DB101 – Logging and recovery

Goetz Graefe – Madison, Wis.

“ACID” transactions

⇒ Atomicity: “all or nothing” success or failure

⇒ Consistency: logical & physical, e.g., integrity constraints & indexes

⇒ Isolation: concurrency control “equivalent to serial execution”

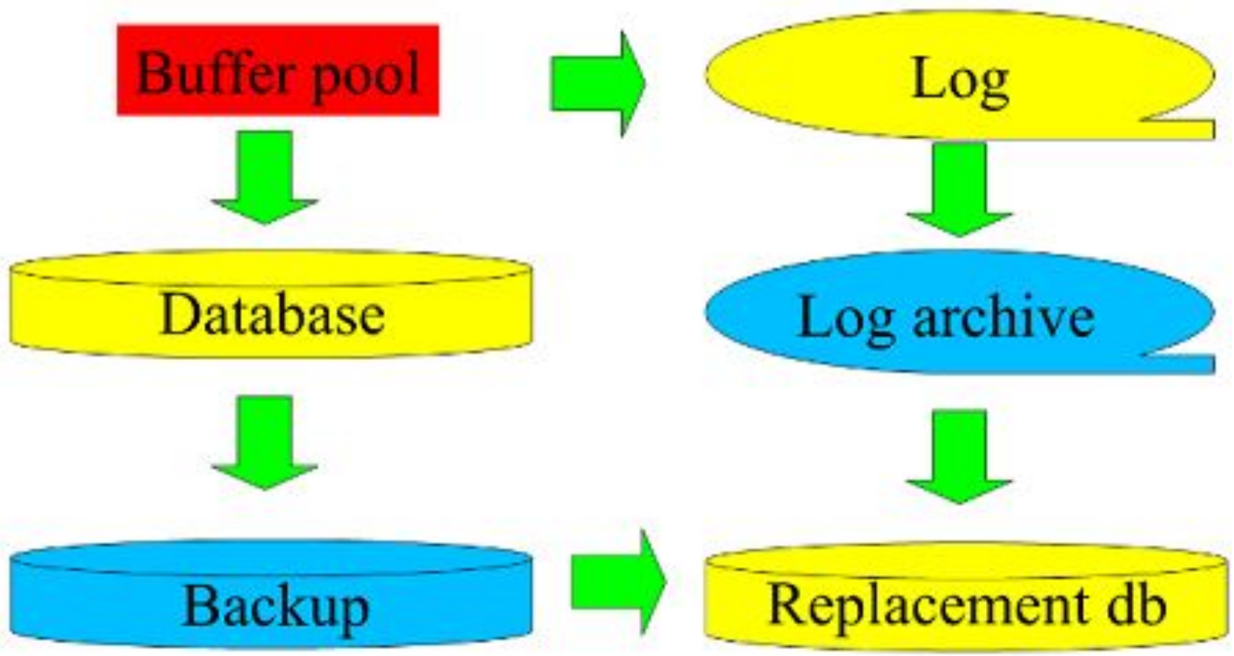
⇒ Durability: persistence “come fire, flood, or insurrection”

Many forms of “cheating”:

- reads and writes at different times
- commit sequence $\ncong$ commit log record sequence
  ⇒ no snapshot isolation transactions, no point-in-time recovery
- …

Update propagation

…must be reliable, simple, efficient, scalable, and robust!



Agenda

- Write-ahead logging
    - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
    - Transaction, system, media failures; double failures; failover
- Single-page failures
    - On-demand local repair: self-repairing indexes
- Instant recovery
    - Instant restart, restore, reboot
    - Single-phase restore, instant backup
    - Instant failover, fail-back
- Conclusions

Omitted topics

- Comprehensive consistency checks of b-trees, tables, databases, backups, recovery logs, log archives
- Recovery from loss of recovery log or log archive
- Allocation-only logging
- Logging and recovery of offline and online index operations (creation)
- Logging in two-phase commit, 3pc, Paxos, Raft, etc.
- Compression of recovery log and log archive
- Transaction rollback and 2pc in multi-version indexes
- Microsoft “accelerated database recovery”
- Instant recovery for file systems and database blobs

Agenda

- **Write-ahead logging**
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Write-ahead logging

Recovery log = recorded history of database updates on "stable storage"

No revising history ⇒ append-only log storage

- Ever-increasing LSNs (log sequence number = address of log record)
- Sequential writing – no gaps in history on stable storage

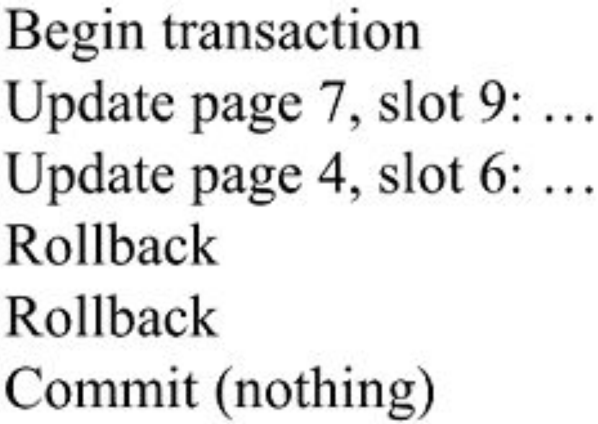
<u>Database history</u>: create, alter, drop, insert, update, delete…

<u>Server history</u>: begin transaction, commit transaction, checkpoint…

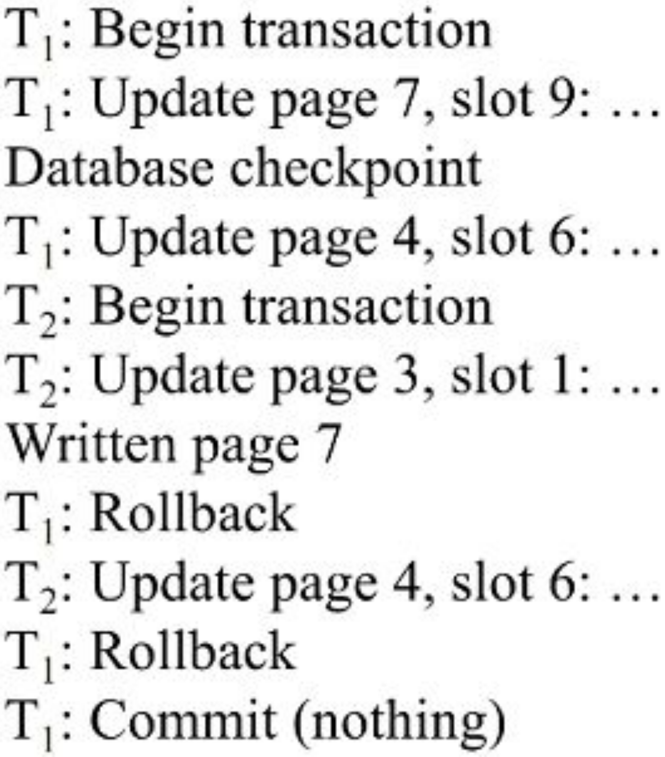
Transaction commit: server history ⇒ database history

Example recovery logs

Transactional database changes
+ server events



Begin transaction
Update page 7, slot 9: …
Update page 4, slot 6: …
Rollback
Rollback
Commit (nothing)



T_1: Begin transaction
T_1: Update page 7, slot 9: …
Database checkpoint
T_1: Update page 4, slot 6: …
T_2: Begin transaction
T_2: Update page 3, slot 1: …
Written page 7
T_1: Rollback
T_2: Update page 4, slot 6: …
T_1: Rollback
T_1: Commit (nothing)

Physical & logical logging

1970s: physical logging = before & after page images

1980s: logical logging: SQL commands & commit sequence

1990s: “physiological” logging: physical “redo”, logical “undo”

- required for row-level locking, e.g., key-value locking in b-trees
- page id + slot #, not byte offset within pages

2000s: logical logging + serial execution in VoltDB

2010s: mirror-don’t-log, append-only data stores, log-only storage…
(is a mirror a page log?)

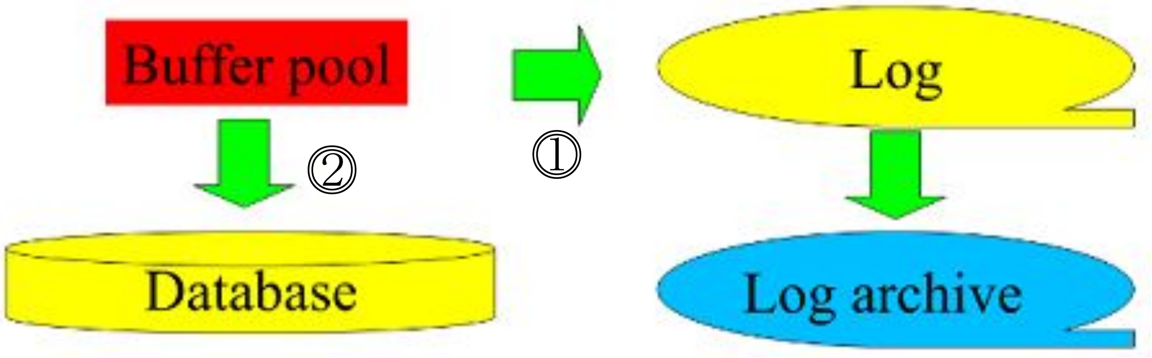
Write-ahead logging

No update-in-place until the “undo” log record is on stable storage

“Redo-undo” log record: new & old values

Allocation-only logging

“Redo-only” log records: rollback actions, system transactions



User transactions & system transactions

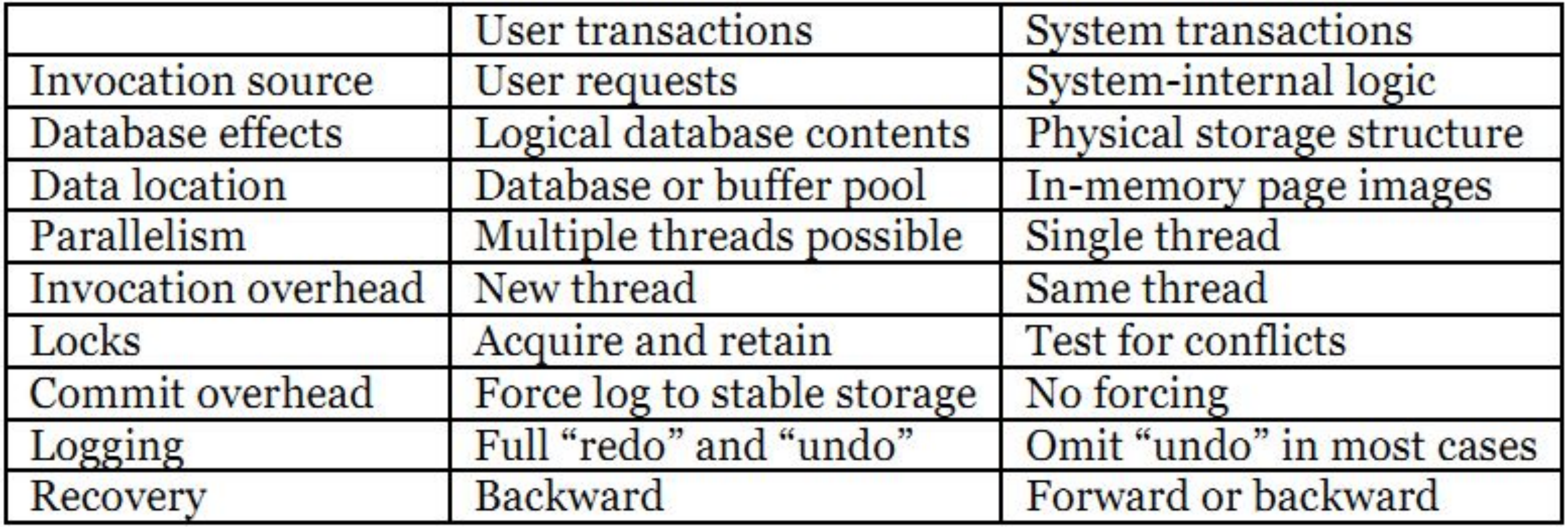
User transactions: retrieve & update logical database contents

- Updates observable with “select count(*) from… where…”

System transactions: modify physical database representation

- E.g., b-tree node split, ghost removal
- No locks on database contents, only latches on data structures
- No log flush on commit, single log record ⇒ no “undo”
- Invoked within thread, no transaction descriptor or identifier
- *All space management actions*!

User transactions & system transactions



	User transactions	System transactions
Invocation source	User requests	System-internal logic
Database effects	Logical database contents	Physical storage structure
Data location	Database or buffer pool	In-memory page images
Parallelism	Multiple threads possible	Single thread
Invocation overhead	New thread	Same thread
Locks	Acquire and retain	Test for conflicts
Commit overhead	Force log to stable storage	No forcing
Logging	Full “redo” and “undo”	Omit “undo” in most cases
Recovery	Backward	Forward or backward

Ghost records, ghost space, ghost keys…

Transaction 1:

1. Begin transaction
2. Delete from Table1 where …
3. …
4. Abort transaction

Transaction 2:

1. Begin transaction
2. …
3. Insert into Table1 values (…)
4. …

- Space management (free bytes)
- Integrity constraints (uniqueness)

Ghost indexes…

Transaction 1:

1. Begin transaction
2. Drop index Orders.CustIdx
3. …
4. Abort transaction

Transaction 2:

1. Begin transaction
2. …
3. Create index PartIdx on LineItem (PartNo…)
4. …

- Space management (free storage)
- Logging volume (rollback “drop”)

Agenda

- Write-ahead logging
  - **Commit**, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Transaction commit

Single time & place for commit decision

- e.g., user commits herself, then system attempts to commit
- updates become durable and then visible
- force commit log record to recovery log on “stable storage”

  transaction commit: server history ⇒ database history
- “Force” policy (i.e., flush dirty pages to database) not required!

Latency vs bandwidth – “group commit”

Small transaction ⇒ ≤ 1 KB of log records

Fast device ⇒ 0.1 ms × 200 MB/s = 20 KB page size
(latency × bandwidth)

Log flush per commit ⇒ >90% of log space & bandwidth wasted!

⇒ group commit = short timeout before log flush, e.g., 1 ms

Two-phase commit

Single time & place for commit decision

- Multiple logs, single commit decision
- Participants “vote”: pledge to realize coordinator’s commit decision
- Pledge must survive lock contention & deadlock, server restart, etc.
- Coordinator decides & logs its decision
- Participants receive & log the decision

Paxos vs two-phase commit – different problems!

Paxos

- Majority agreements on coordinator + outcome
- Orthogonal to concurrency control
- Maintenance of replicas & mirrors

Two-phase commit

- Unanimous pledge, single decision
- Implications for concurrency control (locking)
- Maintenance of shards & partitions

Agenda

- Write-ahead logging
  - Commit, **backups**, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Classification of database backups ~~server backups~~

Contents & frequencies

- Full: all allocated database pages, e.g., every Sunday 2am
- Differential: pages changed since full backup, e.g., since Sunday
- Incremental: pages changed since last backup, e.g., since yesterday

Consistency

- Transaction-consistent: quiescent database server
- Snapshot (point-in-time): transaction-consistent or not
- Fuzzy: database pages as found in buffer & disk-order scan…

Compressed, encrypted, sorted on database page identifier, indexed…

Agenda

- Write-ahead logging
  - Commit, backups, **archives**, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

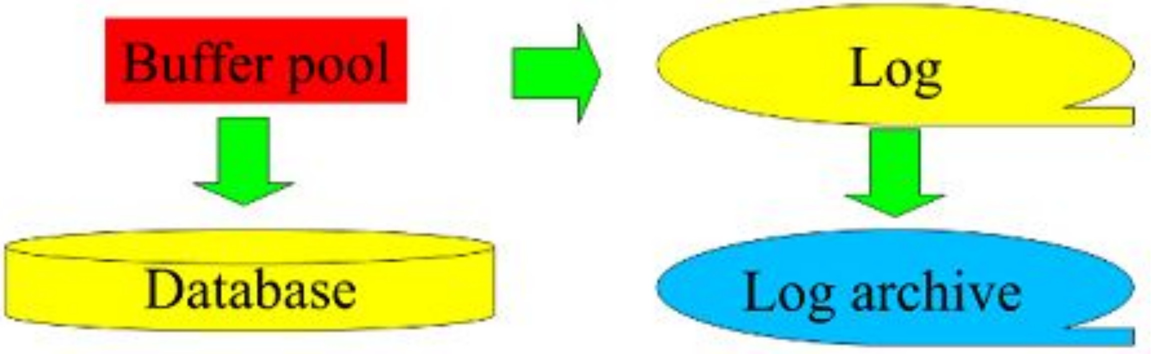
Recovery log vs log archive

Recovery log – written during transaction processing

- Low-latency reliable storage, e.g., non-volatile memory or SSDs

Log archive – written asynchronously

- Cheap but very reliable storage, e.g., RAID-5 or RAID-6



Recovery log vs log archive

Recovery log – written during transaction processing

- Low-latency reliable storage, e.g., SSDs or non-volatile memory
  - Chosen for latency, not for capacity or price/capacity
  - No compression, encryption, sorting, indexing…
- Sequence of log records strictly by time

Log archive – written asynchronously

- Chosen for cheap reliable storage, e.g., RAID-5 or RAID-6
  - Usually compressed & encrypted
- Could be sorted & indexed, e.g., on database page identifier

Agenda

- Write-ahead logging
  - Commit, backups, archives, **checkpoints**
- Classic failures & recovery techniques
  - Transaction, system, media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

~~Database checkpoints~~ Server checkpoints

Checkpoint = log record capturing current server state

- Active transactions & their locks
- Dirty pages & their oldest change

1970s checkpoint: quiescent transactions, flushing all dirty pages

1990s checkpoint: flush dirty pages, “second chance”

2010s checkpoint: multiple log records, partitioned checkpoints

Agenda

- Write-ahead logging
    - Commit, backups, archives, checkpoints
- **Classic failures & recovery techniques**
    - Transaction, system, media failures; double failures; failover
- Single-page failures
    - On-demand local repair: self-repairing indexes
- Instant recovery
    - Instant restart, restore, reboot
    - Single-phase restore, instant backup
    - Instant failover, fail-back
- Conclusions

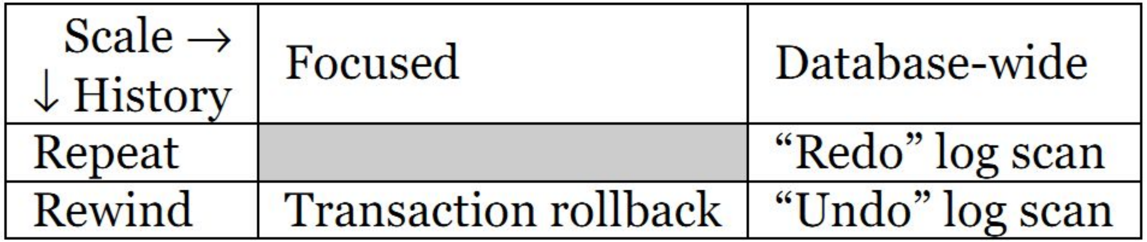
Classic failures & recovery techniques

Classification of all failures into transaction, system, and media failures

- Missing: double failures, node failures, log failures…

Precisely three recovery procedures

- Transaction rollback
- System restart
- Media restore



Scale → ↓ History	Focused	Database-wide
Repeat		"Redo" log scan
Rewind	Transaction rollback	"Undo" log scan

Agenda

- Write-ahead logging
    - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
    - **Transaction**, system, media **failures**; double failures; failover
- Single-page failures
    - On-demand local repair: self-repairing indexes
- Instant recovery
    - Instant restart, restore, reboot
    - Single-phase restore, instant backup
    - Instant failover, fail-back
- Conclusions

Transaction failure ⇒ transaction rollback

Any cause: deadlock, timeout, user cancel, global abort decision…

Linked list of log records per transaction:

- LastLSN in transaction descriptor (in-memory transaction manager)
- PriorLSNsameTx in each log record

1970s: idempotent "undo" – no log of rollback

1990s: "compensation log records" in ARIES

2010s: "rollback log records"

Transaction failure ⇒ transaction rollback

1970s: idempotent “undo” – no log of rollback ⇐ restart with full log

- “Abort” log record, no end-of-transaction log flush

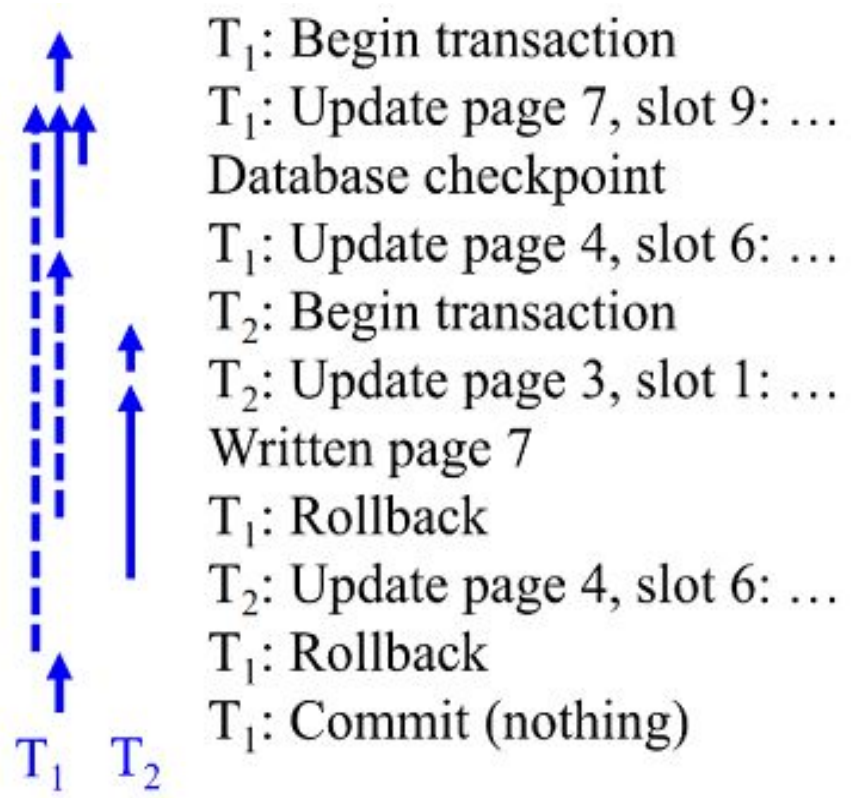
1990s: “compensation log records” in ARIES ⇐ row-level locking

- “Redo” only
- “Update back” vs restore page image
- “Commit nothing”, no end-of-transaction log flush
- Incremental lock release (during rollback)

2010s: “rollback log records”

- No data contents, no linked list per transaction

Per-transaction chains of log records?



T_1: Begin transaction

T_1: Update page 7, slot 9: …

Database checkpoint

T_1: Update page 4, slot 6: …

T_2: Begin transaction

T_2: Update page 3, slot 1: …

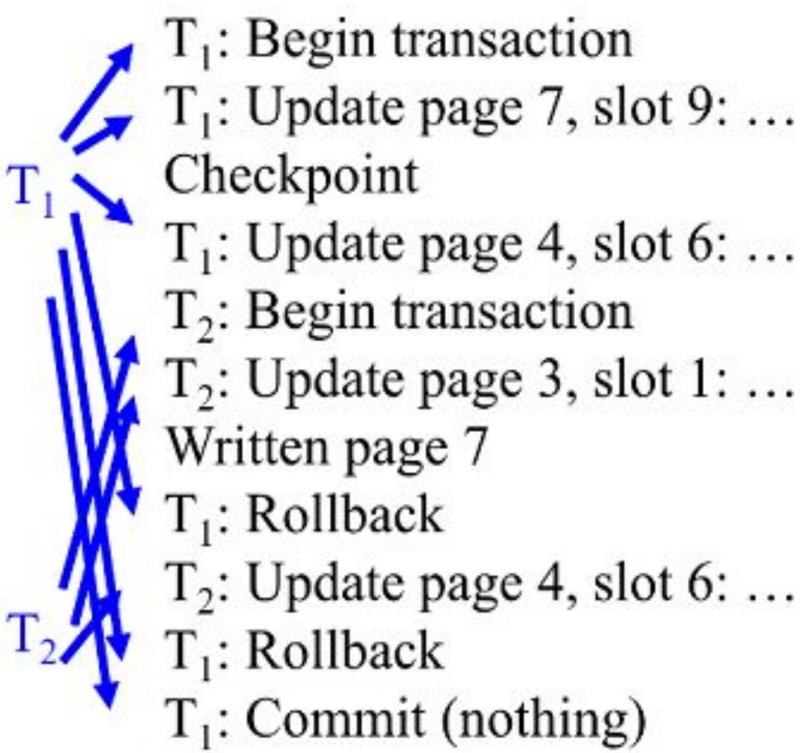
Written page 7

T_1: Rollback

T_2: Update page 4, slot 6: …

T_1: Rollback

T_1: Commit (nothing)



T_1: Begin transaction

T_1: Update page 7, slot 9: …

Checkpoint

T_1: Update page 4, slot 6: …

T_2: Begin transaction

T_2: Update page 3, slot 1: …

Written page 7

T_1: Rollback

T_2: Update page 4, slot 6: …

T_1: Rollback

T_1: Commit (nothing)

Lock management during transaction rollback

1970s: simple & conservative

- Lock release at end-of-transaction

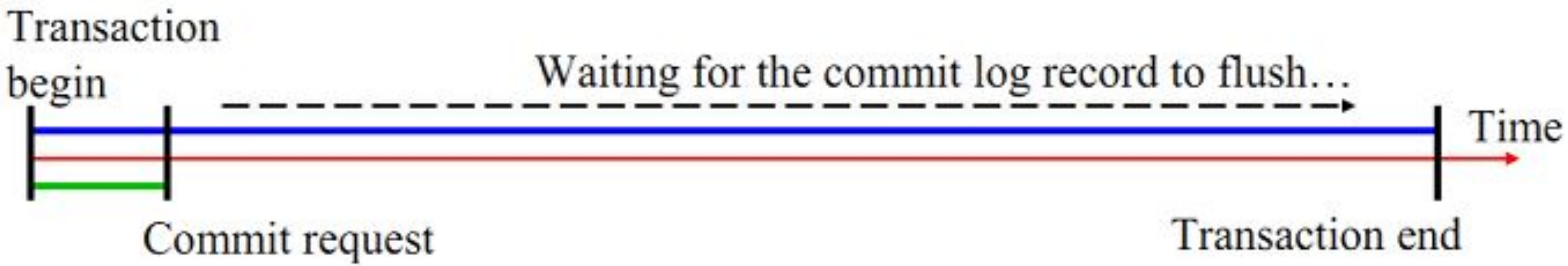
1990s: row-level locking [ARIES]

- Release of read-only locks ⇐ LSN for commit log record
- Incremental release of read-write locks (during rollback)
- “Save points”

2010s: controlled lock violation

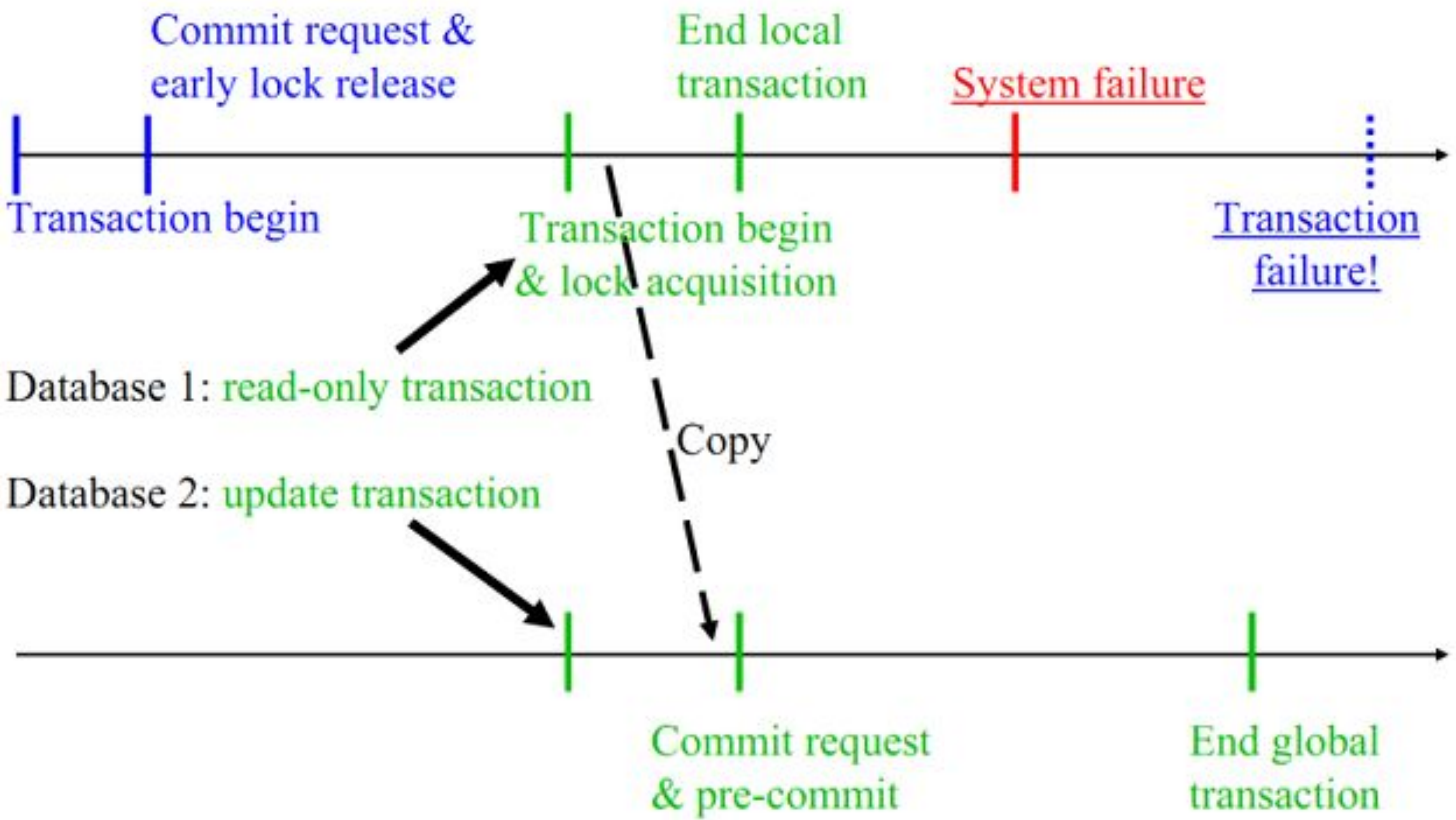
- Repair 1980s early lock release: controlled lock violation

Early lock release, controlled lock violation

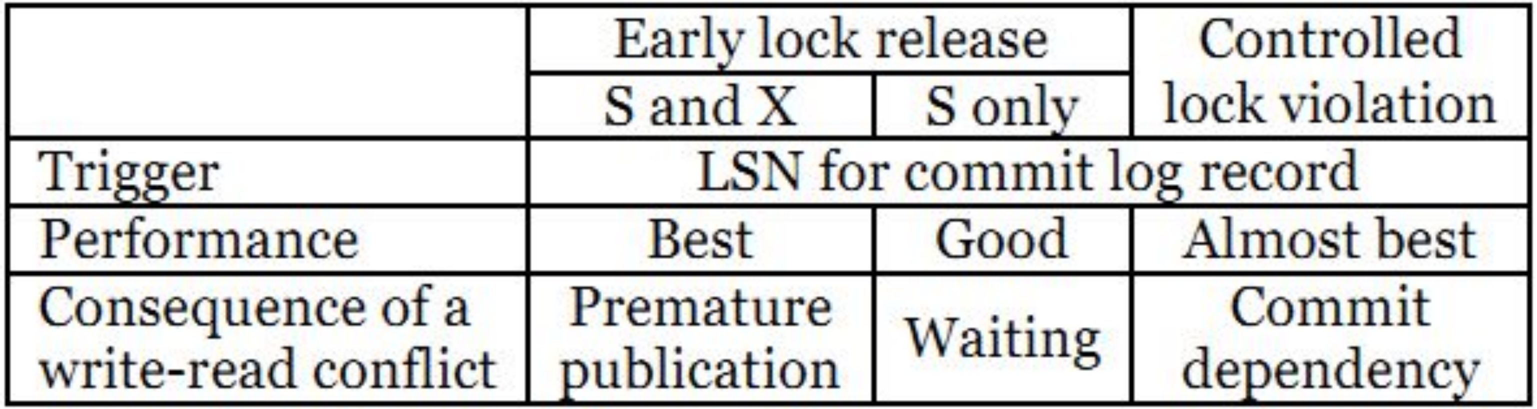


Transaction phases → ↓ Techniques	Read phase = transaction & application logic	Commit logic			Hardening = force log to stable storage
		Commit preparation	Commit log record	Update propagation	
Traditional locking	▓	n/a	▓	n/a	▓
Controlled lock violation	▓		▓		░

Premature publication with early lock release

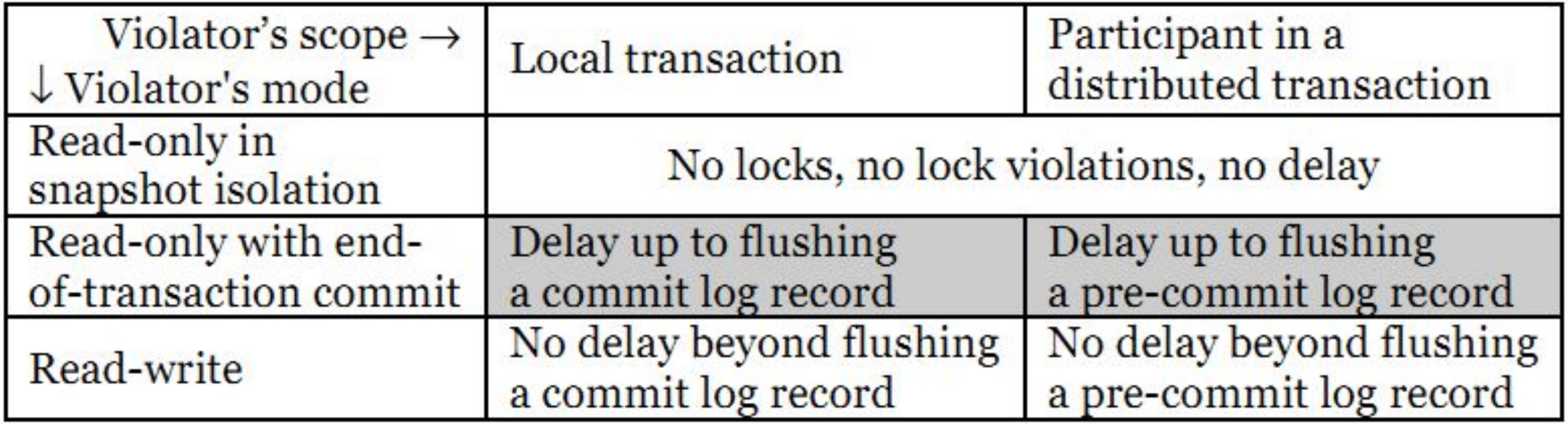


Early lock release vs controlled lock violation



	Early lock release		Controlled lock violation
	S and X	S only	
Trigger	LSN for commit log record		
Performance	Best	Good	Almost best
Consequence of a write-read conflict	Premature publication	Waiting	Commit dependency

Commit delays due to controlled lock violation



Violator’s scope → ↓ Violator's mode	Local transaction	Participant in a distributed transaction
Read-only in snapshot isolation	No locks, no lock violations, no delay	
Read-only with end-of-transaction commit	Delay up to flushing a commit log record	Delay up to flushing a pre-commit log record
Read-write	No delay beyond flushing a commit log record	No delay beyond flushing a pre-commit log record

Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, **system**, media **failures**; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

System failure ⇒ system restart

Bad: server process & memory (transaction, lock, buffer pool managers)

Good: database, recovery log (even if slightly out-of-date)

1970s: System R

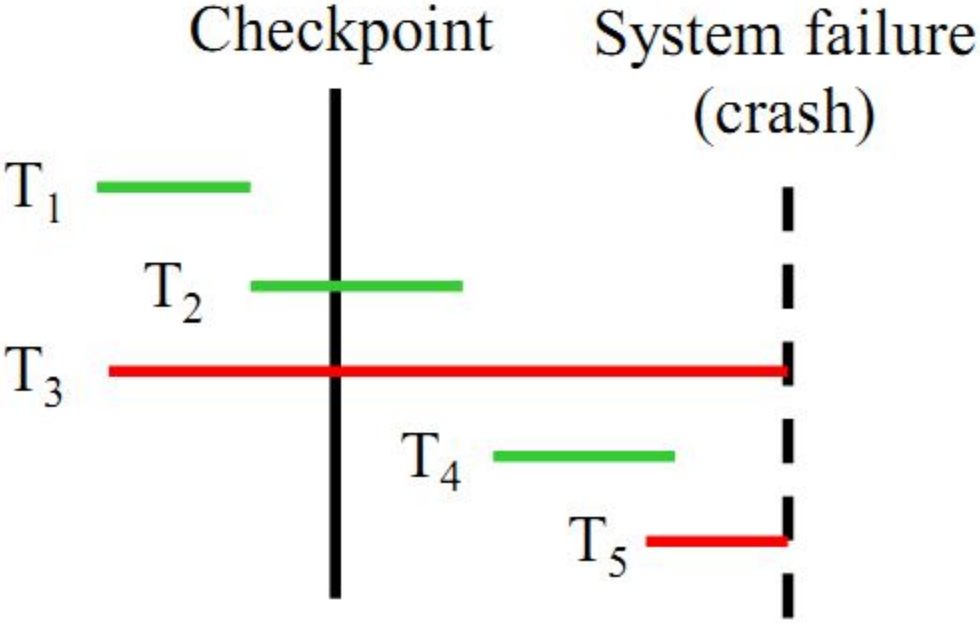
- Restart phases: log analysis, “redo” log scan, “undo” log scan
- Winner vs loser transactions

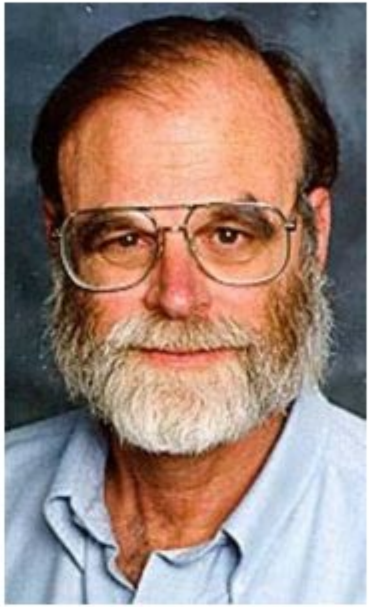
1990s: ARIES

- Lock re-acquisition during “redo”, new transactions during “undo”
- Transaction-by-transaction “undo”, no backward log scan

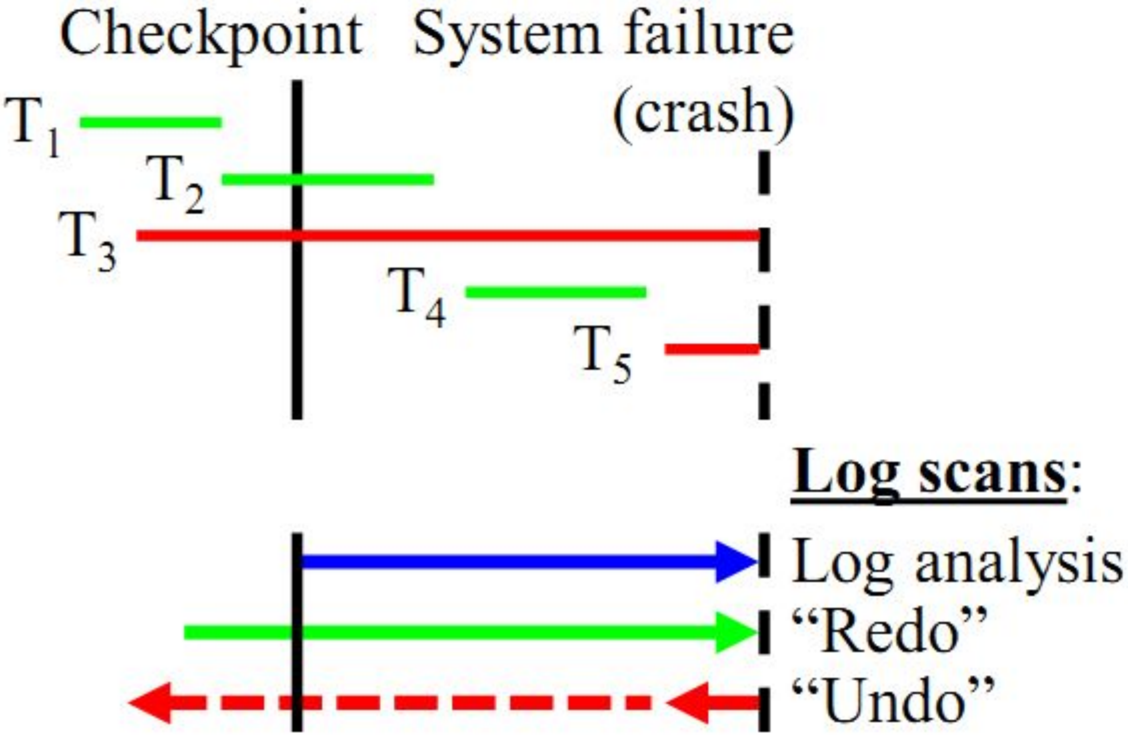
2010s: instant restart

Winner vs loser transactions





ARIES system restart and its log scans



Checkpoint information = current server state

For each active transaction:

- LSN of last update

Atomicity

For "undo" phase with checkpoints and new transactions:

Isolation

- Locks acquired and held (ignoring read-only locks)

For each dirty buffer page:

Durability

- Page identifier
- LSN of the first update (not yet on storage)

Availability

For instant recovery, replace:

- LSN of the last update (instead of the first)

Classic restart: three phases & log scans

Log analysis ⇒ recover pre-crash server state

- Scan log from last good checkpoint to crash
- Identify transactions active at crash ⇒ "losers"

"Redo" ⇒ recover pre-crash database state (in buffer pool)

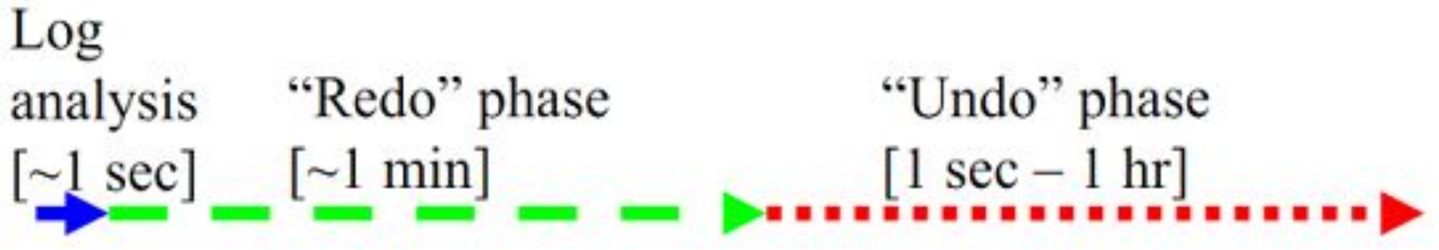
- Scan log from oldest unsaved change to crash
- Replay all missing changes in all database pages

"Undo" ⇒ transaction-consistent pre-crash snapshot

- Scan log from crash to oldest loser's start-of-transaction
- Roll back loser transactions (write compensation log records)

Restart phases & their typical durations

Classic design (1970s):



Log analysis [~1 sec]

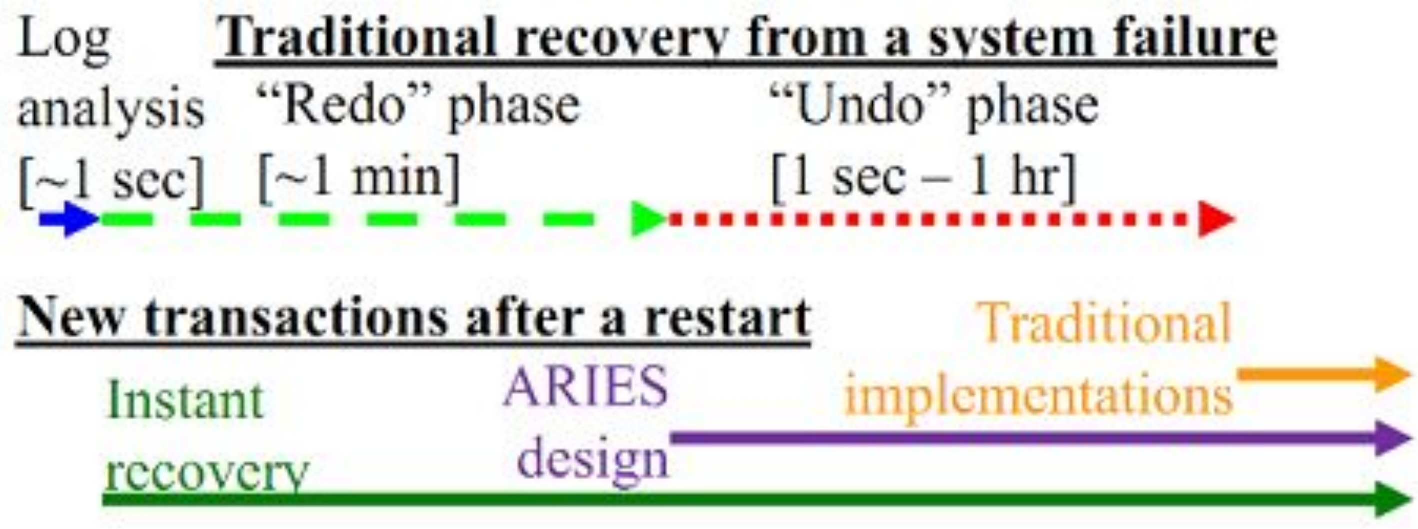
“Redo” phase [~1 min]

“Undo” phase [1 sec – 1 hr]

MS SQL Server improvements (as an example):

- 1998: new transactions after “undo”, parallel “undo” (tx rollback)
- ~2000: new transactions after “redo”, concurrent to “undo”
- ~2017: no “undo” phase, version clean-up like ghost clean-up
  “constant-time recovery” = “accelerated database recovery”

Restart ⇒ new transactions, new checkpoints



↓ (apparent) mean time to repair ⇒ ↑ availability

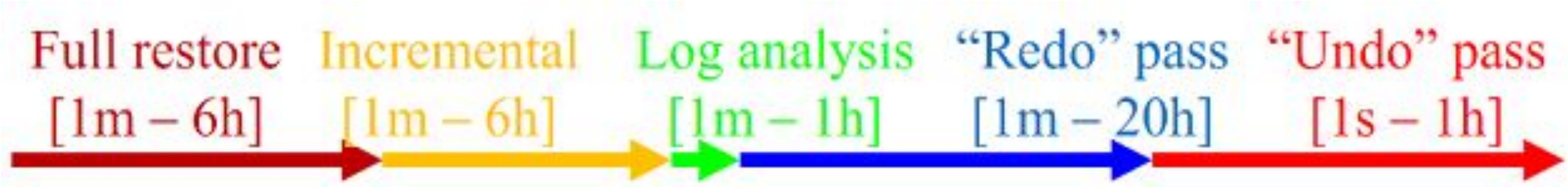
Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, **media failures**; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Media failures ⇒ media restore

Bad: database (maybe only a single device)

Good: server process, log archive, recovery log



Optimizations:

- Remain online with database pages in the buffer pool
- All backups sorted on database page identifier ⇒ merge in single pass
- Sort the log records…

Point-in-time restore

From an earlier backup:

- Repeat missing history
- Roll back incomplete transactions

From a later backup:

- Roll back incomplete transactions
- Roll back transactions committed too late
- Roll back system transactions
- Alternative: roll back database history

Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, media failures; **double failures**; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Multiple failures

Server crash during restart

- Re-start the same restart logic
- Early checkpoints!

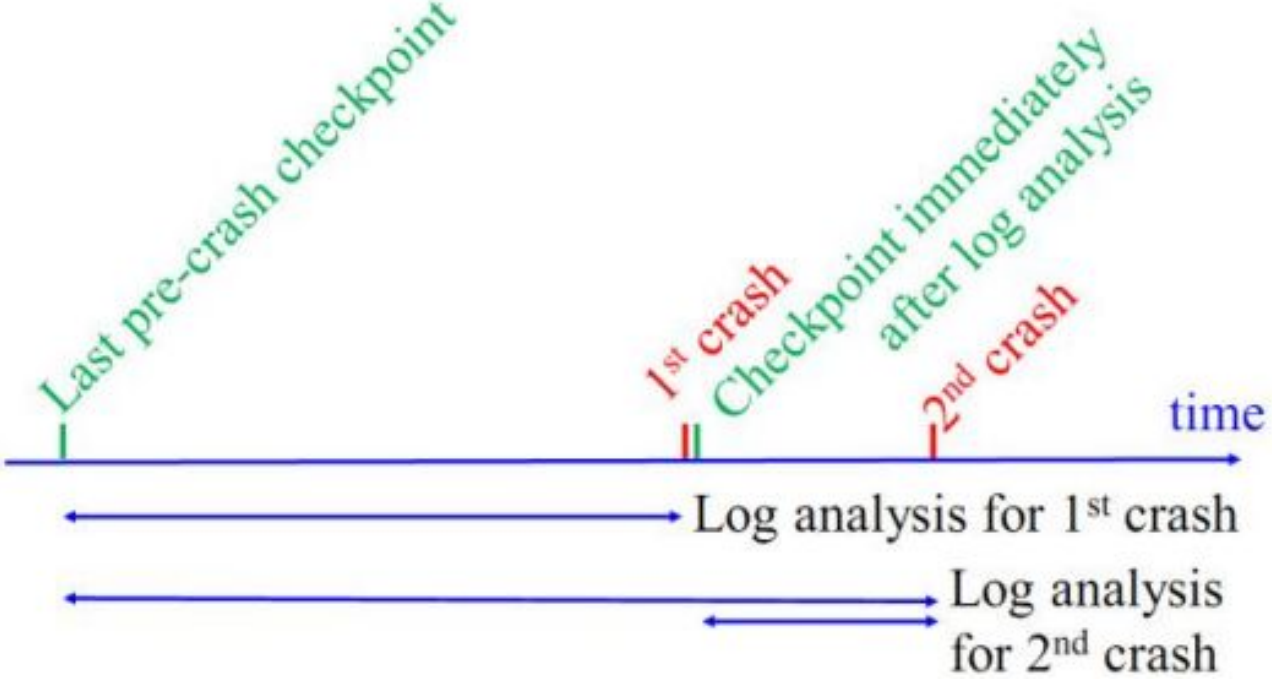
Media loss during restore

- Start over with new replacement media

System failure + media failure ⇒ reboot

- Restore, then restart
- Typical for in-memory databases and node failures

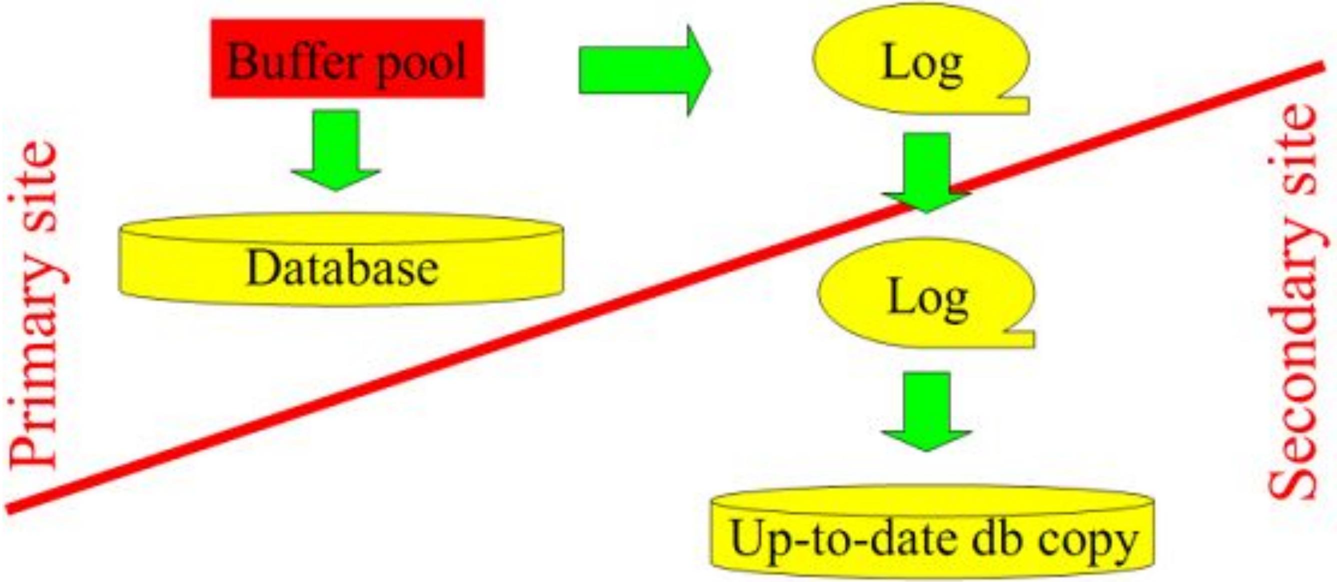
Double failure: checkpoint during restart


Last pre-crash checkpoint
1st crash
Checkpoint immediately after log analysis
2nd crash
time
Log analysis for 1st crash
Log analysis for 2nd crash

Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, media failures; double failures; **failover**
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Traditional log shipping



Node failure ⇒ node failover

Hot stand-by:

- Log shipping and continuous “redo” recovery
- Random I/O as much as the primary server

Cold stand-by:

- Capture backups + log archive
- Rely on restore + restart

Warm stand-by:

- Recent database image + up-to-date metadata + recovery log
- Incremental “redo”

Excursus: “log shipping to storage”

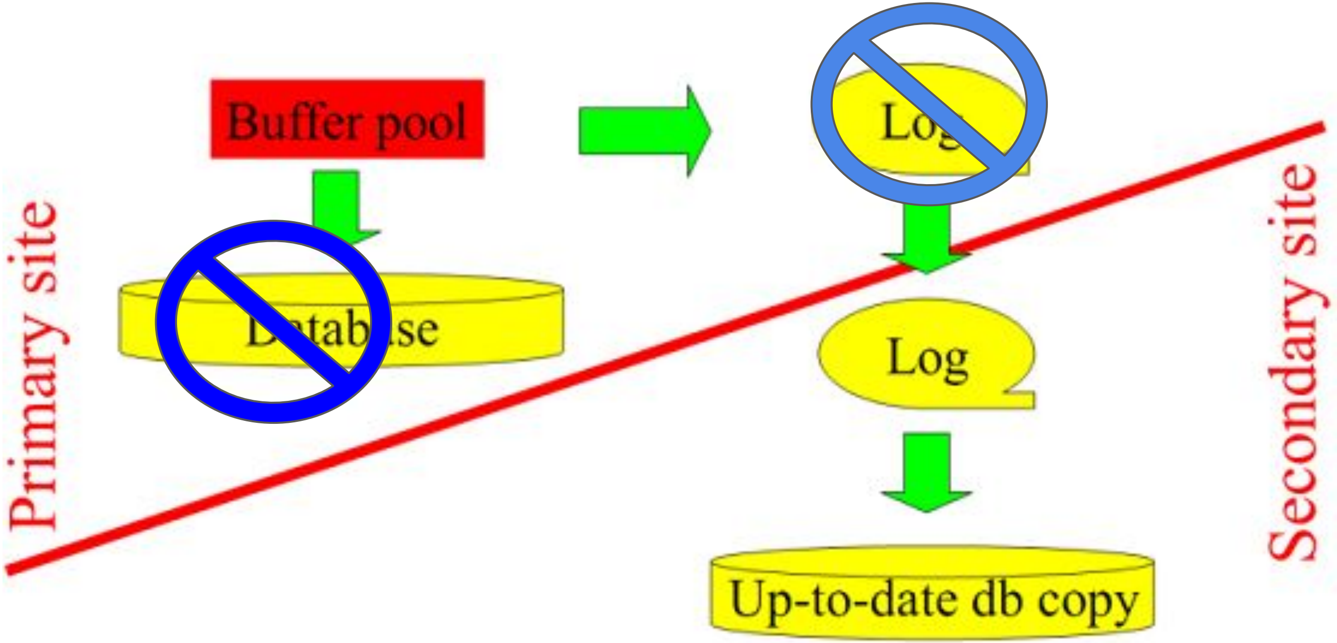
Amazon Aurora:

- Postgres or MySQL front-end
  - Local buffer pool, but no local storage
- “Storage” is another server
  - kept up-to-date by log shipping
  - serves buffer faults of the primary server
  - optional: shards & replicas

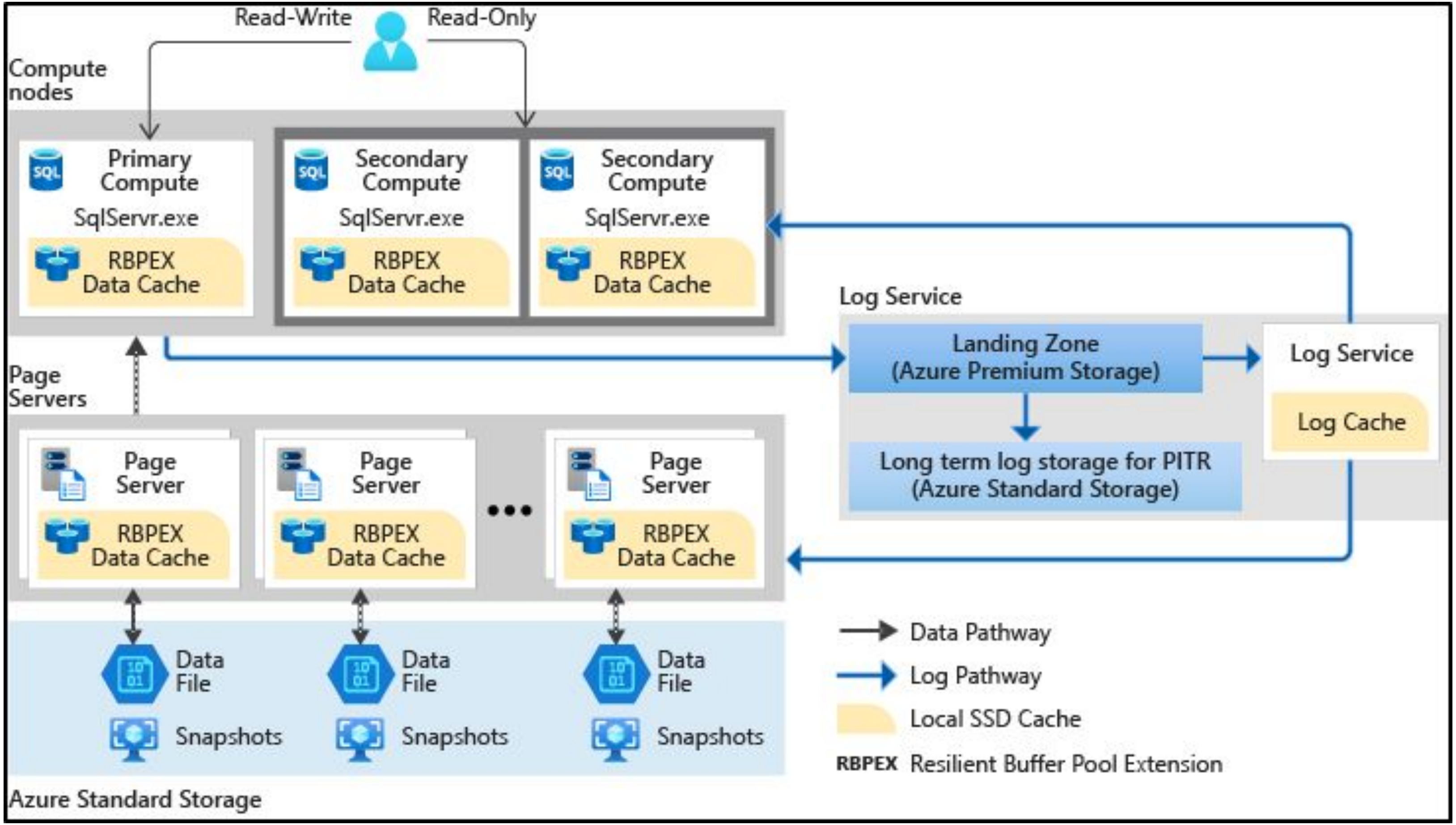
Assumption: storage imposes heavy CPU or memory load

- e.g., Microsoft Socrates = SQL Server hyperscale

“Log shipping to storage”



Microsoft SQL Hyperscale



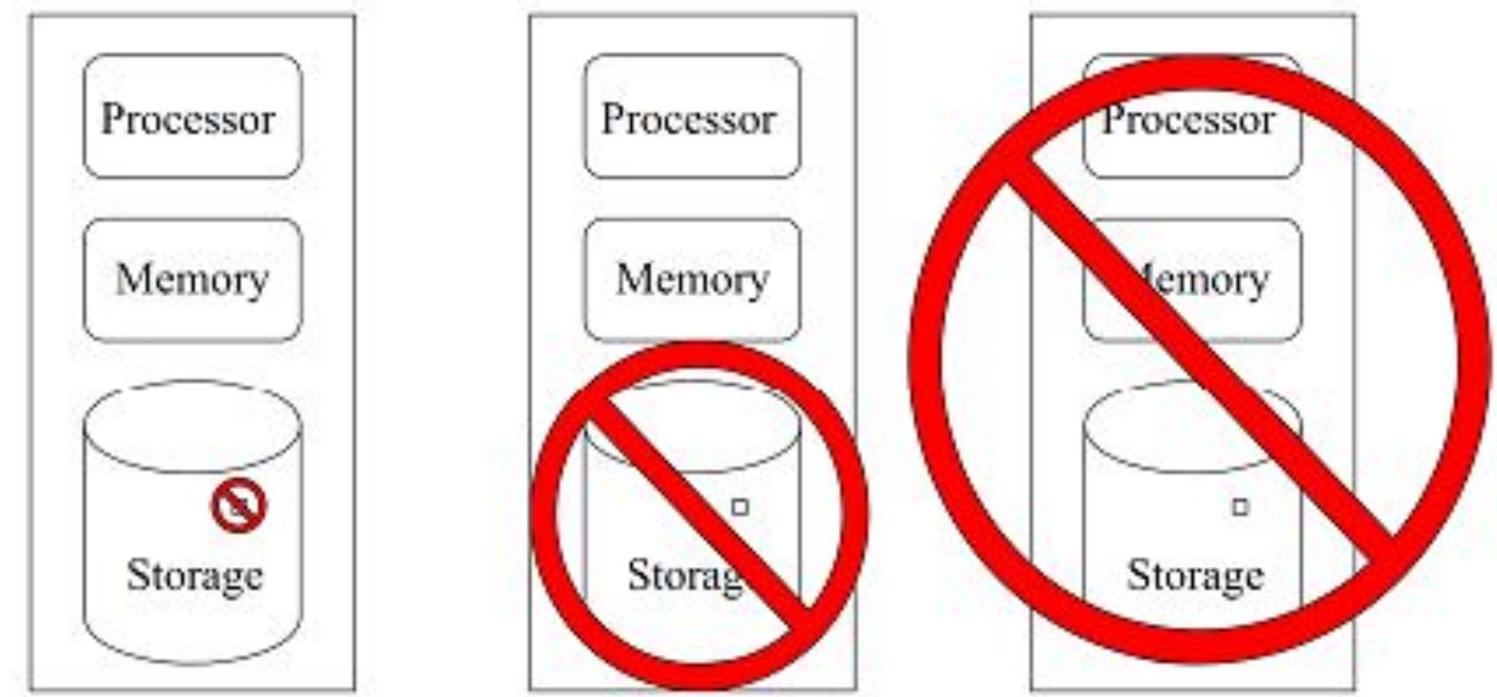
Excursus: “log shipping to self”

- e.g., databases in persistent memory, Halloween protection
- Apply updates only after completed calculation of changes
  - Plan phase
  - Statement
  - Transaction
- ZZZ…

Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, and media failures; double failures; failover
- **Single-page failures**
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Failure scopes



Single-page failure

Media failure

System failure

SQL Server "online page-level restore"

1. Scan backups (full, differential, incremental)
   extract most recent page image(s)
2. Scan log archive + recovery log
   extract + apply "redo" for affected page(s)

Database locks & I/O: very little

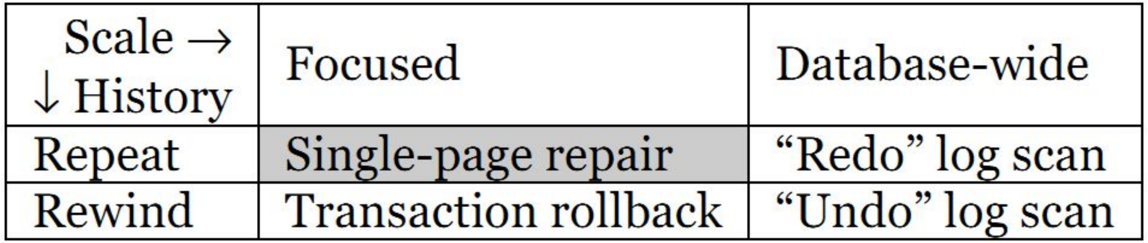
Backup + log archive I/O: lots!

Single-page failure ⇒ single-page repair

Exploit per-page chains of log records

Anchor for the chain:

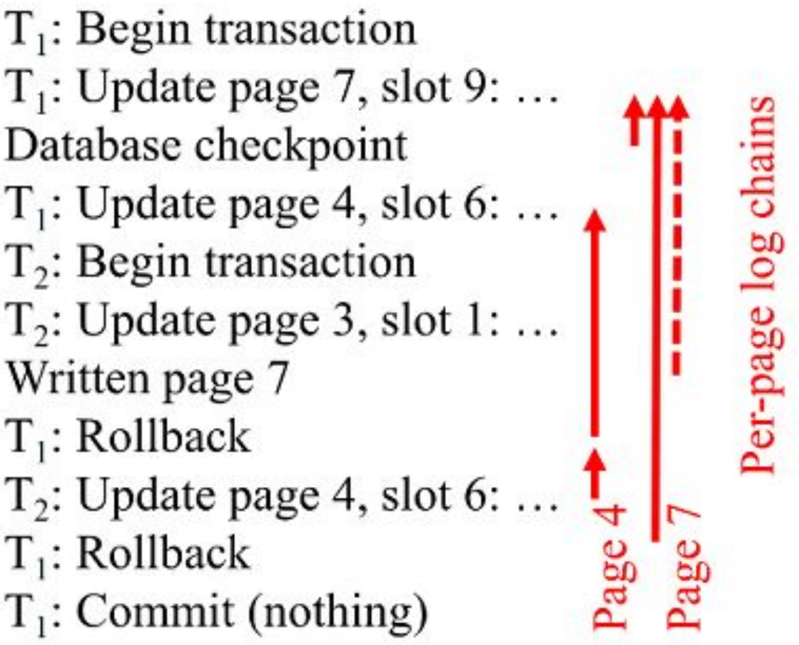
- Metadata
- Index parent node



Scale → ↓ History	Focused	Database-wide
Repeat	Single-page repair	“Redo” log scan
Rewind	Transaction rollback	“Undo” log scan

If fast enough, could be used deliberately – “write elision”, “read elision”

Per-page chains of log records



T_1: Begin transaction

T_1: Update page 7, slot 9: …

Database checkpoint

T_1: Update page 4, slot 6: …

T_2: Begin transaction

T_2: Update page 3, slot 1: …

Written page 7

T_1: Rollback

T_2: Update page 4, slot 6: …

T_1: Rollback

T_1: Commit (nothing)

<u>MS SQL Server</u>: consistency checks in “redo” pass, in log shipping, and in media recovery

<u>Oracle</u>: single-page rollback for snapshot isolation

<u>Single-page failures</u>: “Redo” recovery in reverse order since backup, move, or format

Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, and media failures; double failures; failover
- Single-page failures
  - On-demand local repair: **self-repairing indexes**
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

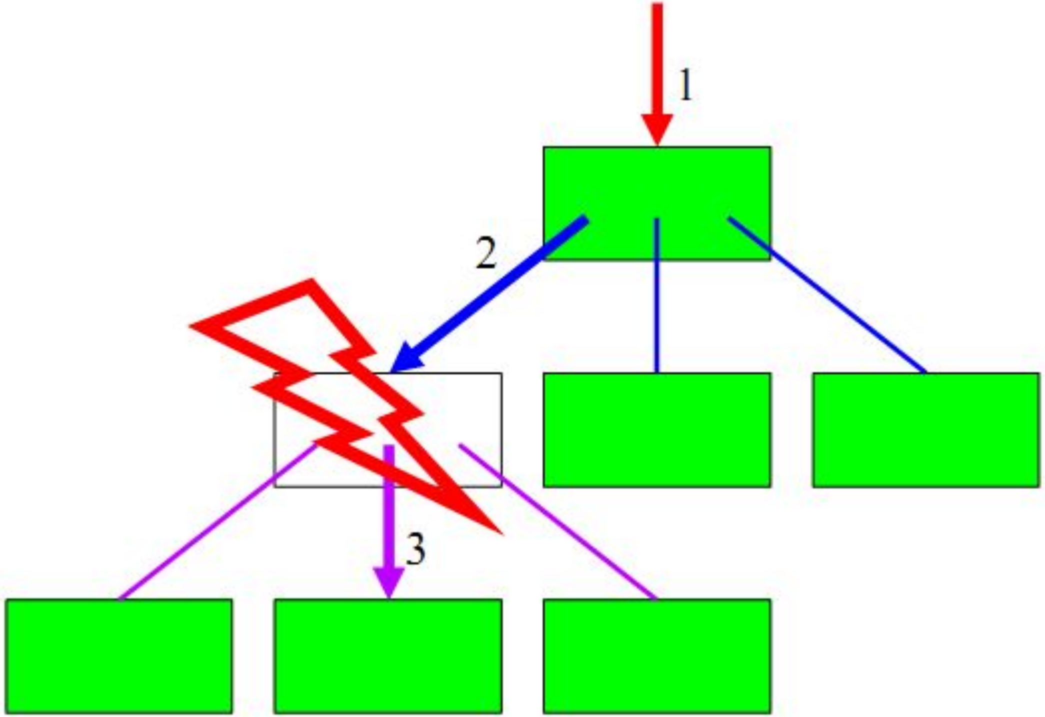
Single-page failure ⇒ single-page repair

Continuous comprehensive consistency checking

- Within nodes
- B-tree structure (pointers)
- Key ranges

B-tree structure

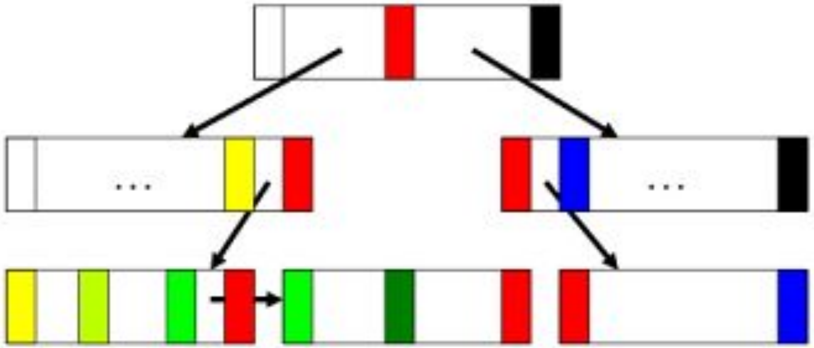
- Single pointer per node
- Fence keys

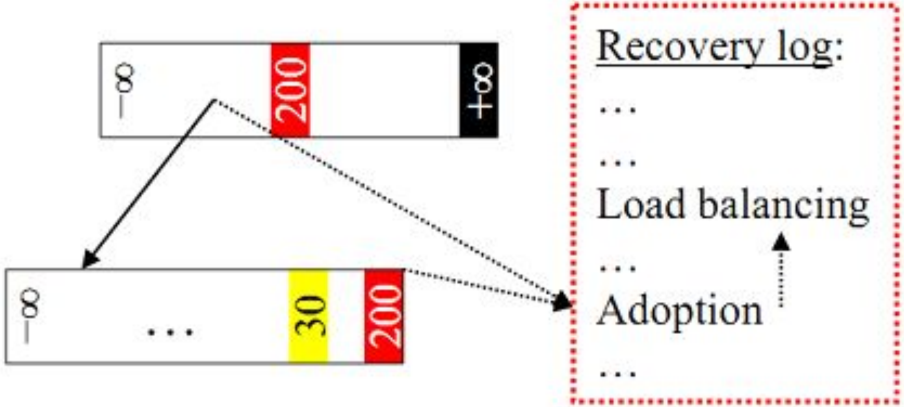


Foster b-trees, self-repairing indexes

Single pointer to each node at all times, local overflow gets “adopted”

ExpectedChildLSN with each parent-to-child pointer

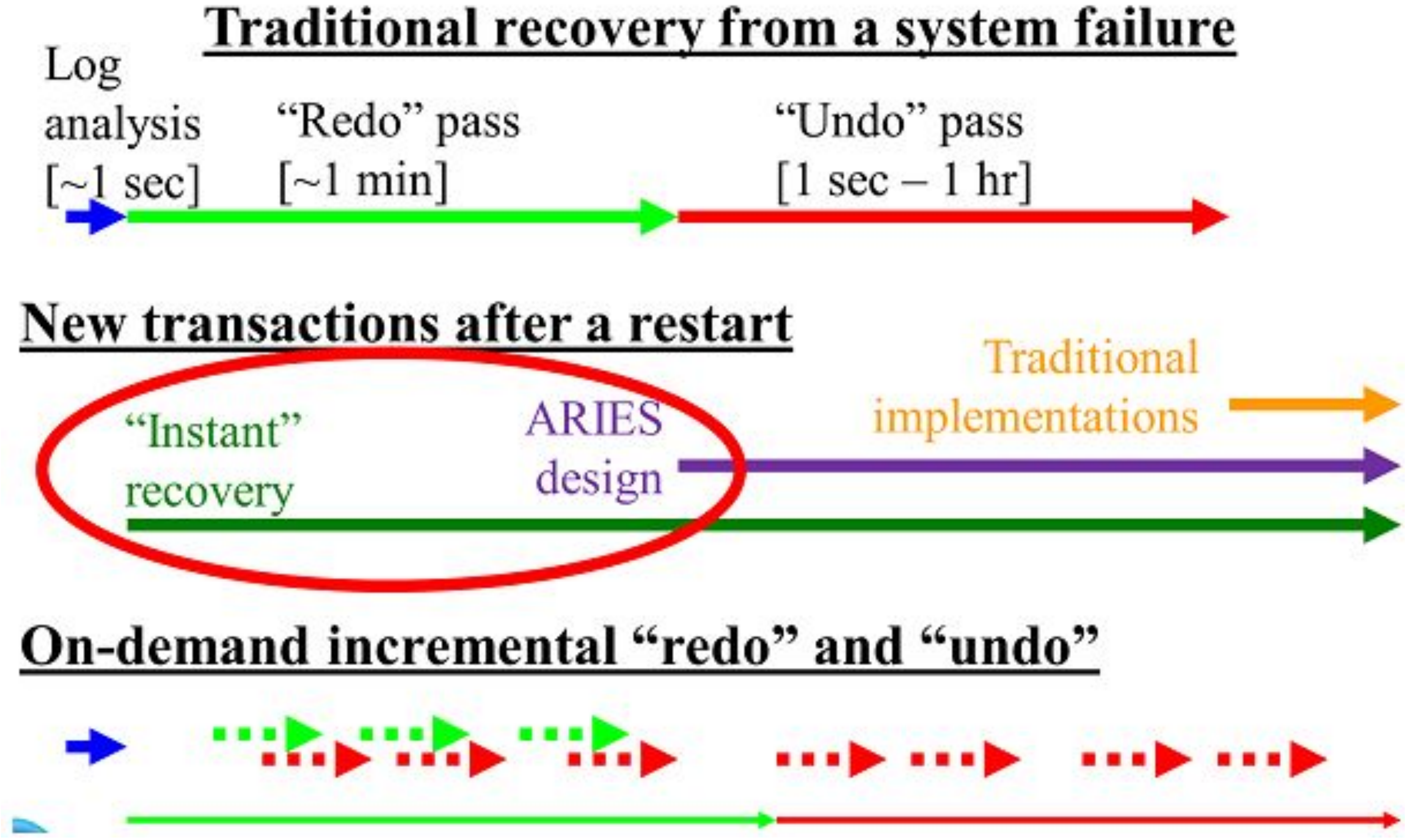




Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, and media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - **Instant restart**, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Restart, new transactions, new checkpoints



Traditional recovery from a system failure

Log analysis [~1 sec]

"Redo" pass [~1 min]

"Undo" pass [1 sec – 1 hr]

New transactions after a restart

"Instant" recovery

ARIES design

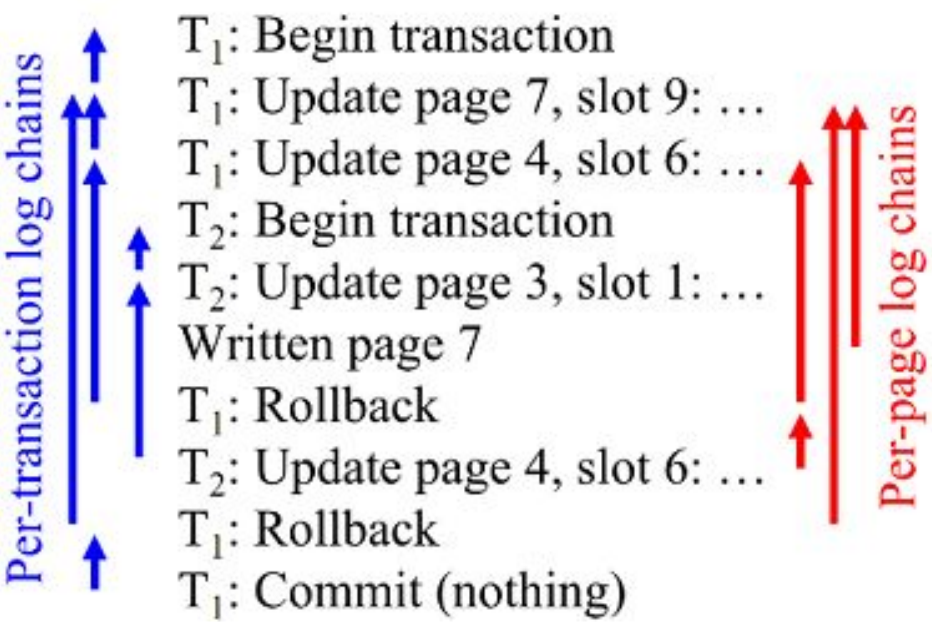
Traditional implementations

On-demand incremental "redo" and "undo"

Instant restart – before log analysis

Two transactions – interleaved

Server events – e.g., page written



T_1: Begin transaction

T_1: Update page 7, slot 9: …

T_1: Update page 4, slot 6: …

T_2: Begin transaction

T_2: Update page 3, slot 1: …

Written page 7

T_1: Rollback

T_2: Update page 4, slot 6: …

T_1: Rollback

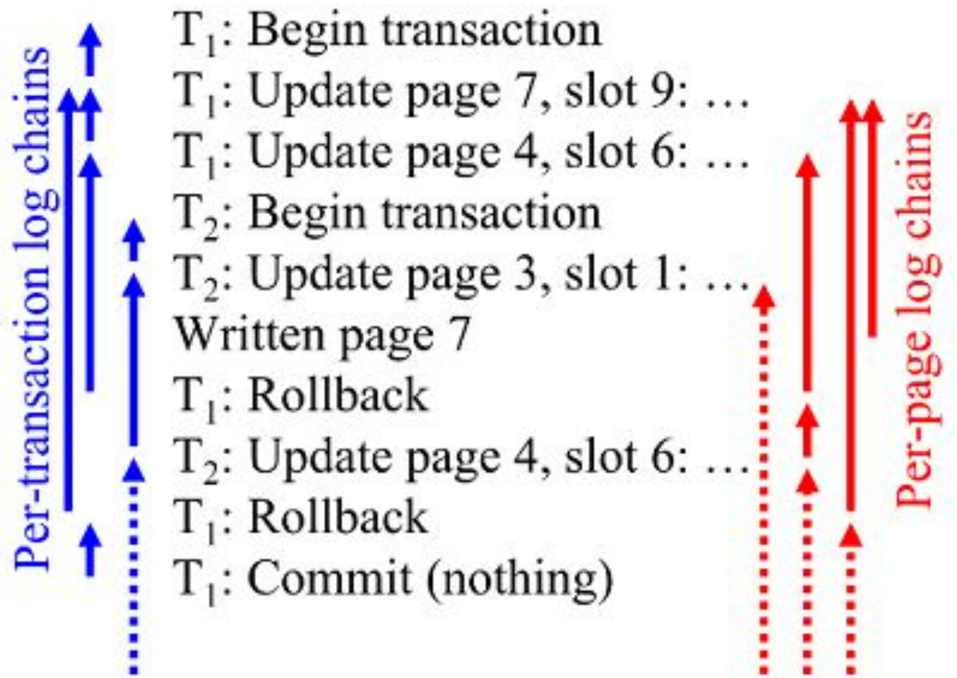
T_1: Commit (nothing)

Instant restart – after log analysis

Log analysis recovers server state

- Transaction manager:
  Loser transactions, LastLSN
- Lock manager:
  Losers' read-write locks
- Buffer pool:
  Dirty pages, LastLSN
  (page descriptors only)

Log analysis enables checkpoints and new user transactions!



T_1: Begin transaction
T_1: Update page 7, slot 9: …
T_1: Update page 4, slot 6: …
T_2: Begin transaction
T_2: Update page 3, slot 1: …
Written page 7
T_1: Rollback
T_2: Update page 4, slot 6: …
T_1: Rollback
T_1: Commit (nothing)

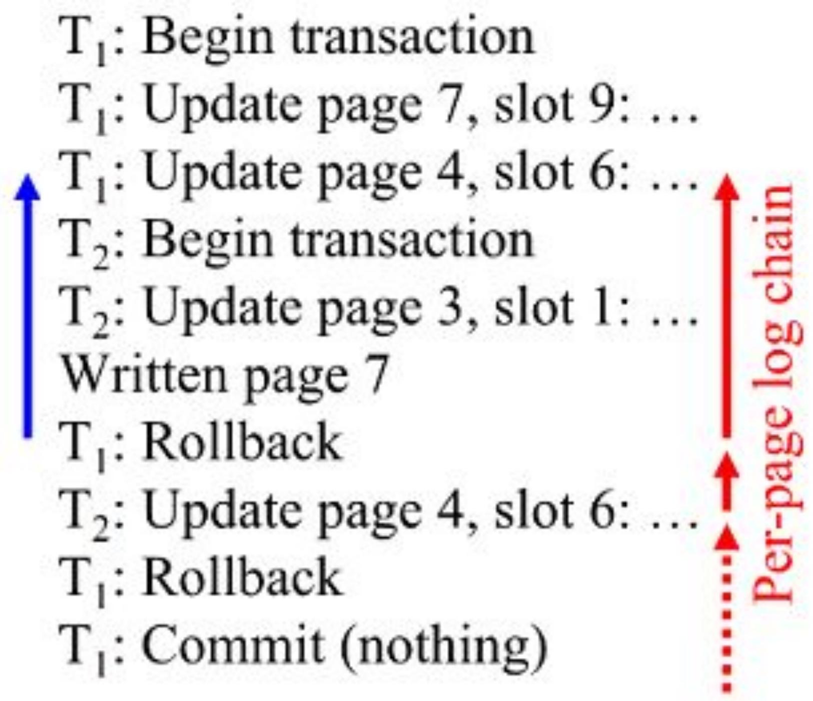
Instant restart – on-demand “redo”

New transaction needs page 4

- triggers single-page repair = incremental “redo”

Optimization opportunity:

- Avoid two updates of slot 6: rollback log record with PriorLSNsamePage == PriorLSNsameTx

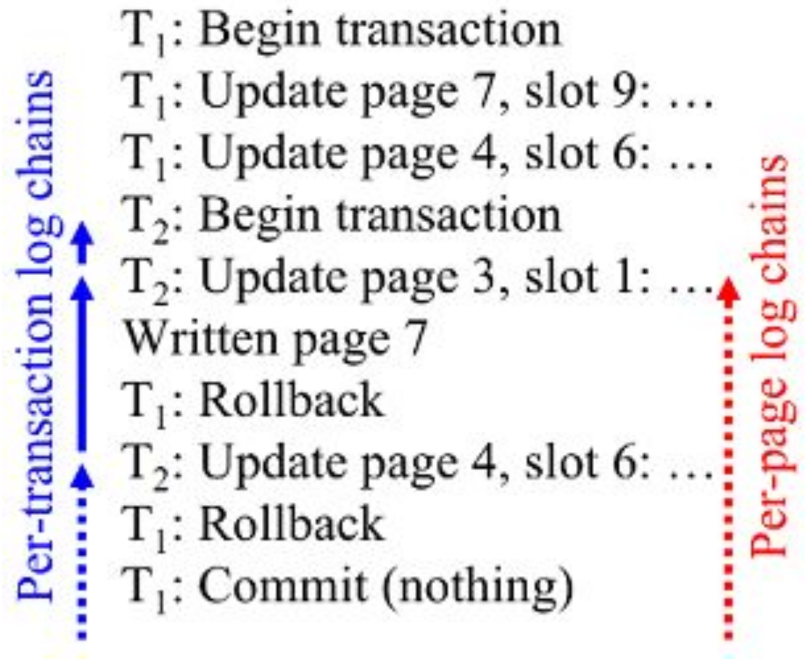


T_1: Begin transaction
T_1: Update page 7, slot 9: …
T_1: Update page 4, slot 6: …
T_2: Begin transaction
T_2: Update page 3, slot 1: …
Written page 7
T_1: Rollback
T_2: Update page 4, slot 6: …
T_1: Rollback
T_1: Commit (nothing)

Instant restart – on-demand “undo”

New transaction conflicts with lock held by loser transaction T2

- triggers transaction rollback
  = incremental “undo”
- can trigger single-page repair
  = incremental “redo”



T_1: Begin transaction
T_1: Update page 7, slot 9: …
T_1: Update page 4, slot 6: …
T_2: Begin transaction
T_2: Update page 3, slot 1: …
Written page 7
T_1: Rollback
T_2: Update page 4, slot 6: …
T_1: Rollback
T_1: Commit (nothing)

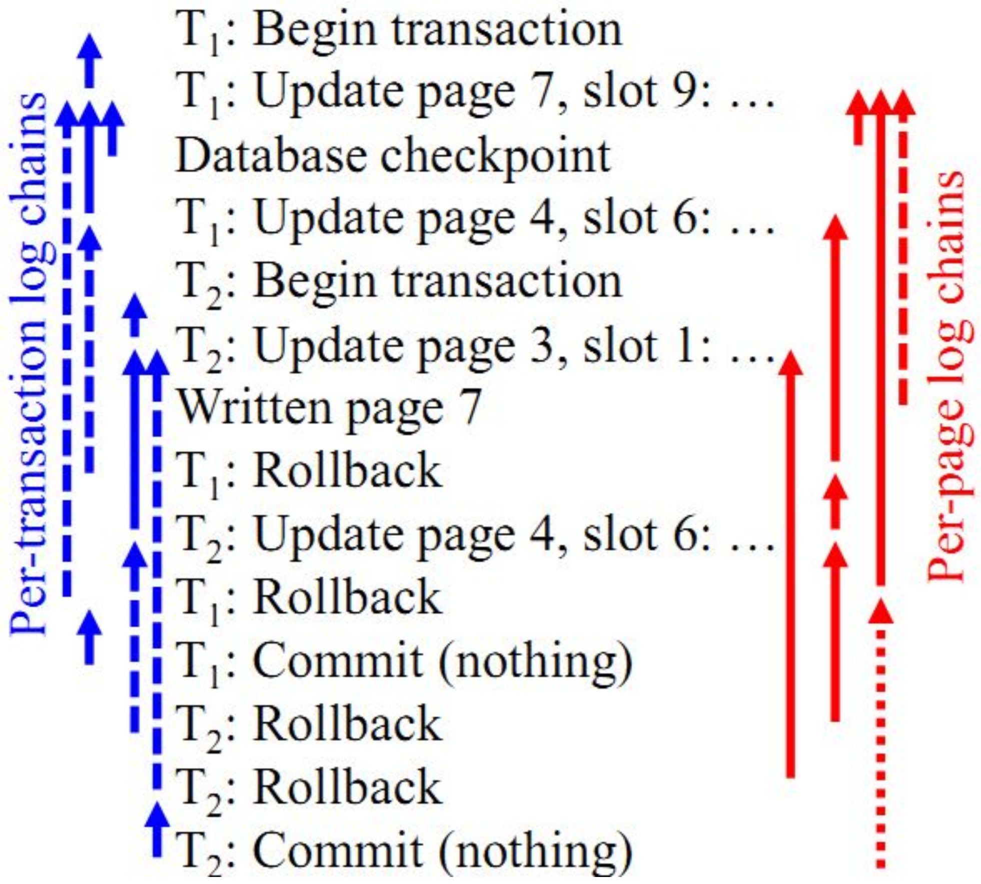
Instant restart – after transaction T2 rolls back

Pages 3 & 4 repaired

- page 7 still “in doubt”

Transaction T2 rolled back

- new rollback log records
- new commit log record



T_1: Begin transaction

T_1: Update page 7, slot 9: …

Database checkpoint

T_1: Update page 4, slot 6: …

T_2: Begin transaction

T_2: Update page 3, slot 1: …

Written page 7

T_1: Rollback

T_2: Update page 4, slot 6: …

T_1: Rollback

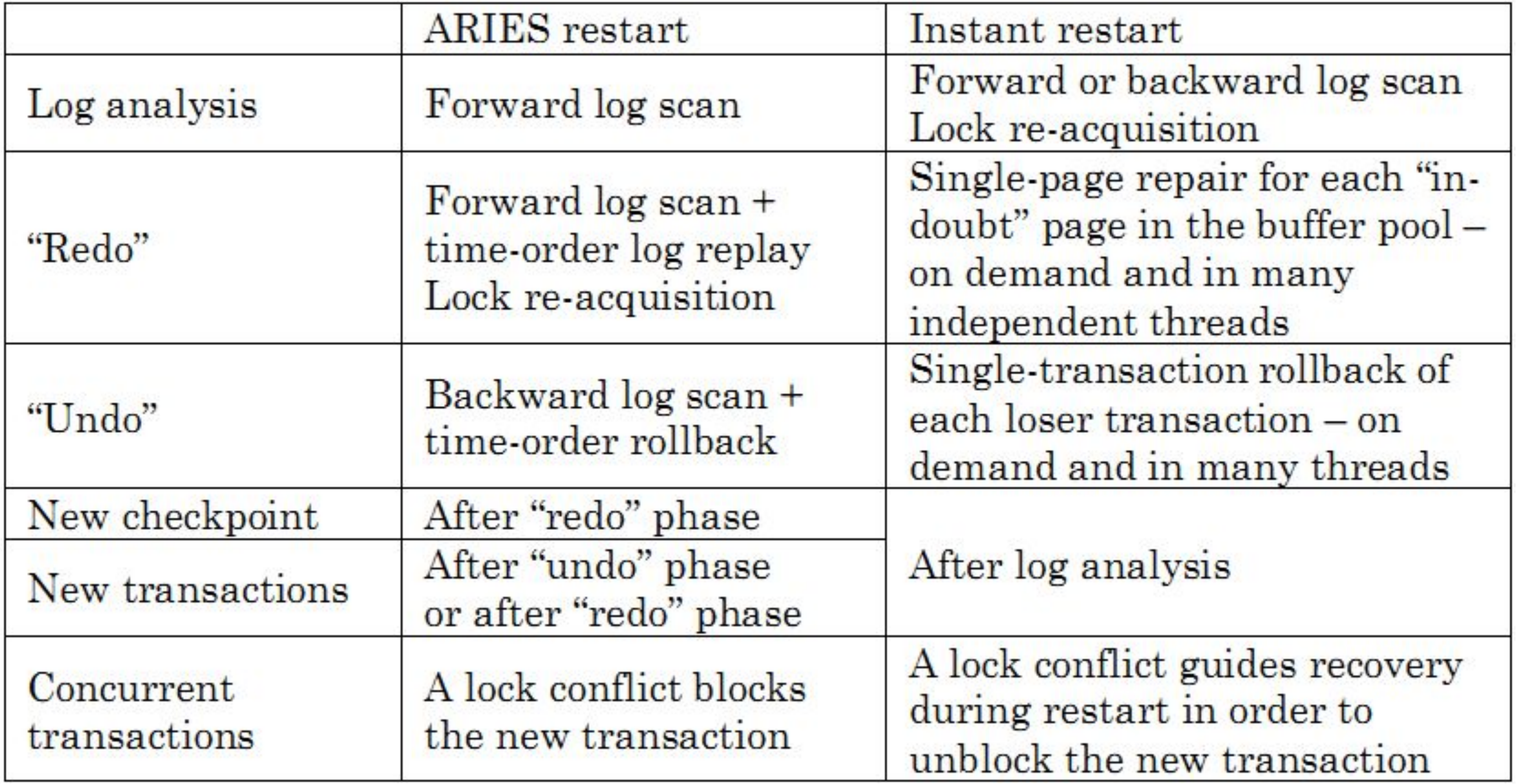
T_1: Commit (nothing)

T_2: Rollback

T_2: Rollback

T_2: Commit (nothing)

ARIES restart vs instant restart



	ARIES restart	Instant restart
Log analysis	Forward log scan	Forward or backward log scan Lock re-acquisition
“Redo”	Forward log scan + time-order log replay Lock re-acquisition	Single-page repair for each “in-doubt” page in the buffer pool – on demand and in many independent threads
“Undo”	Backward log scan + time-order rollback	Single-transaction rollback of each loser transaction – on demand and in many threads
New checkpoint	After “redo” phase	After log analysis
New transactions	After “undo” phase or after “redo” phase	
Concurrent transactions	A lock conflict blocks the new transaction	A lock conflict guides recovery during restart in order to unblock the new transaction

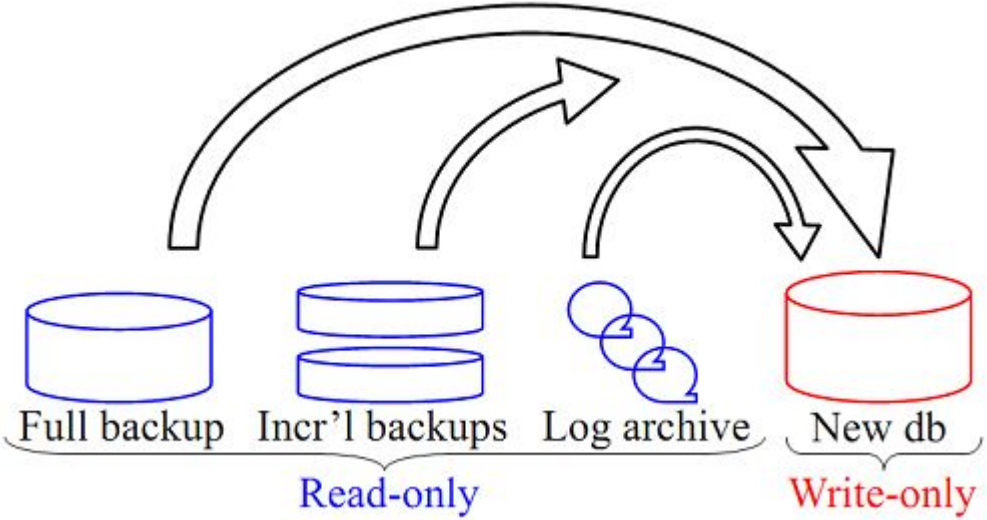
Agenda

- Write-ahead logging
    - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
    - Transaction, system, and media failures; double failures; failover
- Single-page failures
    - On-demand local repair: self-repairing indexes
- Instant recovery
    - Instant restart, restore, reboot
    - **Single-phase restore**, instant backup
    - Instant failover, fail-back
- Conclusions

Single-phase restore

Sorted backups and log archive (partitions)

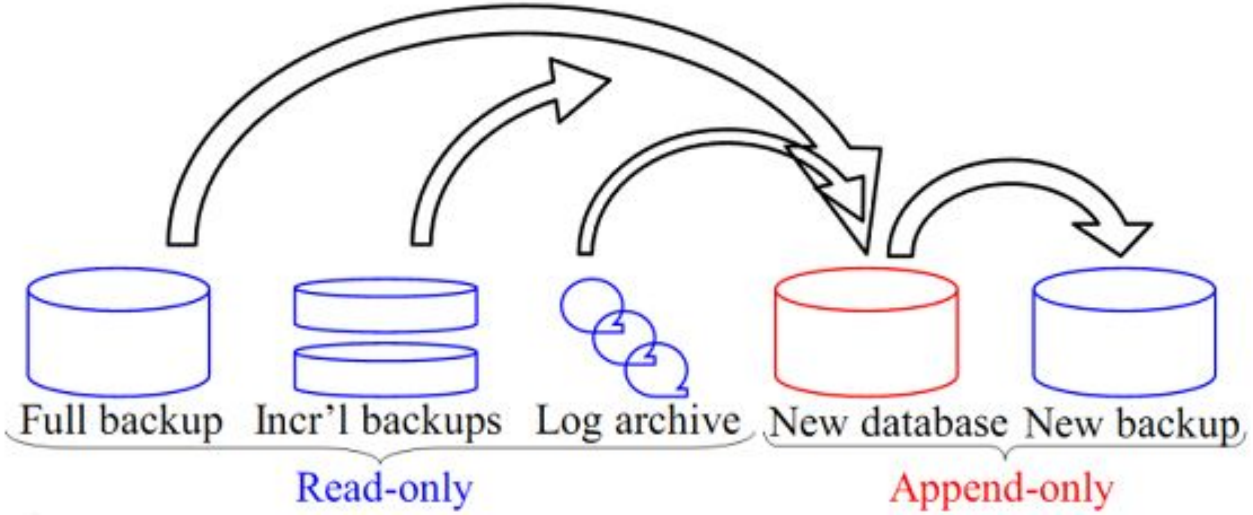
A single merge step!



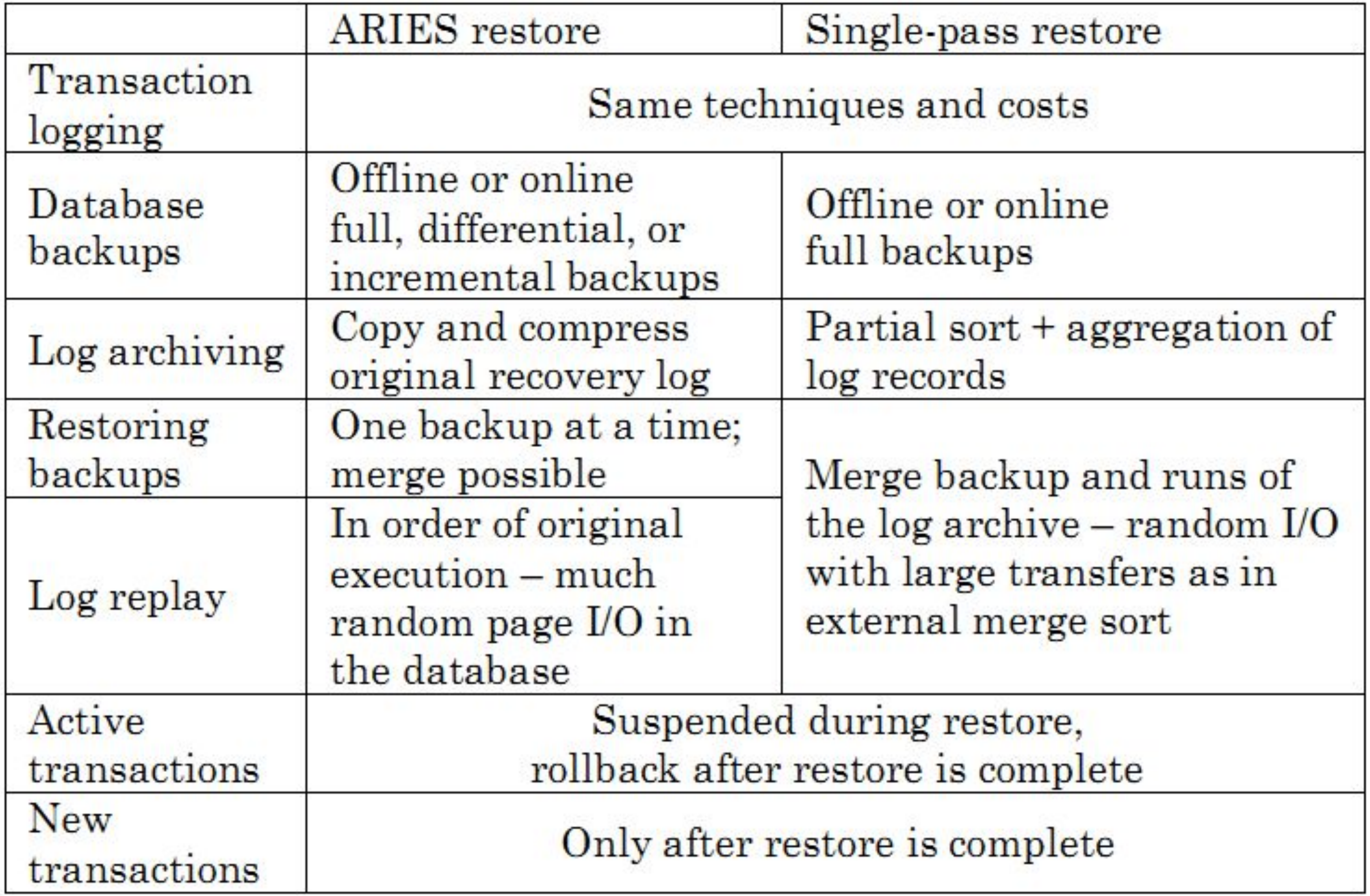
Value-added in single-phase restore

A single pipeline!

Also: consistency checks of old and new backups + new database



ARIES restore vs single-phase restore



	ARIES restore	Single-pass restore
Transaction logging	Same techniques and costs	
Database backups	Offline or online full, differential, or incremental backups	Offline or online full backups
Log archiving	Copy and compress original recovery log	Partial sort + aggregation of log records
Restoring backups	One backup at a time; merge possible	Merge backup and runs of the log archive – random I/O with large transfers as in external merge sort
Log replay	In order of original execution – much random page I/O in the database	
Active transactions	Suspended during restore, rollback after restore is complete	
New transactions	Only after restore is complete	

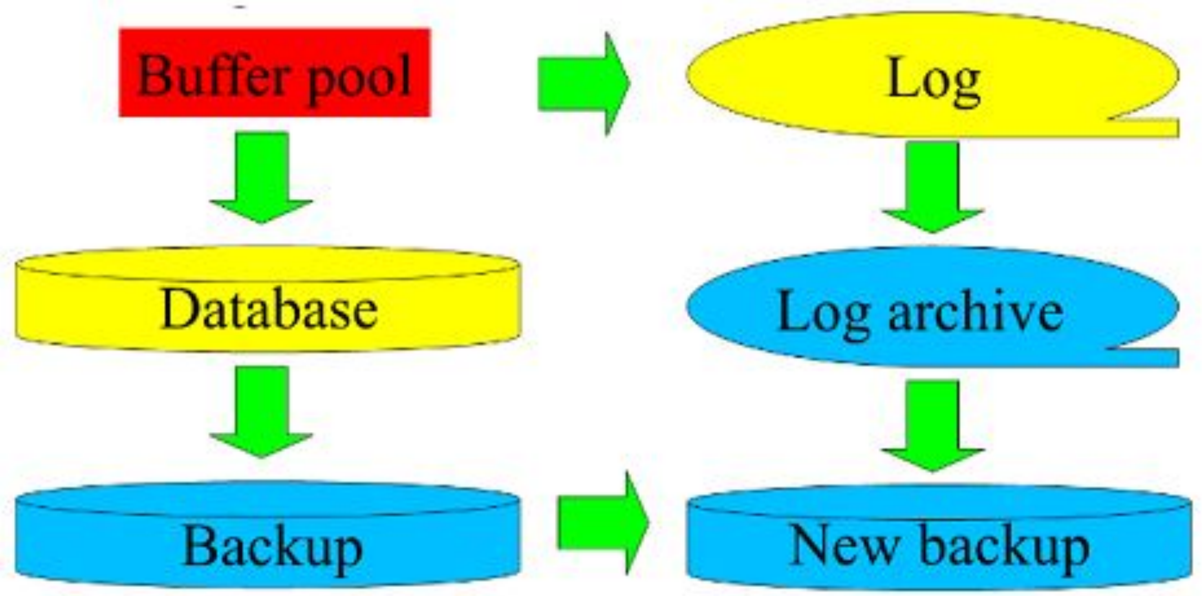
Agenda

- Write-ahead logging
    - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
    - Transaction, system, and media failures; double failures; failover
- Single-page failures
    - On-demand local repair: self-repairing indexes
- Instant recovery
    - Instant restart, restore, reboot
    - Single-phase restore, **instant backup**
    - Instant failover, fail-back
- Conclusions

Virtual backup

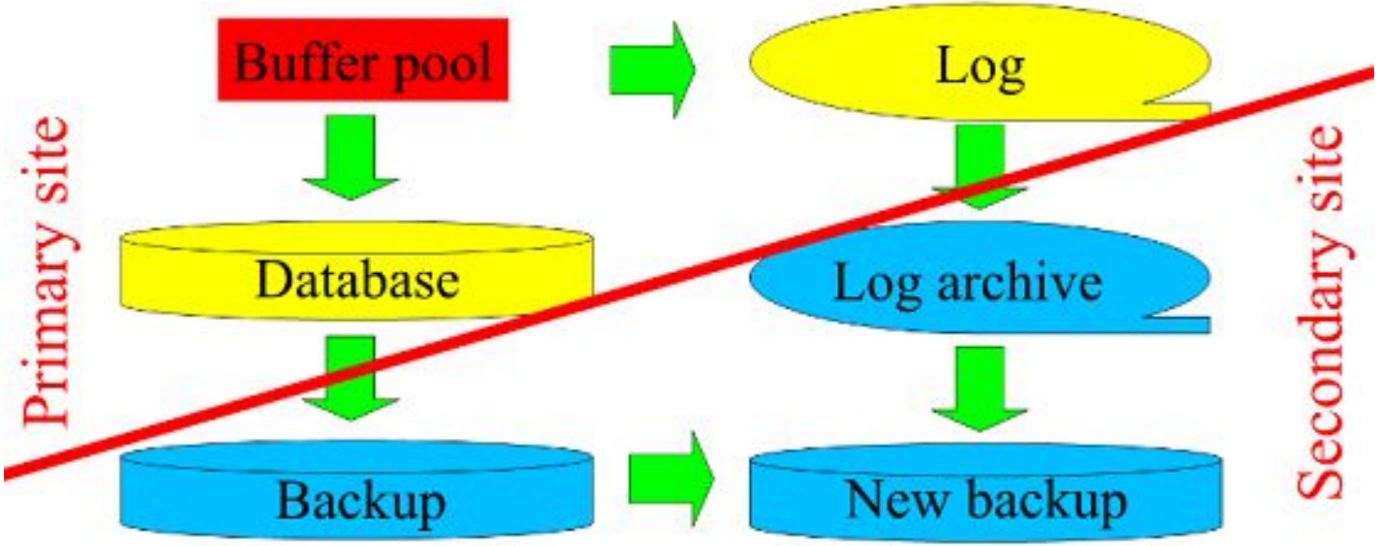
Backup = database + compression

Single-phase, just like single-phase restore



Remote virtual backup

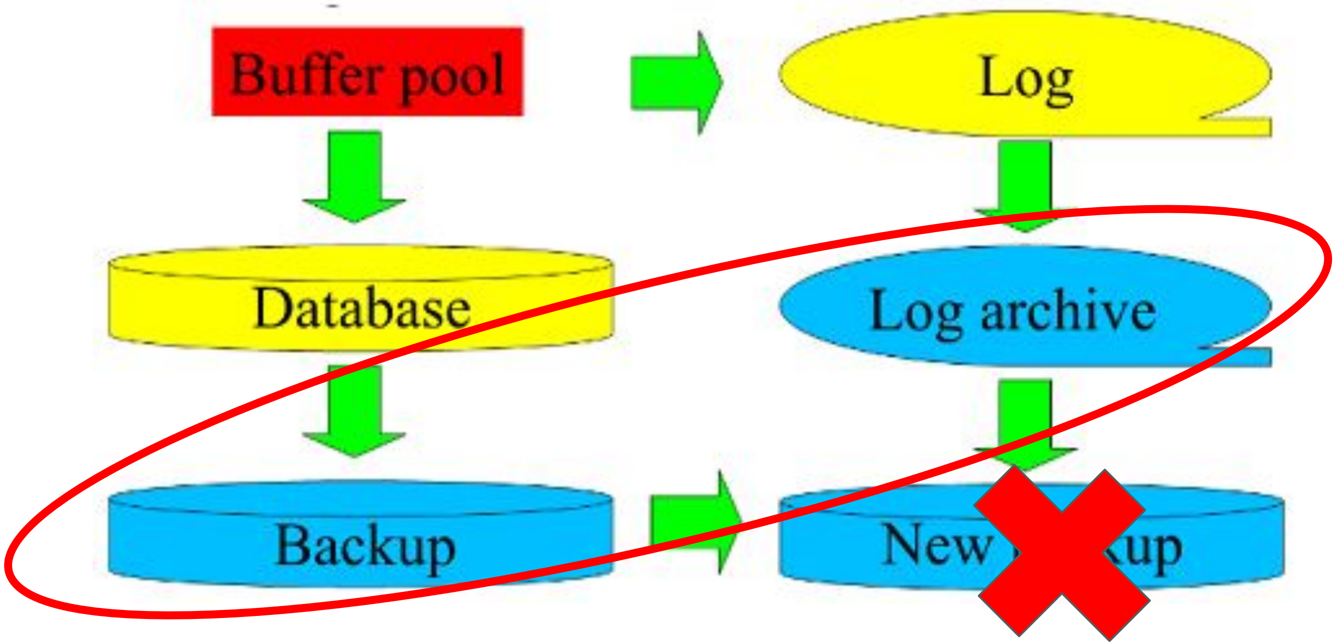
Backup without network load, spike, provisioning



Instant backup

Backup + log archive = single-phase restore

⇒ equivalent to a fresh backup!



Agenda

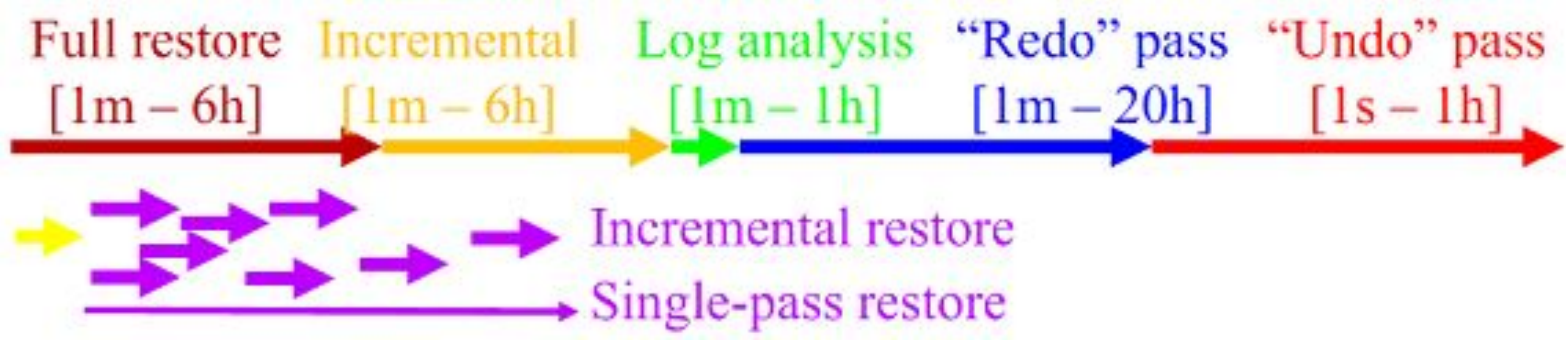
- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, and media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - **Instant** restart, **restore**, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Instant restore

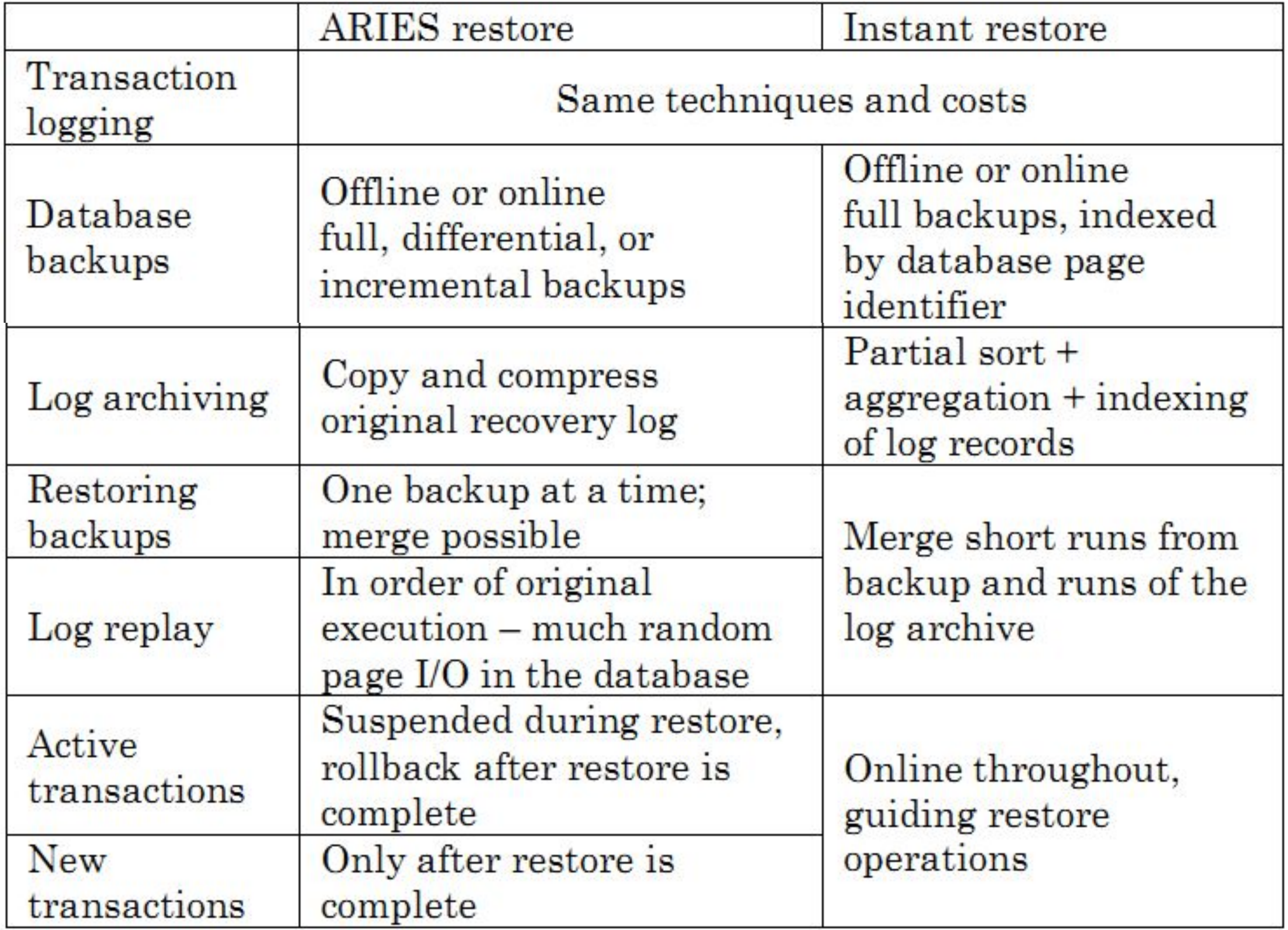
Single-page repair – incremental and on-demand

- Storage device divided into many (1K-1M) segments
- Database backup and log archive (partitions) sorted & indexed

Single-phase restore – efficient and in the background



ARIES restore vs instant restore



	ARIES restore	Instant restore
Transaction logging	Same techniques and costs	
Database backups	Offline or online full, differential, or incremental backups	Offline or online full backups, indexed by database page identifier
Log archiving	Copy and compress original recovery log	Partial sort + aggregation + indexing of log records
Restoring backups	One backup at a time; merge possible	Merge short runs from backup and runs of the log archive
Log replay	In order of original execution – much random page I/O in the database	
Active transactions	Suspended during restore, rollback after restore is complete	Online throughout, guiding restore operations
New transactions	Only after restore is complete	

Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, and media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - **Instant** restart, restore, **reboot**
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Failure & recovery for in-memory databases

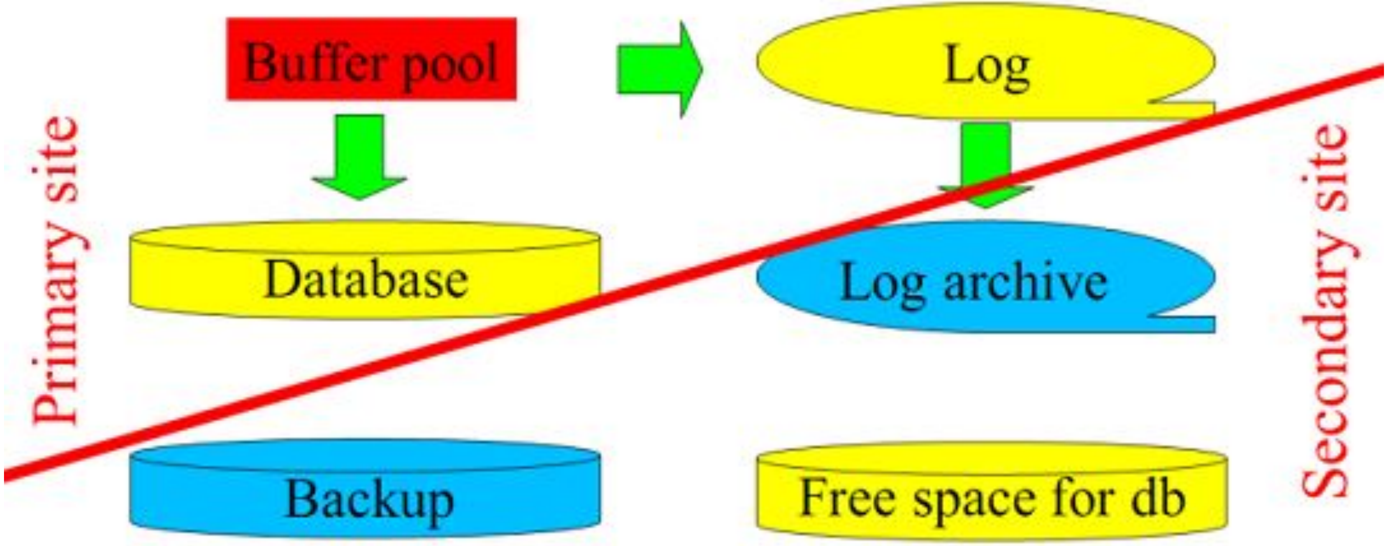
- System failure implies media failure
- Reboot = restart + restore
- Double failure ⇒ double recovery
  - Log analysis
  - Restore from backup + log archive: instant + single-phase
- New transactions & new checkpoints during recovery

Agenda

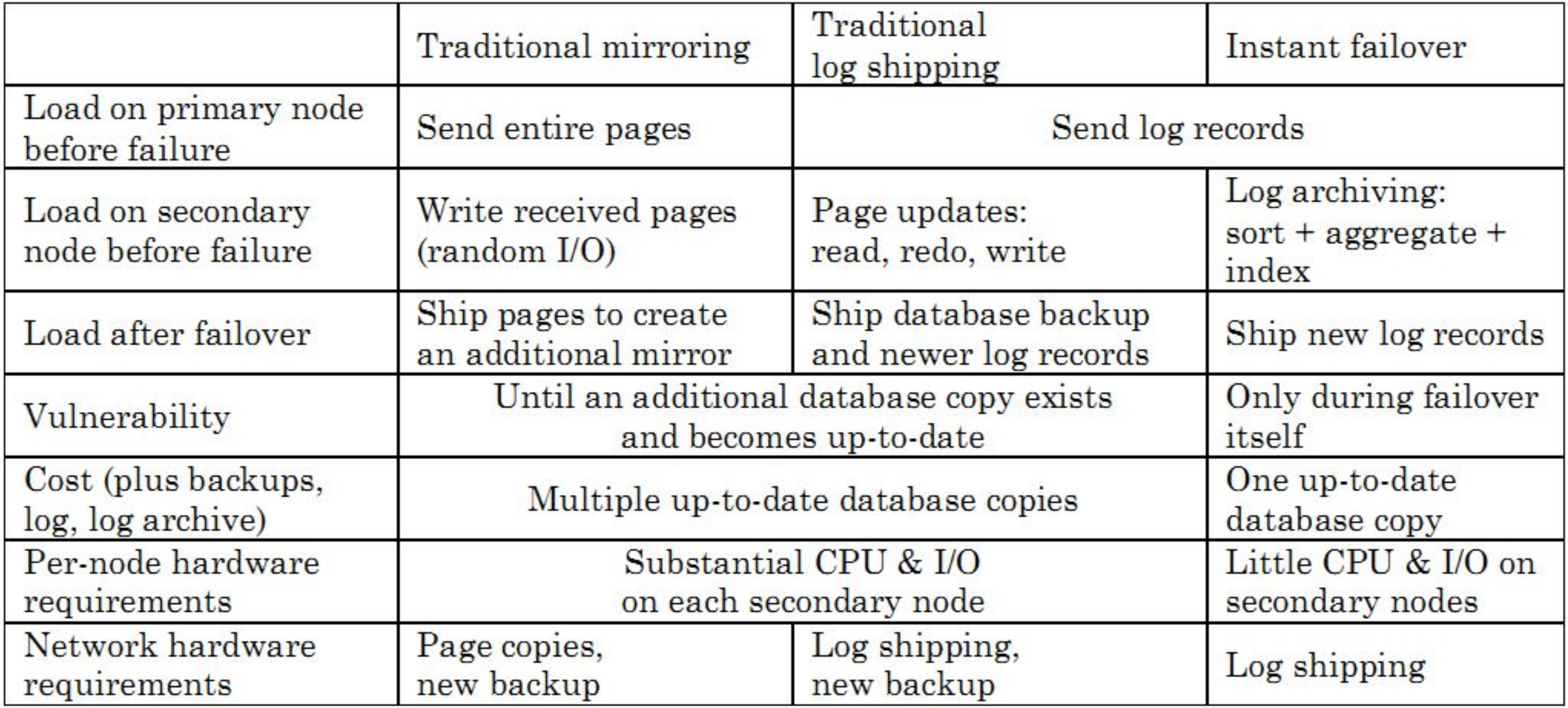
- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, and media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - **Instant failover**, fail-back
- Conclusions

Instant failover (to cold stand-by)

Instant restore + instant restart @ the secondary site



Comparison of techniques for high availability



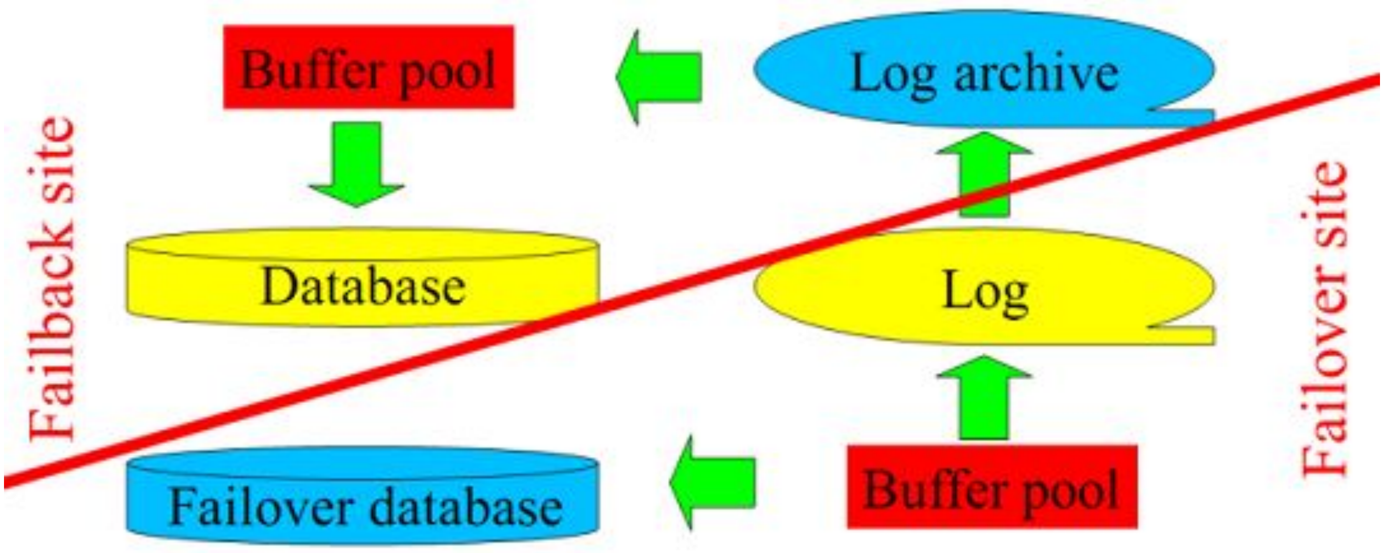
	Traditional mirroring	Traditional log shipping	Instant failover
Load on primary node before failure	Send entire pages	Send log records	
Load on secondary node before failure	Write received pages (random I/O)	Page updates: read, redo, write	Log archiving: sort + aggregate + index
Load after failover	Ship pages to create an additional mirror	Ship database backup and newer log records	Ship new log records
Vulnerability	Until an additional database copy exists and becomes up-to-date		Only during failover itself
Cost (plus backups, log, log archive)	Multiple up-to-date database copies		One up-to-date database copy
Per-node hardware requirements	Substantial CPU & I/O on each secondary node		Little CPU & I/O on secondary nodes
Network hardware requirements	Page copies, new backup	Log shipping, new backup	Log shipping

Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, and media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - **Instant** failover, **fail-back**
- Conclusions

Instant fail-back after repair of primary site

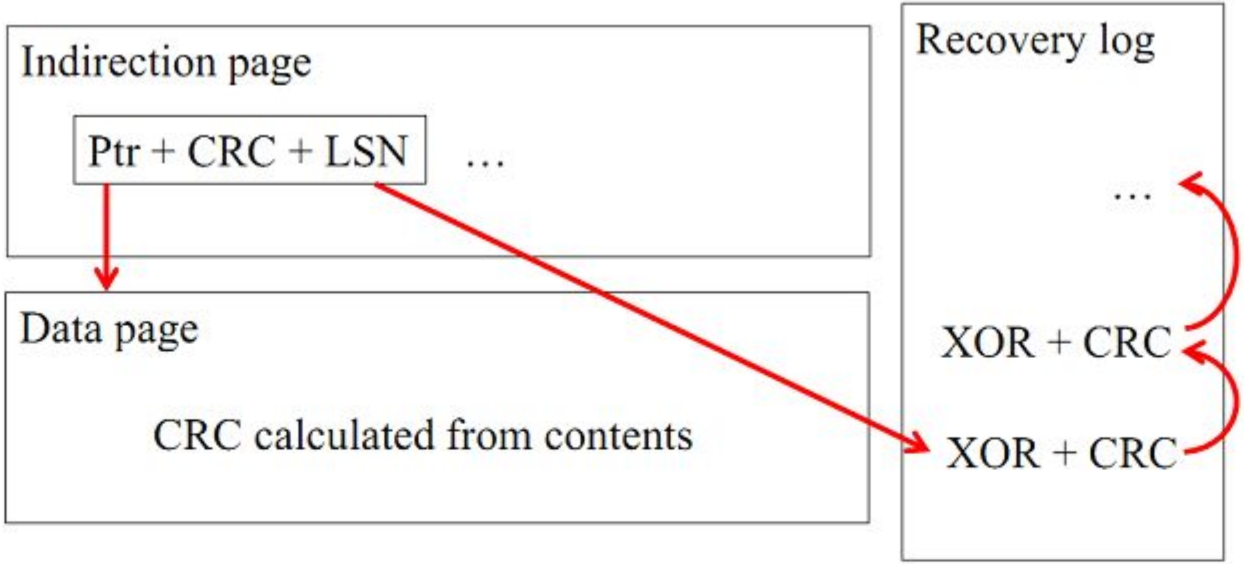
Instant restart @ the original primary site



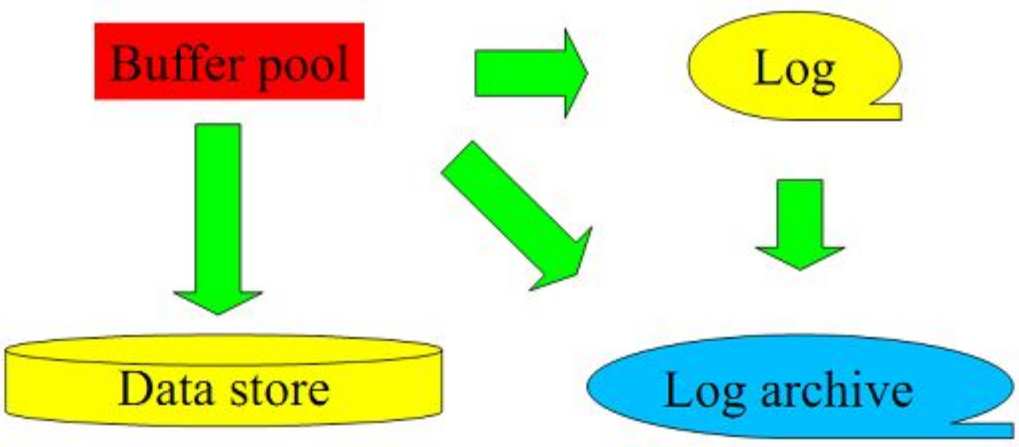
Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, and media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- **Conclusions**

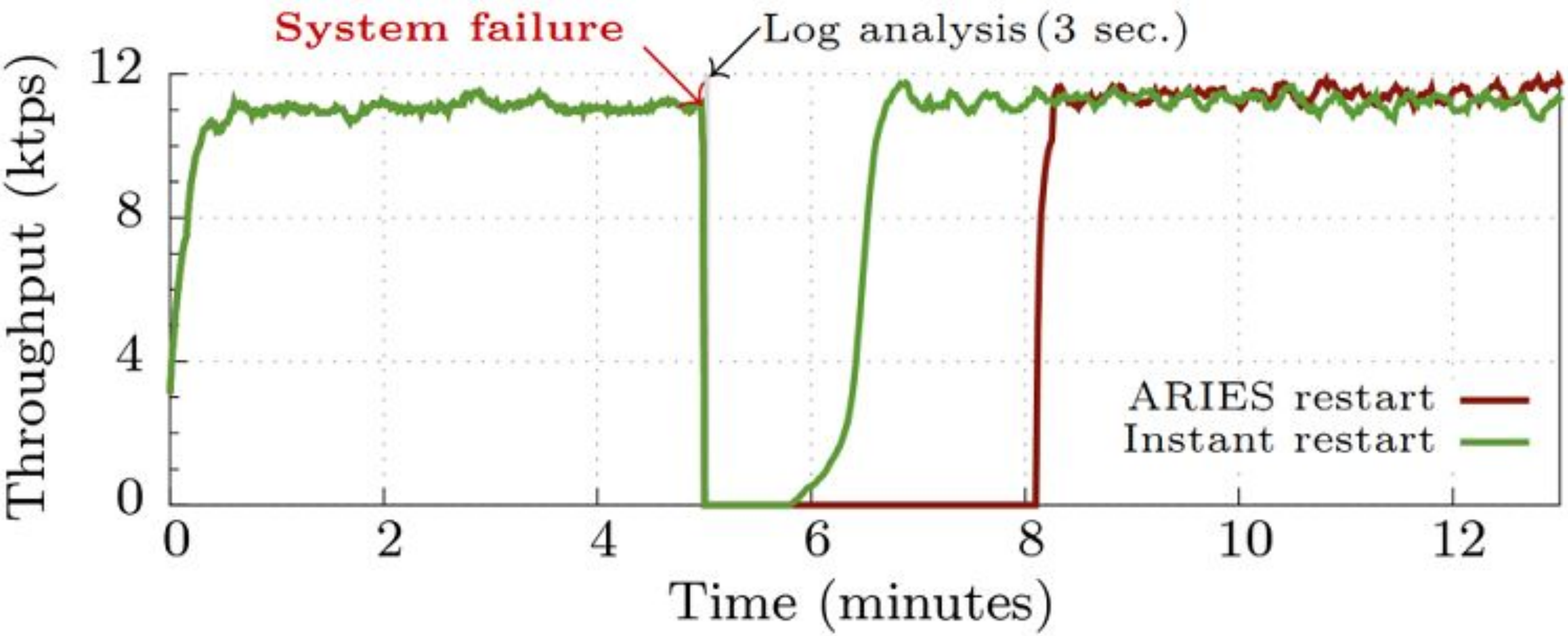
Write-ahead logging ⇒ self-repairing file contents



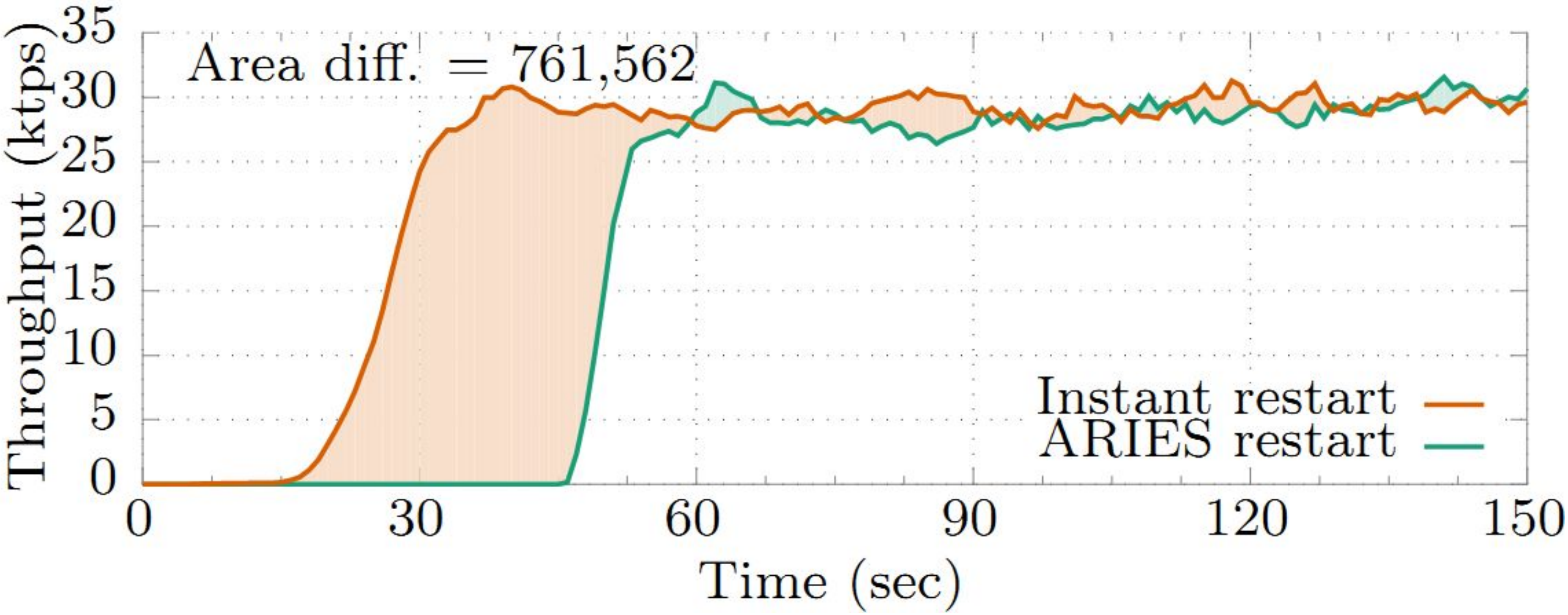
Direct archiving for large file contents



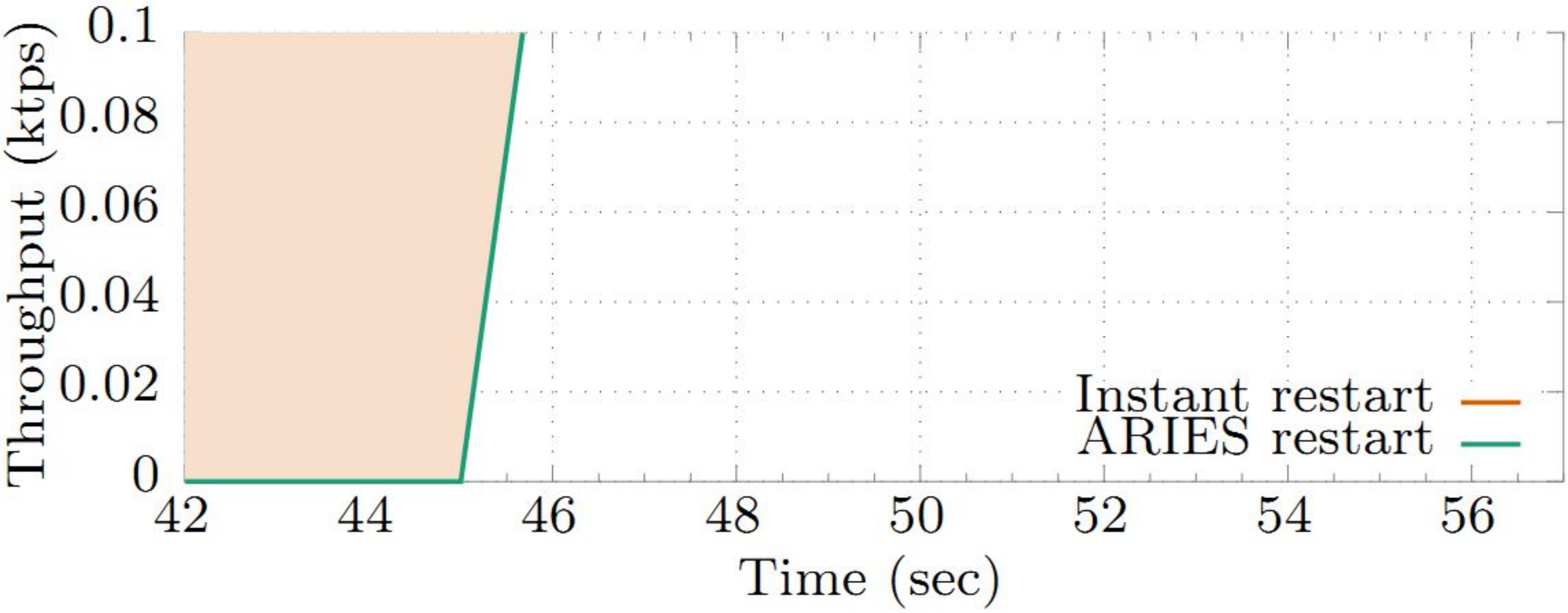
Performance of instant restart



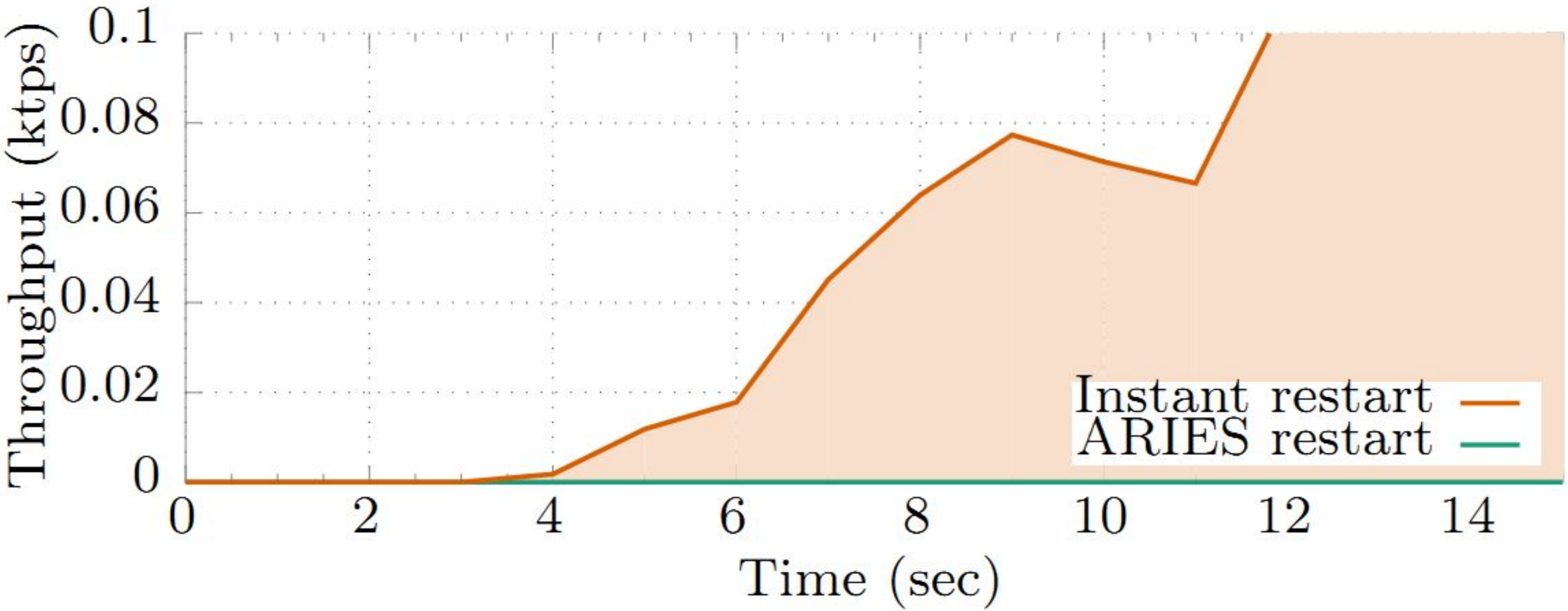
Instant restart



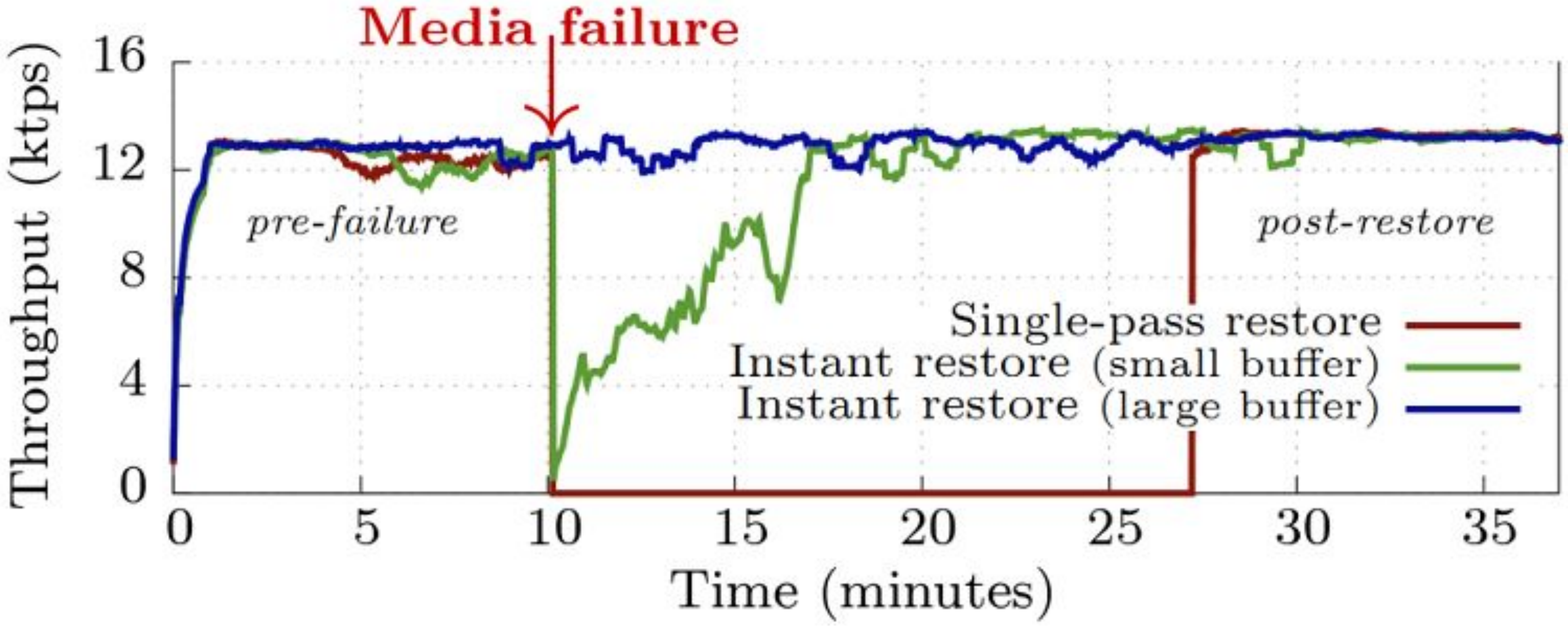
ARIES restart – zoom



Instant restart – zoom



Performance of instant restore



Agenda

- Write-ahead logging
  - Commit, backups, archives, checkpoints
- Classic failures & recovery techniques
  - Transaction, system, and media failures; double failures; failover
- Single-page failures
  - On-demand local repair: self-repairing indexes
- Instant recovery
  - Instant restart, restore, reboot
  - Single-phase restore, instant backup
  - Instant failover, fail-back
- Conclusions

Thank you!