Scaling Machine Learning with the Parameter Server

Alex Smola with

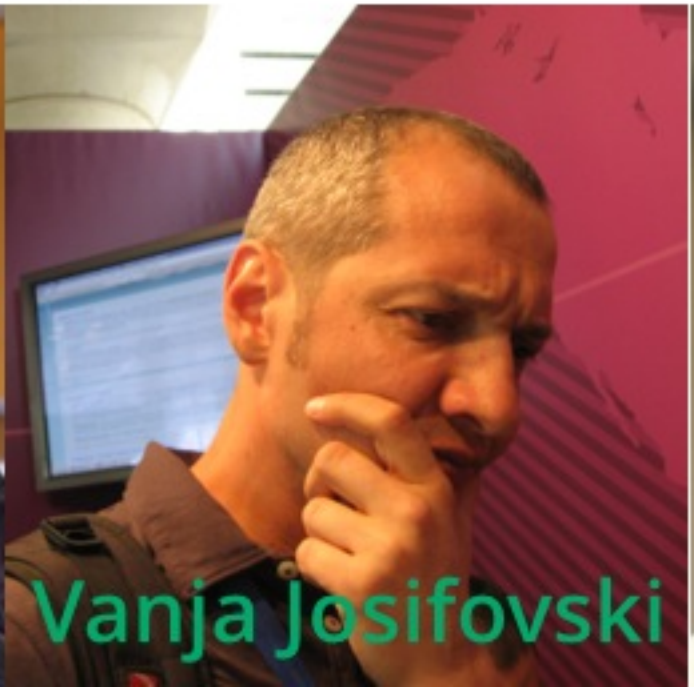
Mu Li, Li Zhou, Dave Andersen, Junwoo Park, Aaron Li, Amr Ahmed, Vanja Josifovski, Bor-Yiing Su, Eugene Shekita, James Long

http://www.istc-cc.cmu.edu/

Intel Science & Technology
Center for Cloud Computing

parameterserver.org

blog.smola.org @smolix




Background

The Challenge

- Scale
  - 100s Terabytes of data
  - 1000s of computers
  - 100 Billions of parameters
- Reality
  - Faulty machines
  - Shared cluster
- Performance
  - Front end serving machines
  - Real time response

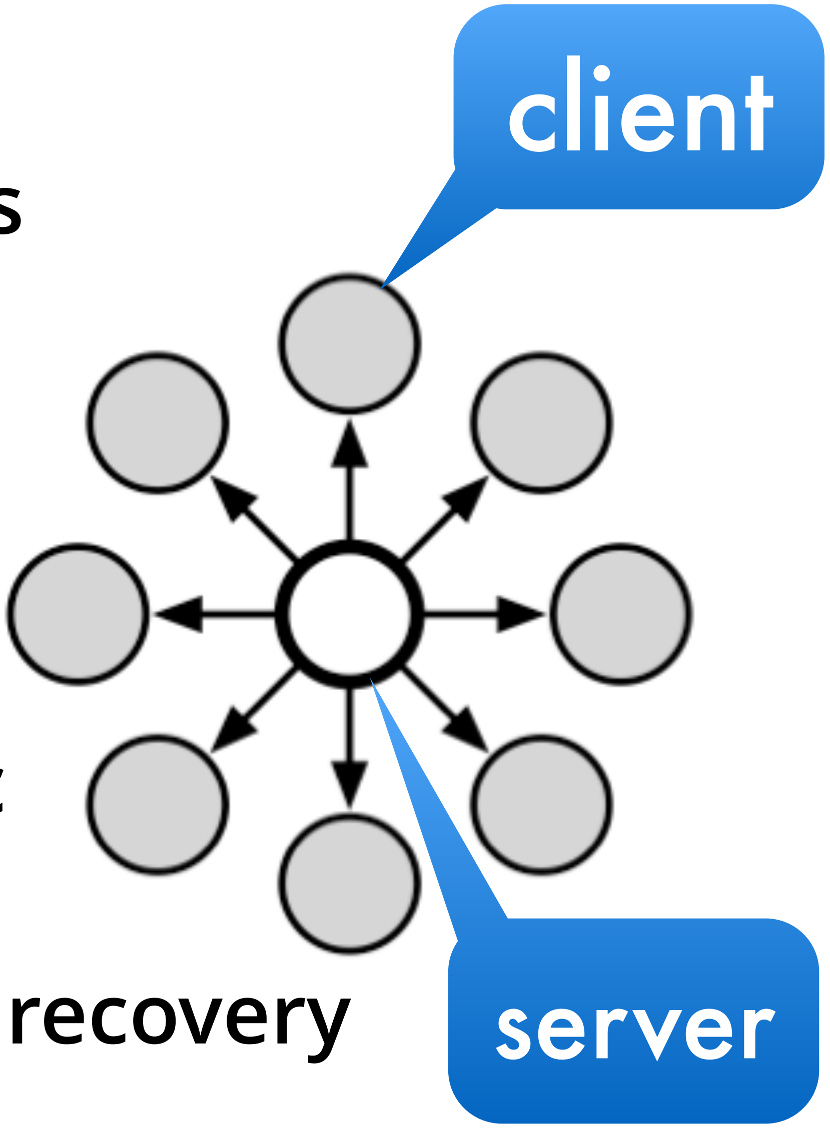
Machine Learning Problems

- Many models have O(1) blocks of O(n) terms (LDA, logistic regression, recommender systems)
- More terms than what fits into RAM (personalized CTR, large inventory, action space)
- Local model typically fits into RAM
- Data needs many disks for distribution
- Decouple data processing from aggregation

- Optimize for the 80% of all ML problems

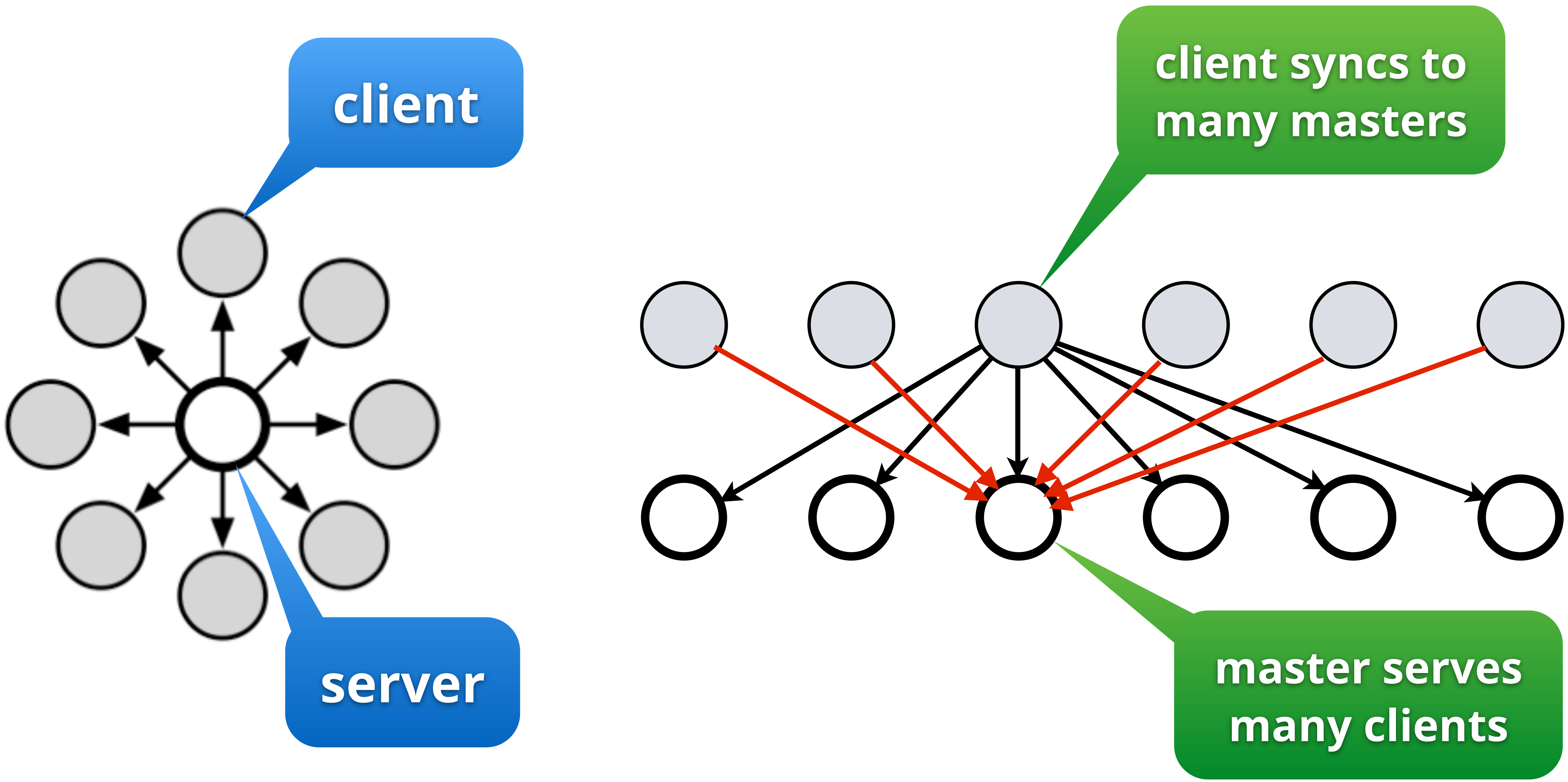
General parallel algorithm template

- Clients have local view of parameters
- P2P is infeasible since $O(n^2)$ connections
- Synchronize with parameter server
  - Reconciliation protocol
    average parameters, lock variables
  - Synchronization schedule
    asynchronous, synchronous, episodic
  - Load distribution algorithm
    uniform distribution, fault tolerance, recovery



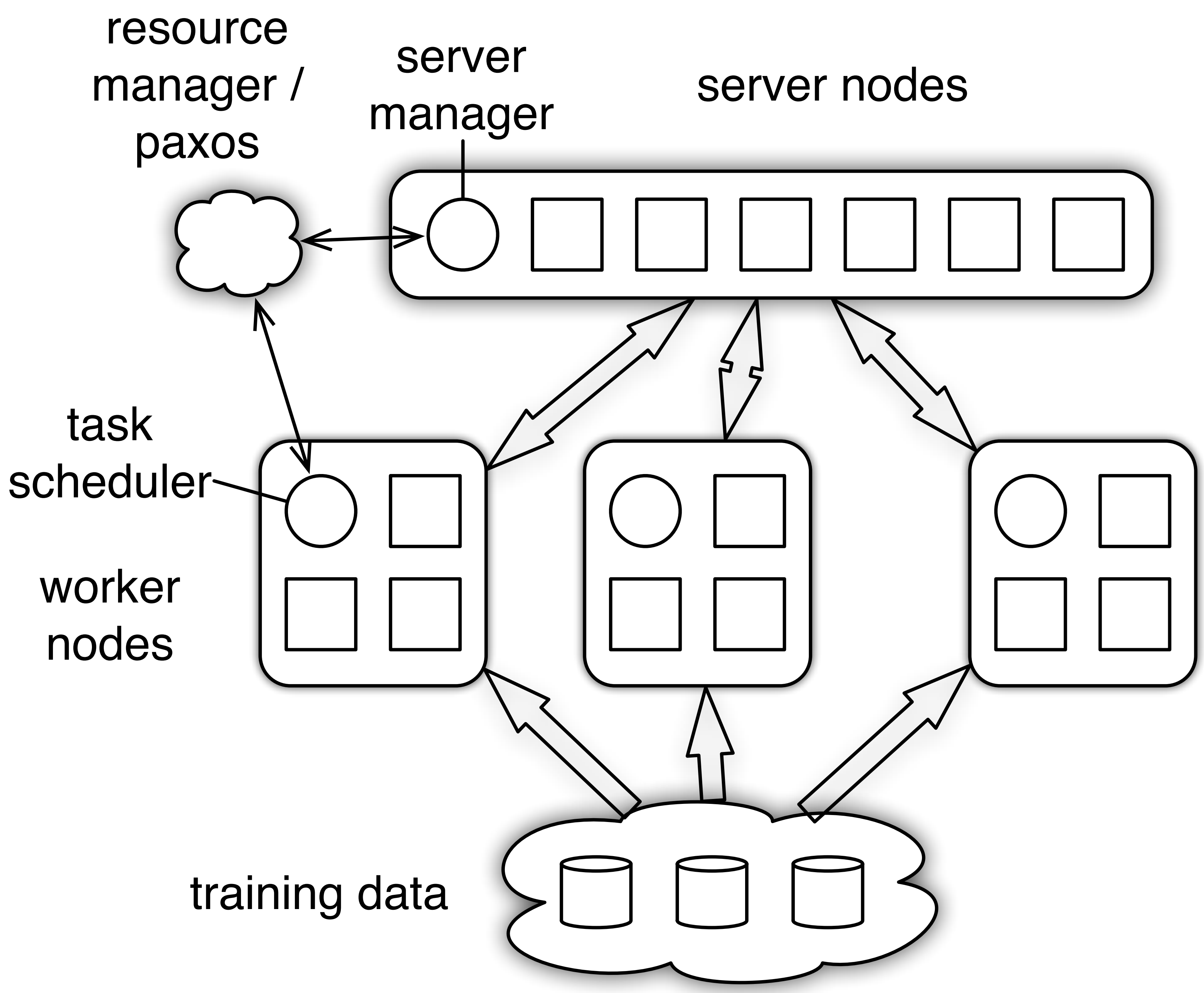
Smola & Narayanamurthy, 2010, VLDB
Gonzalez et al., 2012, WSDM
Shervashidze et al., 2013, WWW

Communication pattern



put(keys,values,clock), get(keys,values,clock)

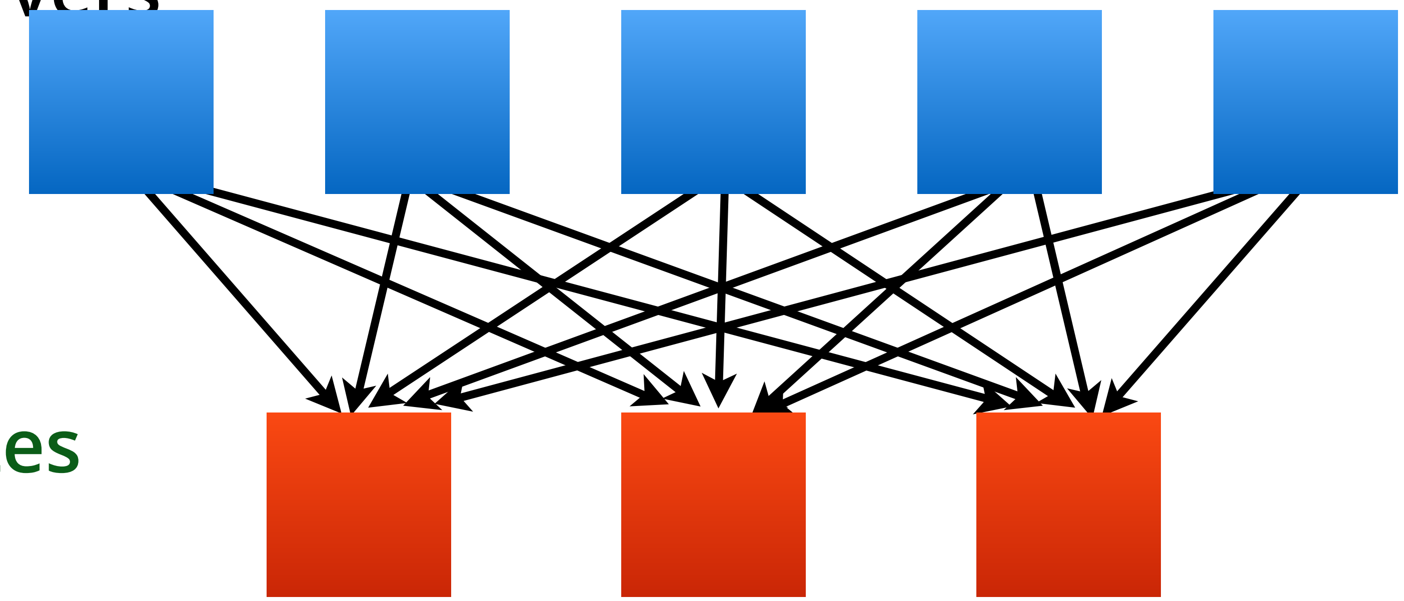
Architecture




Key layout & recovery

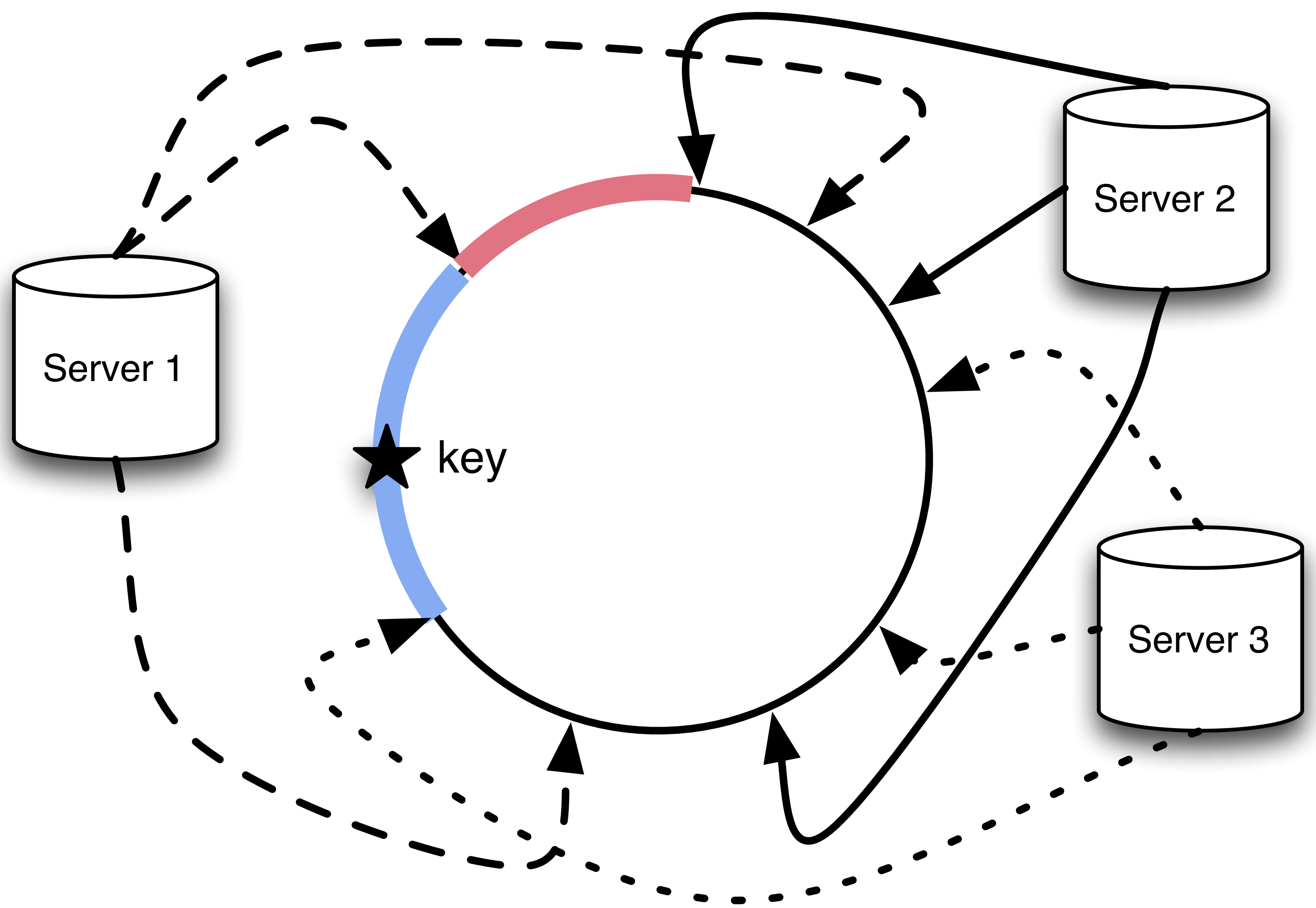
Consistent Hashing

- Caching
  - Store many (key,value) pairs
  - Linear scaling in clients & servers
  - Automatic key distribution
- memcached
  - (key,value) servers
  - client access library distributes access patterns
  - randomized O(n) bandwidth
  - aggregate O(n) bandwidth
  - load balancing via hashing
  - no versioned writes / vector clocks
  - very expensive to iterate over all keys for a given server



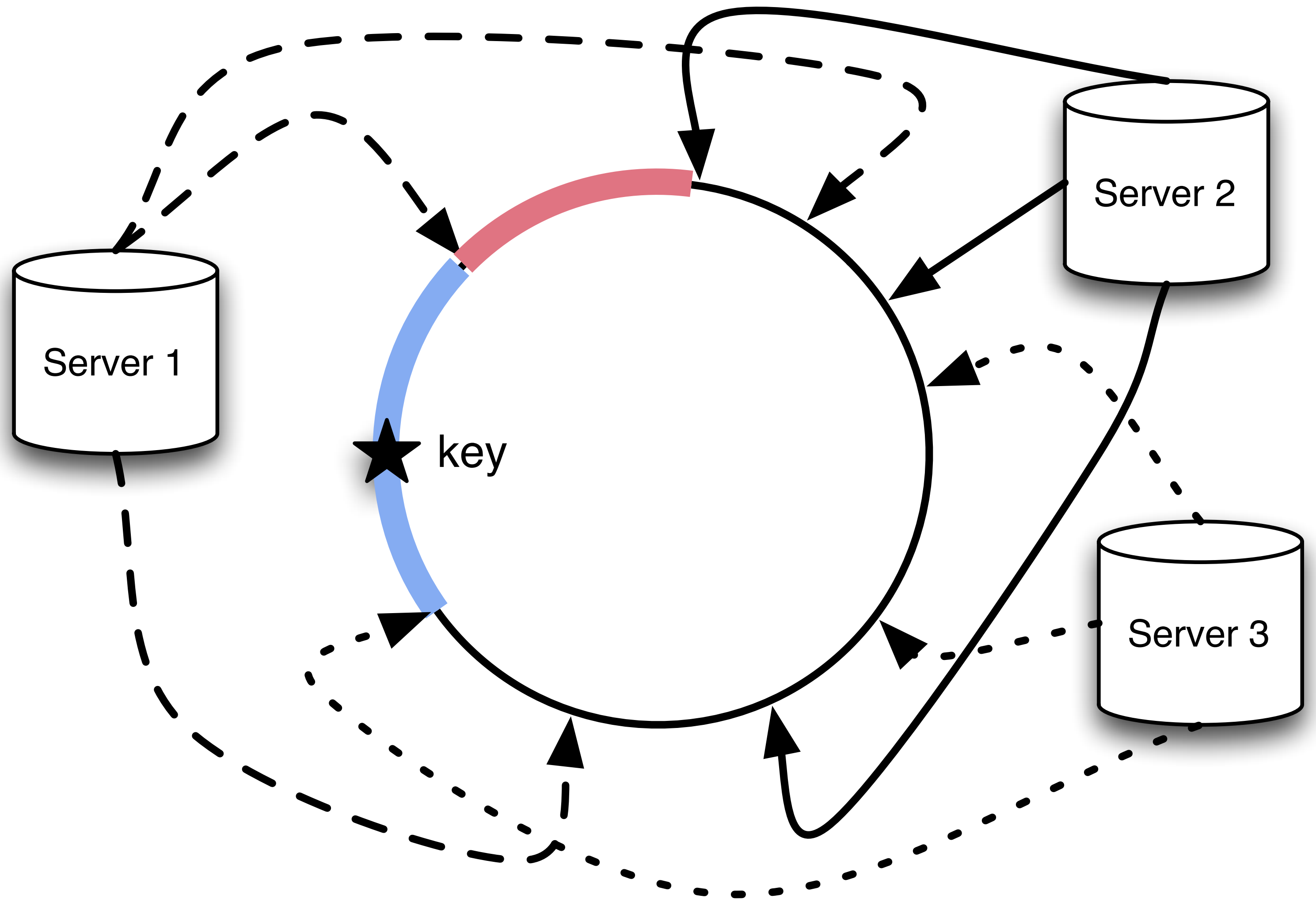
$$m(\text{key}, \mathcal{M}) = \underset{m' \in \mathcal{M}}{\operatorname{argmin}}\, h(\text{key}, m')$$

Keys arranged in a DHT



- Virtual servers
  - loadbalancing
  - multithreading
- DHT
  - contiguous key range for clients
  - easy bulk sync
  - easy insertion of servers
- Replication
  - Machines hold replicas
  - Easy fallback
  - Easy insertion / repair

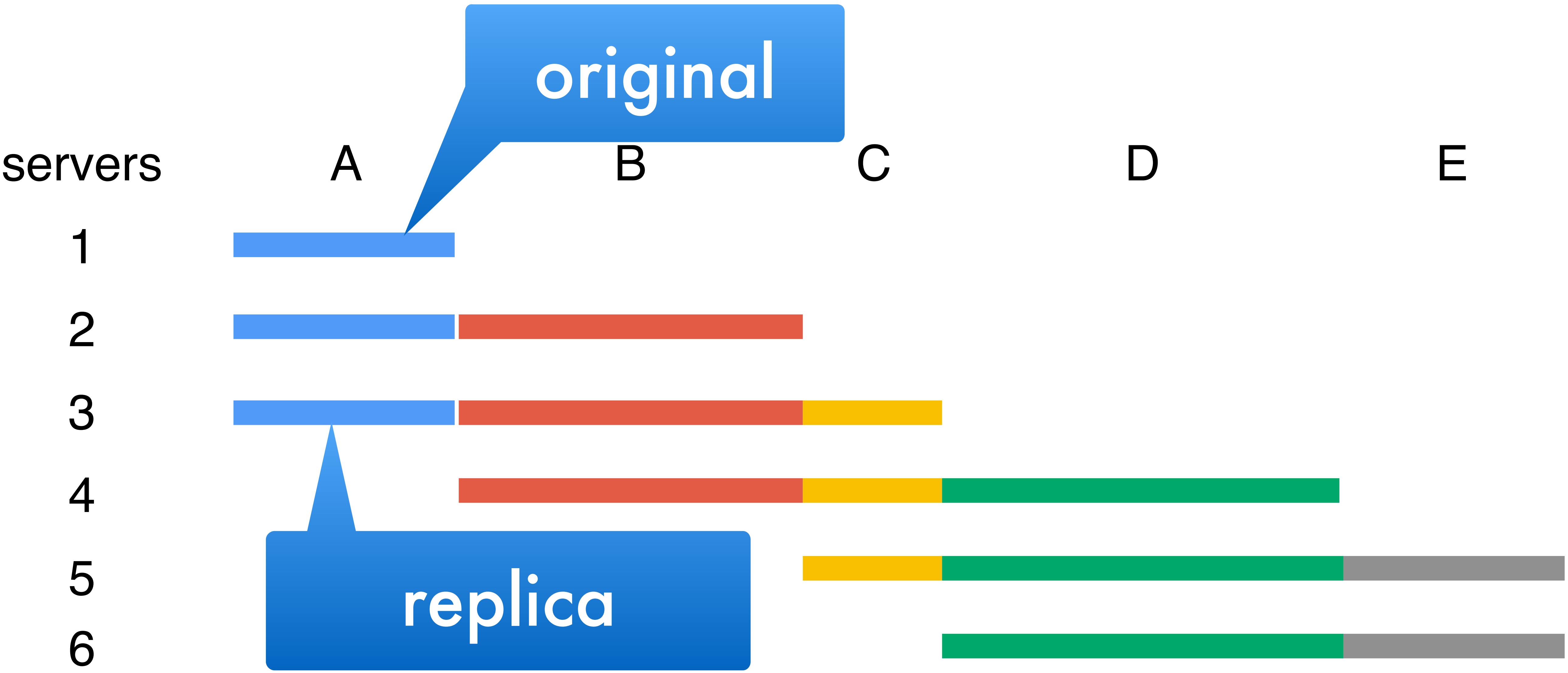
Keys arranged in a DHT



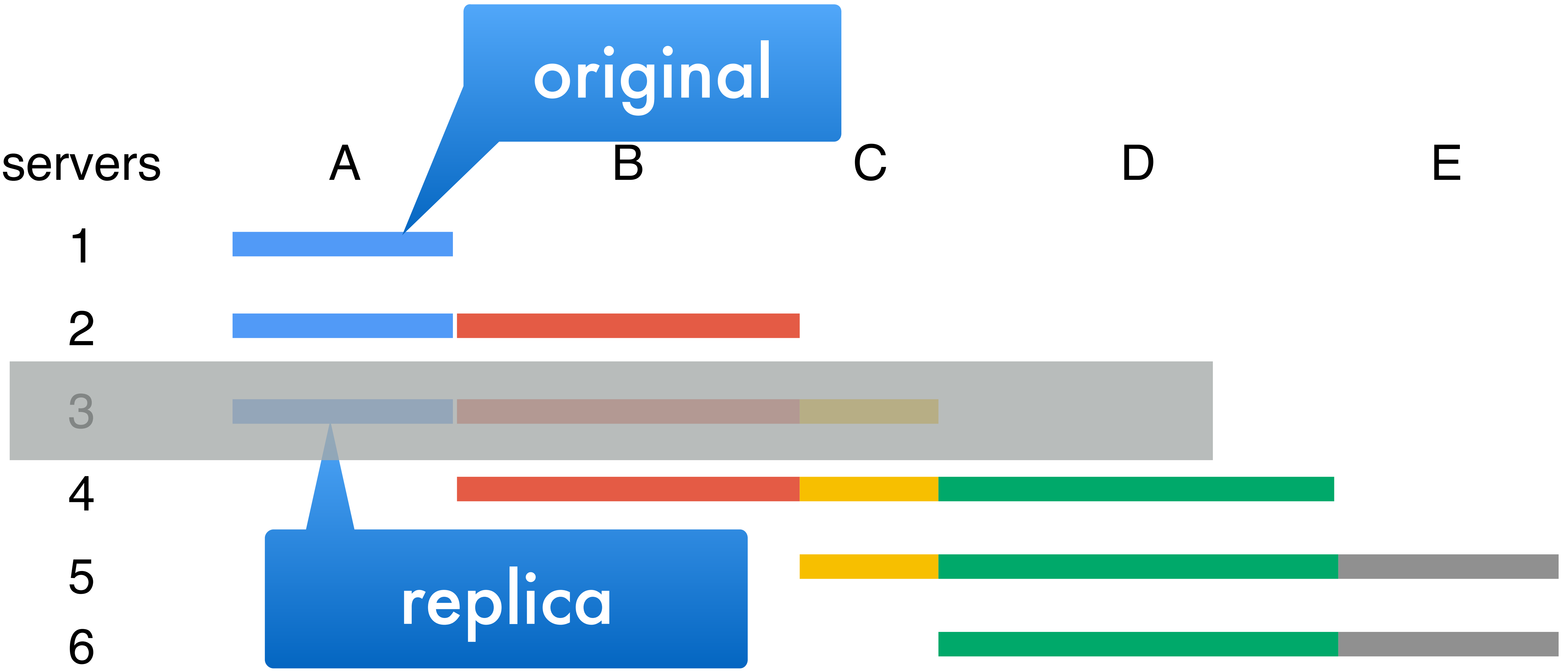
Yes, we screwed up before!
And everyone copied us!

- Virtual servers
  - loadbalancing
  - multithreading
- DHT
  - contiguous key range for clients
  - easy bulk sync
  - easy insertion of servers
- Replication
  - Machines hold replicas
  - Easy fallback
  - Easy insertion / repair

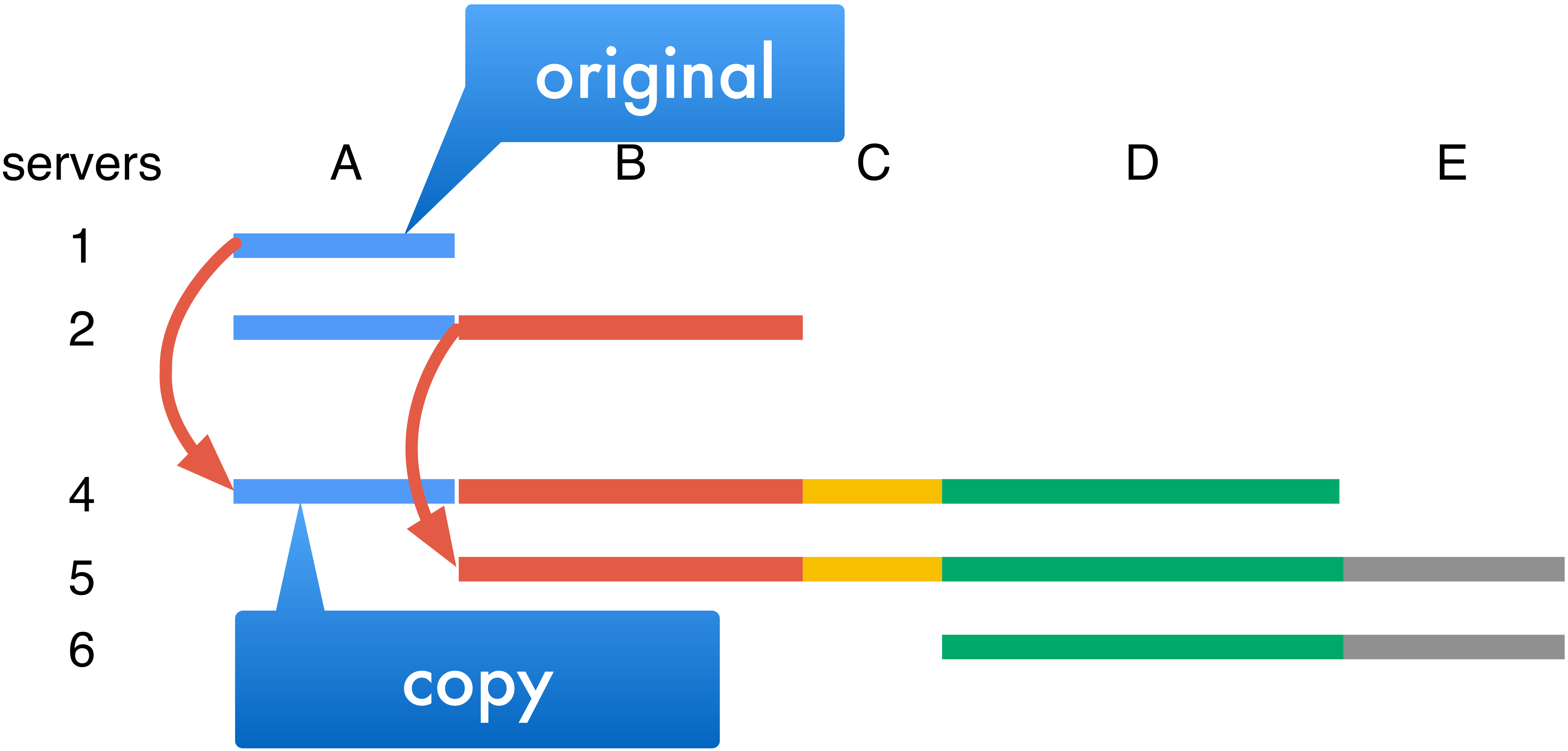
Key layout



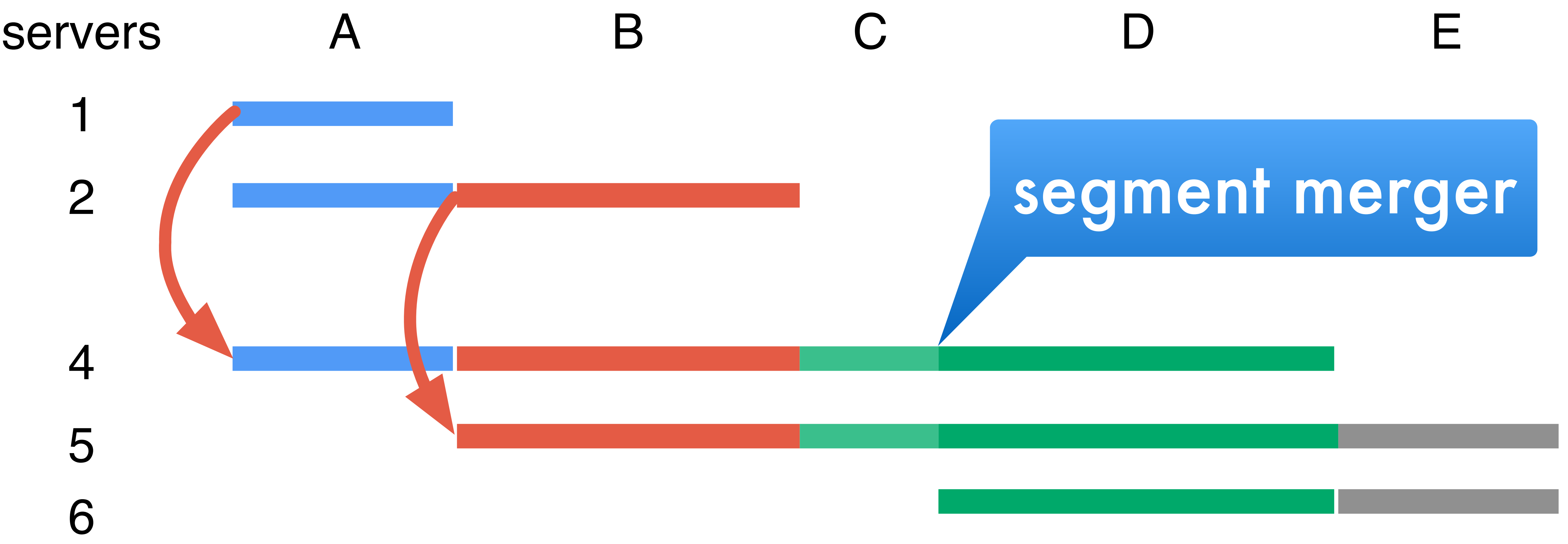
Key layout



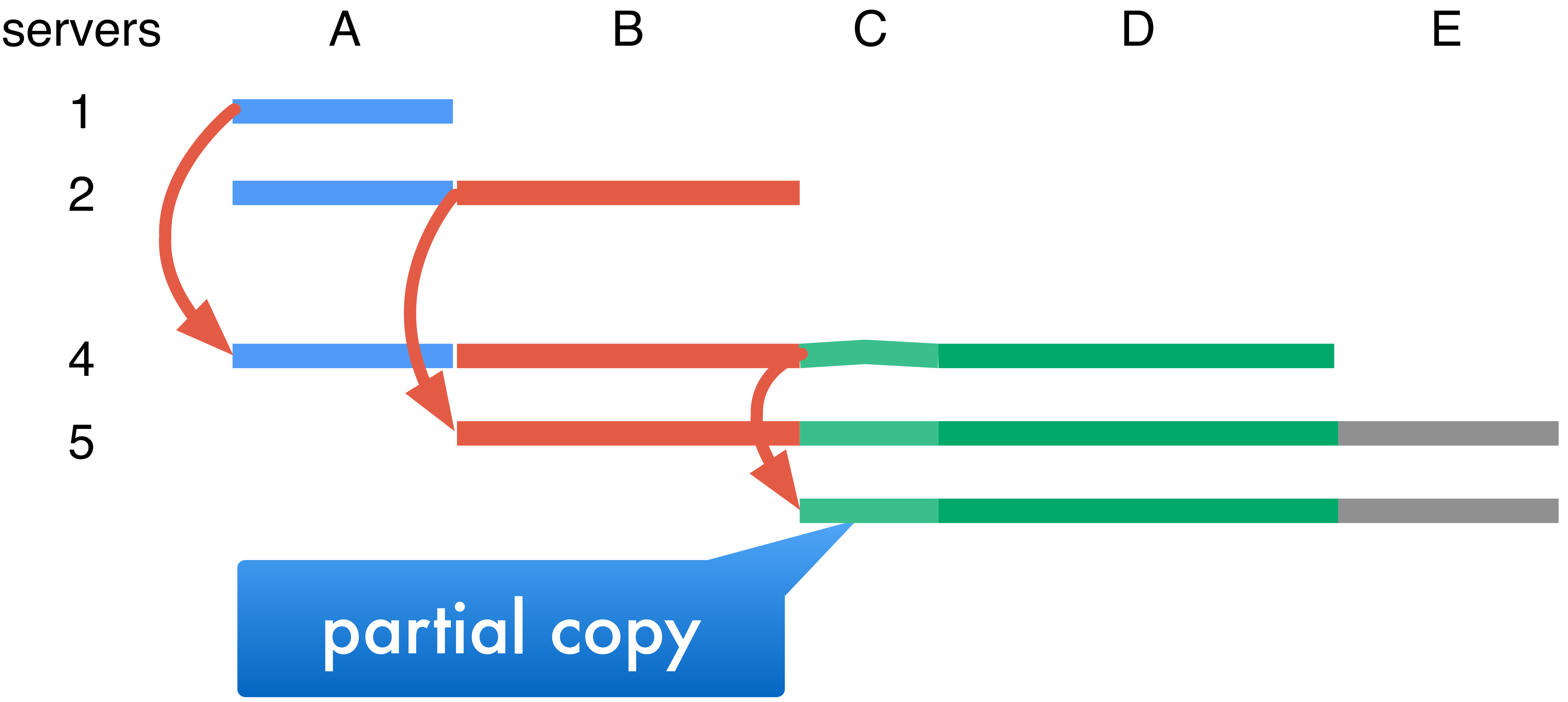
Key layout



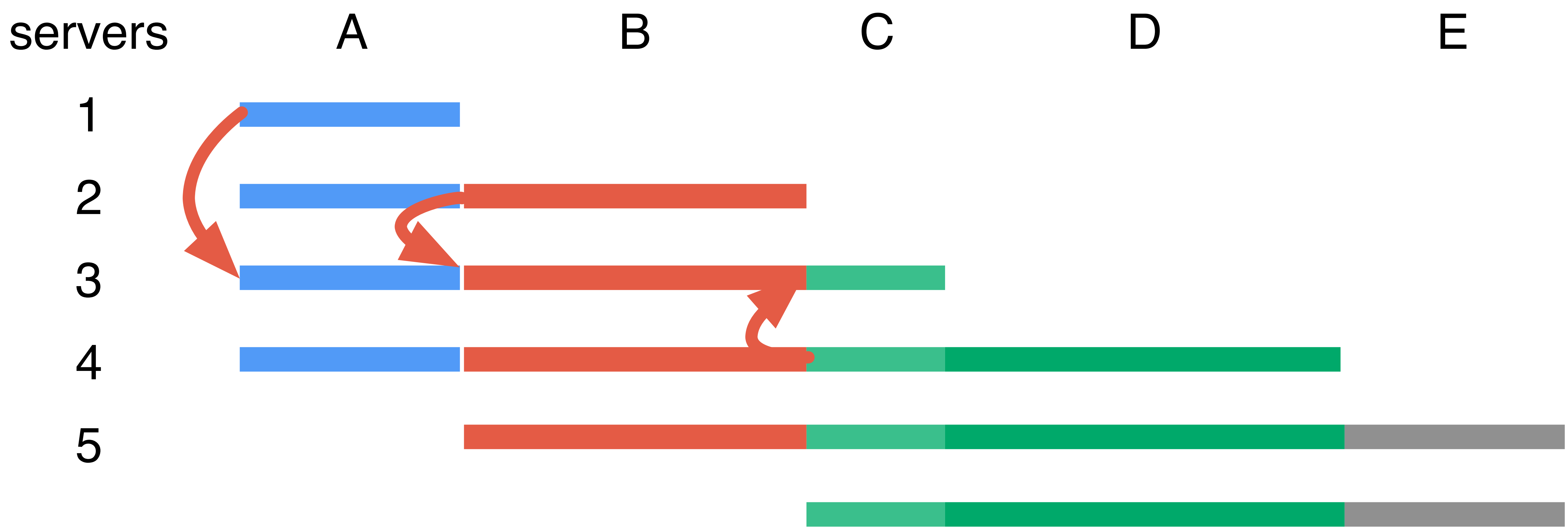
Key layout



Key layout



Recovery / server insertion



- Precopy server content to new candidate (3)
- After precopy ended, send log
- For k virtual servers this causes $O(k^{-2})$ delay
- Consistency using vector clocks

Communication

Message Compression

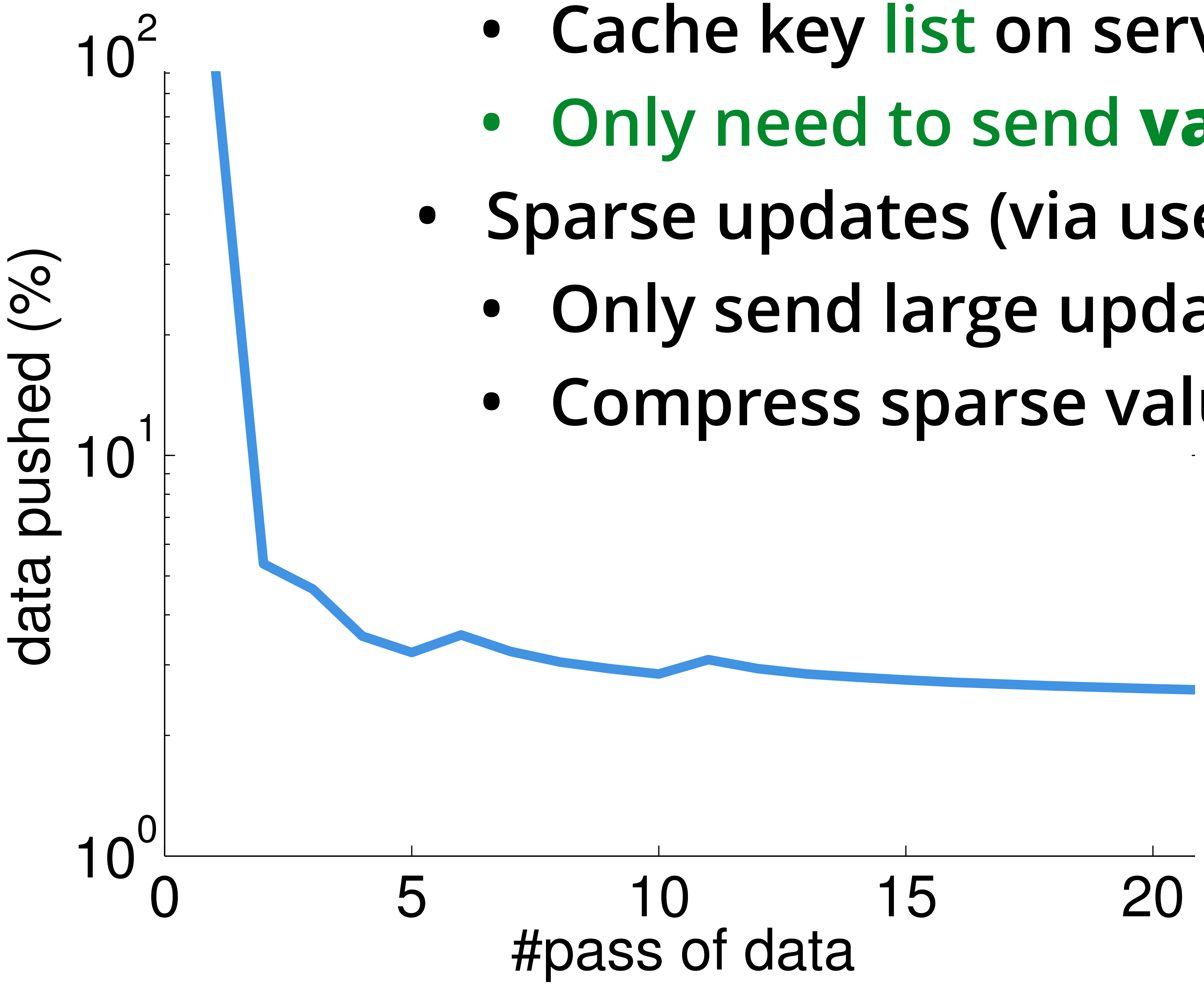
- Convergence speed depends on communication efficiency
  - Sending (key,value) pairs is inefficient
    Send only values (cache key list) instead
  - Sending small gradients is inefficient
    Send only sufficiently large ones instead
  - Updating near-optimal values is inefficient
    Send only large violators of KKT conditions
- Filter data before sending

Filters

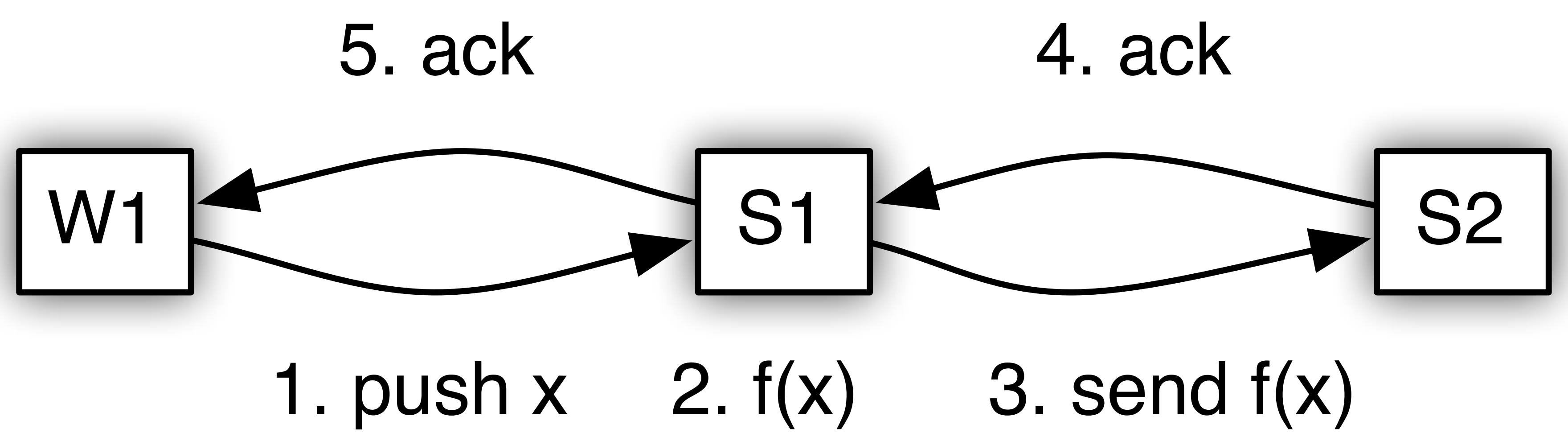
- Scheduling
  have controller decide when to send
  (this requires very smart controller)
- Filtering
  have algorithm decide when to shut up
  - Gradient (only send large gradients)
  - KKT (only send variables violating KKT)
  - Randomized (sparse random vectors)
  - Quantization (reduce accuracy)

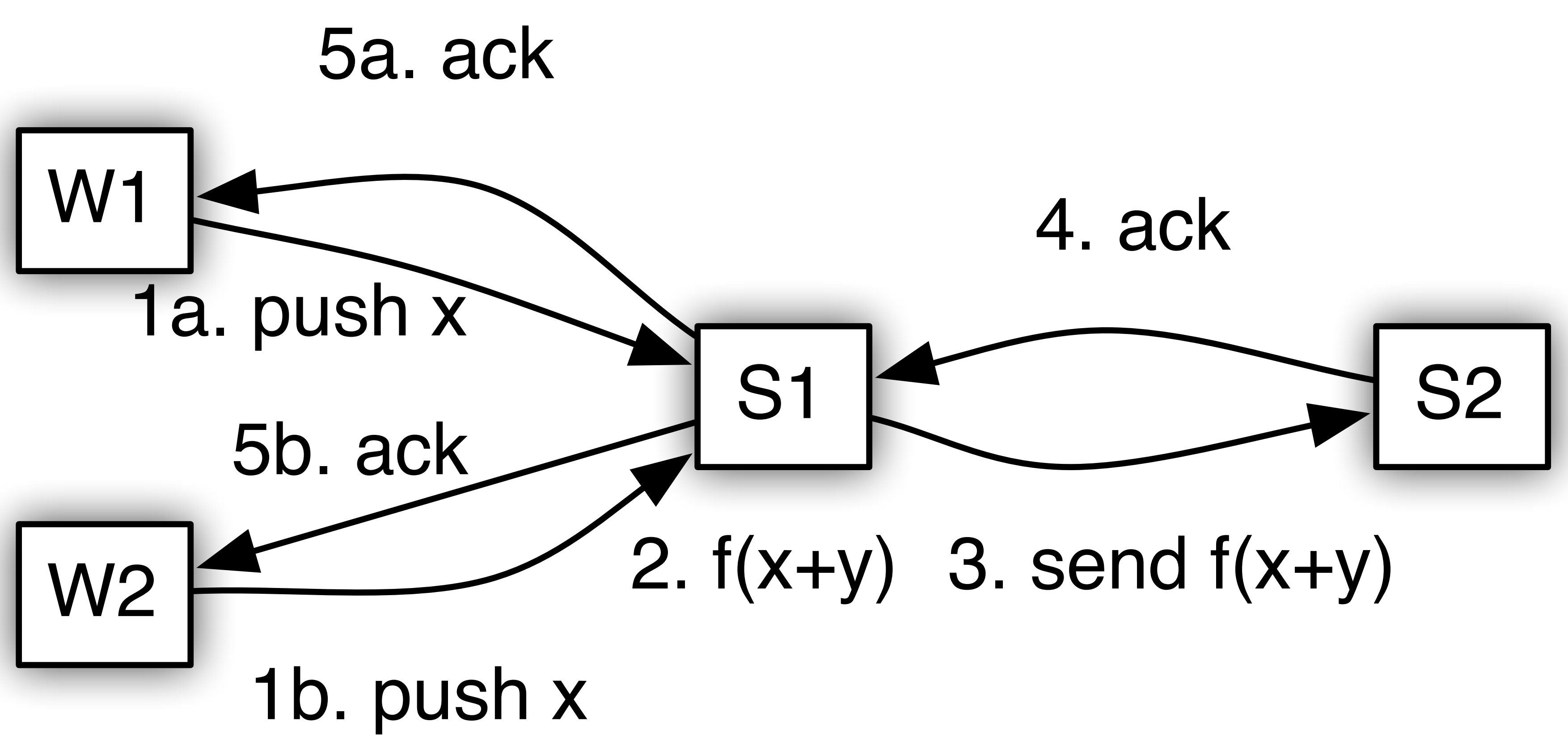
Message Compression

- Sparse Vectors aka (key,value) pairs
  - Cache key list on server
  - Only need to send **values**
- Sparse updates (via user defined filter)
  - Only send large updates
  - Compress sparse value list



Message Aggregation on Server





Messaging

- Datatypes are `eigen3` native
  - Dense vectors
  - Sparse vectors
- `Push(Header flag)`
- `Pull(Header flag)`

  `Flag` may specify
  - Value or delta update
  - key range
  - recipient (all server, all clients, particular node)

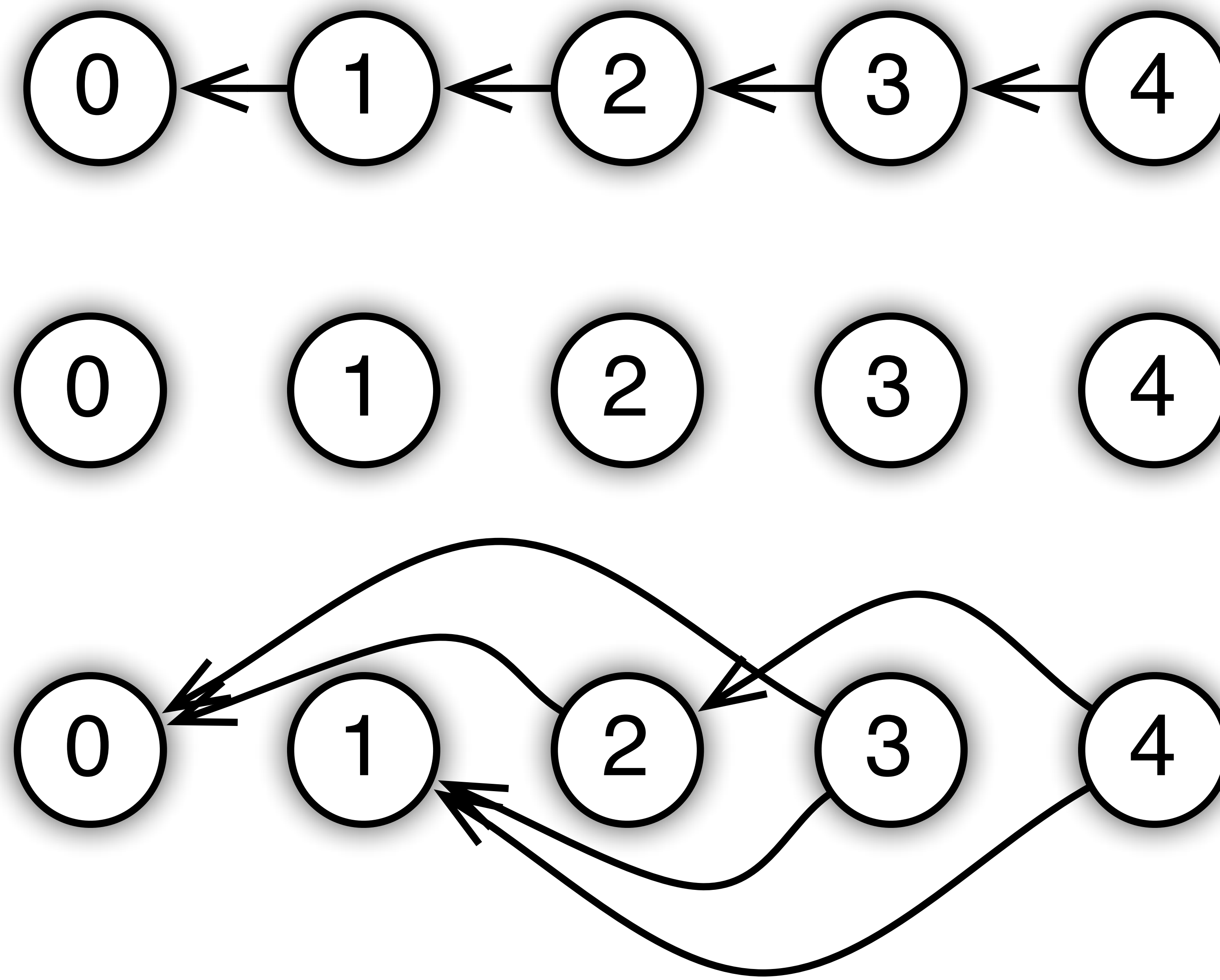
Shared pointer. No copy on queue (by default)!

Consistency models

(a) Sequential

(b) Eventual

(c) Bounded delay



via task processing engine on client/controller

Vector Clocks for Ranges

- Keep track of when we received an update from a client / server.
- For c clients this means O(c) metadata
  This is impossible to store per key (Dynamo)
- Very cheap and feasible for ranges
- When inconsistent ranges, split segments [A,D] splits into [A,B], [B,C] and [C,D] when receiving message for [B,C]
- This is infrequent + defragmentation

Experiments

Guinea pig - logistic regression

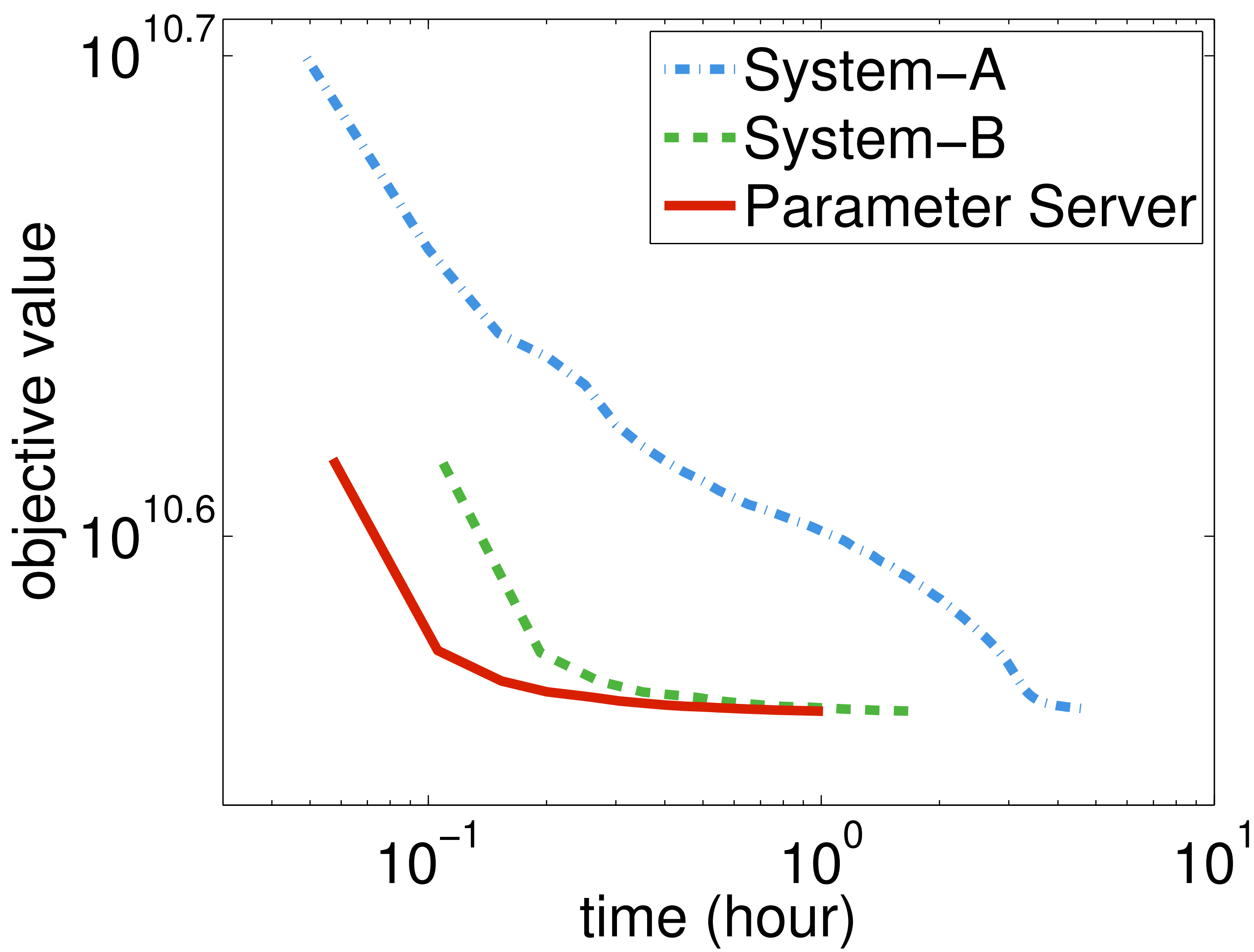
- **Implementation on Parameter Server**

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1$$

	Method	Consistency	LOC
System-A	L-BFGS	Sequential	10,000
System-B	Block PG	Sequential	30,000
Parameter Server	Block PG	Bounded Delay KKT Filter	300

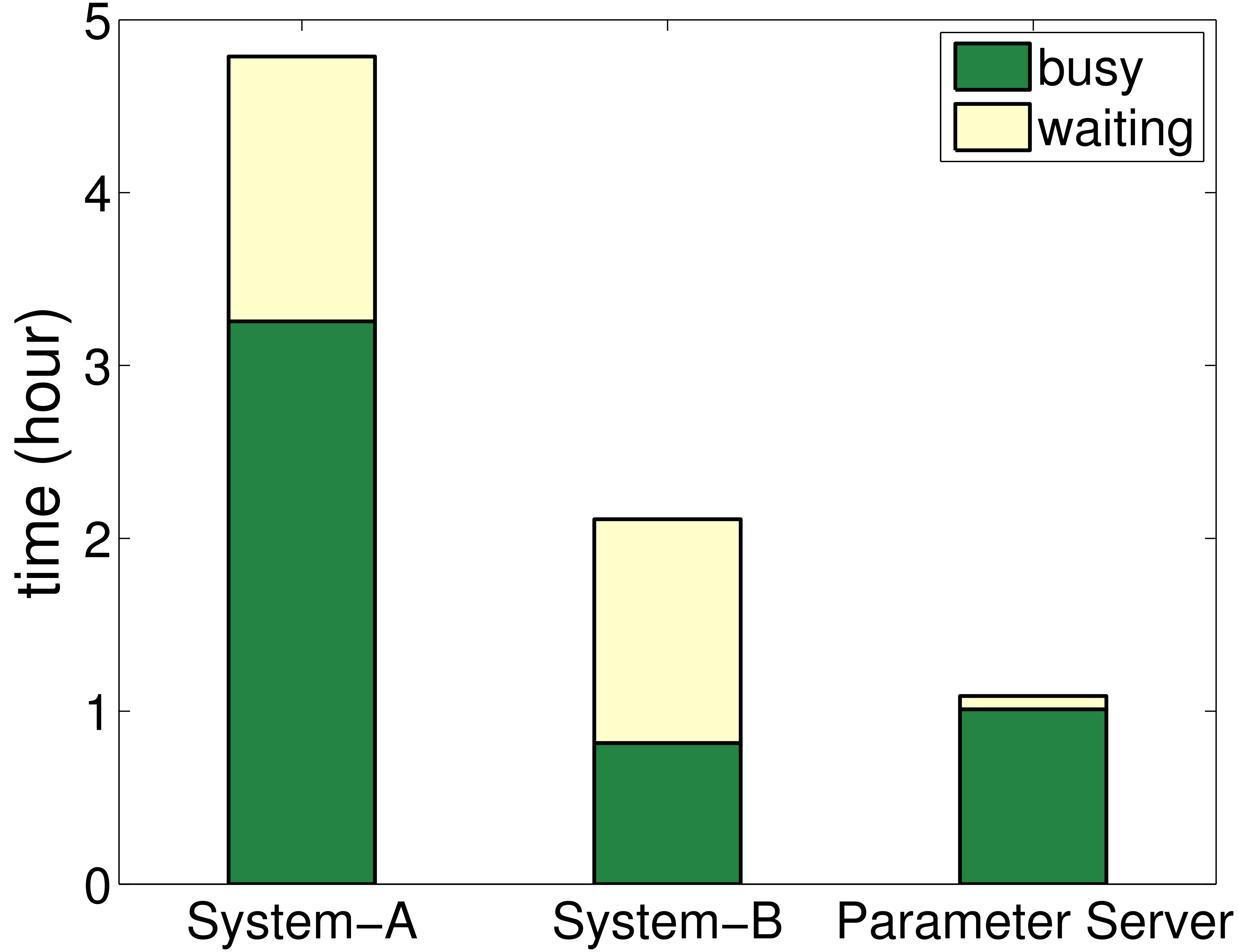
Convergence speed

- System A and B are production systems at a very large internet company …



500TB CTR data
100B variables
1000 machines

Scheduling Efficiency



Distributed CountMin Sketch

- Clients only act as data preprocessors
- Shard keys over servers for balancing
- Replication between machines on DHT
- Servers perform simple updates

- 15 servers, 40GBit network (dedicated)

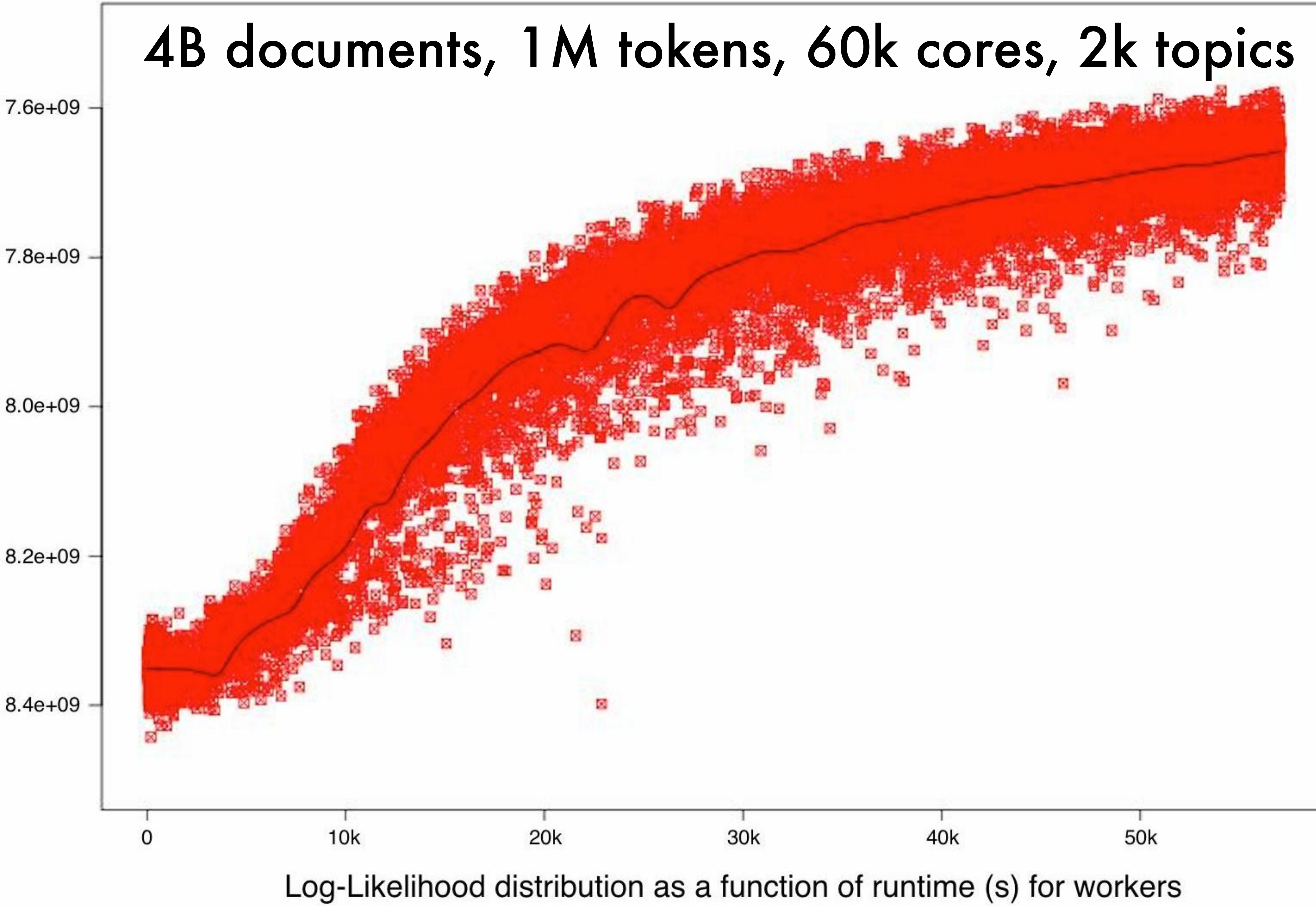
$$M[h(k,j),j] \leftarrow M[h(k,j),j] + v \text{ for all } j \in \{1,\ldots d\}$$

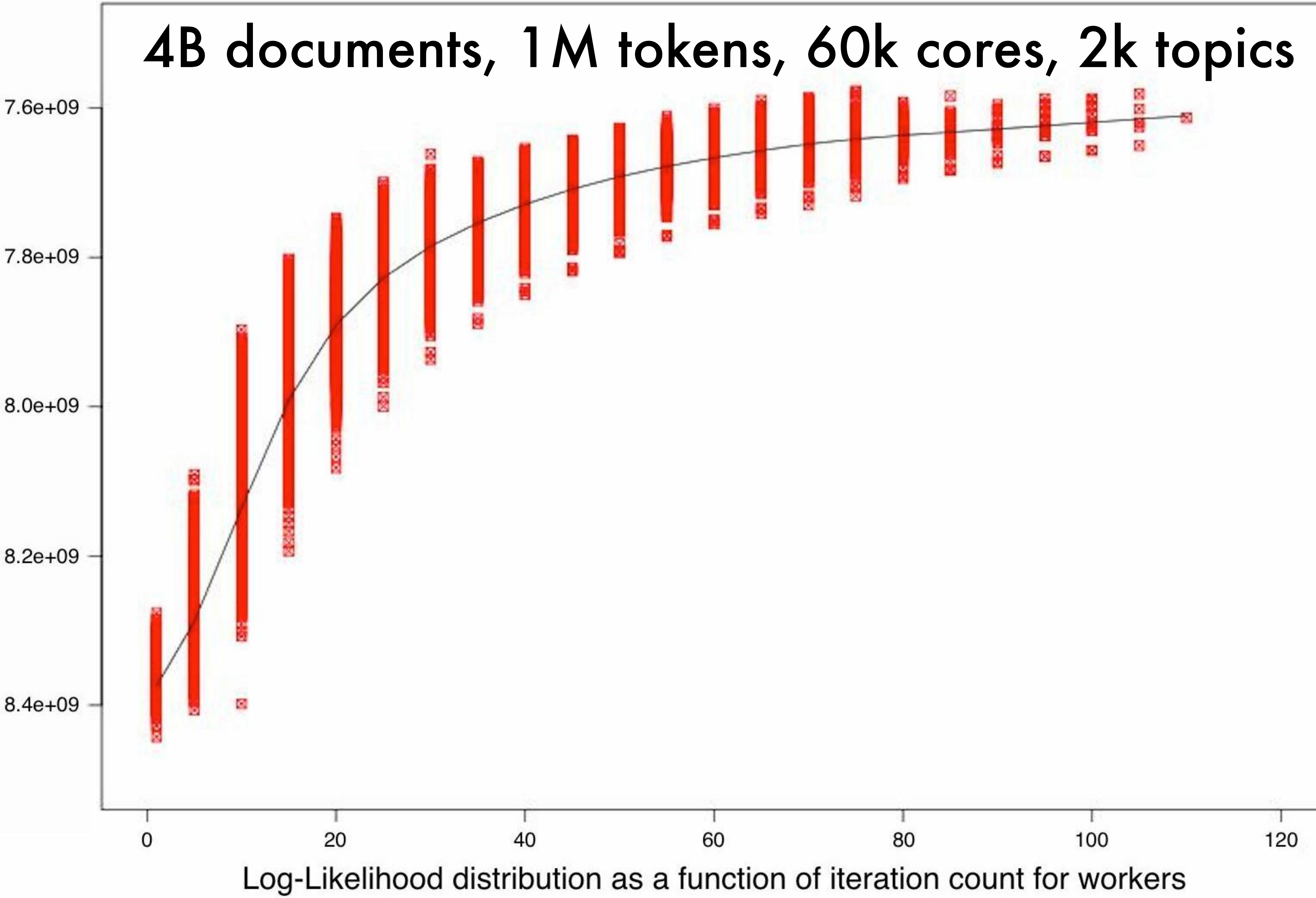
Peak inserts per second	1.3 billion
Average inserts per second	1.1 billion
Peak network bandwidth per machine	4.37 GBit/s
Time to recover a failed node	0.8 second

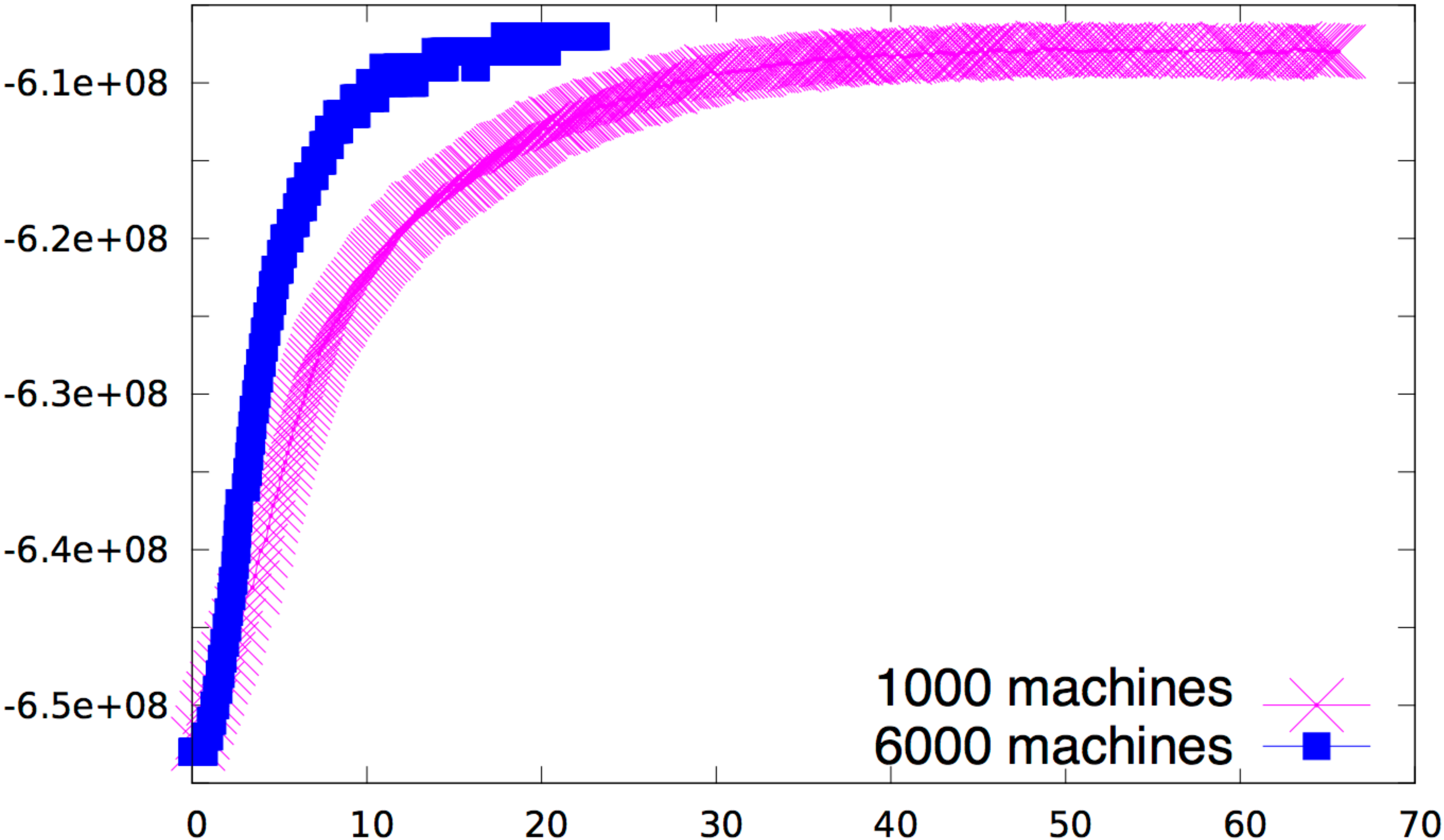
Limited by DRAM Latency

Gibbs Sampler for LDA

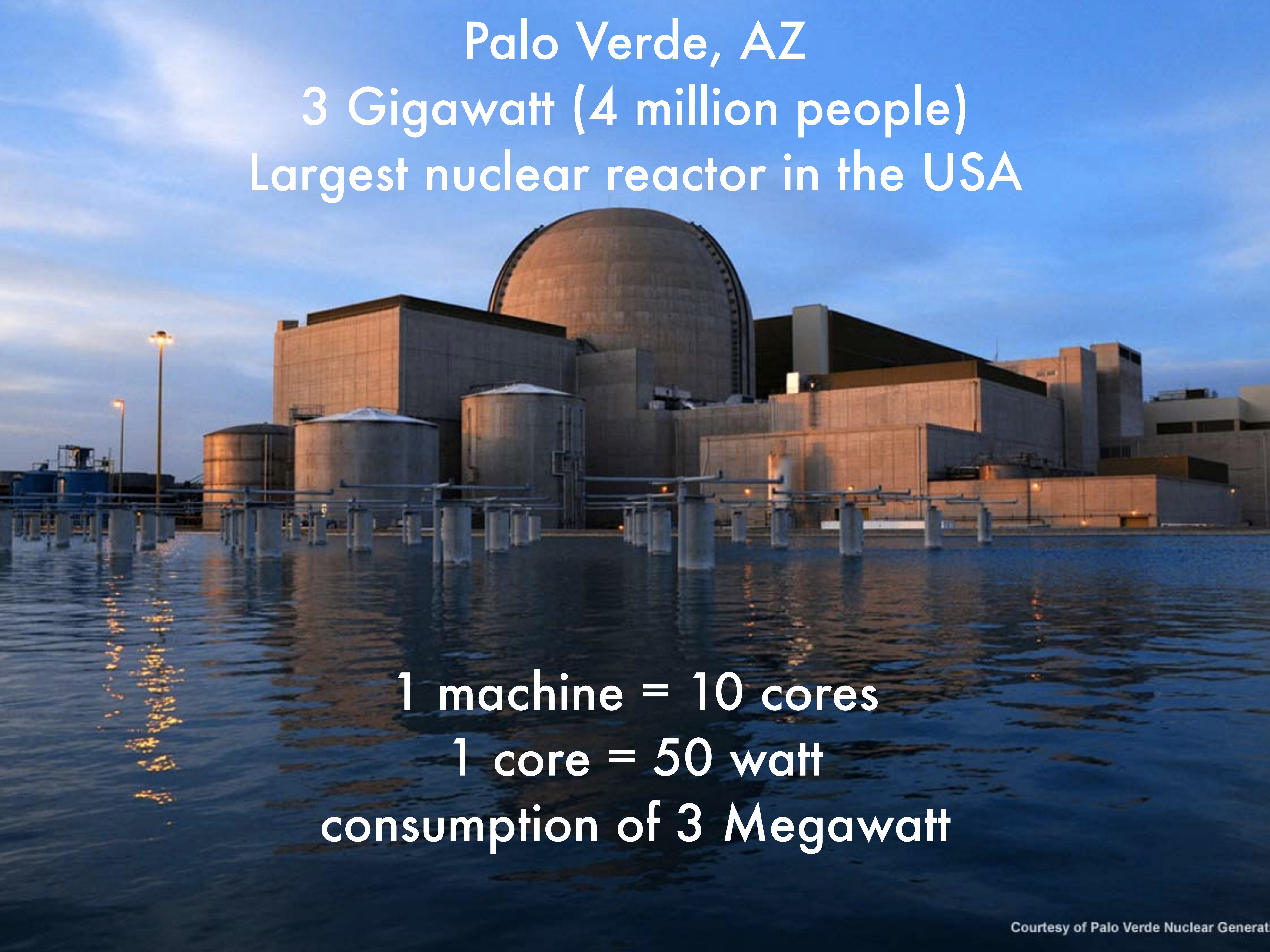
- For 1000 iterations do
  - For each document do
    - For each word in the document do
      - Resample topic for the word
      - Update local (document, topic) table
      - Generate local update message
    - Update local table
      - Lock local (word,topic) table
      - Update local (word,topic) table
      - Unlock local (word,topic) table
    - Synchronize local and global tables


4B documents, 1M tokens, 60k cores, 2k topics
7.6e+09
7.8e+09
8.0e+09
8.2e+09
8.4e+09
0
10k
20k
30k
40k
50k
Log-Likelihood distribution as a function of runtime (s) for workers


4B documents, 1M tokens, 60k cores, 2k topics
7.6e+09
7.8e+09
8.0e+09
8.2e+09
8.4e+09
0
20
40
60
80
100
120
Log-Likelihood distribution as a function of iteration count for workers


-6.1e+08
-6.2e+08
-6.3e+08
-6.4e+08
-6.5e+08
0
10
20
30
40
50
60
70
1000 machines
6000 machines

Palo Verde, AZ
3 Gigawatt (4 million people)
Largest nuclear reactor in the USA


Palo Verde, AZ
3 Gigawatt (4 million people)
Largest nuclear reactor in the USA
1 machine = 10 cores
1 core = 50 watt
consumption of 3 Megawatt
Courtesy of Palo Verde Nuclear Generat

See our paper at OSDI 2014

parameterserver.org

blog.smola.org @smolix