

Prototyping and Deploying an Analysis Facility at a USCMS Tier2

Mat Adamec, Ken Bloom, Oksana Shadura,
University of Nebraska, Lincoln

Garhan Attebury, **Carl Lundstedt**, Derek Wietzel
University of Nebraska Holland Computing Center

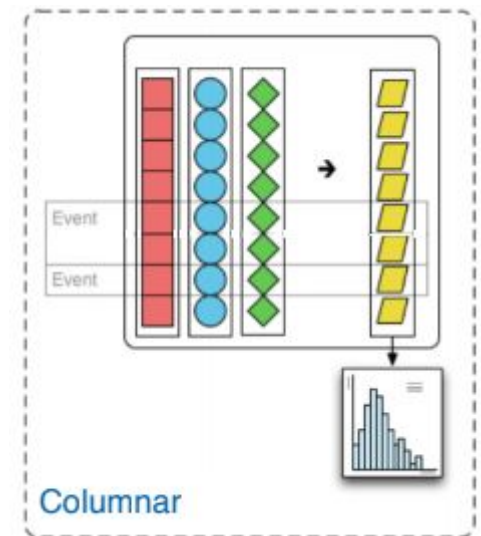
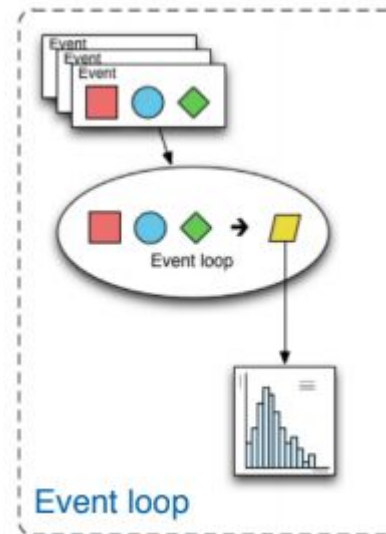
Mátyás Selmeçi
University of Wisconsin, Madison

Brian Bockelman
Morgridge Institute



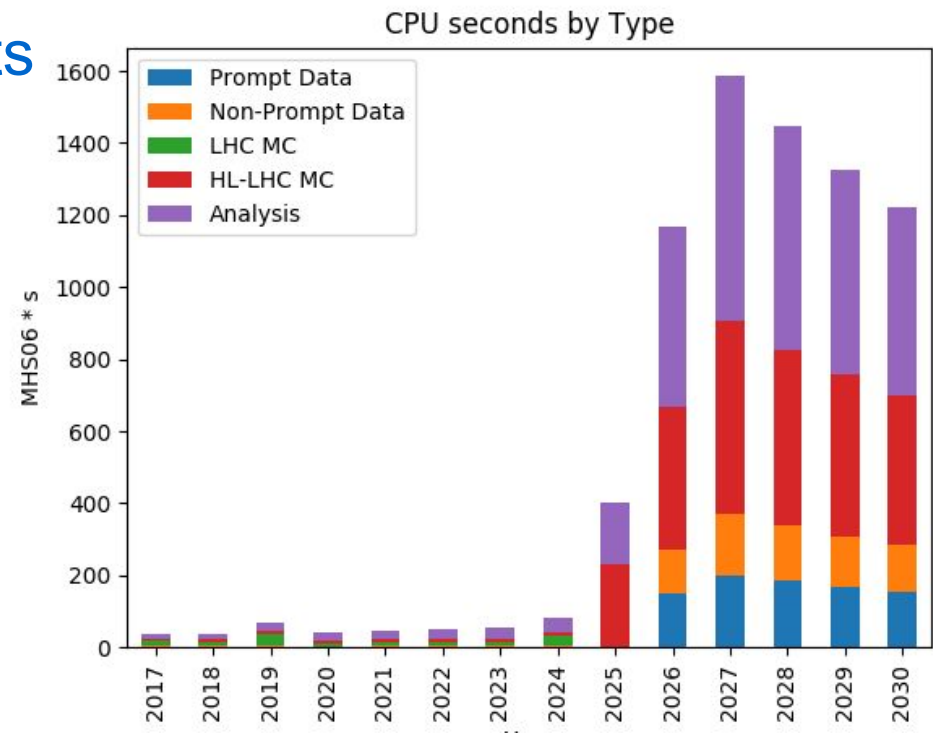
COFFEA - Columnar Object Framework For Effective Analysis

- Leveraging large data and data analysis tools from python to provide an array-based syntax for manipulating HEP event data.
- Stark contrast to venerable, well established event loop techniques.
- Tremendous potential to fundamentally change the time-to-science in HEP.
- Scales well horizontal
- Cannot easily utilize current analysis facilities (T2s) as the analysis is not grid friendly, it's meant to be quasi-interactive



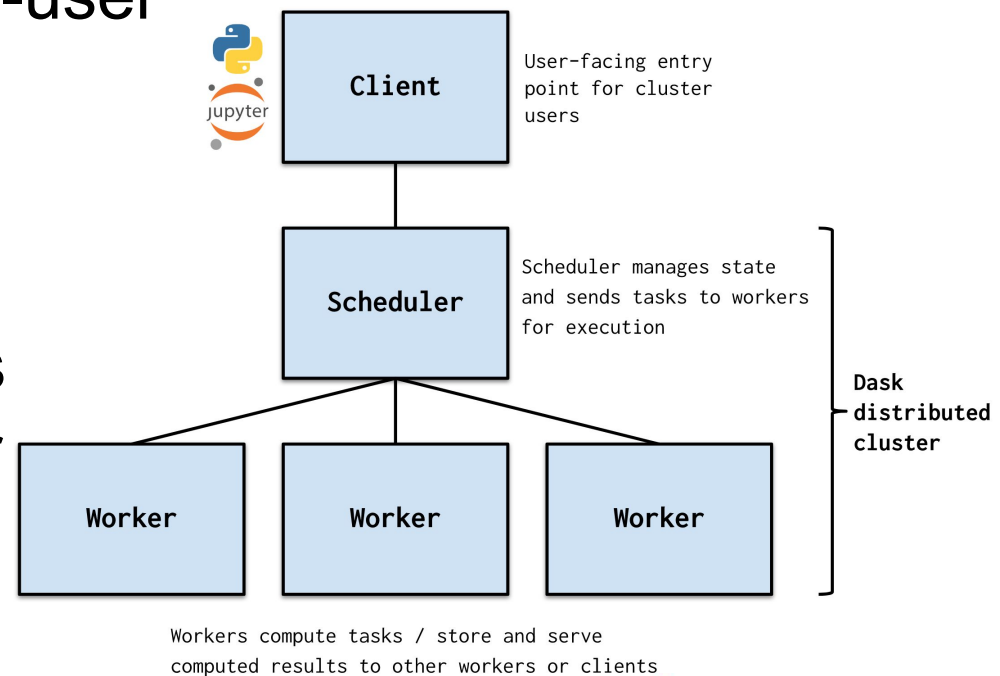
coffea – performance (Motivation)

- The Present Challenge
 - Analyze all LHC Run 2 data: $O(10 \text{ billion events})$
 - Investigate data quality issues with fast time-to-insight
 - Optimize complex (e.g. deep learning algorithms)
- Multiply by $O(1000)$ data analysts
- These challenges magnified 20x in HL-LHC



Dask

- “Dask is a flexible library for parallel computing in Python.”
- Think of Dask as run-time parallel + cluster plugin for python
- Easily installed via Conda as the module “distributed”.
- NOT really designed with multi-user environments in mind out-of-the-box.
- Integrates with HPC clusters running a variety of schedulers including SLURM & HTCondor via “dask-jobqueue”.



Emerging Facility Requirements

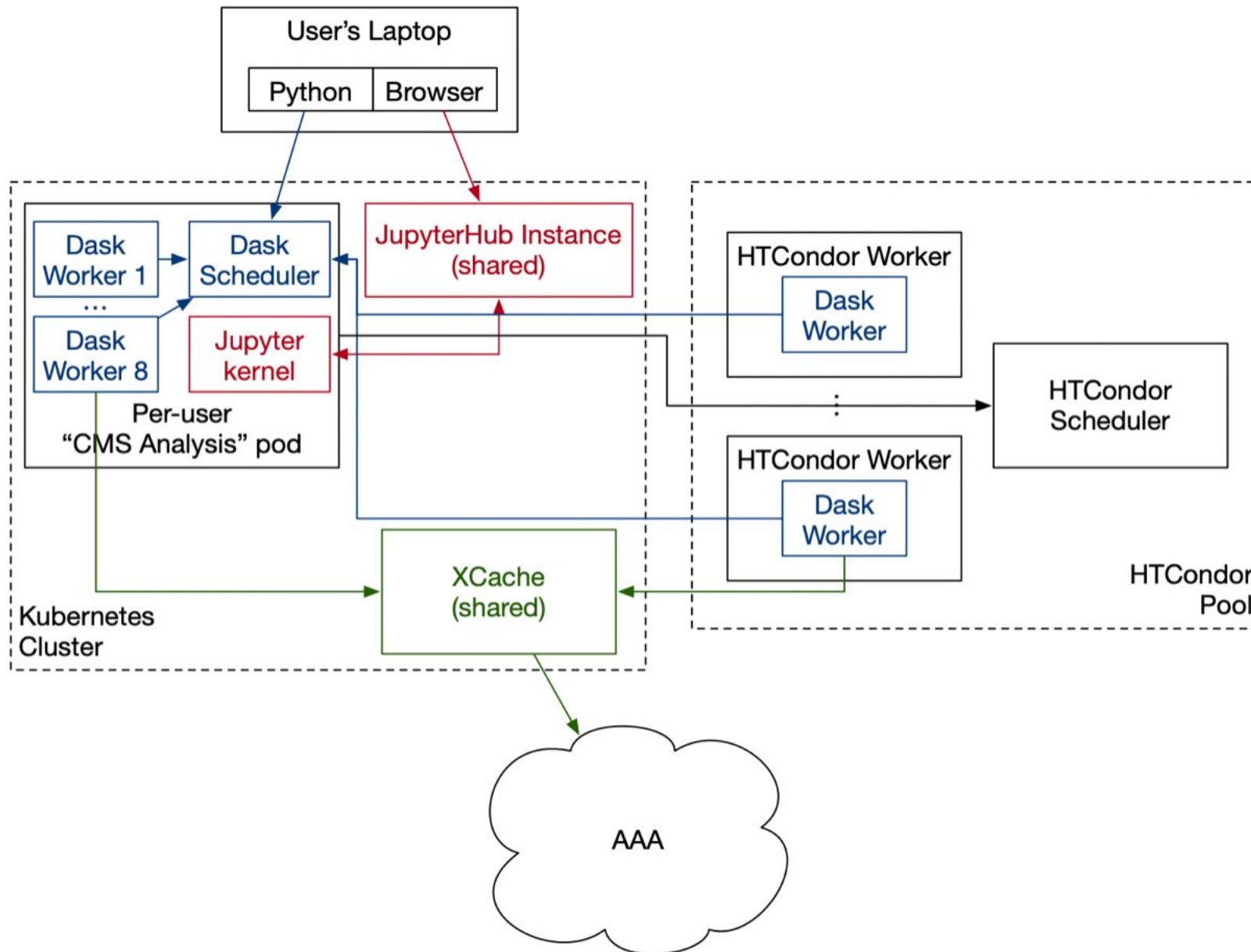
- Analysis Facilities must be (according to me):
 - Easy to use for users
 - Scalable (dynamically/automatically)
 - Responsive/Interactive
 - Utilize currently deployed hardware/middleware
 - Minimally intrusive for site administrators
 - Get work ('effort' & CPU) accounted for by CMS



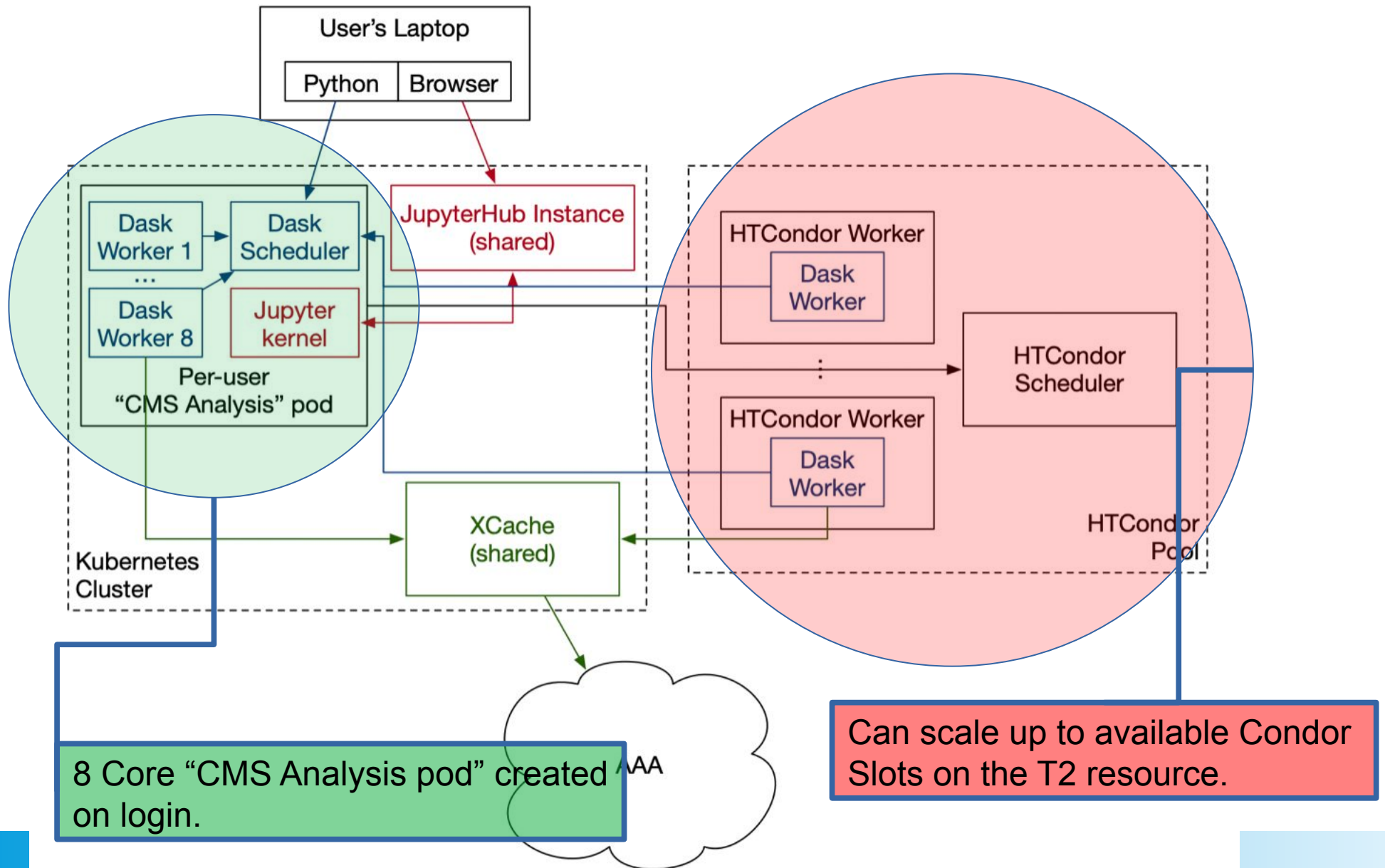
Usage Patterns Are Changing Resources Should Change, Too



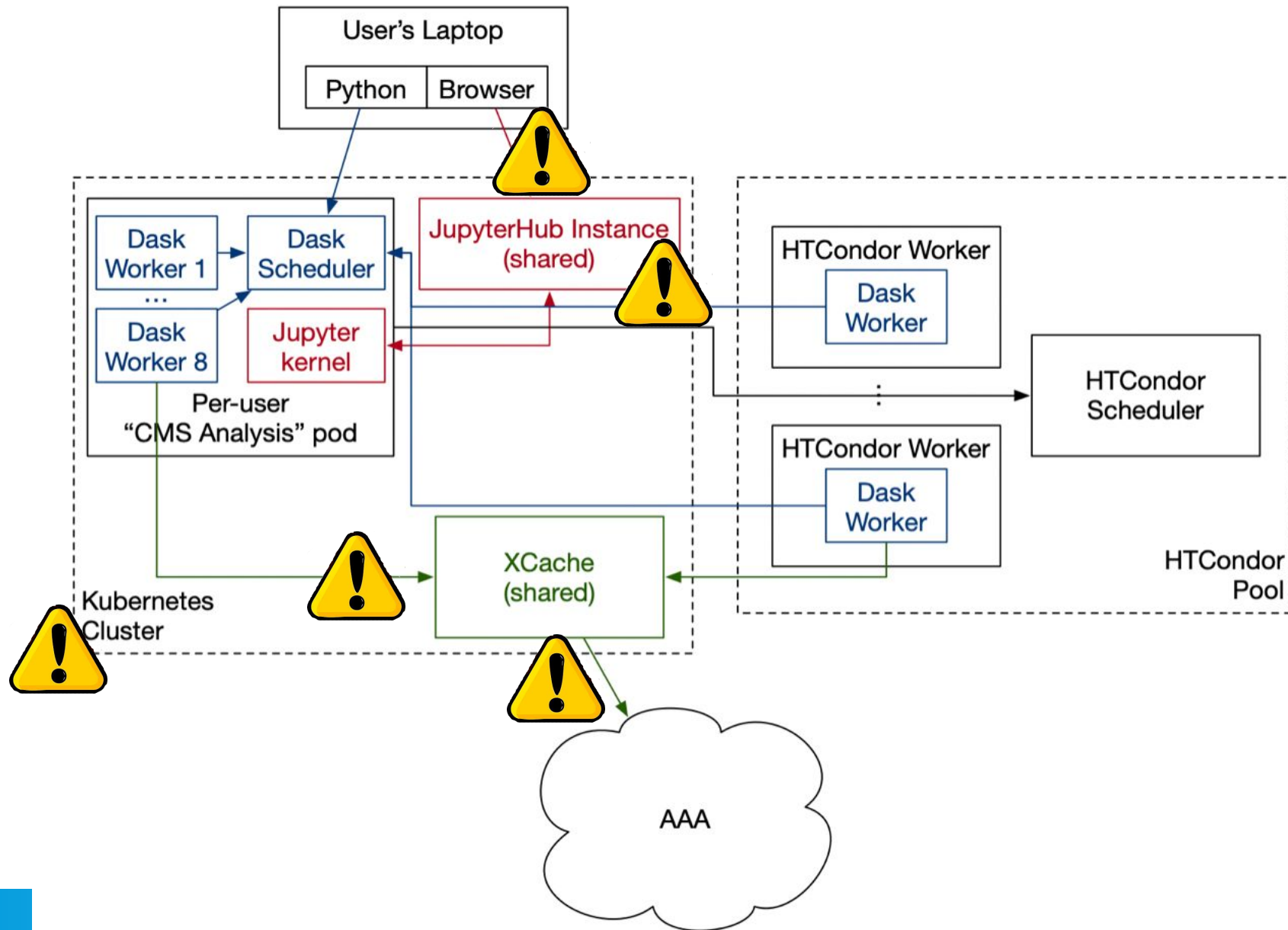
Proposed Analysis @ T2 Nebraska



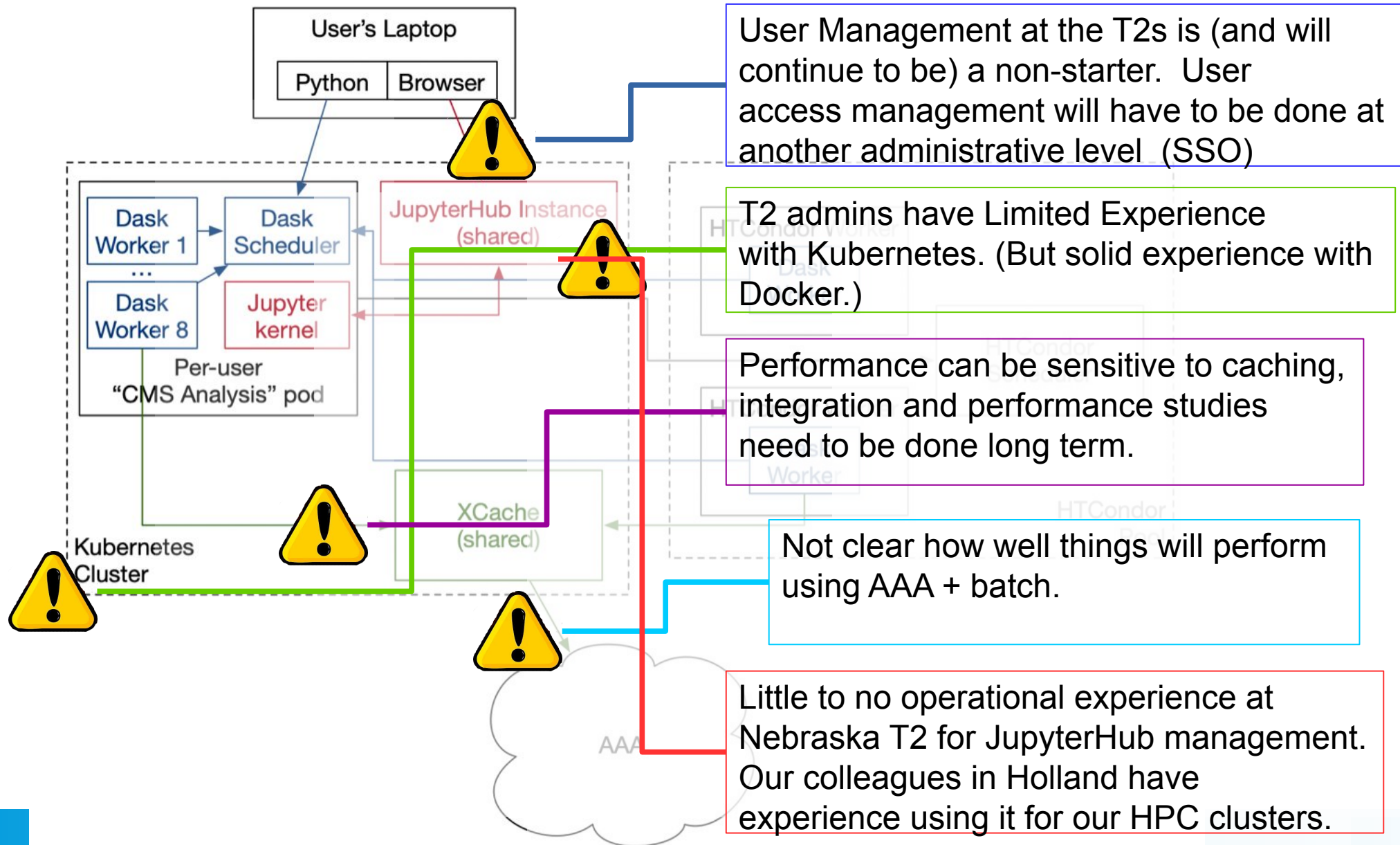
Proposed Scale Up @ T2 Nebraska



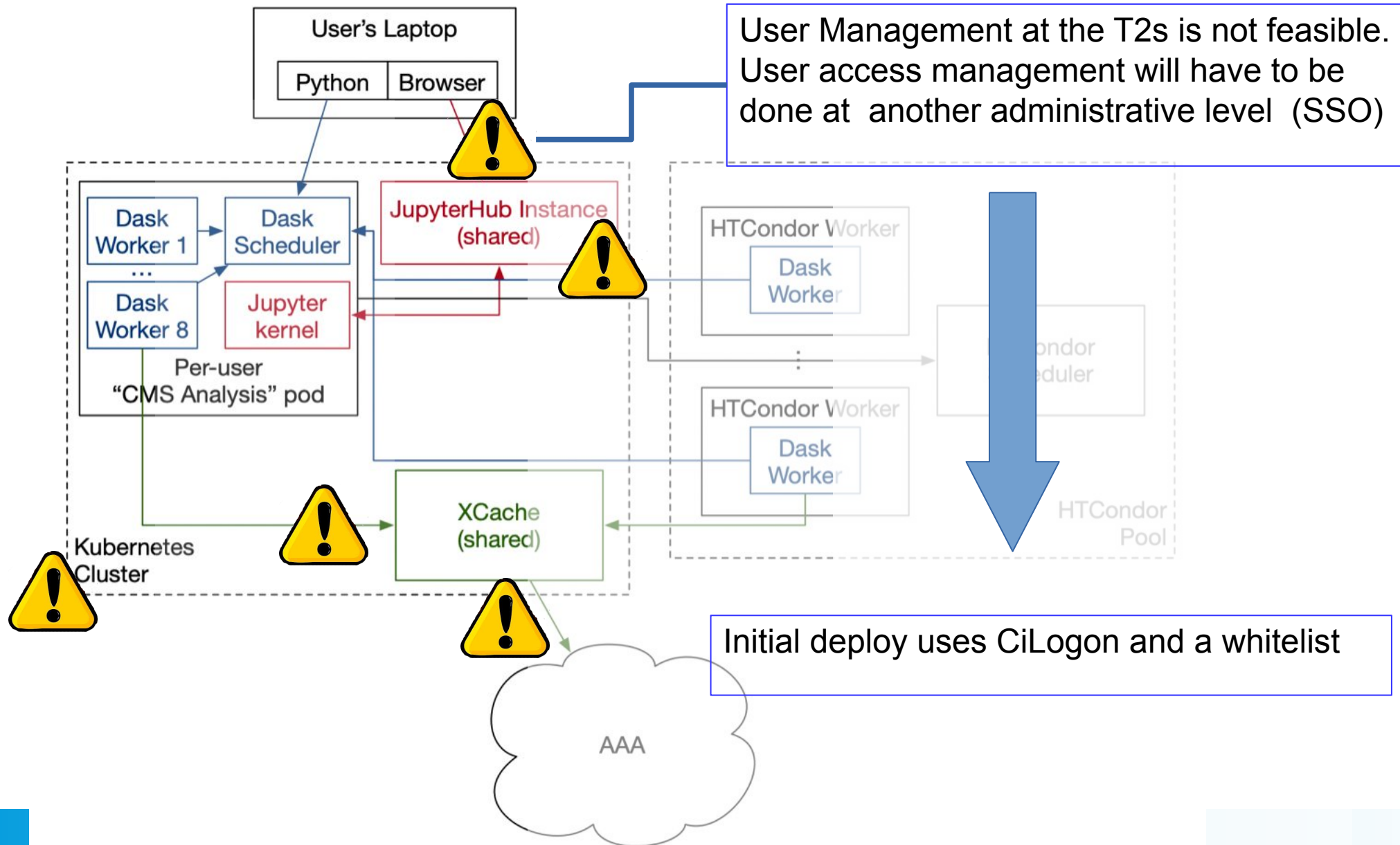
Problem/Uncertain Areas



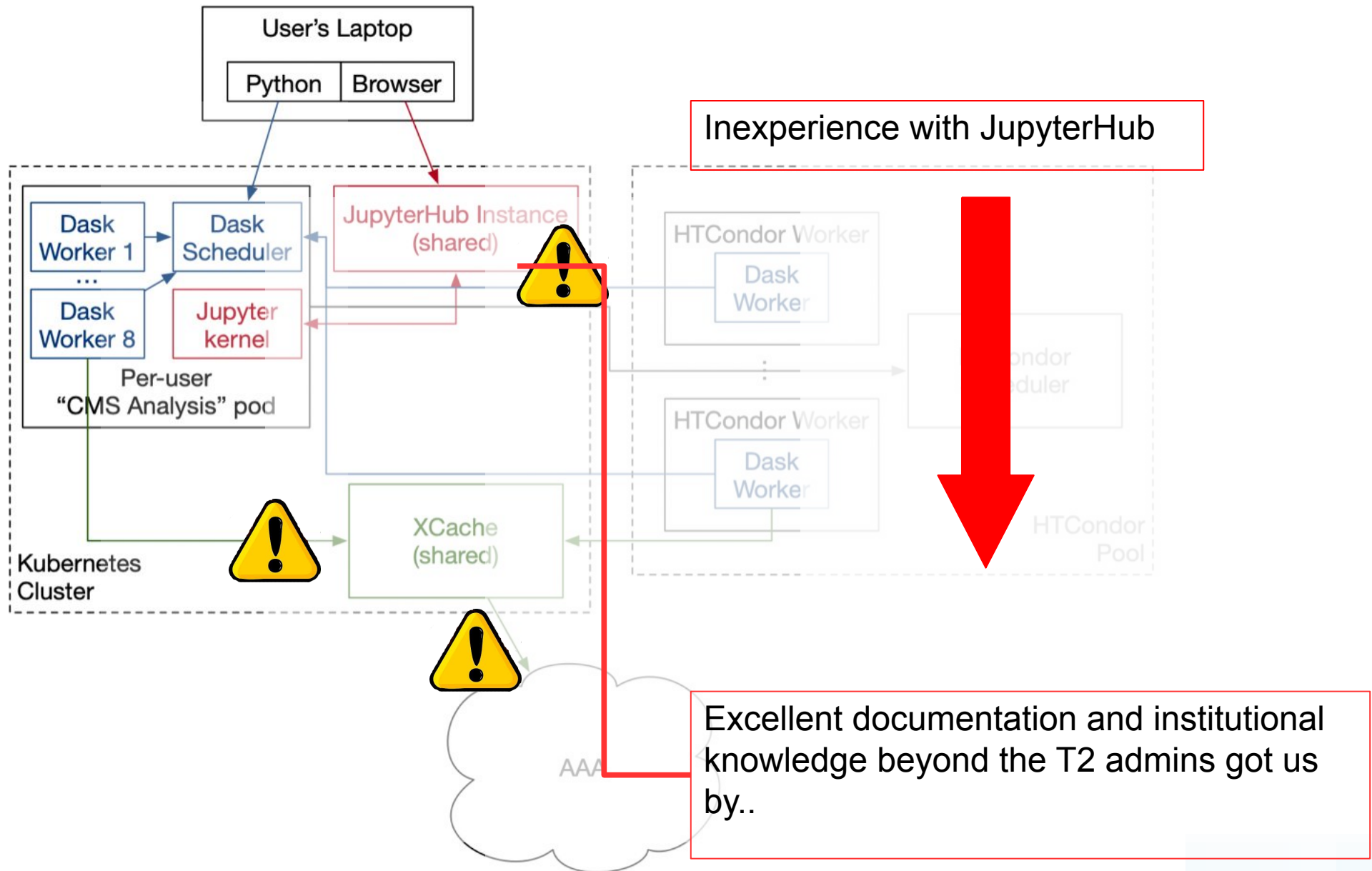
Problem/Uncertain Areas



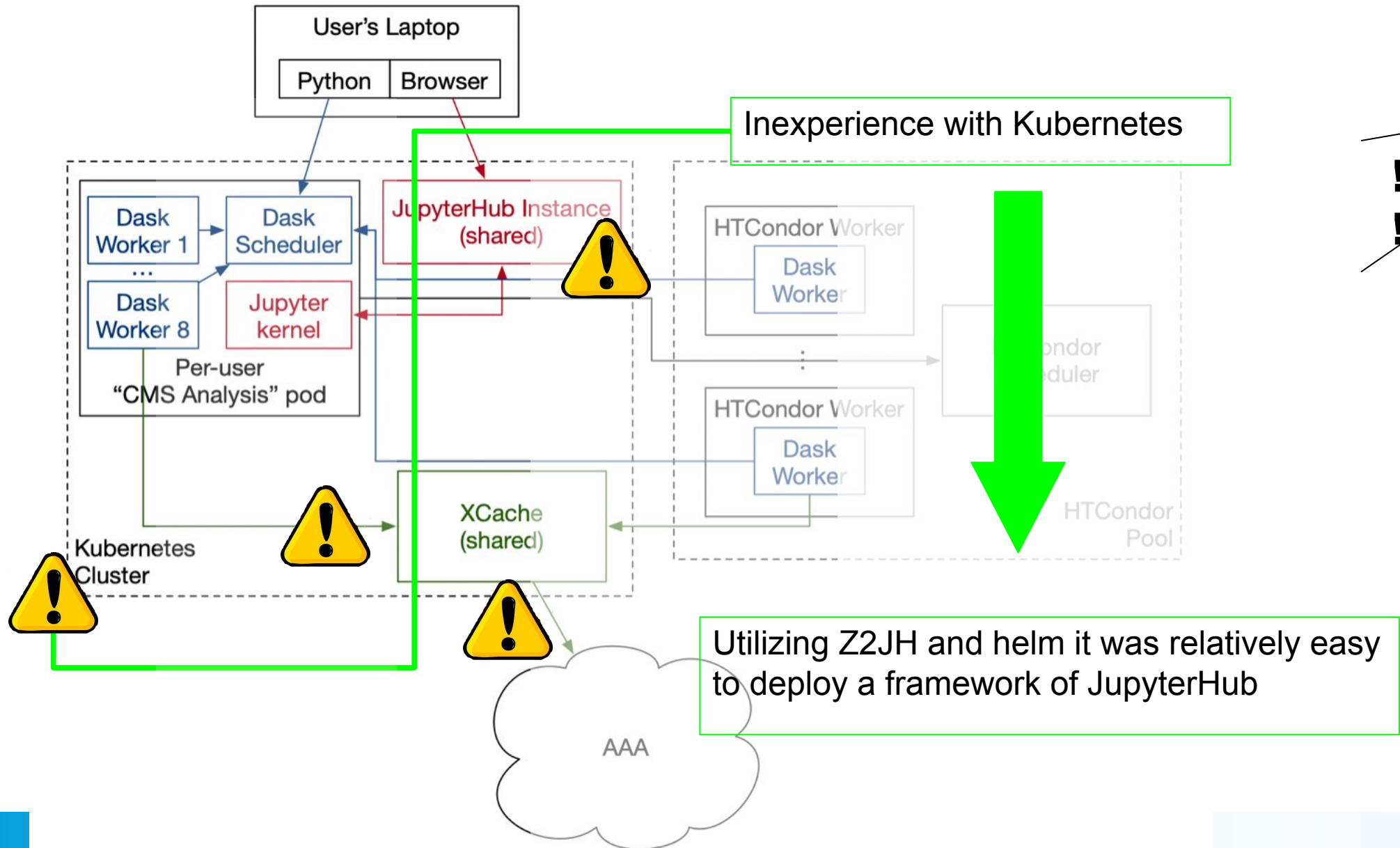
User Management: Sorted



JupyterHub Newbies: Sorted



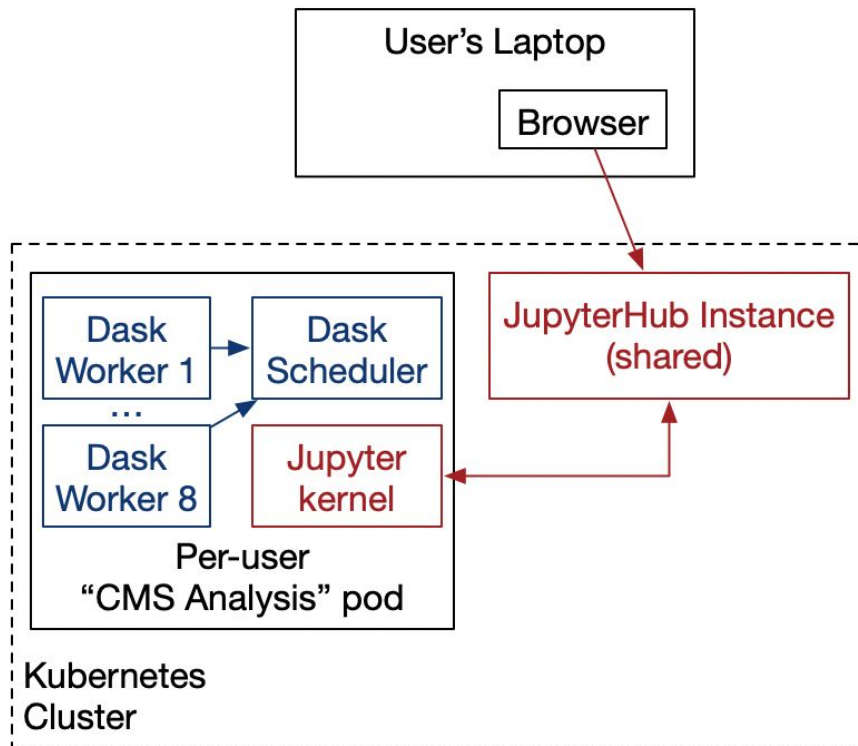
K8s Newbies: Sorted



Hardware Dedicated to Prototype (Kubernetes Cluster)

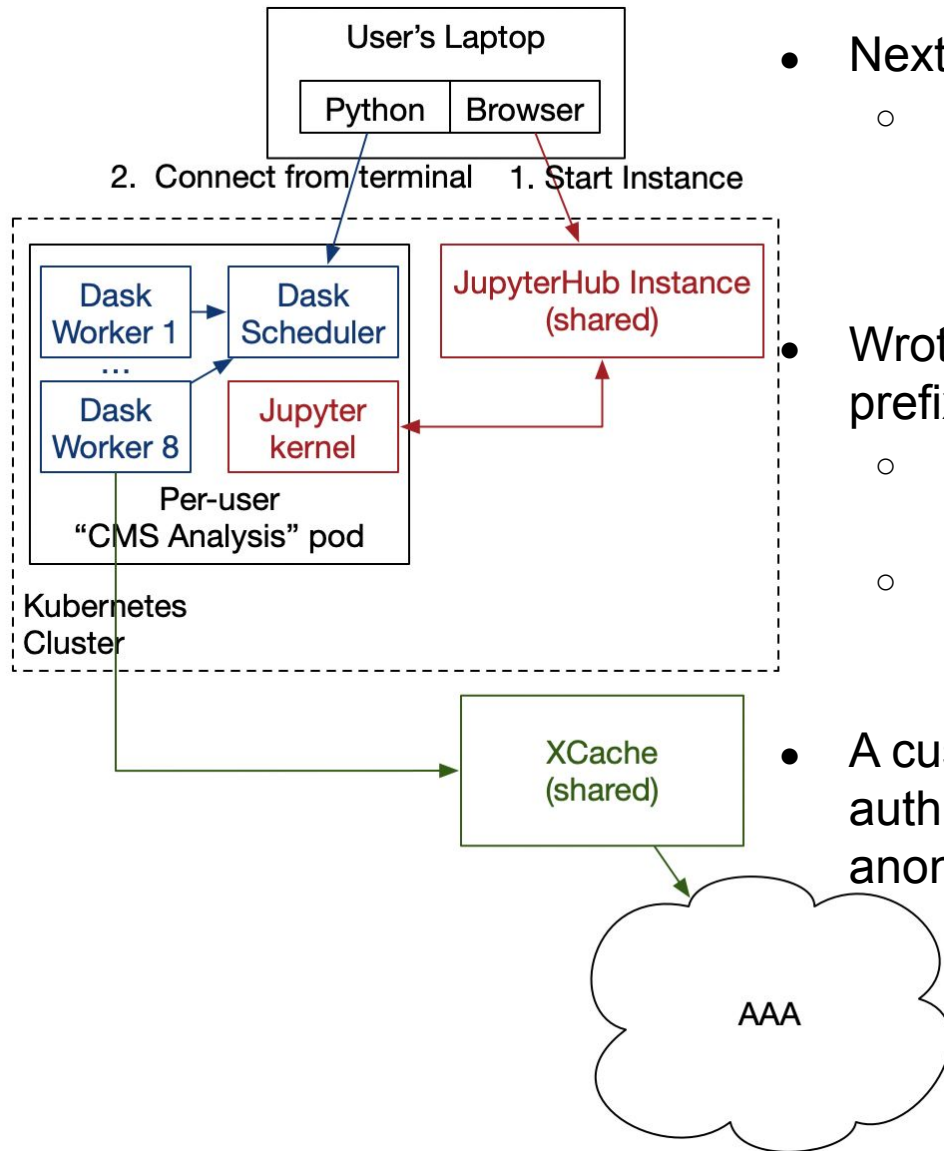
Role	Description	CPU	RAM
masters	2 x VMs living on old Dell machines	2	8GB
workers	4 x Dell R710s with disks for Rook.io	24	96GB
workers	3 x Sun X2200 (12 years old)	8	32GB
workers	5 x Sun X2200 (12 years old)	8	24GB
workers	2 x 4-in-2 Supermicro	16	64GB
workers	1 x 1U Supermicro	8	32GB

Making this a reality



- Deployed Jupyterhub on bare OS to learn Jupyterhub and CiLogon OAuth.
- Enabled token authentication in our Condor infrastructure
- Second was to setup a kubernetes cluster and use the “Zero 2 JupyterHub” (z2jh) project to put together a basic JupyterHub instance.
- We then developed a highly customized “CMS Analysis” container with all the necessary dependencies.
- JupyterHub uses the KubeSpawner to create new pods. We utilize a pod customization hook to create secrets and services so the pod:
 - Can expose the Dask scheduler to the outside world.
 - Can authenticate with services like HTCondor and XRootD.

Making this a reality

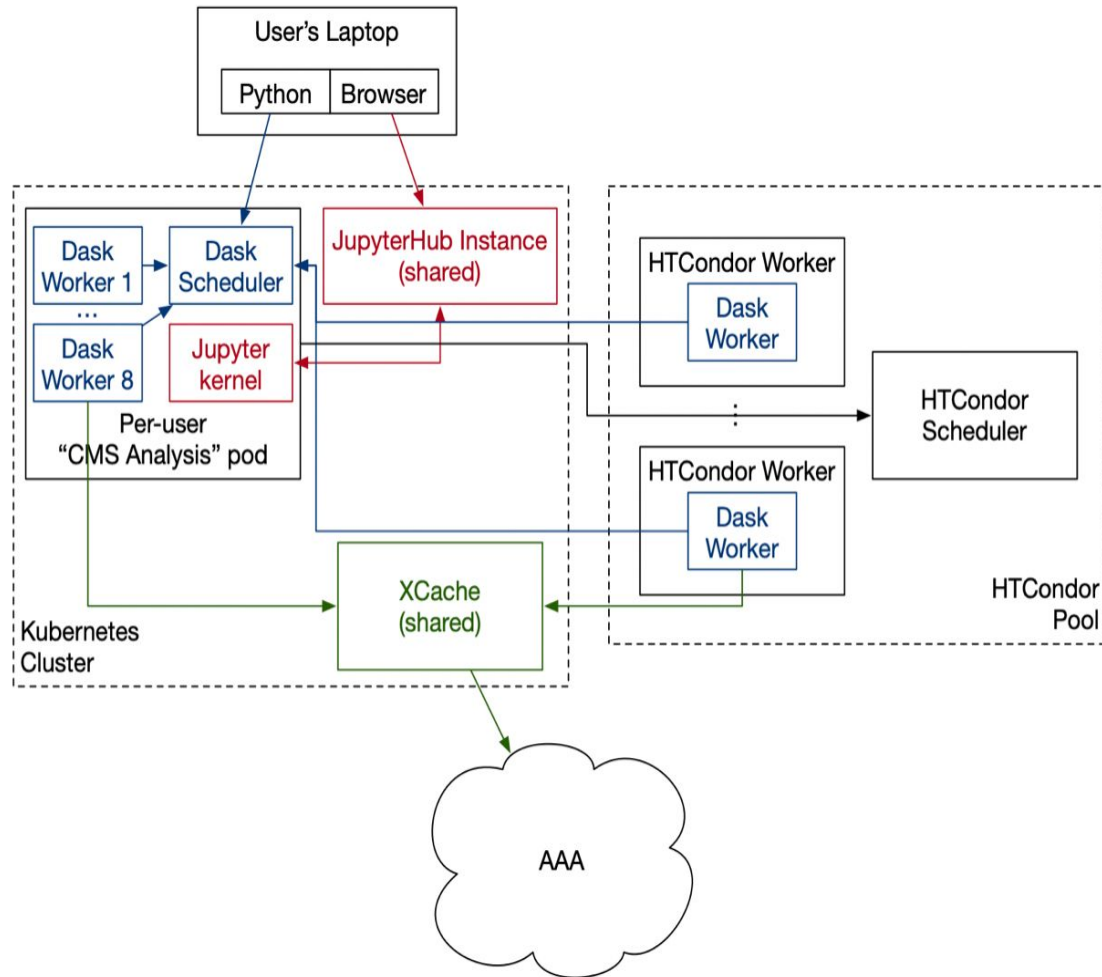


- Next, we needed to integrate XRootD!
 - Each pod's unique secret includes an auto-generated macaroon authorizing the pod to access files at the site XCache server.
- Wrote a custom XRootD client plugin so whenever the prefix `root://xcache/` is used, then:
 - The hostname is replaced with the correct one for the local site (using environment variables)
 - Token authorization is automatically used & embedded in the URL.
- A custom XCache container was made to make GSI auth optional and allow token auth after an anonymous login.

See the plugin code:

<https://github.com/bbockelm/xrdcl-authz-plugin>

Making this a reality



- Finally, we use a slightly-patched version of the HTCondorCluster integration from dask to allow auto-scaling out to the local HTCondor pool.
- Jobs run in the container on the HTCondor worker node; HTCondor exposes an incoming port to provide the necessary connectivity.
- All of this is being incorporated into a Helm chart -- many rough edges, but can eventually be portable to other sites.

CMSAF @ UNL demo Setup

- JH setup: <https://github.com/clundst/jhub> (except specific secrets)
- Docker images for Dask Scheduler and Worker:
<https://github.com/oshadura/coffea-casa>
 - <https://hub.docker.com/r/oshadura/coffea-casa>
 - <https://hub.docker.com/r/oshadura/coffea-casa-analysis>
- Docker image for JupyterHub (to get macaroons in the launch env)
<https://github.com/clundst/jhubDocker>
- Showcase: https://github.com/mat-adamec/cmsaf-jh_showcase

CMSAF @ UNL XCache setup

- <https://github.com/bbockelm/xrdcl-authz-plugin>

```
$ BEARER_TOKEN_FILE=~/projects/xrdcl-authz-plugin/xcache_token
XCACHE_HOST=red-xcache1.unl.edu
XRD_PLUGINCONFDIR=~/projects/xrdcl-authz-plugin/build/release_dir/etc/xr
ootd/client.plugins.d/ xrdcp -f
root://xcache//store/data/Run2017B/SingleElectron/MINIAOD/31Mar2018-v1/60000/9E0F8458-EA37-
E811-93F1-008CFAC919F0.root /dev/null
Looking for token in file /home/cse496/bbockelm/projects/xrdcl-authz-plugin/xcache_token
[3.65GB/3.65GB][100%][=====][934.5MB/s]
```

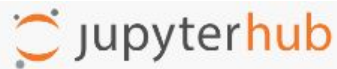
Conclusions

- Work remains.
- Some aspects of the HTCondor integration are slap-dash
- Integrating the native dask scheduler with condor jobs would make scaling trivial.
- Lots of scaling tests needed, both in terms of the jobs and the user base.
- Deployment needs cleaned up and site specific customizations need removed and made more maintainable.

Backup

JupyterHub + JupyterLab + Dask setup @ UNL

- JH is launched using Helm charts (together with users secrets)



CMS Analysis Facility @ T2_US_Nebraska

Authorized CMS Users Only!

To login into Jupyter, use your CiLogon credentials.. If you would like an account or need assistance, please email [HCC Support](#).

Useful Links

- [HCC Support Pages](#)

News

- New CMS Analysis Facility @ T2_US_Nebraska

Authorized CMS Users Only:
Sign in with CiLogon

CILogon



[Consent to Attribute Release](#)

`cmsaf-jh.unl.edu` requests access to the following information. If you do not approve this request, do not proceed.

- Your CILogon user identifier
- Your email address
- Your username and affiliation from your identity provider

Select an Identity Provider

CERN

☐ Remember

☐ By selecting "Log On"

Type to search

Centro de Estudios Interdisciplinarios y Transversales

Centro de Investigación Cooperativa en Biomateriales

Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas

Centro de Investigación y Tecnología Agroalimentaria de Aragón

Centro de Supercomputación de Galicia

Centro Informático Científico de Andalucía

Centro Nacional de Investigaciones Cardiovasculares

Centro Nacional del Hidrógeno

Centrum Wiskunde & Informatica

CEREQ - Centre d'Etudes et de Recherches sur les Qualifications

CERN

CESNET

CETEM - Centro de Tecnología Mineral

We will likely replace this with CMS Auth.

<http://oauth.web.cern.ch/>

Login

- Docker image starting JupyterLab is integrated with HTCondor Dask Scheduler communicating with T3
 - For this purpose we use Dask Labextension, which is integrated in the Docker image

Server Options

<input checked="" type="radio"/>	Coffea Base Image Oksana's build with coffea/dask/condor and cheese
<input type="radio"/>	Minimal environment To avoid too much bells and whistles: Python.
<input type="radio"/>	Datascience environment If you want the additional bells and whistles: Python, R, and Julia.
<input type="radio"/>	Spark environment The Jupyter Stacks spark image!

Start

Can Run Using Built-in Dask Clustering on Host CPUs

dask/dashboard/8f1fb6c5-3e5c-4e73-b173-f18b'

PROGRESS

CPU

PROFILE

NBYTES

MEMORY BY KEY

BANDWIDTH WORKERS

WORKERS

NPROCESSING

PROFILE SERVER

TASK STREAM

BANDWIDTH TYPES

GRAPH

CLUSTER MAP

CLUSTERS

+ NEW

LocalCluster 2

Scheduler Address: tcp://127.0.0.1:42175

Dashboard URL: http://127.0.0.1:8787/status

Number of Cores: 8

Memory: 33.73 GB

Number of Workers: 4

<> SCALE SHUTDOWN

Terminal 4

Dask Workers

adl1.ipynb

Code

git

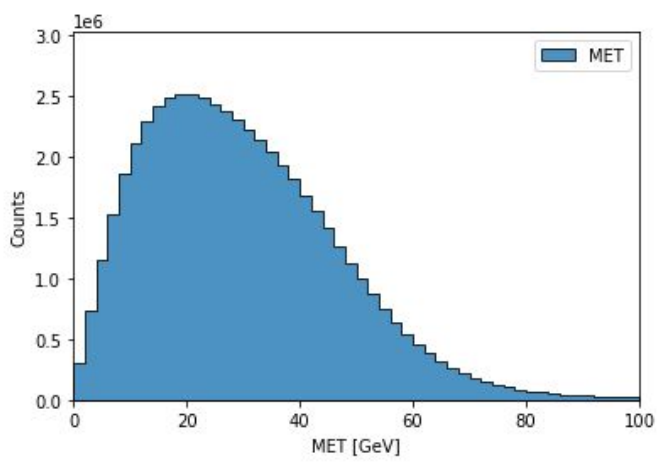
```
[47]: output = processor.run_uproot_job(fileset,
                                     treename='Events',
                                     processor_instance=Processor(),
                                     executor=processor.dask_executor,
                                     executor_args={'client': client},
                                     chunksize = 2500000)

[#####] | 100% Completed | 36.3s

[48]: hist.plot1d(output['MET'], overlay='dataset', fill_opts={'edgecolor': (0,0,0,0.3), 'alpha': 0.8})

/opt/conda/lib/python3.7/site-packages/mplhep/_deprecate.py:56: DeprecationWarning: kward "density" on "histplot" is deprecated and may be removed in future versions: "unit" mode is not useful
  return func(*args, **kwargs)

[48]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6ccb98110>
```



```
[49]: for key, value in output['cutflow'].items():
      print(key, value)

all events 53446198
number of chunks 21
```

CMSAF @ UNL secrets

- All secrets are available in the directory */etc/cmsaf-secrets* at container startup (*but doesn't exist at build time*)
- The *BEARER_TOKEN_FILE* environment variable is going to be set to */etc/cmsaf-secrets/bearer_token*, matching what's expected in the XrdCl plugin.
- */etc/cmsaf-secrets/condor_token* is a condor IDTOKEN useful for submitting to T3.
- */etc/cmsaf-secrets/ca.key* is a CA private key useful for Dask
- */etc/cmsaf-secrets/ca.pem* is a CA public key useful for Dask
- */etc/cmsaf-secrets/hostcert.pem* is a host certificate and private key useful for the Dask scheduler.
- */etc/cmsaf-secrets/usercert.pem* is a user certificate and private key useful for the Dask workers.

HTCondor integration

```
jovyan@jupyter-oksana-2eshadura-40cern-2ech:/opt/app$ condor_q
```

```
-- Schedd: t3.unl.edu : <129.93.239.166:9618?... @ 05/25/20 19:46:47
```

OWNER	BATCH_NAME	SUBMITTED	DONE	RUN	IDLE	TOTAL	JOB_IDS
cms-jovyan	ID: 1586949	5/23 19:21		_	_	_	1 1586949.0

```
Total for query: 1 jobs; 1 completed, 0 removed, 0 idle, 0 running, 0 held,  
0 suspended
```

```
Total for cms-jovyan: 1 jobs; 1 completed, 0 removed, 0 idle, 0 running, 0  
held, 0 suspended
```

```
Total for all users: 14313 jobs; 1 completed, 0 removed, 14310 idle, 2  
running, 0 held, 0 suspended
```

Demo backup

JupyterLab interface showing a terminal window with Python code for distributed computing and data processing.

The code defines a `HTCondorJob` class and a `HTCondorCluster` class, along with a `processor.ProcessorABC` class. It includes configuration for distributed computing, file paths, and logging.

```
(2): from distributed.security import Security
from coffee import hist
from coffee.analysis.objects import JaggedCandidateArray
import coffee.processor as processor

(3): from dask.distributed import Client, LocalCluster
from dask_jobqueue import HTCondorCluster
from dask_jobqueue.htcondor import HTCondorJob

(4): fileset = {
    'files': ['root://eospublic.cern.ch/eos/root-eos/benchmark/Run2012B_SingleMu.root'],
    'treename': 'Events'
}

(5): # This program plots an event-level variable (in this case, MET, but switching it is as easy as a dict-key change). It also demonstrates an easy use of the book-keeping cutflow tool, to keep track of the number of events processed.
# The processor class bundles our data analysis together while giving us some helpful tools. It also leaves looping and chunks to the framework instead of us.
class METProcessor(processor.ProcessorABC):
    def __init__(self):
        # Bins and categories for the histogram are defined here. For format, see https://coffeteam.github.io/coffee/stubs/coffee.hist.hist_tools.Hist.html && https://coffeteam.github.io/coffee/stubs/coffee.hist.hist_tools.Bin.html
        self.columns = ['MET_pt']
        dataset_axis = hist.Cat("dataset", "")
        MET_axis = hist.Bin("MET", "MET [GeV]", 50, 0, 100)
        # The accumulator keeps our data chunks together for histogramming. It also gives us cutflow, which can be used to keep track of data.
        self._accumulator = processor.dict_accumulator({
            'MET': hist.Hist("Counts", dataset_axis, MET_axis),
            'cutflow': processor.defaultdict_accumulator(int)
        })

    @property
    def accumulator(self):
        return self._accumulator

    @property
    def columns(self):
        return self._columns

    def process(self, df):
        output = self.accumulator.Identity()
        # This is where we do our actual analysis. The df has dict keys equivalent to the TTree's.
        dataset = df['dataset']
        MET = df['MET_pt']
        # We can define a new key for cutflow (in this case 'all events'). Then we can put values into it. We need += because it's per-chunk (demonstrated below)
        output['cutflow']['all events'] += MET.size
        output['cutflow']['number of chunks'] += 1
        # This fills our histogram once our data is collected. Always use .flatten() to make sure the array is reduced. The output key will be as defined in __init__ for self._accumulator; the hist key ('MET') will be defined in the bin.
        output['MET'].fill(dataset=dataset, MET=MET.flatten())
        return output

    def postprocess(self, accumulator):
        return accumulator

(6): sec_dask = Security(tls.ca_file="/etc/cmsaf-secrets/ca.pem",
                        tls.worker_cert="/etc/cmsaf-secrets/usercert.pem",
                        tls.worker_keys="/etc/cmsaf-secrets/userkey.pem",
                        tls.scheduler_cert="/etc/cmsaf-secrets/hostcert.pem",
                        tls.scheduler_keys="/etc/cmsaf-secrets/hostkey.pem",
                        require_encryption=True)

(7): HTCondorJob.submit_command = "condor_submit -spool"

(8): cluster = HTCondorCluster(cores=4,
                              memory="2GB",
                              disk="1GB",
                              log_directory="logs",
                              silence_logs="debug",
                              scheduler_options={"dashboard_address": "8786", "port": 8787, "external_address": "129.03.183.33:8787"},
                              # HTCondor submit script
                              job_extra={"universe": "docker", # => Brian's test
                                         # Generated in coffee-casalatest
                                         # "encrypt_input_files": "/etc/cmsaf-secrets/xcache_token",
                                         "docker_network_type": "host",
                                         "docker_image": "shadurac/coffee-casa-analysis:0.1.1", # or docker_image # => Brian's test
                                         "container_service_names": "dask",
                                         "dask_container_port": "8787",
                                         "should_transfer_files": "YES",
                                         "when_to_transfer_output": "ON_EXIT"
                                         })

distributed.scheduler - INFO - Clear task state
distributed.scheduler - INFO - Scheduler at: tcp://192.168.49.178:8787
distributed.scheduler - INFO - dashboard at: 8786
```

Demo backup

JupyterLab interface showing a file browser on the left, a terminal window in the center, and a histogram plot on the right.

File Browser: Shows files in the `coffea-casa-example /` directory. Files include `data`, `logs`, `ad1.py`, `ad1.py`, `ad1.py`, `coffea-casa-nanotest.py`, `coffea-casa-nanotest.py`, and `README.md`.

Terminal Window: Displays the execution of a JupyterLab session. The code executed includes:

```
[9]: SortedDict({})
[10]: cluster.scale(jobs=16)
[11]: client = Client(cluster)#, security=sec_dask)

distributed.scheduler - INFO - Event loop was unresponsive in Scheduler for 36.31s. This is often caused by long-running GIL-holding functions or moving large chunks of data. This can cause timeouts and instability.
distributed.scheduler - INFO - Starting worker compute stream, tcp://129.93.182.169:32769
distributed.core - INFO - Starting established connection
distributed.scheduler - INFO - Register worker <Worker 'tcp://129.93.182.61:32769', name: htcondor--1981987.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tcp://129.93.182.61:32769
distributed.core - INFO - Starting established connection
distributed.scheduler - INFO - Register worker <Worker 'tcp://129.93.182.12:32777', name: htcondor--1981986.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tcp://129.93.182.12:32777
distributed.core - INFO - Starting established connection
distributed.scheduler - INFO - Receive client connection: Client-3dec0ac-9ff7-11ea-83a7-56fca6b1ea3c
distributed.core - INFO - Starting established connection
/opt/conda/lib/python3.7/site-packages/distributed-2.16.0-py3.7.egg/distributed/client.py:1115: VersionMismatchWarning: Mismatched versions found

distributed
-----
| version |
-----+-----
| client  | 2.16.0+0.gf66eda86.dirty |
| scheduler | 2.16.0+0.gf66eda86.dirty |
| tcp://129.93.182.12:32777 | 2.17.0 |
| tcp://129.93.182.169:32769 | 2.17.0 |
| tcp://129.93.182.61:32769 | 2.17.0 |
-----+-----
warnings.warn(version_module.VersionMismatchWarning(msg[0] if "warning" in msg))

[12]: #cachestrategy = 'dask-worker'
exe_args = {
    'client': client,
    '#cachestrategy': cachestrategy,
    '#savemetrics': True,
    '#worker_affinity': True if cachestrategy is not None else False,
}
output = processor.run_uproot_job(fileset,
    treename = 'Events',
    processor_instance = METProcessor(),
    executor = processor.dask_executor,
    executor_args = exe_args
)

[ ] | 0% Completed | 0.4s
distributed.scheduler - INFO - Register worker <Worker 'tcp://129.93.182.61:32768', name: htcondor--1981985.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tcp://129.93.182.61:32768
distributed.core - INFO - Starting established connection
distributed.scheduler - INFO - Register worker <Worker 'tcp://129.93.182.12:32776', name: htcondor--1981984.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tcp://129.93.182.12:32776
distributed.core - INFO - Starting established connection
[ ] | 0% Completed | 0.5s
distributed.scheduler - INFO - Register worker <Worker 'tcp://129.93.182.12:32778', name: htcondor--1981990.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tcp://129.93.182.12:32778
distributed.core - INFO - Starting established connection
##### | 100% Completed | 1min 6.1s

[13]: # Generates a 1D histogram from the data output to the 'MET' key. fill_opts are optional, to fill the graph (default is a line).
hist.plot1d(output['MET'], overlay='dataset', fill_opts={'edgecolor': (0,0,0,0.3), 'alpha': 0.8})

/opt/conda/lib/python3.7/site-packages/matplotlib/mpltools/deprecate.py:56: DeprecationWarning: kwarg "densitymode" in function "histplot" is deprecated and may be removed in future versions: "unit" mode is not useful
    return func(*args, **kwargs)

[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f06d6d2c9d0>

[14]: # Easy way to print all outflow dict values. Can just do print(output['outflow']['KEY_NAME']) for one.
for key, value in output['outflow'].items():
    print(key, value)

all events 53446198
number of chunks 534
```

Histogram Plot: A 1D histogram showing the distribution of MET (GeV) values. The x-axis is labeled 'MET [GeV]' and ranges from 0 to 100. The y-axis is labeled 'Counts' and ranges from 0.0 to 3.0. The plot shows a distribution of 'jets' (blue bars) with a peak around 20-30 GeV.

Demo backup

```
jovyan@jupyter-oksana-2eshadura-40cern-2ech:/opt/app$ condor_q
```

```
-- Schedd: t3.unl.edu : <129.93.239.166:9618?... @ 05/27/20 08:58:07
```

OWNER	BATCH_NAME	SUBMITTED	DONE	RUN	IDLE	TOTAL	JOB_IDS
cms-jovyan ID: 1981982	5/27 08:22	—	—	—	—	1	1981982.0
cms-jovyan ID: 1981983	5/27 08:22	—	—	—	—	1	1981983.0
cms-jovyan ID: 1981984	5/27 08:39	—	1	—	—	1	1981984.0
cms-jovyan ID: 1981985	5/27 08:40	—	1	—	—	1	1981985.0
cms-jovyan ID: 1981986	5/27 08:51	—	1	—	—	1	1981986.0
cms-jovyan ID: 1981987	5/27 08:51	—	1	—	—	1	1981987.0
cms-jovyan ID: 1981988	5/27 08:51	—	1	—	—	1	1981988.0
cms-jovyan ID: 1981990	5/27 08:51	—	1	—	—	1	1981990.0
cms-jovyan ID: 1981991	5/27 08:51	—	—	—	—	1	1981991.0
cms-jovyan ID: 1981992	5/27 08:51	—	1	—	—	1	1981992.0
cms-jovyan ID: 1981993	5/27 08:51	—	1	—	—	1	1981993.0
cms-jovyan ID: 1981994	5/27 08:51	—	1	—	—	1	1981994.0
cms-jovyan ID: 1981995	5/27 08:51	—	1	—	—	1	1981995.0
cms-jovyan ID: 1981996	5/27 08:51	—	1	—	—	1	1981996.0
cms-jovyan ID: 1981997	5/27 08:51	—	1	—	—	1	1981997.0
cms-jovyan ID: 1981998	5/27 08:51	—	1	—	—	1	1981998.0
cms-jovyan ID: 1981999	5/27 08:51	—	1	—	—	1	1981999.0
cms-jovyan ID: 1982000	5/27 08:51	—	1	—	—	1	1982000.0
cms-jovyan ID: 1982001	5/27 08:51	—	1	—	—	1	1982001.0
cms-jovyan ID: 1982002	5/27 08:51	—	—	—	—	1	1982002.0
cms-jovyan ID: 1982003	5/27 08:51	—	1	—	—	1	1982003.0
cms-jovyan ID: 1982004	5/27 08:51	—	1	—	—	1	1982004.0
cms-jovyan ID: 1982005	5/27 08:51	—	1	—	—	1	1982005.0

Total for query: 23 jobs; 4 completed, 0 removed, 0 idle, 19 running, 0 held, 0 suspended

Total for cms-jovyan: 23 jobs; 4 completed, 0 removed, 0 idle, 19 running, 0 held, 0 suspended

Total for all users: 418 jobs; 4 completed, 0 removed, 1 idle, 411 running, 2 held, 0 suspended