Sistemas Distribuídos – 1º semestre de 2021

Professor Mestre Fabio Pereira da Silva

Desafios para a implementação de Sistemas Distribuídos

- Escalabilidade
- Abertura
- Tolerância a Falhas
- Segurança
- Transparência

Desafios para a implementação de Sistemas Distribuídos

Aspecto avaliado	Definição
Escalabilidade	Capacidade de o sistema se manter a funcionar de forma correta e à velocidade desejada independentemente do número de utilizadores.
Tolerância a Falhas	Tolerar uma falha significa conter os seus efeitos de forma a que o sistema continue a funcionar.
Segurança	Garantir a confidencialidade da informação, integridade dos dados e disponibilidade do sistema.
Transparência	O sistema deve ser visto como um todo e não como uma coleção de componentes distribuídos.
Tolerância a Falhas	Tolerar uma falha significa conter os seus efeitos de forma a que o sistema continue a funcionar.

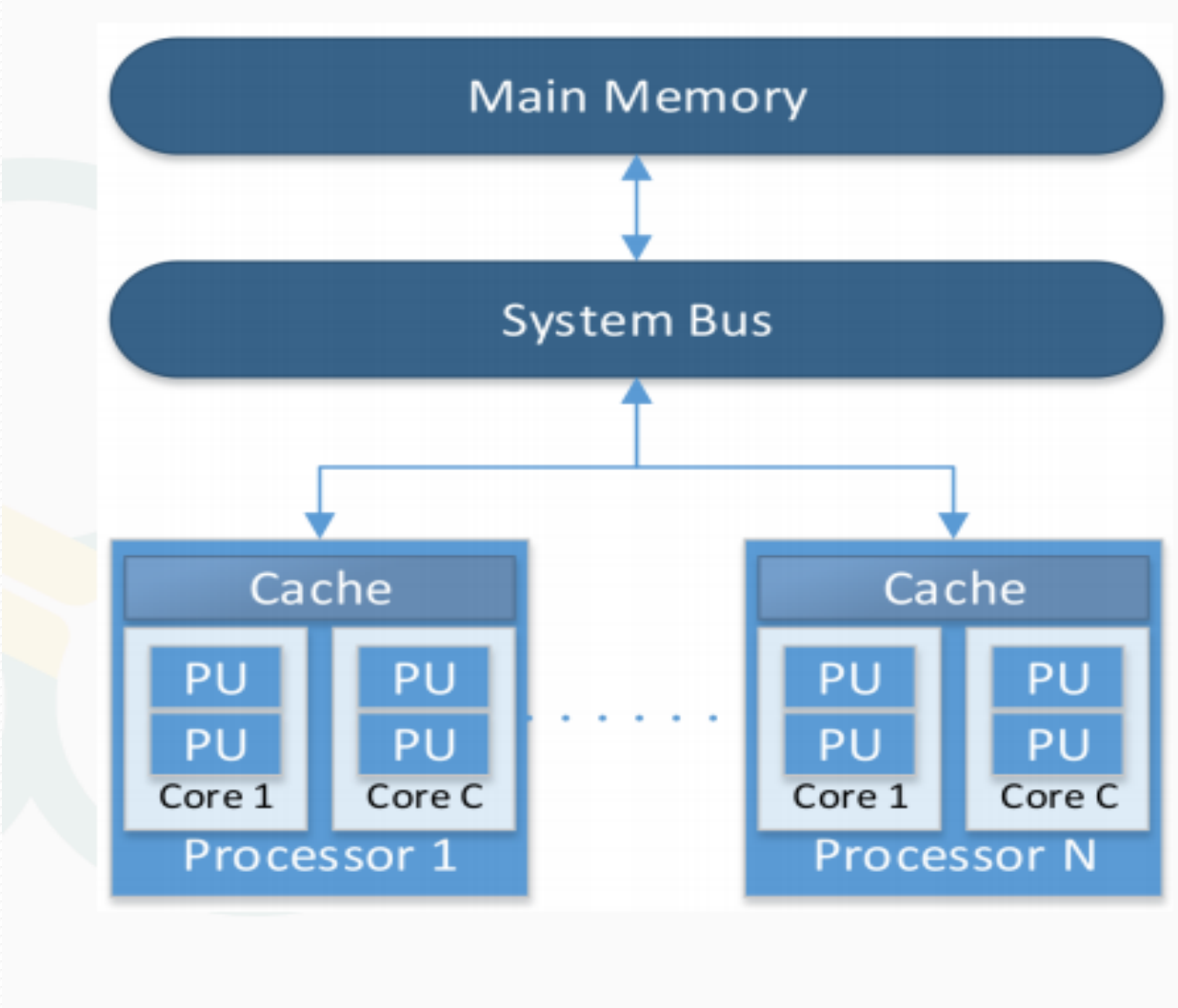
Introdução

- Existência de múltiplos processadores implica em aspectos de gerenciamento de processos que não são estudados no contexto de sistemas operacionais tradicionais

Computadores de memória compartilhada

- Todas as posições de memória podem ser acessadas por todos as unidades de processamento
  - Espaço de endereçamento único
- Symmetric Multiprocessing Platforms (SMP) são arquiteturas de memória compartilhada que possuem ao menos duas unidades de processamento
- UMA - Uniform Memory Access

Computadores de memória compartilhada

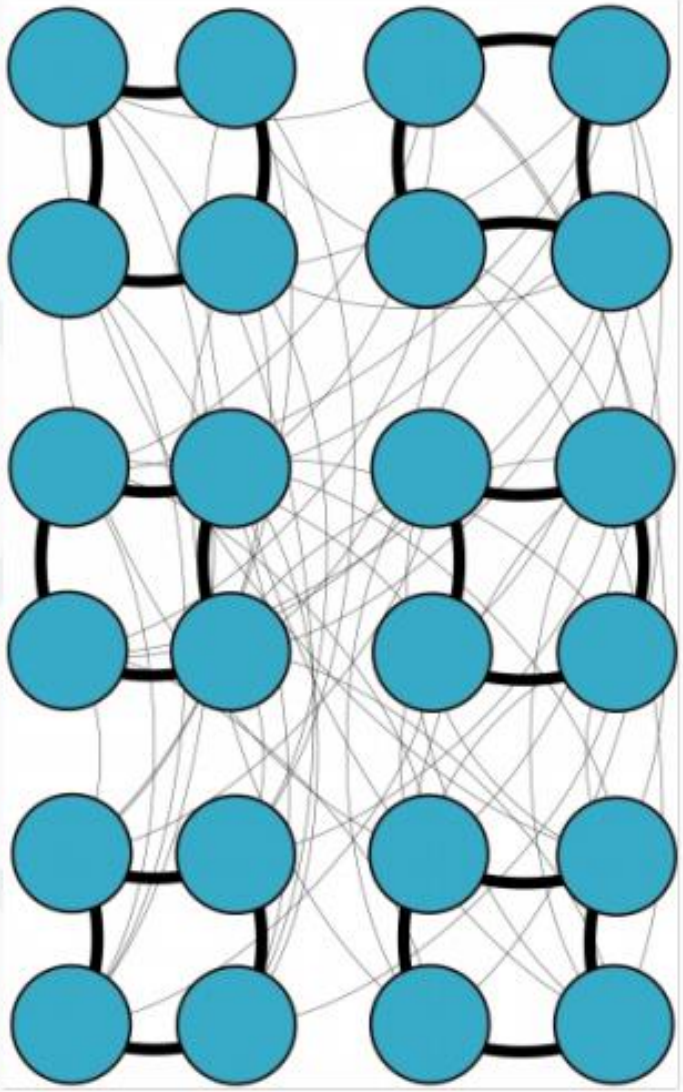


Arquitetura NUMA

- NUMA – Non-Uniform Memory Access
- Pode-se imaginar uma máquina NUMA como sendo um conjunto de máquinas UMA ligadas por uma rede de interconexão de alto desempenho
- Espaço de endereçamento único
- Desempenho para acesso à memória variável

Topologia NUMA

Conhecimento da topologia é essencial para desenvolver programas com um bom desempenho



SGI Altix UV2000 – Hipercubo (5D) com conexões privilegiadas

Processos e Threads

- Ideia básica Construir um processador virtual com software, em cima dos processadores físicos:
- processador: (hardware) provê um conjunto de instruções junto com a capacidade de executá-las automaticamente.
- processo: (software) um processador em cujo contexto pode ser executado uma ou mais threads. Executar um thread significa executar uma série de instruções no contexto daquele processo.

Processos e Threads

- thread: (software) um processador mínimo com um contexto que possui uma série de instruções que podem ser executados. Gravar o contexto de um thread implica parar a execução e guardar todos os dados necessários para continuar a execução posteriormente

Processos e Threads

- Processo: ambiente de execução
  - Espaço de endereçamento
  - Recursos para sincronização e comunicação entre threads
  - Recursos de alto nível (arquivos abertos, janelas)
- Thread: atividade (“linha” de execução)
  - Uma ou múltiplas threads em um processo –
  - Compartilham o espaço de endereçamento de um processo
  - Sem proteção entre threads

Processos

- Uma das abstrações mais importantes de um SO
  - Representam a execução de um programa
  - Execuções simultâneas de um programa são representadas por diversos processos
- Por segurança, os espaços de memória de cada processo são isolados
- Evita problemas que seriam causados por ataques deliberados ou por bugs
- Cada processo é uma linha de execução independente escalonada pelo SO

Concorrência

- Mais de um processo em execução a cada instante:
  - Atividades separadas de usuários
  - Independência de recursos
  - Localização de processos servidores em computadores distintos
- Acesso concorrente a recursos compartilhados requer sincronização

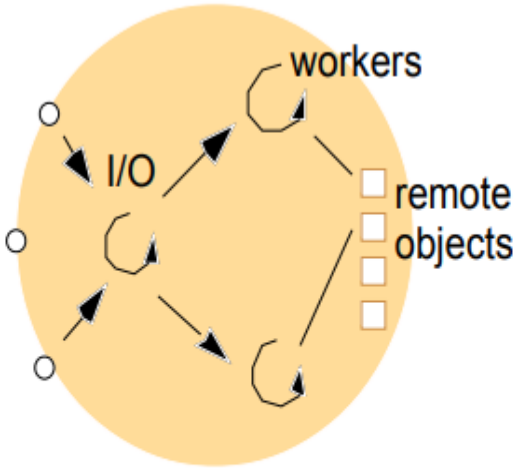
Threads

- Frequentemente um mesmo processo necessita fazer mais de uma coisa por vez. Ex.: Navegador de Internet
- Da mesma maneira que processos fornecem múltiplas linhas de execução em uma máquina, threads permitem múltiplas linhas de execução em um só processo
- Como efetivamente todos os threads são o mesmo processo, todos têm acesso ao mesmo espaço de memória e a todos os recursos disponíveis a este processo

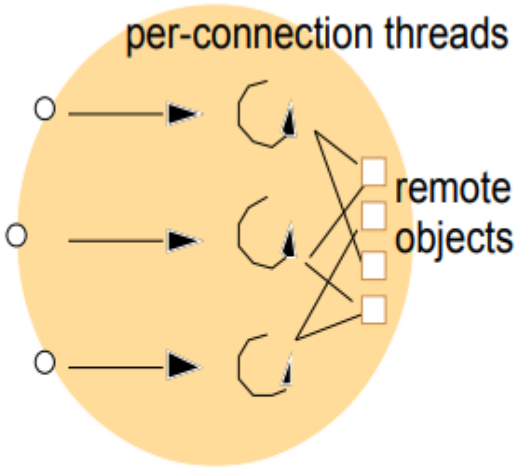
Threads

- O estado de um processo é, na verdade, o estado do thread principal
- Estados de um thread
  - Pronto
  - Executando
  - Bloqueado
  - Terminado
- Hardware
  - Monoprocessador
  - Concorrência entre threads
- Multiprocessador com memória compartilhada
  - Concorrência e paralelismo entre threads

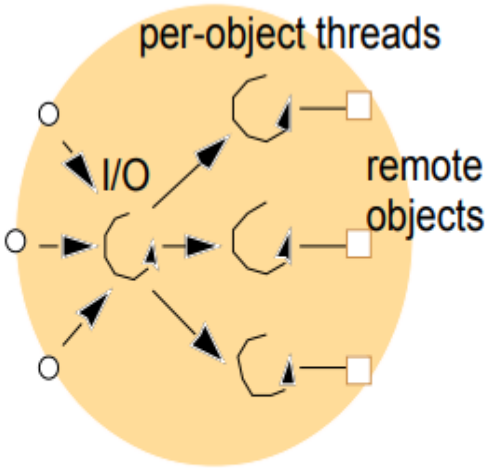
Threads



a. Thread-per-request



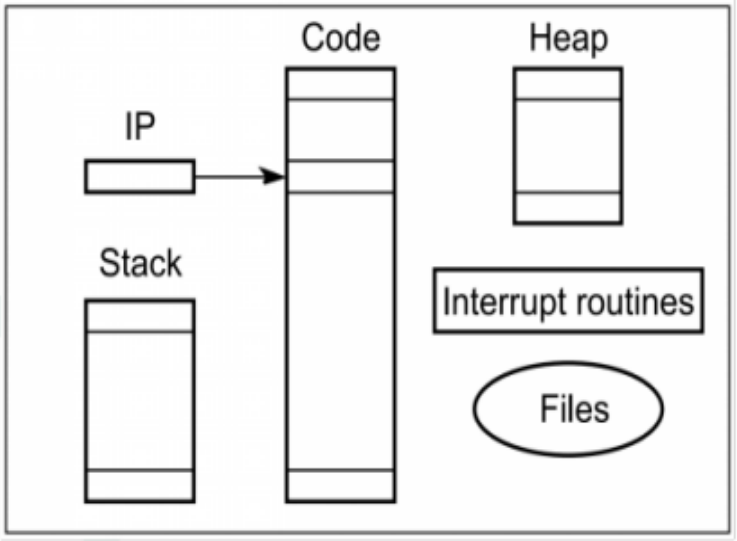
b. Thread-per-connection



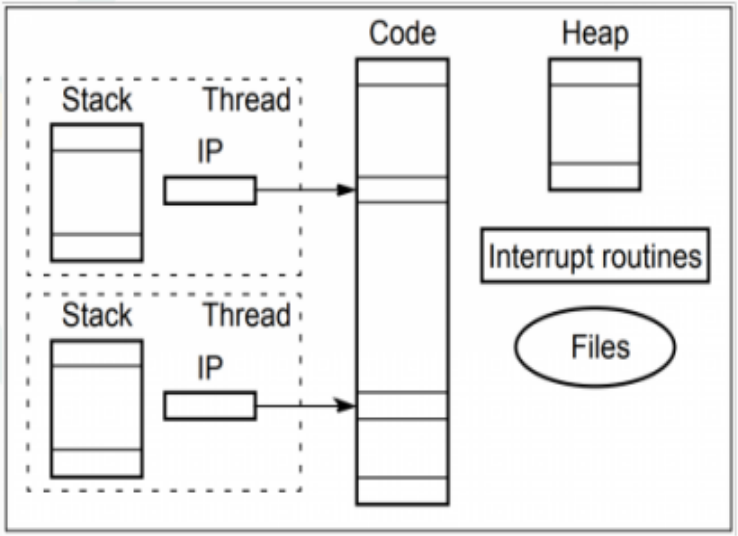
c. Thread-per-object

Processos x Threads

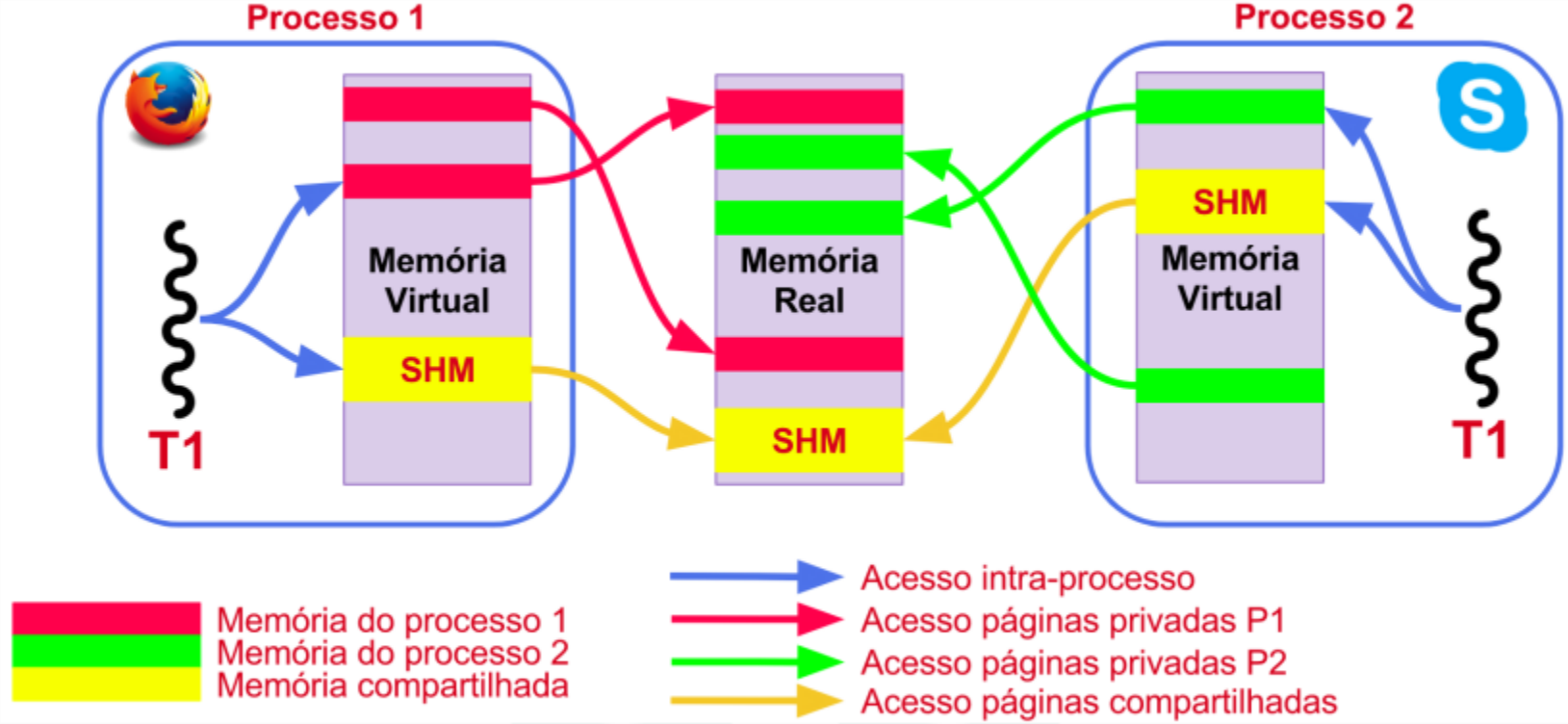
- Processos
  - Separação total dos programas: pilhas, espaço de memória, descritores de arquivos, tabelas de páginas



- Threads
  - Compartilham o espaço de memória, descritores de arquivos, tabelas de páginas
  - Uma pilha e IP por thread



Processos x Threads



Processos x Threads

- SO provê mecanismos para dividir o tempo do processador entre processos e threads, escalonando-os nas unidades de processamento disponíveis
  - Bloqueios por alguma operação de E/S causam uma troca do processo/thread pelo próximo na fila
- **Trocas de contexto entre threads são baratas**
  - Basta trocar o IP e mais alguns registradores
- **Trocas de contexto entre processos são mais caras**
  - Exigem troca da tabela de páginas, troca de IP, troca de rotinas de manuseio de interrupções
- Ainda assim há momentos que o uso de processos pode ser preferível. Exemplo: Google Chrome

Troca de contexto

- **Contexto do processador:**
  - um conjunto mínimo de valores guardados nos registradores do processador, usado para a execução de uma série de instruções (ex: ponteiro de pilha, registradores, contador de programa, etc.)
- **Contexto de thread**
  - um conjunto mínimo de valores guardado em registradores e memória, usado para a execução de uma série de instruções (i.e., contexto do processador e estado)
- **Contexto de processo**
  - um conjunto mínimo de valores guardados em registradores e memória, usados para a execução de uma thread (i.e., contexto de threads e os valores dos registradores de MMU — Memory Management Unit)

Troca de contexto

- 1. Threads compartilham o mesmo espaço de endereçamento. A realização da troca de contexto pode ser feita independentemente do sistema operacional
- 2. A troca de processos é mais custosa, já que envolve o sistema operacional
- 3. Criar e destruir threads é muito mais barato do que fazer isso com processos

Threads

- Não existe proteção entre threads
- Threads compartilham espaço de memória, temporizadores, processos filhos, sinais, etc.
- Estruturas por threads são contador de programa, pilha, valores dos registradores, threads filhos e estado do thread

Usos de Threads

- Criados para permitir combinar paralelismo com execução sequencial e chamadas de sistema bloqueadas
- Organizações de threads podem usar os modelos
  - Entregador/trabalhadores – entregador escolhe thread livre e o entrega a requisição
  - Time – threads são iguais e recebem e processam suas próprias requisições
  - Pipeline – threads executam parte da requisição e passam resultados parciais para o próximo thread do pipeline

Projeto de Threads

- Pacote de threads
- Gerenciamento de threads
  - Threads estáticos – número de threads e tamanho da pilha são decididos em tempo de programação ou compilação. Simples mas inflexível
  - Threads dinâmicos – threads são criados e destruídos durante execução. Cada processo é iniciado com um thread (implícito) e pode criar mais se necessário
- Threads compartilham dados na mémoria, e acesso é controlado usando regiões críticas
- Procedimentos não-reentrantes tem de ser reescritos

Threads e Sistemas Operacionais

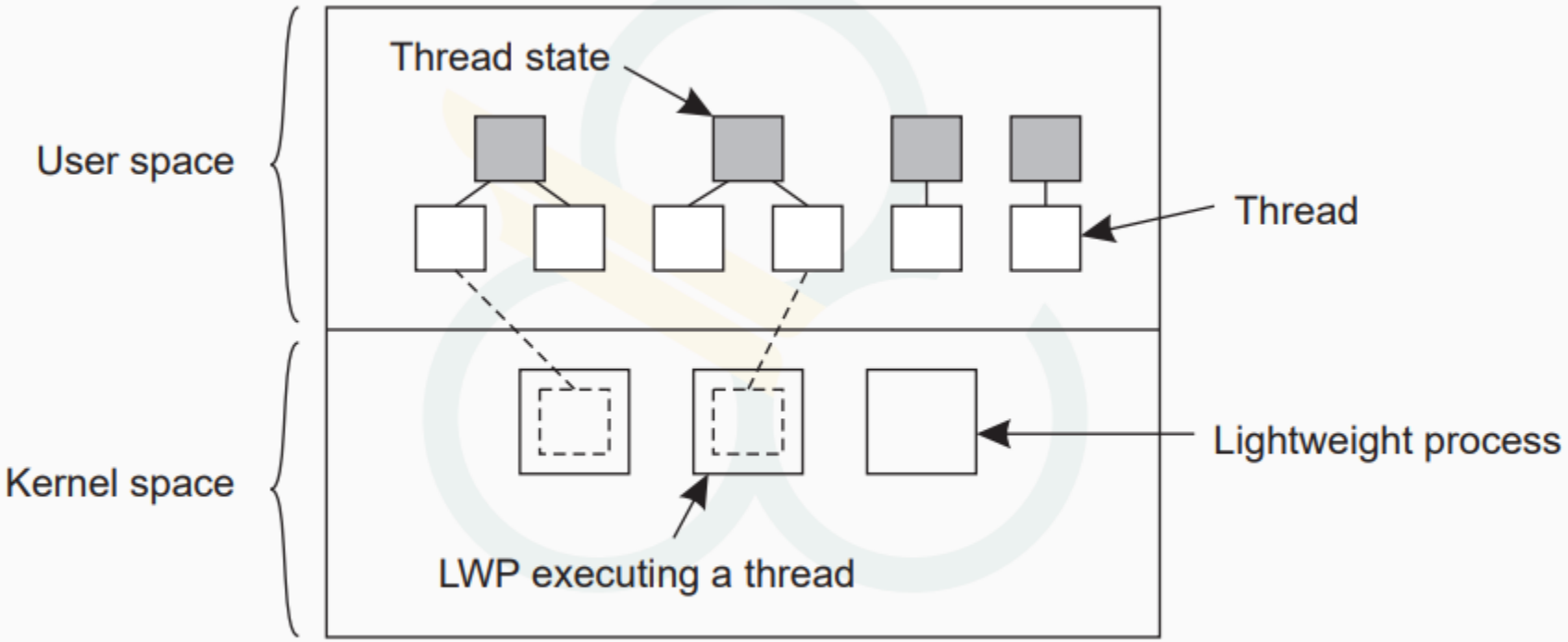
- O núcleo do sistema operacional deve prover threads, ou elas devem ser implementadas em nível de usuário?
- Solução no nível de usuário
  - Todas as operações podem ser realizadas dentro de um único processo — muito mais eficiente
  - Todos os serviços providos pelo núcleo são feitos em nome do processo na qual a thread reside — se o núcleo decidir bloquear a thread, o processo inteiro será bloqueado
  - Threads são usadas quando há muitos eventos externos: threads são bloqueadas com base nos eventos recebidos — se o kernel não puder distinguir as threads, como permitir a emissão de sinais do SO para elas?

Threads e Sistemas Operacionais

- **Solução no nível de Sistemas Operacionais**
- O núcleo contém a implementação do software de threading. Todas as operações são chamadas de sistemas
- Operações que bloqueiam uma thread não são mais um problema: o núcleo escalona outra thread ociosa dentro do mesmo processo
- Tratamento de eventos externos mais simples: o núcleo (que recebe todos os eventos) escalona a thread associada com aquele evento
- O problema é a perda de eficiência devido ao fato de que todas as operações em threads requerem um trap pro núcleo
- **Conclusão O melhor é tentar juntar threads de nível de usuário e de nível do SO em um único conceito**

Threads do Solaris

- Introduz uma abordagem em dois níveis para threads: processos leves que podem ser executar threads de nível de usuário



Implementação de um Pacote de Threads

- Processos podem ter seus próprios algoritmos de escalonamento
- Tem melhor escalabilidade, já que espaço no kernel é limitado

- Desvantagens
  - Implementação de chamadas bloqueadas (usar chamadas não-bloqueadas, reescrever chamadas do sistema como READ – jacket)
  - Tratamento de page-faults
  - Escalonamento de threads (como um thread passa controle para outro?)
  - Programadores querem usar threads em aplicações onde chamadas do sistema são bloqueadas com frequência (se um thread é bloqueado, pode-se transferir o controle para outro thread com overhead mínimo)

Implementação de um Pacote de Threads

- Threads no kernel. Kernel tem uma tabela por processo com uma entrada por thread
- Todas as chamadas que podem bloquear um thread são implementadas como chamdas de sistema
- Vantagens
  - Tratamento simples de page-faults e implementação simples de chamadas bloqueadas
- Desvantagens
  - Operações com threads tem um overhead considerável

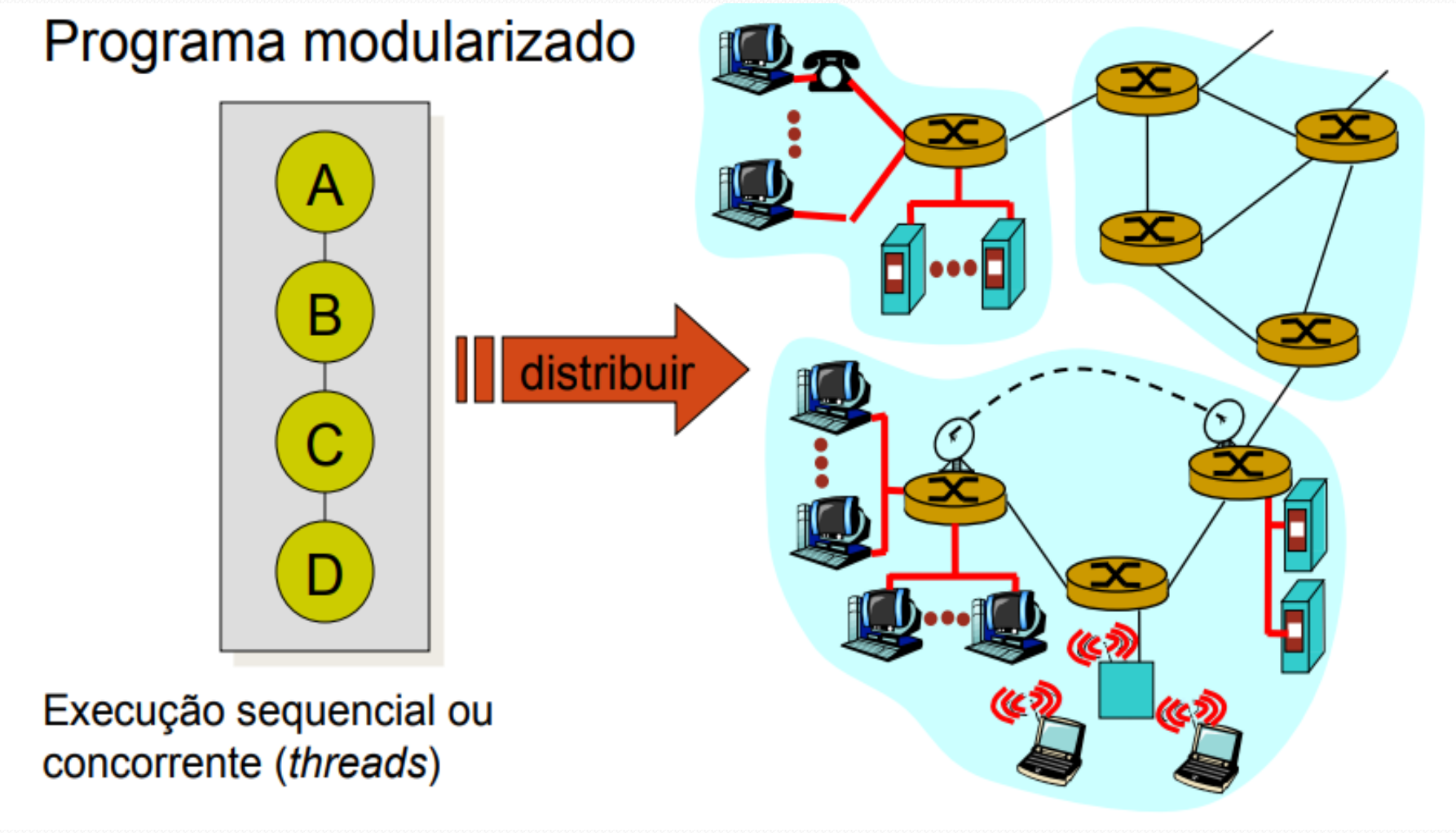
Threads e Sistemas Distribuídos

- **Clientes web multithread— escondem a latência da rede**
  - Navegador analisa a página HTML sendo recebida e descobre que muitos outros arquivos devem ser baixados. •
  - Cada arquivo é baixado por uma thread separada; cada uma realiza uma requisição HTTP (bloqueante)
  - A medida que os arquivos chegam, o navegador os exibem.
- **Múltiplas chamadas requisição–resposta (RPC) para outras máquinas**
  - Um cliente faz várias chamadas simultâneas, cada uma em uma thread diferente •
  - Ele espera até que todos os resultados tenham chegado. •
  - Obs: se as chamadas são a servidores diferentes, você pode ter um speed-up linear

Threads e Sistemas Distribuídos

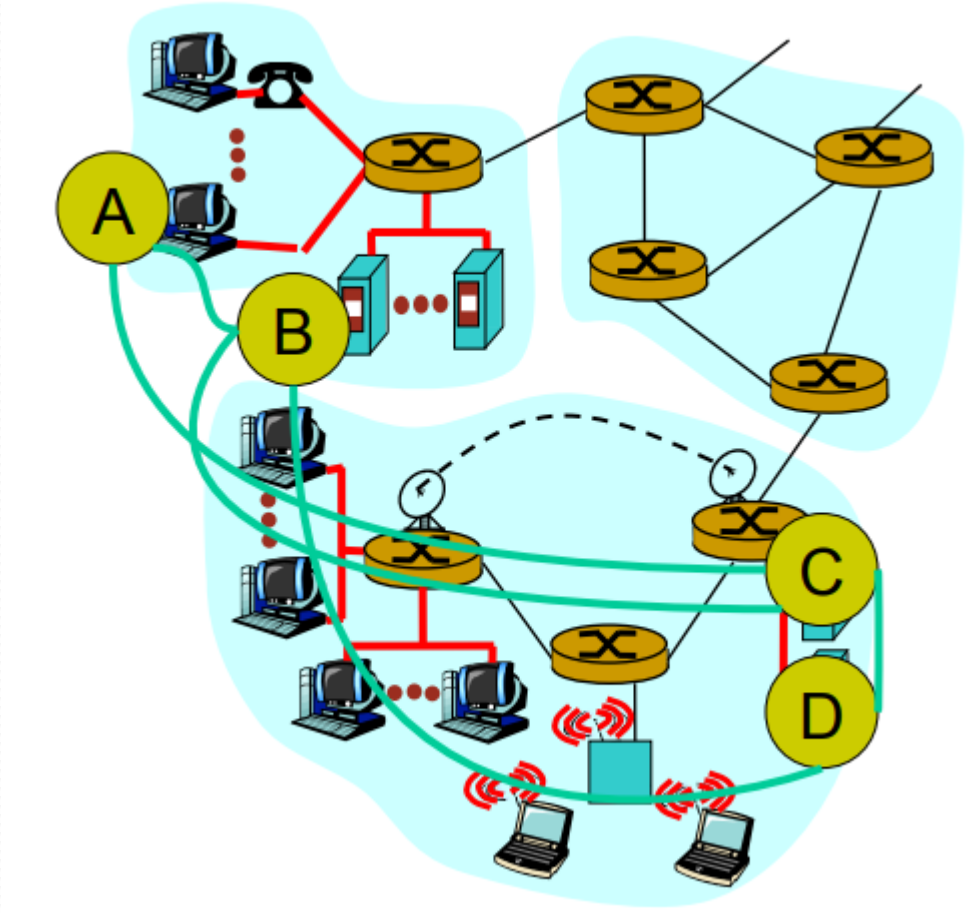
- **Melhorias no desempenho**
  - Iniciar uma thread é muito mais barato do que iniciar um novo processo
  - Ter servidores single-threaded impedem o uso de sistemas multiprocessados
  - Tal como os clientes: esconda a latência da rede reagindo à próxima requisição enquanto a anterior está enviando sua resposta.
- **Melhorias na estrutura**
  - A maioria dos servidores faz muita E/S. Usar chamadas bloqueantes simples e bem conhecidas simplifica o programa.
  - Programas multithreaded tendem a ser menores e mais fáceis de entender, já que simplificam o fluxo de controle.

Threads e Sistemas Distribuídos



Threads e Sistemas Distribuídos

- **Programa distribuído**
- Componentes interligados (comunicação)
- **Processamento**
- (computação) distribuído ou paralelo



Custo benefício

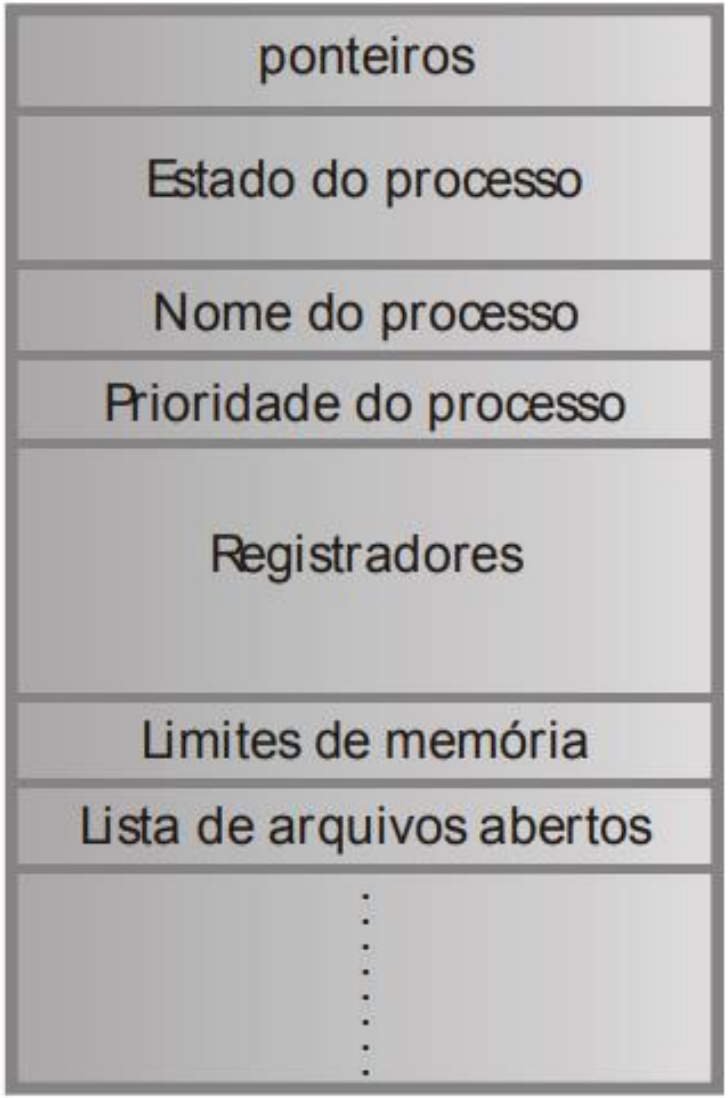
- Potencial para ser mais poderoso do que um sistema centralizado, convencional
- Pode ser mais confiável: toda função pode ser replicada
- Quando um processador falha, outro pode continuar o trabalho
- Crash não necessariamente resulta em perda de dados
- Várias computações podem ser realizadas em paralelo
- **Pode-se considerar tolerância a falha e possibilidade de paralelismos como as propriedades fundamentais de um sistema distribuído**

Modelos de Sistemas

- Em um SD, surge a questão de como se utilizar os diferentes processadores, como alocar processor e/ou threads
- Existem dois modelos básicos: o de estações de trabalho, e o de pool de processadores
- Além desse dois modelos, pode-se implementar um sistema híbrido, com algumas características de ambos

Contexto do processo

- O SO materializa o processo através do bloco de controle de processo – PCB
- O PCB de todos os processos ativos residem na memória principal em uma área exclusiva do SO



Estações de Trabalho

- Sistema é composto de estações de trabalho, que podem ou não ter discos locais
- Porque não ter discos locais? (mais raro atualmente)
- Discos locais podem conter: paginamento e arquivos temporários, executáveis do sistema, cache de arquivos, ou até sistema de arquivos completo

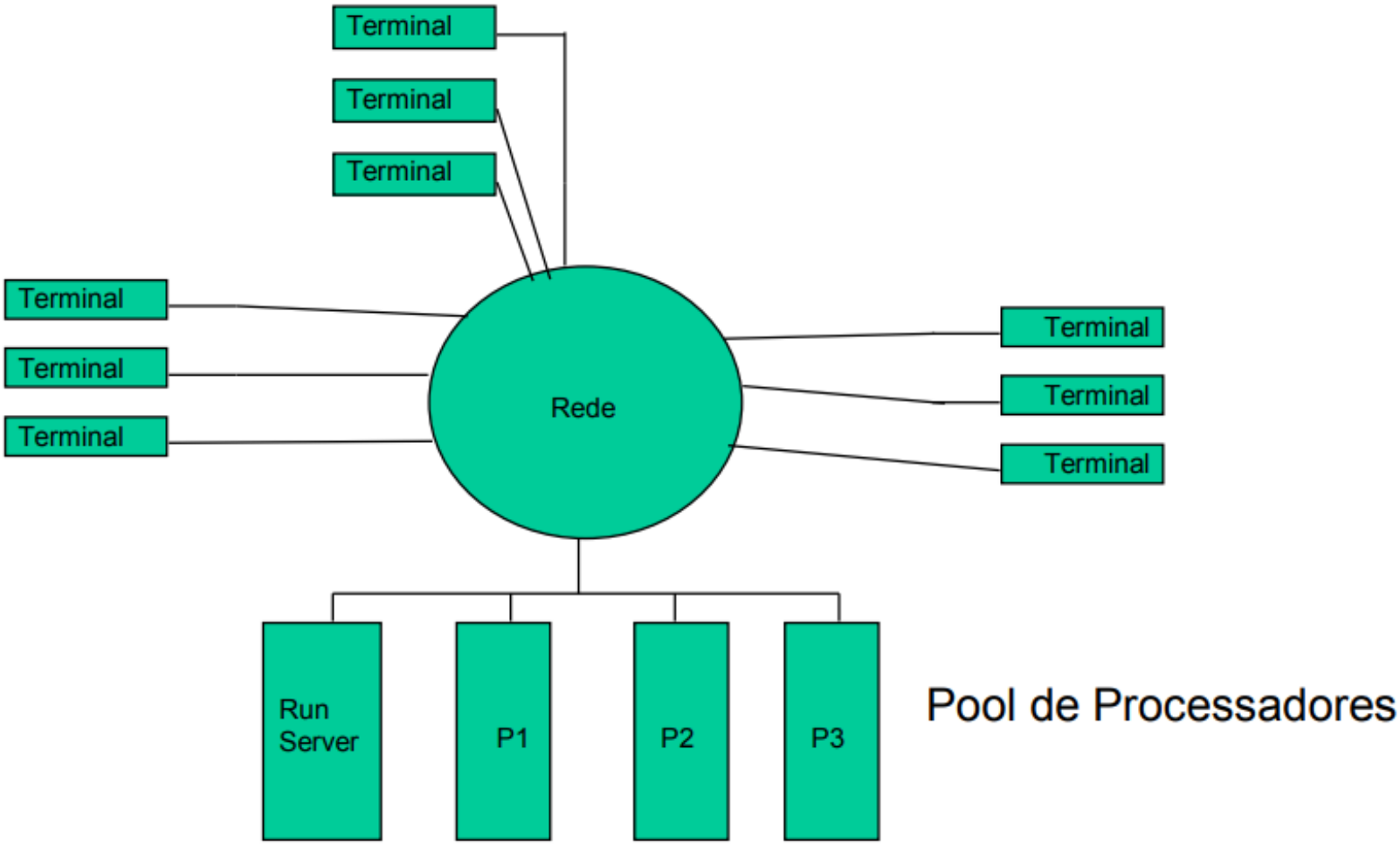
Estações de Trabalho

- Como uma estação de trabalho “desocupada” é achada?
  - Qual a definição de desocupada?
- Como um processo remoto pode ser executado de forma transparente?
  - Visão do sistema de arquivos e variáveis de ambienteda máquina remota deve ser a mesma do sistema local
  - O que deve e o que não deve ser retornado a máquina local
- O que acontece quando o “dono” da máquina volta a usá-la?
  - Matar processo, dar uma sobrevida ao processo ou transferi-lo

Pool de Processadores

- Contrução de um rack cheio de processadores que são alocados sob demanda
- Usados com servidores de arquivos e estações de trabalho sem sistemas de arquivo locais
- Boa escalabilidade
- Processadores não tem dono, como no caso anterior
- Pode-se modelar analiticamente comportamento do sistema

Pool de Processadores



Modelo Híbrido

- Modelo híbrido é mais caro mas combina vantagens dos dois modelos
- Usuários podem utilizar suas estações para trabalhos comuns e pool de processadores para trabalhos que exijam alto poder computacional
- Neste modelo híbrido estações de trabalho desocupadas não são utilizadas

Modelos de Alocação

- Geralmente, artigos assumem que:
  - Computadores são compatíveis em código, podendo diferir no máximo em velocidade
  - Sistema é totalmente interconectado
- Processos podem ser não-migratórios ou migratórios
- O que otimizar na alocação?
  - Utilização das CPUs
  - Tempo de resposta médio

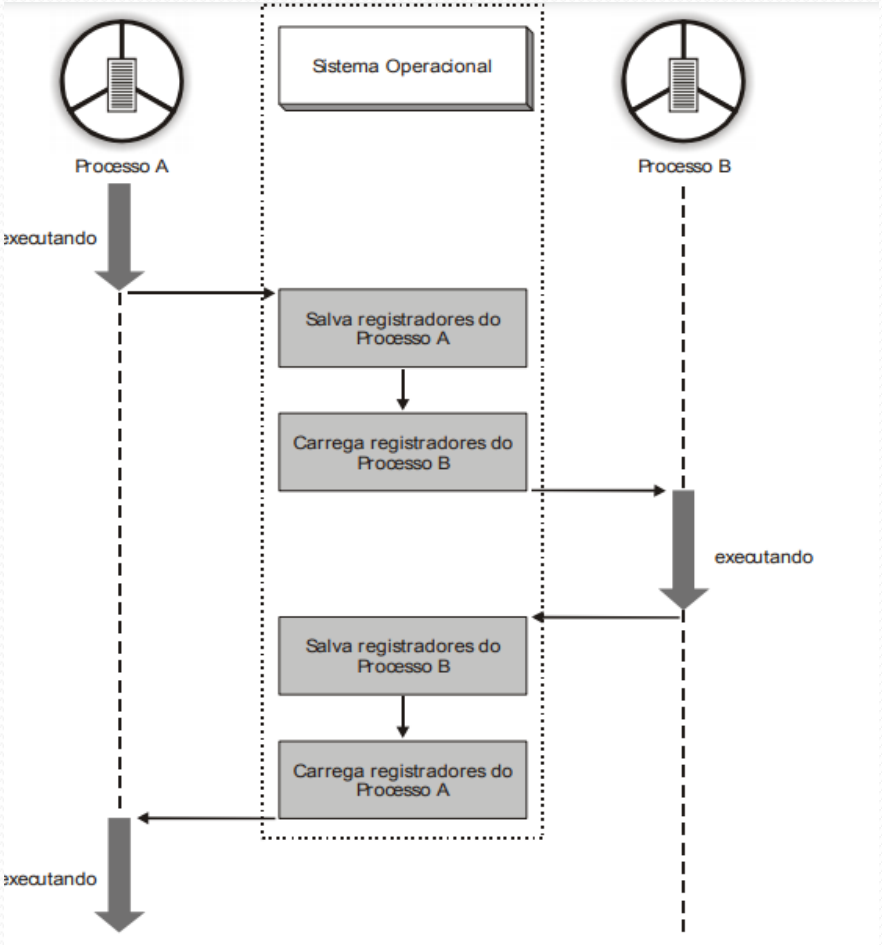
Alocação de Processadores

- Questões de projeto. Usar algoritmos:
  - Determinísticos ou heurísticos
  - Centralizados ou distribuídos
  - Ótimos ou sub-ótimos
  - Locais ou globais
  - Iniciados pelo emissor ou pelo receptor
- Algoritmos determinísticos são apropriados quando se sabe sobre todas as necessidades dos processos antecipadamente

Alocação de Processadores

- Solução ótima pode ser muito mais cara computacionalmente que solução sub-ótima
- Na prática, SDs implemetam algoritmos heurísticos distribuídos sub-ótimos
- Política de transferência determina se processo será executado localmente ou não baseado em informações locais ou globais
- Política de localização indica como máquina ociosa receberá processo

Alocação de Processadores



Implementação

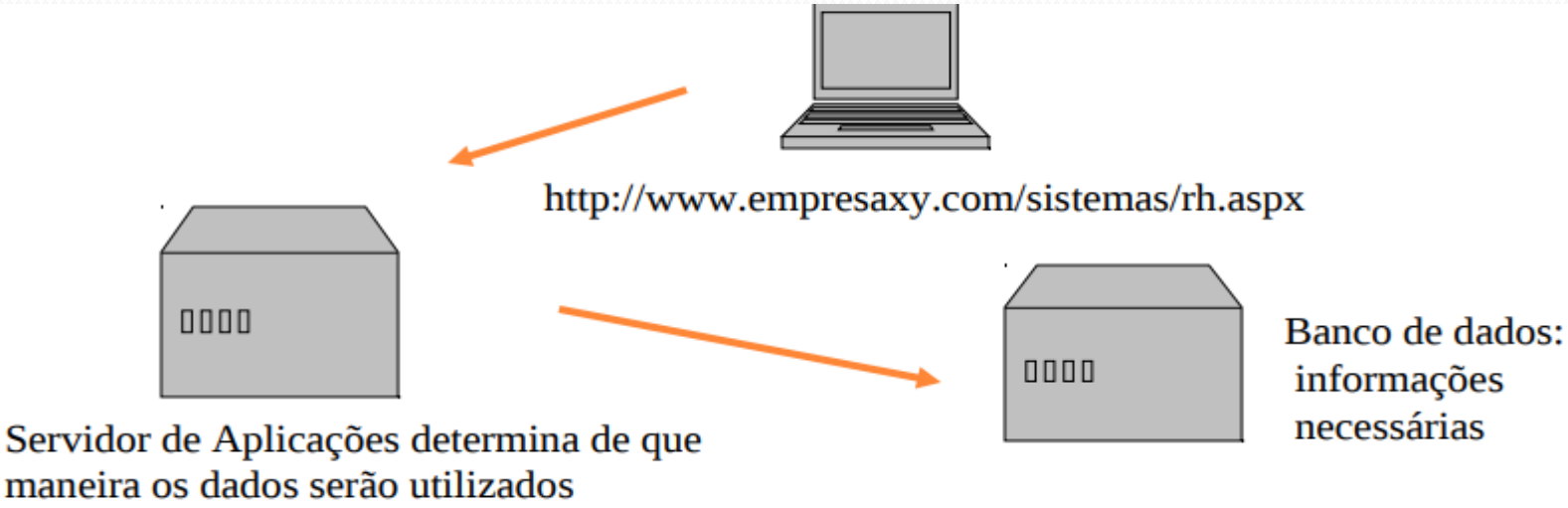
- Algoritmos assumem que máquinas sabem sua própria carga
- Como tratar overhead na transferência de processos? Pode ser mais vantajoso manter um processo na máquina local do que em uma máquina remota mais rápida
- Estabilidade das informações usadas pelos algoritmos podem afetar suas performances

Exemplos Algoritmos

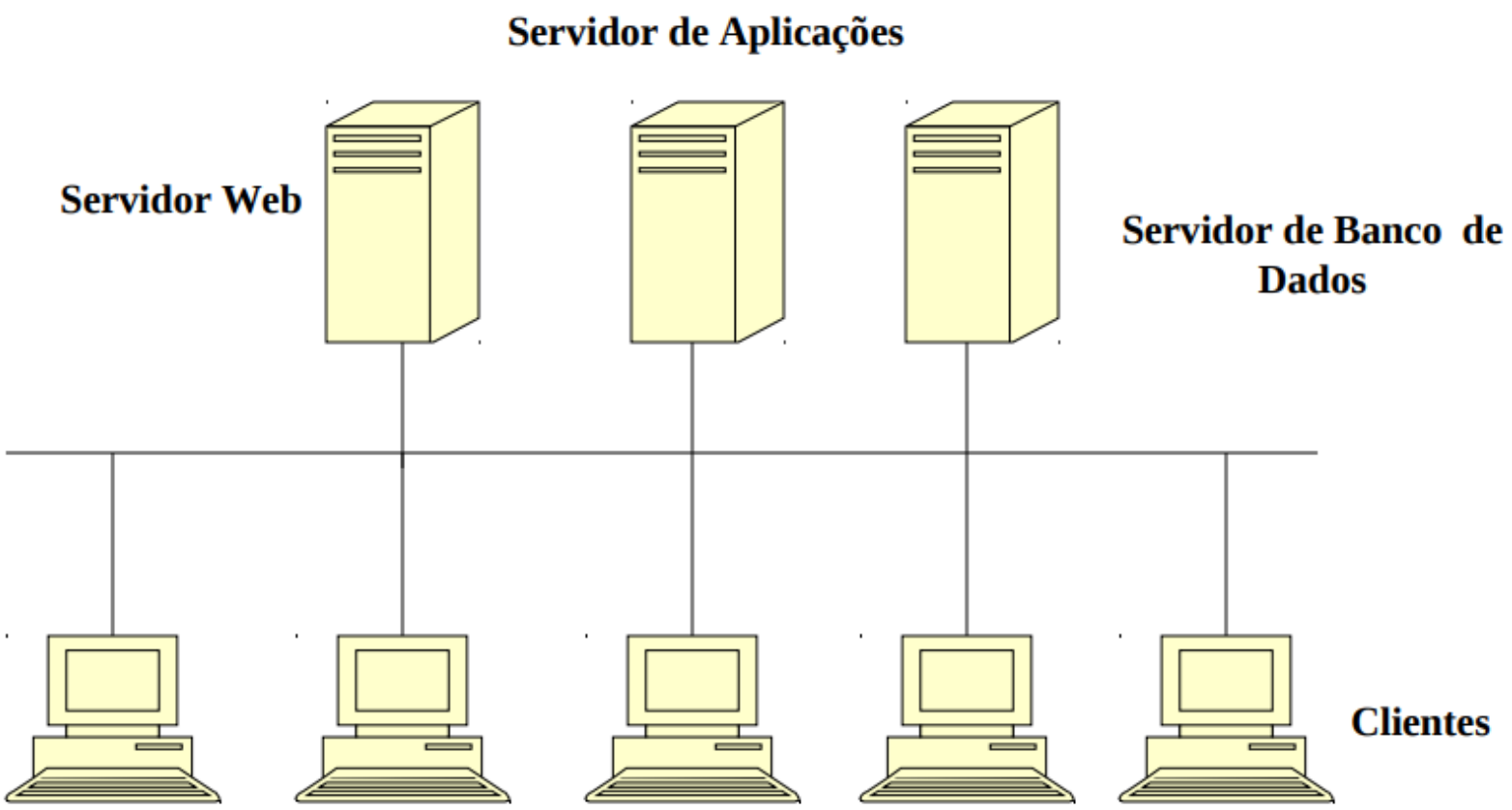
- Algoritmo determinístico com grafos
- Algoritmo up-down (centralizado)
- Algoritmo hierárquico
- Algoritmo heurístico iniciado pelo emissor
- Algoritmo heurístico iniciado pelo receptor
- Algoritmo de lances

Arquitetura em Sistemas Distribuídos

- Cliente quer acessar a aplicação
- Acessa o endereço da mesma através do navegador
- Servidor de aplicações verifica se ao cliente é permitido o acesso
- Banco de dados liberado.



Aplicações em quatro camadas



Aplicações em quatro camadas

- Cliente: Navegador;
- Apresentação: Servidor Web, onde serão feitas as alterações de interface;
- Lógica (Regras do Negócio): Servidor de Aplicações, onde serão feitas as alterações nas regras do negócio, quando necessárias;
- Dados: Servidor de Banco de Dados, com todas as informações necessárias.

Aplicações em quatro camadas

- Vantagens
  - Serviços
  - Complexidade;
  - Recursos;
  - Comunicação.
  - Protocolos;
  - Localização;
  - Descentralização;
  - Escalabilidade;
  - Integridade.
- Desvantagens
  - Complexidade
  - Comunicação

Escalonamento

- Escalonamento é feito por cada processador, mas, para se maximizar paralelismo do sistema, processos que se comuniquem com frequência deveriam rodar em paralelo
- Geralmente é difícil de identificar com antecedência tais processos, mas processos de um grupo são bons candidatos a isto
- O conceito de co-escalonamento usa padrões de comunicação interprocesso no escalonamento
- Sincronização dos processadores é necessária para que slots de escalonamento sejam executados no mesmo período

Tolerância a Falhas

- Diferentes SDs têm diferentes necessidades de tolerância a falhas
- Falhas de componentes são classificadas em:
  - Transientes – Acontecem uma vez e desaparecem
  - Intermitentes – Falha recorrente
  - Permanentes – Definitiva
- Tipos de falhas são:
  - Falha silenciosa – Processador (ou processo) para e não responde mais
  - Falha Bizantina – Processador (ou processo) continua executando mas gerando informações incorretas, maliciosamente ou não

Tolerância a Falhas

- Nesta sub-área um sistema é chamado de síncrono se existe um limite de tempo para recebimento de uma resposta a uma mensagem, caso contrário é um sistema assíncrono
- Os tipos de redundância são:
  - De informação – Exemplo: Código de Hamming
  - De tempo – Repetição de um processo (transações atômicas)
  - Física – Equipamento extra é adicionado

Redundância

- Redundância física pode ser organizada no esquema de replicação ativa ou primário-backup
- Replicação ativa (abordagem de máquina de estados) é muito usada em circuitos eletrônicos
- A redundância modular tripla (TMR) replica cada componente três vezes e adiciona componentes chamados voters
- Quanta replicação é necessária? Um sistema é dito *k*-tolerante a falhas se consegue suportar a quebra de *k* componentes

Redundância

- Uma pré-condição desse modelo é que todos os servidores devem receber as mensagens na mesma ordem (pelo menos as escritas que não comutem)
- No modelo primário-backup, o servidor primário executa todo o trabalho e se ele falha, o backup assume
- Troca de servidores deve ser notada somente pelos SOs das máquinas dos servidores
- Operação normal mais simples (só um servidor)

Redundância

- Requer menos recursos extra, mas quando o backup assume, um novo backup é necessário
- Como distinguir entre um servidor lento e um morto?
- Pode-se usar discos compartilhados ente os servidores, eliminando-se mensagens de atualização entre eles. Como tratar quebras deste disco compartilhado?

Bibliografia

- O conteúdo da aula foi elaborado utilizando os seguintes materiais de apoio:
- http://www.dimap.ufrn.br/~motta/dim070/Processos.pdf
- http://professor.ufabc.edu.br/~e.francesquini/sd/files/aula05.pdf
- http://www.ic.uff.br/~simone/sd/contaulas/aula5.pdf
- https://edisciplinas.usp.br/pluginfile.php/2945126/mod_resource/content/1/22-threads-v8.pdf
- https://www.inf.pucrs.br/~gustavo/disciplinas/ppd/material/slides-intro-novo.pdf
- https://www.cin.ufpe.br/~fjclf/if677/aulas/09_SistemasDistribuidos.pdf