Scaling Apache Spark on Lustre

Nicholas Chaimov (U Oregon)
Costin Iancu, Khaled Ibrahim, Shane Canon, Jay Srinivasan (LBNL)
Allen Malony (U Oregon)

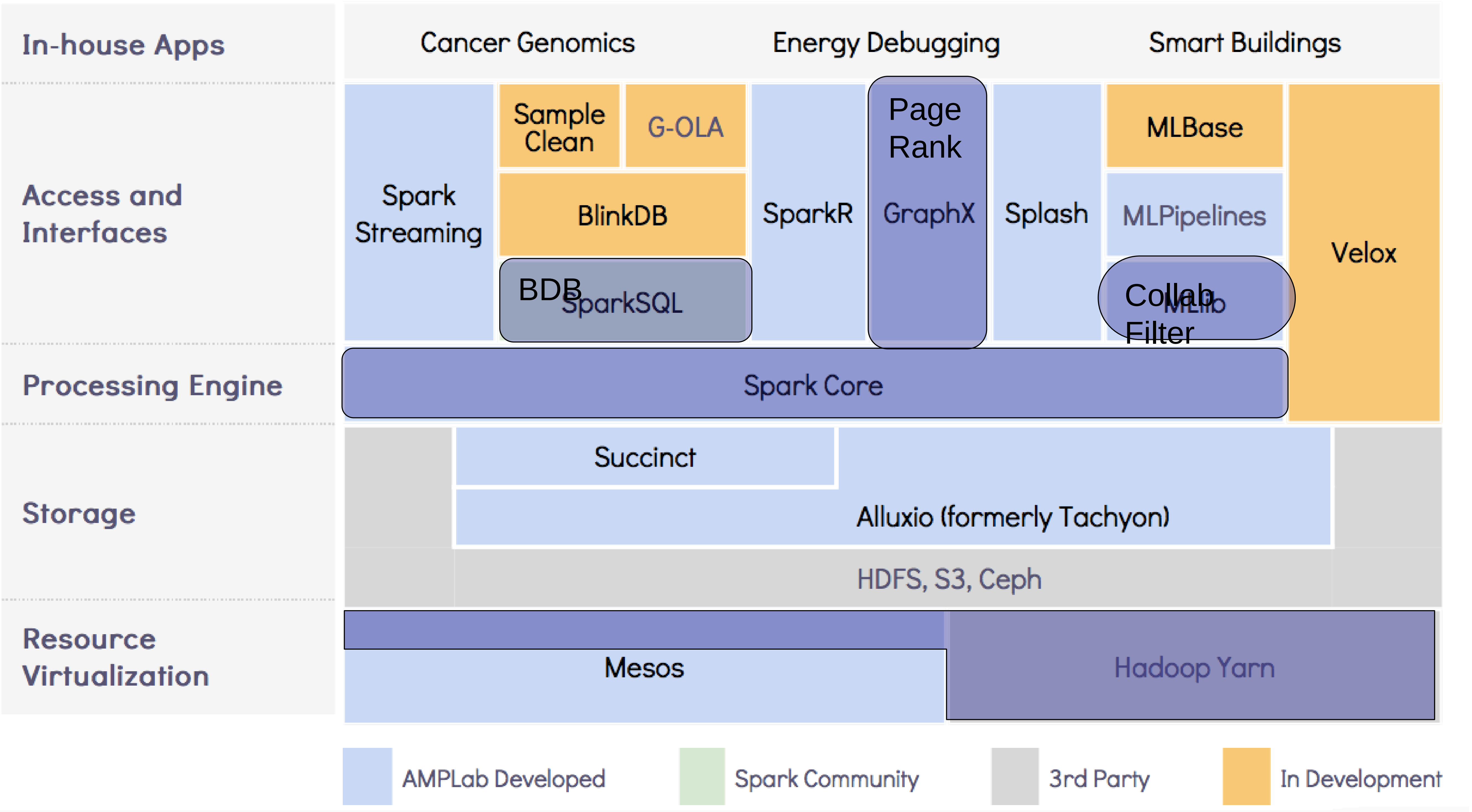
Intel Parallel Computing Center – Big Data Support for HPC

"Scaling Spark on HPC Systems" to appear in HPDC 2016

Data Analytics

- **Spark- “fast and general engine for large-scale data processing”**
- **Specialized runtime provides for**
  - Performance (☺)
  - Elastic parallelism
  - Resilience
- **Improves programmer productivity through**
  - HLL front-ends (Scala, R, SQL)
  - Multiple domain-specific libraries: Streaming, SparkSQL, SparkR, GraphX, Splash, MLLib, Velox
- **Developed for cloud environments, performance “satisfactory”**

Berkeley Data Analytics Stack



From https://amplab.cs.berkeley.edu/software/

HPC  Cloud

- **Differences in architecture and system design impact performance**
  - HPC == Centralized I/O vs. Cloud == Distributed I/O
  - Limited scalability in default configuration
  - HPC -> storage dominates vs Cloud -> network dominates
- **Differences in usage and expectations impact software design, and ultimately performance**
  - HPC enables Global Name Spaces
  - HPC practices encourage tightly coupled parallelism

What’s in Spark?

Spark

- **Central abstraction is the Resilient Distributed Dataset, or RDD.**
  - Composed of **partitions** of data
    - which are composed of **blocks**.
  - RDDs are created from other RDDs by applying **transformations** or **actions**.
  - Has a **lineage** specifying how its blocks are computed.
  - Requesting a block either retrieves from cache or triggers computation.

Word Count Example

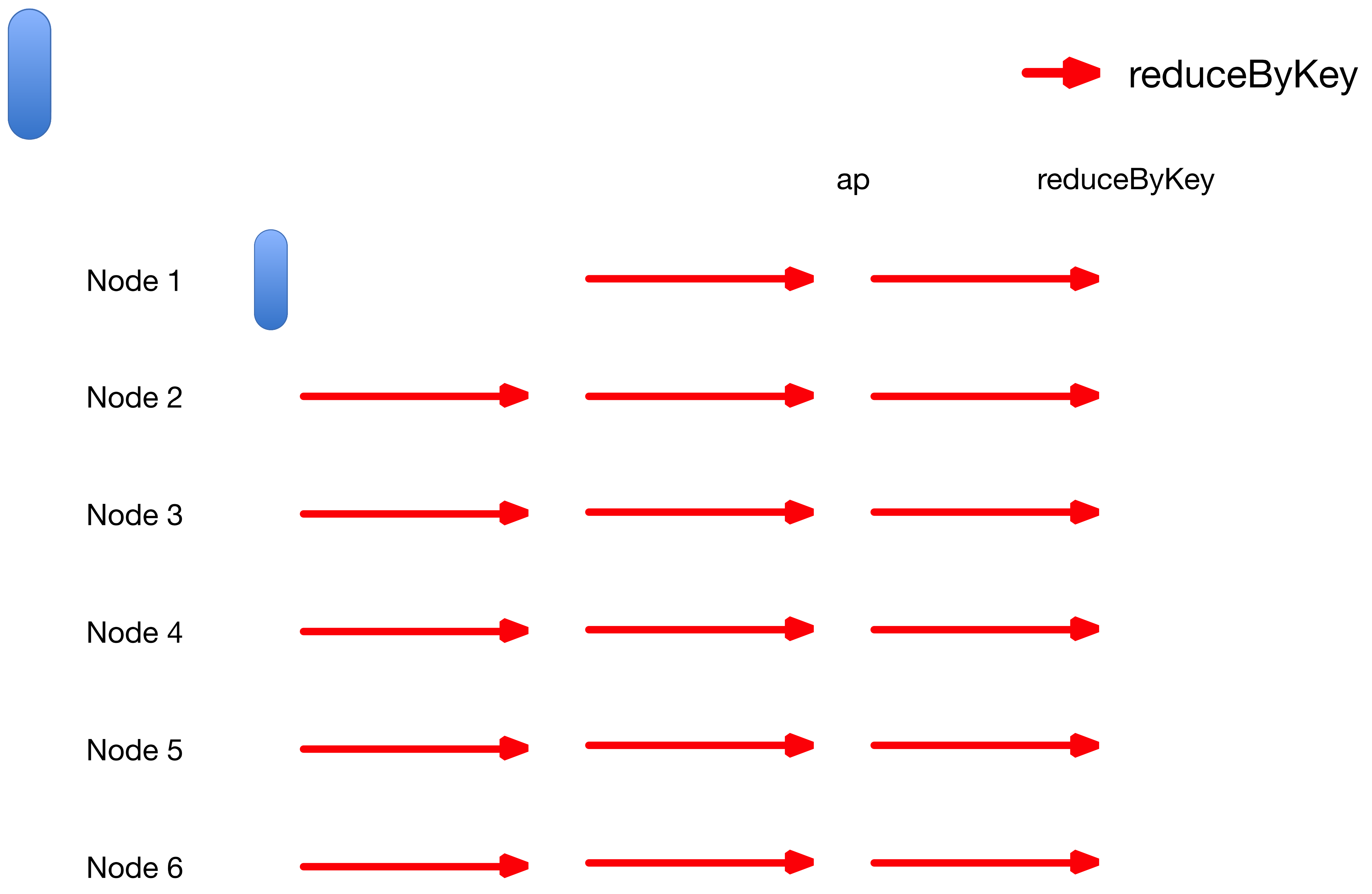
```
val textFile = sc.textFile(”input.txt")
val counts = textFile.flatMap(line => line.split(" "))
                     .map(word => (word, 1))
                     .reduceByKey(_ + _)
counts.collect()
```

textFile

- **Transformations** declare intent, do not trigger computation but simply build the lineage
  - *textFile*, *flatMap*, *map* and *reduceByKey* are **transformations**
- Actions trigger computation on parent RDD
  - *collect* is an **action**
- Data transparently managed by runtime

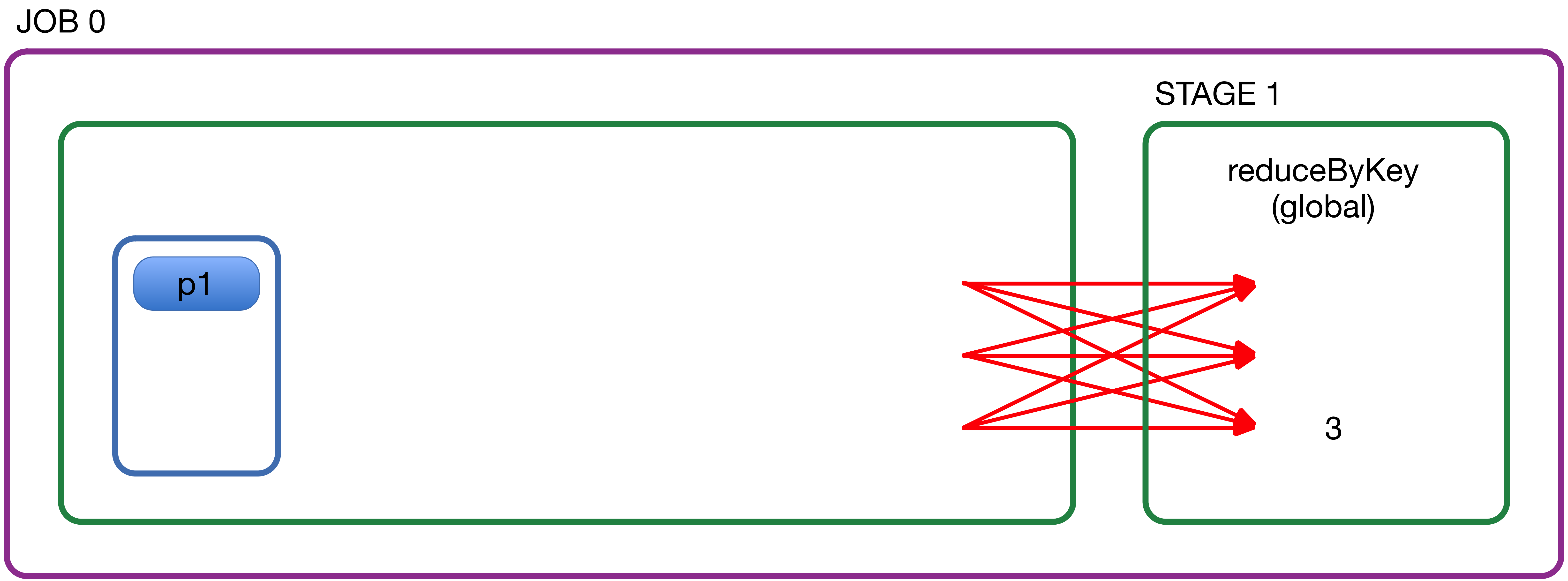
reduceByKey

Partitioning



Data is partitioned by the runtime

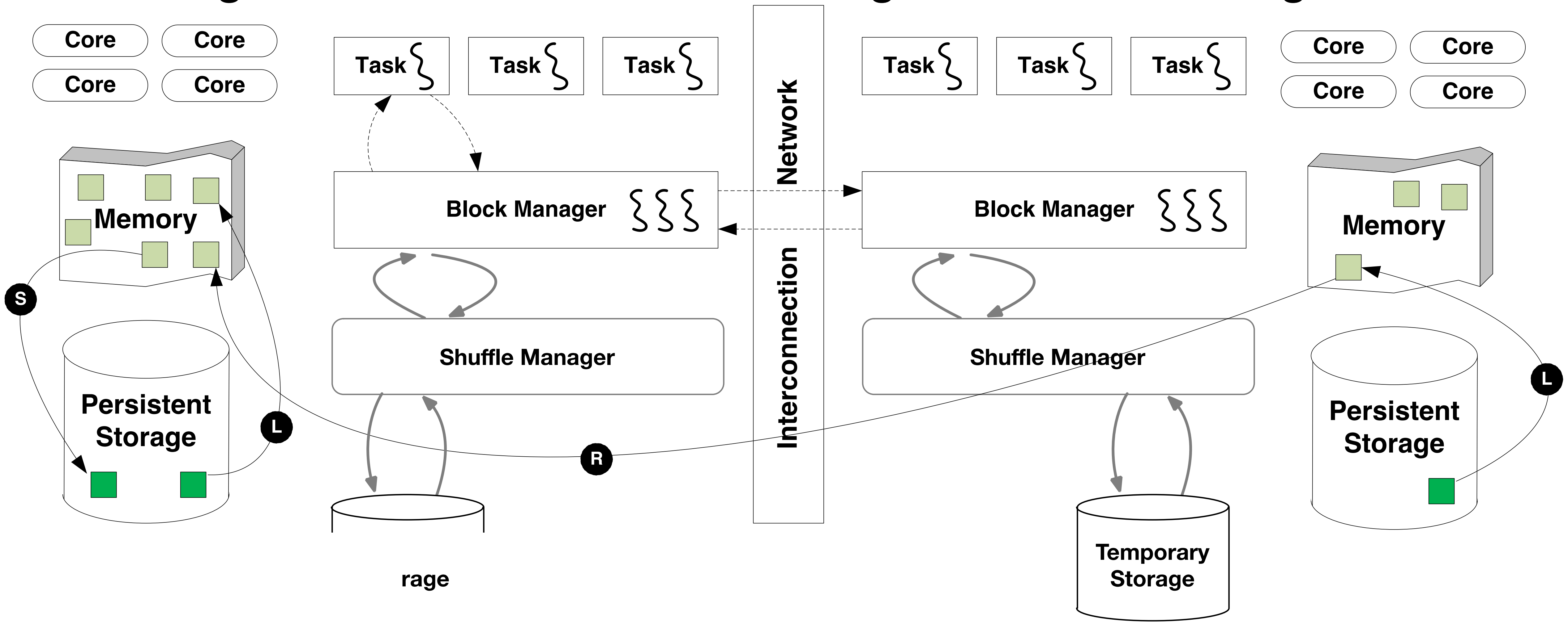
Stages



- Vertical data movement (local) between transformations
- Horizontal (remote) and vertical data movement between stages (shuffle)

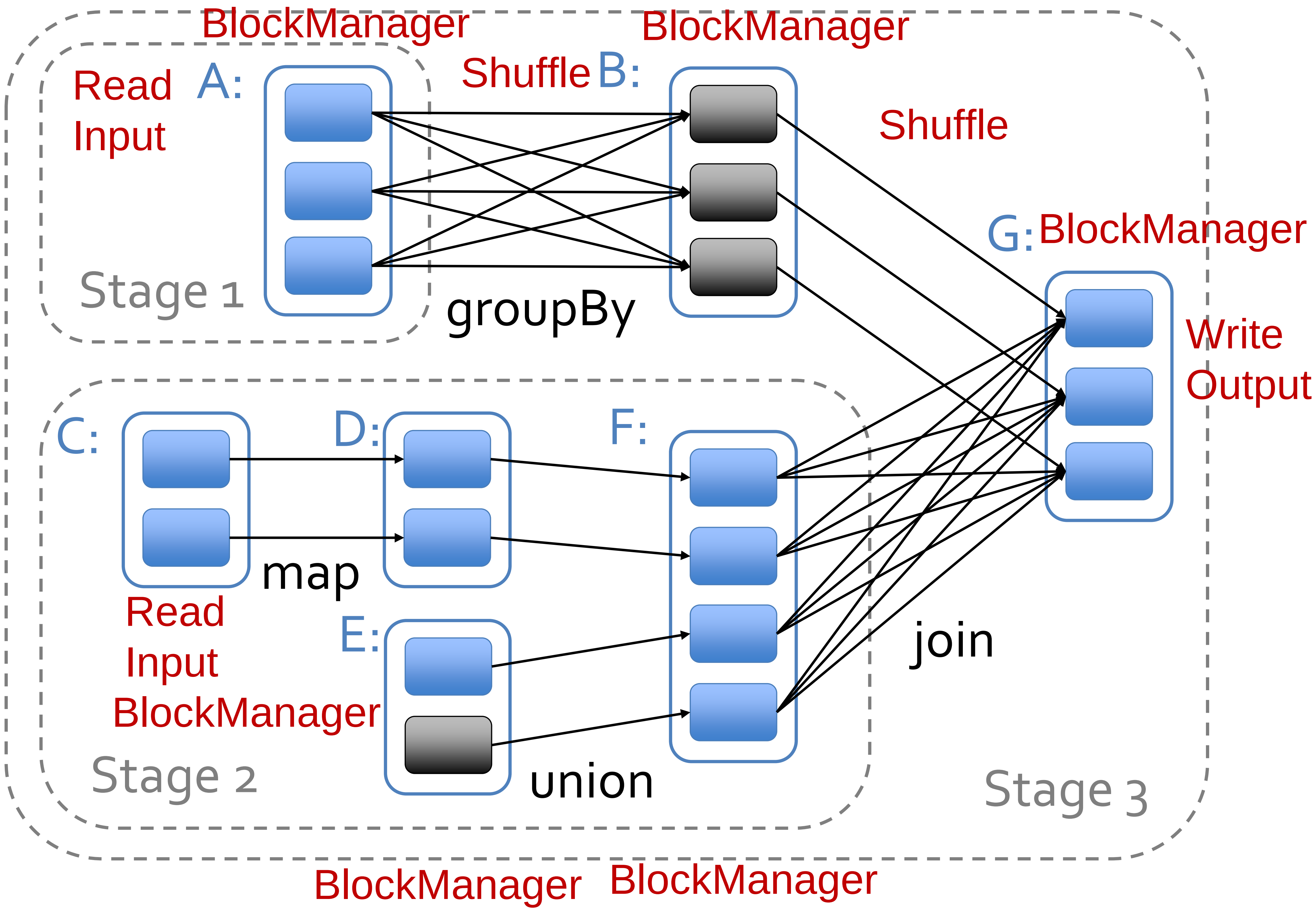
Data Movement in Spark

- **Block is the unit of movement and execution**
  - Ideally, blocks are resident in memory
  - Blocks are cached, evicts or spills to disk as necessary
  - If swapped, bring the block from a persistent storage
  - If remote, request the block from a remote manager
- **Shuffle Manager interacts with Block Manager and local storage**



I/O Happens Everywhere

- **Program input/output**
  - ***Expected to be distributed*** with global namespace (HDFS)
- ***Runtime Managed***
  - ***Expected to be local*** (Java FileOutputStream)
  - Shuffle and Block MAnager



Tuning Spark on Cray XC30

Design Assumptions

- **Spark expects**
  - Local disk with HDFS overlay for distributed file system
  - Fast local disk (SSD) for shuffle files
  - Assumes disk operations are fast

Clouds

Disk I/O optimized for latency

Network optimized for bandwidth

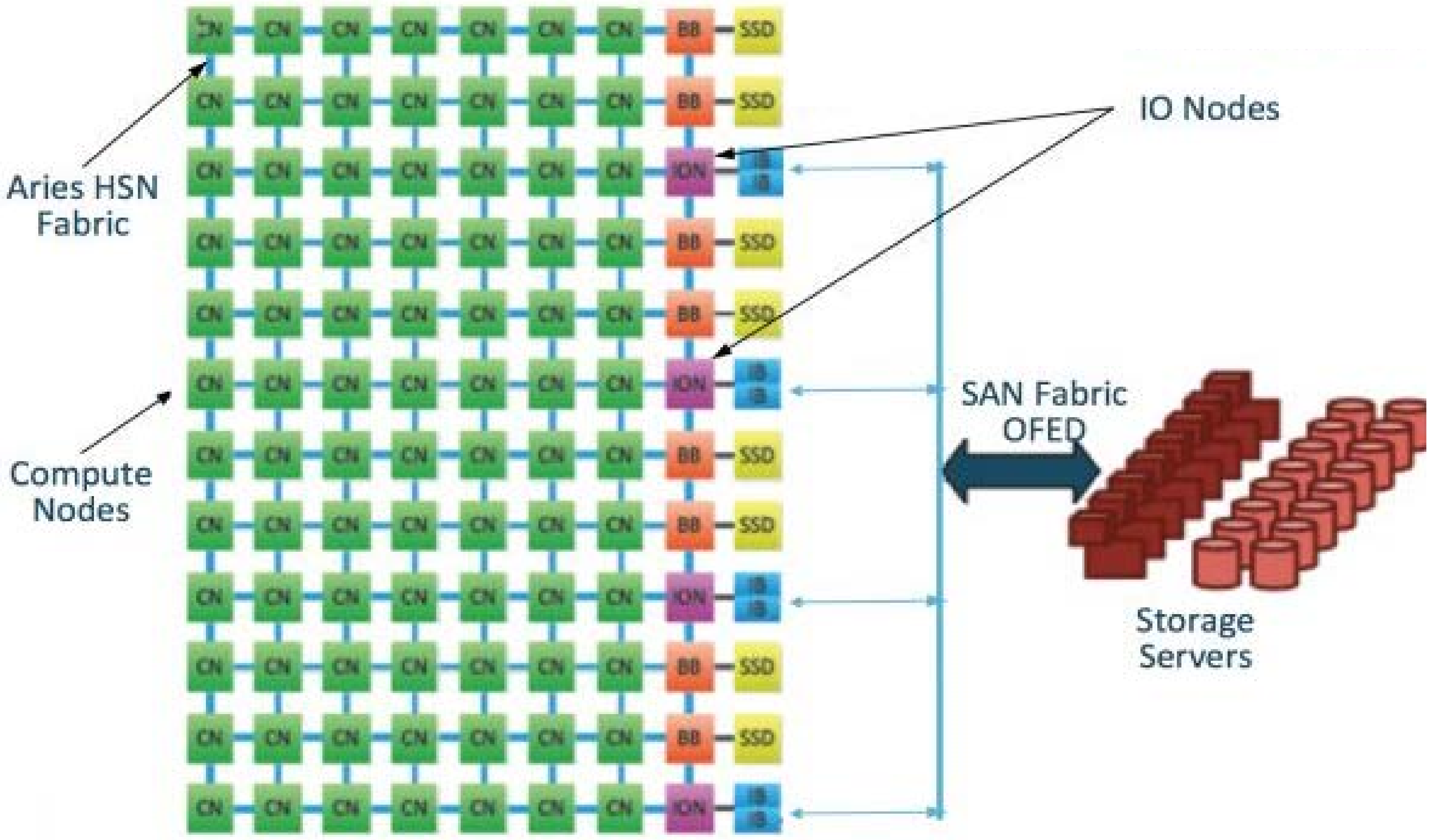
HPC

Disk I/O optimized for bandwidth

Network optimized for latency

Systems Evaluated

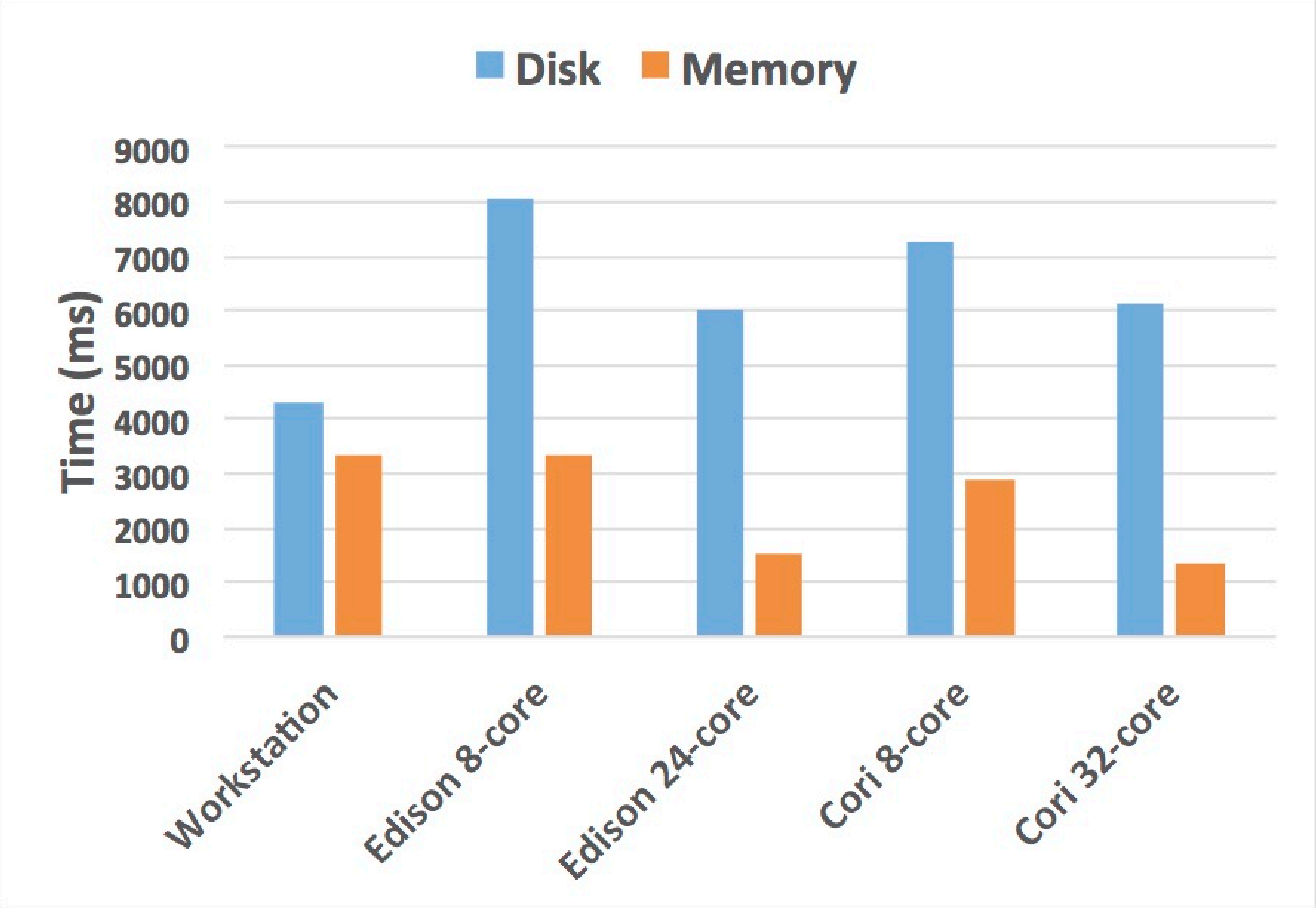
- Cray XC 30 at NERSC (Edison): 2.4 GHz Ivy Bridge
- Cray XC at NERSC (Cori): 2.3 GHz Haswell + Burst Buffer nodes
- Spark 1.5.0



Single Node Performance

- **Cray slower than workstation when data on disk**
  - **Same concurrency — 86% slower on Edison than workstation**
  - **All cores — 40% slower on Edison than workstation when using all cores (24)**

- **Cray matches workstation when data already cached**



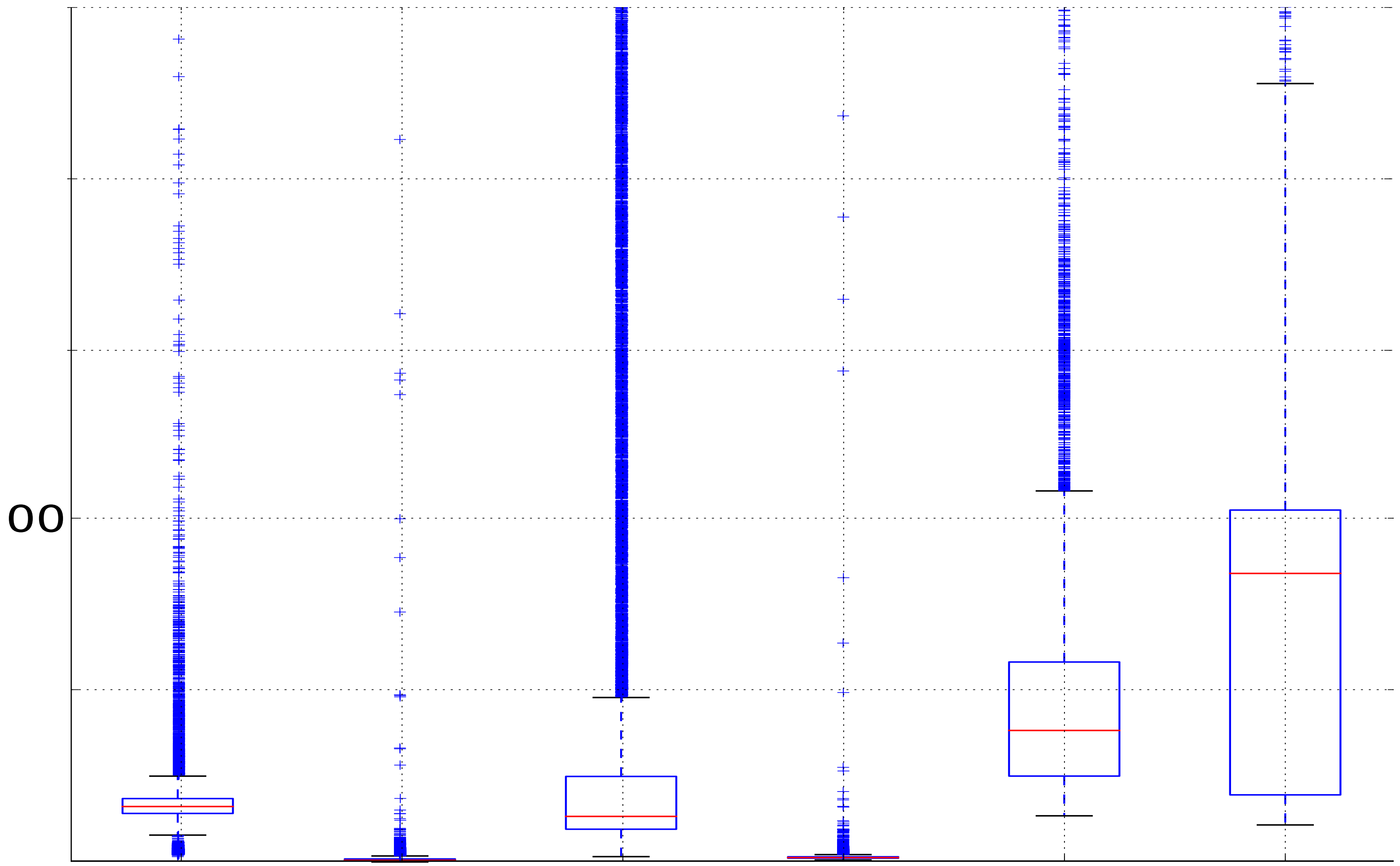
Spark SQL Big Data Benchmark

S3 Suffix	Scale Factor	Rankings **(rows)**	Rankings **(bytes)**	UserVisits **(rows)**	UserVisits **(bytes)**	Documents **(bytes)**
/5nodes/	5	90 Million	6.38GB	775 Million	126.8GB	136.9GB

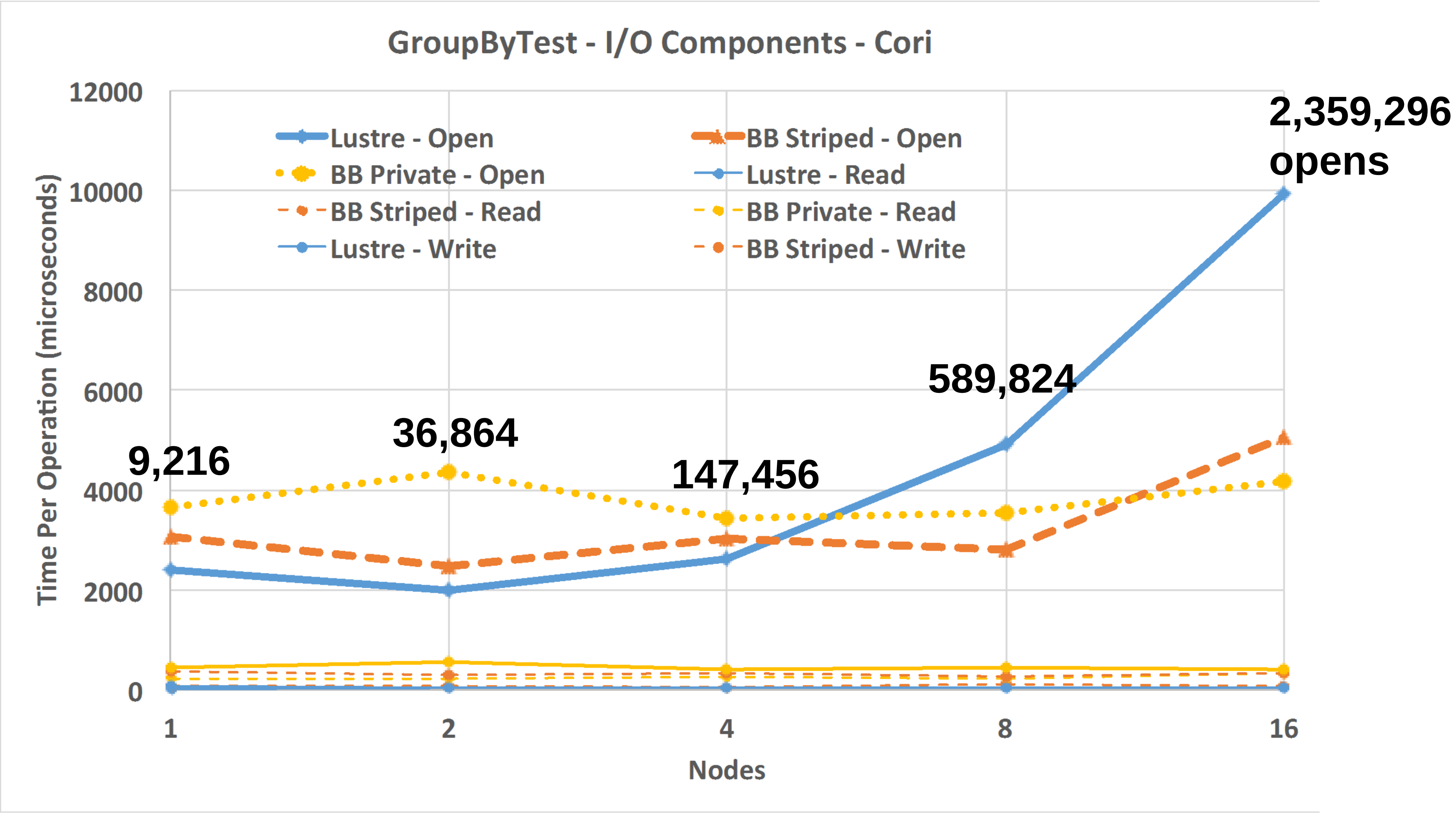
Disk I/O Overheads

Mean	23	1	32	1.2	43	59
Var	14,000	1	34,200	13.5	7,000	8,100

- Biggest difference between workstation and local disk occurs in **time per file open operation**
  - **Highly variable (14,000X larger on Edison Lustre)**
  - **Mean time per open 23x greater on Edison than workstation**



I/O Scalability



 # Shuffle opens = # Shuffle reads — O(cores^2) total

Time per open increases with scale, unlike read/write

Improving I/O Performance

- **Eliminate file operations that affect the metadata server**
  - Combine files within a node (currently per core combine)
  - Keep files open (cache `fopen`)
  - Use memory mapped local file system `/dev/shm` (cannot spill)
  - Use file system backed by single Lustre file (can spill)

- **Partial solutions that need to be used in conjunction**
  - Memory pressure is high is Spark due to resilience and poor garbage collection
  - Fopen() not necessarily from Spark (e.g. Parquet reader)
  - Third party layers not optimized for HPC/Lustre

File Pooling

- **Keep files open during shuffle**
  - User level pool of file stream objects
    - When file is opened, instead borrow from pool if possible
    - When file is closed, instead flush and return to pool
    - When pool is full, close least-recently-used idle stream
  - No distributed information is needed (node-level pools)
  - From $O(N^2)$ to $O(N)$ file opens
  - Can bound the size of the pool
    - Files = n * partitions per core * iterations
    - Max Opens = reader threads * files
  - Pool size limited
    - Need to allow for open files from JVM, other parts of Spark

File-Backed Filesystems

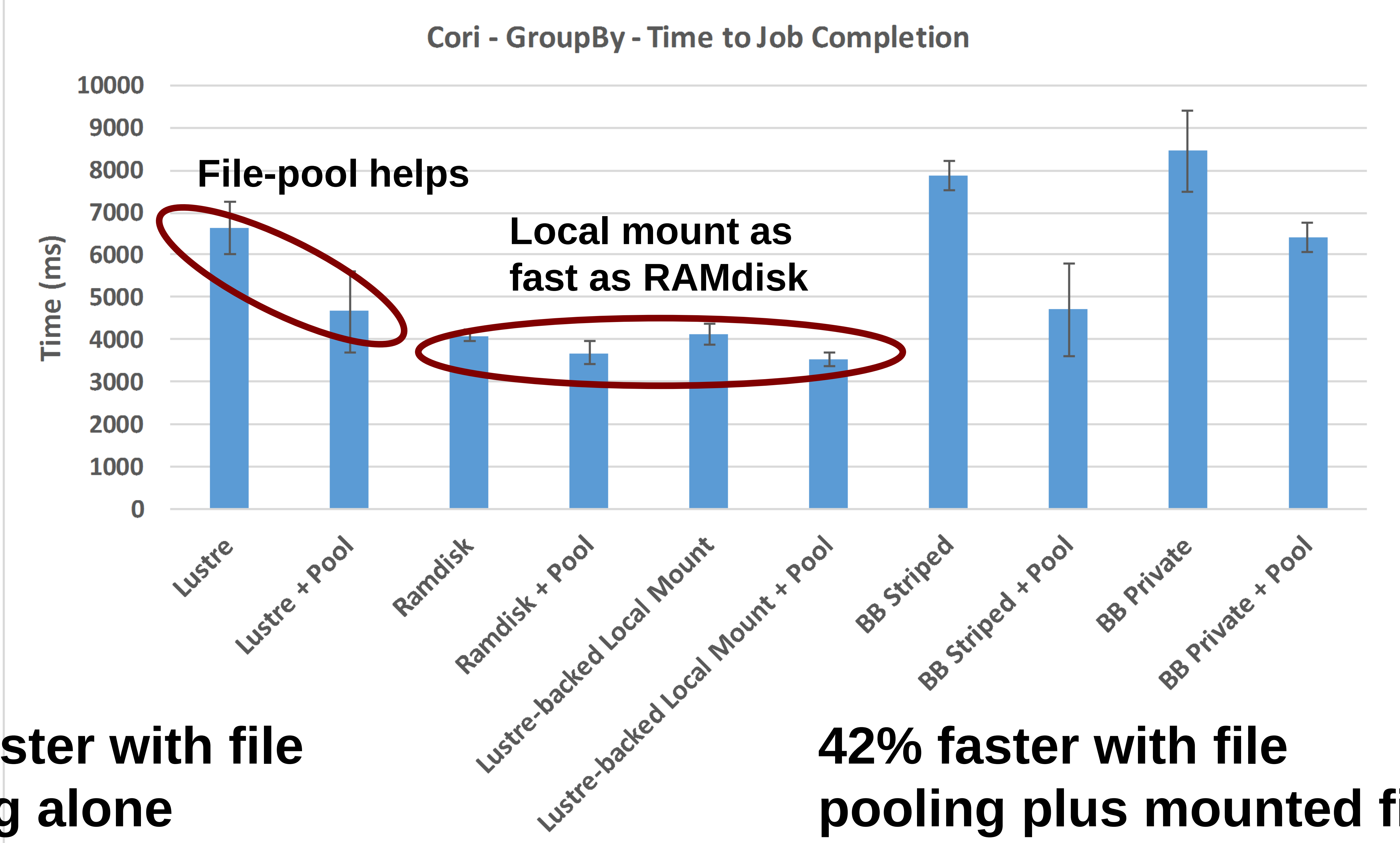
NERSC Shifter

- Lightweight container infrastructure for HPC
- Compatible with Docker images
- Integrated with Slurm scheduler
- Can control mounting of filesystems within container

Per-Node Cache

- `--volume=$SCRATCH/backingFile:/mnt:perNodeCache=size=100G`
- File for each node is created stored on backend Lustre filesystem
- File-backed filesystem mounted within each node's container instance at common path (`/mnt`)
- Single file open — intermediate data file opens are kept local

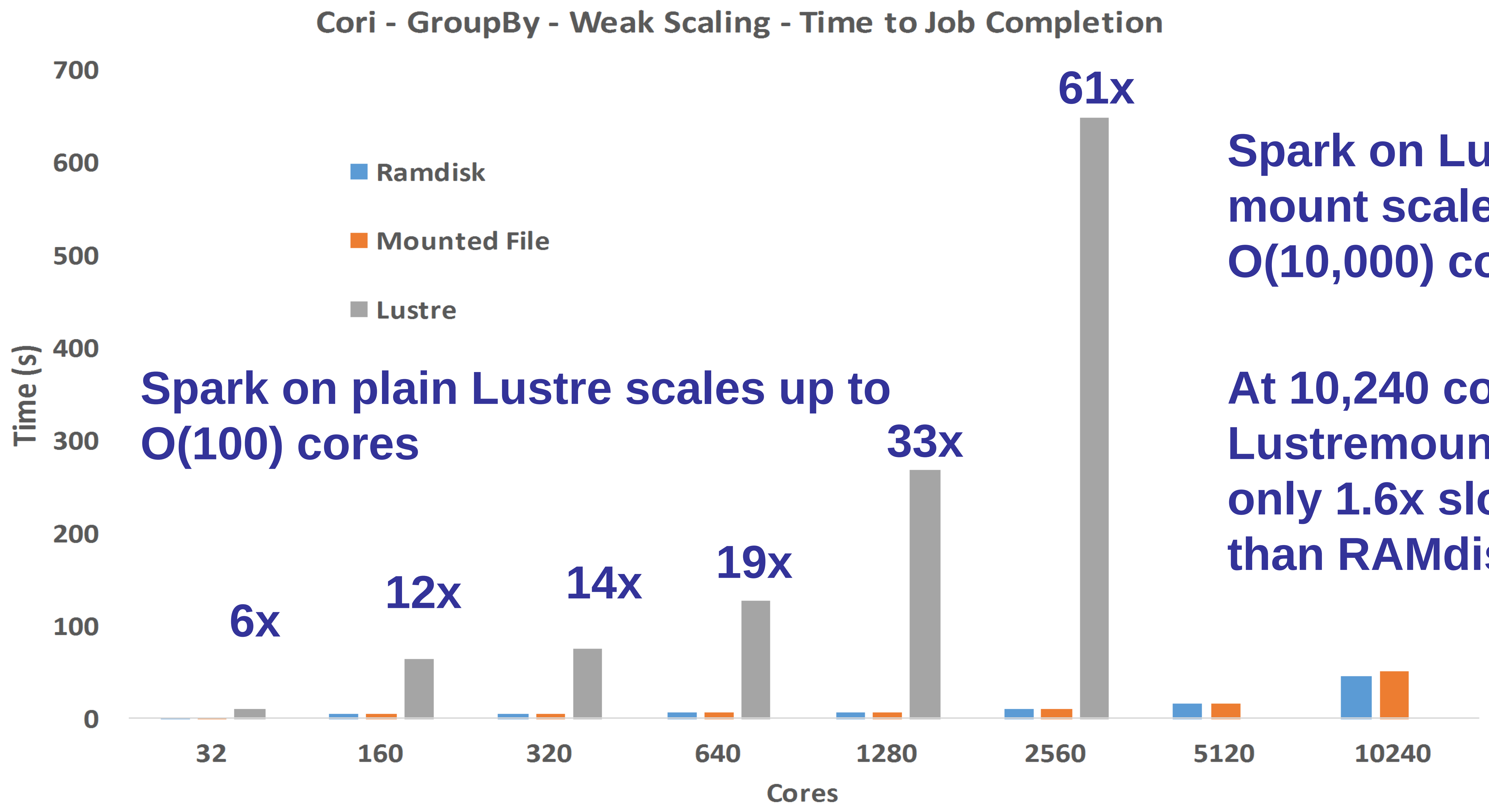
Single Node Performance



30% faster with file pooling alone

42% faster with file pooling plus mounted file

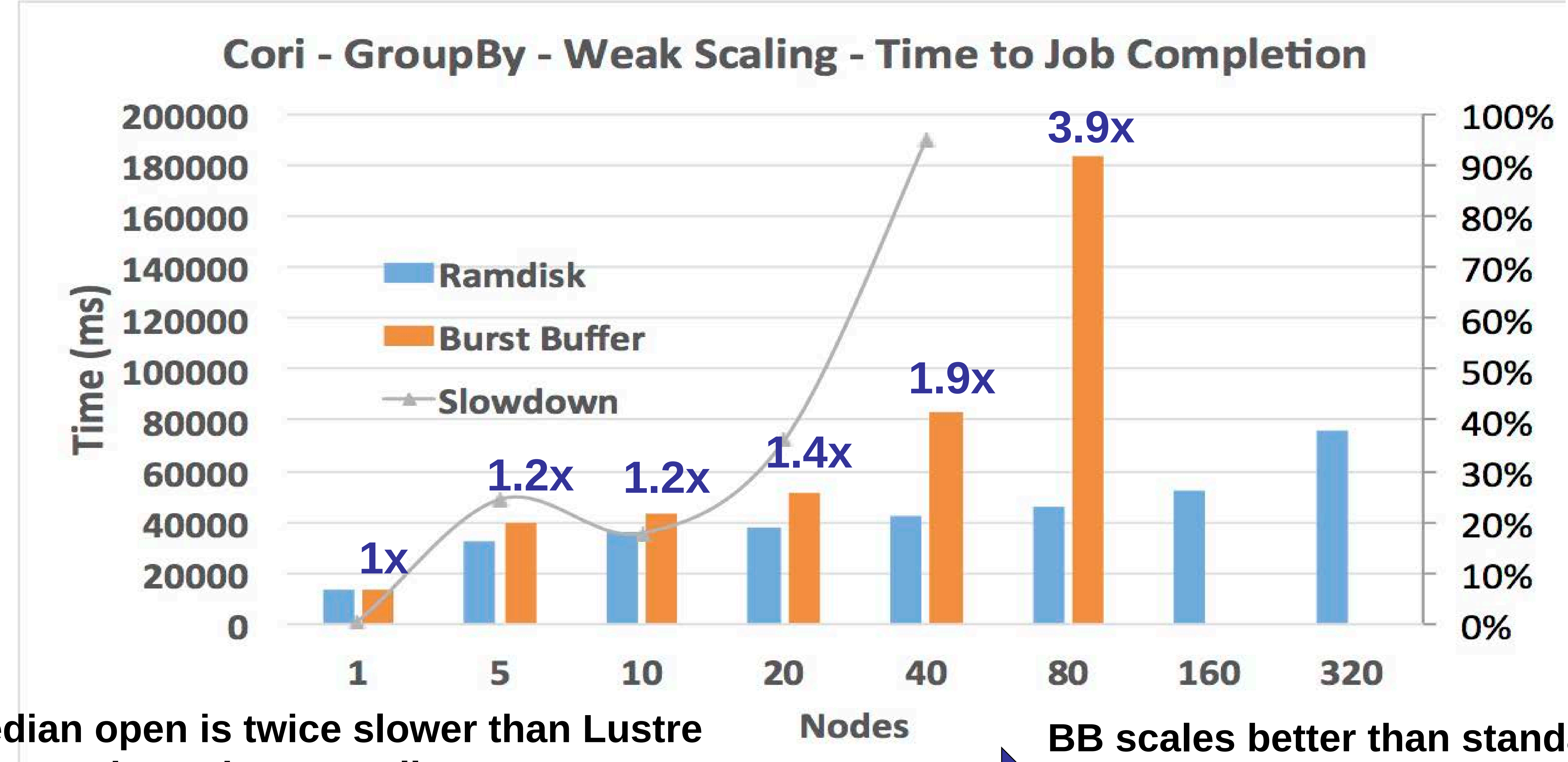
Scalability



Spark on Lustre mount scales up to O(10,000) cores

At 10,240 cores Lustremount is only 1.6x slower than RAMdisk

Optimizing for the Tail Helps



BB median open is twice slower than Lustre
BB open variance is 5x smaller

BB scales better than standalone Lustre

Conclusions

- **Very high rate of metadata operations in Spark limits scalability**
  - Disk I/O dominated Spark performance
- **Our solutions substantially improve scalability**
  - Default configuration scales up to O(100) cores
  - Lustre-mount improves to O(10,000) cores scalability
  - File-pooling adds another 10%
  - File-pooling improves performance even for RAMdisk
- **NERSC and Cray are already using our solutions**
  - File-mounting capabilities built into Shifter

Acknowledgements

- Intel Parallel Computing Center – Big Data Support on HPC

- This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Thanks!