Scaling Up COFFEA at a USCMS Tier2

Lindsey Gray, Matteo Cremonesi, Bo Jayatilaka, Oliver Gutsche, Nick Smith, Allison Hall, Kevin Pedro, Maria Acosta (FNAL); Andrew Melo (Vanderbilt); Stefano Belforte (INFN); Jim Pivarski (Princeton); and others
COFFEA team

Jim Pivarski (Princeton); Ben Galewsky (NCSA); Mark Neubauer (UIUC)
IRIS-HEP

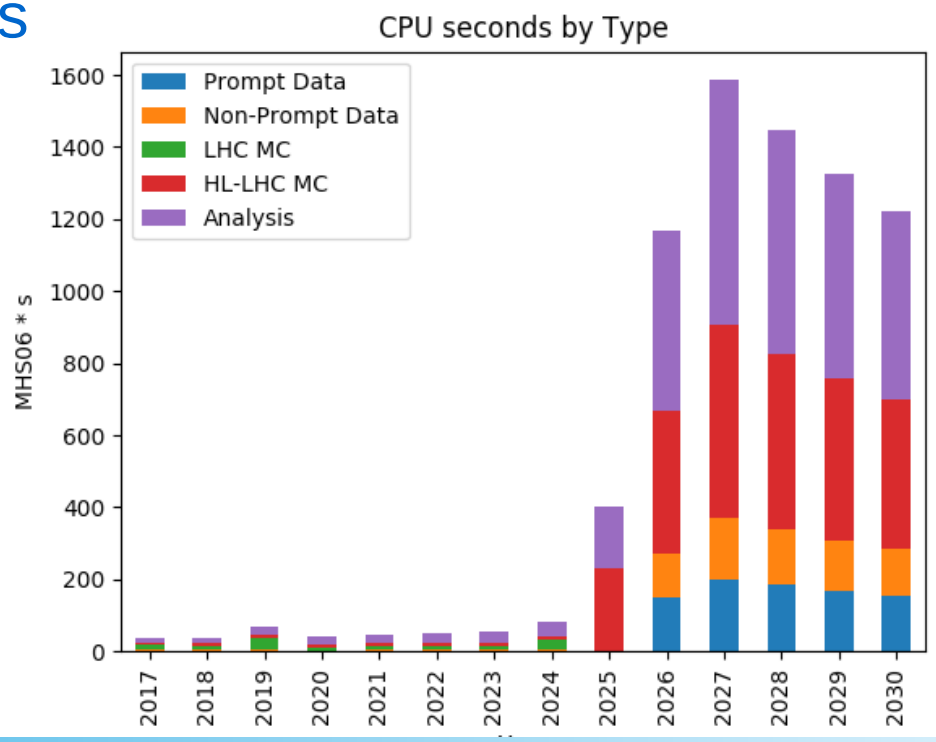
Ken Bloom, Carl Lundstedt, Mat Adamec
University of Nebraska, HTCondor/SLRUM Testing

Norbert Neumeister, Stefan Piperov, Dmitry Kondratyev
Purdue University, SLURM Testing

Brian Bockelman
Morgridge Institute

Motivations

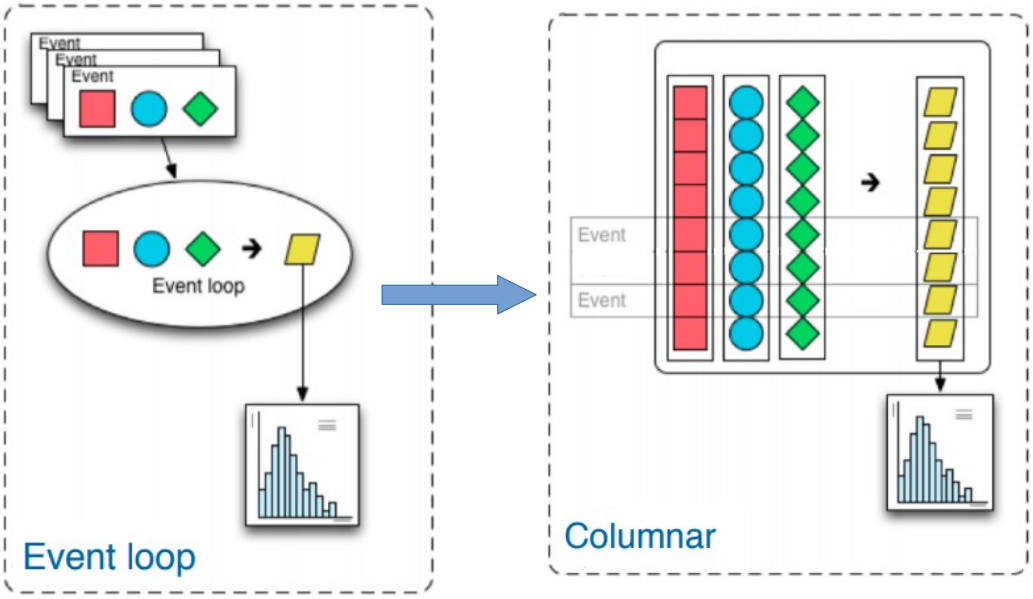
- The Present Challenge
  - Analysze all LHC Run 2 data: O(10 billion events)
  - Investigate data quality issues with fast time-to-insight
  - Optimize complex (e.g. deep learning algorithms)
- Multiply by O(1000) data analysts
- These challenges magnified 20x in HL-LHC



Nick Smith, CHEP 2019

COFFEA - Columnar Object Framework For Effective Analysis

- Leveraging “big data” and other data analysis tools from python to provide an array-based syntax for manipulating HEP event data.
- Stark contrast to common, 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 techique is not grid friendly, it's meant to be quasi-interactive



coffea – performance (Solution Requirements)

- Solutions must be:
  - Easy to use
  - Scalable
  - Fast
  - Utilize currently deployed hardware/middleware (at least in near-term)
  - Minimally intrusive for site administrators
- Work is being done to leverage SPARK
- DASK appears to be more inline with “Easy”
- Nebraska and Purdue investigated using DASK on their T2s

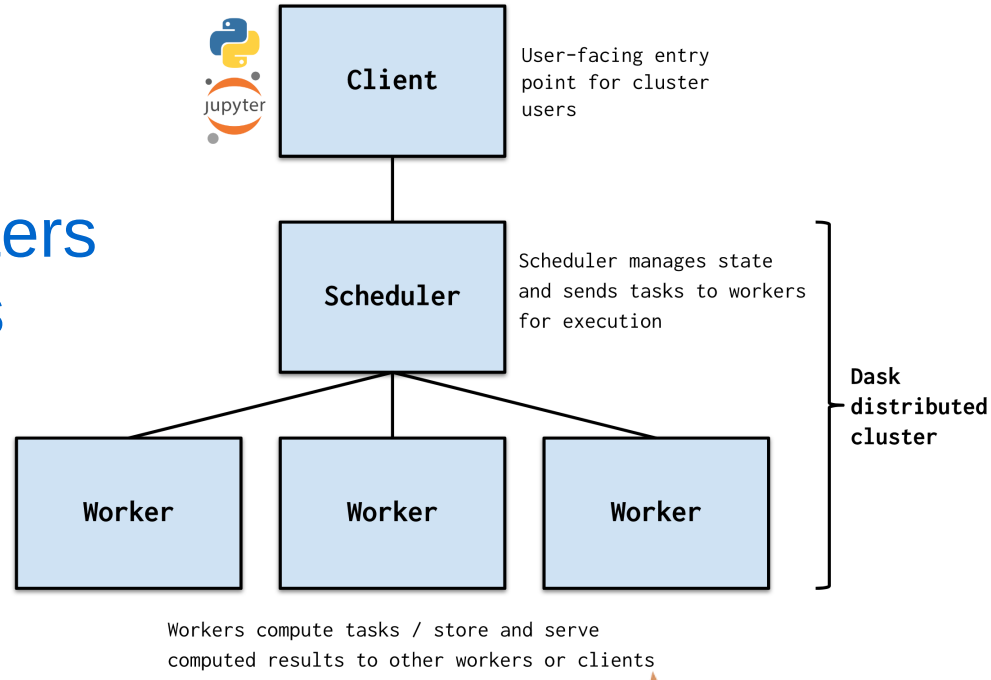
coffea – performance (Solution Requirements)

- Solutions must be:
  - Easy to use
  - Scalable
  - Fast
  - Utilize currently deployed hardware/middleware (at least in near-term)
  - Minimally intrusive for site administrators
- Work is being done to leverage SPARK
- DASK appears to be more inline with “Easy”
- Nebraska and Purdue investigated using DASK on their T2s

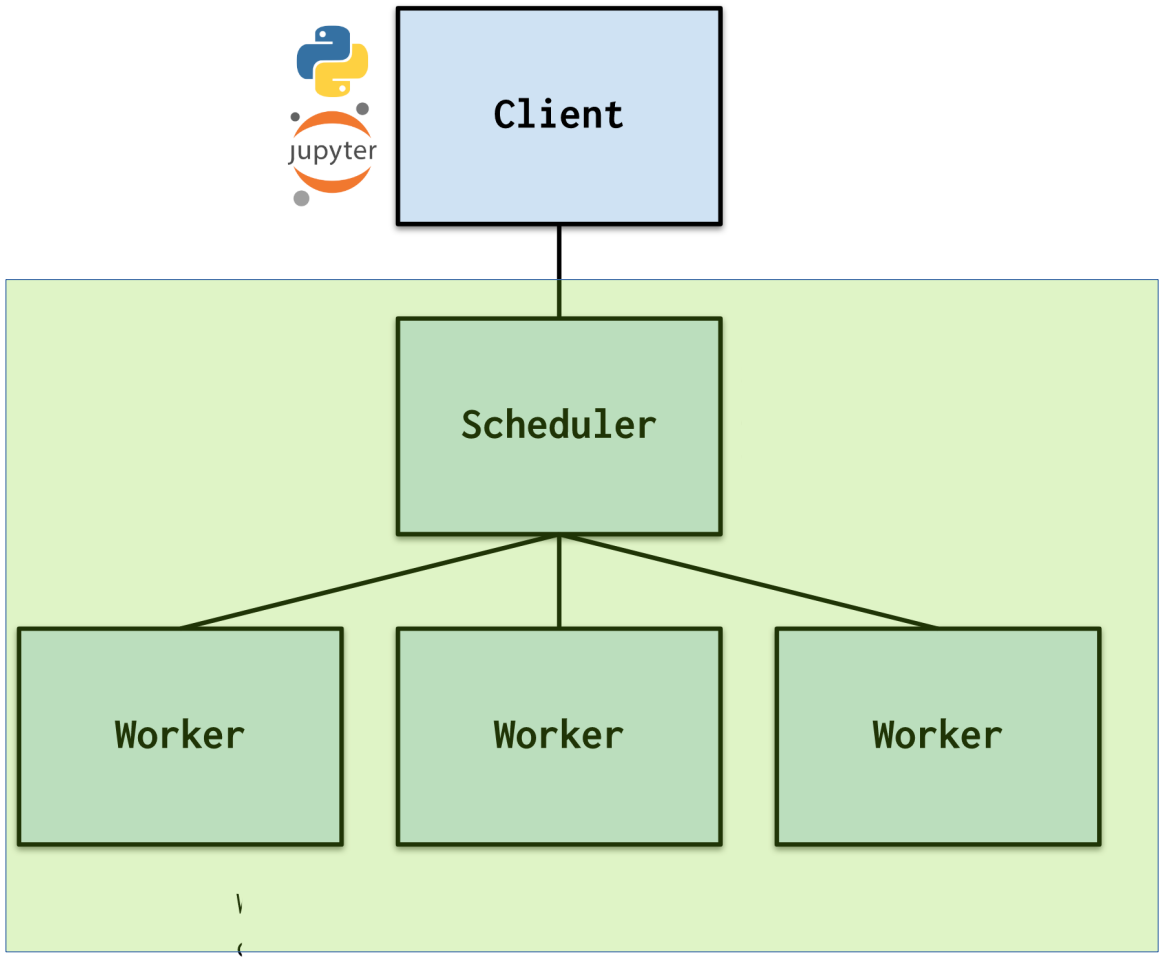
Usage Patterns Are Changing Resources Should Change, Too

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/HTC clusters running a variety of schedulers including SLURM & HTCondor via “dask-jobqueue”.



DASK @ 10,000 feet



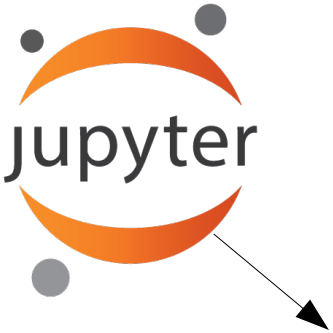
Numerous ways to instantiate the DASK cluster. Scheduler and workers can be run on dedicated hardware, submitted into a job queue OR provisioned at runtime via orchestration.

We investigated several ways to bring a DASK cluster up in our various batch systems.

*Diagram by DASK collaboration

Starting DASK @ Purdue

- Step 0) Set up environment*
  - Setup dask-jobqueue in a python env
- Step 1) Start a scheduler*
  - Scheduler started on a random workernode to avoid resource contention
  - Define Dask worker node size/capability
- Step 2) Scale the cluster*
  - Dask workernodes are not confined to the same node as the scheduler
- Step 3) Create a client*
  - Client handles data/task flow automatically
  - Analysis can access data via AAA/Xrootd
- Step 4) Send some tasks to the workers*
- Step 5) Terminate and deconstruct cluster



*Detailed commands in additional slides

SLURM Testing at Purdue

		one 2017B file		SingleMuon 2017B		Full SingleMuon 2017	
		144k events		136M events		769M events	
	processes	**Time (m:s)**	**Evts/s**	**Time (m:s)**	**Evts/s**	**Time (m:s)**	**Evts/s**
Single node	1	0:21	7k	201:40	11k	too long	
	48	0:13	11k	5:28	415k	34:57	367k
SLURM Cluster	48	0:08	17k	4:13	537k	22:27	570k
	300	0:11	13k	0:46	2.9M	6:46	1.9M
	500	0:08	17k	0:35	3.8M	3:51	3.3M
	500 (setup w/ DNN evaluation)	*0:25*	*6k*	*1:14*	*1.8M*	*9:23*	*1.4M*

*For full analysis detail, see extra slides.

Deploying Dask at Nebraska

- User environment setup is no different than Purdue
- DASK Scheduler was started on the T3 login node (warning)
- Workers started in a variety of ways:
  - By admin on dedicated nodes
  - By user via Condor jobs submitted at the CLI
  - By Jupyter using HTCondorCluster module*

*required a shared filesystem

Scaling*

Dask Workers started in Docker by root user, connect to static,dedicated scheduler

# Events	Nodes	Cores	Time (m:s)	Events per second
3,495,799	1	64 (32 proc, 2 thread)	1:16	45,637
30,268,764	1	64	7:52	64,088
90,988,105	1	64	23:11	65,383
3,495,799	4	256	0:33	105,613
30,268,764	4	256	2:37	192,182
90,988,105	4	256	8:50	171,675

Dask workers started on dedicated nodes

*Take these results as mostly anecdotal

https://github.com/mat-adamec/tHq_analysis

Scaling* (via HTCondor)

Dask Workers started in Docker by HTCondor submitted by user, workers still connect to static,dedicated scheduler

# Events	Nodes	Cores	Time (m:s)	Events per second
3,495,799	1	64 (32 proc, 2 thread)	1:00	57,781
30,268,764	1	64	11:29	43,886
90,988,105	1	64	23:13	65,275
3,495,799	4	256	0:22	153,999
30,268,764	4	256	5:15	96,030
90,988,105	4	256	12:59	116,726

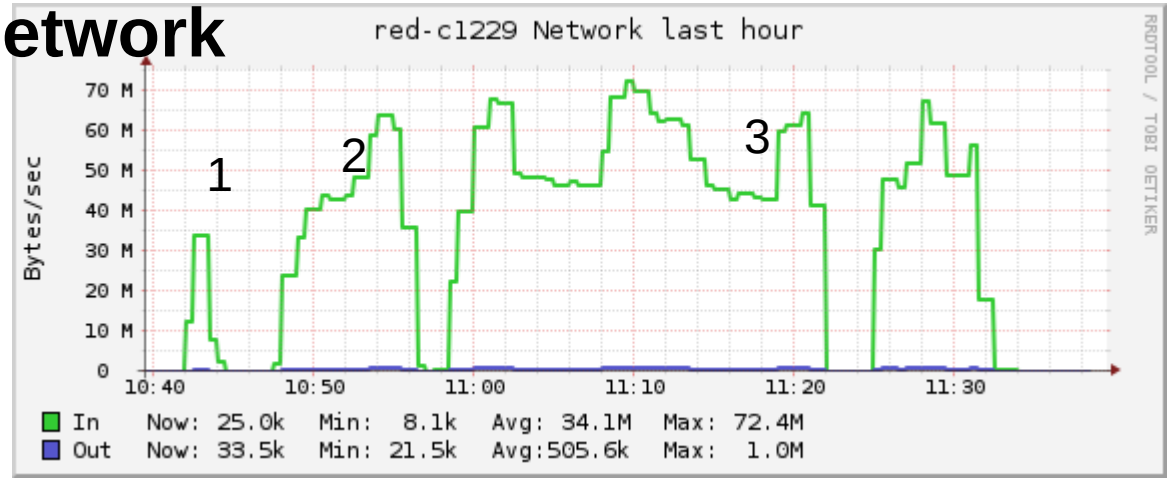
Dask workers started within Docker via Condor by user

*Take these results as mostly anecdotal

https://github.com/mat-adamec/tHq_analysis

Single Node Run Networking and Load

Network

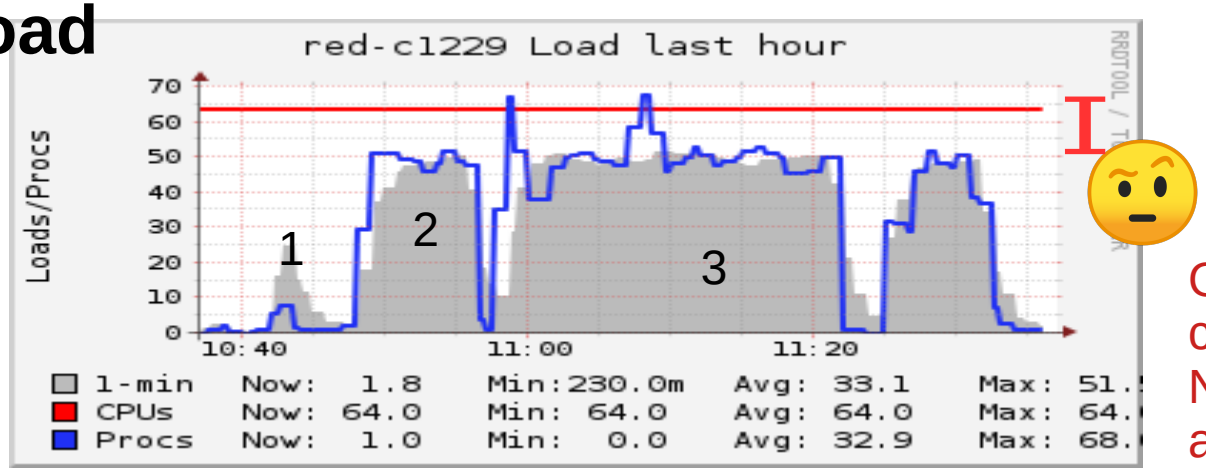


1 – 3.4 million events, 1:16

2 – 30 million events, 7:52

3 – 90.8 million events, 23:11

Load



Gap between load and cpu count I/O blocking? It's not Networking...a discussion for a different day.

Proposed Scale-up @ T2/T3 Nebraska

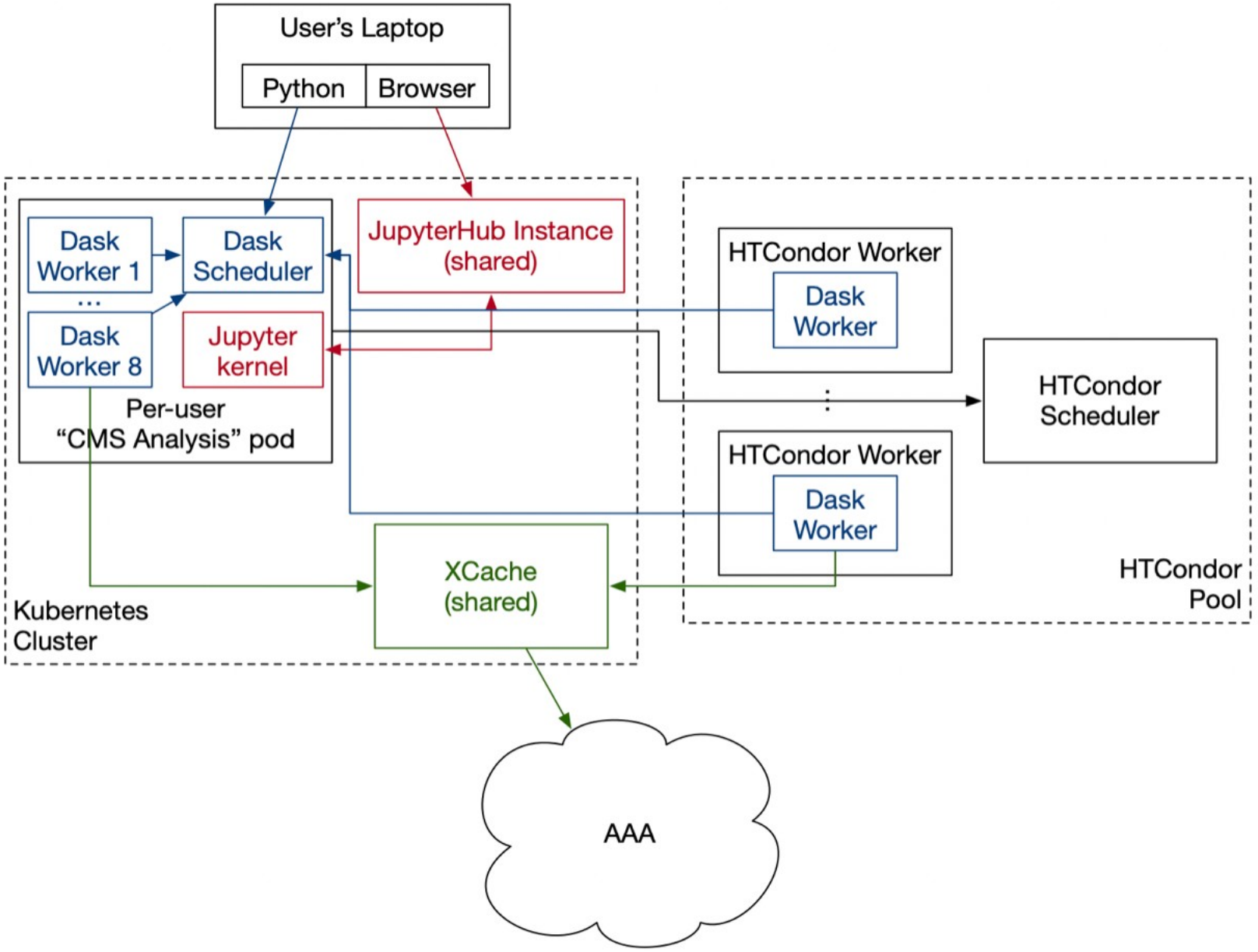


*Diagram by Brian Bockelman

Proposed Scale-up @ T2/T3 Nebraska

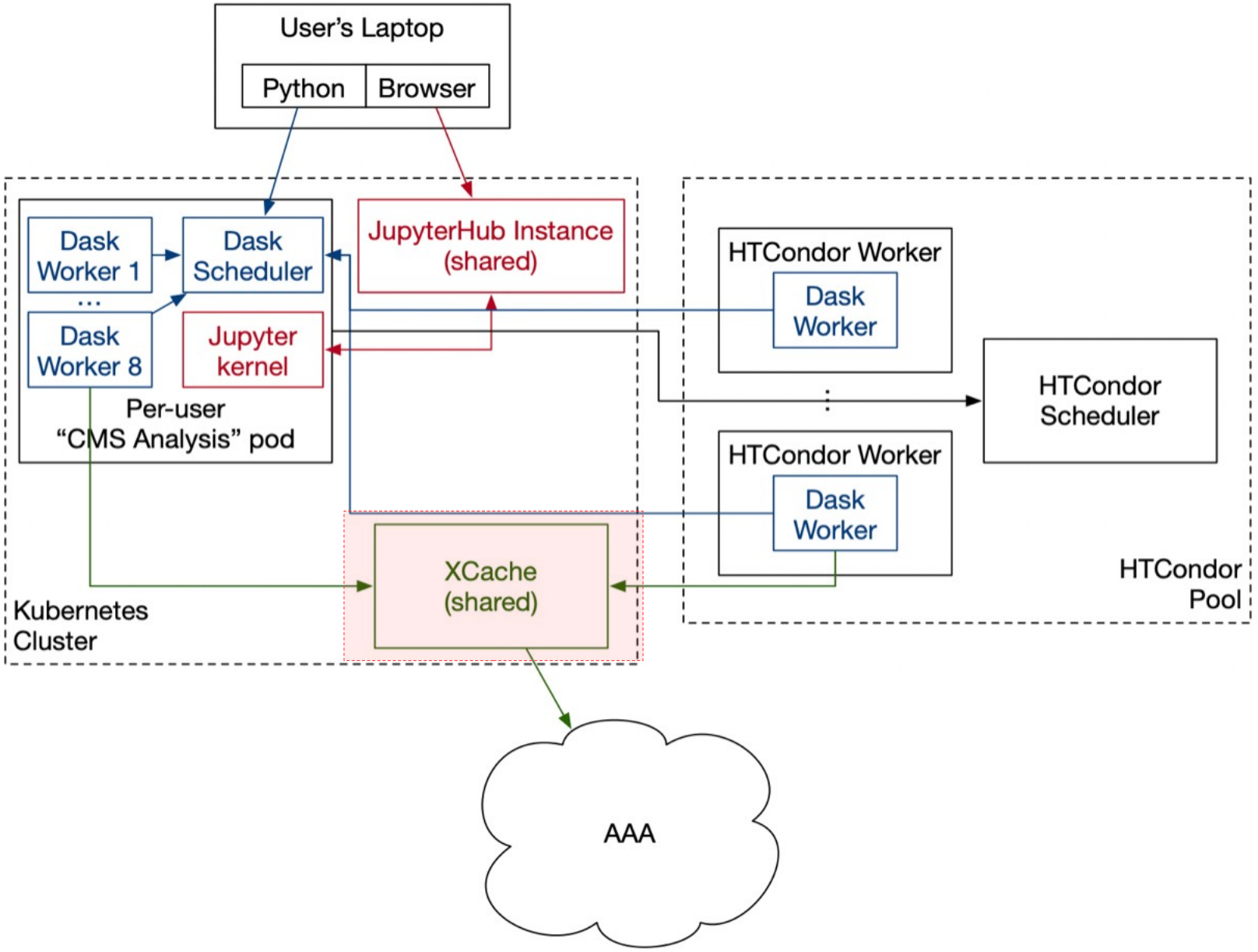


*Diagram by Brian Bockelman

Proposed Scale Up @ T2 Nebraska

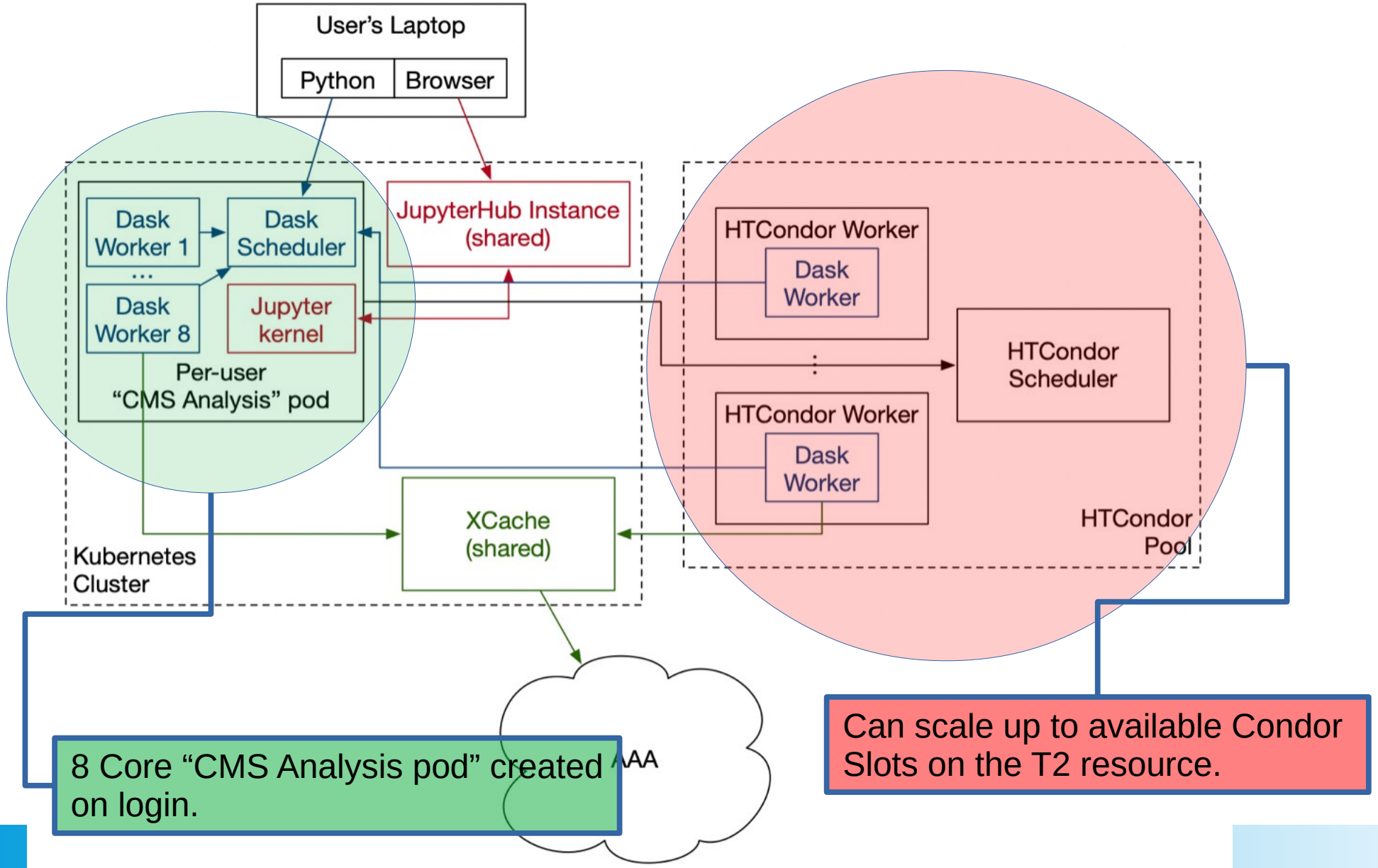
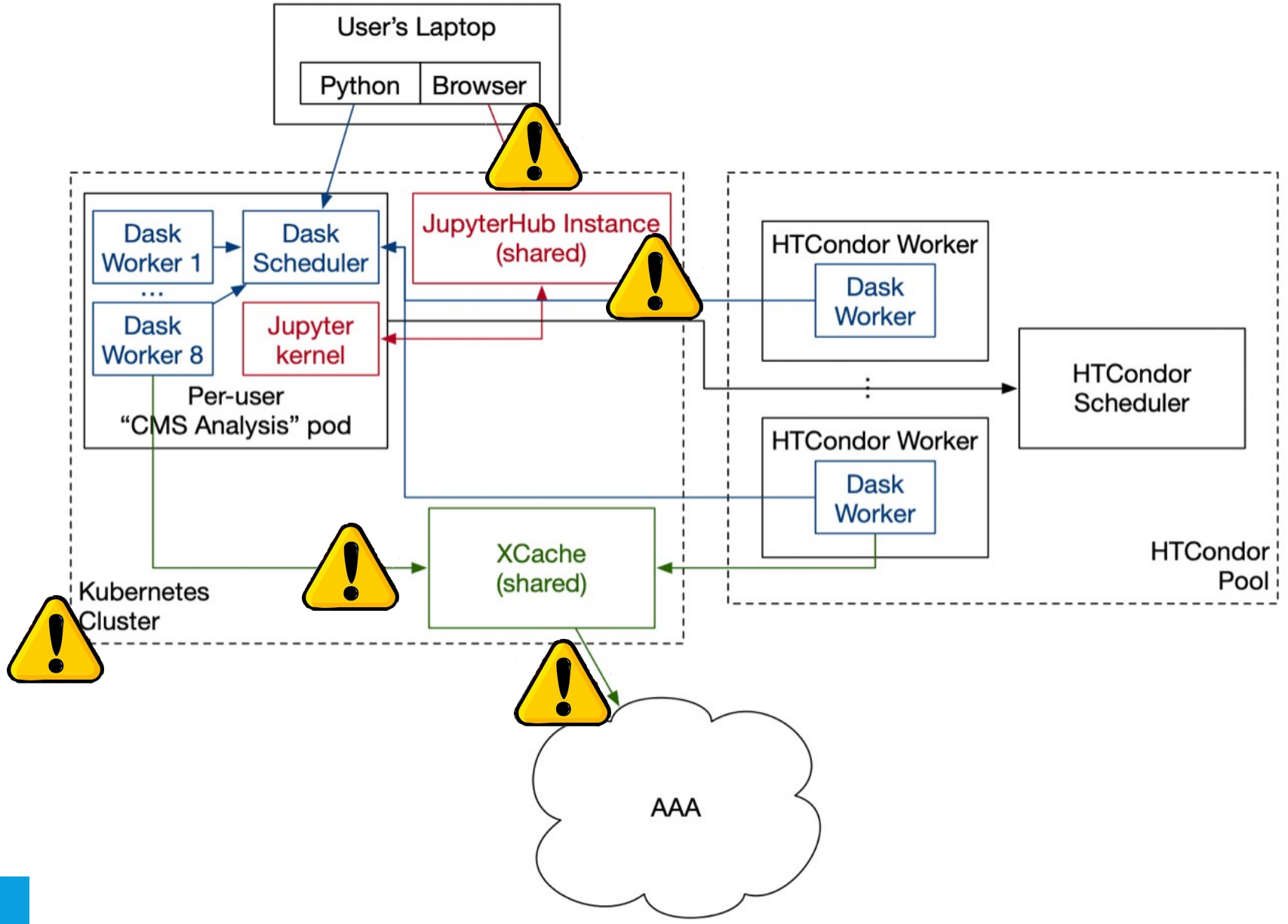
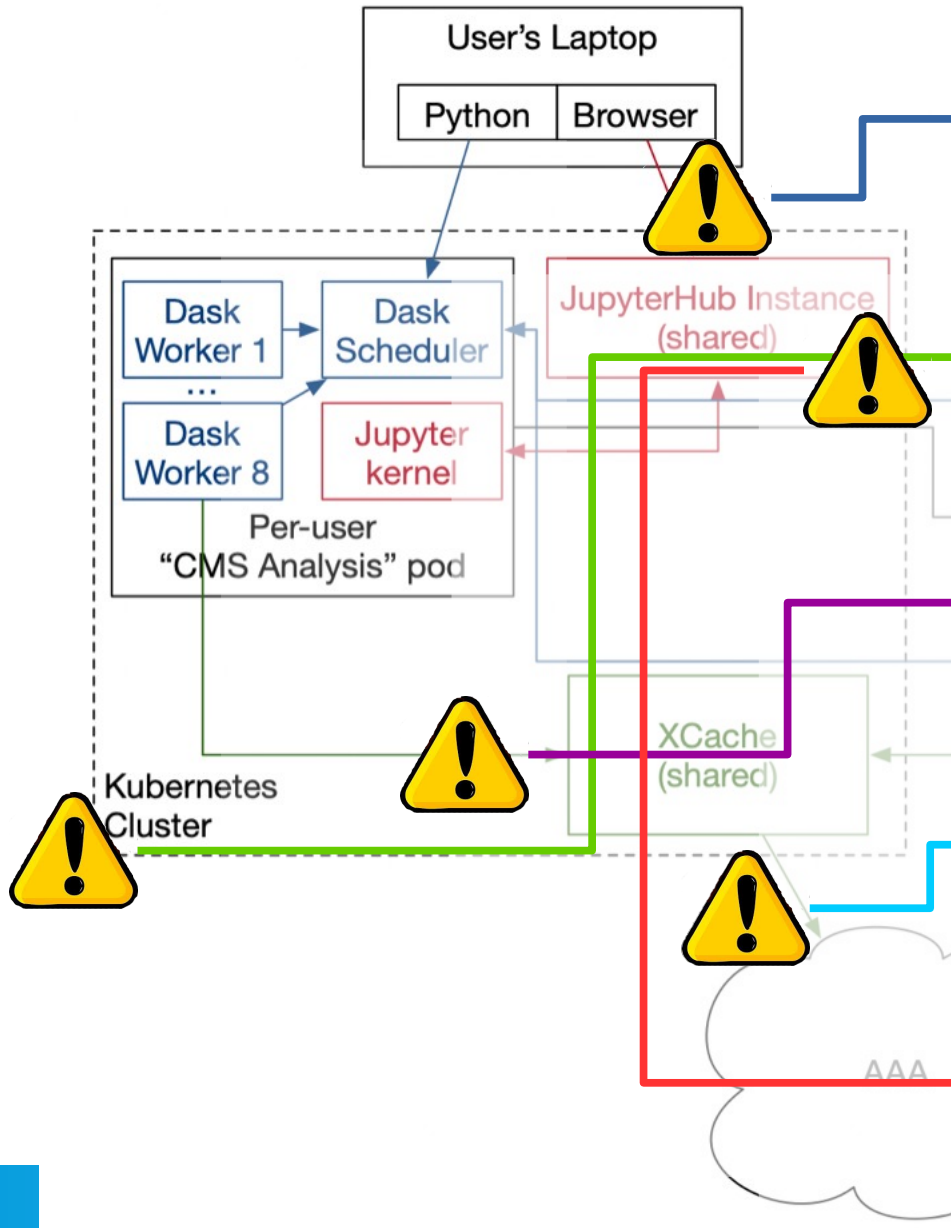


*Diagram by Brian Bockelman

Problem/Uncertain Areas



Problem/Uncertain Areas



User Management at the T2s is touchy and may not scale. User access management will have to be done at another administrative level (SSO/x509/??)

UNL T2 admins have Limited Experience with Kubernetes. (But solid experience with Docker.)

Performance can be sensitive to caching, integration and performance studies need to be done long term. Current tests did not leverage XCache @ Nebraska.

No tests at Nebraska using AAA (yet), but the COFFEA team have done zeroth-order AAA tests and are not worried

Little to no operational experience at Nebraska T2 for JupyterHub management. Our colleagues in Holland have experience using it for our HPC clusters.

Summary & Conclusions

- Investigate COFFEA + DASK on two different and current T2 sites (Purdue & Nebraska)
- DASK is easy to use, easy to hack and stands as an easy way to allow new analysis techniques to access T2 resources
- DASK could be used in an HPC environment **NOW**, how do we move to include all USCMS users and manage the workflows in a meaningful manner?
- Nebraska is going to investigate creating a Condor/Kubernetes based analysis facility as outlined
- Purdue will investigate the creation of an analysis facility as well

Questions

Dataset Info Nebraska(extra)

Small tHQ

```
all events 3495799
all jets 28404552
all muons 3542845
electrons 4490586
loose muons 1487022
fakeable muons 1264087
tight muons 1045695
loose electrons 1389332
fakeable electrons 837772
tight electrons 330102
Tight Dilepton-Channel e 7260
Tight Dilepton-Channel mu 22966
Tight Trilepton-Channel e 1021
Tight Trilepton-Channel mu 3215
Trilepton Jets 4981
Dilepton Jets 61400
passing 1412
```

Medium (+tHW,ttW,WZ)

```
all events 30268764
all jets 173353642
all muons 26816638
electrons 31032707
loose muons 16035325
fakeable muons 12382828
tight muons 10345973
loose electrons 13542971
fakeable electrons 8050688
tight electrons 2993715
Tight Dilepton-Channel e 65959
Tight Dilepton-Channel mu 201125
Tight Trilepton-Channel e 9772
Tight Trilepton-Channel mu 26651
Trilepton Jets 46847
Dilepton Jets 626474
passing 12141
```

Large*

```
all events 90988105
all jets 662845397
all muons 79309765
electrons 110891123
loose muons 39537817
fakeable muons 32283727
tight muons 26262265
loose electrons 35680939
fakeable electrons 21218904
tight electrons 7758698
Tight Dilepton-Channel e 171429
Tight Dilepton-Channel mu 437223
Tight Trilepton-Channel e 39200
Tight Trilepton-Channel mu 116449
Trilepton Jets 238550
Dilepton Jets 1555519
passing 51883
```

'tHq': glob.glob('/mnt/hadoop/user/clundst/data/THQ/*'),
'tHW': glob.glob('/store/mc/RunIISummer16NanoAODv5/THW_Hincl_13TeV-madgraph-pythia8_TuneCUETP8M1/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/70000/*'),
'ttW': glob.glob('/store/mc/RunIISummer16NanoAODv5/TTWJetsToLNu_TuneCUETP8M1_13TeV-amcatnloFXFX-madspin-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7_ext*/*/*'),
'WZ': glob.glob('/store/mc/RunIISummer16NanoAODv5/WZTo3LNu_TuneCUETP8M1_13TeV-powheg-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7*/*/*'),

'TTZ': glob.glob('/store/mc/RunIISummer16NanoAODv5/TTZToLLNuNu_M-10_TuneCUETP8M1_13TeV-amcatnlo-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7_ext*/*/*'),
'ttH': glob.glob('/store/mc/RunIISummer16NanoAODv5/ttHJetToNonbb_M125_13TeV_amcatnloFXFX_madspin_pythia8_mWCutfix/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7_ext1-v1/*/*'),
'WGToLNuG': glob.glob('/store/mc/RunIISummer16NanoAODv5/WGToLNuG_TuneCUETP8M1_13TeV-madgraphMLM-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
'ZGTo2LG': glob.glob('/store/mc/RunIISummer16NanoAODv5/ZGTo2LG_TuneCUETP8M1_13TeV-amcatnloFXFX-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7_ext1-v1/*/*'),

Datasets for Nebraska Large Run (cont)

```
        'TGJets': glob.glob('/store/mc/RunIISummer16NanoAODv5/TGJets_TuneCUETP8M1_13TeV_amcatnlo_madspin_pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7*/*/
*'),
        'TTGJets': glob.glob('/store/mc/RunIISummer16NanoAODv5/TTGJets_TuneCUETP8M1_13TeV-amcatnloFXFX-madspin-pythia8/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7*/*/*'),
         'WpWpJJ_EWK-QCD': glob.glob('/store/mc/RunIISummer16NanoAODv5/WpWpJJ_EWK-QCD_TuneCUETP8M1_13TeV-madgraph-pythia8/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'ZZZ': glob.glob('/store/mc/RunIISummer16NanoAODv5/ZZZ_TuneCUETP8M1_13TeV-amcatnlo-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'WWZ': glob.glob('/store/mc/RunIISummer16NanoAODv5/WWZ_TuneCUETP8M1_13TeV-amcatnlo-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'WZZ': glob.glob('/store/mc/RunIISummer16NanoAODv5/WZZ_TuneCUETP8M1_13TeV-amcatnlo-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'ST_tWll_5f_LO': glob.glob('/store/mc/RunIISummer16NanoAODv5/ST_tWll_5f_LO_13TeV-MadGraph-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'TTTT': glob.glob('/store/mc/RunIISummer16NanoAODv5/TTTT_TuneCUETP8M1_13TeV-amcatnlo-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'WZTo3LNu': glob.glob('/store/mc/RunIISummer16NanoAODv5/WZTo3LNu_TuneCUETP8M1_13TeV-powheg-pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7*/*/*'),
        'ZZTo4L': glob.glob('/store/mc/RunIISummer16NanoAODv5/ZZTo4L_13TeV_powheg_pythia8/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'TTJets_SingleLeptFromTbar': glob.glob('/store/mc/RunIISummer16NanoAODv5/TTJets_SingleLeptFromTbar_TuneCUETP8M1_13TeV-madgraphMLM-pythia8/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7*/*/*'),
        'TTJets_SingleLeptFromT': glob.glob('/store/mc/RunIISummer16NanoAODv5/TTJets_SingleLeptFromT_TuneCUETP8M1_13TeV-madgraphMLM-pythia8/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7*/*/*'),
         'TTJets_DiLept': glob.glob('/store/mc/RunIISummer16NanoAODv5/TTJets_DiLept_TuneCUETP8M1_13TeV-madgraphMLM-pythia8/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7*/*/*'),
        'DYJetsToLL_M-10to50': glob.glob('/store/mc/RunIISummer16NanoAODv5/DYJetsToLL_M-10to50_TuneCUETP8M1_13TeV-amcatnloFXFX-pythia8/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7*/*/*'),
         'DYJetsToLL_M-50': glob.glob('/store/mc/RunIISummer16NanoAODv5/DYJetsToLL_M-50_TuneCUETP8M1_13TeV-madgraphMLM-pythia8/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7_ext*/*/*'),
        'WJetsToLNu': glob.glob('/store/mc/RunIISummer16NanoAODv5/WJetsToLNu_TuneCUETP8M1_13TeV-amcatnloFXFX-pythia8/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7*/*/*'),
         'ST_tW_top_5f': glob.glob('/store/mc/RunIISummer16NanoAODv5/ST_tW_top_5f_inclusiveDecays_13TeV-powheg-pythia8_TuneCUETP8M1/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7_ext1-v1/*/*'),
        'ST_tW_antitop_5f': glob.glob('/store/mc/RunIISummer16NanoAODv5/ST_tW_antitop_5f_inclusiveDecays_13TeV-powheg-pythia8_TuneCUETP8M1/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7_ext1-v1/*/*'),
        'ST_s-channel_4f': glob.glob('/store/mc/RunIISummer16NanoAODv5/ST_s-channel_4f_leptonDecays_13TeV-amcatnlo-pythia8_TuneCUETP8M1/NANOAODSIM/
PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'WWTo2L2Nu': glob.glob('/store/mc/RunIISummer16NanoAODv5/WWTo2L2Nu_13TeV-powheg/NANOAODSIM/PUMoriond17_Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'ttWJets': glob.glob('/store/mc/RunIISummer16NanoAODv5/ttWJets_13TeV_madgraphMLM/NANOAODSIM/Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
        'ttZJets': glob.glob('/store/mc/RunIISummer16NanoAODv5/ttZJets_13TeV_madgraphMLM/NANOAODSIM/Nano1June2019_102X_mcRun2_asymptotic_v7-v1/*/*'),
```

Purdue Test Analysis Details

– VBF channel of H->μμ search, targeting the final state with two muons and two jets.

– The analysis is template-based, a variable sensitive to S/B is constructed using DNN and then fit to data (therefore Data/MC agreement is important)

– Full event selection, muon selection, jet selection, b-jet veto and electron veto implemented

– The most important corrections include FSR recovery, Rochester correction, jet energy corrections (latter two implemented using Coffea tools); also, numerous corrections and weights are implemented with Coffea as lookup tables (pile-up reweighing, Z_pT reweighing, jet PU ID weights).

– Save two types of the output: binned, to quickly make validation plots after processing, and unbinned – to use for MVA training and further statistical analysis

– Some numbers to give an idea about how many events are filtered out

Example for one SingleMuon 2017B file. Selection efficiency will be of course different for MC events:

- Input: ~144500 events
- Passing lumimask, HLT and with at least 2 (any) muons: ~8400 events
- Passing muon selections: ~2600 events
- Have 2 jets passing all jet selections: 444 events
- Falling into VBF category (dijet mass > 400 GeV, rapidity separation between jets deltaEta>2.5): 8 events

Detailed Purdue Setup Steps

Step 0) Set up environment:

```
--------------------------
$ module load anaconda/5.3.1-py37
$ conda create --name daskproj
$ source activate daskproj
$ conda install dask ipython
$ conda install -c conda-forge dask-jobqueue jupyter-server-proxy
```

At command prompt =

In Jupyter →

Step 1) Start a scheduler:

```
--------------------------
$ ssh hammer-c019    # this is a regular SLURM node
$ module load anaconda/5.3.1-py37
$ source activate daskproj
$ ipython
```

```
From dask_jobqueue import SLURMClusterIn [1]: from dask_jobqueue import SLURMCluster

In [2]: cluster = SLURMCluster( account='standby', cores=2, memory='10GB', walltime='00:30:00', job_extra=['--qos=normal', '-o dask_job.%j.
%N.out','-e dask_job.%j.%N.error'])

In [3]: print(cluster)

SLURMCluster('tcp://128.211.149.152:35144', workers=0, threads=0, memory=0 B)  # the <IP:port> is your 'cluster' object to use later

In [4]: cluster.dashboard_link
Out[4]: 'https://128.211.149.152:8787
```

Enter this URL in a web browser to monitor your DASK cluster

Detailed Purdue Steps (cont)

Step 2) Scale the cluster:

Two options:
2a) manually control the size:

```
In [5]: cluster.scale(50)

In [6]: print(cluster)
SLURMCluster('tcp://128.211.149.152:35144', workers=25, threads=50, memory=250.00 GB)
```

Or
2b) Adaptive scaling:

```
In [5]: cluster.adapt(minimum=10, maximum=100)
Out[5]: <distributed.deploy.adaptive.Adaptive at 0x2afc27fd5580>

In [6]: print(cluster)
SLURMCluster('tcp://128.211.149.152:35144', workers=10, threads=20, memory=100.00 GB)
```

Step 3) Create a client:

On the same machine, or on a different one:

```
$ module load anaconda/5.3.1-py37
$ source activate daskproj
$ ipython
```

```
In [1]: from distributed import Client

In [2]: client = Client('128.211.149.152:35144')

In [3]: print(client)
<Client: 'tcp://128.211.149.152:35144' processes=10 threads=20, memory=100.00 GB>
```

Detailed Purdue Steps (cont)

Step 4) Send some tasks to the workers:

The 'client' above can be passed as an argument to Coffea's 'dask_executor', which is loaded using Coffea's function 'run_uproot_job'.

So basically Coffea executor can understand a client instance as a parameter, and start interacting with it.