Sistemas Distribuídos

Prof. Sérgio T. Carvalho
sergio@inf.ufg.br

Modelos e
Arquiteturas de Sistemas Distribuídos

Modelos

- Descrição abstrata e simplificada de aspectos relevantes relacionados ao design de um sistema distribuído
  - Modelos Físicos
  - Modelos Arquiteturais
  - Modelos Fundamentais

Modelos Físicos (1)

- Representação dos elementos de hardware de um sistema
  - Computadores
  - Dispositivos (e.g. móveis)
  - Redes de interconexão
- Abstraem os detalhes das tecnologias computacionais e de redes

Modelos Físicos (2)

Primeiros sistemas

- 10 a 100 nodos em uma LAN
- servidores de arquivo e compartilhamento
- Sem preocupação com extensibilidade e QoS

- Internet-scale
  - Conjunto extensível de nodos (redes de redes)
  - Heterogeneidade
  - Ênfase em padrões abertos, middleware, web services,
  - Preocupação com QoS

Modelos Físicos (3)

Contemporâneo

- Antes: desktops relativamente estáticos, discretos e autônomos
- Desenvolvimento atual:
  - Computação móvel: laptops, smart phones movendo-se; novas demandas: serviço de descoberta; interoperação espontânea.
  - Computação ubíqua: de nodos discretos para arquiteturas onde computadores são embarcados em objetos do dia-a-dia (smart homes)
  - Cloud computing: de nodos autônomos para pools de nodos atuando em conjunto para prover um dado serviço.

Modelos Físicos (4)

- Systems of systems
  - e.g. smart homes

Modelos Físicos (5)

Figure 2.1 Generations of distributed systems

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

Arquitetura de sistemas distribuídos

- Arquitetura: estrutura de um sistema em termos de componentes especificados separadamente
  - alocação dos componentes na rede
  - inter-relacionamento dos componentes
- Preocupações: confiável, gerenciável, adaptável, relação custo-benefício
- Principais modelos (ou estilos) de arquitetura:
  - Cliente-servidor (processos ou objetos)
  - *Peer-to-peer*

Arquitetura de sistemas distribuídos

- Elementos da Arquitetura
  - Objetos
  - Componentes
  - Web services
- Modelos de comunicação
  - Comunicação interprocesso “primitiva”: message-passing, socket, multicast
  - Invocação remota: request-reply, RPC, RMI
  - Comunicação indireta: group communication, publish-subscribe, message-queues, tuple-spaces, distributed shared memory

Arquitetura de sistemas distribuídos

- Blocos básicos de construção de um sistema distribuído
  - Quais as entidades que estão se comunicando no sistema distribuído?
  - Como eles se comunicam, ou, mais especificamente, qual paradigma de comunicação é usado?
  - Quais papéis e responsabilidades eles têm na arquitetura como um todo?
  - Como eles são mapeados para a infraestrutura física distribuída?

Arquitetura de sistemas distribuídos

- Quais as entidades que estão se comunicando no sistema distribuído?
  - Perspectivas: *system-oriented* x *problem-oriented*
  - System-oriented: **processos** se comunicando com base em paradigmas de comunicação entre processos (IPC – Interprocess Communication); **nodos** (redes de sensores); **threads**
  - Problem-oriented (perspectiva de programação): **objetos**, **componentes**, **web services**

Arquitetura de sistemas distribuídos

- Como eles se comunicam, ou, mais especificamente, qual paradigma de comunicação é usado?
  - Tipos de paradigmas:
    - IPC
    - invocação remota
    - comunicação indireta

Arquitetura de sistemas distribuídos

- IPC:
  - comunicação entre processos em um relativo baixo nível;
  - primitivas de passagem de mensagens;
  - acesso direto a API de protocolos Internet (socket);
  - suporte a musticasting.

Arquitetura de sistemas distribuídos

- Invocação remota: mais comum
  - Requisição-resposta
    - Usado no HTTP
  - RPC (Remote Procedure Call)
    - Birrel and Nelson
  - RMI (Remote Method Invocation)
- Envolve um sender (cliente) e um receiver (servidor)
- Receivers conhecem os senders e, tipicamente, ambos devem existir ao mesmo tempo

Arquitetura de sistemas distribuídos

Comunicação indireta: senders e receivers com maior grau de independência (desacoplamento)

- Senders não precisam saber para quem estão enviando: space uncoupling
- Senders e receivers não precisam existir ao mesmo tempo: time uncoupling
  - Comunicação em grupo
  - Publish-subscribe
  - Message queues
  - Espaço de tuplas
  - Memória compartilhada

Publish-Subcribe

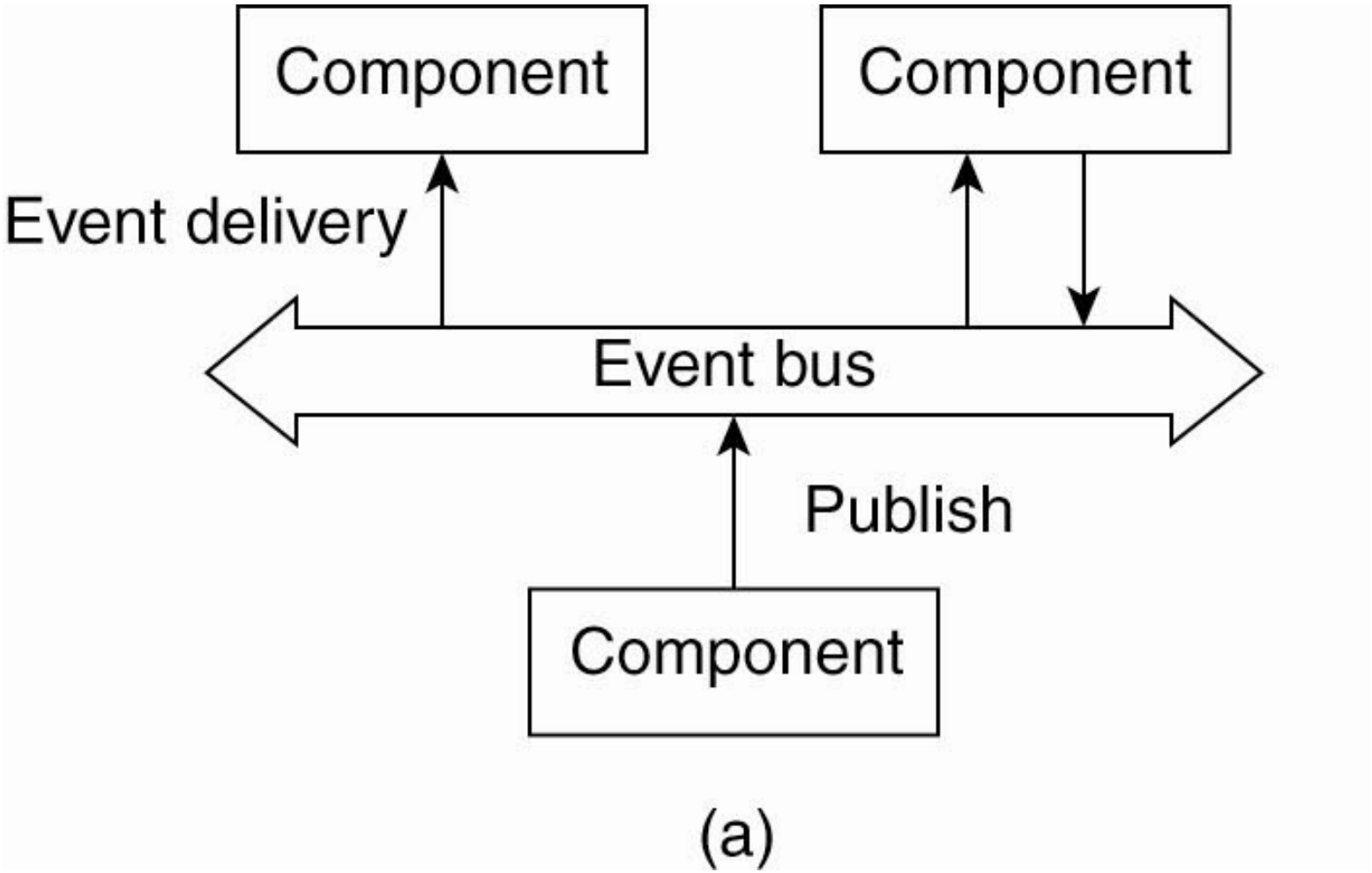


Figure 2-2. (a) The event-based architectural style and …

Memória Compartilhada

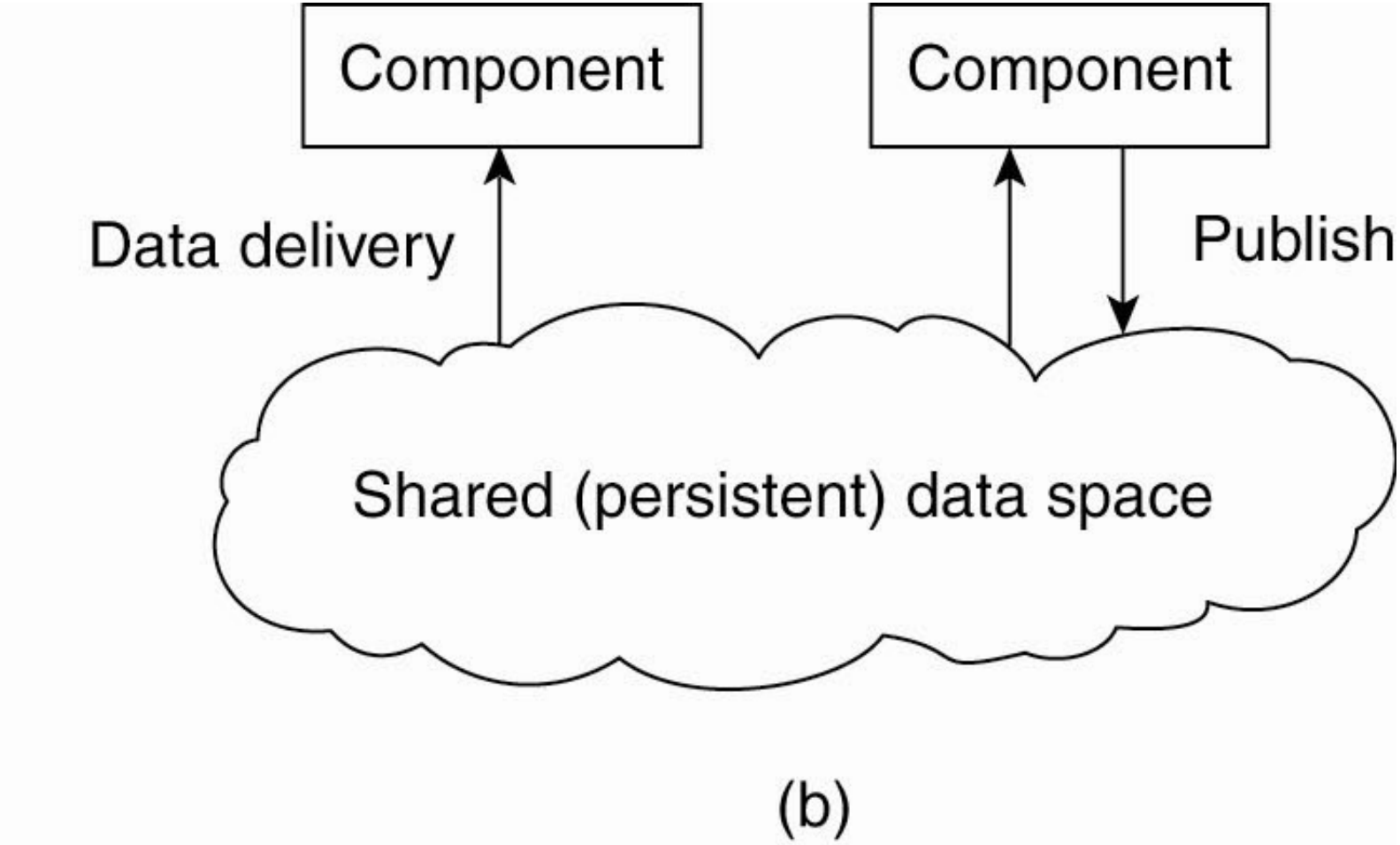


Figure 2-2. (b) The shared data-space architectural style.

Arquitetura de sistemas distribuídos

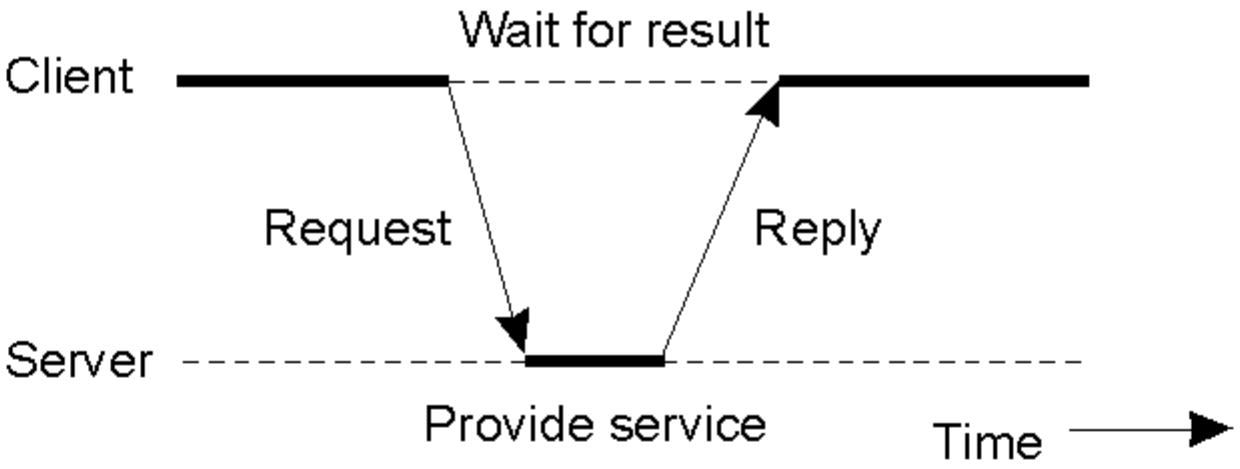
Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

Arquitetura de sistemas distribuídos

Quais papéis e responsabilidades eles têm na arquitetura como um todo?

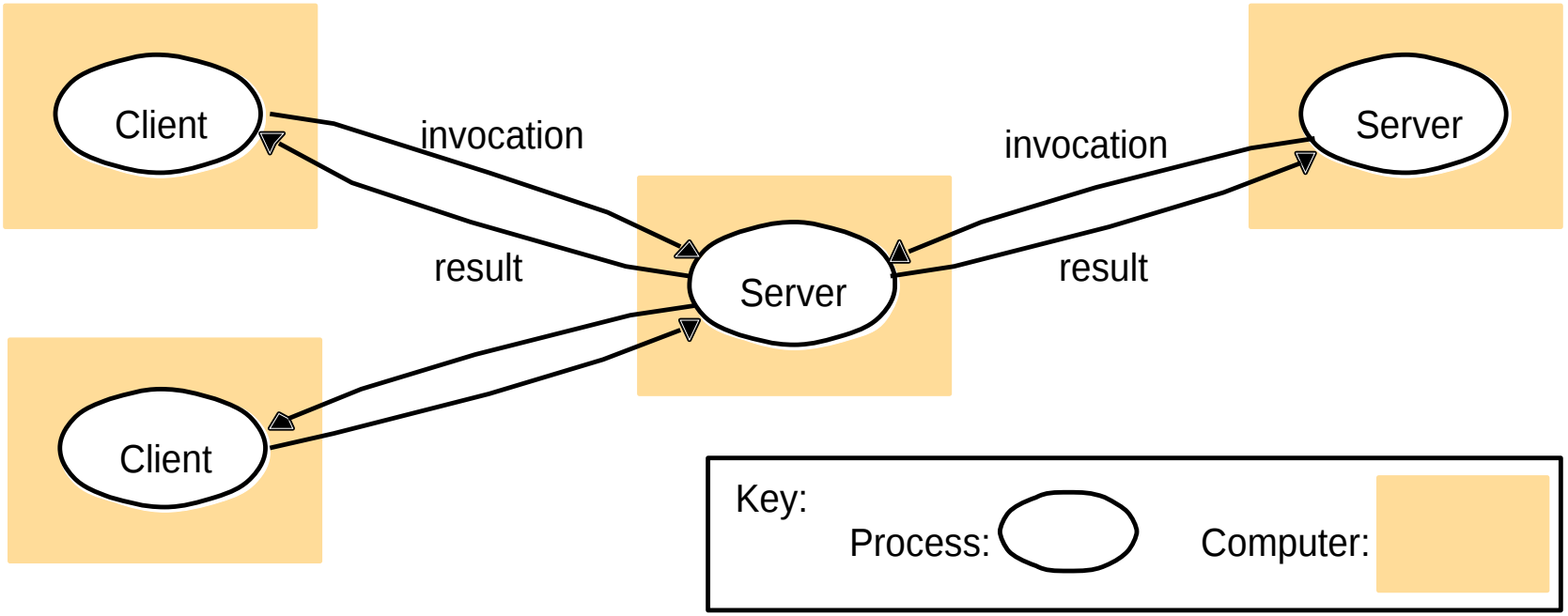
- Client-server
- Peer-to-peer

O modelo cliente-servidor

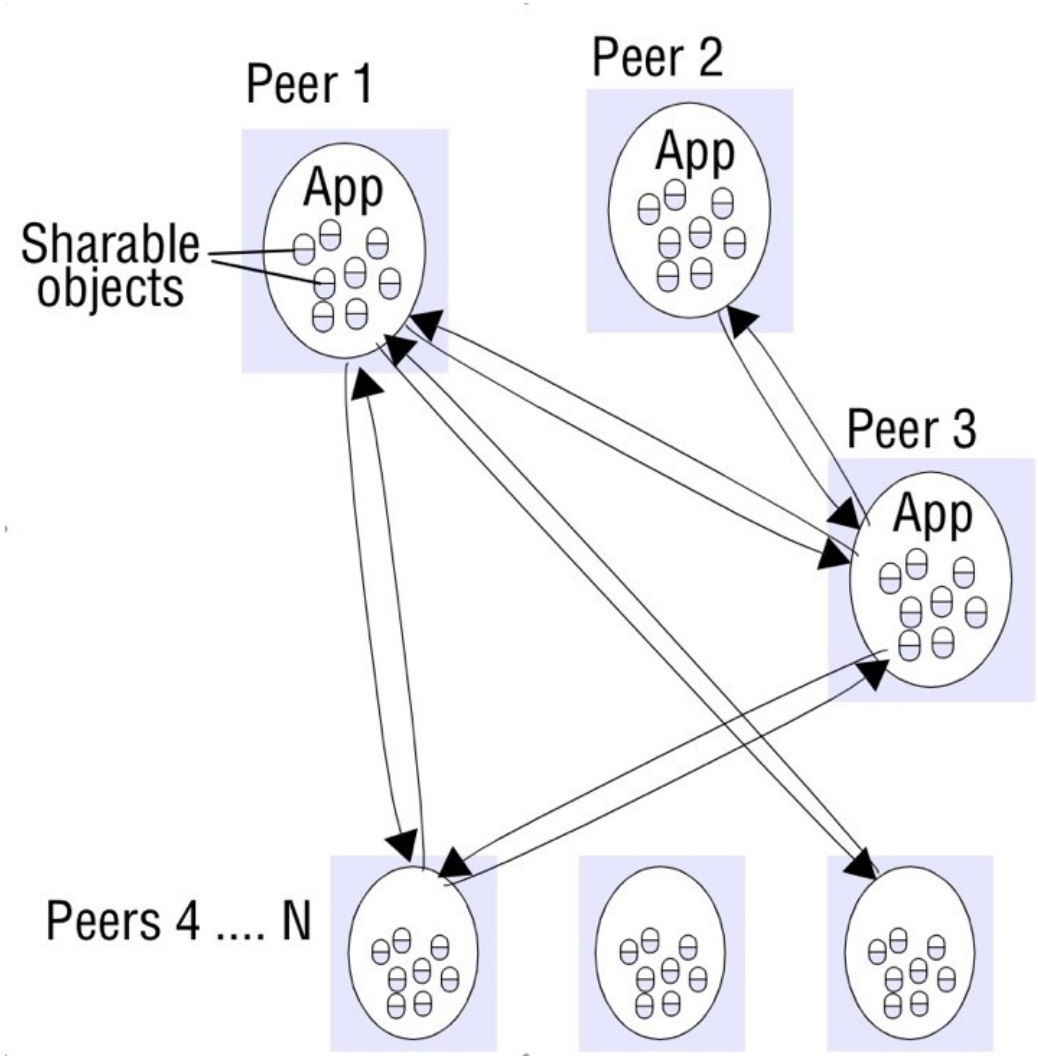


General interaction between a client and a server.

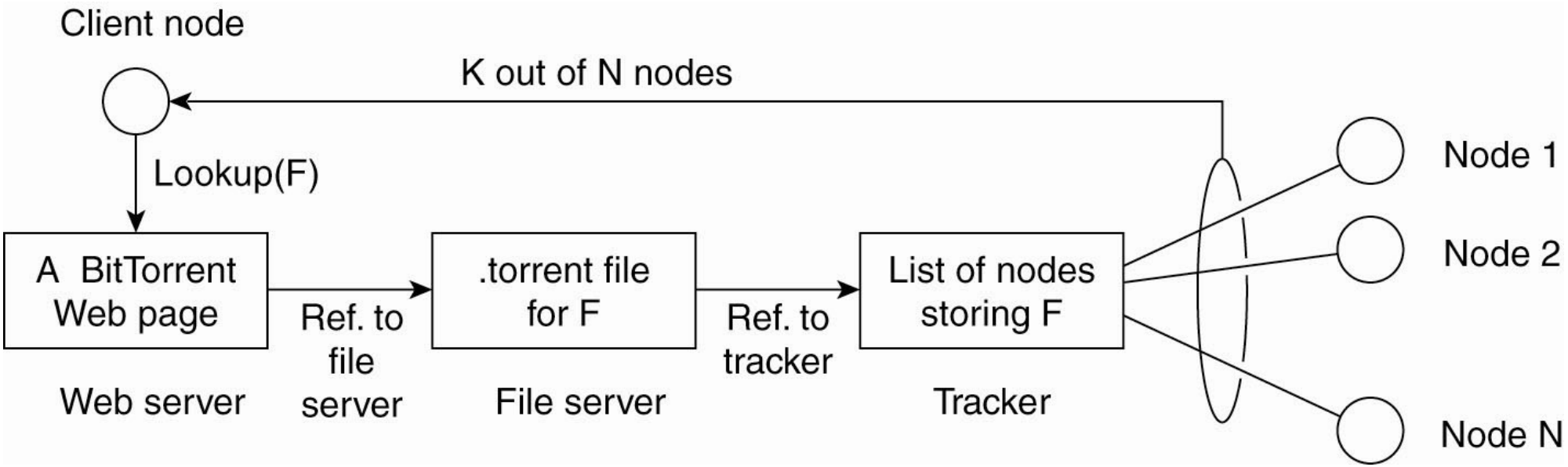
O modelo cliente-servidor



O modelo *peer-to-peer*



Collaborative Distributed Systems

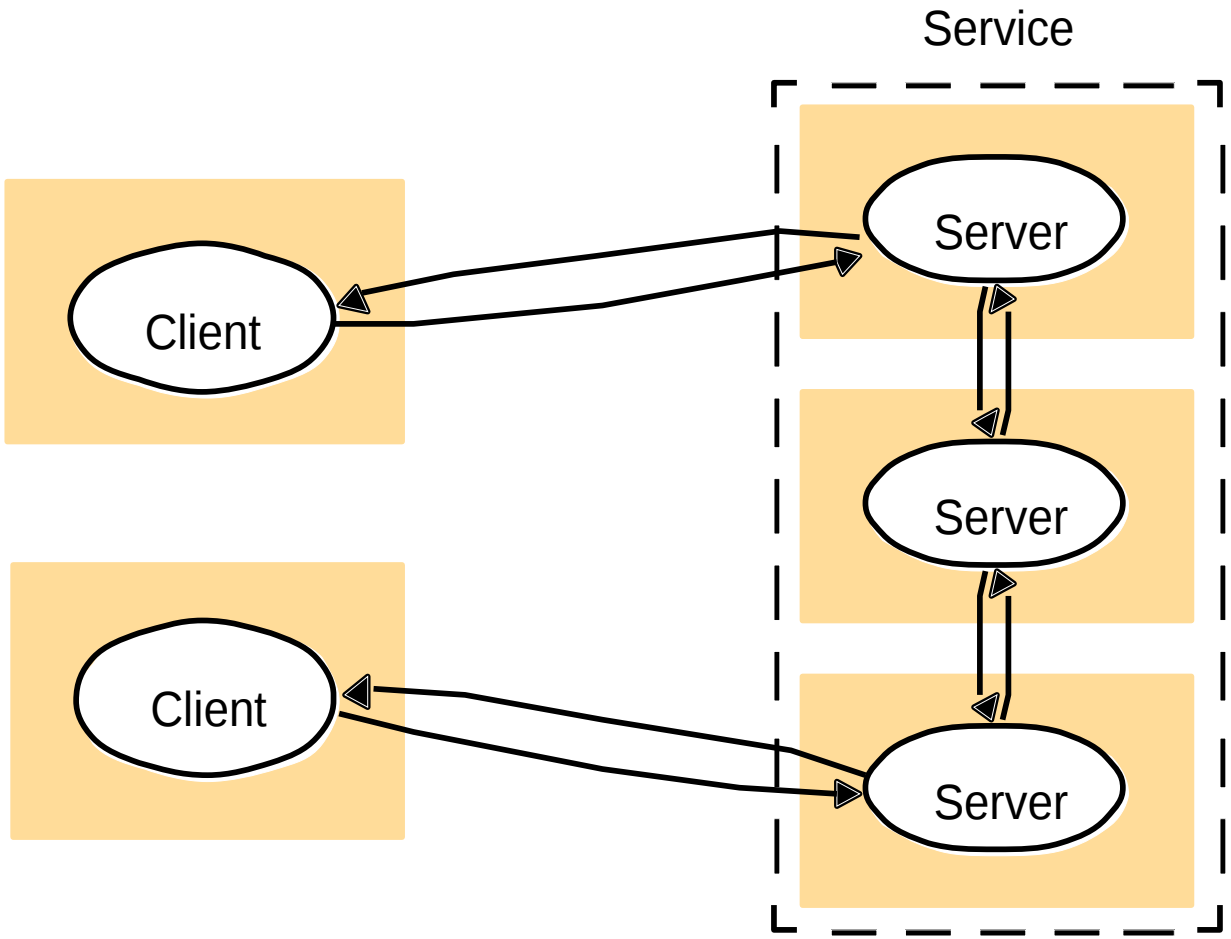


The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)].

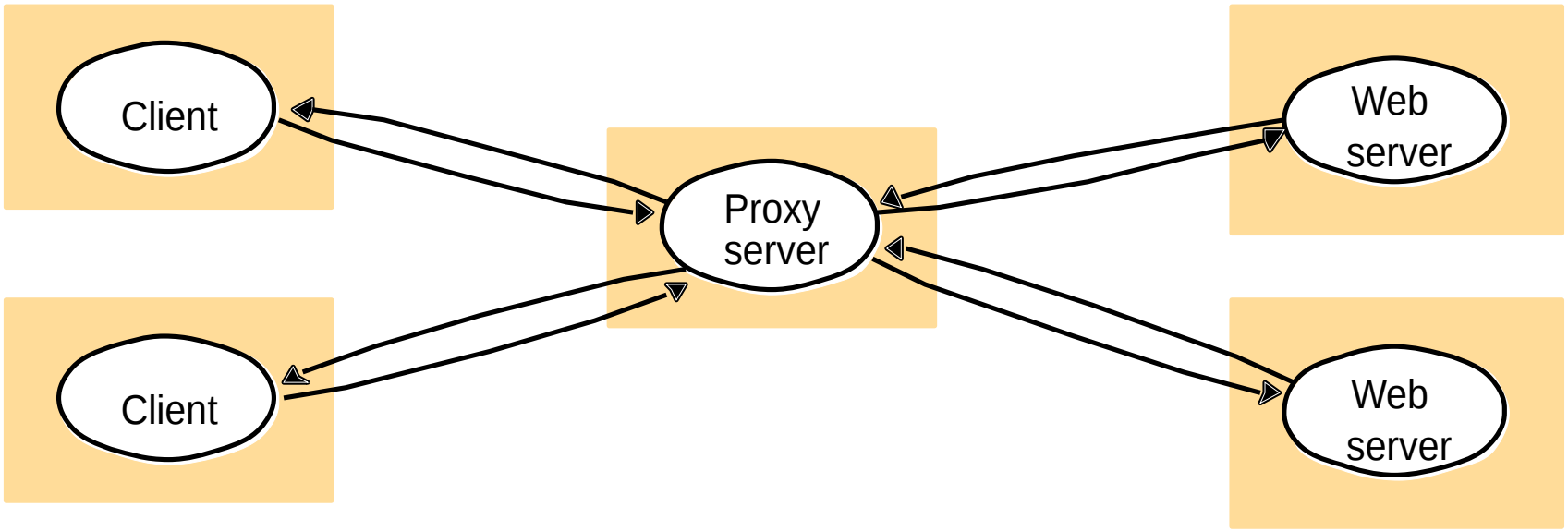
Arquitetura de sistemas distribuídos

- Como eles são mapeados para a infraestrutura física distribuída?
  - Mapeamento de serviços para múltiplos servidores
  - Caching
  - Código móvel
  - Agentes móveis

Um mesmo serviço provido por múltiplos servidores

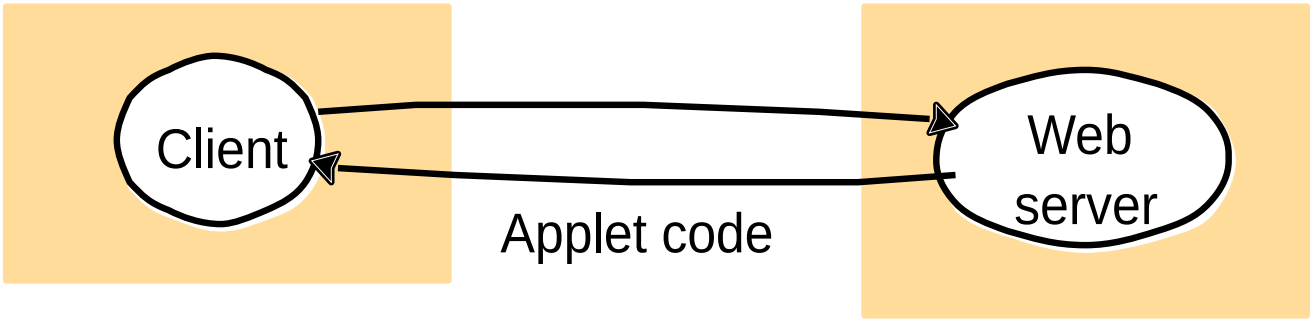


Caching:
Web Proxy Server



Código móvel: Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

Agente móvel

- Programa em execução (dados e código) que se movimenta de um computador para outro na rede, executando alguma tarefa
  - e.g., coletando informações, retornando resultados...
- Pode invocar/usar recursos locais de cada site que visita
  - e.g., bases de dados...
- Pode instalar softwares
  - e.g., para usar recursos inativos (idle)
- Questões de segurança

Padrões Arquiteturais

- Os mais importantes padrões arquiteturais para sistemas distribuídos
  - Em camadas (layering)
  - Hierárquico (tiered)
  - Thin client
  - Proxy, broker, reflection

Architectural Styles

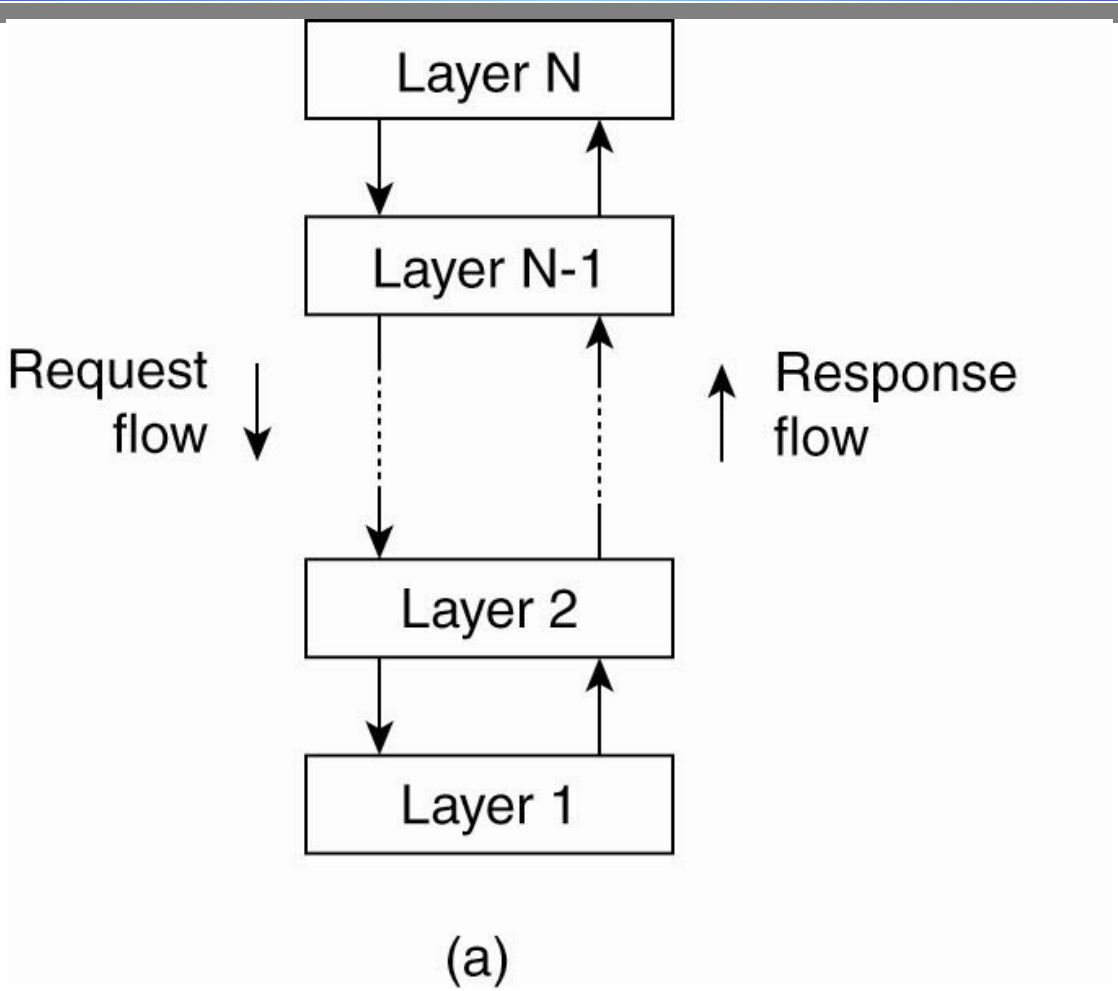
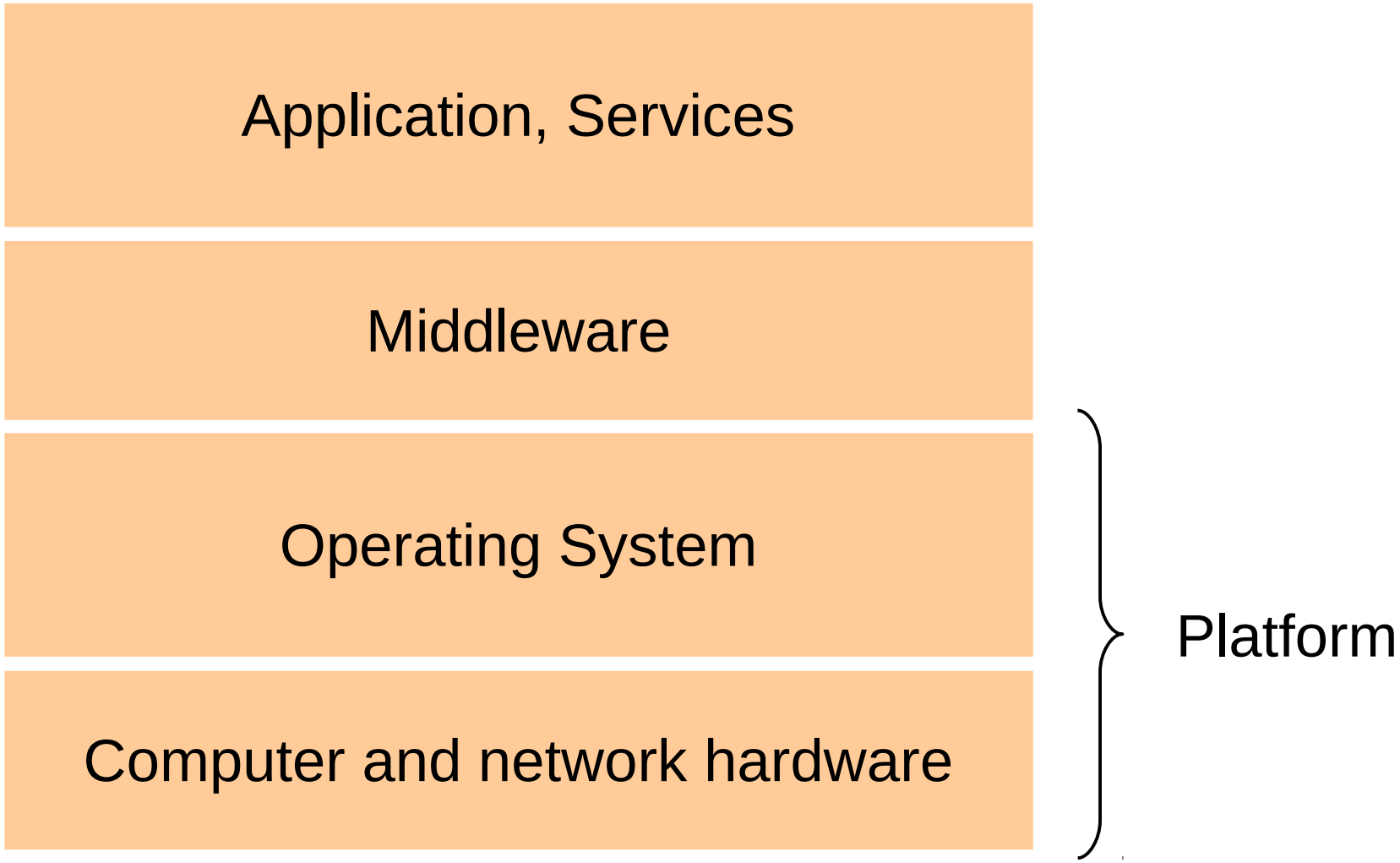
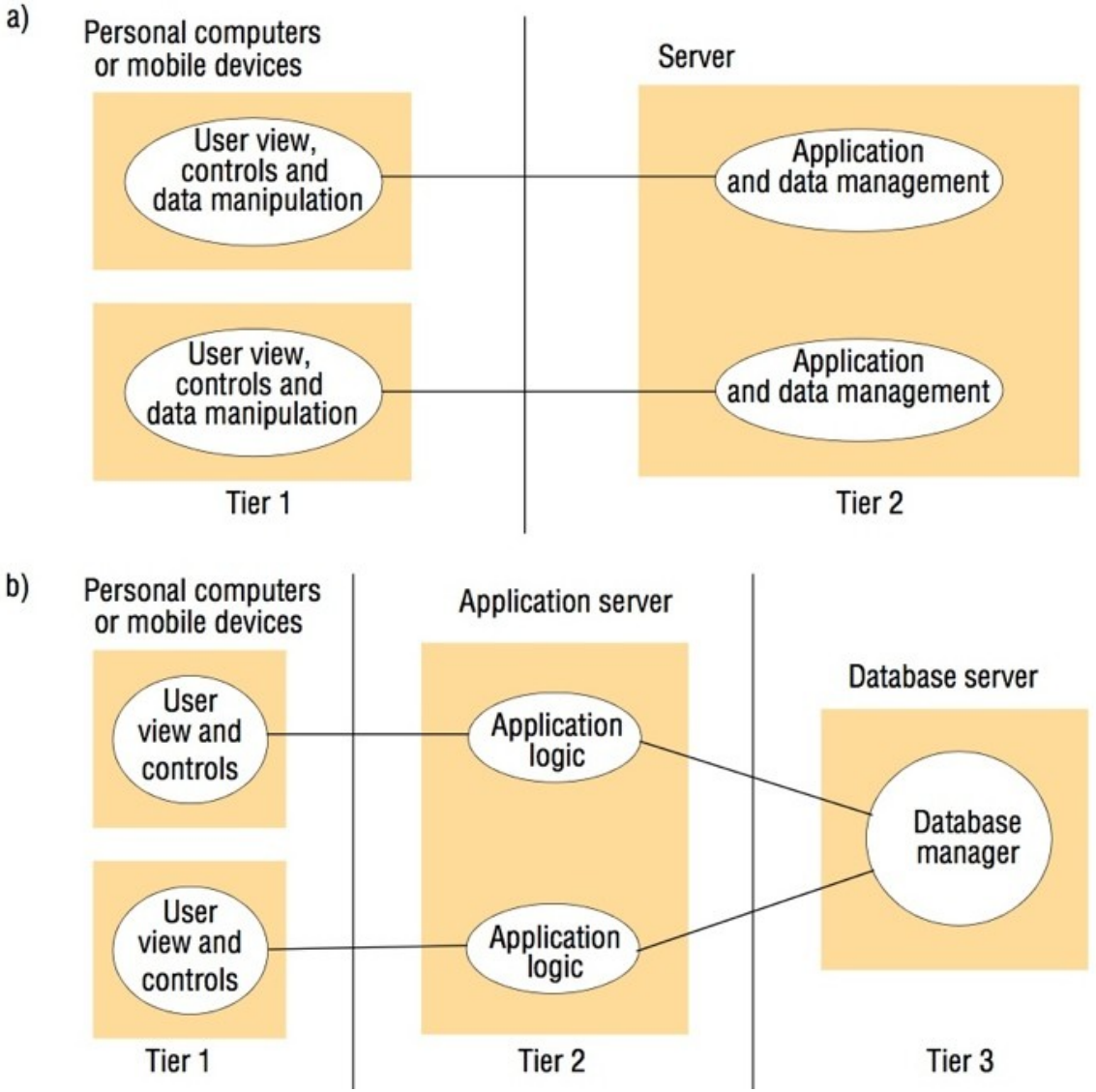


Figure 2-1. The (a) layered architectural style and …

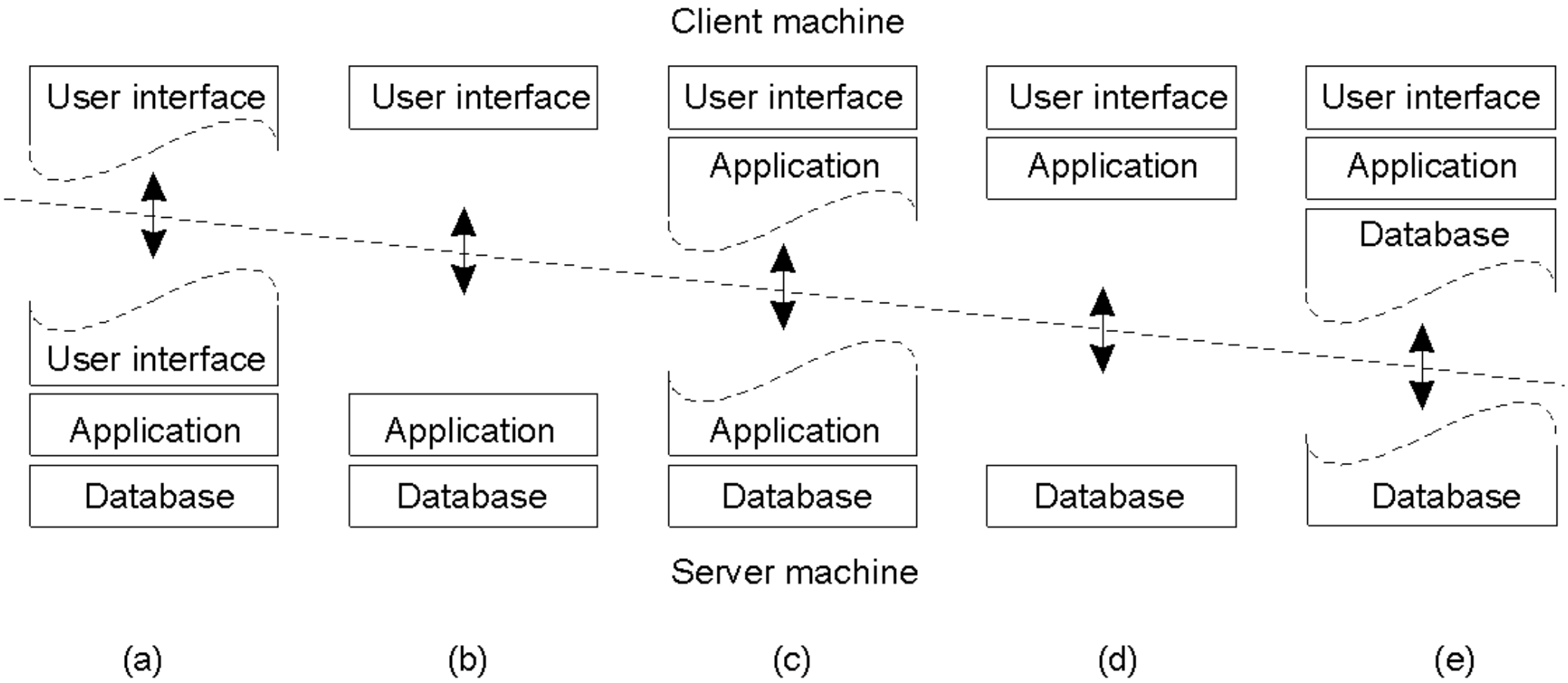
Camadas de software e hardware em sistemas distribuídos



Two-tier and three-tier architectures

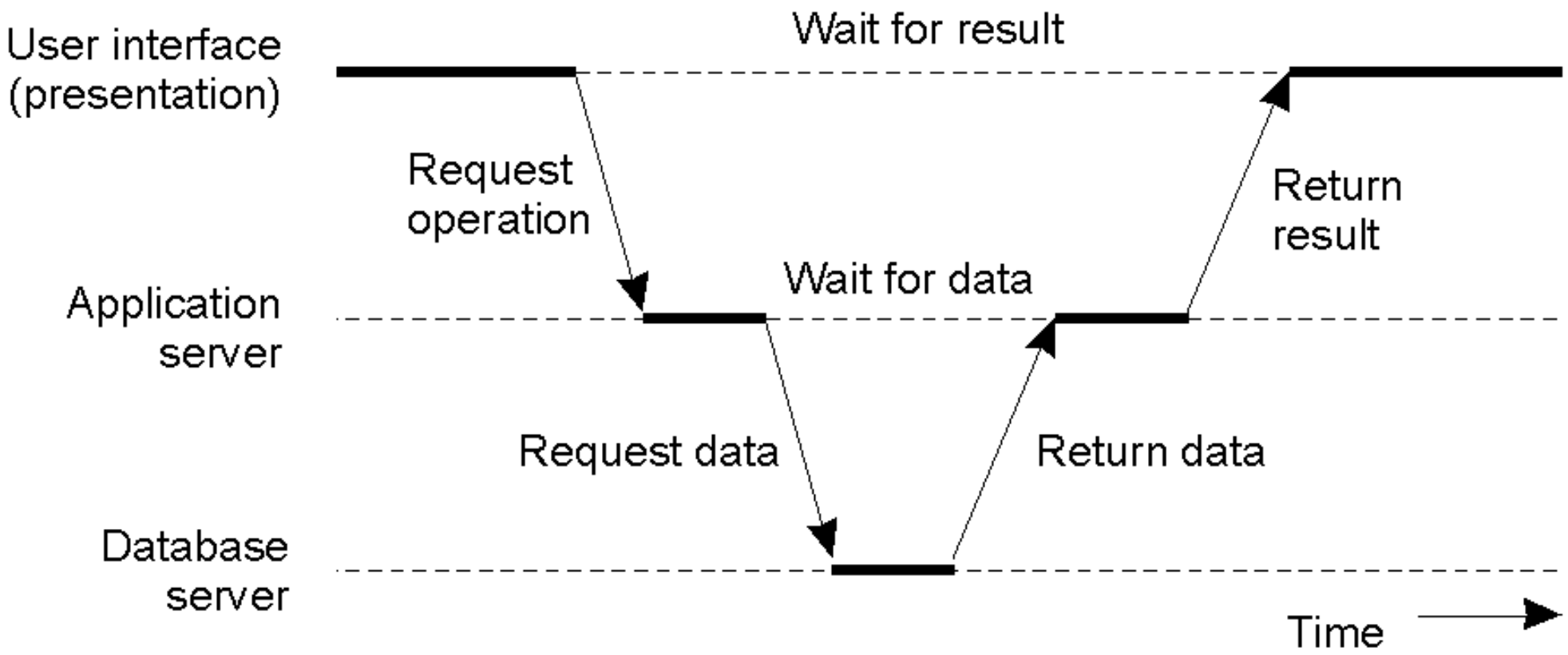


Multitiered Architectures (1)



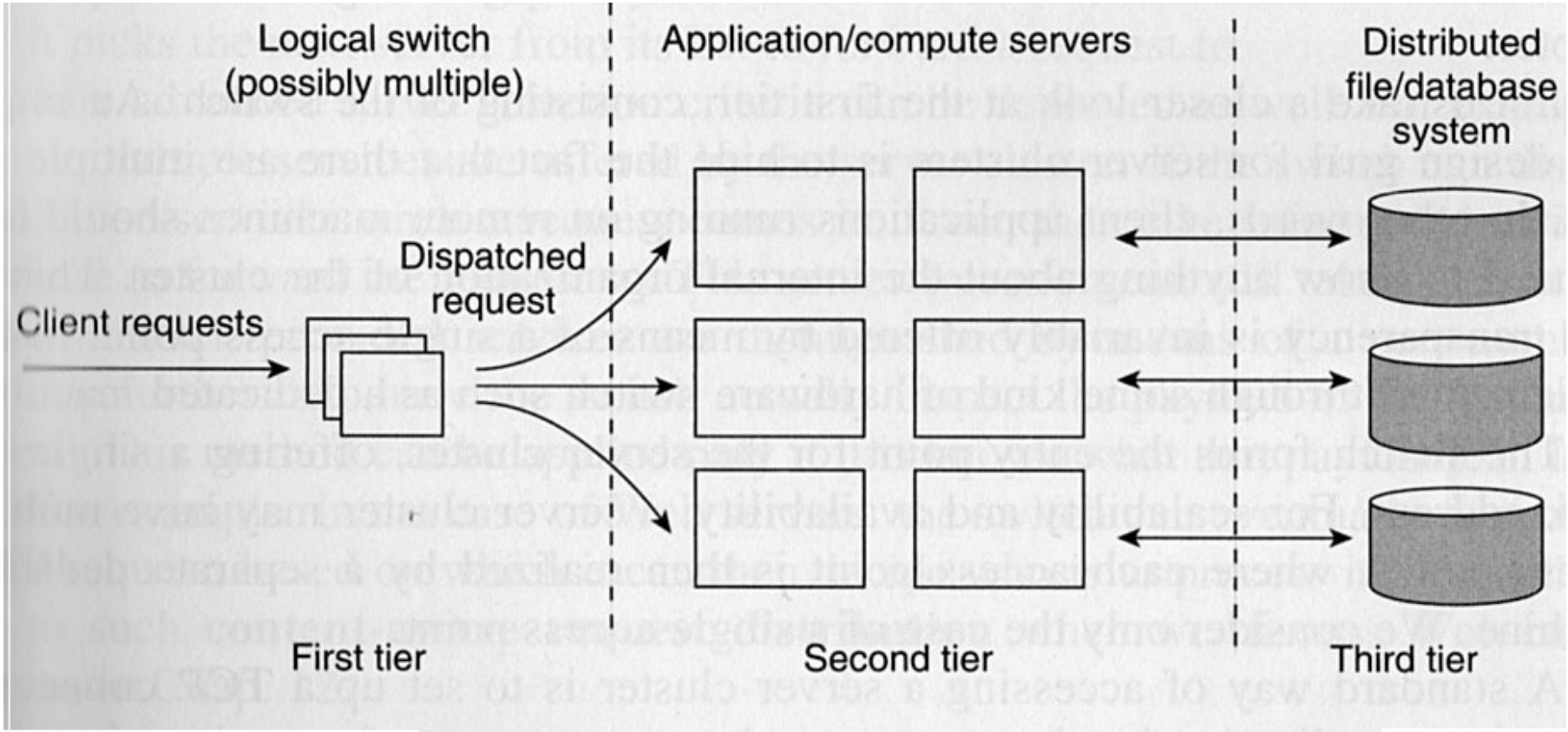
Alternative client-server organizations (a) – (e).

Multitiered Architectures (2)



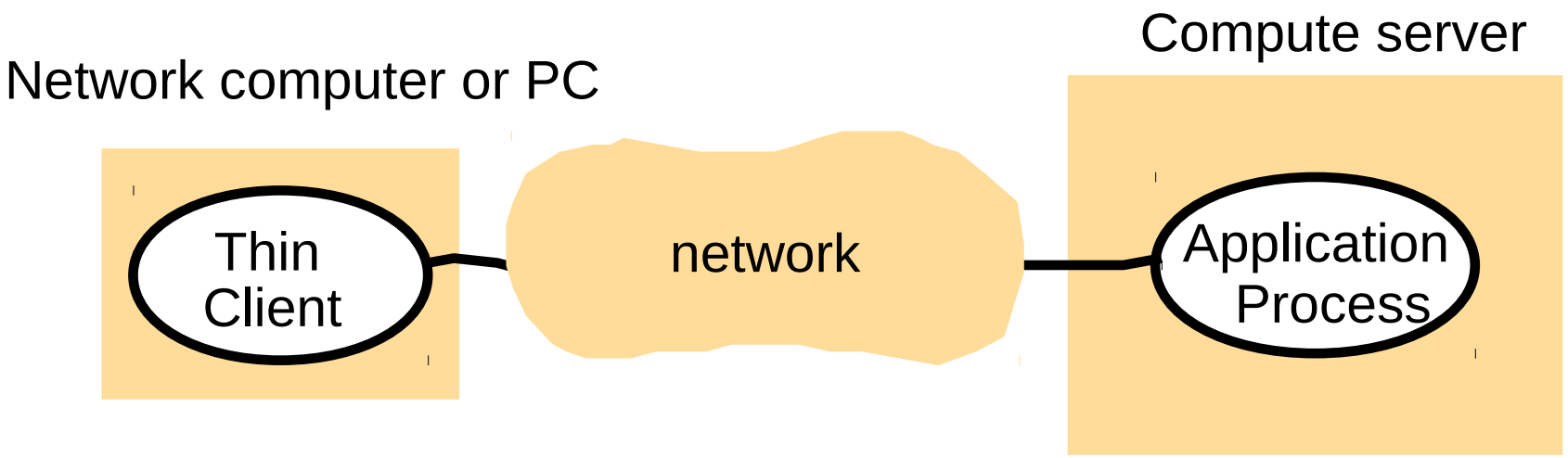
An example of a server acting as a client.

O Paradigma "Servidor Concorrente"

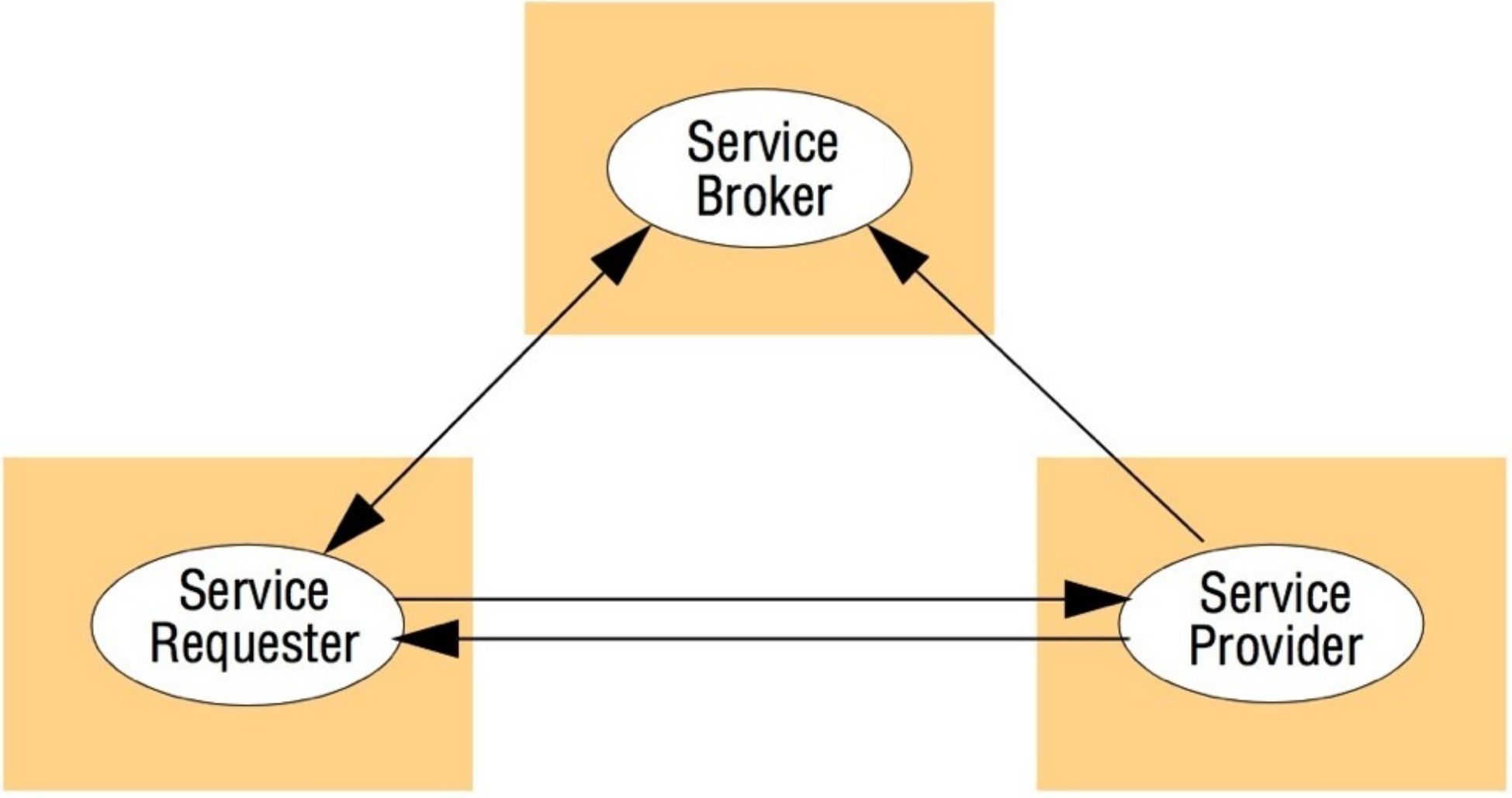


The general organization of a three-tiered server cluster.

Thin Clients



The web service architectural pattern



Importante: Arquitetura de Sistemas Distribuídos está diretamente relacionado a Arquitetura de Software...

Arquitetura de Software (i)

- Define como os **componentes** (incluindo conectores) são organizados e como eles devem interagir
  - Organização lógica / topológica
- Uma instanciação da arquitetura é chamada de **arquitetura do sistema** – ou **configuração**
- O sistema pode monitorar seu comportamento e tomar medidas apropriadas
  - Sistemas autônomos, autoadaptativos

Arquitetura de Software (ii)

- Define como os **componentes** são conectados, como trocam dados e como (em conjunto) são configurados no sistema

- Um componente tem uma **interface** bem definida e pode ser substituído ou trocado

- **Conectores:** intermediam comunicação, coordenação, ou cooperação entre componentes

An introduction to software Architecture

David Garlan & Mary Shaw (1994)

As the size of software systems increases, the algorithms and data structures of the computation no longer constitute the major design problems. When systems are constructed from many components, the organization of the overall system—the software architecture—presents a new set of design problems. This level of design has been addressed in a number of ways including informal diagrams and descriptive terms, module interconnection languages, templates and frameworks for systems that serve the needs of specific domains, and formal models of component integration mechanisms.

In this paper we provide an introduction to the emerging field of software architecture. We begin by considering a number of common architectural styles upon which many systems are currently based and show how different styles can be combined in a single design. Then we present six case studies to illustrate how architectural representations can improve our understanding of complex software systems. Finally, we survey some of the outstanding problems in the field, and consider a few of the promising research directions.

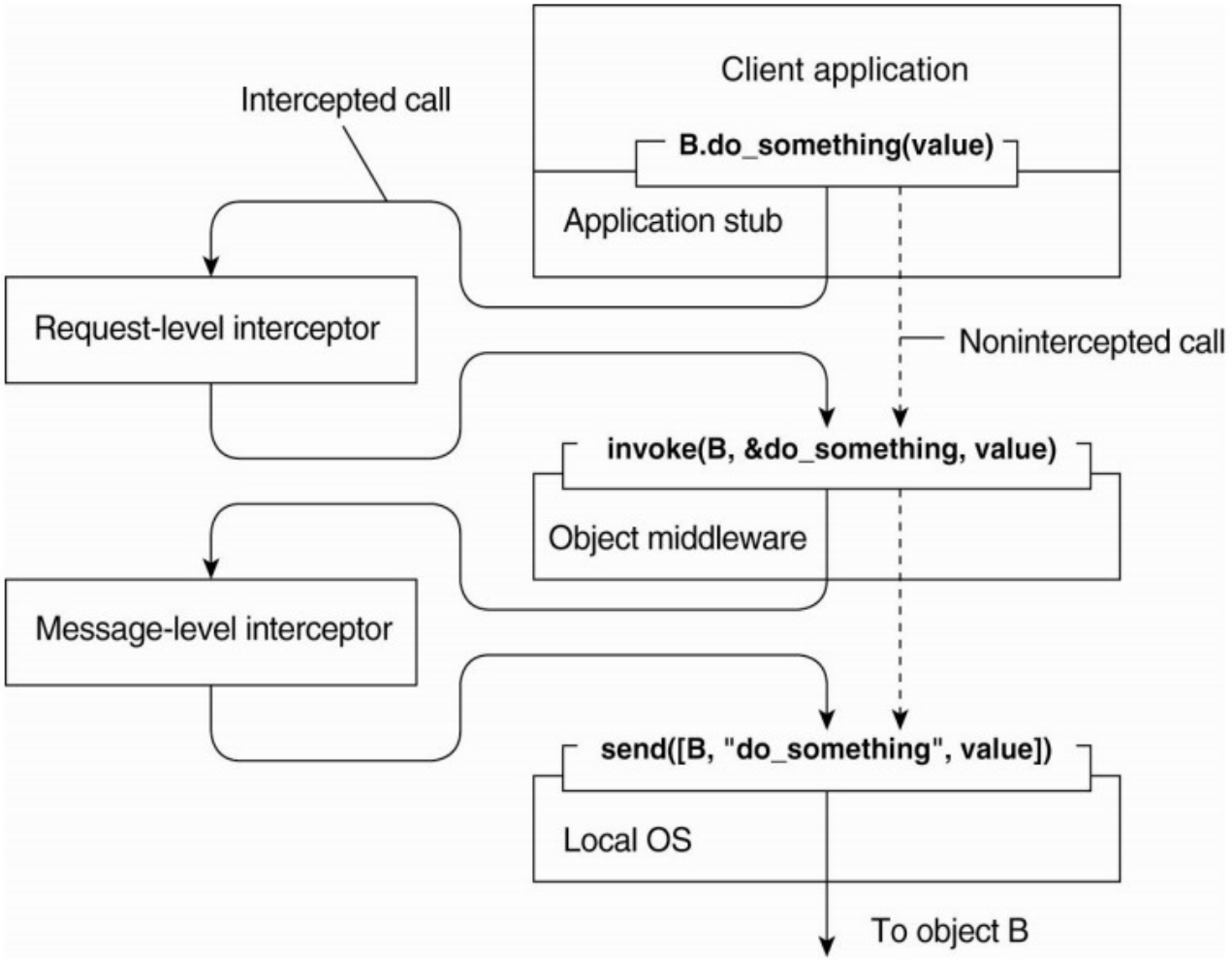
Abstração: O que é? (Jeff Kramer)

- The first emphasizes the process of removing detail to simplify and focus attention based on the definitions:
  - The act of withdrawing or removing something, and;
  - The act or process of leaving out of consideration one or more properties of a complex object so as to attend to others
- The second emphasizes the process of generalization to identify the common core or essence based on the definitions:
  - The process of formulating general concepts by abstracting common properties of instances, and;
  - A general concept formed by extracting common features from specific examples.

Padrões de Projeto em Sistemas Distribuídos

- Wrapper Façade
- Component Configurator
- Interceptor
- Extension Interface
- Reactor
- Proactor
- Asynchronous Completion Token
- Acceptor-Connector
- Scoped Locking
- Strategized Locking
- Thread-safe Interface
- Double-checked Locking Optimization
- Active Object
- Monitor Object
- Half-Sync/Half-Async
- Leader/Followers
- Thread-specific Storage

Interceptor



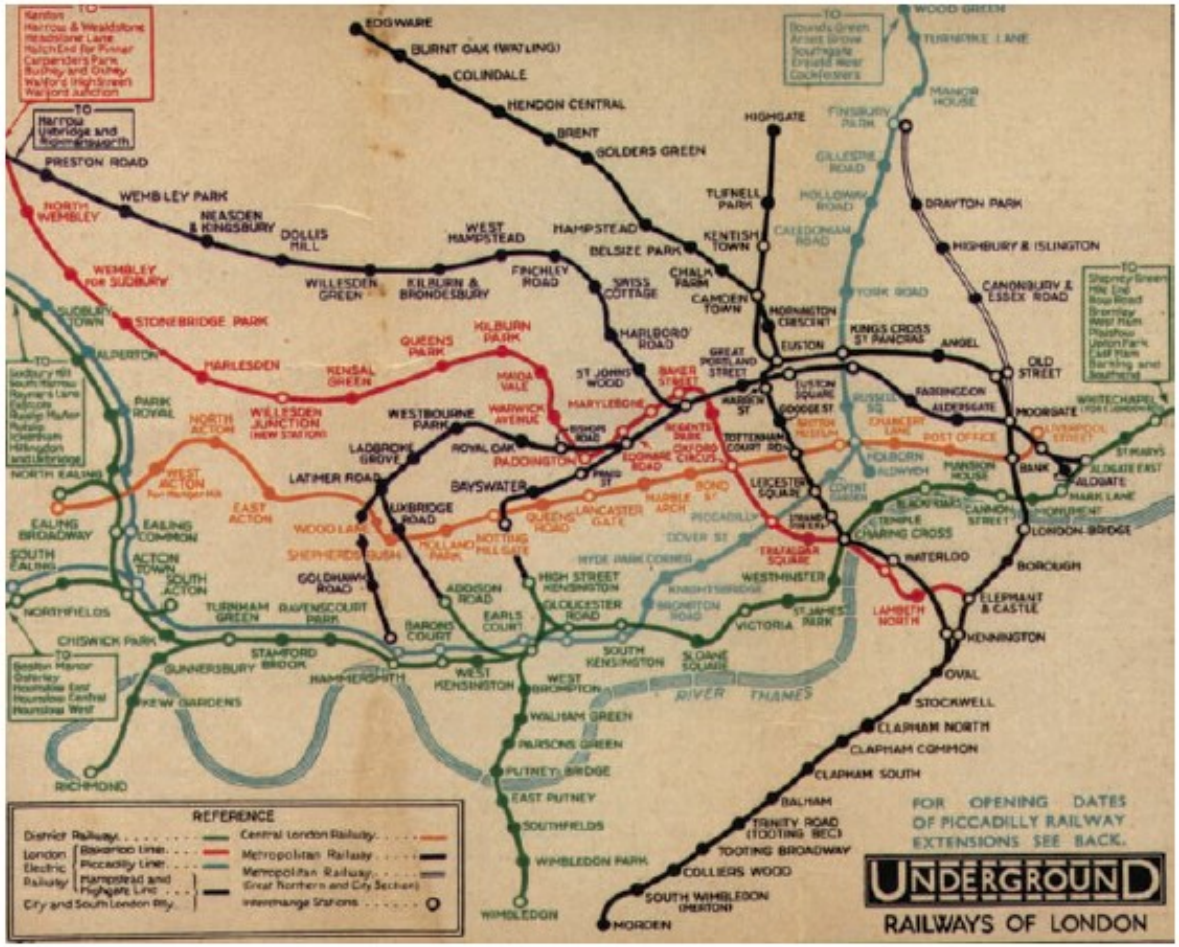
- Using interceptors to handle remote-object invocations.

Abstração:Por que é importante?

- “Once you realize that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs (and systems) is the ability to handle abstractions in a precise manner.” (*Keith Devlin*)


H MATISSE 52

Mapa do Metrô de Londres - 1928



Mapa do Metrô de Londres - 1933- Harry Beck

- Estudar o livro, capítulo 2
- Estudar o artigo *Is abstraction the key to computing?*, Jeff Kramer
- Estudar o artigo *An introduction to software Architecture*, David Garlan & Mary Shaw
- Produzir texto: pontos relevantes, intercessão entre os tópicos focados, contribuição própria

The Data Center is now the Computer*

- Companies like Google are starting to hire computer architects. When I asked Luiz Barroso of Google why, he said: “The data center is now the computer”

- Hence, computer architects are now designing and evaluating data centers

* David Patterson, Communications of the ACM, janeiro 2008

“The Data Center is now the Computer”

- Hence, computer architects are now designing and evaluating data centers

- This is certainly a provocative notion. However, if the data center is the computer, it leads to the even more intriguing question:
  - “What is the equivalent of the ADD instruction for a data center?”

    ➔ **Mapreduce**

“The Data Center is now the Computer”

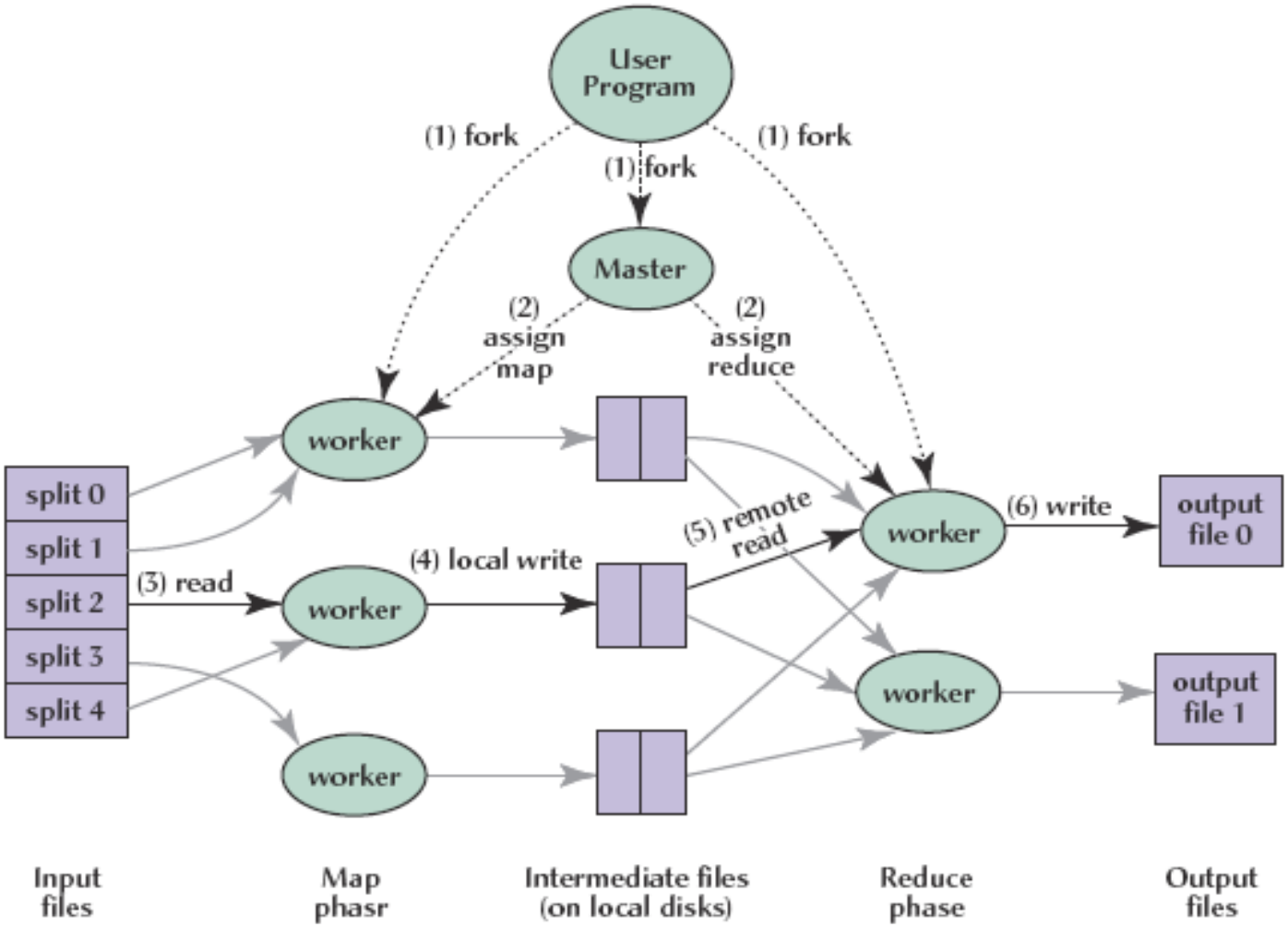


Fig. 1. Execution overview.

Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, OSDI 2004

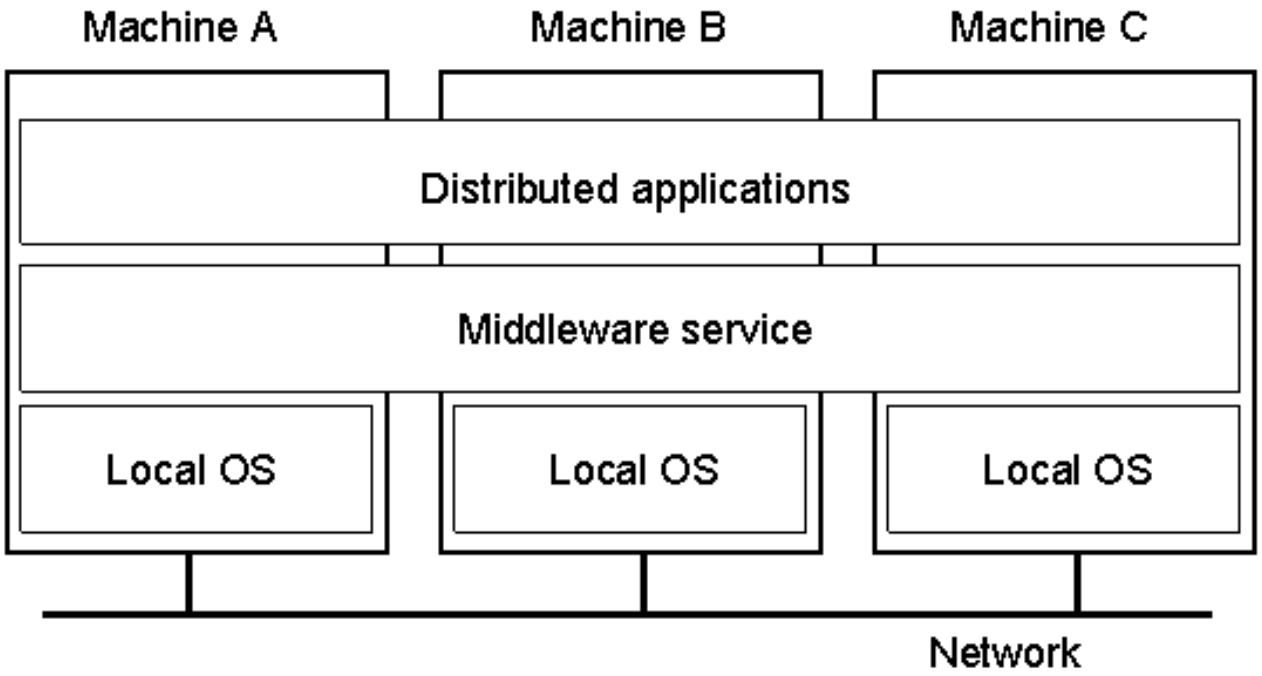
Middleware

- Camada de software que tem por finalidade
  - mascarar a heterogeneidade da plataforma subjacente (hardware, SO, linguagem)
  - resolver demais problemas oriundos da distribuição de forma transparente
  - prover um modelo de programação conveniente para o programador de aplicações
    - ex.: abstrações de alto nível para comunicação remota
  - Prover serviços de infraestrutura padronizados para uso no desenvolvimento de aplicações
    - ex.: resolução de nomes, segurança, transações etc.

Arquitetura e Middleware

- Middlewares em geral seguem um padrão arquitetural
- Soluções específicas muitas vezes devem ser adaptáveis a requisitos da aplicação
- Uma saída: middleware configurável

Middleware de sistemas distribuídos



O que constitui uma plataforma de middleware

- Processos, objetos ou componentes localizados nos computadores do sistema distribuído
- Interagem e cooperam entre si para prover o suporte de comunicação e compartilhamento de recursos necessário às aplicações
- Fornece as primitivas básicas para construção de componentes de software que funcionam cooperativamente em um sistema distribuído
- Plataforma de alto nível para o desenvolvimento de aplicações

Exemplos de middleware

- CORBA
- Java RMI e Jini
- Web services
- DCOM
- .Net
- RM-ODP

- Outros tipos de middleware?

Categories of middleware

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Limitações do Middleware

- Nem todas as decisões sobre questões de distribuição podem ser embutidas no middleware
- Algumas requerem conhecimento específico no nível das aplicações
  - Ex.: tratamento de falhas pode depender da semântica da aplicação
- A implementação destas questões no middleware pode comprometer a corretude e a confiabilidade

Requisitos de projeto para arquiteturas distribuídas

- Desempenho
  - tempo de resposta
    - Carga de processamento e desempenho do servidor e rede
    - Retardo em todos os componentes de software envolvidos (serviços do S.O., da rede, etc.)
  - *throughput* (vazão)
  - balanceamento de carga

Requisitos de projeto

- Qualidade de serviço (QoS)
  - Diz respeito ao grau de satisfação quanto às propriedades não-funcionais do sistema
    - confiabilidade, segurança, desempenho
  - Assume papel especial no caso de sistemas de tempo-real

Requisitos de projeto

- Uso de *caching* e replicação
  - Fator fundamental: melhoria de desempenho e disponibilidade
- Dependabilidade/Confiabilidade
  - Tolerância a falhas
    - Aplicações continuam funcionando corretamente mesmo na presença de falhas de HW, SW ou rede
      - Replicação, reconhecimento de entrega de mensagem
  - Segurança
    - Autenticidade, Confidencialidade, Integridade

Propriedades Fundamentais de Sistemas Distribuídos

- Interação
- Falhas
- Segurança

Modelo de interação

- Processos interagem entre si através da troca de mensagens para coordenar suas atividades
- Um algoritmo distribuído define os passos necessários para essa coordenação
- Cada processo possui seu próprio estado
  - Ausência de estado global
- O desempenho do canal de comunicação subjacente não é previsível (atraso, banda, *jitter*)
- Não é possível manter uma noção global de tempo única/precisa
- É difícil saber o estado do algoritmo distribuído

Desempenho do canal de comunicação

- Atraso: soma de várias componentes:

  transmissão + acesso à rede + propagação + recepção

- Largura de banda
  - Quantidade total de informação que pode ser transmitida em uma unidade de tempo
- *Jitter*
  - Variação do atraso

Relógios e temporização de eventos

- Cada computador possui seu próprio relógio
  - Pode ser usado para marcar o tempo de eventos locais
- Mas sem significância global
  - Cada relógio marca um tempo diferente
  - Defasagem entre os relógios
- Técnicas para sincronização de relógios podem ser aplicadas
  - GPS, algoritmos de sincronização

Duas variantes do modelo de interação

- **SD síncrono** – limites conhecidos para:
  - tempo de execução de cada passo em um processo
  - tempo de transmissão (e recepção) de mensagens
  - taxa de defasagem dos relógios locais
- **SD assíncrono** – limites não são conhecidos para as variáveis acima – mais condizente com a realidade

Duas variantes do modelo de interação

Mais fácil construir aplicações sobre sistemas síncronos
ex.: *timeouts* p/ detectar falhas

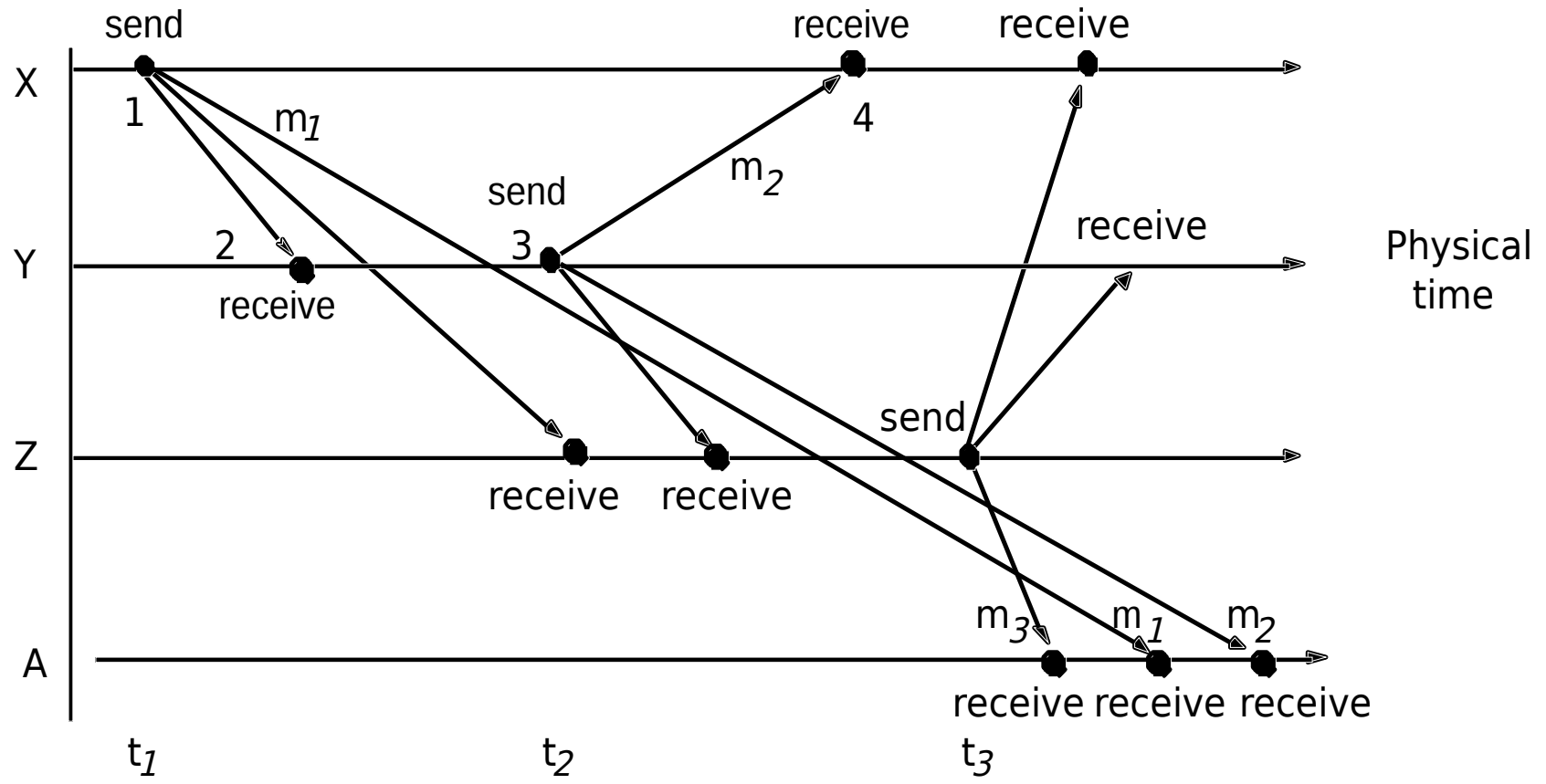
Sistemas síncronos podem ser construídos desde que seja possível reservar recursos de forma determinística

Compartilhamento dos recursos com outras aplicações gera assincronismo

Ordenação de eventos

- Determinar se um evento em um processo ocorreu antes ou depois de outro evento em outro processo
  - ex.: envio e recepção de uma mensagem
- Pode ser baseada em relógios físicos
  - Com o emprego de alguma técnica para sincronização de relógios
  - Tem um limite de precisão que pode não ser tolerado
- Outra alternativa: relógios lógicos
  - Com base na relação de causa-e-efeito entre eventos

Ordenação de eventos: exemplo



Usuário A pode ver:

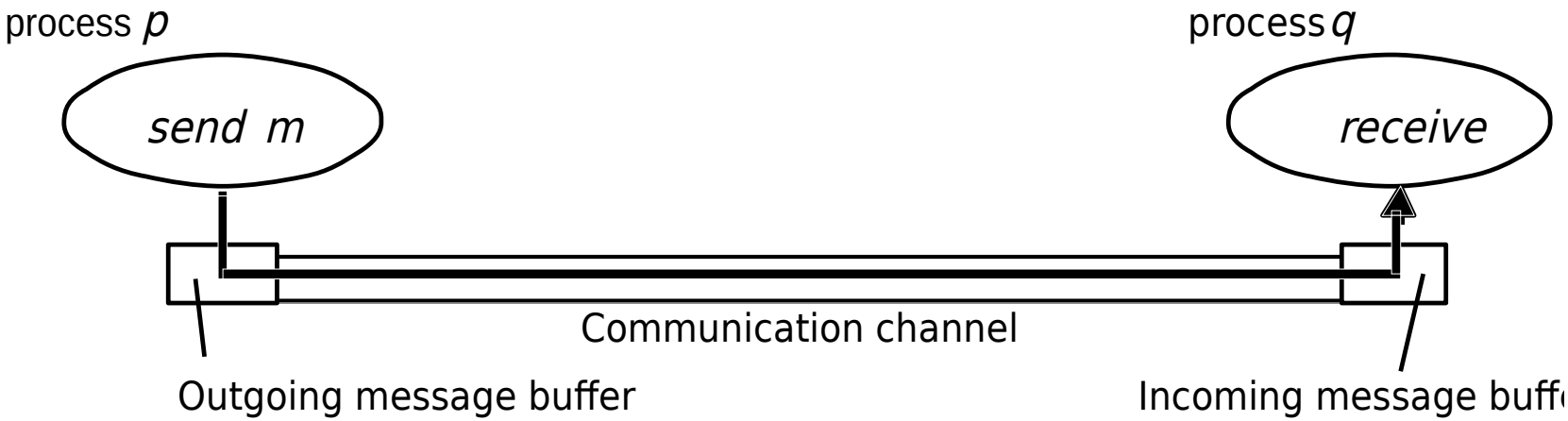
Inbox:		
Item	*From*	*Subject*
23	Z	Re: Meeting
24	X	Meeting
25	Y	Re: Meeting

Modelo de Falhas

- Processos e canais podem falhar
  - Desvio em relação ao comportamento correto

- Falhas de omissão
  - Processo: processo termina inesperadamente
    - Fail-stop: outros processos podem detectar a falha do processo (através de *timeouts*): requer sistema síncrono
  - Canal: mensagens enviadas não são recebidas

O modelo de processos e canais



Modelo de Falhas (2)

- Falhas arbitrárias
  - Qualquer tipo de erro pode acontecer:
    - Canal: mensagens podem ser omitidas, alteradas, duplicadas, etc.
    - Processo: passos em um algoritmo podem ser omitidos...
      - Não é possível detectar a falha
- Falhas de temporização
  - Violação das suposições sobre a temporização de eventos em sistemas síncronos

Falhas de omissão e falhas arbitrárias

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send,* but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Falhas de temporização

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

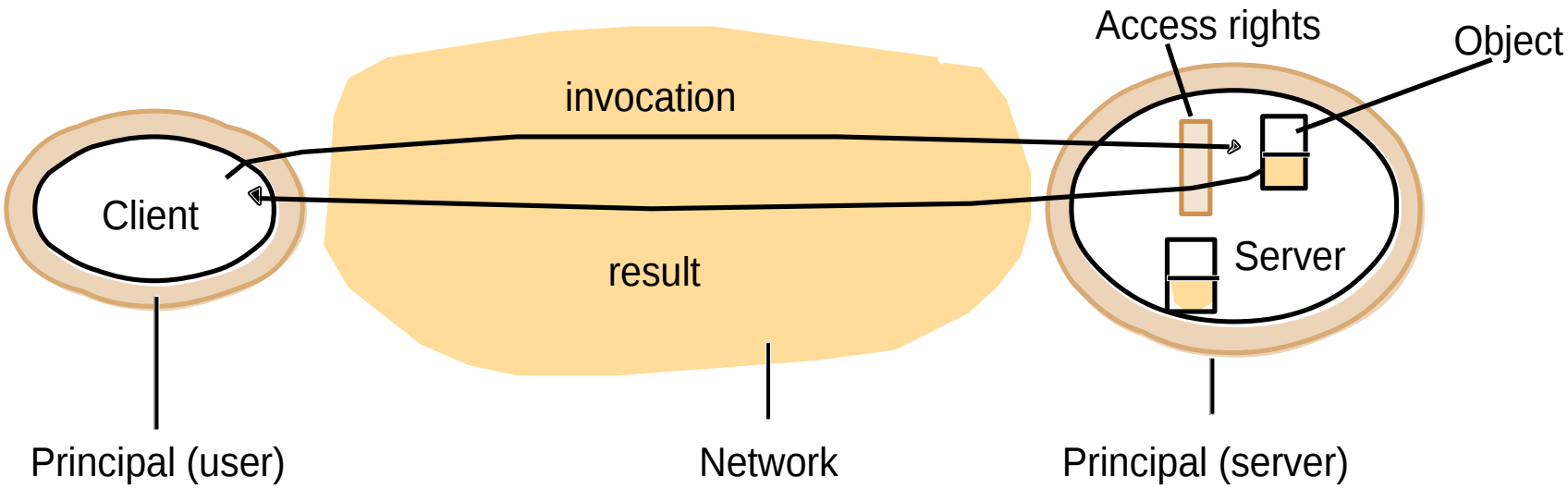
Lidando com falhas

- Mascaramento de falhas
  - Por exemplo, utilizando múltiplos servidores replicados para ocultar a falha de uma das réplicas
- Confiabilidade da comunicação
  - Detecção de mensagens com erro (CRCs)
  - Detecção de mensagens perdidas e retransmissão
  - Detecção de mensagens duplicadas

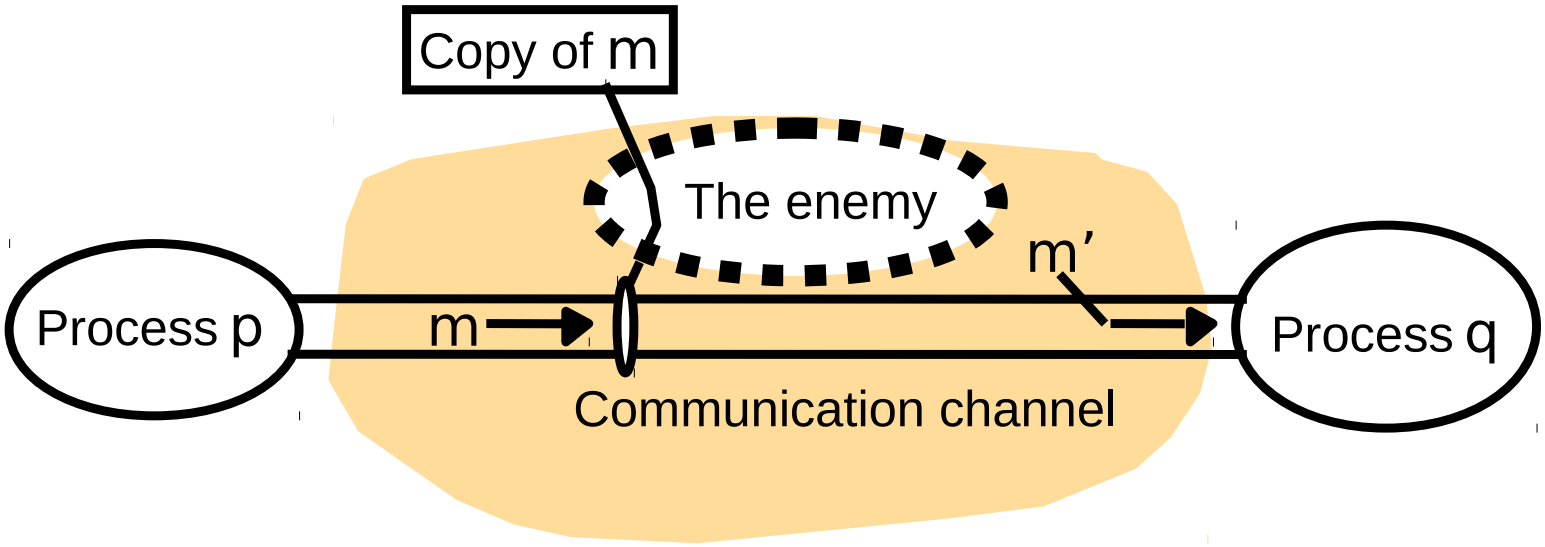
Modelo de segurança

- Medidas de proteção aplicadas a
  - Objetos (ou processos)
    - Direitos de acesso às interface dos objetos
    - Identidade dos objetos (*principals*) e autenticação
  - Canais de comunicação
    - Encriptação dos dados transmitidos
    - Modelo de canal seguro
      - Autenticação dos objetos que se comunicam através do canal
      - Privacidade e integridade dos dados
      - Marcas de tempo nas mensagens para impedir *replay*

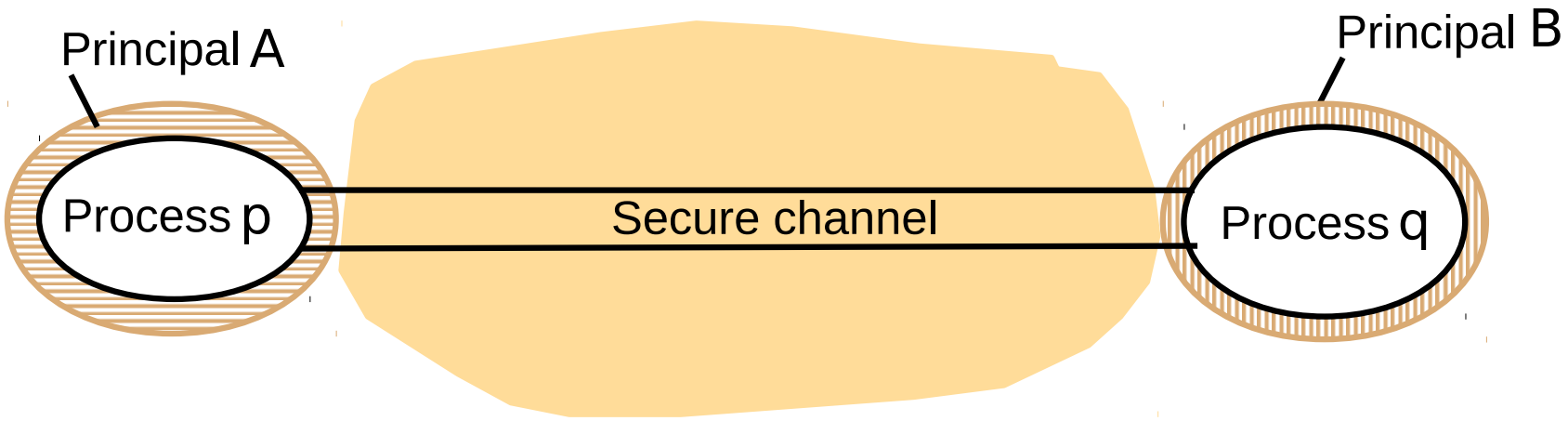
Objetos e "*principals*"



Intrusos na comunicação



Canais seguros



Outros problemas de segurança

- Negação de serviço
- Código móvel
- Um modelo de segurança deve enumerar todos os tipos de ataques aos quais um sistema distribuído estará sujeito
  - E empregar técnicas apropriadas para lidar com cada um
  - O sistema será tão “seguro” quanto esse modelo

Leitura

Capítulo 2 do Distributed Systems, Ed. 5, George Coulouris

Capítulo 2 do Distributed Systems, Ed. 2, Tanenbaum

Créditos

Prof. Sérgio T. Carvalho

sergio@inf.ufg.br