Capítulo 1: Introdução

Ajay Kshemkalyani e Mukesh Singhel

Distributed Computing: Principles, Algorithms, and Systems

Cambridge University Press

Definições

- Processos Autônomos comunicando através de uma rede
- Algumas características
  - Não há relógio físico comum a todos
  - Não há memória compartilhada
  - Separação geográfica
  - Heterogeneidade

Modelo de Sistema Distribuído

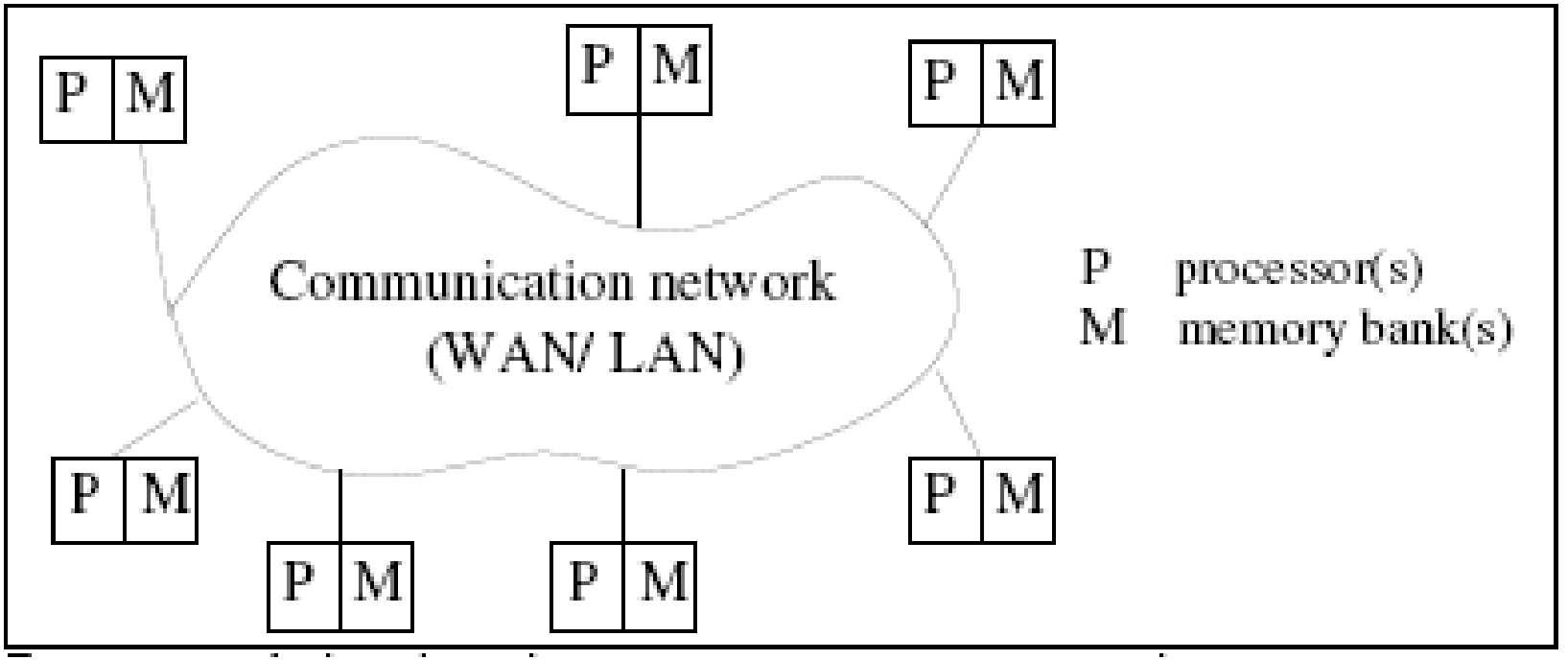

Figura 1.1: Sistema distribuído conectando processadores através de uma rede

Relação entre componentes de software

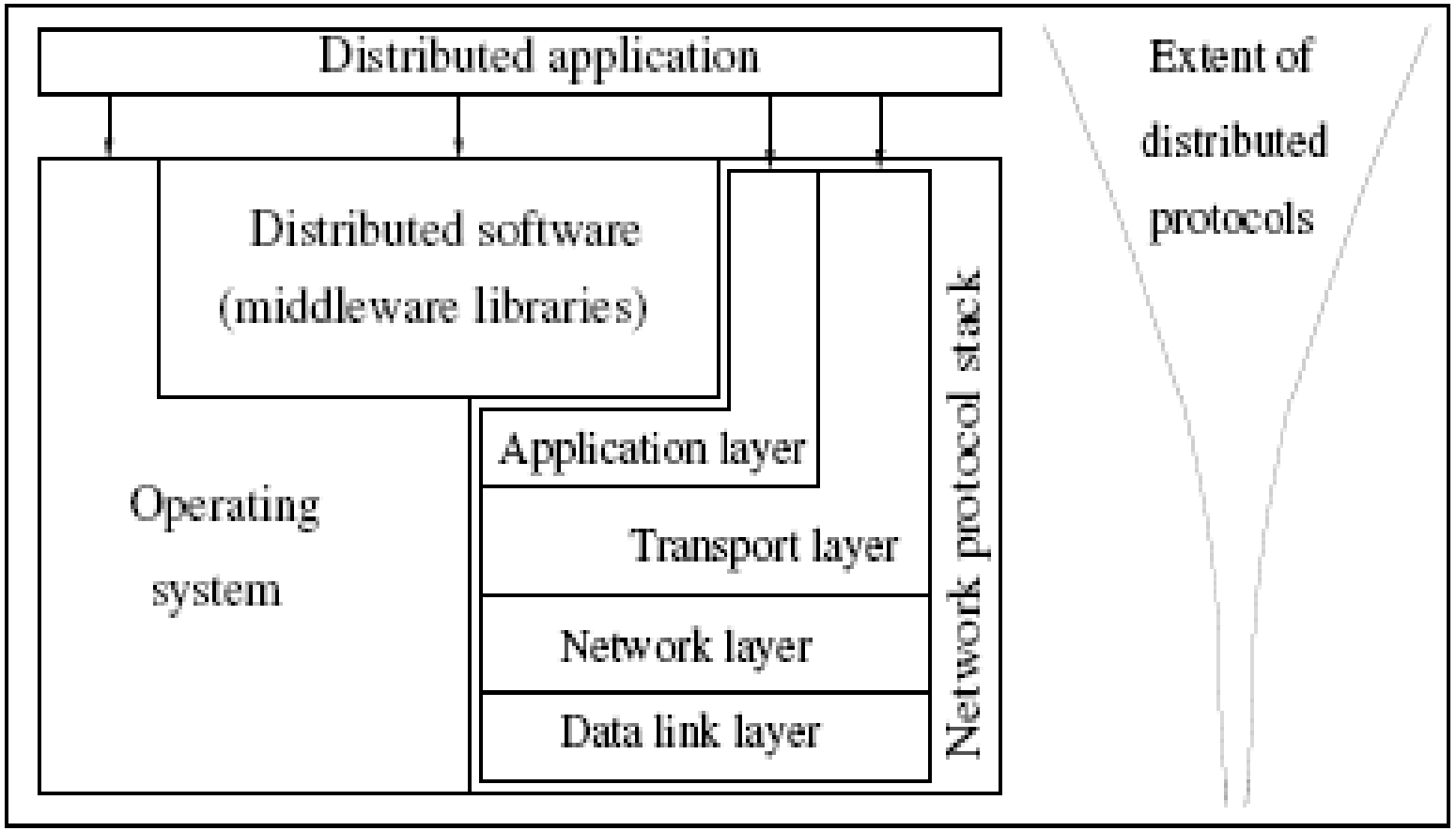

Figura 1.2: Interação dos componentes de software em cada processo

Motivação Para criação de sistemas Distribuídos

- Problemas inerentemente paralelos
- Compartilhamento de recursos
- Acesso a recursos remotos
- Aumento na relação custo / Performance
- Confiabilidade
  - Tolerância a falhas
  - Integridade
  - Disponibilidade
- Escalabilidade
- Modularidade e crescimento incremental

Sistemas Paralelos

- **Sistemas Multiprocessados:**
  - Acesso direto a memória, Modelo UMA
- **Sistemas de Multicomputadores:**
  - Ex. NYU Ultracomputer, IBM Blue Gene
  - Sem acesso direto a memória, Modelo NUMA
- **Array de processadores:**
  - Ex. Aplicações DSP, GPUs
  - Fortemente acoplados, clock comum

Modelo UMA vs. Modelo NUMA

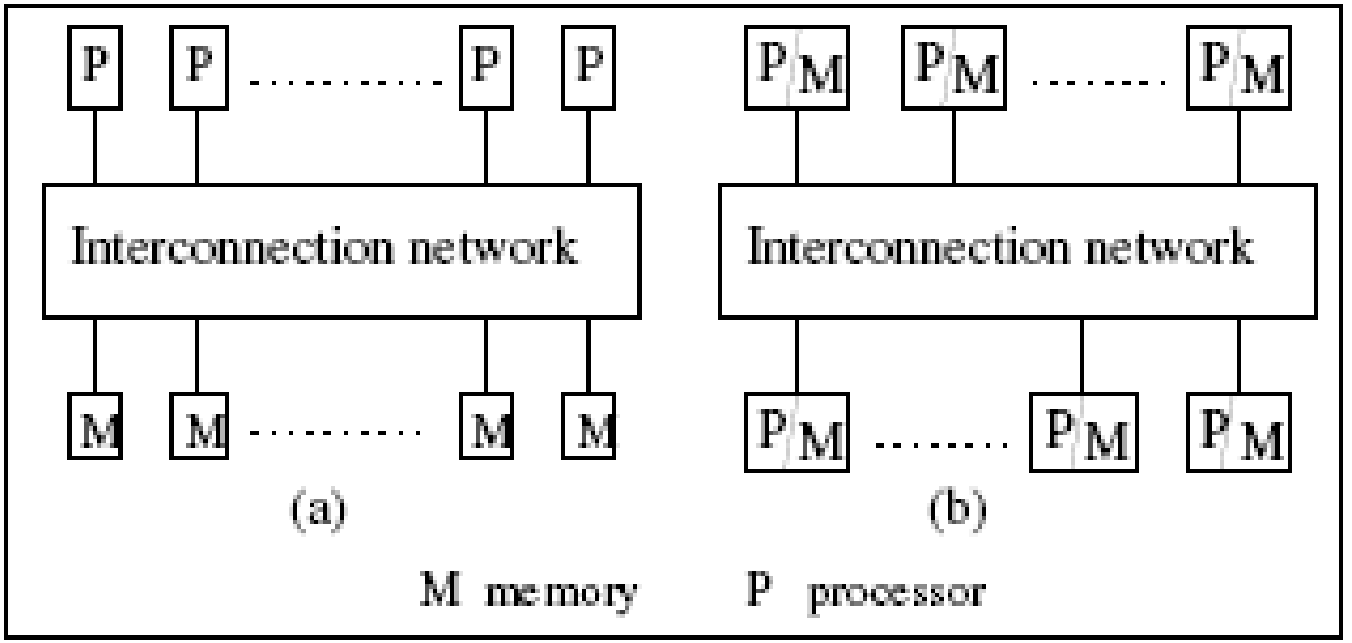

Figura 1.3: Duas arquiteturas paralelas padrão. (a) Unifor Memory Access (UMA). (b) Non-uniform memory access (NUMA). Em ambas as arquiteturas os processadores podem armazenar informação em cache local.

Topologia de interconexão para Multiprocessadores

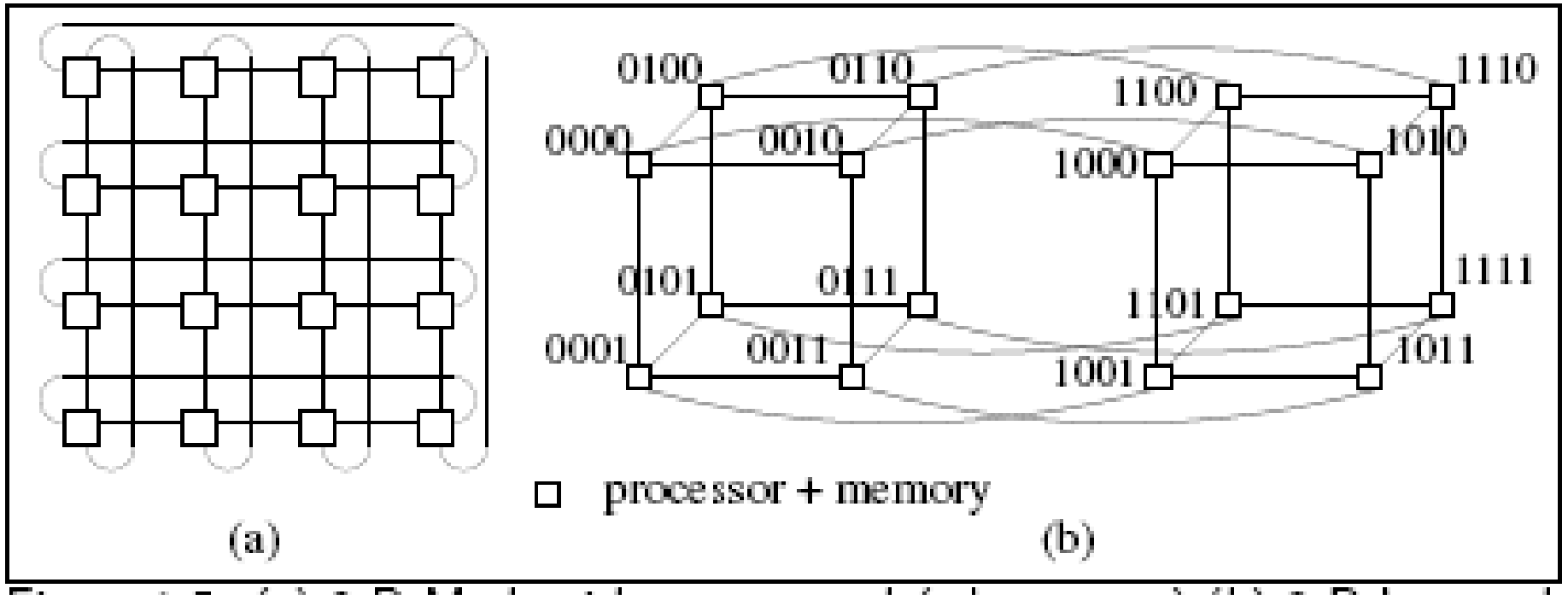

Figura 1.5: (a) Rede Mesh 2D com wraparaund (Ex. Torus). (b) Hipercubo 3D

Taxonomia de Flynn

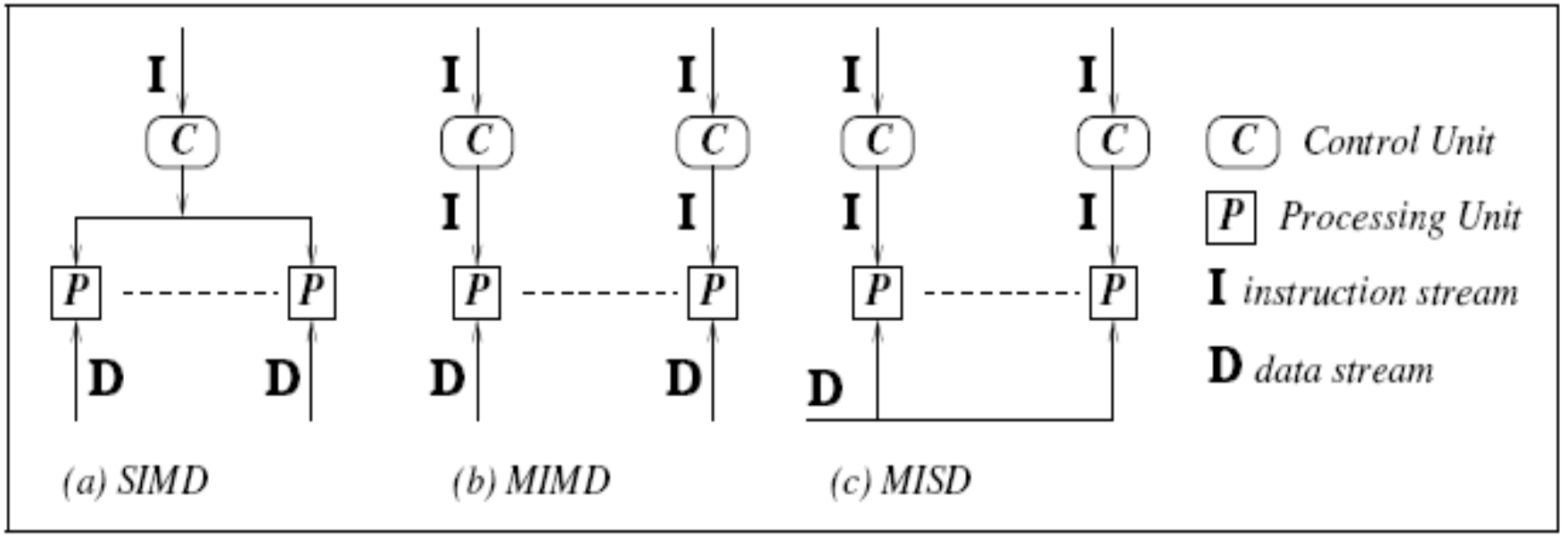

Figura 1.6: Arquiteturas SIMD, MISD e MIMD

Taxonomia de Flynn

- **SISD:** Single Instruction Single Data
  - Arquitetura tradicional
  - Fluxo único de instruções
- **SIMD:** Single Instruction Multiple Data
  - Aplicação em grandes vetores, calculo matricial
  - Maquinas vetoriais
- **MISD**: Multiple Instruction Single Data
  - Visualização
- **MIMD:** Multiple instruction Multiple Data
  - Grande maioria dos sistemas distribuídos

Terminologia

- **Acoplamento**
  - Interdependência entre os módulos, seja hardware ou software (Ex. OS, middleware)
- **Paralelismo: T(1) / T(n)**
  - Função do sistema
- **Concorrência:**
  - Medida entre o tempo produtivo da CPU e tempo perdido em operações de sincronização
- **Granularidade:**
  - Quantidade de computação vs. Quantidade de comunicação.

Troca de mensagens vs. Memória compartilhada

- **Simular troca de mensagens em memória compartilhada:**
  - Há uma área de memória compartilhada entre cada dois processos
  - *Send/Receive* simulado através de escrita/leitura na área de memória compartilhada entre o par de processos
- **Simular memória compartilhada com troca de mensagens:**
  - Cada objeto compartilhado é simulado através de um processo
  - Escrita no objeto simulada através de uma mensagem de escrita para o processo
  - Leitura do objeto simulada através de mensagem de *query*

Classificação de Primitivas

- **(Send / Receive) Síncrono**
  - Handshake entre remetente e destinatário
  - Remetente somente completa a operação quando o destinatário termina a operação
  - Destinatário termina operação quando o dado é totalmente copiado em seu buffer de entrada
- **(Send) Assíncrono**
  - Processo recobra o controle quando o dado é copiado em seu buffer de saída. Não há garantia de que o dado tenha chegado ao destinatário.

Classificação de Primitivas

- **(Send / Receive) Bloqueante**
  - Processo recobra o controle quando a operação termina (seja síncrona ou assíncrona)
- **(Send) Não bloqueante**
  - Processo recobra o controle Imediatamente após o comando.
  - **Send:** Antes de o dado ser copiado no buffer.
  - **Receive:** mesmo sem qualquer dado ter chegado ao buffer de entrada.

Primitivas não bloqueantes

```
Send(X, destination, handle_k)                    //handle_k is a return parameter
...
...
Wait(handle_1, handle_2, ..., handle_k, ..., handle_m)    //Wait always blocks
```

Figura 1.7: Uma primitiva de *send* não bloqueante. Quando a função *Wait* retorna pelo menos um dos parâmetros (handle) está setado.

- **Parâmetros retornados. Handle gerado pelo sistema:**
  - Pode ser utilizado depois simplesmente para checar o status de uma operação (Ex. sucesso ou erro)
  - *Wait* pode realizar espera ocupada pelo *handle* requerido ou bloquear esperando por interrupção do sistema.

Exemplo de Primitivas

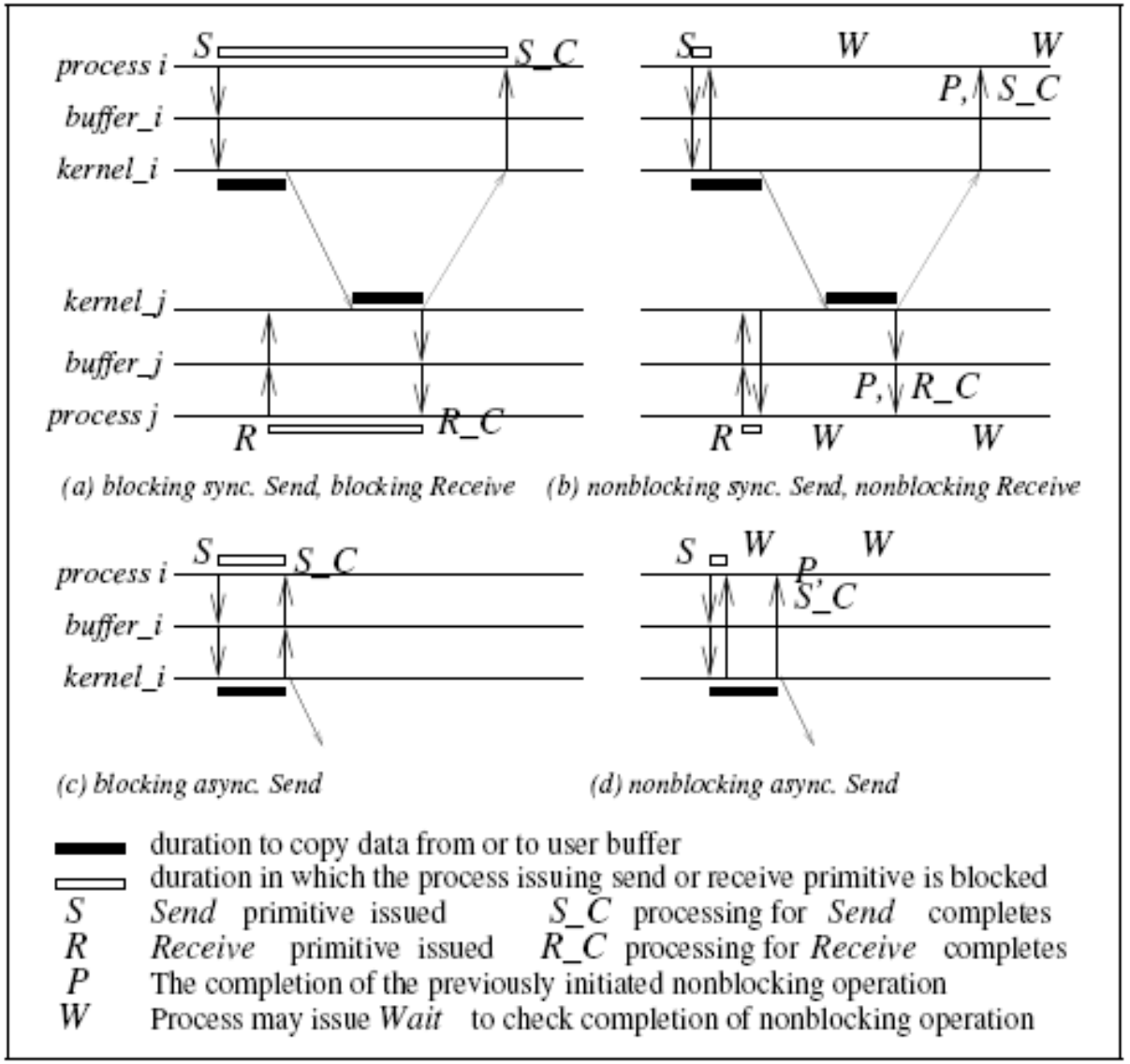

Figura 1.8: Exemplo de 4 primitivas de *send* e 2 de *receives*

Sistemas de troca de mensagens assíncrono

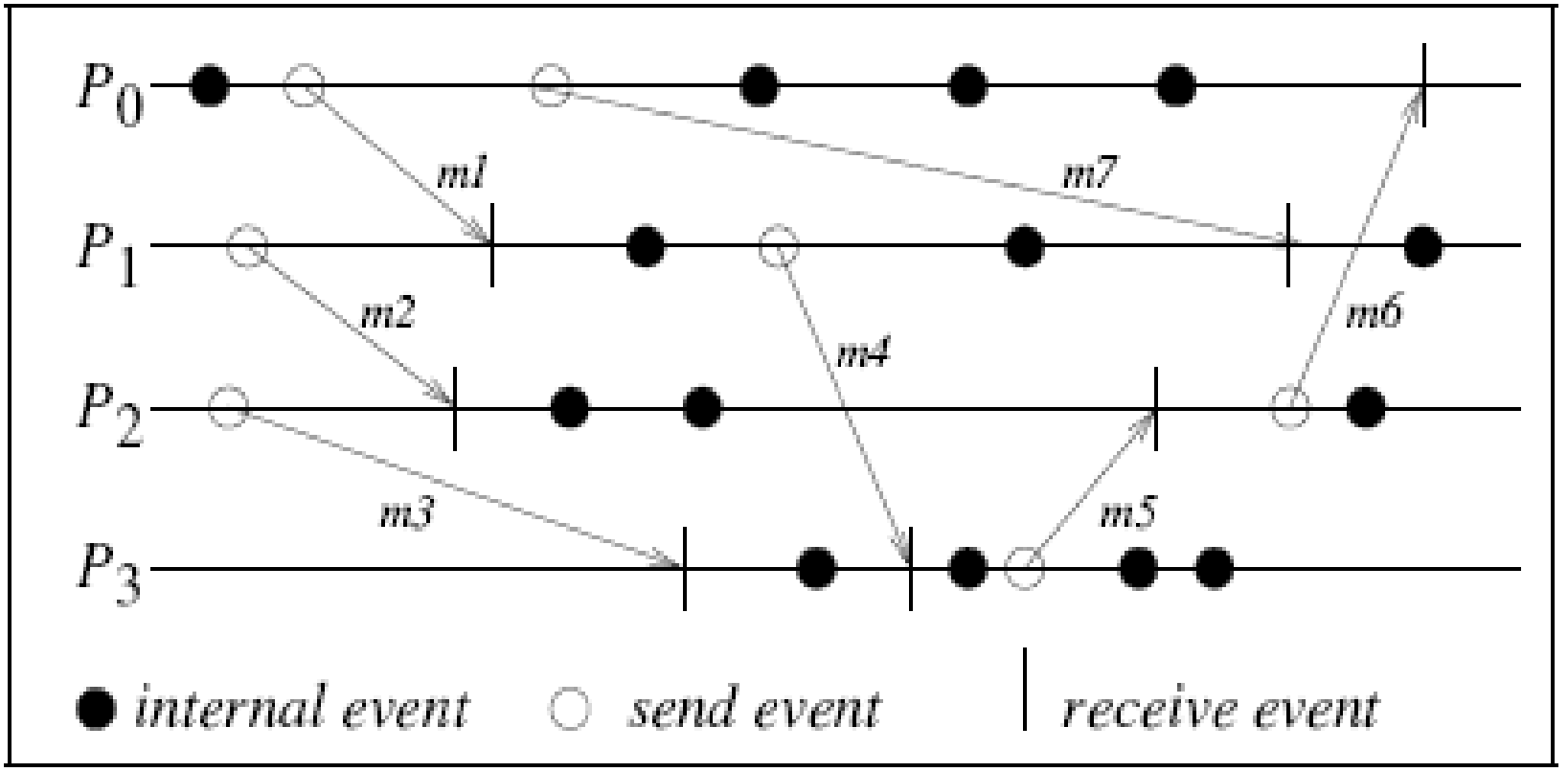

Figura 1.9: Execução Assíncrona em um sistema de troca de mensagens

Sistemas de troca de mensagens assíncrono

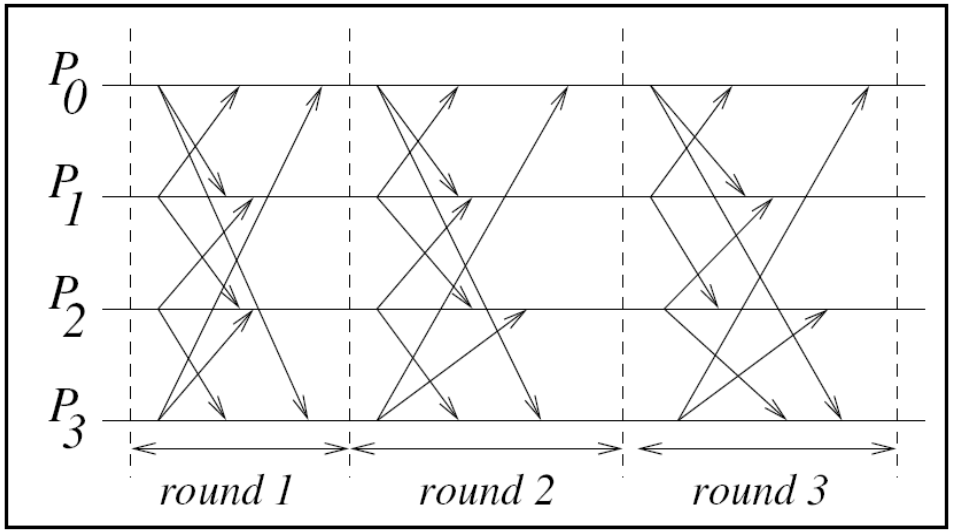

Figura 1.10 Execução síncrona de um sistema de troca de mensagens.

- (1) *Sync_Execution* (**int** k,n) //k rodadas, n processos
- (2) **for** r = 1 **to** k **do**
- (3) **proc** i envia msg para **(i + 1) % n** e **(i - 1) % n**;
- (4) cada **proc** recebe msg de **(i + 1) % n** e **(i - 1) % n**;
- (5) Executa função específica do problema

Execução Síncrona vs. Assíncrona

- **Execução assíncrona:**
  - Não há sincronismo entre processadores. *Clocks* não são sincronizados.
  - Mensagens tem atraso finito, mas indeterminado.
  - Não há limite de tempo para um processador realizar um passo.
- **Execução Síncrona:**
  - Processadores sincronizados. *Clocks* operam a mesma taxa.
  - Troca de mensagens ocorre em um único passo lógicos.
  - Limite superior de tempo para executar um passo em um processo indeterminado.

Execução Síncrona vs. Assíncrona

- **Dificuldade em construir sistemas realmente síncronos. Normalmente simulado em sistemas assíncronos.**
- **Sincronismo virtual:**
  - Execução assíncrona, processos sincronizam quando requerido.
  - Executa em passos.
- **Emulação:**
  - Processos assíncronos em sistemas síncronos. Trivial (Sincronismo é um caso especial de As sincronismo)
  - Processos Síncronos em sistemas assíncronos. Necessita-se de algoritmos de sincronização

Sistemas de troca de mensagens assíncrono

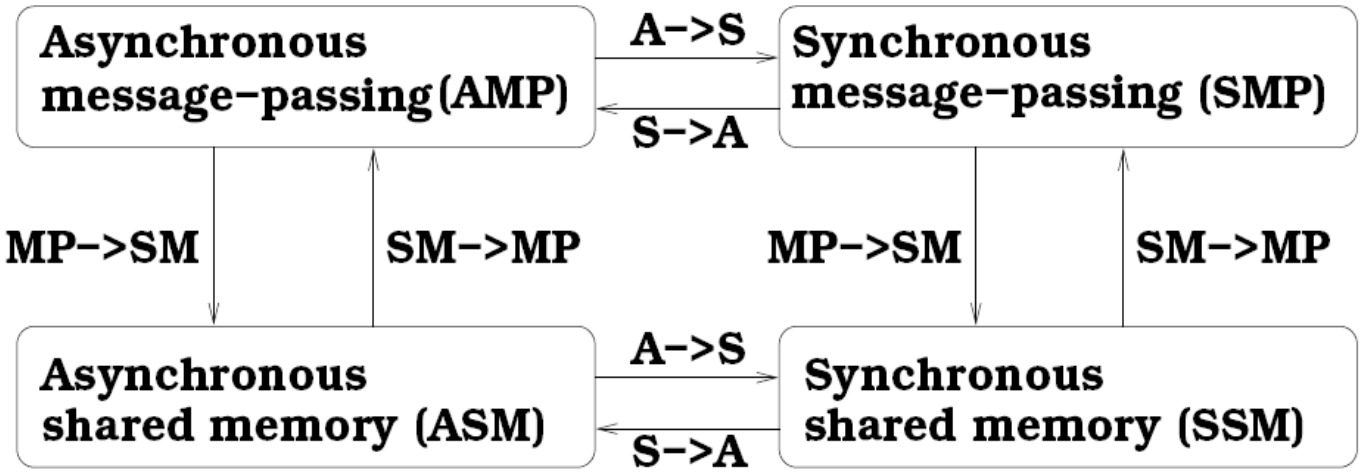

Figura 1.11: Simulações Síncronas < -> Assíncronas, Memória compartilhada < -> Troca de mensagens

- Assumindo-se Sistemas livres de erros
- Sistema A simulado no sistema B
  - Se não é solúvel em B não é solúvel em A
  - Se solúvel em A solúvel em B

Desafios: Sistema

- **Mecanismo de comunicação:** Ex. Remote Procedure Call (**RPC**), Remote Object Invocation (**ROI**), Orientado a mensagem vs. Stream de comunicação
- **Processadores:** Migração de código, Gerencia de processos / threads, design de software.
- **Naming**: Fácil identificação. Processos precisão localizar recursos de forma transparente e escapável.
- **Sincronização**
- **Consistência e replicação**
  - Replicação para acesso rápido evitando gargalos
  - Gerencia de consistência entre as diversas replicas
- **Armazenamento e acesso**
  - Armazenamento e acesso tem de ser rápidos e escaláveis através da rede
  - Reengenharia dos sistemas de armazenamento

Desafios: Sistema

- **Tolerância a falhas:** Operação correta e eficiente apesar de possíveis falhas de processos e rede de comunicação
- **Segurança em sistemas distribuídos:**
  - Canais seguros, acesso controlado. Gerencia de chaves (geração e distribuição), autorização, gerencia de grupos.
- **Escalabilidade e modularidade dos algoritmos, dados e serviços.**
- **Alguns sistemas experimentais: Globe, Globus, Grid.**

Desafios: Sistema

- **API para comunicação:** Fácil utilização
- **Transparência:** Esconder características de implementação do usuário.
  - **Acesso**: Esconder diferenças de replicação através do sistema.
  - **Localização**: Localizar os recursos deve ser feito de forma transparente.
  - **Migração**: Realocar recursos sem problemas explícitos de renomeação.
  - **Realocação**: Realocar recursos enquanto eles são utilizados
  - **Replicação**: Esconder replicação dos usuários
  - **Concorrência**: Resolver problemas de concorrência de forma implícita.
  - **Falha**: Tolerante a falhas e confiável.

Desafios: Design do Algoritmo

- **Framework e modelos úteis:** design correto de programas distribuídos.
  - Ordem Parcial
  - Autômato de Entrada e Saída
  - Ordem temporal lógica de eventos
- **Algoritmos dinâmicos distribuídos de grafos e roteamento:**
  - **Topologia do sistema:** grafos distribuídos com conhecimento apenas dos vizinhos.
  - **Algoritmos de Grafos:** Principio da comunicação em grupos, difusão de dados e localização de objetos.
  - **Algoritmos precisam lidar com mudança dinâmica de grafos.**
  - **Algoritmos eficientes:** Também impacta no consumo de recurso, latência, trafego e congestionamento

Desafios: Sistema

- **Tempo e Estado Global:**
  - Espaço 3D, Tempo unidimensional.
  - Precisão de relógios físicos
  - Tempo lógico define dependências inter-processos e mantêm a contagem relativa de tempo.
  - **Observação do estado global:** natureza distribuída do sistema.
  - **Medida de concorrência:** Concorrência depende da lógica do programa, velocidade de execução e comunicação das threads.

Desafios: Sistema

- **Sincronização e mecanismos de coordenação:**
  - **Sincronização de relógio físico:** Relógios de hardware necessitam de constante correção e ou relógio centralizado.
  - **Eleição de Líder:** Selecionar um processo com funções distintas dos demais.
  - **Exclusão Mútua:** Coordenar acesso a recursos compartilhados.
  - **DeadLock, detecção e resolução** : Necessita observação do estado global, Evitar detecção duplicada e abortos desnecessários.
  - **Detectar terminação:** Estado Global de estabilidade. Sem processamento de CPU e mensagens inter-processos em curso.
  - **Coletor de lixo:** Desalocar memória de objetos não mais apontados por nenhum processo.