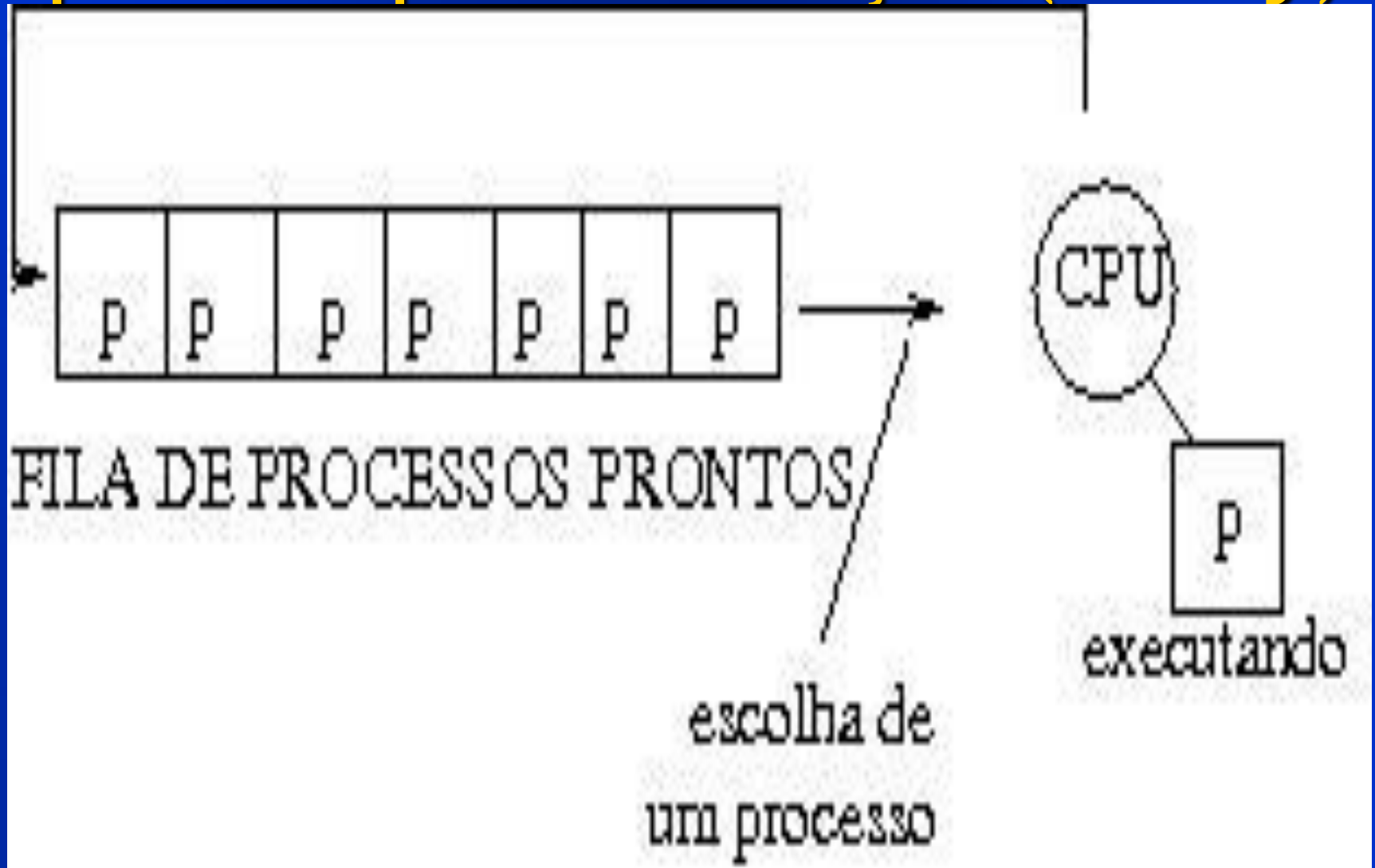
Processo em memória



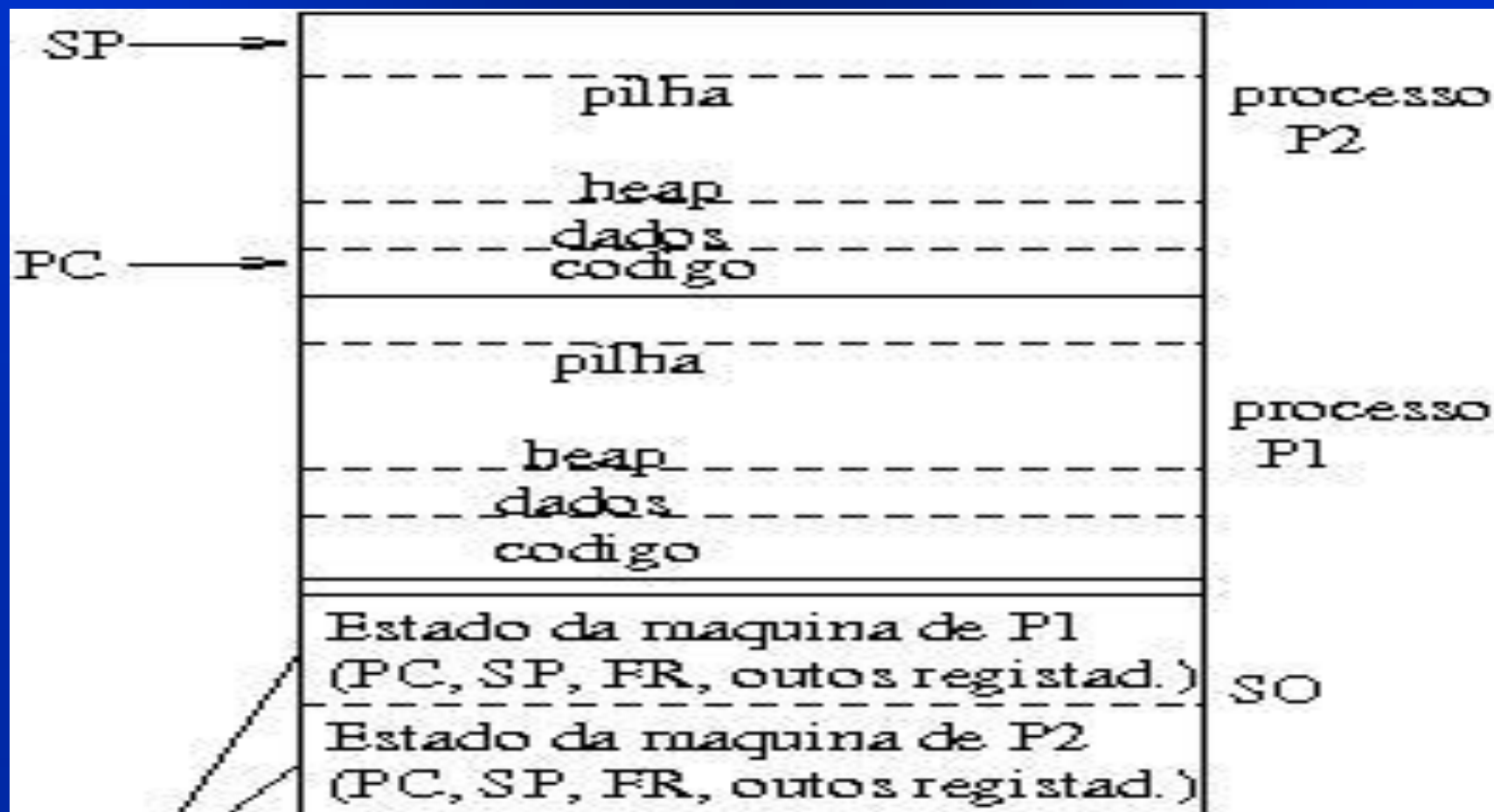
Fila de processos

- Uma vez criado, o processo é posto numa fila de processos, à espera de ser activado para execução:
 - Cada elemento da fila tem o identificador do processo (*Process Identifier*), que indica a respectiva entrada da Tabela de Processos
 - A fila é ordenada segundo as prioridades dos processos

Fila de processos prontos para execução (*Ready*)



Mapas de memória dos processos



entradas na Tabela de processos
que descreve os processos presentes
num dado momento

Instrucao SupervisorCALL Fork()

P1

SupervisorCALL fork()

Eret: pop reg1

interrupt
empilha
PC/flags

cria mapa do filho

StartAddress-filho := PC = Eret

guarda StartAddress do filho
em TabelaProcessosSO

copia informacao do pai

insere filho na FilaProntos

empilha pid-filho na pilha do pai

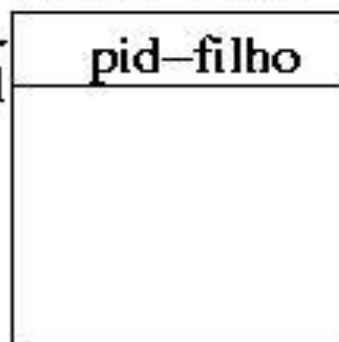
empilha 0 na pilha do filho

retorna ao programa do pai

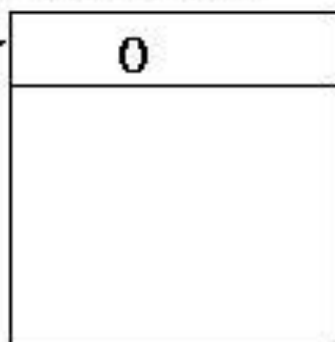
pilha do pai

pilha do filho

sp1



sp2

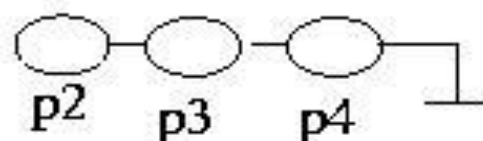


pai estado = executando
PC = Eret; SP = sp1

filho estado = pronto Pid=p4
PC = Eret; SP = sp2

TabelaProcessos

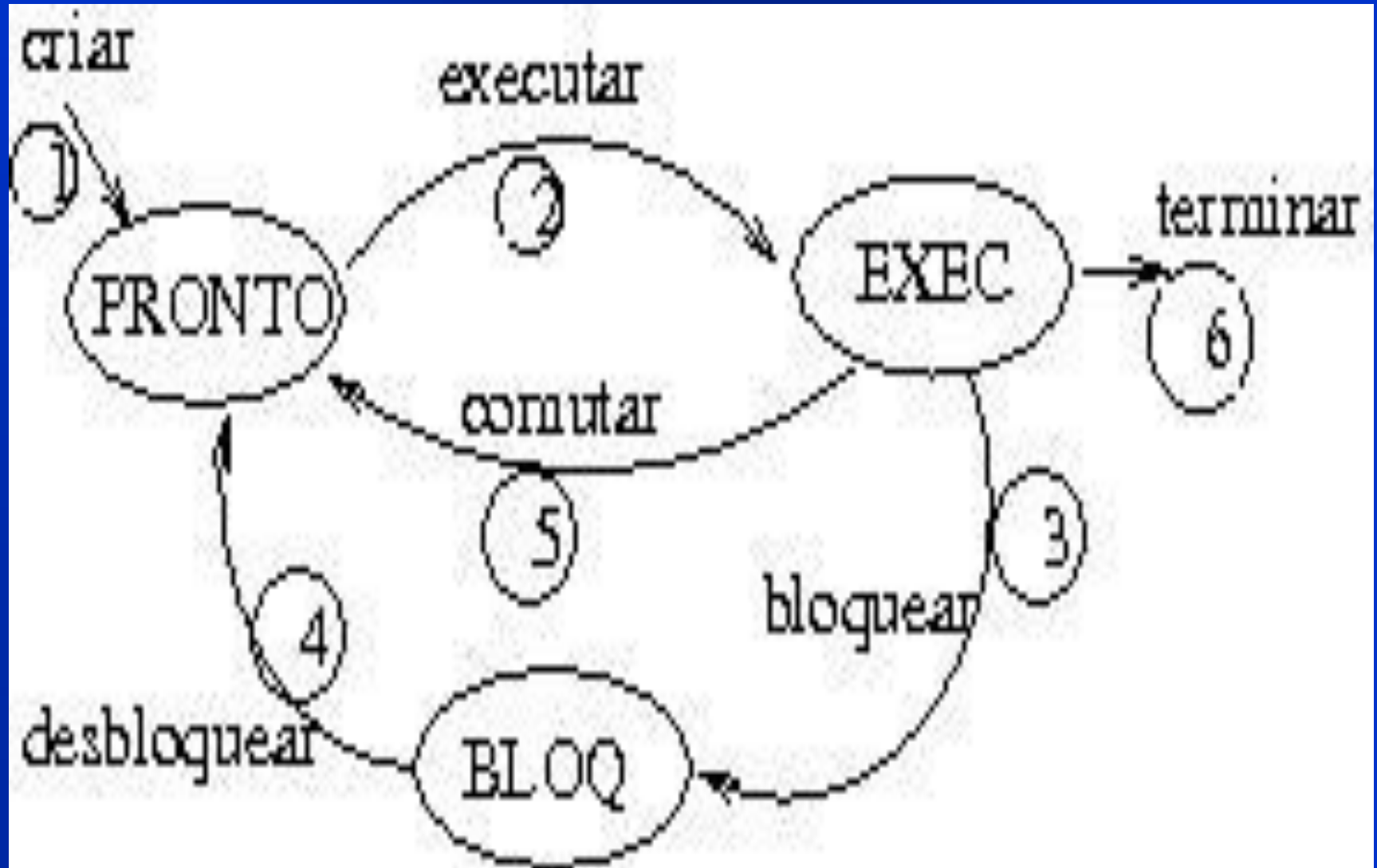
p1



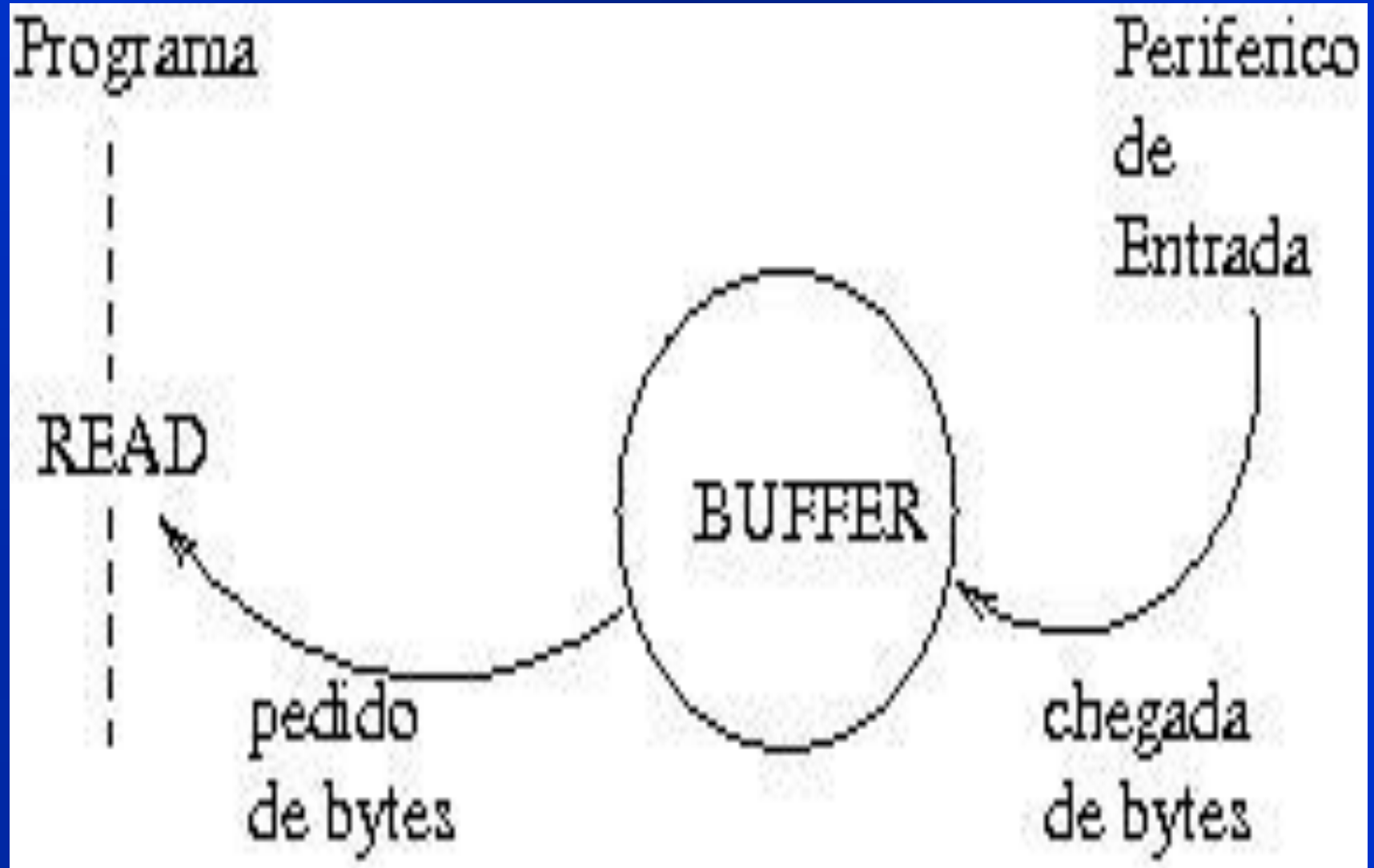
admitindo retorno do resultado
no topo da pilha

– tambem pode ser retornado
num registrador do CPU

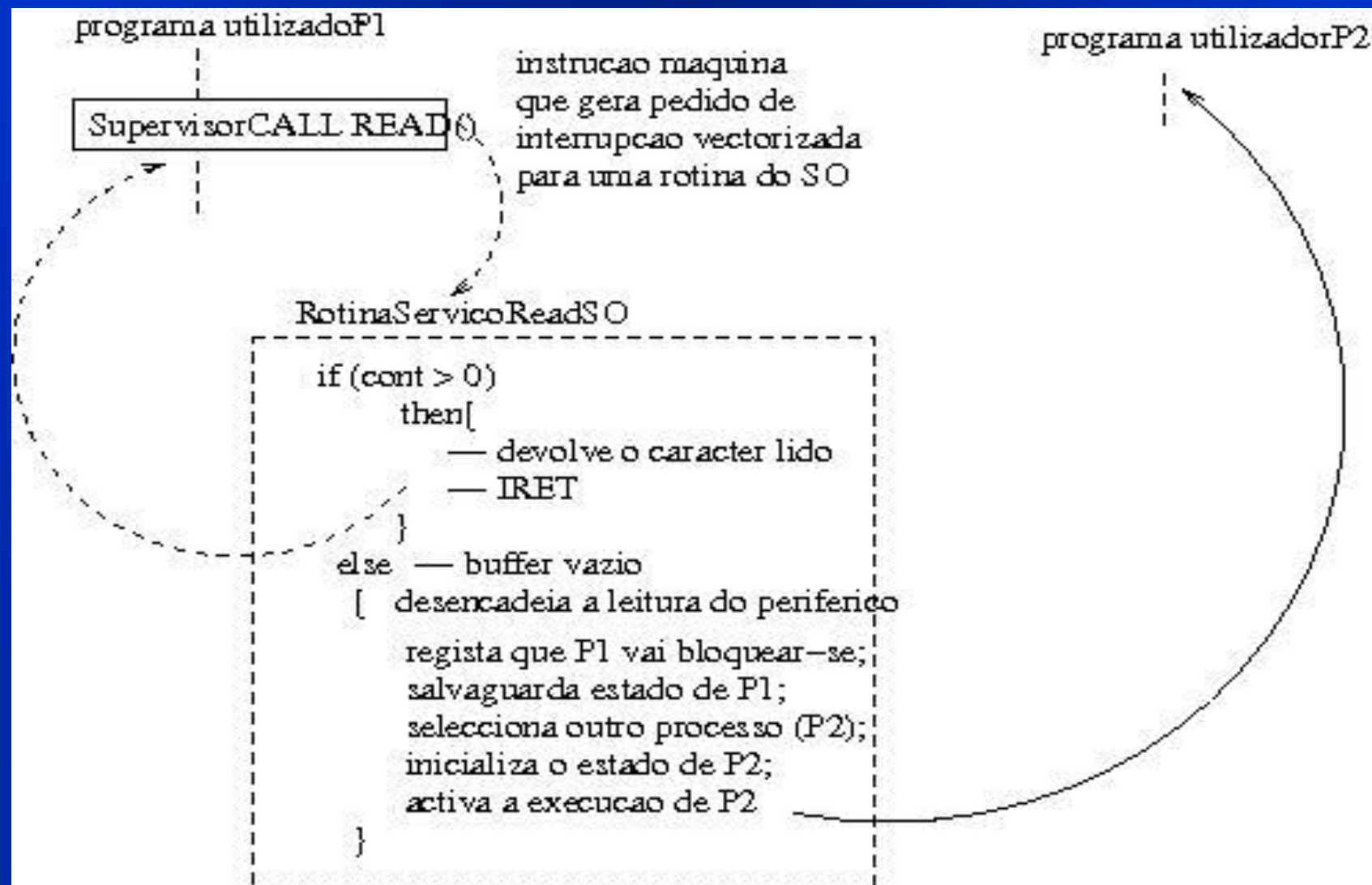
Estados lógicos de um processo



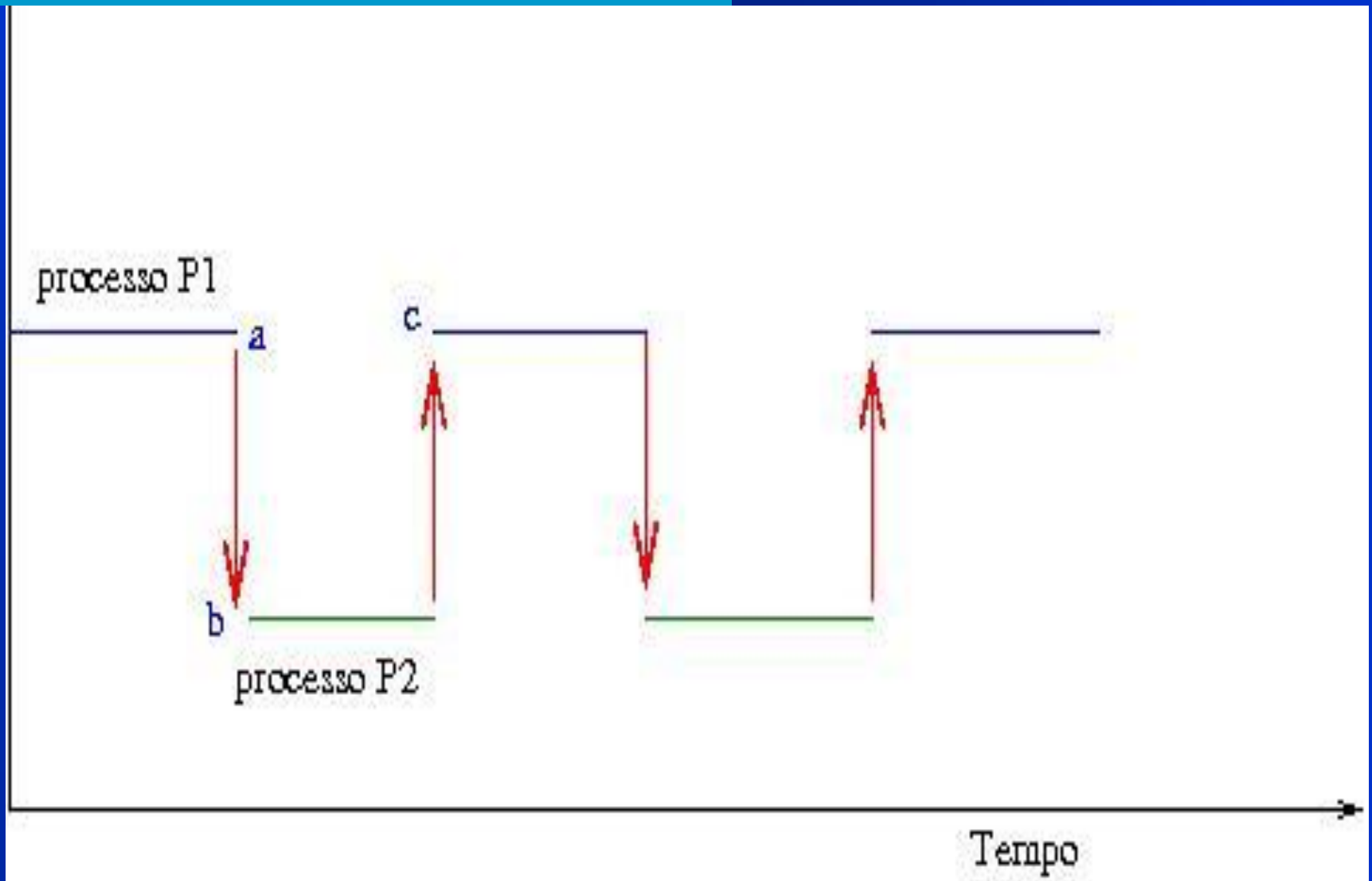
Leitura de dados para *buffers*



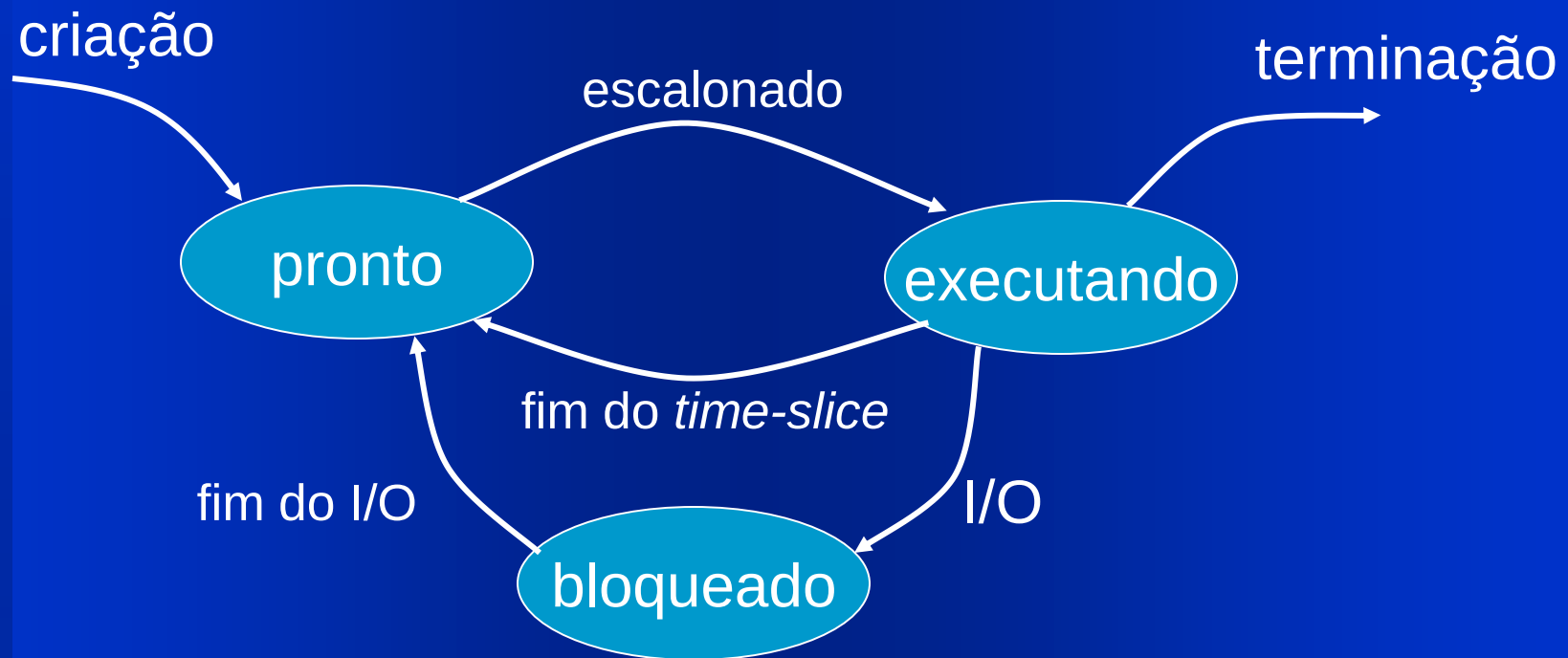
Leitura com bloqueio de P1



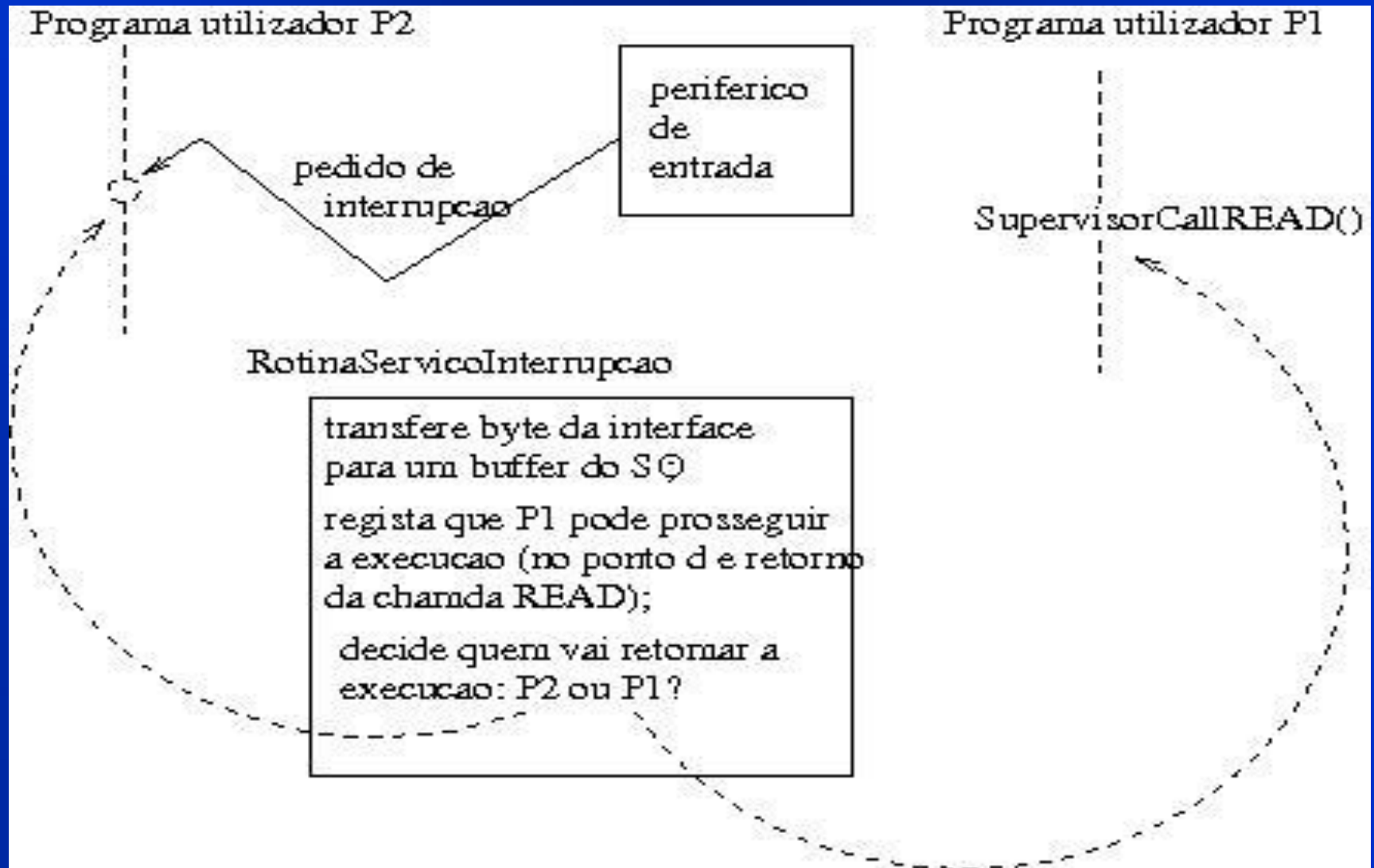
Mudança para Programa P2: (b)



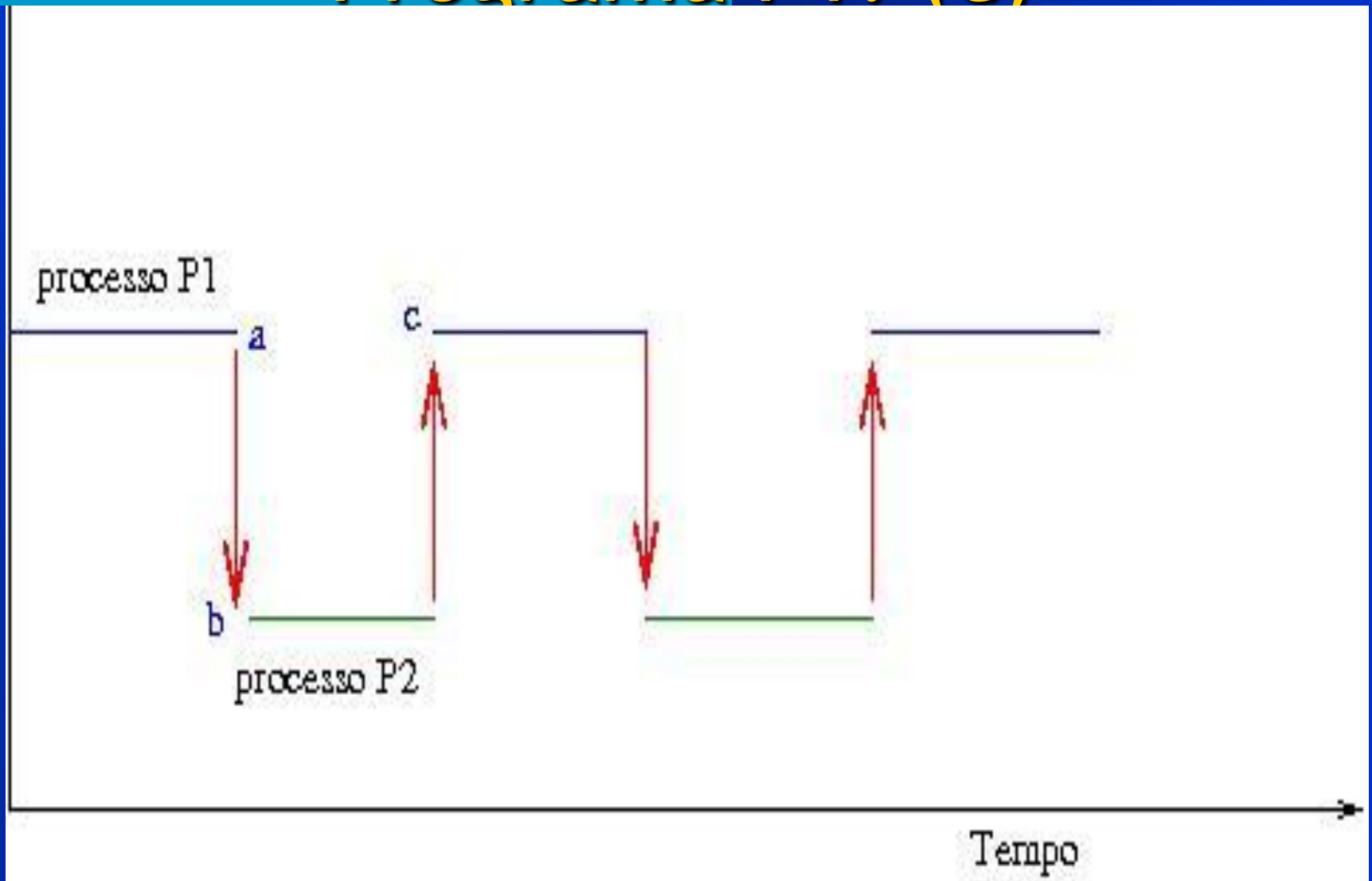
Estados lógicos de um processo



Quando os dados de P1 chegam...



Mudança de volta para Programa P1: (c)



Em resumo...



Contexto de um processo

Informação sobre o estado dos regists. CPU
sobre o mapa de memória
sobre a tabela de canais
estado lógico do processo e em que filas está
identificação e atributos do processo

- Informação dita permanente, não modificada durante a execução (eg PID)
- Informação modificada dinamicamente
(eg Prioridade, Canais abertos, Tempo consumido de CPU)

Distinguir 2 pilhas de execução do processo

Pilha do processo em modo Utilizador

---- quando em modo Utilizador

Pilha do processo quando executa no núcleo

---- quando em modo Supervisor
quando executa código do núcleo
para parâmetros e zonas de activação
das funções do núcleo
apenas acedida pelo SO

Comutação de processos

- quando um P termina
- quando um P se bloqueia explicitamente até que uma condição seja satisfeita
- quando ocorre uma interrupção *Time-slice*
- quando um P mais prioritário deve ser executado em vez do P corrente

Sempre que ocorre uma interrupção, o SO pode decidir comutar de processo

Mecanismo de comutação e despacho

Interrupção →

muda de modo $U \rightarrow S$, comuta para pilha do núcleo onde guarda estado do CPU e endereço de retorno

...

no retorno:

Return_From_Interrupt

antes de efectuar o retorno:

chama uma rotina de Despacho

→ verifica se deve comutar de processo

Acções na comutação de processos

- **Suspend P1:**

- Salvar uma cópia dos registadores do CPU no descritor do processo P1
- Indicar, no descritor de P1, qual a causa da suspensão (aguarda I/O, mensagem, etc.)

- **Activar P2:**

- Escolher o processo mais prioritário da fila
 ''prontos para execução'' : P2
- Carregar os registadores do CPU, a partir dos valores anteriormente salvaguardados no descritor do processo P2

A rotina de Despacho deve ser muito simples e rápida: → é invocada c/ muita frequência limita-se a verificar qual o processo mais prioritário

A estratégia de Escalonamento vai ajustando, com um maior período de intervenção, as prioridades dos processos

Escalonamento (*Scheduling*)

Decide:

que processos devem competir pelo CPU
-- elegíveis para execução
em que instantes e intervalos de tempo

Tendo em conta:

requisitos do regime de utilização

rentabilização e equilíbrio do acesso aos recursos (CPU, Memória, I/O)

Escalonamento em geral

Dois aspectos:

- a) nível de **gestão dos trabalhos** submetidos ao SO para execução
- b) nível de **gestão dos processos**, tarefas já admitidas ao SO, e gestão do seu ciclo de vida e da competição pelos recursos

Parece apenas adequado para sistemas *batch*, mas pode ser considerado mais geral.

a) Nível dos trabalhos submetidos

É um controlo a médio ou longo prazo:

**aceitação de acesso dos utilizadores aos ambientes do SO
identificação, autenticação, autorização dos U
análise dos pedidos dos U pelas descrições dos trabalhos
inicialização e terminação das sessões de trabalho
contabilização (accounting) do uso dos recursos**

Visando objectivos:

de equilíbrio global do uso dos recursos

**de satisfação de um nível de Qualidade de Serviço para os
pedidos dos U**

b) Nível dos processos

É um controlo a curto prazo, da gestão das máquinas virtuais atribuídas a cada processo, com decisões que afectam directamente o uso imediato dos recursos

Nível de escalonamento dos trabalhos

Visa cumprir **objectivos globais:**

face aos pedidos dos U

face ao equilíbrio e rendimento de uso dos recursos

Gere a informação sobre as características e estados dos trabalhos

Estratégias de selecção ou encadeamento de trabalhos para execução

Coordena a fase de iniciação e terminação dos trabalhos, gerindo a afectação de recursos aos trabalhos seleccionados

Em sistemas de recursos partilhados, o escalonamento ao nível dos trabalhos é um factor determinante da qualidade do serviço e da eficiência do sistema.

Exemplo:

se admite uma carga máxima de trabalhos que excede a capacidade do sistema:

**piora a qualidade de serviço
e o rendimento do sistema**

se admite uma carga muito inferior à capacidade do sistema:

**o serviço será bom, mas
o rendimento será baixo, com recursos subutilizados**

Um bom equilíbrio é difícil de atingir, por haver muitos factores envolvidos

A nível global:

uma monitorização contínua do progresso dos trabalhos e
seu comportamento seria o ideal...

mas implicaria muita sobrecarga nas funções do SO.

Assim, há que procurar um equilíbrio.

E há que subdividir os níveis de decisões:

regulações globais e a mais longo prazo

regulações a médio prazo, mais frequentes

e separando ambas dos mecanismos que as implementam a
curto prazo, em particular do despacho e comutação dos
processos.

Factores e critérios

a) Dependentes dos programas U:

exigência de capacidade de Memória

densidade de operações de IO / CPU

utilização de determinados periféricos especiais

tempo de resposta exigido

natureza do trabalho, sua prioridade

tempo estimado de execução (poderá ser conhecido ou não)

Factores e critérios

b) Dependentes dos objectivos de eficiência do sistema:

manter periféricos e processadores ocupados

boa utilização do espaço de memória

má uso de memória pode impedir carregamento de novos processos e originar subutilização de outros recursos

disponibilidade de recursos limitados e de uso exclusivo

Para este efeito conviria ao SO dispor de uma mistura adequada de trabalhos pedidos pelos U, para poder equilibrar bem o uso de todos os recursos

Outros objetivos gerais:

minimizar o tempo médio de resposta dos trabalhos

serviço garantido com tempo de espera limite (deadline)

executar o máximo número de trabalhos / dia – débito ou throughput

justiça ou fairness: não discriminar injustamente entre trabalhos

ex: nenhuma fila de espera de trabalhos longos deve ser indefinidamente protelada, devido a atender um fluxo continuado de trabalhos curtos

Notas

Certos objectivos podem ser contraditórios:

minimizar tempo de resposta favorecendo trabalhos curtos e retardando os longos, pode dar má utilização de certos recursos dedicados (se usados pelos longos)

Deve distinguir-se entre:

decisões globais de exploração do sistema

versus

decisões tomadas automaticamente pelos algoritmos de escalonamento e eventualmente ajustadas dinamicamente a cada momento

Devem impor-se regimes que contrariem tentativas dos U falsearem a caracterização dos trabalhos para obterem mais rápido serviço:

contrariar isto através de mecanismos de monitorização do comportamento real dos trabalhos e estratégias de ajuste dinâmico de prioridades e de controlo forçado da terminação dos trabalhos ou de contabilização do uso do sistema a custo mais elevado

O escalonamento deve ser suficientemente flexível para permitir configurações de acordo com mistura de trabalhos com característica específicas

As soluções

Em teoria, as soluções para um escalonamento óptimo são da classe dos problemas muito difíceis...

Na prática, as soluções adoptadas resultaram de observações experimentais baseadas em medições das características de desempenho:

- dos programas U

- dos níveis de utilização dos recursos

Por outro lado:

- os algoritmos de escalonamento devem ser

 - simples

 - eficientes (com sobrecarga reduzida)

 - fáceis de ajustar

Escalonamento

Uma política global de utilização dos recursos

Idealmente: deveria intervir sempre que o estado de um recurso mudasse...

Na prática, exige-se um compromisso entre uma boa rentabilização do uso dos recursos e

uma pequena penalização do desempenho, devida ao tempo de execução das funções do escalonador (ocupando o CPU...)

Habitualmente, o SO

**ajusta as prioridades dos processos
dinamicamente, ie durante a sua execução.
em função do seu comportamento**

Tendo em conta múltiplos aspectos:

eg, tempo de resposta ao Utilizador

evitar monopolizar o uso do CPU

optimizar operações de I/O

estado corrente de ocupação de memória

etc.

Objectivos do escalonamento

Para os programas Utilizadores:

cumprir os requisitos de tempo de resposta de cada aplicação

- sejam *jobs batch*, U interactivos ou dispositivos de tempo real...

Para o Sistema:

rentabilizar a utilização global dos Recursos
aumentar o débito de execução dos
processos ao longo do tempo

Exemplos de algoritmos de escalonamento

Primeiro o trabalho mais antigo

First In – First Out

atende pela ordem de chegada dos pedidos
só considera o tempo de espera na fila de entrada
para misturas de trabalhos de diferentes tipos:

os mais longos são menos prejudicados

face aos mais curtos...

pode assim, originar:

injustiças

má utilização de recursos

Exemplos de algoritmos de escalonamento

Primeiro o trabalho mais curto

Shortest Job Next (SJN)

considera o menor tempo estimado de serviço

Surgiu nos sistemas batch não interactivos

- visava aumentar o débito de trabalhos
- podia prejudicar a execução dos trabalhos mais longos, que eram preteridos
- sugeriu formas de os tentar compensar para evitar privação (*starvation*):
 - avaliar o tempo de espera e ajustar as prioridades em função de um factor de envelhecimento (*aging*)

Estimativa do tempo de cada trabalho:

- especificada pelo U na descrição do trabalho
- estimada pelo sistema, em função de uma caracterização de trabalhos em categorias, com tempos típicos de execução associados
- Reavaliada em função de um histórico de execuções anteriores

Sempre sujeita a erros nas previsões.

Exemplos de algoritmos de escalonamento

*Primeiro o trabalho com maior
factor de resposta*

$$R = w + ts$$

w tempo de espera de jobs na fila

ts tempo de serviço

Factor de resposta:

$$\frac{R}{ts} = \frac{w + ts}{ts} = \frac{w}{ts} + 1$$

Factor de espera: w / ts

Outros aspectos que ajudam a equilibrar...

Uso de preempção ou não

a) Executa até se completar, sem multiprogramação

b) Executa até se completar ou se bloquear explicitamente (com multiprogramação)

c) Preempção por *Time-slice* + b)

d) Preempção por *Time-slice* e quando ocorre qualquer mudança de estado de outros processos

Imposição de um limite de tempo total de execução (TIME-OUT) frequente em sistemas batch

Outros aspectos que ajudam a equilibrar...

Uso de prioridades

dependentes das características do trabalho

a) especificadas pelo U

- prioridade externa estática

 - exigência de memória

 - tempo estimado de serviço

 - periféricos especiais

b) do seu comportamento dinâmico

- prioridade interna dinâmica

 - tempo de execução decorrido

 - perfil *CPU versus IO - bound*

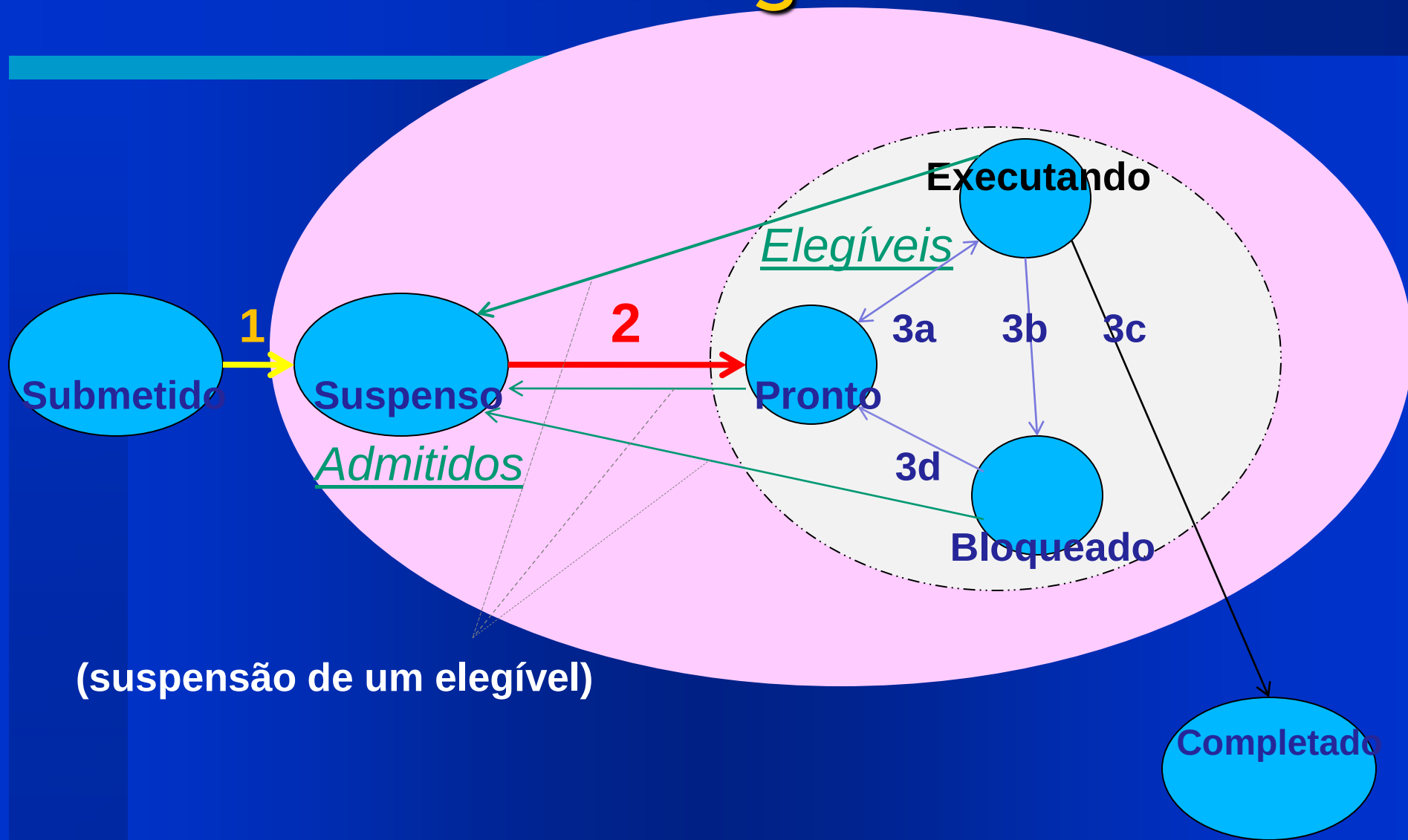
Um modelo geral

Escalonamento de alto nível: subsistema de admissão de trabalhos; manipula uma fila de trabalhos aguardando a entrada no sistema e selecciona os que podem ser iniciados e prosseguirem

Escalonamento de médio nível: gestor global ou coordenador, dos recursos do sistema; manipula uma fila de trabalhos previamente iniciados e controla o seu progresso, ajustando a carga do sistema de modo a satisfazer objectivos globais

Escalonamento de baixo nível: trata do despacho e comutação dos processos e gere as suas transições de estados

Modelo geral



Objectivos: - bom tempo de resposta

→ limitar o número de trabalhos admitidos

- impedir degradação (thrashing)

→ limitar o número de processos elegíveis
multiprogramados

Inicialmente o Controlador da Carga analisa os trabalhos *Submetidos* e admite-os se tal for compatível com a carga admitida pelo SO, com base numa política global de controlo da carga. (→ garantias de tempo de resposta)

Os trabalhos admitidos passam ao estado *Suspenso* onde aguardam selecção pelo Escalonador de Trabalhos.
Se têm os R requeridos, passam ao estado *Pronto*.

Os processos *Pronto* ficam sujeitos ao Escalonador de Processos → o processo mais prioritário para *Executando*

Controlo de carga: eg associar a cada tipo de trabalho (U) - peso em unidades (função R)

1 : U normal; $\frac{1}{2}$: U restringido; $1 \frac{1}{2}$: U batch

Os utilizadores organizados em Classes, cada com um máximo de unidades permitidas.

→ se o max é atingido: não admite mais U
dessa Classe

Em cada Classe: duas subclasses de U:

- os não privilegiados
- os privilegiados podem provocar a suspensão forçada de um não privilegiado

A cada momento: certos processos escolhidos para Elegíveis durante um certo período (eg 2 segundos)
→ passam ao estado Pronto

Processos *Suspensos*:

- múltiplos níveis (filas) de prioridade:
- o processo de mais alta prioridade passa a *Pronto* se houver os Recursos exigidos para o executar
- um processo no estado *Suspense* pode originar a suspensão de um processo Elegível:
 - para garantir tempo de resposta a processos interactivos que estejam *Suspensos*
 - para eleger um *Suspense* cujos Recursos fiquem disponíveis

Comutação de processos: Ajuste dinâmico das prioridades

Sistemas interactivos - *Time-sharing*

Atribuição de um *Time-slice* ou *Quantum*

se não usa prioridades

regime de utilização rotativa do CPU

combinado com prioridades

usa a ocorrência do fim do *Time-slice*

para reavaliar qual o processo mais prioritário

Valores típicos do *Time-slice*: 20, 50, 100 ms

O *Time-slice* pode ser constante ou variável consoante o comportamento do processo.

Notas:

Se um processo não esgota todo o *Time-slice*:

terminar

bloquear-se:

o que fazer quando for reactivado:

dar-lhe um novo *Time-slice* ou apenas o restante do anterior *Time-slice*?

Se um processo esgotou todo o *Time-slice*:

o que fazer ao voltar a ser despachado:

dar-lhe o mesmo *Time-slice*

ou um valor diferente ?

A estratégia rotativa pura (*round robin*):

para N processos

no pior dos casos: um tempo de resposta por processo, da ordem de $N * \textit{Time-Slice}$

Para grandes valores de N, o tempo de resposta pode deixar de ser aceitável para alguns tipos de aplicações.

Soluções?

- usar prioridades
- usar computadores dedicados
- usar servidores com muitos processadores

Ajuste dinâmico de prioridades

Dar preferência aos processos *IO-bound*
aumentar-lhes a prioridade, em função da
sua actividade recente de IO

ir ajustando isto, conforme o processo
vai alterando o seu comportamento:

... fases mais IO-bound,
... fases mais CPU-bound...

Contribui para otimizar:

- o tempo de resposta ao U
- o rendimento de utilização dos recursos

Exemplo:

avaliar a prioridade em função do tempo de CPU já consumido por cada processo

(também beneficia *IO-bound*, se aumentar prioridade quando o tempo de CPU é menor)

O Escalonador reavalia as prioridades periodicamente e reposiciona os processos nas filas associadas ao estado Pronto.

Múltiplas filas do estado *Pronto*

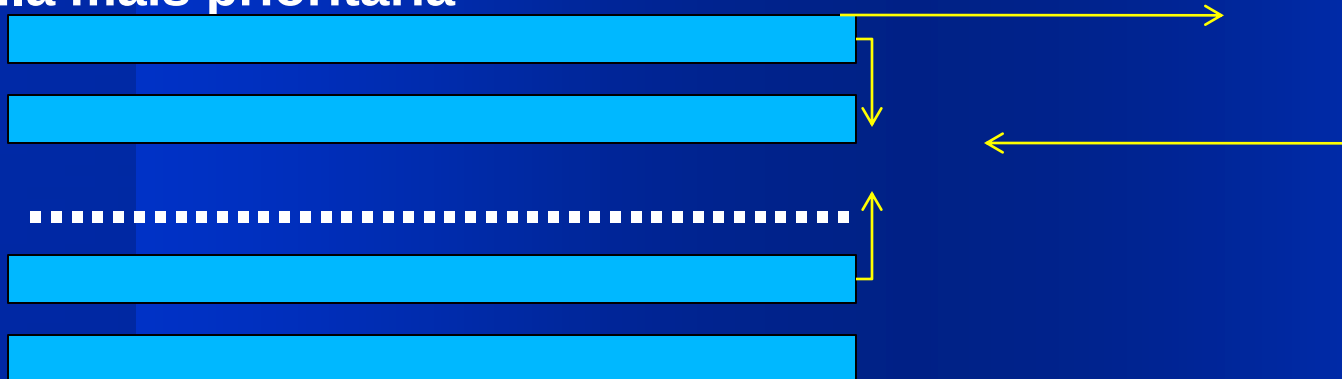
Cada fila corresponde a um nível de prioridade

eg 32 níveis de prioridades com 32 filas

Algoritmos caracterizados por:

- em que fila pôr um processo criado
- como elevar ou reduzir o nível de prioridade de um processo, mudando-o de fila
- em que fila pôr um processo recém- desbloqueado

Fila mais prioritária



Ajuste dinâmico do valor do *Time-slice*

A ideia:

para processos *CPU-bound*:

reduzir a sua prioridade, mas dando valores progressivamente mais altos de *Time-slice*

Por um lado:

dá-lhes menor preferência do que a *IO-bound*

Por outro lado:

permite que terminem com menor número de comutações de processos

Exemplo:

TS: *Time-Slice*

Fila mais prioritária

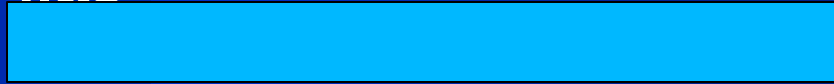
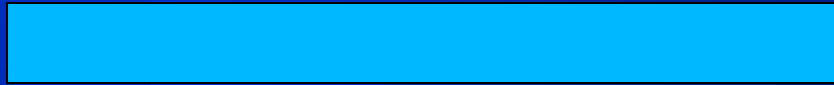
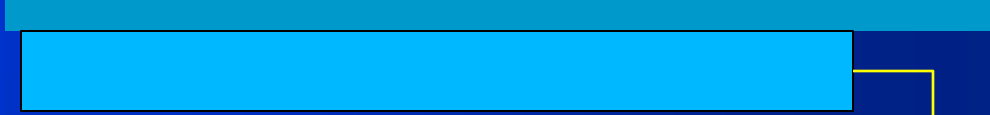
TS1 = 20 ms

Fila2

TS2 = 10* TS 1 = 200 ms

Fila3

TS3 = 100* TS1 = 2000 ms



Aprensão forçada do CPU

Preemption

Após cada evento de mudança de estado de qualquer processo:

invocar a rotina de Despacho para activar o processo mais prioritário

Necessário, por ex, em SO de tempo real.

Com preempção só por *Time-slice*:

o processo corrente só perde o controlo

ao terminar, ao bloquear-se, ou por *Time-slice*

Com preempção:

o processo corrente pode perder o controlo logo que outro processo mais prioritário surja

Nota:

compromisso entre

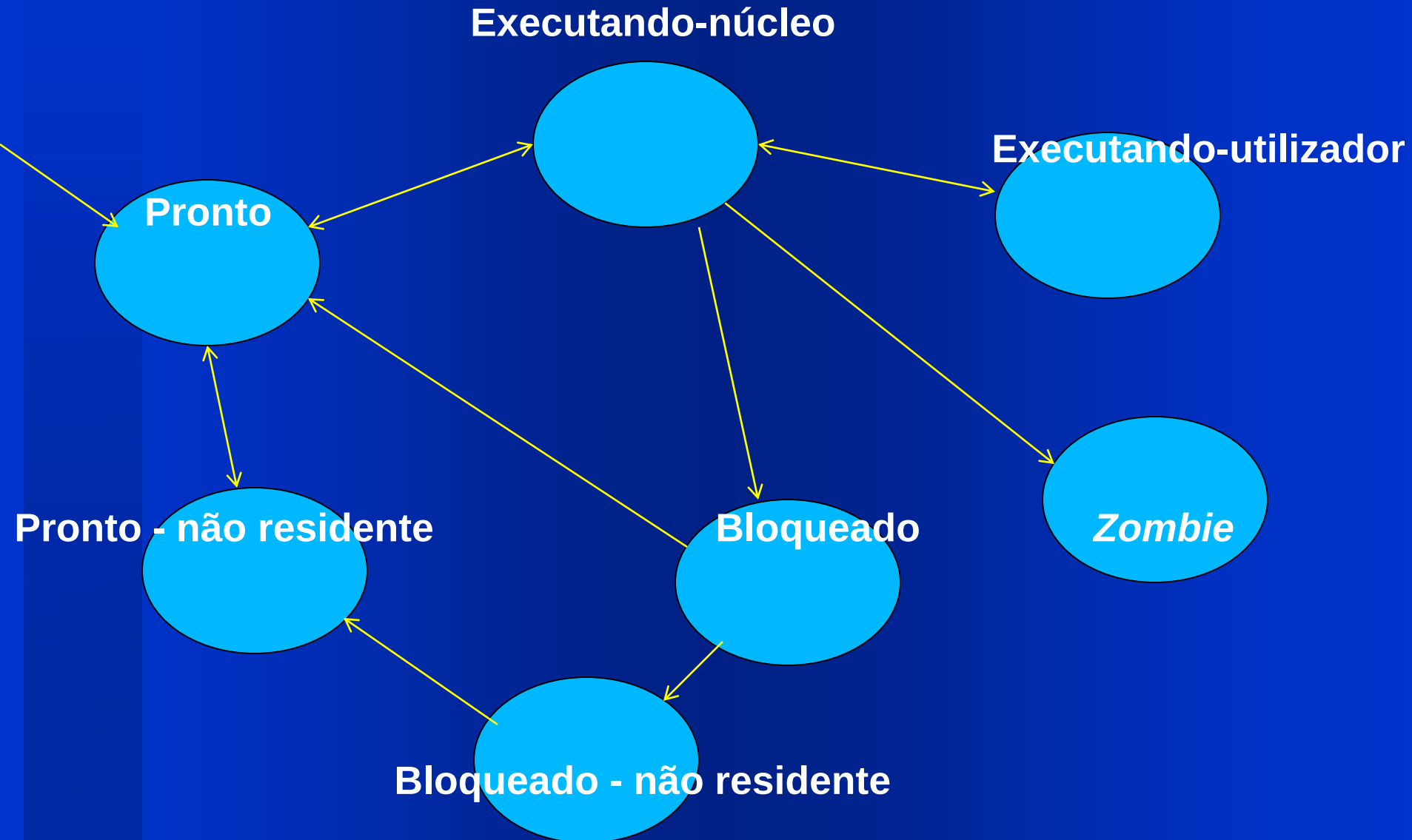
maior justiça da preempção

versus

**maior penalização do desempenho, pela
sobrecarga do seu tratamento pelo SO
e pelo maior número de comutações de
processos**

**Exemplo: garantir que cada processo tem um
tempo mínimo sem preempção, também
para evitar *trashing*...**

Estados dos processos no Unix



Pronto → Executando-núcleo
quando for o de maior prioridade

Executando-utilizador
para executar o Programa

Executando-utilizador $\leftrightarrow$ Executando-núcleo
por cada chamada ao SO e por cada interrupção

Executando-núcleo → Executando-utilizador
a rotina de Despacho testa se processos maior prioridade
(Preempção)

Escalonamento no Unix

Ajuste dinâmico de prioridades

preferência aos IO-bound

sem protelar demais os CPU-bound

Escalonamento com preempção, quando em modo utilizador

testado sempre que um processo Bloqueado → Pronto
podendo apreender o CPU ao processo Executando
a rotina de Despacho é invocada sempre que há o retorno
da execução em modo núcleo para modo utilizador

Escalonamento sem preempção, quando em modo núcleo

(na maioria dos sistemas): não comuta a execução de um
processo em modo núcleo, mesmo que um processo mais
prioritário se desbloquear

mas permite sucessivas interrupções, ainda em modo núcleo

Algoritmo de escalonamento no Unix

Cada sistema Unix pode ter variações nalguns aspectos.

O sistema Linux tem um algoritmo bastante diferente.

Múltiplas filas . Cada fila : uma gama de valores prioridades

Cada processo P – um valor inteiro de prioridade

valor menor == maior prioridade

Após cada Time-slice:

P é colocado na cauda da fila da sua prioridade

a rotina de Despacho é invocada: escolhe o mais prioritário

iguais prioridades → esquema rotativo (round robin)

Time-slice: da ordem dos 100 ms:

compromisso entre regime de time-sharing versus
sobrecarga da comutação de contextos

Prioridades podem ter valores positivos (para os processos em modo utilizador) ou negativos (para o modo núcleo)

Prioridade em modo utilizador:

varia dinamicamente em função do tempo de uso do CPU
maior prioridade se P usou menos tempo de CPU recentemente

Cálculo da prioridade corrente do processo P:

$$P.pri = PrioridadeBase + P_cpu / 2 + nice$$

PrioridadeBase: constante do SO para cada tipo de processo

P.cpu: acumula o tempo de utilização do CPU

nice: definido pelo Utilizador (valor pré-definido de 0)

Periodicamente: os processos são recolocados nas filas

a) (10 ms) $P.cpu = P.cpu + 1$ para o processo em execução

b) (1 s) $P.pri$ e $P.cpu$ de todos os processos são reavaliados

$P.cpu = P.cpu / 2$ (factor esquecimento da utilização do CPU)

Os processos com maior tempo de CPU recentemente

recebem menor prioridade

beneficiando os processos IO-bound

na selecção pela rotina de Despacho

quando passam tempo Bloqueados

são preferidos na passagem para Executando

Um processo que se bloqueia em modo núcleo

recebe um valor negativo, uma maior prioridade

dependendo da causa de bloqueio

para preferir processos que acedem a recursos críticos

que devem libertar o mais rapidamente possível

Um processo desbloqueado continua em execução em modo núcleo com o mesmo valor de prioridade que lhe foi atribuído ao bloquear-se

Prioridades negativas mais altas: para processos bloqueados no núcleo, em recursos muito críticos

Prioridades negativas mais baixas: idem, mas recursos menos críticos

Prioridades positivas mais baixas: os processos utilizadores mais prioritários (com pouco tempo de CPU)

Prioridades positivas mais altas: os processos menos prioritários.

**Prioridade modo utilizador para modo núcleo:
quando um P se bloqueia.**

**Prioridade modo núcleo para modo utilizador:
automaticamente calculada quando o processo retorna a modo utilizador**

Algoritmo de escalonamento Linux

No Unix, para muitos processos, a reavaliação de prioridades todos os segundos → muita sobrecarga do SO

No Linux: o conceito de Época → O tempo de execução é dividido em épocas sucessivas:
uma época termina quando todos os P prontos tiverem completado o seu *Time-slice* ou tenham prescindido dele

No início de uma época:

o Time-slice para cada P depende do tempo restante da época anterior: $TS = TS_base + TS_não_usado_antes / 2$
(TS_base definido pelo SO)

Cálculo das prioridades: ...

**Cálculo de prioridades: inteiros mais altos = maior prioridade
considera o tempo que resta do TS na época corrente**

$$\text{P.prio} = \text{PrioridadeBase} + \text{TS_corrente_por_utilizar} - \text{nice}$$

A rotina de Despacho selecciona o processo com a maior prioridade.

**os processos IO-bound terão valores maiores de TS
se estiveram bloqueados na época anterior**

**Os valores de TS são reavaliados uma vez por cada época
A duração da época aumenta quando aumentam os P em
execução simultânea.**

**Mas cada invocação da rotina de Despacho deve pesquisar o
processo mais prioritário, na fila de Prontos:**

Melhorar: gestão de múltiplas filas de prioridades dinâmicas

Processos Tempo Real no Linux:

com prioridades fixas mais altas do que os outros em modo utilizador

e sempre seleccionados em primeiro lugar face aos outros

Mas, como o SO Linux não tem preempção quando em modo núcleo, não pode garantir o cumprimento de prazos de execução (*deadlines*).

Referências:

Modern Operating Systems, A. Tanenbaum
Sistemas Operativos, J. Alves Marques et al.
Sistemas de Exploração, J. Cardoso e Cunha