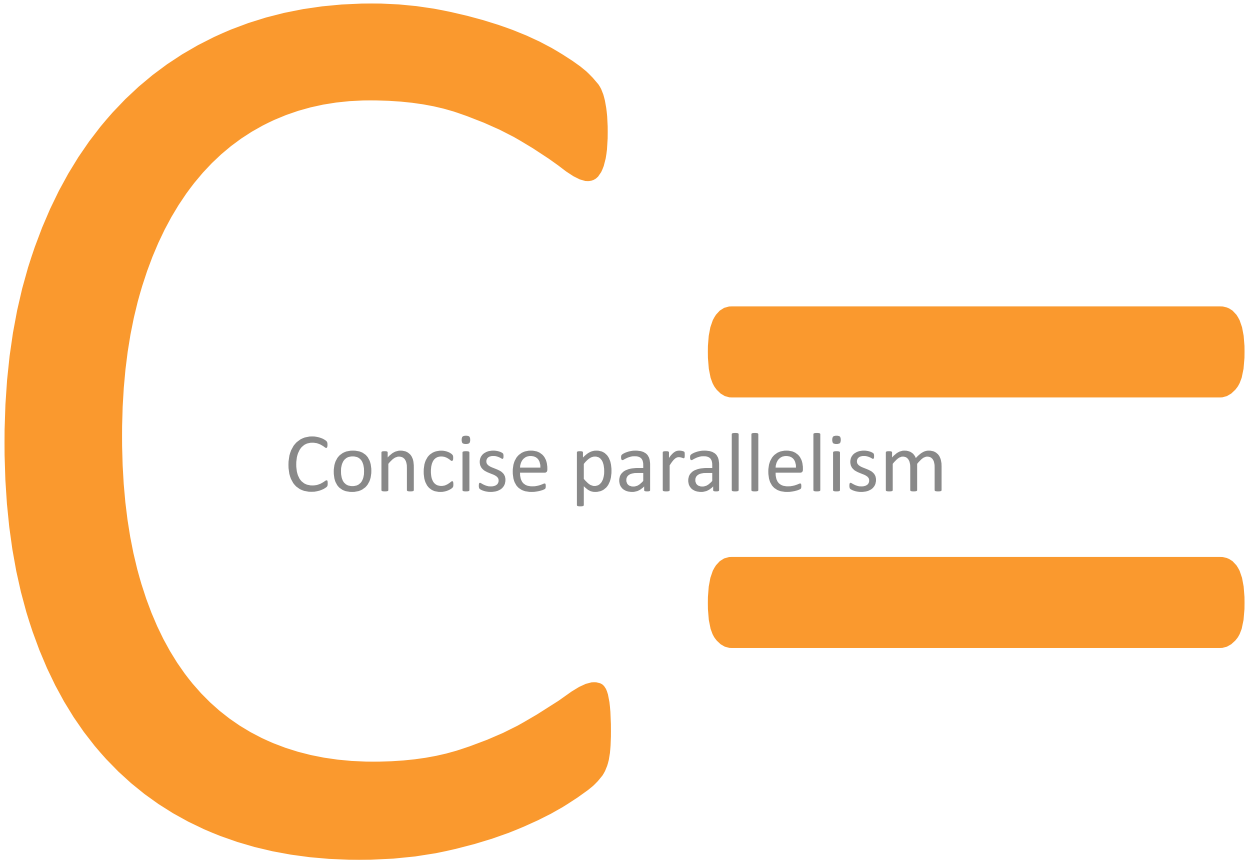
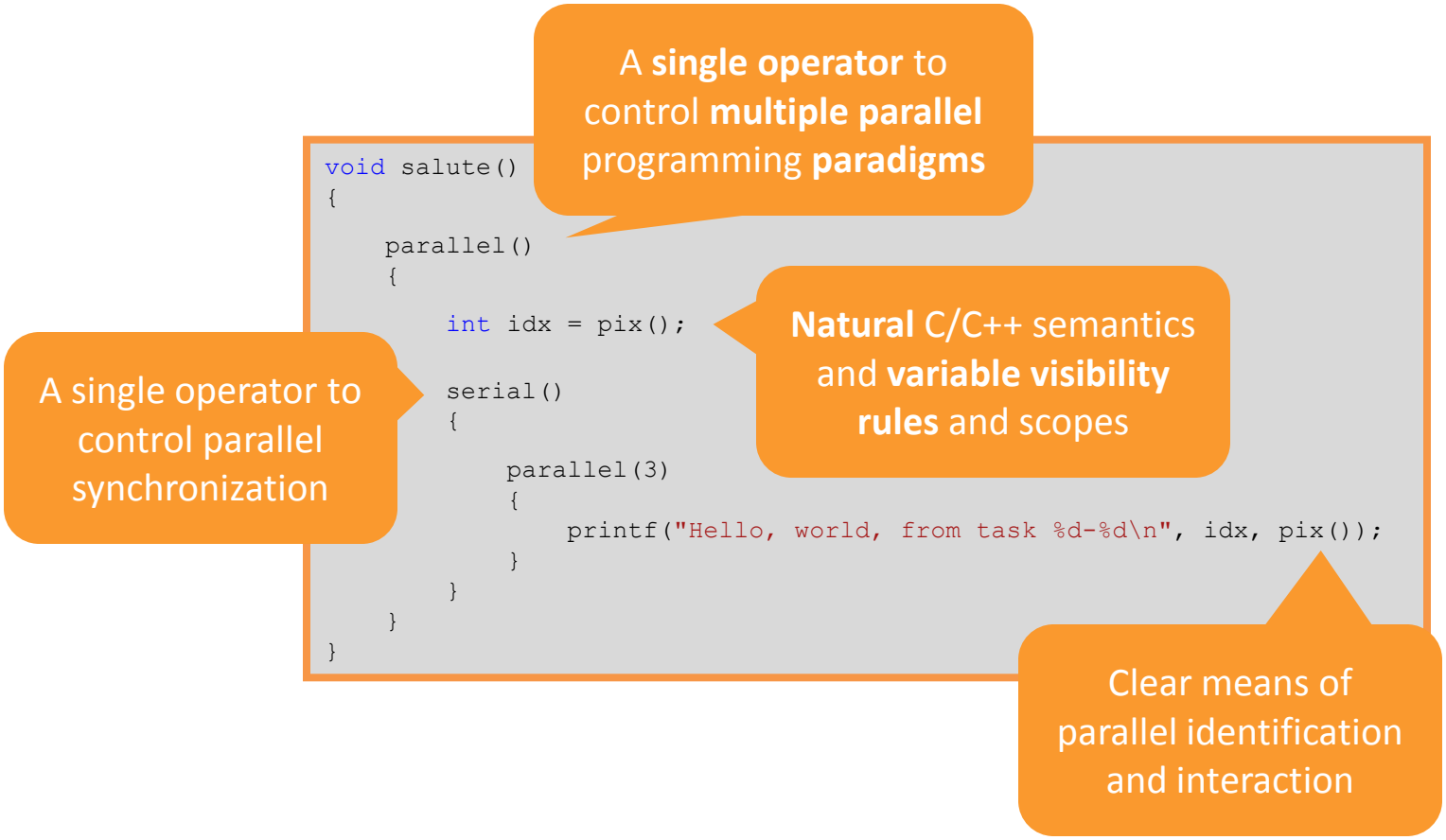

Concise parallelism

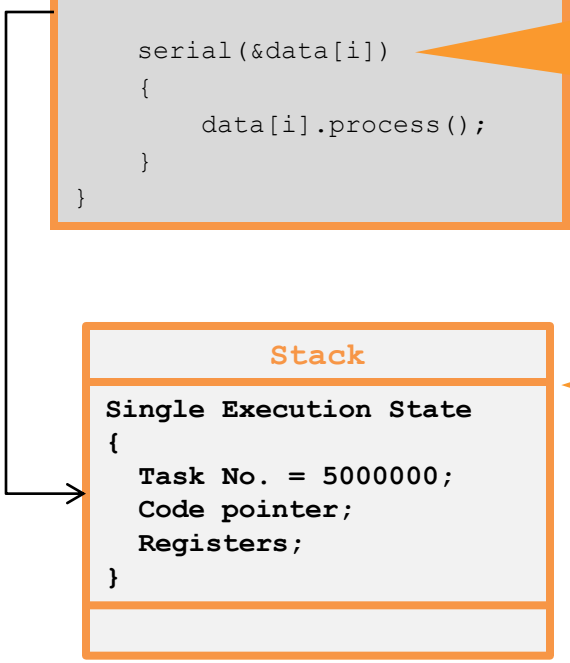
Natural C/C++ Parallelism



Elegant Multitasking

```
std::vector<Data> data;

parallel(5000000)
{
    int i = pix();

    serial(&data[i])
    {
        data[i].process();
    }
}
```

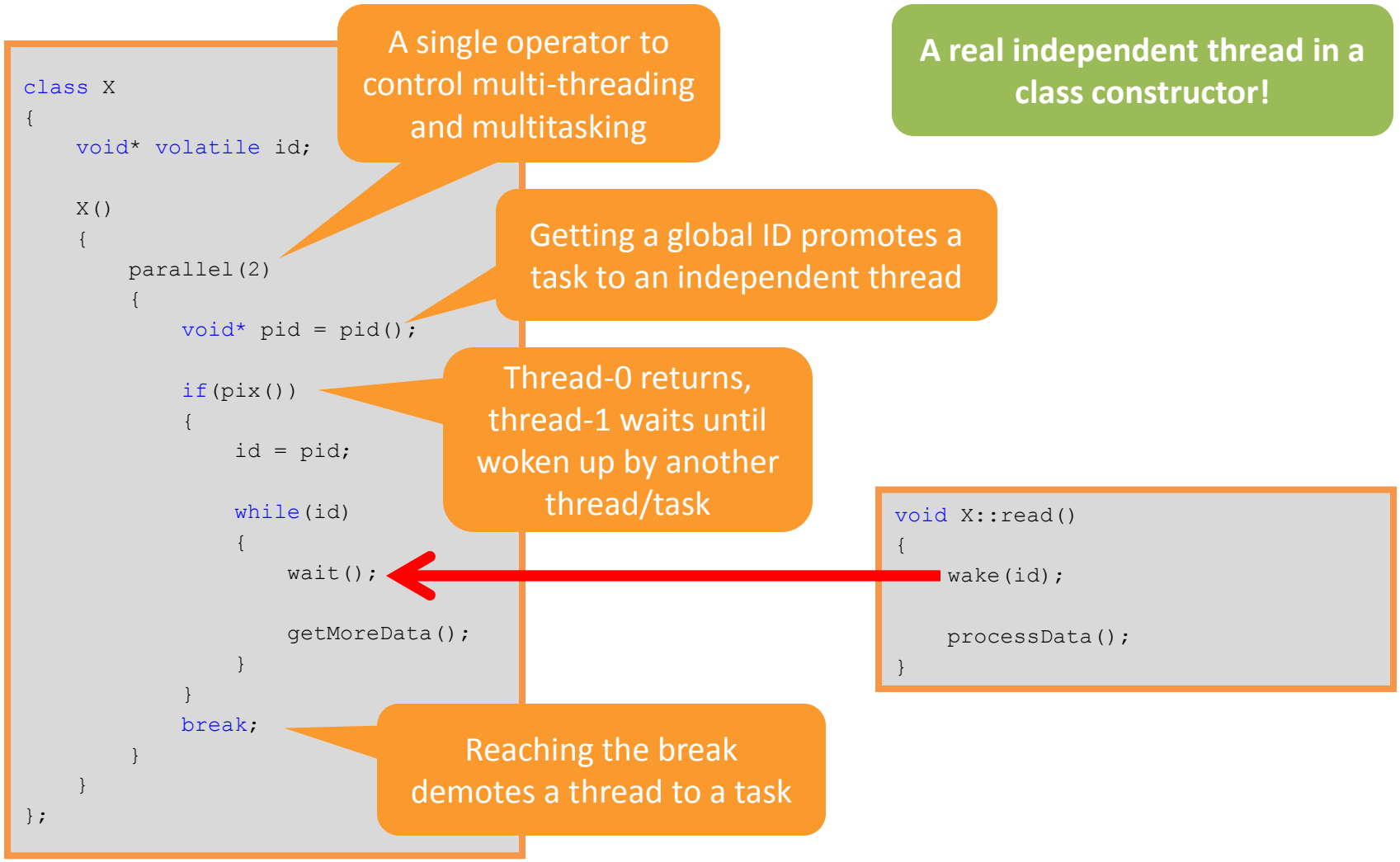
Synchronized access to any data element without introducing synchronization objects



Each thread from a pool decrements the task counter and “creates” a job to execute from a single execution state:

- No CPU oversubscription
- Dynamic work balancing
- Minimal memory footprint
- No task queue management overhead

Language-Friendly Multithreading



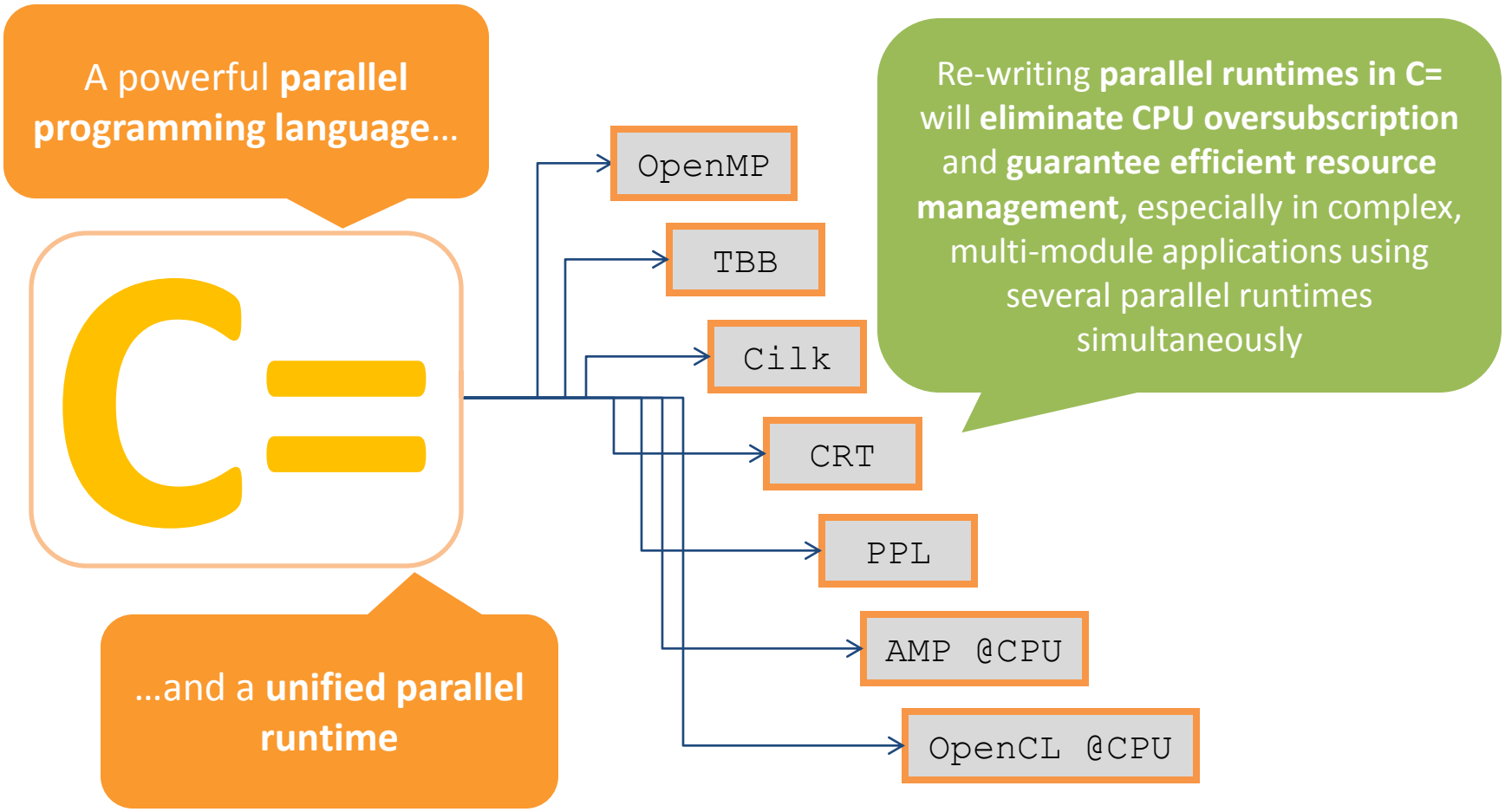
Easy Software Analysis

Use the same **compiler, debugger and profiler** tools as for sequential software

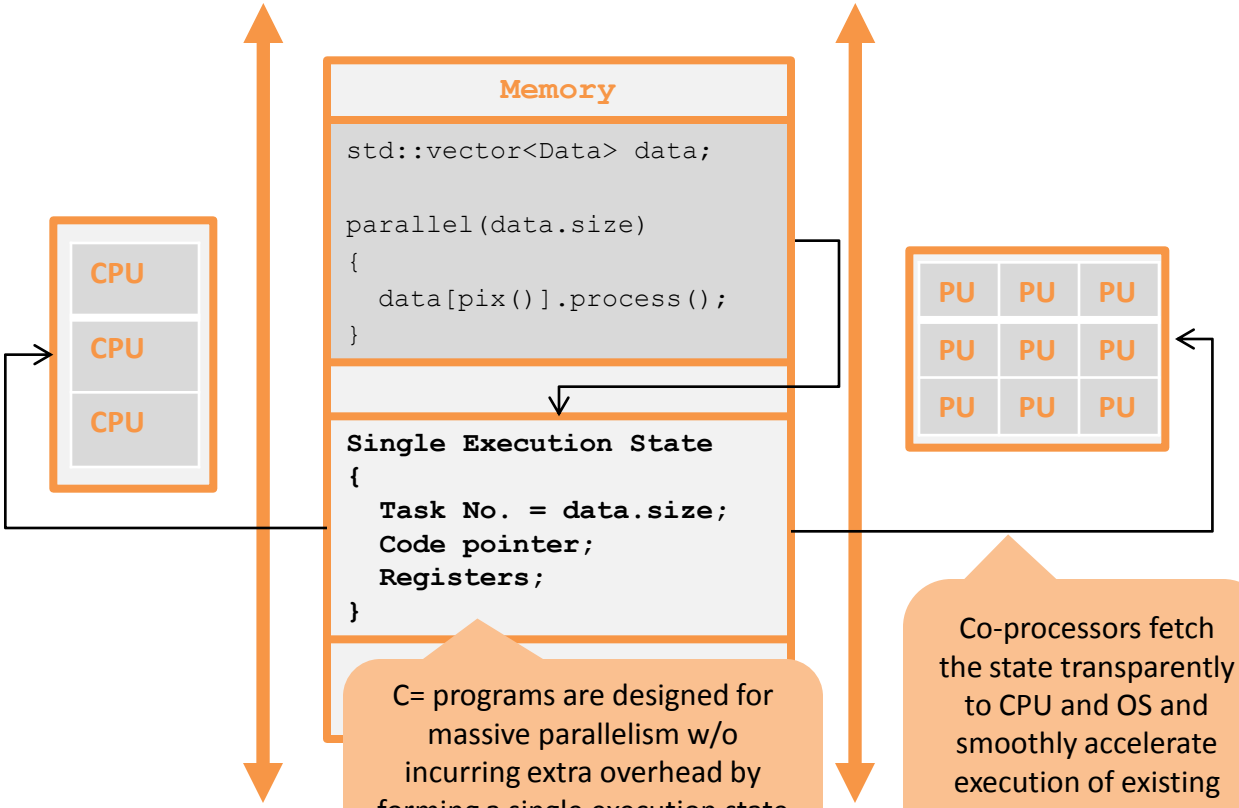
```
std::vector<Data> data;

void f(int n)
{
  parallel(data.size)
  { /// Timing: 5 sec; Parallelism = 95%; Time per CPU: CPU0 = 30%, CPU1 = 30%...
    for(int i = 0; i < n; i++)
    { /// Avrg iterations = 100
      int j = pix();

      parallel()
      { /// Timing: 4.5 sec; Parallelism = 80%; Time per CPU: CPU0 = 30%, CPU1 = 30%...
        data[j].process();

        serial()
        { /// Timing: 4.5 sec; Contention = 30%;
          data[j].reduce();
        }
      }
    }
  }
}
```

C= source code is a perfect performance model by itself: a C= profiler can annotate each parallel, sequential and cyclic region with timings, contention, iterations, balance, etc. exactly in alignment with a corresponding operator

Software Implications



Hardware Implications



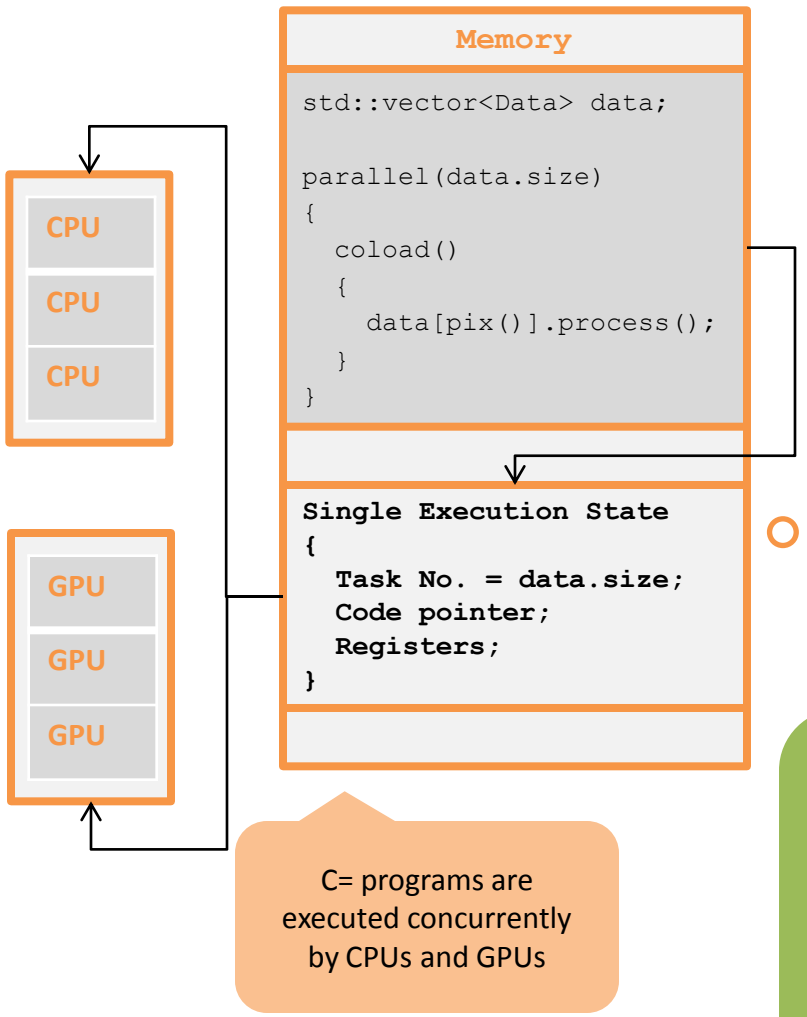
C= programs are designed for massive parallelism w/o incurring extra overhead by forming a single execution state for any number of parallel tasks

Co-processors fetch the state transparently to CPU and OS and smoothly accelerate execution of existing programs

Slide a tablet into an accelerator box and get faster software, vivid graphics, detailed scenes, real-time video encoding – right away!

Truly mobile, data-consistent, cheap and powerful architecture!

One Program Fits All



Remote agents may concurrently “steal” the work from C= execution states and utilize their CPUs and GPUs

Unified Semantic Concept of Parallelism enables distributed heterogeneous programming with a single parallel operator