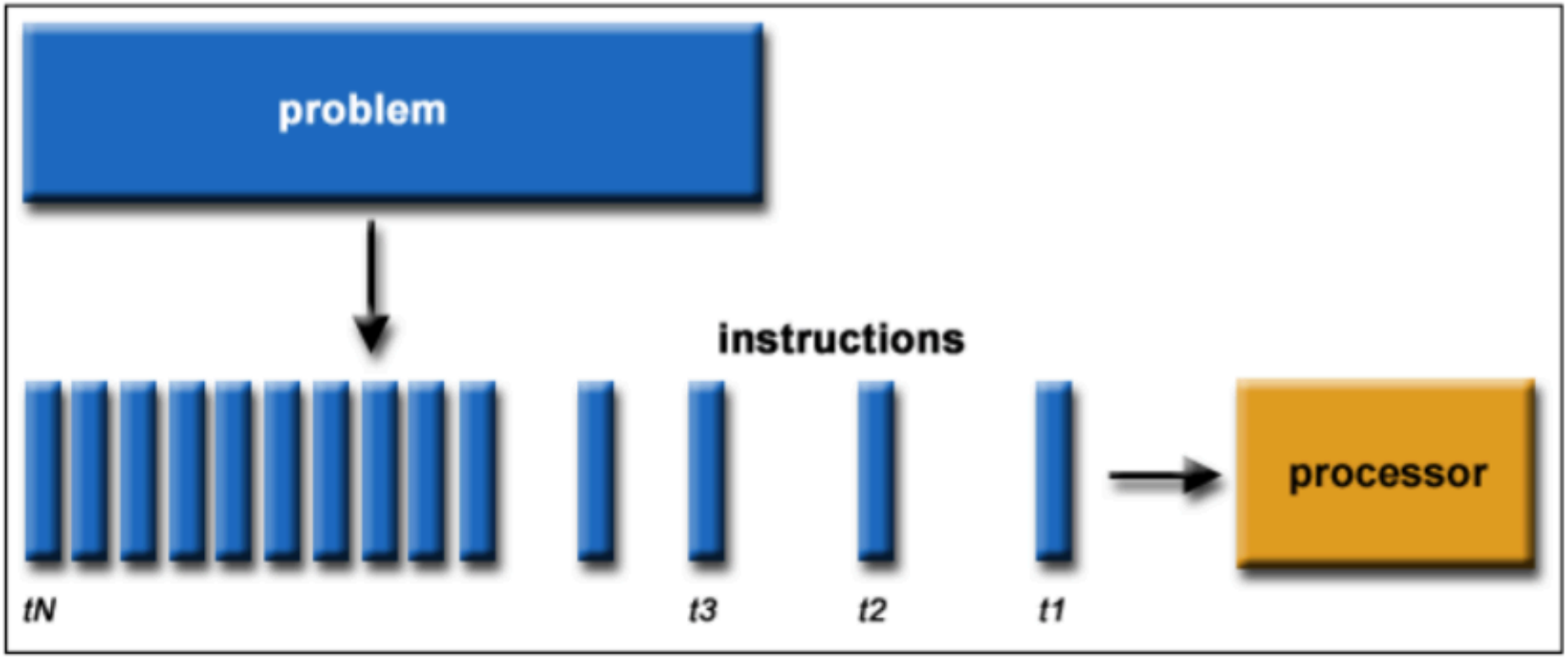
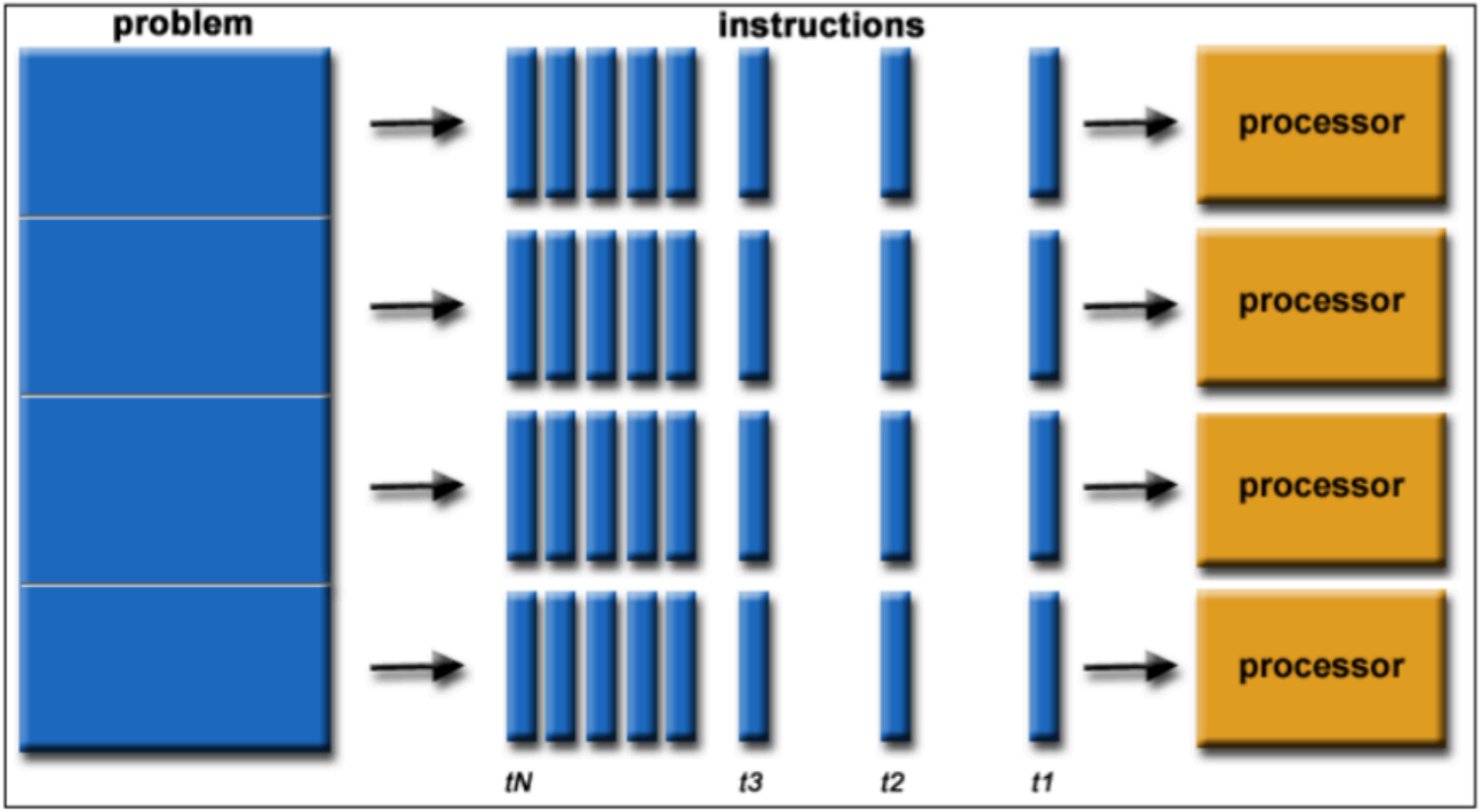
# PTHREADS

# Computação Serial



# Computação Paralela



# Computação Paralela

- Forma de computação em que vários cálculos podem ser feitos simultaneamente.
- Decomposição do algoritmo e/ou dos dados em partes
- Cada parte é uma tarefa que pode ser executada de forma independente
- Quando pode ser feito?

# Computação Paralela

- Quando pode ser feito?
  - Considere as seguintes tarefas a serem executadas:


  - O que acontece se A depende de B?
  - E se não houver dependência?

# Computação Paralela

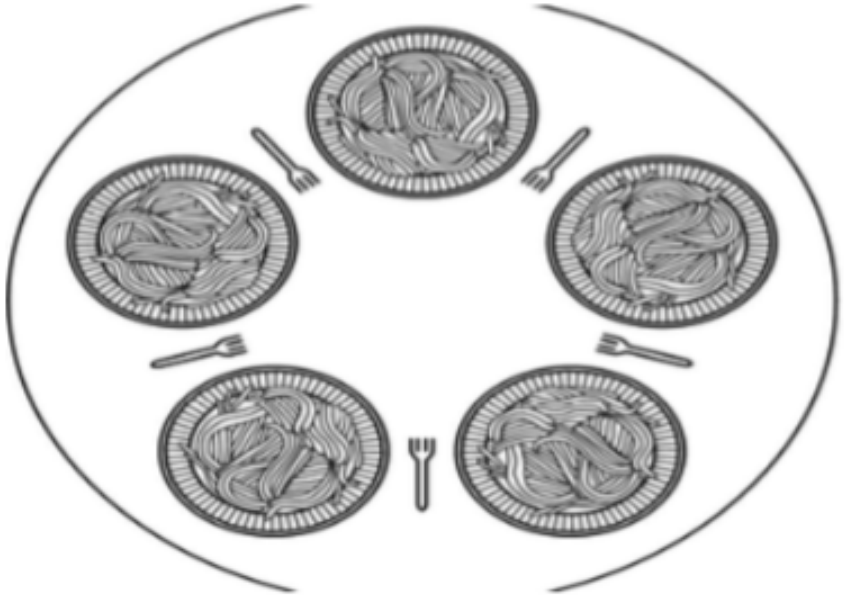
- Dados:
  - Você precisa processar os dados coletados no período de um mês. Para isso, deve-se utilizar o algoritmo X para cada um dos dias.
- Tarefas:
  - Para os dados processados para um dia, você deve retirar várias informações: quais redes enviam mais *spam*? Qual o custo cada uma das redes teve? Avaliar as relações entre as redes?
- Dados e tarefas:
  - Obter todas essas informações para o período de um mês

# Sincronização

- Coordenação de tarefas executadas em paralelo
- O objetivo da sincronização é evitar:
  - Condições de corrida: O resultado do programa depende da ordem que as tarefas são executadas
  - Deadlocks: Dependência circular que provoca a interrupção do processamento
  - Sessão crítica: Região do código que acessa algum recurso compartilhado

# O jantar dos filósofos

- Cinco filósofos em uma mesa circular.
- Eles podem estar pensando ou comendo.
- Para comer eles precisam de dois garfos, um de cada lado.
- Cada garfo é compartilhado por dois filósofos.



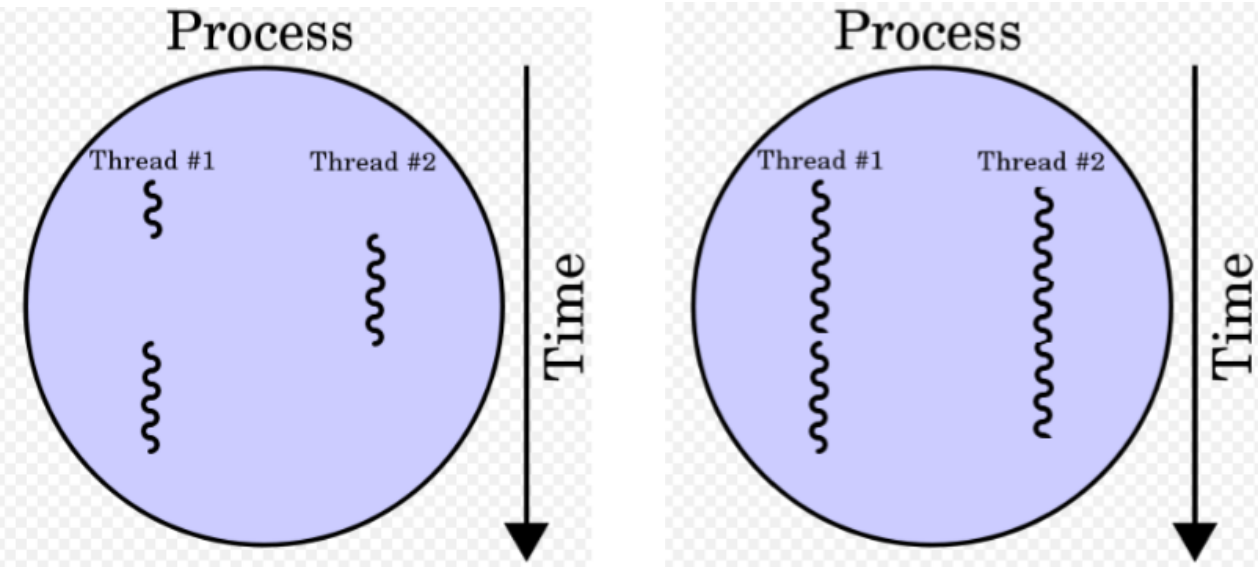
# O jantar dos filósofos

```
filosofo(i):
    while(true):
        wait(garfo[i])
        wait(garfo[i+1%5])
        eat()
        signal(garfo[i])
        signal(garfo[i+1%5])
```

- O acesso aos garfos deve ser sincronizado!

# Como paralelismo funciona em C?

- **Pthreads** é o padrão POSIX para threads.
- **Thread** é a unidade **escalável**.



# API Pthreads

- No Linux: utilizar a biblioteca **#include <pthread.h>**

  - Gerenciamento de threads: criação, terminação, etc.
  - Mutexes: Implementação de exclusão mútua
  - Variáveis de condição: Comunicação entre threads

# API Pthreads: Gerenciamento de threads

- Declaração: pthread_t_thread
- Criação: pthread_create(thread, attr, start routine, arg)
- Liberação: pthread_exit(retval)
- Bloqueio: pthread_join(thread id,status)
- Identificador: pthread_self()

# API Pthreads: Gerenciamento de threads

- Como criar uma thread?

```
pthread_t thread1;
pthread_create(&thread1, <attr>, vfunc, &parm);
void* vfunc(void* param) {
    //operações que a thread vai realizar
}
typedef struct param {
    int inicio, fim;
    long int soma;
}
```

# API Pthreads: Gerenciamento de threads

- **pthread_exit():** Termina a chamada da thread.

```
void *PrintHello(void *threadid) {
    long tid;
    tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
}
```

# API Pthreads: Gerenciamento de threads

- **pthread_join():** Espera a thread finalizar.

```
/* cria duas thread que irão executar a função inc_x(&x) */
pthread_create(&thread1, NULL, inc_x, &x1);
pthread_create(&thread2, NULL, inc_x, &x2);

/* espera as thread 1 e 2 finalizarem */
pthread_join(thread1, NULL); pthread_join(thread2, NULL);
```

# API Pthreads: Gerenciamento de threads

- pthread_attr_getstacksize (attr, stacksize)
  - Pega o tamanho da memória disponível para aquela thread.

- pthread_attr_setstacksize (attr, stacksize)
  - Altera o tamanho da memória disponível para aquela thread.

# API Pthreads: Mutexes

- Para a implementação de exclusão mútua
- Sincronização e proteção de dados compartilhados
- Uso típico:
    1. Cria e inicializa um mutex
    2. Múltiplas threads tentar alocar o mutex
    3. Somente uma thread obt´em o mutex
    4. A thread ”dona” do mutex realiza um conjunto de ac¸˜oes
    5. A thread desaloca o mutex
    6. Outra thread adquire o mutex e repete o processo
    7. O mutex é destruído
- Threads que não conseguem alocar o mutex podem ser bloqueadas ou não

# API Pthreads: Mutexes

- Declaração: p_thread_mutex_t_mutex
- Inicialização: p_thread_mutex_init(mutex,attr)
- Liberação: p_thread_mutex_destroy(mutex)
- Alocação do mutex(com bloqueio): p_thread_lock(mutex)
- Alocação do mutex(sem bloqueio): p_thread_trylock(mutex)
- Liberação do mutex: p_thread_mutex_unlock(mutex)

# API Pthreads: Mutexes

- Variáveis mutex:
  - **pthread_mutex_t** mutexsum;
  - **pthread_mutex_init**(&mutexsum, NULL);

```
void *inc_x(void *x_void_ptr) { int i;
        tinfo *x_ptr = (tinfo *)x_void_ptr;
        /* soma o vetor global */
        pthread_mutex_lock (&mutexsum);
        for (i=x_ptr->inicio; i<x_ptr->fim; i++) {
                soma_total += vetor[i];
        }
        pthread_mutex_unlock (&mutexsum);
}
```

# API Pthreads: Variáveis de Condição

- Utilizada na sincronização entre threads: Enquanto com mutex temos a sincronização através do controlo do acesso aos dados, as variáveis de condição permitem a sincronização de threads através do valor desses dados.
- Permite que uma thread seja suspensa até alguma condição seja atendida
- O acesso à variável de condição deve ser controlado via mutex

# API Pthreads: Variáveis de Condição

Thread A:

1. Trava mutex e checa variável
2. Chama pthread_cond_wait() e espera pela Thread B
3. Acorda e destrava mutex

Thread B:

1. Trava mutex
2. Muda o valor da variável que A está esperando
3. Se a condição foi satisfeita, sinaliza A e destrava mutex