

# Introduction to Comet

## *Launching and Managing Jobs*

**Mahidhar Tatineni**  
**SDSC Summer Institute**  
**July 31, 2017**

# Comet

## “HPC for the long tail of science”



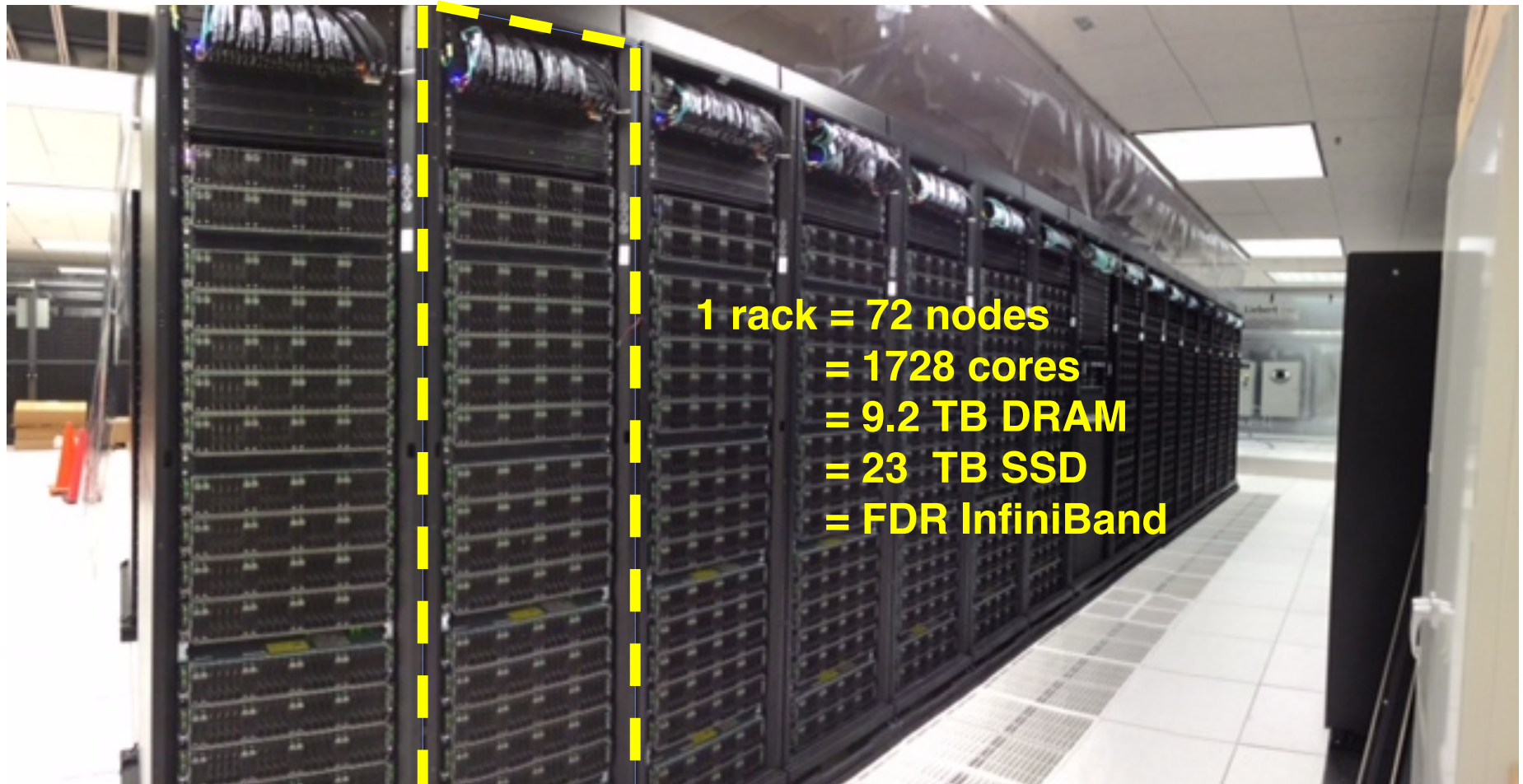
**iPhone panorama photograph of 1 of 2 server rows**

# Comet: System Characteristics

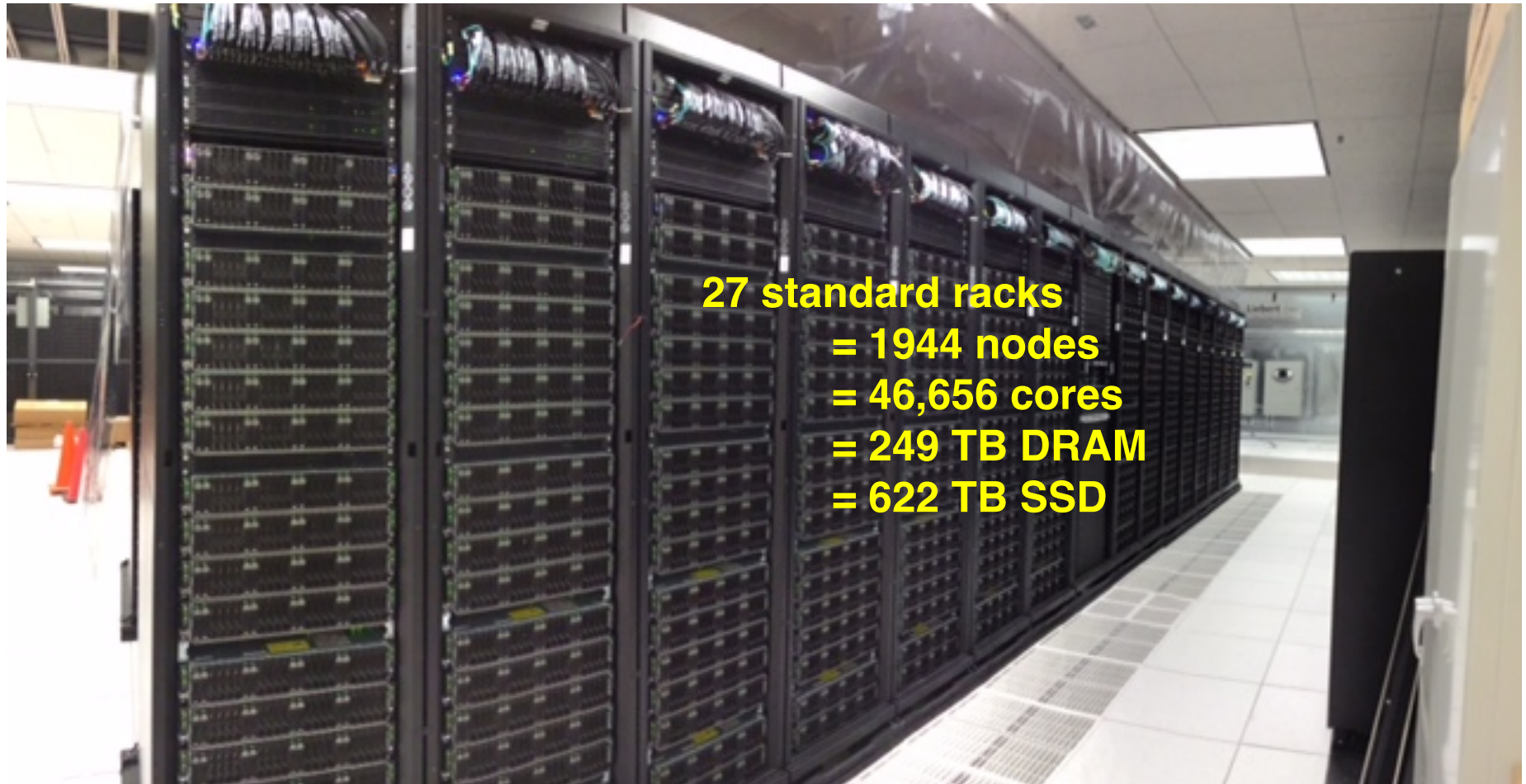
- **Total peak flops ~2.1 PF**
- **Dell primary integrator**
  - Intel Haswell processors w/ AVX2
  - Mellanox FDR InfiniBand
- **1,944 standard compute nodes (46,656 cores)**
  - Dual CPUs, each 12-core, 2.5 GHz
  - 128 GB DDR4 2133 MHz DRAM
  - 2\*160GB GB SSDs (local disk)
- **72 GPU nodes**
  - 36 nodes same as standard nodes *plus* Two NVIDIA K80 cards, each with dual Kepler3 GPUs
  - 36 nodes with 2 14-core Intel Broadwell CPUs plus 4 NVIDIA P100 GPUs
- **4 large-memory nodes**
  - 1.5 TB DDR4 1866 MHz DRAM
  - Four Haswell processors/node
  - 64 cores/node
- **Hybrid fat-tree topology**
  - FDR (56 Gbps) InfiniBand
  - Rack-level (72 nodes, 1,728 cores) full bisection bandwidth
  - 4:1 oversubscription cross-rack
- **Performance Storage (Aeon)**
  - 7.6 PB, 200 GB/s; Lustre
  - Scratch & Persistent Storage segments
- **Durable Storage (Aeon)**
  - 6 PB, 100 GB/s; Lustre
  - Automatic backups of critical data
- **Home directory storage**
- **Gateway hosting nodes**
- **Virtual image repository**
- **100 Gbps external connectivity to Internet2 & ESNet**



# ~67 TF supercomputer in a rack



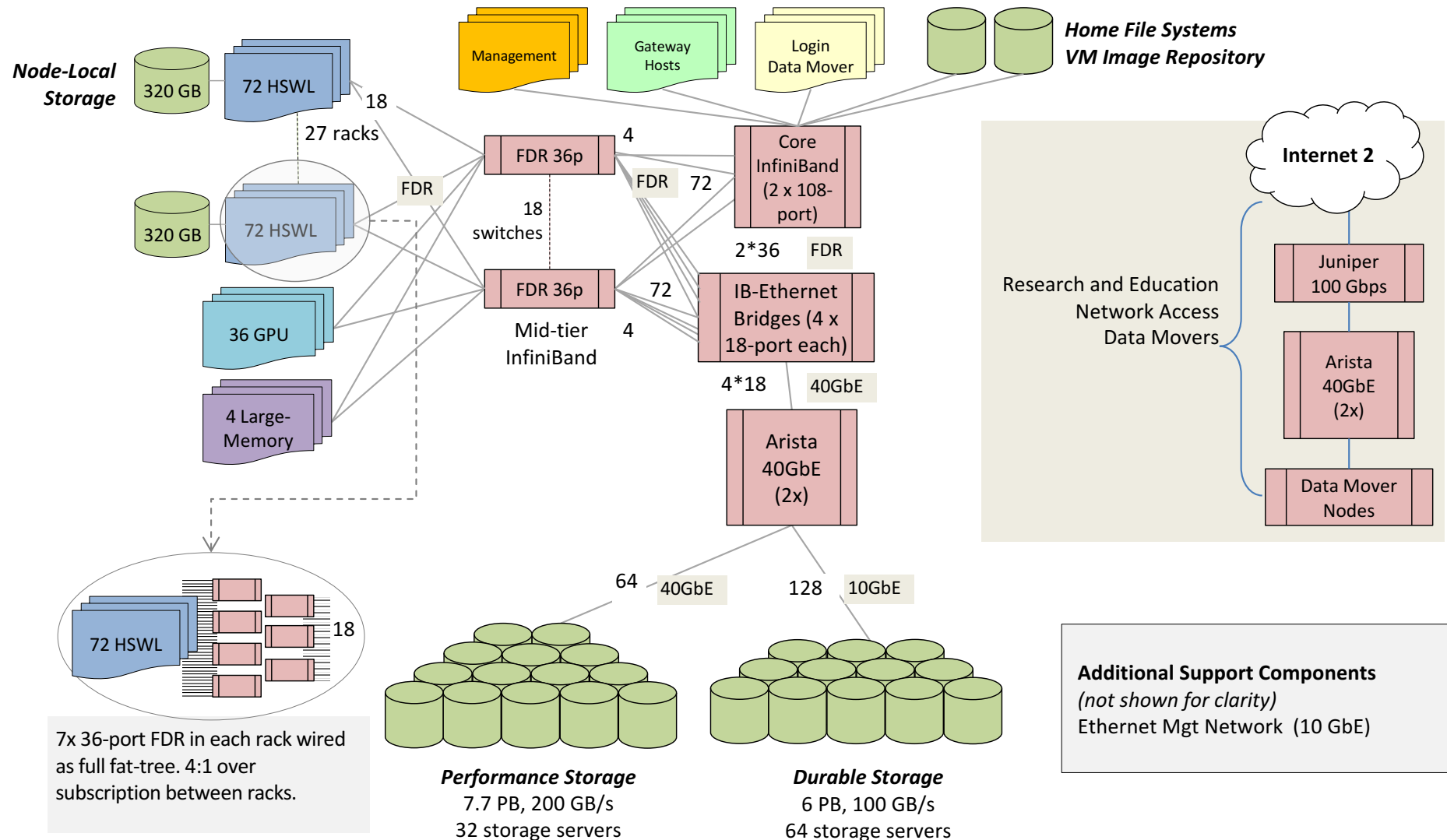
# And 27 single-rack supercomputers



**27 standard racks**  
**= 1944 nodes**  
**= 46,656 cores**  
**= 249 TB DRAM**  
**= 622 TB SSD**

# Comet Network Architecture

## InfiniBand compute, Ethernet Storage



# Getting Started

- **System Access – Logging in**
  - Linux/Mac – Use available ssh clients.
  - ssh clients for windows – Putty, Cygwin
    - <http://www.chiark.greenend.org.uk/~sgtatham/putty/>
  - Login hosts for the SDSC Comet:
    - comet.sdsc.edu

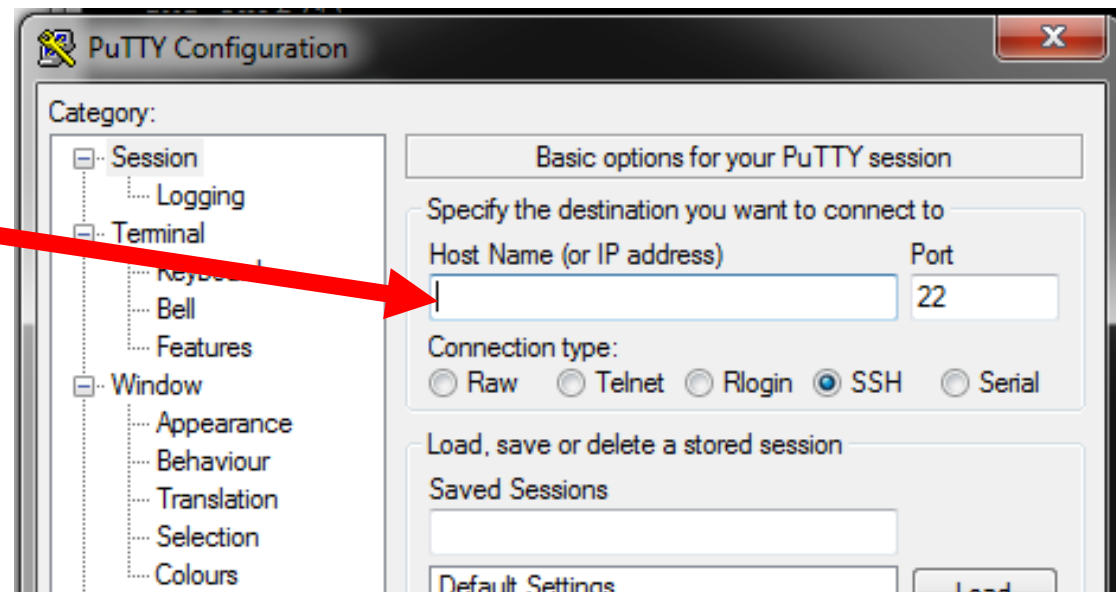
# Logging into Comet

Mac/Linux:

```
ssh username@comet.sdsc.edu
```

Windows (PuTTY):

comet.sdsc.edu





# Comet: Filesystems

- **Lustre filesystems – Good for scalable large block I/O**
  - Accessible from all compute and GPU nodes.
  - /oasis/scratch/comet - 2.5PB, peak performance: 100GB/s. Good location for storing large scale scratch data during a job.
  - /oasis/projects/nsf - 2.5PB, peak performance: 100 GB/s. Long term storage.
  - ***Not good for lots of small files or small block I/O.***
- **SSD filesystems**
  - /scratch local to each native compute node – 210GB on regular compute nodes, 285GB on GPU, large memory nodes, 1.4TB on selected compute nodes.
  - SSD location is good for writing small files and temporary scratch files. Purged at the end of a job.
- **Home directories (/home/\$USER)**
  - Source trees, binaries, and small input files.
  - ***Not good for large scale I/O.***

# Comet: System Environment

- **Modules used to manage environment for users.**
- **Default environment:**

**\$ module li**

Currently Loaded Modulefiles:

1) intel/2013\_sp1.2.144 2) mvapich2\_ib/2.1 3) gnutools/2.69

- **Listing available modules:**

**\$ module av**

----- /opt/modulefiles/mpi/.intel -----

intelmpi/2016.3.210(default) mvapich2\_ib/2.1(default)

mvapich2\_gdr/2.1(default) openmpi\_ib/1.8.4(default)

mvapich2\_gdr/2.2

----- /opt/modulefiles/applications/.intel -----

atlas/3.10.2(default) lapack/3.6.0(default) scalapack/2.0.2(default)

boost/1.55.0(default) mxml/2.9(default) slepc/3.6.2(default)

...

...

# Comet: System Environment

- **Loading modules:**

```
$ module load fftw/3.3.4
```

```
$ module li
```

Currently Loaded Modulefiles:

- 1) intel/2013\_sp1.2.144    3) gnutools/2.69
- 2) mvapich2\_ib/2.1        4) fftw/3.3.4

- **See what a module does:**

```
$ module show fftw/3.3.4
```

```
-----  
/opt/modulefiles/applications/.intel/fftw/3.3.4:  
module-whatism fftw  
module-whatism Version: 3.3.4  
module-whatism Description: fftw  
module-whatism Compiler: intel  
module-whatism MPI Flavors: mvapich2_ib openmpi_ib  
setenv FFTWHOME /opt/fftw/3.3.4/intel/mvapich2_ib  
prepend-path PATH /opt/fftw/3.3.4/intel/mvapich2_ib/bin  
prepend-path LD_LIBRARY_PATH /opt/fftw/3.3.4/intel/mvapich2_ib/lib  
prepend-path LIBPATH /opt/fftw/3.3.4/intel/mvapich2_ib/lib  
-----
```

# Comet: System Environment

**\$ echo \$PATH**

**/opt/fftw/3.3.4/intel/mvapich2\_ib/bin:/share/apps/compute/b  
bftp/bin:/home/mahidhar/pdsh/bin:/opt/gnu/gcc/bin:/opt/gn  
u/bin:/opt/mvapich2/intel/ib/bin:/opt/intel/composer\_xe\_201  
3\_sp1.2.144/bin/intel64:/opt/intel/composer\_xe\_2013\_sp1.2.  
144/mpi/bin/intel64:/opt/intel/composer\_xe\_2013\_sp1.2.14  
4/debugger/gdb/intel64\_mic/bin:/usr/lib64/qt-  
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/s  
bin:/opt/ibutils/bin:/usr/java/latest/bin:/opt/pdsh/bin:/opt/roc  
ks/bin:/opt/rocks/sbin:/opt/sdsc/bin:/opt/sdsc/sbin:/home/m  
ahidhar/bin**

**\$ echo \$FFTWHOME**

**/opt/fftw/3.3.4/intel/mvapich2\_ib**



# Parallel Programming

- Comet supports MPI, OpenMP, and Pthreads for parallel programming. Hybrid modes are possible.
- GPU nodes support CUDA, OpenACC.
- MPI
  - Default: mvapich2\_ib/2.1
  - Other options: openmpi\_ib/1.8.4 (and 1.10.2), Intel MPI
  - mvapich2\_gdr: GPU direct enabled version
- **OpenMP:** All compilers (GNU, Intel, PGI) have OpenMP flags.
- Default Intel Compiler: **intel/2013\_sp1.2.144;**  
*Versions 2015.2.164 and 2016.3.210 available.*

# Running Jobs on Comet

- **Important note: Do not run on the login nodes - even for simple tests.**
- **All runs must be via the Slurm scheduling infrastructure.**
  - Interactive Jobs: Use **srun** command:  
*srun --pty --nodes=1 --ntasks-per-node=24 -p debug -t 00:30:00 --wait 0 /bin/bash*
  - Batch Jobs: Submit batch scripts from the login nodes.  
Can choose:
    - Partition (details on upcoming slide)
    - Time limit for the run (maximum of 48 hours)
    - Number of nodes, tasks per node
    - Memory requirements (if any)
    - Job name, output file location
    - Email info, configuration

# Slurm Partitions

Queue Name	Max Waltime	Max Nodes	Comments
compute	48 hrs	72	Used for access to regular compute nodes
gpu	48 hrs	4	Used for access to the GPU nodes
gpu-shared	48 hrs	1	Used for shared access to a partial GPU node
shared	48 hrs	1	Single-node jobs using fewer than 24 cores
large-shared	48 hrs	1	Single-node jobs using large memory up to 1.45 TB
debug	30 mins	2	Used for access to debug nodes

- Specified using -p option in batch script. For example:

**#SBATCH -p gpu**

# Slurm Commands

- Submit jobs using the **sbatch** command:

```
$ sbatch Localscratch-slurm.sb
```

```
Submitted batch job 8718049
```

- Check job status using the **squeue** command:

```
$ squeue -u $USER
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
8718049	compute	localscr	mahidhar	PD	0:00	1	(Priority)

- Once the job is running:

```
$ squeue -u $USER
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
8718064	debug	localscr	mahidhar	R	0:02	1	comet-14-01



# Comet Compute Nodes

**2-Socket (Total 24 cores) Intel Haswell Processors**

**Hands On Examples using:**

**(1) MPI**

**(2) OpenMP**

**(3) HYBRID**

**(4) Local scratch**

# Comet – Compiling/Running Jobs

- Copy and change to directory (assuming you already copied the PHYS244 directory):

```
cd /home/$USER/SI2017/MPI
```

- Verify modules loaded:

```
module list
```

Currently Loaded Modulefiles:

```
1) intel/2013_sp1.2.144 2) mvapich2_ib/2.1 3) gnutools/2.69
```

- Compile the MPI hello world code:

```
mpif90 -o hello_mpi hello_mpi.f90
```

- Verify executable has been created:

```
ls -lt hello_mpi
```

```
-rwxr-xr-x 1 mahidhar sdsc 721912 Mar 25 14:53 hello_mpi
```

- Submit job from IBRUN directory:

```
cd /home/$USER/SI2017/MPI/IBRUN
```

```
sbatch --res=SI2017DAY1 hellompi-slurm.sb
```

# Comet: Hello World on compute nodes

The submit script is hellompi-slurm.sb:

```
#!/bin/bash
#SBATCH --job-name="hellompi"
#SBATCH --output="hellompi.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00
```

#This job runs with 2 nodes, 24 cores per node for a total of 48 cores.  
#ibrun in verbose mode will give binding detail

```
ibrun -v ./hello_mpi
```

# Comet: Hello World on compute nodes

IBRUN: Command is ../hello\_mpi

IBRUN: Command is /share/apps/examples/MPI/hello\_mpi

...

...

IBRUN: MPI binding policy: **compact/core for 1 threads per rank (12 cores per socket)**

IBRUN: Adding MV2\_CPU\_BINDING\_LEVEL=core to the environment

IBRUN: Adding MV2\_ENABLE\_AFFINITY=1 to the environment

IBRUN: Adding MV2\_DEFAULT\_TIME\_OUT=23 to the environment

IBRUN: Adding **MV2\_CPU\_BINDING\_POLICY=bunch** to the environment

...

...

IBRUN: Added 8 new environment variables to the execution environment

IBRUN: Command string is [**mpirun\_rsh -np 48 -hostfile /tmp/rssSvauaJA -export /share/apps/examples/MPI/hello\_mpi**]

node 18 : Hello world

node 13 : Hello world

node 2 : Hello world

node 10 : Hello world



# Compiling OpenMP Example

- Change to the examples directory:

```
cd /home/$USER/SI2017/OPENMP
```

- Compile using `-openmp` flag:

```
ifort -o hello_openmp -openmp hello_openmp.f90
```

- Verify executable was created:

```
[mahidhar@comet-08-11 OPENMP]$ ls -lt hello_openmp  
-rwxr-xr-x 1 mahidhar sdsc 750648 Mar 25 15:00 hello_openmp
```

# OpenMP job script

```
#!/bin/bash
#SBATCH --job-name="hell_openmp"
#SBATCH --output="hello_openmp.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00

#SET the number of openmp threads
export OMP_NUM_THREADS=24

#Run the job using mpirun_rsh
./hello_openmp
```

# Output from OpenMP Job

**\$ more hello\_openmp.out**

```
HELLO FROM THREAD NUMBER = 7
HELLO FROM THREAD NUMBER = 6
HELLO FROM THREAD NUMBER = 9
HELLO FROM THREAD NUMBER = 8
HELLO FROM THREAD NUMBER = 5
HELLO FROM THREAD NUMBER = 4
HELLO FROM THREAD NUMBER = 0
HELLO FROM THREAD NUMBER = 12
HELLO FROM THREAD NUMBER = 14
HELLO FROM THREAD NUMBER = 3
HELLO FROM THREAD NUMBER = 13
HELLO FROM THREAD NUMBER = 10
HELLO FROM THREAD NUMBER = 11
HELLO FROM THREAD NUMBER = 2
HELLO FROM THREAD NUMBER = 1
HELLO FROM THREAD NUMBER = 15
```

# Running Hybrid (MPI + OpenMP) Jobs

- Several HPC codes use a hybrid MPI, OpenMP approach.
- **“ibrun”** wrapper developed to handle such hybrid use cases. Automatically senses the MPI build (mvapich2, openmpi) and binds tasks correctly.
- **“ibrun -help”** gives detailed usage info.
- **hello\_hybrid.c** is a sample code, and **hello\_hybrid.cmd** shows “ibrun” usage.



# hello\_hybrid.cmd

```
#!/bin/bash
#SBATCH --job-name="hellohybrid"
#SBATCH --output="hellohybrid.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00
```

#This job runs with 2 nodes, 24 cores per node for a total of 48 cores.

# We use 8 MPI tasks and 6 OpenMP threads per MPI task

```
export OMP_NUM_THREADS=6
ibrun --npernode 4 ./hello_hybrid
```

# Hybrid Code Output

[etrain61@comet-ln3 HYBRID]\$ **more hellohybrid.8557716.comet-14-01.out**

Hello from thread 0 out of 6 from process 2 out of 8 on comet-14-01.local

Hello from thread 3 out of 6 from process 2 out of 8 on comet-14-01.local

Hello from thread 4 out of 6 from process 2 out of 8 on comet-14-01.local

Hello from thread 5 out of 6 from process 2 out of 8 on comet-14-01.local

Hello from thread 0 out of 6 from process 3 out of 8 on comet-14-01.local

Hello from thread 2 out of 6 from process 2 out of 8 on comet-14-01.local

Hello from thread 1 out of 6 from process 3 out of 8 on comet-14-01.local

Hello from thread 2 out of 6 from process 3 out of 8 on comet-14-01.local

...

...

Hello from thread 4 out of 6 from process 7 out of 8 on comet-14-02.local

Hello from thread 2 out of 6 from process 7 out of 8 on comet-14-02.local

Hello from thread 3 out of 6 from process 7 out of 8 on comet-14-02.local

Hello from thread 5 out of 6 from process 7 out of 8 on comet-14-02.local

Hello from thread 1 out of 6 from process 6 out of 8 on comet-14-02.local

# Using SSD Scratch

```
#!/bin/bash
```

```
#SBATCH --job-name="localscratch"
```

```
#SBATCH --output="localscratch.%j.%N.out"
```

```
#SBATCH --partition=compute
```

```
#SBATCH --nodes=1
```

```
#SBATCH --ntasks-per-node=24
```

```
#SBATCH --export=ALL
```

```
#SBATCH -t 01:30:00
```

```
#Copy binary to SSD
```

```
cp IOR.exe /scratch/$USER/$SLURM_JOBID
```

```
#Change to local scratch (SSD) and run IOR benchmark
```

```
cd /scratch/$USER/$SLURM_JOBID
```

```
#Run IO benchmark
```

```
ibrun -np 4 ./IOR.exe -F -t 1m -b 4g -v -v > IOR.out.$SLURM_JOBID
```

```
#Copy out data you need
```

```
cp IOR.out.$SLURM_JOBID $SLURM_SUBMIT_DIR
```

# Using SSD Scratch

- Snapshot on the node during the run:

```
$ pwd
```

```
/scratch/mahidhar/435463
```

```
$ ls -lt
```

```
total 22548292
```

```
-rw-r--r-- 1 mahidhar hpss 5429526528 May 15 23:48 testFile.000000001
```

```
-rw-r--r-- 1 mahidhar hpss 6330253312 May 15 23:48 testFile.000000003
```

```
-rw-r--r-- 1 mahidhar hpss 5532286976 May 15 23:48 testFile.000000000
```

```
-rw-r--r-- 1 mahidhar hpss 5794430976 May 15 23:48 testFile.000000002
```

```
-rw-r--r-- 1 mahidhar hpss      1101 May 15 23:48 IOR_native_scratch.log
```

- Performance from single node (in log file copied back):
  - Max Write: 250.52 MiB/sec (262.69 MB/sec)
  - Max Read: 181.92 MiB/sec (190.76 MB/sec)

28

# Comet GPU Nodes

**2 NVIDIA K-80 Cards (4 GPUs total) per node.**

- [1] CUDA code compile and run example**
- [2] Hands On Examples using Singularity to enable Tensorflow**

# Compiling CUDA Example

- **Load the CUDA module:**  
`module load cuda`
- **Compile the code:**  
`cd /home/$USER/SI2017/CUDA`  
`nvcc -o matmul -I. matrixMul.cu`
- **Submit the job:**  
`sbatch --res=SI2017DAY1 cuda.sb`



# CUDA Example: Batch Submission Script

```
#!/bin/bash
#SBATCH --job-name="CUDA"
#SBATCH --output="CUDA.%j.%N.out"
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH -t 01:00:00
```

```
#Load the cuda module
module load cuda
```

```
#Run the job
./matmul
```

# ***Singularity: Provides Flexibility for OS Environment***

- Singularity (<http://singularity.lbl.gov>) is a relatively new development that has become very popular on Comet.
- Singularity allows groups to easily migrate complex software stacks from their campus to Comet.
- Singularity runs in user space, and requires very little special support – in fact it actually reduces it in some cases.
- We have roughly 15 groups running this on Comet.
- Applications include: Tensorflow, Paraview, Torch, Fenics, and custom user applications.
- Docker images can be imported into Singularity.

# Tensorflow via Singularity

```
#!/bin/bash
#SBATCH --job-name="TensorFlow"
#SBATCH --output="TensorFlow.%j.%N.out"
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:k80:1
#SBATCH -t 01:00:00
```

```
#Run the job
#
```

```
module load singularity
```

```
singularity exec /share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img lsb_release -a
```

```
singularity exec /share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img python -m tensorflow.models.image.mnist.convolutional
```

# Tensorflow via Singularity

- **Change to the examples directory:**

```
cd /home/$USER/SI2017/TensorFlow
```

- **Submit the job:**

```
sbatch --res=SI2017DAY1 TensorFlow.sb
```

# Tensorflow Example: Output

Distributor ID: Ubuntu

Description: Ubuntu 16.04 LTS

Release: 16.04

Codename: xenial

I tensorflow/stream\_executor/dso\_loader.cc:108] successfully opened CUDA library libcublas.so locally

I tensorflow/stream\_executor/dso\_loader.cc:108] successfully opened CUDA library libcudnn.so locally

I tensorflow/stream\_executor/dso\_loader.cc:108] successfully opened CUDA library libcufft.so locally

I tensorflow/stream\_executor/dso\_loader.cc:108] successfully opened CUDA library libcuda.so.1 locally

I tensorflow/stream\_executor/dso\_loader.cc:108] successfully opened CUDA library libcurand.so locally

I tensorflow/core/common\_runtime/gpu/gpu\_init.cc:102] Found device 0 with properties:

name: Tesla K80

major: 3 minor: 7 memoryClockRate (GHz) 0.8235

pciBusID 0000:85:00.0

Total memory: 11.17GiB

Free memory: 11.11GiB

I tensorflow/core/common\_runtime/gpu/gpu\_init.cc:126] DMA: 0

I tensorflow/core/common\_runtime/gpu/gpu\_init.cc:136] 0: Y

I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:838] Creating TensorFlow device (/gpu:0) -> (device: 0, name: Tesla K80, pci bus id: 0000:85:00.0)

Extracting data/train-images-idx3-ubyte.gz

...

Step 8500 (epoch 9.89), 11.6 ms

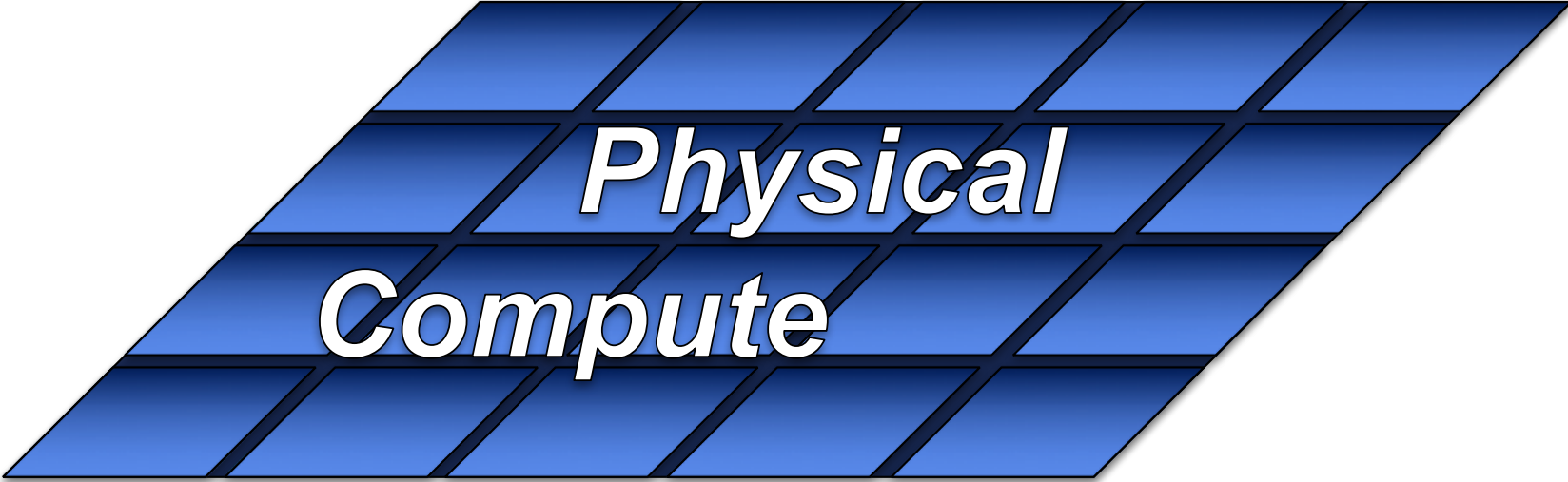
Minibatch loss: 1.601, learning rate: 0.006302

Minibatch error: 0.0%

Validation error: 0.9%

Test error: 0.9%

# Add Data Analysis to Existing Compute Infrastructure



*Physical  
Compute*



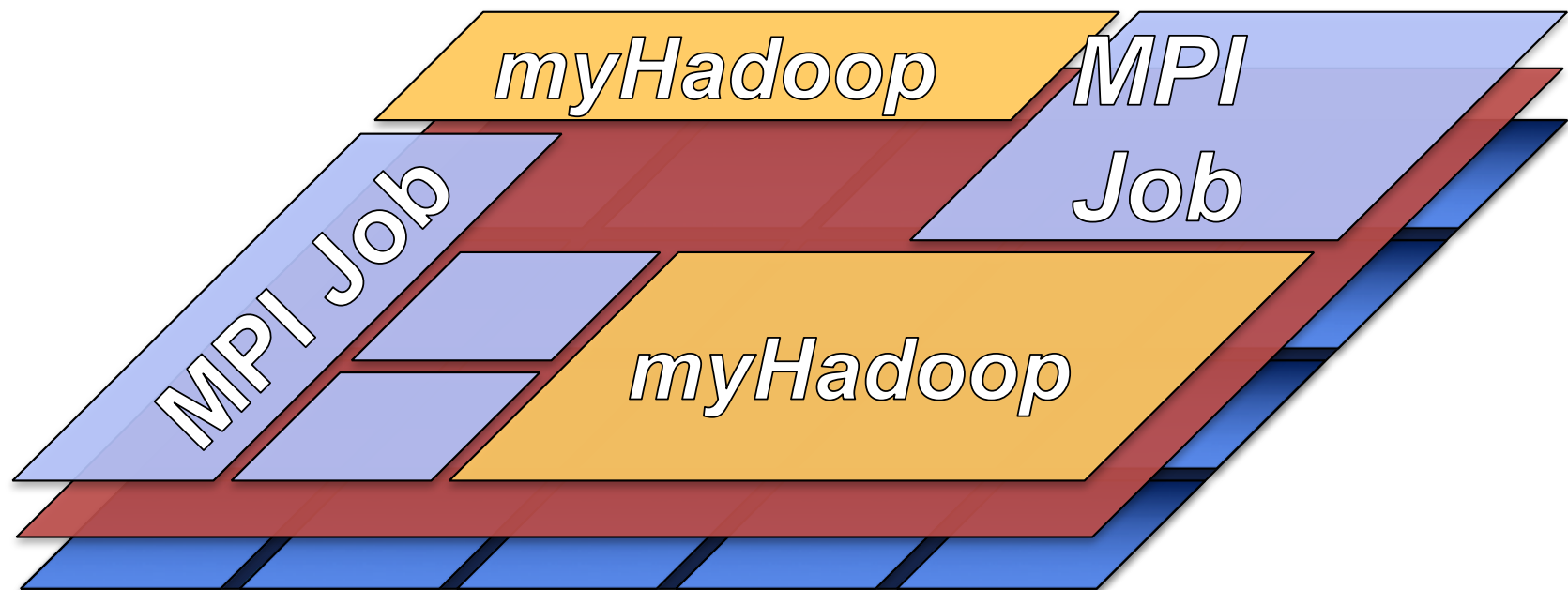
# Add Data Analysis to Existing Compute Infrastructure



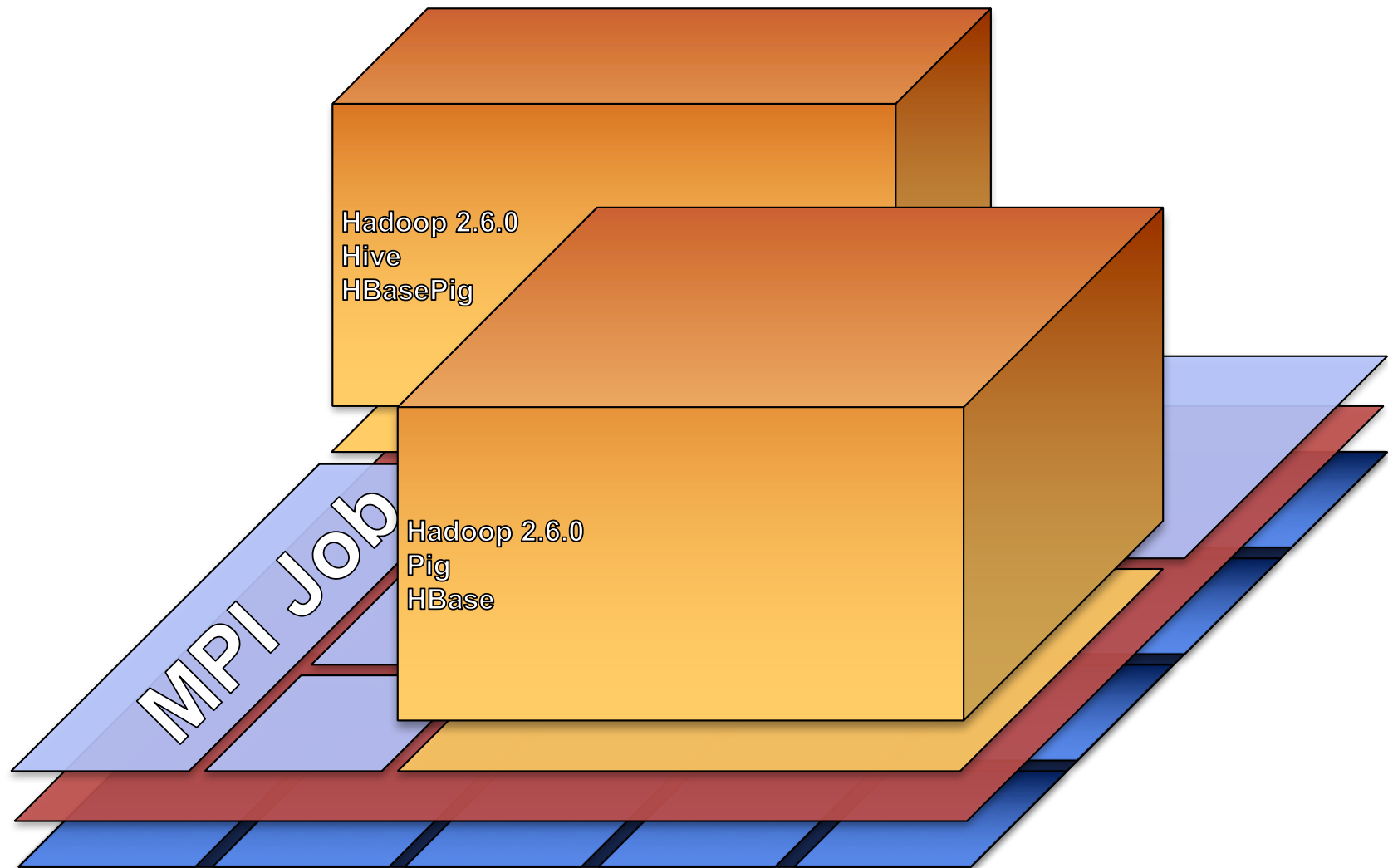
*Resource Manager*

*(Torque, SLURM, SGE)*

# Add Data Analysis to Existing Compute Infrastructure



# Add Data Analysis to Existing Compute Infrastructure



# myHadoop – 3-step Cluster

## 1. Set a few environment variables

```
# sets HADOOP_HOME, JAVA_HOME, and PATH
```

```
$ module load hadoop
```

```
$ export HADOOP_CONF_DIR=$HOME/mycluster.conf
```

## 2. Run myhadoop-configure.sh to set up Hadoop

```
$ myhadoop-configure.sh
```

## 3. Start cluster with Hadoop's start-all.sh

```
$ start-all.sh
```

# Anagram Example – Comet Submit Script

```
#!/bin/bash
#SBATCH --job-name="Anagram"
#SBATCH --output="Anagram.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:00:00
export WRKDIR=`pwd`
myhadoop-configure.sh
start-all.sh
hadoop dfs -mkdir input
hadoop dfs -copyFromLocal $WRKDIR/SINGLE.TXT input/
hadoop jar $WRKDIR/AnagramJob.jar input/SINGLE.TXT output
hadoop dfs -copyToLocal output/part* $PBS_O_WORKDIR
stop-all.sh
myhadoop-cleanup.sh
```

# ANAGRAM Example

- **Change to directory:**

```
cd $HOME/SI2017/hadoop/ANAGRAM_Hadoop2
```

- **Submit job:**

```
sbatch --res=SI2017DAY1 anagram.script
```

- **Check configuration in directory:**

```
ls $HOME/cometcluster
```

# Anagram Example – Sample Output

cat part-00000

...

aabcdelmnu	manducable,ambulanced,
aabcdeorrsst	broadcasters,rebroadcasts,
aabcdeorrst	rebroadcast,broadcaster,
aabcdkrsw	drawbacks,backwards,
aabcdkrw	drawback,backward,
aabceeehlmsst	teachableness,cheatableness,
aabceeehlmsstu	uncreatableness,untraceableness,
aabceeehlrt	recreatable,retraceable,
aabceehl	cheatable,teachable,
aabceellr	lacerable,clearable,
aabceelmrstu	uncreatable,untraceable,
aabceelorrstv	vertebrosacral,sacrovertebral,

...

...



# RDMA-Hadoop and RDMA-Spark

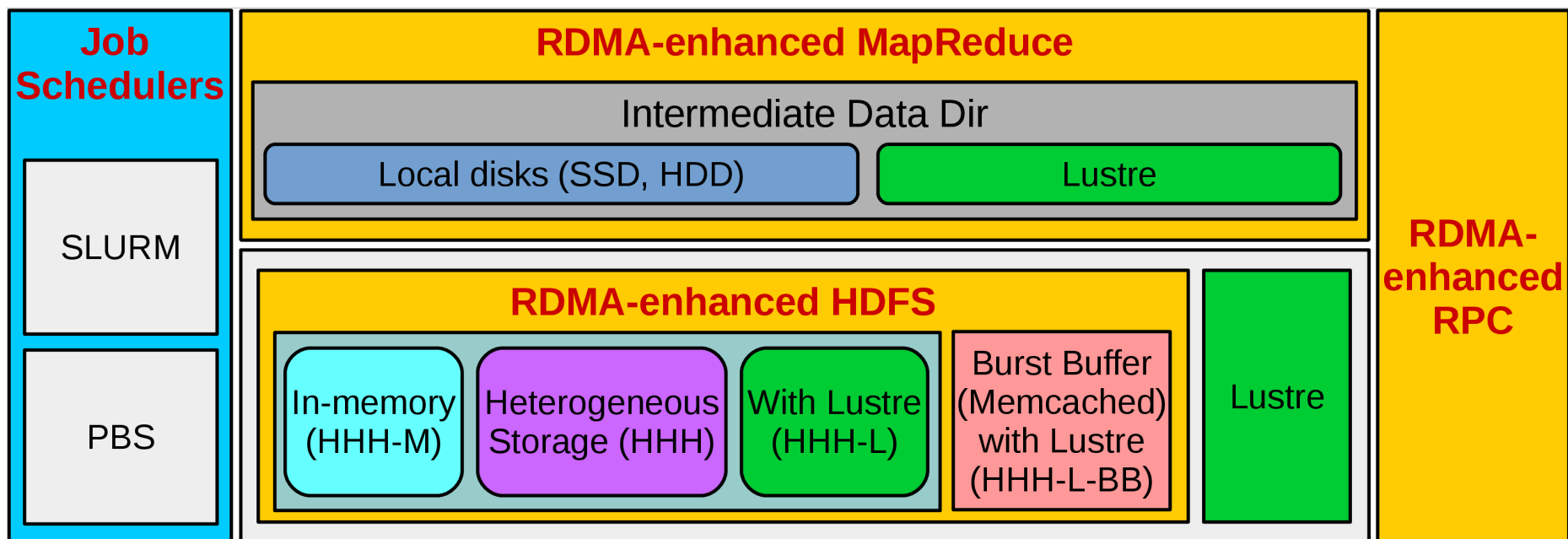
Network-Based Computing Lab, Ohio State University

*NSF funded project in collaboration with Dr. DK Panda*

- HDFS, MapReduce, and RPC over native InfiniBand and RDMA over Converged Ethernet (RoCE).
- Based on Apache distributions of Hadoop and Spark.
- Version **RDMA-Apache-Hadoop-2.x 1.1.0** (based on Apache Hadoop 2.6.0) available on Comet
- Version **RDMA-Spark 0.9.3** (based on Apache Spark 1.5.1) is available on Comet.
- More details on the RDMA-Hadoop and RDMA-Spark projects at:
  - <http://hibd.cse.ohio-state.edu/>

# RDMA-Hadoop, Spark

- Exploit performance on modern clusters with RDMA-enabled interconnects for Big Data applications.
- Hybrid design with in-memory and heterogeneous storage (HDD, SSDs, Lustre).
- Keep compliance with standard distributions from Apache.



# Hands On: Anagram using HHH-M mode

```
#!/bin/bash
#SBATCH --job-name="rdmahadoopanagram"
#SBATCH --output="rdmahadoopanagram.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=24
#SBATCH -t 00:15:00
```

**#Script request 3 nodes - one used for namenode, 2 for data nodes/processing**

**#Set modulepath and load RDMA Hadoop Module**

**export**

**MODULEPATH=/share/apps/compute/modulefiles/applications:\$MODULEPATH**

**module load rdma-hadoop/2x-1.1.0**

# Hands On: Anagram using HHH-M mode

## #Get the host list

```
export SLURM_NODEFILE=`generate_pbs_nodefile`  
cat $SLURM_NODEFILE | sort -u > hosts.hadoop.list
```

## #Use SLURM integrated configuration/startup script

```
hibd_install_configure_start.sh -s -n ./hosts.hadoop.list -i $SLURM_JOBID -h $HA  
DOOP_HOME -j $JAVA_HOME -m hhh-m -r /dev/shm -d /scratch/$USER/$SLURM_JOBID -t /  
scratch/$USER/$SLURM_JOBID/hadoop_local
```

## #Commands to run ANAGRAM example

```
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -mkdir -p /user/$USER  
/input  
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -put SINGLE.TXT /user  
/$USER/input/SINGLE.TXT  
$HADOOP_HOME/bin/hadoop --config $HOME/conf_$SLURM_JOBID jar AnagramJob.jar /use  
r/$USER/input/SINGLE.TXT /user/$USER/output  
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -get /user/$USER/outp  
ut/part* $SLURM_WORKING_DIR
```

## #Clean up

```
hibd_stop_cleanup.sh -d -h $HADOOP_HOME -m hhh-m -r /dev/shm
```

# RDMA-Hadoop: HHH-M Example

- **Change to directory:**

**cd \$HOME/SI2017/hadoop/RDMA-Hadoop/RDMA-HHH-M**

- **Submit job:**

**sbatch --res=SI2017DAY1 anagram.script**

# Summary

- Comet can be directly accessed using a ssh client.
- Always run via the batch scheduler – for both interactive and batch jobs. *Do not run on the login nodes.*
- Choose your filesystem wisely – Lustre parallel filesystem for large block I/O. SSD based filesystems for small block I/O, lots of small files. *Do not use home filesystem for intensive I/O of any kind.*
- Comet can handle MPI, OpenMP, Pthreads, Hybrid, CUDA, and OpenACC jobs. Singularity provides further flexibility.
- Dynamic spin up of Hadoop, Spark instances within Comet scheduler framework.