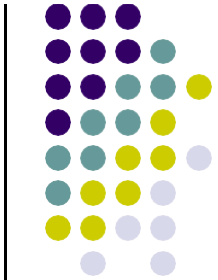
# Comunicação entre Processos

## Troca de Mensagens

# Troca de Mensagens

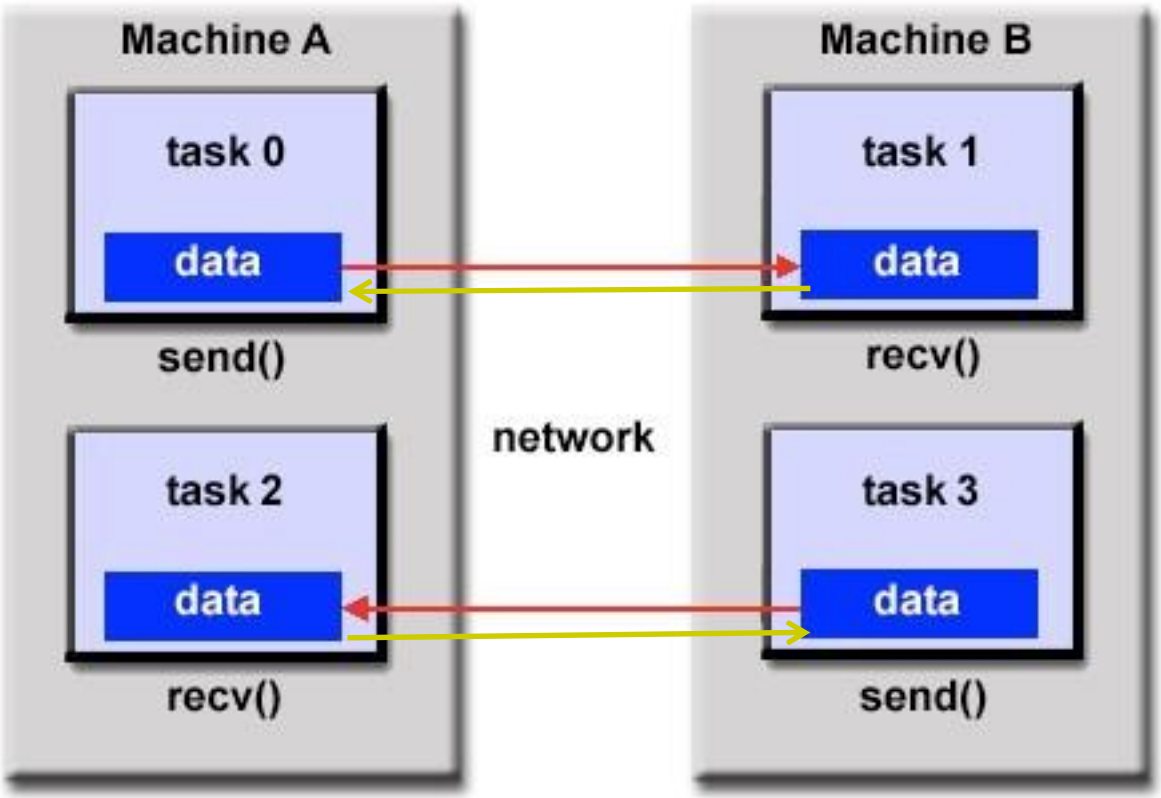
- É uma forma natural de interação entre processos
- Duas primitivas:
  - **send** (destino, *&message*) – tamanho da mensagem fixo ou variável
  - **receive** (fonte, *&message*) - fonte pode ser ANY
- Requer que processos se conheçam mutuamente
  - definem um enlace lógico de comunicação
  - Mensagens são tipadas e possuem header e dados
- Um enlace pode ser:
  - Com processos co-localizados ou remotos
  - confiável (controle de fluxo, erro e congestionamento) ou não-confiável
  - Ponto-a-ponto ou Ponto-a-multiponto
  - Envolve elementos físicos (e.g., memória compartilhada, barramento, rede)

# Troca de Mensagens



- Troca de mensagens é um mecanismo de sincronização mais genérico, porque:
  - Permite também a troca de dados
  - É independente se processos compartilham memória ou não.
  - Qualquer solução baseada em semáforos pode ser resolvida por envio de mensagens (considere: Down ≅receive, Up ≅ send, valor do semáforo = número de mensagens)
- Decisões de projeto do mecanismo:
  - Com ou sem bufferização (send assíncrono ou *Rendezvous*)
  - Quando a comunicação é remota (pela rede) mensagens podem ser perdidas:  são necessárias confirmações, timeouts e re-transmissões
  - Na versão *Rendezvous* podem ocorrer impasses

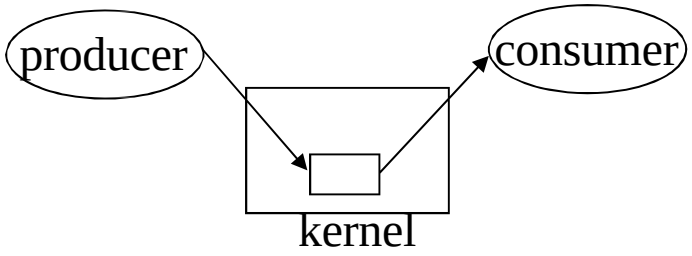
# Troca de Mensagens



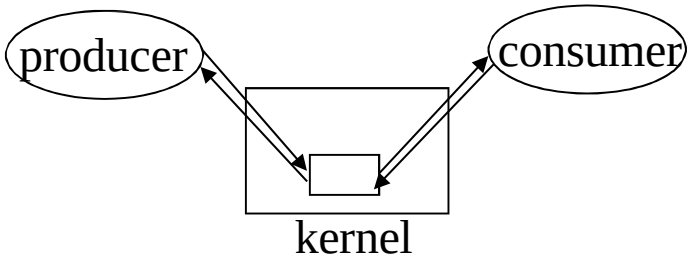
# Tipos de Envios de Mensagem

## Entre processos co-localizados

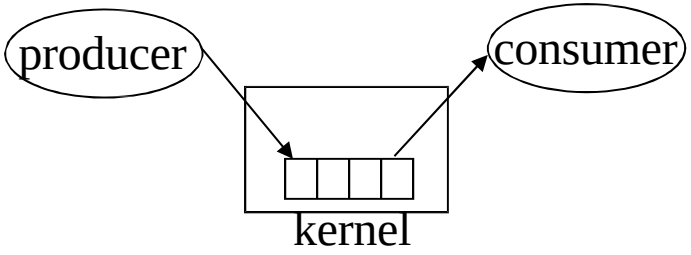
### Rendezvous (bloqueante)



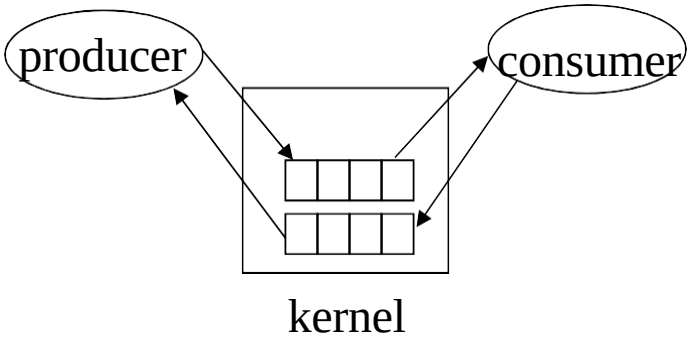

### Request-Reply síncrono




### Não-bloqueante




### Request-Reply, não-bloqueante



# Problema do Produtor-Consumidor com envio de *N* Mensagens

Idéia: consumidor envia mensagem vazia, e produtor responde com a mensagem preenchida.

```
#define N 100                                   /* number of slots in the buffer */

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

     while (TRUE) {
          item = produce_item( );               /* generate something to put in buffer */
          receive(consumer, &m);                /* wait for an empty to arrive */
          build_message(&m, item);              /* construct a message to send */
          send(consumer, &m);                   /* send item to consumer */
     }
}

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

     for (i = 0; i < N; i++) send(producer, &m);  /* send N empties */
     while (TRUE) {
          receive(producer, &m);                /* get message containing item */
          item = extract_item(&m);              /* extract item from message */
          send(producer, &m);                   /* send back empty reply */
          consume_item(item);                   /* do something with the item */
     }
}
```

# Perguntas?