# Undo for Operators: Building an Undoable E-mail Store

CSE 551

Danyang Zhuo

Some slides are borrowed from the authors

# Outline

- Recap of the paper
- Quiz
- Discussion

# Goal

- Dependability is achieved at the interface of the system and its operator
- Undo goal: support the operator
  - create a forgiving operations environment
    - tolerate human error, allow trial-and-error experiments
  - provide last-resort recovery from unknown damage
    - retroactive repair of software bugs, broken upgrades, unknown operator changes, external attacks
  - be easily comprehensible by operator and developer
  - preserve user data while refreshing system state

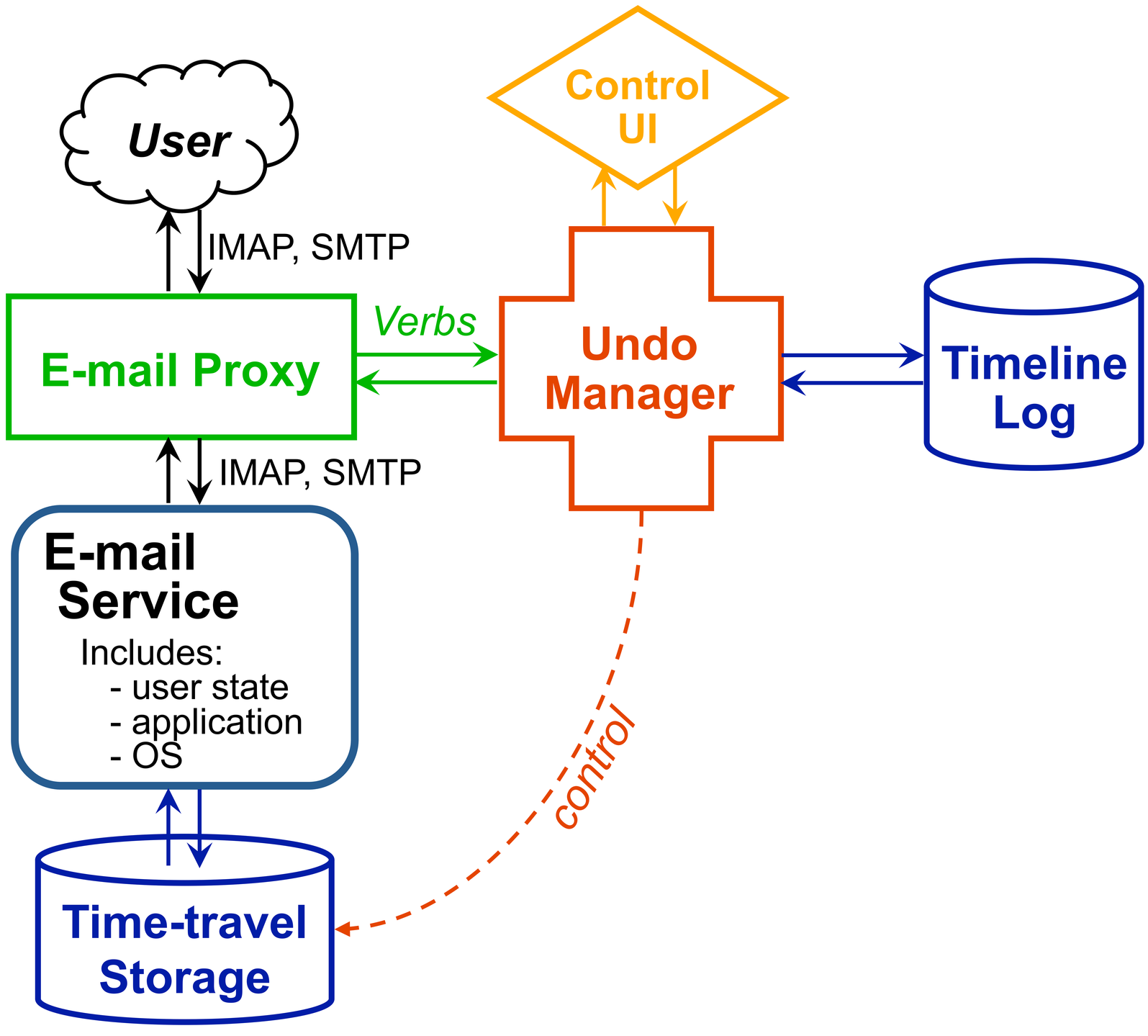
# The Three-R's Undo Model

- Provide Undo as *virtual time travel*
  - **Rewind:** roll back *all* system state, physically
  - **Repair:** make changes to past timeline to avert original disaster
  - **Replay:** roll state forward logically, merging original timeline with effects of repairs
- Key properties of 3R's Undo
  - recovery from problems at any system layer
  - recovery from unanticipated problems
  - no assumptions about correct application behavior

# Building an E-mail Undo Prototype

- Target: an undoable *e-mail store service*
  - a leaf node in the Internet e-mail network
  - accessed via IMAP and SMTP
- Built around existing e-mail store service
  - *e.g.*, sendmail+imapd, iPlanet, Exchange
- Extensible to other apps
  - architected around a reusable undo core with e-mail-specific extensions
- Written in Java

# Undo System Architecture



# Architecture Design Points

- Proxy-based design targeted at services
- Application-neutral core
  - undo manager, timeline log, time-travel storage, UI
  - contains all undo-cycle logic
- E-mail-specific semantics encapsulated
  - **proxy:** encodes e-mail interactions into verbs
  - **verbs:** define e-mail semantics
    - model of preserved state
    - model of acceptable external (observed) consistency

# Verbs

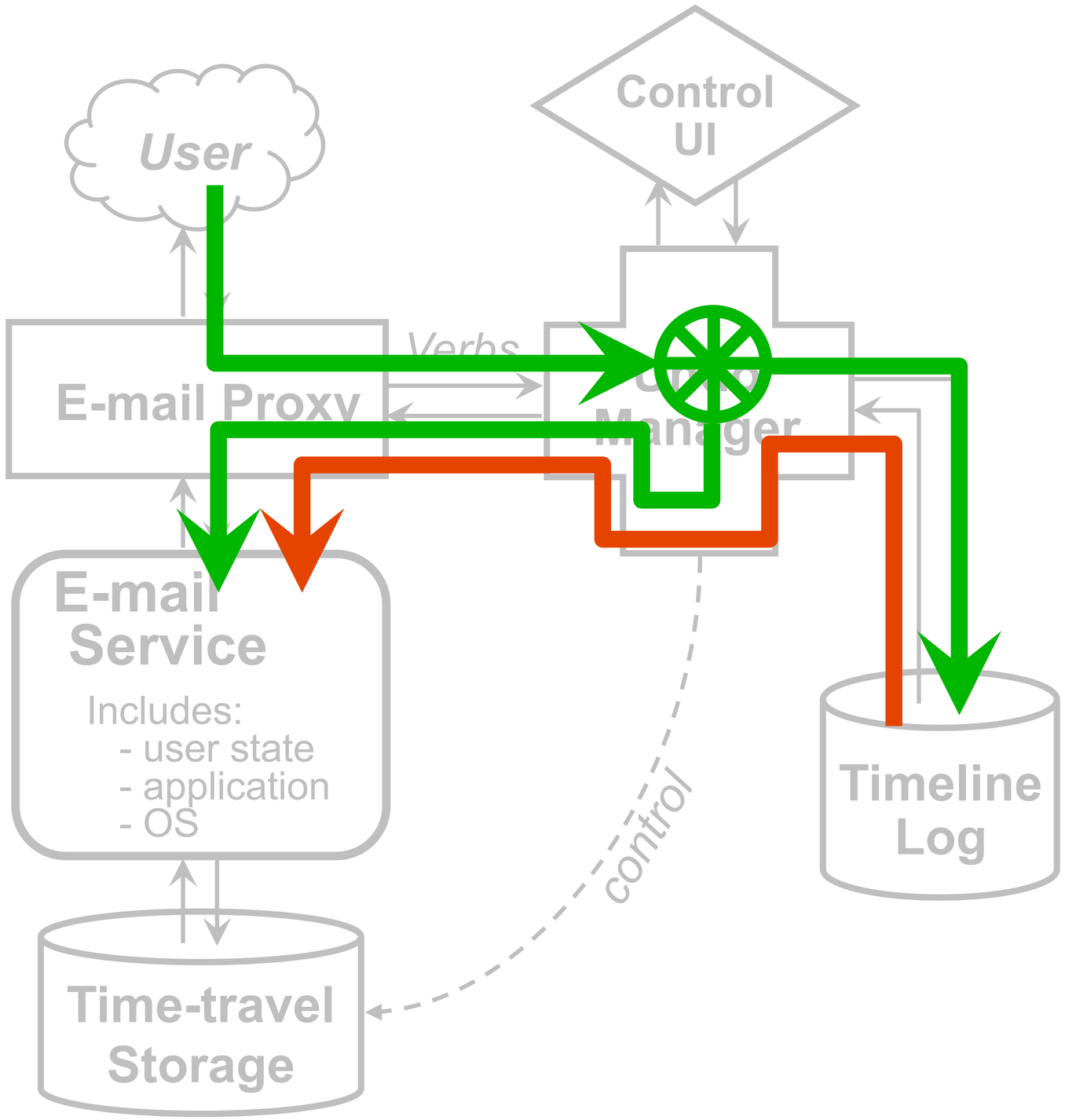
- Key construct in undo architecture
  - represent end-user state changes, state exposure
  - building block of timeline and history
  - used to detect and manage inconsistencies
- Essential for reasoning about undo
  - verbs define exactly what state is preserved during an undo cycle
  - provide framework for defining application's model of acceptable external inconsistency

# Verbs and the Undo Cycle

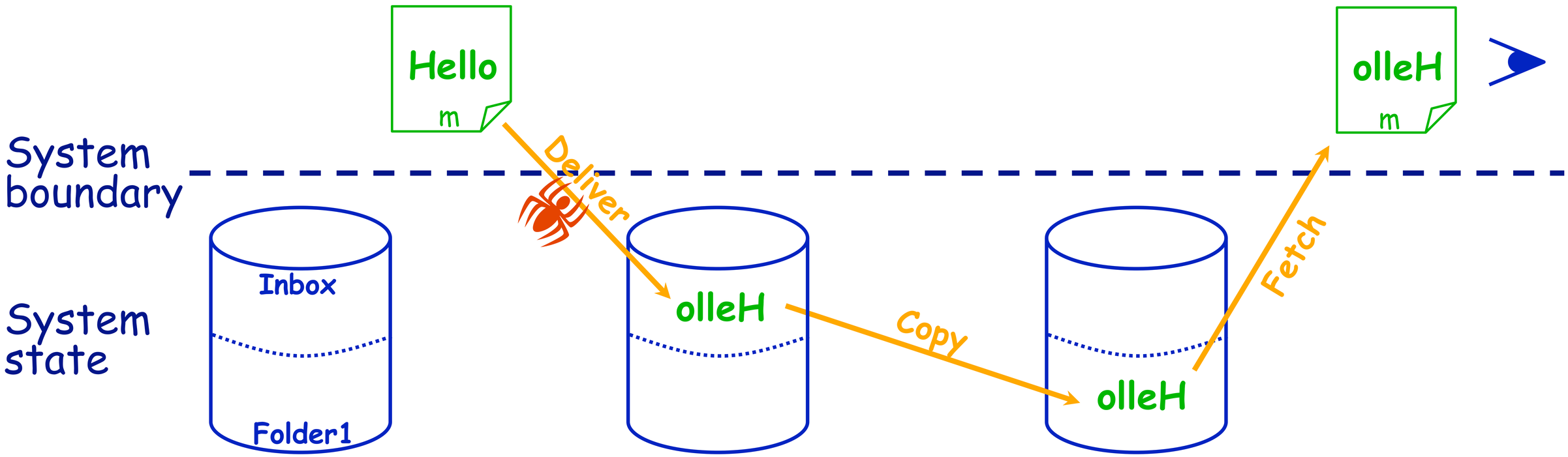
- Verb flow

  → Normal operation

  → Replay



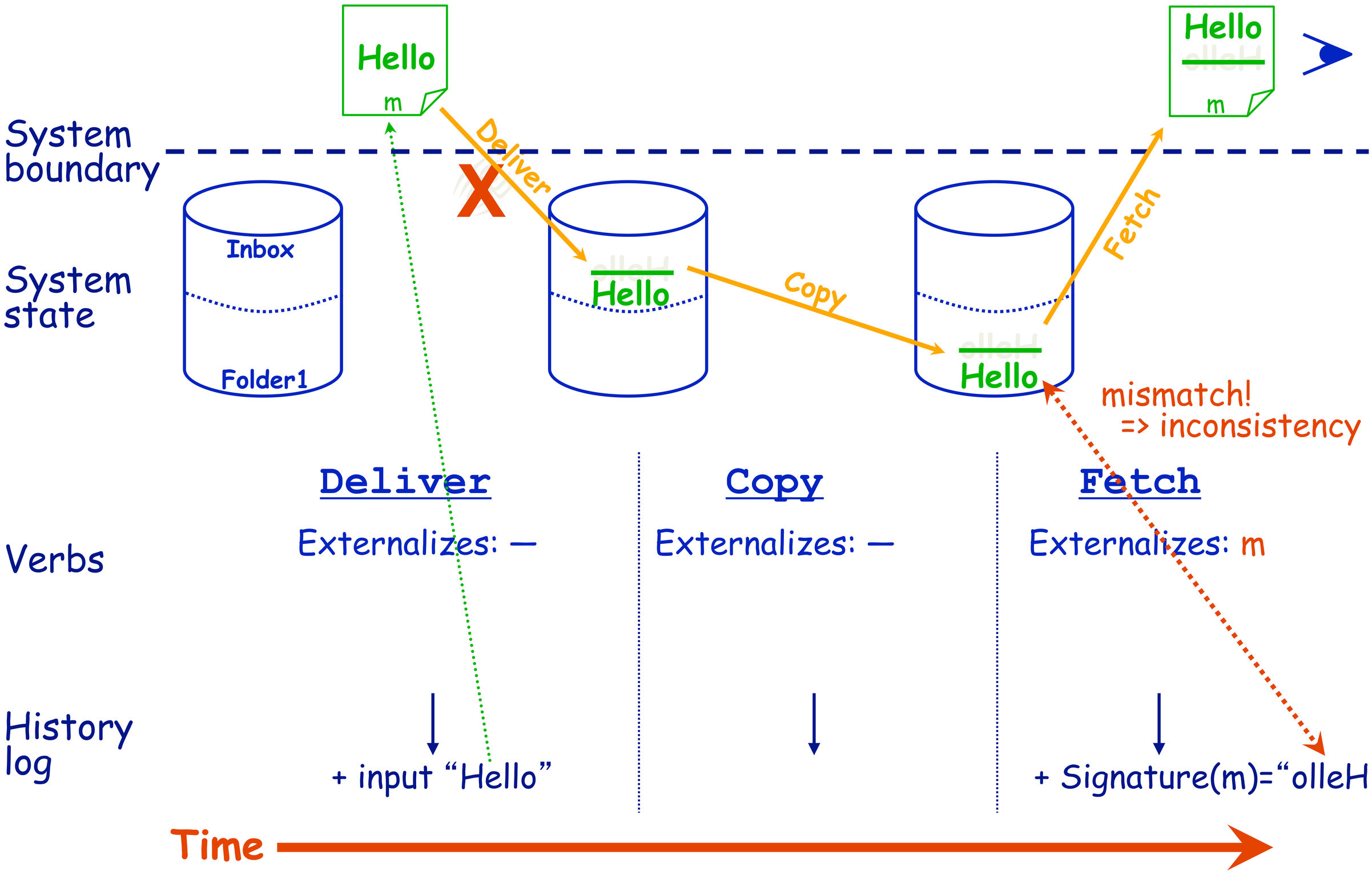
# E-mail example: original timeline




| | **Deliver** | **Copy** | **Fetch** |
|---|---|---|---|
| Verbs | Externalizes: — | Externalizes: — | Externalizes: m |
| History log | + input “Hello” | | + Signature(m)=“olleH” |

Time

# E-mail example: replay timeline



# Verbs and External Inconsistency

- Detecting inconsistency
  - verbs that externalize state record a signature of externalized data and define a comparison function
    - comparison fn. defines the external consistency model
- Handling inconsistency
  - verbs define compensation routines, invoked when inconsistencies are found
  - later non-commuting verbs are squashed to prevent data loss

# Verbs for E-mail

- SMTP & IMAP protocols mapped into 13 verbs
  - **SMTP:** Deliver
  - **IMAP:** Append, Fetch, Store, Copy, List, Status, Select, Expunge, Close, Create, Rename, Delete
    - set could easily be extended with user and subscription management functionality
- Each verb is a Java class implementing a generic, app-neutral `Verb` interface

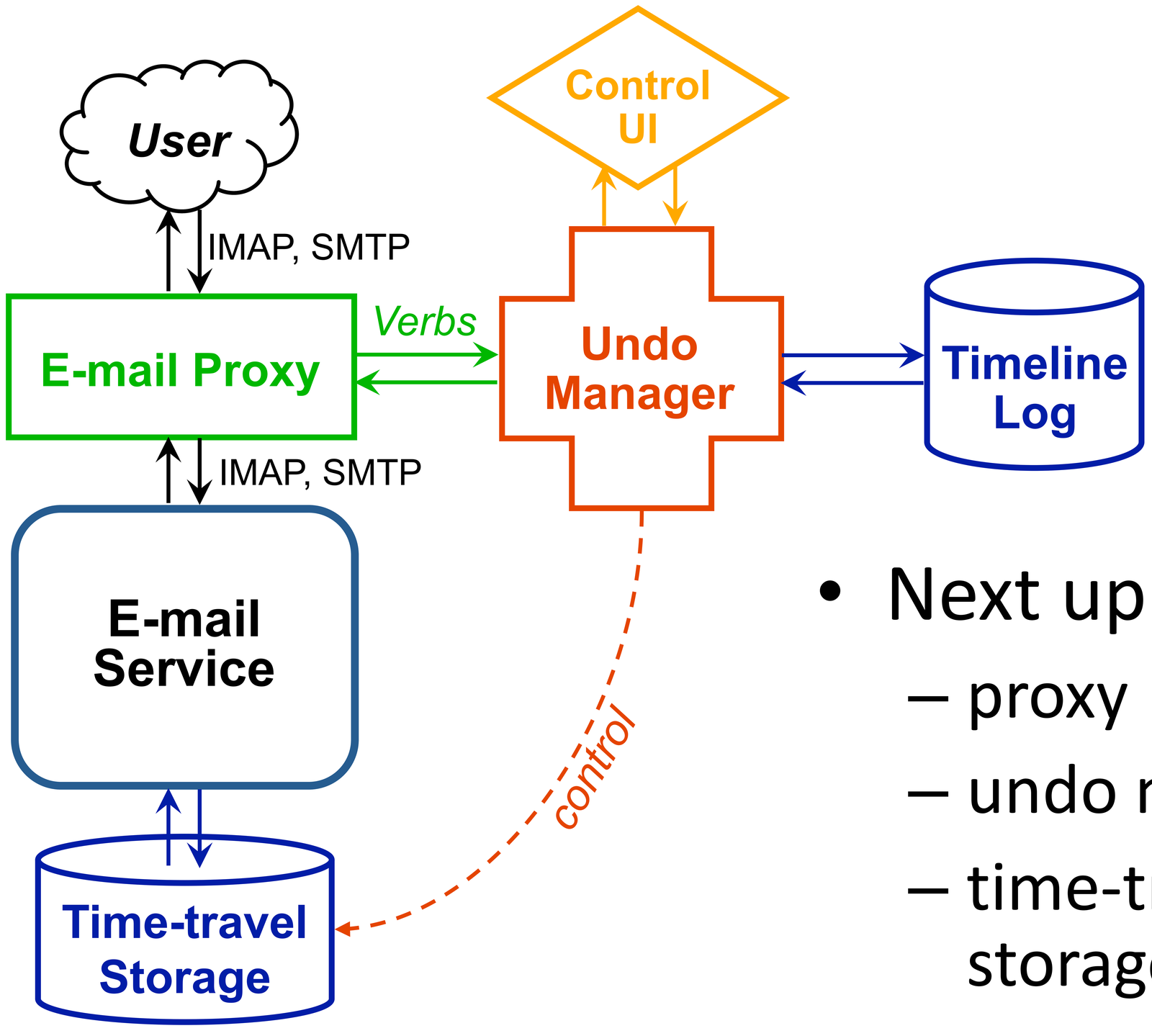
# Example Verb: STORE (set msg flags)

- Tag
  - input: target folder, message IDs, flag value
  - externalized output: resultant flags of messages
- Sequencing tests
  - commutes with & independent of SMTP verbs, all IMAP verbs except Fetch/Store/Close on same folder
- Consistency check
  - new flags must match original flags for all messages in common; no original messages should be missing
- Inconsistency handler
  - leave user a message explaining and listing changes

# An External Consistency Model for E-mail

- Retrieval (IMAP)
  - **tracked state** includes message bodies, key message headers (to/from/subject/cc), flags, folder lists, and execution status of state-altering commands
  - **inconsistency** if objects are missing or altered on replay or commands fail; order & new objects ignored
  - **compensation** via explanatory messages and creation of "lost-and-found" containers
- Delivery (SMTP)
  - only possible inconsistency is in execution status
  - one tricky case: originally-failed delivery succeeds
    - delay bounce to provide window for undo-repair

# Undo System Architecture




- Next up:
  - proxy
  - undo manager
  - time-travel storage layer

# Proxying E-mail

- IMAP proxy loop is straightforward
  - accept client connection and open server connection
  - parse commands and instantiate corresponding verbs
  - invoke undo manager to schedule, execute, log verb
    - passing a handle to the client and server connections
- SMTP is more complicated
  - failed SMTP deliveries must be logged for later retry
  - proxy poses as a server that always accepts deliveries
    - deliver verb is created *after* client has been ACK'd
    - some finesse to avoid being an open-relay
  - if verb fails, proxy sends a bounce after a delay

# Undo Manager Implementation

- Timeline log
  - BerkeleyDB *recno* database storing serialized verbs
  - also stores control records to make undo manager state persistent and recoverable
- Verb scheduling
  - all verb execution passes through undo manager
  - scoreboard-like structure sequences verbs according to independence, commutativity, ordering properties
- External inconsistency management
  - undo manager coordinates verb APIs to ensure that external inconsistencies are detected and handled

# A Time-Travel Storage Layer

- Base: Network Appliance Filer with snapshots
- Java wrapper for Filer's management CLI
  - provides direct control of snapshot create/restore
  - periodically takes snapshots, aging out old ones
    - hierarchical scheduling allows the 31 snapshots to span one month, with density inversely proportional to age
- Rewind restores the closest prior snapshot, then replays up to the exact rewind point
- Challenge: making snapshot restore undoable
  - solution: implement rollback by copying old snapshot forward to present (expensive, but necessary)

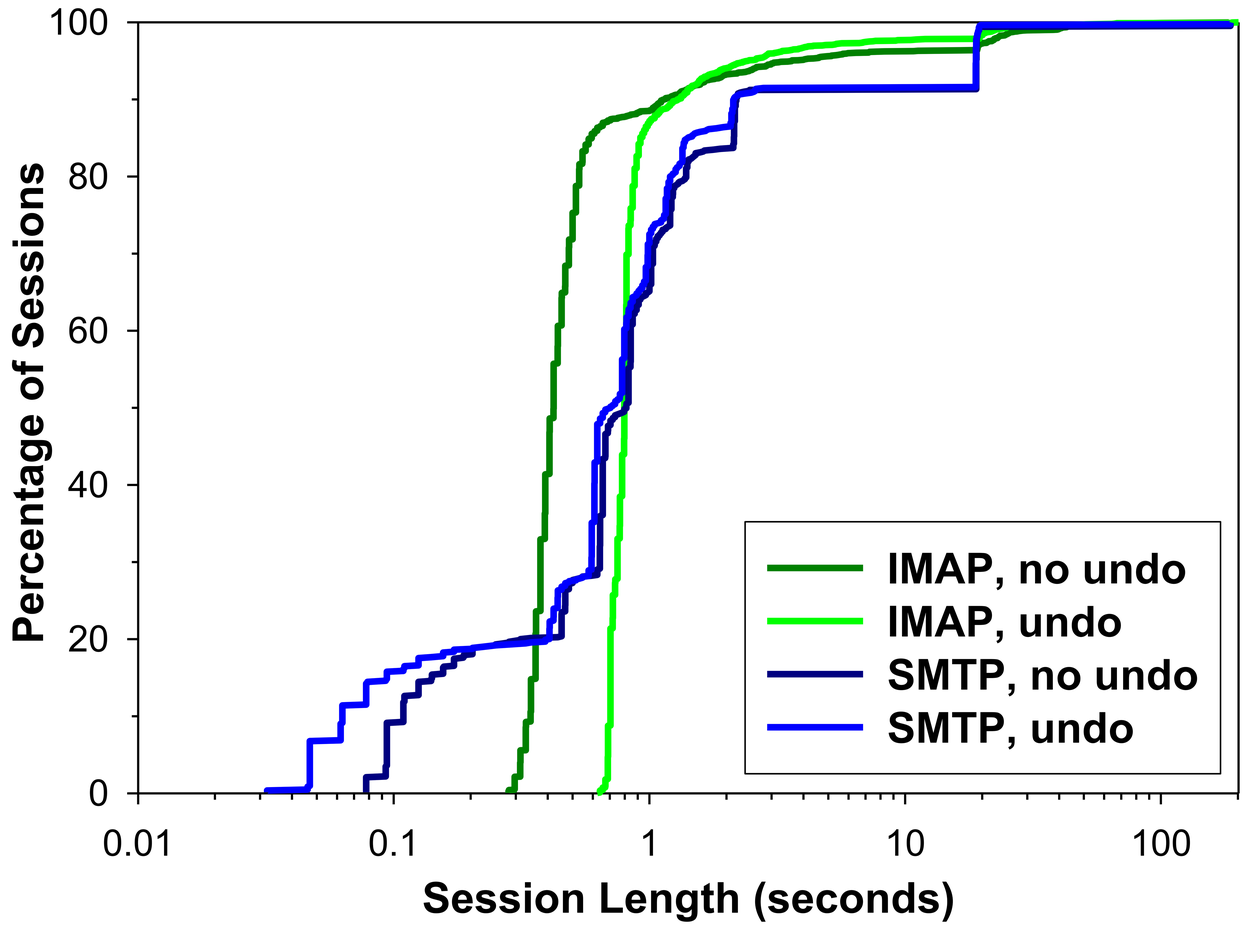
# Measuring Overhead for Undo

- Workload
  - modified SPECmail2000 e-mail benchmark
    - simulates traffic seen by an ISP mail server
    - modified to use IMAP instead of POP, all mail local
  - configured with 5000 users
    - about 56 SMTP cxns/minute, 149 IMAP cxns/minute
- Setup
  - **mail server:** Linux, sendmail + UW imapd, 5000 users
  - **undo proxy:** Win2k, Sun 1.4.1 JDK
  - **workload generator:** Win2k, Sun 1.4.1 JDK
  - **storage:** all mailboxes and logs on NetApp Filer
    - mailboxes accessed via NFS, logs via CIFS

# Overhead Results: Space and Time

- Space overhead
  - 5GB/day for timeline log
    = 1GB/day per 1000 users
  - 325KB per 1000 mail folder name translations
  - per 120GB disk:
    - 7 weeks of timeline for 1000 ISP users
    - or 350 million folder name translations

- **Time overhead**
  - cumulative distribution plot of IMAP and SMTP session lengths



# Quiz

- Where is the output of the system stored?

# Quiz

- How does Undo handle non-determinism?

# Quiz

- How does the Undo manager order the requests into a causal sequence?

# Discussion

- Undo versus Replication
  - Practicality
  - Use case
  - Why does industry use replication but not Undo?

# Discussion

- What makes Email a natural fit for Undo?
- Is the external consistency model generic enough for other applications?

# Discussion

- What components of Undo are trusted? What is the threat model?