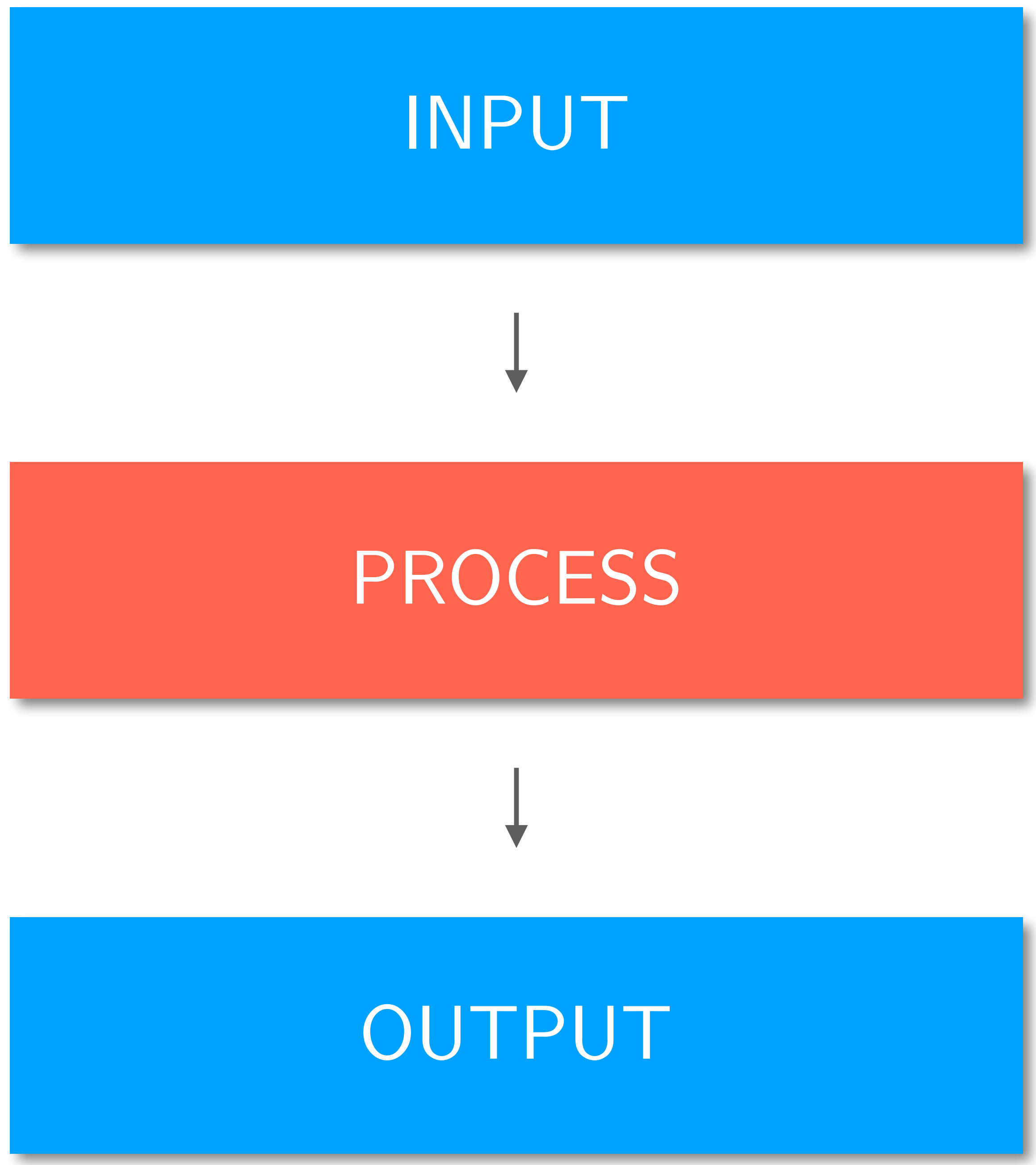
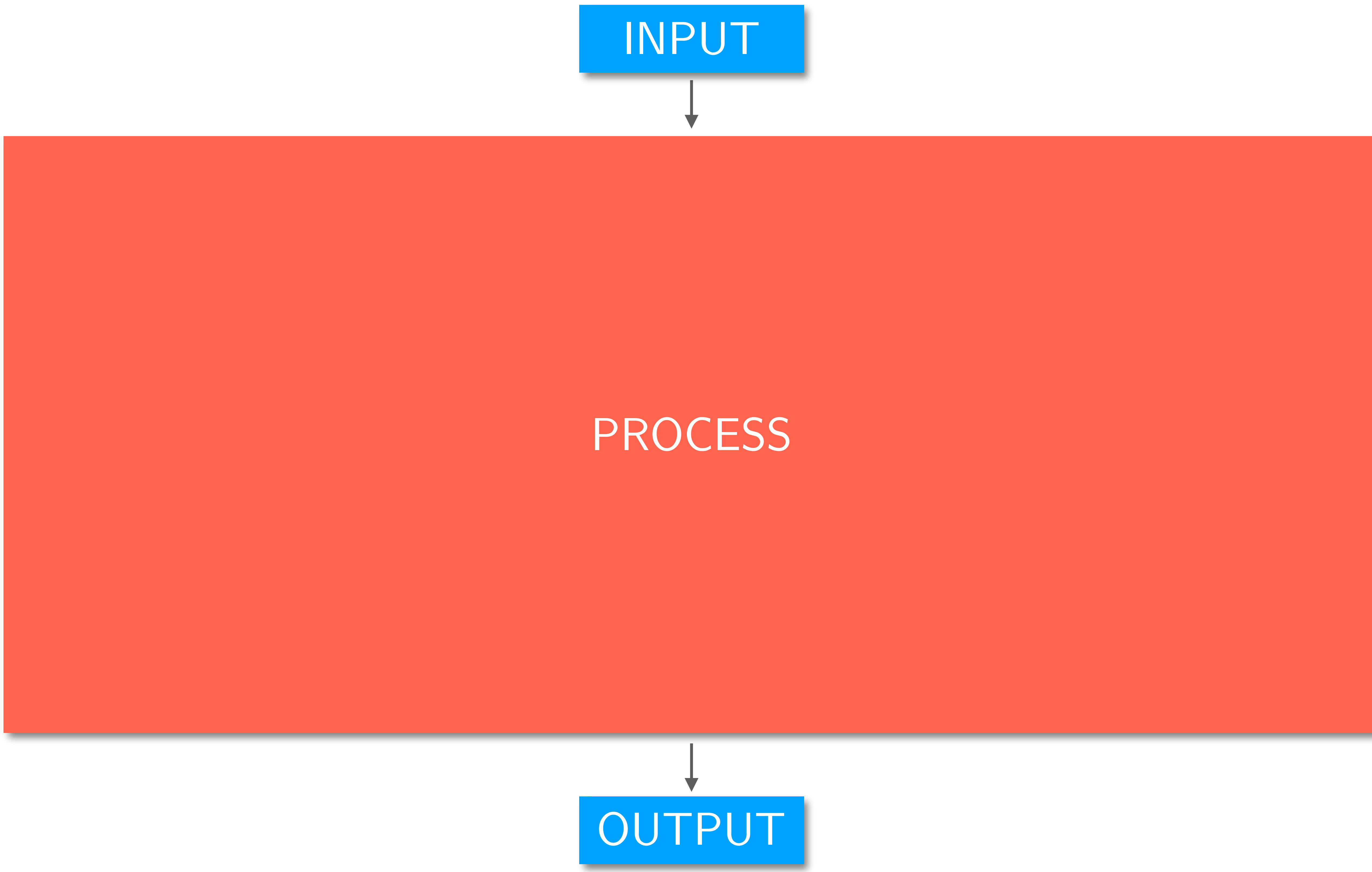
Typical Application

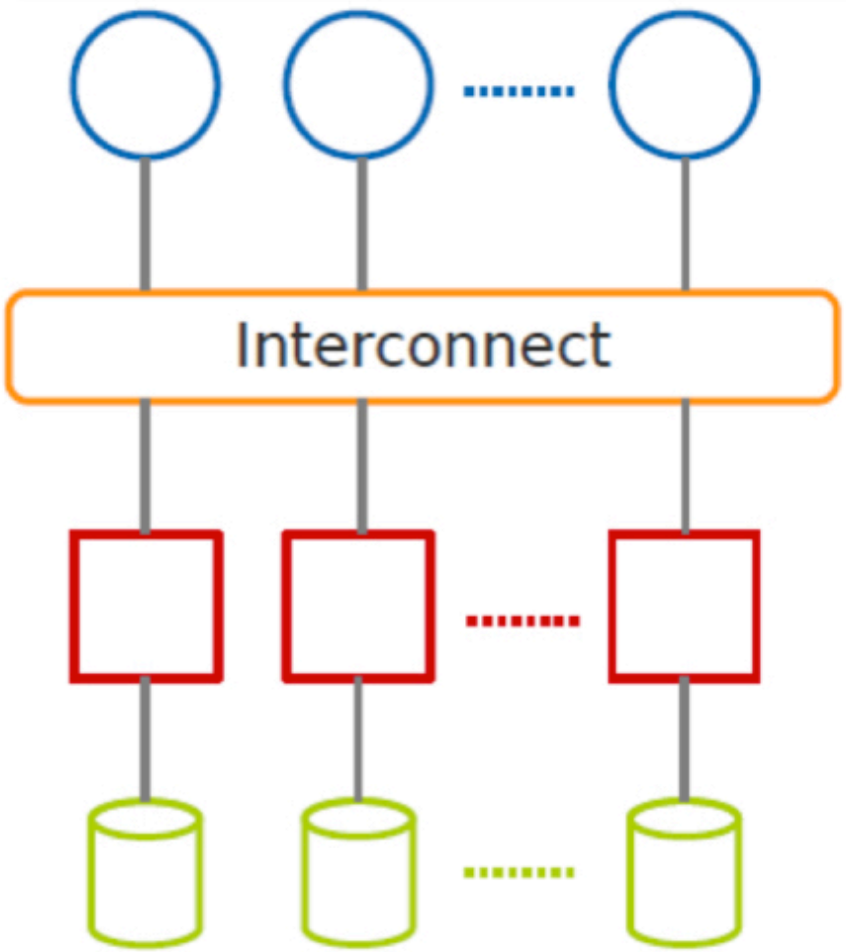


What if?



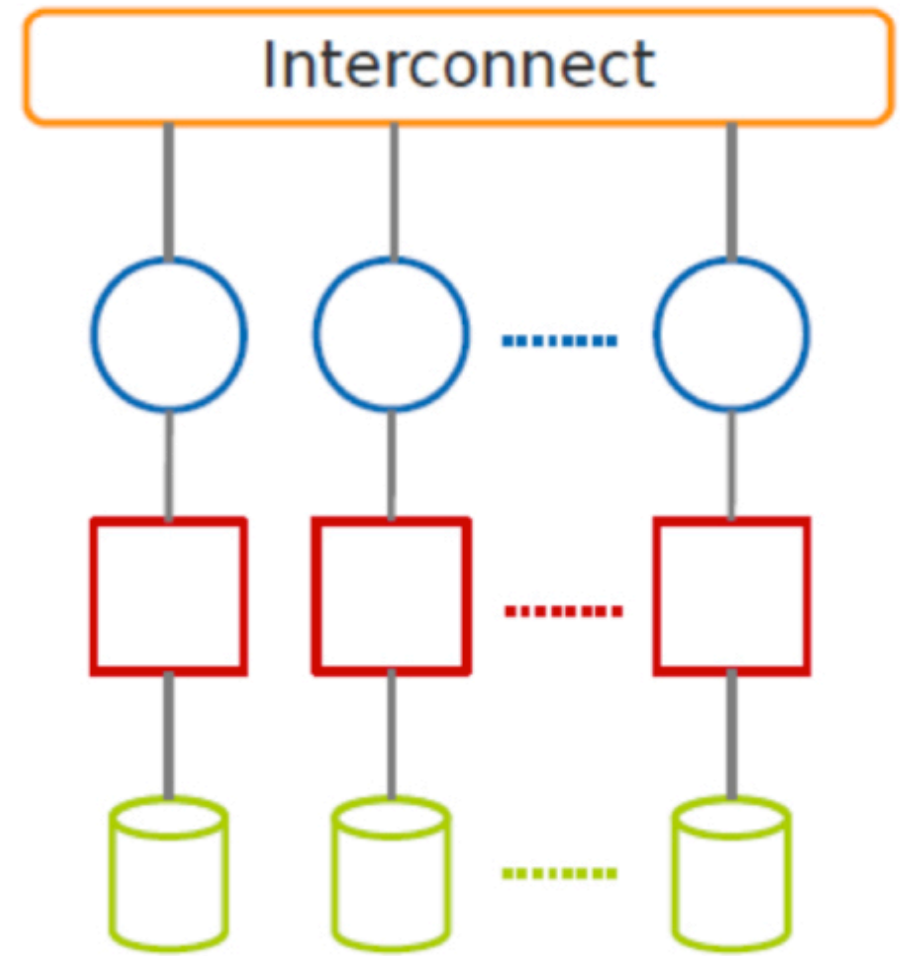
Parallel Architectures

Shared Memory



- Posix Threads
- OpenMP
- Automatic Parallelization (Compiler optimizations)

Message Passing



- Sockets
- PVM - Parallel Virtual Machine (obsolete)
- MPI - Message Passing Interface

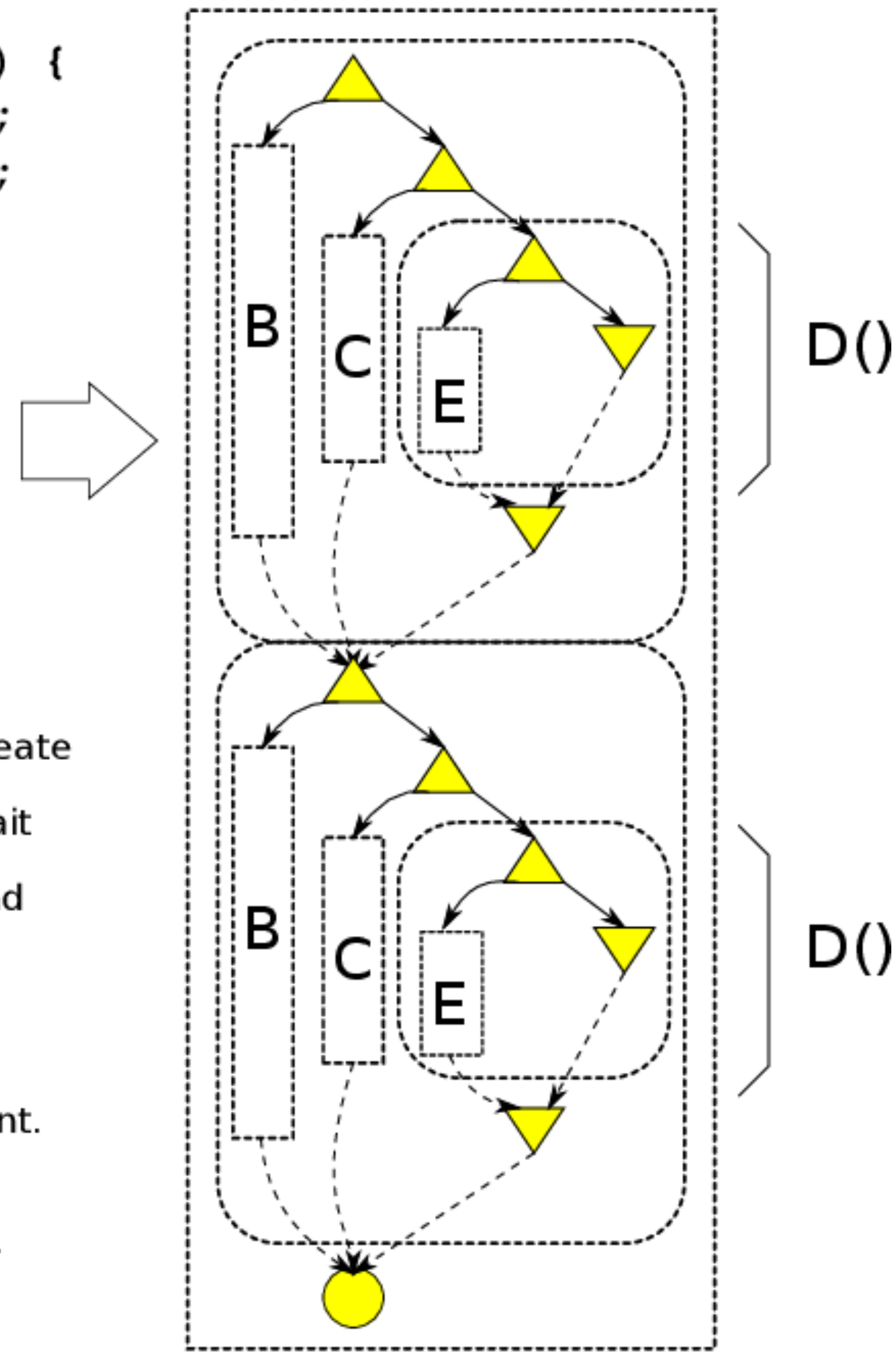
DeWitt and Gray, "Parallel database systems: the future of high performance database systems", ACM Communications, 1992

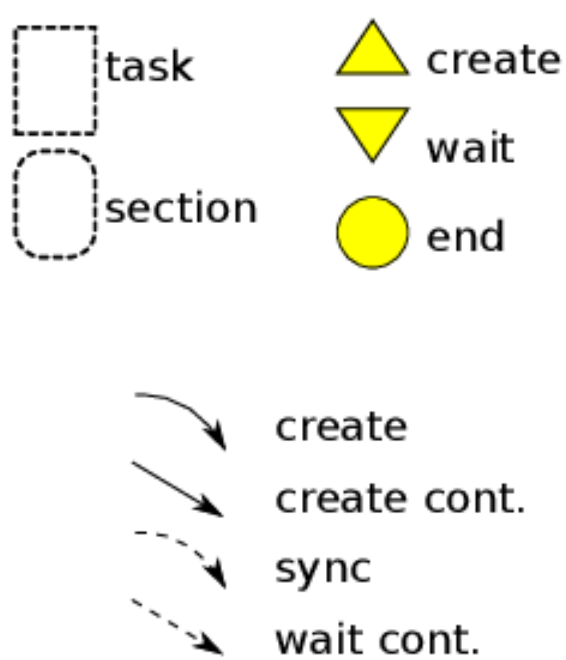
Designing Parallel Algorithms

- Typical steps:
  - Identify what pieces of work can be performed concurrently
  - Partition concurrent work onto independent processors
  - Distribute a program's input, output, and intermediate data
  - Coordinate accesses to shared data: avoid conflicts
  - Ensure proper order of work using synchronization
- Some steps can be omitted
  - For shared memory parallel programming model, there is no need to distributed data
  - For message passing parallel programming model, there is no need to coordinate shared data
  - Processor partition may be done automatically

Task Dependency Graphs

```
A() {
  for(i=0;i<2;i++) {
    CreateTask(B);
    CreateTask(C);
    D();
    WaitTasks();
  }
}
D() {
  CreateTask(E);
  WaitTasks();
}
```





Granularity

- Granularity = task size
  - Fine-grain = small tasks, large number of tasks
  - Coarse-grain = large tasks, small number of tasks
  - Choose the proper granularity based on the problem and hardware
- Example: matrix multiplication
  - N x N matrix A multiply N x 1 vector b give N x 1 vector y
  - Embarrassing parallel: each row of A times vector b gives an element of y
  - Simple decomposition:
    - Task size is uniform
    - No dependencies between task – All tasks share b
  - Fine-grained: each task process one row of A
  - Coarse-grained: each task process three rows of A

Parallelism Level

- Definition
  - number of tasks that can execute in parallel
    - may change during program execution
- Metrics
  - maximum parallelism level: largest # concurrent tasks at any point in the execution
  - average parallelism level: average number of tasks that can be processed in parallel
- Degree of concurrency vs. task granularity
  - inverse relationship

Limits

- What bounds parallel execution time?
  - maximum parallelism degree, e.g. matrix-vector multiplication example $\leq N^2$ concurrent tasks
  - dependencies between tasks
  - parallelization overheads, e.g., cost of communication between tasks
  - fraction of application work that can't be parallelized
- No single decomposition technique works for all problems

Programmer problems: splitting code

- How to divide code into parallel tasks?
- How to distribute the code?
- How to coordinate the execution?
- How to load the data?
- How to store the data?
- What if more tasks than CPUs?
- What if a CPU crashes?
- What if a CPU is taking too long?
- What if the CPUs are different?
- What if we have a new (serial) code?