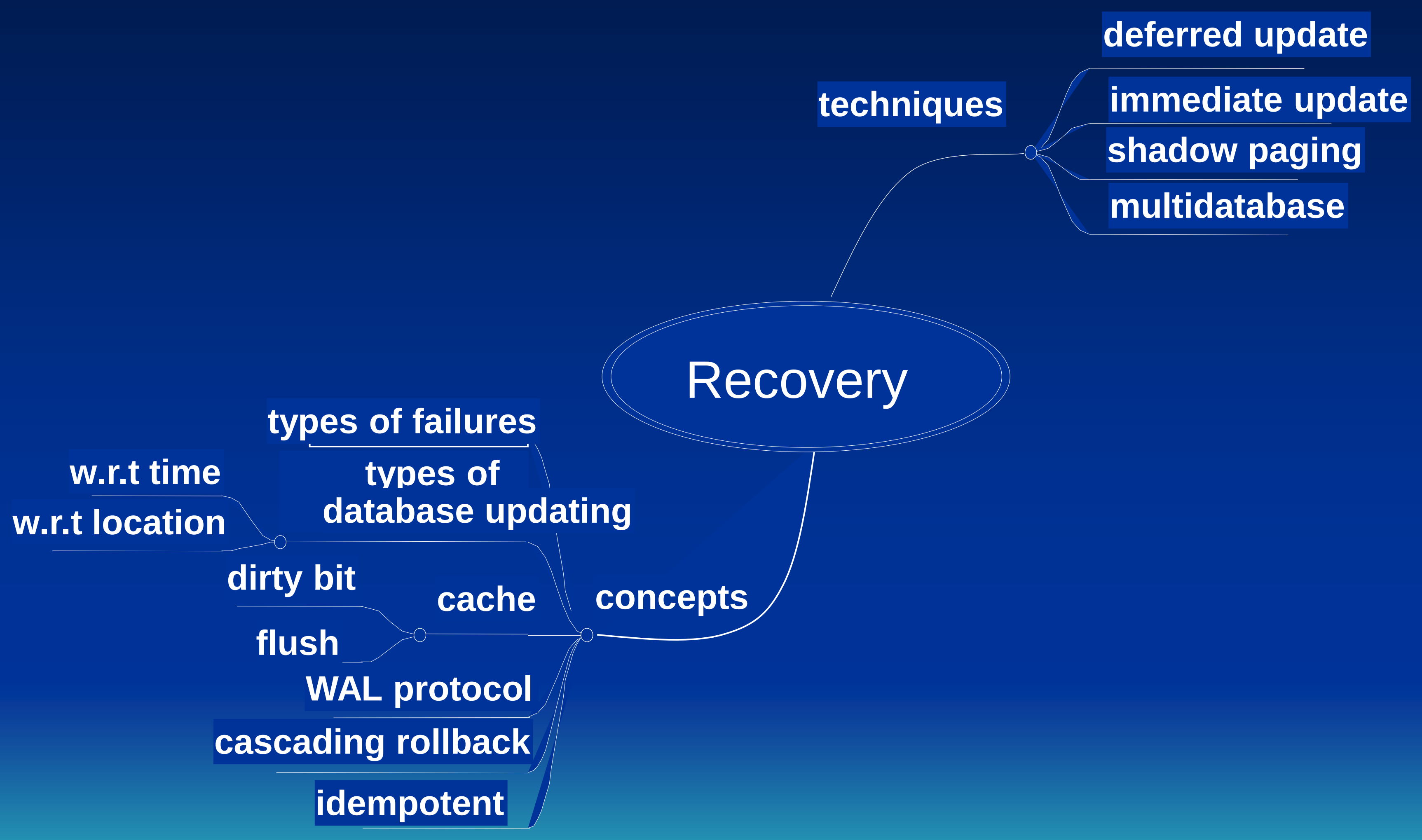
Database recovery techniques
(Ch. 21, 3rd ed. – Ch. 19, 4th and 5th ed. – Ch. 23, 6th ed. – Ch. 22, 7th ed.)



Recovery
techniques
deferred update
immediate update
shadow paging
multidatabase
concepts
types of failures
types of database updating
w.r.t time
w.r.t location
cache
dirty bit
flush
WAL protocol
cascading rollback
idempotent

Concepts

Recovery … "A database is restored to some state from the past so that a correct state - close to the time of failure - can be *reconstructed* from that past state"

Recovery is needed to ensure the atomicity of transactions, and their durability (**A**CI**D** properties)

- How is recovery implemented? ... typically a log plays an important part
    - *BFIM* - before image - an *undo* entry
    - *AFIM* - after image - a *redo* entry

Concepts

Failures are either:

- *catastrophic*

  to recover one restores the database using a past copy, followed by *redo*ing committed transaction operations

- *non-catastrophic*

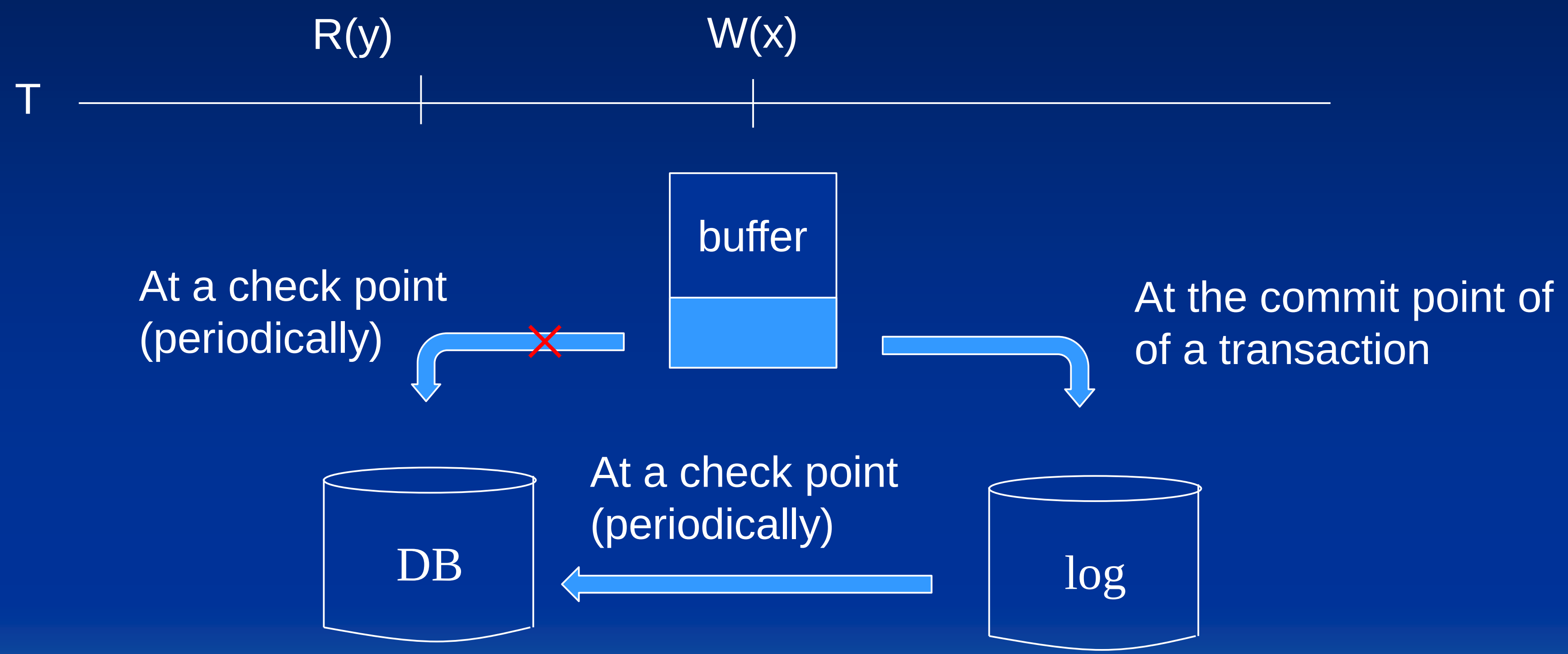
  to maintain atomicity and durability it may be necessary to:

  - *undo* some uncommitted database operations and
  - *redo* other committed database operations

Techniques

An update to the database is called a:

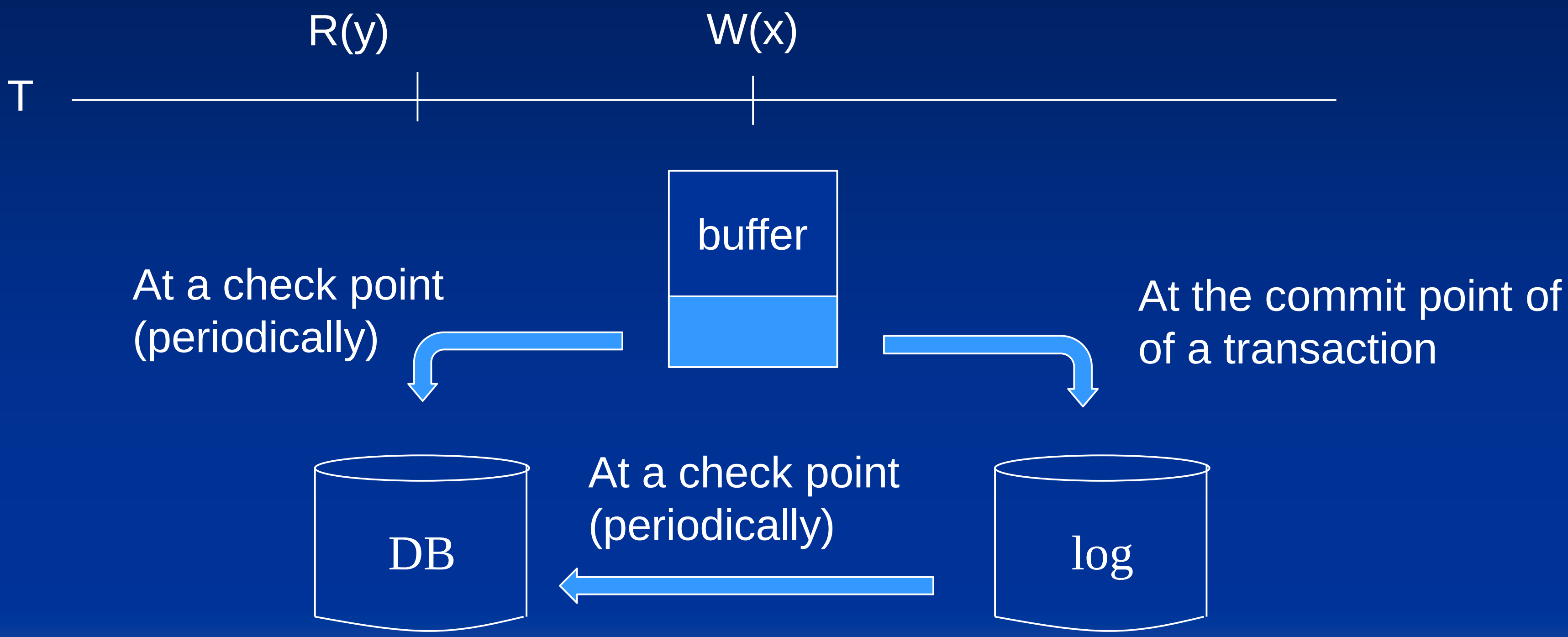
- *deferred update* - the database update does not actually occur until after a transaction reaches its commit point
    - When a transaction reaches its commit point all changes will be recorded (persistently) in the log.
    - However, at checkpoints, only the updates made by committed transactions are stored in database.
    - what are the implications for recovery?
        - Only redo is needed.
        - No undo


R(y)
W(x)
T
buffer
At a check point
(periodically)
At the commit point of
of a transaction
At a check point
(periodically)
DB
log

Techniques

An update to the database is called an:

- *immediate update* - the update can occur before a transaction reaches its commit point
  - At a checkpoint, all the updates made by committed and not yet committed transactions are stored in database.
  - a very typical situation in practice
  - what are the implications for recovery?
    - Both redo and undo are needed.

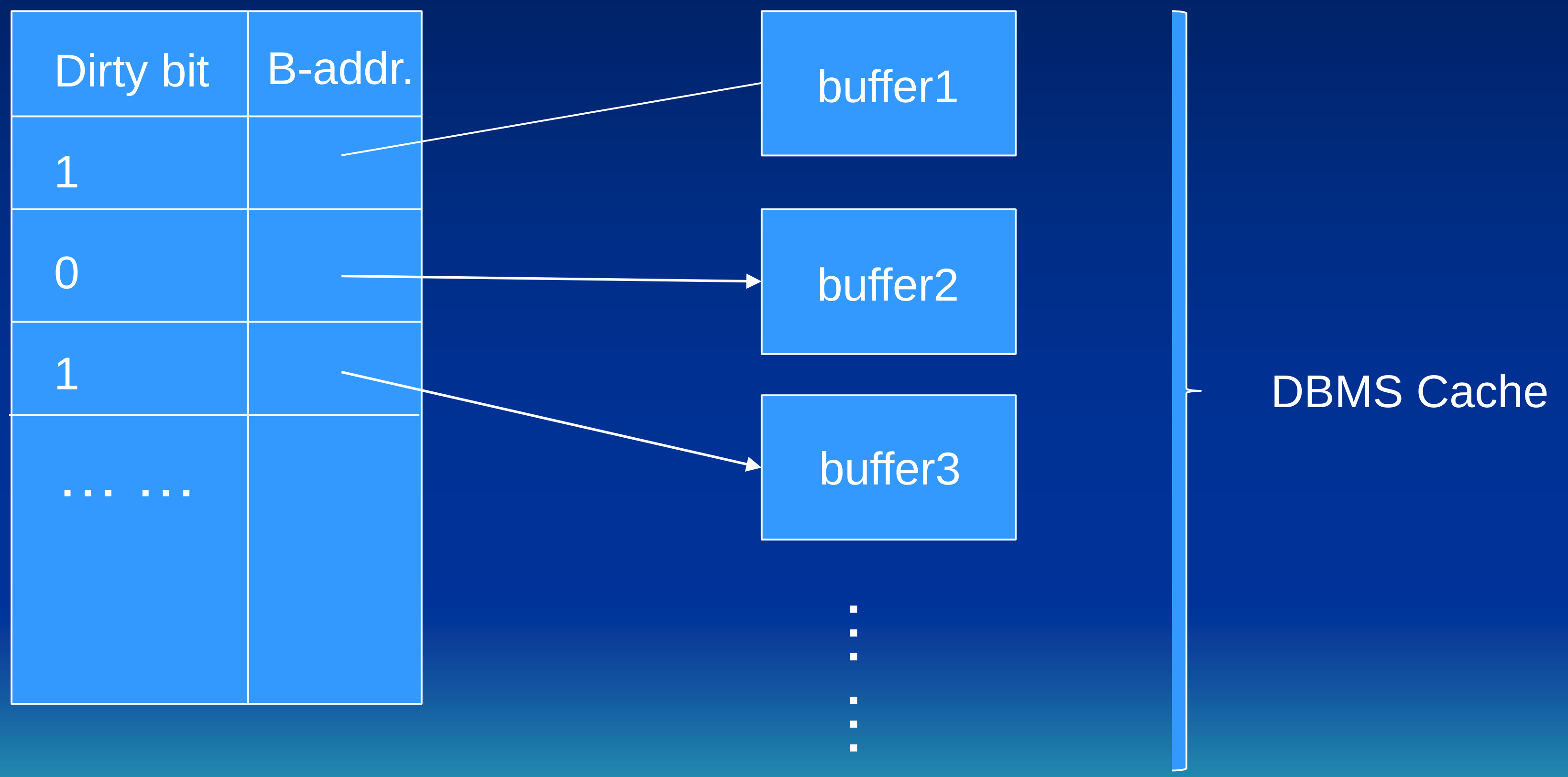

R(y)
W(x)
T
buffer
At a check point
(periodically)
At the commit point of
of a transaction
DB
At a check point
(periodically)
log

Concepts

disk pages are typically *cached* into main memory buffers

- we speak of the *DBMS cache* (a set of buffers)
- the DBMS uses a *directory* to access the cache
- the directory may have a *dirty bit* for each buffer to denote if the data in the buffer has been modified
- from time to time (at commit points, checking points) some of the cache buffers will be *flushed* to disk

directory



Concepts

- when data is written to disk it may be written:
  - as a *shadow*, or
  - *in-place* which requires the *write-ahead logging (WAL)* protocol:
    - a log file is needed, which keeps undo records (BFIM) and redo records (AFIM)
    - data records cannot be overwritten until the undo records have been force-written to the log on disk
    - redo records and the undo records must be force-written to the log on disk before the commit can be considered completed

Concepts

- *Cascading rollback* is a phenomenon where one transaction roll back causes another transaction to be rolled back
  - can be time-consuming
  - avoided with *cascadeless* or *strict* schedules
- Some recovery operations (undo, redo) must be *idempotent*:
  - Redoing a redo operation, REDO(REDO) should produce the same result as a single REDO - note that a system may crash shortly after being restarted, and so ...
  - Undoing an undo operation, UNDO(UNDO) should produce the same result as a single UNDO

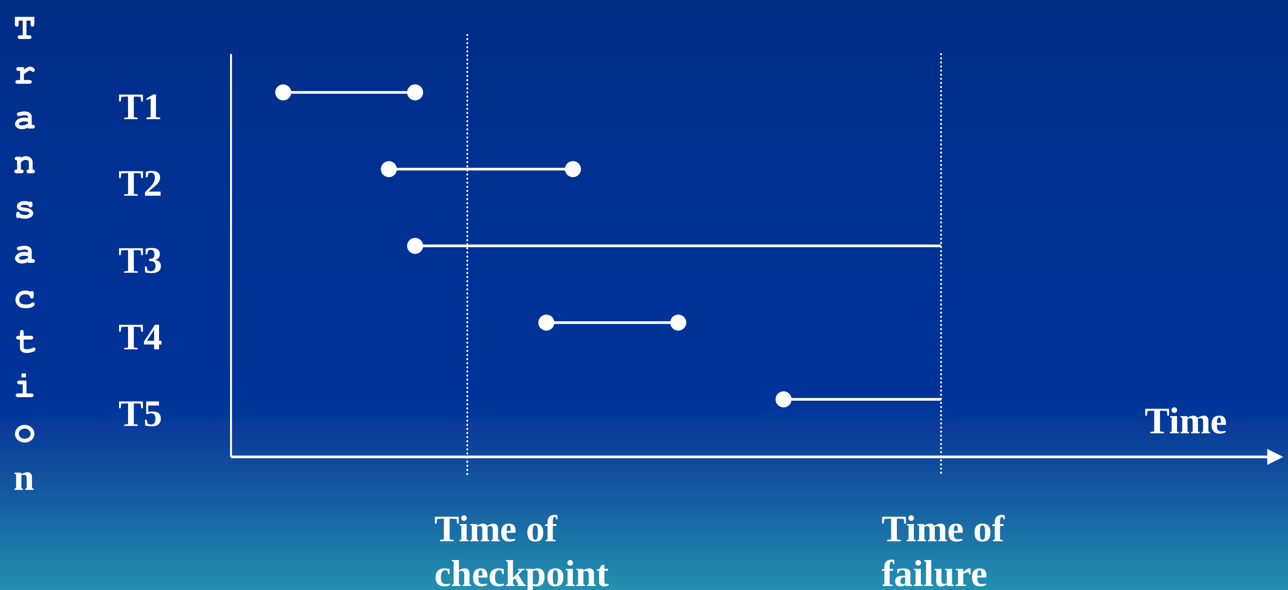
Recovery Technique for Deferred Update

- while a transaction is executing, *no updates* are made to the database and no undo will be required
- when a transaction commits, all updates are recorded in the log, the commit records are recorded in the log (reaches its commit point), and the log is force-written to the disk
    - a redo may be required if a failure occurs just after the commit record is written to log, but before it is written to database
    - no undo is required because the physical updating of the database hasn't happened yet

Recovery Technique for Deferred Update

Transaction types at recovery time

Consider the five types below. Which need to be redone after the crash?



No redo, no undo

redo

No redo, no undo

redo

ignored

Recovery Technique for Immediate Update

- while a transaction is executing, *updates may* be made to the database and so ***undo*** is required (WAL is needed)
- when a transaction has committed, either
  - all updates have been written to the database

    *(As part of commit, changes are written to the log and then to the database – no-**undo/no-redo**)*

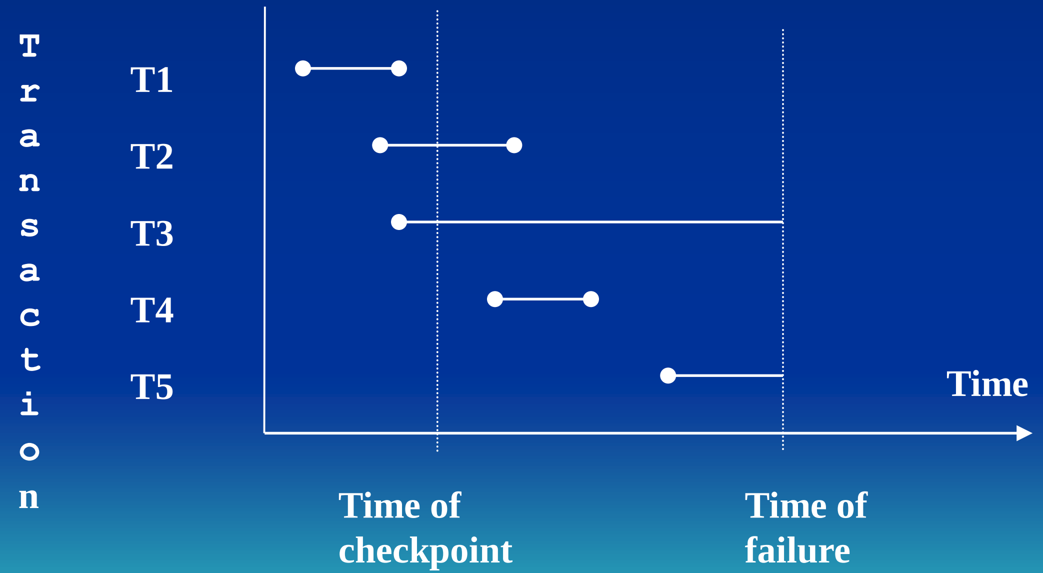
  - or not (only part of updates written to the database)

    *(very common, occurs in practice - **redo**)*

Recovery Technique for Immediate Update

Transaction types at recovery time

Consider the five types below. Which need to be undone / redone after the crash?



No redo, no undo

redo

undo

redo

ignored

Recovery Technique for Shadow Paging

What is shadow paging?

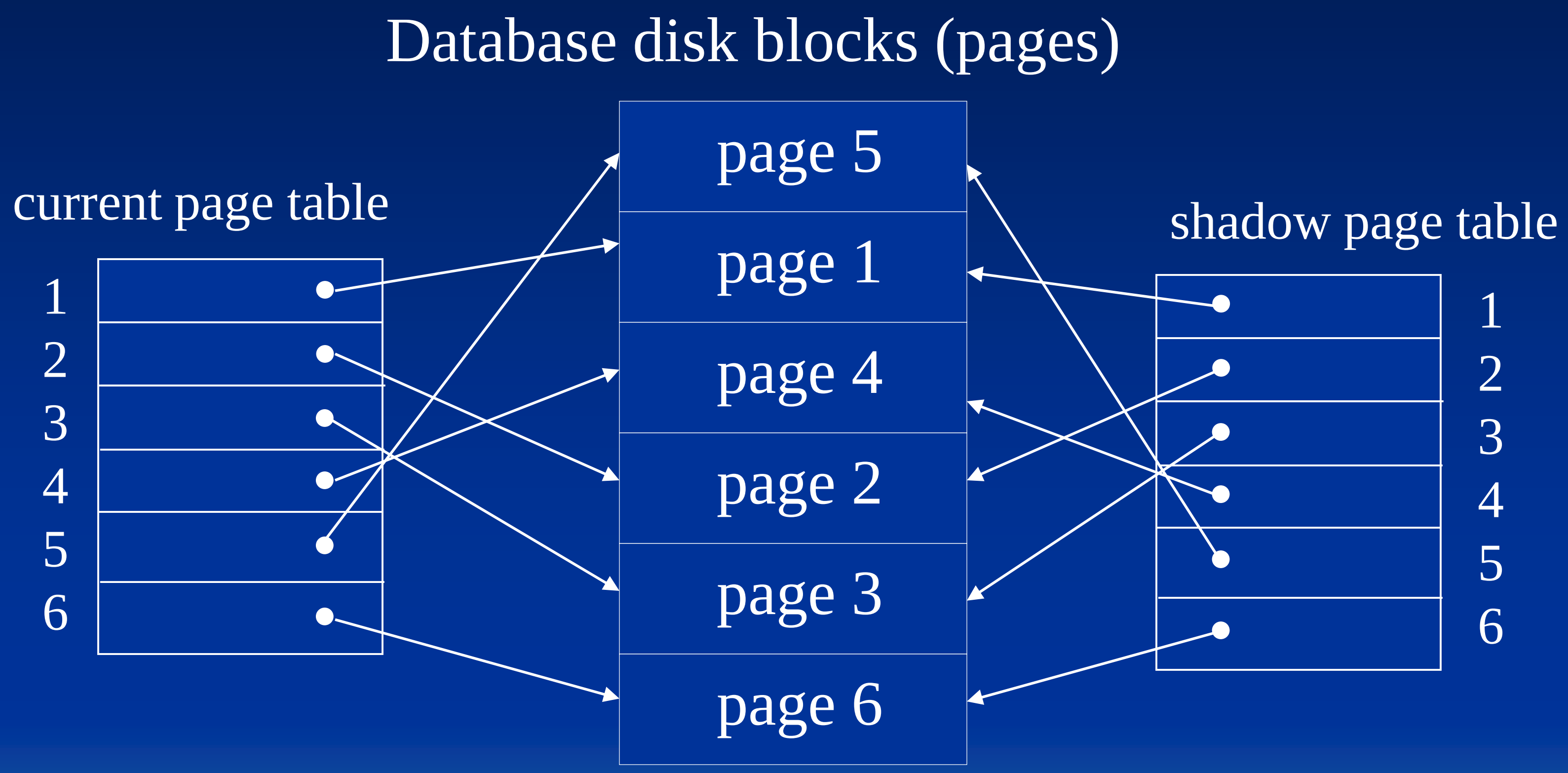
It is a technique pioneered in System R where changes are made to a copy of a page (block). When a transaction commits, the copy becomes the current page and the original is discarded

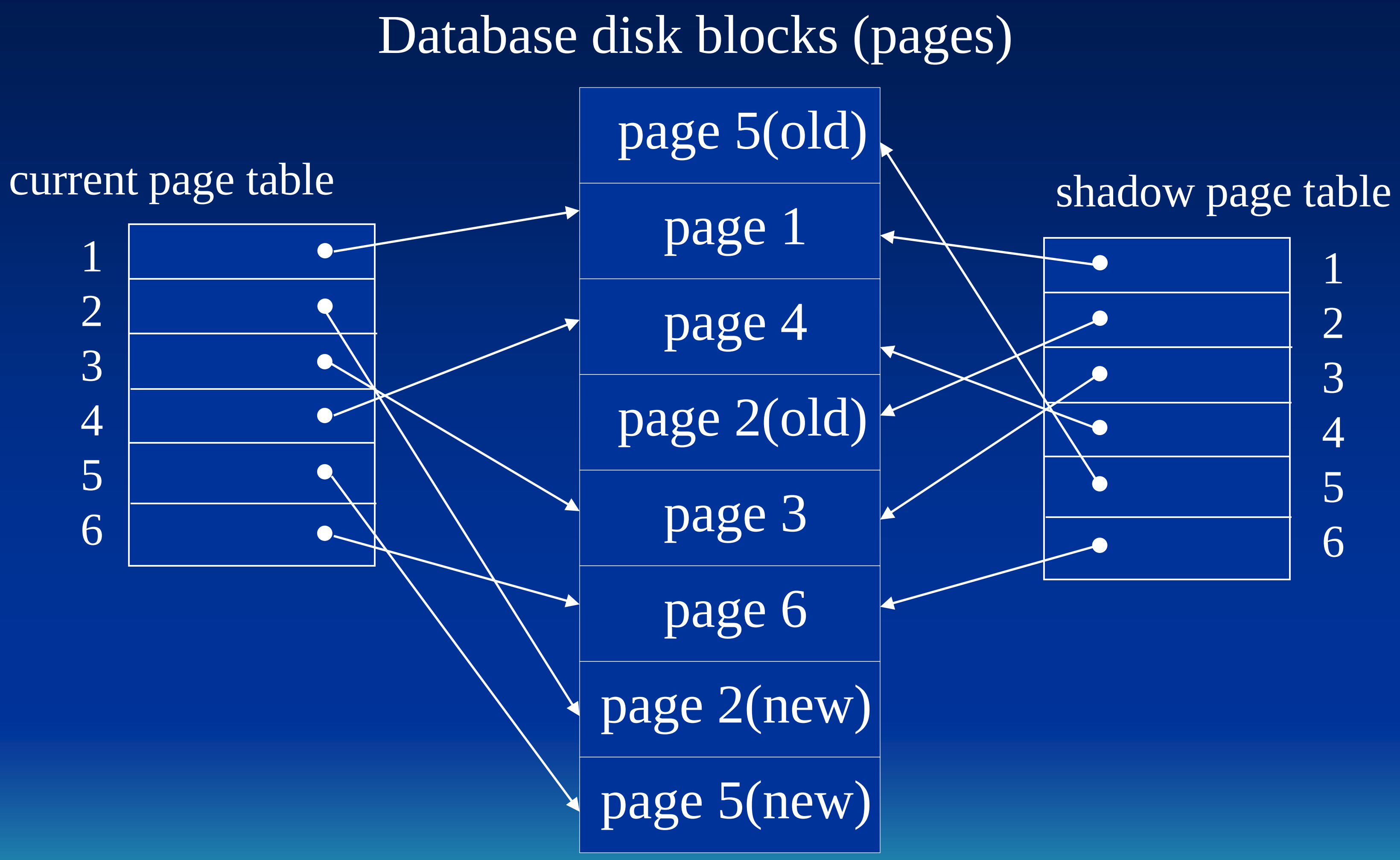
Recovery Technique for Shadow Paging

How a single transaction would be handled:

Suppose transaction A starts up:

- the current page table (directory) is copied to the shadow page table (shadow directory)
- if the transaction updates a page, the original page is not altered, rather a copy is created and that is modified
- the copy is pointed to by the current page table - the shadow page table is never modified


Database disk blocks (pages)
current page table
shadow page table
1
2
3
4
5
6
page 5
page 1
page 4
page 2
page 3
page 6
1
2
3
4
5
6


Database disk blocks (pages)
current page table
1
2
3
4
5
6
page 5(old)
page 1
page 4
page 2(old)
page 3
page 6
page 2(new)
page 5(new)
shadow page table
1
2
3
4
5
6

Recovery Technique for Shadow Paging

How a single transaction would be handled:

What is required to commit a transaction?

- free up any original pages that were updated
- discard the shadow page table

What is required if the system crashes while a transaction is executing?

- free up all modified pages
- discard the current page table
- reinstate the shadow page table as the current page table

Recovery Technique for Shadow Paging

Comments wrt Shadow Paging

- Appears simple for single transaction environments
- Complexity increases for concurrent transactions
- Data clustering diminishes quickly. Therefore, the system performance may be decreased.

Recovery Technique for multidatabase transactions

- includes distributed database environments

- situation occurs when database updates span more than one database system - to maintain atomicity we need the concept of a multidatabase, or distributed, transaction

- usual approach is to follow the *two-phase commit* protocol which involves
  - a coordinator (could be one of the database systems)
  - multiple DBMSs (participants)

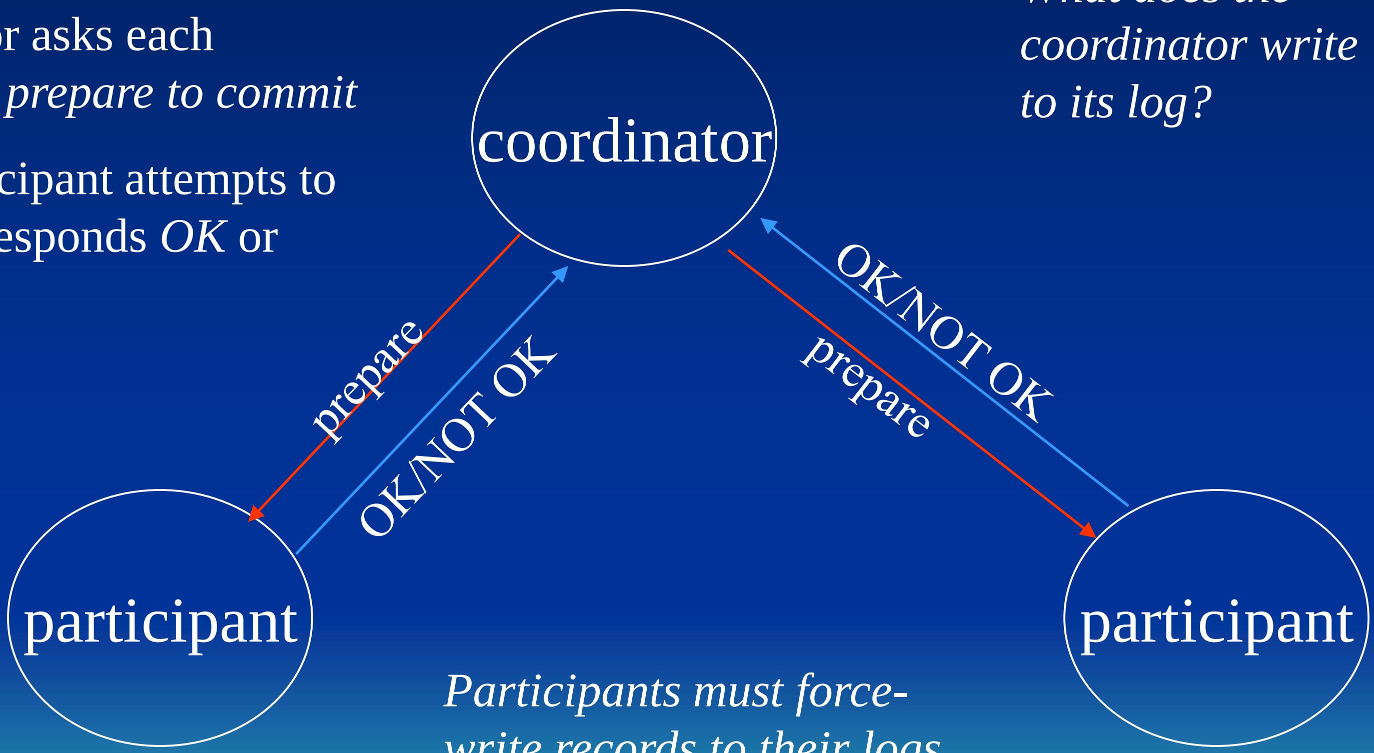
Recovery Technique for multidatabase transactions

Two-phase commit, phase I

1. Coordinator asks each participant to *prepare to commit*

2. Each participant attempts to prepare and responds *OK* or *NOT OK*

What does the coordinator write to its log?



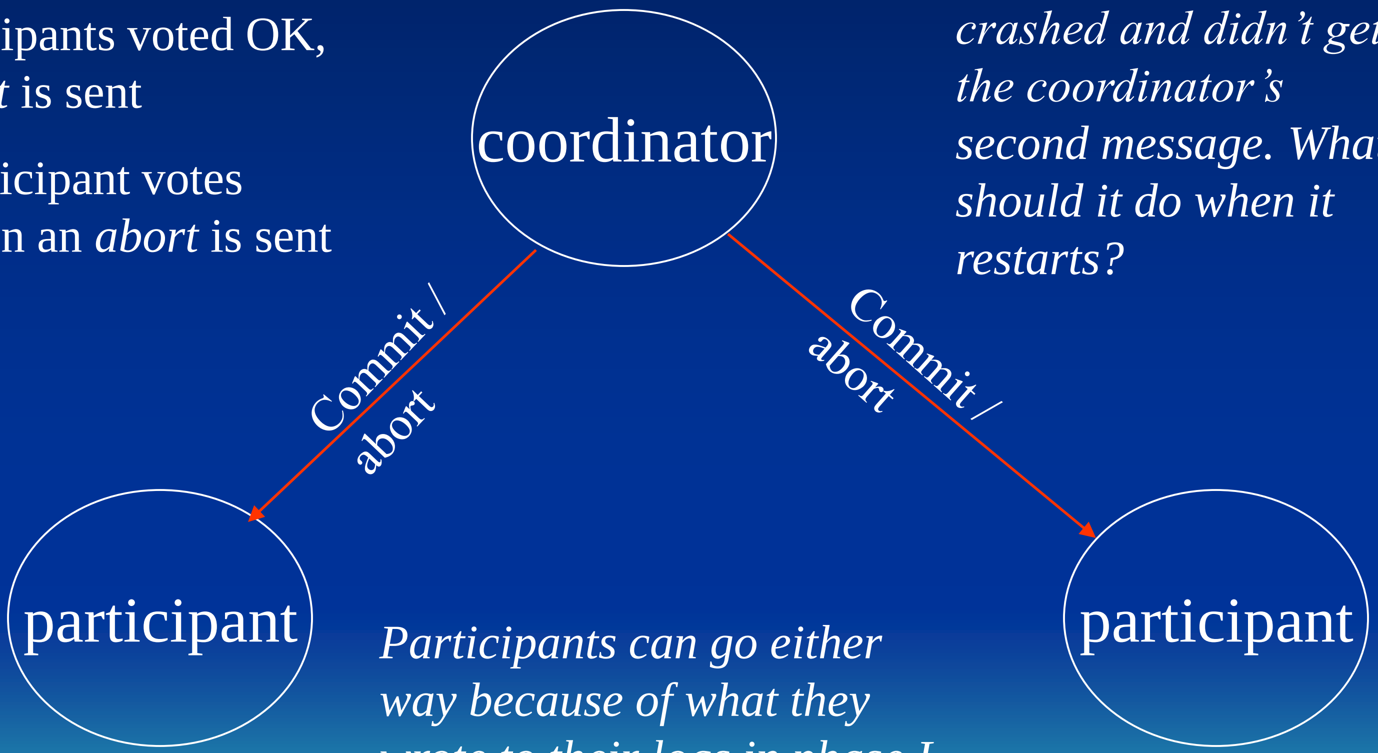
Participants must force-write records to their logs

Recovery Technique for multidatabase transactions

Two-phase commit, phase II

1. If all participants voted OK, then a *commit* is sent

2. If any participant votes NOT OK, then an *abort* is sent



Suppose a participant crashed and didn't get the coordinator's second message. What should it do when it restarts?

Participants can go either way because of what they wrote to their logs in phase I

Recovery Technique for multidatabase transactions

Any recovery manager complements some concurrency control manager

What might the concurrency control manager have that is related to multidatabase transactions? (Could deadlock occur?)