The Bw-Tree: A B-tree for New Hardware Platforms

Author: J. Levandoski et al.

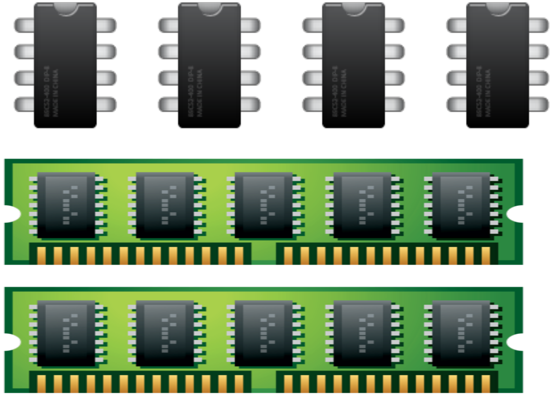
The Bw-Tree: A B-tree for New Hardware Platforms

Buzz **w**ord

DRAM + Flash storage

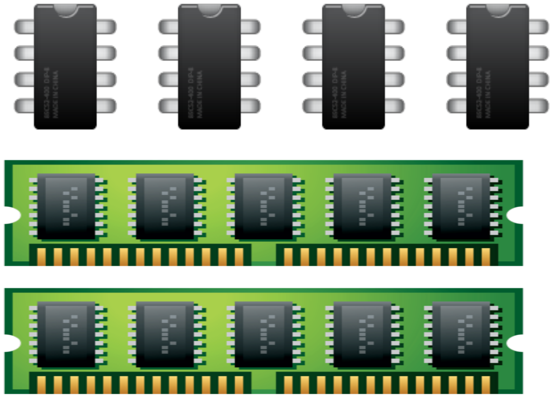
Author: J. Levandoski et al.

Hardware Trends



- Multi-core + large main memories
  - Latch contention
    - Worker threads set latches for accessing data
  - Cache invalidation
    - Worker threads access data from different NUMA nodes

Hardware Trends



- Multi-core + large main memories
  - Latch contention
    - Worker threads set latches for accessing data
  - Cache invalidation
    - Worker threads access data from different NUMA nodes

Delta updates

- No updates in place
- Reduces cache invalidation
- Enable latch-free tree operation

Hardware Trends

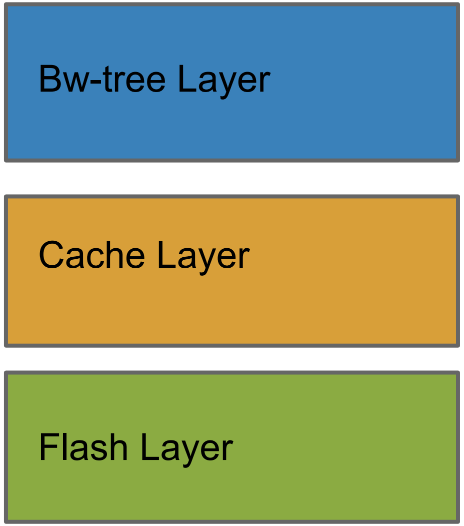
- Flash storage
  - Good at random reads and sequential reads/writes
  - Bad at random writes
    - Erase cycle

Hardware Trends

- Flash storage
  - Good at random reads and sequential reads/writes
  - Bad at random writes
    - Erase cycle

Log-structured storage design

Architecture



Bw-tree Layer

- CRUD API
- Bw-tree search logic
- In-memory pages

Cache Layer

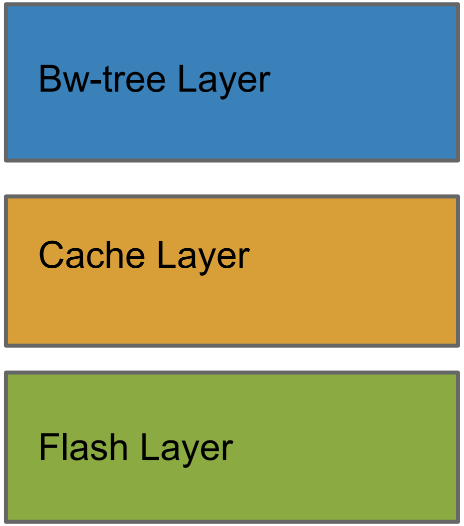
- Logical page abstraction
- Paging between flash and RAM

Flash Layer

- Sequential writes to log-structured storage
- Flash garbage collection

Architecture

Atomic record store, not an ACID transactional database



Bw-tree Layer

- CRUD API
- Bw-tree search logic
- In-memory pages

Cache Layer

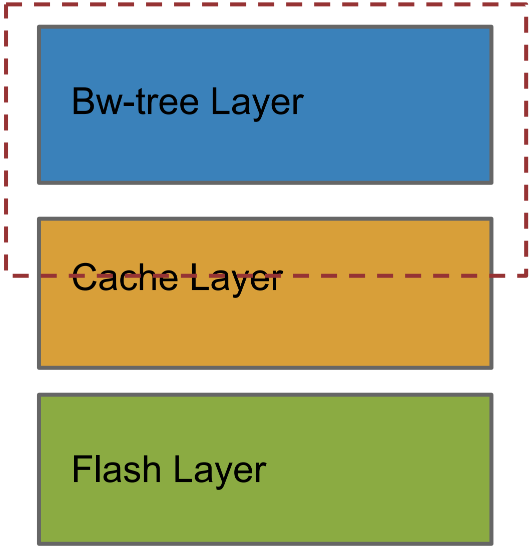
- Logical page abstraction
- Paging between flash and RAM

Flash Layer

- Sequential writes to log-structured storage
- Flash garbage collection

Architecture

Atomic record store, not an ACID transactional database

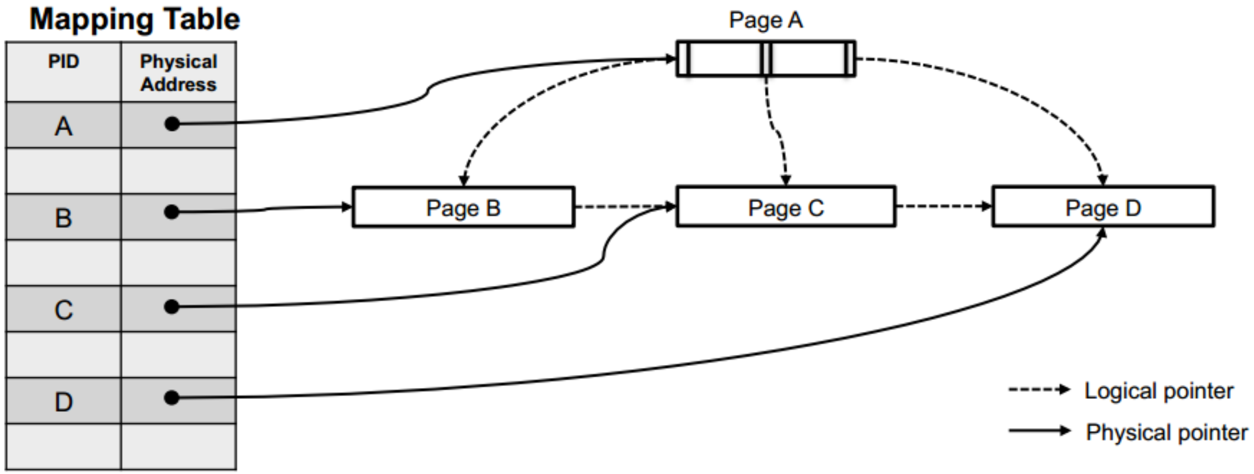


- CRUD API
- Bw-tree search logic
- In-memory pages

- Logical page abstraction
- Paging between flash and RAM

- Sequential writes to log-structured storage
- Flash garbage collection

Logical Pages and Mapping Table



Mapping Table

PID	Physical Address
A	●
B	●
C	●
D	●

Page A

Page B

Page C

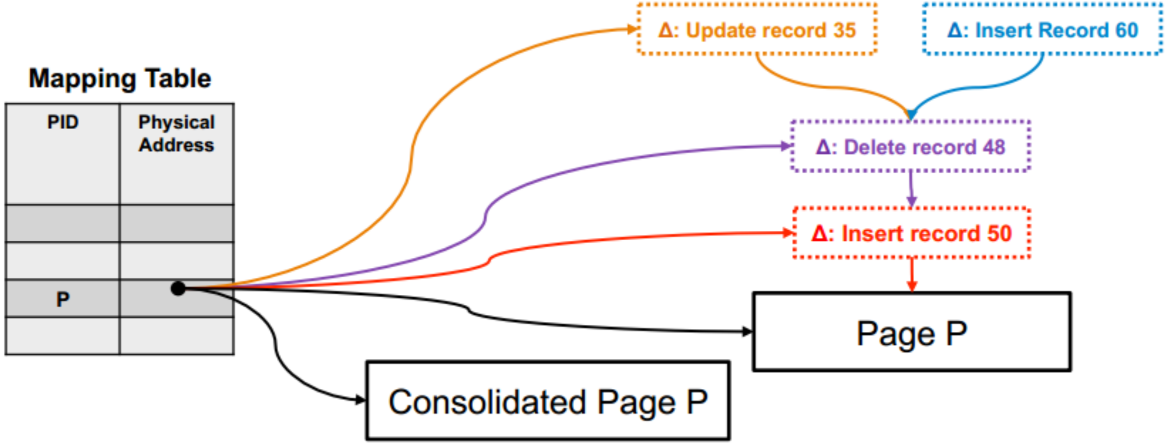
Page D

Logical pointer

Physical pointer

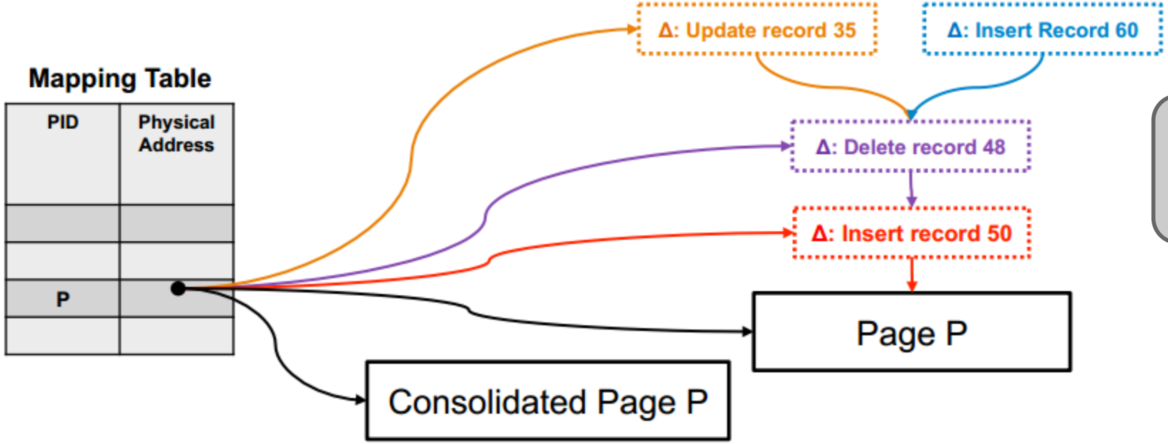
- Logical pages are identified by PIDs stored as Mapping Table keys.
- Physical addresses can be either in main memory or in flash storage.

Delta Updates



- Tree operations are atomic.
- Update operations are “logged” as a lineage of delta records.
- Delta records are incorporated to the base page asynchronously.
- Updates are “installed” to Mapping Table through compare-and-swap.
- Important enabler for latch-freedom and cache-efficiency.

Delta Updates



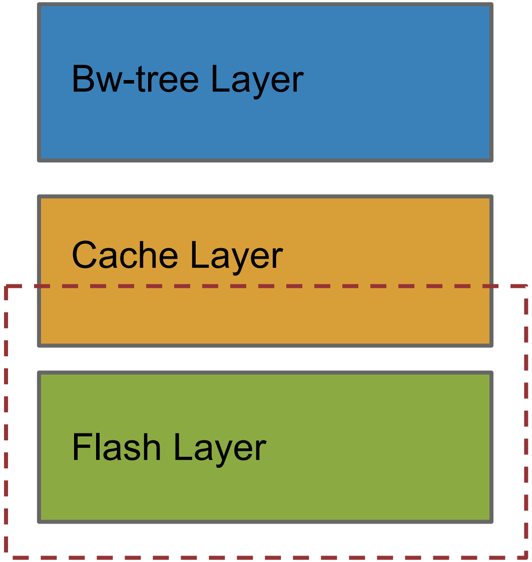
Q: What is the performance of reading data from page P?

- Tree operations are atomic.
- Update operations are “logged” as a lineage of delta records.
- Delta records are incorporated to the base page asynchronously.
- Updates are “installed” to Mapping Table through compare-and-swap.
- Important enabler for latch-freedom and cache-efficiency.

Other details

- SMO: structure modification operations
    - split, merge, consolidate
    - has multiple phases -> how to make SMO atomic?
- In-memory page garbage collection
    - epoch-based.

Architecture

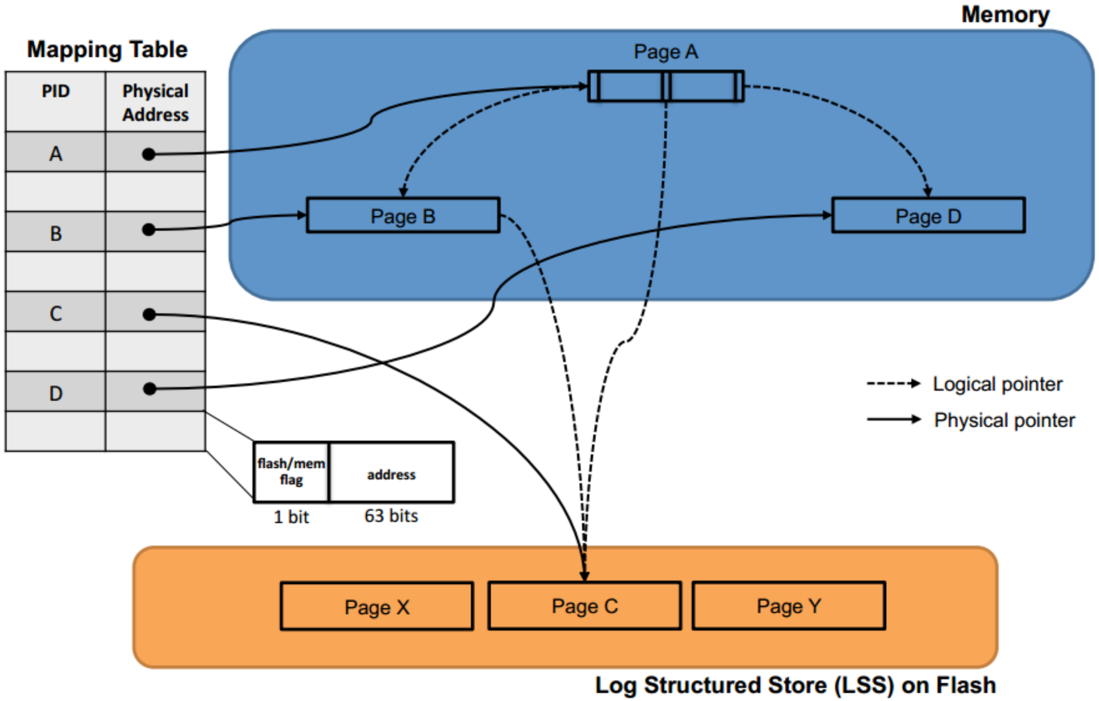


- CRUD API
- Bw-tree search logic
- In-memory pages

- Logical page abstraction
- Paging between flash and RAM

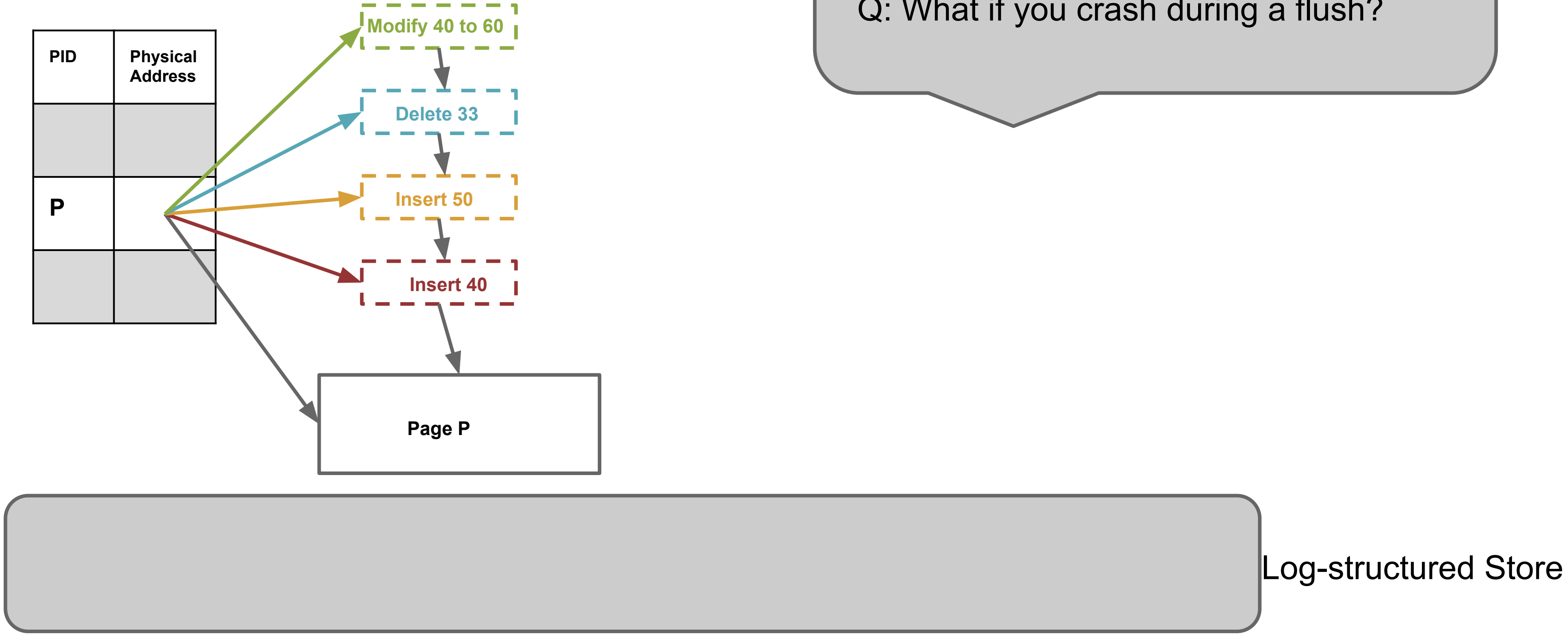
- Sequential writes to log-structured storage
- Flash garbage collection

Flash Layer

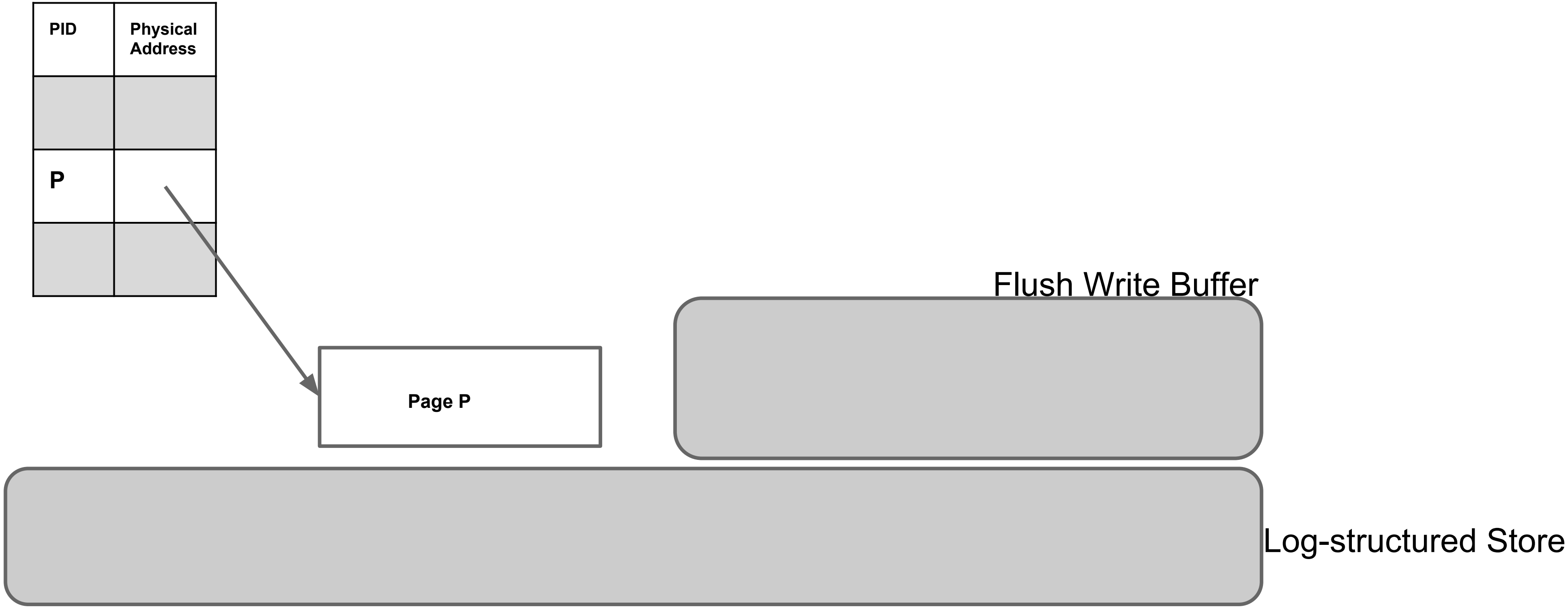

Memory
Mapping Table
PID
Physical Address
A
B
C
D
Page A
Page B
Page D
flash/mem flag
address
1 bit
63 bits
Logical pointer
Physical pointer
Page X
Page C
Page Y
Log Structured Store (LSS) on Flash

Flushing Pages

Q: Why flushing pages?
Q: When to flush pages?
Q: How many pages to flush?
Q: What if you crash during a flush?

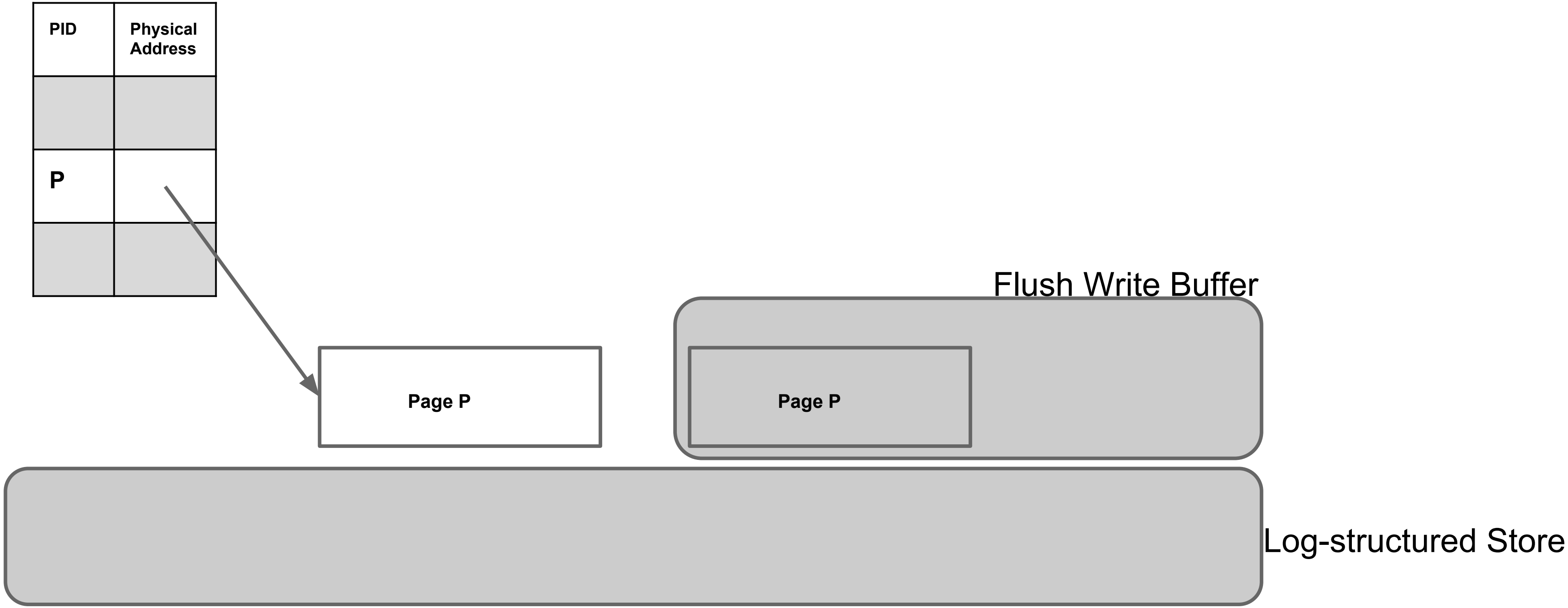


Flushing Pages



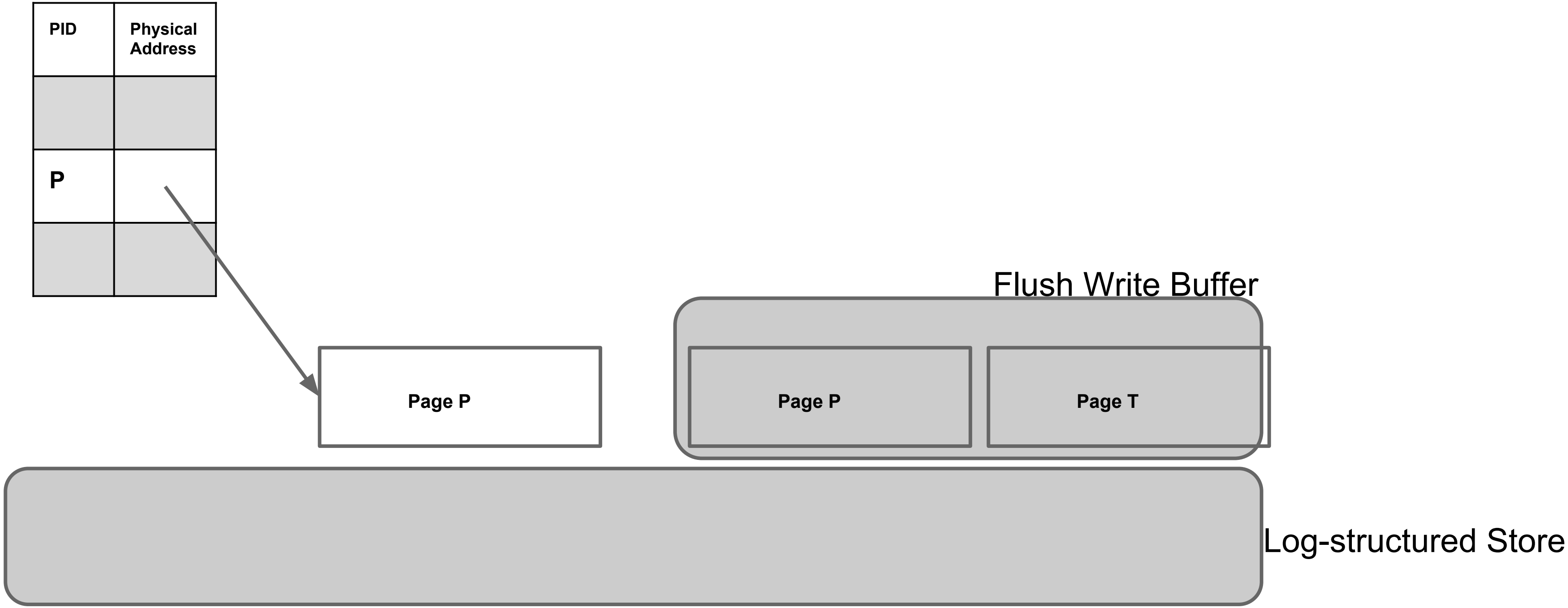
PID	Physical Address
P	

Flushing Pages



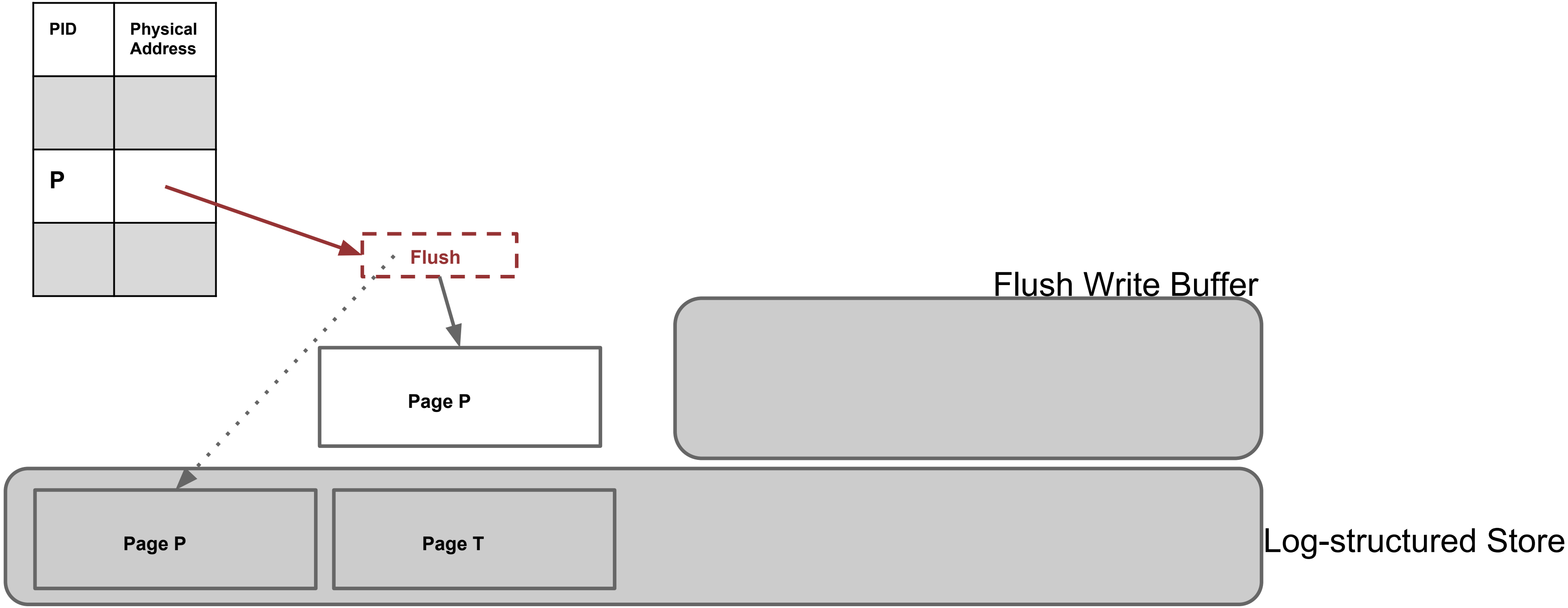
PID	Physical Address
P	

Flushing Pages



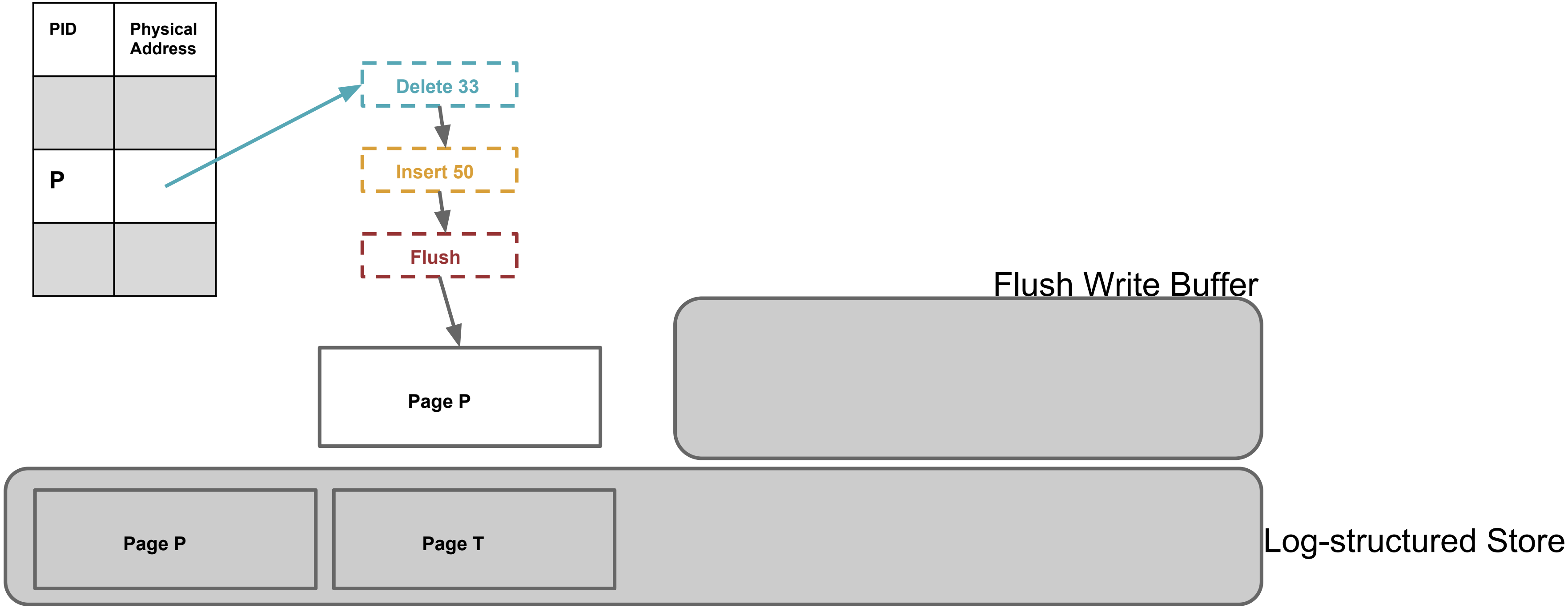
PID	Physical Address
P	

Flushing Pages



PID	Physical Address
P	

Flushing Pages



PID	Physical Address
P	

Delete 33

Insert 50

Flush

Page P

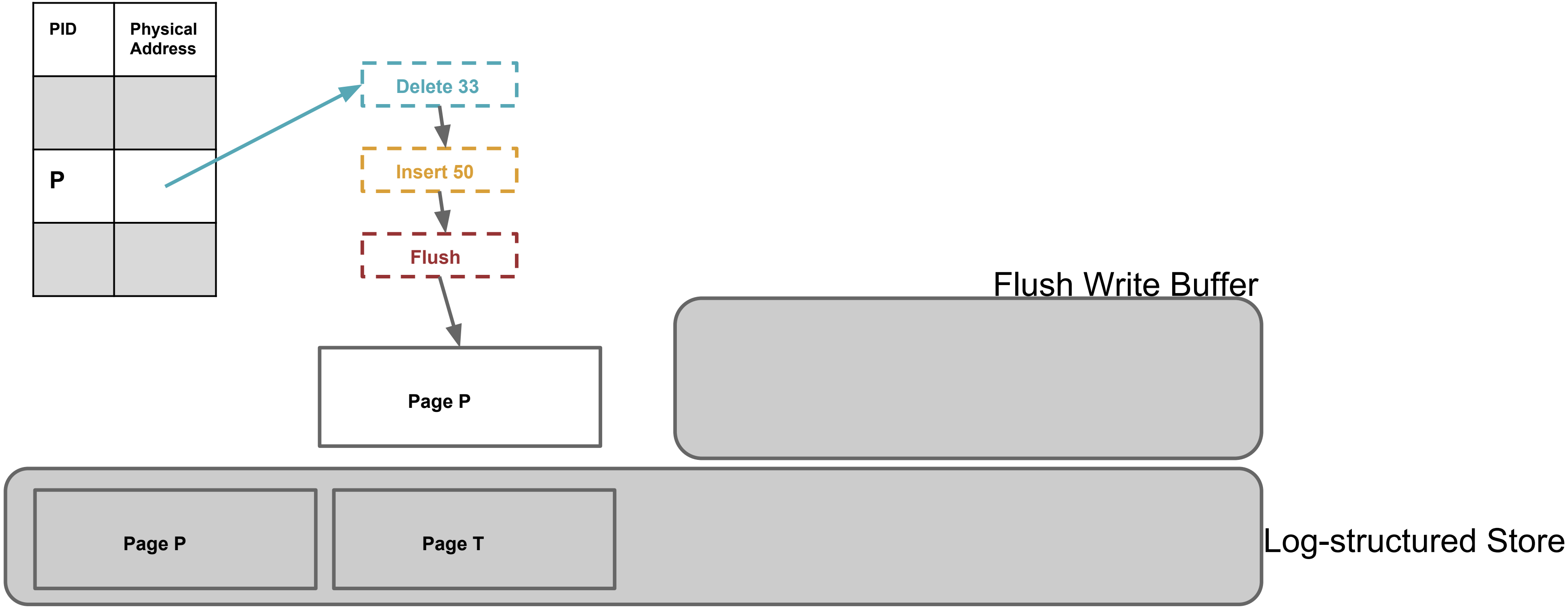
Flush Write Buffer

Page P

Page T

Log-structured Store

Flushing Pages



PID

Physical Address

P

Delete 33

Insert 50

Flush

Page P

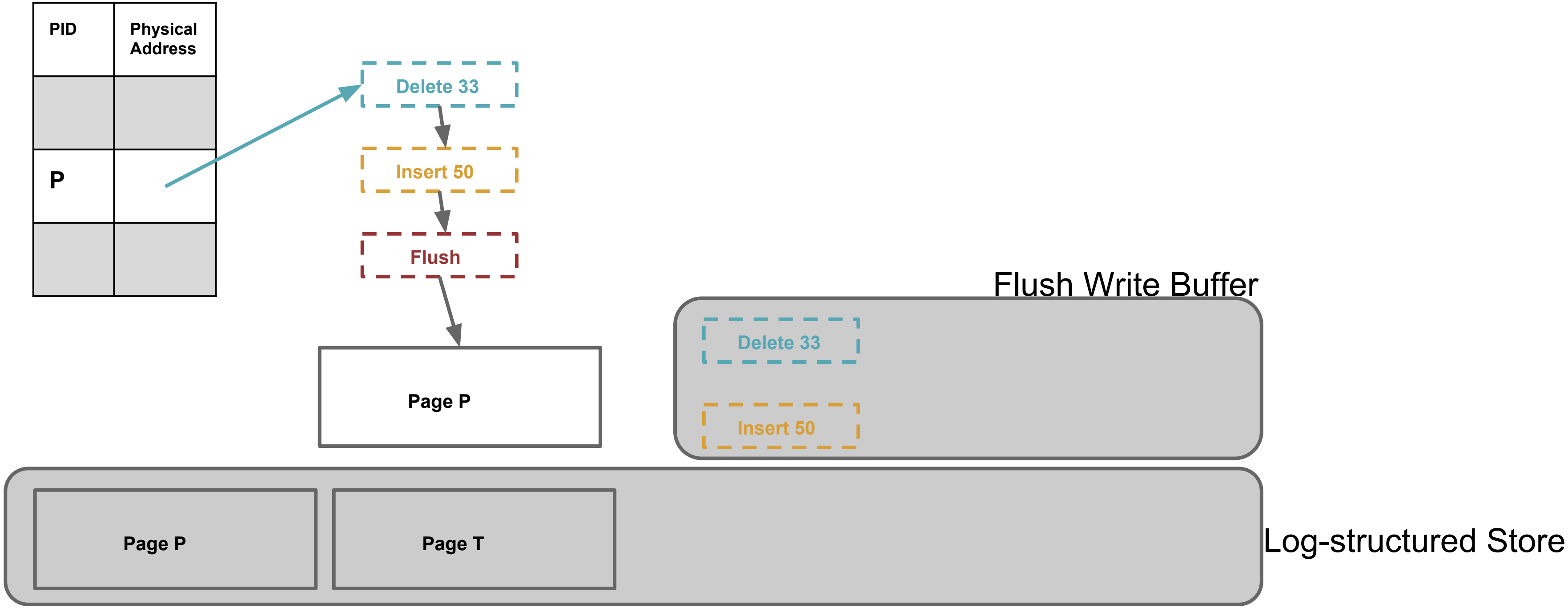
Flush Write Buffer

Page P

Page T

Log-structured Store

Flushing Pages



PID

Physical Address

P

Delete 33

Insert 50

Flush

Page P

Flush Write Buffer

Delete 33

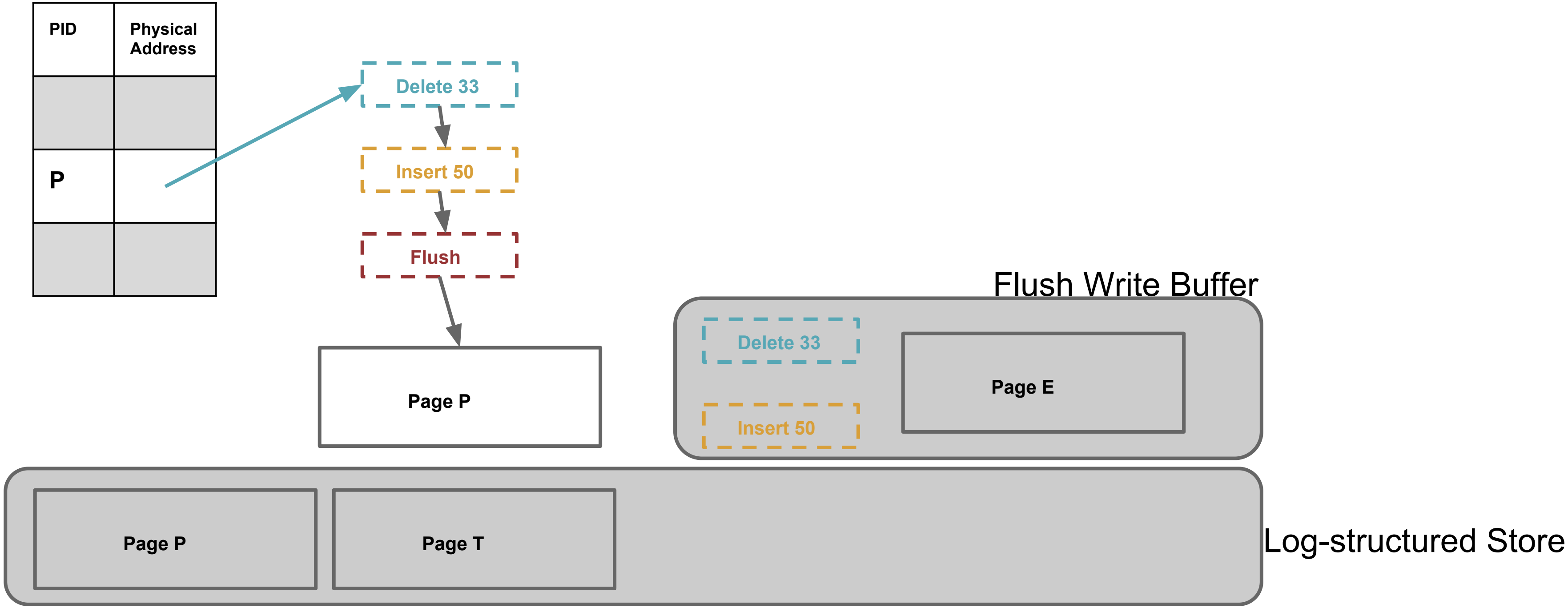
Insert 50

Page P

Page T

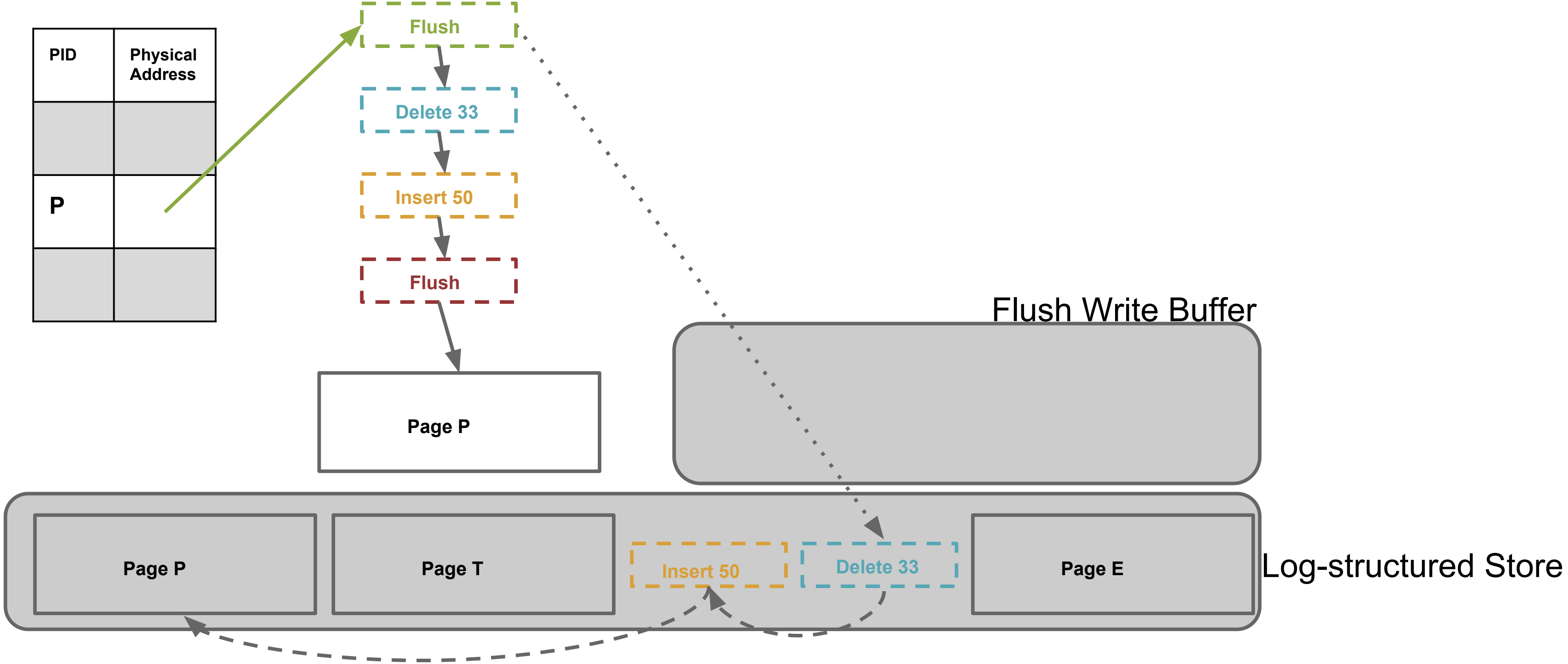
Log-structured Store

Flushing Pages



PID	Physical Address
P	

Flushing Pages



PID	Physical Address
P	

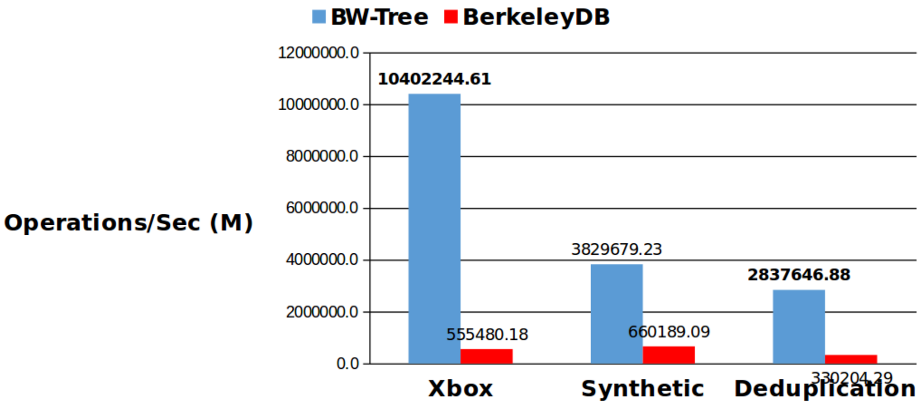
Other details

- Log-structured Store garbage collection
  - Cleans orphaned data unreachable from mapping table
  - Relocates entire pages in sequential blocks (to reduce fragmentation)
- Access method recovery
  - Occasionally checkpoint mapping table
  - Redo-scan starts from last checkpoint

Experiment

- Against
  - BerkeleyDB (without transaction)
  - latch-free skip-list

Experiment



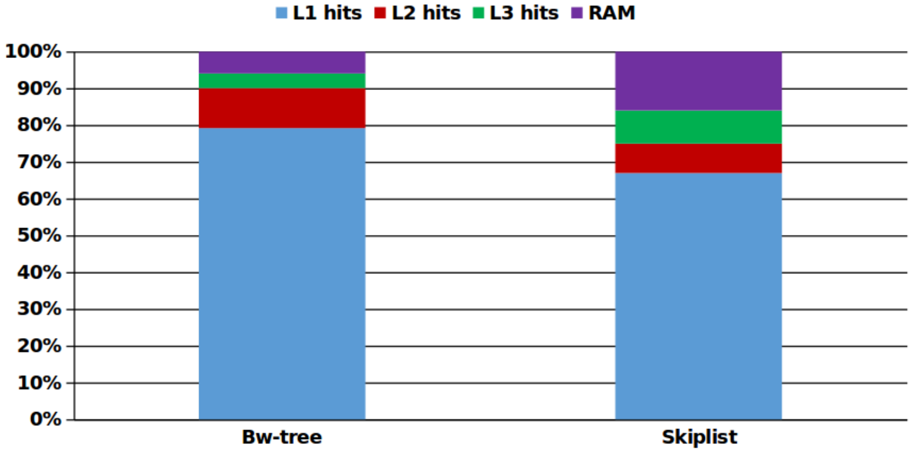
Over BerkerleyDB:

- 18x speedup in read-intensive workload
- 5-8x speedup in update-intensive workload

Over Skip-list:

- 4.4x speedup in read-only workload.
- 3.7x speedup in update-intensive workload.

	Bw-Tree	Skiplist
Synthetic workload	3.83M Ops/Sec	1.02 M Ops/Sec



Thank you!

Slides adapted from http://www.hpts.ws/papers/2013/bw-tree-hpts2013.pdf