



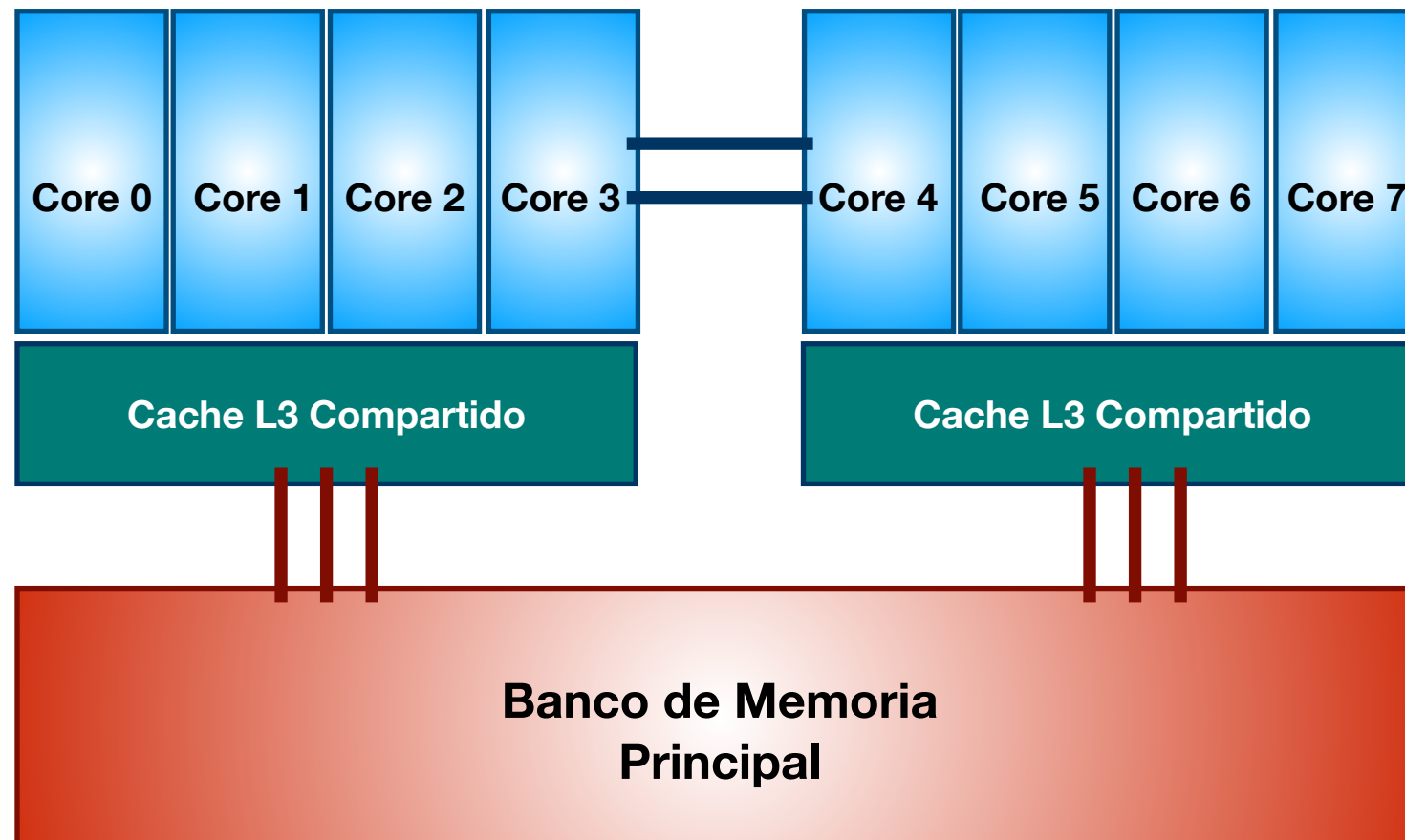
UNIVERSIDAD DE COSTA RICA

# HPC: PARALELISMO DE MEMORIA COMPARTIDA

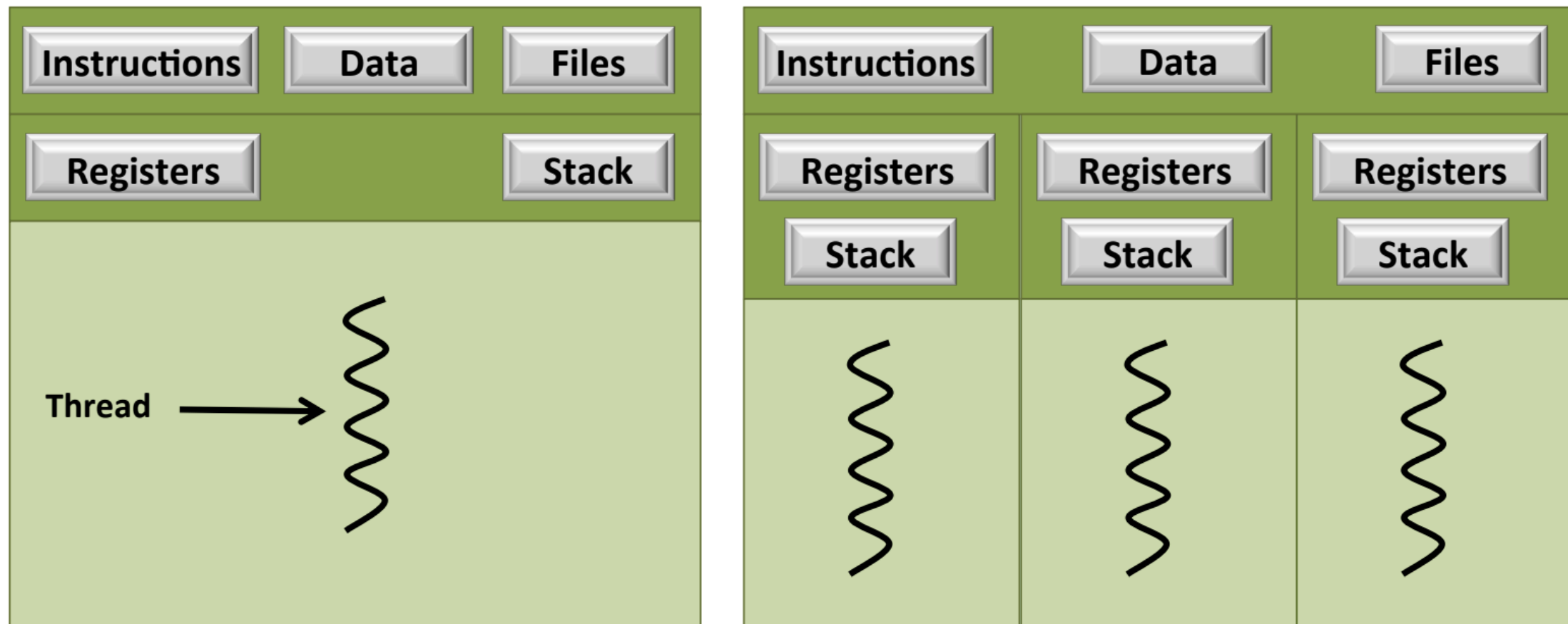
---

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

# Sistema multi-CPU y multinúcleo NUMA



# Procesos e hilos (*threads*)



---

# Hilos

---

- Un hilo es una imagen de un proceso: una instancia del programa con sus propios datos (memoria privada)
- Cada hilo puede seguir su propio flujo de control a través del programa
- Los hilos pueden compartir datos con otros hilos, pero también contienen información privada a esos hilos
- La comunicación inter-hilo ocurre mediante accesos al banco de memoria compartida
- Existe un hilo principal cuya función es coordinar y sincronizar todos los hilos de un grupo

---

# OpenMP

---



# OpenMP


---

- OpenMP (open specialisation for multi-processing) no es un lenguaje de programación
  - Trabaja en conjunto con otros lenguajes existentes de programación como C/C++
  - De momento, no es posible trabajar directamente OpenMP con Python. Esto requiere de bibliotecas externas como Cython o Numba.
  - En Python, usualmente utilizamos multi-threading por debajo mediante diferentes bibliotecas pre-compiladas (e.g., numpy)
- Application programming interface (API)
  - Estas interfaces de programación proveen la forma mas portable para aplicaciones en paralelo
  - Contiene tres componentes principales:
    - Directivas de compilador
    - Rutinas que se invocan al momento de ejecución
    - Variables de ambiente

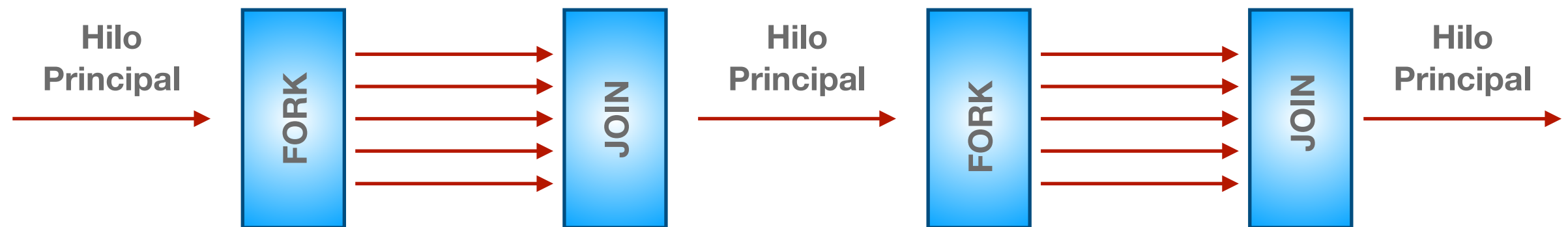
---

# OpenMP

---

- OpenMP trabaja mediante directivas de compilación
  -  • Sin embargo, puede trabajar sin dichas directivas
- Se puede añadir al código de manera incremental (importante)
- Solamente trabaja bajo el paradigma de memoria compartida
  - Nodos con arquitectura multi-procesador y/o muti-núcleos
- OpenMP esconde los llamados a la biblioteca que genera y manipula los hilos
  - Esto da poca flexibilidad en general, sin embargo, requiere de muy poca programación usualmente
- **Peligro:** accesos de escritura a secciones de memoria compartidas puede dar lugar a condiciones de fallo de sincronización y/o corrupción de memoria

# Modelo *Fork-Join*



- Hilos dinámicos
- Paralelismo explícito
- Declaraciones ocurren mediante directivas al compilador



---

# Conceptos básicos

---

- Las construcciones con OpenMP caen dentro de cinco categorías
  - Regiones paralelas
  - Distribución de trabajo
  - Ambientes de datos (*scopes*)
  - Sincronización
  - Funciones que se invocan al momento de ejecución y/o variables de ambiente

---

# Formato de las directivas

---

- Una directiva es una línea especial de código fuente que sólo tiene significado para ciertos compiladores
- Las directivas se introducen al **inicio del scope de la región paralela**

```
#pragma omp parallel
```



- Esta directiva seguida de un scope, inicializa el ambiente de OpenMP

---

# Hola mundo paralelo

---

- `Ver:hello.cpp`
- **Compilación:**
  - `g++ hello.cpp -o hello.x -fopenmp`
- Al ejecutar el binario, las hileras de caracteres impresas en stdout se pueden corromper y los hilos imprimen en desorden. **Porqué?** Intente ejecutar el programa varias veces.

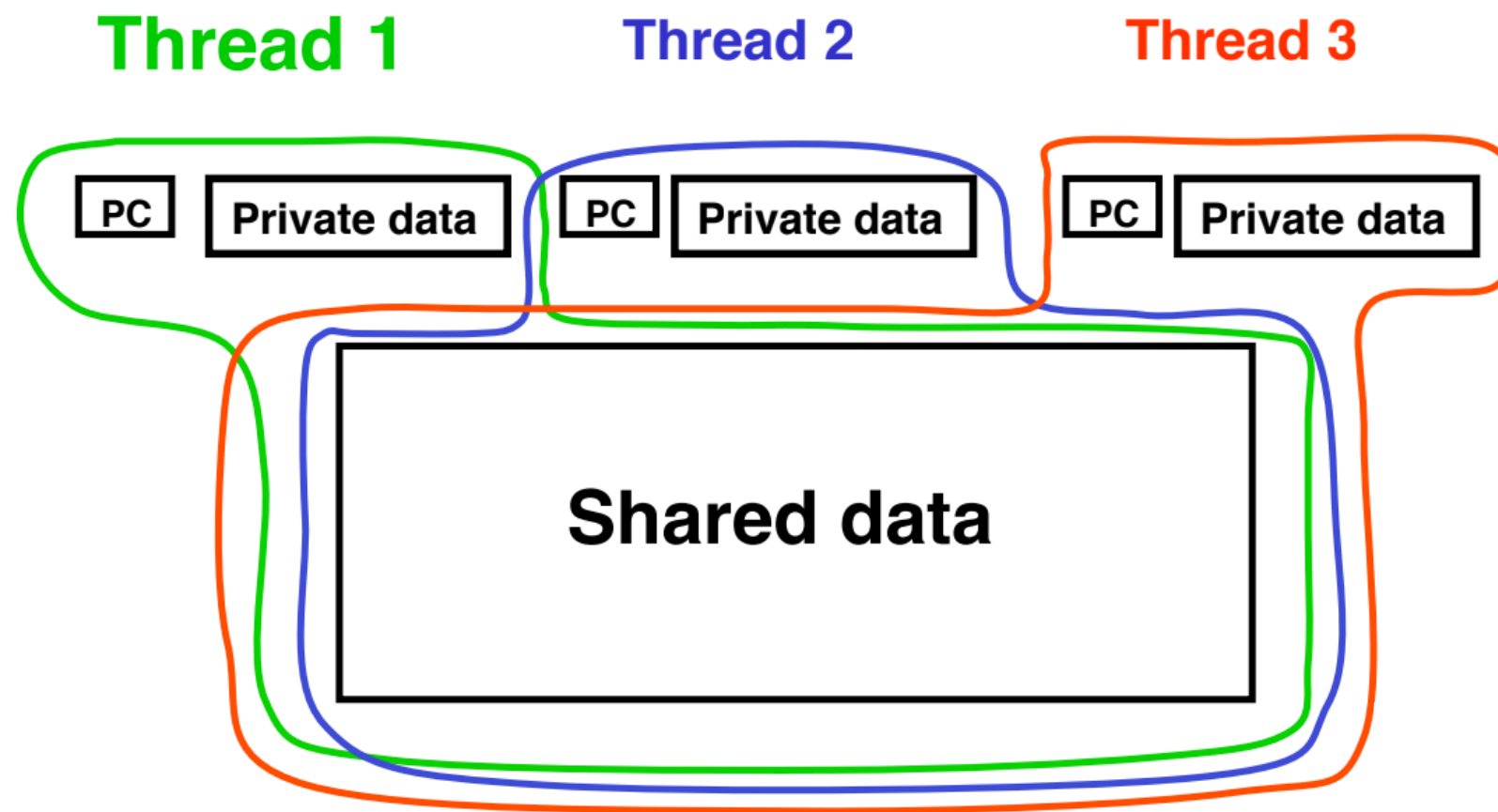
---

# API para directivas al momento de ejecución

---

- `omp_get_num_threads()` : Devuelve el número de hilos actual
- `omp_get_thread_num()` : Devuelve el ID del hilo
- `omp_set_num_threads(n)` : Asigna el número de hilos de ejecución
- `omp_in_parallel` : Devuelve `true` si se invoca dentro de una región paralela
- `omp_get_max_threads` : Devuelve el número máximo de hilos posibles

# Huella de memoria



# Scopes de variables en ambientes paralelos

- Todas las variables declaradas previas a un scope paralelo siguen existiendo dentro del scope paralelo
  - Por defecto, dichas variables son **compartidas** por todos los hilos
- Por otro lado, todas las variables declaradas dentro un scope paralelo son privadas a cada hilo por defecto
  - Se pueden declarar privadas o compartidas como parte de la directiva de compilador
  - Por ejemplo, los índices de un for loop en un ambiente paralelo son variables privadas
- La cláusula `firstprivate` inicializa instancias privadas de una variable u objeto con los contenidos de una variable compartida

---

# Paralelismo de `for` loops

---

- En la gran mayoría de instancias de paralelismo en computación científica, nos interesa explotar el paralelismo para implementar `for` loops de manera concurrente
  - La idea es dividir el número de iteraciones entre el número total de hilos
    - Esto es muy fácil de implementar con OpenMP
    - Da lugar a código muy fácil de leer
    - Es la instancia de paralelismo más utilizada comúnmente

---

# Paralelismo de `for` loops

---

- **Ver:** `vector.cpp`
- **Compilación:**
  - `g++ vector.cpp -o vector.x -fopenmp`
- **Controlar el número de hilos mediante variables de ambiente:**
  - `export OMP_NUM_THREADS=N`
  - `N` corresponde al número de hilos que deseamos utilizar
- **Ejercicio:** calcular la escalabilidad del problema en nuestras computadoras



---

# Laboratorio

---

- Paralelizar el código `pi.cpp`
- Evaluar escalabilidad
- Importante! El código contiene una operación, conocida en el mundo del paralelismo, como operación atómica. Este tipo de operaciones son aquellas en las que utilizamos los operadores `+=`, `-=`, `*=`, `/=`. En este caso, todos los hilos van a querer escribir sobre el mismo espacio de memoria! Si no se hace con cuidado, esto daría lugar a una corrupción de memoria.
  - Con OpenMP, estas operaciones están automatizadas con la cláusula `reduction(op: variables)`