MATLAB EXPO 2016

Navigating Big Data with MATLAB

Isaac Noh

Application Engineer

How big is big?

What does “Big Data” even mean?

> *“Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate to deal with them.”*
>
> *Wikipedia*

So, what’s the (big) problem?

- Traditional tools and approaches won’t work
  - **Getting** the data is hard; **processing** it is even harder
  - Need to learn **new tools** and **new coding styles**
  - Have to rewrite algorithms, often at a lower level of abstraction

- Quality of your results can be impacted
  - e.g., by being forced to work on a subset of your data

Big Data workflow

ACCESS

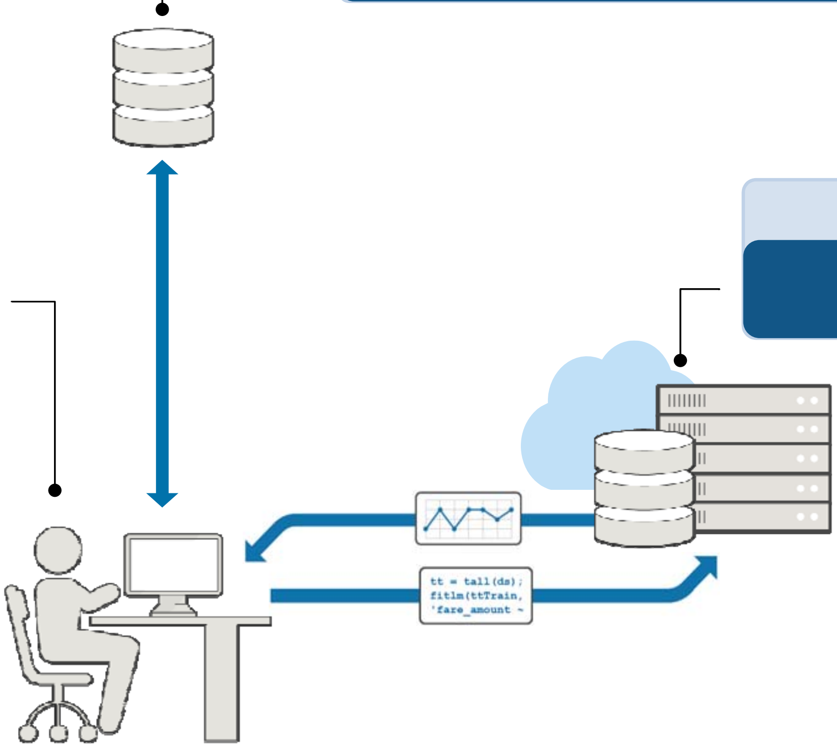
More data and collections of files than fit in memory

PROCESS AND ANALYZE

Adapt traditional processing tools or learn new tools to work with Big Data

SCALE

To Big Data systems like Hadoop



```
tt = tall(ds);
fitlm(ttTrain,
'fare_amount ~
```

Big solutions

Wouldn’t it be nice if you could:

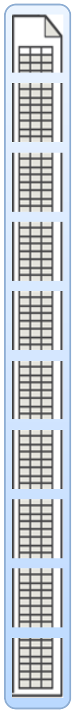
- Easily access data however it is stored
- Prototype algorithms quickly using a local workstation
- Scale up to big data sets running on large clusters
- **Using the same intuitive MATLAB syntax you are used to**

tall arrays R2016b

- For data that doesn’t fit into memory

- Lots of observations (hence “tall”)

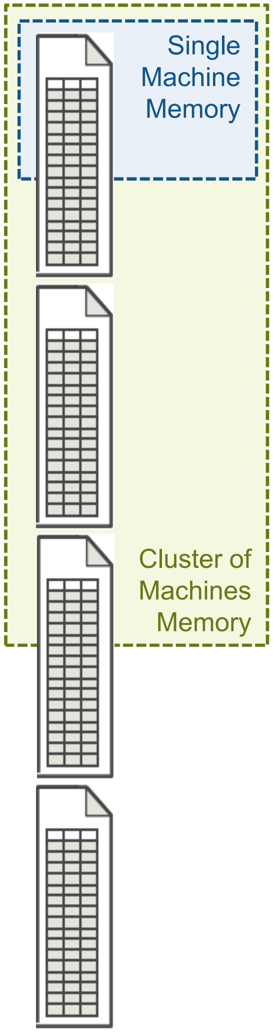
- Looks like a normal MATLAB array
  - Supports numeric types, tables, datetimes, strings, etc…
  - Supports basic math, stats, indexing, etc.
  - **Statistics and Machine Learning Toolbox** support (clustering, classification, etc.)



`tall` arrays R2016b

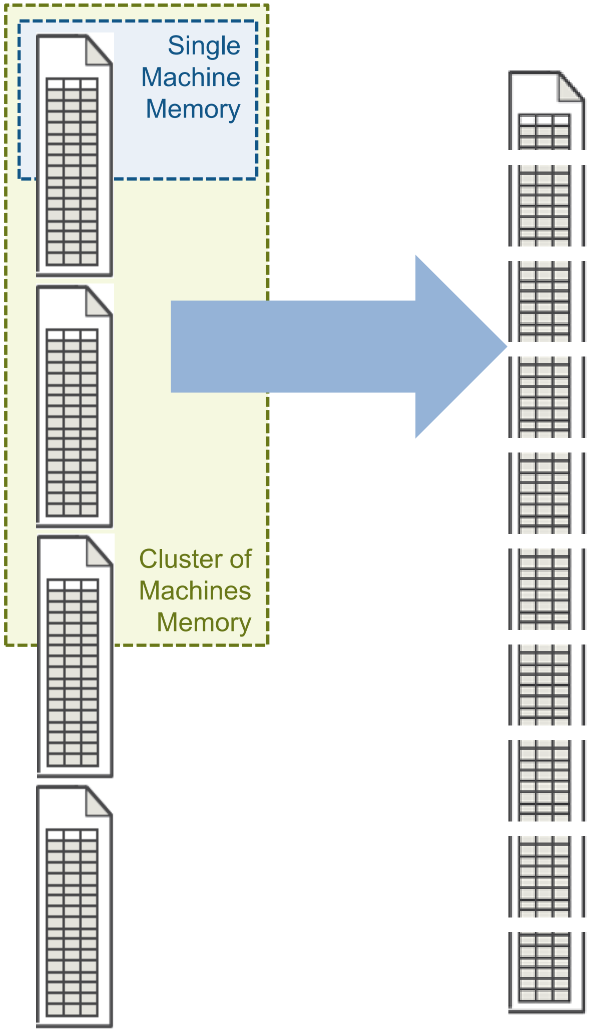
- Data is in one or more files
- Typically tabular data
- Files stacked vertically
- Data doesn't fit into memory

  (even cluster memory)



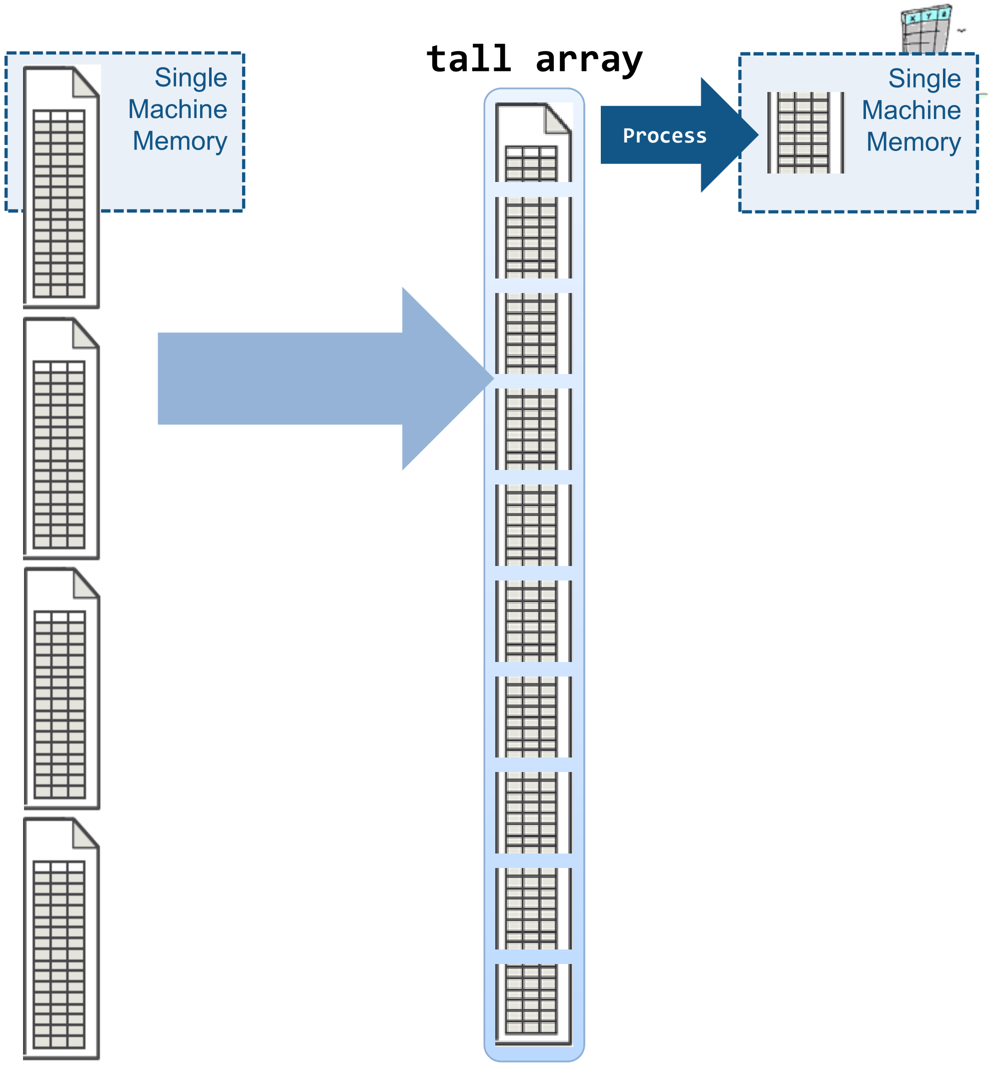
tall arrays R2016b

- Automatically breaks data up into small “chunks” that fit in memory



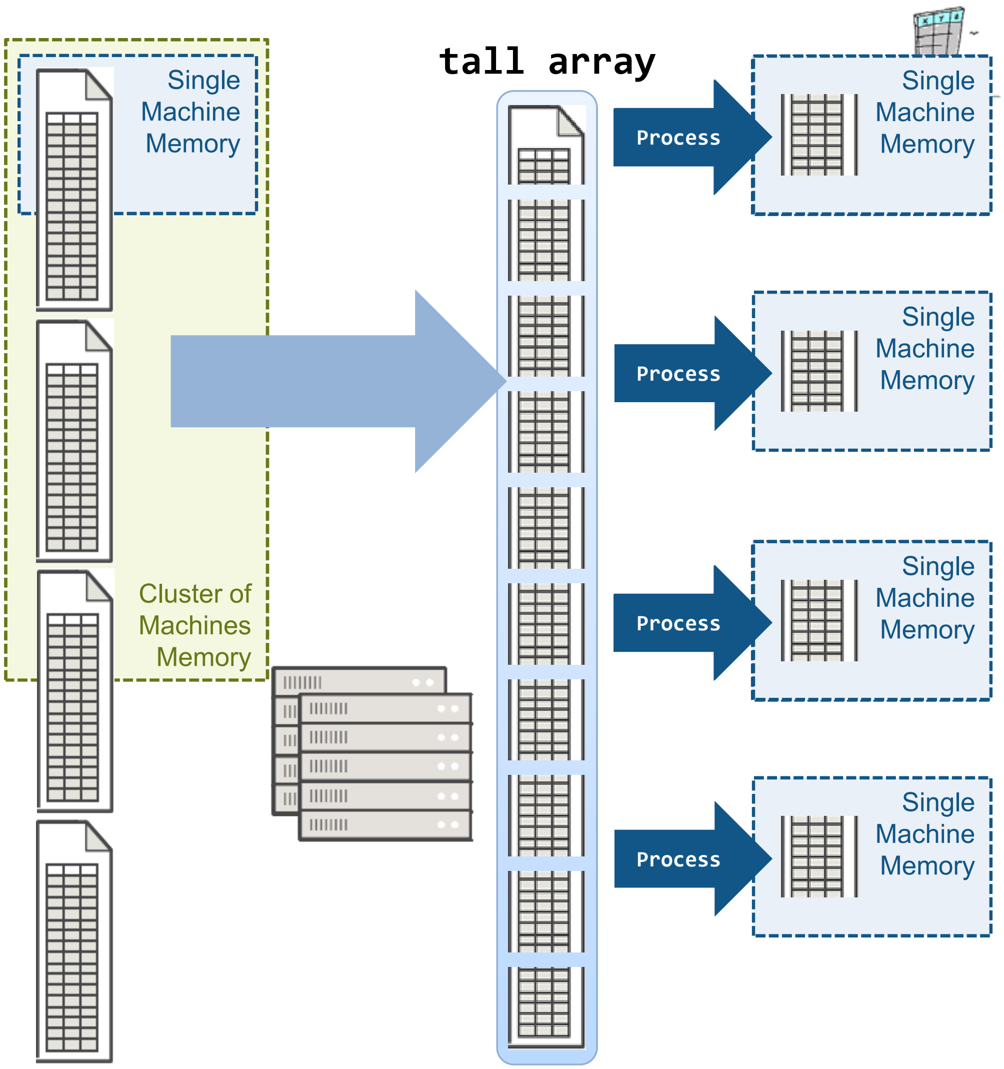
tall arrays R2016b

- “Chunk” processing is handled automatically
- Processing code for tall arrays is the same as ordinary arrays



tall arrays R2016b

- With Parallel Computing Toolbox, process several “chunks” at once
- Can scale up to clusters with MATLAB Distributed Computing Server



Example: Working with Big Data in MATLAB

- **Objective:** Create a model to predict the cost of a taxi ride in New York City

- **Inputs:**
  - Monthly taxi ride log files
  - The local data set is **small** (~2 MB)
  - The full data set is **big** (~25 GB)

- **Approach:**
  - Preprocess and explore data
  - Develop and validate predictive model (linear fit)
    - Work with subset of data for prototyping
    - Scale to full data set on HDFS

Example: Prototyping

Preview Data

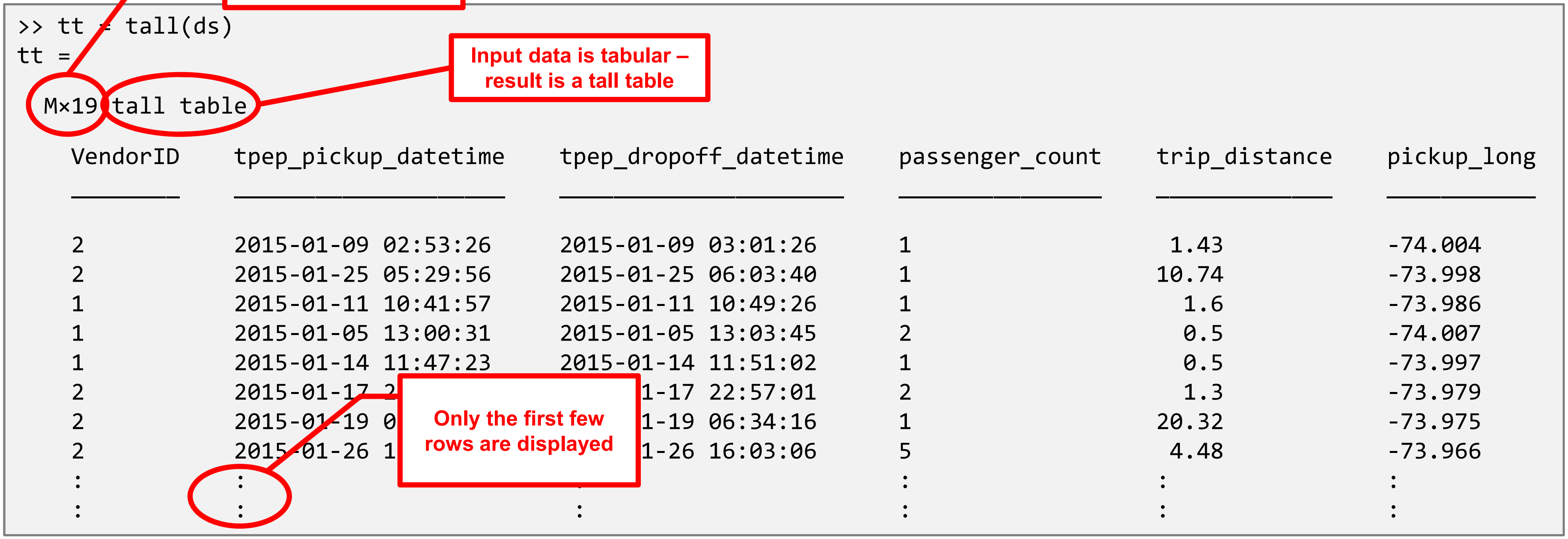
Description

- Location: New York City
- Date(s): (Partial) January 2015
- Data size: **“small data” 13,693 rows / ~2 MB**

```
>> ds = datastore('taxidataNYC_1_2015.csv');
>> preview(ds)

    VendorID    tpep_pickup_datetime    tpep_dropoff_datetime    passenger_count    trip_distance    pickup_long

    ________    ____________________    _____________________    _______________    _____________    ___________

    2           2015-01-09 02:53:26     2015-01-09 03:01:26      1                   1.43            -74.004
    2           2015-01-25 05:29:56     2015-01-25 06:03:40      1                  10.74            -73.998
    1           2015-01-11 10:41:57     2015-01-11 10:49:26      1                   1.6             -73.986
    1           2015-01-05 13:00:31     2015-01-05 13:03:45      2                   0.5             -74.007
    1           2015-01-14 11:47:23     2015-01-14 11:51:02      1                   0.5             -73.997
    2           2015-01-17 22:49:44     2015-01-17 22:57:01      2                   1.3             -73.979
    2           2015-01-19 06:01:36     2015-01-19 06:34:16      1                  20.32            -73.975
    2           2015-01-26 15:17:21     2015-01-26 16:03:06      5                   4.48            -73.966
    2           2015-01-25 04:19:55     2015-01-25 04:24:49      5                   1.28            -73.954
    2           2015-01-31 18:27:28     2015-01-31 18:31:43      5                   1.24            -73.969
```

Example: Prototyping

Create a Tall Array



```
>> tt = tall(ds)
tt =

  M×19 tall table

    VendorID    tpep_pickup_datetime    tpep_dropoff_datetime    passenger_count    trip_distance    pickup_long
    ________    ____________________    _____________________    _______________    _____________    ___________

    2           2015-01-09 02:53:26     2015-01-09 03:01:26      1                   1.43            -74.004
    2           2015-01-25 05:29:56     2015-01-25 06:03:40      1                  10.74            -73.998
    1           2015-01-11 10:41:57     2015-01-11 10:49:26      1                   1.6             -73.986
    1           2015-01-05 13:00:31     2015-01-05 13:03:45      2                   0.5             -74.007
    1           2015-01-14 11:47:23     2015-01-14 11:51:02      1                   0.5             -73.997
    2           2015-01-17 2                  1-17 22:57:01      2                   1.3             -73.979
    2           2015-01-19 0                  1-19 06:34:16      1                  20.32            -73.975
    2           2015-01-26 1                  1-26 16:03:06      5                   4.48            -73.966
    :           :                                                :                  :                :
    :           :                        :                       :                  :                :
```

Number of rows is unknown until all the data has been read

Input data is tabular – result is a tall table

Only the first few rows are displayed

Example: Prototyping

Calling Functions with a Tall Array

Once the tall table is created, can process much like an ordinary table

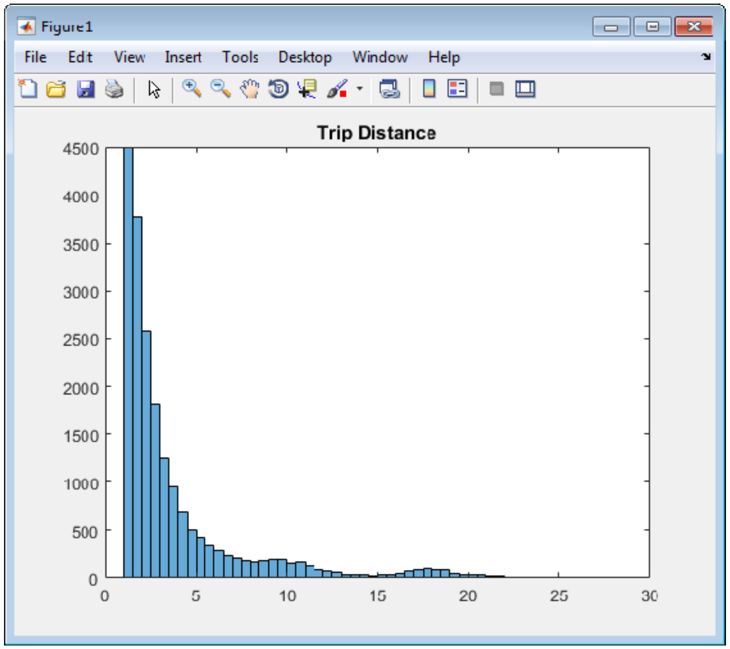
```
% Calculate average trip duration
mnTrip = mean(tt.trip_minutes,'omitnan')

mnTrip =

   tall double

    ?

Preview deferred. Learn more.

% Execute commands and gather results into workspace
mn = gather(mnTrip)

Evaluating tall expression using the Local MATLAB Session:
- Pass 1 of 1: Completed in 4 sec
Evaluation completed in 5 sec

mn =

   15.2648
```

- Most results are evaluated only when explicitly requested (e.g., `gather`)
- MATLAB automatically optimizes queued calculations to minimize the number of passes through the data

Example: Prototyping

Preprocess, clean, and explore data

```
% Remove some bad data
tt.speed_mph = tt.trip_distance ./ (tt.trip_minutes ./ 60);
ignore = tt.trip_minutes <= 1 | ...    % really short
    tt.trip_minutes >= 60 * 12 | ...   % unfeasibly long
    tt.trip_distance <= 1 | ...        % really short
    tt.trip_distance >= 12 * 55 | ...  % unfeasibly far
    tt.speed_mph > 55 | ...            % unfeasibly fast
    tt.total_amount < 0 | ...          % negative fares?!
    tt.total_amount > 10000;           % unfeasibly large fares
tt(ignore, :) = [];

% Explore data
figure
histogram(tt.trip_distance,'BinLimits',[0 30])
title('Trip Distance')

Evaluating tall expression using the Local MATLAB Session:
- Pass 1 of 2: Completed in 6 sec
- Pass 2 of 2: Completed in 6 sec
Evaluation completed in 12 sec
```



Example: Prototyping

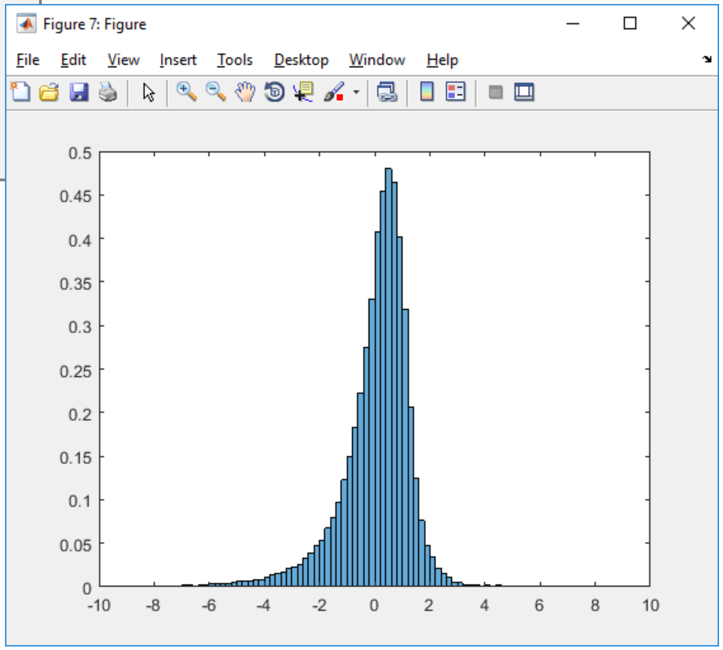
Fit predictive model

```
% Fit predictive model
model = fitlm(ttTrain,'fare_amount ~ 1 + hr_of_day + trip_distance*trip_minutes')

Evaluating tall expression using the Local MATLAB Session:
- Pass 1 of 1: Completed in 7 sec
Evaluation completed in 8 sec

model =

Compact linear regression model:
    fare_amount ~ 1 + hr_of_day + trip_distance*trip_minutes

Estimated Coefficients:
                                   Estimate          SE         tStat       pValue  
                                 ___________    __________    _______    __________
    (Intercept)                       2.8167      0.038002      74.12             0
    trip_distance                     2.2207      0.006166     360.16             0
    hr_of_day                       0.001222     0.0019124    0.63901       0.52282
    trip_minutes                     0.24528      0.001793     136.79             0
    trip_distance:trip_minutes   -0.00053185    0.00012339    -4.3102    1.6336e-05

Number of observations: 58793, Error degrees of freedom: 58788
Root Mean Squared Error: 3.06
R-squared: 0.927,  Adjusted R-Squared 0.927
F-statistic vs. constant model: 1.86e+05, p-value = 0
```

Example: Prototyping

Predict and validate model

```
% Predict and validate
yPred = predict(model,ttValidation);
residuals = yPred - ttValidation.fare_amount;
figure
histogram(residuals,'Normalization','pdf','BinLimits',[-10 10])

Evaluating tall expression using the Local MATLAB Session:
- Pass 1 of 2: Completed in 8 sec
- Pass 2 of 2: Completed in 5 sec
Evaluation completed in 15 sec
```

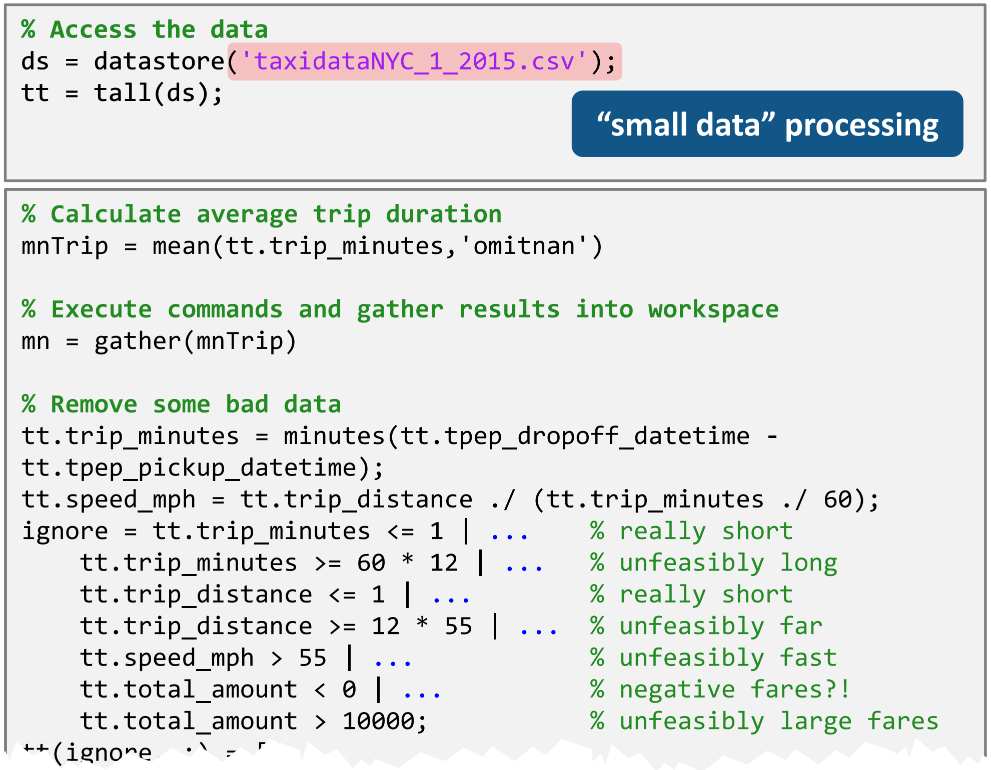


Scale to the Entire Data Set

Description

Location:	New York City	
Date(s):	All of 2015	
Data size:	**“Big Data”**	**150,000,000 rows / ~25 GB**

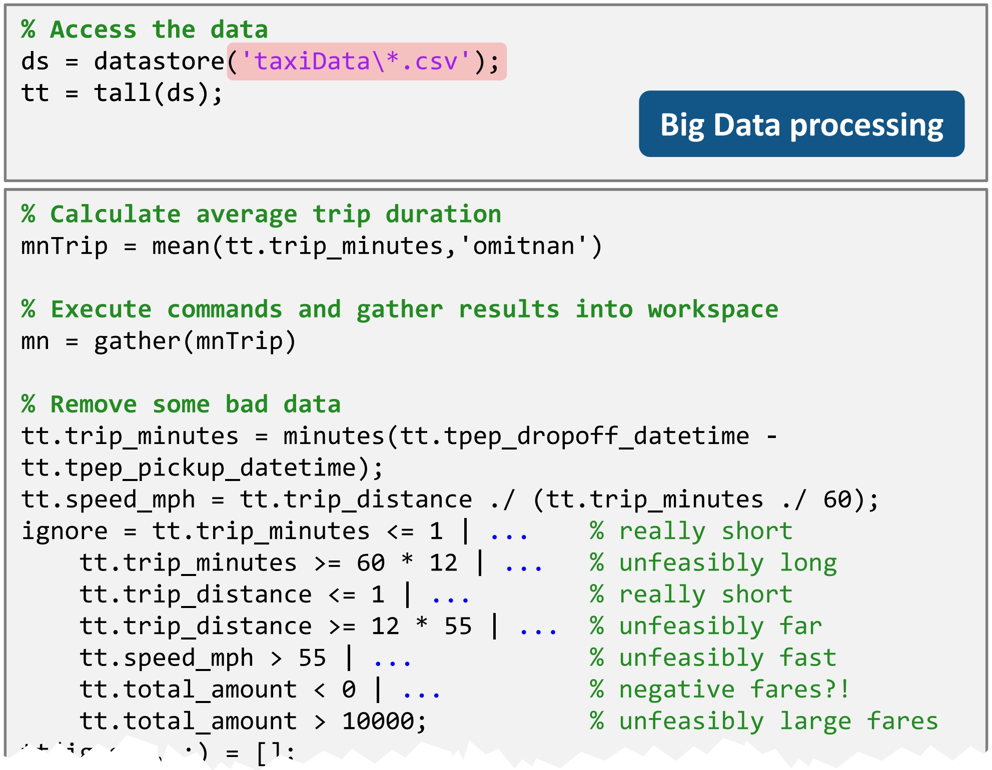
Example: “small data” processing vs. Big Data processing



```
% Access the data
ds = datastore('taxidataNYC_1_2015.csv');
tt = tall(ds);
```

“small data” processing

```
% Calculate average trip duration
mnTrip = mean(tt.trip_minutes,'omitnan')

% Execute commands and gather results into workspace
mn = gather(mnTrip)

% Remove some bad data
tt.trip_minutes = minutes(tt.tpep_dropoff_datetime -
tt.tpep_pickup_datetime);
tt.speed_mph = tt.trip_distance ./ (tt.trip_minutes ./ 60);
ignore = tt.trip_minutes <= 1 | ...    % really short
    tt.trip_minutes >= 60 * 12 | ...   % unfeasibly long
    tt.trip_distance <= 1 | ...        % really short
    tt.trip_distance >= 12 * 55 | ...  % unfeasibly far
    tt.speed_mph > 55 | ...            % unfeasibly fast
    tt.total_amount < 0 | ...          % negative fares?!
    tt.total_amount > 10000;           % unfeasibly large fares
tt(ignore, :) = [
```



```
% Access the data
ds = datastore('taxiData\*.csv');
tt = tall(ds);
```

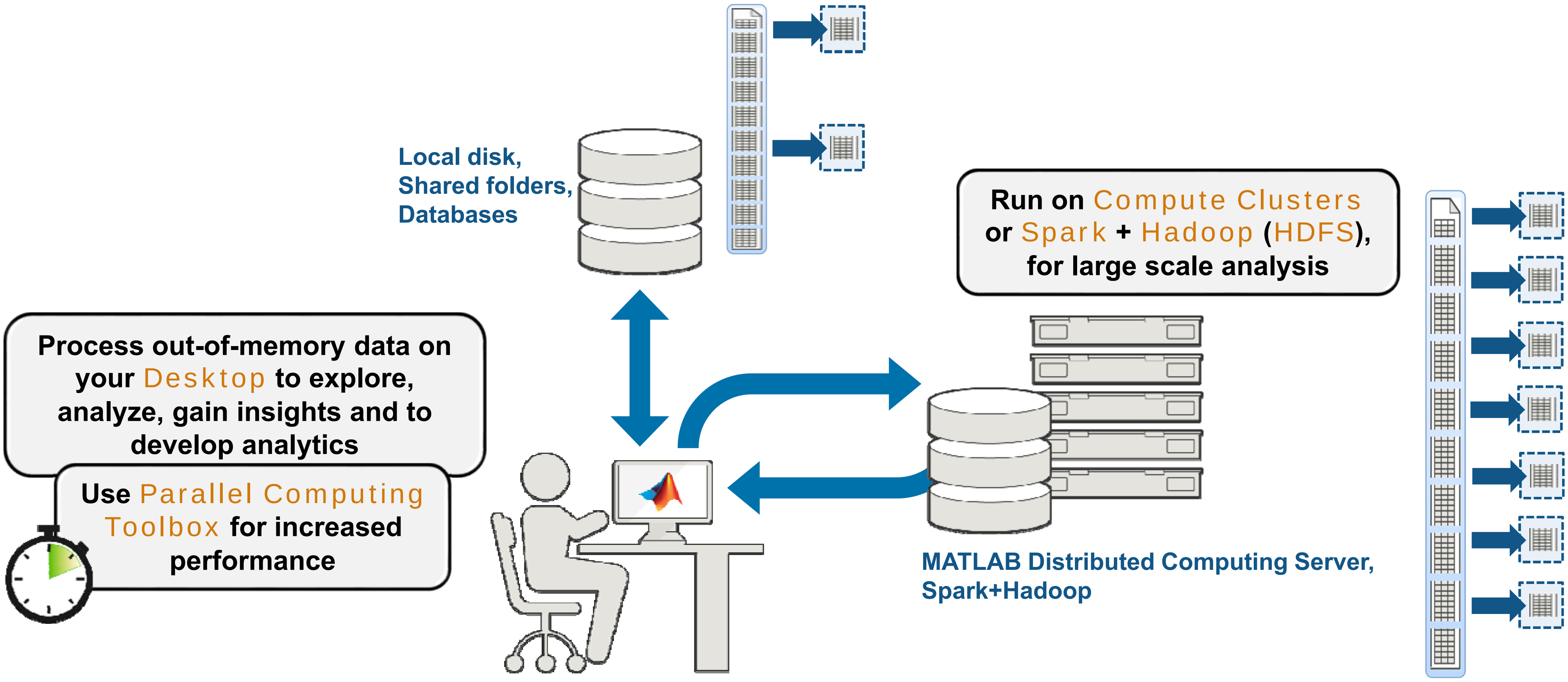
Big Data processing

```
% Calculate average trip duration
mnTrip = mean(tt.trip_minutes,'omitnan')

% Execute commands and gather results into workspace
mn = gather(mnTrip)

% Remove some bad data
tt.trip_minutes = minutes(tt.tpep_dropoff_datetime -
tt.tpep_pickup_datetime);
tt.speed_mph = tt.trip_distance ./ (tt.trip_minutes ./ 60);
ignore = tt.trip_minutes <= 1 | ...    % really short
    tt.trip_minutes >= 60 * 12 | ...   % unfeasibly long
    tt.trip_distance <= 1 | ...        % really short
    tt.trip_distance >= 12 * 55 | ...  % unfeasibly far
    tt.speed_mph > 55 | ...            % unfeasibly fast
    tt.total_amount < 0 | ...          % negative fares?!
    tt.total_amount > 10000;           % unfeasibly large fares
tt(ignore, :) = [];
```

Example: Running on Spark + Hadoop

```
% Hadoop/Spark Cluster
numWorkers = 16;

setenv('HADOOP_HOME', '/dev_env/cluster/hadoop');
setenv('SPARK_HOME', '/dev_env/cluster/spark');

cluster = parallel.cluster.Hadoop;
cluster.SparkProperties('spark.executor.instances') = num2str(numWorkers);
mr = mapreducer(cluster);


% Access the data
ds = datastore('hdfs://hadoop01:54310/datasets/taxiData/*.csv');
tt = tall(ds);
```

Summary for `tall` arrays



Local disk,
Shared folders,
Databases

Process out-of-memory data on your Desktop to explore, analyze, gain insights and to develop analytics

Use Parallel Computing Toolbox for increased performance

Run on Compute Clusters or Spark + Hadoop (HDFS), for large scale analysis

MATLAB Distributed Computing Server,
Spark+Hadoop

Big Data capabilities in MATLAB

ACCESS

Access data and collections of files that do not fit in memory

Datastores

- Images
- Spreadsheets
- Tabular Text
- Custom Files
- SQL
- Hadoop (HDFS)

PROCESS AND ANALYZE

Purpose-built capabilities for domain experts to work with big data locally

Tall Arrays

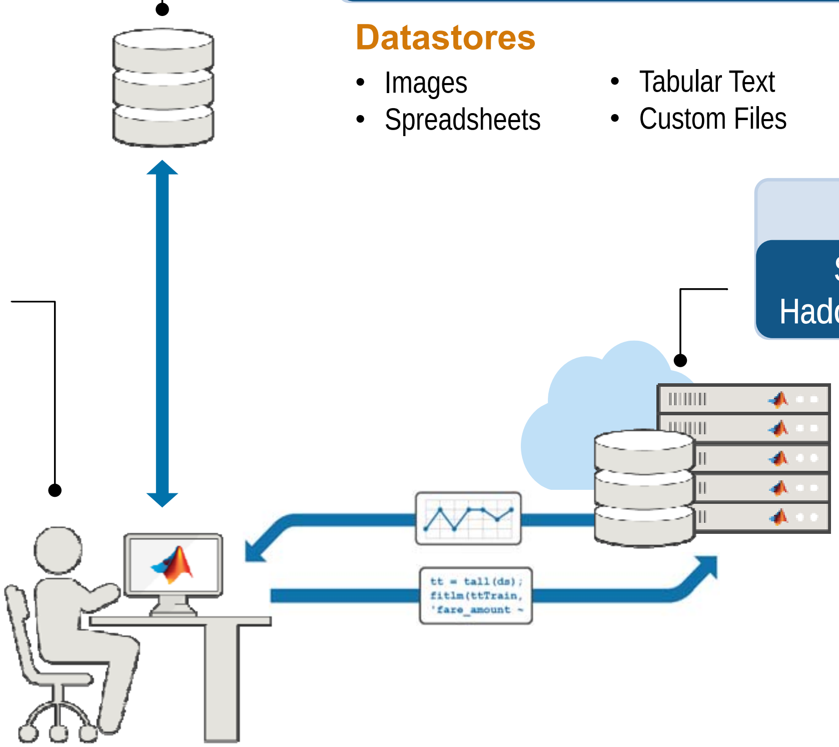
- Math
- Statistics
- Visualization
- Machine Learning

GPU Arrays

- Matrix Math
- Image Processing

Deep Learning

- Image Classification



```
tt = tall(ds);
fitlm(ttTrain,
'fare_amount ~
```

SCALE

Scale to compute clusters and Hadoop/Spark for data stored in HDFS

Tall Arrays

- Math, Stats, Machine Learning on Spark

Distributed Arrays

- Matrix Math on Compute Clusters

MDCS for EC2

- Cloud-based Compute Cluster

MapReduce

MATLAB API for Spark

Summary

- MATLAB makes it easy, convenient, and scalable to work with big data
  - **Access** any kind of big data from any file system
  - Use tall arrays to **process and analyze** that data on your desktop, clusters, or on Hadoop/Spark

There's no need to learn big data programming or out-of-memory techniques -- simply use the same code and syntax you're already used to.

For more information

- **Advanced Data Analytics with MATLAB** kiosk
- Website:
  https://www.mathworks.com/solutions/big-data-matlab
- Web search for:
  "Big Data MATLAB"