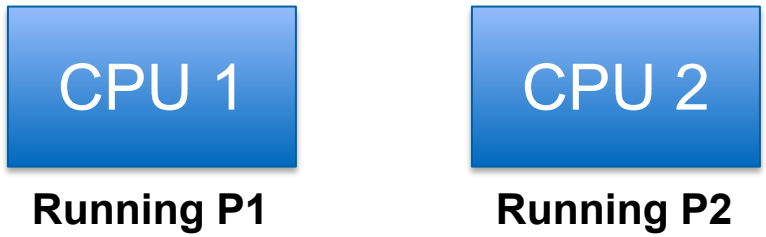
Module 1: Why Use Concurrency?
Topic 1.1: Parallel Execution

Parallel Execution

- Two programs execute in parallel if they **execute at exactly the same time**
- At time t, an instruction is being performed for both P1 and P2



- Need replicated hardware

Why Use Parallel Execution

- **Tasks may complete more quickly**
- Example: Two piles of dishes to wash
  - Two dish washers can complete twice as fast as one
- Some tasks must be performed sequentially
- Example: Wash dish, dry dish
  - Must wash before you can dry
- **Some tasks are parallelizable and some are not**

Module 1: Why Use Concurrency?
Topic 1.2: Von Neumann Bottleneck

Speedup Without Parallelism

- Can we achieve speedup without parallelism?
- Design faster processors
  - Get speedup without changing software
- Design processors with more memory
  - Reduces the **von Neumann bottleneck**
  - Cache access time = 1 clock cycle
  - Main memory access time = ~100 clock cycles
  - Increasing on-chip cache improves performance

Moore's Law

- Predicted that transistor density would double every 2 years
- Not a physical law, just an observation
- **Smaller transistors switch faster**
- Exponential increase in density would lead to exponential increase in speed

Module 1: Why Use Concurrency?
Topic 1.3: Power Wall

Power/Temperature Problem

- Transistors consume power when they switch
- Increasing transistor density leads to increased power consumption
  - Small transistors use less power, but density scaling is faster
- High power leads to high temperature
- Air cooling (fans) can only remove so much heat

Dynamic Power

- **$P = \alpha * CFV^2$**
- α is percent of time switching
- C is capacitance (related to size)
- F is the clock frequency
- V is voltage swing (from low to high)

- Voltage is important
- 0 to 5V uses much more power than 0 to 1.3 V

Dennard Scaling

- **Voltage should scale** with transistor size
- Keeps power consumption, and temperature, low
- Problem: Voltage can't go too low
  - Must stay above **threshold voltage**
  - **Noise** problems occur
- Problem: Doesn't consider **leakage power**
- Dennard scaling must stop

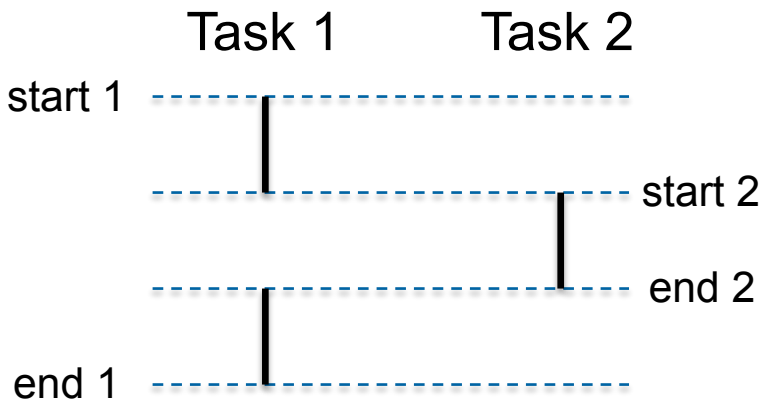
Multi-Core Systems

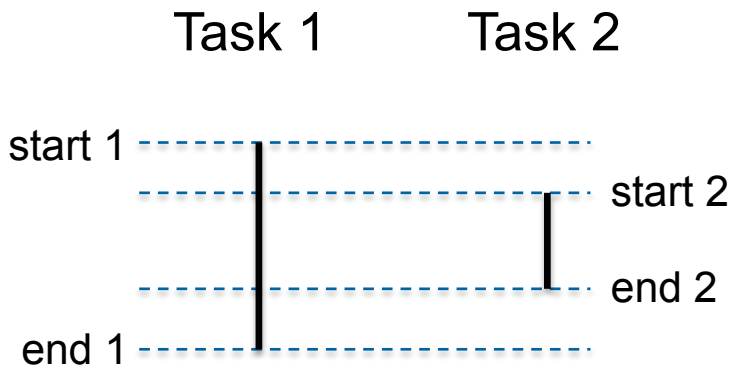
- **$P = \alpha * CFV^2$**
- Cannot increase frequency
- Can still add processor cores, without increasing frequency
  - Trend is apparent today
- **Parallel execution is needed to exploit multi-core systems**
- Code made to execute on multiple cores
- Different programs on different cores

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

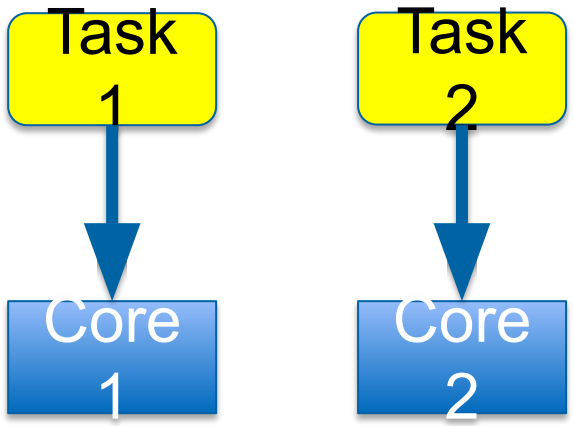
- Programmer determines which tasks can be executed in parallel
- Mapping tasks to hardware
  - Operating system
  - Go runtime scheduler

Hiding Latency

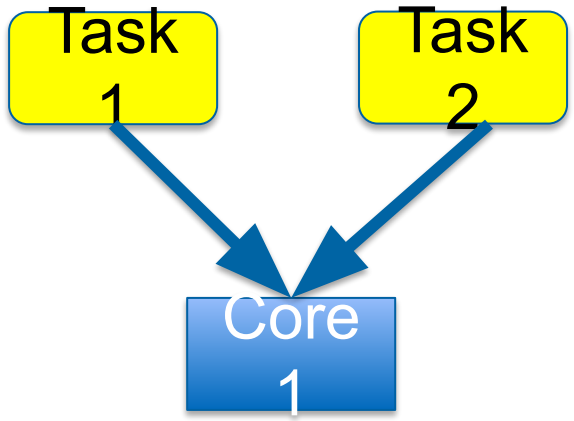
- Concurrency can improve performance, even without parallelism
- Tasks must periodically wait for something
  - i.e. wait for memory
  - `X = Y + Z` read Y, Z from memory
  - May wait 100+ clock cycles
- Other concurrent tasks can operate while one task is waiting

Hardware Mapping

Parallel Execution

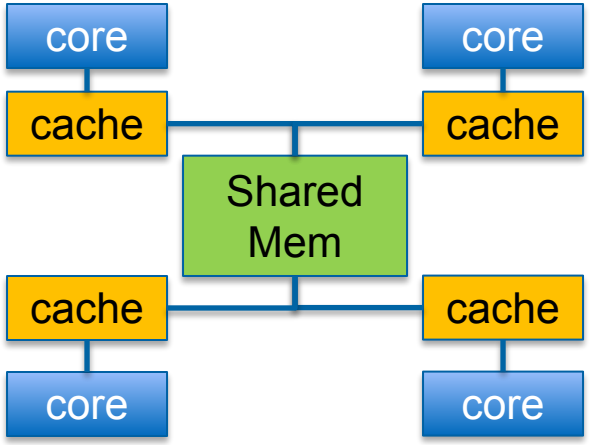


Concurrent Execution



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?



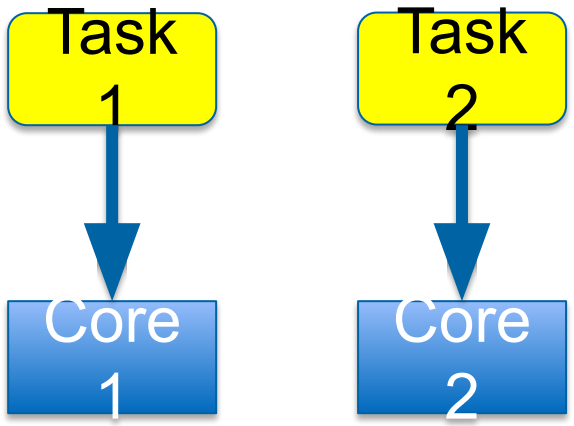
Module 1: Why Use Concurrency
Topic 2.2: Hiding Latency

Hiding Latency

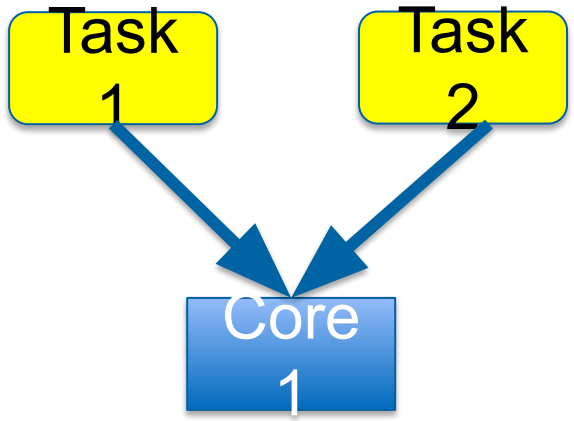
- Concurrency can improve performance, even without parallelism
- Tasks must periodically wait for something
  - i.e. wait for memory
  - `X = Y + Z` read Y, Z from memory
  - May wait 100+ clock cycles
- Other concurrent tasks can operate while one task is waiting

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?