# ACH 2147 — Desenvolvimento de Sistemas de Informação Distribuídos

## Aula 08: Processos (parte 3)

Prof. Renan Alves

Escola de Artes, Ciências e Humanidades — EACH — USP

22/03/2024

# Servidores: Organização geral

## Modelo básico

Um processo que implementa um serviço específico para um conjunto de clientes. Ele aguarda uma requisição de algum cliente e garante que a requisição seja atendida, após o que aguarda a próxima requisição.

# Servidores: Organização geral

## Modelo básico

Um processo que implementa um serviço específico para um conjunto de clientes. Ele aguarda uma requisição de algum cliente e garante que a requisição seja atendida, após o que aguarda a próxima requisição.

## Dois tipos básicos

- Servidor iterativo: O servidor lida com a requisição antes de atender à próxima requisição.
- Servidor concorrente: Usa um dispatcher, que repassa requisições de entrada para uma thread/processo separado.

## Observação

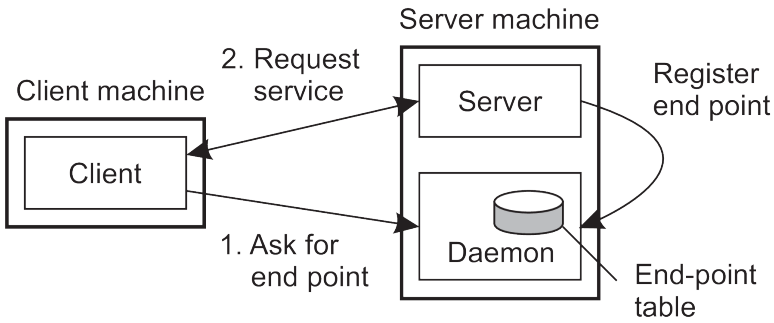
Os servidores concorrentes são a norma: eles podem lidar facilmente com várias solicitações, especialmente na presença de operações bloqueantes (para discos ou outros servidores).

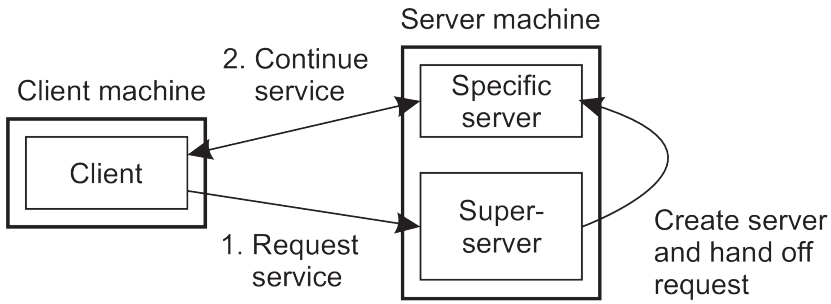
# Contatando um servidor

## Observação: a maioria dos serviços está ligada a uma porta específica

| | | |
|---|---|---|
| ftp-data | 20 | Transferência de Arquivos [Dados Padrão] |
| ftp | 21 | Transferência de Arquivos [Controle] |
| telnet | 23 | Telnet |
| smtp | 25 | Simple Mail Transfer Protocol |
| www | 80 | Web (HTTP) |

## Atribuição dinâmica: duas abordagens



# Comunicação fora da banda (out-of-band)

## Questão

É possível interromper um servidor depois de aceitar (ou estar no processo de aceitar) uma requisição?

# Comunicação fora da banda (out-of-band)

## Questão

É possível interromper um servidor depois de aceitar (ou estar no processo de aceitar) uma requisição?

## Solução 1: Usar uma porta separada para dados urgentes

- O servidor tem uma thread/processo separado para mensagens urgentes
- A mensagem urgente chega $\Rightarrow$ a requisição associada é colocada em espera
- Observação: requer suporte do SO para escalonamento baseado em prioridade

# Comunicação fora da banda (out-of-band)

## Questão

É possível interromper um servidor depois de aceitar (ou estar no processo de aceitar) uma requisição?

## Solução 1: Usar uma porta separada para dados urgentes

- O servidor tem uma thread/processo separado para mensagens urgentes
- A mensagem urgente chega $\Rightarrow$ a requisição associada é colocada em espera
- Observação: requer suporte do SO para escalonamento baseado em prioridade

## Solução 2: Usar as facilidades da camada de transporte

- Exemplo: TCP permite mensagens urgentes na mesma conexão
- Mensagens urgentes podem ser capturadas usando técnicas de sinalização do SO

# Servidores e estado

## Servidores sem estado

Nunca mantêm informações precisas sobre o estado de um cliente após terem atendido uma requisição:

- Não registram se um arquivo foi aberto (simplesmente fechar novamente após o acesso)
- Não prometem invalidar o cache de um cliente
- Não sabem se houve mudanças no clientes
- Meio termo: ”soft state”

# Servidores e estado

## Servidores sem estado

Nunca mantêm informações precisas sobre o estado de um cliente após terem atendido uma requisição:

- Não registram se um arquivo foi aberto (simplesmente fechar novamente após o acesso)
- Não prometem invalidar o cache de um cliente
- Não sabem se houve mudanças no clientes
- Meio termo: ”soft state”

## Consequências

- Clientes e servidores são completamente independentes
- Inconsistências de estado devido a falhas de cliente ou servidor são reduzidas
- Possível perda de desempenho porque, por exemplo, um servidor não pode antecipar o comportamento do cliente (e.g. pré-carregamento de blocos de arquivos)

# Servidores e estado

## Servidores sem estado

Nunca mantêm informações precisas sobre o estado de um cliente após terem atendido uma requisição:

- Não registram se um arquivo foi aberto (simplesmente fechar novamente após o acesso)
- Não prometem invalidar o cache de um cliente
- Não sabem se houve mudanças no clientes
- Meio termo: ”soft state”

## Consequências

- Clientes e servidores são completamente independentes
- Inconsistências de estado devido a falhas de cliente ou servidor são reduzidas
- Possível perda de desempenho porque, por exemplo, um servidor não pode antecipar o comportamento do cliente (e.g. pré-carregamento de blocos de arquivos)

## Pergunta

A comunicação orientada a conexão se encaixa em um design sem estado?

# Servidores e estado

## Servidores com estado

Tem conhecimento do estado e histórico de seus clientes:

- Registra que um arquivo foi aberto anteriormente, para que o pré-carregamento possa ser feito
- Sabe quais dados um cliente armazenou em cache e permite que os clientes mantenham cópias locais de dados compartilhados

# Servidores e estado

## Servidores com estado

Tem conhecimento do estado e histórico de seus clientes:

- Registra que um arquivo foi aberto anteriormente, para que o pré-carregamento possa ser feito
- Sabe quais dados um cliente armazenou em cache e permite que os clientes mantenham cópias locais de dados compartilhados

## Observação

O desempenho dos servidores com estado pode ser extremamente alto, desde que os clientes possam manter cópias locais. Na maioria das vezes, a confiabilidade não é um grande problema.

# Servidores e estado

## Servidores com estado

Tem conhecimento do estado e histórico de seus clientes:

- Registra que um arquivo foi aberto anteriormente, para que o pré-carregamento possa ser feito
- Sabe quais dados um cliente armazenou em cache e permite que os clientes mantenham cópias locais de dados compartilhados

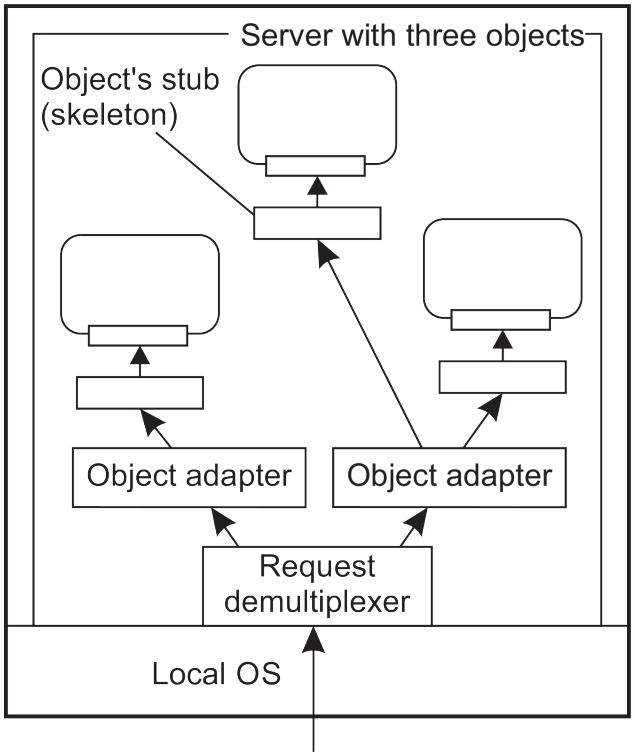
## Observação

O desempenho dos servidores com estado pode ser extremamente alto, desde que os clientes possam manter cópias locais. Na maioria das vezes, a confiabilidade não é um grande problema.

## Pergunta

Cookies: stateful ou stateless?

# Servidores de objetos




- Política de ativação: quais ações tomar quando uma requisição de invocação chega:
  - Onde estão o código e os dados do objeto?
    - Objetos podem ser transientes: existem (no máximo) enquanto servidor existir
  - Como fazer uso de threads?
  - Manter o estado modificado do objeto, se houver?
- Adaptador de objeto: elemento genérico que implementa uma política de ativação específica (como invocar um objeto)

# Exemplo: Ice runtime system– um servidor

```
1  import sys, Ice
2  import Demo
3
4  class PrinterI(Demo.Printer):
5      def __init__(self, t):
6          self.t = t
7
8      def printString(self, s, current=None):
9          print(self.t, s)
10
11 communicator = Ice.initialize(sys.argv)
12
13 adapter = communicator.createObjectAdapterWithEndpoints("SimpleAdapter", "default -p 11000")
14 object1 = PrinterI("Object1 says:")
15 object2 = PrinterI("Object2 says:")
16 adapter.add(object1, communicator.stringToIdentity("SimplePrinter1"))
17 adapter.add(object2, communicator.stringToIdentity("SimplePrinter2"))
18 adapter.activate()
19
20 communicator.waitForShutdown()
```

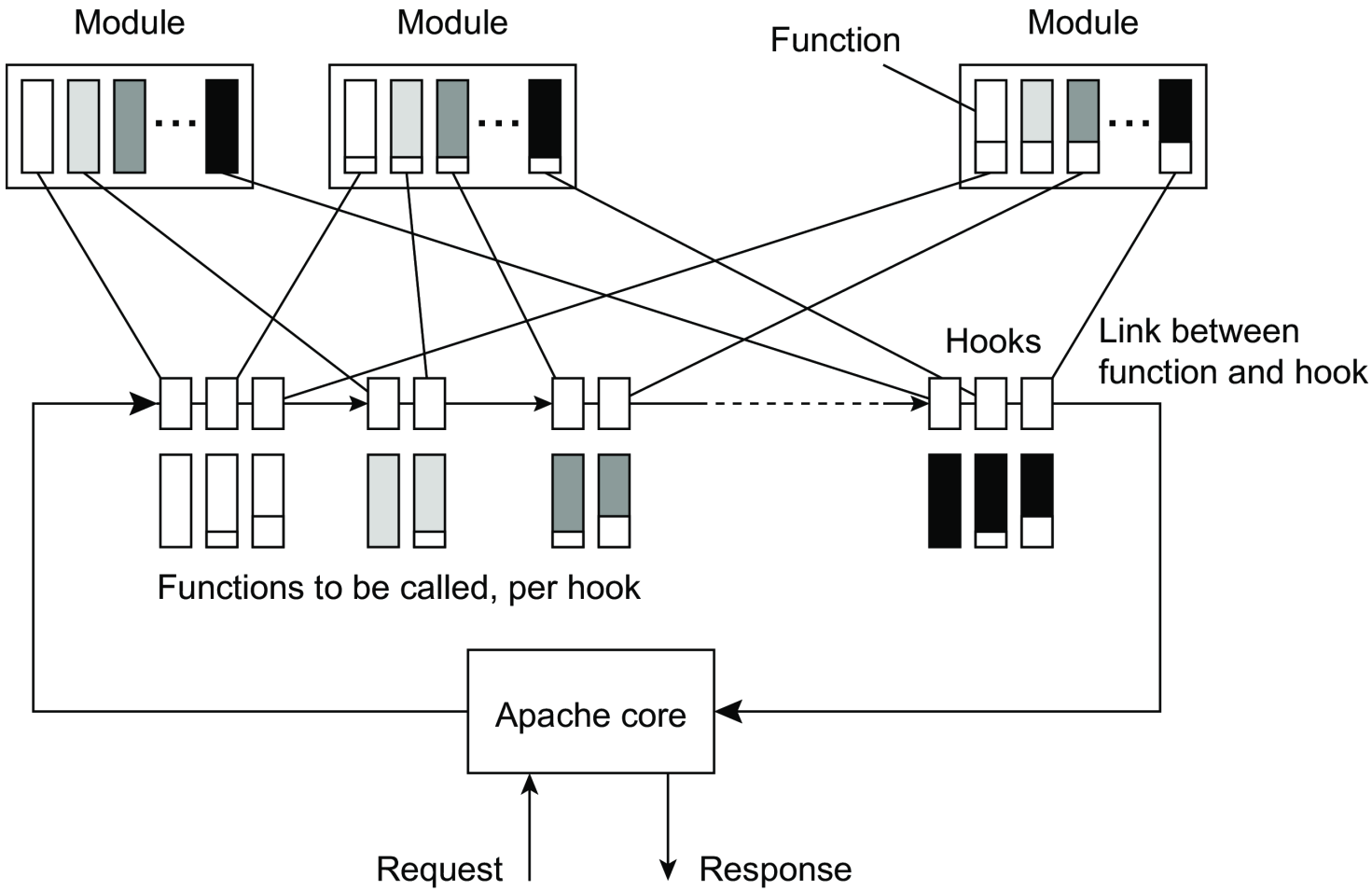
# Exemplo: Ice runtime system – um cliente

```
1  import sys, Ice
2  import Demo
3
4  communicator = Ice.initialize(sys.argv)
5
6  base1 = communicator.stringToProxy("SimplePrinter1:default -p 11000")
7  base2 = communicator.stringToProxy("SimplePrinter2:default -p 11000")
8  printer1 = Demo.PrinterPrx.checkedCast(base1)
9  printer2 = Demo.PrinterPrx.checkedCast(base2)
10 if (not printer1) or (not printer2):
11     raise RuntimeError("Invalid proxy")
12
13 printer1.printString("Hello World from printer1!")
14 printer2.printString("Hello World from printer2!")
15
16 communicator.waitForShutdown()
```

*Object1 says: Hello World from printer1!*
*Object2 says: Hello World from printer2!*

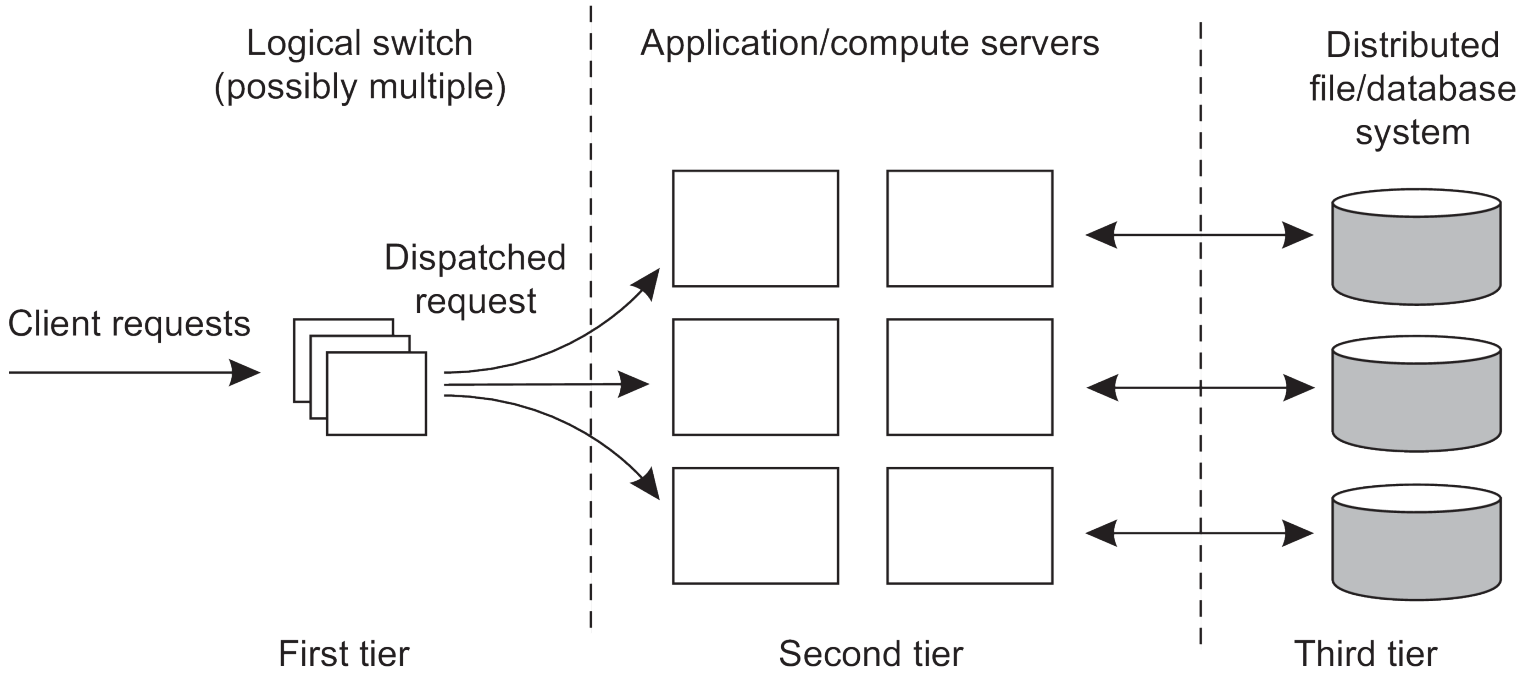
# Exemplo: o servidor web Apache

- Apache Portable Runtime (APR)



# Clusters de servidores: Três diferentes camadas

## Organização comum




## Elemento crucial

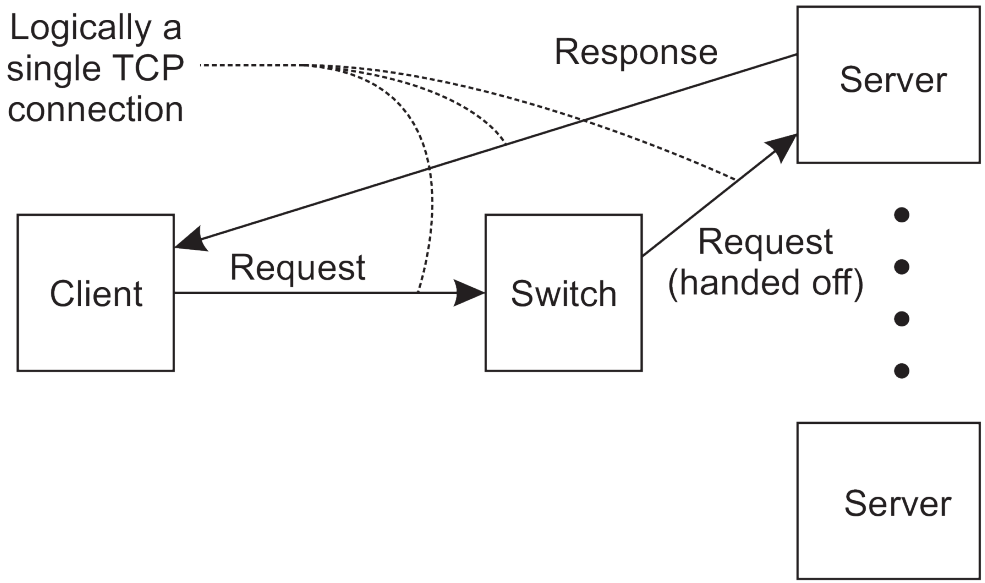
A primeira camada é geralmente responsável por encaminhar requisições para um servidor apropriado: despacho de requisições

# Lidando com requisições

## Observação

Ter a primeira camada lidar com toda a comunicação do/para o cluster pode levar a um gargalo.

## Uma solução: handoff TCP



# Quando os servidores estão espalhados pela Internet

## Observação

Espalhar servidores pela Internet pode introduzir problemas administrativos. Isso pode ser contornado na maioria da vezes usando data centers de um único provedor de nuvem.

## Despacho de requisições: se a localidade for importante

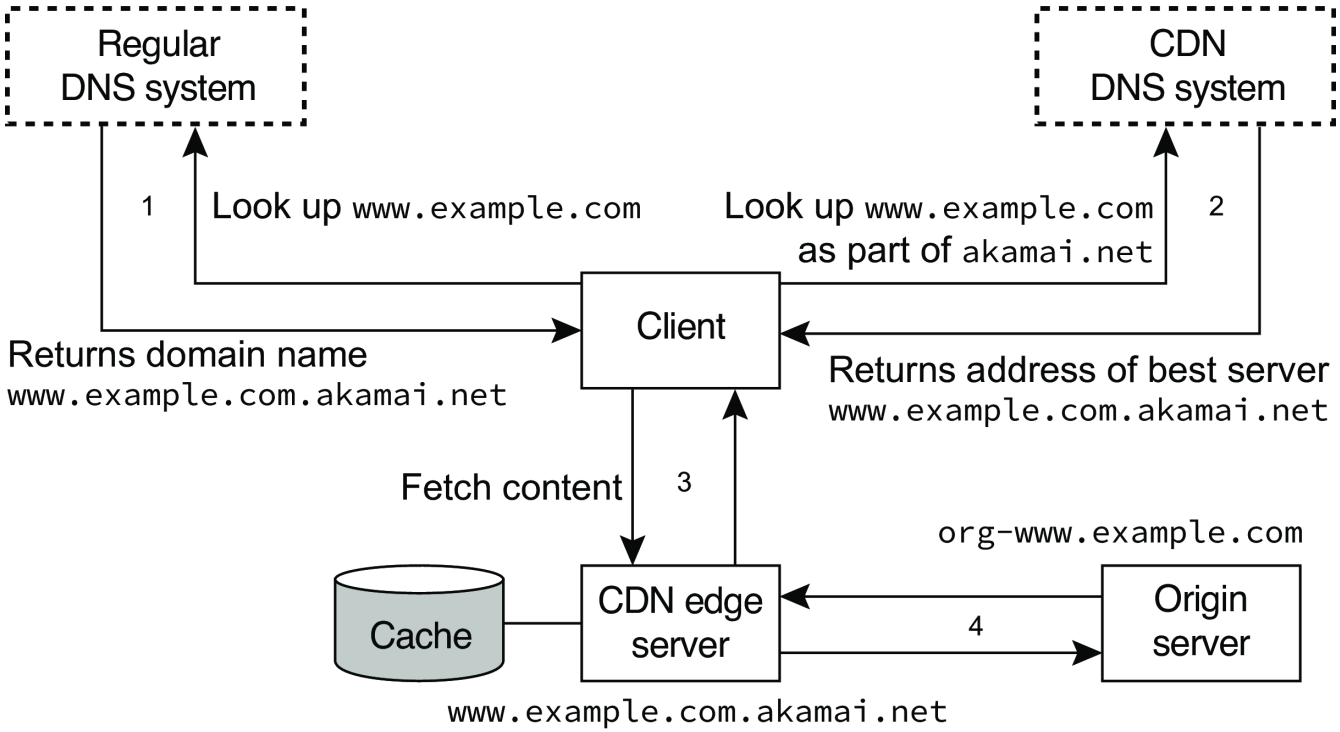
Abordagem comum: usar DNS:

1. O cliente procura um serviço específico através do DNS - o endereço IP do cliente faz parte da solicitação
2. O servidor DNS mantém o controle das réplicas de servidores para o serviço solicitado e retorna o endereço do servidor mais próximo.

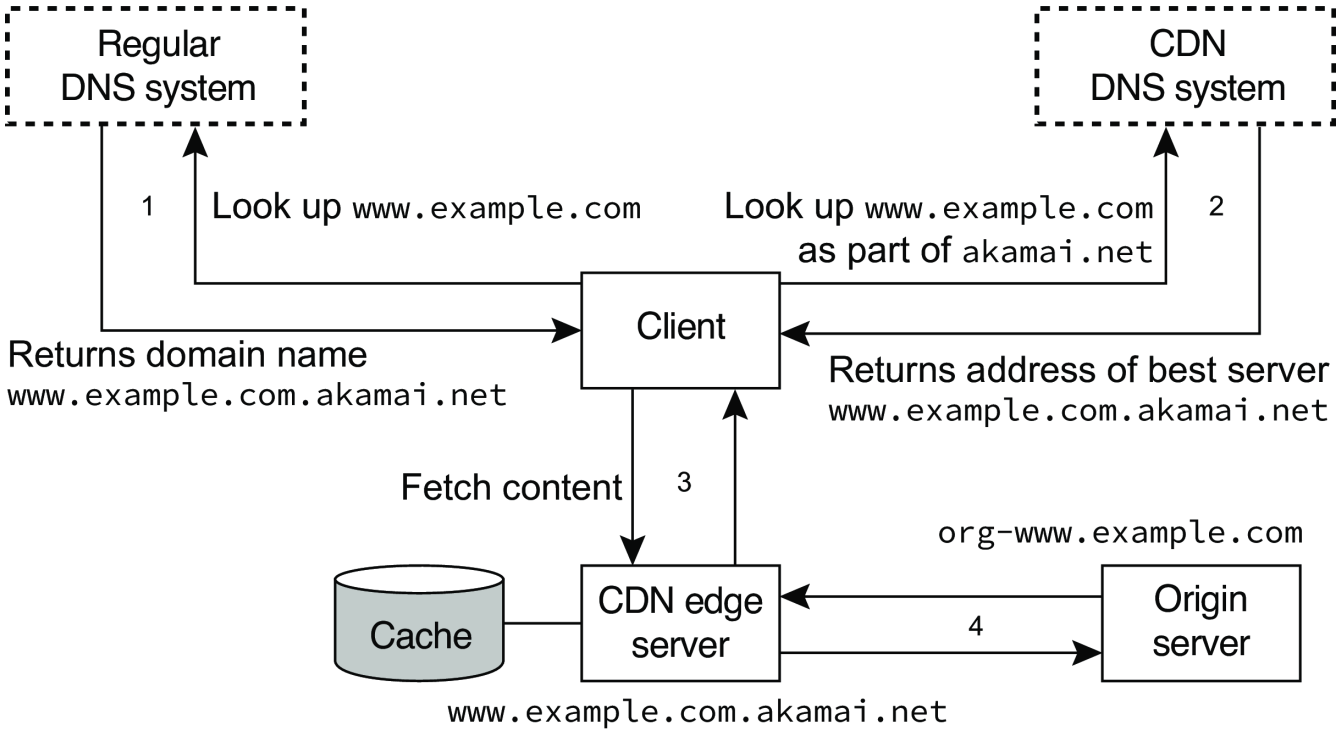
## Transparência para o cliente

Para manter o cliente sem saber da distribuição, permitir que o resolvedor DNS atue em nome do cliente. O problema é que o servidor DNS pode realmente estar longe do local do cliente de fato.

# Uma versão simplificada do CDN da Akamai



# Uma versão simplificada do CDN da Akamai




## Nota importante

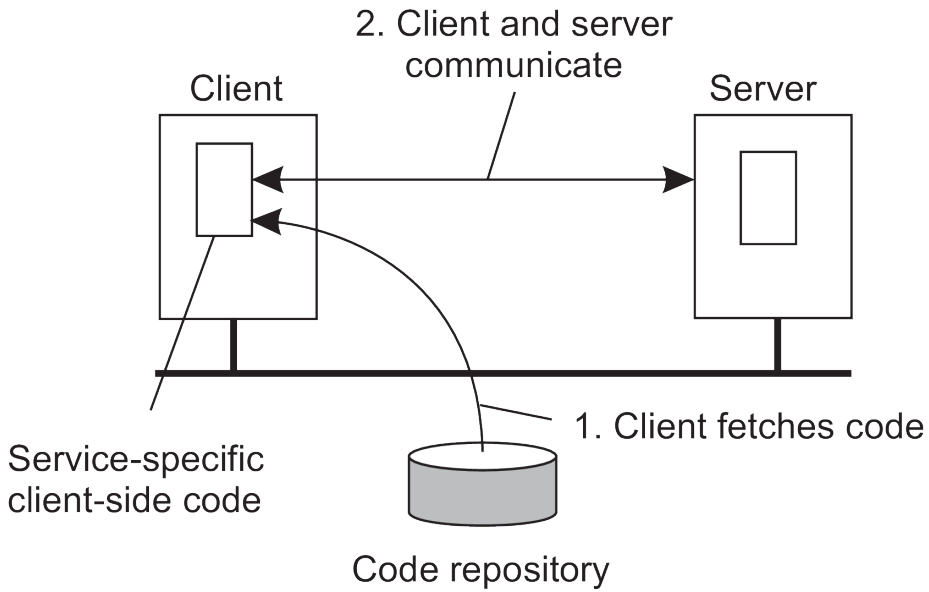
O cache muitas vezes é sofisticado o suficiente para conter mais do que apenas dados passivos. Grande parte do código de aplicação do servidor de origem pode ser movido para o cache também.

# Razões para migrar código

## Distribuição de carga

- Garantir que os servidores em um datacenter estejam suficientemente carregados (por exemplo, para evitar desperdício de energia)
- Minimizar a comunicação garantindo que a computação esteja próxima de onde os dados estão (pense em computação móvel).

## Flexibilidade: movendo código para um cliente quando necessário



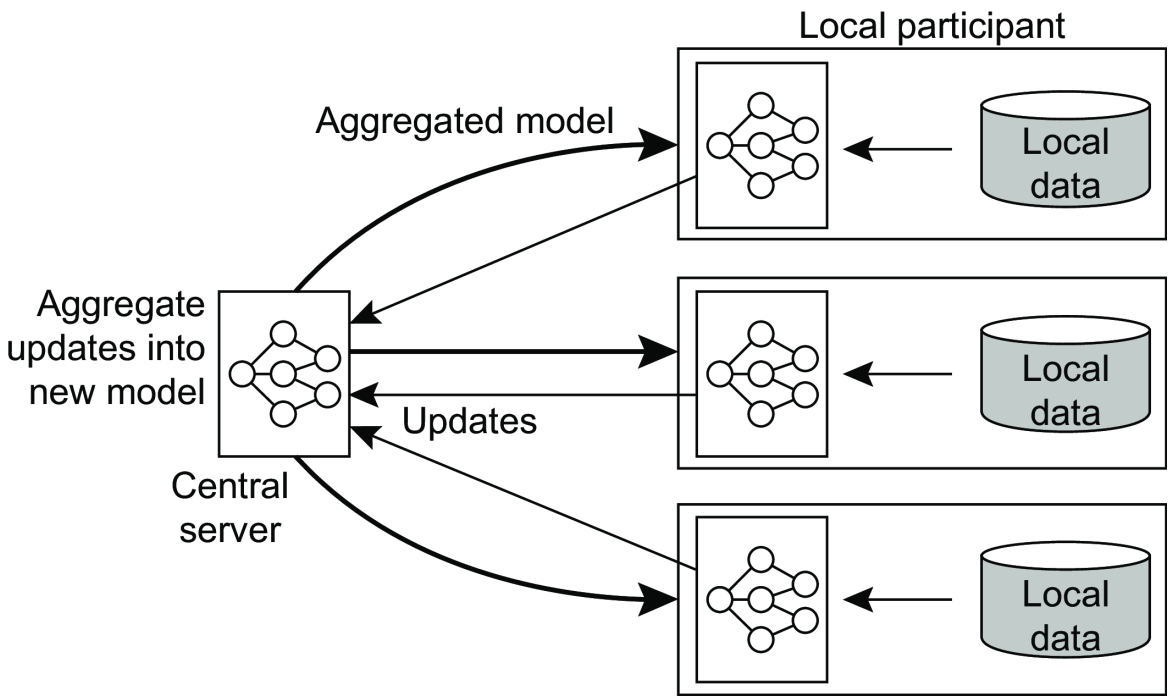

Evita a pré-instalação de software e aumenta a configuração dinâmica.

# Razões para migrar código

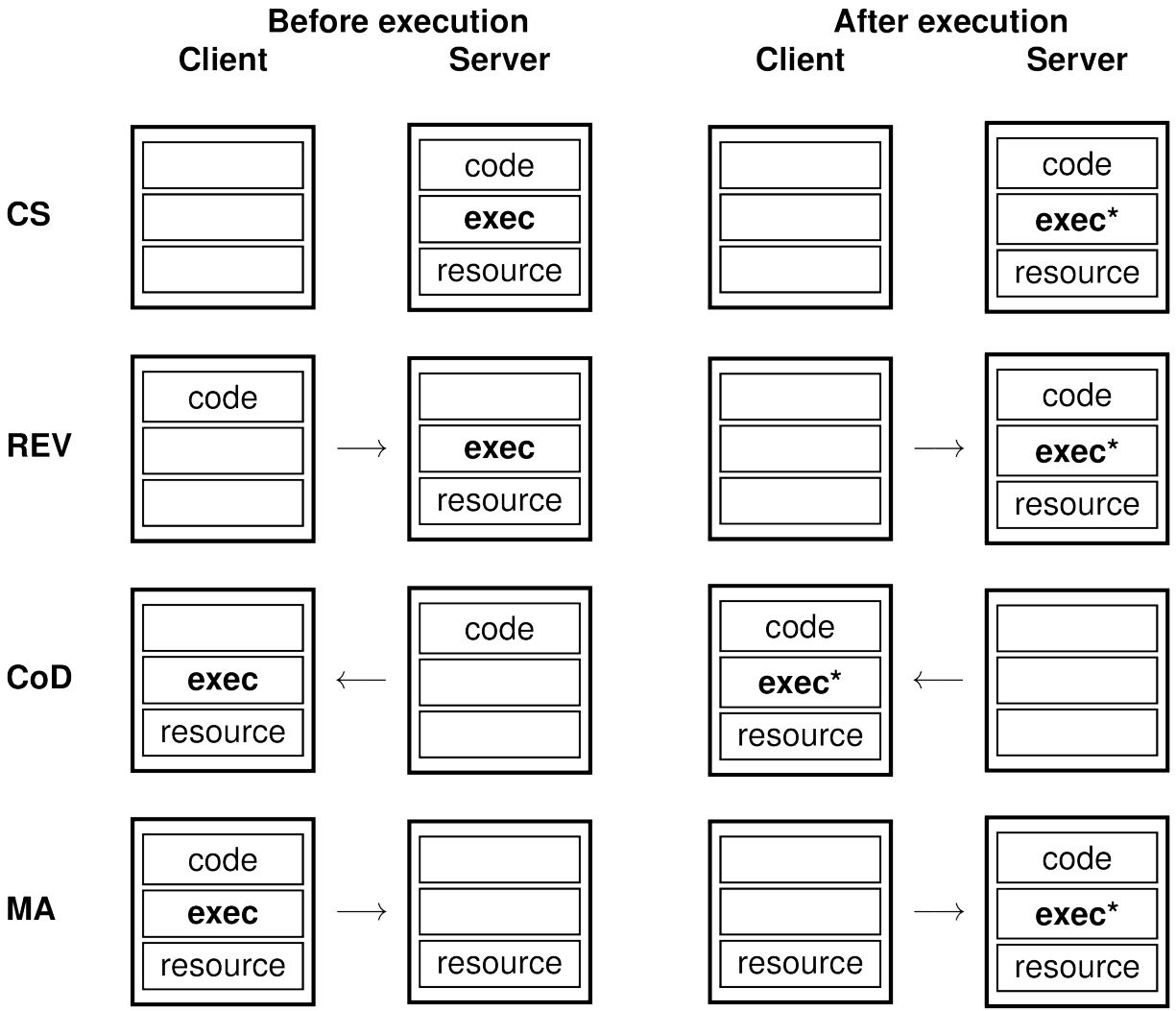
## Privacidade e segurança

Em muitos casos, não se pode mover dados para outra localização, por diversos motivos (muitas vezes motivos legais). Solução: mover o código para os dados.

## Exemplo: aprendizado de máquina federado



# Paradigmas para mobilidade de código




CS: Client-Server
CoD: Code-on-demand

REV: Remote evaluation
MA: Mobile agents

# Mobilidade forte e fraca

## Componentes do processo

- Segmento de código: contém o código real
- Segmento de dados: contém os recursos usados (dados)
- Estado de execução: contém o contexto da thread em execução

## Mobilidade fraca: mover apenas o segmento de código e de dados (e reiniciar a execução)

- Relativamente simples, especialmente se o código for portável
- Distinguir envio de código (push) de obteção de código (pull)

## Mobilidade forte: mover todo o componente, incluindo estado de execução

- Migração: mover processo inteiro de uma máquina para outra
- Clonagem: iniciar um clone e configurá-lo no mesmo estado de execução.

# Migração em sistemas heterogêneos

## Principal problema

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

## Única solução: máquina abstrata implementada em diferentes plataformas

- Linguagens interpretadas, efetivamente tendo sua própria VM
- Monitores de máquina virtual

## Observação

Como os contêineres dependem diretamente do sistema operacional subjacente, sua migração em ambientes heterogêneos está longe de ser trivial, assim como a migração de processos.

# Migrando uma máquina virtual

## Migrando imagens: três alternativas

1. Enviar páginas de memória para a nova máquina e reenviar as que são modificadas posteriormente durante o processo de migração.
2. Parar a máquina virtual atual; migre a memória e inicie a nova máquina virtual.
3. Permitir que a nova máquina virtual puxe as páginas de memória conforme necessário: os processos iniciam na nova máquina virtual imediatamente e copiam páginas de memória sob demanda.

# Desempenho da migração de máquinas virtuais

## Problema

Uma migração completa pode levar dezenas de segundos. Também precisamos levar em conta que durante a migração, o serviço estará completamente indisponível por vários segundos.

## Medição de tempo de resposta durante a migração