Rewind, Repair, Replay:
Three R's to cope with operator error

Aaron Brown
UC Berkeley ROC Group
abrown@cs.berkeley.edu

IBM Almaden, 22 March 2002

Outline

- Recovery-Oriented Computing background
- Motivation: the importance of human operators
- The Three R's: human-centric recovery
- 3R's challenges
- Implementing and evaluating the 3R's
- Status, future directions, conclusions

ROC motivation: the past 15 years

- **Goal #1: Improve performance**
- **Goal #2: Improve performance**
- **Goal #3: Improve cost-performance**
- ***Assumptions***
  - Humans are perfect (they don't make mistakes during installation, wiring, upgrade, maintenance or repair)
  - Software will eventually be bug free
    (Hire better programmers!)
  - Hardware MTBF is already very large (~100 years between failures), and will continue to increase
  - Maintenance costs irrelevant vs. Purchase price
    (maintenance a function of price, so cheaper helps)

Where we are today

- **MAD TV, “Antiques Roadshow, 3005 AD”**

  VALTREX:

  “Ah ha. You paid 7 million Rubex too much. My suggestion: beam it directly into the disposal cube.

  These pieces of crap crashed and froze so frequently that people became violent!

  Hargh!”

“Worthles 0 Rubex”

Recovery-Oriented Computing Philosophy

"If a problem has no solution, it may not be a problem, but a fact, not to be solved, but to be coped with over time"

— Shimon Peres ("Peres's Law")

- **People/HW/SW failures are facts, not problems**
- **Recovery/repair is how we cope with them**
- **Improving recovery/repair improves availability**
  - UnAvailability = $\frac{MTTR}{MTTF}$ *(assuming MTTR much less than MTTF)*
  - 1/10th MTTR just as valuable as 10X MTBF
- **ROC also helps with maintenance/TCO**
  - since major Sys Admin job is recovery after failure
- **Since TCO is 5-10X HW/SW, sacrifice disk/DRAM/CPU for recovery if necessary**

ROC approach

1. **Collect data to see why services fail**
2. **Create benchmarks to measure recovery**
   - use failure data as workload for benchmarks
   - benchmarks inspire and enable researchers / humiliate companies to spur improvements
3. **Create and Evaluate techniques to help recovery**
   - identify best practices of Internet services
   - ROC focus on fast repair (they are facts of life) vs. FT focus longer time between failures (problems)
   - make human-machine interactions synergistic vs. antagonistic

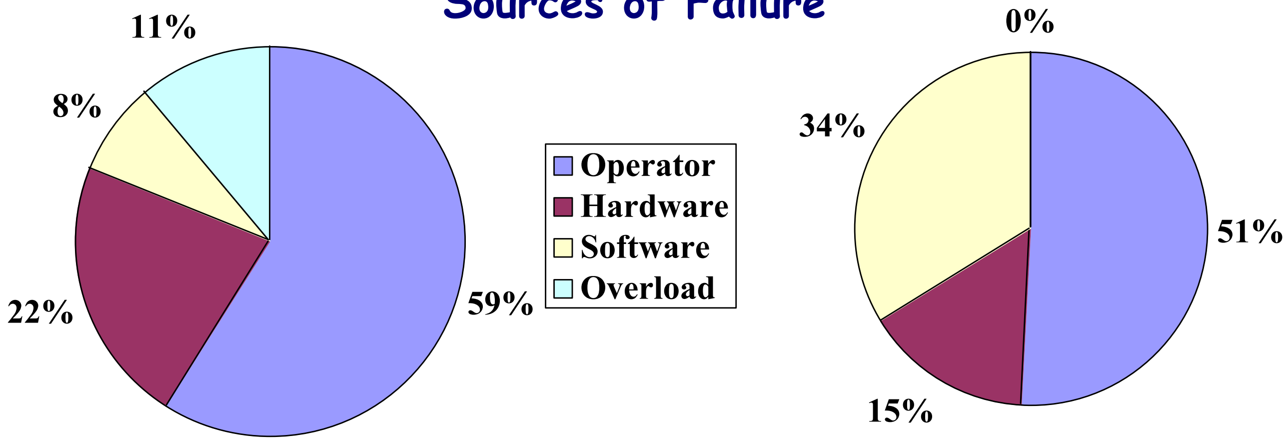
Outline

- Recovery-Oriented Computing background
- **Motivation: the importance of human operators**
- The Three R's: human-centric recovery
- 3R's challenges
- Implementing and evaluating the 3R's
- Status, future directions, conclusions

Human error

- **Human operator error is the leading cause of dependability problems in many domains**

Sources of Failure



Public Switched Telephone Network

Average of 3 Internet Sites

- **Operator error cannot be eliminated**
  - humans inevitably make mistakes: "to err is human"
  - *automation irony* tells us we can't eliminate the human

Source: D. Patterson et al. Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies, UC Berkeley Technical Report UCB//CSD-02-1175, March 2002.

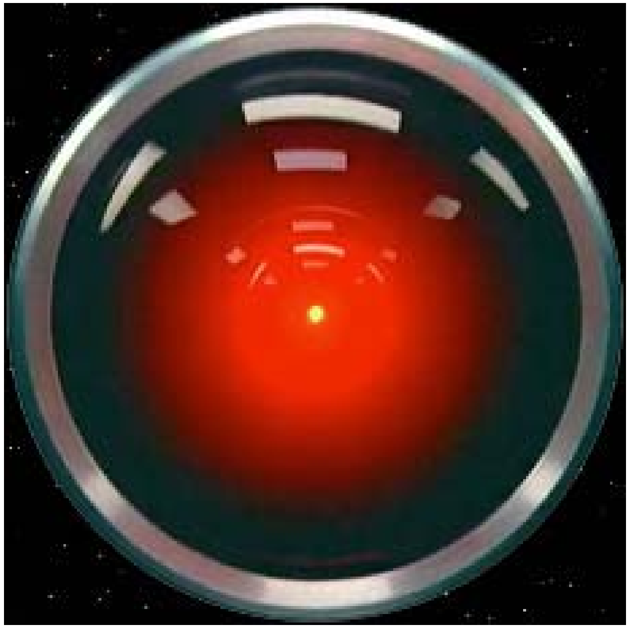
The ironies of automation

- **Automation doesn't remove human influence from system**
  - shifts the burden from operator to designer
    - » designers are human too, and make mistakes
    - » if designer isn't perfect, human operator still needed
- **Automation can make operator's job harder**
  - reduces operator's understanding of the system
    - » automation increases complexity, decreases visibility
    - » no opportunity to learn without day-to-day interaction
  - uninformed operator still has to solve exceptional scenarios missed by (imperfect) designers
    - » exceptional situations are already the most error-prone

Source: J. Reason, Human Error, Cambridge University Press, 1990.

A science fiction analogy

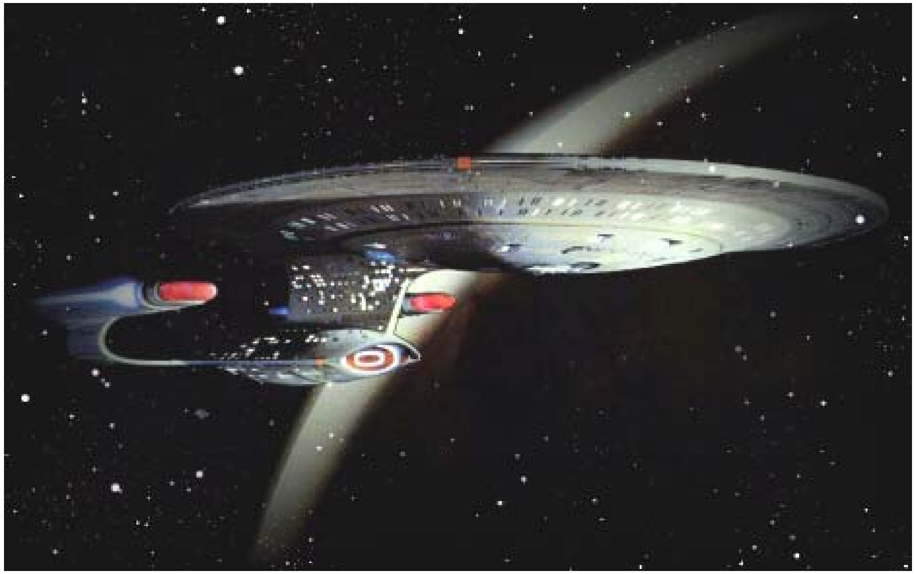
- **Full automation**



HAL 9000 (2001)

- **Suffers from effects of the automation ironies**
  - system is opaque to humans
  - only solution to unanticipated failure is to pull the plug?

- **Human-aware automation**



Enterprise computer (2365)

- **24th-century engineer is like today's SysAdmin**
  - a *human* diagnoses & repairs computer problems
  - automation used in human-operated diagnostic tools

Matching recovery & human behavior

- **Need a recovery mechanism that matches the way humans behave**
  - tolerate inevitable operator errors
    - » even with correct intentions, humans still make "slips"
  - harness hindsight
    - » ~70% of human errors are immediately self-detected
    - » non-human failures are often avoidable in hindsight
      - *e.g.*, misconfigurations, break-ins, viruses, etc.
      - provide *retroactive repair* for these failures
  - support trial & error
    - » today's systems are too complex to understand *a priori*
    - » allow exploration, learning from mistakes

Outline

- Recovery-Oriented Computing background
- Motivation: the importance of human operators
- **The Three R's: human-centric recovery**
- 3R's challenges
- Implementing and evaluating the 3R's
- Status, future directions, conclusions

"Three R's" Recovery

- **Time travel for system operators**
- **Three R's for recovery**
  - **Rewind:** roll all system state backwards in time
  - **Repair:** change system to prevent failure
    - » e.g., fix latent error, retry unsuccessful operation, install preventative patch
  - **Replay:** roll system state forward, replaying end-user interactions lost during rewind
- ***All three R's are critical***
  - rewind enables undo
  - repair lets user/administrator fix problems
  - replay preserves updates, propagates fixes forward

Example 3R's scenarios

- **Direct operator errors**
  - system misconfiguration
    - » configuration file change, email filter installation, ...
  - accidental deletion of data
    - » "rm -rf /", deleting a user's email spool, reversed copy during data reorganization, ...
- **Retroactive repair**
  - mitigate external attacks
    - » retroactively install virus/spam filter on email server; effects are squashed on replay
  - repair broken software installations
    - » mis-installed software patch, installation of software that corrupts data, software upgrade that slows performance

Context

- **Traditional Undo gives only two R's**
  - rewind & repair or rewind & replay
  - *e.g.*, backup/restore, checkpointing
- **RDBMS log-based recovery**
  - typically implements two R's: rewind/replay used to recover from crashes, deadlock, etc.
    - » but no opportunity for repair during rewind/replay cycle
  - DB logging mechanisms could give all 3 R's
    - » but not at whole-system level

Outline

- Recovery-Oriented Computing background
- Motivation: the importance of human operators
- The Three R's: human-centric recovery
- **3R's challenges**
  - delineating state preserved by replay
  - externalized state
  - granularity
  - history model
- Implementing and evaluating the 3R's
- Status, future directions, conclusions

Challenge #1: state delineation

- **What state changes does Replay restore?**
  - ideal: only updates that are important to the end-user
    - » allows effects of repairs to propagate forward
- **Replay should preserve *intent* of updates**
  - not physical manifestation in state
    - » repair might alter the physical representation
  - achieved by protocol-level logging/replay of updates
    - » *e.g.*, SMTP, IMAP, JDBC/SQL, XML/SOAP, ...
    - » argues for proxy-based undo implementations
- **Replay ignores prior repairs lost during rewind**
  - too difficult to record intent of repairs (for now)

Challenge #2: externalized state

- **The equivalent of the "time travel paradox"**
  - the 3R cycle alters state that has previously been seen by an external entity (user or another computer)
  - produces inconsistencies between internal and external views of state after 3R cycle
- **Examples**
  - a formerly-read/forwarded email message is altered
  - a failed request is now successful or vice versa
  - item availability estimates change in e-commerce, affecting orders
- **No complete fix; solutions just manage the inconsistency**

Externalized state: solutions

- **Ignore the inconsistency**
  - let the (human) user tolerate it
  - appropriate where app. already has loose consistency
    - » *e.g.*, email message ordering, e-commerce stock estimates
- **Compensating/explanatory actions**
  - leave the inconsistency, but explain it to the user
  - appropriate where inconsistency causes confusion but not damage
    - » *e.g.*, 3R's delete an externalized email message; compensating action replaces message with a new message explaining why the original is gone
    - » *e.g.*, 3R's cause an e-commerce order to be cancelled; compensating action refunds credit card and emails user

Externalized state: solutions (2)

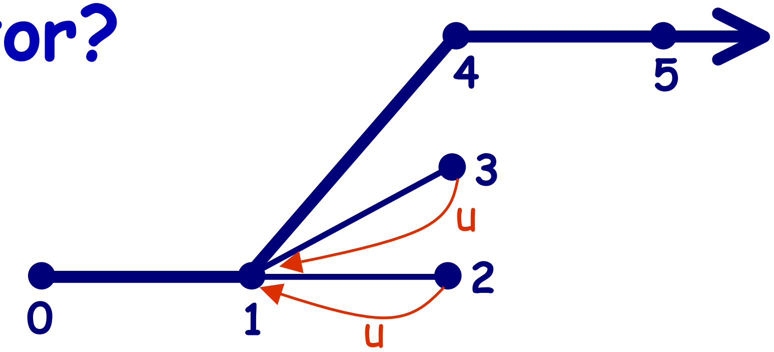
- **Expand the boundary of Rewind**
  - 3R cycle induces rollback of external system as well
    - » external system reprocesses updated externalized data
  - appropriate when externalized state chain is short; external system is under same administrative domain
    - » danger of expensive cascading rollbacks; exploitation
- **Delay execution of externalizing actions**
  - allow inconsistency-free undo only within delay window
  - appropriate for asynchronous, non-time-critical events
    - » *e.g.*, sending mailer-daemon responses in email or delivering email to external hosts

Challenge #3: granularity

- **Making 3R's available at multiple granularities**
  - user, system, cluster, service
- **Why multiple granularities?**
  - efficiency and scalability
    - » limit rollbacks to minimal affected state
  - allow users to repair their own problems, reducing operator's burden
- **Difficulties**
  - coordination of rewind/replay with concurrent undos at different granularities
  - respecting dependencies between shared and per-user state

Challenge #4: history model

- **How should the 3R-altered timeline be presented to the operator?**
  - single rewind/replay?
  - linearized history?
  - full branching history with all time points available?
  - without replaying repairs, best option is multiple-rewind, single-replay



- **What do users see during 3R cycle?**
  - read-only snapshot of unwound state?
    - » easy to implement
  - synthesized view of up-to-date state?
    - » easier for users to understand

Outline

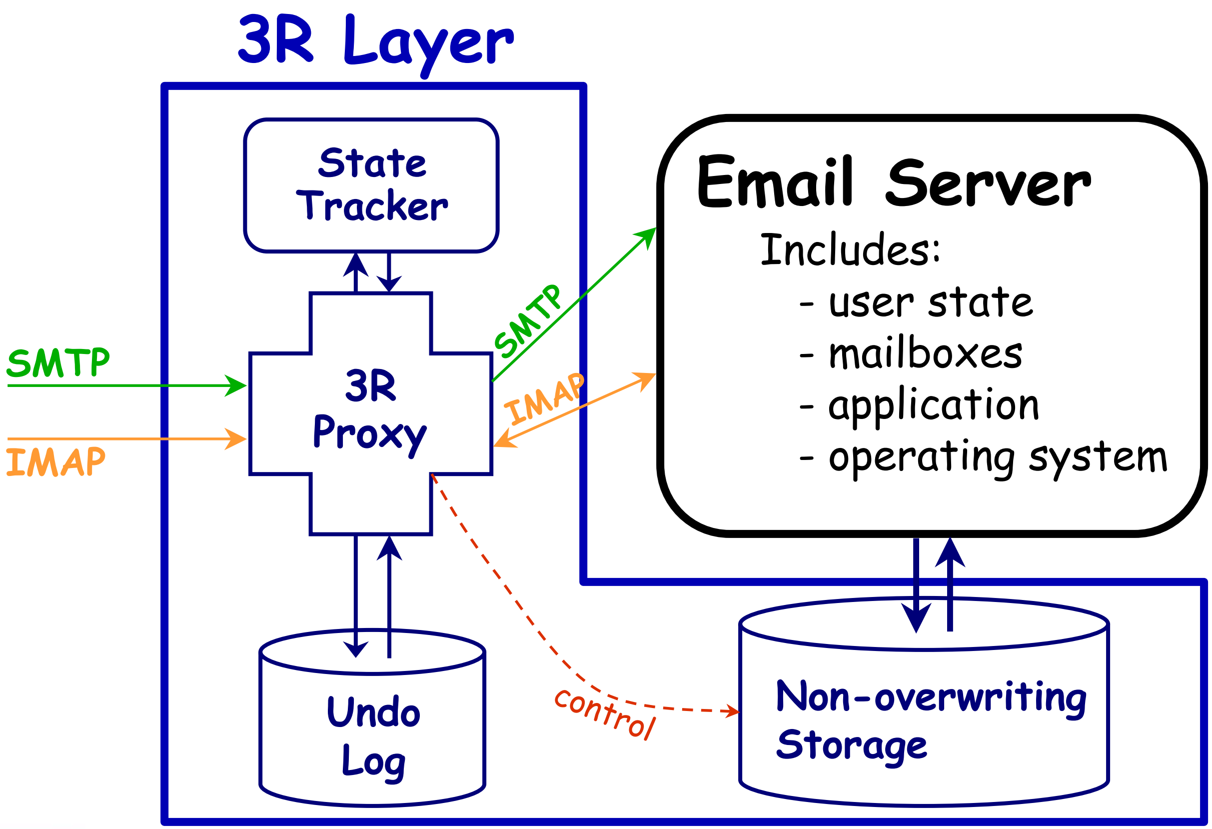
- Recovery-Oriented Computing background
- Motivation: the importance of human operators
- The Three R's: human-centric recovery
- 3R's challenges
- **Implementing and evaluating the 3R's**
- Status, future directions, conclusions

Prototype implementation: an undoable email service

- **Why email?**
  - essential "nervous system" for enterprises, individuals
  - most popular Internet service
  - good balance of hard state and relaxed consistency
  - many opportunities for human error, retroactive repair
- **Prototype goals**
  - demonstrate feasibility and measure overhead
  - explore 3R challenges, especially externalized state
  - use as testbed for developing recovery benchmarks

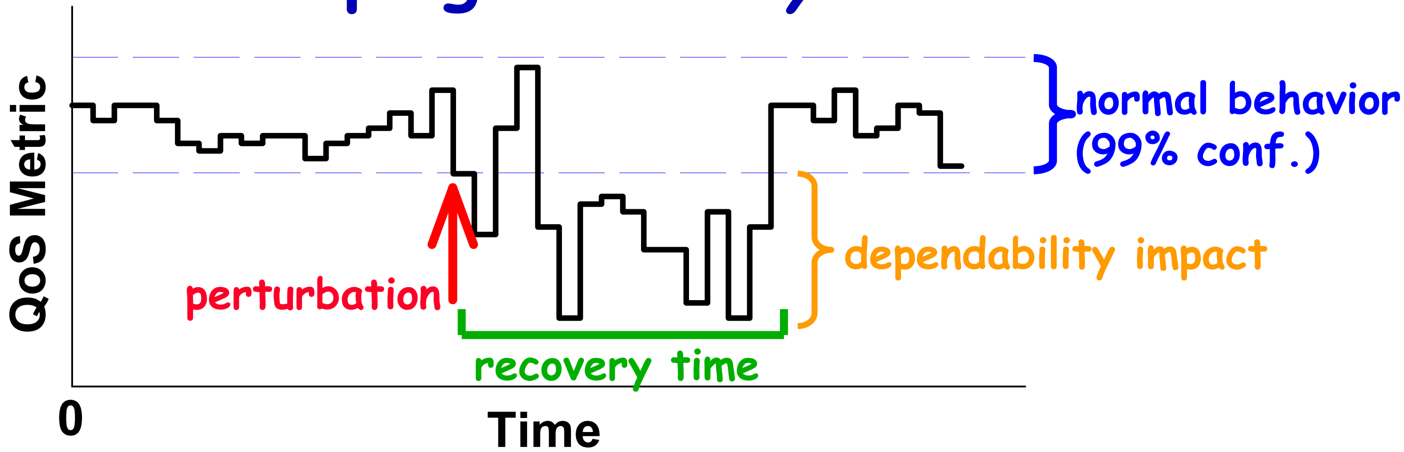
3R's Email Prototype

- **Prototype architecture**
  - proxy implementation wrapping existing mail server
  - non-overwriting storage for rewind
  - SMTP and IMAP logging for replay



Evaluating the three R's

- **Traditional performance benchmarks don't help**
- **We're developing *recovery benchmarks***



- **Human operators participate in benchmarks**
  - diagnose problems, perform repairs, carry out maintenance tasks
  - mistakes act as an additional perturbation source
  - we measure dependability impact, human error rate, required human interaction time

Outline

- Recovery-Oriented Computing background
- Motivation: the importance of human operators
- The Three R's: human-centric recovery
- 3R's challenges
- Implementing and evaluating the 3R's
- Status, future directions, conclusions

Status and future directions

- **Status**
  - currently implementing prototype in email service
  - evaluating solutions to externalized state problem for email
  - starting feasibility studies for recovery benchmarks
- **Future directions**
  - generalize 3R model
    - » examine other applications
    - » extend to lower levels of system: storage, HW
    - » develop model of state organization for 3R-capable systems
  - investigate granularities and richer history models

Conclusions

- **Peres's law suggests new focus on recovery**
- **The three R's provide a recovery mechanism for today's dependability problems**
  - human operator error
  - unanticipated failure compounded by operator reaction
  - maybe even external attack
- **3R's are synergistic with operator behavior**
  - assume mistakes
  - quick recovery even without diagnosis
  - allow trial & error exploration, retroactive repair
- **Many challenges remain in model, implementation**

For more information

- **Web: http://roc.cs.berkeley.edu/**
  - ROC overview, talks, papers
  - Drafts of workshop papers on the 3R's, recovery benchmarks, real-world failure data analysis
- **Email: abrown@cs.berkeley.edu**

Backup Slides

Discussion topics

- Externalized state—do solutions generalize?
- Comparison with existing recovery systems
- Evaluation: tasks for benchmarks?
- Prototype: what non-overwriting storage layer?

A more technical perspective...

- **Services as model for future of IT**
- **Availability is now vital metric for services**
  - near-100% availability is becoming mandatory
    - » for e-commerce, enterprise apps, online services, ISPs
  - but, service outages are frequent
    - » 65% of IT managers report that their websites were unavailable to customers over a 6-month period
      - 25%: 3 or more outages
  - outage costs are high
    - » downtime costs of $14K - $6.5M per hour
    - » social effects: negative press, loss of customers who "click over" to competitor

Source: InternetWeek 4/3/2000

Downtime Costs (per Hour)

- Brokerage operations $6,450,000
- Credit card authorization $2,600,000
- Ebay (1 outage 22 hours) $225,000
- Amazon.com $180,000
- Package shipping services $150,000
- Home shopping channel $113,000
- Catalog sales center $90,000
- Airline reservation center $89,000
- Cellular service activation $41,000
- On-line network fees $25,000
- ATM service fees $14,000

Sources: InternetWeek 4/3/2000 + *Fibre Channel: A Comprehensive Introduction*, R. Kembel 2000, p.8.
"...based on a survey done by Contingency Planning Research."

ACME: new goals for the future

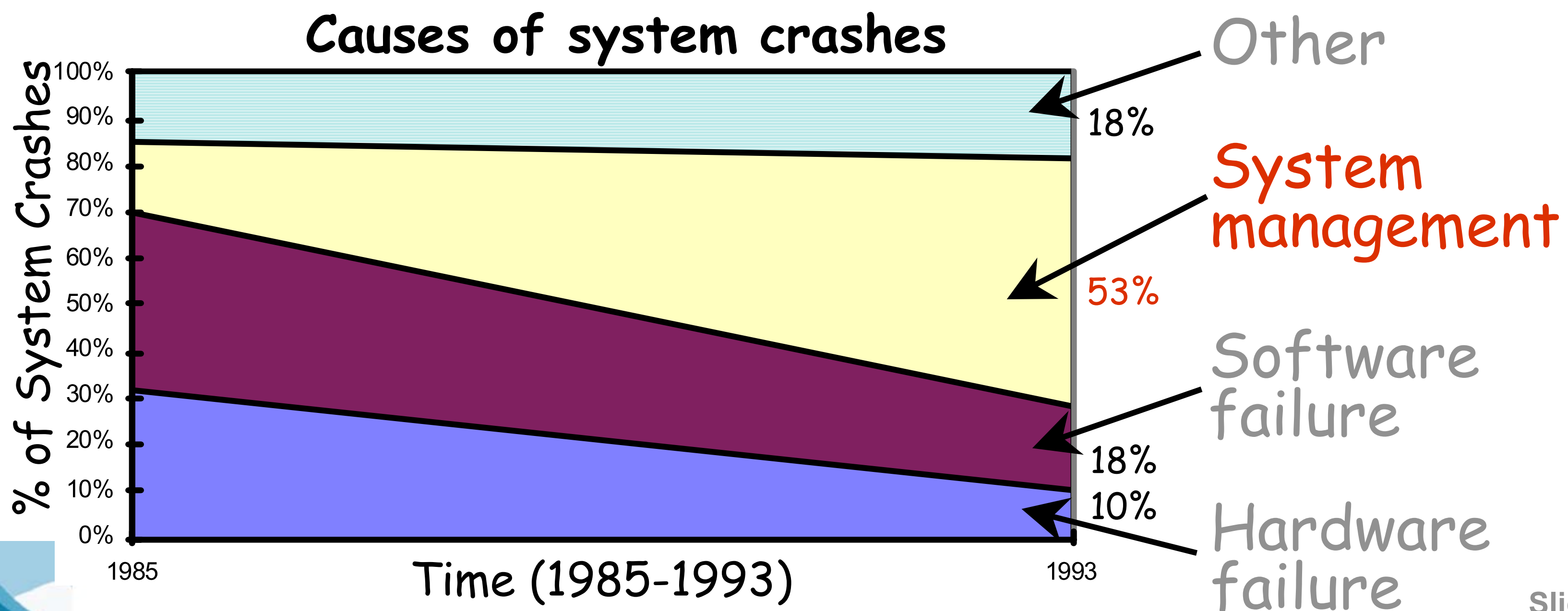
- **Availability**
  - 24x7 delivery of service to users
- **Changability**
  - support rapid deployment of new software, apps, UI
- **Maintainability**
  - reduce burden on system administrators
  - provide helpful, forgiving SysAdmin environments
- **Evolutionary Growth**
  - allow easy system expansion over time without sacrificing availability or maintainability

Where does ACME stand today?

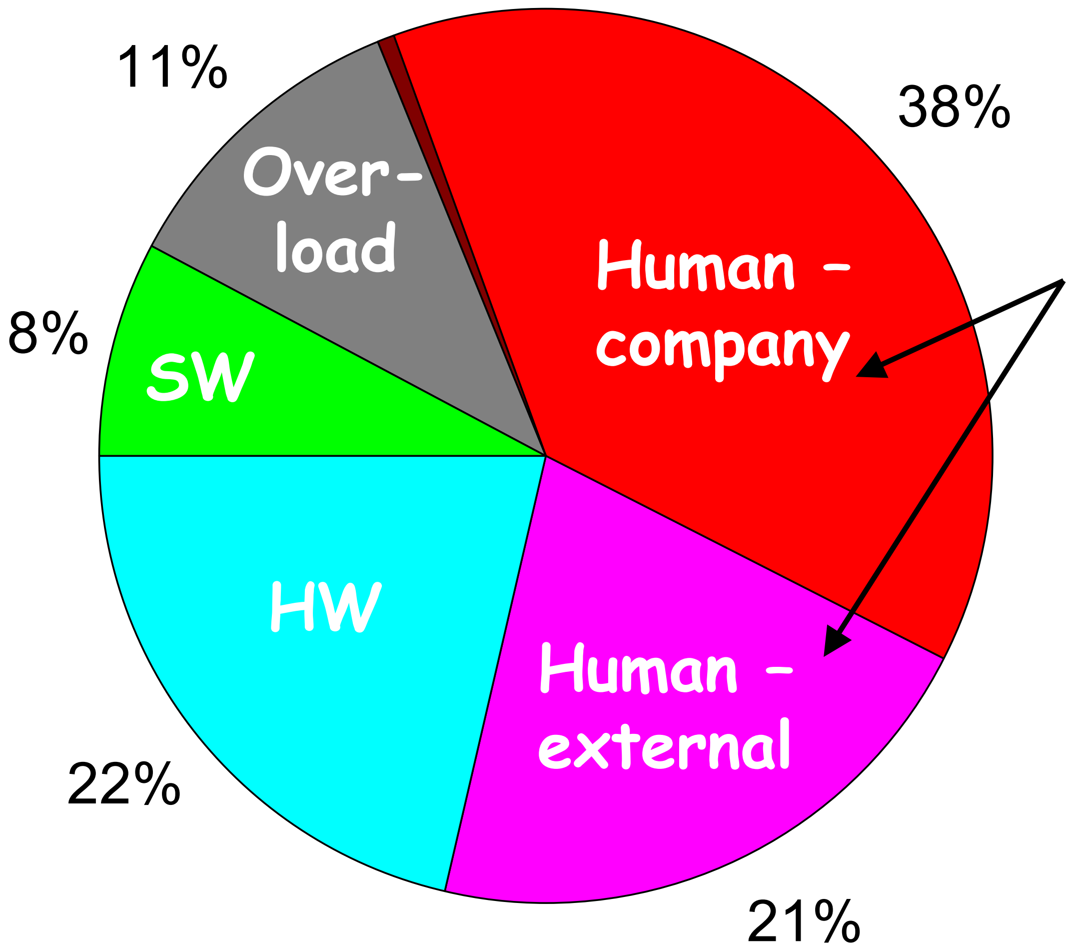
- **Availability: failures are common**
  - Traditional fault-tolerance doesn't solve the problems
- **Changability**
  - In back-end system tiers, software upgrades difficult, failure-prone, or ignored
  - For application service over WWW, daily change
- **Maintainability**
  - system maintenance environments are unforgiving
  - human operator error is single largest failure source
- **Evolutionary growth**
  - 1U-PC cluster front-ends scale, evolve well
  - back-end scalability difficult, operator intensive

ROC Part I: Failure Data
Lessons about human operators

- **Human error is largest single failure source**
  - HP HA labs: human error is #1 cause of failures (2001)
  - Oracle: half of DB failures due to human error (1999)
  - Gray/Tandem: 42% of failures from human administrator errors (1986)
  - Murphy/Gent study of VAX systems (1993):



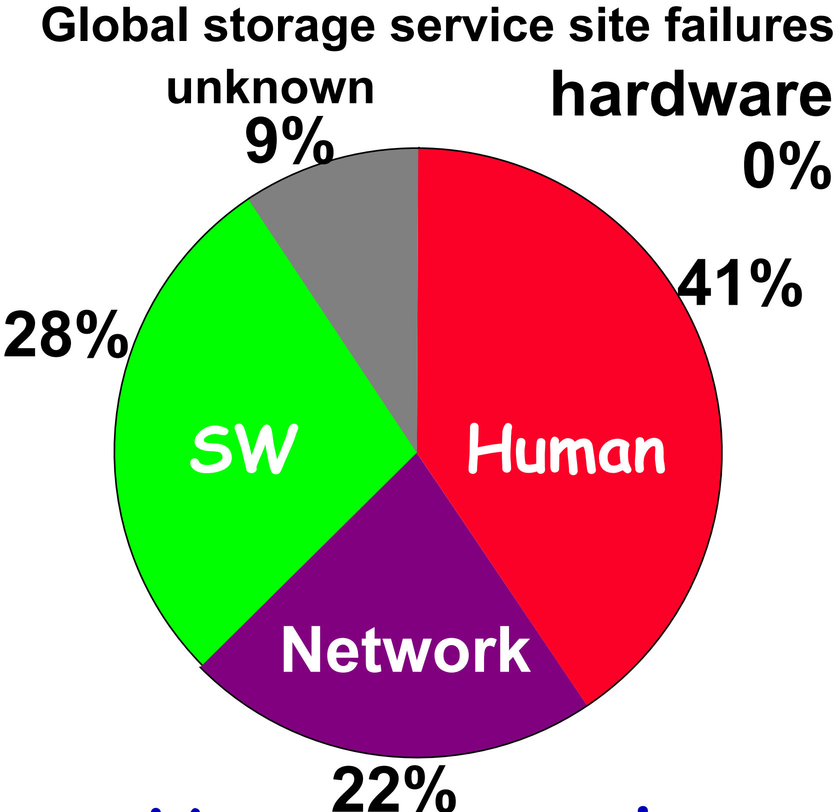
Blocked Calls: PSTN in 2000

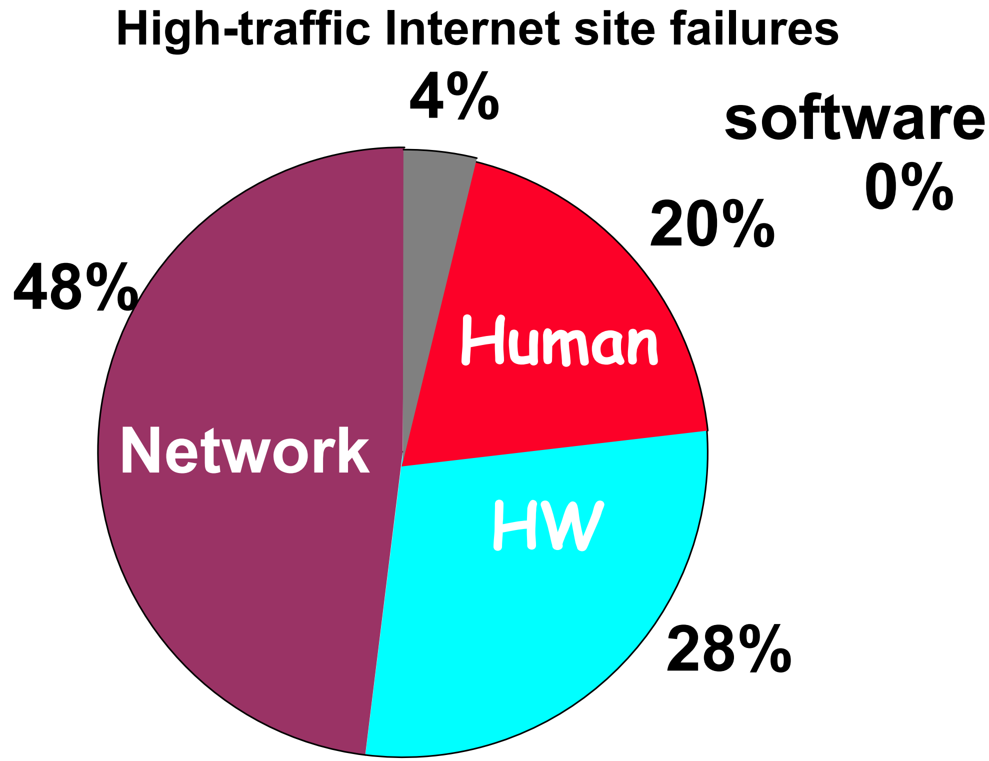


Human error accounts for **59%** of all blocked calls

Source: Patty Enriquez, U.C. Berkeley, in progress.

Internet Site Failures





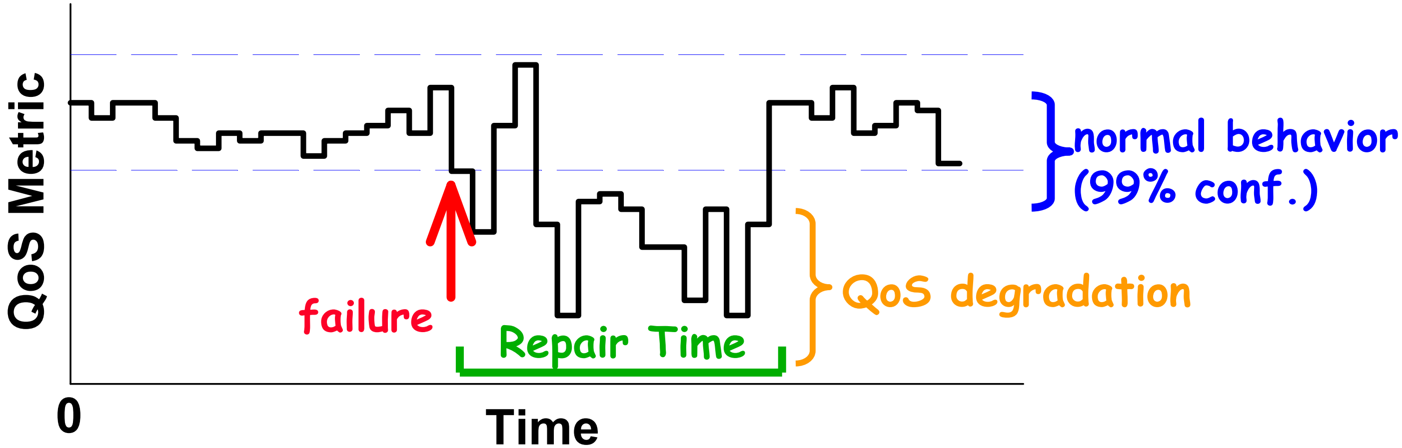
- Human error largest cause of failure in the more complex service, significant in both
- Network problems largest cause of failure in the less complex service, significant in both

ROC Part 2: ACME benchmarks

- **Traditional benchmarks focus on performance**
  - ignore ACME goals
  - assume perfect hardware, software, human operators
- **20th Century Winner: fastest on SPEC/TPC?**
- **21st Century Winner: fastest to recover from failure?**
- **New benchmarks needed to drive progress toward ACME, evaluate ROC success**
  - for example, *availability* and *recovery* benchmarks
  - How else convince developers, customers to adopt new technology?
  - How else enable researchers to find new challenges?

Availability benchmarking 101

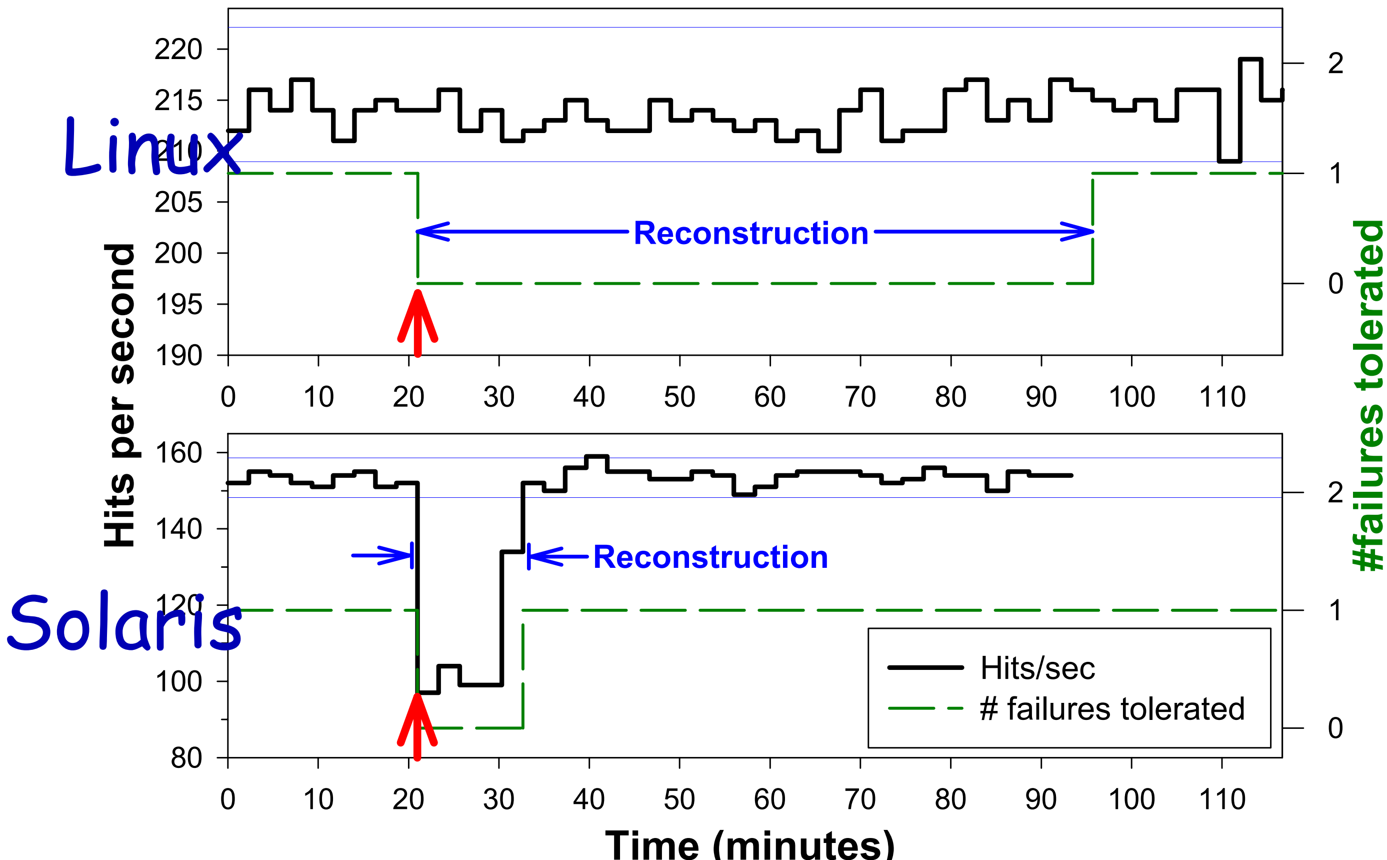
- **Availability benchmarks quantify system behavior under failures, maintenance, recovery**



- **They require**
  - A realistic workload for the system
  - Quality of service metrics and tools to measure them
  - Fault-injection to simulate failures
  - Human operators to perform repairs

Source: A. Brown, and D. Patterson, "Towards availability benchmarks: a case study of software RAID systems," *Proc. USENIX*, 18-23 June 2000

Example: 1 fault in SW RAID



- Compares Linux and Solaris reconstruction
  - **Linux:** minimal performance impact but longer window of vulnerability to second fault
  - **Solaris:** large perf. impact but restores redundancy fast
  - **Windows:** does not auto-reconstruct!

Automation vs. Aid?

- **Two approaches to helping**

1) Automate the entire process as a unit

- the goal of most research into "self-healing", "self-maintaining", "self-tuning", or more recently "introspective" or "autonomic" systems
- What about Automation Irony?

2) ROC approach: provide tools to let human SysAdmins perform job more effectively

- If desired, add automation as a layer on top of the tools
- What about number of SysAdmins as number of computers continue to increase?

A theory of human error

(distilled from J. Reason, Human Error, 1990)

- **Preliminaries: the three stages of cognitive processing for tasks**
  1) planning
     » a goal is identified and a sequence of actions is selected to reach the goal
  2) storage
     » the selected plan is stored in memory until it is appropriate to carry it out
  3) execution
     » the plan is implemented by the process of carrying out the actions specified by the plan

A theory of human error (2)

- **Each cognitive stage has an associated form of error**
  - **slips**: execution stage
    - » incorrect execution of a planned action
    - » example: miskeyed command
  - **lapses**: storage stage
    - » incorrect omission of a stored, planned action
    - » examples: skipping a step on a checklist, forgetting to restore normal valve settings after maintenance
  - **mistakes**: planning stage
    - » the plan is not suitable for achieving the desired goal
    - » example: TMI operators prematurely disabling HPI pumps

Origins of error: the GEMS model

- **GEMS: Generic Error-Modeling System**
  - an attempt to understand the origins of human error
- **GEMS identifies three *levels* of cognitive task processing**
  - **skill-based:** familiar, automatic procedural tasks
    - » usually low-level, like knowing to type "ls" to list files
  - **rule-based:** tasks approached by pattern-matching from a set of internal problem-solving rules
    - » "observed symptoms X mean system is in state Y"
    - » "if system state is Y, I should probably do Z to fix it"
  - **knowledge-based:** tasks approached by reasoning from first principles
    - » when rules and experience don't apply

GEMS and errors

- **Errors can occur at each level**
  - **skill-based:** slips and lapses
    - » usually errors of inattention or misplaced attention
  - **rule-based:** mistakes
    - » usually a result of picking an inappropriate rule
    - » caused by misconstrued view of state, over-zealous pattern matching, frequency gambling, deficient rules
  - **knowledge-based:** mistakes
    - » due to incomplete/inaccurate understanding of system, confirmation bias, overconfidence, cognitive strain, ...
- **Errors can result from operating at wrong level**
  - humans are reluctant to move from RB to KB level even if rules aren't working

Error frequencies

- **In raw frequencies, SB >> RB > KB**
  - 61% of errors are at skill-based level
  - 27% of errors are at rule-based level
  - 11% of errors are at knowledge-based level
- **But if we look at *opportunities* for error, the order reverses**
  - humans perform vastly more SB tasks than RB, and vastly more RB than KB
    - » so a given KB task is more likely to result in error than a given RB or SB task

Error detection and correction

- **Basic detection mechanism is self-monitoring**
  - periodic attentional checks, measurement of progress toward goal, discovery of surprise inconsistencies, ...
- **Effectiveness of self-detection of errors**
  - SB errors: 75-95% detected, avg 86%
    - » but some lapse-type errors were resistant to detection
  - RB errors: 50-90% detected, avg 73%
  - KB errors: 50-80% detected, avg 70%
- **Including correction tells a different story:**
  - SB: ~70% of all errors detected and corrected
  - RB: ~50% detected and corrected
  - KB: ~25% detected and corrected

What is Undo?

- **A system-wide ROC recovery mechanism**
  - designed to reduce MTTR
  - "time travel" for all system hard state: OS, app., user
- **A way to tolerate human operator error**
  - the leading cause of service downtime
- **A familiar recovery paradigm**
  - we use it every day in desktop productivity apps
    - » ROC is extending it to the system level
- **A way to increase synergy of operator-machine interaction**
  - matches human behavioral patterns

Motivation (2)

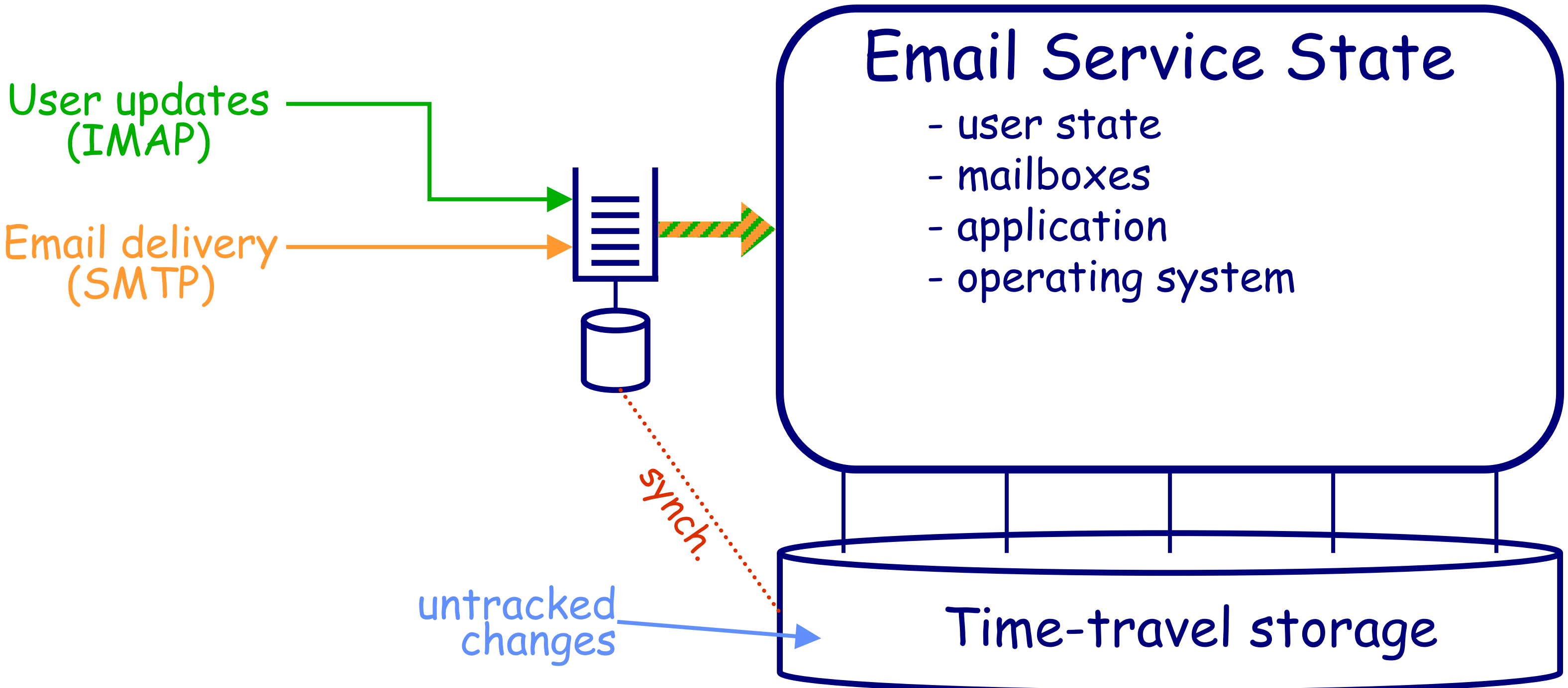
- **Undo “fringe benefits”**
  - makes sysadmin’s job easier, improving maintainability
    - » better maintainability => better dependability
  - enables trial-and-error learning
    - » builds sysadmin’s understanding of system
  - helps shift recovery burden from sysadmin to users
    - » export recovery to users via familiar undo model
    - » example: NetApp snapshots for file restores
  - helps recover from more than just human error
    - » SW/HW failure, security breaches, virus infections, ...

Towards system models for undo

- **Goal: abstract model for undo-capable system**
  - template for constructing undoable services
  - needed to analyze generality and limitations of undo
- **Model components**
  - state entities
  - state update events (analogue of transactions)
  - event queues and logs
  - untracked system changes
- **Assumptions**
  - storage layer that supports bidirectional time-travel
    - » via non-overwriting FS, snapshots, etc.
- **Email as example application**

Simple model

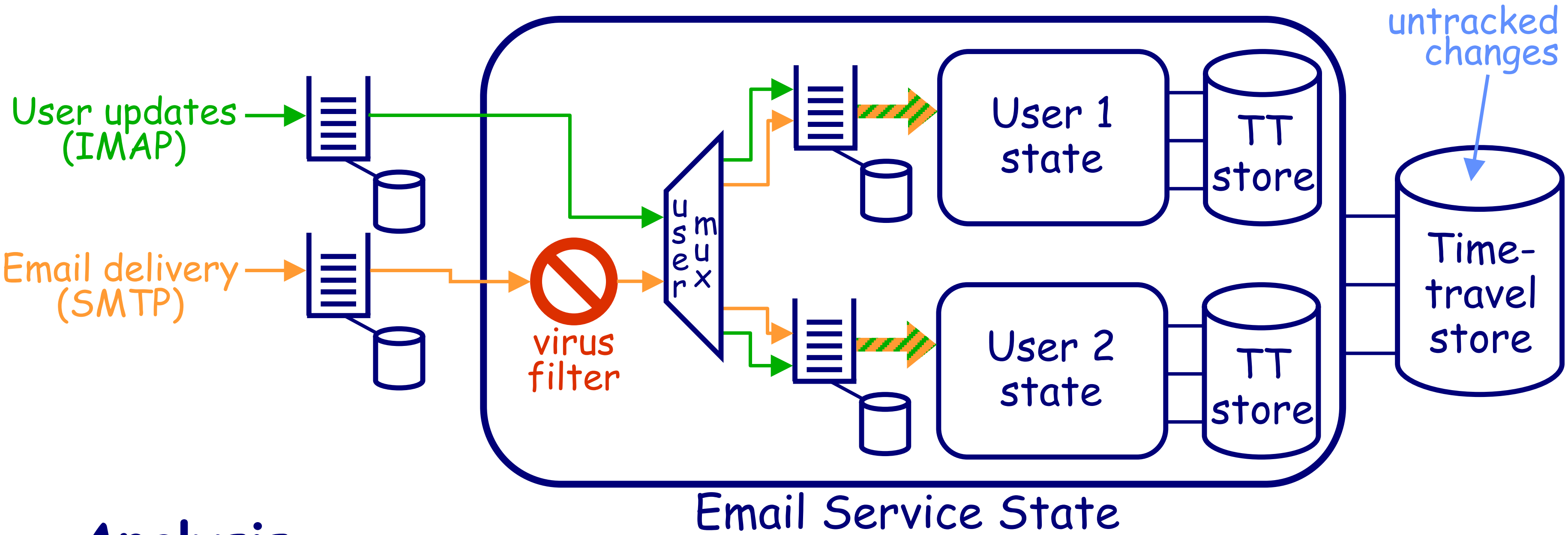
- **Entire system is one state entity**



- **Analysis**
  - + simple, easy to implement, easier to trust, most general
  - – huge overhead for fine-grained undo operations
  - – serialization bottleneck at single queue/log
  - – difficult to distinguish different users' events

Hierarchical model

- **System composed of multiple state entities**
  - each state entity supports undo as in simple model
  - state entities join hierarchically to give multiple granularities of undo



- **Analysis**
  - + multiple undo granularities reduces overhead, bottlenecks
  - + distributed undo possible
  - – greater complexity; tricky to coordinate different layers