Foster B-Trees

Lucas Lersch

M. Sc. Caetano Sauer
Advisor

14.07.2014

Motivation

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Foster B-Trees

B^{link}-Trees:
- multicore
- concurrency

Write-Optimized B-Trees:
- flash memory
- large-writes
- wear leveling
- defragmentation

Fence Keys:
- verification

Agenda

1. **Background**
2. B^{link}-Trees
3. Write-Optimized B-Trees
4. Verification and Fence Keys
5. Foster B-Trees
6. Performance Evaluation

Latches and Locks

Latches

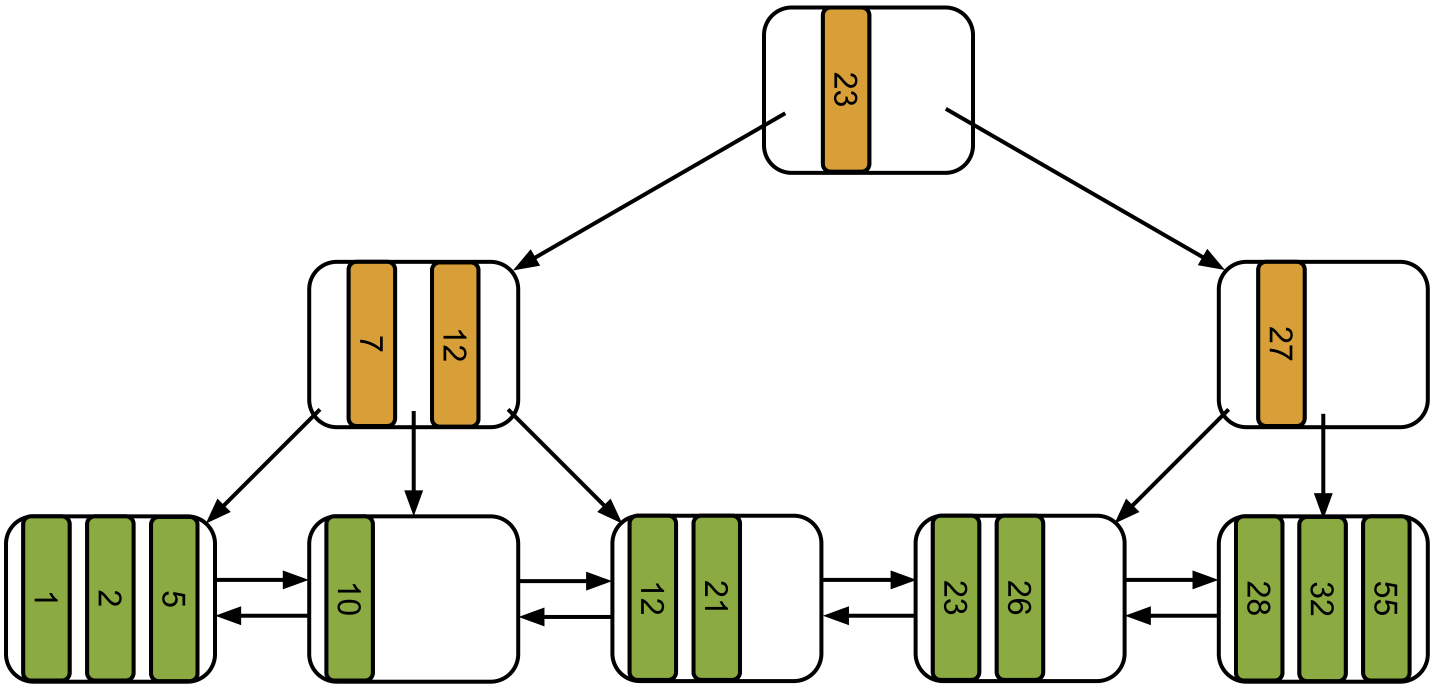
- acquired by threads
- protect in-memory physical structures
- during critical sections
- embedded in the data structure (semaphore)
- deadlock avoidance
- shared and exclusive modes
- simple and efficient

Locks

- acquired by transactions
- protect database logical contents
- during entire transaction
- lock manager (hash table)
- deadlock detection and resolution
- shared, exclusive, update, intention, etc...
- complex and expensive

B-trees

Technische Universität Kaiserslautern



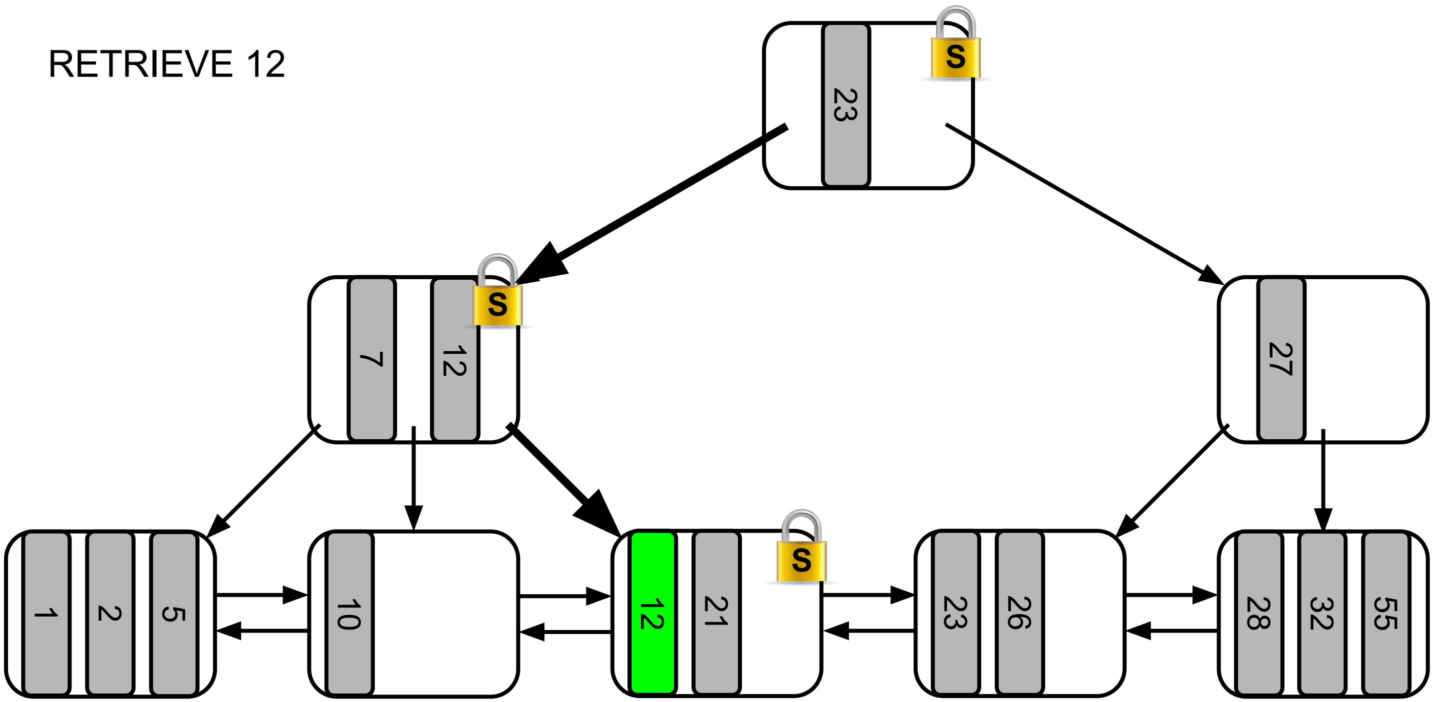
{key , DATA}

{key , pointer}

Retrieval

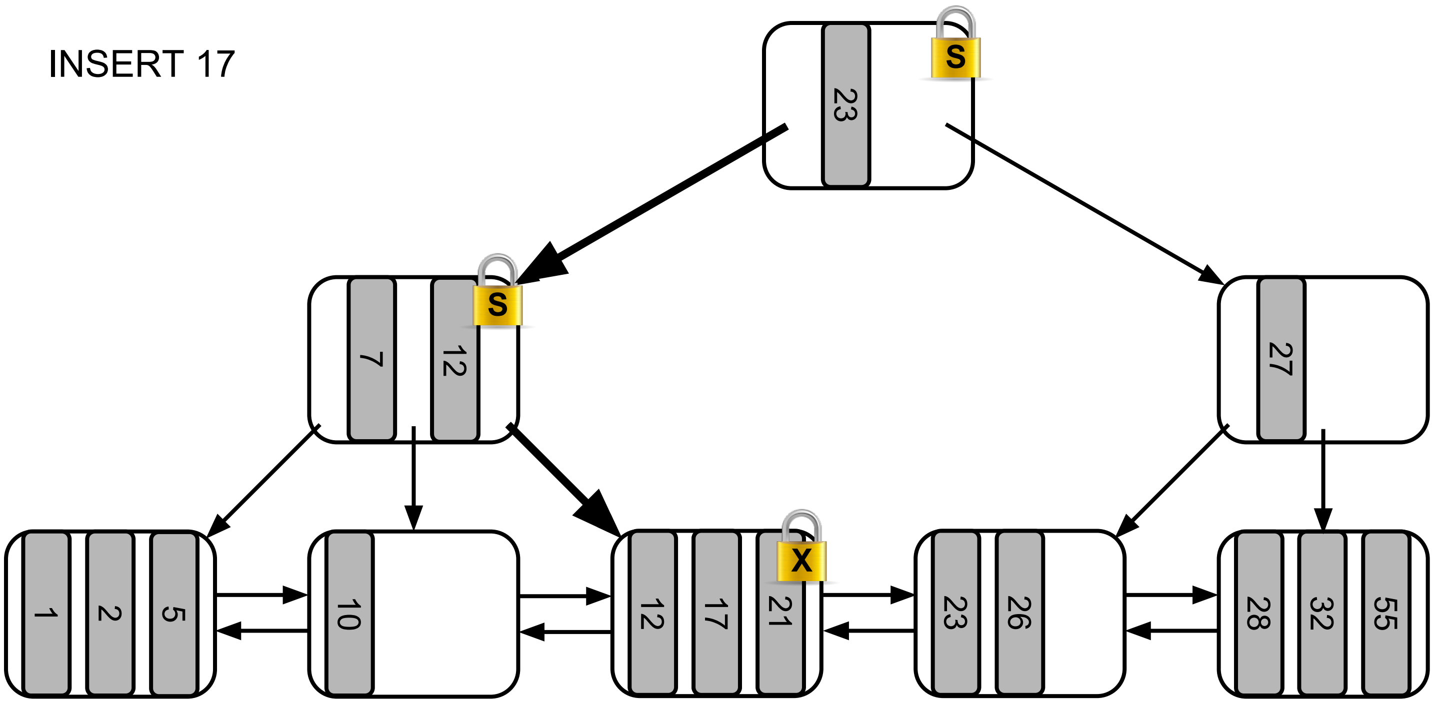
TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

RETRIEVE 12



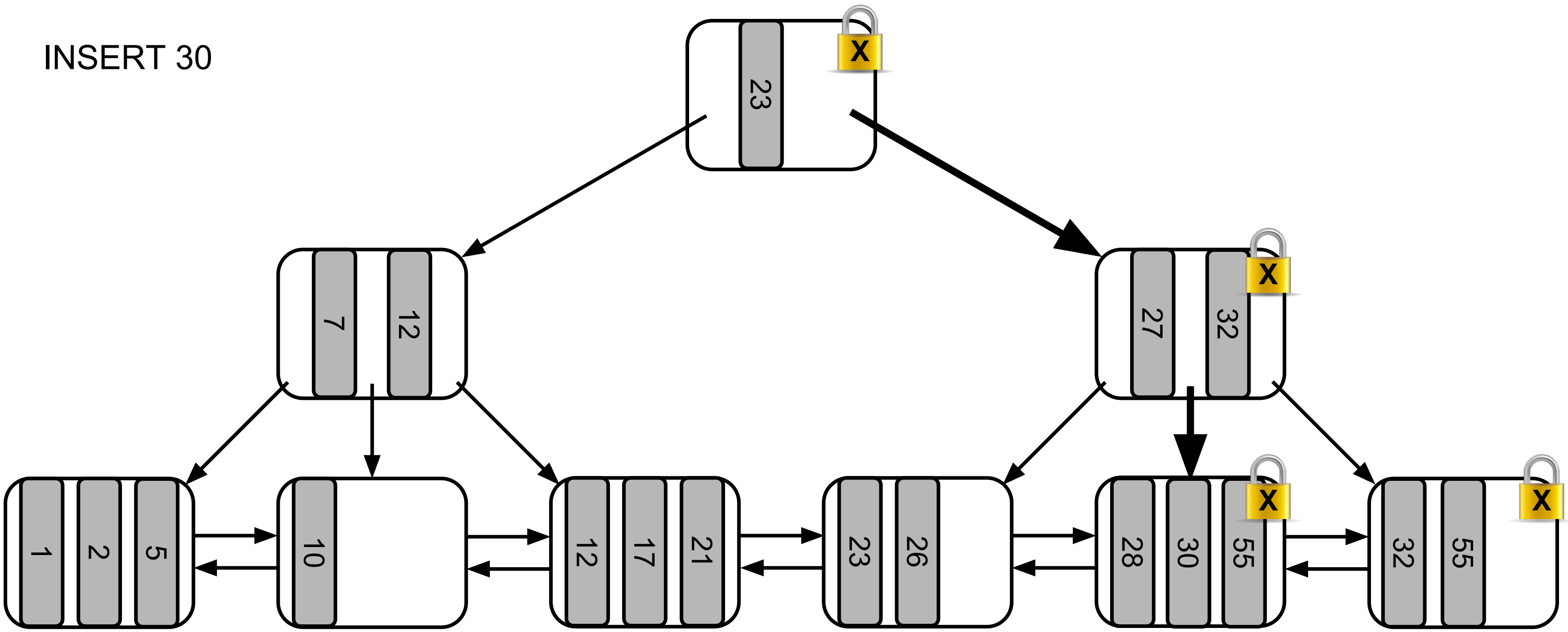
Insertion

INSERT 17



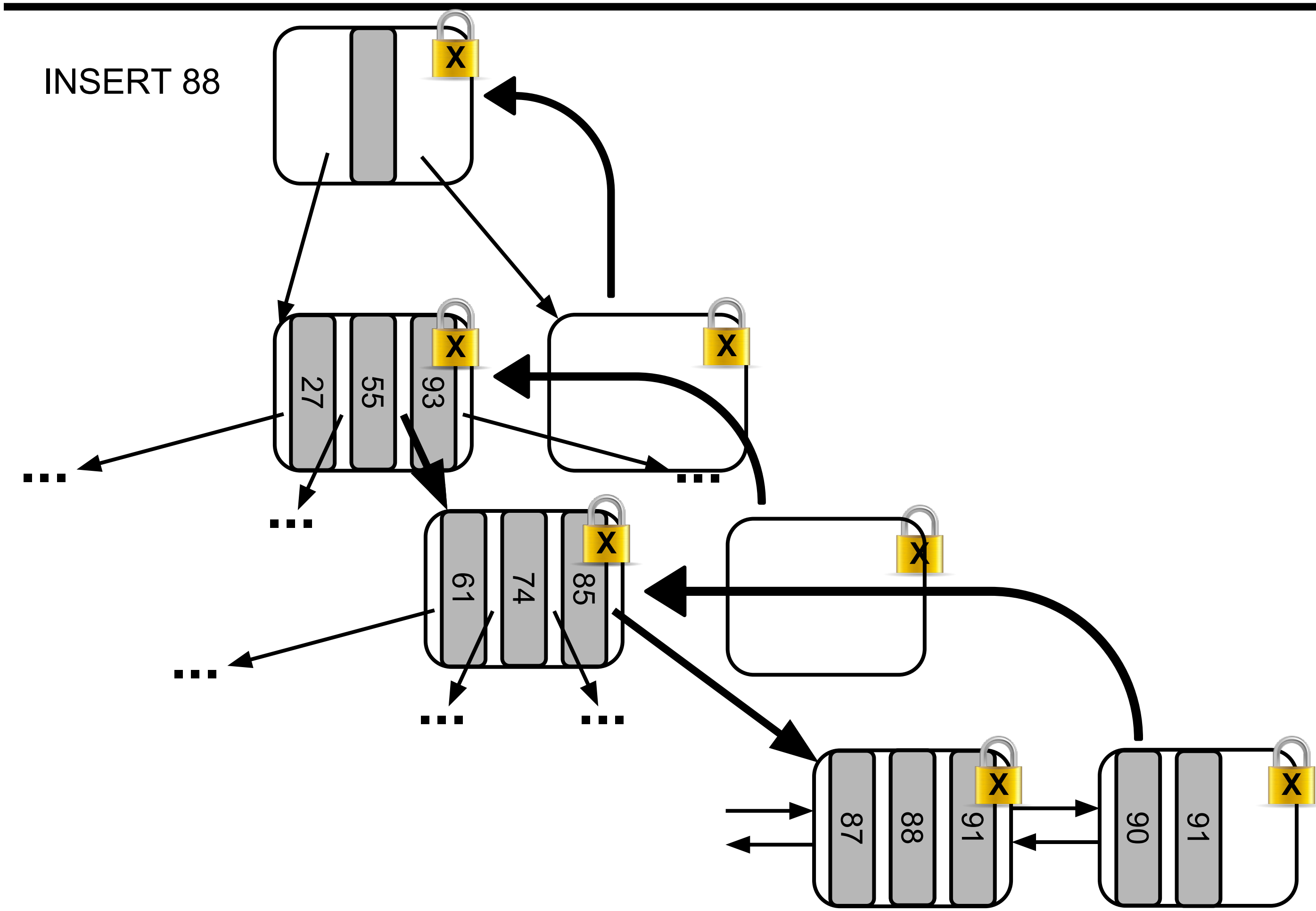
Insertion (node split)

INSERT 30



Insertion (worst case)

INSERT 88



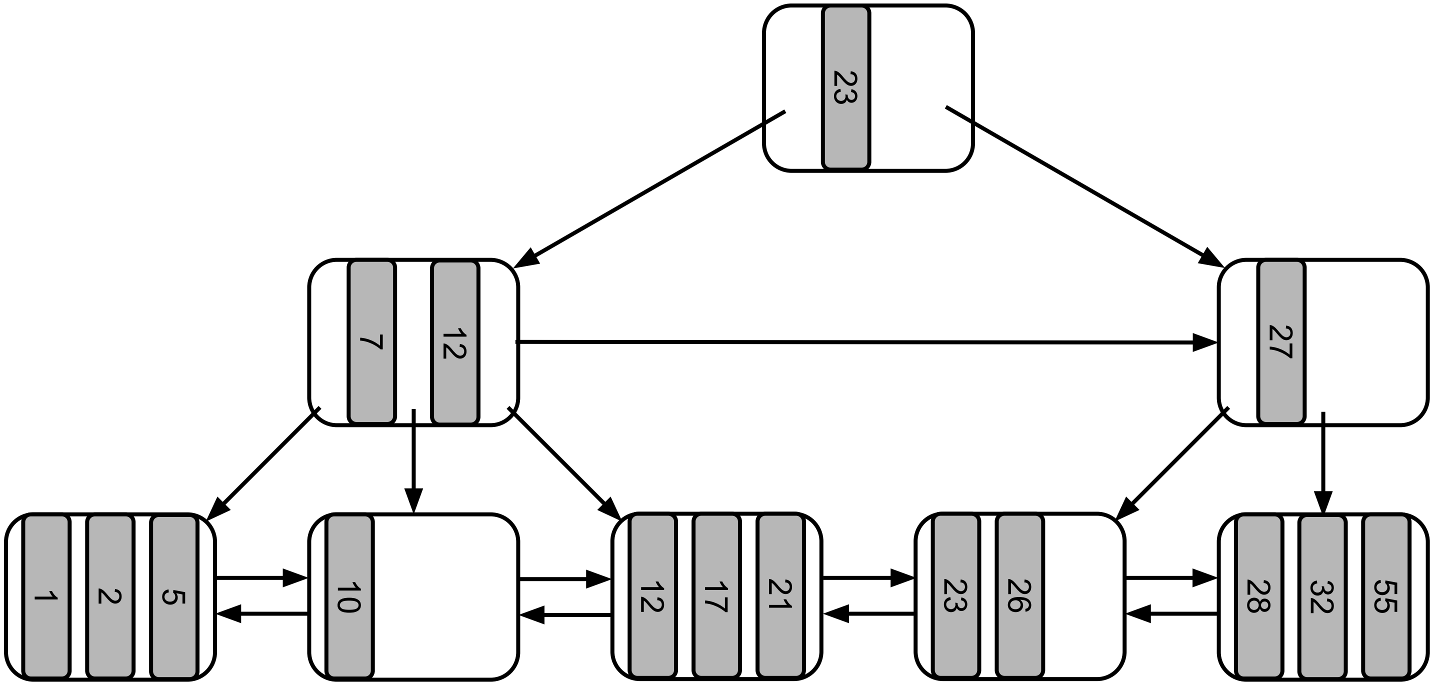
Deletion

- Merge underflowing nodes:
  - Reduce number of internal nodes
  - But complex and expensive
  - Database tend to increase rather than decrease
- Allow nodes to be completely emptied
- Operations must handle empty nodes
- Asynchronous utility for clean-up

Agenda

1. Background
2. **B^{link}-Trees**
3. Write-Optimized B-Trees
4. Verification and Fence Keys
5. Foster B-Trees
6. Performance Evaluation

B^{link}-trees



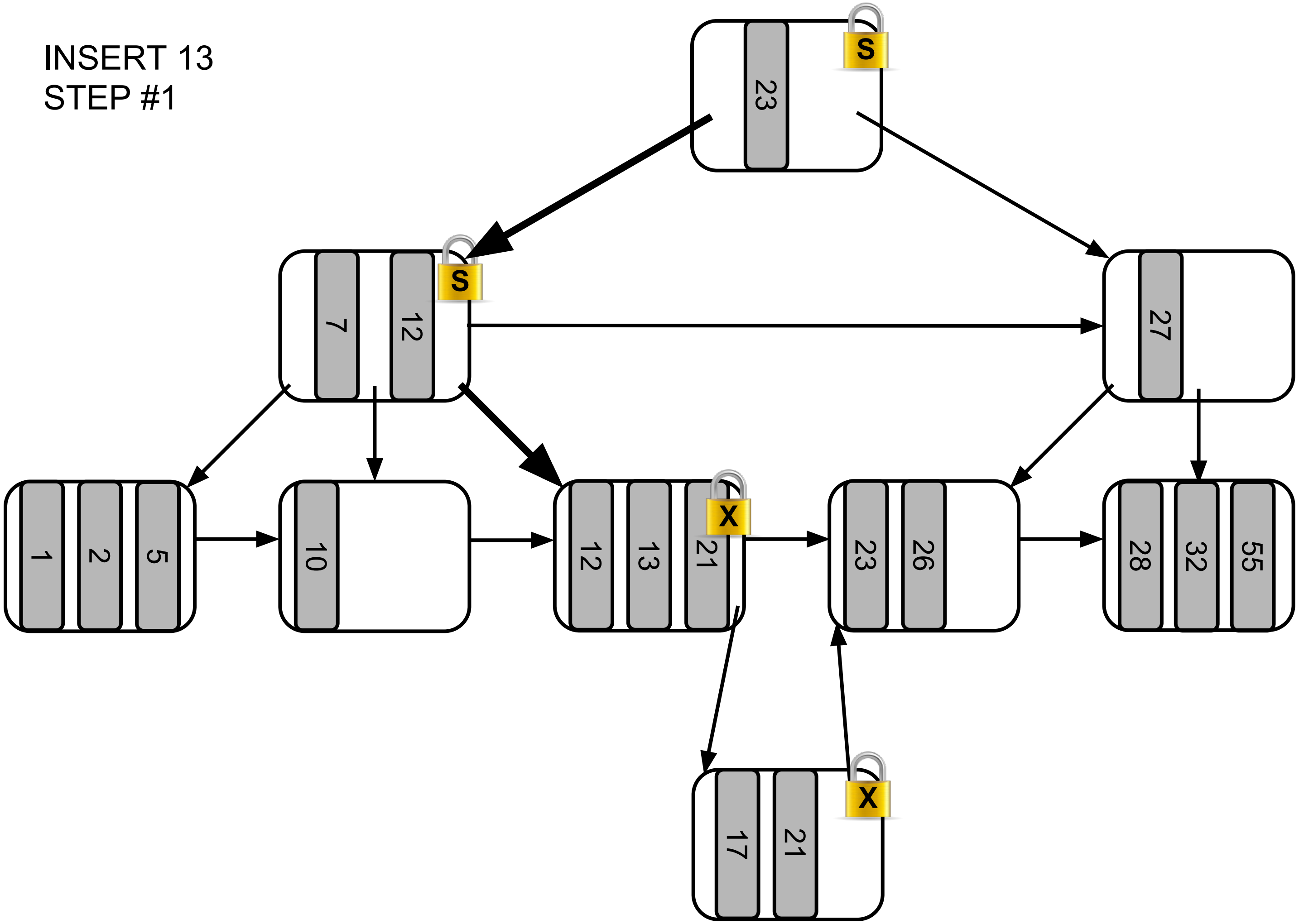
B^{link}-trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- Many-core processors
- Higher concurrency
- Avoid latch contention:
  - reduce number of latches
  - reduce granularity of critical sections
- “Link pointer”
  - additional method to reach any node

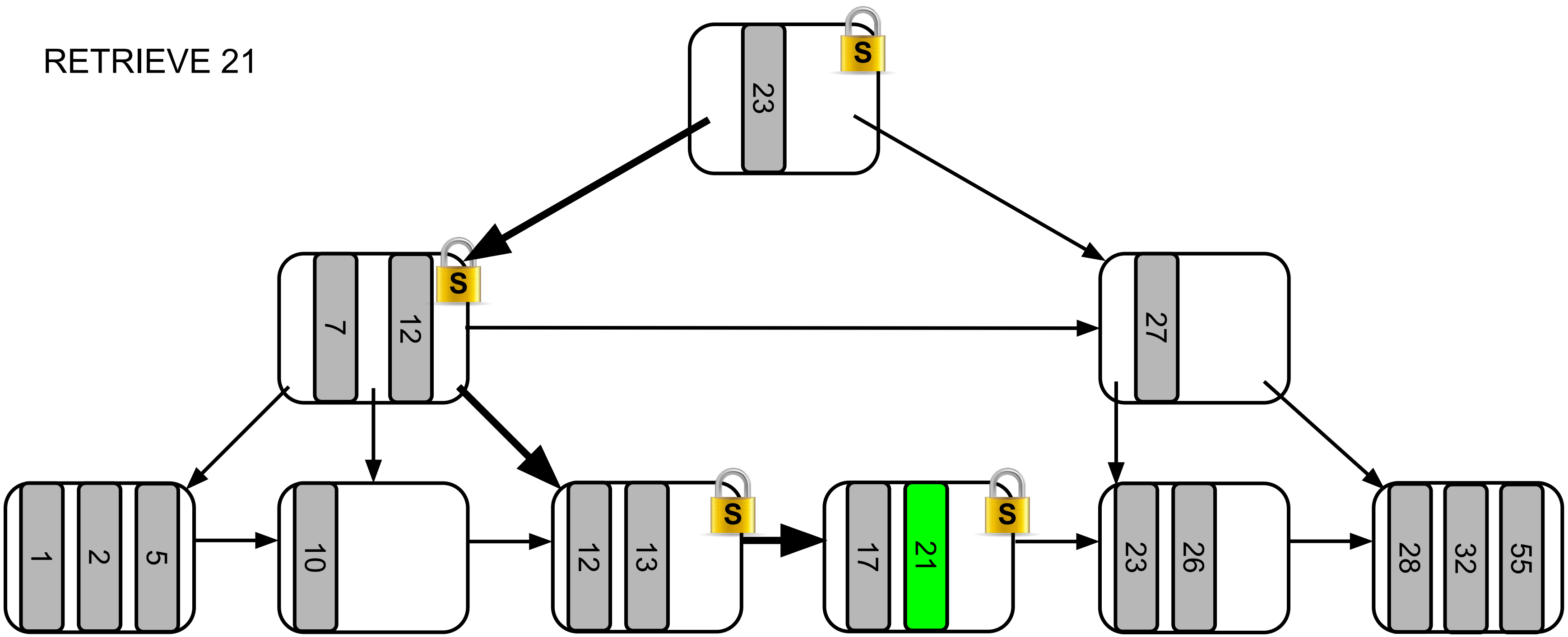
B^{link}-trees Insertion

INSERT 13
STEP #1



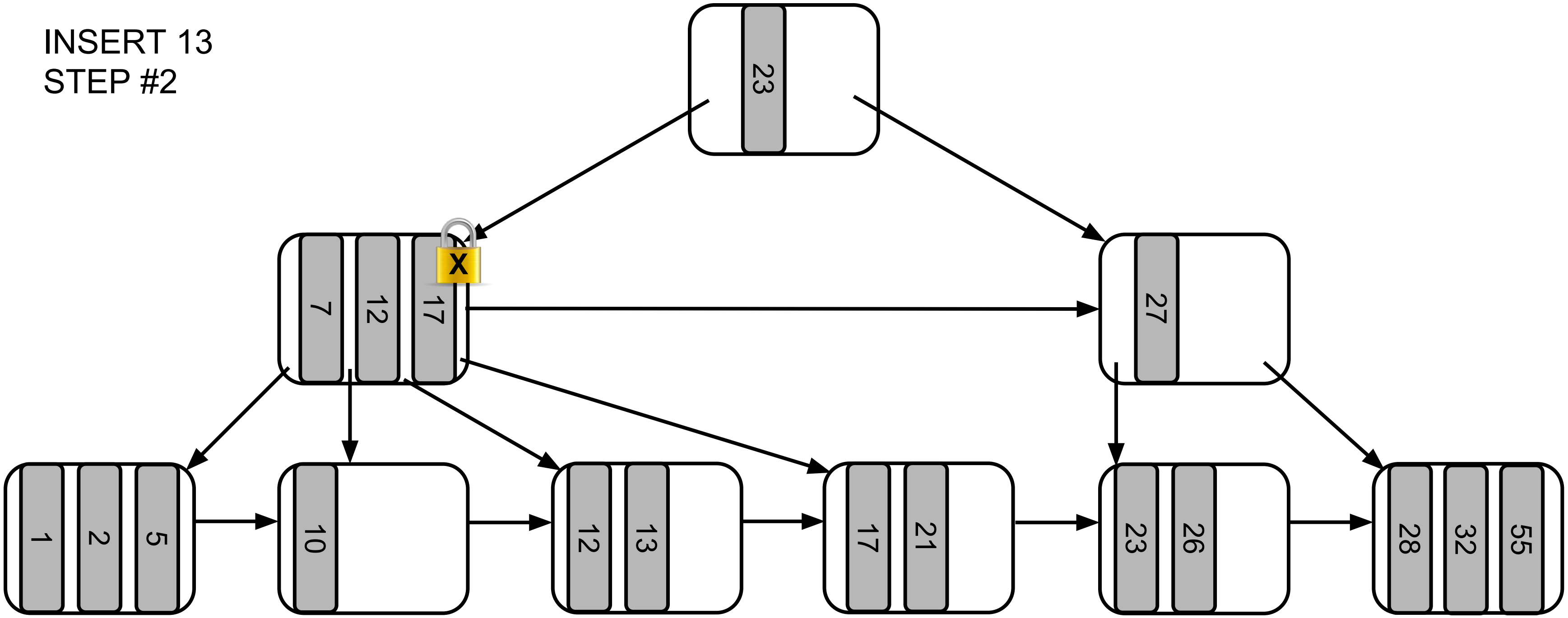
B^{link}-trees Retrieval

RETRIEVE 21



B^{link}-trees Insertion

INSERT 13
STEP #2



Agenda

1. Background
2. B^{link}-Trees
3. **Write-Optimized B-Trees**
4. Verification and Fence Keys
5. Foster B-Trees
6. Performance Evaluation

Write-optimized B-trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- 20~15 years ago: “90% reads, 10% writes”
- Today:
  - memory size grows: increased fraction of writes
  - “33% writes”
- Increase performance of writes!

Write-optimized B-trees

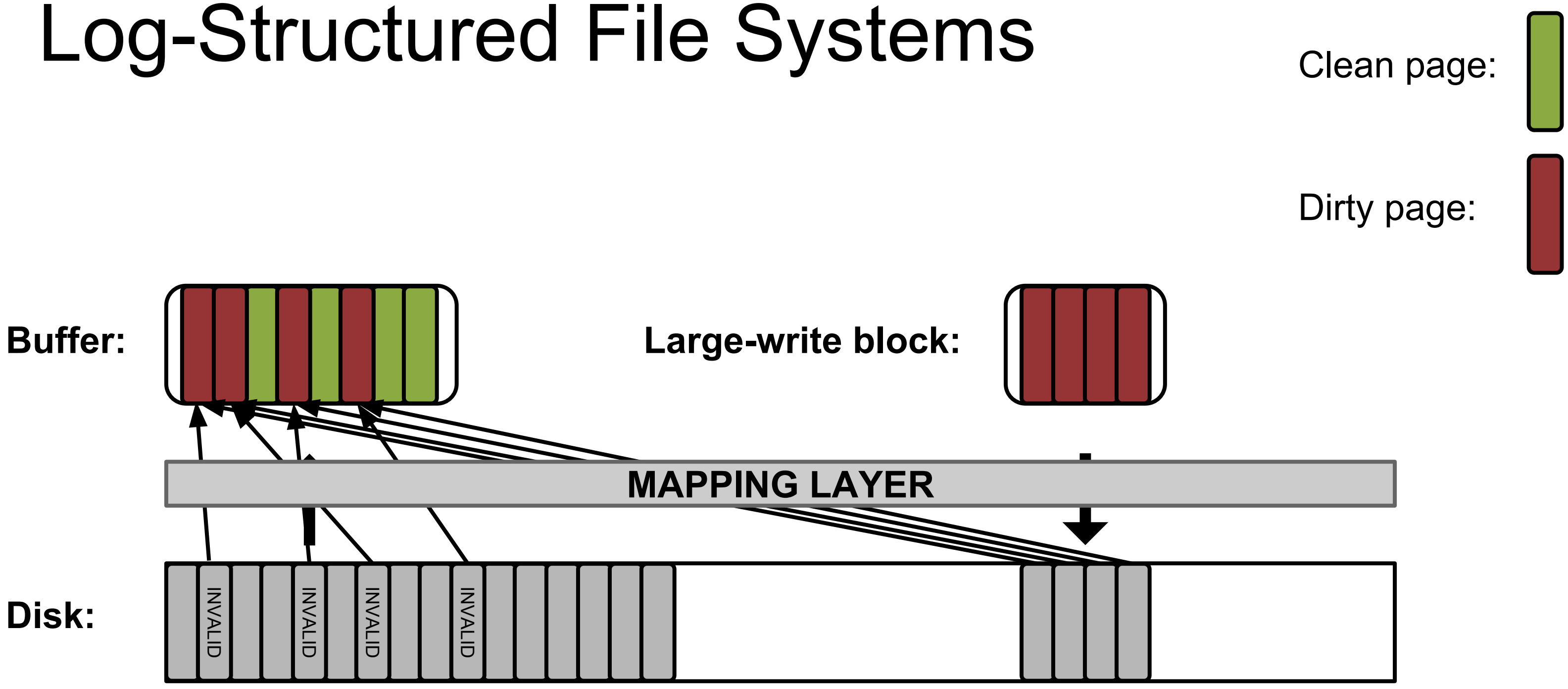
- Classical File Systems:

Clean page:

Dirty page:

Write-optimized B-trees

- Log-Structured File Systems



Write-optimized B-trees

- Log-Structured File Systems:
  - Advantages:
    - large-write operation
    - reduced number of seek operations
    - as large as entire erase blocks of a SSD
    - wear leveling
  - Disadvantages:
    - mapping layer
    - old copies
      - space reclamation
      - defragmentation

write performance to the detriment of scan performance

NOT DESIRABLE IN MOST DATABASE SYSTEMS!

Write-optimized B-trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

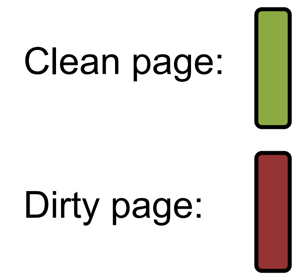
- ~~Database and B-tree indexes over LSFS~~
- Large-write operation into B-tree indexes
  - mapping overhead == B-tree operations
  - update in-place (read optimized)

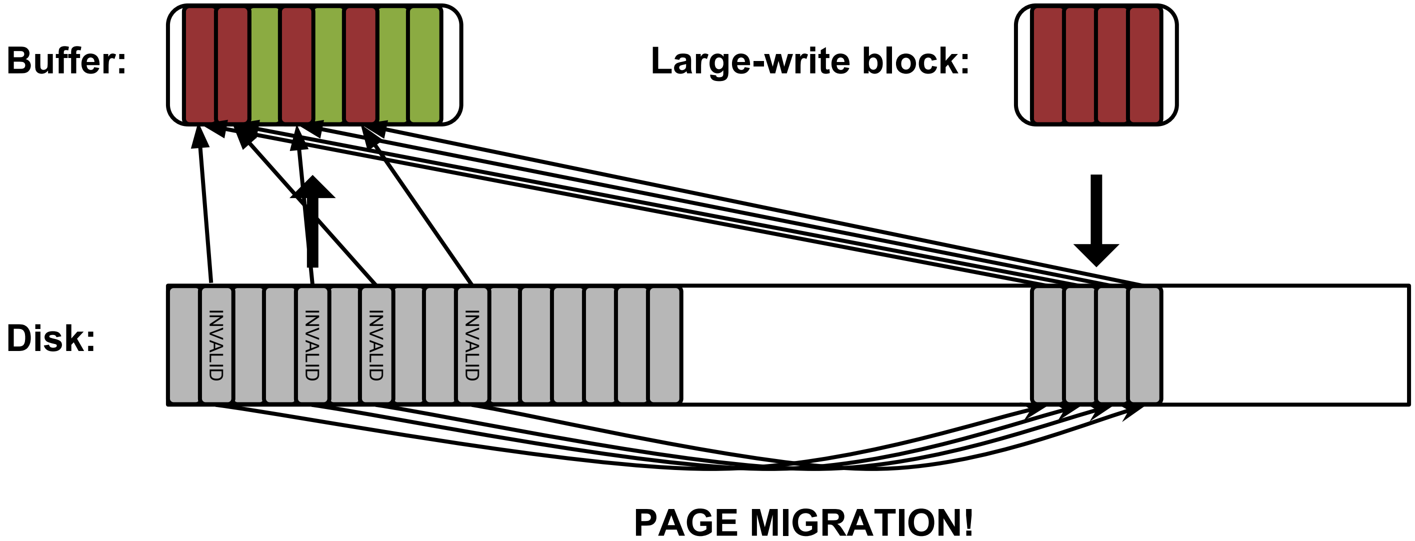
    OR

    large-write (write optimized)

Write-optimized B-trees

- Classical File Systems:

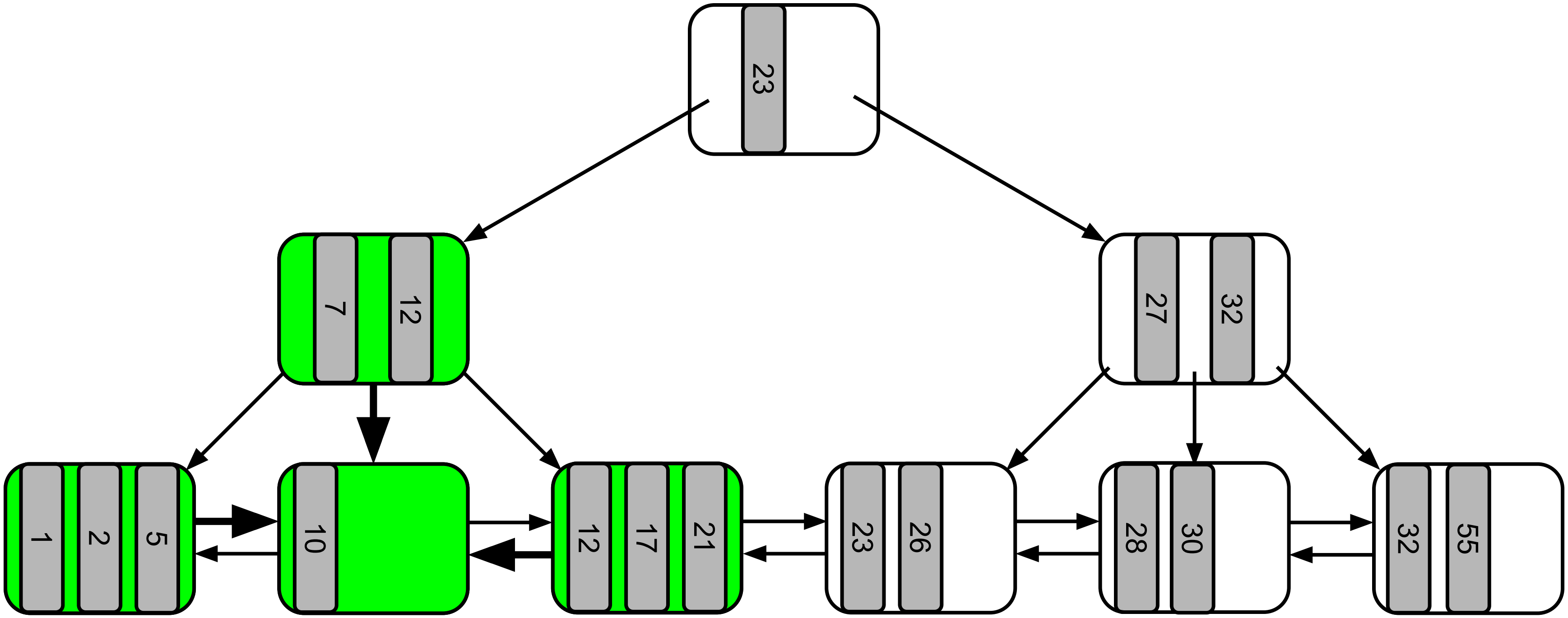




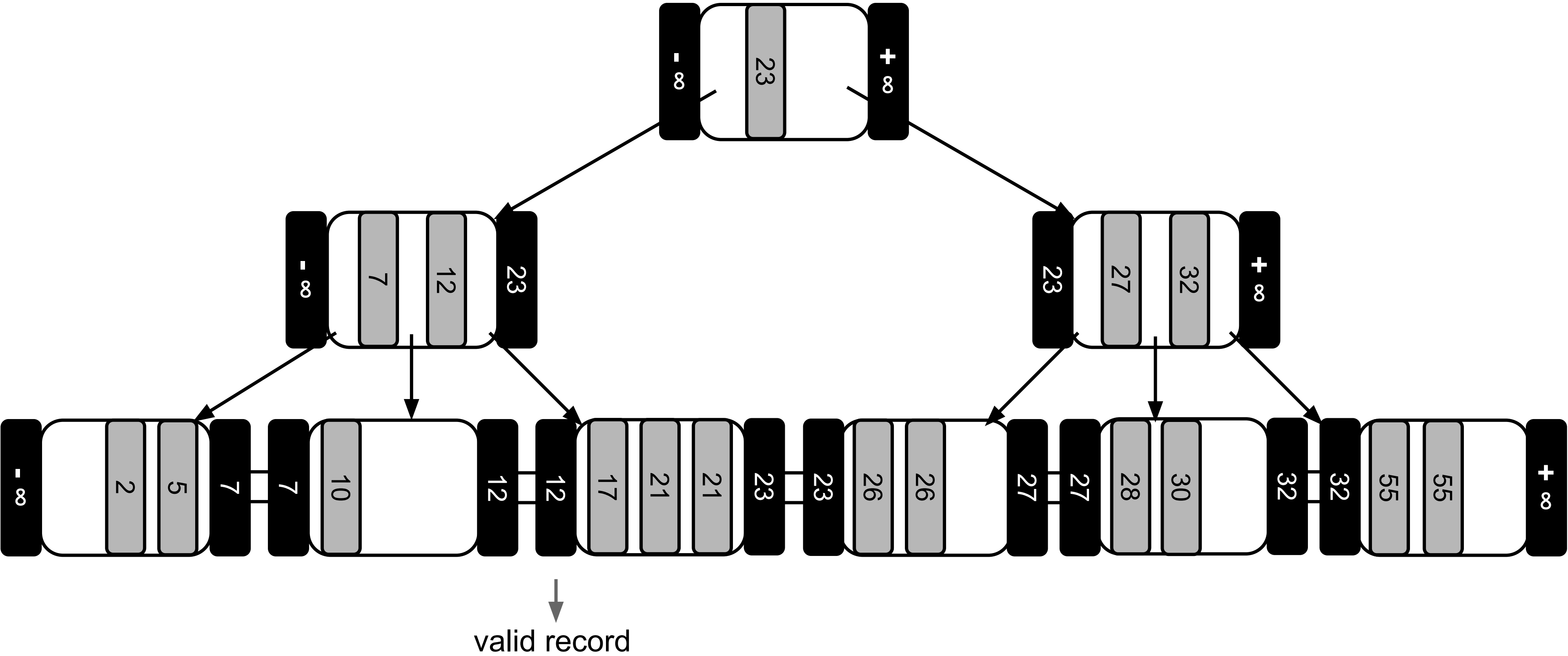
Write-optimized B-trees

- Page migration:
  - large-write
  - defragmentation
  - free space reclamation

Write-optimized B-trees



Write-optimized B-trees



Write-optimized B-trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- Symmetric fence keys concerns:
  - additional storage space in each node
    - prefix and suffix truncation of keys
    - additional compression methods

Write-optimized B-trees

- Symmetric fence keys concerns:
  - accessing the parent node:
    - probe the buffer pool for the parent node
    - link nodes in the buffer pool to their parents
    - mixed approach

Write-optimized B-trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- Logging a page migration:
  - optimized and inexpensive
  - small log records
  - a single log record for an entire operation

Agenda

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

1. Background
2. B^{link}-Trees
3. Write-Optimized B-Trees
4. **Verification and Fence Keys**
5. Foster B-Trees
6. Performance Evaluation

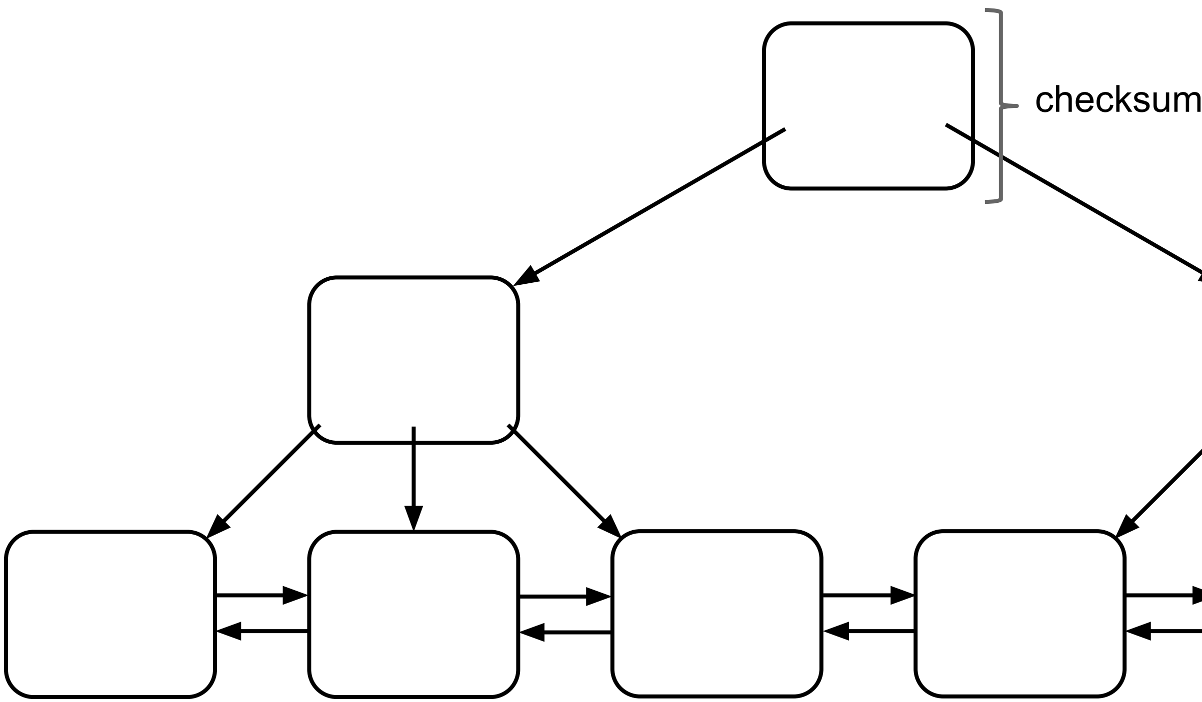
Verification and Fence Keys

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- Verification of physical integrity of a B-tree
  - in-page
  - cross-node
- Careful traversal of the whole B-tree structure
  - offline verification only :(
- Verification as part of regular maintenance
  - online verification
  - efficient

Verification and Fence Keys

- In-page verification
  - checksum of each individual page



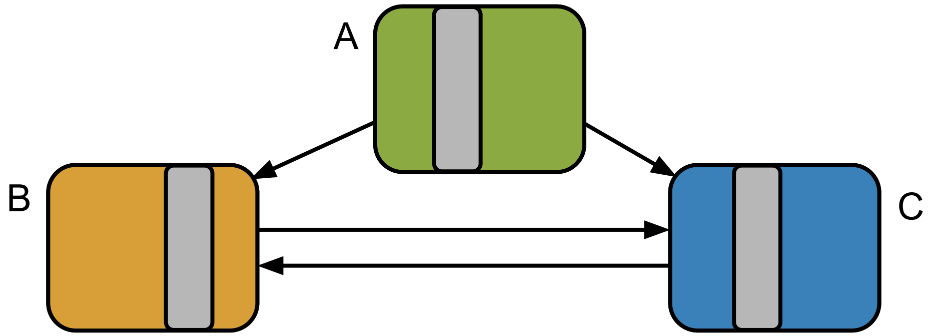
Verification and Fence Keys

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- Cross-node verification
  - Approach 1: navigate the whole index structure
    - from lowest to highest key value (depth-first)
    - matching forward and backward pointers with key ranges
    - advantage: simple
    - disadvantage: repeated read operations for each page deteriorate performance

Verification and Fence Keys

- Approach 2: aggregation of facts
  - Phase 1:



FACTS:

“B is leaf with key range [a,b)”

“C is leaf with key range [b,c)”

“B is leaf with key range [a,b)”

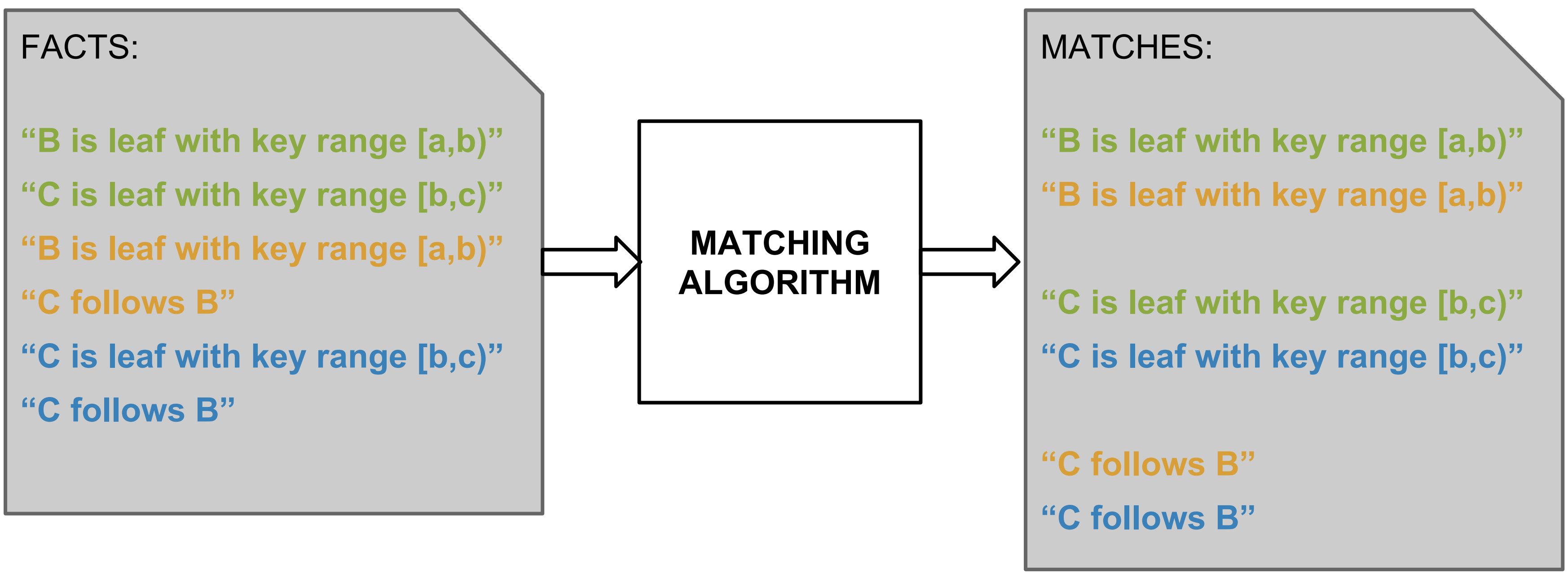
“C follows B”

“C is leaf with key range [b,c)”

“C follows B”

Verification and Fence Keys

- Approach 2: aggregation of facts
  - ⇒ Phase 2: stream the facts through a matching-algorithm



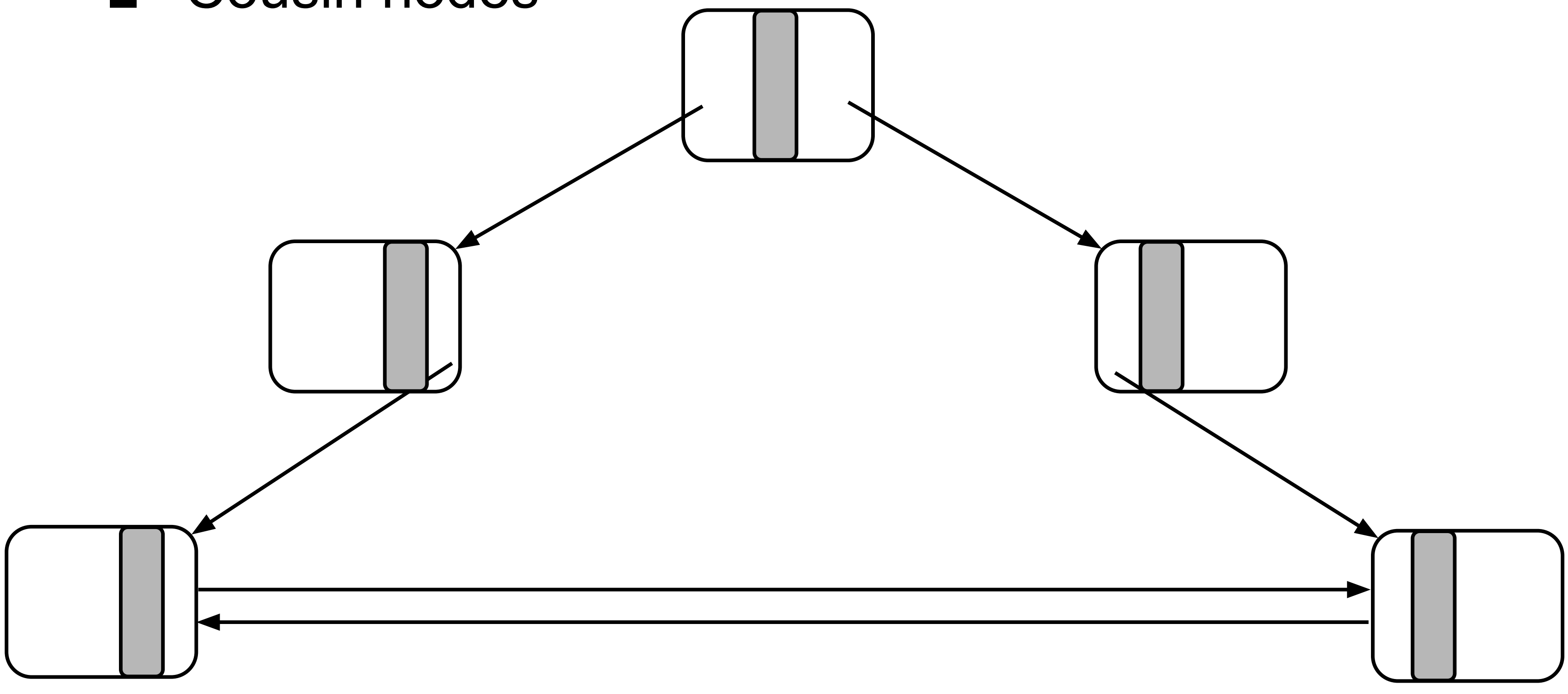
Verification and Fence Keys

- Approach 2: aggregation of facts
  - Fact formats:
    - ⇒ “node Y follows node X”
    - ⇒ “node X at level N+1 has child Y for key range [a,b)”
    - ⇒ “node X at level N has key range [a,b)”

  - “node Y follows node X”
    - ⇒ all keys in Y are greater than X?
    - ⇒ verification by transitivity

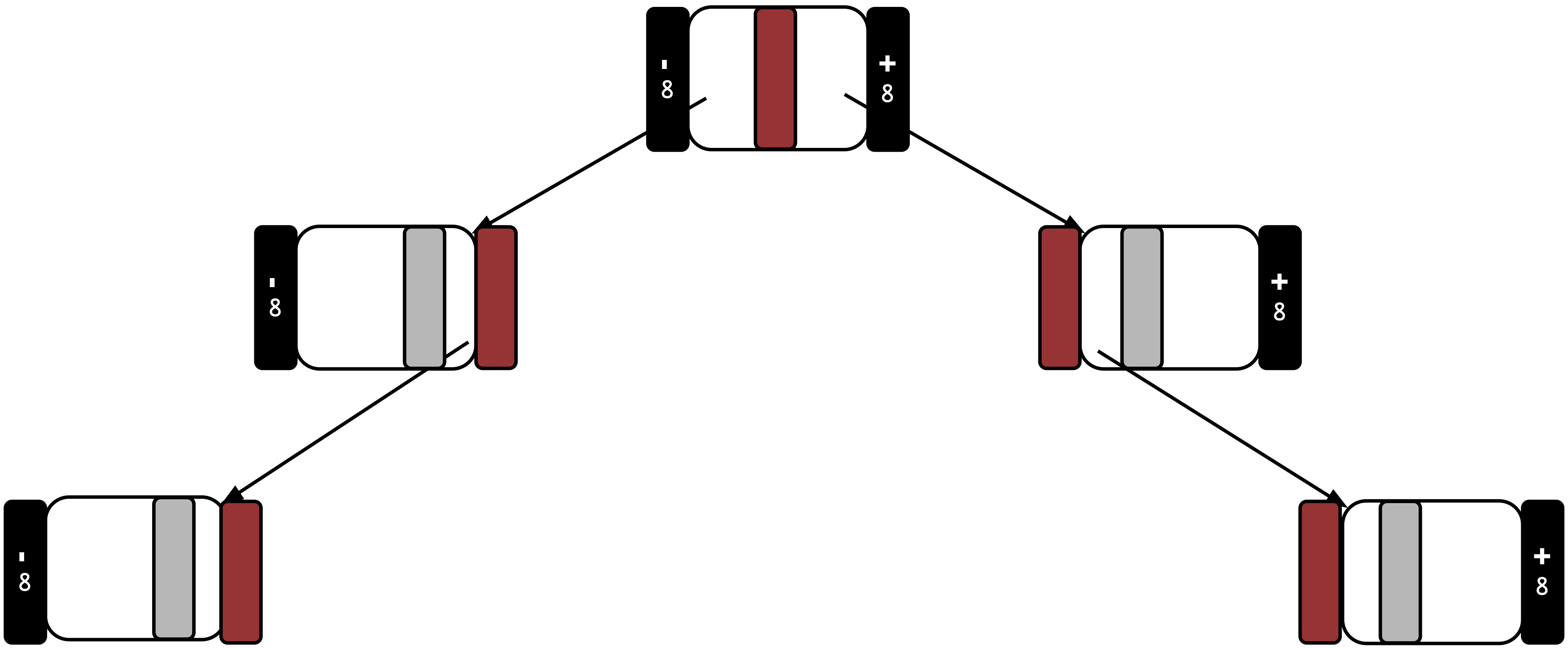
Verification and Fence Keys

- Approach 2: aggregation of facts
  - Cousin nodes



Verification and Fence Keys

- Approach 2: aggregation of facts



Verification and Fence Keys

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- Approach 2: aggregation of facts
  - replace backward and forward pointers with symmetric fence keys
  - facts have a single format:

    “node X at level N has key value V as low/high fence key”
  - each fact is matched with a exact copy that was extracted from the parent node
  - only equality comparisons required for matching facts

- Approach 3: bit vector filtering
  - fact = `{node_id, node_level, key_value, (low,high)_fence}`
  - hash fact to a value
  - reverse the bit in the position indicated by this value in a bitmap
  - matching facts hash to the same value
  - facts match in even numbers
  - at end, bitmap should be back to its original state

Agenda

Technische Universität Kaiserslautern

1. Background
2. B^{link}-Trees
3. Write-Optimized B-Trees
4. Verification and Fence Keys
5. **Foster B-Trees**
6. Performance Evaluation

Foster B-Trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- B^{link}-trees
  - require link-pointer
- Write-optimized B-tree
  - avoid backward and forward pointers for inexpensive page migration
- There is a contradiction. How then?

Foster B-Trees

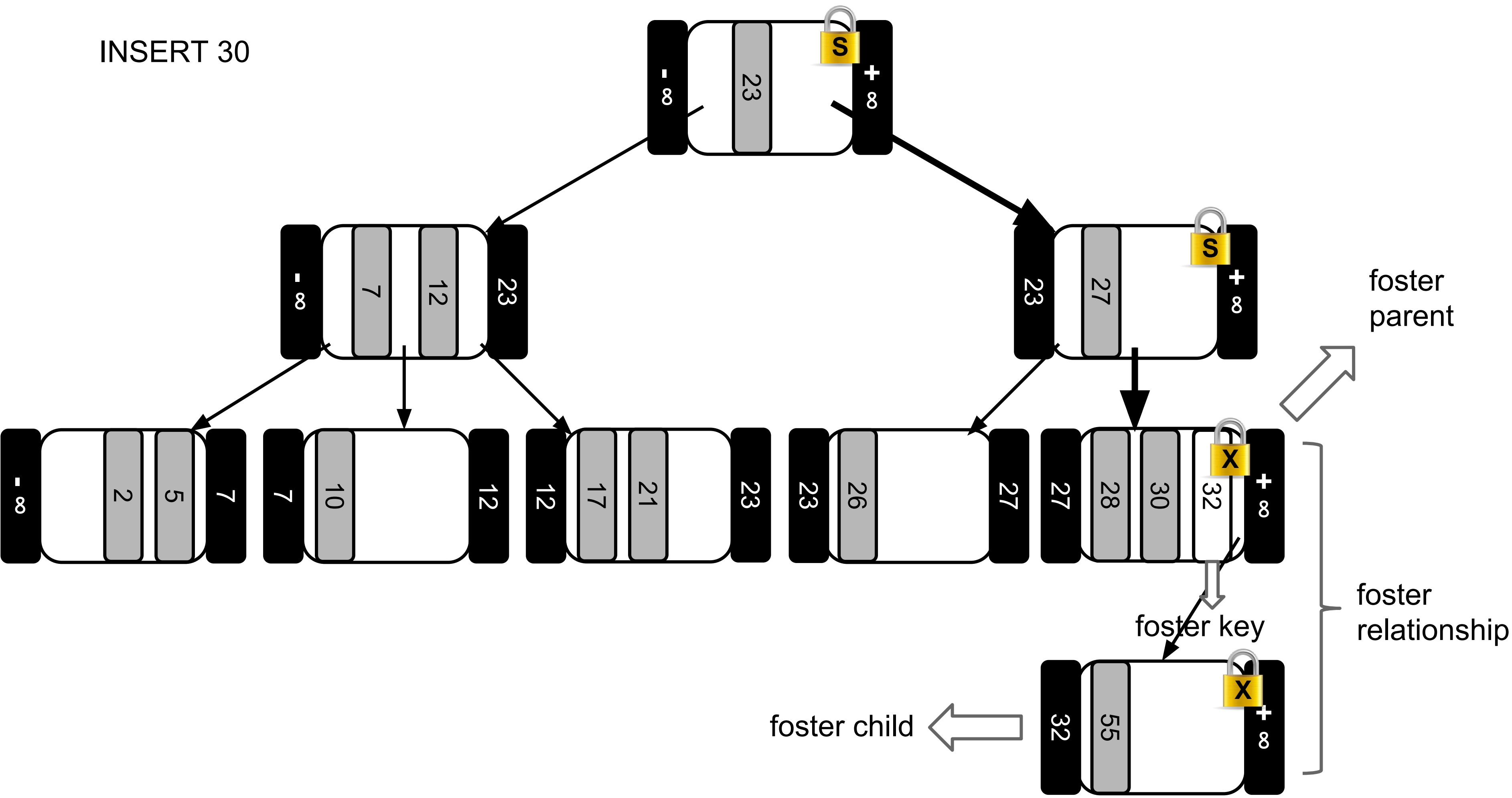
- Foster B-tree relax certain requirements
  - at an estimated small cost

- A Foster B-tree at an stable state looks like a Write-optimized B-tree

- Like a Blink-tree, nodes are split locally
  - no immediate upward propagation
  - intermediate states during a split

Foster B-Trees

INSERT 30

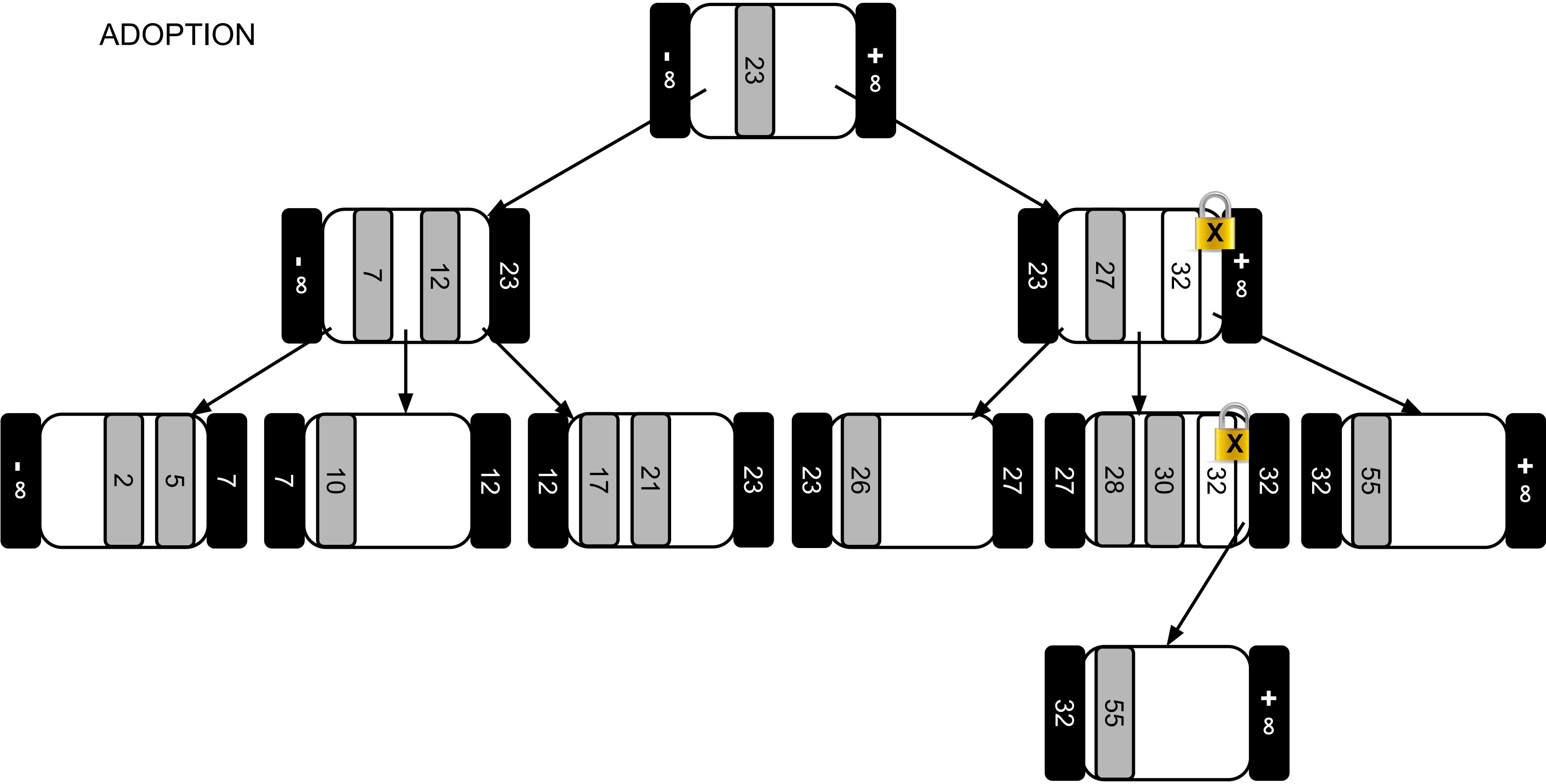


Foster B-Trees

- Foster relationship:
  - transient state
  - foster child act as an extension of foster parent node
  - root-to-leaf traversal may temporarily be longer
  - should be resolved quickly (avoid long foster chains)
    - adoption from foster child by permanent parent
      - opportunistically at root-to-leaf traversal
      - forced, by asynchronous utility

Foster B-Trees

ADOPTION



Agenda

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

1. Background
2. B^{link}-Trees
3. Write-Optimized B-Trees
4. Verification and Fence Keys
5. Foster B-Trees
6. **Performance Evaluation**

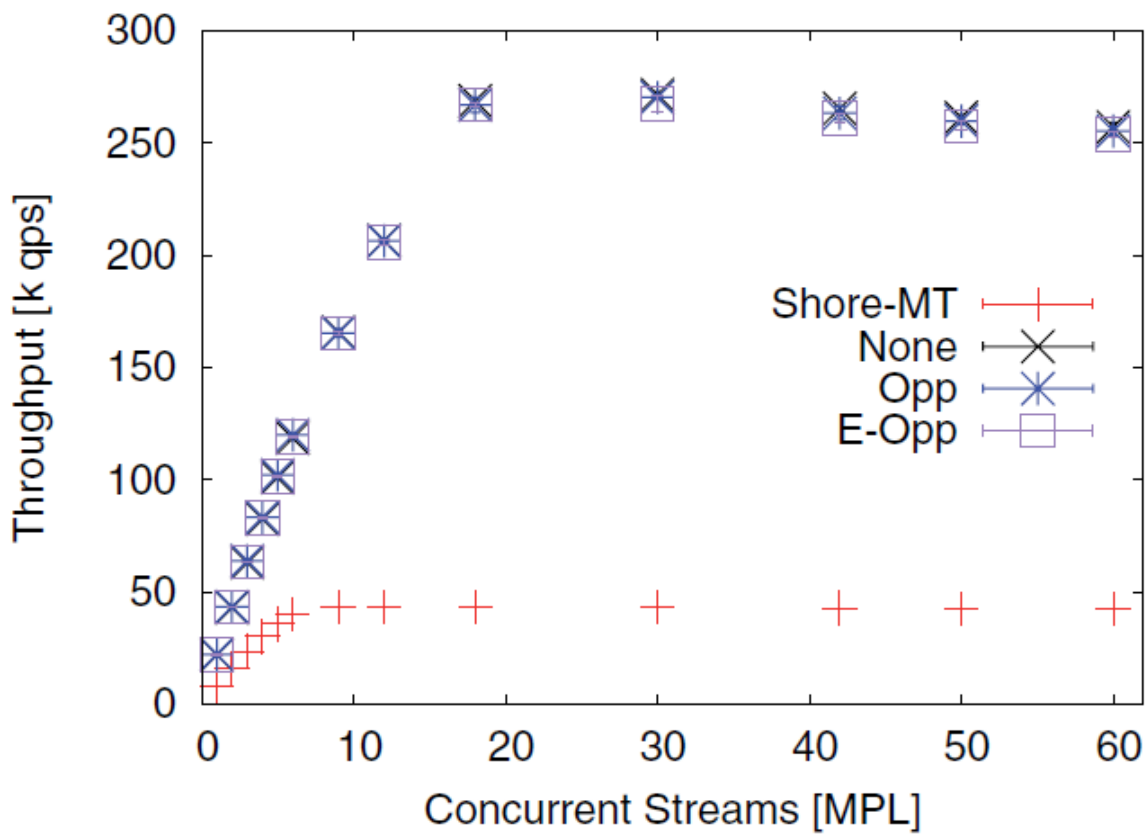
Performance Evaluation

- Shore-MT
  - designed for high concurrency
  - classical B-trees

- Environment
  - 8 CPU cores (64 hardware contexts)
  - 64GB of RAM
  - RAID-1

Performance Evaluation

- Mixed workload
- Foster relations avoid latch contention
- No long chains of foster relations
  - adoption not required

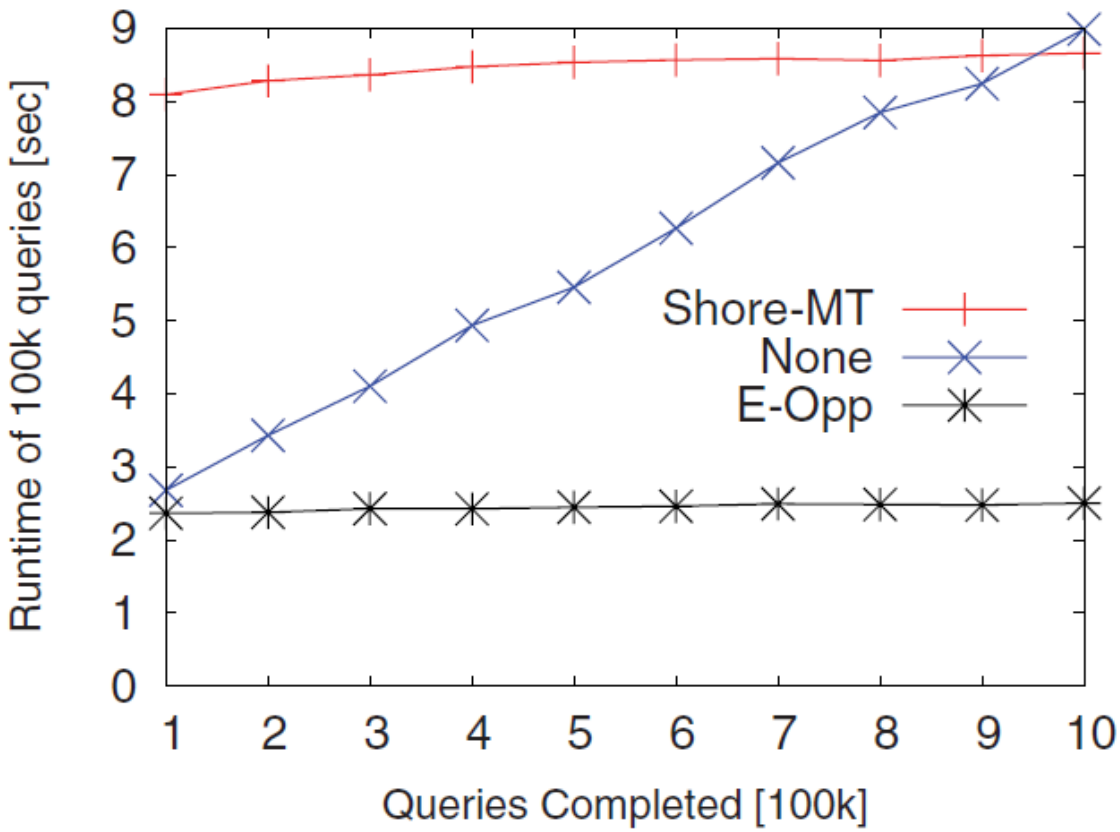


Performance Evaluation

- Mixed workload
  - single thread
  - 80% reads
  - 20% skewed updates
    - force adoption

- E-OPP: queries runtime remains the same

- None: unsolved foster relations, so runtime tend to increase



Conclusion

- B^{link}-trees
  - high concurrency
- Write-optimized B-trees
  - high update rates
- Symmetric fence keys
  - efficient verification

Foster B-trees

simpler

Thank you!

Questions?

Write-optimized B-trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- Symmetric fence keys concerns:
  - additional storage space in each node
    - prefix and suffix truncation of keys
    - additional compression methods
  - inefficient leaf-level scan (no pointers!)
    - ~1% of internal nodes
    - asynchronous read-ahead
    - prefetching of leaf nodes guided by ancestor nodes

Write-optimized B-trees

- Logging a page migration:
  - "Fully-logged"
    - page contents written to log record
    - recovery copy page contents from log
    - expensive

Write-optimized B-trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- “Forced-write”
  - log record = {old_location, new location}
  - single log record for the whole migration transaction:
    - ⇒ transaction begin
    - ⇒ allocation changes
    - ⇒ page migration
    - ⇒ transaction commit
  - requires forcing page contents to new location prior to writing log record(no write-ahead logging!)
  - update global allocation information only after writing log record (preserve old page location and contents)
  - if there is a log record, page is at new location
  - otherwise, migration did not took place and page is at old location

Write-optimized B-trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- "Forced-write"
  - advantages:
    - ⇒ single and small log record
    - ⇒ asynchronous write of log record
  - disadvantages:
    - ⇒ forcing page contents to new location

Write-optimized B-trees

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- “Non-logged”
  - similar to “fully-logged”
  - force page contents to new location
  - introduces a write dependency:
    - ⇒ old page location is deallocated, but...
    - ⇒ do not overwrite contents in older page location before writing page contents to new location
  - weakness: backup and recovery
    - ⇒ backup of currently allocated pages of an index
    - ⇒ log record must be complemented with updated page contents
    - ⇒ same cost of “fully-logged”

Verification and Fence Keys

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- Approach 2: aggregation of facts
  - Phase 2: stream the facts through a matching-algorithm
    - ⇒ From leaf-node X “**node Y follows node X**” matches from node Y “**node Y follows node X**”
    - ⇒ From node X “**node X at level N+1 has child Y from key range [a,b)**” matches from node Y “**node Y at level N has key range [a, b)**”

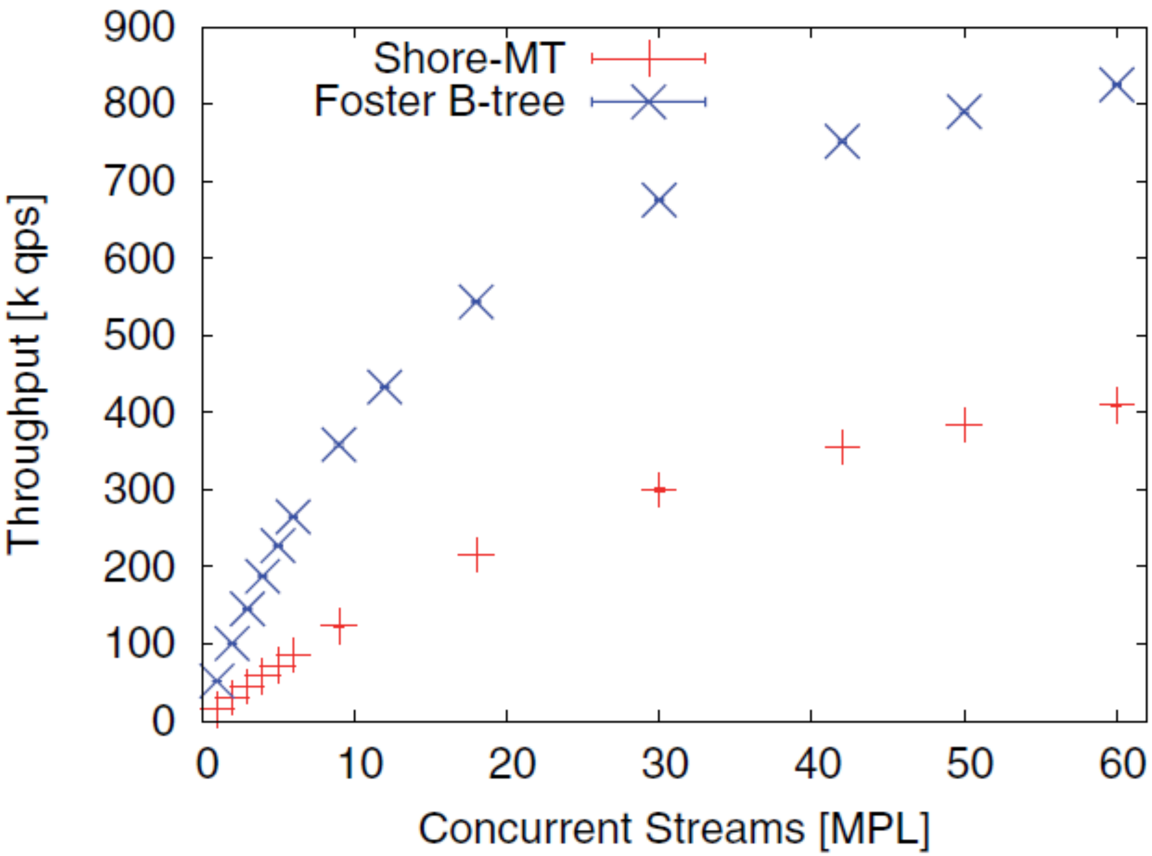
Verification and Fence Keys

- Approach 2: aggregation of facts
  - "node Y follows node X"
  - how to verify that all keys in Y are greater than all the keys in X?
    - ⇒ done transitively by the separator key in the parent of X and Y
  - what if X and Y are neighbors but do not share the same parent, but share a high ancestor?
    - ⇒ X and Y are cousin nodes
    - ⇒ transitive verification is not guaranteed across skipped levels

Performance Evaluation

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

- Selection queries
- Read-only
- No foster relations
- No logging
- No latch conflict
- Shore-MT has a higher compression
- Extra effort for reconstructing and compare a key for binary search



Performance Evaluation

- Similar to previous experiment
  - increasing number of threads

- 80% reads
  - Foster B-trees perform better (as seen)