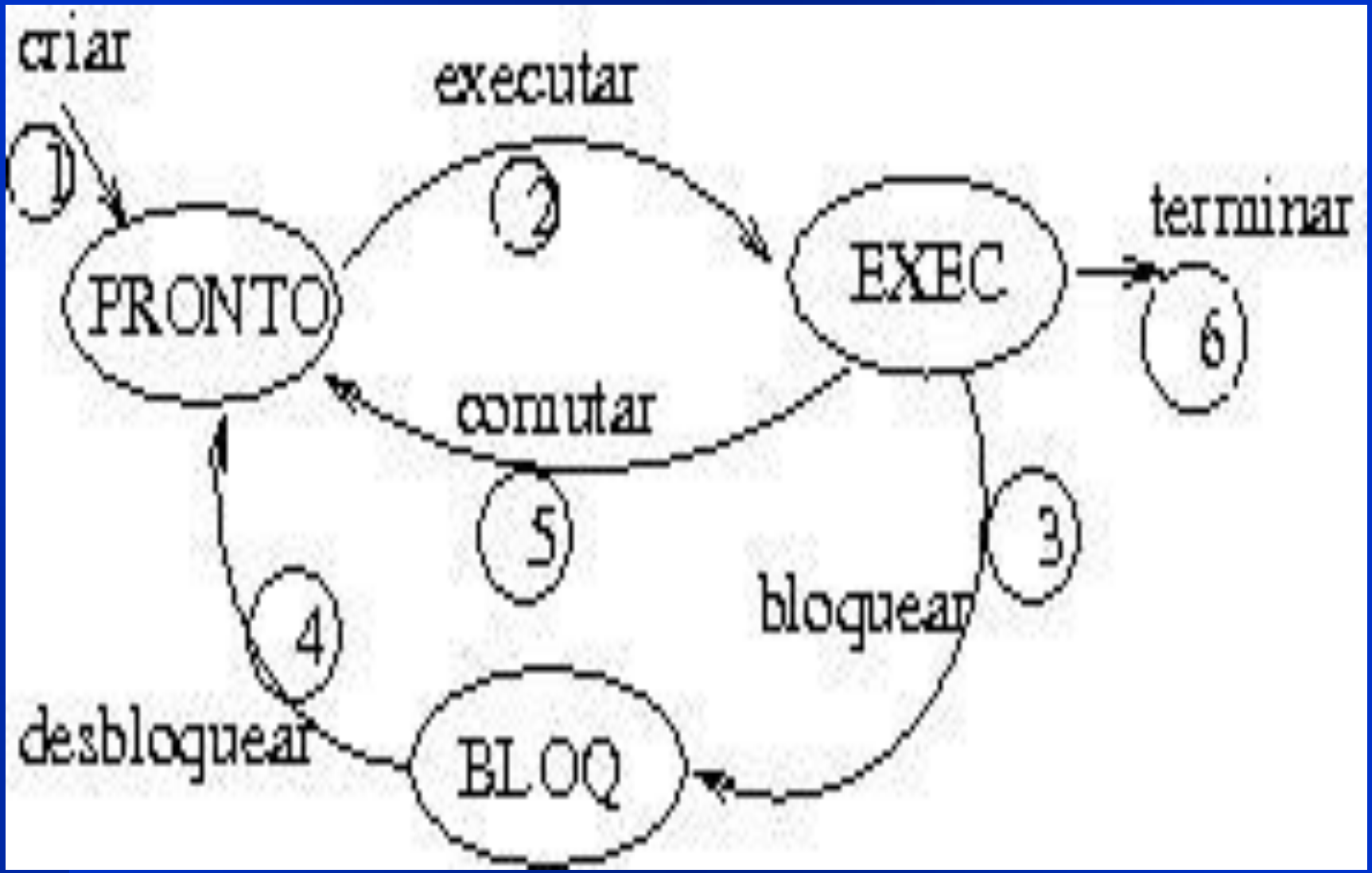
# Estados lógicos de um processo



# Criação de um processo

- **Define:**
  - **O mapa de memória (código, dados, pilha)**

# 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.)**

# 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 (PC→ start-address, SP→topo de pilha, etc)**

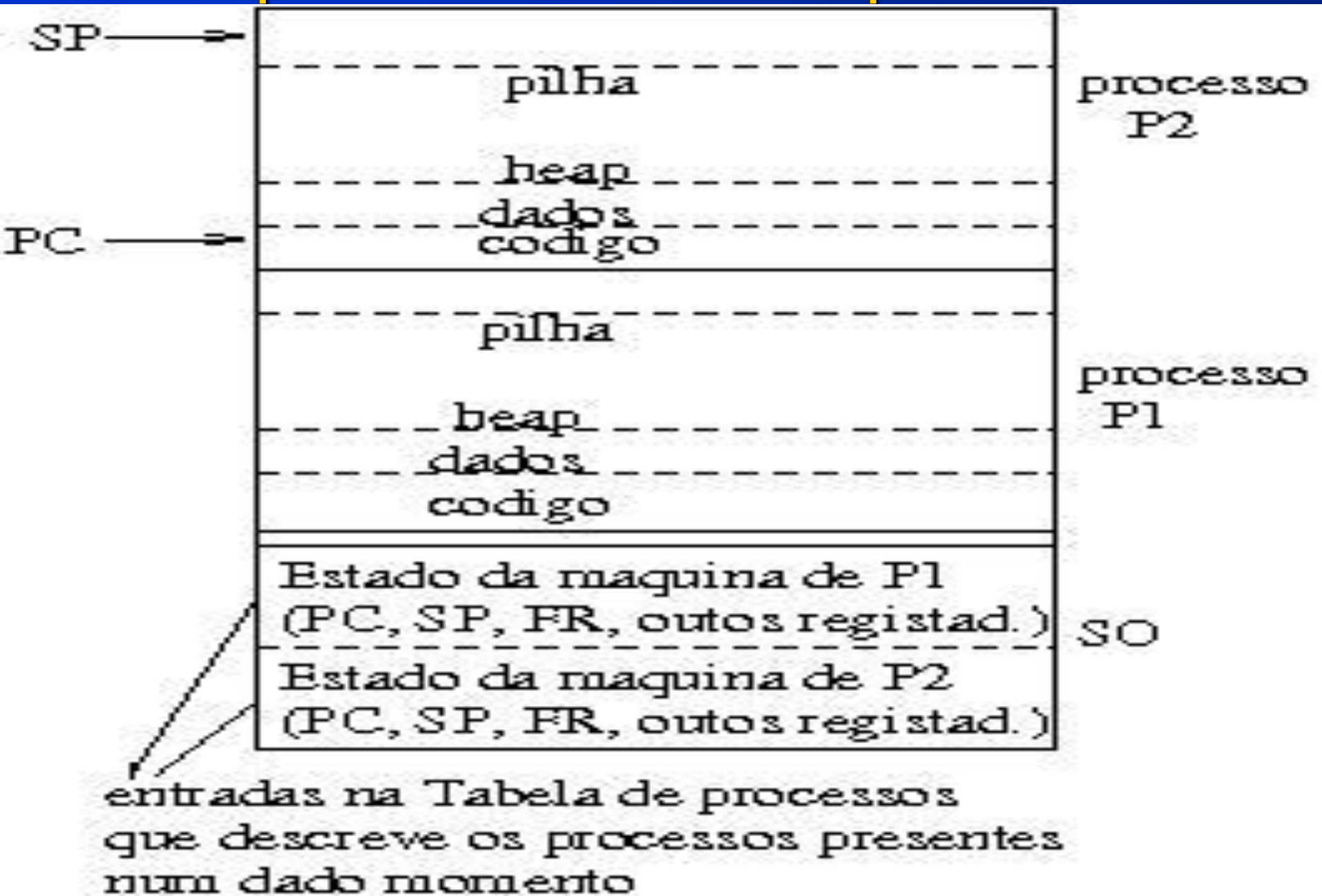
# Criação de um processo

- **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 (PC→ start-address, SP→topo de pilha, etc)**
  - **Os canais iniciais de entrada/saída**

# 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 (PC→ start-address, SP→topo de pilha, etc)**
  - **Os canais iniciais de entrada/saída**
  - **Uma entrada numa Tabela de Processos, interna ao SO, com o Descritor do Processo**

# Mapas de memória dos processos



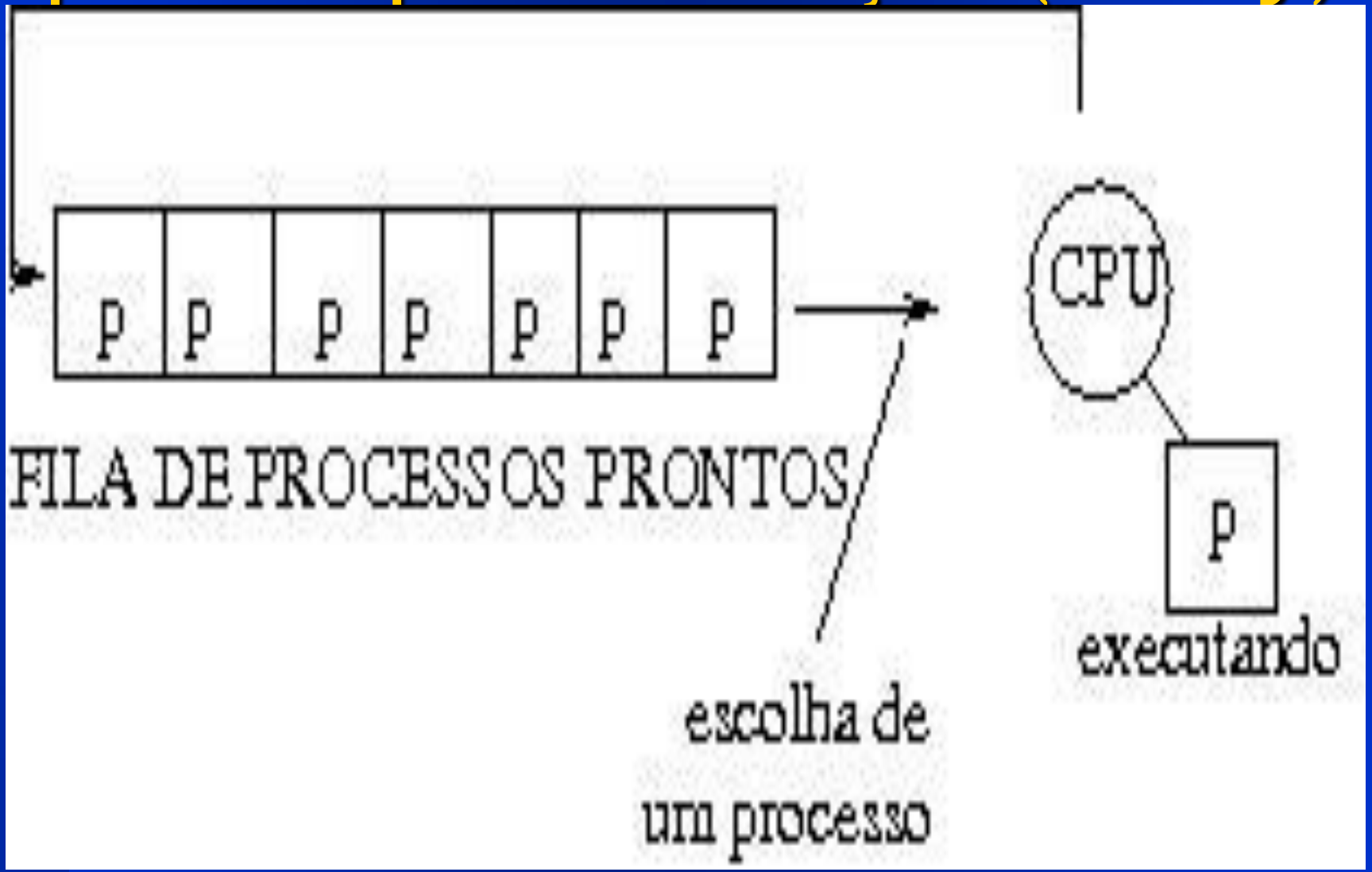
# Fila de processos

- **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**

# Fila de processos

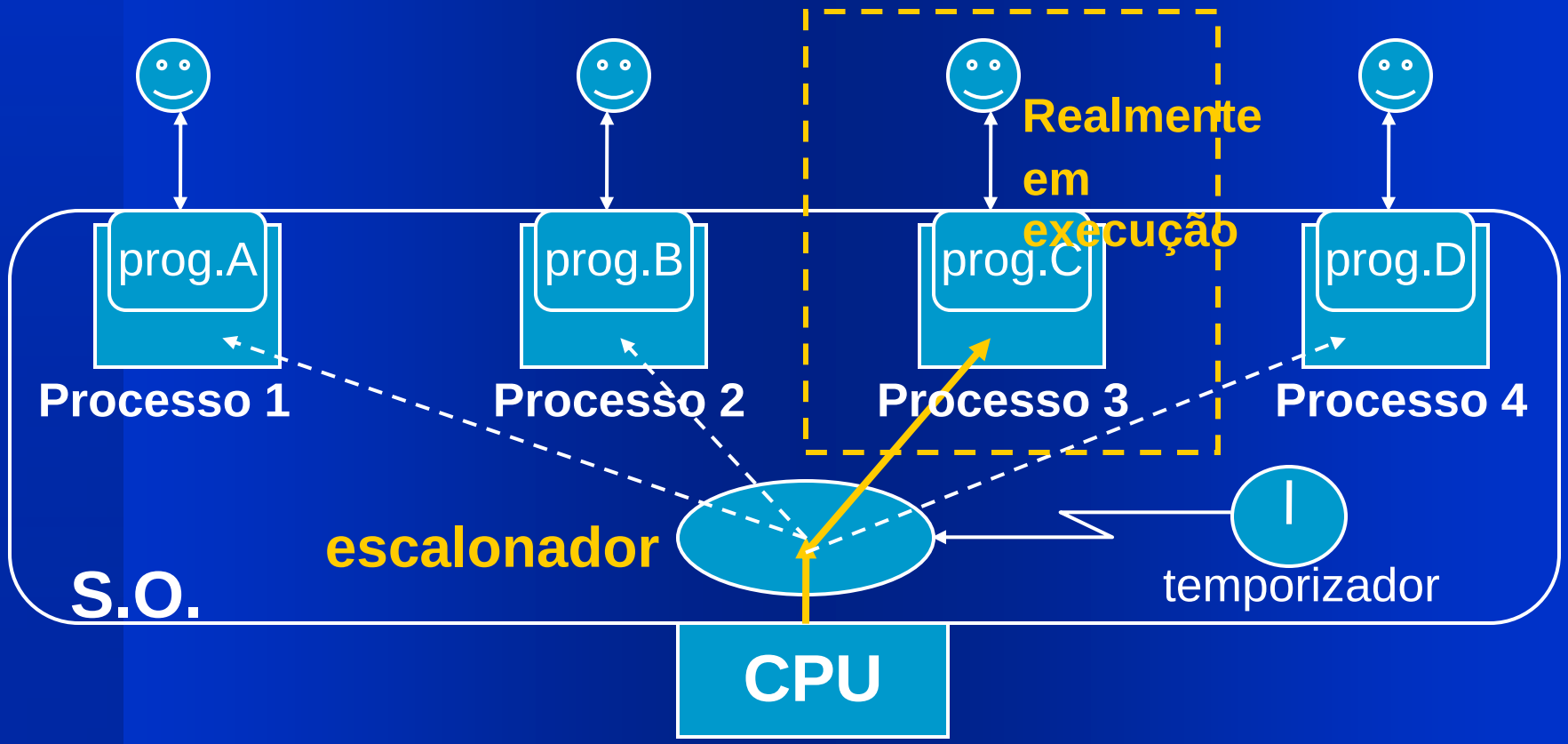
- **Uma vez criado,  o processo é posto numa fila de processos, à espera de ser activado para execução:**
  - **Cada elemento da fila tem o identificador do processo (*Process Identifier*), que indica a respectiva entrada da Tabela de Processos**
  - **A fila é ordenada segundo as prioridades dos processos**

# Fila de processos prontos para execução (*Ready*)



# Multiprogramação

- **Exemplo: 4 processos competem pelo CPU**
- **Se há pedidos de I/O bloqueantes ou o fim de um processo, o SO atribui o CPU a outro processo**



# Regime de *Time-sharing* do CPU

- **Um programa CPU-*bound* poderia monopolizar o CPU:**
  - **Mau para o uso dos periféricos**
  - **Mau para a interacção com o utilizador**

# Regime de *Time-sharing* do CPU

- **Um programa CPU-*bound* poderia monopolizar o CPU:**
  - **Mau para o uso dos periféricos**
  - **Mau para a interacção com o utilizador**
- **Há que partilhar o tempo de CPU**

# Regime de *Time-sharing* do CPU

- **Um programa CPU-*bound* poderia monopolizar o CPU:**
  - **Mau para o uso dos periféricos**
  - **Mau para a interacção com o utilizador**
- **Há que partilhar o tempo de CPU**
- ***Time-slice:* cada programa dispõe, no máximo, de um determinado intervalo de tempo, de cada vez, sendo-lhe o CPU apreendido pelo SO, se gastar todo esse tempo...**

# Estados lógicos de um processo

# Acções na comutação de processos

- **Suspender P1:**
  - **Salvaguardar uma cópia dos registadores do CPU no descritor do processo P1**

# Acções na comutação de processos

- **Suspender P1:**
  - **Salvaguardar 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.)**

# Acções na comutação de processos

- **Suspender P1:**
  - **Salvaguardar 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:**

# Acções na comutação de processos

- **Suspender P1:**
  - **Salvaguardar 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**

# Acções na comutação de processos

- **Suspender P1:**
  - **Salvaguardar 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**

# Regime de Múltiplos utilizadores

- **Suportado por um regime *time-sharing***

# Regime de Múltiplos utilizadores

- **Suportado por um regime *time-sharing***
  - **Um programa "vê" um espaço de memória privado, recolocado e protegido (e só pode aceder a essa memória)**

# Regime de Múltiplos utilizadores

- **Suportado por um regime *time-sharing***
  - **Um programa "vê" um espaço de memória privado, recolocado e protegido (e só pode aceder a essa memória)**
  - **Só o SO acede realmente aos periféricos, partilhados pelos vários programas**

# Regime de Múltiplos utilizadores

- **Suportado por um regime *time-sharing***
  - **Um programa "vê" um espaço de memória privado, recolocado e protegido (e só pode aceder a essa memória)**
  - **Só o SO acede realmente aos periféricos, partilhados pelos vários programas**
- **Cada utilizador *vê* uma máquina virtual**
  - **No disco tem um conjunto de aplicações**

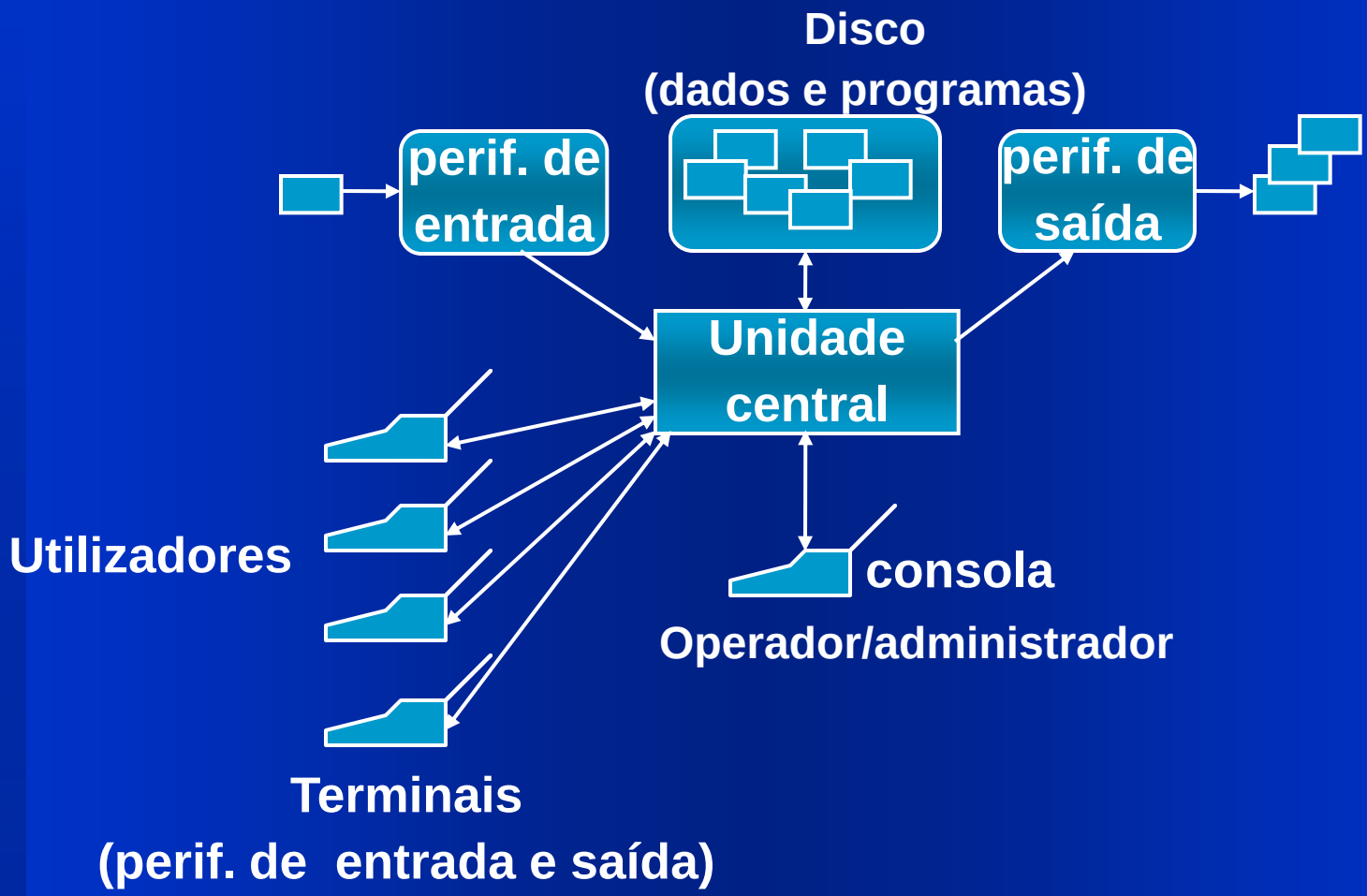
# Regime de Múltiplos utilizadores

- **Suportado por um regime *time-sharing***
  - **Um programa "vê" um espaço de memória privado, recolocado e protegido (e só pode aceder a essa memória)**
  - **Só o SO acede realmente aos periféricos, partilhados pelos vários programas**
- **Cada utilizador *vê* uma máquina virtual**
  - **No disco tem um conjunto de aplicações**
  - **No disco guarda um conjunto de dados**

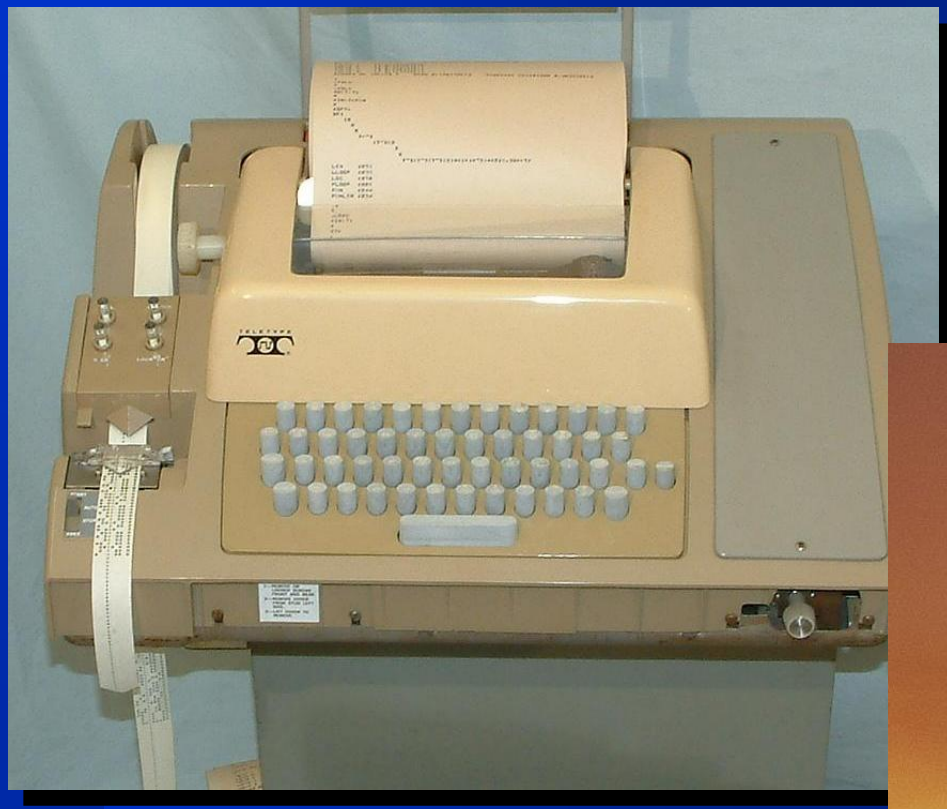
# Regime de Múltiplos utilizadores

- **Suportado por um regime *time-sharing***
  - **Um programa "vê" um espaço de memória privado, recolocado e protegido (e só pode aceder a essa memória)**
  - **Só o SO acede realmente aos periféricos, partilhados pelos vários programas**
- **Cada utilizador *vê* uma máquina virtual**
  - **No disco tem um conjunto de aplicações**
  - **No disco guarda um conjunto de dados**
  - **O utilizador interage dando comandos ao ´´sistema´´ via um *shell (interpretador de comandos)***
    - **pede ao SO para executar programas, p.ex.**

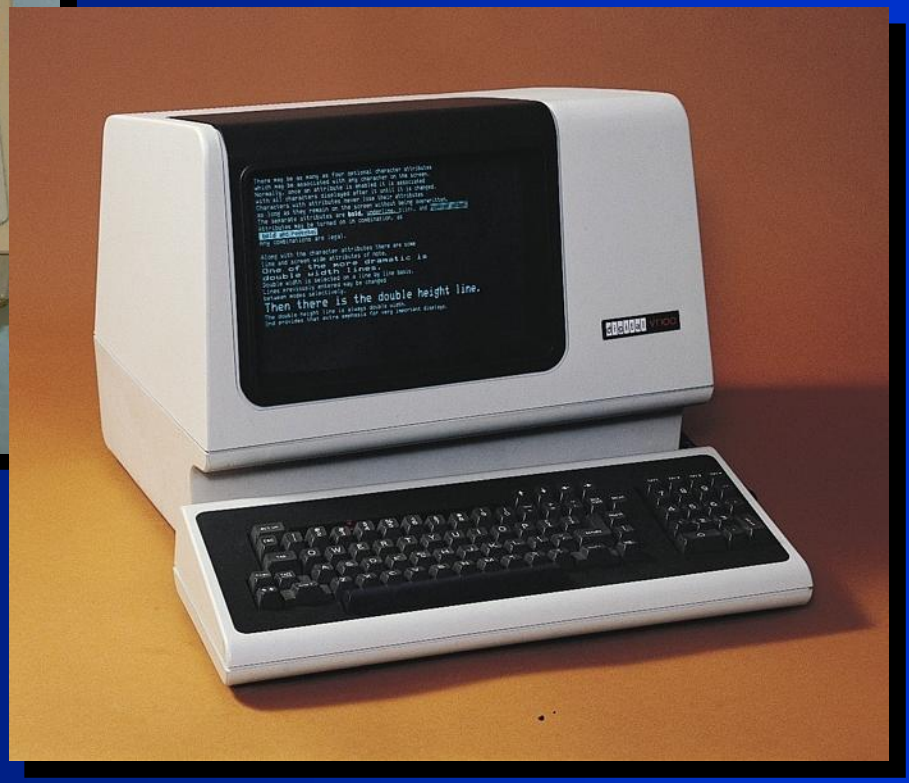
# Sistema interactivo com *time-sharing*



# Exemplos de terminais antigos



Teletype 33-ASR
1965



Digital (DEC) vt100
1978

# Na génese do Unix (Bell-Labs, 1970)

Ritchie

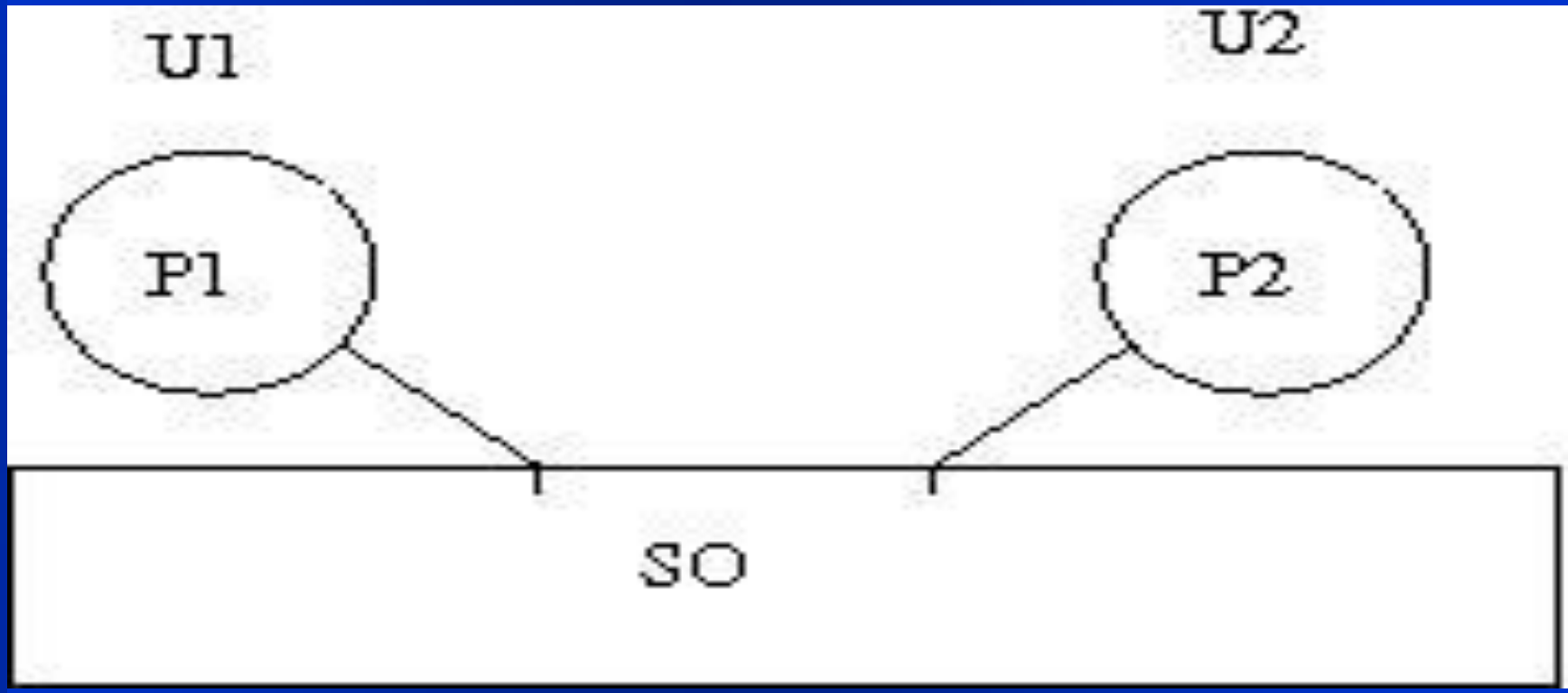
Thompson

Mini-computador
DEC
PDP-11
com vários
periféricos

http://www.bell-labs.com/history/unix/

# Multiprogramação - vantagens

- **Suportar a execução concorrente de múltiplos programas independentes**

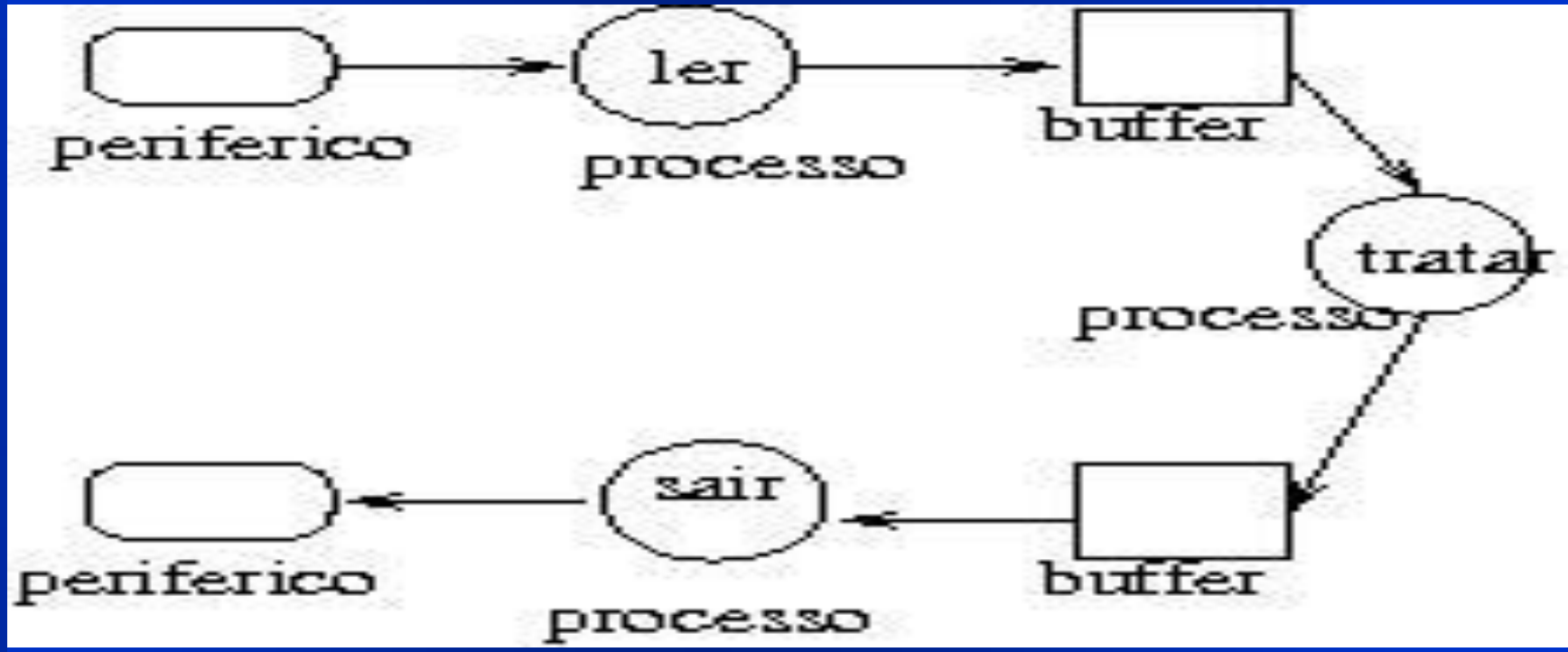


# Multiprogramação - vantagens

- **Tornar mais rentável a utilização do CPU**

# Multiprogramação - vantagens

- **Permitir a cooperação entre processos concorrentes de uma mesma aplicação**



# Objectivos genéricos do SO

- **Melhorar a utilização dos recursos disponíveis, gerindo os recursos *hardware*:**

# Objectivos genéricos do SO

- **Melhorar a utilização dos recursos disponíveis, gerindo o *hardware*:**
  - **Escalonar os vários processos e suportar a multiprogramação**

# Objectivos genéricos do SO

- **Melhorar a utilização dos recursos disponíveis, gerindo os recursos *hardware*:**
  - **Escalonar os vários processos e suportar a multiprogramação**
  - **Suportar memória virtual**

# Objectivos genéricos do SO

- **Melhorar a utilização dos recursos disponíveis, gerindo os recursos *hardware*:**
  - **Escalonar os vários processos e suportar a multiprogramação**
  - **Suportar memória virtual**
  - **Suportar canais virtuais de I/O**

# Objectivos genéricos do SO

- **Melhorar a utilização dos recursos disponíveis, gerindo os recursos *hardware*:**
  - **Escalonar os vários processos e suportar a multiprogramação**
  - **Suportar memória virtual**
  - **Suportar canais virtuais de I/O**
  - **Suportar uma máquina virtual protegida**

# Objectivos genéricos do SO

- **Melhorar a utilização dos recursos disponíveis, gerindo os recursos *hardware*:**
  - **Escalonar os vários processos e suportar a multiprogramação**
  - **Suportar memória virtual**
  - **Suportar canais virtuais de I/O**
  - **Suportar uma máquina virtual protegida**
- **Oferecer uma interface de programação a mais alto nível – Chamadas ao Sistema**
  - **conceitos de Processo, Ficheiro e Canais de Comunicação**

# Abstracções do SO (máquina virtual)

- **Gerir a execução dos programas e os recursos da máquina**

# Abstracções do SO (máquina virtual)

- **Gerir a execução dos programas e os recursos da máquina**
  - **Iniciar/controlar/terminar os processos**

# Abstracções do SO (máquina virtual)

- **Gerir a execução dos programas e os recursos da máquina**
  - **Iniciar/controlar/terminar os processos**
  - **Atribuir espaço de memória a cada processo**

# Abstracções do SO (máquina virtual)

- **Gerir a execução dos programas e os recursos da máquina**
  - **Iniciar/controlar/terminar os processos**
  - **Atribuir espaço de memória a cada processo**
  - **Gerir os ficheiros em disco**

# Abstracções do SO (máquina virtual)

- **Gerir a execução dos programas e os recursos da máquina**
  - **Iniciar/controlar/terminar os processos**
  - **Atribuir espaço de memória a cada processo**
  - **Gerir os ficheiros em disco**
  - **Controlar as acções de I/O sobre os periféricos**

# Abstracções do SO (máquina virtual)

- **Gerir a execução dos programas e os recursos da máquina**
  - **Iniciar/controlar/terminar os processos**
  - **Atribuir espaço de memória a cada processo**
  - **Gerir os ficheiros em disco**
  - **Controlar as acções de I/O sobre os periféricos**
  - ***Time-sharing* do CPU, escondido aos processos**

# Abstracções do SO (máquina virtual)

- **Gerir a execução dos programas e os recursos da máquina**
  - **Iniciar/controlar/terminar os processos**
  - **Atribuir espaço de memória a cada processo**
  - **Gerir os ficheiros em disco**
  - **Controlar as acções de I/O sobre os periféricos**
  - ***Time-sharing* do CPU, escondido aos processos**
- **Gerir as interacções com os utilizadores**
  - **Define que recursos disponibiliza a cada utilizador**

# Resumo *(utilizador vs. computador)*

- **Modo de trabalho do utilizador**
  - **Submissão de trabalhos (lotes batch) não interactivo**
  - **Regime interactivo (aumenta a produtividade)**

# Resumo *(utilizador vs. computador)*

- **Modo de trabalho do utilizador**
  - **Submissão de trabalhos (lotes batch) não interactivo**
  - **Regime interactivo (aumenta a produtividade)**
- **Regimes de execução suportados pelo SO**
  - **Puramente sequencial → monoprogramação**

# Resumo *(utilizador vs. computador)*

- **Modo de trabalho do utilizador**
  - **Submissão de trabalhos (lotes batch) não interactivo**
  - **Regime interactivo (aumenta a produtividade)**
- **Regimes de execução suportados pelo SO**
  - **Puramente sequencial → monoprogramação**
  - **Concorrência: I/O em paralelo com CPU**
    - **Usando lotes ``batch´´ (c/ I/O off-line)**
    - **Usando SPOOL (c/ I/O on-line)**

# Resumo *(utilizador vs. computador)*

- **Modo de trabalho do utilizador**
  - **Submissão de trabalhos (lotes batch) não interactivo**
  - **Regime interactivo (aumenta a produtividade)**
- **Regimes de execução suportados pelo SO**
  - **Puramente sequencial → monoprogramação**
  - **Concorrência: I/O em paralelo com CPU**
    - **Usando lotes ``batch´´ (c/ I/O off-line)**
    - **Usando SPOOL (c/ I/O on-line)**
  - **Concorrência entre múltiplos programas: multiprogramação (aumenta a eficiência do sistema e rentabiliza a utilização da máquina)**

# SPOOL e multiprogramação

- **O conceito de SPOOL continua actual para a partilha de certos recursos**
  - **Exemplo: uma impressora partilhada pelos vários programas só é acessível por meio do SPOOL**

# SPOOL e multiprogramação

- O conceito de SPOOL continua actual para a partilha de certos recursos
  - Exemplo: uma impressora partilhada pelos vários programas só é acessível por meio do SPOOL
    - Um programa que pretenda uma impressão, submete o pedido de um trabalho ao *spooler* da impressora
    - O *spooler* é um processo servidor que vai imprimindo os vários trabalhos, por uma certa ordem, garantido que não há "misturas"

# Computações...

- **Sequenciais: pressupõem uma ordem total**

# Computações...

- **Sequenciais: pressupõem uma ordem total**
  - **Uma computação só se inicia depois da anterior estar completamente concluída**

# Computações...

- **Sequenciais: pressupõem uma ordem total**
  - **Uma computação só se inicia depois da anterior estar completamente concluída**
- **Concorrentes: a ordem não interessa**

# Computações...

- **Sequenciais: pressupõem uma ordem total**
  - **Uma computação só se inicia depois da anterior estar completamente concluída**
- **Concorrentes: a ordem não interessa**
  - **Várias computações independentes "competem" pelo CPU (não há ordem pré-definida, podendo executar-se uma acção de qualquer das computações)**

# Computações...

- **Sequenciais: pressupõem uma ordem total**
  - **Uma computação só se inicia depois da anterior estar completamente concluída**
- **Concorrentes: a ordem não interessa**
  - **Várias computações independentes "competem" pelo CPU (não há ordem pré-definida, podendo executar-se uma acção de qualquer das computações)**
- **Paralelas: há mais do que um processador real**

# Computações...

- **Sequenciais: pressupõem uma ordem total**
  - **Uma computação só se inicia depois da anterior estar completamente concluída**
- **Concorrentes: a ordem de execução não interessa se forem independentes**
  - **Várias computações independentes "competem" pelo CPU (não há ordem pré-definida, podendo executar-se uma acção de qualquer das computações)**
- **Paralelas: há mais do que um processador real e computações concorrentes**
  - **Várias computações executam verdadeiramente em simultâneo (em paralelo), em CPUs distintos**

# Os sistemas de operação suportam

- **Multiprogramação**
  - **Múltiplos programas em execução concorrente**

# Os sistemas de operação suportam

- **Multiprogramação**
  - **Múltiplos programas em execução concorrente**
- **Interactividade**
  - **Permitir a interacção utilizador / programa**

# Os sistemas de operação suportam

- **Multiprogramação**
  - **Múltiplos programas em execução concorrente**
- **Interactividade**
  - **Permitir a interacção utilizador / programa**
- **Escalonamento (*Scheduling*)**
  - **Escolher a ordem de execução dos programas**

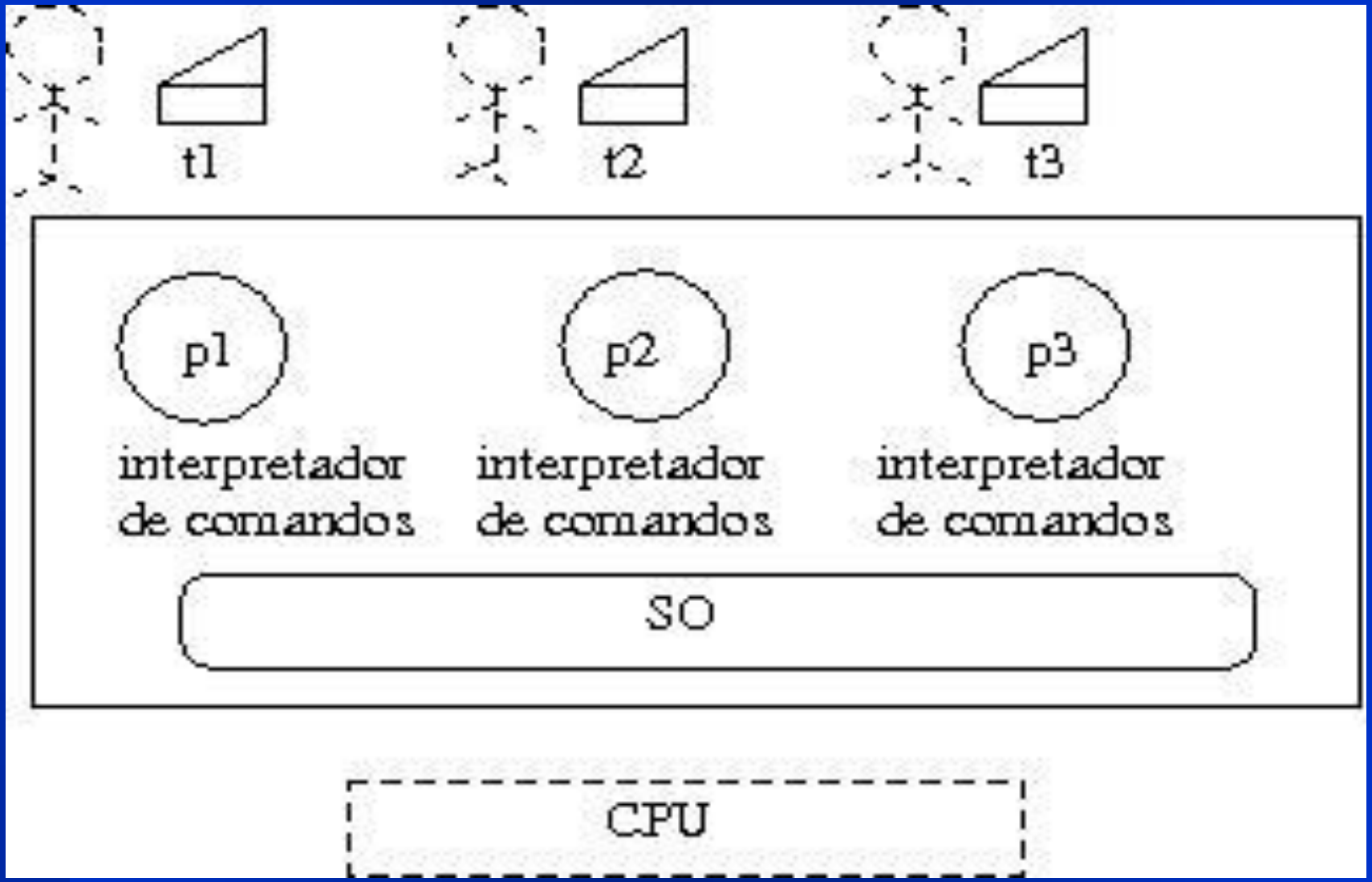
# Os sistemas de operação suportam

- **Multiprogramação**
  - **Múltiplos programas em execução concorrente**
- **Interactividade**
  - **Permitir a interacção utilizador / programa**
- **Escalonamento**
  - **Escolher a ordem de execução dos programas**
- ***Time-sharing de cada CPU***
  - **Concorrência, através da partilha do tempo de utilização de cada CPU**
  - **Um mecanismo de apreensão do CPU (*preemption*) para garantir a sua partilha equilibrada**

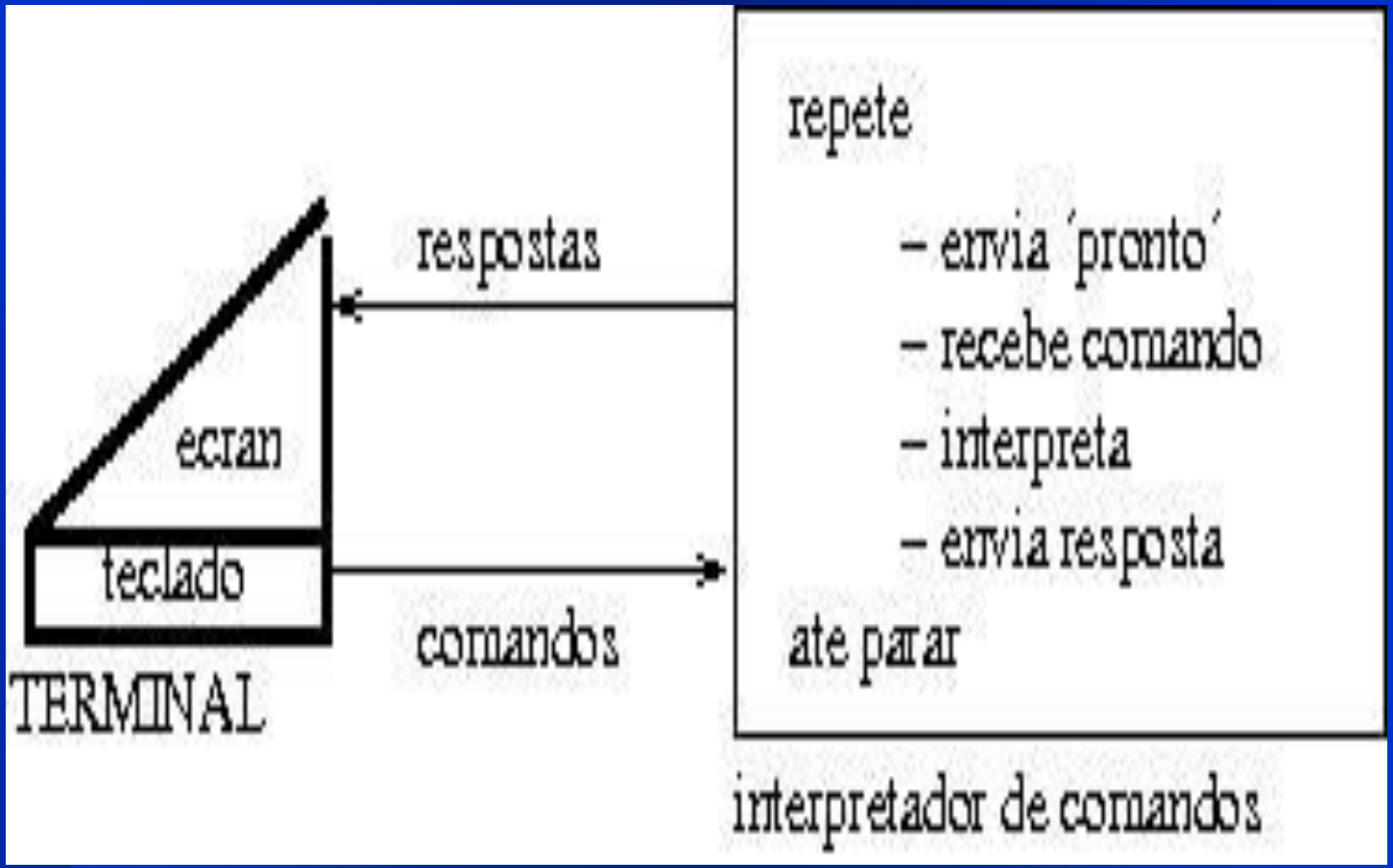
# Mecanismos básicos do SO

- **Processos:** controlo da execução concorrente e da multiprogramação
- **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

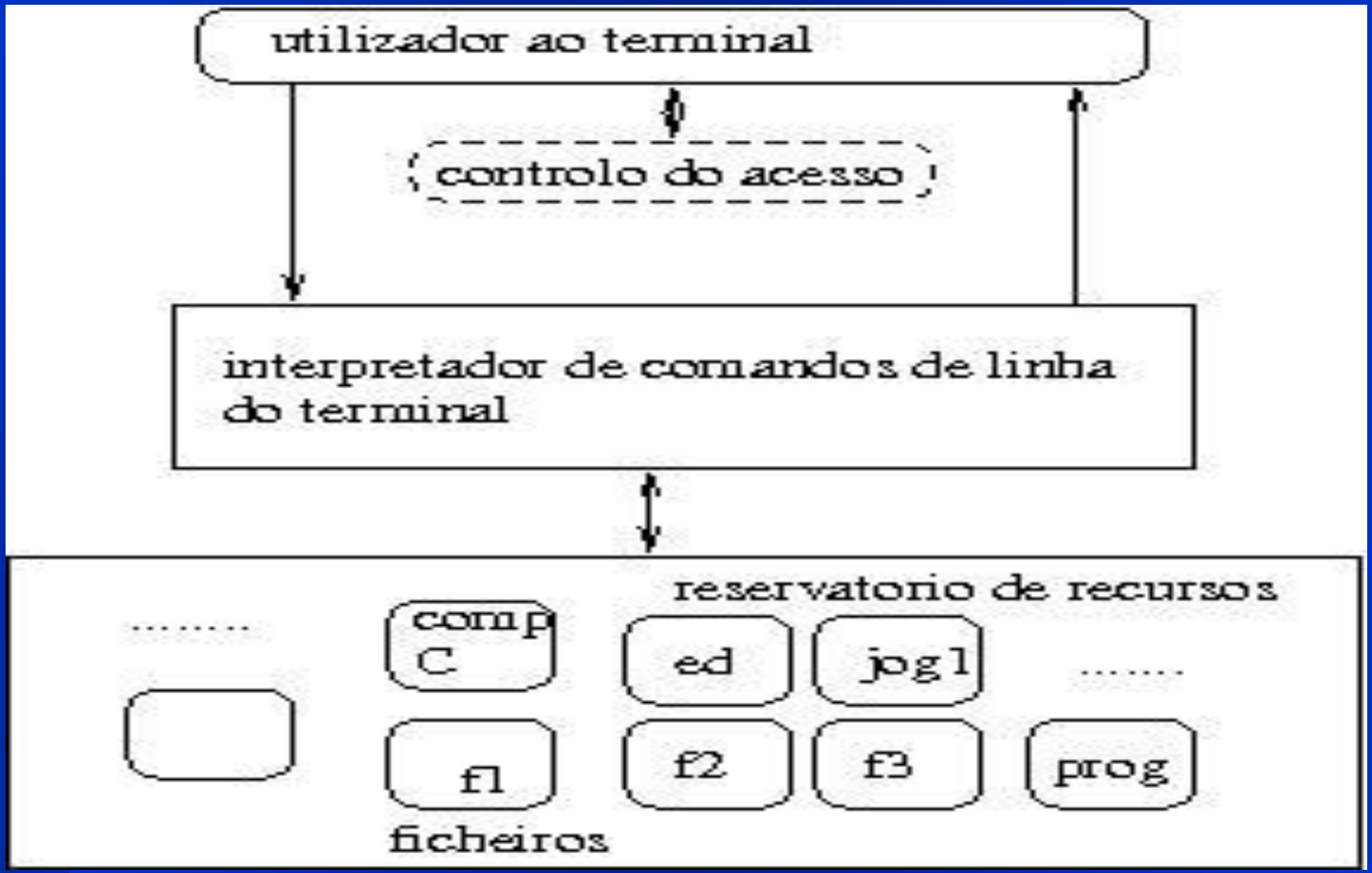
# Sistemas interactivos



# Interpretador de comandos



# Máquina virtual vista p/ utilizador



# Interfaces de acesso ao SO

- **Interpretador de comandos: *shell* ou *menus***
  - **Pedidos através do terminal**

# 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**

# 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´´ (interrupção por *software*)**

# Instruções da máquina virtual

- **Um programa em execução invoca:**
  - **Instruções máquina hardware**

# Instruções da máquina virtual

- **Um programa em execução invoca:**
  - **Instruções máquina hardware**
  - **Chamadas a funções de biblioteca**

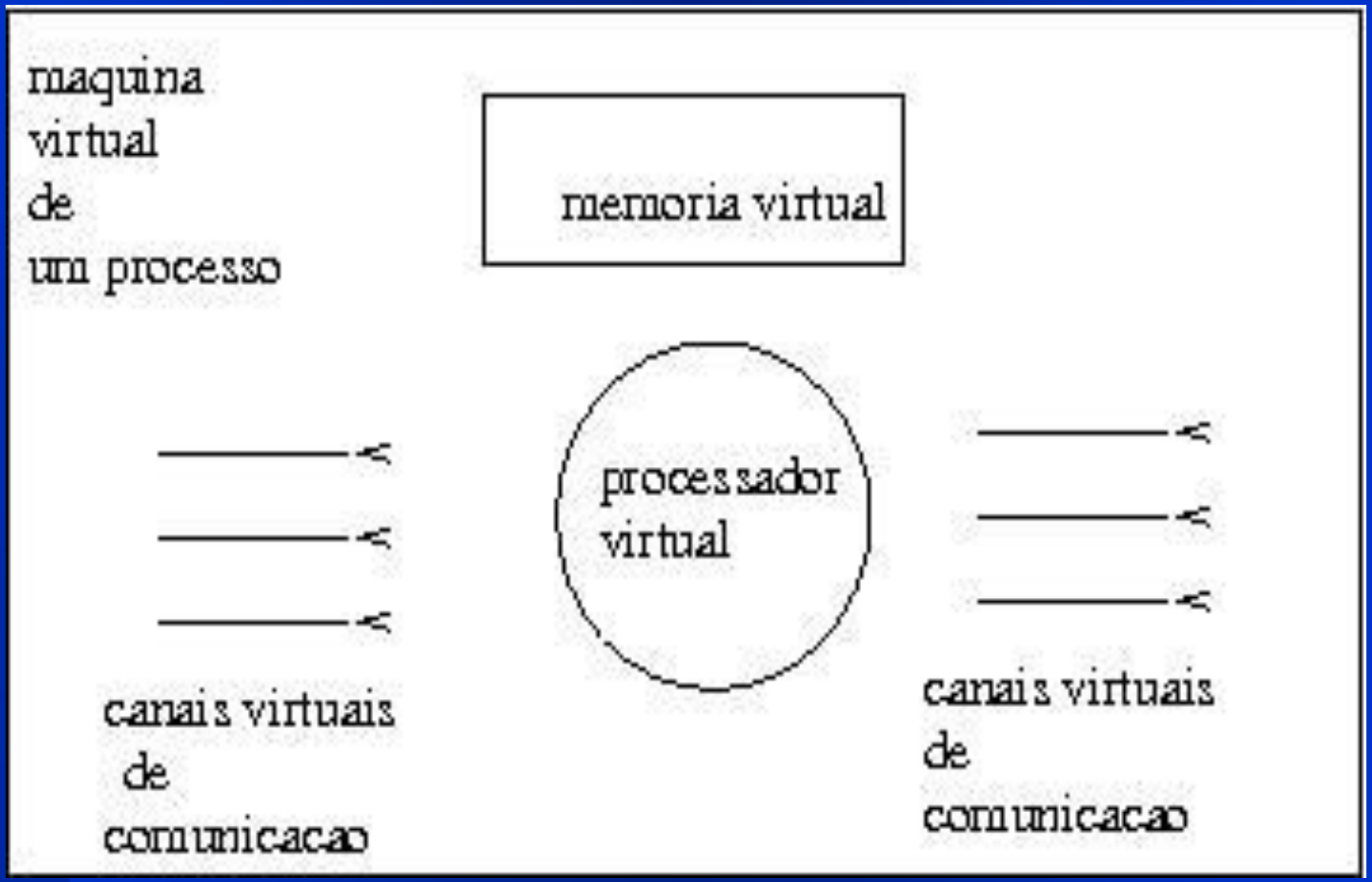
# Instruções da máquina virtual

- **Um programa em execução invoca:**
  - **Instruções máquina hardware**
  - **Chamadas a funções de biblioteca**
  - **Chamadas ao SO**

# Instruções máquina virtual do SO

- **Um programa em execução invoca:**
  - **Instruções 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**

# Máquina virtual do processo



# Processador virtual → real

- **Que processador real (CPU) vai executar as instruções de cada processo?**

# Processador virtual → real

- **Que processador real (CPU) vai executar as instruções de cada processo?**
- **A) Se só 1 CPU → multiprogramação**

# Processador virtual → real

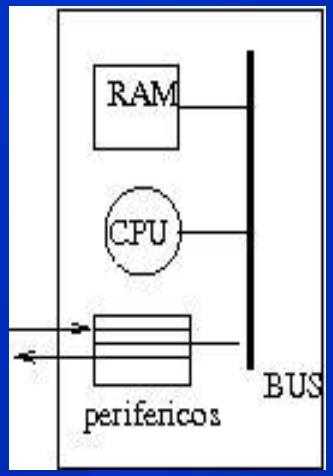
- **Que processador real (CPU) vai executar as instruções de cada processo?**
- **A) Se só 1 CPU → multiprogramação**
- **B) Se múltiplos CPU → execução paralela: escolhem-se os CPU livres para executar os processos que forem sendo pedidos…**

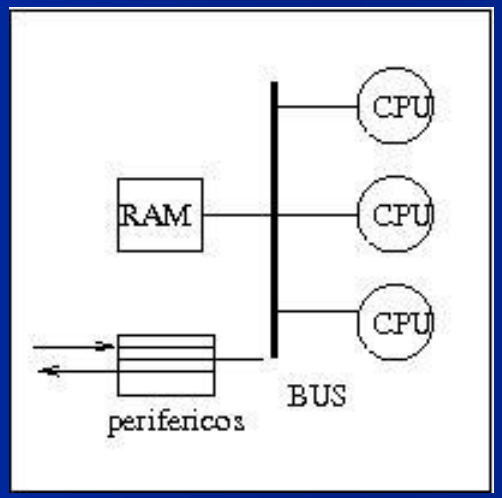
# Processador virtual → real

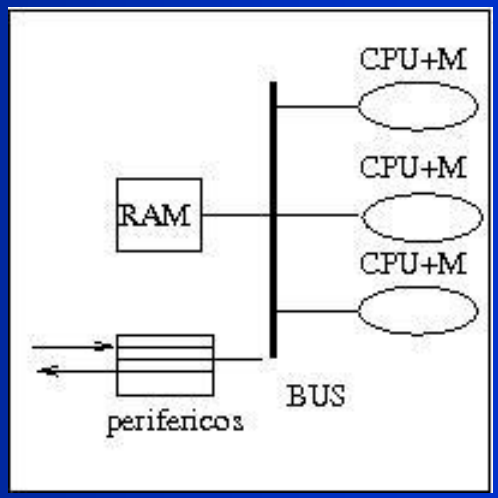
- Que processador real (CPU) vai executar as instruções de cada processo?
- A) Se só 1 CPU → multiprogramação
- B) Se múltiplos CPU → execução paralela: escolhem-se os CPU livres para executar os processos que forem sendo pedidos…

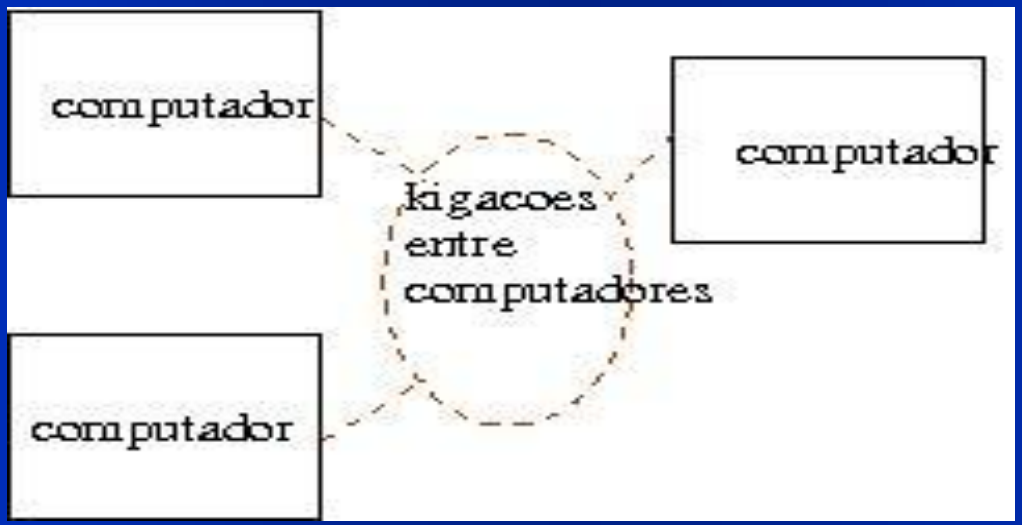
Quando todos os CPU ocupados → passar a multiprogramar cada um deles

# 1- Processador virtual → real

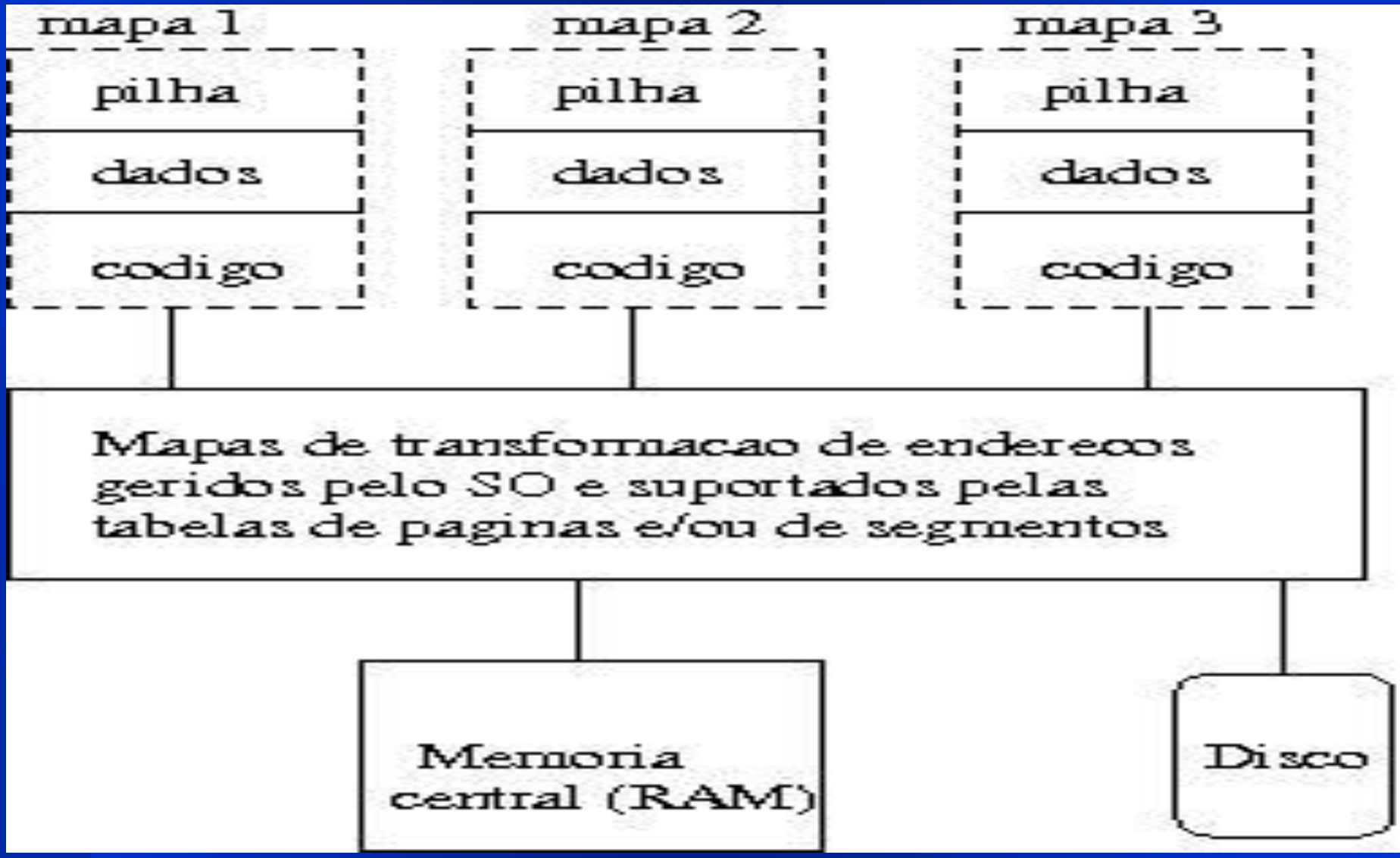


# 2- Memória virtual → real



# Memória virtual num computador

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

**Endereços Virtuais (instruções):**

**64 bits → $2^{64}$ bytes**

**Endereços Reais (*bus*):**

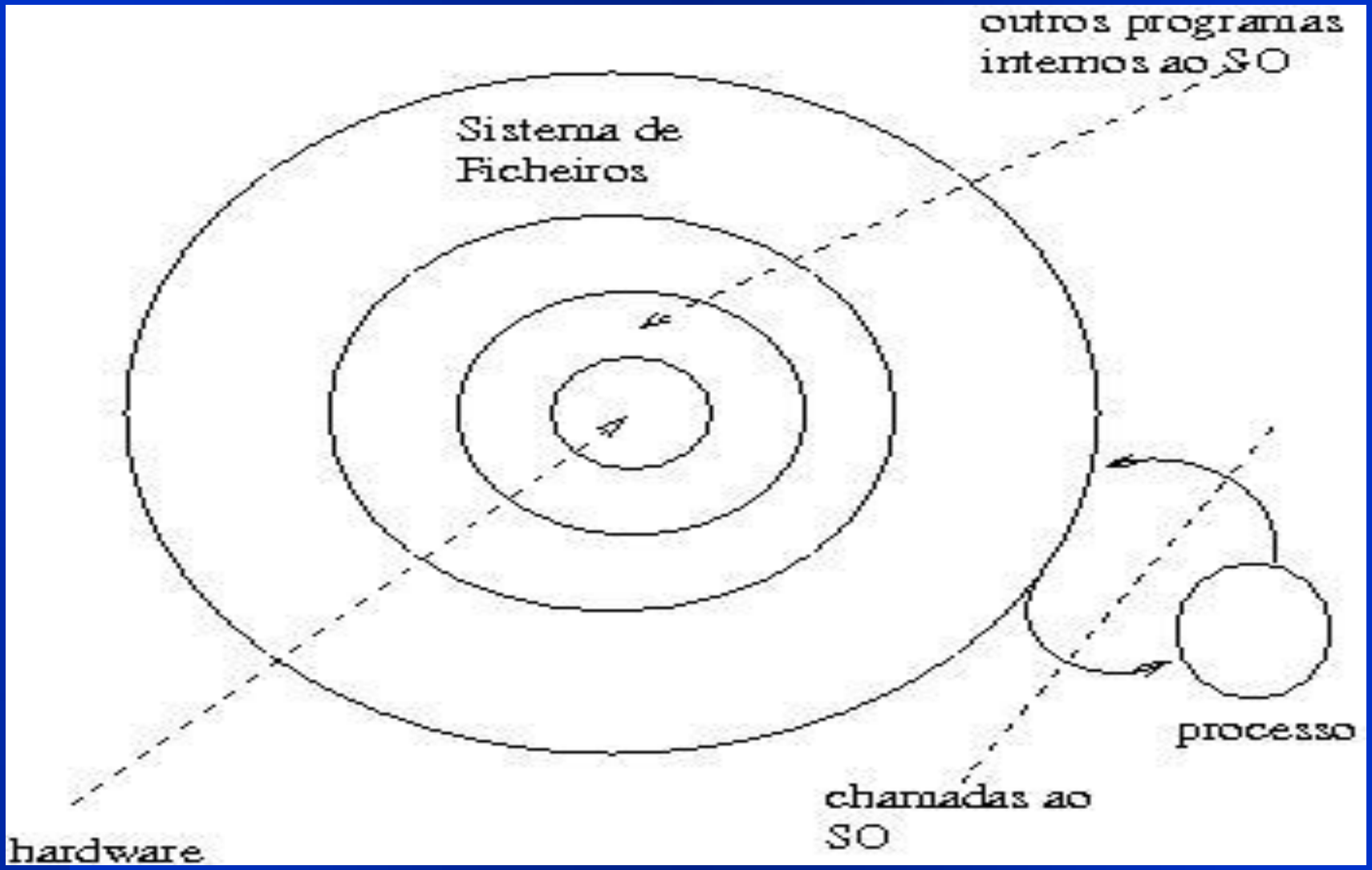
**32 bits → $2^{32}$ bytes**

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

# 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



# 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**

# Conceito de ficheiro

- **Representa:**
  - **Arquivo lógico de informação**
  - **Dispositivo de I/O, externo ao processo**

# Conceito de ficheiro

- **Representa:**
  - **Arquivo lógico de informação**
  - **Dispositivo de I/O, externo ao processo**
- **É caracterizado por:**
  - **Um nome simbólico**

# Conceito de ficheiro

- **Representa:**
  - **Arquivo lógico de informação**
  - **Dispositivo de I/O, externo ao processo**
- **É caracterizado por:**
  - **Um nome simbólico**
  - **Uma determinada organização lógica:**
    - **Uma sequência de *bytes***
    - **Uma sequência de  registos lógicos (*records*)**

# Conceito de ficheiro

- **Representa:**
  - **Arquivo lógico de informação**
  - **Dispositivo de I/O, externo ao processo**
- **É caracterizado por:**
  - **Um nome simbólico**
  - **Uma determinada organização lógica:**
    - **Uma sequência de *bytes***
    - **Uma sequência de registos lógicos (*records*)**
  - **Um conjunto de operações permitidas**

# Conceito de ficheiro

- **Representa:**
  - **Arquivo lógico de informação**
  - **Dispositivo de I/O, externo ao processo**
- **É caracterizado por:**
  - **Um nome simbólico**
  - **Uma determinada organização lógica:**
    - **Uma sequência de *bytes***
    - **Uma sequência de registos lógicos (*records*)**
  - **Um conjunto de operações permitidas**
  - **Um determinado suporte físico**

# Funções do SO rel. a ficheiros

- **Gere as estruturas de dados que representam os ficheiros existentes**

# Funções do SO rel. a ficheiros

- **Gere as estruturas de dados que representam os ficheiros existentes**
- **Decide onde e como os ficheiros são armazenados nos seus suporte físicos**

# Funções do SO rel. a ficheiros

- **Gere as estruturas de dados que representam os ficheiros existentes**
- **Decide onde e como os ficheiros são armazenados nos seus suporte físicos**
- **Decide quem tem acesso a eles**

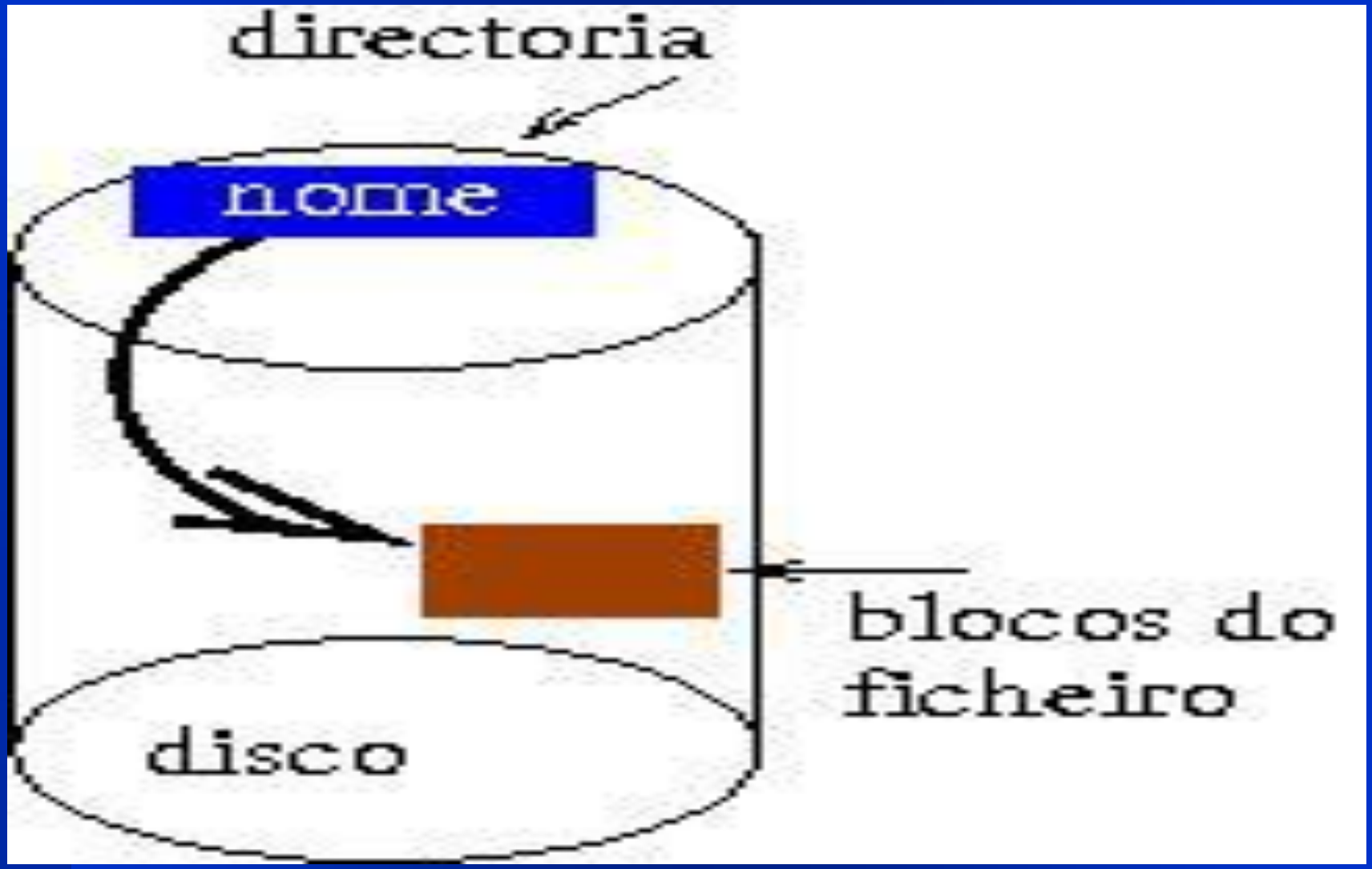
# Funções do SO rel. a ficheiros

- Gere as estruturas de dados que representam os ficheiros existentes
- Decide onde e como os ficheiros são armazenados nos seus suporte físicos
- Decide quem tem acesso a eles
- Localiza e dá acesso aos ficheiros, quando os programas fazem chamadas ao SO

# Funções do SO rel. a ficheiros

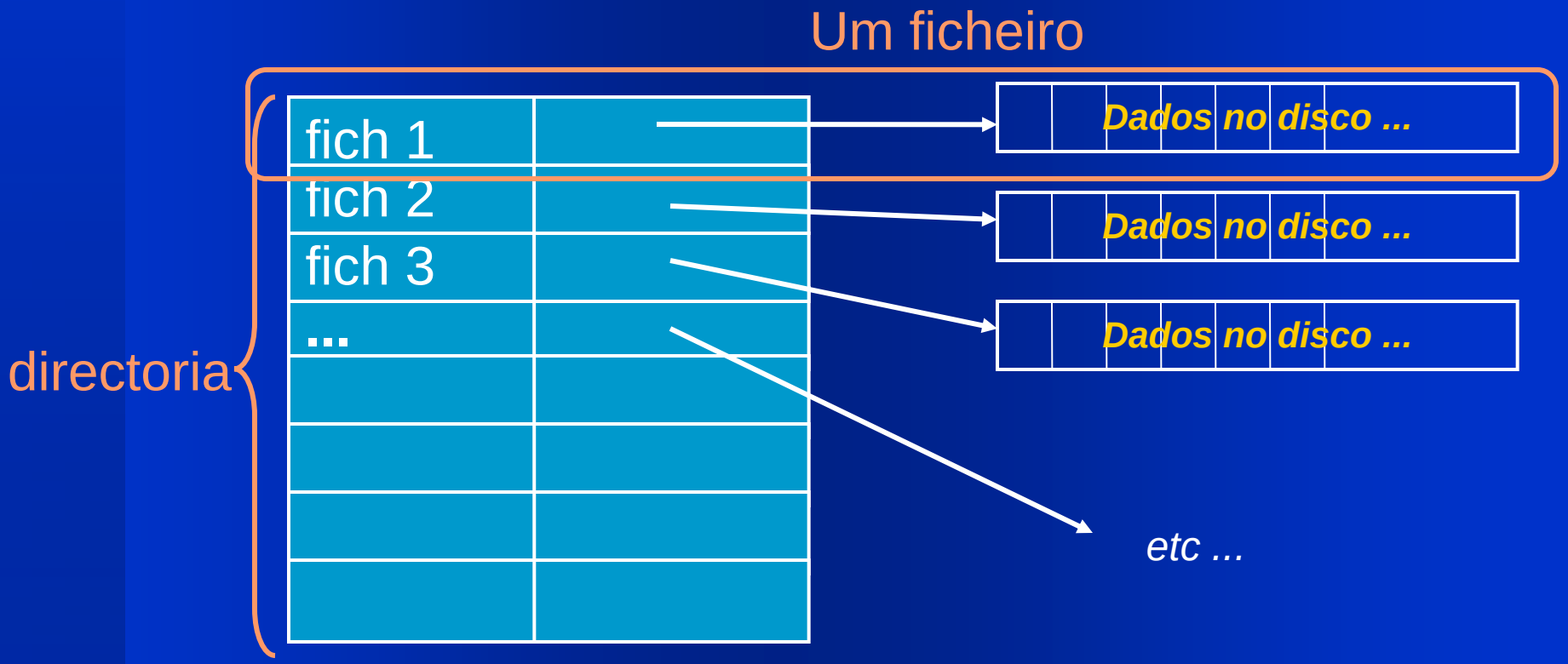
- Gere as estruturas de dados que representam os ficheiros existentes
- Decide onde e como os ficheiros são armazenados nos seus suporte físicos
- Decide quem tem acesso a eles
- Localiza e dá acesso aos ficheiros, quando os programas fazem chamadas ao SO
- Reserva e liberta o espaço ocupado pelos ficheiros

- Como gerir o espaço em disco?
  - Que blocos ocupados/livres?
  - Onde está a informação pretendida?
- Como nomear, localizar e aceder aos ficheiros?
  - Através das directorias


directoria
nome
blocos do ficheiro
disco

# Uma directoria de ficheiros

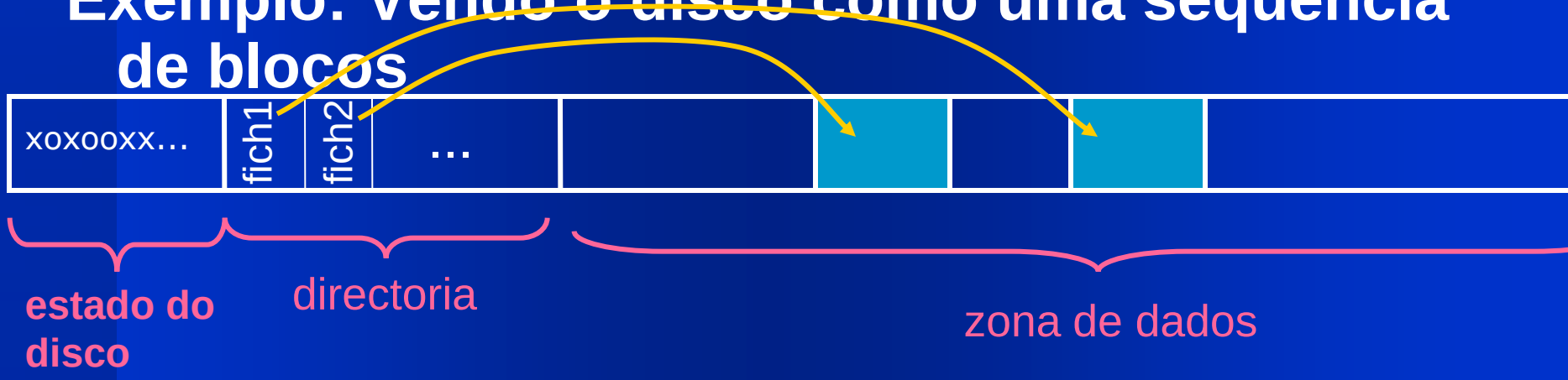
- **Exemplo simples:**



# Um sistema de ficheiros em disco

- As estruturas de dados que descrevem o sistema de ficheiros também têm de ser guardadas no disco
- Só o SO manipula estas estruturas!

Exemplo: Vendo o disco como uma sequência de blocos



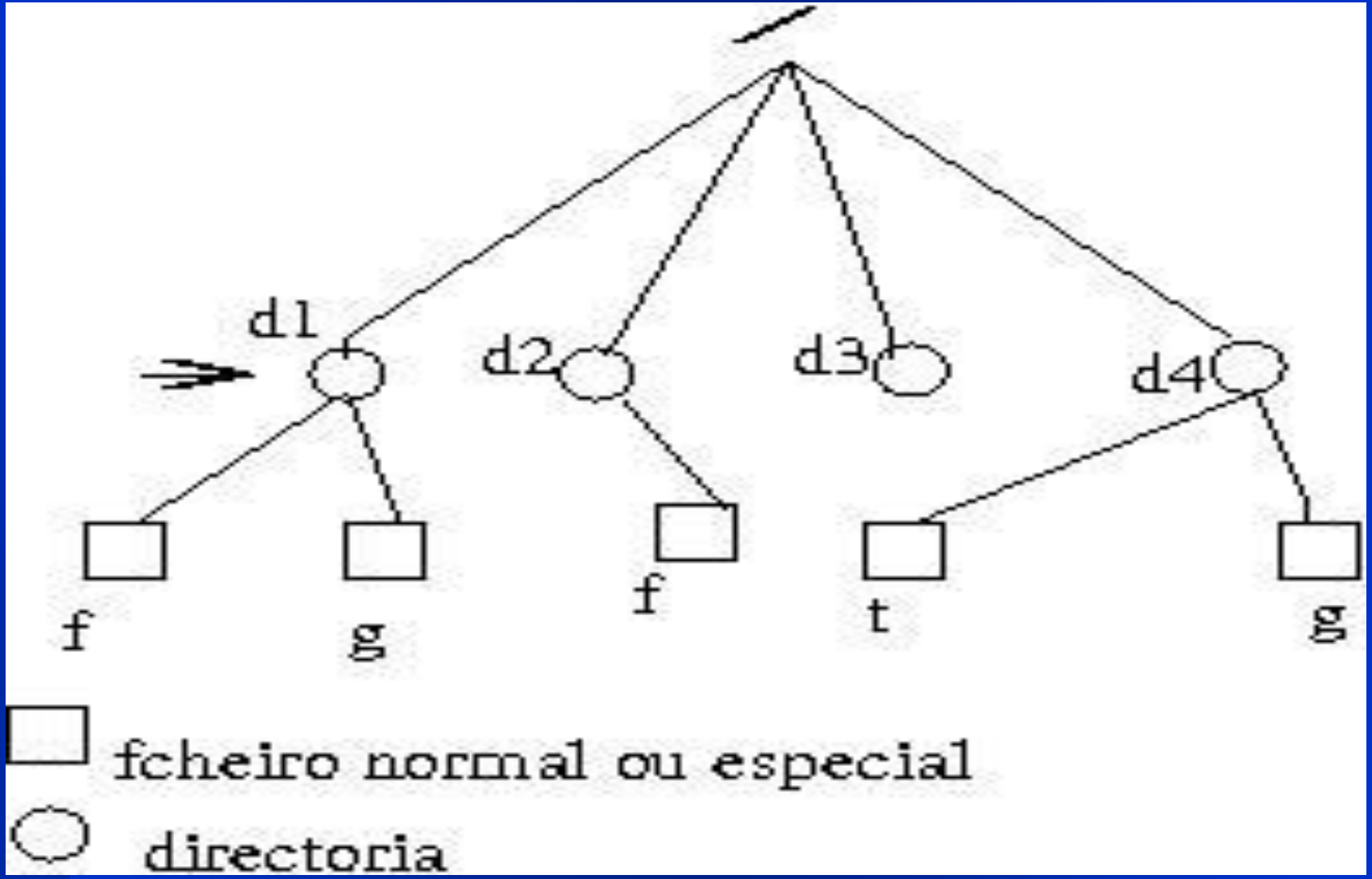
# No Unix: SF simbólico e SF físico

- **SF simbólico:**
  - **Organizado em directorias com os nomes simbólicos dos ficheiros**
  - **Permite operações sobre cada nome simbólico de um ficheiro, independente da representação física**

# No Unix: SF simbólico e SF físico

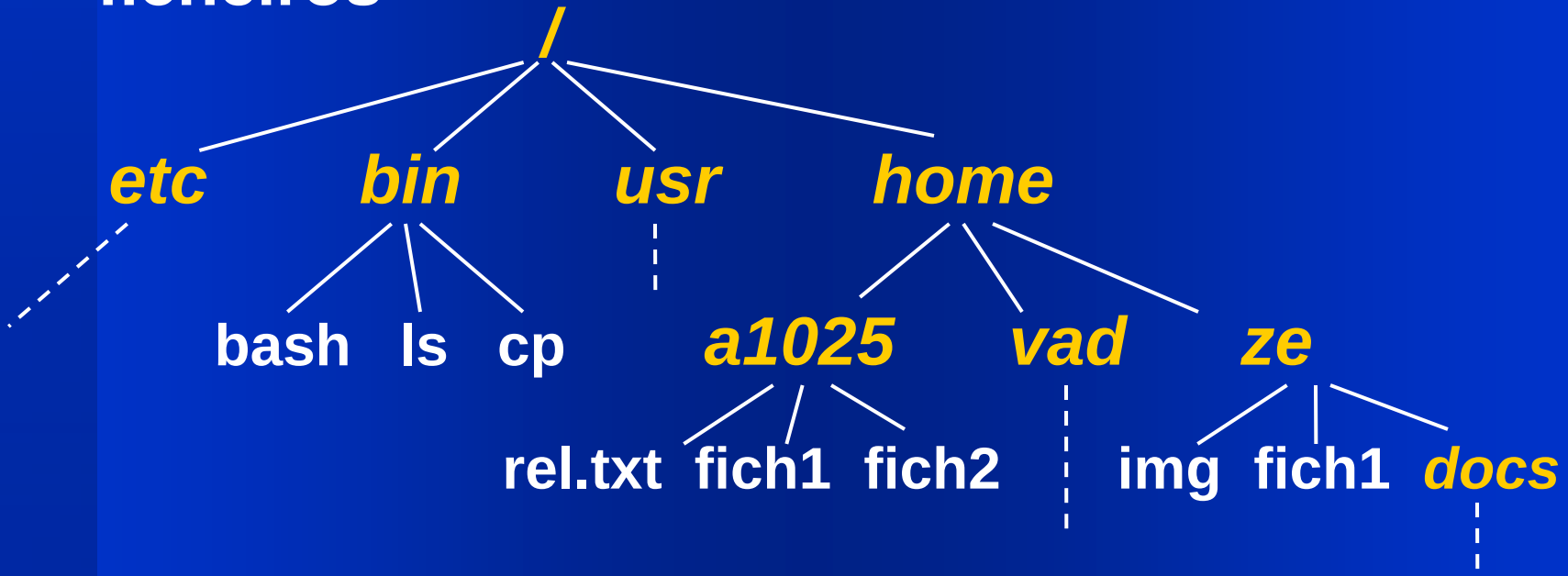
- **SF simbólico:**
  - **Organizado em directorias com os nomes simbólicos dos ficheiros**
  - **Permite operações sobre cada nome simbólico de um ficheiro independente da representação física**
- **SF físico:**
  - **Cada ficheiro tem um Descritor (*i-node* no Unix) que contém os seus atributos e dá acesso à sua representação física**

# SF simbólico



# Directorias e ficheiros

- Espaço de nomes hierárquico
- Cada nome pode ser um ficheiro ou uma directoria
- Cada directoria contém outras directorias e ficheiros



# Nomes de ficheiros e directorias

- **Um nome no sistema de ficheiros inclui o caminho desde a raiz (nome absoluto):**

  `/home/a1025/fich1`

  `/home/ze/fich1`

- **Cada processo tem a sua directoria corrente de trabalho**
  - **Pode-se abreviar o nome:**

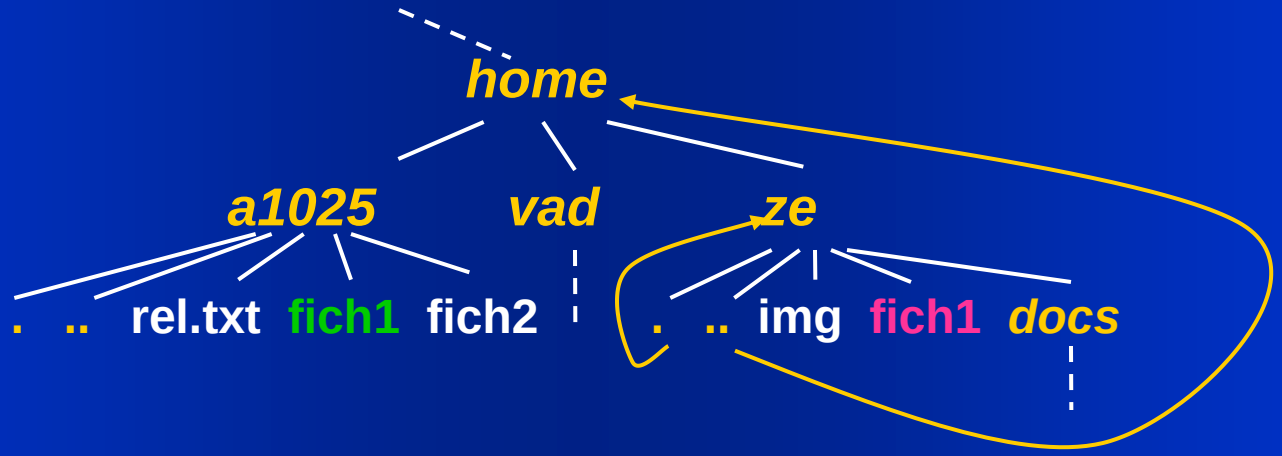
  **No contexto de** `/home/ze`

  **O ficheiro** `´´fich1´´` **indica**

  `/home/ze/fich1`

# Nomes de ficheiros e directorias (2)

- **Cada directoria tem sempre dois elementos**
  - **´´.´´ (a própria)** **´´..´´ (a predecessora)**
- **Podemos usar nomes relativos:**
  - **se a dir. corrente é /home/ze**
    - **fich1 = ./fich1 = /home/ze/fich1**
    - **/home/a1025/fich1 = ../a1025/fich1**



# SF simbólico

- **Directorias: são ficheiros cujo conteúdo é interpretado como uma lista de pares:**

  **(Nome simbólico, Número interno (i-node))**

# SF simbólico

- **Directorias: são ficheiros cujo conteúdo é interpretado como uma lista de pares:**

  **(Nome simbólico, Número interno (i-node))**

- **O i-node é um índice numa Tabela interna do SO que contém os descritores dos ficheiros:**

# SF simbólico

- **Directorias: são ficheiros cujo conteúdo é interpretado como uma lista de pares:**

  **(Nome simbólico, Número interno (i-node))**

- **O i-node é um índice numa Tabela interna do SO que contém os descritores dos ficheiros:**
  - **O ficheiro é normal, é directoria ou é especial**

# SF simbólico

- **Directorias: são ficheiros cujo conteúdo é interpretado como uma lista de pares:**

  **(Nome simbólico, Número interno (i-node))**

- **O i-node é um índice numa Tabela interna do SO que contém os descritores dos ficheiros:**
  - **O ficheiro é normal, é directoria ou é especial**
  - **Nome do dono**

# SF simbólico

- **Directorias: são ficheiros cujo conteúdo é interpretado como uma lista de pares:**

  **(Nome simbólico, Número interno (i-node))**

- **O i-node é um índice numa Tabela interna do SO que contém os descritores dos ficheiros:**
  - **O ficheiro é normal, é directoria ou é especial**
  - **Nome do dono**
  - **Direitos de acesso (R, W, E)**

# SF simbólico

- **Directorias: são ficheiros cujo conteúdo é interpretado como uma lista de pares:**

  **(Nome simbólico, Número interno (i-node))**

- **O i-node é um índice numa Tabela interna do SO que contém os descritores dos ficheiros:**
  - **O ficheiro é normal, é directoria ou é especial**
  - **Nome do dono**
  - **Direitos de acesso (R, W, E)**
  - **Datas de criação, acesso, modificação de atributos**

# SF simbólico

- **Directorias: são ficheiros cujo conteúdo é interpretado como uma lista de pares:**

  **(Nome simbólico, Número interno (i-node))**

- **O i-node é um índice numa Tabela interna do SO que contém os descritores dos ficheiros:**
  - **O ficheiro é normal, é directoria ou é especial**
  - **Nome do dono**
  - **Direitos de acesso (R, W, E)**
  - **Datas de criação, acesso, modificação de atributos**
  - **Localização física (em disco ou noutro periférico)**

# SF simbólico

- **Directorias: são ficheiros cujo conteúdo é interpretado como uma lista de pares:**

  **(Nome simbólico, Número interno (i-node))**

- **O i-node é um índice numa Tabela interna do SO que contém os descritores dos ficheiros:**
  - **O ficheiro é normal, é directoria ou é especial**
  - **Nome do dono**
  - **Direitos de acesso (R, W, E)**
  - **Datas de criação, acesso, modificação de atributos**
  - **Localização física (em disco ou noutro periférico)**
  - **Outras informações**

# Directoria

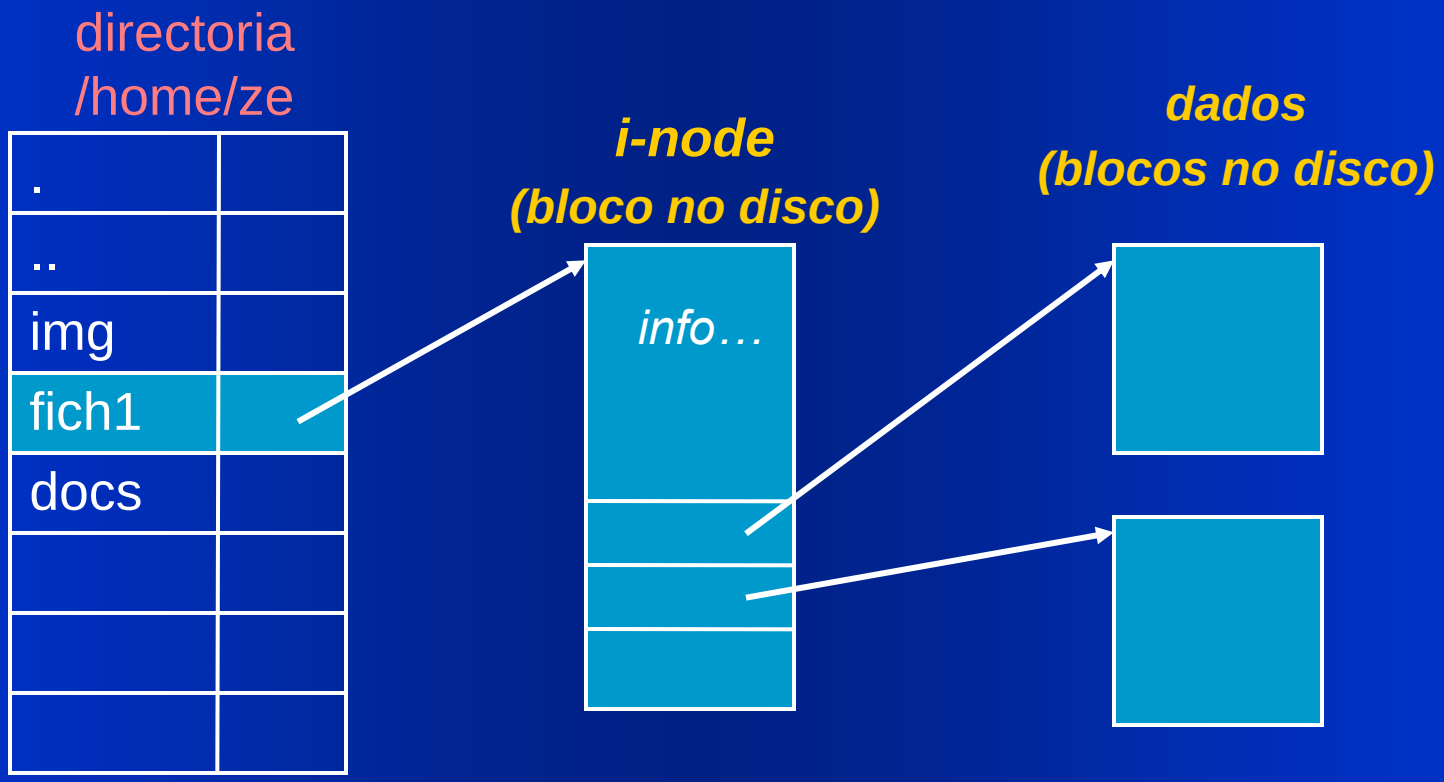
- **Cada ficheiro descrito por uma entrada**

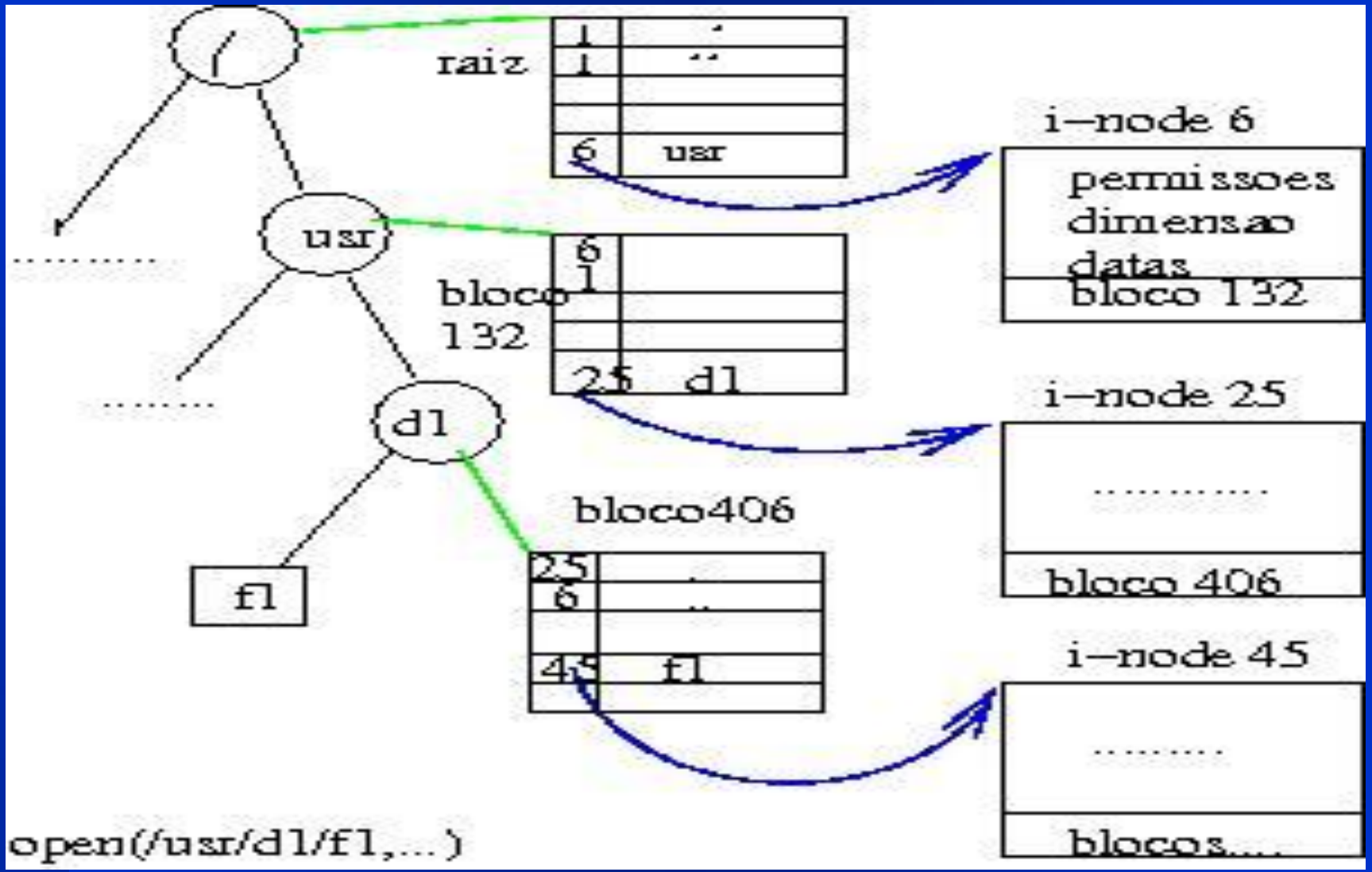
directoria
/home/ze

| | |
|---|---|
| . | |
| .. | |
| img | |
| fich1 | |
| docs | |
| | |
| | |
| | |

# Directoria

- **Cada entrada aponta um i-node**




/
raiz
1 .
1 ..
6 usr
i-node 6
permissoes
dimensao
datas
bloco 132
usr
6
1
bloco
132
25 d1
i-node 25
d1
bloco406
25 .
6 ..
45 f1
bloco 406
f1
i-node 45
blocos....
open(/usr/d1/f1,...)

# Protecção a nível do Sistema de Ficheiros

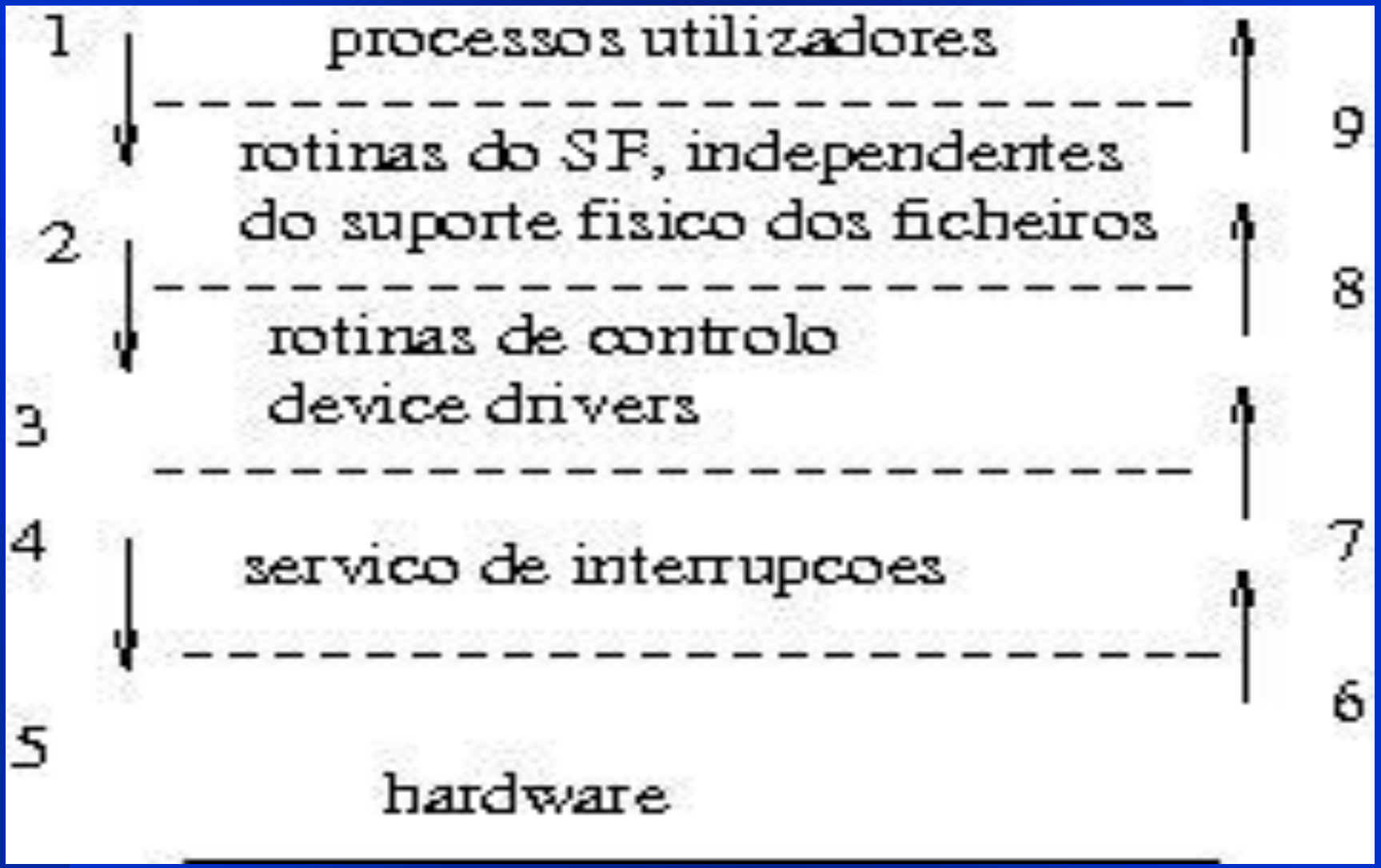
- **Utilizador tem um nome – User ID (UID) e pertence a um grupo de utilizadores (GID)**
- **Define o dono (‘owner’) dos ficheiros que os seus programas criam.**
- **É usado para validar o acesso aos ficheiros.**
- **O ‘superuser (root)’ tem todos os privilégios.**

# Permissões de acesso a ficheiros

- **Classes de utilizadores nos ficheiros:**
  - ***Owner*: cujo UID foi associado ao ficheiro na sua criação ou explicitamente indicado**
  - ***Group*: grupo (GID) a cujo UID pertence ou explicitamente indicado para o ficheiro**
  - ***Other*: nem dono nem do grupo**
- **Modos de acesso:**
  - **ler (Read) – escrever (Write) – executar (eXecute)**
- **Exemplo (várias representações):**

| Owner | Group | Other | |
|---|---|---|---|
| `rwx` | `r-x` | `r--` | (texto: ls -l) |
| `111` | `101` | `100` | (binário) |
| `7` | `5` | `4` | (base octal) |

# I/O -- Hierarquia de camadas



# Passos na execução (1)

- **Um programa invoca uma chamada ao SO**

# Passos na execução (1)

- Um programa invoca uma chamada ao SO
- As rotinas de nível superior do SF:
  - Localizam o ficheiro pelo seu nome simbólico

# Passos na execução (1)

- Um programa invoca uma chamada ao SO
- As rotinas de nível superior do SF:
  - Localizam o ficheiro pelo seu nome simbólico
  - Acedem às estruturas em disco e trazem-nas para memória

# Passos na execução (1)

- Um programa invoca uma chamada ao SO
- As rotinas de nível superior do SF:
  - Localizam o ficheiro pelo seu nome simbólico
  - Acedem às estruturas em disco e trazem-nas para memória
  - Verificam os direitos de acesso

# Passos na execução (1)

- **Um programa invoca uma chamada ao SO**
- **As rotinas de nível superior do SF:**
  - **Localizam o ficheiro pelo seu nome simbólico**
  - **Acedem às estruturas em disco e trazem-nas para memória**
  - **Verificam os direitos de acesso**
  - **Se os *buffers* não estão vazios: retornam os dados**

# Passos na execução (1)

- Um programa invoca uma chamada ao SO
- As rotinas de nível superior do SF:
  - Localizam o ficheiro pelo seu nome simbólico
  - Acedem às estruturas em disco e trazem-nas para memória
  - Verificam os direitos de acesso
  - Se os *buffers* não estão vazios: retornam os dados
  - Senão: invocam as rotinas dos *Device Drivers*, pedindo a operação física de I/O → aguardar...

# Passos na execução (2)

- **Completada a operação física de I/O:**

# Passos na execução (2)

- **Completada a operação física de I/O:**
  - **O periférico gera um pedido de interrupção**

# Passos na execução (2)

- **Completada a operação física de I/O:**
  - **O periférico gera um pedido de interrupção**
  - **A RotinaServiçoInterrupção é invocada (por *hardware*):**
    - **acede às portas da interface de I/O**
    - **Obtém os dados, que passa para o nível superior**
    - **Retorna**

# Passos na execução (2)

- **Completada a operação física de I/O:**
  - **O periférico gera um pedido de interrupção**
  - **A RotinaServiçoInterrupção é invocada (por *hardware*):**
    - **acede às portas da interface de I/O**
    - **Obtém os dados, que passa para o nível superior**
    - **Retorna**
- **Retorna-se às rotinas de nível superior, que preparam o retorno ao programa utilizador**

# Os conceitos de ficheiros no Unix

- **Ficheiro:**

# Os conceitos de ficheiros no Unix

- **Ficheiro:**
  - **uma sequência de *bytes*, sem qualquer estrutura pré-definida**

# Os conceitos de ficheiros no Unix

- **Ficheiro:**
  - **uma sequência de *bytes*, sem qualquer estrutura pré-definida**
  - ***Bytes* desde 0 até ao N-1 (se o ficheiro N *bytes*)**

# Os conceitos de ficheiros no Unix

- **Ficheiro:**
  - **uma sequência de bytes, sem qualquer estrutura pré-definida**
  - ***Bytes* desde 0 até ao N-1 (se o ficheiro N bytes)**
  - **Sem qualquer interpretação de caracteres especiais de controlo (*end_of_file*, etc)**

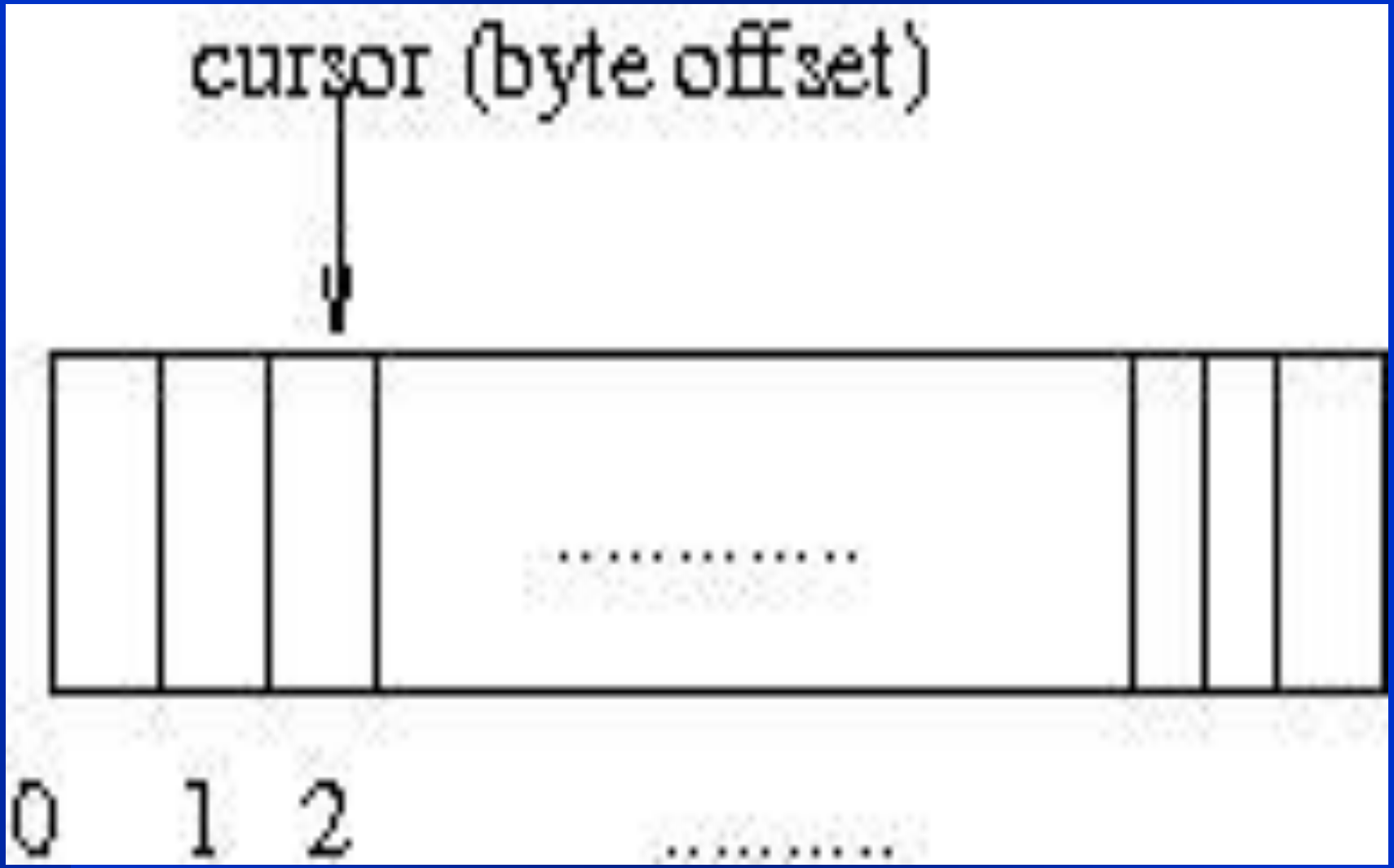
# Os conceitos de ficheiros no Unix

- **Ficheiro:**
  - **uma sequência de bytes, sem qualquer estrutura pré-definida**
  - **Bytes desde 0 até ao N-1 (se o ficheiro N bytes)**
  - **Sem qualquer interpretação de caracteres especiais de controlo (end_of_file, etc)**
  - **Sem qualquer interpretação de tipo de ficheiro especial: isso compete às aplicações…**

# Os conceitos de ficheiros no Unix

- **Ficheiro:**
  - **uma sequência de bytes, sem qualquer estrutura pré-definida**
  - **Bytes desde 0 até ao N-1 (se o ficheiro N bytes)**
  - **Sem qualquer interpretação de caracteres especiais de controlo (end_of_file, etc)**
  - **Sem qualquer interpretação de tipo de ficheiro especial: isso compete às aplicações…**
  - **Acesso sequencial ou acesso directo**

# Ficheiro = *stream* de *bytes*



# Acesso (quase) uniforme

- **ficheiros ´´normais´´ em disco**

# Acesso (quase) uniforme

- **ficheiros ´´normais´´ em disco**
- **directorias: listas de nomes de ficheiros e seus descritores internos**

# Acesso (quase) uniforme

- ficheiros ´´**normais**´´ em disco
- **directorias**: listas de nomes de ficheiros e seus descritores internos
- **dispositivos especiais**:
  - Teclado
  - Écran
  - Impressora
  - Disco físico
  - Memória RAM
  - Dispositivos de comunicação

# Acesso uniforme e canais virtuais

- **Como se consegue?**
  - **Separando as noções:**
    - **de comunicação através de um canal do processo**

    **e**
    - **de dispositivo (ficheiro) externo ao processo**

# Acesso uniforme e canais virtuais

- **Como se consegue?**
  - **Separando as noções:**
    - **de comunicação através de um canal do processo**

    **e**
    - **de dispositivo (ficheiro) externo ao processo**
- **Suportando operações uniformes:**
  - **Para abrir canais de leitura/escrita, ligando-os aos ficheiros e ler/escrever dos canais**

# Acesso uniforme e canais virtuais

- **Como se consegue?**
  - **Separando as noções:**
    - **de comunicação através de um canal do processo**

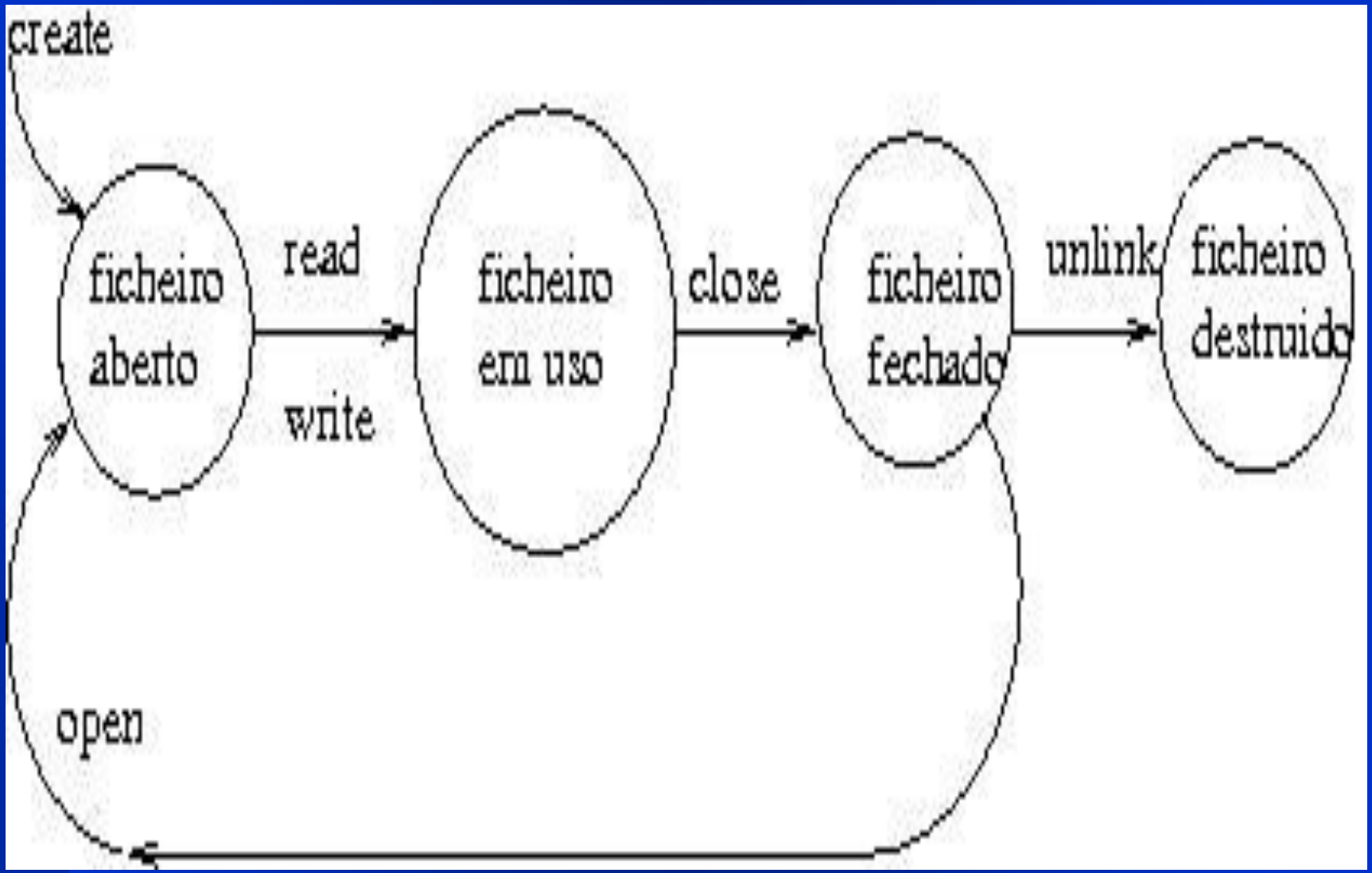
    **e**
    - **de dispositivo (ficheiro) externo ao processo**
- **Suportando operações uniformes:**
  - **Para abrir canais de leitura/escrita, ligando-os aos ficheiros e ler/escrever dos canais**
  - **Para fechar canais, desligando-os dos ficheiros**

# Acesso uniforme e canais virtuais

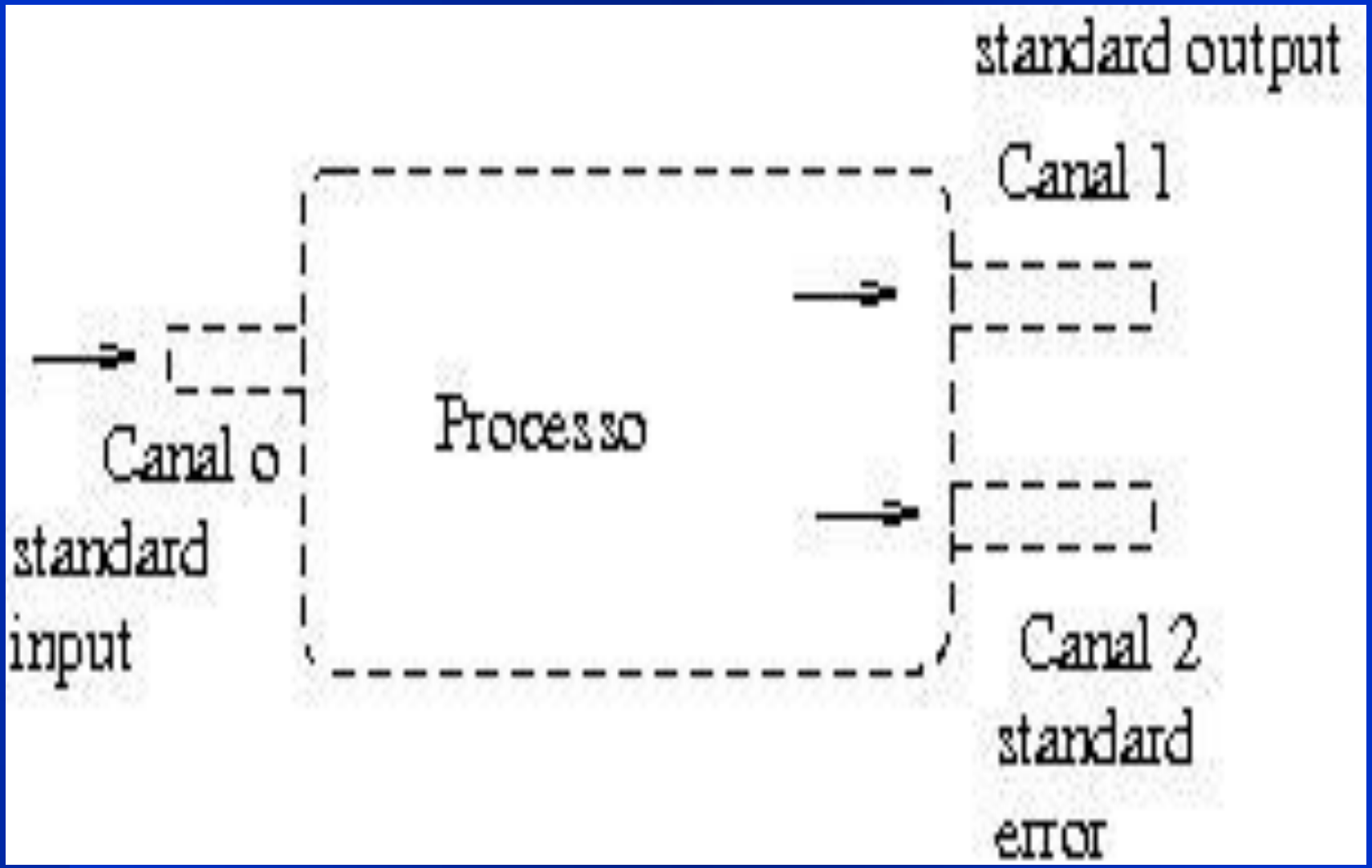
- **Como se consegue?**
  - **Separando as noções:**
    - **de comunicação através de um canal do processo**

    **e**
    - **de dispositivo (ficheiro) externo ao processo**
- **Suportando operações uniformes:**
  - **Para abrir canais de leitura/escrita, ligando-os aos ficheiros e ler/escrever dos canais**
  - **Para fechar canais, desligando-os dos ficheiros**
  - **Para criar e destruir ficheiros e definir os seus atributos**

# Operações sobre ficheiros e canais



# Canais I/O lógicos *standard*



# Canais de I/O *standard*

- **Automaticamente inicializados na criação do processo:**
  - **Habitualmente:**
    - **Canal 0 ligado ao Teclado (*input standard*)**
    - **Canal 1 ligado ao Écran (*output standard*)**
    - **Canal 2 ligado ao Écran (*error standard*)**

# Canais de I/O standard

- **Automaticamente inicializados na criação do processo:**
  - **Habitualmente:**
    - **Canal 0 ligado ao Teclado (input standard)**
    - **Canal 1 ligado ao Écran (output standard)**
    - **Canal 2 ligado ao Écran (error standard)**
- **Podem ser redirigidos para outros ficheiros**

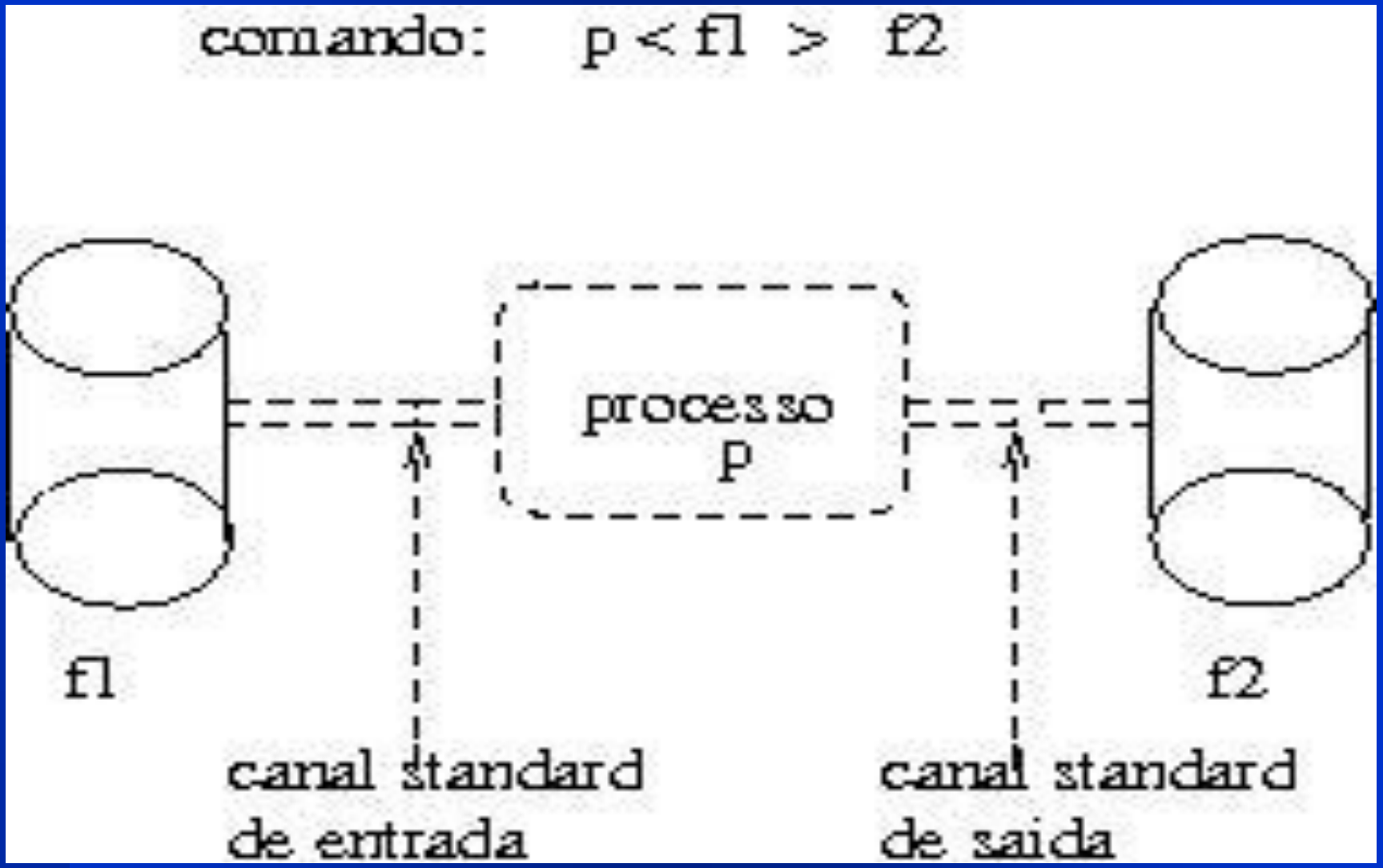
# Canais de I/O standard

- **Automaticamente inicializados na criação do processo:**
  - **Habitualmente:**
    - **Canal 0 ligado ao Teclado (input standard)**
    - **Canal 1 ligado ao Écran (output standard)**
    - **Canal 2 ligado ao Écran (error standard)**
- **Podem ser redirigidos para outros ficheiros**
- **As operações de ler / escrever só referem o número do canal**

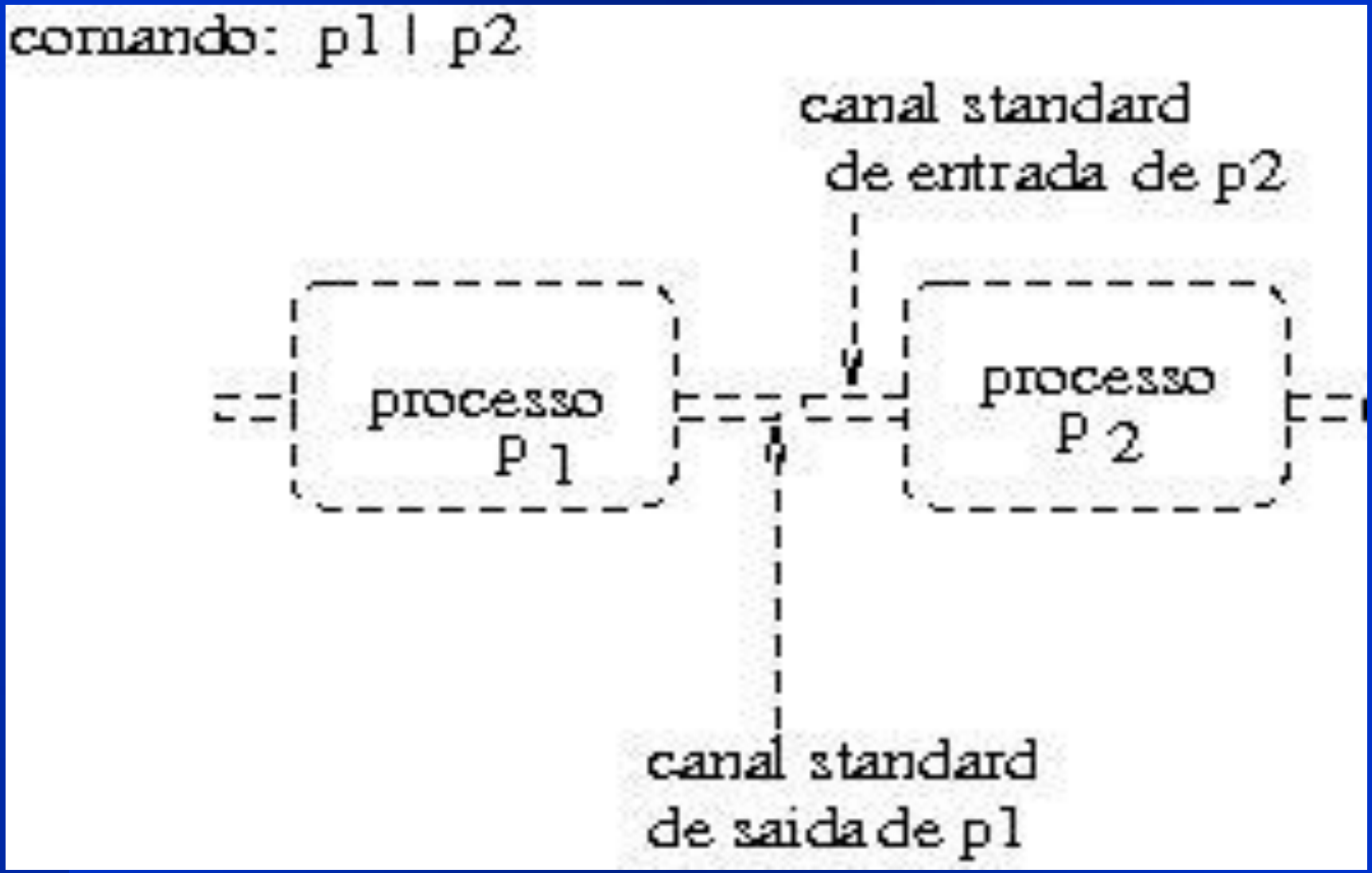
# Canais de I/O standard

- **Automaticamente inicializados na criação do processo:**
  - **Habitualmente:**
    - **Canal 0 ligado ao Teclado (input standard)**
    - **Canal 1 ligado ao Écran (output standard)**
    - **Canal 2 ligado ao Écran (error standard)**
- **Podem ser redirigidos para outros ficheiros**
- **As operações de ler / escrever só referem o número do canal**
- **O código do programa fica independente do nome dos ficheiros ligados aos canais std**

# Redirecção de canais

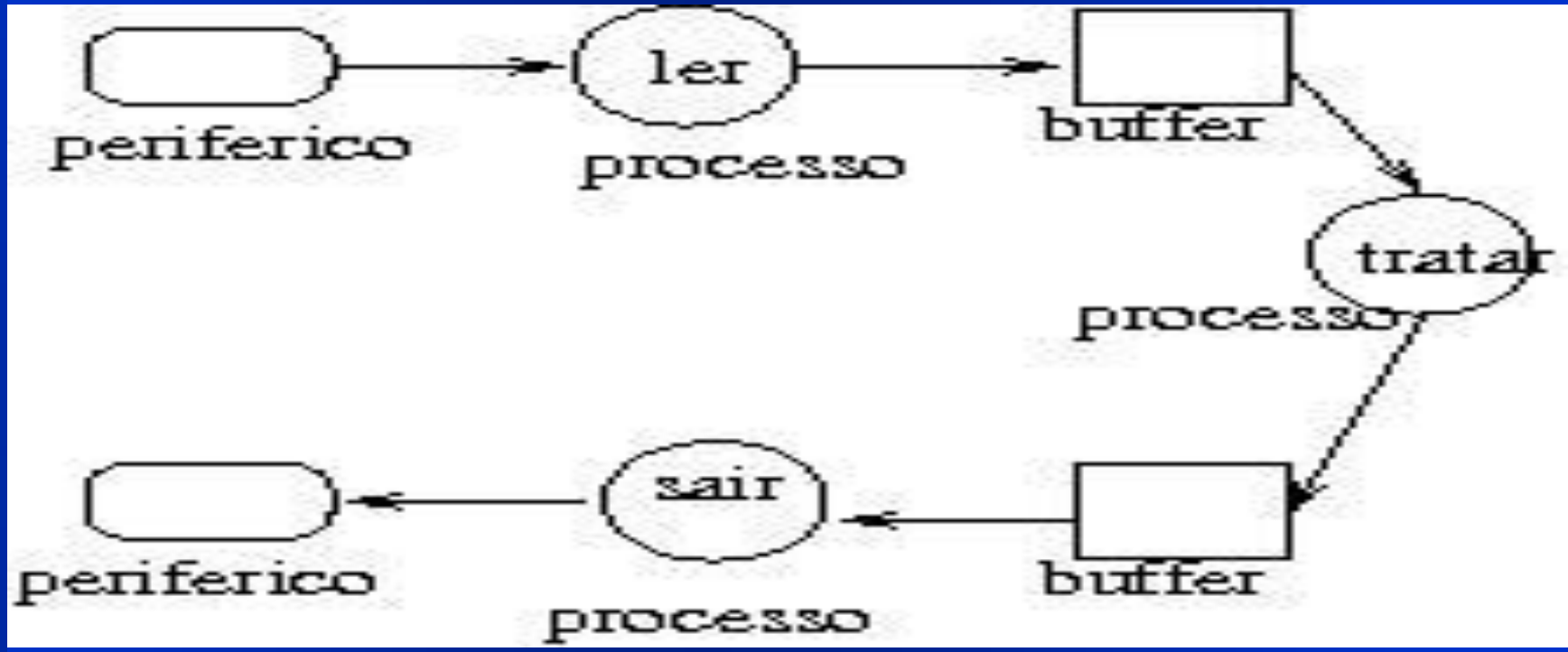


# Pipeline de processos



# Processos concorrentes

- **Permitir a cooperação entre processos concorrentes de uma mesma aplicação**



# Resumo: canais de I/O no Unix

- **O acesso a ficheiros no Unix faz-se estabelecendo canais de ligação entre um processo e um ficheiro**

# Resumo: canais de I/O no Unix

- **O acesso a ficheiros no Unix faz-se estabelecendo canais de ligação entre um processo e um ficheiro**
- **Para cada processo, os canais são identificados por números inteiros, a partir de 0,**

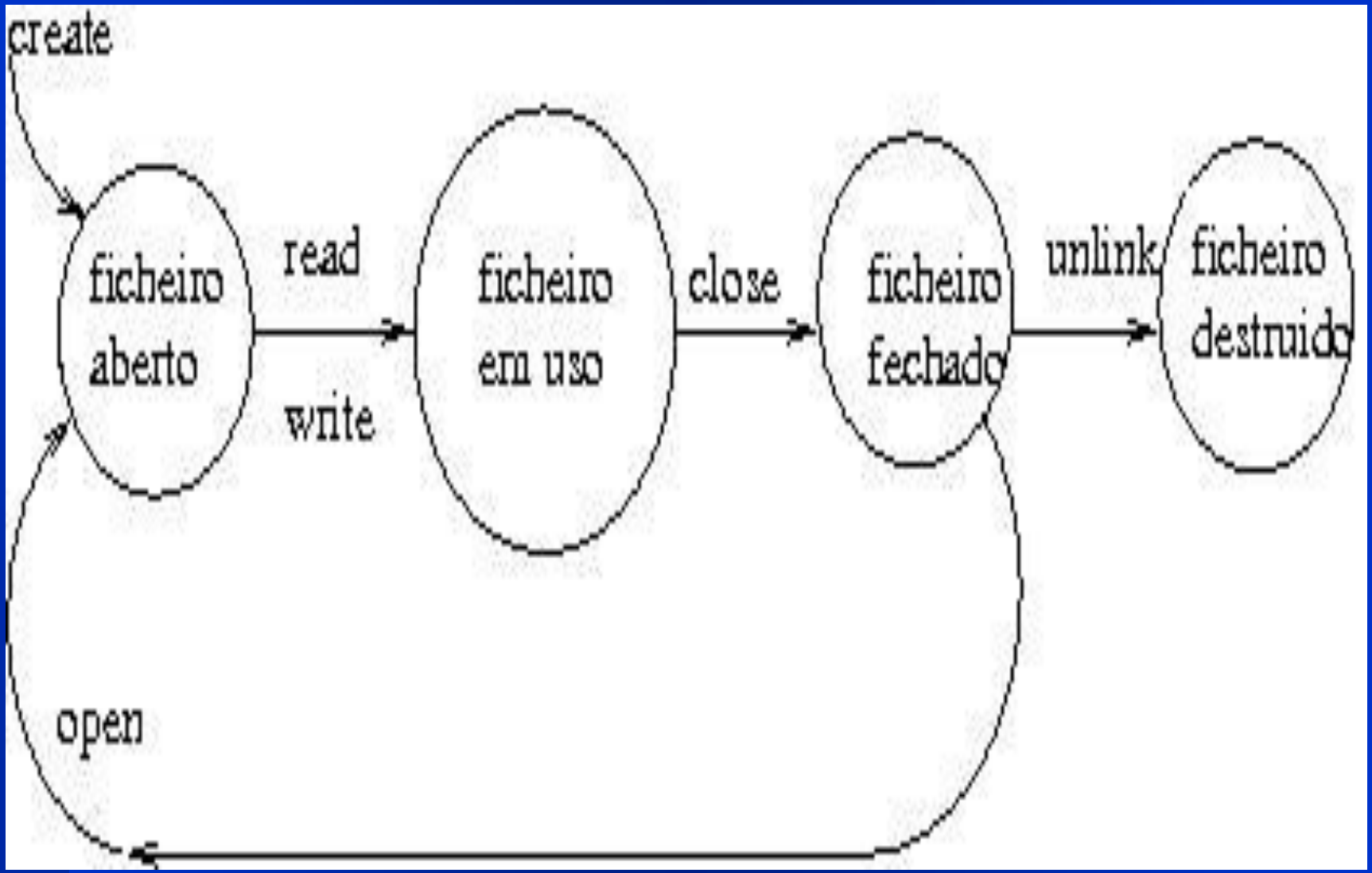
# Resumo: canais de I/O no Unix

- **O acesso a ficheiros no Unix faz-se estabelecendo canais de ligação entre um processo e um ficheiro**
- **Para cada processo, os canais são identificados por números inteiros, a partir de 0,**

  **e podem ser redirigidos dinamicamente durante a execução,**

  **de modo  a dar acesso a outros ficheiros.**

# Operações sobre ficheiros e canais



# Operações do Sistema de Ficheiros

**Sobre nomes simbólicos:**

- **Create: criar ficheiro com nome simbólico**
- **Link: associar um novo nome simbólico a um ficheiro. Pode ter vários nomes.**
- **Unlink: eliminar um nome simbólico associado a um ficheiro**

# Operações do Sistema de Ficheiros

**Sobre nomes simbólicos:**

- **Create: criar ficheiro com nome simbólico**
- **Link: associar um novo nome simbólico a um ficheiro. Pode ter vários nomes.**
- **Unlink: eliminar um nome simbólico associado a um ficheiro**

**Sobre canais:**

- **Open: abrir canal e ligá-lo a um ficheiro**
- **Read / Write: ler / escrever *bytes* através de um canal**
- **Close: fechar a ligação de um canal a um ficheiro**

# Criação de ficheiro: creat

```
int creat(char *filename, int mode)
```

´´filename´´ indica o nome simbólico

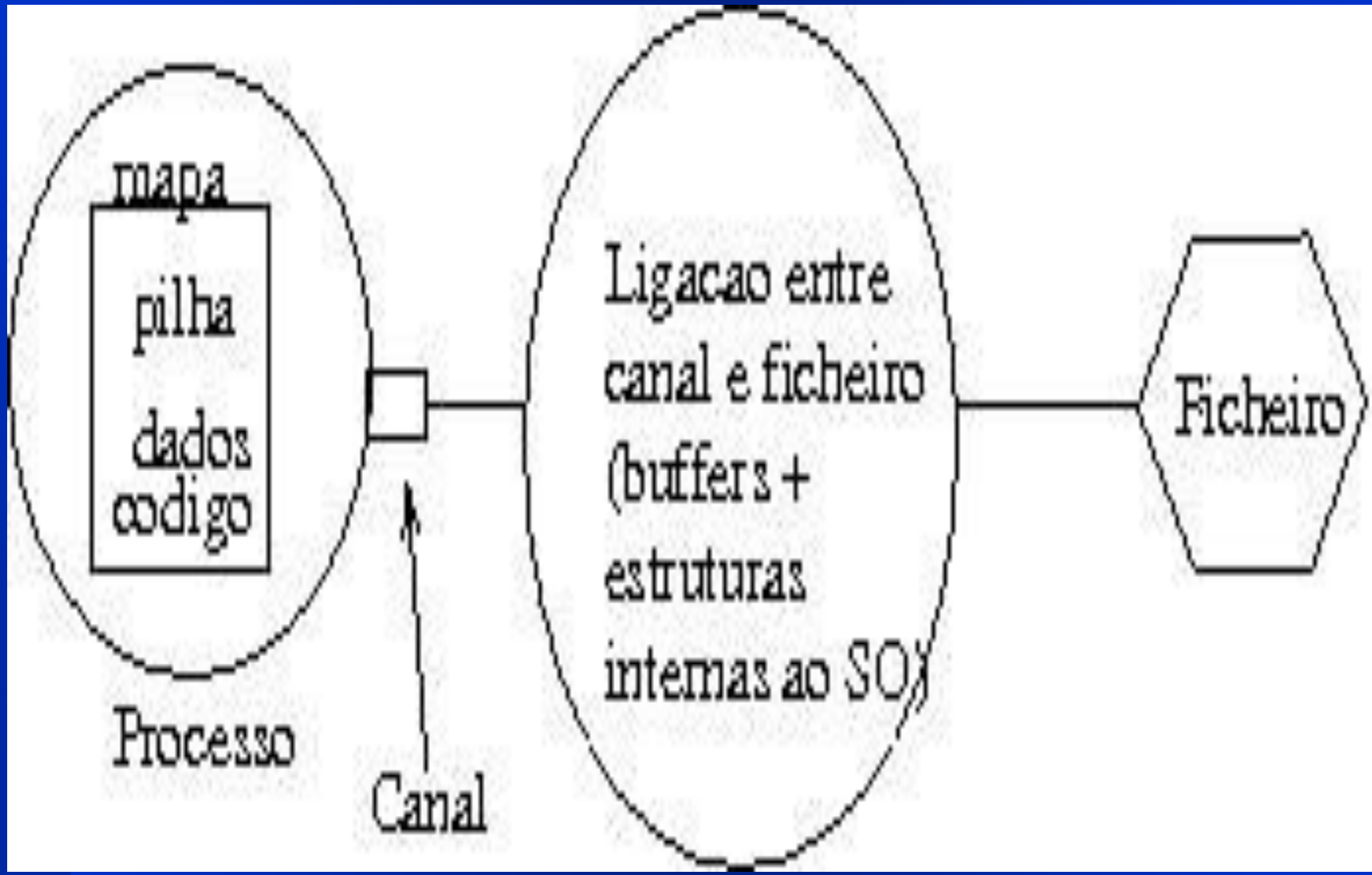
´´mode´´ indica as permissões de acesso

devolve o número de um canal aberto para o ficheiro, que pode ser usado em operações seguintes

# Acesso a um ficheiro

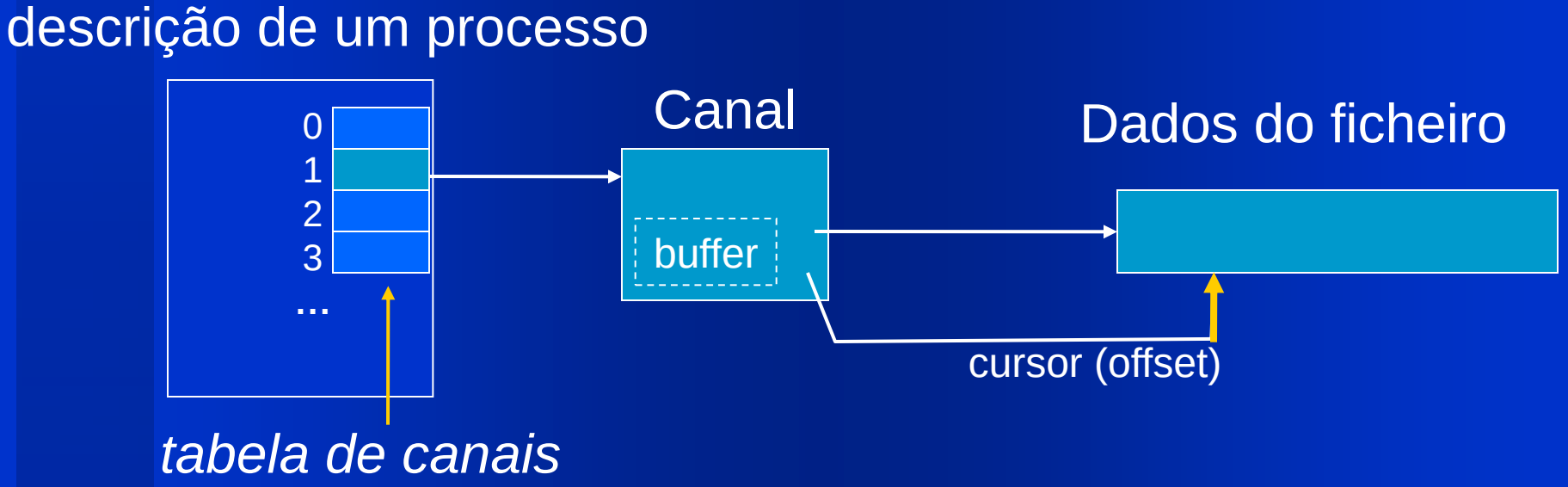
- **1º passo - pedir ao SO acesso ao ficheiro. Permite ao SO:**
  - **Verificar se o ficheiro existe e onde estão os seus dados**
  - **Verificar se o processo pode usar o ficheiro**
  - **Inicializar um novo canal (*buffers* de sistema e um cursor de posição no ficheiro)**
- **Depois - as restantes operações serão mais fáceis (referem apenas o canal)**
  - **Leitura ou escrita no canal**

# Canal dá acesso a buffers de SO



# Canal aberto

- **Canal de entrada/saída: abstracção do SO**
- **Um processo acede a um ficheiro pedindo ao SO um canal de acesso**



# open()

```
int open(char *filename, int flags)
```

abre um canal para aceder ao ficheiro ´´**filename**´´

com os modos de acesso indicados em ´´**flags**´´ (eg. ler, escrever)

devolve o número de canal aberto: um número: 0..N (N é um parâmetro do SO)

(chamado *filedescriptor* no Unix)

# Retorno das chamadas ao SO

- **As chamadas ao sistema retornam inteiros:**
  - **o valor devolvido pela operação:**

```
f = open( "fich1", O_RDONLY);
```

  - **´´f´´ é um número de canal´válido >= 0**

**ou**

  - **uma indicação de erro: -1**
    - **neste caso a variável global `errno` tem o número do erro**

```
f = open("fich1",O_RDONLY);
if (f == -1)  perror("fich1");
else …
```

    - **o manual descreve os erros possíveis**

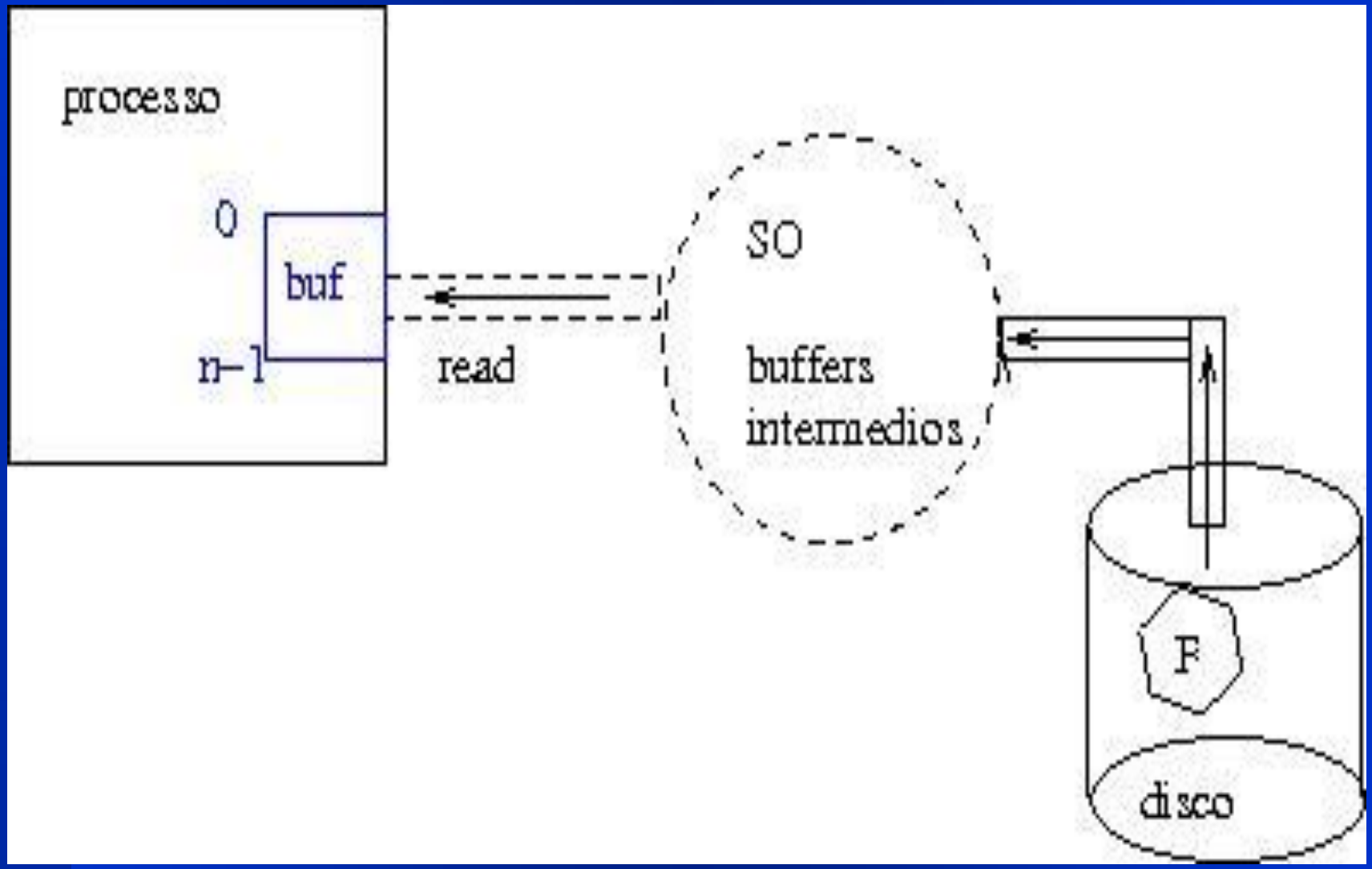
# Ler de um canal aberto

```
int read(int fd,void *buf,int count)
```

```
char buffer[3];
… f = open(…,O_RDONLY);
n = read( f, buffer, 3);
…
```

**Ler de um canal anteriormente aberto**

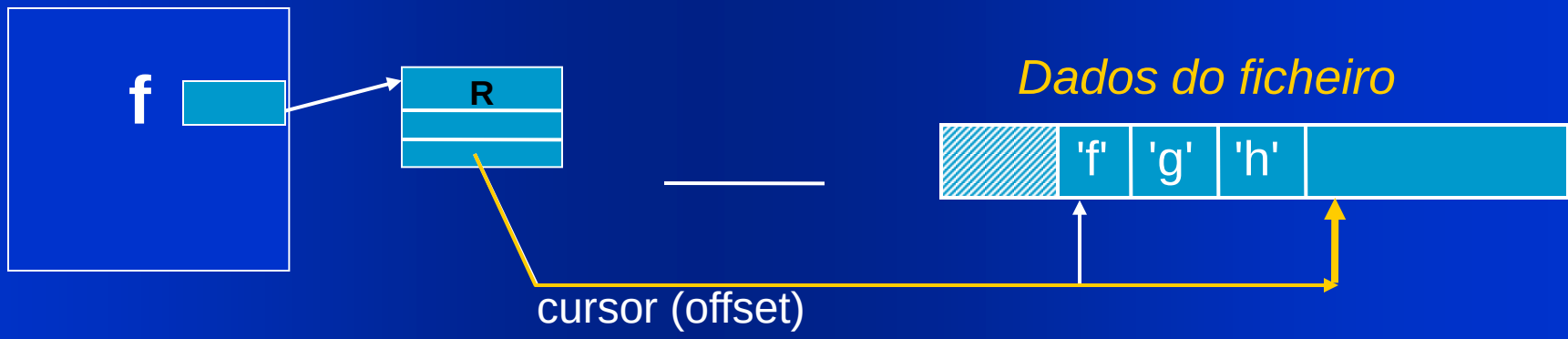
# Buffers: locais ao processo e de SO



# Leitura sequencial

```
int read(int fd, void *buf, int count)
```

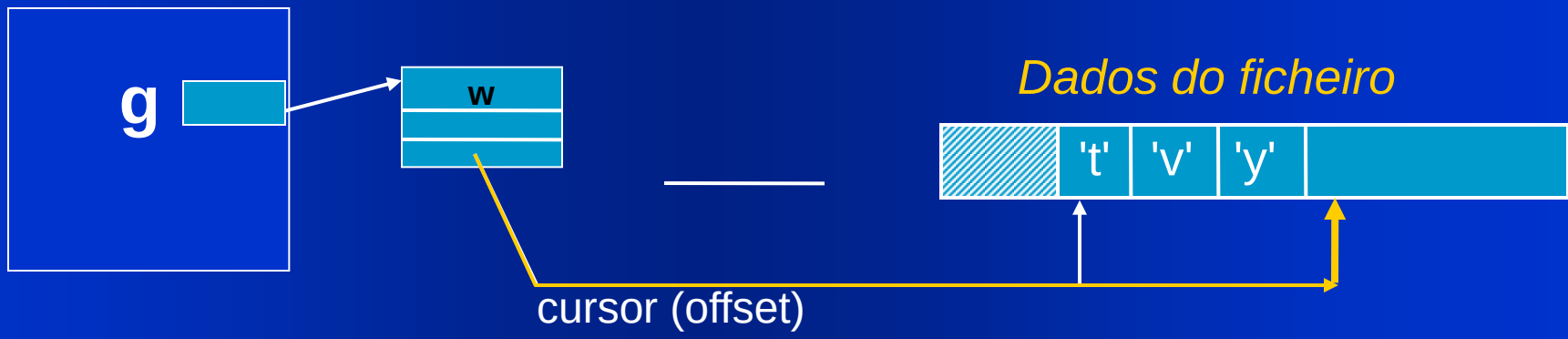
```
char buffer[3];
… f = open(…,O_RDONLY);
n = read( f, buffer, 3);
…
```



# Escrita num canal aberto

```
int write(int fd, void *buf, int count)
```

```
char buffer[3] = {'t','v','y'};
… g = open(…,O_WRONLY);
n = write(g, buffer, 3);
…
```



# Copiar do canal 0 para o 1

...

```
while ((n = read(0, buf, BUFSIZE)) > 0)
        write(1, buf,n);
```

...

# Implementação

...

...

# Abertura de canal: open()

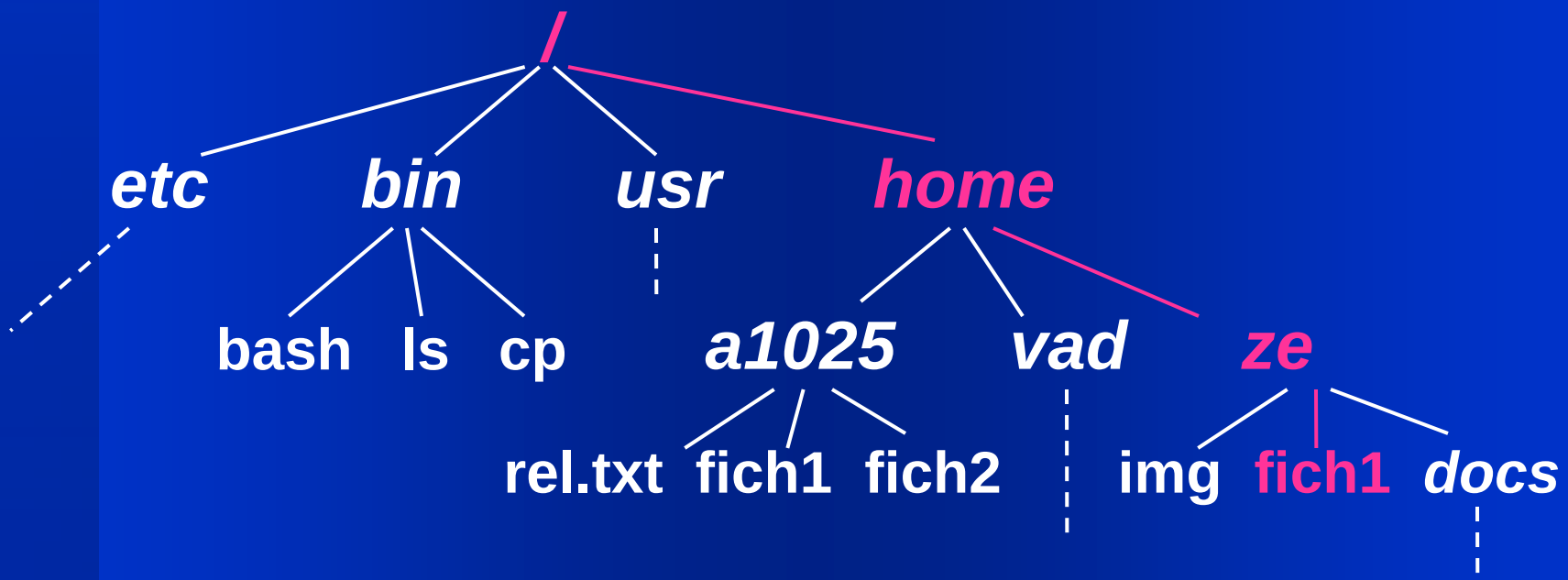
```
int open(char *filename, int flags)
```

**1º Percorre as directorias para verificar permissões e encontrar o ficheiro...**

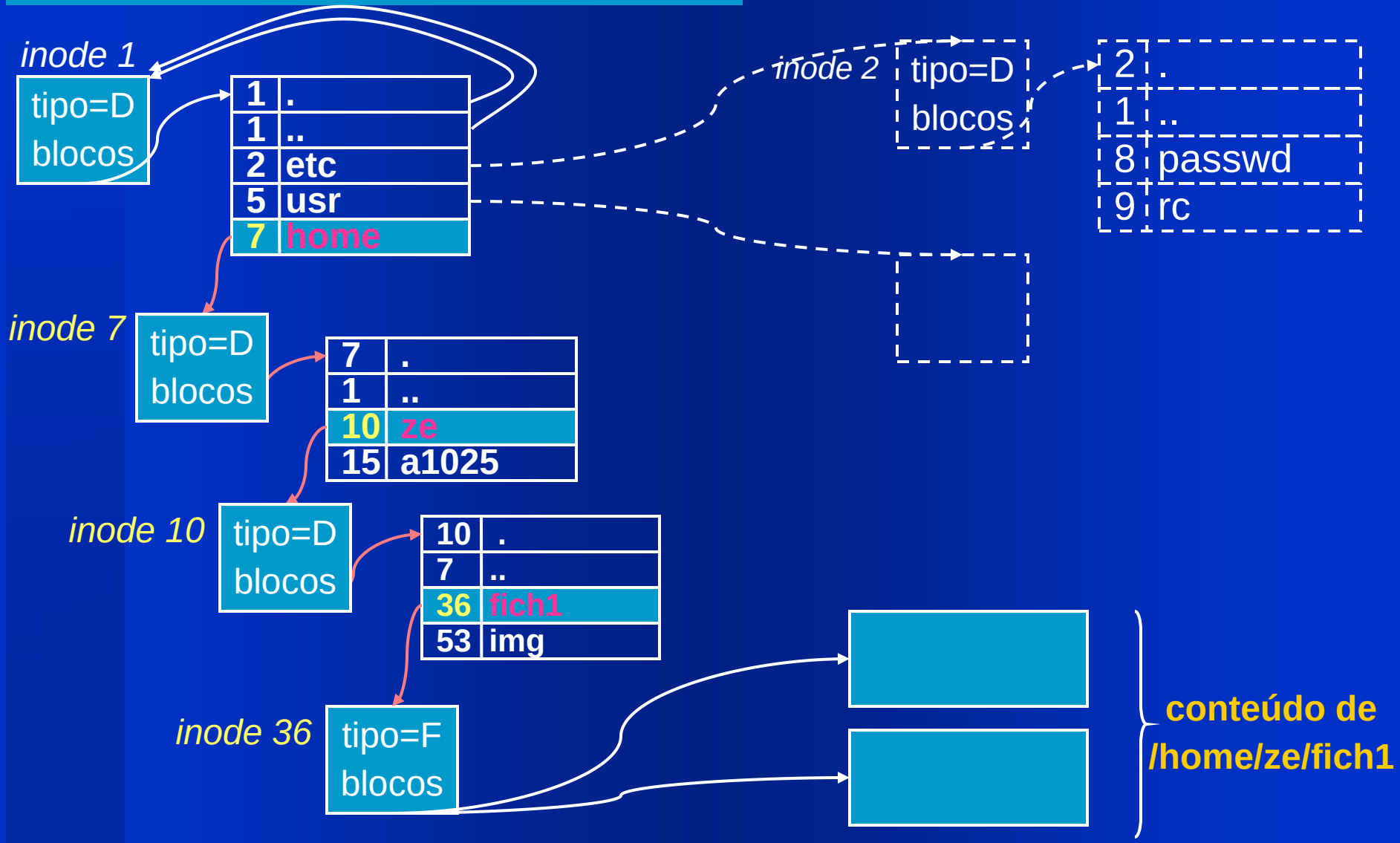
**e trazer o seu descritor (i-node) de disco para memória...**

**no caminho, vai trazendo para memória todos os descritores (i-nodes) encontrados no ´´path name´´ absoluto do ficheiro...**

# Directorias e ficheiros



# Caminho /home/ze/fich1



# 2º passo: regista o canal no SO

- **Reserva uma entrada na Tab. Canais do processo com um apontador para a Tabela Global de Ficheiros Abertos**

# 2º passo: regista o canal no SO

- **Reserva uma entrada na Tab. Canais do processo com um apontador para a Tabela Global de Ficheiros Abertos**
- **Na Tabela Global cada entrada tem:**
  - **o valor corrente do cursor (offset);**

# 2º passo: regista o canal no SO

- **Reserva uma entrada na Tab. Canais do processo com um apontador para a Tabela Global de Ficheiros Abertos**
- **Na Tabela Global cada entrada tem:**
  - **o valor corrente do cursor (offset);**
  - **o modo de abertura do canal (leitura, escrita ou leitura e escrita);**

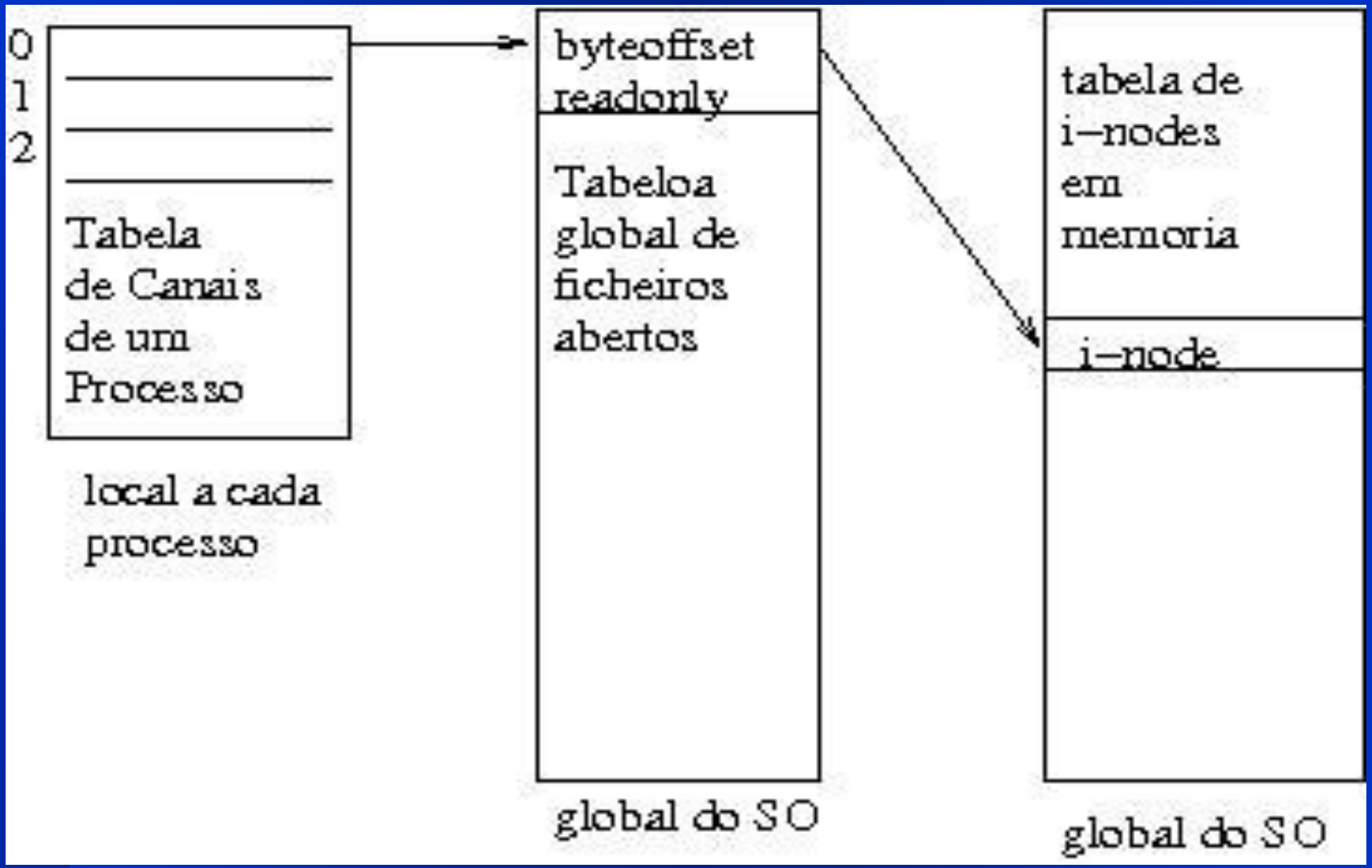
# 2º passo: regista o canal no SO

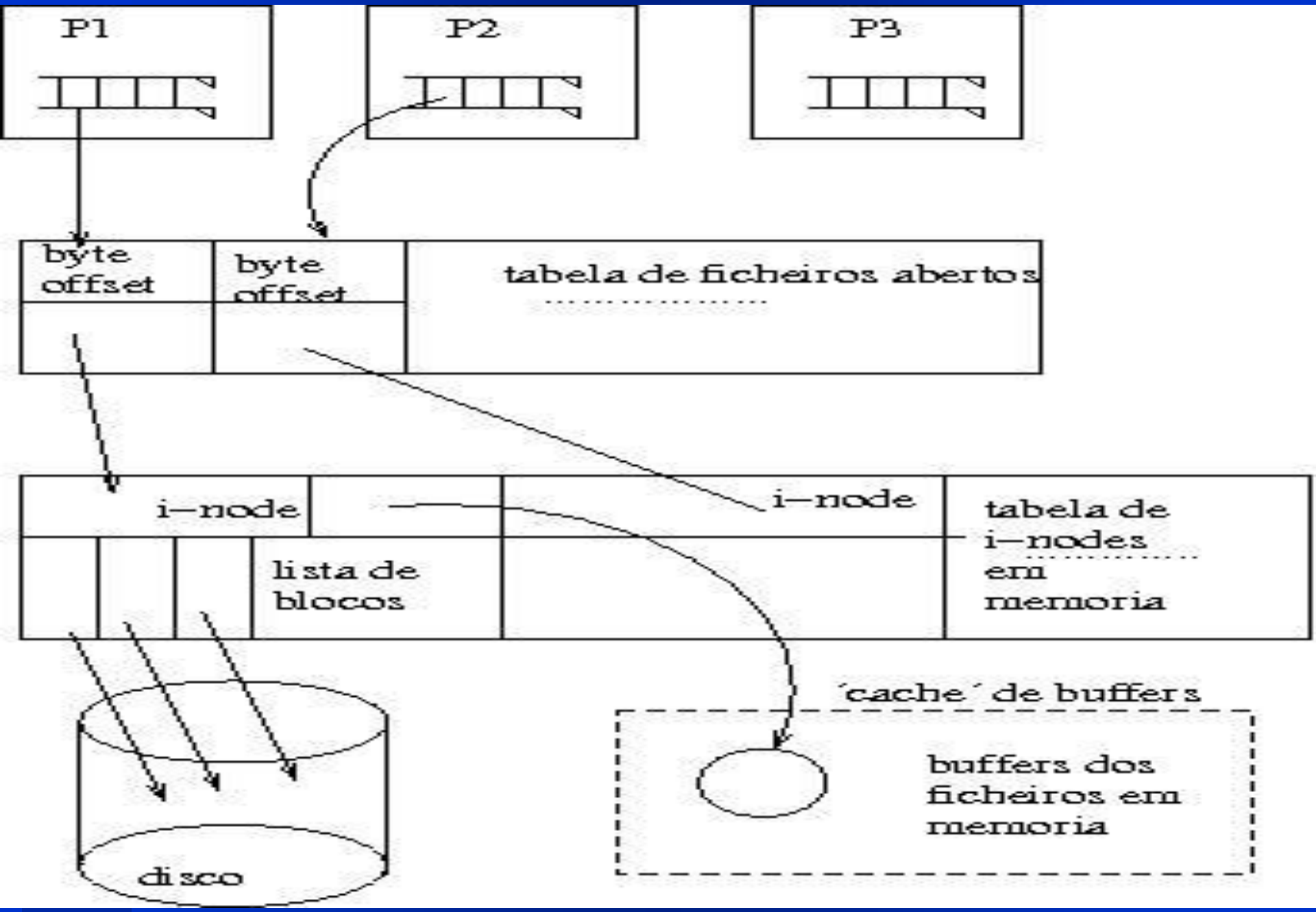
- **Reserva uma entrada na Tab. Canais do processo com um apontador para a Tabela Global de Ficheiros Abertos**
- **Na Tabela Global cada entrada tem:**
  - **o valor corrente do cursor (offset);**
  - **o modo de abertura do canal (leitura, escrita ou leitura e escrita);**
  - **um apontador para a Tabela de i-nodes em memória**

# 2º passo: regista o canal no SO

- **Reserva uma entrada na Tab. Canais do processo com um apontador para a Tabela Global de Ficheiros Abertos**
- **Na Tabela Global cada entrada tem:**
  - **o valor corrente do cursor (offset);**
  - **o modo de abertura do canal (leitura, escrita ou leitura e escrita);**
  - **um apontador para a Tabela de i-nodes em memória**
- **Na Tabela de i-nodes fica o descritor do ficheiro aberto**

# Tabelas: canais, ficheiros e inodes




P1
P2
P3
byte offset
byte offset
tabela de ficheiros abertos
i–node
i–node
tabela de i–nodes em memoria
lista de blocos
´cache´ de buffers
buffers dos ficheiros em memoria
disco

# Abrir um ficheiro de disco

**Reserva entrada na Tab.i-nodes em memória:**
**lê o i-node de disco para essa entrada**

# Abrir um ficheiro de disco

- **Reserva entrada na Tab.i-nodes em memória:**
  - **lê o i-node de disco para essa entrada**
- **As modificações do ficheiro: na versão do i-node em memória; são periodicamente escritas em disco, ou quando fechar o ficheiro**

# Abrir um ficheiro de disco

- **Reserva entrada na Tab.i-nodes em memória: lê o i-node de disco para essa entrada**
- **As modificações do ficheiro: na versão do i-node em memória; são periodicamente escritas em disco, ou quando fechar o ficheiro**
- **Abrir um ficheiro já aberto: usa-se a mesma entrada na Tab.i-nodes em memória**

# Abrir um ficheiro de disco

- **Reserva entrada na Tab.i-nodes em memória: lê o i-node de disco para essa entrada**
- **As modificações do ficheiro: na versão do i-node em memória; são periodicamente escritas em disco, ou quando fechar o ficheiro**
- **Abrir um ficheiro já aberto: usa-se a mesma entrada na Tab.i-nodes em memória**
- **Abrir um ficheiro: uma nova entrada na TabGlobalFicheirosAbertos (byte offset, modo, i-node)**

# Abrir um ficheiro de disco

- **Reserva entrada na Tab.i-nodes em memória: lê o i-node de disco para essa entrada**
- **As modificações do ficheiro: na versão do i-node em memória; são periodicamente escritas em disco, ou quando fechar o ficheiro**
- **Abrir um ficheiro já aberto: usa-se a mesma entrada na Tab.i-nodes em memória**
- **Abrir um ficheiro: uma nova entrada na TabGlobalFicheirosAbertos (byte offset, modo, i-node)**
- **Abrir um ficheiro: uma nova entrada na TabCanais do processo (aponta TabGlobal)**

# Ajustar posição do cursor: acesso directo

```
lseek(int fd, int offset, int origin)
```

origin: 0 - início do ficheiro

1 – posição corrente do cursor

2 – fim do ficheiro

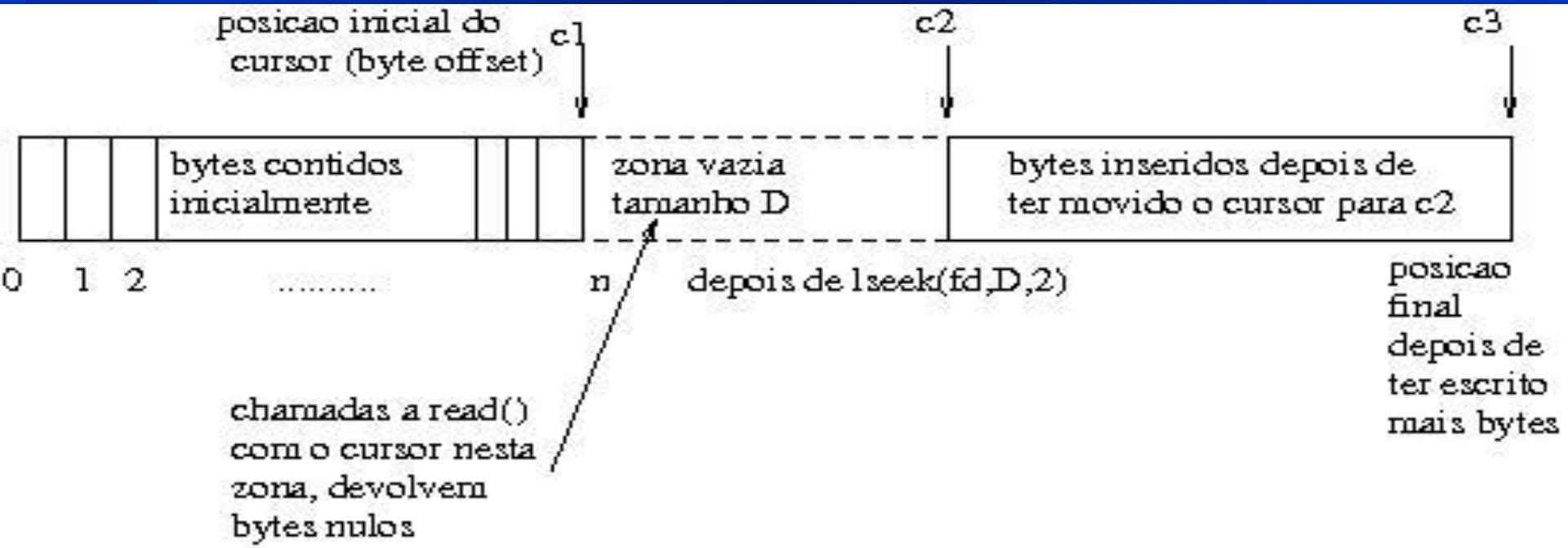
# Ajustar posição do cursor: acesso directo

**`lseek`(int fd, int offset, int origin)**

origin: 0 - início do ficheiro

1 – posição corrente do cursor

2 – fim do ficheiro



# Cursor associado ao canal

- **O cursor e os direitos R, W, RW ficam associados ao canal, não ao ficheiro**

# Cursor associado ao canal

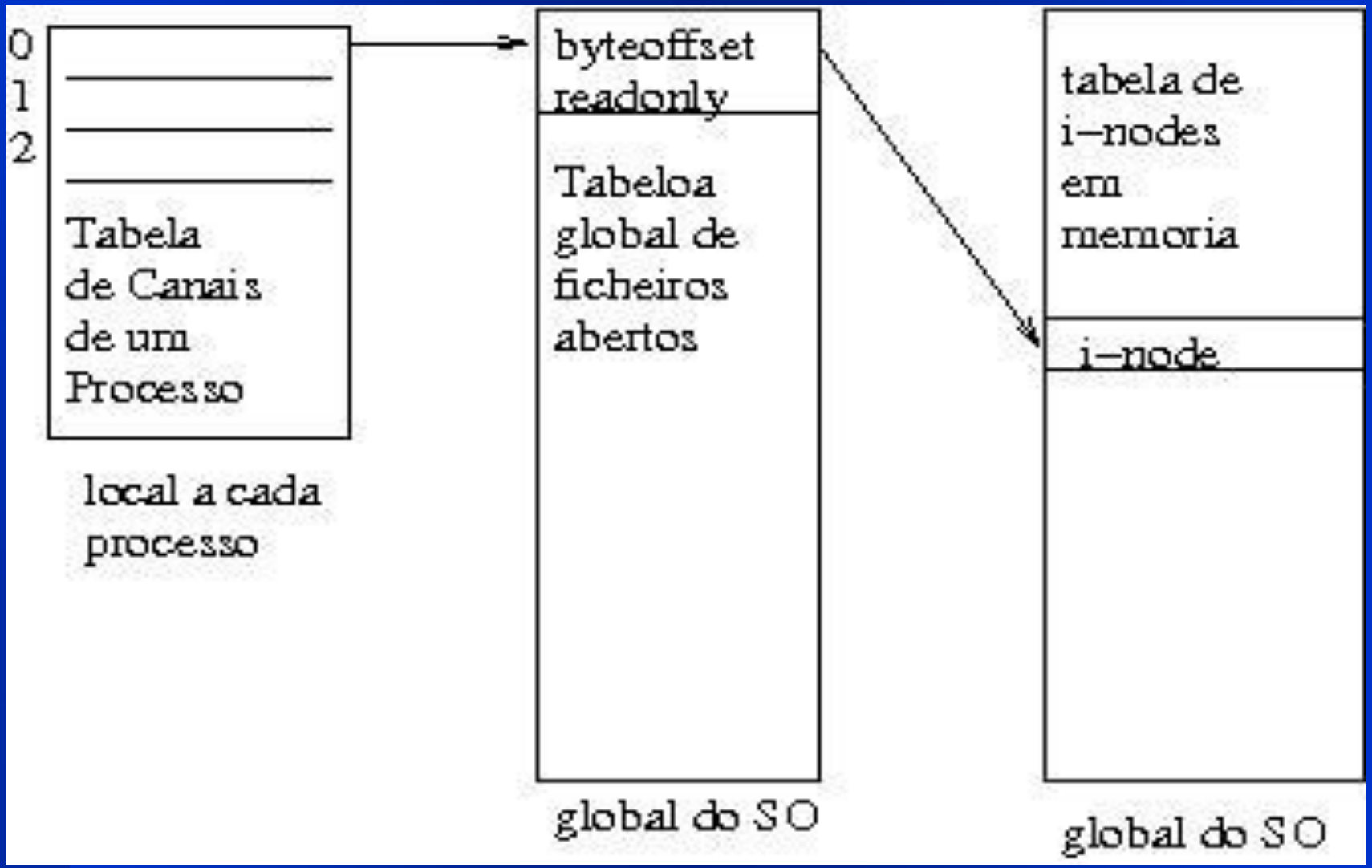
- **O cursor e os direitos R, W, RW ficam associados ao canal, não ao ficheiro**
- **Um processo pode abrir múltiplos canais para um mesmo ficheiro:**
  - **Uns para leitura … open(…, O_RDONLY)**
  - **Outros para escrita … open(…, O_WRONLY)**
  - **Para cada canal: → há um cursor associado**

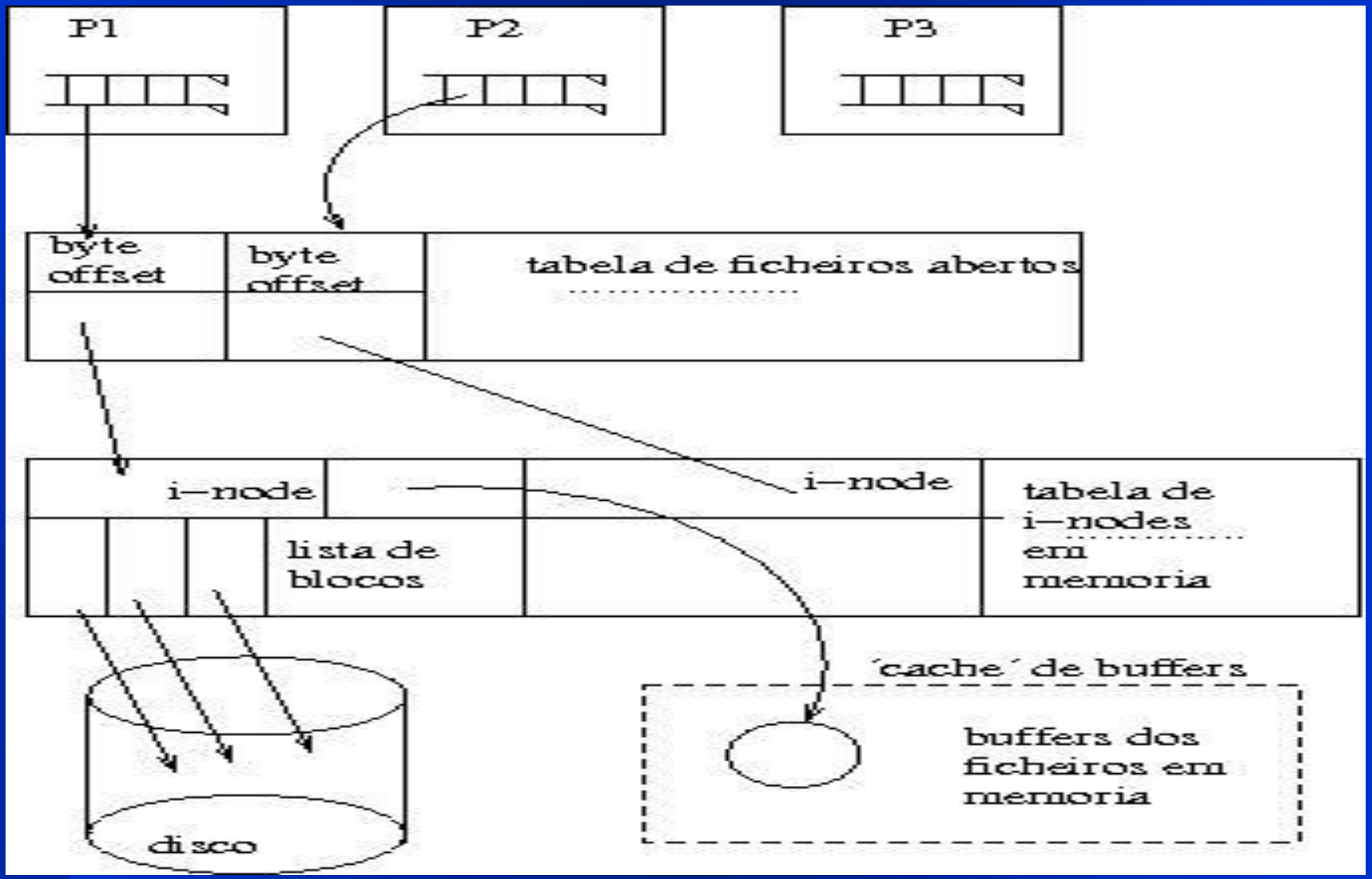
- Para cada canal aberto para um ficheiro, o SO regista o valor corrente do cursor, numa Tabela_Global_de_Ficheiros_Abertos
- Nessa tabela, cada entrada contém:
  - O valor corrente do cursor
  - O modo de acesso com que o canal foi aberto
  - Um apontador para o Descritor único (i-node) do ficheiro na

    Tabela_de_I-nodes_em_Memória

    Esta tabela dá acesso aos buffers em memória e aos blocos do ficheiro em disco

# Tabelas: canais, ficheiros e inodes




P1
P2
P3
byte offset
byte offset
tabela de ficheiros abertos
i–node
i–node
tabela de i–nodes em memoria
lista de blocos
´cache´ de buffers
buffers dos ficheiros em memoria
disco

# Abrir um ficheiro em disco

- **Reserva entrada na Tab.Inodes em memória: lê o i-node de disco para essa entrada**

# Abrir um ficheiro em disco

- **Reserva entrada na Tab.Inodes em memória: lê o i-node de disco para essa entrada**
- **Abrir um ficheiro já aberto: usa-se a mesma entrada na Tab.i-nodes em memória**
  - **incrementa contador de uso dessa entrada**

# Abrir um ficheiro em disco

- **Reserva entrada na Tab.Inodes em memória: lê o i-node de disco para essa entrada**
- **Abrir um ficheiro já aberto: usa-se a mesma entrada na Tab.i-nodes em memória**
  - **incrementa contador de uso dessa entrada**
- **Uma nova entrada na Tab. Global Ficheiros Abertos *(inicia: offset, modo, i-node)***

  ***(a partilha desta entrada fica para mais tarde…)***

# Abrir um ficheiro em disco

- **Reserva entrada na Tab.Inodes em memória: lê o i-node de disco para essa entrada**
- **Abrir um ficheiro já aberto: usa-se a mesma entrada na Tab.i-nodes em memória**
  - **incrementa contador de uso dessa entrada**
- **Uma nova entrada na Tab. Global Ficheiros Abertos *(inicia: offset, modo, i-node)***

  ***(a partilha desta entrada fica para mais tarde…)***
- **Uma nova entrada na Tab. Canais do processo (que passa a apontar a entrada na Tab. Global)**

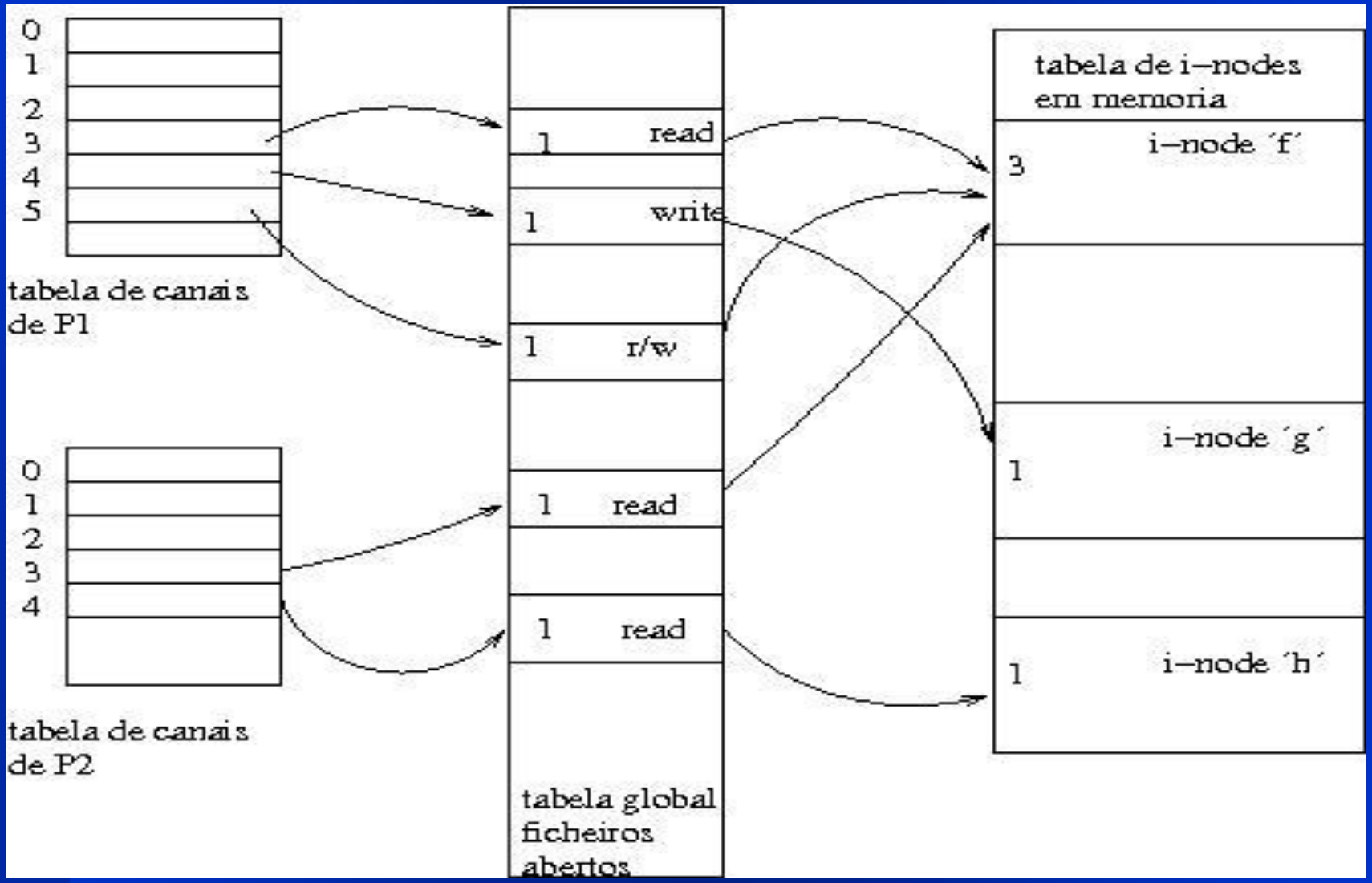
# Exemplo: acesso a 2 ficheiros

**No processo P1:** **na sua Tabela de Canais:**

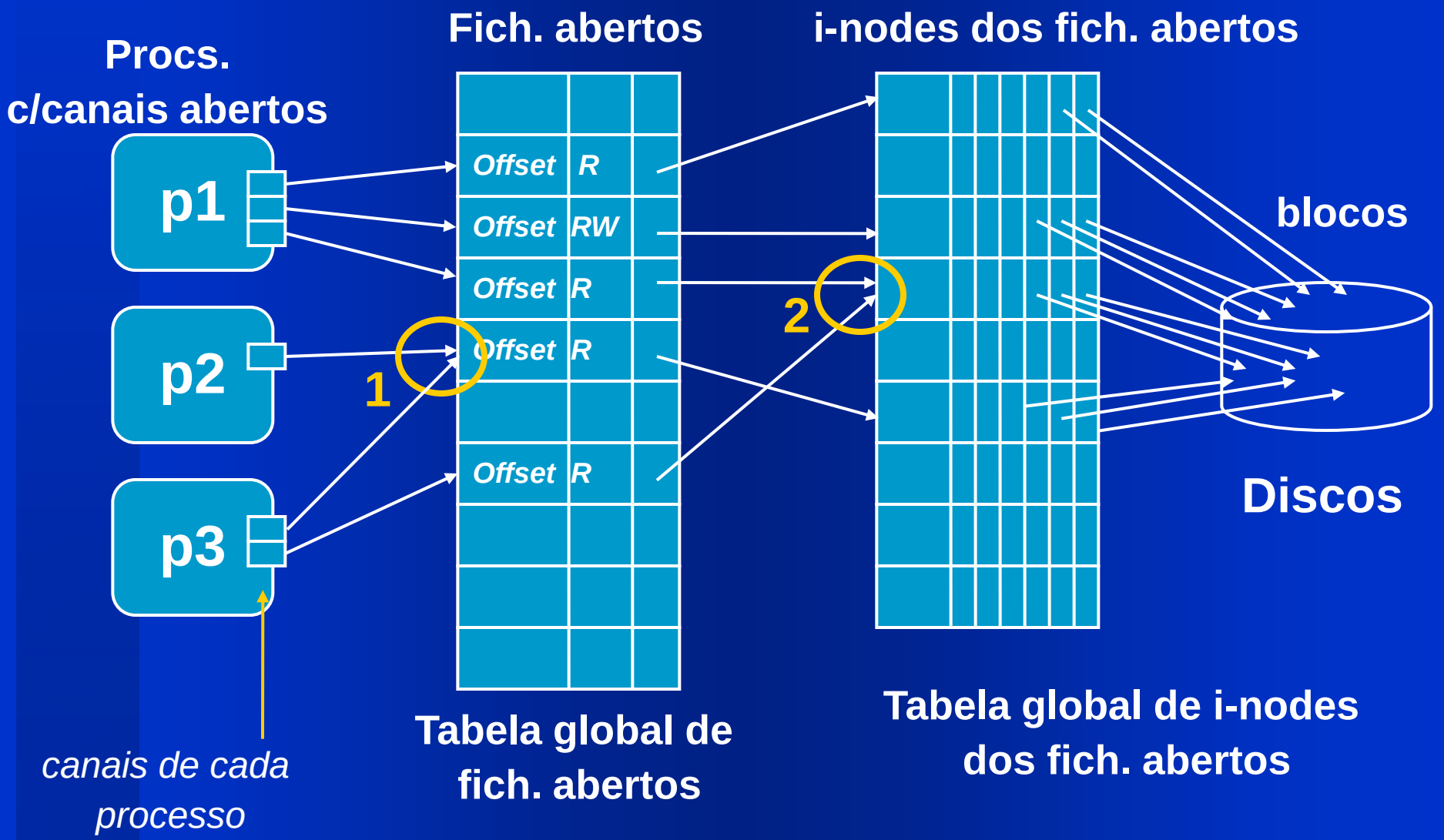
```
fd1 = open(´´f´´, O_RDONLY);      fd = 3
fd2 = open(´´g´´,O_WRONLY);       fd = 4
fd3= open(´´f´´,O_RDWR);          fd = 5
```

**No processo P2:** **na sua Tabela de Canais:**

```
fd4= open(´´f´´,O_RDONLY);        fd = 3
fd5= open(´´h´´,O_RDONLY);        fd = 4
```

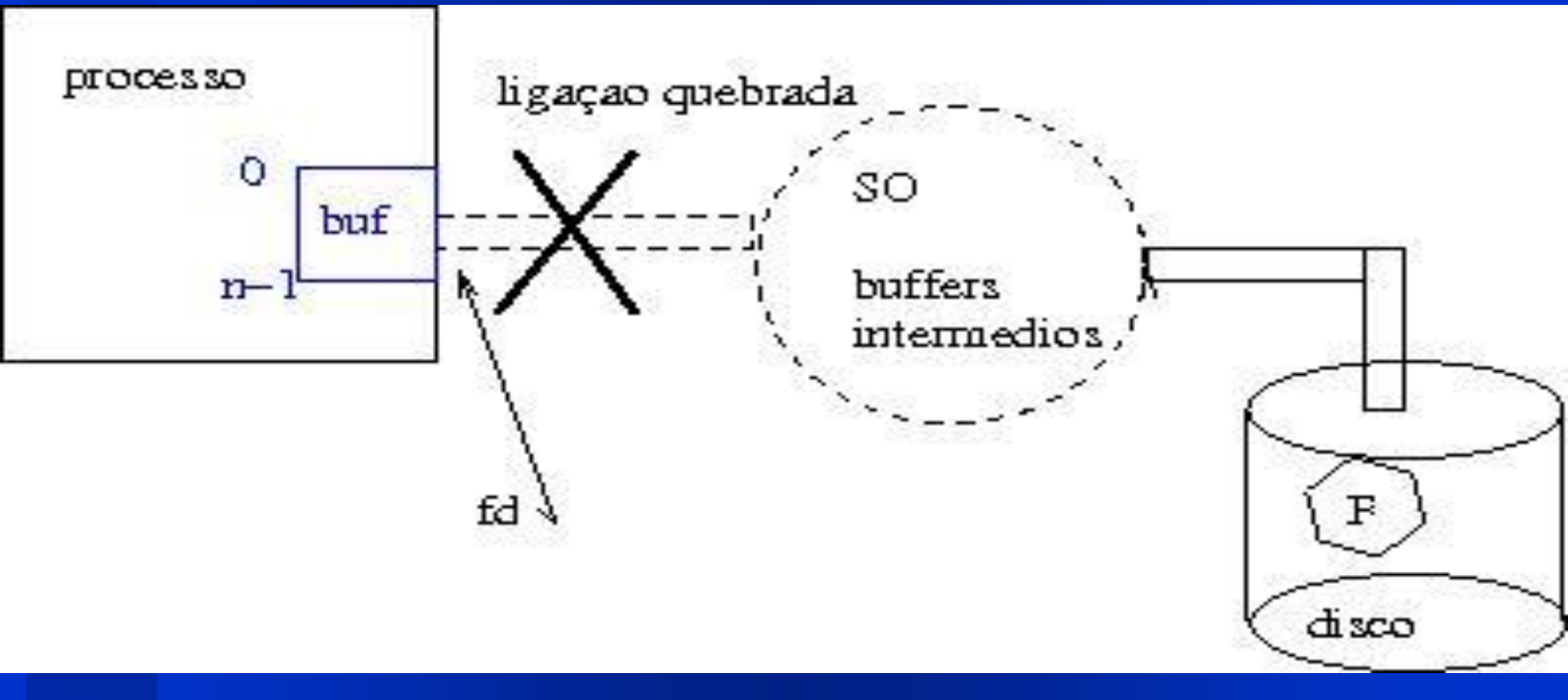

0
1
2
3
4
5
tabela de canais de P1
0
1
2
3
4
tabela de canais de P2
1 read
1 write
1 r/w
1 read
1 read
tabela global ficheiros abertos
tabela de i-nodes em memoria
3
i-node ´f´
1
i-node ´g´
1
i-node ´h´

# Tabelas em memória



# Fechar canal

**int close(int fd)**



# Fecho de ficheiro: close()

```
int close(int fd)
```

- **Liberta a entrada respectiva na tabela de canais do processo:**
  - **Este canal deixa de poder ser lido/escrito**
  - **A entrada respectiva da tabela será reutilizada pelo SO**

# Fecho de ficheiro: close()

```
int close(int fd)
```

- **Liberta a entrada respectiva na tabela de canais do processo:**
  - **Este canal deixa de poder ser lido/escrito**
  - **A entrada respectiva da tabela será reutilizada pelo SO**
- **Indica ao SO o fecho deste canal**
  - **Pode ainda haver outros canais abertos para este ficheiro...**
  - **Só liberta os buffers quando não houver mais canais abertos para um ficheiro**

# Fecho de ficheiro: close()

```
int close(int fd)
```

- Liberta a entrada respectiva na tabela de canais do processo:
  - Este canal deixa de poder ser lido/escrito
  - A entrada respectiva da tabela será reutilizada pelo SO
- Indica ao SO o fecho deste canal
  - Pode ainda haver outros canais abertos para este ficheiro...
  - Só liberta os buffers quando não houver mais canais abertos para um ficheiro
- Quando todos os canais fechados:
  - O i-node em memória é libertado
  - Mas o ficheiro continua a existir em disco...

# Libertação das entradas

- **As entradas nas duas tabelas só são libertadas quando os seus contadores de uso ficam nulos**

# Libertação das entradas

- As entradas nas duas tabelas só são libertadas quando os seus **contadores de uso** ficam nulos
- Na operação **close**(fd):
  - liberta a entrada fd na Tabela de Canais do processo

# Libertação das entradas

- As entradas nas duas tabelas só são libertadas quando os seus **contadores de uso** ficam nulos
- Na operação **close**(fd):
  - liberta a entrada fd na Tabela de Canais do processo
  - decrementa contador da entrada na Tabela Global antes referida por este descritor fd

# Libertação das entradas

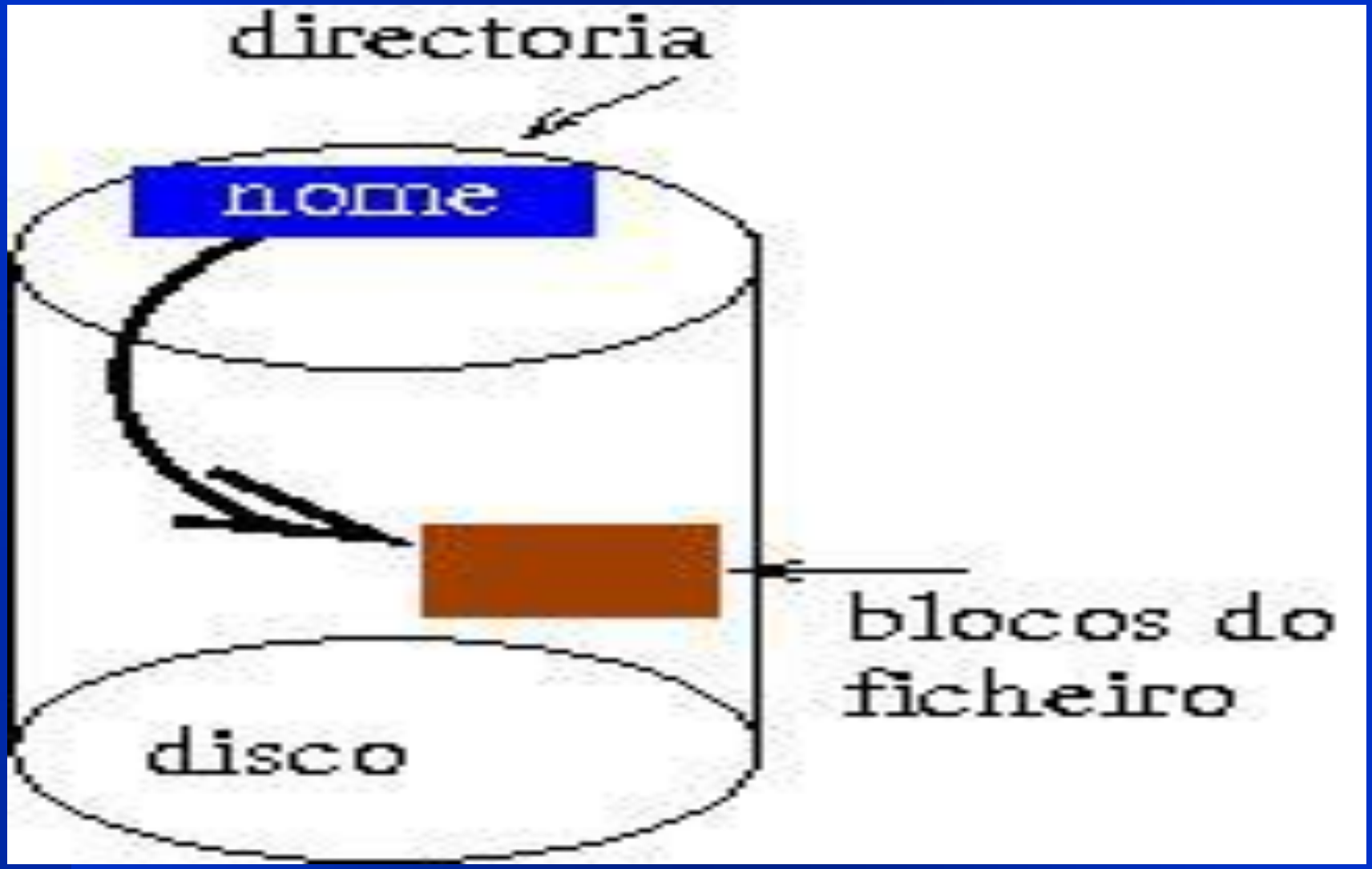
- As entradas nas duas tabelas só são libertadas quando os seus **contadores de uso** ficam nulos
- Na operação **close**(fd):
  - liberta a entrada fd na Tabela de Canais do processo
  - decrementa contador da entrada na Tabela Global antes referida por este descritor fd
  - se chegou a zero:
    - liberta esta entrada (este canal deixa de existir)

# Libertação das entradas

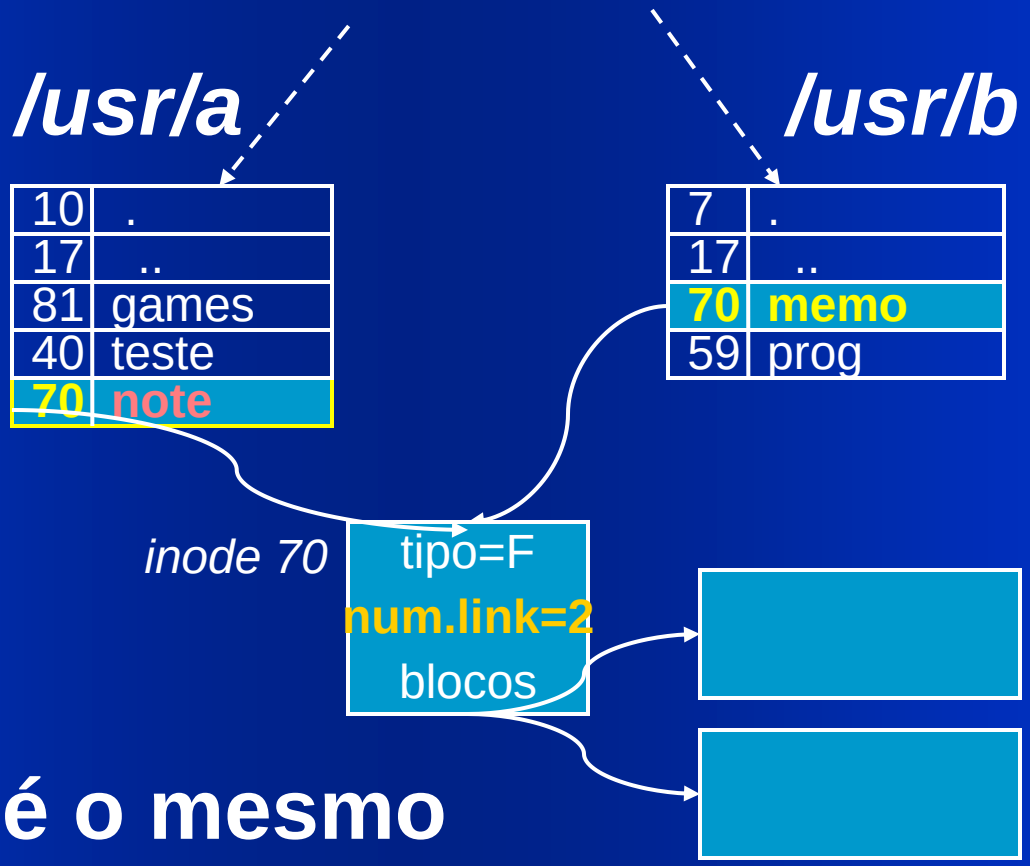
- As entradas nas duas tabelas só são libertadas quando os seus **contadores de uso** ficam nulos
- Na operação **close**(fd):
  - liberta a entrada fd na Tabela de Canais do processo
  - decrementa contador da entrada na Tabela Global antes referida por este descritor fd
  - se chegou a zero:
    - liberta esta entrada (este canal deixa de existir)
  - decrementa a respectiva entrada na Tabela de i-nodes
    - se esta chega a zero:
      - liberta esta entrada na Tabela de i-nodes (o ficheiro deixa de estar em uso)

# Destruir um ficheiro

- **Significa eliminar todos os seus nomes simbólicos: o ficheiro deixa de ser conhecido:**
  - ***Quem não tem nome, não existe…***


directoria
nome
blocos do ficheiro
disco

# Múltiplos nomes para um inode?




- **/usr/a/note é o mesmo ficheiro que /usr/b/memo**
- **cada entrada numa directoria é um *link***

# Definir sinónimos (‘*hard links*’)

```
int link(char *antigo, char *novo)
```

# Definir sinónimos (‘*hard links*’)

```
int link(char *antigo, char *novo)
```

- O nome novo fica associado ao mesmo inode que o nome antigo.

# Definir sinónimos (‘*hard links*’)

```
int link(char *antigo, char *novo)
```

- O nome **novo** fica associado ao mesmo inode que o nome **antigo**.
- Link cria uma nova entrada numa directoria, com um novo nome, mas com um inode igual ao de um ficheiro existente.

# Definir sinónimos (‘*hard links*’)

```
int link(char *antigo, char *novo)
```

- O nome **novo** fica associado ao mesmo inode que o nome **antigo**.
- Link cria uma nova entrada numa directoria, com um novo nome, mas com um inode igual ao de um ficheiro existente.
- Um mesmo ficheiro pode ser nomeado por múltiplos nomes diferentes.

# Definir sinónimos (‘*hard links*’)

```
int link(char *antigo, char *novo)
```

- O nome **novo** fica associado ao mesmo inode que o nome **antigo**.
- Link cria uma nova entrada numa directoria, com um novo nome, mas com um inode igual ao de um ficheiro existente.
- Um mesmo ficheiro pode ser nomeado por múltiplos nomes diferentes.
- Pode-se remover um dos nomes e o ficheiro permanece com o outro nome.

# Múltiplos 'links' de um inode

*/usr/a/*

games 81
test 40

*/usr/b/*

memo 70
prog 59

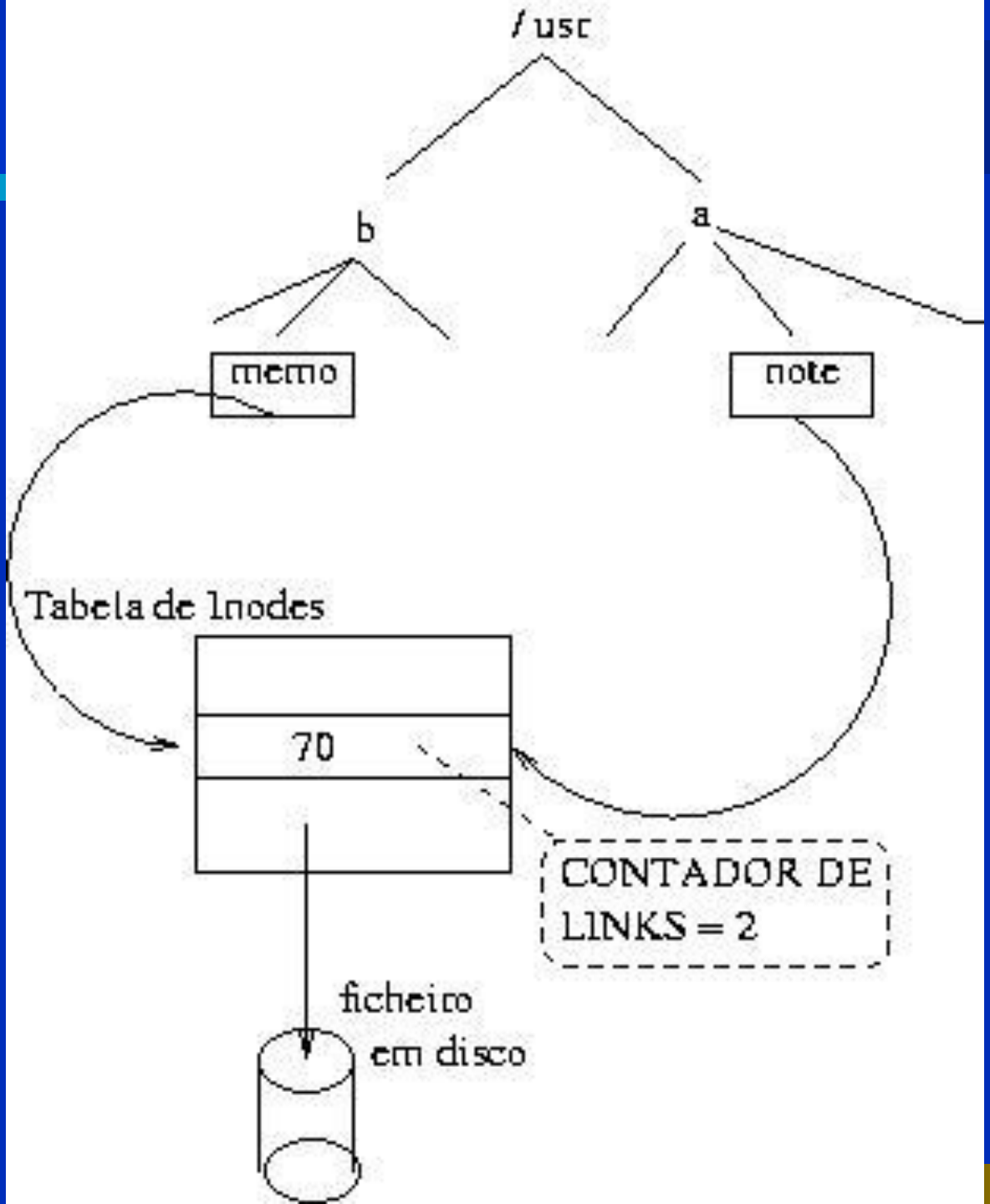
```
link("/usr/b/memo","/usr/a/note");
```

*/usr/a/*

games 81
test 40
note 70

*/usr/b/*

memo 70
prog 59


/ usr
b
a
memo
note
Tabela de Inodes
70
CONTADOR DE
LINKS = 2
ficheiro
em disco

# Destruir um ficheiro: unlink

- **Eliminar da directoria o par (nome, inode)**

```
int unlink(char *fname)
```

# Destruir um ficheiro: unlink

- **Eliminar da directoria o par (nome, inode)**

```
int unlink(char *fname)
```

- **Elimina o i-node, quando for possível :**
  - **pode haver outros nomes, sinónimos, para o mesmo ficheiro (decrementa contador links)**

# Destruir um ficheiro: unlink

- **Eliminar da directoria o par (nome, inode)**

```
int unlink(char *fname)
```

- **Elimina o i-node, quando for possível :**
  - **pode haver outros nomes, sinónimos, para o mesmo ficheiro (decrementa contador links)**
  - **pode haver múltiplos canais abertos**
    - **canal aberto:**

      **i-node em memória com contador de utilização > 0**

# Tratamento do unlink

- **Remove o nome da entrada da directoria**

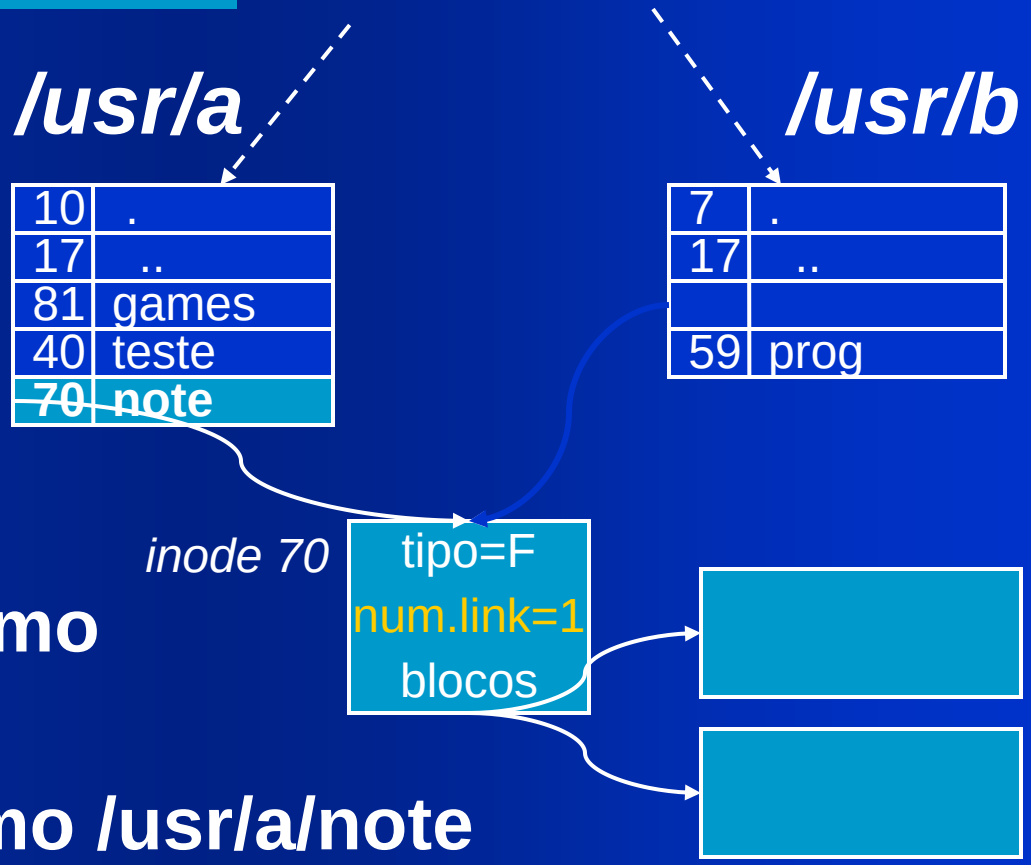
# Tratamento do unlink

- Remove o nome da entrada da directoria
- Decrementa o contador de links do inode
- Se contlinks = 0 →
  - se contador de uso > 0
  - (há canais abertos para o ficheiro)
    - mantém o i-node em memória

# Tratamento do unlink

- Remove o nome da entrada da directoria
- Decrementa o contador de links do inode
- Se **contlinks = 0** →
  - se **contador de uso > 0**
  - (há canais abertos para o ficheiro)
    - mantém o i-node em memória
  - quando **contador de uso = 0**
    - liberta o i-node de memória e de disco

# Renomear ficheiros




- **deixa de existir como /usr/b/memo**
- **passa a existir como /usr/a/note**
- **mudou de nome (incluindo a própria directoria)**

# Renomear ficheiros: rename

```
int rename(char *old, char *new)
```

- muda o nome de ficheiros e directorias
  - permite mudar o nome e mudar de directoria
- garante que o ficheiro não desaparece
- nas directorias mantém a sua consistência