Java is a trademark of Sun Microsystems, Inc.

Java

JavaOne℠

Hadoop, a Highly Scalable, Distributed File & Data Processing System implemented in Java

Sanjay Radia

Yahoo Inc

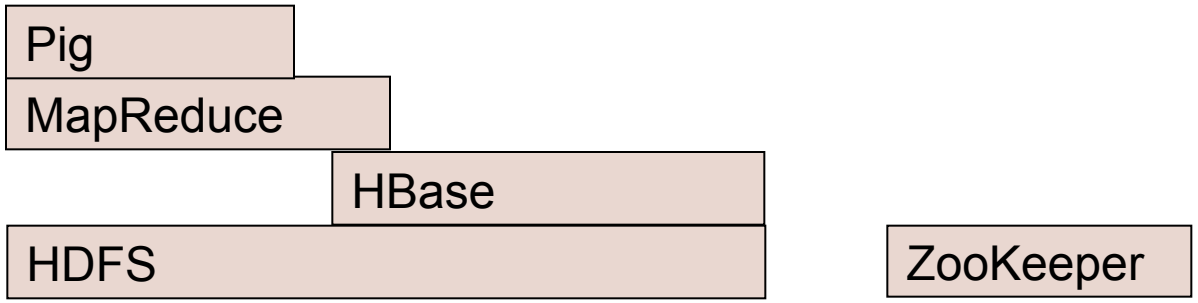
Cloud Computing & Data Infrastructure

Outline

- Overview
- Storage: HDFS
- Processing: MapReduce & Pig
- Hadoop and Java

Hadoop

- A framework for storing & processing Petabyte of data using commodity hardware and storage
- Storage: HDFS, HBase
- Processing: MapReduce, Pig
- Sister project: Zookeeper – distributed coordination



Hadoop Characteristics

- Commodity HW + Horizontal scaling
  - Add inexpensive servers with JBODS
  - Storage servers and their disks are ***not*** assumed to be highly reliable and available
- Use replication across servers to deal with unreliable storage/servers
- Metadata-data separation - simple design
  - Storage scales horizontally
  - Metadata scales vertically (today)
- Slightly Restricted file semantics
  - Focus is mostly sequential access
  - Single writers
  - No file locking features
- Support for moving computation close to data
  - i.e. servers have 2 purposes: data storage and computation
- MapReduce Data processing framework

Simplicity of design

why a small team could build such a large system in the first place

Hadoop
Cost effective Scalable Storage & Processing Grid

- Horizontally scale storage, IO Bandwidth, and CPU
  - Add commodity servers with JBODs: Each server adds
    - 4TB-12TB raw == 1TB-3TB effective
    - IO: 20MB/s - 100MB/s
- Can add/upgrade servers over time
- *The servers are also available for application computation*

- Equivalent cost systems:
  - Hadoop System
    - 750 servers (3K Cores) - both storage and computation
    - .75 PB effective storage - grows by simply adding bigger disks or servers over time
    - >> 15GB/s (Peak 75GB/s !!)
  - Oracle Rac + EMC solution
    - 0.3 PB
    - 8 Servers (32 cores)
    - 6 GB/s

Hadoop, an Open Source Project

- Implemented in Java
- Apache Top Level Project http://hadoop.apache.org/core/
- Community of contributors is growing
  - Yahoo: HDFS and MapReduce
  - Powerset: HBase
  - Facebook: Hive and FairShare scheduler
  - IBM: Eclipse plugins

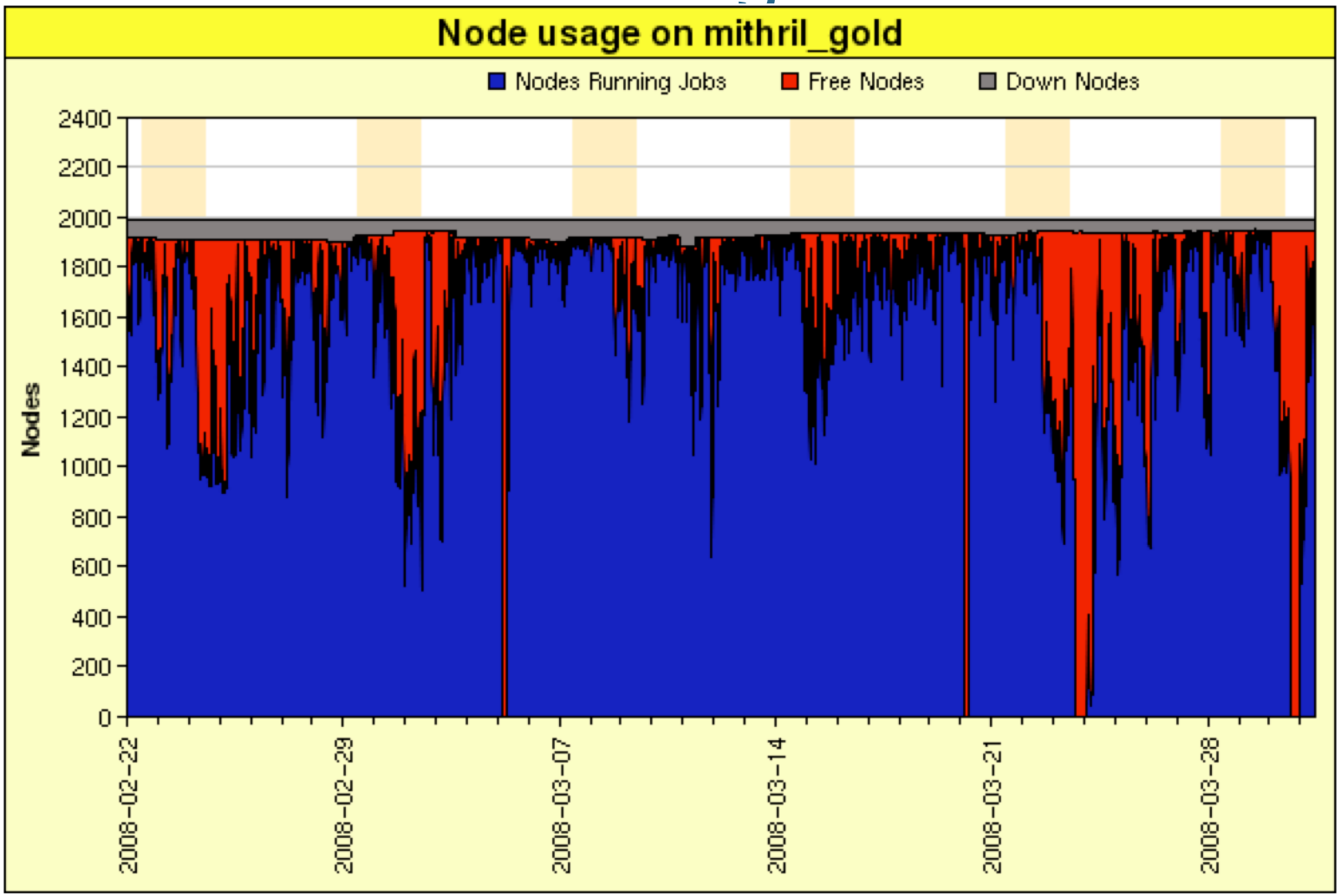
Hadoop Users

- **Commercial: Clusters:** few nodes to 3k nodes
  - Yahoo, Last.fm, Joost, Facebook, A9, …
  - Production use at Yahoo
    - An internal cloud service: 20K nodes, each cluster up to 3K nodes
  - Hadoop as a public cloud service: Amazon Elastic MapReduce
- **Academic:**
  - IBM/Google cloud computing initiative
    - A 40 node cluster + Xen based VMs …
  - M45/Yahoo Academic Cluster
    - 500 node cluster
    - CMU and other universities
- **Conferences:**
  - Hadoop Summit hosted by Yahoo! March 2008, June 2009
  - Whole day Hadoop tracks at Apache Con

Hadoop clusters

- Yahoo has ~20,000 machines running Hadoop
- Our largest clusters are currently 3000 nodes
- Several Petabyte of user data (compressed, unreplicated)
- We run hundreds of thousands of jobs every month
- Load 30-50TB/day

Research Cluster Usage

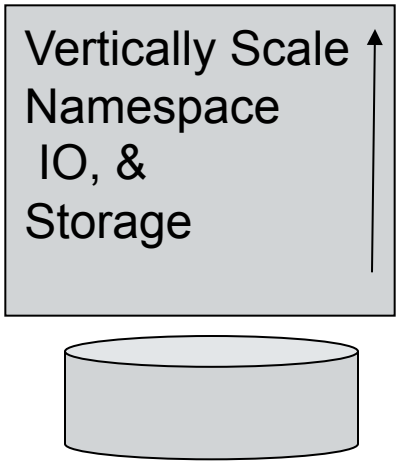


The Storage:

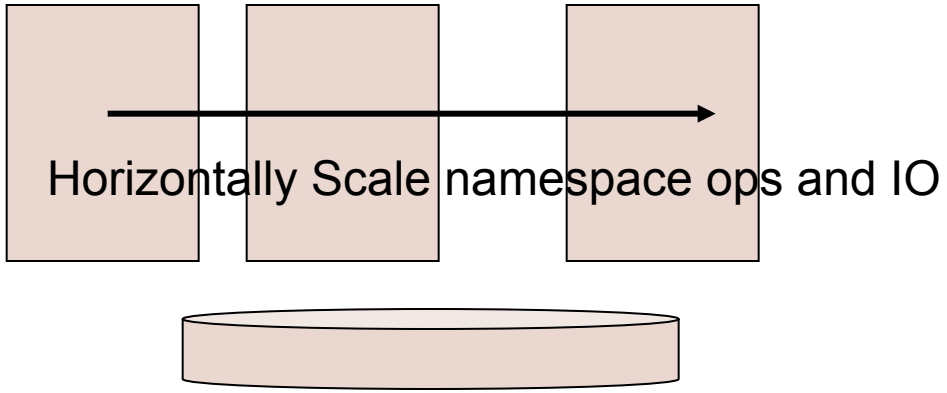
HDFS

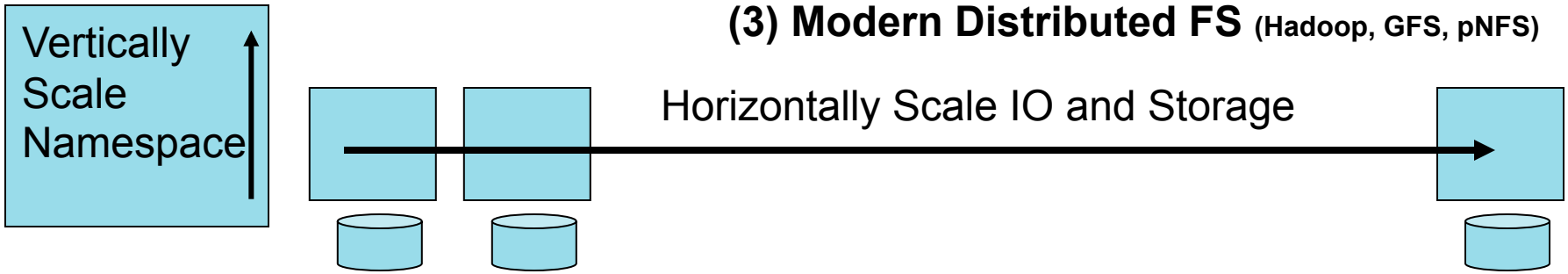
File Systems Background(1) : Scaling

(1) Mainframe FS



(2) Single System Image Distributed FS





(3) Modern Distributed FS (Hadoop, GFS, pNFS)

File Systems Background(2)
Federating Namespaces

- Newcastle connection (1982)
  - /.. Mounts
    - (plus remote Unix semantics)
- Andrew (1988)
  - Federated mount of file systems on /afs
    - (plus local caches and disconnected operations)
- Many others ….

File systems Background (3): leading to Google FS and HDFS

- ***Separation of metadata from data - 1978, 1980***
  - "Separating Data from Function in a Distributed File System" (1978)
    - by J E Israel, J G Mitchell, H E Sturgis
  - "A universal file server" (1980) by A D Birrell, R M Needham
- ***Horizontal scaling of storage nodes and io bandwidth (1999-2003)***
  - Several startups building scalable NFS – late 1990s
  - Luster (1999-2002)
  - Google FS (2003)
  - (Hadoop's HDFS, pNFS)
- ***Commodity HW with JBODs, Replication, Non-posix semantics***
  - Google FS (2003)
- ***Computation close to the data***
  - Parallel DBs
  - Google FS/MapReduce

URI-based + Pluggable File Systems - HDFS is the main FS

- URI-based file names
  - Access any hadoop fs on the network: hdfs://nn_ip:port/path
  - Can set your default file system: simply /foo/bar/…

- ***FileSystem.java*** - Virtual FS for impl & accessing files
  - FileSystem has multiple implementations:
    - Hdfs: “hdfs://” Local file system: “file://”
    - Amazon S3: “s3://” Kosmos “kfs://”
    - Archive “har://”
  - MapReduce uses the FileSystem interface
    - hence MR can run on different FSs

- *Note the similarities to JSR 203*
  - *a HDFS plugin for JSR 203 is in progress*

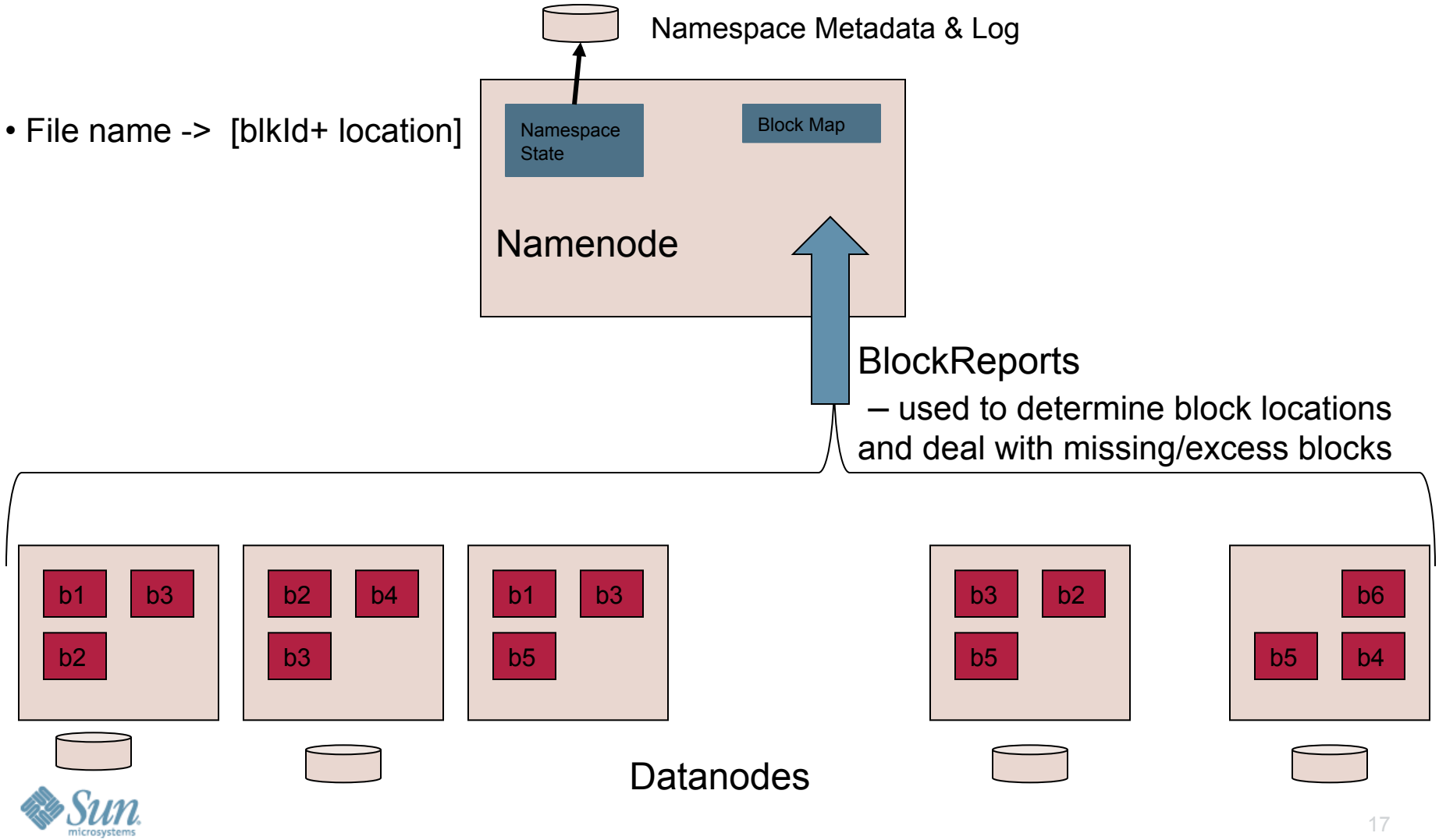
HDFS: Directories, Files & Blocks

- Data is organized into files and directories
  - Files are divided into uniform sized blocks and distributed across cluster nodes
- HDFS (& FileSystem) exposes block placement so that computation can be migrated to data
- Blocks are replicated to handle hardware failure
- HDFS keeps checksums of data for corruption detection and recovery

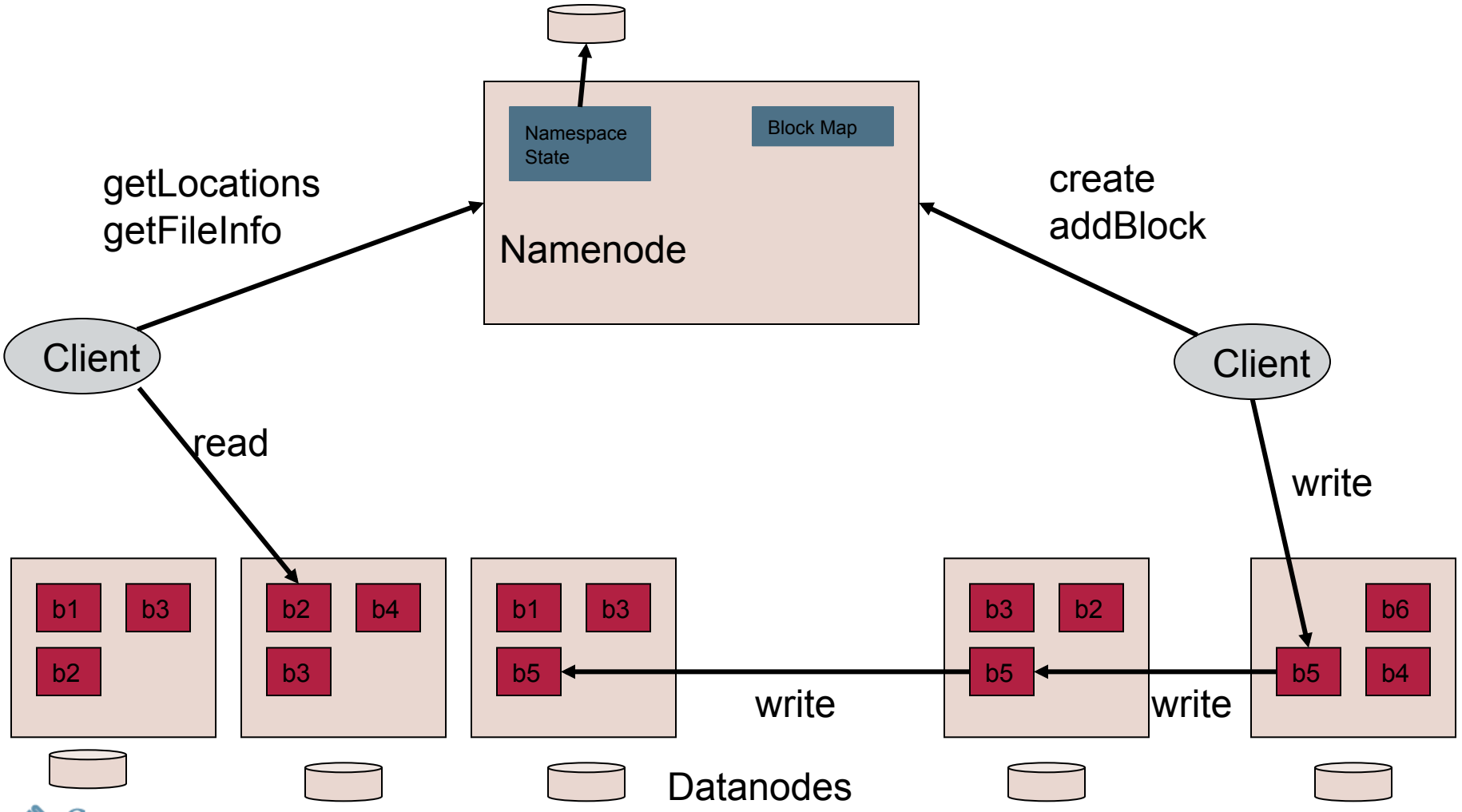
HDFS Architecture (1)

- Files broken into blocks of 128MB (per-file configurable)
- **Namenode (currently single)**
  - Manages the file namespace
  - File name to list blocks + location mapping
  - File metadata (i.e. “inode”)
  - Authorization and authentication
  - Collect block reports from Datanodes on block locations
  - Replicate missing blocks
  - Implementation detail:
    - Keeps ALL namespace in memory plus checkpoints & journal
      - 60M objects on 16G machine (e.g. 20M files with 2 blocks each)
- **Datanodes (thousands) handle block storage**
  - Clients access the blocks directly from data nodes
  - Data nodes periodically send block reports to Namenode
  - Implementation detail:
    - Datanodes store the blocks using the underlying OS’s files

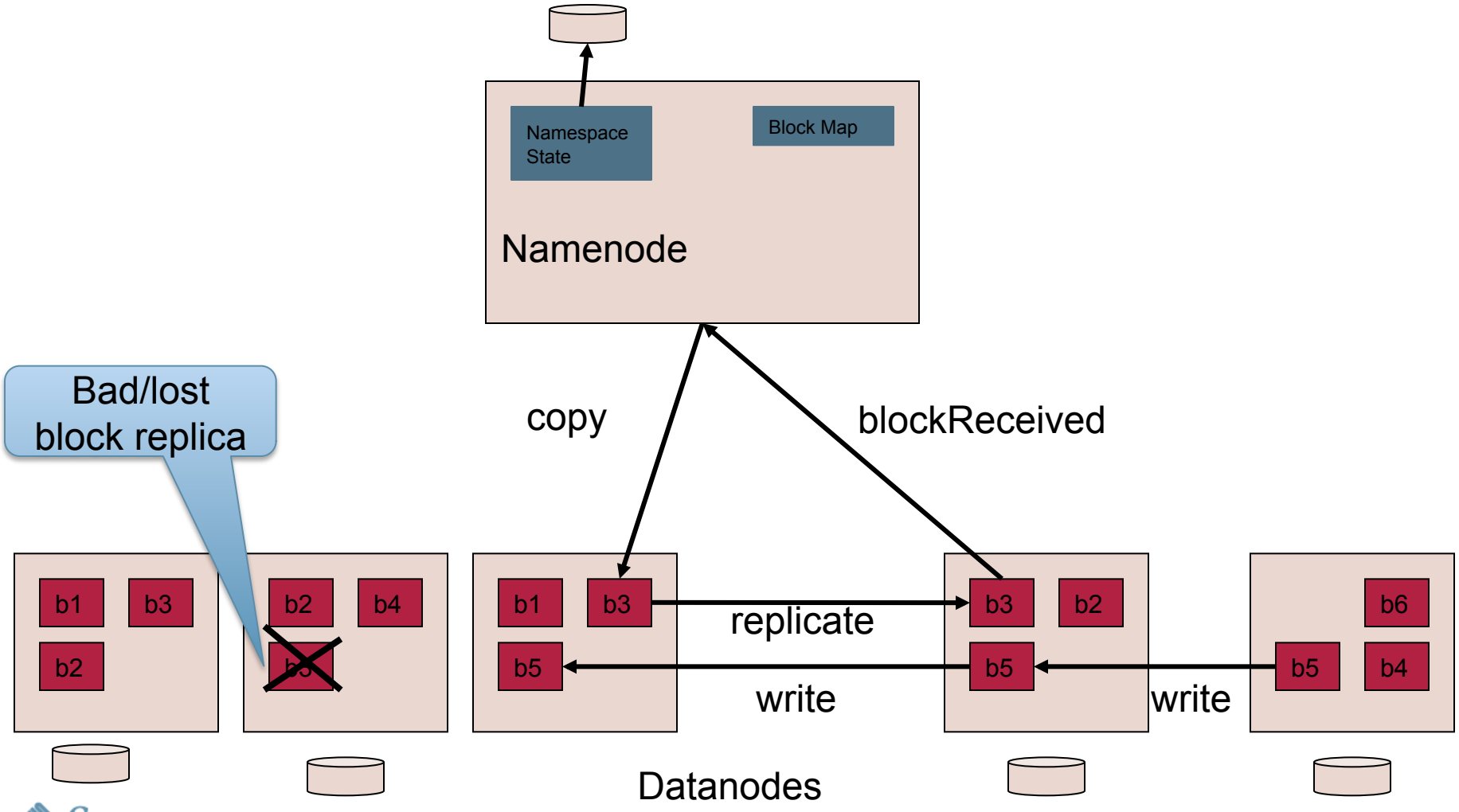
HDFS Architecture (2)



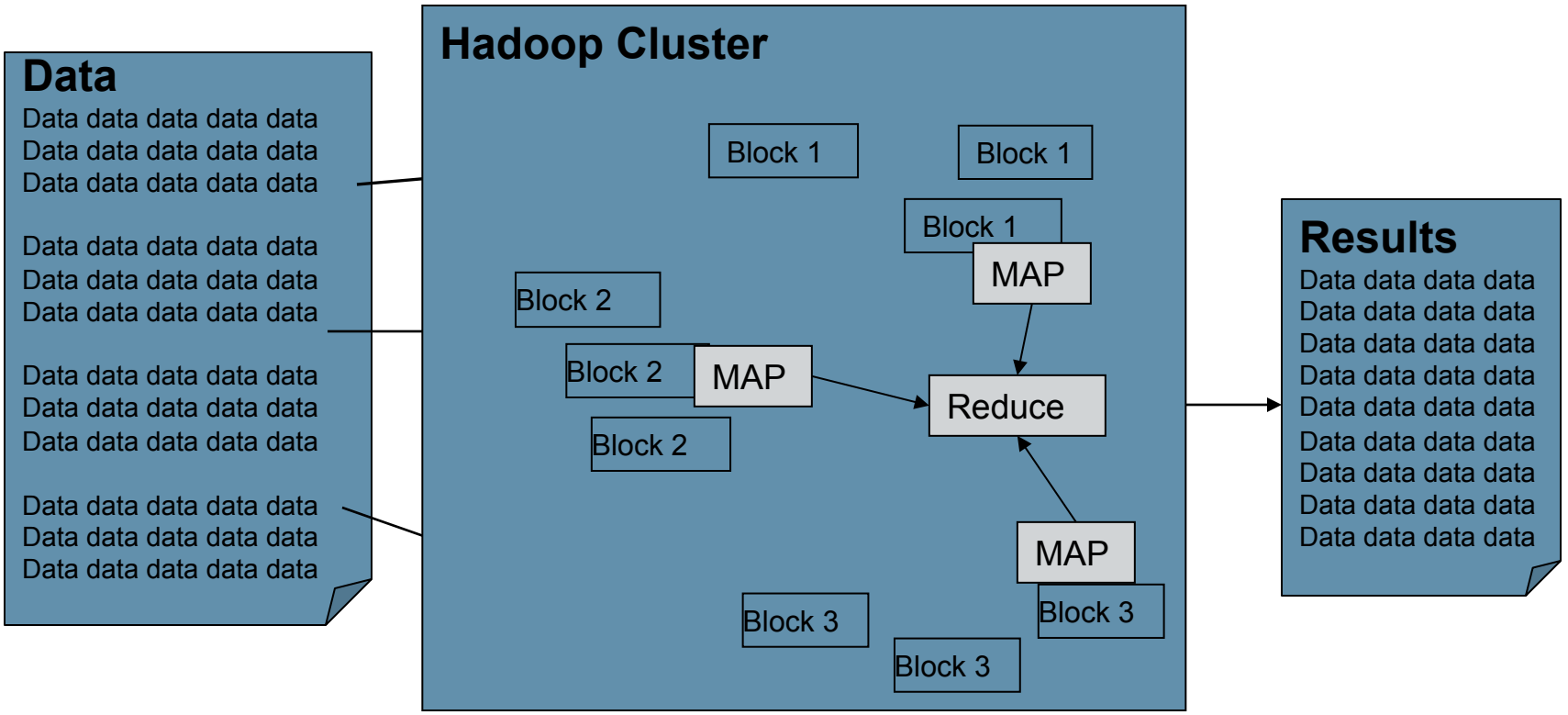
HDFS Architecture (3)



HDFS Architecture (4)



HDFS Architecture (5): Computation close to the data



HDFS Reads, Writes, Block Placement and Replication

- Reads: from the nearest replica
- Writes: pipelined to block replicas
  - Append – being implemented
    - Concurrent appends are likely to happen in the future
  - No plans for random writes so far.

- Replication and block placement
  - A file's replication factor can be changed dynamically (default 3)
  - Block placement is rack aware
  - Block under-replication & over-replication is detected by Namenode
    - triggers a copy or delete operation
  - Balancer application rebalances blocks to balance DN utilization

Details: The Protocols

- Client to Namenode
  - RPC
- Client to Datanode
  - Streaming writes/reads
    - On reads data shipped directly from OS
  - Considering RPC for pread(offset, bytes)
- Datanode to Namenode
  - RPC (heartbeat, blockReport …)
  - RPC Reply is the command from Namenode to Datanode (copy block …)
- RPC
  - Not cross language but that was the goal (hence not RMI …)
  - Current rpc is quite good
    - manages queues/buffers and spikes well
  - Upcoming Avro serialization will deal with versioning

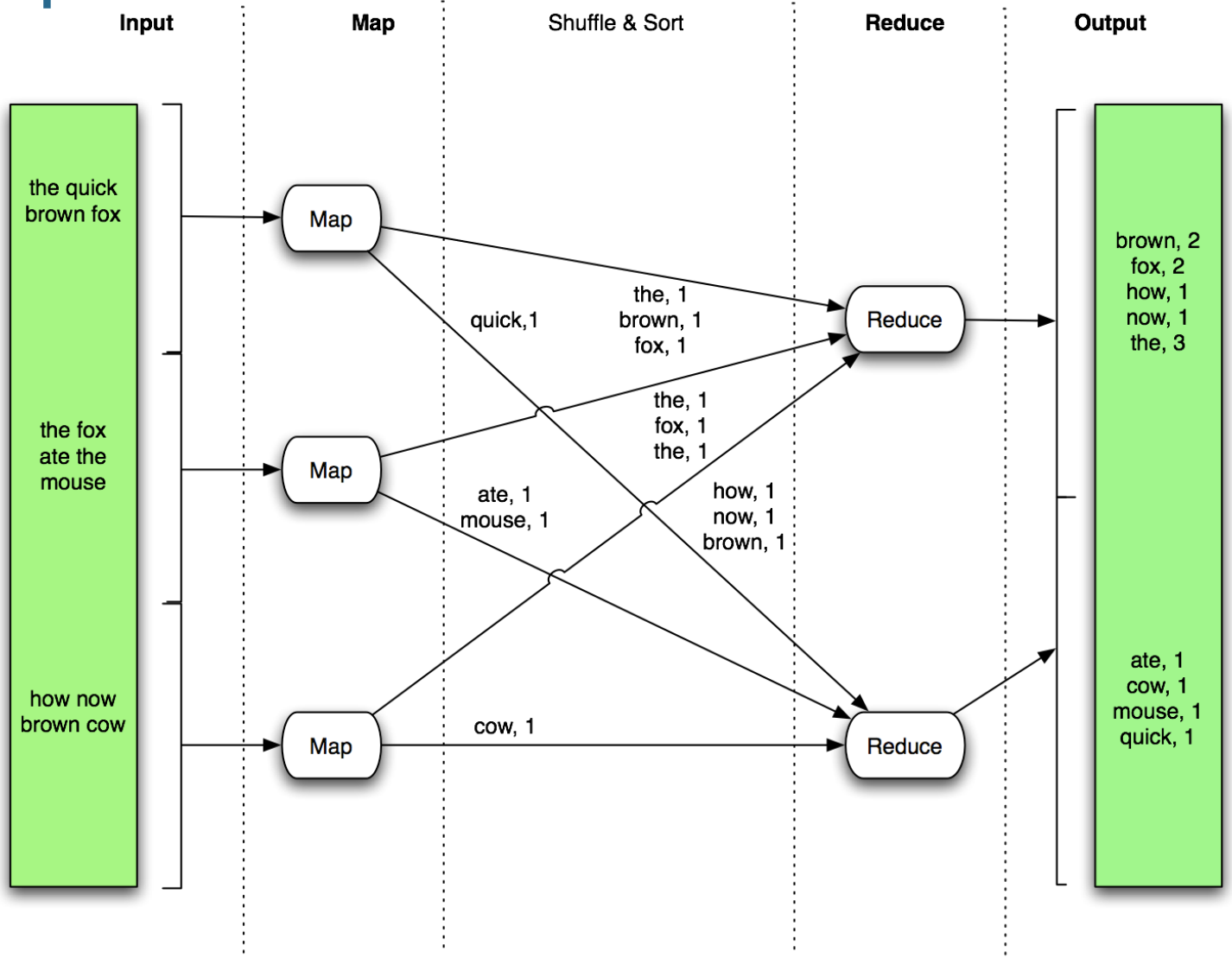
The Processing:

MapReduce

MapReduce

- MapReduce: programming model for efficient & reliable distributed computing
- Like a Unix pipeline:
  - cat input | grep | sort | uniq -c | cat > output
  - Input | Map | Shuffle & Sort | Reduce | Output
- Efficiency from
  - Streaming through data, reducing seeks
  - Pipelining
- A good fit for a lot of applications
  - Log processing
  - Web index building
  - Data mining and machine learning
- Example: Terasort competition:
  - 2008 and 2009 winner – 2008: 217 sec
    - Terabyte sort :2008 time : 209 sec, 2009 time: 62 sec
    - Petabyte sort: 975 min

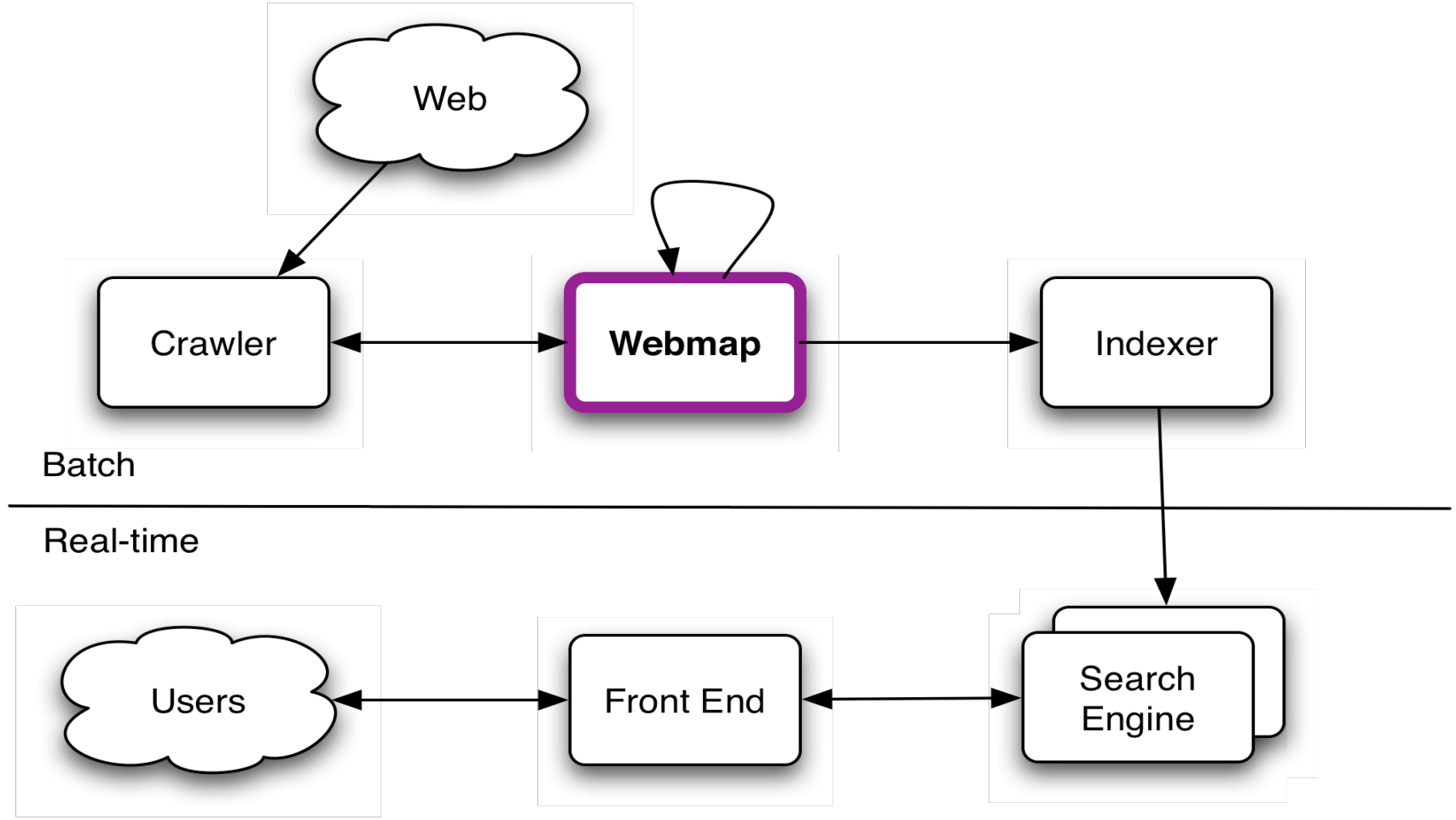
Example: Word Count Dataflow



MapReduce features

- Fine grained Map and Reduce tasks
  - Improved load balancing
  - Faster recovery from failed tasks
- Automatic re-execution on failure
  - In a large cluster, some nodes are always slow or flaky
  - Framework re-executes failed tasks
- Locality optimization
  - With large data, bandwidth to data is a problem
  - MapReduce + HDFS is a very effective solution
  - MapReduce queries HDFS for locations of input data
  - Map tasks are scheduled close to inputs when possible
- Pluggable Scheduler
  - Yahoo: Guaranteed-Capacity scheduler
  - Berkeley: FairShare scheduler (in use at Facebook)

Example: Search Dataflow



Yahoo's Production WebMap

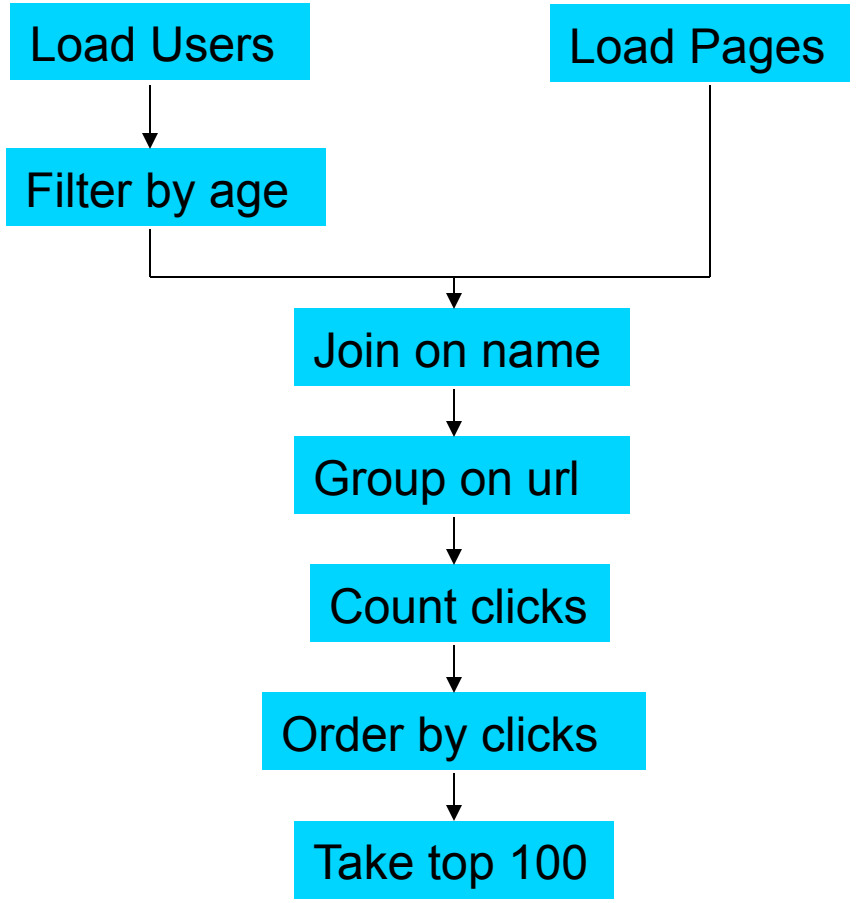
- Search needs a graph of the "known" web
  - Invert edges, compute link text, whole graph heuristics
- Periodic batch job using MapReduce
  - Uses a chain of ~100 MapReduce jobs
- Scale
  - 100 billion nodes and 1 trillion edges
  - Largest shuffle is 450 TB
  - Final output is 300 TB compressed
  - Runs on 10,000 cores
- Written mostly using Hadoop's C++ interface

Pig

- A high level language that is automatically translated into a series of MapReduce jobs

Pig: An Example Problem

Suppose you have user data in one file, website data in another, and you need to find the top 100 most visited sites by users aged 18 - 25.



In Pig Latin

```
Users = load 'users' as (name, age);
Fltrd = filter Users by
        age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
       COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top100 = limit Srtd 100;
store Top100 into 'top100sites';
```

HADOOP AND JAVA

Factors in Hadoop's success:

- Simple design
- The Team and Yahoo's support
- Java
  - Reliability and Stability of the system
  - Why a small team could deliver the platform in 3 years
  - *But Java has provided some challenges …*

Memory management

- What we do:
  - NN – keeps all the name space in memory (28+GB heap size)
    - 28G == 120M object == 40M Files and 80M blocks
    - Have combined some classes “uglier data structures” to avoid object overheads
    - Limit number of files based on memory size
  - Pig – try to spill to disk when low on memory
    - GC threshold too late, Usage threshold includes garbage objects
    - Considering managing our own buffers
- Concerns:
  - Efficiency of memory usage
  - Efficient GC and stability on large heaps
  - Direct buffer soft leaks lead to OutOfMemory exceptions

IO

- What we do
  - TransferTo to send data to socket
  - Own input and output streams over NIO sockets using our own selectors
- Concerns for IO
  - Extra copies for non-direct buffers in NIO
  - 2GB limit on TransferTo
  - write(ByteBuffer[]) always creates new direct buffers
  - Clean up of per-thread selectors in JDK is tied to GC
    - Fd leaks
  - A reader blocks a writer (and vice versa)
  - No write timeout
    - Both NIO and java.net.Socket

Efficiency of processing data

- What we do
  - Use direct buffers for compression
  - JNI for compression

- Concerns
  - Efficient use of C via JNI (compression, checksums)
  - 2GB limit on byte []
  - Efficient manipulation of byte data, UTF8 data
  - better implementations of zlib/gzip compression & other compressions
    - algorithms such as bzip which allow compressed data to be splitable

Summary

- A framework for storing & processing Petabyte of data using commodity hardware and storage
  - Cost effective, horizontally scalable
- In production use on over 20K nodes at Yahoo! and in smaller clusters elsewhere
- Apache open source project
- Implementing a file system & data processing system in Java is not such a crazy idea!
  - Indeed one of factors in Hadoop's success

More Info

- Main Web sites
  - http://hadoop.apache.org/core/
  - http://wiki.apache.org/hadoop/
  - http://wiki.apache.org/hadoop/GettingStartedWithHadoop
  - http://wiki.apache.org/hadoop/HadoopMapReduce

Sanjay Radia
sradia@yahoo-inc.com

Cloud Computing & Data Infrastructure
Yahoo!

Sun microsystems