Inside the Progress OpenEdge RDBMS

Before-Images, Checkpoints, Crashes

Gus Björklund

PROGRESS EXCHANGE 2013
DISCOVER. DEVELOP. DELIVER.

Abstract

In this talk we examine the "before-image file", what it's for, how it works, and how you can configure it properly. You might get answers to questions that have been troubling people for over 25 * 10-2 centuries:

- Why doesn't the before-image file have before-images?
- Why aren't the data on disk ever current?
- What are checkpoints?
- Why do we have them?
- When your system crashes (and they all do eventually) how can the RDBMS recreate all the data that were lost in the crash and restore your database to a consistent state?

The OpenEdge RDBMS is brought to you by

PROGRESS

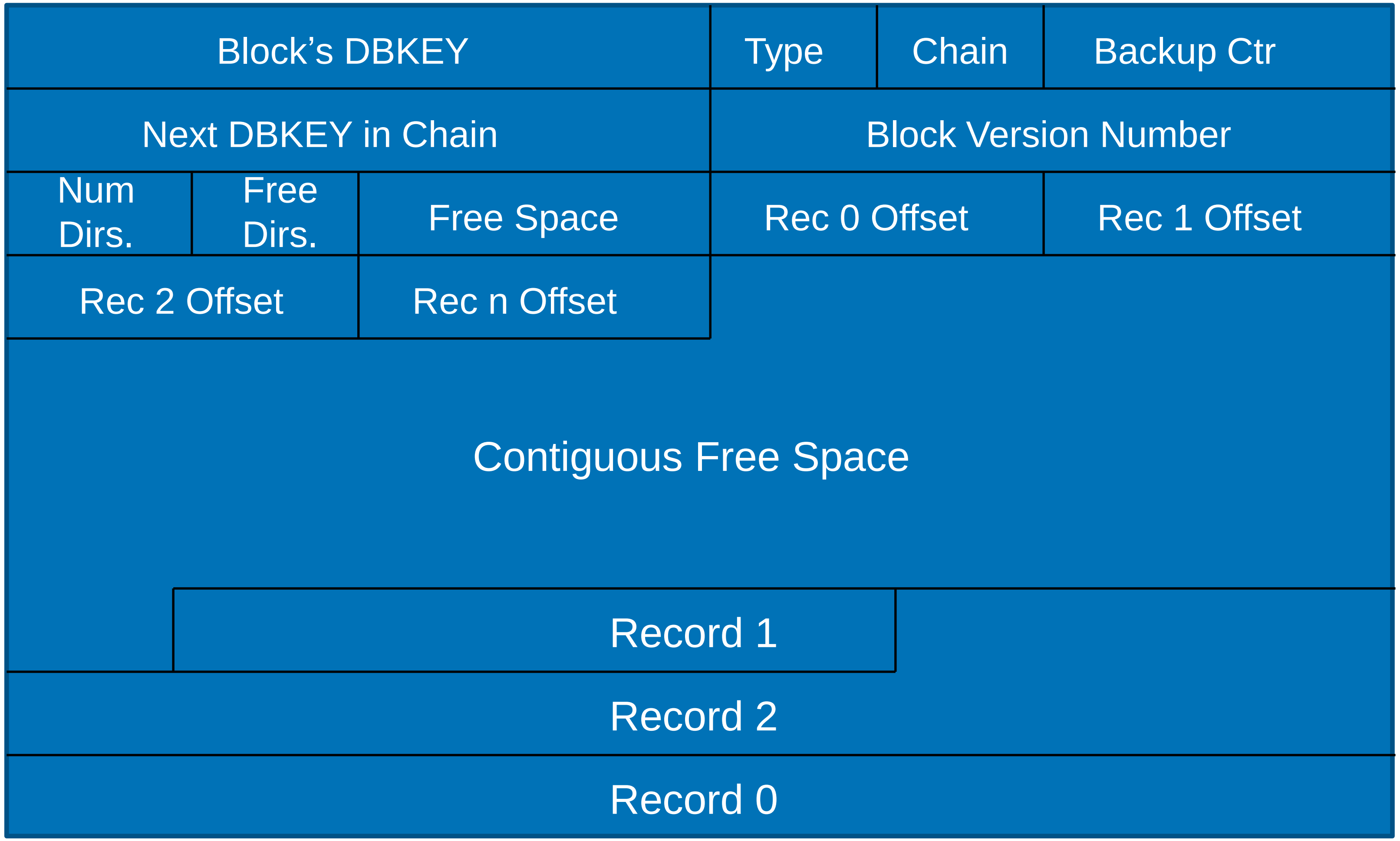
Engine Crew

Builders of The Best RDBMS
on the Third Planet From The Sun

The So-Called "Before-Image" File Is NOT

- Does not really contain before images
- It has a record of all *recent* database changes
- The data are sufficient to:
  - Undo or roll back transactions
  - Perform crash recovery

A Typical Data Block – for Records



Block's DBKEY

Type

Chain

Backup Ctr

Next DBKEY in Chain

Block Version Number

Num Dirs.

Free Dirs.

Free Space

Rec 0 Offset

Rec 1 Offset

Rec 2 Offset

Rec n Offset

Contiguous Free Space

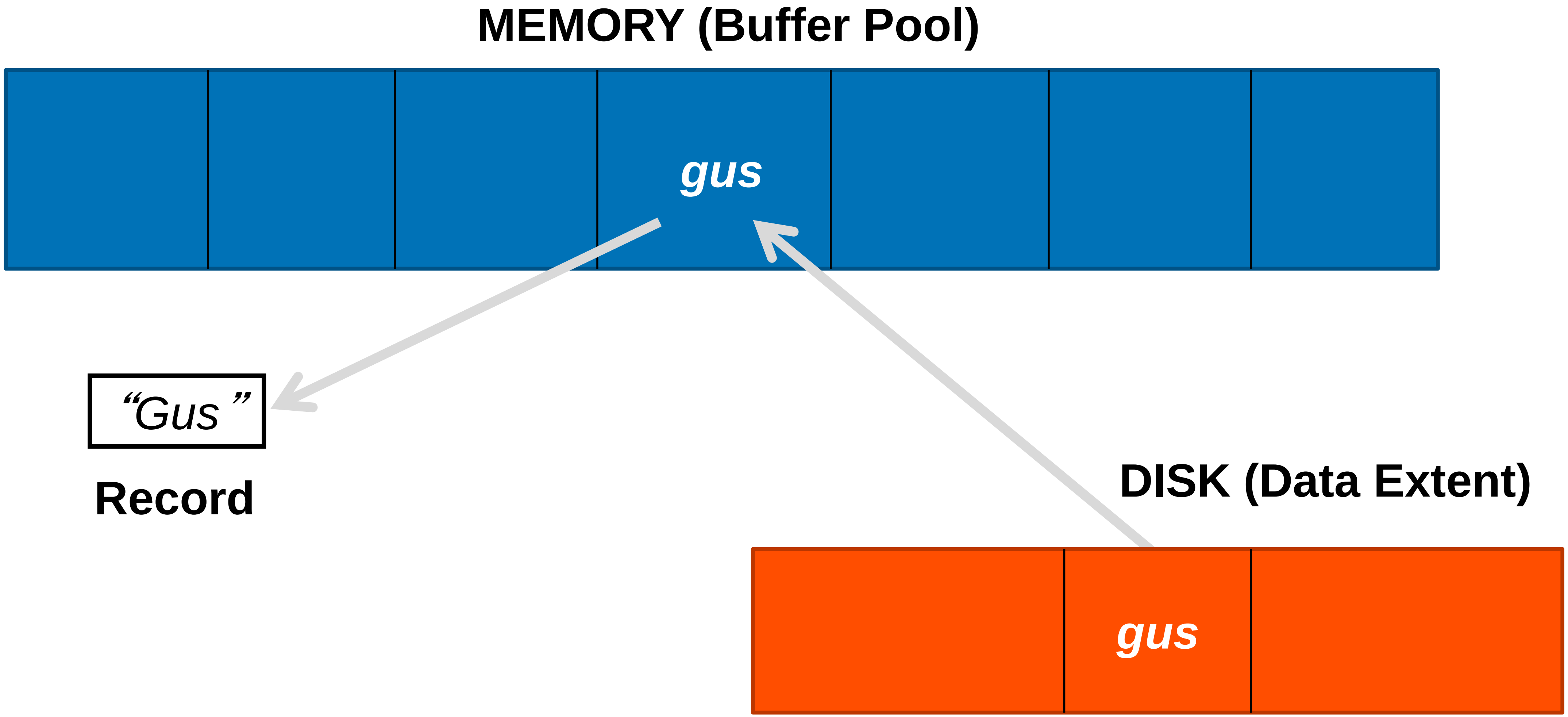
Record 1

Record 2

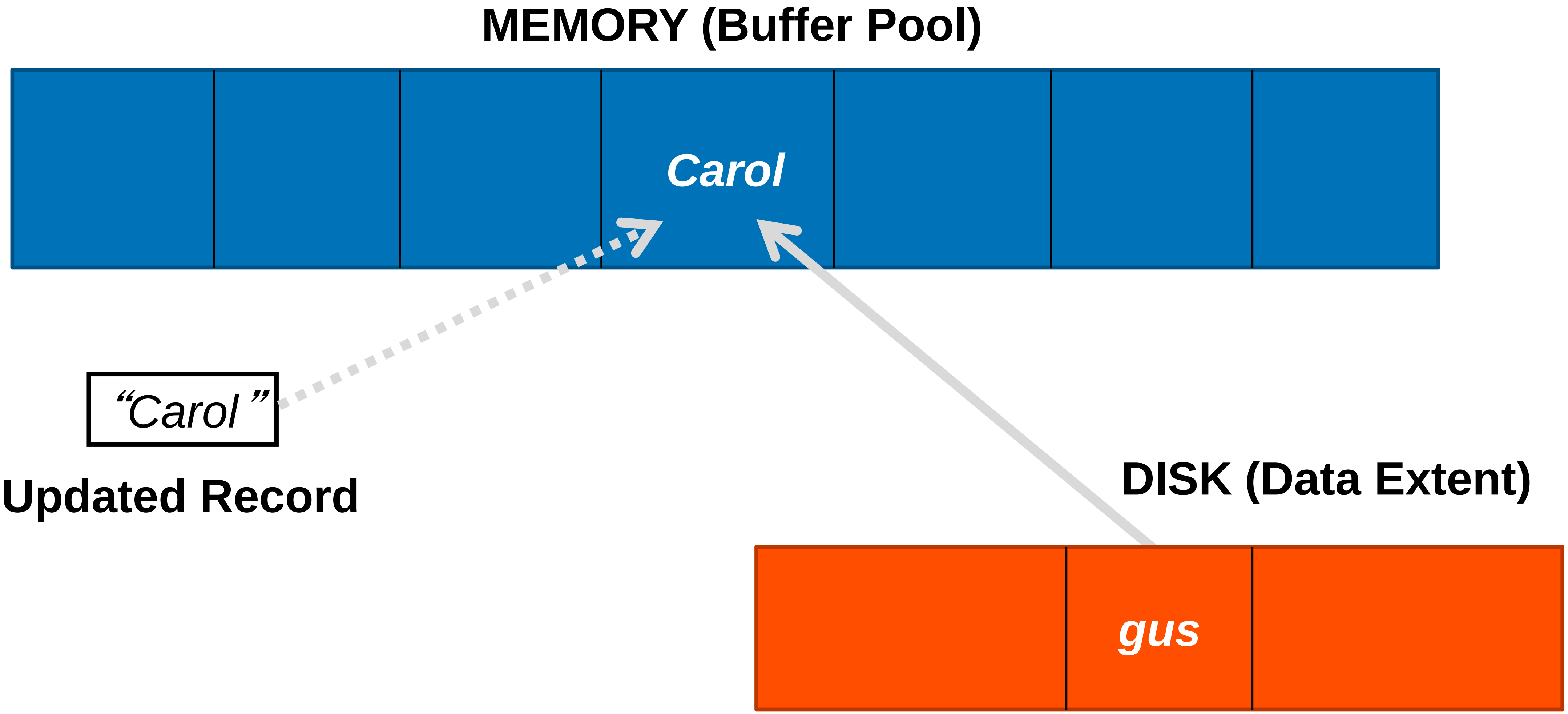
Record 0

Let's Do an Update

Data Block – Before the Update



Data Block – After



But… We Changed Memory Only – Not Disk

- What if someone unplugs server to plug in vacuum cleaner?
- What if we want to undo (roll back)?
- What if we make several more changes and only one block of a fragmented record chain is written to disk to make room in the buffer pool?
- What if an asteroid wipes out all the data centers?

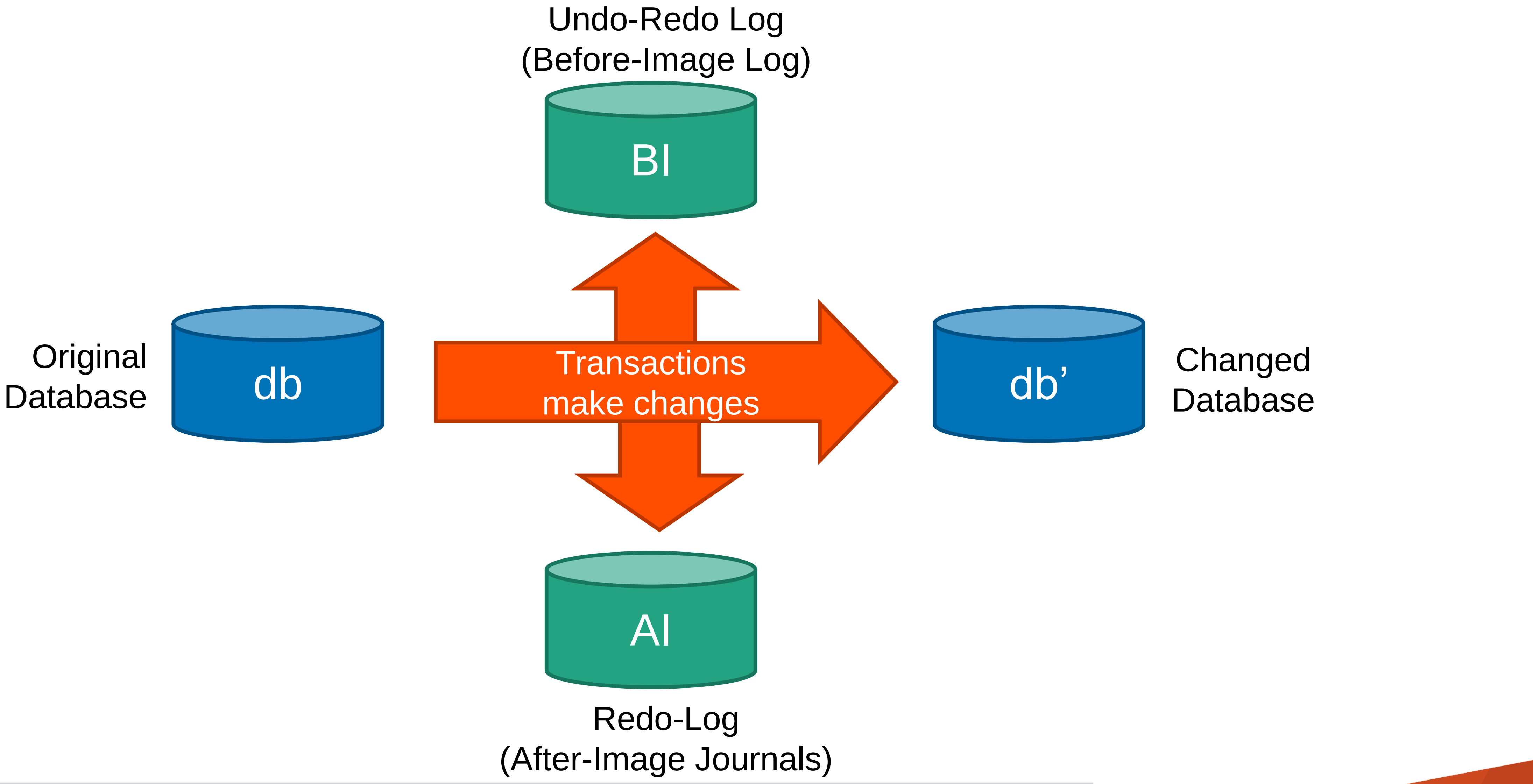
But We Changed Memory Only – No Disk Write

- What if someone unplugs server to plug in vacuum cleaner?
  - The change will be lost
- What if we want to undo (rollback) ?
  - We don’t know the old value or how to undo
- What if we make several more changes and only one block of a fragmented record chain is written to disk to make room in the buffer pool ?
  - The database will be corrupted
- What if an asteroid wipes out all the data centers?
  - The database will disappear completely

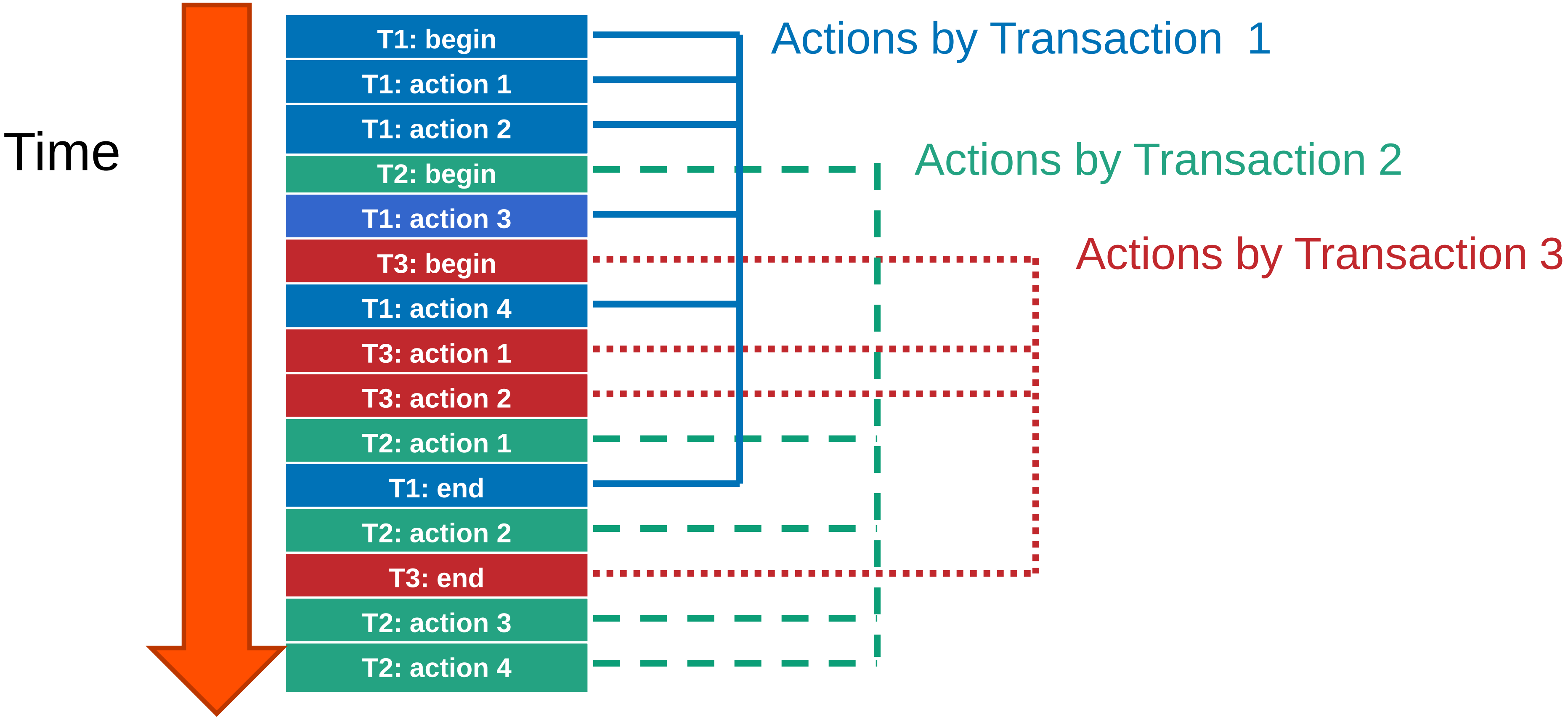
These are all bad things (tm)

Transaction Logging to the Rescue!

Two Transaction Logs



Transaction Log Records (aka "Notes")



Notes form a complete history of everything

Log Records (Notes)

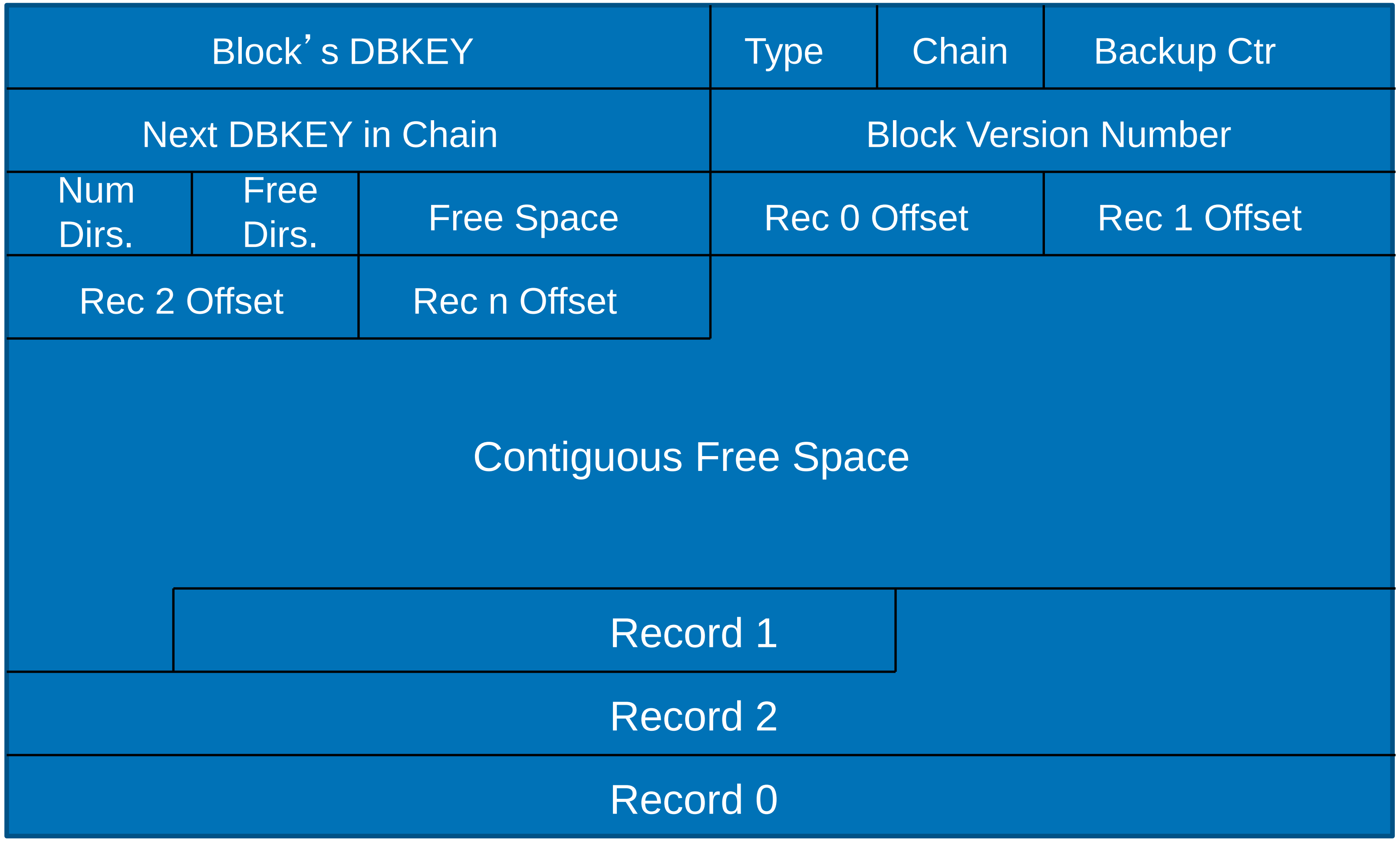
- Generated for every change to database
- Each describes exactly one change to one database block
  - Almost - there are log records that describe changes to purely memory-resident data structures like the transaction table
- Apply only to specific version number of block
- Some operations require more than one change
  - Index splits, multi-block records
- Written in same order changes are executed.
- Notes from concurrent transactions are mixed together

Undo-Redo (BI) Log Records

- Each log record (or “note”) contains:
  - Data area number
  - Database block number (its dbkey)
  - Database block's version number
  - Note type – specifies what operation to perform
  - Any information needed to undo the operation
    - In case we have to roll back
  - Any information needed to redo the operation
    - In case we lose the result before writing to disk

Let's Do an Update, with Notes this Time

A Typical Data Block – for Us to Update



Block’s DBKEY

Type

Chain

Backup Ctr

Next DBKEY in Chain

Block Version Number

Num Dirs.

Free Dirs.

Free Space

Rec 0 Offset

Rec 1 Offset

Rec 2 Offset

Rec n Offset

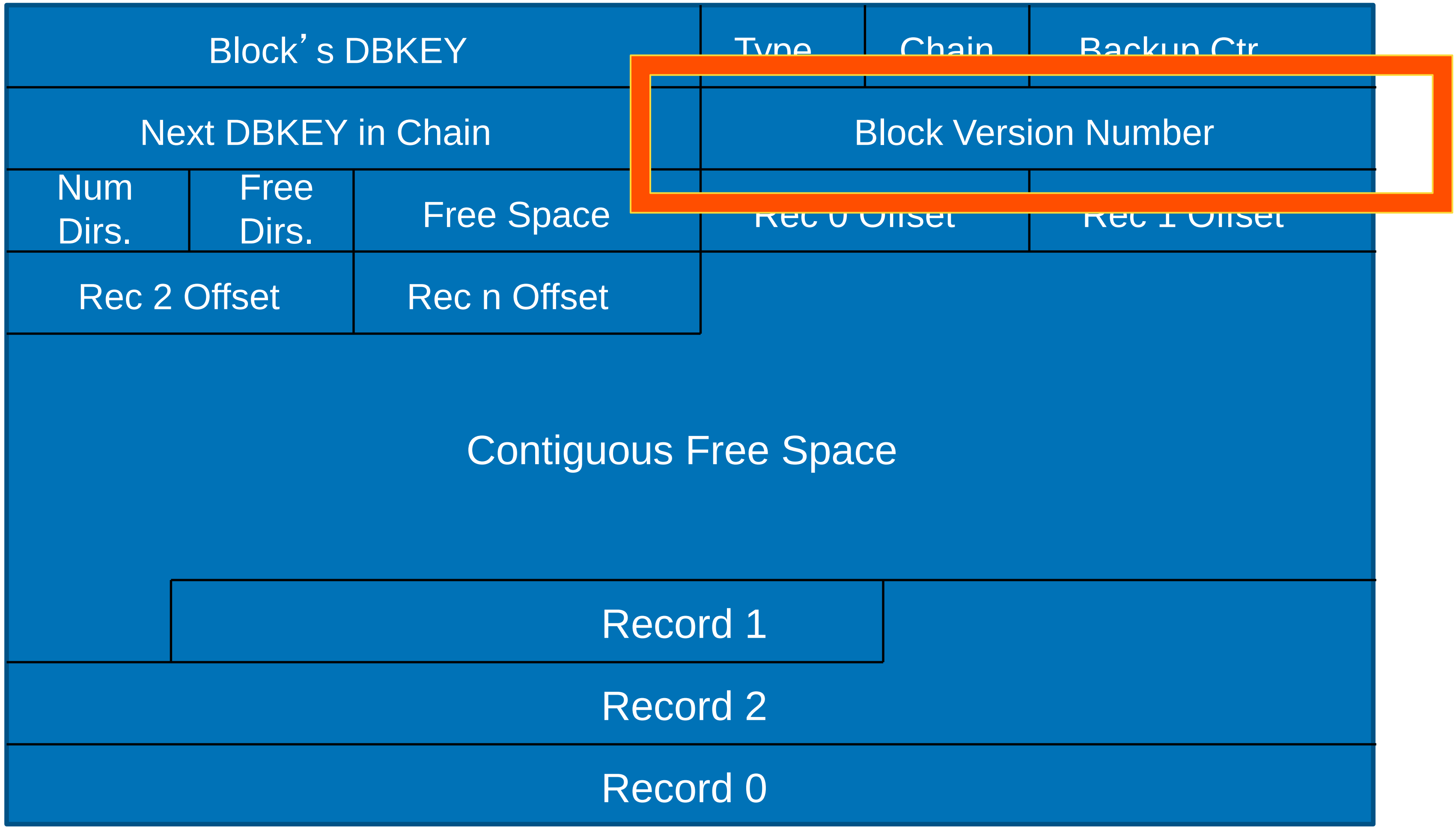
Contiguous Free Space

Record 1

Record 2

Record 0

A Typical Data Block – for Us to Update



Block’s DBKEY

Type

Chain

Backup Ctr

Next DBKEY in Chain

Block Version Number

Num Dirs.

Free Dirs.

Free Space

Rec 0 Offset

Rec 1 Offset

Rec 2 Offset

Rec n Offset

Contiguous Free Space

Record 1

Record 2

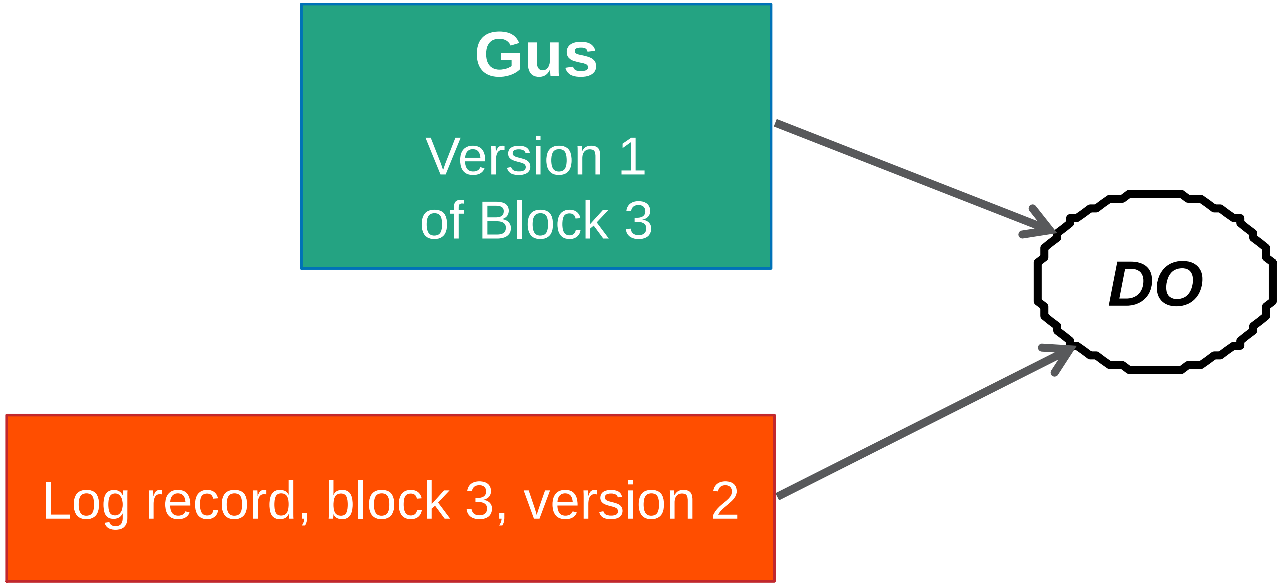
Record 0

Updating a Block – Revisited

Gus

Version 1
of Block 3

Updating a Block – Revisited



Gus

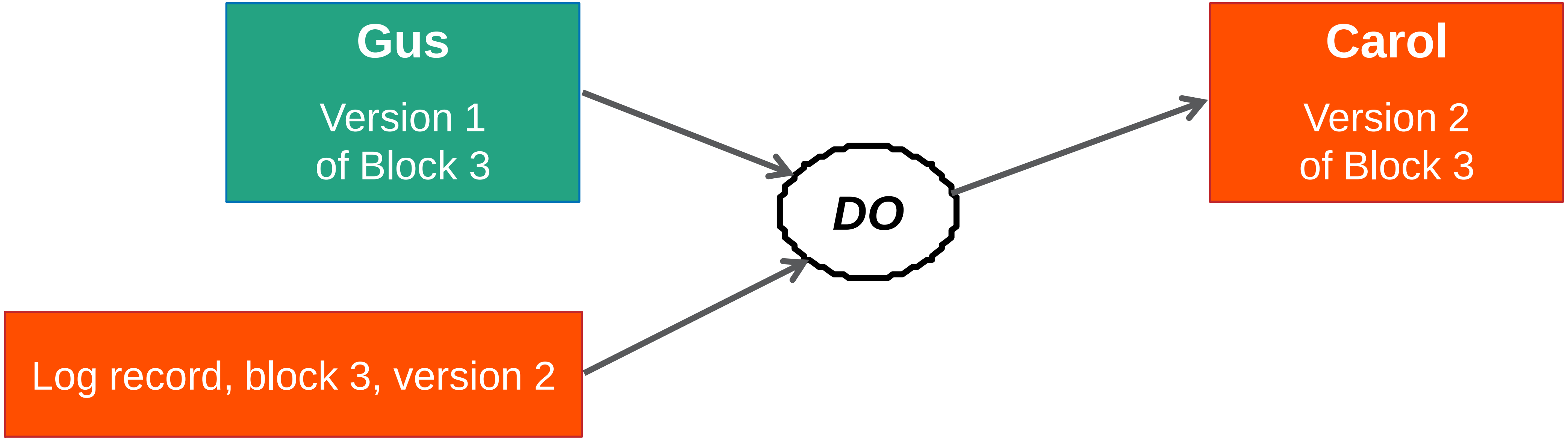
Version 1
of Block 3

Log record, block 3, version 2

DO

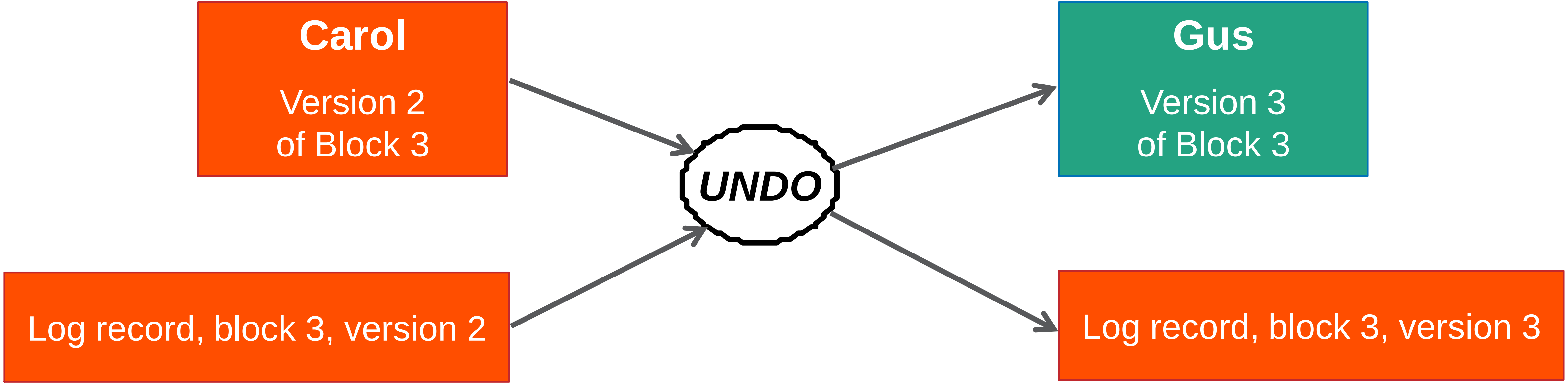
Updating a Block – Revisited

Operation produces
new data values
new version



Updating a Block – Undoing

Operation produces original data values but new version



Carol

Version 2 of Block 3

Log record, block 3, version 2

UNDO

Gus

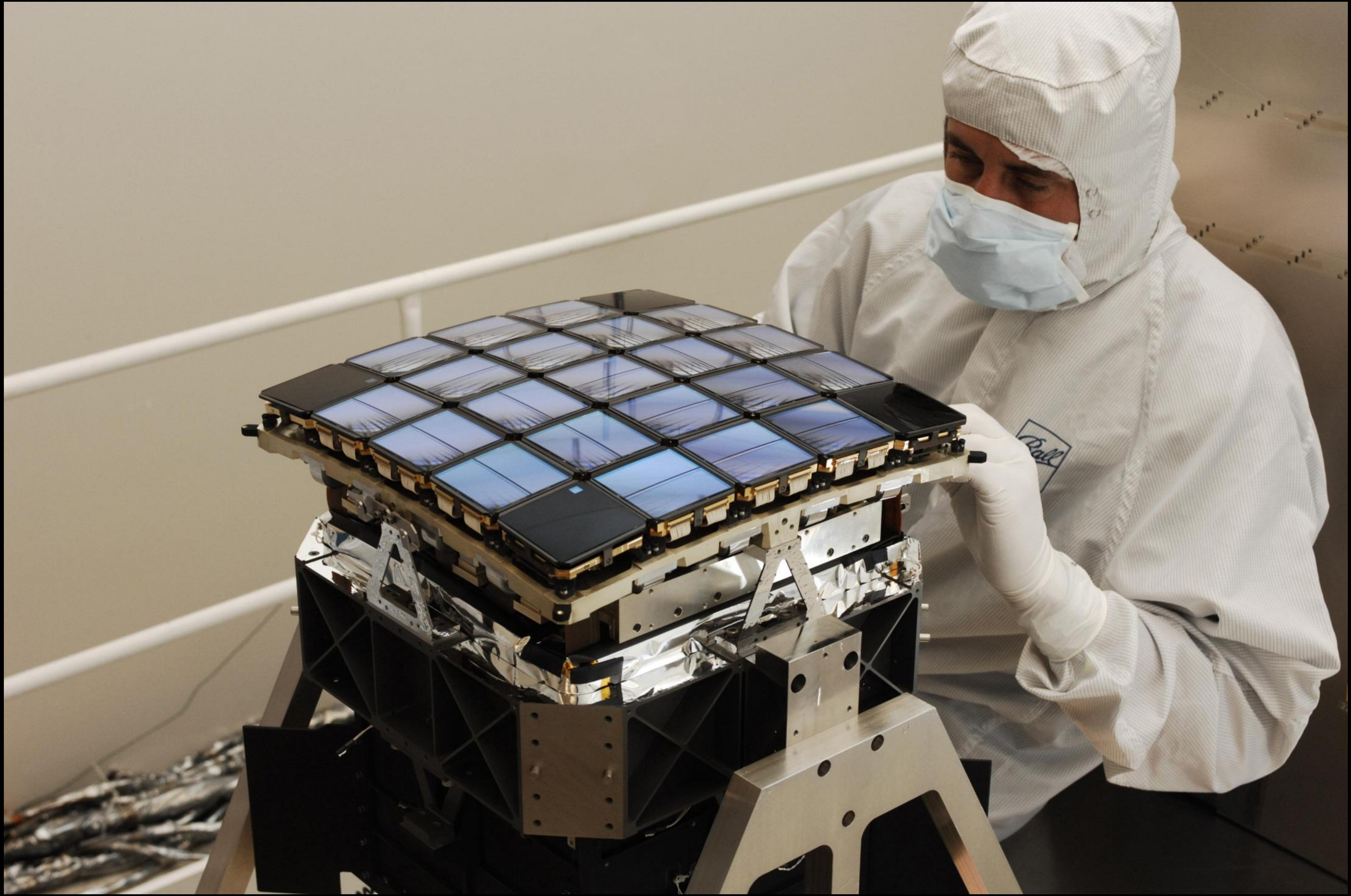
Version 3 of Block 3

Log record, block 3, version 3

Putting things back the way
they were before you touched them

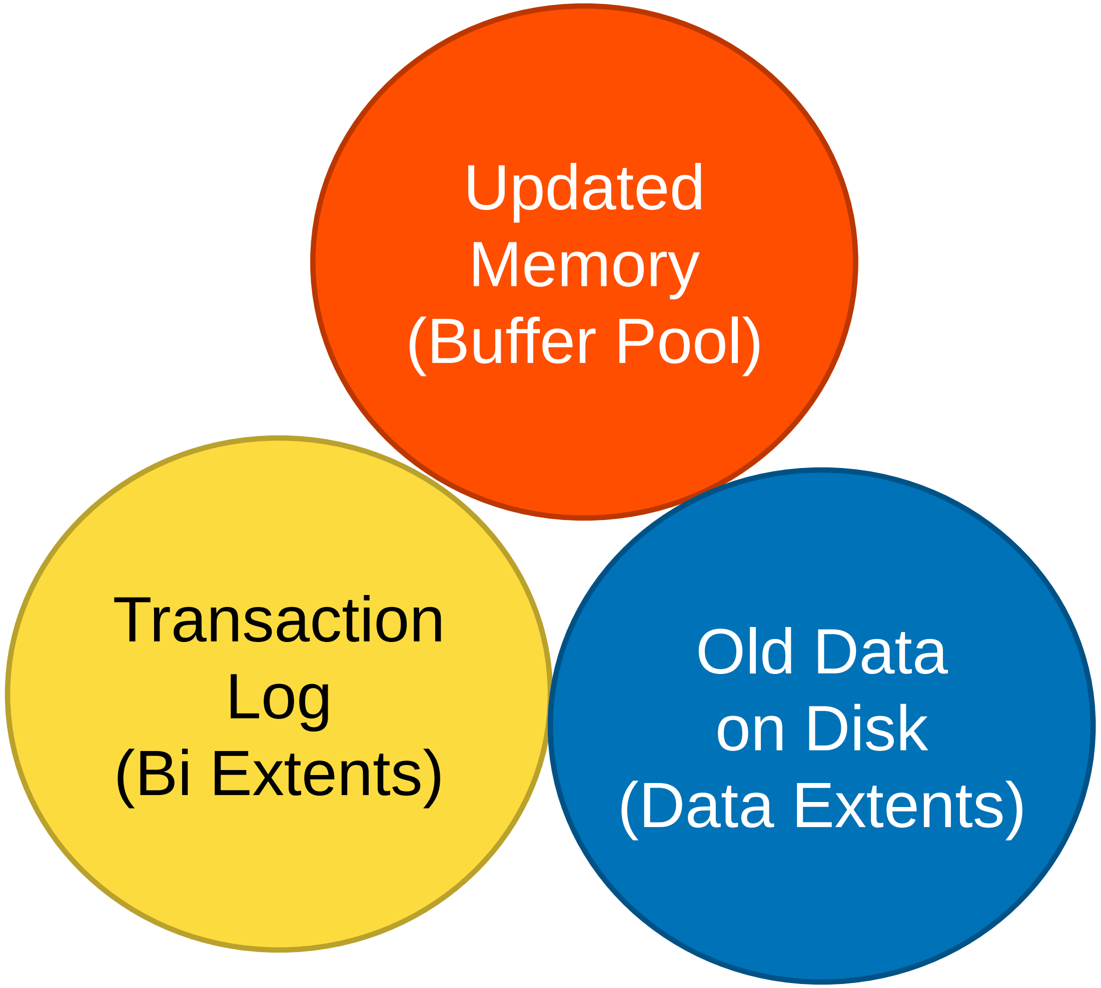
Houston, We Have a Problem!

- Notice that we did the change just in memory
- We are logging the changes, and we can undo if necessary, but
  - How about writing changes to disk?
  - When?
  - What if server unplugged?



The Checkpoint Process

Complete Database State – in 3 Part Harmony



Updated
Memory
(Buffer Pool)

Transaction
Log
(Bi Extents)

Old Data
on Disk
(Data Extents)

Database Checkpoints

- We have memory resident database state (updates are done in memory)
- Must update disk resident data once in a while
- Definition:
  A checkpoint is a process for making what is on disk consistent with the changed or updated database parts that are present only in memory

  It is a process, not an event

Benefits of Checkpointing (1)

- Smaller undo-redo (BI) transaction logs
  - Space can be re-used when the recovery information is no longer needed
- Example:
  - 1,000,000 transactions
  - 350 bytes logged per transaction

Benefits of Checkpointing (1)

- Smaller undo-redo (BI) transaction logs
  - Space can be re-used when the recovery information is no longer needed
- Example:
  - 1,000,000 transactions
  - 350 bytes logged per transaction
  - So:
    - About 350 megabytes of log data
    - Can execute thousand times more transactions a day
    - How much space will that take?
    - Most transactions are larger

Benefits of Checkpointing (1)

- Smaller undo-redo (BI) transaction logs
  - Space can be re-used when the recovery information is no longer needed
- Example:
  - 1,000,000 transactions
  - 350 bytes logged per transaction
  - So:
    - About 350 megabytes of log data
    - Could execute a thousand times more transactions a day
    - How much space will that take? → 350 gigabytes
    - Most transactions are larger

Benefits of Checkpointing (2)

- Shorter Recovery time
  - Fewer changes must be repeated when a crash occurs
- Example:
  - 1,000,000 transactions
  - 3.2 disk io’s per transaction
  - assume disks do about 100 io’s per second
  - Arrival rate of seconds is fixed at 86,400 per day
  - So:

Benefits of Checkpointing (2)

- Shorter Recovery time
  - Few changes must be repeated when a crash occurs
- Example:
  - 1,000,000 transactions
  - 3.2 disk i/o’s per transaction
  - Modern disks do 100 io’s per second
  - Arrival rate of seconds is fixed at 86,400 per day
  - So:
    - 320,000 seconds (3.7 days) to recover
    - What if you had to recover a thousand times more?

Drawbacks of Checkpointing

- Not free!
  - Requires (some) extra processing
  - Requires (some) extra io
  - Takes (some) time
  - Can freeze all database updates for a (short) time

Well worth the costs!

Checkpoint Process

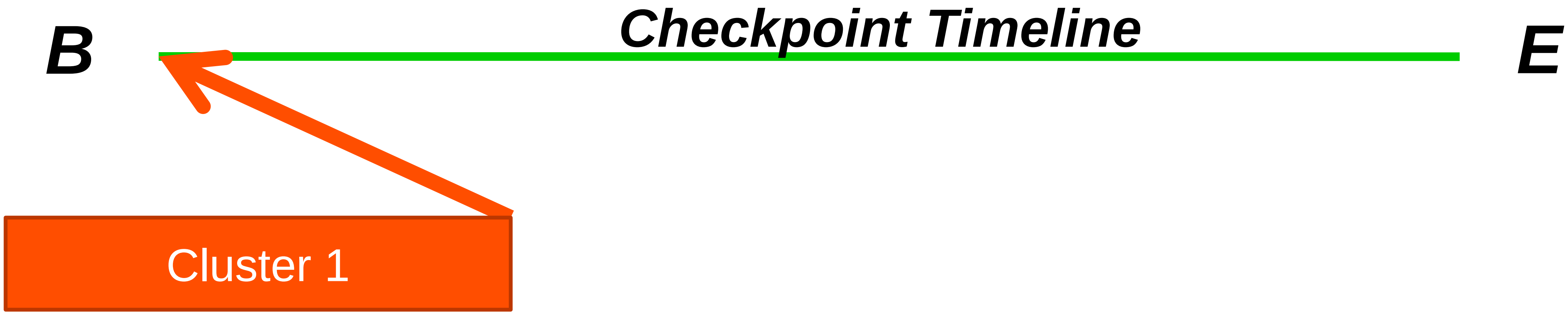
- There are 3 phases to a checkpoint

Checkpoint Process

- There are 3 phases to a checkpoint
  - Beginning
  - Middle
  - And End

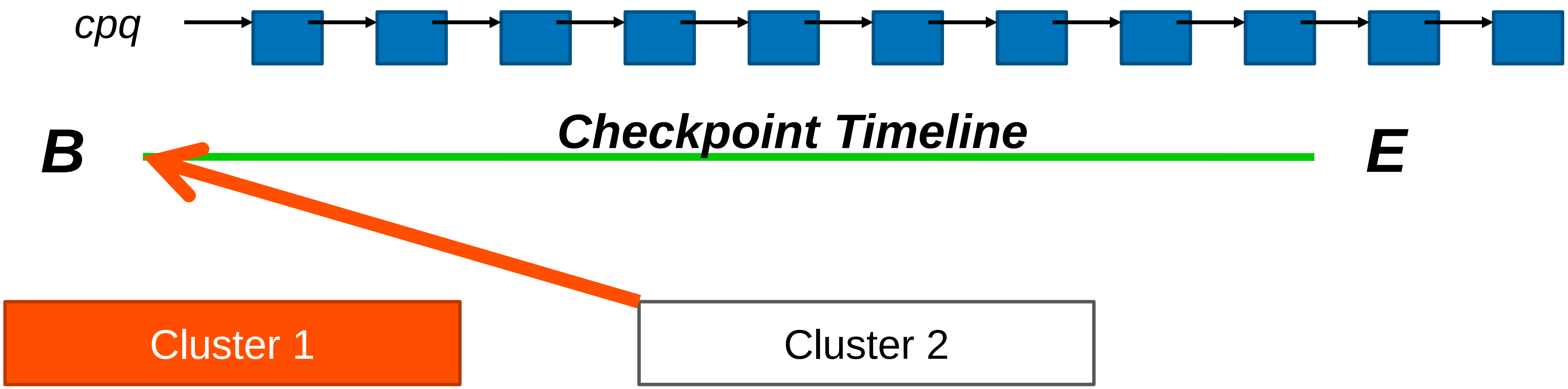
Checkpoint Phase 1 (Begin)

- Unwritten BI and AI buffers forced to disk
- All dirty blocks placed on checkpoint queue
- Next BI cluster opened
  - (May require formatting if new)



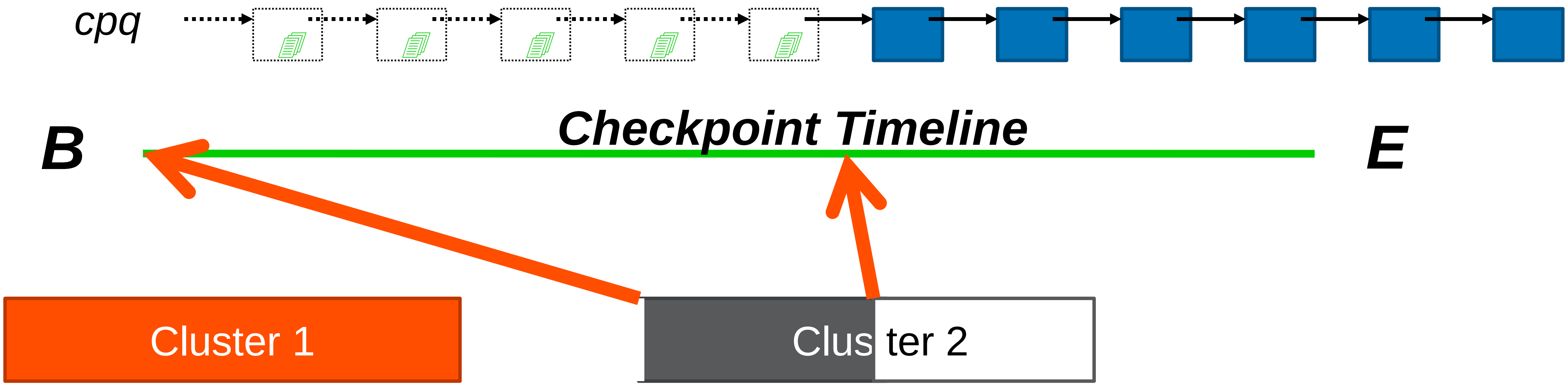
Checkpoint Phase 1 (Begin)

- Unwritten BI and AI buffers forced to disk
- All dirty blocks placed on checkpoint queue
- Next BI cluster opened



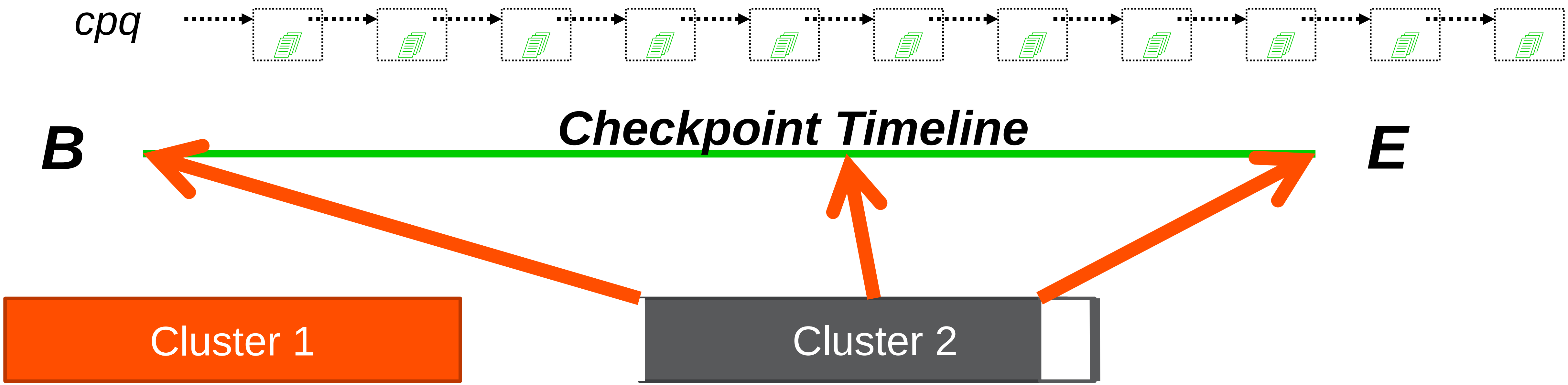
Checkpoint Phase 2 (Middle)

- Asynchronous Page Writers take blocks off the Checkpoint Queue and write them to disk
- APW’s pace themselves



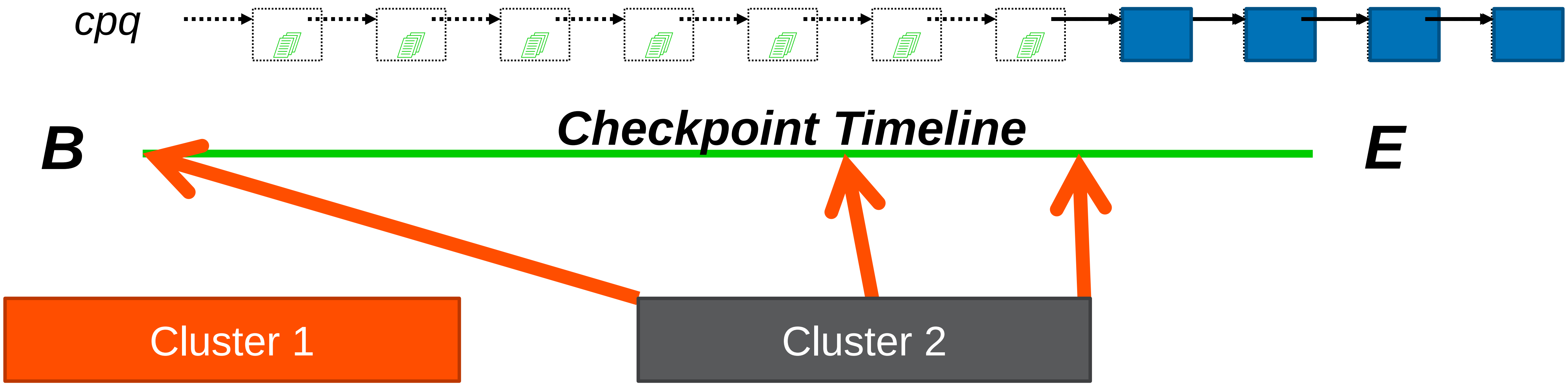
Checkpoint Phase 3 (End)

- As cluster approaches full, all blocks from checkpoint queue have been written to disk
- Checkpoint queue now empty



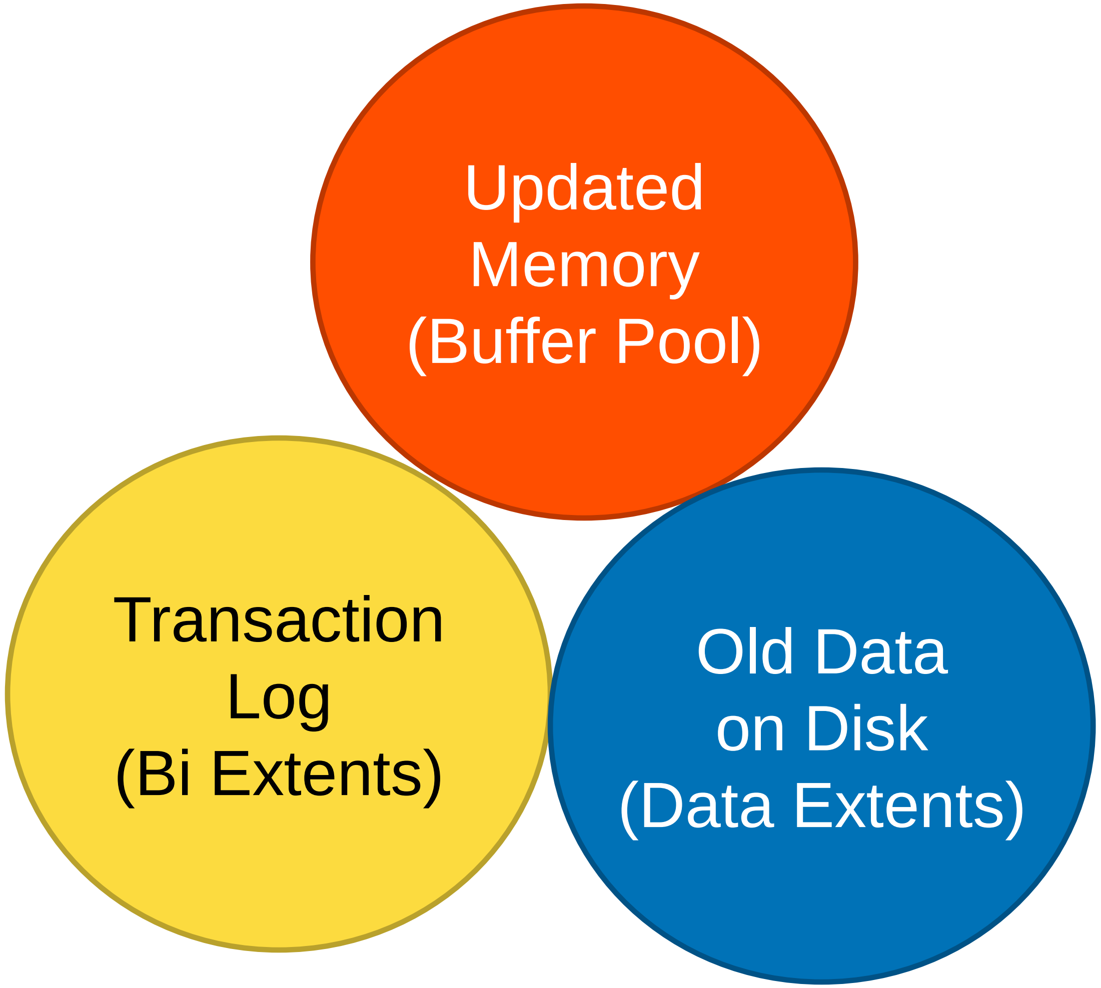
Checkpoint Phase 3 (Alternate Ending)

- Cluster might fill *before* queue emptied
- Now we have to flush remaining blocks
- Delay! AND: fdatasync() calls take more time than normal – more delay



Crash Recovery

Complete Database State – in 3 Part Harmony

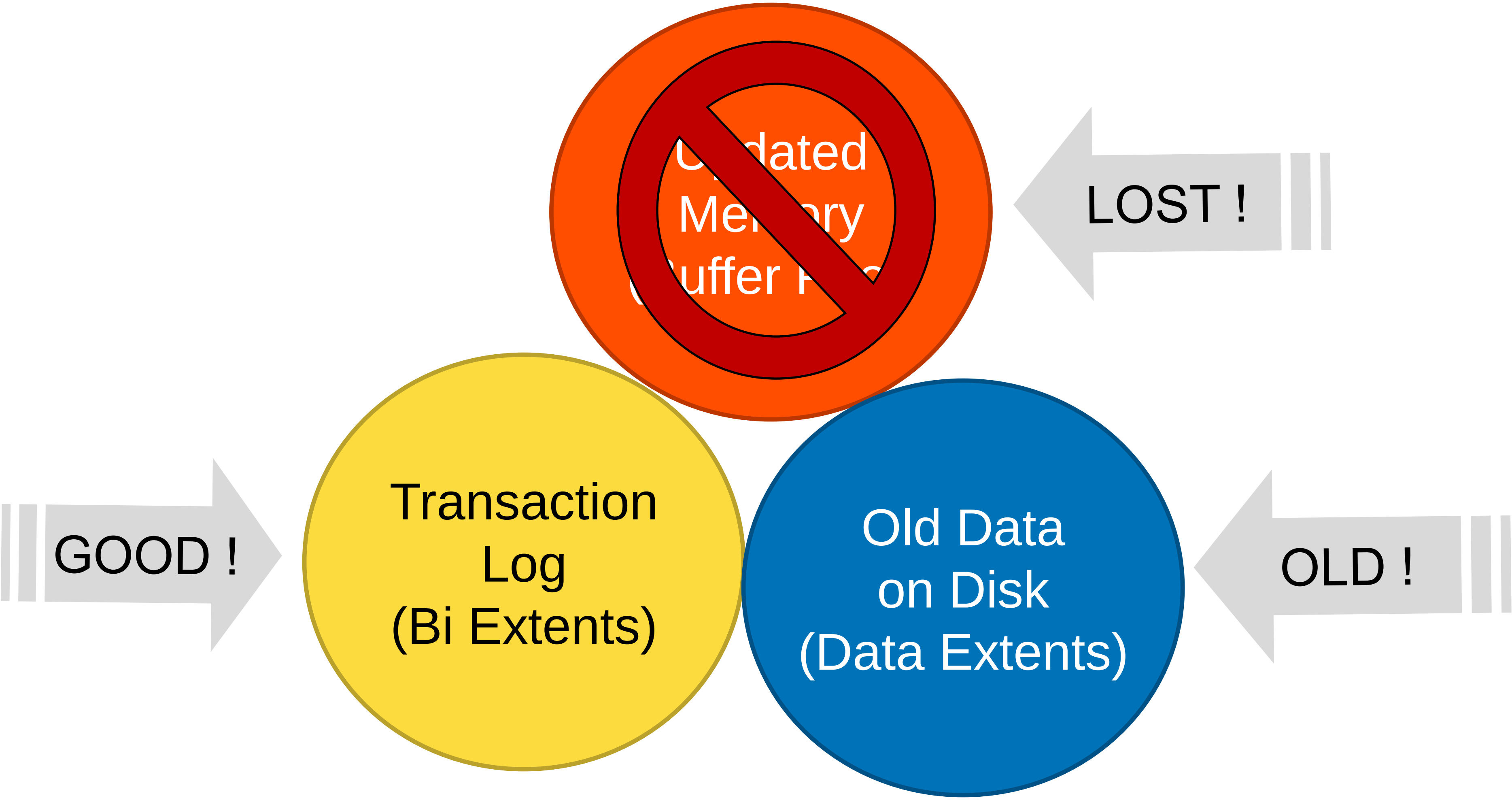


Updated Memory (Buffer Pool)

Transaction Log (Bi Extents)

Old Data on Disk (Data Extents)

Disaster Strikes



U dated
Me ory
(uffer P o

LOST !

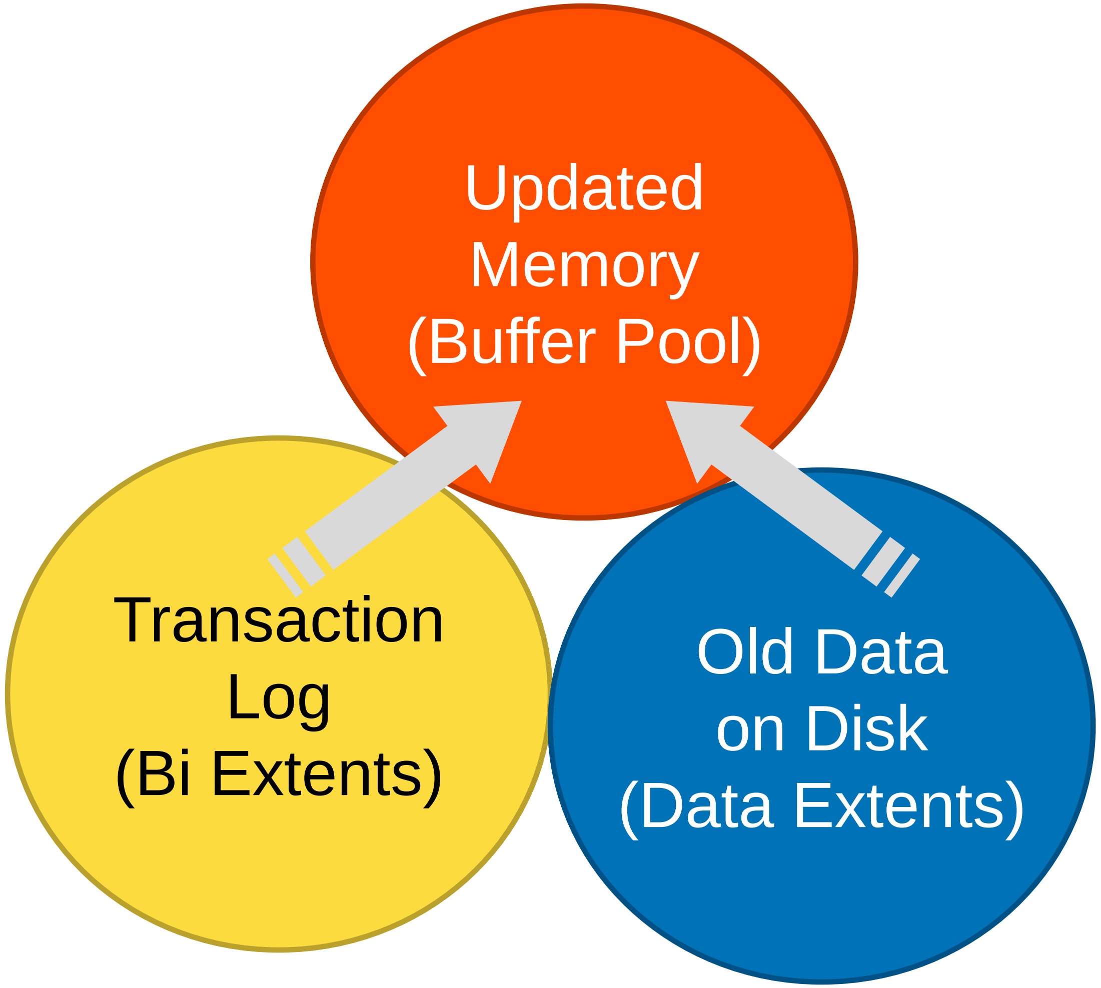
Transaction
Log
(Bi Extents)

GOOD !

Old Data
on Disk
(Data Extents)

OLD !

Reconstructive Surgery

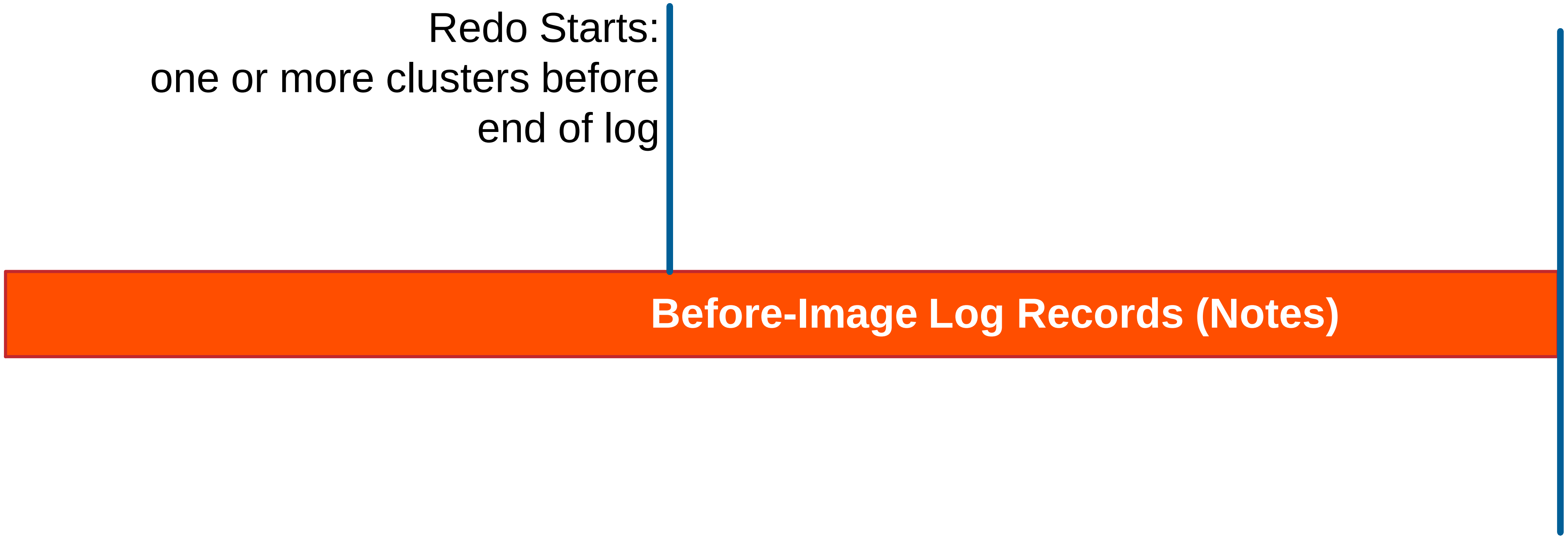


Updated Memory (Buffer Pool)

Transaction Log (Bi Extents)

Old Data on Disk (Data Extents)

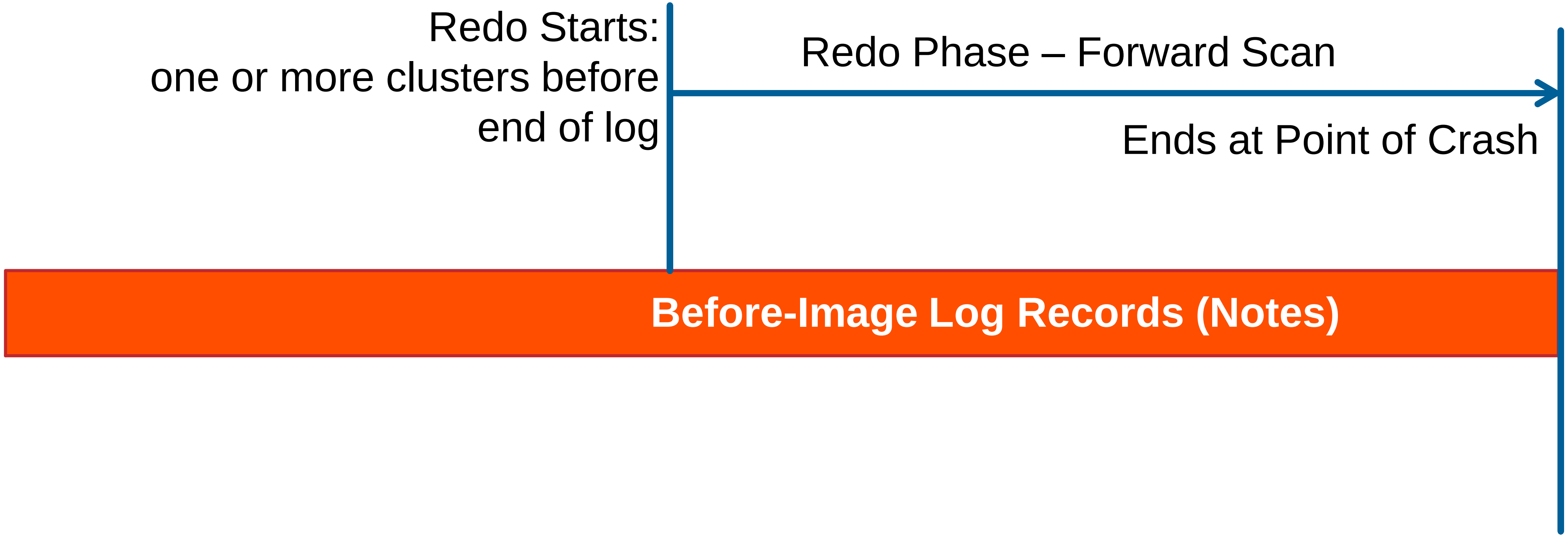
Crash Recovery Processing – Redo Phase



Redo Starts:
one or more clusters before
end of log

Before-Image Log Records (Notes)

Crash Recovery Processing – Redo Phase



Redo Starts:
one or more clusters before
end of log

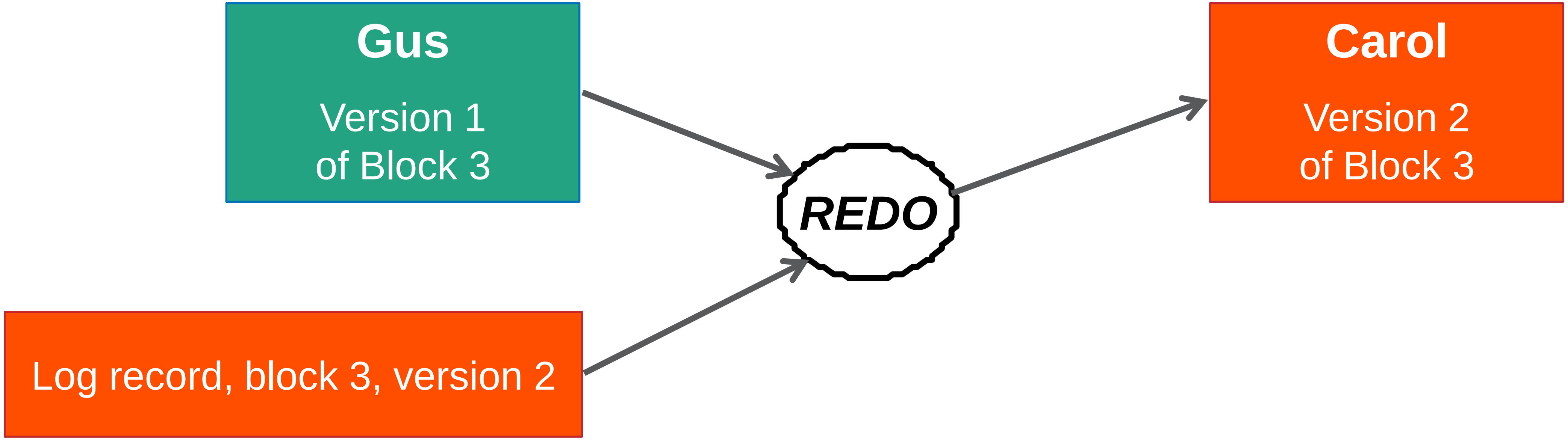
Redo Phase – Forward Scan

Ends at Point of Crash

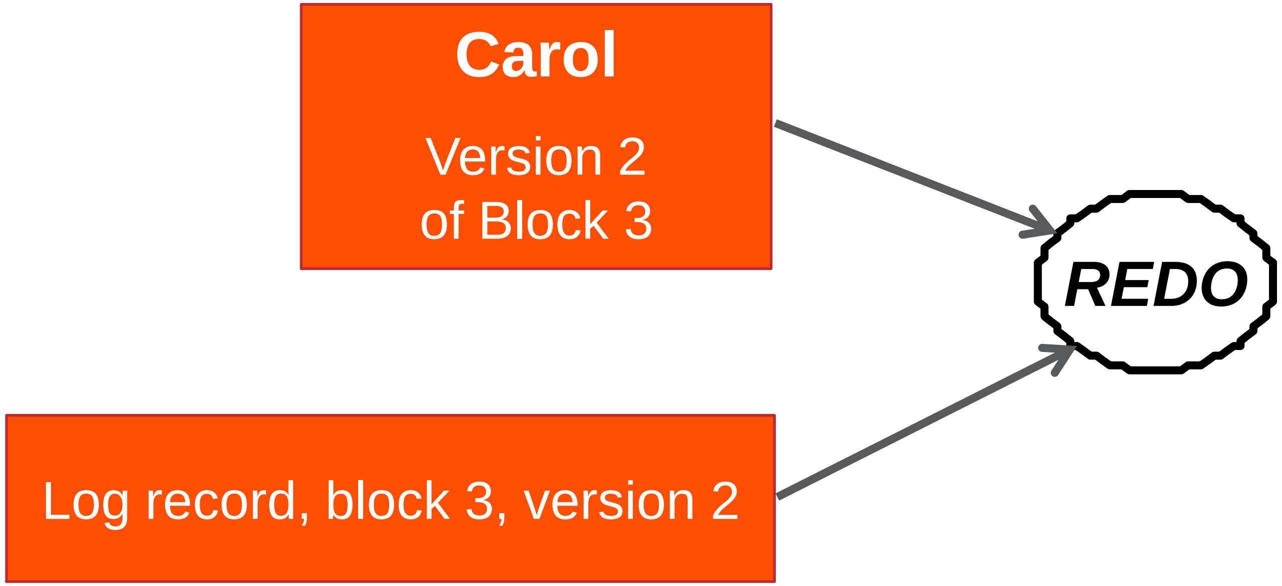
Before-Image Log Records (Notes)

Redo a Change

Redo produces
new data values
new version



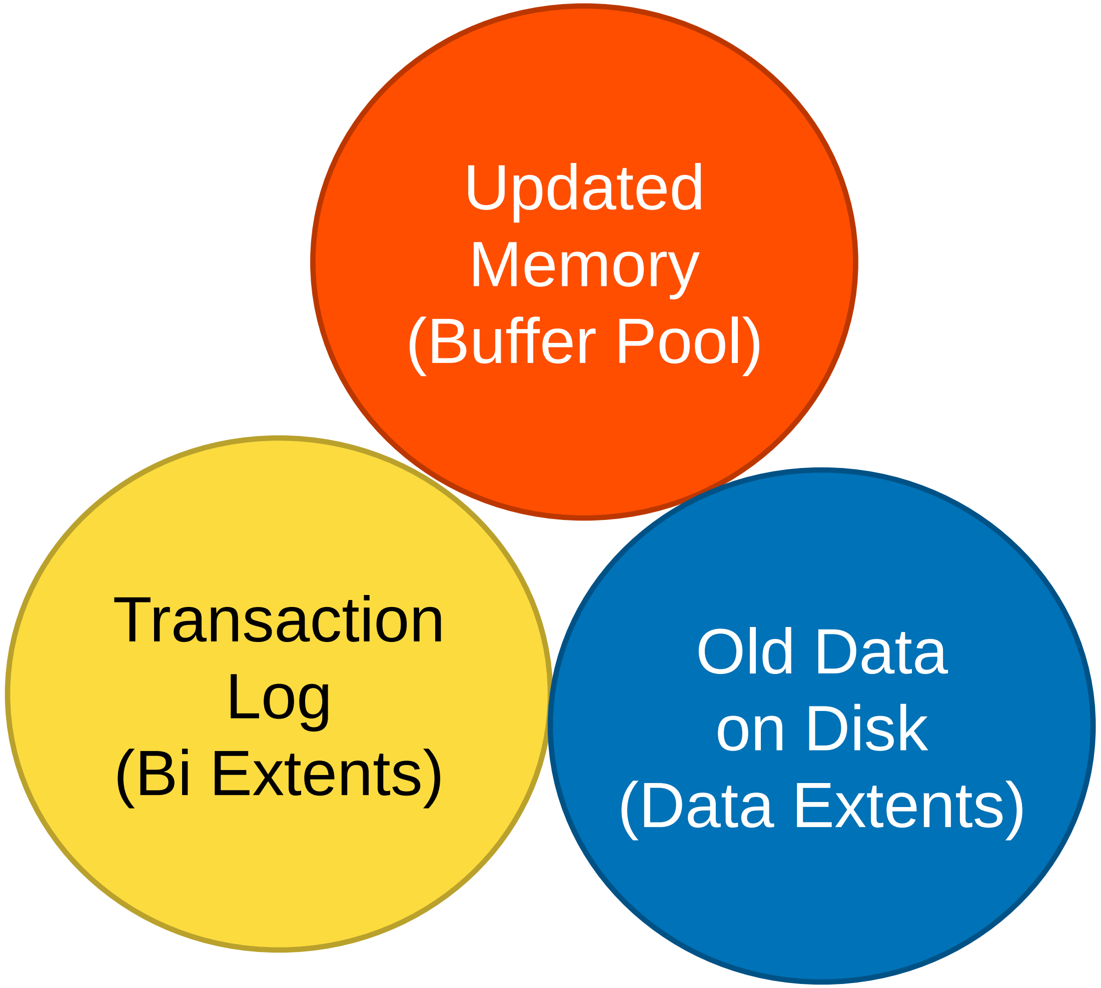
Not Redoing a Change



Nothing to do
We already have
version 2 of the block

Note is skipped

Complete Database State - 3 Parts

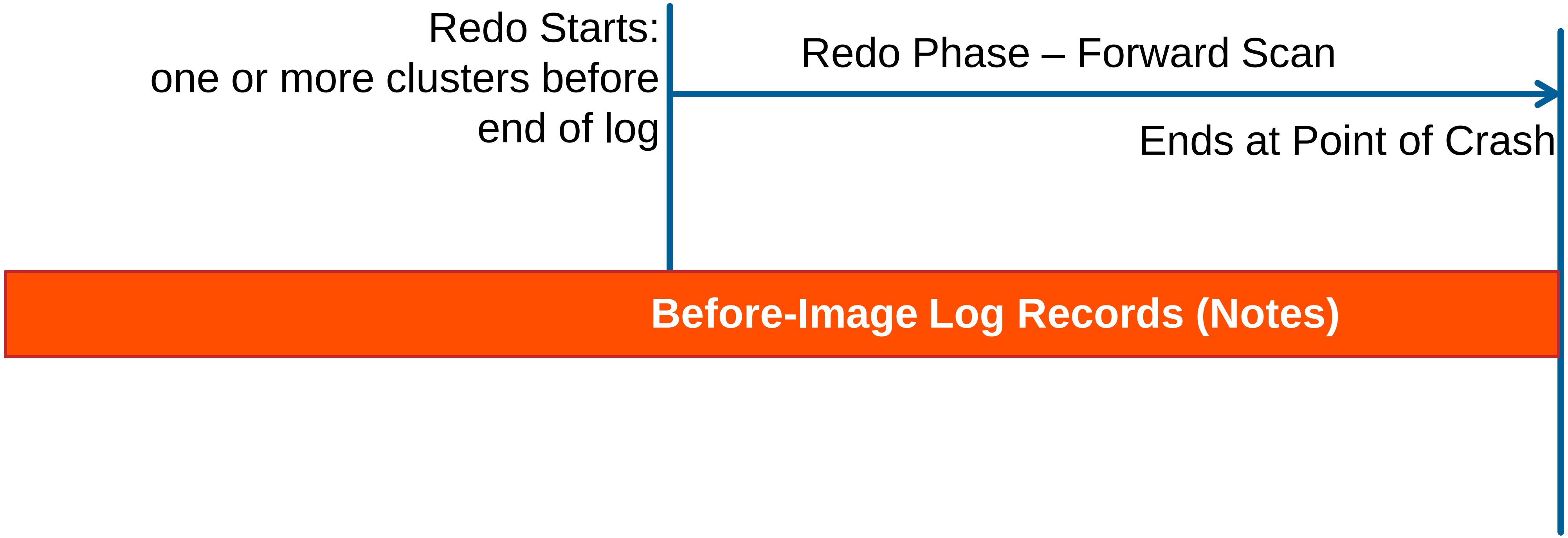


Updated Memory (Buffer Pool)

Transaction Log (Bi Extents)

Old Data on Disk (Data Extents)

Crash Recovery Processing – Redo Phase Completed



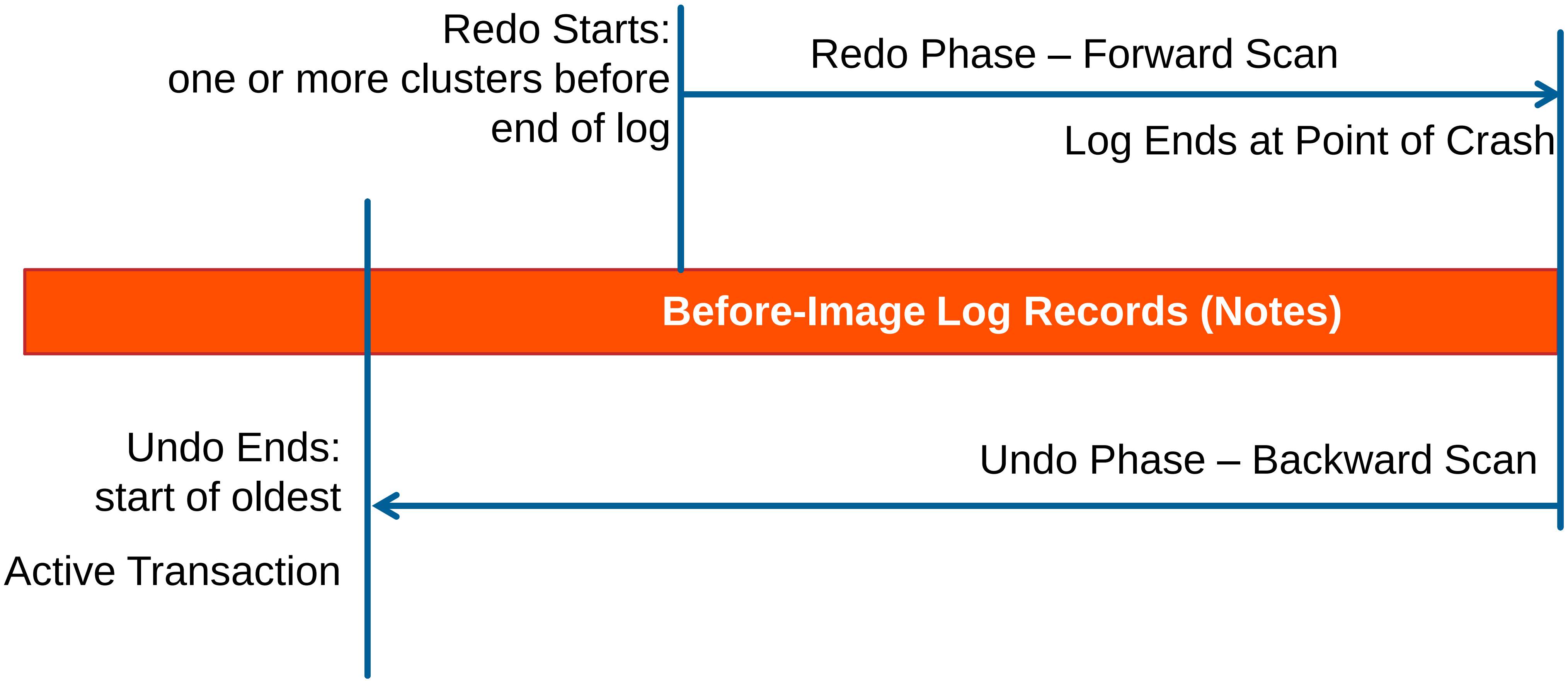
Redo Starts:
one or more clusters before
end of log

Redo Phase – Forward Scan

Ends at Point of Crash

Before-Image Log Records (Notes)

Crash Recovery Processing – Undo Phase



Redo Starts:
one or more clusters before
end of log

Redo Phase – Forward Scan

Log Ends at Point of Crash

Before-Image Log Records (Notes)

Undo Ends:
start of oldest

Active Transaction

Undo Phase – Backward Scan

Now We are Good

Everything is Back the Way it Was Before You Touched it

That’s all we have
time for today, except

Answers

?

email: gus@progress.com

PROGRESS