ACH2147 - Desenvolvimento de Sistemas de Informação Distribuídos

Aula 08 – Clientes, Servidores e Migração de Código

Norton Trevisan Roman

28 de abril de 2022

Clientes, Servidores e Migração de Código

- Clientes
- Servidores
- Migração de Código

Clientes, Servidores e Migração de Código

- **Clientes**
- Servidores
- Migração de Código

Clientes

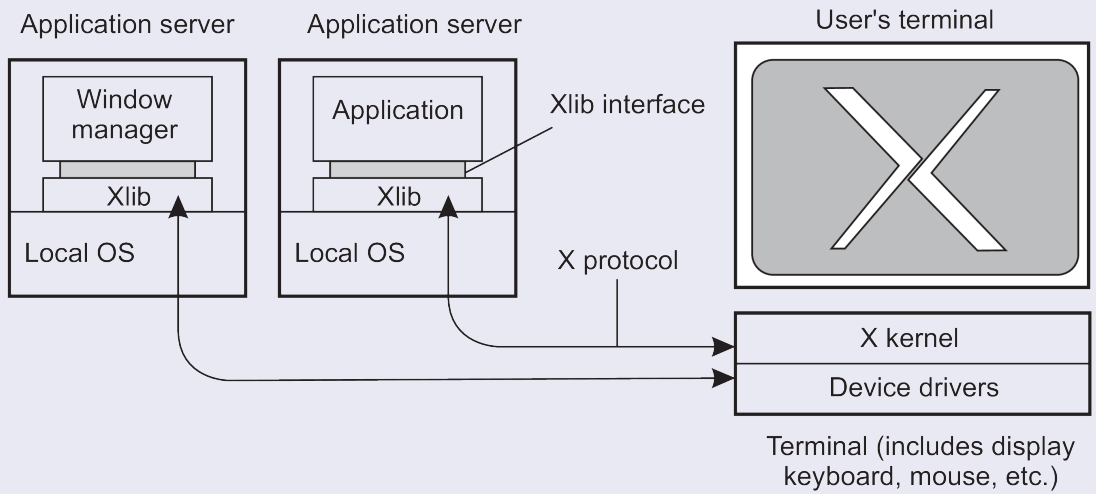
Interfaces de usuários em rede

- A principal tarefa de máquinas cliente é fornecer os meios para os usuários interagirem com servidores remotos
  - A maior parte dos softwares do lado do cliente é especializada em interfaces (gráficas) de usuário

Clientes

Interfaces de usuários em rede

- A principal tarefa de máquinas cliente é fornecer os meios para os usuários interagirem com servidores remotos
  - 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*



Clientes

Interfaces de usuários em rede

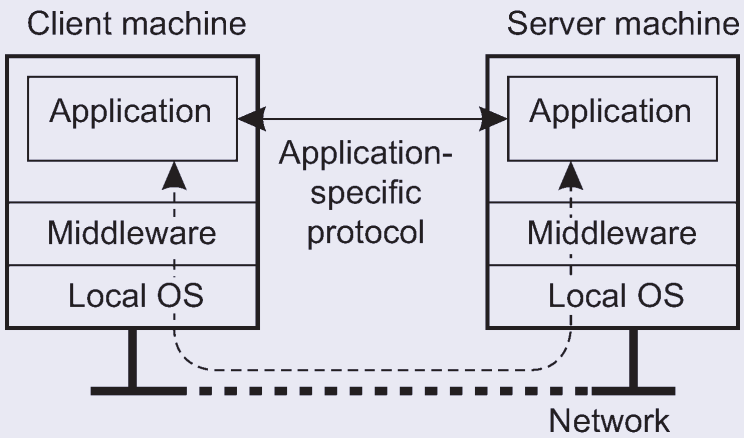
- Há 2 modos de se implementar essa interação

Clientes

Interfaces de usuários em rede

- Há 2 modos de se implementar essa interação

Para cada serviço remoto o cliente tem uma contraparte provedora do serviço no lado do servidor

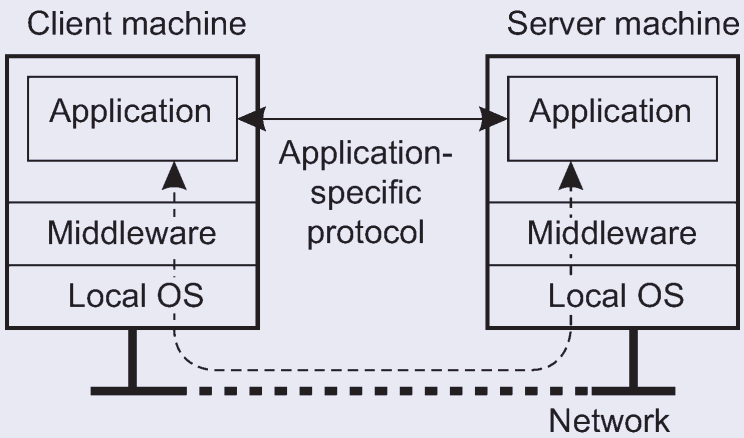


Clientes

Interfaces de usuários em rede

- Há 2 modos de se implementar essa interação

Caso em que um **protocolo em nível de aplicação** trata da sincronização (ex: calendários em *smartphones*)

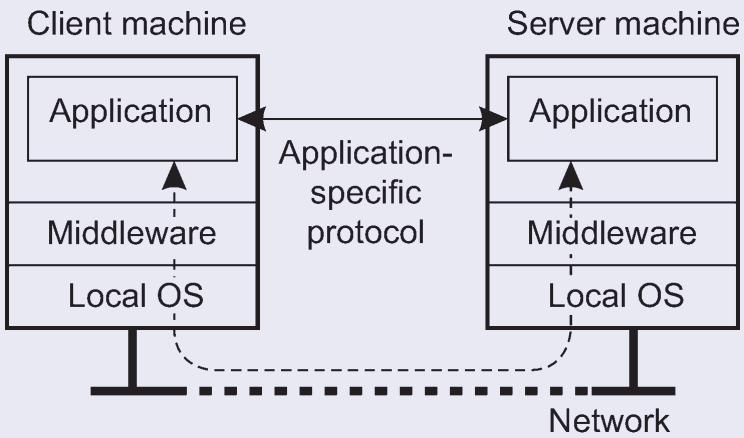


Clientes

Interfaces de usuários em rede

- Há 2 modos de se implementar essa interação

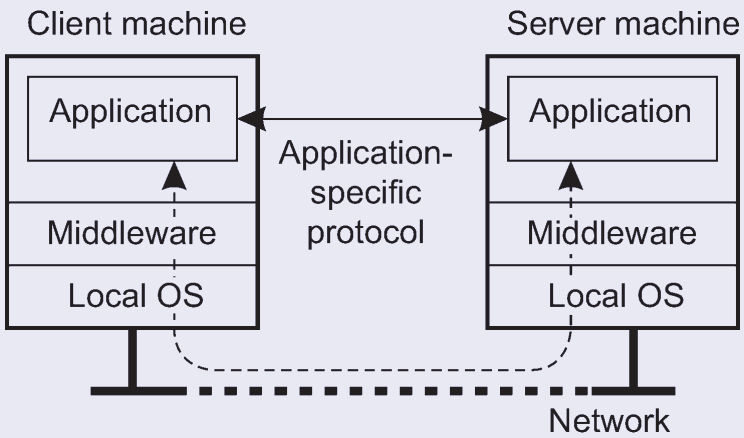
Solução em nível de aplicação



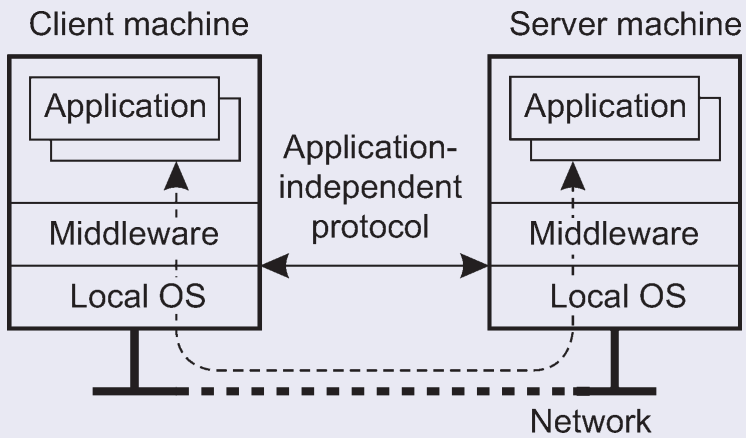
Interfaces de usuários em rede

- Há 2 modos de se implementar essa interação

Solução em nível de aplicação



Alternativamente, podemos fornecer acesso aos serviços remotos via uma interface com o cliente

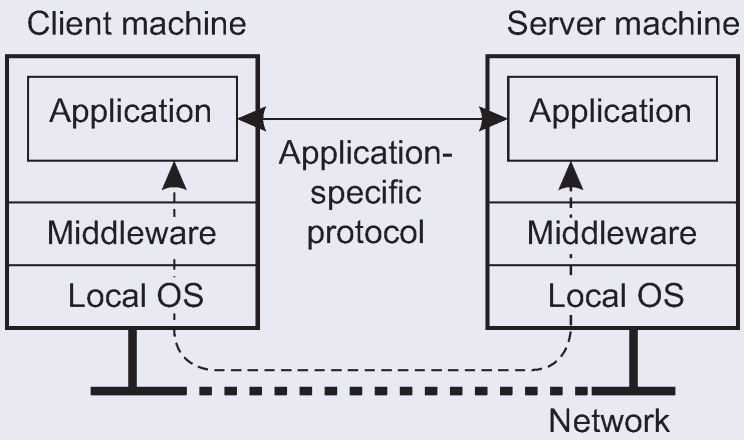


Clientes

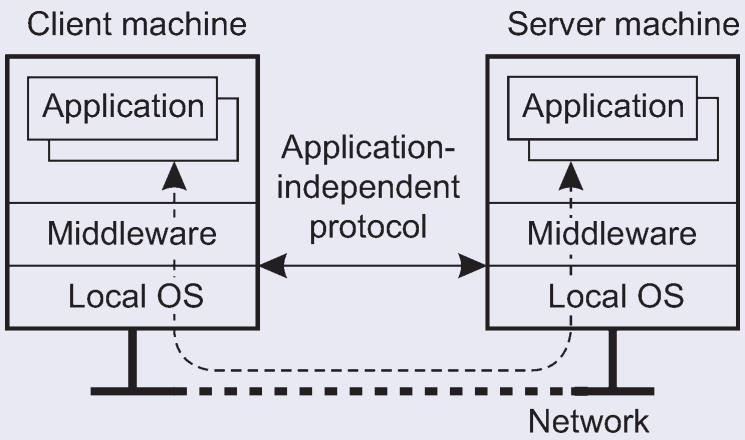
Interfaces de usuários em rede

- Há 2 modos de se implementar essa interação

Solução em nível de aplicação



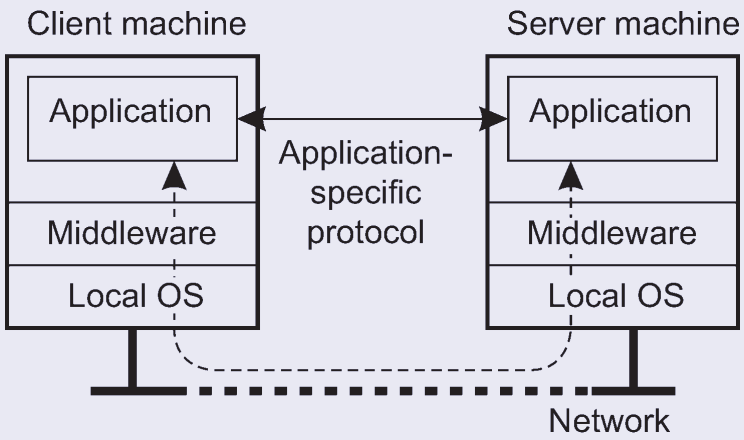
A máquina cliente é usada apenas como um terminal, sem armazenamento local



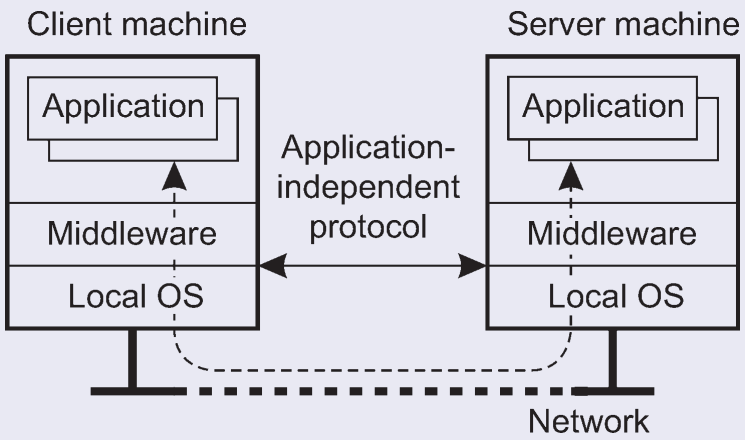
Interfaces de usuários em rede

- Há 2 modos de se implementar essa interação

Solução em nível de aplicação



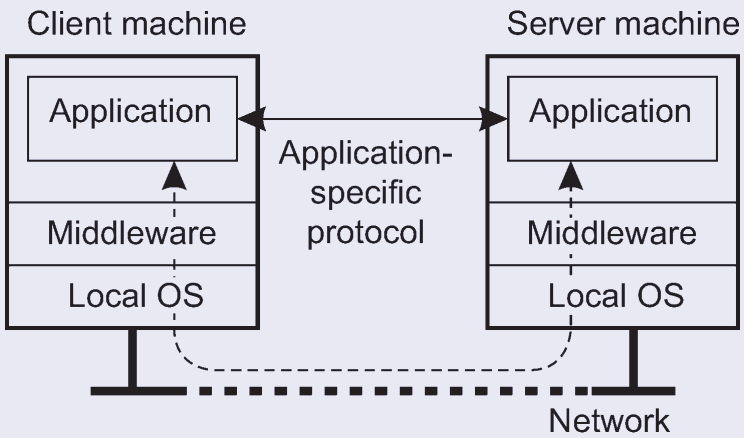
Abordagem cliente magro: tudo é processado e armazenado no servidor



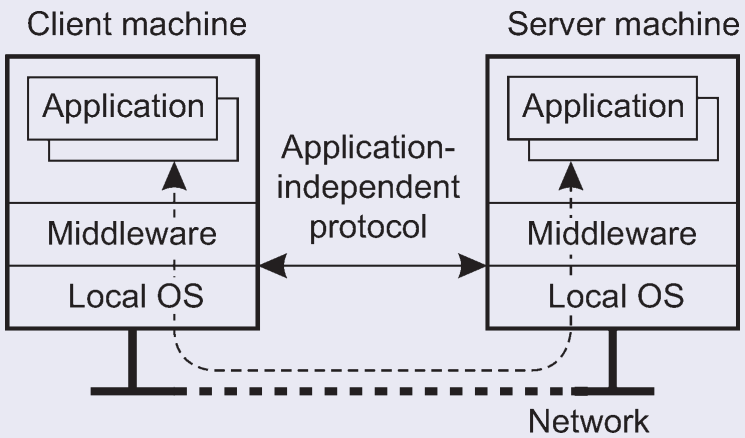
Interfaces de usuários em rede

- Há 2 modos de se implementar essa interação

Solução em nível de aplicação



A sincronização é feita por um **protocolo independente de aplicação**, no nível do middleware

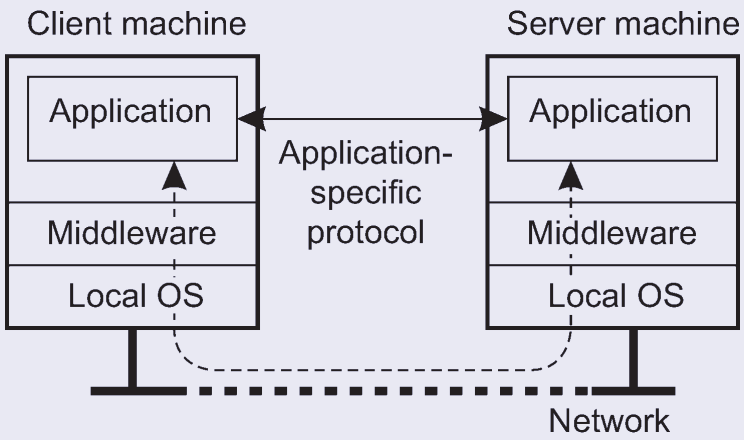


Clientes

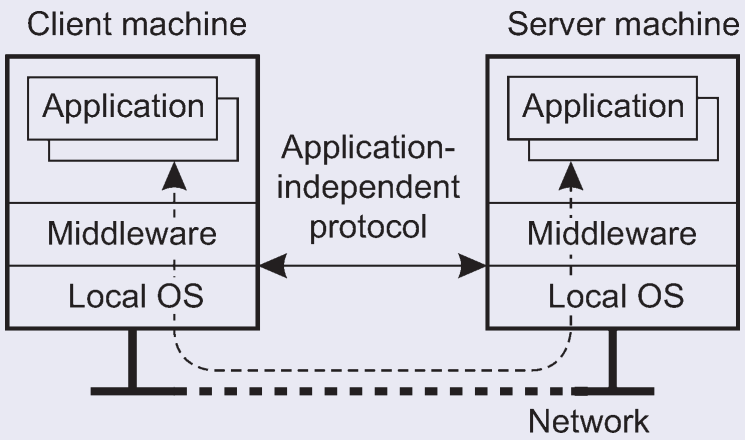
Interfaces de usuários em rede

- Há 2 modos de se implementar essa interação

Solução em nível de aplicação



Solução em nível de *middleware*



Clientes

Geralmente adaptado para transp. de distribuição

- Transparência de acesso
  - Obtida via *stubs* do cliente para RPC

Clientes

Geralmente adaptado para transp. de distribuição

- Transparência de acesso
  - Obtida via *stubs* do cliente para RPC
- Transparência de localização/migração/relocalização
  - Deixe o *middleware* do cliente manter o controle sobre a localização atual

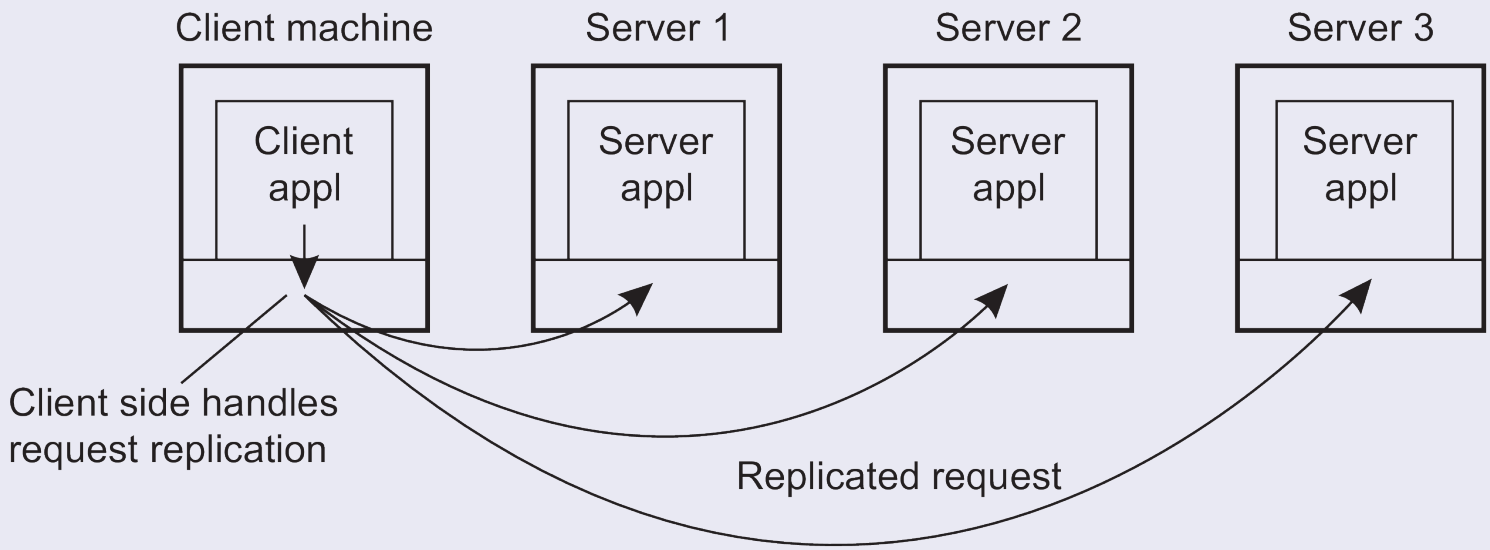
Clientes

Geralmente adaptado para transp. de distribuição

- Transparência de acesso
  - Obtida via *stubs* do cliente para RPC
- Transparência de localização/migração/relocalização
  - Deixe o *middleware* do cliente manter o controle sobre a localização atual
- Transparência de falhas
  - Podem com frequência ser de responsabilidade só do cliente (que tenta esconder falhas de comunicação e do servidor)

Geralmente adaptado para transp. de distribuição

- Transparência de replicação
  - Múltiplas requisições são gerenciadas pelo *stub* do cliente, que coleta as respostas, passando uma única resposta à aplicação cliente



Clientes, Servidores e Migração de Código

- Clientes
- **Servidores**
- Migração de Código

Servidores

Organização geral

- Um servidor é 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 seja tratada e, em seguida, passa a esperar pela próxima requisição

Dois tipos básicos

- **Servidores iterativos**
  - O próprio servidor trata da requisição, retornando a resposta ao cliente
  - Cada requisição é tratada antes de atender a próxima

Servidores

Dois tipos básicos

- **Servidores iterativos**
  - O próprio servidor trata da requisição, retornando a resposta ao cliente
  - Cada requisição é tratada antes de atender a próxima
- **Servidores concorrentes**
  - Não trata da requisição, usa um despachante (*dispatcher*), que pega a requisição e repassa seu tratamento a uma *thread*/processo separado, esperando então pela próxima requisição

Servidores

Contatando um servidor

- Como clientes contatam um servidor?
  - Eles enviam requisições a portas (ou *end points*) na máquina do servidor. Cada servidor escuta uma porta específica

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)

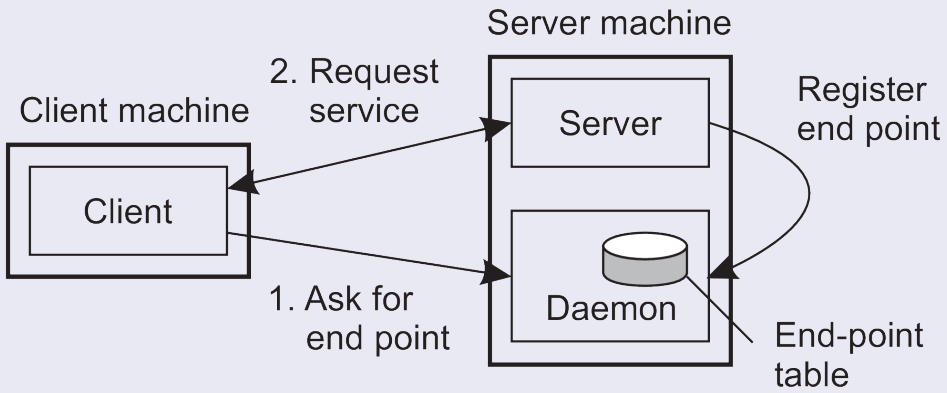
Servidores

Contatando um servidor

- A associação de portas a serviços pode ser dinâmica

Contatando um servidor

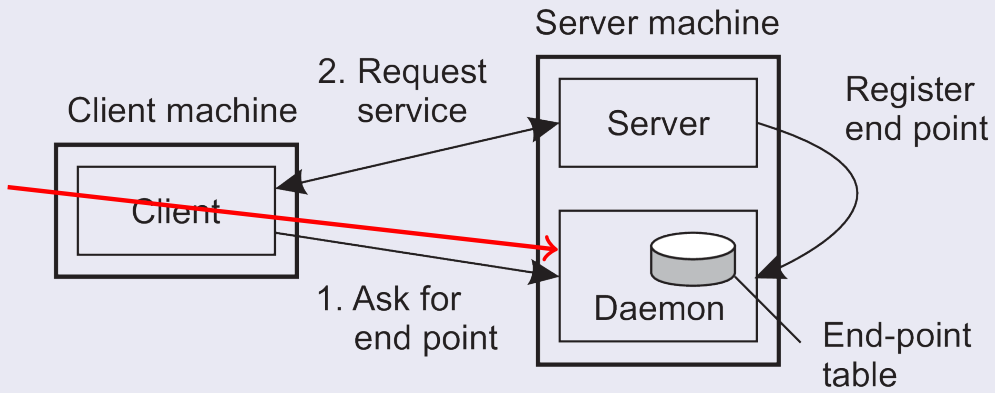
- A associação de portas a serviços pode ser dinâmica



Contatando um servidor

- A associação de portas a serviços pode ser dinâmica

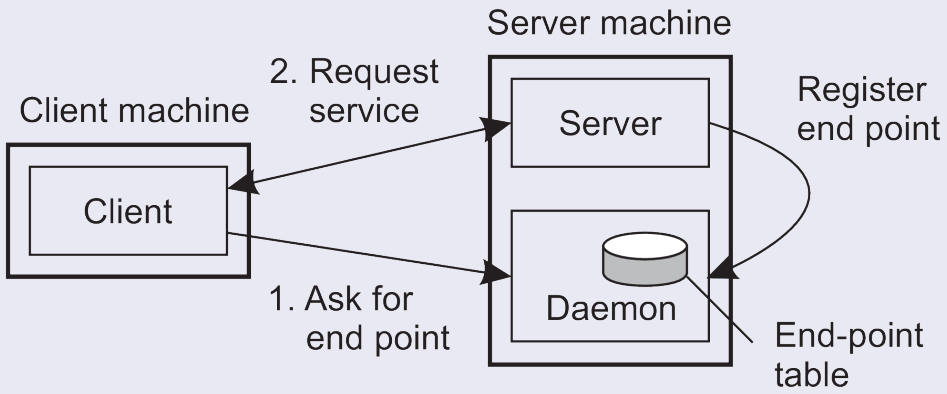
Um daemon rastreia a porta de cada serviço implementado por servidores na mesma máquina



Contatando um servidor

- A associação de portas a serviços pode ser dinâmica

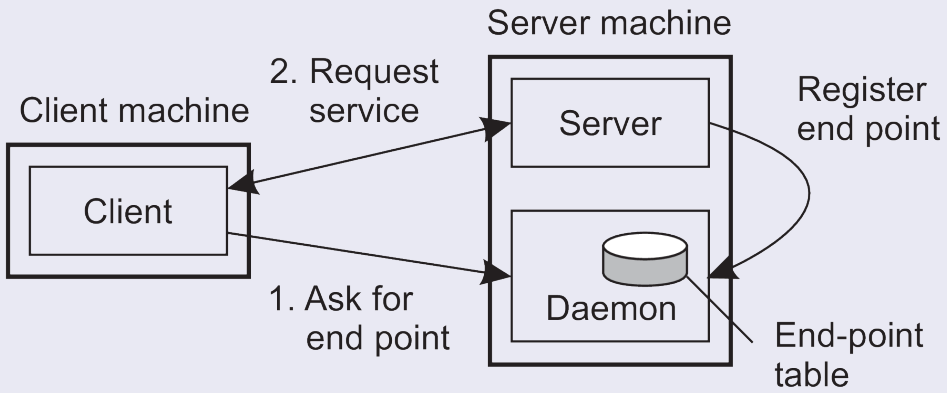
O daemon em si escuta uma porta bem definida



Contatando um servidor

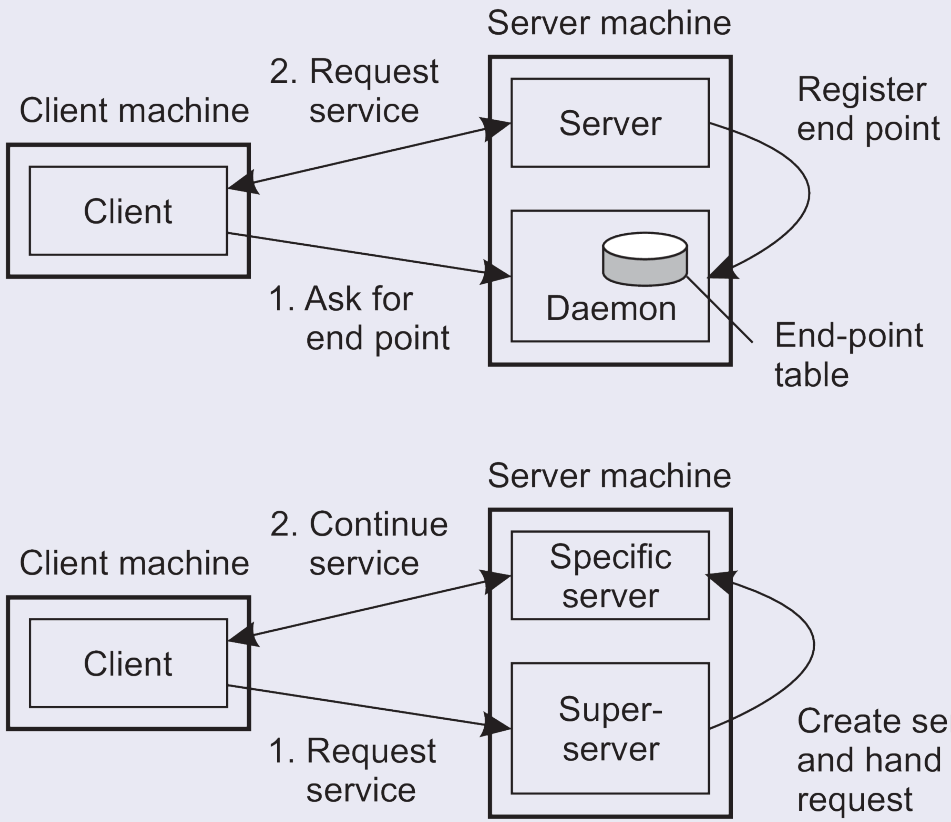
- A associação de portas a serviços pode ser dinâmica

O cliente primeiro contata o daemon, pedindo a porta do serviço, e então contacta o servidor específico



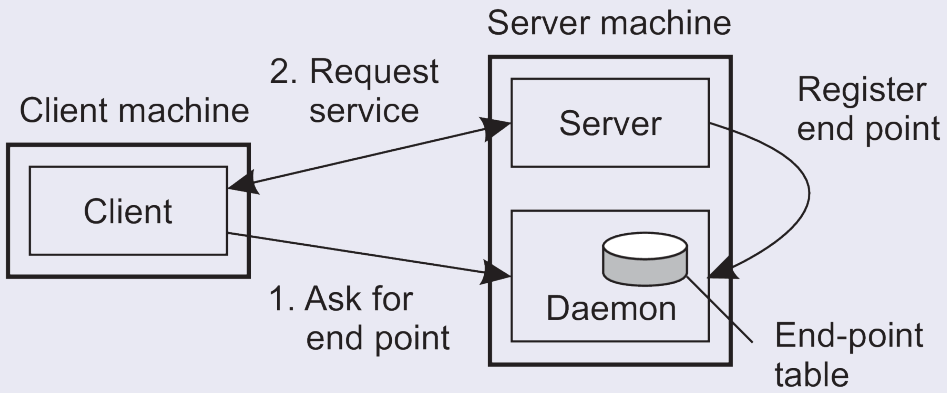
Contatando um servidor

- A associação de portas a serviços pode ser dinâmica

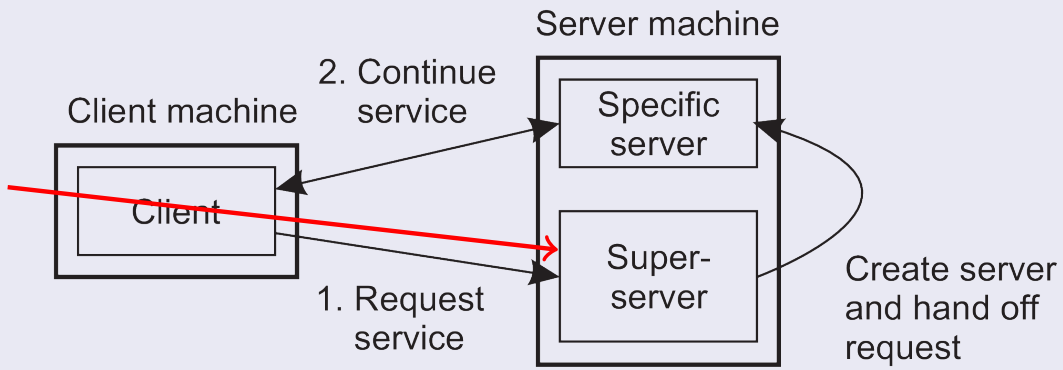


Contatando um servidor

- A associação de portas a serviços pode ser dinâmica

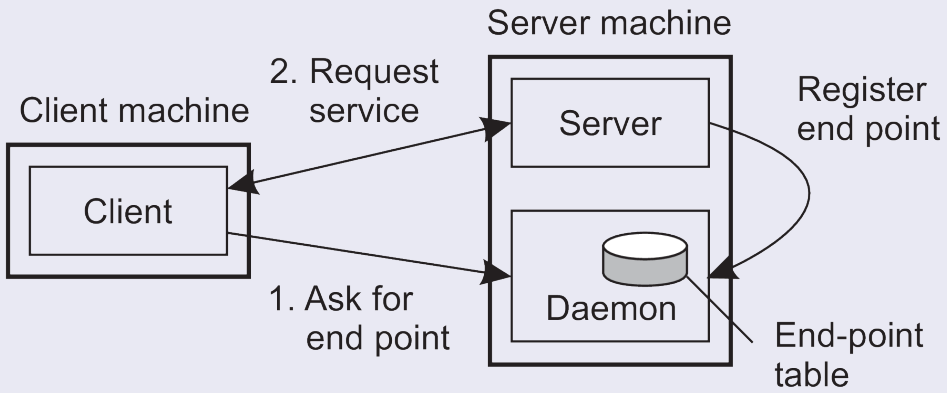


Alternativamente, usamos um **superserver**, escutando todas as portas associadas a serviços (ex: UNIX inetd)

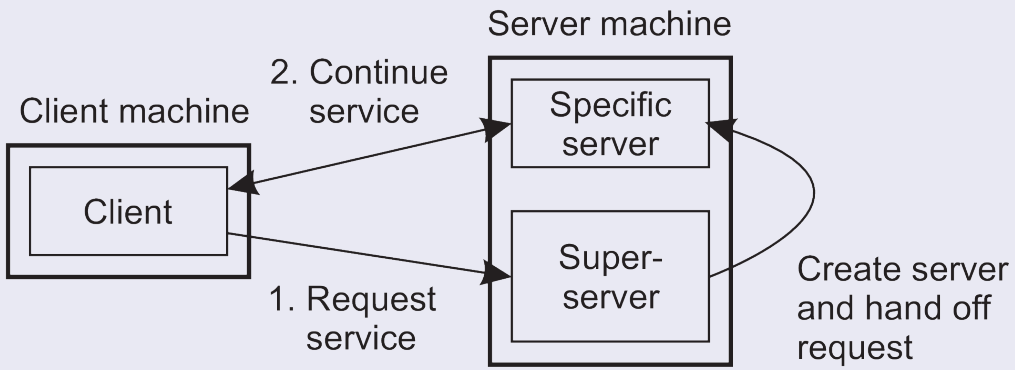


Contatando um servidor

- A associação de portas a serviços pode ser dinâmica

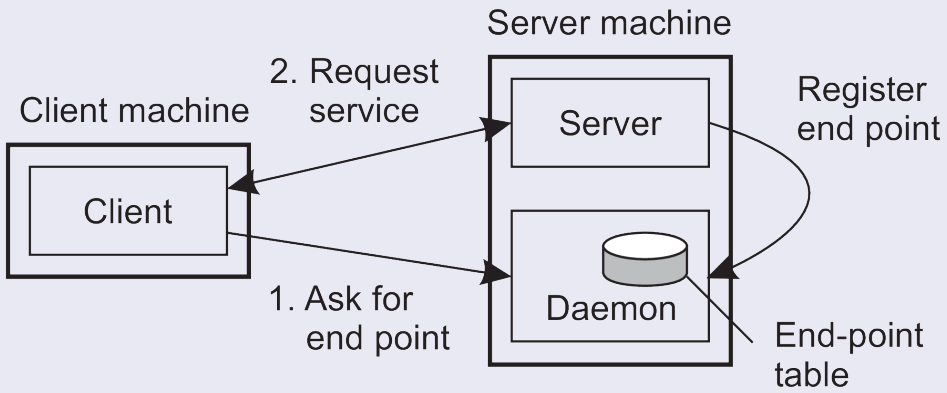


Quando uma requisição chega, o superservidor cria um subprocesso (*fork*) para tratá-la

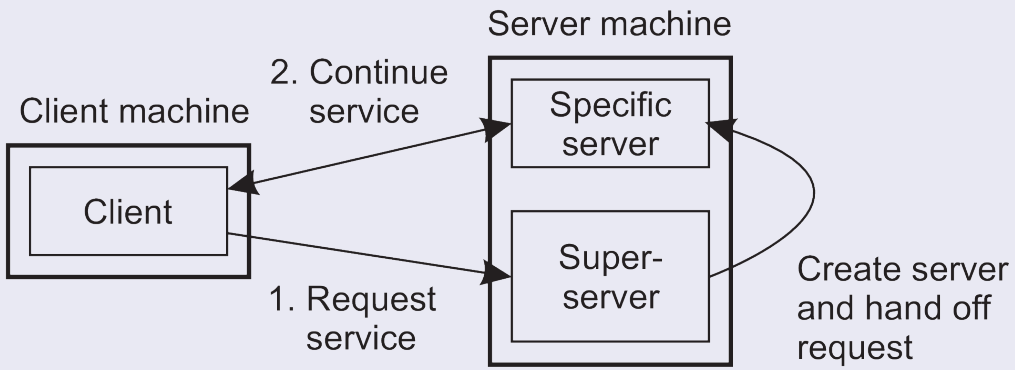


Contatando um servidor

- A associação de portas a serviços pode ser dinâmica



Esse subprocesso é então fechado, quando terminar de atender a requisição



Servidores

Interrompendo um servidor

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

Servidores

Interrompendo um servidor

- É possível interromper um servidor uma vez que ele já tiver aceitado (ou estiver processando) uma requisição de serviço?
- Solução 1: o usuário termina a aplicação cliente, interrompendo a conexão
  - Ele então a reinicia, fingindo que nada aconteceu
  - O servidor irá destruir a conexão antiga, pensando que o cliente travou

Interrompendo um servidor

- Solução 2: permitir que se envie *out-of-band data*
  - Dados que devem ser processados pelo servidor antes de qualquer outro dado do cliente

Servidores

Interrompendo um servidor

- Solução 2: permitir que se envie *out-of-band data*
  - Dados que devem ser processados pelo servidor antes de qualquer outro dado do cliente
- Possibilidade 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

Servidores

Interrompendo um servidor

- Possibilidade 2: enviar os dados *out-of-band* na mesma conexão da requisição, usando as facilidades da camada de transporte
  - TCP permite o envio de mensagens urgentes na mesma conexão
  - Mensagens urgentes podem ser capturadas (via interrupção) usando tratamento de sinais do SO

Servidores e estado

Servidores *stateless*

- Não mantém informação 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 e estado

Servidores *stateless*

- Não mantém informação 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
- Alguns mantêm informação do cliente
  - Mas a perda da informação não leva à interrupção do serviço

Servidores e estado

Servidores *stateless*

- 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* blocos de arquivo)

Servidores e estado

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 e estado

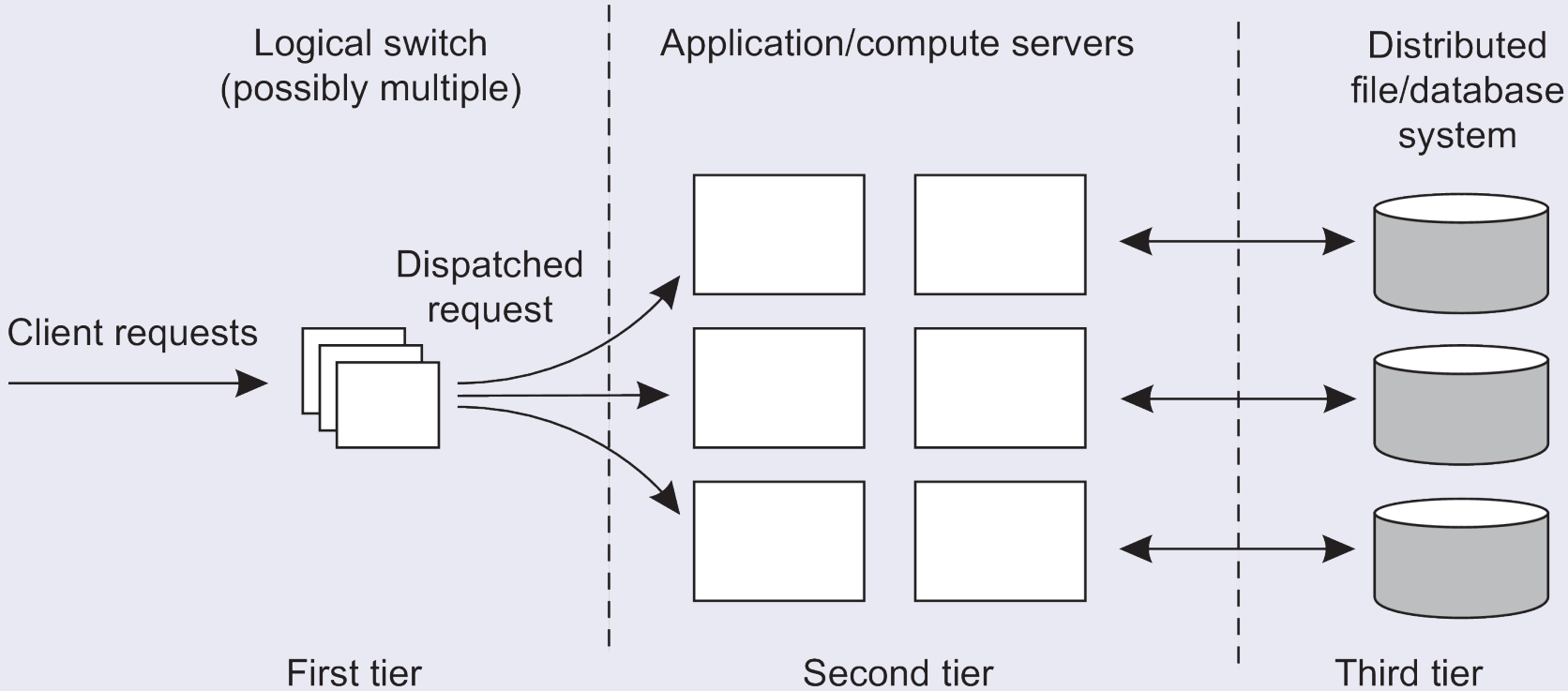
Servidores com estado (*stateful*)

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

Aglomerados de servidores (*clusters*)

Três camadas diferentes

- Em geral, um *cluster* é organizado em 3 camadas:

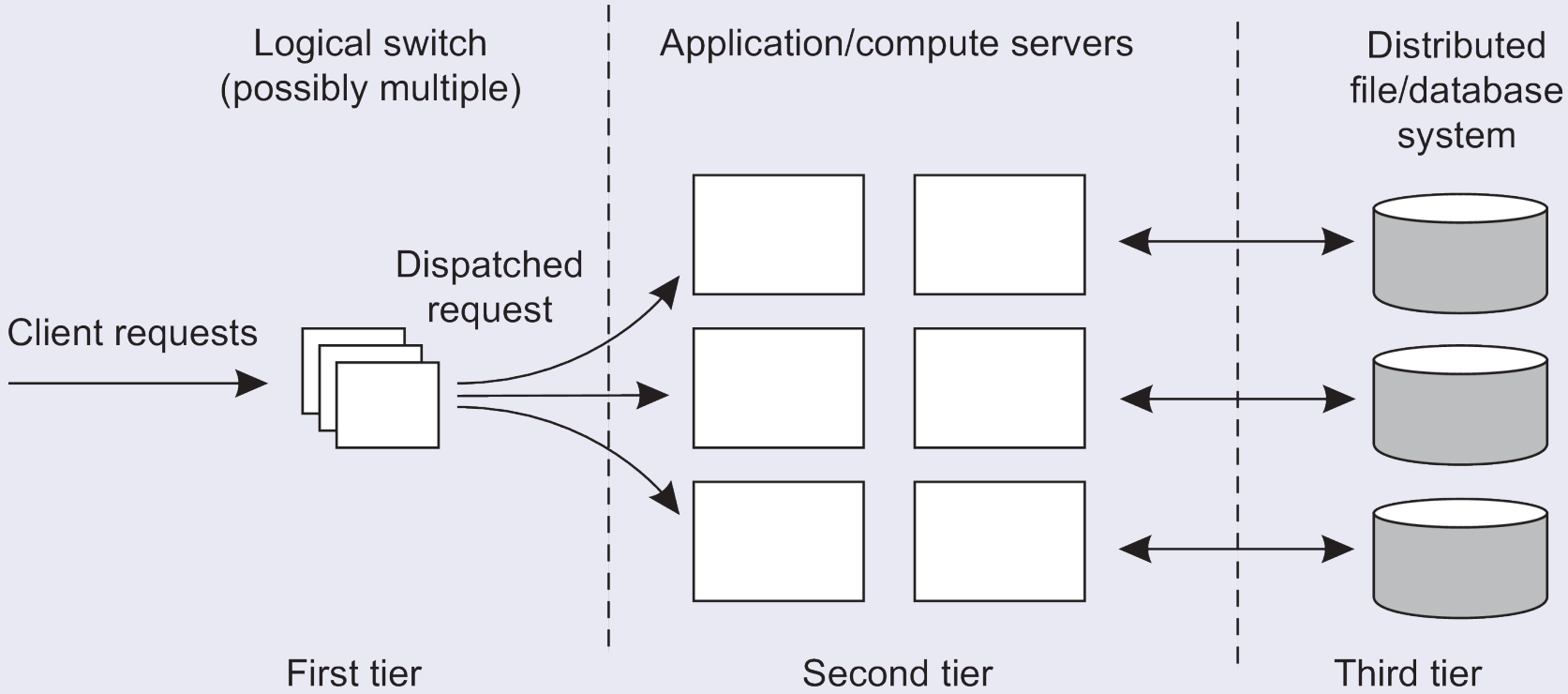


Aglomerados de servidores (*clusters*)

Três camadas diferentes

- Em geral, um *cluster* é organizado em 3 camadas:

A primeira camada (*front end*) é responsável por repassar as requisições para um servidor apropriado

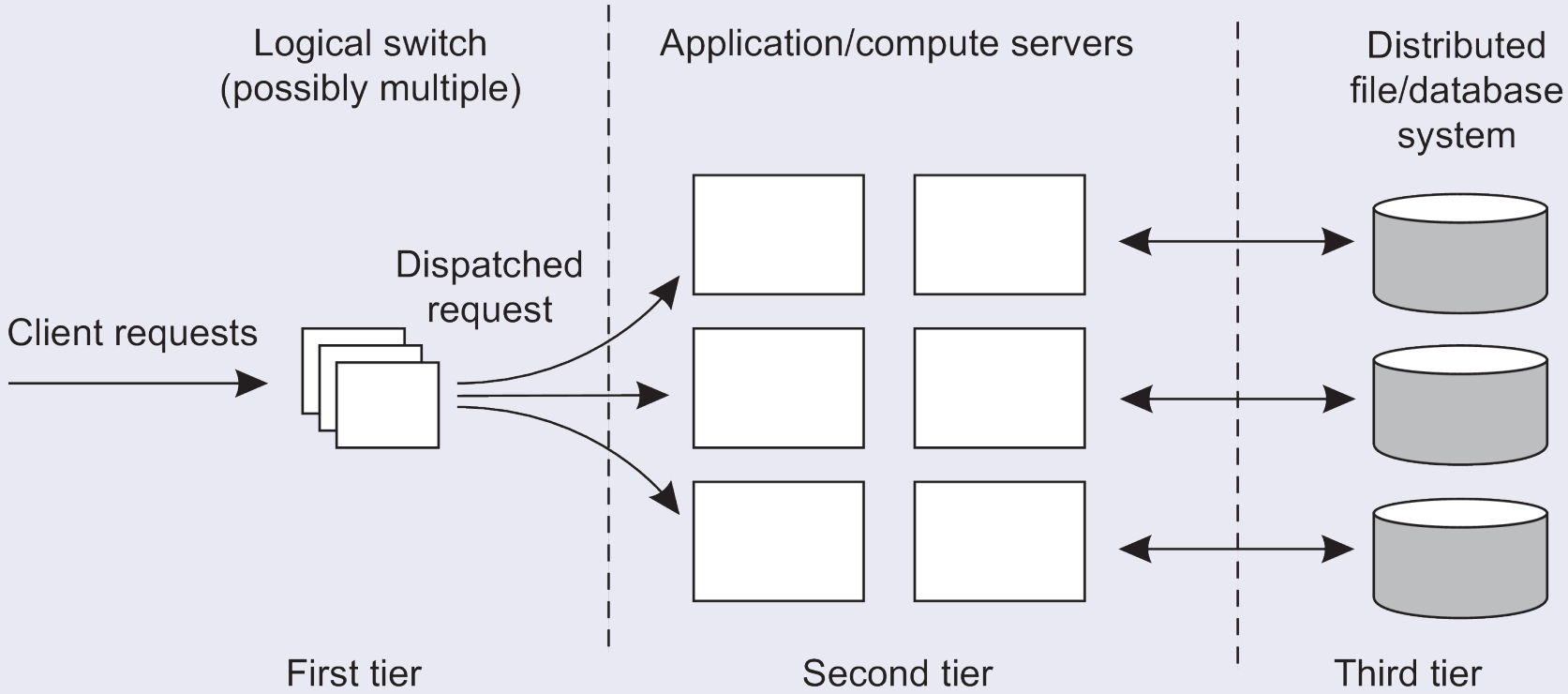


Aglomerados de servidores (*clusters*)

Três camadas diferentes

- Em geral, um *cluster* é organizado em 3 camadas:

A terceira camada consiste de servidores para processamento dos dados



Aglomerados de servidores (*clusters*)

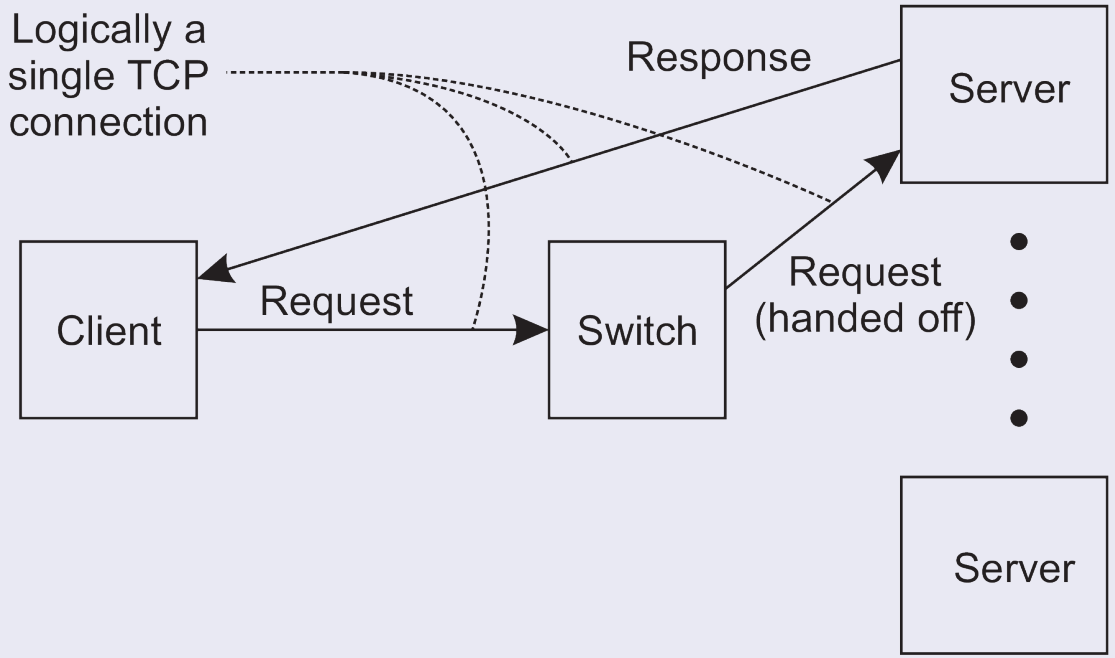
Tratamento de requisições

- Ter uma única camada tratando toda a comunicação externa pode levar a um gargalo

Aglomerados de servidores (*clusters*)

Tratamento de requisições

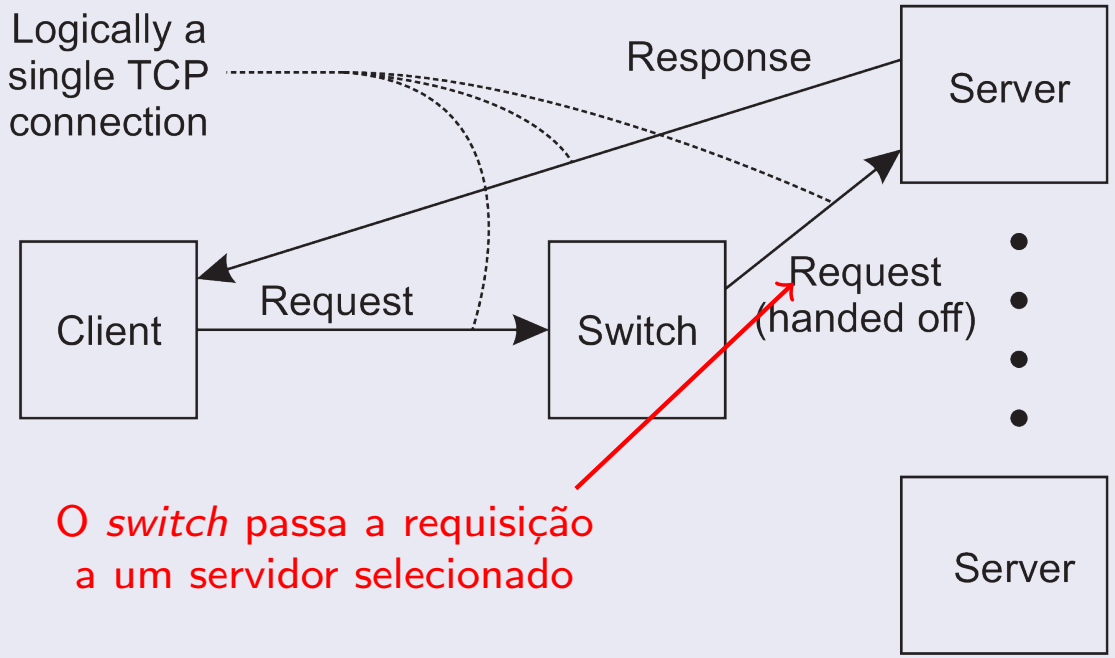
- Ter uma única camada tratando toda a comunicação externa pode levar a um gargalo
  - Solução: *TCP handoff*



Aglomerados de servidores (*clusters*)

Tratamento de requisições

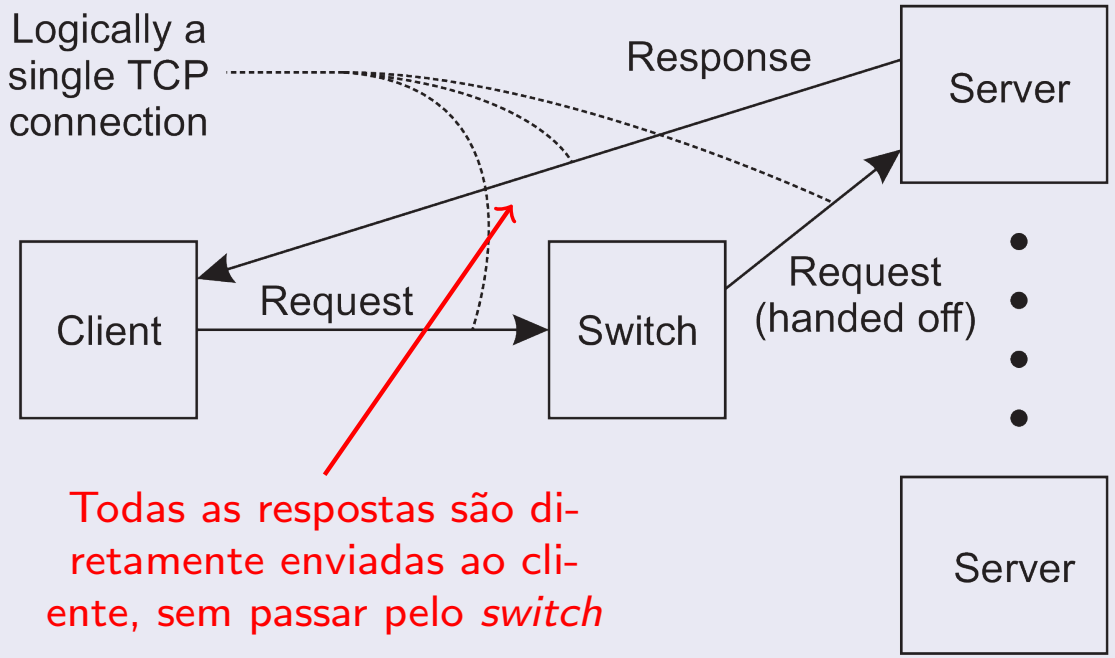
- Ter uma única camada tratando toda a comunicação externa pode levar a um gargalo
  - Solução: *TCP handoff*



Aglomerados de servidores (*clusters*)

Tratamento de requisições

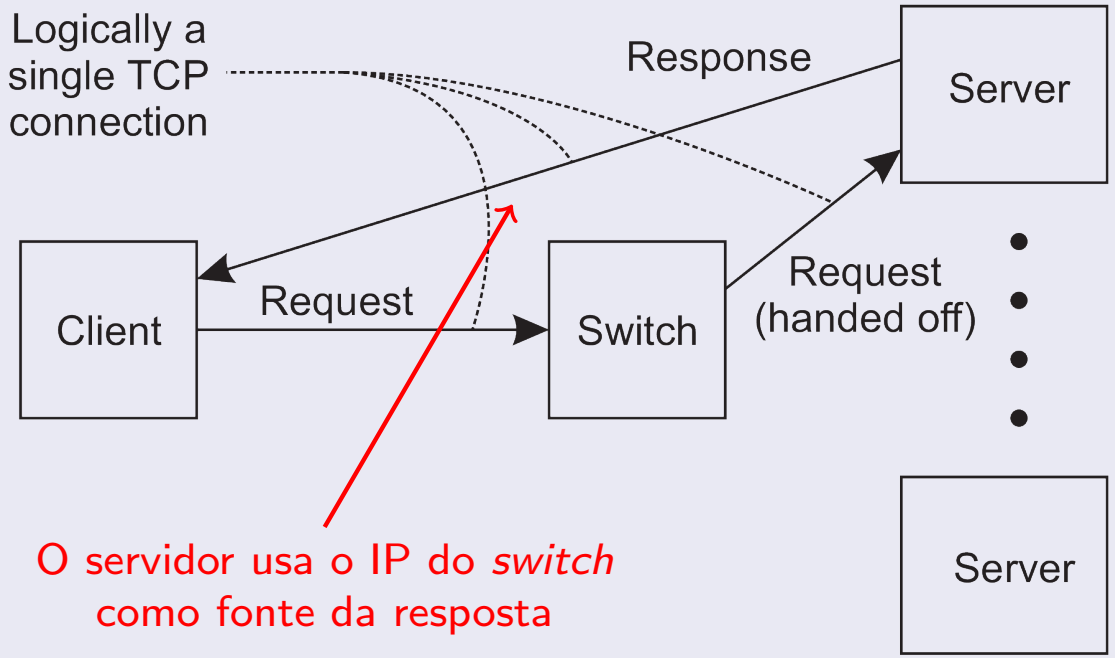
- Ter uma única camada tratando toda a comunicação externa pode levar a um gargalo
  - Solução: *TCP handoff*



Aglomerados de servidores (*clusters*)

Tratamento de requisições

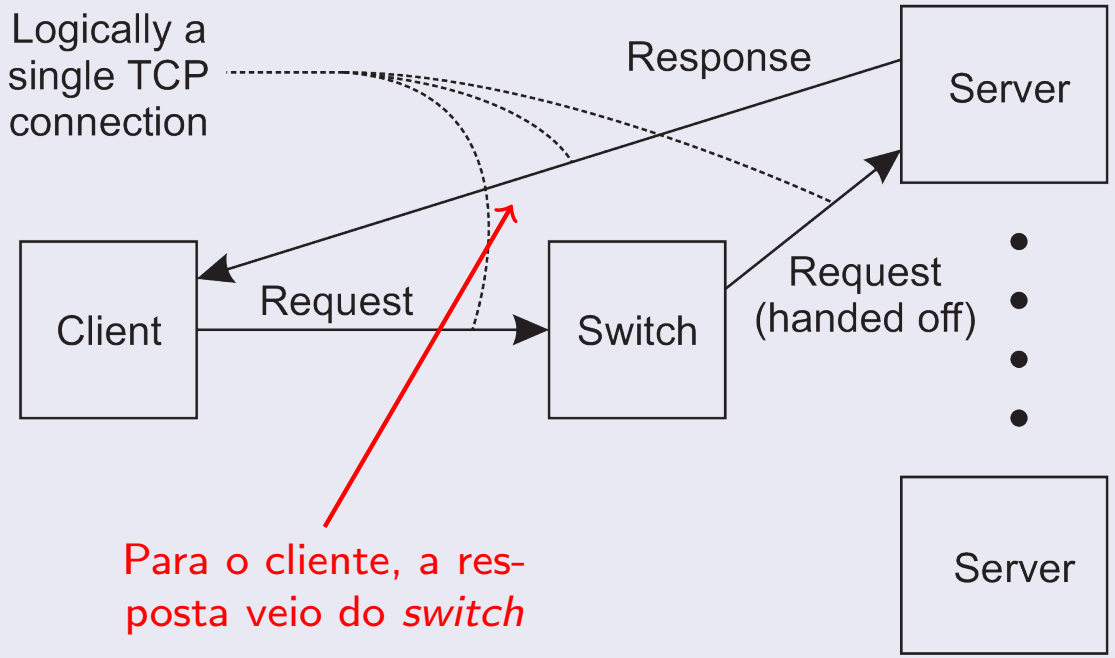
- Ter uma única camada tratando toda a comunicação externa pode levar a um gargalo
  - Solução: *TCP handoff*



Aglomerados de servidores (*clusters*)

Tratamento de requisições

- Ter uma única camada tratando toda a comunicação externa pode levar a um gargalo
  - Solução: *TCP handoff*



Clientes, Servidores e Migração de Código

- Clientes
- Servidores
- **Migração de Código**

Migração de Código

Definição

- Trata-se de passar programas inteiros de uma máquina a outra (por vezes, mesmo enquanto estão executando)
  - Tradicionalmente, movendo processos inteiros de um nó a outro (*process migration*)

Migração de Código

Definição

- Trata-se de passar programas inteiros de uma máquina a outra (por vezes, mesmo enquanto estão executando)
  - Tradicionalmente, movendo processos inteiros de um nó a outro (*process migration*)
- Ex: agentes móveis

Migração de Código

Razões para migrar código

- Melhora do desempenho geral
  - Ao movermos processos de máquinas muito carregadas para pouco carregadas

Migração de Código

Razões para migrar código

- Melhora do desempenho geral
  - Ao movermos processos de máquinas muito carregadas para pouco carregadas
- Reduzir a comunicação
  - Garantindo que a computação é feita próximo de onde o dado está
  - Ex: enviar parte da aplicação cliente ao servidor, enviando apenas os resultados de volta

Migração de Código

Razões para migrar código

- Tradicionalmente, aplicações distribuídas são construídas separando-as em partes distintas
  - Decidindo de antemão onde cada parte deve ser executada

Migração de Código

Razões para migrar código

- Tradicionalmente, aplicações distribuídas são construídas separando-as em partes distintas
  - Decidindo de antemão onde cada parte deve ser executada

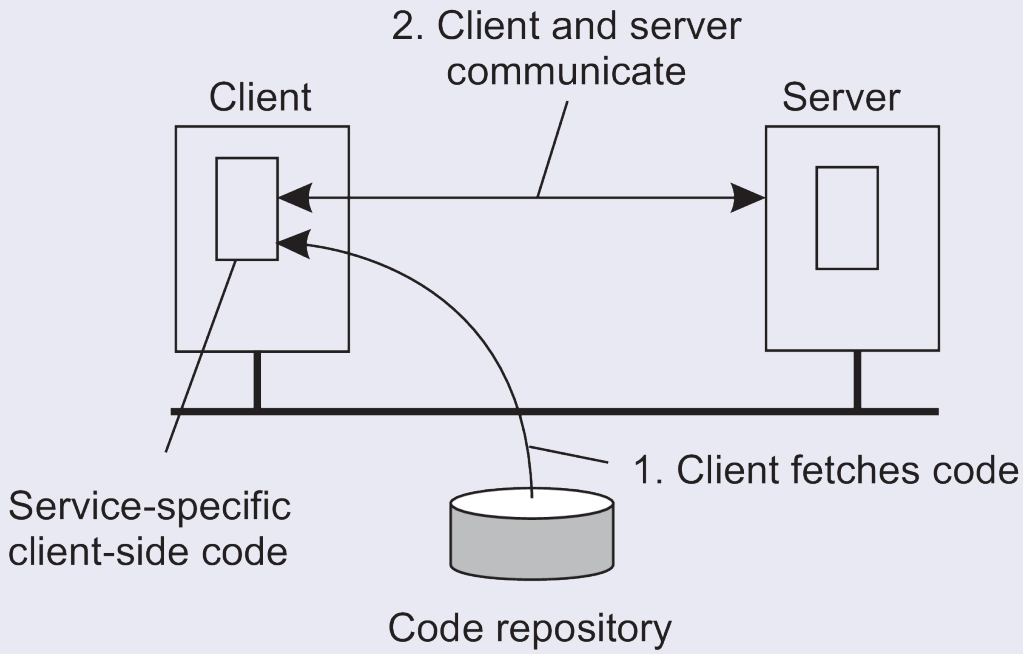
Contudo, se pudermos mover código entre máquinas diferentes, podemos configurar o sistema dinamicamente

Migração de Código

Razões para migrar código

- Tradicionalmente, aplicações distribuídas são construídas separando-as em partes distintas
  - Decidindo de antemão onde cada parte deve ser executada

Ex: o servidor pode fornecer o código de que o cliente precisa apenas quando for necessário

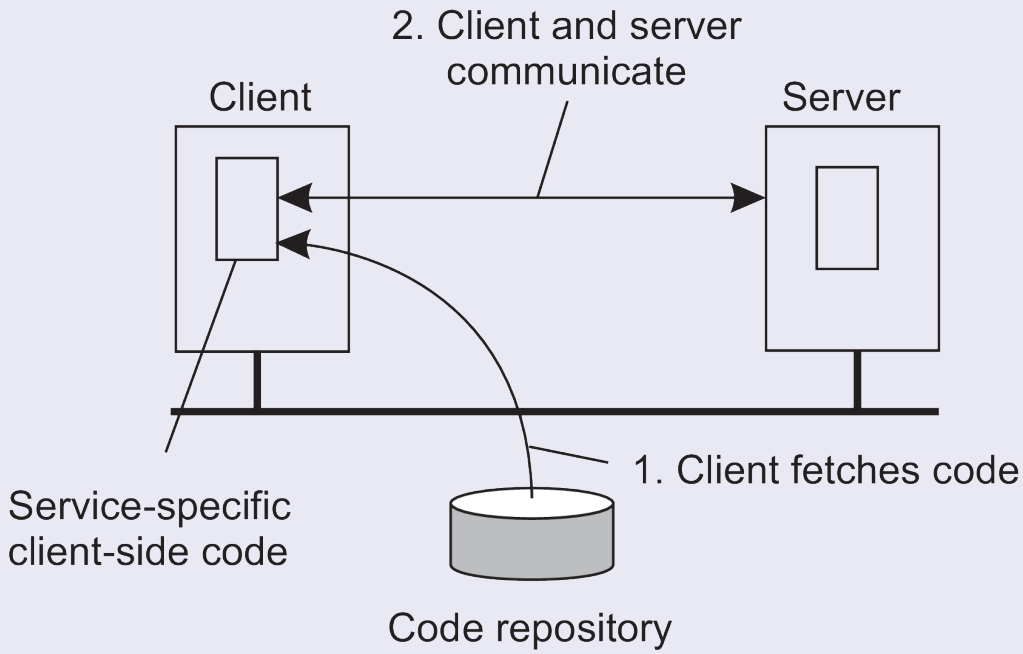


Migração de Código

Razões para migrar código

- Tradicionalmente, aplicações distribuídas são construídas separando-as em partes distintas
  - Decidindo de antemão onde cada parte deve ser executada

O cliente não precisa assim ter todo o software pré-instalado para conversar com o servidor

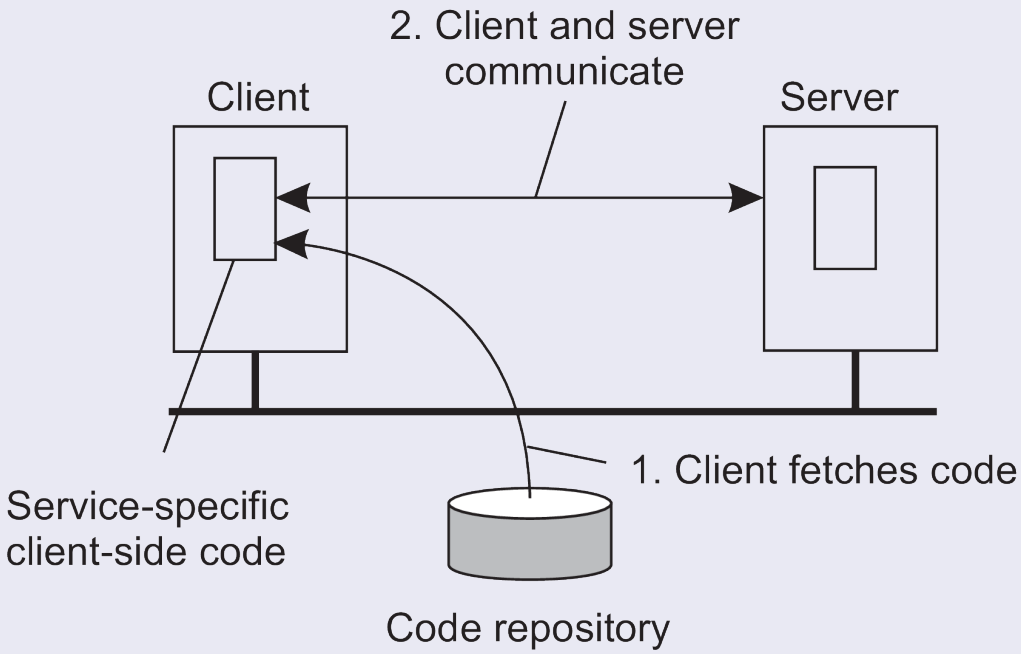


Migração de Código

Razões para migrar código

- Tradicionalmente, aplicações distribuídas são construídas separando-as em partes distintas
  - Decidindo de antemão onde cada parte deve ser executada

Isso, contudo, exige que se confie cegamente que o código baixado implementa apenas o que se deseja



Migração de Código

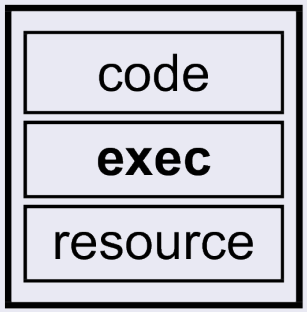
Modelos de migração

- Migração iniciada pelo emissor
  - A migração é iniciada na máquina onde o código reside ou está sendo executado
- Migração iniciada pelo receptor
  - A iniciativa para a migração é tomada pela máquina alvo

Migração de Código

Modelos de migração

- Migração iniciada pelo emissor
  - A migração é iniciada na máquina onde o código reside ou está sendo executado
- Migração iniciada pelo receptor
  - A iniciativa para a migração é tomada pela máquina alvo
- Nesse *framework*, um processo consiste de 3 segmentos:

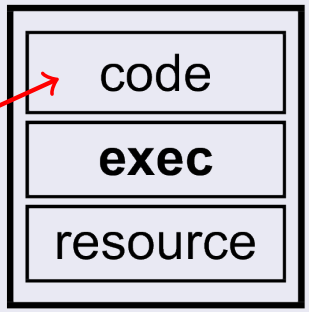


Migração de Código

Modelos de migração

- Migração iniciada pelo emissor
  - A migração é iniciada na máquina onde o código reside ou está sendo executado
- Migração iniciada pelo receptor
  - A iniciativa para a migração é tomada pela máquina alvo
- Nesse *framework*, um processo consiste de 3 segmentos:

Segmento de código, contendo as instruções do programa

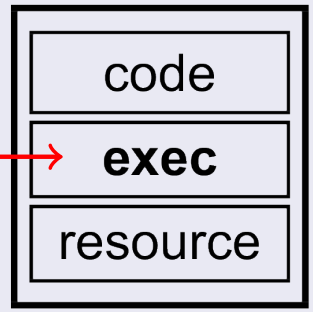


Migração de Código

Modelos de migração

- Migração iniciada pelo emissor
  - A migração é iniciada na máquina onde o código reside ou está sendo executado
- Migração iniciada pelo receptor
  - A iniciativa para a migração é tomada pela máquina alvo
- Nesse *framework*, um processo consiste de 3 segmentos:

Segmento de execução, armazenando o estado atual de execução do processo (dados privados, pilha, e *program counter*)

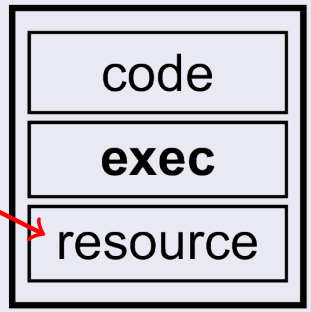


Migração de Código

Modelos de migração

- Migração iniciada pelo emissor
  - A migração é iniciada na máquina onde o código reside ou está sendo executado
- Migração iniciada pelo receptor
  - A iniciativa para a migração é tomada pela máquina alvo
- Nesse *framework*, um processo consiste de 3 segmentos:

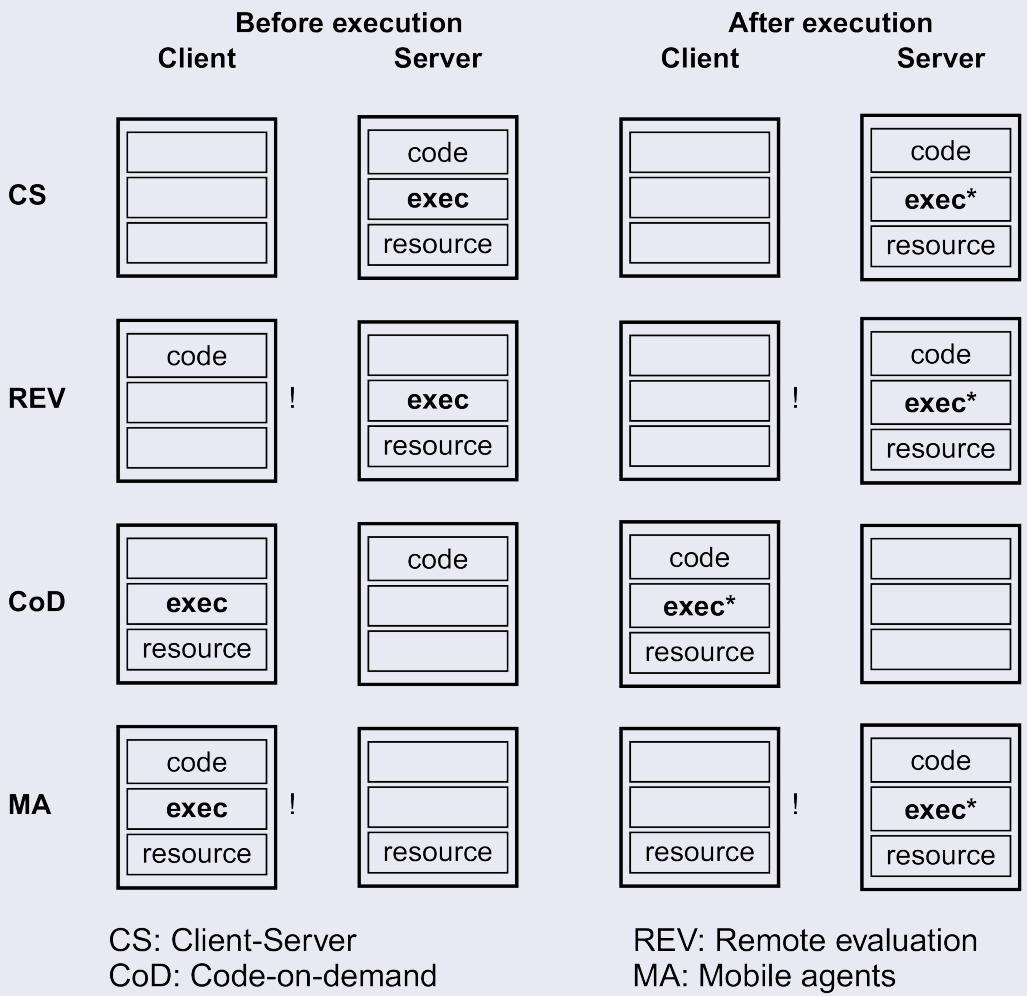
Segmento de recursos, contendo referências aos recursos externos necessários ao processo



Migração de Código

Paradigmas de migração

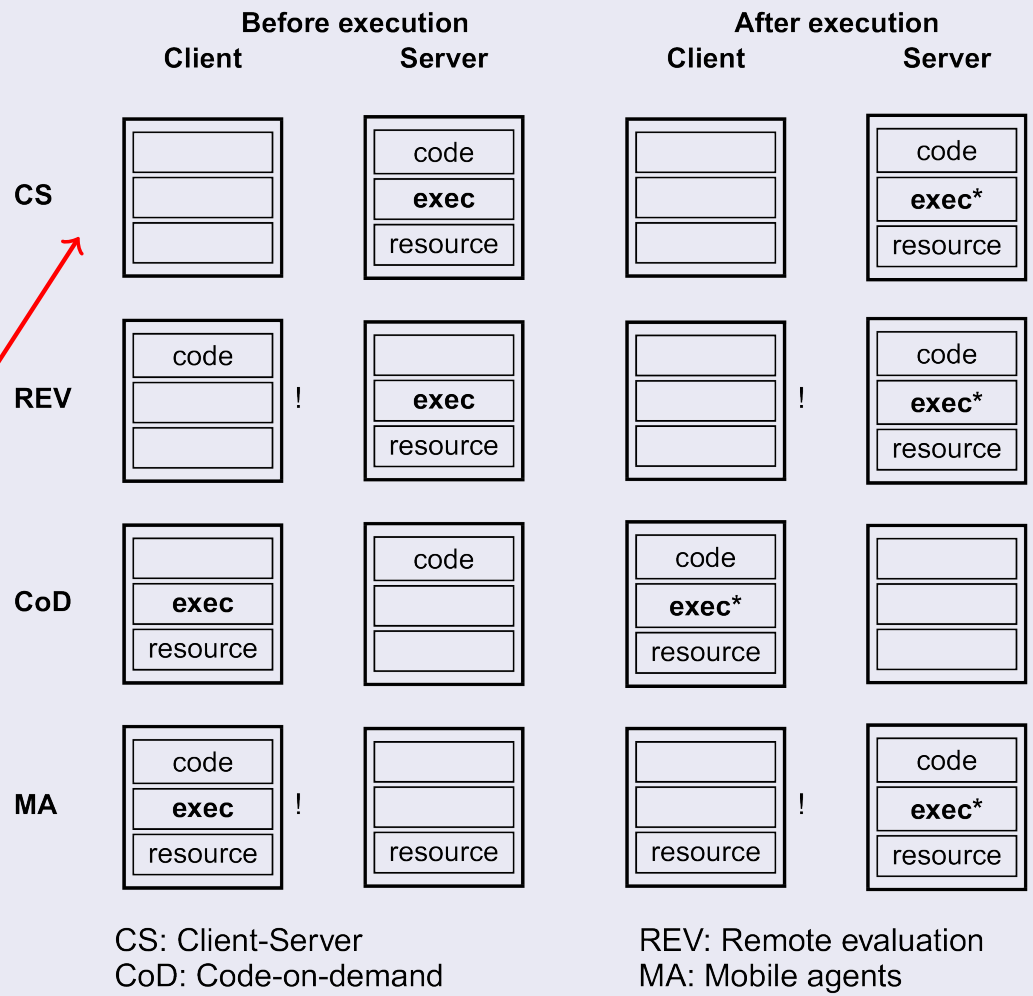
Temos então 4 paradigmas para mobilidade de código



Migração de Código

Paradigmas de migração

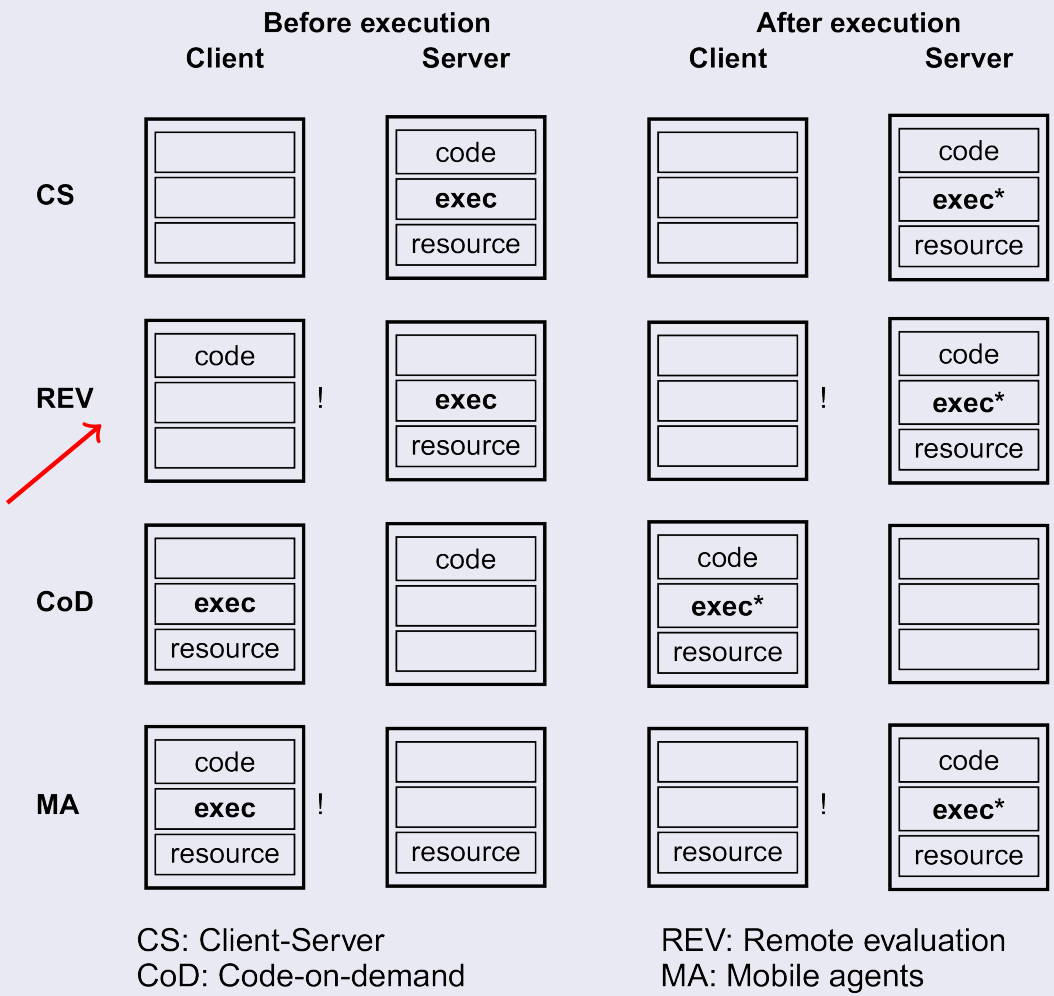
Cliente-Servidor: os 3 segmentos estão no servidor. Com sua execução, apenas o estado de execução é modificado no servidor



Migração de Código

Paradigmas de migração

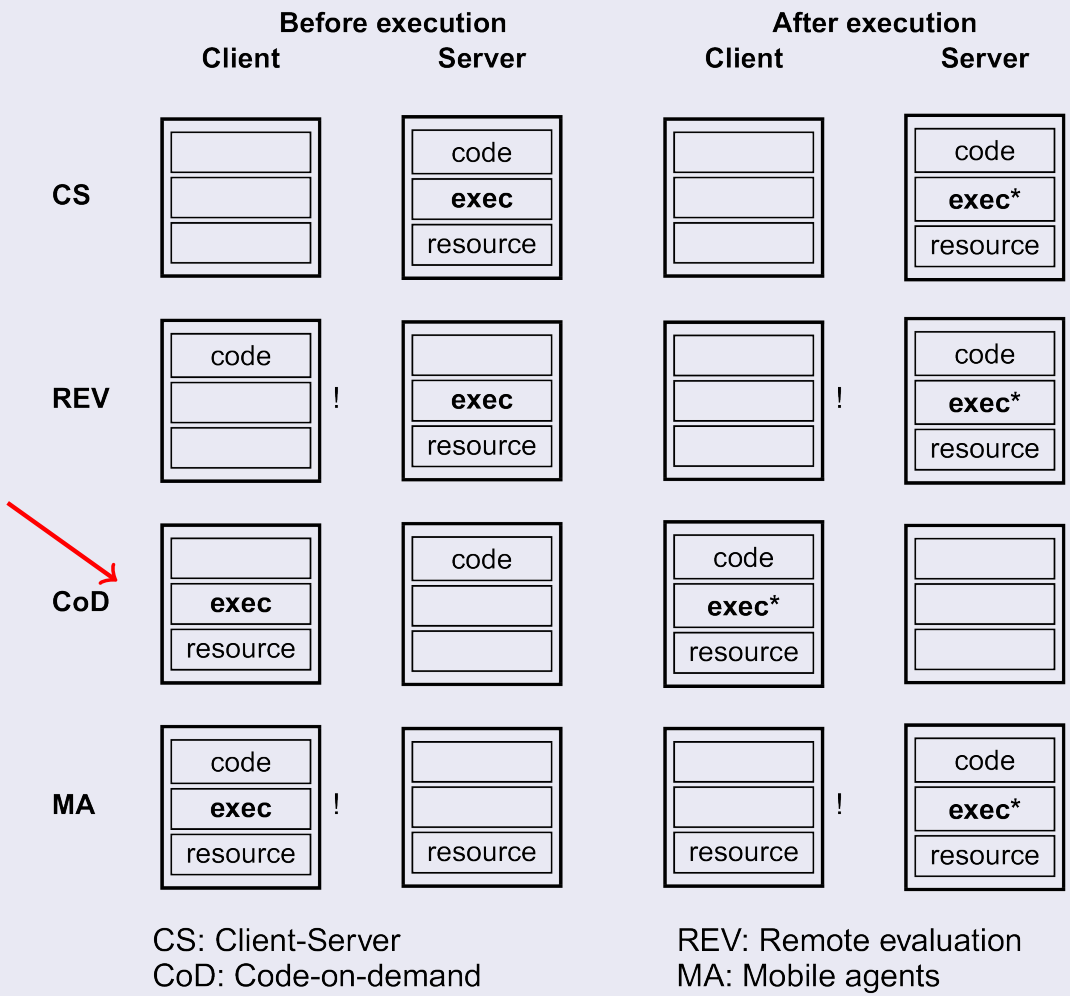
Remote evaluation: iniciada pelo emissor, o cliente migra o código para o servidor, onde ele é executado. Com sua execução, o estado de execução é modificado no servidor



Migração de Código

Paradigmas de migração

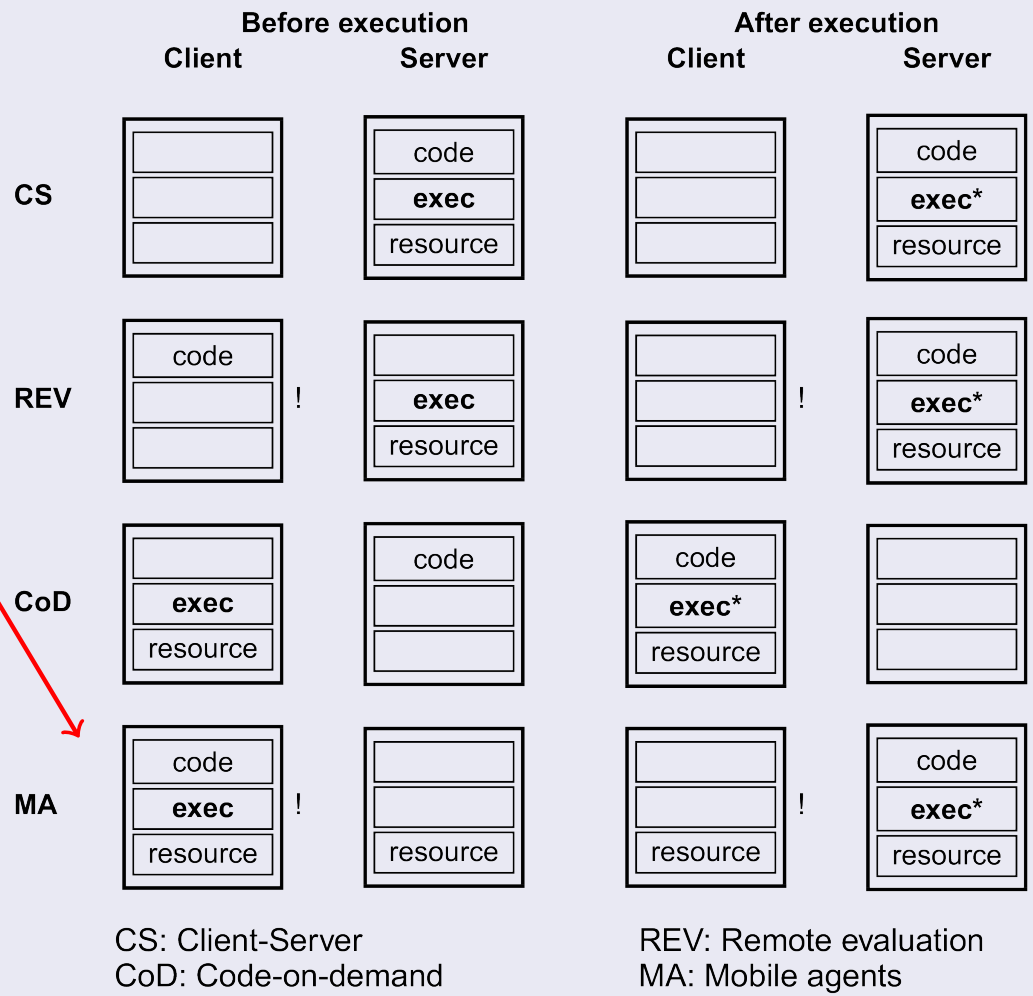
Code-on-demand: esquema iniciado no receptor (cliente), em que o cliente obtém o código. Sua execução modifica o estado de execução no lado do cliente, operando nos recursos do cliente



Migração de Código

Paradigmas de migração

Agentes móveis: tipicamente iniciados pelo emissor, movendo código e estado de execução do cliente para o servidor, operando tanto nos recursos do cliente quanto do servidor. Com sua execução, o estado de execução é modificado no servidor



Migração de Código

Mobilidade forte e fraca

- Mobilidade fraca
  - Transfere apenas o segmento de código, possivelmente junto com alguns dados de inicialização
  - **Característica principal**: Um programa transferido é iniciado novamente
  - Relativamente simples, especialmente se o código é portátil
  - Duas modalidades: envio de código (*push*) e busca de código (*pull*)
  - Ex: Applets Java

Migração de Código

Mobilidade forte e fraca

- Mobilidade forte
  - O segmento de execução pode ser transferido também
  - **Característica principal**: um processo em execução deve poder ser parado, subsequentemente movido a outra máquina, e então retomar sua execução exatamente de onde parou
  - Migração de prcesso: move um processo em execução
  - Clonagem: inicia um clone do processo original e o configura para o mesmo estado de execução

Migração de Código

Migração em sistemas heterogêneos

- 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
- Solução
  - Usar uma máquina abstrata implementada nas diferentes plataformas:
    - Linguagens interpretadas, que possuem suas próximas MVs
    - Monitores de MV

Migração de Código

Migrando uma máquina virtual

- Uma alternativa é migrar a própria máquina virtual
  - Não somente processos

Migração de Código

Migrando uma máquina virtual

- Uma alternativa é migrar a própria máquina virtual
  - Não somente processos
- Vantagem: os processos ignoram a migração
  - Não precisam ser interrompidos
  - Não experimentam problemas com os recursos usados

Migração de Código

Migrando uma máquina virtual

- Uma alternativa é migrar a própria máquina virtual
  - Não somente processos
- Vantagem: os processos ignoram a migração
  - Não precisam ser interrompidos
  - Não experimentam problemas com os recursos usados
- Problemas:
  - Migração da imagem de memória inteira
  - Migração dos vínculos a recursos locais

Migração de Código

Migrando uma MV – Migração de imagem

- Três alternativas

Migração de Código

Migrando uma MV – Migração de imagem

- Três alternativas
  - Enviar as páginas de memória para a nova máquina e reenviar aquelas que forem modificadas durante o processo de migração

Migração de Código

Migrando uma MV – Migração de imagem

- Três alternativas
  - Enviar as páginas de memória para a nova máquina e reenviar aquelas que forem modificadas durante o processo de migração
  - Interromper a máquina virtual, migrar a memória, e iniciar a nova máquina virtual

Migração de Código

Migrando uma MV – Migração de imagem

- Três alternativas
  - Enviar as páginas de memória para a nova máquina e reenviar aquelas que forem modificadas durante o processo de migração
  - Interromper a máquina virtual, migrar a memória, e iniciar a nova máquina virtual
  - 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

Migrando uma MV – Desempenho

- Problema
  - Uma migração completa pode levar dezenas de segundos
  - Durante a migração, um serviço poderá ficar completamente indisponível por vários segundos

Tempo de resposta de uma VM durante uma migração