# Python for HPC

Andrea Zonca - SDSC

# Jupyter Notebook

Data exploration in your browser

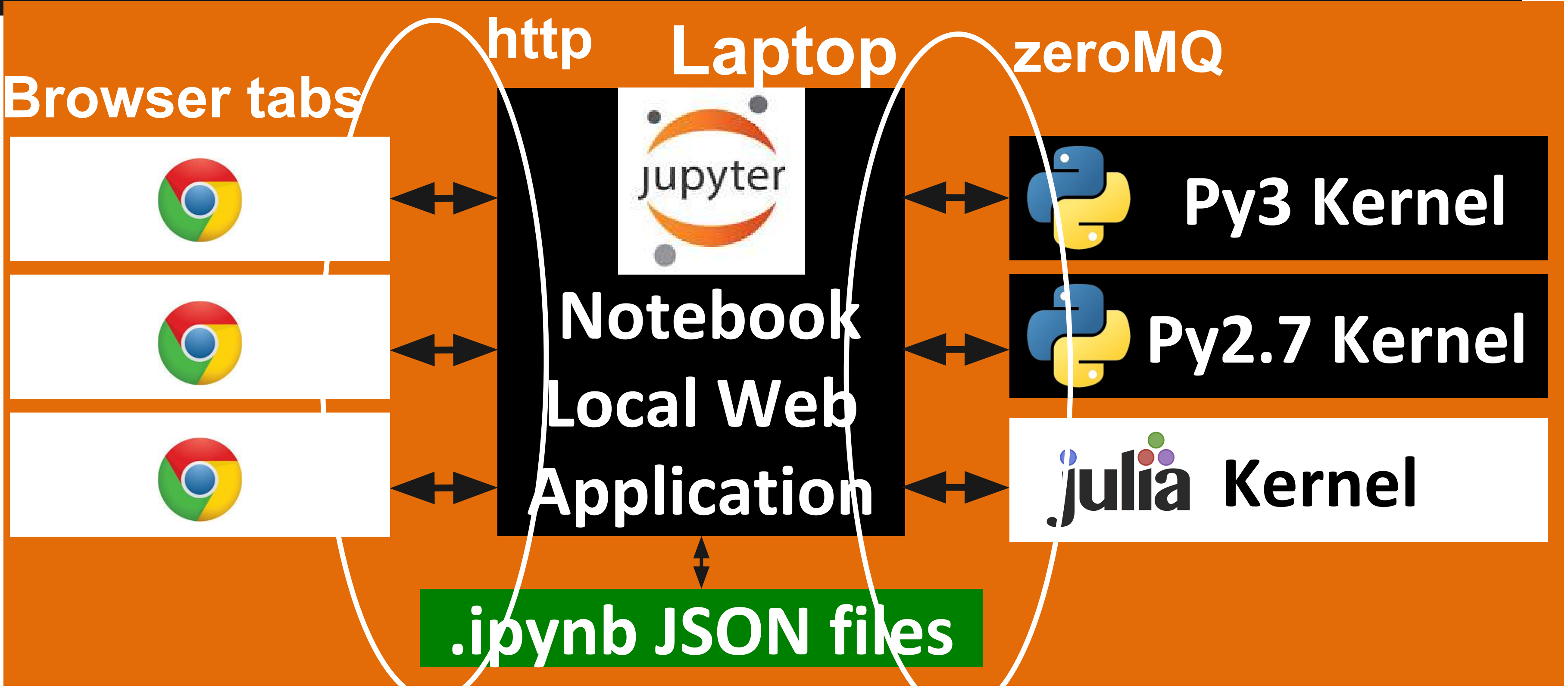
## What is the notebook?

- Browser based interactive console
- Supports multiple sessions in browser tabs
- Each session has a Kernel executing computation
- Saved in JSON format

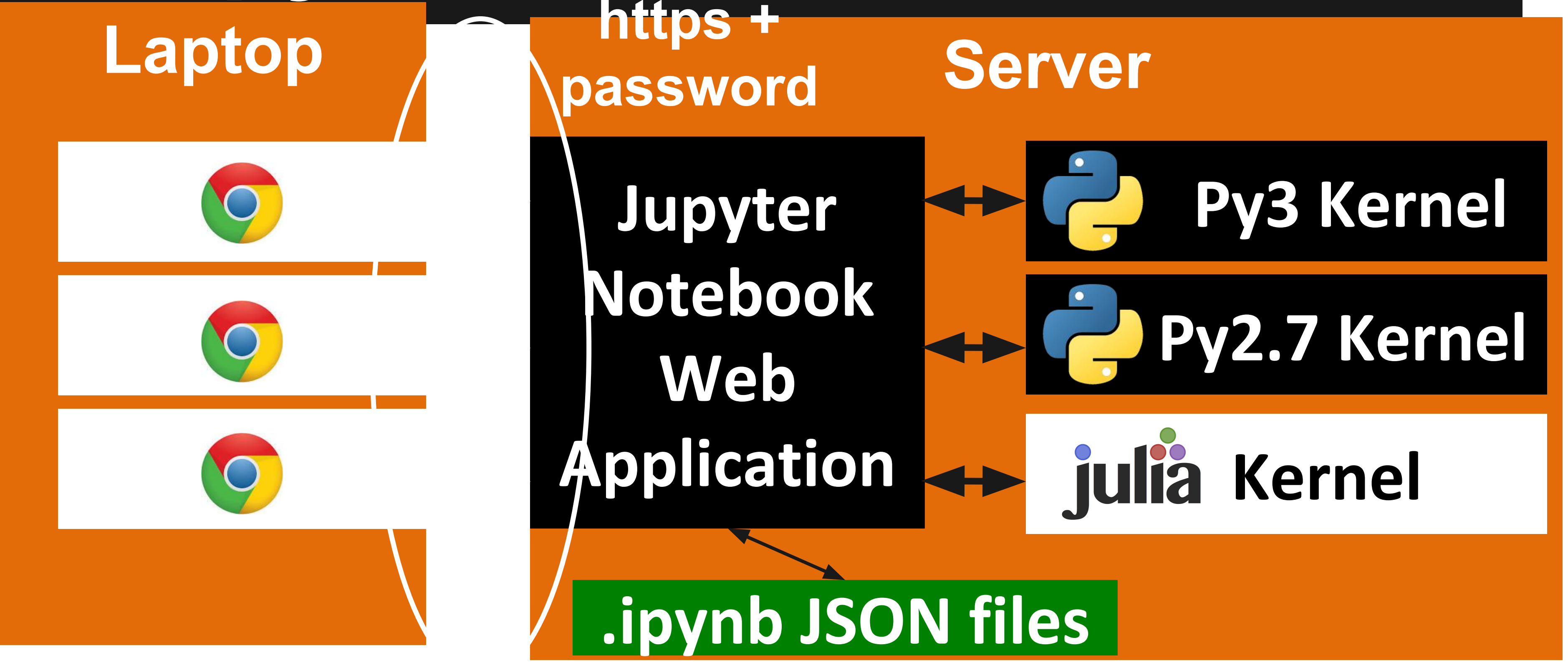
## Notebooks on Nature

http://www.nature.com/news/interactive-notebooks-sharing-the-code-1.16261

# Jupyter notebook local



# Jupyter notebook remote



## Clone workshop repository

ssh into comet with training account

```
git clone URL
```

Find url at the address bit.ly/cometucsb

## More secure setup

http://zonca.github.io/2015/09/ipython-jupyter-notebook-sdsc-comet.html

## IPython notebook demo

- Python code
- Formatted text
- Equations
- Plots
- Cells execution, cells order
- Clear output

## Why the notebook?

- Literate programming: code and explanation together
- Reproducible science: document easily every step
- Easy to share computations: send one single notebook instead of scripts/plots/.doc

## ipynb documents

- JSON format
- includes plots in binary format
- easy to convert to .html/.pdf for sharing
- http://nbviewer.ipython.org
- Recently rendered automatically on Github

## HPC: interactive notebooks

- Analyze large amount of data
- In-situ visualization
- Centralized Python stack
- Check long-running computations
- Prepare and submit batch jobs

## Notebooks as scripts

- Run notebooks not-interactively:

```
jupyter  nbconvert --execute  --to notebook
input_notebook.ipynb  --output
executed_notebook.ipynb
```

# Dask

## Multithreading and Distributed computing

## Numba

Run code on GPU with Python

## JIT compiler for Python

- based on LLVM (compiler infrastructure behind clang, Apple's C++ compiler)
- turns Python code into machine code
- on-the-fly

## Numba

```
export NUMBAPRO_NVVM=/usr/local/cuda-7.0/nvvm/lib64/libnvvm.so
export NUMBAPRO_LIBDEVICE=/usr/local/cuda-7.0/nvvm/libdevice/
```

## Interactive GPU node

```
salloc --nodes=1 --tasks-per-node=24
--partition=gpu -t 01:00:00
```

```python
from numba import jit
from numpy import arange

# jit decorator tells Numba to compile this function.
# The argument types will be inferred by Numba when function is called.
@jit
def sum2d(arr):
    M, N = arr.shape
    result = 0.0
    for i in range(M):
        for j in range(N):
            result += arr[i,j]
    return result

a = arange(9).reshape(3,3)
print(sum2d(a))
```

## Numba CPU

run with %timeit

increase size of matrix to see performance improvements

# Numba GPU

```python
from numba import cuda

@cuda.jit
def matmul(A, B, C):
    """Perform square matrix multiplication of C = A * B
    """
    i, j = cuda.grid(2)
    if i < C.shape[0] and j < C.shape[1]:
        tmp = 0.
        for k in range(A.shape[1]):
            tmp += A[i, k] * B[k, j]
        C[i, j] = tmp

import numpy as np
shape = (5,5)
a = np.ones(shape)
b = np.ones(shape) * 4
c = np.zeros(shape)
matmul[1,(16,16)](a,b,c)
print(c)
```

## Hands-on

- create a loop that runs matmul with different matrix sizes
- compare with np.dot
- range from 20x20 to 10000x10000
- plot timing

## Advanced CUDA

Tiled matrix multiplication to exploit GPU fast local memory:

http://numba.pydata.org/numba-doc/0.27.0/cuda/examples.html