Spanner: Google's Globally-Distributed Database

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Courtesy

- Nazmus Saquib
- Ishtiyaque Ahmad

Paper Information

- **Paper title:** Spanner: Google's Globally-Distributed Database
- **Authors:** Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman JJ, Ghemawat S, Gubarev A, Heiser C, Hochschild P, Hsieh W, Kanthak S, Kogan E, Li H, Lloyd A, Melnik S, Mwaura D, Nagle D, Quinlan S, Rao R, Rolig L, Saito Y, Szymaniak M, Taylor C, Wang R, Woodford D
- **Published in:** 10th USENIX Symposium on Operating Systems Design and Implementation (2012)
- **DOI:** 10.1145/361002.361005

Motivation

- Google's advertising backend: **F1**
  - originally based on MySQL
  - sharded manually
  - data too large for MySQL
- Stored some data to external BigTable:
  - difficult to use for applications with complex, evolving schema
  - no cross-row transaction

Spanner came up with solutions to all these and much more

Motivation

- Google's advertising backend: **F1**
  - originally based on MySQL
  - sharded manually
  - data too large for MySQL
- Stored some data to external BigTable:
  - difficult to use for applications with complex, evolving schema
  - no cross-row transaction

Spanner came up with solutions to all these and much more

Motivation

- Google's advertising backend: **F1**
  - originally based on MySQL
  - sharded manually
  - data too large for MySQL
- Stored some data to external BigTable:
  - difficult to use for applications with complex, evolving schema
  - no cross-row transaction

Spanner came up with solutions to all these and much more

Motivation

- Google's advertising backend: **F1**
  - originally based on MySQL
  - sharded manually
  - data too large for MySQL
- Stored some data to external BigTable:
  - difficult to use for applications with complex, evolving schema
  - no cross-row transaction

Spanner came up with solutions to all these and much more

Motivation

- Google's advertising backend: **F1**
  - originally based on MySQL
  - sharded manually
  - data too large for MySQL
- Stored some data to external BigTable:
  - difficult to use for applications with complex, evolving schema
  - no cross-row transaction

Spanner came up with solutions to all these and much more

Motivation

- Google's advertising backend: **F1**
  - originally based on MySQL
  - sharded manually
  - data too large for MySQL
- Stored some data to external BigTable:
  - difficult to use for applications with complex, evolving schema
  - no cross-row transaction

Spanner came up with solutions to all these and much more

Motivation

- Google's advertising backend: **F1**
  - originally based on MySQL
  - sharded manually
  - data too large for MySQL
- Stored some data to external BigTable:
  - difficult to use for applications with complex, evolving schema
  - no cross-row transaction

Spanner came up with solutions to all these and much more

Motivation

- Google's advertising backend: **F1**
  - originally based on MySQL
  - sharded manually
  - data too large for MySQL
- Stored some data to external BigTable:
  - difficult to use for applications with complex, evolving schema
  - no cross-row transaction

Spanner came up with solutions to all these and much more

Motivation

- Google's advertising backend: **F1**
  - originally based on MySQL
  - sharded manually
  - data too large for MySQL
- Stored some data to external BigTable:
  - difficult to use for applications with complex, evolving schema
  - no cross-row transaction

Spanner came up with solutions to all these and much more

Features

- Globally distributed
- Scalable - *"Spanner is designed to scale up to millions of machines across hundreds of datacenters and trillions of database rows."*
- Cross-datacenter replication of data
- Semi-relational
- Multiversioned

Features

- Globally distributed
- Scalable - *"Spanner is designed to scale up to millions of machines across hundreds of datacenters and trillions of database rows."*
- Cross-datacenter replication of data
- Semi-relational
- Multiversioned

Features

- Globally distributed
- Scalable - *"Spanner is designed to scale up to millions of machines across hundreds of datacenters and trillions of database rows."*
- Cross-datacenter replication of data
- Semi-relational
- Multiversioned

Features

- Globally distributed
- Scalable - *"Spanner is designed to scale up to millions of machines across hundreds of datacenters and trillions of database rows."*
- Cross-datacenter replication of data
- Semi-relational
- Multiversioned

Features

- Globally distributed
- Scalable - *"Spanner is designed to scale up to millions of machines across hundreds of datacenters and trillions of database rows."*
- Cross-datacenter replication of data
- Semi-relational
- Multiversioned

Features *(Cntd.)*

- SQL based query language
- External consistency of distributed transactions
  - $T(e_{1commit}) < T(e_{2start}) \implies s_1 < s_2$
  - first system at global scale
- Lock free distributed read transactions

Features *(Cntd.)*

- SQL based query language
- External consistency of distributed transactions
  - $T(e_{1commit}) < T(e_{2start}) \implies s_1 < s_2$
  - first system at global scale
- Lock free distributed read transactions

Features *(Cntd.)*

- SQL based query language
- External consistency of distributed transactions
  - $T(e_{1commit}) < T(e_{2start}) \implies s_1 < s_2$
  - first system at global scale
- Lock free distributed read transactions

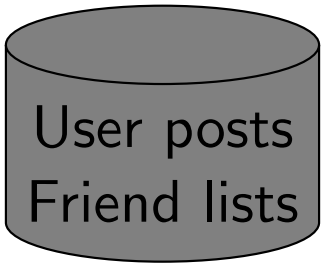
Features *(Cntd.)*

- SQL based query language
- External consistency of distributed transactions
  - $T(e_{1commit}) < T(e_{2start}) \implies s_1 < s_2$
  - first system at global scale
- Lock free distributed read transactions

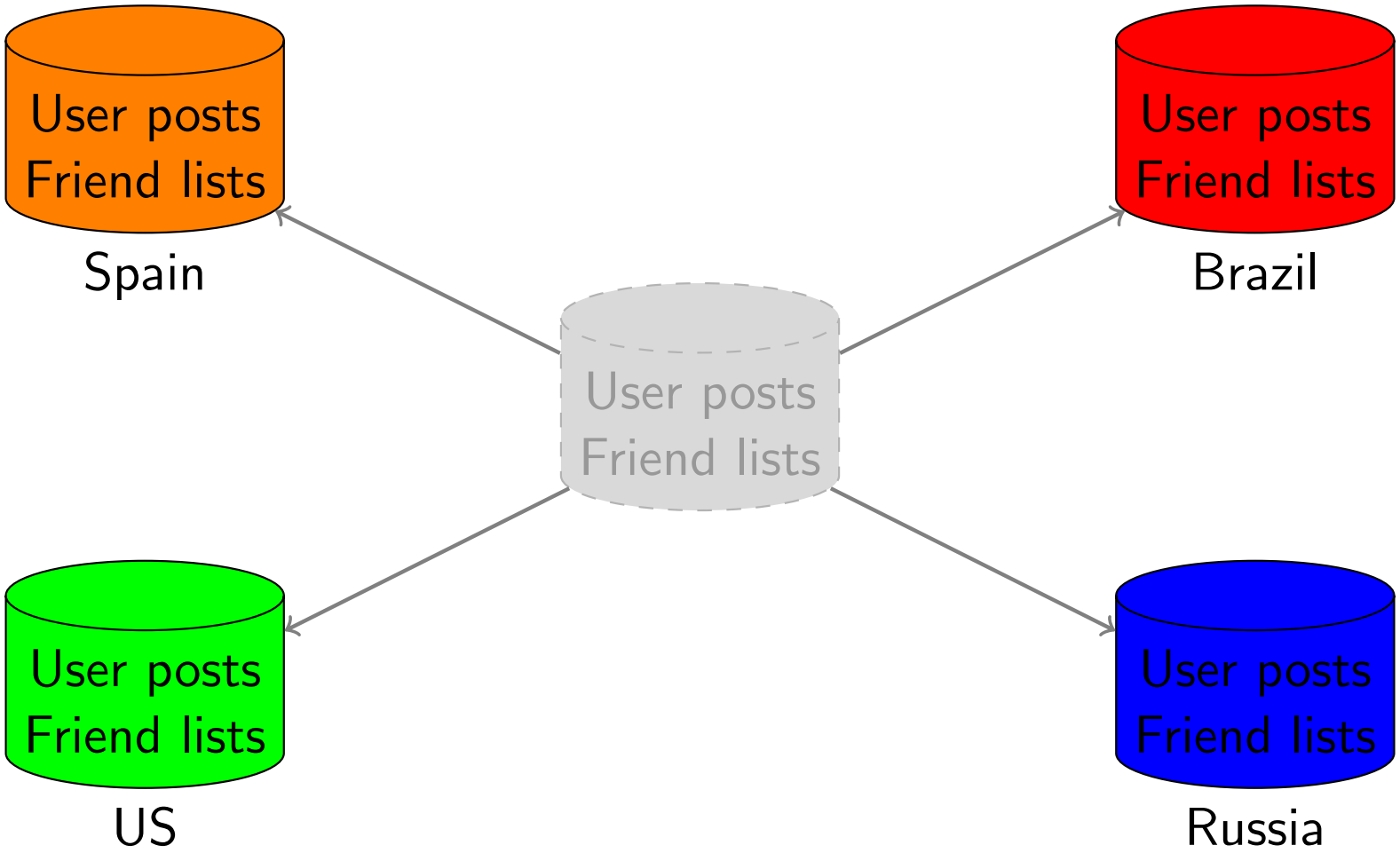
Features *(Cntd.)*

- SQL based query language
- External consistency of distributed transactions
  - $T(e_{1commit}) < T(e_{2start}) \implies s_1 < s_2$
  - first system at global scale
- Lock free distributed read transactions

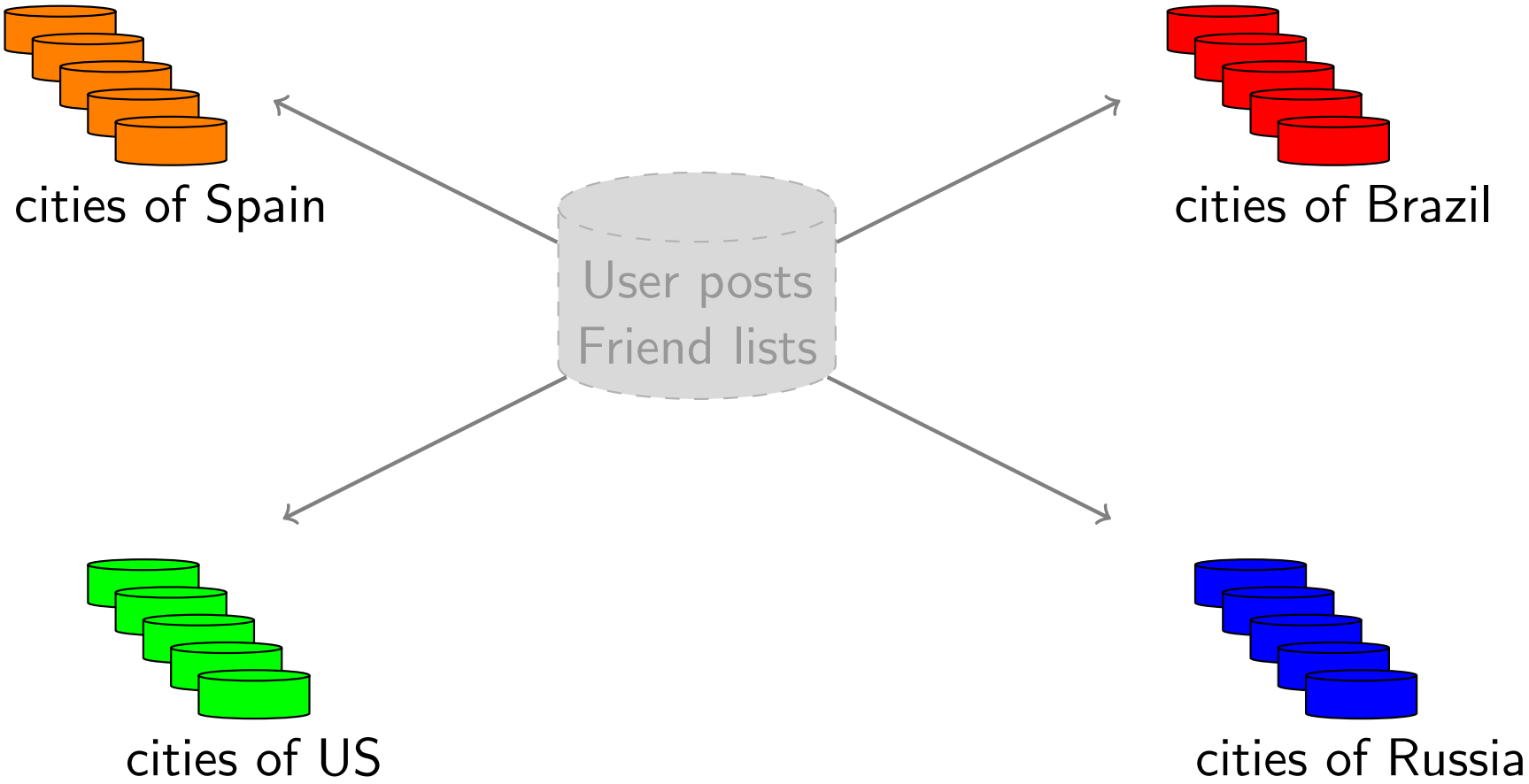
Case Study: Social Network



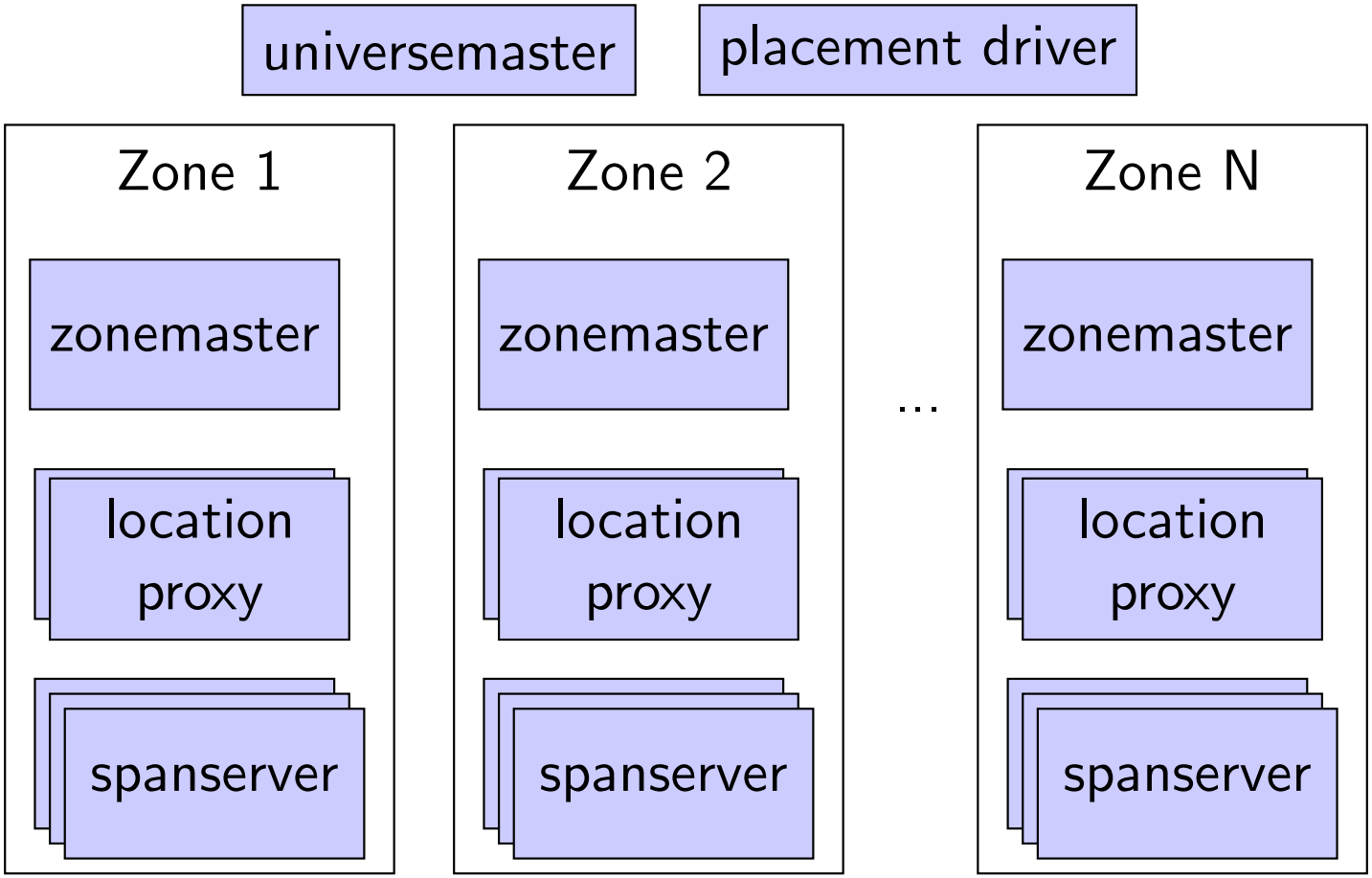
Case Study: Social Network *(Cntd.)*



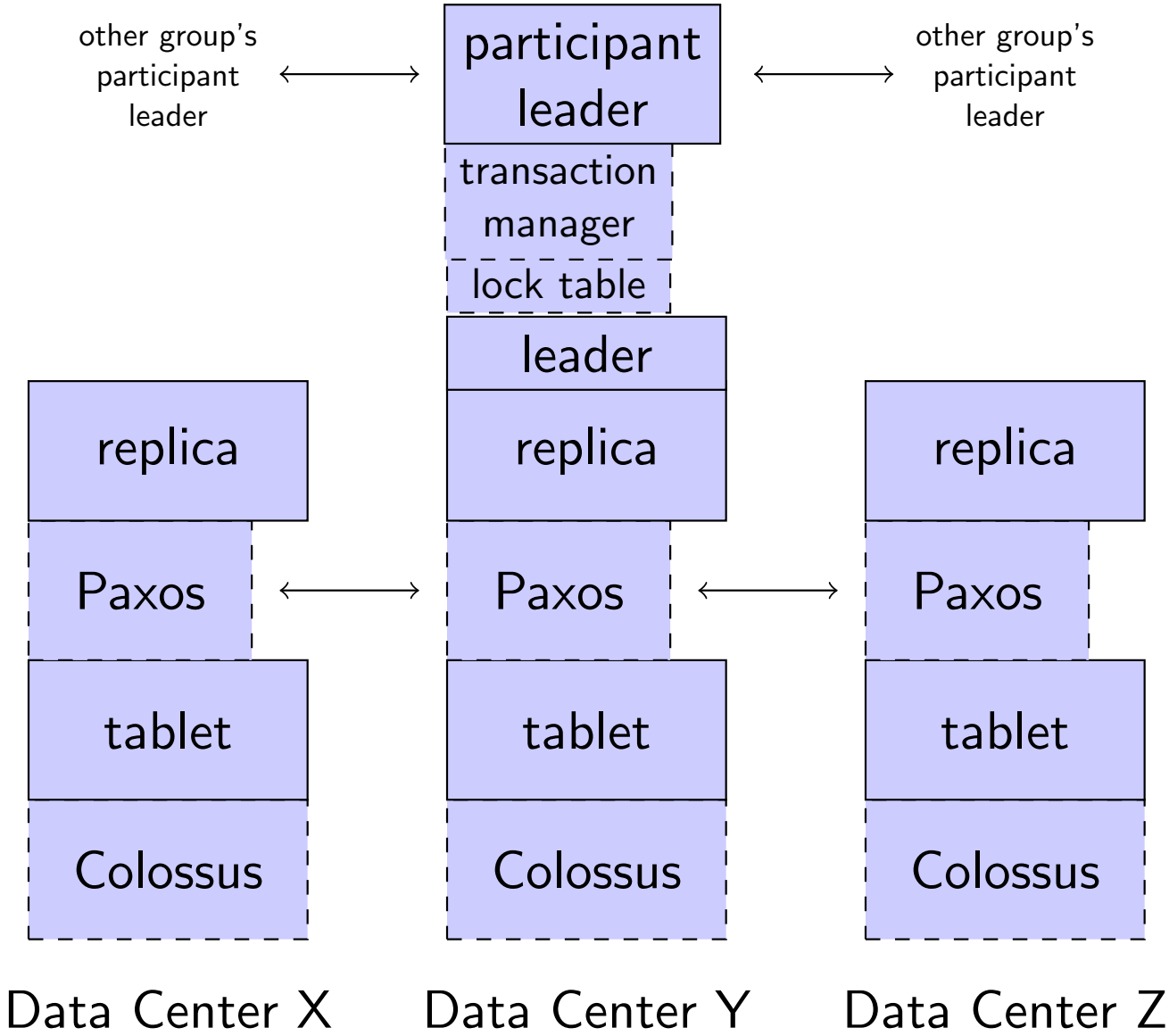
Case Study: Social Network *(Cntd.)*



Spanner Server Organization



Spanserver Software Stack



Data Model

- (key:string, timestamp:int64) $\rightarrow$ string
- Directory:
  - set of contiguous keys that share a common prefix
  - unit of data placement and movement
  - might be reduced to fragments

Read Transactions

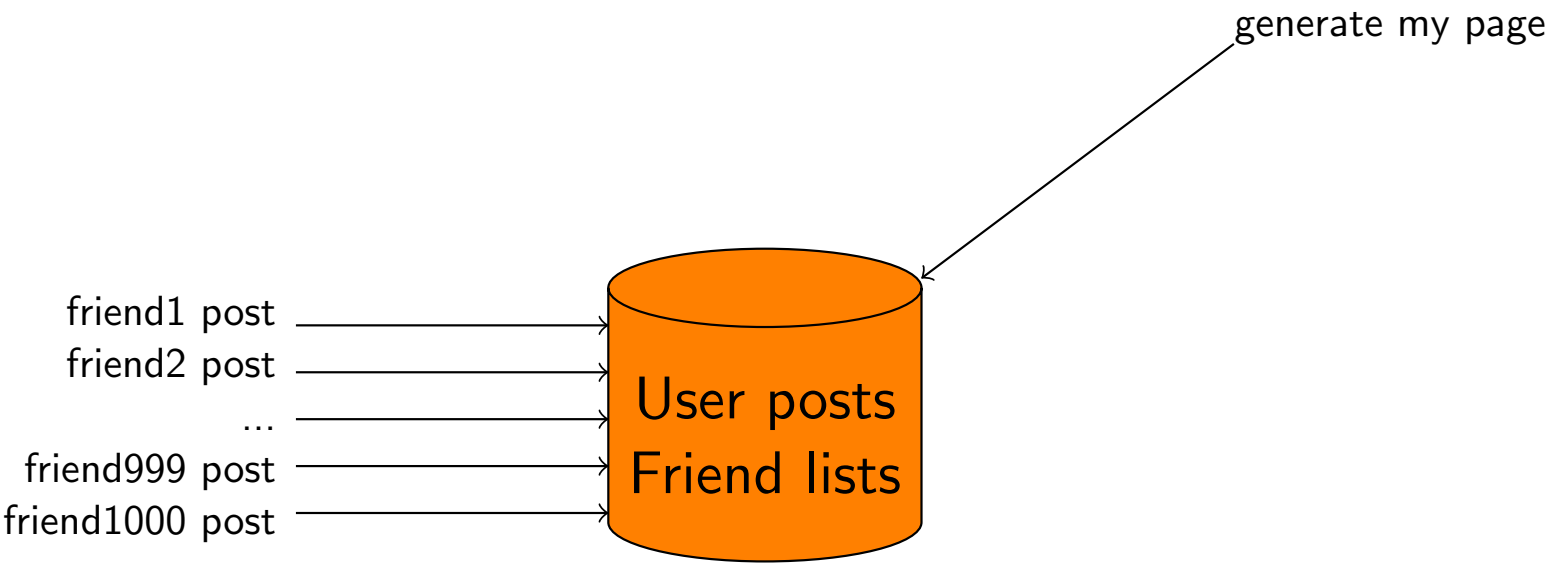
Why consistency matters?

- Generate a page of friends' recent posts
  - Consistent view of friend list and their posts

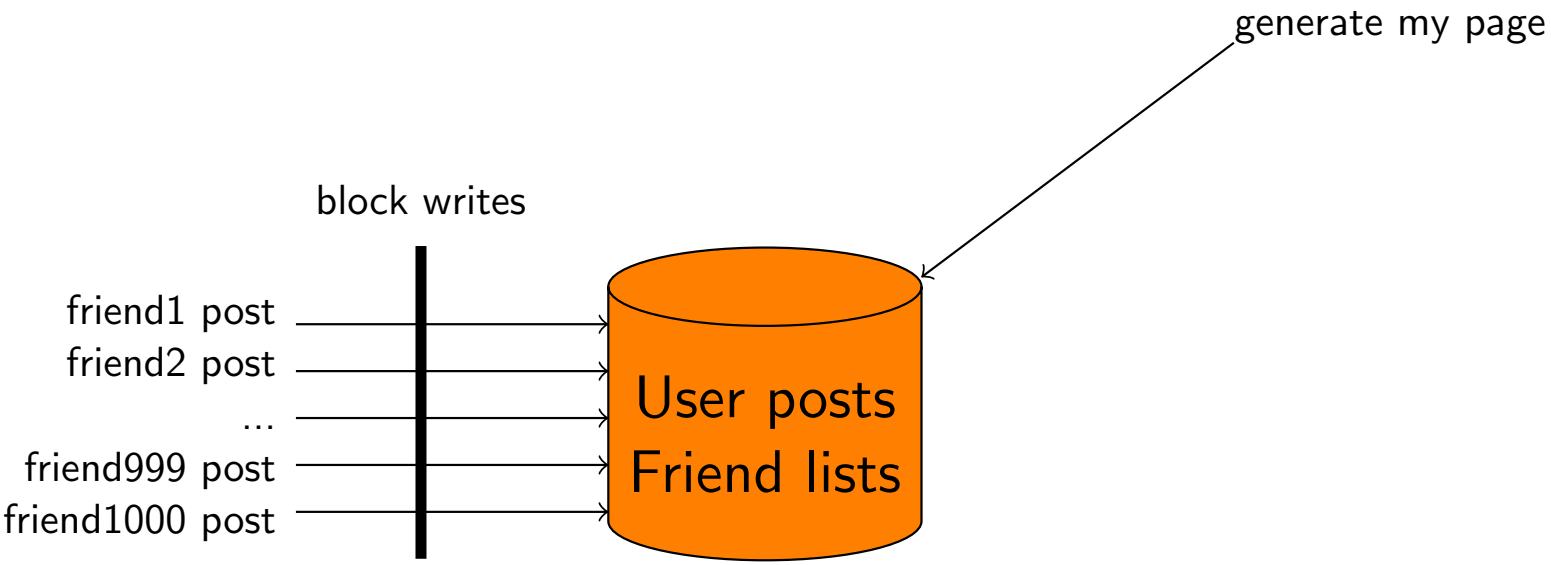
Why Consistency Matters

1. Remove untrustworthy person X as friend
2. Post P: "My government is repressive"

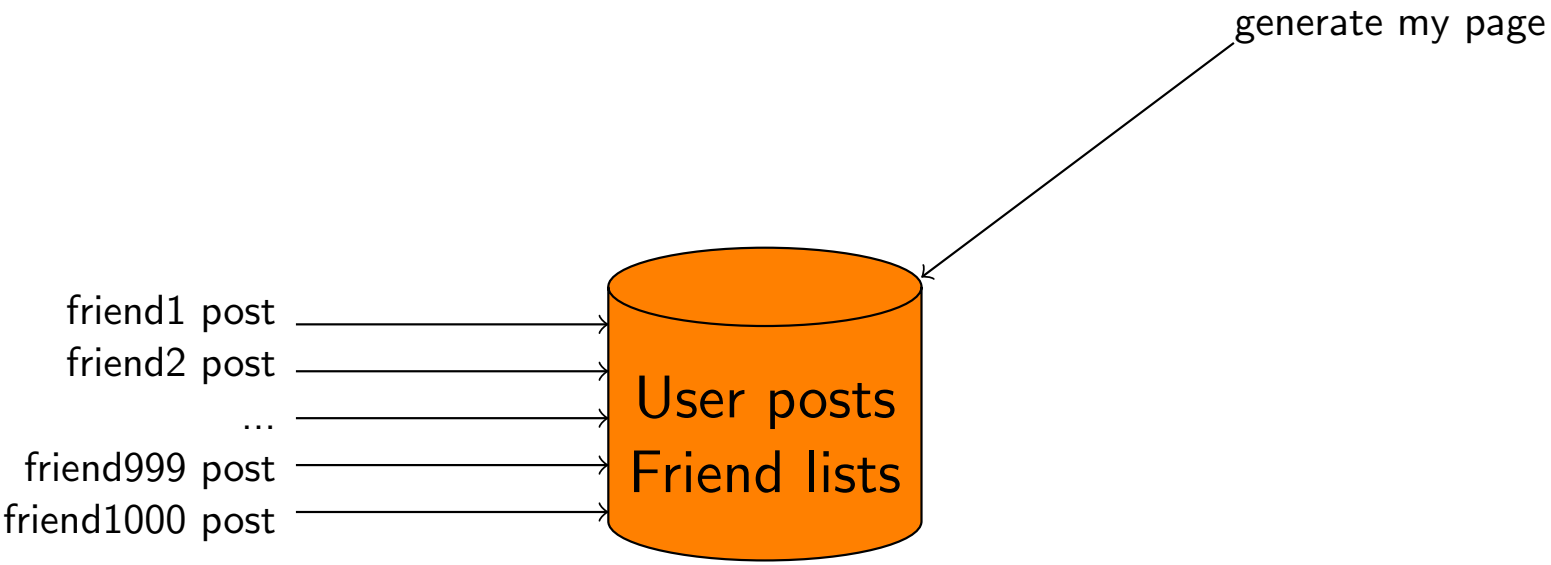
Single Machine



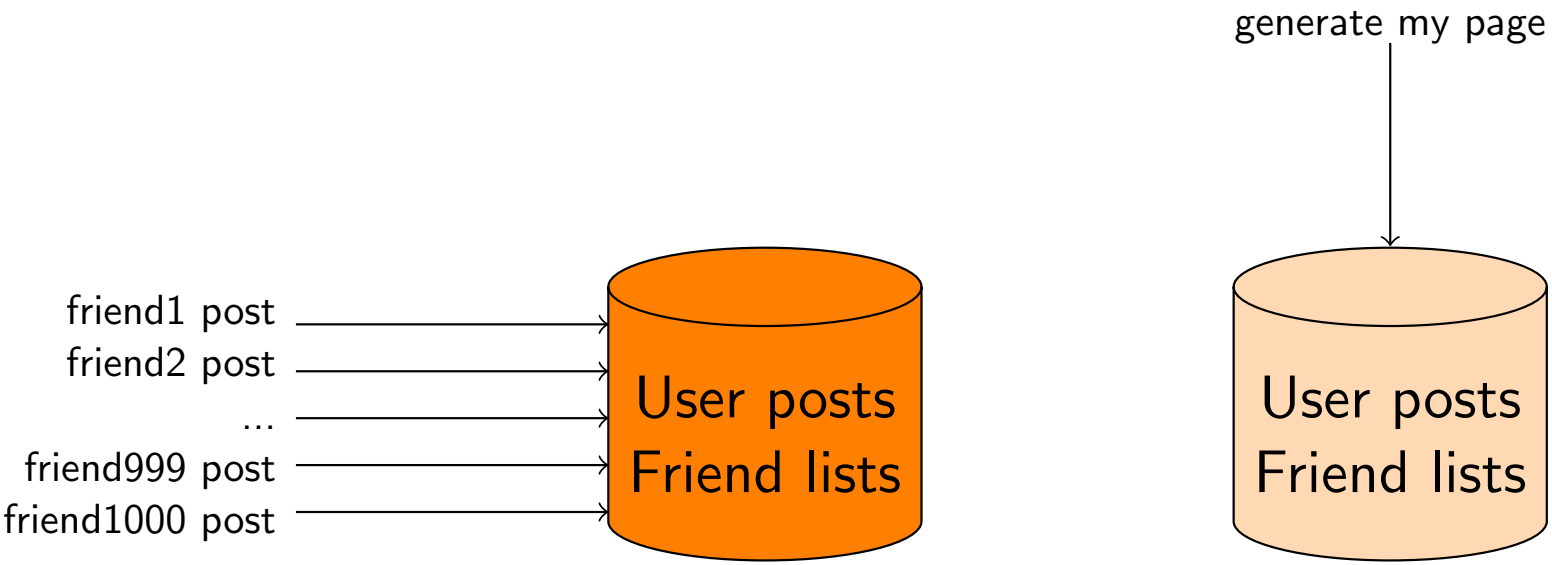
Single Machine *(Cntd.)*



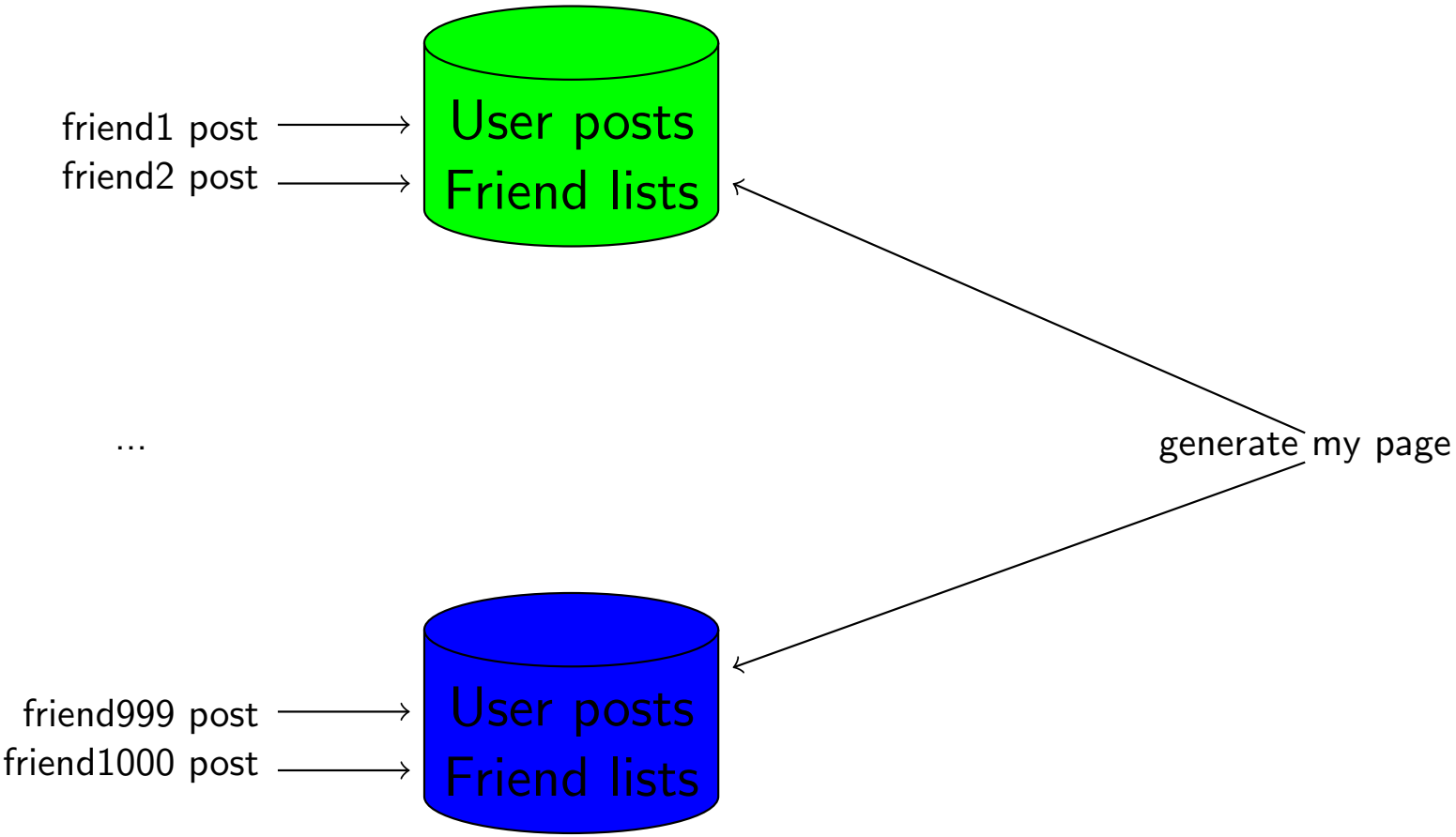
Single Machine *(Cntd.)*



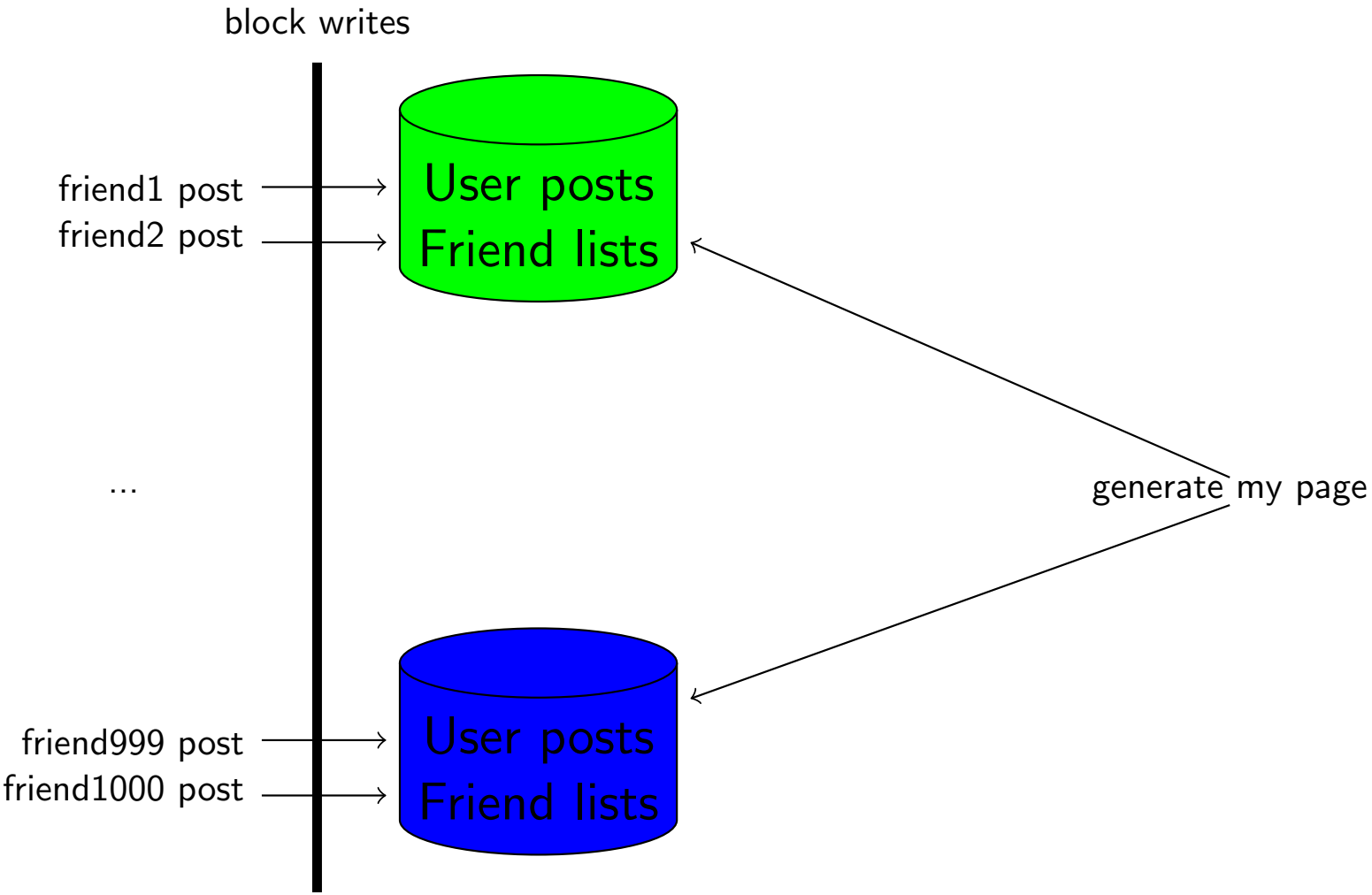
Single Machine *(Cntd.)*



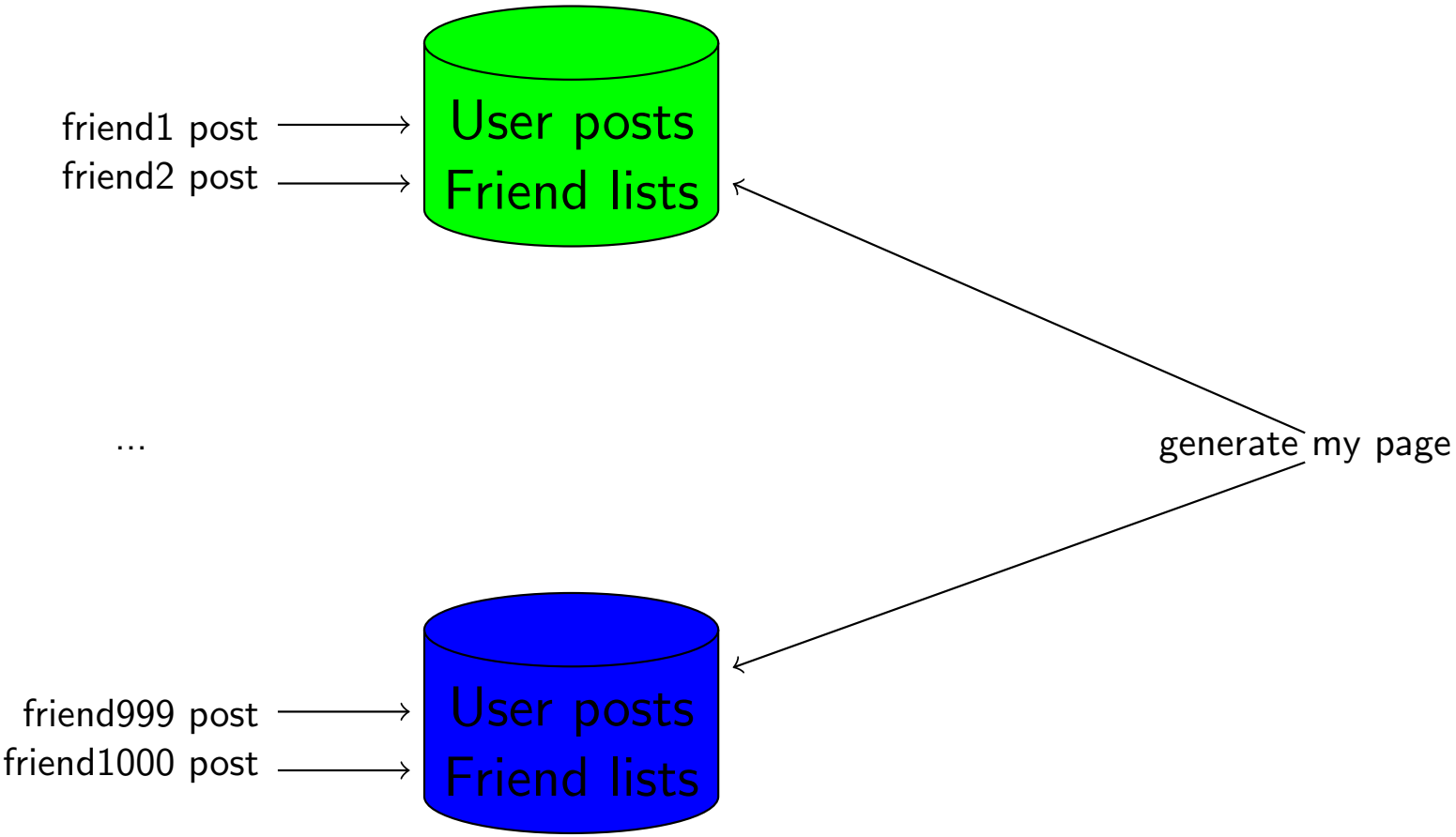
Multiple Machines



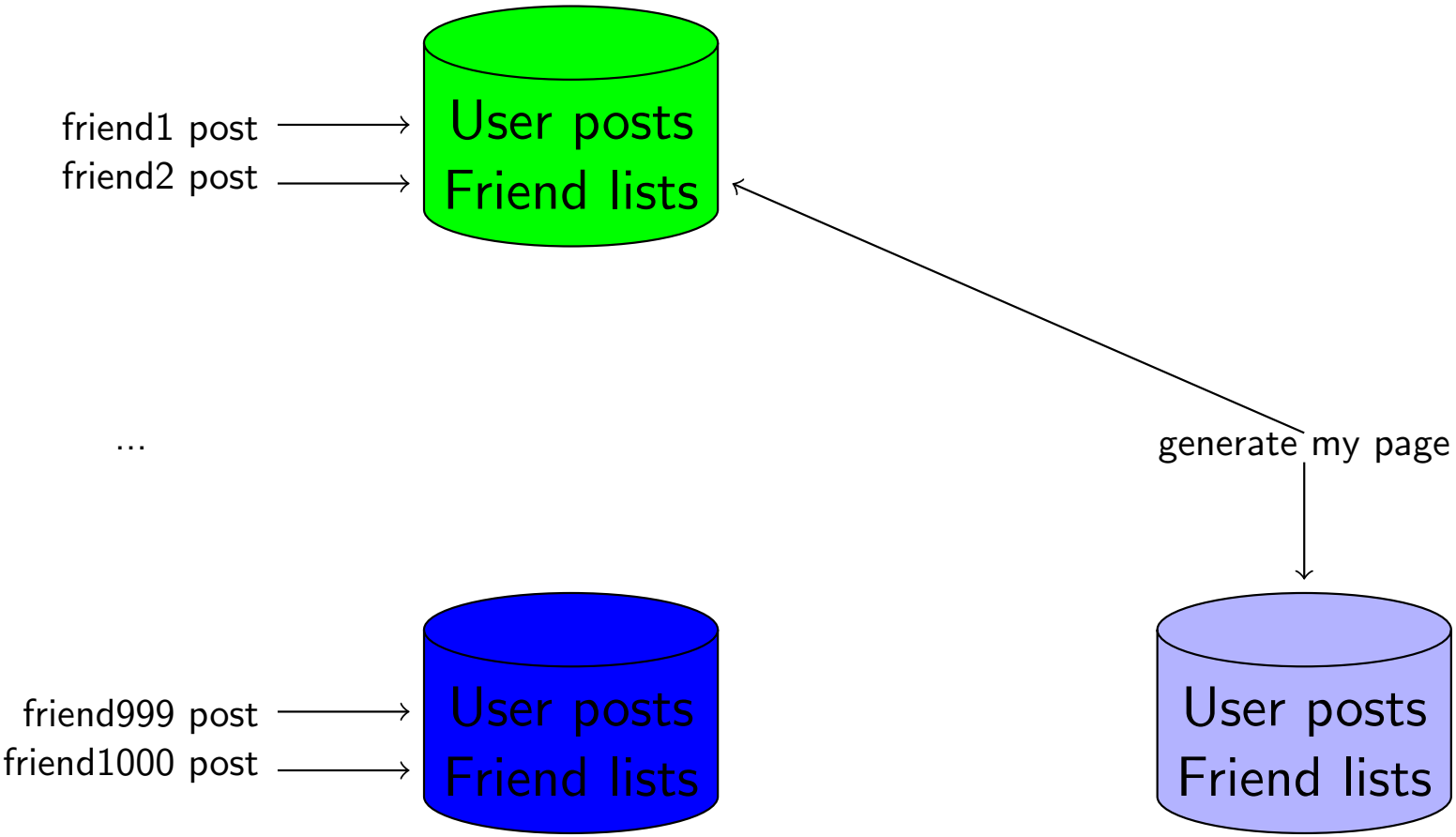
Multiple Machines *(Cntd.)*



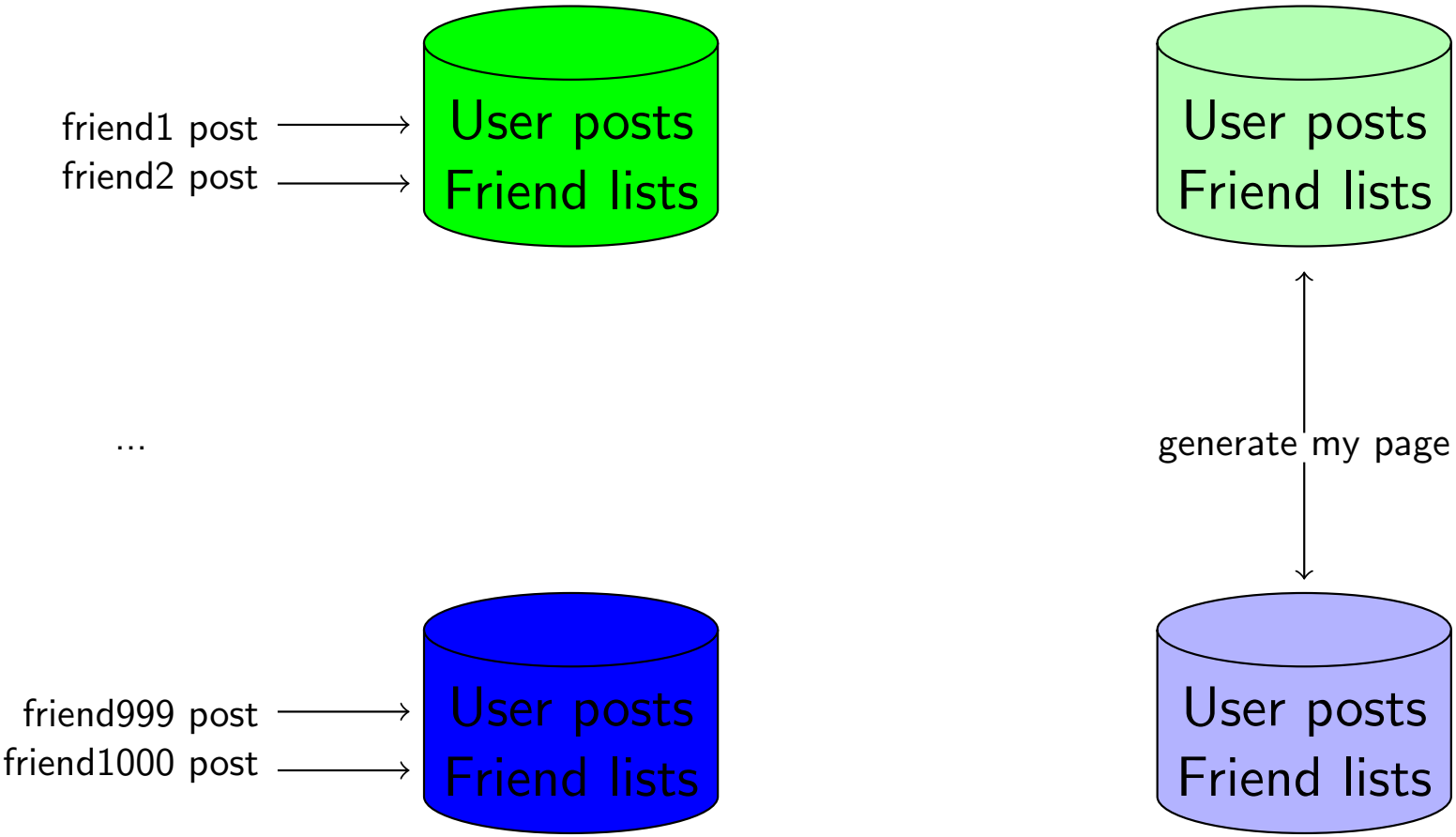
Multiple Machines *(Cntd.)*



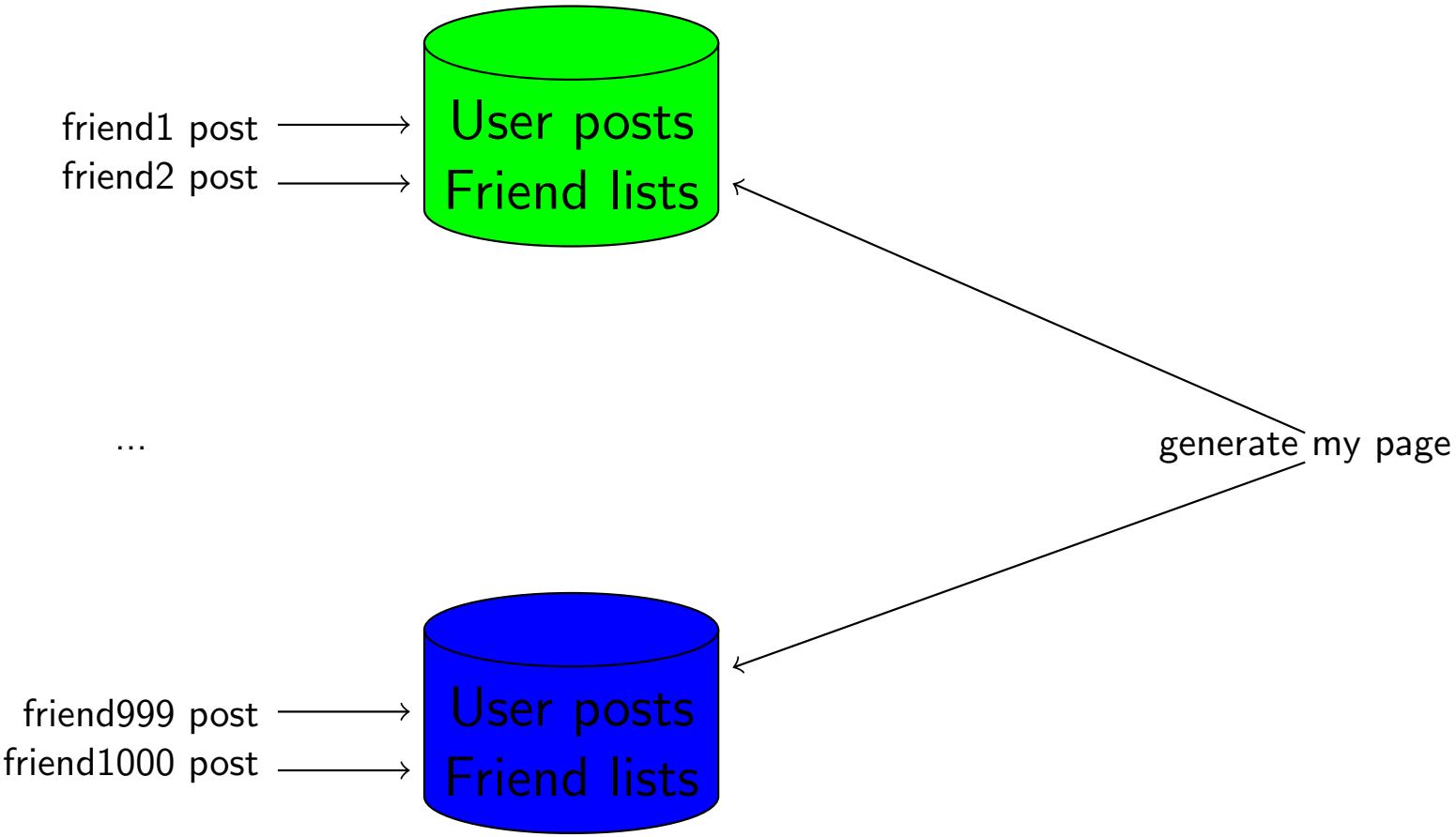
Multiple Machines *(Cntd.)*



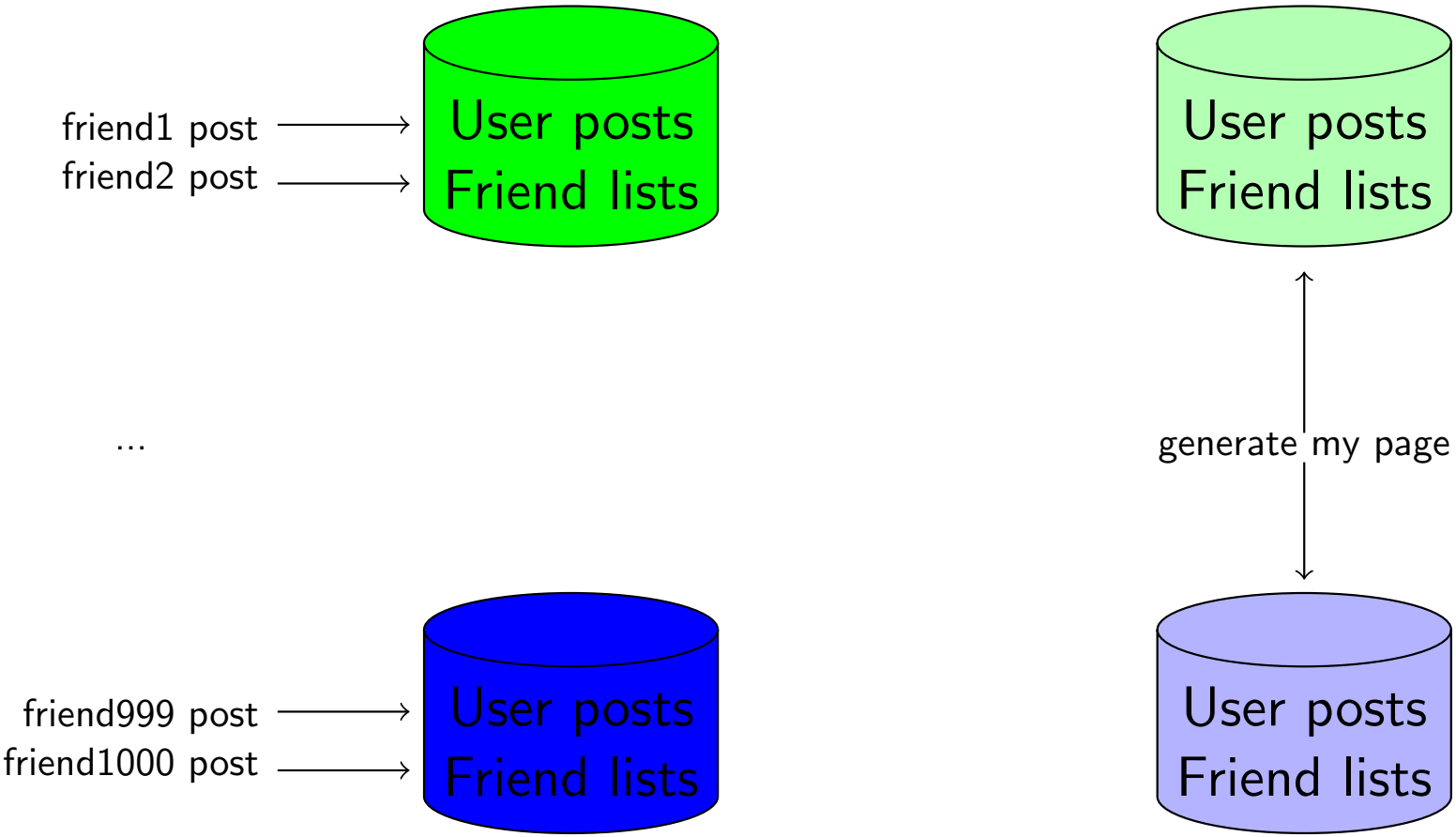
Multiple Machines *(Cntd.)*



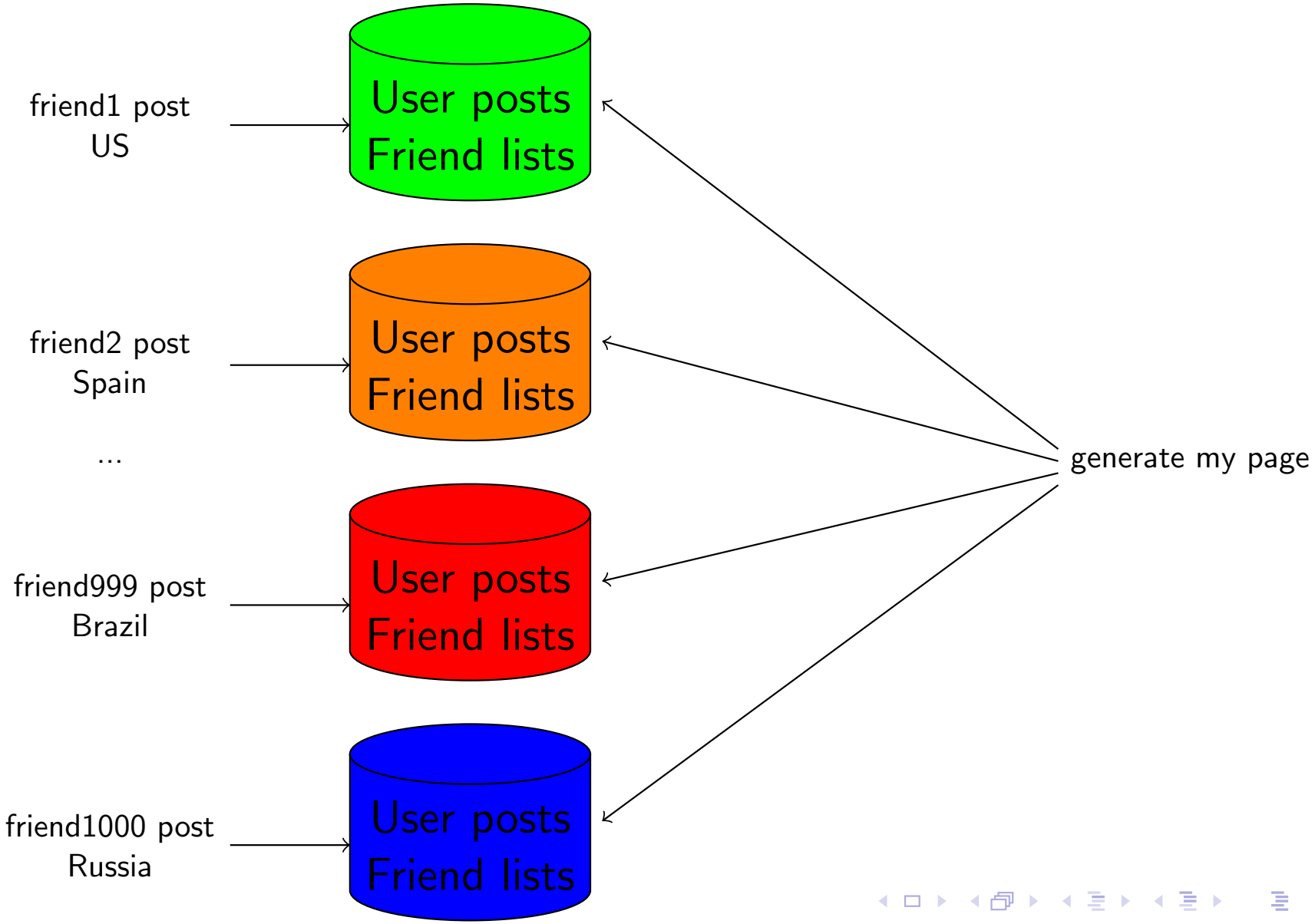
Multiple Machines *(Cntd.)*



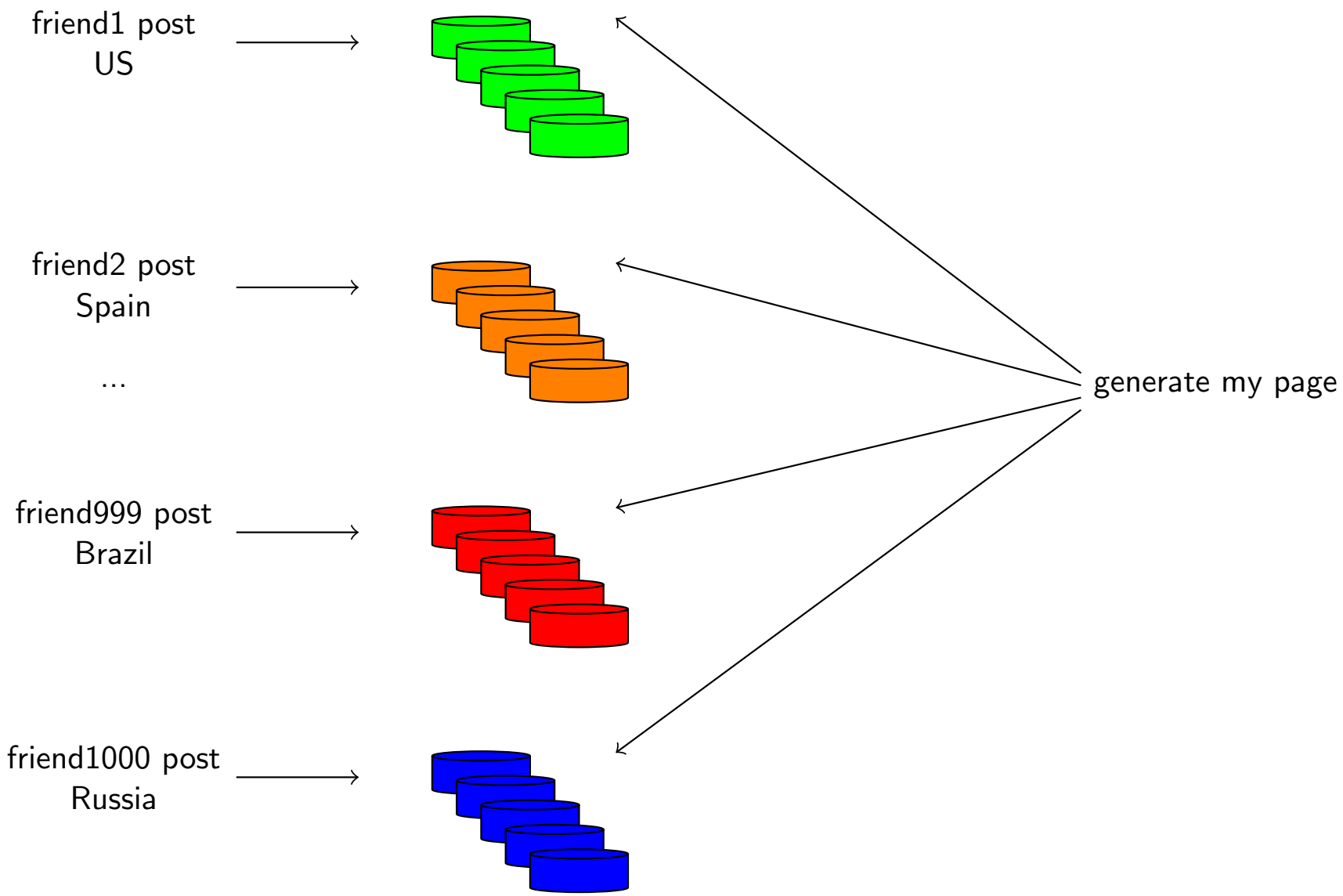
Multiple Machines *(Cntd.)*



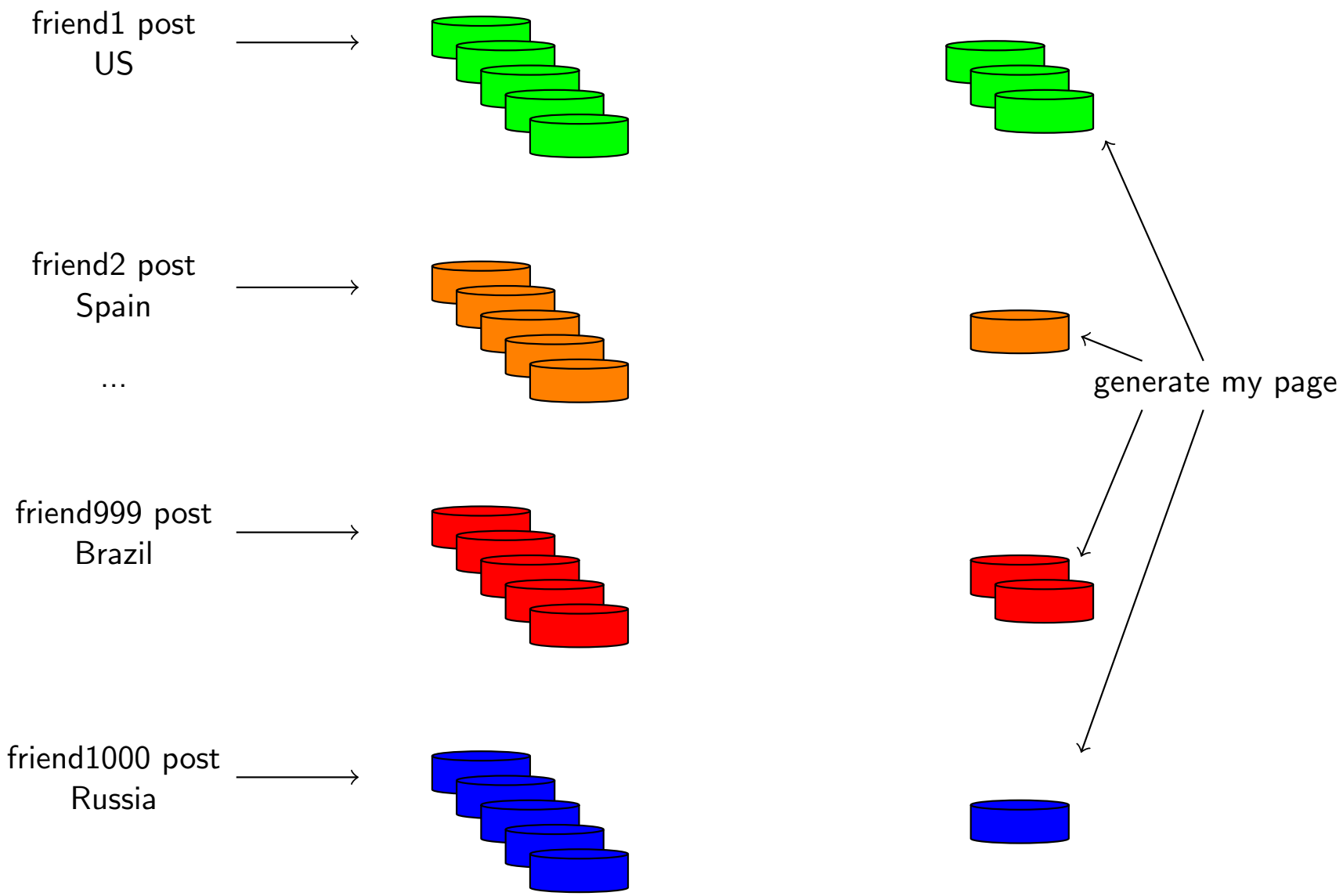
Multiple Datacenters



Multiple Datacenters *(Cntd.)*



Multiple Datacenters *(Cntd.)*



Version Management

- Transactions that write use strict 2PL
  - Each transaction T is assigned a timestamp s
  - Data written by T is timestamped with s

time	<8	8	15
my friends	[X]	[ ]	
my posts			[P]
X's friends	[me]	[ ]	

Synchronizing Snapshots

global wall-clock time

==

external consistency:
commit order respects global wall-time order

==

timestamp order respects global wall-time order
given
timestamp order == commit order

Synchronizing Snapshots

global wall-clock time

==

external consistency:
commit order respects global wall-time order

==

timestamp order respects global wall-time order
given
timestamp order == commit order

Synchronizing Snapshots

global wall-clock time

==

external consistency:
commit order respects global wall-time order

==

timestamp order respects global wall-time order
given
timestamp order == commit order

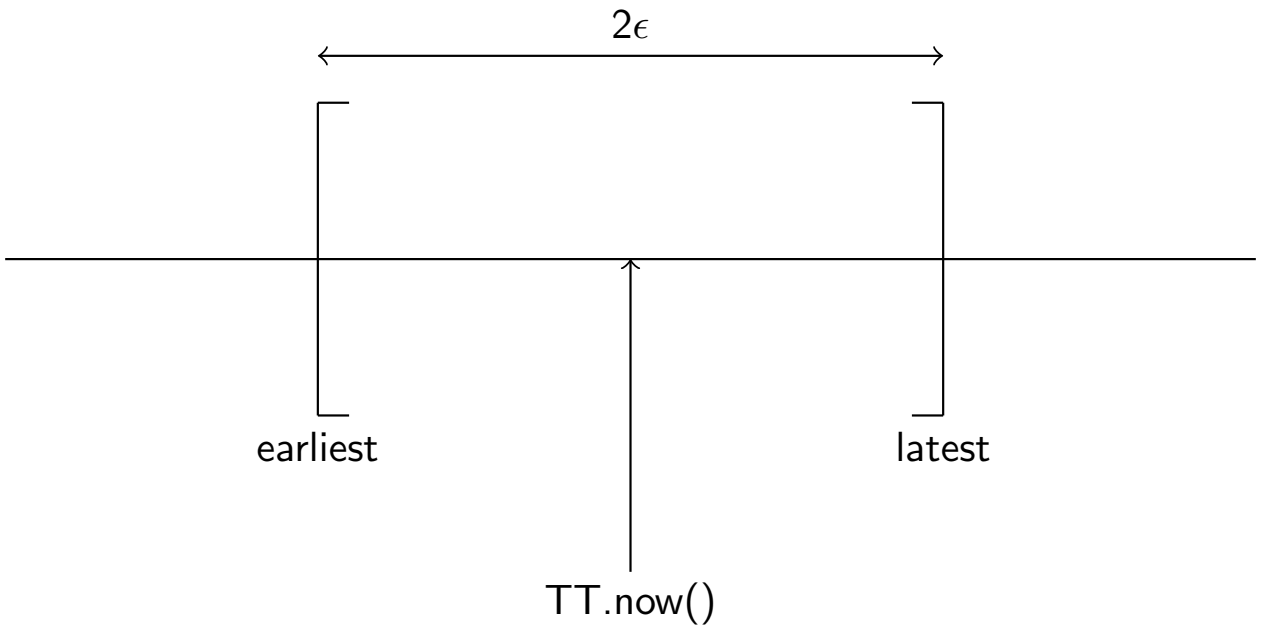
TrueTime

- Represents time as a TTinterval
- GPS and atomic clocks
- Marzullo's algorithm
- Slowly increasing time uncertainty ϵ (sawtooth function of time)

Known unknowns are better than unknown unknowns

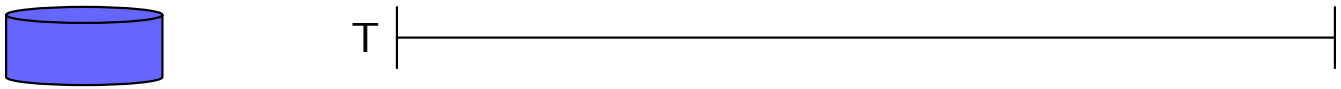
TT.Now()

- "global wall-clock time" with bounded uncertainty



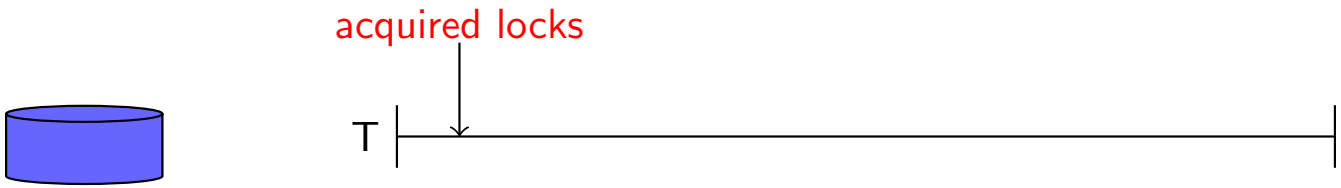
Timestamps and Global Clock

- Strict two phase locking for write transactions
- Assign timestamp while locks are held



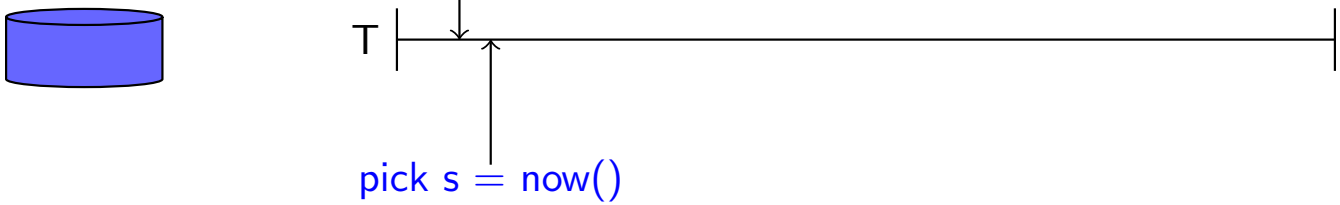
Timestamps and Global Clock *(Cntd.)*

- Strict two phase locking for write transactions
- Assign timestamp while locks are held



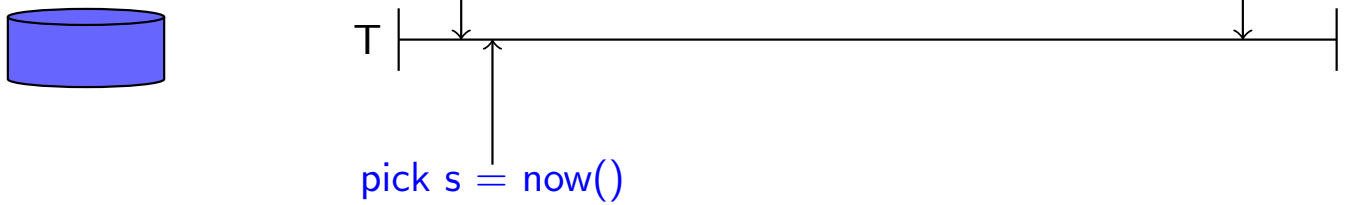
Timestamps and Global Clock *(Cntd.)*

- Strict two phase locking for write transactions
- Assign timestamp while locks are held



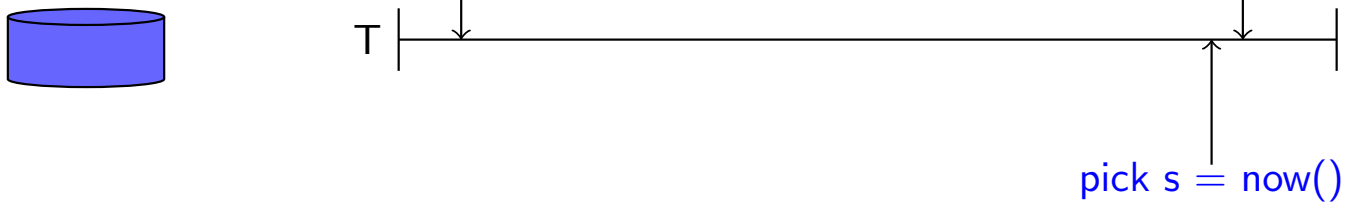
Timestamps and Global Clock *(Cntd.)*

- Strict two phase locking for write transactions
- Assign timestamp while locks are held



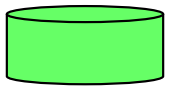
Timestamps and Global Clock *(Cntd.)*

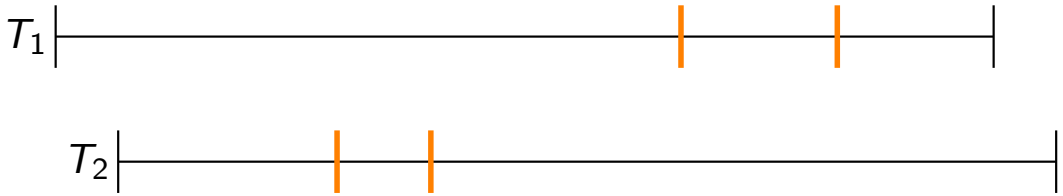
- Strict two phase locking for write transactions
- Assign timestamp while locks are held



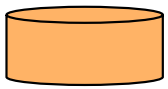
Timestamp Invariants

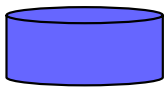
- timestamp order == commit order



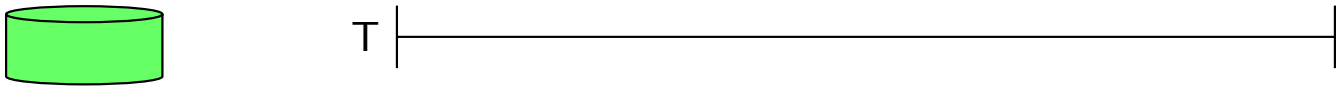


- timestamp order respects global wall-time order

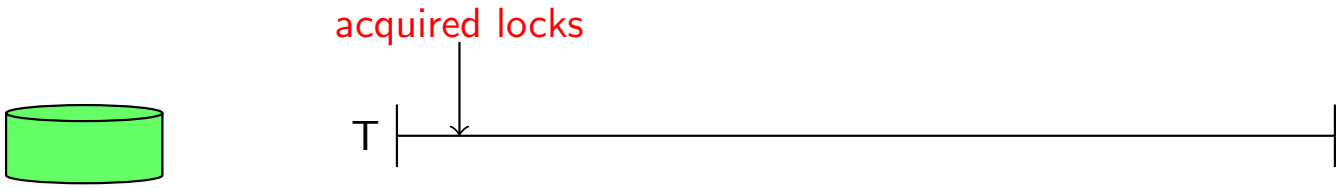




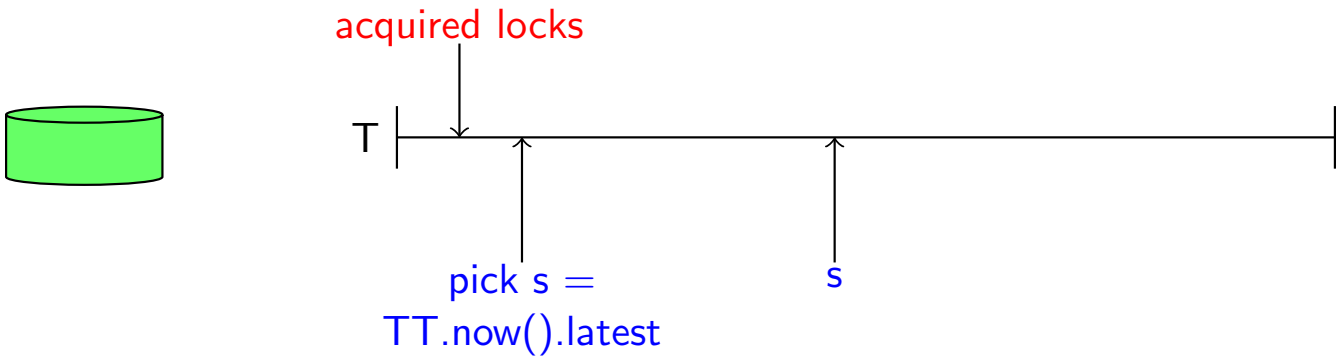
Timestamps and TrueTime



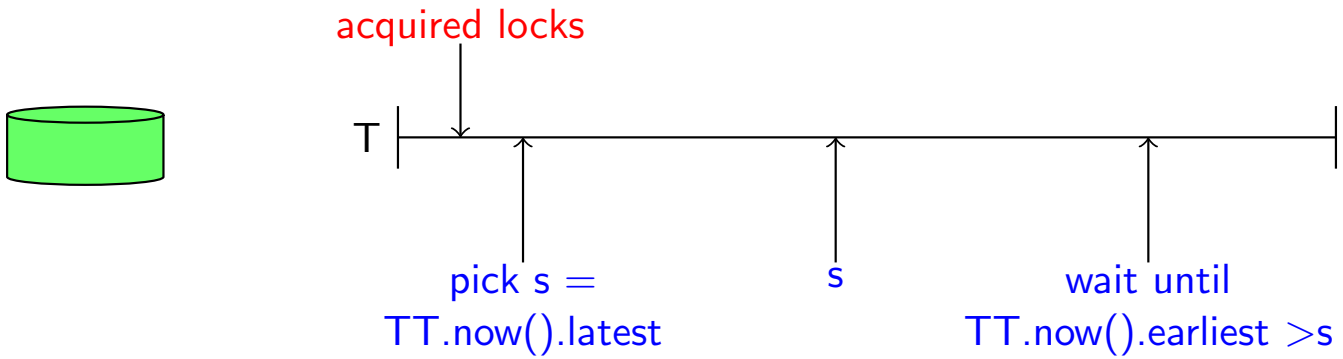
Timestamps and TrueTime *(Cntd.)*



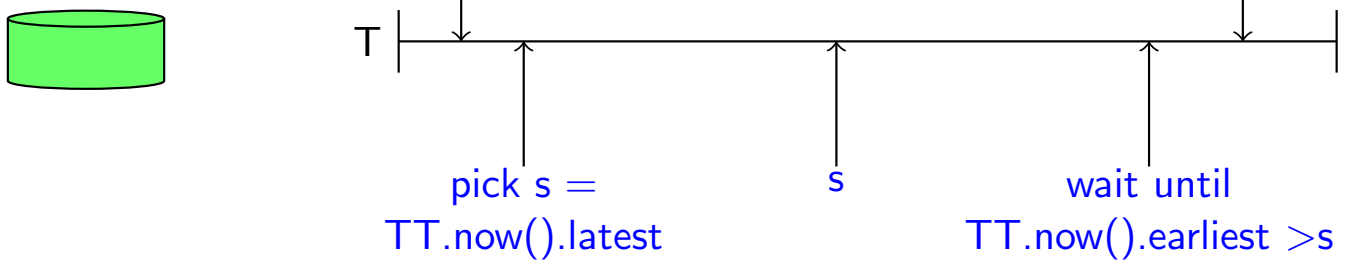
Timestamps and TrueTime *(Cntd.)*



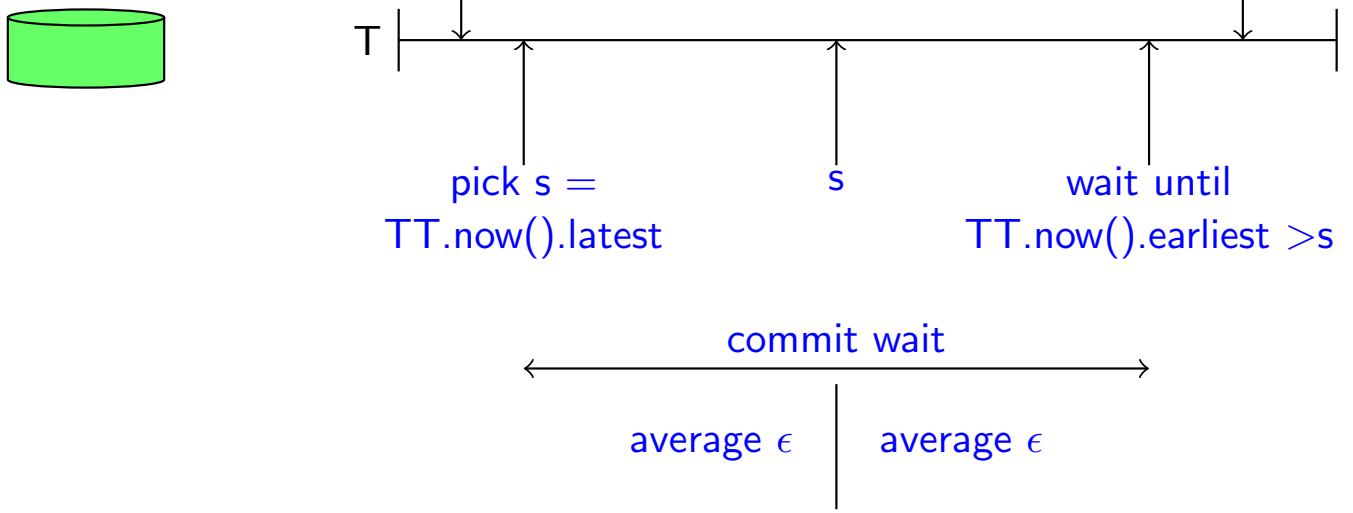
Timestamps and TrueTime *(Cntd.)*



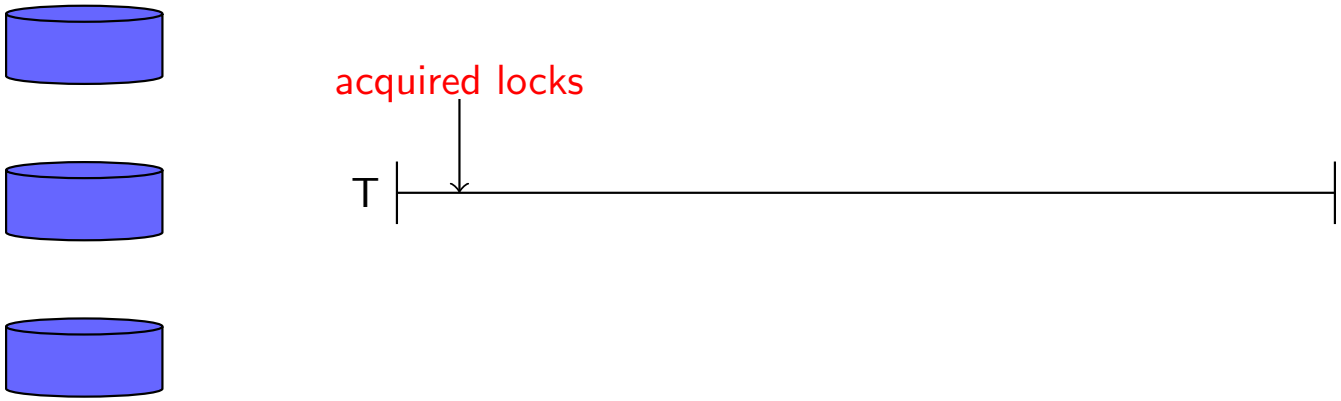
Timestamps and TrueTime *(Cntd.)*



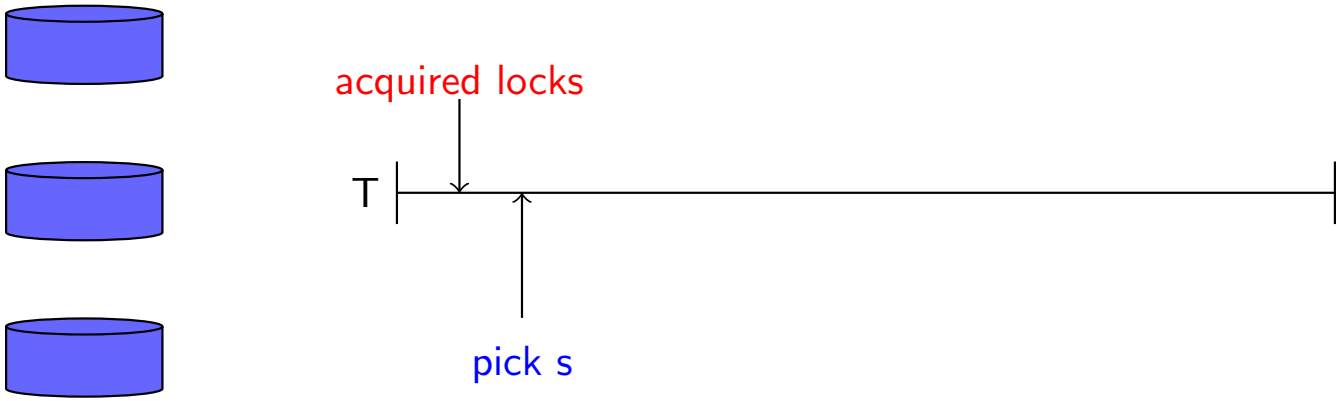
Timestamps and TrueTime *(Cntd.)*



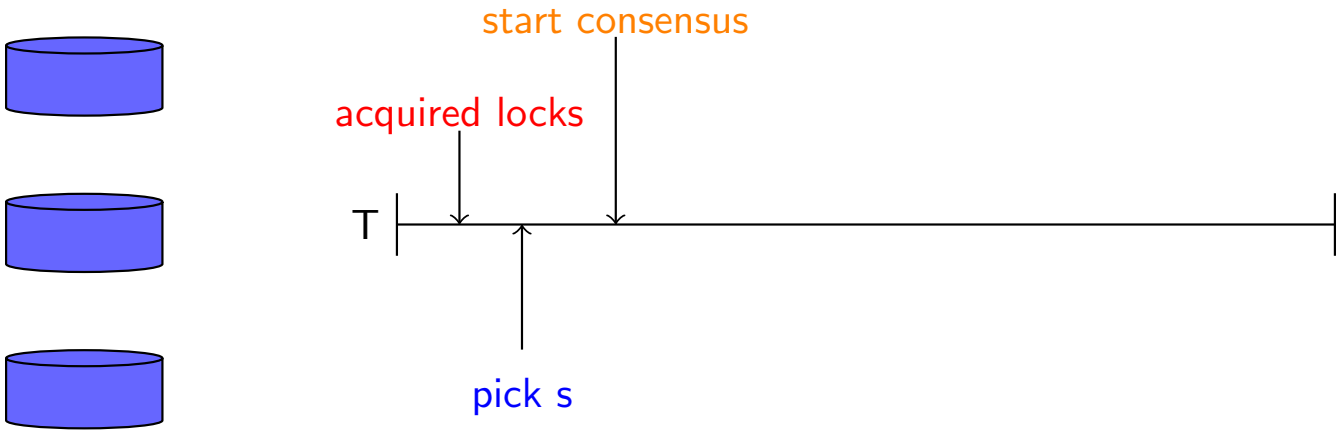
Commit Wait and Replication



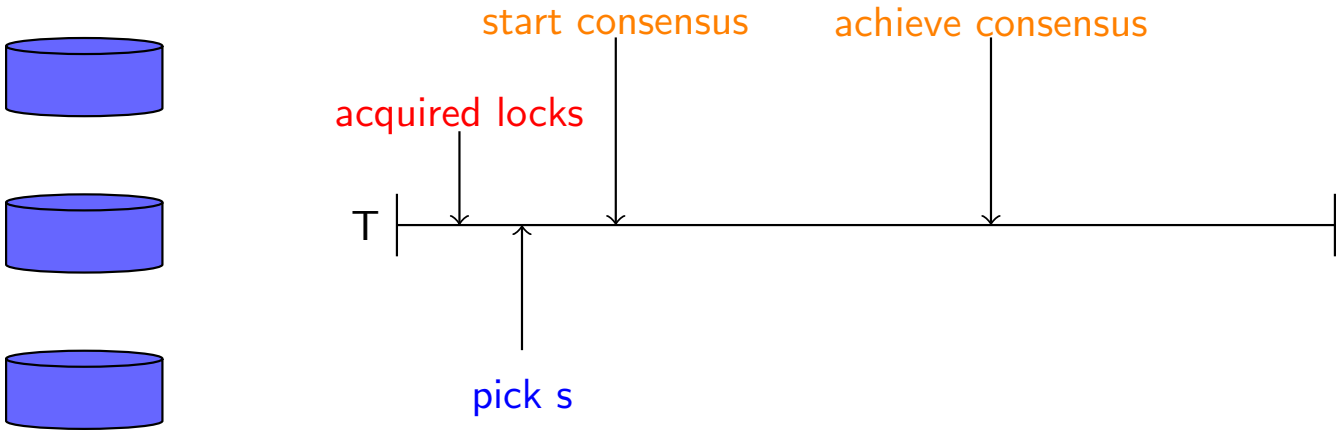
Commit Wait and Replication *(Cntd.)*



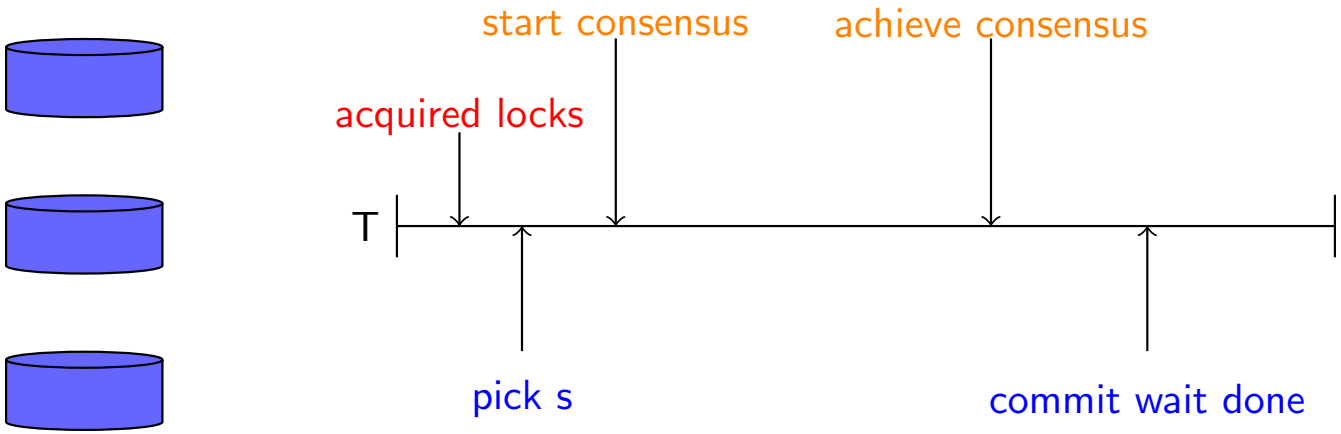
Commit Wait and Replication *(Cntd.)*



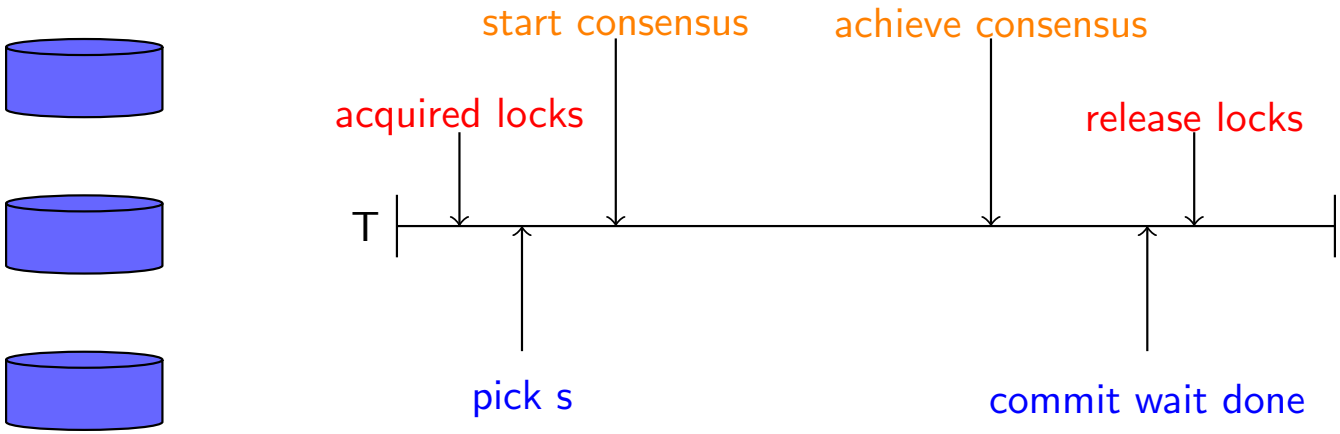
Commit Wait and Replication *(Cntd.)*



Commit Wait and Replication *(Cntd.)*



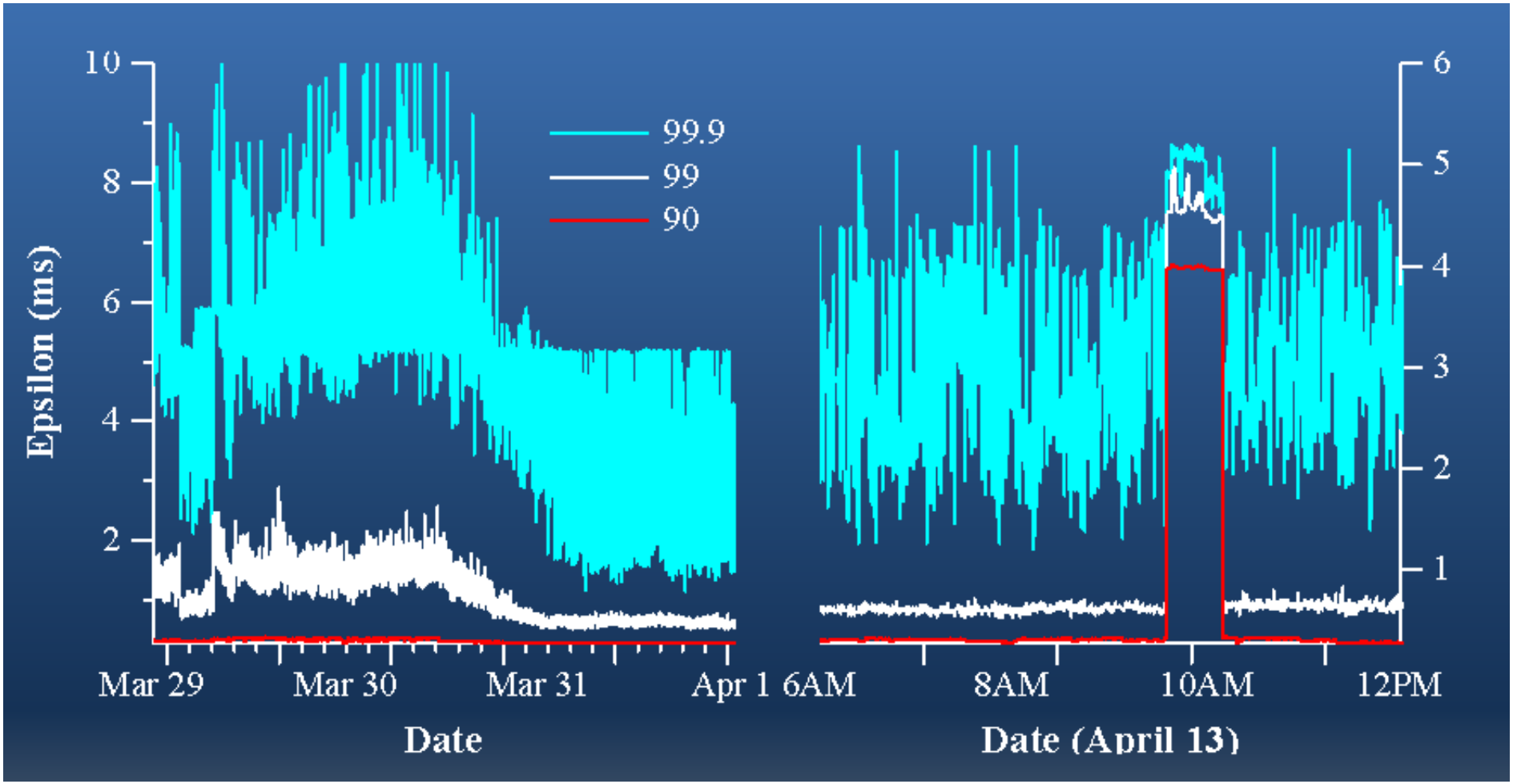
Commit Wait and Replication *(Cntd.)*



What if a Clock goes Rogue

- Timestamp assignment would violate external consistency
- Empirically unlikely based on 1 year of data
  - bad CPUs 6 times more likely than bad clocks

Network-Induced Uncertainty



What's in the Literature

- External consistency/linearizability
- Distributed database
- Concurrency control
- Replication
- Time (NTP, Marzullo)

Future Work

- Improving TrueTime
  - lower ϵ $<1ms$
- Building out database features
  - finish implementing basic features
  - efficiently support rich query patterns

Questions?