Lecture #07 – Indexing (OLTP)

TODAY’S AGENDA

Latch Implementations
Modern OLTP Indexes
COMPARE-AND-SWAP

Atomic instruction that compares contents of a memory location $M$ to a given value $V$
→ If values are equal, installs new given value $V'$ in $M$
→ Otherwise operation fails
COMPARE-AND-SWAP

Atomic instruction that compares contents of a memory location $M$ to a given value $V$
→ If values are equal, installs new given value $V'$ in $M$
→ Otherwise operation fails

```c
__sync_bool_compare_and_swap(&M, 20, 30)
```
COMPARE-AND-SWAP

Atomic instruction that compares contents of a memory location $M$ to a given value $V$

→ If values are equal, installs new given value $V'$ in $M$

→ Otherwise operation fails

```c
__sync_bool_compare_and_swap(&M, 20, 30)
```
COMPARE-AND-SWAP

Atomic instruction that compares contents of a memory location $M$ to a given value $V$

→ If values are equal, installs new given value $V'$ in $M$

→ Otherwise operation fails

M

20

Address

__sync_bool_compare_and_swap(&M, 20, 30)
COMPARE-AND-SWAP

Atomic instruction that compares contents of a memory location $M$ to a given value $V$
→ If values are equal, installs new given value $V'$ in $M$
→ Otherwise operation fails

% \_\_sync\_bool\_compare\_and\_swap(\&M, 20, 30)
COMPARE-AND-SWAP

Atomic instruction that compares contents of a memory location $M$ to a given value $V$
→ If values are equal, installs new given value $V'$ in $M$
→ Otherwise operation fails

```
__sync_bool_compare_and_swap(&M, 20, 30)
```
**COMPARE-AND-SWAP**

Atomic instruction that compares contents of a memory location $M$ to a given value $V$
→ If values are equal, installs new given value $V'$ in $M$
→ Otherwise operation fails

```c
__sync_bool_compare_and_swap(&M, 20, 30)
```
COMPARE-AND-SWAP

Atomic instruction that compares contents of a memory location $M$ to a given value $V$
→ If values are equal, installs new given value $V'$ in $M$
→ Otherwise operation fails

```c
__sync_bool_compare_and_swap(&M, 20, 30)
```
Atomic instruction that compares contents of a memory location \( M \) to a given value \( V \)

\[ \rightarrow \text{If values are equal, installs new given value } V' \text{ in } M \]

\[ \rightarrow \text{Otherwise operation fails} \]
COMPARE-AND-SWAP

Atomic instruction that compares contents of a memory location $M$ to a given value $V$
→ If values are equal, installs new given value $V'$ in $M$
→ Otherwise operation fails

```c
__sync_bool_compare_and_swap(&M, 25, 35)
```
LATCH IMPLEMENTATIONS

Blocking OS Mutex
Test-and-Set Spinlock
Queue-based Spinlock
Reader-Writer Locks
LATCH IMPLEMENTATIONS

Choice #1: Blocking OS Mutex
→ Simple to use
→ Non-scalable (about 25ns per lock/unlock invocation)
→ Example: `pthread_mutex_t` (calls `futex`)
LATCH IMPLEMENTATIONS

Choice #1: Blocking OS Mutex
→ Simple to use
→ Non-scalable (about 25ns per lock/unlock invocation)
→ Example: `pthread_mutex_t` (calls `futex`)

```c
pthread_mutex_t lock;

: 
pthread_mutex_lock(&lock);
// Do something special...
pthread_mutex_unlock(&lock);
```
LATCH IMPLEMENTATIONS

Choice #1: Blocking OS Mutex
→ Simple to use
→ Non-scalable (about 25ns per lock/unlock invocation)
→ Example: `pthread_mutex_t` (calls `futex`)

```c
pthread_mutex_t lock;
:
pthread_mutex_lock(&lock);
// Do something special...
pthread_mutex_unlock(&lock);
```
LATCH IMPLEMENTATIONS

Choice #1: Blocking OS Mutex
→ Simple to use
→ Non-scalable (about 25ns per lock/unlock invocation)
→ Example: \texttt{pthread_mutex_t} (calls \texttt{futex})

```c
pthread_mutex_t lock;
:
pthread_mutex_lock(&lock);
// Do something special...
pthread_mutex_unlock(&lock);
```
LATCH IMPLEMENTATIONS

Choice #2: Test-and-Set Spinlock (TAS)
→ Very efficient (single instruction to lock/unlock)
→ Non-scalable, not cache friendly
→ Example: `std::atomic_flag`
Choice #2: Test-and-Set Spinlock (TAS)
→ Very efficient (single instruction to lock/unlock)
→ Non-scalable, not cache friendly
→ Example: `std::atomic_flag`

```cpp
std::atomic_flag lock;
:
while (lock.test_and_set(...)) {
    // Yield? Abort? Retry?
}
```
Choice #2: Test-and-Set Spinlock (TAS)
→ Very efficient (single instruction to lock/unlock)
→ Non-scalable, not cache friendly
→ Example: `std::atomic_flag`

```cpp
std::atomic_flag lock;
⋮
while (lock.test_and_set(...)) {
    // Yield? Abort? Retry?
}
```
Choice #2: Test-and-Set Spinlock (TAS)
→ Very efficient (single instruction to lock/unlock)
→ Non-scalable, not cache friendly
→ Example: `std::atomic_flag`

```cpp
std::atomic_flag lock;
:
while (lock.test_and_set(…)) {
    // Yield? Abort? Retry?
}
```
LATCH IMPLEMENTATIONS

Choice #3: Queue-based Spinlock (MCS)
→ More efficient than mutex, better cache locality
→ Non-trivial memory management
→ Example: `std::atomic_flag`
LATCH IMPLEMENTATIONS

Choice #3: Queue-based Spinlock (MCS)
→ More efficient than mutex, better cache locality
→ Non-trivial memory management
→ Example: \texttt{std::atomic\_flag}
LATCH IMPLEMENTATIONS

Choice #3: Queue-based Spinlock (MCS)
→ More efficient than mutex, better cache locality
→ Non-trivial memory management
→ Example: `std::atomic_flag`

Base Lock

Mellor-Crummey and Scott
LATCH IMPLEMENTATIONS

Choice #3: Queue-based Spinlock (MCS)
→ More efficient than mutex, better cache locality
→ Non-trivial memory management
→ Example: `std::atomic_flag`

Mellor-Crummey and Scott

Base Lock

CPU1
LATCH IMPLEMENTATIONS

Choice #3: Queue-based Spinlock (MCS)
→ More efficient than mutex, better cache locality
→ Non-trivial memory management
→ Example: `std::atomic_flag`

Mellor-Crumme and Scott
LATCH IMPLEMENTATIONS

Choice #3: Queue-based Spinlock (MCS)
→ More efficient than mutex, better cache locality
→ Non-trivial memory management
→ Example: `std::atomic_flag`

Mellor-Crummey and Scott

```
Base Lock

next

CPU1 Lock

next

CPU1

CPU2
```
LATCH IMPLEMENTATIONS

Choice #3: Queue-based Spinlock (MCS)
→ More efficient than mutex, better cache locality
→ Non-trivial memory management
→ Example: `std::atomic_flag`

Mellor-Crummey and Scott

Base Lock

CPU1 Lock

CPU1

CPU2
LATCH IMPLEMENTATIONS

Choice #3: Queue-based Spinlock (MCS)
→ More efficient than mutex, better cache locality
→ Non-trivial memory management
→ Example: `std::atomic_flag`

`Mellor-Crummey and Scott`
LATCH IMPLEMENTATIONS

Choice #3: Queue-based Spinlock (MCS)
 → More efficient than mutex, better cache locality
 → Non-trivial memory management
 → Example: std::atomic_flag

Mellor-Crummey and Scott
LATCH IMPLEMENTATIONS

Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
LATCH IMPLEMENTATIONS

Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks

Latch

- read
  - = 0
  - = 0

- write
  - = 0
  - = 0
LATCH IMPLEMENTATIONS

Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
LATCH IMPLEMENTATIONS

Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
LATCH IMPLEMENTATIONS

Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
Choice #4: Reader-Writer Locks

→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
LATCH IMPLEMENTATIONS

Choice #4: Reader-Writer Locks
→ Allows for concurrent readers
→ Have to manage read/write queues to avoid starvation
→ Can be implemented on top of spinlocks
MODERN INDEXES

Bw-Tree (Hekaton)
Concurrent Skip Lists (MemSQL)
ART Index (HyPer)
Latch-free B+Tree index
→ Threads never need to set latches or block.

Key Idea #1: Deltas
→ No updates in place
→ Reduces cache invalidation.

Key Idea #2: Mapping Table
→ Allows for CAS of physical locations of pages.
# BW-TREE: MAPPING TABLE

## Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

### Index Page

- **Logical Pointer**
- **Physical Pointer**
**BW-TREE: MAPPING TABLE**

**Mapping Table**

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

**Index Page**

102 -> 101 -> 104

**Logical Pointer**

**Physical Pointer**
## BW-TREE: MAPPING TABLE

### Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

### Index Page

- 101
- 102
- 104
BW-TREE: MAPPING TABLE

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Index Page
BW-TREE: MAPPING TABLE

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Index Page

102 104

104 102
# BW-TREE: DELTA UPDATES

## Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer → Page 102

Physical Pointer → Page 102

Source: Justin Levandoski
BW-TREE: DELTA UPDATES

Each update to a page produces a new delta.
Each update to a page produces a new delta.

Source: Justin Levandoski
BW-TREE: DELTA UPDATES

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Each update to a page produces a new delta.

Delta physically points to base page.

Source: Justin Levandoski
Each update to a page produces a new delta.

Delta physically points to base page.

Install delta address in physical address slot of mapping table using CAS.

Source: Justin Levandoski
Each update to a page produces a new delta.

Delta physically points to base page.

Install delta address in physical address slot of mapping table using CAS.

Source: Justin Levandoski
Each update to a page produces a new delta.

Delta physically points to base page.

Install delta address in physical address slot of mapping table using CAS.

Source: Justin Levandoski
BW-TREE: DELTA UPDATES

Each update to a page produces a new delta.

Delta physically points to base page.

Install delta address in physical address slot of mapping table using CAS.

Source: Justin Levandoski
Each update to a page produces a new delta.

Delta physically points to base page.

Install delta address in physical address slot of mapping table using CAS.

Source: Justin Levandoski
Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

△Delete 48

△Insert 50

Page 102
**BW-TREE: SEARCH**

**Mapping Table**

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Traverse tree like a regular B+tree.

**Logical Pointer**

**Physical Pointer**
BW-TREE: SEARCH

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Traverse tree like a regular B+tree.

If mapping table points to delta chain, stop at first occurrence of search key.
**Mapping Table**

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

 Traverse tree like a regular B+tree.

- If mapping table points to delta chain, stop at first occurrence of search key.
- Otherwise, perform binary search on base page.
BW-TREE: CONTENTION UPDATES

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Page 102

Insert 50
**BW-TREE: CONTENTION UPDATES**

### Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Threads may try to install updates to the same state of the page.

- Logical Pointer
- Physical Pointer

Page 102

▲ Insert 50
BW-TREE: CONTENTION UPDATES

**Mapping Table**

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Threads may try to install updates to the same state of the page.

- Logical Pointer
- Physical Pointer

Page 102

Insert 50
Threads may try to install updates to the same state of the page.
BW-TREE: CONTENTION UPDATES

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

△Insert 50
△Insert 16
△Delete 48

Page 102

Threads may try to install updates to same state of the page.
Winner succeeds, any losers must retry or abort
BW-TREE: CONTENTION UPDATES

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

- ▲ Insert 50
- ▲ Delete 48
- ▲ Insert 16

Page 102

Threads may try to install updates to same state of the page.

Winner succeeds, any losers must retry or abort
BW-TREE: CONTENTION UPDATES

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Threads may try to install updates to the same state of the page.

Winner succeeds, any losers must retry or abort.
Thread may try to install updates to same state of the page.

Winner succeeds, any losers must retry or abort.
Record Update Deltas
→ Insert/Delete/Update of record on a page

Structure Modification Deltas
→ Split/Merge information
BW-TREE: CONSOLIDATION

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

- **Insert 55**
- **Delete 48**
- **Insert 50**

Page 102
BW-TREE: CONSOLIDATION

Consolidate updates by creating new page with deltas applied.

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer  
Physical Pointer
Consolidate updates by creating new page with deltas applied.
Consolidate updates by creating new page with deltas applied.
Consolidate updates by creating new page with deltas applied.

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>
Consolidate updates by creating a new page with deltas applied.

CAS-ing the mapping table address ensures no deltas are missed.
Consolidate updates by creating new page with deltas applied.

CAS-ing the mapping table address ensures no deltas are missed.
Consolidate updates by creating new page with deltas applied.

CAS-ing the mapping table address ensures no deltas are missed.

Old page + deltas are marked as garbage.
BW-TREE: GARBAGE COLLECTION

Operations are tagged with an **epoch**
→ Each epoch tracks the threads that are part of it and the objects that can be reclaimed.
→ Thread joins an epoch prior to each operation and post objects that can be reclaimed for the current epoch (not necessarily the one it joined)

Garbage for an epoch reclaimed only when all threads have exited the epoch.
**BW-TREE: GARBAGE COLLECTION**

### Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

### Logical Pointer

#### Physical Pointer

- **Insert 55**
- **Delete 48**
- **Insert 50**

Page 102

New 102

Epoch Table
BW-TREE: GARBAGE COLLECTION

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Insert 55

Delete 48

Insert 50

Page 102

New 102

Insert 55

Delete 48

CPU1

Epoch Table
## BW-TREE: GARBAGE COLLECTION

### Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

### Logical Pointer

#### Physical Pointer

- **Insert 50**
- **Delete 48**
- **Insert 55**

- **Page 102**

- **New 102**

### Epoch Table

- **CPU1**
**Mapping Table**

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

**Logical Pointer**

**Physical Pointer**

**Insert 55**

**Delete 48**

**Insert 50**

**Page 102**

**New 102**

**Epoch Table**

CPU1

CPU2
BW-TREE: GARBAGE COLLECTION

**Mapping Table**

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

**Logical Pointer**

**Physical Pointer**

**Epoch Table**

- CPU1
- CPU2

**Insert 55**

**Delete 48**

**Insert 50**

**Page 102**

**New 102**
BW-TREE: GARBAGE COLLECTION

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Epoch Table

CPU1
- Insert 55
- Delete 48
- Insert 50

CPU2
- Page 102

Logical Pointer

Physical Pointer

New 102
**Mapping Table**

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

**Logical Pointer**

**Physical Pointer**

** BW-TREE: GARBAGE COLLECTION **

**New 102**

**Page 102**

**Epoch Table**

- **CPU1**
  - Insert 55
  - Delete 48
  - Insert 50

- **CPU2**
  - Insert 55
  - Delete 48
  - Insert 50

  Page 102
Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

New 102

Epoch Table

CPU1

Insert 55

Delete 48

Insert 50

Page 102
BW-TREE: GARBAGE COLLECTION

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

New 102

Physical Pointer

Page 102

Epoch Table

- Insert 55
- Delete 48
- Insert 50
- Insert 55
- Delete 48
- Insert 50

Page 102
BW-TREE: GARBAGE COLLECTION

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer:

Physical Pointer:

Epoch Table

New 102

Page 102

- Insert 55
- Delete 48
- Insert 50
BW-TREE: GARBAGE COLLECTION

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

New 102

Epoch Table

Page 102
Page sizes are elastic
→ No hard physical threshold for splitting.
→ This allows the tree to split a page when convenient.

Tree supports “half-split” without latching
→ Install split at child level by creating new page
→ Install new separator key and pointer at parent level
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer
Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer
**BW-TREE: STRUCTURE MODIFICATIONS**

**Mapping Table**

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

---

**Logical Pointer**

**Physical Pointer**
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Split
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Split
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Split
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Split
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>●</td>
</tr>
<tr>
<td>102</td>
<td>●</td>
</tr>
<tr>
<td>103</td>
<td>●</td>
</tr>
<tr>
<td>104</td>
<td>●</td>
</tr>
<tr>
<td>105</td>
<td>●</td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Split
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Split
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Separator

Split

101
102
103
104
105
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

 Logical Pointer

 Physical Pointer

▲ Separator

▲ Split

[∞, 3) [3, 7) [7, ∞)

1 2

3 4

5 6

7 8

101

102

103

104

105
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Separation of intervals: $[\infty, 3)$, $[3, 7)$, $[7, \infty)$
### BW-TREE: STRUCTURE MODIFICATIONS

#### Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

#### Logical Pointer

- Logical Pointer to Physical Pointer

#### Physical Pointer

- Physical Pointer to Logical Pointer

#### Diagram

- Separator
- Split
- [5, 7)
- [∞, 3)
- [3, 7)
- [7, ∞)

- Logical Pointer
- Physical Pointer

- 102
- 103
- 104
- 105
BW-TREE: STRUCTURE MODIFICATIONS

Mapping Table

<table>
<thead>
<tr>
<th>PID</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

Logical Pointer

Physical Pointer

Separator

Split

[∞,3) [3,7) [7,∞) [5,7)
BW-TREE: PERFORMANCE

Bw-Tree  B+Tree  Skip List

Operations/sec (M)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Bw-Tree</th>
<th>B+Tree</th>
<th>Skip List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xbox</td>
<td>10.4</td>
<td>0.56</td>
<td>0.33</td>
</tr>
<tr>
<td>Synthetic</td>
<td>4.23</td>
<td>0.66</td>
<td>0.72</td>
</tr>
<tr>
<td>Deduplication</td>
<td>3.83</td>
<td>1.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.84</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Justin Levandoski
CONCURRENT SKIP LIST

Can implement insert and delete without locks using only CAS operations.

Perform lazy deletion of towers.
SKIP LISTS: INSERT

Levels

- **P=N/4**
  - **P=N/2**
  - **P=N**

```
          V1
          K1
          P=N
          `-----------------------`
        /
      K2      K2
      /        /
    V2        V3
    `--------` `--------`
  K3      K4
  /        /
V4        V6
```

End

- **∞**
  - **∞**
  - **∞**
**Operation**: Insert K5
**Operation:** Insert K5
**SKIP LISTS: INSERT**

**Operation:** Insert K5
**Operation:** Insert K5

**Levels**

- **P=N**
  - K1, V1
- **P=N/2**
  - K2, V2
- **P=N/4**
  - K3, V3
- **K4, V4**

**End**

- **∞**
**Operation:** Insert K5
Operation: Insert K5
**Operation:** Insert K5
**Operation:** Insert K5
Operation: Insert K5
**SKIP LISTS: INSERT**

**Operation:** Insert K5

---

**Levels**

- **P=N**
  - **K1**
    - **V1**
- **P=N/2**
  - **K2**
    - **V2**
- **P=N/4**
  - **K3**
    - **V3**
  - **K4**
    - **V4**
  - **K5**
    - **V5**
- **P=N**
  - **K6**
    - **V6**

**End**

- **∞**
**Levels**

- $P = N$ (Level 1)
- $P = N/2$ (Level 2)
- $P = N/4$ (Level 3)

**Operation:** Insert $K5$

**End**

- $\infty$ (Level 3)
- $\infty$ (Level 2)
- $\infty$ (Level 1)
**Operation:** Insert K5
**Operation:** Insert K5

Levels

- **P=N/4**
- **P=N/2**
- **P=N**

**End**

- **∞**
**Skip Lists: Delete**

**Operation:** Delete K5

```
Levels

- P=N
- P=N/2
- P=N/4

K1  V1
  ↓   ↓
K2  V2
  ↓   ↓
K3  V3
  ↓   ↓
K4  V4
  ↓   ↓
K5  V5
  ↓   ↓
K6  V6
  ↓   ↓
End

∞

Operation: Delete K5
```
**Operation:** Delete K5

**Levels:**
- $P = N$ (Level 1)
- $P = N/2$ (Level 2)
- $P = N/4$ (Level 3)

**End:**
- $\infty$ (Level 4)

**Del true** indicates the deletion operation is successful at that level.
**SKIP LISTS: DELETE**

**Operation:** Delete K5

- **Levels**
  - $P=N$
  - $P=N/2$
  - $P=N/4$

- **Operation Details**
  - Delete K5
  - $\text{Del} \, \text{false}$

- **End**
  - $\infty$
SKIP LISTS: DELETE

Operation: Delete K5

Levels

P=N

P=N/4

P=N/2

K1

V1

Del false

K2

V2

Del false

K3

V3

Del false

K4

V4

Del false

K5

Del true

End

P=N

∞

P=N/2

∞

P=N/4

∞

∞

CMU 15-721 (Spring 2016)
SKIP LISTS: DELETE

Operation: Delete K5

Levels

P=N

K1
V1 Del false

K2
V2 Del false

K3
V3 Del false

K4
V4 Del false

K5
V5 Del true

K6
V6 Del false

P=N/4

P=N/2

End

K5

∞
**Operation:** Delete K5

**Levels**

- **P=N**
  - K1
    - V1
     - Del false
  - K2
    - V2
     - Del false
  - K3
    - V3
     - Del false
  - K4
    - V4
     - Del false
  - K5
    - Del true
     - V5
     - Del false

**End**

- ∞
**Operation:** Delete K5

**Levels**

- **P=N/4**
  - K1
  - V1: Del false
- **P=N/2**
  - K2
  - V2: Del false
- **P=N**
  - K3
  - V3: Del false
  - K4
  - V4: Del false
  - K5
  - V5: Del true
  - K6
  - V6: Del false

**End**

K5
**Operation**: Delete K5

**Levels**
- **P=N**: K1 → V1
- **P=N/2**: K2 → V2
- **P=N/4**: K3 → V3

**End**
- ∞

**Operation Details**
- Delete K5
  - Del: false
  - Del: false
  - Del: false
  - Del: false
  - Del: false
  - Del: true
**Operation:** Delete K5

**Levels**

- **P=N**: Nodes K1 and V1 with Del = false.
- **P=N/2**: Nodes K2 and V2 with Del = false.
- **P=N/4**: Nodes K3 and V3 with Del = false.
- **P=N/8**: Nodes K4 and V4 with Del = false.

**End**

Nodes K5 and K6 with Del = false.

Nodes K7 and V6 with Del = false.
SKIP LISTS: REVERSE SEARCH

Levels

- $P = N/4$
- $P = N/2$
- $P = N$

End

- $\infty$
- $\infty$
- $\infty$

Source: Mark Papadakis
Skip Lists: Reverse Search

**Txn #1: Find [K4,K2]**

Source: Mark Papadakis
Levels

Txn #1: Find [K4,K2]

Source: Mark Papadakis
SKIP LISTS: REVERSE SEARCH

Txn #1: Find [K4,K2]
**Txn #1: Find [K4,K2]**

```
Levels

- P=N
- P=N/2
- P=N/4

K2 < K5

End

∞
```

Source: Mark Papadakis
**Levels**

- **K2 < K5**
  - $P = N/4$

- **K2 = K2**
  - $P = N/2$

- **K1, V1**
- **K2, V2**
- **K3, V3**
- **K4, V4**
- **K5, V5**
- **K6, V6**

**End**

**SKIP LISTS: REVERSE SEARCH**

**Txn #1: Find [K4, K2]**

- $K2 < K5$
- $K2 = K2$

Source: Mark Papadakis
Levels

```
$K_1$ $V_1$
$P=N$

$K_2$ $V_2$
$P=N/2$

$K_2$ $V_2$
$P=N/4$

$K_2 < K_5$

$K_2 = K_2$

$K_4$ $V_4$

$K_3$ $V_3$

$K_4$ $V_4$

$K_5$ $V_5$

$K_6$ $V_6$

End

```

**Txn #1:** Find $[K_4,K_2]$

Source: Mark Papadakis
**SKIP LISTS: REVERSE SEARCH**

**Txn #1: Find [K4,K2]**

Source: Mark Papadakis
Levels

Txn #1: Find [K4, K2]

Source: Mark Papadakis
SKIP LISTS: REVERSE SEARCH

Txn #1: Find [K4, K2]

Source: Mark Papadakis
**Skip Lists: Reverse Search**

**Txn #1: Find \([K4, K2]\)**

**Stack:**

Levels

- **K2 < K5**
  - \(P = N/4\)
  - \(K1 < V1\)

- **K2 = K2**
  - \(P = N/2\)
  - \(K2 < V2\)

- **K2 < K4**
  - \(P = N\)
  - \(K2 < V3\)

- **K4**
  - \(K4 < V4\)

- **K5**
  - \(K5 < V5\)

- **K6**
  - \(K6 < V6\)

End

\(\infty\)

Source: Mark Papadakis
**Skip Lists: Reverse Search**

**Txn #1: Find [K4,K2]**

**Stack:**

- **Levels**
  - **P=N**
  - **P=N/2**
  - **P=N/4**

- **K2<K5**
- **K2=K2**
- **K2<K4**

- **Stack:**
  - **K1**
  - **V1**
  - **K2**
  - **V2**
  - **K3**
  - **V3**
  - **K4**
  - **V4**
  - **K5**
  - **V5**
  - **K6**
  - **V6**

- **End**
  - **K2**
  - **∞**

**Source:** Mark Papadakis
**SKIP LISTS: REVERSE SEARCH**

**Txn #1: Find [K4, K2]**

**Stack:**

1. **Levels**
   - **P=N**
     - **K1**
     - **V1**
   - **P=N/2**
     - **K2**
     - **V2**
   - **P=N/4**
     - **K2**
     - **K3**

2. **Stack:**
   - **K2 < K4**
   - **K2 = K2**
   - **K2 < K5**

3. **End:**
   - **K3**
   - **K2**
   - **∞**
   - **∞**
   - **∞**

---

Source: Mark Papadakis

CMU 15-721 (Spring 2016)
**SKIP LISTS: REVERSE SEARCH**

**Txn #1: Find [K4,K2]**

**Stack:**

1. K4
2. K2
3. K3
4. K2

**Levels**

- **P=N**
  - **K1**
  - **V1**
- **P=N/2**
  - **K2**
  - **V2**
  - **K2**
- **P=N/4**
  - **K2**
  - **V3**
  - **K3**
  - **K2**

**Source:** Mark Papadakis
**Levels**

<table>
<thead>
<tr>
<th>Levels</th>
<th>P=N</th>
<th>K1</th>
<th>V1</th>
<th>K2</th>
<th>V2</th>
<th>K3</th>
<th>V3</th>
<th>K4</th>
<th>V4</th>
<th>K5</th>
<th>V5</th>
<th>K6</th>
<th>V6</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K2 &lt; K5$</td>
<td>$P=N/4$</td>
<td>$K2$</td>
<td>$V2$</td>
<td>$K2$</td>
<td>$V2$</td>
<td>$K3$</td>
<td>$V3$</td>
<td>$K4$</td>
<td>$V4$</td>
<td>$K5$</td>
<td>$V5$</td>
<td>$K6$</td>
<td>$V6$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$K2 = K2$</td>
<td>$P=N/2$</td>
<td>$K2$</td>
<td>$V2$</td>
<td>$K2$</td>
<td>$V2$</td>
<td>$K3$</td>
<td>$V3$</td>
<td>$K4$</td>
<td>$V4$</td>
<td>$K5$</td>
<td>$V5$</td>
<td>$K6$</td>
<td>$V6$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$K2 &lt; K4$</td>
<td>$P=N$</td>
<td>$K1$</td>
<td>$V1$</td>
<td>$K2$</td>
<td>$V2$</td>
<td>$K3$</td>
<td>$V3$</td>
<td>$K4$</td>
<td>$V4$</td>
<td>$K5$</td>
<td>$V5$</td>
<td>$K6$</td>
<td>$V6$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

**Txn #1:** Find $[K4, K2]$

**Stack:**

| K4 | K3 | K2 |

**End**

Source: Mark Papadakis
**SKIP LISTS: REVERSE SEARCH**

**Levels**

- **P=N**: $K_1, V_1$
- **P=N/2**: $K_2, V_2$
- **P=N/4**: $K_3, V_3$

**Txn #1: Find \([K_4, K_2]\\)**

- **Stack:** $K_4, K_3, K_2$

**End**

- **∞**

Source: Mark Papadakis
**SKIP LISTS: REVERSE SEARCH**

**Txn #1: Find [K4, K2]**

**Stack:**

```
K5
K4
K3
K2
```

**Levels**

1. **P=N**
   - **K1**
     - **V1**

2. **P=N/2**
   - **K2**
     - **V2**

3. **P=N/4**
   - **K2 < K5**
   - **K2 = K2**
   - **K2 < K4**

**Source:** Mark Papadakis
**Levels**

- **K2 < K5**
  - P = N/4

- **K2 = K2**
  - P = N/2

- **K2 < K4**
  - P = N

**Txn #1: Find [K4, K2]**

- Stack: K4, K3, K2

Source: Mark Papadakis
ADAPATIVE RADIX TREE (ART)

Uses a digital representation of keys to examine key prefixes one-by-one instead of comparing the entire key.

Radix trees properties:
→ The height of the tree depends on the length of keys.
→ Does not require rebalancing
→ The path to a leaf node represents the key of the leaf
→ Keys are stored implicitly and can be reconstructed from paths.
TRIE VS. RADIX TREE

**Trie**

```
   H
  /|
 E A
/ |/
L V T
/ |/
L E
```

**Radix Tree**

```
   H
  /|
 ELLO
 /  |
 A  VE
```

**Keys:** hello, hat, have
ART INDEX: MODIFICATIONS
Operation: Insert hair
**Operation:** Insert hair
Operation: Insert hair

Operation: Delete hat, have
Operation: Insert hair
Operation: Delete hat, have
ART INDEX: MODIFICATIONS

Operation: Insert hair
Operation: Delete hat, have
**Operation:** Insert hair

**Operation:** Delete hat, have
Operation: Insert hair
Operation: Delete hat, have
Operation: Insert hair
Operation: Delete hat, have
ART INDEX: MODIFICATIONS

Operation: Insert hair
Operation: Delete hat, have

Note: The ART index described in 2013 is **not** latch-free.
Not all attribute types can be decomposed into binary comparable digits for a radix tree.

→ **Unsigned Integers:** Byte order must be flipped for little endian machines.

→ **Signed Integers:** Flip two’s-complement so that negative numbers are smaller than positive.

→ **Floats:** Classify into group (neg vs. pos, normalized vs. denormalized), then store as unsigned integer.

→ **Compound:** Transform each attribute separately.
PARTING THOUGHTS

Bw-Tree is probably the most dank database data structure in recent years.

Skip List is really easy to implement.
NEXT CLASS

Indexing for OLAP workloads.
→ More from Microsoft Research...

Project #2 Announcement