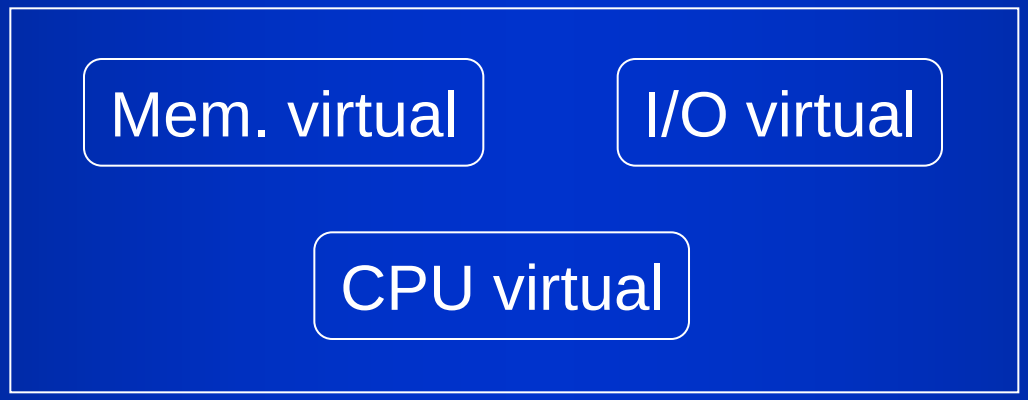
Threads concorrentes dentro de um mesmo Processo

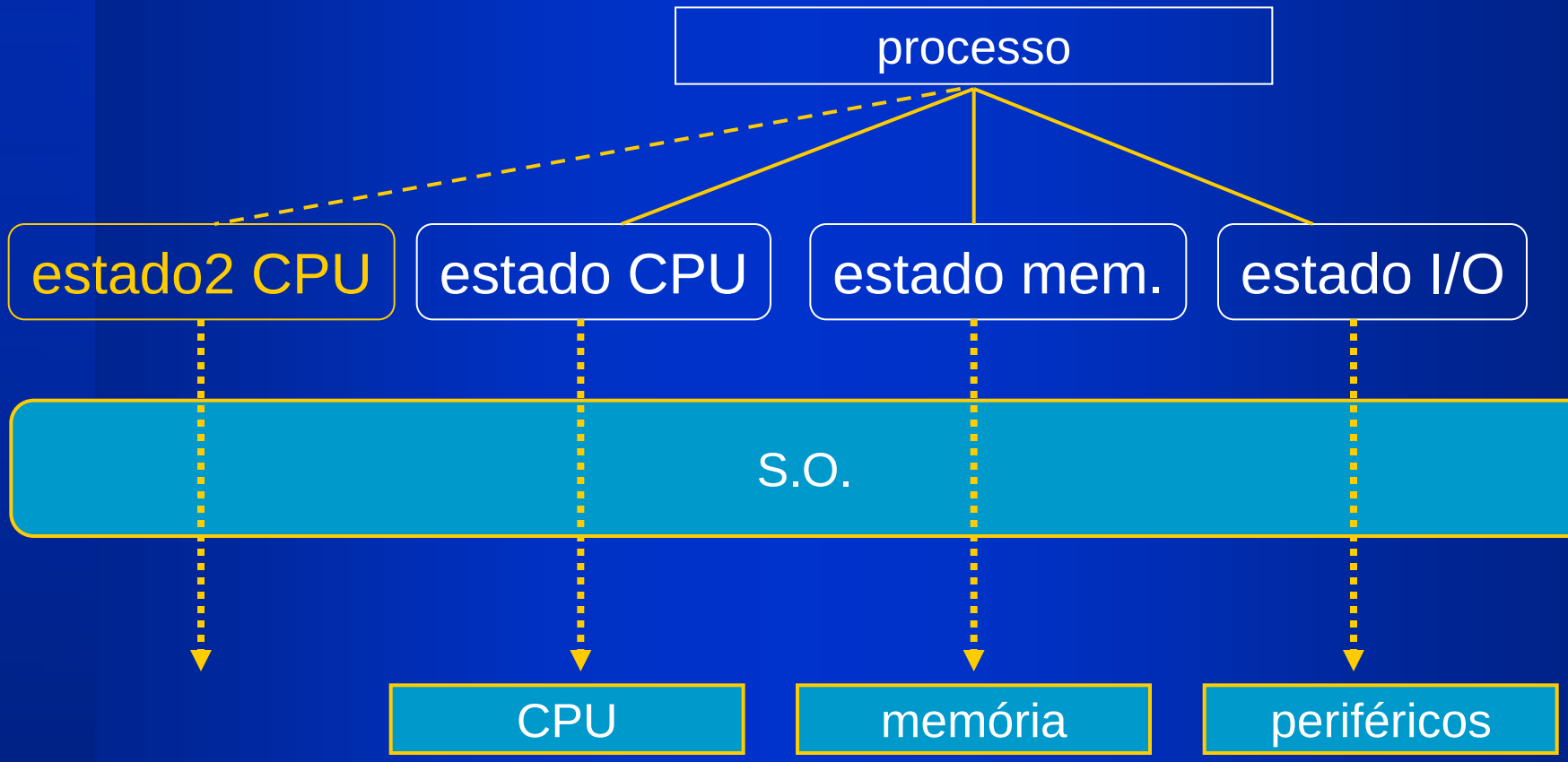
Processo

- **Unidade de execução concorrente**
  - sequência (fio ou fluxo) de execução de instruções (*execution thread*)
    - estado de execução: estado do CPU
- **Unidade de gestão de recursos**
  - memória: dados, código e pilha
    - estado da memória: mapa de memória e seu conteúdo
  - comunicação (I/O): ficheiros, IPC, outras abstracções para comunicar…
    - estado das comunicações: canais de I/O, etc…

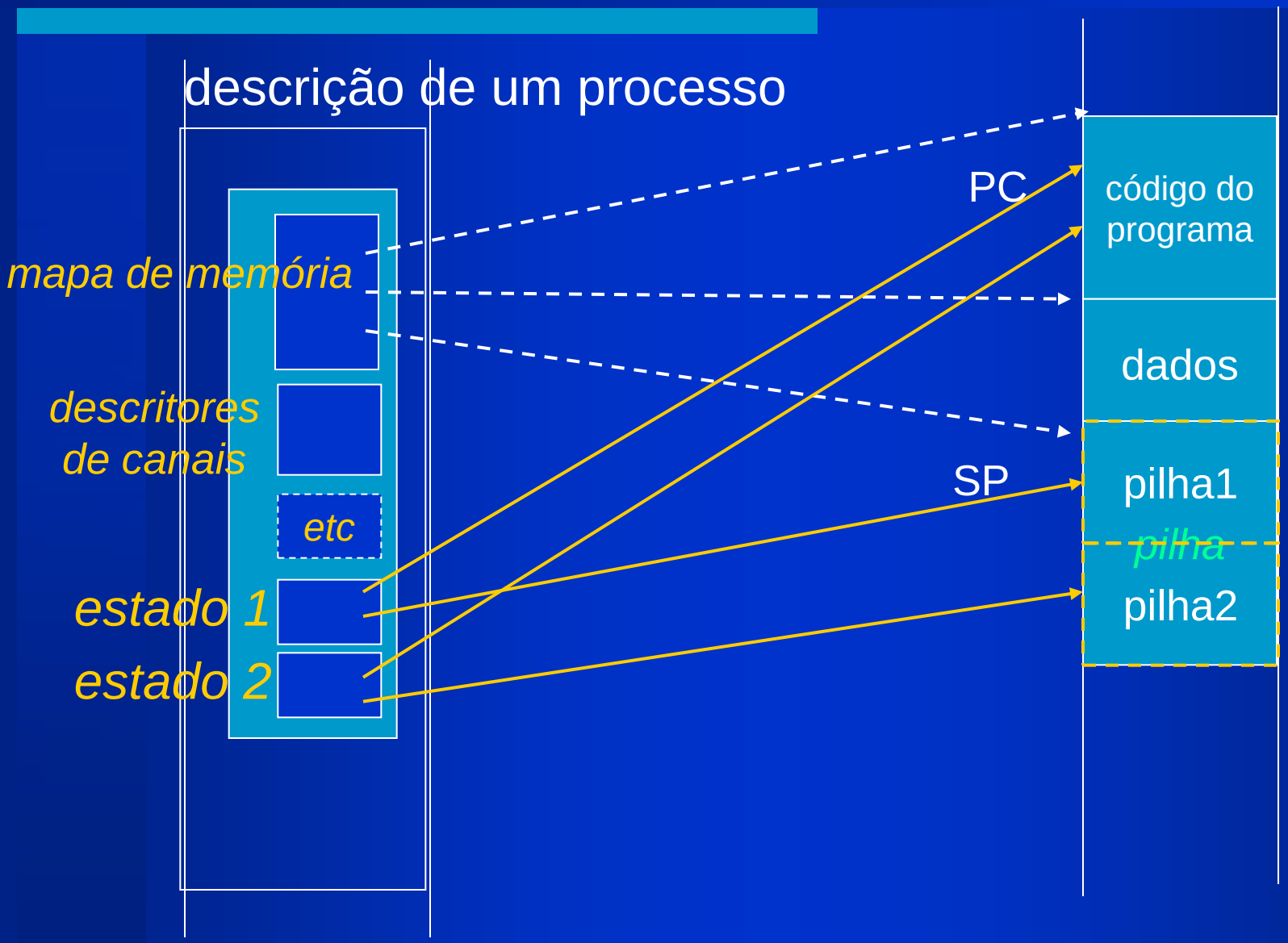
Máquina virtual de um processo



- **O SO gere e projecta-a na máquina real**
  - **Mem. Virtual → mapa de memória**
  - **CPU virtual → *gestão do CPU real***
  - **I/O virtual → gestão dos canais de I/O**


processo
estado2 CPU
estado CPU
estado mem.
estado I/O
S.O.
CPU
memória
periféricos

Processo em memória



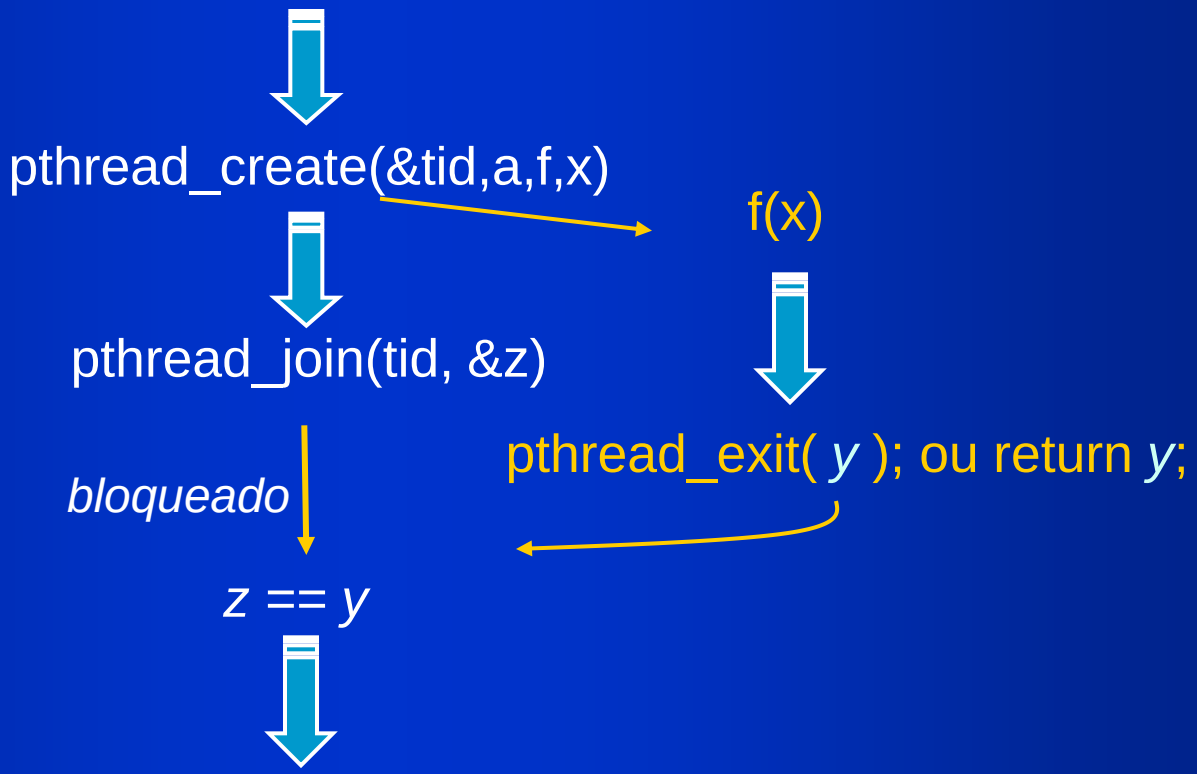
Novos *threads*

- **Cada processo é criado com um único *thread***
  - **este executa a partir do `fork()`**
  - **perante um `exec()`, o *thread* passa a executar a partir do `main()`**
- **Criar um novo *thread*:**

```
int pthread_create( pthread_t *tid,
                    pthread_attr_t *attr,
                    void *(*startF)(void *),
                    void *arg )
```

Vida de um *thread*: *Detached vs joinable*

- **`pthread_join()` aguarda pela terminação de um *joinable thread* (e pode receber um valor de retorno)**



- **Se não se pretende fazer *join* com um *thread* deve-se torná-lo *detached***

Chamadas p/threads vs processos

- pthread_create
- *"attributes"*
- pthread_exit
  - ou return
- pthread_join

- fork/execve
- *atributos diferentes*
- exit
  - ou return
- waitpid

Atributos (pthread_attr_t)

- **Definem um conjunto de características:**
  - **tipo de escalonamento, prioridade, "detachable/joinable"…**

```
pthread_attr_init, pthread_attr_destroy
pthread_attr_set*, pthread_attr_get*
```

  - **exemplo:**

```
int pthread_attr_setdetachstate(set, val)
int pthread_attr_getdetachstate(set)
```

  - **alternativa para este exemplo:**

```
pthread_detach(tid)
```

Argumentos do novo *thread*

- **Exemplo, criar um *thread* e passar-lhe um int:**

```
void *thmain( int *arg )
{…}
…
int arg=10;
…
pthread_create(&td,NULL, thmain, &arg);
    …
```

- **O novo *thread* começa executando:**

```
thmain(&arg)
```

Exemplo: servidor concorrente

- **Pseudo-código do ciclo principal para a criação dinâmica de trabalhadores:**

  (assumindo: pedido_t *pede)

```
while(1)
{
   recebe(pedido)
   ped = duplica(pedido);
   pthread_create(&tid,NULL,
                        trata, pede);
   pthread_detach(tid);
}
```

Problemas da concorrência interna

- **Vários threads num processo**
  - **podem existir situações de bloqueio que bloqueiam o processo (todos os threads)?**
  - **acessos concorrentes a dados e a I/O**
    - **p.e. acessos a estruturas de dados globais**
    - **situações menos óbvias: funções em bibliotecas**
  - **problemas de concorrência agora dentro do processo**
    - **necessidade de mecanismos de sincronização**
    - ***"tudo" é partilhável…***

Mutex - exclusão mútua

- **Mecanismo para exclusão mútua entre *threads* do mesmo processo**
- **Tipo de dados: `pthread_mutex_t`**
- **Possui dois estados:**
  - **fechado ou aberto**
- **Operações:**

```
pthread_mutex_init( mutex, attr )
pthread_mutex_destroy( mutex )
pthread_mutex_lock( mutex )
pthread_mutex_unlock( mutex )
```

Uso de um *mutex*

- **Garantir o acesso exclusivo a um recurso partilhado:**

```
pthread_mutex_t exmut;
pthread_mutex_init( exmut, attr );
```

em cada *thread*:

```
pthread_mutex_lock( exmut );
Reg. Crítica…
pthread_mutex_unlock( exmut );
```

- **Semelhante a um semáforo binário, mas não é um semáforo:**
  - **só possui dois estados: *locked, unlocked***
  - **não é incrementado ou decrementado**
  - **não funciona como contador ou para sinalização entre *threads***

Fork/execve e *threads*

- ***Estado das threads* após um `fork`**
  - **É duplicado o processo apenas com a *thread* que chamou o `fork`**
  - **`Mas: dados herdados podem estar inconsistentes no mapa do filho...`**
- **O mais seguro é não misturar as duas**
- ***Estado das threads* após um `execve`**
  - **Termina todas as *threads* antes de invocar a função `main do novo executável`**
  - ***A call to any exec function from a process with more than one thread shall result in all threads being terminated and the new executable image being loaded and executed.***