Google Bigtable

Aishwarya Panchbhai

Plan for today …

- Google Scale – Motivation for Bigtable
- How do existing storage solutions compare?
- Overview of Bigtable – Data Model
- A Typical Bigtable Cell
- Compactions
- Performance Evaluation
- Lessons learnt

Google Scale

- **Workload**

  - Tens of billions of documents/ hundreds ?

  - 10 kb/doc => 100’s of Terra bytes

  - Web growing at ~ 5 Exabytes/year (growing at 30 %) *

  Q: How much is an Exabyte ? 1000^6

- **Lots of Different kinds of data!**

  - Crawling system

    URL’s, contents, links, anchors, pagerank etc

  - Per-user data: preferences, recent queries/ search history

  - Geographic data, images etc …

** Source: How much information is out there?*

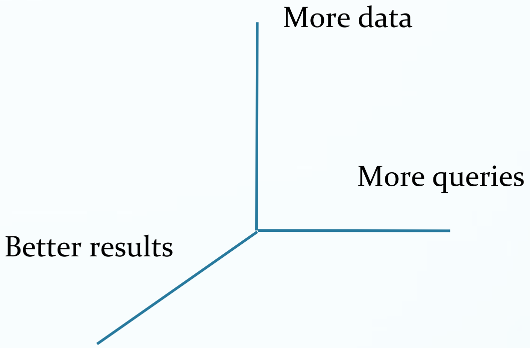
Google Philosophy

- Problem : Every Google service sees continuing growth in computational needs
  - More Queries

    More Users
  - More Data

    Bigger web, mailbox, blog etc
  - Better Results

    Find the Right information, and find it faster



- Solution?

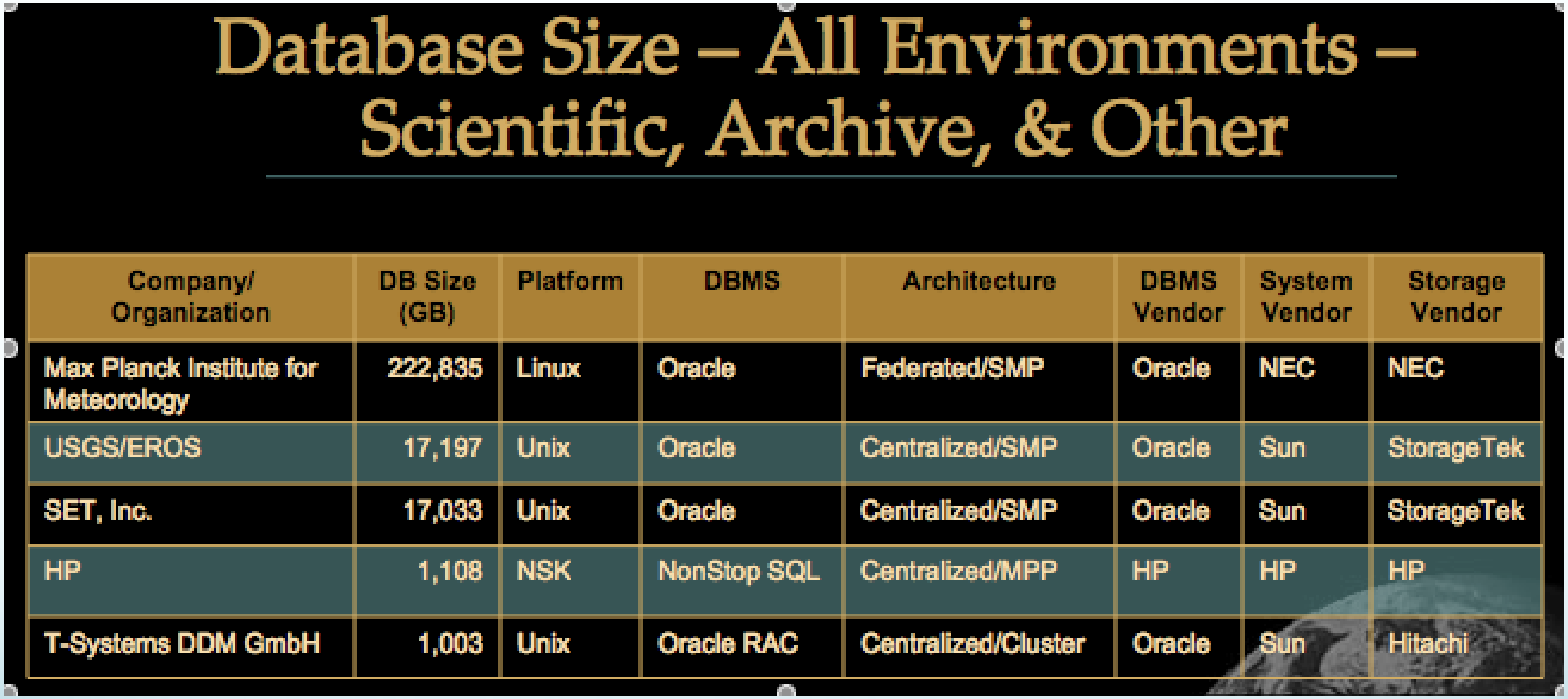
Need for more computing power – large, scalable infrastructure

Existing storage solutions?

- Scale is too large for commercial databases
- May not run on their commodity hardware
- No dependence on other vendors
- Optimizations
- Better Price/Performance
- Building internally means the system can be applied across many projects for low incremental cost.

Q: How much is the largest database installation ?

2005 WinterCorp TopTen Survey



Database Size – All Environments – Scientific, Archive, & Other

Company/ Organization	DB Size (GB)	Platform	DBMS	Architecture	DBMS Vendor	System Vendor	Storage Vendor
Max Planck Institute for Meteorology	222,835	Linux	Oracle	Federated/SMP	Oracle	NEC	NEC
USGS/EROS	17,197	Unix	Oracle	Centralized/SMP	Oracle	Sun	StorageTek
SET, Inc.	17,033	Unix	Oracle	Centralized/SMP	Oracle	Sun	StorageTek
HP	1,108	NSK	NonStop SQL	Centralized/MPP	HP	HP	HP
T-Systems DDM GmbH	1,003	Unix	Oracle RAC	Centralized/Cluster	Oracle	Sun	Hitachi

Bigtable

- Distributed multi-level map
- Fault-tolerant, persistent => GFS
- Scalable

  - 1000’s of servers

  - Millions of reads/writes, efficient scans
- Self-managing

  - Servers can be added/removed dynamically

  - Servers adjust to load-imbalance

Bigtable Vs DBMS

- Fast Query rate

  - No Joins, No SQL support, column-oriented database

  - Uses one Bigtable instead of having many normalized tables

- Is not even in 1NF in a traditional view

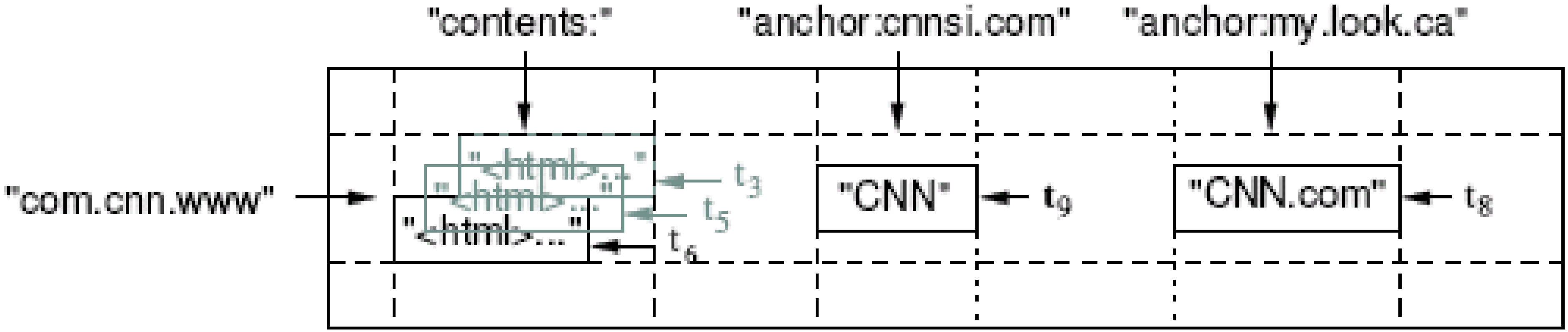
- Designed to support historical queries

  timestamp field => what did this webpage look like yesterday ?

- Data compression is easier – rows are sparse

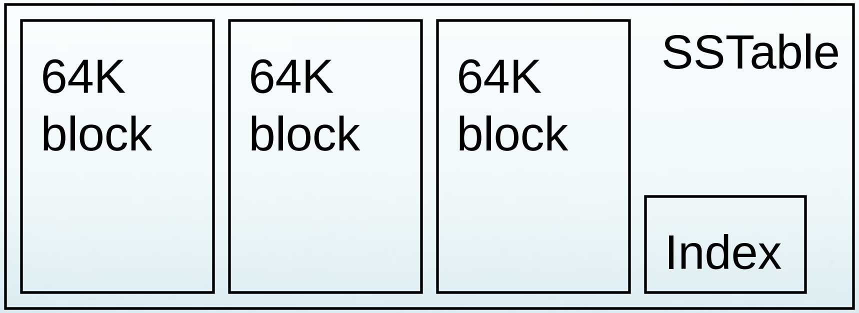
Data model: a big map

- <Row, Column, Timestamp> triple for key - lookup, insert, and delete API
- Arbitrary “columns” on a row-by-row basis
    - Column family:qualifier. Family is heavyweight, qualifier lightweight
    - Column-oriented physical store- rows are sparse!
- Does not support a relational model
    - No table-wide integrity constraints
    - No multirow transactions



SSTable

- Immutable, sorted file of key-value pairs
- Chunks of data plus an index
  - Index is of block ranges, not values



Tablet

- Large tables broken into tablets at row boundaries

  - Tablets hold contiguous rows

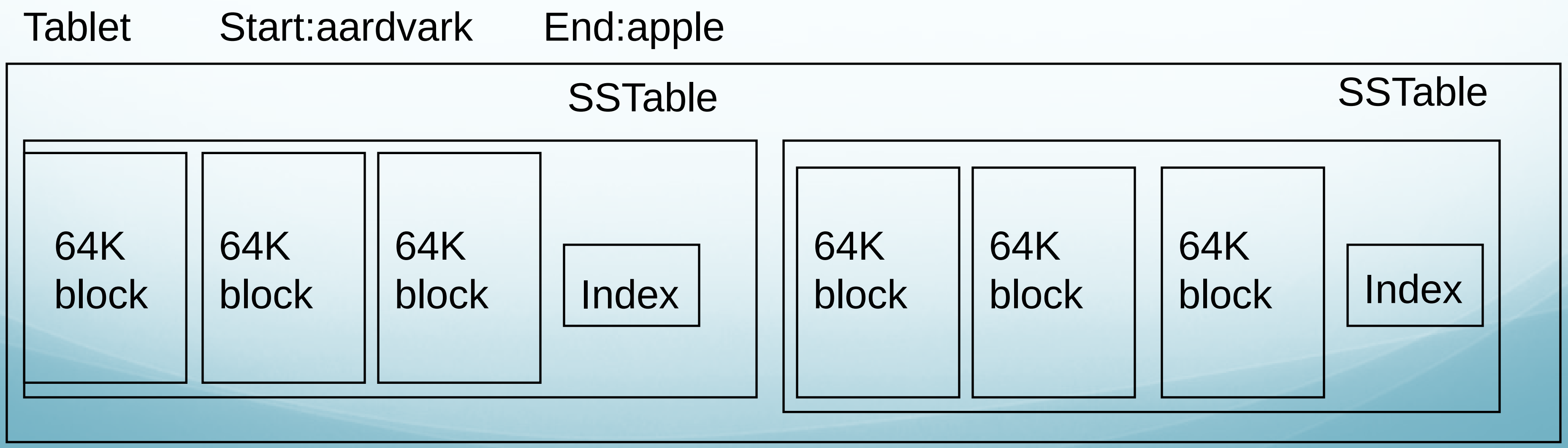
  - Approx 100 – 200 MB of data per tablet

- Approx 100 tablets per machine

  - Fast recovery

  - Load-balancing

- Built out of multiple SSTables



Tablet Start:aardvark End:apple

SSTable

64K block

64K block

64K block

Index

SSTable

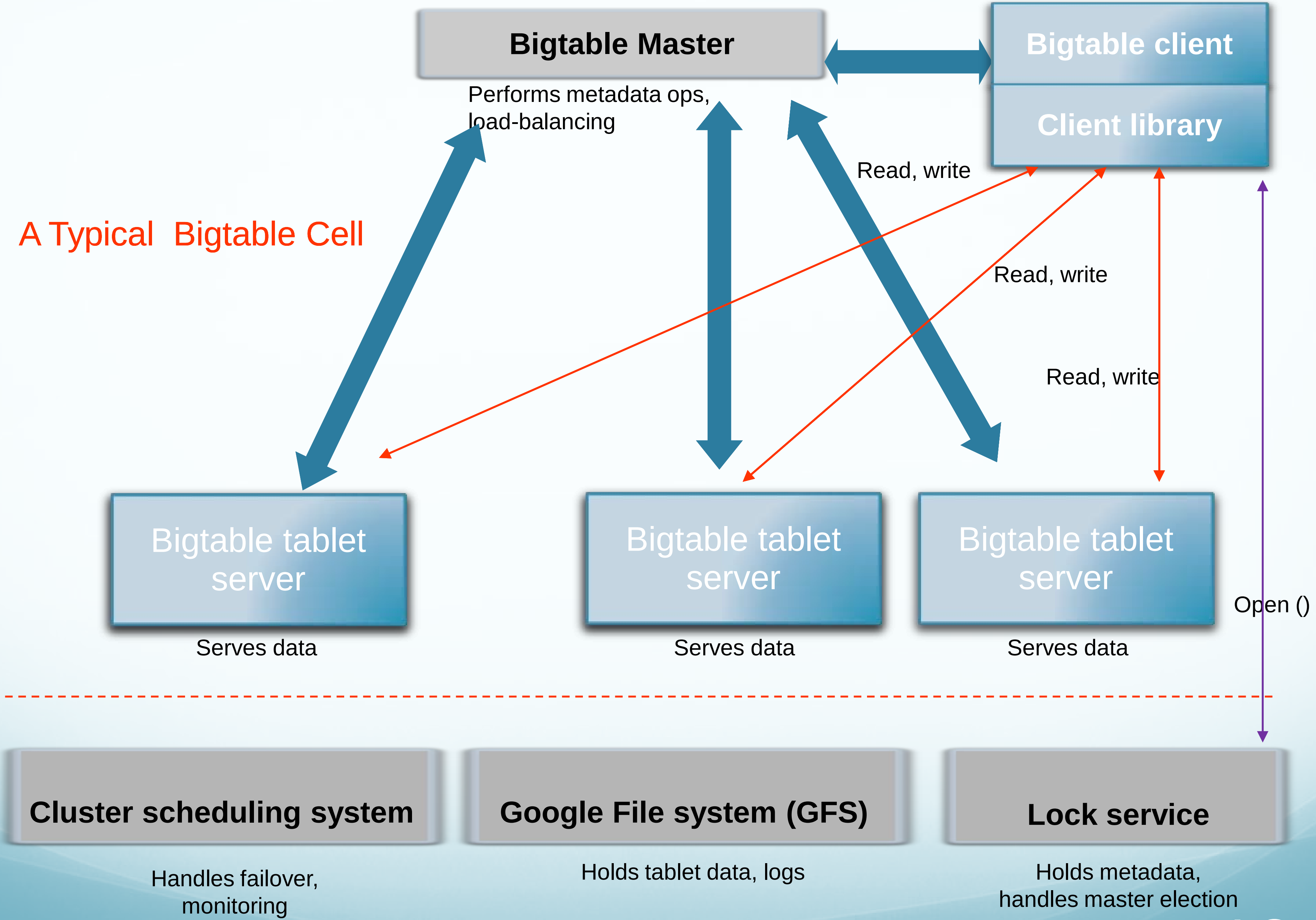
64K block

64K block

64K block

Index

A Typical Bigtable Cell



Bigtable Master

Performs metadata ops, load-balancing

Bigtable client

Client library

Read, write

Read, write

Read, write

Open ()

Bigtable tablet server

Serves data

Bigtable tablet server

Serves data

Bigtable tablet server

Serves data

Cluster scheduling system

Handles failover, monitoring

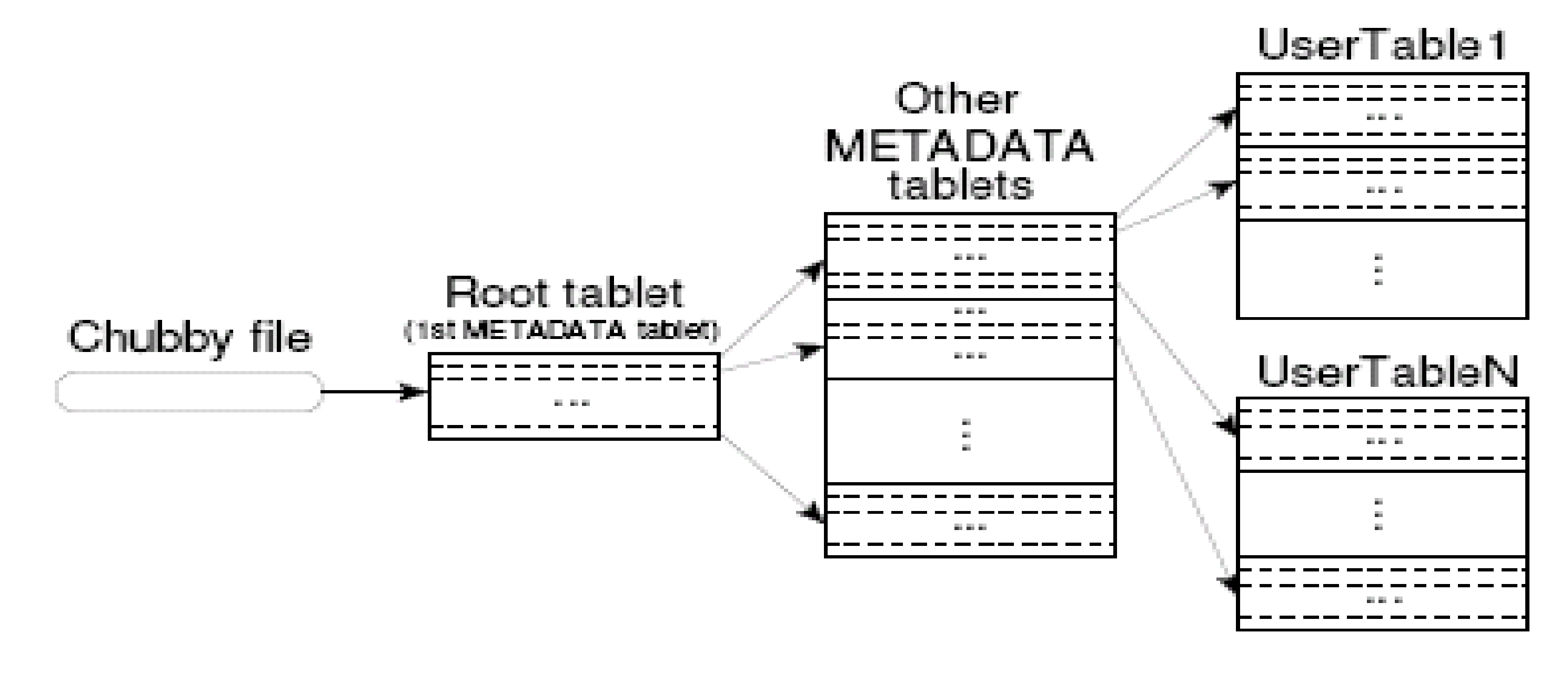
Google File system (GFS)

Holds tablet data, logs

Lock service

Holds metadata, handles master election

Finding a tablet



3-level look up scheme

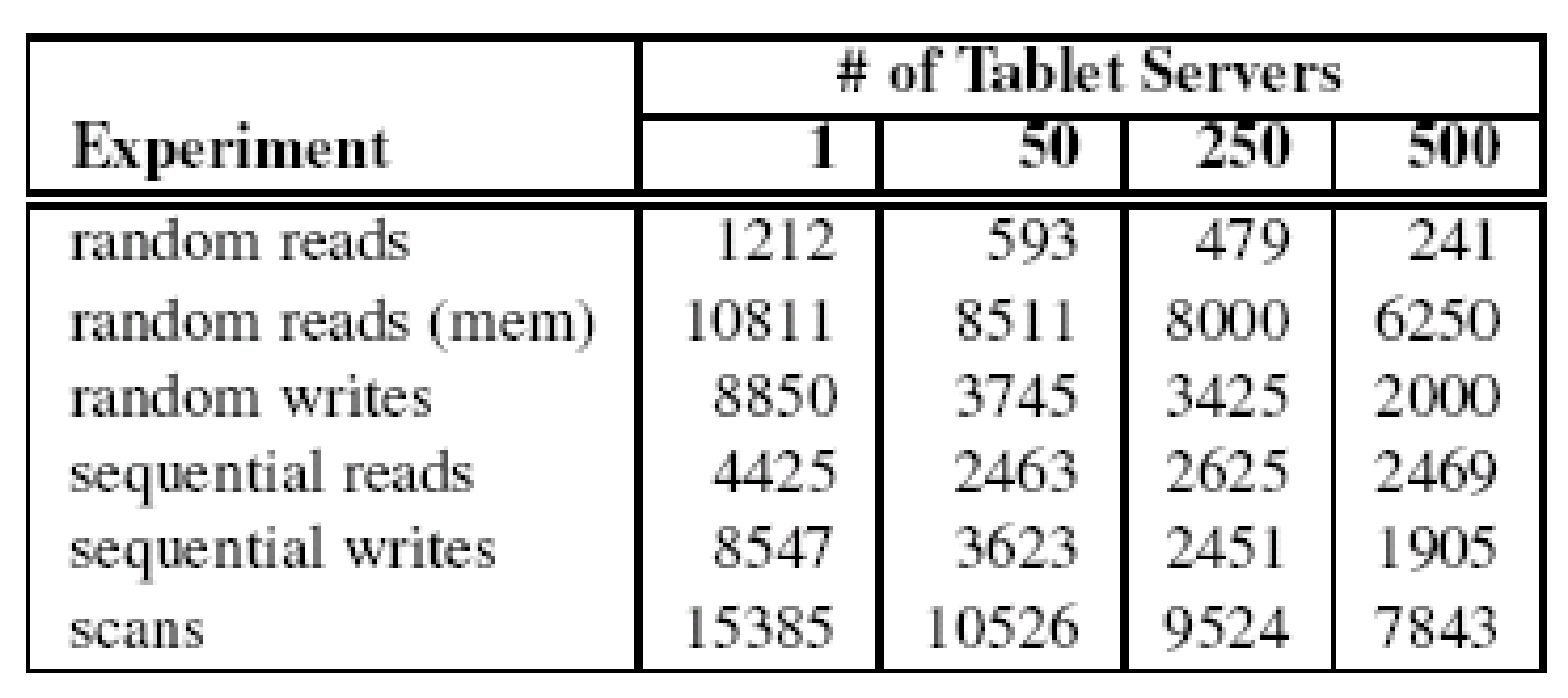
Compactions

- Minor compaction – convert the memtable into an SSTable
  - Reduce memory usage
  - Reduce log traffic on restart
- Merging compaction
  - Periodically executed in the background
  - Reduce number of SSTables
  - Good place to apply policy “keep only N versions”
- Major compaction
  - Merging compaction that results in only one SSTable
  - No deletion records, only live data
  - Reclaim resources.

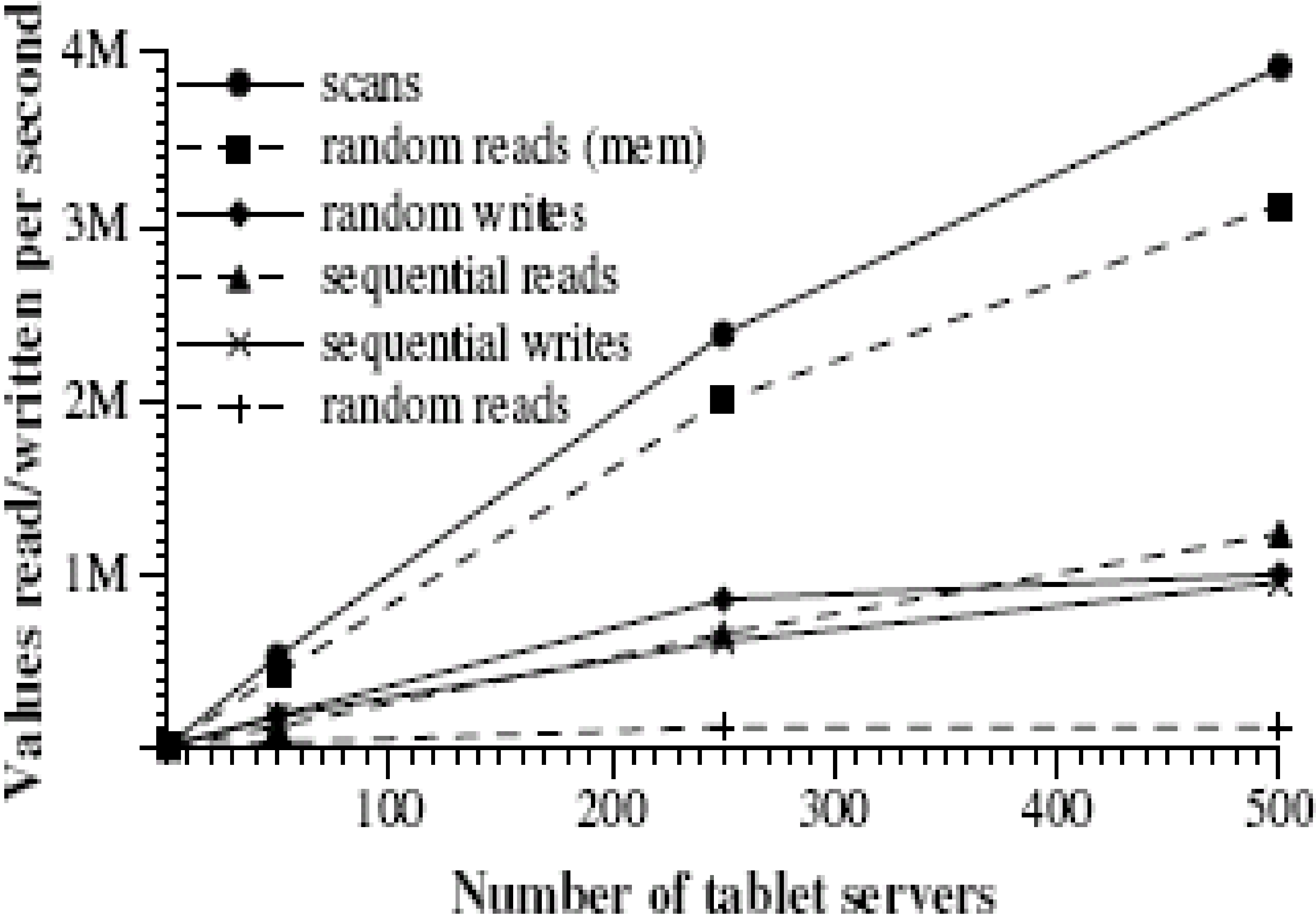
Locality Groups

- Group column families together into an SSTable
  - Avoid mingling data, ie page contents and page metadata
  - Can keep some groups all in memory
- Can compress locality groups
- Bloom Filters on locality groups – avoid searching SSTable

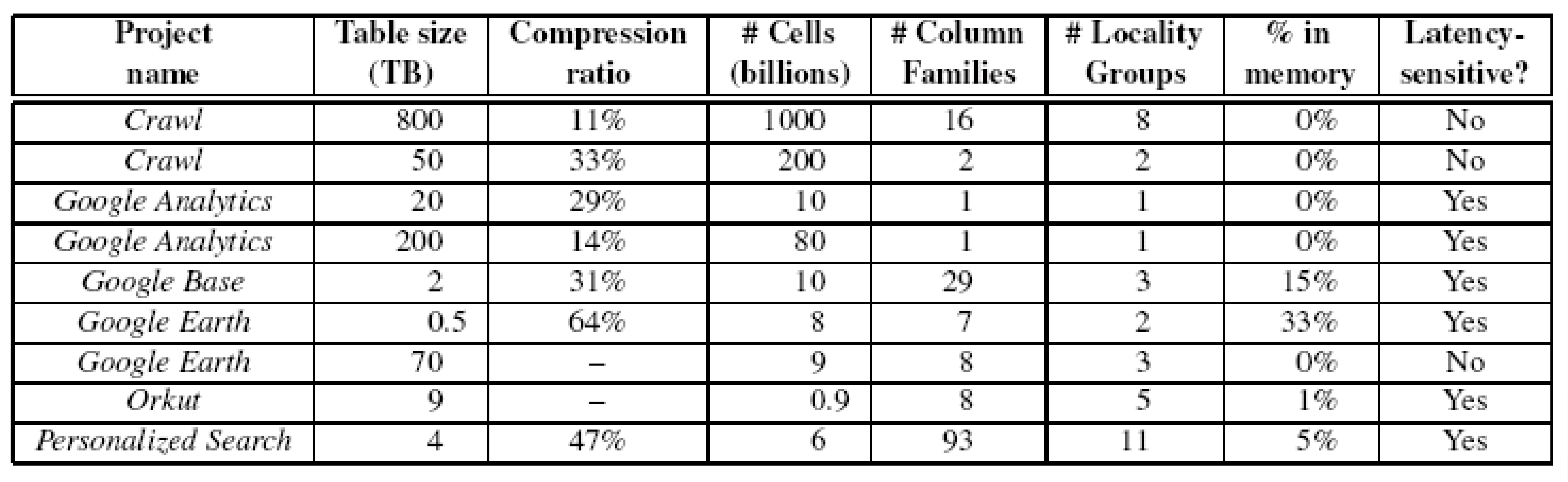
Microbenchmarks



Experiment	# of Tablet Servers			
	1	50	250	500
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843


Values read/written per second
4M
3M
2M
1M
scans
random reads (mem)
random writes
sequential reads
sequential writes
random reads
100
200
300
400
500
Number of tablet servers

Application at Google



Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes

Lessons learned

- Interesting point- only implement some of the requirements, since the last is probably not needed
- Many types of failure possible
- Big systems need proper systems-level monitoring
- Value simple designs

Thank You For Your Time!

QUESTIONS ?