

Managing Historical Retention in Database Systems

Gerome Miklau

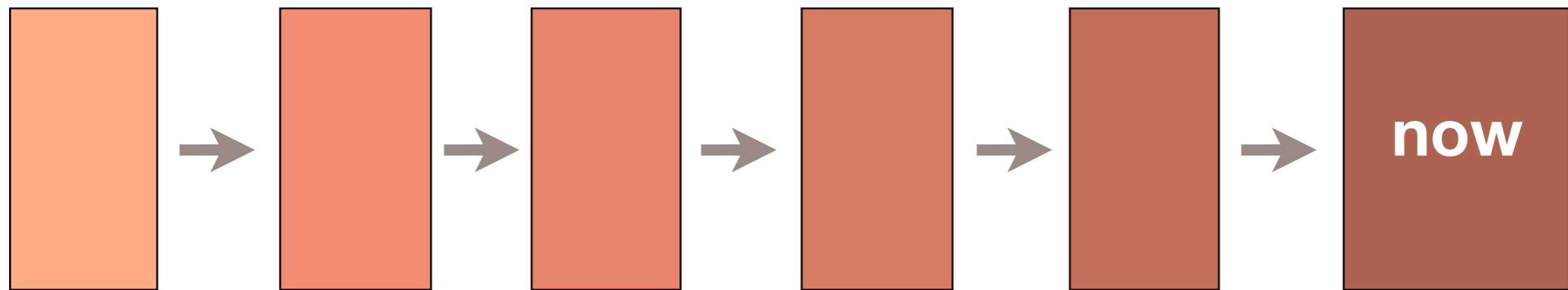
Joint work with Brian Levine, Patrick Stahlberg, Wentian Lu

University of Massachusetts, Amherst



History has benefits

History: a stored record of data and operations performed on a system.



- Arguments **for** preserving history
 - Protection against loss
 - History is useful: **accountability**
 - Storage is cheap

holding people/
programs responsible
for actions taken.

History has risks

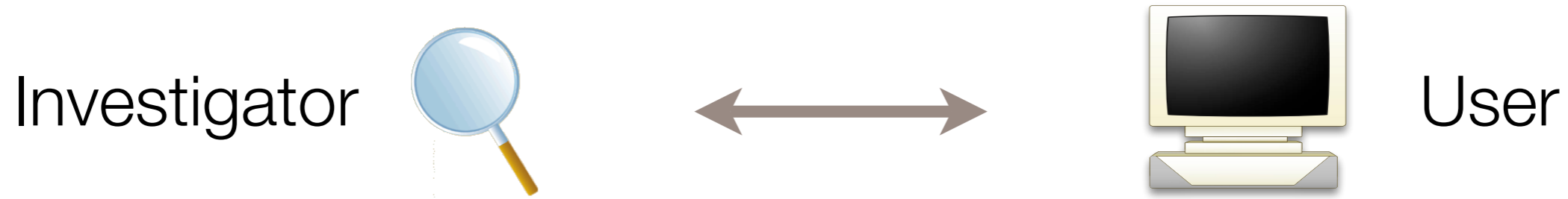
- Arguments **against** preserving history
 - Persistence threatens privacy.
 - Institutions can be compelled to reveal retained data (even if they don't want to).
 - There are significant benefits to **institutional forgetfulness.**

Retention policies

Privacy and **accountability** balanced through retention policies

| Institution | Collected Info | Retention Policy |
|---------------|----------------------------------|---|
| Russian KGB | speech, actions, etc. | хранить вечно “to be preserved forever” |
| Credit agency | late payments, defaults, etc. | 7 years |
| Google | search engine queries | 18 months |

Securing history



Central issue: how and when historical data is retained in systems, who can recover and analyze it.

- To support **privacy**: “memory-less” systems
- To support **accountability**: preserve needed history efficiently, permit analysis, protection mechanisms.

Databases don't forget

Unintentionally retained data is recovered by
forensic analysis.

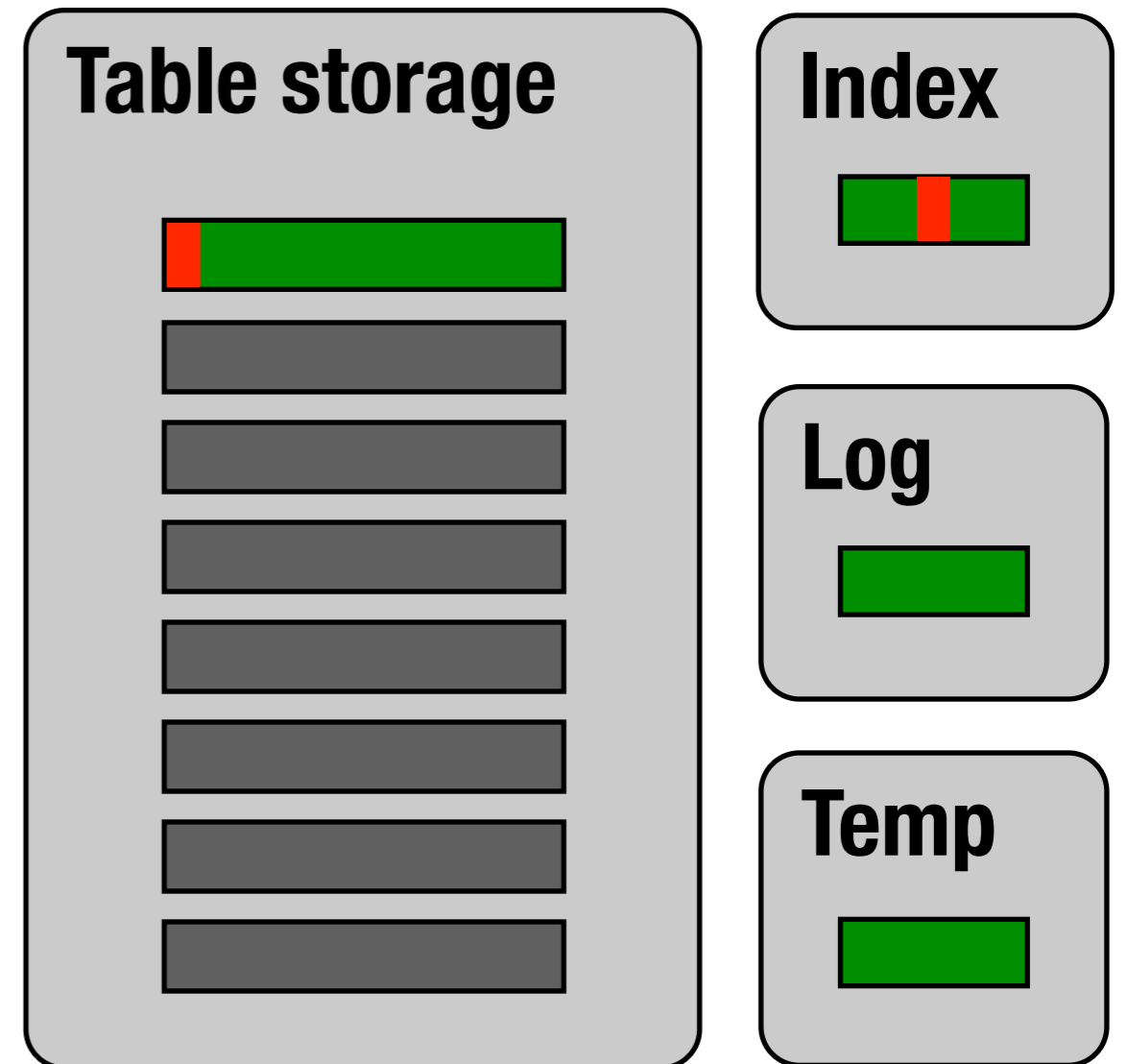
A forensic investigator is a powerful adversary:

- access to persistent storage at time t
- goal: recover expired data and/or history of operations

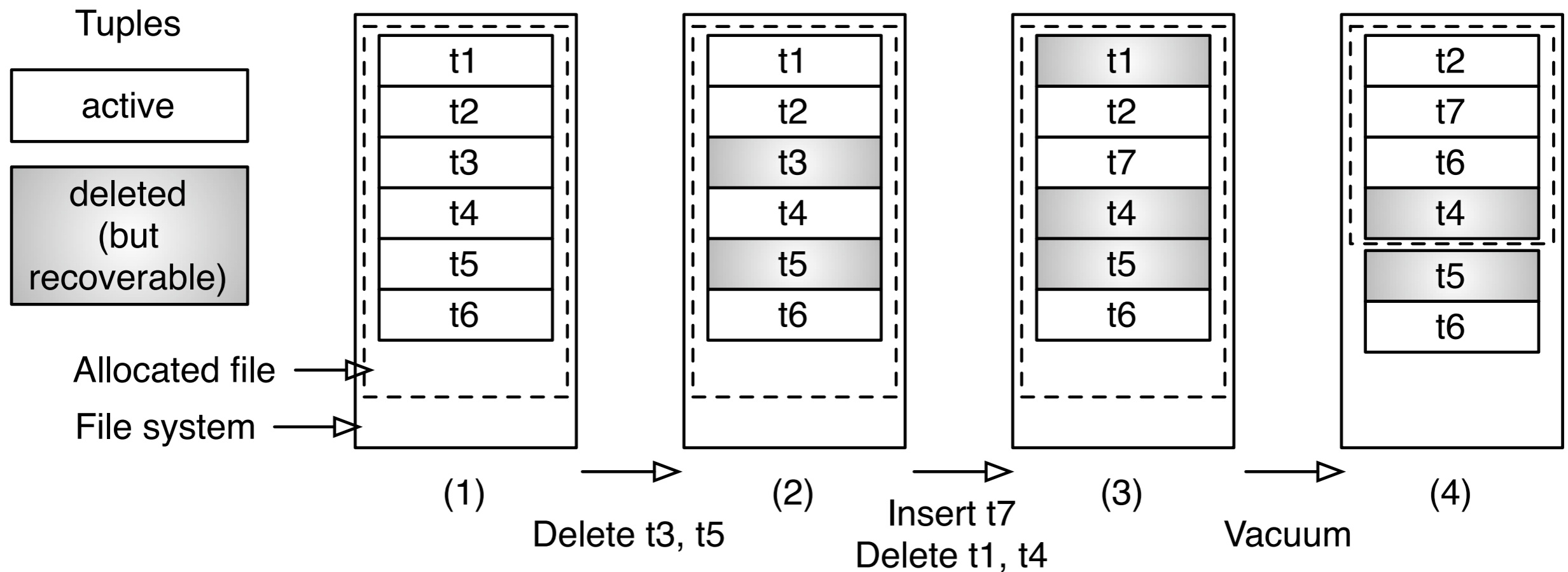
*(Threats to Privacy in the Forensic Analysis
of Database Systems. SIGMOD 2007)*

Propagation of sensitive data

- INSERT **sensitive record**
- (later) **DELETE** the record
 - deletion is “logical” -- data is not destroyed
- actual persistence of data is hard to predict, and virtually impossible to control.



Slack data in table storage



Deletion is insecure
Vacuum is insecure

Database slack
Filesystem slack

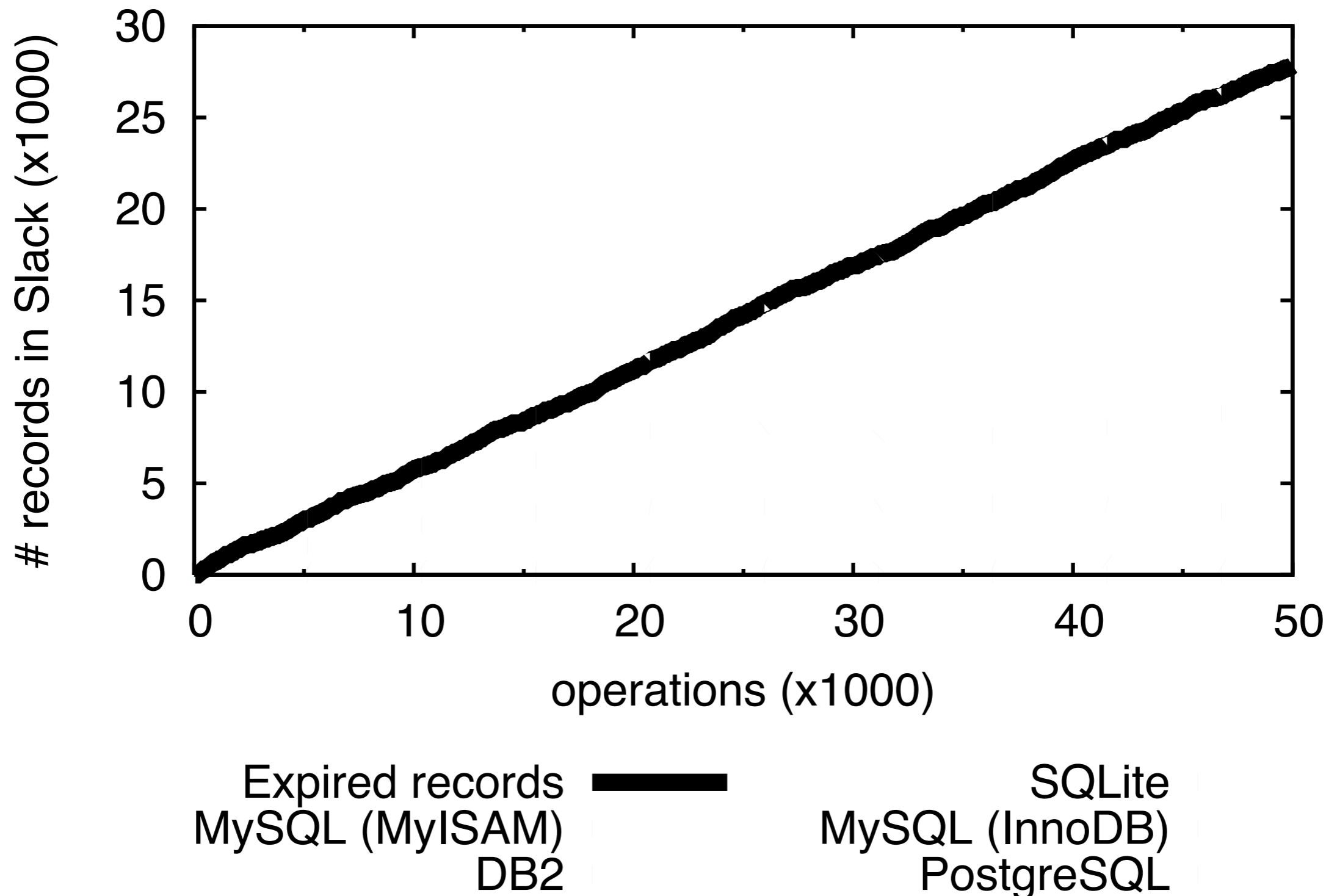
Experiments

- We studied:

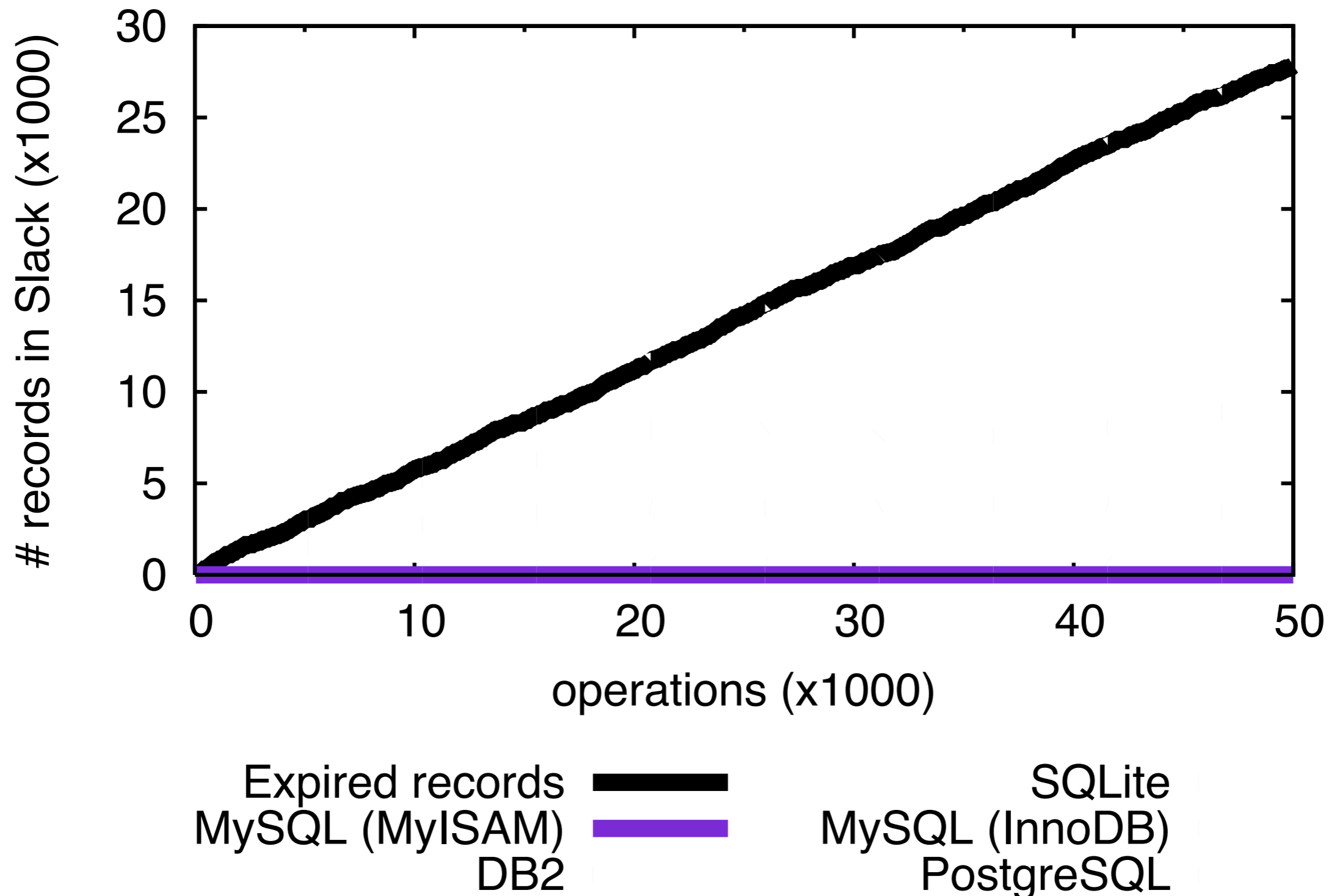


- Built forensic recovery tools which scan database pages, recovering expired tuples.
- Table storage
 - deletion is insecure in all systems
 - database and file system slack data generated in proportion to
 - workload, vacuum, clustering.

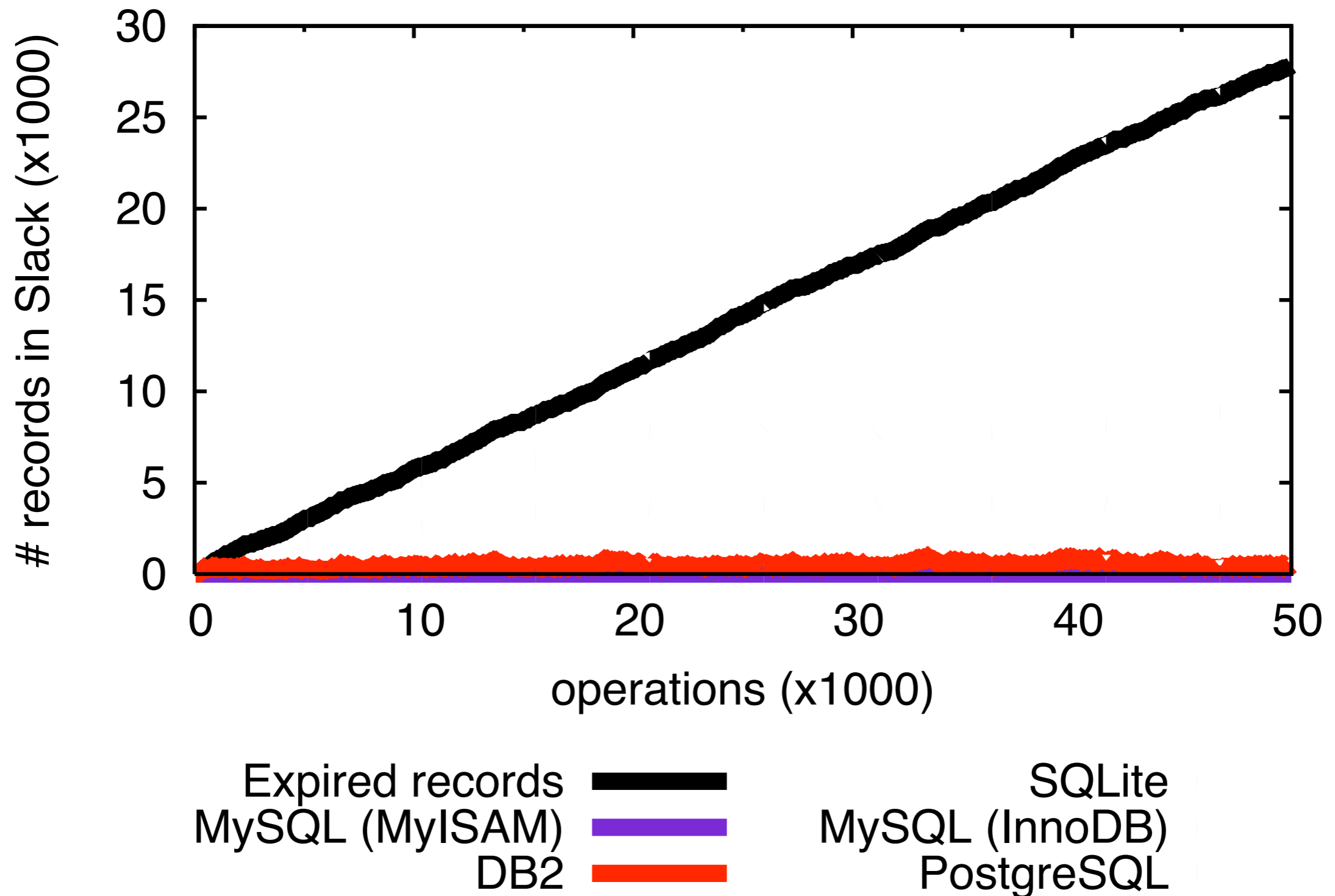
Recoverable database slack



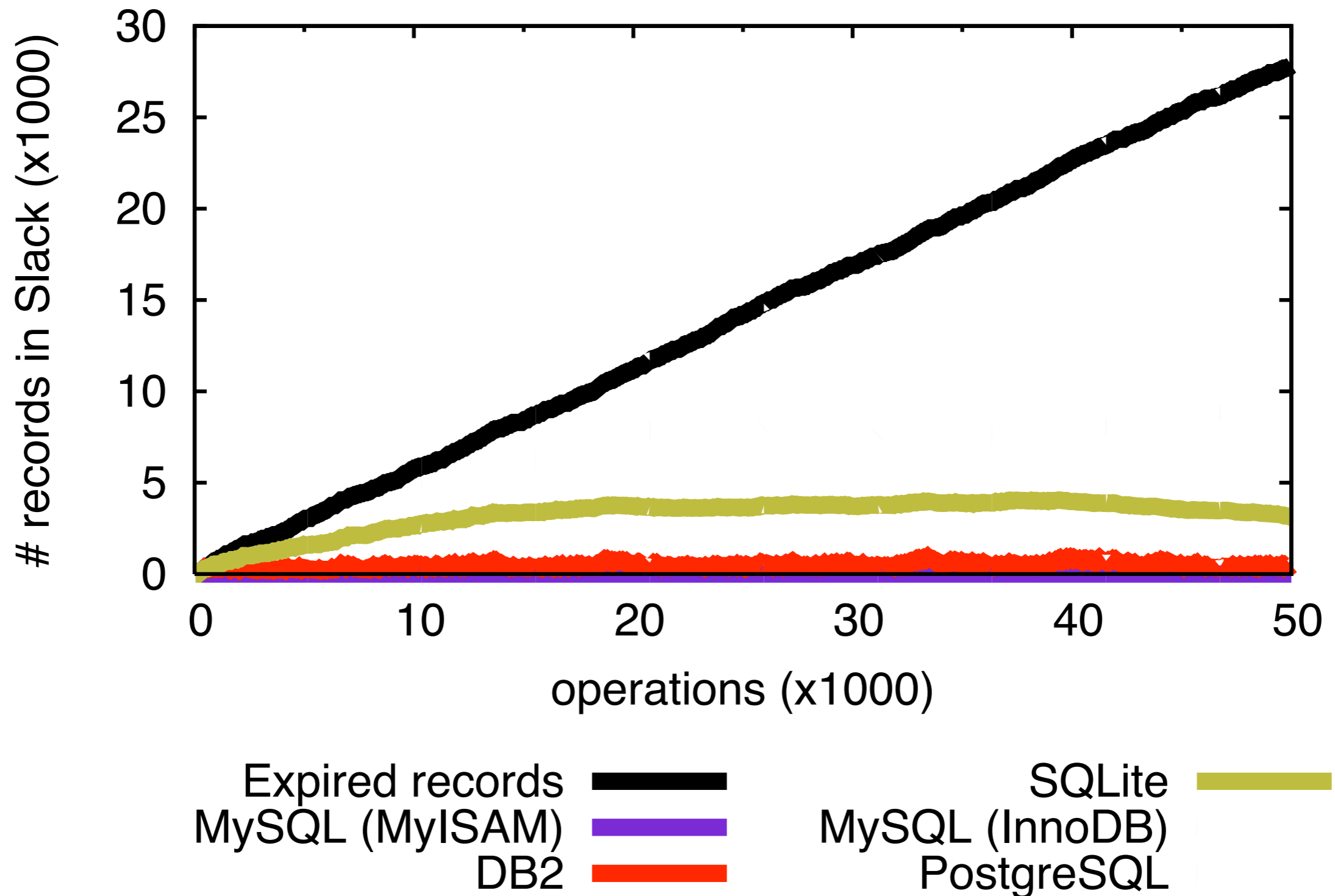
Recoverable database slack



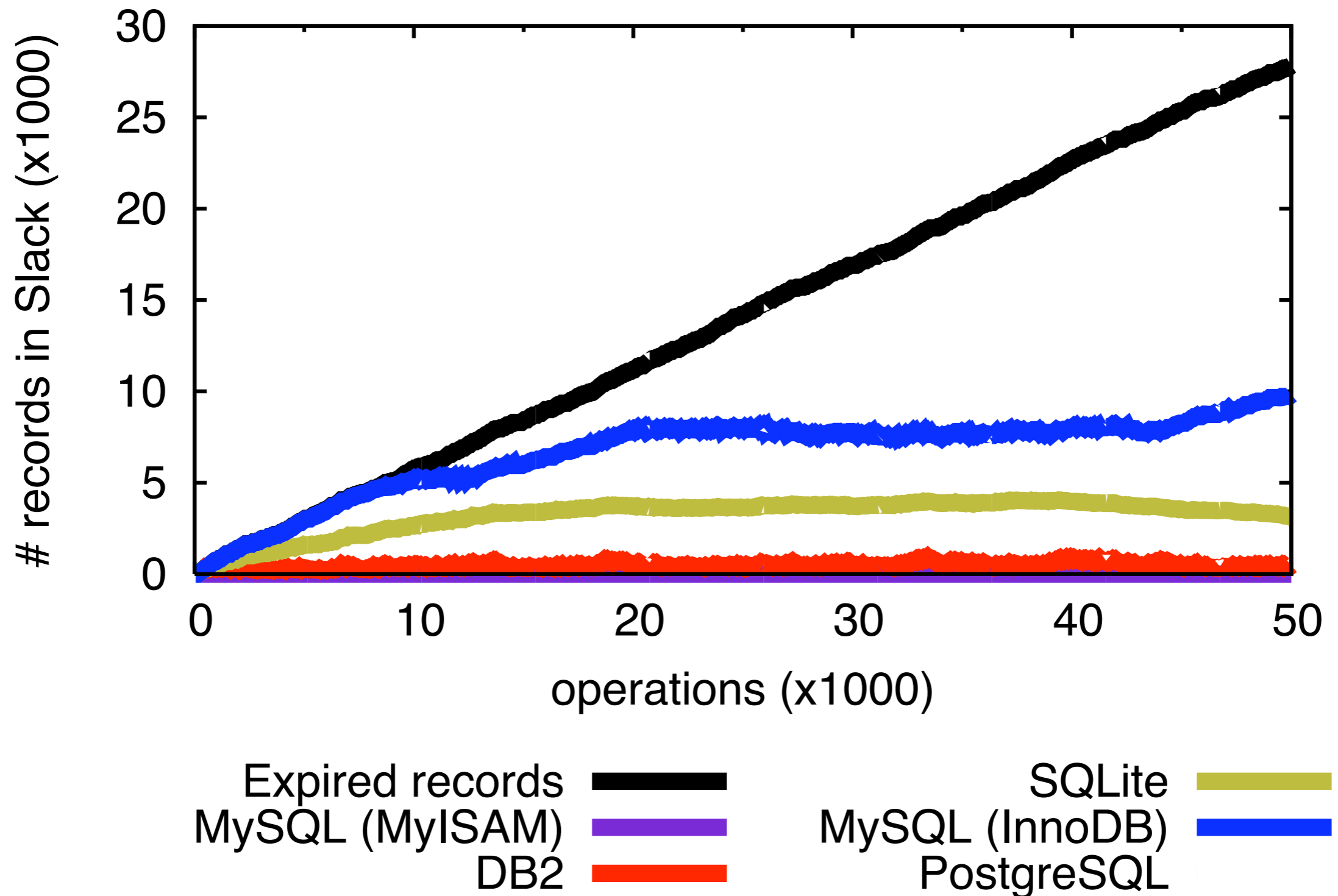
Recoverable database slack



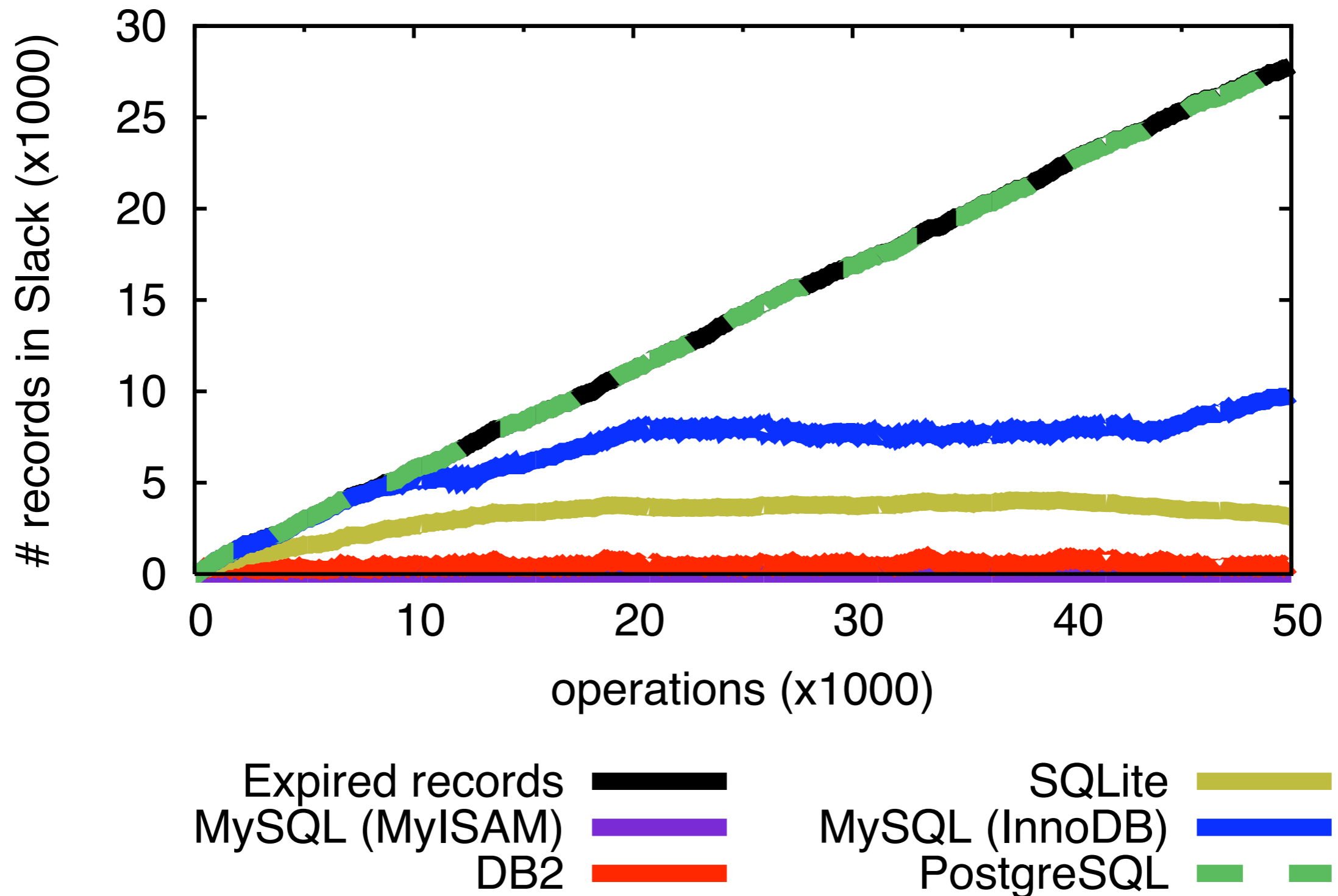
Recoverable database slack



Recoverable database slack



Recoverable database slack



Other system components

- **Indexes**

- Sequence of past operations that led to current state may be revealed by:
 - structure, physical representation (in memory or on disk)
- B+Trees are not history-independent

- **Transaction log**

- Log usually contains the before and after image of each DB modification
- Bounds on retention depend on:
 - workload, checkpointing frequency, size of log device, etc.

Problem with forensic data recovery

- Intended interface of database (SQL) does not reliably represent the stored contents of the database
 - e.g. deleted tuples do not appear in query results, but are recoverable.
 - tuples do not have “age” or order in data model, but this info can be recovered from disk image.

Transparent systems

Clarity of interfaces

- The system should provide users with clear, accurate bounds on the persistence of data in the system.

Purposeful retention

- Data retained after deletion must have a legitimate purpose, and data should be removed once that purpose is no longer valid.

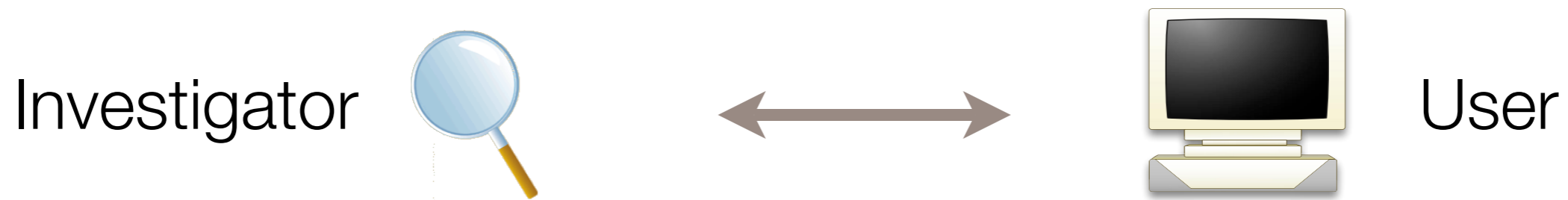
Complete removal

- Deleted data must be destroyed, including copies and derived versions.

Secure deletion in DBMS

- Two basic strategies for secure deletion:
 - overwrite data with zeroes
 - store data in encrypted form, delete by disposing of keys.
- For table storage:
 - pages are read and written often
 - prefer secure deletion and vacuum using overwriting
- For transaction log:
 - sequential writes, easily identifiable point of expiry
 - use encryption with key disposal

Databases can remember, but not safely



Who did what to the database, and when?

- Existing capabilities
 - Transaction logs, audit logs, point-in-time recovery
 - Postgres, temporal DBs, transaction-time DBs
- Limitations
 - Insufficient information retained, inefficient access
 - All-or-nothing protection model

Audit queries

- Audit the history of modifications to the database
- Note: we are not auditing database **reads**.
- For example:
 - What was Bob's lowest salary?
 - How many times was Bob's salary changed?
 - Who made the last update to Bob's salary?

A transaction-time data model

Audit Log

| User | Operation | | Time |
|------|-----------|-----------------------|------|
| Mary | Insert | (Bob,50k) | 1995 |
| Joe | Update | Bob salary=60k | 2000 |

Database

| Name | Salary | Start | End |
|------------|------------|-------|------|
| Bob | 50k | 1995 | 2000 |
| Bob | 60k | 2000 | 2008 |

- What was Bob's lowest salary? **50k**
- How many times was Bob's salary changed? **1**
- Who made the last update to Bob's salary? **Joe**

Retention policy

- Policies limiting retention require removing parts of history.
 - Expunge particular records, time periods, etc.
 - Redact records (by removing sensitive values)
 - Compress time periods by summarization

Example Policy:
Redact Bob's salary prior to 2002

- Intuition: we shouldn't need to know the value of Bob's salary to perform interesting audit queries.

Transforming history

| User | Operation | | Time |
|------|-----------|-----------------------|------|
| Mary | Insert | (Bob,50k) | 1995 |
| Joe | Update | Bob salary=60k | 2000 |



| User | Operation | | Time |
|------|-----------|-----------------------|----------|
| Mary | Insert | (Bob,??) | 1995 |
| Joe | Update | Bob salary=60k | T |

for **T** in (1995,2002)

| Name | Salary | Start | End |
|------------|------------|-------|------|
| Bob | 50k | 1995 | 2000 |
| Bob | 60k | 2000 | 2008 |



| Name | Salary | Start | End |
|------------|-------------|----------|----------|
| Bob | NULL | 1995 | T |
| Bob | 60k | T | 2008 |

- What was Bob's lowest salary?
- How many times was Bob's salary changed?
- Who made the last update to Bob's salary?

unknown

1

Joe

Challenges

- A representation system for an **incomplete** audit log
- Answering audit queries over incomplete history
- Deeper issue: how much information can we preserve for accountability, while achieving the privacy goals of the retention policy?

- Note that temporal incompleteness also occurs when:
 - Auditor has imperfect observations of the past.
 - Efficiency concerns mean we can't store everything.

Conclusion

- History should be a “first-class” part of a DBMS
- The safe, accurate configuration of the system’s historical memory allows needed balance between **privacy** and **accountability**.
- Transparency requirements:
 - Interface should faithfully represent stored contents.
- Auditing and retention:
 - Techniques to sanitize history while preserving auditing capabilities.

Questions?