



# Calcolo Parallelo e Distribuito

---

Software parallelo – introduzione alla libreria OpenMP  
l'ambiente MIMD-SM

**Docente:** Prof. L. Marcellino

**Tutor:** Prof. P. De Luca

# Shared vs Distributed

Calcolatori MIMD a  
memoria **condivisa**

(shared)

Necessità di  
sincronizzare gli  
accessi in scrittura e  
organizzare il lavoro

MEMORIA

Calcolatori MIMD a  
memoria **distribuita**

(distributed)

Necessità di  
organizzare le  
comunicazioni tra i  
processi

MEMORIA

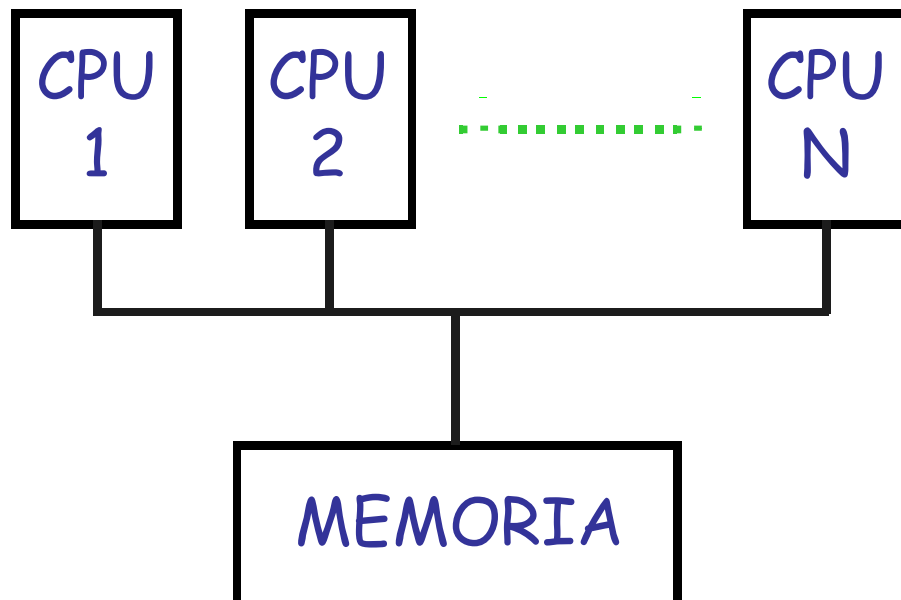
CPU  
1

CPU  
2

CPU  
N

# Shared vs Distributed

Calcolatori MIMD a  
memoria **condivisa**  
(shared-memory)



MIMD-SM

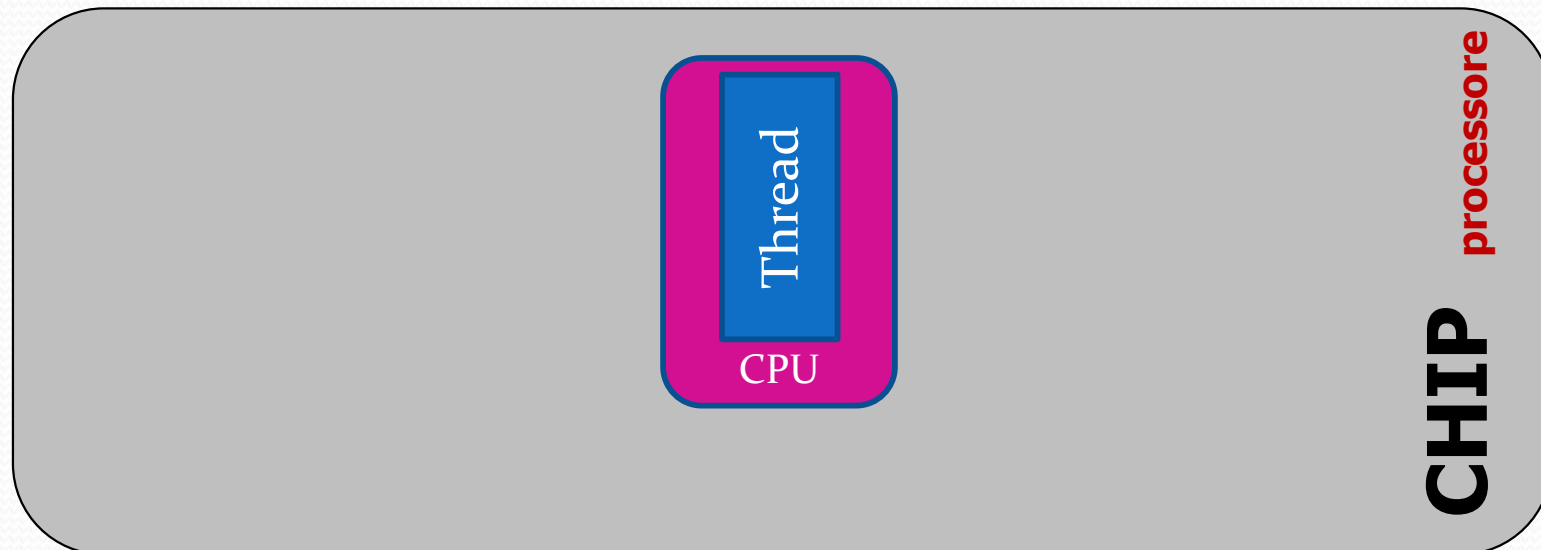
multicore

Necessità di  
sincronizzare gli  
accessi in scrittura e  
organizzare il lavoro

# Threads

Per un sistema operativo moderno, l'unità base di utilizzo della CPU è il **thread**

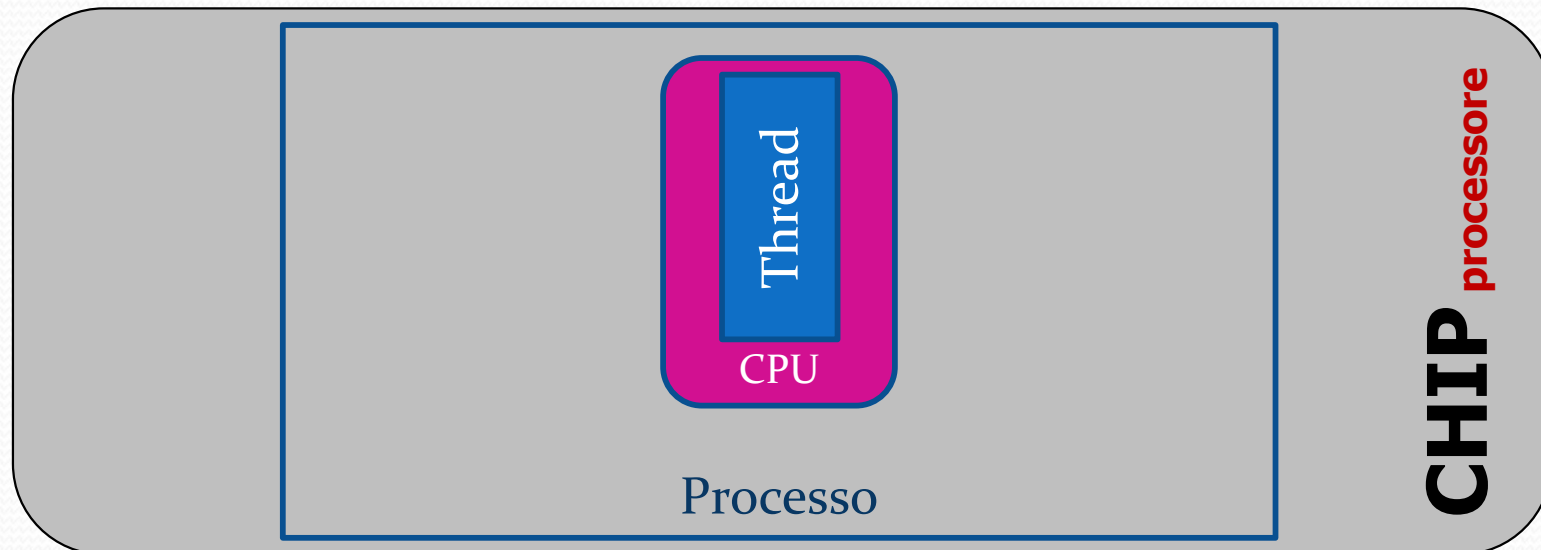
**Un thread è quindi un flusso di istruzioni indipendente che deve essere eseguito sequenzialmente su una CPU**



# Threads vs Processi

Un **processo** si definisce banalmente come  
“un programma in esecuzione”.

Un **processo** è costituito da almeno un thread ...

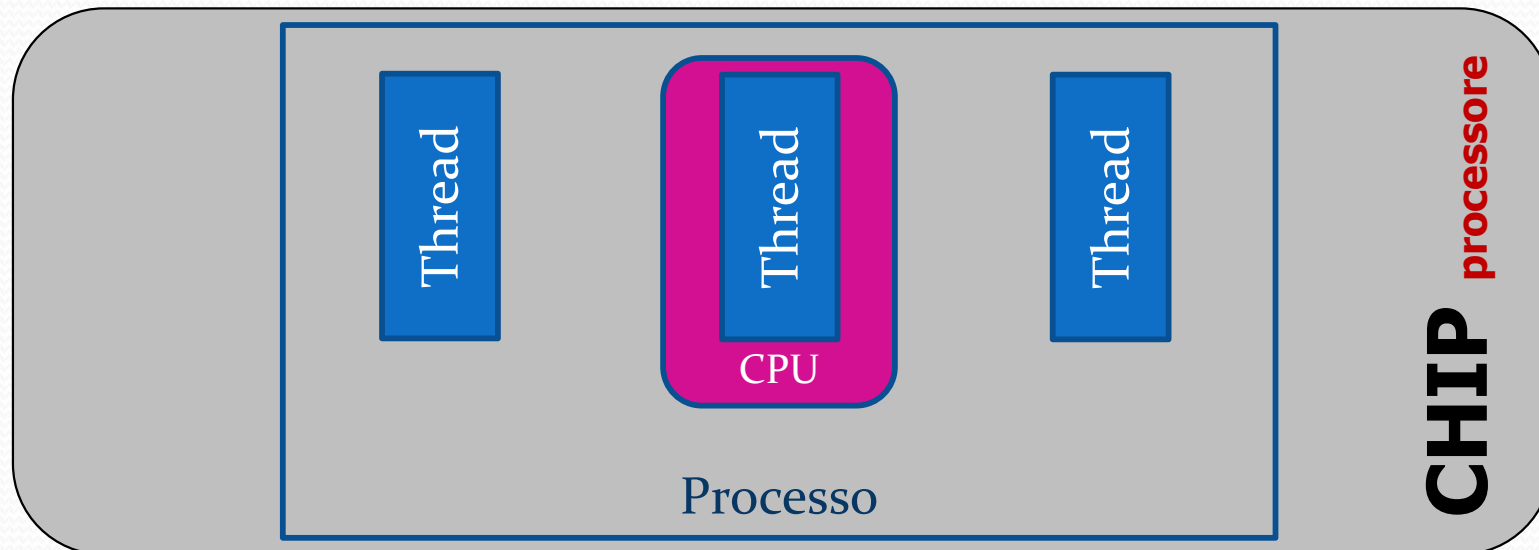




# Threads vs Processi

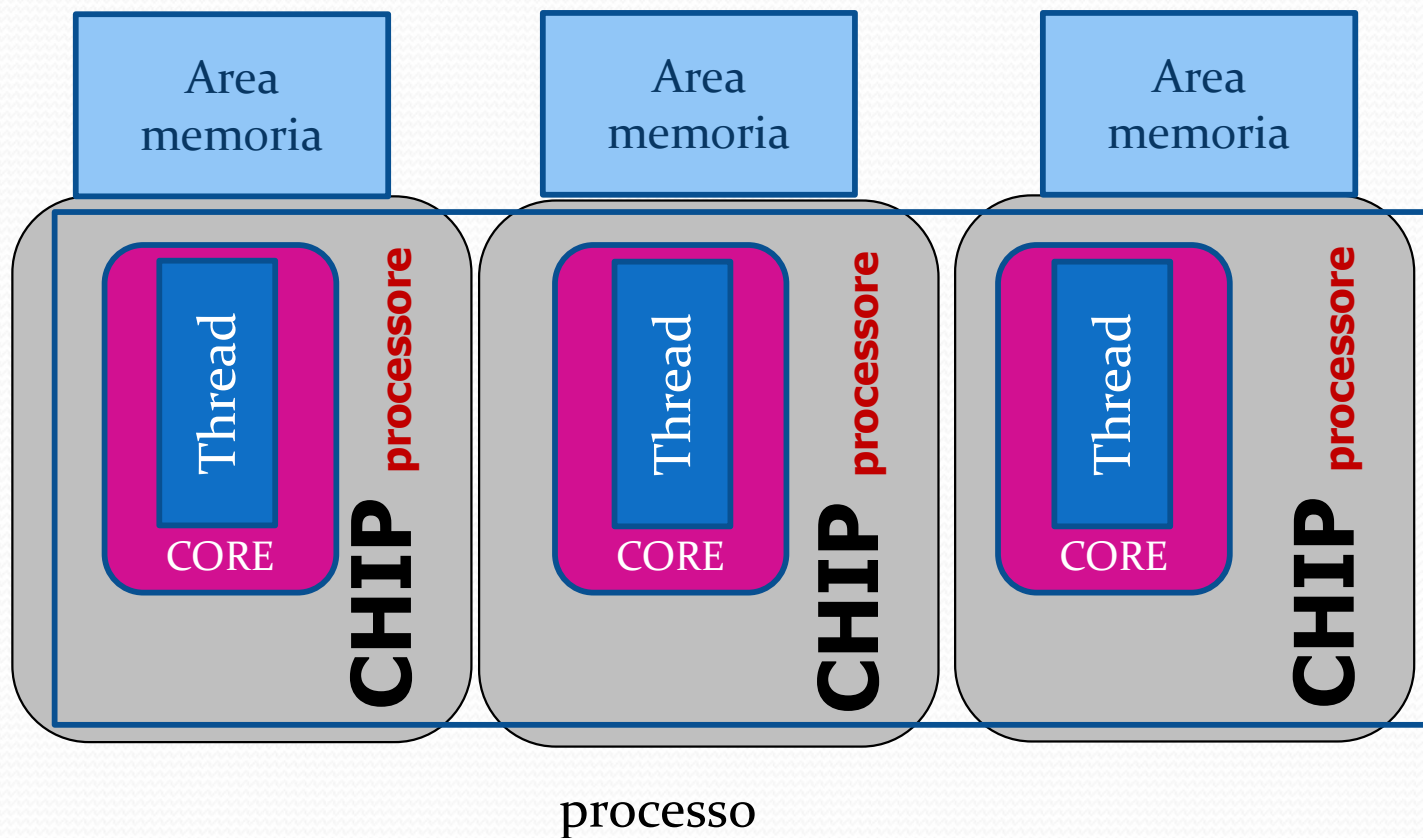
Un **processo** si definisce banalmente come  
“un programma in esecuzione”.

Un **processo** è costituito da almeno un thread ...  
... ma può contenerne più di uno



# Threads vs Processi

Thread diversi possono essere eseguiti indipendentemente su  
**cpu** diverse **MIMD-DM**

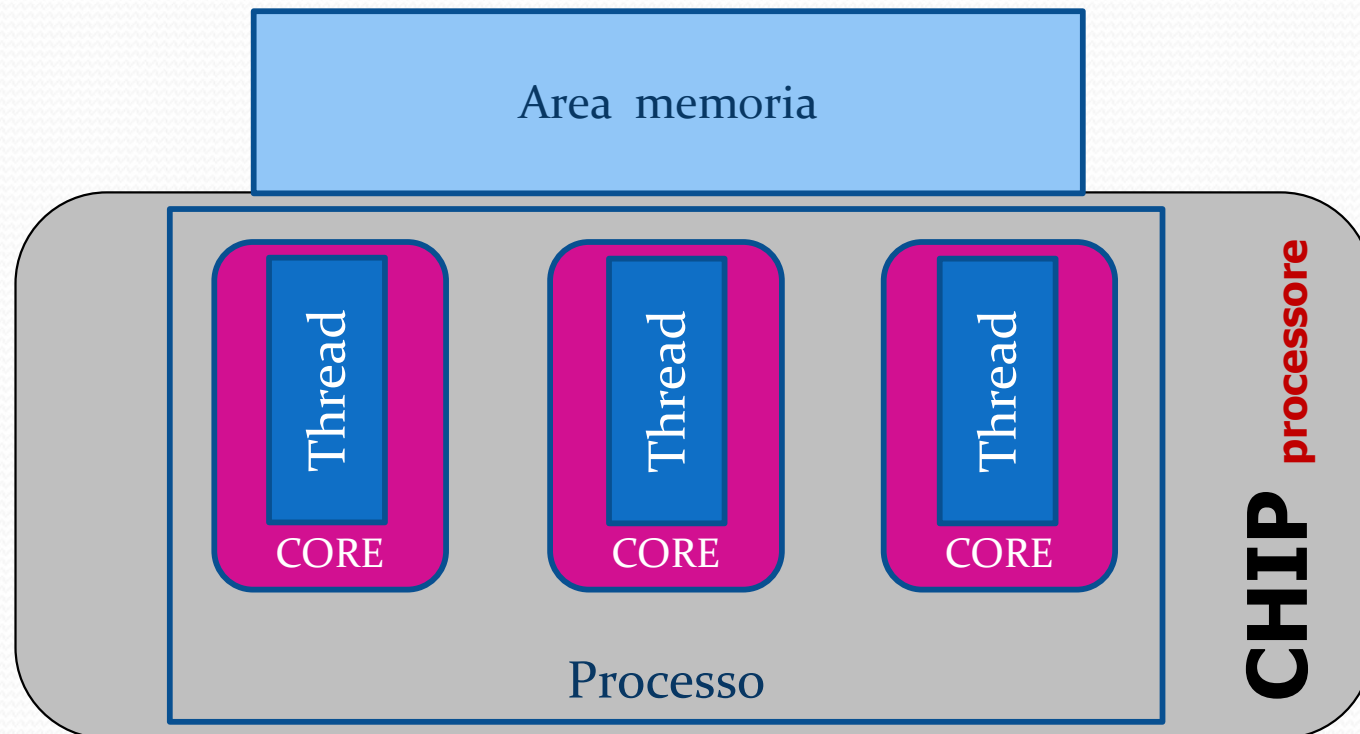




# Threads vs Processi

Thread diversi possono essere eseguiti indipendentemente

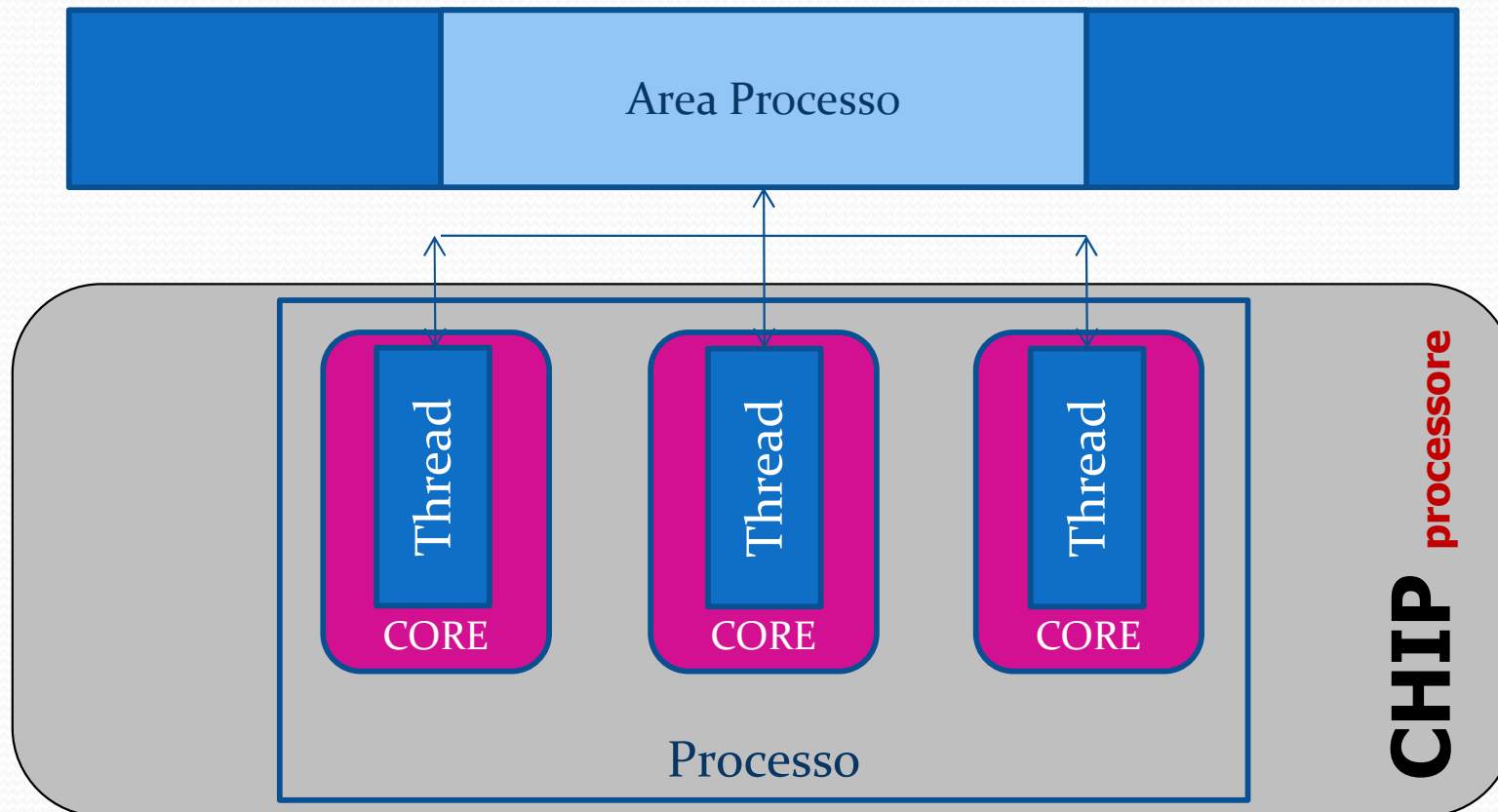
**core** diversi **multicore** **MIMD-SM**





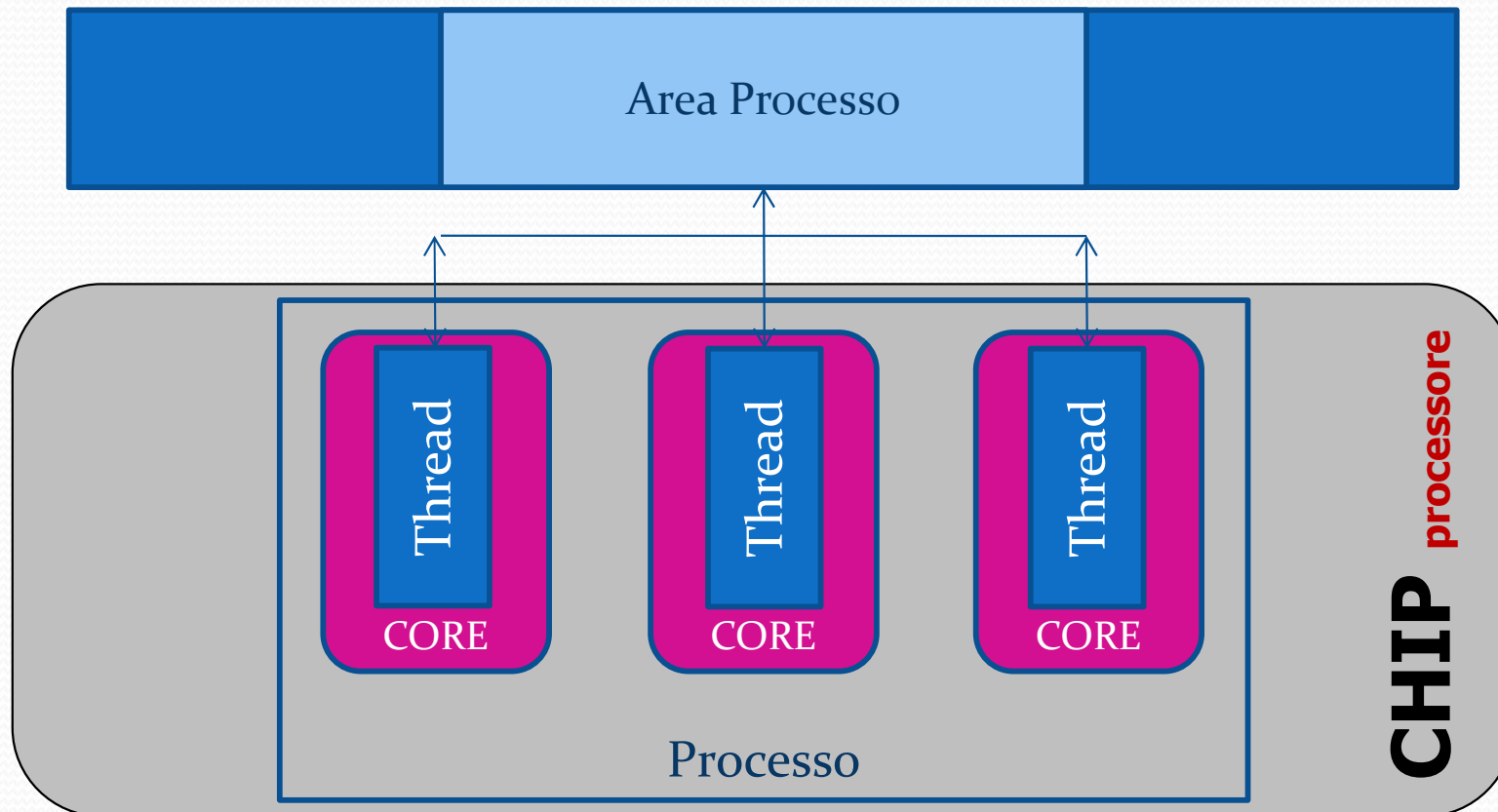
# Architettura multicore

I thread di uno stesso processo **condividono** la stessa area di memoria



# Architettura multicore

Lavorano insieme in maniera naturale.

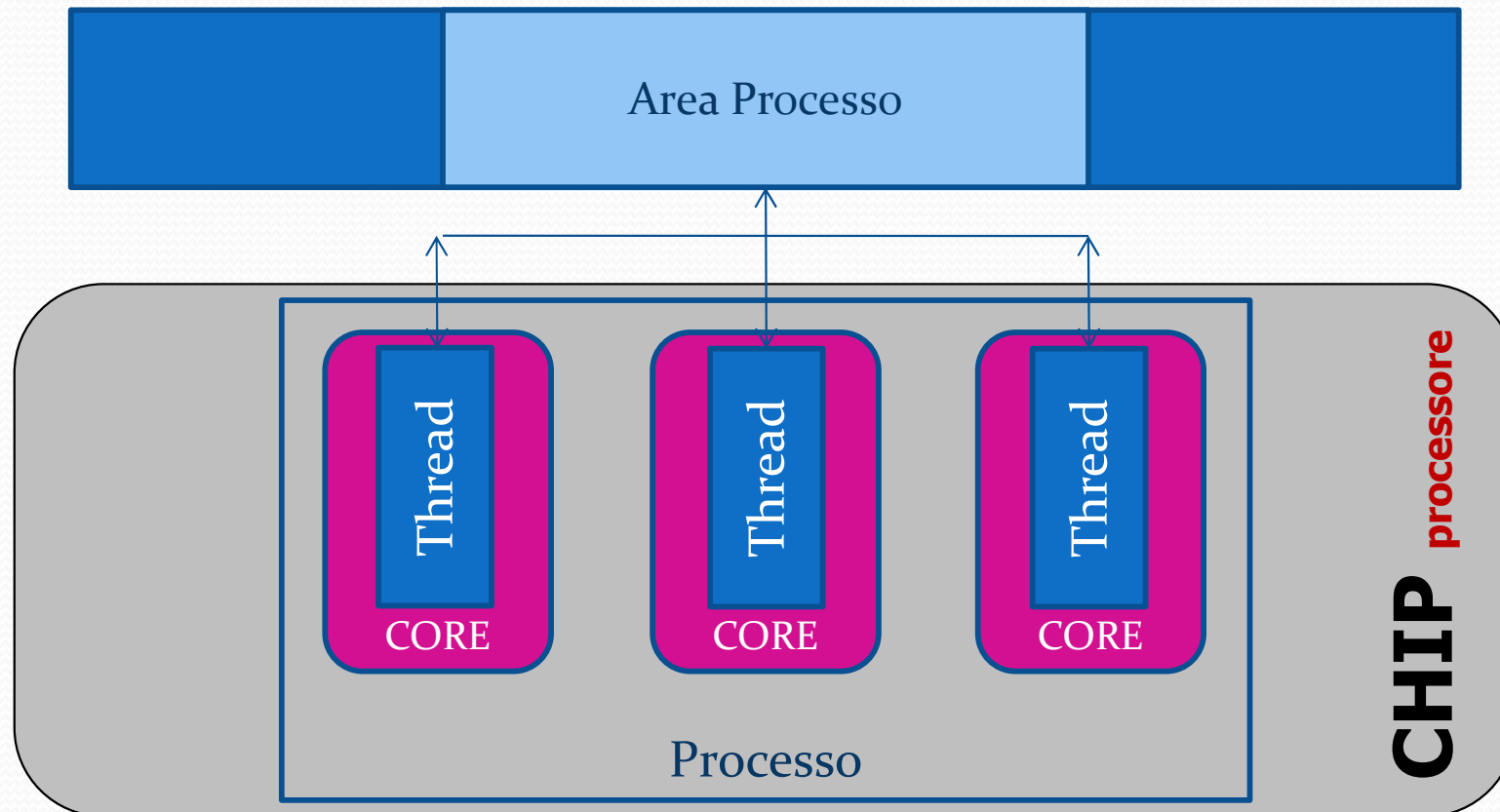




# Architettura multicore

Vantaggio: leggerezza ed efficienza

Svantaggio: i dati non sono protetti, è necessaria **sincronizzazione**

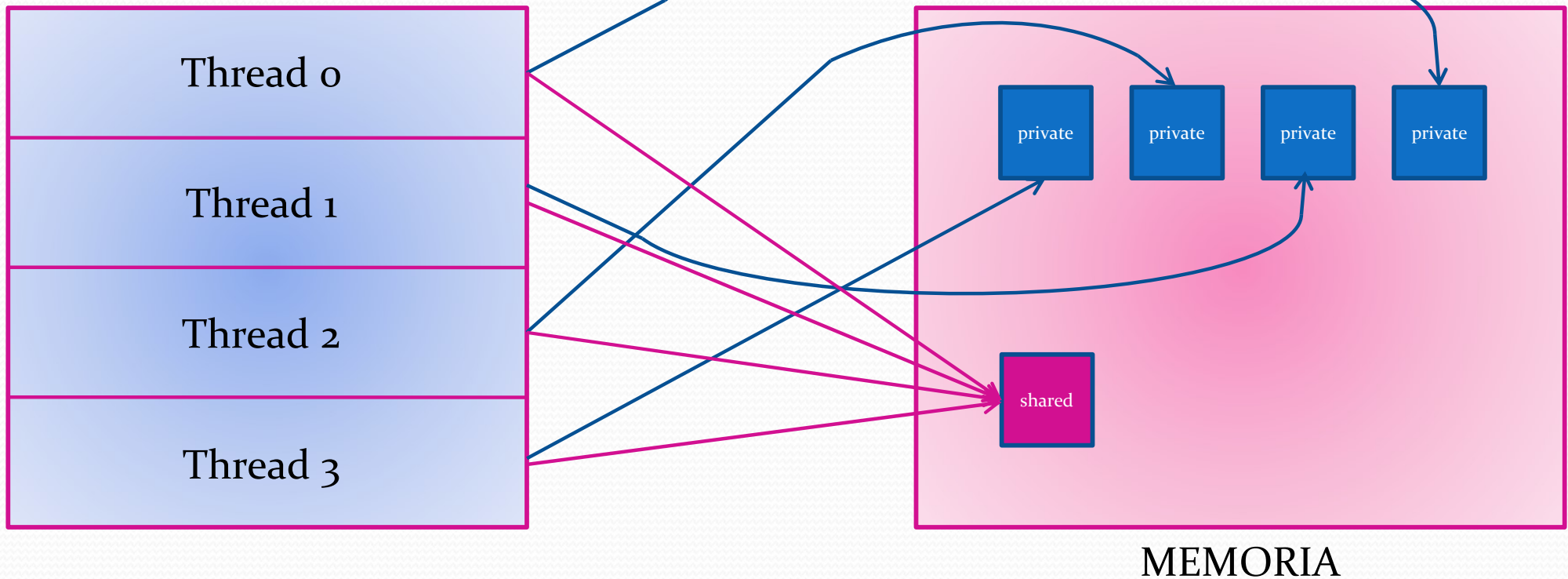





# Threads

I thread vengono coordinati attraverso la **sincronizzazione** degli accessi alle variabili condivise.

Processo multi-thread





Esistono diversi strumenti per lo sviluppo di software in ambiente di calcolo MIMD-Shared Memory

OpenMp, Pthreads,  
Windows threads...



# Introduzione ad OpenMp

## Open specifications for Multi Processing

- Prevede Application Program Interface (API) per gestire il parallelismo shared-memory multi-threaded
- Consente un approccio ad alto livello, user-friendly
- Portabile: Fortran e C/C++, Unix/Linux e Windows

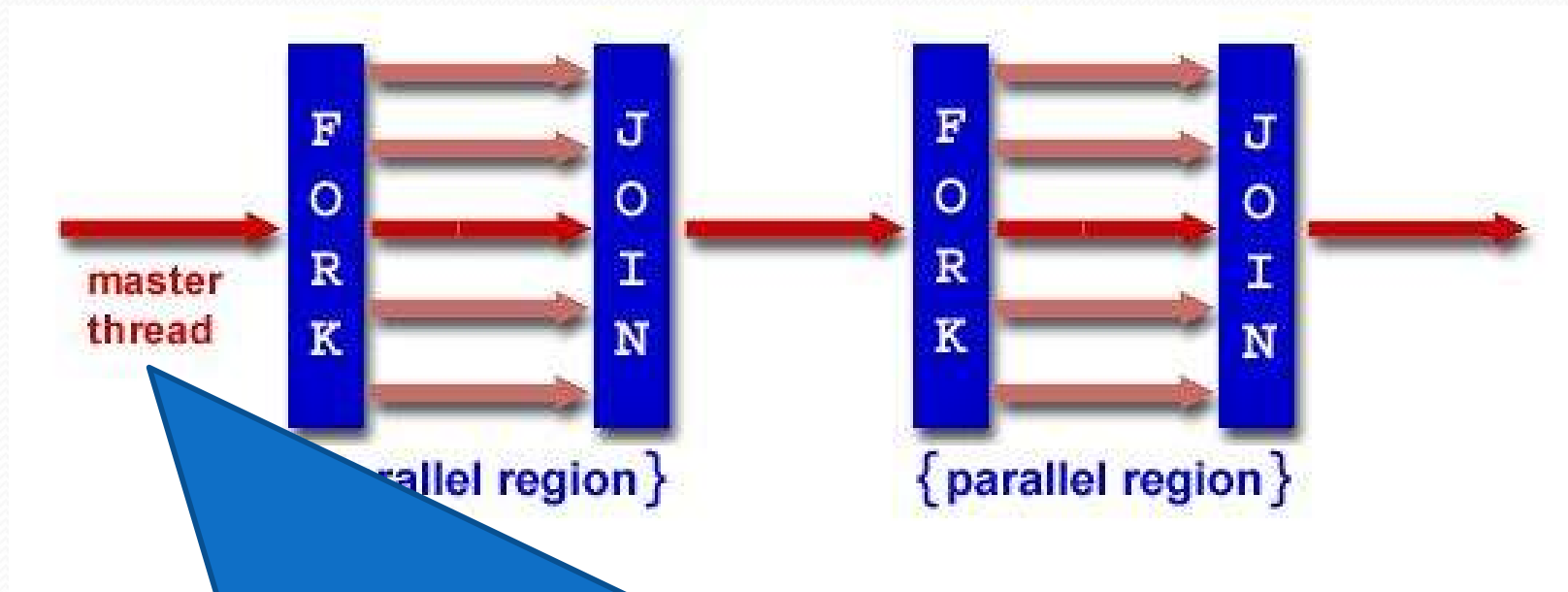
Facile  
trasformare  
un codice  
sequenziale  
in parallelo





# Introduzione ad OpenMp

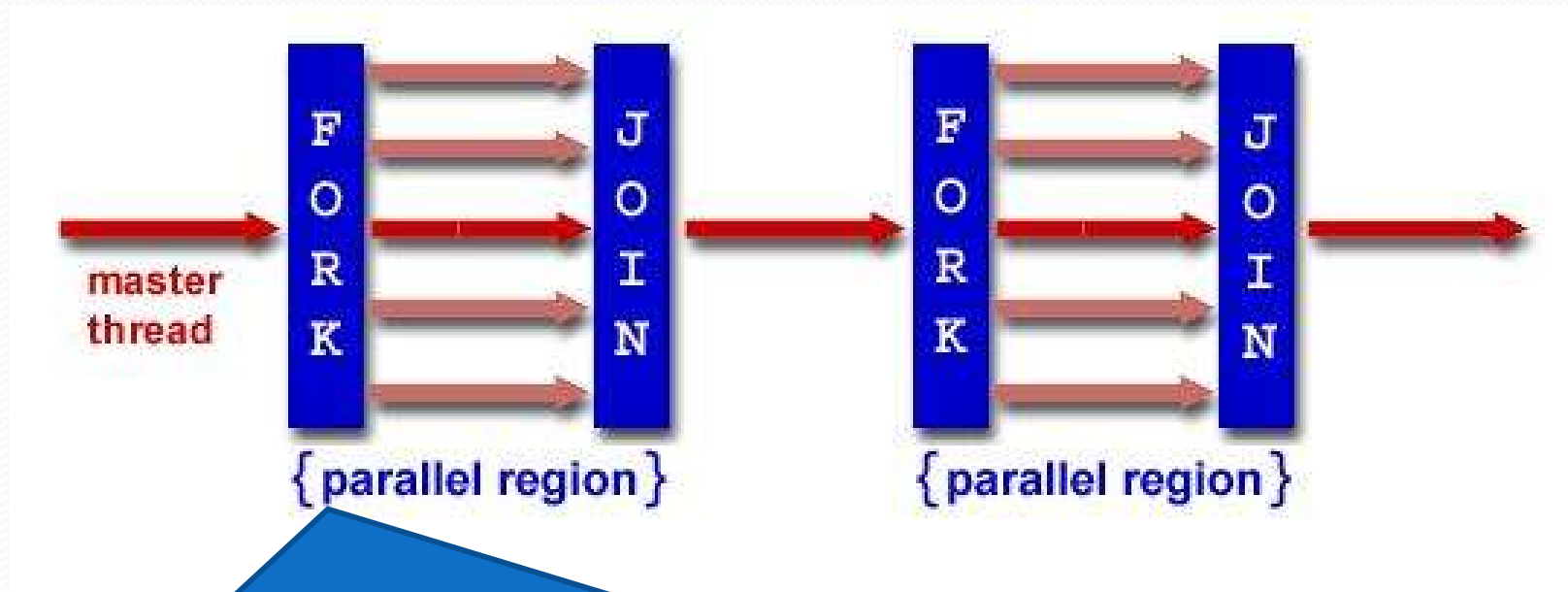
- Il modello d'esecuzione parallela è quello ***fork-join***



Tutti i processi cominciano con un solo thread (*master thread*) che esegue in maniera sequenziale

# Introduzione ad OpenMp

- Il modello d'esecuzione parallela è quello ***fork-join***

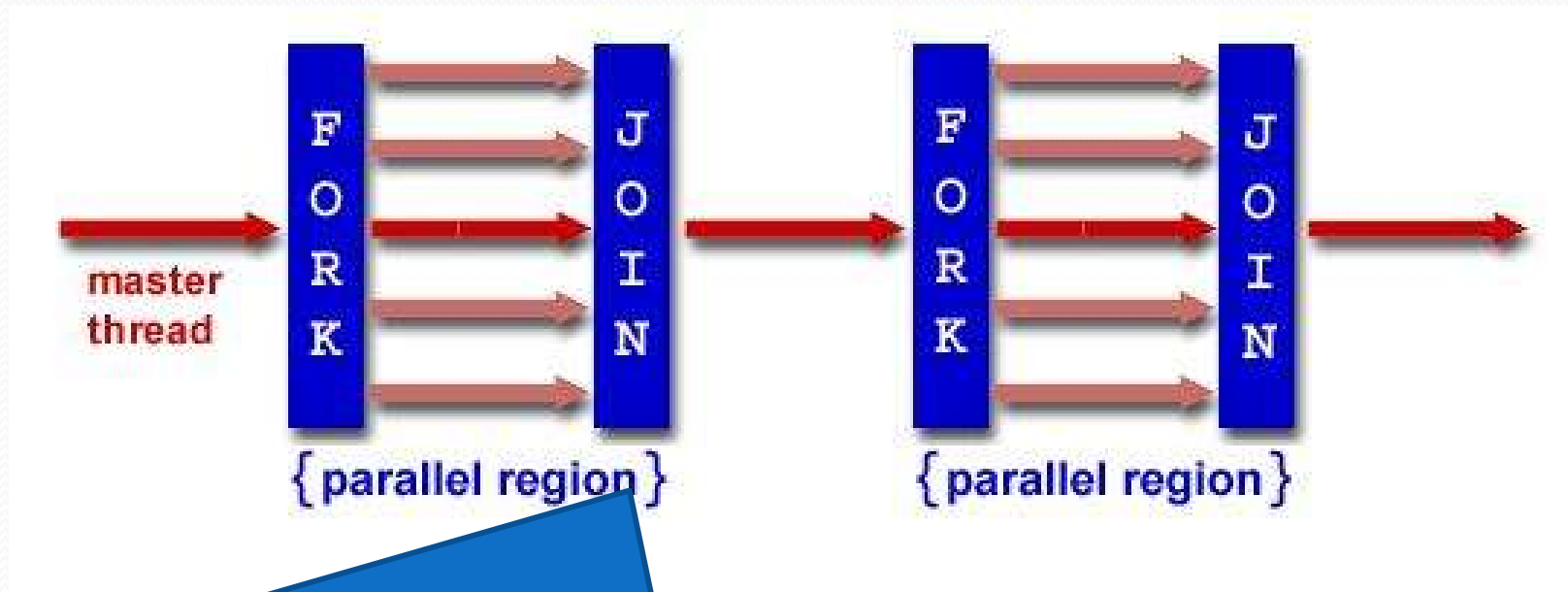


Fork: comincia una *regione parallela*, viene quindi creato un team di thread che procede parallelamente



# Introduzione ad OpenMp

- Il modello d'esecuzione parallela è quello ***fork-join***



**Join:** tutti i thread del team hanno terminato le istruzioni della regione parallela, si sincronizzano e terminano, lasciando proseguire solo il master thread.



# Introduzione ad OpenMp

- I thread portano i propri dati in **cache**



Non è garantita automaticamente e costantemente la **consistenza** della memoria



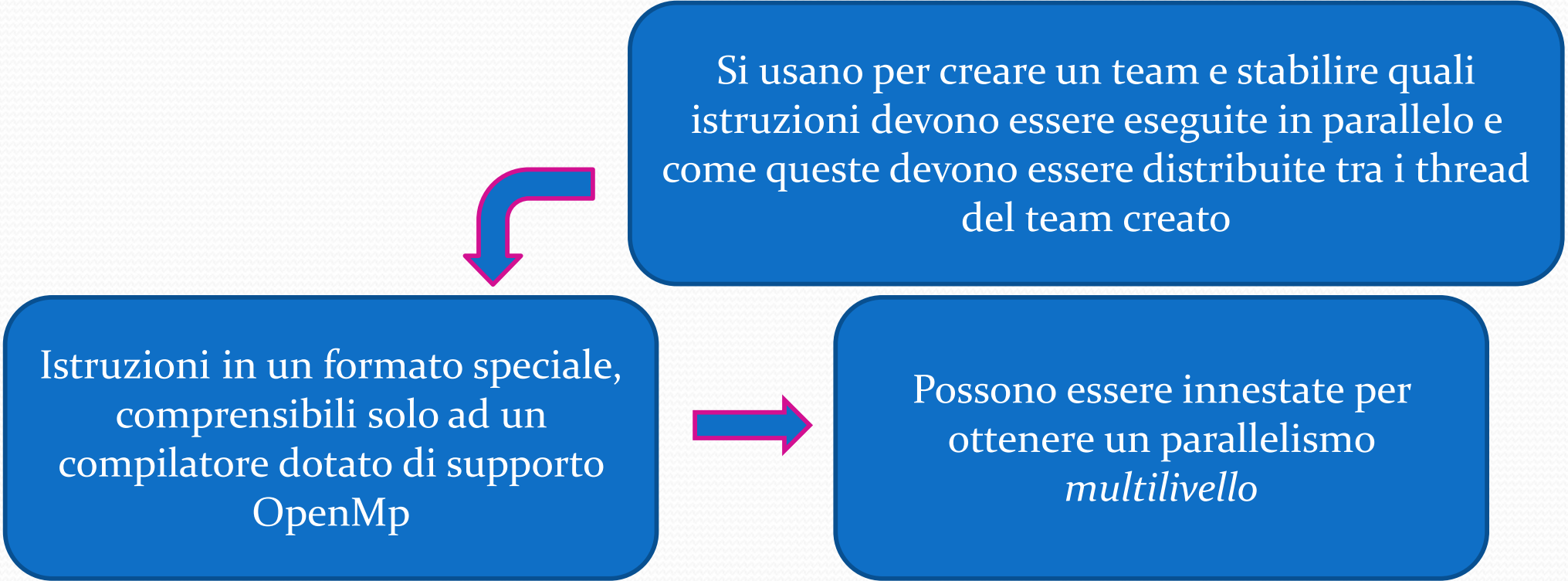
Molta attenzione alla scelta delle variabili **condivise/private**

(non ci sono regole automatiche per la gestione della memoria)

# La libreria OpenMp:

è fondamentalmente composto da:

- **Direttive per il compilatore**



Si usano per creare un team e stabilire quali istruzioni devono essere eseguite in parallelo e come queste devono essere distribuite tra i thread del team creato

Istruzioni in un formato speciale, comprensibili solo ad un compilatore dotato di supporto OpenMp

Possono essere innestate per ottenere un parallelismo *multilivello*



# Struttura generica del codice

```
main ()  
{
```

```
    int var1, var2, var3;
```

```
    ...
```

```
    Parte che deve rimanere sequenziale
```

```
    ...
```

```
    ...
```

```
    ...
```

```
    ...
```

```
    ...
```

```
    Sezione che può essere eseguita concorrentemente da più thread
```

```
    ...
```

```
    ...
```

```
    ...
```

```
    Parte che deve rimanere sequenziale
```

```
    ...
```

```
}
```

... introdurre nelle zone concorrenti le opportune direttive OpenMp, con le relative clausole



# Struttura generica del codice

```
#include <omp.h>
main ()
{
    int var1, var2, var3;
    ...
    Parte sequenziale
    ...
    Inizio della regione parallela:.
    si genera un team di thread e si specifica lo scopo delle variabili

    #pragma omp parallel private(var1, var 2) shared(var3)
    {
        Sezione parallela eseguita da tutti i thread
        ...
        Tutti i thread confluiscono nel master thread
    }

    Ripresa del codice sequenziale
    ...
}
```

... si introducono le opportune direttive OpenMp, con le relative clausole

# La libreria OpenMp

è fondamentalmente composto da:

- **Direttive per il compilatore**
  - Completate eventualmente da **clausole** che ne dettagliano il comportamento



# Direttive

## La direttiva

Si usano per creare un team e stabilire quali istruzioni devono essere eseguite in parallelo e come queste devono essere distribuite tra i thread del team creato

```
#pragma omp [clause], [clause] ...
```

prevede:

- il costrutto *parallel*, che forma un team di thread ed avvia così un'esecuzione parallela

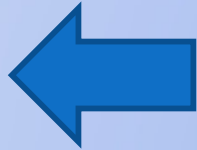
```
#pragma omp parallel [clause], [clause] ..  
{  
    structured-block  
}
```



# Direttive

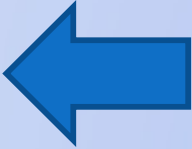
- Alla fine del blocco di istruzioni è sottintesa una barriera di sincronizzazione: tutti i thread si fermano ad aspettare che tutti gli altri abbiano completato l'esecuzione, prima di ritornare ad una esecuzione sequenziale.

```
#pragma omp parallel [clause], [clause] ...  
{  
    structured-block  
}
```



# Direttive

- Alla fine del blocco di istruzioni è sottintesa una barriera di sincronizzazione: tutti i thread si fermano ad aspettare che tutti gli altri abbiano completato l'esecuzione, prima di ritornare ad una esecuzione sequenziale.
- Tutto quello che segue questa direttiva verrà eseguito da ogni thread

```
#pragma omp parallel [clause], [clause] ...  
{  
      
}
```



# Direttive

- Alla fine del blocco di istruzioni è sottintesa una barriera di sincronizzazione: tutti i thread si fermano ad aspettare che tutti gli altri abbiano completato l'esecuzione, prima di ritornare ad una esecuzione sequenziale.
- Tutto quello che segue questa direttiva verrà eseguito da ogni thread
- Dopo aver generato i thread, è necessario stabilire anche la distribuzione del lavoro tra i thread del team, per evitare ridondanze inutili e/o dannose, per fare ciò....



# Direttive

Le direttive sono completate eventualmente da clausole che ne dettagliano il comportamento

Non è  
necessario  
un  
particolare  
ordine  
delle  
clausole

```
#pragma omp parallel [clause], [clause] ...  
{  
}  
clause:  
  num_threads(integer-expression)  
  default(shared | none)  
  private(list)  
  firstprivate(list)  
  shared(list)  
  reduction(operator: list)
```

specifica delle  
clausole

# Clausole

- Non tutte le clausole sono valide per tutti i costrutti
- Quasi tutte accettano una lista di argomenti separati da virgole

default

```
#pragma omp parallel [clause], [clause] ...  
{  
}  
clause:  
  num_threads(integer-expression)  
  default(shared | none)  
  private(list)  
  firstprivate(list)  
  shared(list)  
  reduction(operator: list)
```



# Clausole

- *default(shared|none)*: controlla gli attributi di data-sharing delle variabili in un costrutto

**shared**: tutte le variabili saranno considerate condivise

**none**: deciderà tutto il programmatore



# Clausole

- Non tutte le clausole sono valide per tutte le direttive
- Quasi tutte accettano una lista di argomenti separati da virgole

```
#pragma omp parallel [clause], [clause] ...
```

```
{
```

```
}
```

```
clause:
```

```
num_threads(integer-expression)
```

```
default(shared | none)
```

```
private(list)
```

```
firstprivate(list)
```

```
shared(list)
```


```
reduction(operator: list)
```

shared

private

# Clausole

- *default(shared|none)*: controlla gli attributi di data-sharing delle variabili in un costrutto
- *shared(list)*: gli argomenti contenuti in *list* sono condivisi tra i thread del team
- *private(list)*: gli argomenti contenuti in *list* sono privati per ogni thread che li utilizza



Ogni thread avrà la propria copia delle variabili private



# Clausole

- Non tutte le clausole sono valide per tutte le direttive
- Quasi tutte accettano una lista di argomenti separati da virgole

```
#pragma omp parallel [clause], [clause] ...  
{  
}  
clause:  
num_threads(integer-expression)  
default(shared | none)  
private(list)  
firstprivate(list)  
shared(list)  
reduction(operator: list)
```

firstprivate

# Clausole

- *default(shared|none)*: controlla gli attributi di data-sharing delle variabili in un costrutto
- *shared(list)*: gli argomenti contenuti in *list* sono condivisi tra i thread del team
- *private(list)*: gli argomenti contenuti in *list* sono privati per ogni thread che li utilizza, hanno valore indefinito
- *firstprivate(list)*: gli argomenti contenuti in *list* sono privati per i thread e vengono inizializzati con il valore che avevano gli originali al momento in cui è stato incontrato il costrutto in questione

All'ingresso le copie private sono pre-inizializzate con il valore che ha la variabile con lo stesso nome prima di incontrare la regione parallela



# Clausole

- *default(shared|none)*: controlla gli attributi di data-sharing delle variabili in un costrutto
- *shared(list)*: gli argomenti contenuti in *list* sono condivisi tra i thread del team
- *private(list)*: gli argomenti contenuti in *list* sono privati per ogni thread
- *firstprivate*: All'uscita le variabili manterranno come valore l'ultimo degli argomenti privati per i thread della sezione parallela
- *lastprivate(list)*: gli argomenti contenuti in *list* sono privati per i thread e quelli originali verranno aggiornati al termine della regione parallela

**Attenzione:** *firstprivate* si usa con **parallel**, invece *lastprivate* si usa solo con un altro costrutto il **for** (che vedremo più avanti)

# Clausole


- Non tutte le clausole sono valide per tutte le direttive
- Quasi tutte accettano una lista di argomenti separati da virgole

```
#pragma omp parallel [clause], [clause] ...new-line  
{  
}  
clause: if(scalar-expression)  
num_threads(integer-expression)  
default(shared | none)  
private(list)  
firstprivate(list)  
shared(list)  
reduction(operator: list)
```

reduction



# Clausole

- *default(shared|none)*: controlla gli attributi di data-sharing delle variabili in un costrutto
- *shared(list)*: gli argomenti contenuti in *list* sono condivisi tra i thread. L'ordine di esecuzione dei thread non è specificato ... quindi ATTENZIONE con i valori f.p.! Risultati numericamente non determinati. 
- *lastprivate(list)*: gli argomenti contenuti in *list* sono privati per ogni thread. Gli argomenti non sono inizializzati con il valore che avevano gli argomenti originali. Ogni thread avrà una copia privata delle variabili in *list*, e al termine del costrutto la variabile sarà condivisa.
- *reduction(operator:list)*: gli argomenti contenuti in *list* verranno combinati utilizzando l'operatore associativo specificato.

# Clausole

- Non tutte le clausole sono valide per tutte le direttive
- Quasi tutte accettano una lista di argomenti separati da virgole

Setta il numero dei thread che lavoreranno nella direttiva definita

```
#pragma omp parallel [clause], [clause] ...new-line
{
}
clause: if(scalar-expression)
num_threads(integer-expression)
default(shared | none)
private(list)
firstprivate(list)
shared(list)
copyin(list)
reduction(operator: list)
```



# La libreria OpenMp

è fondamentalmente composto da:

- **Direttive per il compilatore**
  - Completate eventualmente da **clausole** che ne dettagliano il comportamento
- **Runtime Library Routines**, per intervenire sulle **variabili di controllo interne** a run-time  
(deve essere incluso il file omp.h).  
Es. Numero di thread, informazioni sullo scheduling,...



per intervenire sulle variabili di  
controllo interne a run-time  
Es. Numero di thread, informazioni  
sullo scheduling,...

# Runtime Library Routines

- **omp\_set\_num\_threads(*scalar-integer-expression*)**: definisce il numero di thread da utilizzare
- **omp\_get\_max\_threads()**: restituisce il numero massimo di thread disponibili per la prossima regione parallela
- **omp\_set\_dynamic(*scalar-integer-expression*)**: permesso (0) o meno (1) al sistema di riadattare il numero di thread utilizzati.
- **omp\_get\_thread\_num()**: restituisce l'id del thread
- **omp\_get\_num\_procs()**: restituisce il numero di core disponibili per il programma al momento della chiamata
- **omp\_get\_num\_threads()**: restituisce il numero di thread del team



# La libreria OpenMp

è fondamentalmente composto da:

- **Direttive per il compilatore**
  - Completate eventualmente da **clausole** che ne dettagliano il comportamento
- **Runtime Library Routines**, per intervenire sulle **variabili di controllo interne** a run-time  
(deve essere incluso il file omp.h).  
Es. Numero di thread, informazioni sullo scheduling,...
- **Variabili d'ambiente**, per modificare il valore delle **variabili di controllo interne** prima dell'esecuzione.



per modificare il valore delle variabili  
di controllo interne prima  
dell'esecuzione  
Es. Numero di thread, informazioni  
sullo scheduling,...

## Variabili d'ambiente

- **OMP\_NUM\_THREADS**: numero di thread che verranno utilizzati nell'esecuzione/i successiva/e
  - `OMP_NUM_THREADS(integer)`
  - `sh/bash: export OMP_NUM_THREADS=integer`
- **OMP\_DYNAMIC**: permesso (true) o meno (false) al sistema di riadattare il numero di thread utilizzati



# Riferimenti bibliografici

## **Using OpenMp**

*B. Chapman, G. Jost, R. van der Pas*

The MIT Press

<http://openmp.org/>

<https://computing.llnl.gov/tutorials/openMP/>