ACH2147 - Desenvolvimento de Sistemas de Informação Distribuídos

Aula 07 – Processos distribuídos

Norton Trevisan Roman

24 de abril de 2022

Processos distribuídos

- Revisão de Processos e Threads
- Threads no lado do cliente
- Threads no lado do servidor
- Virtualização

Processos distribuídos

- **Revisão de Processos e Threads**
- Threads no lado do cliente
- Threads no lado do servidor
- Virtualização

(Breve) Revisão de Processos e *Threads*

Processos × *Threads*

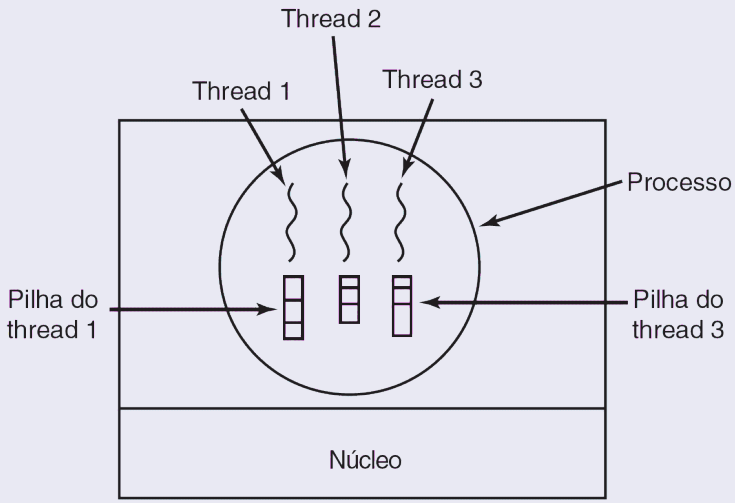
Processo com uma única *thread*

Processo com várias *threads*

(Breve) Revisão de Processos e *Threads*

Processos × *Threads*

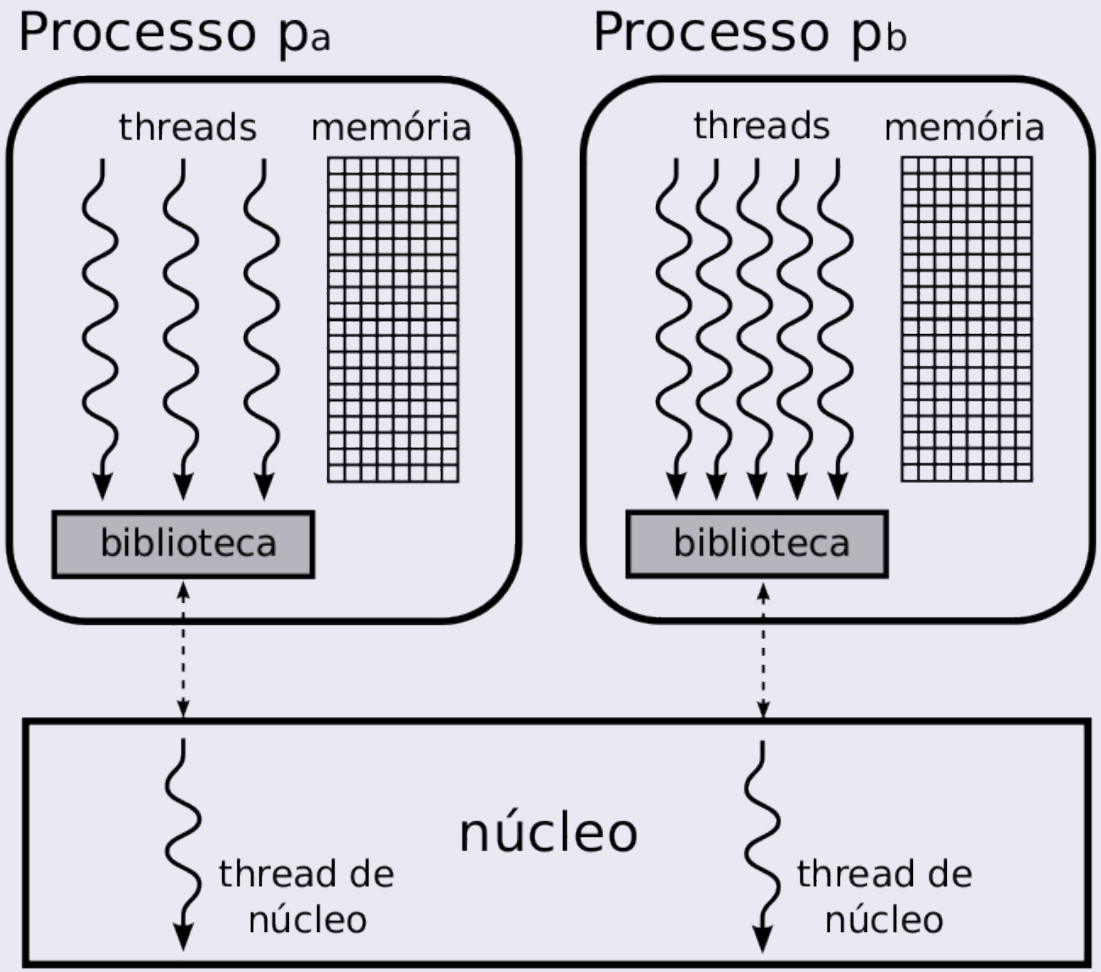
Itens por processo	Itens por thread
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais e manipuladores de sinais	
Informação de contabilidade	



Cada *thread* tem sua própria pilha de execução (pois chamam rotinas diferentes), embora compartilhe o espaço de endereçamento e todos seus dados

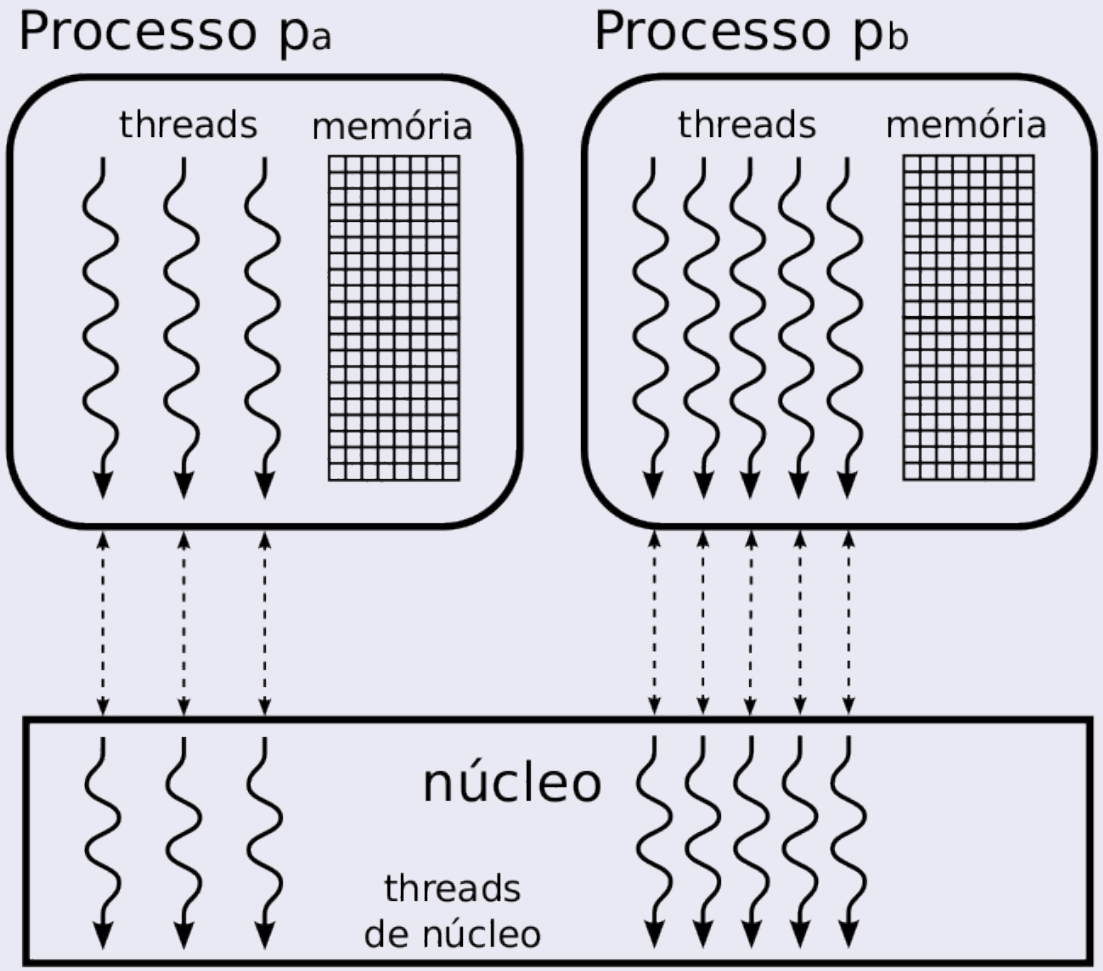
(Breve) Revisão de Processos e *Threads*

Threads no Espaço do Usuário: N para 1

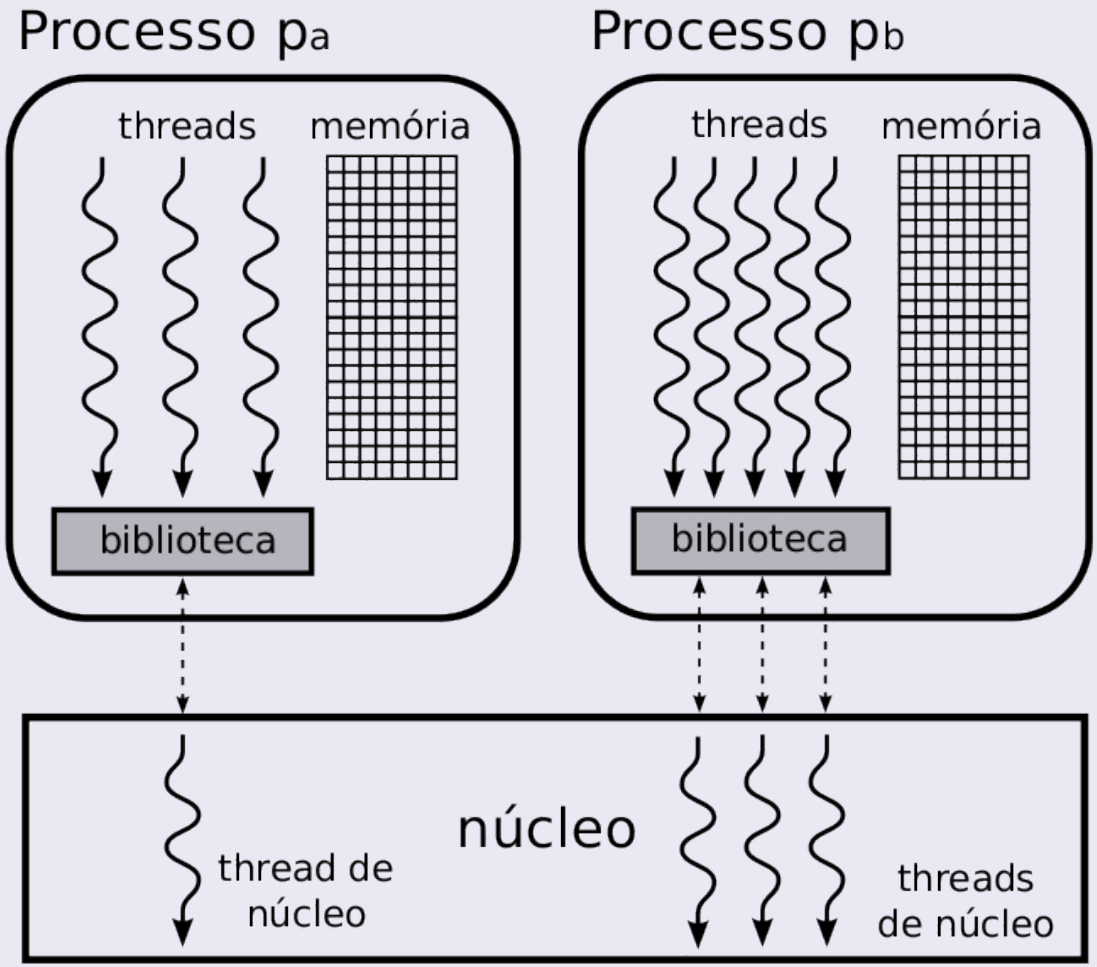


(Breve) Revisão de Processos e *Threads*

Threads no Espaço do Núcleo: 1 para 1



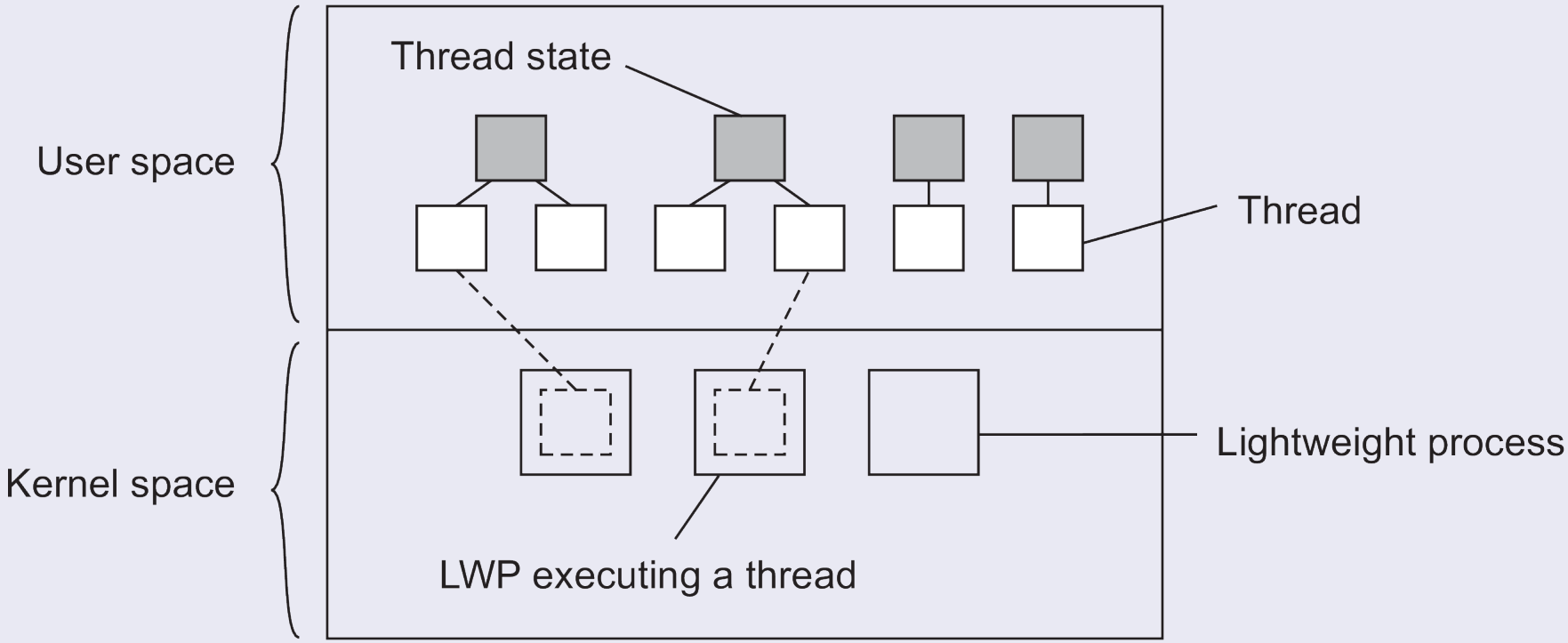
Threads Híbridas: N para M



(Breve) Revisão de Processos e *Threads*

Threads Híbridas: Solaris

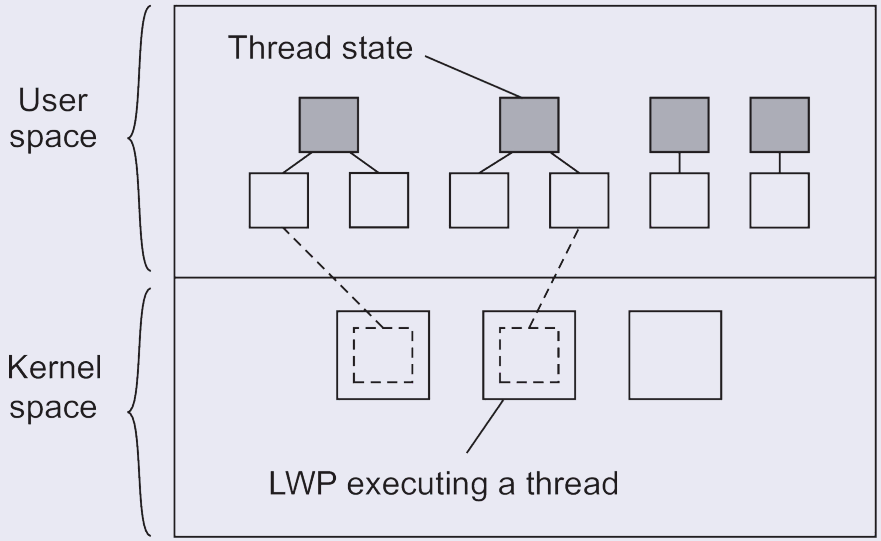
- Introduz uma abordagem em dois níveis:
  - Processos leves que podem executar *threads* de nível de usuário



(Breve) Revisão de Processos e *Threads*

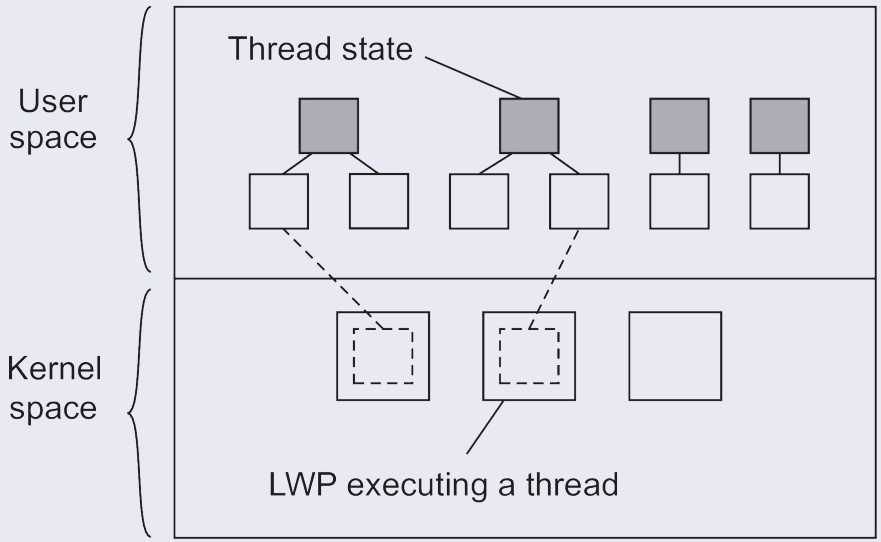
Threads no Solaris: Operação

- Uma thread de nível de usuário realiza uma chamada ao sistema
  - O LWP (*light-weight process*) que estiver executando aquela *thread* bloqueia
  - A *thread* continua associada àquele LWP



Threads no Solaris: Operação

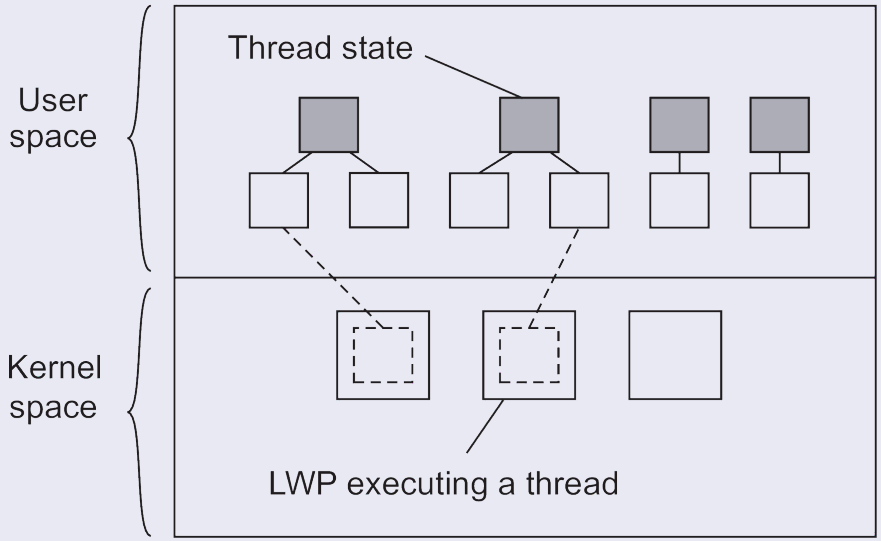
- O *kernel* pode escalonar outro LWP que possua uma *thread* pronta para execução
  - Essa thread pode ser trocada por qualquer outra *thread* de nível de usuário que esteja pronta



(Breve) Revisão de Processos e *Threads*

Threads no Solaris: Operação

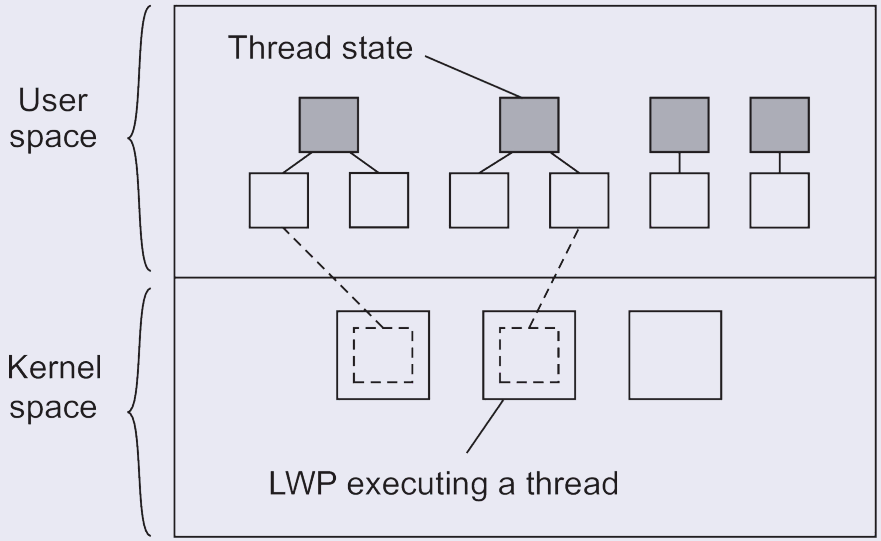
- Uma *thread* executa uma operação bloqueante (no nível de usuário, sem chamada ao sistema)
  - Faça a troca de contexto para uma *thread* pronta (e então a associe ao mesmo LWP)



(Breve) Revisão de Processos e *Threads*

Threads no Solaris: Operação

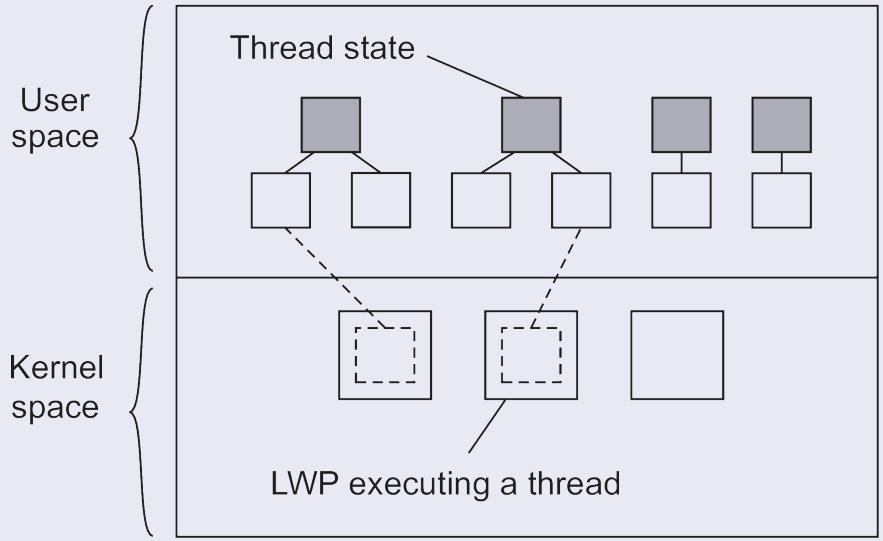
- Quando não há *threads* para executar, um LWP pode ficar ocioso, ou mesmo ser destruído pelo *kernel*



(Breve) Revisão de Processos e *Threads*

Threads no Solaris: Operação

- Quando não há *threads* para executar, um LWP pode ficar ocioso, ou mesmo ser destruído pelo *kernel*



Este conceito foi virtualmente abandonado – temos hoje *threads* ou no nível do usuário ou do *kernel*

Processos distribuídos

- Revisão de Processos e Threads
- **Threads no lado do cliente**
- Threads no lado do servidor
- Virtualização

Threads no lado do cliente

Escondendo a latência da rede

- Modo conveniente de permitir chamadas bloqueantes sem bloquear o processo
  - Tornam mais fácil a comunicação na forma de múltiplas conexões ao mesmo tempo

Threads no lado do cliente

Escondendo a latência da rede

- Modo conveniente de permitir chamadas bloqueantes sem bloquear o processo
  - Tornam mais fácil a comunicação na forma de múltiplas conexões ao mesmo tempo
- Estabelecem assim um alto grau de transparência de distribuição
  - Pois escondem as latências de comunicação
  - Geralmente iniciando a comunicação e imediatamente procedendo com alguma outra tarefa

Threads no lado do cliente

Exemplo

- Um navegador analisa a página HTML sendo recebida e descobre que *muitos outros arquivos devem ser baixados*

Threads no lado do cliente

Exemplo

- Um navegador analisa a página HTML sendo recebida e descobre que *muitos outros arquivos devem ser baixados*
- Cada arquivo é baixado por uma thread separada
  - Cada uma realiza uma requisição HTTP (bloqueante)

Threads no lado do cliente

Exemplo

- Um navegador analisa a página HTML sendo recebida e descobre que *muitos outros arquivos devem ser baixados*
- Cada arquivo é baixado por uma thread separada
  - Cada uma realiza uma requisição HTTP (bloqueante)
- À medida em que os arquivos chegam, o navegador os exibem

Threads no lado do cliente

Outro exemplo

- Múltiplas chamadas requisição–resposta (RPC) para outras máquinas

Threads no lado do cliente

Outro exemplo

- Múltiplas chamadas requisição–resposta (RPC) para outras máquinas
- Um cliente faz várias chamadas simultâneas, cada uma em uma thread diferente

Threads no lado do cliente

Outro exemplo

- Múltiplas chamadas requisição–resposta (RPC) para outras máquinas
- Um cliente faz várias chamadas simultâneas, cada uma em uma thread diferente
- Ele espera até que todos os resultados tenham chegado
  - Se as chamadas são a servidores diferentes, você pode ter um speed-up linear

Processos distribuídos

- Revisão de Processos e Threads
- Threads no lado do cliente
- **Threads no lado do servidor**
- Virtualização

Threads no lado do servidor

Melhor desempenho

- Iniciar uma thread é **muito** mais barato do que iniciar um novo processo

Threads no lado do servidor

Melhor desempenho

- Iniciar uma thread é **muito** mais barato do que iniciar um novo processo
- Ter servidores single-threaded impedem o uso de sistemas multiprocessados

Threads no lado do servidor

Melhor desempenho

- Iniciar uma thread é **muito** mais barato do que iniciar um novo processo
- Ter servidores single-threaded impedem o uso de sistemas multiprocessados
- Tal como nos clientes, esconda a latência da rede reagindo à próxima requisição enquanto a anterior está enviando sua resposta

Threads no lado do servidor

Melhor estrutura

- A maioria dos servidores faz muita E/S
  - Usar chamadas bloqueantes simples e bem conhecidas simplifica a estrutura geral

Threads no lado do servidor

Melhor estrutura

- A maioria dos servidores faz muita E/S
  - Usar chamadas bloqueantes simples e bem conhecidas simplifica a estrutura geral
- Programas multithreaded tendem a ser menores e mais fáceis de entender
  - Uma vez que simplificam o fluxo de controle

Threads no lado do servidor

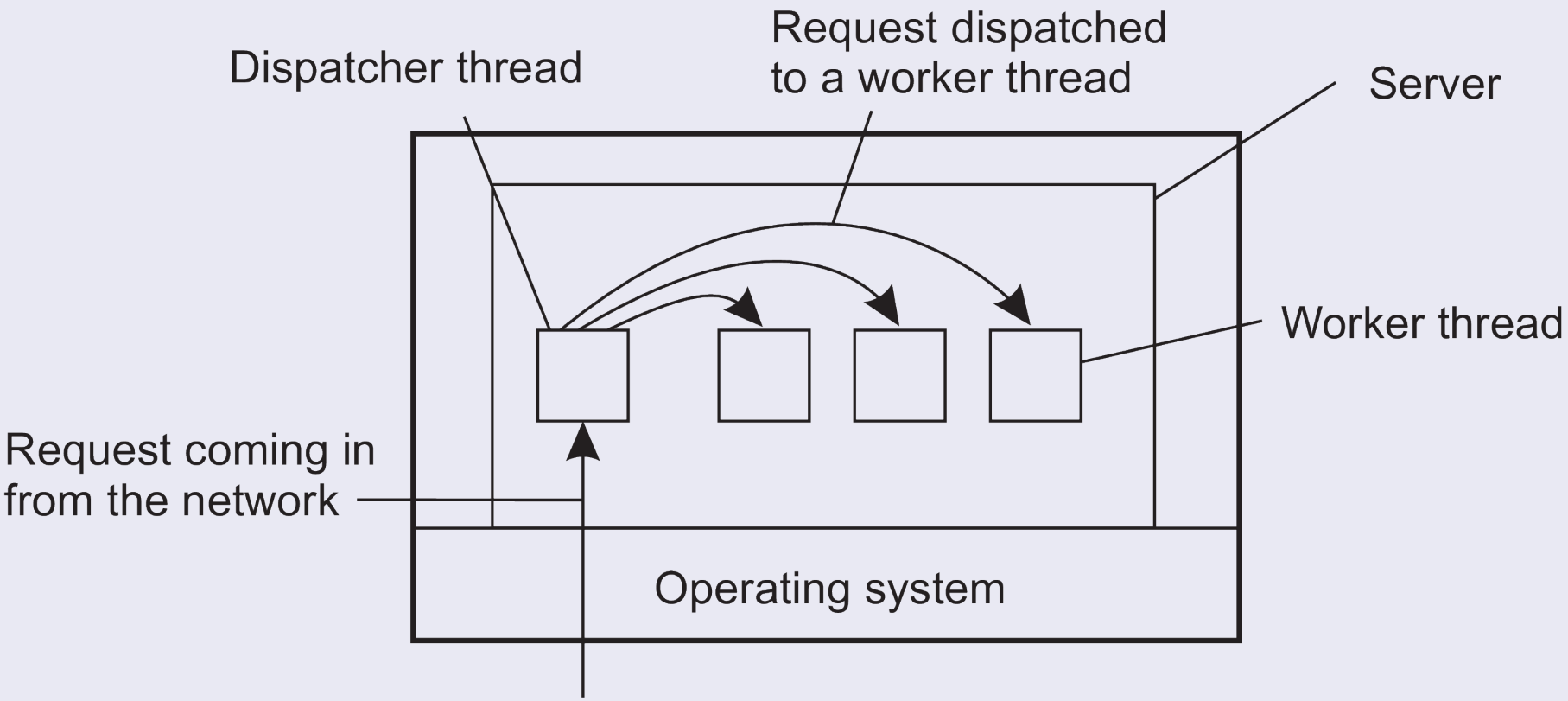
Melhor estrutura

- A maioria dos servidores faz muita E/S
  - Usar chamadas bloqueantes simples e bem conhecidas simplifica a estrutura geral
- Programas multithreaded tendem a ser menores e mais fáceis de entender
  - Uma vez que simplificam o fluxo de controle

Uma das principais razões da popularidade de multithreading é sua organização

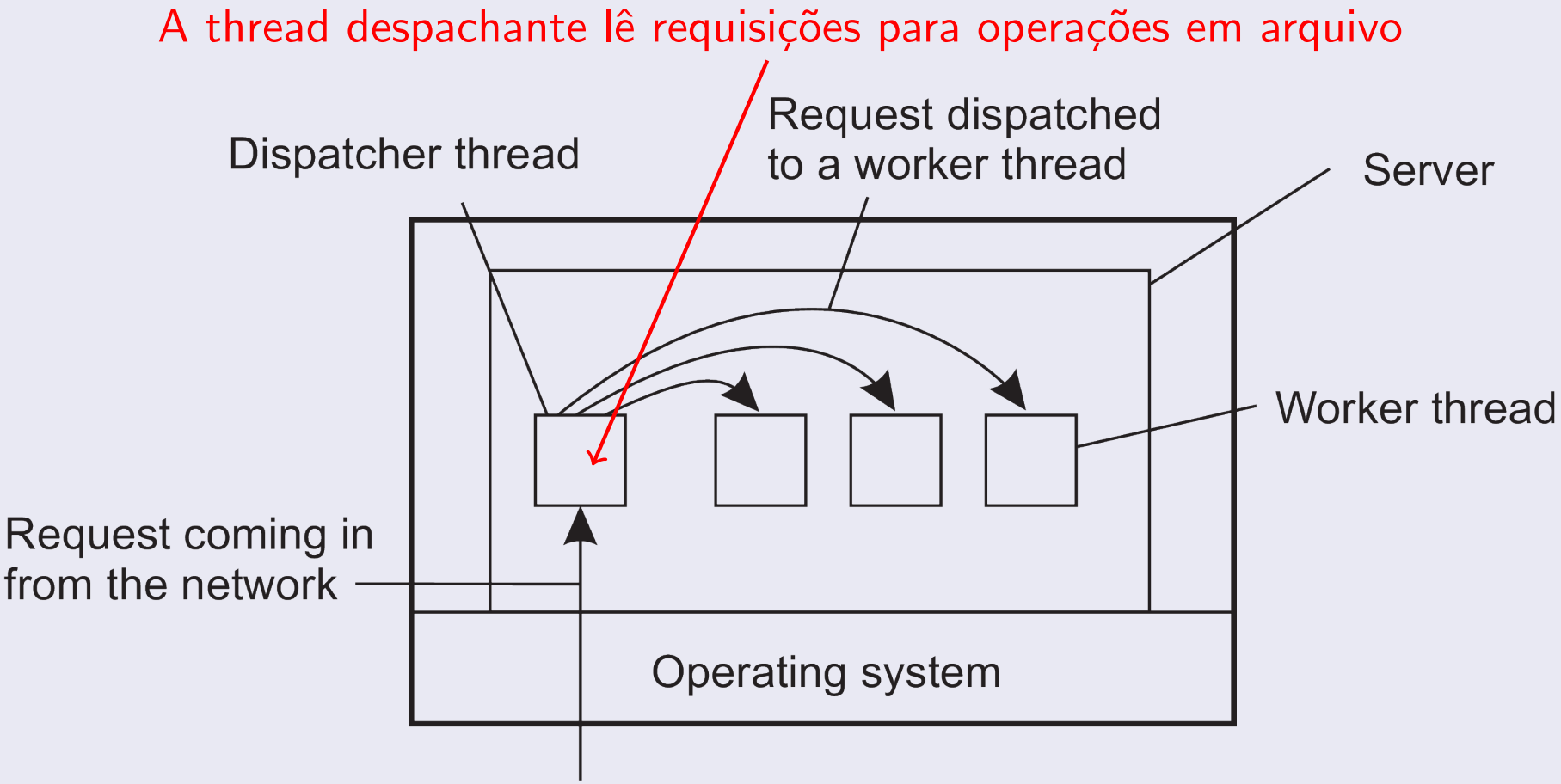
Threads no lado do servidor

Ex: Servidor de arquivos



Threads no lado do servidor

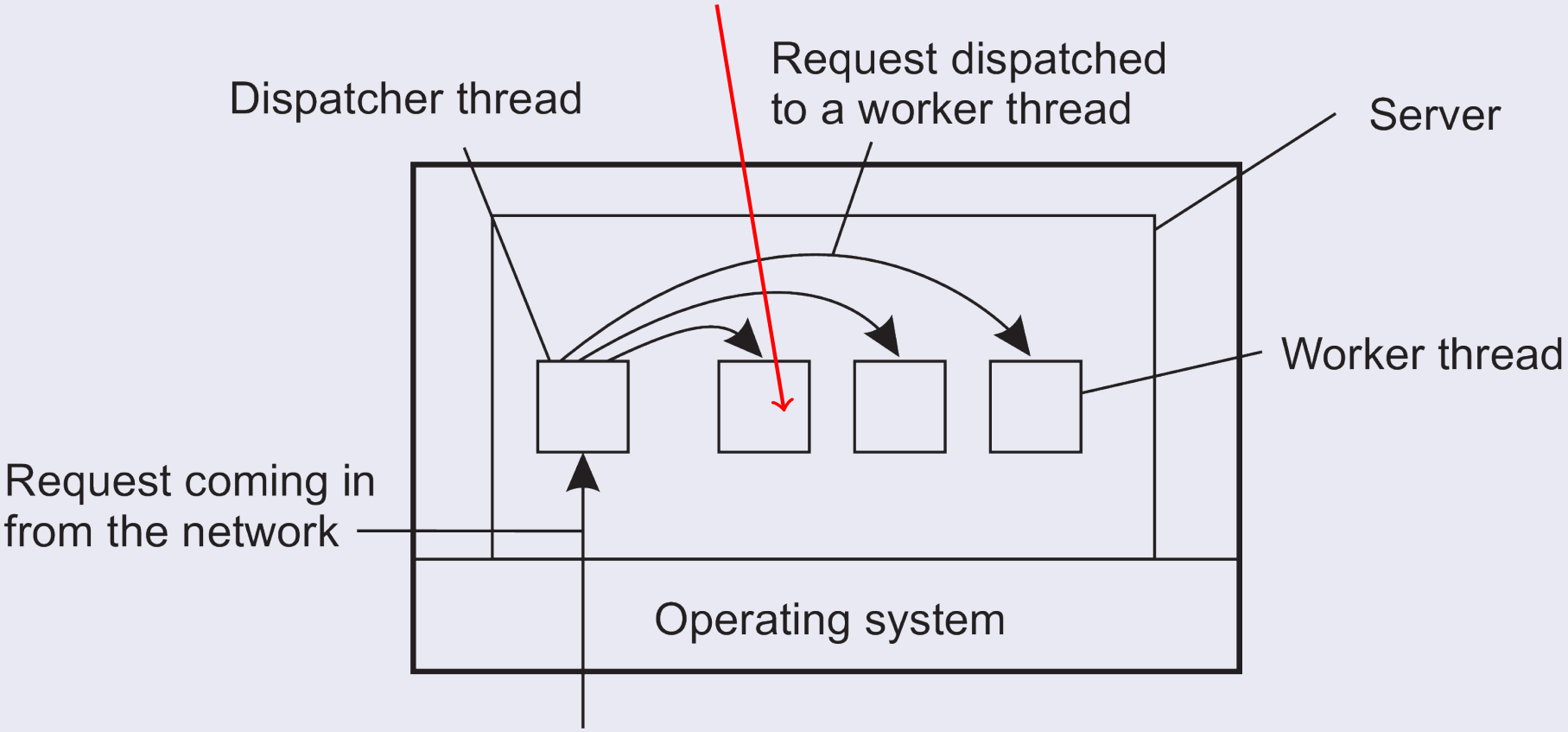
Ex: Servidor de arquivos



Threads no lado do servidor

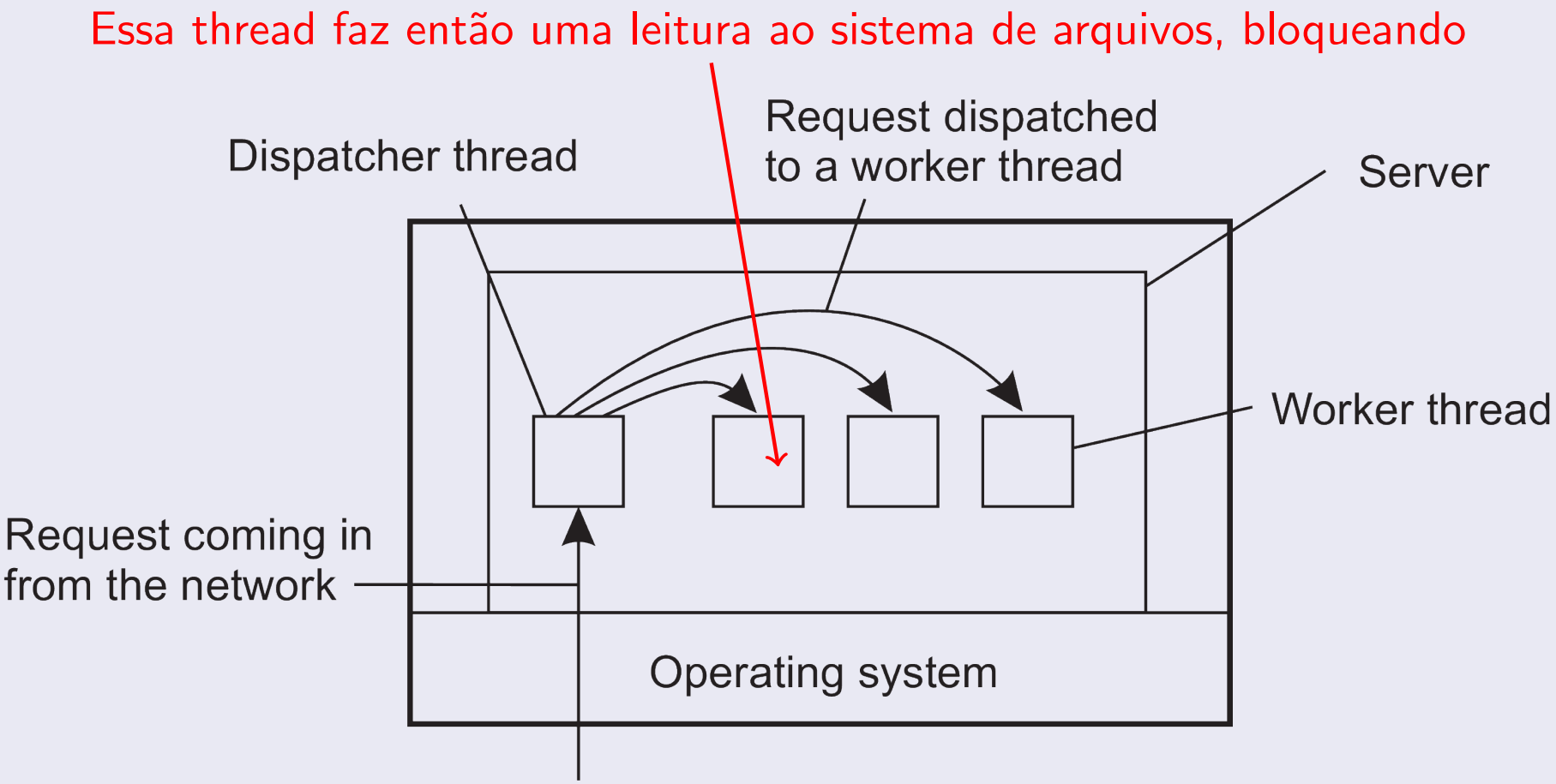
Ex: Servidor de arquivos

Ela então escolhe uma thread operária ociosa e passa a requisição a ela



Threads no lado do servidor

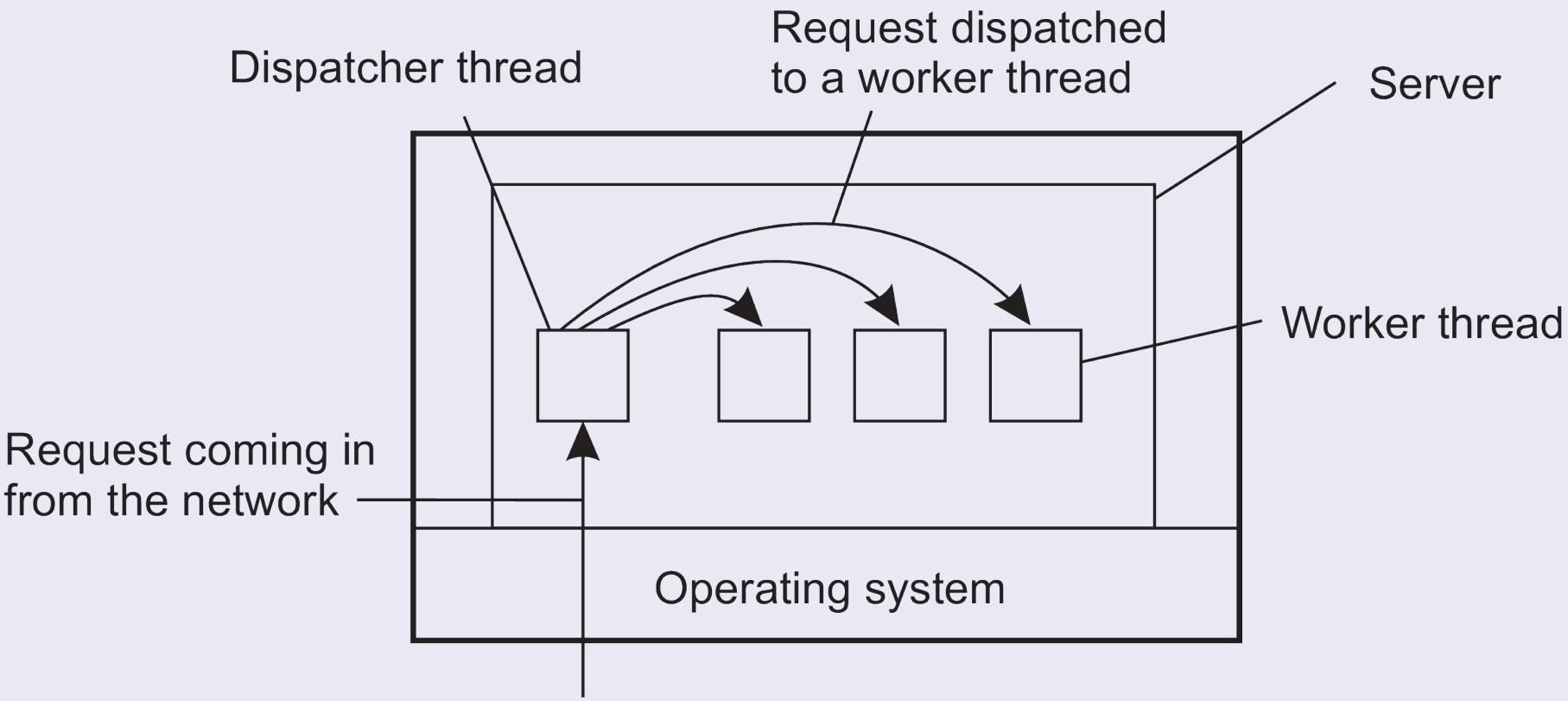
Ex: Servidor de arquivos



Threads no lado do servidor

Ex: Servidor de arquivos

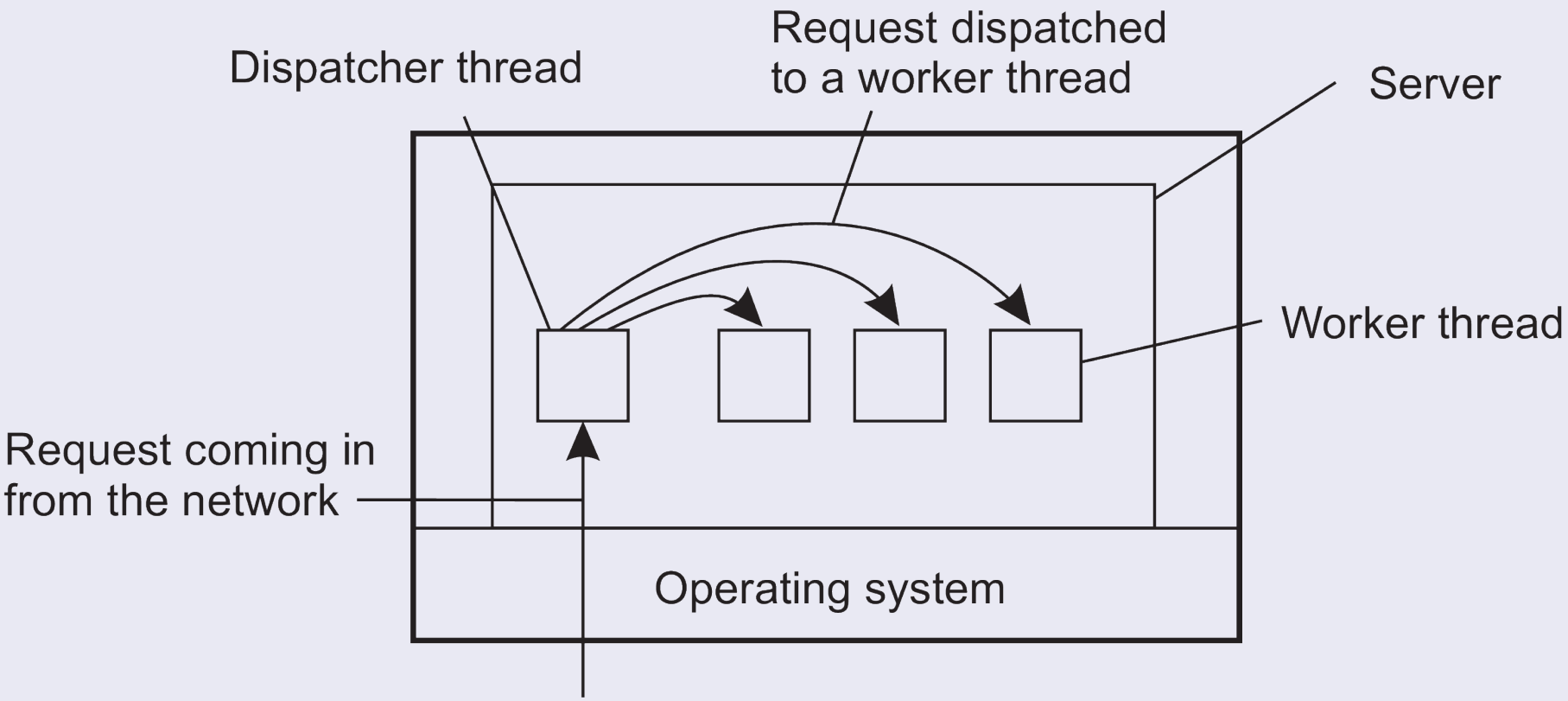
Outra thread pode então ser selecionada para rodar



Threads no lado do servidor

Ex: Servidor de arquivos

Modelo Despachante/operaria (*Dispatcher/worker*)

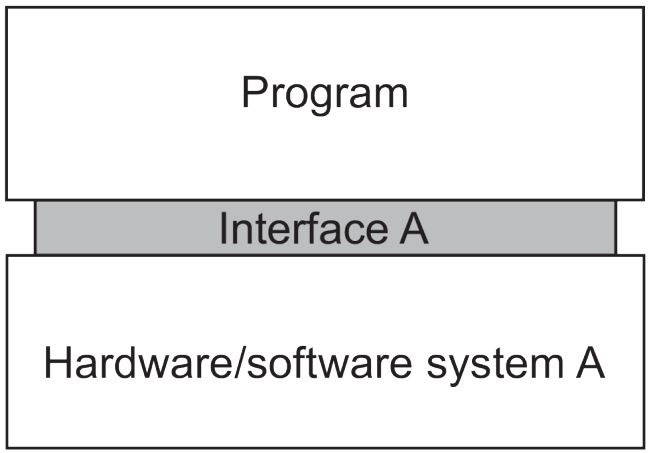


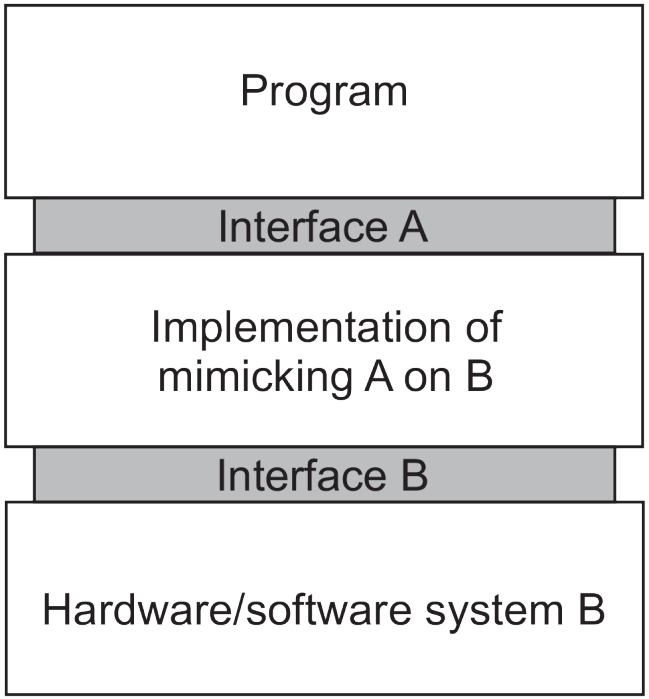
Processos distribuídos

- Revisão de Processos e Threads
- Threads no lado do cliente
- Threads no lado do servidor
- **Virtualização**

Virtualização

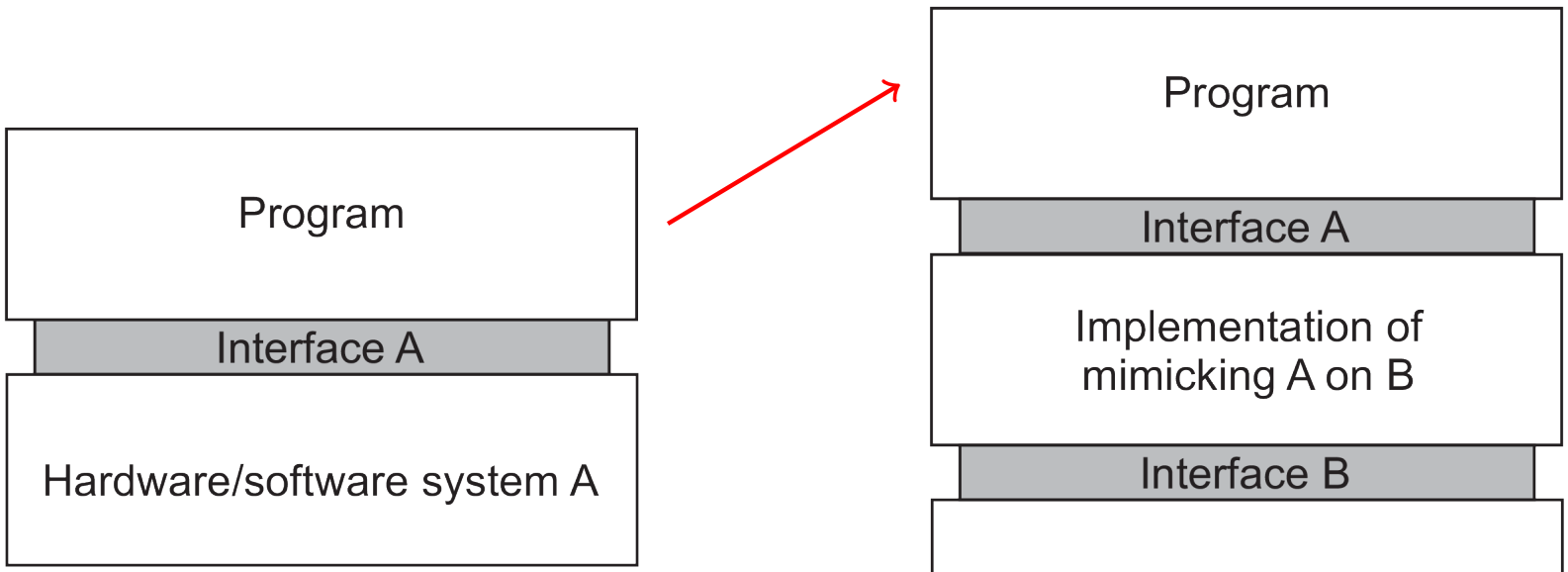
Essencialmente, trata da extensão ou substituição de uma interface existente de modo a imitar o comportamento de outro sistema





Virtualização

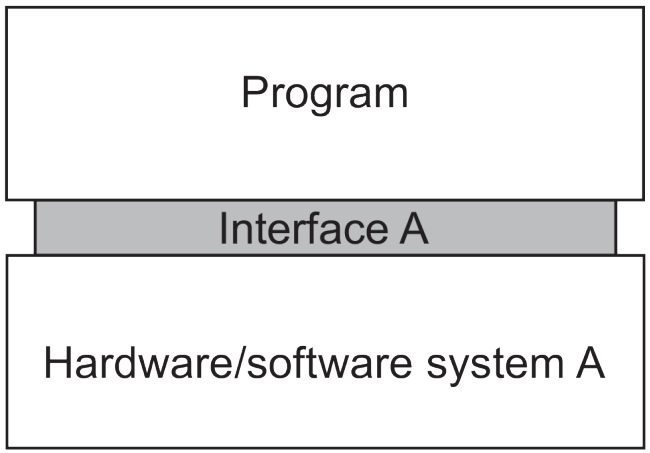
Essencialmente, trata da extensão ou substituição de uma interface existente de modo a imitar o comportamento de outro sistema

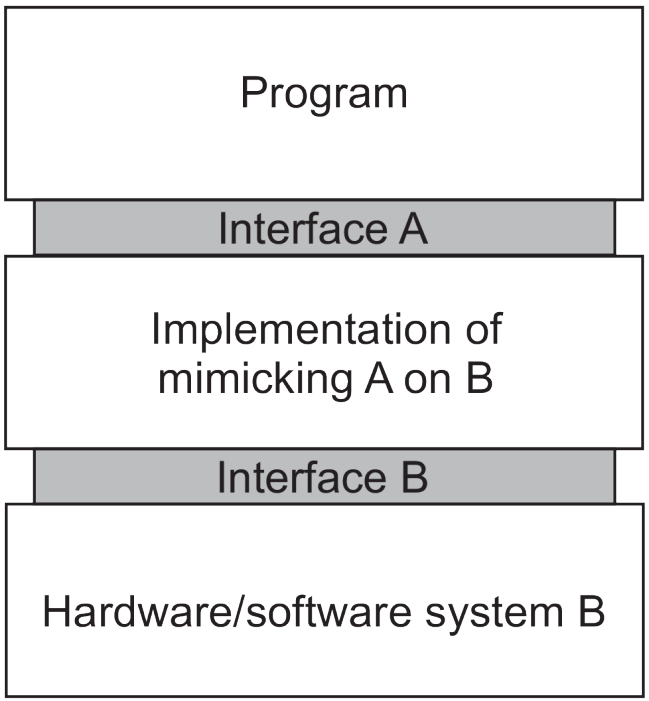


Permite assim a execução de programas de um sistema em outro

Virtualização

Essencialmente, trata da extensão ou substituição de uma interface existente de modo a imitar o comportamento de outro sistema



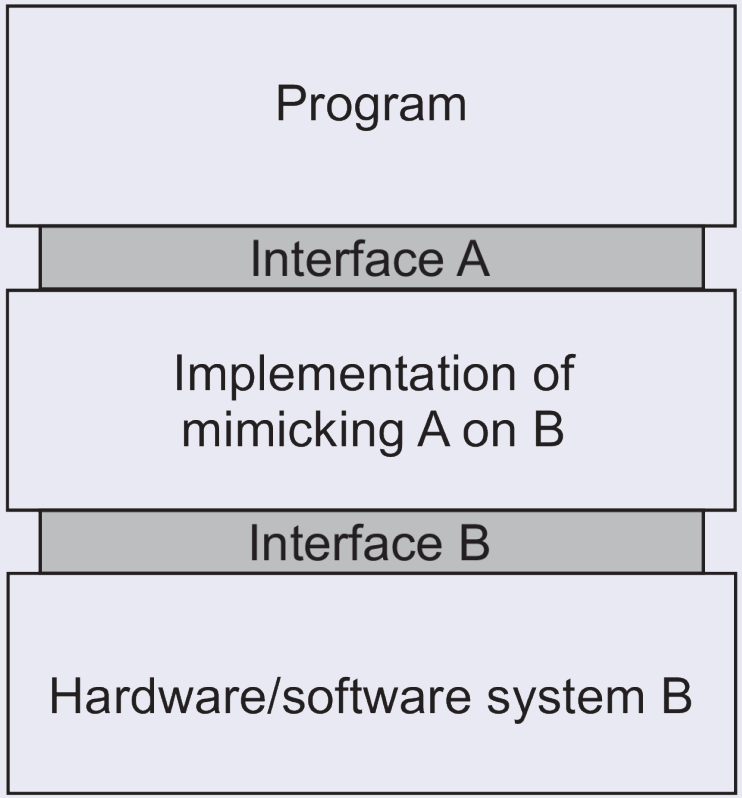


Útil quando portamos interfaces legadas a novas plataformas

Virtualização

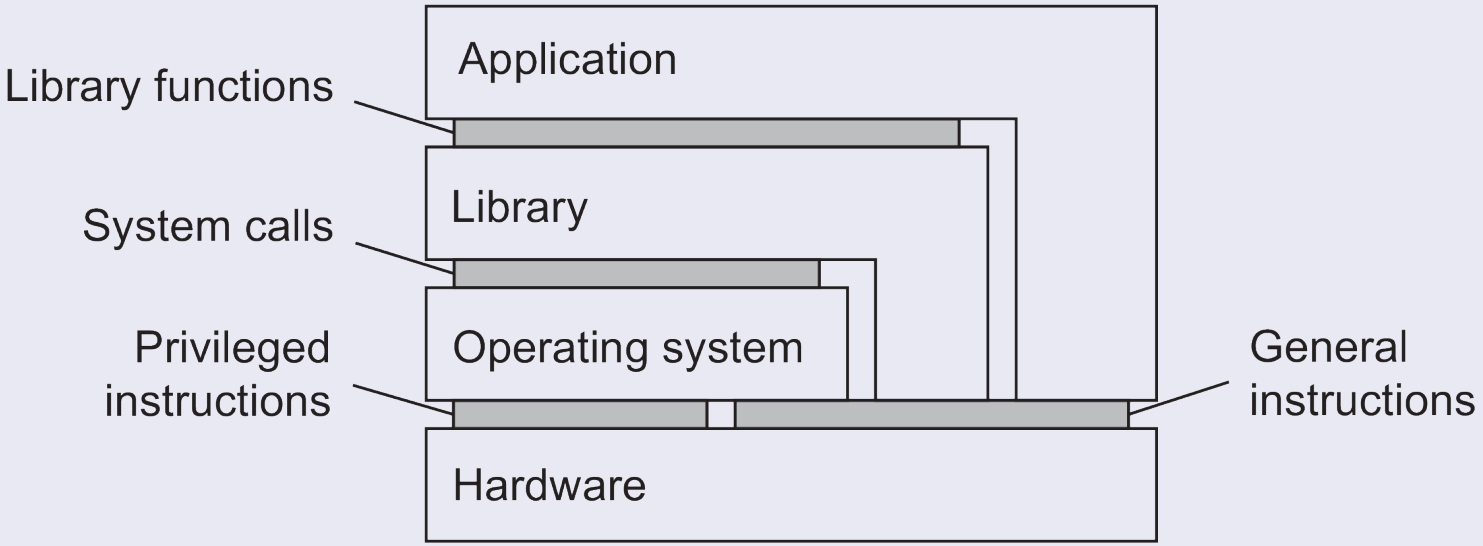
Importância

- Hardware muda mais rápido do que software
- Facilita a portabilidade e a migração de código
- Provê isolamento de componentes com falhas ou sendo atacados



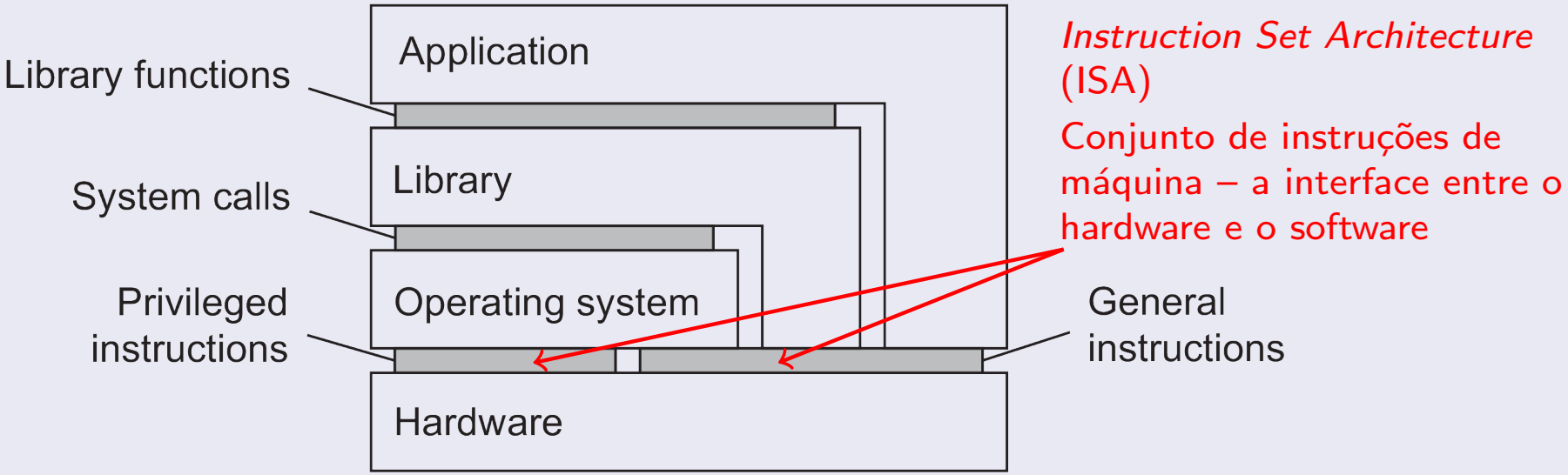
Tipos de Virtualização

- Virtualização pode ocorrer em diferentes níveis, dependendo das interfaces oferecidas pelos diferentes componentes do sistema
  - Em geral, temos 4 tipos de interfaces, em 3 níveis diferentes



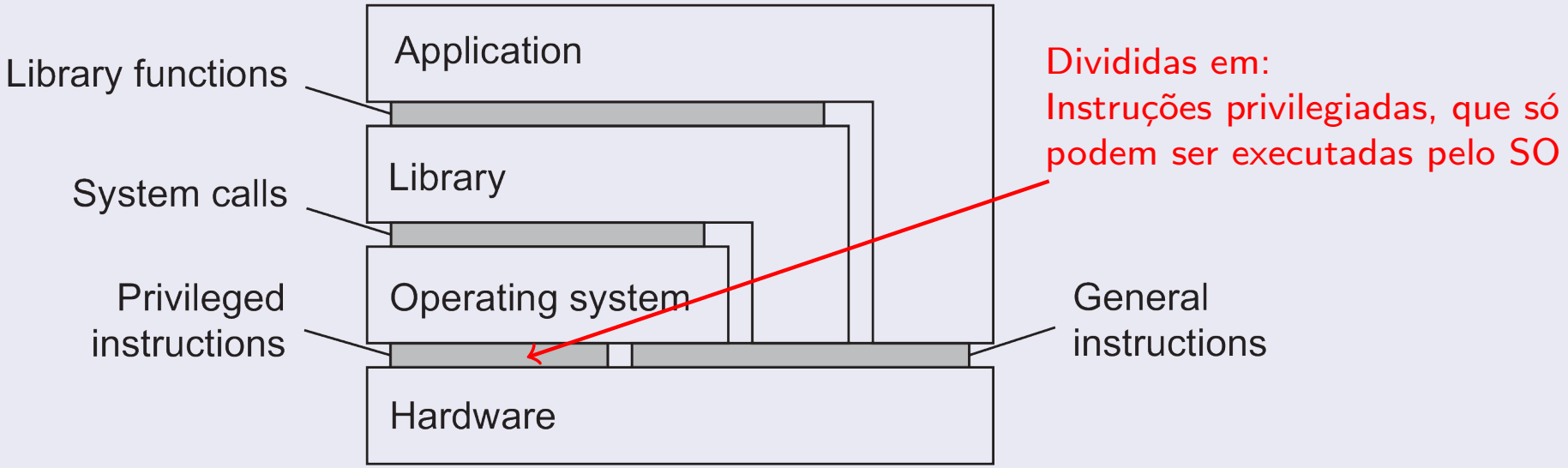
Tipos de Virtualização

- Virtualização pode ocorrer em diferentes níveis, dependendo das interfaces oferecidas pelos diferentes componentes do sistema
  - Em geral, temos 4 tipos de interfaces, em 3 níveis diferentes



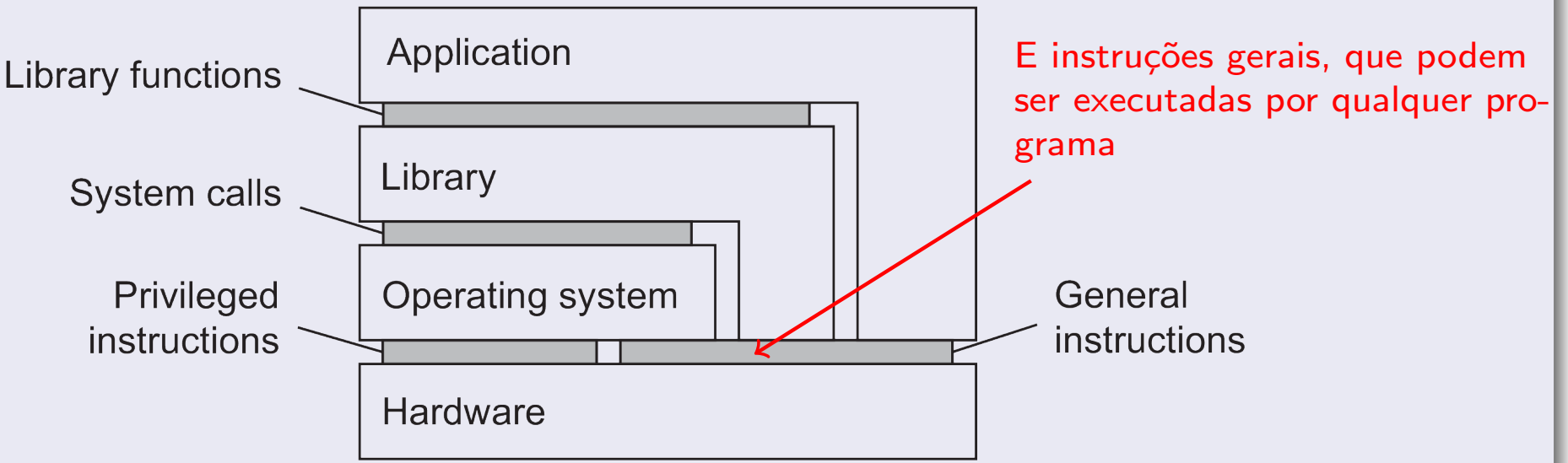
Tipos de Virtualização

- Virtualização pode ocorrer em diferentes níveis, dependendo das interfaces oferecidas pelos diferentes componentes do sistema
  - Em geral, temos 4 tipos de interfaces, em 3 níveis diferentes



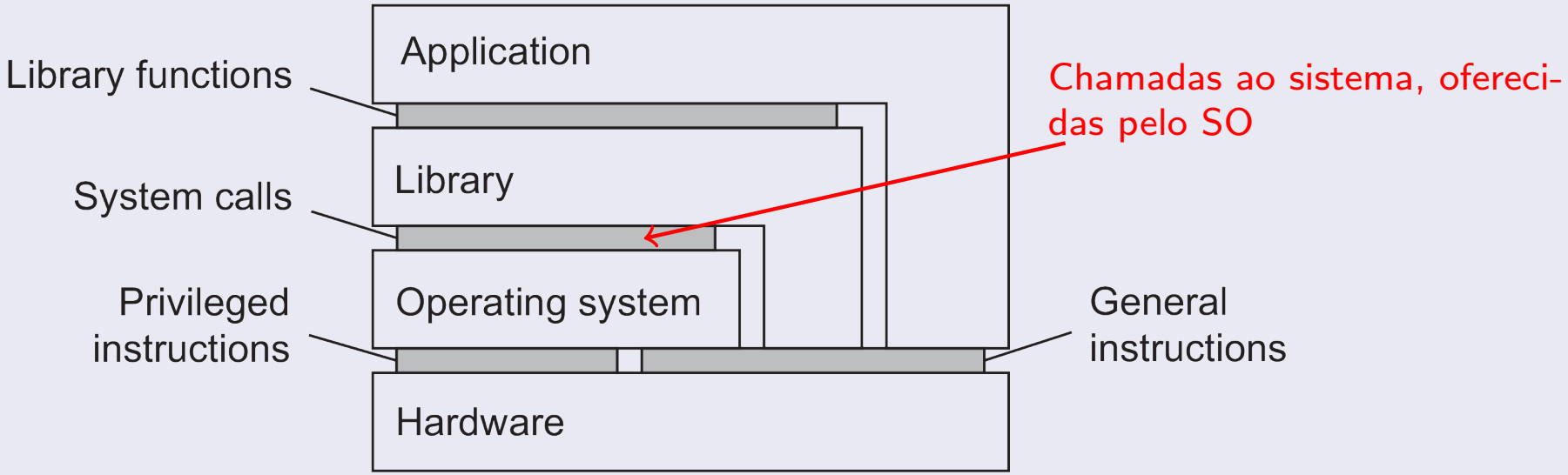
Tipos de Virtualização

- Virtualização pode ocorrer em diferentes níveis, dependendo das interfaces oferecidas pelos diferentes componentes do sistema
  - Em geral, temos 4 tipos de interfaces, em 3 níveis diferentes



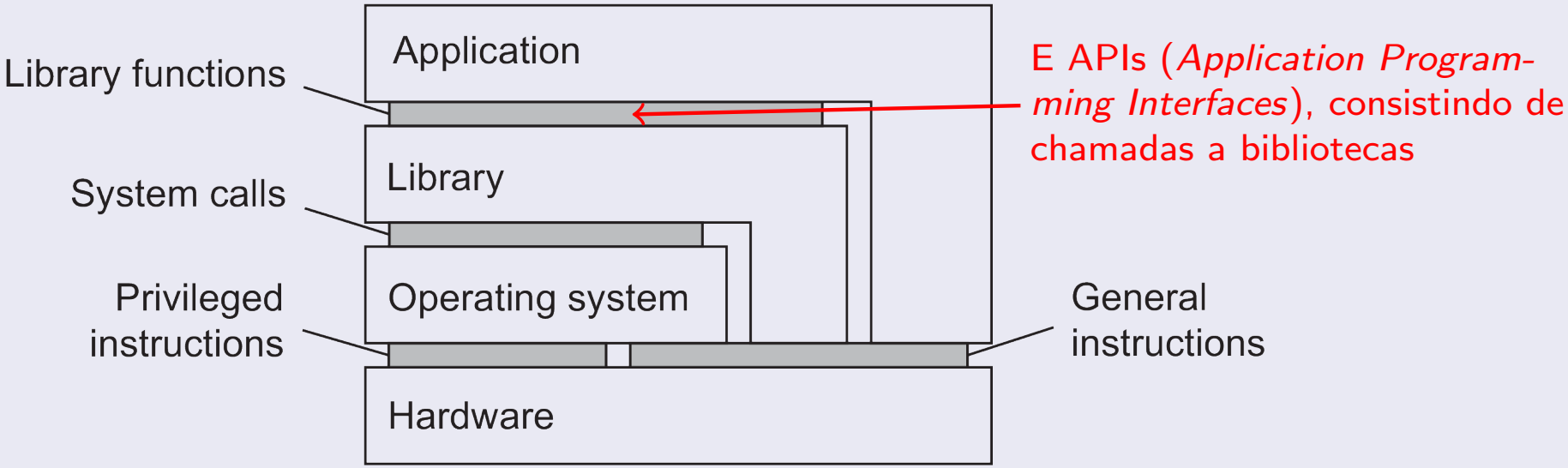
Tipos de Virtualização

- Virtualização pode ocorrer em diferentes níveis, dependendo das interfaces oferecidas pelos diferentes componentes do sistema
  - Em geral, temos 4 tipos de interfaces, em 3 níveis diferentes



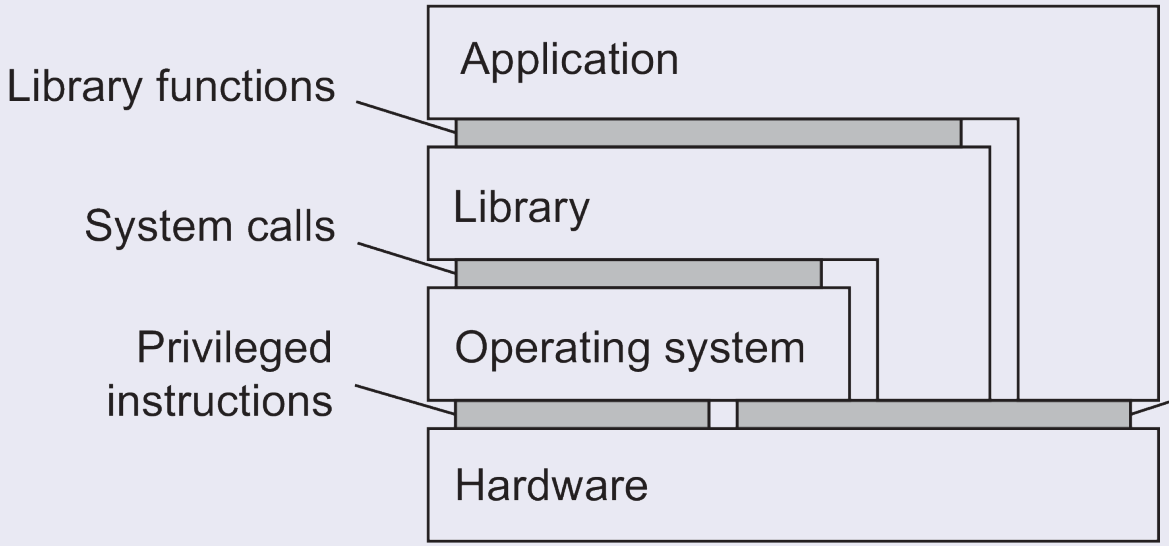
Tipos de Virtualização

- Virtualização pode ocorrer em diferentes níveis, dependendo das interfaces oferecidas pelos diferentes componentes do sistema
  - Em geral, temos 4 tipos de interfaces, em 3 níveis diferentes



Tipos de Virtualização

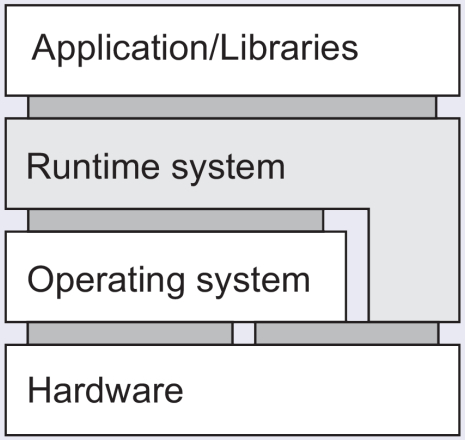
- Virtualização pode ocorrer em diferentes níveis, dependendo das interfaces oferecidas pelos diferentes componentes do sistema
  - Em geral, temos 4 tipos de interfaces, em 3 níveis diferentes

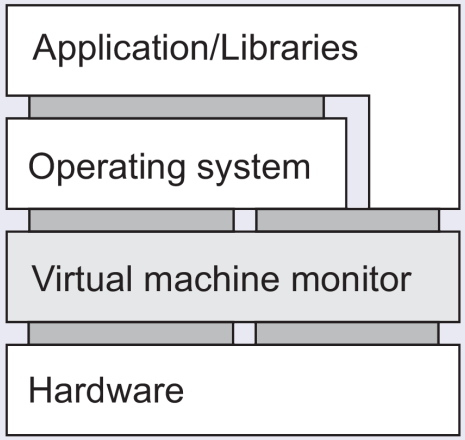


A essência da virtualização é imitar o comportamento dessas interfaces

Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

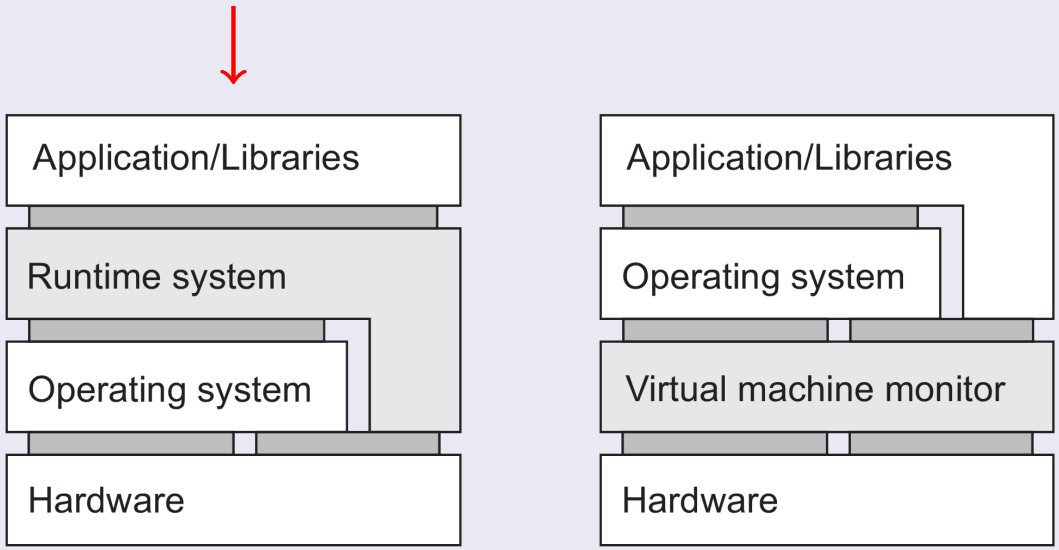




Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

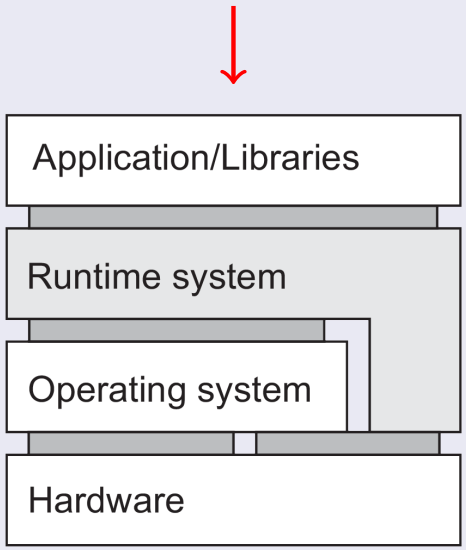
Via um *runtime system*, fornecendo um conjunto abstrato de instruções para as aplicações

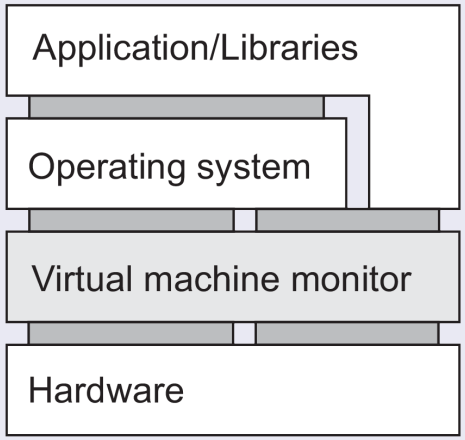


Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

Modelo de **máquina virtual de processo** – a virtualização ocorre para um único processo

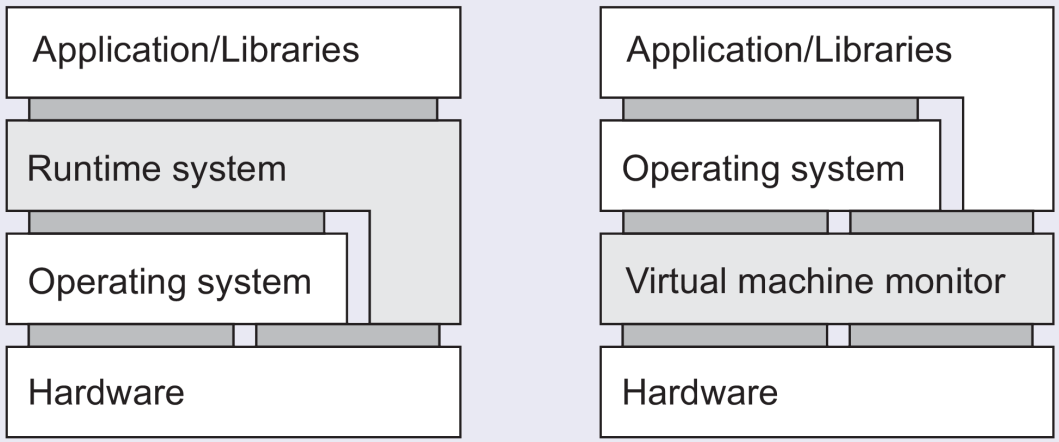




Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

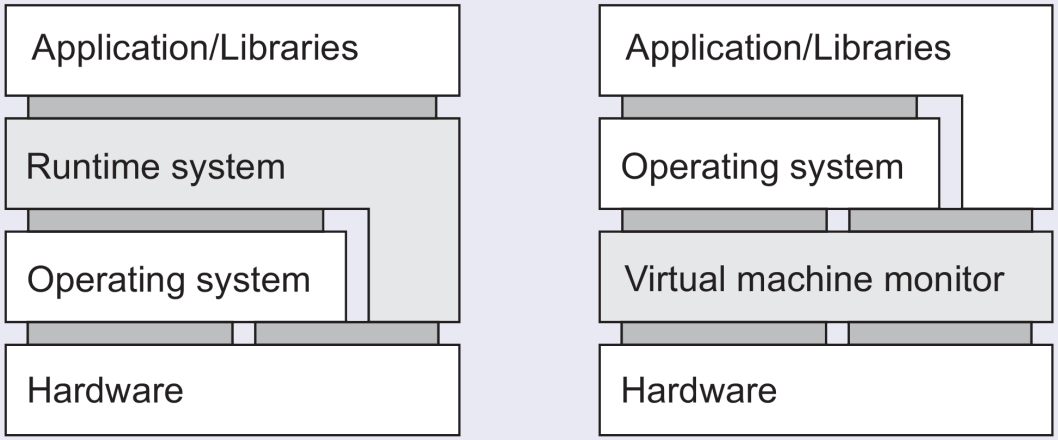
Conjunto separado de instruções e um interpretador/emulador, rodando sobre o SO (Ex: Java VM)



Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

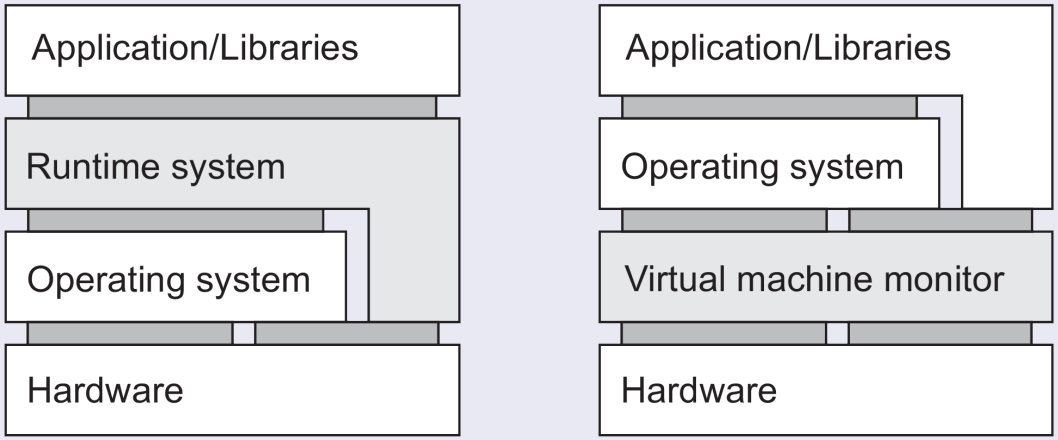
Ou via uma camada acima do hardware, oferecendo um conjunto completo de instruções como interface



Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

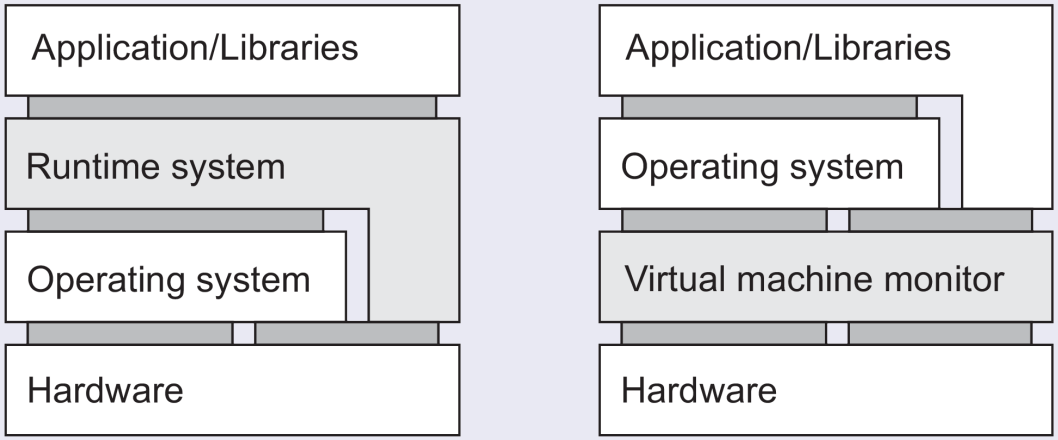
Modelo de **monitor nativo de máquina virtual**, simultaneamente oferecido a diferentes processos



Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

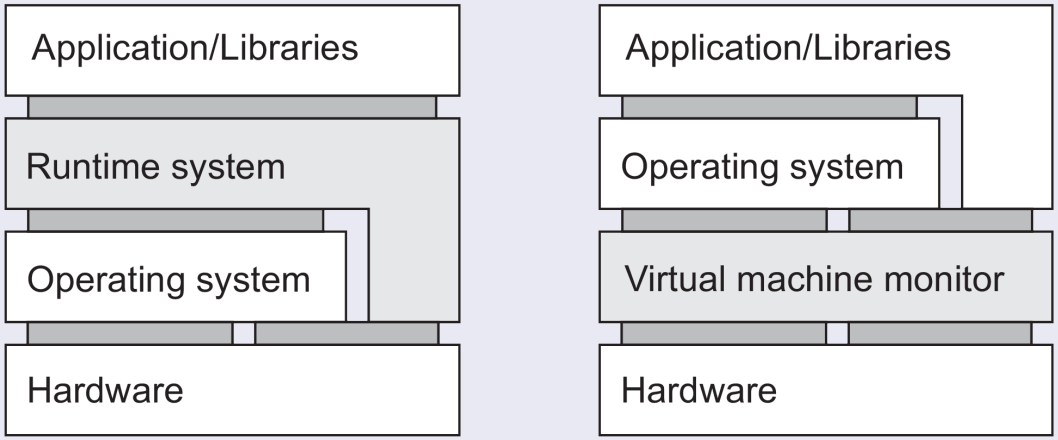
Podendo inclusive executar mútiplos SOs e suas aplicações (ex: VMware)



Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

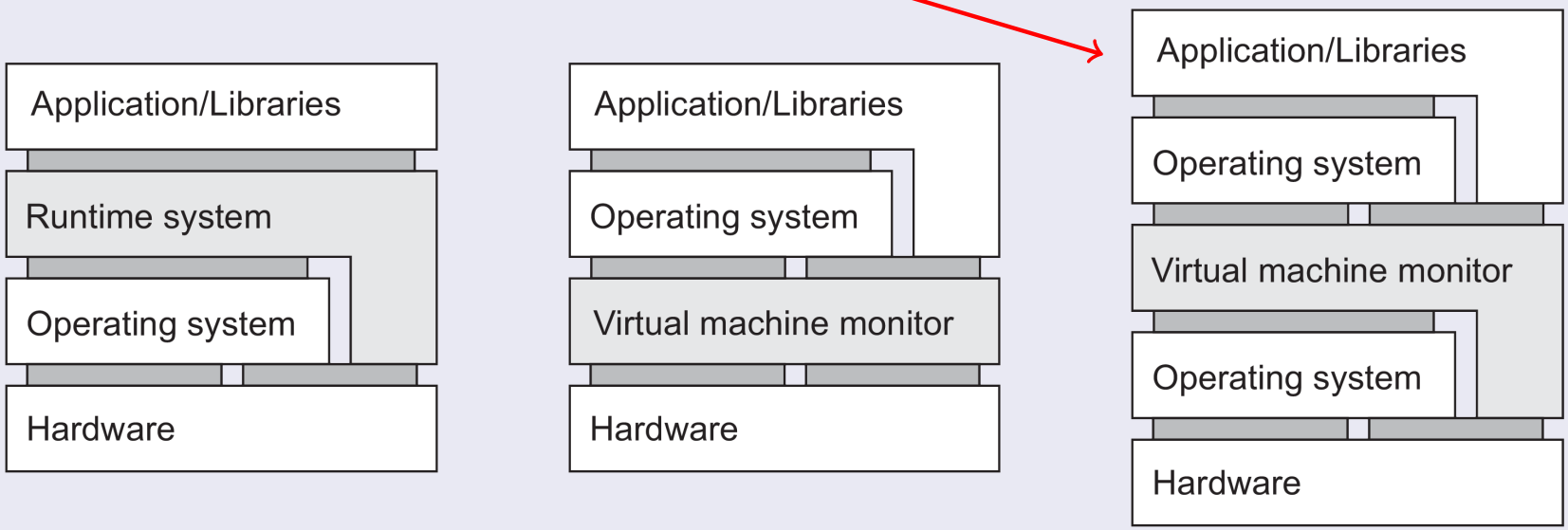
Contendo instruções de baixo nível,
junto com um SO mínimo



Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

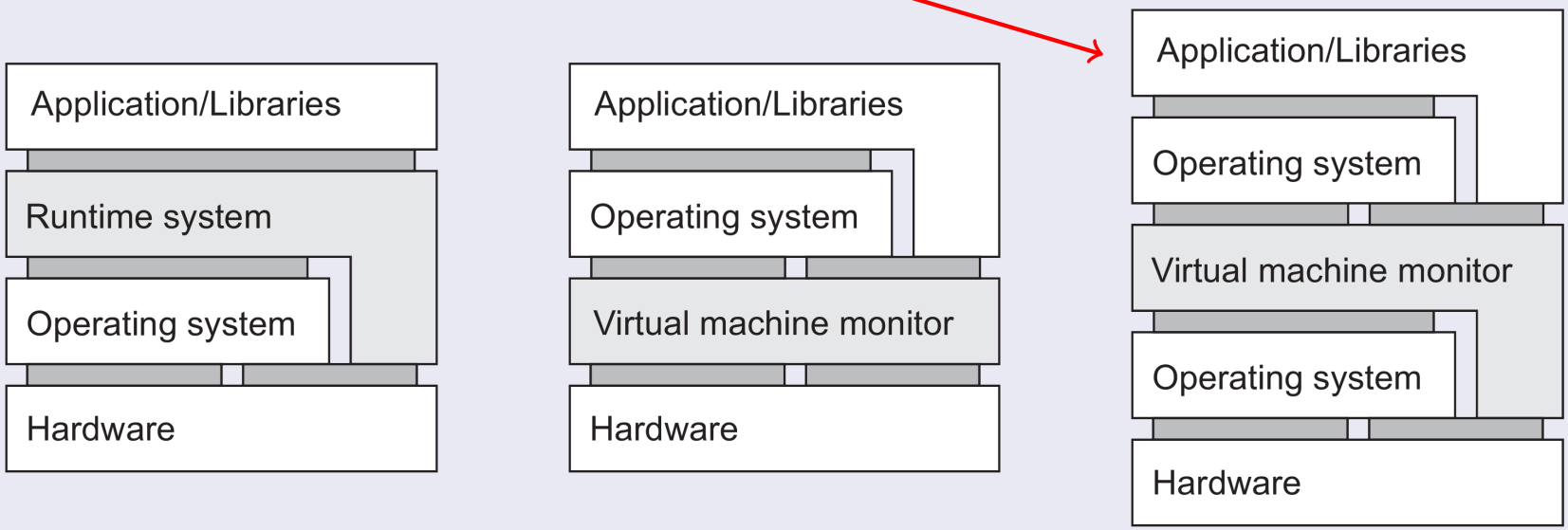
Em vez de implementar um SO, uma alternativa é fornecer um **monitor de máquina virtual hospedado**



Tipos de Virtualização

- Virtualização pode ser feita de 2 maneiras:

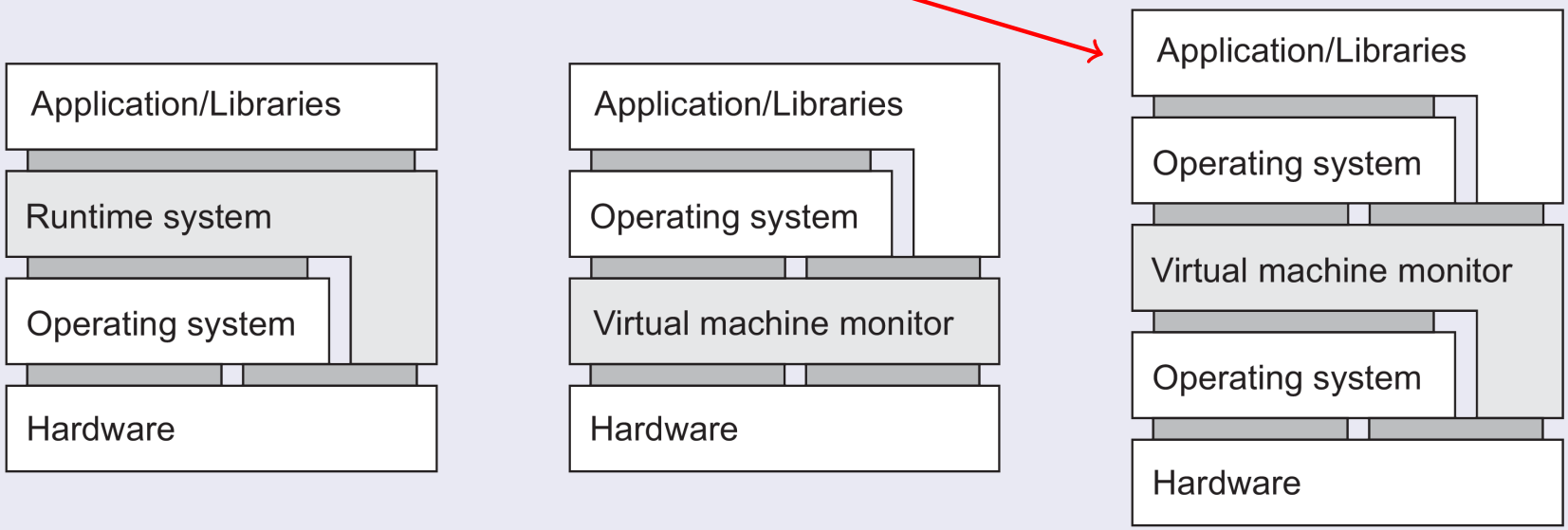
Rodando em cima de um SO anfitrião, contendo instruções de baixo nível, mas delegando a maior parte do trabalho a esse SO



Tipos de Virtualização

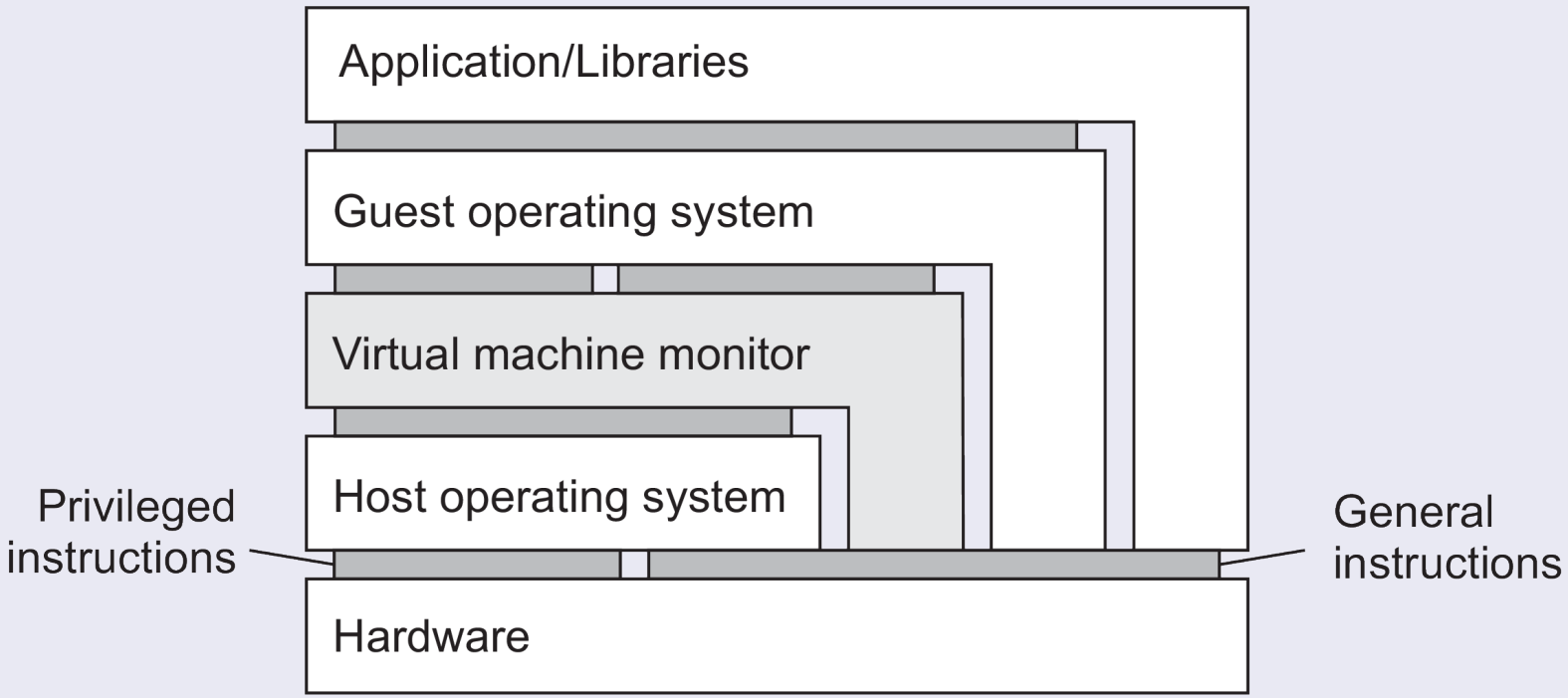
- Virtualização pode ser feita de 2 maneiras:

Modelo bastante popular em sistemas distribuídos modernos



Monitor de MV Hospedado: Desempenho

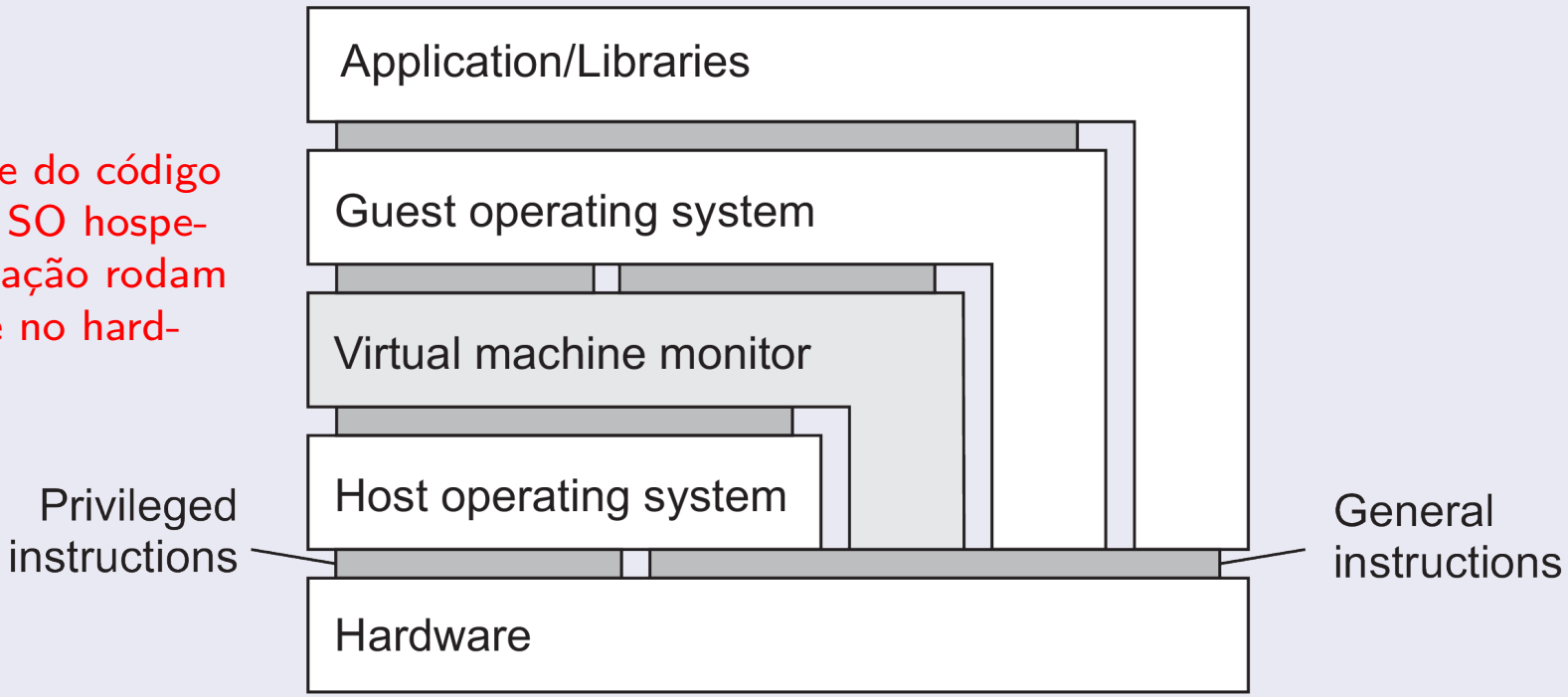
- Refinando a organização



Monitor de MV Hospedado: Desempenho

- Refinando a organização

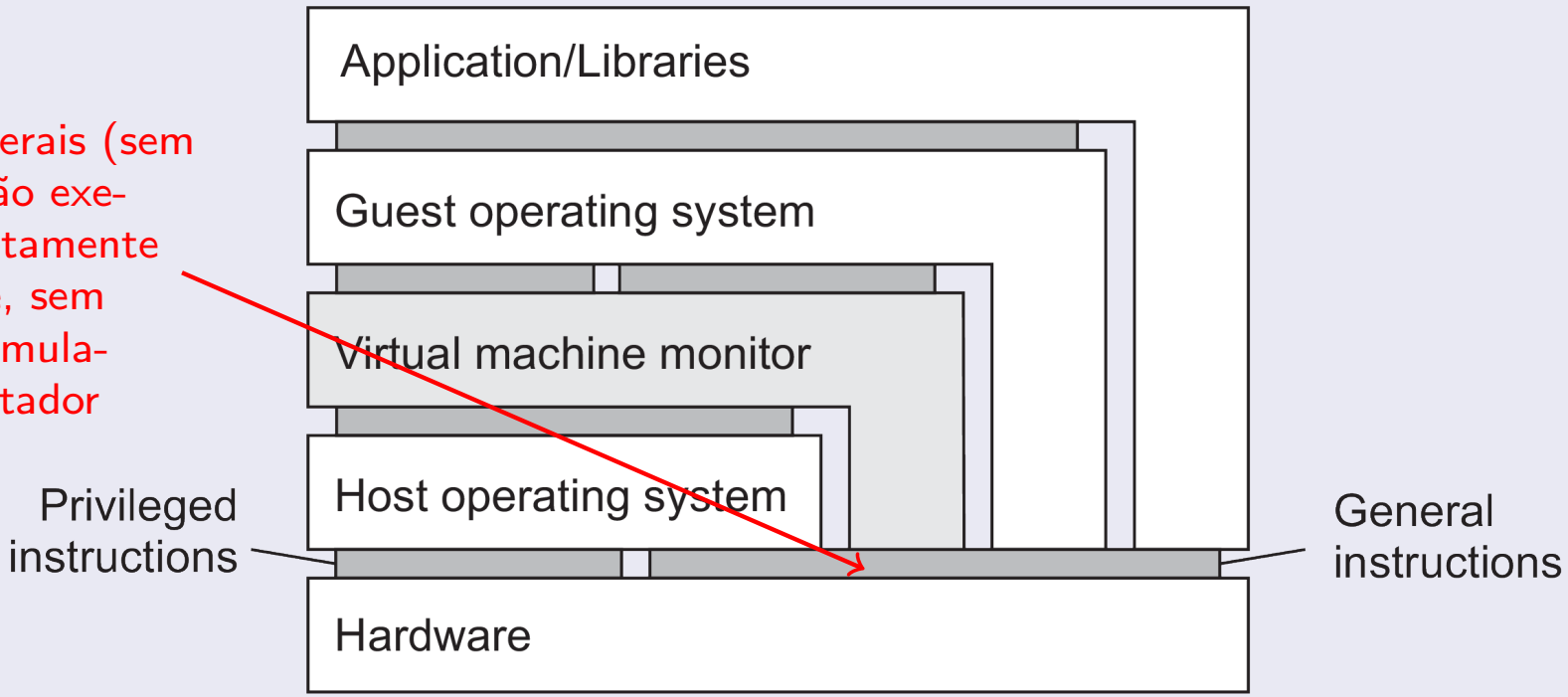
Grande parte do código do monitor, SO hospedado e aplicação rodam nativamente no hardware



Monitor de MV Hospedado: Desempenho

- Refinando a organização

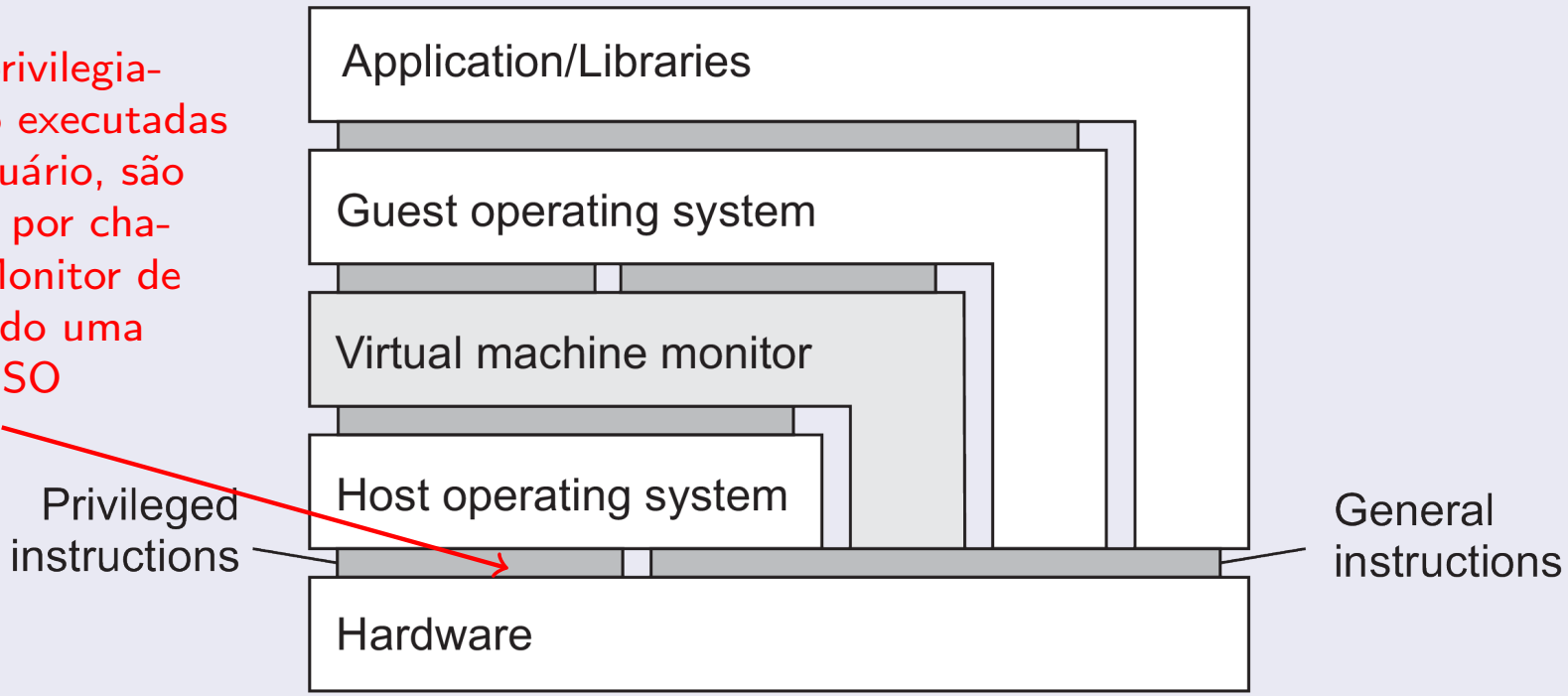
Instruções gerais (sem privilégio) são executadas diretamente no hardware, sem passar por emulador/interpretador



Monitor de MV Hospedado: Desempenho

- Refinando a organização

Instruções privilegiadas, quando executadas no modo usuário, são substituídas por chamadas ao Monitor de VM, causando uma *trap* para o SO



Virtualização

Aplicações de MVs a Sistemas Distribuídos

- Computação de nuvem, oferecendo 3 serviços:

Aplicações de MVs a Sistemas Distribuídos

- Computação de nuvem, oferecendo 3 serviços:
  - Infrastructure-as-a-Service: estrutura básica
    - Em vez de alugar uma máquina física, o provedor do serviço aluga uma máquina virtual (um monitor)
    - Permite um isolamento quase completo entre clientes
    - Embora o compartilhamento de recursos acabe por reduzir o desempenho, comparando-se a uma máquina isolada

Virtualização

Aplicações de MVs a Sistemas Distribuídos

- Computação de nuvem, oferecendo 3 serviços:
  - Infrastructure-as-a-Service: estrutura básica
    - Em vez de alugar uma máquina física, o provedor do serviço aluga uma máquina virtual (um monitor)
    - Permite um isolamento quase completo entre clientes
    - Embora o compartilhamento de recursos acabe por reduzir o desempenho, comparando-se a uma máquina isolada
  - Platform-as-a-Service: serviços no nível do sistema

Virtualização

Aplicações de MVs a Sistemas Distribuídos

- Computação de nuvem, oferecendo 3 serviços:
  - Infrastructure-as-a-Service: estrutura básica
    - Em vez de alugar uma máquina física, o provedor do serviço aluga uma máquina virtual (um monitor)
    - Permite um isolamento quase completo entre clientes
    - Embora o compartilhamento de recursos acabe por reduzir o desempenho, comparando-se a uma máquina isolada
  - Platform-as-a-Service: serviços no nível do sistema
  - Software-as-a-Service: aplicações