ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

ARQUITETURAS E PROCESSOS DE SISTEMAS DISTRIBUÍDOS

Daniel Cordeiro

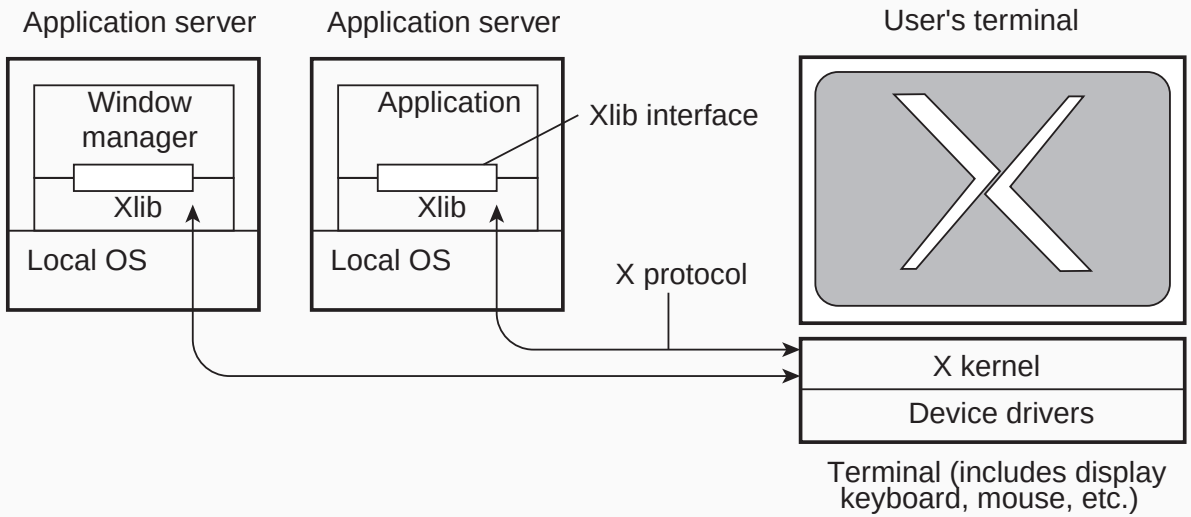
18 e 20 de abril de 2017

Escola de Artes, Ciências e Humanidades | EACH | USP

Clientes

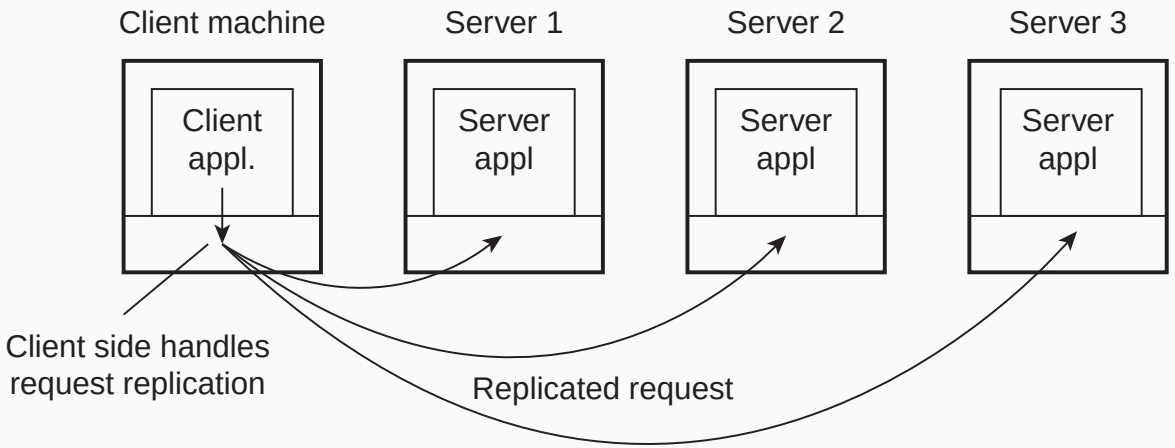
CLIENTES: INTERFACES DE USUÁRIOS

A maior parte dos softwares do lado do cliente é especializada em interfaces (gráficas) de usuário. O *X protocol* é um exemplo de *thin-client network computing*.



Geralmente adaptado para transparência de distribuição

- transparência de acesso: *stubs* do cliente para RPC
- transparência de localização/migração: deixe o software cliente manter o controle sobre a localização atual
- transparência de replicação: múltiplas evocações são gerenciadas pelo stub do cliente:



- transparência de falhas: podem geralmente ser responsabilidade só do cliente (que tenta esconder falhas de comunicação e do servidor)

Servidores

Modelo básico

Um processo que implementa um serviço específico em nome de uma coleção de clientes. Ele espera pela requisição de um cliente, garante que a requisição será tratada e, em seguida, passa a esperar pela próxima requisição.

Dois tipos básicos:

Servidores iterativos o servidor trata uma requisição antes de atender a próxima

Servidores concorrentes usa um despachante (*dispatcher*), que pega uma requisição e repassa seu tratamento a uma *thread*/processo separado

Observação

É mais comum encontrarmos servidores concorrentes: eles podem tratar múltiplas requisições mais facilmente, principalmente se for necessário realizar operações bloqueantes (em discos ou outros servidores).

ftp-data	20	File Transfer [Default Data]
ftp	21	File Transfer [Control]
telnet	23	Telnet
smtp	25	Simple Mail Transfer
login	49	Login Host Protocol
sunrpc	111	SUN RPC (portmapper)

Cada requisição a uma porta é atribuída a um processo dinamicamente, via *superservers* (processo que inicia subprocesso para tratar a requisição; ex: UNIX inetd) ou *daemons* (processos que se registram em uma porta).

Problema:

É possível interromper um servidor uma vez que ele já tiver aceito (ou estiver processando) uma requisição de serviço?

Solução 1: usar uma porta diferente para dados urgentes

- O servidor mantém uma thread/processo separado para mensagens urgentes
- Se uma mensagem urgente chegar, a requisição associada é colocada em espera
- É necessário que o SO ofereça escalonamento por prioridade

Problema:

É possível interromper um servidor uma vez que ele já tiver aceito (ou estiver processando) uma requisição de serviço?

Solução 1: usar uma porta diferente para dados urgentes

- O servidor mantém uma thread/processo separado para mensagens urgentes
- Se uma mensagem urgente chegar, a requisição associada é colocada em espera
- É necessário que o SO ofereça escalonamento por prioridade

Solução 2: usar comunicação de controle da camada de transporte

- TCP permite o envio de mensagens urgentes na mesma conexão
- Mensagens urgentes podem ser recebidas usando tratamento de sinais do SO

Servidores *stateless*

Não mantém informação exata sobre o status de um cliente após ter processado uma requisição:

- Não guarda se um arquivo foi aberto (simplesmente fecha-o e abre de novo se necessário)
- Não promete invalidar o cache do cliente
- Não rastreia os seus clientes

Servidores *stateless*

Não mantém informação exata sobre o status de um cliente após ter processado uma requisição:

- Não guarda se um arquivo foi aberto (simplesmente fecha-o e abre de novo se necessário)
- Não promete invalidar o cache do cliente
- Não rastreia os seus clientes

Consequências

- Clientes e servidores são completamente independentes
- Inconsistências de estado devido a problemas no cliente ou servidor são reduzidas
- Possível perda de desempenho. Um servidor não pode antecipar o comportamento do cliente (ex: *prefetching*)

O uso de comunicação orientada a conexão viola o modelo stateless?

O uso de conexões com estado não violam o fato de que os servidores não guardam estado.

Mas é necessário ter em mente que a camada de transporte, sim, mantém estado. Melhor seria usar um protocolo stateless combinado com operações idempotentes[1].

[1] A idempotência é a propriedade que algumas operações têm de poderem ser aplicadas várias vezes sem que o valor do resultado se altere após a aplicação inicial.

Servidores com estado (*stateful*)

Guardam o status de seus clientes:

- Registram quando um arquivo foi aberto para realização de *prefetching*
- Sabem quando o cliente possui cache dos dados e permitem que os clientes mantenham cópias locais de dados compartilhados

Servidores com estado (*stateful*)

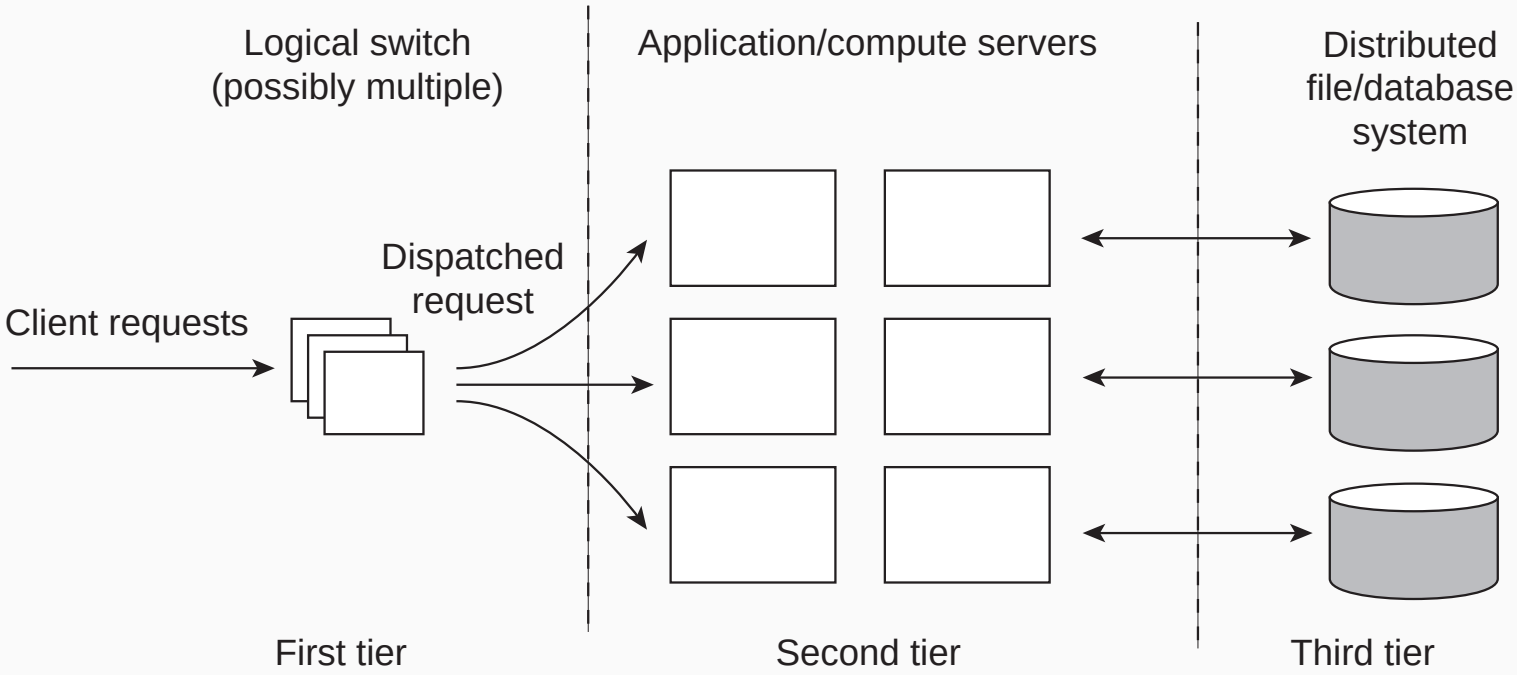
Guardam o status de seus clientes:

- Registram quando um arquivo foi aberto para realização de *prefetching*
- Sabem quando o cliente possui cache dos dados e permitem que os clientes mantenham cópias locais de dados compartilhados

Observação:

O desempenho de servidores *stateful* pode ser extremamente alto, desde que seja permitido que os clientes mantenham cópias locais dos dados. Nesses casos, confiabilidade não é o maior problema.

AGLOMERADOS DE SERVIDORES: TRÊS CAMADAS DIFERENTES



Elemento crucial

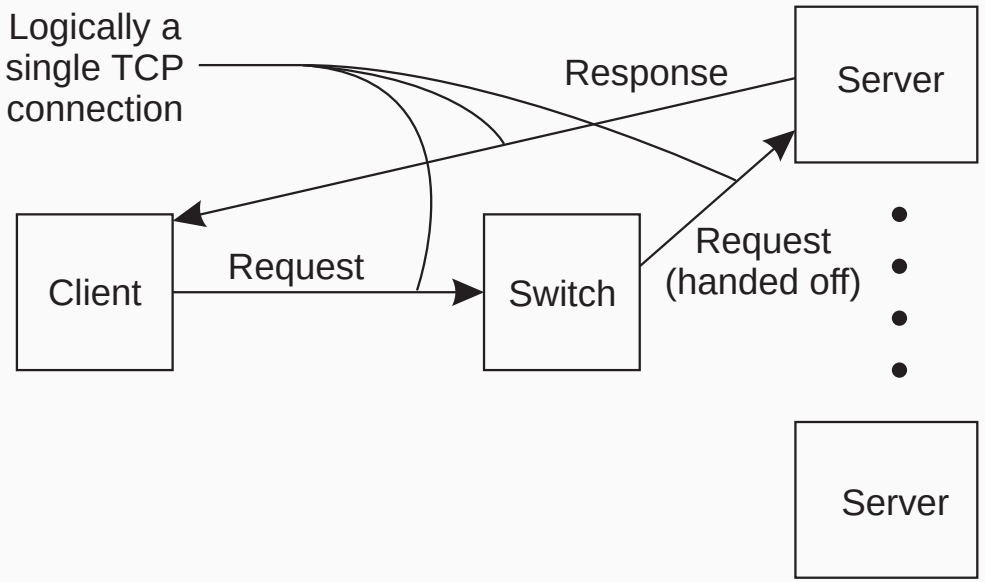
A primeira camada é responsável por repassar as requisições para um servidor apropriado.

Observação:

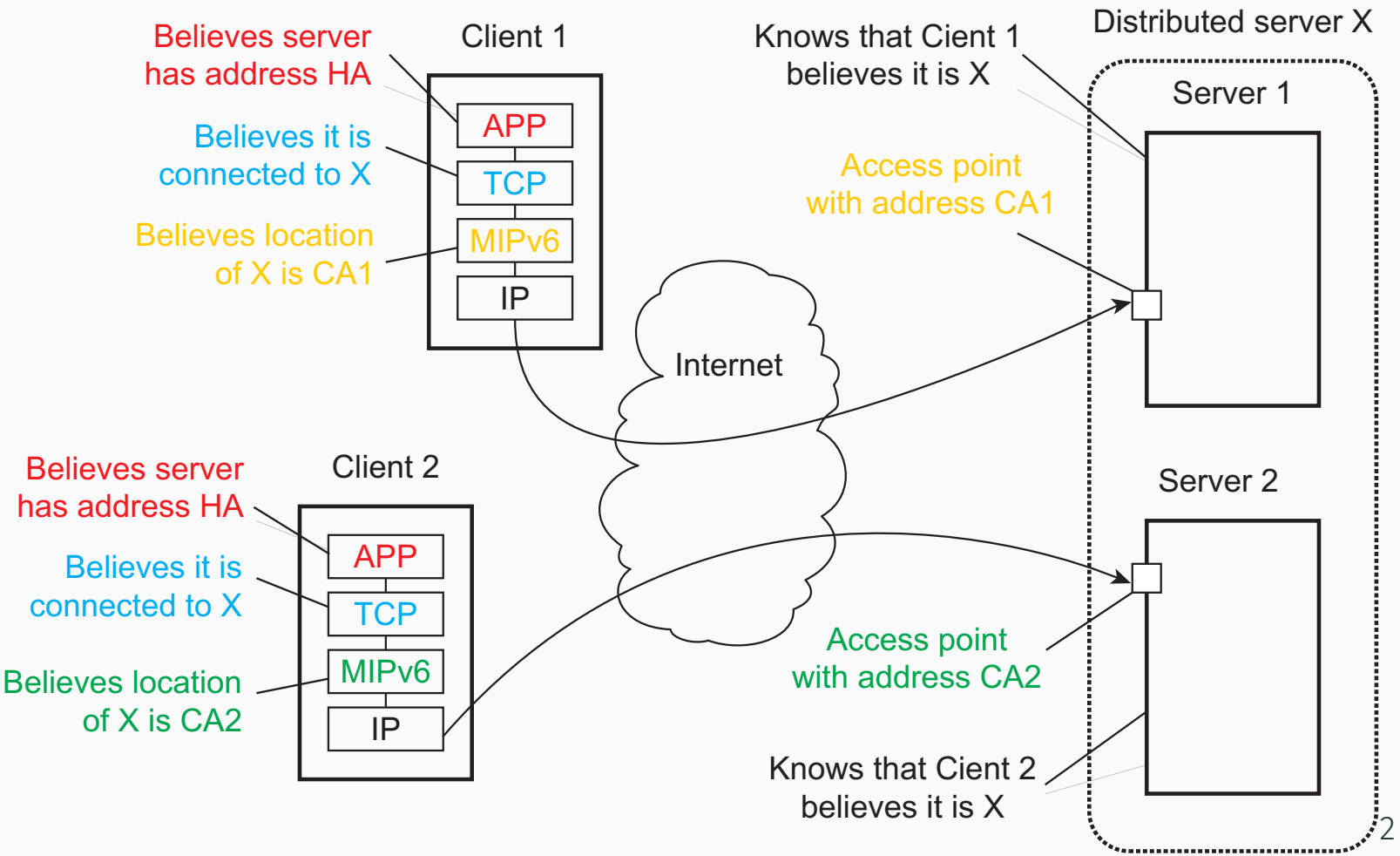
Ter uma única camada tratando toda a comunicação de/para o aglomerado pode levar a criação de um gargalo.

Solução:

Várias, mas uma popular é o chamado TCP-handoff:



SERVIDORES DISTRIBUÍDOS COM ENDEREÇOS IPV6 ESTÁVEIS



[2] MIPv6 = Mobile IPV6; HA = Home Address; CA = Care-of Address

Clientes com Mobile IPv6 podem criar conexões com qualquer outro par de forma transparente:

- Cliente *C* configura uma conexão IPv6 para o home address *HA*.
- *HA* é mantido (no nível da rede) por um home agent, que repassa a conexão para um endereço care-of *CA* registrado
- *C* aplica uma otimização de rota ao encaminhar os pacotes diretamente para o endereço do *CA*, sem passar pelo *home agent*.

Clientes com Mobile IPv6 podem criar conexões com qualquer outro par de forma transparente:

- Cliente *C* configura uma conexão IPv6 para o home address *HA*.
- *HA* é mantido (no nível da rede) por um home agent, que repassa a conexão para um endereço care-of *CA* registrado
- *C* aplica uma otimização de rota ao encaminhar os pacotes diretamente para o endereço do *CA*, sem passar pelo *home agent*.

CDNs colaborativas

O servidor original mantém um *home address*, mas repassa as conexões para o endereço para um servidor colaborador. O original e o colaborador "parecem" um único servidor.

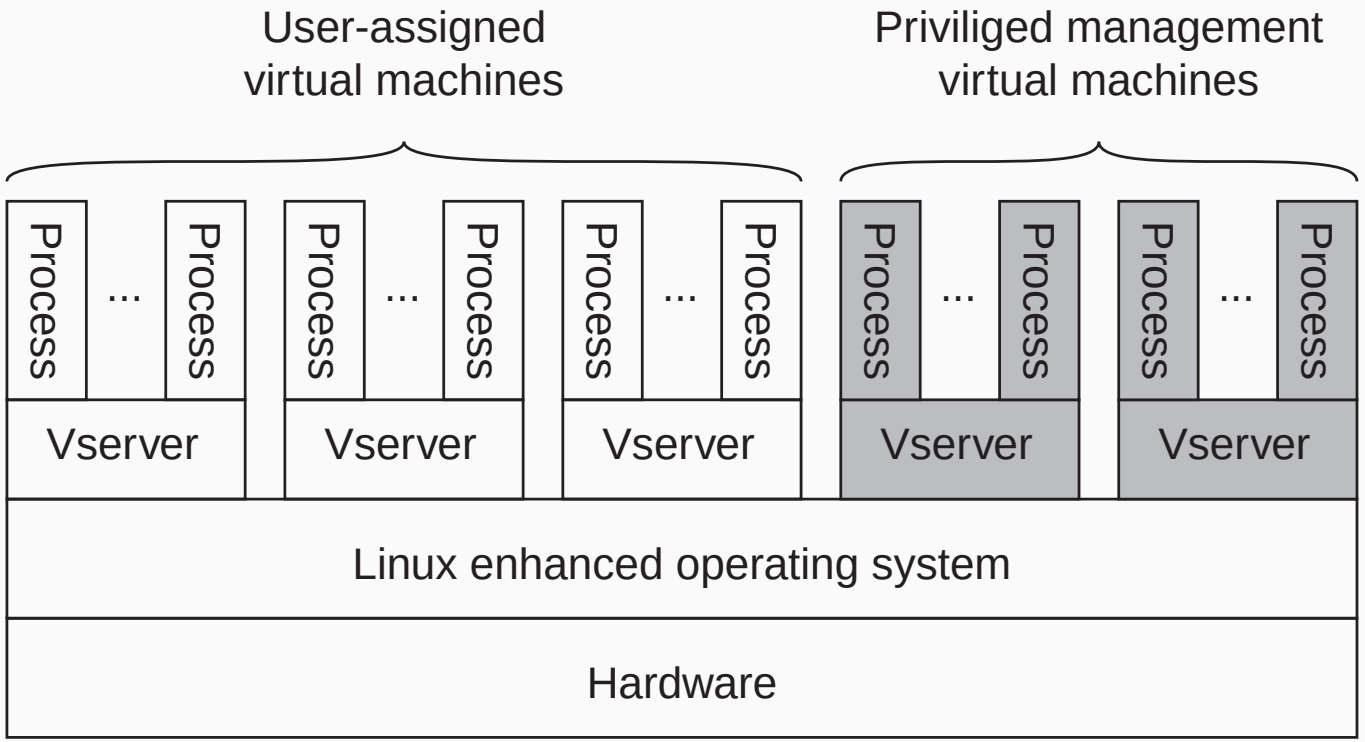
EXEMPLO: PLANETLAB

Diferentes organizações contribuem com máquinas, que serão compartilhadas em vários experimentos.

Problema:

É preciso garantir que as diferentes aplicações distribuídas não atrapalhem umas às outras. Solução: virtualização.

EXEMPLO: PLANETLAB



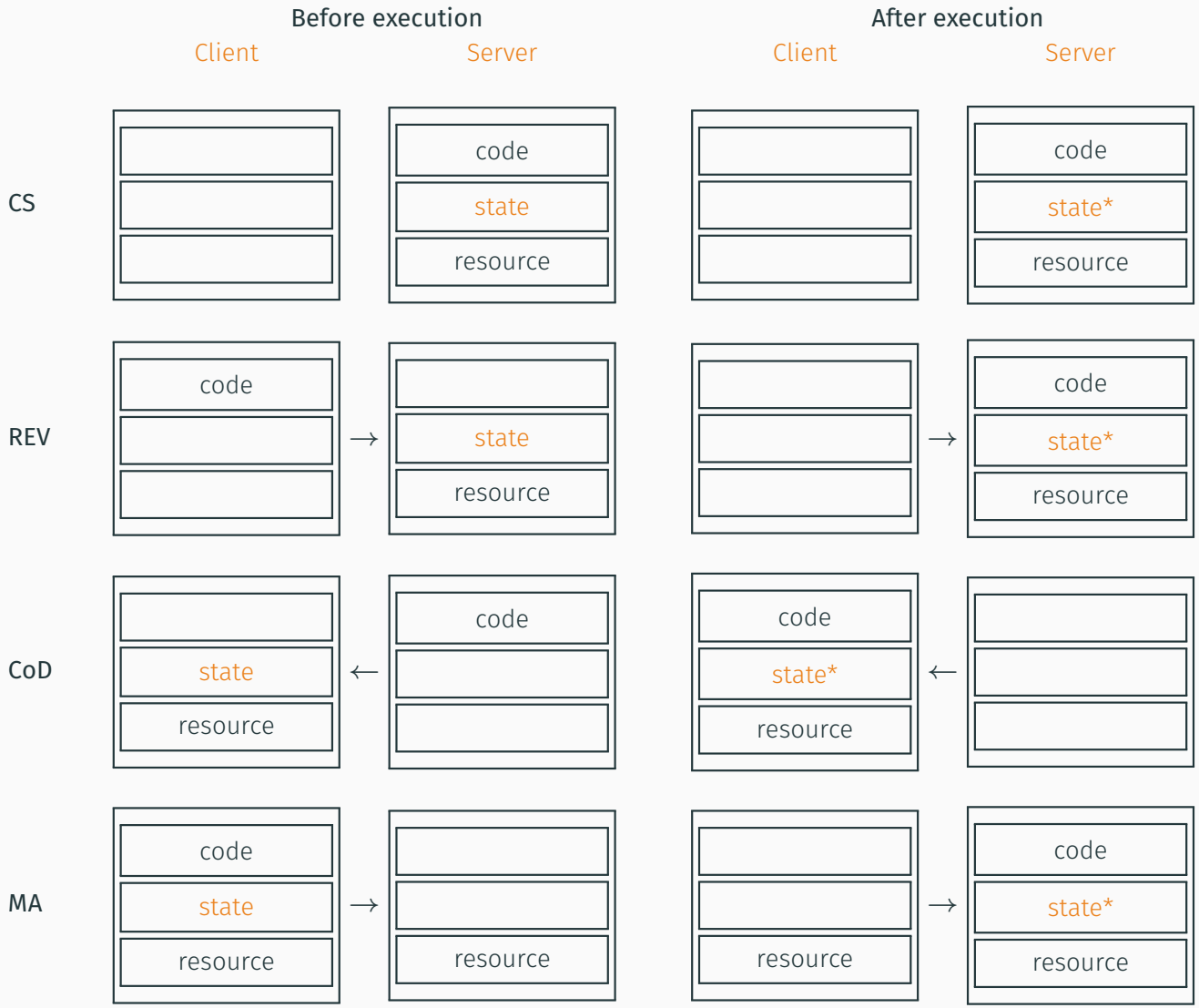
Vserver: ambiente independente e protegido com suas próprias bibliotecas, versões do servidor, etc. Aplicações distribuídas são atribuídas a uma **coleção** de Vservers distribuídas entre múltiplas máquinas físicas (*slice*).

Migração de código

MIGRAÇÃO DE CÓDIGO

- Abordagens para realização de migração de código
- Migração e recursos locais
- Migração em sistemas heterogêneos

MIGRAÇÃO DE CÓDIGO: CONTEXTO



CS: Cliente–Servidor
CoD: Code-on-demand

REV: Remote evaluation
MA: Agentes móveis

Componentes do objeto:

Segmento de código contém o código real

Segmento de dados contém o estado

Estado da execução contém o contexto das threads executando o código do objeto

Mobilidade fraca

Apenas os segmentos de código e dados são migrados (e a execução é reiniciada):

- Relativamente simples, especialmente se o código é portátil
- Duas modalidades: envio de código (*push*) e busca de código (*pull*)

Mobilidade forte

Move o componente inteiro, incluindo o seu estado de execução.

- Migração: move o objeto inteiro de uma máquina para outra
- Clonagem: inicia um clone e o configura para o mesmo estado de execução.

Problema:

Um objeto usa recursos locais que podem não estar disponíveis no novo local.

Tipos de recursos

Fixos: o recurso não pode ser migrado (ex: hardware)

Anexado: a princípio o recurso pode ser migrado, mas migração terá alto custo (ex: banco de dados local)

Desanexado: o recurso pode ser facilmente movido junto com o objeto (ex: um cache)

Ligação objeto–recurso

Por identificador: o objeto requer uma instância específica de um recurso (ex: um banco de dados específico)

Por valor: o objeto requer o valor de um recurso (ex: o conjunto de entradas no cache)

Por tipo: o objeto requer que um determinado tipo de recurso esteja disponível (ex: um monitor colorido)

GERENCIAMENTO DE RECURSOS LOCAIS

	Desanexado	Anexado	Fixo
ID	MV (ou GR)	GR (ou MV)	GR
Valor	CP (ou MV,GR)	GR (ou CP)	GR
Tipo	RB (ou MV, GR)	RB (ou GR, CP)	RB (ou GR)

GR = Estabelecer referência global no sistema

MV = Mover o recurso

CP = Copiar o valor do recurso

RB = Religa a um recurso local disponível

Problema principal

- A máquina destino pode não ser adequada para executar o código migrado
- A definição de contexto de thread/processo/processador é altamente dependente do hardware, sistema operacional e bibliotecas locais

Única solução

Usar alguma máquina abstrata que é implementada nas diferentes plataformas:

- Linguagens interpretadas, que possuem suas próximas MVs
- Uma MV Virtual (como vimos no início da aula)

Migração de imagens: três alternativas

1. Enviar as páginas de memória para a nova máquina e reenviar aquelas que forem modificadas durante o processo de migração
2. Interromper a máquina virtual, migrar a memória, e iniciar a nova máquina virtual
3. Fazer com que a nova máquina virtual recupere as páginas de memória conforme for necessário: processos são iniciados na nova máquina imediatamente e copiam as páginas de memória sob demanda

Problema

Uma migração completa pode levar dezenas de segundos. Além disso, é preciso ficar atento ao fato de que um serviço poderá ficar indisponível por vários segundos durante a migração.

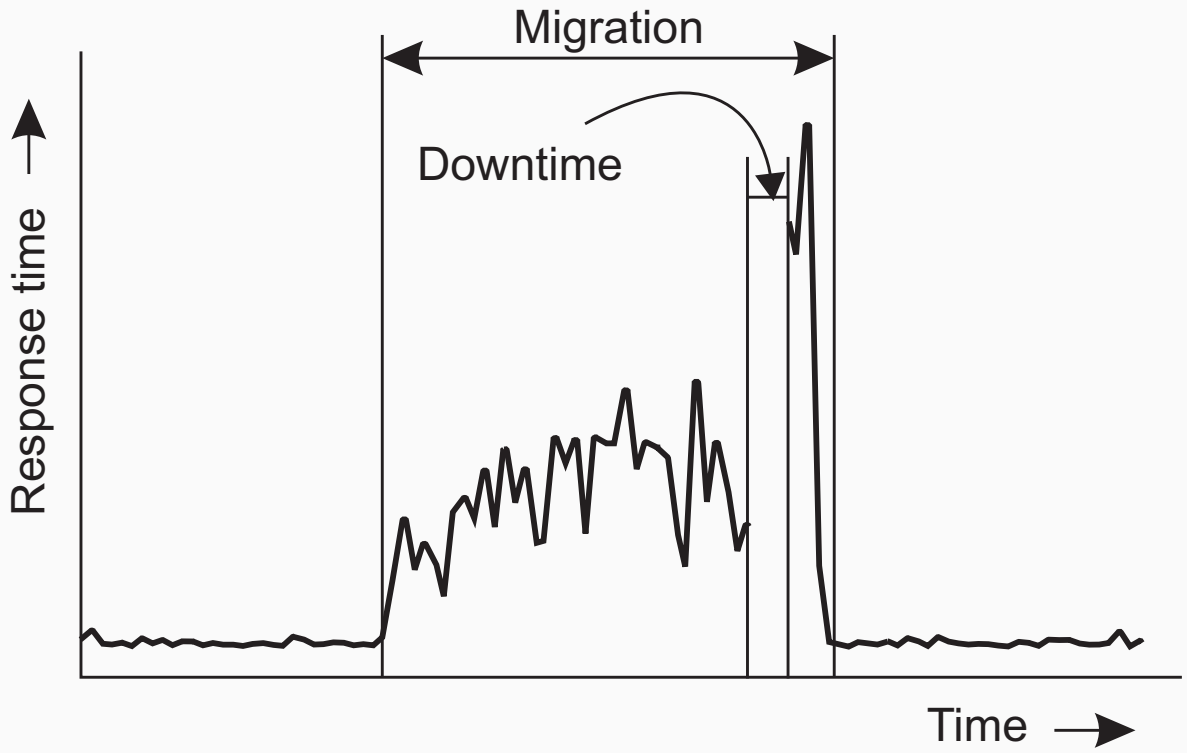


Figura: Medições do tempo de resposta de uma VM durante uma migração