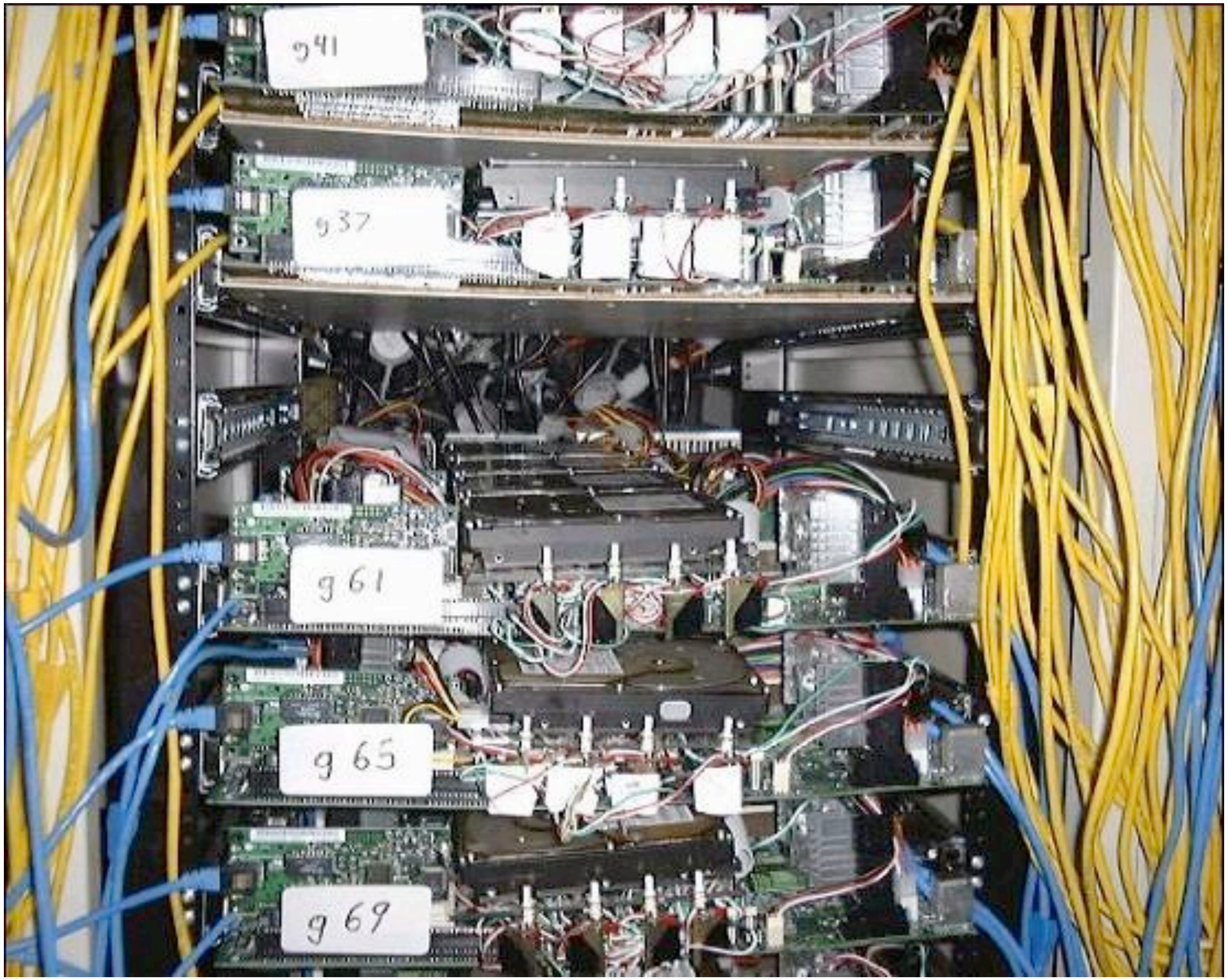
Lessons Learned While Building Infrastructure Software at Google

Jeff Dean
jeff@google.com

“Google” Circa 1997 (google.stanford.edu)

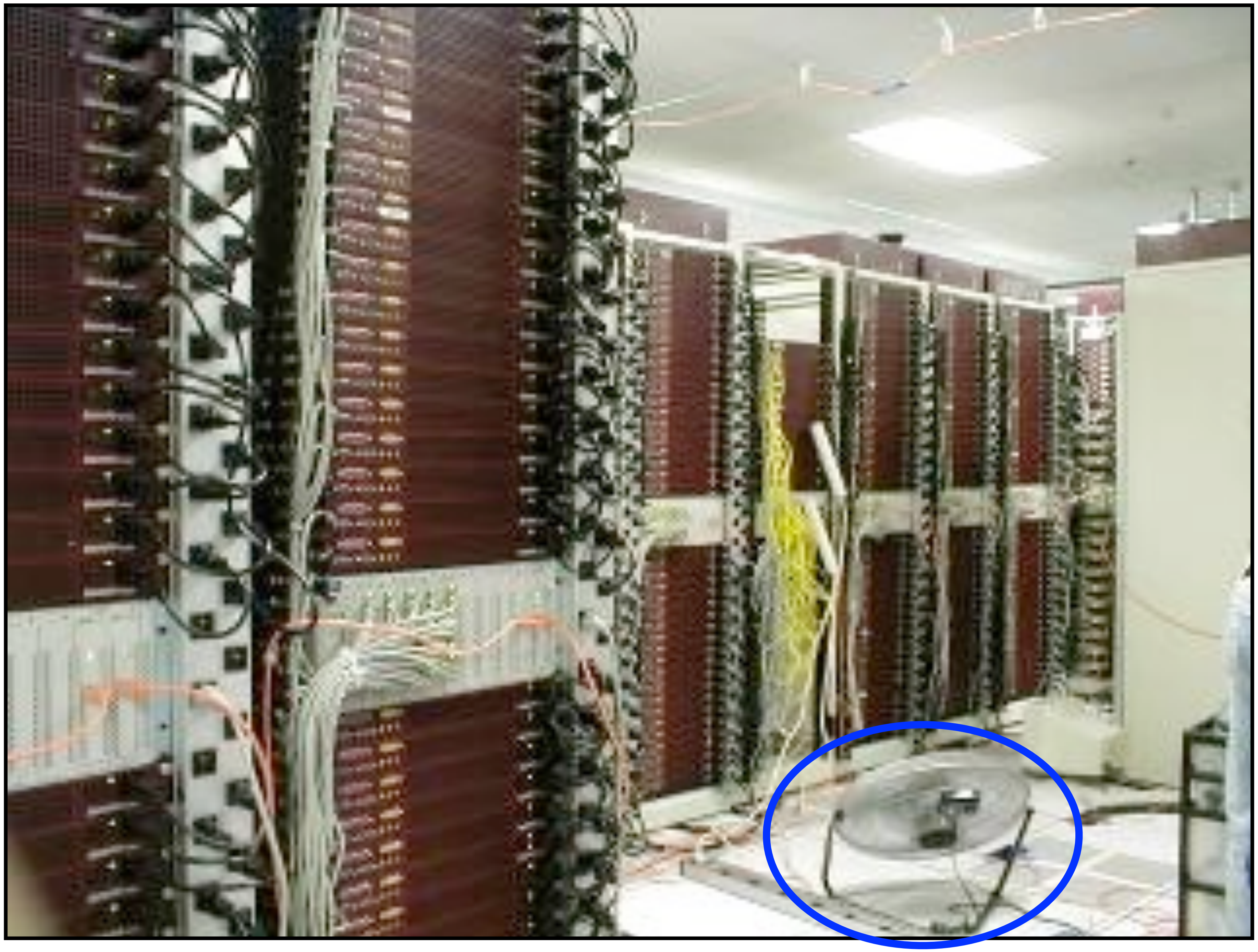
“Corkboards” (1999)



Google Data Center (2000)

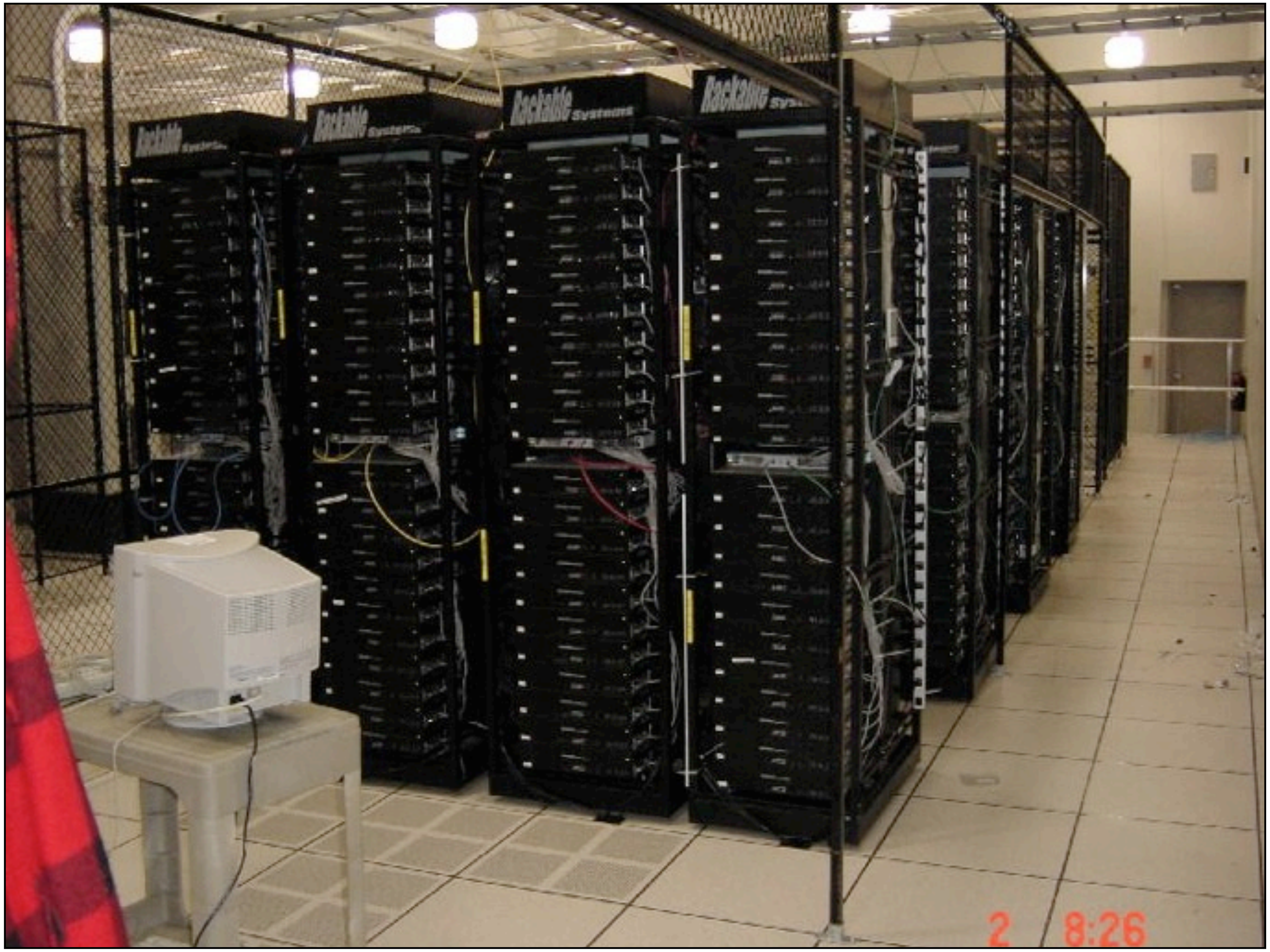
Google Data Center (2000)

Google Data Center (2000)



Google (new data center 2001)

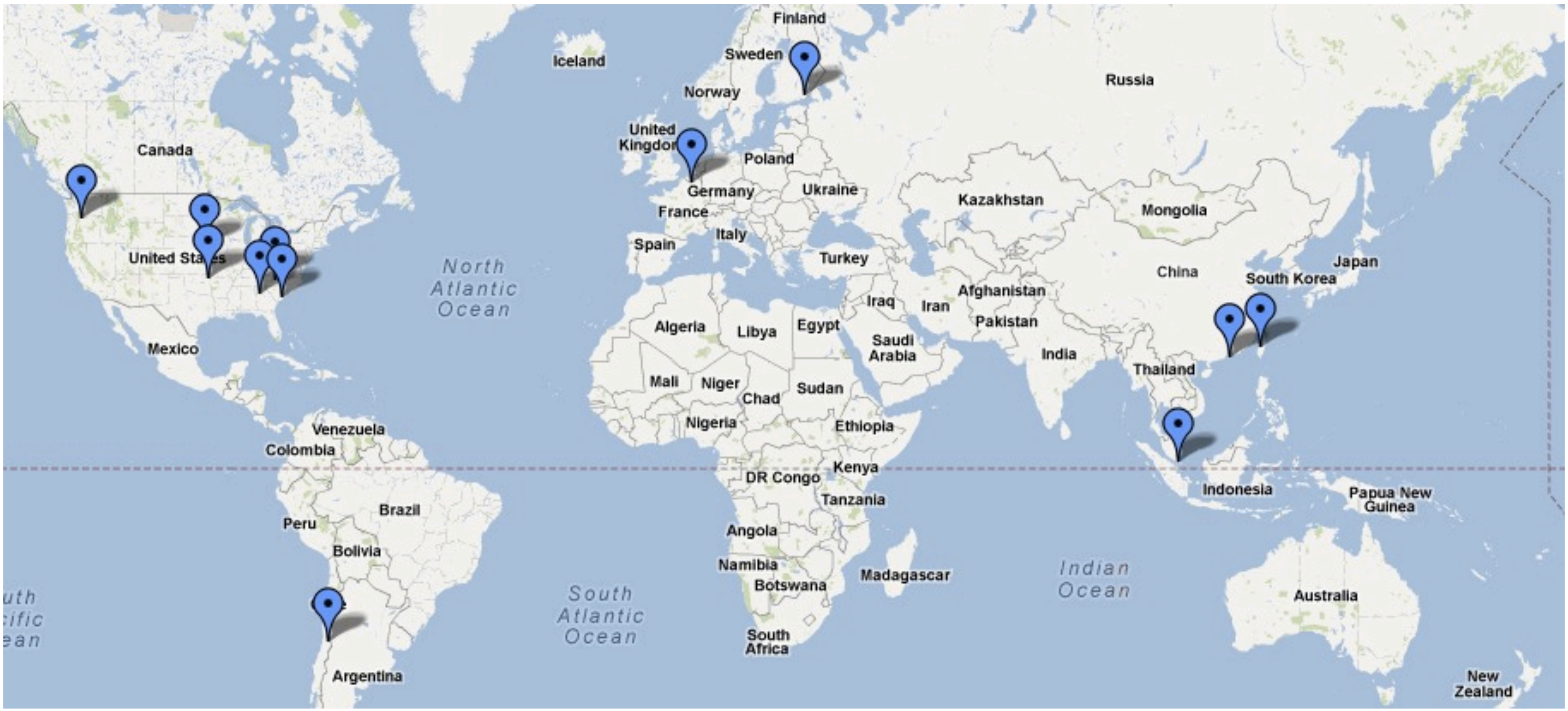
Google Data Center (3 days later)



Google

Google's Computational Environment Today

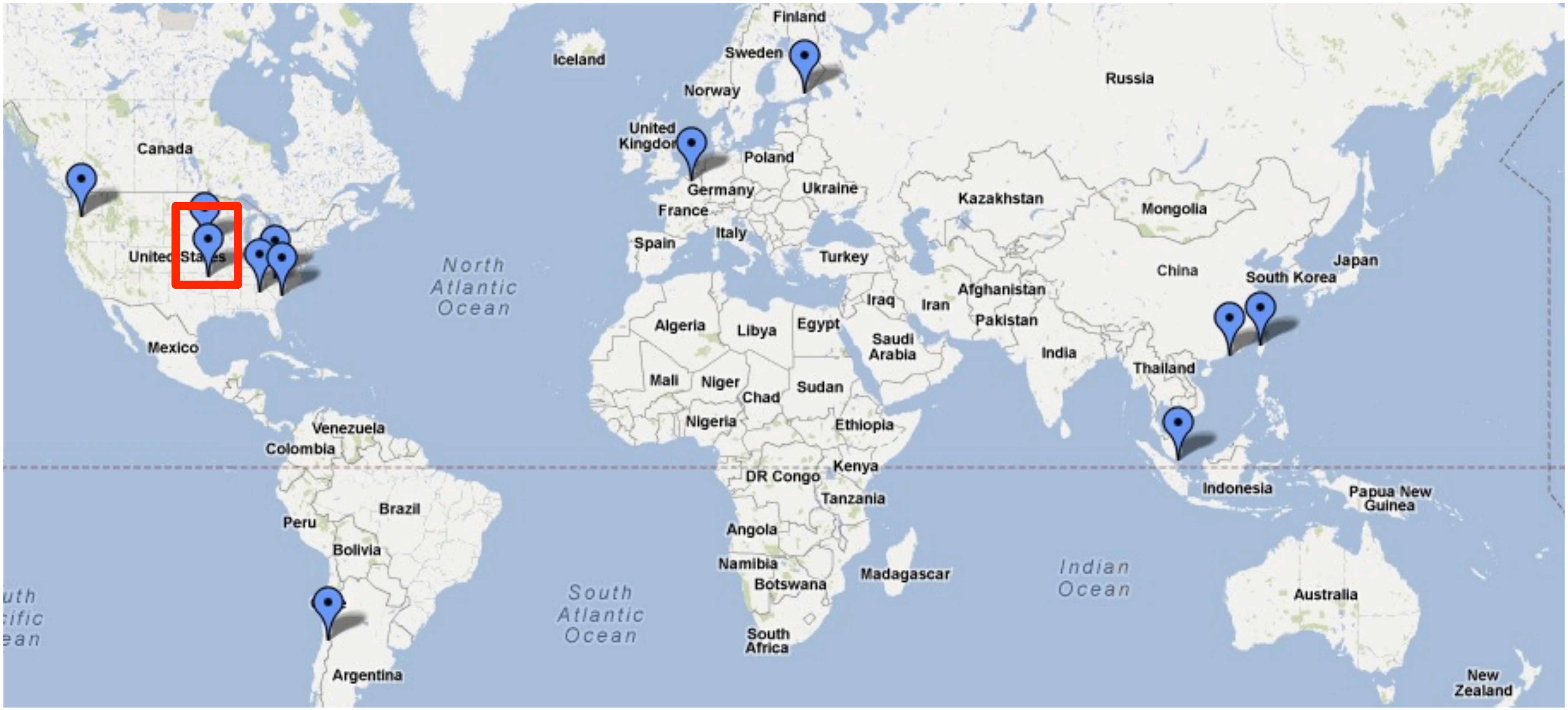
- Many datacenters around the world



Google

Google's Computational Environment Today

- Many datacenters around the world

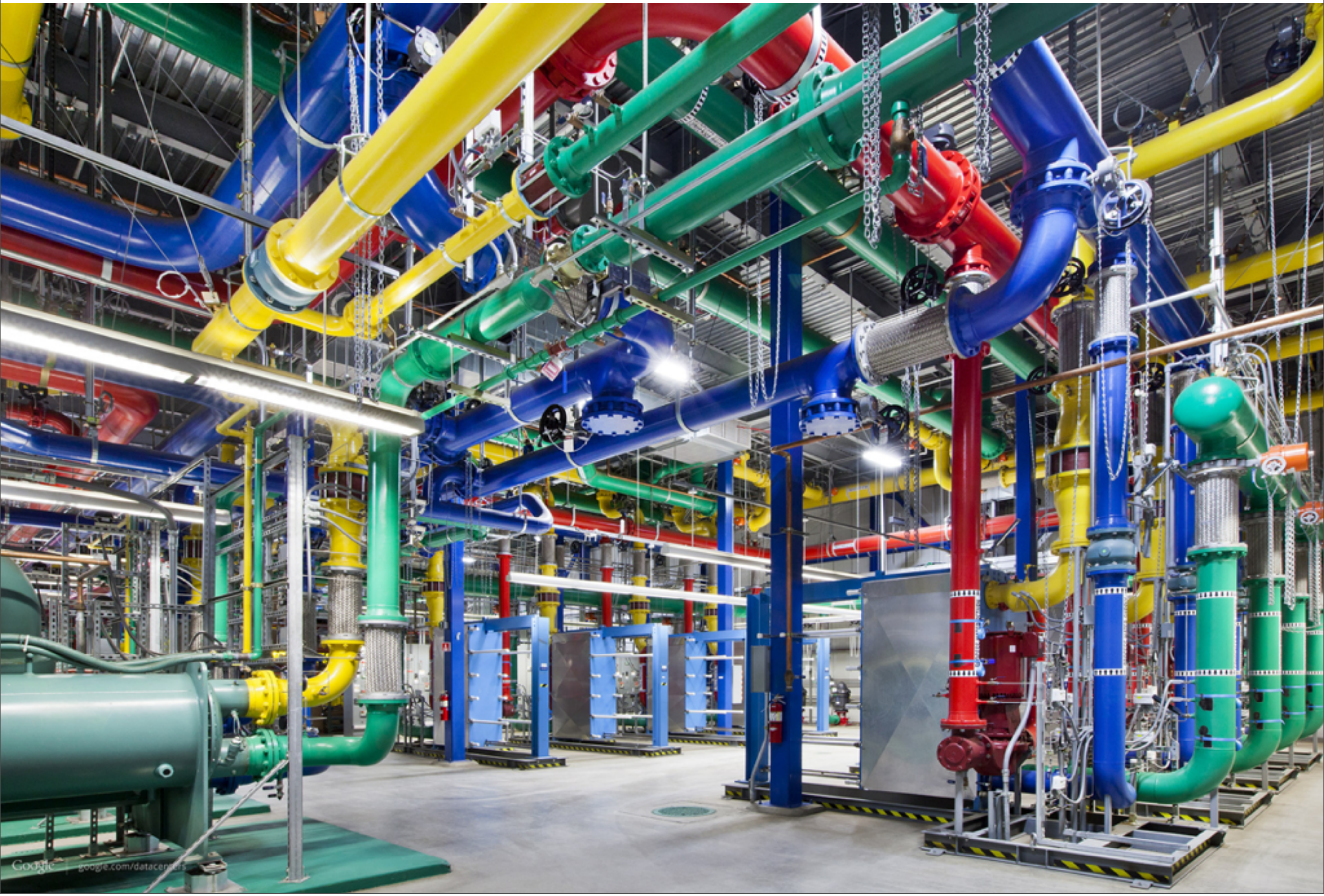


Google

Zooming In…

Lots of machines...

Cool...



Low-Level Systems Software Desires

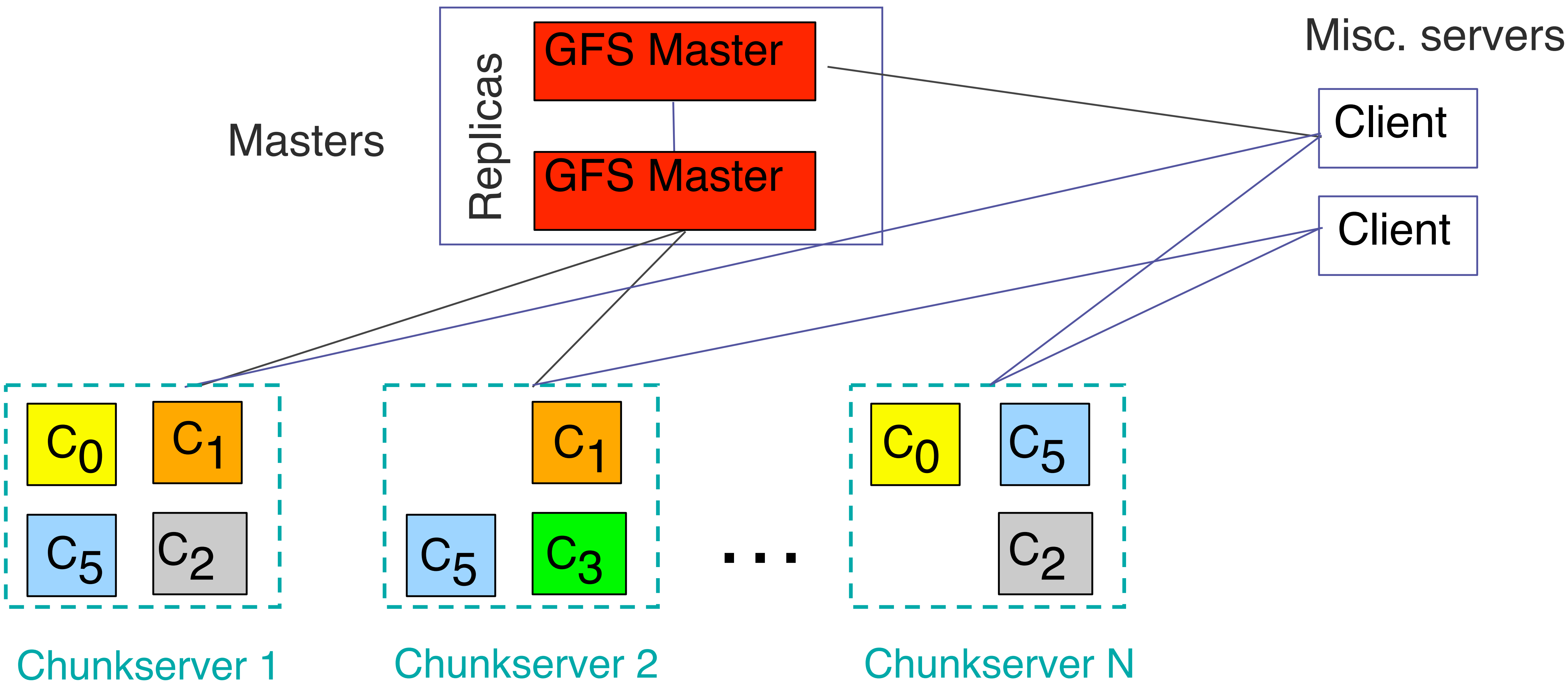
- If you have lots of machines, you want to:

- Store data persistently
  - w/ high availability
  - high read and write bandwidth

- Run large-scale computations reliably
  - without having to deal with machine failures

- GFS, MapReduce, BigTable, Spanner, ...

Google File System (GFS) Design



- Master manages metadata
- Data transfers are directly between clients/chunkservers
- Files broken into chunks (typically 64 MB)
- Chunks replicated across multiple machines (usually 3)

GFS Motivation and Lessons

- Indexing system clearly needed a large-scale distributed file system
  - wanted to treat whole cluster as single file system
- Developed by subset of same people working on indexing system
- Identified minimal set of features needed
  - *e.g.* Not POSIX compliant
  - actual data was distributed, but kept metadata centralized
    - Colossus: Follow-on system developed many years later distributed the metadata
- Lesson: Don't solve everything all at once

Google

MapReduce History

- 2003: Sanjay Ghemawat and I were working on rewriting indexing system:
  - starts with raw page contents on disk
  - many phases:
    - (near) duplicate elimination, anchor text extraction, language identification, index shard generation, etc.
  - end result is data structures for index and doc serving

- Each phase was hand written parallel computation:
  - hand parallelized
  - hand-written checkpointing code for fault-tolerance

MapReduce

- A simple programming model that applies to many large-scale computing problems
  - allowed us to express all phases of our indexing system
  - since used across broad range of computer science areas, plus other scientific fields
  - Hadoop open-source implementation seeing significant usage

- Hide messy details in MapReduce runtime library:
  - automatic parallelization
  - load balancing
  - network and disk transfer optimizations
  - handling of machine failures
  - robustness
  - **improvements to core library benefit all users of library!**

Google

Typical problem solved by MapReduce

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, or transform
- Write the results

Outline stays the same,
User writes Map and Reduce functions to fit the problem

Google

MapReduce Motivation and Lessons

- Developed by two people that were also doing the indexing system rewrite
  - squinted at various phases with an eye towards coming up with common abstraction
- Initial version developed quickly
  - proved initial API utility with very simple implementation
  - rewrote much of implementation 6 months later to add lots of the performance wrinkles/tricks that appeared in original paper
- Lesson: Very close ties with initial users of system make things happen faster
  - in this case, we were both building MapReduce and using it simultaneously

Google

BigTable: Motivation

- Lots of (semi-)structured data at Google
  - URLs: Contents, crawl metadata, links, anchors, pagerank, …
  - Per-user data: User preferences, recent queries, …
  - Geographic locations: Physical entities, roads, satellite image data, user annotations, …
- Scale is large
- Want to be able to grow and shrink resources devoted to system as needed

BigTable: Basic Data Model

- Distributed multi-dimensional sparse map

  (row, column, timestamp) → cell contents

Columns

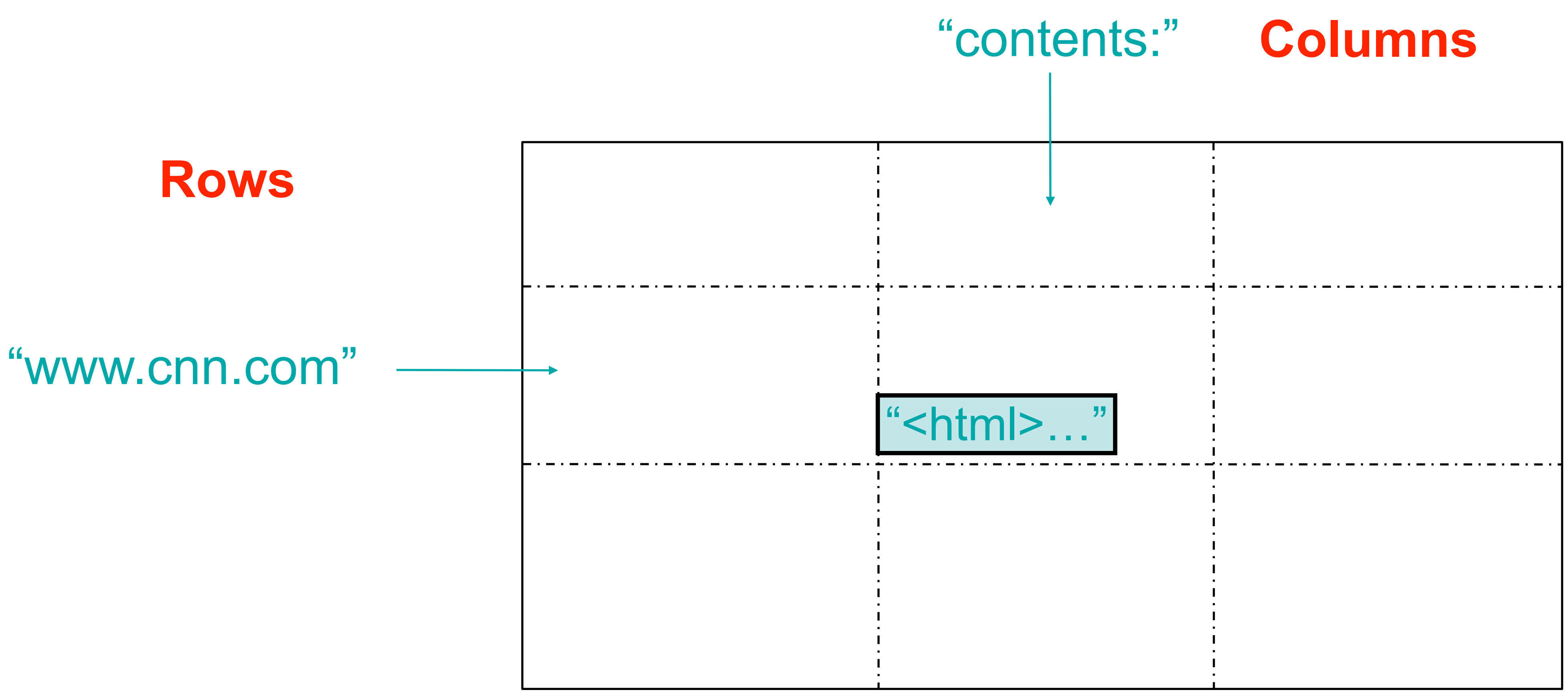
Rows

- Rows are ordered lexicographically
- Good match for most of our applications

BigTable: Basic Data Model

- Distributed multi-dimensional sparse map

  (row, column, timestamp) → cell contents

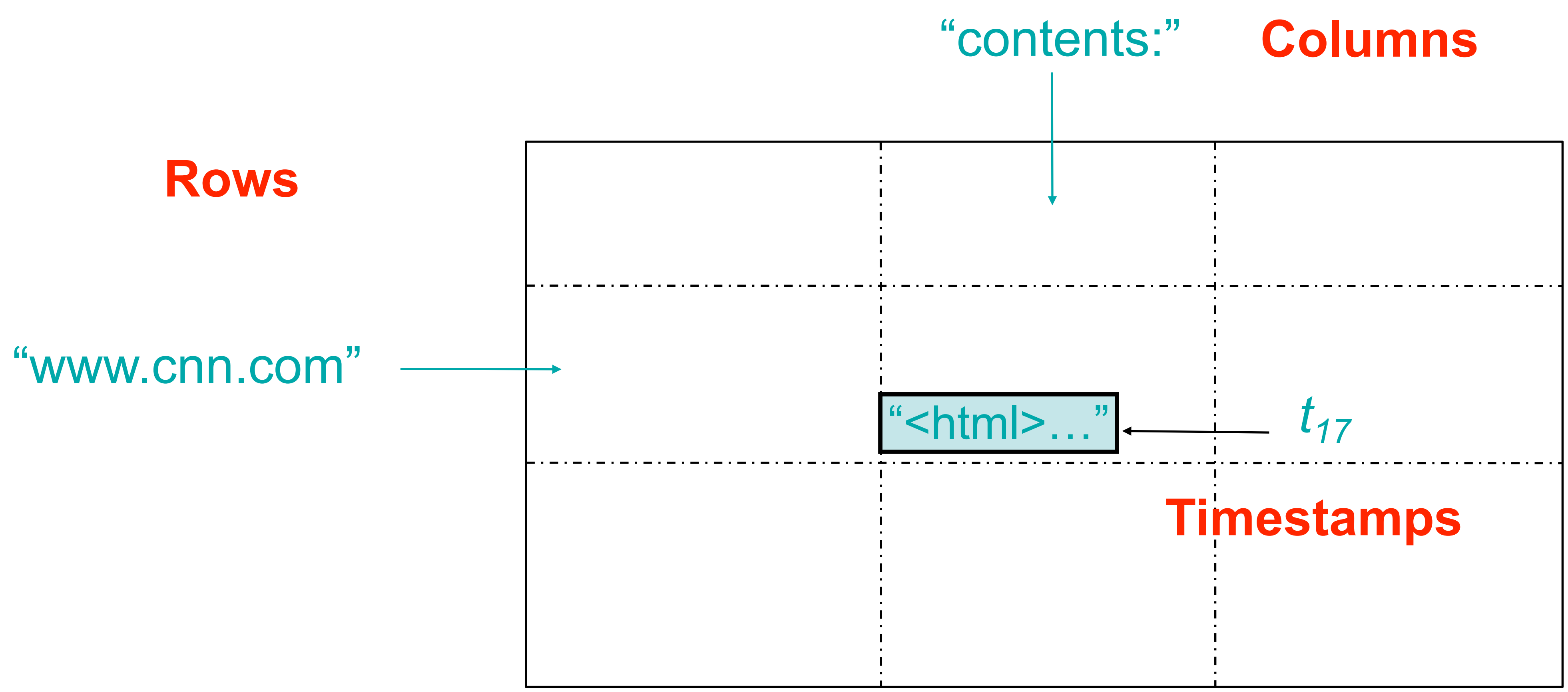


- Rows are ordered lexicographically
- Good match for most of our applications

BigTable: Basic Data Model

- Distributed multi-dimensional sparse map

  (row, column, timestamp) → cell contents



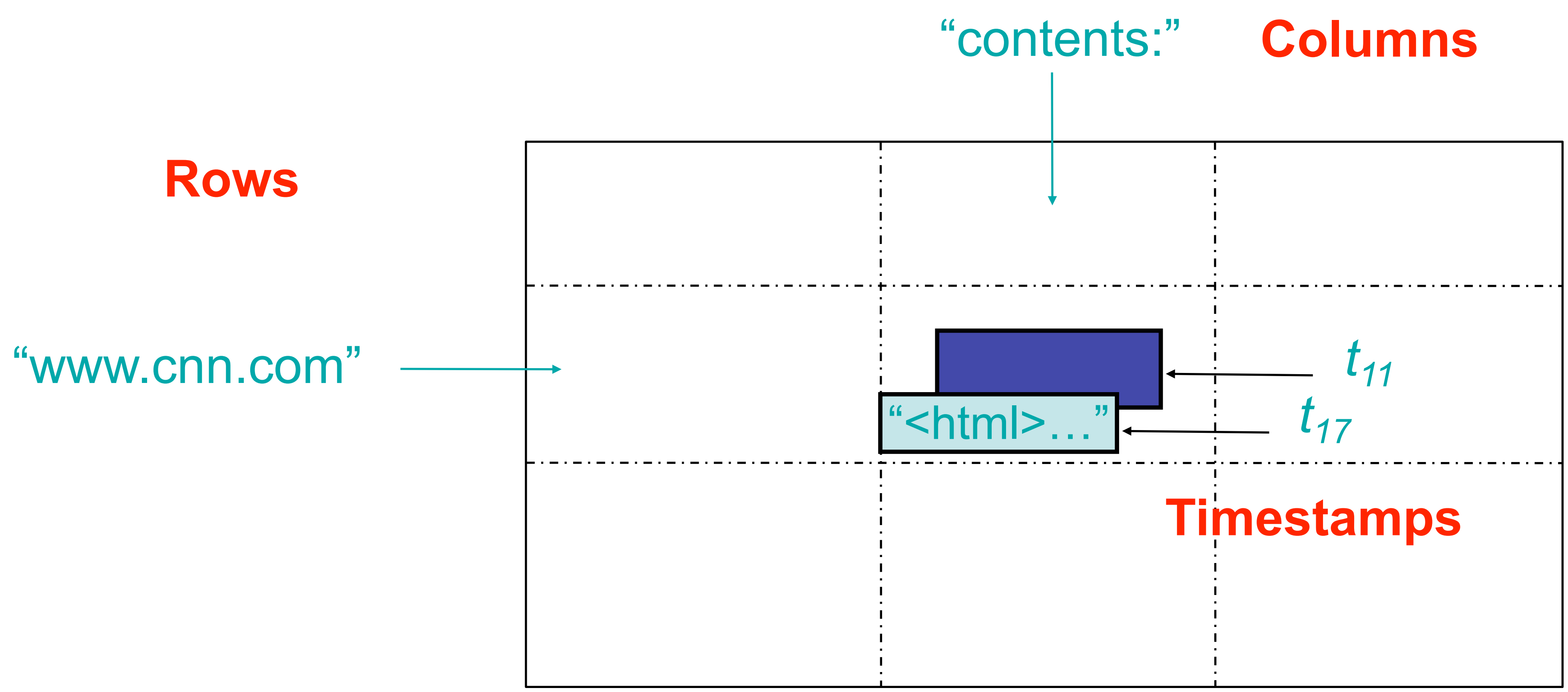
- Rows are ordered lexicographically
- Good match for most of our applications

Google

BigTable: Basic Data Model

- Distributed multi-dimensional sparse map

  (row, column, timestamp) → cell contents

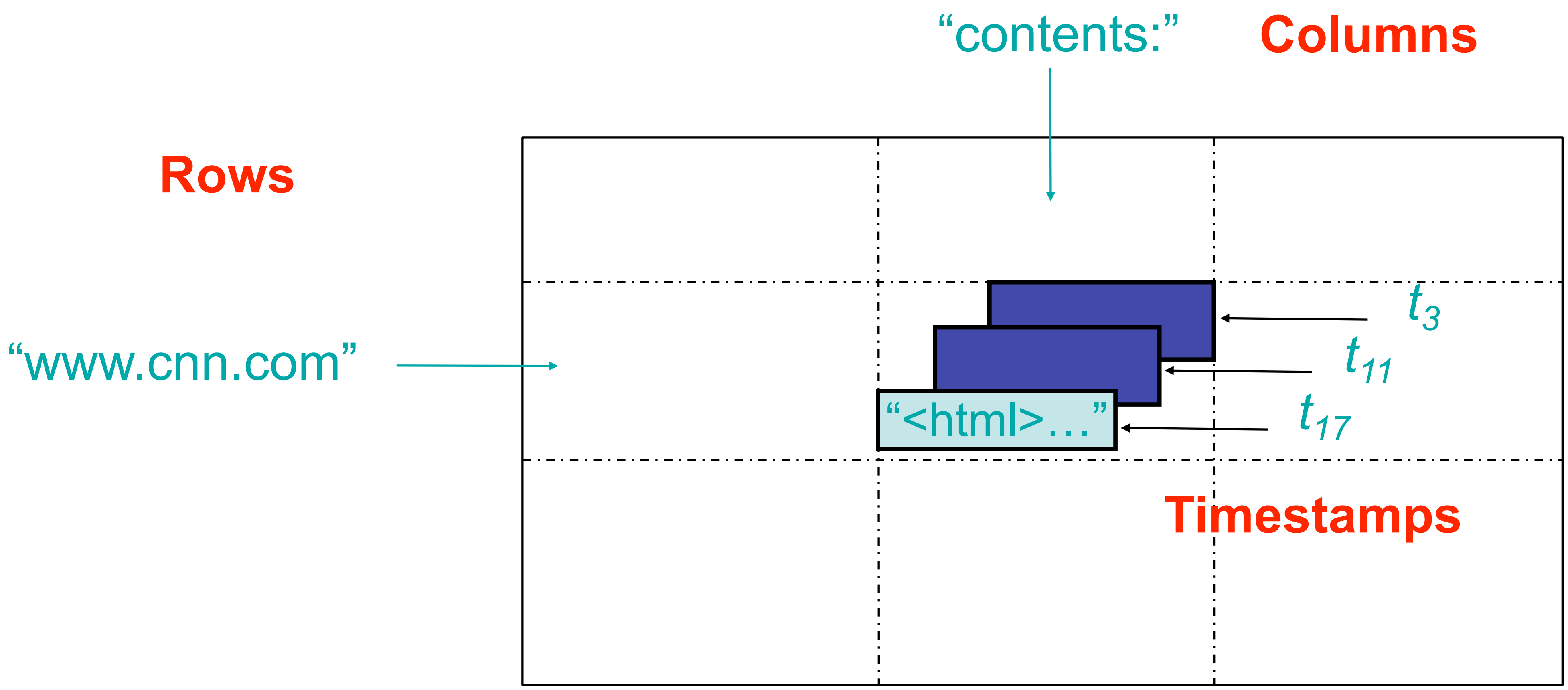


- Rows are ordered lexicographically
- Good match for most of our applications

BigTable: Basic Data Model

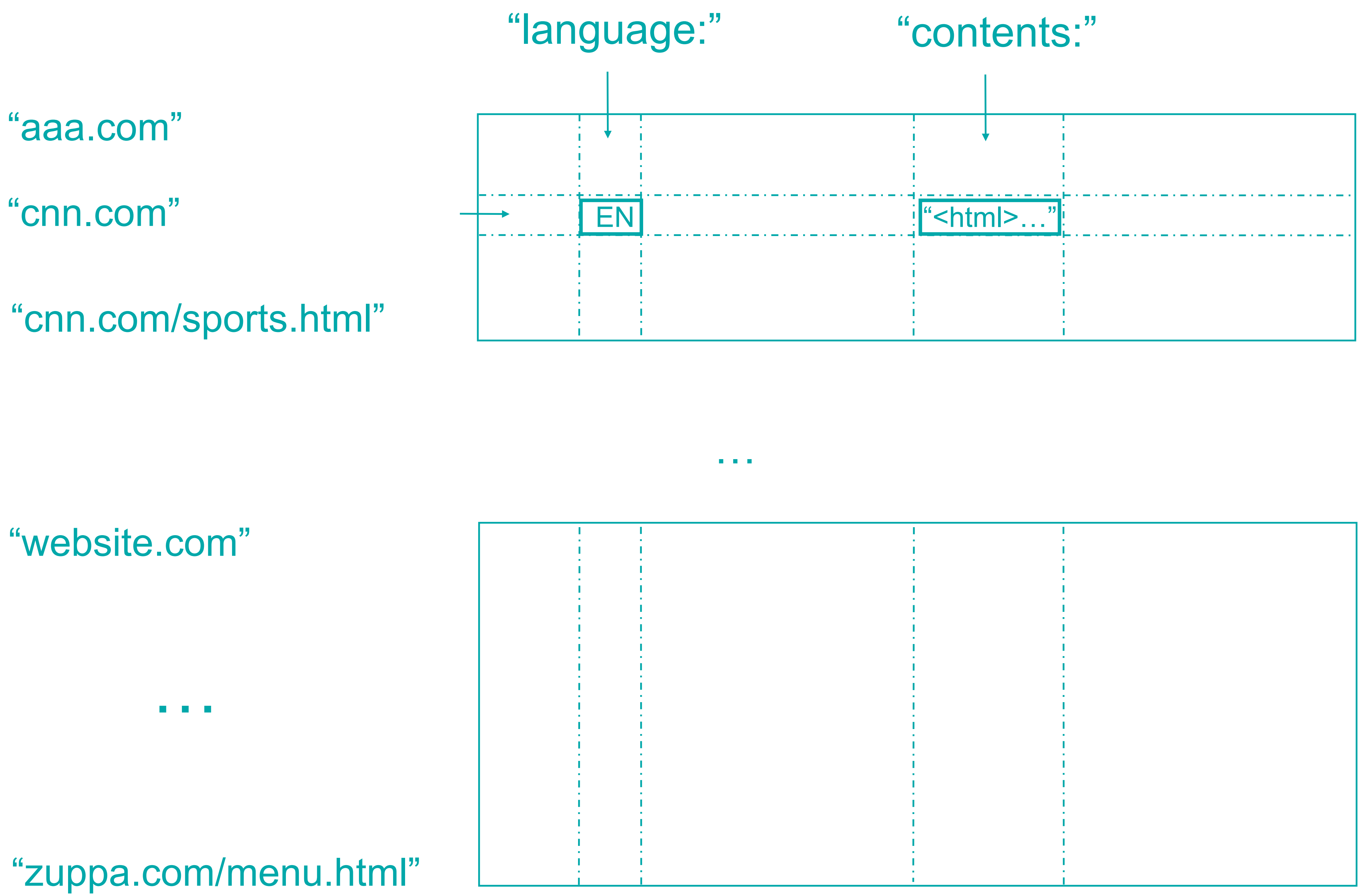
- Distributed multi-dimensional sparse map

  (row, column, timestamp) → cell contents

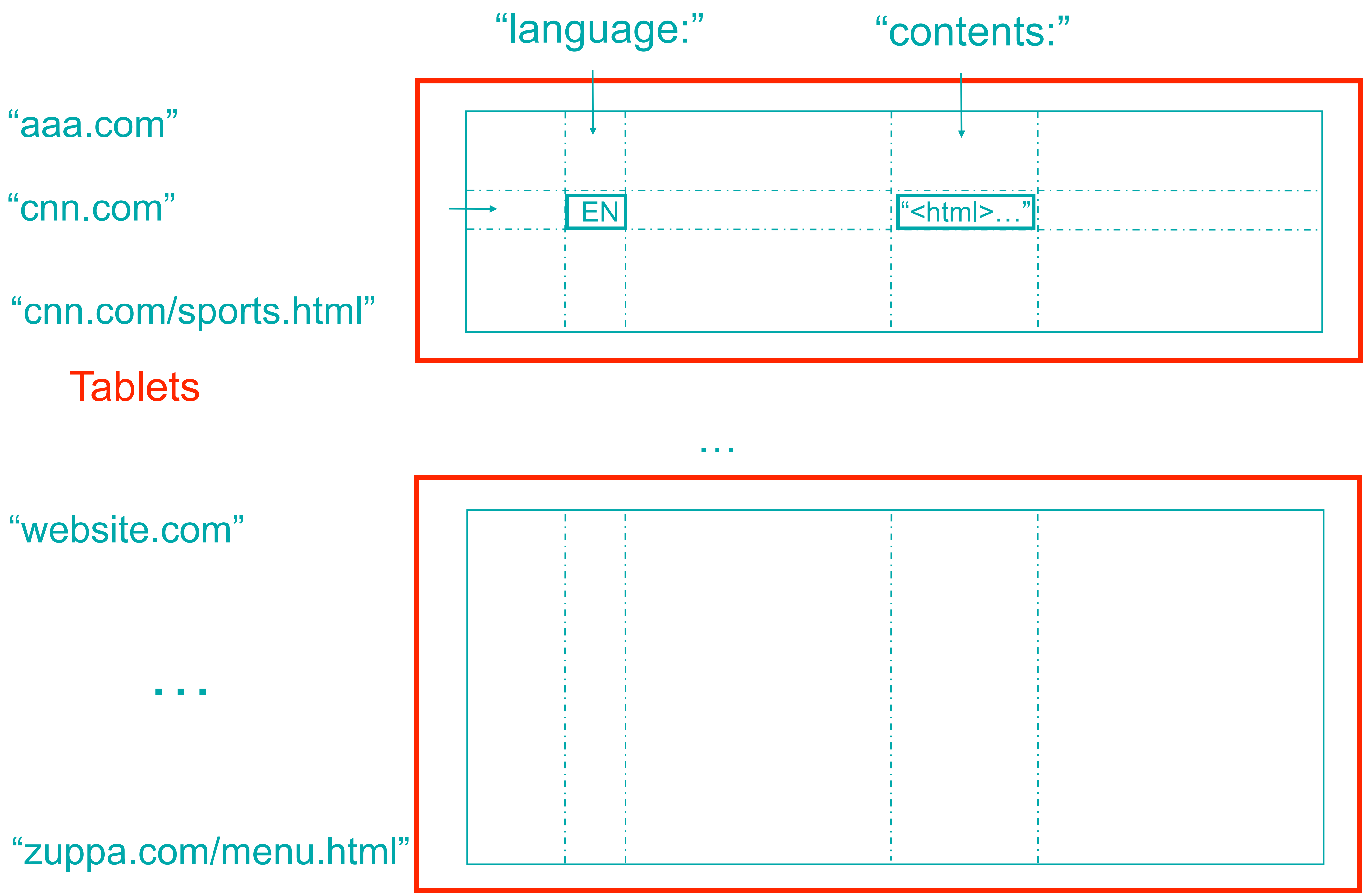


- Rows are ordered lexicographically
- Good match for most of our applications

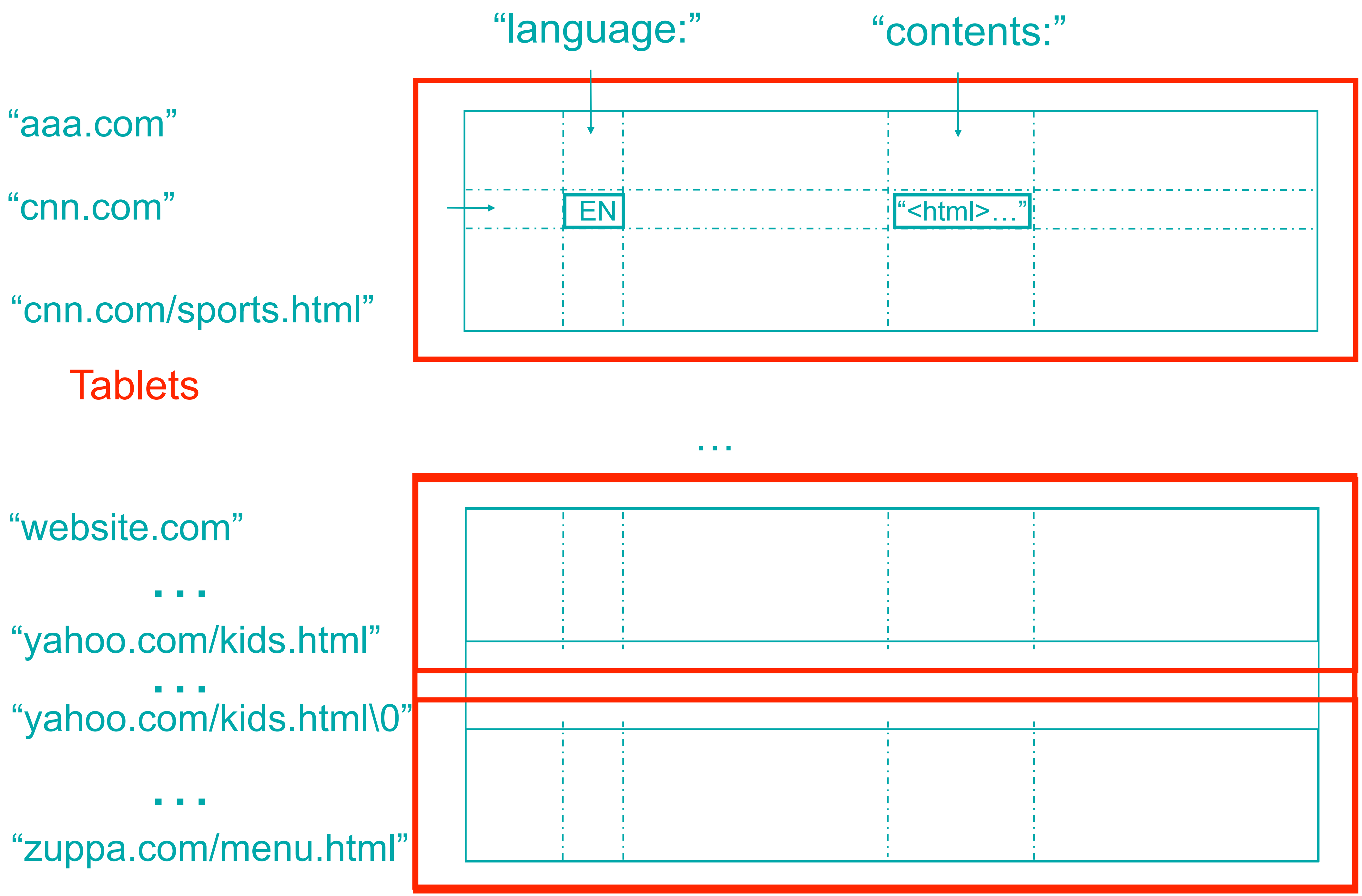
Tablets & Splitting



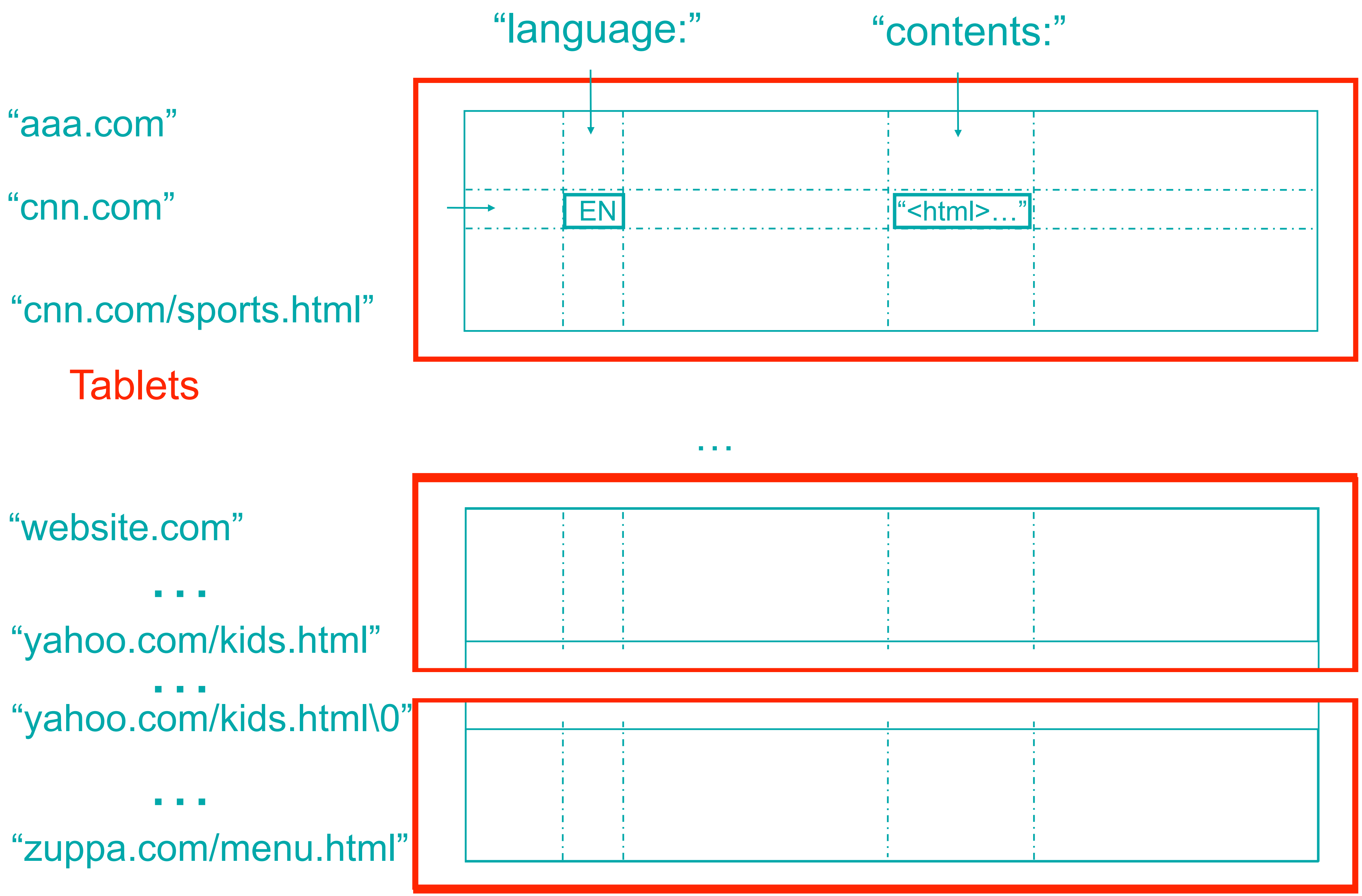
Tablets & Splitting



Tablets & Splitting



Tablets & Splitting



Google

BigTable System Structure

Bigtable Cell

Bigtable master

Bigtable tablet server   Bigtable tablet server   …   Bigtable tablet server

BigTable System Structure

Bigtable Cell

Bigtable master

performs metadata ops +
load balancing

Bigtable tablet server   Bigtable tablet server   …   Bigtable tablet server

Google

BigTable System Structure

Bigtable Cell

Bigtable master

performs metadata ops + load balancing

Bigtable tablet server | Bigtable tablet server | … | Bigtable tablet server

serves data | serves data | serves data

Google

BigTable System Structure

Bigtable Cell

Bigtable master

performs metadata ops + load balancing

Bigtable tablet server	Bigtable tablet server	…	Bigtable tablet server
serves data	serves data		serves data

Cluster scheduling system

Cluster file system

Lock service

BigTable System Structure

Bigtable Cell

Bigtable master

performs metadata ops + load balancing

Bigtable tablet server — serves data

Bigtable tablet server — serves data

…

Bigtable tablet server — serves data

Cluster scheduling system

schedules tasks onto machines

Cluster file system

Lock service

BigTable System Structure

Bigtable Cell

Bigtable master

performs metadata ops +
load balancing

Bigtable tablet server

serves data

Bigtable tablet server

serves data

…

Bigtable tablet server

serves data

Cluster scheduling system

schedules tasks onto machines

Cluster file system

holds tablet data, logs

Lock service

BigTable System Structure

Bigtable Cell

Bigtable master

performs metadata ops + load balancing

Bigtable tablet server	Bigtable tablet server	…	Bigtable tablet server
serves data	serves data		serves data

Cluster scheduling system	Cluster file system	Lock service
schedules tasks onto machines	holds tablet data, logs	holds metadata, handles master-election

Google

BigTable System Structure

Bigtable Cell

Bigtable client

Bigtable client library

Bigtable master

performs metadata ops + load balancing

Bigtable tablet server

serves data

Bigtable tablet server

serves data

…

Bigtable tablet server

serves data

Cluster scheduling system

schedules tasks onto machines

Cluster file system

holds tablet data, logs

Lock service

holds metadata, handles master-election

BigTable System Structure

Bigtable client

Bigtable client library

Bigtable Cell

Bigtable master

performs metadata ops + load balancing

Open()

Bigtable tablet server

serves data

Bigtable tablet server

serves data

…

Bigtable tablet server

serves data

Cluster scheduling system

schedules tasks onto machines

Cluster file system

holds tablet data, logs

Lock service

holds metadata, handles master-election

BigTable System Structure

Bigtable Cell

Bigtable client

Bigtable client library

Bigtable master

performs metadata ops + load balancing

read/write

Open()

Bigtable tablet server

serves data

Bigtable tablet server

serves data

…

Bigtable tablet server

serves data

Cluster scheduling system

schedules tasks onto machines

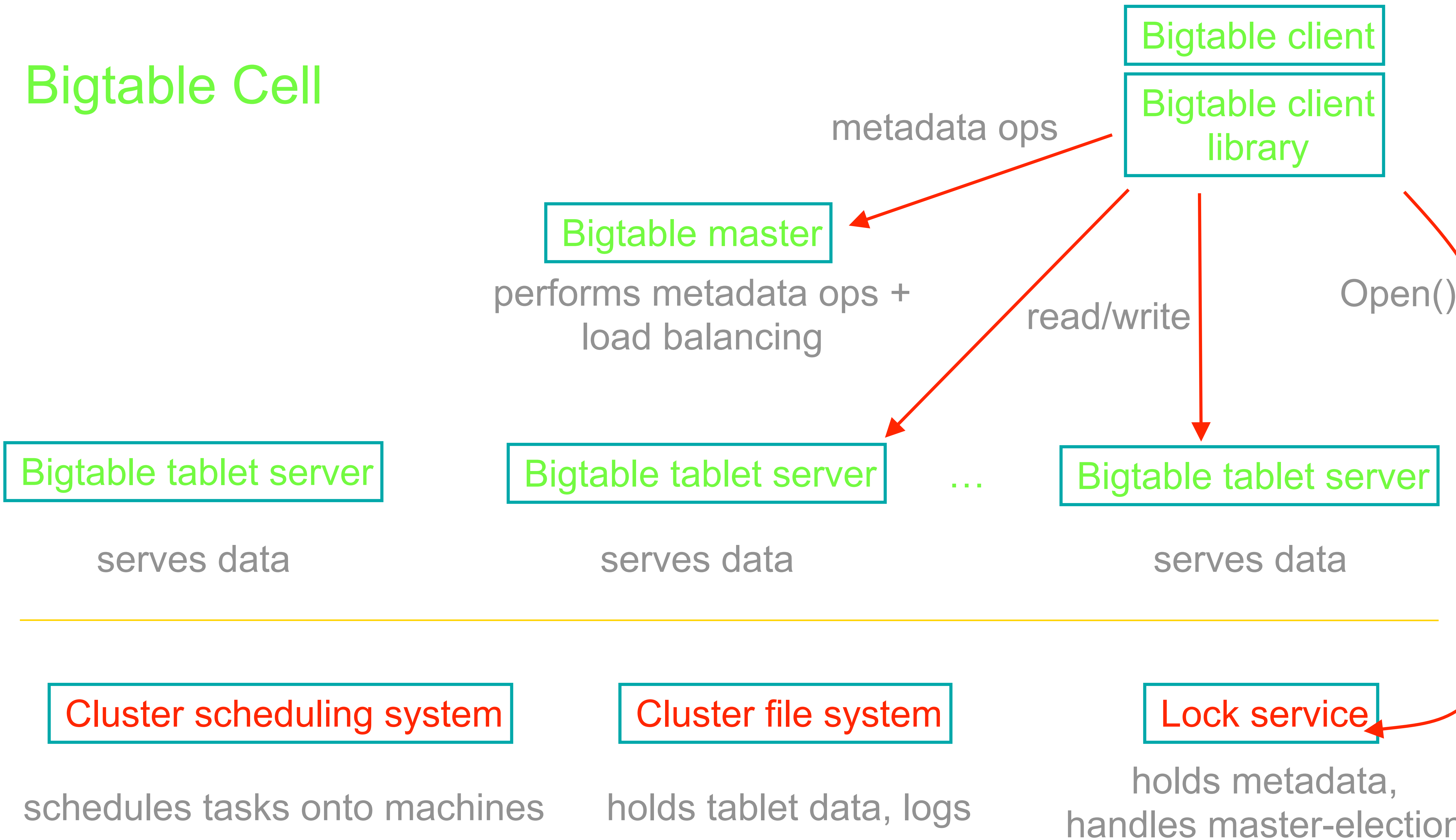
Cluster file system

holds tablet data, logs

Lock service

holds metadata, handles master-election

BigTable System Structure



BigTable Status

- Production use for 100s of projects:
  - Crawling/indexing pipeline, Google Maps/Google Earth/Streetview, Search History, Google Print, Google+, Blogger, ...
- Currently 500+ BigTable clusters
- Largest cluster:
  - 100s PB data; sustained: 30M ops/sec; 100+ GB/s I/O
- Many asynchronous processes updating different pieces of information
  - no distributed transactions, no cross-row joins
  - initial design was just in a single cluster
  - follow-on work added eventual consistency across many geographically distributed BigTable instances

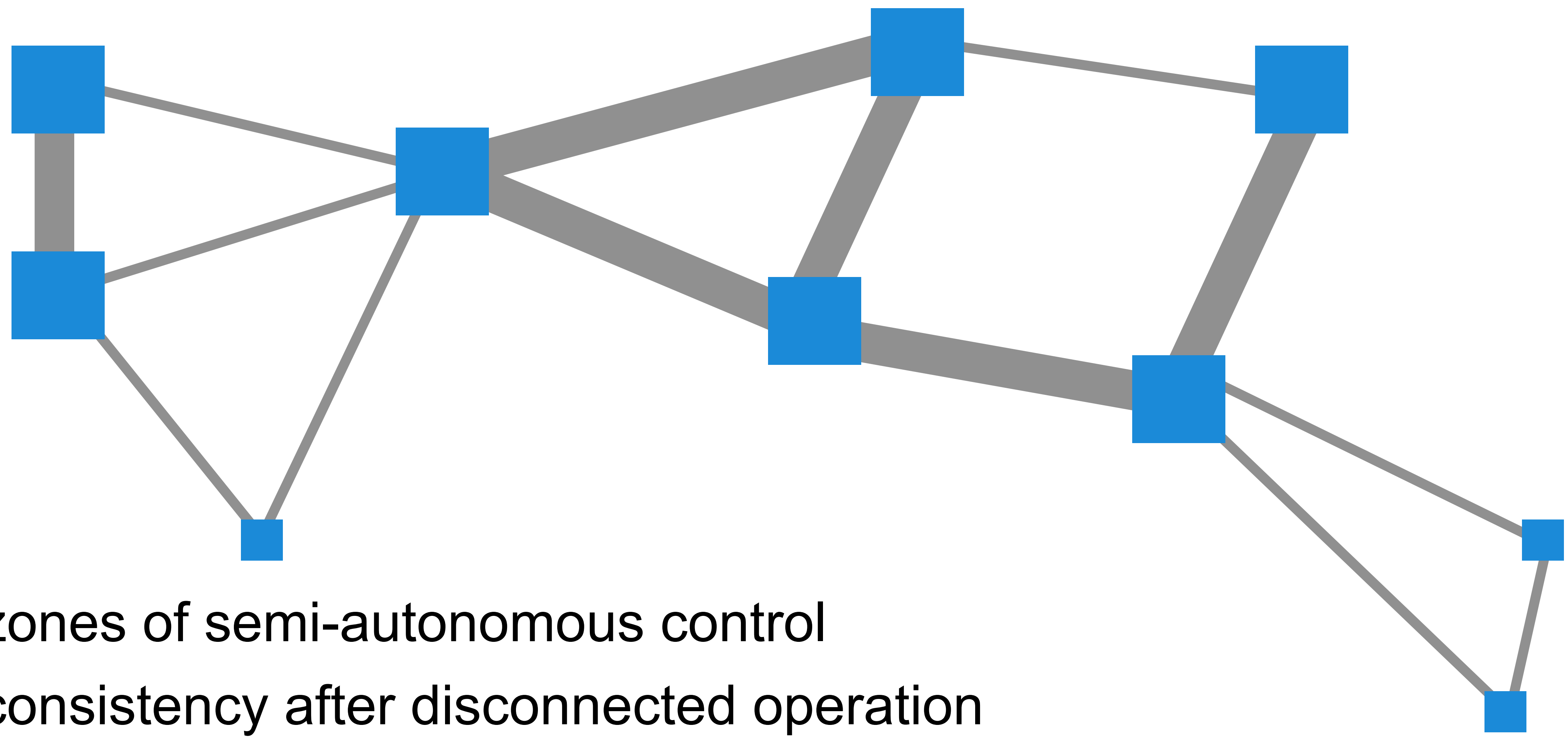
Google

Spanner

- Storage & computation system that runs across many datacenters
  - single global namespace
    - names are independent of location(s) of data
    - fine-grained replication configurations
  - support mix of strong and weak consistency across datacenters
    - Strong consistency implemented with Paxos across tablet replicas
    - Full support for distributed transactions across directories/machines
  - much more automated operation
    - automatically changes replication based on constraints and usage patterns
    - automated allocation of resources across entire fleet of machines

Design Goals for Spanner

- Future scale: ~10^5 to 10^7 machines, ~10^{13} directories, ~10^{18} bytes of storage, spread at 100s to 1000s of locations around the world



  - zones of semi-autonomous control
  - consistency after disconnected operation
  - users specify high-level desires:

    “99%ile latency for accessing this data should be <50ms”

    “Store this data on at least 2 disks in EU, 2 in U.S. & 1 in Asia”

Spanner Lessons

- Several variations of eventual client API
- Started to develop with many possible customers in mind, but no particular customer we were working closely with

- Eventually we worked closely with Google ads system as initial customer
  - first real customer was very demanding (real $$): good and bad

- Different API than BigTable
  - Harder to move users with existing heavy BigTable usage

Designing & Building Infrastructure

Identify common problems, and build software systems to address them in a general way

- Important to not try to be all things to all people
  - Clients might be demanding 8 different things
  - Doing 6 of them is easy
  - …handling 7 of them requires real thought
  - …dealing with all 8 usually results in a worse system
    - more complex, compromises other clients in trying to satisfy everyone

Designing & Building Infrastructure (cont)

Don't build infrastructure just for its own sake:

- Identify common needs and address them
- Don't imagine unlikely potential needs that aren't really there

Best approach: use your own infrastructure (especially at first!)

- (much more rapid feedback about what works, what doesn't)

If not possible, at least work very closely with initial client team

- ideally sit within 50 feet of each other
- keep other potential clients needs in mind, but get system working via close collaboration with first client first

Google

Thanks!

Further reading:

- Ghemawat, Gobioff, & Leung. *Google File System*, SOSP 2003.
- Barroso, Dean, & Hölzle. *Web Search for a Planet: The Google Cluster Architecture*, IEEE Micro, 2003.
- Dean & Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*, OSDI 2004.
- Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, Fikes, & Gruber. *Bigtable: A Distributed Storage System for Structured Data*, OSDI 2006.
- Corbett et al. *Spanner: Google's Globally Distributed Database*, OSDI 2012.
- Burrows. *The Chubby Lock Service for Loosely-Coupled Distributed Systems*. OSDI 2006.
- Pinheiro, Weber, & Barroso. *Failure Trends in a Large Disk Drive Population*. FAST 2007.
- Barroso & Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Morgan & Claypool Synthesis Series on Computer Architecture, 2009.
- Malewicz *et al*. *Pregel: A System for Large-Scale Graph Processing*. PODC, 2009.
- Schroeder, Pinheiro, & Weber. *DRAM Errors in the Wild: A Large-Scale Field Study*. SEGMETRICS'09.
- Protocol Buffers. http://code.google.com/p/protobuf/

See: http://research.google.com/papers.html

http://research.google.com/people/jeff

Google