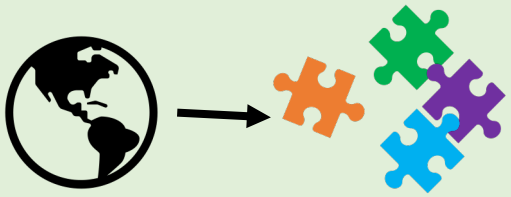
HPC



1)Algorithms

2)Architecture

Solutions

3)Shared Memory
OpenMP

4)Distributed Memory
MPI

5) Unified Engine
Spark

What is the average age in the classroom?

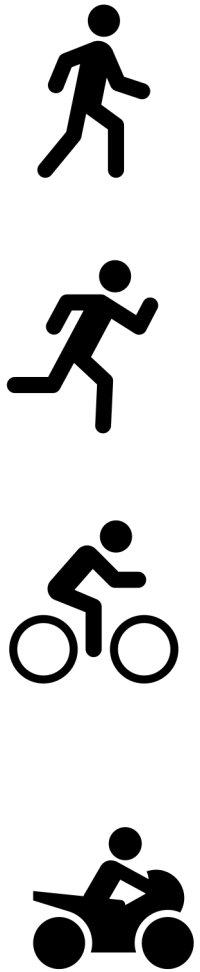
What problems did we encounter?

2)Shared Memory

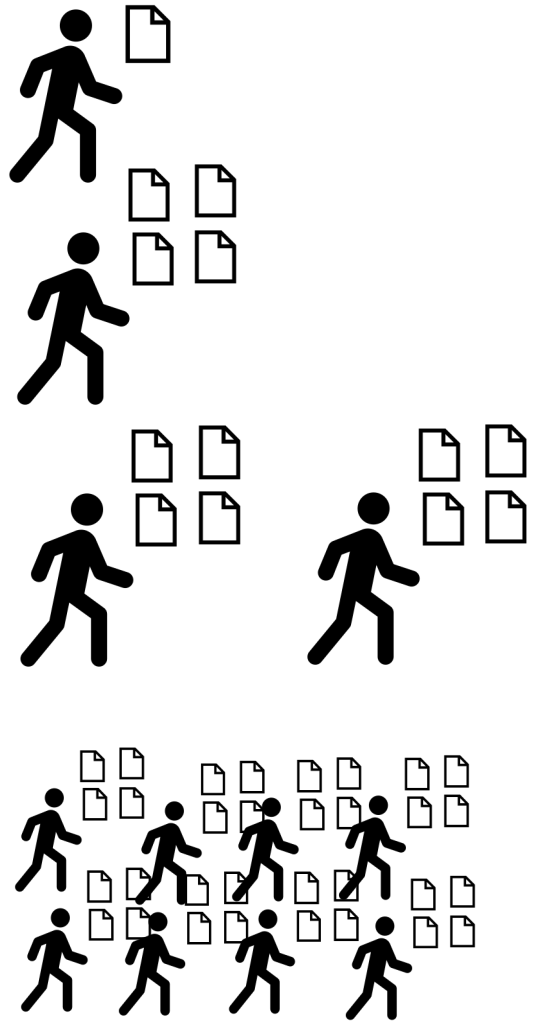
OpenMP

Communication decisions from Laptop to HPC cluster

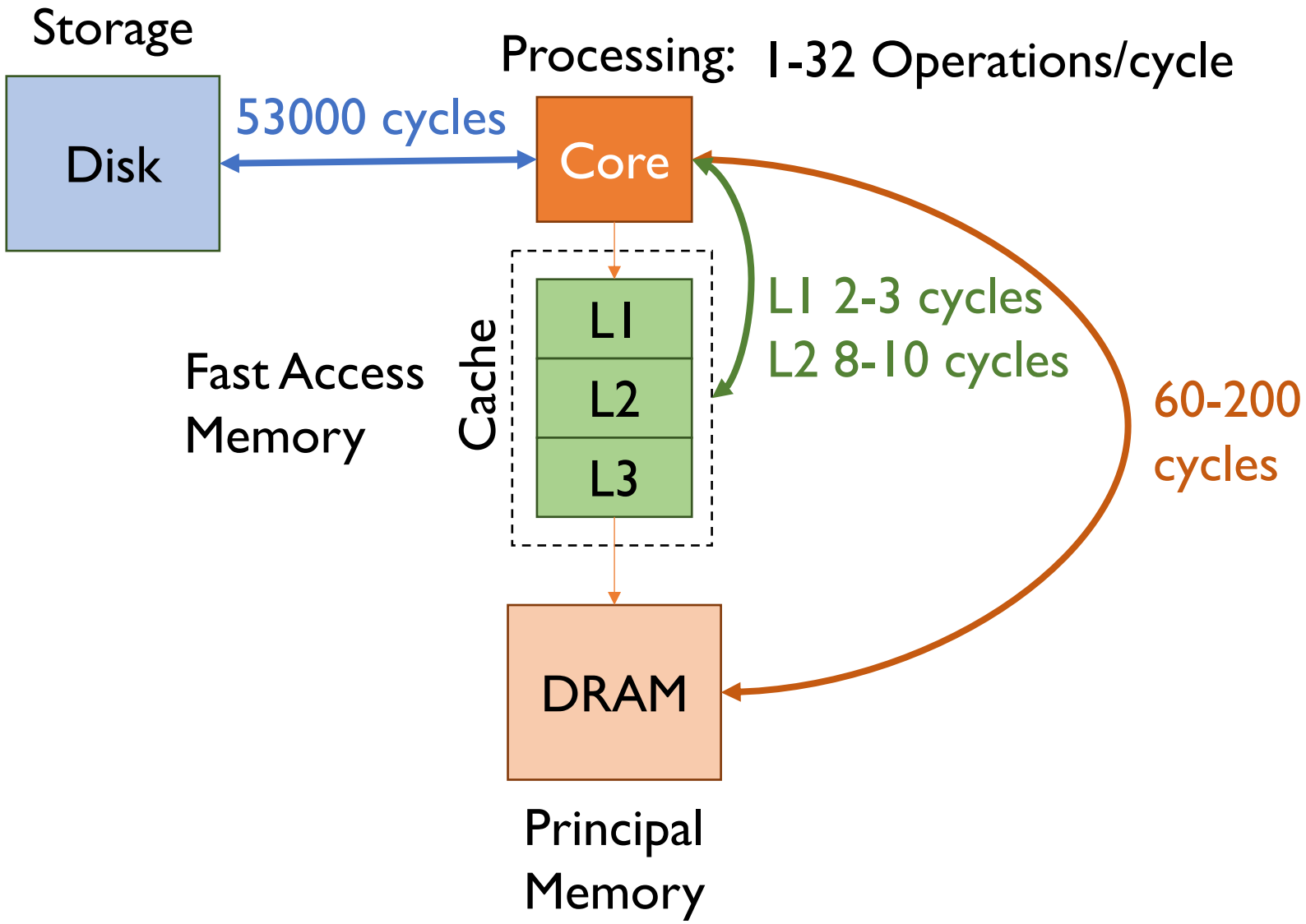
¿… ?

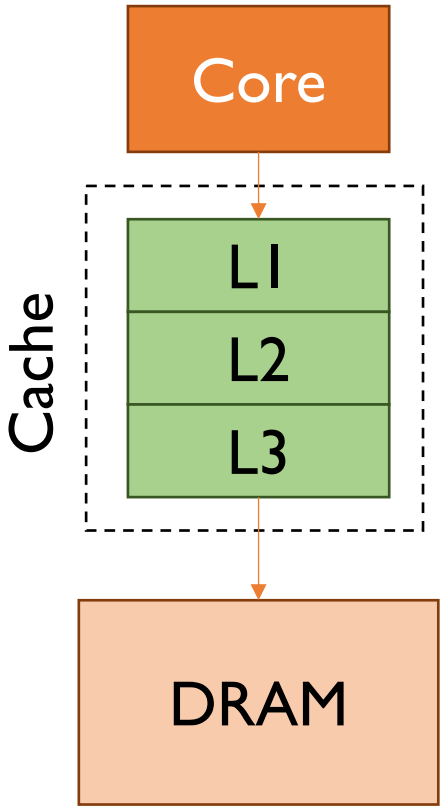


Latency vs. Throughput Bandwidth

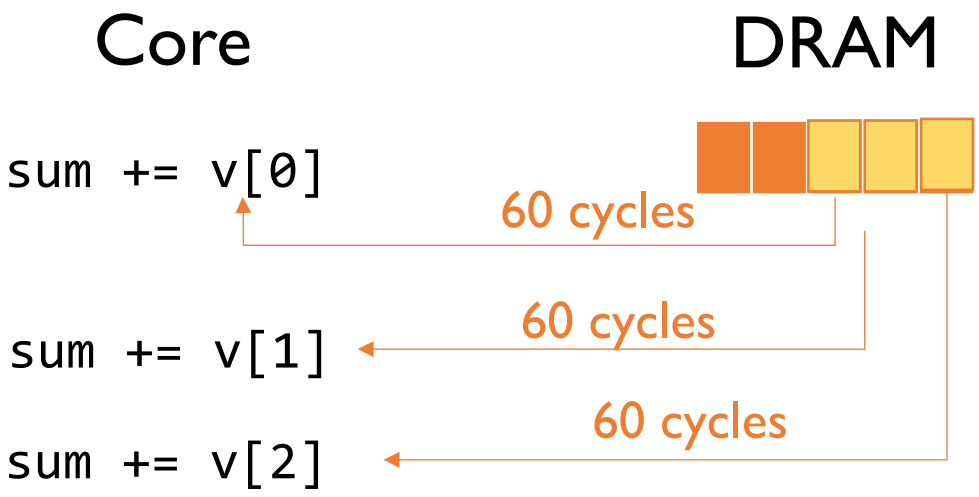


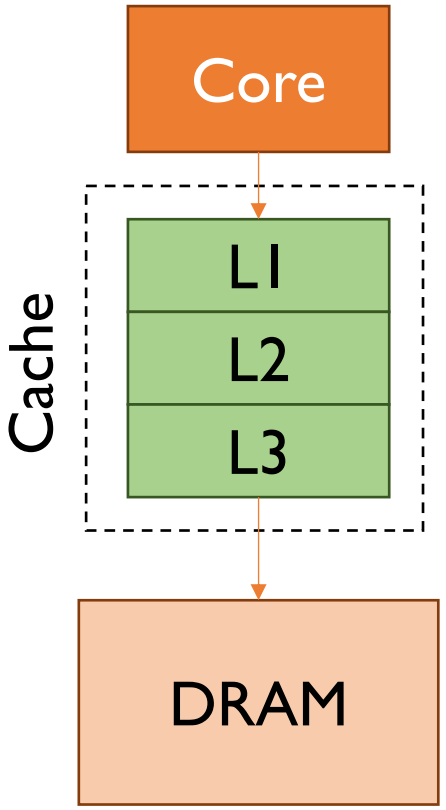
Single Processor



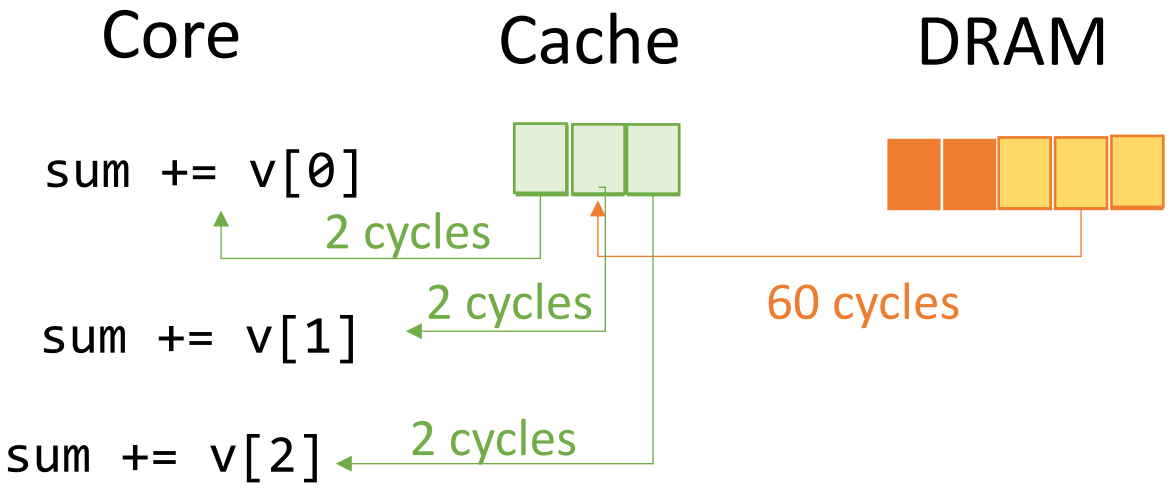


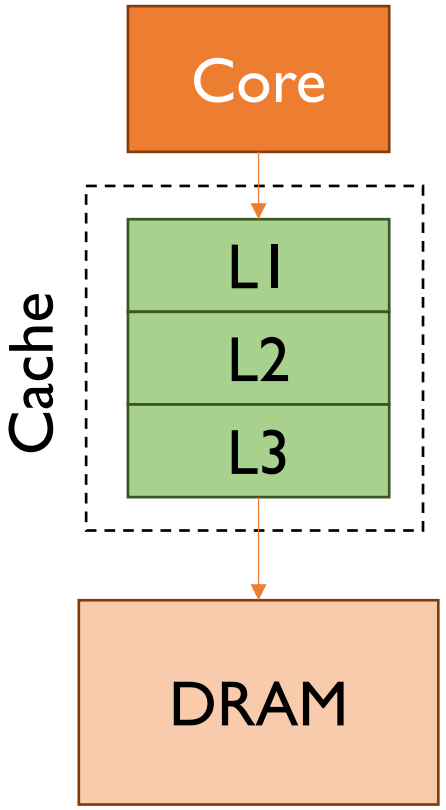

```
int sum = 0;
std::vector<int> v = {1,2,3,4,5};
…
for (int i = 0; i< v.size(); i++){
    sum += v[i];
}
```



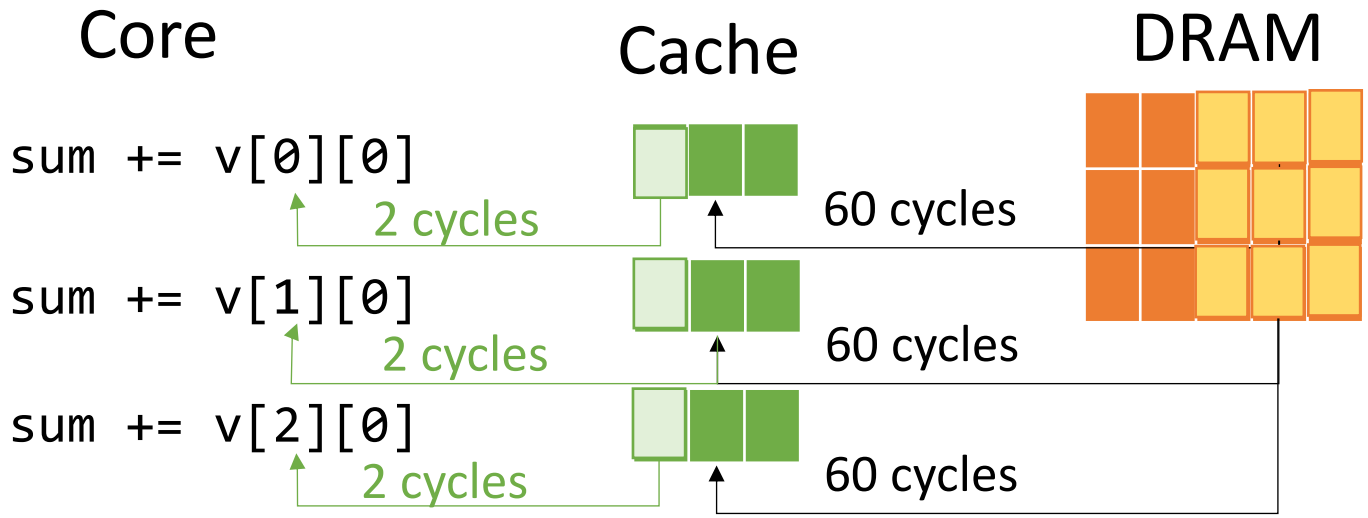


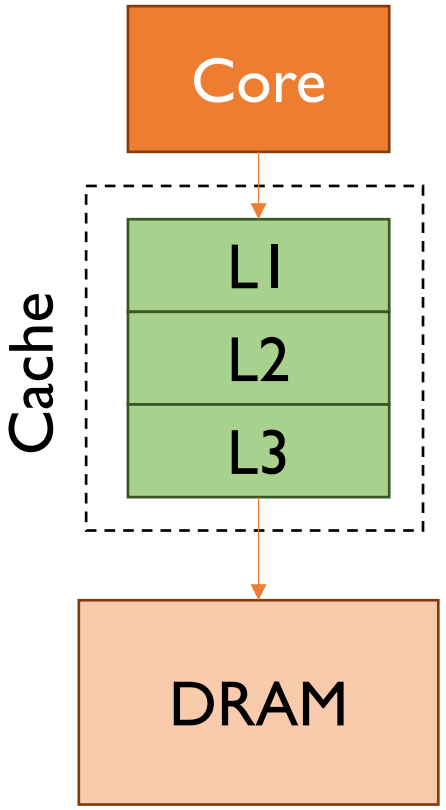

```
int sum = 0;
std::vector<int> v = {1,2,3,4,5};
…
for (int i = 0; i< v.size(); i++){
    sum += v[i];
}
```



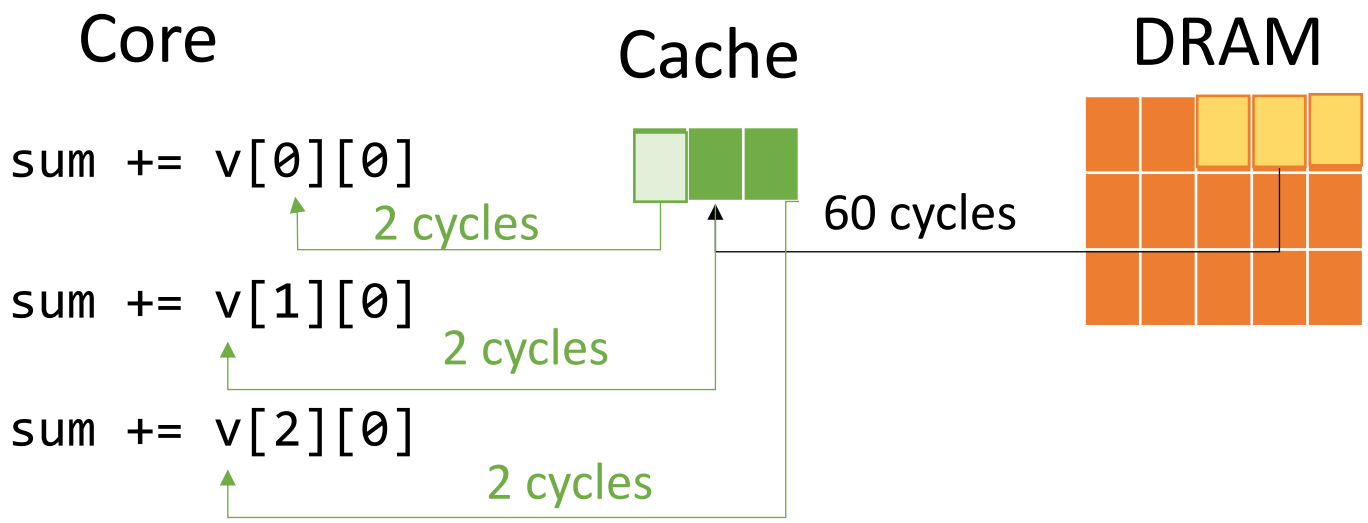



```
int sum = 0;
std::vector<std::vector<int>> v;
…
for (int j = 0; j<ncols; ++j){
    for (int i = 0; i<nrows; ++i)
        sum += v[i][j];
}
```

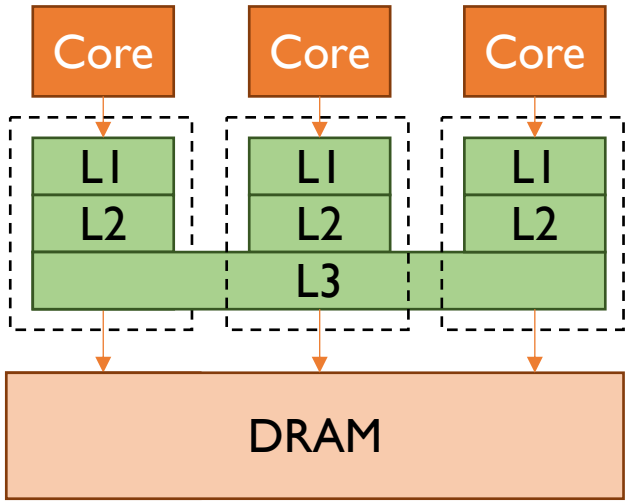


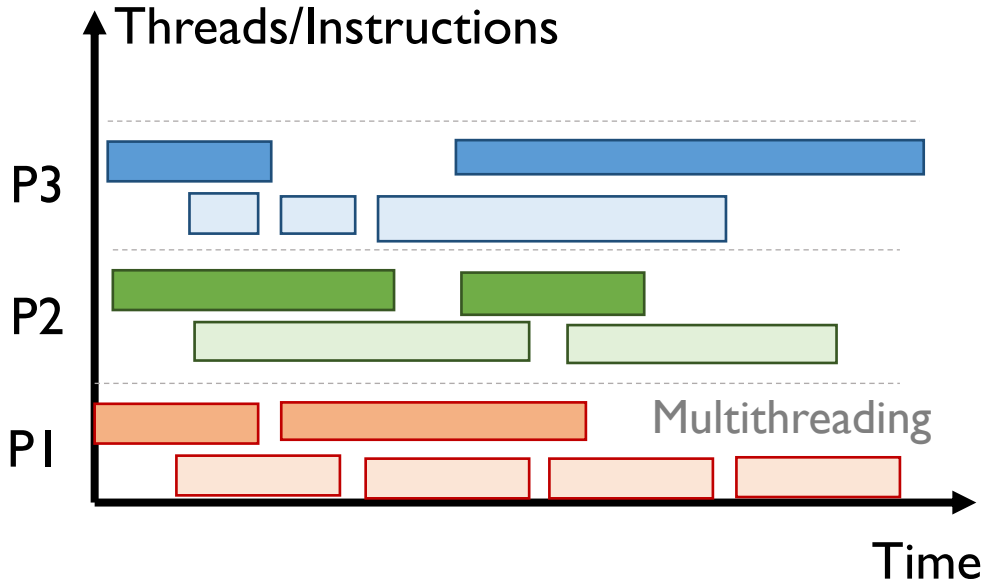



```
int sum = 0;
std::vector<std::vector<int>> v;
…
for (int i = 0; i<nrows; ++i){
    for (int j = 0; j<ncols; ++j)
        sum += v[i][j];
}
```

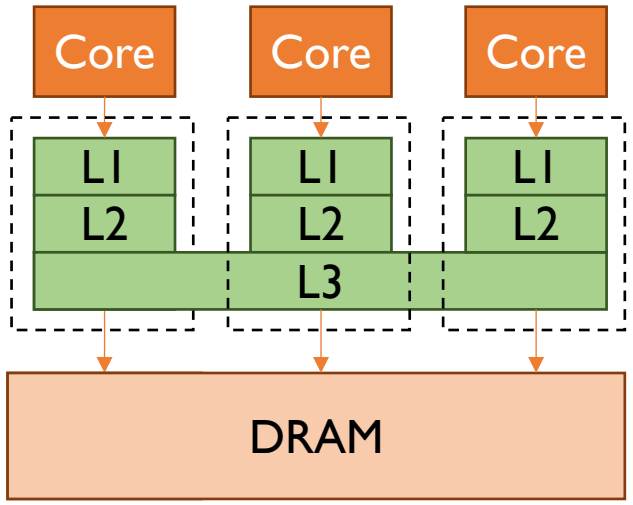


Multiple Cores



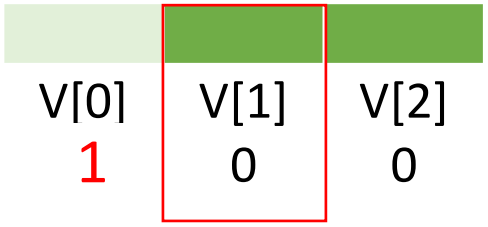


Cache Coherence



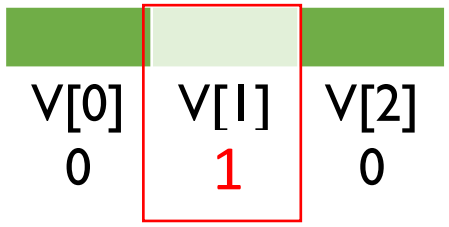
Core 1

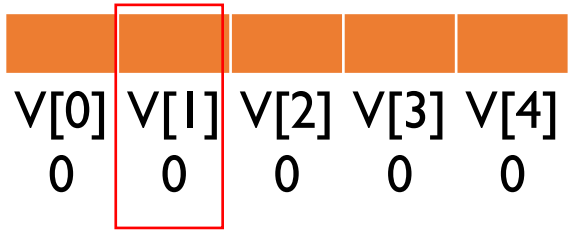
```
v[0]=1

v[2]=v[0]+v[1]
```



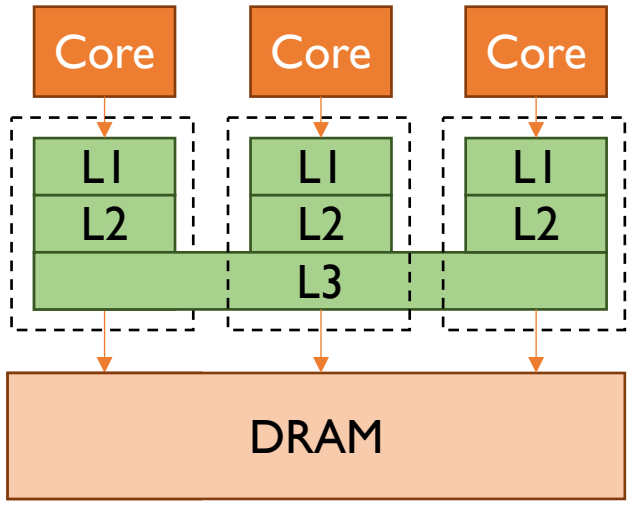
Core 2

```
v[1]=1
```



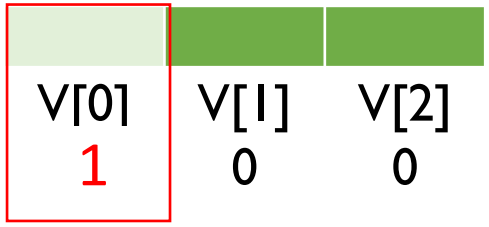


Race Condition



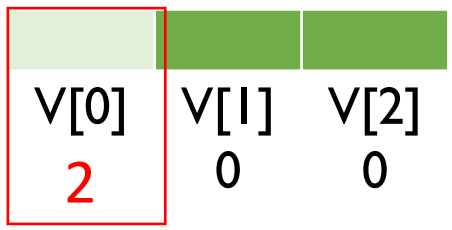
Core 1

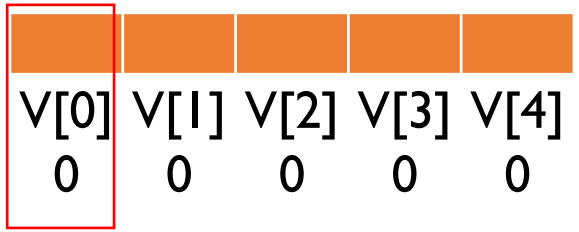
```
v[0]=1
```



Core 2

```
v[0]=2
```





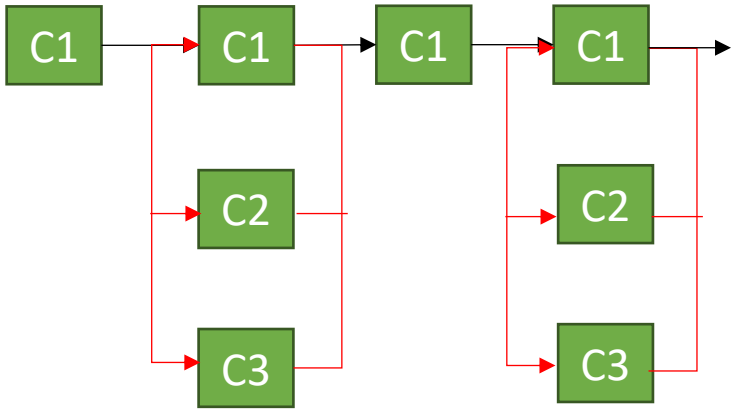
What do we need in a shared memory system?

Distribute instructions

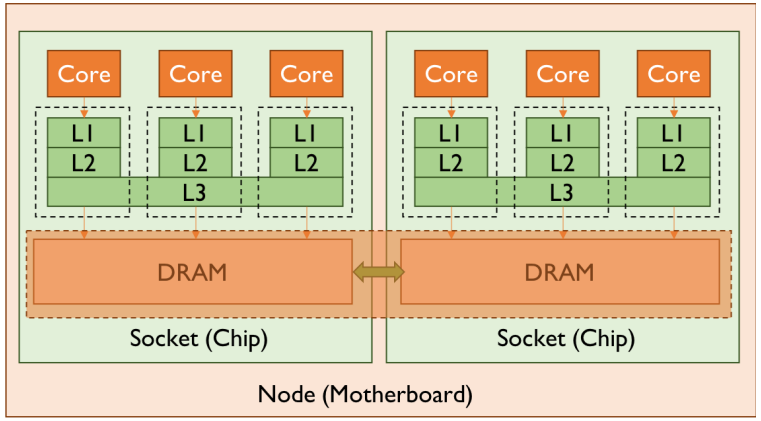
Manage memory conflicts

OpenMP

Medium granularity



Shared Memory



The simplest solution for simple architectures

Instructions

Medium Level of granularity

1. Decide when code is parallel or not. Parallel Region
2. Distribute work between threads:
   i. SISD: master/single/critical
   ii. SIMD: for / reductions
   iii. MIMD: sections/tasks
3. Synchronize workers: barrier/wait

Communication

Shared Memory

1. Avoid race condition atomic
2. Maintain cache coherence flush

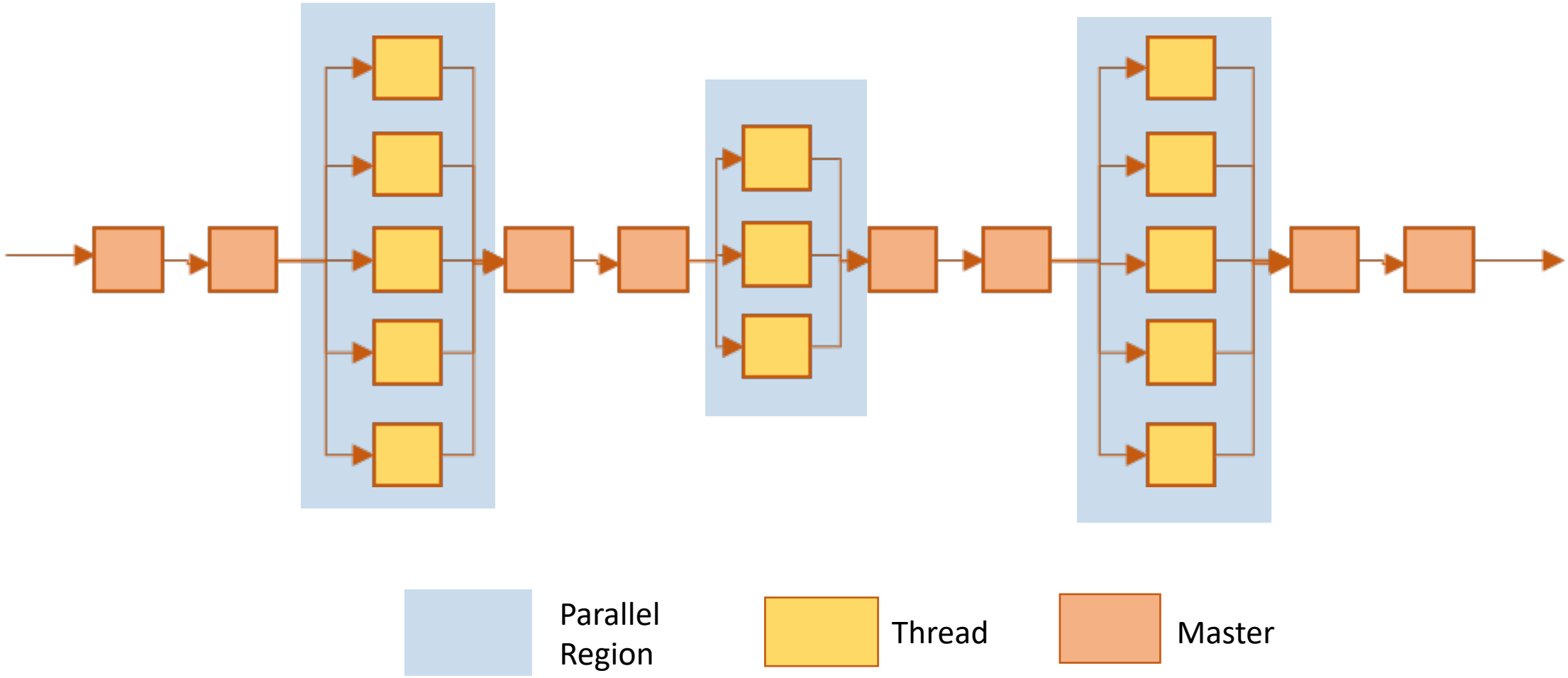
Control

1. Decide number of workers num_threads, max_threads
2. Identify workers thread_id

Done in the code (i.e. C/FORTRAN)

Parallel region `#pragma omp parallel`

Fork and join



SISD

```
#pragma omp master
```

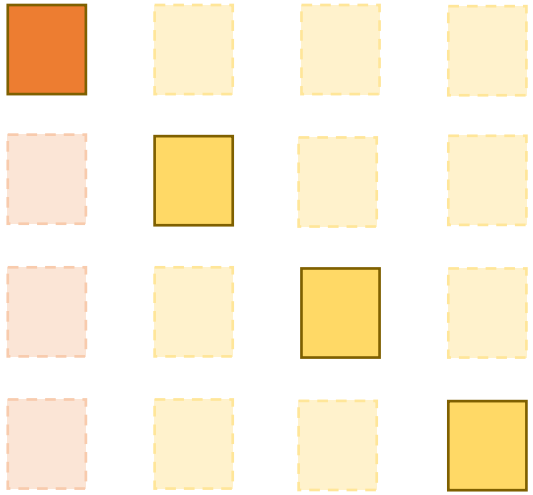
Only Thread 1

```
#pragma omp single
```

Only one Thread

```
#pragma omp critical
```

Only one thread at the time



SIMD: omp for

```
#pragma omp for …  reduction(red_identifier:list)
```

How to distribute the work	`schedule(modifier:kind,chunk_size)`

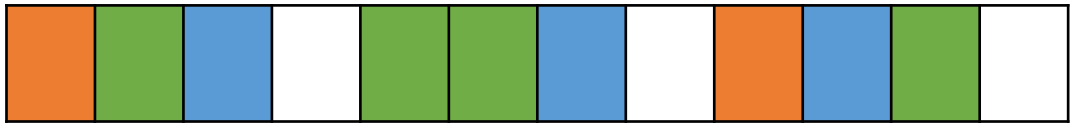
How to initialize variables	`private(list)`
	`firstprivate(list)`
	`lastprivate(list)`

Schedule

static - prefixed

dynamic

first come first served



guided

size changes at runtime

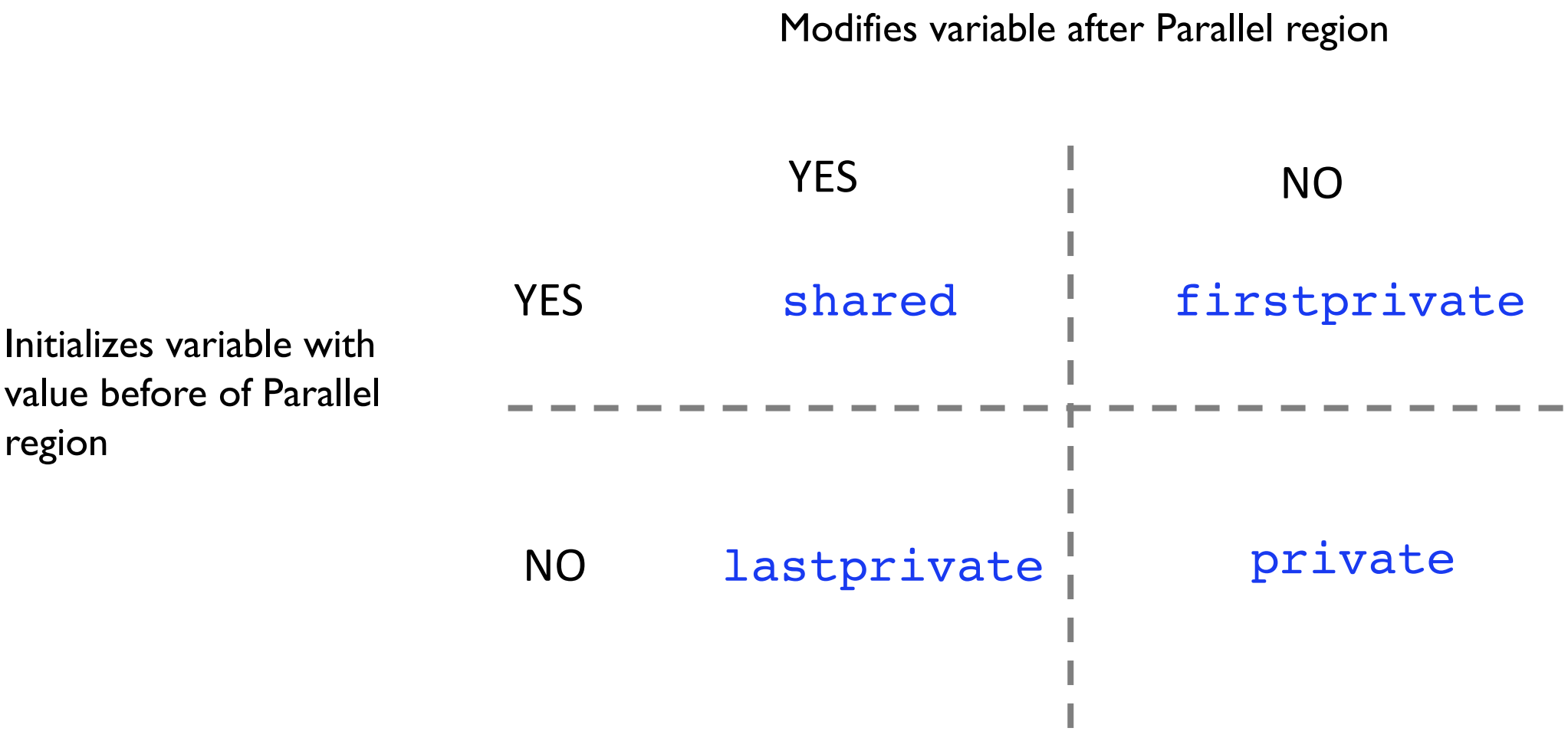
auto

runtime

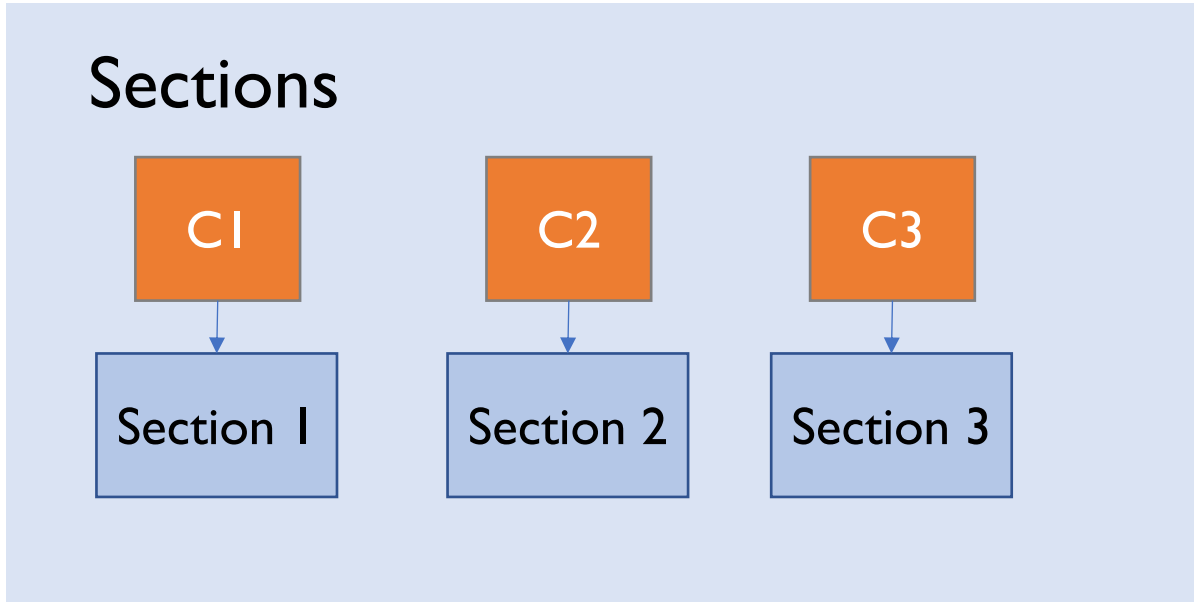
monotonic

nonmonotonic

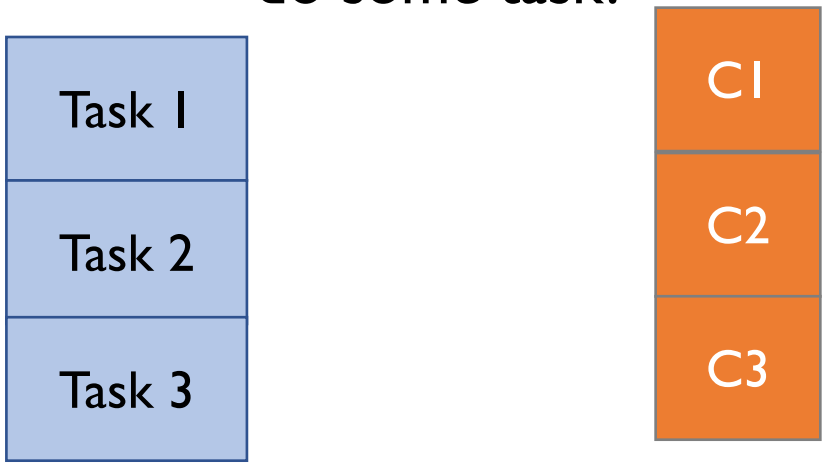
Variable type



MIMD: omp sections/omp tasks



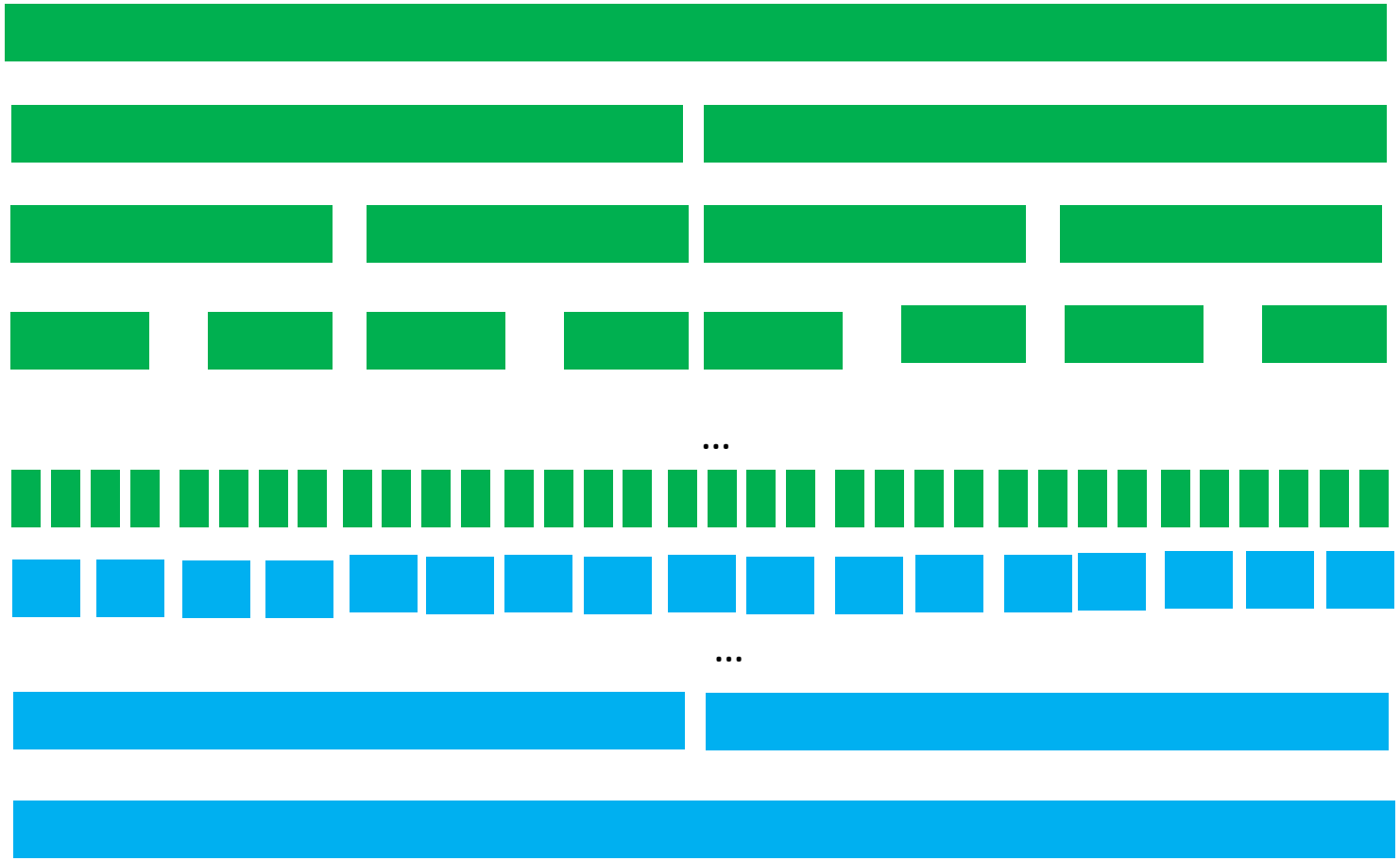
Who is free to do some task?




```
#pragma omp sections
  #pragma omp section
```

```
#pragma omp task
```

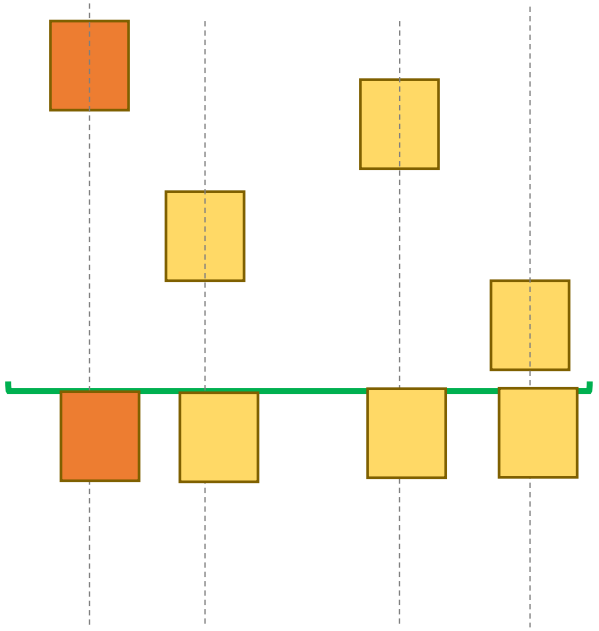
Example: Merge sort



Merge sort

```
void merge_sort(list)
  If len(list)<=1
    return list
  End
#pragma omp parallel sections
{
  #pragma omp section
  sorted_left  = merge_sort(list[0:len(list)/2])
 #pragma omp section
  sorted_right = merge_sort(list[len(list)/2:len(list)])
}
  return merge(sorted_left, sorted_right)
```

Synchronization



```
#pragma omp barrier
```

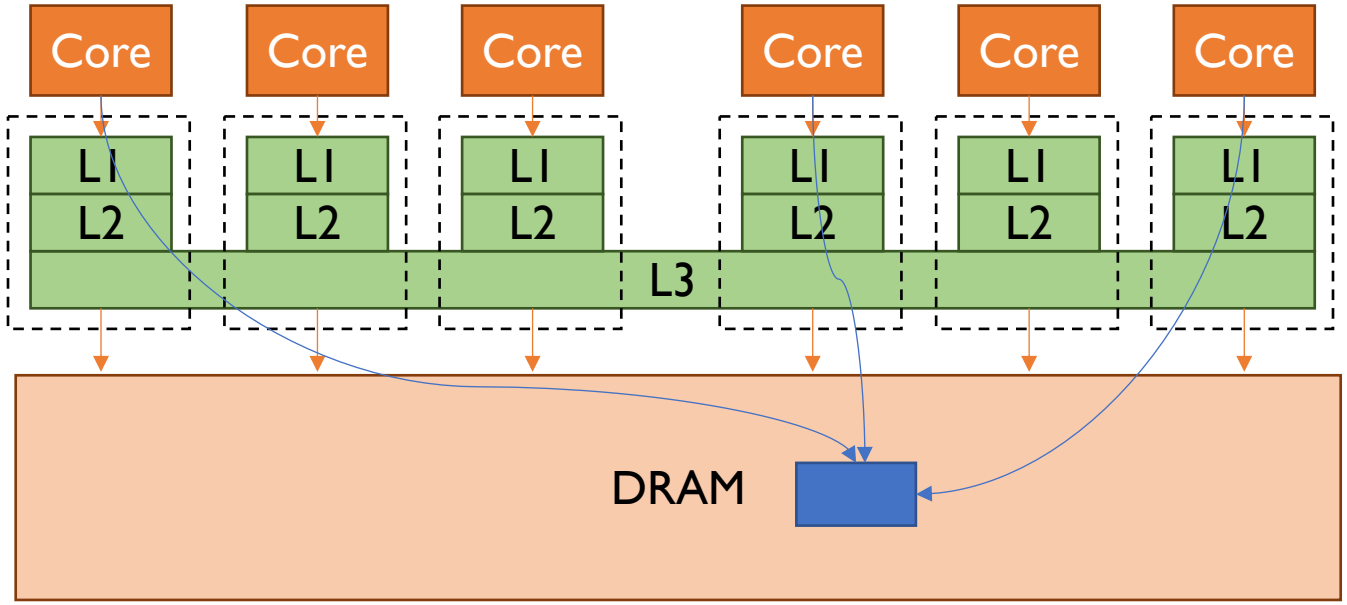
All the threads wait until they are in the point in the code

```
#pragma omp taskgroup
```

```
#pragma omp taskwait
```

All the threads wait until all the tasks from a given **taskgroup** are done

Communication: Race Condition



Core 1
v[1]=1

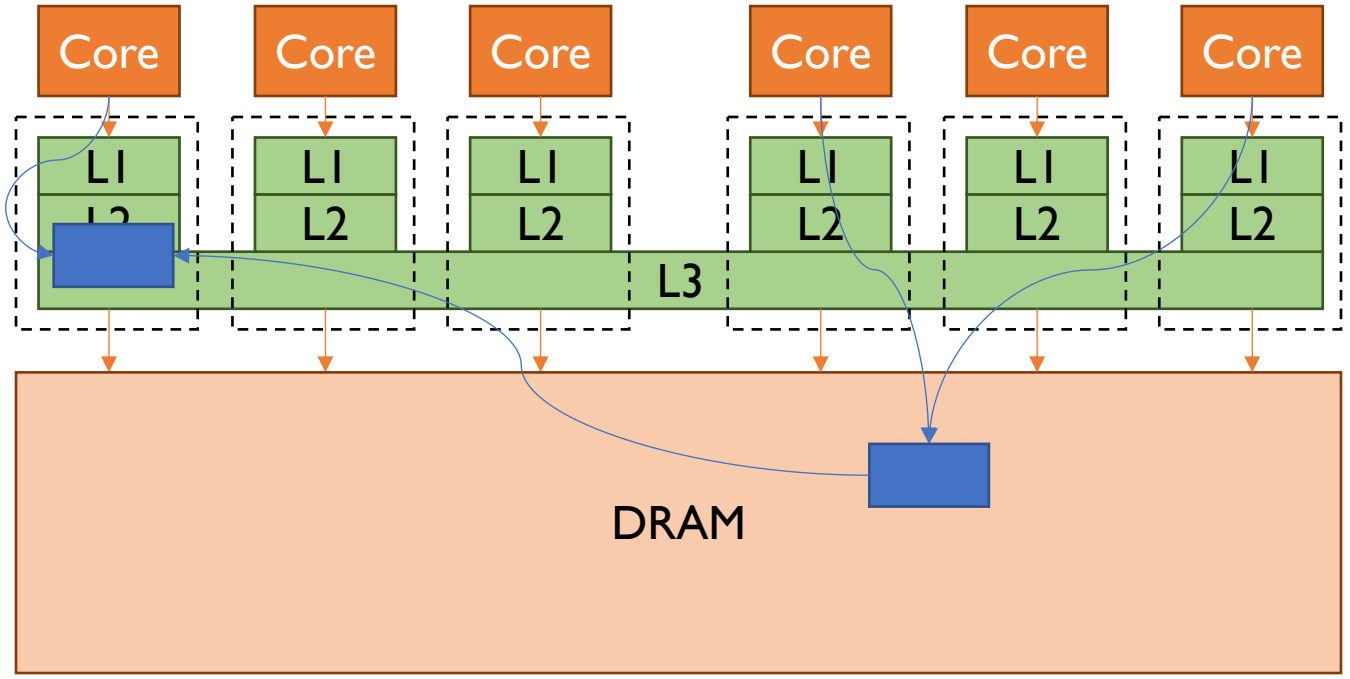
Core 4
v[1]=2

Core 6
v[1]=3

Only 1 Thread reads at the time

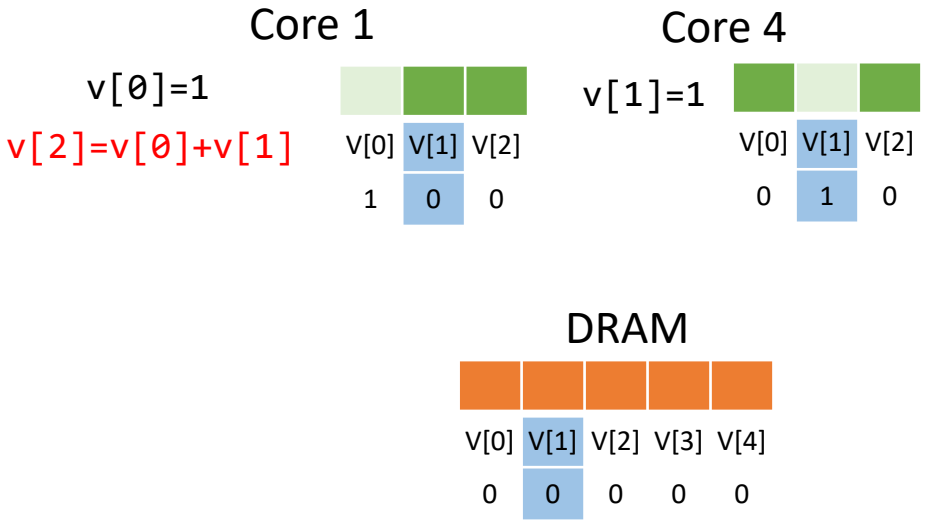
```
#pragma omp atomic update
                    read
                    write
                    capture
```

Communication: Cache Coherence



Update values in cache

```
#pragma omp flush
```



Control

Environment Variables

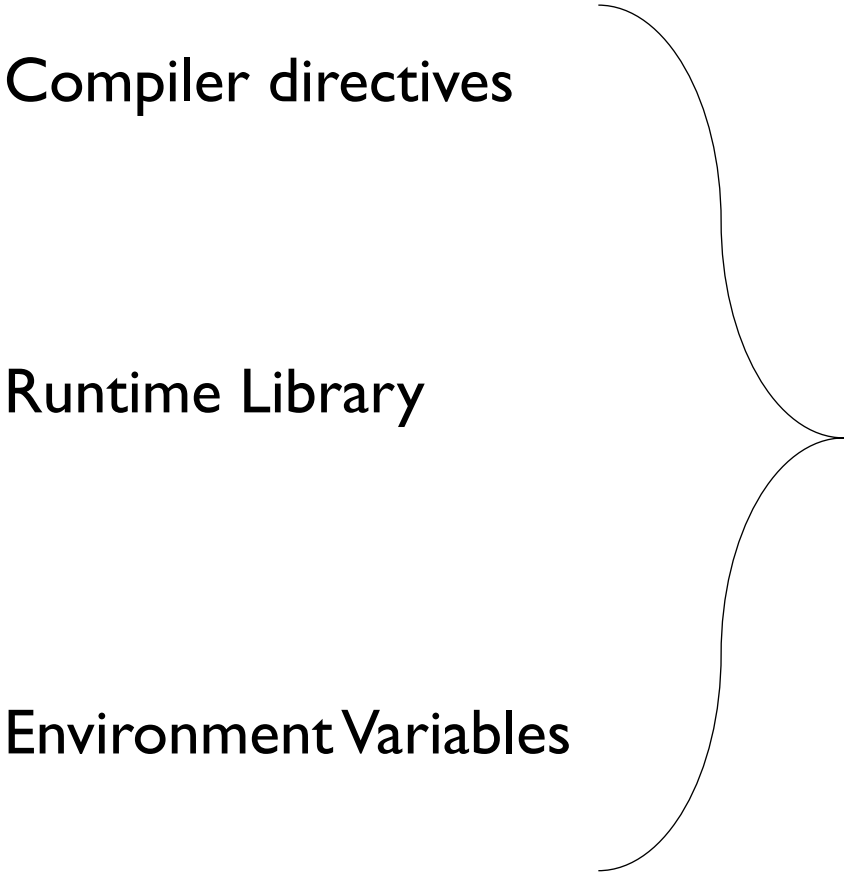
```
OMP_NUM_THREADS=
```

Runtime Library

```
omp_set_num_threads(int num_threads)
omp_get_num_threads()
omp_get_max_threads()
```

```
omp_get_thread_num()
omp_in_parallel()
```

Why the simplest solution?



Manage threads + shared memory

C/C++ & Fortran applications

Relies on compiler

Full responsibility in programmer, no correctness check

http://www.openmp.org

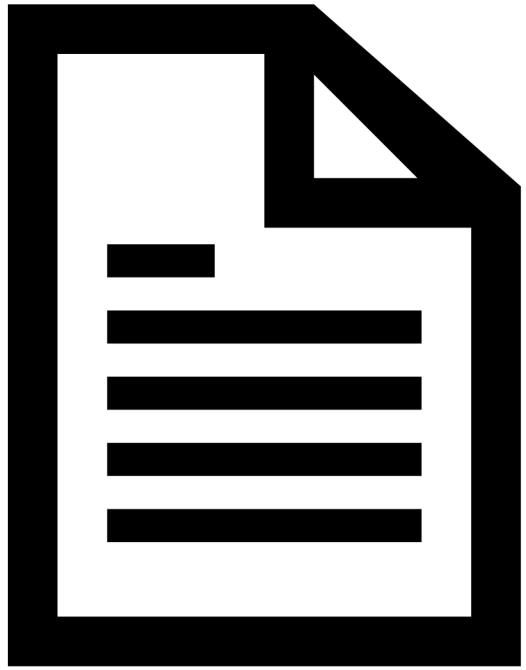
References

https://www.openmp.org/specifications/

- OpenMP 4.5 Complete Specifications
- OpenMP 4.5 Examples
- OpenMP 4.5 Summary Card C/C++

https://www.openacc.org/specification

- OpenACC 2.6 Specification



How to run my application?

1) Have compiler that supports OpenMP

   gcc, clang , Intel, IBM and Cray

2) In file include library: `#include <omp.h>`

3) To compile add flag `-fopenmp`

```
gcc -fopenmp hello_world_openmp.c -o hello
```

4) Execute defining number of threads

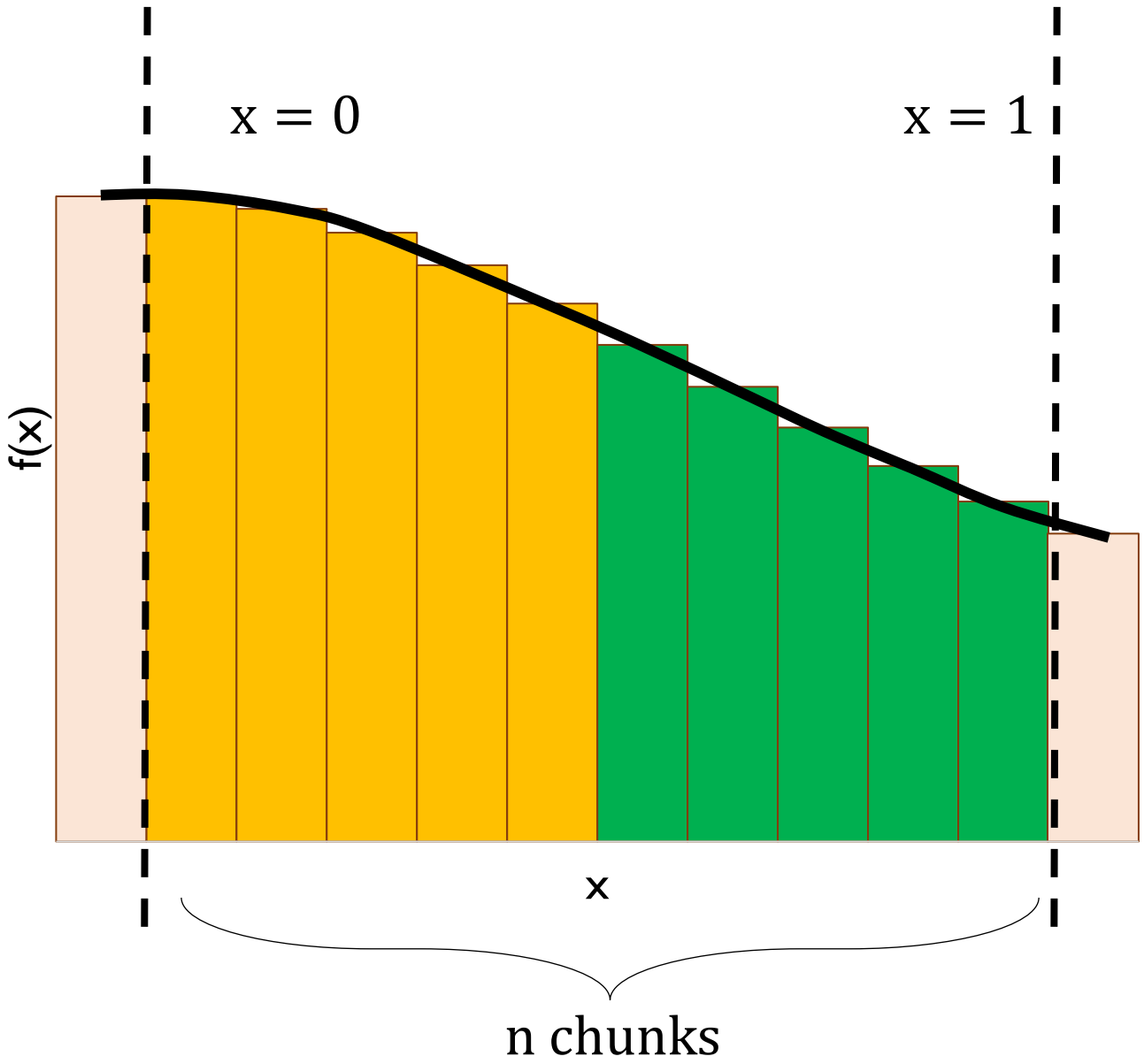
```
OMP_NUM_THREADS=4 ./hello
```

Example Compute Pi ($\pi = 3.14159\ldots$) in parallel

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx$$

P 0 $= \sum_{i=0}^{\frac{n}{2}-1} f(x_i) * \frac{1}{n}$

P 1 $= \sum_{i=n/2}^{n-1} f(x_i) * \frac{1}{n}$



Variable type

```
int value = 6;
int n = 3;
#pragma omp parallel for num_threads(n) <…>
(value)
for(int i = 0; i<n ; ++i)
{
  int id = omp_get_thread_num();
  value = value + id;
  printf("value is %d in proc %d\n",value,id);
}
printf("value is %d\n",value);
```

```
private
value is 0 in proc 0
value is 1 in proc 1
value is 2 in proc 2
value is 6
firstprivate
value is 6 in proc 0
value is 7 in proc 1
value is 8 in proc 2
value is 6
lastprivate
value is 0 in proc 0
value is -119 in proc 1
value is -118 in proc 2
value is -118
shared*
value is 6 in proc 0
value is 7 in proc 1
value is 9 in proc 2
value is 9
```