



Introdução/Arquitecturas

Computação Distribuída 2022/23

dgomes@ua.pt

Processos



*“Sometimes I think the collaborative process
would work better without you.”*

Processos vs Threads

- Criados pelo SO
- Podem ter múltiplas threads
- Maior overhead – levam mais tempo a abrir e fechar *→ Se tiver um processo por cliente estou a atrair TUDO! //*
- Partilha de informação entre processos é lenta pois não partilham memória
- São mini-processos (dentro de um processo)
- Partilham memória pelo que leem e escrevem eficientemente as mesma variáveis
- Em Python não trazem benefício devido ao GIL (Global Interpreter Lock)

⊕ Simples

Python examples

```
import threading

def count(prefix, n):
    for x in range(n):
        print(prefix, x)

if __name__ == "__main__":
    thread1 = threading.Thread(target=count, args=("a", 100))
    thread2 = threading.Thread(target=count, args=("b", 100))

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()
```

↑
Threads pode ser bom
para Chat-Servers
Priorizar Comunicação!
Mas tem Over-Head

```
import multiprocessing

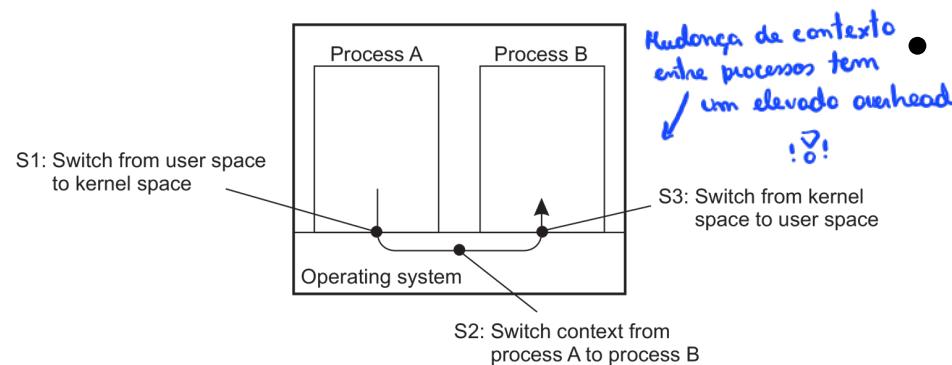
def count(prefix, n):
    for x in range(n):
        print(prefix, x)

if __name__ == "__main__":
    p1 = multiprocessing.Process(target=count, args=("a", 1000))
    p2 = multiprocessing.Process(target=count, args=("b", 1000))

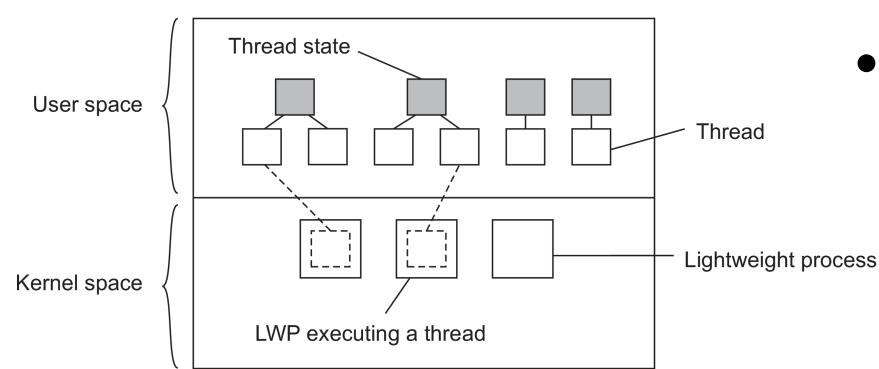
    p1.start()
    p2.start()

    p1.join()
    p2.join()
```

Threads em sistemas não distribuídos



- Inter-process Communication (IPC)



- Implementação

- Lightweight Processes (LWP)
- Scheduler activations

O SG ativa sempre com eventos de I/O

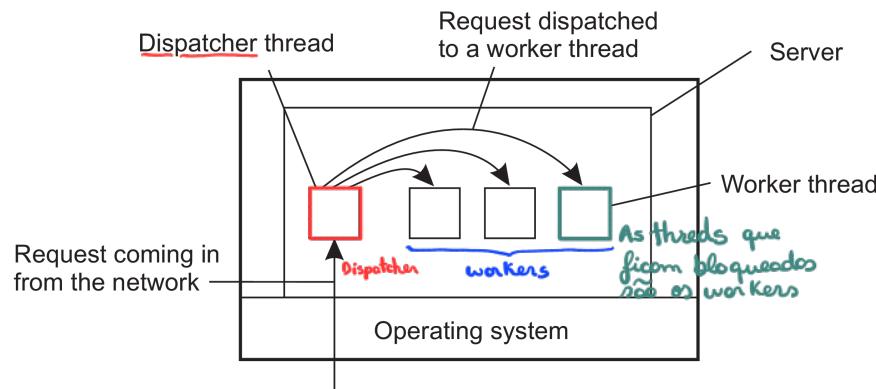
⇒ Evento ativo com uma mensagem na rede (Select)
↳ Biblioteca: "Selectors" } Projeto com Selectors e LWP !?

Threads em sistemas distribuídos

Multithreads Servers:

→ e.g., occasionally has to block waiting for the disk

⇒ clients not wait for the request...



- Servidor *multithread* organizado em modelo *dispatcher/worker*
 - Recebe pedidos e distribui
 - Processa os pedidos

Modelo	Características
Threads ↳ Thread por cliente ↳ Se tiver de montar estados...	Paralelismo, Chamadas ao sistema bloqueiam → Pode ficar bloqueada!,,
Processo Single-thread ↳ Isolados	Sem paralelismo, Chamadas ao sistema bloqueiam
Maquina de estados finitos LWP	Paralelismo, Chamadas ao sistema não bloqueiam → <u>Não PARA!</u>

↑ Em python ele bloqueia quando escrevemos no teclado !!!

Virtualização

- Eficiência
- Partilha de Recursos → 3 conceitos que todos os virtualizações tem de ter em mente ...
- Equivalência

Conceito muito abstracto...

Memória Virtual, Filesystem Virtual, Redes Virtuais (VPN's)

Ficheiro partilhado remotamente
Processos têm "acesso" a toda a RAM

Interface de Rede Virtual em cima da mesma
Overlay

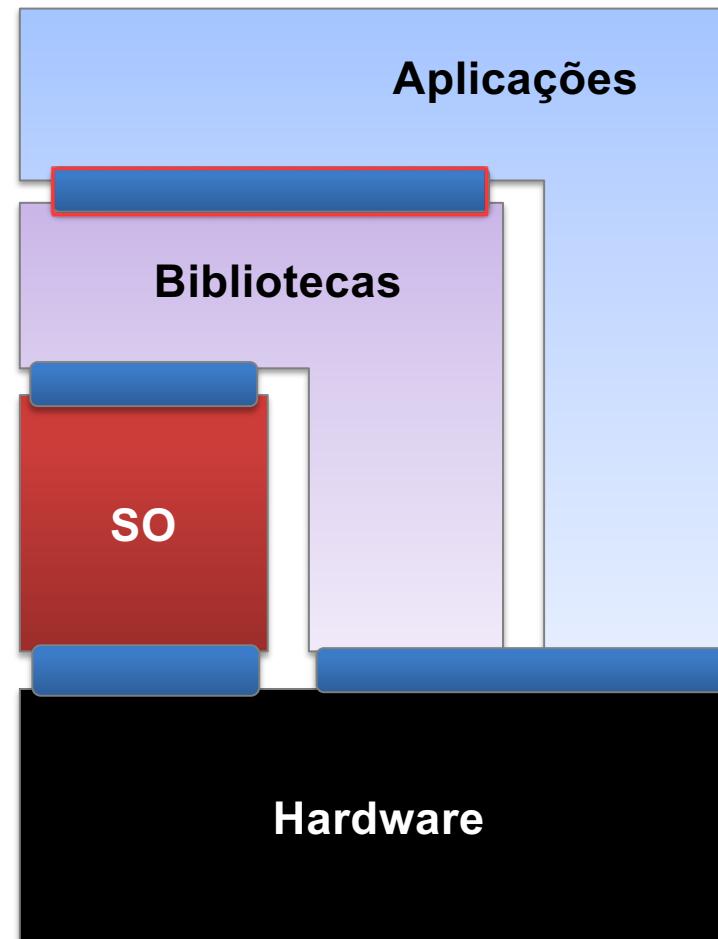
Virtualização permite que um único computador possa desempenhar o papel de vários computadores, através da partilha de recursos de hardware por vários ambientes.

e.g.: O virtual Box está entre os 2 SO, é um middleware!
→ uma abstração para o SO de cima a pensar que é um PC, simula um hardware

Virtualização é a técnica que permite correr vários sistemas distintos no mesmo hardware, de forma completamente separada.

Virtualização

- Virtualização pode ser conseguida a vários níveis:
- Chamadas bibliotecas (^{JVM})
 - Mesmo SO/HW, bibliotecas individuais
- Chamadas ao sistema (^{Memória Virtual})
 - Mesmo HW, Visão do SO individual
- Chamadas ao Hardware (^{Virtual Box})
 - Hardware individual



Virtualização

- Citação mais popular:
 - “Formal Requirements for Virtualizable Third Generation Architectures” Gerald J. and Robert P. Goldberg, Communications of the ACM, Volume 17, Issue 7, Julho 1974
 - “VMM satisfies efficiency, resource control, equivalence” → Nós vamos utilizar este! → Definição muito IMPORTANTE
 - “Theorem – For any conventional third generation machine, a VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.” Von Neuman Machine



Ano 2000: Os computadores TODOS tiveram de ser reescritos para passar de 2 dígitos para 4 dígitos

↑
e.g.

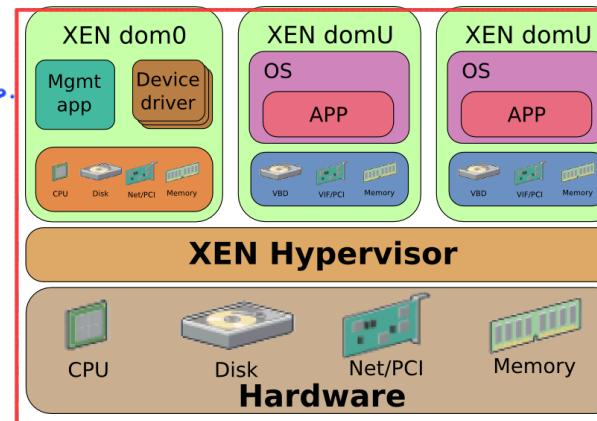
Virtualização – História



- **Mainframe IBM S/360**
→ Ainda existe!
Permite o timeshare de aplicações através do particionamento de recursos (1960/70)
Introdução do conceito de LPAR (Logical resource Partitioning) e mais tarde DLPAR (LPAR dinâmico)
Multiplos SO
↑ Partilhando o Recurso Físico!

• VMWare (1998)

Fundado em Stanford
Parte da EMC/Dell → Broad.com
Disponibiliza hypervisors que permitem isolamento completo
→ Na mesma máquina vários SO a partilhar hardware
→ Isolamento TOTAL! → ↗ Segurança



- **Bochs** Ambiente emulado... (criar software em hardware em desenvolvimento)
Disponibiliza ambientes emulados (1990)
Principalmente para debugging
Ambiente hardware completo por emulação

→ e.g.: Jogos de PS1 a correr num computador
↓
Ambientes de emulação
↑
Util para programar Micro-controladoras sem o ter...
Mais é um emulador ⇒
→ não é eficiente
→ não partilha recursos

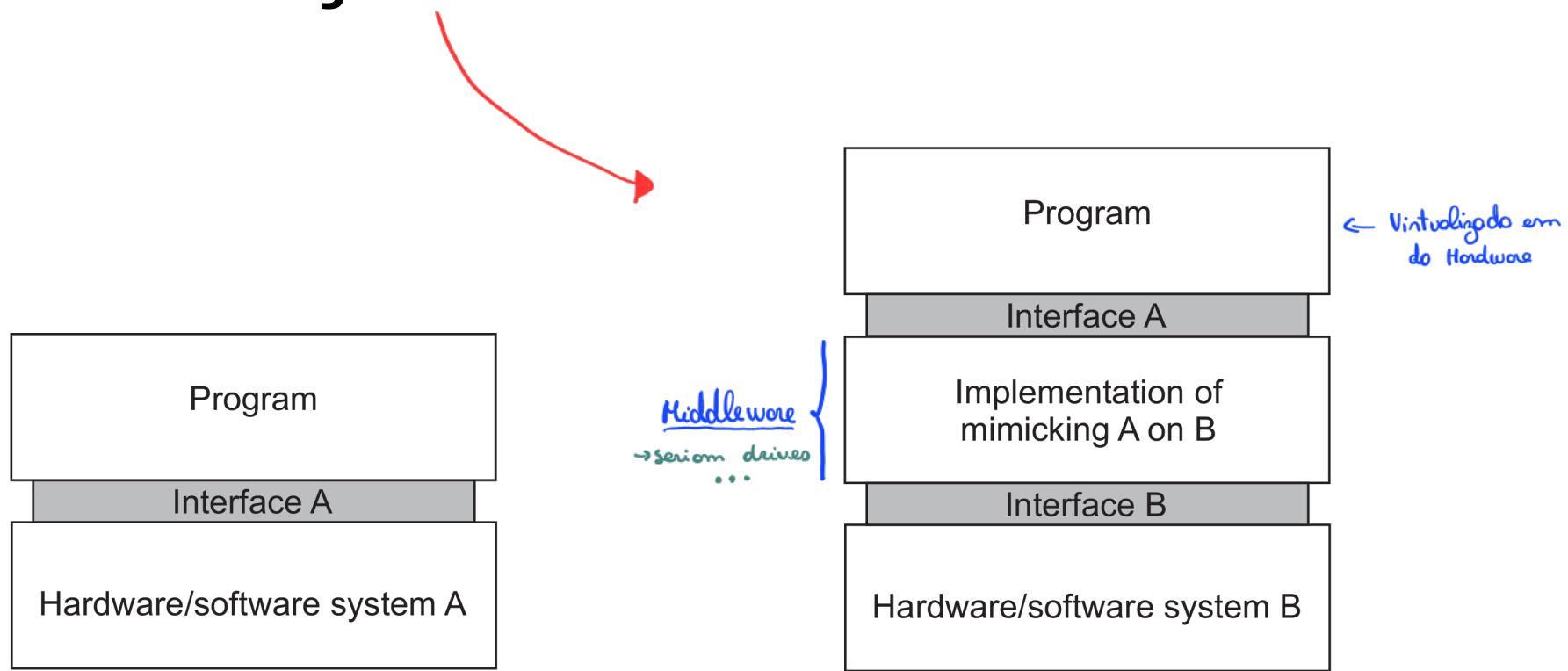
• XEN (2003)

Fundado em Cambridge
Propõe para-virtualização (host assisted) ⇒ Para quem quer Eficiência!
→ State of the Art!

→ Para-virtualização ⇒ hospedamos num dos SO

↗ Eficiência (Diversos sabem que estão a trabalhar na Virtual Box
⇒ Segurança
→ Aumenta Eficiência)

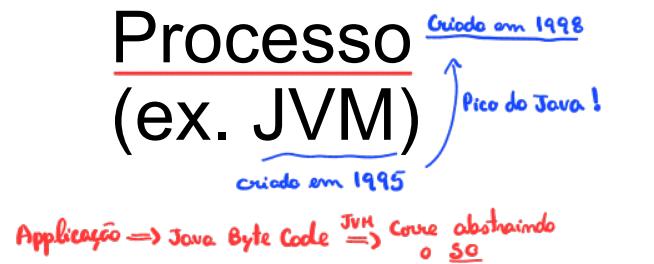
Virtualização



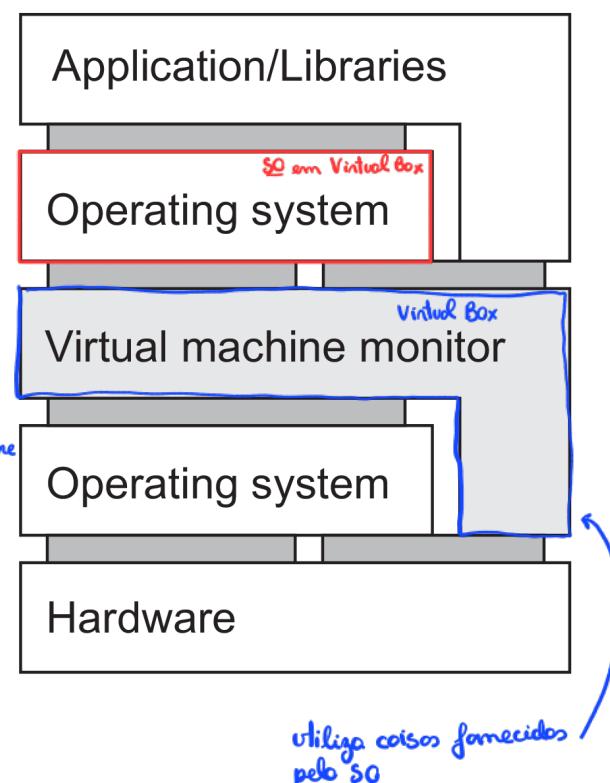
- Programa, Interface e Sistema
- Sistema virtualizado A em cima do sistema B

Arquitecturas de VM's

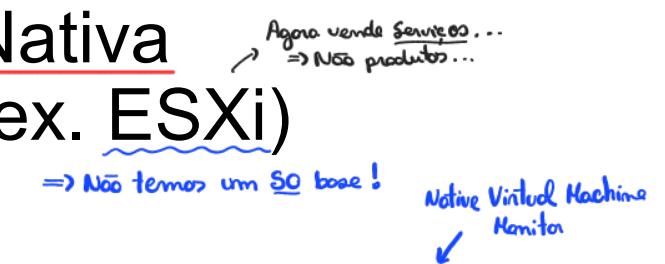
- Virtual Machine
Processo
(ex. JVM)



- Virtual Machine
Hosted
(ex. VirtualBox)



- Virtual Machine
Nativa
(ex. ESXi)



Para servidores não preciso de SO na base... Quero só um SO virtual

→ Queremos criar uma Aplicação (técnicos de Virtualização) que cobra em todo o Lado!

Eco-sistema

Containers

- coloca o código em "imagens"
- cria a base dependendo do SO que o cliente usa...

• Em Sistemas Distribuídos também precisamos de virtualização:
 → Confiabilidade
 → Segurança
 permitem um isolamento, ou seja, um erro ou uma falha não é propagada

⇒ Escalher conforme o Custo ...

System Virtualization

Hardware Level

Bare-Metal/ Hypervisor

- HP Integrity VM
- IBM zSeries z/VM
- VMware ESX Server
- Xen

Hosted

- Microsoft Virtual Server
- Microsoft Virtual PC
- Parallels Desktop
- VMware Player
- VMware Workstation
- VMware Server

High-Level Language

- Java
- Microsoft .NET / Mono
- Smalltalk

OS Level

- FreeBSD Jail
- HP Secure Resource Partitions
- Sun Solaris Zones
- SWsoft Virtuozzo
- User-Mode Linux

Para-virtualization

- Virtual Iron → Microsoft
- VMware VMI
- Xen

Docker: → Gestão de containers

→ Implementação

free do user-Mode Linux

=> cria condições para simular o Isolamento (No nível do SO)

Mais eficiente possível →
Só para Linux →
Para Windows precisamos de uma VH

Emulators

- Bochs
- Microsoft VPC for Mac
- QEMU → usado em Android Studio
- Virtutech Simics

⇒ Analisa os vestígios da memória numa aplicação

Cibersegurança

⇒ Não são eficientes

↓
Não são virtualizáveis

Cientes

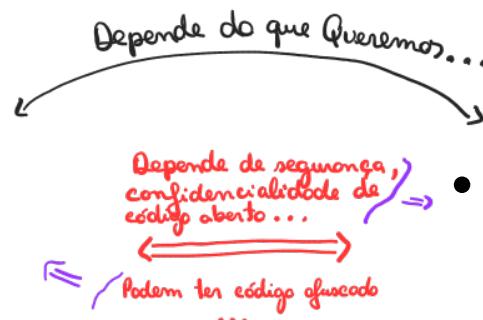
(Pode ser Cliente de outro Cliente)
Peer -two -Peer

- Em ambos os casos precisamos de um Middleware que saiba interpretar ...

- Uma aplicação em rede com o seu próprio protocolo

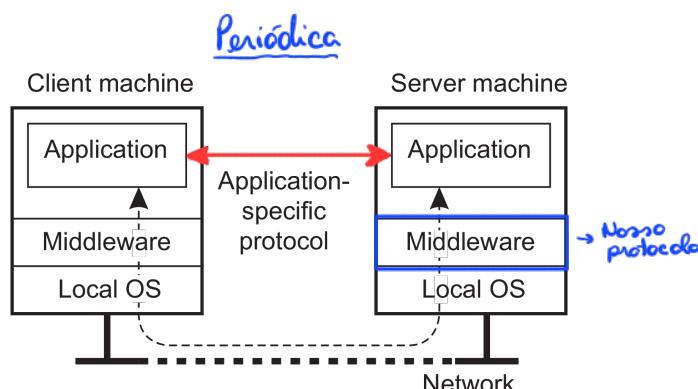
→ e.g.: Num jogo...

↳ tenta de fazer o protocolo...



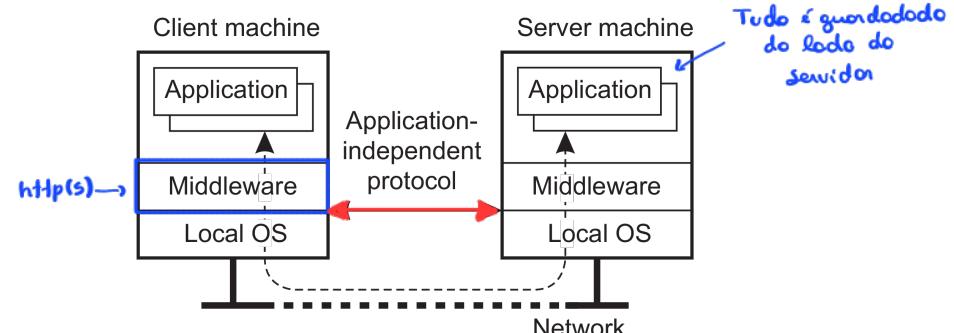
- Uma solução genérica que permite o acesso remoto a aplicações

e.g.: Web Browser ⇒ Mesmo protocolo para todos os App's



e.g., sincronização do calendário do telemóvel com o remoto

• Protocolo de sincronização no nível da aplicação ...

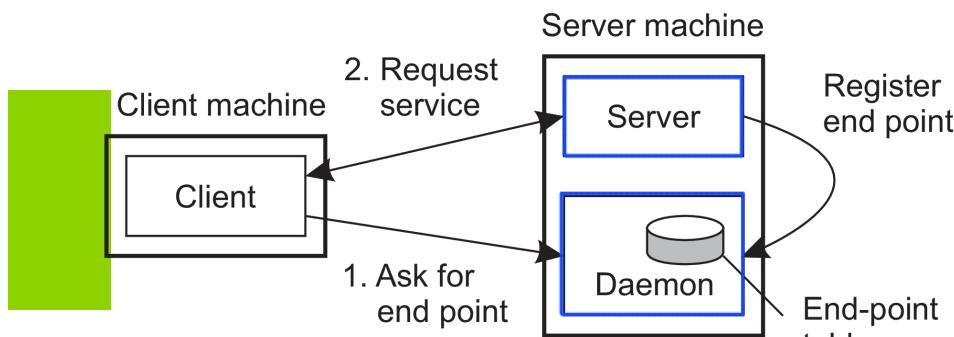


• Pedidos feitos de forma não periódica

Servidores

- Ligação Cliente-Servidor através de um daemon.

→ Dá-nos um ponto de contato com o cliente

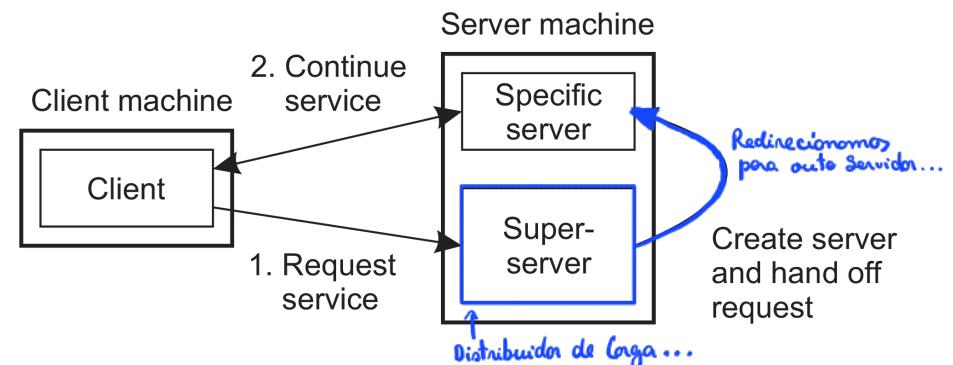


• Daemon trata do processo de ligação e de processamento em background

→ No nosso Projeto 1 só tudo no mesmo processo (usamos Selectors)

• Os clientes contactam o Servidor para pedir um "End-point" (um ponto)

- Ligação Cliente-Servidor através de um super-servidor.



• Super-Server trata apenas da ligação Cliente-Servidor

Érro muitas vezes cometido ...

Servidores Stateless vs Stateful

• Stateless

e.g.:
Super-Server
→ Atende todos e
distribui e equaciona
"Stateless"

- Não guardam qualquer informação entre pedidos do cliente *⇒ e.g.: Server usa o Cookies para não ter de guardar estados !!!
"Trabalho pelos clientes"*

- Evitam sobrecarga do servidor por necessidade de guardar informação e recuperar a mesma



Indicado para
Sistemas Distribuídos

(mas depende SEMPRE do)
contexto ...

• Stateful

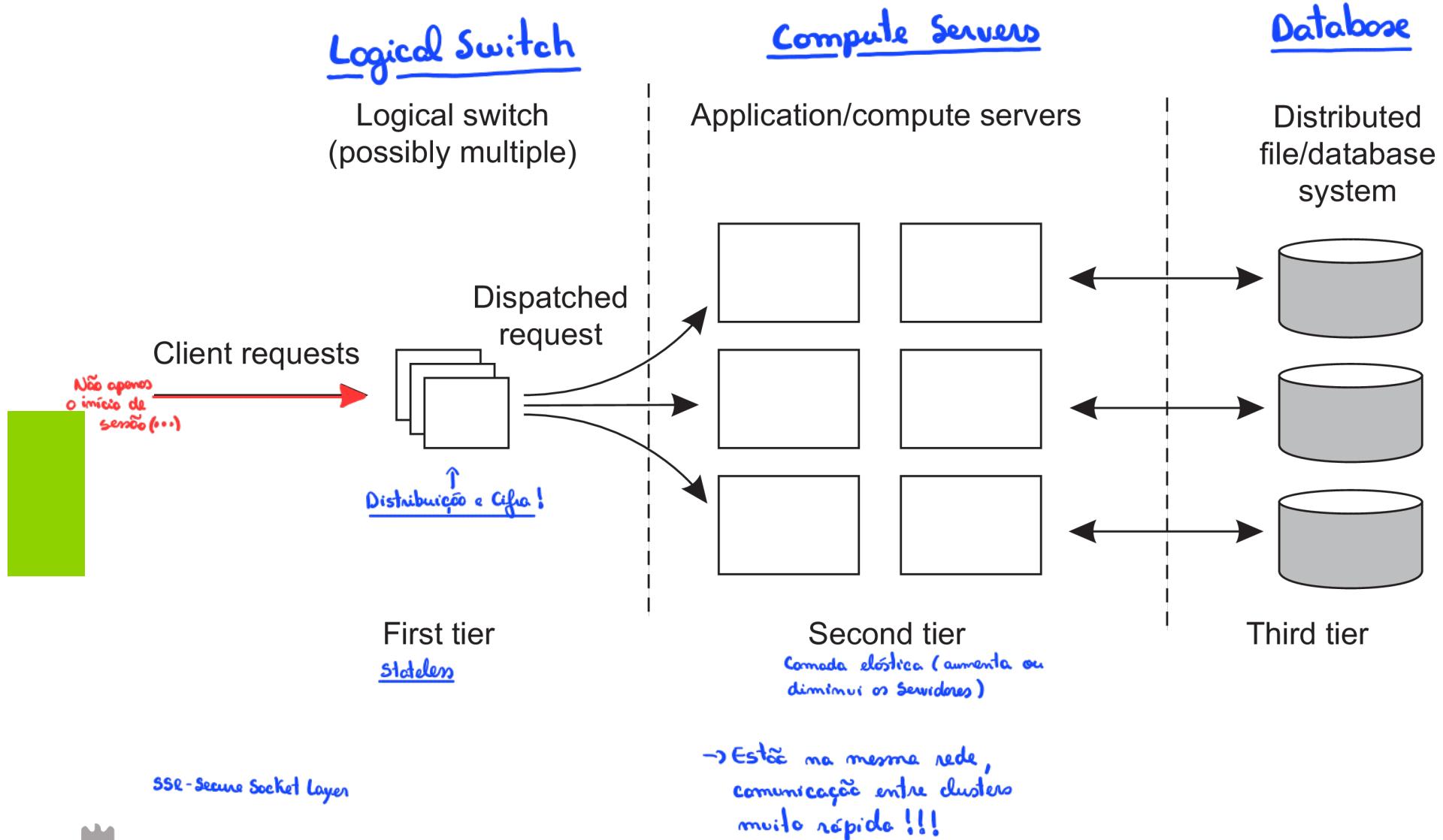
- Guardam informação de sessão *⇒ Server tem de guardar e monitorizar o estado da sessão (tema BD ou uma cache)*

- Permitem maior eficiência de comunicação, pois mantêm informação de contexto



Clusters de Servidores

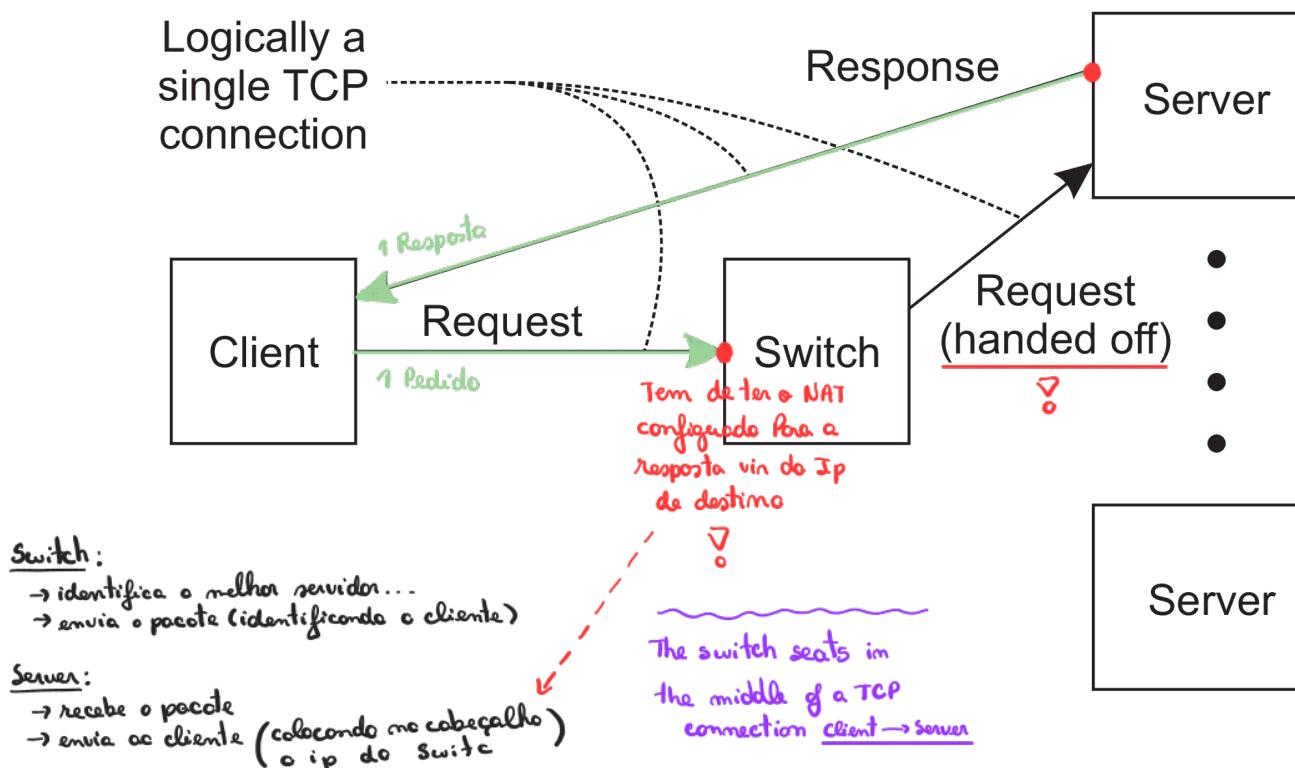
(Mais utilizada!)



→ No mono Projeto 1 é no mesmo processo...

TCP Handoff

Na prática...



Eficiente quando a resposta é muito maior que o pedido...
(é o caso dos Web Servers)

- O switch tem um papel importante na distribuição pelos cluster servers
- Nota: o switch tem que manter a relação Cliente-Servidor até a ligação TCP fechar...
Não é 100% Stateless!

↑ So far, we have been mainly concerned with distributed systems in which communication is limited to passing data.

Migração de Código

• Modelos:

④ Simples

- Mobilidade fraca
 - Envia também → contexto ↗
- Mobilidade forte
 - Envio o código a segmento de execução para ser reconhecido (pouca portabilidade...)
- Iniciado pelo emissor
 - ④ Difícil, ..., quem recebe tem de verificar autenticação ↗
 - e.g., descarregar programas para um "compute-server" enviar uma query para uma base de dados...
- Iniciado pelo receptor
 - ④ Fácil, só podes receber...
 - O cliente que é só um pode facilmente tratar desse código...

Reasons to do:

→ Performance

• Servidores sobrecarregados expalam partes da execução... (optimização de energia)

→ Privacy and security

• leva os modelos da AI para cada hospital em vez de centralizar...

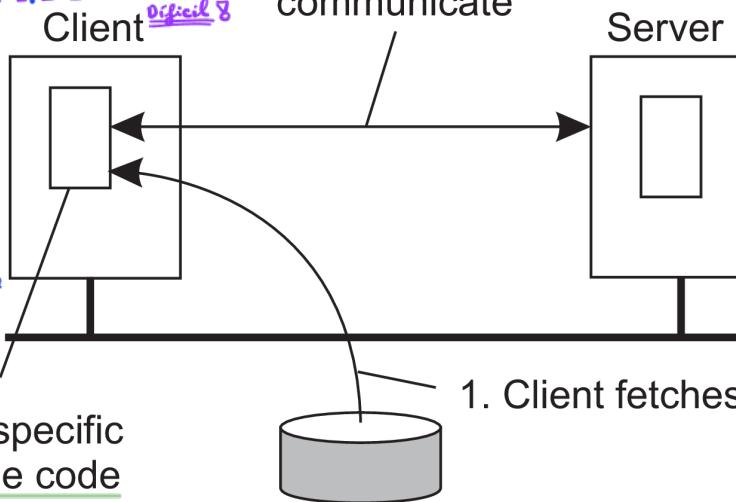
→ Flexibility

• Modularizar o Sistema e enviar...

Now, we pass programs.

(sometimes even while they are being executed)

2. Client and server communicate



Service-specific client-side code

→ Mete o código a carregar no cliente ⇒ Ou seja, não é que carregamos os JavaScript dos websites ↗

⇒ O Java foi feito para responder a isto!,,

⇒ O cliente não pode alterar o código... ↗

Code repository

⇒ Mobilidade fraca, iniciado pelo receptor

↳ Atenção à Segurança !!!

Migração e recursos locais

⇒ Simples quando temos, por exemplo, servidores...

• Ligação Processo-Recurso

• Ligação por identificador

Ex.: URL ⇒ Para um Recurso de um Servidor!
Ligaçao

→ Identificação:

- Servidor
- Base Dados (tudo tem de estar identificado)
- Recurso/Ligação

• Ligação por valor

Ex.: C/Java

• Ligação por tipo

Ex.: referências a tipos: monitor, impressora, etc

Os recursos podem ser acedidos (ou não) em função da sua natureza e identificadores

• Ligação Recurso-Máquina

• Não ligado

Ex.: ficheiros ⇒ Pode ordenar de um lado para o outro

→ Manda um Broadcast a dizer "Estou aqui, Sou uma impressora"

⇒ A aplicação não sabe qual impressora...

⇒ São fog a ligação

• Apertado

Ex.: DB's

⇒ Mudar implica sobre todos os identificadores! (Computação Distribuída precisa de controlar tudo...)

• Fixo

Ex.: dispositivos físicos, cartões, GPU's

⇒ Hardware ...