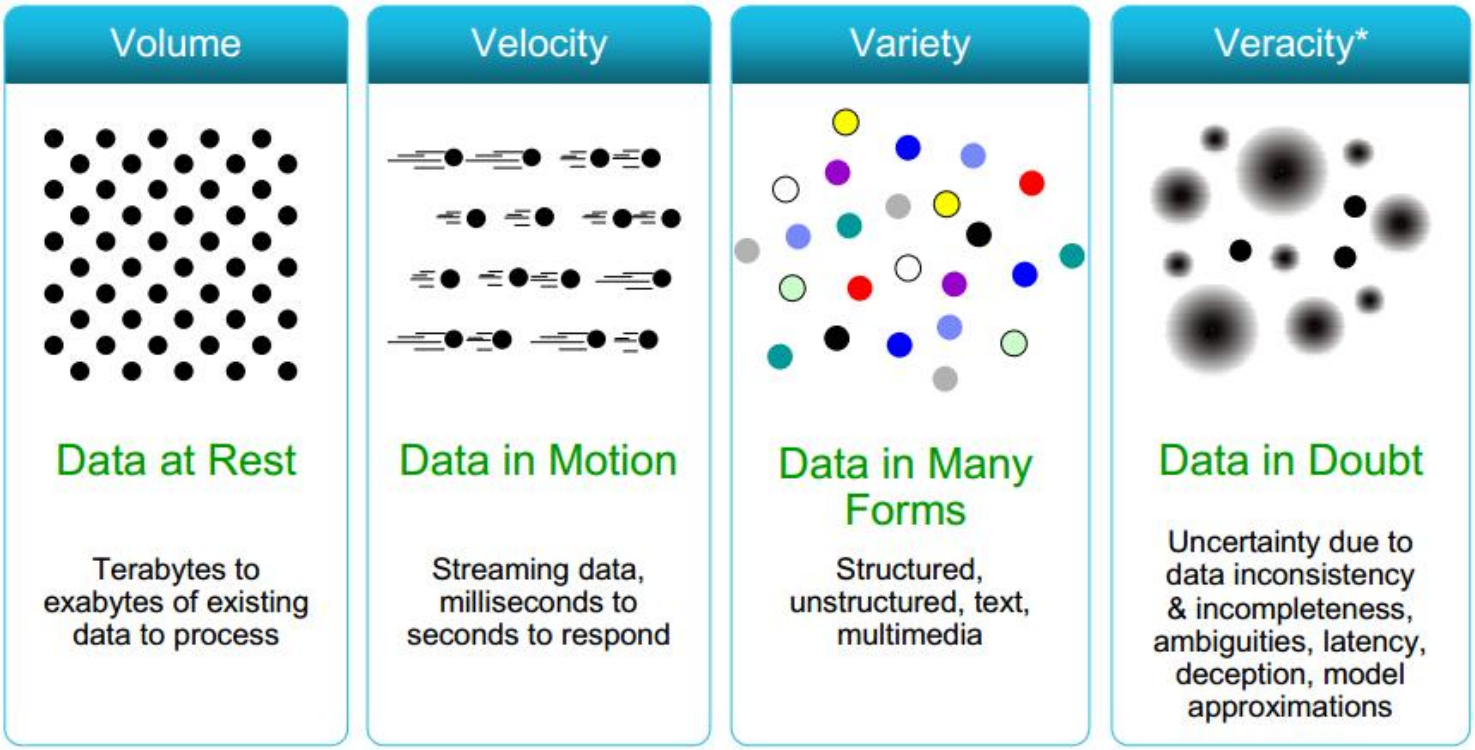
Big Data

Chương trình

Chương	*Nội dung*	*Phương pháp*	*Tài liệu tham khảo*
1	Introduction to Big Data	Giảng	
2	Parallel & Distributed computing	Giảng	
3	Map-Reduce (Hadoop)	Seminar	
4	GPUs & Xeon Phi	Seminar	
5	Storage: GFS, HDFS, Lustre	Seminar	
6	Streaming & data stream	Seminar	
7	Big data analytics: clustering and classification	Seminar	
8	Big data analytics: machine learning	Seminar	
9	Spark and Data analytics	Seminar	
10	Graph computing	Seminar	
11	Big data visualization	Seminar	
12	Big data in Time Series	Seminar	

Ch1 - Introduction to Big Data

- What is Big Data?
- 4V's



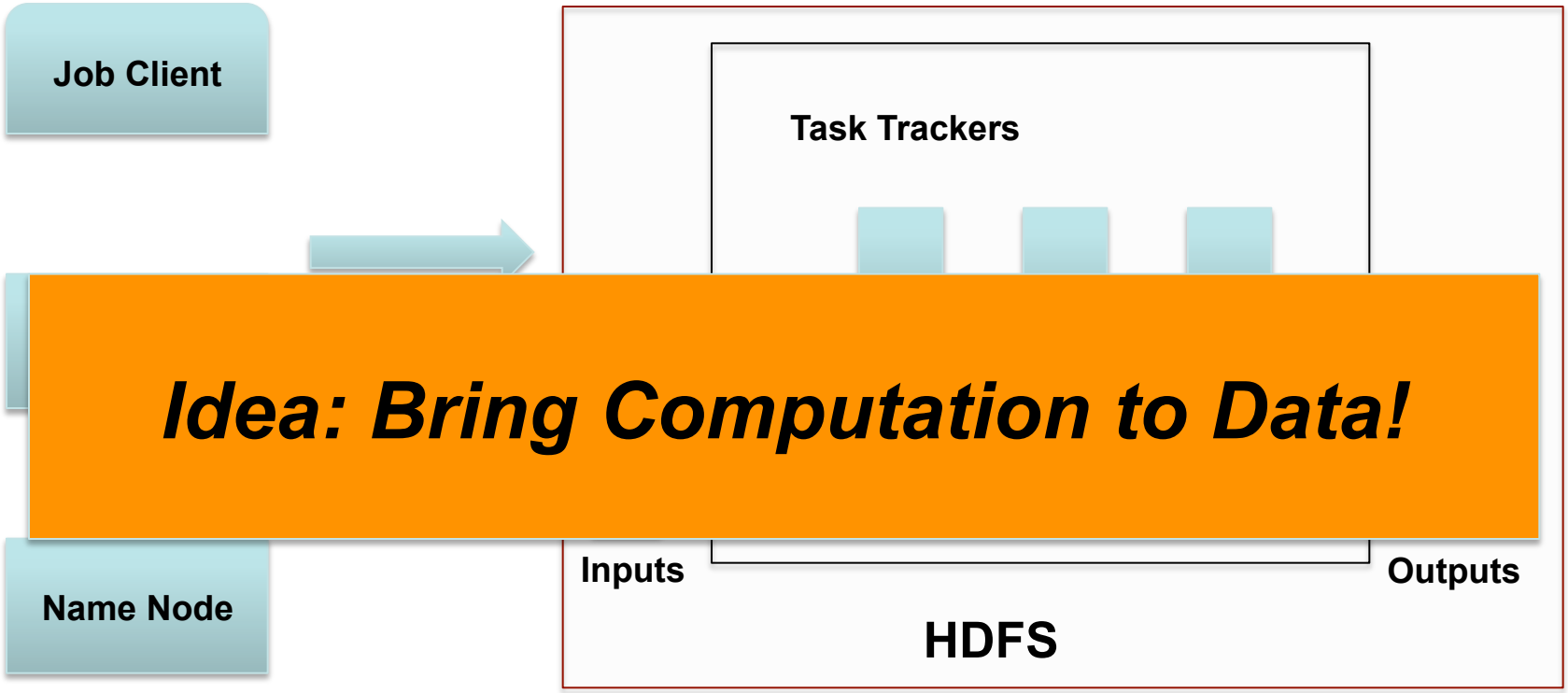
Ch2: Parallel & Distributed computing

- Pipeline, data parallelism, control parallelism
- Speedup, efficiency
- HPC & HTC
- Multiprocessor (UMA & NUMA), Multicomputer
- Parallel techniques
  - Embarrassingly Parallel Computations
  - Partitioning and Divide-and-Conquer Strategies
  - Pipelined Computations
  - Synchronous Computations

Ch3: Map-Reduce

- What is Map-Reduce
- MapReduce Architecture
- Algorithms
  - <key, value>

MapReduce Architecture: Master-Slaves



Job Client: Submit Jobs

Task Tracker: Execute Jobs

Job Tracker: Coordinate Jobs (Scheduling, Phase Coordination, etc.)

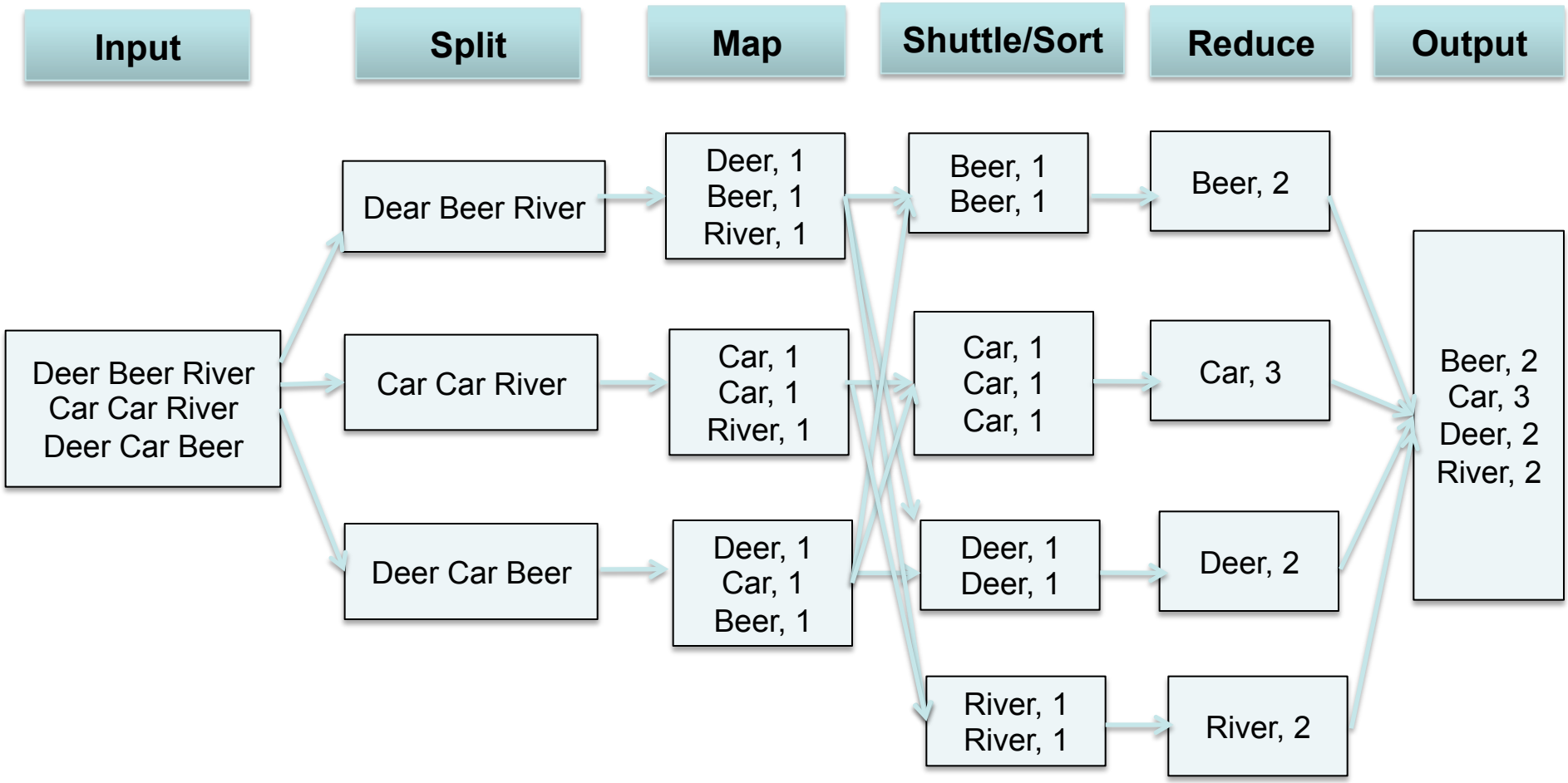
Job: MapReduce Function+ Config

Map-Reduce: example 1

Coins Deposit

- **Mapper:** Categorize coins by their face values
- **Reducer:** Count the coins in each face value ***in parallel***

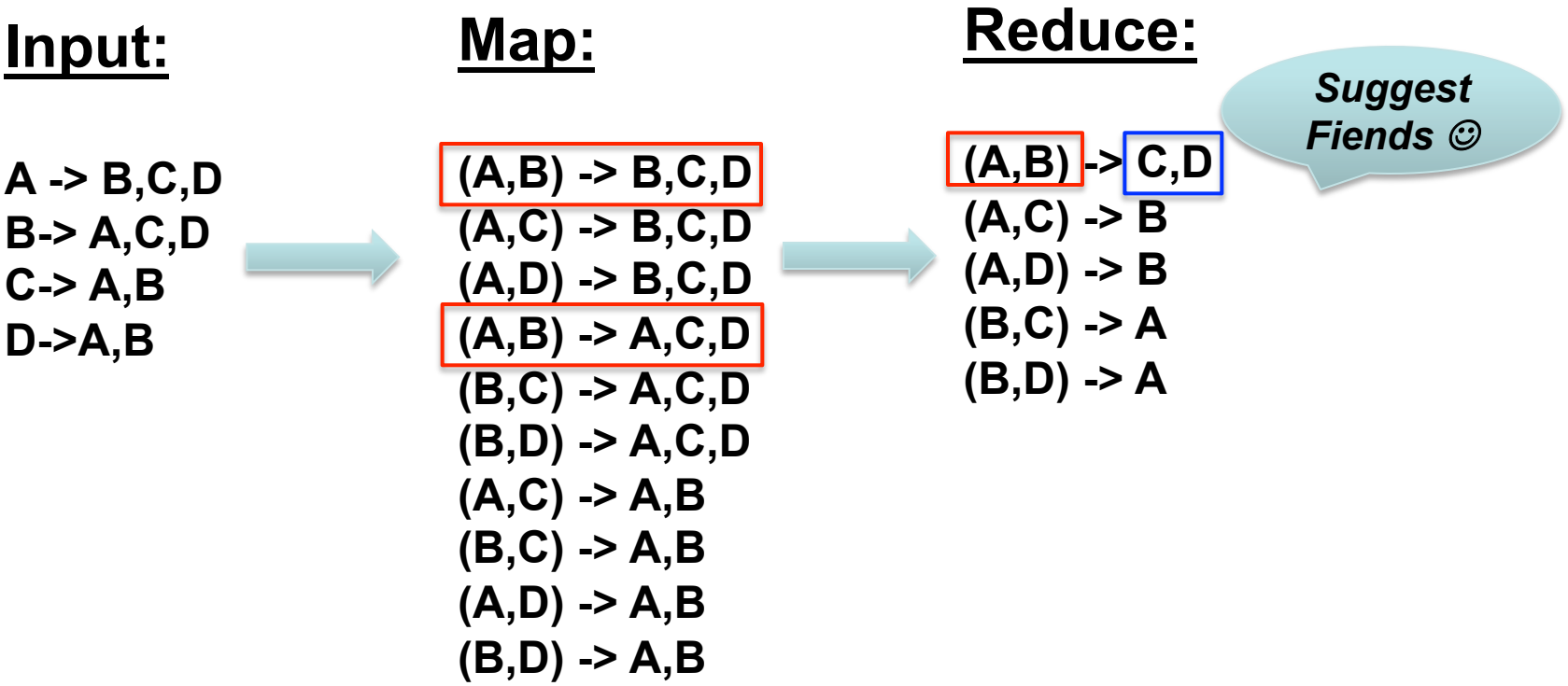
Map-Reduce: example 2



Similar Flavor of Coins Deposit ? ☺

Map Reduce Problems Discussion

- **Problem 3:** Find Common Friends
- **Mapper and Reducer:**



Ch4 - GPUs & Xeon Phi

- Coprocessor, accelerator
- Many core (multicore)
- SIMD
- Programming models

GPUs

1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

Intel Xeon Phi Execution Models: Offload Mode

- The host system (Xeon E5 processor on Stampede) offloads part or all of the computation to the coprocessor
- The application starts execution on the host
- As the computation proceeds, the host can decide to send data to the coprocessor and let that work on it
- The host and the coprocessor may or may not work in parallel
- Note: Offload code will not run on the 61st core of Phi as the offload daemon runs here

Explicit Offload: Pragma-based approach

"Hello World" in the explicit offload model:

```
1  #include <stdio.h>
2  int main(int argc, char * argv[] ) {
3     printf("Hello World from host!\n");
4  #pragma offload target(mic)
5        {
6           printf("Hello World from coprocessor!\n"); fflush(0);
7        }
8     printf("Bye\n");
9  }
```

Application runs on the host, but some parts of code and date are moved ("offloaded") the coprocessor.

Compiling and Running an Offload Application

```
user@host% icpc hello_offload.cpp -o hello_offload
user@host% ./hello_offload
Hello World from host!
Bye
Hello World from coprocessor!
```

- No additional arguments if compiled with an Intel compiler
- Run application on host as a regular application
- Code inside of **#pragma offload** is offloaded automatically

  Console output on Intel Xeon Phi coprocessor is buffered and mirrored to the host console

  If coprocessor is not installed, code inside **#pragma offload** runs on the host system

Intel Xeon Phi Execution Models: Coprocessor Native Execution Mode

- The coprocessor can be viewed as another compute node and users run their applications directly on it without offload from a host system
- In order to run natively, an application has to be cross-compiled for Xeon Phi operating environment
- Intel Composer XE provides simple switch (`-mmic`) to generate cross-compiled code
- Execute the compiled code on host or `ssh` to `mic0` to run on Phi

Native Mode

Compile and run the same code on the coprocessor in the native mode:

```
user@host% icc hello.c -mmic
user@host% scp a.out mic0:~/
a.out 100% 10KB 10.4KB/s 00:00
user@host% ssh mic0
user@mic0% pwd
/home/user
user@mic0%
ls a.out
user@mic0% ./a.out
Hello world! I have 240 logical cores.
user@mic0%
```

- Use –mmic to produce executable for MIC architecture
- Must transfer executable to coprocessor (or NFS-share) and run from shell

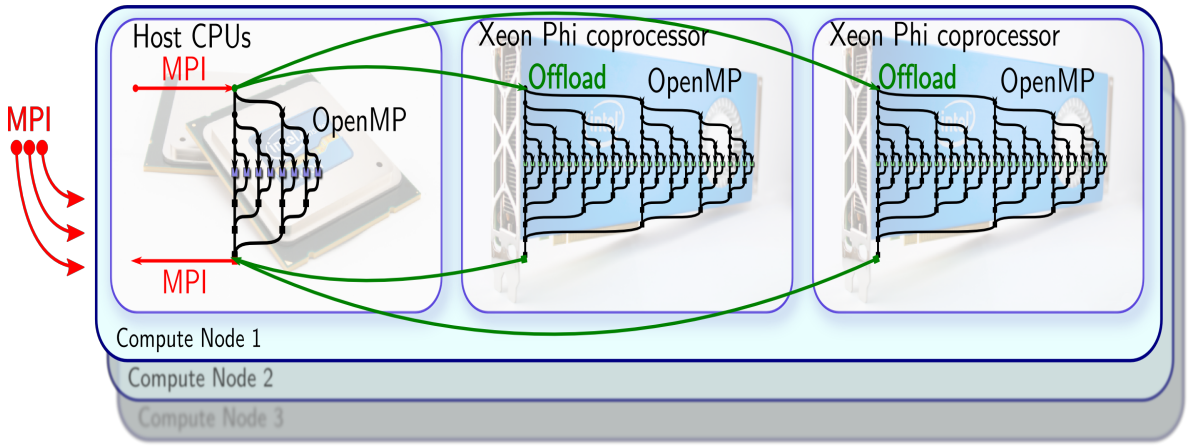
  Native MPI applications work the same way (need Intel MPI library)

Intel Xeon Phi Execution Models:
Symmetric Execution Mode

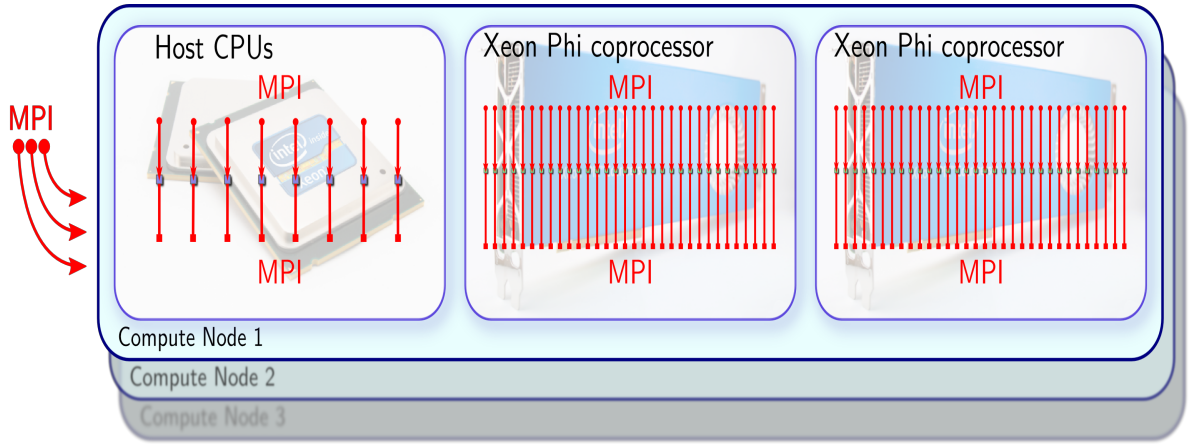
- In this case the application processes run on both the host and the Intel Xeon Phi coprocessor
- They usually communicate through a message passing interface
- This execution environment treats Xeon Phi card as another node in a cluster in a heterogeneous cluster environment
- Note: Load-balancing between Xeon cores and Xeon Phi cores needed

Clusters with Intel Xeon Phi Coprocessors

- Option 1: run MPI processes on hosts and offload to coprocessors:



- Option 2: MPI as native applications on hosts and coprocessors:



Ch5 - Storage: GFS & HDFS

- GFS
  - Easy maintenance, the cheapest solution for Google
  - Architecture
  - Read, write, append simultaneously
  - Advance features: (1) High applicability, (2) High scalability, (3) Fault-tolerance
- HDFS
  - Hadoop = HDFS + MapReduce
  - Architecture

GFS

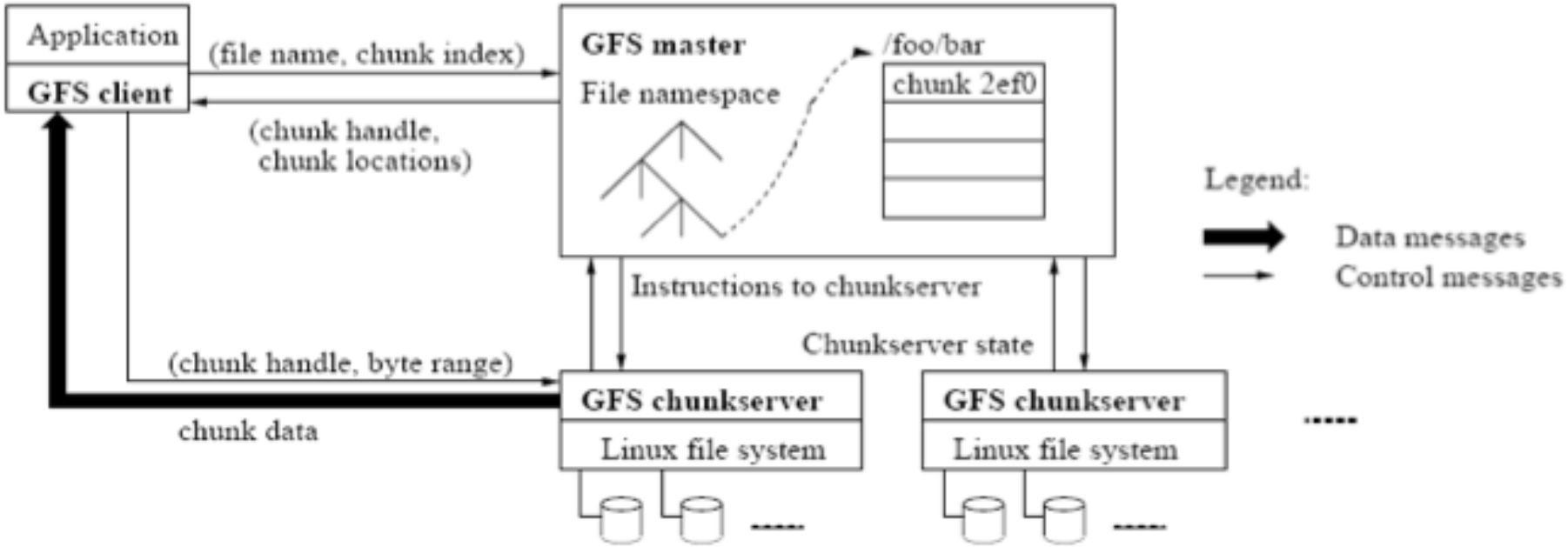
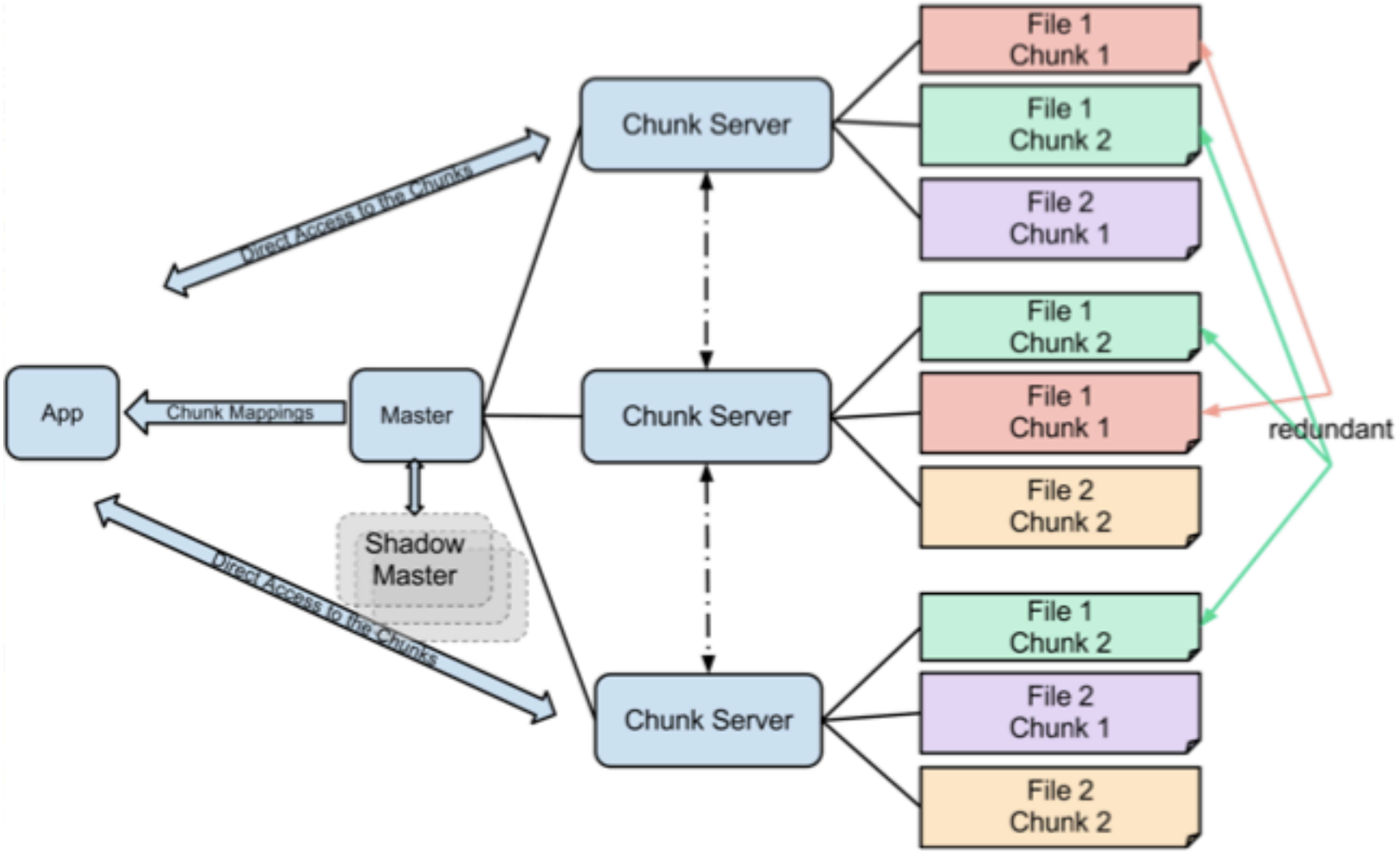


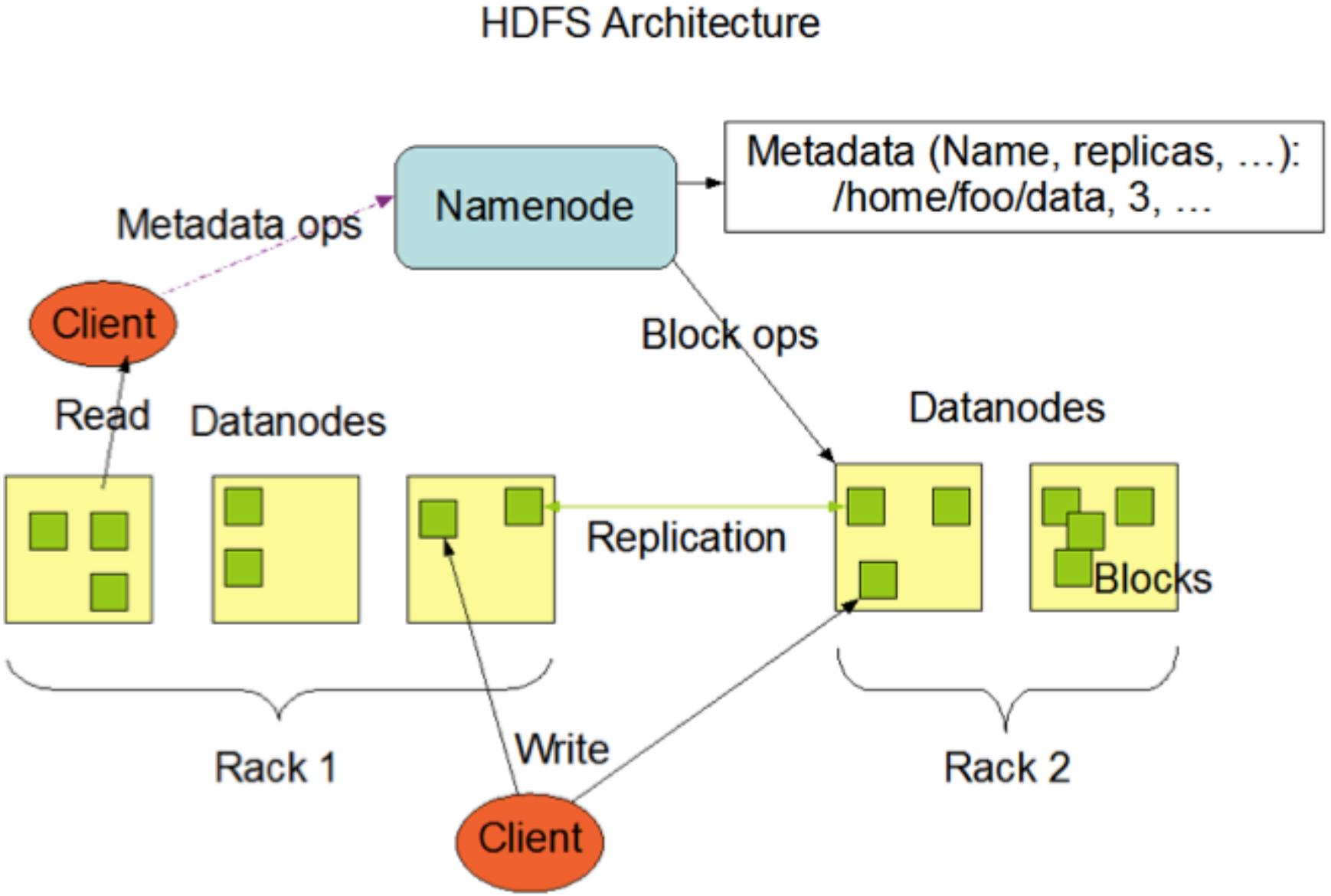
Figure 1: GFS Architecture

GFS


Direct Access to the Chunks
Chunk Server
File 1
Chunk 1
File 1
Chunk 2
File 2
Chunk 1
App
Chunk Mappings
Master
Chunk Server
File 1
Chunk 2
File 1
Chunk 1
File 2
Chunk 2
redundant
Shadow
Master
Direct Access to the Chunks
Chunk Server
File 1
Chunk 2
File 2
Chunk 1
File 2
Chunk 2

HDFS

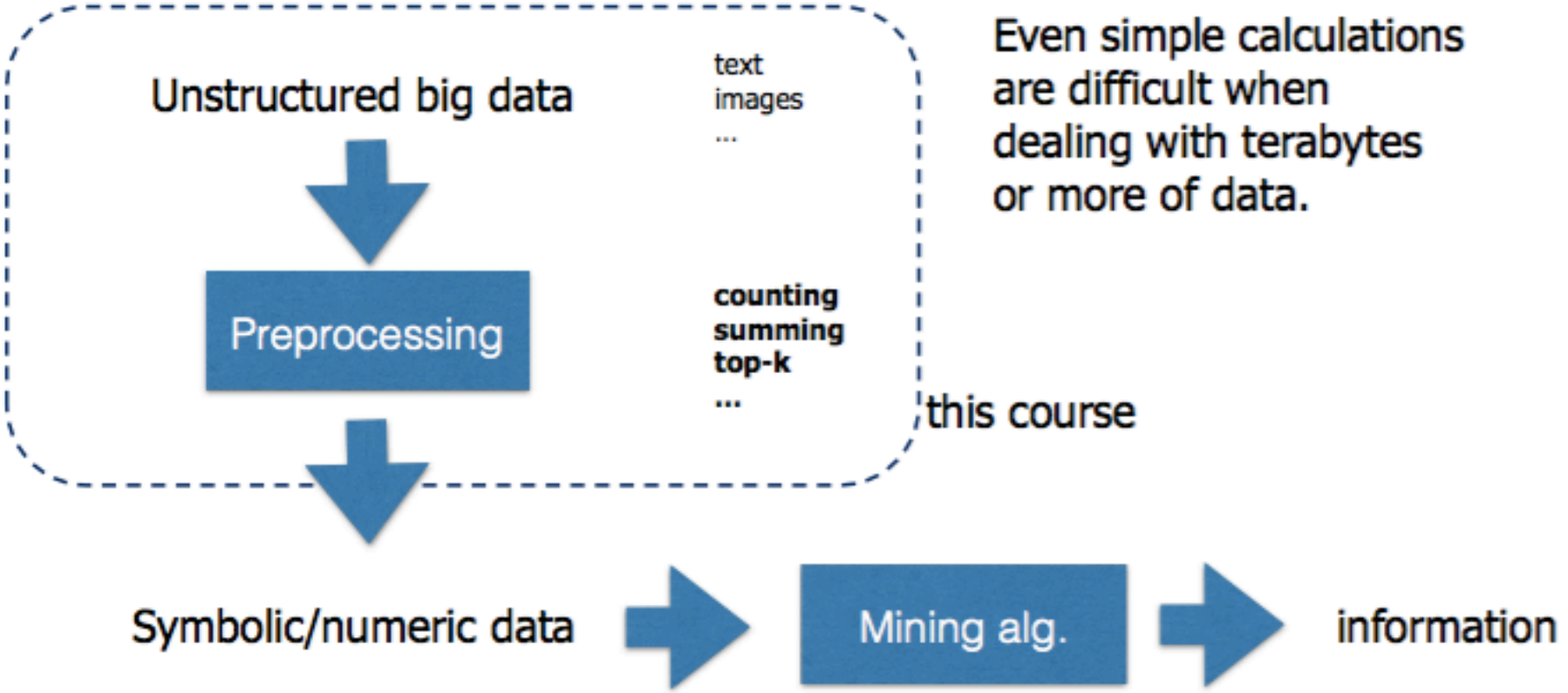
- Master / worker design
- Runs on commodity hardware
- HDFS is resilient (even in case of node failure)
- Data is replicated
- HDFS is better suited for large files
- Files are write-once only (not updateable)


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

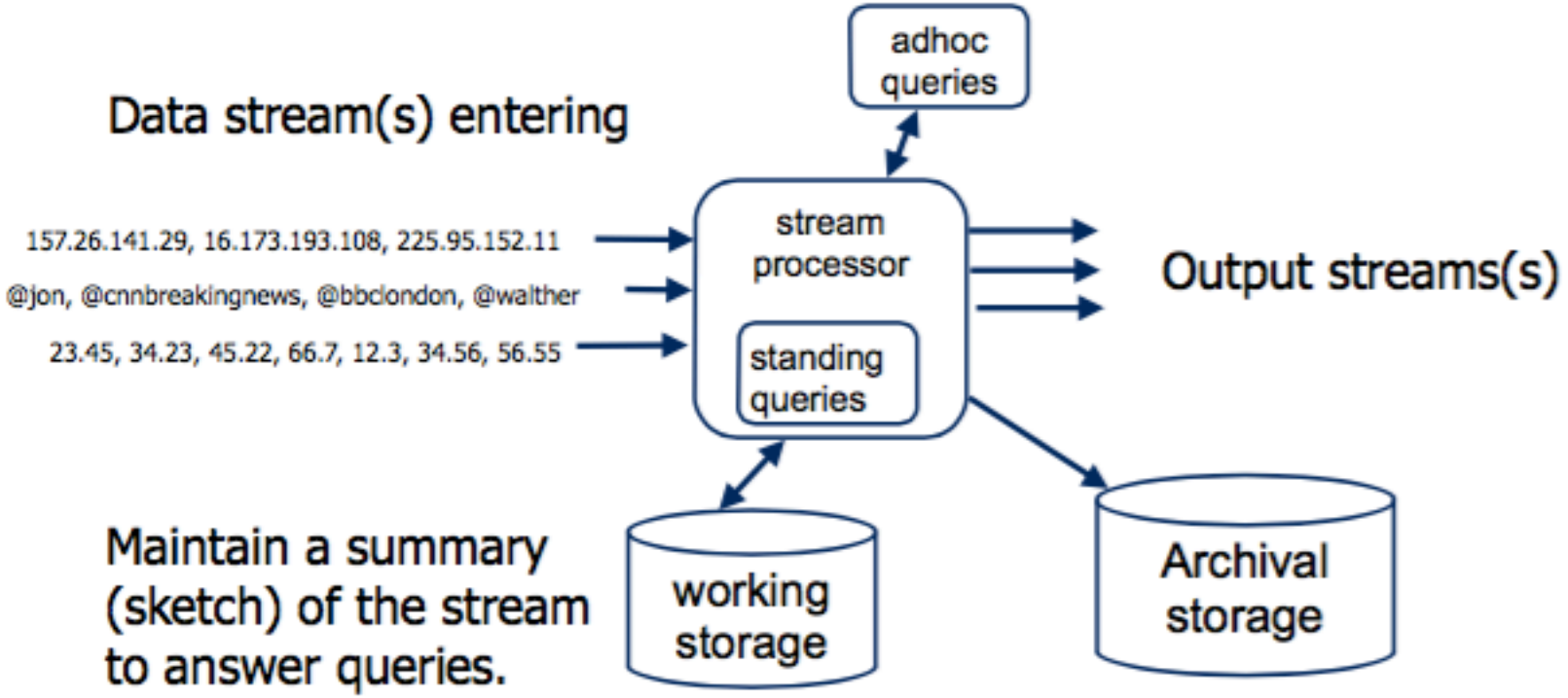
Ch6 - Streaming & data stream

- Architecture
- Models
- Algorithms
  - Sampling
  - Frequency counter
  - Filtering

BDP vs. Data Mining, Pattern Recognition, AI, …



Streaming architecture



Data streaming scenario

- **Continuous** and rapid input of data
- **Limited memory** to store the data (less than linear in the input size)
- **Limited time** to process each element
  Sequential access
- Algorithms have **one** or very **few passes** over the data

Data streaming scenario

- Typically: **simple functions** of the stream are computed and used as input to other algorithms
  - Number of distinct items
  - Heavy hitters
  - Longest increasing subsequence
  - ....
- Closed form solutions are rare – **approximation** and **randomization** are the norm

Data streaming algorithm

- Sampling data in a stream
  - Reservoir sampling
- Filtering streams
  - Bloom filter
- Counting distinct elements
  - Flajolet-Martin

Ch7 – Classification and Clustering

- Classification
  - Decision tree
  - Bayes method
  - Rule-based method
- Clustering
  - K-means

Ch8 - Big data analytics: Machine Learning (ML)

- What is ML?
- Big data & ML
- Linear Regression

Ch9 - Big data visualization

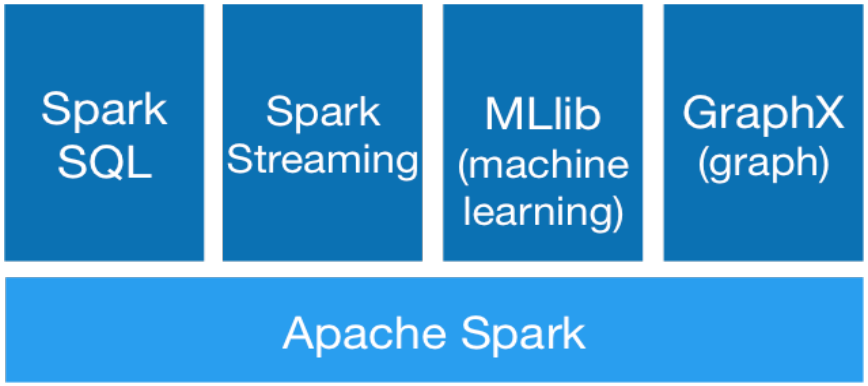
- Data Visualization
- Big Data Visualization
  - Challenges
  - Techniques
- How can we visualize big data
  - Key techniques (Open source tools)

Ch10 - Graph computing

- Real-world problems
- Adjacency matrices, adjacency list
- Algorithms
  - Breadth-first search
  - Single-source shortest path
  - Betweenness
- Web graph
  - PageRank
- (Pregel, BSP, Giraph)

Ch11 - Spark and Data analytics

- Spark architecture
- Advantages of Spark
- RDDs (resilient distributed dataset)



Ch12 – Big data on Time Series

- Time series
- Applications

Đề thi

- Thời gian 90 phút
- Trắc nghiệm 20 câu: 5 điểm
- Tự luận 3-5 câu: 5 điểm

- Tài liệu: được phép sử dụng

Điểm tổng kết

Cuối kỳ: 50%

Seminar: 20%

Tiểu luận: 30%