# Understanding Computer Storage & Big Data

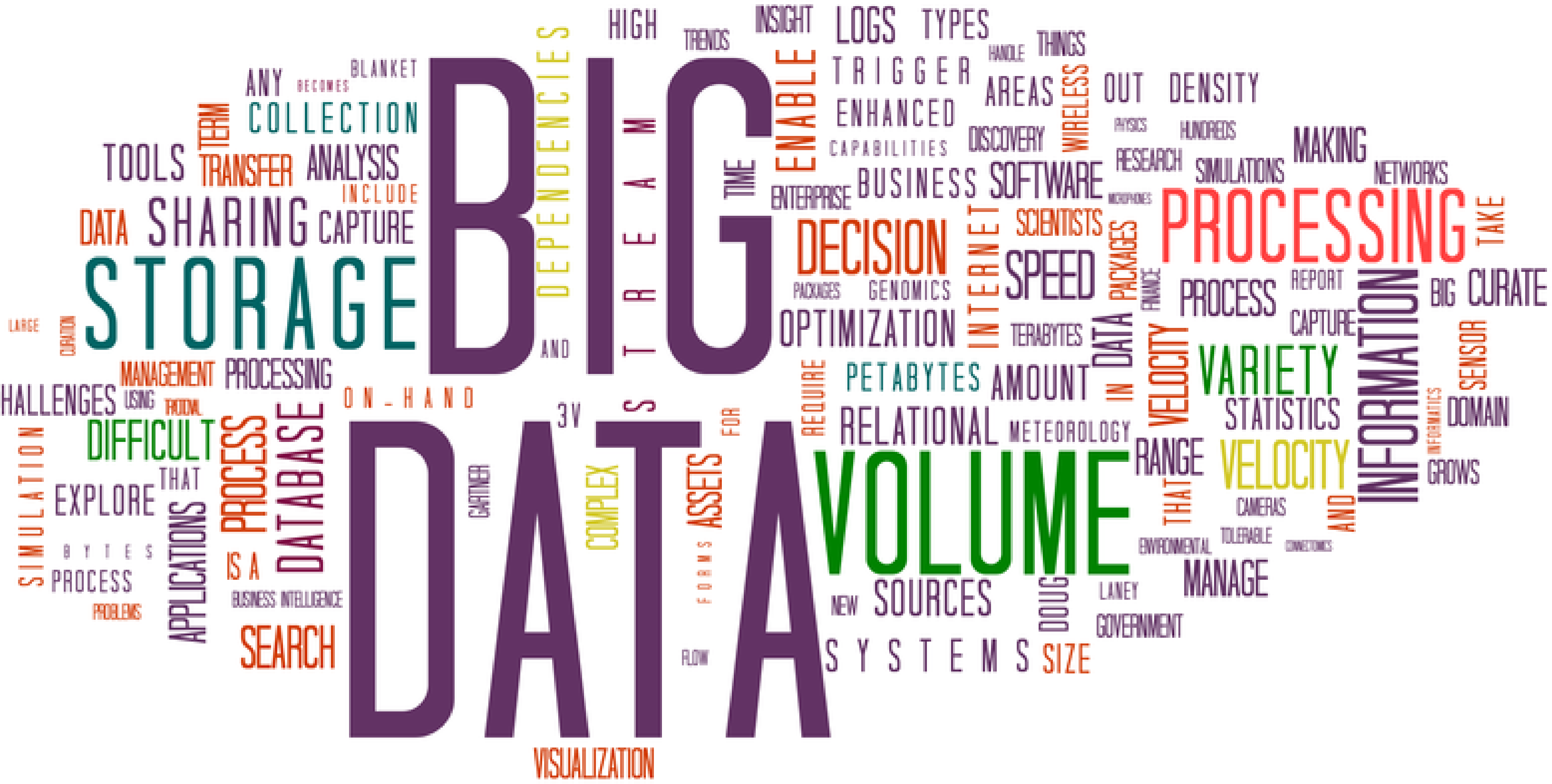
PARALLEL PROGRAMMING WITH DASK IN PYTHON

**Dhavide Aruliah**

Director of Training, Anaconda

datacamp

# What is "big data"?



"Data > one machine"

| | | |
|---|---|---|
| watt | W | |
| Kilowatt | KW | $10^3$ W |
| Megawatt | MW | $10^6$ W |
| Gigawatt | GW | $10^9$ W |
| Terawatt | TW | $10^{12}$ W |

| | | |
|---|---|---|
| Byte | B | $2^3$ bits |
| Kilobyte | KB | $2^{10}$ Bytes |
| Megabyte | MB | $2^{20}$ Bytes |
| Gigabyte | GB | $2^{30}$ Bytes |
| Terabyte | TB | $2^{40}$ Bytes |

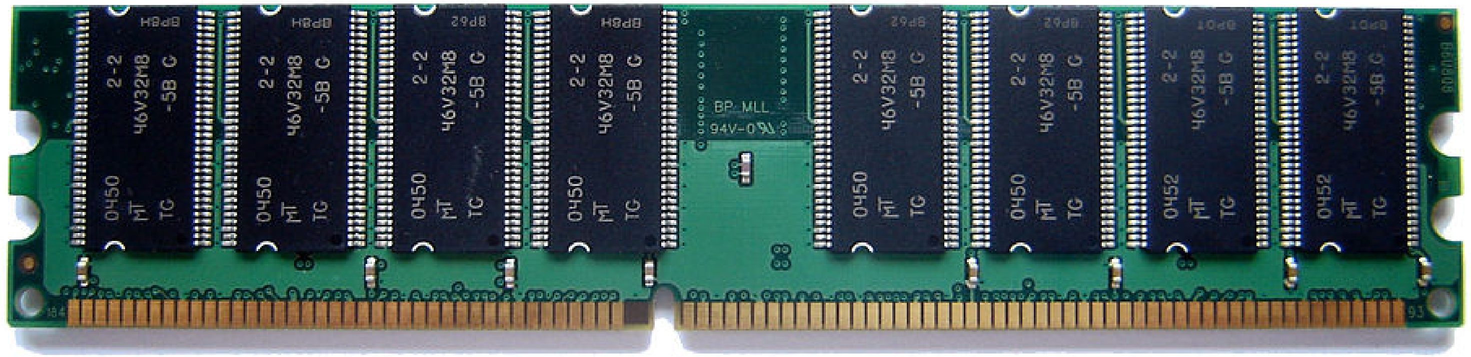
- Conventional units: factors of 1000
  - Kilo → Mega → Giga → Tera → $\cdots$

- Binary computers: base 2:
  - Binary digit (bit)
  - Byte: $2^3$ bits = $8$ bits
  - $10^3 = 1000 \mapsto 2^{10} = 1024$

# Hard disks

- Hard storage: hard disks (permanent, big, **slow**)

# Random Access Memory (RAM)



- Soft storage: RAM (temporary, small, **fast**)

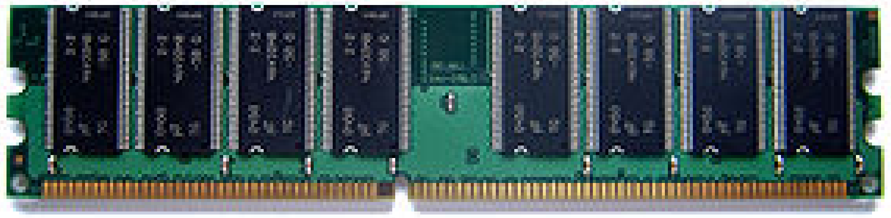
# Time scales of storage technologies

| Storage medium | Access time |
|---|---|
| RAM | 120 ns |
| Solid-state disk | 50-150 µs |
| Rotational disk | 1-10 ms |
| Internet (SF to NY) | 40 ms |

| Storage medium | Rescaled |
|---|---|
| RAM | 1 s |
| Solid-state disk | 7-21 min |
| Rotational disk | 2.5 hr - 1 day |
| Internet (SF to NY) | 3.9 days |

# Big data in practical terms

- RAM: **fast** (ns-µs)
- Hard disk: **slow** (µs-ms)
- I/O (input/output) is **punitive!**



# Querying Python interpreter's memory usage

```
import psutil, os
def memory_footprint():
...:    '''Returns memory (in MB) being used by Python process'''
...:    mem = psutil.Process(os.getpid()).memory_info().rss
...:    return (mem / 1024 ** 2)
```

# Allocating memory for an array

```
import numpy as np
before = memory_footprint()
N = (1024 ** 2) // 8 # Number of floats that fill 1 MB
x = np.random.randn(50*N) # Random array filling 50 MB
after = memory_footprint()
print('Memory before: {} MB'.format(before))
```

```
Memory before: 45.68359375 MB
```

```
print('Memory after: {} MB'.format(after))
```

```
Memory after: 95.765625 MB
```

# Allocating memory for a computation

```
before = memory_footprint()
x ** 2 # Computes, but doesn't bind result to a variable
```

```
array([ 0.16344891,  0.05993282,  0.53595334, ...,
0.50537523, 0.48967157,  0.06905984])
```

```
after = memory_footprint()
print('Extra memory obtained: {} MB'.format(after - before))
```

```
Extra memory obtained: 50.34375 MB
```

# Querying array memory Usage

```
x.nbytes  # Memory footprint in bytes (B)
```

```
52428800
```

```
x.nbytes // (1024**2) # Memory footprint in megabytes (MB)
```

```
50
```

# Querying DataFrame memory usage

```
df = pd.DataFrame(x)
```

```
df.memory_usage(index=False)
```

```
0    52428800
dtype: int64
```

```
df.memory_usage(index=False) // (1024**2)
```

```
0    50
dtype: int64
```

# Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON

# Thinking about Data in Chunks

PARALLEL PROGRAMMING WITH DASK IN PYTHON

**Dhavide Aruliah**

Director of Training, Anaconda

# Using pd.read_csv() with chunksize

```
filename = 'NYC_taxi_2013_01.csv'
for chunk in pd.read_csv(filename, chunksize=50000):
...:     print('type: %s shape %s' %
...:           (type(chunk), chunk.shape))
```

```
type: <class 'pandas.core.frame.DataFrame'> shape (50000, 14)
type: <class 'pandas.core.frame.DataFrame'> shape (50000, 14)
type: <class 'pandas.core.frame.DataFrame'> shape (50000, 14)
type: <class 'pandas.core.frame.DataFrame'> shape (49999, 14)
```

# Examining a chunk

```
chunk.shape
```

```
(49999, 14)
```

```
chunk.info()
```

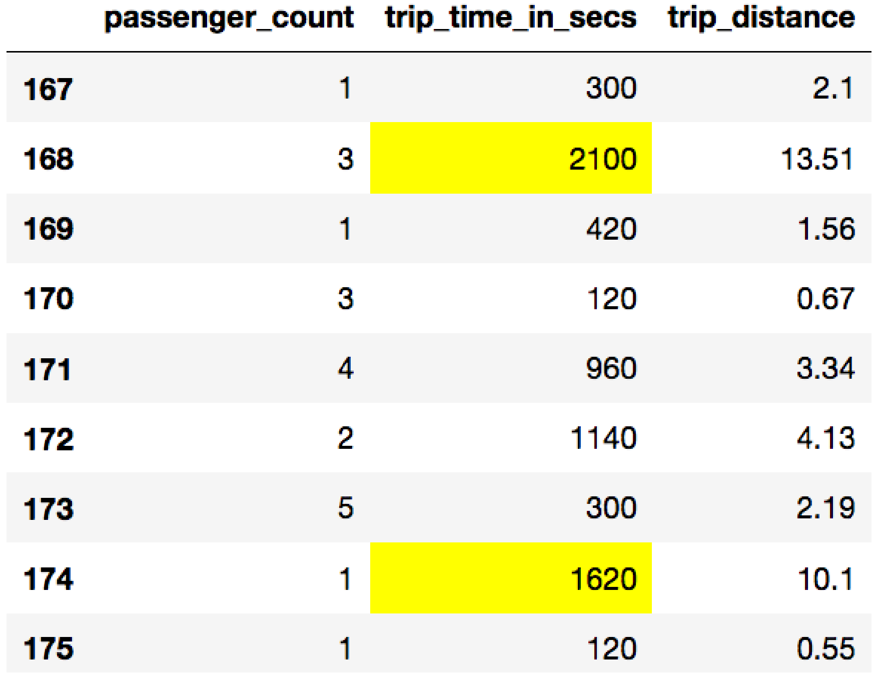
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 150000 to 199998
Data columns (total 14 columns):
medallion             49999 non-null object
...
dropoff_latitude      49999 non-null float64
dtypes: float64(5), int64(3), object(6)
memory usage: 5.3+ MB
```

# Filtering a chunk

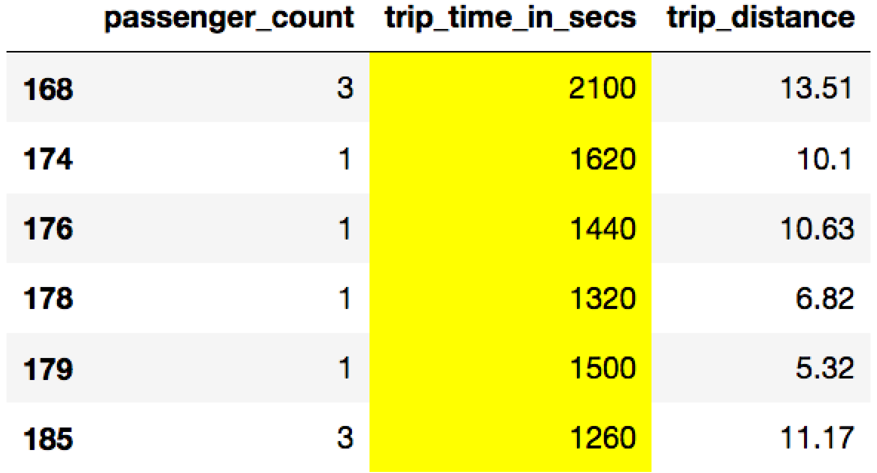
```
is_long_trip = (chunk.trip_time_in_secs > 1200)
```

```
chunk.loc[is_long_trip].shape
```

```
(5565, 14)
```



|  | passenger_count | trip_time_in_secs | trip_distance |
|---|---|---|---|
| **167** | 1 | 300 | 2.1 |
| **168** | 3 | 2100 | 13.51 |
| **169** | 1 | 420 | 1.56 |
| **170** | 3 | 120 | 0.67 |
| **171** | 4 | 960 | 3.34 |
| **172** | 2 | 1140 | 4.13 |
| **173** | 5 | 300 | 2.19 |
| **174** | 1 | 1620 | 10.1 |
| **175** | 1 | 120 | 0.55 |



|  | passenger_count | trip_time_in_secs | trip_distance |
|---|---|---|---|
| **168** | 3 | 2100 | 13.51 |
| **174** | 1 | 1620 | 10.1 |
| **176** | 1 | 1440 | 10.63 |
| **178** | 1 | 1320 | 6.82 |
| **179** | 1 | 1500 | 5.32 |
| **185** | 3 | 1260 | 11.17 |

# Chunking & filtering together

```
def filter_is_long_trip(data):
...:   "Returns DataFrame filtering trips longer than 20 minutes"
...:   is_long_trip = (data.trip_time_in_secs > 1200)
...:   return data.loc[is_long_trip]
```

```
chunks = []
for chunk in pd.read_csv(filename, chunksize=1000):
...:     chunks.append(filter_is_long_trip(chunk))
```

```
chunks = [filter_is_long_trip(chunk)
...:           for chunk in pd.read_csv(filename,
...:           chunksize=1000) ]
```

# Using pd.concat()

```
len(chunks)
```

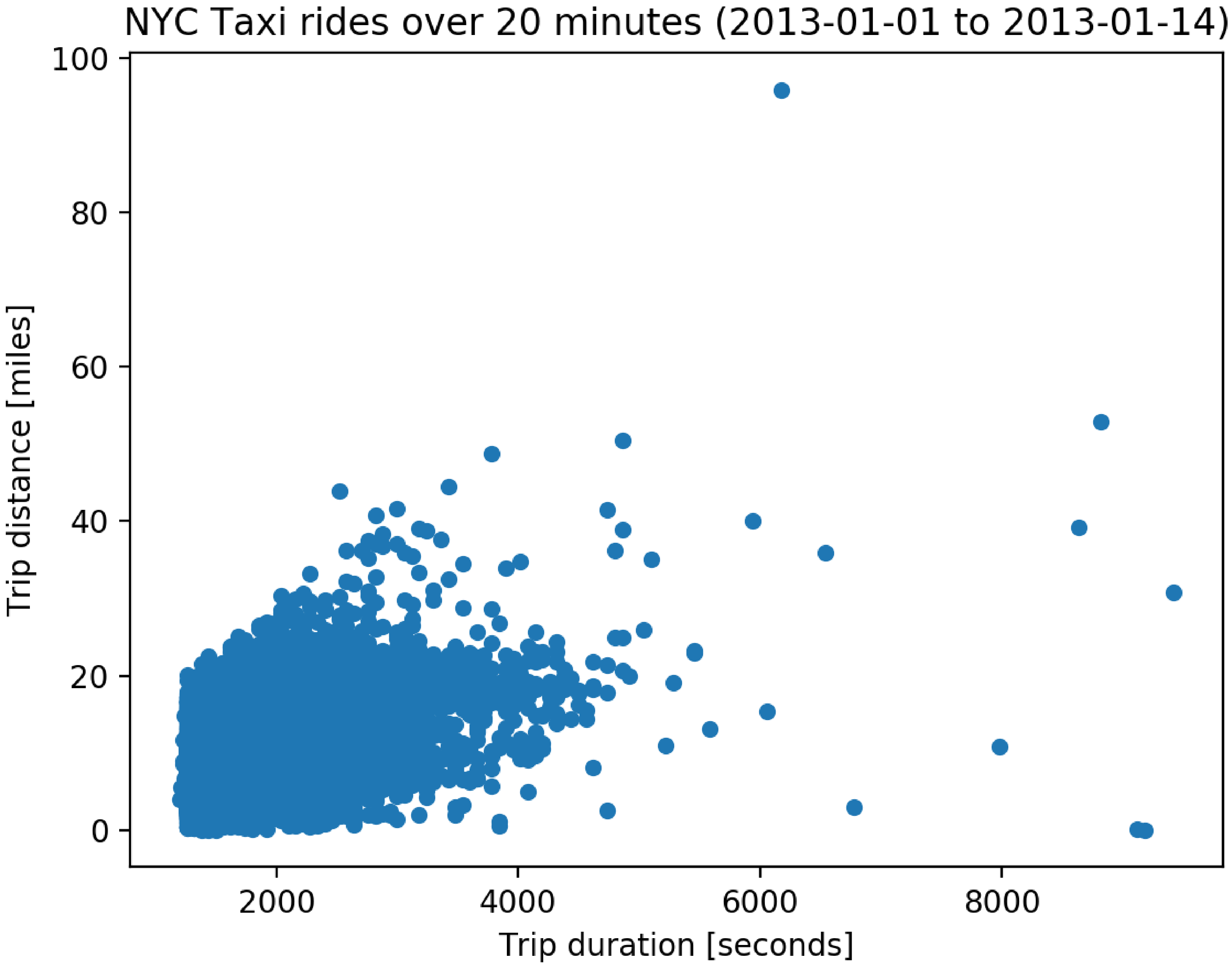
```
200
```

```
lengths = [len(chunk) for chunk in chunks]
lengths[-5:]   # Each has ~100 rows
```

```
[115, 147, 137, 109, 119]
```

```
long_trips_df = pd.concat(chunks)
long_trips_df.shape
```

```
(21661, 14)
```


NYC Taxi rides over 20 minutes (2013-01-01 to 2013-01-14)
Trip distance [miles]
0
20
40
60
80
100
2000
4000
6000
8000
Trip duration [seconds]

# Plotting the filtered results

```python
import matplotlib.pyplot as plt
long_trips_df.plot.scatter(x='trip_time_in_secs',
                           y='trip_distance');
plt.xlabel('Trip duration [seconds]');
plt.ylabel('Trip distance [miles]');
plt.title('NYC Taxi rides over 20 minutes (2013-01-01
to 2013-01-14)');
plt.show();
```

# Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON

# Managing Data with Generators

PARALLEL PROGRAMMING WITH DASK IN PYTHON

**Dhavide Aruliah**

Director of Training, Anaconda

# Filtering in a list comprehension

```
import pandas as pd
filename = 'NYC_taxi_2013_01.csv'
def filter_is_long_trip(data):
    "Returns DataFrame filtering trips longer than 20 mins"
    is_long_trip = (data.trip_time_in_secs > 1200)
    return data.loc[is_long_trip]
chunks = [filter_is_long_trip(chunk)
          for chunk in pd.read_csv(filename,
          chunksize=1000)]
```

# Filtering & summing with generators

```
chunks = (filter_is_long_trip(chunk)
             for chunk in pd.read_csv(filename,
             chunksize=1000))
distances = (chunk['trip_distance'].sum() for chunk in chunks)
sum(distances)
```

```
230909.56000000003
```

# Examining consumed generators

```
distances
```

```
<generator object <genexpr> at 0x10766f9e8>
```

```
next(distances)
```

```
StopIteration        Traceback (most recent call last)
<ipython-input-10-9995a5373b05> in <module>()
```

# Reading many files

```
template = 'yellow_tripdata_2015-{:02d}.csv'
filenames = (template.format(k) for k in range(1,13)) # Generator
for fname in filenames:
...:     print(fname) # Examine contents
```

```
yellow_tripdata_2015-01.csv
yellow_tripdata_2015-02.csv
yellow_tripdata_2015-03.csv
yellow_tripdata_2015-04.csv
...
yellow_tripdata_2015-09.csv
yellow_tripdata_2015-10.csv
yellow_tripdata_2015-11.csv
yellow_tripdata_2015-12.csv
```

# Examining a sample DataFrame

```python
df = pd.read_csv('yellow_tripdata_2015-12.csv', parse_dates=[1, 2])
df.info()  # Columns deleted from output
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71634 entries, 0 to 71633
Data columns (total 19 columns):
VendorID                 71634 non-null int64
tpep_pickup_datetime     71634 non-null datetime64[ns]
tpep_dropoff_datetime    71634 non-null datetime64[ns]
passenger_count          71634 non-null int64
...
...
dtypes: datetime64[ns](2), float64(12), int64(4), object(1)
memory usage: 10.4+ MB
```

# Examining a sample DataFrame

```
def count_long_trips(df):
...:     df['duration'] = (df.tpep_dropoff_datetime -
...:                       df.tpep_pickup_datetime).dt.seconds
...:     is_long_trip = df.duration > 1200
...:     result_dict = {'n_long':[sum(is_long_trip)],
...:                    'n_total':[len(df)]}
...:     return pd.DataFrame(result_dict)
```

# Aggregating with Generators

```
def count_long_trips(df):
...:     df['duration'] = (df.tpep_dropoff_datetime -
...:                         df.tpep_pickup_datetime).dt.seconds
...:     is_long_trip = df.duration > 1200
...:     result_dict = {'n_long':[sum(is_long_trip)],
...:                    'n_total':[len(df)]}
...:     return pd.DataFrame(result_dict)
filenames = [template.format(k) for k in range(1,13)] # Listcomp
dataframes = (pd.read_csv(fname, parse_dates=[1,2])
...:                   for fname in filenames)  # Generator
totals = (count_long_trips(df) for df in dataframes) # Generator
annual_totals = sum(totals) # Consumes generators
```

# Computing the fraction of long trips

```
print(annual_totals)
```

```
   n_long  n_total
0  172617   851390
```

```
fraction = annual_totals['n_long'] / annual_totals['n_total']
print(fraction)
```

```
0    0.202747
dtype: float64
```

# Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON

# Delaying Computation with Dask

PARALLEL PROGRAMMING WITH DASK IN PYTHON

**Dhavide Aruliah**
Director of Training, Anaconda

datacamp

# Composing functions

```
from math import sqrt
def f(z):
...:    return sqrt(z + 4)

def g(y):
...:    return y - 3

def h(x):
...:    return x ** 2
```

```
print(f(g(h(x)))) # Equal
```

```
x = 4
y = h(x)
z = g(y)
w = f(z)

print(w) # Final result
```

```
4.123105625617661
```

```
4.123105625617661
```

# Deferring computation with `delayed`

```python
from dask import delayed
y = delayed(h)(x)
z = delayed(g)(y)
w = delayed(f)(z)
print(w)
```

```
Delayed('f-5f9307e5-eb43-4304-877f-1df5c583c11c')
```

```python
type(w) # a dask Delayed object
```

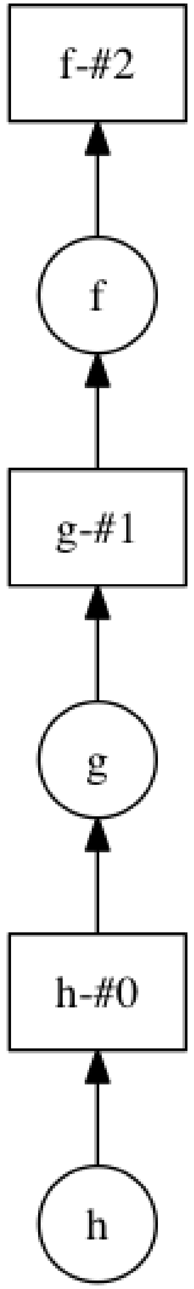
```
dask.delayed.Delayed
```

```python
w.compute() # Computation occurs now
```

```
4.123105625617661
```

# Visualizing a task graph

```
w.visualize()
```



# Renaming decorated functions

```
f = delayed(f)
g = delayed(g)
h = delayed(h)
w = f(g(h(4))
```

```
type(w) # a dask Delayed object
```

```
dask.delayed.Delayed
```

```
w.compute() # Computation occurs now
```

```
4.123105625617661
```

# Using decorator @-notation

```
def f(x):
...:     return sqrt(x + 4)
f = delayed(f)
@delayed # Equivalent to definition in above 2 cells
...: def f(x):
...:     return sqrt(x + 4)
```

# Deferring Computation with Loops

```
@delayed
...: def increment(x):
...:    return x + 1

@delayed
...: def double(x):
...:    return 2 * x

@delayed
...: def add(x, y):
...:    return x + y
```

```
data = [1, 2, 3, 4, 5]
output = []
for x in data:
...:     a = increment(x)
...:     b = double(x)
...:     c = add(a, b)
...:     output.append(c)

total = sum(output)
```

# Deferring computation with loops 2

```
total
```

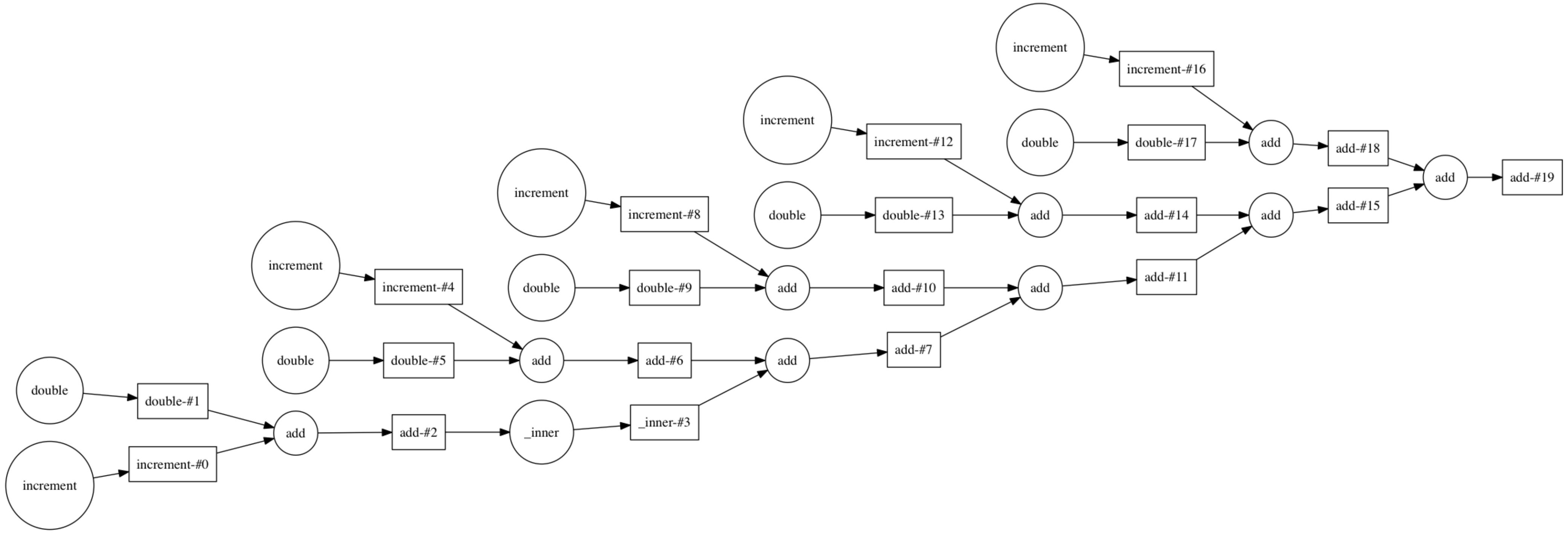
```
Delayed('add-c6803f9e890c95cec8e2e3dd3c62b384')
```

```
output
```

```
[Delayed('add-6a624d8b-8ddb-44fc-b0f0-0957064f54b7'),
 Delayed('add-9e779958-f3a0-48c7-a558-ce47fc9899f6'),
 Delayed('add-f3552c6f-b09d-4679-a770-a7372e2c278b'),
 Delayed('add-ce05d7e9-42ec-4249-9fd3-61989d9a9f7d'),
 Delayed('add-dd950ec2-c17d-4e62-a267-1dabe2101bc4')]
```

```
total.visualize()
```

# Visualizing the task graph



# Aggregating with delayed Functions

```
template = 'yellow_tripdata_2015-{:02d}.csv'
filenames = [template.format(k) for k in range(1,13)]
@delayed
...: def count_long_trips(df):
...:     df['duration'] = (df.tpep_dropoff_datetime -
...:                       df.tpep_pickup_datetime).dt.seconds
...:     is_long_trip = df.duration > 1200
...:     result_dict = {'n_long':[sum(is_long_trip)],
...:                    'n_total':[len(df)]}
...:     return pd.DataFrame(result_dict)
@delayed
...: def read_file(fname):
...:     return pd.read_csv(fname, parse_dates=[1,2])
```

# Computing fraction of long trips with `delayed` functions

```
totals = [count_long_trips(read_file(fname)) for fname in filenames]
annual_totals = sum(totals)
annual_totals = annual_totals.compute()
```

```
   n_long  n_total
0  172617   851390
```

```
fraction = annual_totals['n_long']/annual_totals['n_total']
print(fraction)
```

```
0    0.202747
dtype: float64
```

# Let's practice!

PARALLEL PROGRAMMING WITH DASK IN PYTHON