# Big Data on HPC

Introduction to Big Data with High-Performance Computing

Charles Peterson

# 💡 Overview

In this workshop, we will dive into utilizing Big Data techniques with High-Performance Computing Resources 🖥️💥

We will cover:

- 🔍 General concepts of Big Data
- 🧪 Simple examples on Hoffman2


💌 Suggestions and Feedback,
Email cpeterson@oarc.ucla.edu

# 📁 Files for this Presentation

This presentation and accompanying materials are available on our UCLA OARC's GitHub repository 📚

🔗 UCLA OARC GitHub Repository

View slides in:

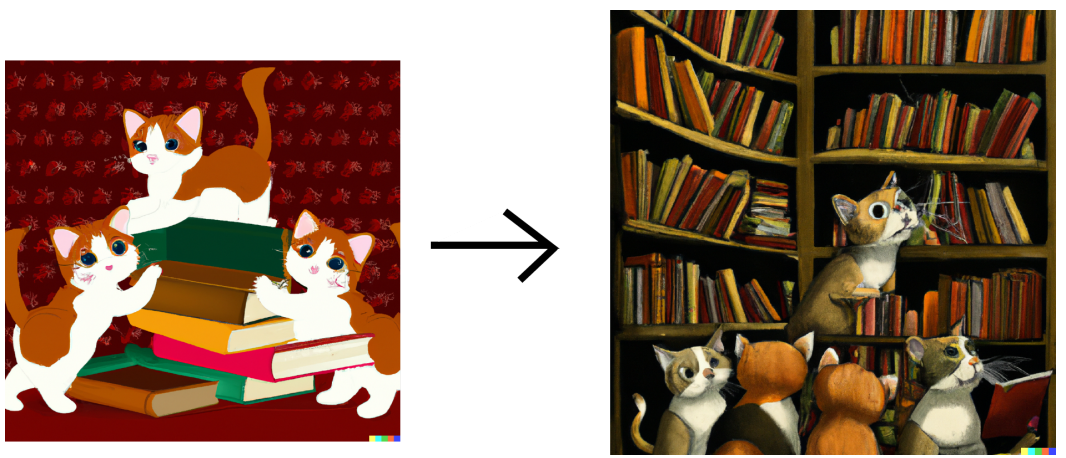
- 📄 PDF format - BigDataHPC.pdf
- 🌐 HTML format: Workshop Slides

You may want to install Spark/Dask now to follow along - `INTRO.md`

> **Note:** 🛠️ This presentation was built using Quarto and RStudio. Quarto file: `BigDataHPC.qmd`

# 🤔 What is Big Data?

The term **Big Data** refers to datasets, data processing, data modeling, and other data science tasks that become too large and complex for traditional techniques 💥



# ✨ Is Big Data for Me?

Are you working with data? Then, absolutely YES! 🎉

Big Data provides solutions for diverse research areas, scaling up research to new heights! 🚀

- Real-life Examples
  - Analyzing social media data for trends
  - Weather forecasting using data from sensors worldwide
  - Genomic data processing in biology

# 🚧 Challenges with LOTS of Data

Projects with lots of **DATA** come with their own set of challenges 😰:

- 🧠 Not enough RAM to accommodate large datasets.
- ⏳ Data processing is too time-consuming.
  - Traditional tasks struggle with large datasets.
  - They take forever to compute
- 🤖 Machine Learning models grow complex.
  - Training sophisticated models may require computational resources for better accuracy.

High-Performance Computing (HPC) resources can supercharge 🚀 the solving of Big Data challenges by providing much more computing power than typical workstations 💪.

# The 3 V's of Big Data 🌐

Big Data is often characterized by:

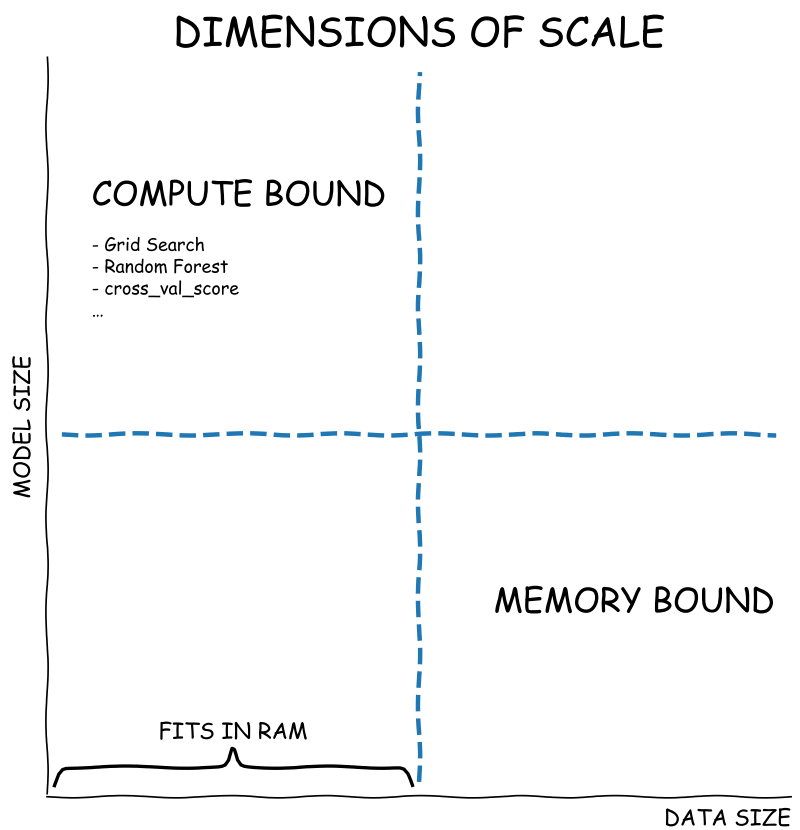
- **Volume**: The sheer size of data. 📚
- **Velocity**: The speed at which data is generated, processed, and transferred. 🌪️
  - Includes data transfer rates, in-memory processing vs. disk-based processing, etc.
- **Variety**: The diversity in types, sources, and nature of data. This includes structured vs unstructured data, and preprocessing or data cleanup requirements. 🎨

# The Other V's of Big Data 🧩

- **Value**: The potential insights and 'worth' that can be extracted from the data. 💡
- **Veracity**: The reliability, authenticity, and overall quality of the data. Includes handling missing values, data imputation, etc. 👍
- **Variability**: The adaptability of data in different formats, sources, and alignment with current data science methods. Raw and unstructured data can be tricky to manage. 🔃

💡 **Understanding your data can help you make informed decisions on which Big Data techniques to employ.**

# Big Data Challenges




- **Scaling Data Size** 📈
  - Datasets can become so large that they can't fit into RAM 😱
- **Scaling Model/Task Size** 🤖
  - Machine Learning or other tasks become so complex that a single CPU core is not adequate 🐌

# 📘 Big Data and HPC Concepts

- **HPC**
  - High-Performance Computing involves the use of powerful processors, networks, and parallel processing techniques
  - **Cluster**: A group of computers working together
  - **Node**: A single computer in a cluster
- **Parallel Computing** 🚦
  - Executing tasks simultaneously over multiple CPUs can make code run faster 🏎️
  - Typical Python/R tasks may run on a single CPU core.
  - Many packages/libraries are available for running tasks over multiple CPUs:
    - Python: Python Parallel Processing
    - R: R Parallel Computation

- **Cluster Computing** 🖥️🔗
  - Involves multiple computing resources working together.
  - HPC resources like Hoffman2 are excellent examples:
    - Multiple compute nodes with access to many cores.

# 🧐 More Concepts

- **Symmetric Multiprocessing (SMP) Programming** 🧠💼
  - SMP sets up parallel tasks over shared memory.
  - Typically, these tasks can ONLY run on a single compute node.
  - Uses a single process over multiple threads.
- **Cluster or Distributed Programming** 🌐💻
  - Distributes tasks over multiple processes.
  - Allows tasks to be distributed across multiple compute nodes.

# ⚙️ Even More Concepts

- **Lazy Evaluation ⏳**
  - Delays a task or expression until the resulting value is needed.
  - Functions and calculations can be set up so they can later run in parallel.

- **In-memory Computing 🚀**
  - Performs tasks on data in the RAM (memory) of the computer.
  - Data is computed in RAM instead of disk for faster processing.

# Big Data Tools

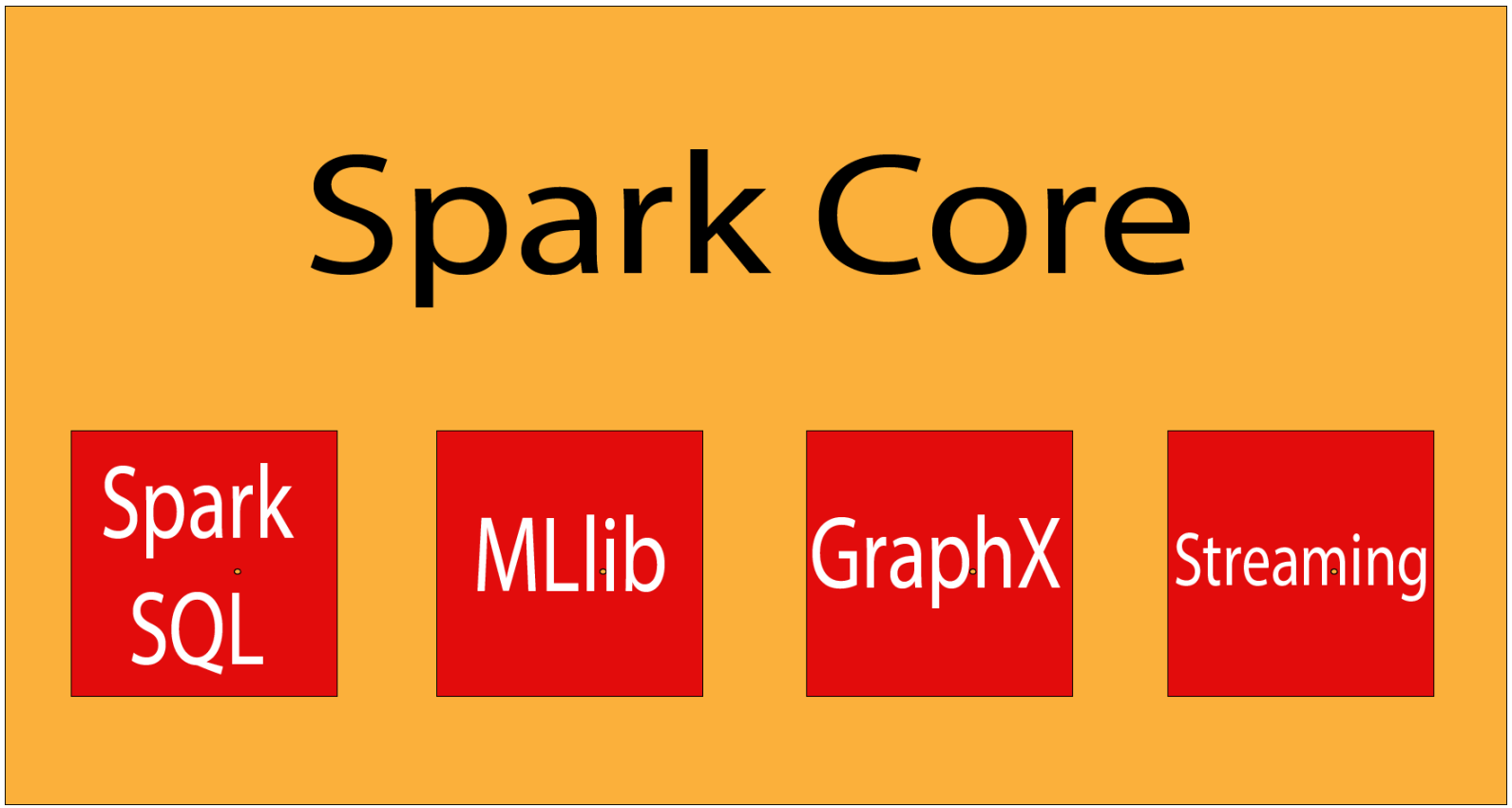
Various frameworks, APIs, and libraries for Big Data projects:






Spark

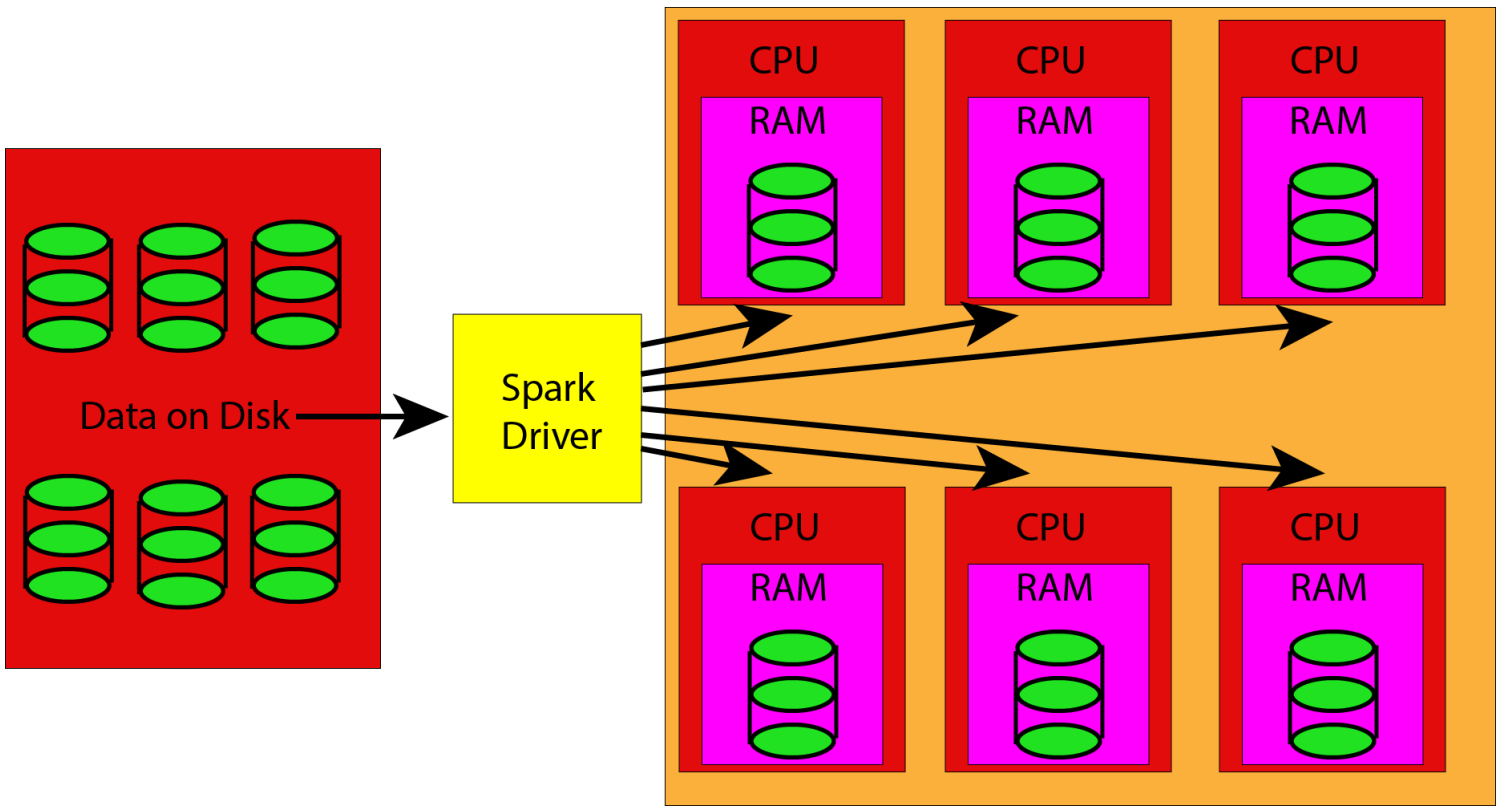
# Components of Spark 




- Spark has APIs that can work with
  - Python (PySpark) 🐍
  - R (SparkR)

# Spark Data 💾

## RDD - Resilient Distributed Dataset




- Large datasets can be distributed across multiple compute nodes 💻

# Data Persistence 💿

Spark supports different levels of persistence for performance optimization.

- MEMORY_ONLY
  - All RDD data is stored in-memory 🧠
    - Fastest processing time, but requires RAM to fit all data
- MEMORY_AND_DISK
  - RDD data is stored in memory and spills to disk if necessary
  - Balances between memory usage and processing speed
- DISK_ONLY
  - RDD data is stored only on disk
  - Minimal memory usage, but slower processing time

# Spark DataFrames 

Along with RDDs, Spark also has an API for DataFrames (similar to Pandas).

- SQL-like library that can handle data with named columns
- Great for structured/semi-structured data

# Spark Session 

SparkSession is the entry point for using the DataFrame and Dataset API.

- Underlying Spark functionality
- Creates the main Spark driver
- Used to create DataFrames and SQL-like tasks

```
spark = SparkSession.builder \
          .appName("MyPySpark") \
          .config("spark.driver.memory", "15g") \
          .getOrCreate()
```

SparkContext is the entry point for creating RDDs.

- Created with the `spark` object from `SparkSession.builder`

```
sc = spark.sparkContext
```

# Installing PySpark 🔧

Easiest way to install PySpark is by anaconda3.

This is great when running PySpark on a single compute node.

```
module load anaconda3
conda create -n mypyspark openjdk pyspark python=3.9 \
                          pyspark=3.3.0 py4j jupyterlab findspark \
                          h5py pytables pandas \
                          -c conda-forge -y
conda activate mypyspark
pip install ipykernel
ipython kernel install --user --name=mypyspark
```

This will create a conda env named, mypyspark, with access to Jupyter

This conda env will have both Spark and PySpark installed

> **Note**
>
> Information on using Anaconda can be found from a previous workshop
>
> https://github.com/ucla-oarc-hpc/H2HH_anaconda

# PySpark: Basic Operations 📋

Let's practice basic PySpark functions with examples.

- Download the workshop content from the GitHub repository
- We'll work with a Jupyter Notebook: Spark_basics.ipynb
- Jupyter Notebook: `Spark_basics.ipynb` from `spark-ex1`

```
cd $SCRATCH
git clone https://github.com/ucla-oarc-hpc/WS_BigDataOnHPC
cd WS_BigDataOnHPC
cd spark-ex1
```

We will download "The Hound of the Baskervilles", by Arthur Conan Doyle

- Data source: Project Gutenberg

```
wget https://www.gutenberg.org/files/3070/3070.txt
```

# PySpark: Basic operations: Starting the notebook

We will use the `h2jupynb` script to start Jupyter on Hoffman2

You will run this on your LOCAL computer.

```
wget https://raw.githubusercontent.com/rdauria/jupyter-notebook/main/h2jupynb
chmod +x h2jupynb

#Replace 'joebruin' with you user name for Hoffman2
#You may need to enter your Hoffman2 password twice

python3 ./h2jupynb -u joebruin -t 5 -m 10 -e 2 -s 1 -a intel-gold\\* \
                   -x yes -d /SCRATCH/PATH/WS_BigDataOnHPC/spark-ex1
```

> **Note**
>
> The `-d` option in the `python3 ./h2jupynb` will need to have the `$SCRATCH/WS_BigDataOnHPC` full PATH directory

This will start a Jupyter session on Hoffman2 with ONE entire intel-gold compute node (36 cores)

More information on the `h2jupynb` can be found on the Hoffman2 website

# PySpark: Machine Learning 🤖

This example will use Spark's Machine Learning library (MLlib)

We will use data from the Million song subset

This subset has ~500,000 songs with:

- Year of the song
- 90 features relating to the timbre average and covariance of the song

Download the dataset

```
cd $SCRATCH/WS_BigDataOnHPC
cd spark-ex2
wget https://archive.ics.uci.edu/ml/machine-learning-databases/00203/YearPredictionMSD.tx
unzip YearPredictionMSD.txt.zip
```

We will use multiple nodes to run Spark

# PySpark: Multi-node setup

In the previous example, we used pyspark with 1 (36-core) compute node.

- We will run PySpark over multiple nodes. This will:
  - increase the number of cores
  - increase the available RAM to fit a large dataset

To do this, we will NOT use the Spark installation from our conda install, but use spark from a build that we will download from the spark website.

```
mkdir -pv $SCRATCH/WS_BigDataOnHPC/apps/spark
cd $SCRATCH/WS_BigDataOnHPC/apps/spark

wget https://archive.apache.org/dist/spark/spark-3.3.0/spark-3.3.0-bin-hadoop3.tgz
tar -vxf spark-3.3.0-bin-hadoop3.tgz
```

**Note**

Though we will not use the Spark from conda, we will still use the PySpark package that was install with conda. The Spark and PySpark packages will need to be the same version (3.3.0 in this example)

# PySpark: Multi-node setup - Starting the notebook 

Since we are using our Spark build that we just downloaded, we will start spark and submit it as a job, then connect to jupyter.

- Example: `spark-ex2`
  - Job script `pyspark-multi-jupyter.job`
  - Starts Spark and a Jupyter session for us to connect.

In this example, we will use 3 compute nodes in total.

- One compute node will be the master that will run the Spark driver.
- The other two will be workers that will run the tasks

> **Tip**
>
> For large data jobs, I like to have the Spark driver to be separate from the workers.

Large data jobs may require the Spark driver to have a heavy CPU load and memory.

# PySpark: Multi-node setup - Starting the notebook 

- Submitting the Job

```
qsub pyspark-multi-jupyter.job
```

- Once the job starts, we will connect to this Jupyter session
  - The `spark-test.JOBID` file will display the MASTER node name
  - This master node will have the Spark driver and Jupyter process

Run this `ssh -L` command on your LOCAL computer

```
# Replace NODENAME with the name of the MASTER node
# Replace joebruin with you Hoffman2 user name
ssh -L 8888:NODENAME:8888 joebruin@hoffman2.idre.ucla.edu
```

- This will create a SSH tunnel to the master compute node so we can open Jupyter at `http://localhost:8888`
- Then we can open the notebook named `MSD.ipynb`

# Spark dashboard 📈

Spark has a visual dashboard that can view the tasks in real-time

- By default, Spark will run this dashboard on port 4040
- Create a ssh tunnel to the compute node to view the dashboard on your local machine

```
1  ssh -L 4040:NODENAME:4040 joebruin@hoffman2.idre.ucla.edu
```

You will need to replace NODENAME with the master compute node that has your Spark job

# Spark batch job ⏳

You can run Spark as a batch job to run non-interactively.

- We will use the command `spark-submit` to start the pyspark calucation located at:
  - `$SPARK_HOME/bin/spark-submit`

```
1 qsub pyspark-multi-batch.job
```

# Spark Bonus Example 

I have another Machine Learning example for Spark that I may not have time to go over in this workshop.

In this example, we will train a Machine Learning model using data from LIBSVM

- Inside of `spark-bonus`
  - Jupyter notebook: `ML-bonus.ipynb`
- Download the dataset

```
cd $SCRATCH/WS_BigDataOnHPC
cd spark-ex2
wget https://raw.githubusercontent.com/apache/spark/master/data/mllib/sampl
```

# Dask 🛠️

dask

# Intro to Dask 💡

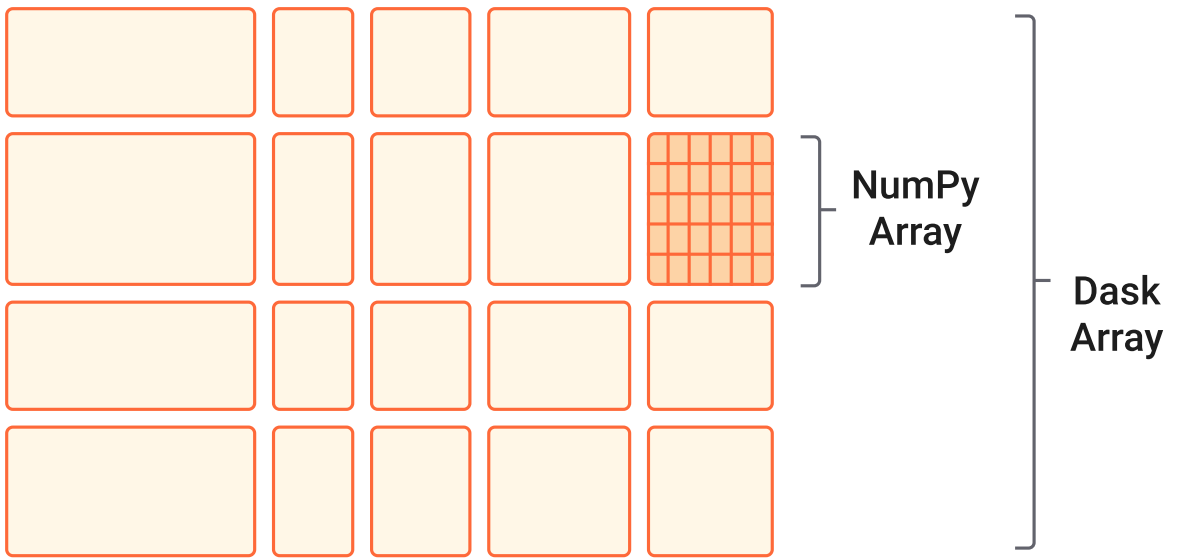
- Dask is a parallel computing library for Python 🐍
- Dask uses multiple cores to run tasks 💻
- Can use GPUs to speed up tasks 🚀
- Very minimal changes to Python code is required for Dask
  - Dask Arrays and Dataframes are similar to Numpy and Pandas

# Dask Array

Dask has an Arrays API created from NumPy-like chunks

## Typically Numpy code

```
1  import numpy as np
2
3  numpy_array = np.ones((100))
4
5  print(np.average(numpy_array))
```



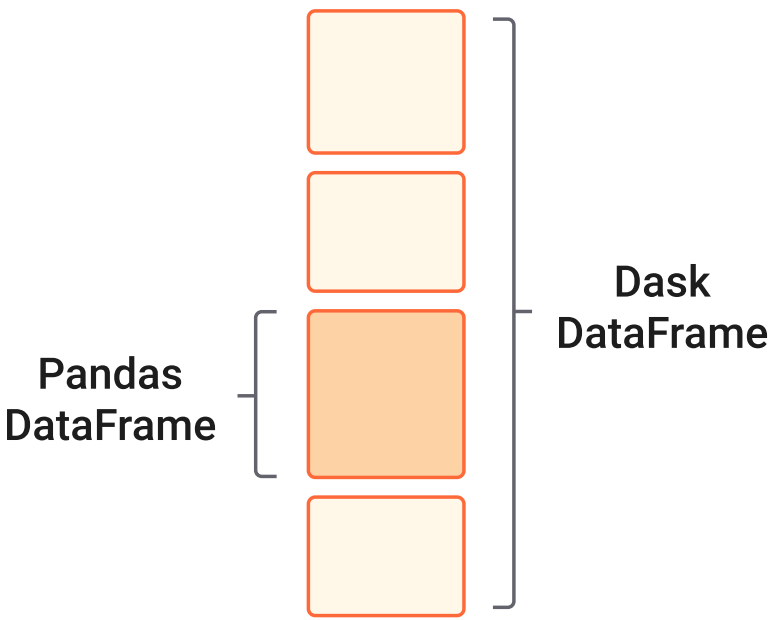

## Dask code

```
1  import dask.array as da
2
3  dask_array = np.ones((100), chunks=(10))
4
5  print(da.average(dask_array).compute())
```

- Dask Arrays can process chunks over multiple cores 🧠
- Great for larger-than-memory arrays; chunks are computed in memory
- Dask Arrays have similar functions and methods as Numpy objects

# Dask DataFrames :page_facing_up

Dask DataFrames are Pandas-like objects and are composed of Pandas-like “chunks”.




- Can lazily read data files (CSV, hdf5, etc.)

# Dask installation 🔧

```
module load anaconda3
conda create -n mydask python pandas jupyterlab  joblib seaborn \
                        dask dask-ml nodejs graphviz python-graphviz \
                        -c conda-forge -y
conda activate mydask
pip install ipykernel
ipython kernel install --user --name=mydask
```

This will create a conda env, `mydask`, that will have

- Dask
- Dask-ml (Machine Learning)
- Jupyter
- scikit-learn/joblib

# Dask: Basic Operations 📋

We will use the `h2jupynb` script to start Jupyter on Hoffman2

You will run this on your **LOCAL** computer.

```
python3 ./h2jupynb -u joebruin -t 5 -m 10 -e 2 -s 1 -a intel-gold\\* -x yes \
                   -d /SCRATCH/PATH/WS_BigDataOnHPC/dask-ex1
```

Replace `joebruin` with your Hoffman2 user account.

Replace `/SCRATCH/PATH/WS_BigDataOnHPC` with the full PATH name of the workshop on Hoffman2

- Let's go to `dask-ex1`
  - Jupyter notebook `dask_basic.ipynb`

# Dask: Machine Learning 🤖

Dask has a Dask-ML library with scalable Machine Learning methods. There is also integration with:

- Scikit-Learn and Joblib
- XGBoost
- PyTorch
- TensorFlow and Keras

# Dask ML Example 

This example will use Scikit-Learn with Dask

We will use data from the Million song subset

- Download the dataset

```
cd $SCRATCH/WS_BigDataOnHPC/dask-ex2
wget https://archive.ics.uci.edu/ml/machine-learning-databases/00203/YearPredictionMSD.txt.zip
unzip YearPredictionMSD.txt.zip
```

- Start Jupyter notebook
  - Replace `joebruin` with your Hoffman2 user account
  - Replace `/SCRATCH/PATH/WS_BigDataOnHPC` with the full PATH name of the workshop on Hoffman2

```
python3 ./h2jupynb -u joebruin -t 5 -m 10 -e 2 -s 1 -a intel-gold\\* -x yes -d /SCRATCH/PATH/WS_BigDa
```

- Jupyter Notebook: `MSD-dask.ipynb`

# Dask dashboard 📈

Dask has a visual dashboard that can view the tasks in real-time

- By default, Dask will run this dashboard on port 8787

- Create a ssh tunnel to the compute node to view the dashboard on your local machine

```
ssh -L 8787:NODENAME:8787 joebruin@hoffman2.idre.ucla.edu
```

You will need to replace NODENAME with the compute node that has your Dask job

# Wrap-up 

# Big Data on HPC: A Taste of Possibilities

- 🚀 Just the Beginning: Today's experience is a glimpse into the vast world of Big Data.
- 🌎 Big Data for All: Accessible to every field! 👥
- 💡 Scale Up Your Research: Enhance your analysis capabilities regardless of your study or profession.
  - 🔍 Analyze More, Larger Datasets: Delve deeper into your data.
  - ⚙️ Complex Methods & Models: Execute more sophisticated techniques.
- 🧠 Think Critically about your data processing:
  - 📁 Data Storage: Reflect on how your data is stored.
  - 💻 Parallel Processing: Consider the number of CPU cores running in parallel.
  - 🧮 Memory Matters: Be mindful of memory considerations.
- 🛠️ Powerful Frameworks:
  - ✨ Spark & Dask: Just two of the many robust Big Data processing frameworks available.

# Final thoughts 💭

- ⚙️ CPU/Cores Utilization: Keep track of CPU and core usage:
  - 📋 Did you request the same amount of cores via SGE?
  - 🔧 Is your code set up to run with multiple cores?
  - 💻 Multi-node Computing: Are you using multiple compute nodes?
- 🧠 Memory Requirements: Evaluate how much memory is needed.
  - 💾 Data in Memory: Does all your data fit in memory?
  - 🚀 Out-of-Memory Tasks: Are you utilizing tasks that exceed available memory?

Optimize your resources for seamless project execution! 💪📈

# Thank you! ❤️

Questions? Comments?

- cpeterson@oarc.ucla.edu
- Look at for more Hoffman2 workshops at https://idre.ucla.edu/calendar