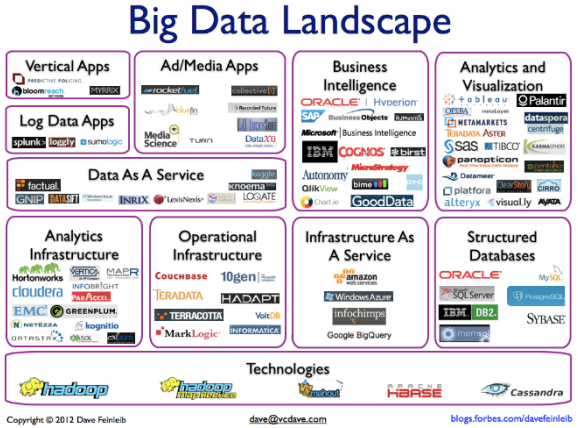


Google big data techniques (2)

Lecturer: Jiaheng Lu

Fall 2016

Outline

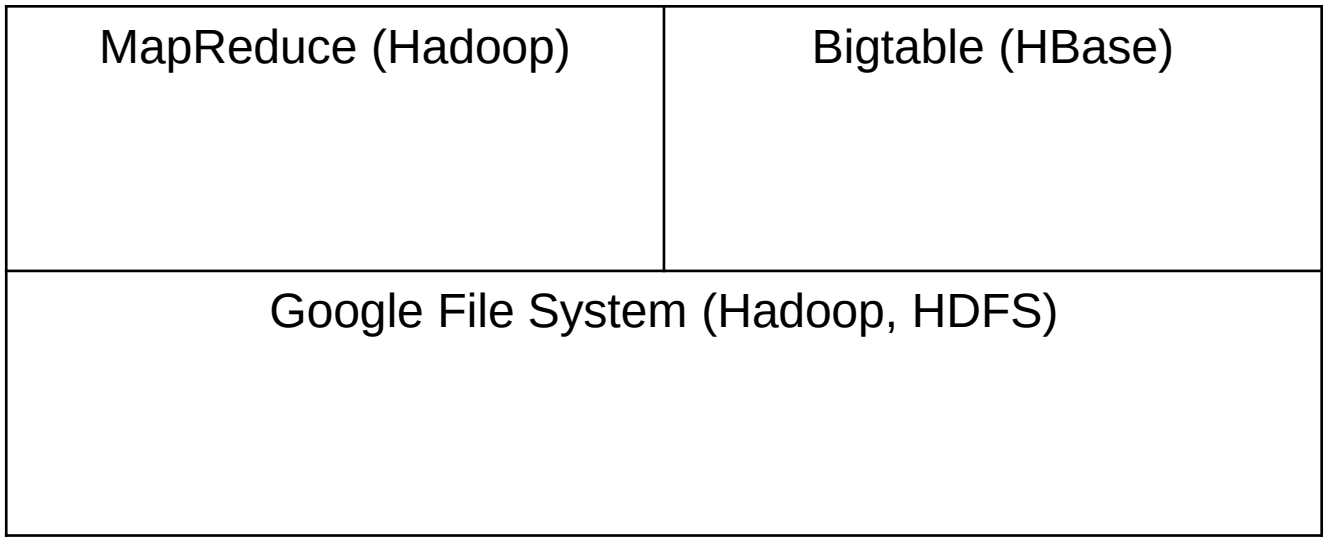
- Google File System and HDFS
- Relational DB V.S. Big data system
- Google Bigtable and NoSQL databases

Google		Apache
GFS	→	HDFS
Map Reduce	→	Apache's Map Reduce
Big Table	→	Apache's HBASE

The Google File System

The Google File System(GFS)

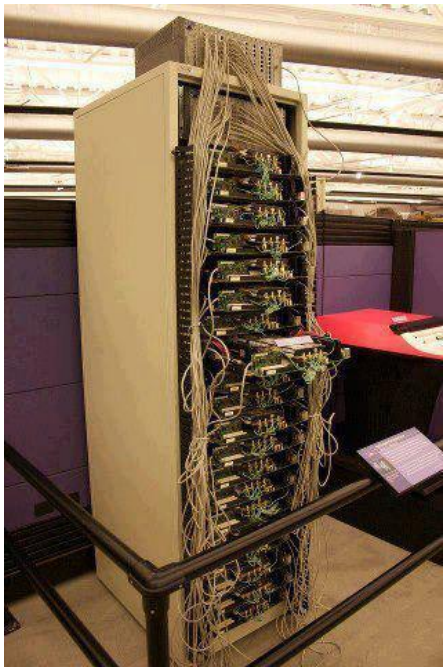
- A scalable distributed file system for large distributed data intensive applications



Motivation: Google history

Early days at Stanford (1995)
Larry Page and Sergey Brin

Google's first production server rack, 1998



Today…

Google Server Farms
Google has more than 900,000 servers,
indexed more than 50 billion pages

GFS: Introduction

- Shares many same goals as previous distributed file systems
  - performance, scalability, reliability, etc
  - GFS design has been driven by four key observation of Google application workloads and technological environment

Intro: Observations

- **1. Component failures are the norm**
  - constant monitoring, error detection, fault tolerance and automatic recovery are integral to the system
- **2. Huge files (by traditional standards)**
  - Multi GB files are common
  - I/O operations and blocks sizes must be revisited

Intro: Observations (Contd)

- **3. Most files are mutated by appending new data**

  This is the focus of performance optimization and atomicity guarantees

- **4. Co-designing the applications and APIs benefits overall system by increasing flexibility**

The Design

- *Cluster consists of a single* master *and multiple* chunkservers *and is accessed by multiple* clients

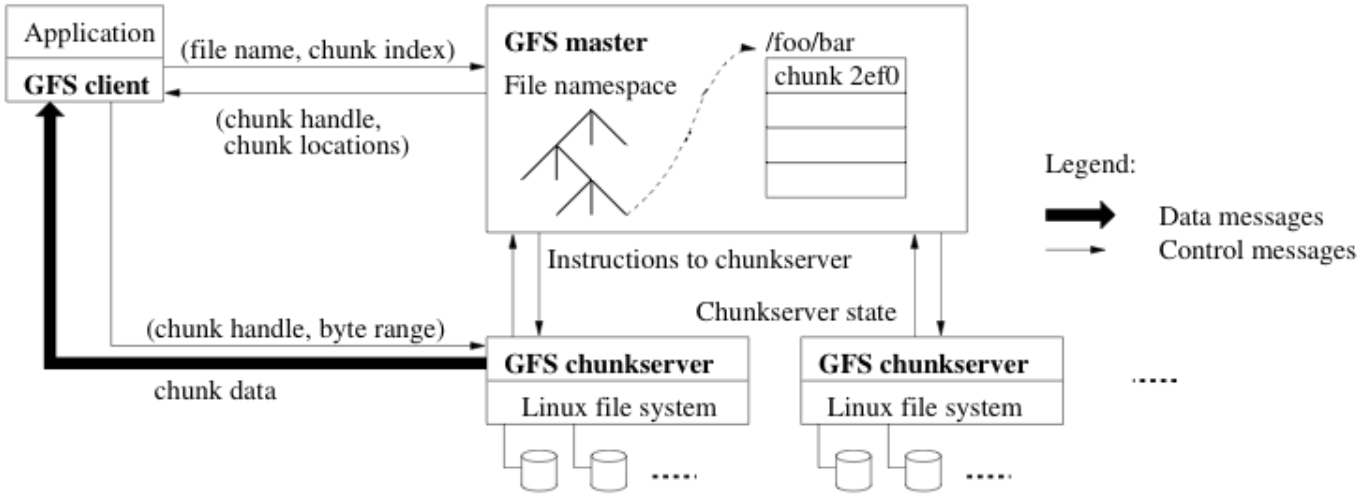
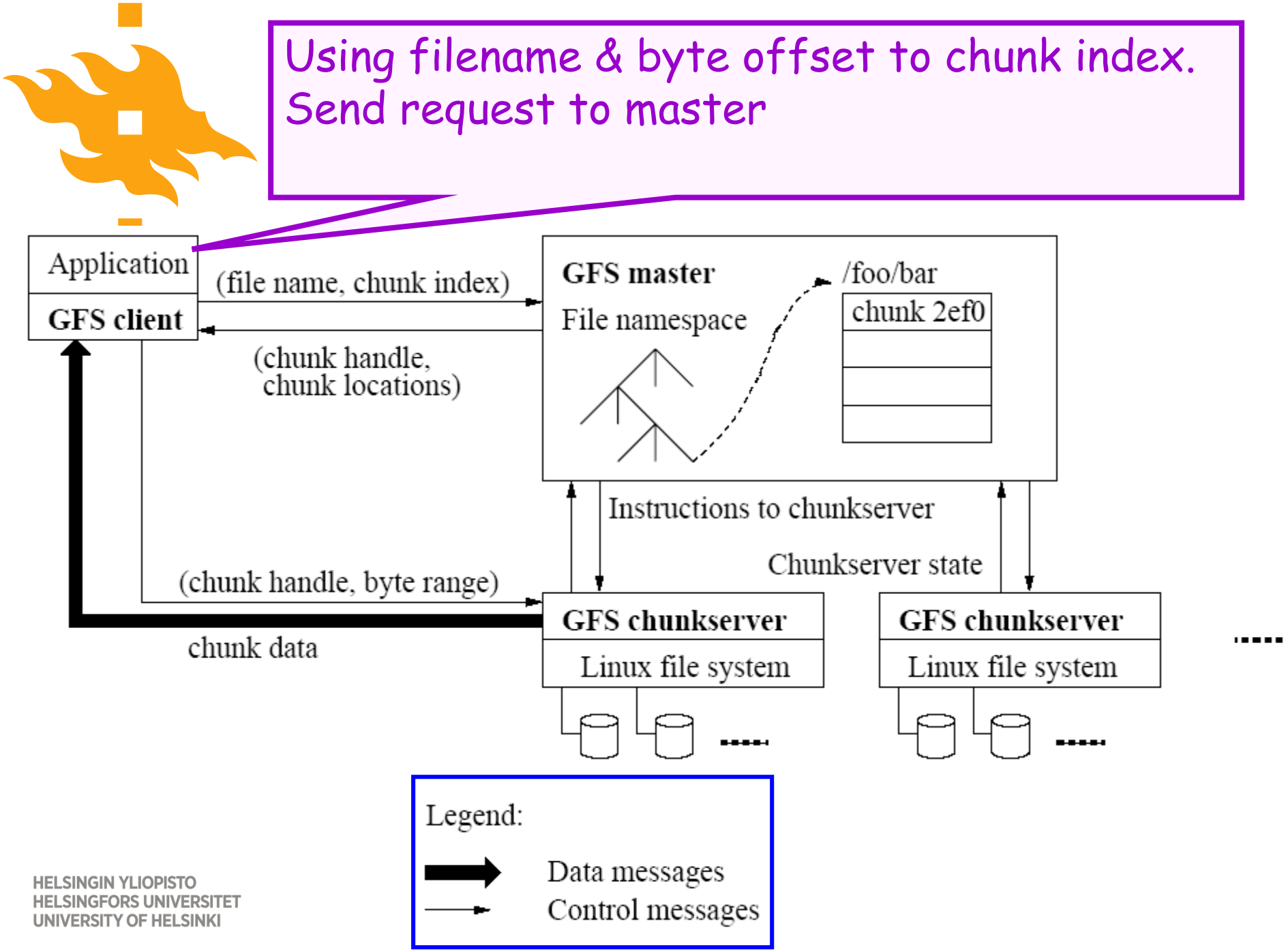
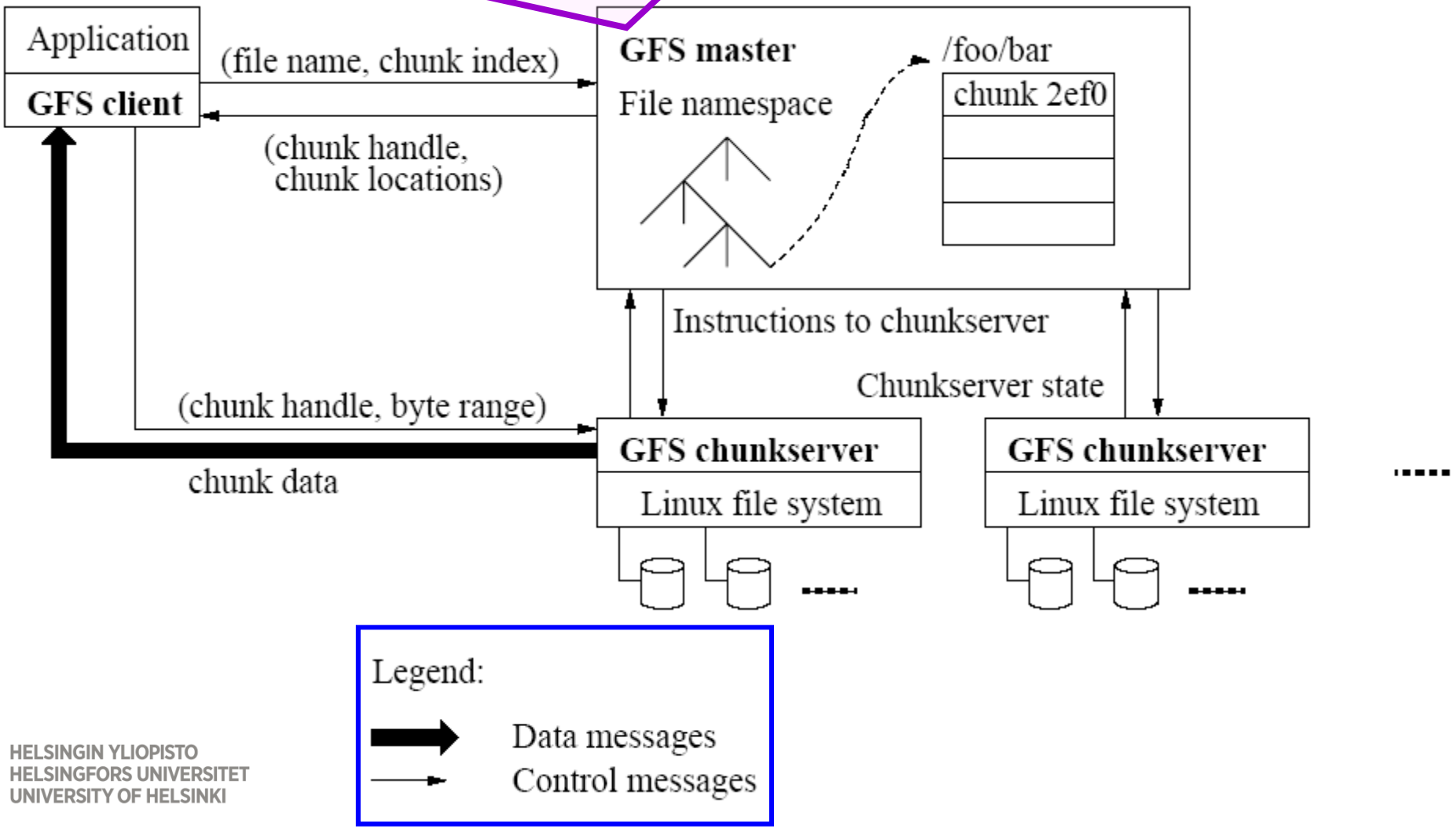


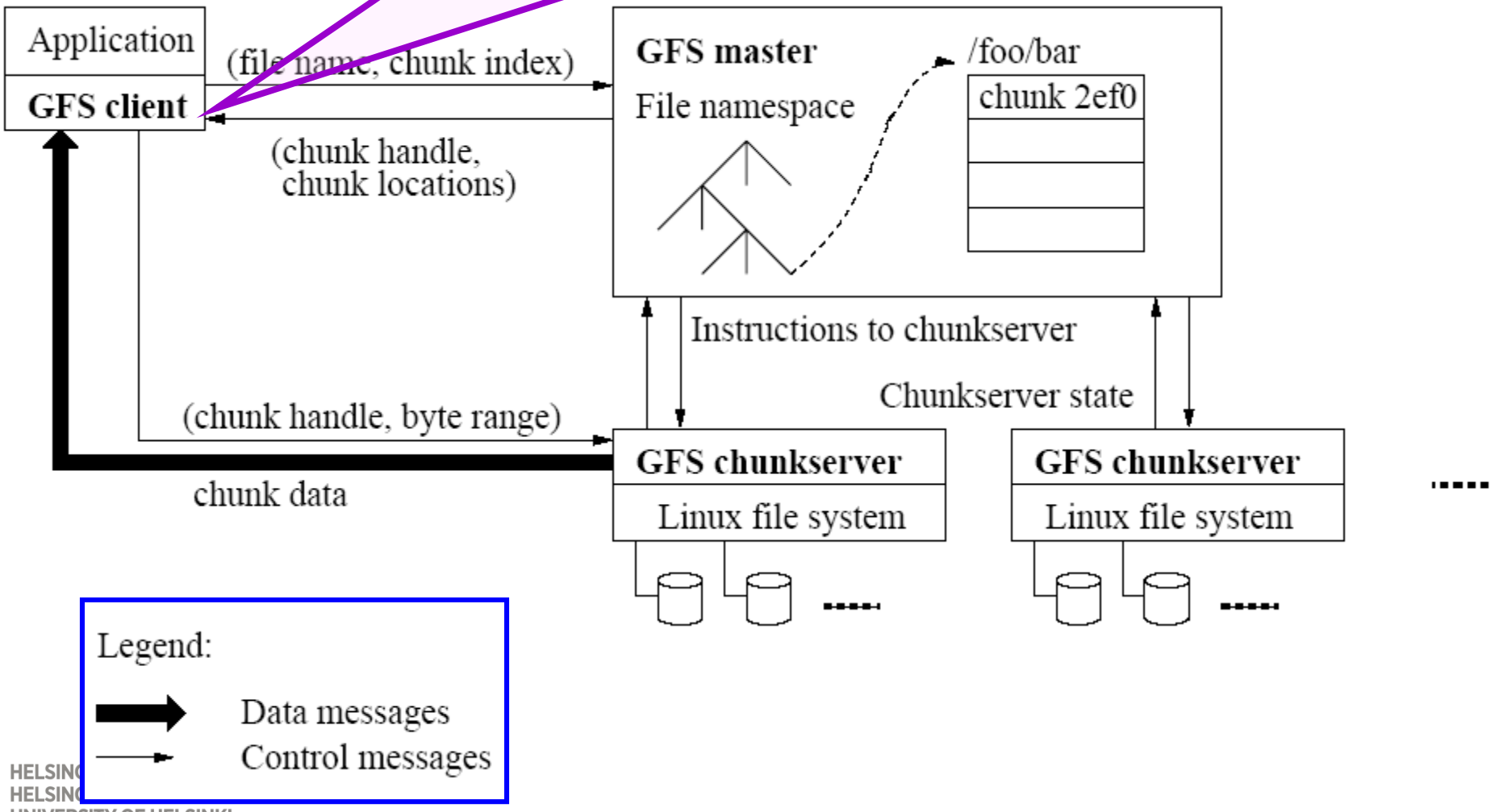
Figure 1: GFS Architecture

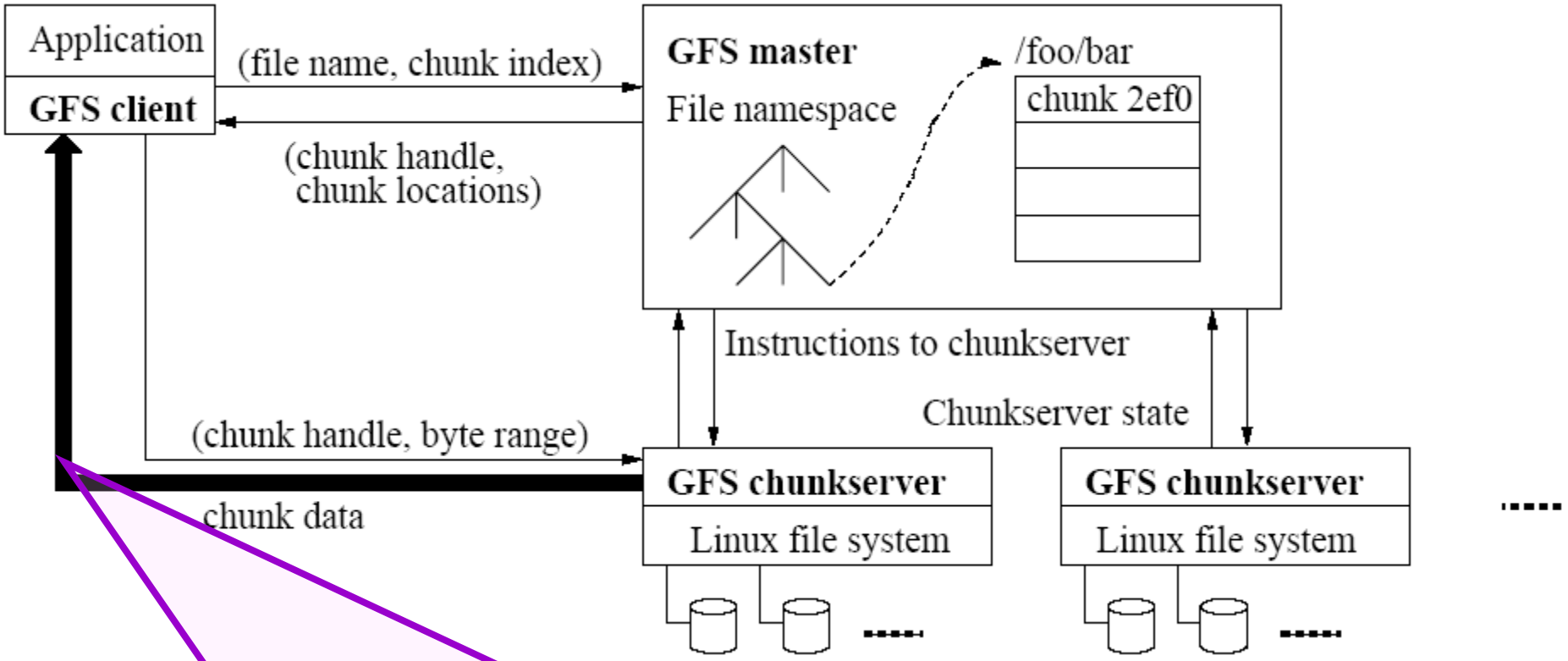

Using filename & byte offset to chunk index.
Send request to master
Application
GFS client
(file name, chunk index)
(chunk handle,
chunk locations)
GFS master
File namespace
/foo/bar
chunk 2ef0
Instructions to chunkserver
Chunkserver state
(chunk handle, byte range)
chunk data
GFS chunkserver
Linux file system
GFS chunkserver
Linux file system
Legend:
Data messages
Control messages

Replies with chunk handle & location of chunkserver replicas (including which is 'primary')



using filename & chunk index as key

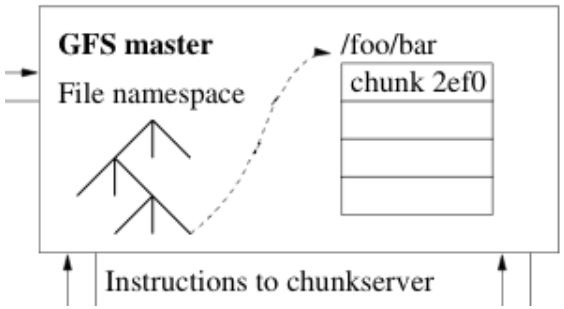




Request data from nearest chunkserver
"chunkhandle & index into chunk"

The Master

- *Maintains all file system metadata.*
  - *names space, access control info, file to chunk mappings, chunk (including replicas) location, etc.*
- *Periodically communicates with chunkservers in* HeartBeat *messages to give instructions and check state*

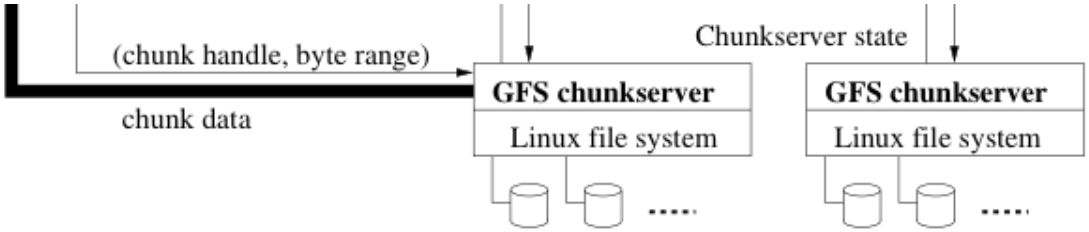


The Master

- *Helps make sophisticated chunk placement and replication decision, using global knowledge*
- *For reading and writing, client contacts Master to get chunk locations, then deals directly with chunkservers*
  - *Master is not a bottleneck for reads/writes*

Chunkservers

- *Files are broken into* chunks. *Each chunk has a globally unique 64-bit* chunk-handle.
  - *handle is assigned by the master at chunk creation*
- ***Chunk size is 64 MB***
- *Each chunk is replicated on 3 (default) servers*



Clients

- *Linked to apps using the file system API.*
- *Communicates with master and chunkservers for reading and writing*
  - *Master interactions only for metadata*
  - *Chunkserver interactions for data*
- *Only caches metadata information*
  - *Data is too large to cache.*

GFS paper

- More information on data update and performance of GFS, read the original paper:

- http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

Google

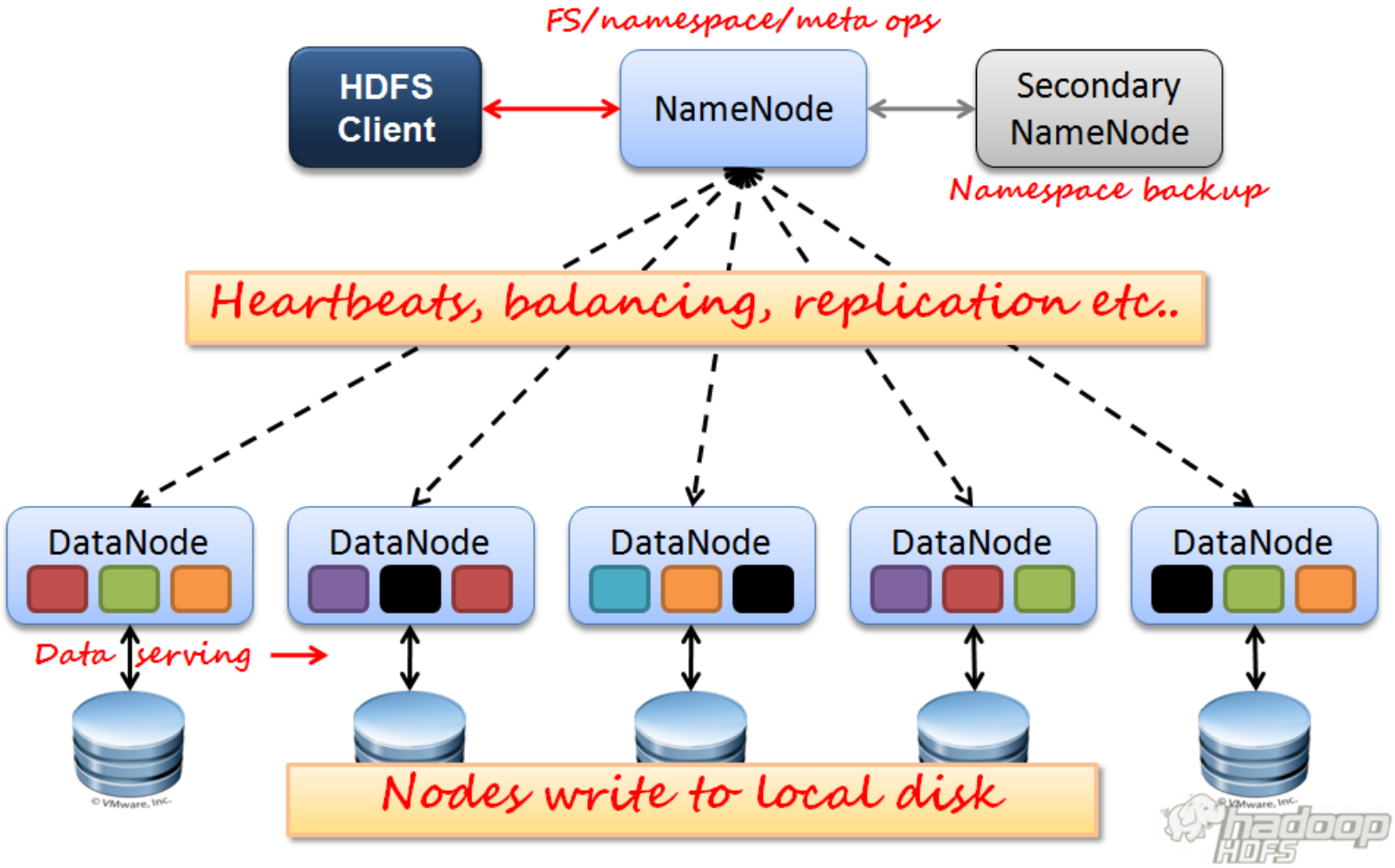
Apache

GFS → HDFS

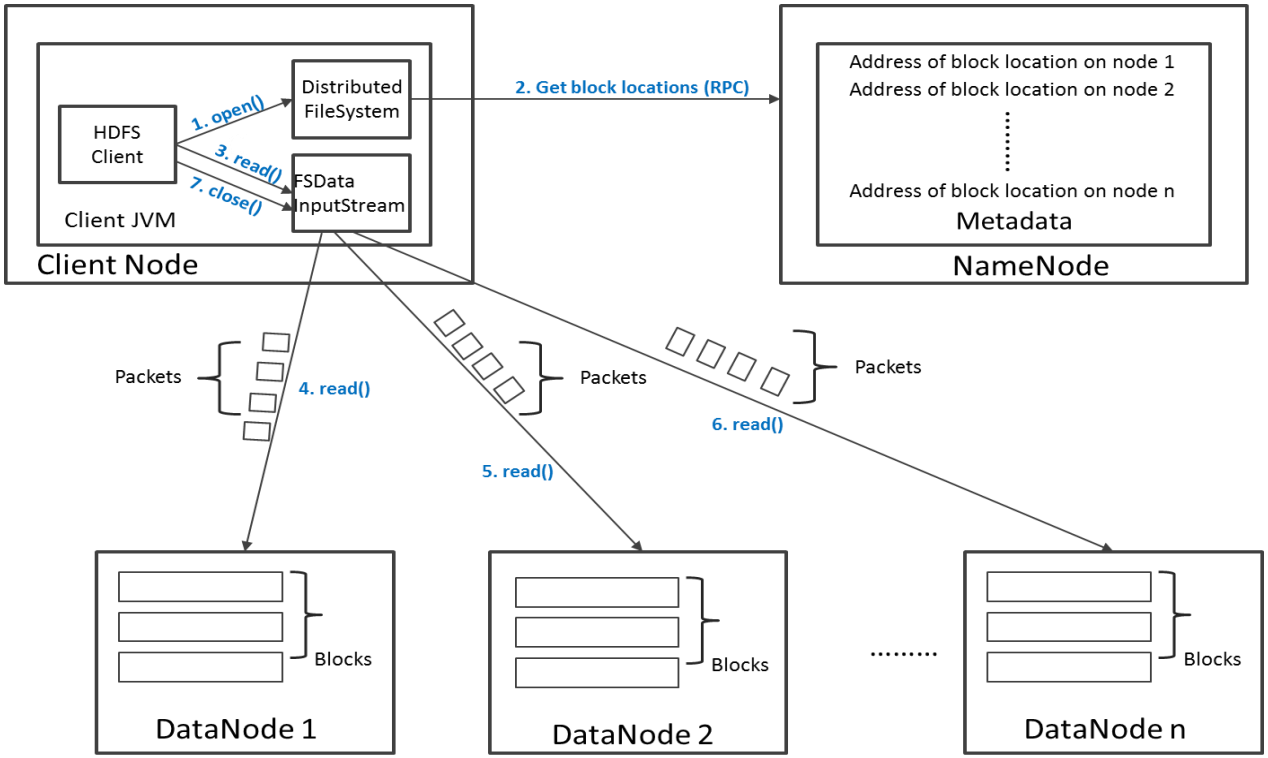
Map Reduce → Apache's Map Reduce

Big Table → Apache's HBASE

HDFS Architecture



Read Operation in HDFS



Disclaimer

- GFS/HDFS are not a good fit for:
- Low latency data access (in the milliseconds range)
- Many small files
- Constantly changing data

- Watch a video on “How big is Google”

- https://www.youtube.com/watch?v=-79uIRQiAFM

- Google File System and HDFS
- Relational DB V.S. Big data system
- Google Bigtable and NoSQL databases

Difference between the traditional DBMS and big data system(1)

- Query languages are different
- Relational DB uses SQL language
- Big data system uses NoSQL language: XML, JSON, Graph query language.

Difference between the traditional DBMS and big data system(2)

- System architectures are different
- Most relational DB uses a single machine, scale-up
- Big data system uses multiple machines, scale-out
  - Potentially thousands of machines
  - Potentially distributed around the world

Difference between the traditional DBMS and big data system(3)

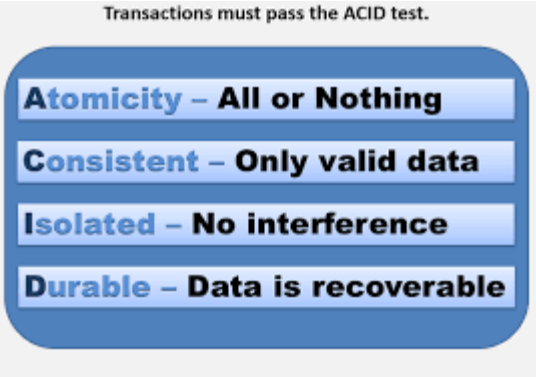
- Models are different
- Most relational DB uses a relational model
- Big data system uses NoSQL model, including graph model, document model, key-value model

Difference between the traditional DBMS and big data system(4)

- Schemas are different
- Most relational DB uses fixed schemas
- Big data system uses flexible schemas or schema-less

Difference between the traditional DBMS and big data system(5)

- Consistency models are different
- Relational DB uses ACID



- Big data system uses weak consistency
  - **B**asically **A**vailable,
  - **S**oft state,
  - **E**ventually Consistent

Difference between the traditional DBMS and big data system(6)

- Update model are different
- DBMS: frequent updates
- Big data: Mostly query, few updates

- Google File System and HDFS
- Relational DB V.S. Big data system
- Google Bigtable and NoSQL databases

BigTable: A Distributed Storage System for Structured Data

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Introduction

- BigTable is a distributed storage system for managing structured data.

- Designed to scale to a very large size
  - Petabytes of data across thousands of servers

Motivation

- Lots of (semi-) structured data at Google
  - URLs:
    - Contents, crawl metadata, links, anchors, pagerank, …
  - Per-user data:
    - User preference settings, recent queries/search results, …
  - Geographic locations:
    - Physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …

Why not just use commercial DB?

- Scale is too large for most commercial databases
- Even if it weren't, cost would be very high
  - Building internally means system can be applied across many projects for low incremental cost
- Low-level storage optimizations help performance significantly
  - Much harder to do when running on top of a database layer

Goals

- Need to support:
  - Very high read/write rates (millions of ops per second)
  - Efficient scans over all or interesting subsets of data
  - Efficient joins of large one-to-one and one-to-many datasets

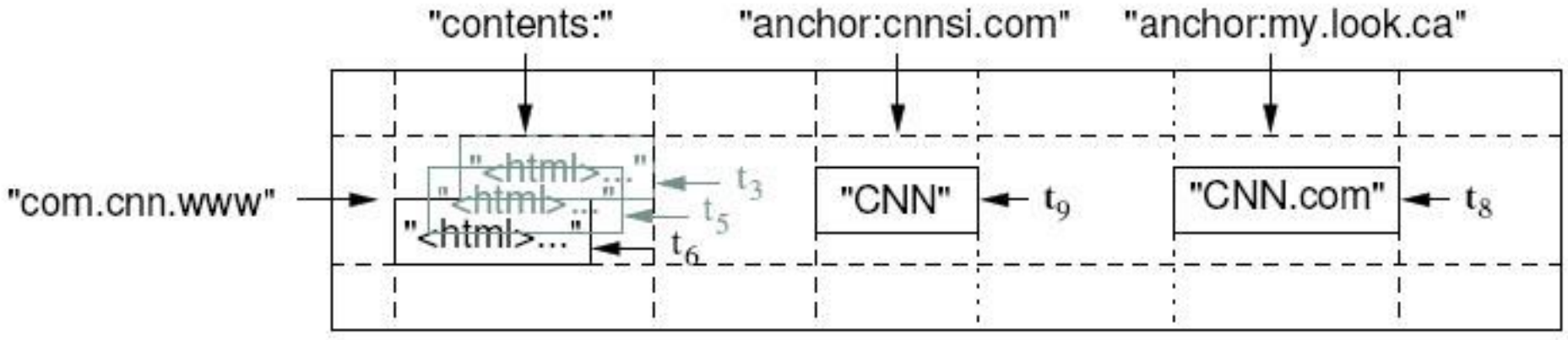
BigTable

- Distributed multi-level map
- Fault-tolerant, persistent
- Scalable
  - Thousands of servers
  - Terabytes of in-memory data
  - Petabyte of disk-based data
  - Millions of reads/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance

Basic Data Model

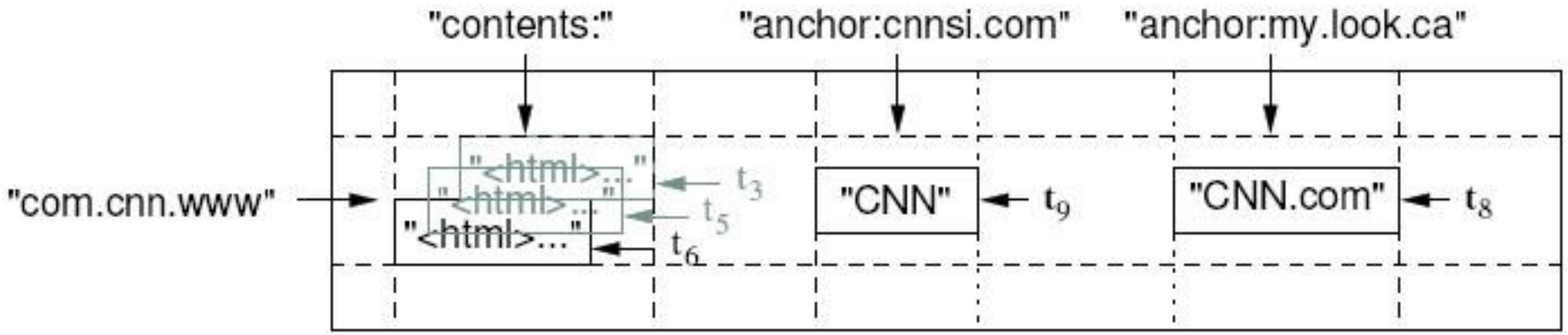
- A BigTable is a sparse, distributed persistent multi-dimensional sorted map

(row, column, timestamp) -> cell contents



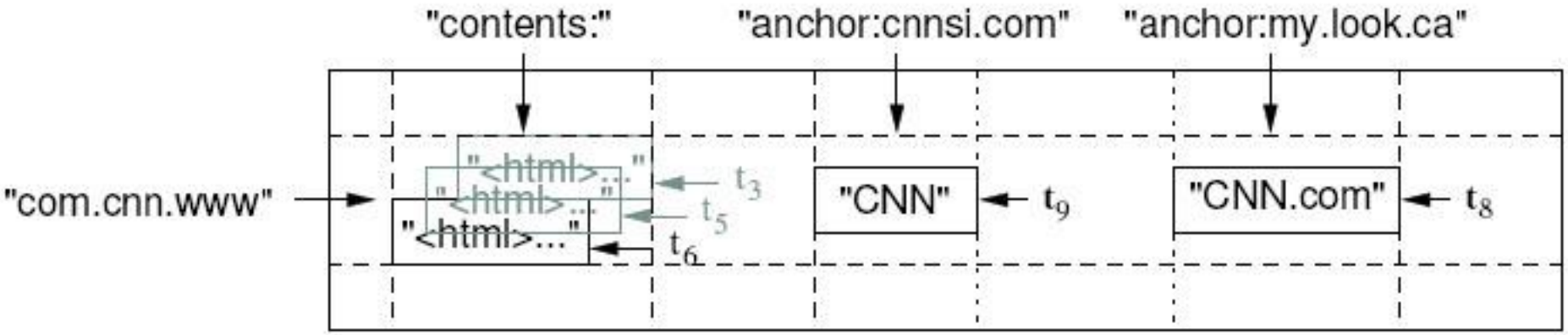
- Good match for most Google applications

WebTable Example



- Want to keep copy of a large collection of web pages and related information
- Use URLs as row keys
- Various aspects of web page as column names
- Store contents of web pages in the `contents:` column under the timestamps when they were fetched.

Rows



- Name is an arbitrary string
  - Access to data in a row is atomic
  - Row creation is implicit upon storing data
- Rows ordered lexicographically
  - Rows close together lexicographically usually on one or a small number of machines

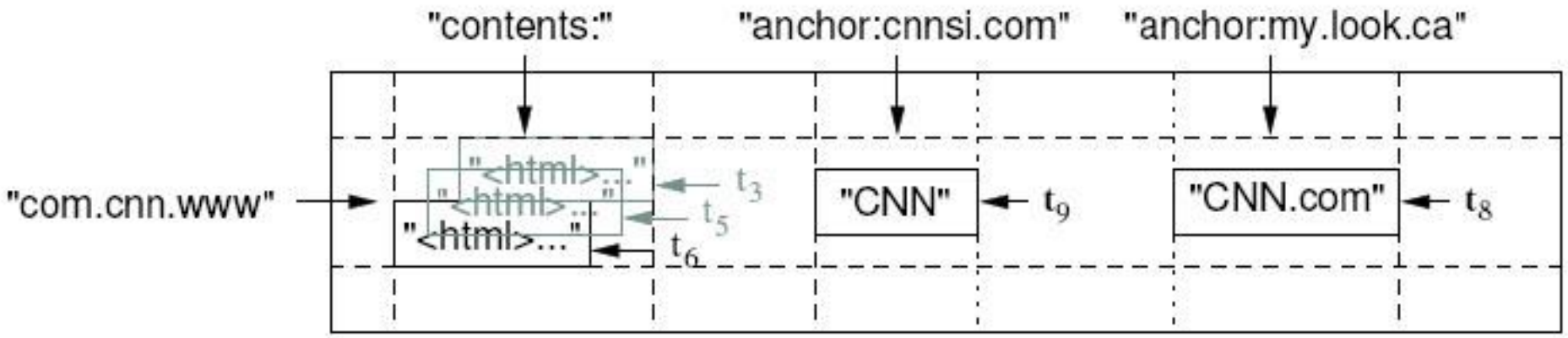
Rows (cont.)

Reads of short row ranges are efficient and typically require communication with a small number of machines.

- Can exploit this property by selecting row keys so they get good locality for data access.
- Example:

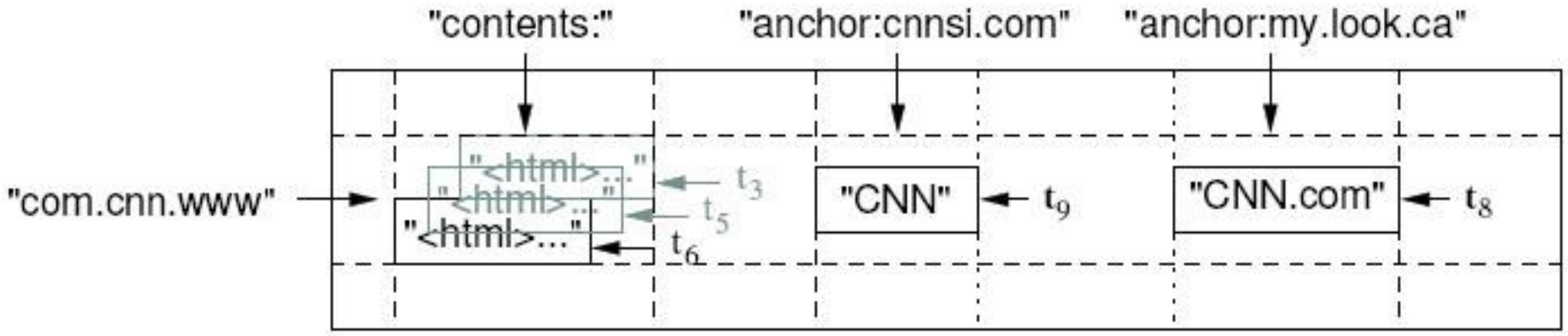
```
math.gatech.edu, math.uga.edu, phys.gatech.edu,
phys.uga.edu

VS

edu.gatech.math, edu.gatech.phys, edu.uga.math,
edu.uga.phys
```

Columns



- Columns have two-level name structure:
  - family:optional_qualifier
- Column family
  - Unit of access control
  - Has associated type information
- Qualifier gives unbounded columns
  - Additional levels of indexing, if desired

Timestamps



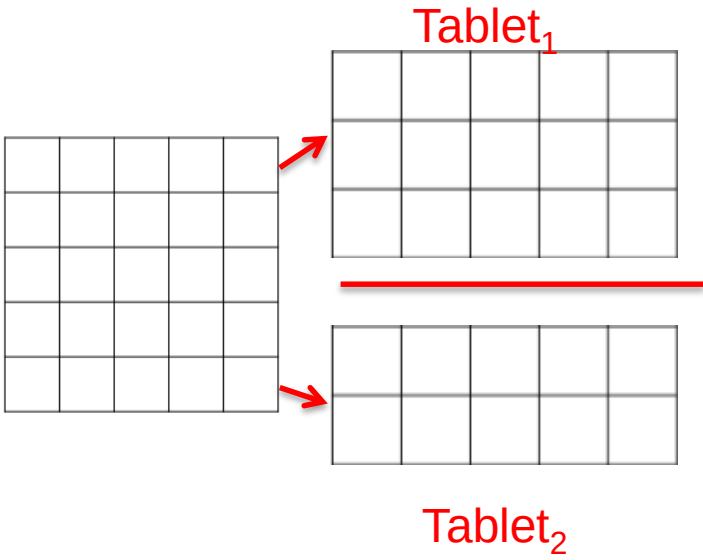
- Used to store different versions of data in a cell
  - New writes default to current time, but timestamps for writes can also be set explicitly by clients
- Lookup options:
  - *"Return most recent K values"*
  - *"Return all values in timestamp range (or all values)"*
- Column families can be marked w/ attributes:
  - *"Only retain most recent K values in a cell"*
  - *"Keep values until they are older than K seconds"*

Implementation – Three Major Components

- Library linked into every client
- One master server
  - Responsible for:
    - Assigning tablets to tablet servers
    - Detecting addition and expiration of tablet servers
    - Balancing tablet-server load
    - Garbage collection
- Many tablet servers
  - Tablet servers handle read and write requests to its table
  - Splits tablets that have grown too large

Tablets

- The entire BigTable is split into tablets of contiguous ranges of rows
  - Approximately 100MB to 200MB each
- One machine services 100 tablets



Tablet Assignment

- Each tablet is assigned to one tablet server at a time.
- Master server keeps track of the set of live tablet servers and current assignments of tablets to servers. Also keeps track of unassigned tablets.
- When a tablet is unassigned, master assigns the tablet to an tablet server with sufficient room.

Bigtable paper

- More details on Bigtable, read the paper:

- http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

Types and examples of NoSQL databases

Types	Examples
Column	Accumulo, Cassandra, Druid, HBase, Vertica
Document	HyperDex, Lotus Notes, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB
Key-value:	Aerospike, Dynamo, FairCom, c-treeACE, HyperDex, MemcacheDB, MUMPS
Graph	Allegro, InfiniteGraph, MarkLogic, Neo4J, OrientDB, Virtuoso, Stardog
Multi-model	Alchemy Database, ArangoDB, CortexDB, FoundationDB, MarkLogic, OrientDB

Column stores

- A column-oriented DBMS is a database management system (DBMS) that stores data tables as sections of columns of data rather than as rows of data.

- This column-oriented DBMS has advantages for data warehouses, clinical data analysis, customer relationship management (CRM) systems, and library card catalogs, and other ad hoc inquiry systems

Example of column stores

RowId	EmpId	Name	Age
1	123	Anna	34
2	456	Mikko	30
3	789	Emilia	44

Row-oriented storage:
1:123, Anna,34; 2:456,Mikko,30; 3:789,Emilia,44

Column-oriented storage:
123:1,456:2, 789:3; Anna:1, Mikko:2,Emilia:3; 34:1,30:2,44:3

Key-value stores

- Key-value (KV) stores use the associative array as their fundamental data model.

- In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection.

Example of Key-value stores

RowId	EmpId	Name	Age
1	123	Anna	34
2	456	Mikko	30
3	789	Emilia	44

1: (123,Anna,34); 2: (2,456,Mikko,30); 3: (789,Emilia,44)

Document store

- The central concept of a document store is the notion of a "document".

- Encodings in use include XML, YAML, and JSON as well as binary forms like BSON.

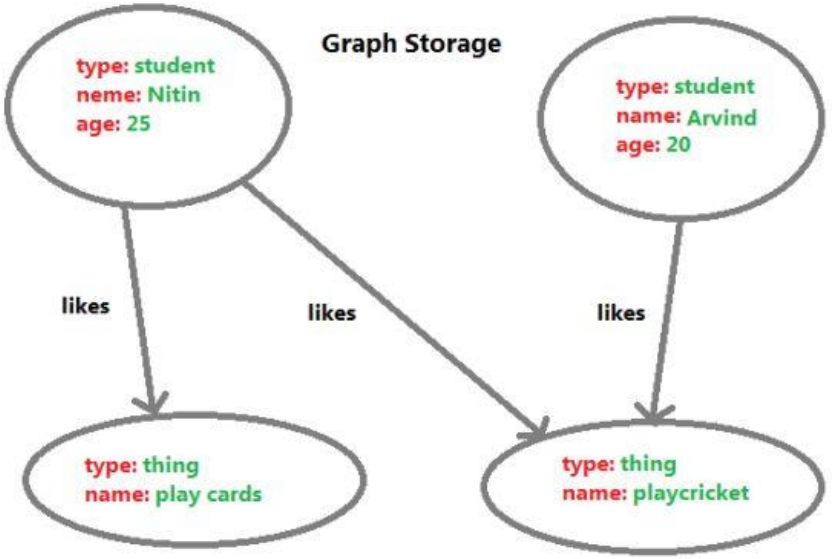
Example of document store

University of Helsinki
Yliopistonkatu 4,
00100 Helsinki
Finland

```
XML: <contact>
   <company> Universtiy of Helsinki </company>
   <address> Yliopistonkatu 4 </address >
   <city>Helsinki</city>
   <zip> 00100 </zip>
   <country>Finland</country>
  </contact>

   JSON: “contact": {
      “company": "Universtiy of Helsinki",
      " address ": " Yliopistonkatu 4 ",
      “city": " Helsinki ",
      “zip": "00100“,
     “country”:”Finland”
    },
```

Graph stores

- Designed for graph data
- Applications: social relations, public transport links, road maps or network topologies, etc.



Multi-model stores

- Support multiple data models against a single, integrated backend: Document, graph, relational, and key-value models are examples of data models

Database	Key-value	SQL	Document	Graph	Object	Transaction
OrientDB	Yes	Yes	Yes	Yes	Yes	Full ACID, even distributed
Couchbase	Yes	Yes	Yes	No	Yes	
Marklogic	Yes	Yes	Yes	Yes	No	Full ACID

- Watch a short video on NoSQL database
- https://www.youtube.com/watch?v=qUV2j3XBRHc&t=24s

Summary of this course

- Three topics on big data
- Data models (relation, XML, JSON, graph)
- Data sketches (Bloom filter, Count-min, Count-sketch, FM sketch)
- Google big data techniques (MapReduce, GFS, Bigtable)

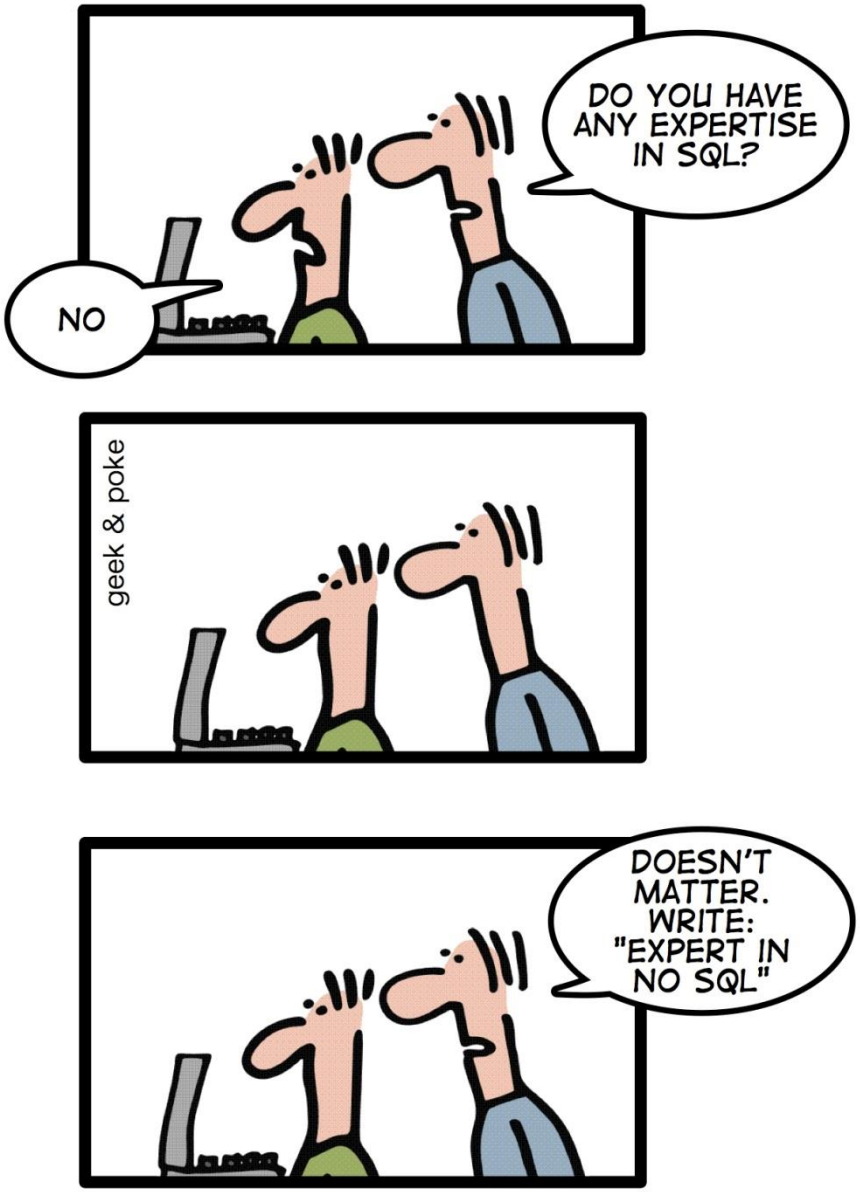
Examination

- **Examination time: 21.12 Wednesday 8.00AM Room CK112**

- The exam covers the lectures (including self-assessment questions) and the exercises.

- The exam lasts 2.5 hours. No notes, computer or other material is allowed in the exam.

HOW TO WRITE A CV



Leverage the NoSQL boom