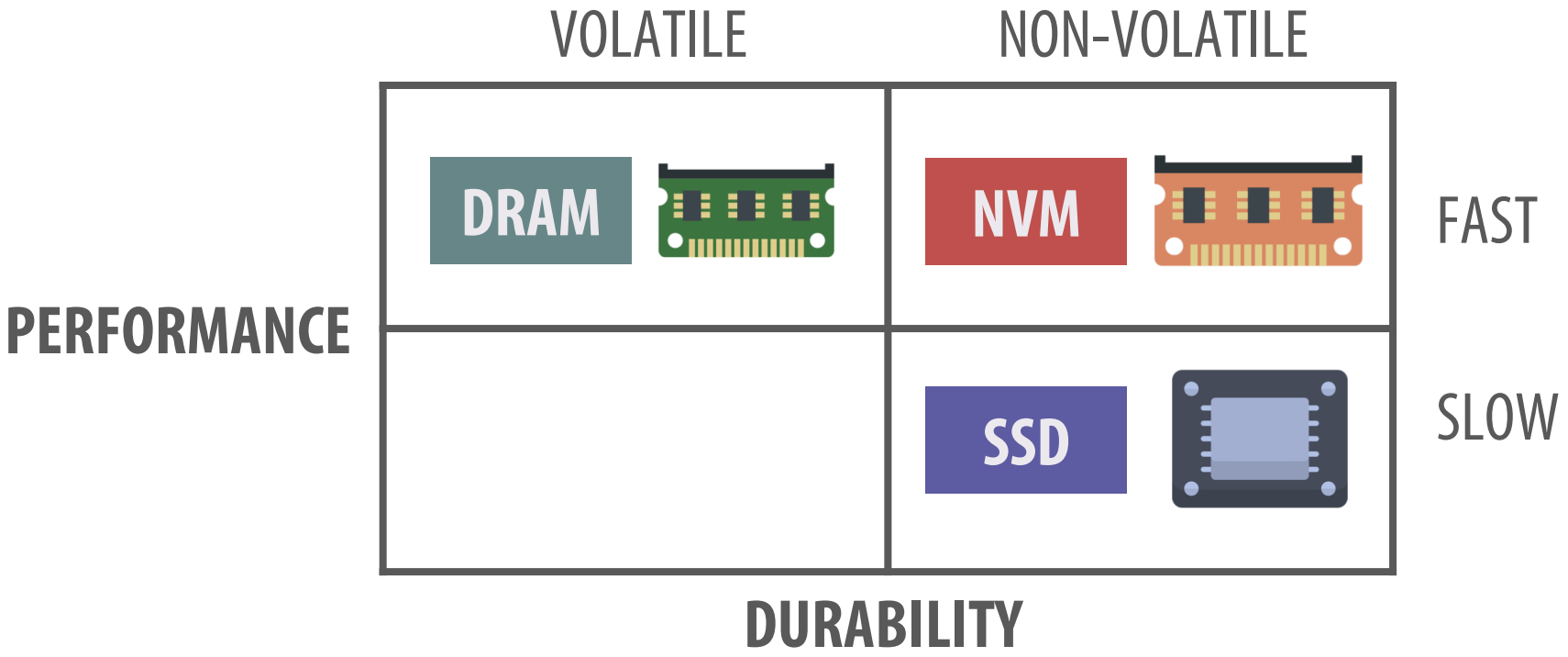
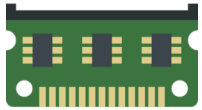
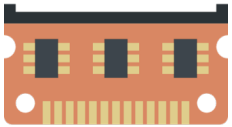
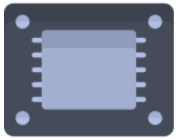
BzTree: A High-Performance Latch-free Range Index for Non-Volatile Memory

JOY ARULRAJ
JUSTIN LEVANDOSKI
UMAR FAROOQ MINHAS
PER-AKE LARSON
Microsoft Research

NON-VOLATILE MEMORY [NVM]

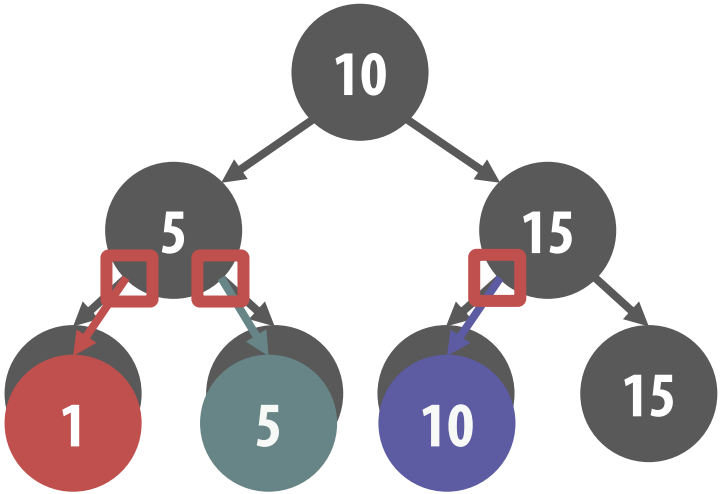


DEVICE CHARACTERISTICS

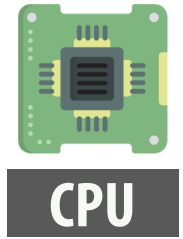




CHARACTERISTIC	DRAM	NVM	SSD
Device Latency	1x	10x	1000x
Byte-Addressability	✓	✓	✗
Durability	✗	✓	✓
High Capacity	✗	✓	✓

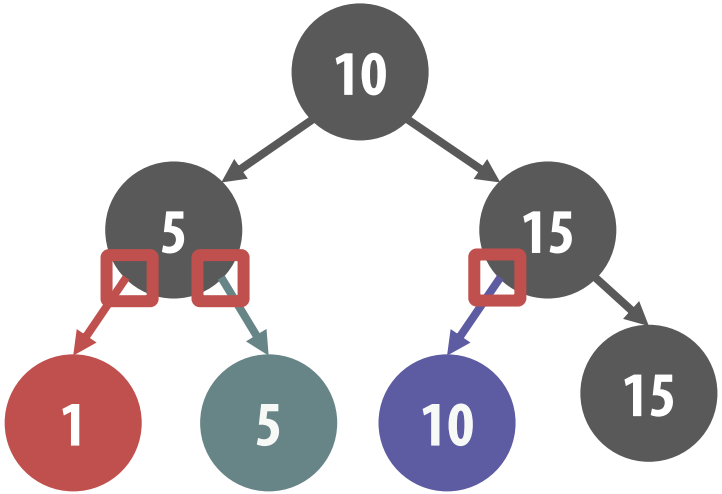
BWTREE: LATCH-FREE B+TREE



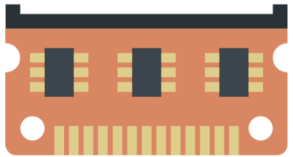
SINGLE-WORD
COMPARE-AND-SWAP
INSTRUCTION



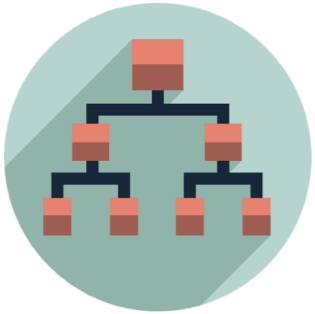
BZTREE: NVM-CENTRIC LATCH-FREE B+TREE



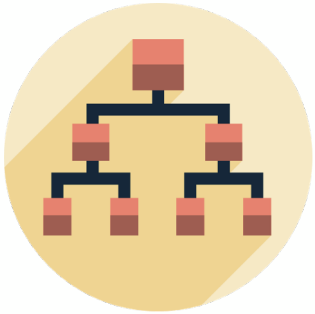
LATCH-FREE
B+TREE



NON-VOLATILE
MEMORY



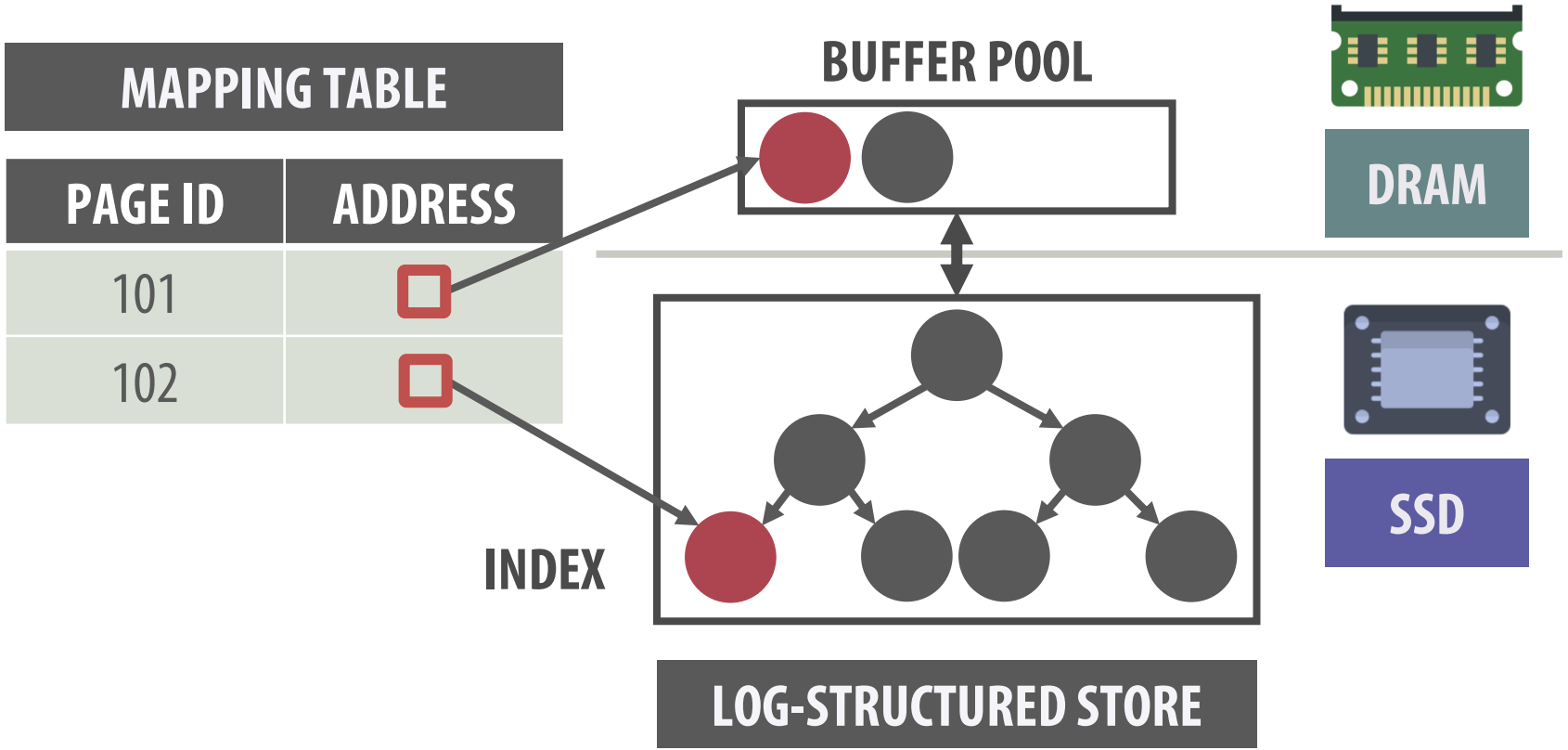
BWTREE
INDEX



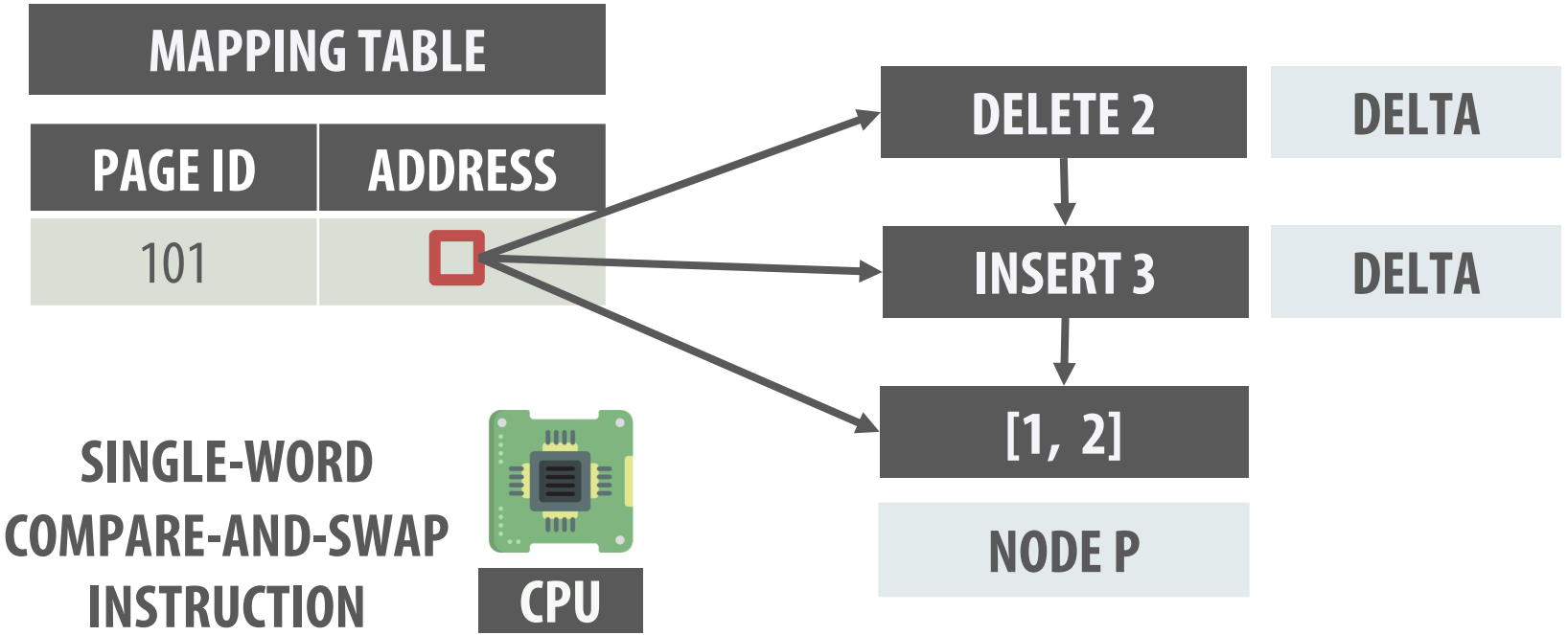
BZTREE
INDEX

EXPERIMENTAL
RESULTS

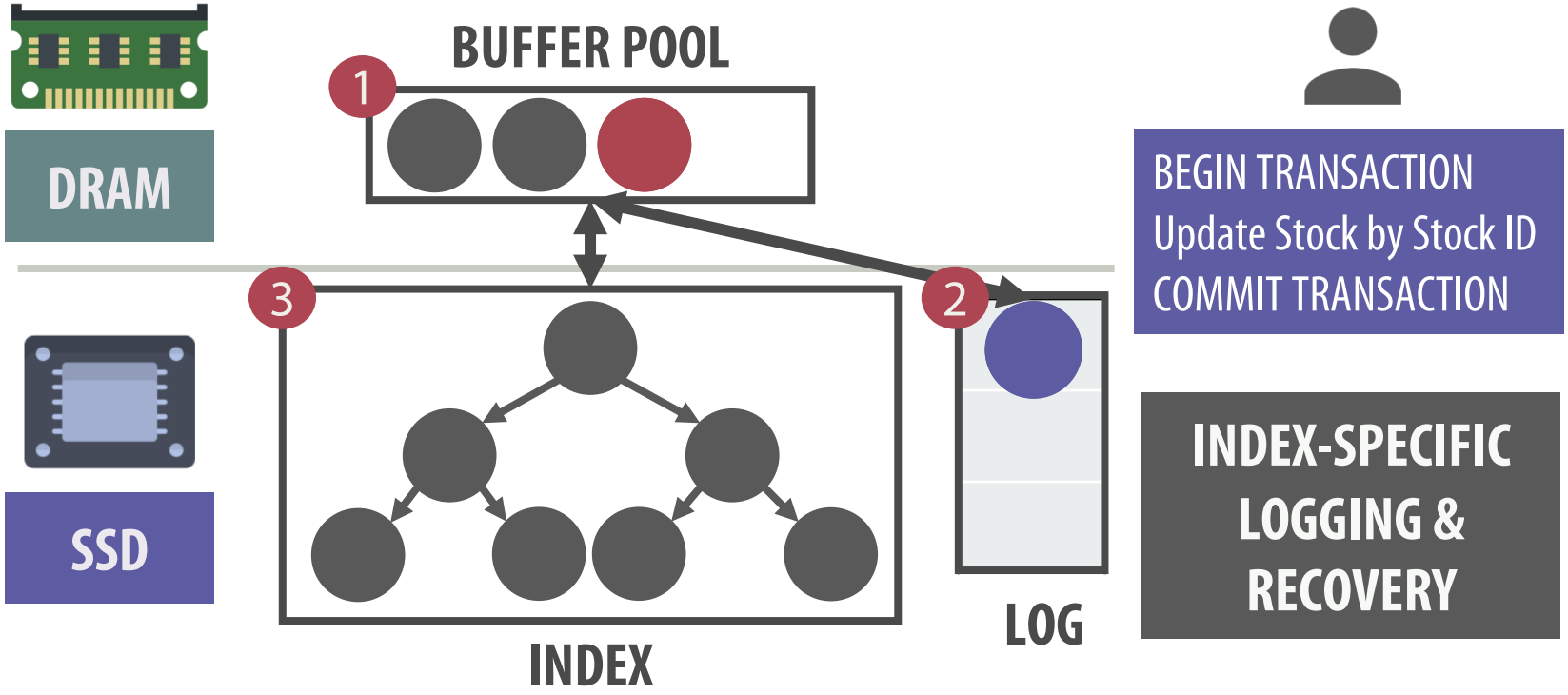
BWTREE: SSD-CENTRIC ARCHITECTURE



BWTREE: LATCH-FREE ALGORITHMS



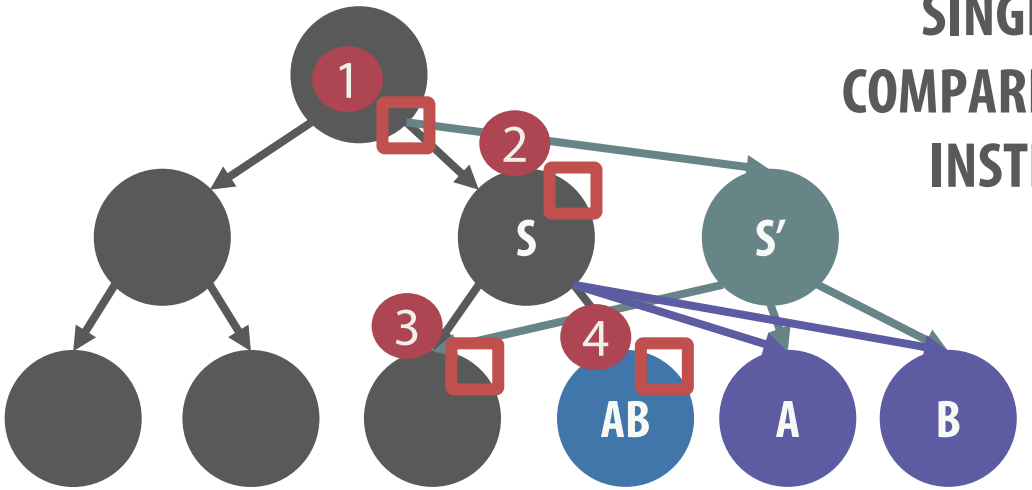
BWTREE: LOGGING & RECOVERY PROTOCOL



BWTREE: RECAP

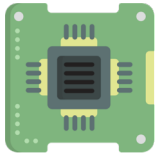
- Delivers high performance on a DRAM + SSD system
  - *SSD-centric **architecture***
  - *Latch-free **algorithms***
  - *Logging & recovery **protocol***
- Limitations
  - *NVM invalidates the key design assumptions of BwTree*
  - *Challenging to design & extend such latch-free data structures*

PROBLEM #1: ALGORITHMIC COMPLEXITY



SPLITTING
A NODE

SINGLE-WORD
COMPARE-AND-SWAP
INSTRUCTION

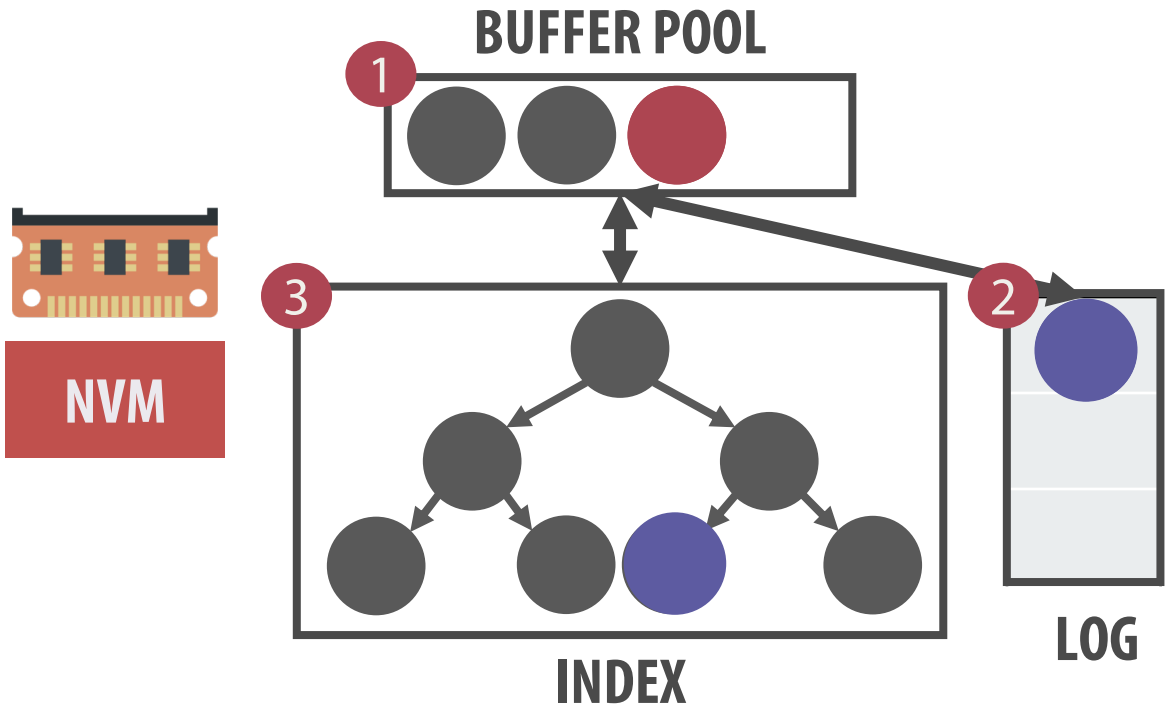


CPU

LATCH-FREEDOM

INTERMEDIATE
STATES

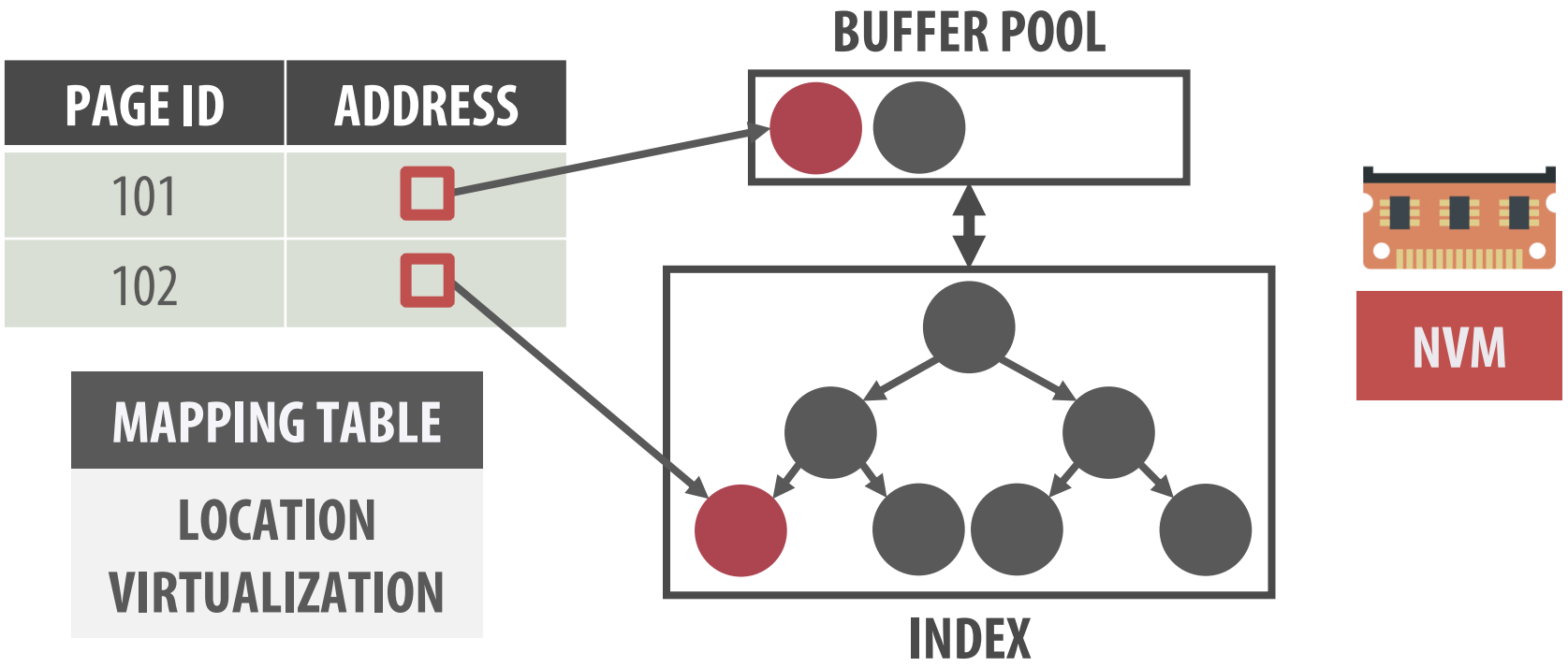
PROBLEM #2: PROTOCOL COMPLEXITY



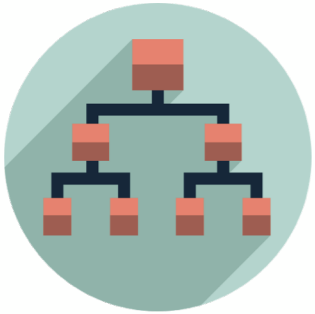
DURABILITY & ATOMICITY

INDEX-SPECIFIC LOGGING & RECOVERY

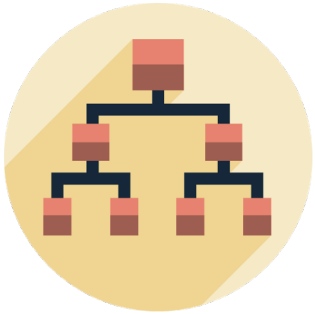
PROBLEM #3: ARCHITECTURAL COMPLEXITY



HOW CAN WE SIMPLIFY LATCH-FREE PROGRAMMING ON NON-VOLATILE MEMORY?



BWTREE
INDEX



BZTREE
INDEX

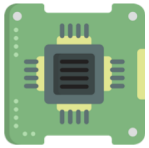
EXPERIMENTAL
RESULTS

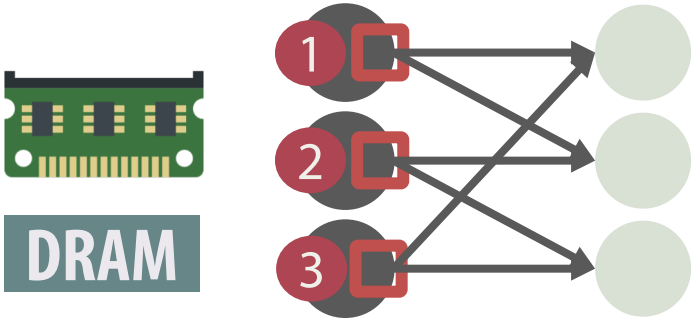
BZTREE: OVERVIEW

- NVM-centric design
  - *Based on a new NVM-centric software primitive*
  - *Provides same guarantees as disk-centric BwTree*
- B**z**Tree supersedes B**w**Tree (skipped B**x**Tree and B**y**Tree)
  - *Because we think that it is the last index you will ever need!*
- Key techniques
  - *Adopt a simpler NVM-centric architecture*
  - *Reduce complexity using software primitive*

NVM-CENTRIC SOFTWARE PRIMITIVE

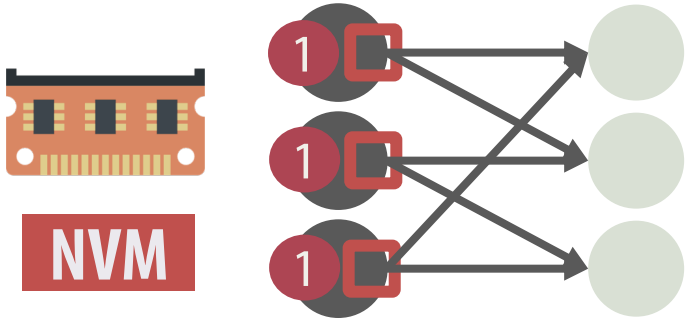
HARDWARE PRIMITIVE





VOLATILE SINGLE-WORD COMPARE-AND-SWAP

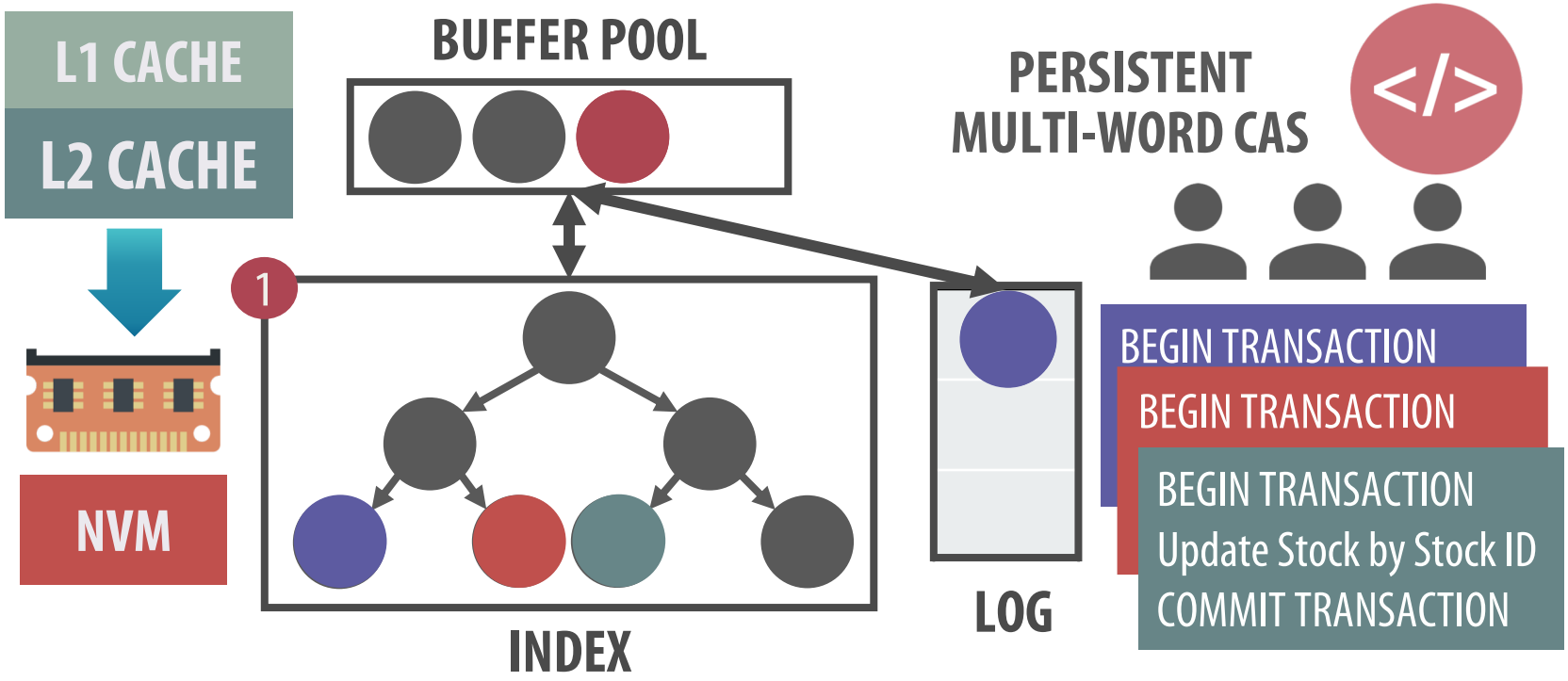
SOFTWARE PRIMITIVE



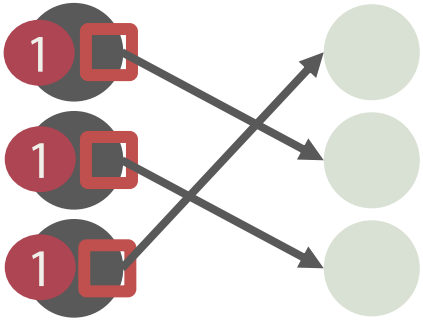
PERSISTENT MULTI-WORD COMPARE-AND-SWAP

EASY LOCK-FREE INDEXING IN NON-VOLATILE MEMORY
ICDE 2018

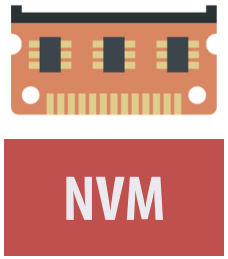
BZTREE: NVM-CENTRIC ARCHITECTURE



BZTREE: DURABILITY & ATOMICITY



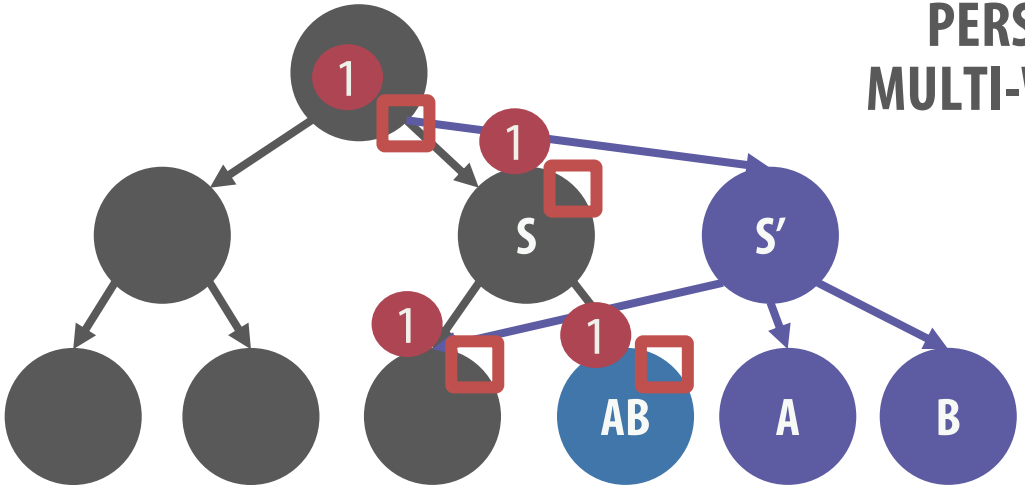
PERSISTENT MULTI-WORD CAS



OPERATION TABLE

LOCATION	EXPECTED OLD VALUE	NEW VALUE	FLUSHED
0x100	OLD CHILD POINTER	NEW CHILD POINTER	1
0x200	OLD NODE STATUS	NEW NODE STATUS	1
0x300	OLD PARENT POINTER	NEW PARENT POINTER	0

SOLUTION #1: ALGORITHMIC COMPLEXITY

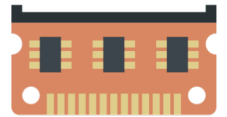


SPLITTING
A NODE

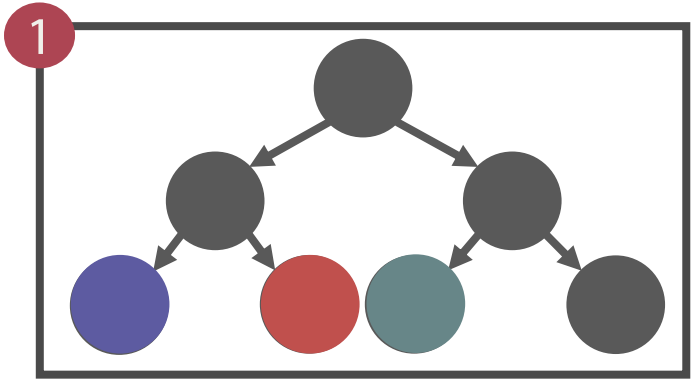
PERSISTENT
MULTI-WORD CAS

EXPONENTIALLY
FEWER
INTERMEDIATE
STATES

SOLUTION #2: PROTOCOL COMPLEXITY



NVM



INDEX

PERSISTENT MULTI-WORD CAS

LOCATION	OLD VALUE	NEW VALUE	FLUSHED
0x100	OLD CHILD POINTER	NEW CHILD POINTER	1
0x200	OLD NODE STATUS	NEW NODE STATUS	1
0x300	OLD PARENT POINTER	NEW PARENT POINTER	0

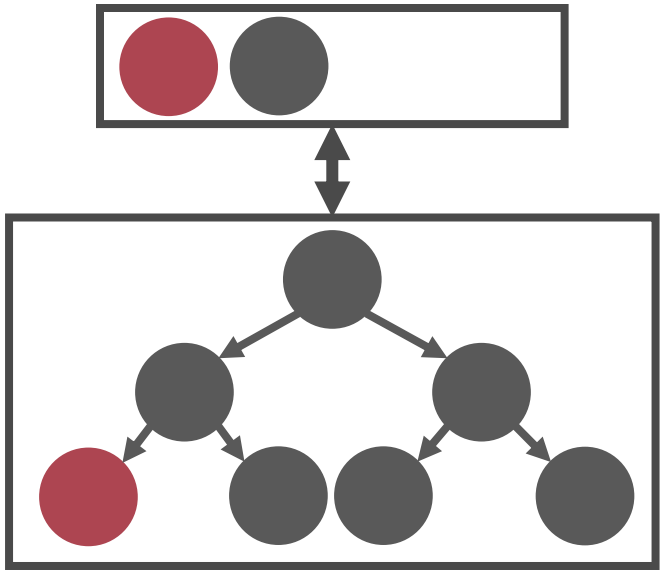
NO INDEX-SPECIFIC PROTOCOL

DURABILITY & ATOMICITY

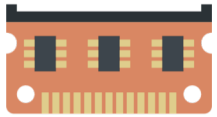
SOLUTION #3: ARCHITECTURAL COMPLEXITY

NO MAPPING TABLE

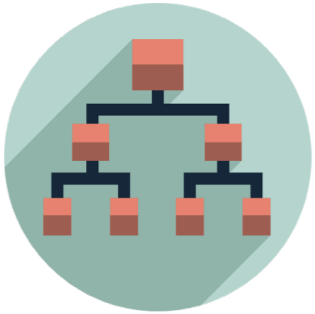
NO DELTA RECORDS & INDIRECTION OVERHEAD



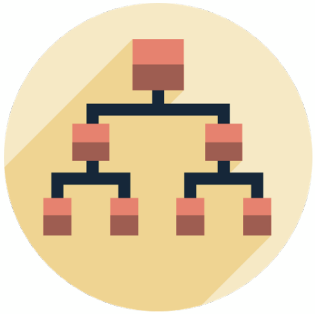
NO LOG STRUCTURED STORE



NVM



BWTREE
INDEX



BZTREE
INDEX

EXPERIMENTAL
RESULTS

EVALUATION

- Index data structures: BzTree vs. BwTree index
  - *Code complexity*
  - *Runtime performance*
  - *Recovery time*
- Benchmark: Yahoo Cloud Serving benchmark
  - *Read-mostly & Balanced workloads*
- Storage device
  - *Emulated Non-Volatile Memory*

CODE COMPLEXITY

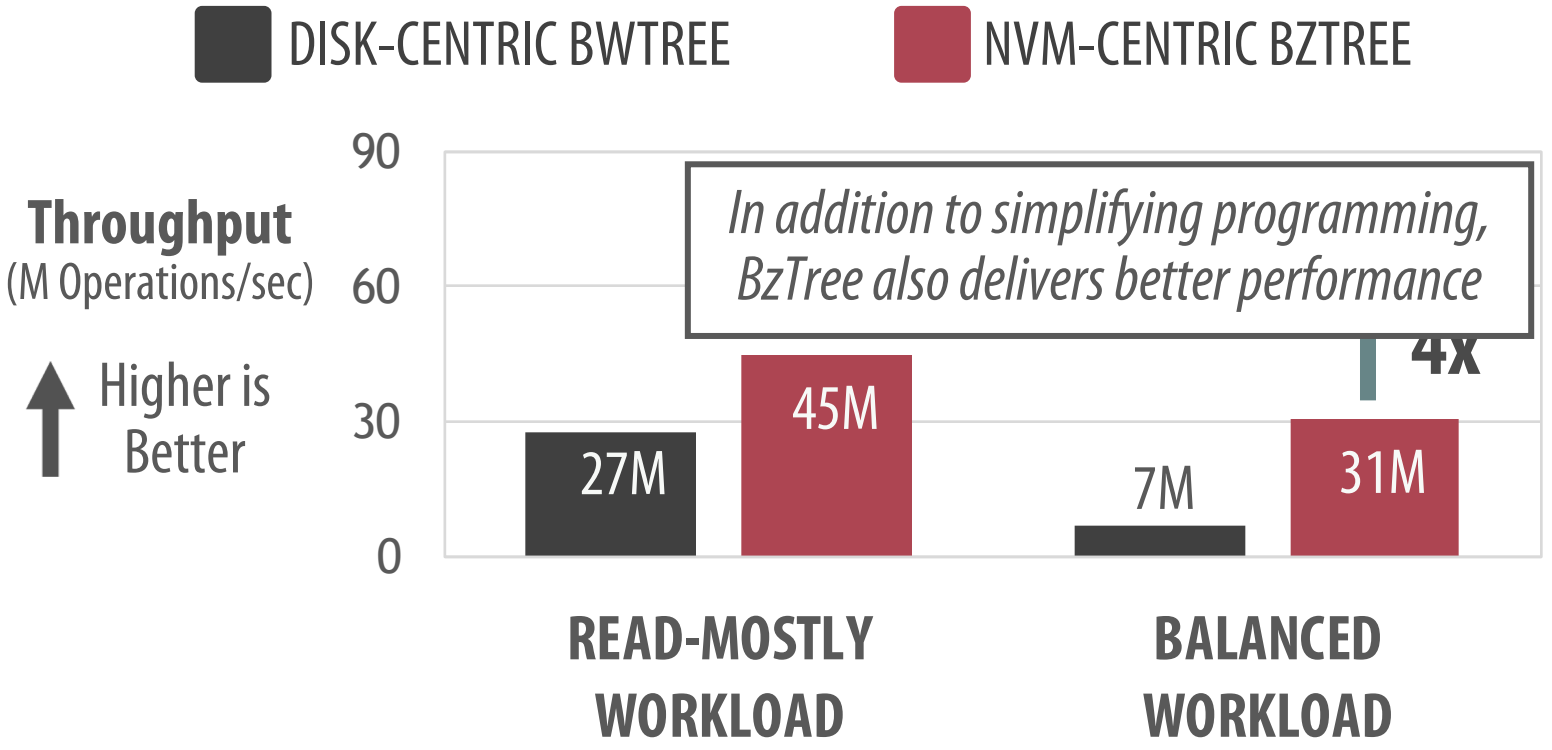
Lower is Better

CODE COMPLEXITY METRIC	BWTREE	BZTREE	
CYCLOMATIC COMPLEXITY	12	7	↓ 2x
LINES OF CODE	750	200	↓ 4x

1. FEWER INTERMEDIATE STATES

2. NO INDEX-SPECIFIC LOGGING PROTOCOL

RUNTIME PERFORMANCE



RECOVERY TIME

- BzTree: no recovery logic
  - *Recovery is entirely handled by software primitive*
  - *Rolls back operations that were in progress during the crash*

Lower is Better

	BWTREE	BZTREE
RECOVERY TIME	~5000 us	145 us

30x

CONCLUSION

- NVM invalidates design assumptions in data structures
- Presented the design of a NVM-centric latch-free B+tree
- Importance of tailoring data structures for NVM

DEVELOPMENT COST

PERFORMANCE

RECOVERY TIME