# CS3223 Lecture 10
# Crash Recovery

# Recovery

- **Recovery manager** guarantees atomicity and durability properties of Xacts
  - Undo: remove effects of aborted Xact to preserve atomicity
  - Redo: re-installing effects of committed Xact for durability
- Types of failure
  - Transaction failures: transaction aborts
    - Application rollbacks transaction
    - System rollbacks transaction (e.g. deadlock, violation of integrity constraint)
  - System crash: loss of volatile memory contents
    - Power failure
    - Bug in DBMS/OS
    - Hardware malfunction
  - Media failures: data is lost/corrupted on non-volatile storage
    - Disk head crash / Failure during data transfer

# Recovery Manager

Processes three operations:

- Commit(T) - install T's updated pages into database
- Abort(T) - restore all data that T updated to their prior values
- Restart - recover database to a consistent state from system failure
  - abort all active Xacts at the time of system failure
  - installs updates of all committed Xacts that were not installed in the database before the failure

Desirable properties:

- Add little overhead to the normal processing of Xacts
- Recover quickly from a failure

# Interaction of Recovery & Buffer Managers

Two issues with dirty pages in buffer pool:

1. **Can a dirty page updated by Xact T be written to disk before T commits?**
   - Yes $\Longrightarrow$ steal policy
   - No $\Longrightarrow$ no-steal policy
2. **Must all dirty pages that are updated by Xact T be written to disk when T commits?**
   - Yes $\Longrightarrow$ force policy
   - No $\Longrightarrow$ no-force policy

# Interaction of Recovery & Buffer Managers (cont.)

- Steal policy: allows dirty pages updated by Xact T to be replaced from buffer pool before T commits
- Force policy: requires all dirty pages updated by Xact T to be written to disk when T commits
- Four possible design options:

| | Force | No-force |
|---|---|---|
| **Steal** | undo & no redo | undo & redo |
| **No-steal** | no undo & no redo | no undo & redo |

  - No-steal policy $\Longrightarrow$ no undo
  - Force policy $\Longrightarrow$ no redo

# Example: Crash Recovery

Accounts

| name | balance |
|---|---|
| Alice | 200 |
| Bob | 800 |
| Carol | 300 |
| Dave | 500 |
| Eve | 600 |
| Fred | 200 |

Transactions:

$T_1$: Transfer \$100 from Alice to Carol
$T_2$: Deposit \$200 to Bob
$T_3$: Withdraw \$500 from Eve

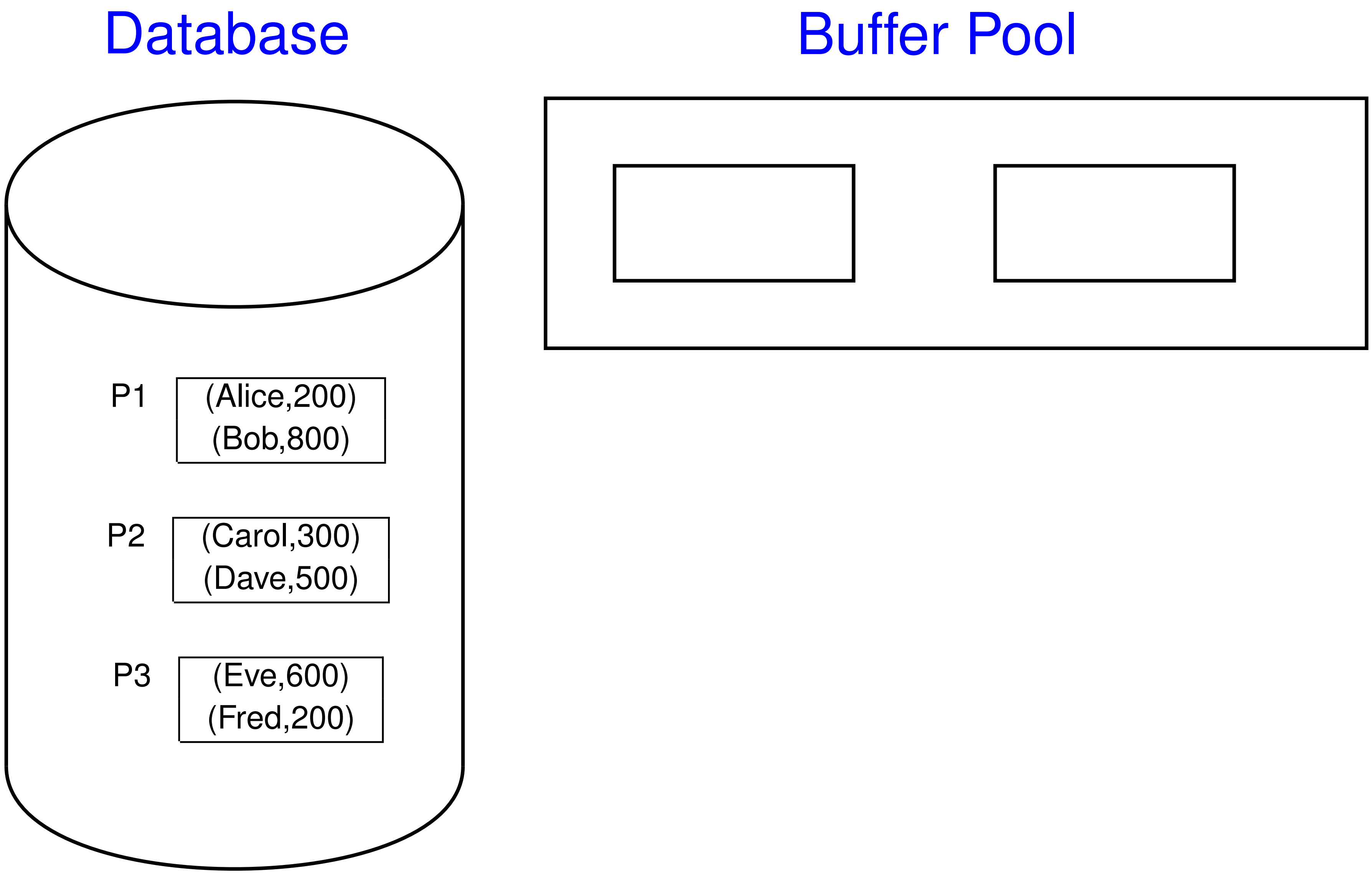
$T_1$: $R_1$(Alice, 200), $R_1$(Carol, 300), $W_1$(Alice, 100), $W_1$(Carol, 400)
$T_2$: $R_2$(Bob, 800), $W_2$(Bob, 1000)
$T_3$: $R_3$(Eve, 600), $W_3$(Eve, 100)

# Example: Crash Recovery (cont.)

Schedule

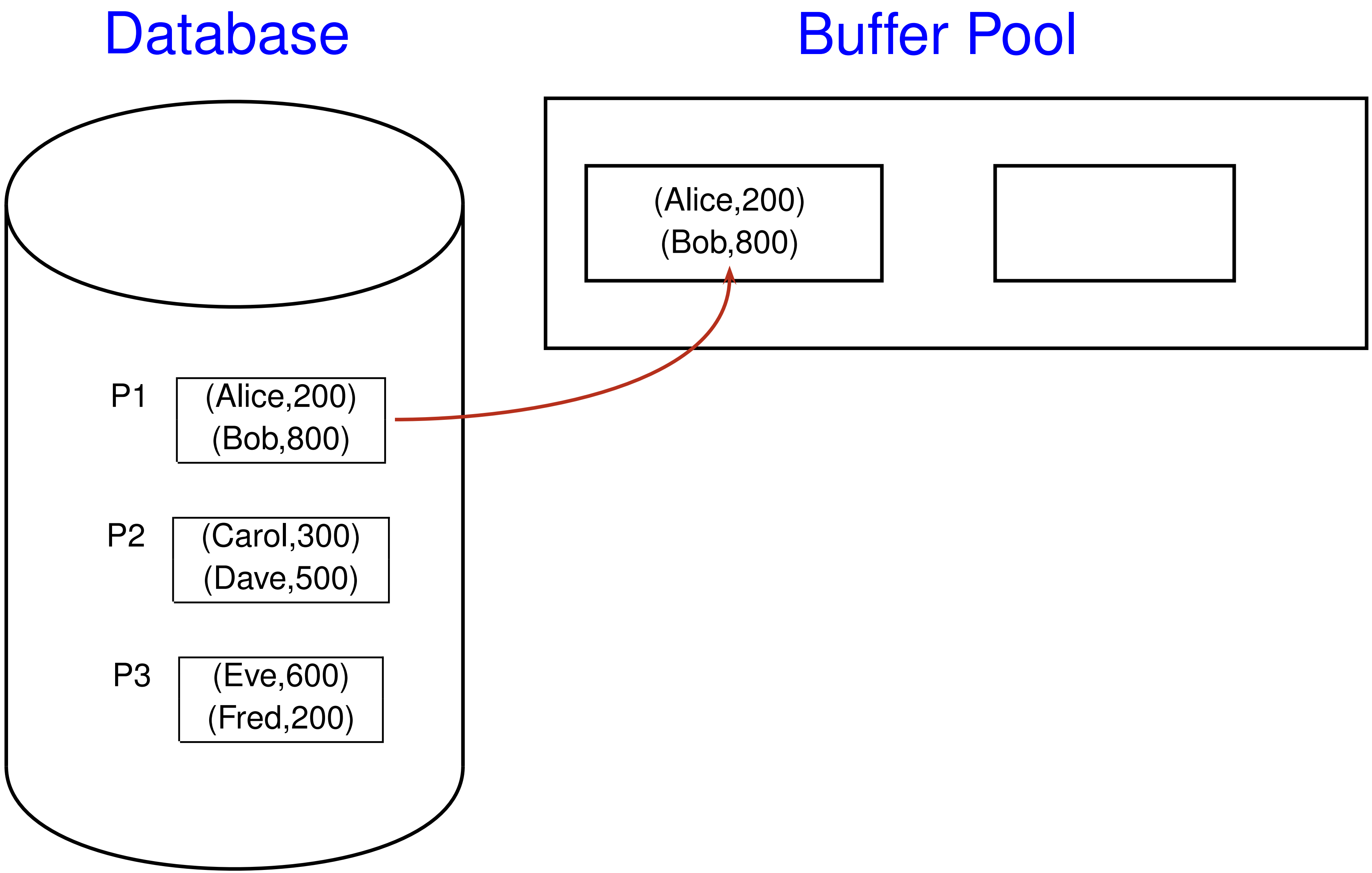
| Step | $T_1$ | $T_2$ | $T_3$ |
|---|---|---|---|
| 1 | $R_1$(Alice, 200) | | |
| 2 | $R_1$(Carol, 300) | | |
| 3 | $W_1$(Alice, 100) | | |
| 4 | | $R_2$(Bob, 800) | |
| 5 | | $W_2$(Bob, 1000) | |
| 6 | | $Commit_2$ | |
| 7 | $W_1$(Carol, 400) | | |
| 8 | | | $R_3$(Eve, 600) |
| 9 | | | $W_3$(Eve, 100) |
| 10 | | | $Commit_3$ |
| 11 | | SYSTEM CRASH! | |

# Example: Crash Recovery (cont.)



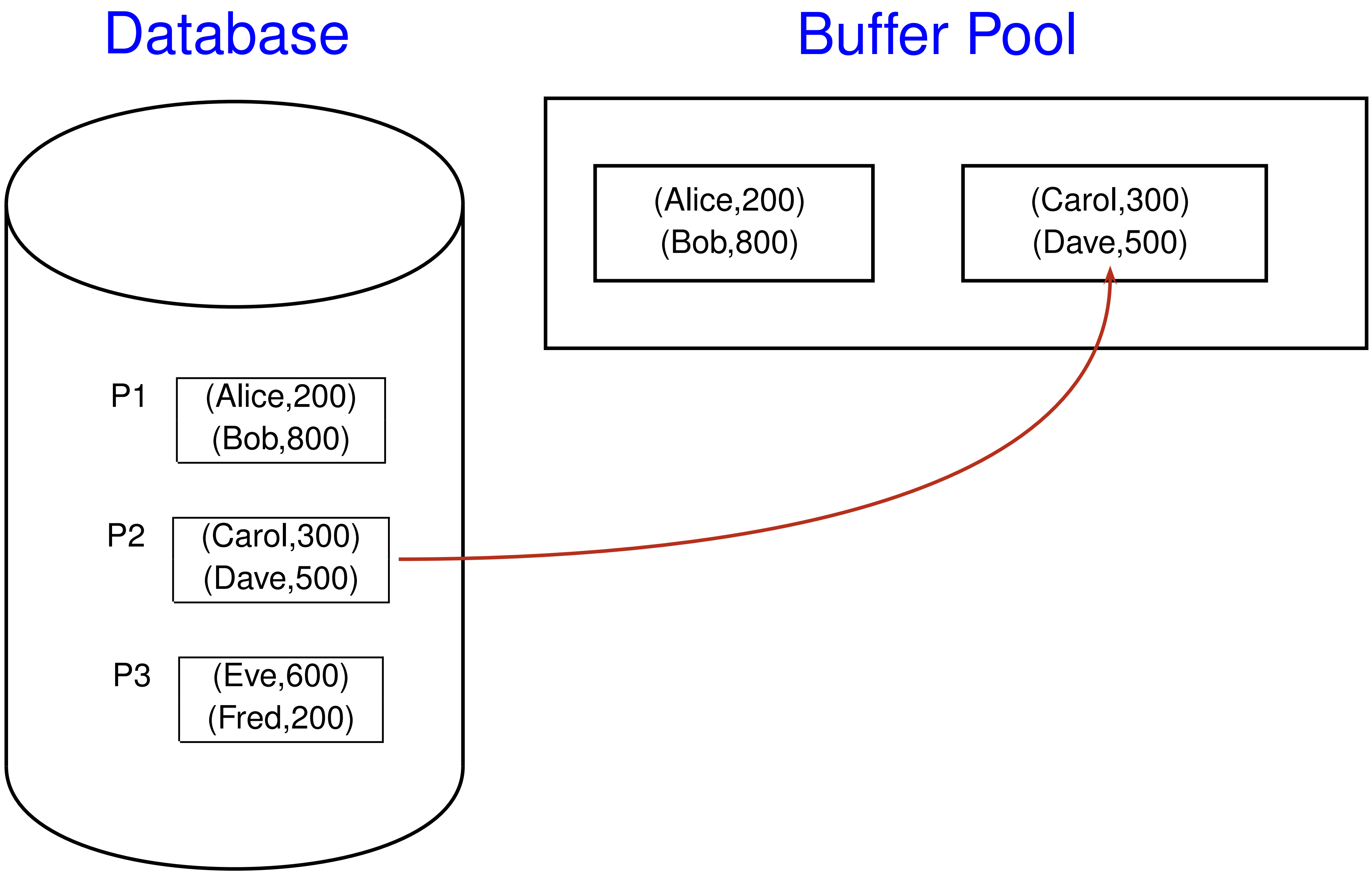

Initial State

# Example: Crash Recovery (cont.)



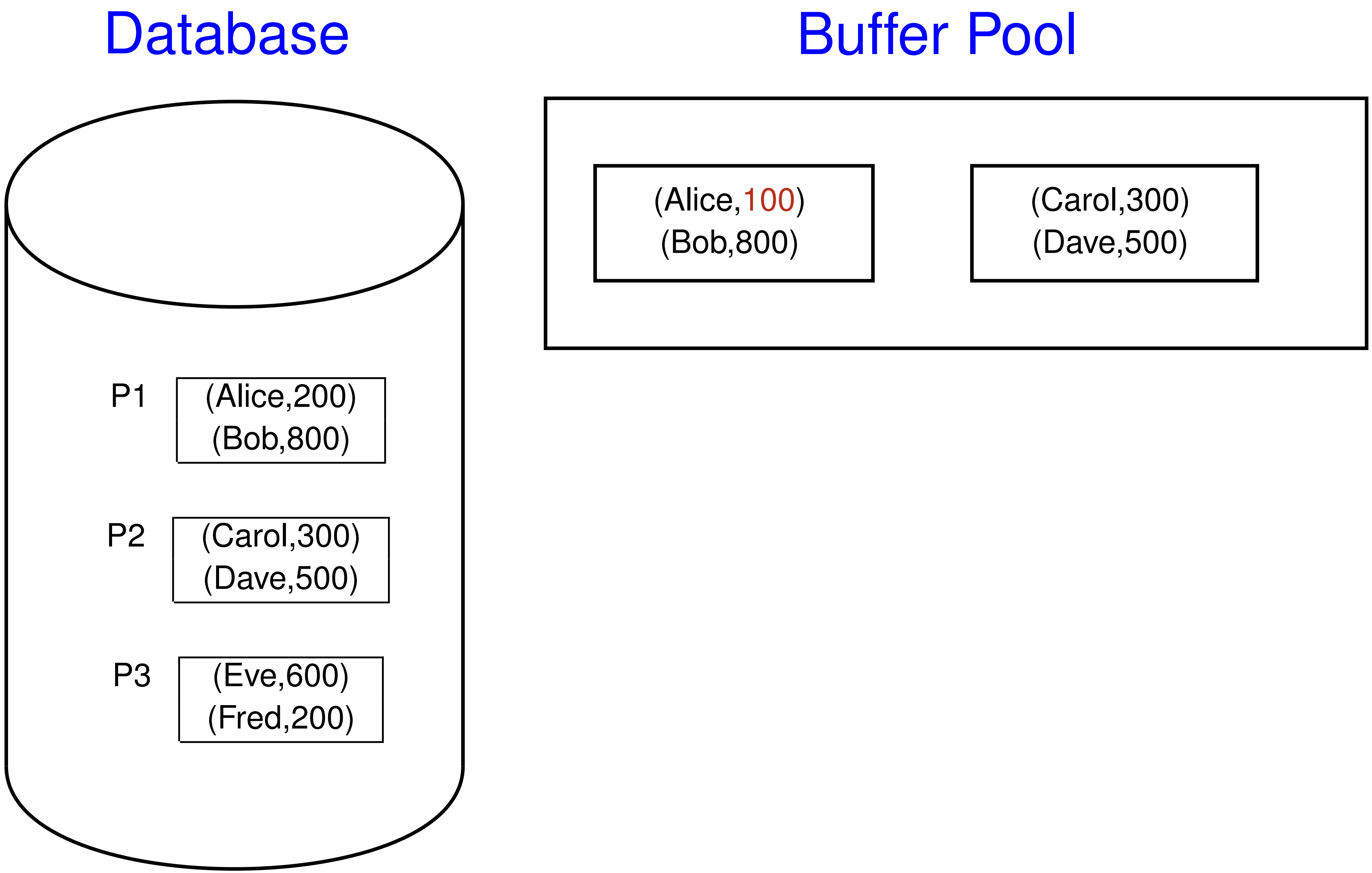

$R_1$(Alice,200): P1 is read into buffer pool

# Example: Crash Recovery (cont.)



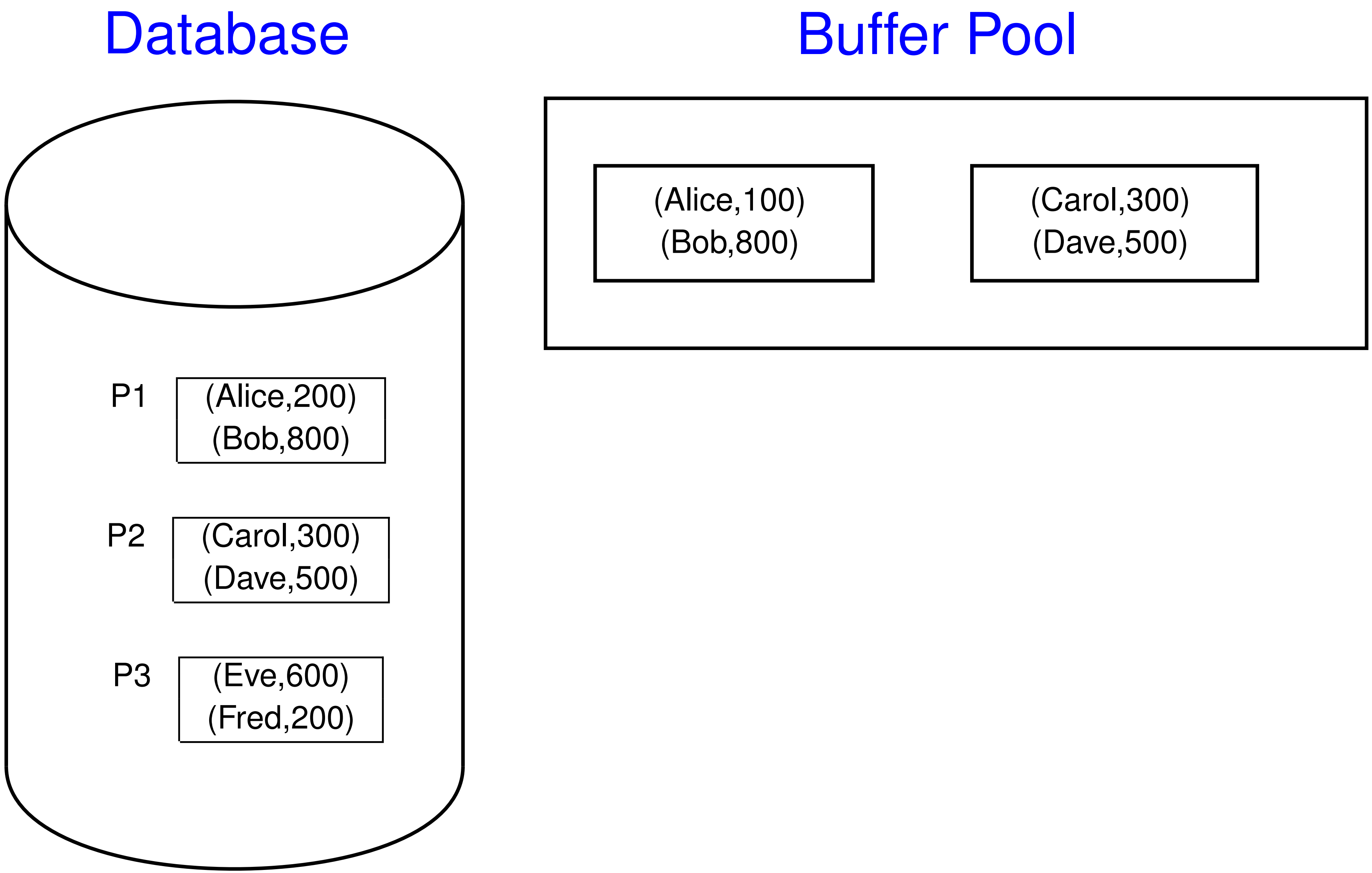

$R_1$(Carol,300): P2 is read into buffer pool

# Example: Crash Recovery (cont.)



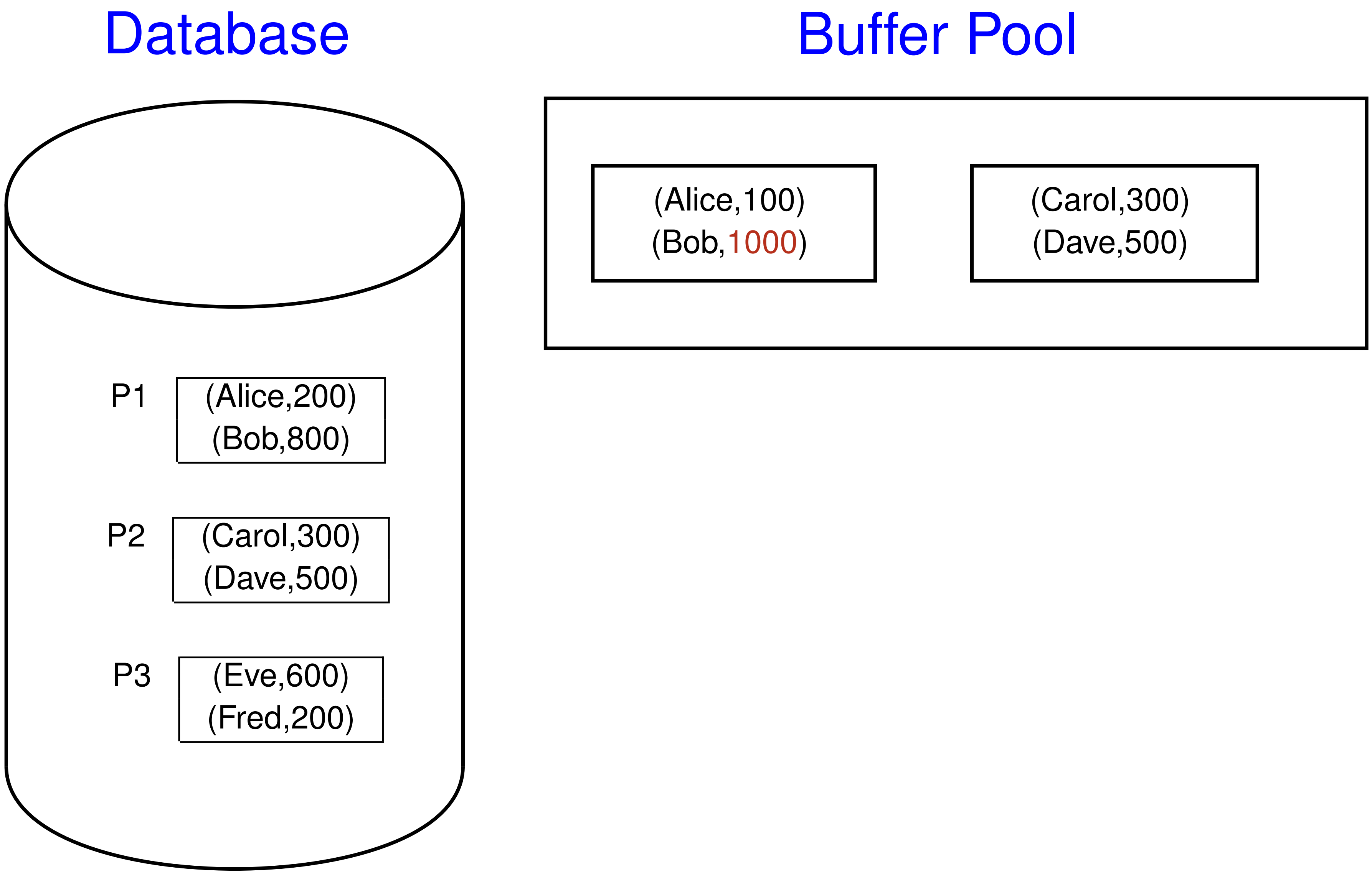

$W_1$(Alice, 100)

# Example: Crash Recovery (cont.)



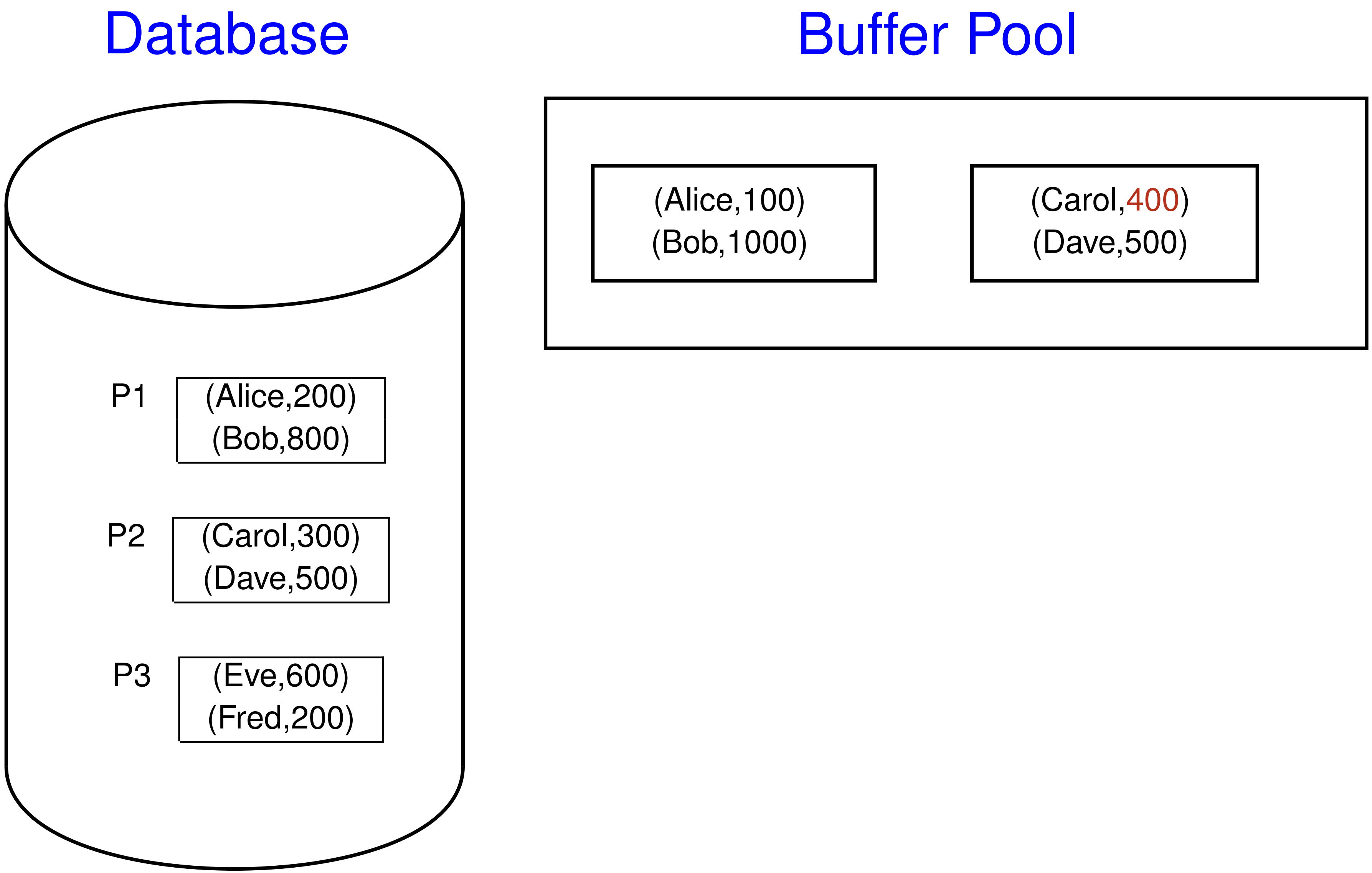

$R_2$(Bob,800)

# Example: Crash Recovery (cont.)



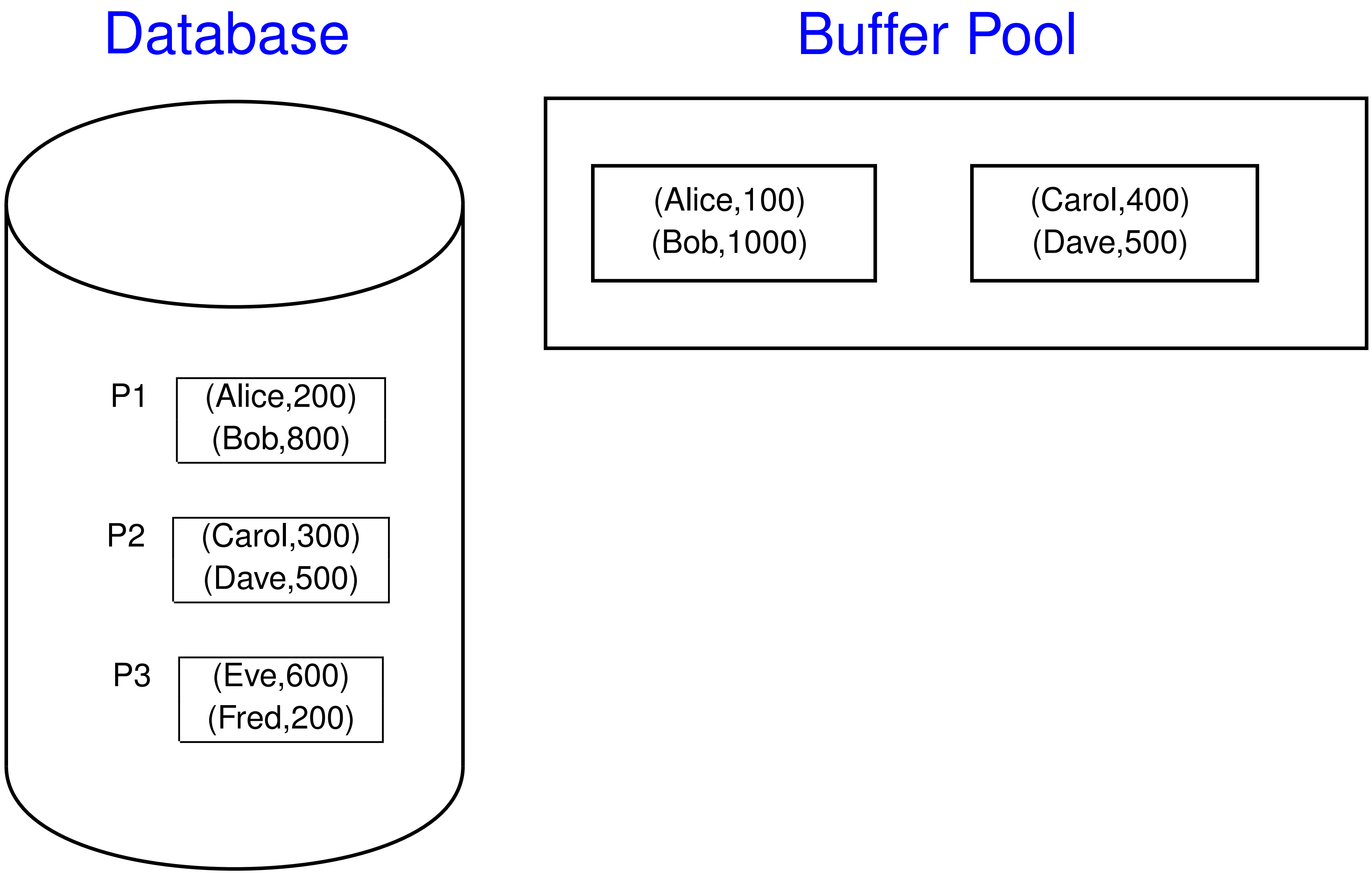

$W_2$(Bob,1000), $Commit_2$

# Example: Crash Recovery (cont.)



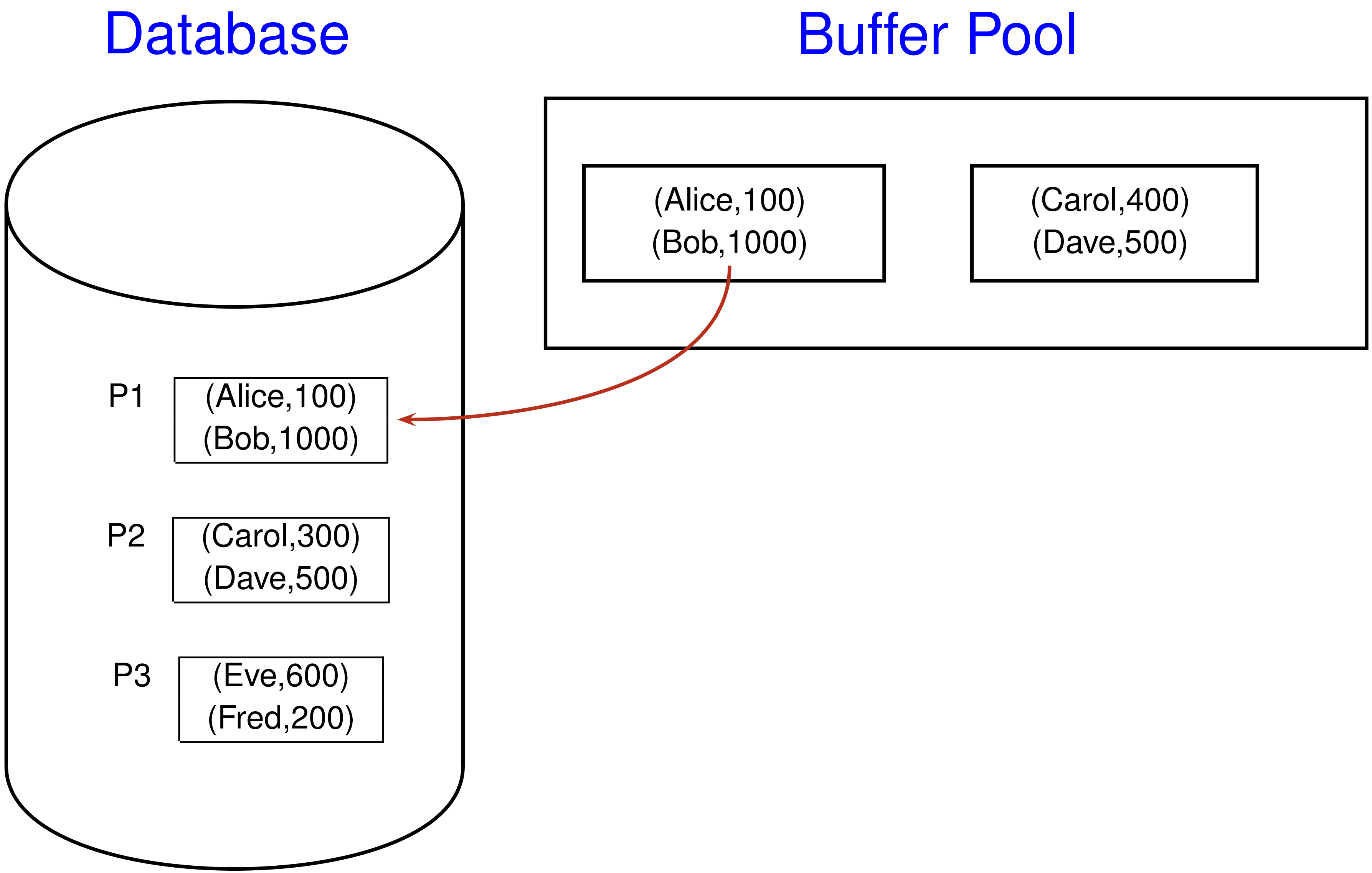

$W_1$(Carol,400)

# Example: Crash Recovery (cont.)



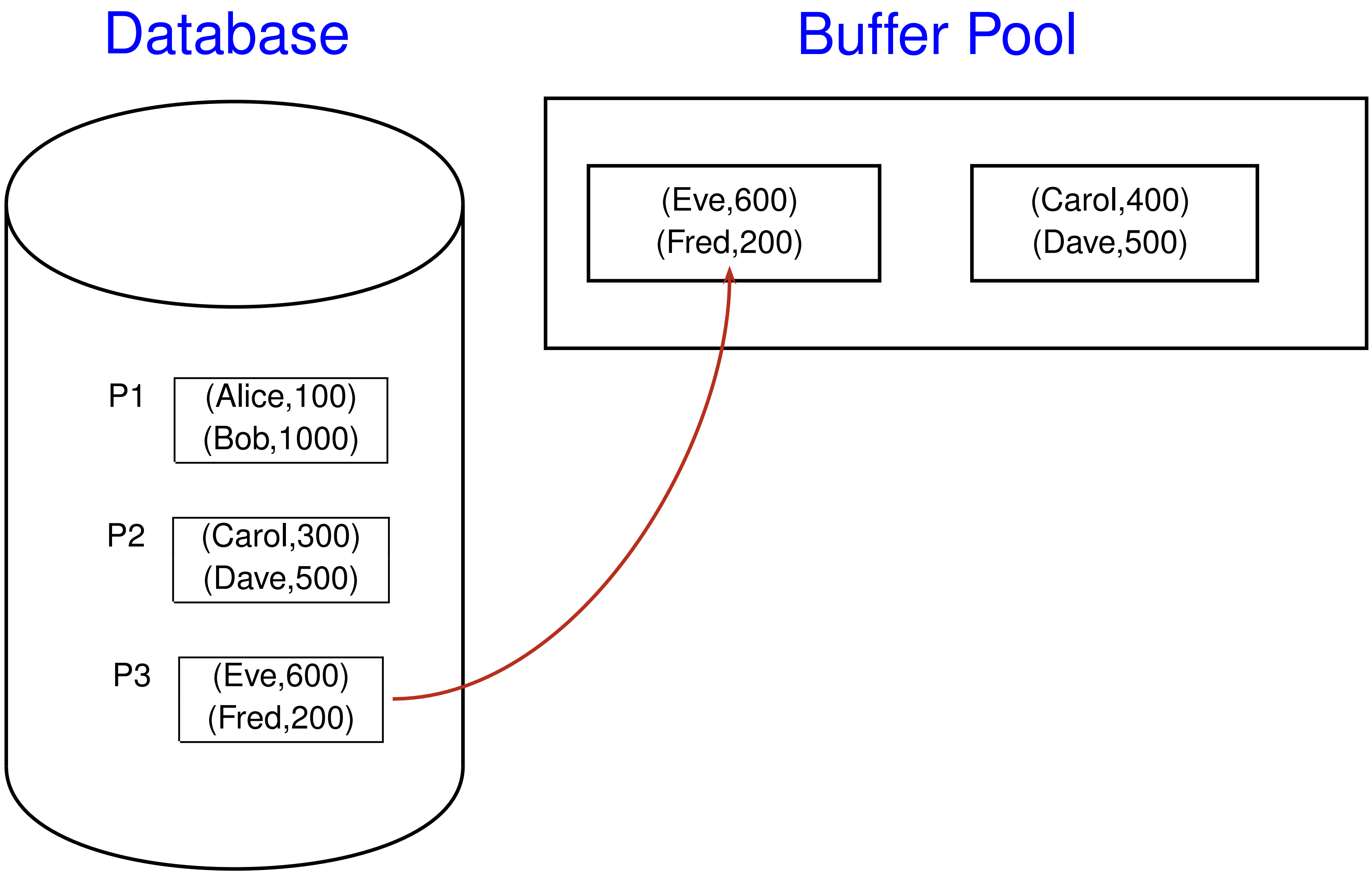

$R_3$(Eve,600): buffer pool manager picks P1 for replacement

# Example: Crash Recovery (cont.)




$R_3$(Eve,600): P1 is written to disk

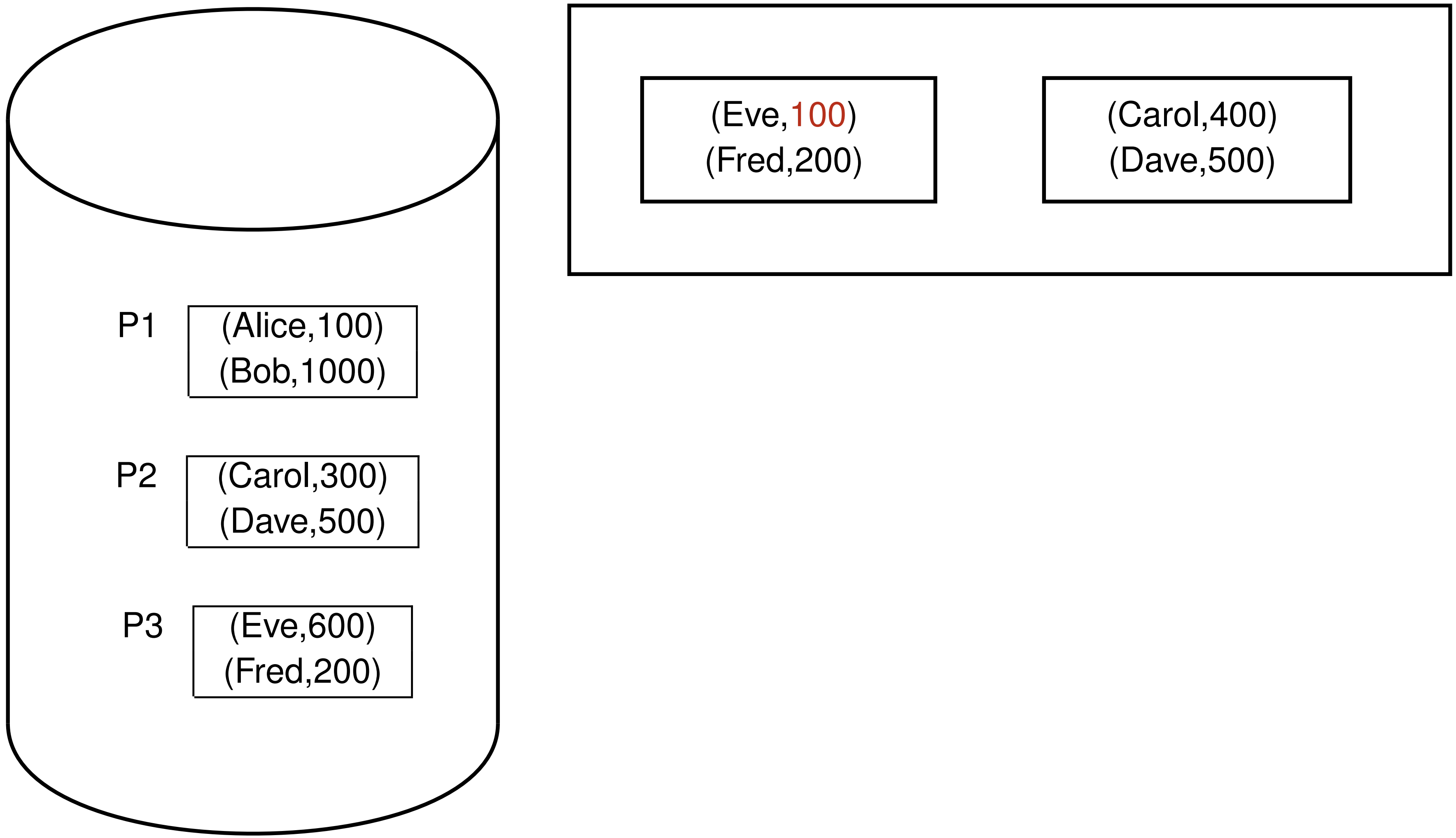
# Example: Crash Recovery (cont.)




$R_3$(Eve,600): P3 is read into buffer pool

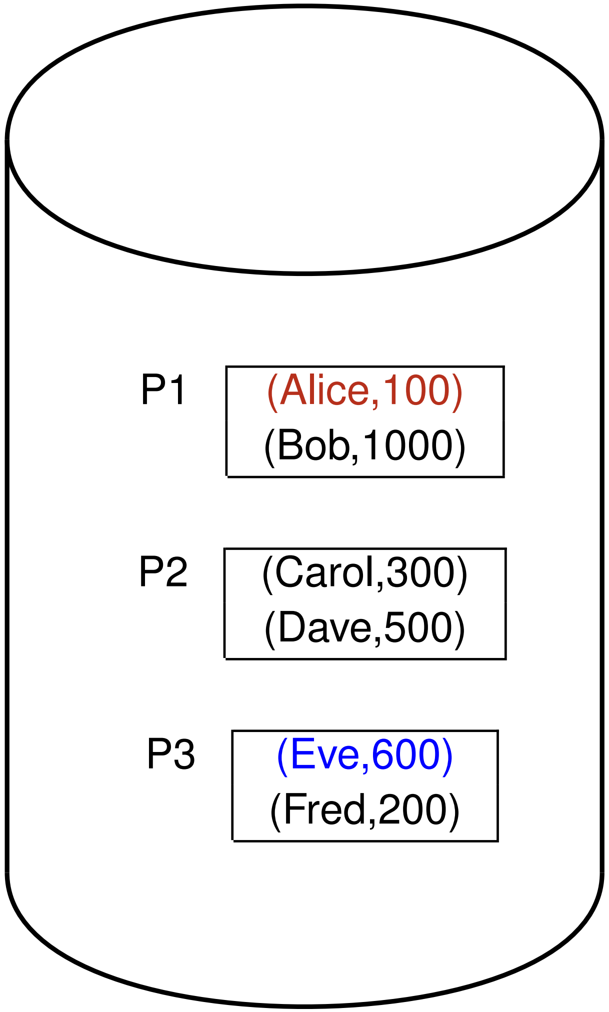
# Example: Crash Recovery (cont.)

Database　　　　Buffer Pool




$W_3$(Eve,100), $Commit_3$

# Example: Crash Recovery (cont.)




| **Step** | **$T_1$** | **$T_2$** | **$T_3$** |
|---|---|---|---|
| 1 | $R_1$(Alice, 200) | | |
| 2 | $R_1$(Carol, 300) | | |
| 3 | $W_1$(Alice, 100) | | |
| 4 | | $R_2$(Bob, 800) | |
| 5 | | $W_2$(Bob, 1000) | |
| 6 | | $Commit_2$ | |
| 7 | $W_1$(Carol, 400) | | |
| 8 | | | $R_3$(Eve, 600) |
| 9 | | | $W_3$(Eve, 100) |
| 10 | | | $Commit_3$ |
| 11 | | SYSTEM CRASH! | |

- Need to abort $T_1$: undo $W_1$(Alice, 100) $\longrightarrow$ (Alice,200)
- Need to redo $W_3$(Eve, 600) $\longrightarrow$ (Eve,100)

# Log-based Database Recovery

- Log (aka trail/journal): history of actions executed by DBMS
  - Contains a log record for each write, commit, & abort
- Log is stored as a sequential file of records in stable storage
- Log is stored in stable storage (storage that can survive crashes & media failures)
  - Implemented by maintaining multiple copies of information (possibly at different locations) on non-volatile storage devices
- Each log record has a unique identifier called **Log Sequence Number (LSN)**
  - LSN of an earlier log record is smaller than that of a later log record

# ARIES Recovery Algorithm

- Algorithm for Recovery and Isolation Exploiting Semantics
- Designed to work with a steal, no-force approach
- Assumes strict 2PL for concurrency control
- ARIES is invented by IBM and used in several products (e.g., DB2, Microsoft SQL Server)

# Recovery-related Structures

- Log file
- Transaction table (TT)
  - One entry for each active Xact
  - Each entry contains:
    - XactID: Transaction identifier
    - lastLSN: LSN of the most recent log record for this Xact
    - Xact status (C or U)

      C = Xact has committed

      U = Xact has not committed
- Dirty page table (DPT)
  - One entry for each dirty page in buffer pool
  - Each entry contains:
    - pageID: page ID of dirty page
    - recLSN: LSN of the earliest log record for an update that caused the page to be dirty
- All these structures are updated during normal processing

# Log records

Information in log records

- type of log record (e.g., update, commit, abort)
- identifier of Xact
- prevLSN: LSN of the previous log record for the same Xact
- other type-specific information

Update log record

- identifier of page being updated
- before-image of update
- after-image of update

# Implementing Abort

- Undo all updates by Xact to database pages
- Write-ahead logging (WAL) protocol
  Do not flush an uncommitted update to the database until the log record containing its before-image has been flushed to the log
- How to enforce WAL protocol?
  - Each database page contains the LSN of the most recent log record (a.k.a. pageLSN) that describes an update to this page
  - Before flushing a **database page P** to disk, ensure that all the log records up to the log record corresponding to **P's pageLSN** have been flushed to disk

# Implementing Abort (cont.)

- How to undo all the updates of Xact?
  - For each log record of Xact in reverse order, restore log record's before-image
- How to efficiently retrieve Xact's log records in reverse order?
  - Xact Table (TT) - maintains one entry for each active Xact
  - Each TT entry stores the LSN of most recent log record for Xact (a.k.a. lastLSN)
  - Use **lastLSN** to retrieve the most recent log record for Xact; the other log records for Xact are retrieved via the **prevLSN** of each retrieved log record
- Logging Changes During Undo: Changes made to database while undoing a Xact are also logged to ensure that an action is not repeated in the event of repeated undos

# Implementing Commit

- Need to ensure that that all the updates of Xact must be in stable storage (database or log) before Xact is committed
- Force-at-commit protocol

  Do not commit a Xact until the after-images of all its updated records are in stable storage (database or log)
- How to enforce force-at-commit protocol?
  - Write a commit log record for Xact
  - Flush all the log records for Xact to disk
- Xact is considered to have committed if its commit log record has been written to stable storage

# Implementing Restart

- Recovery from system crashes consists of three phases:
  1. Analysis phase: identifies dirtied buffer pool pages & active Xacts at time of crash
  2. Redo phase: redo actions to restore database state to what it was at the time of crash
  3. Undo phase: undo actions of Xacts that did not commit
- Repeating History During Redo: During restart following a crash, first restore system to the state before crash, and then undo the actions of Xacts that are active at the time of crash

# Normal Transaction Processing

## Updating Xact Table (transID, lastLSN, status)

- When the first log record is created for Xact *T*, create a new entry for *T* with status = U
- When a new log record *r* is created for Xact *T*, update `lastLSN` for *T*'s entry to be r's LSN
- If Xact *T* commits, update `status` for *T*'s entry to be C
- When an end log record is generated for Xact *T*, remove *T*'s entry

## Updating Dirty Page Table (pageID, recLSN)

- When a page *P* in buffer pool is updated & DPT has no entry for *P*, create a new table entry for *P* with recLSN = LSN of log record corresponding to update
- When a dirtied page *P* in buffer pool is flushed to disk, remove entry for *P*

# Normal Xact Processing: Example

**LOG**

| | prevLSN | XactID | type | pageID | length | offset | before image | after image |
|---|---|---|---|---|---|---|---|---|
| 10 | - | $T_1$ | update | P500 | 3 | 21 | ABC | DEF |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |

**XACT TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 10 | U |

# Normal Xact Processing: Example

**LOG**

| | prevLSN | XactID | type | pageID | length | offset | before image | after image |
|---|---|---|---|---|---|---|---|---|
| 10 | - | $T_1$ | update | P500 | 3 | 21 | ABC | DEF |
| 20 | - | $T_2$ | update | P600 | 3 | 41 | HIJ | KLM |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |

**XACT TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 10 | U |
| $T_2$ | 20 | U |

# Normal Xact Processing: Example

**LOG**

| | prevLSN | XactID | type | pageID | length | offset | before image | after image |
|---|---|---|---|---|---|---|---|---|
| 10 | - | $T_1$ | update | P500 | 3 | 21 | ABC | DEF |
| 20 | - | $T_2$ | update | P600 | 3 | 41 | HIJ | KLM |
| 30 | 20 | $T_2$ | update | P500 | 3 | 20 | GDE | QRS |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |

**XACT TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 10 | U |
| $T_2$ | 30 | U |

# Normal Xact Processing: Example

**LOG**

| | prevLSN | XactID | type | pageID | length | offset | before image | after image |
|---|---|---|---|---|---|---|---|---|
| 10 | - | $T_1$ | update | P500 | 3 | 21 | ABC | DEF |
| 20 | - | $T_2$ | update | P600 | 3 | 41 | HIJ | KLM |
| 30 | 20 | $T_2$ | update | P500 | 3 | 20 | GDE | QRS |
| 40 | 10 | $T_1$ | update | P505 | 3 | 21 | TUV | WXY |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**XACT TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |
| $T_2$ | 30 | U |

# Types of Log Records

- All log records have the following information:
  (1) type of log record (e.g., update, commit, abort),
  (2) identifier of Xact, and
  (3) prevLSN (LSN of the previous log record for the same Xact)
- Update log record
  - After updating a page P, create an update log record r
  - Update pageLSN of P = LSN of r
  - Additional fields in update log record:
    - **pageID**: identifier of page being updated
    - **offset**: byte offset within page indicating the beginning of updated portion
    - **length**: number of bytes for updated portion of data page
    - **before-image**: value of the changed bytes before update
    - **after-image**: value of the changed bytes after update

# Types of Log Records (cont.)

- **Compensation log record (CLR)**
  - When the update described by an update log record (ULR) is undone, create a compensation log record
  - Additional fields in CLR:
    - page ID
    - undoNextLSN = LSN of next log record to be undone (i.e., prevLSN in ULR)
    - action taken to undo update
- **Commit log record**
  - When a Xact is to be committed, create a commit log record r
  - All log records (up to and including r) are force-written to stable storage
  - Xact is considered committed once r has been written to stable storage

# Types of Log Records (cont.)

- Abort log record
  - When a Xact is to be aborted, create an abort log record
  - Undo is initiated for this Xact
- End log record
  - Once the additional follow-up processing initiated by a aborted/committed Xact has completed, create an end log record
- Checkpoint log record
  - To be discussed later
- Update log records & CLRs are classified as redoable log records

# Analysis Phase

- Performs three tasks:
  1. Determines the point in the log to start the Redo phase
  2. Determines the superset of buffer pages that were dirty at the time of crash
  3. Identifies active Xacts at the time of crash

# Analysis Phase (cont.)

- Initialize dirty page table (DPT) & Xact table (TT) to be empty
- Scan the log in forward direction to process each log record r (for Xact T):
  - If r is an end log record
    - Remove *T* from TT

    Else
    - Add an entry in TT for *T* if *T* is not in TT
    - Update lastLSN of entry to be r's LSN
    - Update status of entry to C if *r* is a commit log record
  - If (*r* is a redoable log record for page *P*) & (*P* is not in DPT), then
    - Create an entry for P in DPT with pageID of entry = P's pageID and recLSN of entry = r's LSN
- At the end of Analysis phase:
  - Xact table = list of all active Xacts (with status = U) at time of crash
  - dirty page table = superset of dirty pages at time of crash

# Analysis Phase: Example

**LOG BUFFER**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |
| $T_2$ | 30 | U |

# Analysis Phase: Example

**LOG BUFFER**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |
| $T_2$ | 50 | C |

# Analysis Phase: Example

**LOG BUFFER**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

# Analysis Phase: Example

**LOG BUFFER**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |
| 70 | update: (40, $T_1$,P700,3,51,PQR,IJK) |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |
| P700 | 70 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 70 | U |

# Analysis Phase: Example

**LOG BUFFER**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |
| 70 | update: (40, $T_1$,P700,3,51,PQR,IJK) |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |
| P700 | 70 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 70 | U |

- Suppose that the system crashes & all the log records (except for the last record) have been flushed to disk before the crash

# Analysis Phase: Example (cont.)

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| | |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| | | |

# Analysis Phase: Example (cont.)

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 10 | U |

# Analysis Phase: Example (cont.)

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| → 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 10 | U |
| $T_2$ | 20 | U |

# Analysis Phase: Example (cont.)

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| → 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 10 | U |
| $T_2$ | 30 | U |

# Analysis Phase: Example (cont.)

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| $\rightarrow$ 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |
| $T_2$ | 30 | U |

# Analysis Phase: Example (cont.)

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| → 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |
| $T_2$ | 50 | C |

# Analysis Phase: Example (cont.)

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| $\rightarrow$ 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

# Analysis Phase: Example (cont.)

**LOG**

| LSN | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

- Active Xacts: $T_1$
- Dirty Pages: P500, P600, P505

# Redo Phase

- RedoLSN = smallest recLSN among all dirty pages in DPT
- Let *r* be the log record with LSN = RedoLSN
- Scan the log in forward direction starting from *r*
- If (*r* is an update log record) or (*r* is a CLR) then
  - fetch page *P* that is associated with *r*
  - If (P's pageLSN $<$ r's LSN) then
    - ★ Reapply logged action in *r* to *P*
    - ★ update P's pageLSN = r's LSN
- At the end of Redo Phase,
  - Create end log records for Xacts with status = C in Xact Table & remove their entries from Xact Table
  - System is restored to state at time of crash

# Redo Phase: Example

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

- RedoLSN = min$\{10, 20, 40\} = 10$

# Redo Phase: Example

**LOG**

| | |
|---|---|
| → 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

- RedoLSN = min$\{10, 20, 40\} = 10$
- log record r with LSN = 10: **Update: (-,$T_1$,P500,3,21,ABC,DEF)**
  - Fetch page P500. Suppose P500's pageLSN = 10
  - r's LSN $=$ P500's pageLSN: No need to redo update on P500

# Redo Phase: Example

**LOG**

| | |
|---|---|
| 10 | update: (-,$T_1$,P500,3,21,ABC,DEF) |
| → 20 | update: (-,$T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20,$T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10,$T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

- RedoLSN = $\min\{10, 20, 40\} = 10$
- log record r with LSN = 20: **Update: (-,$T_2$,P600,3,41,HIJ,KLM)**
  - Fetch page P600. Suppose P600's pageLSN = 20
  - r's LSN $=$ P600's pageLSN: No need to redo update on P600

# Redo Phase: Example

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| → 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

- RedoLSN = min$\{10, 20, 40\} = 10$
- log record r with LSN = 30: **Update: (20,$T_2$,P500,3,30,GDE,QRS)**
  - Fetch page P500. P500's pageLSN = 10
  - r's LSN $>$ P500's pageLSN: Redo update on P500
  - Update P500's pageLSN = 30

# Redo Phase: Example

**LOG**

| LSN | |
|---|---|
| 10 | update: (-, $T_1$,P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$,P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$,P500,3,20,GDE,QRS) |
| $\rightarrow$ 40 | update: (10, $T_1$,P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

- RedoLSN = min$\{10, 20, 40\} = 10$
- log record r with LSN = 40: **Update: (10,$T_1$,P505,3,31,TUV,WXY)**
  - Fetch page P505. Suppose P505's pageLSN = 40
  - r's LSN $=$ P505's pageLSN: No need to redo update on P505

# Redo Phase: Example

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$, P500, 3, 21, ABC, DEF) |
| 20 | update: (-, $T_2$, P600, 3, 41, HIJ, KLM) |
| 30 | update: (20, $T_2$, P500, 3, 20, GDE, QRS) |
| 40 | update: (10, $T_1$, P505, 3, 21, TUV, WXY) |
| $\rightarrow$ 50 | commit: $T_2$ |
| 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

- RedoLSN = $\min\{10, 20, 40\} = 10$

# Redo Phase: Example

**LOG**

| | |
|---|---|
| 10 | update: (-, $T_1$, P500,3,21,ABC,DEF) |
| 20 | update: (-, $T_2$, P600,3,41,HIJ,KLM) |
| 30 | update: (20, $T_2$, P500,3,20,GDE,QRS) |
| 40 | update: (10, $T_1$, P505,3,21,TUV,WXY) |
| 50 | commit: $T_2$ |
| $\rightarrow$ 60 | end: $T_2$ |

**DIRTY PAGE TABLE**

| pageID | recLSN |
|---|---|
| P500 | 10 |
| P600 | 20 |
| P505 | 40 |

**TRANSACTION TABLE**

| XactID | lastLSN | status |
|---|---|---|
| $T_1$ | 40 | U |

- RedoLSN = min$\{10, 20, 40\} = 10$
- System is restored to state at time of crash

# Undo Phase

- **Goal**: abort active Xacts at time of crash (loser Xacts)
  - Abort loser Xacts by undoing their actions in reverse order
- Initialize L = set of lastLSNs (with status = U) from TT
- Repeat until L becomes empty
  - delete the largest lastLSN from L
  - let r be the log record corresponding to this lastLSN
  - if r is an update log record for Xact *T* on page *P* then
    - create a CLR $r_2$ for *T*: $r_2$'s undoNextLSN = r's prevLSN
    - update *T*'s entry in TT: lastLSN = $r_2$'s LSN
    - undo the logged action on page *P*
    - update P's pageLSN = $r_2$'s LSN
    - `Update-L-and-TT` (r's prevLSN)
  - else if r is a CLR for Xact *T* then
    - `Update-L-and-TT` (r's undoNextLSN)
  - else if r is an abort log record for Xact *T* then `Update-L-and-TT` (r's prevLSN)
- **Update-L-and-TT** (lsn)
  - if lsn is not null then add lsn to L
  - else create an end log record for *T* & remove *T*'s entry from TT

# Undo Phase: Example

- Recall that at the end of Analysis Phase, $T_1$ is the only active Xact to be undone ($T_1$'s lastLSN in TT is 40)

| LSN | LOG |
| --- | --- |
| 10 | update: $T_1$ writes P500 |
| 20 | update: $T_2$ writes P600 |
| 30 | update: $T_2$ writes P500 |
| 40 | update: $T_1$ writes P505 |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |
| | CRASH |

# Undo Phase: Example

- Recall that at the end of Analysis Phase, $T_1$ is the only active Xact to be undone ($T_1$'s lastLSN in TT is 40)

| **LSN** | **LOG** |
|---|---|
| 10 | update: $T_1$ writes P500 |
| 20 | update: $T_2$ writes P600 |
| 30 | update: $T_2$ writes P500 |
| 40 | update: $T_1$ writes P505 |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |
| | CRASH |
| 70 | CLR: Undo $T_1$ LSN 40 |

undoNextLSN

# Undo Phase: Example

- Recall that at the end of Analysis Phase, $T_1$ is the only active Xact to be undone ($T_1$'s lastLSN in TT is 40)

| LSN | LOG |
|---|---|
| 10 | update: $T_1$ writes P500 |
| 20 | update: $T_2$ writes P600 |
| 30 | update: $T_2$ writes P500 |
| 40 | update: $T_1$ writes P505 |
| 50 | commit: $T_2$ |
| 60 | end: $T_2$ |
| | CRASH |
| 70 | CLR: Undo $T_1$ LSN 40 |
| 80 | CLR: Undo $T_1$ LSN 10 |
| 90 | end: $T_1$ |

undoNextLSN

# Undo Phase: Another Example

| LSN | LOG |
|---|---|
| 10 | update: $T_1$ writes P5 |
| 20 | update: $T_2$ writes P3 |
| 30 | abort: $T_1$ |
| 40 | CLR: Undo $T_1$ LSN 10 |
| 45 | end: $T_1$ |
| 50 | update: $T_3$ writes P1 |
| 60 | update: $T_2$ writes P5 |
| | CRASH |

# Undo Phase: Another Example

| LSN | LOG |
|---|---|
| 10 | update: $T_1$ writes P5 |
| 20 | update: $T_2$ writes P3 |
| 30 | abort: $T_1$ |
| 40 | CLR: Undo $T_1$ LSN 10 |
| 45 | end: $T_1$ |
| 50 | update: $T_3$ writes P1 |
| 60 | update: $T_2$ writes P5 |
| | CRASH |

- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 60 | U |
| $T_3$ | 50 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{50, 60\}$

# Undo Phase: Another Example

| LSN | LOG |
|---|---|
| 10 | update: $T_1$ writes P5 |
| 20 | update: $T_2$ writes P3 |
| 30 | abort: $T_1$ |
| 40 | CLR: Undo $T_1$ LSN 10 |
| 45 | end: $T_1$ |
| 50 | update: $T_3$ writes P1 |
| 60 | update: $T_2$ writes P5 |
| | CRASH |
| 70 | CLR: Undo $T_2$ LSN 60 |

- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 60 | U |
| $T_3$ | 50 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{50, 60\}$
  - LSN=60: $L = \{20, 50\}$

# Undo Phase: Another Example

| LSN | LOG |
|---|---|
| 10 | update: $T_1$ writes P5 |
| 20 | update: $T_2$ writes P3 |
| 30 | abort: $T_1$ |
| 40 | CLR: Undo $T_1$ LSN 10 |
| 45 | end: $T_1$ |
| 50 | update: $T_3$ writes P1 |
| 60 | update: $T_2$ writes P5 |
| | CRASH |
| 70 | CLR: Undo $T_2$ LSN 60 |
| 80 | CLR: Undo $T_3$ LSN 50 |
| 85 | end: $T_3$ |

- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 60 | U |
| $T_3$ | 50 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{50, 60\}$
  - LSN=60: $L = \{20, 50\}$
  - LSN=50: $L = \{20\}$

# Undo Phase: Another Example

| LSN | LOG |
|---|---|
| 10 | update: $T_1$ writes P5 |
| 20 | update: $T_2$ writes P3 |
| 30 | abort: $T_1$ |
| 40 | CLR: Undo $T_1$ LSN 10 |
| 45 | end: $T_1$ |
| 50 | update: $T_3$ writes P1 |
| 60 | update: $T_2$ writes P5 |
| | CRASH |
| 70 | CLR: Undo $T_2$ LSN 60 |
| 80 | CLR: Undo $T_3$ LSN 50 |
| 85 | end: $T_3$ |
| | CRASH |

- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 60 | U |
| $T_3$ | 50 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{50, 60\}$
  - LSN=60: $L = \{20, 50\}$
  - LSN=50: $L = \{20\}$

# Undo Phase: Another Example

| LSN | LOG |
|---|---|
| 10 | update: $T_1$ writes P5 |
| 20 | update: $T_2$ writes P3 |
| 30 | abort: $T_1$ |
| 40 | CLR: Undo $T_1$ LSN 10 |
| 45 | end: $T_1$ |
| 50 | update: $T_3$ writes P1 |
| 60 | update: $T_2$ writes P5 |
| | CRASH |
| 70 | CLR: Undo $T_2$ LSN 60 |
| 80 | CLR: Undo $T_3$ LSN 50 |
| 85 | end: $T_3$ |
| | CRASH |

- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 60 | U |
| $T_3$ | 50 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{50, 60\}$
  - LSN=60: $L = \{20, 50\}$
  - LSN=50: $L = \{20\}$
- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 70 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{70\}$

# Undo Phase: Another Example

| LSN | LOG |
|---|---|
| 10 | update: $T_1$ writes P5 |
| 20 | update: $T_2$ writes P3 |
| 30 | abort: $T_1$ |
| 40 | CLR: Undo $T_1$ LSN 10 |
| 45 | end: $T_1$ |
| 50 | update: $T_3$ writes P1 |
| 60 | update: $T_2$ writes P5 |
| | CRASH |
| 70 | CLR: Undo $T_2$ LSN 60 |
| 80 | CLR: Undo $T_3$ LSN 50 |
| 85 | end: $T_3$ |
| | CRASH |

- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 60 | U |
| $T_3$ | 50 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{50, 60\}$
  - LSN=60: $L = \{20, 50\}$
  - LSN=50: $L = \{20\}$
- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 70 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{70\}$
  - LSN=70: $L = \{20\}$

# Undo Phase: Another Example

| LSN | LOG |
|---|---|
| 10 | update: $T_1$ writes P5 |
| 20 | update: $T_2$ writes P3 |
| 30 | abort: $T_1$ |
| 40 | CLR: Undo $T_1$ LSN 10 |
| 45 | end: $T_1$ |
| 50 | update: $T_3$ writes P1 |
| 60 | update: $T_2$ writes P5 |
| | CRASH |
| 70 | CLR: Undo $T_2$ LSN 60 |
| 80 | CLR: Undo $T_3$ LSN 50 |
| 85 | end: $T_3$ |
| | CRASH |
| 90 | CLR: Undo $T_2$ LSN 20 |
| 95 | end: $T_2$ |

- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 60 | U |
| $T_3$ | 50 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{50, 60\}$
  - LSN=60: $L = \{20, 50\}$
  - LSN=50: $L = \{20\}$
- Analysis Phase:

| pageID | recLSN |
|---|---|
| P1 | 50 |
| P3 | 20 |
| P5 | 10 |

| XactID | lastLSN | status |
|---|---|---|
| $T_2$ | 70 | U |

- Redo Phase: RedoLSN = 10
- Undo Phase: $L = \{70\}$
  - LSN=70: $L = \{20\}$
  - LSN=20: $L = \{\}$

# Checkpointing

- Perform checkpoint operations periodically to speed up restart recovery
- Checkpointing synchronizes state of log with database state

## Simple Checkpointing

1. Stop accepting any new update, commit, & abort operations
2. Wait till all active update, commit, & abort operations have finished
3. Flush all dirty pages in buffer
4. Write a checkpoint log record containing Xact table
5. Resume accepting new update, commit, & abort operations

- During restart recovery, Analysis Phase begins with the latest checkpoint log record (CPLR)
  - Initialize Xact table with CPLR's Xact table
  - Initialize dirty page table to be empty

# Fuzzy Checkpointing in ARIES

1. Let **DPT'** be the dirty page table & **TT'** be the Xact table
2. Write a begin_checkpoint log record
3. Write a end_checkpoint log record containing **DPT'** & **TT'**
4. Write a special master record containing the LSN of the **begin_checkpoint log record** to a known place on stable storage

- During restart recovery, Analysis Phase starts with the **begin_checkpoint log record** (BCPLR) identified by the **master record**
  - Let ECPLR denote the **end_checkpoint log record** (ECPLR) corresponding to BCPLR
  - Assume that there are no log records between BCPLR & ECPLR
  - Initialize Xact table with ~~BCPLR's~~ ECPLR's Xact table
  - Initialize dirty page table with ~~BCPLR's~~ ECPLR's dirty page table

# Revisiting Redo Phase

- RedoLSN = smallest recLSN among all dirty pages in DPT
- Let *r* be the log record with LSN = RedoLSN
- Scan the log in forward direction starting from *r*
- If (*r* is an update log record) or (*r* is a CLR) then
  - fetch page *P* that is associated with *r*
  - If (P's pageLSN $<$ r's LSN) then
    - Reapply logged action in *r* to *P*
    - update P's pageLSN = r's LSN
- At the end of Redo Phase,
  - Create end log records for Xacts with status = C in Xact Table & remove their entries from Xact Table
  - System is restored to state at time of crash

# Revisiting Redo Phase: Optimization

- Goal: Exploit information in DPT to avoid retrieving $P$
- Optimization condition:
  ($P$ is not in DPT) or ($P$'s recLSN in DPT $>$ r's LSN)
- If optimization condition holds, then
  - the update of $r$ has already been applied to $P$
  - $r$ can be ignored & no need to fetch $P$!
- If ($r$ is redoable) and (optimization condition does not hold) then
  - fetch page $P$ that is associated with $r$
  - If (P's pageLSN $<$ r's LSN) then
    - ★ Reapply logged action in $r$ to $P$
    - ★ update P's pageLSN = r's LSN

    Else
    - ★ // recLSN $\leq$ r's LSN $\leq$ P's pageLSN
      Update P's entry in DPT: recLSN = P's pageLSN + 1