UNIVERSITY of VIRGINIA

CS4740
CLOUD COMPUTING

Consensus

Prof. Chang Lou, UVA CS, Fall 2025

CONTEXT

- We learned how to achieve atomicity, isolation in a sharded database.

- Today we learn how to achieve fault tolerance through replication. Problem of maintaining multiple replicated shards can ultimately be reduced to consensus.

- We discuss Paxos and Raft, the best known consensus protocols.

GAME: CONSENSUS!

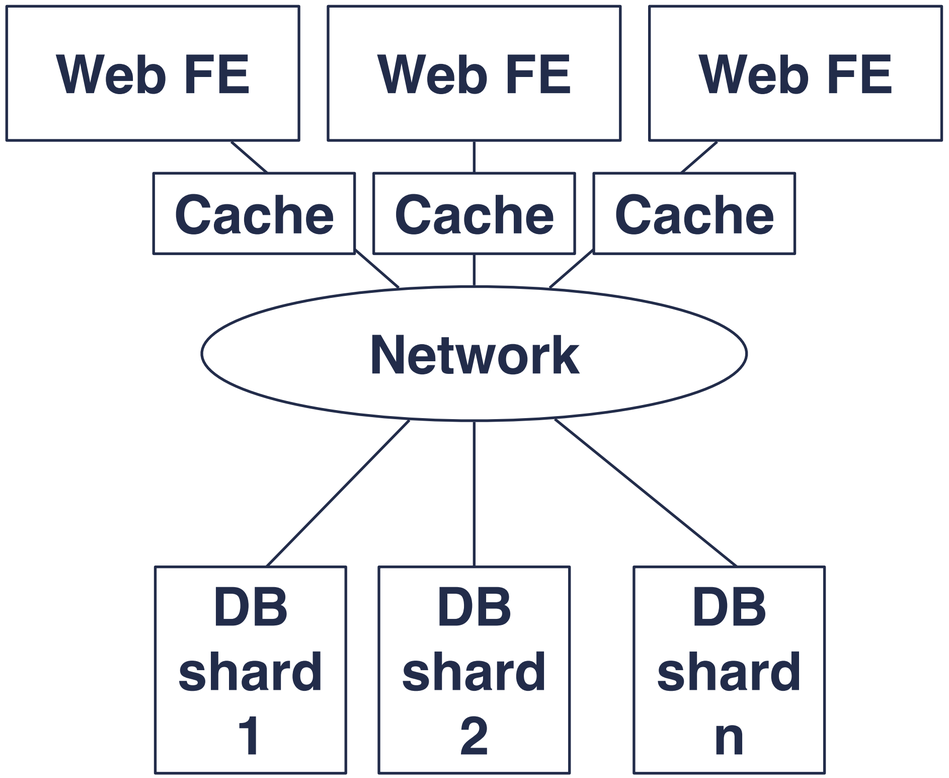
- Each student votes on an integer between 1 - 100.
  - Can only vote once, repeated votes become invalid.
- **Win**: if the majority of voters have chosen the same number, everyone in the quorum gets extra credits in the final.
- **Lose**: no quorum reached.
- **Extra**: before checking results, Chang can void a number.
  - Shouting out a number may not sound like a good idea now?
- You have 5 min to discuss a strategy.
- Submit your vote to https://shorturl.at/4lg1l

OUTLINE

- Problem: Replicating ACID shards
- Mock protocol with 2PC
- Consensus protocols
- Raft Visuals

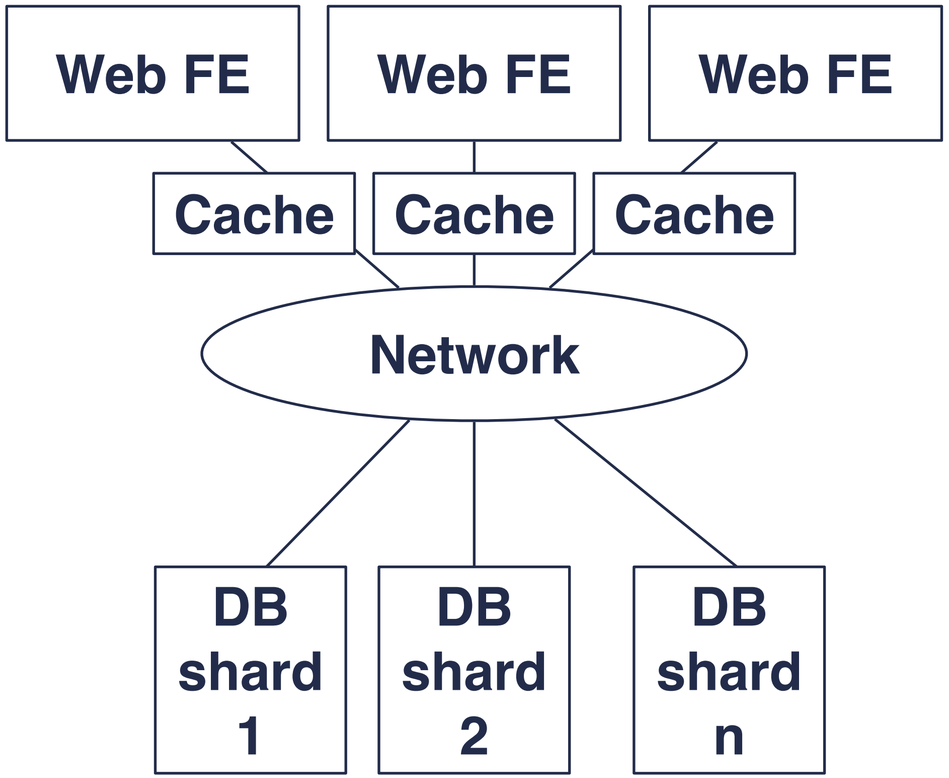
Problem: Replicating ACID shards

EXAMPLE: WEB SERVICE WITH TRANSACTIONS



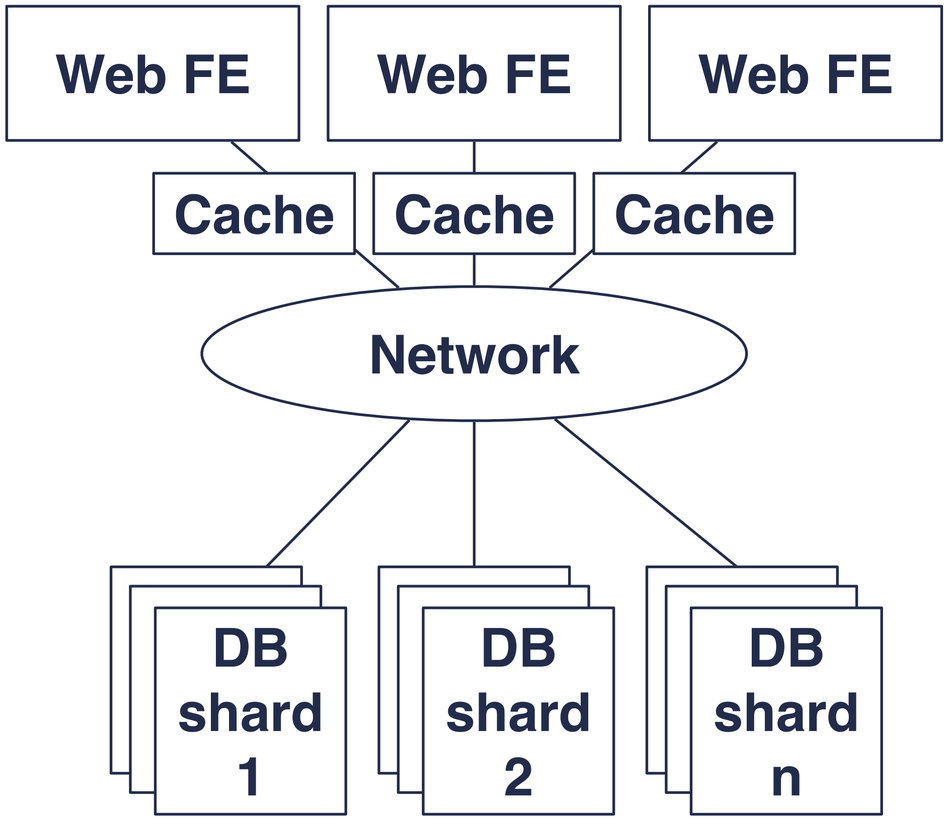
- Assume: sharded database. Each DB shard runs an ACID engine (so runs 2PL+WAL). The shards coordinate via 2PC.
- Question: Without shard replication, what fault tolerance problems can arise?

EXAMPLE: WEB SERVICE WITH TRANSACTIONS



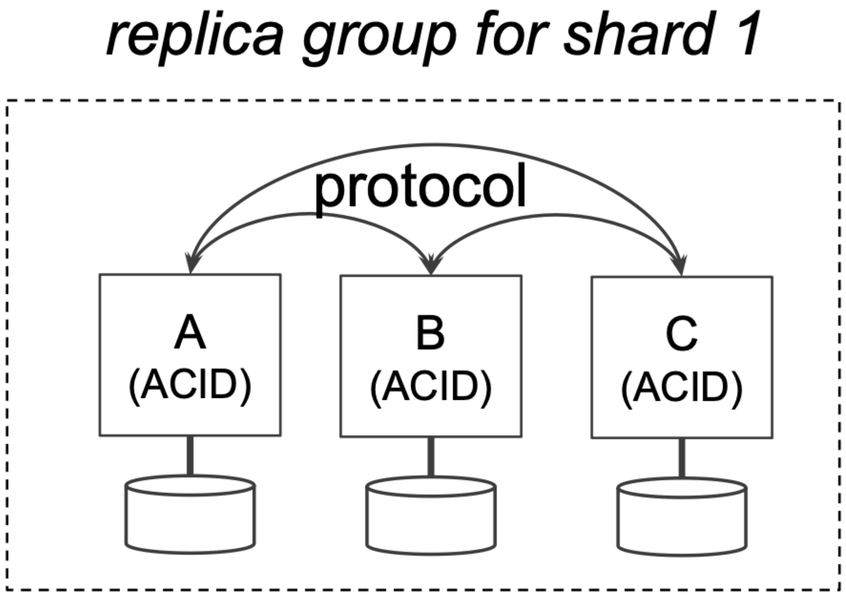
- Fault tolerance problems w/o replication:
  - Data, WAL for each shard are stored on one disk. If disk dies, shard's data is lost. **Durability** problem!
  - Even if disks don't fail, recall that 2PC can block if a shard server fails at inopportune time. Transactions interacting with the failed server block, along with many new transactions that transitively depend on rows locked by the blocked transactions. **Availability** problems!

SOLUTION: REPLICATION



- Architecture:
  - Replicate each shard across multiple servers (each ACID, so they maintain WAL and do 2PL).
  - Replicas of a shard coordinate to maintain their state “in sync,” ideally giving the illusion that they are a single, (almost) always-on server.
  - 2PC is executed across replica groups (we’ll discuss how in future lectures). Because replica groups “never” die or become partitioned, 2PC “never” blocks.

QUESTION 1: WHAT STATE TO REPLICATE?



- Disk image?
- In-memory image?
- Locks?
- ... Anything else?

BASIC ANSWER: REPLICATE WAL

— Claim: If all replicas execute all **WAL** ops, in the same order, then all other state (DB image, locks, ...) will be reconstructed in the same way across replicas (assuming deterministic operation).

— It can be useful to be able to push checkpoints of the DB to a recovering/new replica, but we'll ignore that for now and focus on replicating the WAL.

QUESTION 2: WHAT SEMANTIC TO REQUIRE?

- Requirement: all replicas apply (1) the same **log entries**, (2) in the **same order**.
- Otherwise, inconsistencies can occur.
- As examples, consider:
  - One replica skips log entry for an update while others apply it.
  - One replica receives two updates for a particular row in one order while another receives them reversed.

QUESTION 3: HOW TO REPLICATE?

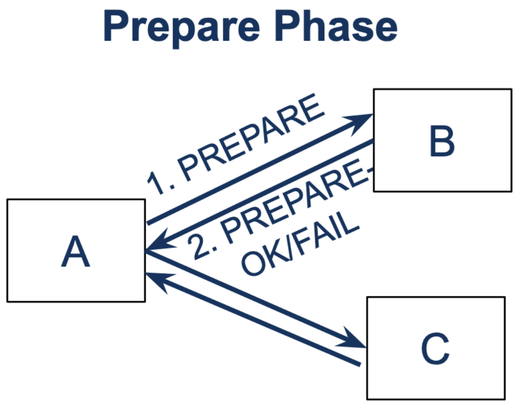
- Requirement: all replicas apply (1) the same log entries, (2) in the same order.

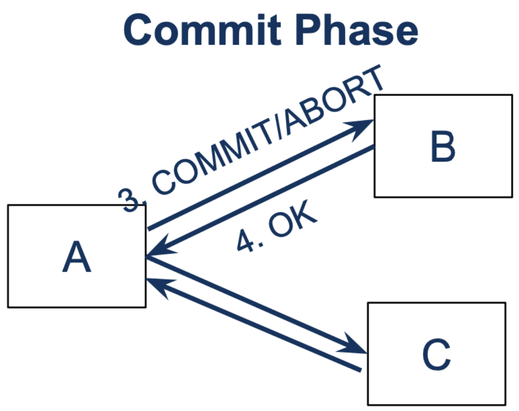
- One idea: 2PC.
  - 2PC ensures that all participants either do all ops or don’t do any of the operations.
  - Could we use this protocol for WAL replication?

Mock protocol based on 2PC

MOCK 2PC-BASED REPLICATION

- A, B, C are replicas of a single shard. They need to coordinate to apply all WAL entries in the same order.
- Q: how might it work and what problems would arise.





MOCK DESIGN

- One replica assigned as TC. TC decides on order of ops in the log and performs 2PC for each log entry, every time blocking for the protocol to finish before launching a 2PC for the next log entry.
- This ensures that all replicas:
  - Apply all log entries (thanks to 2PC).
  - Apply log entries in the same order (thanks to sequential way in which TC performs log entry pushes to participants).

PROBLEMS WITH MOCK

- NOT fault tolerant (but durable):
  - Because TC must wait for all replicas to reply that they are going to perform the update, the coordinator needs to block every time one replica is slow, disconnected, or dead.
  - But the mock does provide more durability than 2PC across shards.
- When the coordinator dies, someone else must become coordinator. Yet, we must have only one (at most) coordinator, otherwise different coordinators may impose different orders on log entries. This is called leader election and is not addressed in 2PC, which assumes a static coordinator!

Consensus protocols

Compile Your Creativity.

girls who code at the University of Virginia

@Girls Hoo Hack 2025: Hacks and Crafts!
October 25-26, Rice Hall

Girls Hoo Hack is an in person 48-hour hackathon hosted by Girls Who Code at UVA that aims to foster a community of diverse thought and skill.

Scan here to register and follow us @girlshoohack on Instagram to stay updated!

- Open to all women + allies
- All skill levels welcome!
- Fun workshops, including:
  - Intro to APIs
  - Developing Soft Skills
  - Matcha Meet
  - More!
- Networking opportunities

CONSENSUS PROTOCOLS

- Require only a majority of nodes to be up at any time in order to make progress.
- Similar to 2PC, but instead of waiting for all participants to respond, they wait for a majority of the replicas to respond.
  - In a fail-stop failure model (i.e., nodes are not malicious), the majority needed is a simple majority; i.e., one can tolerate f simultaneous failures with 2f+1 replicas.
  - In a malicious failure model, one needs a super-majority, i.e., one can tolerate f simultaneous failures with 3f+1 replicas.

SIMPLE MAJORITIES

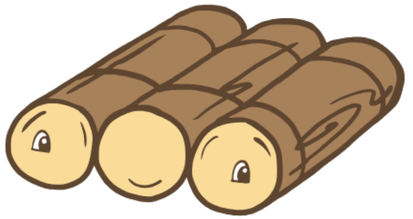
- There cannot exist two majorities in a given group at the same time.
  - This means that if a node obtains OKs from a majority of nodes – say in a first phase like 2PC's – then another node (e.g., another simultaneous coordinator) is guaranteed to not have obtained OKs from a majority of the nodes.
  - This lets us replace a dead Coordinator with a new one without introducing inconsistencies. That's how we address the leader election problem.
- Any two majorities of a group will overlap in at least one node.
  - This means that if an old Coordinator obtained OKs from a majority of the nodes, then sent COMMIT messages that were received by a majority of the nodes, and subsequently crashed before it could inform the other nodes of the COMMIT outcome, then a new Coordinator that is "elected" subsequently, will learn about the outcome by talking to any (other) majority, and so it can continue the commit process that the first (now dead) Coordinator began.

PAXOS AND RAFT

— Paxos [Lamport-1998]: Original protocol. Solves the basic consensus problem as defined in the Agreement lecture (consensus on the value of a write-once register, with the consistency, validity, and termination requirements).

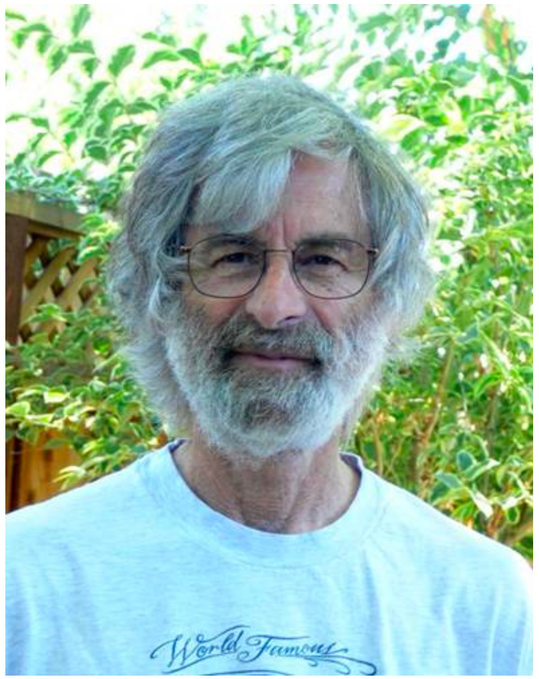
Paxos

— RAFT [Ongaro-Ousterhout-2014]: More recent, operates at a higher level of abstraction, and shows very clearly how to replicate the WAL (for example) to implement fault-tolerant transactions.

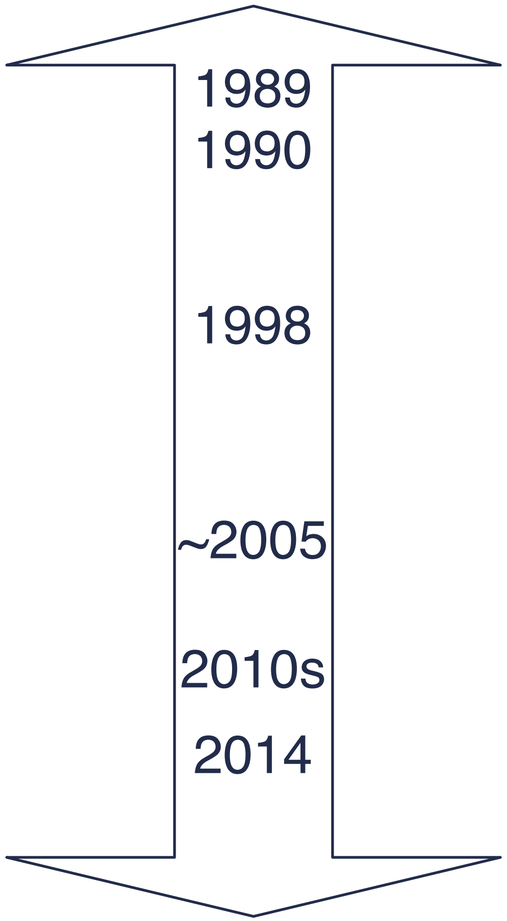


LESLIE LAMPORT

- Winner of the 2013 Turing Award
- Known for imposing clear, well-defined coherence on the seemingly chaotic behavior of distributed computing systems
- Also known for
  - LaTeX (typesetting system)
  - Byzantine Fault Tolerance
  - TLA+ (formal verification tools)
  - ...



A BRIEF HISTORY OF PAXOS



1989 Viewstamped Replication – Liskov & Oki

1990 Paxos – Leslie Lamport, “**The Part-Time Parliament**”

1998 Paxos paper published

~2005 First practical deployments

2010s Widespread use!

2014 Lamport wins Turing Award

THE PART-TIME PARLIAMENT

first submitted in 1990, published in 1998, won ACM SIGOPS Hall of Fame Award in 2012

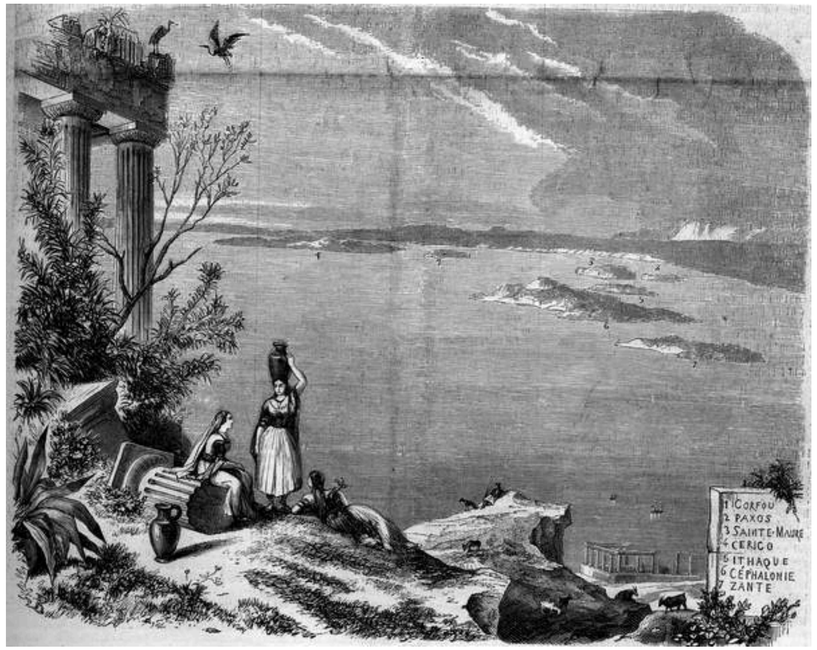
— The initial draft was obscure with an "archaeological" tone

This submission was recently discovered behind a filing cabinet in the *TOCS* editorial office. Despite its age, the editor-in-chief felt that it was worth publishing. Because the author is currently doing field work in the Greek isles and cannot be reached, I was asked to prepare it for publication.

The author appears to be an archeologist with only a passing interest in computer science. This is unfortunate; even though the obscure ancient Paxon civilization he describes is of little interest to most computer scientists, its legislative system is an excellent model for how to implement a distributed computer system in an asynchronous environment. Indeed, some of the refinements the Paxons made to their protocol appear to be unknown in the systems literature.

The author does give a brief discussion of the Paxon Parliament's relevance to distributed computing in Section 4. Computer scientists will probably want to read that section first. Even before that, they might want to read the explanation of the algorithm for computer scientists by Lampson [1996]. The algorithm is also described more formally by De Prisco et al. [1997]. I have added further comments on the relation between the ancient protocols and more recent work at the end of Section 4.

Keith Marzullo
University of California, San Diego



PAXOS MADE SIMPLE

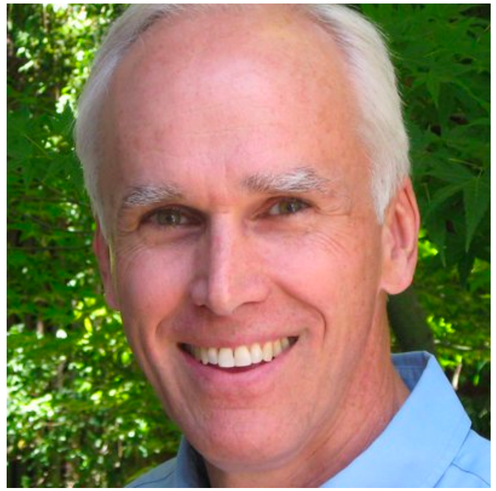
- Three years after publishing original Paxos paper, Lamport published a simplified version to explain the protocol
  - "The current version is 13 pages long, and contains no formula more complicated than n1 > n2."
  - Maybe a bit over-simplified..

Abstract

The Paxos algorithm, when presented in plain English, is very simple.

Contents

RAFT



John Ousterhout
Professor of Computer Science
Stanford University

Diego Ongaro
Ph.D., Computer Science
Stanford University

RAFT

Diego Ongaro and John Ousterhout, 2014

— *Unfortunately, Paxos is quite **difficult to understand**, in spite of numerous attempts to make it more approachable...*

— *..we set out to find a new consensus algorithm that could provide a better foundation for system building and education. Our approach was unusual in that our primary goal was **understandability**: could we define a consensus algorithm for practical systems and describe it in a way that is significantly easier to learn than Paxos?"*

A PREVIEW OF RAFT W/ VISUALIZATION

https://thesecretlivesofdata.com/raft/

UNIVERSITY of VIRGINIA

TAKEAWAYS

- Reaching consensus in DS is crucial but challenging.
- Key: from a single static coordinator to majorities.
- Approach:Replicating WAL and ensure all logs in the same order.
- Next class: **Consensus** (contd.).

ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON COURSE MATERIALS SHARED BY PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS, PROF. MICHAEL FREEDMAN, PROF. KYLE JAMIESON, PROF. WYATT LLOYD AND PROF. ROXANA GEAMBASU. MANY APPRECIATIONS FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.

THIS SLIDES INCLUDES CONTENTS FROM PROF. DAN PORTS' DISTRIBUTED SYSTEMS COURSE (UW)