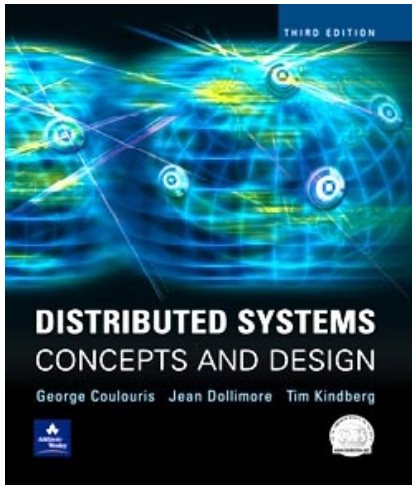
Slides for Chapter 2: Systems Models



From Coulouris, Dollimore and Kindberg

Distributed Systems:

Concepts and Design

Modelos de Sistemas

- Um **modelo arquiteturial** de um SD diz respeito com a **colocação de suas partes e os relacionamentos** entre elas.
- Exemplos incluem:
  - modelo **cliente/servidor**;
  - modelo **peer-to-peer**;
  - a partição de dados ou **replicação** em servidores cooperantes;
  - o "caching" de dados por **servidores proxy** e **clientes**;
  - o uso de código móvel e **agentes móveis**;
  - **dispositivos móveis** em uma rede.

Modelos de Sistemas

- **Modelos de Sistemas** – descrição formal do **comportamento** ou das **propriedades** que são comuns em todos os **modelos arquiteturiais**.

- **Não existe tempo global em um SD**, assim os clocks em diferentes computadores não necessariamente fornecem o mesmo tempo ...

- Toda **comunicação** entre processos é alcançada por meio de troca de mensagens.

Modelos de Sistemas

- **Comunicação** de mensagens sobre uma rede de computadores **pode ser afetada por retardos** (atrasos), podem sofrer de uma **variedade de falhas** e é vulnerável a **ataques contra segurança**.

- Estas questões são enfocadas por três modelos:
  - Modelo de Interação
  - Modelo de Falhas
  - Modelo de Segurança

Modelo de Interação

Trata com a performance e a dificuldade de se estabelecer limites de tempo em um SD, por exemplo, para entrega de mensagens.

Modelo de Falhas

Especificação precisa das falhas que podem ser exibidas por processos e canais de comunicação. Define comunicação confiável e processos corretos.

Modelo de Segurança

Discute as possíveis ameaças a processos e canais de comunicação. Introduz o conceito de canal seguro, o qual é seguro contras as ameaças.

2.1 Introdução

- **Sistemas Distribuídos** devem ser projetados para funcionarem corretamente na mais ampla e possível gama de circunstâncias e em face de muitas possíveis **dificuldades e ameaças**.

2.1 Introdução

- Dificuldades e Ameaças para SDs

São os problemas que projetistas de SDs enfrentam.

Modos de uso variando amplamente:

As partes componentes de sistemas estão sujeitas a variações em carga de trabalho (workload) – páginas web são acessadas muitas e muitas vezes por dia. Algumas partes de um sistema podem ser desconectadas, ou fracamente conectadas por algum tempo – por exemplo, quando computadores móveis são incluídos em um sistema. Algumas aplicações têm requisitos especiais como alta largura de banda e baixa latência, como em aplicações multimídias.

Ampla gama de ambientes de sistemas:

Um SD deve acomodar HW heterogêneo, SOs e redes. As redes podem diferir amplamente em performance – redes sem fio operam em uma fração da velocidade de redes locais. Sistemas diferindo amplamente em escalas – desde dezenas de computadores a milhões de computadores – devem ser suportados.

Dificuldades e Ameaças para SDs

Problemas Internos:

- **Clocks** não sincronizados,
- inconsistências em atualizações de dados e
- muitos modos de HW e SW envolvendo os componentes individuais de um sistema.

Ameaças Externas:

- Ataques sobre a integridade e sigilo (secrecy) dos dados e
- Recusa de serviços (denial of sevice).

2.1 Introdução

- SDs de diferentes tipos compartilham importantes propriedades fundamentais e fazem surgir **problemas de projeto comuns**.

- As **propriedades comuns** e as **questões de projeto** para SDs são abordadas, neste capítulo, na forma de **modelos descritivos**.

2.1 Introdução

- Cada **modelo descritivo** é voltado para prover uma descrição abstrata, simplificada, mas consistente de um aspecto de projeto de sistema distribuído.

2.1 Introdução

- Um **Modelo Arquiteturial** define o modo no qual os componentes de sistemas interagem e o modo no qual eles são mapeados sobre uma rede de computadores.

2.1 Introdução

- Na seção 2.2 ...

- A **estrutura em camadas do software** de SDs e o **modelos arquiteturiais principais** que determinam as localizações e as interações dos componentes.

2.1 Introdução

- Variantes do modelo Cliente/Servidor, incluindo aqueles devido ao uso de código móvel.
- As características de um SD para o qual dispositivos móveis podem ser adicionados ou removidos convenientemente.
- Requisitos de projeto gerais para SDs.

2.1 Introdução

- Na seção 2.3 ...

- Introduz-se os **três modelos fundamentais.**
- O propósito é **especificar as questões de projeto**, dificuldades e ameaças que devem ser resolvidas, no sentido de desenvolver SDs que executam suas tarefas **corretamente**, **confiavelmente** e com **segurança**.

2.1 Introdução

- Os modelos fundamentais proporcionam visões abstratas daquelas características de SDs que afetam suas características de "***dependability***" – ***corretude (***correctness***)***, ***confiabilidade*** *(*reliability*)* *e* ***segurança (***security***).***

Dependability

- É um requisito na maioria dos domínios da aplicação.

- É crucial não somente em atividades de sistemas de comando e controle, onde a vida pode estar em jogo, mas também em muitas aplicações comerciais, incluindo comércio na Internet, onde a segurança financeira e consistência dos participantes depende sobre a **"*dependability*"** dos sistemas que eles operam.

Dependability

- Define-se a dependability de sistemas de computadores como:
  - Correctness
  - Security
  - Faul Tolerance

Correctness (Corretude)

- Técnicas de **verificar e garantir a correção** de programas concorrentes e distribuídos.
- **Redes de Petri** (Carl Petri, 1970)
- **CCS** (Calculus of Communicating Systems – R. Milner, 1980)
- **CSP** (Communicating Sequential Processes – C. A. R. Hoare, 1978)
- **Lógica Temporal** (para verificar sistemas orientados a propriedades)

2.2 Modelos Arquiteturiais

- A arquitetura de um sistema é sua estrutura em termos de componentes especificados separadamente.

- A meta é garantir que a estrutura satisfará as demandas presentes e futuras sobre o sistema.

- Interesses importantes: tornar o sistema confiável, gerenciável e adaptável e de custo-efetivo (custo real).

Seção 2.2

- Principais modelos empregados em SDs:

  estilos de arquiteturas de SDs

- Modelos são construídos em termos de **processos** ou de **objetos**.

Um modelo abstrai e simplifica as funções dos componentes individuais e então considera:

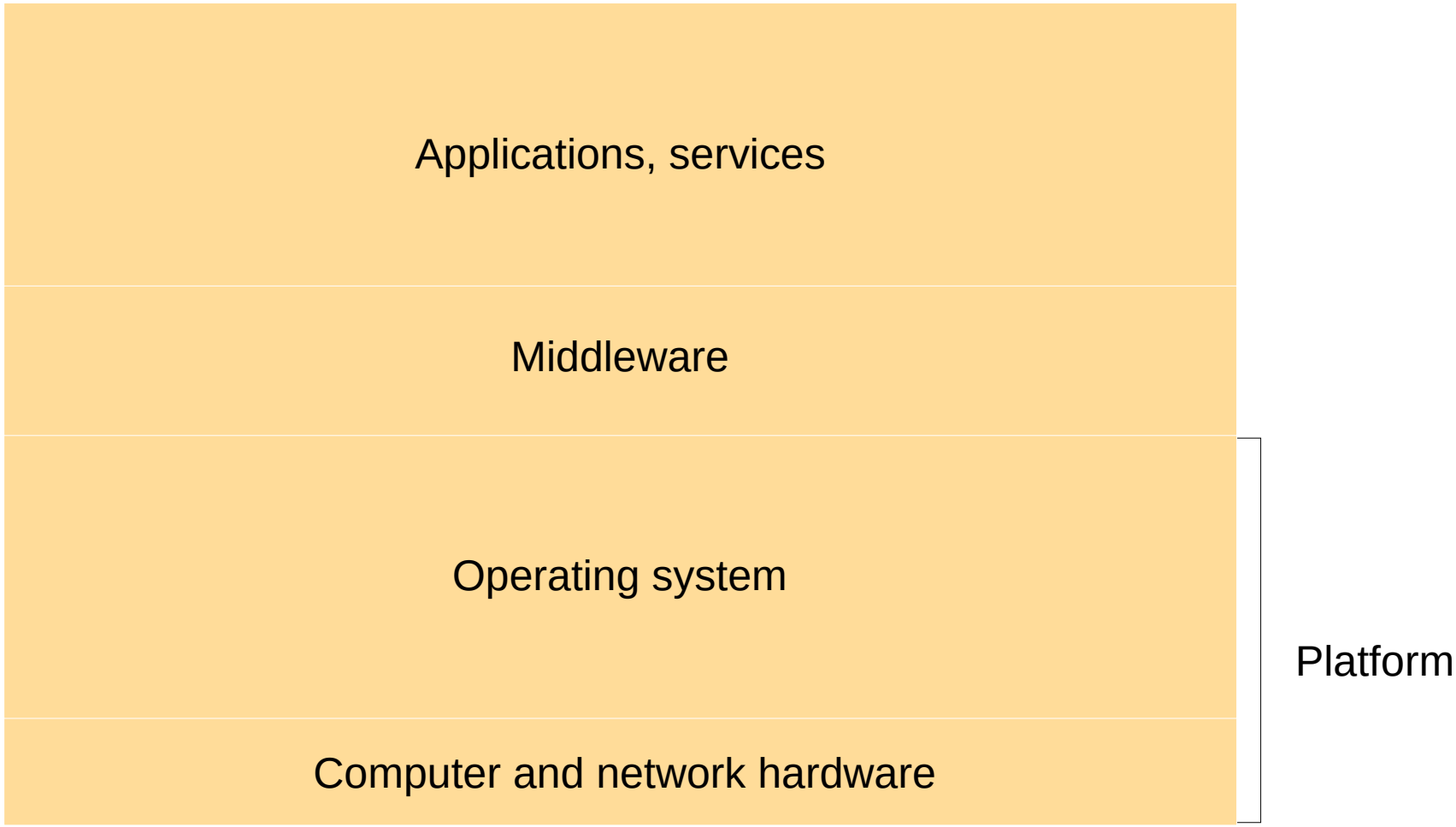
- a colocação componentes através de uma rede, buscando definir padrões úteis para a distribuição de dados e carga de trabalho;

- as inter-relações entre os componentes: seus papéis e padrões de comunicação entre eles.

Seção 2.2.1 – Camadas de Software

- Hardware de mais baixo nível e camadas de software.

Figure 2.1
Software and hardware service layers in distributed systems



Middleware

- Uma camada de software cujo propósito é mascarar heterogeneidade e prover um modelo de programação conveniente para programadores de aplicação.
- Processos ou objetos.
- Invocações de métodos, comunicações entre um grupo de processos, notificação de eventos, replicação de dados compartilhados e transmissão de dados multimídia em tempo real.

Middleware

- Java RMI (Remote Method Invocation)
- CORBA (Common Object Request Broker Architecture)

2.2.2 Arquiteturas de Sistemas

- Modelo Cliente/Servidor
- Serviços providos por múltiplos servidores.
- Servidores Proxies e Cache
- Processos Peer-to-Peer

Figure 2.2
Clients invoke individual servers

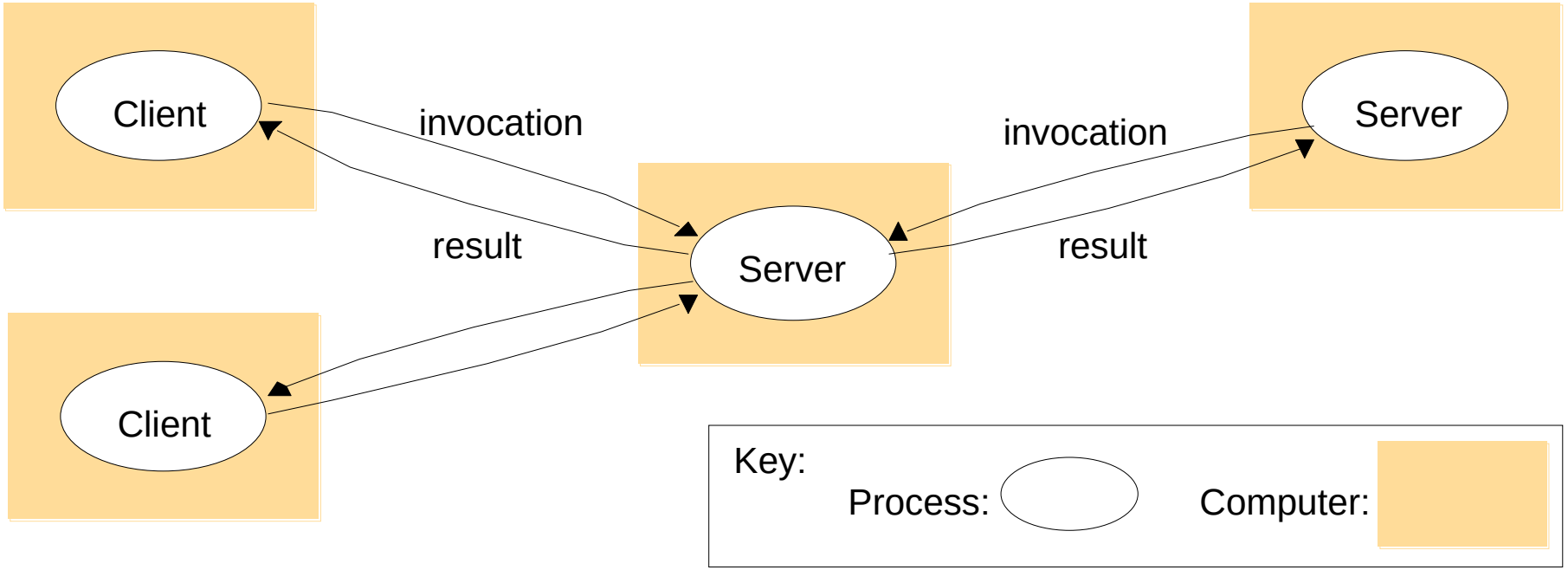


Figure 2.2

- Um servidor Web é frequentemente cliente de um servidor de arquivos que gerencia os arquivos nos quais páginas Web são armazenadas.

- Servidores Web e a maioria dos serviços Internet são clientes do serviço DNS.

Figure 2.2

- Máquinas de Busca, que habilitam usuários a procurar sumários de informação disponíveis em páginas Web em sites Internet.
- Sumários são feitos por Web Crawlers rodam em background em sites de busca.
- A máquina de busca é um servidor e um cliente. Ela responde a uma consulta de um browser e roda web crawlers que agem como clientes, fazendo requests HTTP para outros servidores Web.

Figure 2.3
A service provided by multiple servers

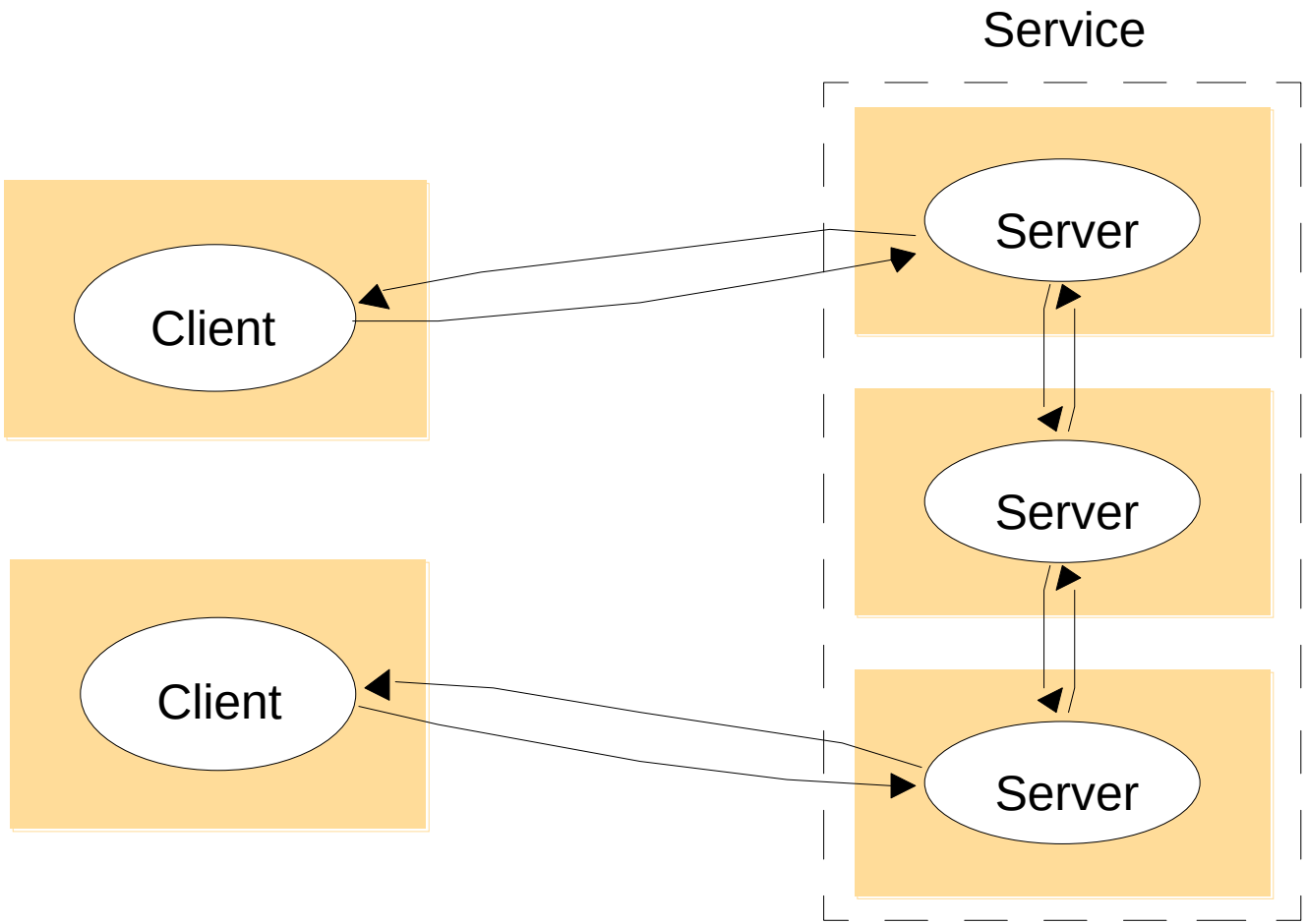


Figure 2.3

- **Serviços** podem ser implementados como diversos processo servidores, em computadores separados, interagindo quando necessário para prover um serviço a processos clientes.

Figure 2.3

- Os servidores podem **particionar** o conjunto de objetos sobre os quais o serviço é baseado e distribuído entre eles próprios, ou podem **replicar** cópias sobre os servidores.

Figure 2.3

- **Replicação** – aumentar performance e disponibilidade e melhorar tolerância a falhas.

- Serviço Web provido por *altavista.digital.com*

  é mapeado sobre diversos servidores que têm uma base de dados replicada em memória.

- Sun NIS (Network Information Services) que é usado por computadores em uma LAN, quando usuários fazem log in. Cada servidor NIS tem sua própria réplica do arquivo de senhas contendo uma lista de nomes de login de usuários e senhas criptografadas.

Figure 2.4
Web proxy server

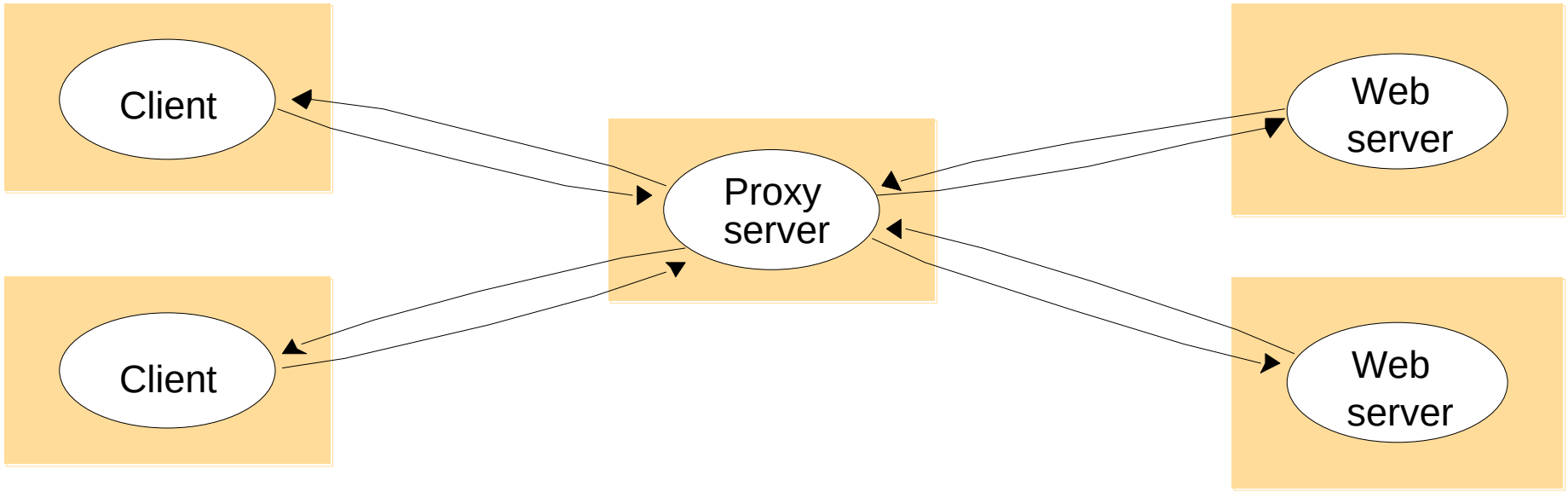
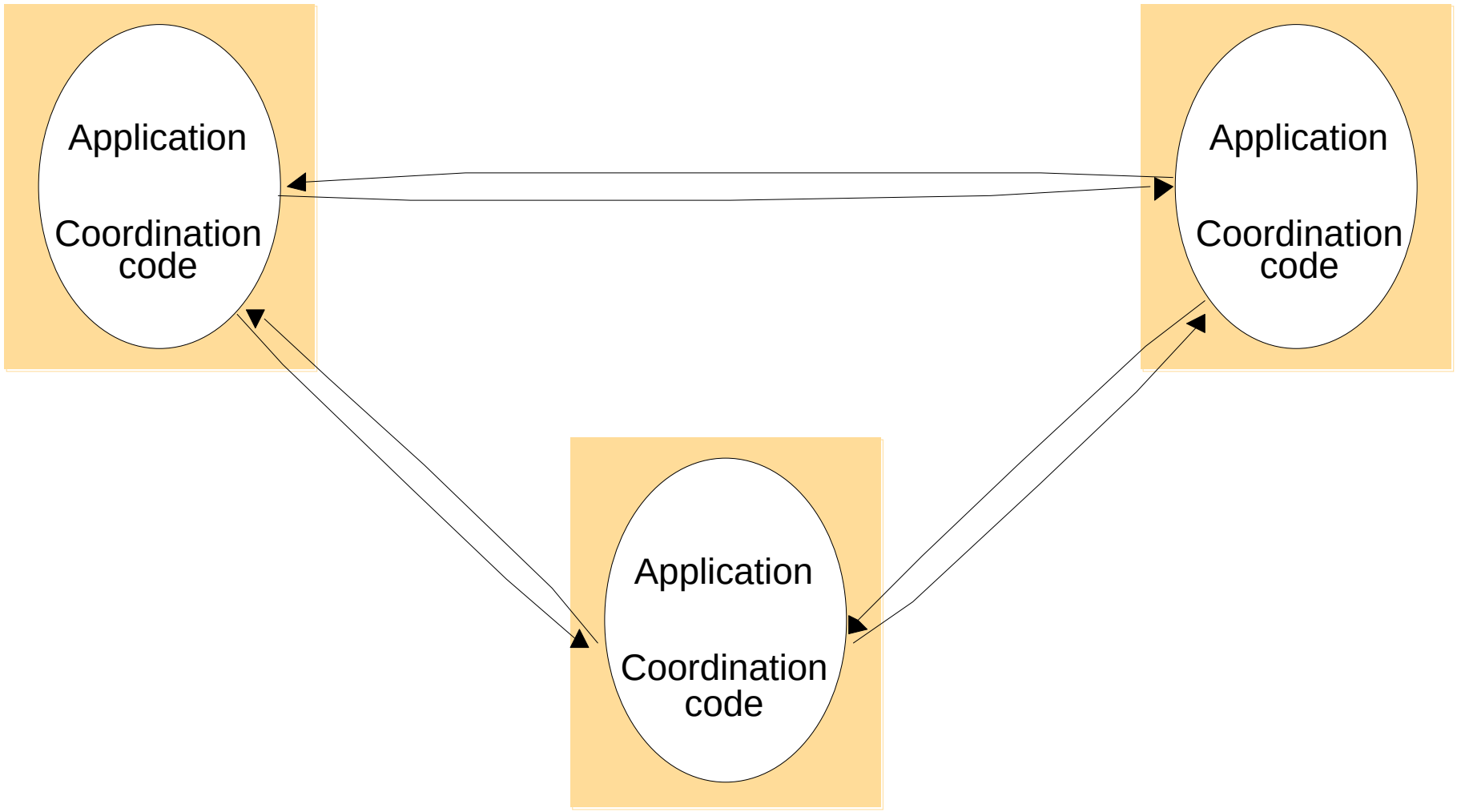


Figure 2.5
A distributed application based on peer processes

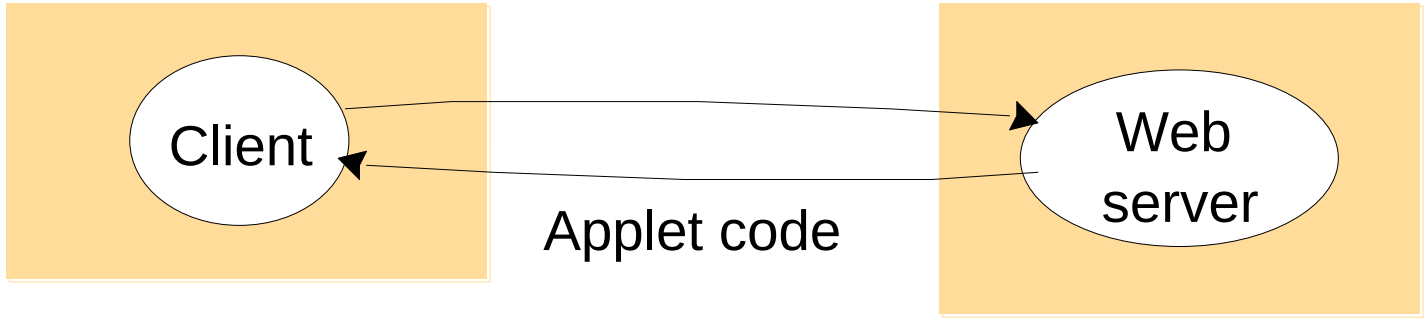


2.2.3 Variações sobre o Modelo Cliente/Servidor

- Código Móvel
- Agentes Móveis
- Thin Clients
- Dispositivos móveis e redes espontâneas

Figure 2.6
Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

Figure 2.7
Thin clients and compute servers

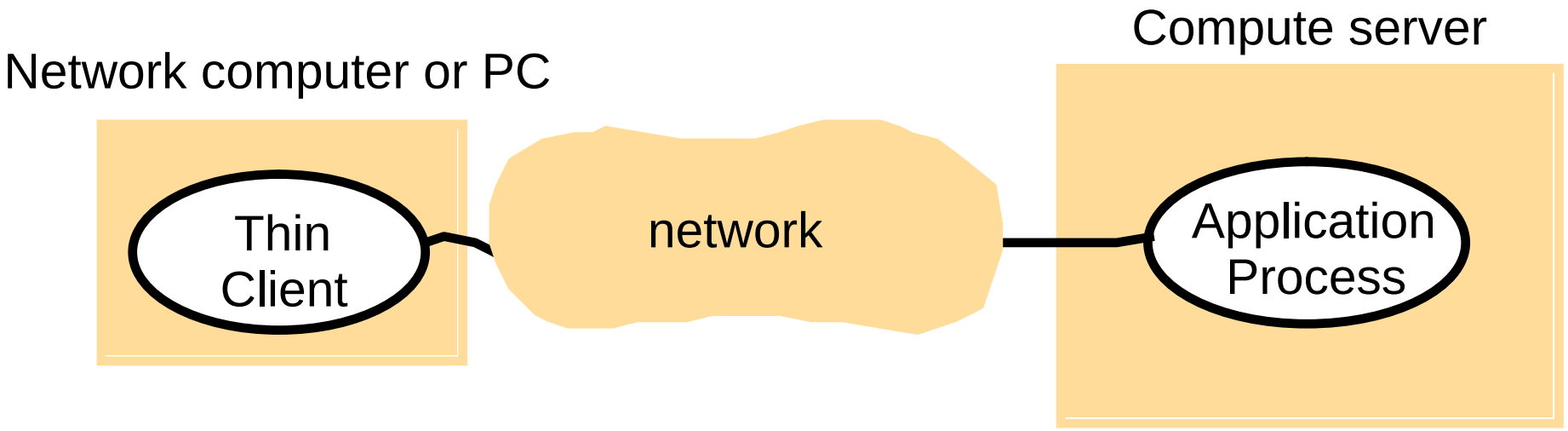
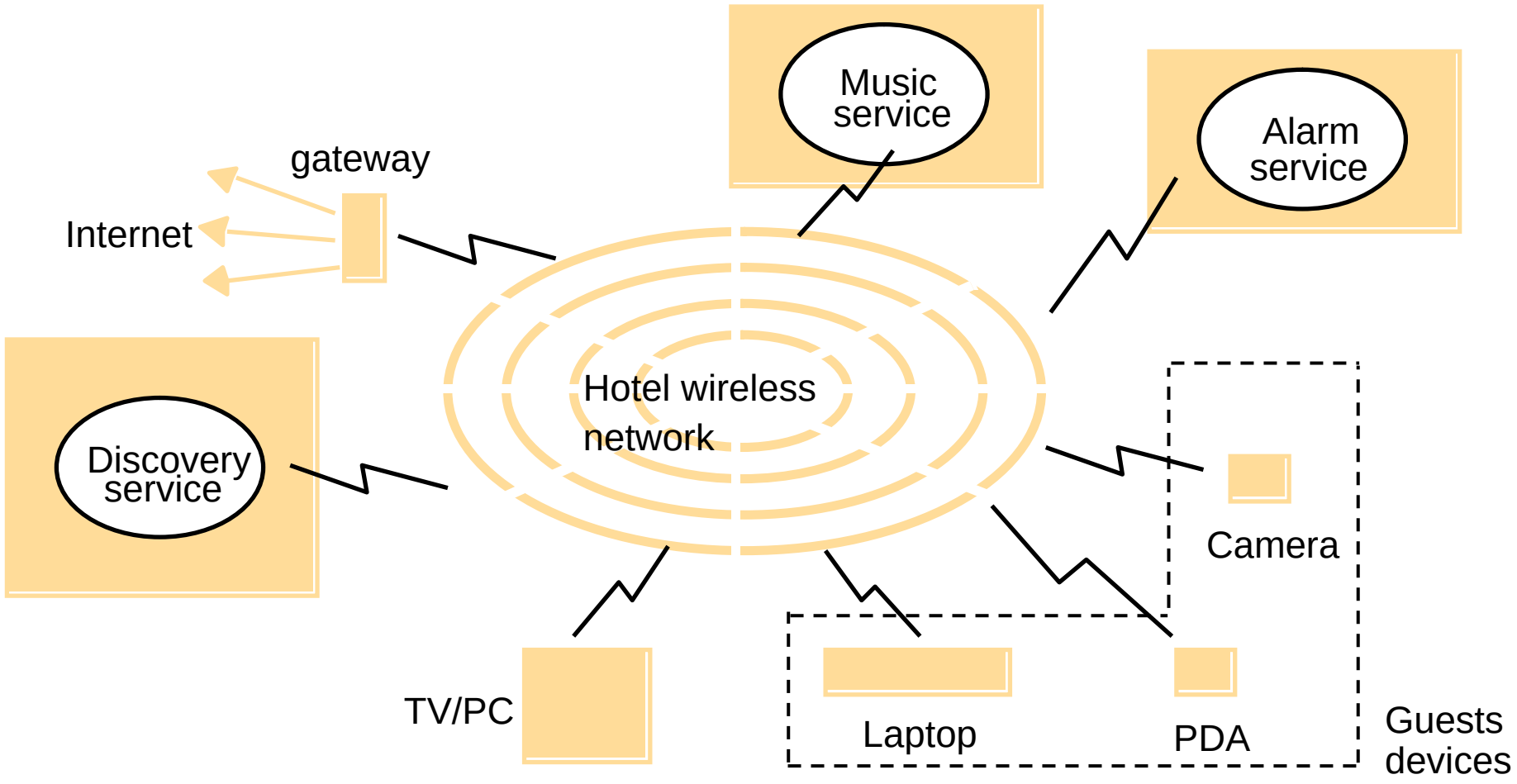


Figure 2.8
Spontaneous networking in a hotel



Spontaneous networking in a hotel

- Pequenos dispositivos de computação portáteis: laptops, handheld (PDAs), fones móveis, cameras digitais, relógios “inteligentes” e dispositivos embutidos em máquinas domésticas.

- Muitos destes dispositivos funcionam em redes sem fio, metropolitanas ou redes maiores (GSM, CDPD), em centenas de metros (WaveLAN) ou em poucos metros (Blue Tooth, infra-vermelho, HomeRF).

- Redes mais curtas tem largura de banda em torno de 10 Megabits/segundo e GSM, na ordem de centenas de kilobits/segundo.

2.2.4 Interfaces e Objetos

- Definições de Interfaces
- O conjunto de funções disponíveis para invocação em um processo ou objeto.
- C++, Java
- Java RMI, CORBA

Questões de Performance

Responsiveness: a velocidade na qual a resposta é gerada é determinada, não apenas pela carga e performance do servidor e a rede, mas também pelo atraso em todos os componentes de software envolvidos.

Throughtput: a taxa (número de *requests* que um servidor suporta numa unidade de tempo) na qual o trabalho computacional é feito.

2.2.5 Requisitos de Projeto para arquiteturas distribuídas

- **Qualidade de Serviço**

  Usuários devem ser providos com a funcionalidade que eles requerem de um serviço.

- **Cache e Replicação**

- **Dependability:** Correção, Segurança e Tolerância a Falhas.

2.3 Modelos Fundamentais

- Modelos:

Em geral, um modelo contém somente os ingredientes essenciais que necessitamos no sentido de entender e raciocinar sobre alguns aspectos do **comportamento** de um sistema.

Modelos

- Um modelo objetiva responder as seguintes questões:

1. Quais são as principais entidades no sistema?
2. Como elas interagem?
3. Quais são as características que afetam seu comportamento?

Modelos

- O propósito de um modelo é:

1. Tornar explícito todas as hipóteses relevantes do sistema que está se modelando.

2. Fazer generalizações concernentes ao que é possível ou impossível, dados aquelas hipóteses. As generalizações podem tomar a forma de algoritmos de propósitos gerais ou **propriedades desejáveis** que devem ser garantidas. Essas garantias são dependentes de análise lógica e, onde for apropriada, deve-se usar prova matemática.

Modelos

- Os aspectos de SDs que desejamos capturar nos modelos fundamentais são voltados para auxiliar na discussão e raciocínio sobre:

- Interação
- Falhas
- Segurança

Modelo de Interação

- Dois fatores significantes afetando processos interagindo em um SD:

1. Performance de comunicação é frequentemente uma característica limitante.

2. É impossível manter um única noção global de tempo.

Performance de Comunicação

- Comunicação sobre uma rede tem as características de performance relacionadas a:

1. **Latência**: O atraso entre o início da transmissão de uma mensagem a partir de um processo e o início de sua recepção por um outro processo. Latência inclui: Tempo tomado por uma string de bits, o atraso no acesso à rede e o tempo tomado nos sistemas operacionais

Performance de Comunicação

2. **Largura de Banda** de uma rede: quantidade total de informação que pode ser transmitida sobre a rede em um dado intervalo de tempo. Canais compartilhando um mesmo meio de transmissão têm de compartilhar a largura de banda disponível.

3. **Jitter** : **Variação no tempo tomado para entregar uma série de mensagens**. Relevante em SDs Multimídia: se uma amostra consecutiva de dados de áudio são escutados com diferentes intervalos de tempo, então o som escutado aparece distorcido.

Clocks de Computadores e Temporização de Eventos

- Cada computador em um SD tem seu próprio *clock* interno, o qual pode ser usado por processos locais para obterem o valor do tempo corrente.

- Portanto, dois processos rodando sobre diferentes computadores podem associar ***timestamps (rótulos de tempo)*** com seus eventos (temporização de eventos).

Clocks de Computadores e Temporização de Eventos

- Contudo, se dois processos lêem seus *clocks* ao mesmo tempo, seus *clocks* locais podem fornecer diferentes valores de tempo.

- Isto é porque *clocks* de computadores ***drift*** (diferem do tempo perfeito) e, mais importante, suas ***drift rates*** diferem uma da outra.

Clocks de Computadores e Temporização de Eventos

- O termo ***drift rate*** refere-se à quantidade relativa que um *clock* de computador difere de um *clock* de referência perfeita.

- Mesmo se os *clocks* sobre todos os computadores em um SD são estabelecidos a um mesmo tempo, inicialmente, seus clocks ***eventualmente*** variarão significativamente, a menos que correções sejam aplicadas.

Clocks de Computadores e Temporização de Eventos

- Existem diversas abordagens para corrigir os tempos sobre *clocks* de computadores. Por exemplo, computadores podem usar receptores de rádio para receber o tempo de GPS (**Global Positioning System**) com uma precisão em torno de 1 microsegundo.

- Mas, receptores GPS não operam dentro de edifícios, e nem mesmo o custo pode ser justificado para todo computador.

Clocks de Computadores e Temporização de Eventos

- Ao invés disso, um computador que tem uma fonte de tempo precisa, tal como GPS, poderia enviar mensagens para outros computadores numa rede.

- O acordo resultante entre os tempos sobre os clocks locais, contudo, poderiam ser afetados pela variável atraso das mensagens (message delays).

- Discussão mais detalhada sobre ***clock drift*** e ***clock syncronization***, ver no capítulo 11.

Modelos Fundamentais
Dois Variantes do Modelo de Interação

- Em SDs é difícil estabelecer **limites de tempo** sobre o tempo tomado para execução de processos, entrega de mensagens ou ***clocks's drifts***.

- Duas posições extremas, opostas, proporcionam um par de modelos simples: o primeiro, tendo uma forte hipótese sobre o tempo e o segundo, não fazendo nenhuma hipótese sobre o mesmo.

Sistemas Distribuídos Síncronos

- Hadzilacos e Toueg [1994] definem um SD **síncrono**, como sendo um sistema com os seguintes limites definidos:

  - o tempo para executar cada etapa de um processo tem limites inferiores e superiores conhecidos;

  - cada mensagem transmitida sobre um canal é recebida dentro de um tempo limitado conhecido;

  - cada processo tem um clock local cuja ***drift rate*** de tempo real tem um limite conhecido.

Sistemas Distribuídos Síncronos

- É possível sugerir limites inferiores e superiores para o tempo de execução de processos, retardo de mensagens e *clock drift* em um SD.

- Mas, é difícil chegar em valores realísticos e prover garantia dos valores escolhidos.

- A menos que os valores dos limites possam ser garantidos, qualquer projeto baseado sobre os valores escolhidos, não poderá ser confiável.

Sistemas Distribuídos Síncronos

- Contudo, modelando um algoritmo como um sistema síncrono, pode ser útil para dar uma idéia de como ele se comportará em um sistema distribuído real.

- Em um SD síncrono é possível usar *timeouts*, por exemplo, para detectar a falha de um processo, como mostrado na seção sobre **modelo de falhas**.

Sistemas Distribuídos Síncronos

- SDs síncronos podem ser construídos.
- O que é requerido é que os processos realizem tarefas com requisitos de recursos conhecidos para os quais eles possam ser garantidos ciclos de processador suficientes e capacidade de rede, bem como, os processos serem supridos com clocks com ***drift rates*** limitados.

Sistemas Distribuídos Assíncronos (Definição)

- Muitos SDs, por exemplo, a Internet, são muitos úteis sem serem qualificados como SDs síncronos.

- Portanto, necessitamos **um modelo alternativo**: um SD assíncrono é um sistema no qual **não existem limites** sobre:

Sistemas Distribuídos Assíncronos (Definição)

- **Rapidez na execução de processos –**

  Por exemplo, uma etapa de um processo pode tomar 1 **pico-segundo** (1seg x 10E-12) e uma outra etapa 1 **nano-segundo** (1 seg x 10E-9); tudo o que se pode dizer é que cada etapa pode levar um **tempo longo arbitrariamente**.

Relembrando Frações do Segundo

- 1 segundo;
- Décimo de segundo (10E-1);
- Centésimo de segundo (10E-2);
- Milésimo de segundo (10E-3);
- Milionésimo de segundo **(10E-6) (Micro-Segundo)**;
- Bilionésimo de segundo **(10E-9) (Nano-Segundo)**;
- Trilionésimo de segundo **(10E-12) (Pico-Segundo)**;

- **Atraso na transmissão de mensagens –**

Por exemplo, uma mensagem de um processo A para um processo B pode ser entregue em **tempo mínimo** e uma outra pode tomar alguns minutos ou horas. Em outras palavras, uma mensagem pode ser recebida após **um tempo longo arbitrário**.

Sistemas Distribuídos Assíncronos (Definição)

- **Drift Rates dos clocks –**

  A drift rate de um clock é arbitrária.

- O modelo assíncrono não permite nenhuma hipótese sobre intervalos de tempo envolvida em qualquer execução.

- Isto, exatamente modela a Internet, na qual não existe nenhum limite intrínseco sobre carga de servidores e carga de rede, e portanto, sobre quanto tempo ela leva para transferir um arquivo usando FTP. Algumas vezes uma mensagem de *email* pode levar dias para chegar

Sistemas Distribuídos Assíncronos

- Alguns problemas de projeto podem ser resolvidos mesmo com estas hipóteses.

- Por exemplo, embora a Web não possa sempre prover uma resposta particular dentro de um limite de tempo razoável, *browsers* tem sido projetados para permitir usuários fazerem outras coisas enquanto eles estão esperando.

Sistemas Distribuídos Assíncronos

- Qualquer solução que é válida para um sistema distribuído assíncrono é também válida para um sistema síncrono.

- SDs reais são **muito frequentemente assíncronos** por causa da necessidade para processos compartilharem processadores e para canais de comunicação compartilharem a rede (o meio de transmissão).

Sistemas Distribuídos Assíncronos

- Por exemplo, se vários processos de caráter desconhecido estão compartilhando um processador, então a **performance** resultante de qualquer um deles não pode ser garantida.

- Mas, existem muitos problemas de projeto que não podem ser resolvidas para SDs Assíncronos, mas que podem ser resolvidas quando alguns **aspectos de tempo** são usados.

- Por exemplo, a necessidade de um ***stream* de dados multimídia** para ser entregue **antes** do seu ***deadline*** é um desses problemas. Para problemas tais como este, **um modelo sincronizado** é requerido.

- A impossibilidade de se **sincronizar *clocks*** em um **sistema assíncrono**:

Ordenação de Eventos (Event Ordering)

- Em muitos casos estamos interessados em saber se um evento (envio ou recebimento de uma mensagem) em um processo, **ocorreu antes**, **após ou concorrentemente** a um outro evento em um outro processo.

- A execução de um SD pode ser descrita em termos de **eventos e sua ordenação**, apesar da falta de *clocks* precisos.

Exemplo de Ordenação de Eventos (Event Ordering)

- Considere um grupo de usuários em uma lista de emails: X, Y, Z e A.

1. Usuário X envia um mensagem com o assunto *Meeting*;

2. Usuário Y e Z respondem enviando uma mensagem com o assunto *Re:Meeting*;

Exemplo de Ordenação de Eventos (Event Ordering)

- Em tempo real, a mensagem de X foi primeiro enviada, Y lê ela e responde; Z lê ambas as mensagens de X e Y e então envia uma outra resposta que referencia as mensagens de X e de Y.

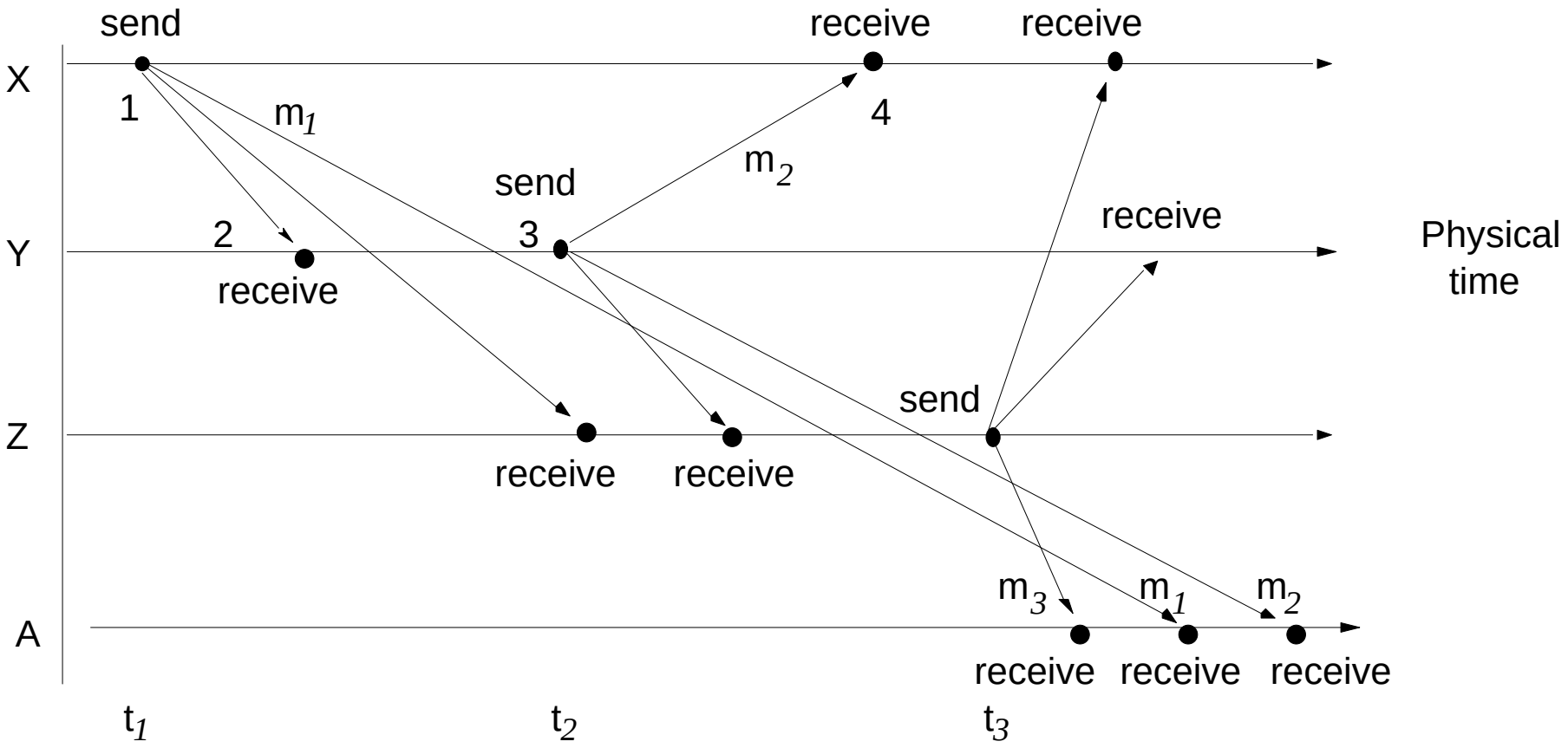
- Mas, devido aos **atrasos nas entregas das mensagens**, essas podem ser entregues como na **Figure 2.9** no próximo slide, e alguns usuários poderiam ver estas mensagens de resposta (replies) na ordem errada, como por exemplo, o usuário A poderia ver na Figure 2.9 a seguir:

Exemplo de Ordenação de Eventos (Event Ordering)

- Inbox do usuário A:

From:	Subject:
Z	Re:Meeting
X	Meeting
Y	Re:Meeting

Figure 2.9
Real-time ordering of events (Exemplo em uma Mailing List)



Ordenação de Eventos (Event Ordering)

- Se os clocks dos computadores de X, Y e Z pudessem estar sincronizados, então cada mensagem poderia portar o tempo do clock do computador local (um rótulo de tempo), quando ela fosse enviada.

- Por exemplo, mensagens m1, m2 e m3 poderiam portar os tempos t1, t2 e t3, onde t1<t2<t3.

Ordenação de Eventos (Event Ordering)

- As mensagens recebidas seriam entregues aos usuários de acordo as suas **ordens de tempo**.

- Se os clocks são aproximadamente sincronizados, então esses rótulos de tempo, **frequentemente** estarão na ordem correta.

Ordenação de Eventos (Event Ordering)

- Dado que clocks não podem ser sincronizados perfeitamente através de um SD, Lamport [1978] propôs um **modelo de tempo lógico** que pode ser usado para prover uma ordenação entre vários eventos em processos rodando em diferentes computadores em um SD.

Ordenação de Eventos (Event Ordering)

- **Tempo lógico** permite a ordem na qual as mensagens são enviadas/recebidas, sem recorrer a clocks físicos das máquinas.

- Como o **modelo de tempo lógico** pode ser aplicado ao Exemplo da Lista e Emails?

Ordenação de Eventos (Event Ordering) usando o Modelo de Tempo Lógico

- Exemplo: Lista de Emails

Logicamente, sabemos que uma mensagem é recebida após ela ser enviada, e portanto podemos estabelecer uma ordem lógica de pares de eventos, mostrada na Figure 2.9, considerando somente os eventos relacionados a X e Y:

X envia m1 antes de Y receber m1;

Y envia m2 antes de X receber m2.

Ordenação de Eventos (Event Ordering)

- Também sabemos que as respostas (replies) são enviadas após receber mensagens, portanto, temos a seguinte ordem lógica para Y:

  Y recebe m1 antes de enviar m2.

- O **Modelo de Tempo Lógico** atribui um número a cada evento, correspondendo a sua ordenação lógica, de modo que eventos ocorrendo mais tarde têm números mais altos do que os eventos que ocorrem mais cedo. Por exemplo, a Figure 2.9 mostra os números de 1 a 4 sobre os eventos em X e Y (representando a ordem dos eventos).

Modelo de Falhas

- Em um SD, os processos e canais de comunicação podem falhar – podem afastar-se do que é considerado ser o comportamento desejável.

- O modelo de falhas define os modos nos quais, falhas podem ocorrer, para prover um entendimento dos efeitos das falhas.

- Hardzilacos e Toueg [1994] definiram uma taxionomia que distingue entre as falhas de processos e falhas dos canais de comunicação.

- Tais falhas são apresentadas, caracterizando falhas por omissão, falhas arbitrárias e falhas de temporização.

Falhas por Omissão (Omission Failures)

- Refere-se a casos quando um processo ou canal de comunicação falha (fails) para realizar ações que se supõem fazerem.

Falha por Omissão
Falha por Crash

- A principal **falha por omissão** de um processo é a falha por *crash*.

- Um processo falha por ***crash***, quando ele pára e não mais pode executar quaisquer etapas de seu programa.

Falhas por Omissão (Processos)

- O projeto de serviços que podem sobreviver na presença de *faults*, pode ser simplificado, se no projeto pode ser assumido que os serviços providos por processos, poderão entrar em *crash*. Isto é, ou os processos funcionam corretamente ou eles param.

Falhas por Omissão (Processos)

- Outros processos podem detectar um *crash*, pelo fato que o processo repetidamente falha para responder a mensagens de invocação.

Falhas por Omissão (Processos)

- Neste caso, o método de detecção do *crash*, é baseado no uso de *timeouts* – isto é, um método no qual, um processo permite um período de tempo fixado para alguma coisa ocorrer.

Falhas por Omissão (Processos)

- Em um sistema distribuído assíncromo, um *timeout* pode indicar somente que um processo não está respondendo. Ele pode estar em *crash*, ou pode estar lento, ou as mensagens dele podem não ter chegado.

Falhas por Omissão (Processos)

- Um processo em *crash* é chamado *fail-stop*, se outros processos podem detectar que o processo está em *crash*.

Falhas por Omissão (Processos)

- O comportamento *fail-stop* pode ser produzido em um SD síncrono, se os processos usam *timeouts* para detectar quando outros processos falham para responder, e mensagens são garantidas serem entregues.

Falhas por Omissão (Processos)

- Um exemplo:

  Se processos *p* e *q* são programados para o processo *q* responder a uma mensagem de *p*, e se o processo *p* não recebe nenhuma resposta de *q*, em um tempo máximo medido no *clock* local de *p*, então o processo *p* pode concluir que o processo *q*, falhou (has failed).

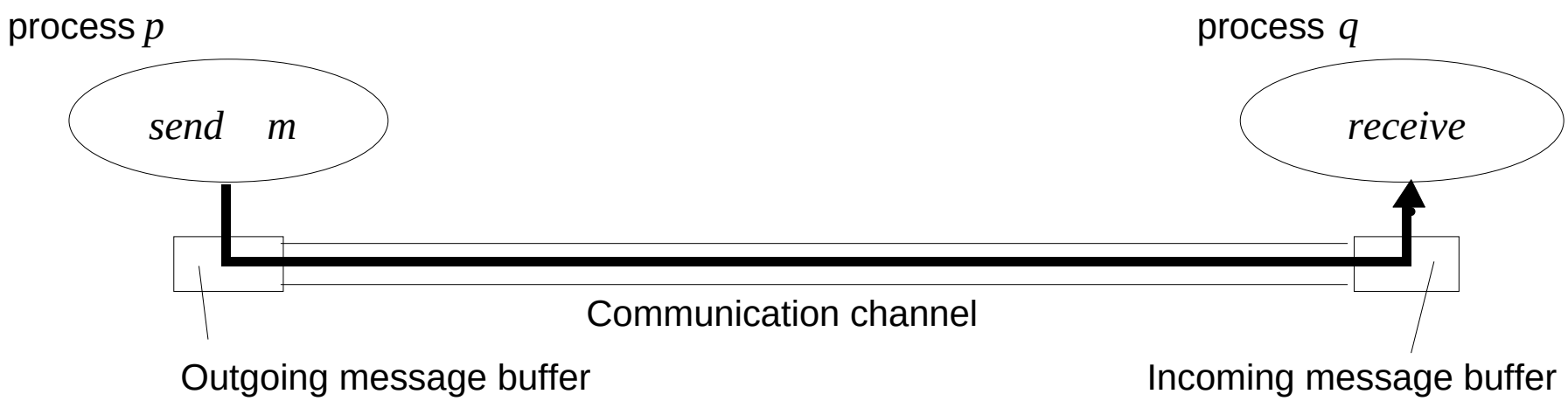
Falhas por Omissão (Comunicação)

- Considere as primitivas de comunicação *send* e *receive*.

- Um processo *p* executa um *send*, inserindo uma mensagem *m* em seu buffer de mensagens saindo. O canal de comunicação transporta *m* para o buffer de mensagens chegando do processo *q*. O processo *q* executa um *receive*, tomando *m* de seu buffer e entregando a mensagem *m.*

- Os buffers são providos pelo sistema operacional.

Figure 2.10
Processes and channels



Falhas por Omissão (Comunicação)

- O canal de comunicação produz uma falha por omissão, se ele não transporta a mensagem *m*, do buffer de mensagens saindo de *p*, para o buffer de mensagens chegando de *q.*

Falhas por Omissão (Comunicação)

- Isto é conhecido como *"dropping messages"* e é geralmente causado pela falta de espaço no buffer do receptor, ou em um *gateway* intermediário, ou por um erro de transmissão detectado por *cheksum*.

Falha por Omissão de Trasmissão

- Hadzilacos e Toueg [1994] referem-se a perda de mensagens entre o processo transmissor e o buffer de mensagens saindo (*outgoing message buffer*) como falhas por omissão de transmissão (*send-omission failures*).

Falha por Omissão de Recebimento

- A perda de mensagens entre o buffer de mensagens chegando (*incoming message buffer*) e o processo recebendo, como falhas por omissão de recebimento.

Falha por Omissão no Canal

- Perda de mensagens no canal de comunicação (depois de sair do *outgoing message buffer* e antes de chegar no *incoming message buffer*), é chamada de falhas por omissão no canal (*channel omission failures*).

Detecção de Falhas

- Em um **sistema distribuído assíncrono**, nenhum dos processos pode distinguir se o outro está na presença de falha, ou se o tempo para as mensagens se propagarem no canal é muito longo.

Detecção de Falhas

- Nos **sistemas assíncronos**, não existem limites sobre as velocidades de execução dos processos, quanto aos atrasos na transmissão de mensagens e sobre as *clock drift rates*.

Detecção de Falhas

- Em **sistemas distribuídos síncronos**, um processo pode distinguir, com certeza, se outro está na presença de falha.

Detecção de Falhas

- Nos sistemas distribuídos síncronos existem limites definidos sobre: o tempo de execução em cada etapa de um processo, sobre cada mensagem transmitida sobre um canal, e sobre a *clock drift rate* do clock local de cada processo.

Falhas Arbitrárias (Arbitrary Failures)

- O termo **falha arbitrária** é usado para descrever o pior dos casos possíveis na semântica de falhas, no qual qualquer tipo de erro pode ocorrer.

Falhas Arbitrárias (Arbitrary Failures)

- Exemplo:

  Um processo pode estabelecer valores errados em seus itens de dados, ou podem retornar um valor errado em resposta a uma invocação.

Falhas Arbitrárias (Arbitrary Failures)

- Um falha arbitrária de um processo, é uma falha na qual ele arbitrariamente omite etapas de processamento intencionadas (*intended*) ou toma etapas de processamento não intencionadas (*unintended*).

Falhas Arbitrárias (Arbitrary Failures)

- Falhas arbitrárias não podem ser detectadas, verificando-se se o processo responde a invocações, porque ele poderia arbitrariamente omitir responder.

Falhas Arbitrárias (Arbitrary Failures)

- Exemplos:

Canais de comunicação podem sofrer de falhas arbitrárias:

- o conteúdo de mensagens pode ser corrompido,
- ou mensagens não-existentes podem se entregues,
- ou mensagens reais podem ser entregues mais de uma vez.

Falhas Arbitrárias (Arbitrary Failures)

- Falhas arbitrárias de canais são raras, porque o software de comunicação (protocolos) é capaz de reconhecer falhas e rejeitar as mensagens com falha.

- *Checksums* são usados para detectar mensagens corrompidas,

- Numeração das mensagens são usadas para detectar mensagens não existentes ou mensagens duplicadas.

Figure 2.11
Omission and arbitrary failures

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.

Severidade (severity)

- Falhas podem ser categorizadas por sua severidade.
- Todas as falhas descritas até aqui são falhas benignas.
- A maior parte das falhas em SD são benignas.
- Falhas Benignas: Omissão, Temporização e Performance.

Falhas de Temporização (Timing Failures)

- São aplicadas a Sistemas Distribuídos Síncronos, onde os limites de tempo são estabelecidos para tempo de execução de processos, tempo de entrega de mensagens e *clock drift rate*.

Figure 2.12
Timing failures (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.

Falhas de Temporização

- Qualquer dessas falhas pode resultar em respostas não disponíveis a clientes, dentro de um intervalo de tempo especificado.

- Em um Sistema Distribuído Assíncrono, um servidor sobrecarregado pode responder lentamente demais, mas não se pode dizer que o servidor tem uma falha de temporização, dado que nenhuma garantia é oferecida pelas características desse tipo de sistema.

Falhas de Temporização

- Sistemas operacionais de tempo-real são projetados a prover garantias de temporização (envolvendo restrições de tempo de resposta), mas sendo mais complexos para projetar, podem requerer redundância de hardware.

Falhas de Temporização

- Temporização é mais relevante em sistemas multimídia, com áudio e canais de vídeo.

- Informações de vídeo podem requerer uma quantidade muito grande de informação sendo transmitida.

Falhas de Temporização

- Para entregar tal informação sem falhas de temporização, o sistema é capaz de fazer muitas demandas especiais (aqui entra o entendimento de Sistemas Multimídias Distribuídos) sobre, ambos, o sistema operacional e o sistema de comunicação.

Mascarando Falhas
(Masking Failures)

- É possível construir serviços confiáveis, mas na presença de componentes que exibem falhas.

- Exemplo: múltiplos servidores que armazenam réplicas de dados, podem continuar a prover um serviço, quando um dos servidores entra em *crash*.

Mascarando Falhas (Masking Failures)

- Um conhecimento das características da falha de um componente pode habilitar um novo serviço ser projetado para mascarar a falha dos componentes sobre os quais o a falha pode ocorrer.

Mascarando Falhas (Masking Failures)

- Esse serviço mascara uma falha, ou por ocultá-la completamente, ou por convertê-la em um tipo de falha mais aceitável.

- Exemplo do último caso:

Checksums são usados para mascarar mensagens corrompidas, convertendo uma falha arbitrária numa falha por omissão, a qual pode ser ocultada usando-se um protocolo que retransmite mensagens que não chegam a seu destino.

Mascarando Falhas (Masking Failures)

- Replicação, por redundância de hardware, é um meio de mascarar falhas, pela substituição automática de um servidor em *crash*, por outro em funcionamento pleno.

- Modelos de Replicação: Passivo e Ativo.

Confiabilidade de Comunicação Um-a-Um

- Embora um canal de comunicação básico possa exibir falhas por omissão, é possível usá-lo para construir um serviço de comunicação que mascare algumas das falhas.

Confiabilidade de Comunicação Um-a-Um

- O termo comunicação confiável é definido em termos de validade e integridade como segue:

- Validade: qualquer mensagem no *outgoing message buffer* é **eventualmente** (num tempo finito mas não especificado) entregue ao *incoming message buffer*.

- Integridade: a mensagem recebida é idêntica a aquela enviada, e nenhuma mensagem é entregue duas vezes.

Confiabilidade de Comunicação Um-a-Um

- As ameaças para a **integridade** vem de duas fontes independentes:

  1 - Qualquer protocolo que retransmite mensagens, mas não rejeita uma mensagem que chega duas vezes.

  - Protocolos podem prover números de sequência de mensagens, e assim como detectar aquelas que são entregues duas vezes.

Confiabilidade de Comunicação Um-a-Um

- A segunda fonte:

2. Usuários maliciosos que podem injetar mensagens espúrias, repetem mensagens ou alteram mensagens.

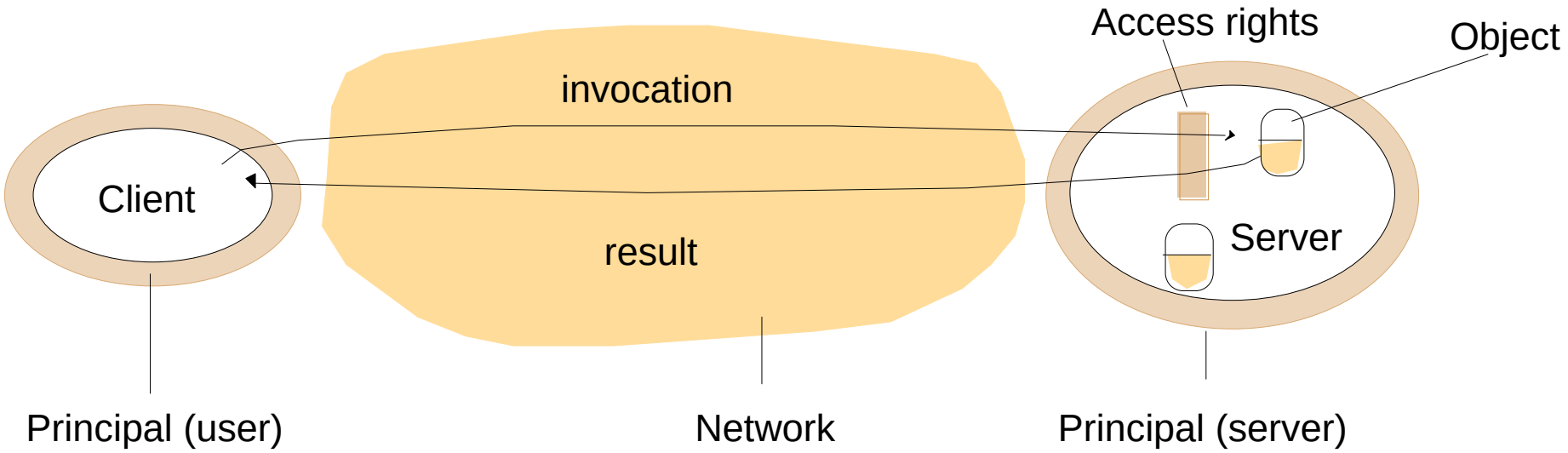
2.3.3 - O MODELO DE SEGURANÇA

Modelo de Segurança

- O modelo arquiteturial provê a base para o modelo de segurança.

- A segurança de um SD pode ser alcançada por prover segurança aos processos e aos canais usados para interações e prover proteção aos recursos do sistema contra acessos não autorizados, no sentido de se obter privacidade, confidencialidade, integridade, autenticidade e disponibilidade.

Figure 2.13
Objects and principals

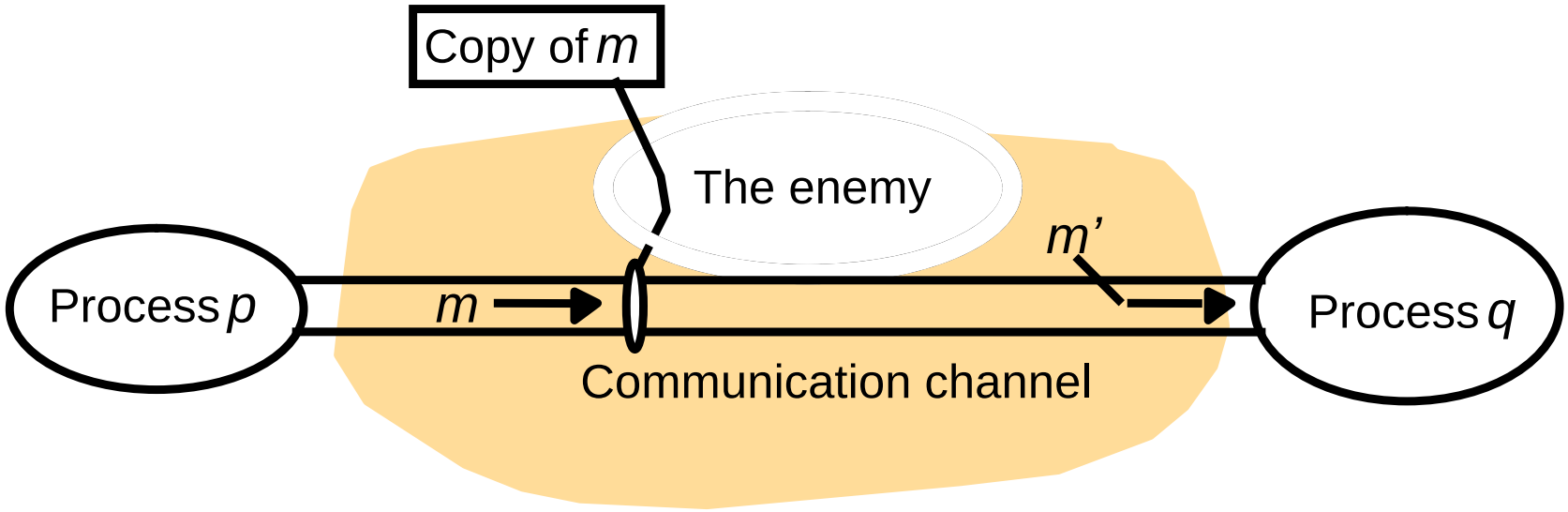


Ameaças

- Para modelar ameaças de segurança, é postulado um inimigo como na Figura 2.14.

- Ameaças de um inimigo potencial são discutidas a partir das ameaças a processos, ameaças aos canais e “denial of service” (recusa de serviço).

Figure 2.14
The enemy



Ameaças a processos

- Falsificação da identidade do processo-fonte.
- Falsificação de endereço IP.

Ameaças aos canais

- É a tentativa de salvar cópias de mensagens e repetí-las em tempos mais tarde, tornando possível a reutilização da mesma mensagem outra vez.

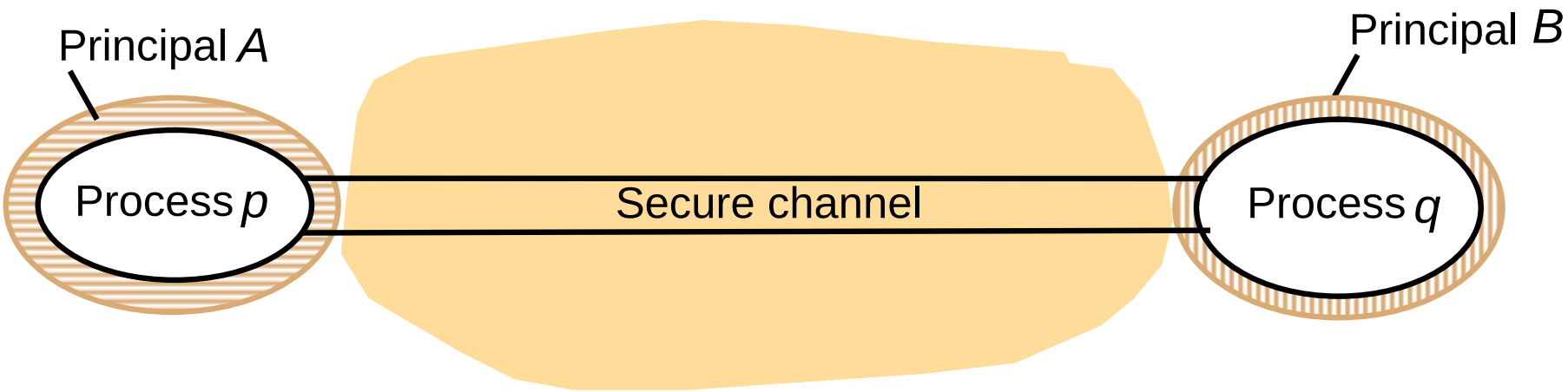
Código Móvel

- Problema para qualquer que receba e execute código de programa de outro lugar transferido pela rede.
- Cavalos de Tróia. Aplets em Java.
- Agentes móveis de software em geral.

Dificultando ameaças

- Políticas de Segurança (Modelo de Segurança).
- Criptografia (simétrica, assimétrica).
- Métodos de autenticação (assinaturas e certificados).
- Uso de canais seguros (SSL, TLS, IPSec, VPN).

Figure 2.15
Secure channels



O uso de modelos de segurança

- Implementação de segurança requer processamento substancial e gerenciamento de custos.

- O modelo provê a base para análise e projeto de SDs seguros, nos quais esses custos são levados a um mínimo.

Modelo de Ameaças

- Mas, as ameaças surgem em muitos pontos, e uma cuidadosa análise feita a partir de todas as possíveis fontes, no ambiente de rede, ambiente físico e ambiente humano.

Tarefa 2: Lista de Exercícios

- Escolher ou fazer todos os exercícios listados no capitulo 2. Tente respondê-los com suas próprias palavras e sua organização.

- Caso não consiga, recorra ás respostas dos autores. Mas, lembre as vezes as respostas dos autores não estão tão claras e necessitamos reescrever com nosso sentimento.

Modelo de Ameaças

- Essa análise envolve a construção de um Modelo de Ameaças listando todas as possíveis formas de ataque, seus riscos e seus impactos.

- Comparar os custos das técnicas de segurança contra os impactos.