Carving Database Storage to Detect and Trace Security Breaches

James Wagner, Alexander Rasin, Boris Glavic, Karen Heart, Jacob Furst, Lucas Bressan, Jonathan Grier

DEPAUL

GRIER FORENSICS

ILLINOIS INSTITUTE OF TECHNOLOGY

DePaul Database Research Group

Alexander Rasin

Boris Glavic

Karen Heart

Jacob Furst

Lucas Bressan

Jonathan Grier

Database Attacks by Privileged Users

- Sensitive data is commonly stored in Database Management Systems(DBMS)
  - Social Networks (e.g., Facebook or Twitter)
  - Ecommerce (e.g., Uber)
  - Media (e.g., Netflix or Spotify)
  - Banks (e.g., JP Morgan Chase)
  - Healthcare (e.g., Bayer)
  - Government (e.g., IL Department of Revenue)
- How do you protect your database against insider attacks?

Motivation: Malicious Administrators

Security Approaches Against Insiders

- Defense (e.g., access control) vs. Detection (e.g., audit logs)
- DBMSes maintain a history of SQL queries in an audit log.
- DBMSes can't guarantee that audits logs are accurate.

Reliability of DBMS Audit Logs

- Log integrity verification with 3rd party tools
- What if logging was disabled?
  - DBAs have the legitimate privilege to disable logging.

Goal: Detect activity missing from log files.

- This will be done by carving storage artifacts.

Malicious DBA Example

1. Alex is a DBA for a government agency that keeps track of criminal records.

DBA

Database Storage

Del. Flag	Page Type: Table Table: CrimeReport
✓	Jonathan, 2005, piracy
✓	Karen, 2007, fraud
✓	Boris, 2012, shoplifting

Audit Log File

```
T1, INSERT INTO CrimeReport
      VALUES ('Jonathan', 2005, 'piracy');
T2, INSERT INTO CrimeReport
      VALUES ('Karen', 2007, 'fraud');
T3, UPDATE CrimeReport
      SET Crime = 'shoplifting'
      WHERE Name = 'Boris';
```

Malicious DBA Example

2. Jonathan tells Alex he'll pay him to erase his criminal record.

Criminal

DBA

Database Storage

Del. Flag	Page Type: Table Table: CrimeReport
✓	Jonathan, 2005, piracy
✓	Karen, 2007, fraud
✓	Boris, 2012, shoplifting

Audit Log File

```
T1, INSERT INTO CrimeReport
      VALUES ('Jonathan', 2005, 'piracy');
T2, INSERT INTO CrimeReport
      VALUES ('Karen', 2007, 'fraud');
T3, UPDATE CrimeReport
      SET Crime = 'shoplifting'
      WHERE Name = 'Boris';
```

Malicious DBA Example

3. Alex agrees to accept Jonathan’s bribe.
   A. Disables logging
   B. Deletes Jonathan’s criminal record
   C. Re-enables logging

Malicious DBA

Database Storage

Del. Flag	Page Type: Table Table: CrimeReport
✗	Jonathan, 2005, piracy
✓	Karen, 2007, fraud
✓	Boris, 2012, shoplifting

Audit Log File

```
T1, INSERT INTO CrimeReport
      VALUES (‘Jonathan’, 2005, ‘piracy’);
T2, INSERT INTO CrimeReport
      VALUES (‘Karen’, 2007, ‘fraud’);
T3, UPDATE CrimeReport
      SET Crime = ‘shoplifting’
      WHERE Name = ‘Boris’;
```

Malicious DBA Example

4. No one is aware that Jonathan's criminal history has been deleted.
    - No evidence in the audit log
    - Deleted records can not be queried. Example:

SELECT *
~~FROM~~ CrimeReport
WHERE *Record IS Deleted*

Good Citizen

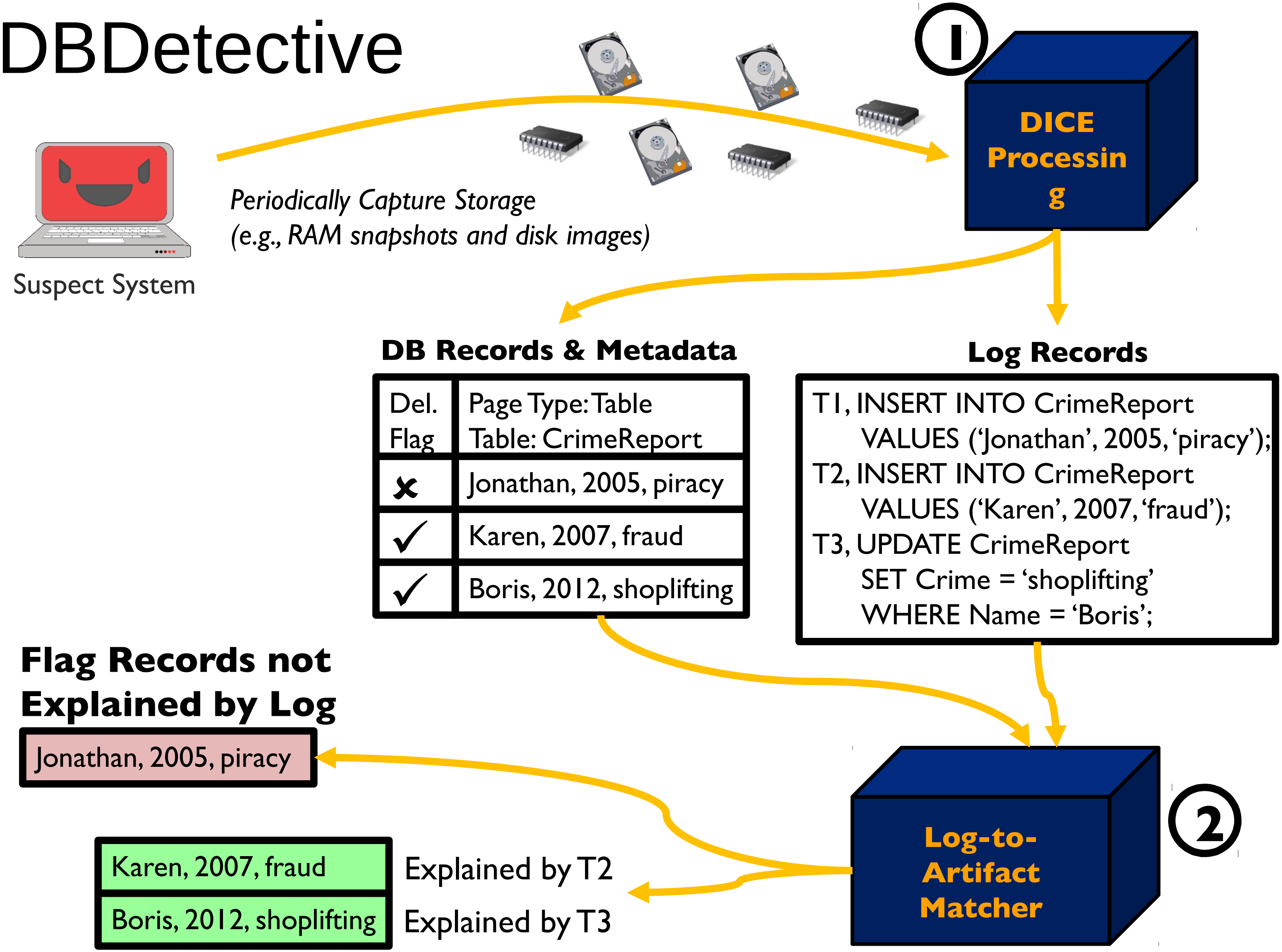
Database Storage

Del. Flag	Page Type: Table Table: CrimeReport
✗	Jonathan, 2005, piracy
✓	Karen, 2007, fraud
✓	Boris, 2012, shoplifting

Audit Log File

```
T1, INSERT INTO CrimeReport
      VALUES ('Jonathan', 2005, 'piracy');
T2, INSERT INTO CrimeReport
      VALUES ('Karen', 2007, 'fraud');
T3, UPDATE CrimeReport
      SET Crime = 'shoplifting'
      WHERE Name = 'Boris';
```

DBDetective



Periodically Capture Storage
(e.g., RAM snapshots and disk images)

DICE
Processin
g

DB Records & Metadata

Del. Flag	Page Type: Table Table: CrimeReport
✗	Jonathan, 2005, piracy
✓	Karen, 2007, fraud
✓	Boris, 2012, shoplifting

Log Records

```
T1, INSERT INTO CrimeReport
      VALUES ('Jonathan', 2005, 'piracy');
T2, INSERT INTO CrimeReport
      VALUES ('Karen', 2007, 'fraud');
T3, UPDATE CrimeReport
      SET Crime = 'shoplifting'
      WHERE Name = 'Boris';
```

Log-to-
Artifact
Matcher

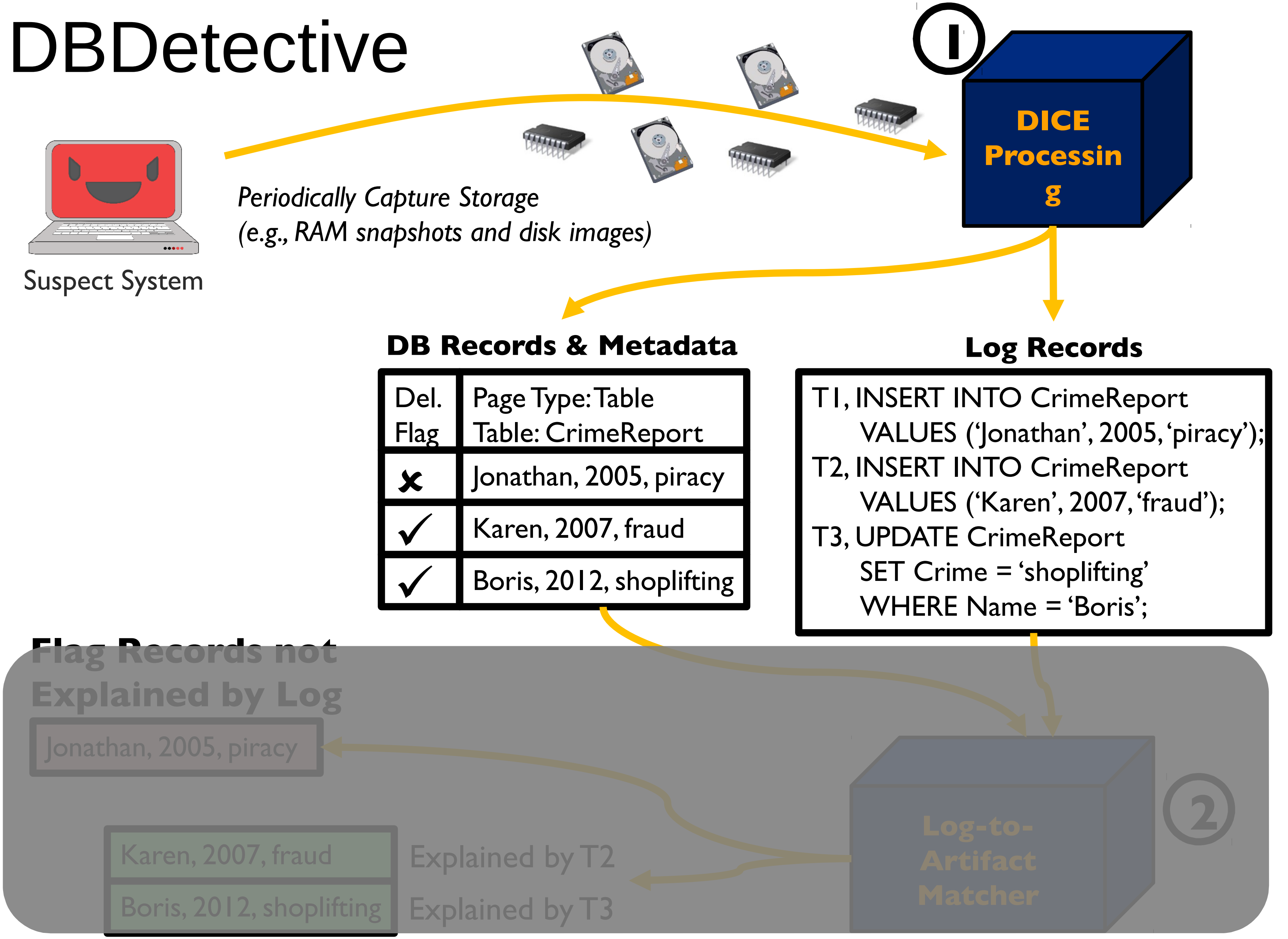
Flag Records not Explained by Log

Jonathan, 2005, piracy

Karen, 2007, fraud — Explained by T2

Boris, 2012, shoplifting — Explained by T3

DBDetective



Periodically Capture Storage
(e.g., RAM snapshots and disk images)

DB Records & Metadata

Del. Flag	Page Type: Table Table: CrimeReport
✗	Jonathan, 2005, piracy
✓	Karen, 2007, fraud
✓	Boris, 2012, shoplifting

Log Records

```
T1, INSERT INTO CrimeReport
      VALUES ('Jonathan', 2005, 'piracy');
T2, INSERT INTO CrimeReport
      VALUES ('Karen', 2007, 'fraud');
T3, UPDATE CrimeReport
      SET Crime = 'shoplifting'
      WHERE Name = 'Boris';
```

Flag Records not Explained by Log

Jonathan, 2005, piracy

Karen, 2007, fraud — Explained by T2

Boris, 2012, shoplifting — Explained by T3

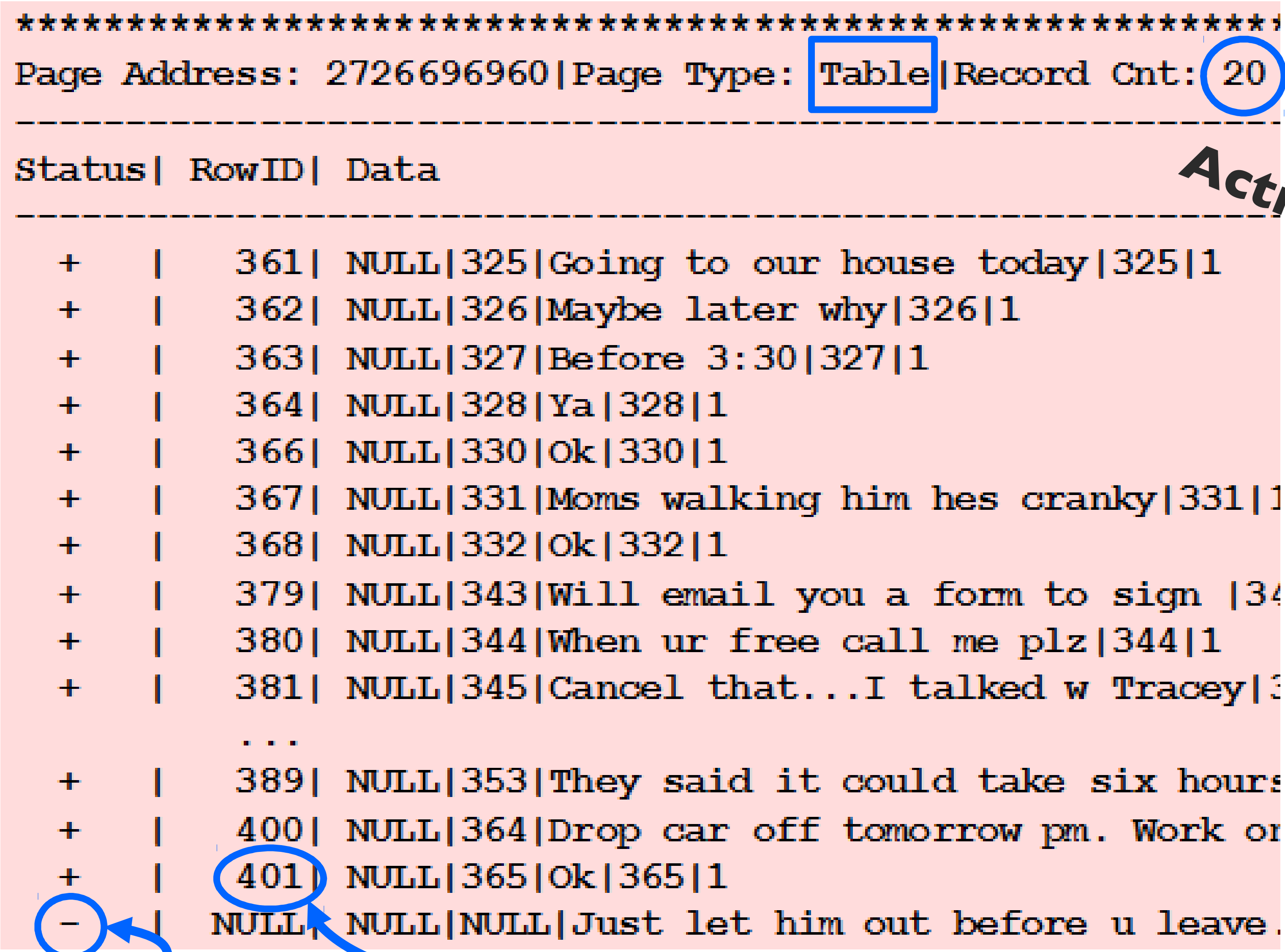
Database Image Content Explorer: DICE

- Implementation of **database page carving**

- Database page carving:
  - a solution to file carving for database files
  - reconstructs the database at the page level
  - returns records, indexes, metadata, and other artifacts
  - database files, disk images, or RAM snapshots

- Previously presented at DFRWS USA 2015 & 2016:
  - J. Wagner, A. Rasin, J. Grier, Database Forensic Analysis through Internal Structure Carving
  - J. Wagner, A. Rasin, J. Grier, Database Image Content Explorer: Carving Data that does not Officially Exist

DICE Example: Android Phone Data



```
*******************************************************
Page Address: 2726696960|Page Type: Table|Record Cnt: 20
-------------------------------------------------------
Status| RowID| Data
-------------------------------------------------------
  +   |   361| NULL|325|Going to our house today|325|1
  +   |   362| NULL|326|Maybe later why|326|1
  +   |   363| NULL|327|Before 3:30|327|1
  +   |   364| NULL|328|Ya|328|1
  +   |   366| NULL|330|Ok|330|1
  +   |   367| NULL|331|Moms walking him hes cranky|331|1
  +   |   368| NULL|332|Ok|332|1
  +   |   379| NULL|343|Will email you a form to sign |34
  +   |   380| NULL|344|When ur free call me plz|344|1
  +   |   381| NULL|345|Cancel that...I talked w Tracey|3
             ...
  +   |   389| NULL|353|They said it could take six hours
  +   |   400| NULL|364|Drop car off tomorrow pm. Work on
  +   |   401| NULL|365|Ok|365|1
  -   |  NULL| NULL|NULL|Just let him out before u leave.
```

Active Rows

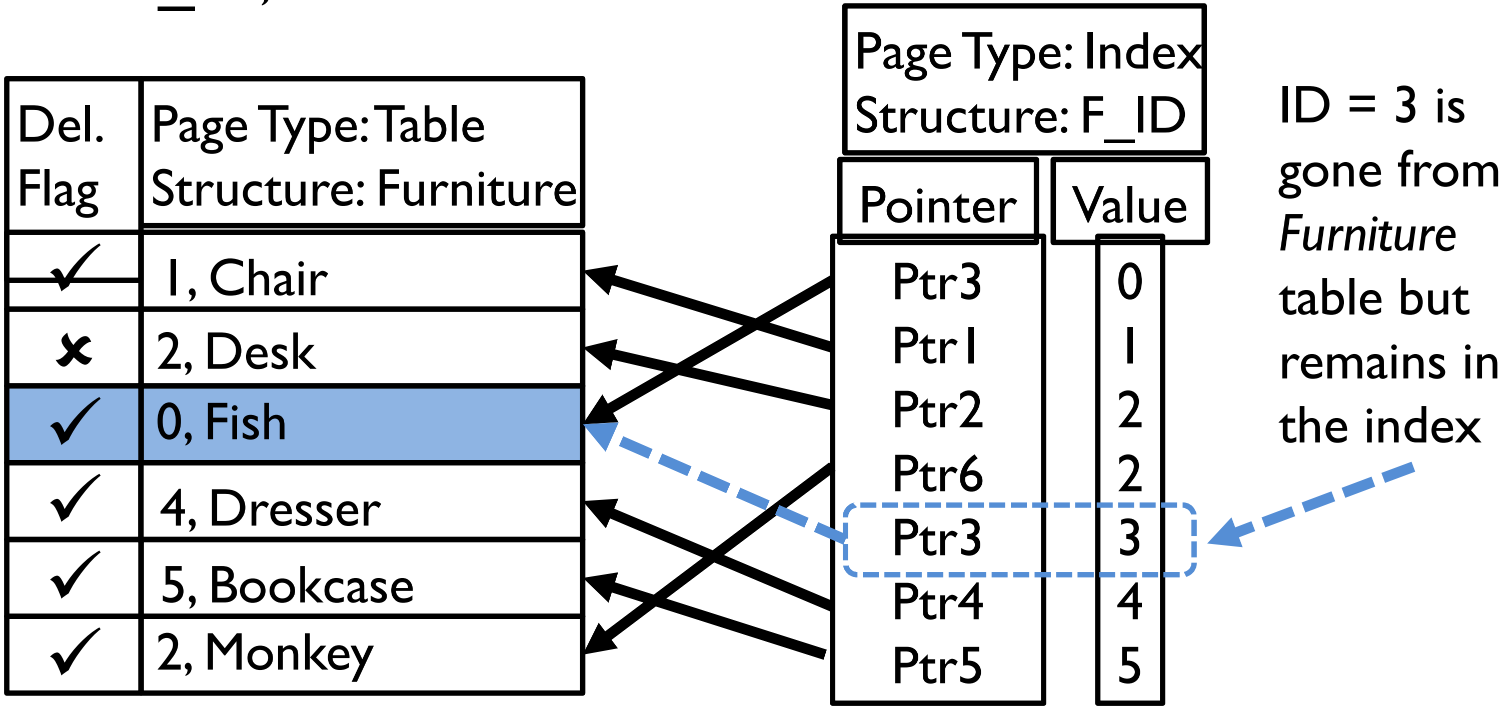
Deleted Row

Internal RowID (not part of user data)

DICE Example: Carving Indexes

- Indexes store value-pointer pairs used to read specific table pages
- Deleted index values often have a longer lifetime than deleted table records.
- Indexes can’t be directly queried. Ex.

✗
```
SELECT *
FROM F_ID;
```



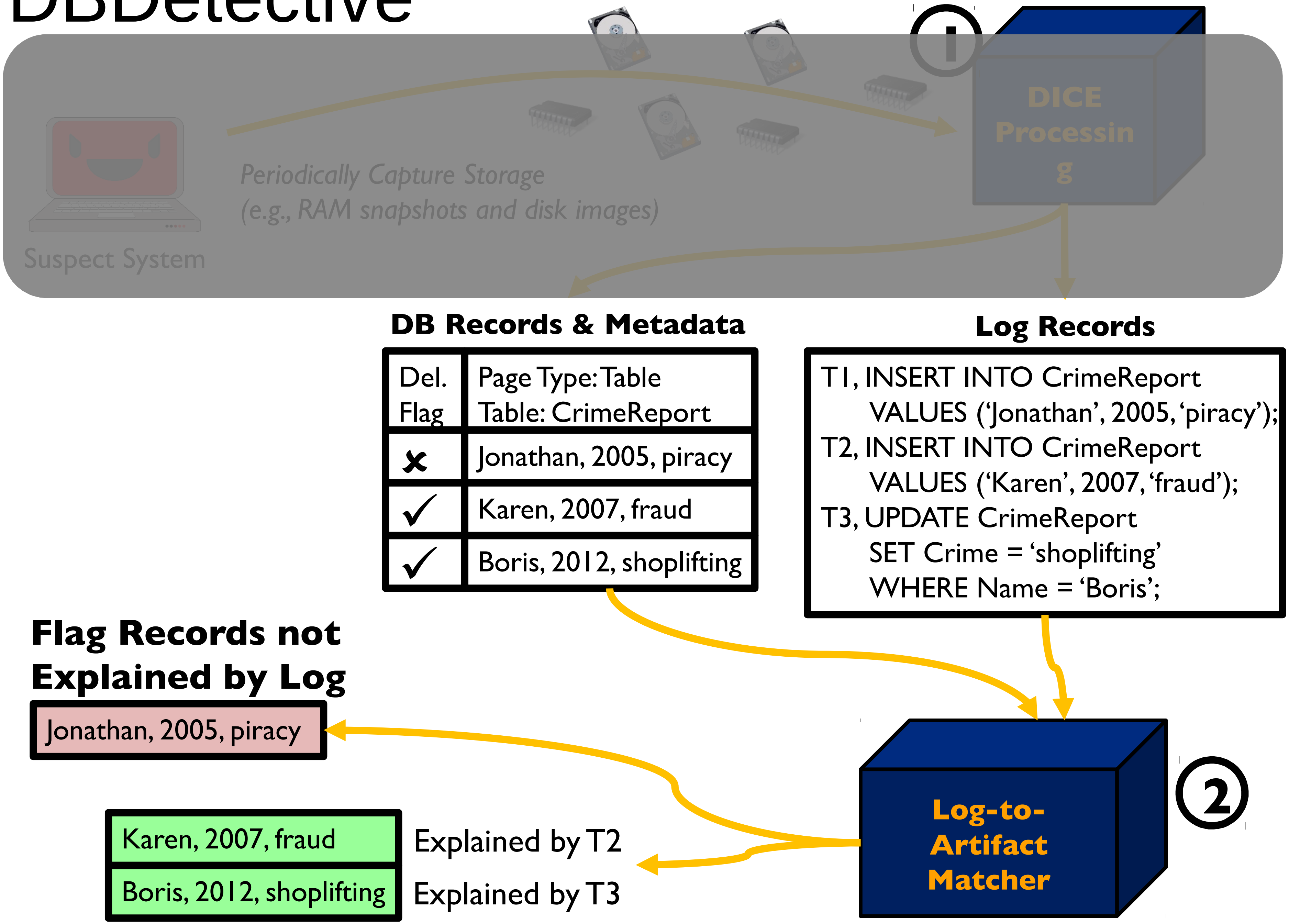
Del. Flag	Page Type: Table Structure: Furniture
✓	1, Chair
✗	2, Desk
✓	0, Fish
✓	4, Dresser
✓	5, Bookcase
✓	2, Monkey

Page Type: Index
Structure: F_ID

Pointer	Value
Ptr3	0
Ptr1	1
Ptr2	2
Ptr6	2
Ptr3	3
Ptr4	4
Ptr5	5

ID = 3 is gone from *Furniture* table but remains in the index

DBDetective



Suspect System

Periodically Capture Storage
(e.g., RAM snapshots and disk images)

① DICE Processin g

DB Records & Metadata

Del. Flag	Page Type: Table Table: CrimeReport
✗	Jonathan, 2005, piracy
✓	Karen, 2007, fraud
✓	Boris, 2012, shoplifting

Log Records

```
T1, INSERT INTO CrimeReport
      VALUES ('Jonathan', 2005, 'piracy');
T2, INSERT INTO CrimeReport
      VALUES ('Karen', 2007, 'fraud');
T3, UPDATE CrimeReport
      SET Crime = 'shoplifting'
      WHERE Name = 'Boris';
```

② **Log-to-Artifact Matcher**

Flag Records not Explained by Log

Jonathan, 2005, piracy

Karen, 2007, fraud	Explained by T2
Boris, 2012, shoplifting	Explained by T3

Log-to-Artifact Matcher

- Evaluate the integrity of carved data and metadata using log entries.

Data modifications (disk images):

1. DELETE

2. INSERT

3. UPDATE

★ **D**ata **D**efinition **L**anguage (e.g., CREATE, ALTER, DROP)

Read-only queries (RAM snapshots):

4. SELECT

★ Read only queries do not leave evidence on disk

Deleted Record-to-Log Matching

- Only the deleted records from the DICE output need to be considered.
- Only the DELETE commands from the log are considered.

DICE Output

Del. Flag	Page Type: Table Structure: Customer
✗	1, Christine, Chicago
✓	2, George, New York
✗	3, Christopher, Seattle
✗	4, Thomas, Austin
✓	5, Mary, Boston

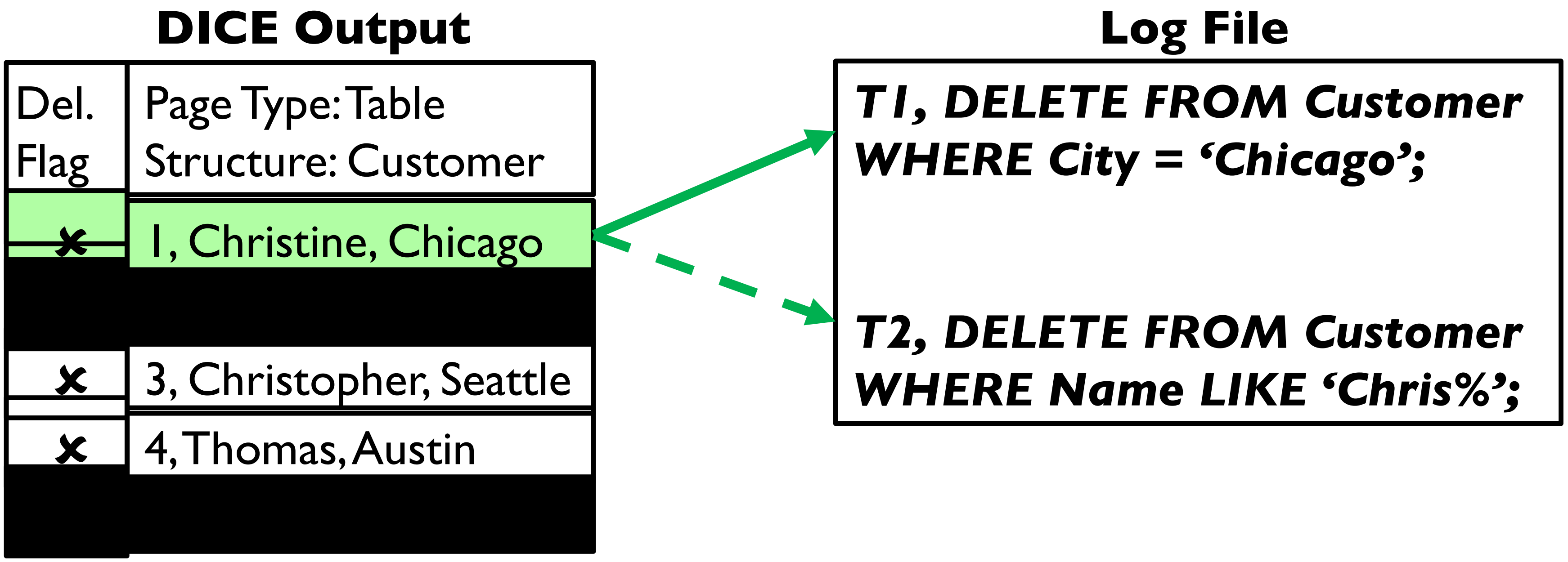
Log File

T1, DELETE FROM Customer WHERE City = ‘Chicago’;

T2, DELETE FROM Customer WHERE Name LIKE ‘Chris%’;

Deleted Record-to-Log Matching

- Any command that explains a record is considered, not the specific command.
- Name LIKE ‘Chris%’ vs City = ‘Chicago’ ➡ (1, Christine, Chicago)



DICE Output

Del. Flag	Page Type: Table Structure: Customer
✗	1, Christine, Chicago
[redacted]	[redacted]
✗	3, Christopher, Seattle
✗	4, Thomas, Austin
[redacted]	[redacted]

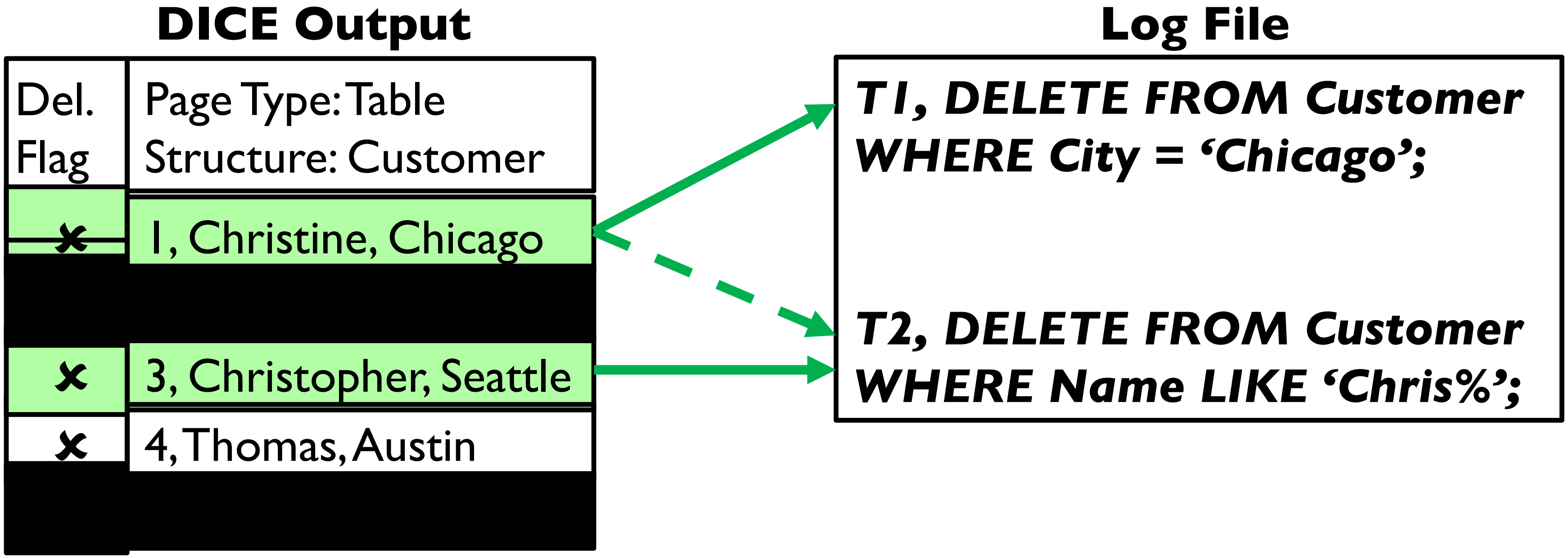
Log File

T1, DELETE FROM Customer WHERE City = ‘Chicago’;

T2, DELETE FROM Customer WHERE Name LIKE ‘Chris%’;

Deleted Record-to-Log Matching

- Name LIKE 'Chris%' explains the deleted record (3, Christopher, Seattle)



DICE Output

Del. Flag	Page Type: Table Structure: Customer
✗	1, Christine, Chicago
[redacted]	[redacted]
✗	3, Christopher, Seattle
✗	4, Thomas, Austin
[redacted]	[redacted]

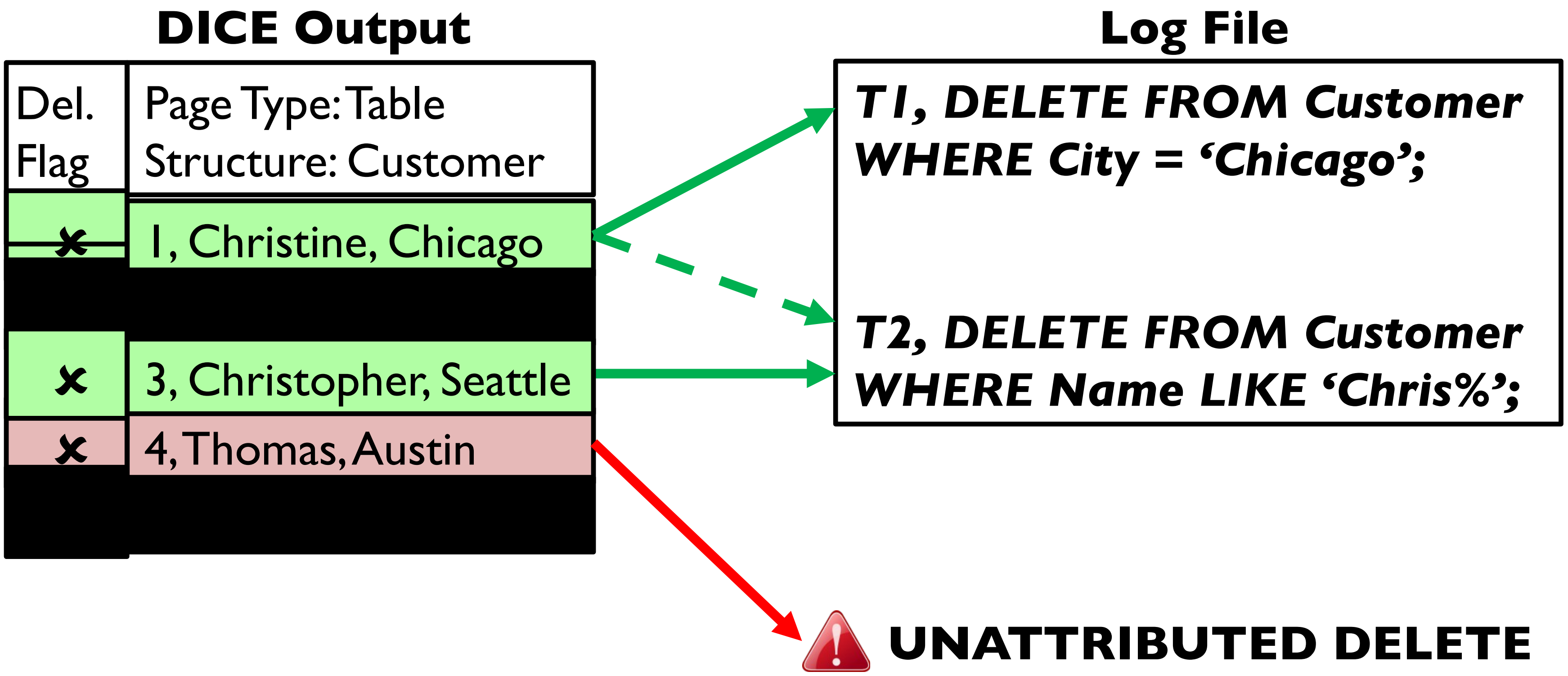
Log File

T1, DELETE FROM Customer WHERE City = 'Chicago';

T2, DELETE FROM Customer WHERE Name LIKE 'Chris%';

Deleted Record-to-Log Matching

- None of the DELETE commands can explain the deleted record (4, Thomas, Austin)



DICE Output

Del. Flag	Page Type: Table Structure: Customer
✗	1, Christine, Chicago
[redacted]	[redacted]
✗	3, Christopher, Seattle
✗	4, Thomas, Austin
[redacted]	[redacted]

Log File

T1, DELETE FROM Customer WHERE City = 'Chicago';

T2, DELETE FROM Customer WHERE Name LIKE 'Chris%';

UNATTRIBUTED DELETE

Log-to-Artifact Matcher

- Evaluate the integrity of carved data and metadata using log entries.

Data modifications (disk images):

1. DELETE
2. **INSERT**
3. UPDATE

★ **D**ata **D**efinition **L**anguage (e.g., CREATE, ALTER, DROP)

Read-only queries (RAM snapshots):

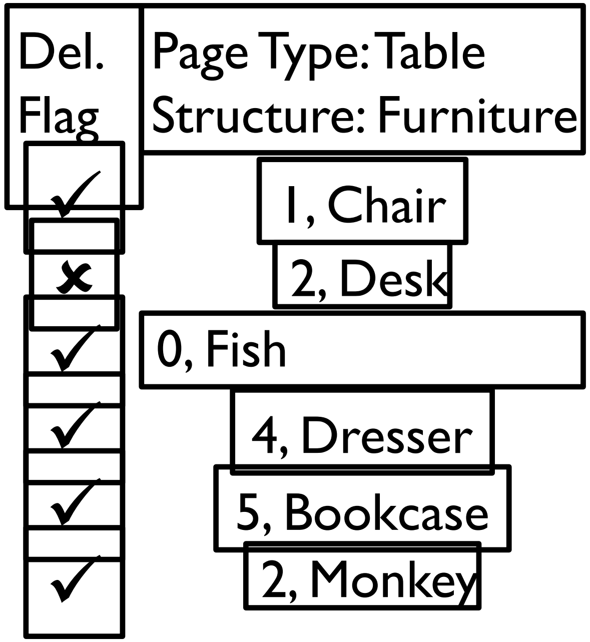
4. SELECT

★ Read only queries do not leave evidence on disk

Active Record-to-Log Matching

- Similar process as deleted record-to-log matching.

DICE Output



Del. Flag	Page Type: Table Structure: Furniture
✓	1, Chair
✗	2, Desk
✓	0, Fish
✓	4, Dresser
✓	5, Bookcase
✓	2, Monkey

Log File

T1, INSERT INTO Furniture
VALUES (1, ‘Chair’);

T2, INSERT INTO Furniture
VALUES (2, ‘Desk’);

T3, INSERT INTO Furniture
VALUES (3, ‘Lamp’);

T4, INSERT INTO Furniture
VALUES (4, ‘Dresser’);

T5, DELETE FROM Furniture
WHERE Name LIKE ‘Lamp’;

T6, INSERT INTO Furniture
VALUES (5,‘Bookcase’);

T7, UPDATE Furniture
SET Item = ‘Monkey’
WHERE ID = 2;

Active Record-to-Log Matching

- Only active records and INSERT commands are considered.

DICE Output

Del. Flag	Page Type: Table Structure: Furniture
✓	1, Chair
[redacted]	[redacted]
✓	0, Fish
✓	4, Dresser
✓	5, Bookcase
✓	2, Monkey

Log File

T1, INSERT INTO Furniture
VALUES (1, 'Chair');

T2, INSERT INTO Furniture
VALUES (2, 'Desk');

T3, INSERT INTO Furniture
VALUES (3, 'Lamp');

T4, INSERT INTO Furniture
VALUES (4, 'Dresser');

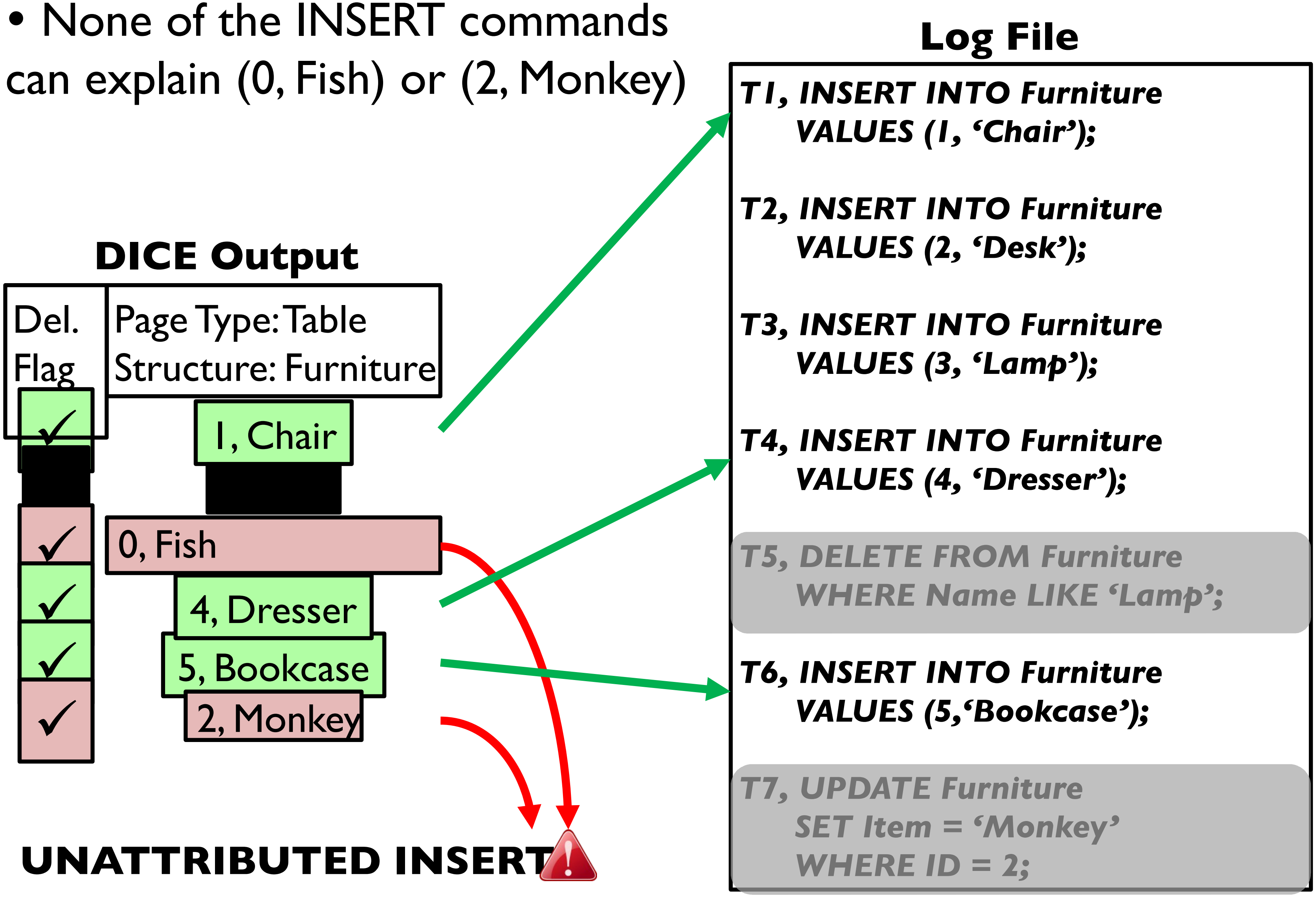
T5, DELETE FROM Furniture
WHERE Name LIKE 'Lamp';

T6, INSERT INTO Furniture
VALUES (5,'Bookcase');

T7, UPDATE Furniture
SET Item = 'Monkey'
WHERE ID = 2;

Active Record-to-Log Matching

- None of the INSERT commands can explain (0, Fish) or (2, Monkey)



DICE Output

Del. Flag	Page Type: Table Structure: Furniture
✓	1, Chair
✓	0, Fish
✓	4, Dresser
✓	5, Bookcase
✓	2, Monkey

Log File

```
T1, INSERT INTO Furniture
    VALUES (1, ‘Chair’);

T2, INSERT INTO Furniture
    VALUES (2, ‘Desk’);

T3, INSERT INTO Furniture
    VALUES (3, ‘Lamp’);

T4, INSERT INTO Furniture
    VALUES (4, ‘Dresser’);

T5, DELETE FROM Furniture
    WHERE Name LIKE ‘Lamp’;

T6, INSERT INTO Furniture
    VALUES (5,‘Bookcase’);

T7, UPDATE Furniture
    SET Item = ‘Monkey’
    WHERE ID = 2;
```

UNATTRIBUTED INSERT

Log-to-Artifact Matcher

- Evaluate the integrity of carved data and metadata using log entries.

Data modifications (disk images):

1. DELETE
2. INSERT
3. **UPDATE**

★ **D**ata **D**efinition **L**anguage (e.g., CREATE, ALTER, DROP)

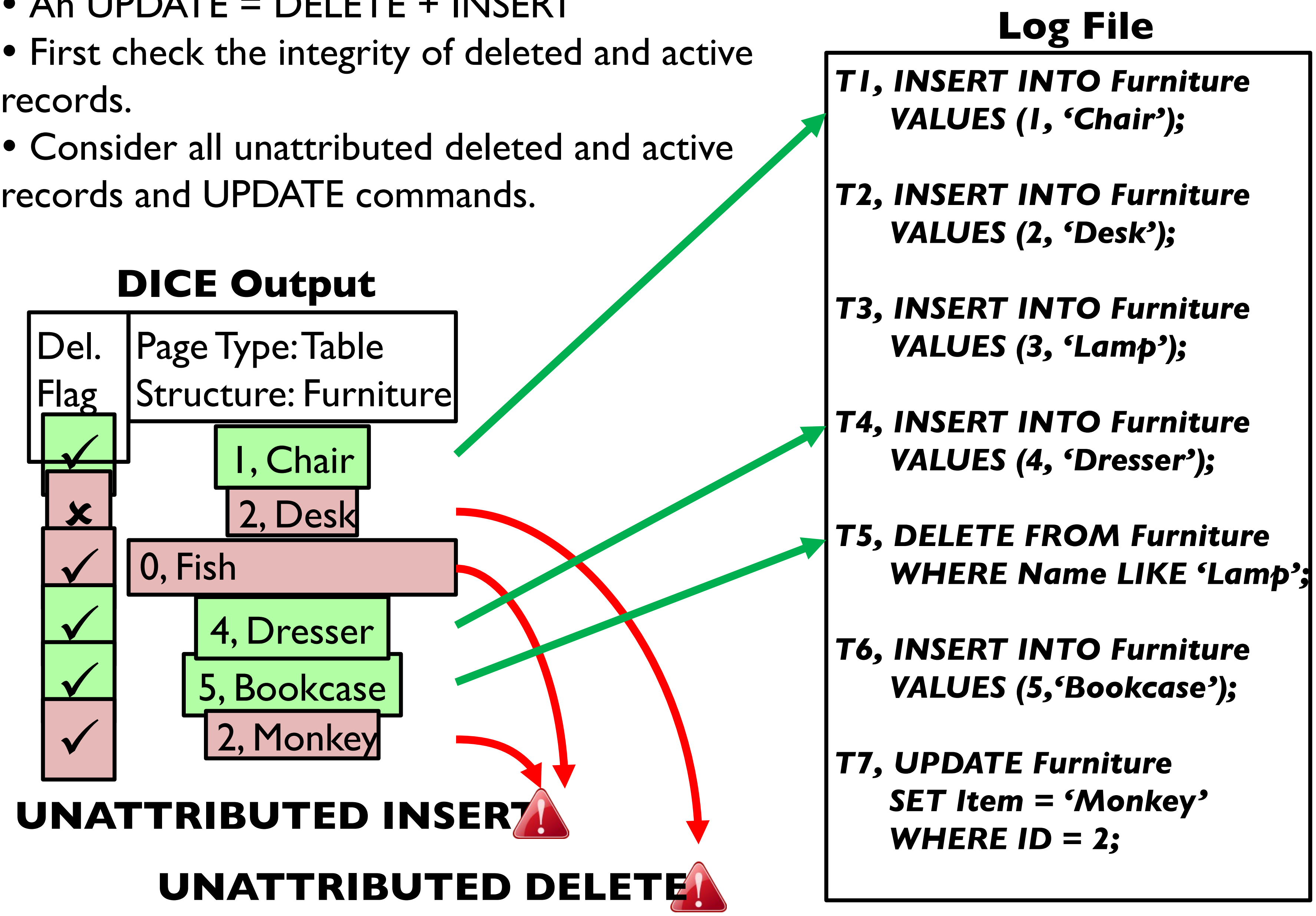
Read-only queries (RAM snapshots):

4. SELECT

★ Read only queries do not leave evidence on disk

Updated Record-to-Log Matching

- An UPDATE = DELETE + INSERT
- First check the integrity of deleted and active records.
- Consider all unattributed deleted and active records and UPDATE commands.



DICE Output

Del. Flag	Page Type: Table Structure: Furniture
✓	1, Chair
✗	2, Desk
✓	0, Fish
✓	4, Dresser
✓	5, Bookcase
✓	2, Monkey

UNATTRIBUTED INSERT

UNATTRIBUTED DELETE

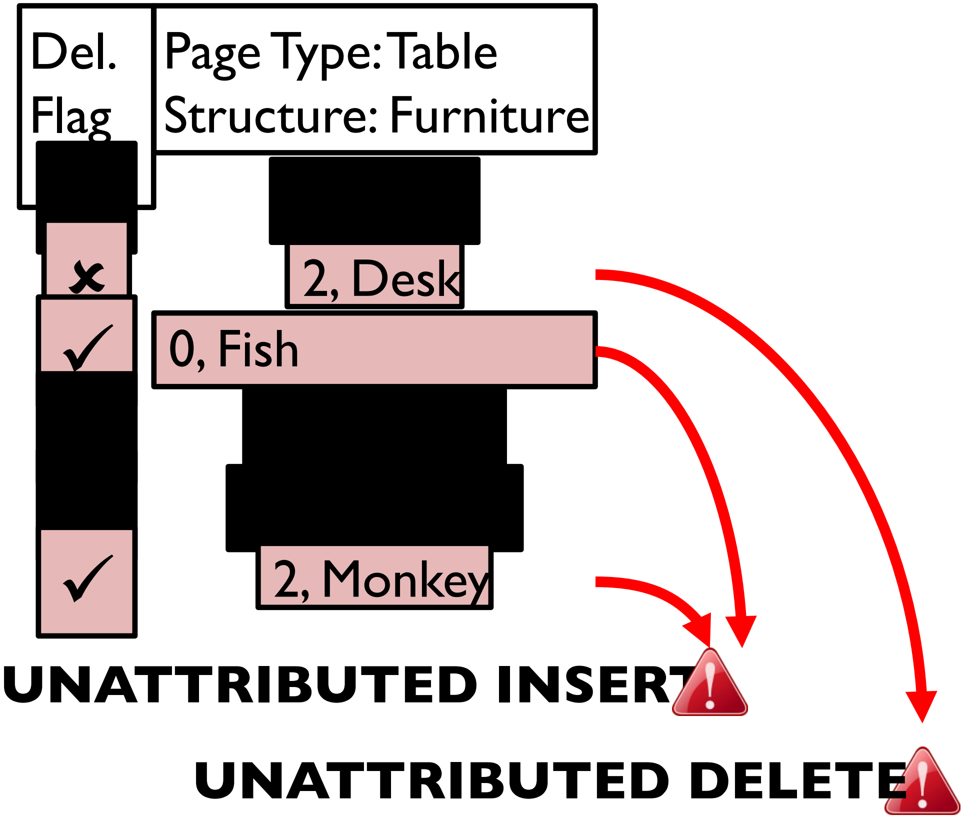
Log File

```
T1, INSERT INTO Furniture
    VALUES (1, 'Chair');

T2, INSERT INTO Furniture
    VALUES (2, 'Desk');

T3, INSERT INTO Furniture
    VALUES (3, 'Lamp');

T4, INSERT INTO Furniture
    VALUES (4, 'Dresser');

T5, DELETE FROM Furniture
    WHERE Name LIKE 'Lamp';

T6, INSERT INTO Furniture
    VALUES (5,'Bookcase');

T7, UPDATE Furniture
    SET Item = 'Monkey'
    WHERE ID = 2;
```

Updated Record-to-Log Matching

- Consider all unattributed deleted and active records and UPDATE commands.
  - deleted record ➡ WHERE clause
  - active records ➡ SET clause

DICE Output



Del. Flag	Page Type: Table Structure: Furniture
[redacted]	[redacted]
✗	2, Desk
✓	0, Fish
[redacted]	[redacted]
[redacted]	[redacted]
✓	2, Monkey

UNATTRIBUTED INSERT

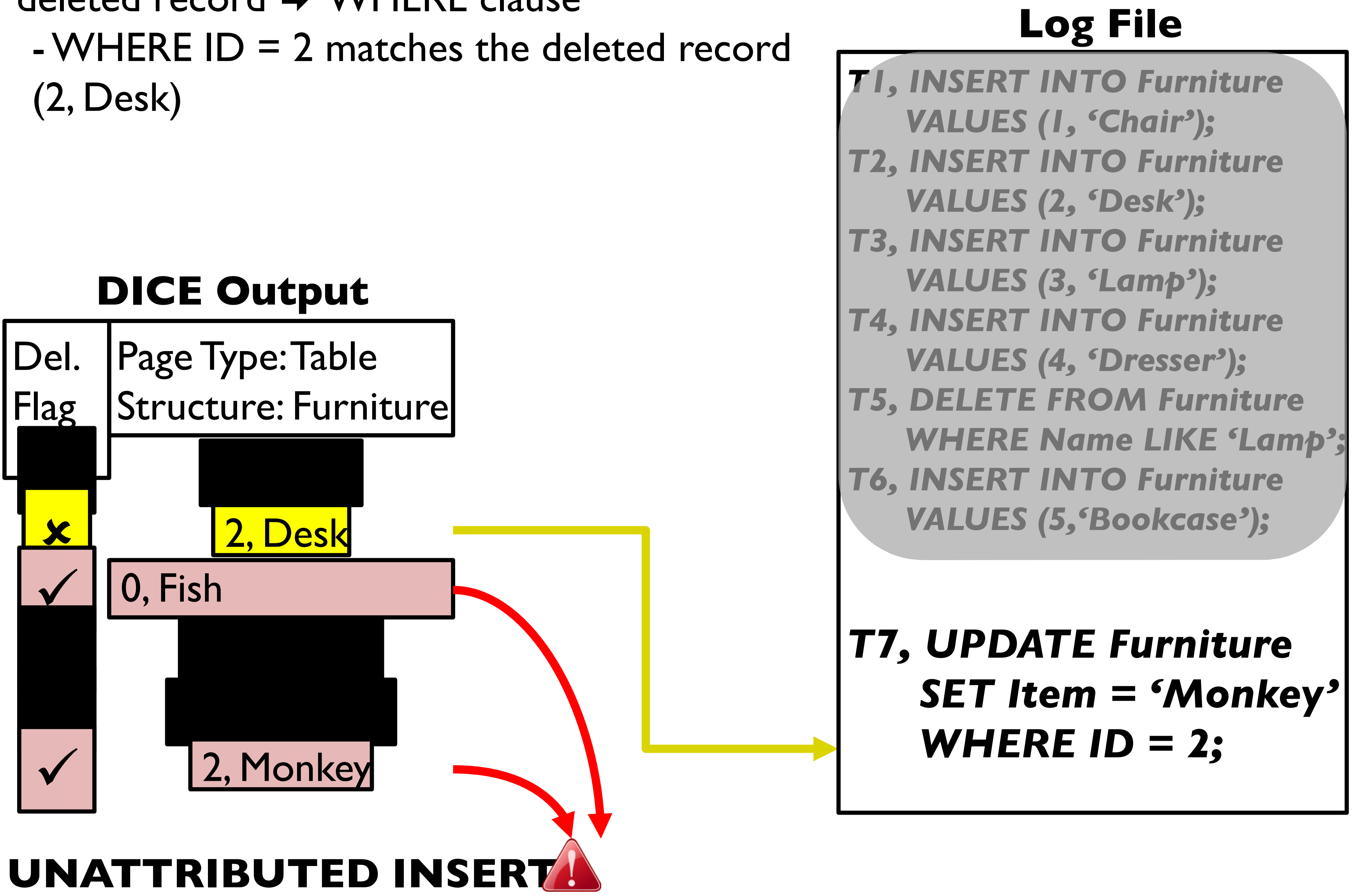
UNATTRIBUTED DELETE

Log File

```
T1, INSERT INTO Furniture
    VALUES (1, 'Chair');
T2, INSERT INTO Furniture
    VALUES (2, 'Desk');
T3, INSERT INTO Furniture
    VALUES (3, 'Lamp');
T4, INSERT INTO Furniture
    VALUES (4, 'Dresser');
T5, DELETE FROM Furniture
    WHERE Name LIKE 'Lamp';
T6, INSERT INTO Furniture
    VALUES (5,'Bookcase');

T7, UPDATE Furniture
     SET Item = 'Monkey'
     WHERE ID = 2;
```

Updated Record-to-Log Matching

- deleted record ➡ WHERE clause
  - WHERE ID = 2 matches the deleted record (2, Desk)



DICE Output

Del. Flag	Page Type: Table Structure: Furniture
✗	2, Desk
✓	0, Fish
✓	2, Monkey

UNATTRIBUTED INSERT

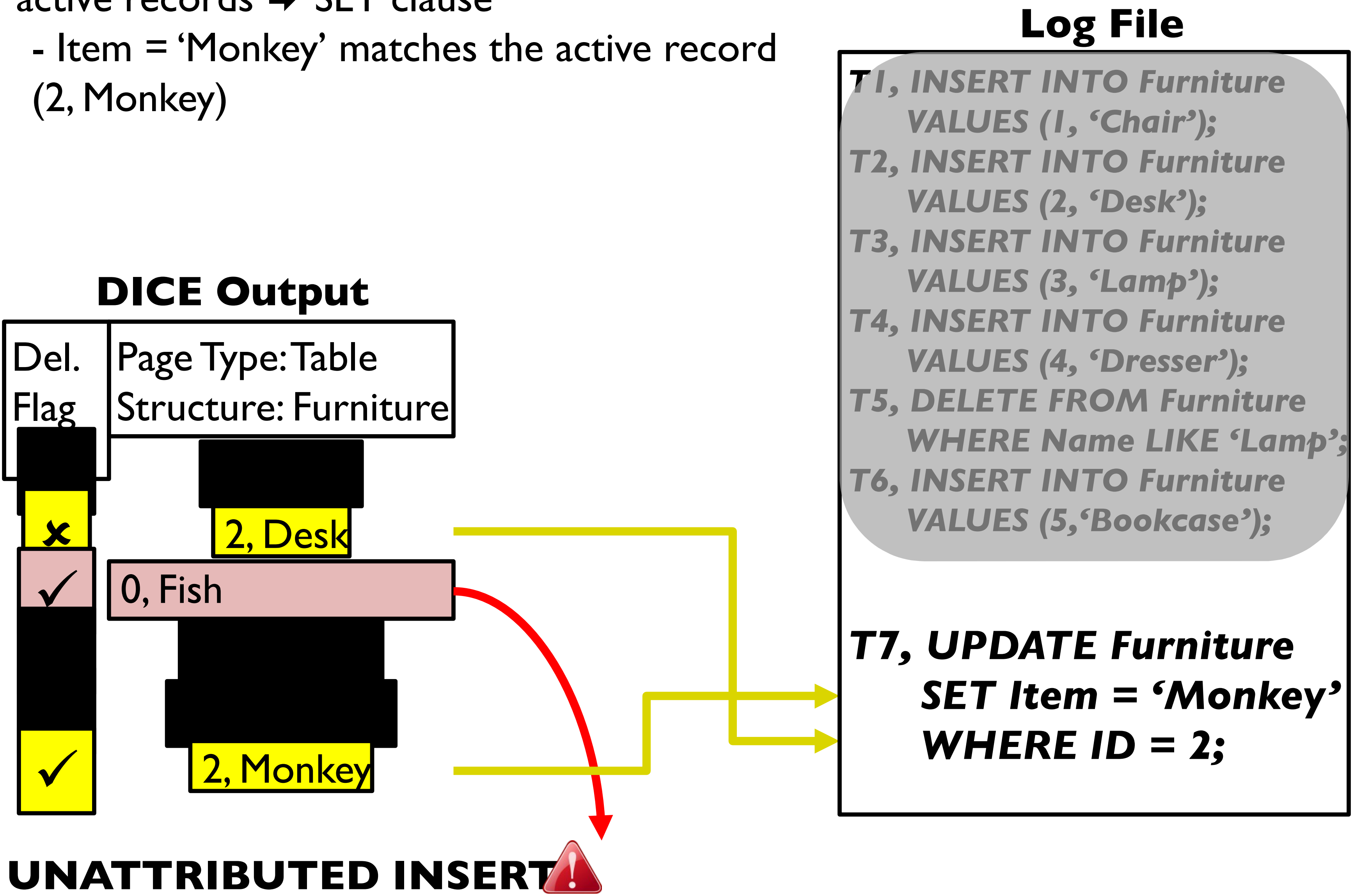
Log File

T1, INSERT INTO Furniture
VALUES (1, 'Chair');
T2, INSERT INTO Furniture
VALUES (2, 'Desk');
T3, INSERT INTO Furniture
VALUES (3, 'Lamp');
T4, INSERT INTO Furniture
VALUES (4, 'Dresser');
T5, DELETE FROM Furniture
WHERE Name LIKE 'Lamp';
T6, INSERT INTO Furniture
VALUES (5,'Bookcase');

T7, UPDATE Furniture
SET Item = 'Monkey'
WHERE ID = 2;

Updated Record-to-Log Matching

- active records ➡ SET clause
  - Item = ‘Monkey’ matches the active record (2, Monkey)



DICE Output

Del. Flag	Page Type: Table Structure: Furniture
✗	2, Desk
✓	0, Fish
✓	2, Monkey

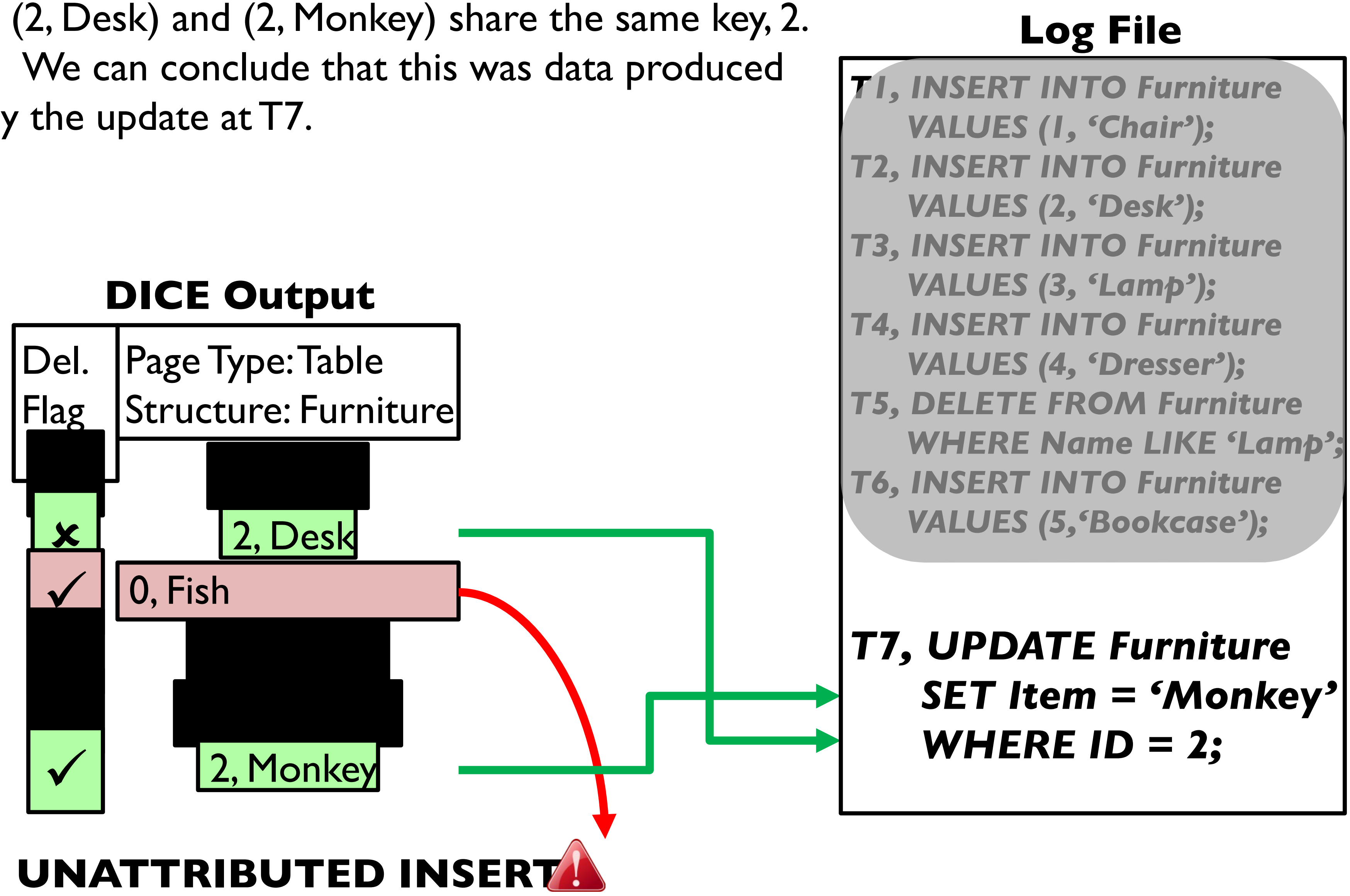
UNATTRIBUTED INSERT

Log File

```
T1, INSERT INTO Furniture
    VALUES (1, ‘Chair’);
T2, INSERT INTO Furniture
    VALUES (2, ‘Desk’);
T3, INSERT INTO Furniture
    VALUES (3, ‘Lamp’);
T4, INSERT INTO Furniture
    VALUES (4, ‘Dresser’);
T5, DELETE FROM Furniture
    WHERE Name LIKE ‘Lamp’;
T6, INSERT INTO Furniture
    VALUES (5,‘Bookcase’);

T7, UPDATE Furniture
    SET Item = ‘Monkey’
    WHERE ID = 2;
```

Updated Record-to-Log Matching

- (2, Desk) and (2, Monkey) share the same key, 2.
- We can conclude that this was data produced by the update at T7.



Log File

T1, INSERT INTO Furniture
VALUES (1, 'Chair');
T2, INSERT INTO Furniture
VALUES (2, 'Desk');
T3, INSERT INTO Furniture
VALUES (3, 'Lamp');
T4, INSERT INTO Furniture
VALUES (4, 'Dresser');
T5, DELETE FROM Furniture
WHERE Name LIKE 'Lamp';
T6, INSERT INTO Furniture
VALUES (5,'Bookcase');

T7, UPDATE Furniture
SET Item = 'Monkey'
WHERE ID = 2;

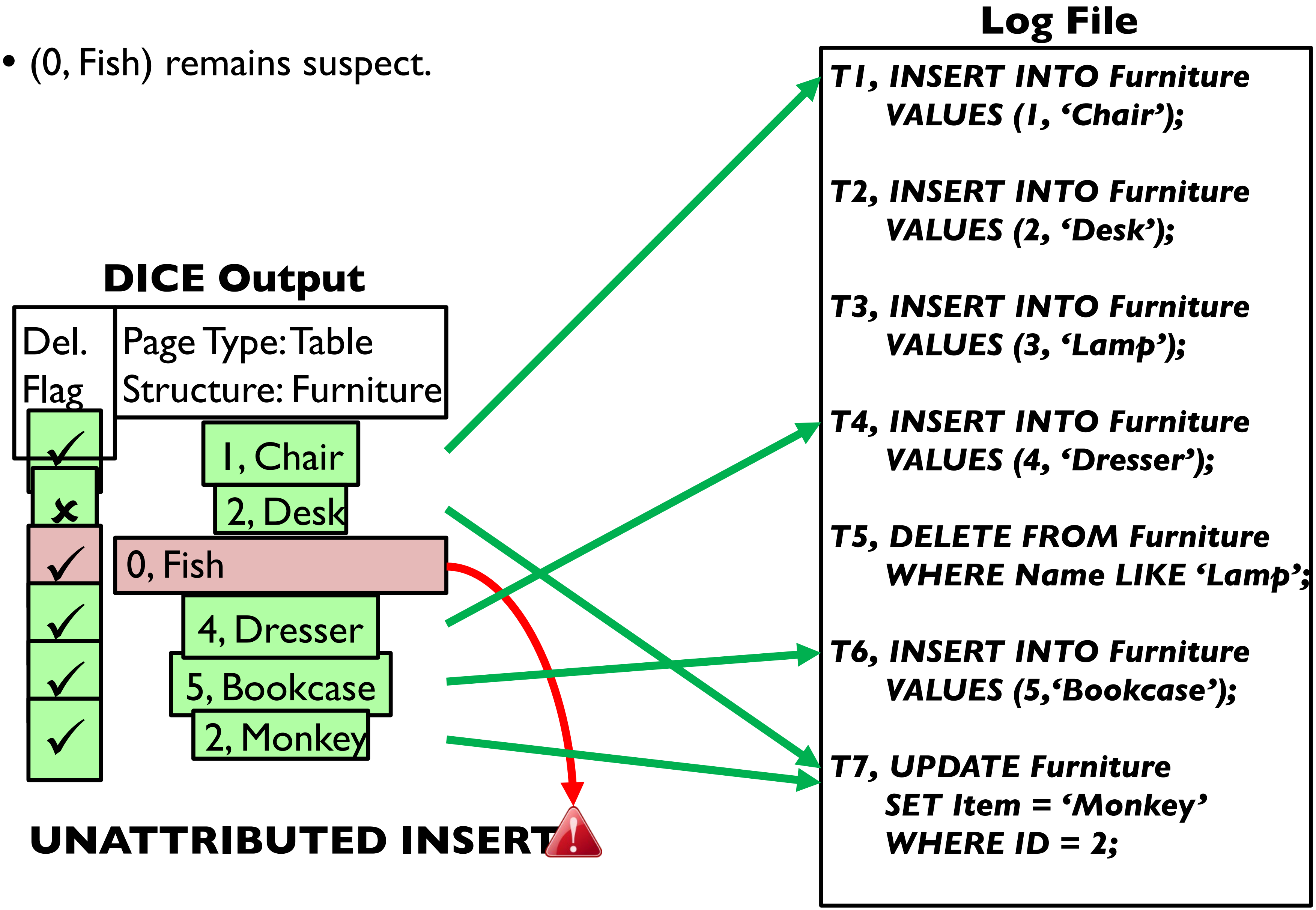
DICE Output

Del. Flag	Page Type: Table Structure: Furniture
✗	2, Desk
✓	0, Fish
✓	2, Monkey

UNATTRIBUTED INSERT

Updated Record-to-Log Matching

- (0, Fish) remains suspect.



DICE Output

Del. Flag	Page Type: Table Structure: Furniture
✓	1, Chair
✗	2, Desk
✓	0, Fish
✓	4, Dresser
✓	5, Bookcase
✓	2, Monkey

UNATTRIBUTED INSERT

Log File

```
T1, INSERT INTO Furniture
    VALUES (1, ‘Chair’);

T2, INSERT INTO Furniture
    VALUES (2, ‘Desk’);

T3, INSERT INTO Furniture
    VALUES (3, ‘Lamp’);

T4, INSERT INTO Furniture
    VALUES (4, ‘Dresser’);

T5, DELETE FROM Furniture
    WHERE Name LIKE ‘Lamp’;

T6, INSERT INTO Furniture
    VALUES (5,‘Bookcase’);

T7, UPDATE Furniture
    SET Item = ‘Monkey’
    WHERE ID = 2;
```

Log-to-Artifact Matcher

- Evaluate the integrity of carved data and metadata using log entries.

Data modifications (disk images):

1. DELETE
2. INSERT
3. UPDATE

★ **D**ata **D**efinition **L**anguage (e.g., CREATE, ALTER, DROP)

Read-only queries (RAM snapshots):

4. SELECT

★ Read only queries do not leave evidence on disk

Select Query-to-Log Matching

- All SELECT queries use either a:
  1. Full table scan (FTS)
  2. Index access.

  ★ Views ultimately access tables and materialized views behave

  similar to tables.

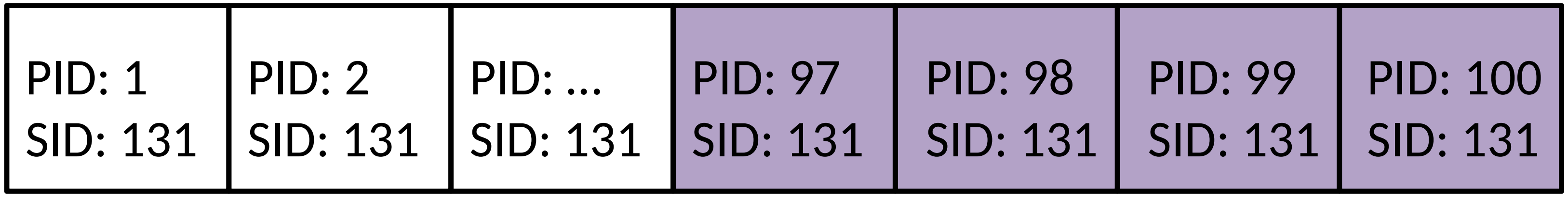
FTS

- the entire table is scanned
- the DBMS allocates a limited amount of memory
- the end result of a FTS is the last N pages of the table from the database file (i.e., a repeating pattern)

Full Table Scan: Example

- Table Employee has 100 pages.
- The DBMS allocates 4 pages to a FTS.
- A FTS of Employee leaves pages 97, 98, 99, and 100 in RAM.

Table Employee on Disk



PID: 1 SID: 131

PID: 2 SID: 131

PID: … SID: 131

PID: 97 SID: 131

PID: 98 SID: 131

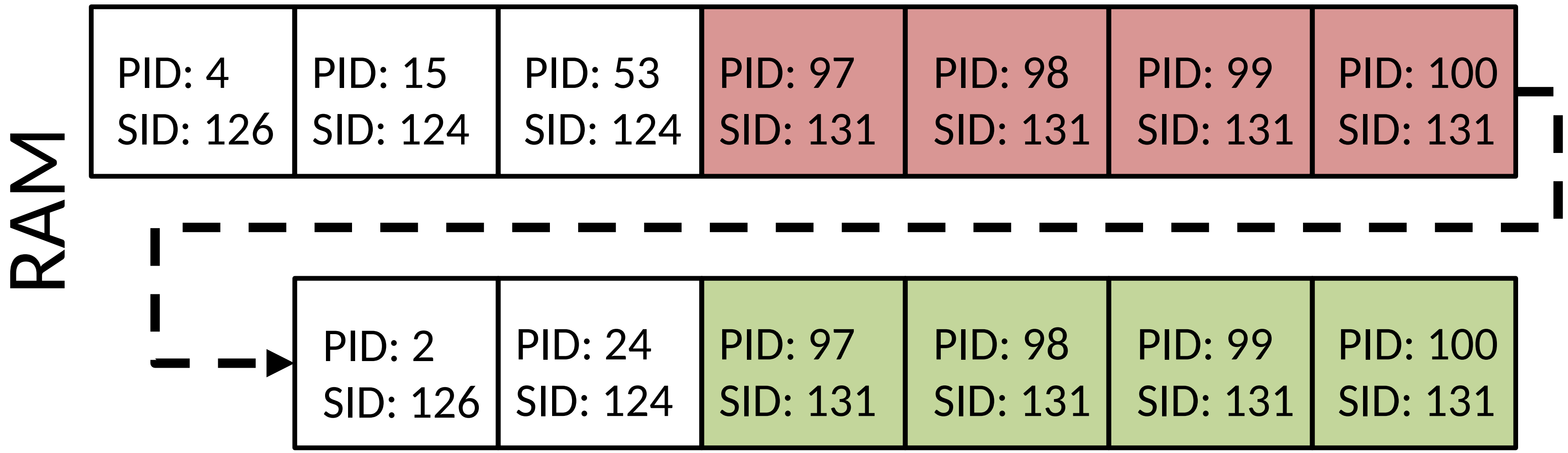
PID: 99 SID: 131

PID: 100 SID: 131

▨ Full table scan result in buffer cache

PID: Page ID
SID: Structure ID

Full Table Scans



RAM

PID: 4 SID: 126 | PID: 15 SID: 124 | PID: 53 SID: 124 | PID: 97 SID: 131 | PID: 98 SID: 131 | PID: 99 SID: 131 | PID: 100 SID: 131

PID: 2 SID: 126 | PID: 24 SID: 124 | PID: 97 SID: 131 | PID: 98 SID: 131 | PID: 99 SID: 131 | PID: 100 SID: 131

Audit Log

```
T1, SELECT C_Name
FROM Customer
WHERE C_City = ‘Jackson’;

T2, SELECT E_Name, E_Salary
FROM Employee;

T3, SELECT *
FROM Customer
WHERE C_City = ‘Dallas’;

T4, SELECT *
FROM Employee
WHERE E_Name LIKE ‘%ne’;
```

Select Query-to-Log Matching

- All SELECT queries use either a:
  1. Full table scan (FTS)
  2. Index access

  ★ Views ultimately access tables and materialized views behave similar to tables.

Index Access

- an index stores value-pointer pairs to access specific table pages
- both the index pages and table pages are cached
- table pages may contain data unrelated to query
- index pages contain ordered values giving us a range of possible filters

Index Access: Example 1

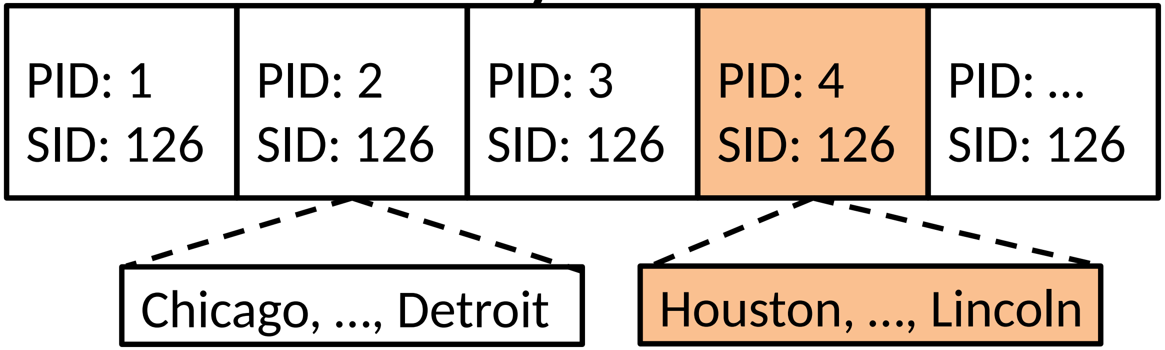
Query I

SELECT C_Name
FROM Customer
WHERE C_City = ‘Jackson’;

- Query I filters on City = ‘Jackson’
- Page ID = 4 gets read into RAM
- The relevant table pages are read into RAM

PID: Page ID
SID: Structure ID

Index Customer City on Disk



Index Access: Example 2

Query 2

SELECT *

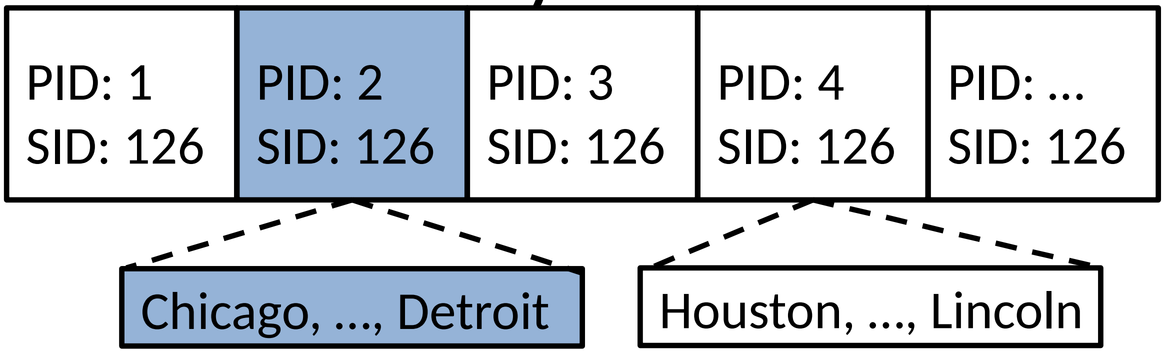
FROM Customer

WHERE C_City = ‘Dallas’;

- Query filters on City = ‘Dallas’
- Page ID = 2 gets read into RAM
- The relevant table pages are read into RAM

PID: Page ID
SID: Structure ID

Index Customer City on Disk



PID: 1
SID: 126

PID: 2
SID: 126

PID: 3
SID: 126

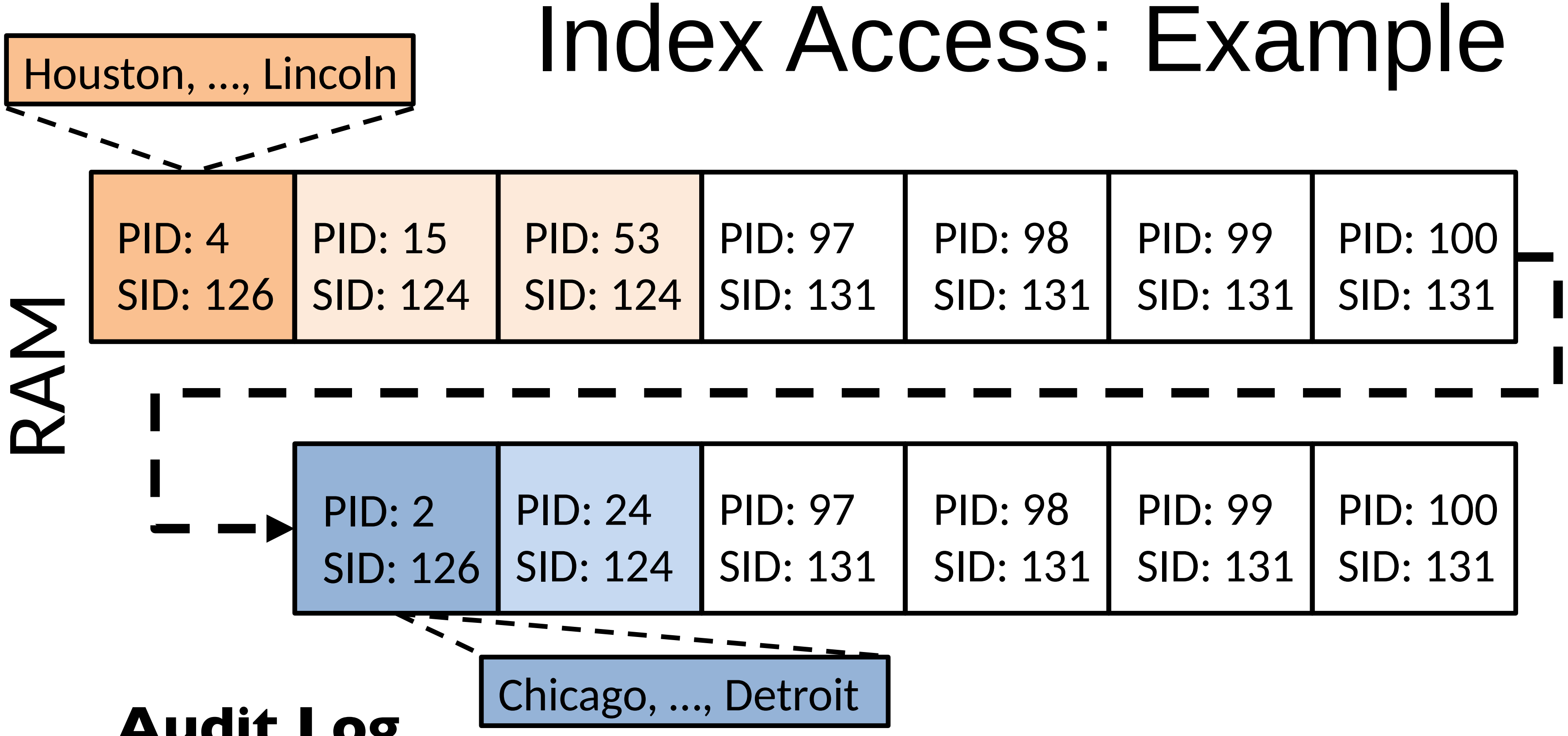
PID: 4
SID: 126

PID: …
SID: 126

Chicago, …, Detroit

Houston, …, Lincoln

Index Access: Example



PID: 4 SID: 126	PID: 15 SID: 124	PID: 53 SID: 124	PID: 97 SID: 131	PID: 98 SID: 131	PID: 99 SID: 131	PID: 100 SID: 131

PID: 2 SID: 126	PID: 24 SID: 124	PID: 97 SID: 131	PID: 98 SID: 131	PID: 99 SID: 131	PID: 100 SID: 131

Audit Log

T1, SELECT C_Name
FROM Customer
WHERE C_City = ‘Jackson’;

T2, SELECT E_Name, E_Salary
FROM Employee;

T3, SELECT *
FROM Customer
WHERE C_City = ‘Dallas’;

T4, SELECT *
FROM Employee
WHERE E_Name LIKE ‘%ne’;

Performance

DICE Processing

- DICE carves ~ 1.1 MB/s.
  - Several files from 1MB to 3 GB were tested.
- Carving is dependent on the # of pages, not file size:
  - 2.5 GB process snapshot with 600 MB of pages ➡ 4.2 MB/s
  - 8 GB RAM snapshot with 600 MB of pages ➡ 13.2 MB/s

Log-to-Artifact Matching

- Assuming the log can fit into memory, the cost is linear.
- DBDetective operates independent of the DBMS
- Carving and log-to-artifact matching can occur offsite

*checksum evaluation limits page parsing and artifact matching

Future Work

Timeline of Events

- Match data and metadata to specific commands in the log.
- Ex. Name LIKE ‘Chris%’ vs City = ‘Chicago’ ➡ (1, Christine, Chicago)

Detect Direct File Modifications

- Ex. ‘Karen’ was changed to ‘Boris’ in the database file using Python rather than SQL.
- Protect against other privileged users (system admin)
- All DBMS security (defense and detection) are bypassed

Questions?

Accuracy: False-Negatives

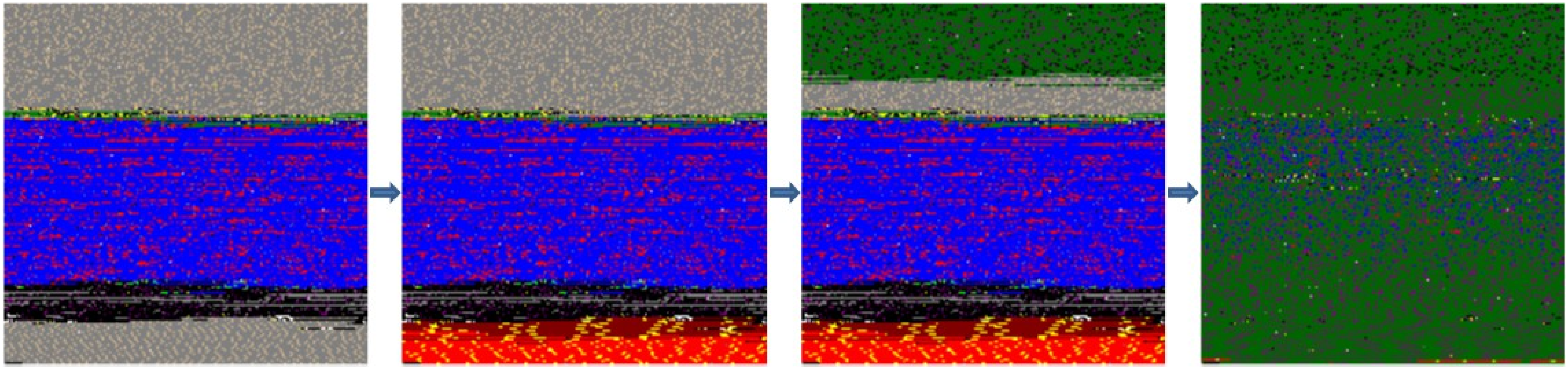
- The conditions for operations may overlap, creating false-negatives. Ex. Name LIKE ‘Chris%’ vs City = ‘Chicago’ ➡ (1, Christine, Chicago)

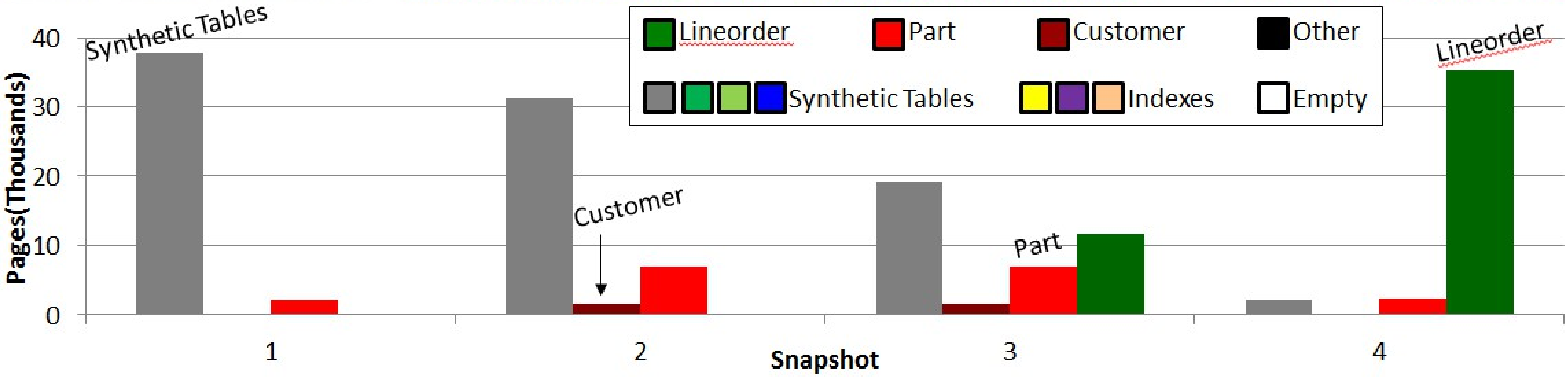
- We are interested in identifying data that does not match *any* log operation.

- False-negative present a problem if it the tampered data matches a pre-existing log operation. Ex.

  T1. DELETE FROM Customer WHERE Name LIKE ‘Chris%’;
  T2. DELETE FROM Customer WHERE City = ‘Chicago’;
  T3. INSERT INTO Customer VALUES (1, ‘Christine’, ‘Chicago’);
  T4. Logging is disabled and (1, Christine, Chicago) is deleted

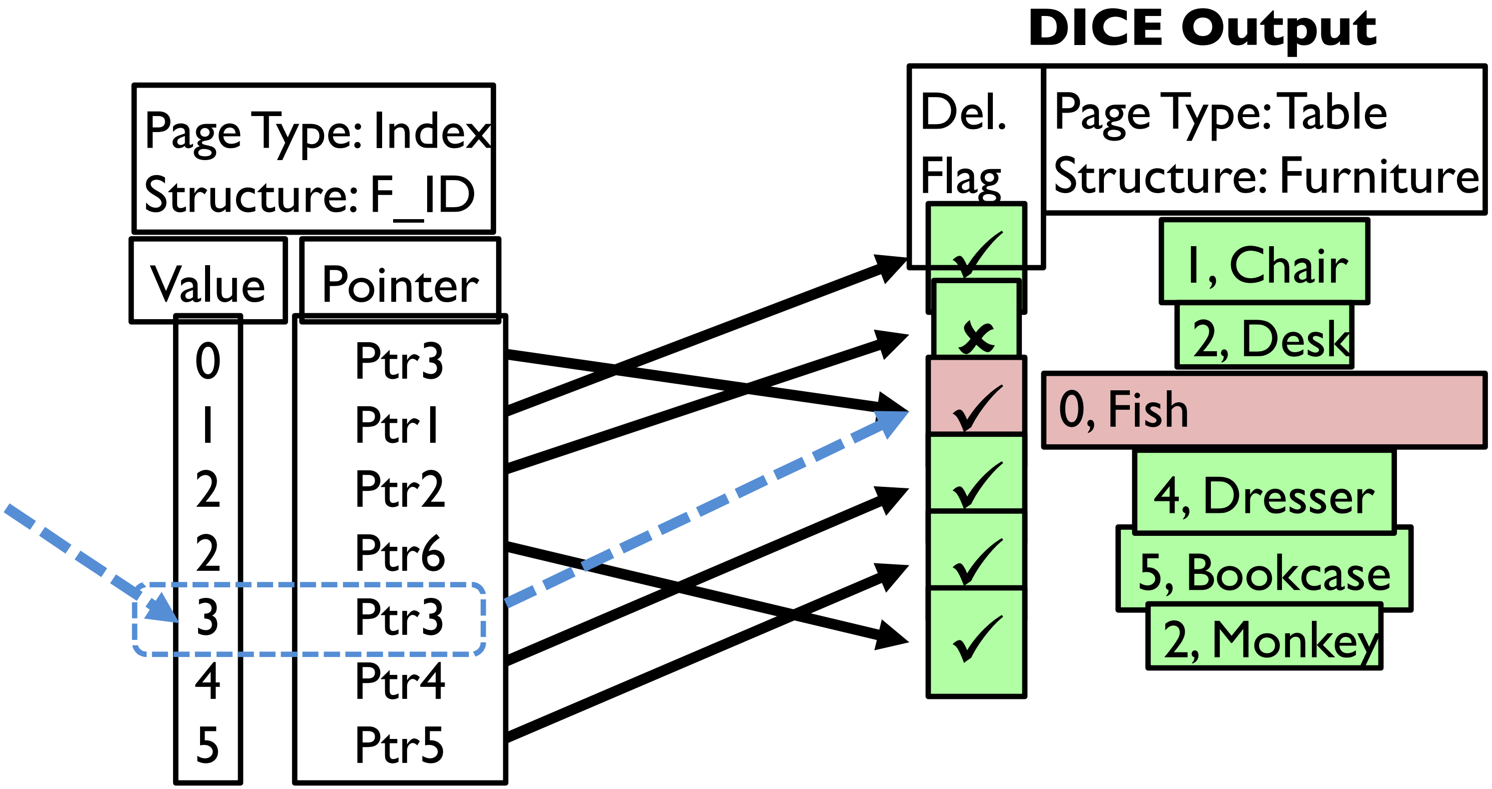
DICE Example: RAM Monitoring





Indexes

Value 3 is gone from *Furniture* table but remains in the index



DICE Output

Page Type: Index
Structure: F_ID

Value	Pointer
0	Ptr3
1	Ptr1
2	Ptr2
2	Ptr6
3	Ptr3
4	Ptr4
5	Ptr5

Del. Flag	Page Type: Table Structure: Furniture
✓	1, Chair
✗	2, Desk
✓	0, Fish
✓	4, Dresser
✓	5, Bookcase
✓	2, Monkey