Analytics and Machine Learning on Cray XC and Intel Systems

Michael Ringenburg and Kristyn Maschhoff (Cray)

Lisa Gerhardt, Rollin Thomas, and Richard Canon (NERSC)

Jing Huang and Vivek Rane (Intel)

CUG 2017 CAFFEINATED COMPUTING

Redmond, Washington May 7-11, 2017

Overall Agenda

- **Session I: 1:00-2:30**
  - Apache Spark (Michael Ringenburg, Cray)
  - Anaconda Python and Dask Distributed (Michael Ringenburg, Cray)
- **CUG Break: 2:30-3:00**
- **Session II: 3:00-4:30**
  - R (Kristyn Maschhoff, Cray)
  - Cray Graph Engine (Kristyn Maschhoff, Cray)
- **CUG Track Change: 4:30-4:40**
- **Session III: 4:40-6:20**
  - TensorFlow (Jing Huang and Vivek Rane, Intel)

Training Accounts

- **NERSC has provided temporary training accounts on Cori for tutorial attendees.**
  - You should have received a slip of paper with your username and password when you signed the user agreement
  - Login: ssh <username>@cori.nersc.gov
  - Allocation via slurm:
    - Haswell nodes: salloc --reservation=CUG2C -N 5 -p regular -C haswell -t 60 -A ntrain
    - KNL nodes: salloc --reservation=CUG2C -N 1 -p regular -C knl -t 60 -A ntrain
  - 150 Haswell and 30 KNL nodes total available in this reservation – please don't exceed your fair share

Tutorial: Spark on Cray XC Systems

Michael Ringenburg, mikeri@cray.com
Principal Engineer, Analytics R&D, Cray Inc

CUG.2017.CAFFEINATED COMPUTING

Redmond, Washington May 7-11, 2017

Agenda

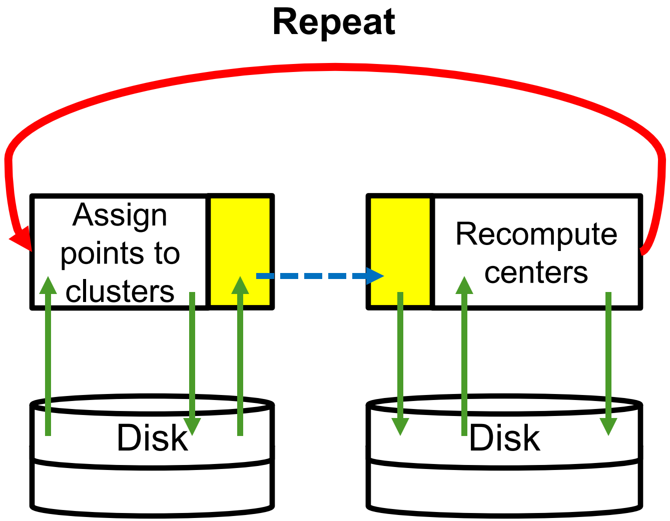
- **Introduction to Spark**
  - History and Background
  - Computation and Communication Model
- **Spark on the XC40**
  - Installation and Configuration
  - Local storage
- **Running Spark on Cori**
- **Questions?**

In the beginning, there was Hadoop MapReduce…

- **MapReduce: simplified parallel programming model**
  - All computations broken into two parts
    - **Embarassingly parallel map phase**: apply single operation to every key,value-pair, produce new set of key,value-pairs
    - **Combining reduce phase**: Group all values with identical key, performing combining operation to get final value for key
  - Can perform multiple iterations for computations that require
  - I/O intensive
    - Map writes to local storage. Data shuffled to reducer's local storage, reduce reads.
    - Additional I/O between iterations in multi-iteration algorithms (map reads from HDFS, reduce writes to HDFS)
  - Effective model for many data analytics tasks
- **HDFS distributed file system (locality aware – move compute to data)**
- **YARN cluster resource manager**

Example: K-Means Clustering with MapReduce

- **Initially: Write out random cluster centers**
- **Map:**
  - Read in cluster centers
  - For each data point, compute nearest cluster center and write <key: nearest cluster, value: data point>
- **Reduce:**
  - For each cluster center (key) compute average of datapoints
  - Write out this value as new cluster center
- **Repeat until convergence (clusters don't change)**



MapReduce Problems

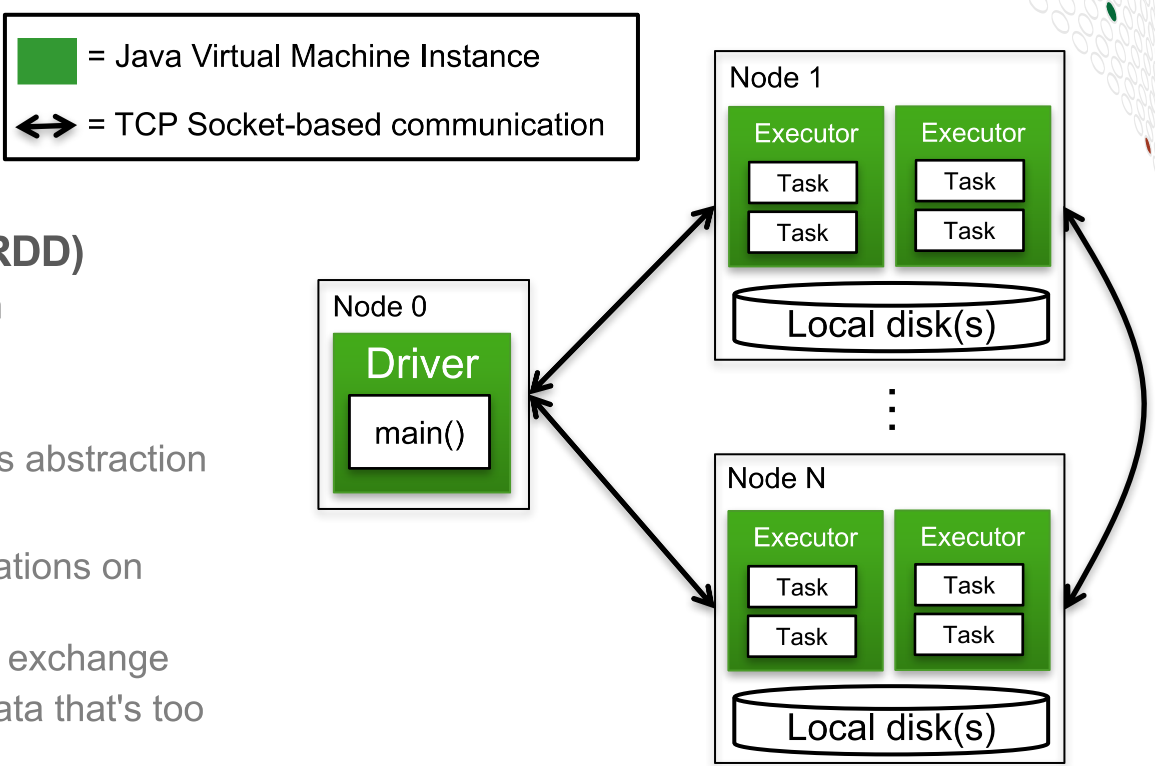
- **Gated on IO bandwidth, possibly interconnect as well**
  - Must write and read between map and reduce phases
  - Multiple iterations must write results in next time (e.g., new cluster centers)
- **No ability to persist reused data**
- **Must re-factor all computations as map then reduce (rinse and repeat?)**

What is Spark?

- **Newer (2014) analytics framework**
  - Originally from Berkeley AMPLab/BDAS stack, now Apache project
  - Native APIs in Scala. Java, Python, and R APIs available as well.
  - Many view as successor to Hadoop MapReduce. Compatible with much of Hadoop Ecosystem.
- **Aims to address some shortcomings of Hadoop MapReduce**
  - More programming flexibility – not constrained to one map, one reduce, write, repeat.
  - Many operations can be pipelined into a single in-memory task
  - Can "persist" intermediate data rather than regenerating every stage

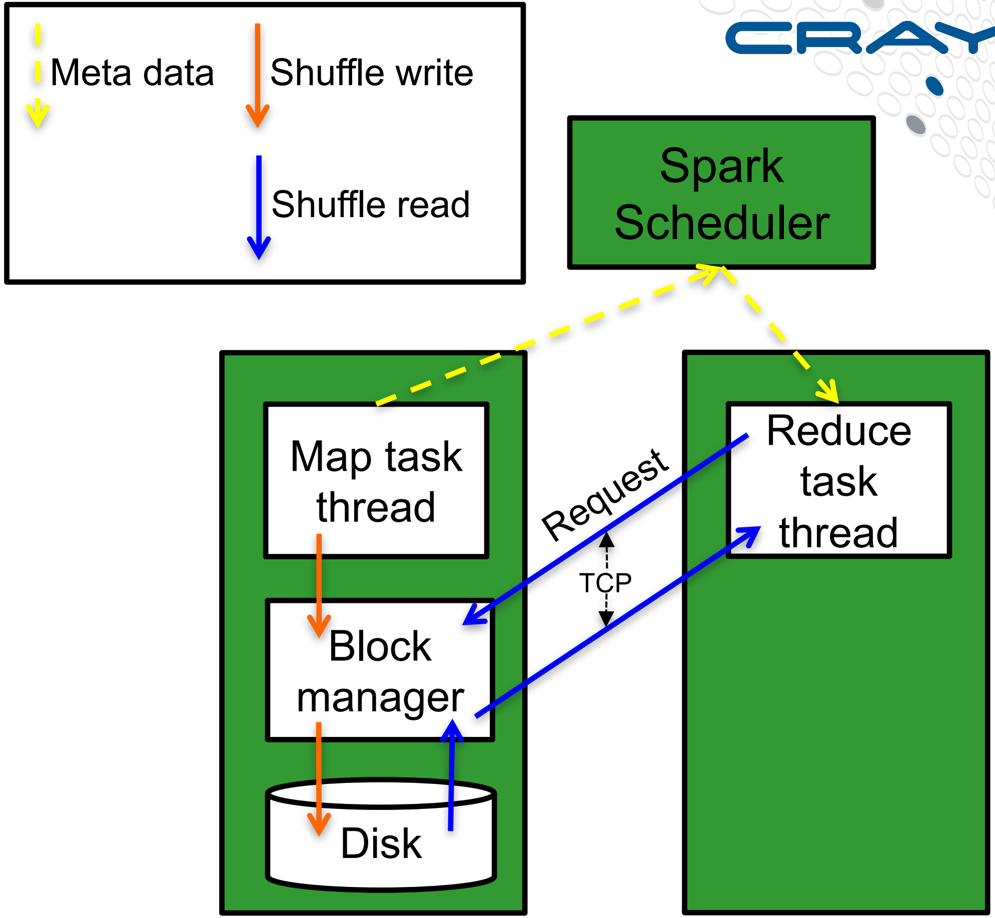
Spark Execution Model

- **Master-slave parallelism**
- **Driver (master)**
  - Executes main
  - Distributes work to executors
- **Resilient Distributed Dataset (RDD)**
  - Spark's original data abstraction
  - Partitioned amongst executors
  - Fault-tolerant via lineage
  - Dataframes/Datasets extend this abstraction
- **Executors (slaves)**
  - Lazily execute tasks (local operations on partitions of the RDD)
  - Global all-to-all shuffles for data exchange
  - Rely on local disks for spilling data that's too large, and storing shuffle data



Spark Communication Model (Shuffles)

- **All data exchanges between executors implemented via *shuffle***
  - Senders (“mappers”) send data to block managers; block managers write to disks, tell scheduler *how much* destined for each reducer
  - Barrier until all mappers complete shuffle writes
  - Receivers (“reducers”) request data from block managers *that have data for them*; block managers read and send



RDDs (and DataFrames/DataSets)

- **RDDs are original data abstraction of Spark**
  - DataFrames add structure to RDDs: named columns
  - DataSets add strong typing to columns of DataFrames (Scala and Java only)
  - Both build on the basic idea of RDDs
    - DataFrames were originally called SchemaRDDs
- **RDD data structure contains a description of the data, partitioning, and computation, but not the actual data … why?**
  - Lazy evaluation

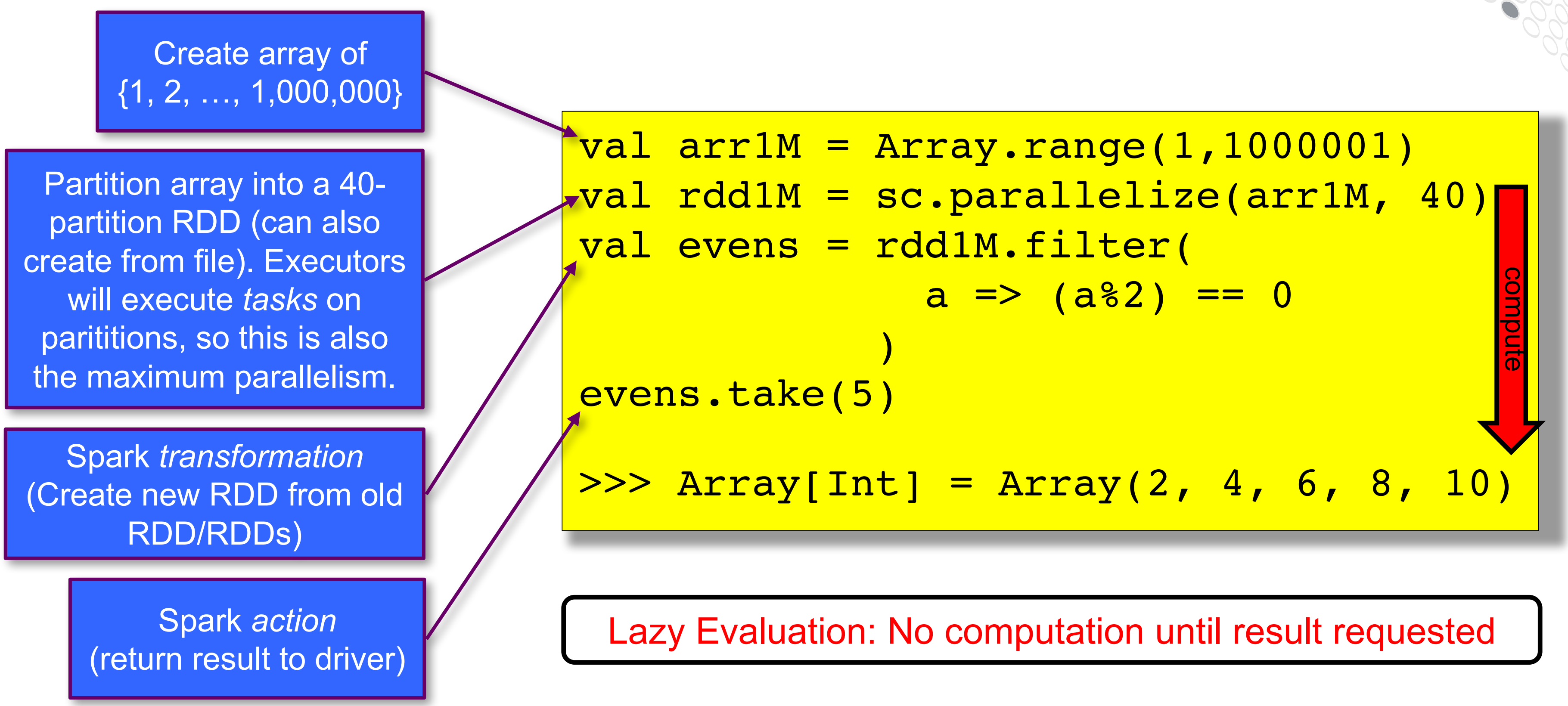
Lazy Evaluation and DAGs

- **Spark is lazily evaluated**
  - Spark operations are only executed when and if needed
  - Needed operations: produce a result for driver, or produce a parent of needed operation (recursive)
- **Spark DAG (Directed Acyclic Graph)**
  - Calls to transformation APIs (operations that produce a new RDD/DataFrame from one or more parents) just add a new node to the DAG, indicating data dependencies (parents) and transformation operation
  - Action APIs (operations that return data) trigger execution of necessary DAG elements
- **Example shortly…**

Tasks, Stages, and Pipelining

- **If an RDD partition's dependencies are on a single other RDD partition (or on *co-partitioned* data), the operations can be *pipelined* into a single *task***
  - **Co-partitioned**: all of the parent RDD partitions are co-located with child RDD partitions that need them
  - **Pipelined**: Operations can occur as soon as the local parent data is ready (no synchronization)
  - **Task**: A pipelined set of operations
  - **Stage**: Execution of same task on all partitions
- **Every stage ends with a shuffle, an output, or returning data back to the driver.**
  - Global barrier between stages. All senders complete shuffle write before receivers request data (shuffle read)

Spark Programming Model: Example



Create array of {1, 2, …, 1,000,000}

Partition array into a 40-partition RDD (can also create from file). Executors will execute *tasks* on parititions, so this is also the maximum parallelism.

Spark *transformation* (Create new RDD from old RDD/RDDs)

Spark *action* (return result to driver)

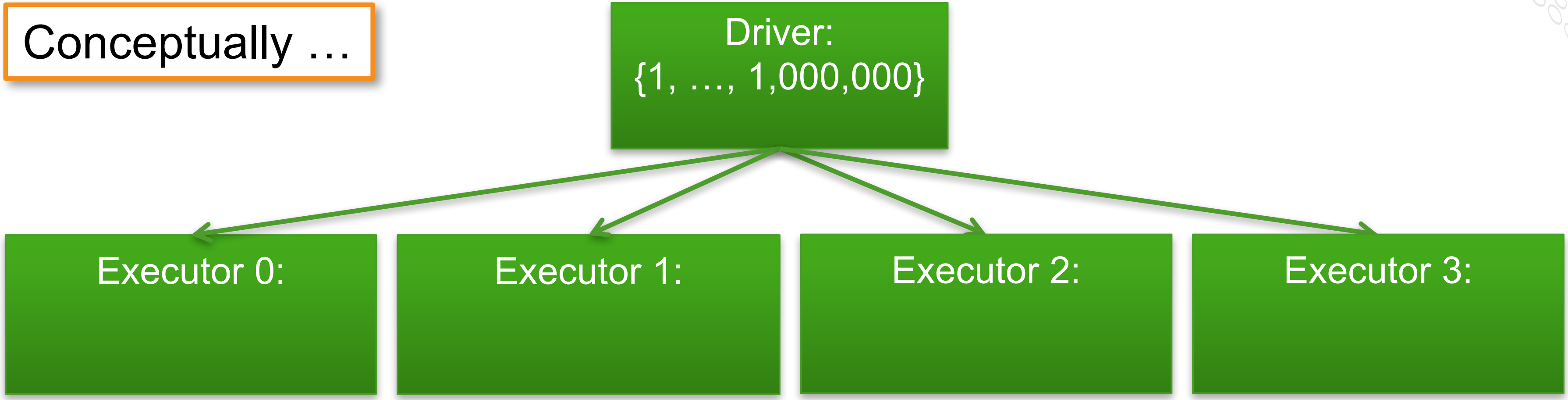
```
val arr1M = Array.range(1,1000001)
val rdd1M = sc.parallelize(arr1M, 40)
val evens = rdd1M.filter(
              a => (a%2) == 0
            )
evens.take(5)

>>> Array[Int] = Array(2, 4, 6, 8, 10)
```

Lazy Evaluation: No computation until result requested

Example: Line-by-line

Conceptually …



Driver:
{1, …, 1,000,000}

Executor 0:

Executor 1:

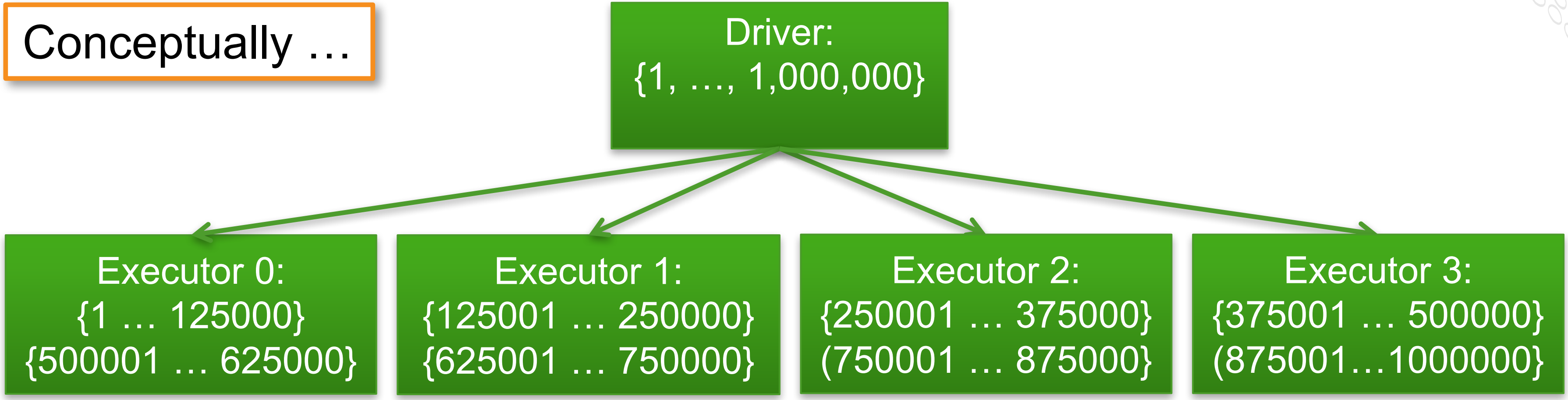
Executor 2:

Executor 3:

```
val arr1M = Array.range(1,1000001)
```

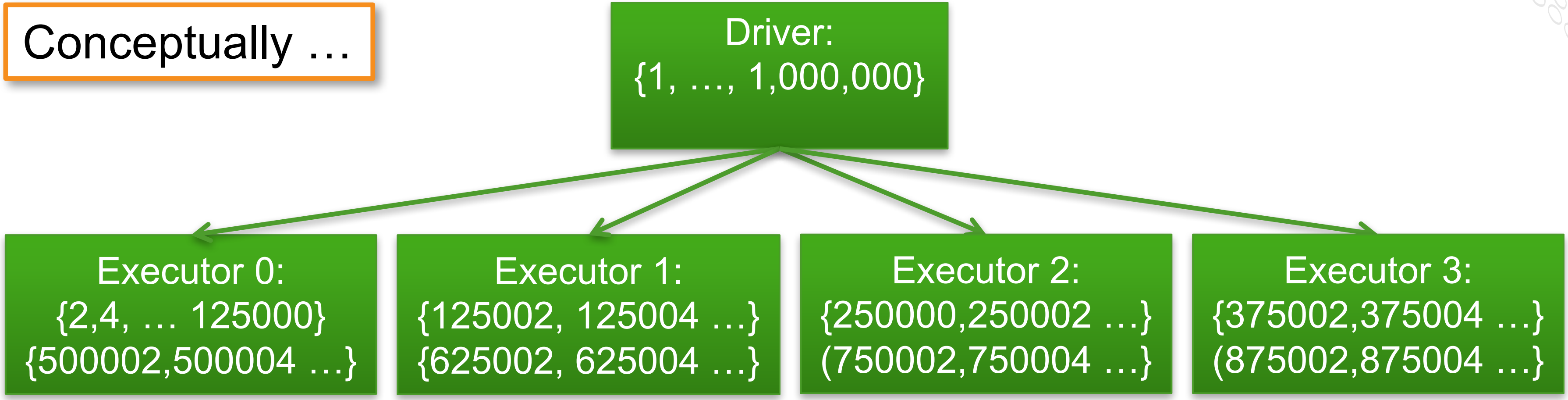
Example: Line-by-line

Conceptually …



```
val rdd1M = sc.parallelize(arr1M, 8)
```

Example: Line-by-line



Conceptually …

Driver:
{1, …, 1,000,000}

Executor 0:
{2,4, … 125000}
{500002,500004 …}

Executor 1:
{125002, 125004 …}
{625002, 625004 …}

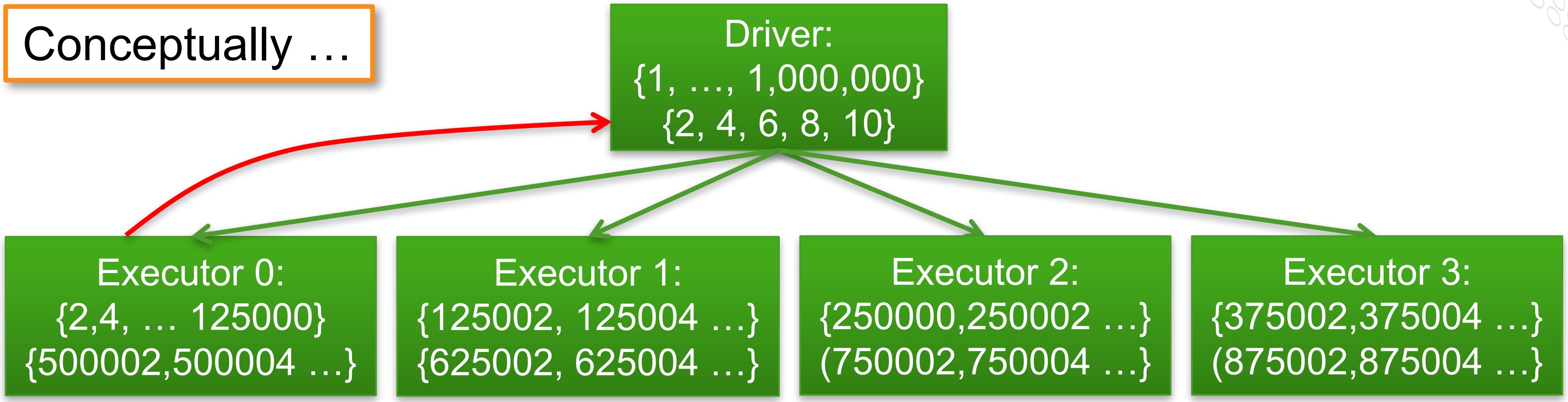
Executor 2:
{250000,250002 …}
(750002,750004 …}

Executor 3:
{375002,375004 …}
(875002,875004 …}

```
val evens = rdd1M.filter(a => a%2==0)
```

Example: Line-by-line

Conceptually …



Driver:
{1, …, 1,000,000}
{2, 4, 6, 8, 10}

Executor 0:
{2,4, … 125000}
{500002,500004 …}

Executor 1:
{125002, 125004 …}
{625002, 625004 …}

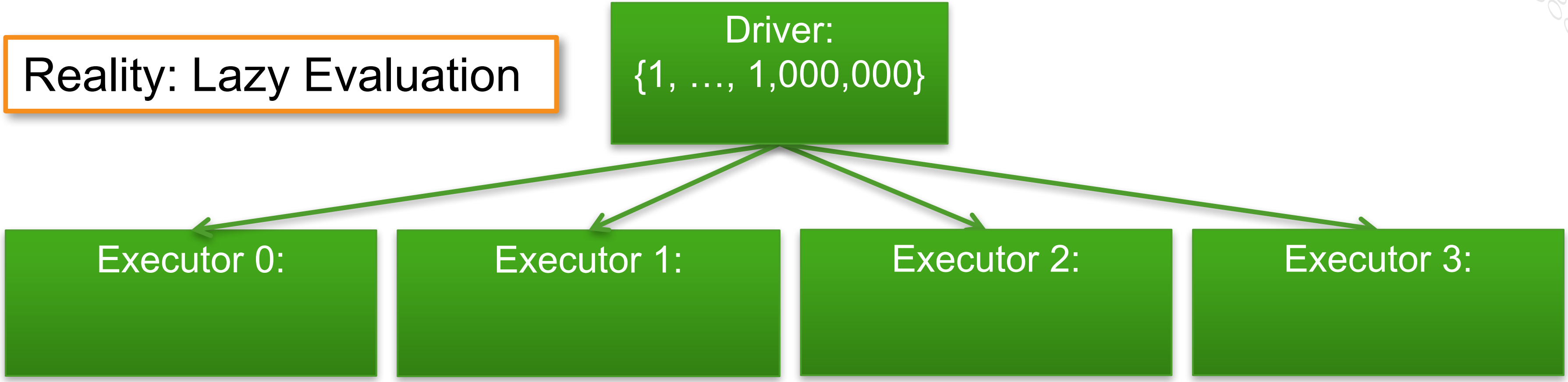
Executor 2:
{250000,250002 …}
(750002,750004 …}

Executor 3:
{375002,375004 …}
(875002,875004 …}

```
evens.take(5)
```

Example: Line-by-line

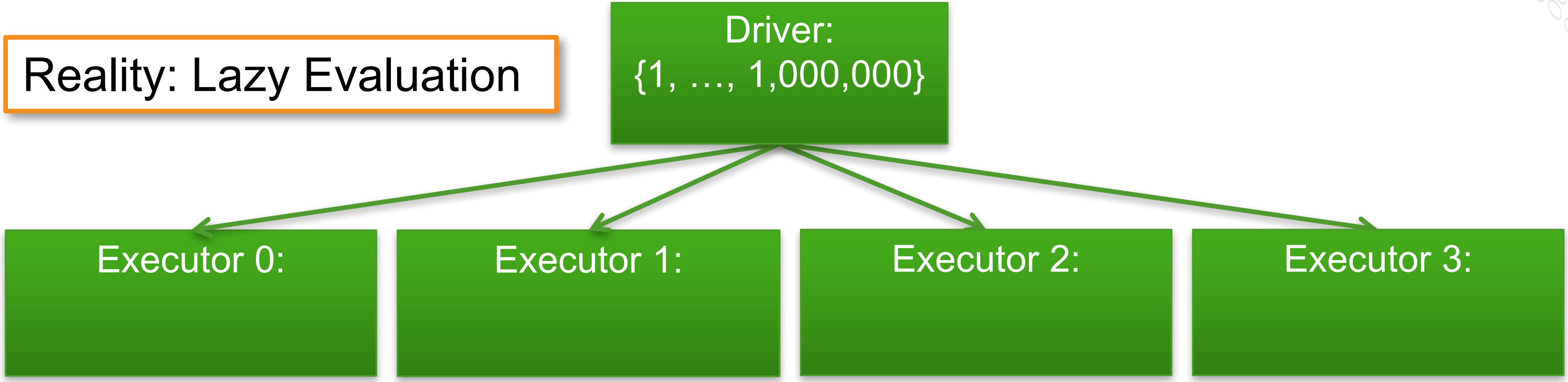
Reality: Lazy Evaluation



```
val arr1M = Array.range(1,1000001)
```

Example: Line-by-line

Reality: Lazy Evaluation

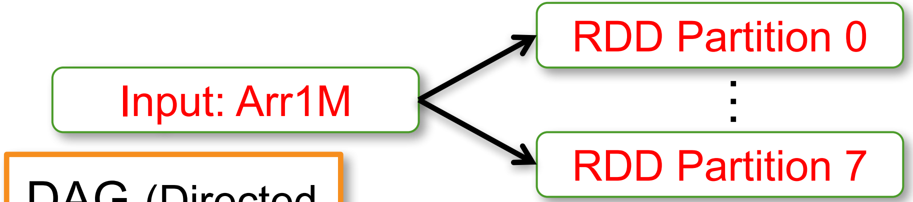

Driver:
{1, …, 1,000,000}

Executor 0:

Executor 1:

Executor 2:

Executor 3:


Input: Arr1M

RDD Partition 0

⋮

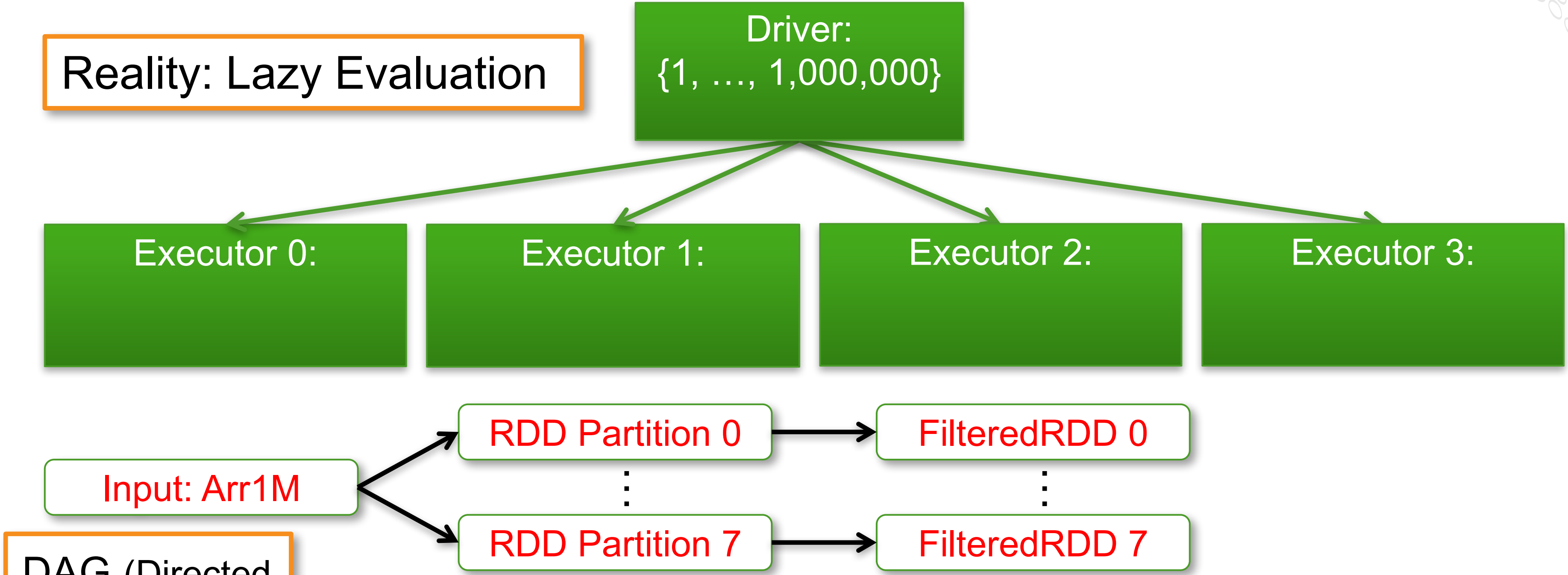
RDD Partition 7

DAG (Directed Acyclic Graph) schedule

```
val rdd1M = sc.parallelize(arr1M, 8)
```

Example: Line-by-line

Reality: Lazy Evaluation

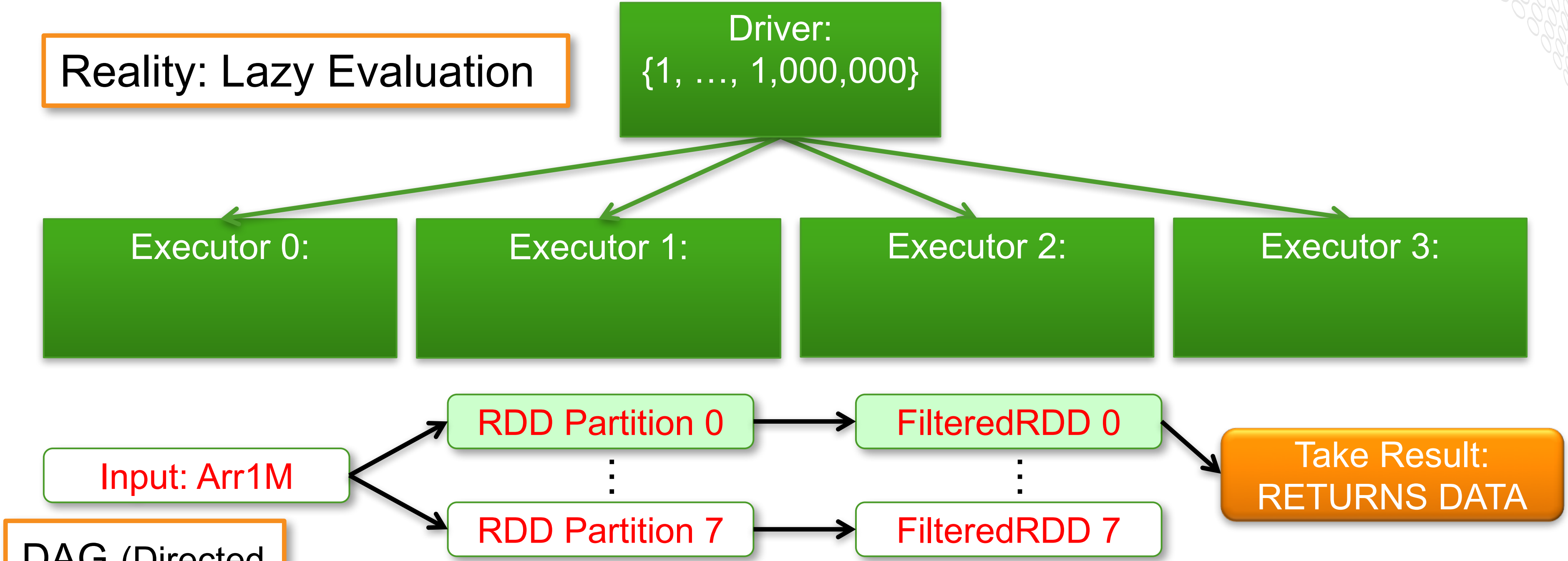


DAG (Directed Acyclic Graph) schedule

```
val evens = rdd1M.filter(a => a%2==0)
```

Example: Line-by-line

Reality: Lazy Evaluation



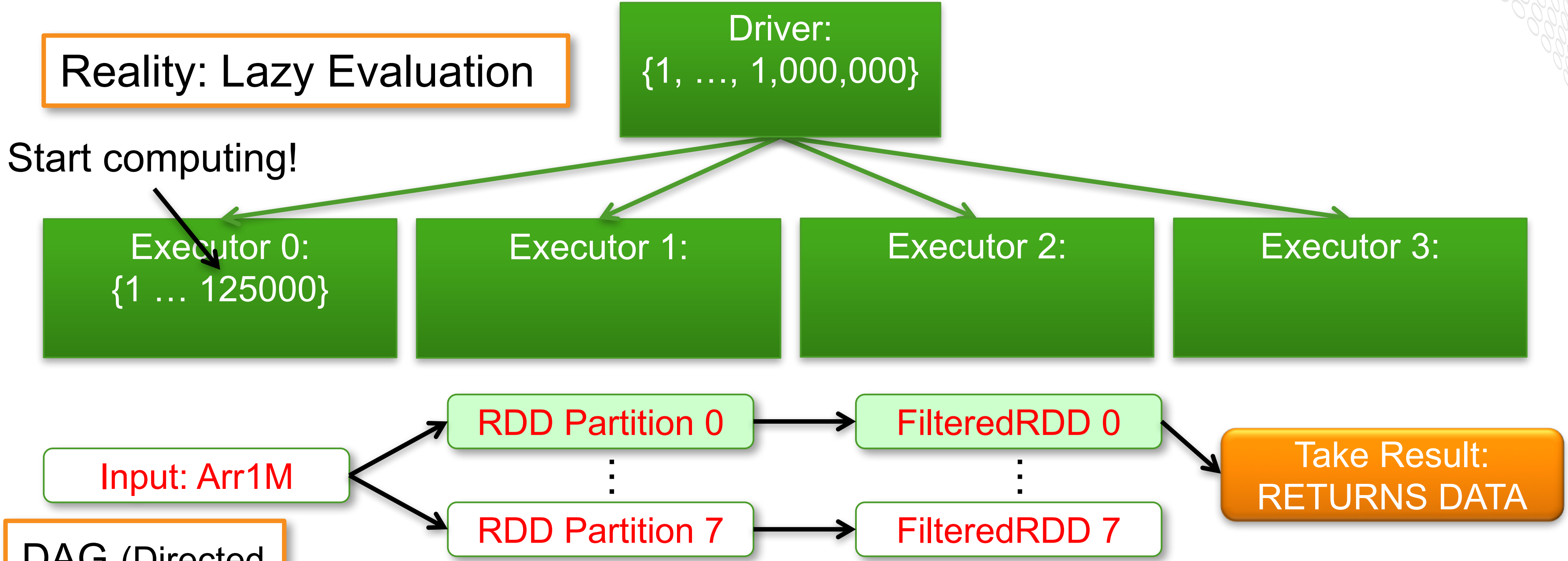
DAG (Directed Acyclic Graph) schedule

```
evens.take(5)
```

Example: Line-by-line

Reality: Lazy Evaluation

Start computing!

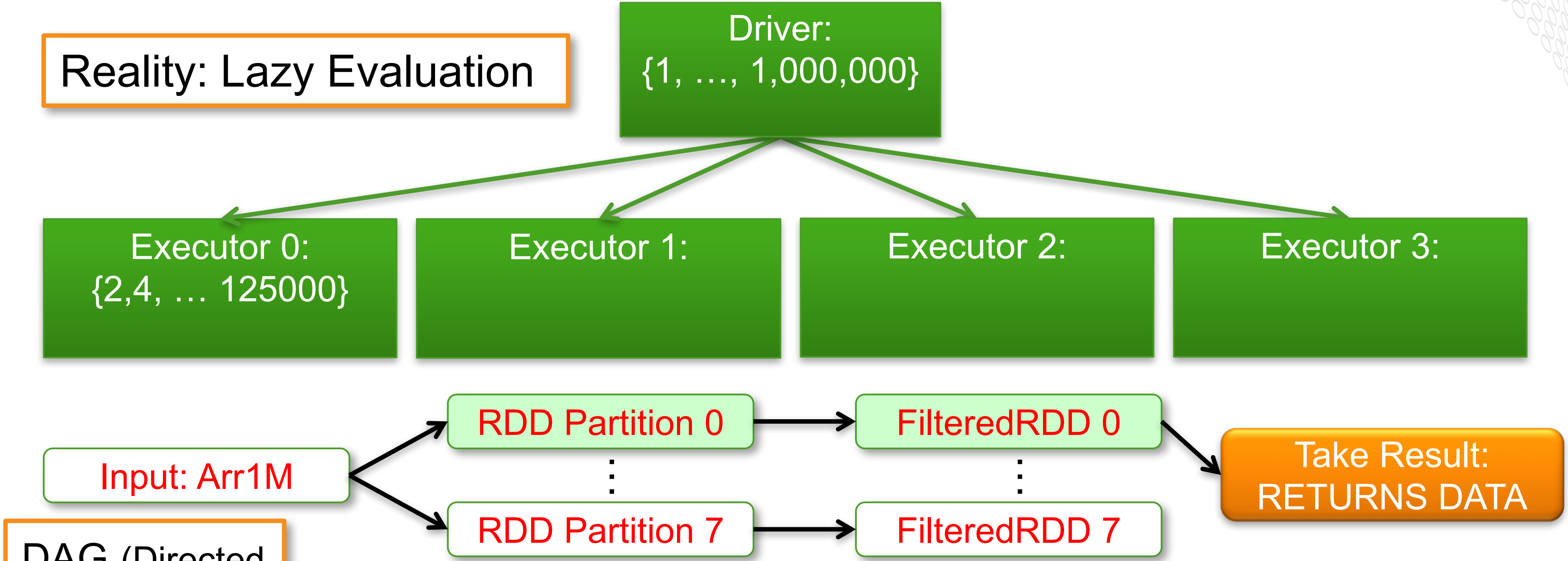


DAG (Directed Acyclic Graph) schedule

```
evens.take(5)
```

Example: Line-by-line

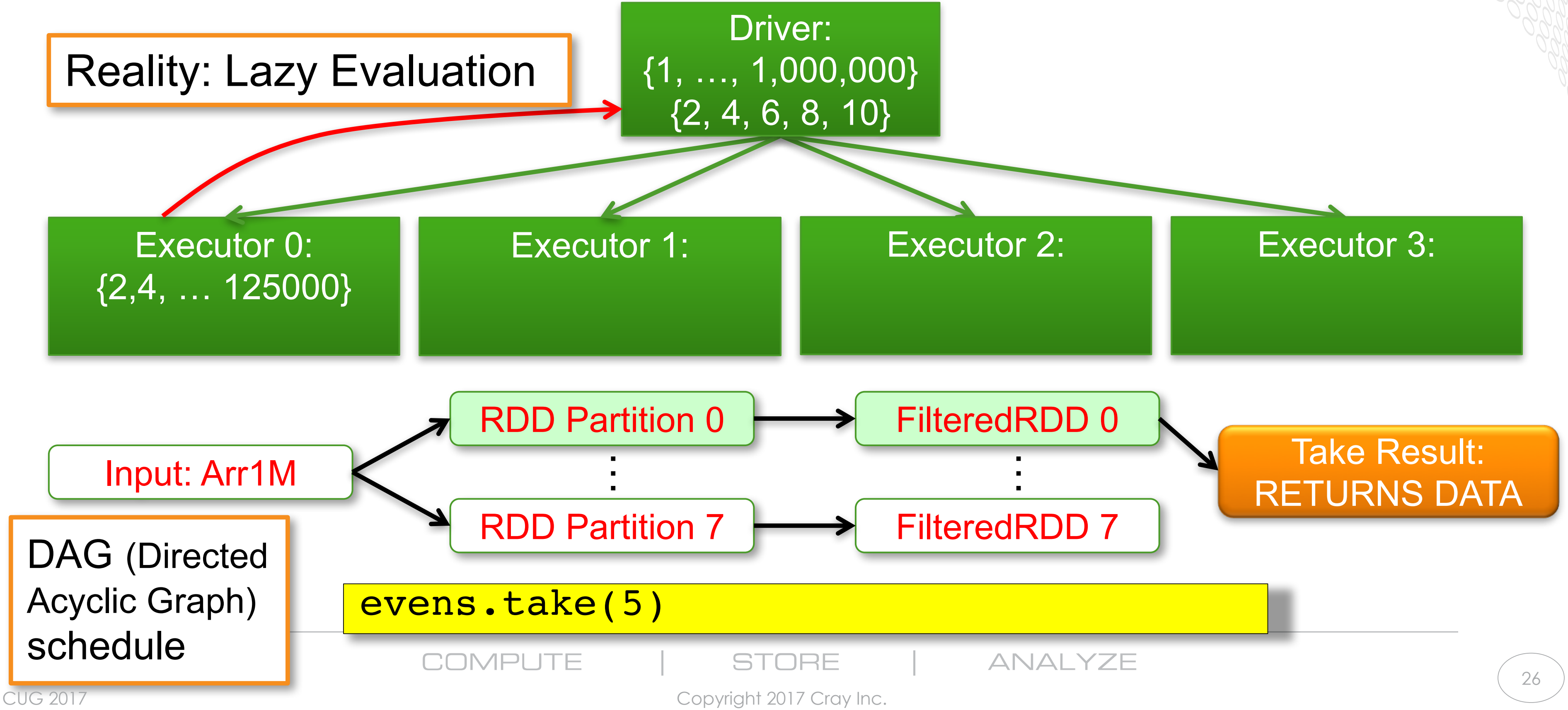
Reality: Lazy Evaluation



DAG (Directed Acyclic Graph) schedule

```
evens.take(5)
```

Example: Line-by-line



```
evens.take(5)
```

Wait a second …

- **How did Spark know that take() would only require data from one partition?**
  - What if filter() left fewer than 5 elements in the first partition?

Wait a second …

- **How did Spark know that take() would only require data from one partition?**
  - What if filter() left fewer than 5 elements in the first partition?
- **Answer … It didn't.**
  - Take is typically used to fetch a small initial piece of the data
  - Spark guesses that it will all be available in the first partition
  - If not, tries the first four partitions …
  - Then the first 16 …
  - Etc…

Modified example

Count returns the total size of an RDD

Reduce performs a reduction over the dataset, combining elements with the argument function.

```
val arr1M = Array.range(1,1000001)
val rdd1M = sc.parallelize(arr1M, 8)
val evens = rdd1M.filter(a => (a%2) == 0)
val firstFiveEvens = evens.take(5)
// How many evens?
val totalEvens = evens.count()
// Sum of evens
val evenSum = evens.reduce((a,b) => a+b)
```

- **Imagine we want to perform a number of actions on (i.e., return different data about) our filtered RDD.**
- **For each action, Spark computes all the DAG steps…**

Modified example

Count returns the total size of an RDD

Reduce performs a reduction over the dataset, combining elements with the argument function.

```
val arr1M = Array.range(1,1000001)
val rdd1M = sc.parallelize(arr1M, 8)
val evens = rdd1M.filter(a => (a%2) == 0)
val firstFiveEvens = evens.take(5)
// How many evens?
val totalEvens = evens.count()
// Sum of evens
val evenSum = evens.reduce((a,b) => a+b)
```

- **Problem: This means recomputing the filtered "evens" RDD three times – inefficient.**

Modified example

Persist tells Spark to keep the data in memory even after it is done with the action. Allows future actions to reuse without recomputing. Cache is synonym for default storage level (memory). Can also persist on disk, etc.

```
val arr1M = Array.range(1,1000001)
val rdd1M = sc.parallelize(arr1M, 8)
val evens = rdd1M.filter(a => (a%2) == 0)
evens.persist()   // or cache()
val firstFiveEvens = evens.take(5)
// How many evens?
val totalEvens = evens.count()
// Sum of evens
val evenSum = evens.reduce((a,b) => a+b)
```

- **Problem: This means recomputing the filtered "evens" RDD three times – inefficient.**
- **Solution: Persist the RDD!***

*Relies on immutability of val

Multi-stage Spark Example: Word Count

Load file

flatMap maps one value to (possibly) many, instead of one-to-one like map

groupByKey combines all key-value pairs with the same key (k, v1), …, (k,vn) into a single key-value pair (k, (v1, …, vn)).

Collect returns all elements to the driver

More efficient: replace group and sum with reduceByKey

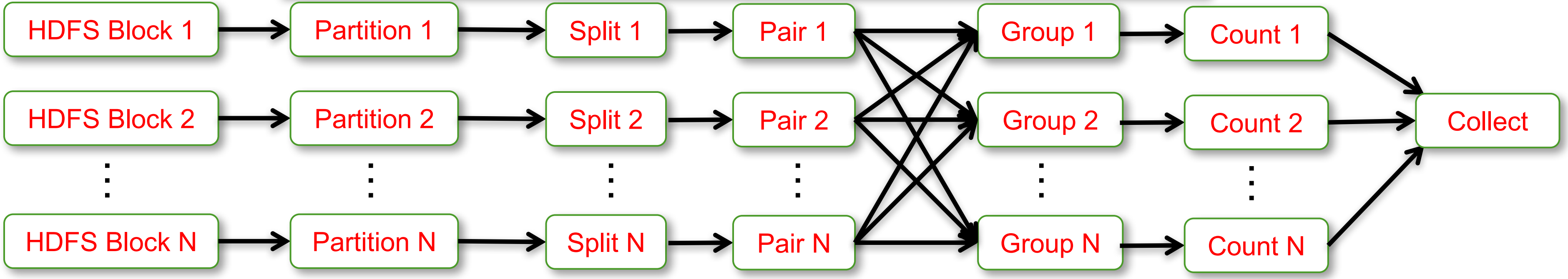
```
val lines = sc.textFile("mytext")
val words = lines.flatMap (
              line => line.split(" ")
           )
val wordKV = words.map(s => (s, 1))
val groupedWords = wordKV.groupByKey()
val wordCounts = groupedWords.map(
                   t => (t._1, t._2.sum)
                )
val counts = wordCounts.collect()
```

- Let's like at a simple example: computing the number of times each word occurs
  - Load a text file
  - Split it into words
  - Group same words together (all-to-all communication)
  - Count each word

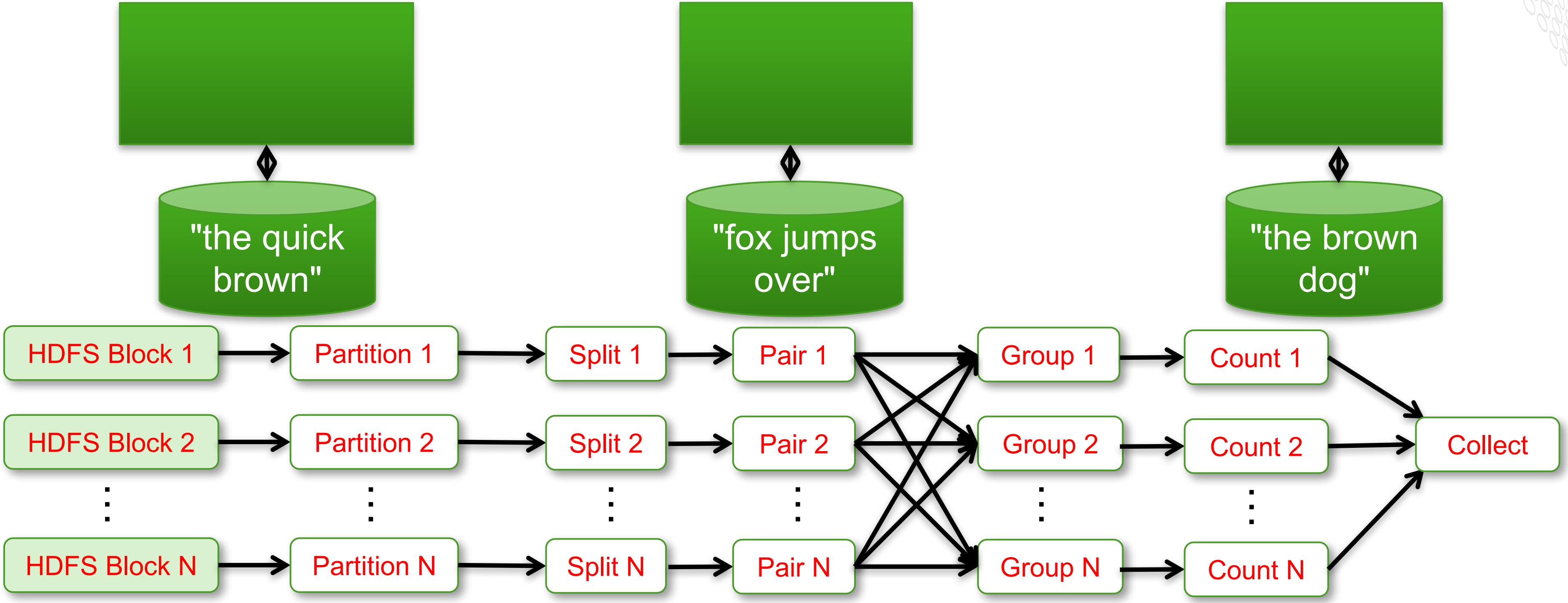
The Spark DAG

```
val lines = sc.textFile("mytext")
val words = lines.flatMap (
              line => line.split(" ")
            )
val wordKV = words.map(s => (s, 1))
val groupedWords = wordKV.groupByKey()
val wordCounts = groupedWords.map(
                   t => (t._1, t._2.sum)
                 )
val counts = wordCounts.collect()
```

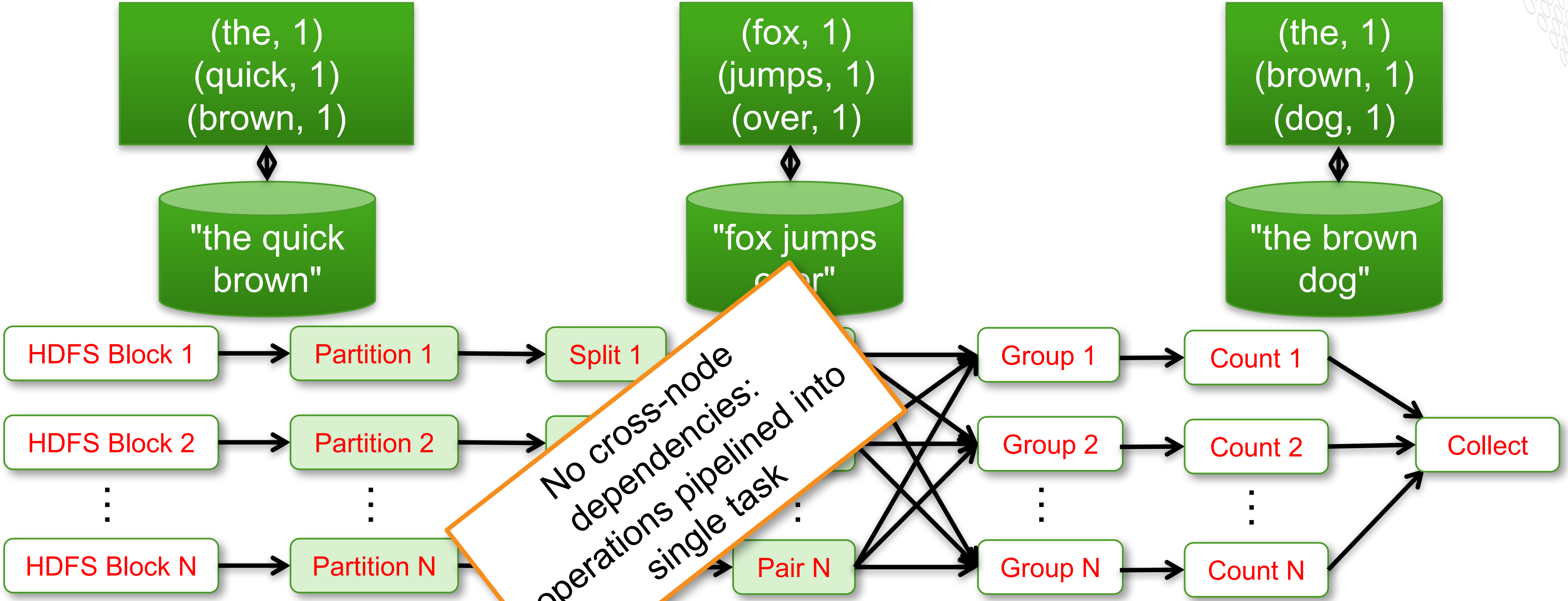
Execute!



Execution


"the quick brown"
"fox jumps over"
"the brown dog"
HDFS Block 1
HDFS Block 2
HDFS Block N
Partition 1
Partition 2
Partition N
Split 1
Split 2
Split N
Pair 1
Pair 2
Pair N
Group 1
Group 2
Group N
Count 1
Count 2
Count N
Collect

Execution



(the, 1)
(quick, 1)
(brown, 1)

"the quick brown"

(fox, 1)
(jumps, 1)
(over, 1)

"fox jumps over"

(the, 1)
(brown, 1)
(dog, 1)

"the brown dog"

HDFS Block 1 → Partition 1 → Split 1

HDFS Block 2 → Partition 2

⋮

HDFS Block N → Partition N → Pair N

Group 1 → Count 1

Group 2 → Count 2

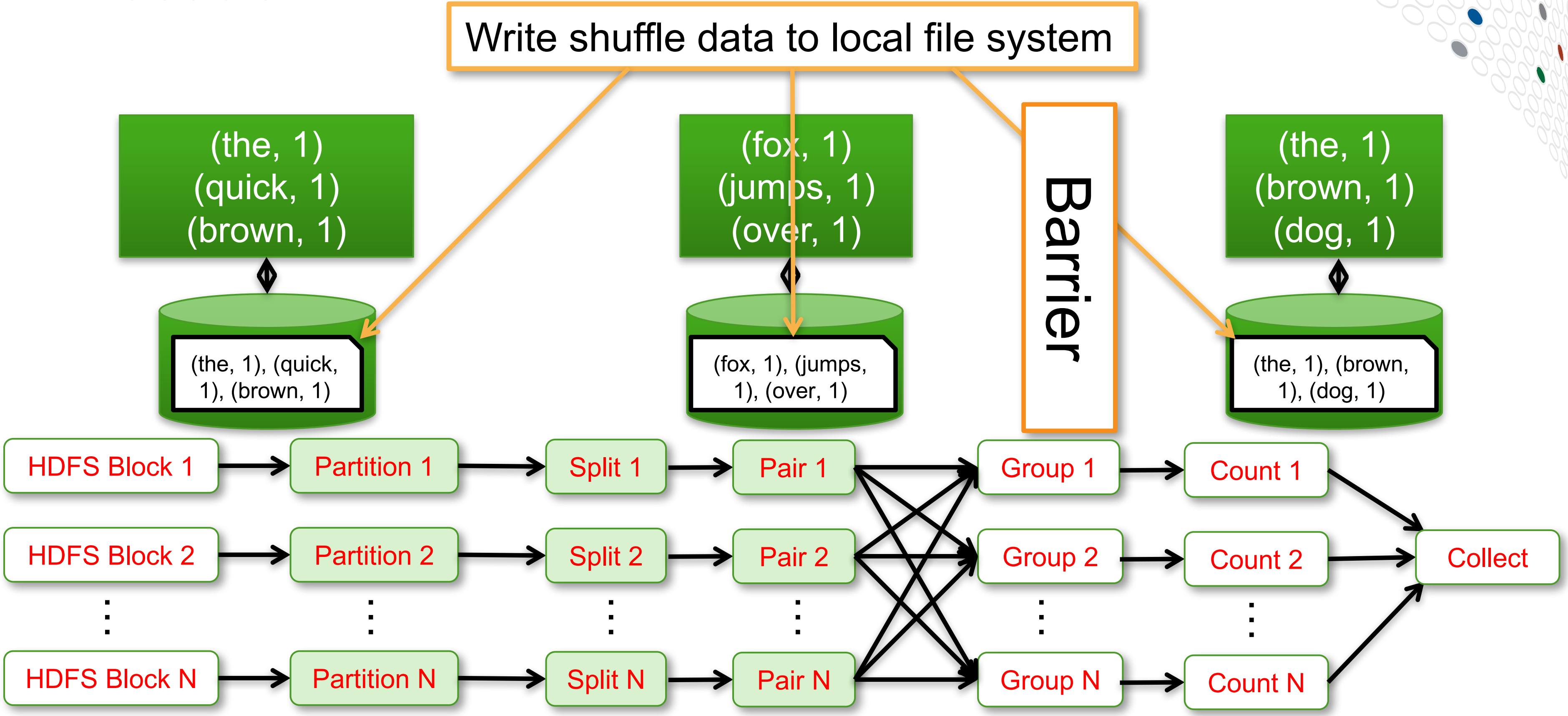
⋮

Group N → Count N

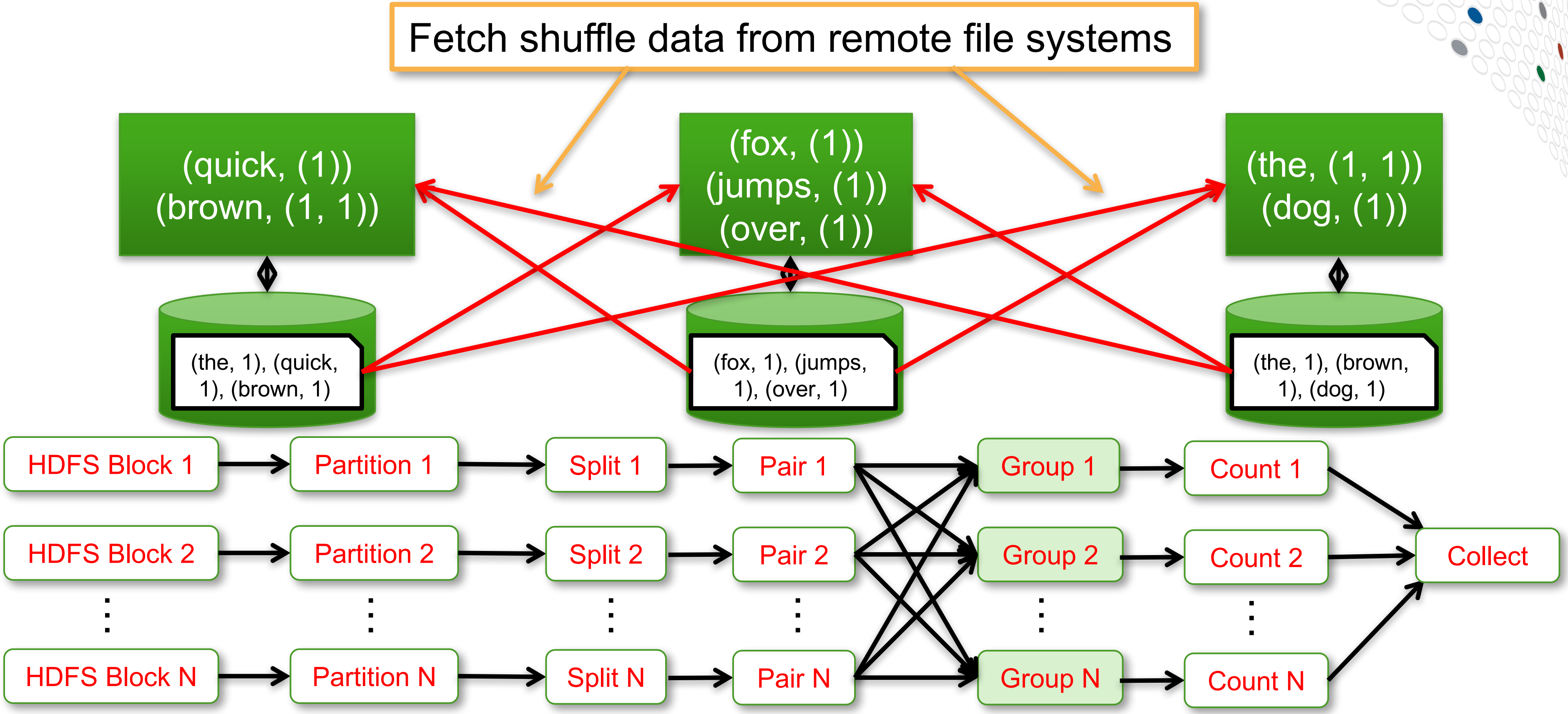
Collect

No cross-node dependencies: operations pipelined into single task

Execution


Write shuffle data to local file system
(the, 1)
(quick, 1)
(brown, 1)
(the, 1), (quick, 1), (brown, 1)
(fox, 1)
(jumps, 1)
(over, 1)
(fox, 1), (jumps, 1), (over, 1)
Barrier
(the, 1)
(brown, 1)
(dog, 1)
(the, 1), (brown, 1), (dog, 1)
HDFS Block 1
HDFS Block 2
HDFS Block N
Partition 1
Partition 2
Partition N
Split 1
Split 2
Split N
Pair 1
Pair 2
Pair N
Group 1
Group 2
Group N
Count 1
Count 2
Count N
Collect

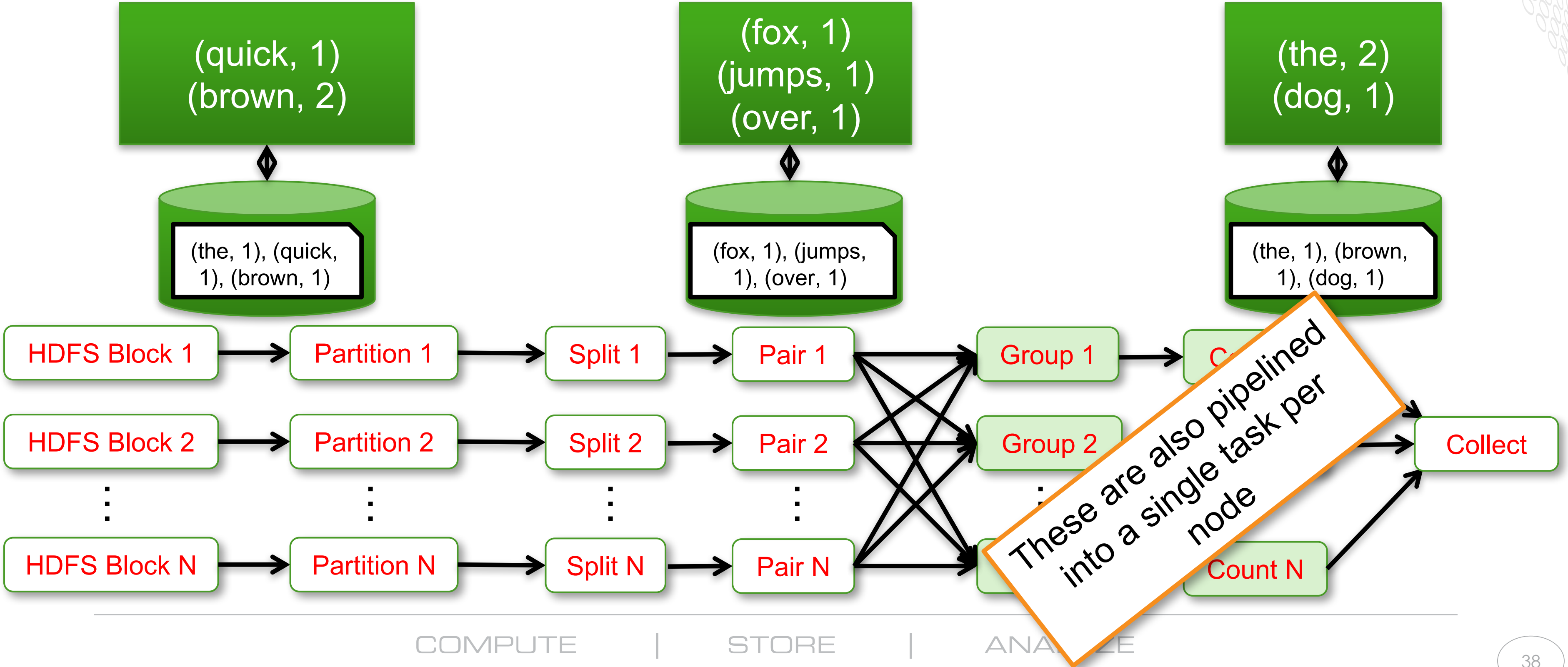
Execution



Fetch shuffle data from remote file systems

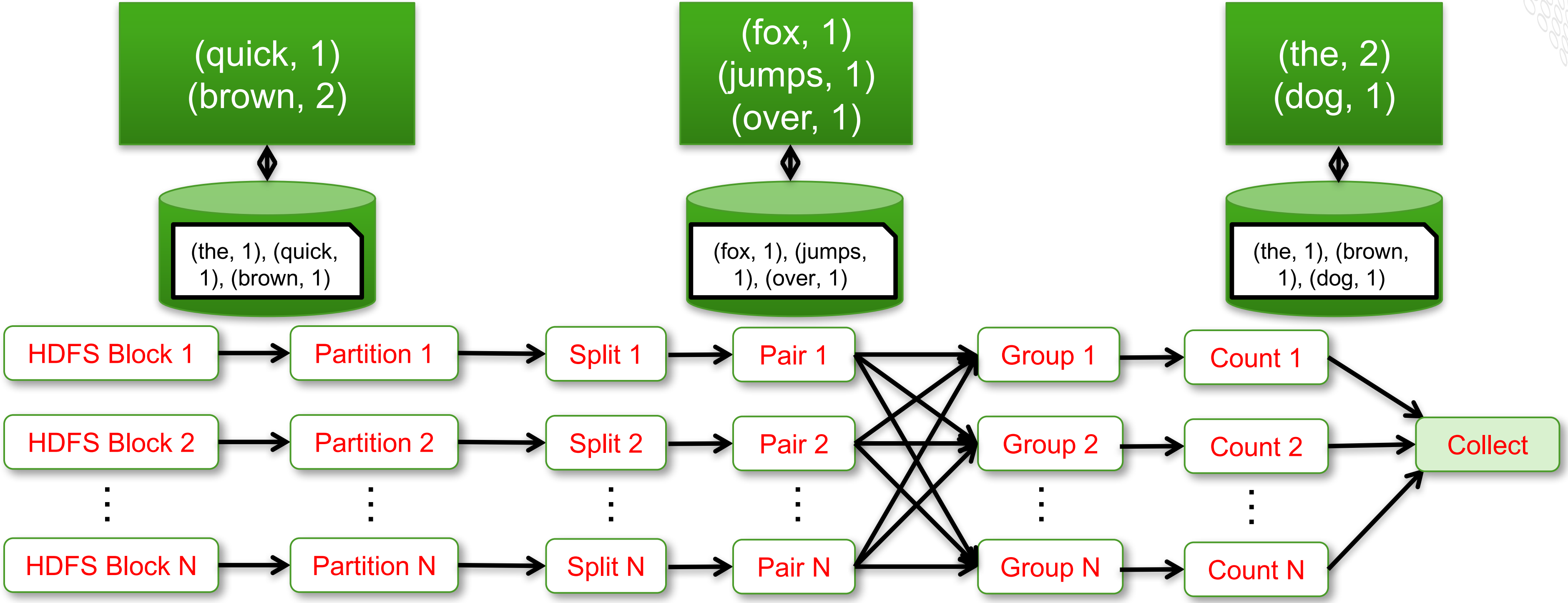
COMPUTE | STORE | ANALYZE

Execution



These are also pipelined into a single task per node

Execution



(quick, 1)
(brown, 2)

(fox, 1)
(jumps, 1)
(over, 1)

(the, 2)
(dog, 1)

(the, 1), (quick, 1), (brown, 1)

(fox, 1), (jumps, 1), (over, 1)

(the, 1), (brown, 1), (dog, 1)

HDFS Block 1

HDFS Block 2

HDFS Block N

Partition 1

Partition 2

Partition N

Split 1

Split 2

Split N

Pair 1

Pair 2

Pair N

Group 1

Group 2

Group N

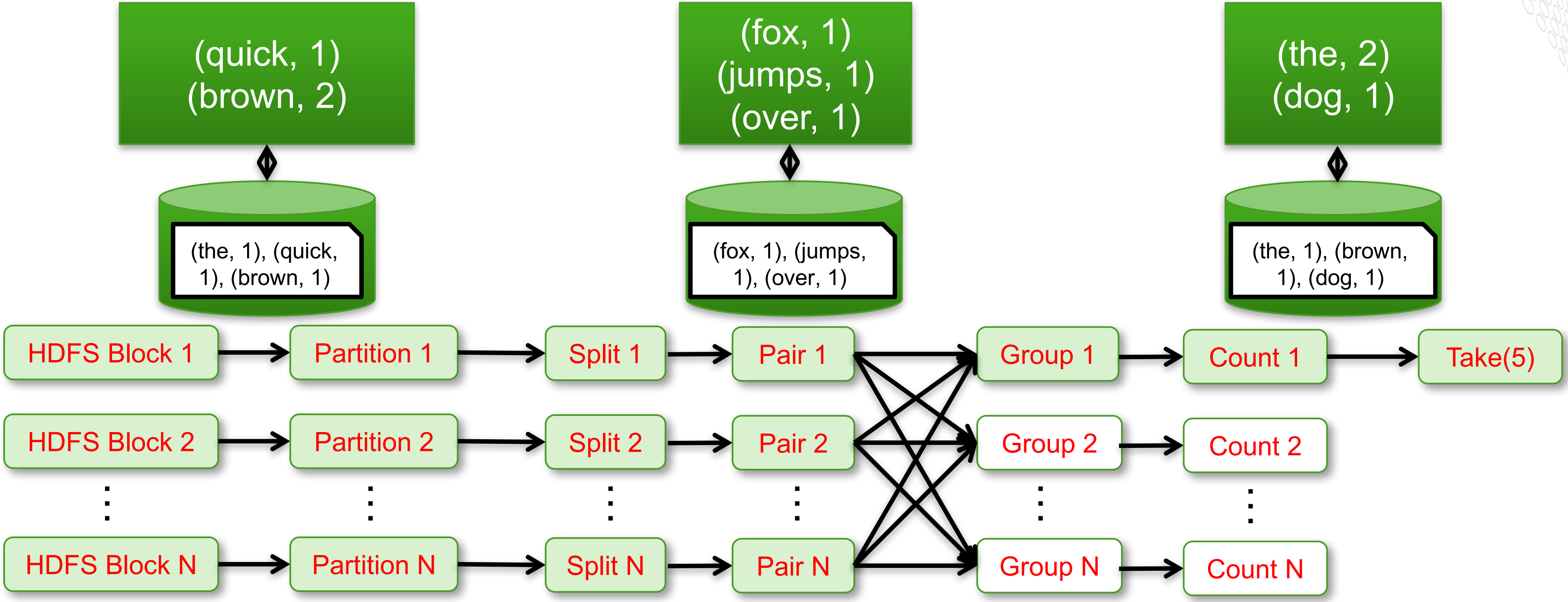
Count 1

Count 2

Count N

Collect

Execution


(quick, 1)
(brown, 2)
(fox, 1)
(jumps, 1)
(over, 1)
(the, 2)
(dog, 1)
(the, 1), (quick, 1), (brown, 1)
(fox, 1), (jumps, 1), (over, 1)
(the, 1), (brown, 1), (dog, 1)
HDFS Block 1
Partition 1
Split 1
Pair 1
Group 1
Count 1
Take(5)
HDFS Block 2
Partition 2
Split 2
Pair 2
Group 2
Count 2
HDFS Block N
Partition N
Split N
Pair N
Group N
Count N

Spark on Cray XC

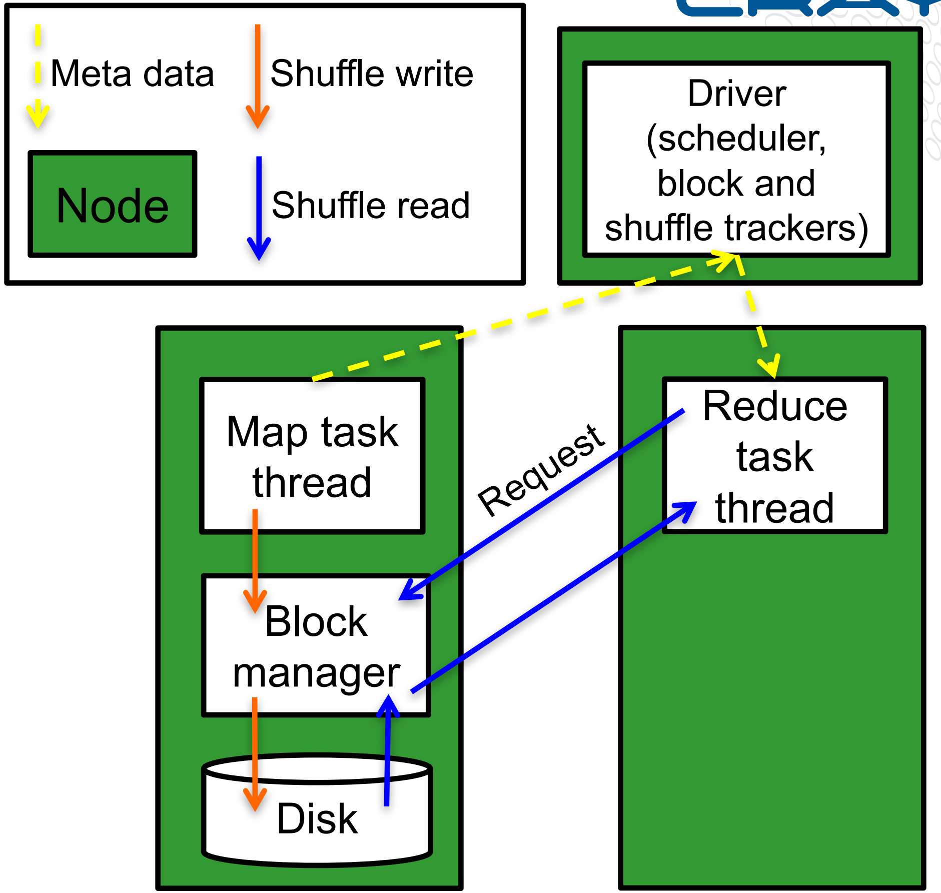
Spark on XC: Setup options

- **Cluster Compatibility Mode (CCM) option**
  - Set up and launch standalone Spark cluster in CCM mode, run interactively from mom node, or submit batch script
  - Exact details vary based on CLE version and workload manager
    - An example recipe can be found in:

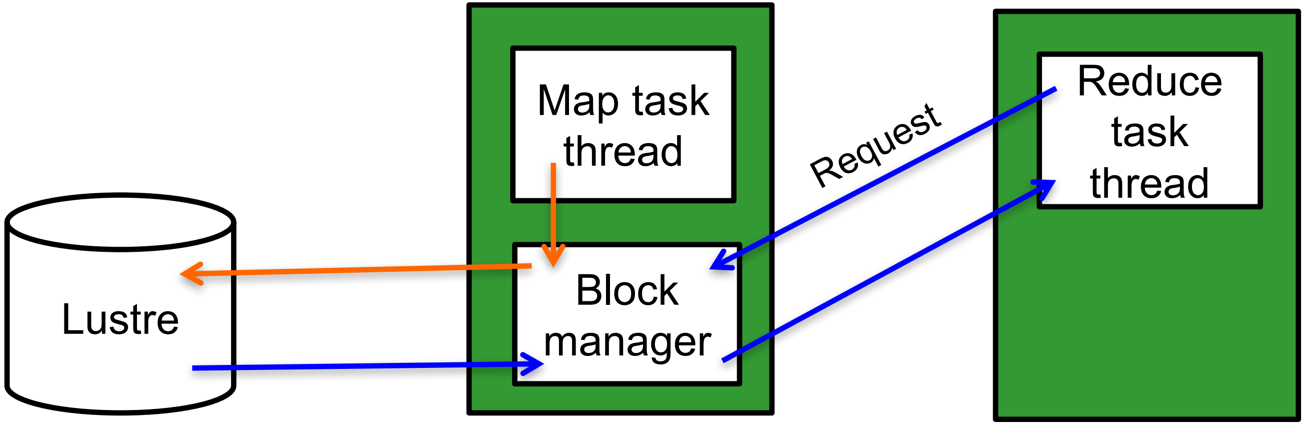
      "Experiences Running and Optimizing the Berkeley Data Analytics Stack on Cray Platforms", Maschhoff and Ringenburg, CUG 2015
- **Shifter option**
  - Shifter containerizer (think "Docker for XC") developed at NERSC
  - Acquire node allocation
    - Run master image on one node
    - Interactive image on another (or login)
    - Worker images on rest
  - Cray's analytics on XC product (in beta testing) uses this approach
- **Challenge: Lack of local storage for Spark shuffles and spills**

Reminder: Spark Shuffle – Standard Implementation

- Senders (“mappers”) send data to block managers; block managers write to **local disks**, tell driver how much destined for each reducer
- Barrier until all mappers complete shuffle writes
- Receivers (“reducers”) request data from block managers that have data for them; block managers read from **local disk** and send
- Key assumption: large, fast local block storage device(s) available on executor nodes

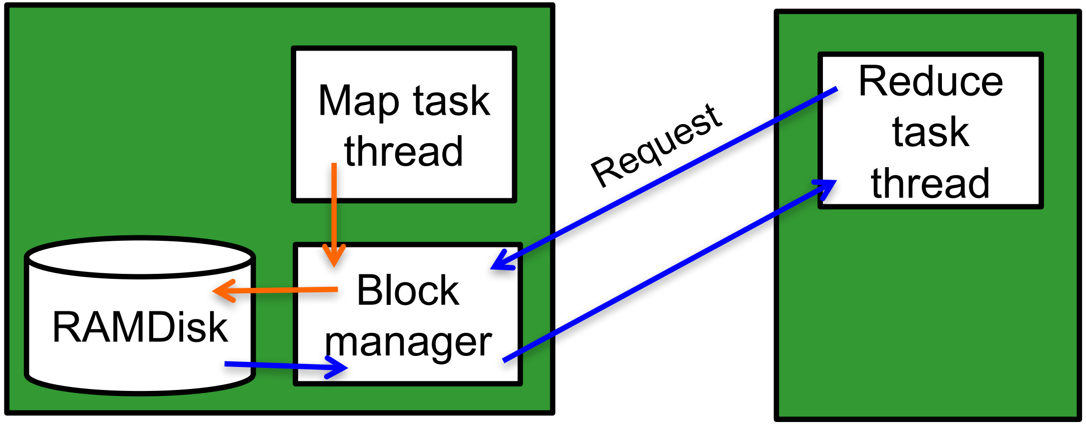


Shuffle on XC – Version 1



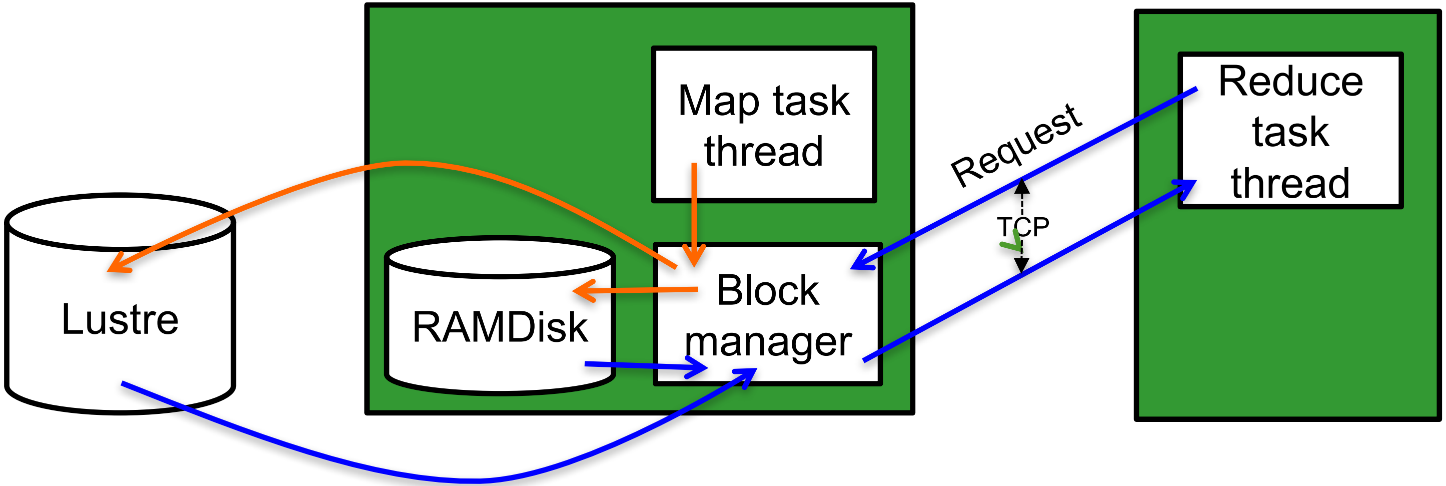
- **Problems: No local disk on standard XC40**
- **First try: Write to lustre instead**
  - Biggest Issue: Poor file access pattern for lustre (lots of small files, constant opens/closes). Creates a major bottleneck on Lustre Metadata Server (MDS).
  - Issue 2: Unnecessary extra traffic through network

Shuffle on XC – Version 2



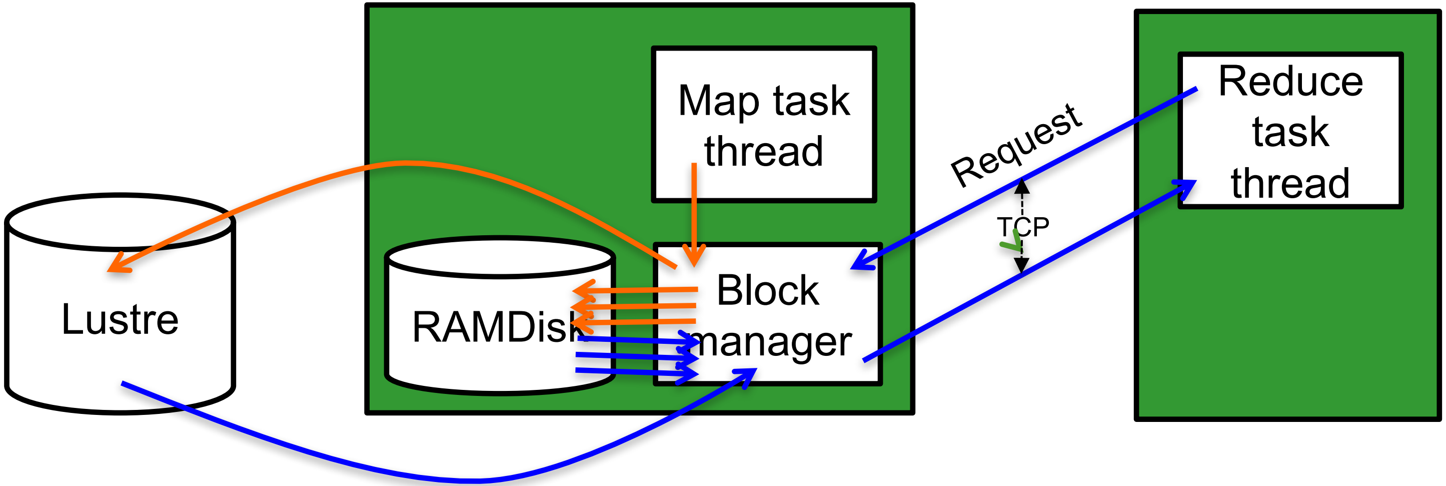
- **Second try: Write to RAMDisk**
  - Much faster, but …
  - Issues: Limited to lessor of: 50% of node DRAM or unused DRAM; Fills up quickly; takes away memory that could otherwise be allocated to Spark
  - Spark behaves unpredictably when it's local scratch space fills up (failures not always simple to diagnose)

Shuffle on XC – Version 3



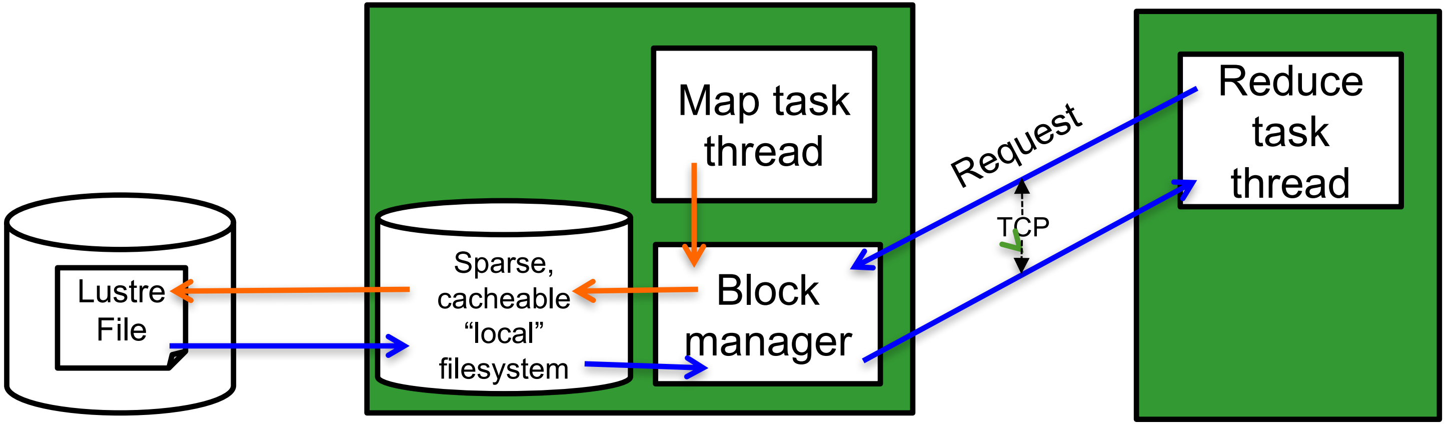
- **Third try: Write to RAMDisk and Lustre**
  - Set local directories to RAMdisk and lustre (can be list)
  - Initially fast and keeps working when RAMDisk full
  - Issues: Slow once RAMDisk fills; Round robin between directories (no bias towards faster RAM)

Shuffle on XC – Version 3



- **Third try: Write to RAMDisk and Lustre**
  - Set local directories to RAMdisk and lustre (can be list)
  - Initially fast and keeps working when RAMDisk full
  - Issues: Slow once RAMDisk fills; Round robin between directories (no bias towards faster RAM), *but can specify multiple RAM directories*

Shuffle on XC – with Shifter PerNodeCache



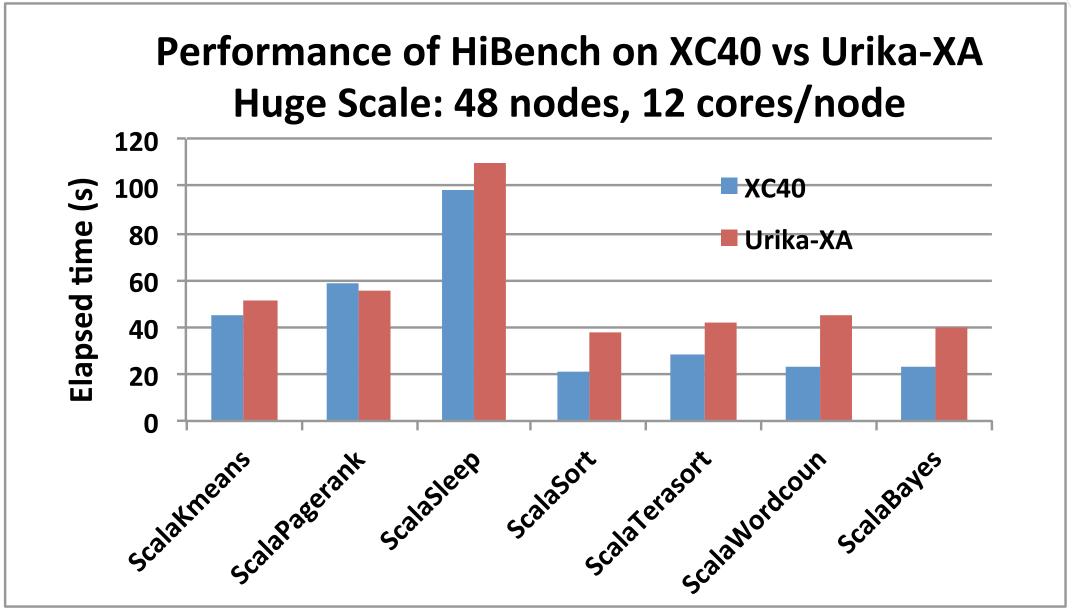
- **Shifter implementation: Per-node loopback file system**
  - NERSC's Shifter containerization (in Cray CLE6) provides optional loopback-mounted per-node temporary filesystem
  - Local to each node – fully cacheable
  - Backed by a single sparse file on Lustre – greatly reduced MDS load, plenty of capacity, doesn't waste space
  - Performance comparable to RAMDisk, without capacity constraints (Chaimov et al, CUG '16)
- **Cray's Analytics on XC project (in beta) will ship as a Shifter image, and use this approach**

Other Spark Configurations

- **Many config parameters … some of the more relevant:**
  - **spark.shuffle.compress**: Defaults to true. Controls whether shuffle data is compressed. In many cases with fast interconnect, compression and decompression overhead can cost more than the transmission time savings. However, can still be helpful if limited shuffle scratch space.
  - **spark.locality.wait**: Defaults to 3 (seconds). How long to wait for available resources on a node with data locality before trying to execute tasks on another node. Worth playing around with - decrease if seeing a lot of idle executors. Increase if seeing poor locality. (Can check both in history server.) Do not set to 0!

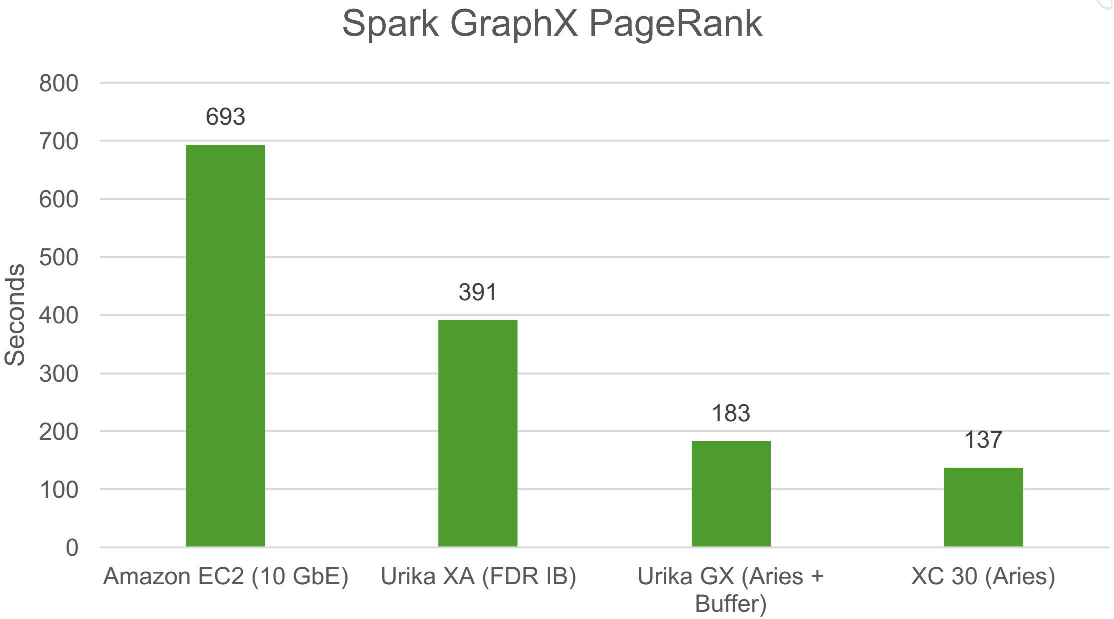
Spark Performance on XC: HiBench

- **Intel HiBench**
  - Originally MapReduce, Spark added in version 4
- **Compared performance with Urika XA system**
  - XA: FDR Infiniband, XC40: Aries
  - Both: 32 core Haswell nodes
  - XA: 128 GB/node, XC40: 256 GB/node (problems fit in memory on both)
- **Similar performace on Kmeans, PageRank, Sleep**
- **XC40 faster for Sort, TeraSort, Wordcount, Bayes**



Spark Performance on XC: GraphX

- **GraphX PageRank**
  - 20 iterations on Twitter dataset
  - Interconnect sensitive
- **GX has slightly higher latency and lower peak TCP bandwidth than XC due to buffer chip**



Spark on KNL

- **Cray and Intel have recently started a collaboration to investigate and improve Spark on KNL performance**
  - Java and Spark currently run
  - Performance vs Skylake varies from 20% slower to >4x slower
  - “Typical” benchmarks at larger sizes ~3x slower than a dual-socket Skylake node
    - Still early … just starting to benchmark and profile.
  - Looking at issues, profiling, attempting to identify causes and potential solutions.

Early findings and tips

- **Lots of skinny executors work better than fewer fatter executors**
  - On Xeon-based nodes this is not necessarily the case – fat often works nearly as well or occasionally better
  - On KNL, though, often find best results with 1-2 cores per executor
    - Make sure to adjust executor memory appropriately – all about memory/core
    - E.g., 64 executors with 1 core and 2GB each, rather than 1 executor with 64 cores and 128 GB
  - Skinny executors have better memory locality
  - Skinny executors also have less JVM overhead
  - JVM has issues scaling to many threads, e.g., https://issues.scala-lang.org/browse/SI-9823 (cache thrashing with isInstanceOf)
- **Hyperthreading generally not helpful for Spark (on either Xeon or Xeon Phi)**

Early findings and tips

- **Limit GC parallelism from JVM**
  - E.g., -XX:+UseParallelOldGC -XX:ParallelGCThreads=<N>, where N ≤ available threads/# executors
  - Especially important with lots of skinny JVMs
    - Otherwise each JVM will try to grab 5/8 total threads
- **MCDRAM configured as cache works best with Spark**
  - Seeing ~43% of accesses coming from MCDRAM, ~11% directly from DDR
  - Currently no ability in JVM to take advantage of MCDRAM in flat mode

Running Preinstalled Spark on Cori

- Login to your training accounts
- Allocate Haswells, e.g.,
  - salloc --reservation=CUG2C -N 5 -p regular -C haswell -t 60 -A ntrain
- module load spark
- start-all.sh
- Scala shell: spark-shell
- Python shell: pyspark
- Submit spark application: spark-submit
- When done: stop-all.sh

WordCount demo

- **spark-shell --executor-cores 12**
  - val lines = sc.textFile("/global/cscratch1/sd/mikeri/enron_mail/words.*.json.gz")
  - val words = lines.flatMap (line => line.split(" "))
  - val wordKV = words.map(s => (s, 1))
  - val wordCounts = wordKV.reduceByKey((a, b) => a+b)
  - wordCounts.persist()
  - wordCounts.count()
  - wordCounts.take(5)
  - val sorted=wordCounts.sortBy(-_._2)
  - sorted.take(5)

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, REVEAL, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Q&A

Michael Ringenburg
<mikeri@cray.com>

Tutorial: Anaconda Python and Dask

Michael Ringenburg, mikeri@cray.com
Principal Engineer, Analytics R&D, Cray Inc

CUG 2017 CAFFEINATED COMPUTING

Redmond, Washington May 7-11, 2017

Install Anaconda

- Run the following in your home directory:
  - wget https://repo.continuum.io/archive/Anaconda3-4.3.1-Linux-x86_64.sh
  - chmod +x ./Anaconda3-4.3.1-Linux-x86_64.sh
  - ./Anaconda3-4.3.1-Linux-x86_64.sh
  - Follow the prompts
- On cori, it is preinstalled for you:
  - module load python/3.5-anaconda (or 2.7-anaconda)
  - conda config --add envs_dirs $HOME/.conda_env
- Cray's analytics on XC product (in beta) includes anaconda in the analytics shifter container

Using the Conda environment manager

- **Create a new conda environment with conda create**
  - E.g., create an environment with Python 3.5 and biopython:

```
conda create --name bio biopython python=3.5
```

- **Activate your environment:**

```
source activate bio
(bio) mikeri@cori07:~>
```

- **Python 3.5 with biopython will now be your default python:**

```
(bio) mikeri@cori07:~> python
Python 3.5.3 |Continuum Analytics, Inc.| (default, Mar  6 2017, 11:58:13)
>>> import Bio
>>> from Bio.Seq import Seq
>>> my_seq = Seq('CATGTAGACTAG')
>>> my_seq.translate()
Seq('HVD*', HasStopCodon(ExtendedIUPACProtein(), '*'))
```

More Conda commands

- Deactivate an environement: **source deactivate**
- Get rid of an environment: **conda remove**
- Clone an environment: **conda clone**
- List environments: **conda info --envs**
- Find available packages: **conda search**
- List packages: **conda list**
- Add package to current environment: **conda install**
- More in docs: https://conda.io/docs/index.html

Using Anaconda with PySpark

- Start up Spark cluster (see previous slides)
- Activate your conda environment

```
source activate bio
```

- Set PYSPARK_PYTHON to point to environment python

```
export PYSPARK_PYTHON=$CONDA_PREFIX/bin/python
```

- Run pyspark

```
pyspark
>>> import Bio
```

Using PySpark and Anaconda to Complement a Set of Orchid Genomes

- **In pyspark**
  - import Bio
  - from Bio import SeqIO
  - sequences = [seq_record.seq for seq_record in SeqIO.parse("/global/cscratch1/sd/mikeri/ls_orchid.fasta", "fasta")]
  - seqRDD = sc.parallelize(sequences, 100)
  - complementRDD = seqRDD.map(lambda seq: seq.complement())
  - seqRDD.take(2)
  - complementRDD.take(2)
  - complementRDD.count()
  - seqRDD.count()

Anaconda and PySpark on XC Demo

Setting up a dask.distributed Cluster, continued

- Add $HOME/anaconda3/bin to PATH, or load Cori anaconda 3.5 module
- Set up a dask distributed environment
  - `conda create --name mydask dask distributed`
- Get allocation
  - `salloc -N 4 -t 30 -C haswell`
- Activate dask distributed
  - `source activate mydask`
- Start scheduler
  - (If necessary `export LC_ALL=en_US.UTF-8`)
  - `dask-scheduler --scheduler-file $HOME/.dask_sched &`

Setting up a dask.distributed Cluster, continued

- Start workers on each node
  - echo $SLURM_NODELIST
    - **nid00[620-623]**
  - which dask-worker
    - **/global/homes/m/mikeri/.conda_env/mydask/bin/dask-worker**
    - ssh -o StrictHostKeyChecking=no -p 22 ***nid00621*** LC_ALL=en_US.UTF-8 ***/global/homes/m/mikeri/.conda_env/mydask/bin/dask-worker*** --scheduler-file $HOME/.dask_sched --nthreads 0 --nprocs 1 --host ***nid00621*** &
- Cray’s Analytics on XC product will include an option to automatically set up a Dask Distributed scheduler and workers in your containers

Using dask.distributed for word count

- **Word count using Dask bag and distributed scheduler**
  - from dask import bag as db
  - from distributed import Client, progress
  - client = Client(scheduler_file='/global/homes/m/mikeri/.dask_sched')
  - client.scheduler_info()
  - email = db.read_text('/global/cscratch1/sd/mikeri/enron_mail/mailbag.*.json.gz')
  - emailwords = email.str.split().concat()
  - words = client.persist(emailwords)
  - wordcount = words.frequencies().topk(300, lambda x: x[1])
  - wc_future = client.compute(wordcount)
  - progress(wc_future)
  - result = client.gather(wc_future)
  - result

Revised word count

- Reuse persisted words bag
  - wordcount2 = words.filter(lambda s: len(s) > 3 and s.isalpha()).map(lambda s: s.upper()).frequencies().topk(300, lambda x: x[1])
  - wc2_future = client.compute(wordcount2)
  - progress(wc2_future)
  - result = client.gather(wc2_future)
  - result

Demo: dask.distributed on XC

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, REVEAL, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Q&A

Michael Ringenburg
<mikeri@cray.com>

Things to try during break

- salloc for 5 Haswell nodes on Cori:
  - salloc --reservation=CUG2C -N 5 -p regular -C haswell -t 60 -A ntrain
- Spark
  - module load spark
  - start-all.sh
  - Run spark commands, e.g.: spark-shell, pyspark, spark-submit
  - Enron words dataset: /global/cscratch1/sd/mikeri/enron_mail/words.*.json.gz
  - When done: stop-all.sh
- Anaconda
  - module load python/3.5-anaconda (or 2.7-anaconda)
  - conda config --add envs_dirs $HOME/.conda_env
  - Try a biopython environment:
    - conda create --name bio biopython python=3.5
    - source activate bio
    - Orchid Dataset: /global/cscratch1/sd/mikeri/ls_orchid.fasta

Analytics on XC
Setting up an R Environment

CUG 2017 CAFFEINATED COMPUTING

Redmond, Washington May 7-11, 2017

What is R?

- **R project for Statistical Computing**
  - https://www.r-project.org
  - Environment for statistical computing and graphics
  - “GNU S”
  - Freely available – but note most R packages have licenses
    - (GPL-2, GPL-3, MIT, Apache, etc.)
  - Latest Version R 3.4.0 (You Stupid Darkness)
    - R version 3.4.0 (2017-04-21) -- "You Stupid Darkness"

- **CRAN - The Comprehensive R Archive Network**
  - https://cran.r-project.org
  - Network of ftp and web servers that store identical, up-to-date, versions of code and documentation for R
  - R manuals
    - https://cran.r-project.org/doc/manuals/

What we plan to cover in the tutorial

- Setting up an R environment on XC
- Update on R support on XC built with Cray libsci
  - module load cray-R
  - Currently support R-3.3.3
- Build Instructions:
  - Build R using gcc/gfortran
  - Build R using Intel C++ and Fortran Compilers + MKL
  - Build R using gcc/gfortran + MKL
- Using Anaconda to manage R packages and multiple R versions (environments)
- Setting up a R cluster using “parallel” package
- Setting up a pdbR environment (pdbMPI)

Setting up an R environment on XC

- **Base R Install**
  - Easiest way to install on XC is to build from source
    - Allows one to build optimized versions which use optimized math libraries (Cray libsci, Intel MKL)
    - Download most recent version from CRAN
      - R-3.4.0.tar.gz
      - wget https://cran.r-project.org/src/base/R-3/R-3.4.0.tar.gz
  - CRAN repository also provides precompiled binaries
    - Linux, OS X, Windows
  - Anaconda R
    - Quite useful for managing R packages and multiple R environments on XC
    - List of R language packages available for install from conda is located at http://repo.continuum.io/pkgs/r/

Simple R build from source using gcc/gfortran

- module load gcc
- wget https://cran.r-project.org/src/base/R-3/R-3.4.0.tar.gz
- (Note: other mirror sites work as well, for example
- wget http://cran.rstudio.com/src/base/R-3/R-3.4.0.tar.gz
- tar -xzf R-3.4.0.tar.gz
- cd R-3.4.0/

- ./configure --prefix=/lus/scratch/kristyn/R/R-3.4.0/R-3.4.0
- make
- make check; make install
- cd /lus/scratch/kristyn/R/R-3.4.0/bin
- ./R
- file R
  - (R: POSIX shell script, ASCII text executable)
- file exec/R
  - (exec/R: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 3.0.0

Installed Packages – Base Install

> installed.packages()[,c("Version","License")]

```
           Version   License
base       "3.4.0"   "Part of R 3.4.0"
boot       "1.3-19"  "Unlimited"
class      "7.3-14"  "GPL-2 | GPL-3"
cluster    "2.0.6"   "GPL (>= 2)"
codetools  "0.2-15"  "GPL"
compiler   "3.4.0"   "Part of R 3.4.0"
datasets   "3.4.0"   "Part of R 3.4.0"
foreign    "0.8-67"  "GPL (>= 2)"
graphics   "3.4.0"   "Part of R 3.4.0"
grDevices  "3.4.0"   "Part of R 3.4.0"
grid       "3.4.0"   "Part of R 3.4.0"
KernSmooth "2.23-15" "Unlimited"
lattice    "0.20-35" "GPL (>= 2)"
MASS       "7.3-47"  "GPL-2 | GPL-3"
Matrix     "1.2-9"   "GPL (>= 2) | file LICENCE"
```

```
           Version   License

methods    "3.4.0"   "Part of R 3.4.0"
mgcv       "1.8-17"  "GPL (>= 2)"
nlme       "3.1-131" "GPL (>= 2) | file LICENCE"
nnet       "7.3-12"  "GPL-2 | GPL-3"
parallel   "3.4.0"   "Part of R 3.4.0"
rpart      "4.1-11"  "GPL-2 | GPL-3"
spatial    "7.3-11"  "GPL-2 | GPL-3"
splines    "3.4.0"   "Part of R 3.4.0"
stats      "3.4.0"   "Part of R 3.4.0"
stats4     "3.4.0"   "Part of R 3.4.0"
survival   "2.41-3"  "LGPL (>= 2)"
tcltk      "3.4.0"   "Part of R 3.4.0"
tools      "3.4.0"   "Part of R 3.4.0"
utils      "3.4.0"   "Part of R 3.4.0"
```

Build of R using gcc/gfortran + Cray libsci

- **Cray is providing a cray-R/3.3.3 rpm that is scheduled for release in May, built with gcc/6.1.0**
  - module load cray-R
- **Example build recipe: (for those needing additional customization)**

```
module load gcc
wget https://cran.r-project.org/src/base/R-3/R-3.4.0.tar.gz
tar -xzf R-3.4.0.tar.gz
cd R-3.4.0/
./configure --build=x86_64-suse-linux --prefix=%{install_dir} --with-blas="-fopenmp -L/opt/cray/pe/libsci/16.11.1/GNU/5.1/x86_64/lib -lsci_gnu_51_mp" --with-lapack
make
make install
```

Build R using Intel C++ and Fortran Compilers + MKL

```
>   module swap PrgEnv-cray PrgEnv-intel
>   wget https://cran.r-project.org/src/base/R-3/R-3.4.0.tar.gz
>   tar -xzf R-3.4.0.tar.gz

>   setenv CC icc
>   setenv CXX icpc
>   setenv AR xiar
>   setenv LD xild

>   setenv CFLAGS “-03 –ipo –qopenmp –xHost”
>   setenv CXXFLAGS “-03 –ipo –qopenmp –xHost”
>   setenv MKL “-lmkl_gf_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread”

>   ./configure --prefix=/lus/scratch/kristyn/R/R-3.4.0/R-3.4.0 CC="icc -mkl" CXX="icpc -mkl" FC="ifort -mkl" F77="ifort -mkl" FPICFLAGS="-fPIC" AR=xiar LD=xild --with-x=no --with-blas=-lmkl --with-lapack=-lmkl

>   make
>   make install
>   cd /lus/scratch/kristyn/R/R-3.4.0/bin
```

https://software.intel.com/en-us/articles/build-r-301-with-intel-c-compiler-and-intel-mkl-on-linux

Simple build using gcc/gfortran + MKL

```
module load PrgEnv-intel
module load gcc

setenv CC gcc
setenv F77 gfortran
setenv AR xiar
setenv LD xild
setenv MKL “-lmkl_gf_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread”

./configure --prefix=/lus/scratch/kristyn/R/R-3.4.0/R-3.4.0 --with-blas="$MKL" --with-lapack
make
make install
cd /lus/scratch/kristyn/R/R-3.4.0/bin
./R
```

https://cran.r-project.org/doc/manuals/r-release/R-admin.html#MKL

Build R + MKL build notes

Default is to build shared libraries:

Useful to print out shared library dependencies to verify MKL is being used

```
> ldd exec/R
linux-vdso.so.1 (0x00007ffc247ed000)
libmkl_gf_lp64.so =>
/opt/intel/compilers_and_libraries_2017.1.132/linux/mkl/lib/intel64_lin/libmkl_gf_lp64.so
(0x00007f47f09da000)
libmkl_intel_thread.so =>
/opt/intel/compilers_and_libraries_2017.1.132/linux/mkl/lib/intel64_lin/libmkl_intel_thread.so
(0x00007f47eefcc000)
libmkl_core.so => /opt/intel/compilers_and_libraries_2017.1.132/linux/mkl/lib/intel64_lin/libmkl_core.so
(0x00007f47ed525000)
libiomp5.so => /opt/intel/compilers_and_libraries_2017.1.132/linux/compiler/lib/intel64/libiomp5.so
(0x00007f47ed182000)
```

Can also specify to build static binary by using --enable-static when running ./configure

Set up simple modulefile

- Create a modulefiles directory
  - /lus/scratch/kristyn/modulefiles/R
- module use /lus/scratch/kristyn/modulefiles
- module load R/R-3.4.0

where the file R-3.4.0 contains

```
#%Module2.0
##
module load java
module load gcc

set R_VERSION R-3.4.0
set R_PATH /lus/scratch/kristyn/R/$R_VERSION/

prepend-path PATH $R_PATH/bin
prepend-path LD_LIBRARY_PATH $R_PATH/lib64/R/library
prepend-path MANPATH $R_PATH/share/man
```

R/3.3.2 installed on Cori

```
module load R/3.3.2

kristyn@cori01:~> module display R/3.3.2
-------------------------------------------------------------------
/usr/common/software/modulefiles/R/3.3.2:


conflict         R
module-whatis    R is a free software environment for statistical computing and graphics.
                 Built with Intel MKL support.
prepend-path     PATH /global/common/cori/software/R/3.3.2/bin
prepend-path     LD_LIBRARY_PATH /global/common/cori/software/R/3.3.2/lib64:
                 /global/common/cori/software/R/3.3.2/lib64/R/lib
-------------------------------------------------------------------
```

Installing R Packages from CRAN

- **Bring up R on login node and install needed packages**
  - Need external access to download packages
  - In general, most tested, and most reliable compiler for R packages are the GNU compilers (gcc, gfortran)
  - Note, if using a site-installed version, any additional installed packages will be saved to a location in your home directory
    - ~/R/x86_64-suse-linux-gnu-library/3.3
- **R packages we will be using for the tutorial**
  - install.packages("foreach")
  - install.packages("doParallel")
  - install.packages("rlecuyer")
  - install.packages("randomForest")
  - install.packages("SPARQL")

Managing R using Anaconda

- **Anaconda R**
  - Quite useful for managing R packages and multiple R environments on XC
  - List of R language packages available for install from conda is located at http://repo.continuum.io/pkgs/r/
  - R Essentials bundle includes about 100 of the most popular packages for R

```
> conda create --name myR -c r r-essentials
> source activate myR
```

- **Also can specify specific versions of R**

```
> conda create --name myR_3.2.2 -c r r=3.2.2
```

- **When using an older version of R I found it works better to create the conda environment first, activate this, then install the allowing packages, allowing conda to manage the package version dependencies**

```
> source activate myR_3.2.2
> conda install -c r r-essentails r-xml
```

Running R using CCM

```
salloc -N 4 --partition=ccm_queue

# Determine nid allocations
echo “$SLURM_NODELIST” or env | grep SLURM
    SLURM_NODELIST=nid0000[4-7]

# load R module
module use /lus/scratch/R/modulefiles
module load R/R-3.4.0

# Log into head node and propagate environment
module load ccm
ccmlogin –V

# Start up R on head node
R
```

Note: CCM may not be available on all XC systems. This is a site-configuration.
Cori no longer has CCM running, but is set up so one can use ssh between nodes within a job.

See /global/cscratch1/sd/kristyn/CUG2017/R/README for additional details.

R using “parallel” package using CCM mode
Setting up a simple parallel socket cluster

“parallel” package

https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

```
>    library(parallel)
>    machineVec <- c(rep("nid00004",4), rep("nid00005",4), rep("nid00006",4), rep("nid00007",4))

>    machineVec
>     [1] "nid00004" "nid00004" "nid00004" "nid00004" "nid00005" "nid00005"
>     [7] "nid00005" "nid00005" "nid00006" "nid00006" "nid00006" "nid00006"
>    [13] "nid00007" "nid00007" "nid00007" "nid00007"
>    cl <- makeCluster(machineVec)
>    cl
>    socket cluster with 16 nodes on hosts 'nid00004', 'nid00005', 'nid00006', 'nid00007’

>    help(makeCluster)

>    stopCluster(cl)
```

Simple Parallel Socket Cluster

- **Basic functionality**
  - Runs 'Rscript' on the specified host(s) to set up a worker process which listens on a socket for expressions to evaluate, and returns the results (as serialized objects).
- **Commonly used R packages which then build upon the "parallel" package**
  - "foreach" package
    - Provides looping construct
  - "doParallel" package
    - Provides mechanism needed to execute **foreach** loops in parallel
  - https://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf

Example datasets

```
>   # Base install of R already includes several datasets
>   # To look at the datasets available in loaded packages
>   data()

>   # load the iris dataset
>   data(iris)
>   head(iris)

>   # Many R packages also contain additional datasets
>   install.package(‘rattle’)
>   data(wine, package='rattle')

>   # Also can import data directly
>   # Here read.table reads a file in table format and creates a dataframe from it
>   url <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv'
>   whitewine <- read.table(url,header=TRUE,sep=“;”)
>   head(whitewine)
```

Example Code: using foreach and doParallel

```
> library(parallel)
> library(foreach)
> library(doParallel)
> machineVec <- c(rep("nid00004",4), rep("nid00005",4), rep("nid00006",4), rep("nid00007",4))

> cl <- makeCluster(machineVec)
> # To use the "foreach", we need to register the cluster with
> registerDoParallel(cl)
> getDoParWorkers()

> # sequential execution
> system.time(foreach(i=1:100000) %do% sum(tanh(1:i)))
> # parallel execution
> system.time(foreach(i=1:10000) %dopar% sum(tanh(1:i)))

> mcoptions <- list(preschedule=FALSE, set.seed=FALSE, cores=4)
> system.time(foreach(i=1:100000,.options.multicore=mcoptions) %dopar% sum(tanh(1:i)))
```

Example Code: randomForest

```
> # Parallel execution of randomForest
> x <- matrix(runif(500), 100)
> y <- gl(2, 50)
>
> library(randomForest)
>
> rf <- foreach(ntree=rep(25000, 6), .combine=combine,
  .multicombine=TRUE, .packages='randomForest')
  %dopar% { randomForest(x, y, ntree=ntree)}
```

Programming with Big Data in R (pbdR)

- Set of highly scalable R packages for distributed computing in data science
  - http://r-pbd.org/
- George Ostrouchov, Wei-Chen Chen, Drew Schmidt, Pragneshkumar Patel
- Winner of the Oak Ridge National Laboratory 2016 Significant Event Award for "Harnessing HPC Capability at OLCF with the R Language for Deep Data Science

Installing pdbMPI package

- **If not already installed, install rlecuyer package**
  - wget https://cran.r-project.org/src/contrib/rlecuyer_0.3-4.tar.gz
  - R CMD INSTALL --no-test-load rlecuyer_0.3-4.tar.gz
- **Install pdbMPI package**
  - wget https://cran.r-project.org/src/contrib/pbdMPI_0.3-3.tar.gz
  - R CMD INSTALL pbdMPI_0.3-3.tar.gz --configure-args="--with-mpi=/opt/cray/pe/mpt/default/gni/mpich-gnu/51/ --disable-opa --with-Rmpi-type=MPICH2" --no-test-load

pdbMPI: run “Hello World”

Create file mpi_hello_world.r

```
     # load the package
     suppressMessages(library(pbdMPI, quietly = TRUE))

     # initialize the MPI communicators
     init()

     # Hello world
     message <- paste("Hello from rank", comm.rank(), "of", comm.size())
    comm.print(message, all.rank=TRUE, quiet=TRUE)

    # shut down the communicators and exit
    finalize()
```

```
> srun -N 4 Rscript mpi_hello_world.r
```

pbdMPi – beyond “Hello World”

- HPSC Cookbook – Wei-Chen Chen
- https://snoweye.github.io/hpsc/cookbook.html

- In addition there are several tutorials available with source code available for download
- Tutorials 1 and 2 both use the Iris dataset already available with base R install

Parallel (SPMD) pi Example (from HPSC)

```
# File name: ex_pi_spmd.r
# Run: srun -N 2 Rscript --vanilla ex_pi_spmd.r

### Load pbdMPI and initial the communicator.
library(pbdMPI, quiet = TRUE)
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Compute pi.
n <- 1000
totalcpu <- .comm.size
id <- .comm.rank + 1
mypi <- 4*sum(1/(1+((seq(id,n,totalcpu)-.5)/n)^2))/n    # The example from Rmpi.
mypi <- reduce(mypi, op = "sum")

### Output from RANK 0 since mpi.reduce(...) will dump only to 0 by default.
comm.print(mypi)
finalize()
```

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

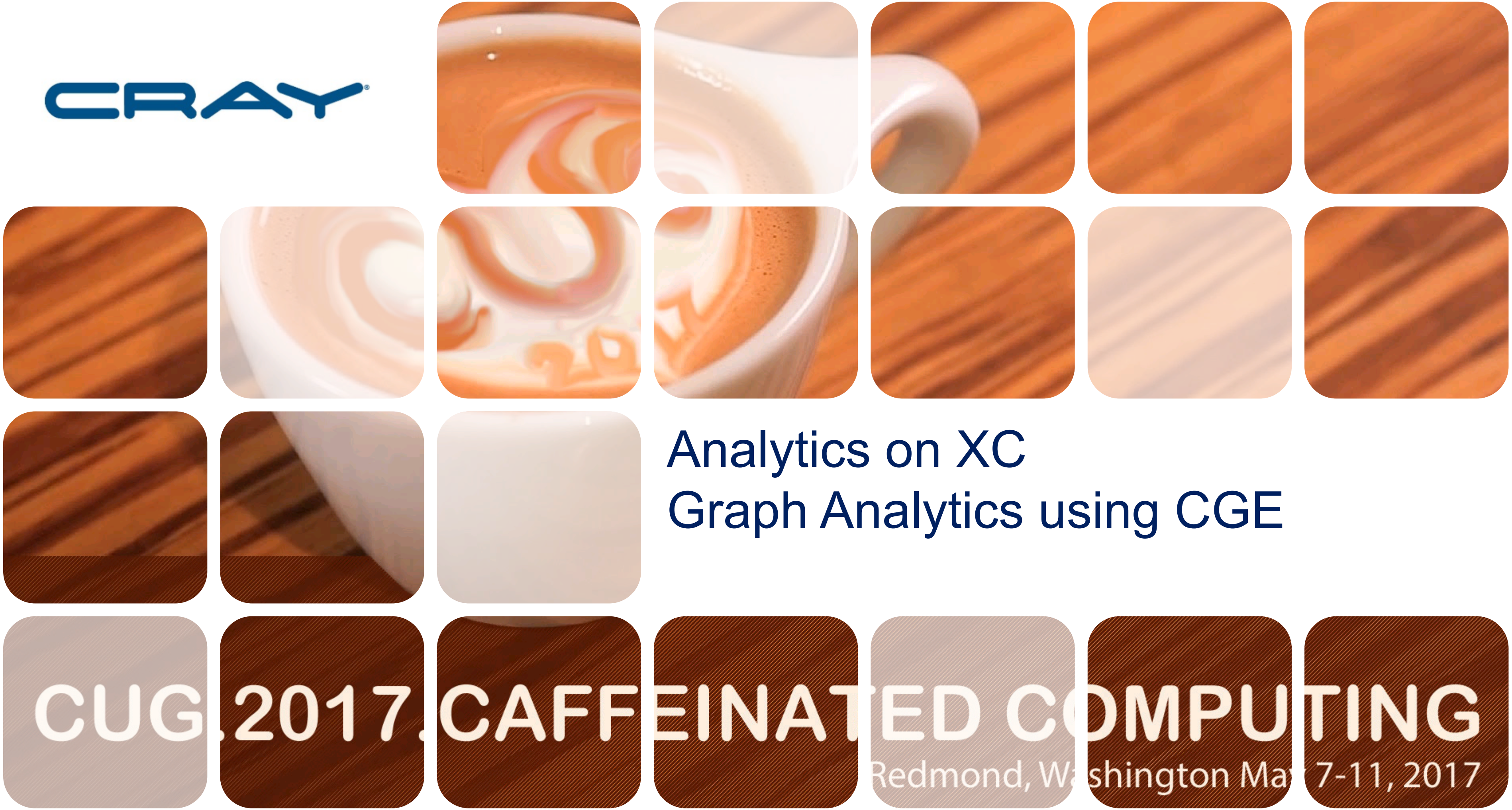
Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, REVEAL, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Q&A

Kristi Maschhoff
kristyn@cray.com



Analytics on XC
Graph Analytics using CGE

CUG.2017.CAFFEINATED COMPUTING

Redmond, Washington May 7-11, 2017

Cray Graph Engine: Overview

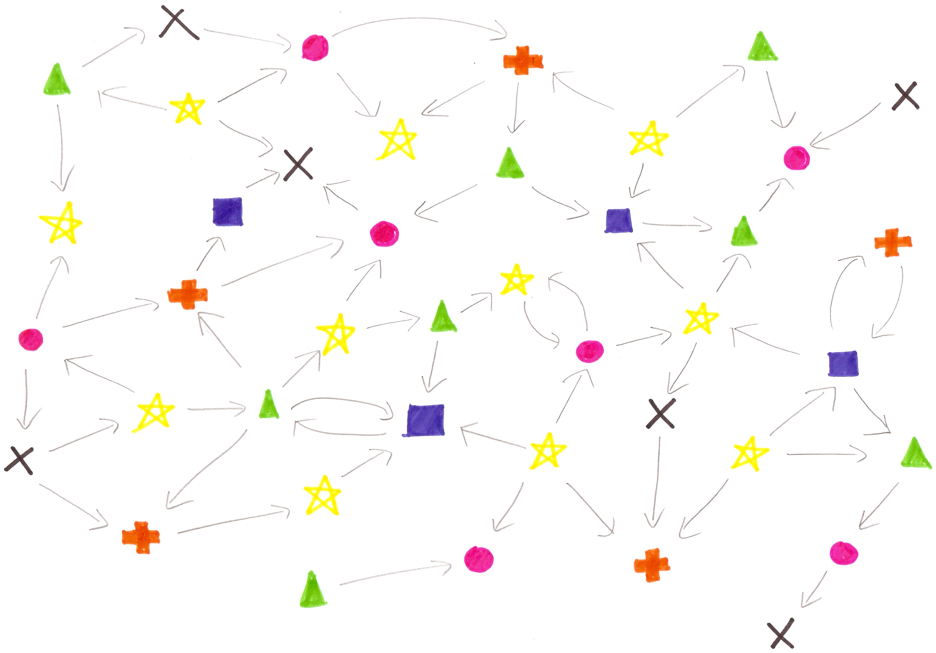
- **The Cray Graph Engine CGE**
  - An analytic in-memory graph database – capable of both basic graph pattern search (using SPARQL) and linear-algebraic search (graph theoretic using CGE-BGF)
  - Built for "vertical scaling" based on parallel and distributed computing principles- competitors are all horizontally scaled
  - Can handle 1000x the size of competing in-memory graph databases
  - Can handle complex data (e.g. hot-spot vertices, long-diameter, etc.)
  - At least a 10x speed-up on query retrieval times and can be 100x faster on massive graph-theoretic workloads (50 GBs+ - 512 TBs)
  - Brings interactivity to graph-based discovery over triple stores, disk-based graph databases and graph-analytic toolkits.

Cray Graph Engine: Updates and Features

- **Multi-Architecture Support**
  - CGE is available on the Urika-GX and the XC platforms.
  - Strong Scaling becomes a key differentiator
    - Bigger datasets => more nodes => better performance
- **Integration with Spark (new)**
  - Interface to data sources
  - Support for end-end analytic workflow realization
- **Integration with Python/Jupyter Notebooks**
  - Connect to SPARQL endpoint using sparqlwrapper or sparql-client packages
  - CGE Python API – utilizes the CGE Java API
    - Start up server, run queries, updates, checkpoint, shut down
- **Integration with R**
  - SPARQL package – connect to SPARQL endpoint, run queries, updates

- **Don’t miss Rob Vesse’s talk on Thursday!**
  - Thursday, Technical Session 27B
  - “Quantifying Performance of CGE: A Unifed Scalable Pattern Mining and Search System”

Graph analysis workloads

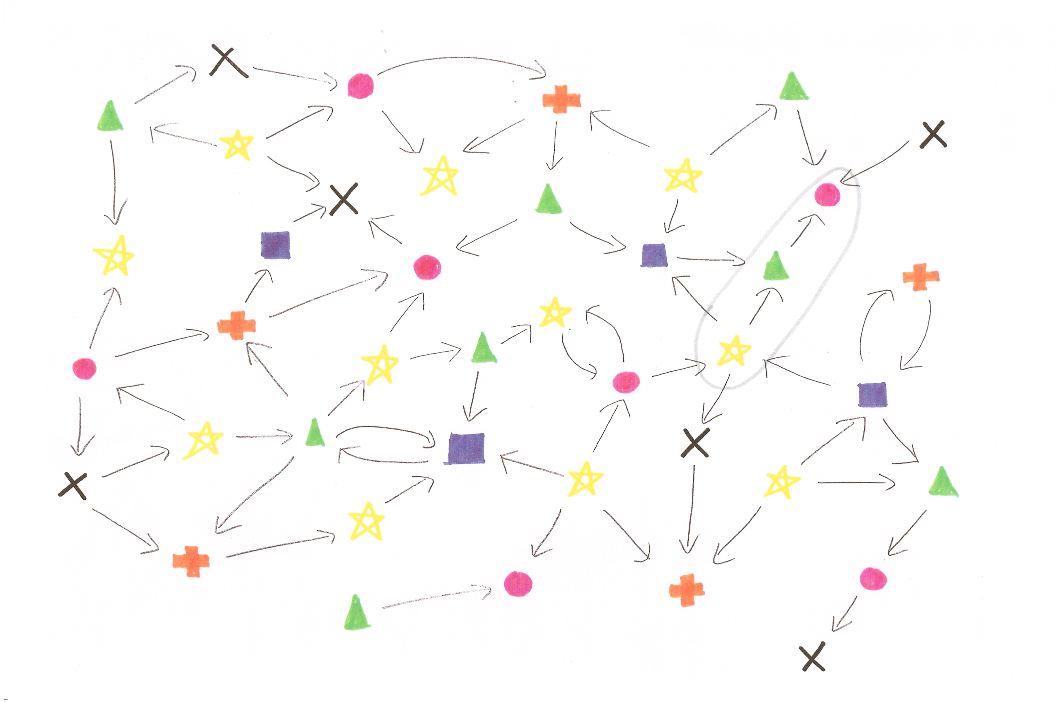


- Two main workloads
  - Pattern matching
  - Whole graph analysis
- Typical systems only good at one
- CGE excels at both

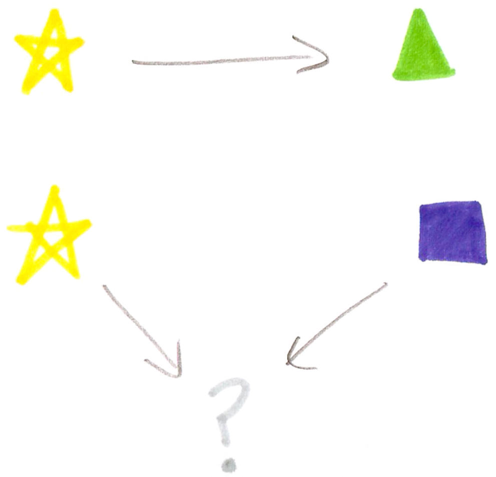
What we plan to cover in the tutorial

- **Background on CGE**
  - Pattern matching, whole-graph analysis
  - Benchmarking results demonstrating CGE scaling on XC
- **Hands-on exercises**
  - Build and start up a database (cge-launcher)
  - Run queries
    - Using the cge-cli command line
    - Using the CGE Web UI
  - Integration with R and Python
    - Connecting to the CGE SPARQL endpoint
      - Using R SPARQL package
      - Using Python SPARQLwrapper package

A Graph-pattern matching workload



Given a pattern of interest find all instances thereof…

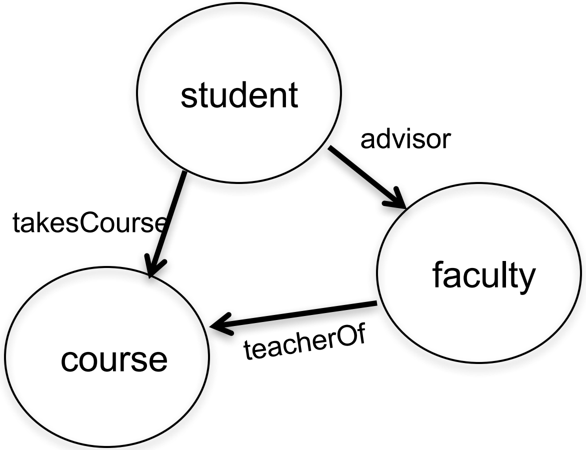


What SPARQL Can Do

- Subgraph isomorphism on specific, fixed patterns

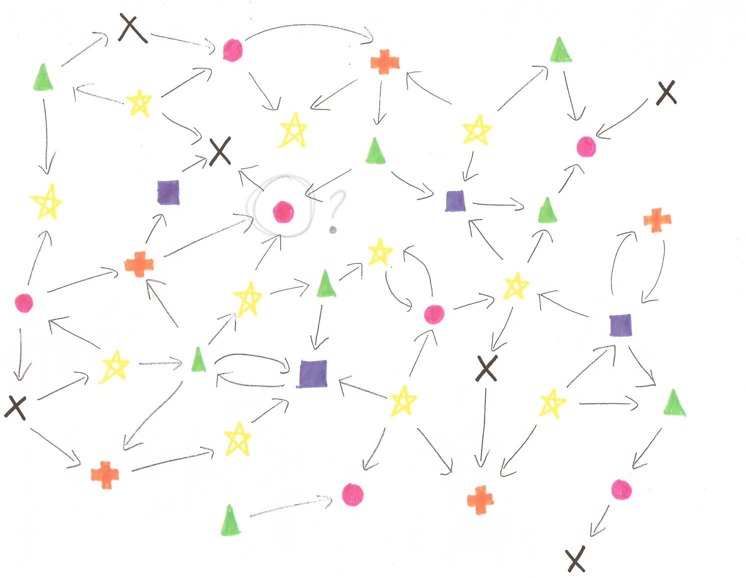
“LUBM Query 9”

```
SELECT ?X, ?Y, ?Z
WHERE
{ ?X rdf:type ub:Student .
  ?Y rdf:type ub:Faculty .
  ?Z rdf:type ub:Course .
  ?X ub:advisor ?Y .
  ?Y ub:teacherOf ?Z .
  ?X ub:takesCourse ?Z}
```



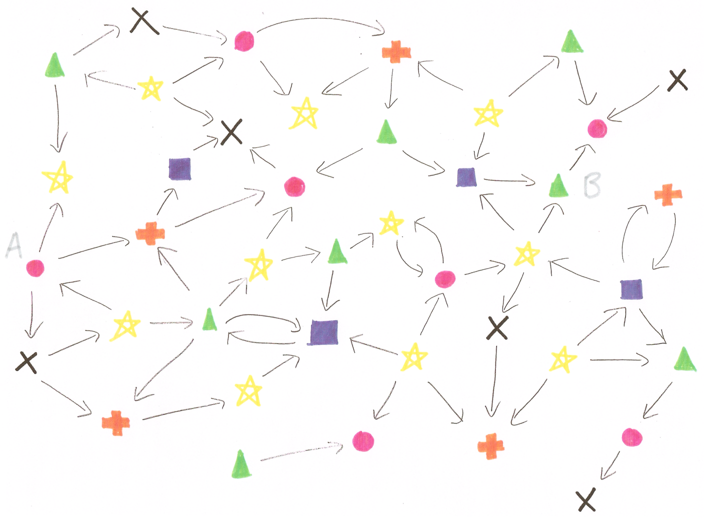
- `Plus lots of useful database features: filter, group, update…`

A Graph-theoretic Workload



- What is the ranking of the targeted vertex?

- What's the shortest route from A to B?



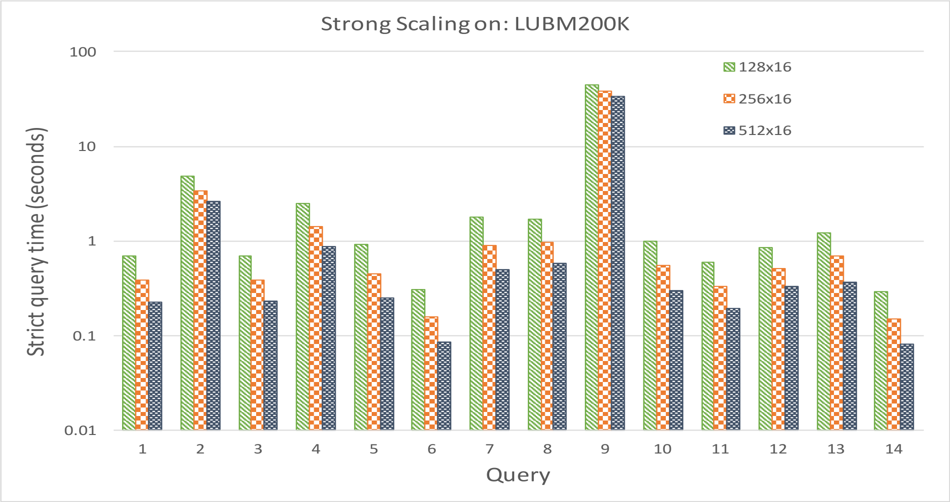
Built-in Graph Functions (BGFs)

- RDF and SPARQL are graph-oriented, but SPARQL is limited in its ability to express graph processing
- We augmented SPARQL with a capability of calling library graph algorithms
- You can go from SPARQL to a graph algorithm and back to SPARQL for further refinement
- The whole is greater than the sum of its parts.

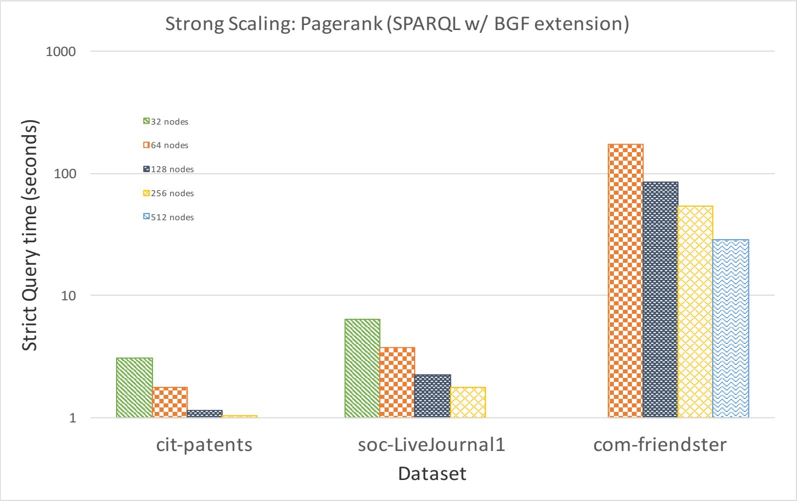
Cray Graph Engine: Benchmarks

- Evidence of Strong Scaling

Graph Pattern Search



Graph-Theoretic Algorithms

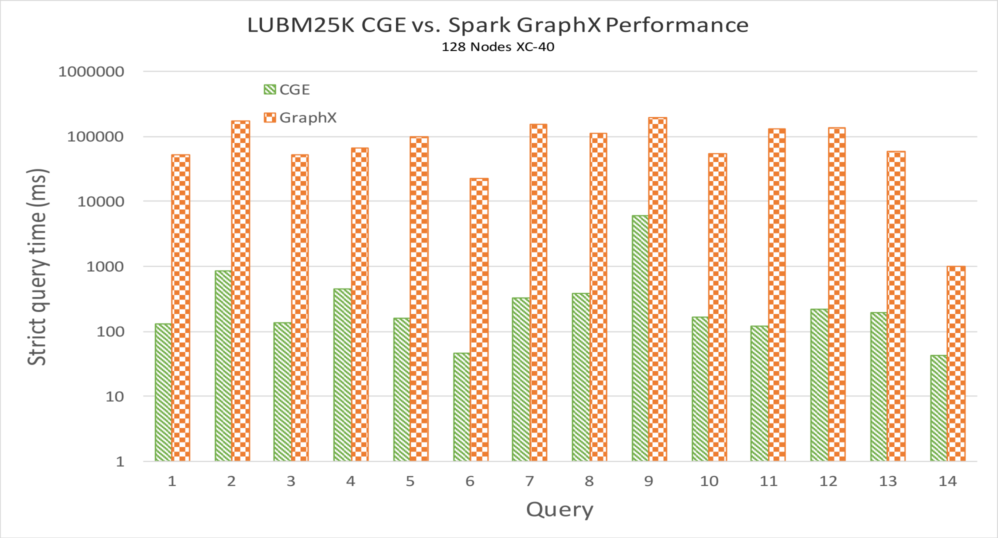


More nodes => Better performance

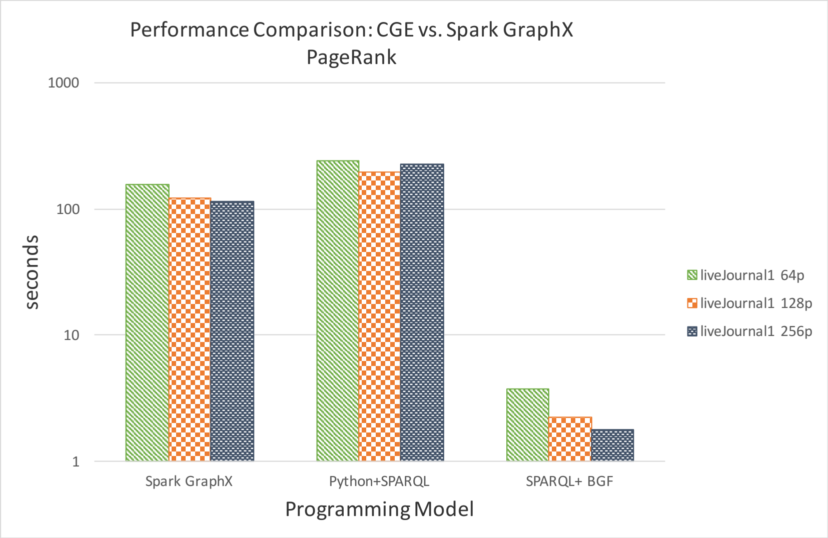
Cray Graph Engine: Benchmarks

- Comparison to Spark+GraphX

Graph Pattern Search



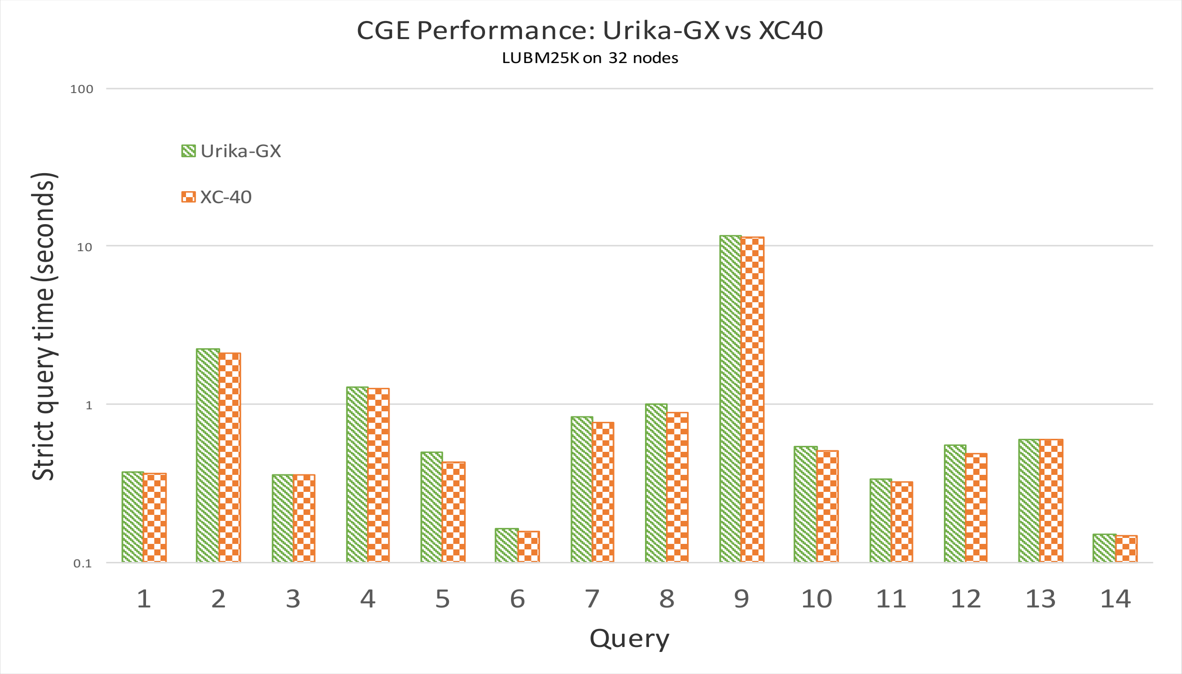
Graph-Theoretic Algorithms



Nearly a 100x performance speed-up over GraphX....

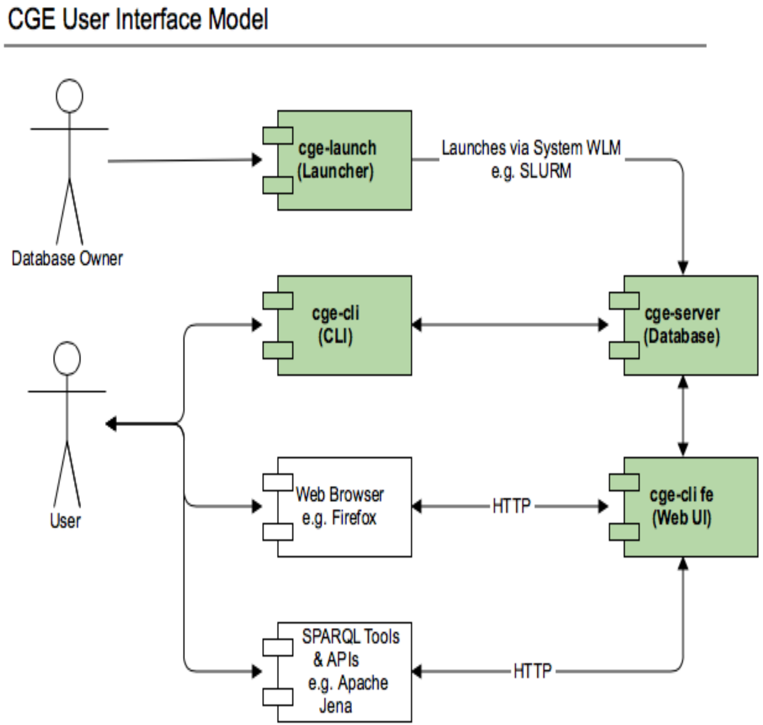
Cray Graph Engine: Benchmarks

- Architecture Portability



Porting does not affect performance

CGE User Interface Model



- Database owner launches the database server
- Users interact via their preferred interface
  - Command Line
  - Web Browser
  - SPARQL Tools & APIs
- CLI may be used for scripted workflows

Building and launching

- cge-launch is used to build databases:

```
cge-launch –N 8 –I 16 –o /mnt/lustre/myresults –d
/mnt/lustre/mydata –l logfile
```

- `cge-launch` is a script that takes care of resource allocation for the user!

- After a successful build, the database directory will contain:

  dataset.nt
  rules.txt
  dbQuads
  string_table_chars
  string_table_chars.index
  graph.info

The database port

- A TCP port used for communication with this server instance:

```
cge-launch –N 8 –I 16 –p 3750 …
```

- The default is 3750
- Changing this port allows multiple versions

The database directory

- The database directory, typically:

```
/mnt/lustre/user/datasets/lubm0
```

- Is the start of a directory tree containing all checkpoints, and potentially authorized_keys
- It can be moved, archived and returned (!)
- Multiple users can access it, with permissions

The Command Line Interface (CLI)

```
cge-cli –db-port 3750 query myquery.rq
```

- The CLI is used for most interactions with the server, and has many options…
- `cge-cli help` (or `cge-cli help checkpoint`) will give verbose information on options
- Designed for scripted control, querying and updates with database server
- Communications are secure SSH

Most common options

`query` – submits SPARQL queries

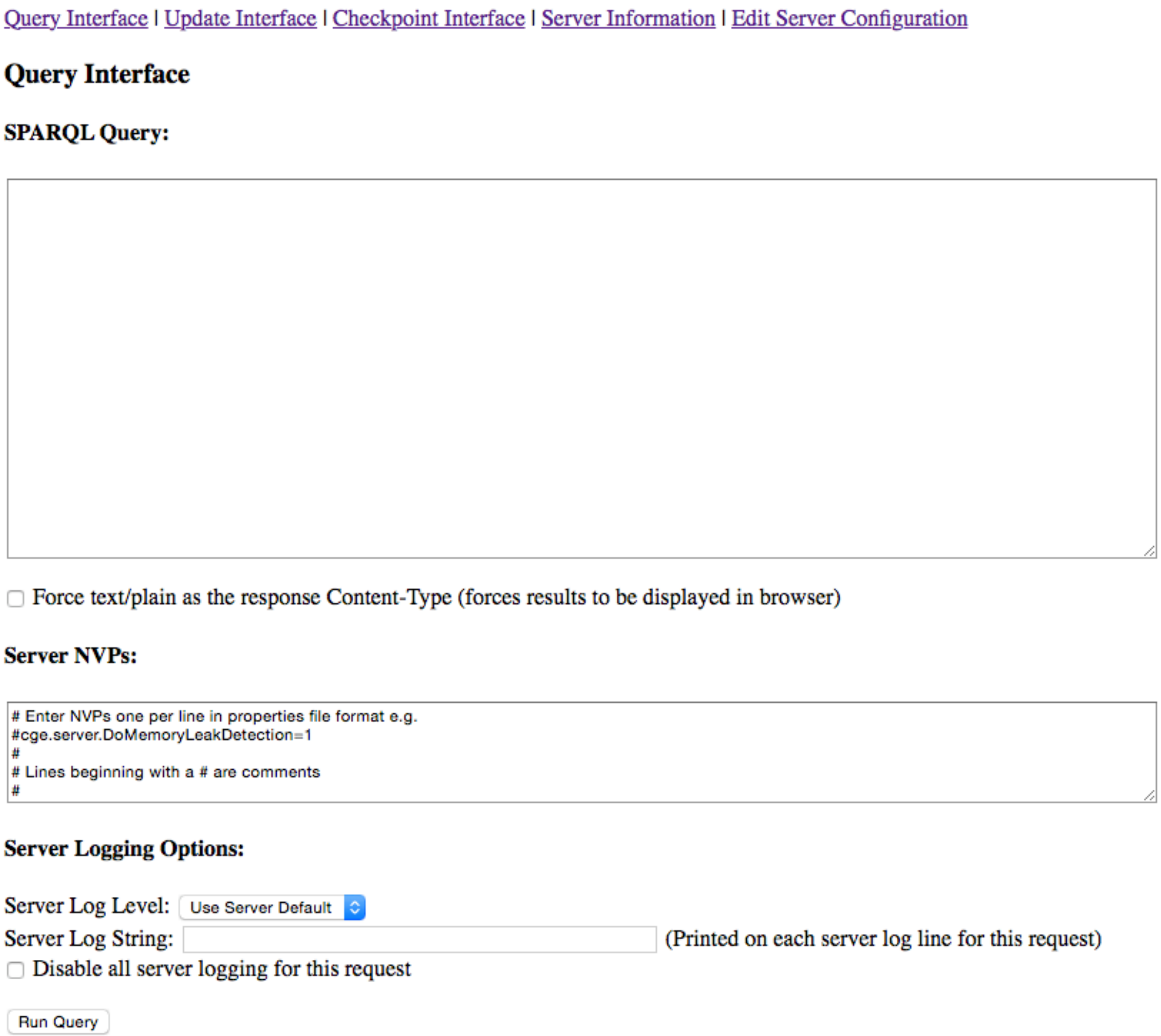
`update` – submits SPARUL updates

`sparql` – submits both queries and updates

`checkpoint` – creates a database checkpoint

`echo` – check status of server

Main



Query Interface | Update Interface | Checkpoint Interface | Server Information | Edit Server Configuration

Query Interface

SPARQL Query:

□ Force text/plain as the response Content-Type (forces results to be displayed in browser)

Server NVPs:

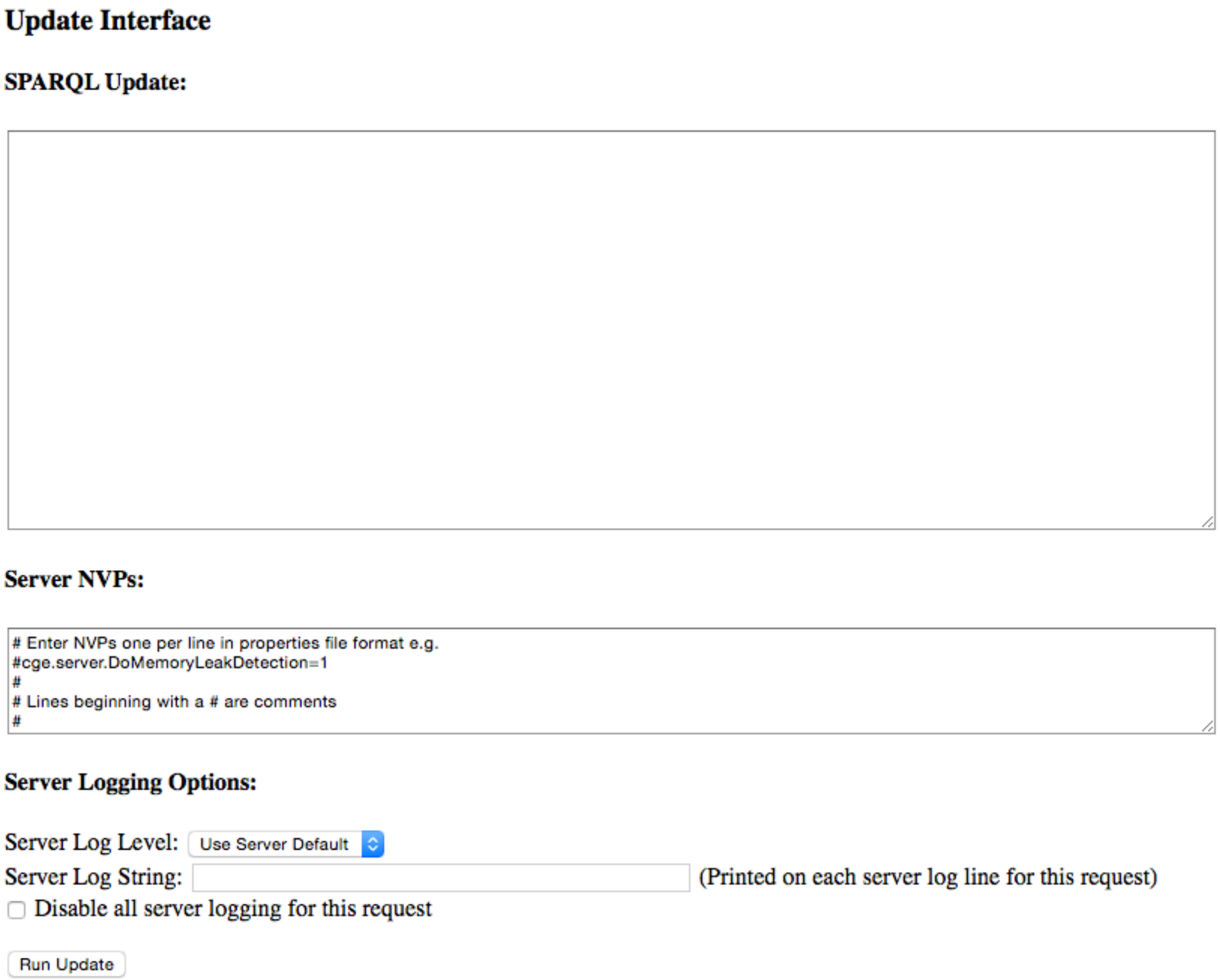
```
# Enter NVPs one per line in properties file format e.g.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

Server Logging Options:

Server Log Level: Use Server Default

Server Log String: (Printed on each server log line for this request)

□ Disable all server logging for this request



Update Interface

SPARQL Update:

Server NVPs:

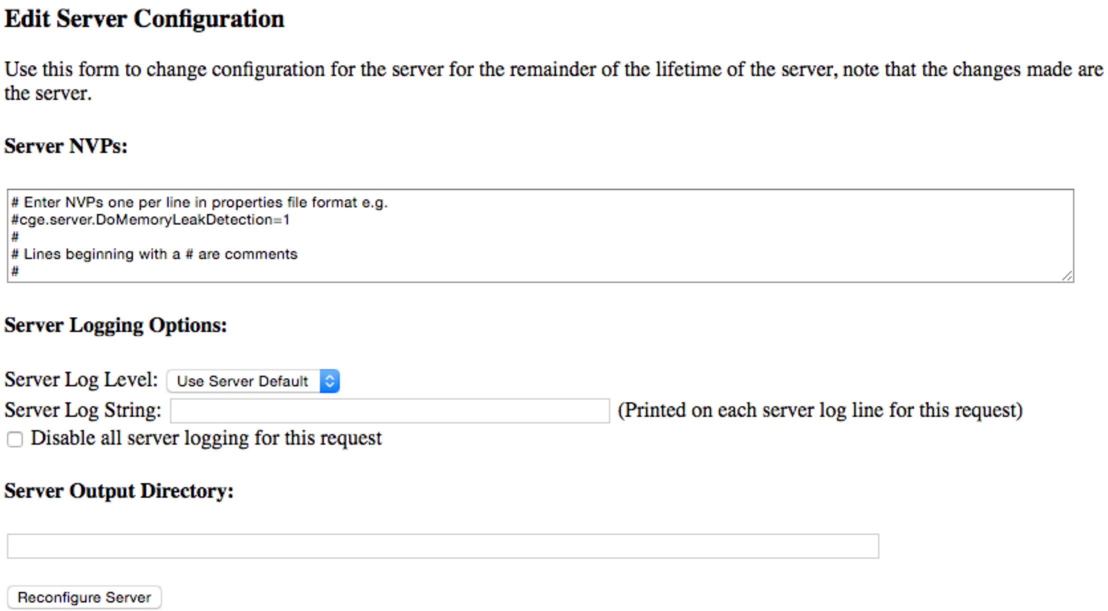
```
# Enter NVPs one per line in properties file format e.g.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

Server Logging Options:

Server Log Level: Use Server Default

Server Log String: (Printed on each server log line for this request)

☐ Disable all server logging for this request



Edit Server Configuration

Use this form to change configuration for the server for the remainder of the lifetime of the server, note that the changes made are not persistent beyond the lifetime of the server.

Server NVPs:

```
# Enter NVPs one per line in properties file format e.g.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

Server Logging Options:

Server Log Level: Use Server Default

Server Log String: (Printed on each server log line for this request)

□ Disable all server logging for this request

Server Output Directory:

Hands on Exercises: Running CGE on Cori

- **See README for instructions and exercises**
  - /global/cscratch1/sd/kristyn/CUG2017/CGE/README
- **To use CGE Web UI, need to set up ssh tunneling**
  - Current cge-launch script for XC depends on xtprocadmin,
    - Only available on internal Cori MOM nodes cmom02 and cmom05, need to ssh to these nodes from login node
  - Create tunnel from my laptop to internal cmom02 node on Cori
    - Use a random port number (8022) to connect to ssh port 22
      - ssh –L localhost:8022:cmom02:22 cori.nersc.gov
    - Then ssh directly into cmom02, choosing another random port number (15000) for CGE fe
      - ssh –p 8022 –L localhost:15000:localhost:15000 localhost
- **Set up database directory on Lustre**
  - Make sure Lustre striping is set
    - lfs setstripe –c 16 –stripe-size 16m .
  - Needed files: dataset.nt, graph.info, rules.txt
- **Set up query_results directory on Lustre**
  - Make sure Lustre stripiing is set
- **Be sure to set passwordless ssh**
  - ssh-keygen
  - cat id_dsa.pub >> authorized_keys

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, REVEAL, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Q&A

Kristi Maschhoff
kristyn@cray.com

Back-Up Slides

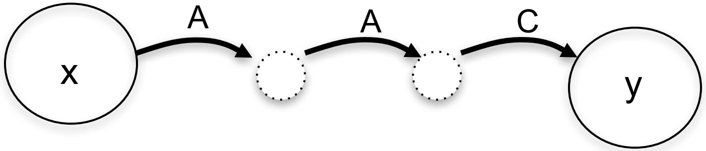
(in case WiFi connectivity is poor)

SPARQL isn't So Hot at…

- Breadth-first search
- Connected components
- Community detection
- …anything else that entails an indefinite-length search of the graph

- With one fairly unimportant exception: "property paths"

?x (A* | B*) / C ?y



Real-world Example:

Find Communities Among Botnets

```
PREFIX cray:    <http://cray.com/>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
CONSTRUCT {             build a graph, using the counts as weights
  ?ip1 ?nInstances ?ip2
                                     …and count them
} WHERE {
   SELECT ?ip1 ?ip2 (COUNT(?ip2) as ?nInstances)
   WHERE {
     ?uid <http://cs.org/p/hasOrigAddr> ?ip1 .       pick out communication pairs
     ?uid <http://cs.org/p/hasRespAddr> ?ip2 .
     ?uid <http://cs.org/p/hasOrigPort> ?port1URI .       get their port IDs
     ?uid <http://cs.org/p/hasRespPort> ?port2URI .
                                                        pull the integers from their port IDs
                                                          ( <http://cs.org/port#742> )
     BIND(xsd:integer(strafter(str(?port1URI),"http://cs.org/port#")) AS ?port1)
     BIND(xsd:integer(strafter(str(?port2URI),"http://cs.org/port#")) AS ?port2)
                                              pick out the port IDs >= 2000, that botnets use
     FILTER(?port1 >= 2000 && ?port2 >= 2000 && !sameTerm(?ip1, ?ip2))
   } GROUP BY ?ip1 ?ip2
                                    group all the distinct pairs…
}
```

NOW RUN COMMUNITY DETECTION ON THAT GRAPH!

How We Extended SPARQL

- INVOKE is paired with SPARQL's existing CONSTRUCT operator

```
CONSTRUCT  {
?ip1 ?nInstances ?ip2
WHERE {
…}
INVOKE  <http://cray.com/graphAlgorithm.community> (…)
```

- We extended SPARQL so that you can *nest* a CONSTRUCT/INVOKE pair.
- A new PRODUCING clause maps results back into SPARQL

```
PRODUCING ?vtx  ?communID
```

Botnets Revisited

```
PREFIX cray:    <http://cray.com/>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
CONSTRUCT {
  ?ip1 ?nInstances ?ip2
} WHERE {
    SELECT ?ip1 ?ip2 (COUNT(?ip2) as ?nInstances)
    WHERE {
      ?uid <http://cs.org/p/hasOrigAddr> ?ip1 .
      ?uid <http://cs.org/p/hasRespAddr> ?ip2 .
      ?uid <http://cs.org/p/hasOrigPort> ?port1URI .
      ?uid <http://cs.org/p/hasRespPort> ?port2URI .

      BIND(xsd:integer(strafter(str(?port1URI),"http://cs.org/port#")) AS ?port1)
      BIND(xsd:integer(strafter(str(?port2URI),"http://cs.org/port#")) AS ?port2)

      FILTER(?port1 >= 2000 && ?port2 >= 2000 && !sameTerm(?ip1, ?ip2))
    } GROUP BY ?ip1 ?ip2
}
INVOKE  <http://cray.com/graphAlgorithm.community> ()
PRODUCING ?vtx  ?communID
```

Applications for Available Algorithms

- **Search / neighborhood identification and extraction**
  - Pattern-matching / subgraph isomorphism: (Core functionality)
  - **Cybersecurity application: Context and search, data exfiltration, beaconing, attack identification**
- **Community detection**
  - Modularity:
  - Relaxed clique
  - **Cybersecurity application: Botnet detection and server hierarchy mapping**
- **Path finding**
  - Shortest path, S-T connectivity
  - **Cybersecurity application: Identify likely paths for information flow between nodes**
- **Key node / edge identification**
  - Betweenness centrality
  - **Cybersecurity application: find the vulnerable points in network configurations**
- **Anomaly identification and clustering**
  - **Cybersecurity application: Unknown-unknown identification**
  - **Cybersecurity application: BadRank: finds likely worst actors by association with known bad actors, a la PageRank**