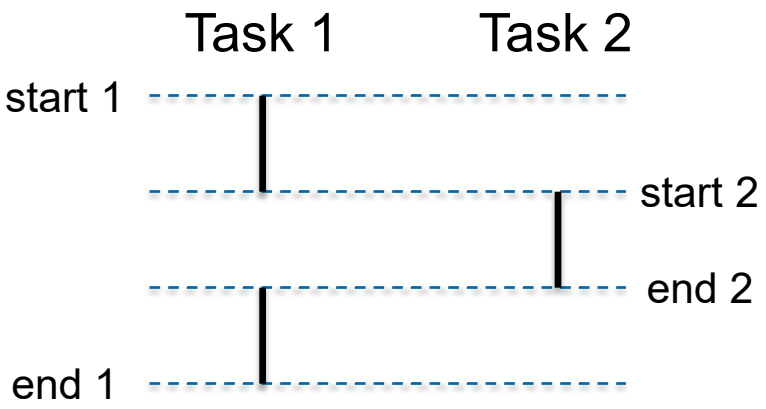
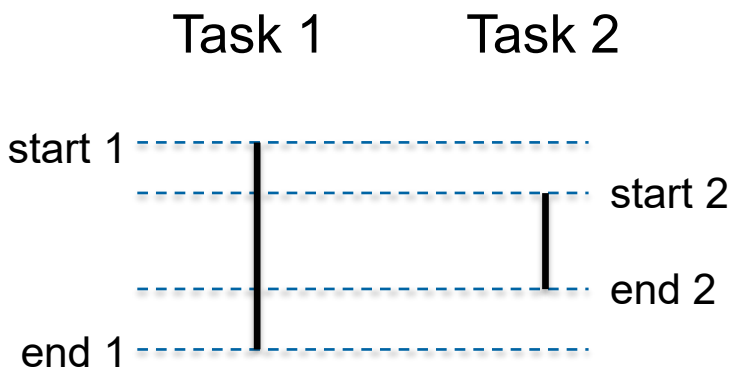
# Module 1: Why Use Concurrency
# Topic 2.1: Concurrent vs Parallel

# Concurrent Execution

- Concurrent execution is not necessarily the same as parallel execution
- **Concurrent**: start and end times overlap
- **Parallel**: execute at exactly the same time



# Concurrent vs. Parallel

- Parallel tasks must be executed on different hardware
- Concurrent tasks **may be** executed on the same hardware
  - Only one task actually executed at a time
- Mapping from tasks to hardware is not directly controlled by the programmer
  - At least not in Go

# Concurrent Programming

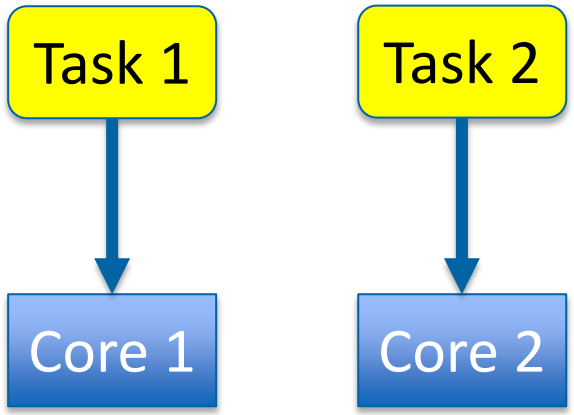
- Programmer determines which tasks can be executed in parallel
- Mapping tasks to hardware
  - Operating system
  - Go runtime scheduler

# Hiding Latency

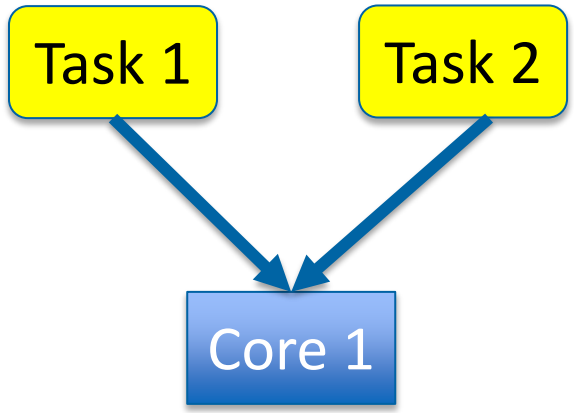
- Concurrency can improve performance, even without parallelism
- Tasks must periodically wait for something
  - i.e. wait for memory
  - `X = Y + Z` read Y, Z from memory
  - May wait 100+ clock cycles
- Other concurrent tasks can operate while one task is waiting

# Hardware Mapping

Parallel Execution




Concurrent Execution



# Hardware Mapping in Go

- Programmer does not determine the hardware mapping
- Programmer makes parallelism possible
- Hardware mapping depends on many factors
  - Where is the data?
  - What are the communication costs?