TRANSACTIONAL FLASH

Vijayan Prabhakaran, Thomas L. Rodeheffer, Lidong Zhou

CARSTEN WEINHOLD

MOTIVATION

- Transactions have proven useful:
  - File systems
  - Databases systems
- Common approaches:
  - Copy on write
  - Write ahead logging
- Hard to get right
- Everybody reinvents the wheel ...

BETTER DISKS?

- Transaction support could be built into disk
- Much simpler file systems / databases
- Problem:
  - Copy on write causes fragmentation
  - Slow seeks needed when reading
- Solutions:
  - Reorganize data in cleaning process
  - Checkpointing + update home location

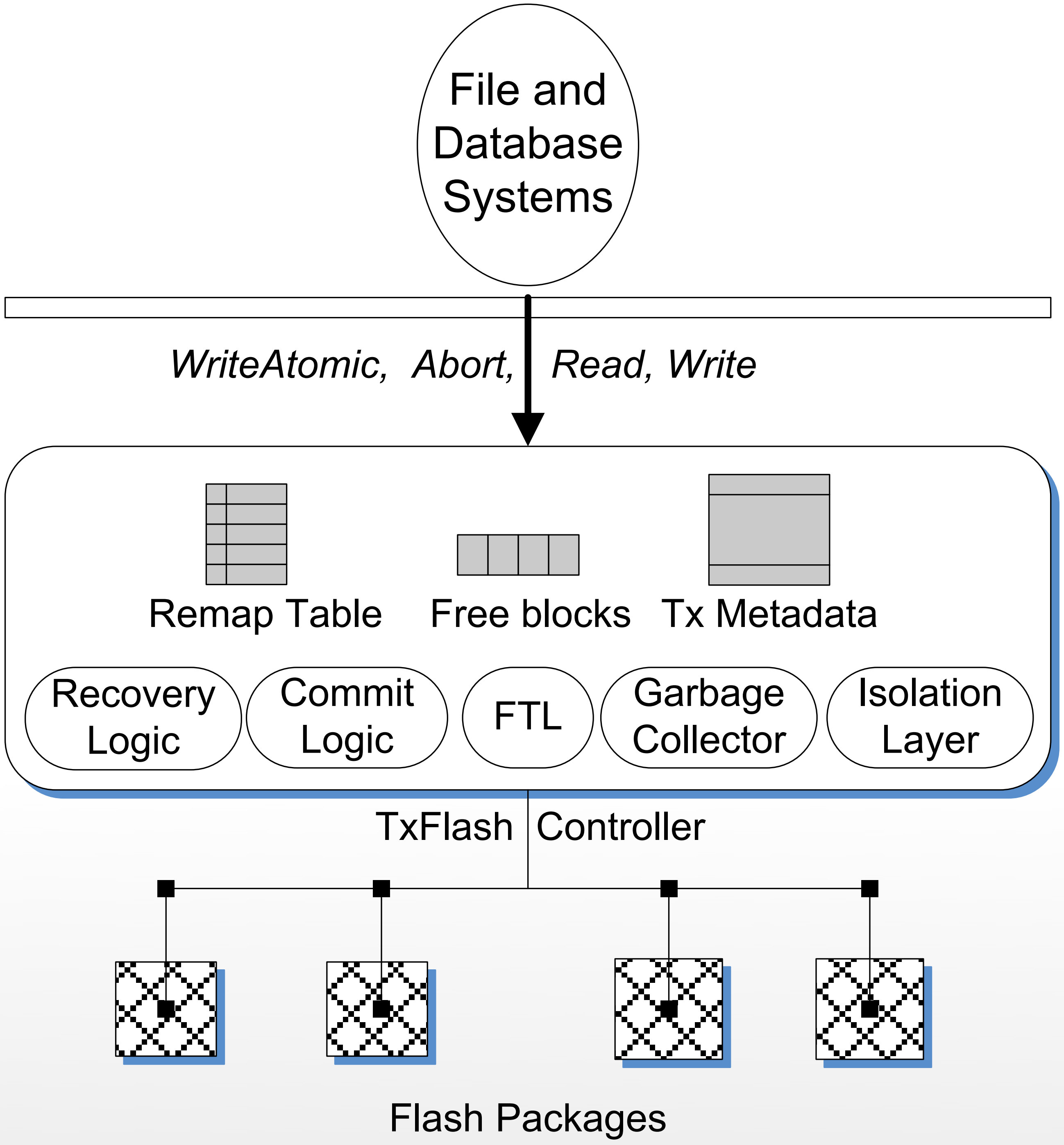
FLASH BASICS

- Typical solid state disk:
  - Controller + multiple flash packages
  - Small mount of RAM to buffer I/O requests, internal data structures
- Data organization:
  - Packages contain planes, blocks, pages
  - 128 bytes of metadata for each 4 KB page
  - Spare memory for data from damaged areas

HOW IT WORKS

- Random reads / writes are fast
- Overwriting is slow:
  - Entire block must be erased
  - Takes in the order of milliseconds
  - Limited number of erase / write cycles
- Out-of-place updates avoid overwriting
- Garbage collection reclaims old pages
- Wear leveling minimizes per-block erasure

TXFLASH


File and Database Systems
WriteAtomic, Abort, Read, Write
Remap Table
Free blocks
Tx Metadata
Recovery Logic
Commit Logic
FTL
Garbage Collector
Isolation Layer
TxFlash Controller
Flash Packages

TXFLASH

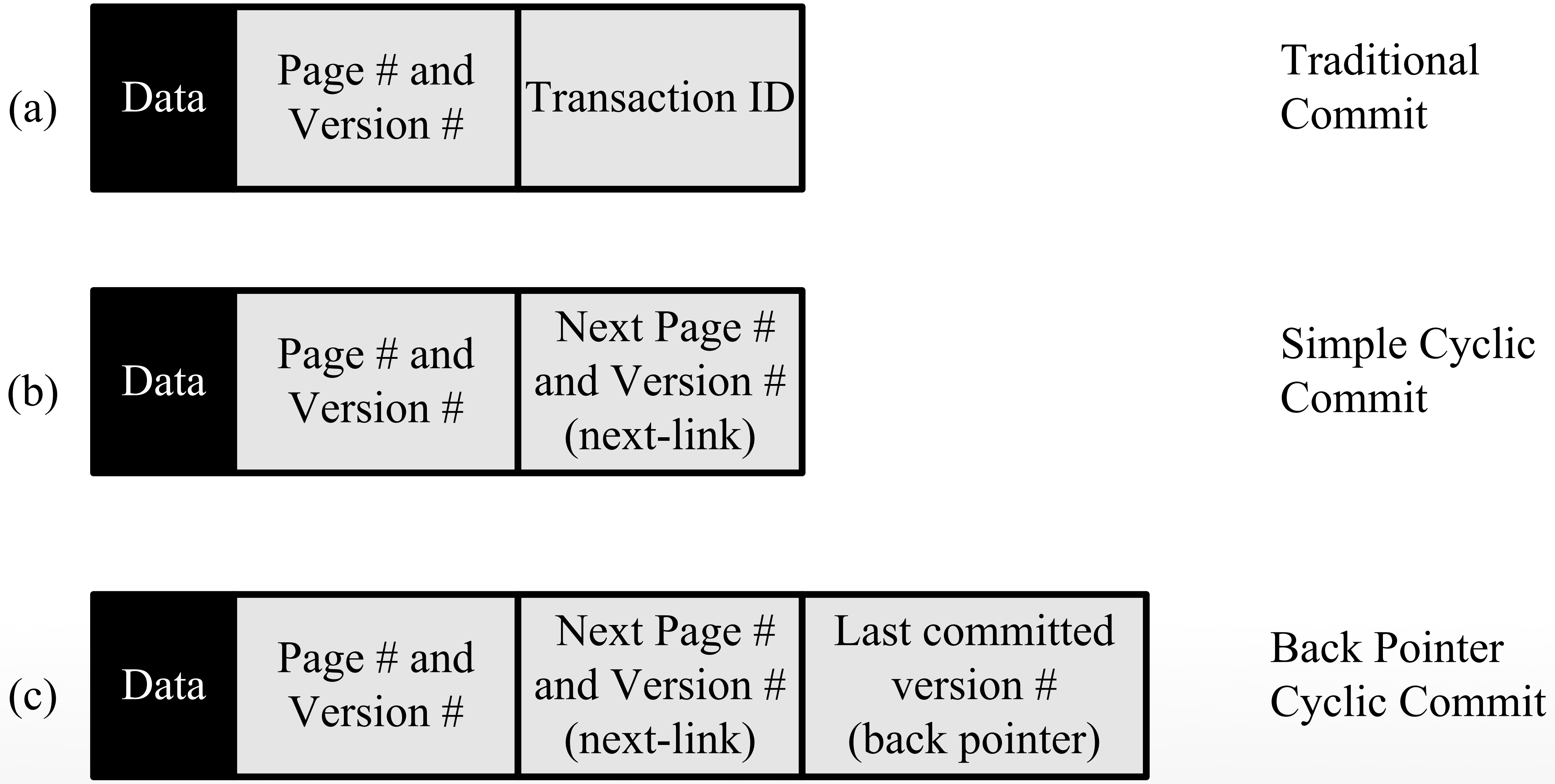
- TxFlash builds on top of existing flash storage controllers
- Introduces additional command:

WriteAtomic(p_0, p_1, ... p_{n-1})

REDO LOGGING

- Most file systems use redo logging:
  - Intention records written to storage:
    - Pages with data
    - Metadata describing location, etc.
  - Extra write for commit record, after intention records are persistent
  - Data from log copied to home locations in checkpoint process
- Recovery: redo committed transactions

TXN PROTOCOLS



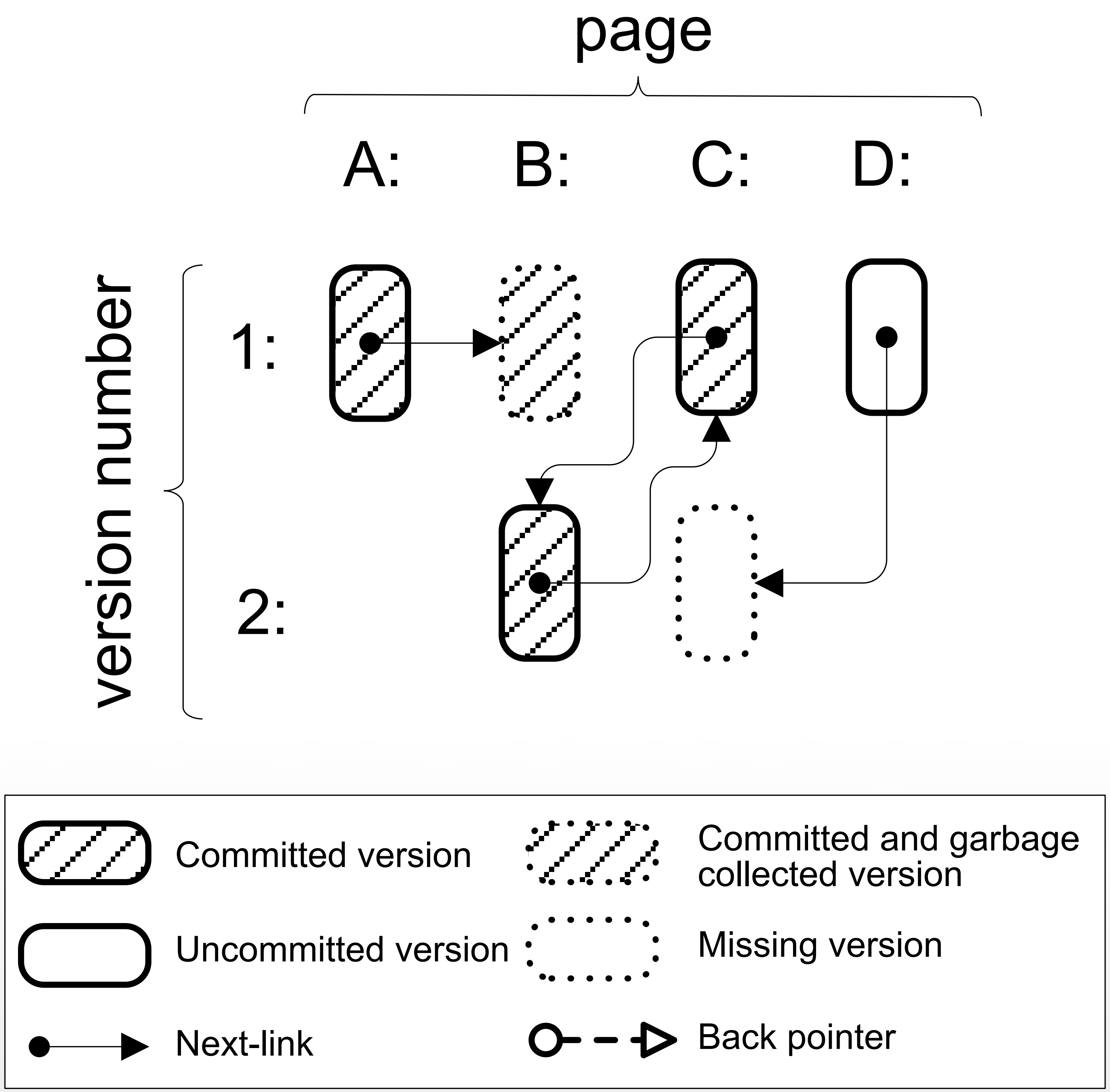
(a) Data | Page # and Version # | Transaction ID — Traditional Commit

(b) Data | Page # and Version # | Next Page # and Version # (next-link) — Simple Cyclic Commit

(c) Data | Page # and Version # | Next Page # and Version # (next-link) | Last committed version # (back pointer) — Back Pointer Cyclic Commit

CYCLIC COMMIT

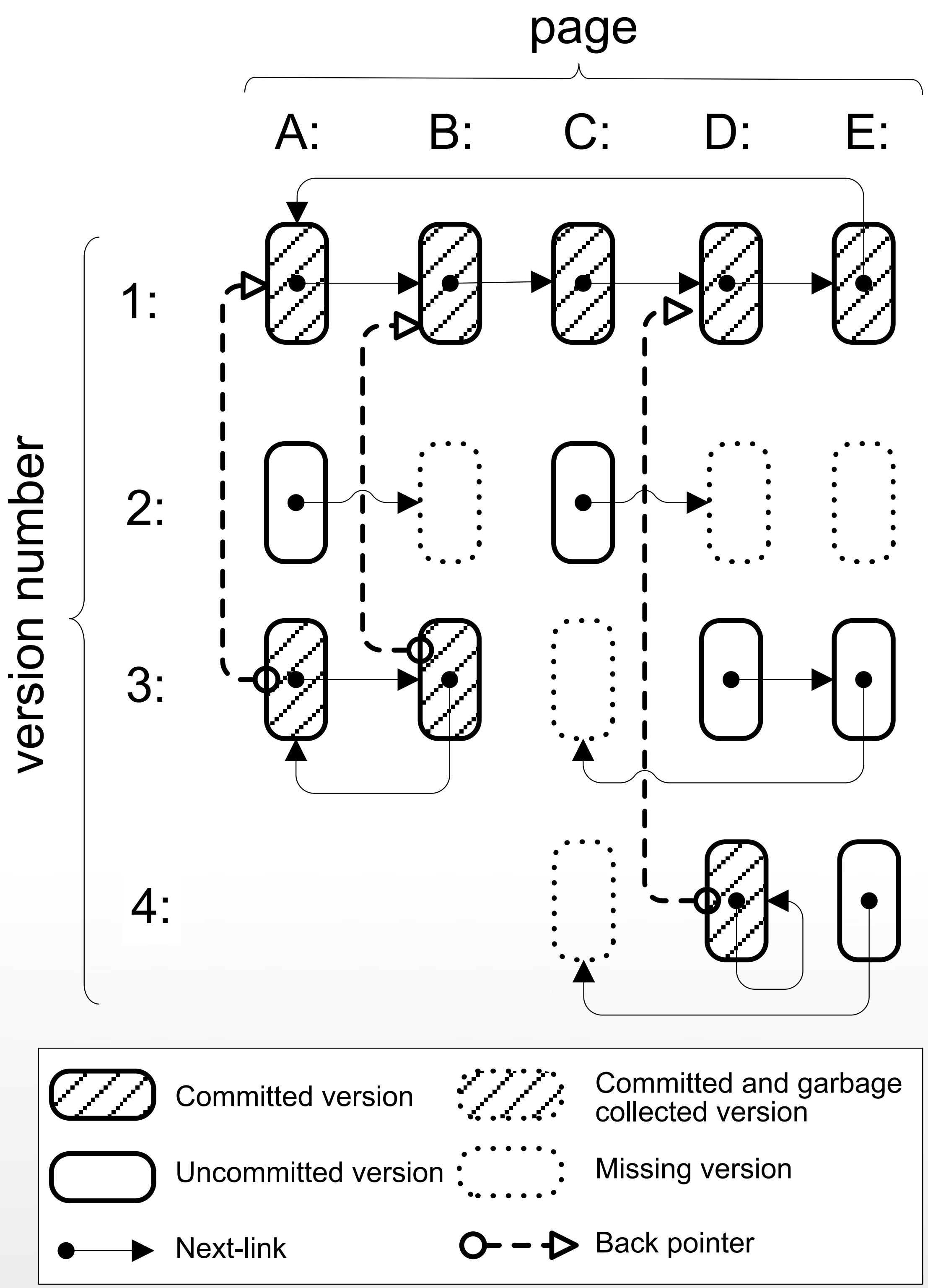
- Requirement: data + metadata can be stored together efficiently
- No extra write for commit record:
  - Each intention record has next link
  - Last intention record points to first one
  - Concurrent writes possible for all records
  - Recovery: full cycle in storage describes committed transaction


page
A:
B:
C:
D:
version number
1:
2:
Committed version
Committed and garbage collected version
Uncommitted version
Missing version
Next-link
Back pointer

SCC VS. BPCC

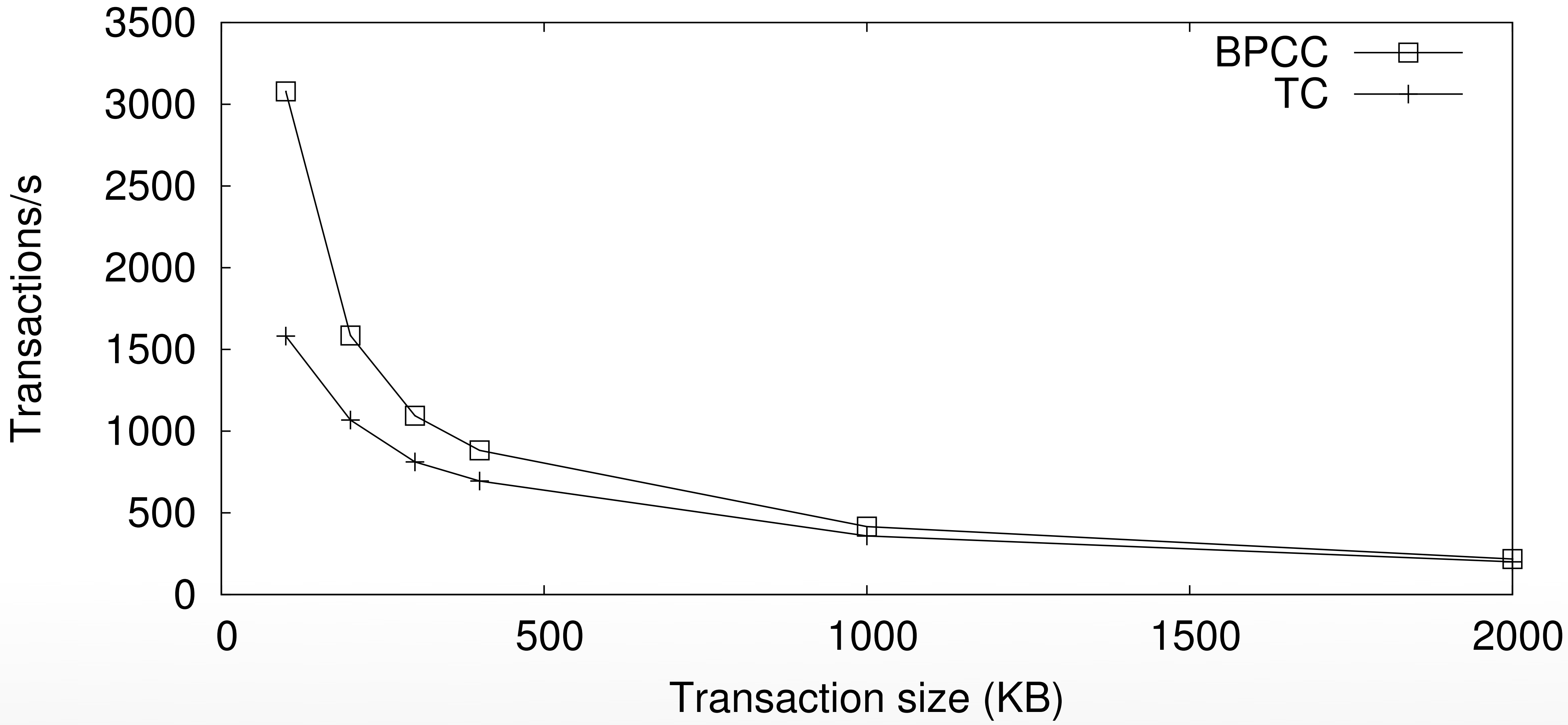
- Simple Cyclic Commit:
  - No overlapping transactions (isolation)
  - Uncommited intention records must be erased before starting new transaction
- Back Pointer Cyclic Commit:
  - Extra back pointer: last committed transaction
  - Avoids erase cycle after aborted transaction
  - More complex garbage collection / recovery

BPCC

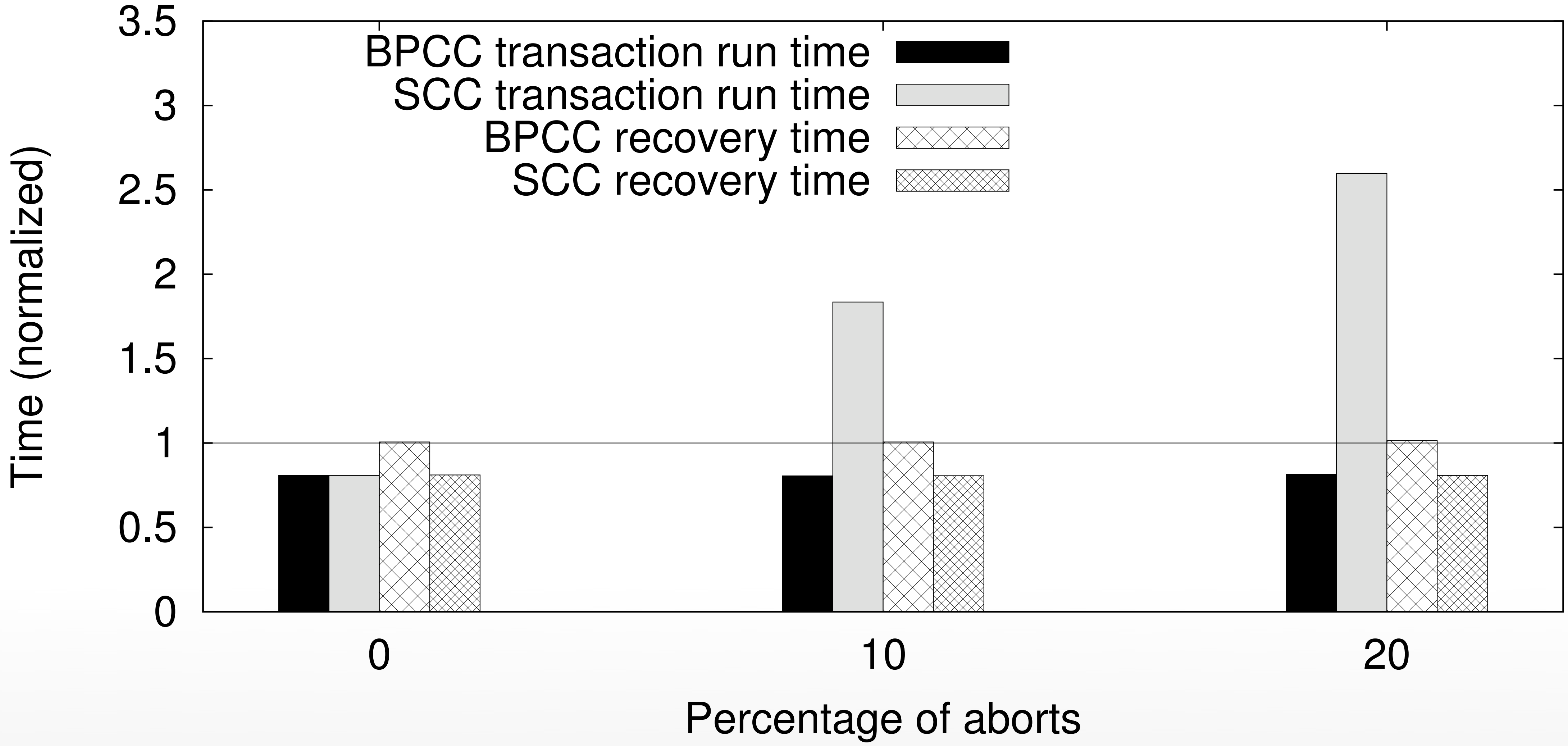

page
A:
B:
C:
D:
E:
version number
1:
2:
3:
4:
Committed version
Committed and garbage collected version
Uncommitted version
Missing version
Next-link
Back pointer

EVALUATION

TXN SIZE

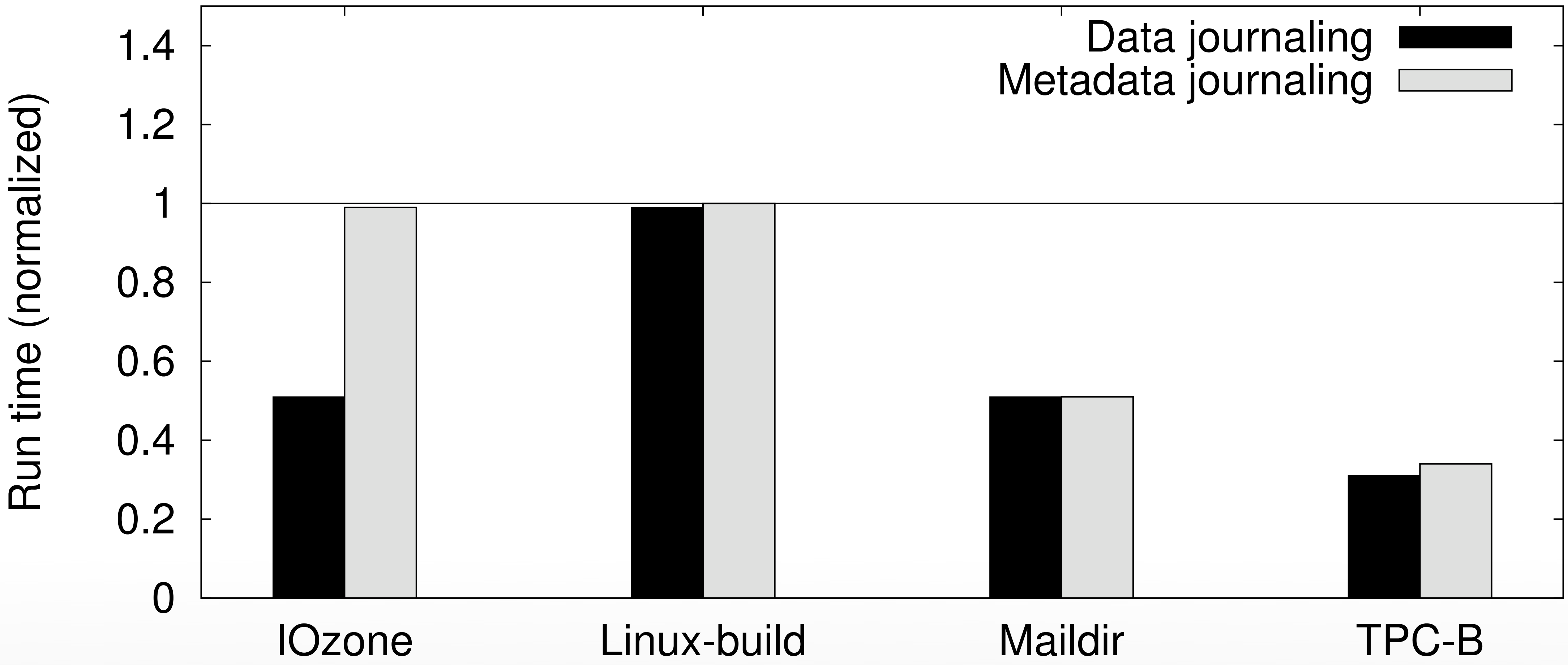

3500
3000
2500
2000
1500
1000
500
0
Transactions/s
0
500
1000
1500
2000
Transaction size (KB)
BPCC □
TC +

ABORTED TXNS



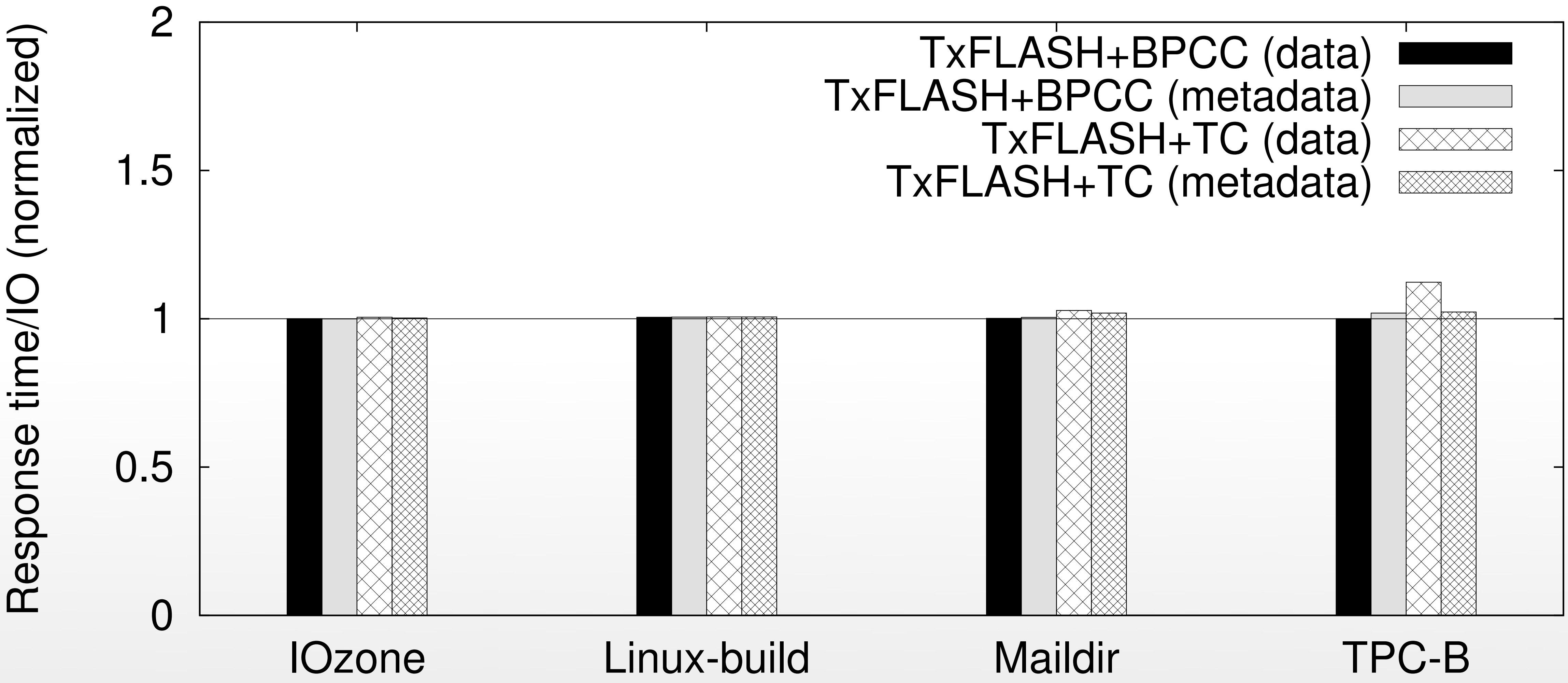
TECHNISCHE UNIVERSITÄT DRESDEN

PERFORMANCE



COSTS

		TxFlash		
	SSD	+TC	+SCC	+BPCC
LOC	7621	9094	9219	9495

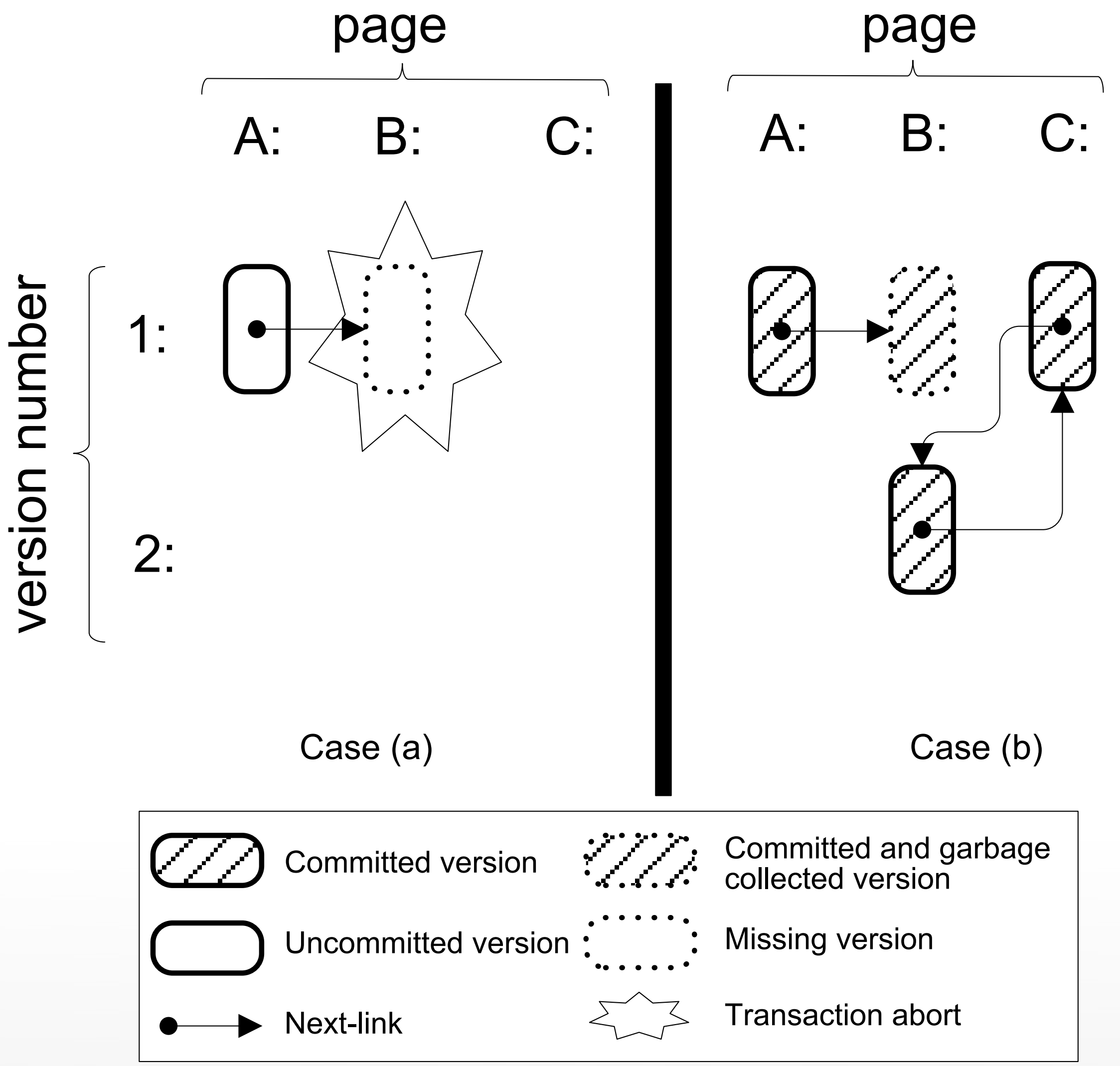


DISCUSSION

- Model-checked! Cool!
- Is the cyclic commit protocol really new?
- File systems do not cancel transactions. Do we really need BPCC?
- Databases cancel transactions, but TxFlash is not fit for them yet. Will BPCC still suffice?
- Are page writes atomic?

BACKUP SLIDES

ABORTED TXNS


page
A:
B:
C:
version number
1:
2:
Case (a)
page
A:
B:
C:
Case (b)
Committed version
Committed and garbage collected version
Uncommitted version
Missing version
Next-link
Transaction abort