# 15-721 DATABASE SYSTEMS

## Lecture #08 – Latch-free OLTP Indexes (Part II)

@Andy_Pavlo // Carnegie Mellon University // Spring 2017

# TODAY'S AGENDA

Bw-Tree Index

ART Index

Profiling in Peloton

# OBSERVATION

We cannot have reverse pointers in a latch-free concurrent Skip List because CaS can only update a single address at a time.

# BW-TREE

Latch-free B+Tree index

→ Threads never need to set latches or block.

**Key Idea #1: Deltas**

→ No updates in place

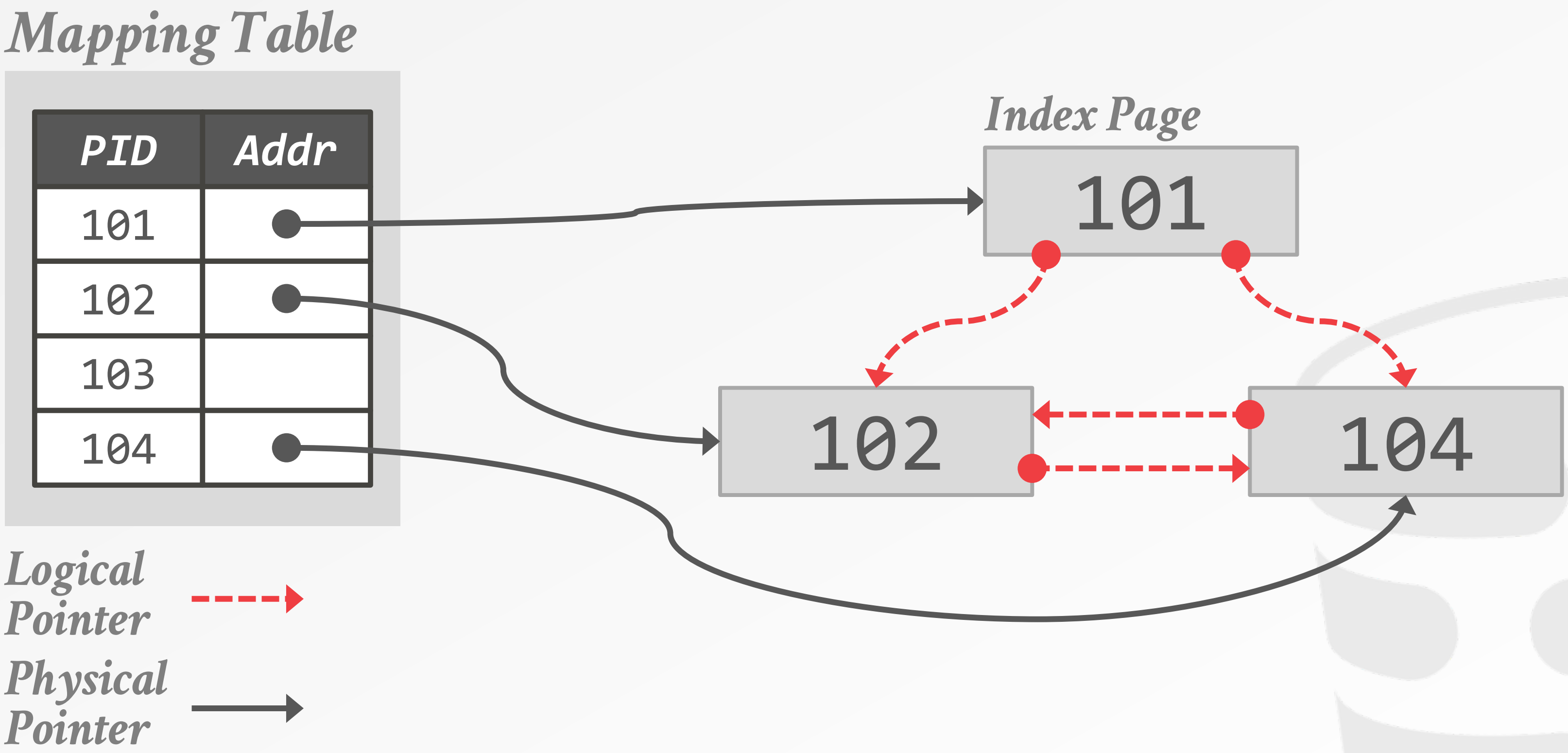
→ Reduces cache invalidation.

**Key Idea #2: Mapping Table**

→ Allows for CAS of physical locations of pages.

THE BW-TREE: A B-TREE FOR NEW HARDWARE
*ICDE 2013*

# BW-TREE: MAPPING TABLE

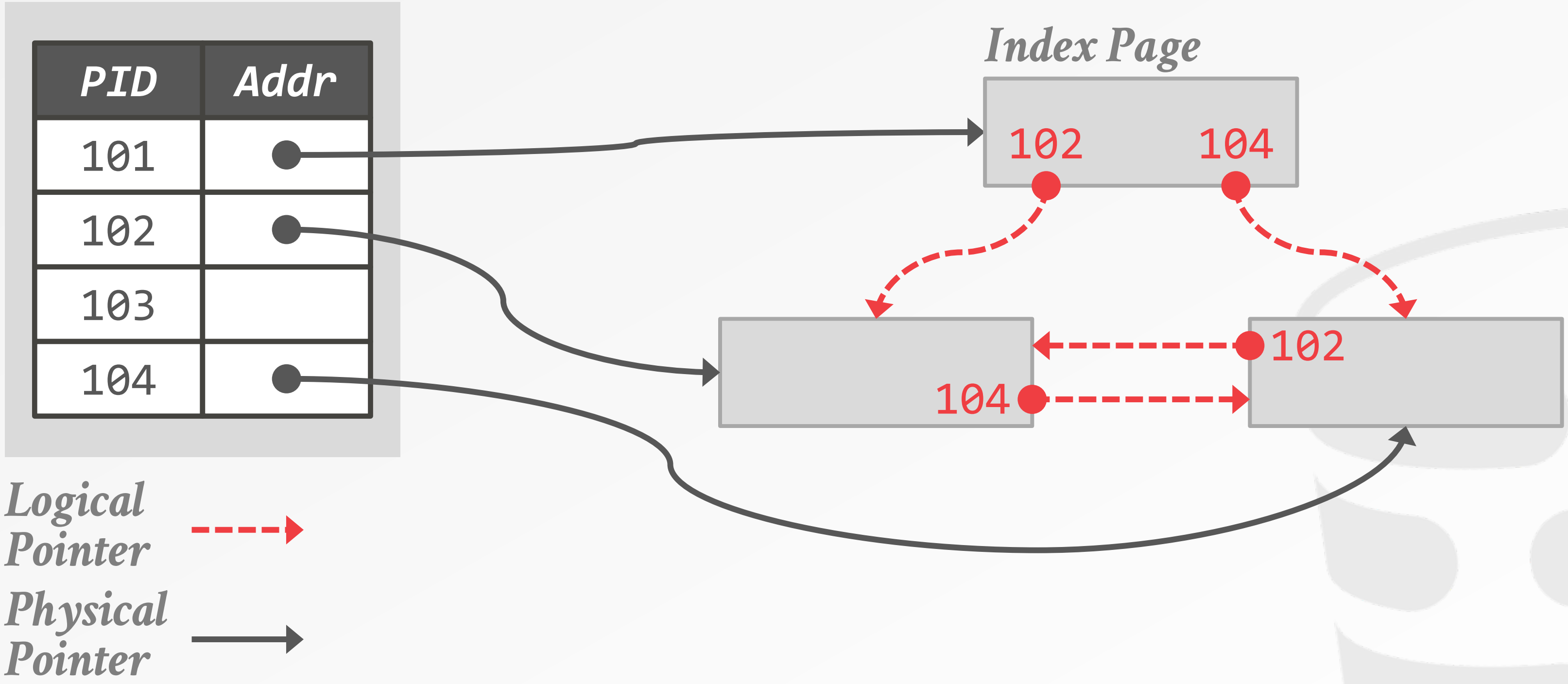


*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | |
| 104 | ● |

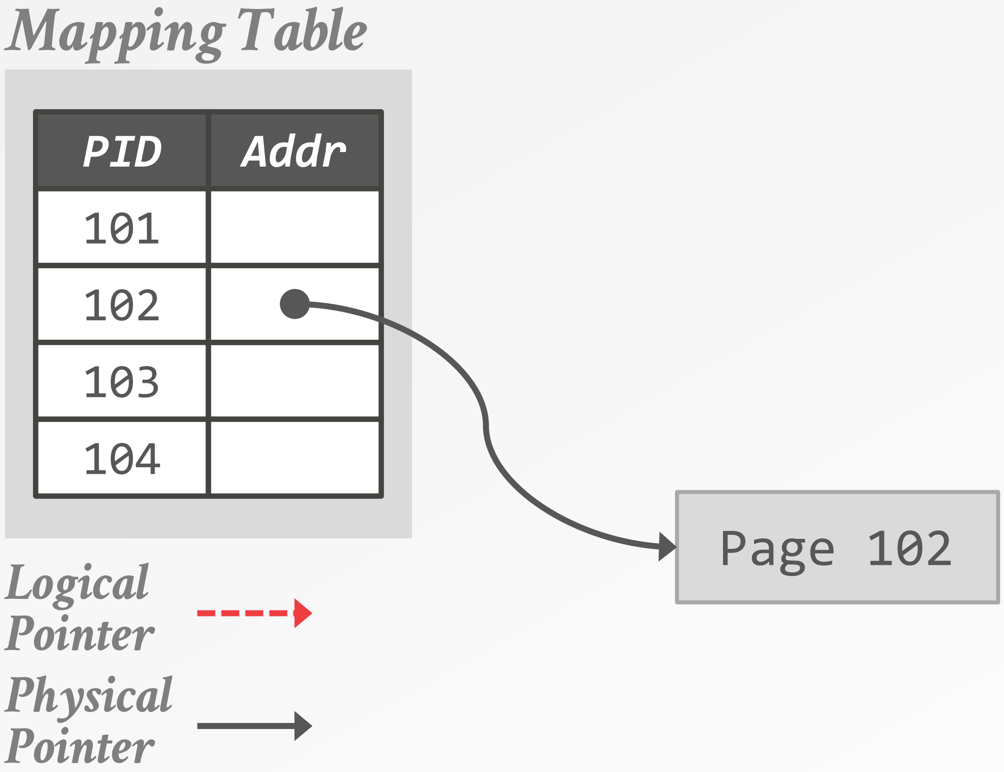
# BW-TREE: MAPPING TABLE

*Mapping Table*



| PID | Addr |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | |
| 104 | ● |

# BW-TREE: DELTA UPDATES

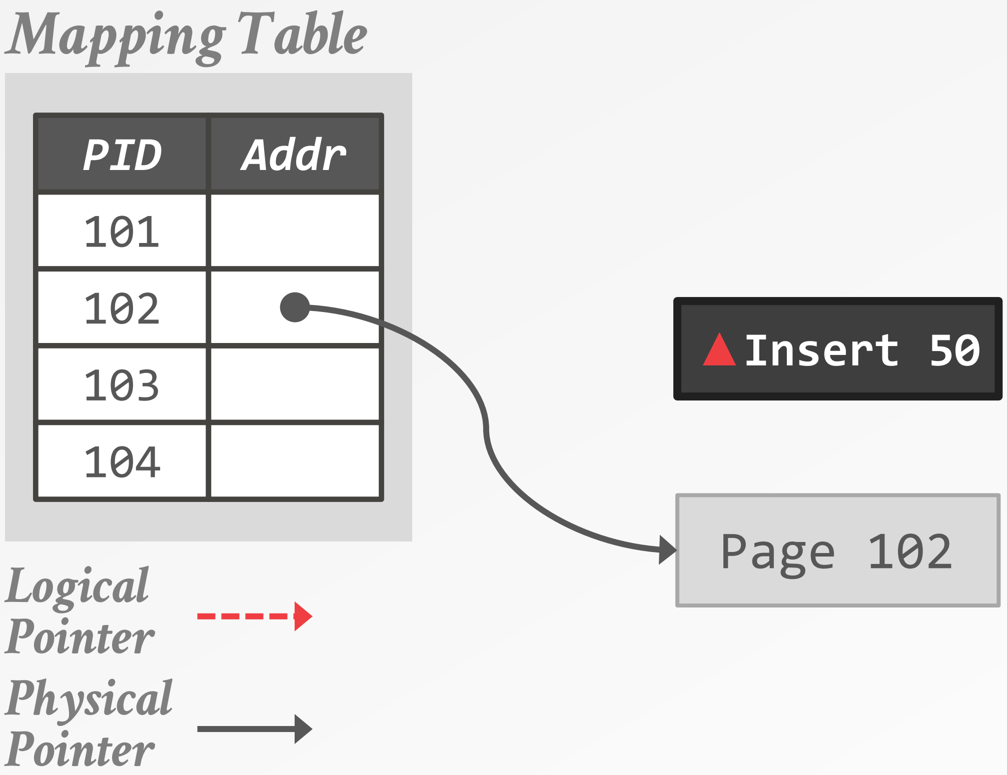


| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Each update to a page produces a new delta.

Source: Justin Levandoski

# BW-TREE: DELTA UPDATES



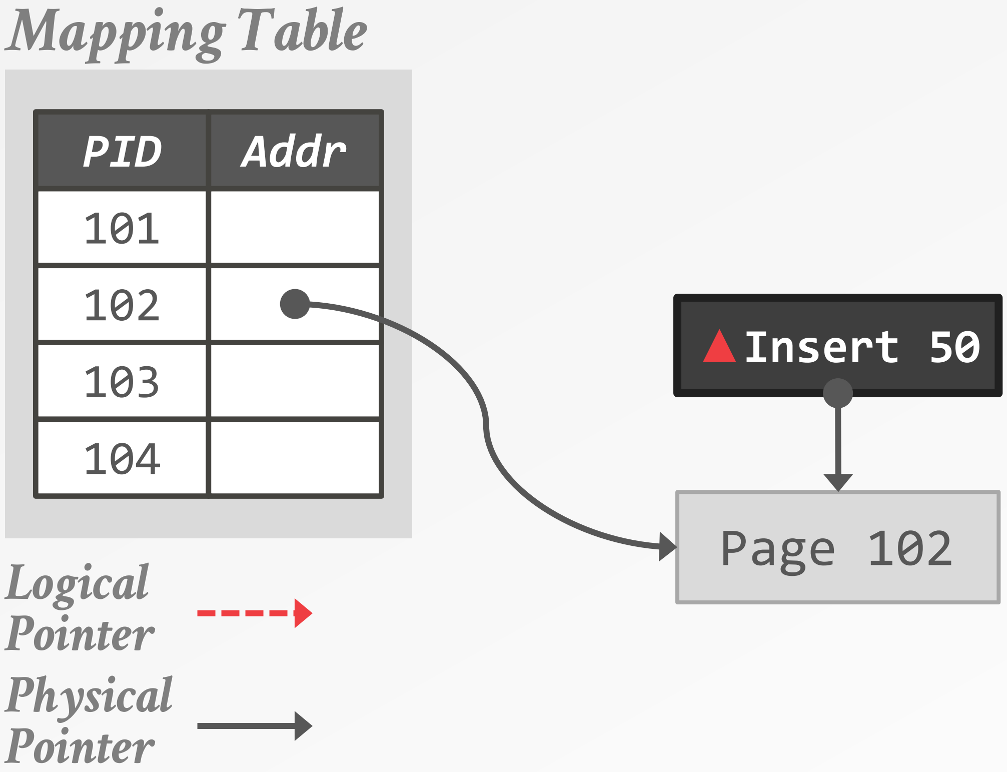
*Mapping Table*

| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Each update to a page produces a new delta.

Source: Justin Levandoski

# BW-TREE: DELTA UPDATES



Mapping Table

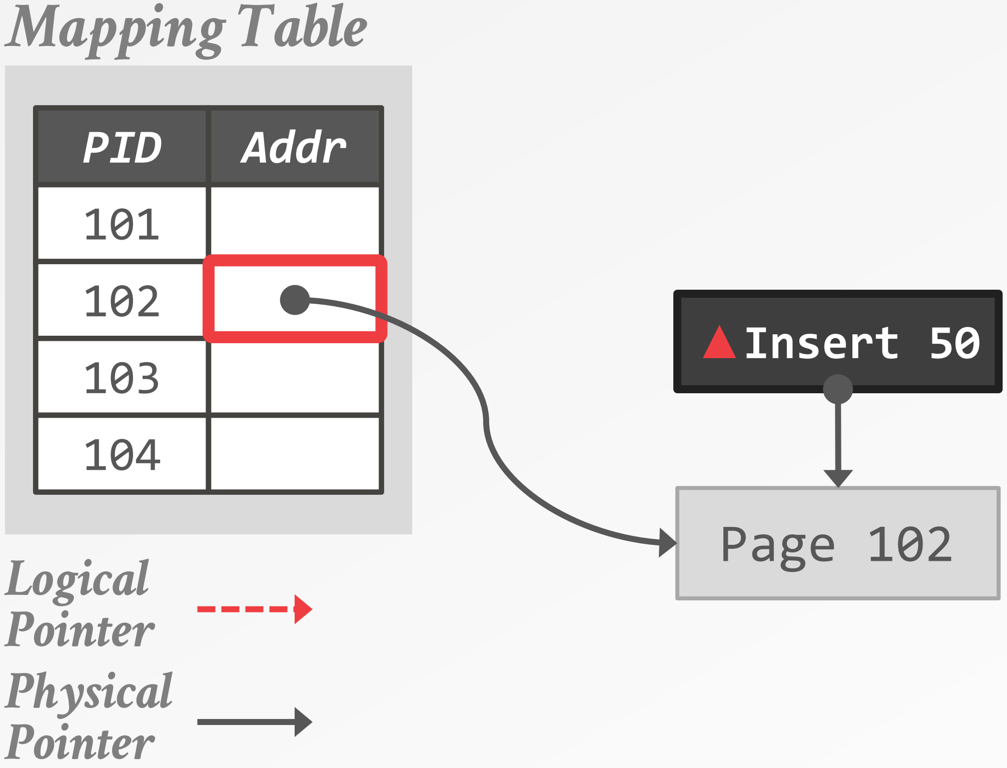
| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Each update to a page produces a new delta.

Delta physically points to base page.

Source: Justin Levandoski

# BW-TREE: DELTA UPDATES



*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

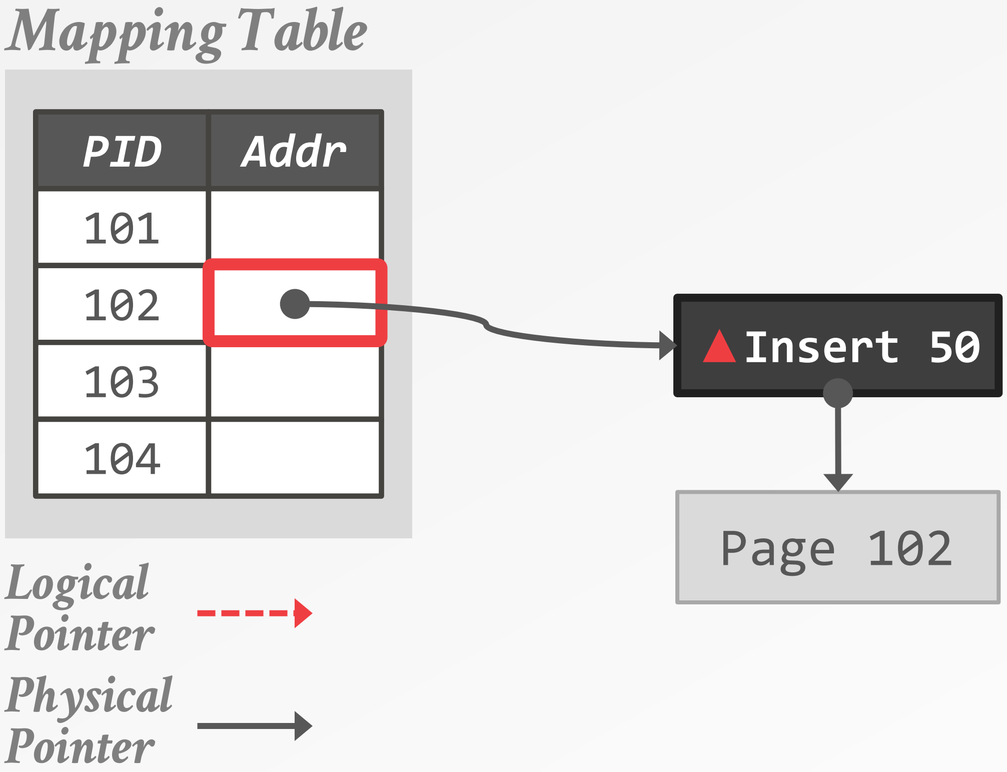
Each update to a page produces a new delta.

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



*Mapping Table*

| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Each update to a page produces a new delta.

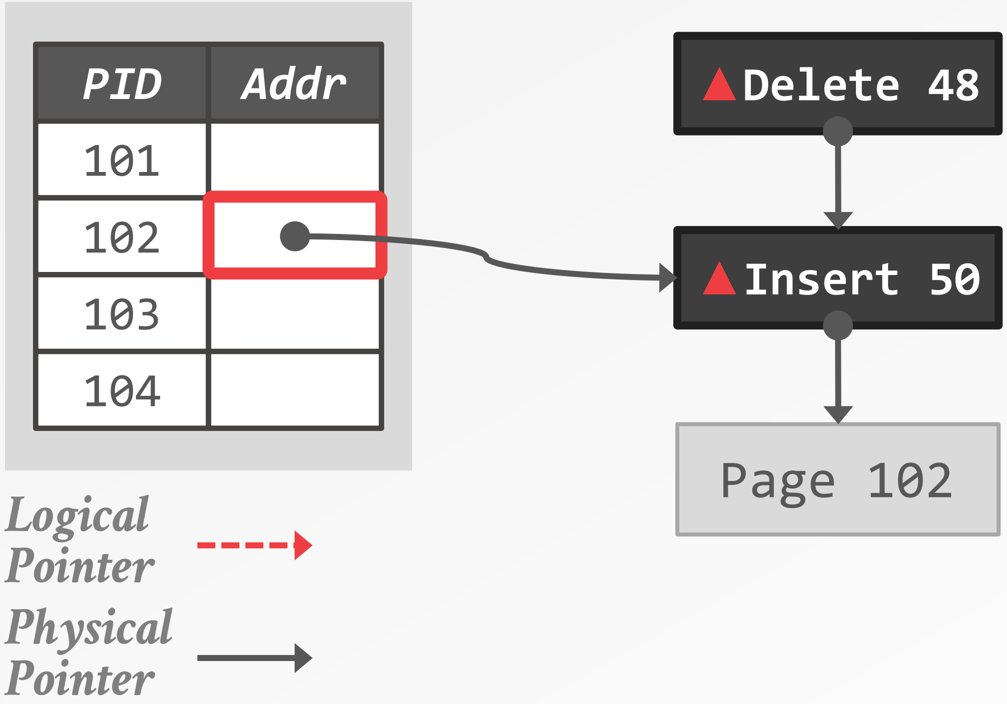
Delta physically points to base page.

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

Source: Justin Levandoski

# BW-TREE: DELTA UPDATES

*Mapping Table*



| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Each update to a page produces a new delta.

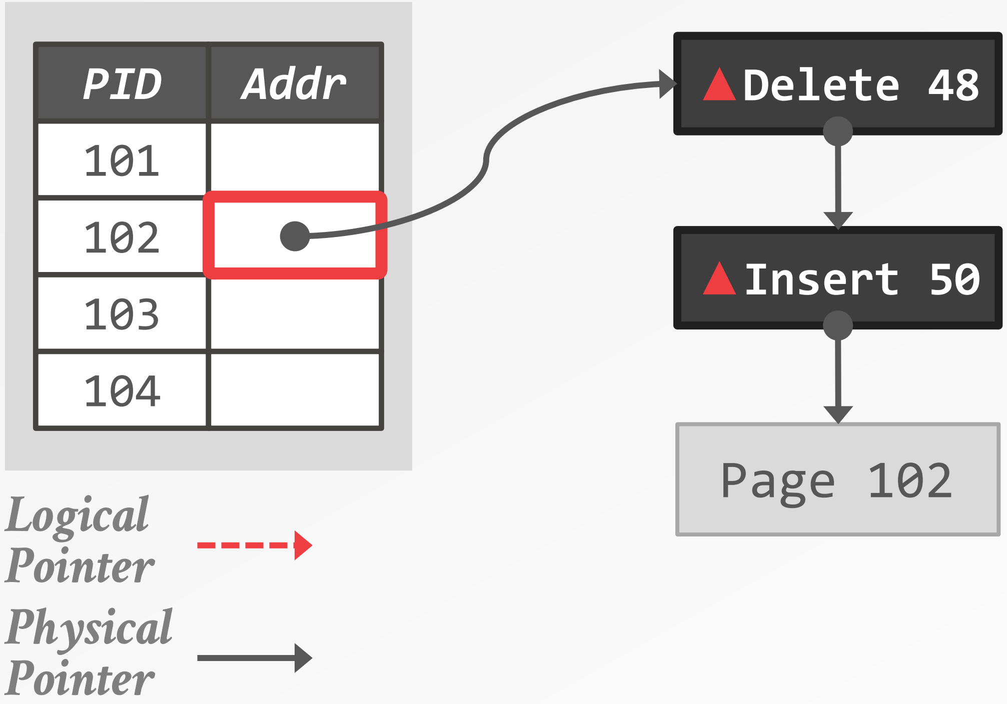
Delta physically points to base page.

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

Source: Justin Levandoski

# BW-TREE: DELTA UPDATES

*Mapping Table*



| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

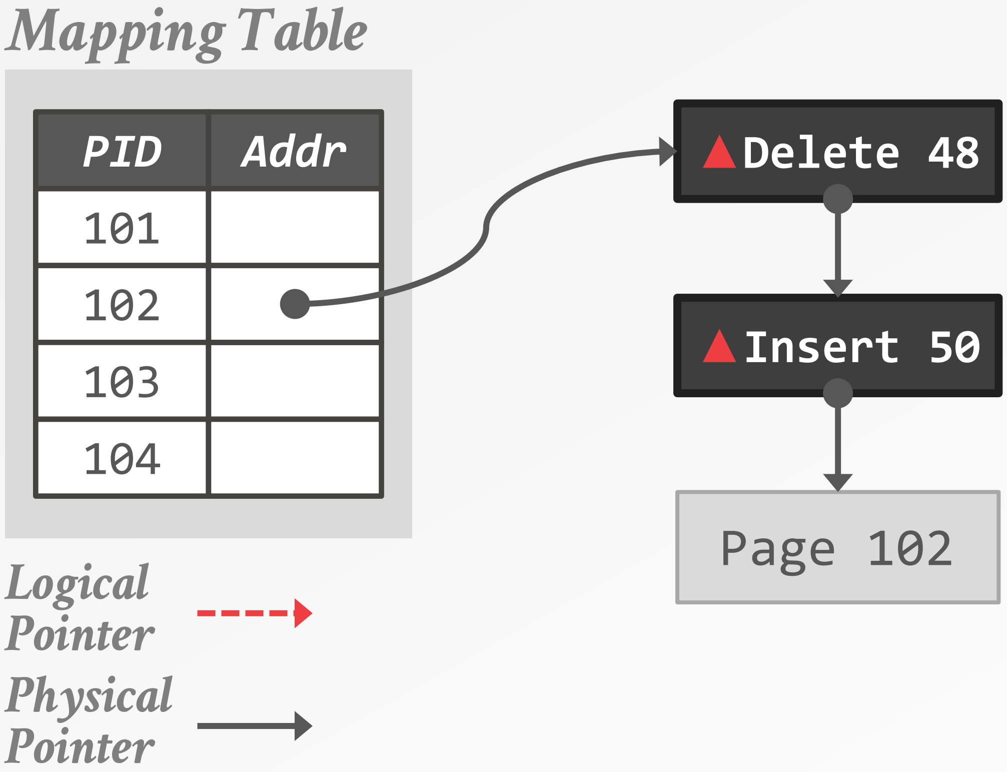
Each update to a page produces a new delta.

Delta physically points to base page.

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

Source: Justin Levandoski

# BW-TREE: SEARCH

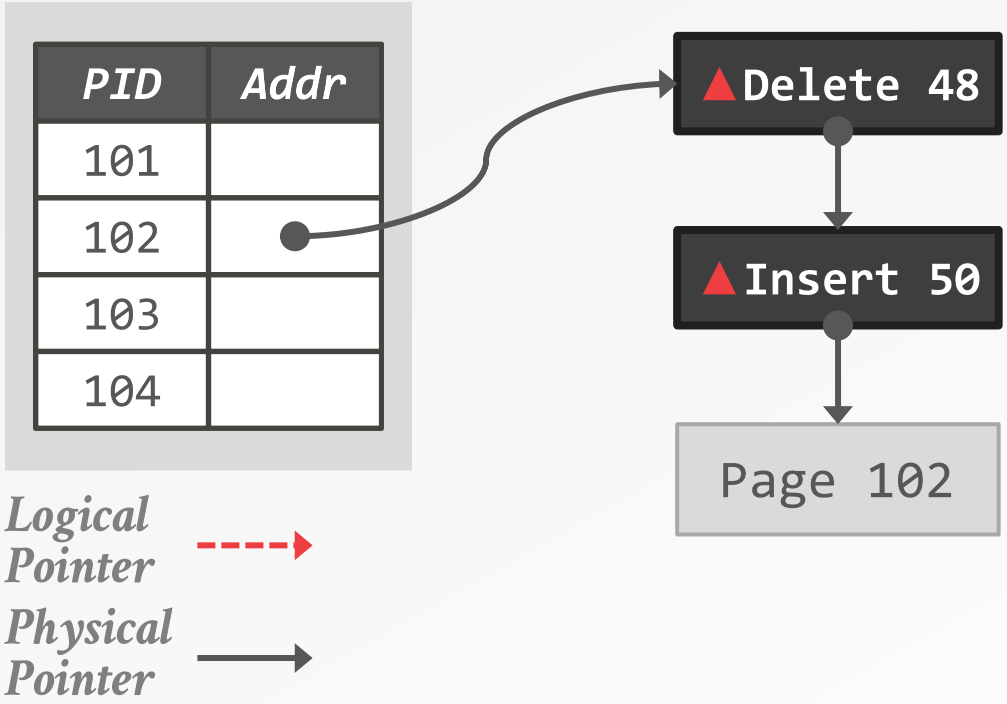


| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Traverse tree like a regular B+tree.

# BW-TREE: SEARCH

*Mapping Table*

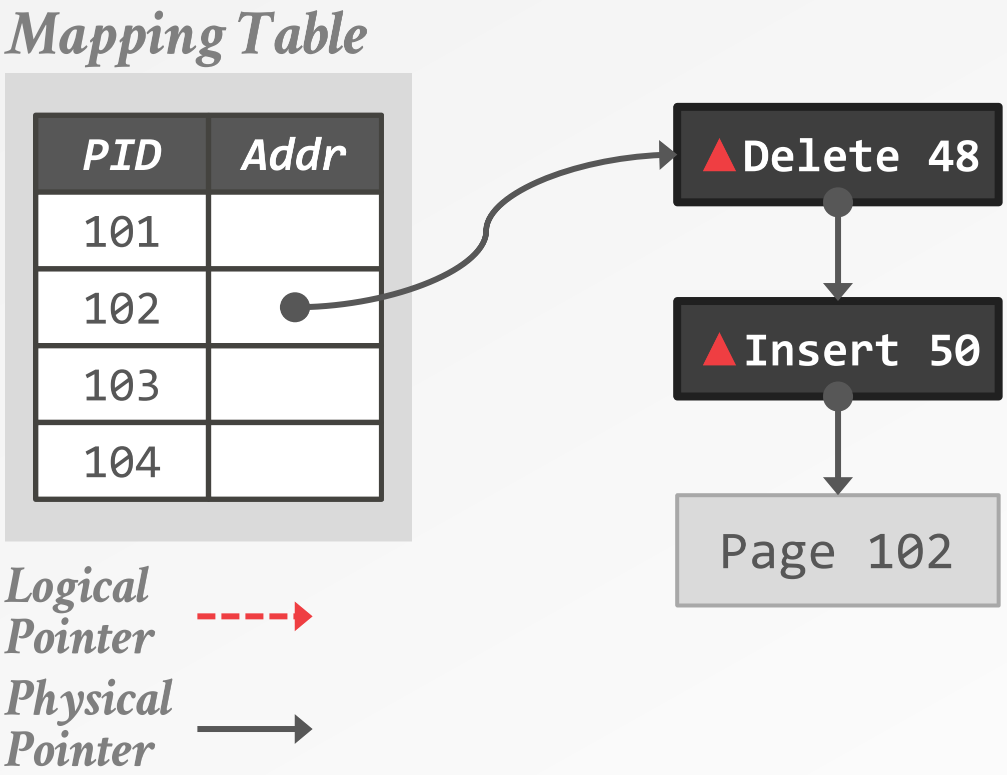


| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Traverse tree like a regular B+tree.

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

# BW-TREE: SEARCH



*Mapping Table*

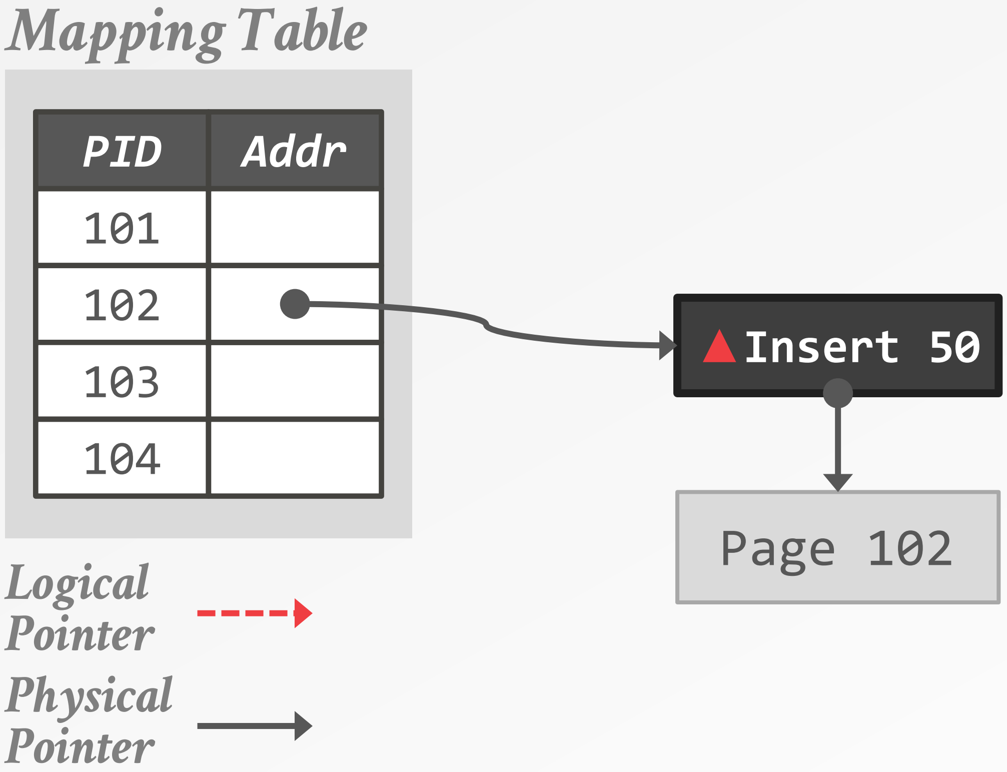
| *PID* | *Addr* |
|---|---|
| 101 | |
| 102 | |
| 103 | |
| 104 | |

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

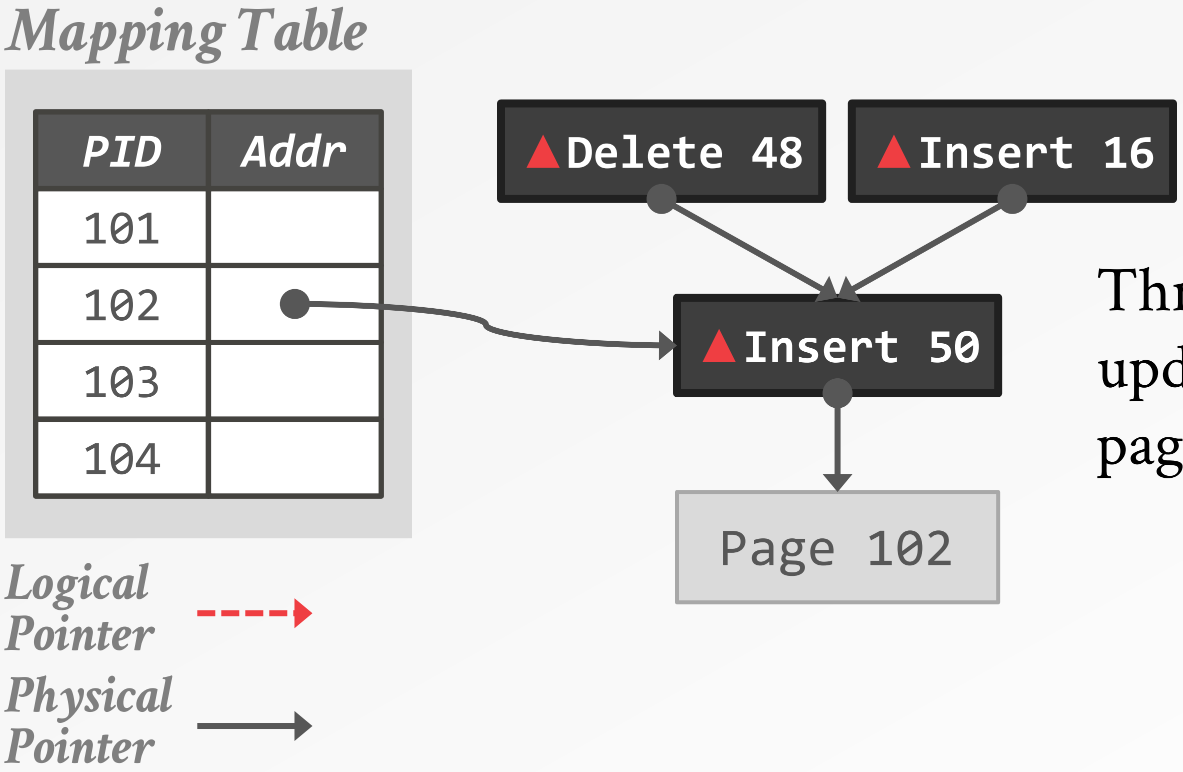


*Mapping Table*

| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

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

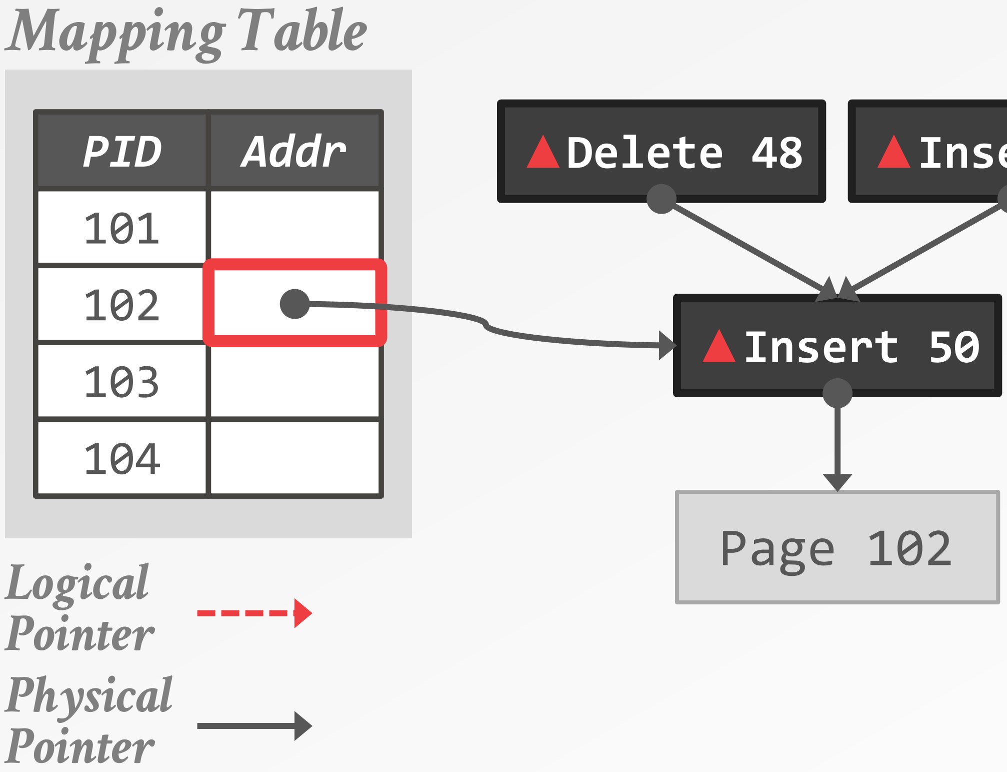
# BW-TREE: CONTENTION UPDATES



| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

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

# BW-TREE: CONTENTION UPDATES



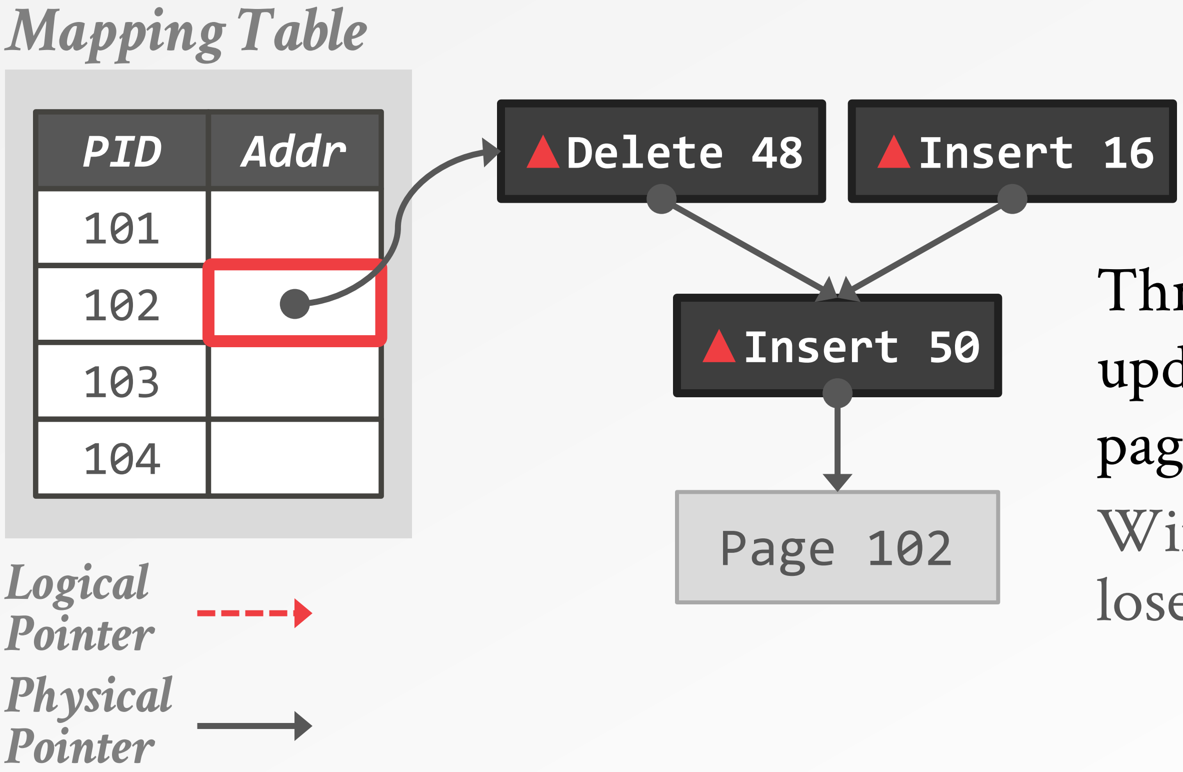
*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

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

Winner succeeds, any losers must retry or abort

# BW-TREE: CONTENTION UPDATES

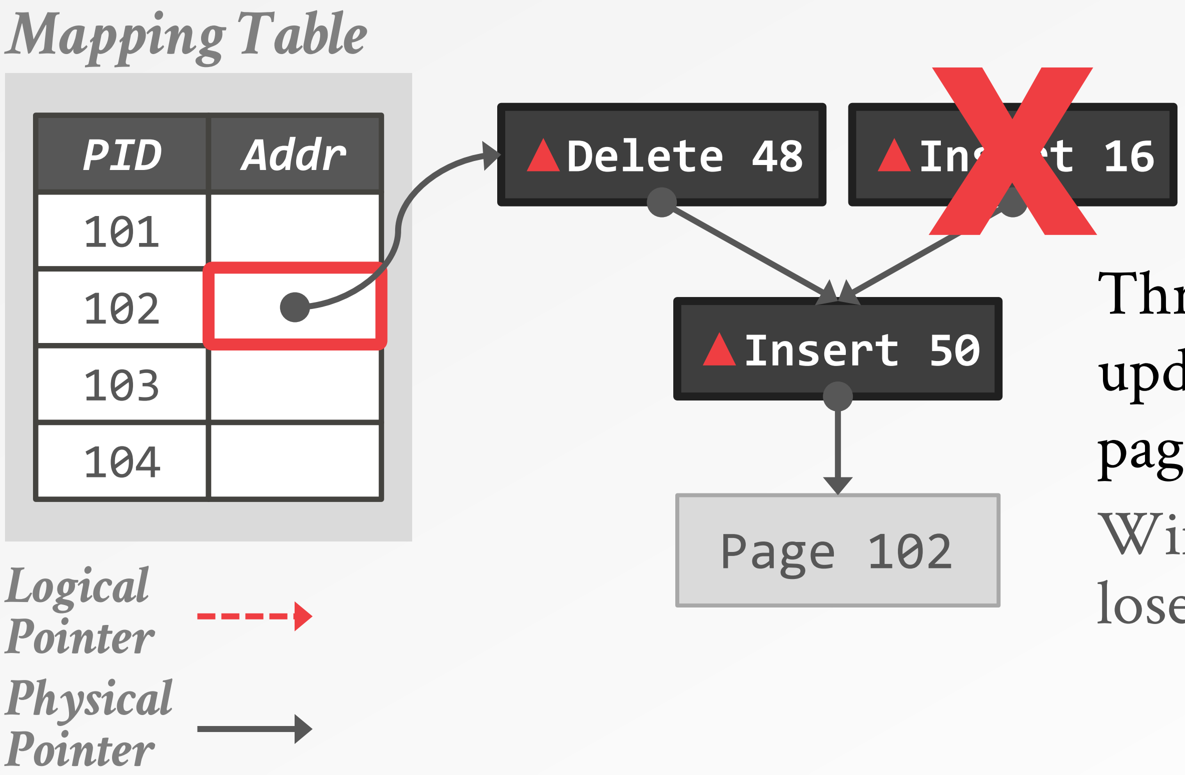


| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

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*

| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

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

Winner succeeds, any losers must retry or abort

# BW-TREE: DELTA TYPES

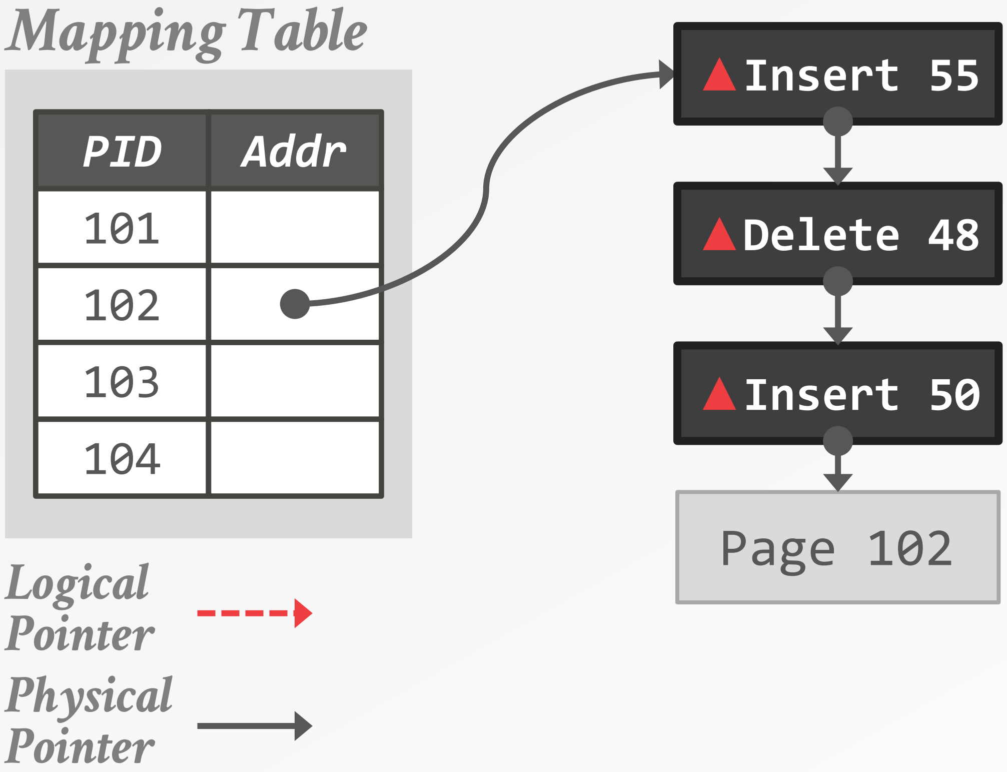
**Record Update Deltas**

→ Insert/Delete/Update of record on a page

**Structure Modification Deltas**

→ Split/Merge information

# BW-TREE: CONSOLIDATION

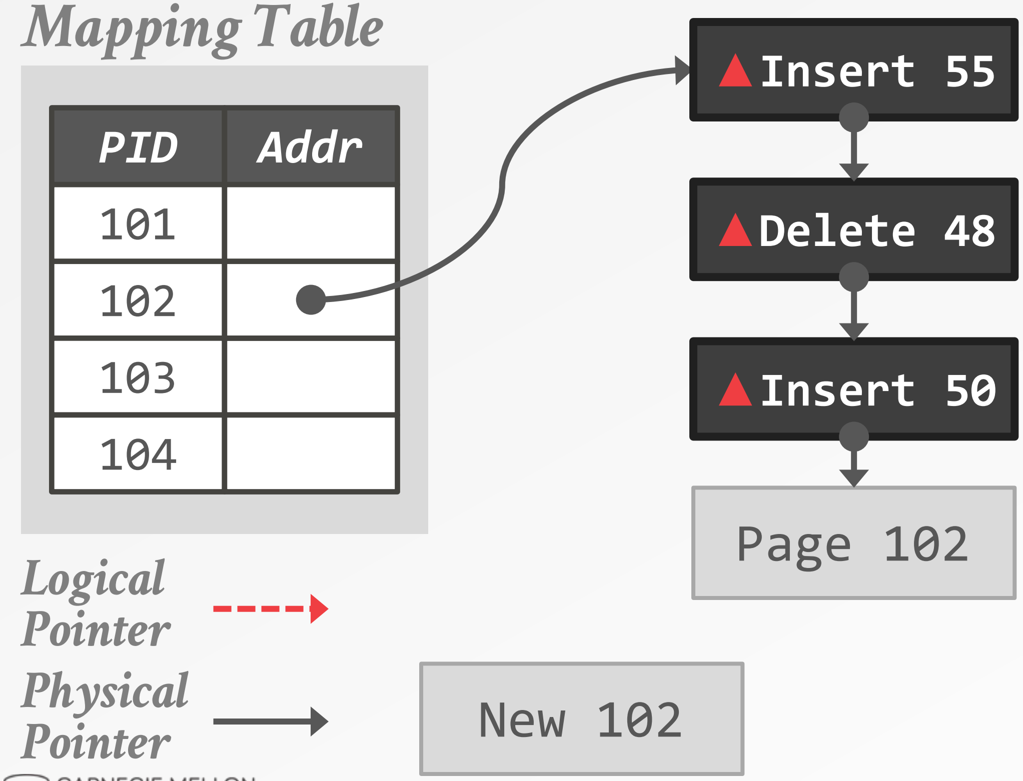


*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Consolidate updates by creating new page with deltas applied.

# BW-TREE: CONSOLIDATION

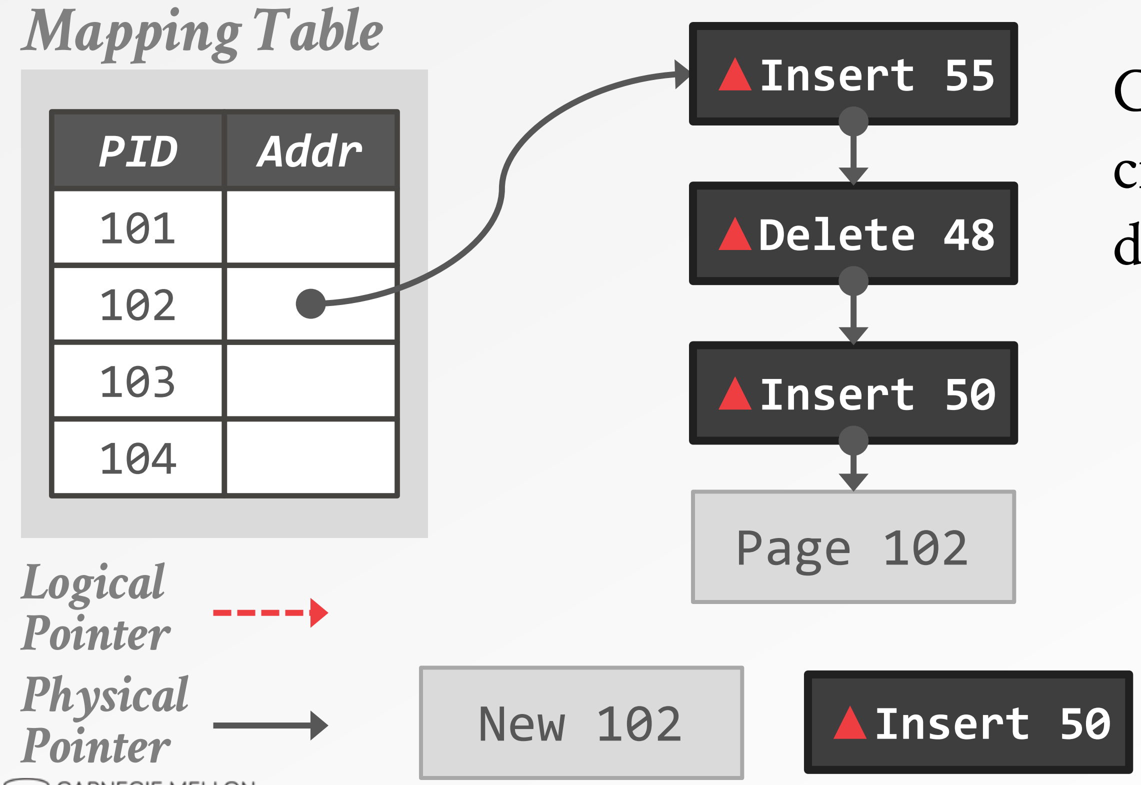


*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Consolidate updates by creating new page with deltas applied.

# BW-TREE: CONSOLIDATION

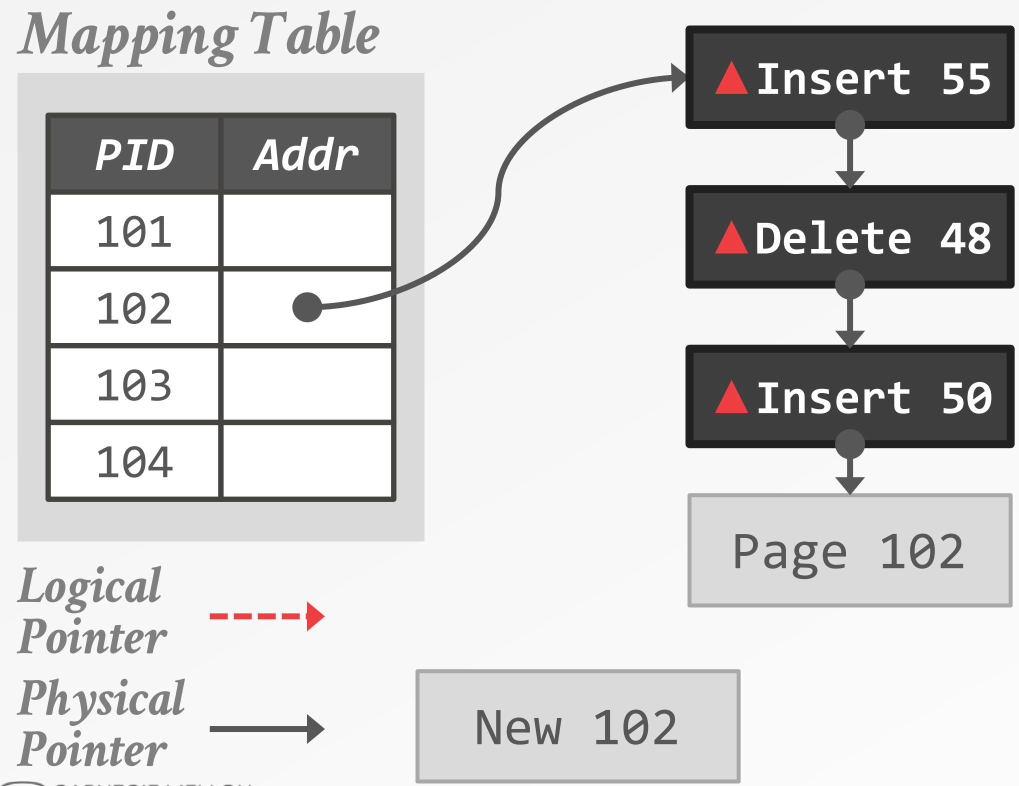


*Mapping Table*

| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Consolidate updates by creating new page with deltas applied.

# BW-TREE: CONSOLIDATION



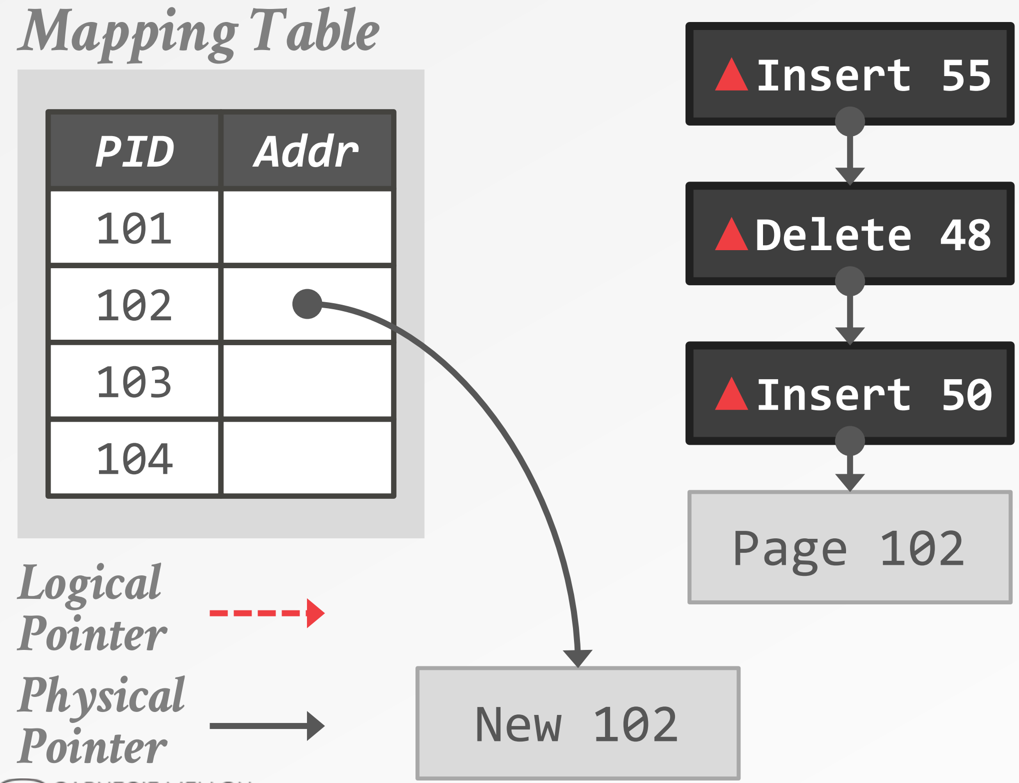
*Mapping Table*

| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Consolidate updates by creating new page with deltas applied.

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

# BW-TREE: CONSOLIDATION



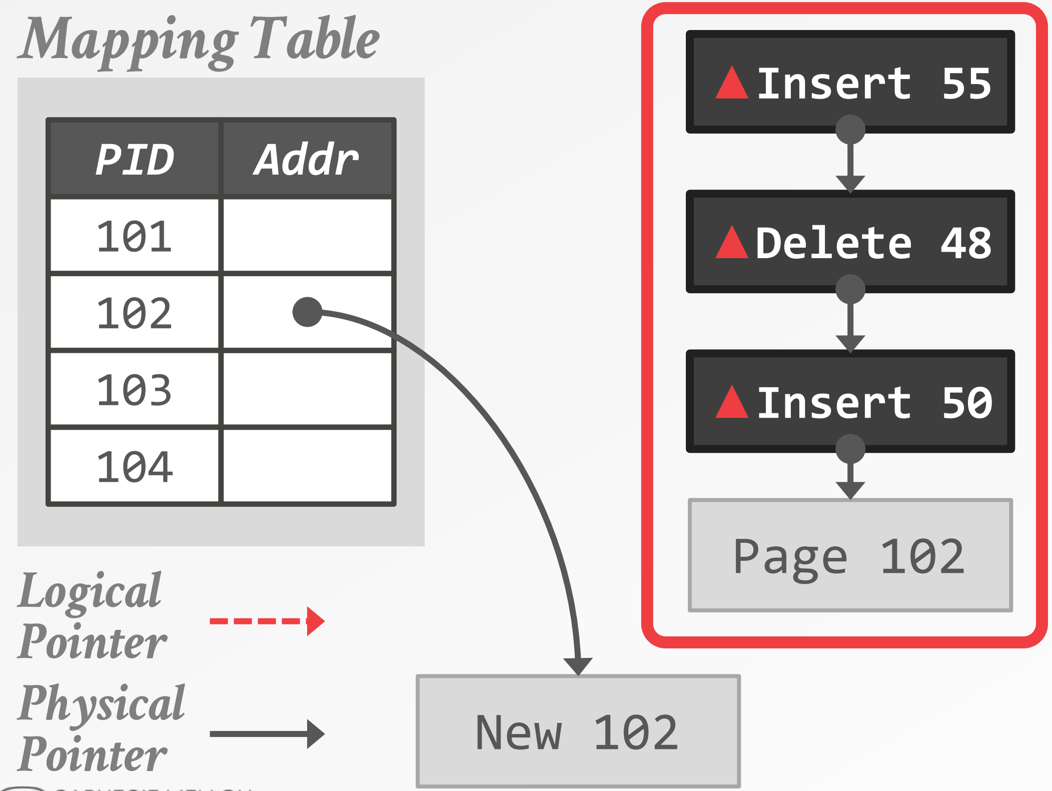
Mapping Table

| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Consolidate updates by creating new page with deltas applied.

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

# BW-TREE: CONSOLIDATION



*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

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

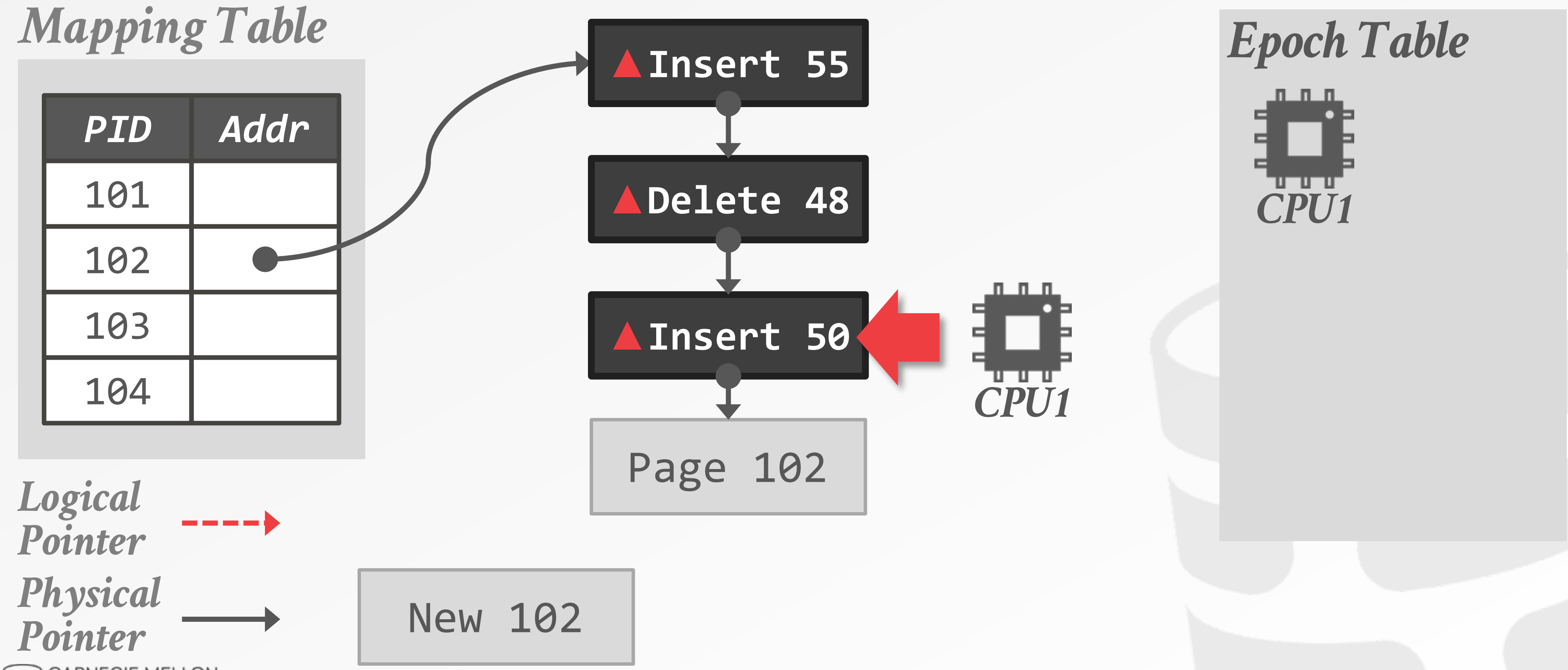
Operations are tagged with an **epoch**

→ 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.

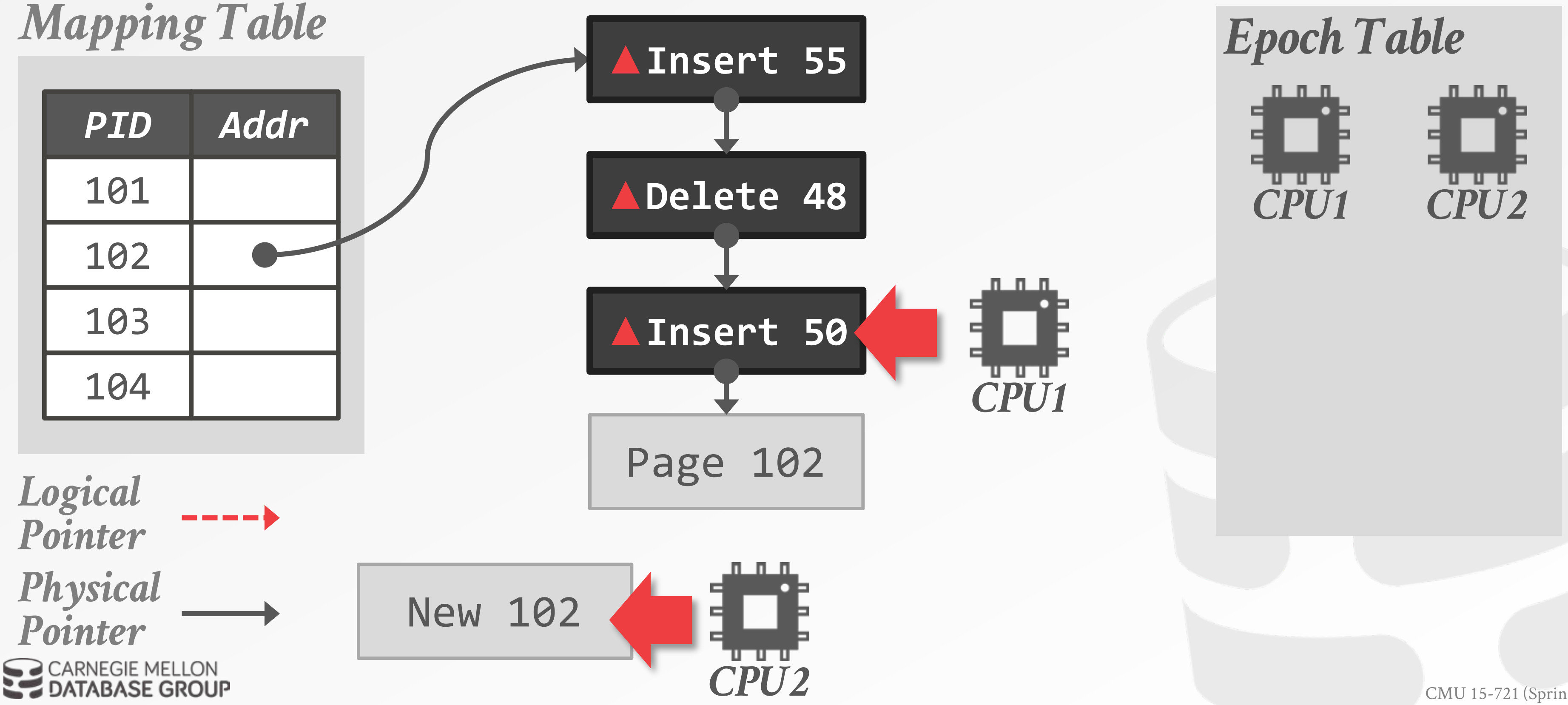
# BW-TREE: GARBAGE COLLECTION



*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

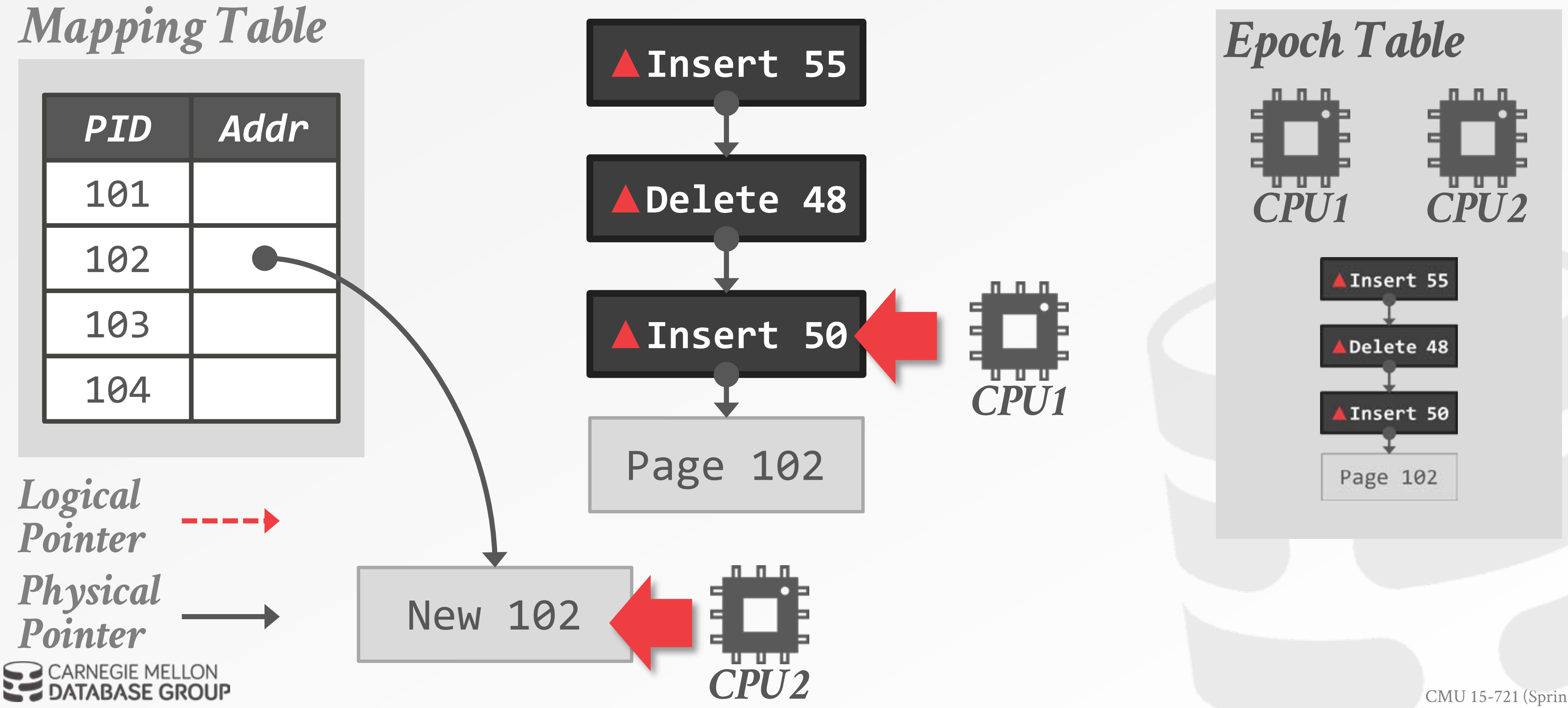
# BW-TREE: GARBAGE COLLECTION



Mapping Table

| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

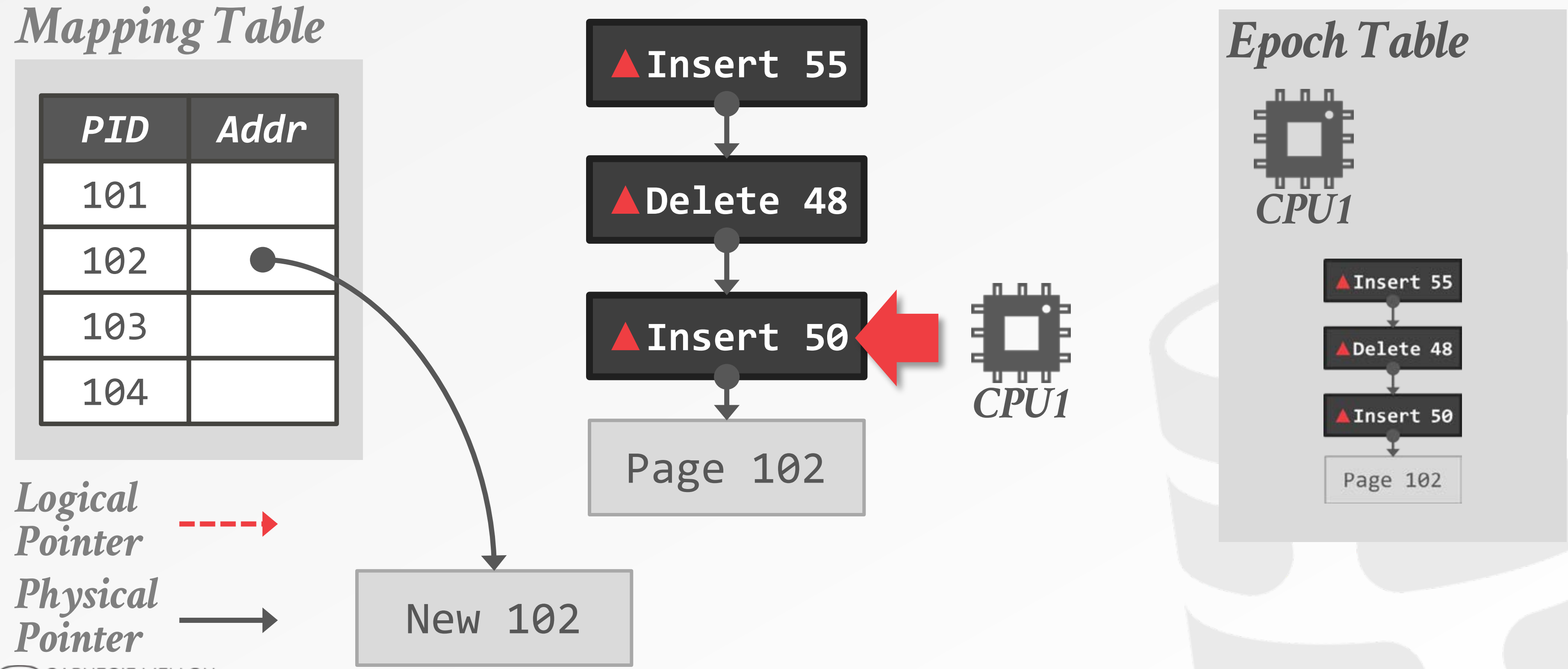
# BW-TREE: GARBAGE COLLECTION



*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

# BW-TREE: GARBAGE COLLECTION



*Mapping Table*

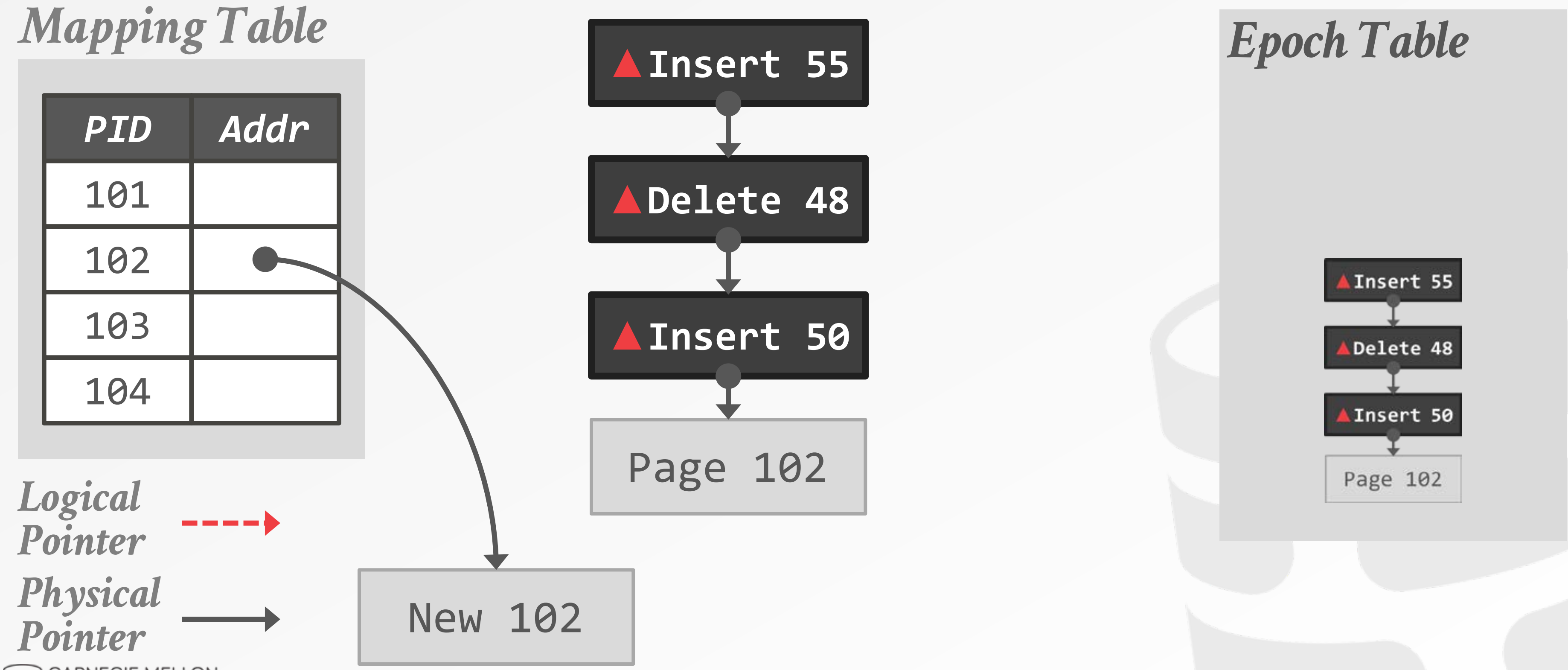
| *PID* | *Addr* |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

*Logical Pointer*

*Physical Pointer*

*Epoch Table*

# BW-TREE: GARBAGE COLLECTION



Mapping Table

| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

Logical Pointer

Physical Pointer

▲Insert 55

▲Delete 48

▲Insert 50

Page 102

New 102

Epoch Table

▲Insert 55

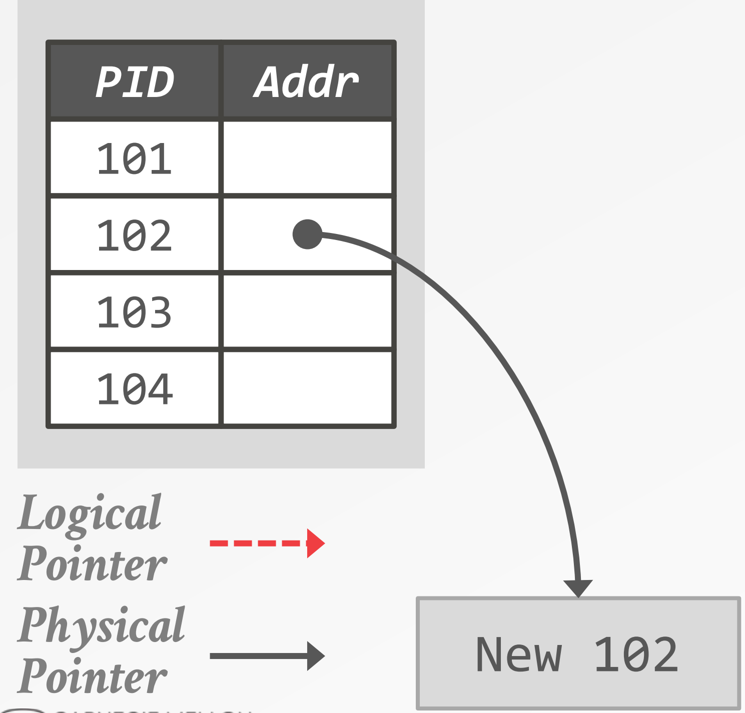
▲Delete 48

▲Insert 50

Page 102

# BW-TREE: GARBAGE COLLECTION

*Mapping Table*

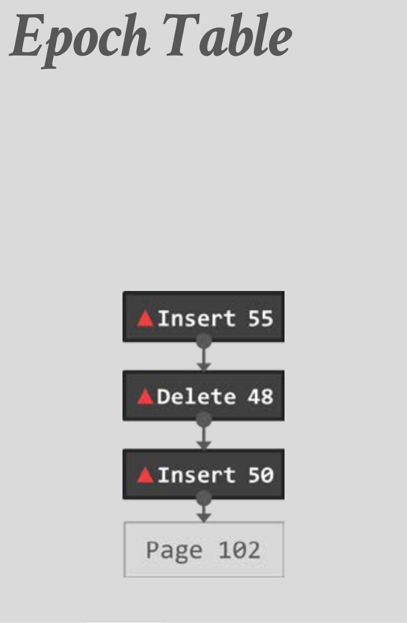


| PID | Addr |
|---|---|
| 101 | |
| 102 | ● |
| 103 | |
| 104 | |

*Logical Pointer*

*Physical Pointer*

*Epoch Table*



# BW-TREE: STRUCTURE MODIFICATIONS

## Split Delta Record

→ Mark that a subset of the base page's key range is now located at another page.

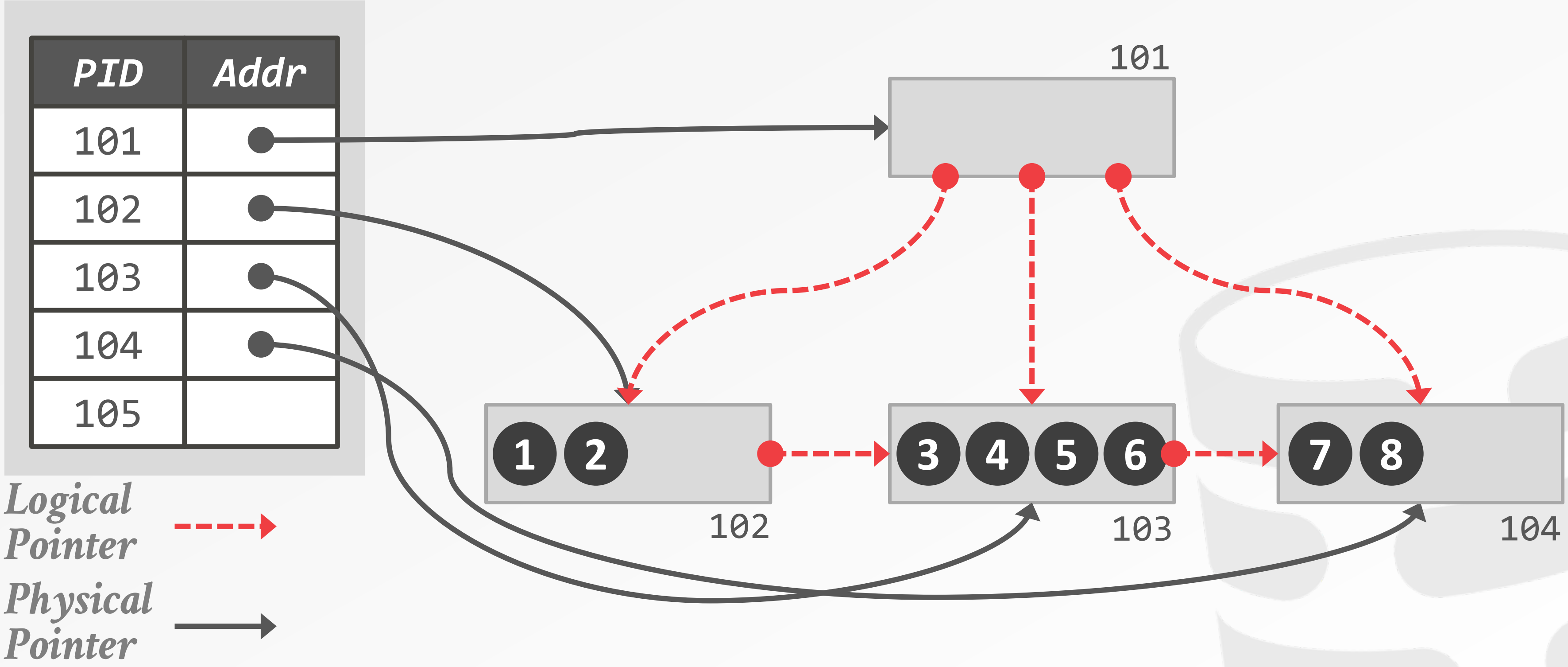
→ Use a logical pointer to the new page.

## Separator Delta Record

→ Provide a shortcut in the modified page's parent on what ranges to find the new page.

# BW-TREE: STRUCTURE MODIFICATIONS

*Mapping Table*

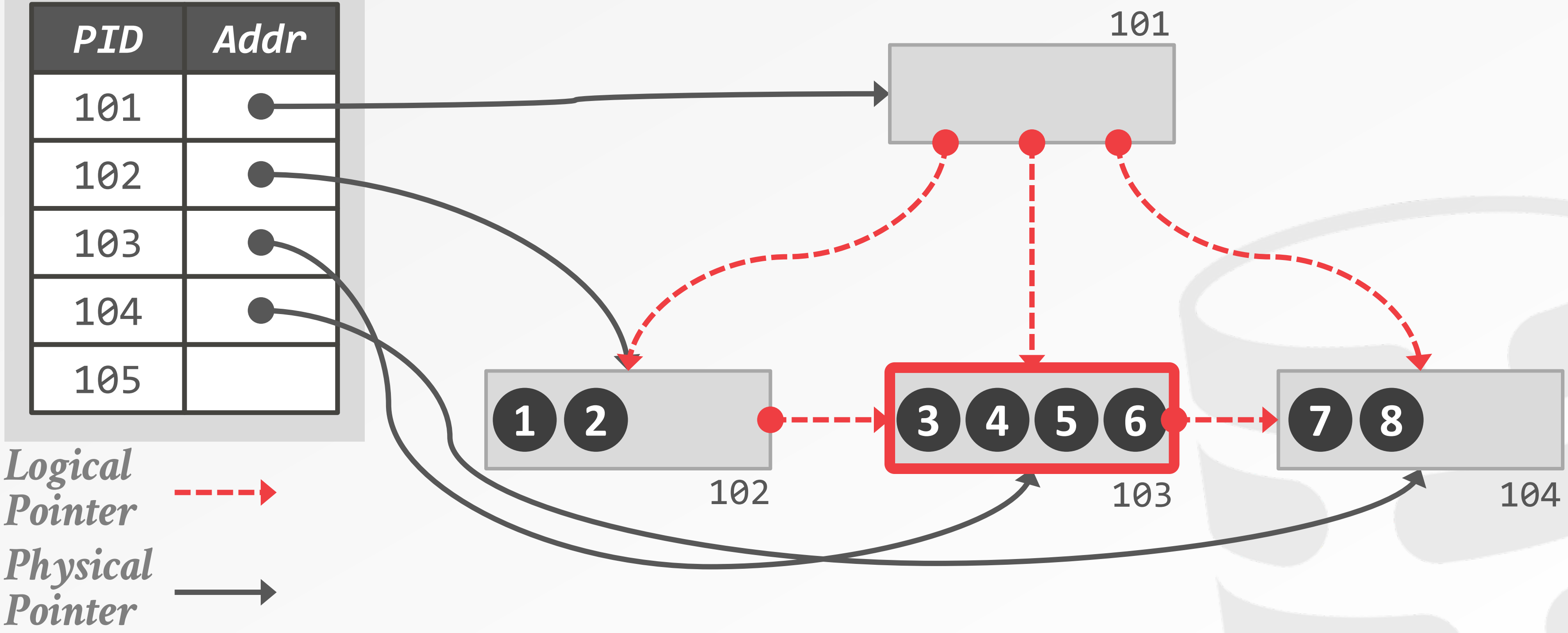


| PID | Addr |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | |

# BW-TREE: STRUCTURE MODIFICATIONS

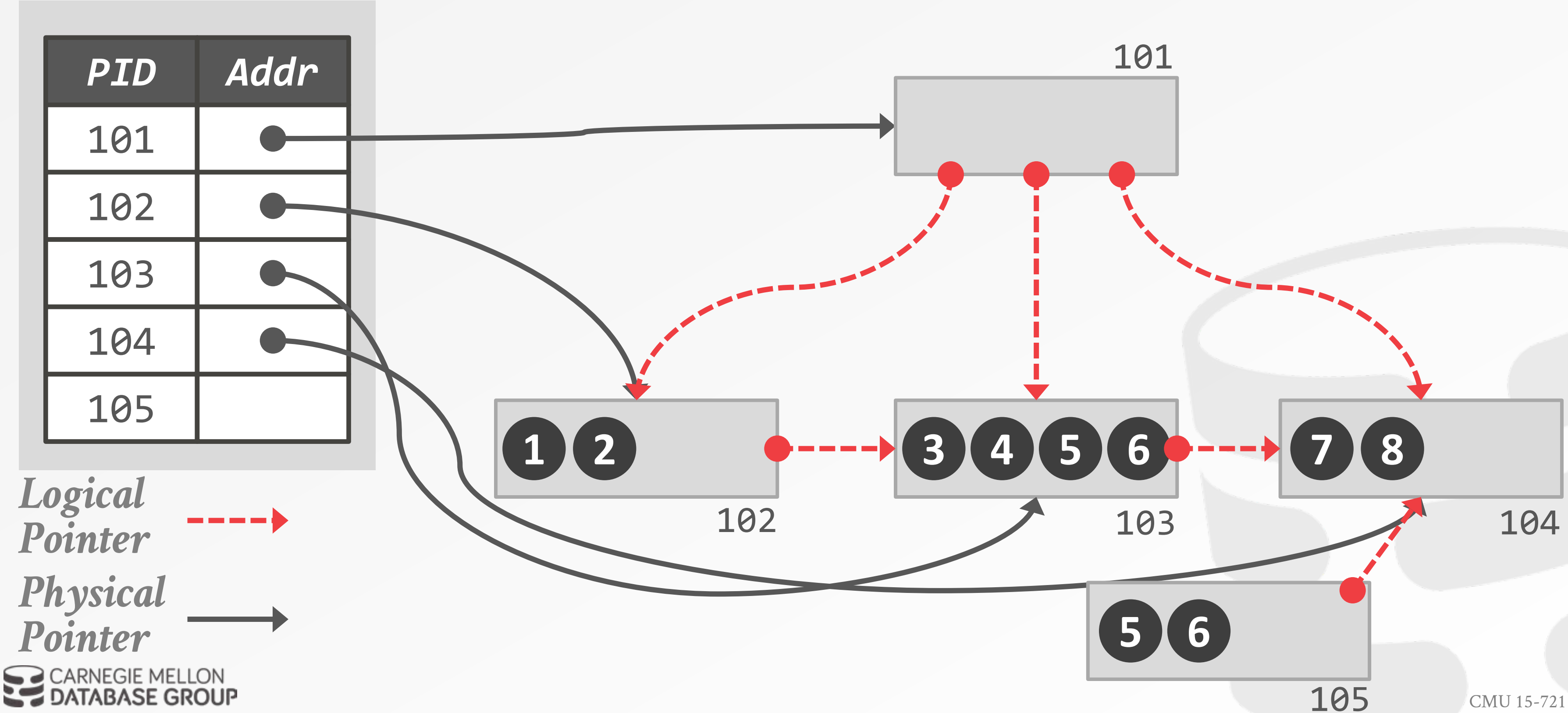
*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | |



# BW-TREE: STRUCTURE MODIFICATIONS

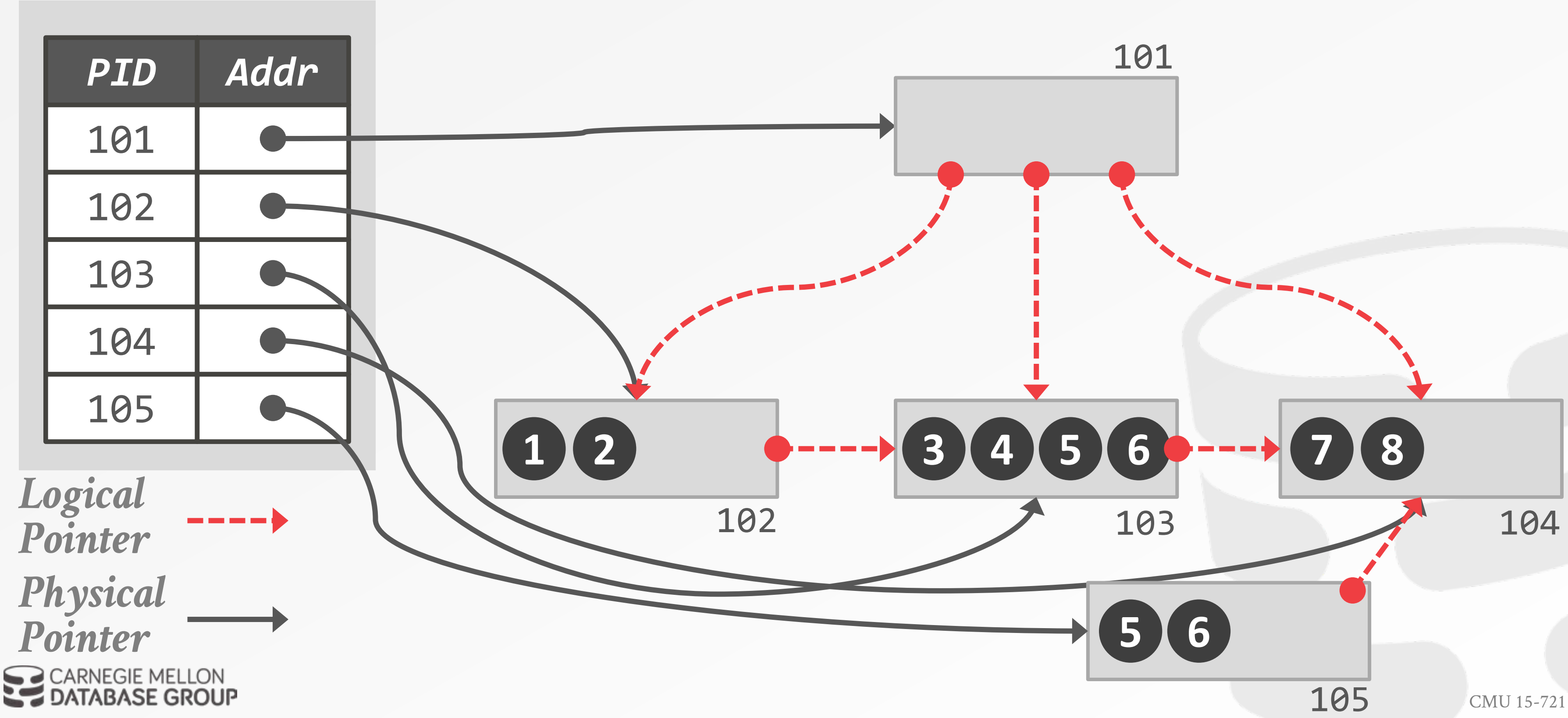
*Mapping Table*



| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | |

# BW-TREE: STRUCTURE MODIFICATIONS

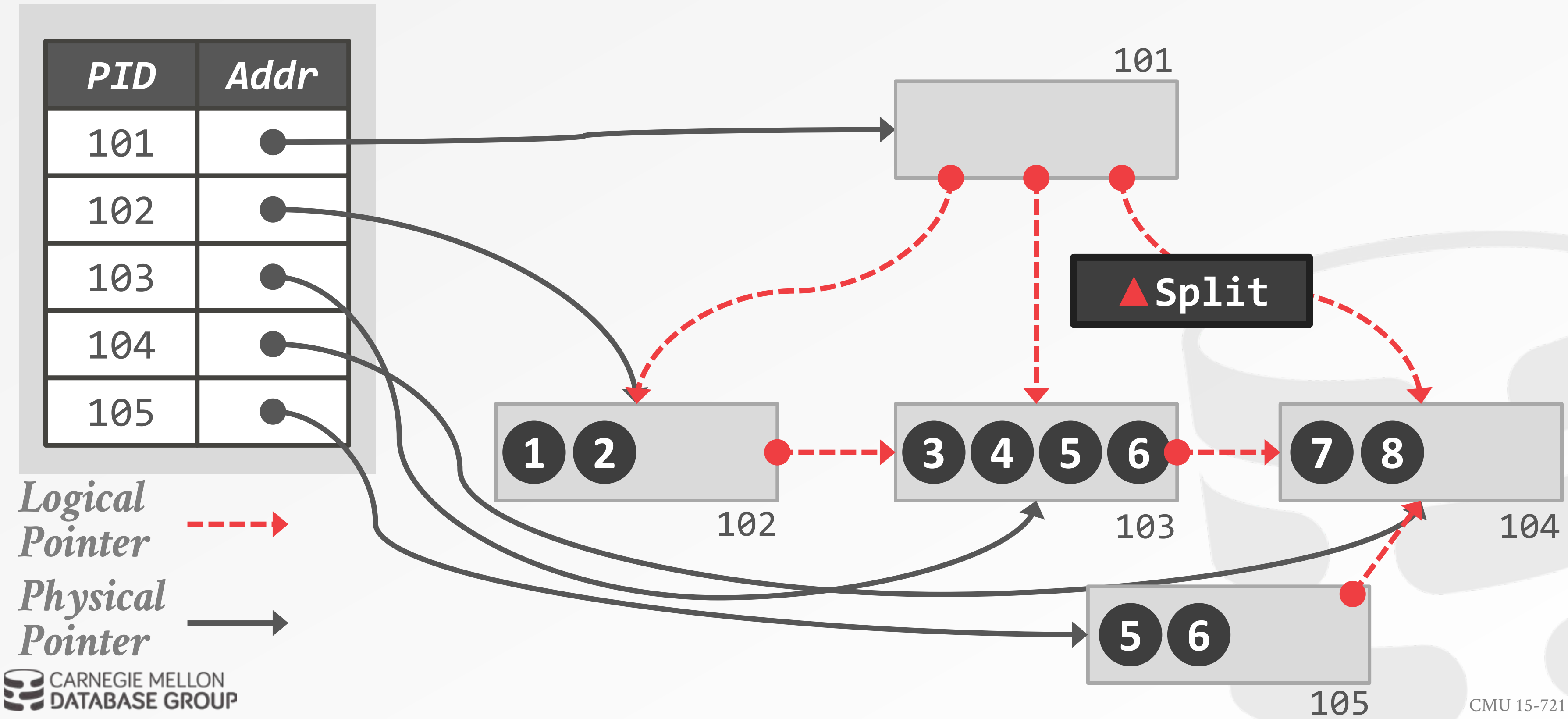
*Mapping Table*



| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

# BW-TREE: STRUCTURE MODIFICATIONS

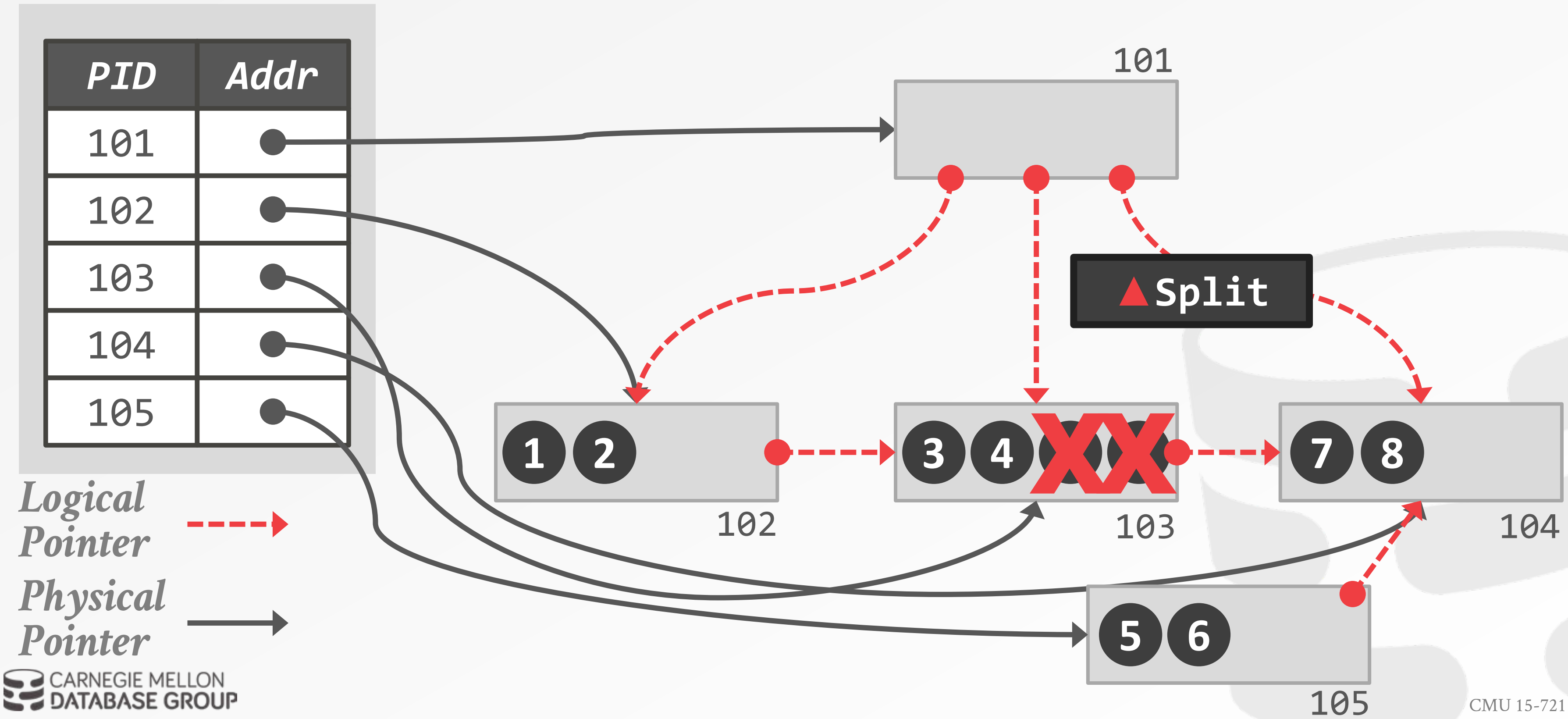
*Mapping Table*



| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

# BW-TREE: STRUCTURE MODIFICATIONS

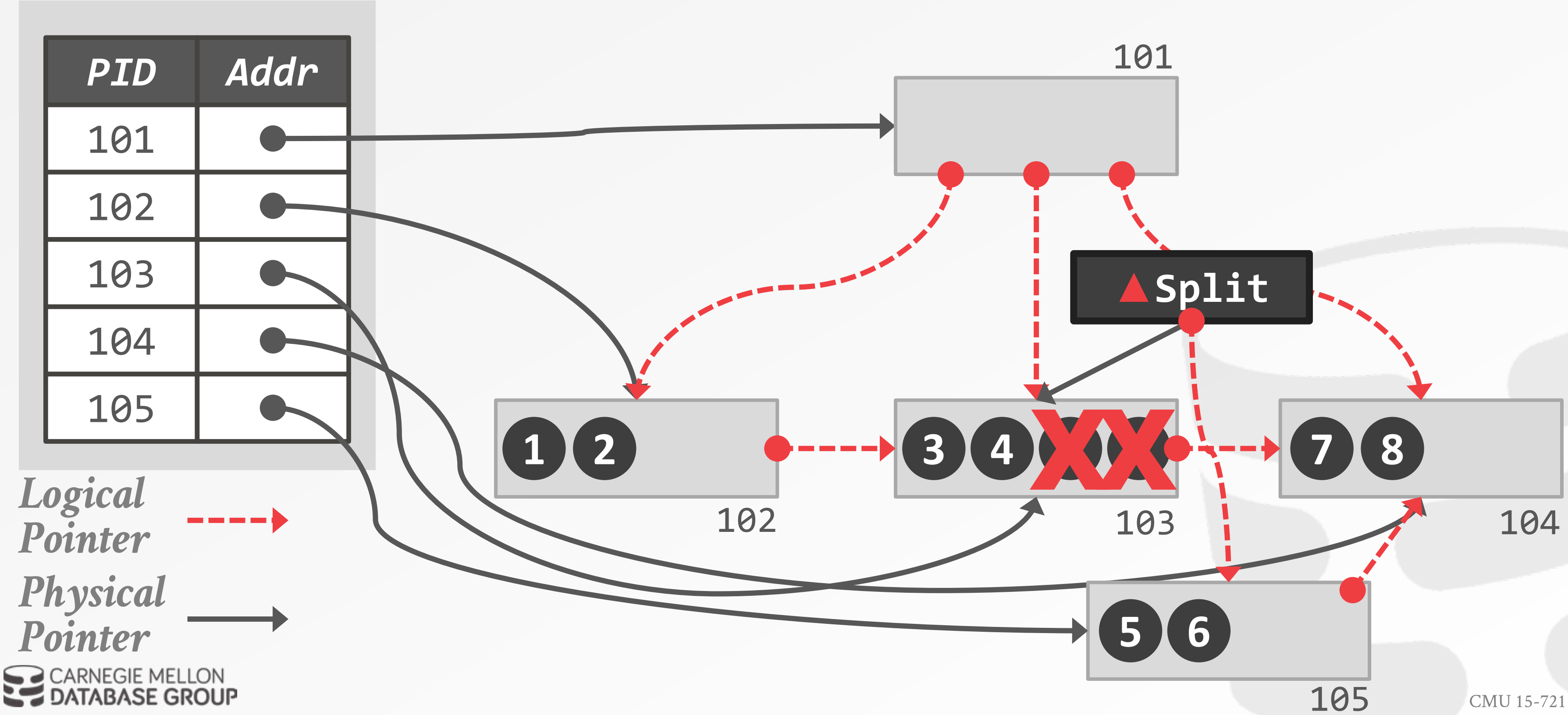
*Mapping Table*



| PID | Addr |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

# BW-TREE: STRUCTURE MODIFICATIONS

*Mapping Table*



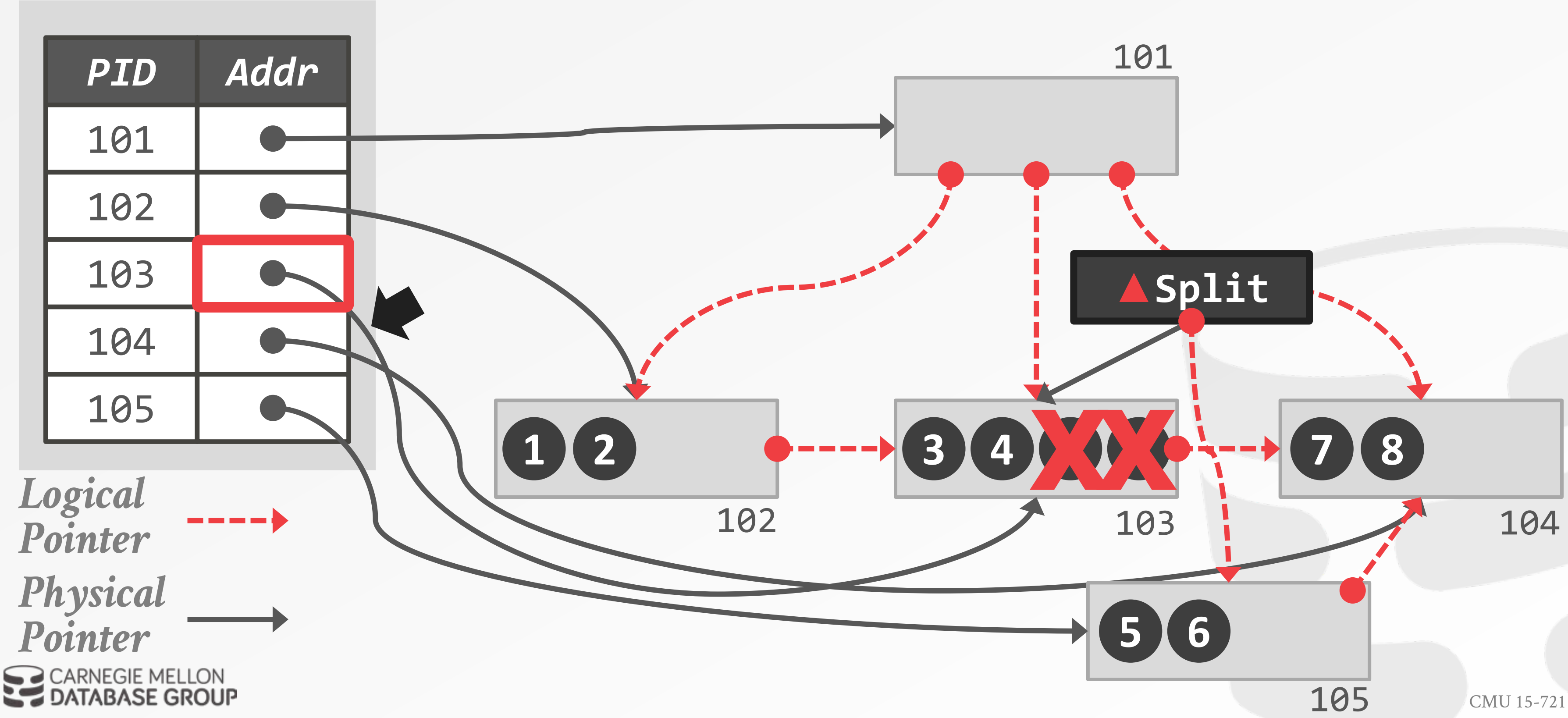
| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

*Logical Pointer*

*Physical Pointer*

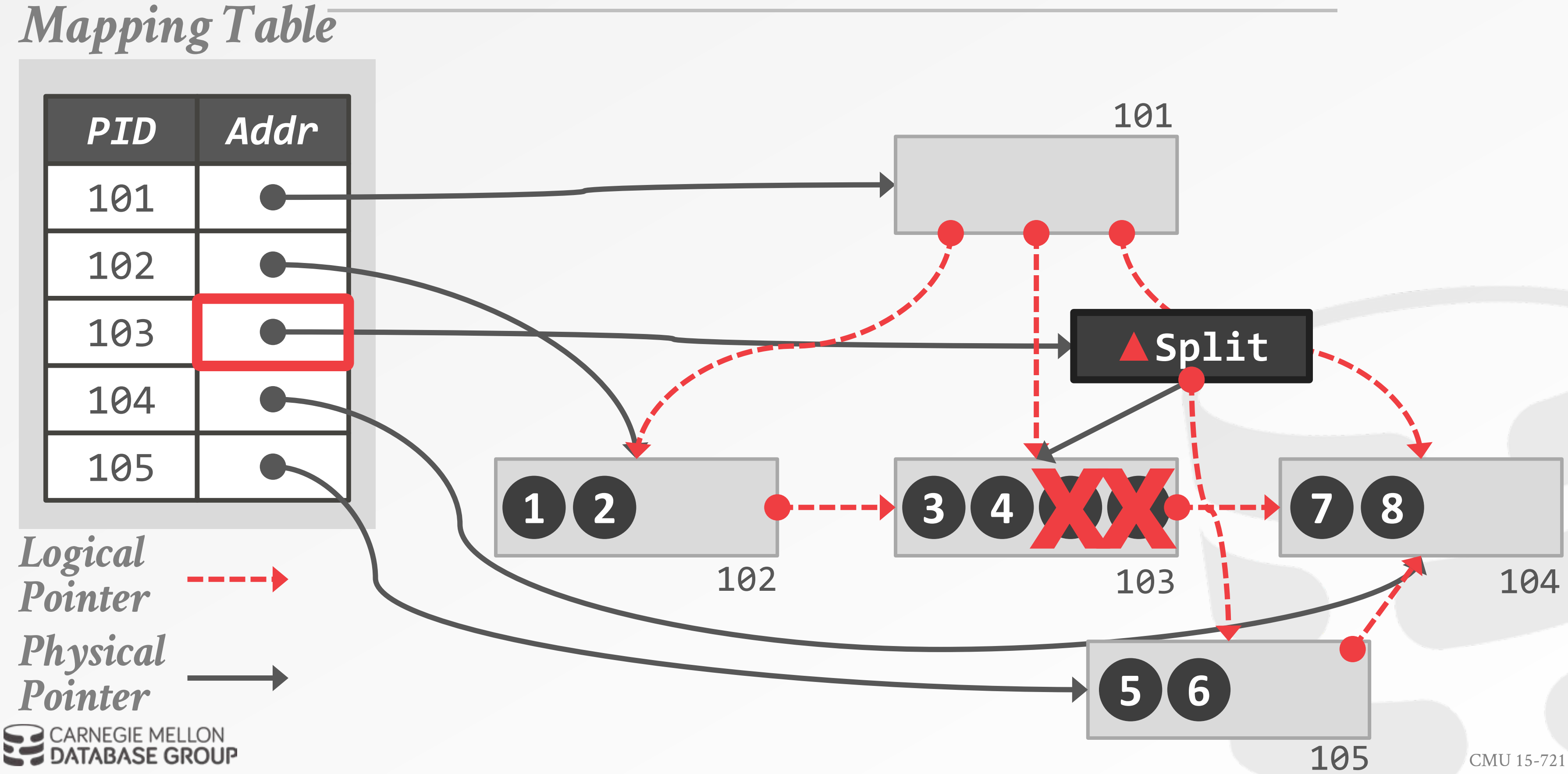
# BW-TREE: STRUCTURE MODIFICATIONS

*Mapping Table*



| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

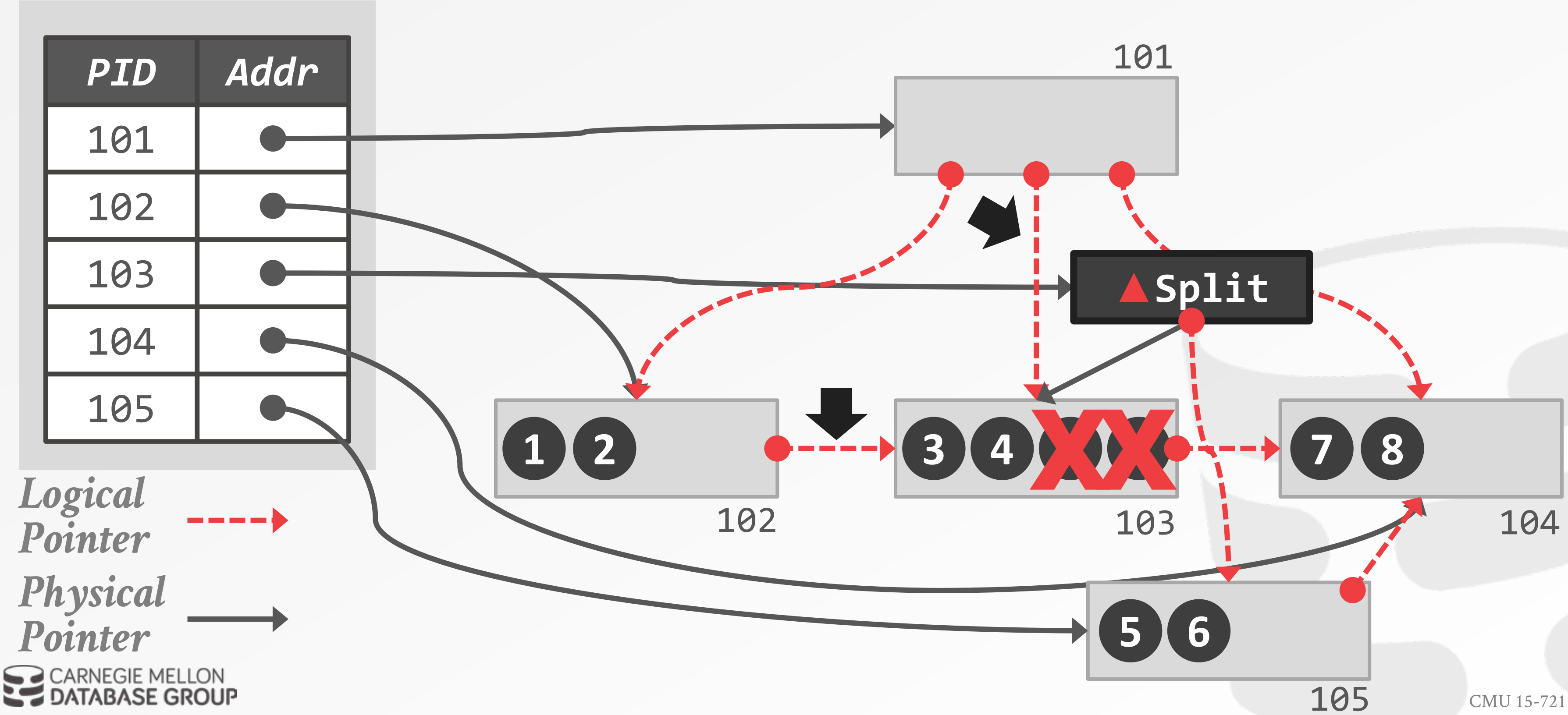
# BW-TREE: STRUCTURE MODIFICATIONS



# BW-TREE: STRUCTURE MODIFICATIONS

*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

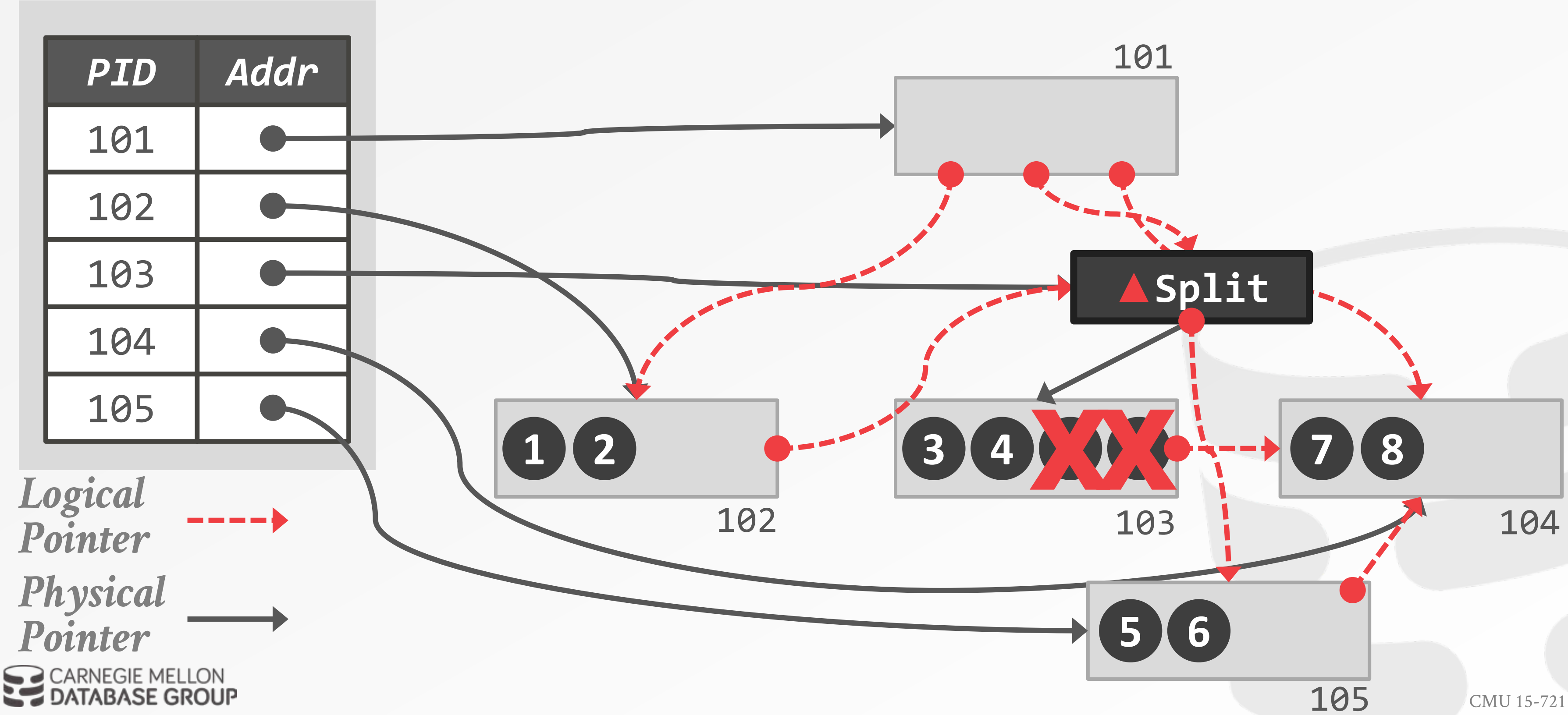


*Logical Pointer*

*Physical Pointer*

# BW-TREE: STRUCTURE MODIFICATIONS

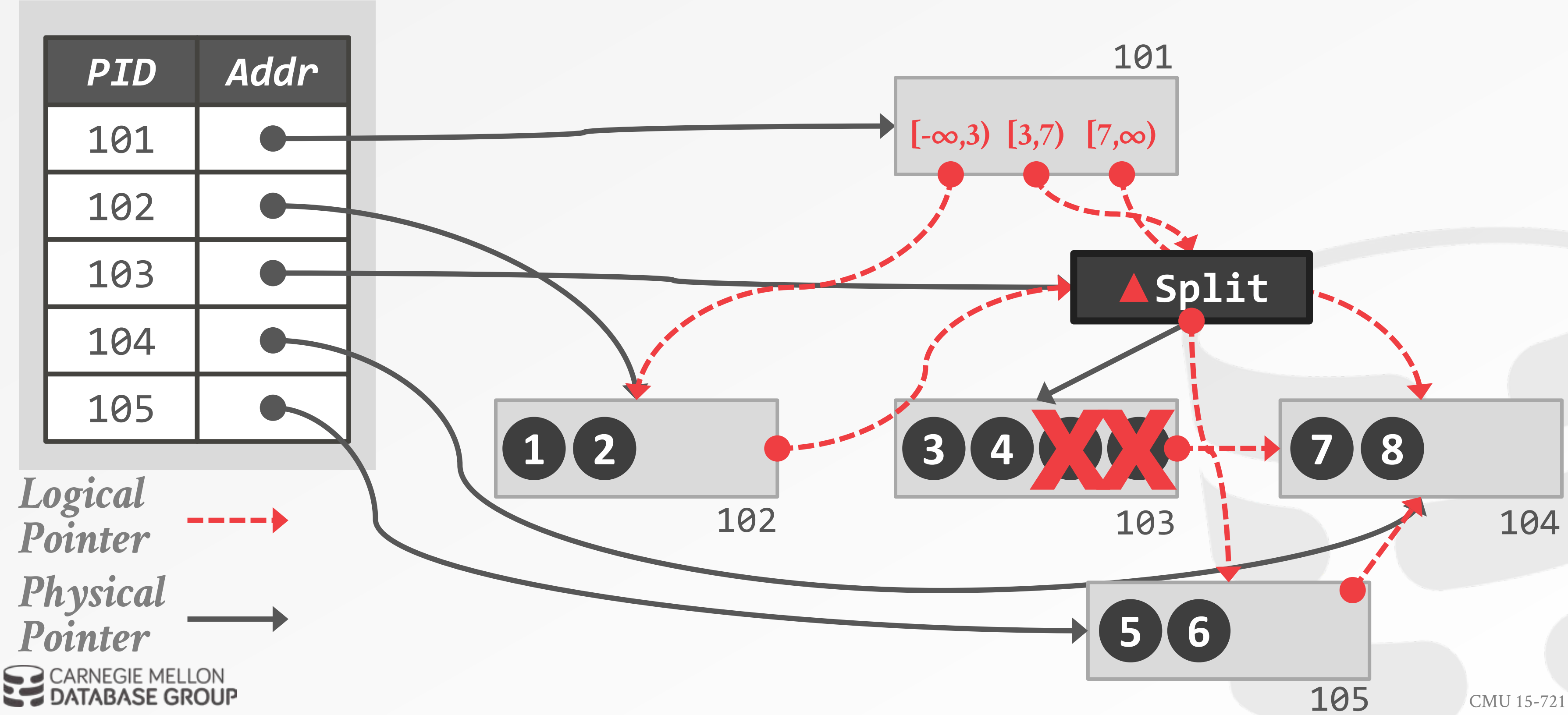
*Mapping Table*



| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

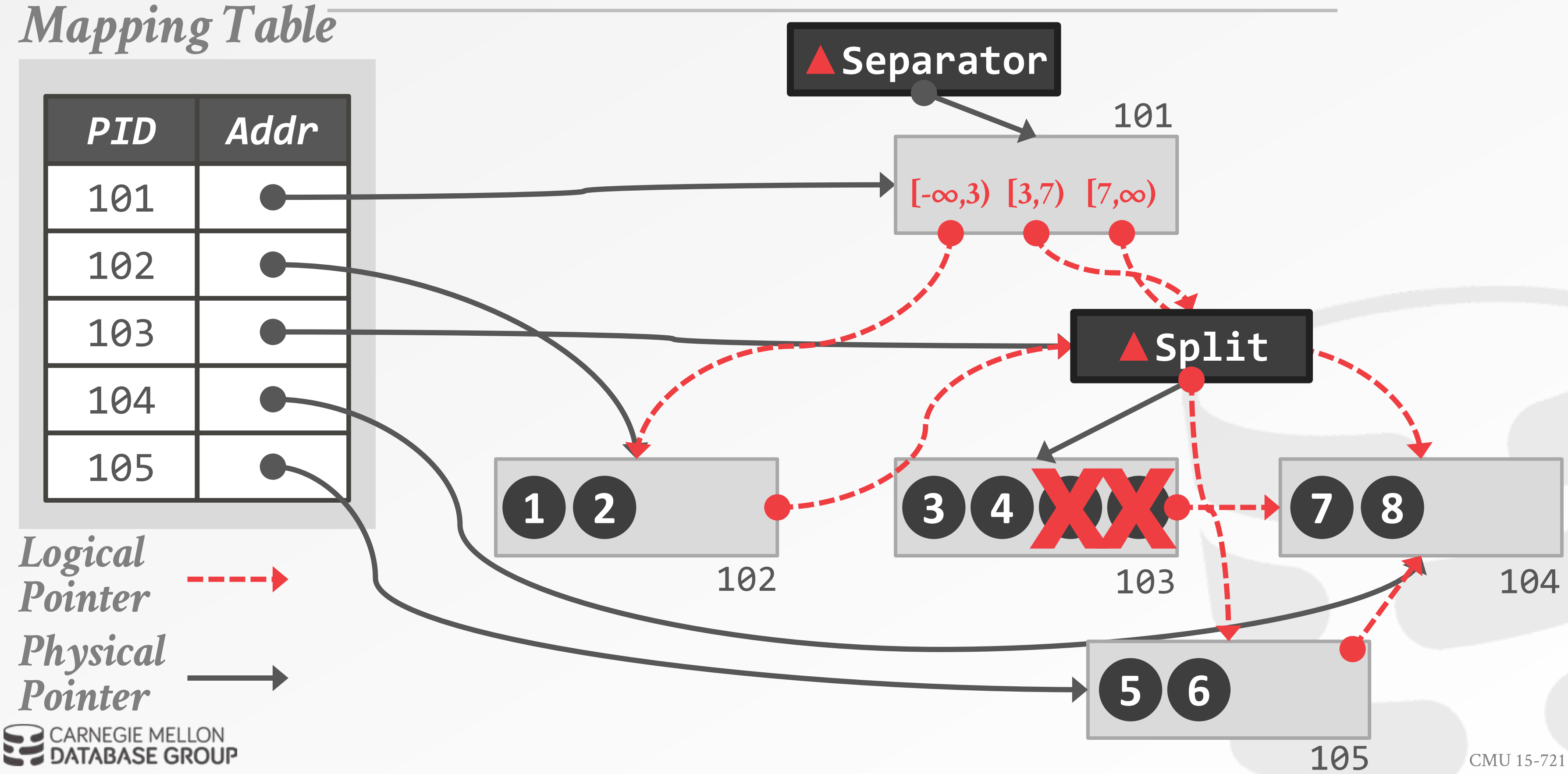
# BW-TREE: STRUCTURE MODIFICATIONS

*Mapping Table*



| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

# BW-TREE: STRUCTURE MODIFICATIONS



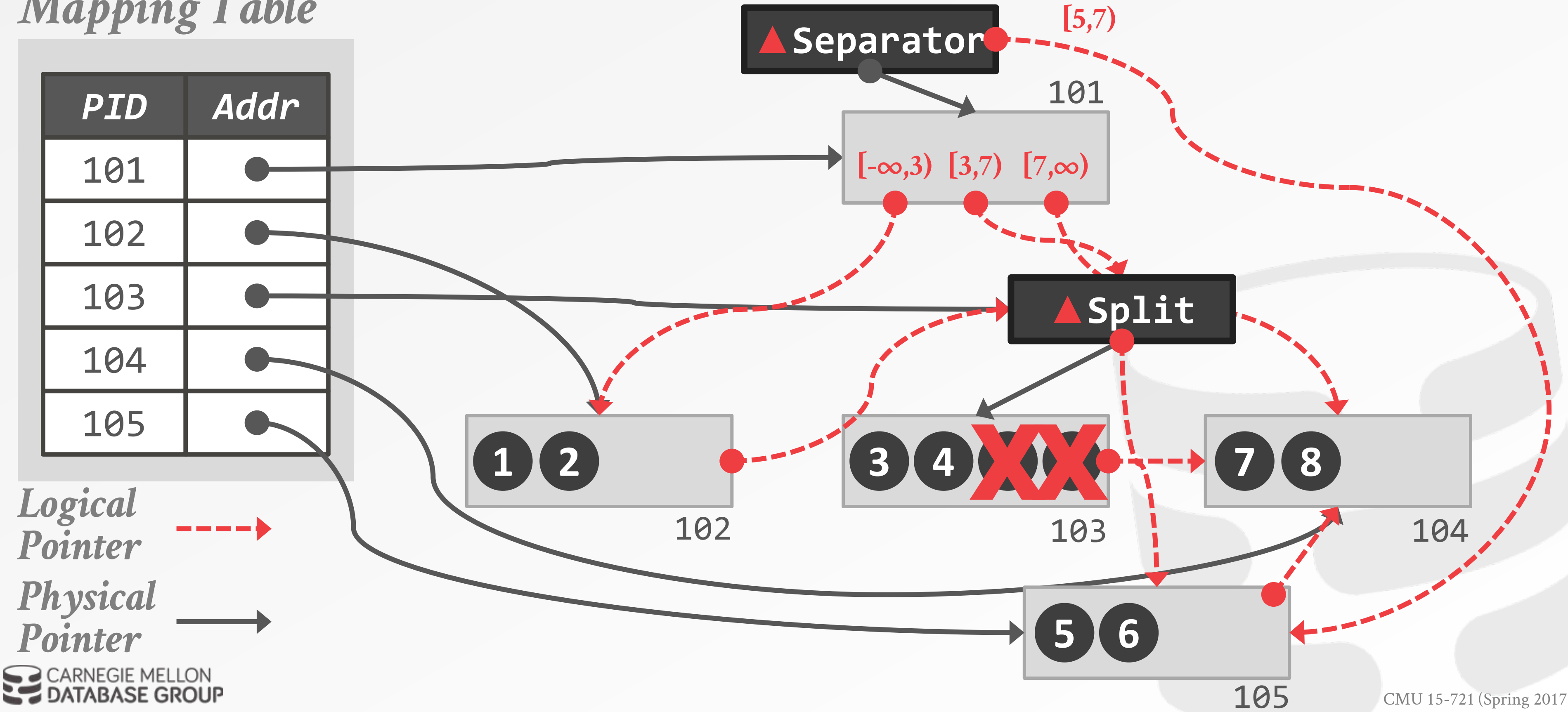
*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

# BW-TREE: STRUCTURE MODIFICATIONS

*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |

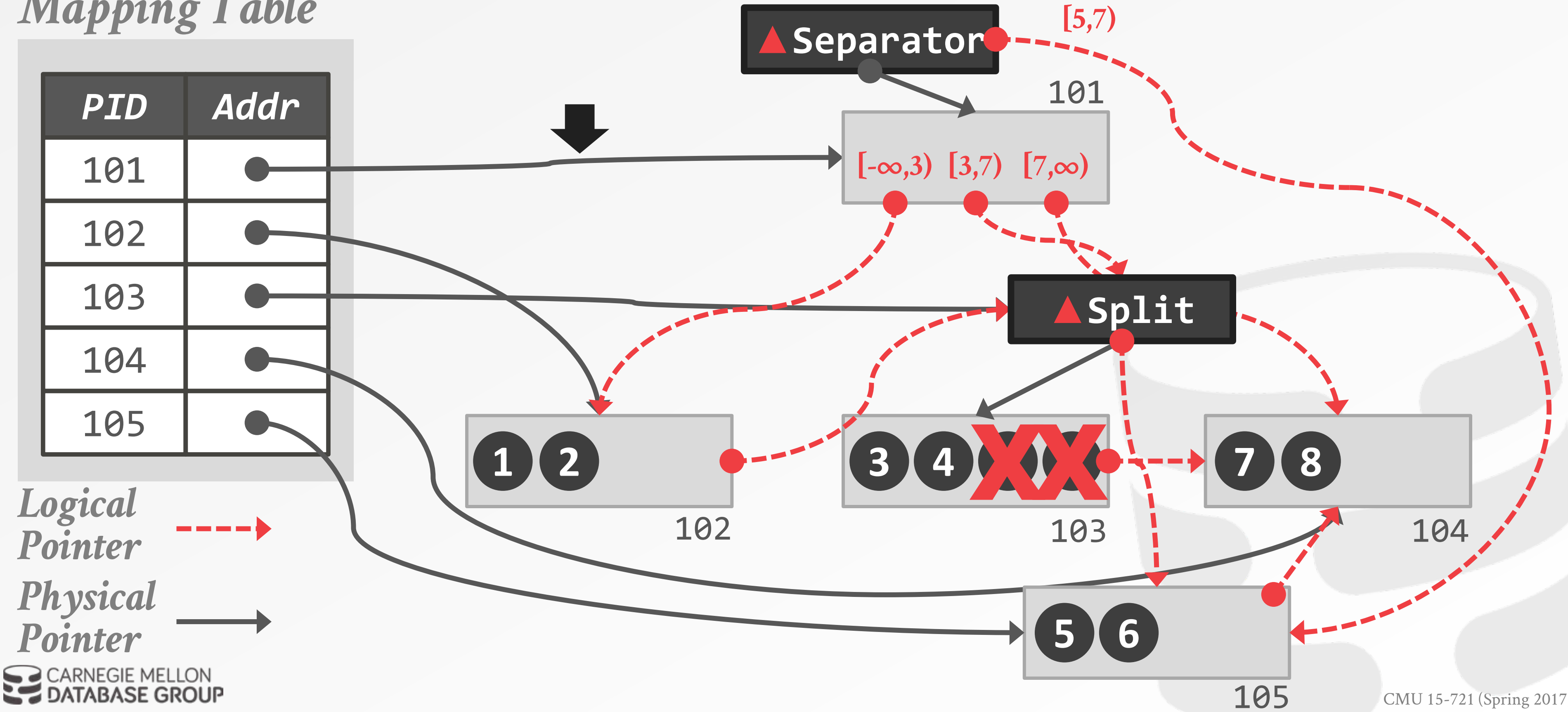


CARNEGIE MELLON DATABASE GROUP

# BW-TREE: STRUCTURE MODIFICATIONS

*Mapping Table*

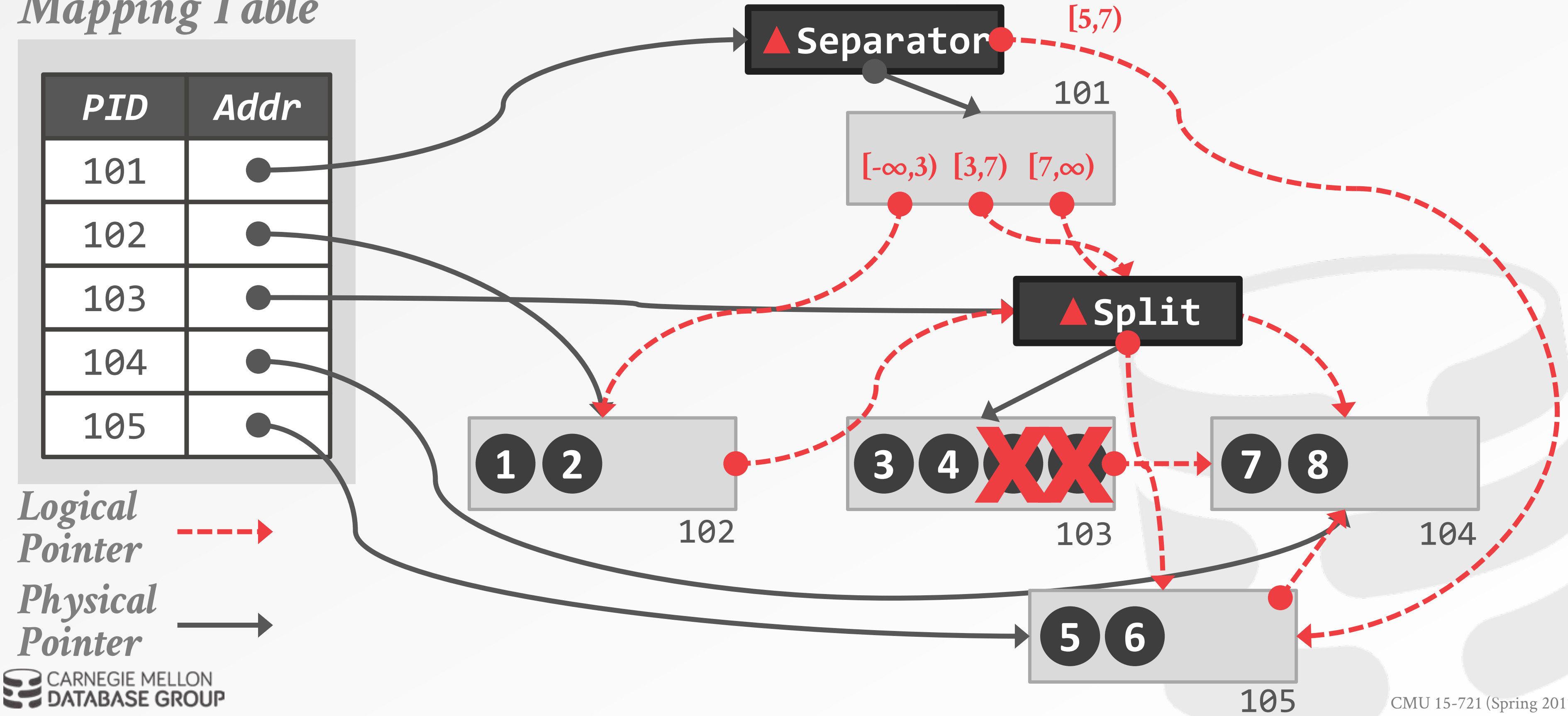
| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |



# BW-TREE: STRUCTURE MODIFICATIONS

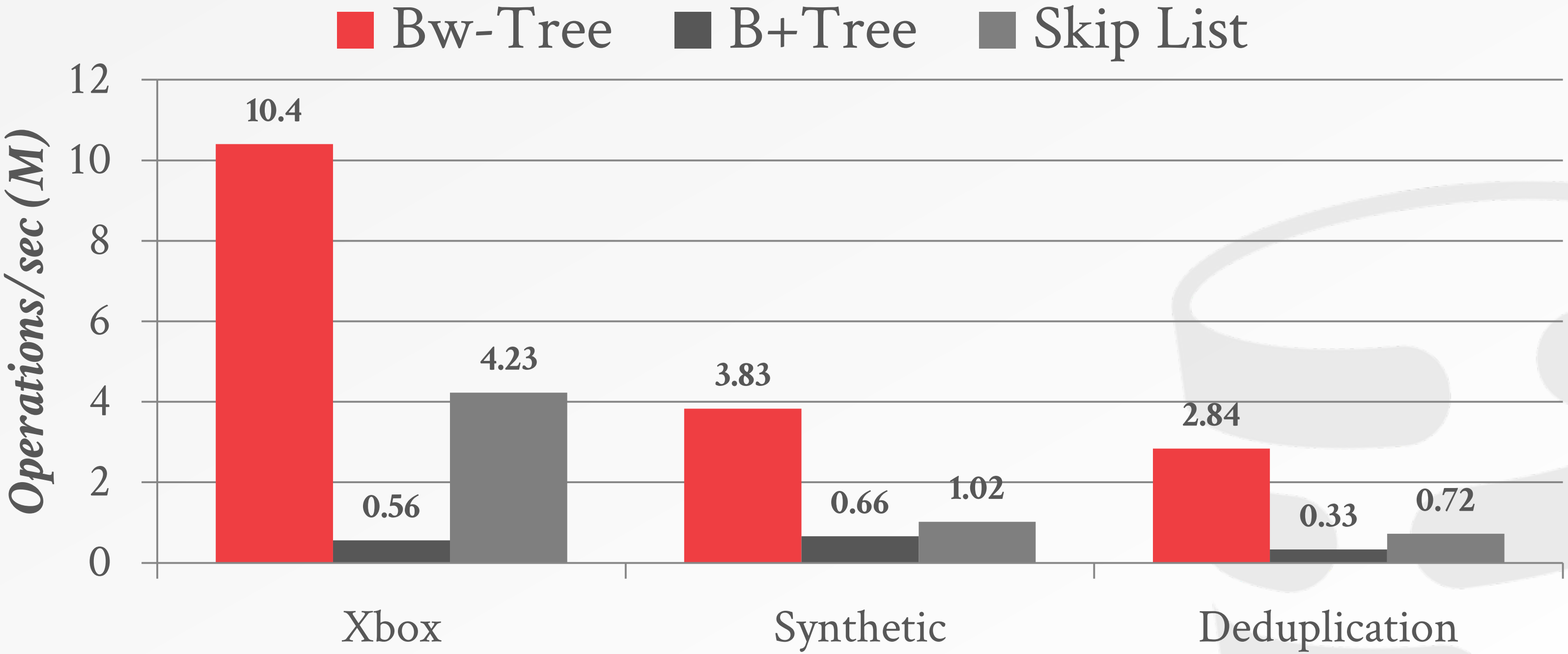
*Mapping Table*

| *PID* | *Addr* |
|---|---|
| 101 | ● |
| 102 | ● |
| 103 | ● |
| 104 | ● |
| 105 | ● |



# BW-TREE: PERFORMANCE

Processor: 1 socket, 4 cores w/ 2×HT



Source: Justin Levandoski

# ADAPATIVE RADIX TREE (ART)

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

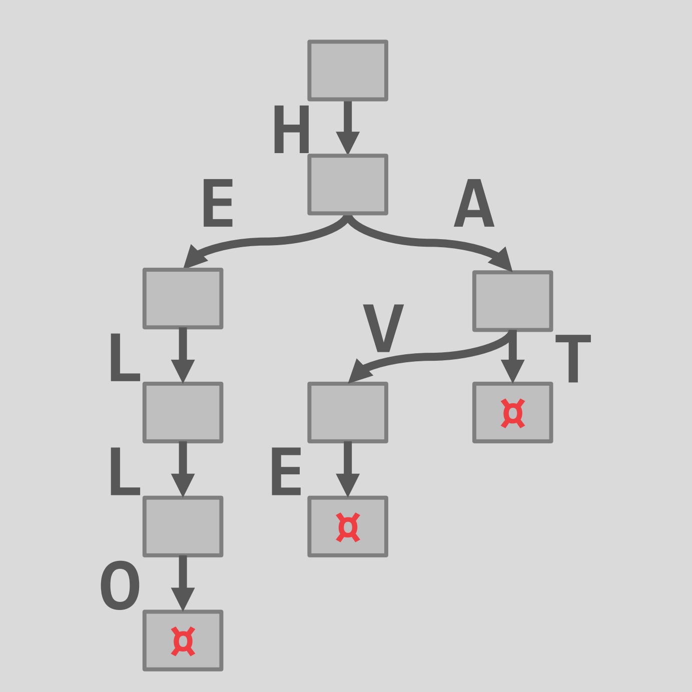
Radix trees properties:

→ 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.

THE ADAPTIVE RADIX TREE: ARTFUL INDEXING FOR MAIN-MEMORY DATABASES
*ICDE 2013*

# TRIE VS. RADIX TREE

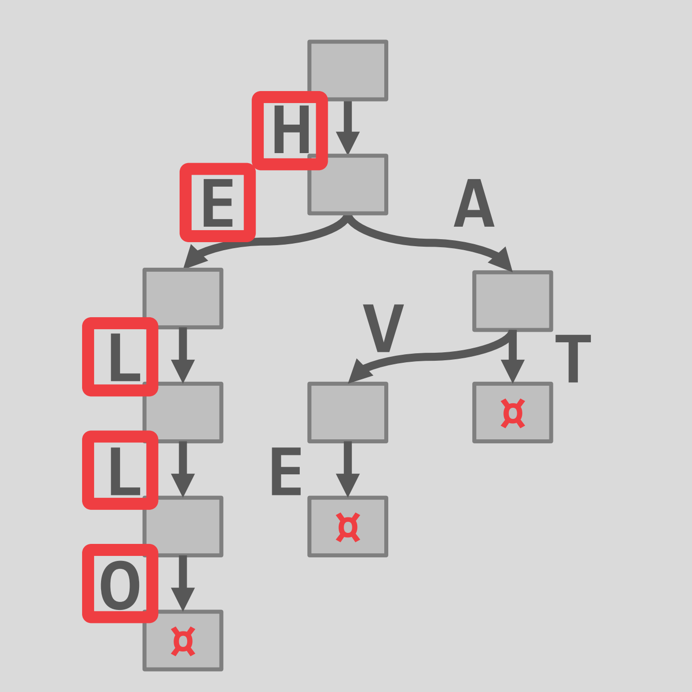
*Trie*



Keys: HELLO, HAT, HAVE

# TRIE VS. RADIX TREE

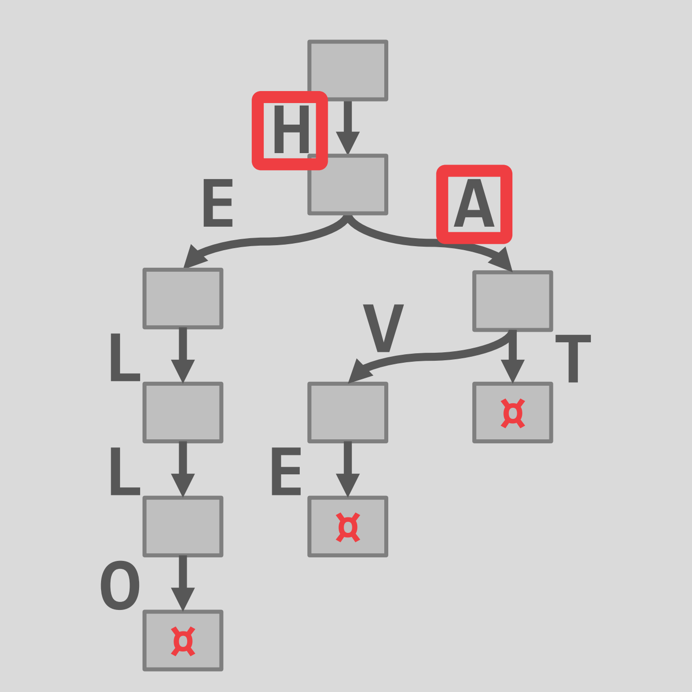
*Trie*



Keys: HELLO, HAT, HAVE

# TRIE VS. RADIX TREE

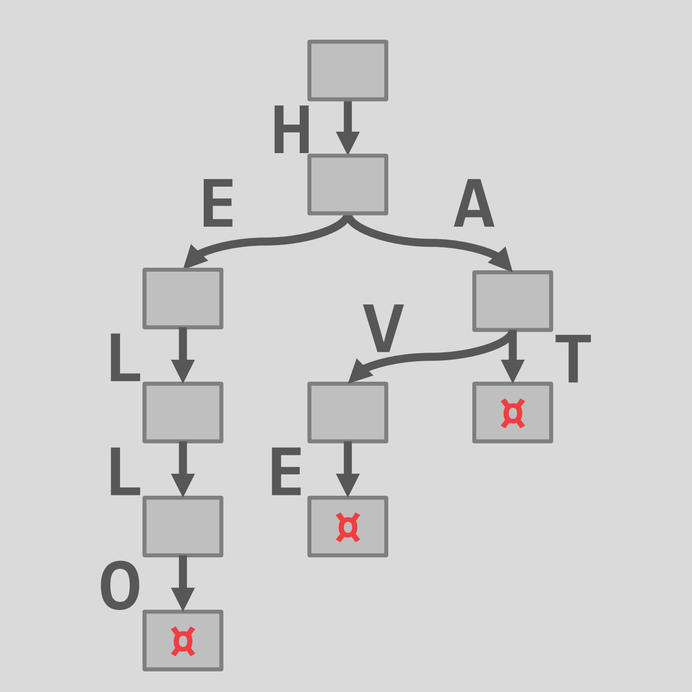
*Trie*



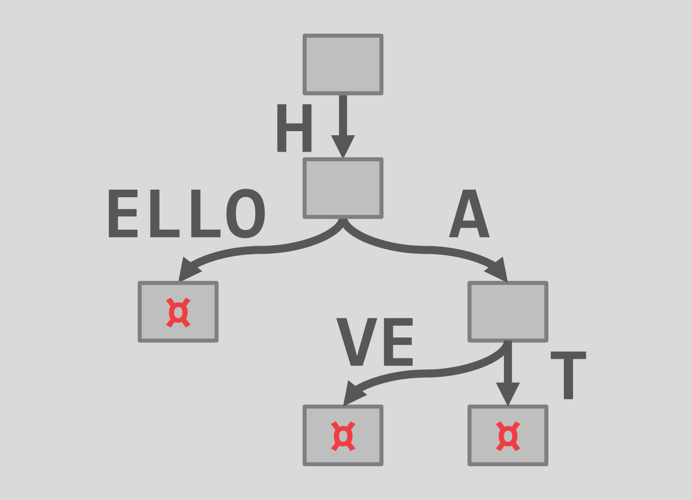
Keys: HELLO, HAT, HAVE

# TRIE VS. RADIX TREE

*Trie*



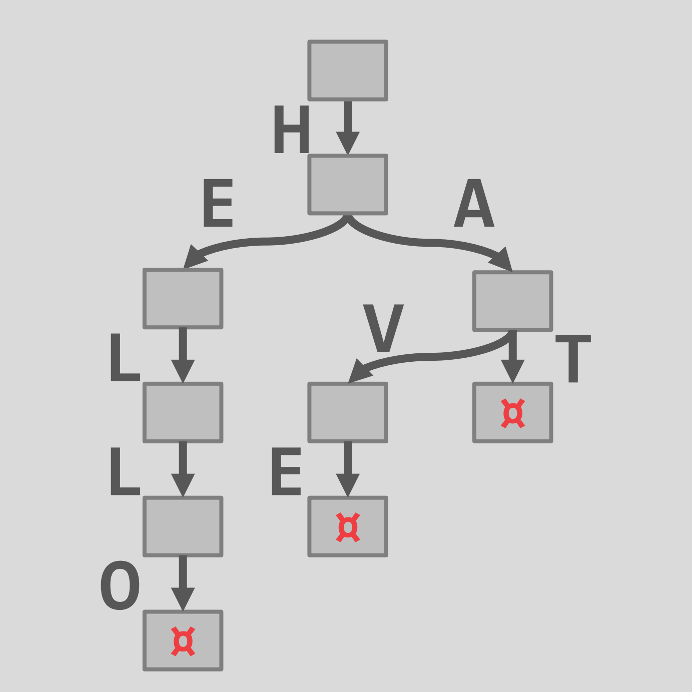
*Radix Tree*



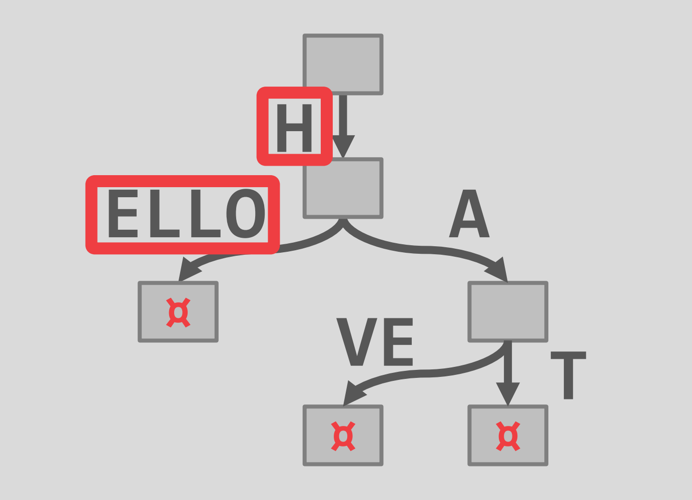
Keys: **HELLO, HAT, HAVE**

# TRIE VS. RADIX TREE

*Trie*

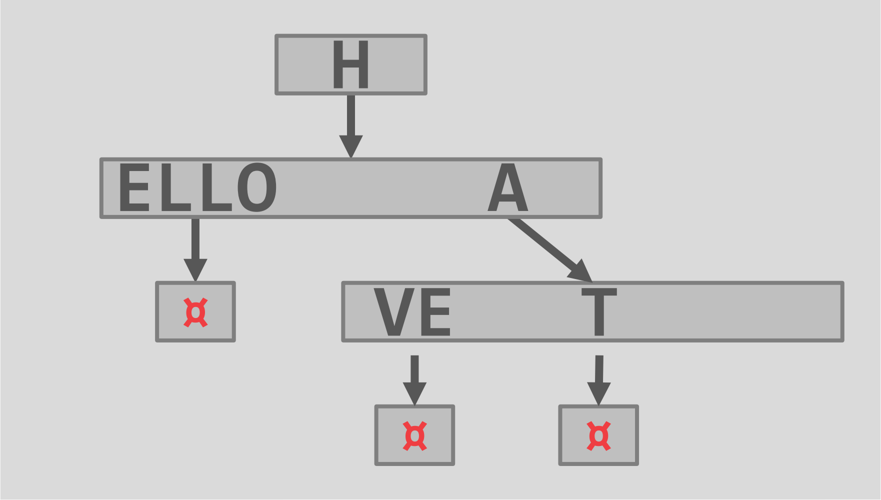


*Radix Tree*

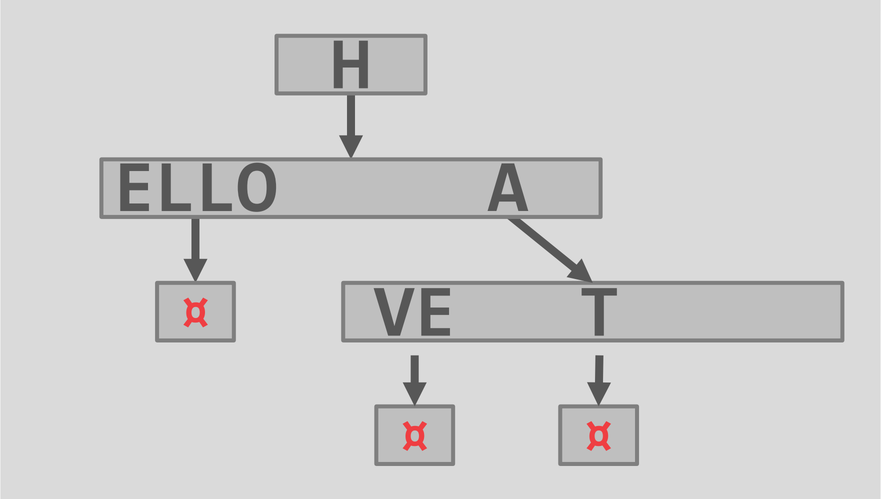


Keys: HELLO, HAT, HAVE

# ART INDEX: MODIFICATIONS

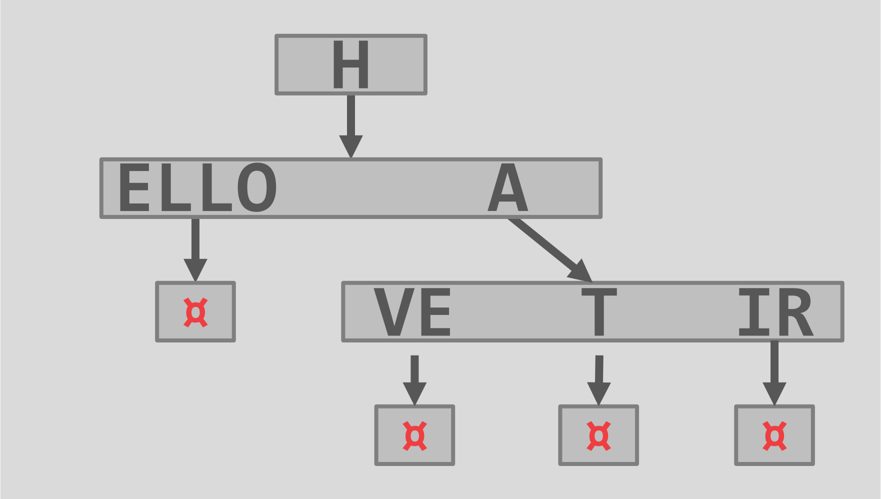


# ART INDEX: MODIFICATIONS



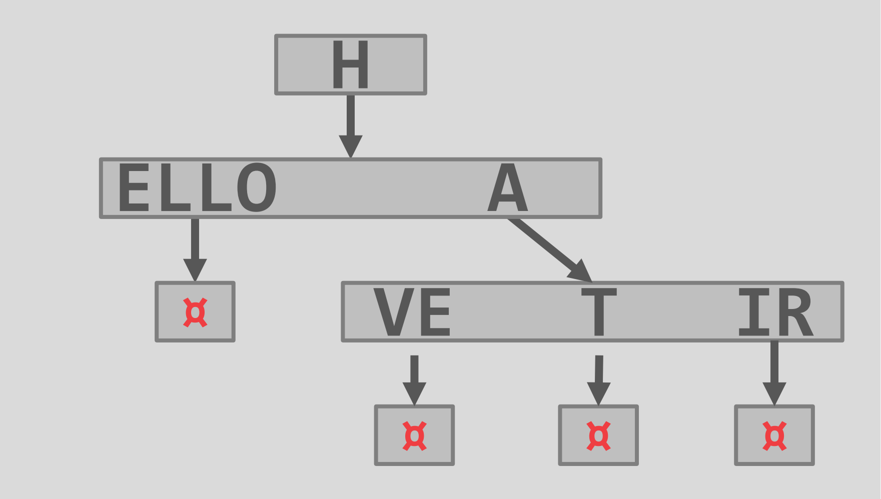
**Operation:** Insert **HAIR**

# ART INDEX: MODIFICATIONS



**Operation:** Insert **HAIR**

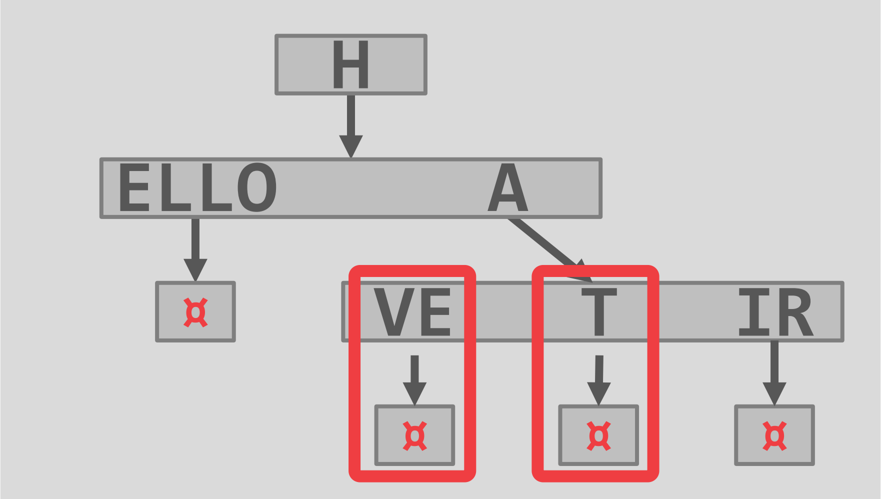
# ART INDEX: MODIFICATIONS



**Operation:** Insert **HAIR**

**Operation:** Delete **HAT**, **HAVE**

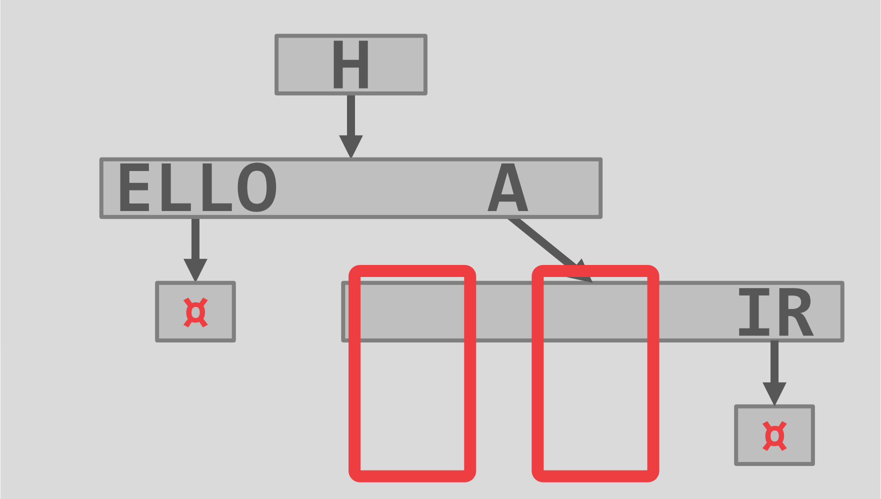
# ART INDEX: MODIFICATIONS



**Operation:** Insert **HAIR**

**Operation:** Delete **HAT**, **HAVE**

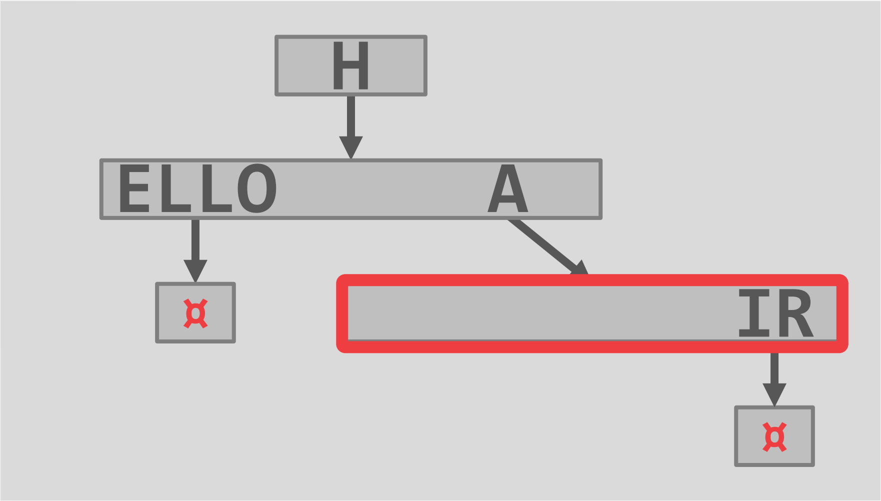
# ART INDEX: MODIFICATIONS



**Operation:** Insert **HAIR**

**Operation:** Delete **HAT**, **HAVE**

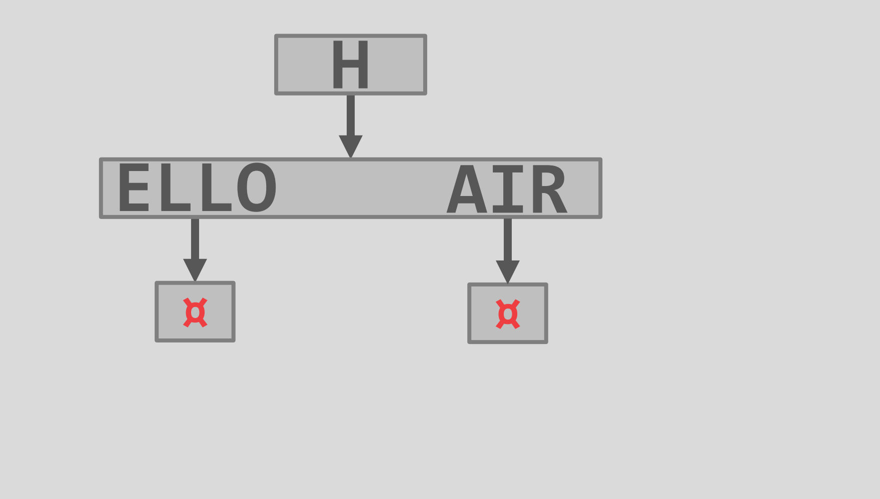
# ART INDEX: MODIFICATIONS



**Operation:** Insert **HAIR**

**Operation:** Delete **HAT**, **HAVE**

# ART INDEX: MODIFICATIONS



**Operation:** Insert **HAIR**

**Operation:** Delete **HAT**, **HAVE**

# ART INDEX: BINARY COMPARABLE KEYS

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.

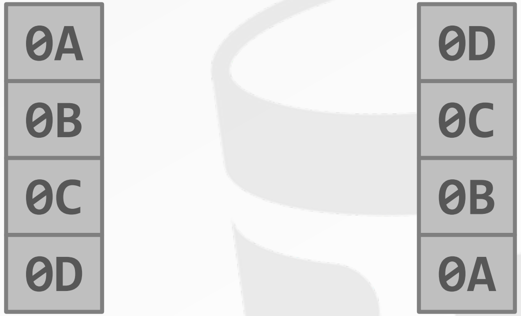
→ **Floats**: Classify into group (neg vs. pos, normalized vs. denormalized), then store as unsigned integer.

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

# ART INDEX: BINARY COMPARABLE KEYS

**Int Key: 168496141**

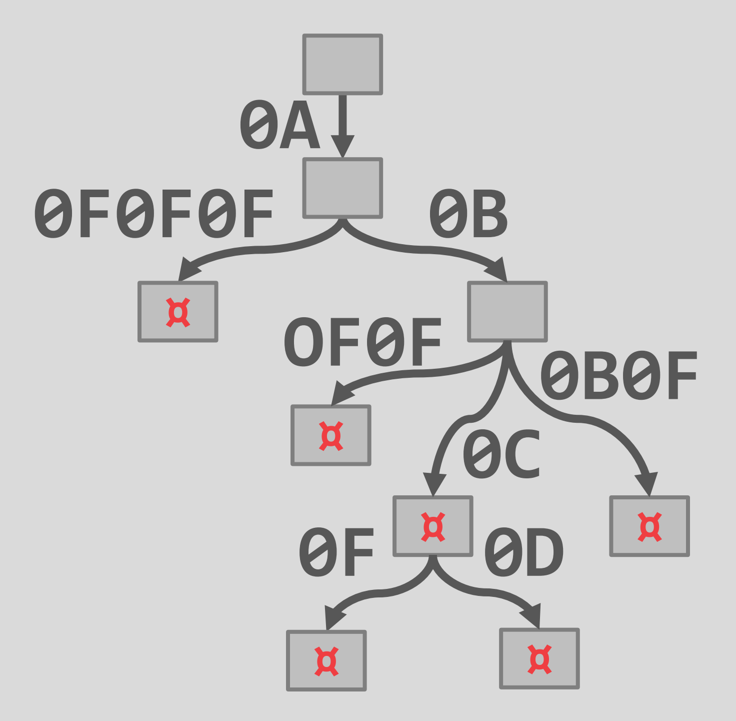
**Hex Key: 0A 0B 0C 0D**



| Big Endian | Little Endian |
|---|---|
| 0A | 0D |
| 0B | 0C |
| 0C | 0B |
| 0D | 0A |

*Big Endian* *Little Endian*

# ART INDEX: BINARY COMPARABLE KEYS



**Int Key: 168496141**

**Hex Key: 0A 0B 0C 0D**

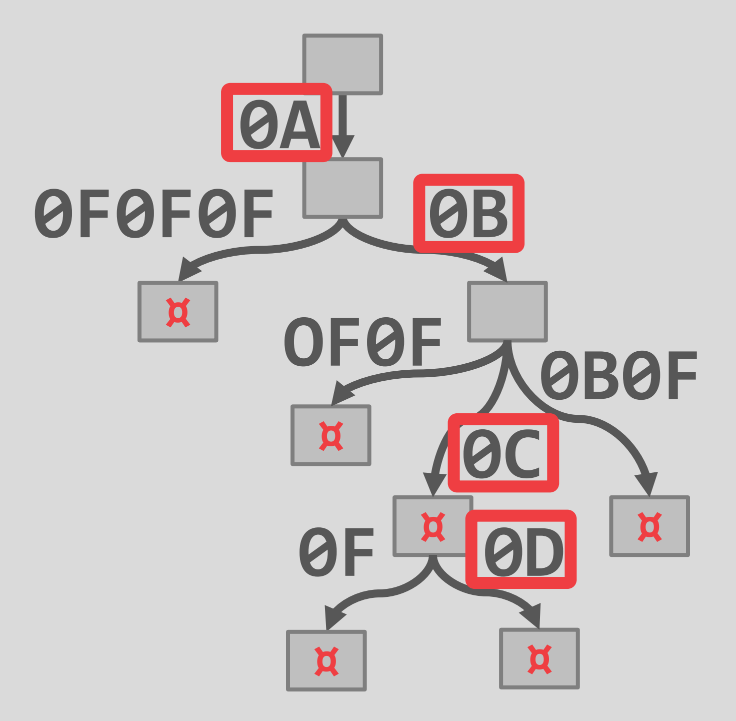
| 0A |
|---|
| 0B |
| 0C |
| 0D |

*Big Endian*

| 0D |
|---|
| 0C |
| 0B |
| 0A |

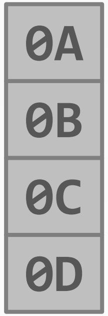
*Little Endian*

# ART INDEX: BINARY COMPARABLE KEYS

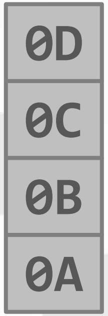


**Int Key: 168496141**

**Hex Key: 0A 0B 0C 0D**



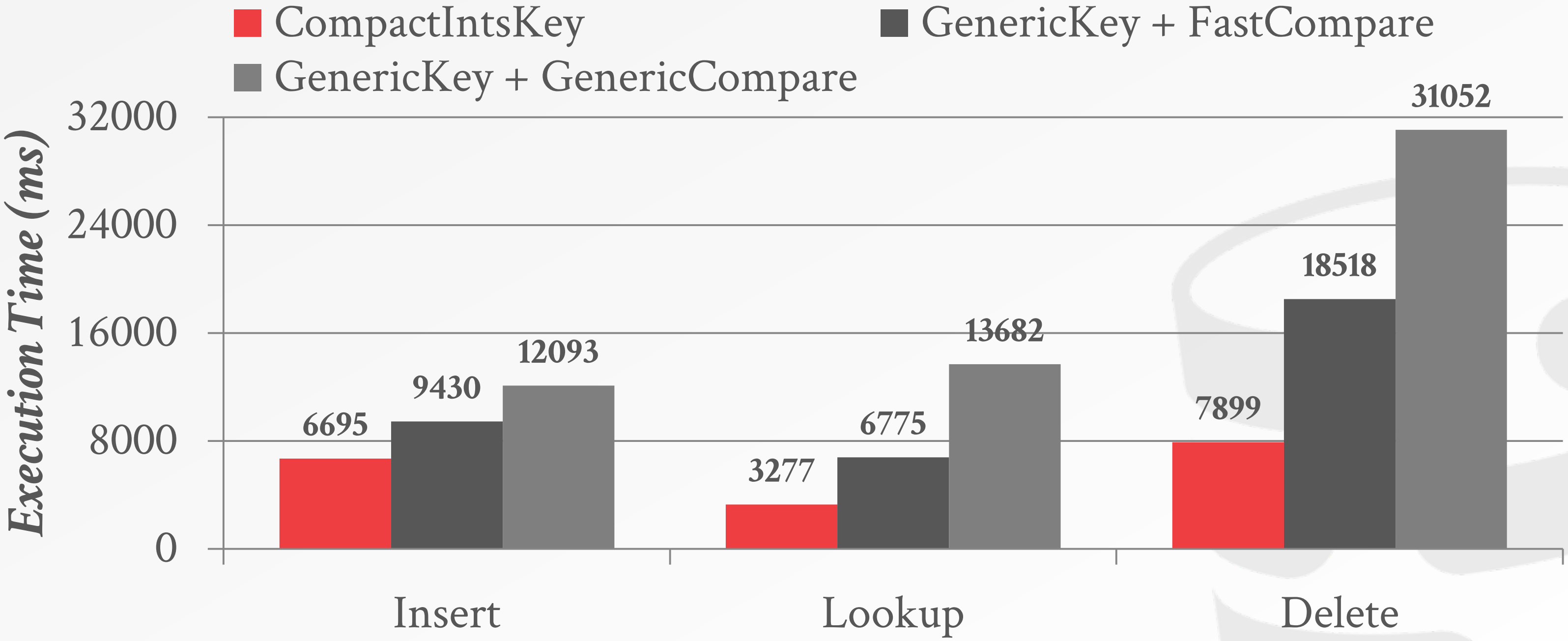
*Big Endian*



*Little Endian*

# BINARY COMPRABLE KEYS

Peloton w/ Bw-Tree Index
Data Set: 10m keys (three 64-bit ints)



# CONCURRENT ART INDEX

HyPer's ART is **not** latch-free.

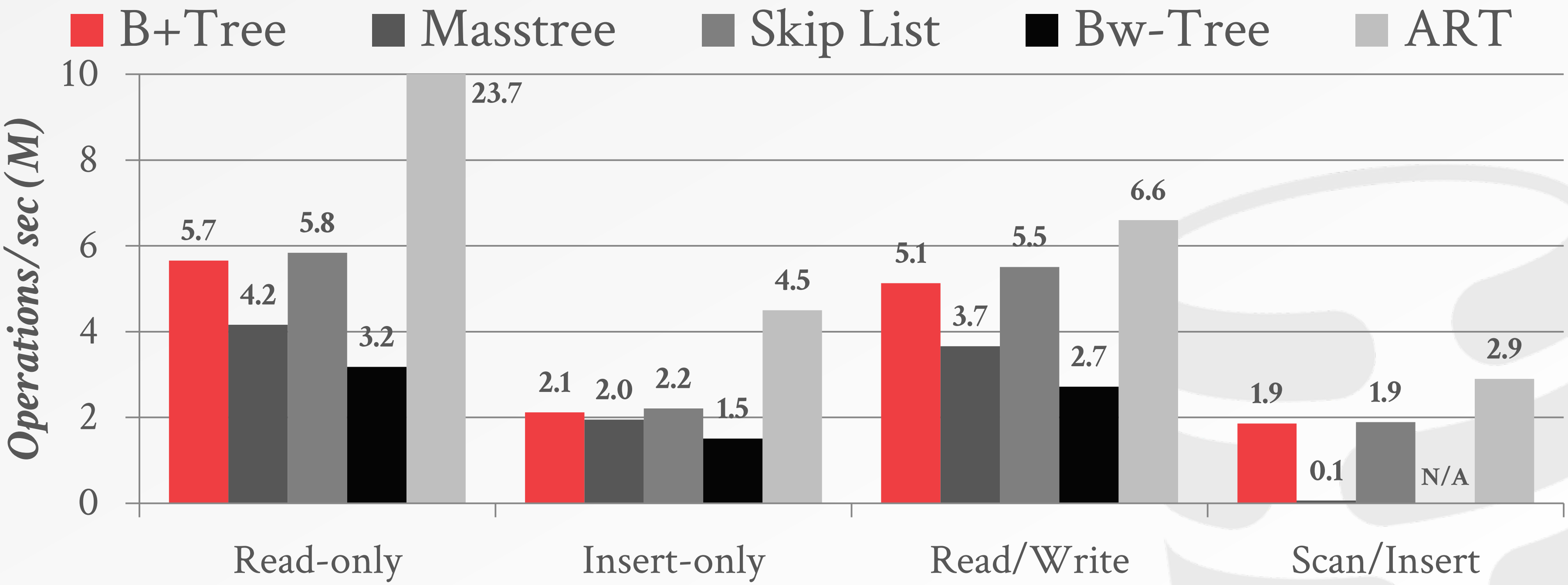
Optimistic crabbing scheme where writers are not blocked on readers.

→ Writers increment counter when they acquire latch.
→ Readers can proceed if a node's latch is available.
→ It then checks whether the latch's counter has changed from when it checked the latch.

THE ART OF PRACTICAL SYNCHRONIZATION
*DaMoN 2016*

# SINGLE-THREADED PERFORMANCE

Data Set: 30m Random 64-bit Integers



Source: Huanchen Zhang

# PARTING THOUGHTS

Bw-Tree is probably the most dank latch-free index in recent years.

ART has amazing performance. Need to understand it better.

*ANDY'S*

# TIPS FOR PROFILING

# MOTIVATION

Consider a program with functions **foo** and **bar**.

How can we speed it up with only a debugger ?
→ Randomly pause it during execution
→ Collect the function call stack

# RANDOM PAUSE METHOD

Consider this scenario

- → Collected 10 call stack samples
- → Say 6 out of the 10 samples were in **`foo`**

What percentage of time was spent in **`foo`**?

- → Roughly 60% of the time was spent in **`foo`**
- → Accuracy increases with # of samples

# AMDAHL'S LAW

Say we optimized **foo** to run 2 times faster

What's the expected overall speedup ?

→ 60% of time spent in **foo** drops in half

→ 40% of time spent in **bar** unaffected

→ ***p*** = percentage of time spent in optimized task

→ ***s*** = speed up for the optimized task

→ Overall speedup = $\frac{\quad}{\quad}$ = 1.4 times faster

# AMDAHL'S LAW

1  0.6 2 +0.4 1 1  0.6 2 +0.4  0.6 2 0.6 0.6 2 2 0.6
2 +0.4 1  0.6 2 +0.4  = 1.4 times faster

1  0.6 2 +0.4 1 1  0.6 2 +0.4  0.6 2 0.6 0.6 2 2 0.6
2 +0.4 1  0.6 2 +0.4  = 1.4 times faster

1 +(1- ) +(1− ) 1
+(1− )

*Say* we optimized **foo** to run 2 times faster

What's the expected overall speedup ?

→ 60% of time spent in **foo** drops in half
→ 40% of time spent in **bar** unaffected

# PROFILING TOOLS FOR REAL

## Choice #1: Valgrind

→ Heavyweight instrumentation framework with a lot of tools

→ Sophisticated visualization tools

## Choice #2: Perf

→ Lightweight tool that can record different kinds of events

→ Console-oriented visualization tools

# CHOICE #1: VALGRIND

Instrumentation framework for building dynamic analysis tools

→ **memcheck**: a memory error detector

→ **callgrind**: a call-graph generating profiler

Using **callgrind** to profile the index test and Peloton in general:

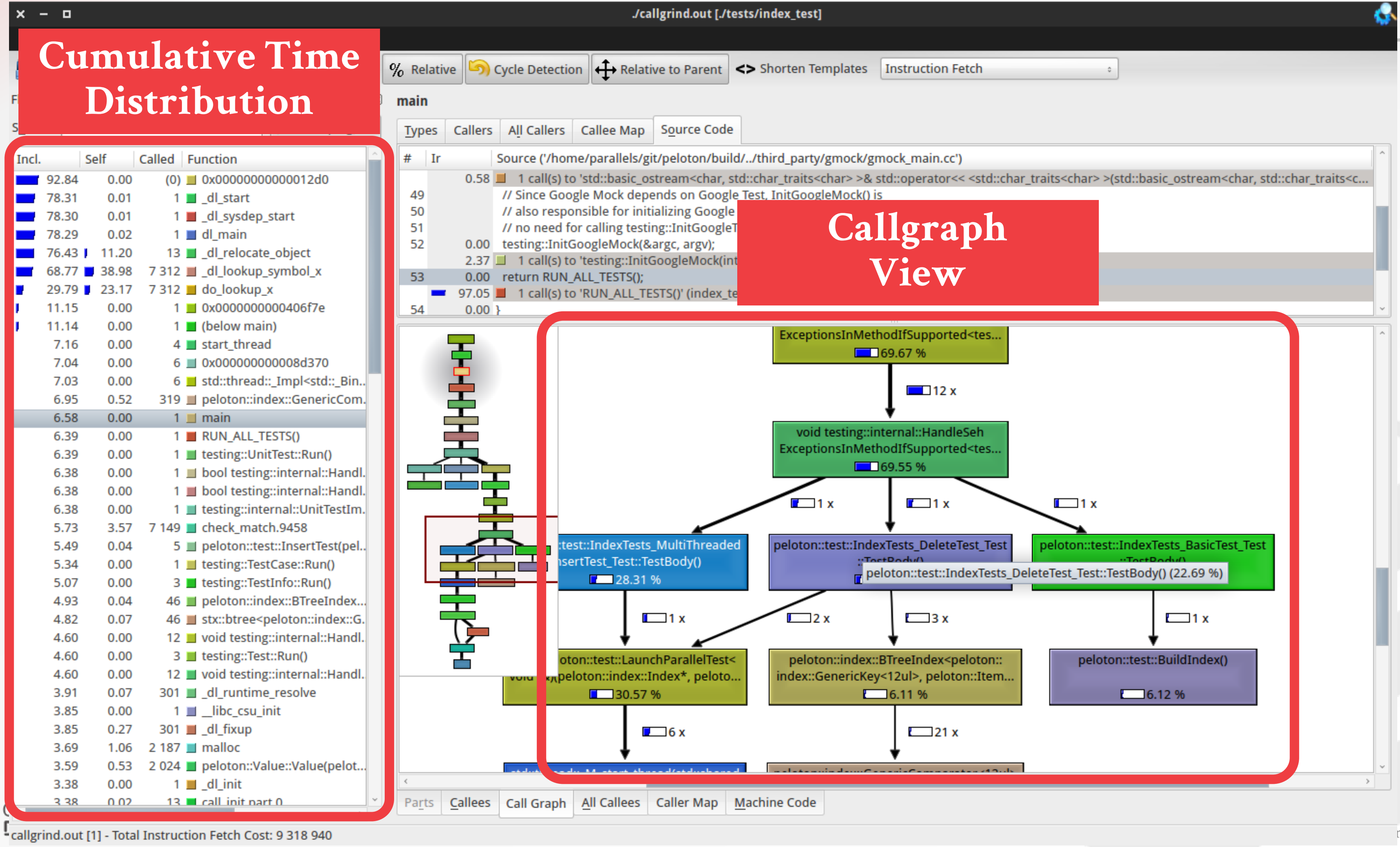
```
$ valgrind --tool=callgrind --trace-children=yes
./tests/skiplist_index_test

$ valgrind --tool=callgrind --trace-children=yes
./bin/peloton &> /dev/null&
```

# KCACHEGRIND

Profile data visualization tool

```
$ kcachegrind callgrind.out.12345
```


./callgrind.out [./tests/index_test]
% Relative
Cycle Detection
Relative to Parent
<> Shorten Templates
Instruction Fetch
main
Types
Callers
All Callers
Callee Map
Source Code
Parts
Callees
Call Graph
All Callees
Caller Map
Machine Code
ExceptionsInMethodIfSupported<tes...
69.67 %
12 x
void testing::internal::HandleSeh
ExceptionsInMethodIfSupported<tes...
69.55 %
1 x
1 x
1 x
test::IndexTests_MultiThreaded
sertTest_Test::TestBody()
28.31 %
peloton::test::IndexTests_DeleteTest_Test
::TestBody()
peloton::test::IndexTests_DeleteTest_Test::TestBody() (22.69 %)
peloton::test::IndexTests_BasicTest_Test
::TestBody()
1 x
2 x
3 x
1 x
oton::test::LaunchParallelTest<
void ...)(peloton::index::Index*, peloto...
30.57 %
peloton::index::BTreeIndex<peloton::
index::GenericKey<12ul>, peloton::Item...
6.11 %
peloton::test::BuildIndex()
6.12 %
6 x
21 x

# Cumulative Time Distribution

# Callgraph View

| Incl. | Self | Called | Function |
|---|---|---|---|
| 92.84 | 0.00 | (0) | 0x00000000000012d0 |
| 78.31 | 0.01 | 1 | _dl_start |
| 78.30 | 0.01 | 1 | _dl_sysdep_start |
| 78.29 | 0.02 | 1 | dl_main |
| 76.43 | 11.20 | 13 | _dl_relocate_object |
| 68.77 | 38.98 | 7 312 | _dl_lookup_symbol_x |
| 29.79 | 23.17 | 7 312 | do_lookup_x |
| 11.15 | 0.00 | 1 | 0x0000000000406f7e |
| 11.14 | 0.00 | 1 | (below main) |
| 7.16 | 0.00 | 4 | start_thread |
| 7.04 | 0.00 | 6 | 0x000000000008d370 |
| 7.03 | 0.00 | 6 | std::thread::_Impl<std::_Bin.. |
| 6.95 | 0.52 | 319 | peloton::index::GenericCom.. |
| 6.58 | 0.00 | 1 | main |
| 6.39 | 0.00 | 1 | RUN_ALL_TESTS() |
| 6.39 | 0.00 | 1 | testing::UnitTest::Run() |
| 6.38 | 0.00 | 1 | bool testing::internal::Handl.. |
| 6.38 | 0.00 | 1 | bool testing::internal::Handl.. |
| 6.38 | 0.00 | 1 | testing::internal::UnitTestIm.. |
| 5.73 | 3.57 | 7 149 | check_match.9458 |
| 5.49 | 0.04 | 5 | peloton::test::InsertTest(pel.. |
| 5.34 | 0.00 | 1 | testing::TestCase::Run() |
| 5.07 | 0.00 | 3 | testing::TestInfo::Run() |
| 4.93 | 0.04 | 46 | peloton::index::BTreeIndex... |
| 4.82 | 0.07 | 46 | stx::btree<peloton::index::G.. |
| 4.60 | 0.00 | 12 | void testing::internal::Handl.. |
| 4.60 | 0.00 | 3 | testing::Test::Run() |
| 4.60 | 0.00 | 12 | void testing::internal::Handl.. |
| 3.91 | 0.07 | 301 | _dl_runtime_resolve |
| 3.85 | 0.00 | 1 | __libc_csu_init |
| 3.85 | 0.27 | 301 | _dl_fixup |
| 3.69 | 1.06 | 2 187 | malloc |
| 3.59 | 0.53 | 2 024 | peloton::Value::Value(pelot... |
| 3.38 | 0.00 | 1 | _dl_init |
| 3.38 | 0.02 | 13 | call_init.part.0 |

```
#   Ir      Source ('/home/parallels/git/peloton/build/../third_party/gmock/gmock_main.cc')
      0.58    1 call(s) to 'std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::basic_ostream<char, std::char_traits<c...
49           // Since Google Mock depends on Google Test, InitGoogleMock() is
50           // also responsible for initializing Google
51           // no need for calling testing::InitGoogleT
52    0.00   testing::InitGoogleMock(&argc, argv);
      2.37    1 call(s) to 'testing::InitGoogleMock(int
53    0.00   return RUN_ALL_TESTS();
     97.05    1 call(s) to 'RUN_ALL_TESTS()' (index_te
54    0.00   }
```

callgrind.out [1] - Total Instruction Fetch Cost: 9 318 940

# CHOICE #2: PERF

Tool for using the performance counters subsystem in Linux.

→ **-e** = sample the event `cycles` at the user level only
→ **-c** = collect a sample every 2000 occurrences of event

```
$ perf record -e cycles:u -c 2000
./tests/skiplist_index_test
```

Uses counters for tracking events

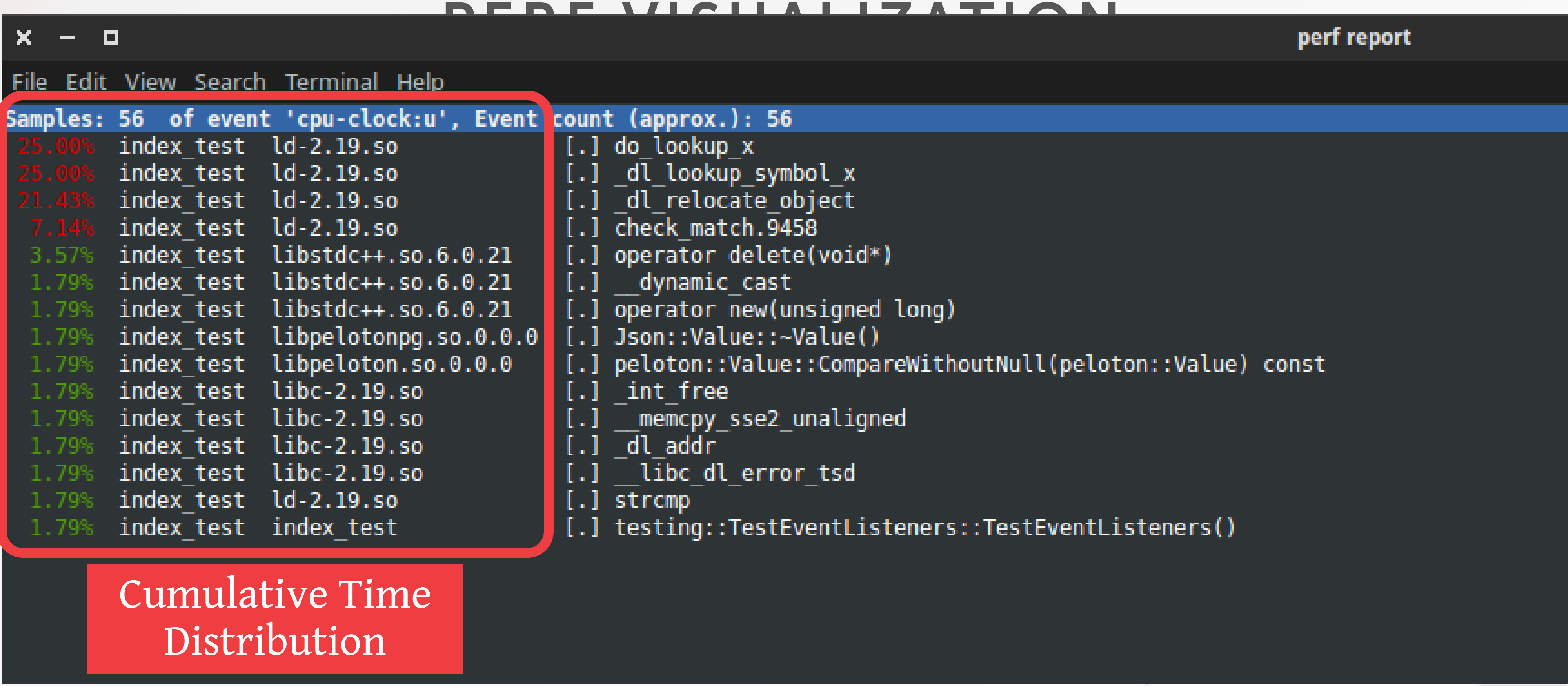
→ On counter overflow, the kernel records a sample
→ Sample contains info about program execution

# PERF VISUALIZATION

We can also use **perf** to visualize the generated profile for our application.

```
$ perf report
```

# PERF VISUALIZATION



```
Samples: 56  of event 'cpu-clock:u', Event count (approx.): 56
 25.00%  index_test  ld-2.19.so            [.] do_lookup_x
 25.00%  index_test  ld-2.19.so            [.] _dl_lookup_symbol_x
 21.43%  index_test  ld-2.19.so            [.] _dl_relocate_object
  7.14%  index_test  ld-2.19.so            [.] check_match.9458
  3.57%  index_test  libstdc++.so.6.0.21   [.] operator delete(void*)
  1.79%  index_test  libstdc++.so.6.0.21   [.] __dynamic_cast
  1.79%  index_test  libstdc++.so.6.0.21   [.] operator new(unsigned long)
  1.79%  index_test  libpelotonpg.so.0.0.0 [.] Json::Value::~Value()
  1.79%  index_test  libpeloton.so.0.0.0   [.] peloton::Value::CompareWithoutNull(peloton::Value) const
  1.79%  index_test  libc-2.19.so          [.] _int_free
  1.79%  index_test  libc-2.19.so          [.] __memcpy_sse2_unaligned
  1.79%  index_test  libc-2.19.so          [.] _dl_addr
  1.79%  index_test  libc-2.19.so          [.] __libc_dl_error_tsd
  1.79%  index_test  ld-2.19.so            [.] strcmp
  1.79%  index_test  index_test            [.] testing::TestEventListeners::TestEventListeners()
```

Cumulative Time Distribution

# PERF EVENTS

Supports several other events like:

→ L1-dcache-load-misses
→ branch-misses

To see a list of events:

```
$ perf list
```

Another usage example:

```
$ perf record -e cycles,LLC-load-misses -c 2000
./tests/skiplist_index_test
```

# REFERENCES

## Valgrind

→ The Valgrind Quick Start Guide
→ Callgrind
→ Kcachegrind
→ Tips for the Profiling/Optimization process

## Perf

→ Perf Tutorial
→ Perf Examples
→ Perf Analysis Tools

# NEXT CLASS

Indexing for OLAP workloads.

→ More from Microsoft Research…