

Sistemas Operacionais

Sincronização de Processos

Parte 3

Prof. Otávio Gomes

otavio.gomes@unifei.edu.br



Condições de corrida

e o problema da Seção Crítica

- Acesso concorrente a dados compartilhados pode resultar em **inconsistências**;
- **Condições de corrida** (*race conditions*):
 - Situação onde dois ou mais processos acessam e manipulam recursos compartilhados simultaneamente.
 - **Seção crítica:** N processos competem para usar alguma estrutura de dados compartilhada.
 - Cada processo possui uma seção crítica de código, onde há a **manipulação dos seus dados**.
 - Para resolver a questão de seção crítica cada processo deve pedir **permissão para entrar na região crítica**, após a utilização da seção crítica, seguir com a execução das ações.
 - Especialmente difícil resolver este problema em **kernel preemptivo**.

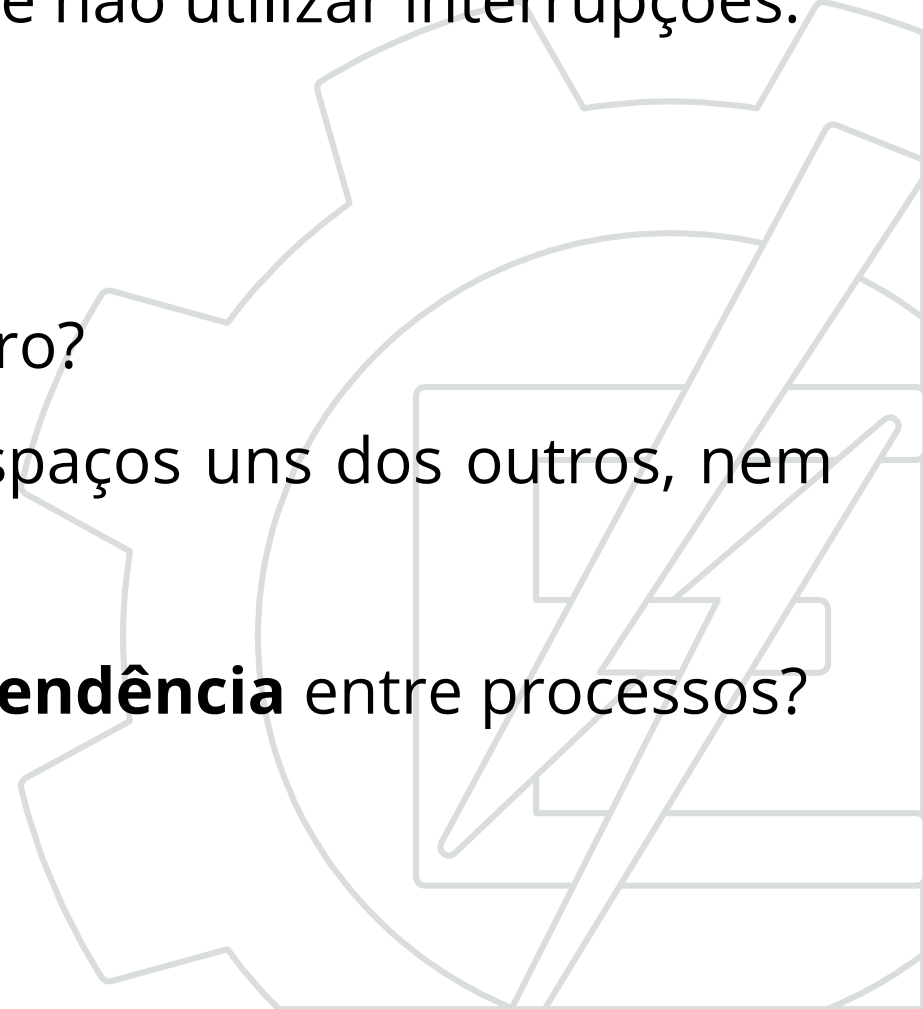
Seção crítica

Requisitos

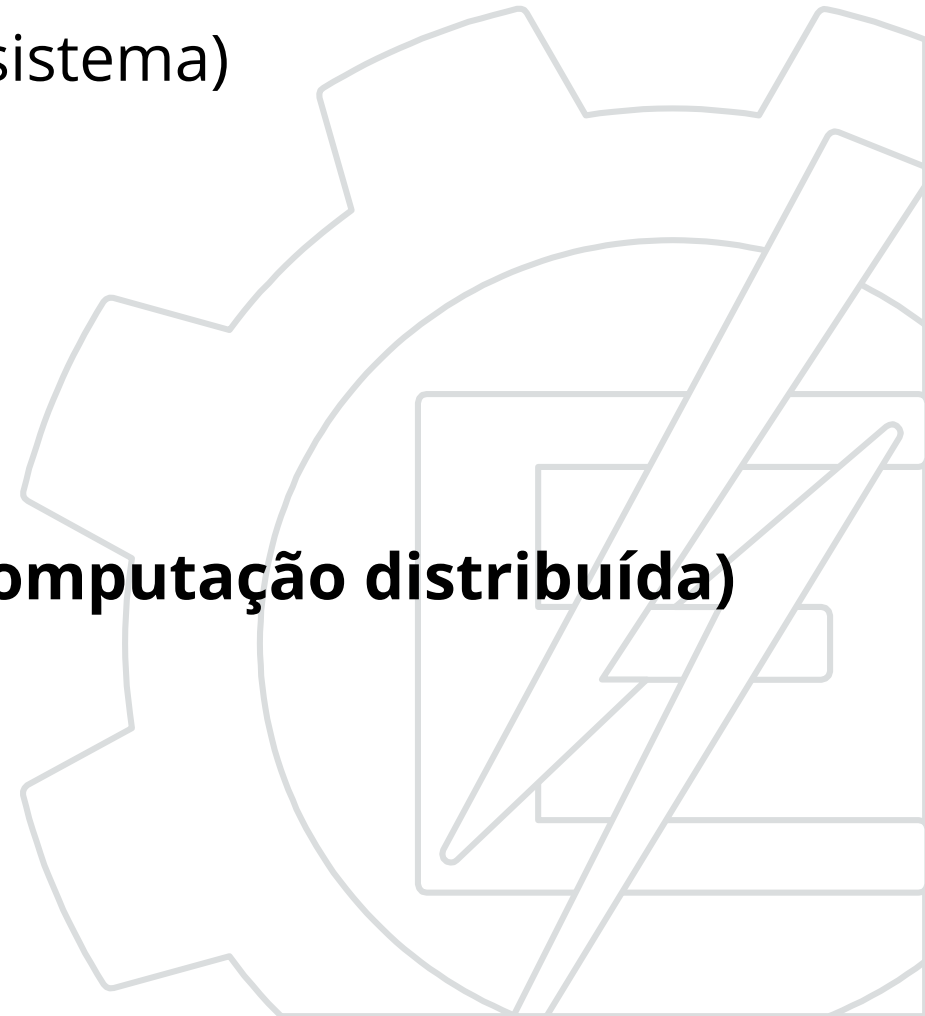
- Uma boa solução para o problema da seção crítica deve satisfazer os seguintes requisitos:
 - 1) Exclusão mútua:** se um processo i (P_i) está na seção crítica, nenhum outro processo pode entrar nela;
 - 2) Progresso garantido:** se nenhum outro processo está na seção crítica, um processo que tente fazê-lo não pode ser detido indefinidamente;
 - 3) Espera limitada:** se um processo deseja entrar na seção crítica, há um limite na quantidade de vezes que outros processos que podem entrar nela antes dele (evitar a inanição - *starvation*);
 - 4) Independência da arquitetura:** o processo não pode funcionar somente se estiver sendo executado em uma configuração específica de dispositivo, por exemplo: quantidade de núcleos e/ou frequência determinados.

Comunicação entre Processos

InterProcess Communication (IPC)

- Frequentemente processos precisam se comunicar.
 - A comunicação é mais eficiente se for estruturada e não utilizar interrupções.
 - Questões importantes:
 - **Como** um processo passa informação para outro?
 - Como garantir que processos não **invadam** espaços uns dos outros, nem entrem em **conflito**?
 - Qual a sequência adequada quando existe **dependência** entre processos?
- 

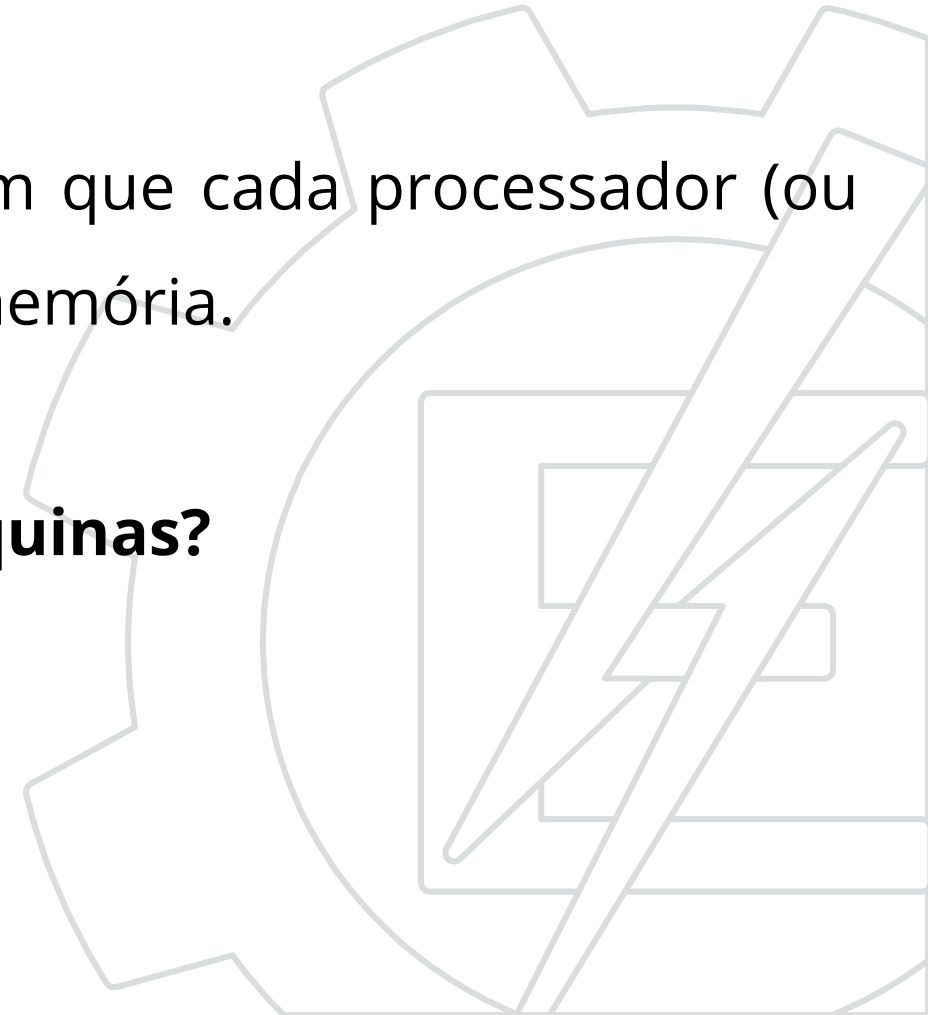
- Espera ocupada (*busy waiting*)
- *Sleep / WakeUp* (primitivas - chamadas de sistema)
- Semáforos (variáveis de controle)
- Monitores (primitiva de alto nível)
- **Troca de Mensagens** (ambiente de computação distribuída)



Troca de Mensagens

ou Passagem de Mensagens

- Semáforos e Monitores:
 - Projetados para exclusão mútua em processadores que compartilham algum espaço de memória;
 - Não funcionam com sistemas distribuídos, em que cada processador (ou grupo de processadores) possui sua própria memória.
- **Como passar informação entre diferentes máquinas?**



Troca de Mensagens

ou Passagem de Mensagens

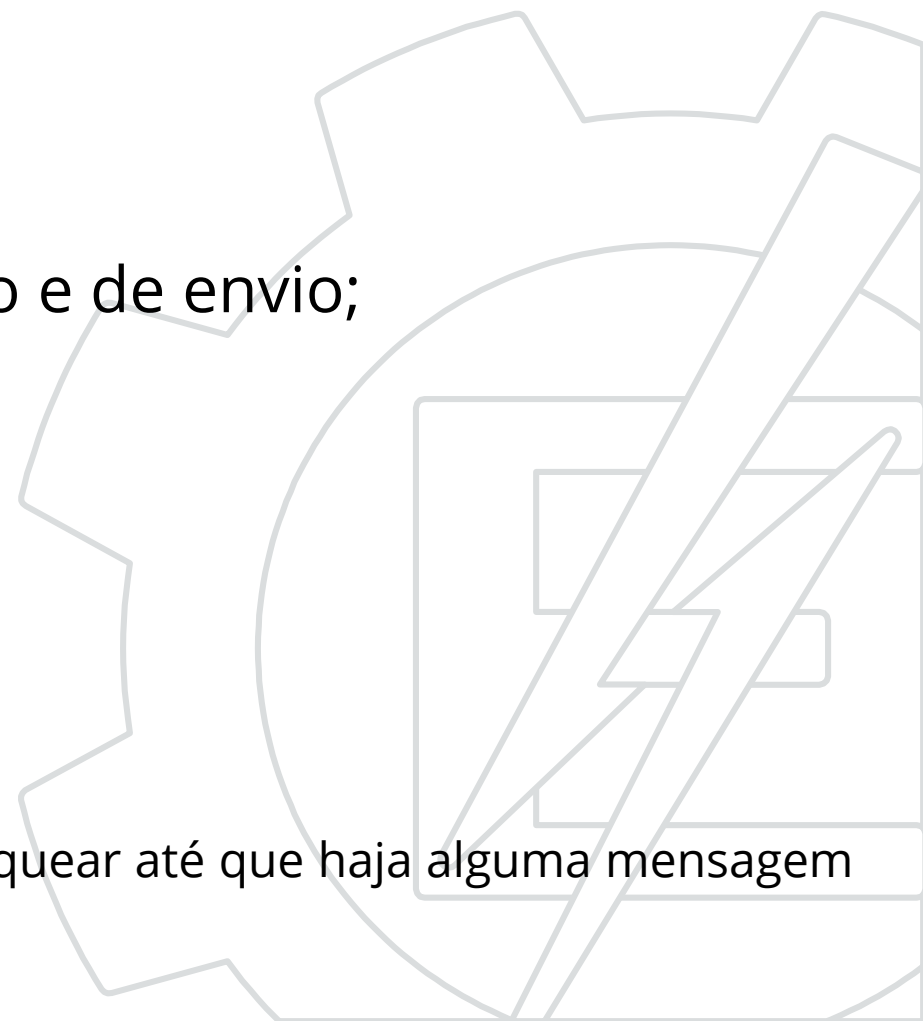
- Implementação do *link* de comunicação:
 - Físico:
 - Memória compartilhada
 - Barramento de *hardware*
 - Rede
 - Lógico:
 - Direto ou indireto
 - Síncrono ou assíncrono
 - Automático ou *buffer* explícito



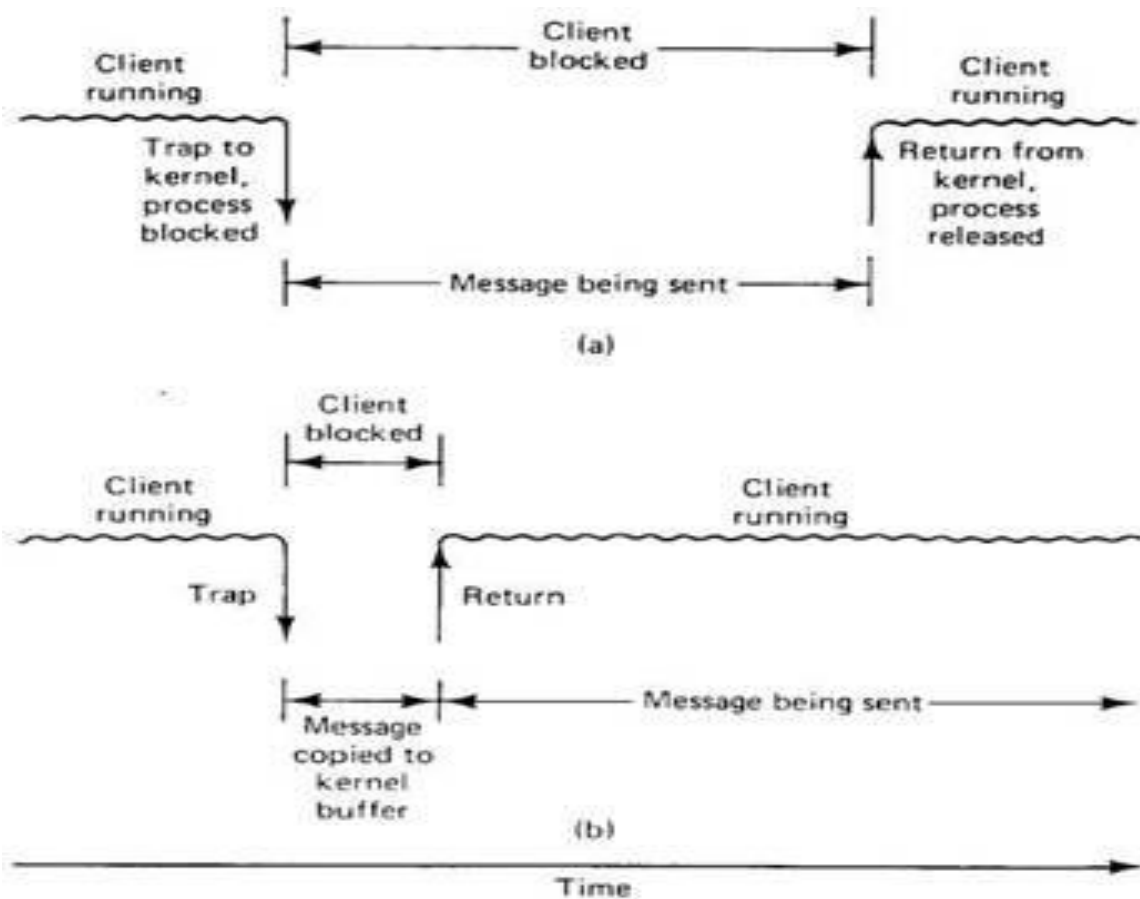
Troca de Mensagens

ou Passagem de Mensagens

- Processos enviam e recebem **mensagens** em vez de ler e escrever em variáveis compartilhadas;
- Podem ser bloqueantes ou não-bloqueantes;
- Mecanismos de registro/confirmação recebimento e de envio;
- **Chamadas de sistema** – Primitivas:
 - ***send***(*destino, &msg*)
 - ***receive***(*fonte, &smg*)
 - Se não houver mensagem disponível, o receptor pode bloquear até que haja alguma mensagem (*blocked*) ou retornar com mensagem de erro (*unblocked*).



Troca de Mensagens ou Passagem de Mensagens



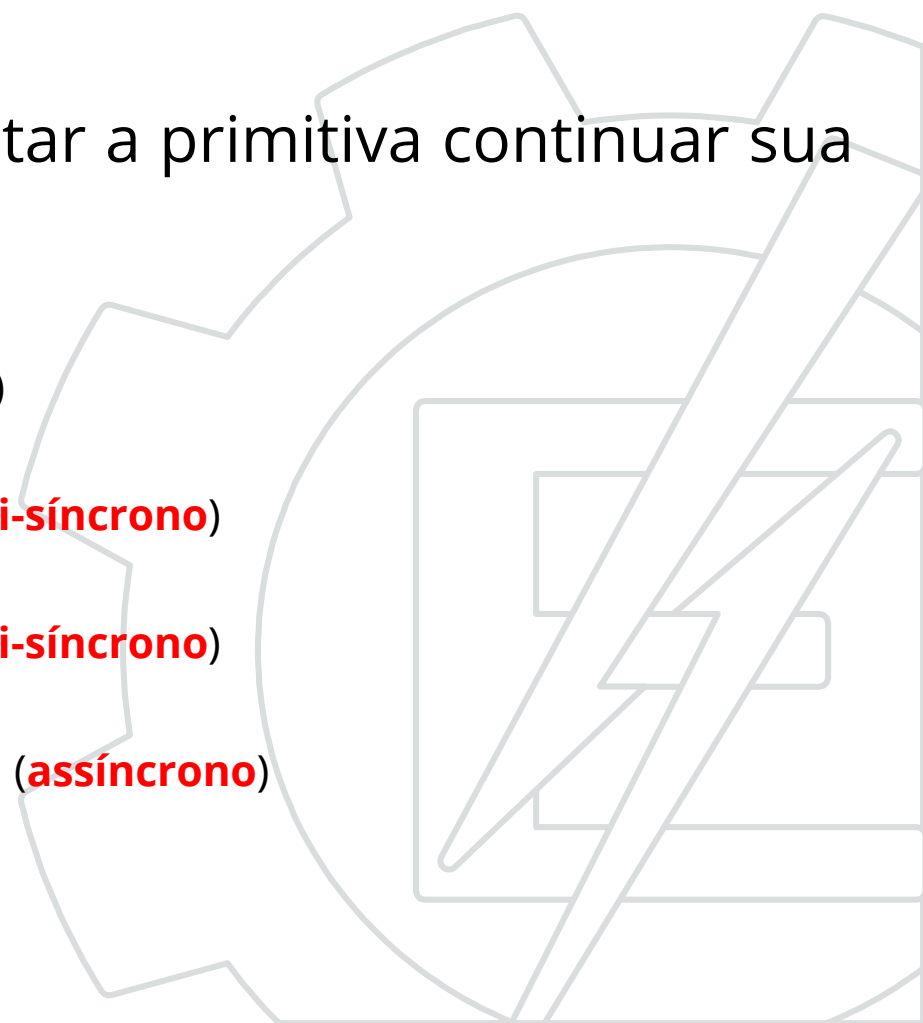
- **Bloqueantes:** quando o processo que as executa fica bloqueado até que a operação seja bem sucedida.
- **Não-bloqueantes:** quando o processo que executar a primitiva continuar sua execução normal.

(a) Primitiva *send* bloqueante. (b) Primitiva *send* não-bloqueante.

Troca de Mensagens

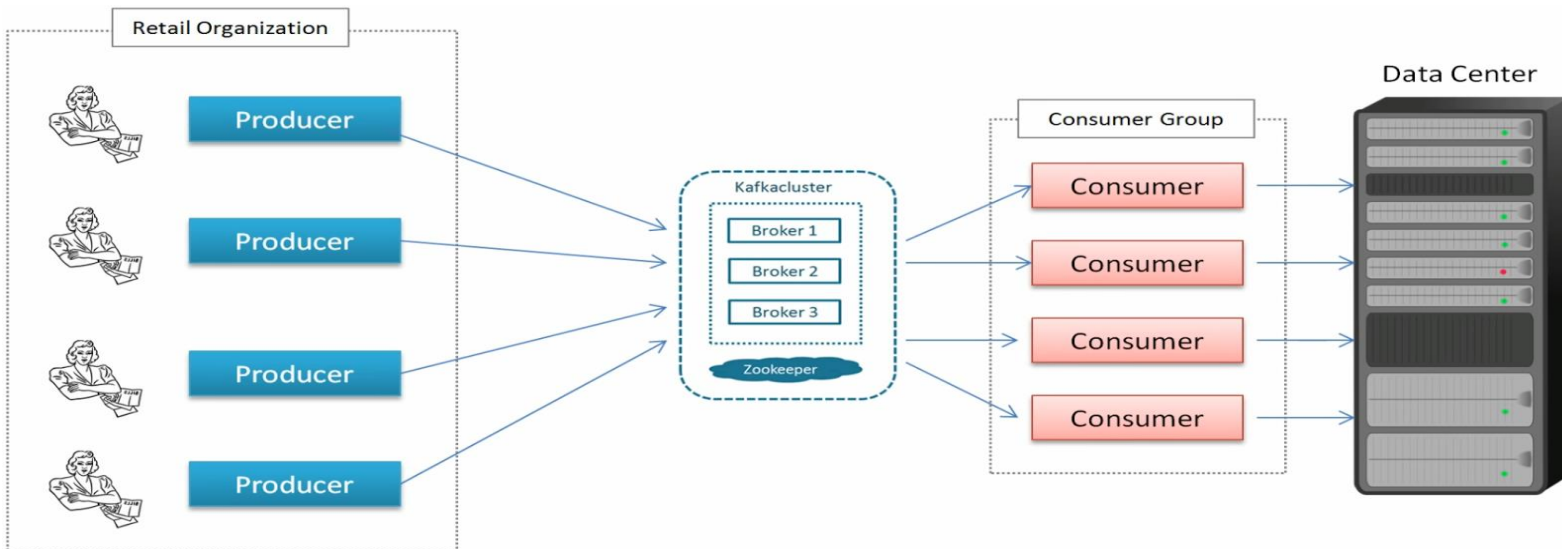
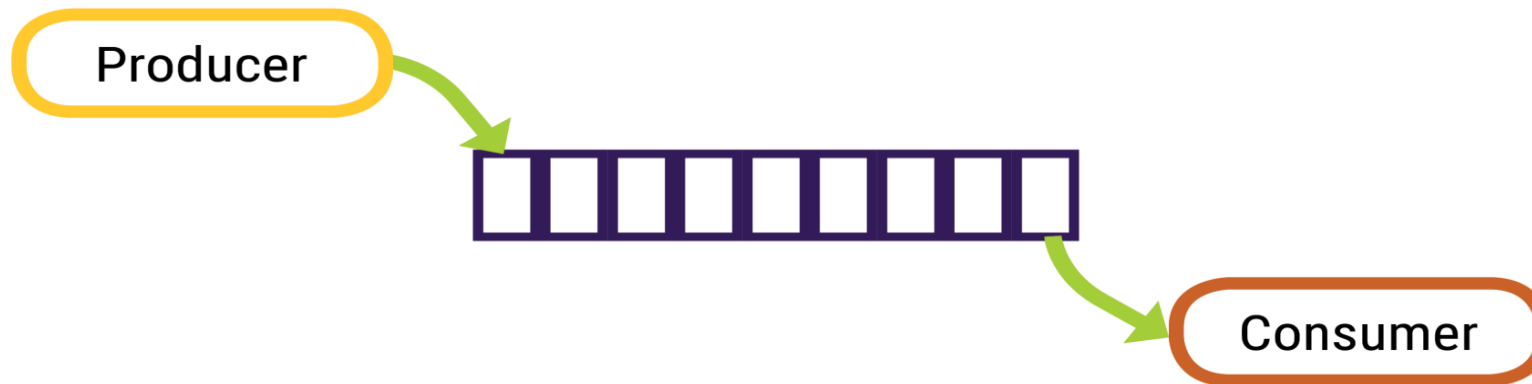
ou Passagem de Mensagens

- **Bloqueantes:** quando o processo que as executa fica bloqueado até que a operação seja bem sucedida.
- **Não-bloqueantes:** quando o processo que executar a primitiva continuar sua execução normal.
 1. Envio Bloqueante → Recebimento Bloqueante (**síncrono**)
 2. Envio Bloqueante → Recebimento Não-bloqueante (**semi-síncrono**)
 3. Envio Não-bloqueante → Recebimento Bloqueante (**semi-síncrono**)
 4. Envio Não-bloqueante → Recebimento Não-bloqueante (**assíncrono**)



Troca de Mensagens ou Passagem de Mensagens

- Assumimos que as mensagens enviadas e não lidas são guardadas pelo S.O.
- Neste caso, usamos um número de mensagens igual ao do *buffer*.



Troca de Mensagens

ou Passagem de Mensagens

- Assumimos que as mensagens enviadas e não lidas são guardadas pelo S.O.
- Neste caso, usamos um número de mensagens igual ao do *buffer*.

```
#define N 100

void producer(void) {
    int item;
    message m;

    while (TRUE) {
        item = produce_item();
        receive(consumer, &m);
        build_message(&m, item);
        send(consumer, &m);
    }
}
```

```
void consumer() {
    int item, i;
    message m;

    for(i=0; i<N; i++)
        send(producer, &m);

    while (TRUE) {
        receive(producer, &m);
        item = extract_item(&m);
        send(producer, &m);
        consume_item(item);
    }
}
```

Troca de Mensagens

ou Passagem de Mensagens

Envio Bloqueante → Recebimento Bloqueante

- Mecanismos de comunicação **síncronos**:
 - RPC (*Remote Procedure Call*);
 - RMI (*Remote Method Invocation* - Java);
 - Caixas postais (*mailboxes*); e
 - Portos (*ports*).



Troca de Mensagens

Remote Procedure Call (RPC)

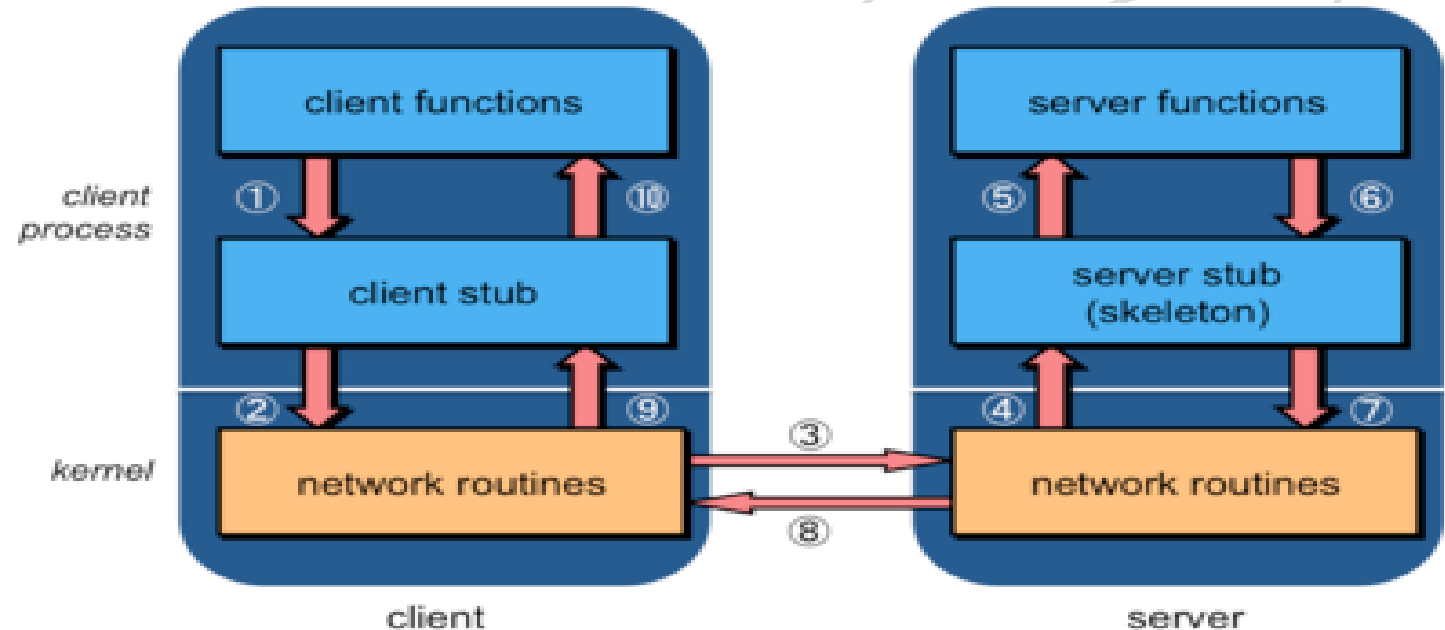


Troca de Mensagens

Remote Procedure Call (RPC)

- Um processo pode comandar a execução de um procedimento situado em outra máquina.
- O processo chamador deverá ficar bloqueado até que o procedimento chamado termine (mecanismo síncrono).
- Tanto a chamada quanto o retorno podem envolver a troca de mensagem, passando parâmetros.

Os itens 3 e 8 são mensagens, os demais itens são chamadas ou retornos de procedimentos locais comuns.



Troca de Mensagens

Remote Procedure Call (RPC)

- As mensagens trocadas em uma comunicação RPC são bem estruturadas e não são nada mais do que pacotes de dados.
- Cada mensagem é endereçada a um **RPC daemon** ouvinte para uma porta a um sistema remoto e cada uma delas contém a identificação da função específica a ser executada e os parâmetros que devem ser passados para a função.
- A função é executada de acordo com os requisitos e qualquer resposta ou saída é enviada ao requisitante como uma mensagem separada.
- A porta é simplesmente um número incluído no início do pacote da mensagem.

Troca de Mensagens

Remote Procedure Call (RPC)

- **Desafios:**

- Dificuldade de passagem de **parâmetros por referência**.
- Se servidor e cliente possuem **diferentes representações de informação**, existe a necessidade de conversão.
- Diferenças de arquitetura: as máquinas podem **diferir no armazenamento de palavras**.
Por exemplo: *long* do C tem tamanhos diferentes, dependendo se o S.O. for 32 ou 64 bits.
- **Falhas semânticas:** se o servidor para de funcionar quando executava uma RPC – O que dizer ao cliente?
 - Ele pode tentar novamente – o que pode não ser desejável (p.ex.: tarefas simples vs. atualização de BD).
 - Principais abordagens: **“no mínimo uma vez”**, **“exatamente uma vez”** e **“no máximo uma vez”**.

Troca de Mensagens

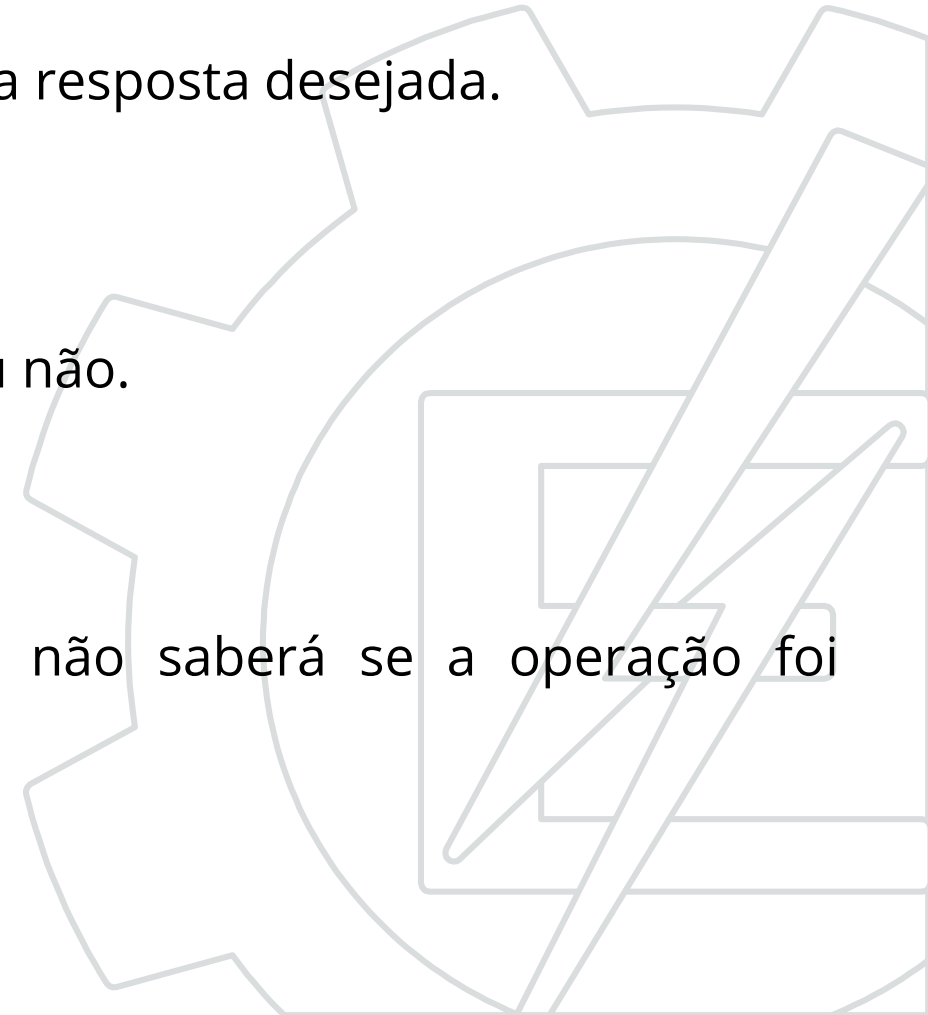
Remote Procedure Call (RPC)

- **Diferentes representações de informação:**
 - É necessário manter a atenção às **diferentes formas de representação das informações** entre clientes e servidores.
 - Um exemplo são as representações conhecidas como ***big-endian*** e ***little-endian*** onde, no primeiro caso o *byte* mais significativo é armazenado primeiro, enquanto que, no segundo caso, o *byte* menos significativo é armazenado primeiro.
 - Vários sistemas RPC apresentam uma representação dos dados independente da máquina. Essa representação é conhecida como ***external data representation (XDR)***.

Troca de Mensagens

Remote Procedure Call (RPC)

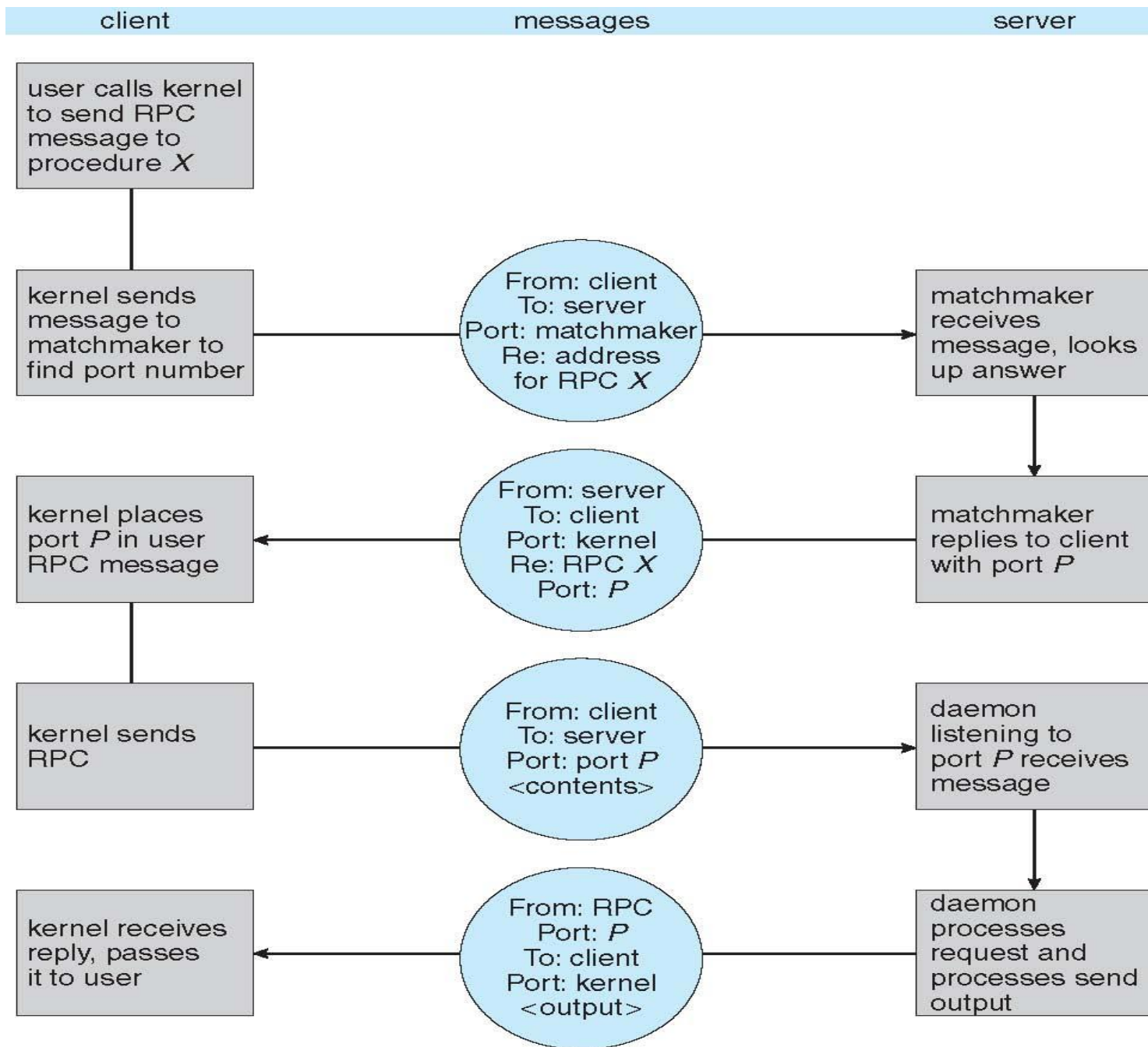
- **Falhas semânticas:**
 - **No mínimo uma vez** (*at-least-once*):
 - O cliente fica retransmitindo o pedido até que tenha a resposta desejada.
 - **Exatamente uma vez** (*maybe*):
 - Toda chamada é executada exatamente uma vez;
 - Cliente não sabe se o servidor processou o pedido ou não.
 - Não há medidas de tolerância à falhas.
 - **No máximo uma vez** (*at-most-once*):
 - Se o servidor cai, o cliente saberá do erro, mas não saberá se a operação foi executada.



Troca de Mensagens

Remote Procedure Call (RPC)

Mecanismo de tolerância a falhas			Semântica
Retransmissão de request	Filtro de duplicatas	Re-execução do método ou retransmissão do reply	
Não	---	---	Talvez
Sim	Não	Re-execução do método	No mínimo uma vez
Sim	Sim	Retransmissão do reply	No máximo uma vez



Troca de Mensagens

Remote Procedure Call (RPC)



Troca de Mensagens

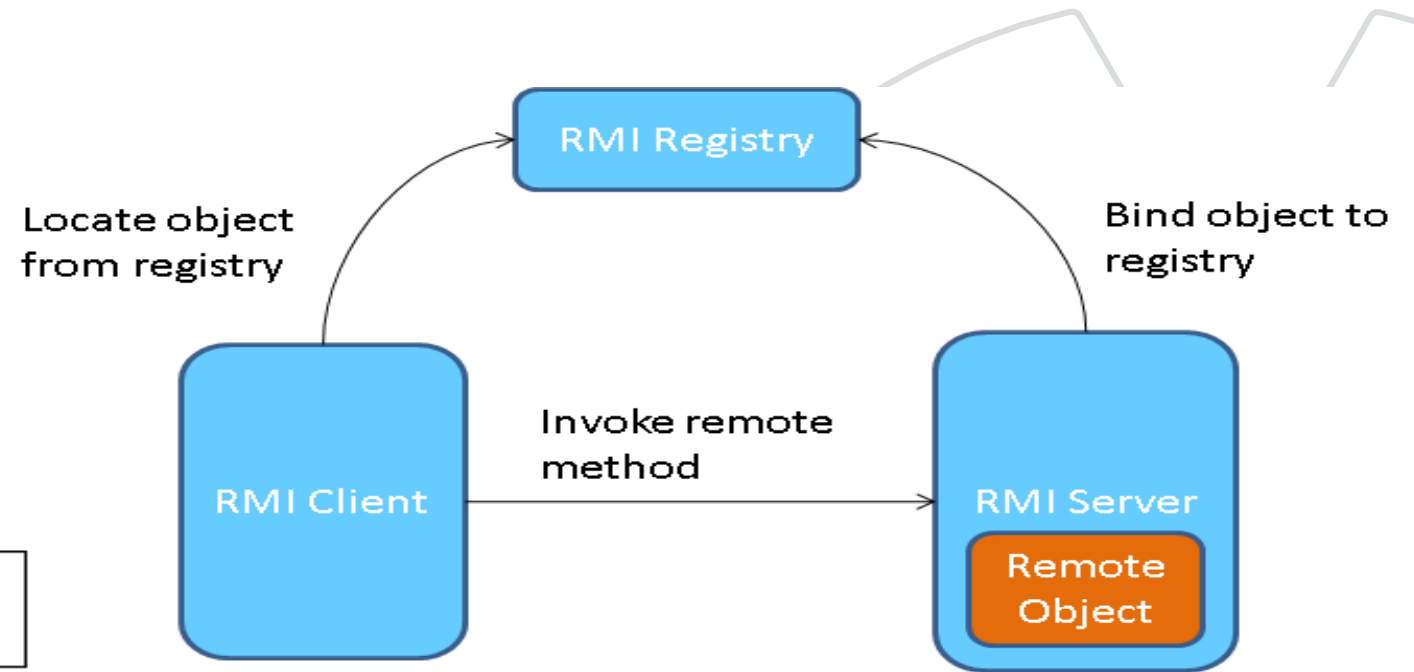
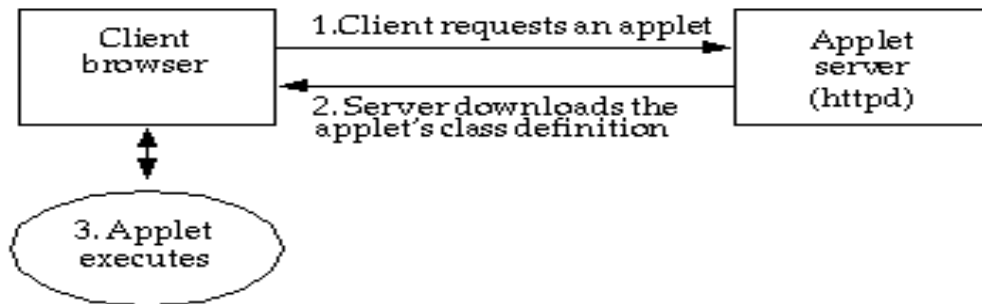
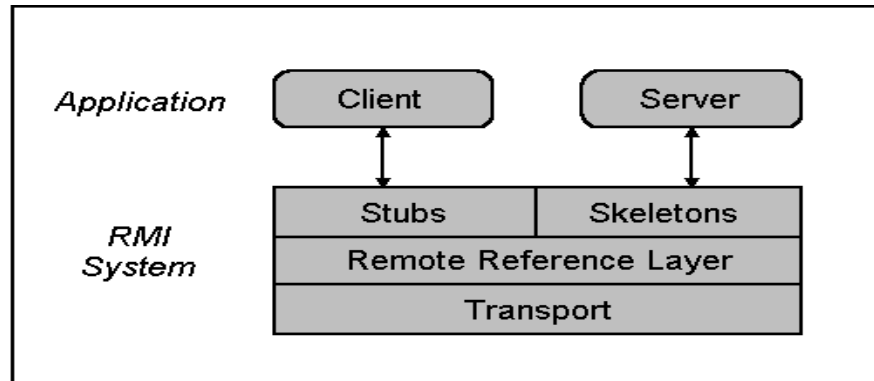
RMI (*Remote Method Invocation*) - Java



Troca de Mensagens

RMI (*Remote Method Invocation* - Java)

- Permite que um **objeto** ativo possa interagir com objetos de outras máquinas virtuais Java.



Troca de Mensagens

Caixas postais (*mailboxes*)



Troca de Mensagens

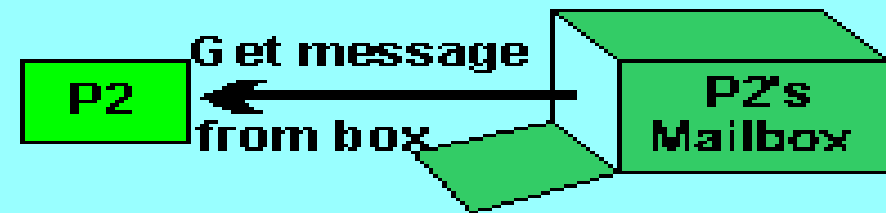
Caixas postais (*mailboxes*)

- Estruturas de dados;
- São **filas de mensagens** não associadas, a princípio, a nenhum processo;
- Lugar para se colocar um certo número de mensagens (limitado).
- Mensagens são enviadas ou lidas da caixa postal e não diretamente dos processos.
- Quando um processo tenta enviar para uma caixa cheia, ele é suspenso até que uma mensagem seja removida da caixa.

Send(P2, &message)

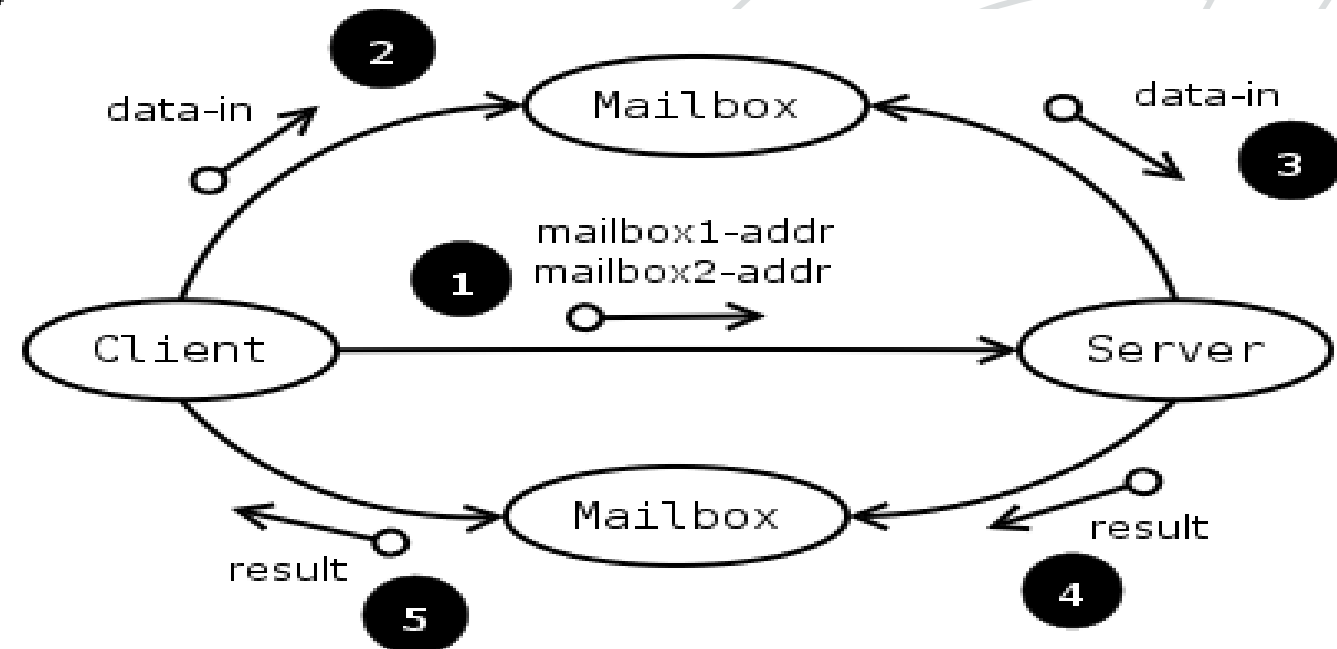
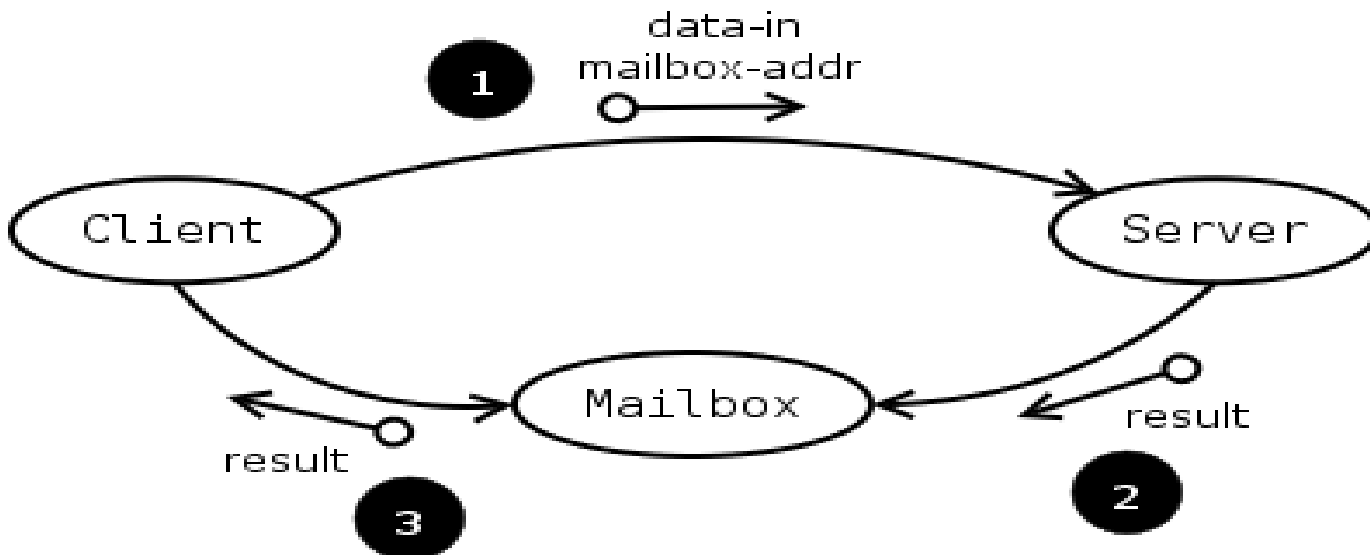


Receive(P1, &message)



Troca de Mensagens

Caixas postais (*mailboxes*)



Troca de Mensagens

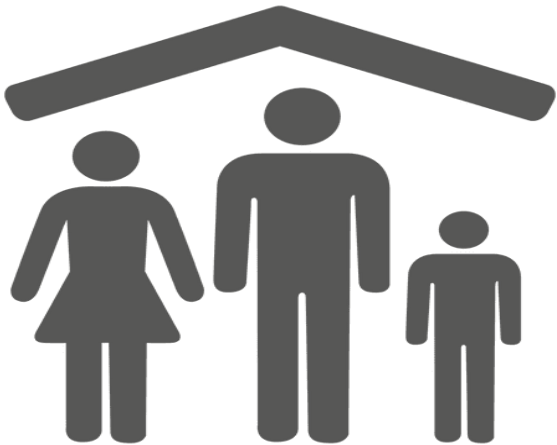
Portos (*ports*)



Troca de Mensagens

Portos (*ports*)

- Consiste em elementos do sistema que permitem a comunicação entre conjunto de processos.
- Conexão de dados virtual ou lógica, utilizada para que programas troquem informação diretamente. Ex.: TCP – numerados de 1 a 65535.
- Cada porto é como uma caixa postal, porém com um **dono**, que será o processo que o criar.



Port Number	Protocol	Application
20	TCP	FTP Data
21	TCP	FTP Control
22	TCP	SSH
23	TCP	Telnet
25	TCP	SMTP
53	UDP,TCP	DNS
67,68	UDP	DHCP
69	UDP	TFTP
80	TCP	HTTP
110	TCP	POP3
161	UDP	SNMP
443	TCP	SSL
16,384-32,767	UDP	RTP-based Voice and Video

TCP/UDP Port Numbers

7 Echo	554 RTSP	2745 Bagle.H	6891-6901 Windows Live
19 Chargen	546-547 DHCPv6	2967 Symantec AV	6970 Quicktime
20-21 FTP	560 rmonitor	3050 Interbase DB	7212 GhostSurf
22 SSH/SCP	563 NNTP over SSL	3074 XBOX Live	7648-7649 CU-SeeMe
23 Telnet	587 SMTP	3124 HTTP Proxy	8000 Internet Radio
25 SMTP	591 FileMaker	3127 MyDoom	8080 HTTP Proxy
42 WINS Replication	593 Microsoft DCOM	3128 HTTP Proxy	8086-8087 Kaspersky AV
43 WHOIS	631 Internet Printing	3222 GLBP	8118 Privoxy
49 TACACS	636 LDAP over SSL	3260 iSCSI Target	8200 VMware Server
53 DNS	639 MSDP (PIM)	3306 MySQL	8500 Adobe ColdFusion
67-68 DHCP/BOOTP	646 LDP (MPLS)	3389 Terminal Server	8767 TeamSpeak
69 TFTP	691 MS Exchange	3689 iTunes	8866 Bagle.B
70 Gopher	860 iSCSI	3690 Subversion	9100 HP JetDirect
79 Finger	873 rsync	3724 World of Warcraft	9101-9103 Bacula
80 HTTP	902 VMware Server	3784-3785 Ventrilo	9119 MXit
88 Kerberos	989-990 FTP over SSL	4333 mSQL	9800 WebDAV
102 MS Exchange	993 IMAP4 over SSL	4444 Blaster	9898 Dabber
110 POP3	995 POP3 over SSL	4664 Google Desktop	9988 Rbot/Spybot
113 Ident	1025 Microsoft RPC	4672 eMule	9999 Urchin
119 NNTP (Usenet)	1026-1029 Windows Messenger	4899 Radmin	10000 Webmin
123 NTP	1080 SOCKS Proxy	5000 UPnP	10000 BackupExec
135 Microsoft RPC	1080 MyDoom	5001 Slingbox	10113-10116 NetIQ
137-139 NetBIOS	1194 OpenVPN	5001 iperf	11371 OpenPGP
143 IMAP4	1214 Kazaa	5004-5005 RTP	12035-12036 Second Life
161-162 SNMP	1241 Nessus	5050 Yahoo! Messenger	12345 NetBus
177 XDMCP	1311 Dell OpenManage	5060 SIP	13720-13721 NetBackup
179 BGP	1337 WASTE	5190 AIM/ICQ	14567 Battlefield
201 AppleTalk	1433-1434 Microsoft SQL	5222-5223 XMPP/Jabber	15118 Dpnet/Oddbob
264 BGMP	1512 WINS	5432 PostgreSQL	19226 AdminSecure
318 TSP	1589 Cisco VQP	5500 VNC Server	19638 Ensim
381-383 HP Openview	1701 L2TP	5554 Sasser	20000 Usermin
389 LDAP	1723 MS PPTP	5631-5632 pcAnywhere	24800 Synergy
411-412 Direct Connect	1725 Steam	5800 VNC over HTTP	25999 Xfire
443 HTTP over SSL	1741 CiscoWorks 2000	5900+ VNC Server	27015 Half-Life
445 Microsoft DS	1755 MS Media Server	6000-6001 X11	27374 Sub7
464 Kerberos	1812-1813 RADIUS	6112 Battle.net	28960 Call of Duty
465 SMTP over SSL	1863 MSN	6129 DameWare	31337 Back Orifice
497 Retrospect	1985 Cisco HSRP	6257 WinMX	33434+ traceroute
500 ISAKMP	2000 Cisco SCCP	6346-6347 Gnutella	
512 rexec	2002 Cisco ACS	6500 GameSpy Arcade	
513 rlogin	2049 NFS	6566 SANE	
514 syslog	2082-2083 cPanel	6588 AnalogX	
515 LPD/LPR	2100 Oracle XDB	6665-6669 IRC	
520 RIP	2222 DirectAdmin	6679/6697 IRC over SSL	
521 RiPing (IPv6)	2302 Halo	6699 Napster	
540 UUCP	2483-2484 Oracle DB	6881-6999 BitTorrent	

IANA port assignments published at <http://www.iana.org/assignments/port-numbers>

Troca de Mensagens

Portos (ports)



- Legend
- Chat
 - Encrypted
 - Gaming
 - Malicious
 - Peer to Peer
 - Streaming

Problemas clássicos de sincronização

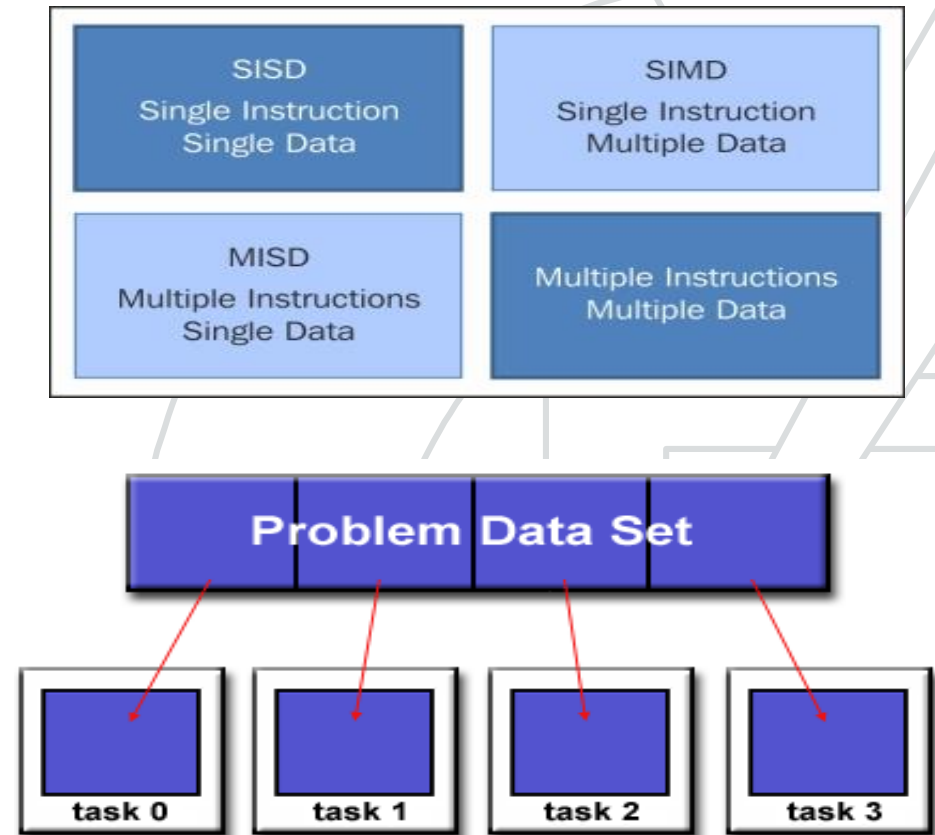
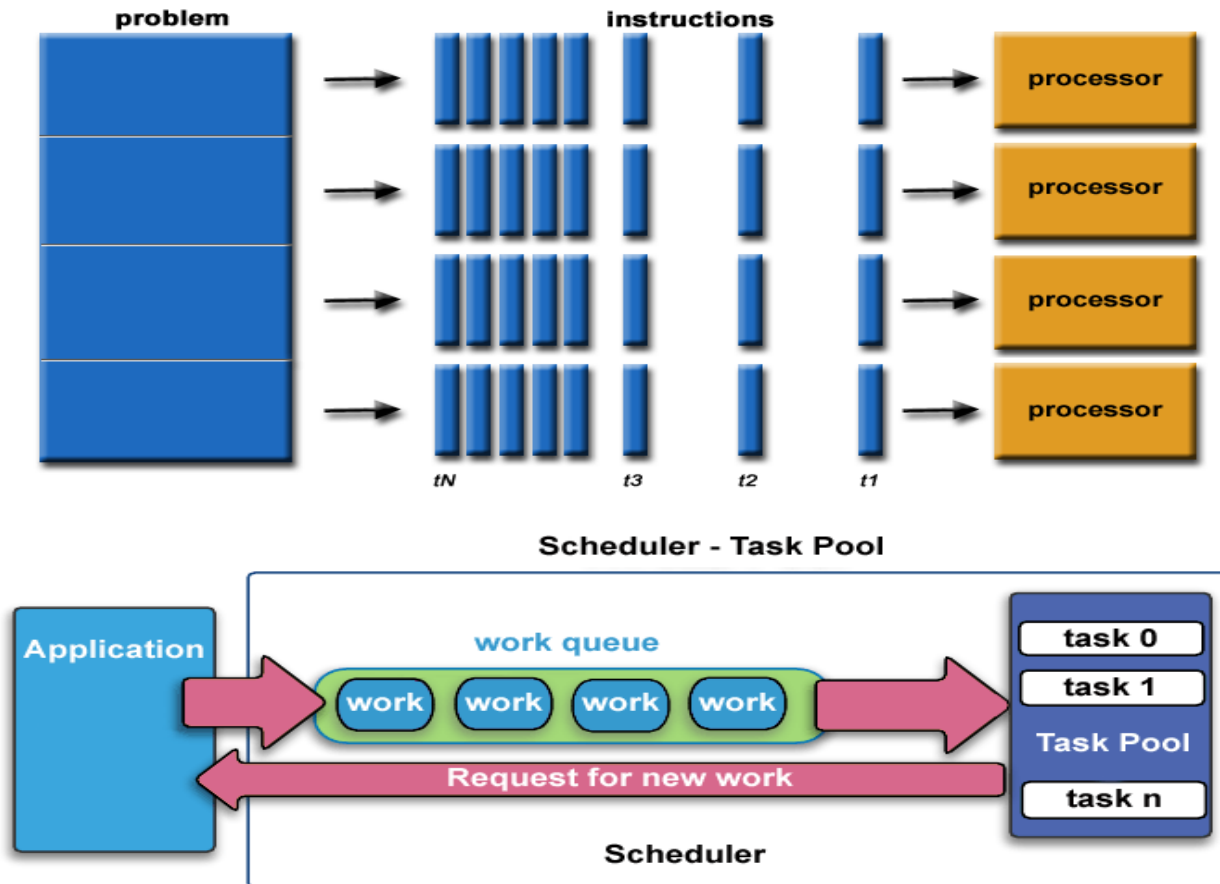
InterProcess Communication (IPC)



Problemas clássicos de sincronização

InterProcess Communication (IPC)

- Estes problemas apresentam modelos de comunicação entre processos que envolvem **sincronização e paralelismo**.



Problemas clássicos de sincronização

InterProcess Communication (IPC)

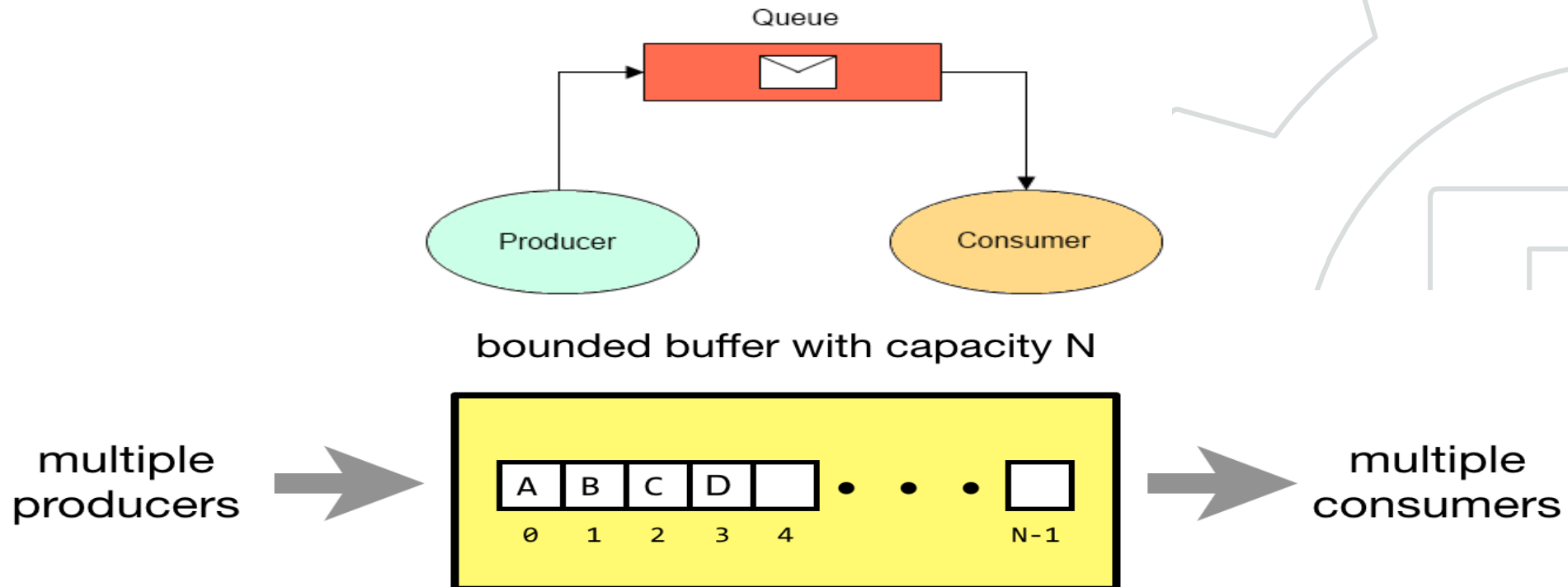
- Estes problemas apresentam **modelos de comunicação** entre processos que envolvem sincronização e paralelismo:
 1. Produtor/consumidor
 2. Leitores e escritores
 3. Jantar dos filósofos



Produtor/Consumidor

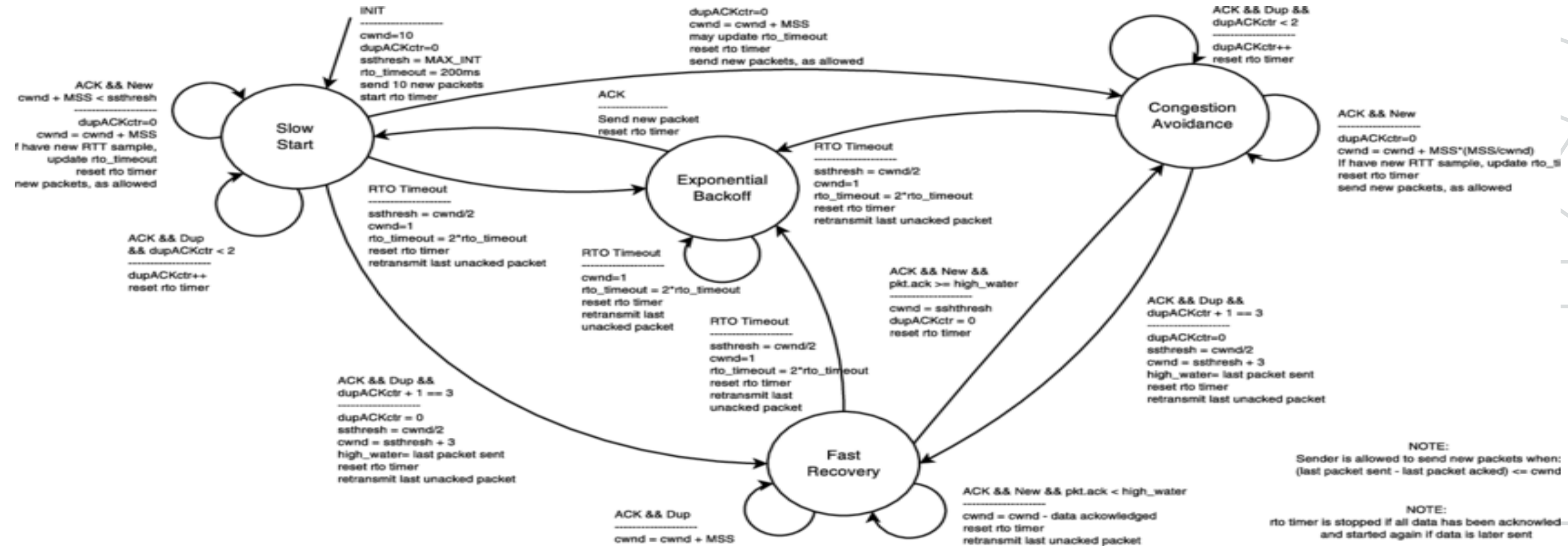
(Problema do *Buffer* Limitado)

- Já foi apresentado e utilizado em assuntos anteriores.
- É composto por entidades produtoras e entidades consumidoras e pelo compartilhamento de uma área de armazenamento limitada.
- Auxiliou na modelagem dos principais protocolos da internet.



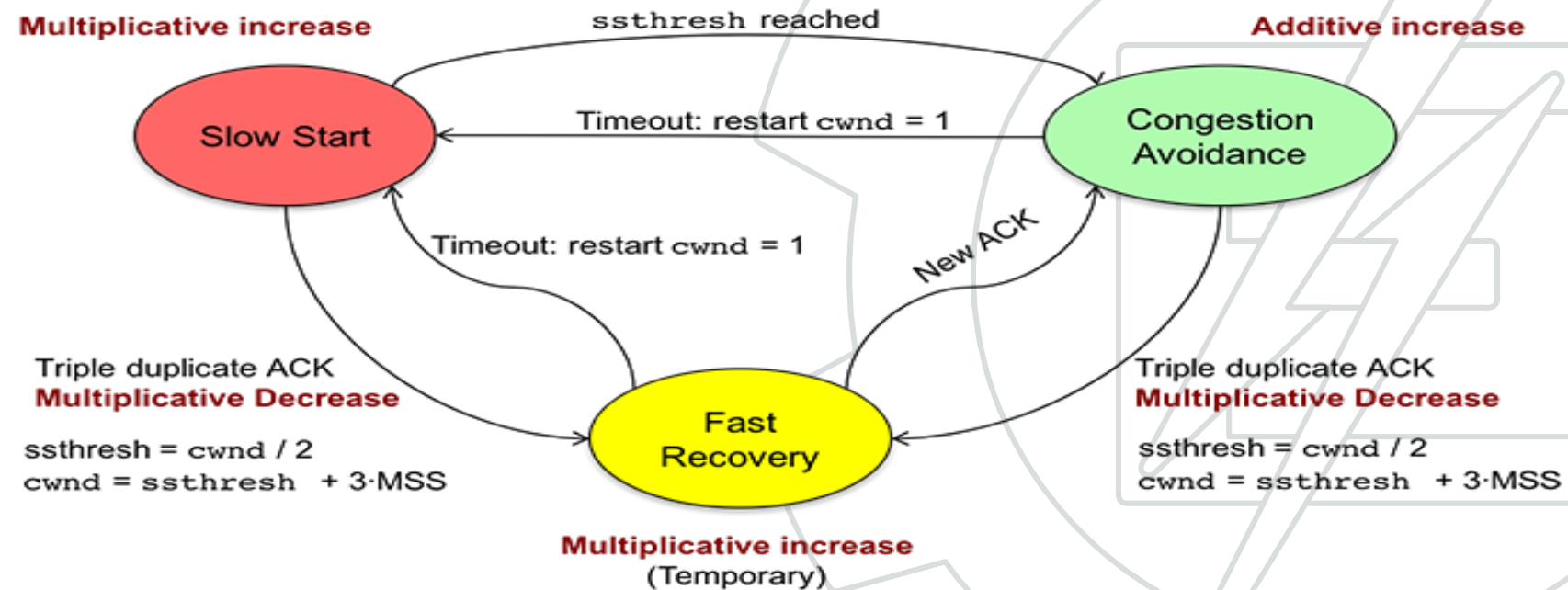
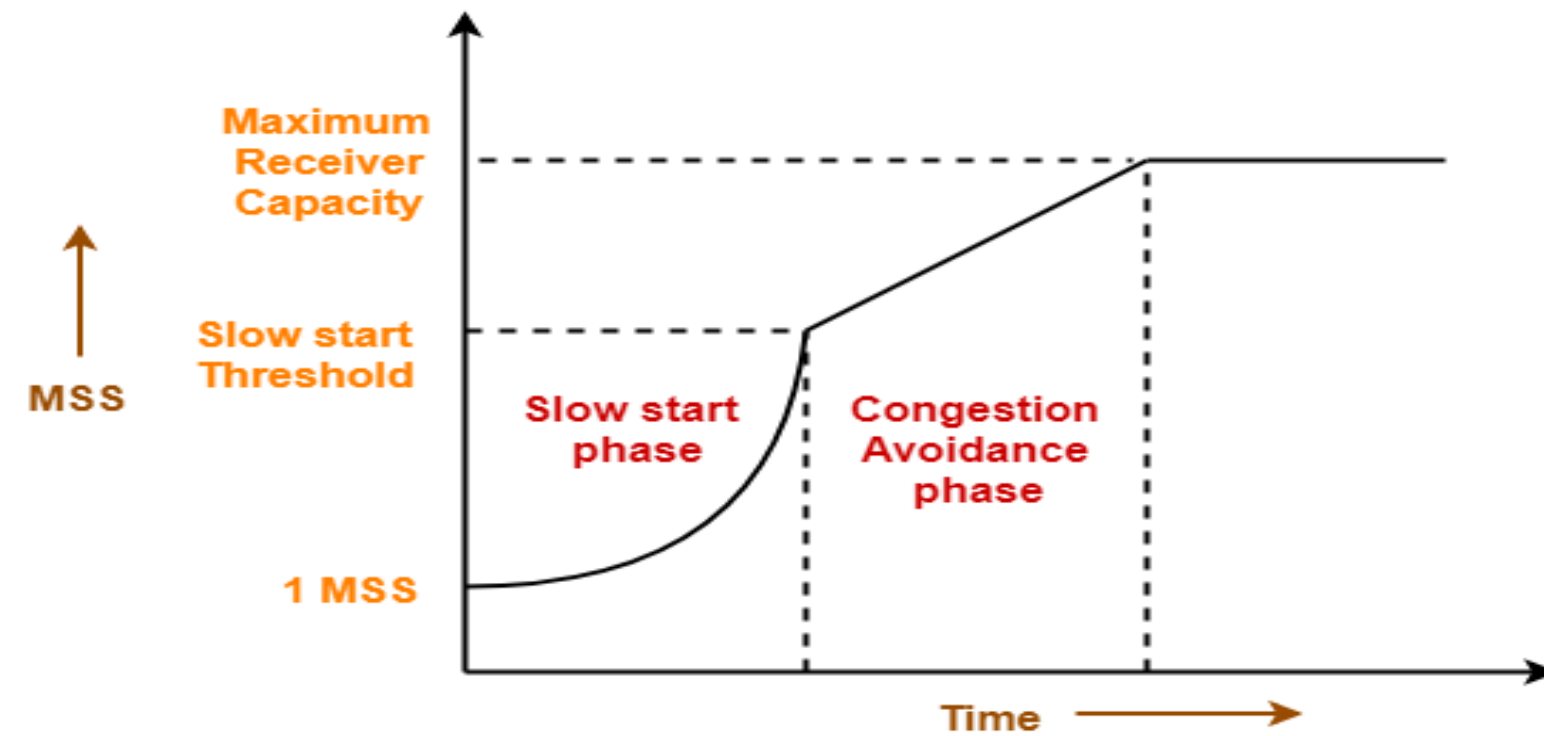
Produtor/Consumidor

TCP - Controle de congestionamento



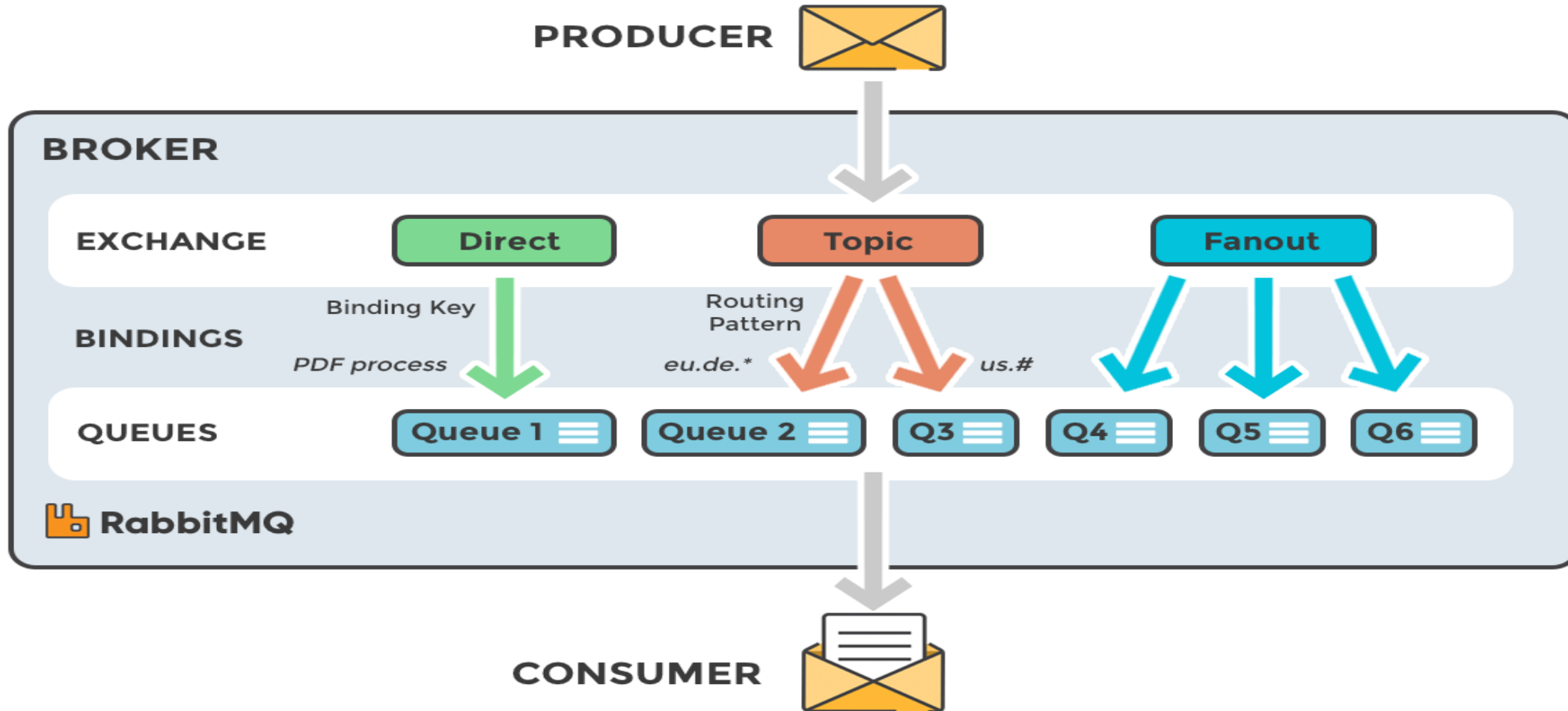
Produtor/Consumidor

- Controle de congestionamento



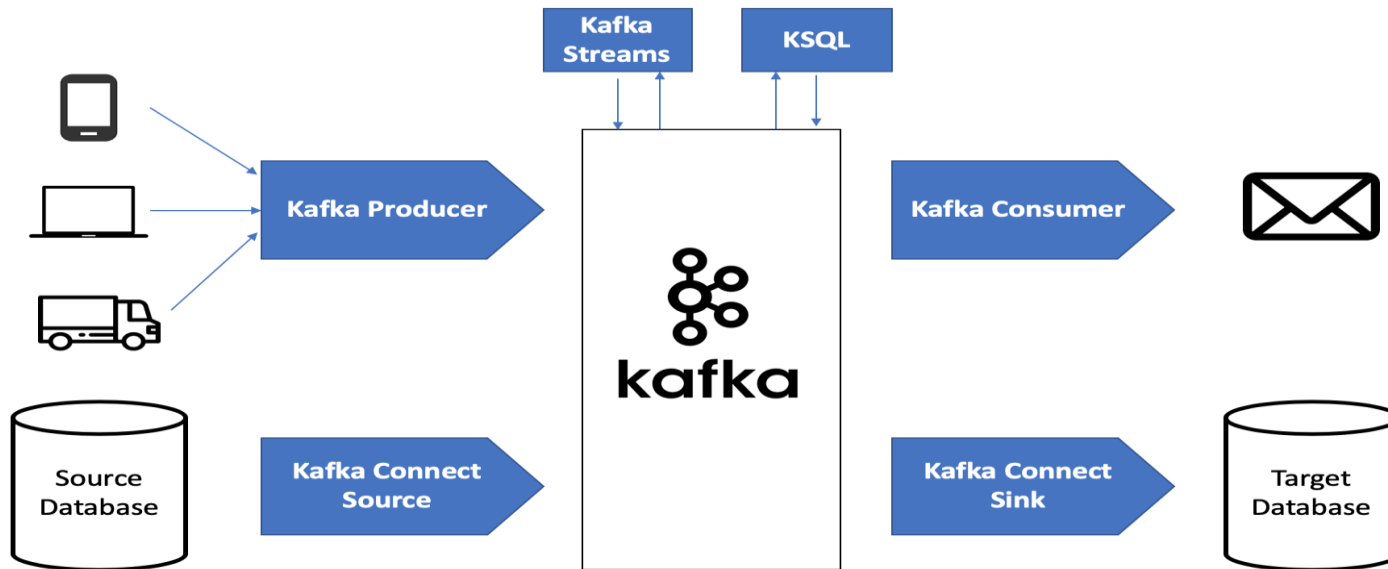
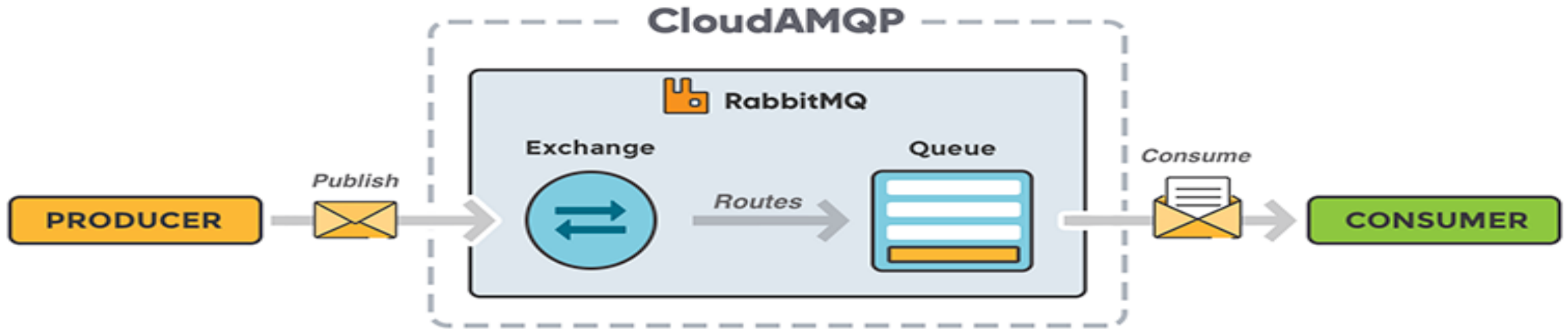
Produtor/Consumidor

(Problema do *Buffer* Limitado)



Produtor/Consumidor

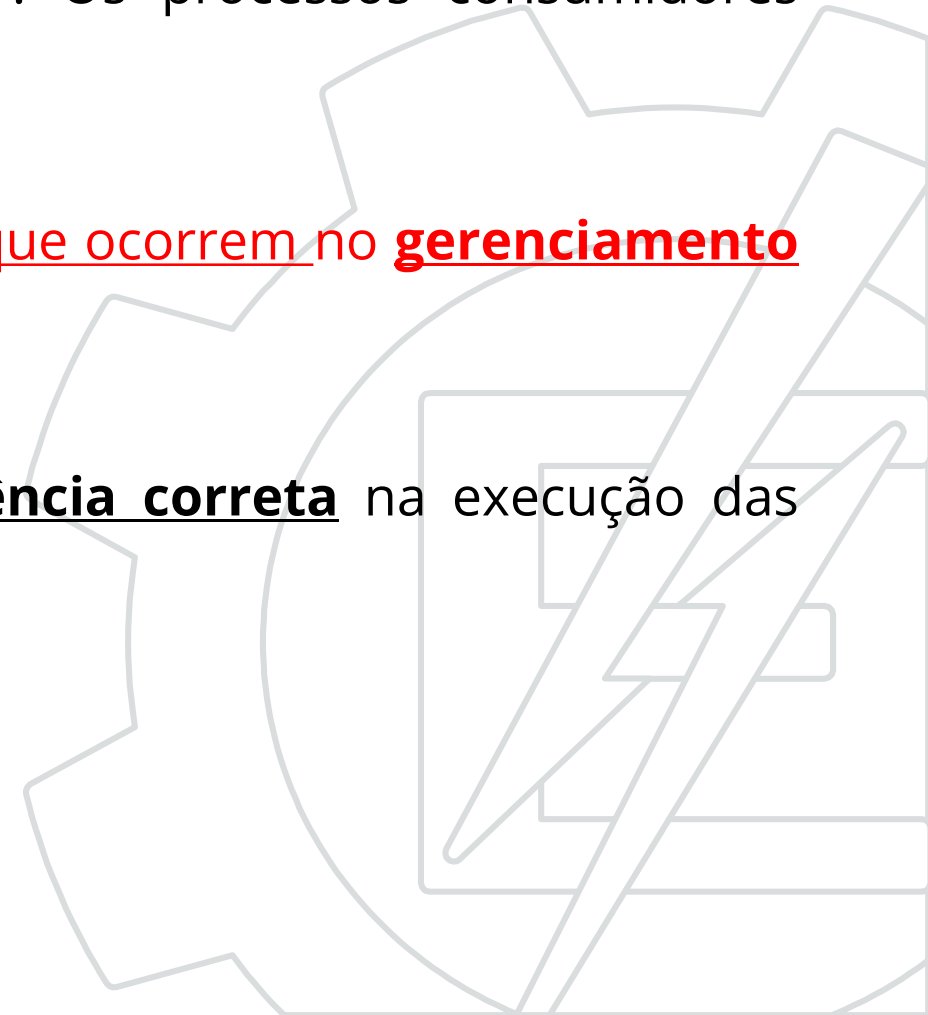
(Problema do *Buffer* Limitado)



Produtor/Consumidor

(Problema do *Buffer* Limitado)

- Consiste em um conjunto de processos que compartilham um mesmo *buffer*.
- Os processos produtores inserem informação no *buffer*. Os processos consumidores retiram informação deste *buffer*.
- Busca exemplificar de forma clara, situações de impasses que ocorrem no gerenciamento de processos de um sistema operacional.
- A solução precisa garantir a exclusão mútua e a sequência correta na execução das operações.

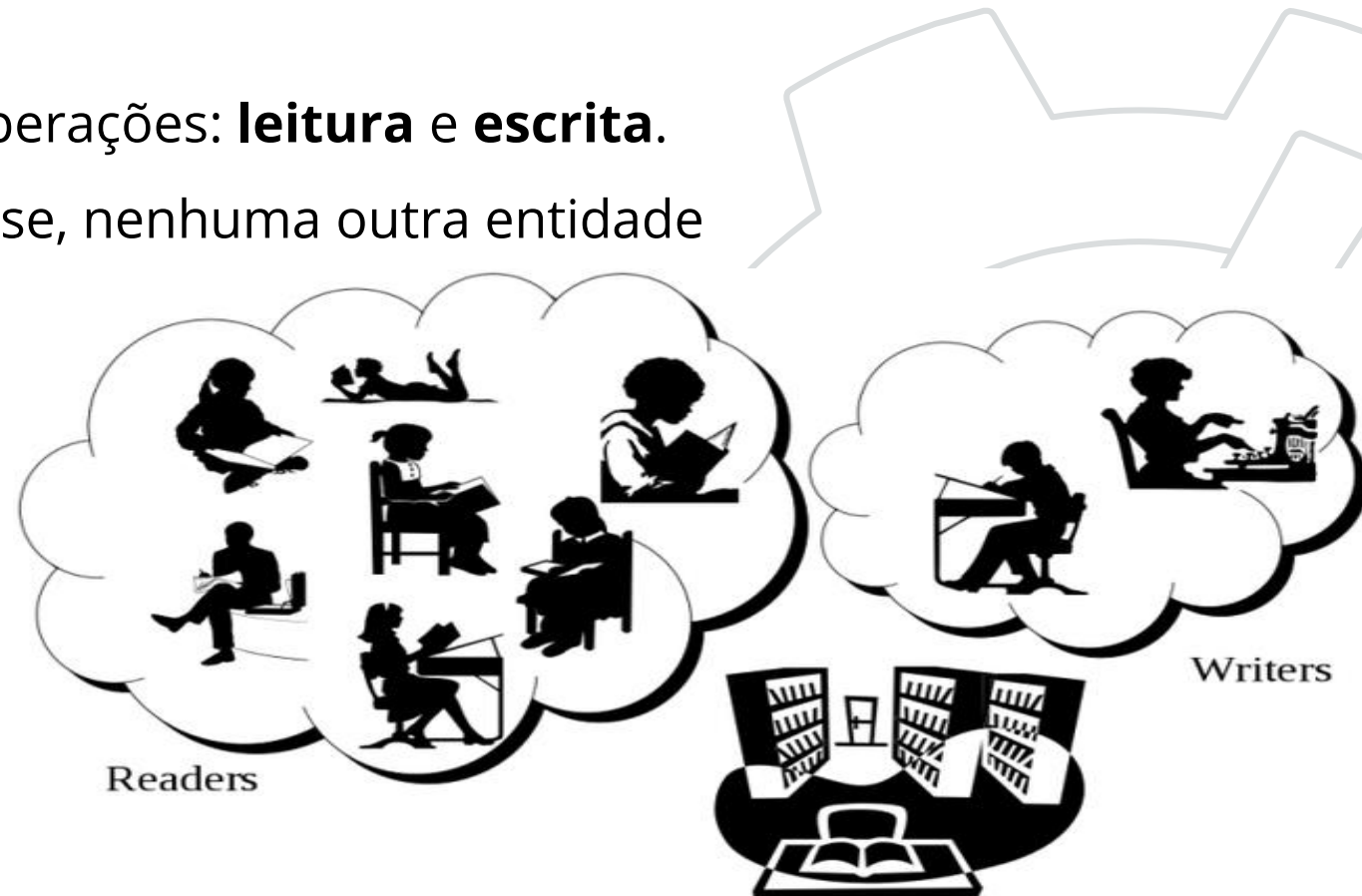


Leitores e escritores



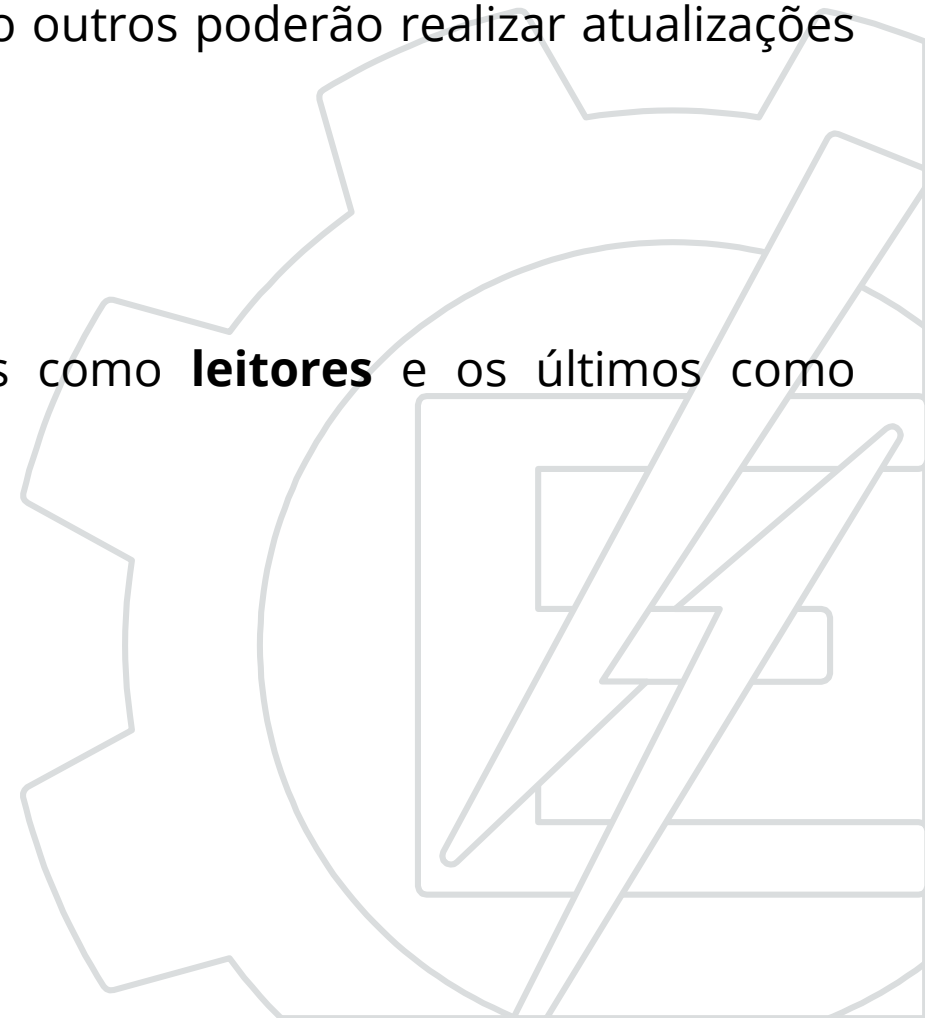
Leitores e escritores

- Modela **acessos a uma base de dados**.
- Um sistema com uma base de dados é **acessado simultaneamente** por diversas entidades.
- Essas entidades realizam dois tipos de operações: **leitura** e **escrita**.
- Se um processo necessita escrever na base, nenhuma outra entidade pode estar realizando acesso a ela.
Nem mesmo acesso de leitura.



Leitores e escritores

- Suponha que a base de dados será compartilhada com **diversos processos concorrentes**. Alguns destes processos podem somente ler a base de dados, enquanto outros poderão realizar atualizações na mesma (leitura e escrita).
- A distinção entre estes papéis é feita definindo os primeiros como **leitores** e os últimos como **escritores**.



Leitores e escritores

- Obviamente, se dois leitores acessam simultaneamente uma informação não há problema algum. Se um escritor ou um outro processo (leitor ou escritor) quiserem acessar simultaneamente a base de dados podem ocorrer alguns problemas.
- Para nos certificar que estes problemas não irão ocorrer, precisamos garantir que os **escritores têm acesso exclusivo à base de dados enquanto a atualizam** (bloqueiam a base de dados).

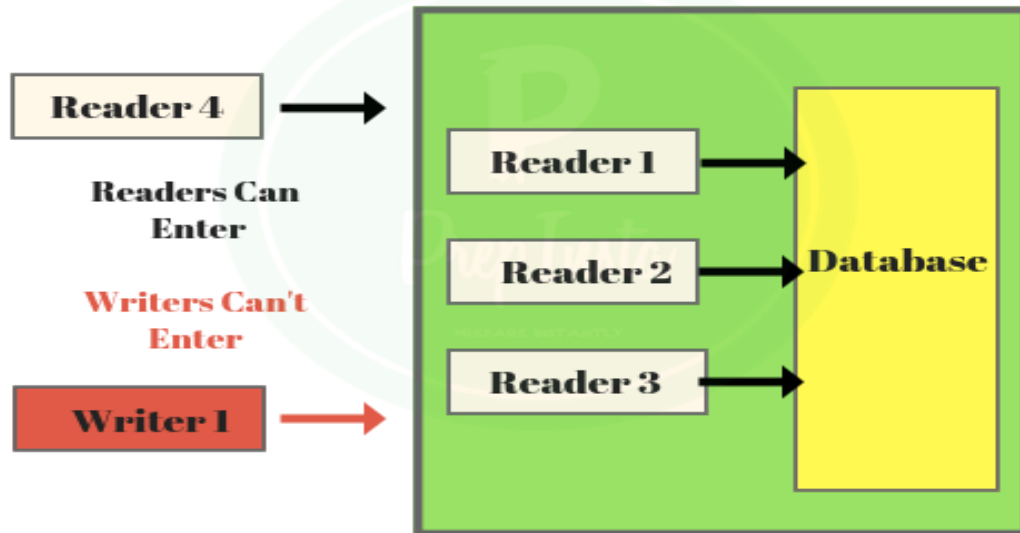
Leitores e escritores



Readers-Writers Operating System



When Readers are Accessing the Database



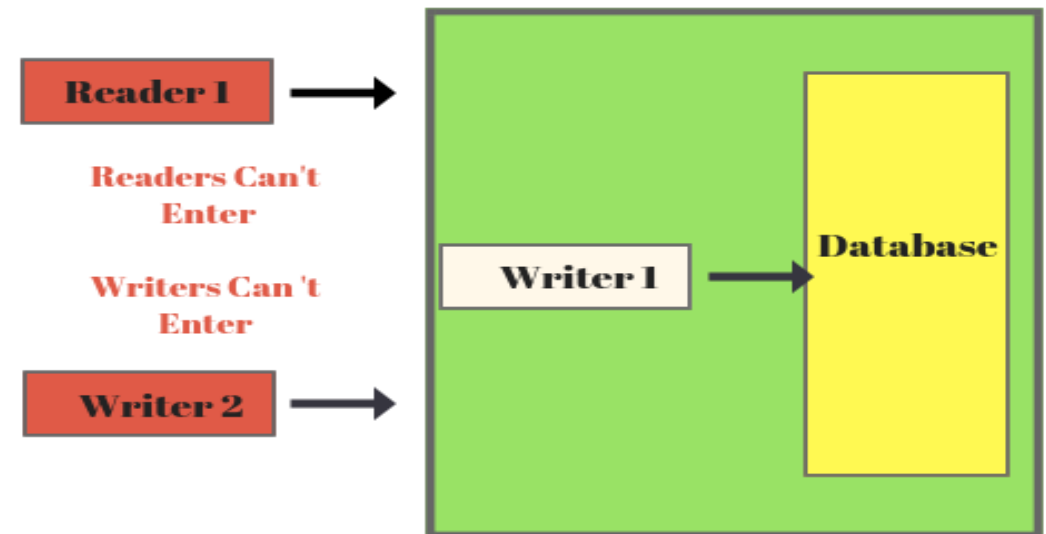
Access to DataBase



Readers-Writers Operating System



When Writer is Writing in the Database



Access to DataBase

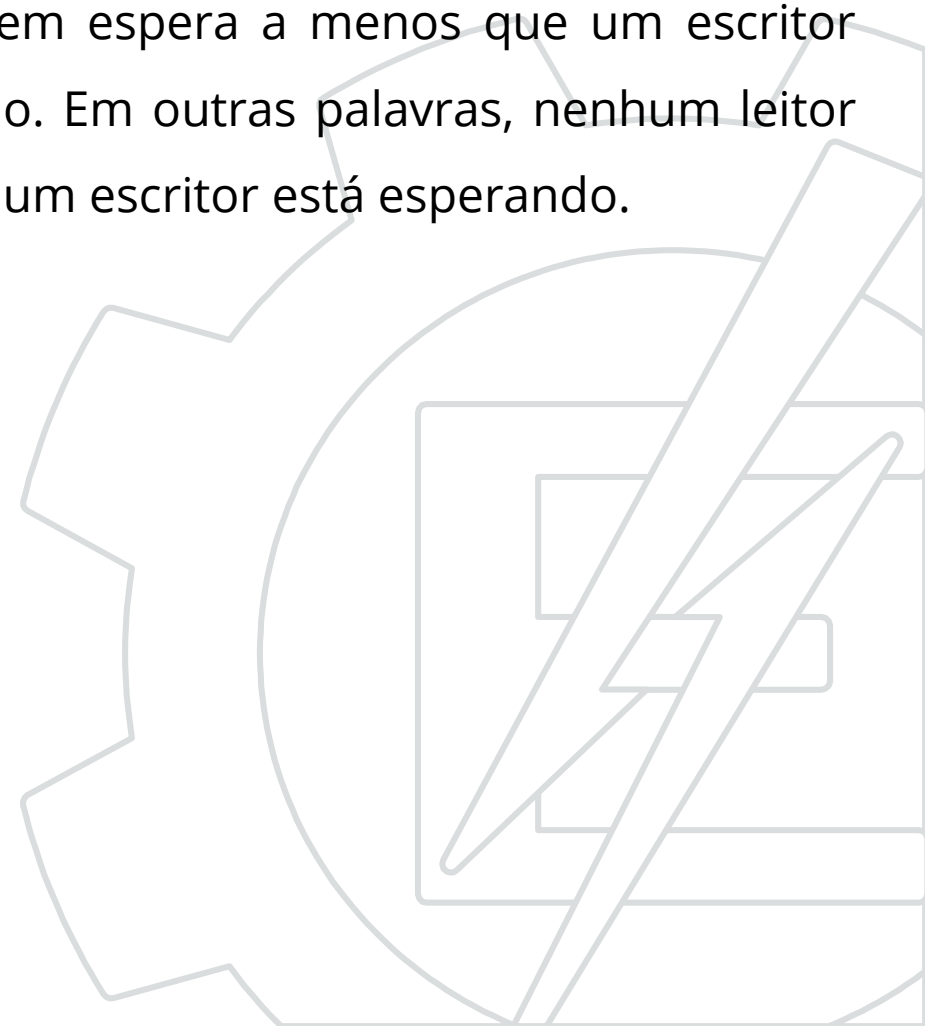
Leitores e escritores

- Escritores devem bloquear a base de dados.
- Leitores:
 - Se a base estiver desbloqueada:
 - Se for o 1º Leitor, deve bloqueá-la para evitar Escritor
 - Se já houver outro Leitor, basta utilizar a base de dados
 - Ao sair, verificar se há outro Leitor:
 - Se houver, deixa a base de dados bloqueada;
 - Se não houver, desbloqueia a base de dados.



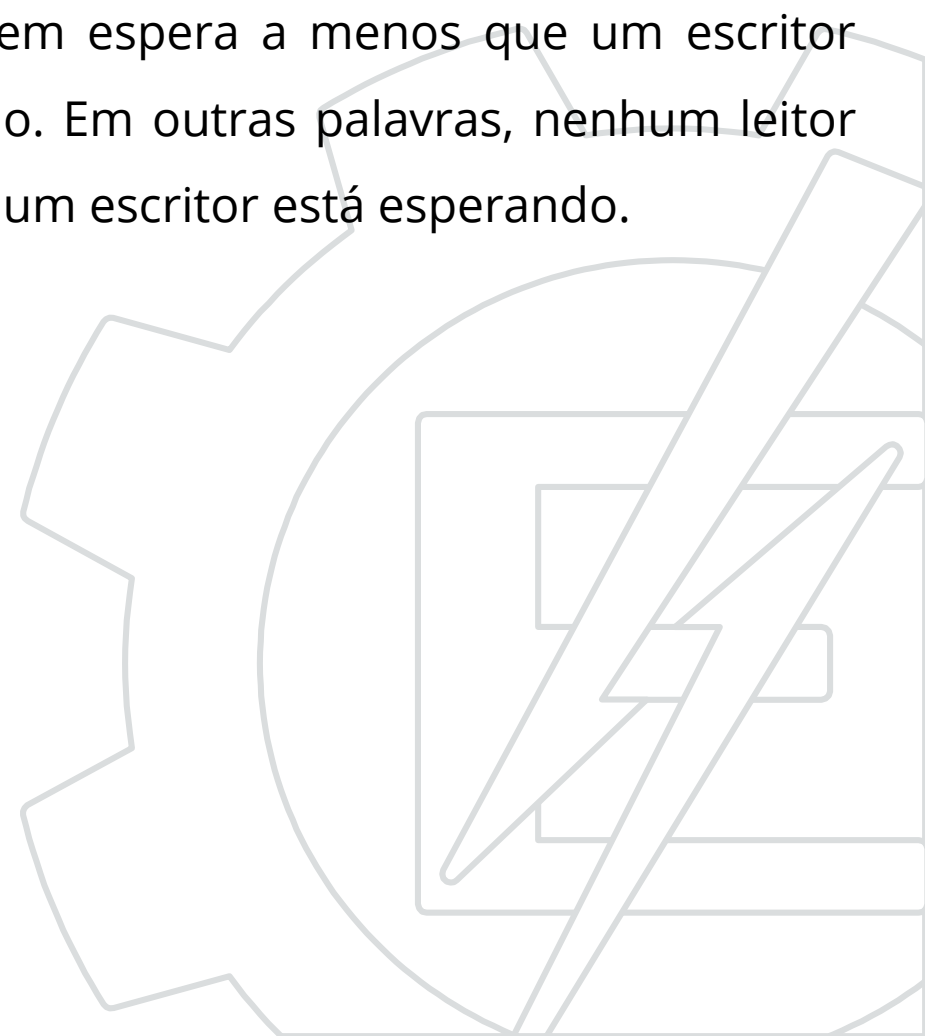
Leitores e escritores

- Desde que foi inicialmente proposto, este problema foi utilizado para testar as **primitivas de sincronização**. O problema de leitores-escretores possui algumas variações:
 1. A primeira delas requer que nenhum leitor seja mantido em espera a menos que um escritor tenha obtido permissão para utilizar o objeto compartilhado. Em outras palavras, nenhum leitor deve esperar que outros leitores terminem somente porque um escritor está esperando.



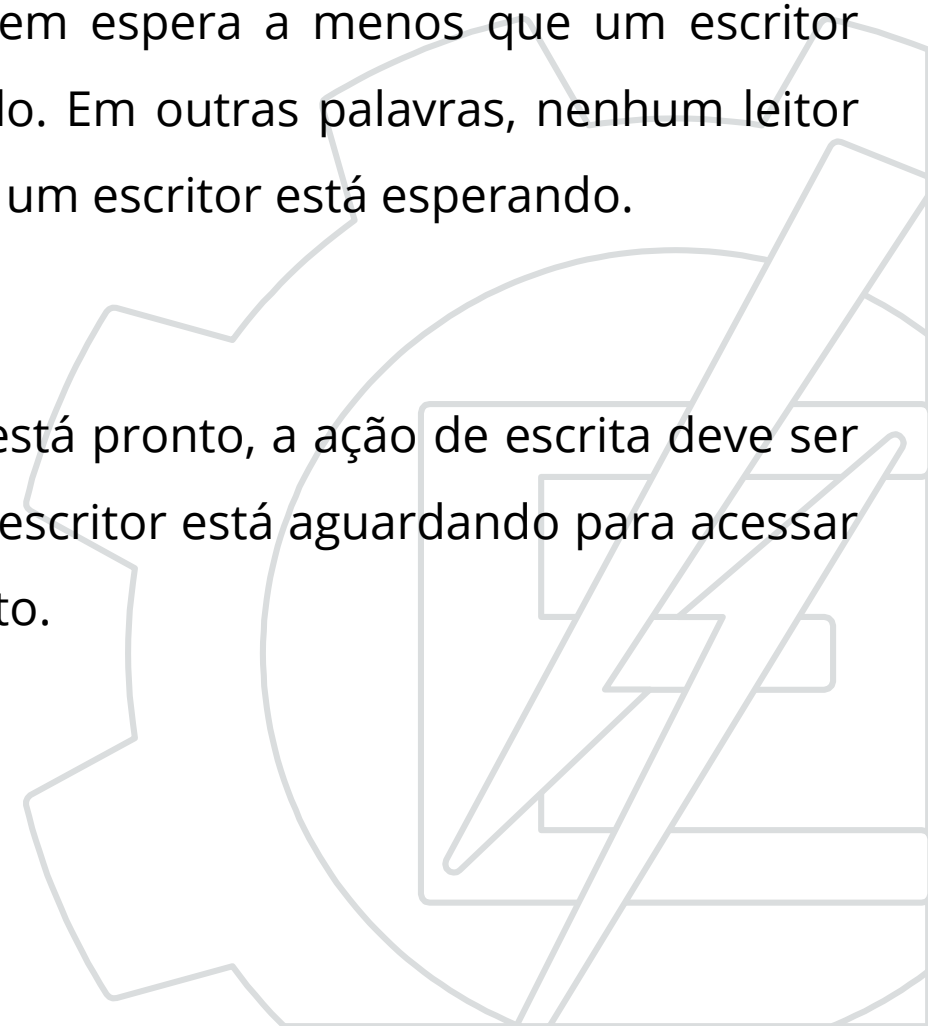
Leitores e escritores

- Desde que foi inicialmente proposto, este problema foi utilizado para testar as **primitivas de sincronização**. O problema de leitores-escretores possui algumas variações:
 1. A primeira delas requer que nenhum leitor seja mantido em espera a menos que um escritor tenha obtido permissão para utilizar o objeto compartilhado. Em outras palavras, nenhum leitor deve esperar que outros leitores terminem somente porque um escritor está esperando.
 - **Possível problema: Inanição do Escritor.**



Leitores e escritores

- Desde que foi inicialmente proposto, este problema foi utilizado para testar as **primitivas de sincronização**. O problema de leitores-escritores possui algumas variações:
 1. A primeira delas requer que nenhum leitor seja mantido em espera a menos que um escritor tenha obtido permissão para utilizar o objeto compartilhado. Em outras palavras, nenhum leitor deve esperar que outros leitores terminem somente porque um escritor está esperando.
 - Possível problema: Inanição do Escritor.
 2. A segunda abordagem requer que, uma vez que o escritor está pronto, a ação de escrita deve ser realizada o mais rápido possível. Em outras palavras, se um escritor está aguardando para acessar um objeto, nenhum novo leitor deve começar a ler este objeto.



Leitores e escritores

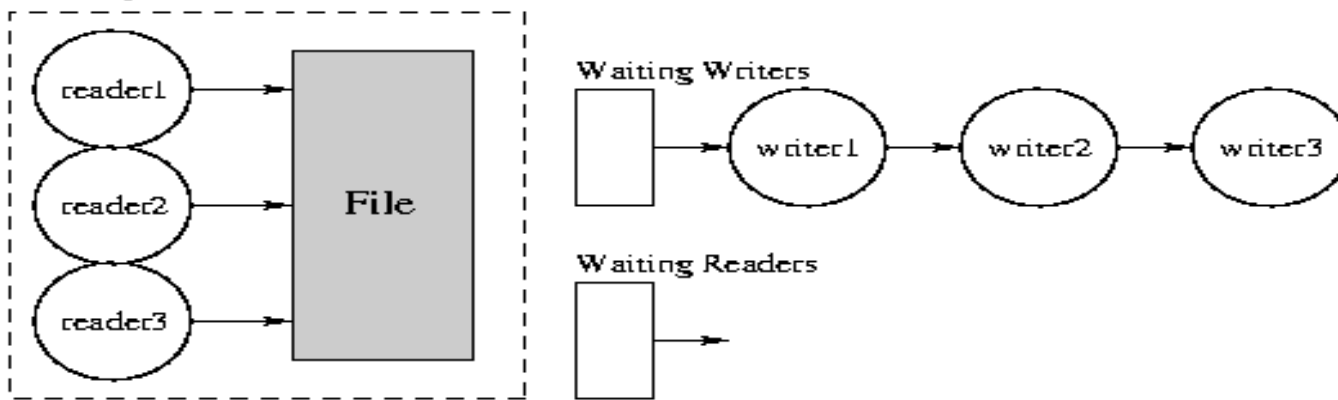
- **Leitores:** não requerem excluir uns aos outros (entre eles).
- **Escritores:** requerem excluir todos os outros (leitores e escritores).
- A escrita exige exclusão mútua de modo a garantir a consistência dos dados.



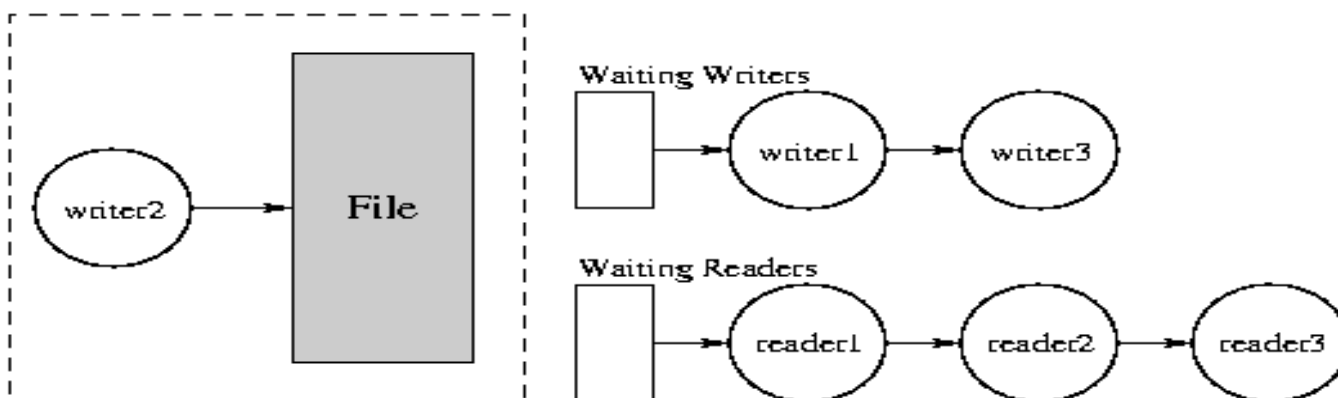
Leitores e escritores

- **Leitores:** não requerem excluir uns aos outros (entre eles).
- **Escritores:** requerem excluir todos os outros (leitores e escritores).
- A escrita exige exclusão mútua de modo a garantir a consistência dos dados.

Multiple Readers



One Writer



Leitores e escritores

- O Bloqueio no problema Leitor-Escritor é a solução mais utilizada nas seguintes situações:
 1. Em aplicações onde é fácil identificar que alguns processos somente lêem os dados compartilhados e que outros processos somente atualizam os dados compartilhados.
 2. Em aplicações que possuem mais leitores do que escritores. Isto é porque o bloqueio (*lock*) geralmente requer mais sobrecarga (*overhead*) para ser estabelecido do que semáforos e exclusão-mútua. O aumento da concorrência para permitir múltiplos leitores compensa a sobrecarga do sistema.

Leitores e escritores

Código exemplo

```
#define READERS  20
#define WRITERS  3

void reader() {
    while (1) {
        pthread_mutex_lock(&mutex);
        rc=rc+1;

        if(rc==1) pthread_mutex_lock(&db);
        pthread_mutex_unlock(&mutex);

        read_data_base();

        pthread_mutex_lock(&mutex);
        rc=rc-1;

        if(rc==0) pthread_mutex_unlock(&db);
        pthread_mutex_unlock(&mutex);

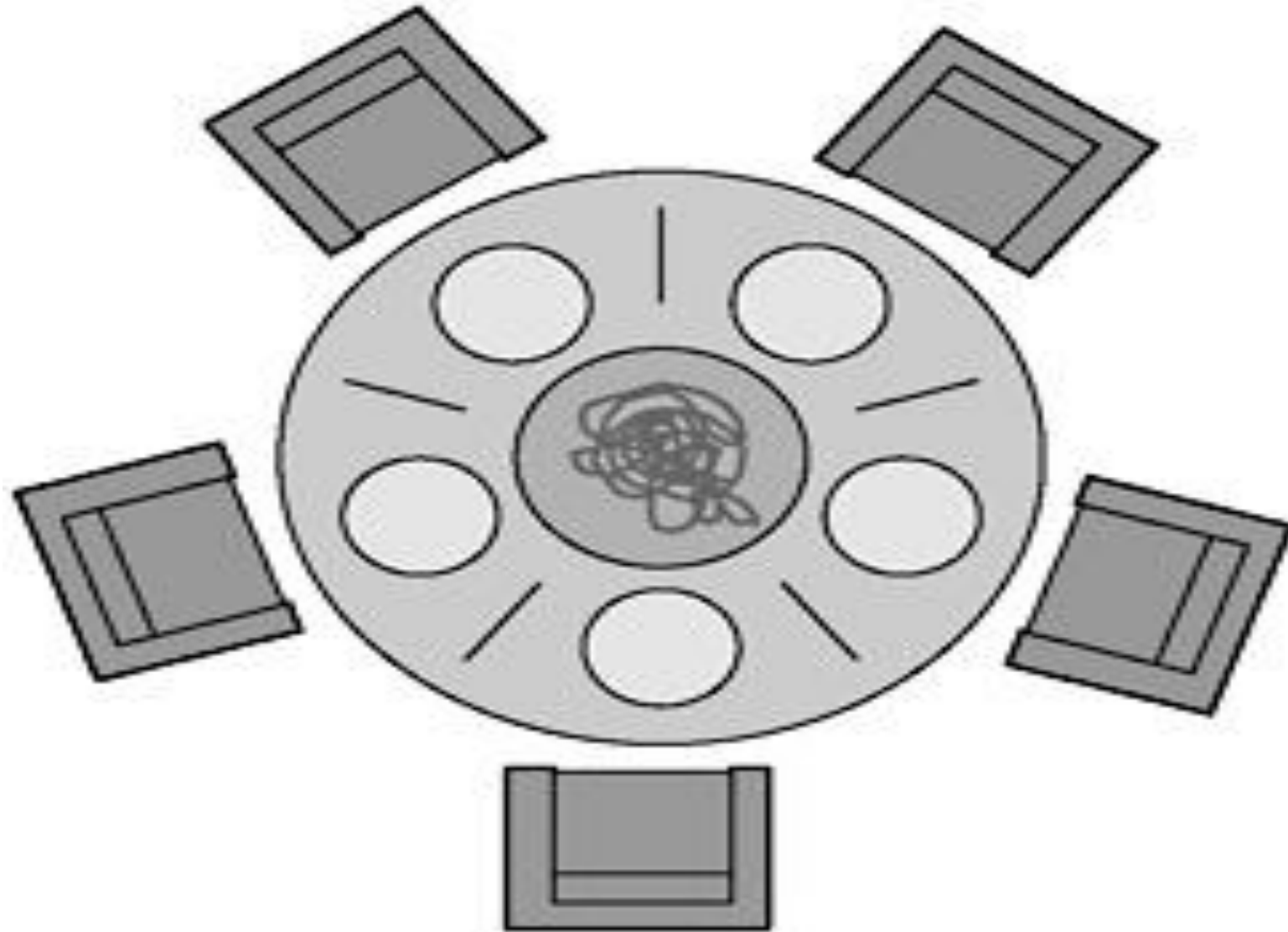
        use_data_read();
    }
}
```

```
void writer() {
    while(1) {
        think_up_data();
        pthread_mutex_lock(&db);
        write_data_base();
        pthread_mutex_unlock(&db); }
}
```

```
void think_up_data() {
    int thinktime;
    thinktime = rand() % 10;
    printf("Escritor pensando no que irá escrever\n");
    sleep(thinktime);
}
```

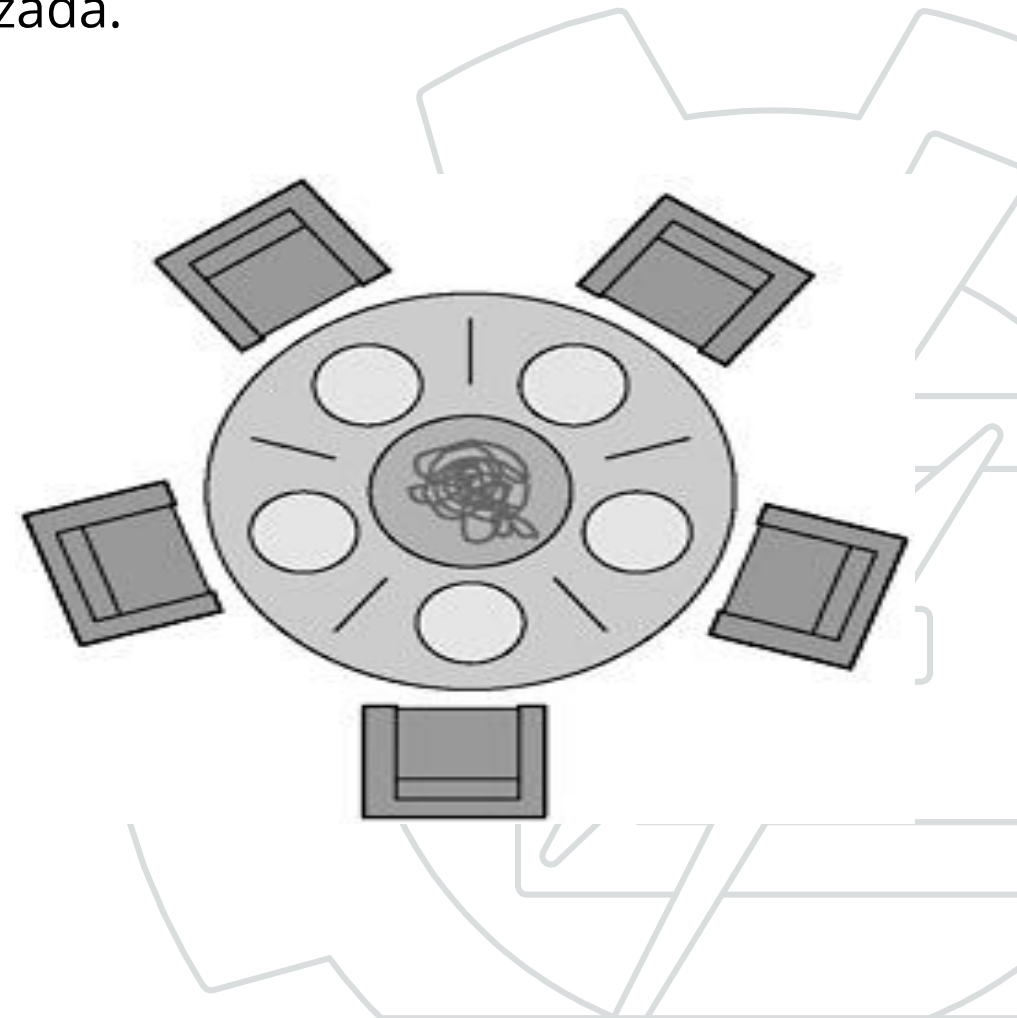
```
void write_data_base() {
    int writetime;
    writetime = rand() % 6;
    printf("Escritor escrevendo no banco de dados\n");
    sleep(writetime);
}
```

Jantar dos filósofos



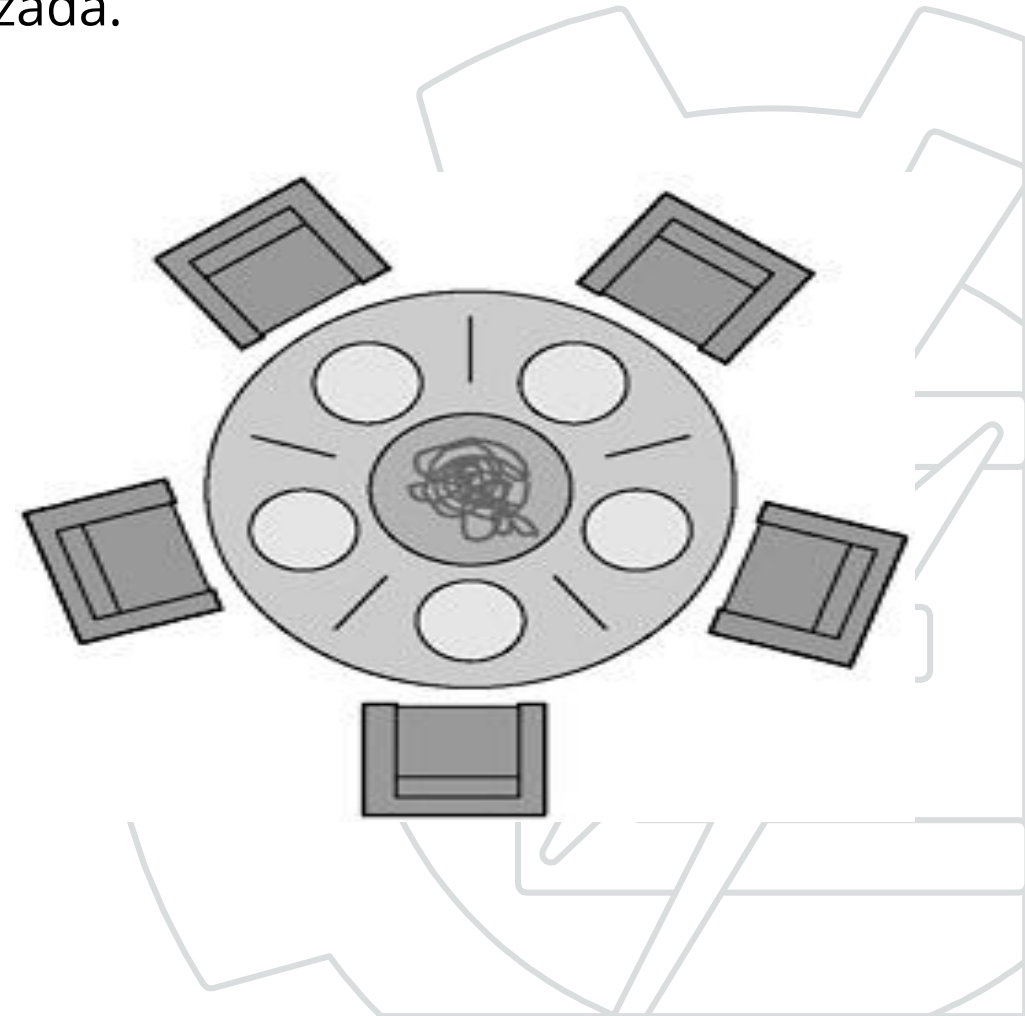
Jantar dos filósofos

- Cinco filósofos estão sentados ao redor de uma mesa circular para o jantar.
- Entre cada par de pratos existe apenas um *hashi*.
- *Hashis* precisam ser compartilhados de forma sincronizada.
- Cada filósofo possui um prato de macarrão.
- Cada filósofo precisa de 2 *hashis*.



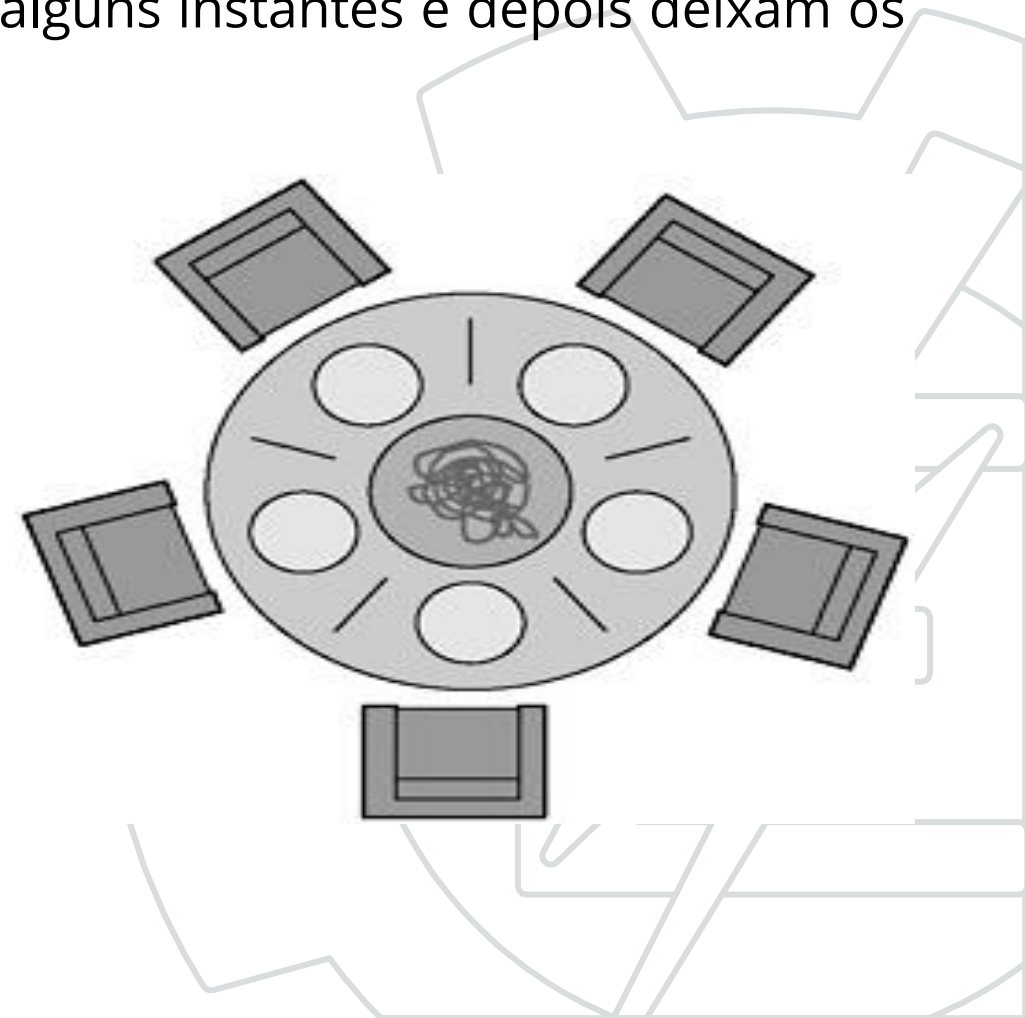
Jantar dos filósofos

- Cinco filósofos estão sentados ao redor de uma mesa circular para o jantar.
- Entre cada par de pratos existe apenas um *hashi*.
- *Hashis* precisam ser compartilhados de forma sincronizada.
- Cada filósofo possui um prato de macarrão.
- Cada filósofo precisa de 2 *hashis*.
- Utilização de recursos compartilhados.



Jantar dos filósofos

- Os filósofos comem e pensam alternadamente.
- Além disso, quando comem, pegam apenas um *hashi* por vez.
- Se conseguirem pegar os dois *hashis* eles comem por alguns instantes e depois deixam os *hashis* nos locais originais.

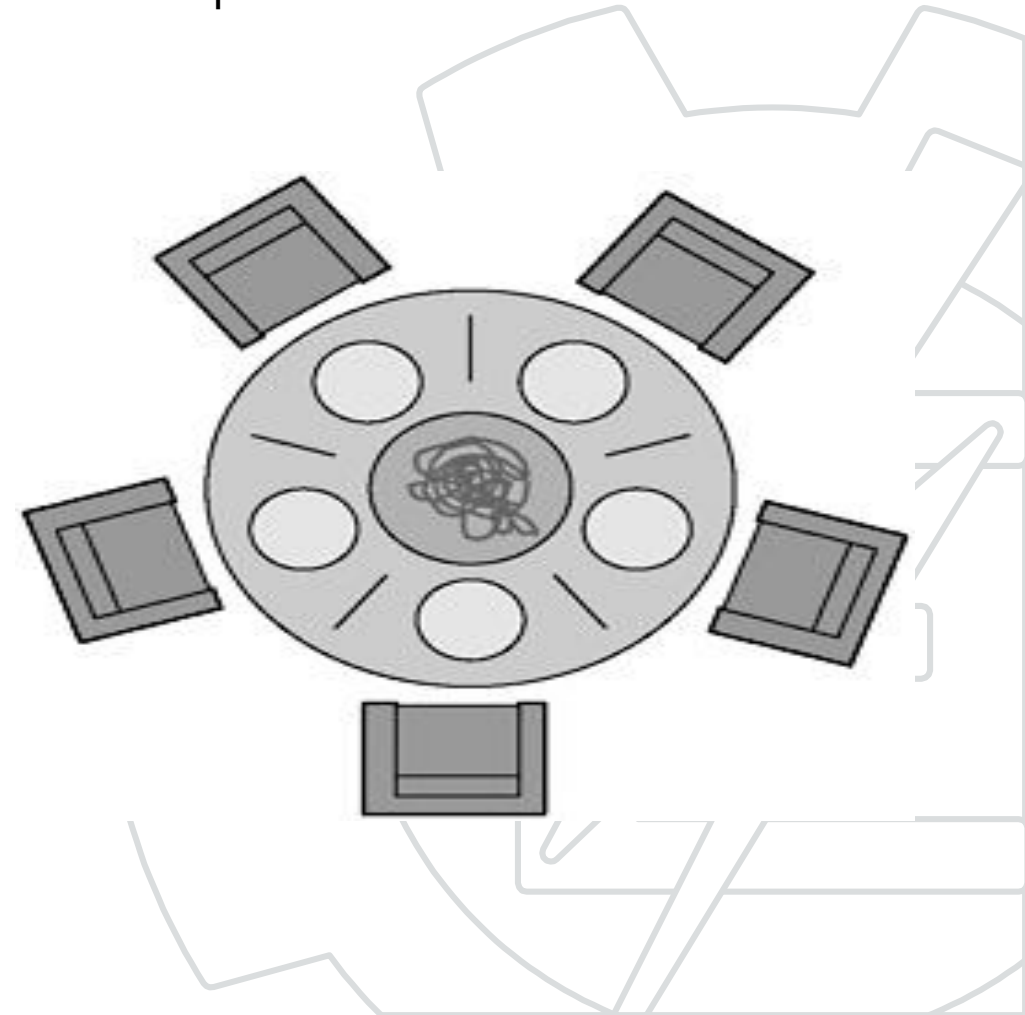


Jantar dos filósofos

- Foi proposto por Edsger W. Dijkstra (1965) como um problema de sincronização.
- Cada filósofo alterna entre duas tarefas: **comer** ou **pensar**.
- Cada um dos 5 filósofos pode ser modelado conforme o seguinte comportamento:

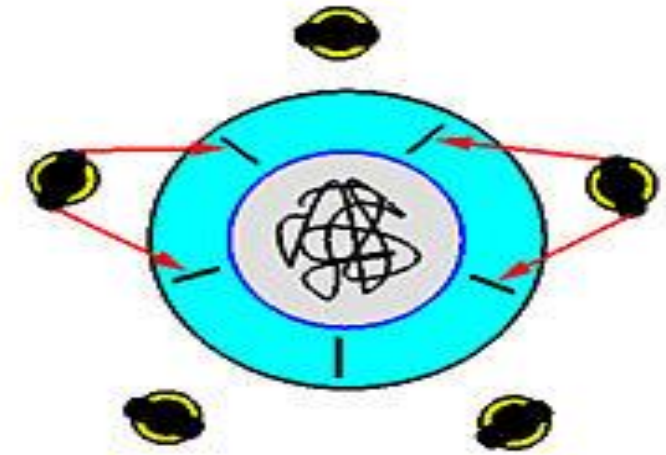
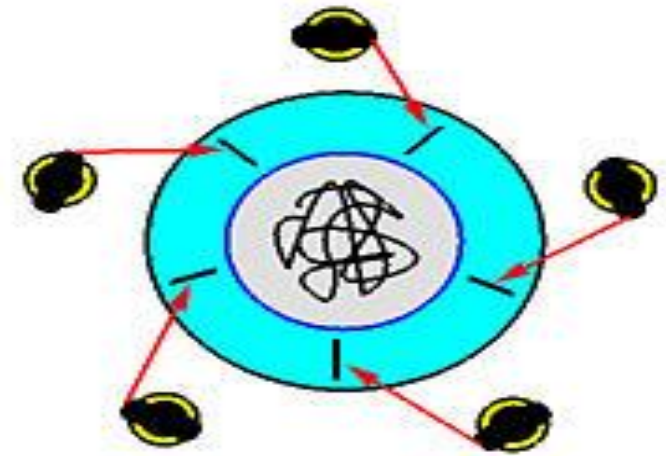
```
while (true){  
    meditar (duração aleatória)  
    pegar o palito à sua esquerda  
    pegar o palito à sua direita  
    comer (duração aleatória)  
    soltar o palito à sua esquerda  
    soltar o palito à sua direita  
}
```

- Impasse (*deadlock*) e inanição (*starvation*)



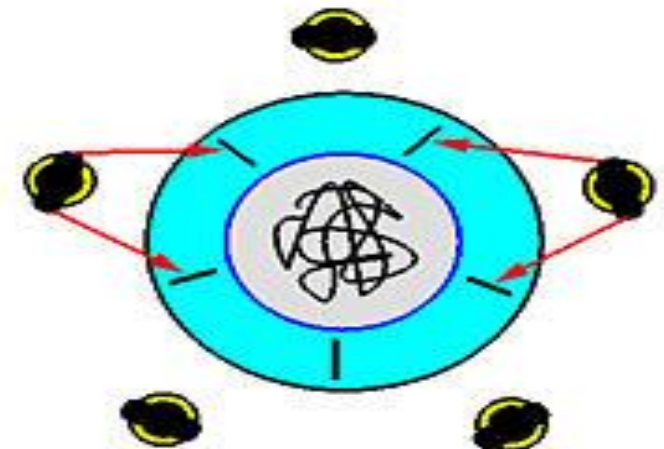
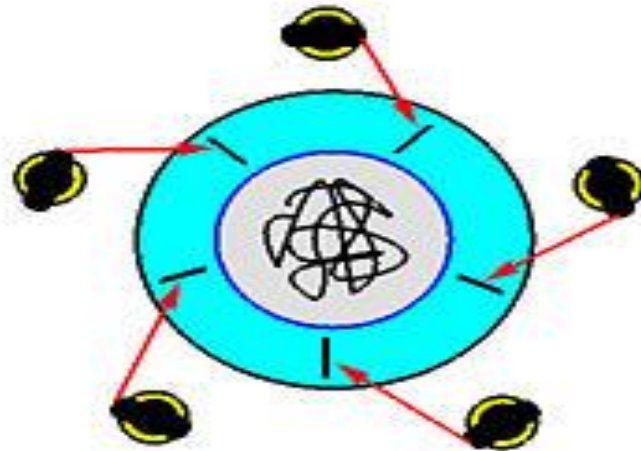
Jantar dos filósofos

- Proposta de solução – Deve seguir a sequência cíclica:
 - **Pensar** por um tempo Δt ;
 - Pegar os *hashis*:
 - Lock Esquerdo;
 - Lock Direito;
 - **Comer** por um tempo Δe ;
 - Deixar os *hashis*:
 - Unlock Esquerdo;
 - Unlock Direito.
- Como evitar que fiquem bloqueados?



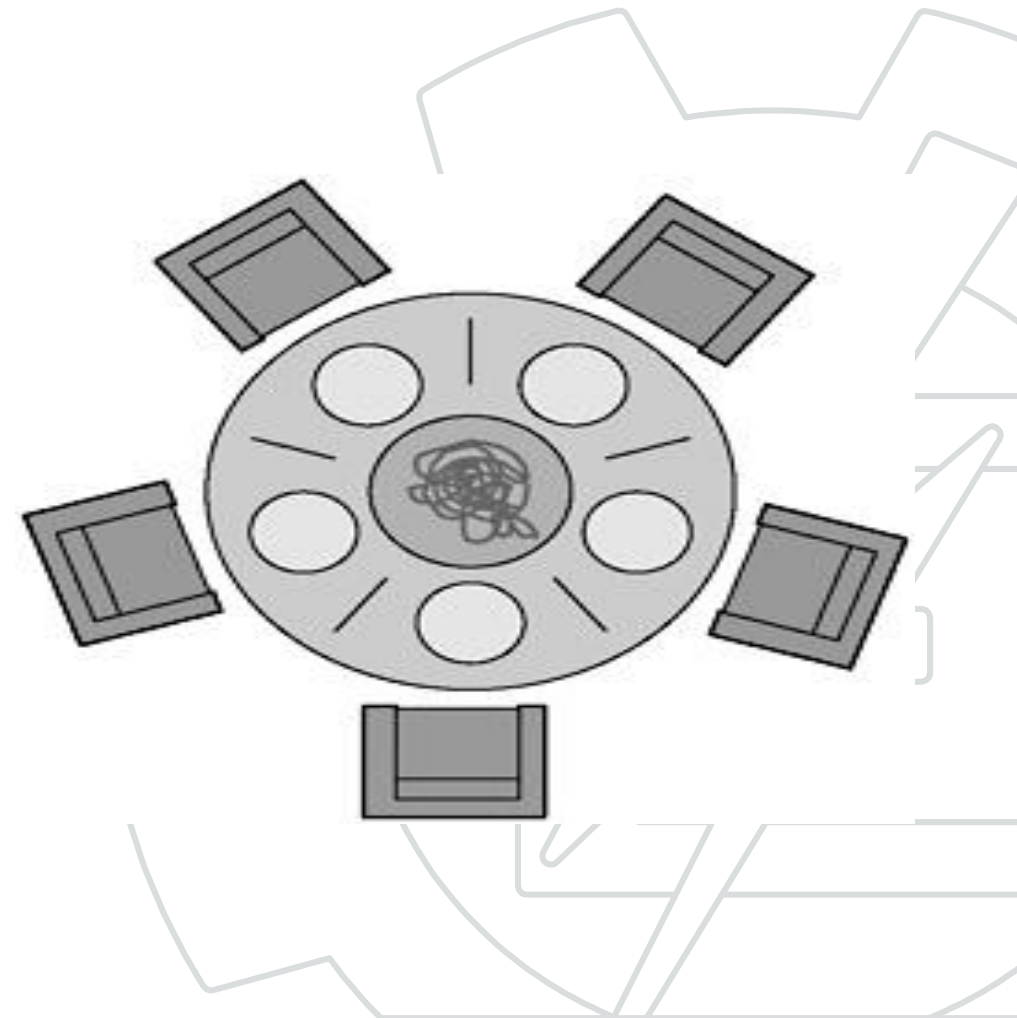
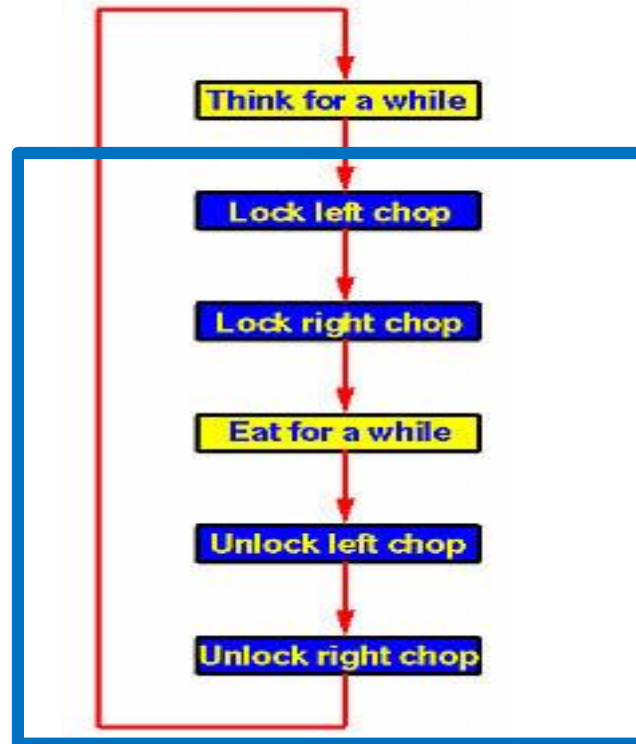
Jantar dos filósofos

- A proposta de solução anterior funciona bem?
 - Possíveis problemas:
 - **Deadlock** (ou Impasse): todos os filósofos pegam um único *hashi* ao mesmo tempo;
 - **Starvation** (ou Inanição): os filósofos ficam indefinidamente pegam os *hashis* simultaneamente; ou nunca conseguem pegar dois *hashis* ao mesmo tempo.



Jantar dos filósofos

- Como evitar múltiplas tentativas?
 - Semáforo binário (*up/down*) – Exclusão mútua
 - Problema: somente um filósofo come por vez



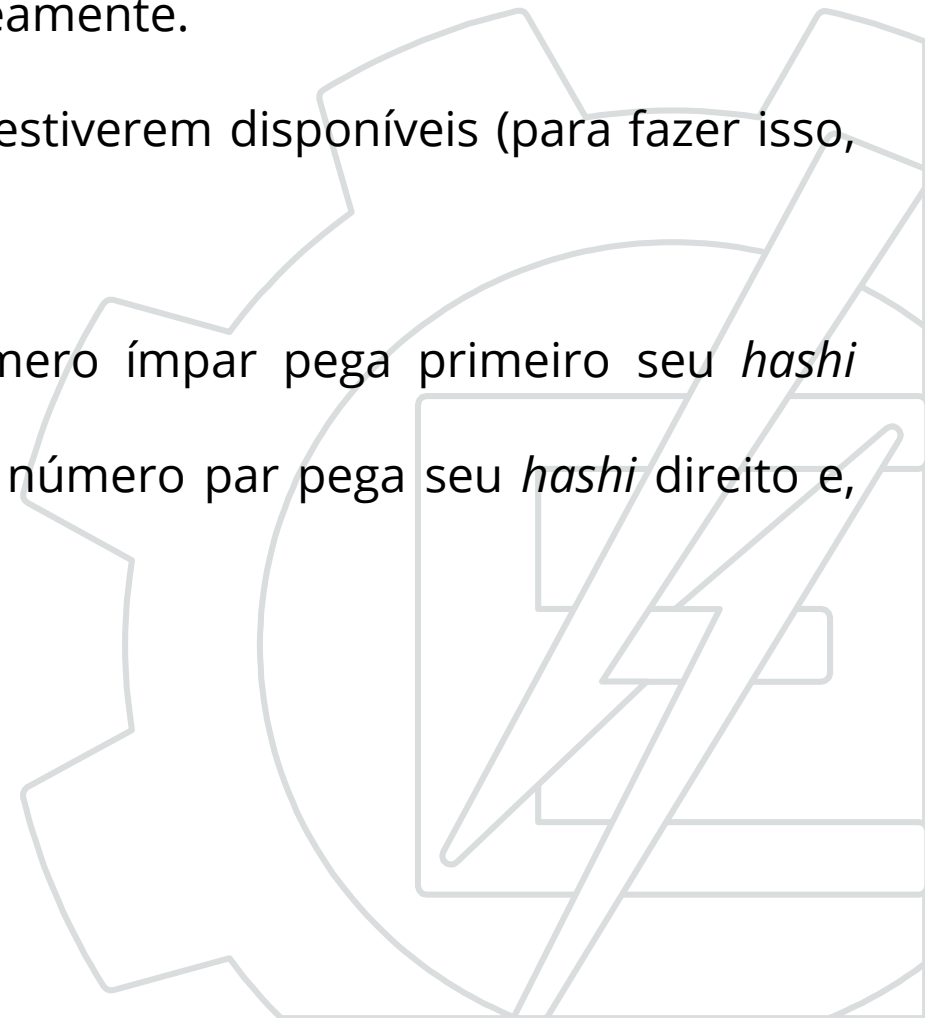
Jantar dos filósofos

- O problema do jantar dos filósofos é considerado um problema clássico de sincronização pois é um exemplo de problema de **controle de concorrência em larga escala**.
- É a simples representação da necessidade de alocação de diversos recursos entre diversos processos de modo a não causar *deadlock* ou *starvation*.



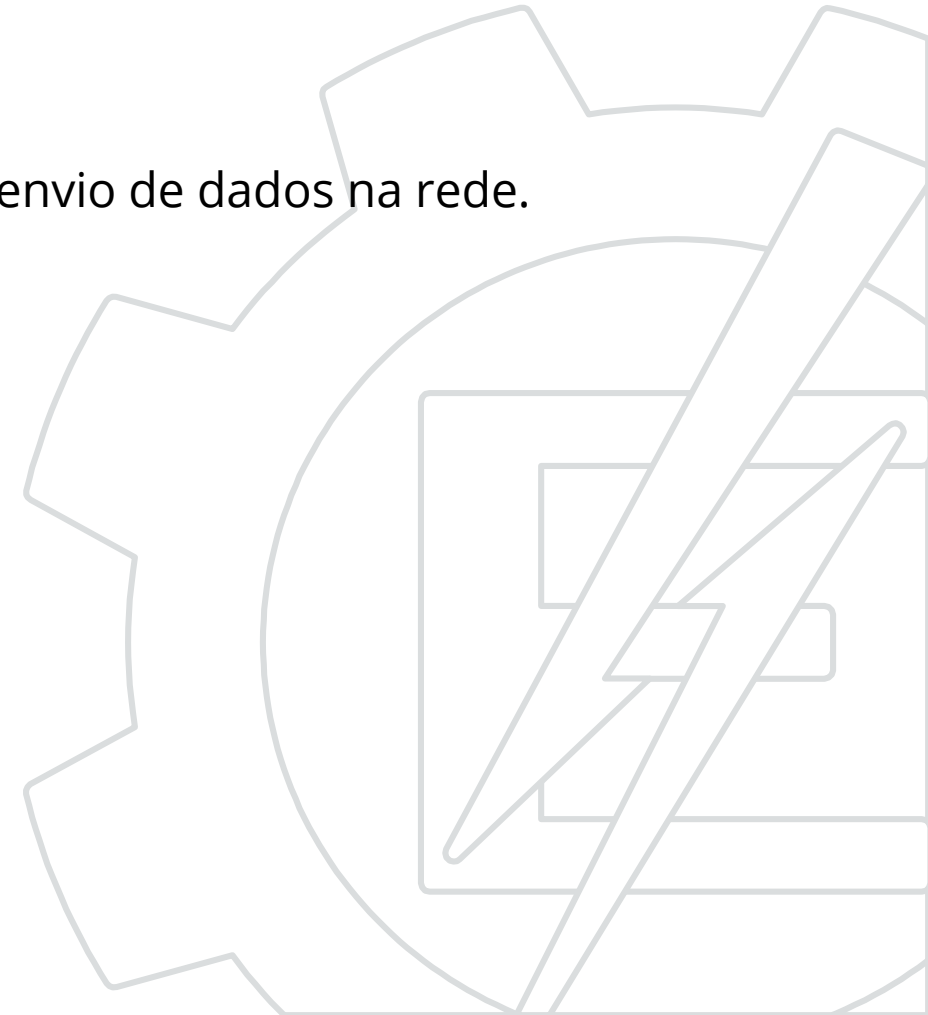
Jantar dos filósofos

- Várias soluções possíveis para o problema do *deadlock* podem ser substituídas por:
 - Permitir no máximo quatro filósofos sentados à mesa simultaneamente.
 - Permitir que um filósofo pegue seus *hashis* apenas se os dois estiverem disponíveis (para fazer isso, ele deve pegá-los em uma seção crítica).
 - Usar uma solução assimétrica — isto é, um filósofo de número ímpar pega primeiro seu *hashi* esquerdo e, então, seu *hashi* direito, enquanto um filósofo de número par pega seu *hashi* direito e, então, seu *hashi* esquerdo.



Jantar dos filósofos

- Este problema é útil para modelar processos que **competem por acesso exclusivo** a um **número limitado de recursos**:
 - Periféricos em geral, particularmente os não-preemptivos;
 - O protocolo *Ethernet* utiliza parte desta solução para modelar o envio de dados na rede.
- Ajuda a modelar uma solução livre de ***starvation*** e ***deadlocks***.



Jantar dos filósofos

Código exemplo

```
// The existence of a Philosopher
void exist () {
    while (true) {
        think();
        take_fork();
        take_fork();
        eat();
        put_fork();
        put_fork();
    }
}
```

```
void take_fork (){
    if (n_forks == 0) {
        if (!table.get_one_fork(philosopher_id)) {
            return;
        }
    } else if (n_forks == 1) {
        if (!table.get_one_fork(philosopher_id)) {
            put_fork();
            sleep_for(milliseconds(5000));
        }
    }
    n_forks++;
}
```

```
void take_fork (){
    if (n_forks == 0) {
        if (!table.get_one_fork(philosopher_id)) {
            return;
        }
    } else if (n_forks == 1) {
        if (!table.get_one_fork(philosopher_id)) {
            put_fork();
            sleep_for(milliseconds(rand() % 10000));
        }
    }
    n_forks++;
}
```

O filósofo poderá morrer de fome. Ele não ficará travado, mas irá ficar tentando fazer a mesma operação de pegar e depois devolver o garfo pra sempre.

Com tempos aleatórios de espera para tentar pegar novamente um garfo a probabilidade de *starvation* é menor.

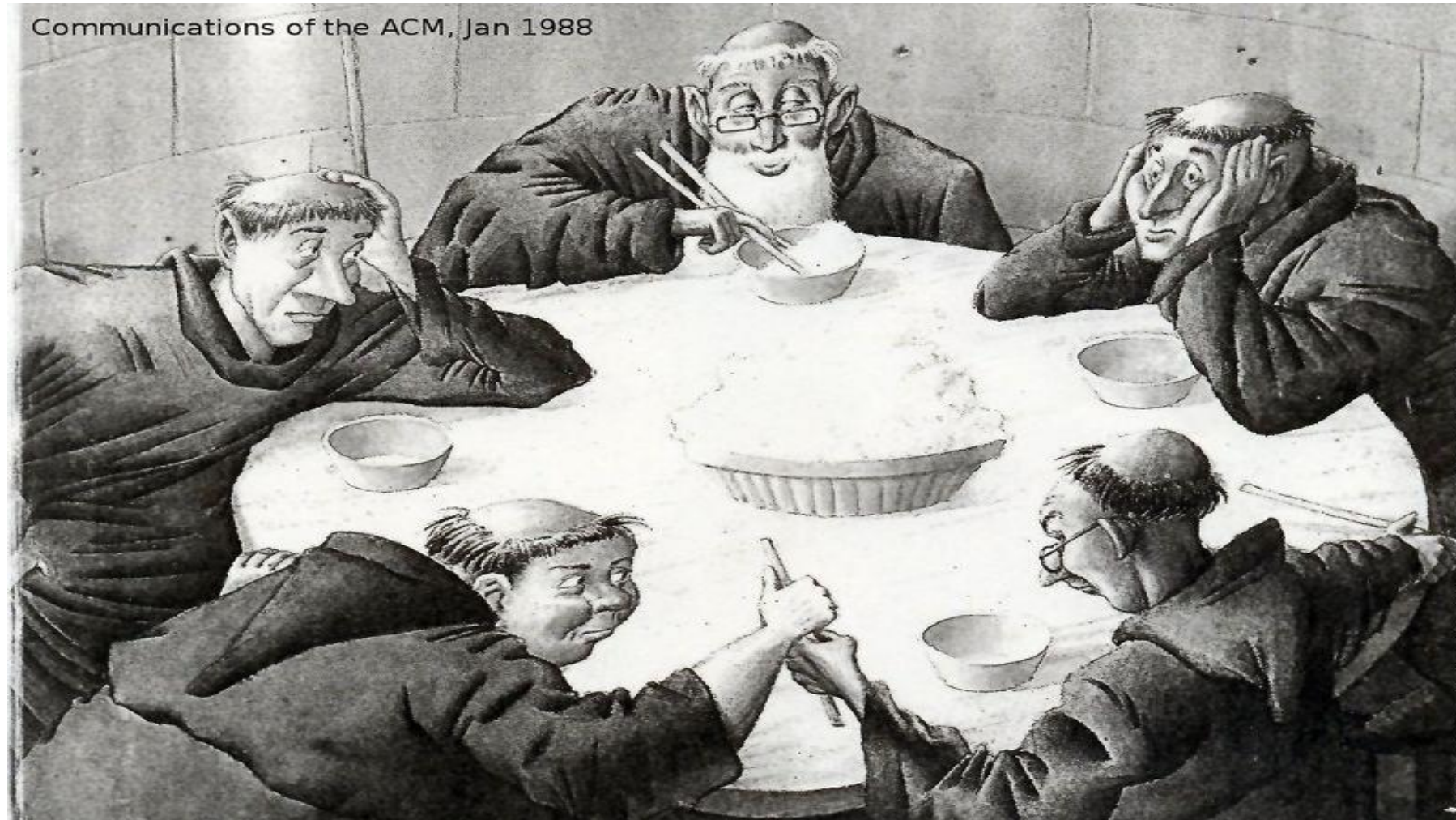
No entanto, não é impossível!

Impasse (*Deadlock*)

Um conjunto de processos bloqueados, cada um de posse de um recurso e esperando por outro, já obtido por algum outro processo no conjunto.

Condições necessárias:

- Exclusão mútua
- Posse durante a espera
- Inexistência de preempção
- Espera circular



Bibliografia

- TANENBAUM, Andrew S; BOS, Herbert. Sistemas operacionais modernos. 4a ed. São Paulo: Pearson Education do Brasil, 2016.

Capítulo 2.

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/1233>

- DEITEL, H.M; DEITEL, P.J; CHOFFNES,D.R. Sistemas Operacionais. 3a ed. São Paulo: Pearson Prentice Hall, 2005. **Capítulos 5 e 6.**

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/315>



Sistemas Operacionais

Prof. Otávio Gomes

otavio.gomes@unifei.edu.br

