Fundamentals of Programming and Computations
CS 15-112

Advanced Python Topics: Distributed Computing

April 21, 2024

Hend Gedawy

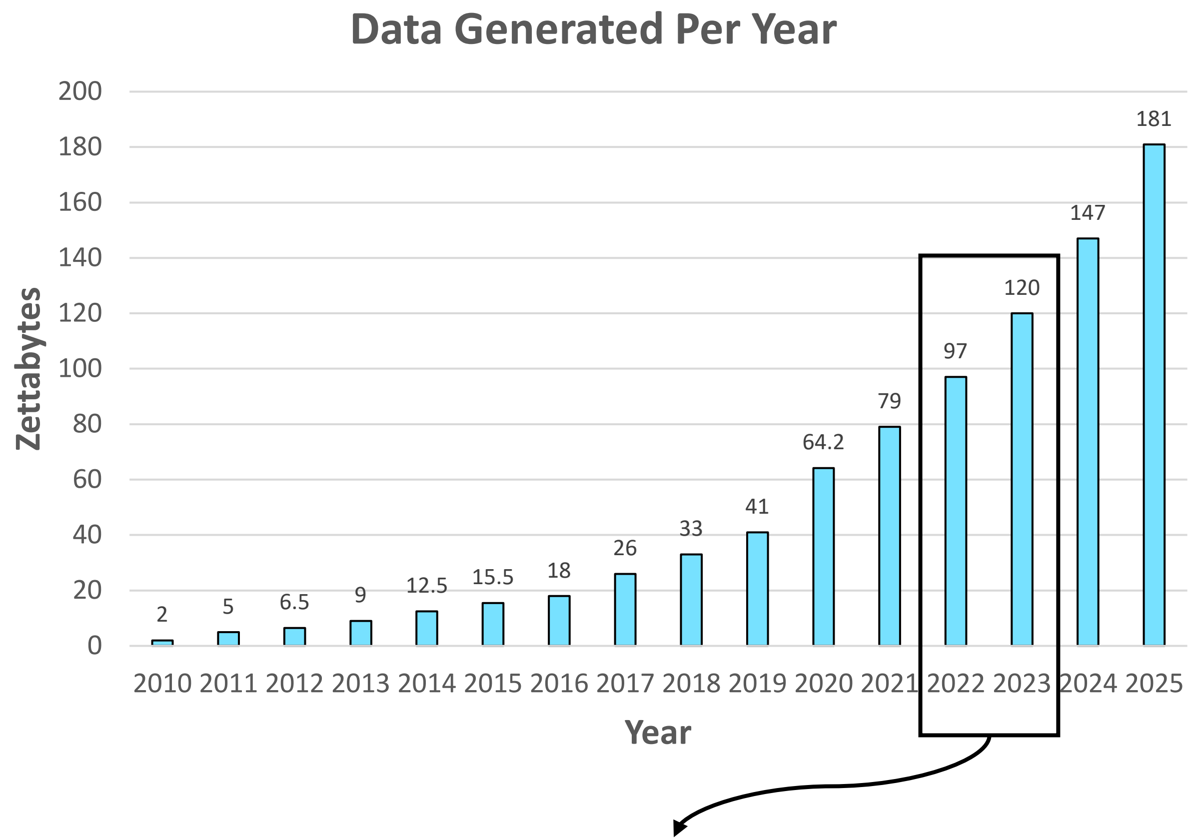
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

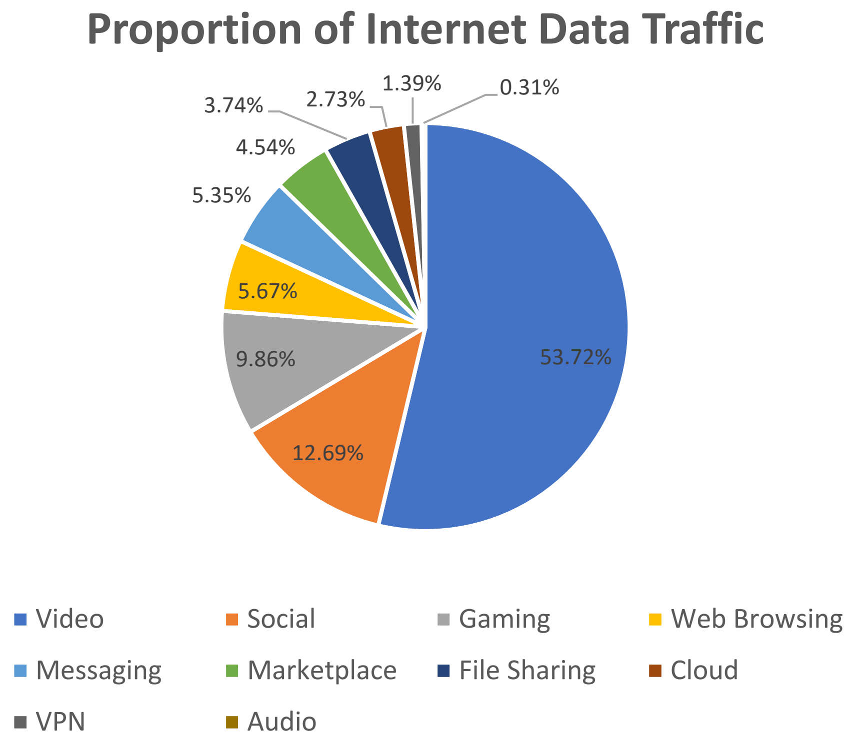
Outline

- Motivation
- Examples of distributed systems
- Building and running a distributed computing system
- Demo

A Common Theme is Data

A Common Theme is Data





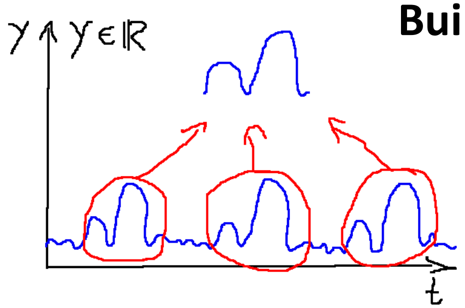
~70% of the world's data was generated only over the past two years

NOTE: Slide borrowed from 15-440 Distributed Systems course offered by Dr. Mohammed Hammoud

What Do We Do With All This Data

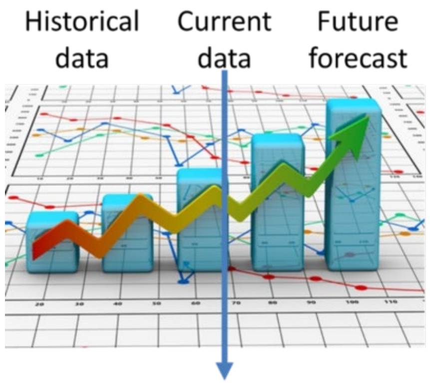
What Do We Do with This Data?

Build algorithms and techniques that enable computers to learn from this data and be able to :



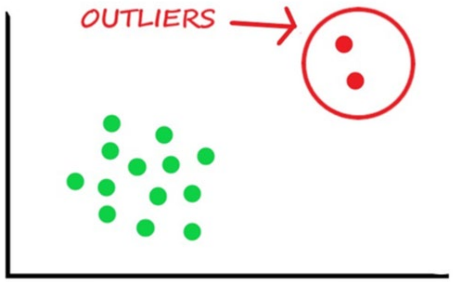
Recognize Patterns and relationships between data

(App. e.g. predicting and diagnosing diseases)



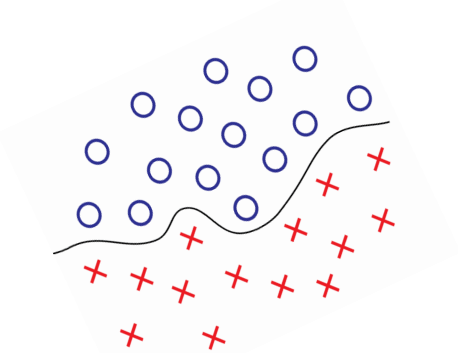
Make predictions about future events based on past observations

(App. e.g. forecasting energy demand)



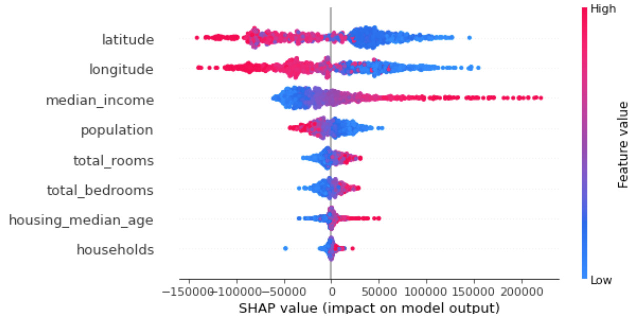
Identify anomalies and outliers

(App. e.g. detecting scams/misleading financial operations)



Classify data into different categories or classes based on their features

(App. e.g. better environment interaction and navigation for robots)

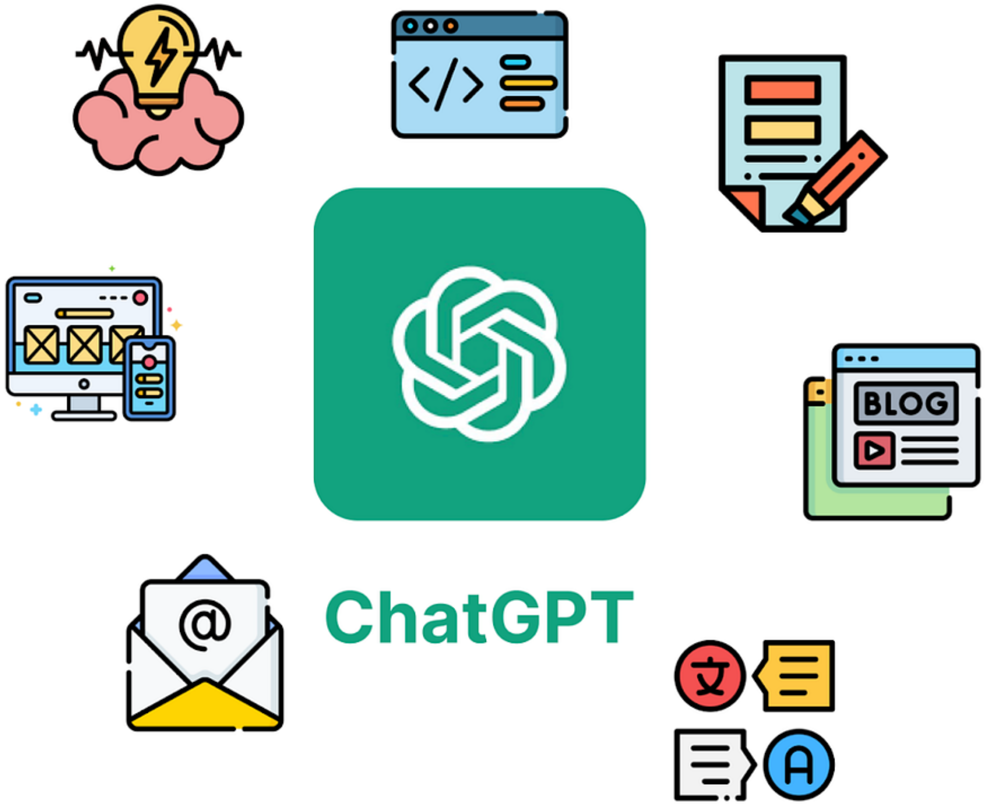


Determine the importance of different factors or variables in predicting the target outcome

(App. e.g. Identifying dominant factors for telecommunication customers churn)

Example: ChatGPT

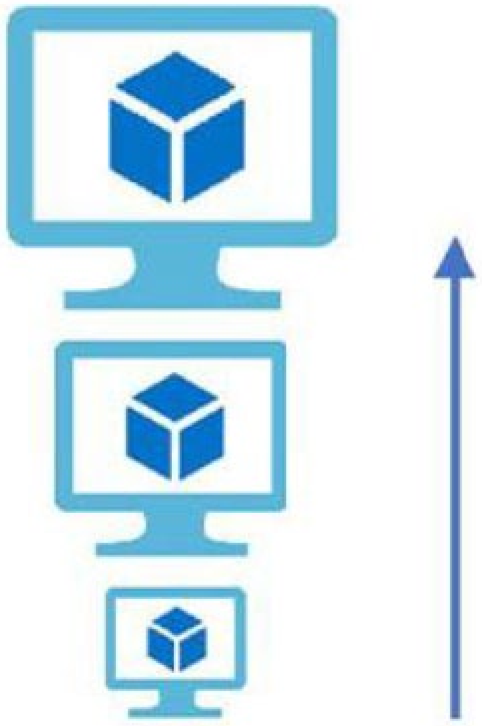
- AI language model developed by OpenAI
- Created using large-scale datasets obtained from various sources, including books, websites, and other texts
- It learned from this data to develop **a wide-ranging understanding of human language**.
    - generating human-like text responses to questions,
    - providing information,
    - engaging in conversations
    - offering assistance on a wide range of topics.



Where To Process All This Data?

Option 1: Hardware Upgrades

- E.g., faster CPU, more memory, and/or larger disk
  - This is What we Call Vertical Scaling



Option 1: Hardware Upgrades

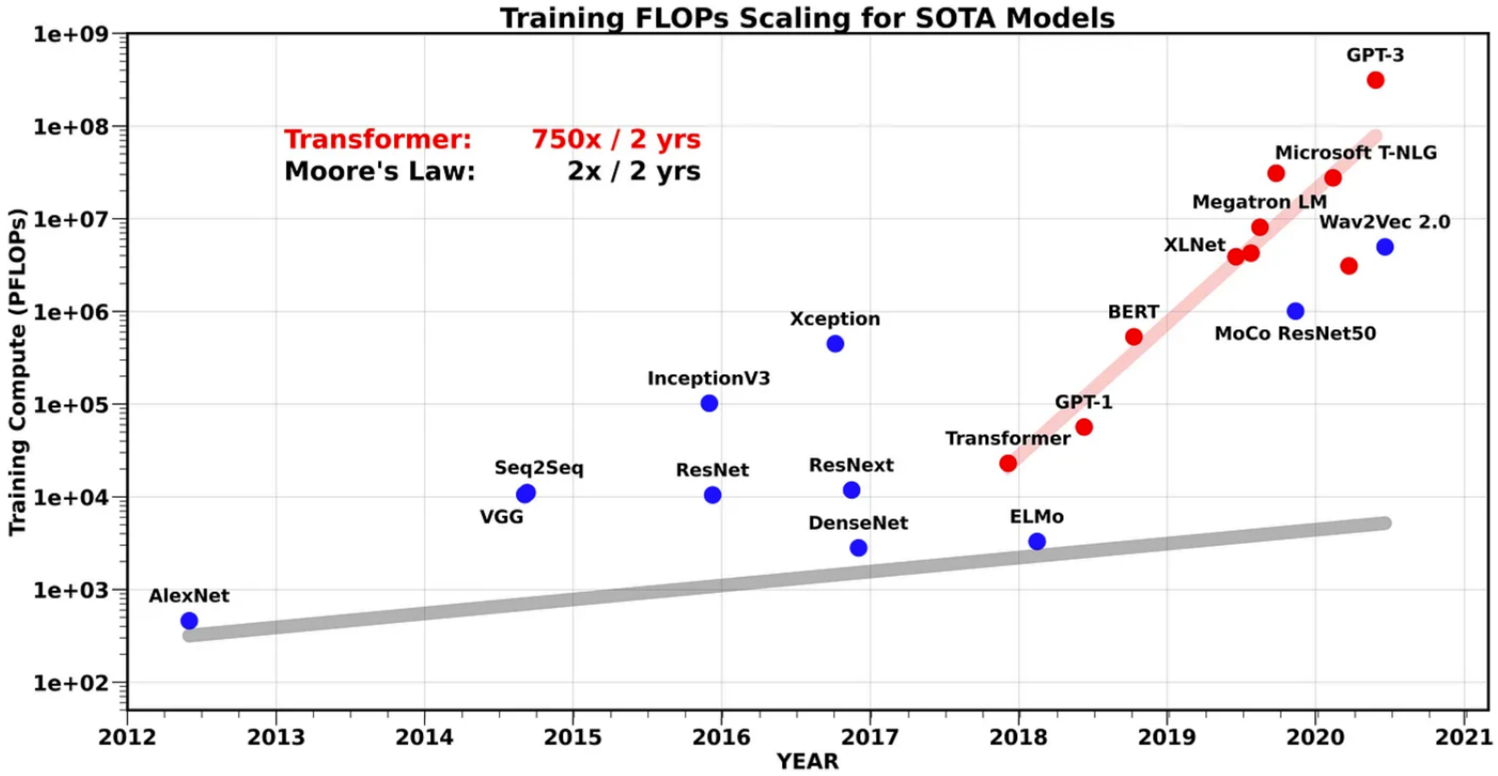


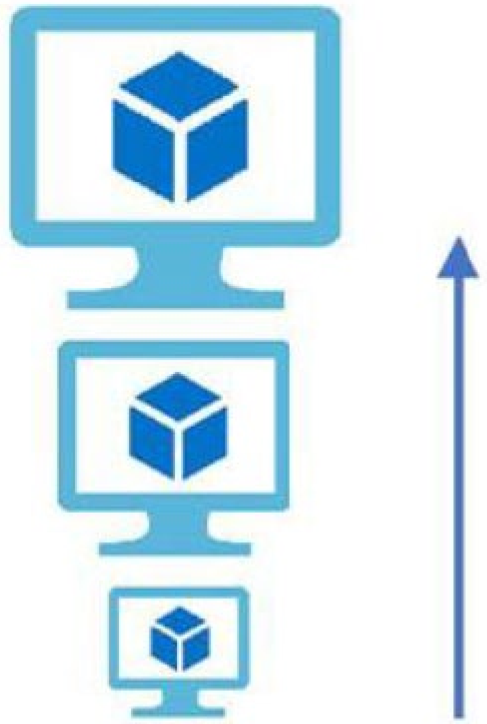
Figure 1: The amount of compute, measured in Peta FLOPs, needed to train SOTA models, for different CV, NLP, and Speech models, along with the different scaling of Transformer models (750x/2yrs)*1 [Source]

Individual computers still suffer from *limited resources* with respect to the scale of today's problems

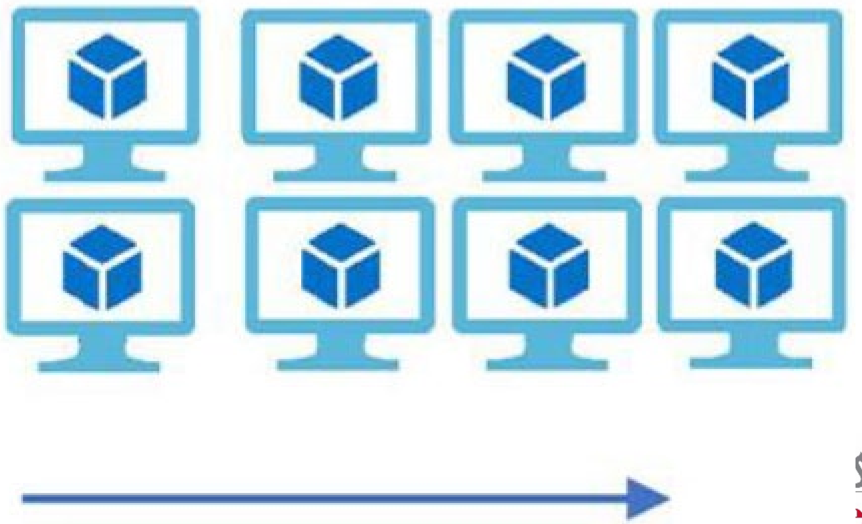
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Option 2: Adding More Machines

Option 1: Vertical Scaling
(Upgrade an Instance;
RAM, CPU, etc.)



Option 2: Horizontal Scaling
(Add more instances)



Option 2: Adding More Machines

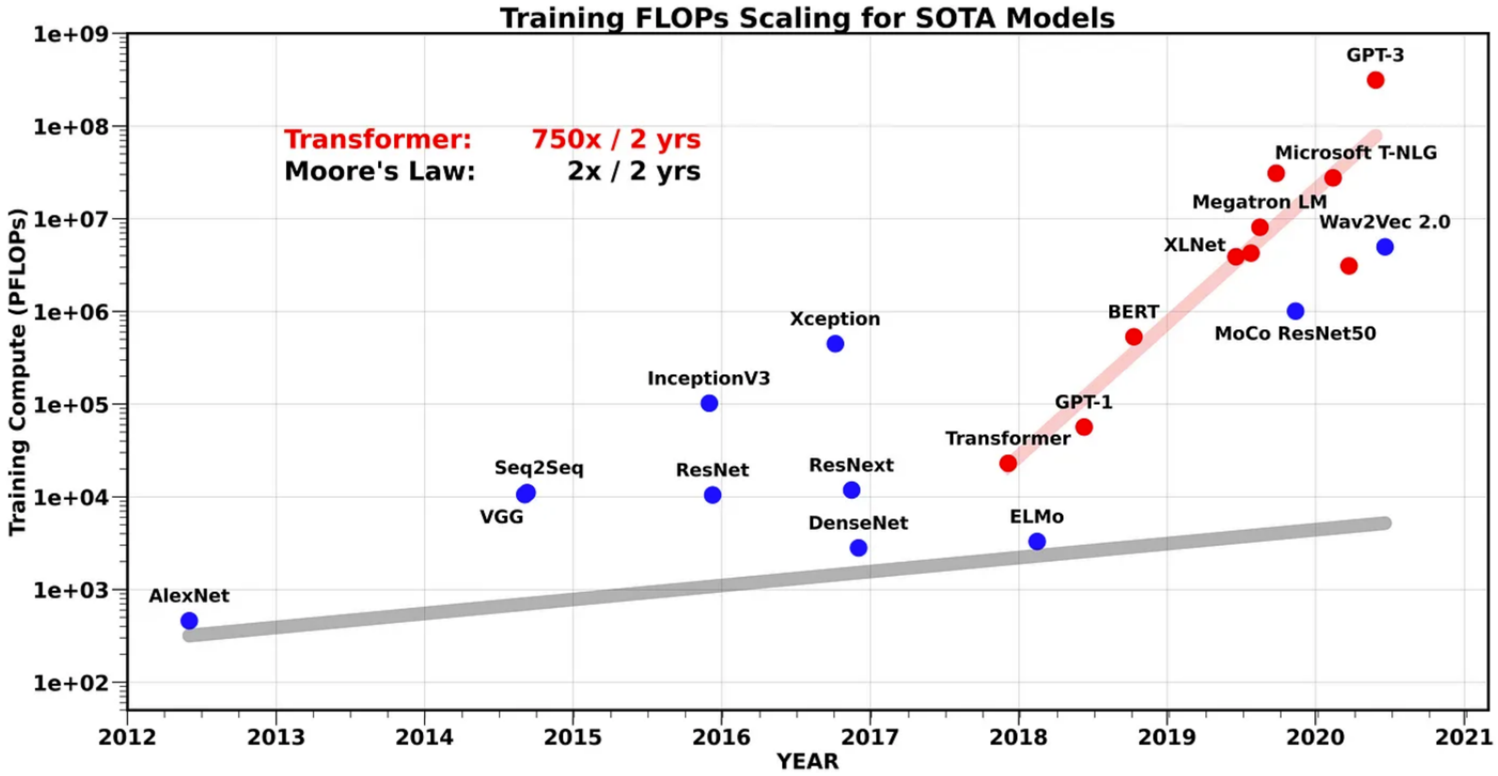


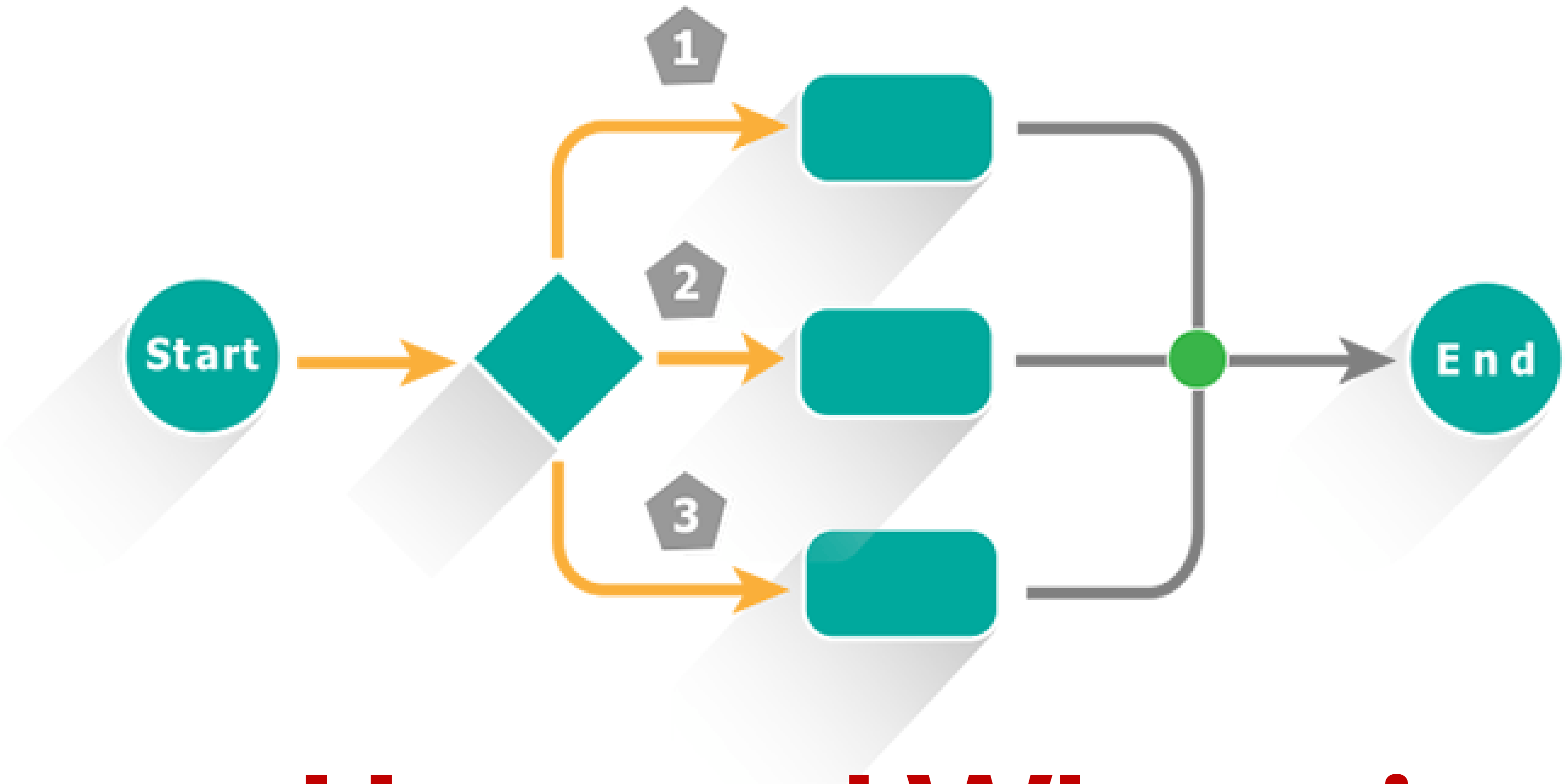
Figure 1: The amount of compute, measured in Peta FLOPs, needed to train SOTA models, for different CV, NLP, and Speech models, along with the different scaling of Transformer models (750x/2yrs)*1 [Source]

According to unverified information leaks, **GPT-4** was **trained** on about 25,000 Nvidia A100 GPUs for **90–100 days**.

Assuming that the GPUs were installed in Nvidia HGX servers which can host 8 GPUs each, meaning 25,000 / 8 = **3,125 servers** were needed.

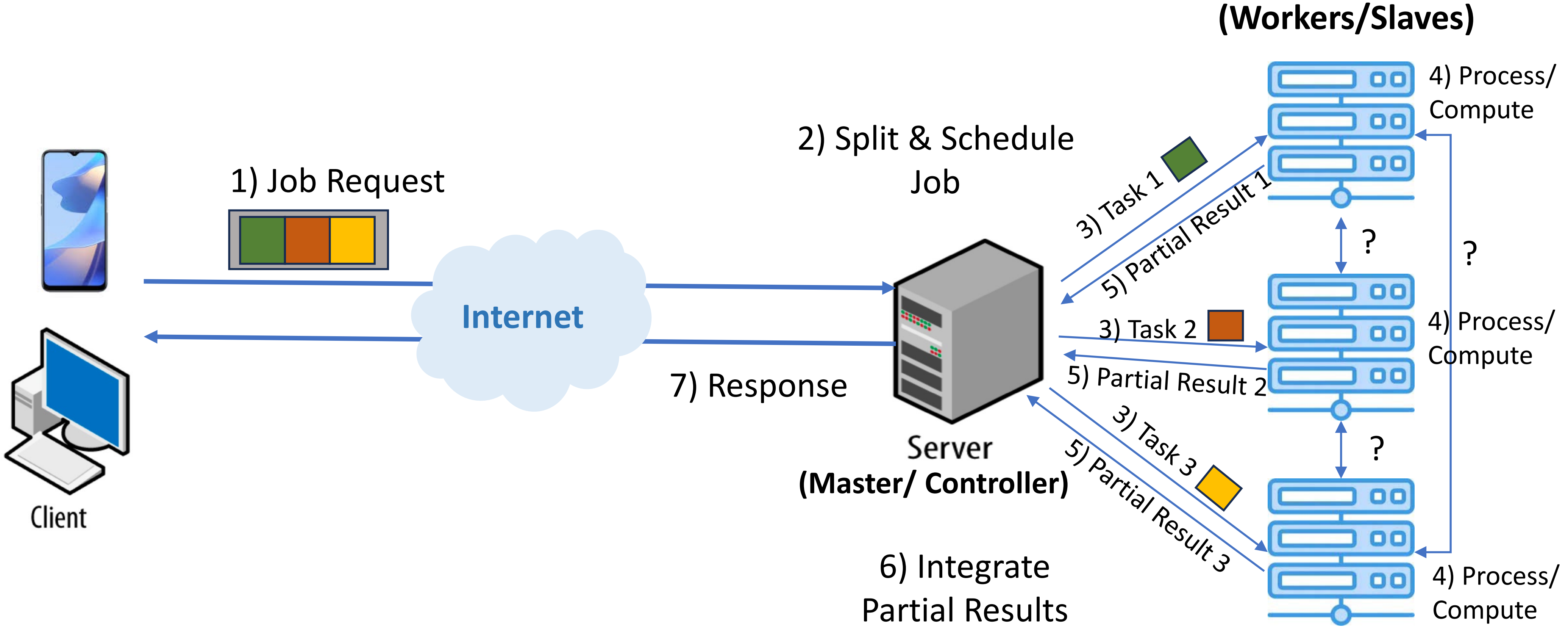
source

Examples of Systems that Leverage Distributed Computing



How and Where is Distributed Computing Run

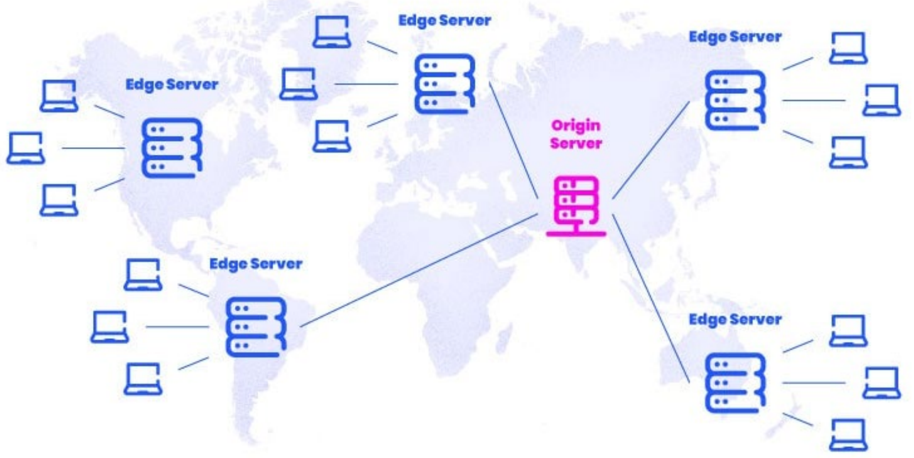
Distributed Computing Process



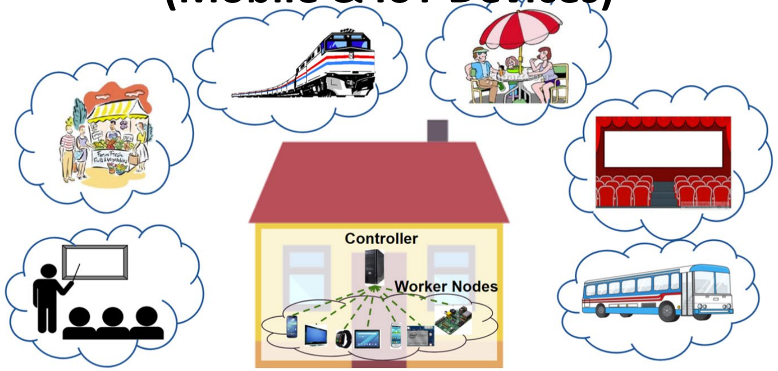
Where is Computing Distributed Hosted Different Paradigms

Cloud Computing (Data Centers)

Edge Computing (Edge Servers)



FemtoClouds, Mobile Device Clouds, or IoT Clouds (Mobile & IoT Devices)



What It Takes To Build A Distributed Computing System

Distributed Computing System Requirements

- A way to express the problem in terms of parallel processes and execute them on different machines (*Programming and Concurrency Models*)

- A way to organize processes (*Architectures*)

- A way for distributed processes to exchange information (*Communication Paradigms*)

- A way to locate and share resources (*Naming Protocols*)

NOTE: Slide borrowed from 15-440 Distributed Systems course offered by Dr. Mohammed Hammoud

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Distributed Computing Sytems Requirements

- A way for distributed processes to cooperate, synchronize with one another, and agree on shared values (*Synchronization*)

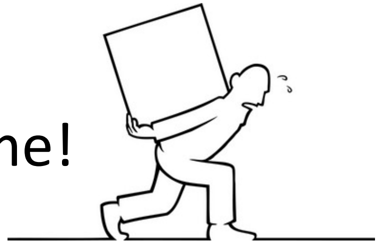
- A way to reduce latency, enhance reliability, and improve performance (*Caching, Replication, and Consistency*)

- A way to enhance load scalability, reduce diversity across heterogeneous systems, and provide a high degree of portability and flexibility (*Virtualization*)

- A way to recover from partial failures (*Fault Tolerance*)

Distributed Systems: Two Options

- We can create a custom distributed system (or program) for each new algorithm
  - Cumbersome!



- Or utilize modern distributed frameworks, which:
  - Relieve programmers from worrying about the many difficult aspects of distributed systems
  - Allow programmers to focus on ONLY the sequential parts of their programs
  - E.g., MapReduce (or *Hadoop*), **GraphLab (Turi later)**, and Ray

Framework

RAY

www.ray.io

Companies Using Ray - Examples

VISA

OpenAI ChatGPT 4.0

Microsoft

shopify

intel

Uber

NVIDIA

Alibaba.com

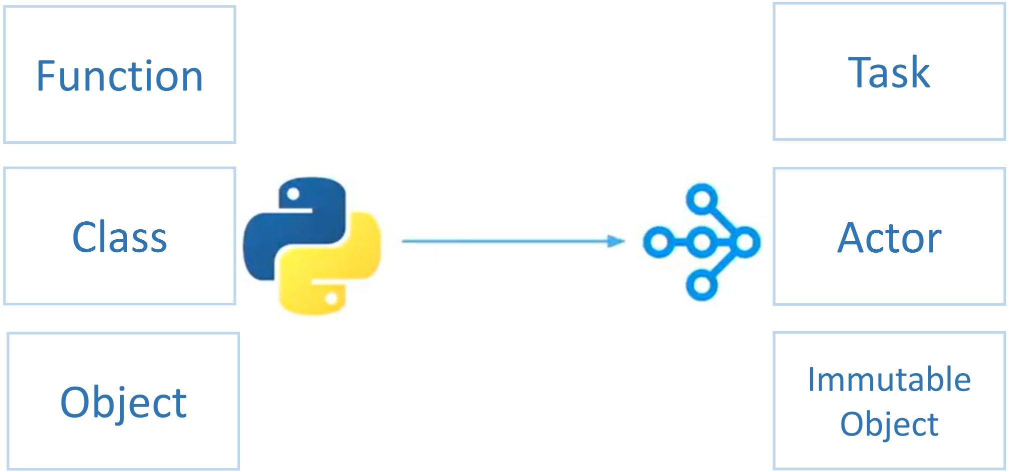
amazon

ERICSSON

instacart

Python - Ray

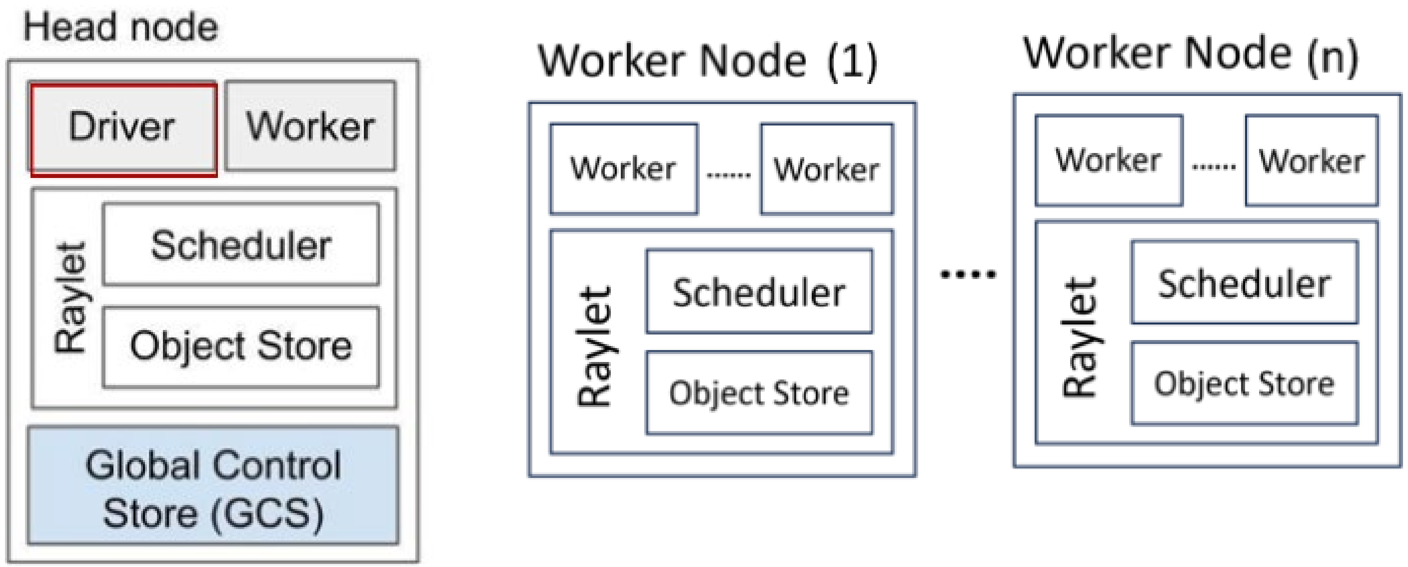
Ray API allows serial applications to be parallelized without major modifications.



Ray takes the existing concepts of **functions** and **classes** and translates them to the distributed setting as **tasks** and **actors**.


D E M O

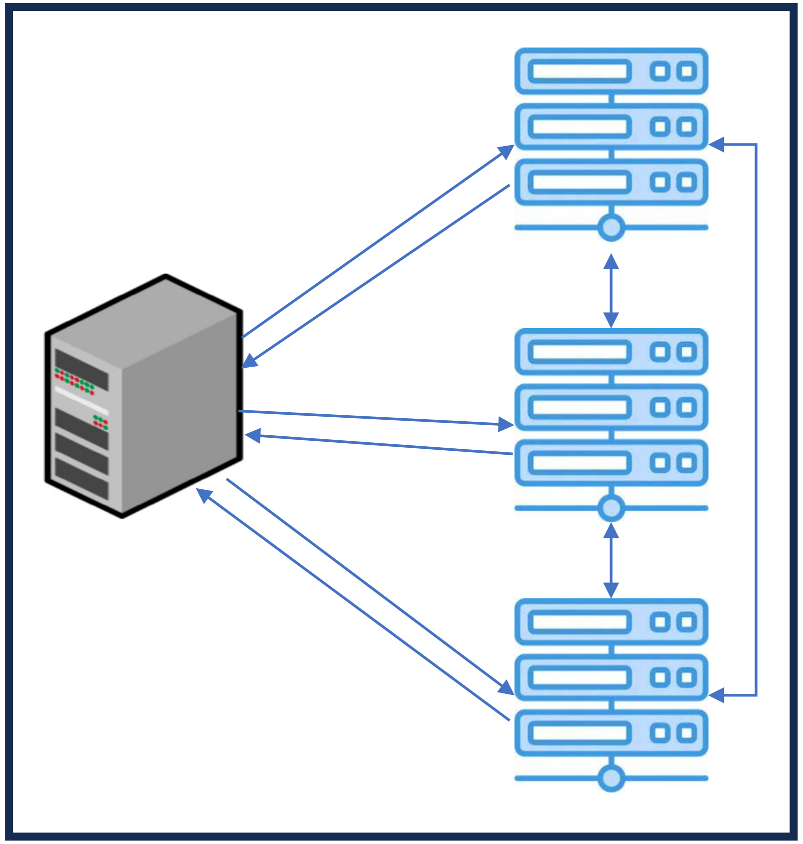
Ray Cluster: Master-Slave Architecture



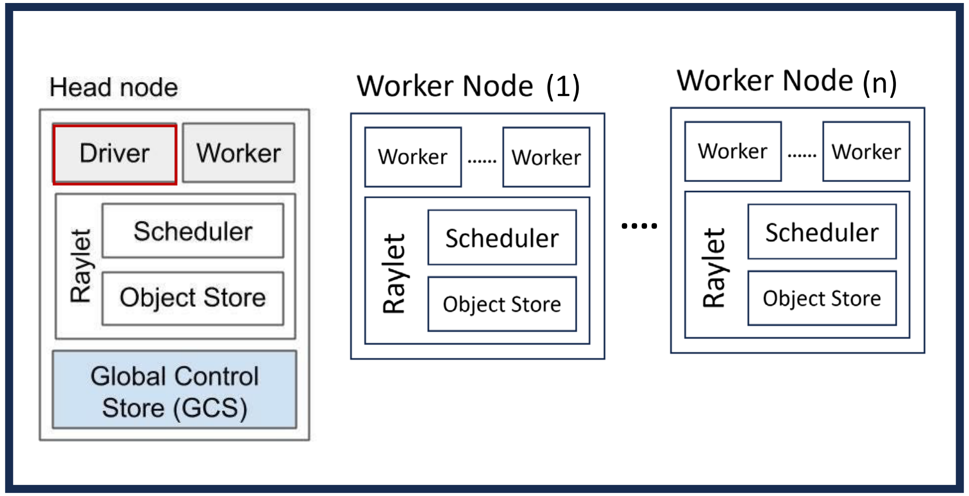
Setting Up & Running A Program on Ray Cluster

- Make sure ray is stopped in all nodes (sudo ray stop --force)
- **Start Ray @ Head Node:**
  - sudo ray start --head --include-dashboard 1 --dashboard-host 0.0.0.0
- **Include more worker machines**:
  - ssh to the worker node and start ray using the following command:
    - sudo ray start --address=‘headNodeIP:headPortNum’
- Run the program @Head Node:
    - sudo python3 <Program python file> <Program Parameters>
- **To view the dashboard of your cluster, go to your web browser and put**
  headNodeIP:dashboardPortNumber ← Given when head started
- When Done, run (sudo ray stop --force) on all nodes

Our Ray Cluster – Setup



Distributed System



Ray Cluster

Ray Dashboard

Overview | Jobs | Serve | Cluster | Actors | Metrics | Logs

NODES

Auto Refresh:

Request Status: Node summary fetched.

Node Statistics

TOTALx 4 | ALIVEx 4

Node List

Host | IP | State | Page Size | Sort By | Reverse: | TABLE | CARD

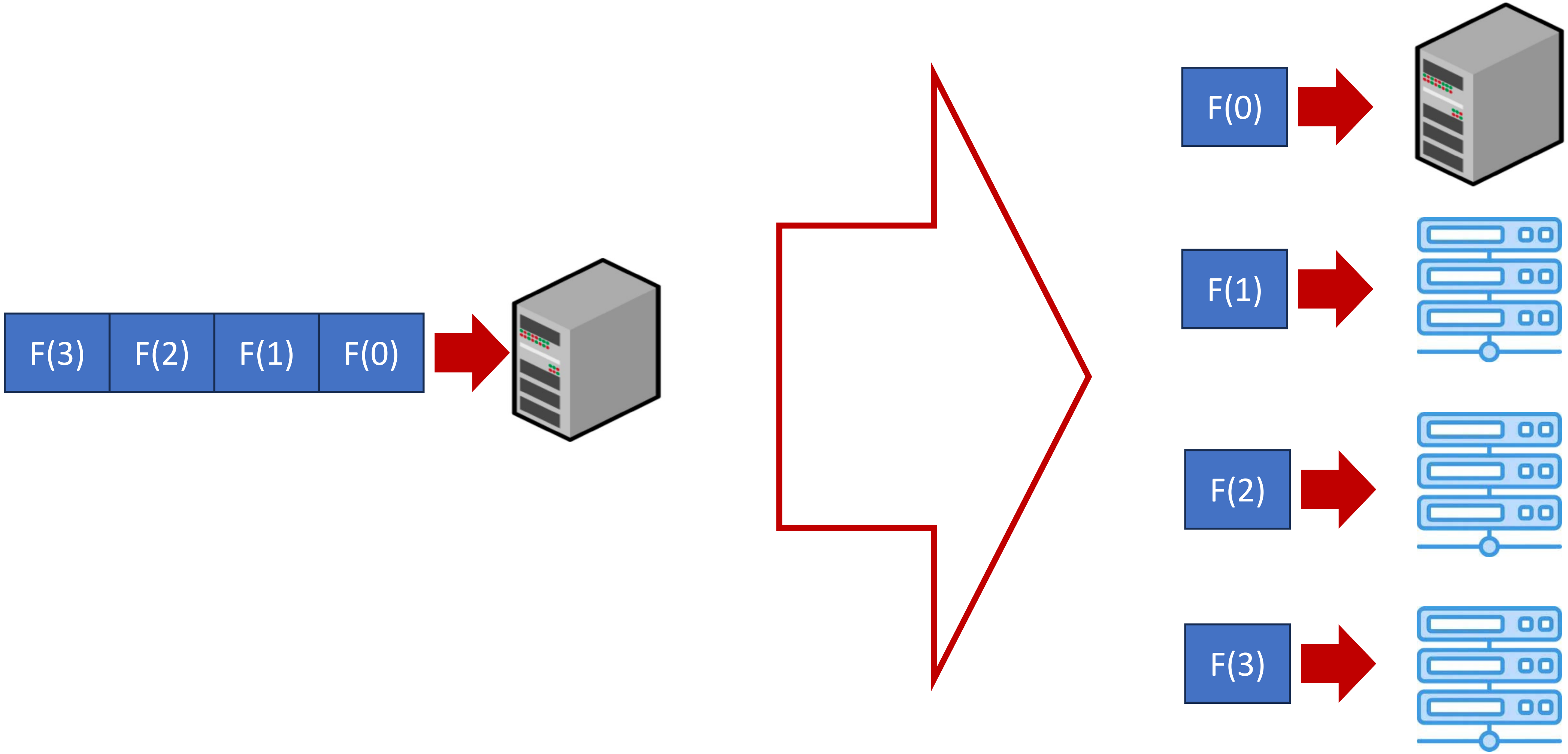
1

	Host / Worker Process name	State	ID	IP / PID	Actions	CPU	Memory	GPU	GRAM	Object Store Memory	Disk(root)	Sent	Received	Logical Resources
>	15440-ray-01	ALIVE	defeb...	172.20.247.85 (Head)	Log	2%	860.19MB/7.56GB(11.1%)	N/A	N/A	0.0000B/2.11GB(0.0%)	15.92GB/18.60GB(90.3%)	10.07KB/s	6.34KB/s	0.0/4.0 CPU 0B/4.21... Expand
>		ALIVE	16646...		Log	0%		N/A	N/A	0.0000B/2.16GB(0.0%)		0.0000B/s	0.0000B/s	0.0/4.0 CPU 0B/5.04... Expand
>		ALIVE	40e2f...		Log	0%		N/A	N/A	0.0000B/2.16GB(0.0%)		0.0000B/s	0.0000B/s	0.0/4.0 CPU 0B/5.04... Expand
>	15440-ray-02	ALIVE	7a8c3...	172.20.247.86	Log	0.8%	439.58MB/7.56GB(5.7%)	N/A	N/A	0.0000B/2.16GB(0.0%)	5.19GB/18.60GB(29.4%)	5.02KB/s	1.96KB/s	0.0/4.0 CPU 0B/5.05... Expand

Sequential to Parallel in Ray

```python
import time

def f(i):
    time.sleep(1)
    return i

t1= time.time()

results=[]
for i in range(4):
    results.append(f(i))

print("results: ", results)
print("sequential time: ", time.time()-t1)
```

Sequential to Parallel in Ray



جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Sequential to Parallel in Ray

```
import ray
import time

#start Ray
ray.init()

@ray.remote
def f(i):
    time.sleep(1)
    return i


start_time= time.time()

futures= [f.remote(i) for i in range(4)]

# Continue executing other tasks or operations
print("DOING OTHER TASKS")
print("curr time: ", time.time()- startTime)

# Retrieve the result of the future
result = ray.get(futures)
print("Result:", result)

print("completion time: ", time.time()-start_time)
```

Remote function: can be executed remotely and asynchronously

4 copies of f are executed in parallel (on different machines).

The function call returns immediately a reference to the eventual output

The actual function execution (i.e. Task) is taking place in the background

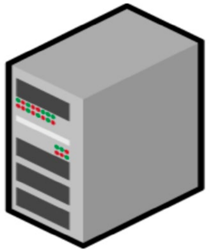
It is called a **future**, because we don't get the output immediately, we get a promise that we will get the output at some point

Application – Most Frequent Word (Sequential Implementation)

```python
import time

def mostFrequentWordv1(wordList):
    maxword = None
    maxcnt = 0
    for word in wordList:
        cnt = wordList.count(word)
        if cnt > maxcnt:
            maxcnt = cnt
            maxword =word
    return (maxword, maxcnt)





def loadBook(filename):
    with open(filename,"r", encoding='utf-8') as f:
        theText = f.read()
    toRemove = [",",";",".","\"","\' ","\n\'"," \'","!","(",")","?","--","'",'"',"]","[",":",'*']

    theText = theText.lower()

    for badChar in toRemove:
        theText = theText.replace(badChar," ")
    return theText.split()

allWordsList = loadBook("alice.txt")
print(f"Loaded text with {len(allWordsList)} words")

start = time.time()
ans = mostFrequentWordv1(allWordsList)
end = time.time()
elapsed1 = end - start
print("List-based\nAnswer {} in {:0.4f} seconds".format(ans, elapsed1))
```

Application – Most Frequent Word (How to Parallelize)

List of all words in the document (l)



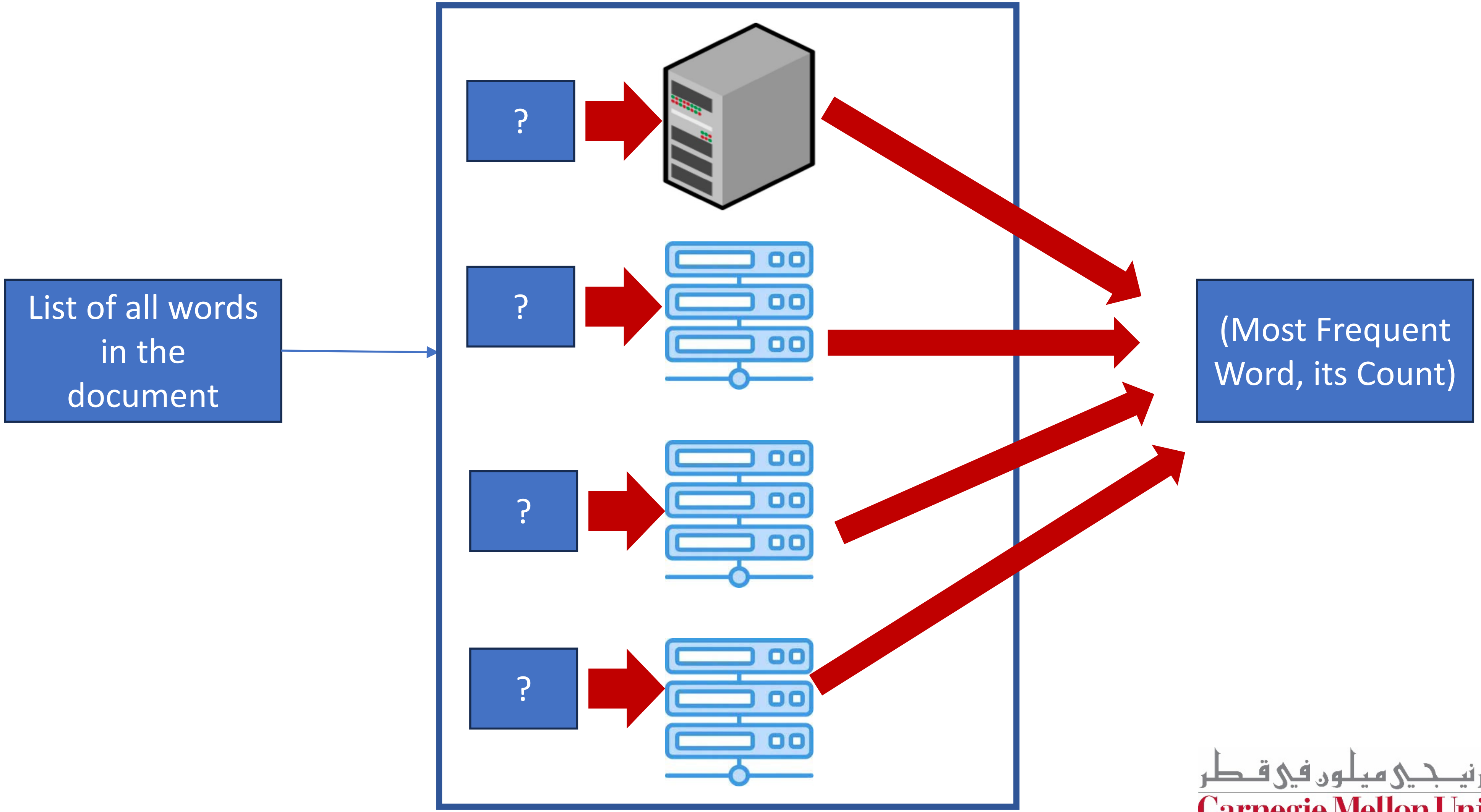
mostFrequentWord(l)

(Most Frequent Word, its Count)

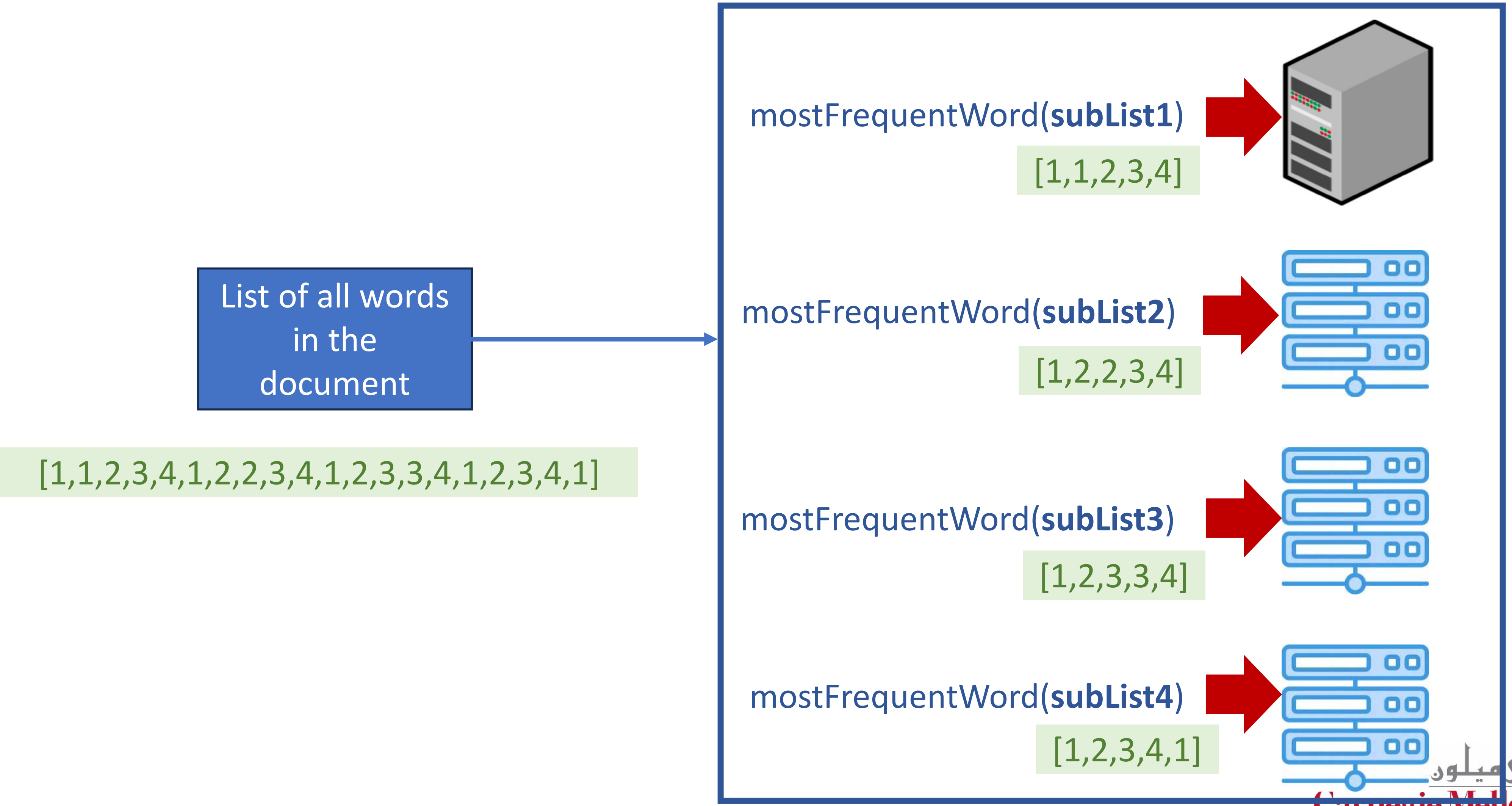
[1,1,2,3,4,1,2,2,3,4,1,2,3,3,4,1,2,3,4,1]

(1,6)

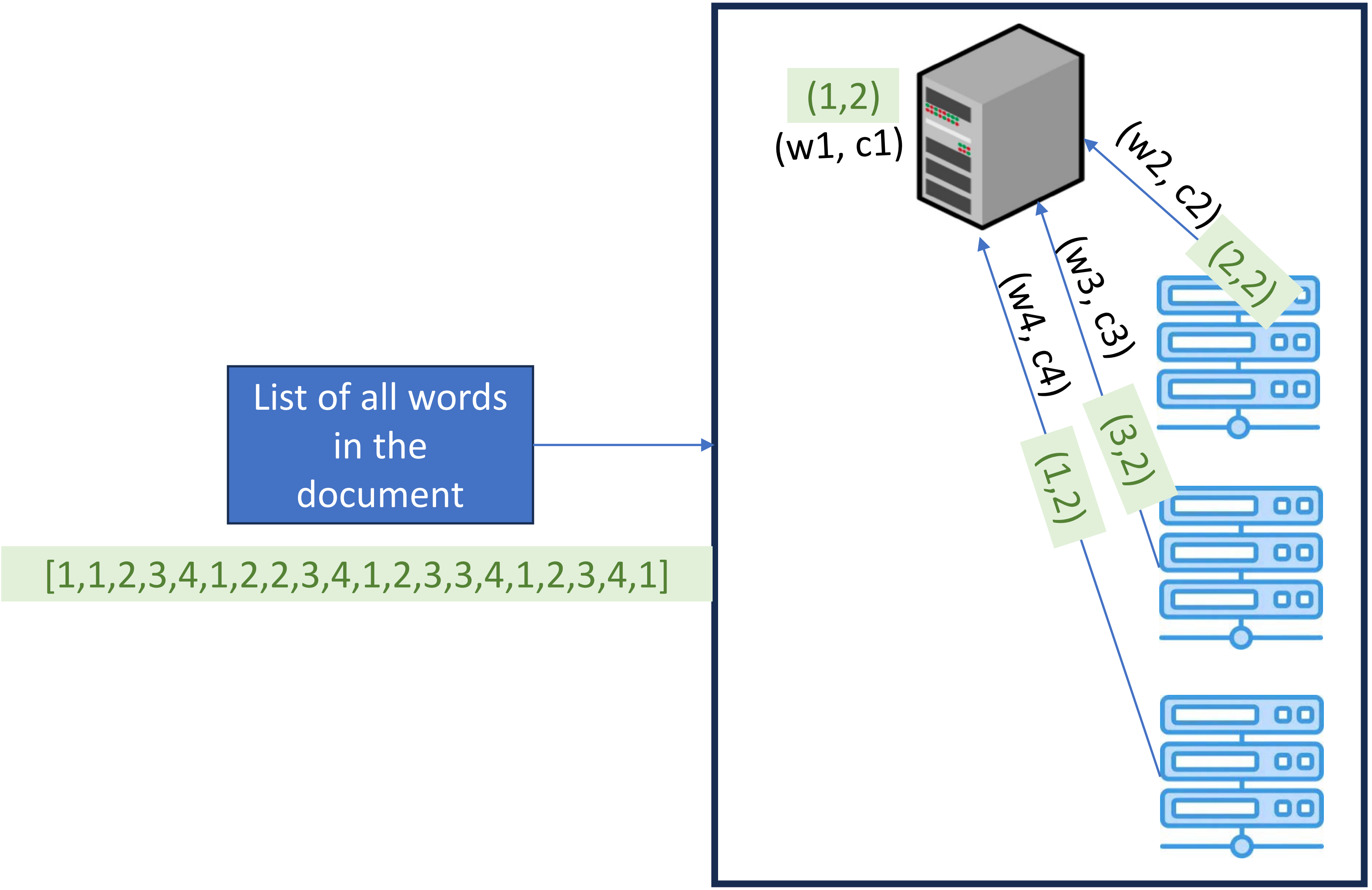
Application – Most Frequent Word (How to Parallelize)



Application – Most Frequent Word (How to Parallelize)



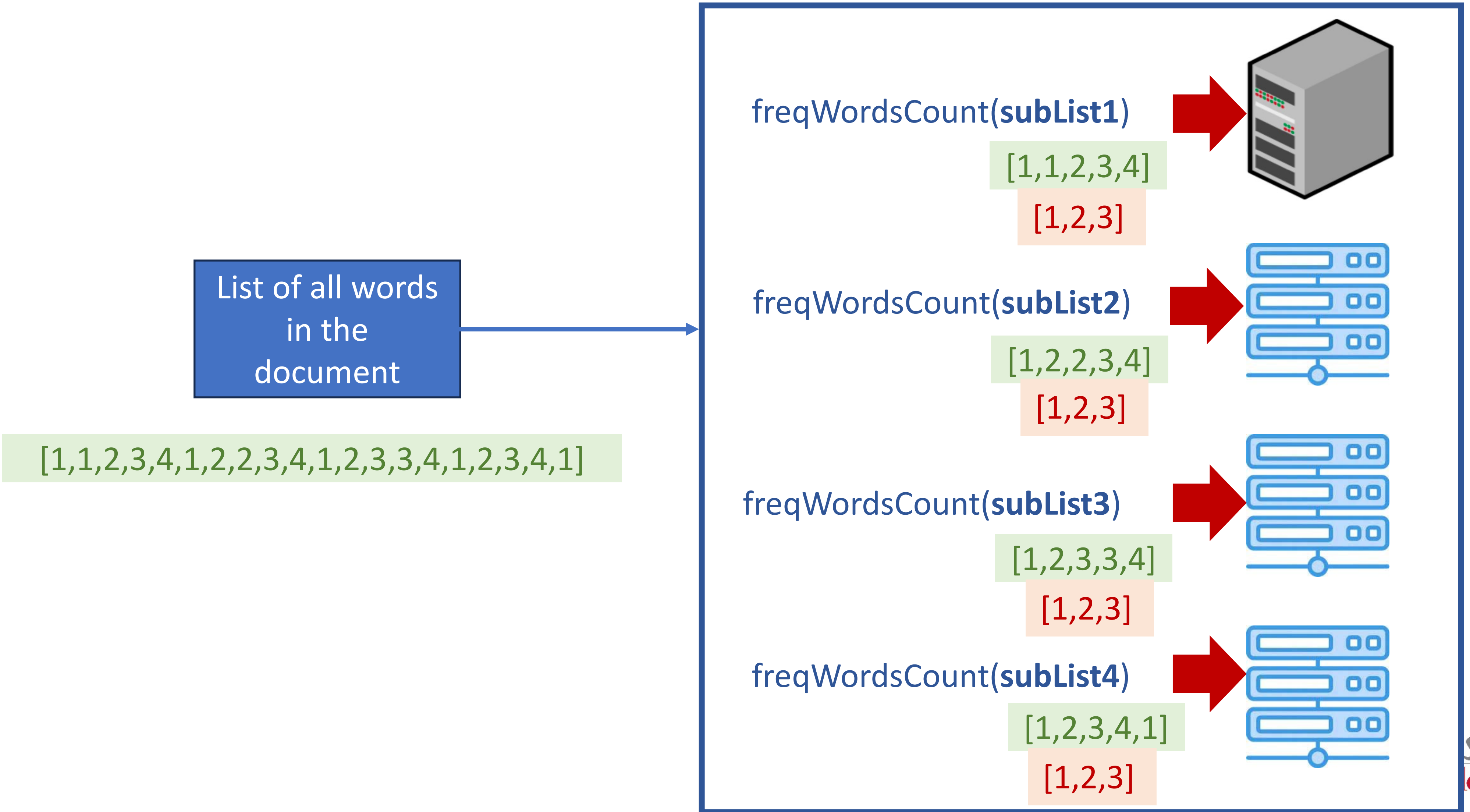
Application – Most Frequent Word



Each Node Returned the most Common Word and its count in the assigned sublist

Now we need to know the count of each of these most common words [1,2,3] in the sublist assigned to each node (Another Distributed Computing Round)

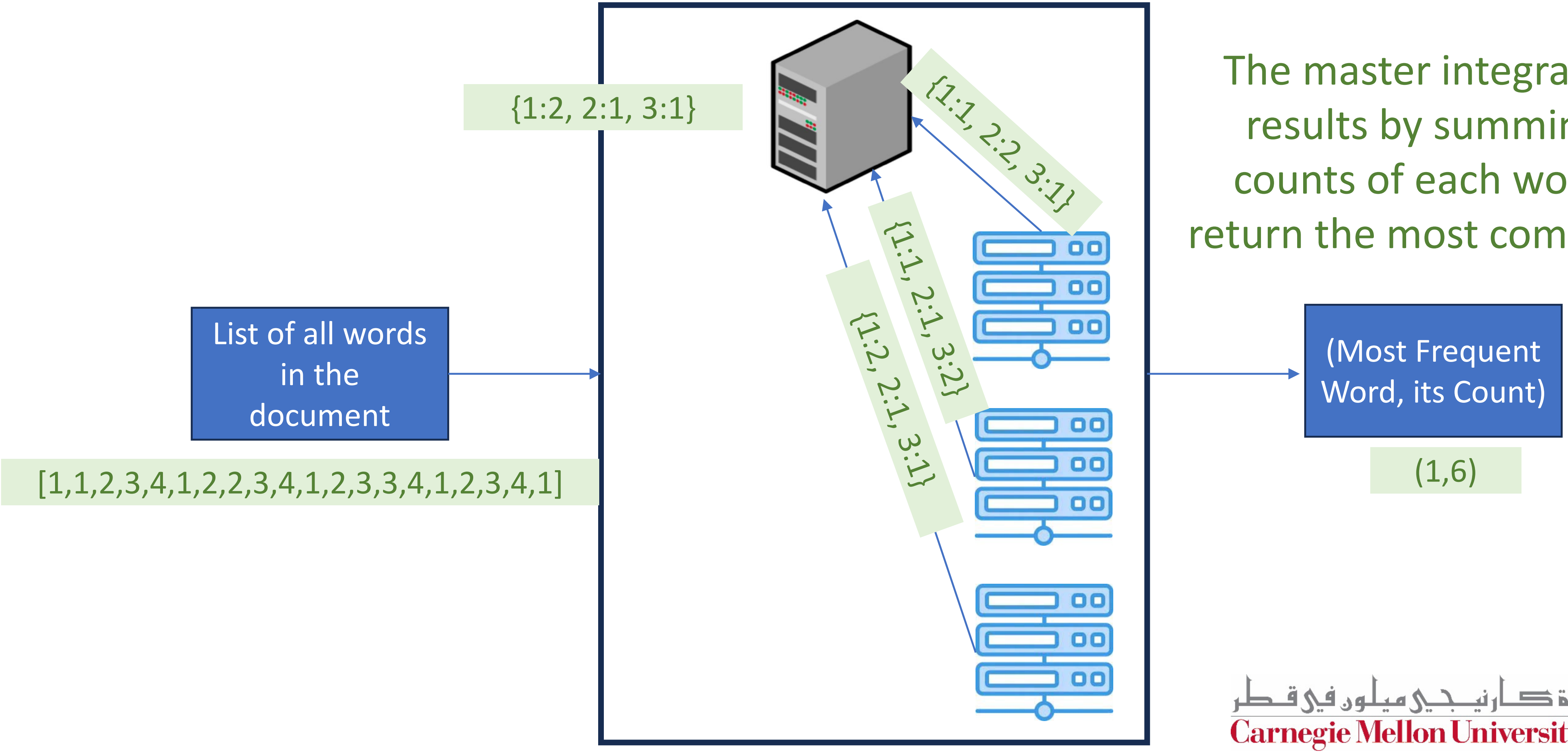
Application – Most Frequent Word (Another Distributed Round)



Each node again receives the sublist of words and a list of the most common words [1,2,3]

Each Node Counts the number of occurrences of each of the common words

Application – Most Frequent Word (Aggregating Results)



The master integrates the results by summing the counts of each word and return the most common one

Most Frequent Word Solution – Using Ray Tasks (Python Functions)

```python
import time
import ray

ray.init()

#l is a list of dicts
def finalMerge(l):
        finalD={}
        maxW=''
        for d in l:
                for w in d:
                        finalD[w]= finalD.get(w, 0)+d[w]
                        if maxW=='' or finalD[w] > finalD[maxW]:
                                maxW= w

        return maxW, finalD[maxW]

@ray.remote
def mostFrequentWordv1(wordList):
    maxword = None
    maxcnt = 0
    for word in wordList:
        cnt = wordList.count(word)
        if cnt > maxcnt:
            maxcnt = cnt
            maxword =word
    return (maxword, maxcnt)

@ray.remote
def freqWordsCount(mostFreqWords, wordList):
        d={}
        for w in mostFreqWords:
                d[w]= wordList.count(w)
        return d

def loadBook(filename):
    with open(filename,"r", encoding='utf-8') as f:
        theText = f.read()
    toRemove = [",",";",".","\"","\' ","\n\'"," \'","!","(",")","?","--","'",'"',"]","[",":",'*']

    theText = theText.lower()

    for badChar in toRemove:
        theText = theText.replace(badChar," ")
    return theText.split()

allWordsList = loadBook("alice.txt")
#allWordsList= [1,1,2,3,4,1,2,2,3,4,1,2,3,3,4,1,2,3,4,1]
print(f"Loaded text with {len(allWordsList)} words")

start = time.time()
l=  len(allWordsList)
sliceL= l//4

futures1 = [mostFrequentWordv1.remote(allWordsList[i:i+sliceL])  for i in range(0, l, sliceL)]
initialResults= ray.get(futures1) # list of tuples .. each node returns a tuple of the most common word it has in its assigned sublist and the word count (mostCommonWord, wordCount)
mostCommonInAllNodes= set([t[0] for t in initialResults]) #this creates a set of all words that were most common in the sublists

# we now each counts the occurances of each common word in its sublist... and returns a dictionary where each entry is commonWord:commonWordCount
futures2 = [freqWordsCount.remote(mostCommonInAllNodes, allWordsList[i:i+sliceL])  for i in range(0, l, sliceL)]
finalResults= ray.get(futures2)

ans= finalMerge(finalResults) # applies a final merge for the returned dictionary
end = time.time()
elapsed1 = end - start
print("List-based\nAnswer {} in {:0.4f} seconds".format(ans, elapsed1))
```

Most Frequent Word Solution – Using Ray Actors (Python Classes)

```python
import time
import ray

ray.init()

@ray.remote
class Node(object):

    def __init__(self, sublist):
        self.wordList= sublist

    def mostFrequentWordv1(self):
        maxword = None
        maxcnt = 0
        for word in self.wordList:
            cnt = self.wordList.count(word)
            if cnt > maxcnt:
                maxcnt = cnt
                maxword =word
        return (maxword, maxcnt)

    def freqWordsCount(self, mostFreqWords):
            d={}
            for w in mostFreqWords:
                    d[w]= self.wordList.count(w)
            return d

#l is a list of dicts
def finalMerge(l):
        finalD={}
        maxW=''
        for d in l:
                for w in d:
                        finalD[w]= finalD.get(w, 0)+d[w]
                        if maxW=='' or finalD[w] > finalD[maxW]:
                                maxW= w

        return maxW, finalD[maxW]

def loadBook(filename):
    with open(filename,"r", encoding='utf-8') as f:
        theText = f.read()
    toRemove = [",",";",".","\"","\' ","\n\'"," \'","!","(",")","?","--","'",'"',"]","[",":",'*']

    theText = theText.lower()

    for badChar in toRemove:
        theText = theText.replace(badChar," ")
    return theText.split()

allWordsList = loadBook("alice.txt")
#allWordsList= [1,1,2,3,4,1,2,2,3,4,1,2,3,3,4,1,2,3,4,1]
print(f"Loaded text with {len(allWordsList)} words")

start = time.time()
l=  len(allWordsList)

nodes= []
sliceL= l//4

for i in range(0, l, sliceL):
    nodes.append(Node.remote(allWordsList[i:i+sliceL]))

futures1 = [nodes[i].mostFrequentWordv1.remote()  for i in range(4)]
initialResults= ray.get(futures1) # list of tuples .. each node returns a tuple of the most common word it has in its assigned sublist and the word count (mostCommonWord, wordCount)
mostCommonInAllNodes= set([t[0] for t in initialResults]) #this creates a set of all words that were most common in the sublists

# we now each counts the occurances of each common word in its sublist... and returns a dictionary where each entry is commonWord:commonWordCount
futures2 = [nodes[i].freqWordsCount.remote(mostCommonInAllNodes)  for i in range(4)]
finalResults= ray.get(futures2)

ans= finalMerge(finalResults) # applies a final merge for the returned dictionary
end = time.time()
elapsed1 = end - start
print("List-based\nAnswer {} in {:0.4f} seconds".format(ans, elapsed1))
```