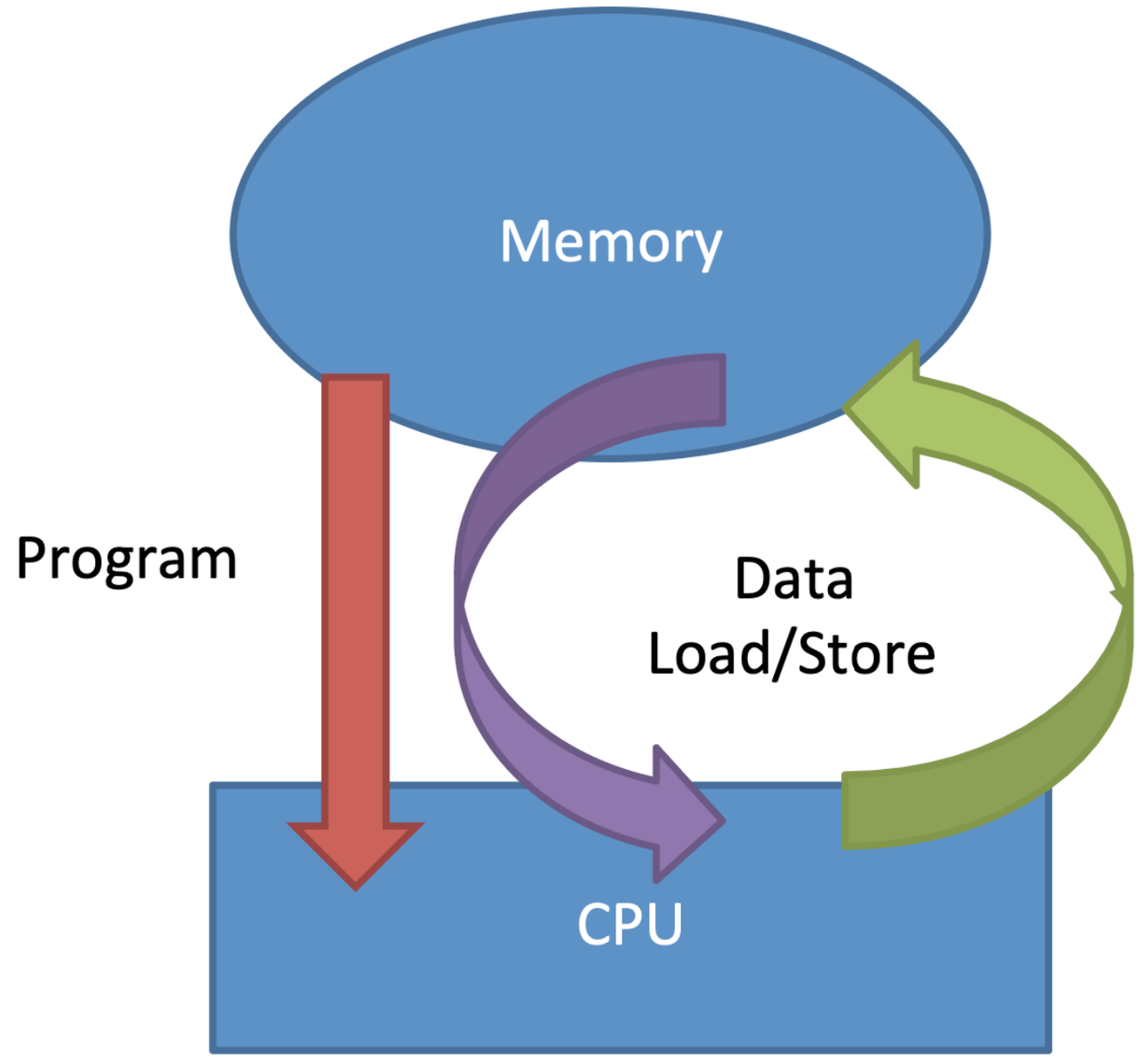
UNIVERSIDAD DE COSTA RICA

# HPC: PARALELISMO DE MEMORIA DISTRIBUIDA

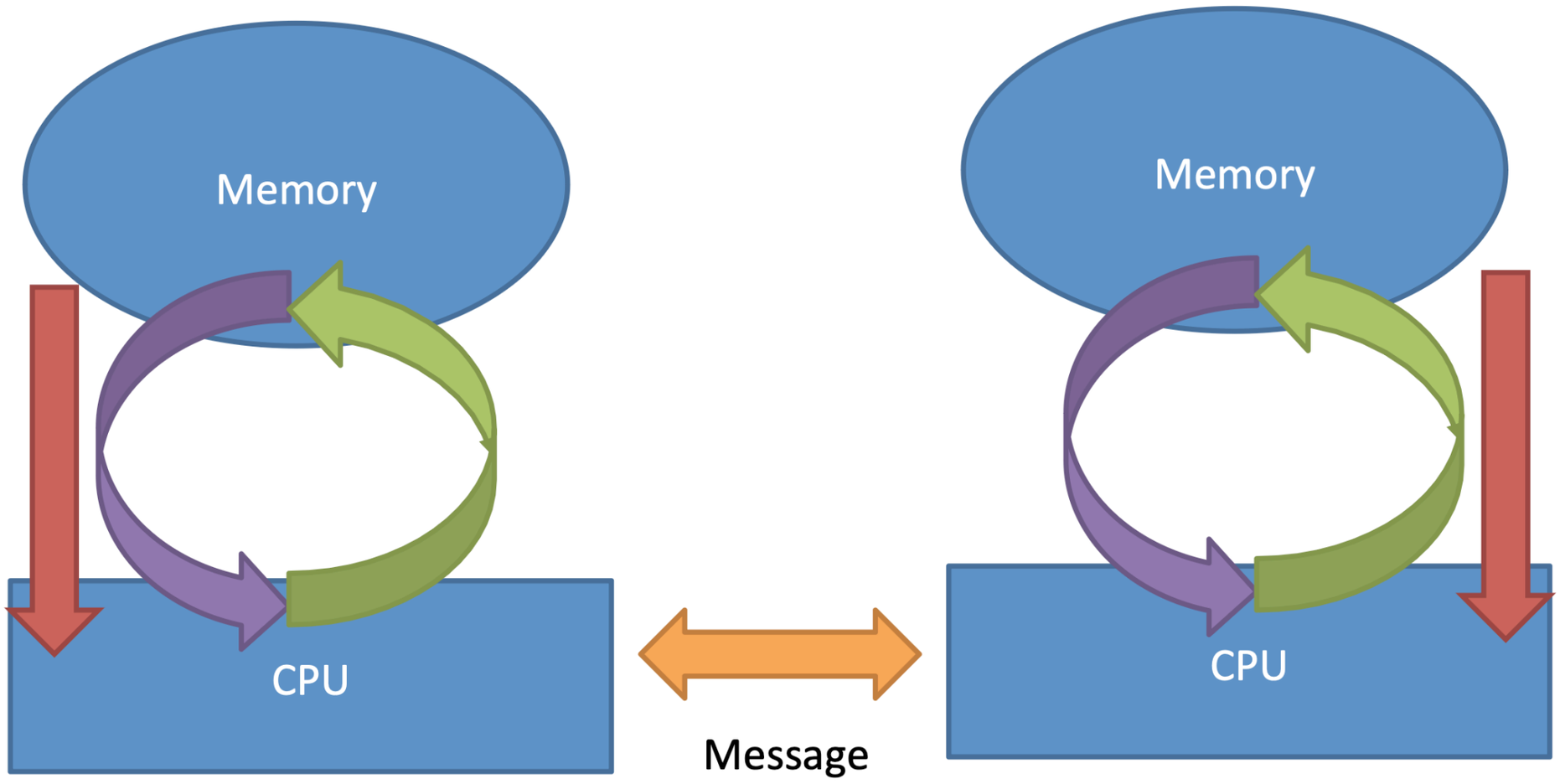
Prof. Marlon Brenes y Prof. Federico Muñoz
Escuela de Física, Universidad de Costa Rica

EFis Escuela de Física

# Programación en serie

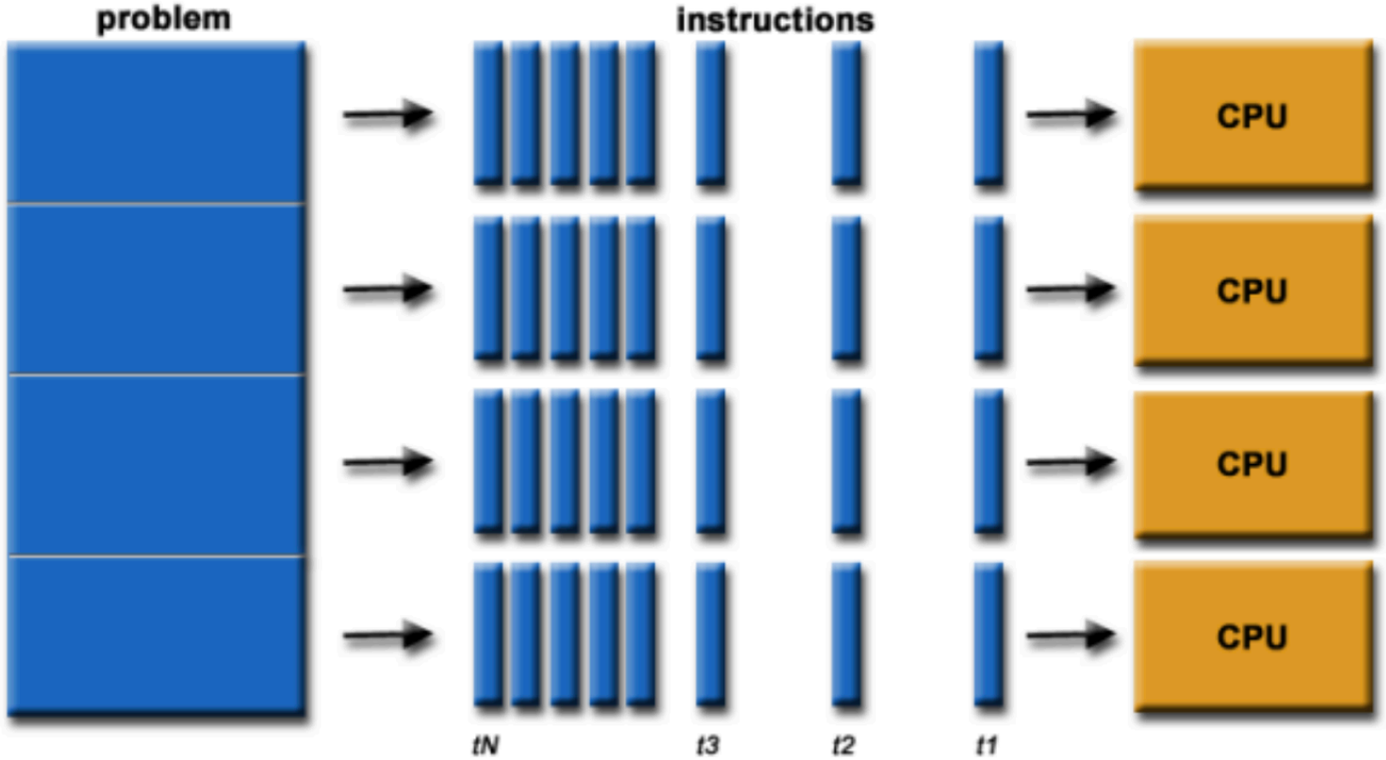


# Programación en paralelo

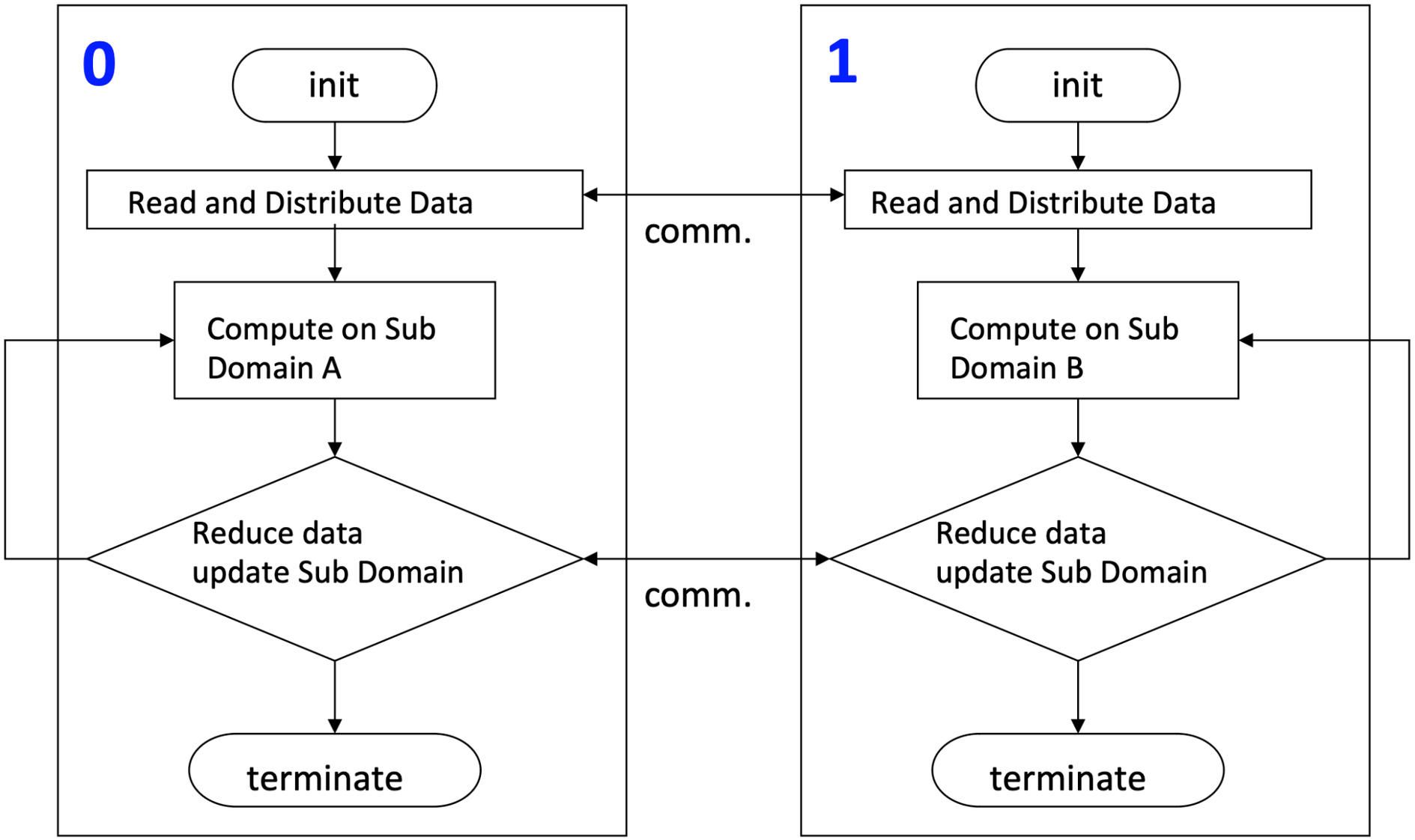


# Concurrencia

- El primer paso en el desarrollo de un algoritmo paralelo es descomponer el problema en tareas que pueden ser ejecutadas de manera **concurrente**
    - **El problema se divide en partes discretas**
    - **Cada unidad se divide en una serie de instrucciones**
    - **Un mecanismo de control/coordinación es usado**

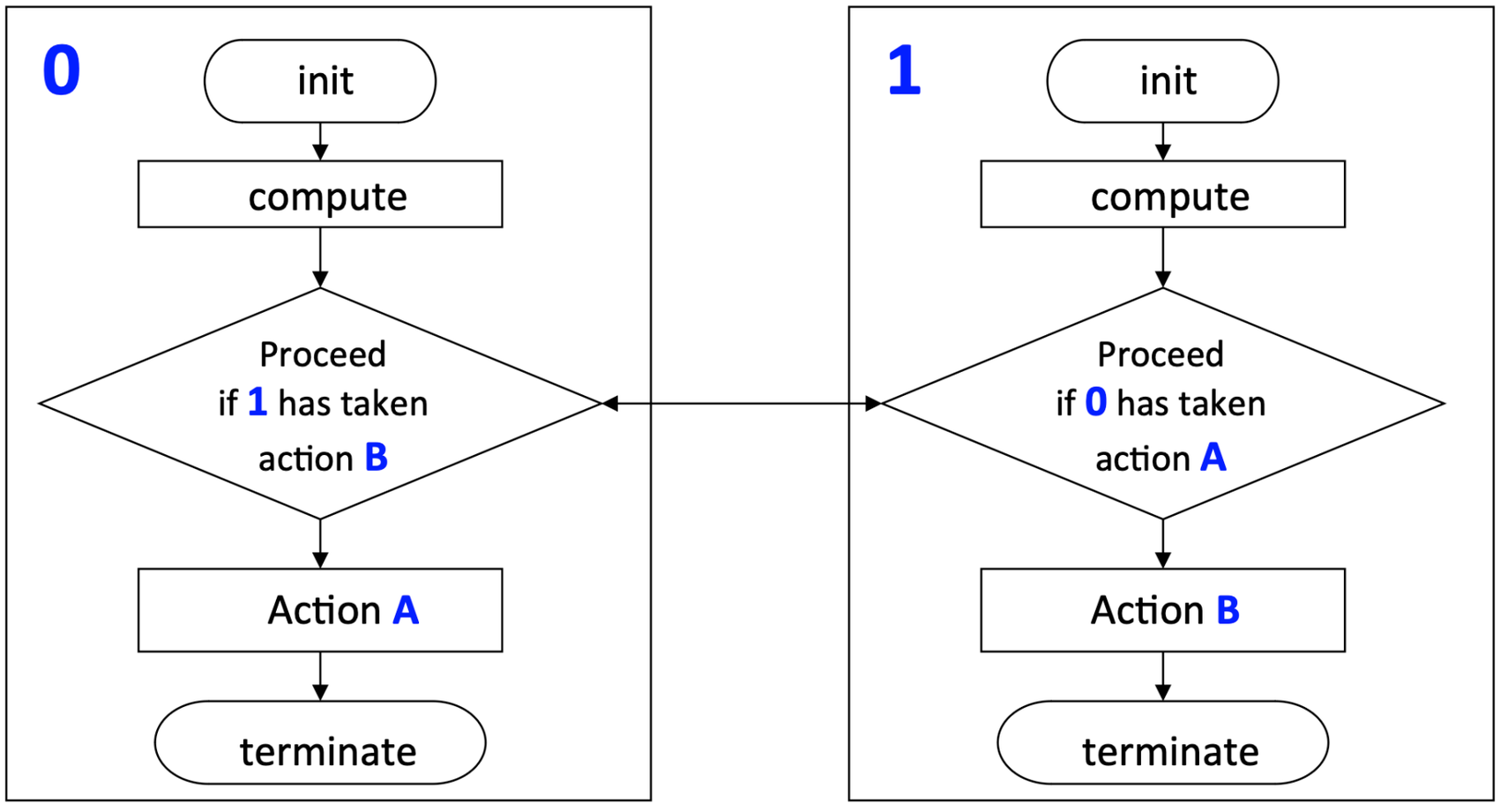


# Que es un programa paralelo?



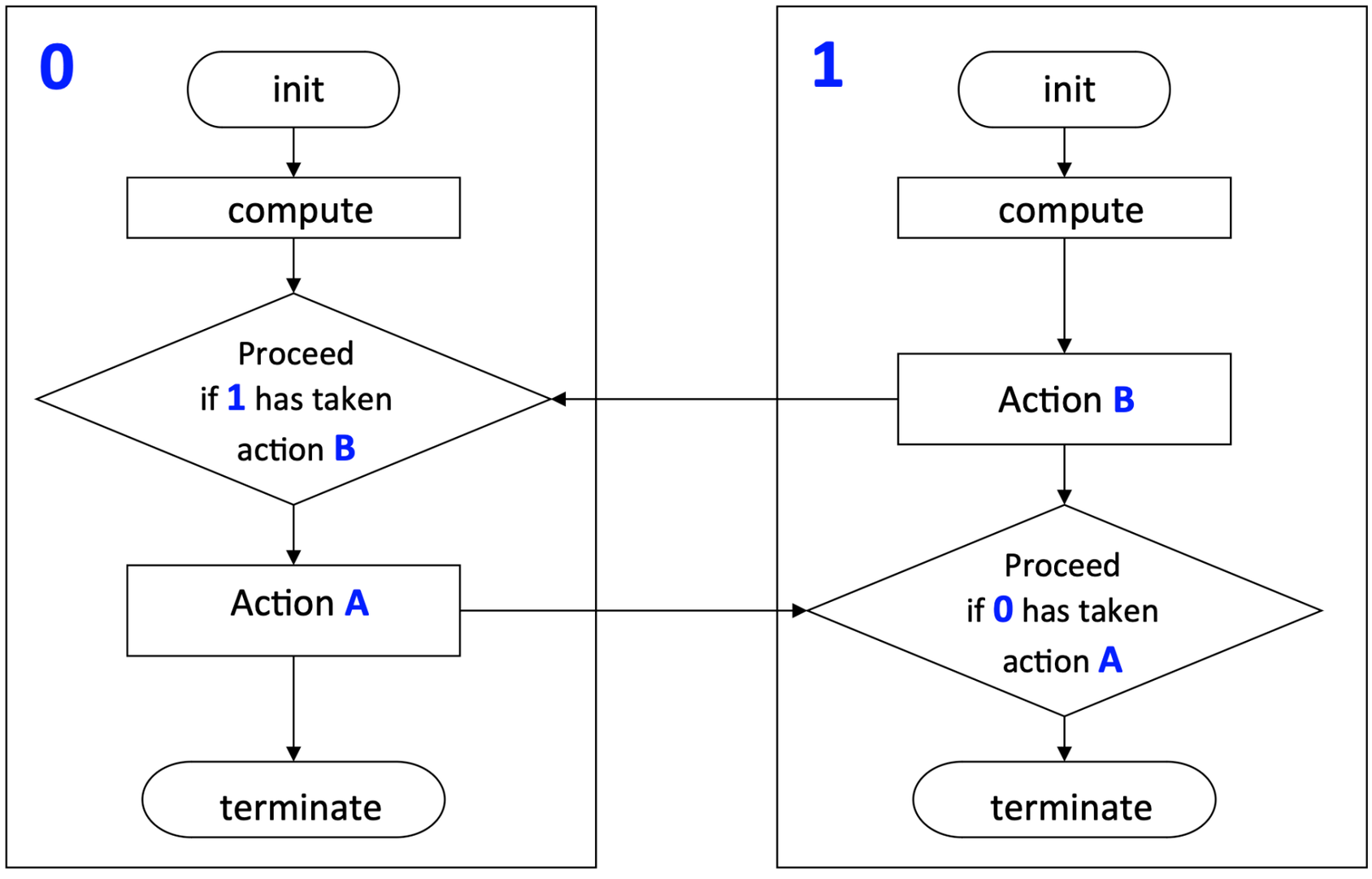
# Paradigma de paso de mensajes

- Deadlock:
  - Un deadlock ocurre cuando dos o más procesos se bloquean entre ellos y esperan a que los otros progresen



# Paradigma de paso de mensajes

- Deadlock: Como evitarlo

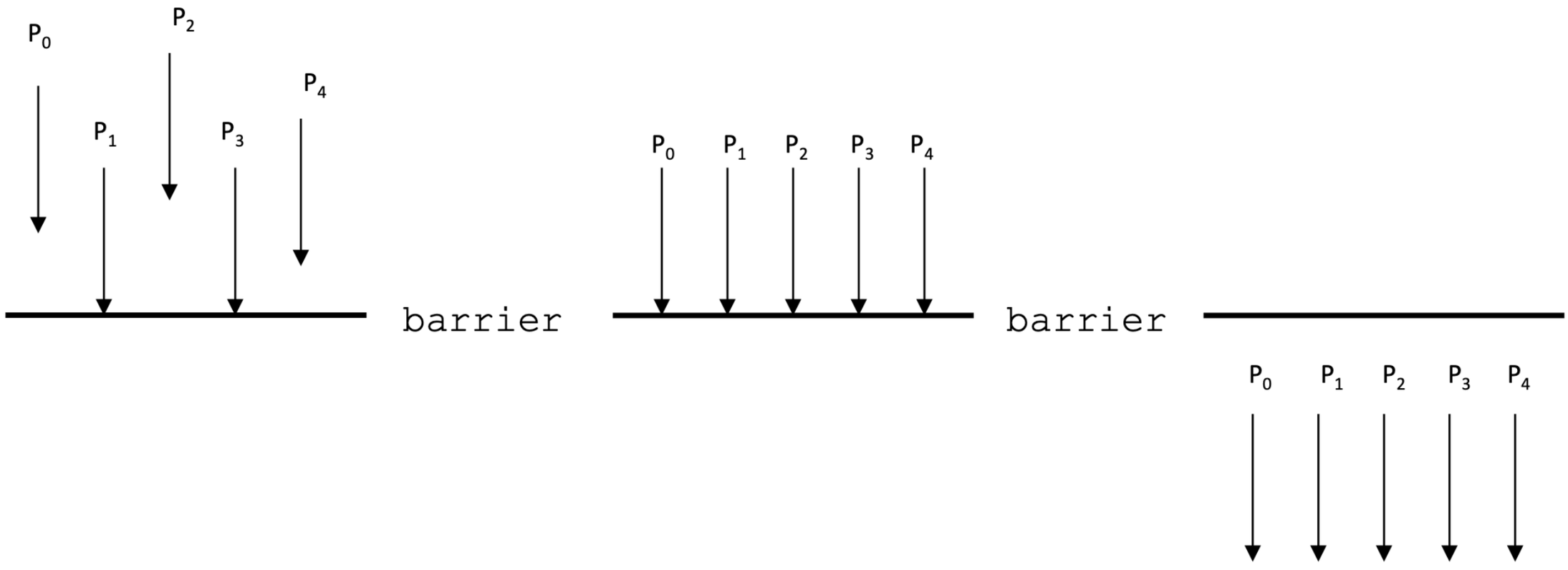


# Paradigma de paso de mensajes

- A la medida de lo posible:
    - Reducir eventos de comunicación
    - Agrupar comunicaciones pequeñas en un solo mensaje grande

- Eliminar sincronización siempre que sea posible
    - Cada sincronización reduce el desempeño a aquel del proceso mas lento

# Sincronización



# Diseño de programa: MPI (message passing interface)

- Procesos múltiples y separados (los cuales pueden ser locales o remotos) son coordinados de manera concurrente para intercambiar datos a través de **mensajes**
    - Los procesos, en principio, no comparten memoria con otros procesos
- La idea es distribuir campos de datos muy grandes, replicando la menor cantidad de información
- Ley de Amdahl: la aceleración está limitada por la fracción serial más la comunicación

# MPI

- MPI es un estándar
  - Existe documentación para describir como las funciones de la API deben ser llamadas y su comportamiento general
  - Existen tres niveles: MPI-1 (básico), MPI-2 (intermedio) y MPI-3 (avanzado)
  - http://www.mpi-forum.org
  - https://mpitutorial.com/

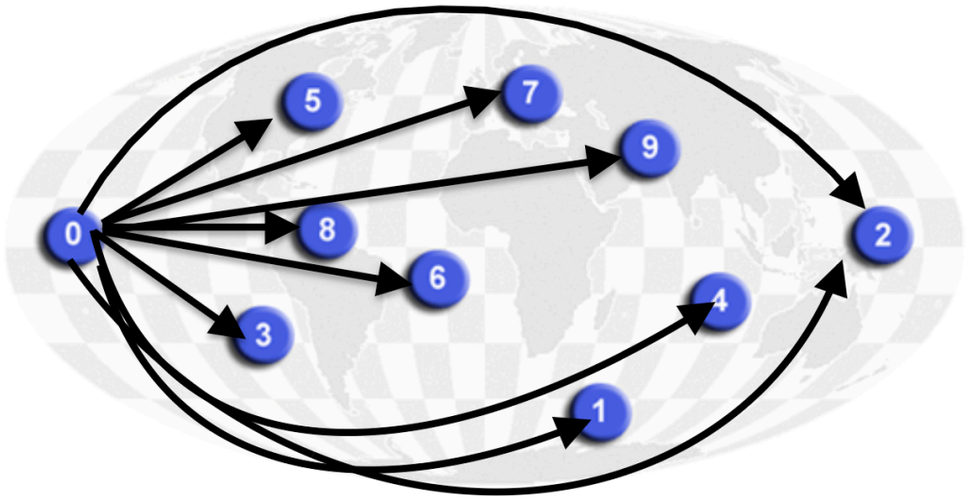
- Las implementaciones de MPI se utilizan para esconder los detalles de la comunicación que ocurre a nivel de hardware

- Existen diversas implementaciones, e.g., OpenMPI e Intel MPI

# Comunicadores

- Los comunicadores son el objeto fundamental que provee MPI
  - La comunicación y el paso de mensajes ocurre a través de un comunicador
  - Las fuentes y los destinatarios se identifican con un **rango**

**MPI_COMM_WORLD**




- Por defecto, el comunicador principal involucra a todos los procesos que son invocados al momento de la ejecución
  - Size: el número de procesos MPI
  - Rank: el ID del proceso dentro del grupo de comunicación

# Hola mundo

- **Ver:** `hello.cpp`

- Para compilar: `mpicxx hello.cpp -o hello.x`
    - Asumiendo el compilador de Intel

- Para ejecutar: `mpirun -np X ./hello.x`
    - Donde X es el número de procesos a utilizar

# Fases de un programa MPI

- I) Inicialización:
    - Paso de argumentos a la aplicación
    - Identificación de la región paralela
    - Lectura y distribución de datos
- II) Ejecución:
    - Ejecución de subrutinas con trabajo en paralelo
    - Estas subrutinas pueden ser la misma o diferentes, dependiendo del rango del proceso
- III) Cierre y limpieza

# El mensaje

- Un mensaje es un arreglo de objetos de cierto tipo MPI

- MPI define los tipos que pueden ser enviados como mensajes

Message Structure

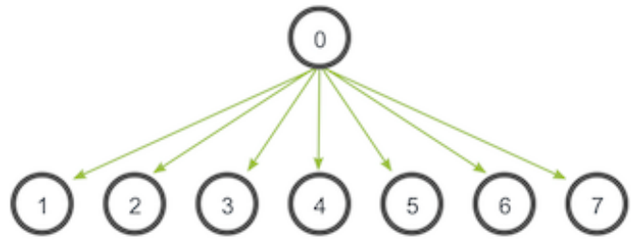
| envelope | | | | body | | |
|---|---|---|---|---|---|---|
| source | destination | communicator | tag | buffer | count | datatype |

# MPI datatypes

| MPI datatype | C datatype | C++ datatype |
|---|---|---|
| MPI::CHAR | `char` | `char` |
| MPI::SHORT | `signed short` | `signed short` |
| MPI::INT | `signed int` | `signed int` |
| MPI::LONG | `signed long` | `signed long` |
| MPI::LONG_LONG | `signed long long` | `signed long long` |
| MPI::SIGNED_CHAR | `signed char` | `signed char` |
| MPI::UNSIGNED_CHAR | `unsigned char` | `unsigned char` |
| MPI::UNSIGNED_SHORT | `unsigned short` | `unsigned short` |
| MPI::UNSIGNED | `unsigned int` | `unsigned int` |
| MPI::UNSIGNED_LONG | `unsigned long` | `unsigned long int` |
| MPI::UNSIGNED_LONG_LONG | `unsigned long long` | `unsigned long long` |
| MPI::FLOAT | `float` | `float` |
| MPI::DOUBLE | `double` | `double` |
| MPI::LONG_DOUBLE | `long double` | `long double` |
| MPI::BOOL | | `bool` |
| MPI::COMPLEX | | `Complex<float>` |
| MPI::DOUBLE_COMPLEX | | `Complex<double>` |
| MPI::LONG_DOUBLE_COMPLEX | | `Complex<long double>` |
| MPI::WCHAR | `wchar_t` | `wchar_t` |
| MPI::BYTE | | |
| MPI::PACKED | | |

# Broadcast

- Se utiliza para pasar un mensaje de un proceso al resto de los procesos asignados en el momento de la ejecución



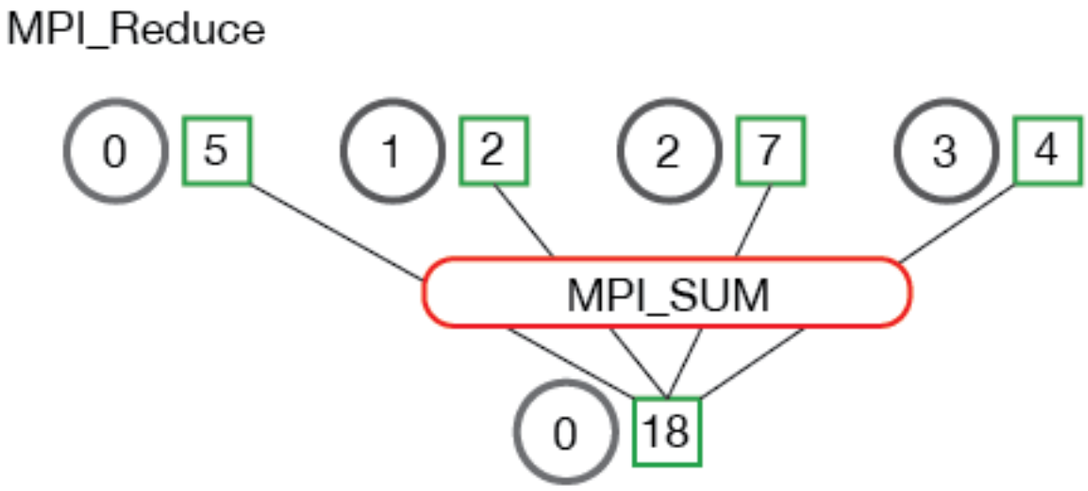

- Ver: `broadcast.cpp`

```
MPI_Bcast(
    void* data,
    int count,
    MPI_Datatype datatype,
    int root,
    MPI_Comm communicator)
```

# Reducción

- La reducción es una operación fundamental en computación científica

```
MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm communicator)
```



# Ejemplo

- Ver: `pi.cpp`