High Performance Programming

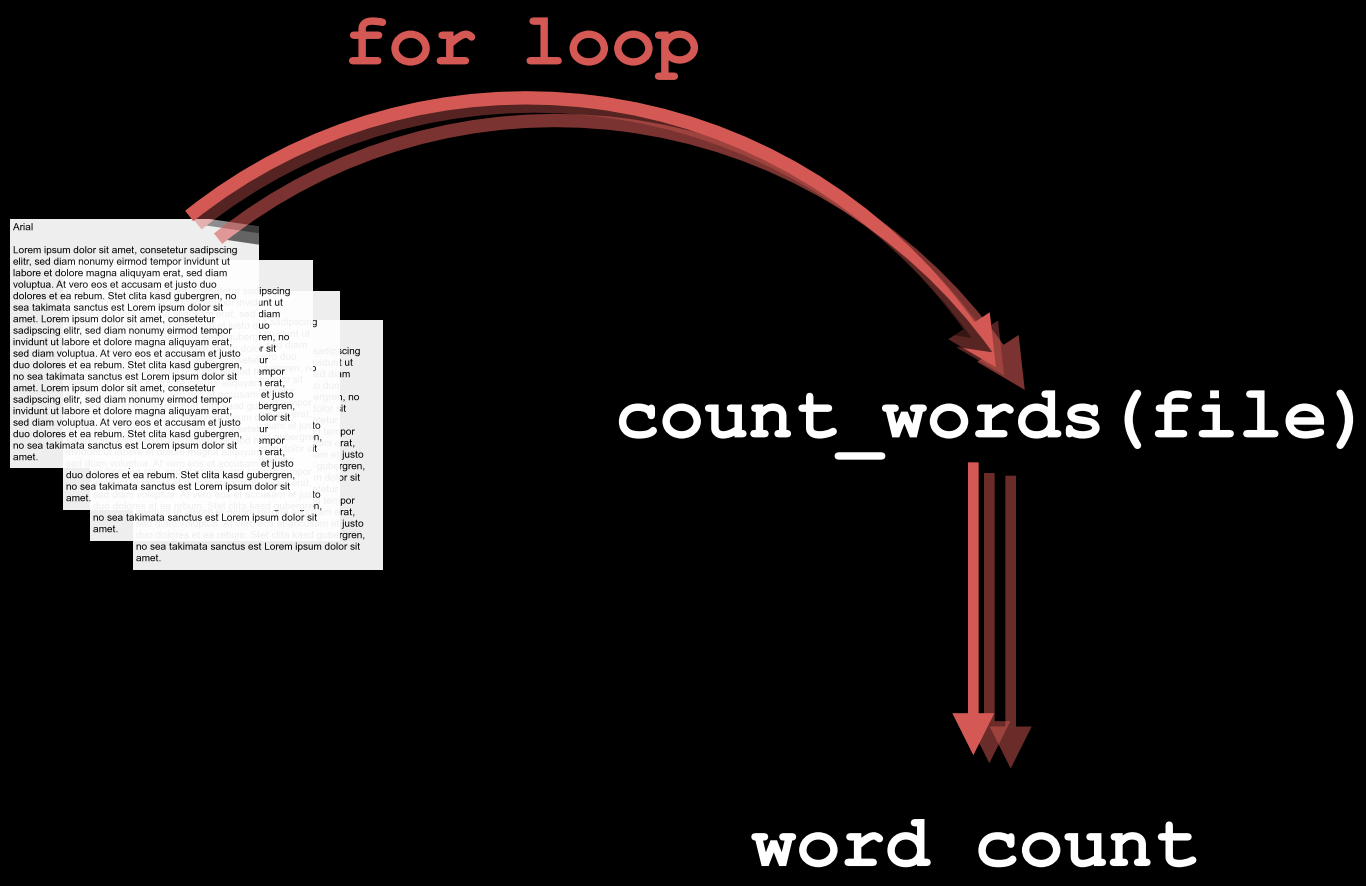
Goals

- Motivation
- Intro to computing resources
- Multi-core processing
  - ⋆ Parallelism
- Threading
  - ⋆ Concurrency

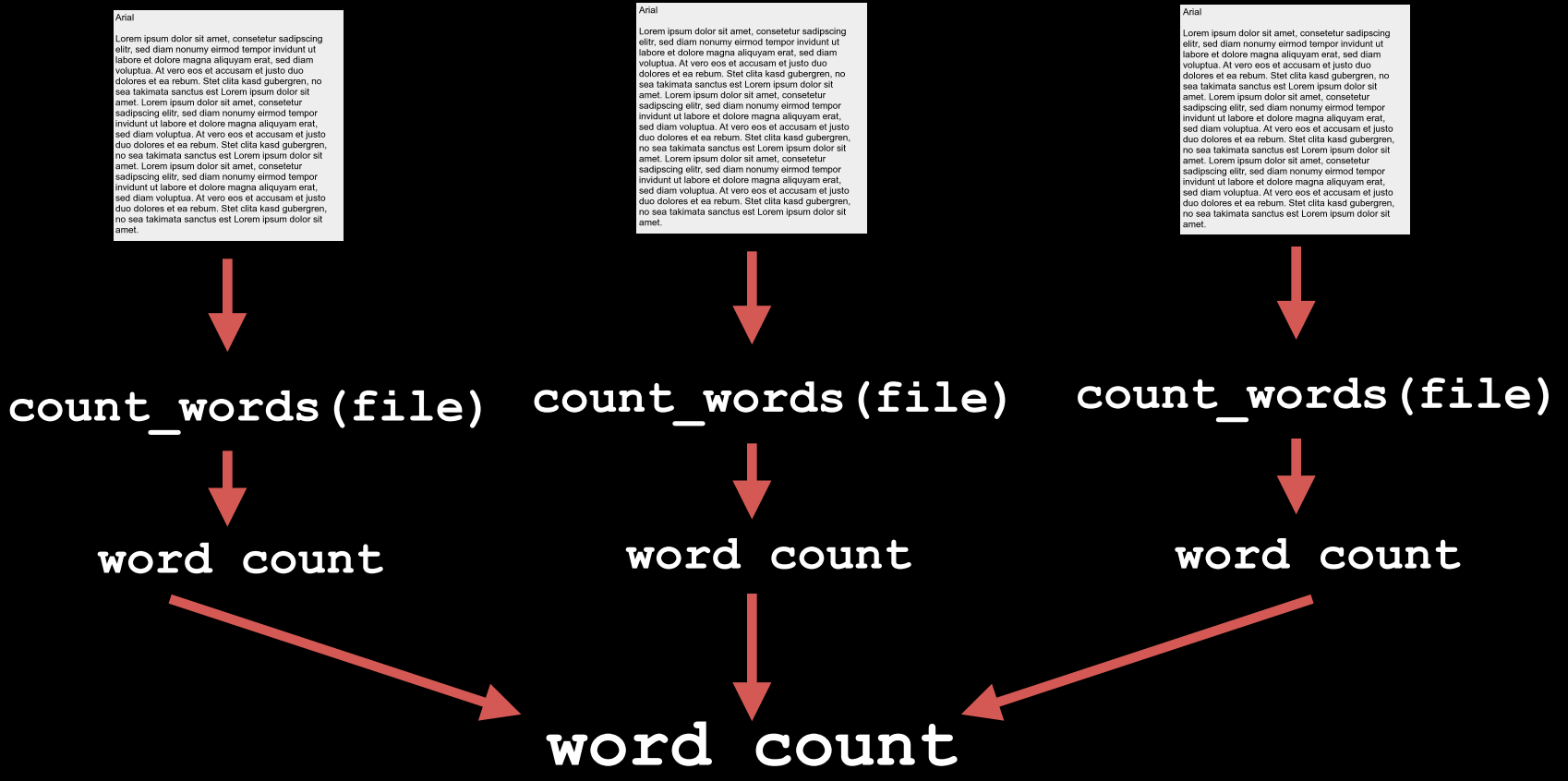
Goals

- Motivation
- Intro to computing resources
- Multi-core processing
  - ★ Parallelism
- Threading
  - ★ Concurrency

Counting words in documents



Counting words in documents (in parallel)



Motivation

- Process biggish data (≥ 5GB depending on task)
- Saves time
- More efficient use of CPU resources

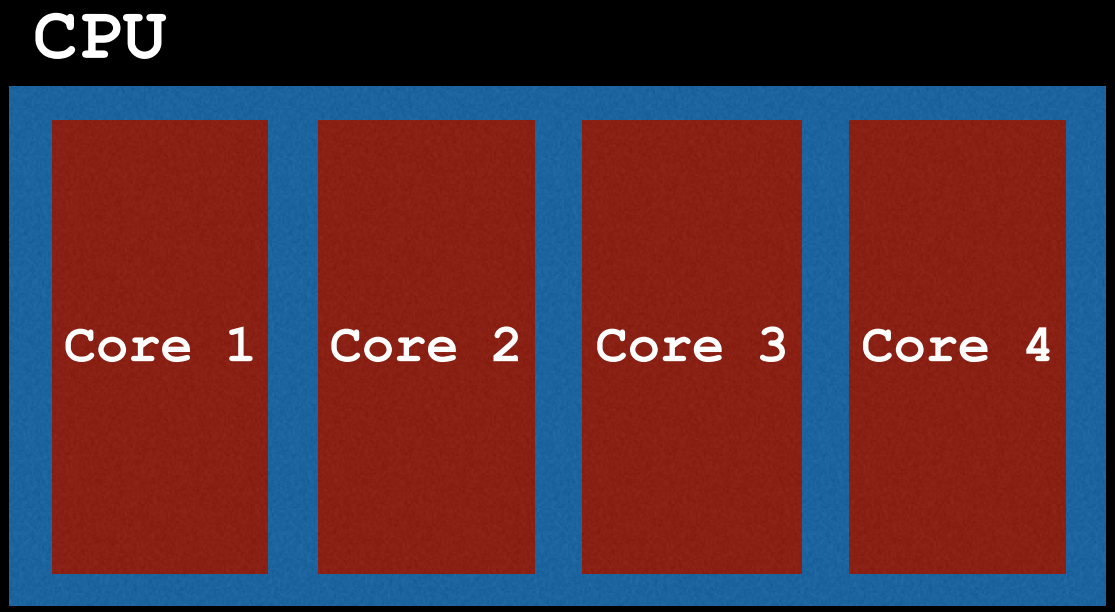
Goals

- Motivation
- Intro to computing resources
- Multi-core processing
  - ★ Parallelism
- Threading
  - ★ Concurrency

Types of Computing Resources

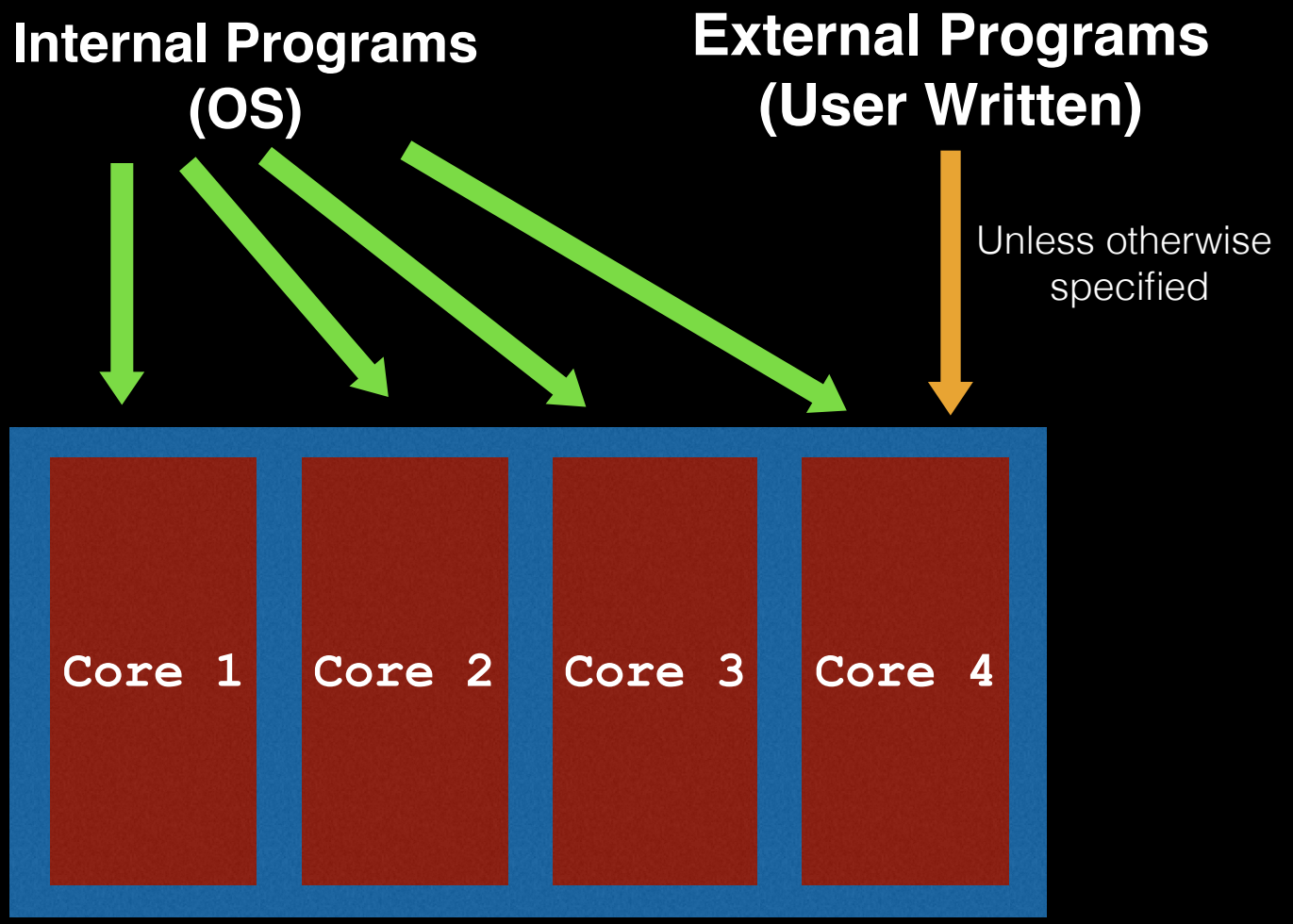
- **Central Processing Unit (CPU)**
  - ★ `Unit: GHz (instructions/second)`
- **Random Access Memory (RAM / Memory)**
  - ★ `Unit: GB (Gigabytes)`

CPU Cores



- A CPU can hold multiple cores
- Each core is a self-contained processor that can execute programs

How are CPU Cores used



What happens when a program runs

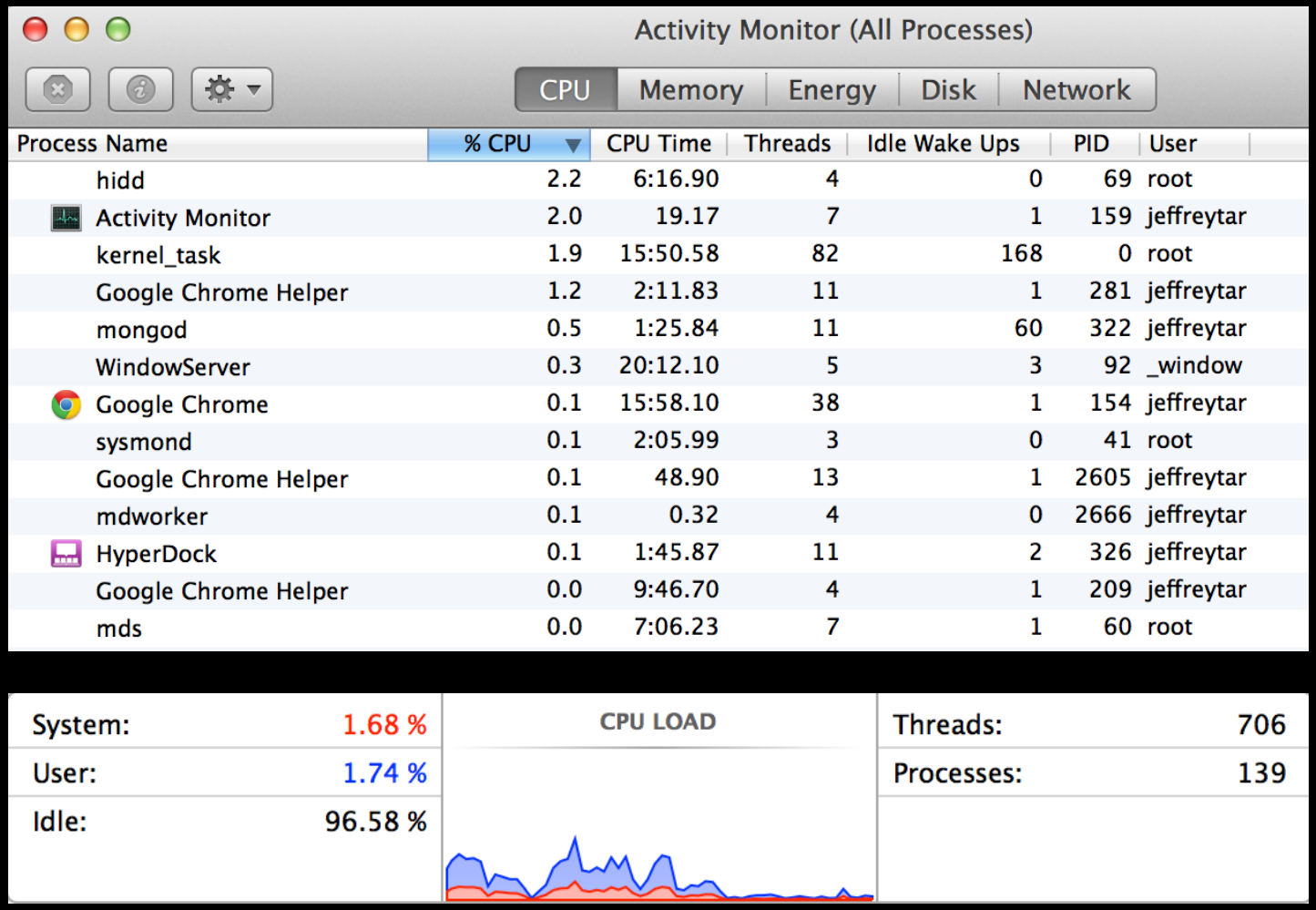
Program → **Process**

```
def edges_from_osmdb(osmdb, vertex_namespace, slogs, profiledb=None):
    """generates (vertex1_label, vertex2_label, edgepayload) from osmdb"""

    street_id_counter = 0
    street_names = {}

    # for each edge in the osmdb
    for i, (id, parent_id, node1, node2, distance, geom, tags) in enumerate( osmdb.edges() ):

        # Find rise/fall of edge, if profiledb is given
        rise=0
        fall=0
        if profiledb:
            profile = profiledb.get( id )
            if profile:
                rise, fall = get_rise_and_fall( profile )

        # insert end vertices of edge to graph
        vertex1_label = "%s-%s"%(vertex_namespace,node1)
        vertex2_label = "%s-%s"%(vertex_namespace,node2)

        # create ID for the way's street
        street_name = tags.get("name")
        if street_name is None:
            street_id_counter += 1
            street_id = street_id_counter
        else:
            if street_name not in street_names:
                street_id_counter += 1
                street_names[street_name] = street_id_counter
            street_id = street_names[street_name]

        # Create edges to be inserted into graph
        s1 = Street( id, distance, rise, fall )
        s2 = Street( id, distance, fall, rise, reverse_of_source=True )
        s1.way = street_id
        s2.way = street_id

        # See if the way's highway tag is penalized with a 'slog' value; if so, set it in the edges
        slog = slogs.get( tags.get("highway") )
        if slog:
            s1.slog = s2.slog = slog

        # Add the forward edge and the return edge if the edge is not oneway
    yield vertex1_label, vertex2_label, s1

        oneway = tags.get("oneway")
        if oneway != "true" and oneway != "yes":
        yield vertex2_label, vertex1_label, s2
```

1. Process ID (pid)
2. Program Code

3. CPU Scheduling
4. Memory

How much CPU / Memory



How much CPU / Memory

Command line tool: top

```
Processes: 163 total, 2 running, 7 stuck, 154 sleeping, 785 threads                                14:09:10
Load Avg: 1.66, 1.71, 1.72  CPU usage: 0.67% user, 3.36% sys, 95.96% idle  SharedLibs: 71M resident, 0B data, 12M linkedit.
MemRegions: 42943 total, 2229M resident, 63M private, 1895M shared. PhysMem: 7607M used (1073M wired), 283M unused.
VM: 425G vsize, 1313M framework vsize, 1159144(0) swapins, 2470739(0) swapouts. Networks: packets: 481571/399M in, 200872/77M out.
Disks: 1065467/56G read, 274229/20G written.

PID   COMMAND      %CPU TIME     #TH  #WQ  #PORT #MREG MEM    RPRVT  PURG  CMPRS  VPRVT  VSIZE  PGRP PPID STATE    UID  FAULTS    COW
3274  screencaptur 0.0  00:00.12 2    0    47    91    1916K  888K   16K   0B     21M    2439M  162  162  sleeping 501  3753      243
3273  top          10.3 00:02.84 1/1  0    23    40    2524K+ 2296K+ 0B    0B     45M    2403M  3273 3252 running  0    38122+    91
3252  zsh          0.0  00:00.06 1    0    19    51    1960K  1816K  0B    0B     37M    2396M  3252 3251 sleeping 501  1874      501
3251  login        0.0  00:00.02 2    0    30    46    908K   580K   0B    0B     53M    2411M  3251 155  sleeping 0    811       143
3245  CVMCompiler  0.0  00:00.08 2    1    32    67    12M    11M    0B    0B     63M    2440M  3245 143  sleeping 501  5723      244
3224  zsh          0.0  00:00.06 1    0    19    52    1976K  1820K  0B    0B     45M    2404M  3224 3223 sleeping 501  1844      479
3223  login        0.0  00:00.02 2    0    30    46    996K   676K   0B    0B     53M    2411M  3223 155  sleeping 0    814       140
```

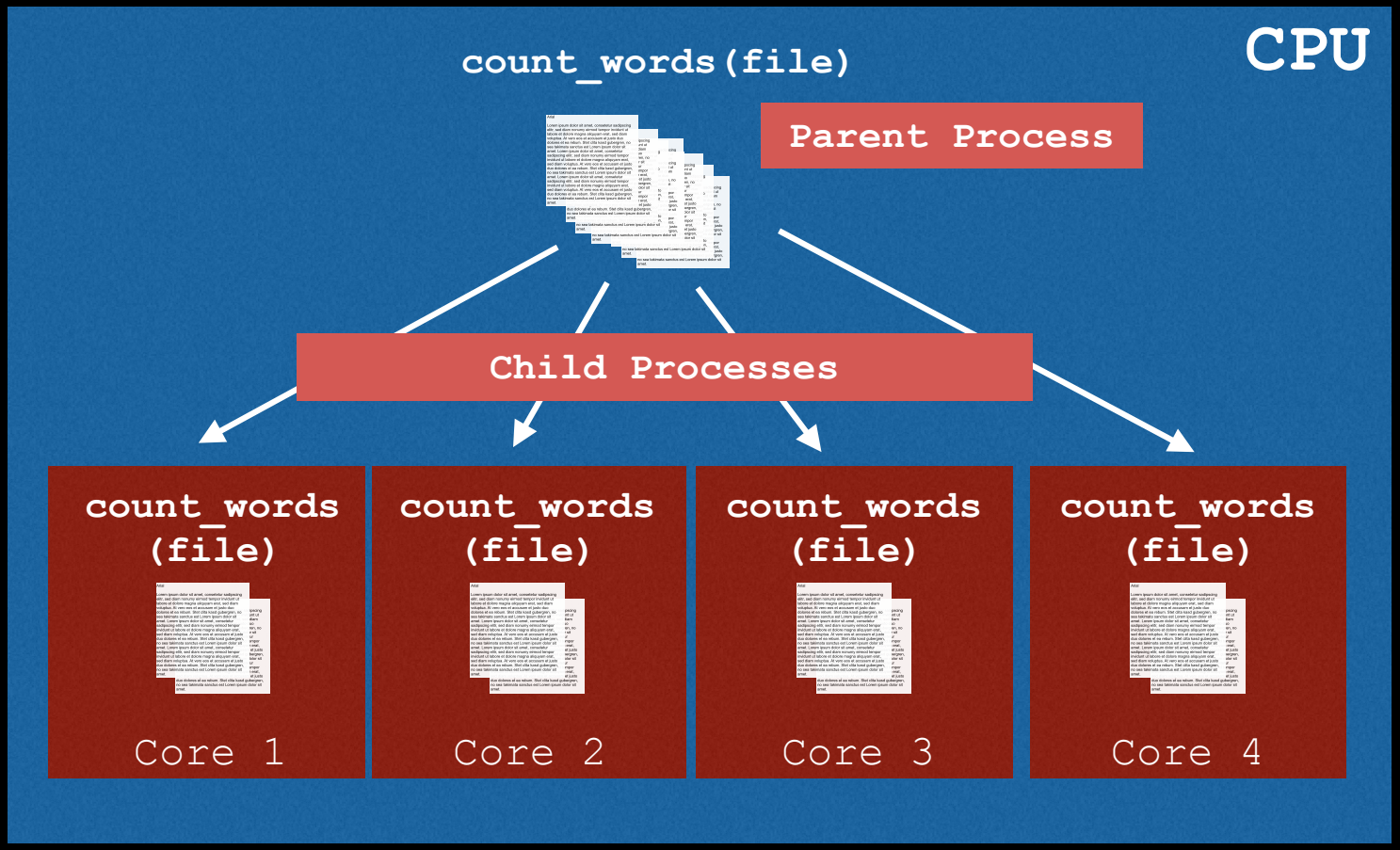
Checkpoint

- **You should now understand:**
  - ★ CPU
  - ★ Memory
  - ★ Program and Process

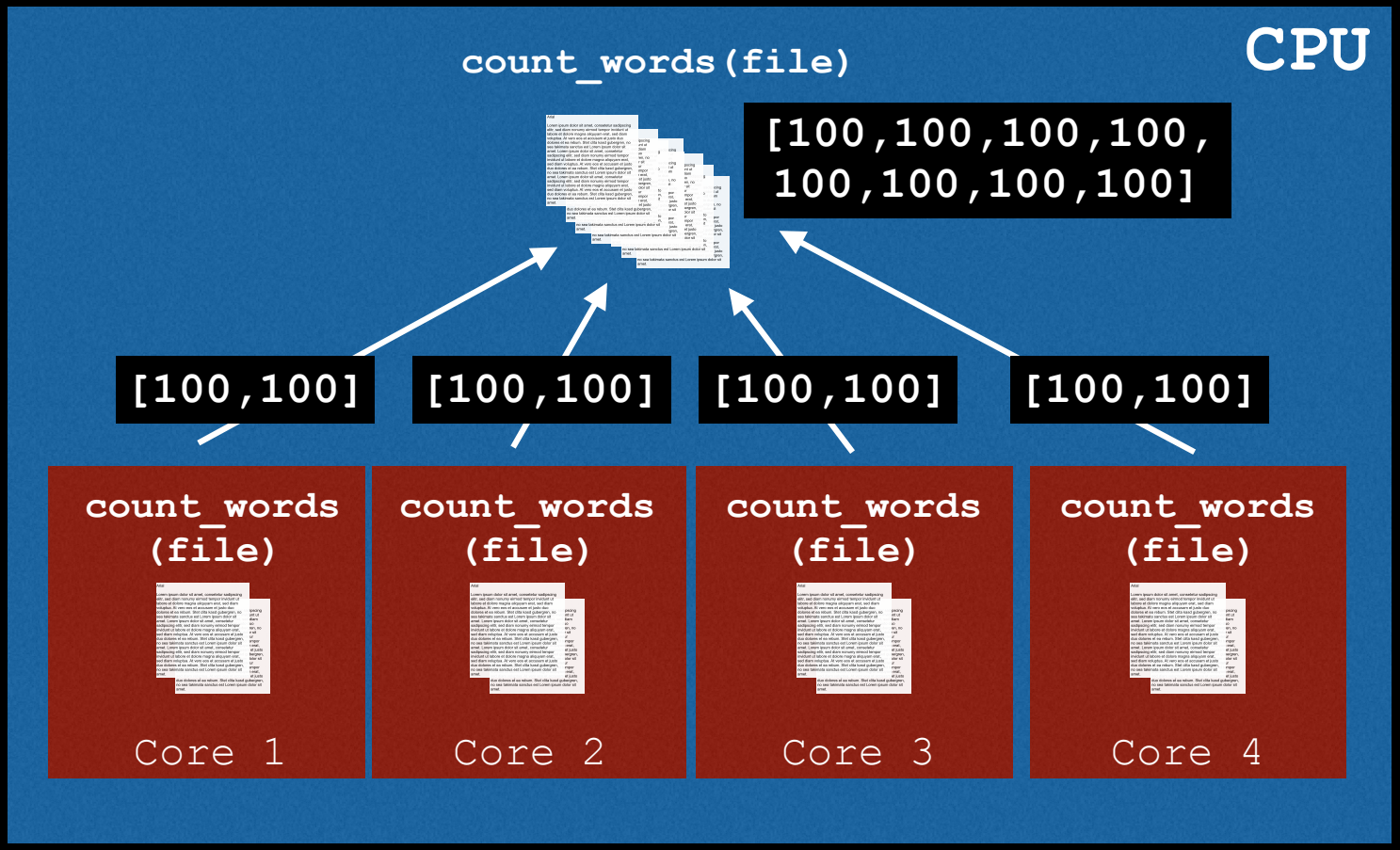
Goals

- Motivation
- Intro to computing resources
- Multi-core processing
  - ★ Parallelism
- Threading
  - ★ Concurrency

Parallelism
Multicore Processing



Parallelism
Multicore Processing

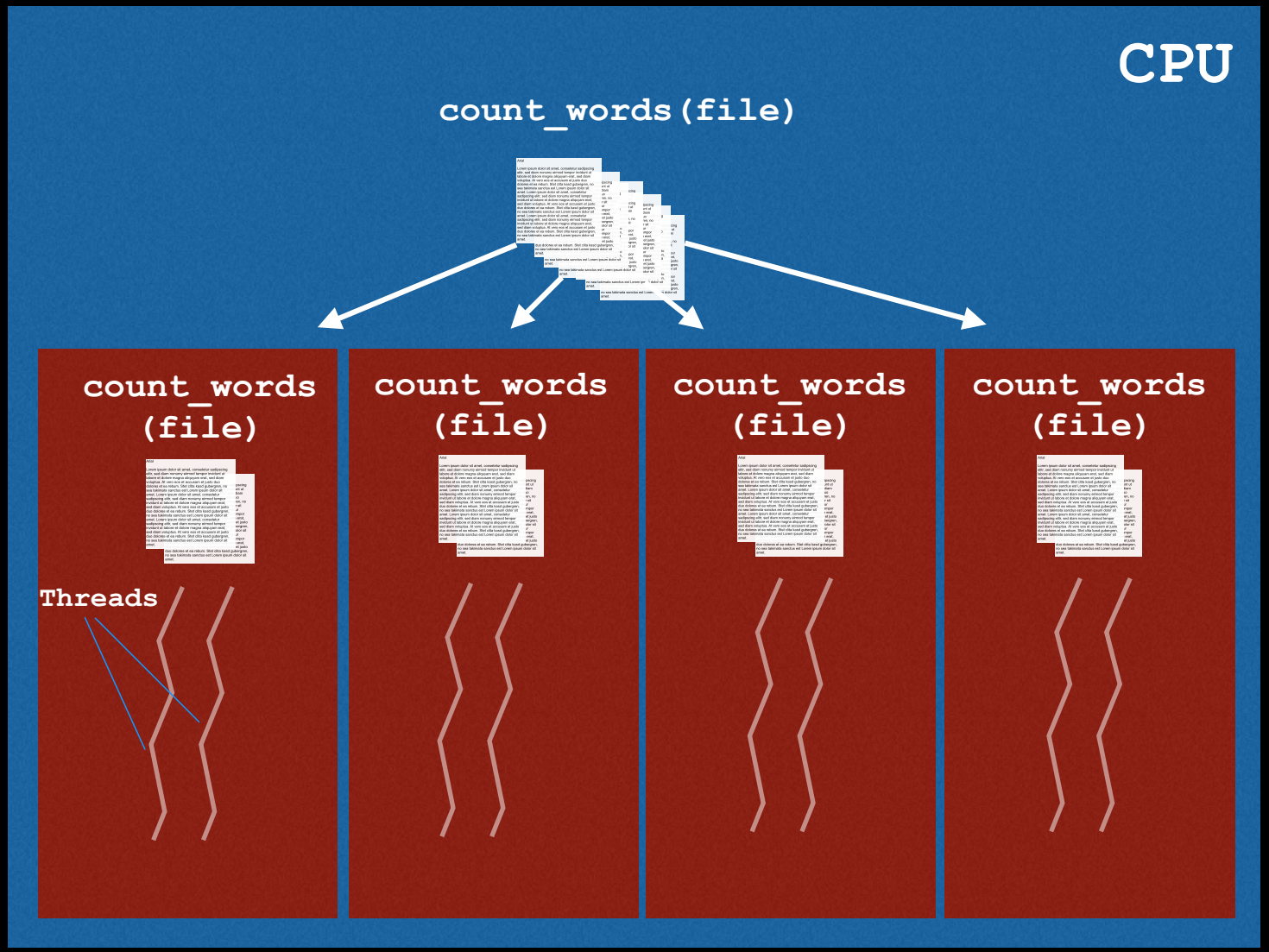


```python
from multiprocessing import Pool
import os

# Count the number of words in a file
def word_count(f):
    return len(open(f).read().split())

# Use a regular loop to count words in files
def sequential_word_count(lst_of_files):
    return sum([word_count(f) for f in lst_of_files])

# Use multiple cores to count words in files
def parallel_word_count(lst_of_files):
    pool = Pool(processes=4)
    results = pool.map(word_count, lst_of_files)
    print results
    # [100,100,100,100,100,100,100,100] # 8 files
    return sum(results)
```

Goals

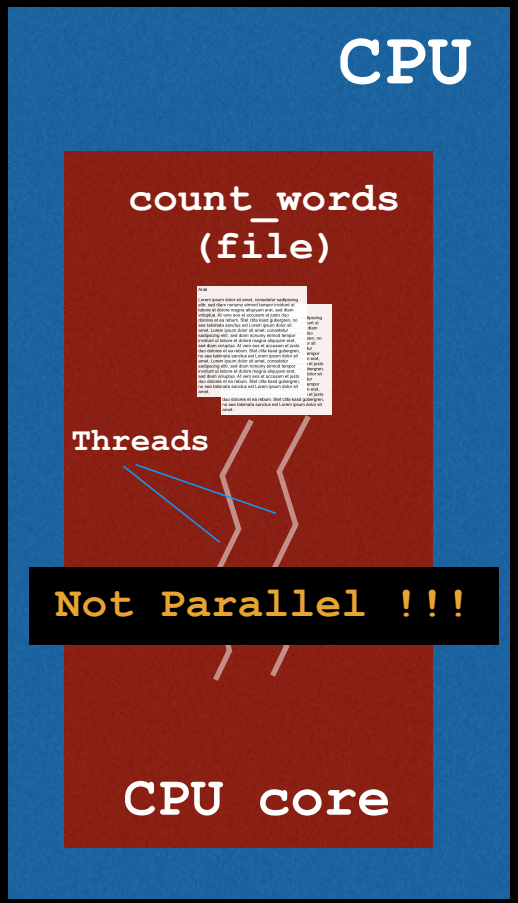
- Motivation
- Intro to computing resources
- Multi-core processing
  - ★ Parallelism
- Threading
  - ★ Concurrency

Concurrency
Threading



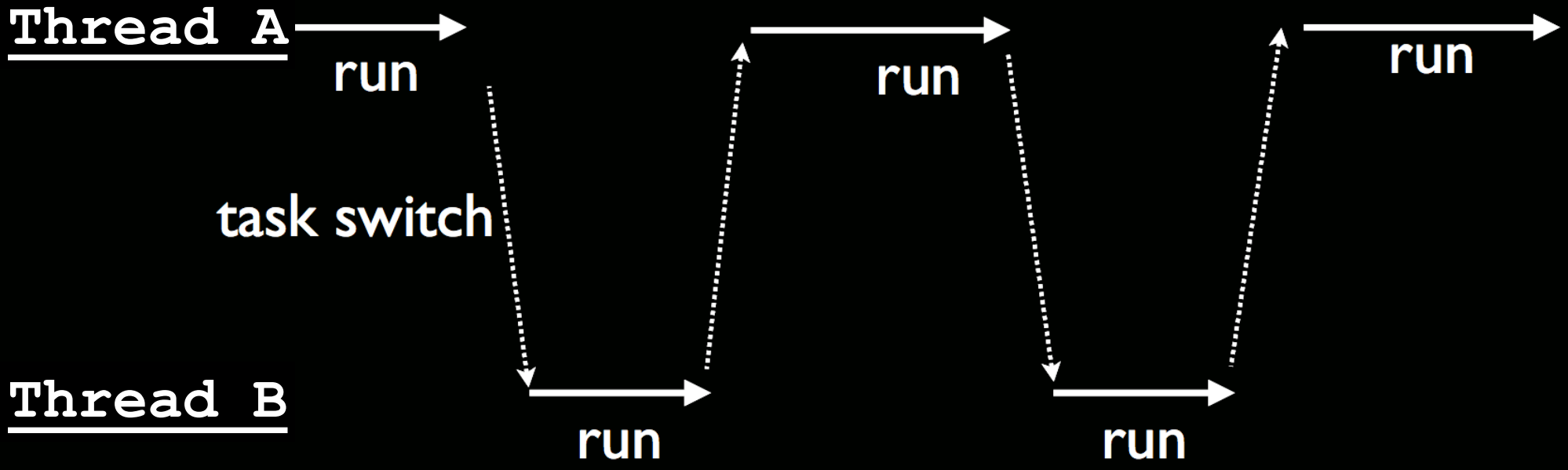
Threads

- Threads within ***the same process*** are ***not parallel***



Concurrency

- Threads **in the same process / core** are **concurrent**
- **1 process switching between multiple tasks**
- Not multi-core (multiple processes)

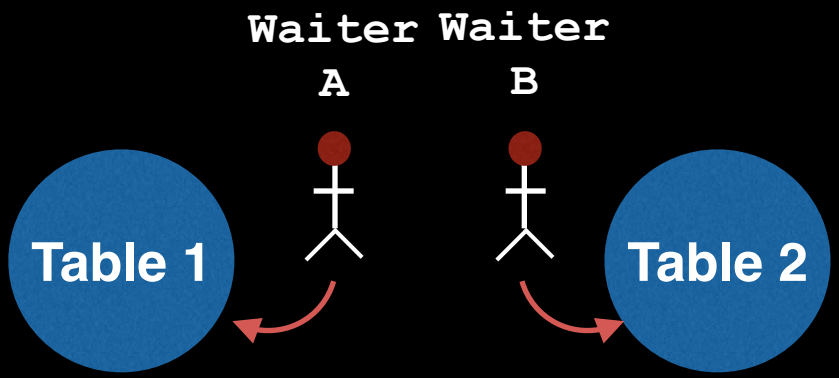

Thread A
run
run
run
task switch
Thread B
run
run

Sharing between Threads

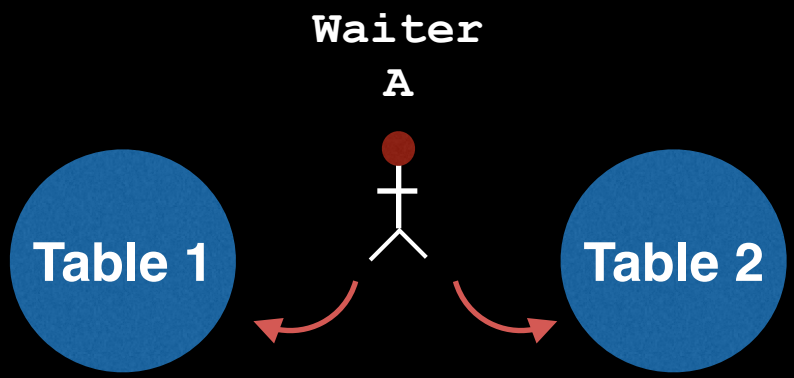
- **Share with other threads in the process:**
  - Memory allocated to the process
  - Code of the process
  - Context (i.e. variables)

Parallelism vs Concurrency
Analogy

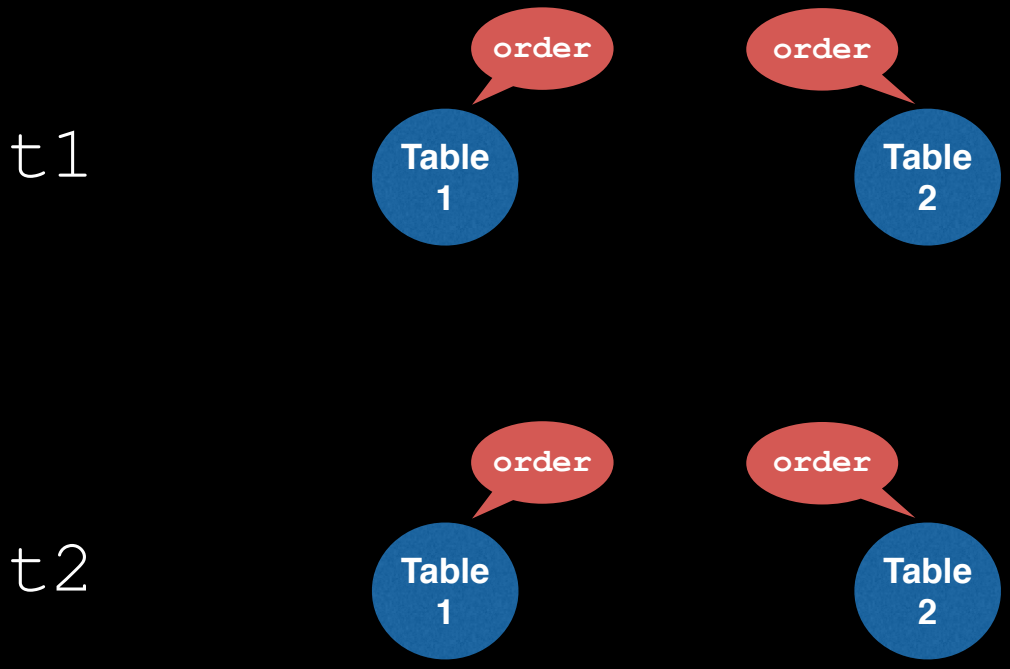
Parallelism



Concurrency



Parallelism Use Case



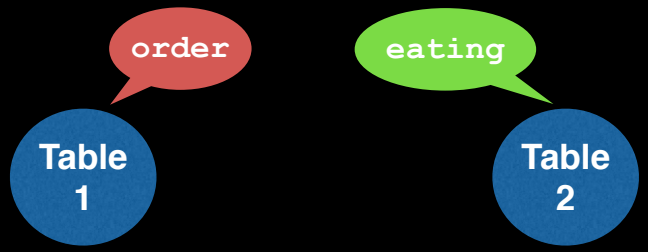
Parallelism Use Case

- **Table 1** and **Table 2** both order dishes all the time and constantly need service
- CPU-bound problem

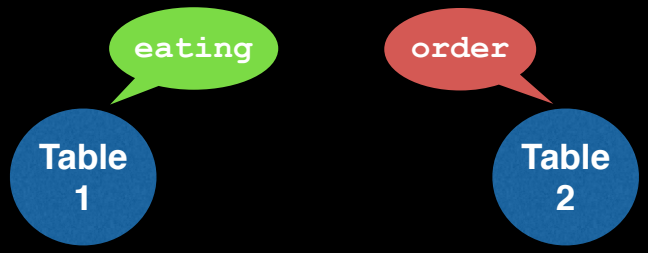
Tasks that use heavy computations with no wait time in between

Concurrency Use Case

t1



t2



Concurrency Use Case

- **Table 1** and **Table 2** order and can only order again when each finish the dish
- The wait time allow 1 waiter to service 2 tables
- I/O bound problem
    - **Read / Writing Files**
    - **Making Web Requests**

Python Library
Threading

```
import threading

jobs = []

# Initiate and Start Threads
for i in range(num_threads):
    t = threading.Thread(target=target_function, args=(arg1, arg2))
    jobs.append(t)
    t.start()

# "join" to make sure wait until the thread terminates
results = []
for t in jobs:
    t.join()
    # Access the result of the thread (if any)
    # Append to list
    results.append(t.result)
```

Summary

- Parallelism on multiple cores
- Threading within a core
- **`multiprocessing`** for parallelism
- **`threading`** for concurrency
- Parallelism —> CPU-bound problems
- Concurrency —> I/O-bound problems