Towards end-to-end data-management for large scale x-ray facilities

Professor Brian Vinter

Niels Bohr Institute
University of Copenhagen

MSCA-ITN-2017 under the European Union’s H2020 program

Grant No.765604

Towards end-to-end data-management for large scale ~~x-ray~~ facilities

Professor Brian Vinter

Niels Bohr Institute
University of Copenhagen

MSCA-ITN-2017 under the European Union's H2020 program

Grant No.765604

Tons of good news in x-ray technology

- Sources are getting brighter
- Sources are getting still more stable
- Spatial detector resolution grows exponentially
- Temporal detector resolution grows exponentially
- The x-ray user community is growing

The downside

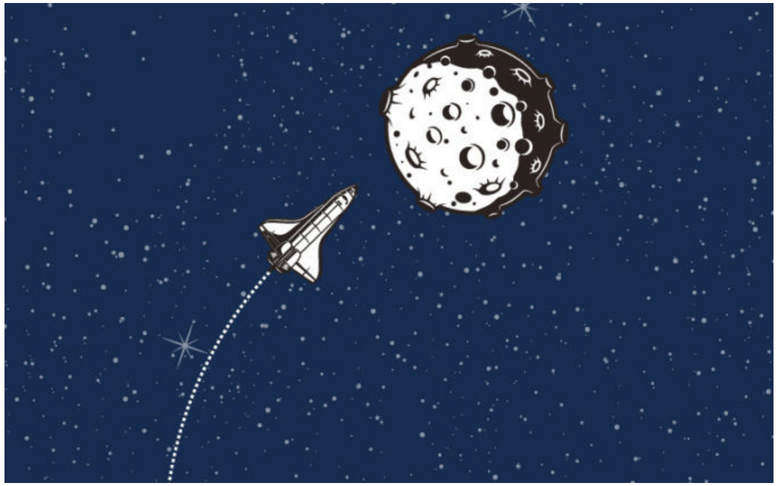
- The size of each detection grows exponentially
- Frequency of detection grows exponentially
- Frequency of experiment grows rapidly

The challenges

- We need to store much more data
- Individual datasets are too large for a PC to store
- Individual datasets are too large for a PC to process
- Hand-me-down Matlab scripts are not usable for such large datasets
- Many of the new user communities are not computing natives

Moonshot proposal

Can we build a software framework that supports huge datasets, has a user friendly interface, offers an easy-to-use compute service, and facilitates cross-organizational collaboration?

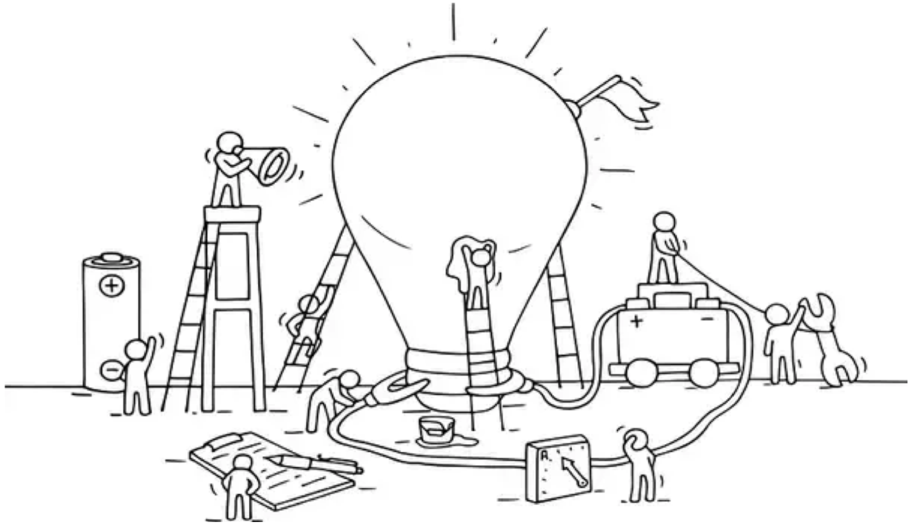


Initial Requirements

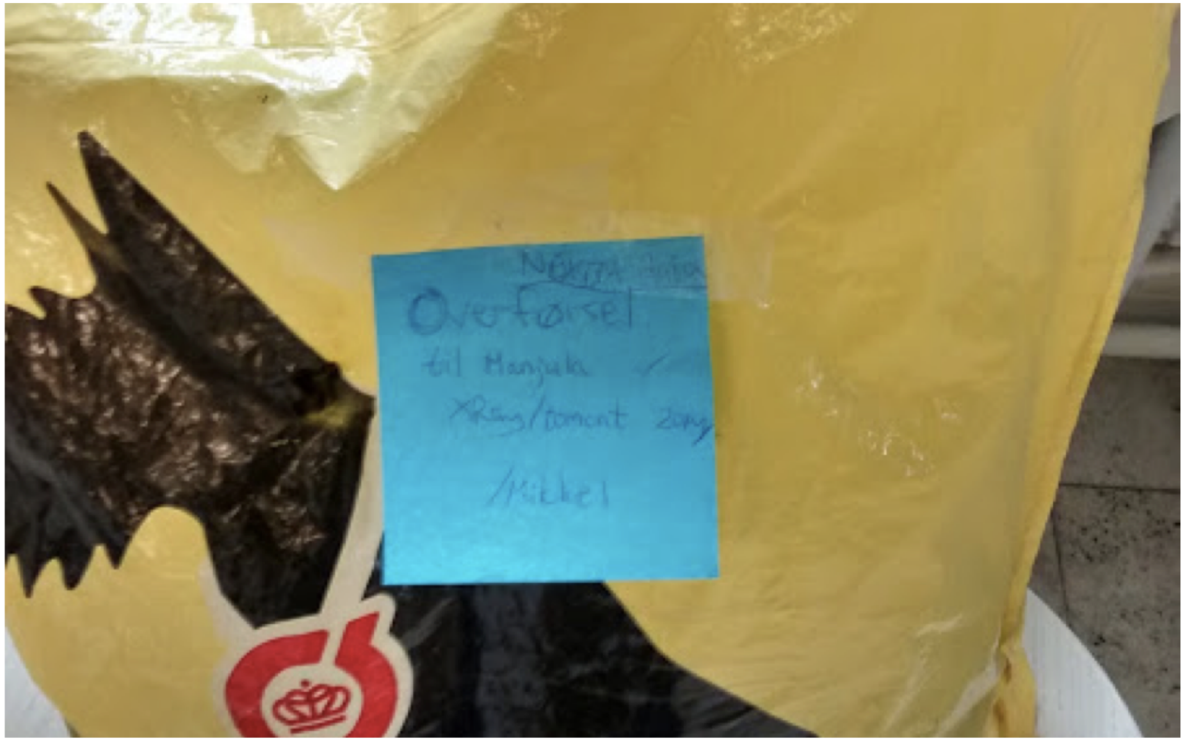
- Large data-storage
  - Fast storage
  - Data Management features
    - Online Inspection
    - Archiving
- Build-in processing support
  - Interactive
  - Batch Processing
- Cross Organization Support

Current Prototype

- Imaging Data Management System
  - Not really imaging specific so poor choice of name!
- Alternative entry to UCPH ERDA system
  - 10 PB storage
  - File system
  - Project sharing
  - Folder Synchronization
  - Jupyter Interactive Processing
  - MiG grid-backend

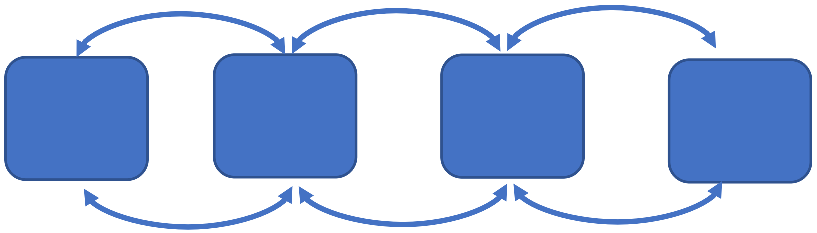


Large Data Storage

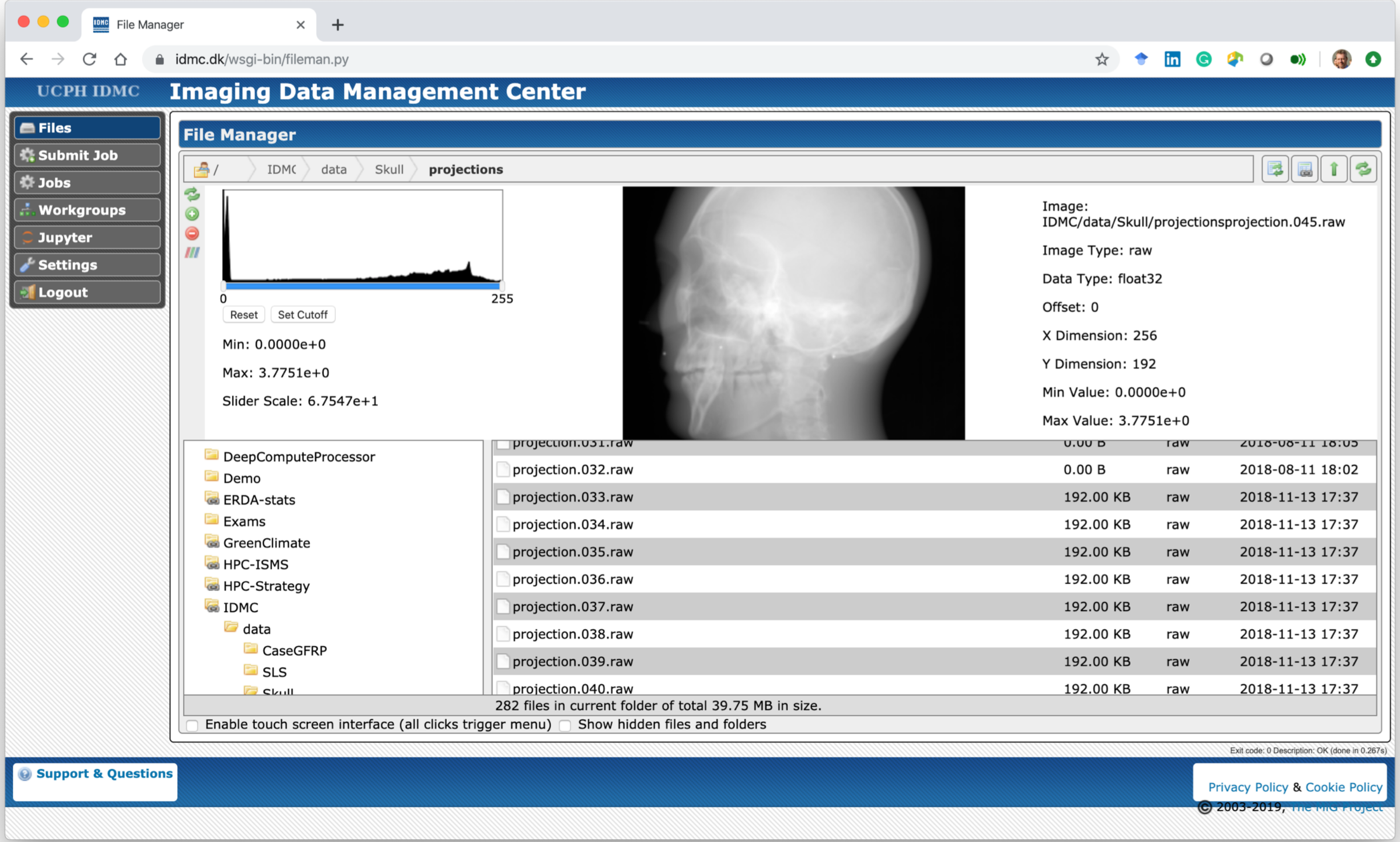


Large Storage

- Dirt cheap
- Much larger disk systems per node
- Disk redundancy
- Server redundancy
- 2 x 100 Gb input



Data Management



Min: 0.0000e+0

Max: 3.7751e+0

Slider Scale: 6.7547e+1

Image: IDMC/data/Skull/projectionsprojection.045.raw

Image Type: raw

Data Type: float32

Offset: 0

X Dimension: 256

Y Dimension: 192

Min Value: 0.0000e+0

Max Value: 3.7751e+0

- DeepComputeProcessor
- Demo
- ERDA-stats
- Exams
- GreenClimate
- HPC-ISMS
- HPC-Strategy
- IDMC
  - data
    - CaseGFRP
    - SLS
    - Skull

projection.031.raw	0.00 B	raw	2018-08-11 18:05
projection.032.raw	0.00 B	raw	2018-08-11 18:02
projection.033.raw	192.00 KB	raw	2018-11-13 17:37
projection.034.raw	192.00 KB	raw	2018-11-13 17:37
projection.035.raw	192.00 KB	raw	2018-11-13 17:37
projection.036.raw	192.00 KB	raw	2018-11-13 17:37
projection.037.raw	192.00 KB	raw	2018-11-13 17:37
projection.038.raw	192.00 KB	raw	2018-11-13 17:37
projection.039.raw	192.00 KB	raw	2018-11-13 17:37
projection.040.raw	192.00 KB	raw	2018-11-13 17:37

282 files in current folder of total 39.75 MB in size.

□ Enable touch screen interface (all clicks trigger menu) □ Show hidden files and folders

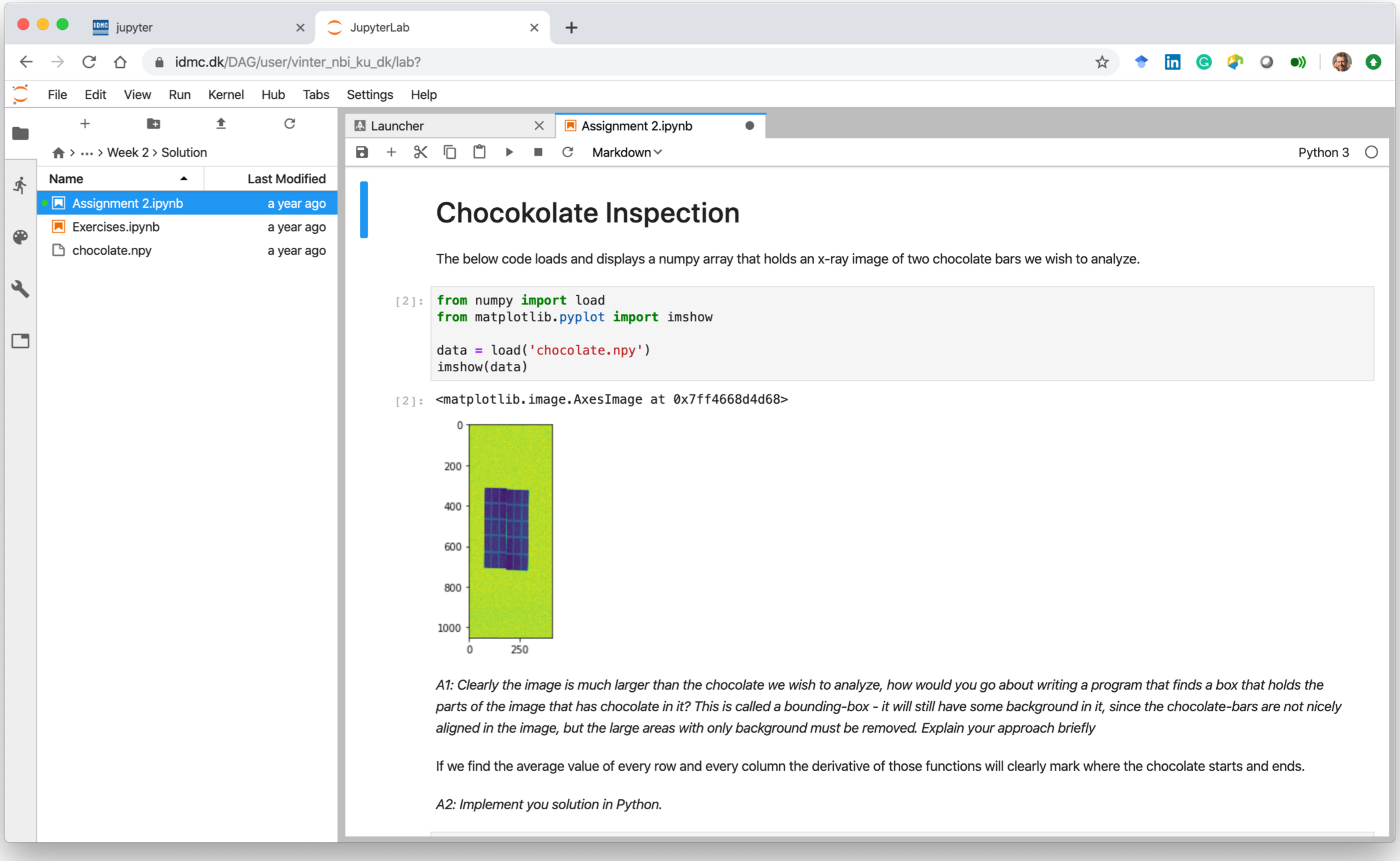
Exit code: 0 Description: OK (done in 0.267s)

© 2003-2019, The MIG Project

Interactive Processing

- Jupyter based interface
  - Python
  - R
  - C++
  - C#
  - (others are possible)
- Three kinds of resources
  - DAG (64 cores 256 GB memory)
  - HEL (DGX-1)
  - MODI (Cluster: 512 cores, 1TB memory)

Interactive Processing


jupyter
JupyterLab
idmc.dk/DAG/user/vinter_nbi_ku_dk/lab?
File Edit View Run Kernel Hub Tabs Settings Help
Week 2 > Solution
Name
Last Modified
Assignment 2.ipynb a year ago
Exercises.ipynb a year ago
chocolate.npy a year ago
Launcher
Assignment 2.ipynb
Markdown
Python 3
0
200
400
600
800
1000
0
250

Chocokolate Inspection

The below code loads and displays a numpy array that holds an x-ray image of two chocolate bars we wish to analyze.

[2]:
```
from numpy import load
from matplotlib.pyplot import imshow

data = load('chocolate.npy')
imshow(data)
```

[2]:
```
<matplotlib.image.AxesImage at 0x7ff4668d4d68>
```

A1: Clearly the image is much larger than the chocolate we wish to analyze, how would you go about writing a program that finds a box that holds the parts of the image that has chocolate in it? This is called a bounding-box - it will still have some background in it, since the chocolate-bars are not nicely aligned in the image, but the large areas with only background must be removed. Explain your approach briefly

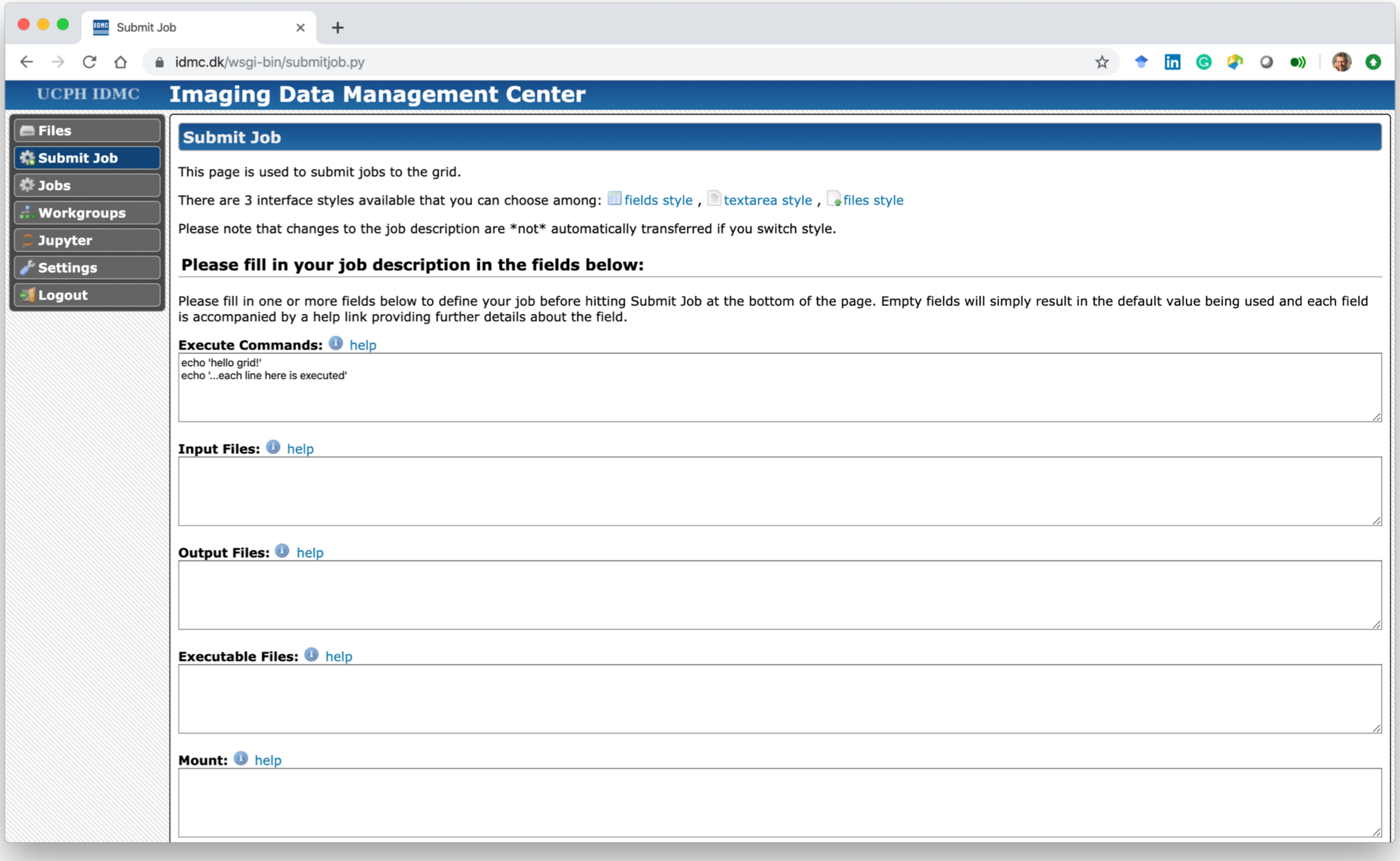
If we find the average value of every row and every column the derivative of those functions will clearly mark where the chocolate starts and ends.

A2: Implement you solution in Python.

Batch Processing

- Run completely at user-level
- No custom grid software
  - On Unix based systems
- Resource Owners decide which projects can use their computers
  - And when
- Automatic error recovery

Batch Processing



Imaging Data Management Center

Submit Job

This page is used to submit jobs to the grid.

There are 3 interface styles available that you can choose among: fields style , textarea style , files style

Please note that changes to the job description are *not* automatically transferred if you switch style.

Please fill in your job description in the fields below:

Please fill in one or more fields below to define your job before hitting Submit Job at the bottom of the page. Empty fields will simply result in the default value being used and each field is accompanied by a help link providing further details about the field.

Execute Commands: help

```
echo 'hello grid!'
echo '...each line here is executed'
```

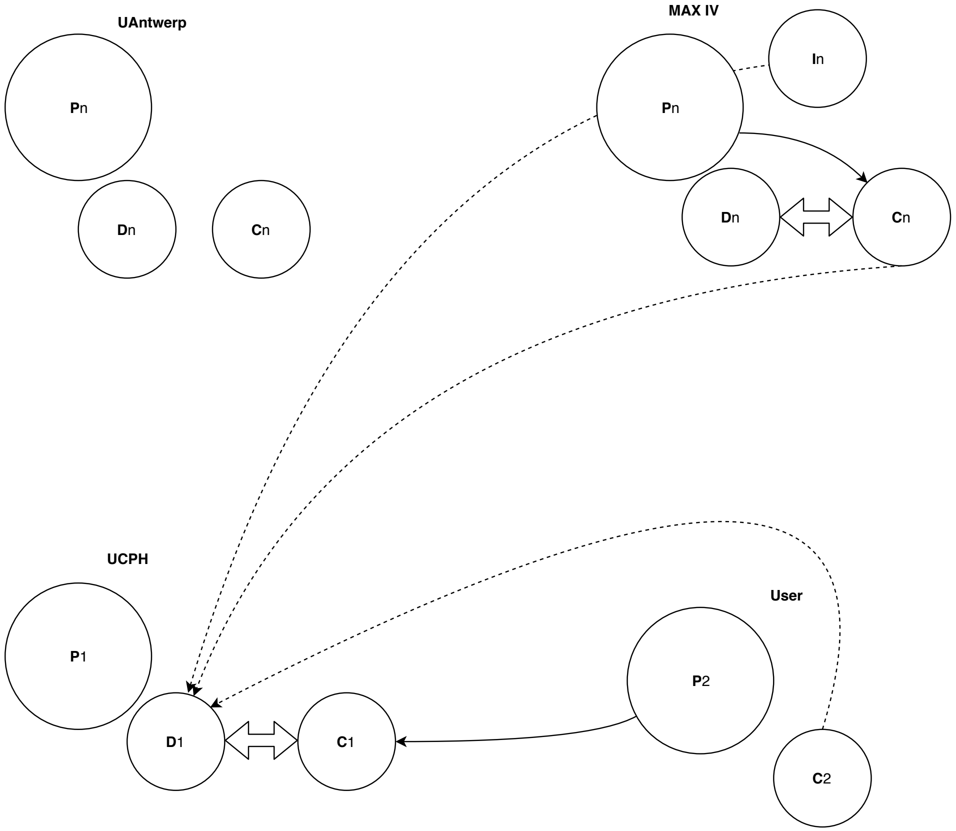
Input Files: help

Output Files: help

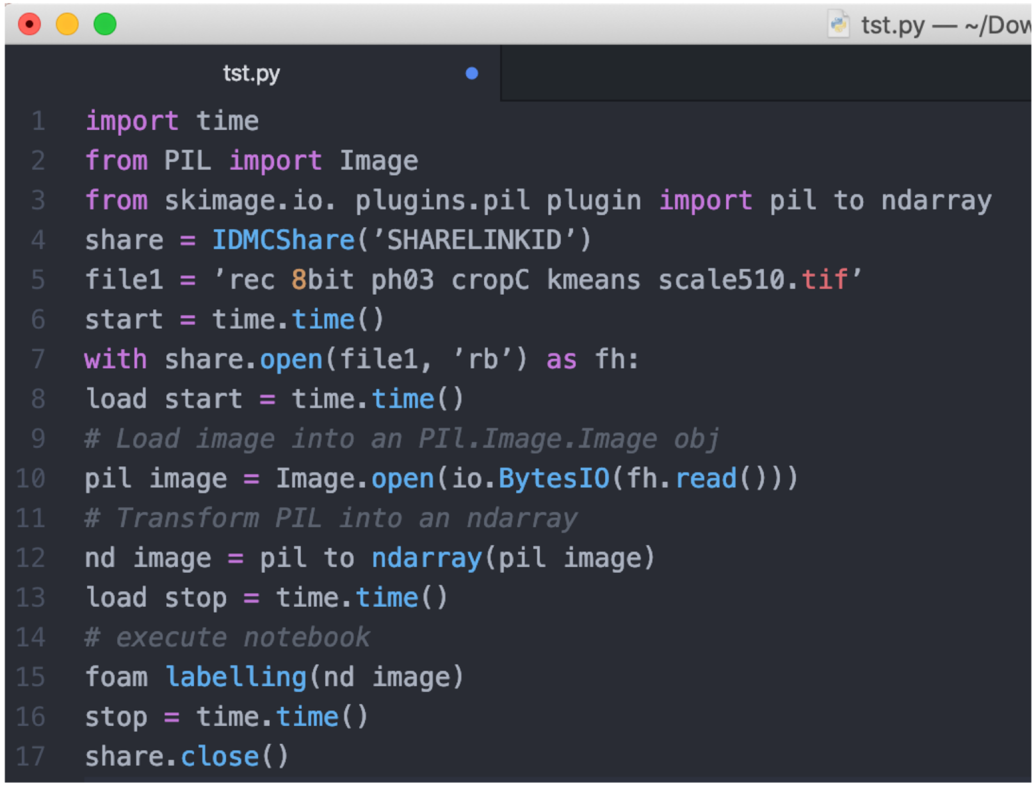
Executable Files: help

Mount: help

Cross Organization Support



Cross Organization Support



```
import time
from PIL import Image
from skimage.io. plugins.pil plugin import pil to ndarray
share = IDMCShare(’SHARELINKID’)
file1 = ’rec 8bit ph03 cropC kmeans scale510.tif’
start = time.time()
with share.open(file1, ’rb’) as fh:
load start = time.time()
# Load image into an PIL.Image.Image obj
pil image = Image.open(io.BytesIO(fh.read()))
# Transform PIL into an ndarray
nd image = pil to ndarray(pil image)
load stop = time.time()
# execute notebook
foam labelling(nd image)
stop = time.time()
share.close()
```

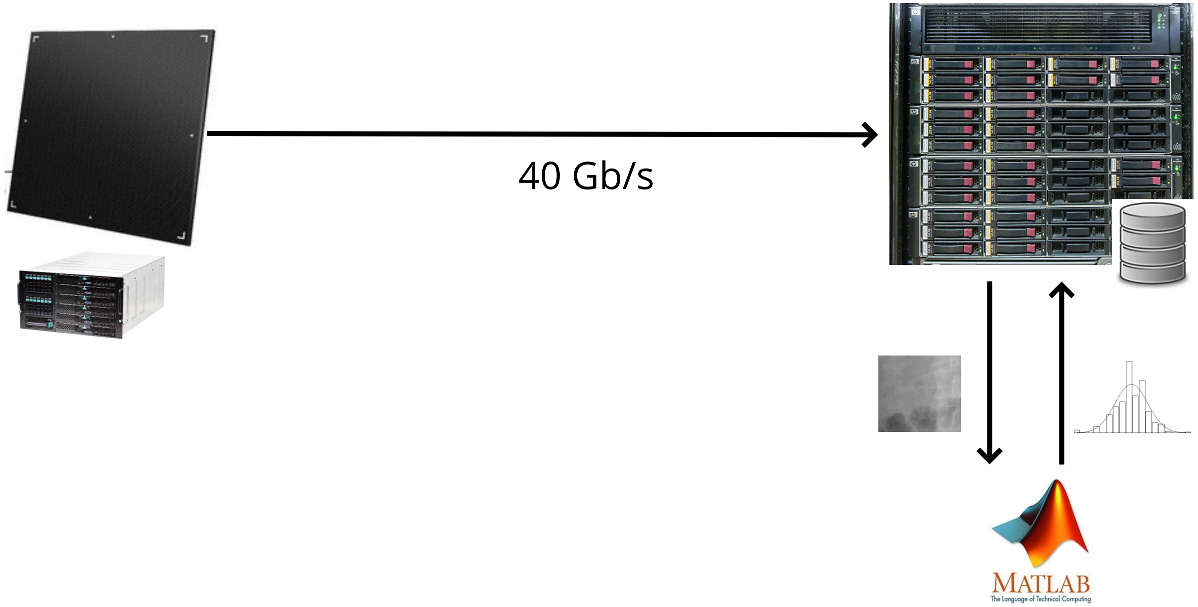
Initial Requirements

- ✓Large data-storage
- ✓Fast storage
- ✓Data Management features
  - ✓Online Inspection
  - ✓Archiving
- ✓Build-in processing support
  - ✓Interactive
  - ✓Batch Processing
- ✓Cross Organization Support

Future Developments

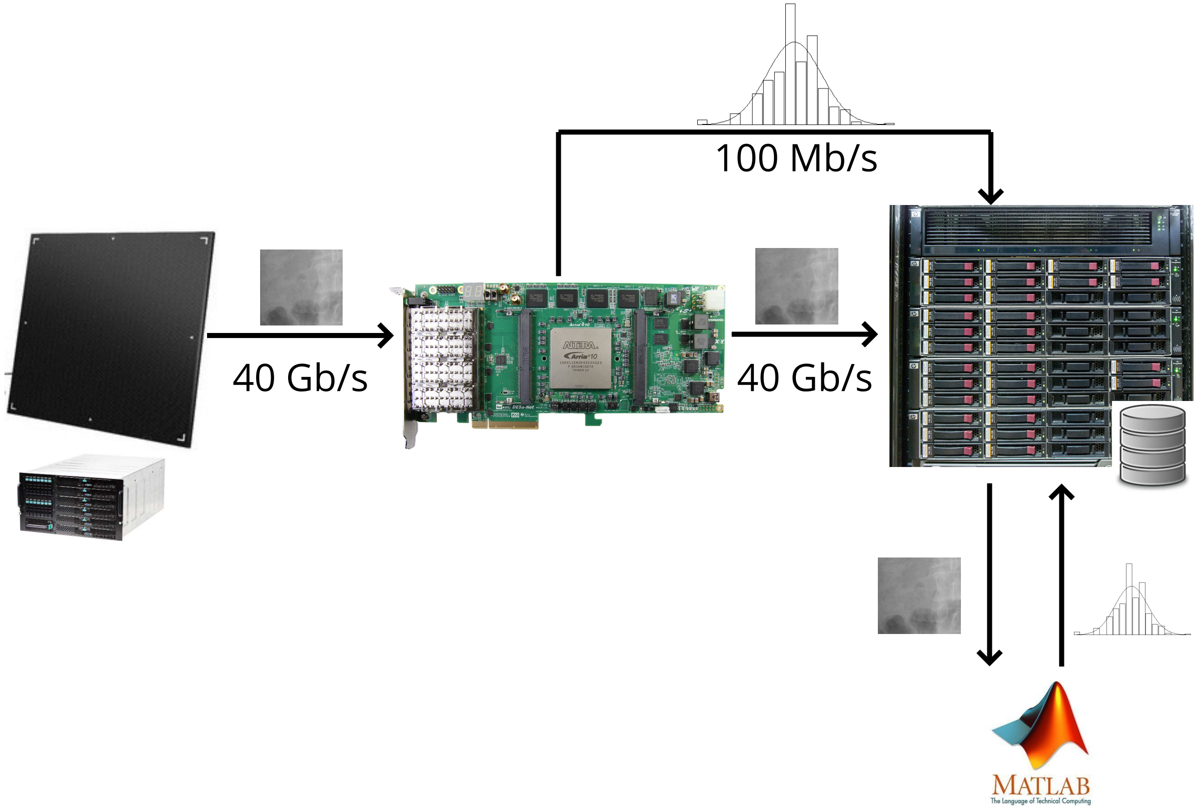
- High Speed Real Time Data Analysis
- Usage of low power storage
- Integrating batch-setups in Jupyter
- Securing data integrity with signing

High Speed Real Time Data Analysis



40 Gb/s

High Speed Real Time Data Analysis

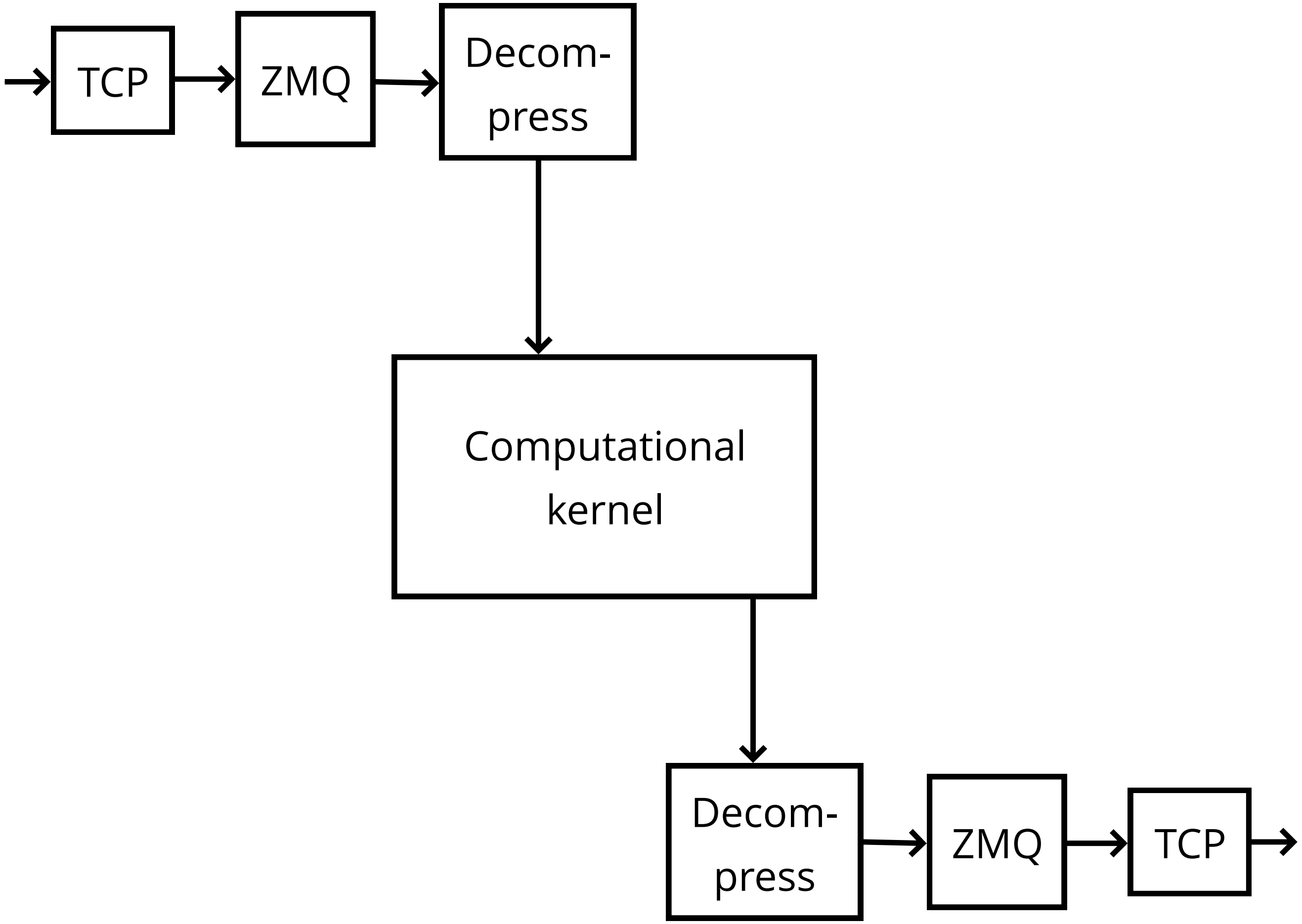


100 Mb/s

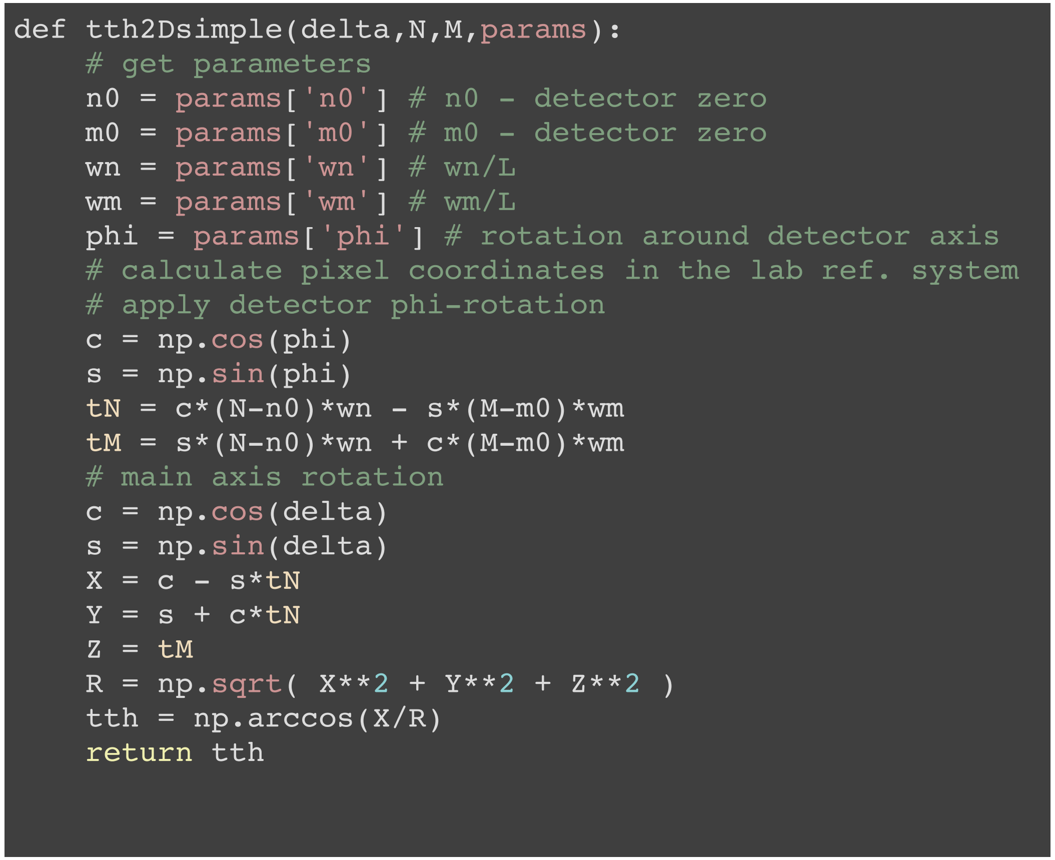
40 Gb/s

40 Gb/s

High Speed Real Time Data Analysis



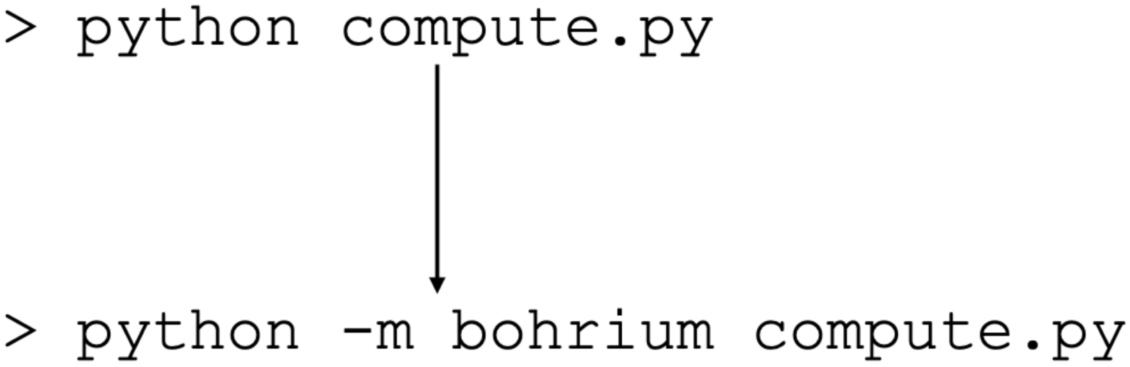
High Speed Real Time Data Analysis



```
def tth2Dsimple(delta,N,M,params):
    # get parameters
    n0 = params['n0'] # n0 - detector zero
    m0 = params['m0'] # m0 - detector zero
    wn = params['wn'] # wn/L
    wm = params['wm'] # wm/L
    phi = params['phi'] # rotation around detector axis
    # calculate pixel coordinates in the lab ref. system
    # apply detector phi-rotation
    c = np.cos(phi)
    s = np.sin(phi)
    tN = c*(N-n0)*wn - s*(M-m0)*wm
    tM = s*(N-n0)*wn + c*(M-m0)*wm
    # main axis rotation
    c = np.cos(delta)
    s = np.sin(delta)
    X = c - s*tN
    Y = s + c*tN
    Z = tM
    R = np.sqrt( X**2 + Y**2 + Z**2 )
    tth = np.arccos(X/R)
    return tth
```

Bohrium

- Bohrium provides automatic acceleration of array operations in Python/NumPy, C, and C++ targeting multi-core CPUs and GP-GPUs.



```
> python compute.py
```

```
> python -m bohrium compute.py
```

Bohrium

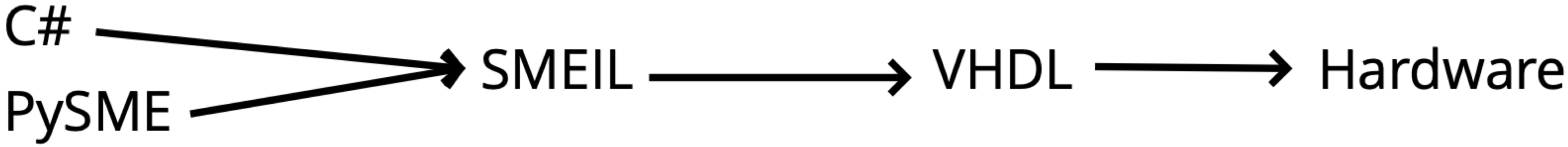
```python
import numpy as np

a = np.arange(10)
b = (a + 2) * a
c = np.histogram(b)
```

```
a = [10]
t0 = [10]
arange a, 10
add, t0, a, 2
mul b, t0, a
hist c, b
```

OpenMP

OpenCL CUDA

Synchronous Message Exchange



C#

PySME

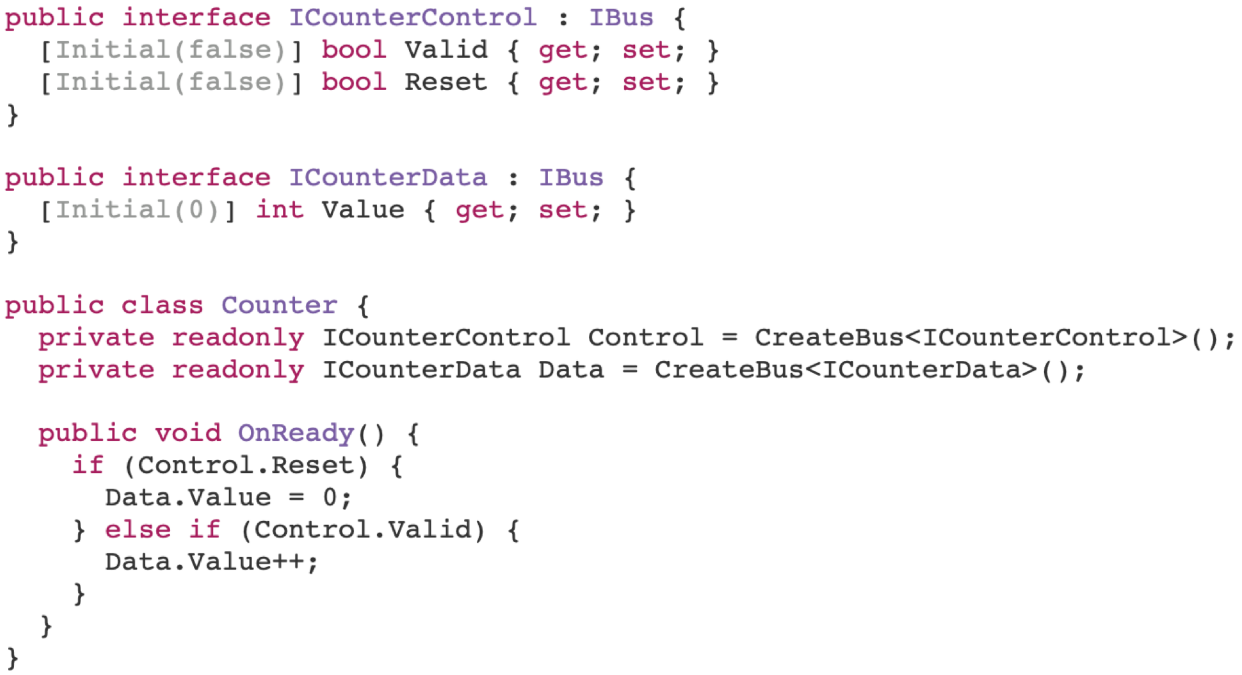
SMEIL

VHDL

Hardware

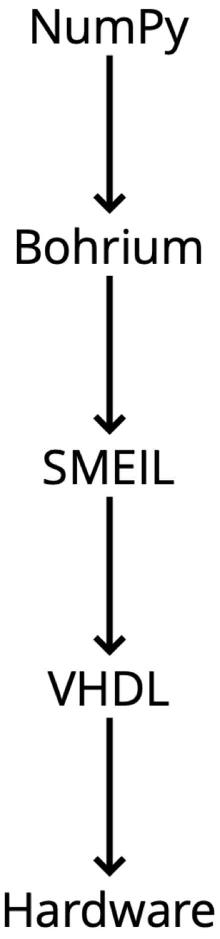
Synchronous Message Exchange

- Simple testing and debugging
- Human readable VHDL
- Automatic testbench



```
public interface ICounterControl : IBus {
  [Initial(false)] bool Valid { get; set; }
  [Initial(false)] bool Reset { get; set; }
}

public interface ICounterData : IBus {
  [Initial(0)] int Value { get; set; }
}

public class Counter {
  private readonly ICounterControl Control = CreateBus<ICounterControl>();
  private readonly ICounterData Data = CreateBus<ICounterData>();

  public void OnReady() {
    if (Control.Reset) {
      Data.Value = 0;
    } else if (Control.Valid) {
      Data.Value++;
    }
  }
}
```

Vision



NumPy

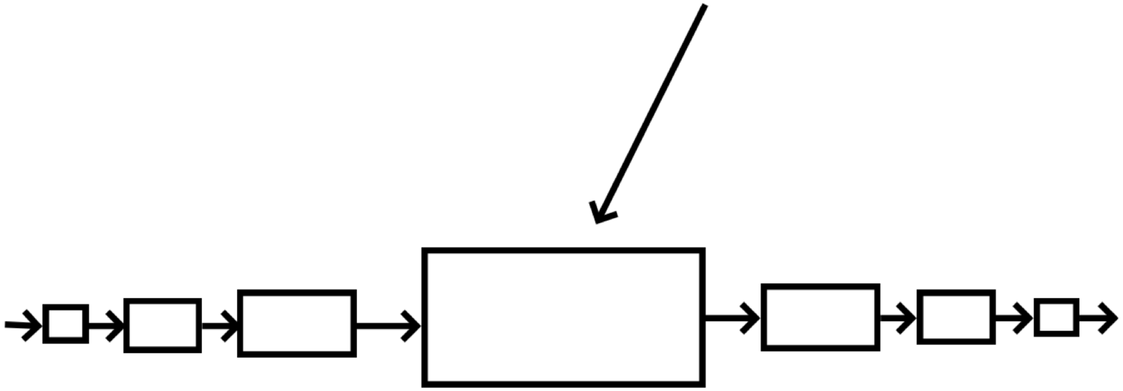
Bohrium

SMEIL

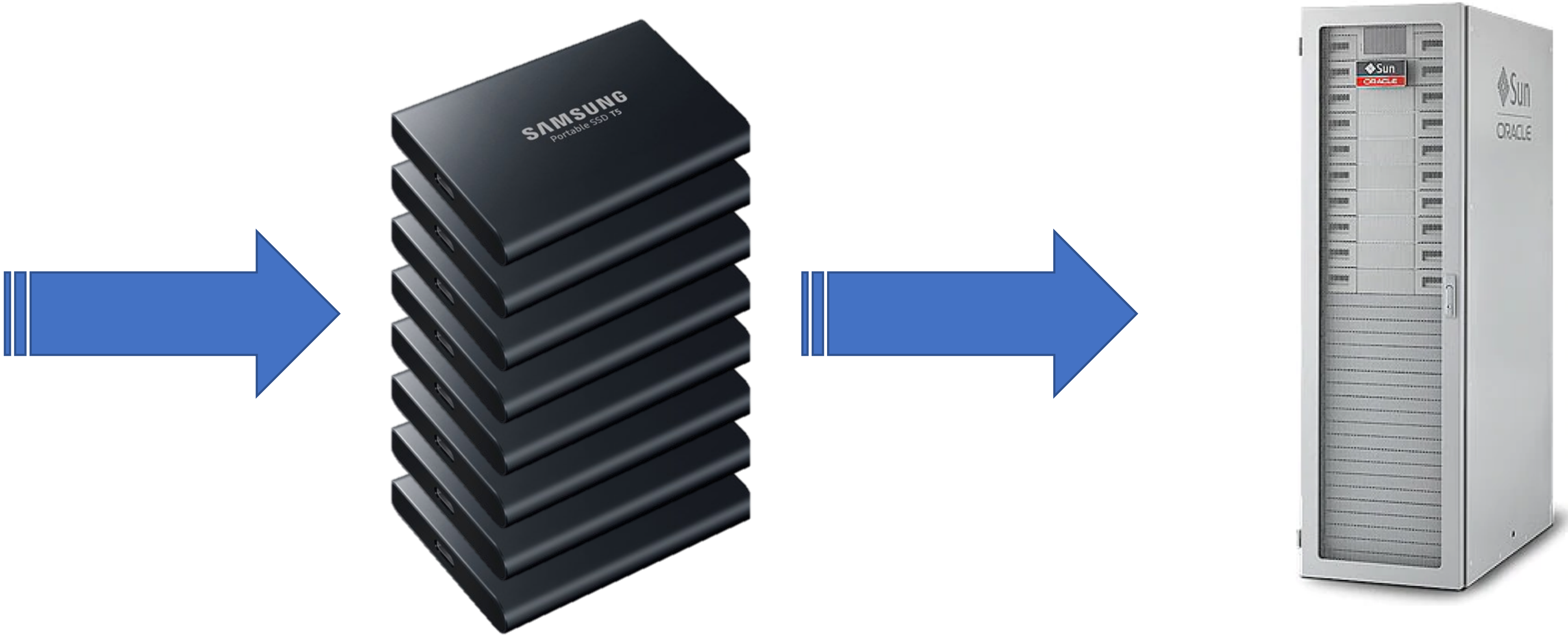
VHDL

Hardware

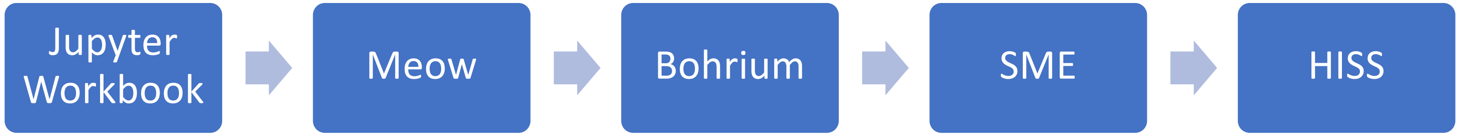
```
import numpy as np

def kernel(a, c):
  a[:] = np.arange(10)
  b = (a + 2) * a
  c[:] = np.histogram(b)
```



Grand Vision



Grand Vision



Proposal: Fighting Scientific Misconduct

- Scientific misconduct is a problem
- With FAIR it may become worse (or not!)
- Proposal: We establish a scientific block-chain
  - Instruments signs the raw data
  - Software that is provided by the facility also sign the result
- Outcome: We can trace the validity of data until the researcher runs untrusted software on the data
  - Which makes it very clear where the problem arises

That's all Folks!