CSE 487/587
Data Intensive Computing

Lecture 2: More Introduction

Vipin Chaudhary

vipin@buffalo.edu

716.645.4740
305 Davis Hall

Overview of Today's Lecture

- Administrative Matters Review
- More Introduction

CSE487/587: Data Intensive Computing

Instructor: Vipin Chaudhary
Office: Davis Hall 305, 645-4740
Email: vipin@buffalo.edu
Office Hours : Mon: 1:30-2:30pm

Teaching Assistant: Ruhan Sa, Yi Wei

Guest Lectures:

Text: None

Midterm: #1: March 11, 2015; #2: April 13, 2015

Final: May 11, 2015; 7:15-10:15pm, Knox 110

Reference: Several papers will be posted and discussed in class. Many online tutorials will be used. Some assignments will not be graded (you will be notified) but will help you with your midterms and Programming Assignments.

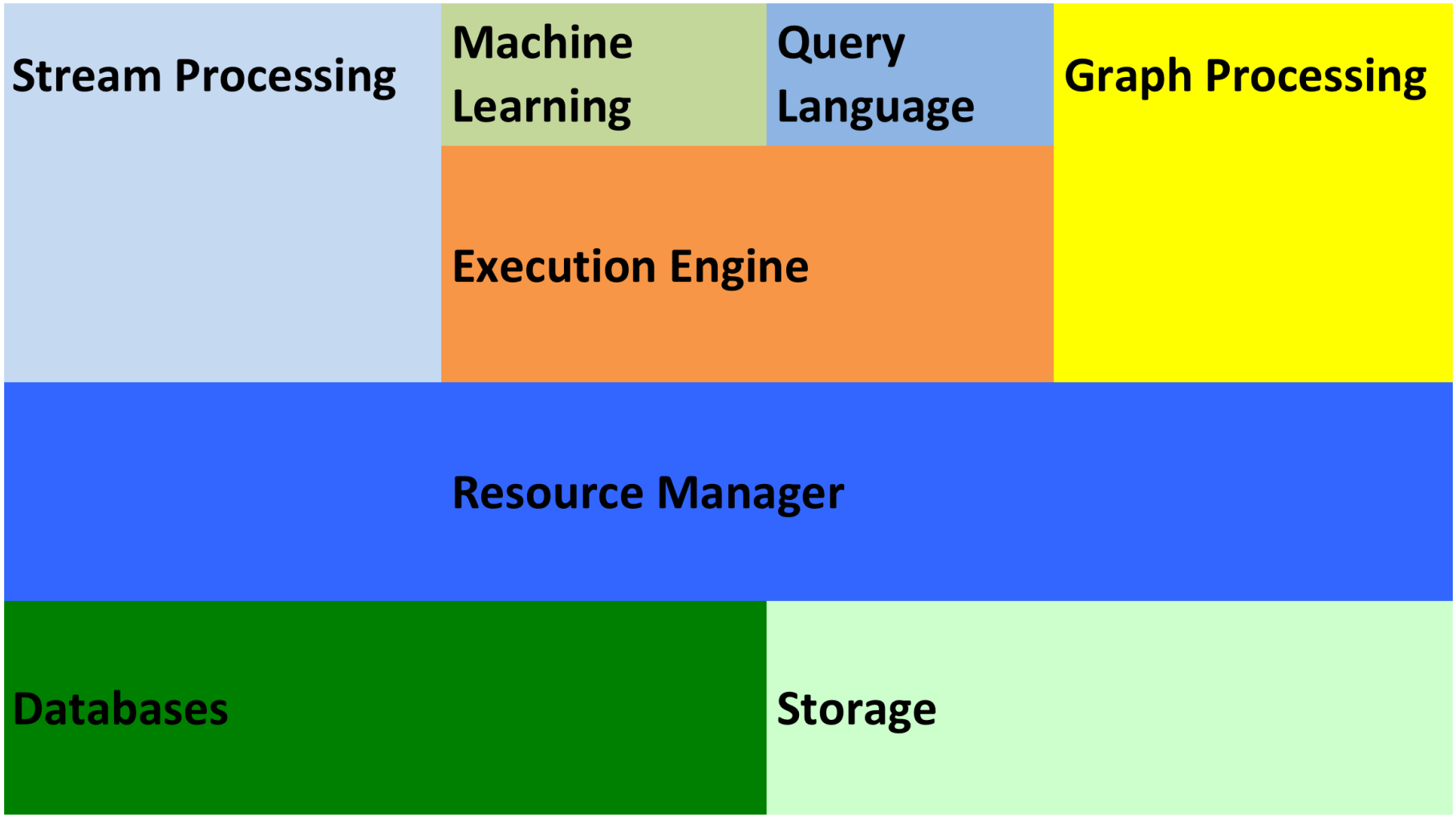
Programming Assignments Due Dates: Feb 23, March 11, March 30, April 15, May 4.

TA Officer Hours: Tu 2-4pm and Fr 3-4pm in Davis 302;
Fr 4-5pm (Bell 138).

Fun Problem of the Week #1

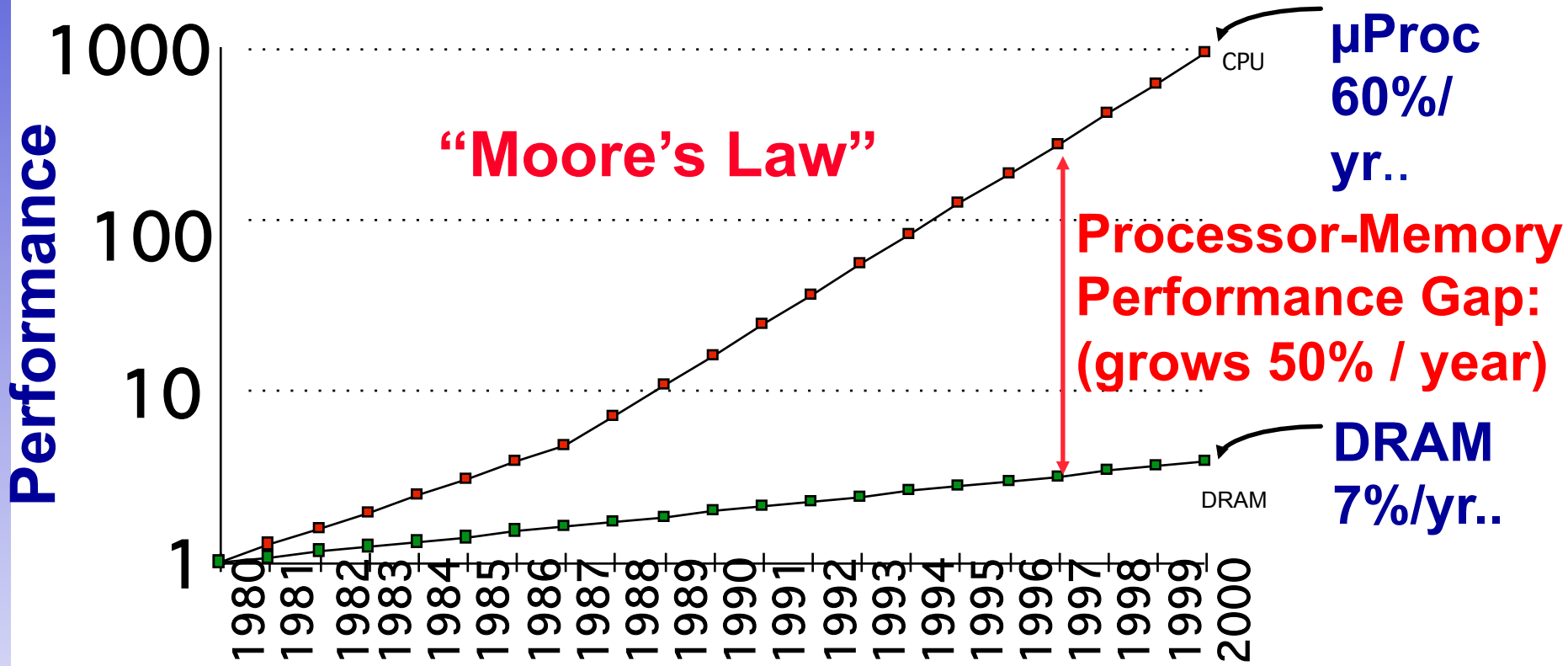
- Compute the total cost of building a data center that houses 10 ExaBytes of HDD
  - HDDs, networking, racks, building (real-estate), power, cooling, …
  - You can get the cost of various parts online
  - Assume the datacenter is built in Buffalo
  - State all your assumptions
  - Due 1/28 in class

How are Topics Related



Architecture Trends

Processor Limit: DRAM Gap

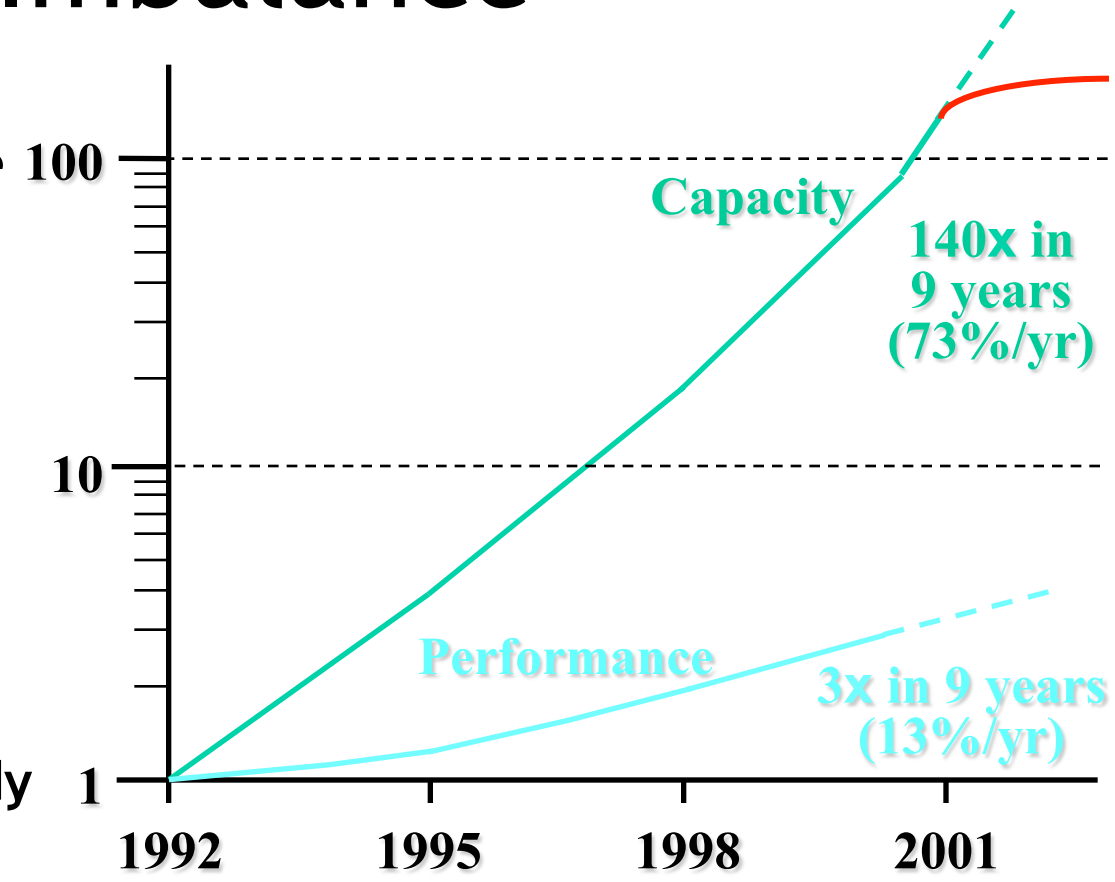


- Alpha 21264 full cache miss / instructions executed:
  180 ns/1.7 ns =108 clks x 4 or 432 instructions
- Caches in Pentium Pro: 64% area, 88% transistors

Source: Gordon Bell, DOE Science Computing Conference

Disk Capacity / Performance Imbalance

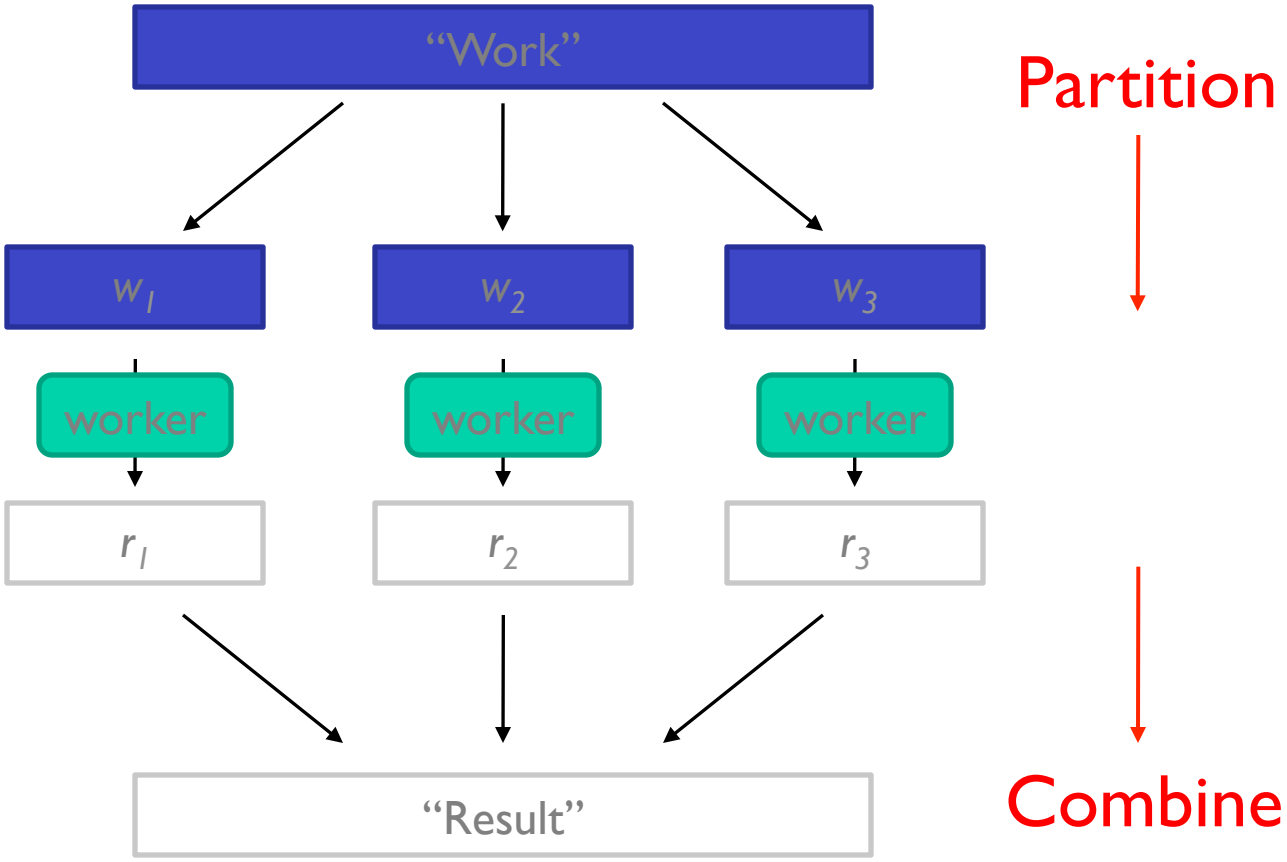
- Capacity growth outpacing performance growth
- Difference must be made up by better caching and load balancing
- Actual disk capacity may be capped by market (red line); shift to smaller disks (already happening for high speed disks)



Source: Gordon Bell, DOE Science Computing Conference

Source: Google

Divide and Conquer



Source: Google

What's the point?

- It's all about the right level of abstraction
  - Moving beyond the von Neumann architecture
  - We need better programming models
- Hide system-level details from the developers
  - No more race conditions, lock contention, etc.
- Separating the *what* from *how*
  - Developer specifies the computation that needs to be performed
  - Execution framework ("runtime") handles actual execution

“Big Ideas”

- Scale “out”, not “up”
  - Limits of SMP and large shared-memory machines
- Move processing to the data
  - Cluster have limited bandwidth
- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable
- Seamless scalability
  - From the mythical man-month to the tradable machine-hour

Source: Google

Typical Big Data Problem

Map

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results

Reduce

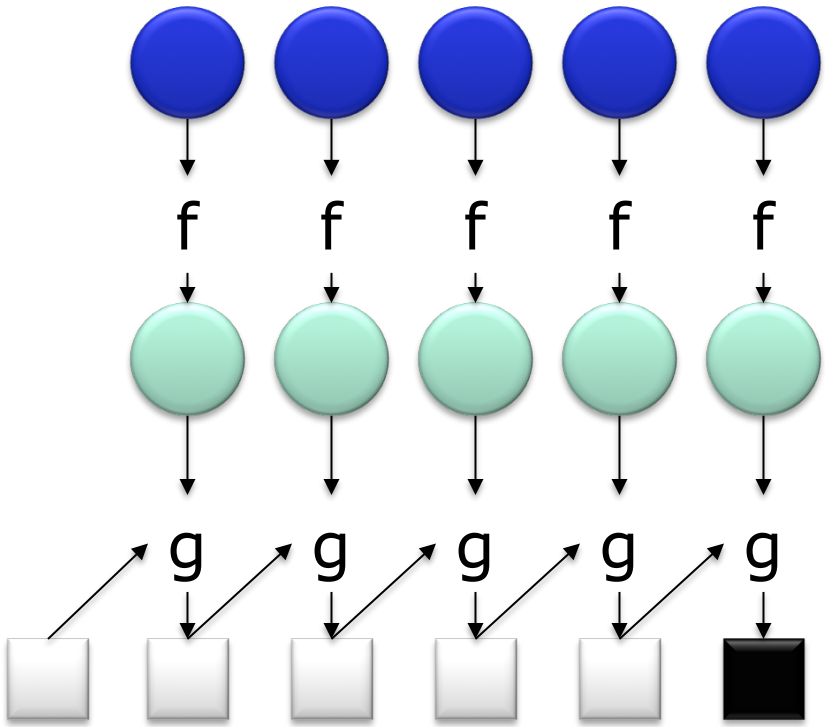
- Aggregate intermediate results
- Generate final output

Key idea: provide a functional abstraction for these two operations

Roots in Functional Programming

Map

Fold



MapReduce

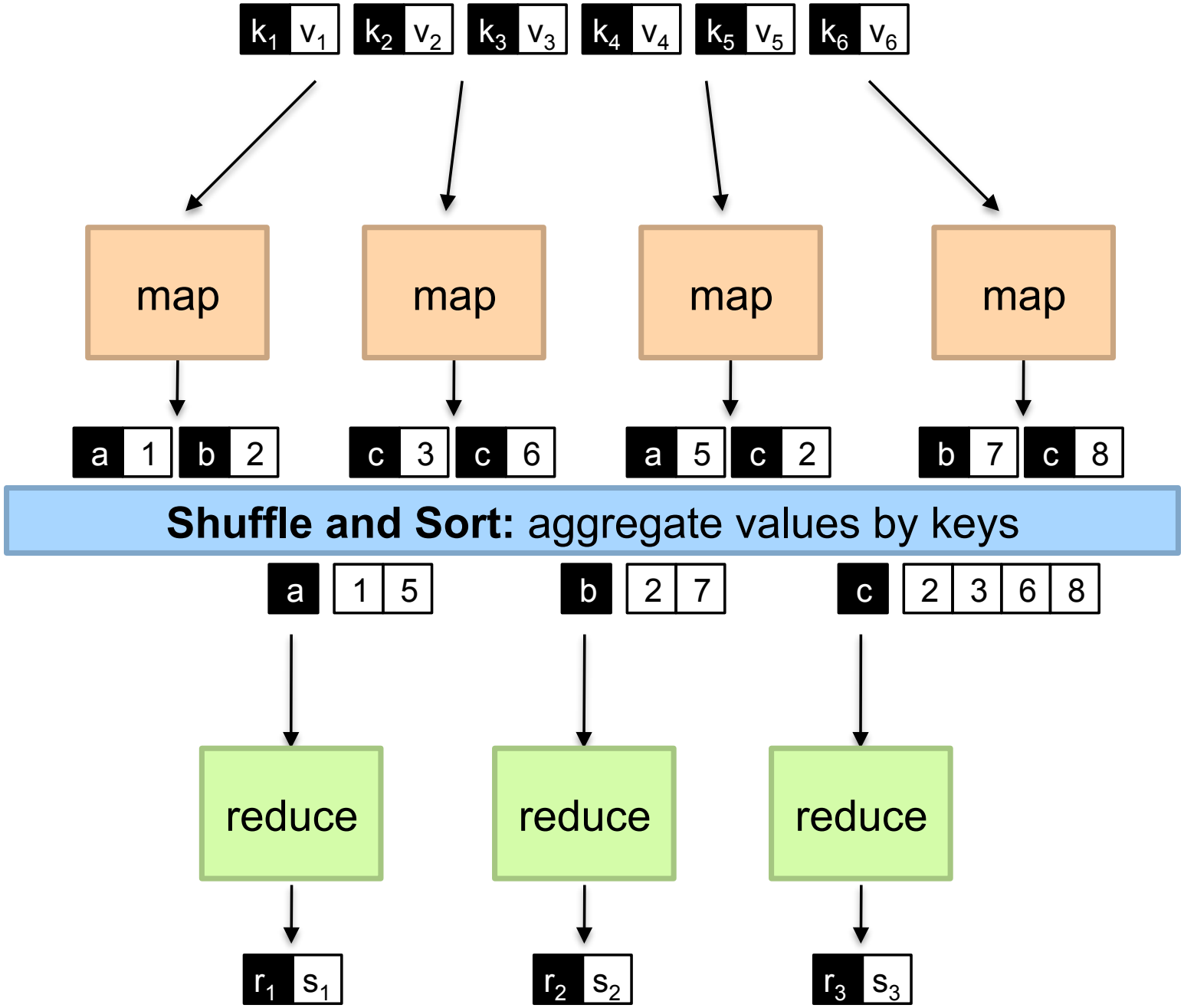
- Programmers specify two functions:

  map $(k_1, v_1) \rightarrow [<k_2, v_2>]$

  reduce $(k_2, [v_2]) \rightarrow [<k_3, v_3>]$

  - All values with the same key are sent to the same reducer

- The execution framework handles everything else…


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

MapReduce

- Programmers specify two functions:

  map (k, v) → <k', v'>*

  reduce (k', v') → <k', v'>*

  - All values with the same key are sent to the same reducer

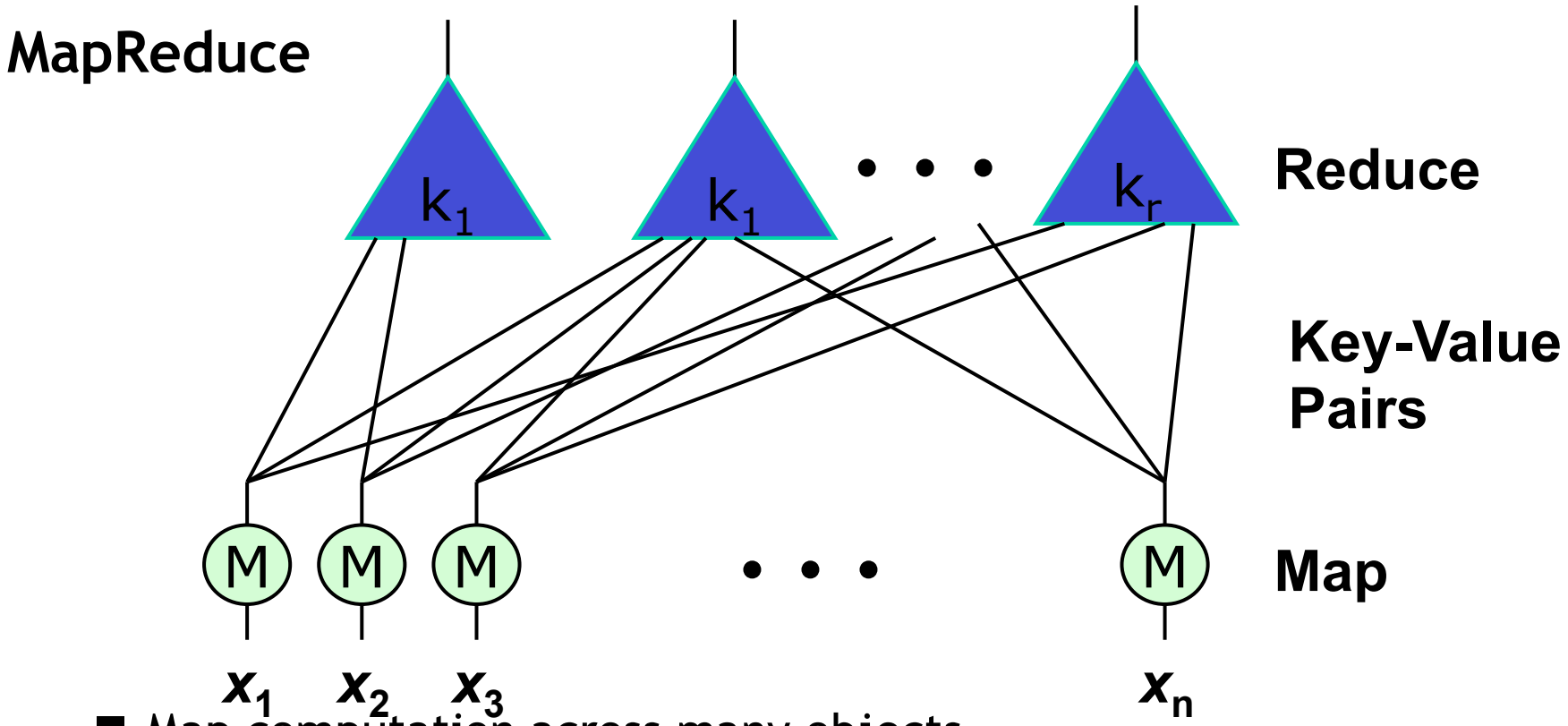
- The execution framework handles everything else…

What's "everything else"?

MapReduce “Runtime”

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS (later)

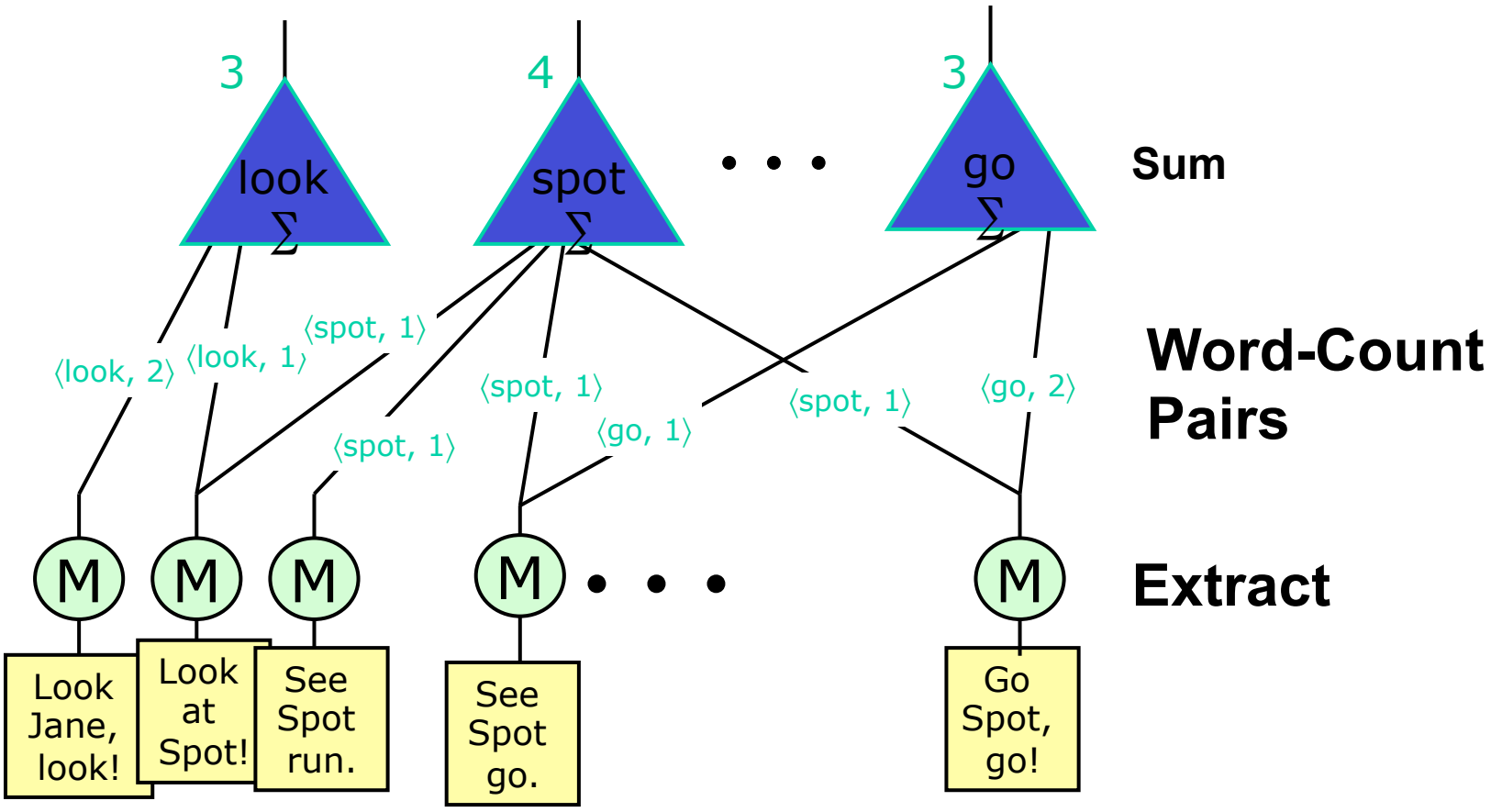
Google's Programming Model



- Map computation across many objects
  - E.g., 10^{10} Internet web pages
- Aggregate results in many different ways
- System deals with issues of resource allocation & reliability

Dean & Ghemawat: "MapReduce: Simplified Data Processing on Large Clusters", OSDI 2004

MapReduce Example



- Create an word index of set of documents
- Map: generate ⟨word, count⟩ pairs for all words in document
- Reduce: sum word counts across documents

The Power of Data + Computation

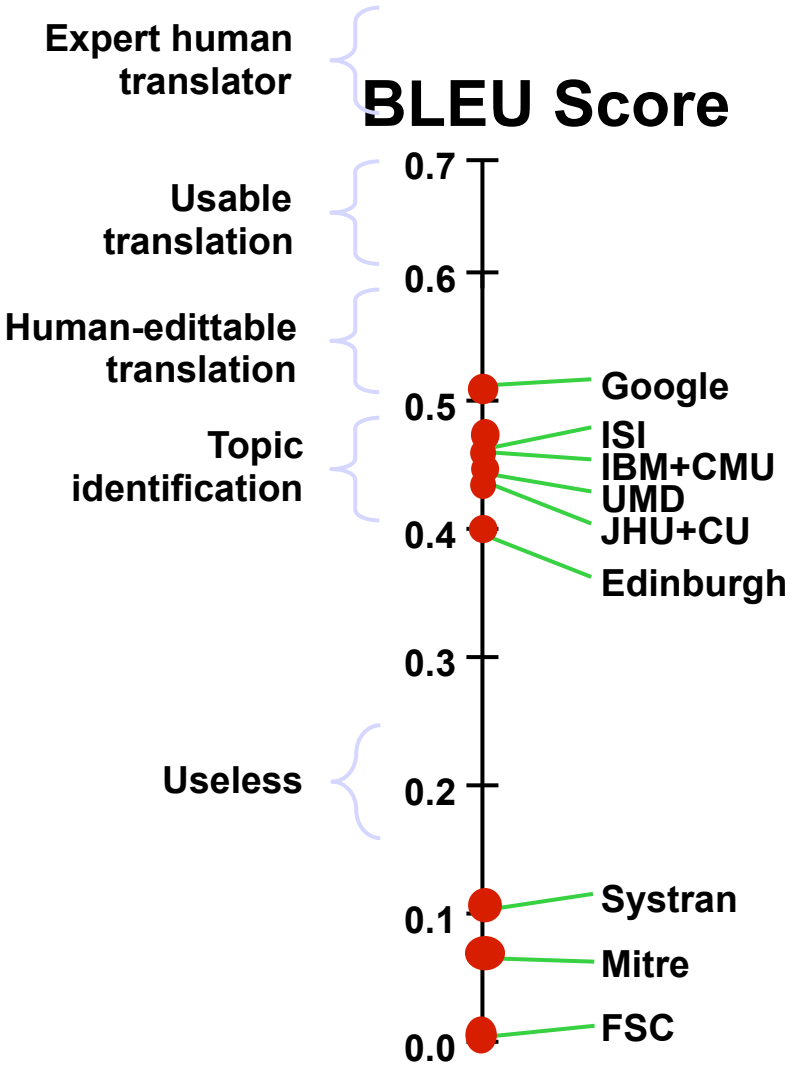
2005 NIST Machine Translation Competition

- Translate 100 news articles from Arabic to English

Google's Entry

- First-time entry
  - Highly qualified researchers
  - No one on research team knew Arabic
- Purely statistical approach
  - Create most likely translations of words and phrases
  - Combine into most likely sentences
- Trained using United Nations documents
  - 200 million words of high quality translated text
  - 1 trillion words of monolingual text in target language
- During competition, ran on 1000-processor cluster
  - One hour per sentence (gotten faster now)

2005 NIST Arabic-English Competition Results



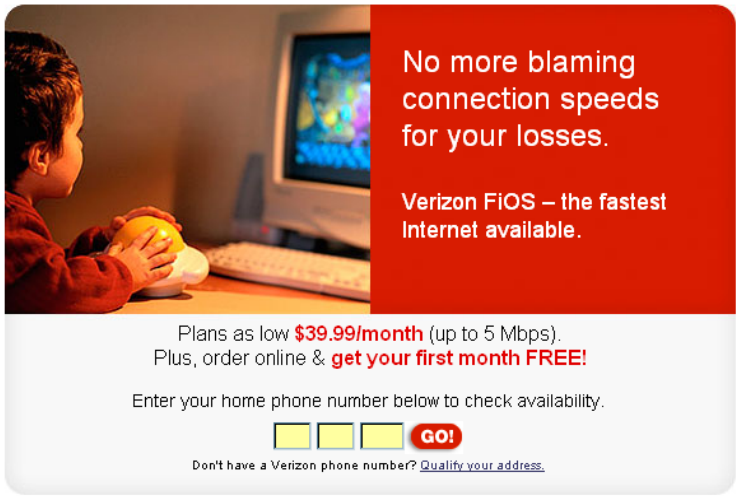
BLEU Score (Bilingual Evaluation Understudy)

- Statistical comparison to expert human translators
- Scale from 0.0 to 1.0

Outcome

- Google's entry qualitatively better
- Not the most sophisticated approach
- But lots more training data and computer power

Oceans of Data, Skinny Pipes



1 Terabyte

- Easy to store
- Hard to move

Disks	MB / s	Time
Seagate Barracuda	78	3.6 hours
Seagate Cheetah	125	2.2 hours
Networks	**MB / s**	**Time**
Home Internet	< 0.625	> 18.5 days
Gigabit Ethernet	< 125	> 2.2 hours
PSC Teragrid Connection	< 3,750	> 4.4 minutes

Data-Intensive System Challenge

For Computation That Accesses 1 TB in 5 minutes

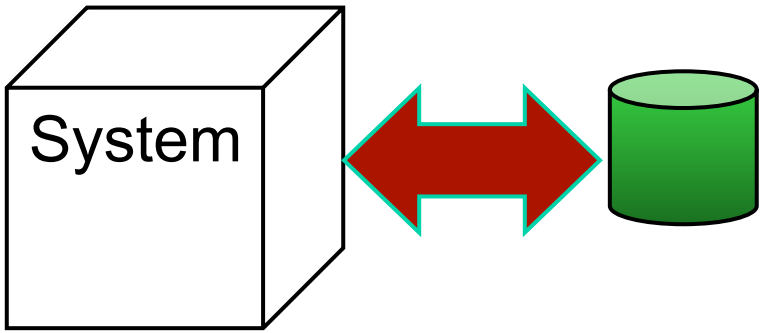
- Data distributed over 100+ disks
  - Assuming uniform data partitioning
- Compute using 100+ processors
- Connected by gigabit Ethernet (or equivalent)

System Requirements

- Lots of disks
- Lots of processors
- Located in close proximity
  - Within reach of fast, local-area network

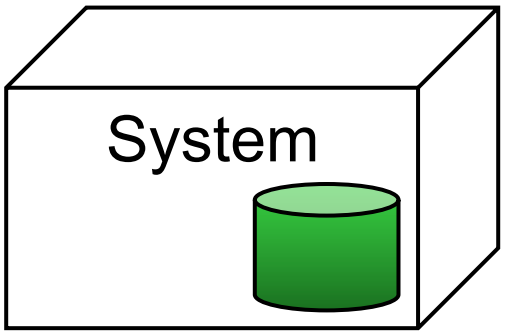
System Comparison: Data

Conventional Supercomputers



- Data stored in separate repository
  - No support for collection or management
- Brought into system for computation
  - Time consuming
  - Limits interactivity

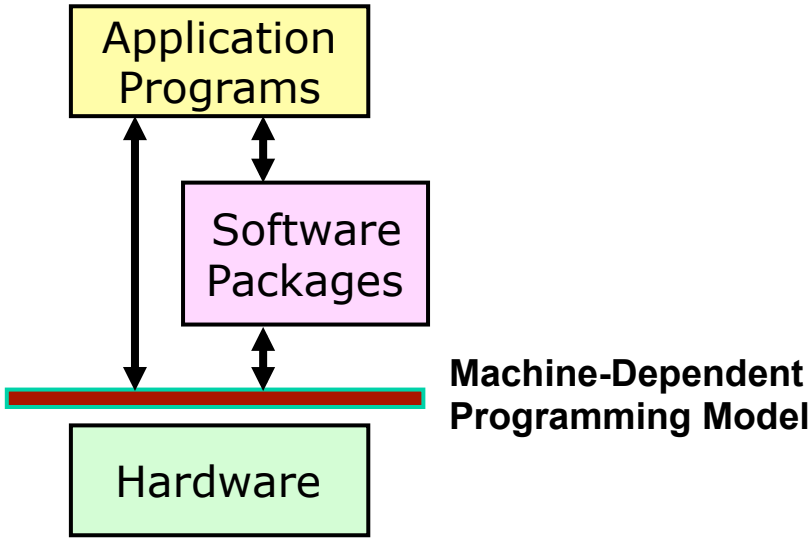
DISC



- System collects and maintains data
  - Shared, active data set
- Computation colocated with storage
  - Faster access

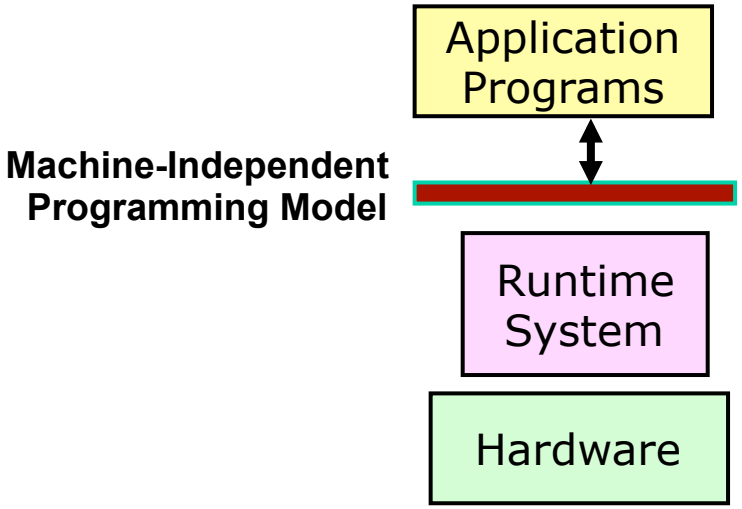
System Comparison: Programming Models

Conventional Supercomputers



- Programs described at very low level
  - Specify detailed control of processing & communications
- Rely on small number of software packages
  - Written by specialists
  - Limits classes of problems & solution methods

DISC



- Application programs written in terms of high-level operations on data
- Runtime system controls scheduling, load balancing, …

System Comparison: Interaction

Conventional Supercomputers

Main Machine: Batch Access

- Priority is to conserve machine resources
- User submits job with specific resource requirements
- Run in batch mode when resources available

Offline Visualization

- Move results to separate facility for interactive use

DISC

Interactive Access

- Priority is to conserve human resources
- User action can range from simple query to complex computation
- System supports many simultaneous users
  - Requires flexible programming and runtime environment

System Comparison: Reliability

Runtime errors commonplace in large-scale systems

- Hardware failures
- Transient errors
- Software bugs

Conventional Supercomputers

"Brittle" Systems

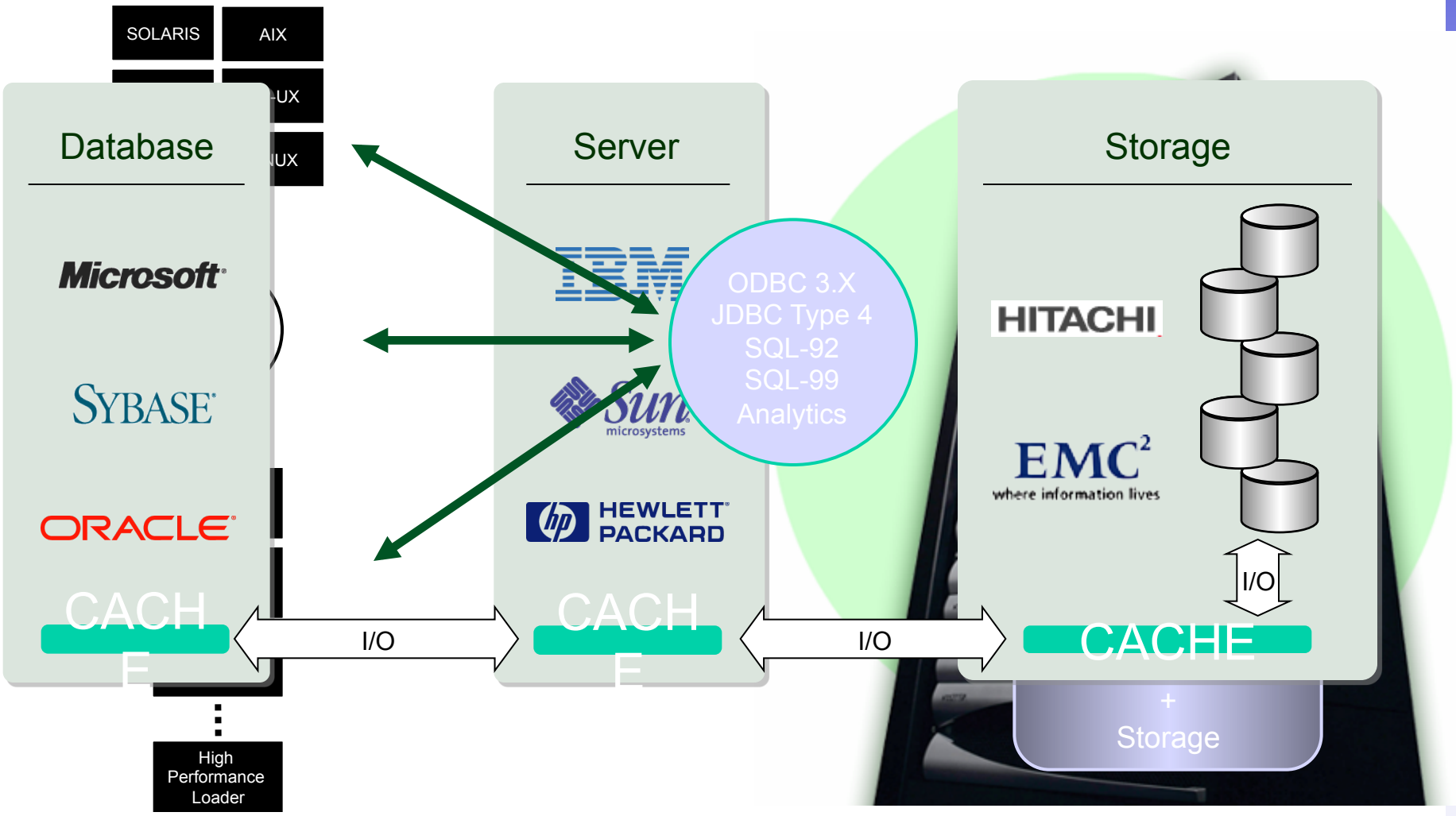
- Main recovery mechanism is to recompute from most recent checkpoint
- Must bring down system for diagnosis, repair, or upgrades
- Unless you add fault-tolerance mechanisms

DISC

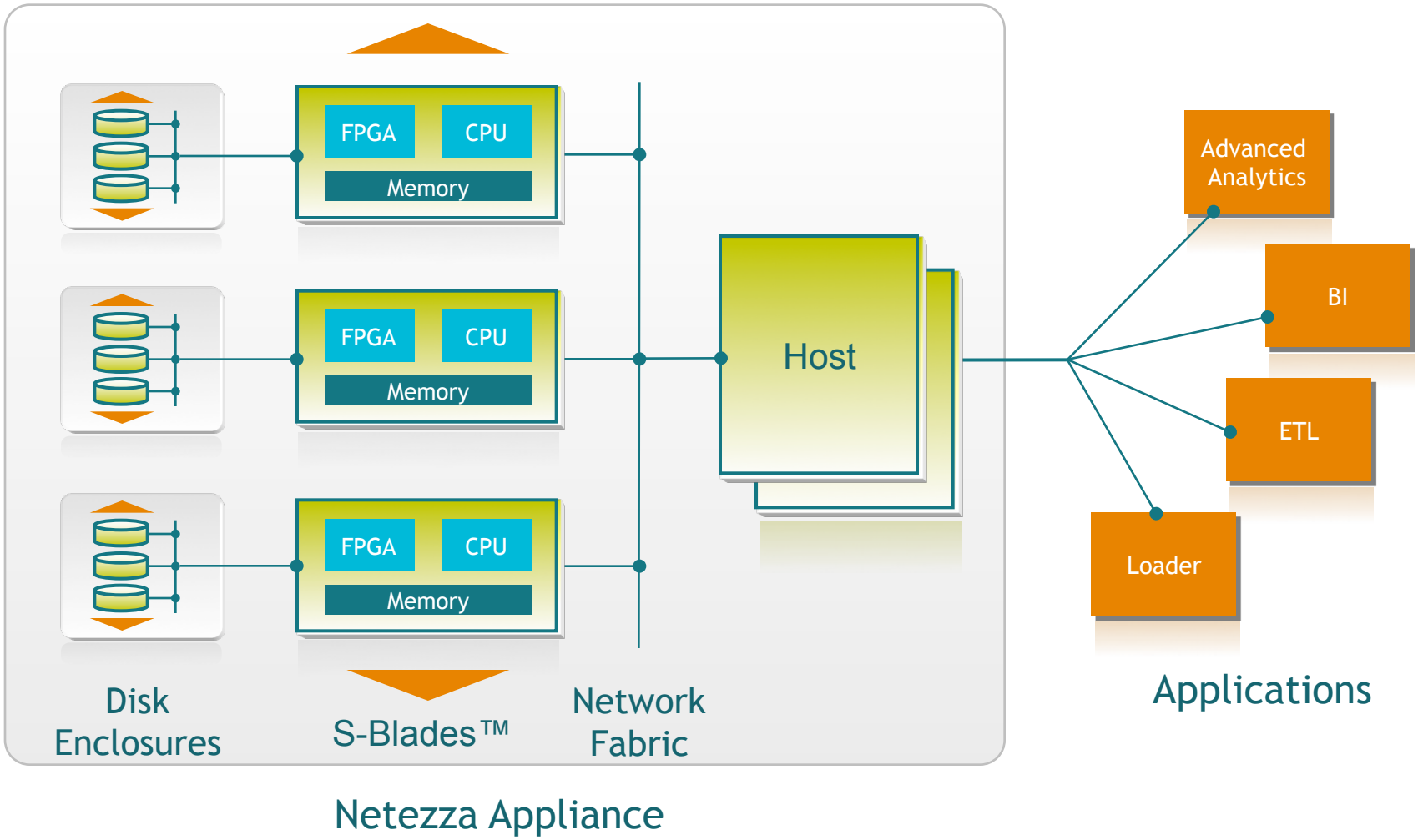
Flexible Error Detection and Recovery

- Runtime system detects and diagnoses errors
- Selective use of redundancy and dynamic recomputation
- Replace or upgrade components while system running
- Requires flexible programming model & runtime environment

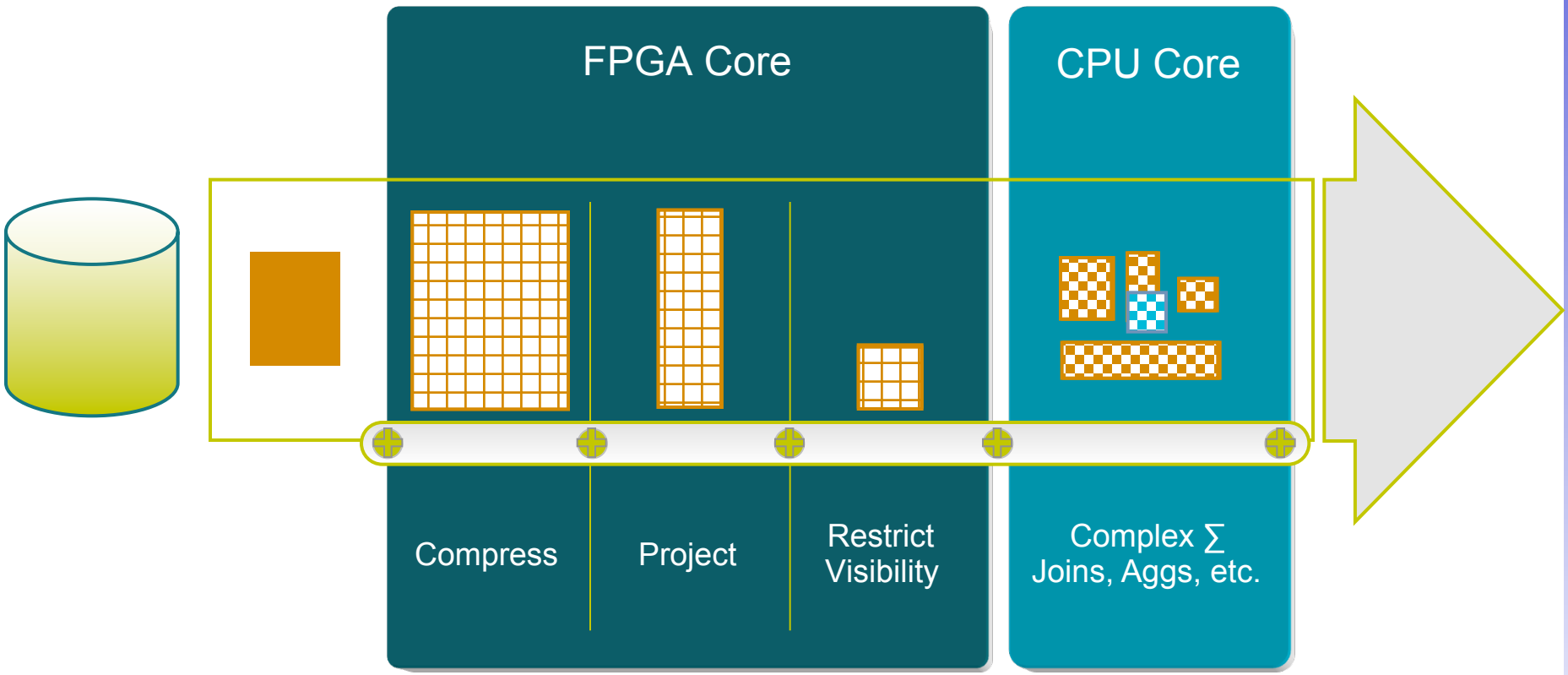
Active Disk Approach



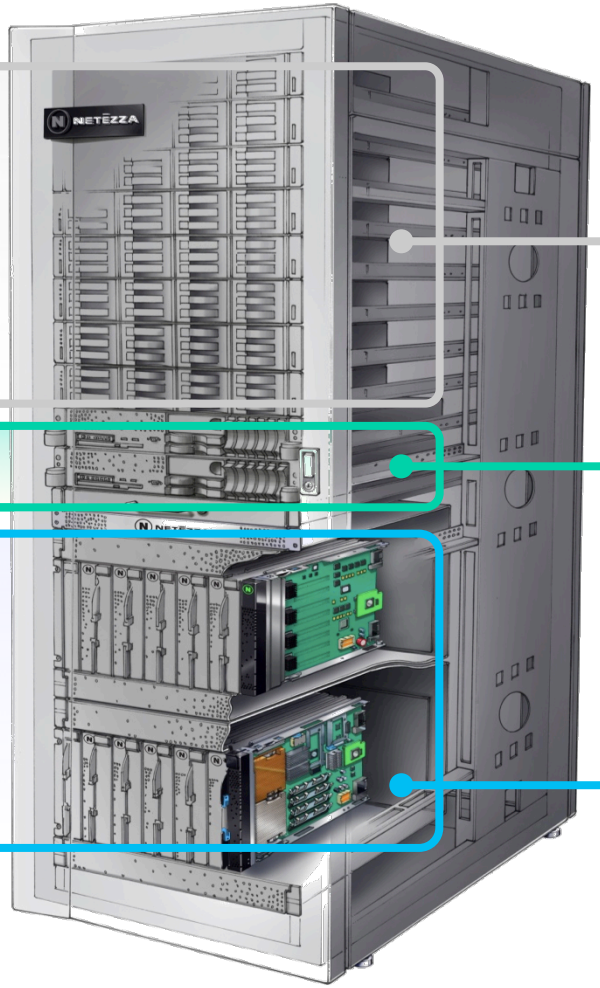
Netezza AMPP™: Asymmetric Massively Parallel Processing



Data Stream Processing



The Netezza TwinFin™ Appliance



Disk Enclosures

SMP Hosts

Snippet Blades™ (S-Blades™)

Slice of User Data
Swap and Mirror partitions
High speed data streaming

SQL Compiler
Query Plan
Optimize
Admin

Processor & streaming DB logic
High-performance database engine streaming joins, aggregations, sorts, etc.

CS Research Issues

Applications

- Language translation, image processing, …

Application Support

- Machine learning over very large data sets
- Web crawling

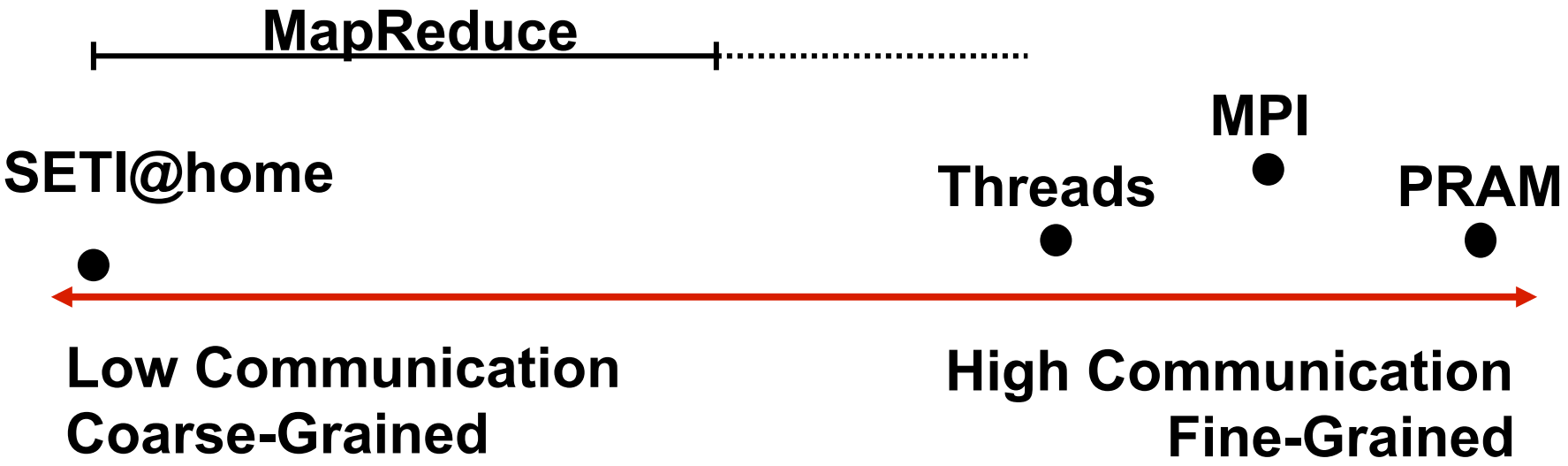
Programming

- Abstract programming models to support large-scale computation
- Distributed databases

System Design

- Error detection & recovery mechanisms
- Resource scheduling and load balancing
- Distribution and sharing of data across system

Exploring Parallel Computation Models



DISC + MapReduce Provides Coarse-Grained Parallelism

- Computation done by independent processes
- File-based communication

Observations

- Relatively "natural" programming model
  - If someone else worries about data distribution & load balancing
- Research issue to explore full potential and limits

Example: Sparse Matrices with MapReduce

MapReduce

- Programmers specify two functions:

  map (k, v) → <k', v'>*

  reduce (k', v') → <k', v'>*

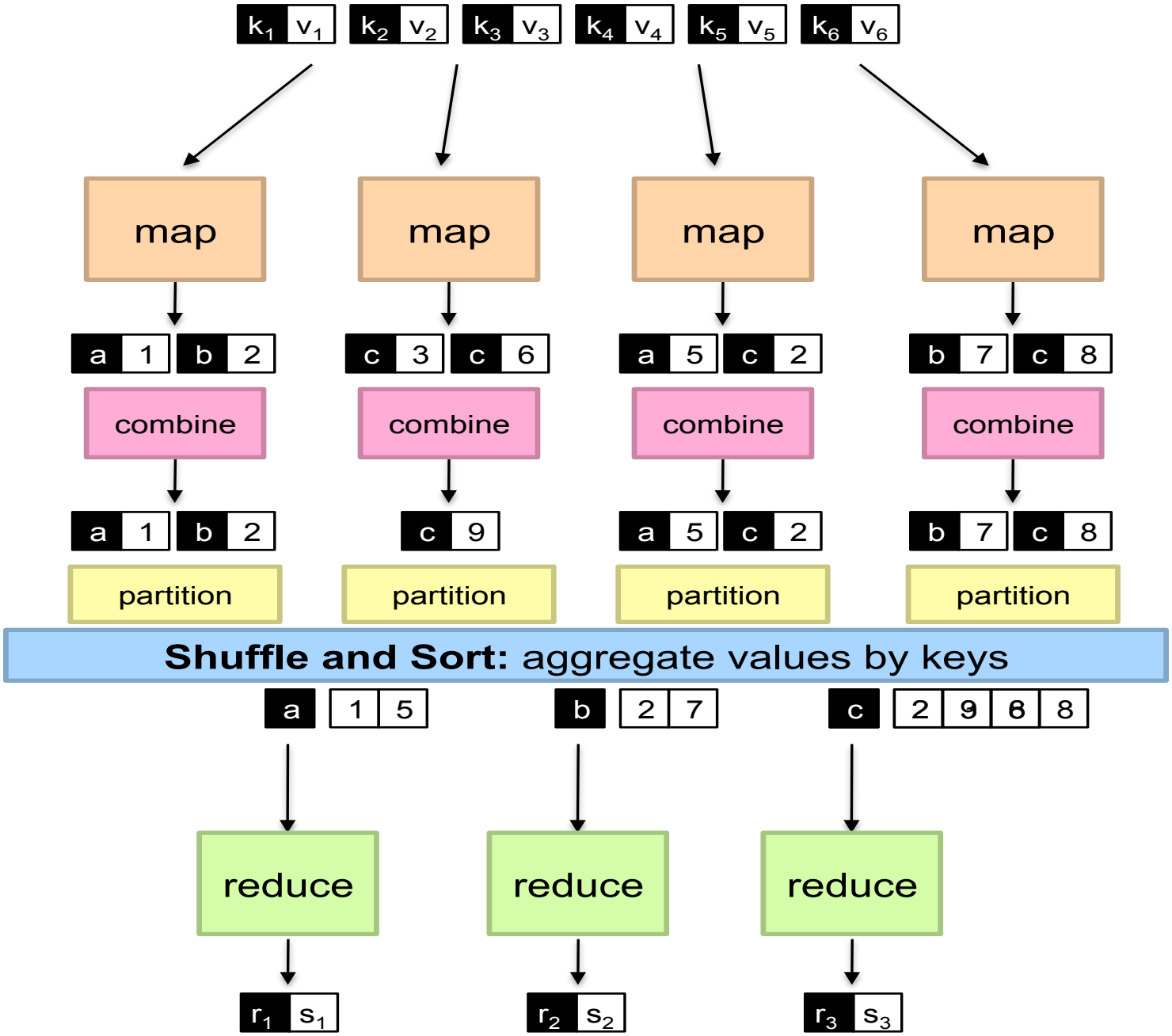
  - All values with the same key are reduced together
- The execution framework handles everything else…
- Not quite…usually, programmers also specify:

  partition (k', number of partitions) → partition for k'

  - Often a simple hash of the key, e.g., hash(k') mod n
  - Divides up key space for parallel reduce operations

  combine (k', v') → <k', v'>*

  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Two more details…

- Barrier between map and reduce phases
  - But we can begin copying intermediate data earlier
- Keys arrive at each reducer in sorted order
  - No enforced ordering *across* reducers

“Hello World”: Word Count

```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);

Reduce(String term, Iterator<Int> values):
    int sum = 0;
    for each v in values:
        sum += v;
        Emit(term, value);
```

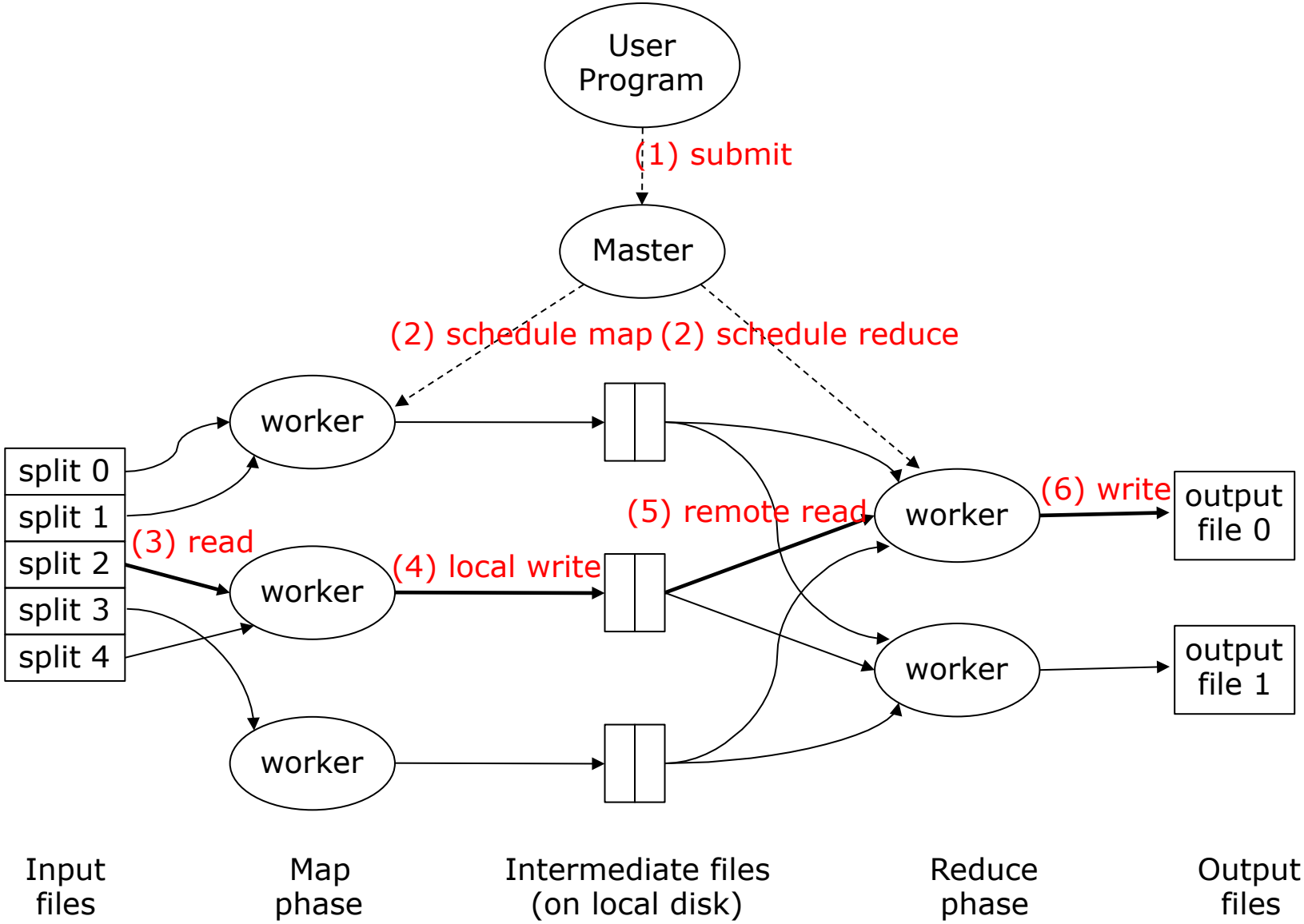
MapReduce can refer to…

- The programming model
- The execution framework (aka “runtime”)
- The specific implementation

Usage is usually clear from context!

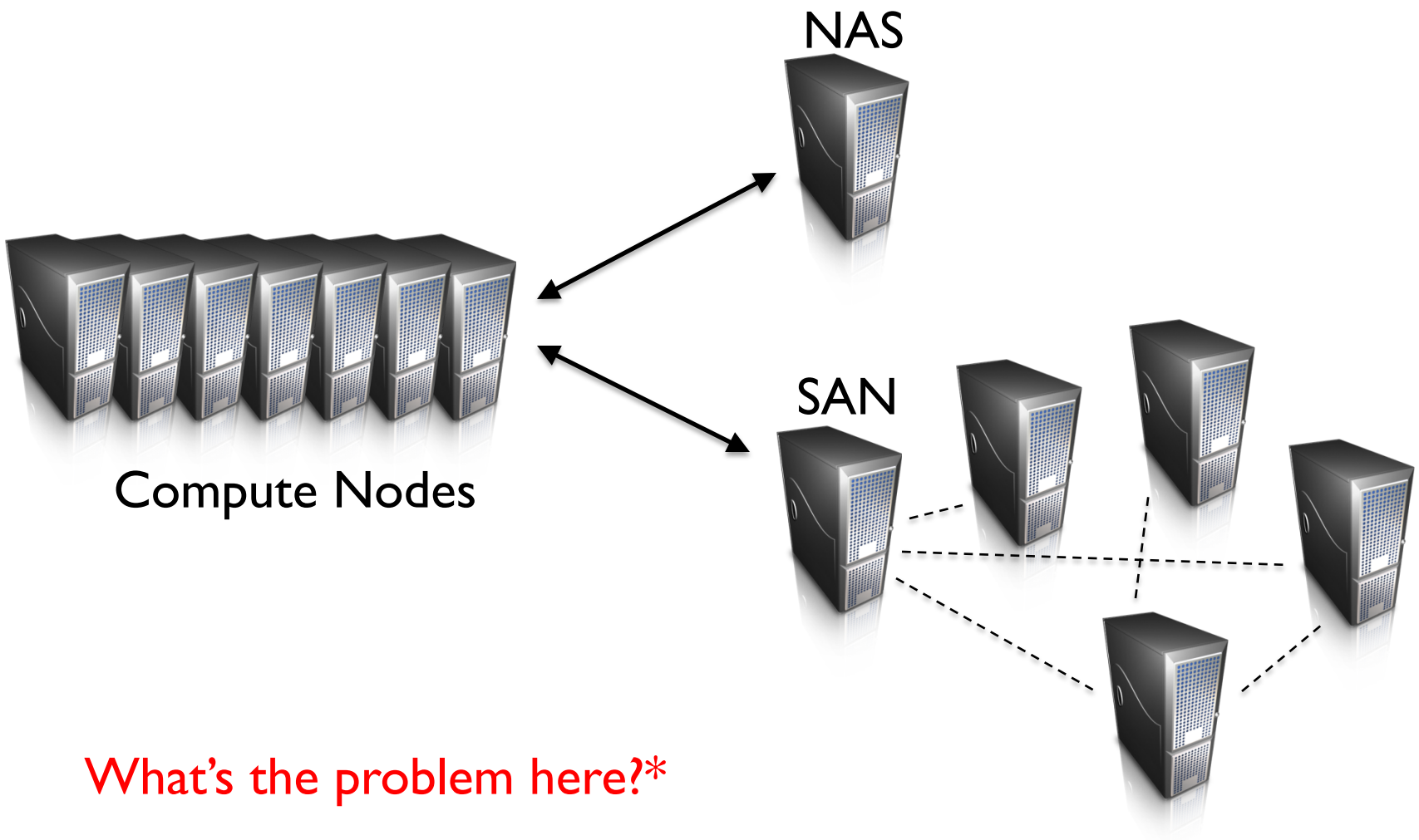
MapReduce Implementations

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, now an Apache project
  - Used in production at Yahoo, Facebook, Twitter, LinkedIn, Netflix, …
  - The *de facto* big data processing platform
  - Rapidly expanding software ecosystem
- Lots of custom research implementations
  - For GPUs, cell processors, etc.



Adapted from (Dean and Ghemawat, OSDI

How do we get data to the workers?



What's the problem here?*

* Really a problem?

Distributed File System

- Don’t move data to workers… move workers to the data!
  - Store data on the local disks of nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, but disk throughput is reasonable
- A distributed file system is the answer
  - GFS (Google File System) for Google’s MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

GFS: Assumptions

- Commodity hardware over “exotic” hardware
  - Scale “out”, not “up”
- High component failure rates
  - Inexpensive commodity components fail all the time
- “Modest” number of huge files
  - Multi-gigabyte files are common, if not encouraged
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads over random access
  - High sustained throughput over low latency

GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large datasets, streaming reads
- Simplify the API
  - Push some of the issues onto the client (e.g., data layout)

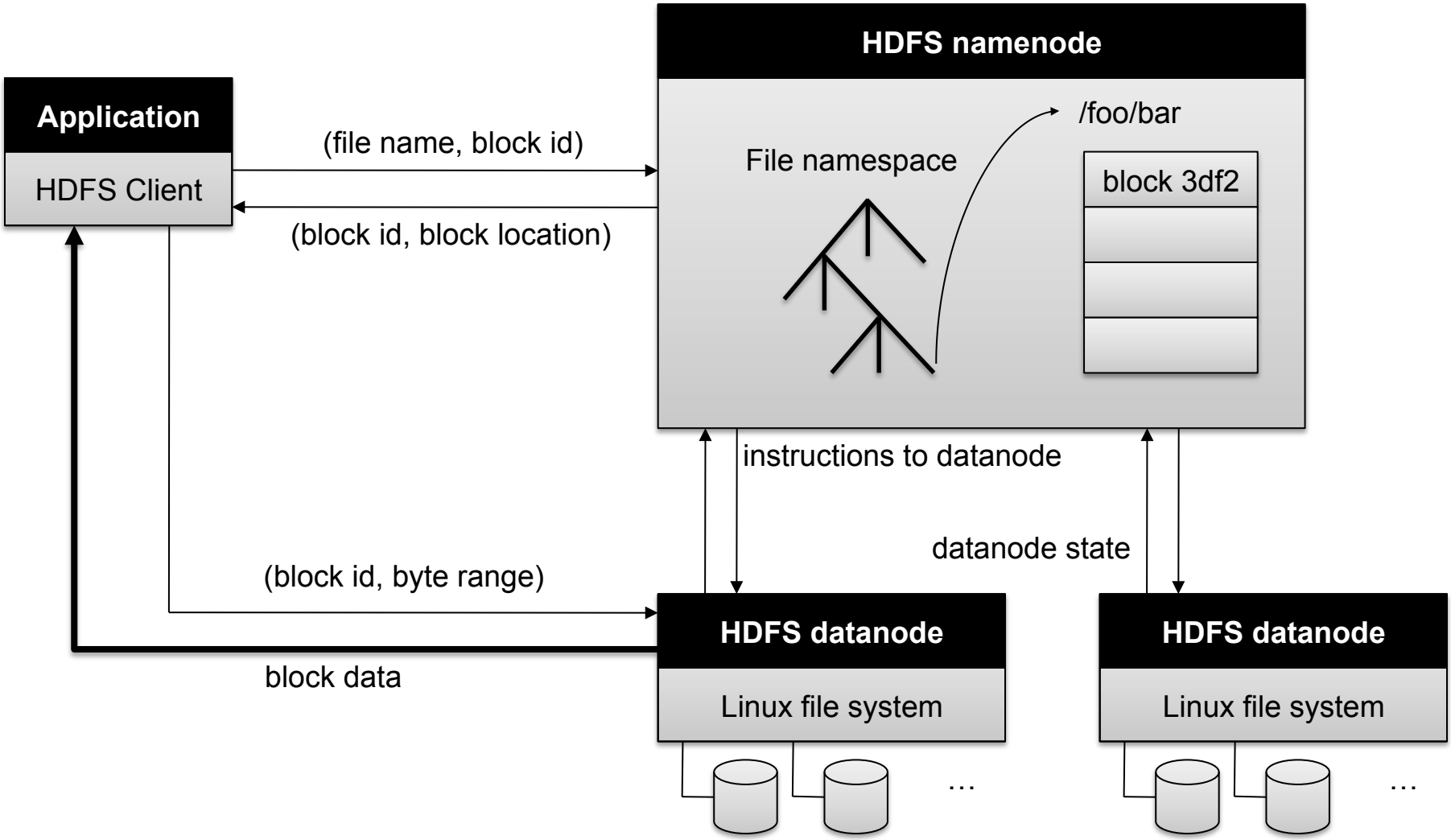
HDFS = GFS clone (same basic ideas)

From GFS to HDFS

- Terminology differences:
  - GFS master = Hadoop namenode
  - GFS chunkservers = Hadoop datanodes

- Differences:
  - Different consistency model for file appends
  - Implementation
  - Performance

HDFS Architecture

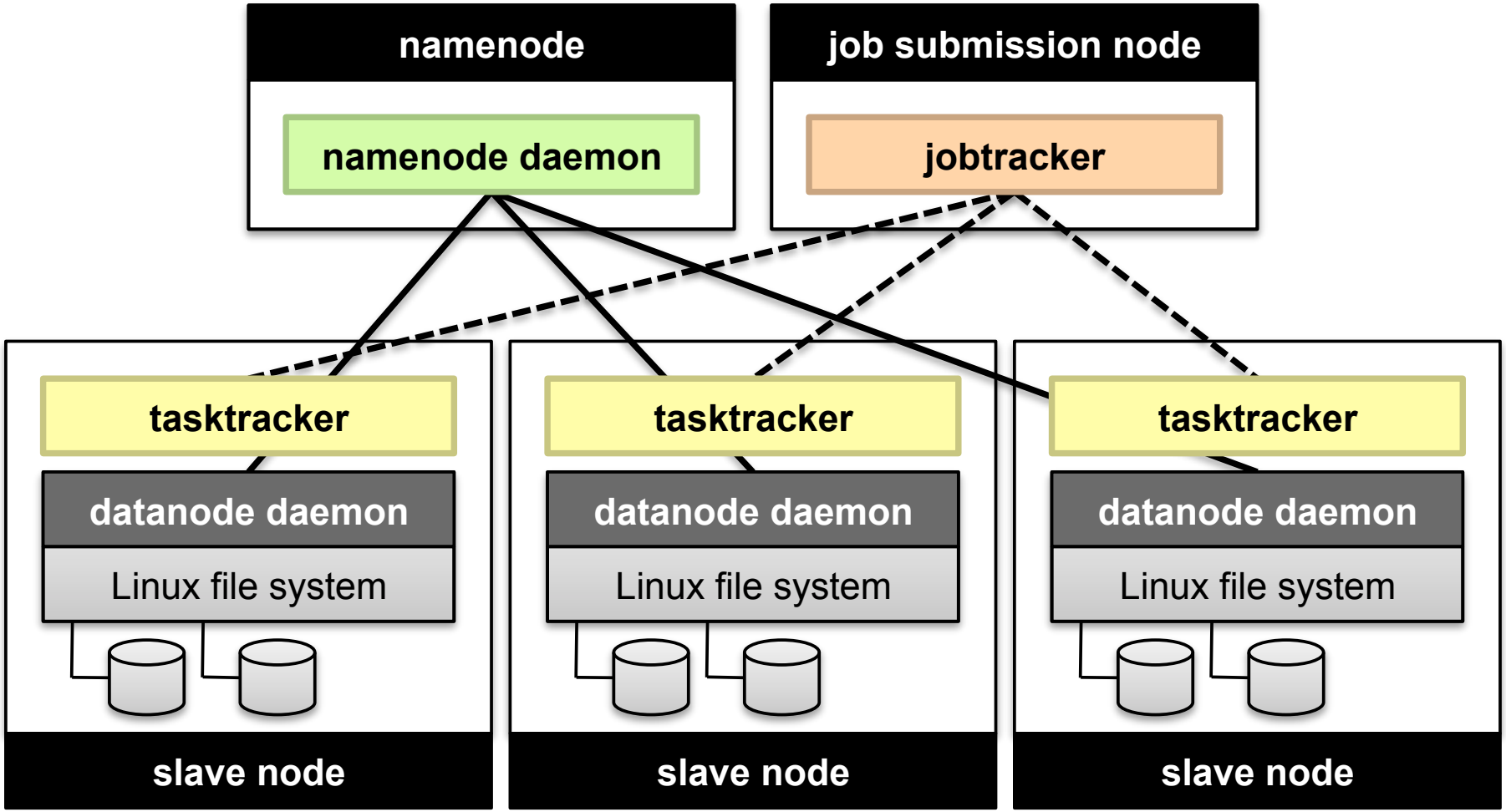


Adapted from (Ghemawat et al., SOSP 2003)

Namenode Responsibilities

- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
- Coordinating file operations:
  - Directs clients to datanodes for reads and writes
  - No data is moved through the namenode
- Maintaining overall health:
  - Periodic communication with the datanodes
  - Block re-replication and rebalancing
  - Garbage collection

Putting everything together…



(Not Quite… We'll come back to YARN later)

Thank you!