# Outros mecanismos de sincronização e comunicação

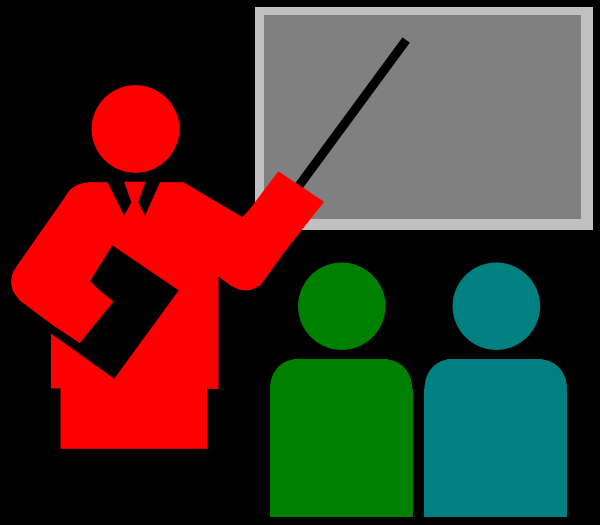
**Volnys Borges Bernal**
**volnys.bernal@usp.br**

**Depto. de Eng. de Sistemas Eletrônicos**
**Escola Politécnica da USP**

# Sumário

- Mecanismos de sincronização e comunicação
  - Barreira
  - Monitor
  - Troca de mensagens

# Métodos de sincronização

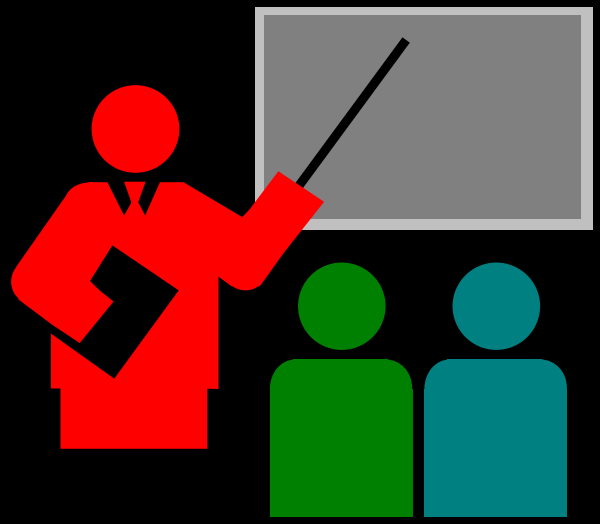


# Métodos de sincronização

- **Métodos e usos típicos**

| Método de sincronização | Principal uso |
|---|---|
| Mutex | Sincronização para exclusão mútua |
| Variável de condição | Sincronização para gestão de recursos |
| Semáforo | Sincronização para gestão de recursos |
| Barreira | Sincronização para cooperação |
| Monitor | Sincronização para gestão de recursos |
| Troca de mensagens | Sincronização e comunicação |

# Barreira

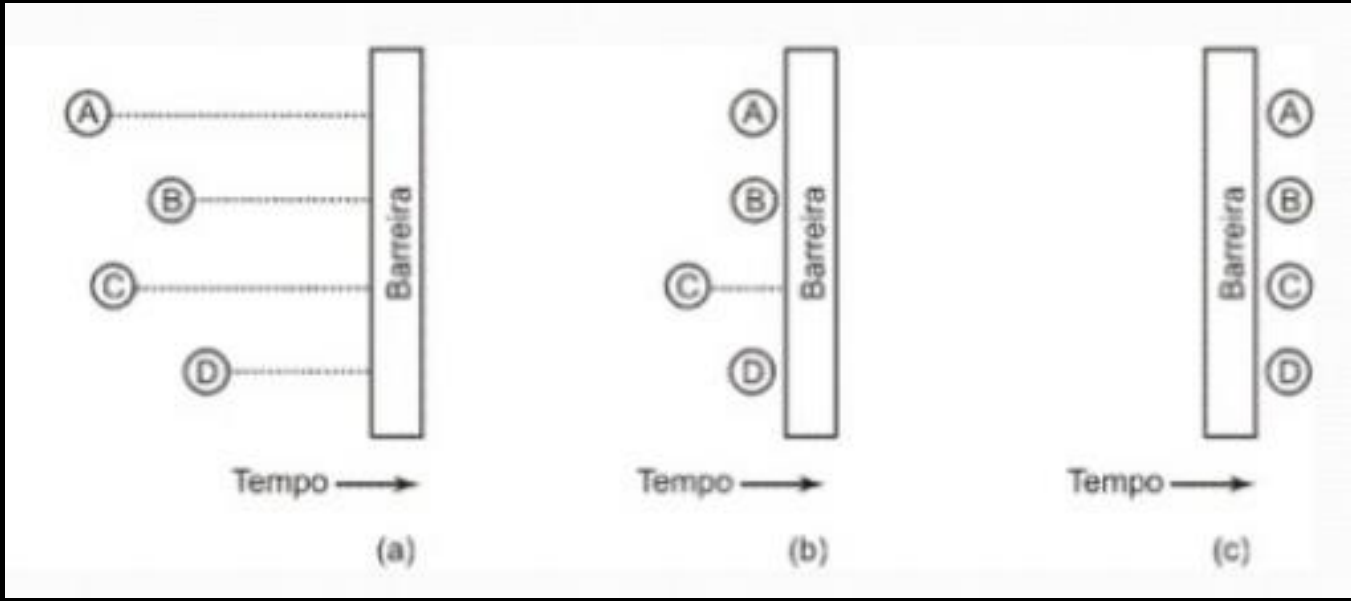


# Barreira

- Método de sincronização voltado à cooperação entre entidades de processamento
- Permite a criação de uma “barreira” de sincronização, na qual todos os membros de um grupo de threads precisa chegar e esperar pela chegada de todos para poder prosseguir.
- Muito utilizado no processamento numérico de alto desempenho.

# Barreira

- Exemplo do funcionamento da barreira
  a) Entidades se aproximam da barreira
  b) Entidades que chegam à barreira esperam pelas outras
  c) Quando todas entidades chegam à barreira, todos podem continuar



# Barreira

- **Interface Pthreads para barreira**
  - **`Tipo barreira`**

| Tipo | Descrição |
|---|---|
| `pthread_barrier_t` | Representa o tipo de um objeto barreira. |

  - **`Primitivas para barreira`**

| Primitiva | Descrição |
|---|---|
| `pthread_barrier_init` | Iniciação dinâmica do objeto barreira. |
| `pthread_barrier_wait` | Espera um determinado número de entidades chegar na barreira. Quando todos chegam, a espera termina. |
| `pthread_barrier_destroy` | Destrói uma barreira. |

# Barreira

- Sintaxe Pthreads para barreira

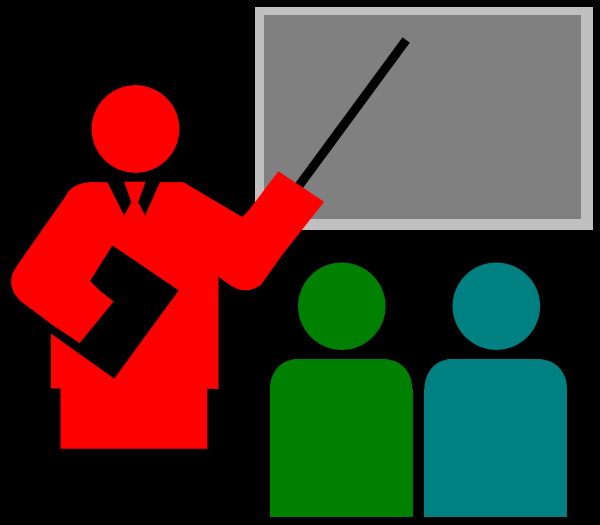
```
#include <pthread.h>

int pthread_barrier_init(pthread_barrier_t * barrier,
                         const pthread_barrierattr_t * attr,
                         unsigned int count);
int pthread_barrier_wait(pthread_barrier_t *barrier);
int pthread_barrier_destroy(pthread_barrier_t *barrier);
```

# Barreira

- **Exemplo de uso de barreira Pthread:**

```
#include <pthread.h>

...
pthread_barrier_t * mybarrier;
...
status = pthread_barrier_init(&mybarrier,0,4);
...
status = pthread_barrier_wait(&mybarrier);
...
status = pthread_barrier_destroy(&mybarrier);
...
```

# Monitor



# Monitor

- **Método de sincronização de alto nível**
- **Suportado por uma linguagem de programação**
- **Monitor é um módulo (ou pacote)**

  - **Conceito de módulo:**
    - Agrupamento de procedimentos (funções) e dados
    - Caixa preta: os dados internos (variáveis e estruturas) não são visíveis e acessíveis fora do módulo.
    - Os dados internos são acessíveis somente pelos procedimentos (funções) do módulo.

# Monitor

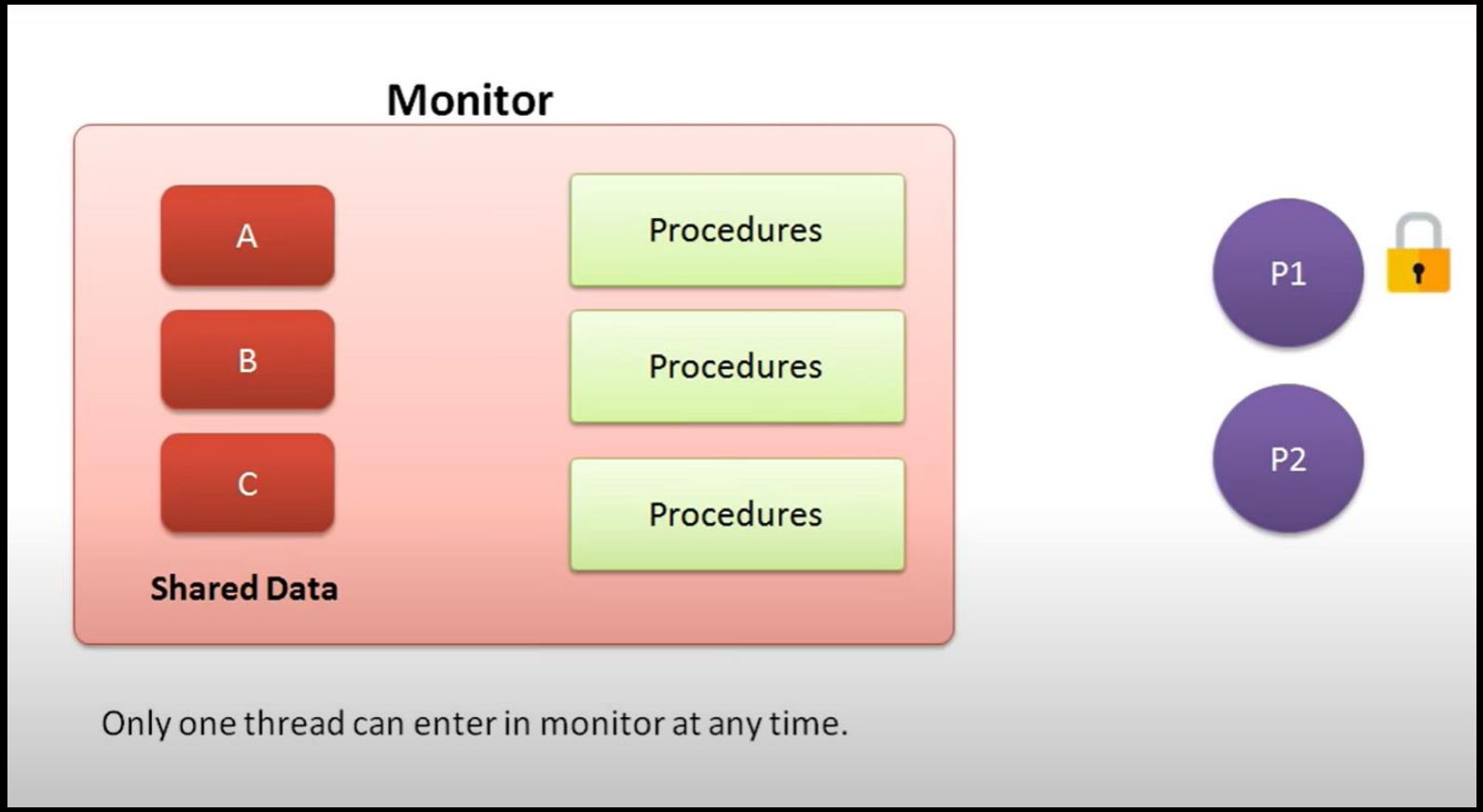
- **Sincronização**
  - **Exclusão mútua da execução dos procedimentos de um monitor**
    - Somente uma entidade (processo ou thread) por vez pode executar um procedimento (função) de um monitor
    - Cada Monitor possui um mutex implícito.
      - Como se existisse um lock na entrada e um unlock na saída de cada função do monitor (incluindo sua função de iniciação).
      - Como o conceito de monitor é suportado por uma linguagem de programação, cabe ao compilador incluir as primitivas de exclusão mútua no momento da geração do código.
  - **Espera por recursos**
    - Primitivas wait e signal
    - Permite a espera por recursos

# Monitor (exemplo)

```
monitor <nome do monitor>

    <declaração de variáveis compartilhadas>

    procedure P1(…)
        . . .
        end

    . . .

    procedure Pn(…)
        . . .
        end

    init
        <código de inicialização>
        end

    end monitor.
```

# Monitor

- Conceito



# Monitor

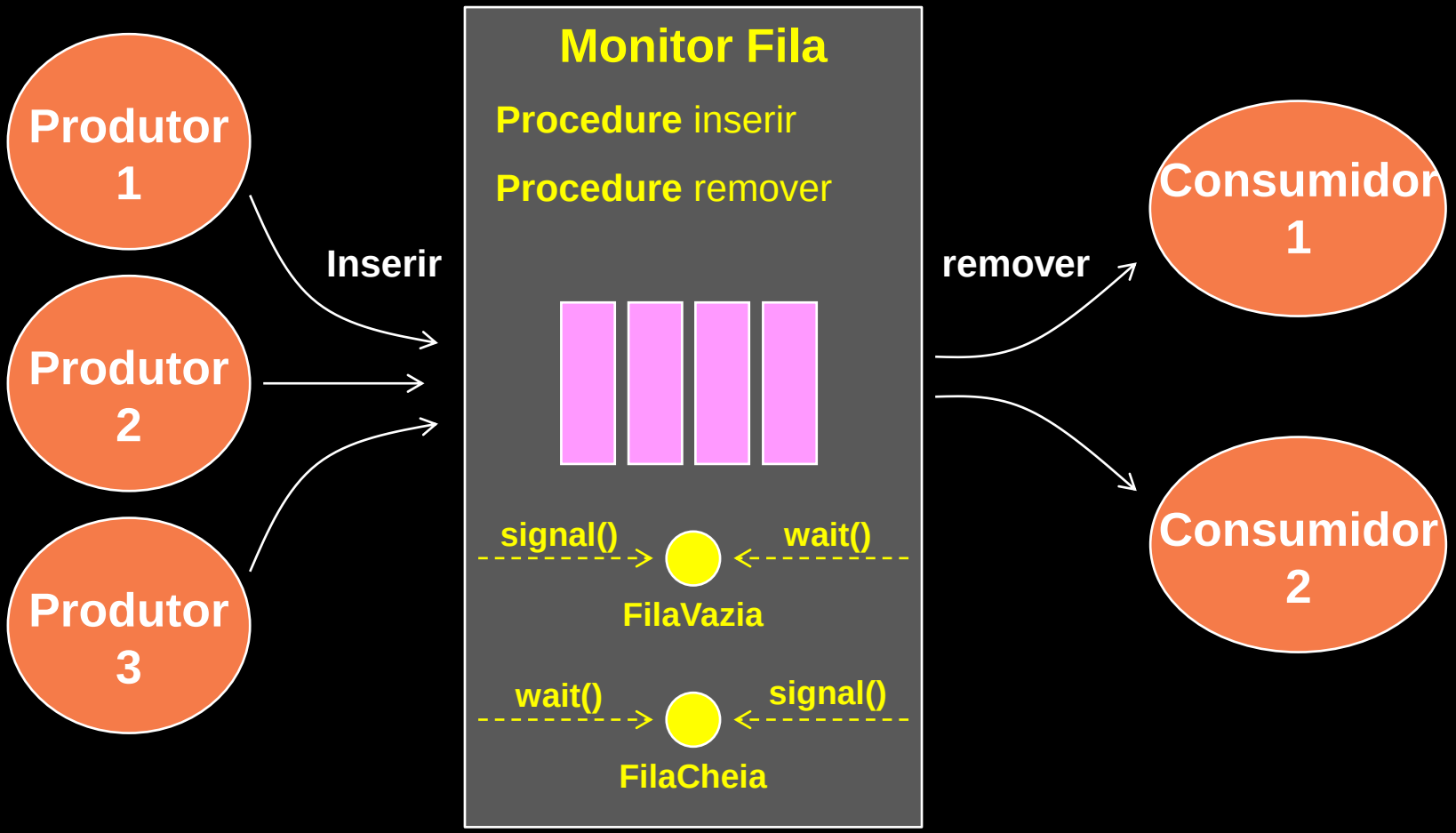
- Solução do problema produtor-consumidor

```
Produtor
    Repetir
        Produzir(E);
        Fila.inserir(E);

Consumidor
    Repetir
        E = Fila.remover();
        Processar(E);

Monitor Fila
    cond FilaCheia
    cond FilaVazia
    int  c
```

```
procedure inserir(item)
        se c == N então
            wait(FilaCheia)
        inserir(item)
        c = c + 1
        se c == 1 então
            signal(FilaVazia)

procedure remover(item)
        se c == 0 então
            wait(FilaVazia)
        item = remover()
        c = c -1
        se c == N-1 então
            signal(FilaCheia)
```

# Monitor

- Solução do problema produtor-consumidor

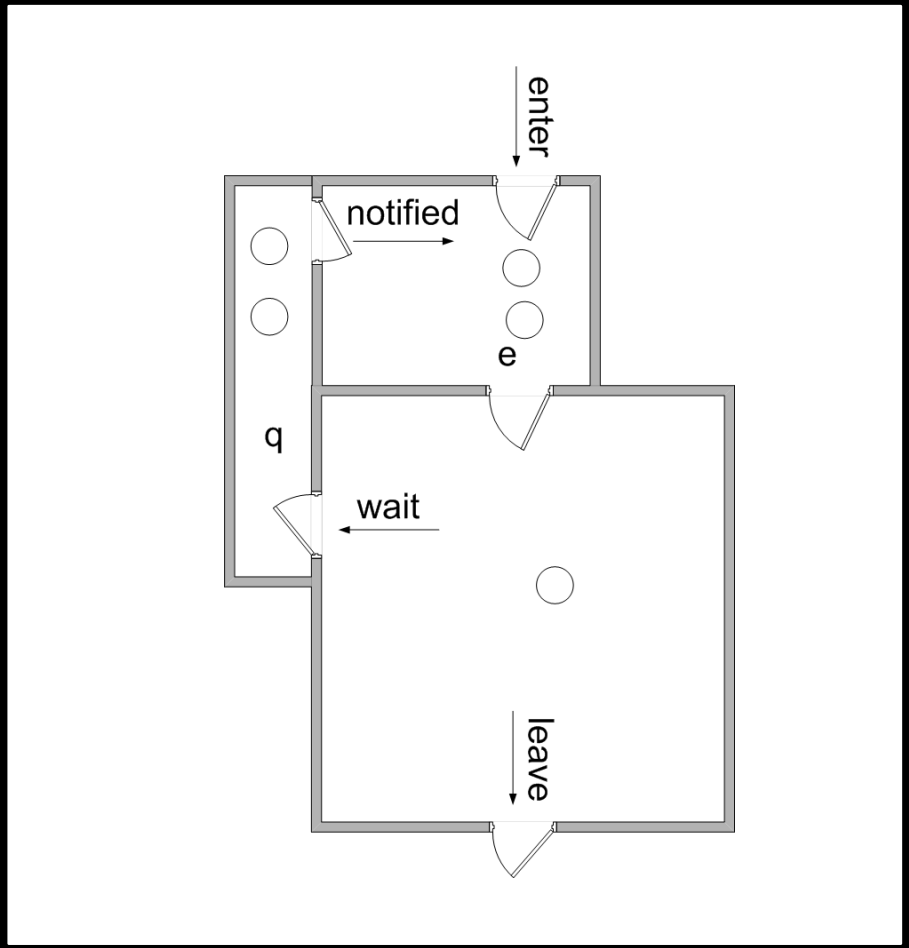


# Monitor

- **Linguagem Java**
  - **Linguagem orientada a objetos**
  - **Suporta threads de usuário**
  - **Permite que métodos (procedimentos) sejam agrupados em classes (definições de objetos)**
  - **Suporta o conceito de monitor:**
    - Exclusão mútua: Método sincronizado
      - Quando é adicionada a palavra-chave Synchronized à sua declaração
      - Somente um thread por vez pode executar métodos sincronizados de uma determinada classe
    - Espera por recursos
      - Primitivas wait e notify (similares a wait e signal)

# Monitor

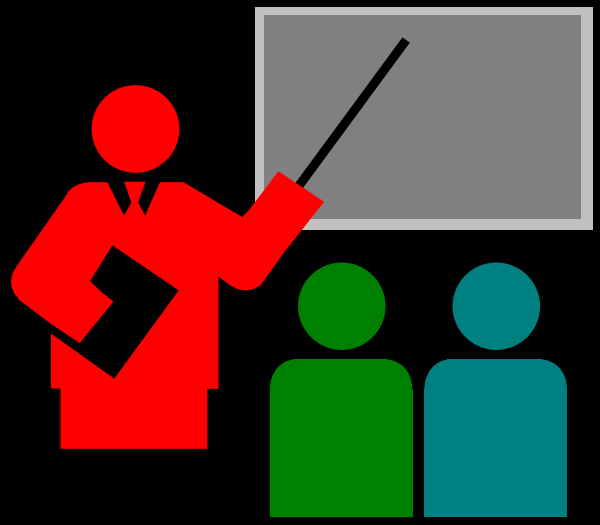
- Espera por recursos
  - Primitivas wait & notify



# Monitor

- **Solução do problema produtor-consumidor com java**
  - **Threads:**
    - Produtores
    - Consumidores
  - **Monitor Fila:**
    - Classe fila
    - Métodos sincronizados insert e remove
  - **Exemplo de implementação:**
    - Mostrada no livro do Tanenbaum de Sistemas Operacionais

# Troca de mensagem



# Troca de mensagem

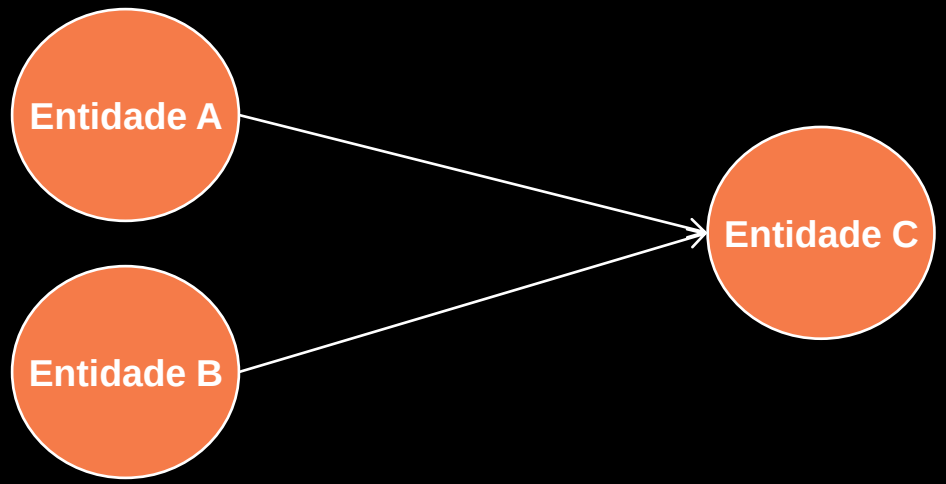
- Mecanismo muito utilizado para sincronização e comunicação entre processos
- Primitivas
  - Send(destinatário, mensagme)
  - Receive(remetente, mensagem)

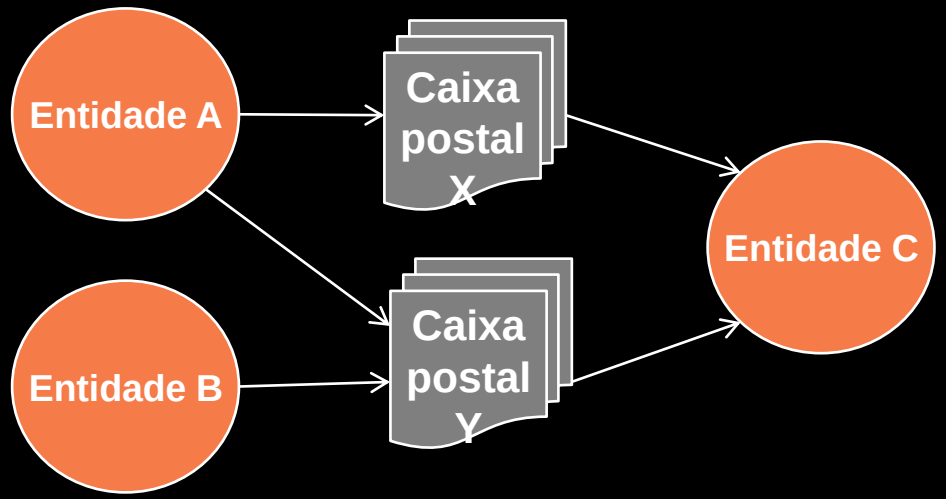
# Troca de mensagem

- **Tipos de primitivas**
  - **Síncrona**
    - Send() bloqueante
      - Entidade é bloqueada até a entidade parceira ativar o Receive()
    - Receive() bloqueante
      - Entidade é bloqueado até a entidade parceira ativar Send()
    - Não necessita de buffers temporários
  - **Assíncrona**
    - Send() não é bloqueante
    - Receive() é bloqueante
    - Necessita de gerenciamento de buffers temporários

# Troca de mensagem

- **Endereçamento**
  - **Baseada no endereço de cada entidade (thread ou processo)**
    - Síncrono ou assíncrono
  - **Intermediada por caixa postal**
    - Tipicamente assíncrono



# Troca de mensagens

- Solução do problema produtor-consumidor

```
#define N 100                                   /* número de lugares no buffer */

void producer(void)
{
     int item;
     message m;                                 /* buffer de mensagens */

     while (TRUE) {
          item = produce_item( );               /* gera alguma coisa para colocar no buffer */
          receive(consumer, &m);                /* espera que uma mensagem vazia chegue */
          build_message(&m, item);              /* monta uma mensagem para enviar */
          send(consumer, &m);                   /* envia item para consumidor */
     }
}

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

     for (i = 0; i < N; i++) send(producer, &m);  /* envia N mensagens vazias */
     while (TRUE) {
          receive(producer, &m);                /* pega mensagem contendo item */
          item = extract_item(&m);              /* extrai o item da mensagem */
          send(producer, &m);                   /* envia a mensagem vazia como resposta */
          consume_item(item);                   /* faz alguma coisa com o item */
     }
}
```