MySQL Proposal: InnoDB B+Tree Performance optimization

Insert Path Improvment and Concurrent Split Handing

Zhao Song 2026-05-26

About me

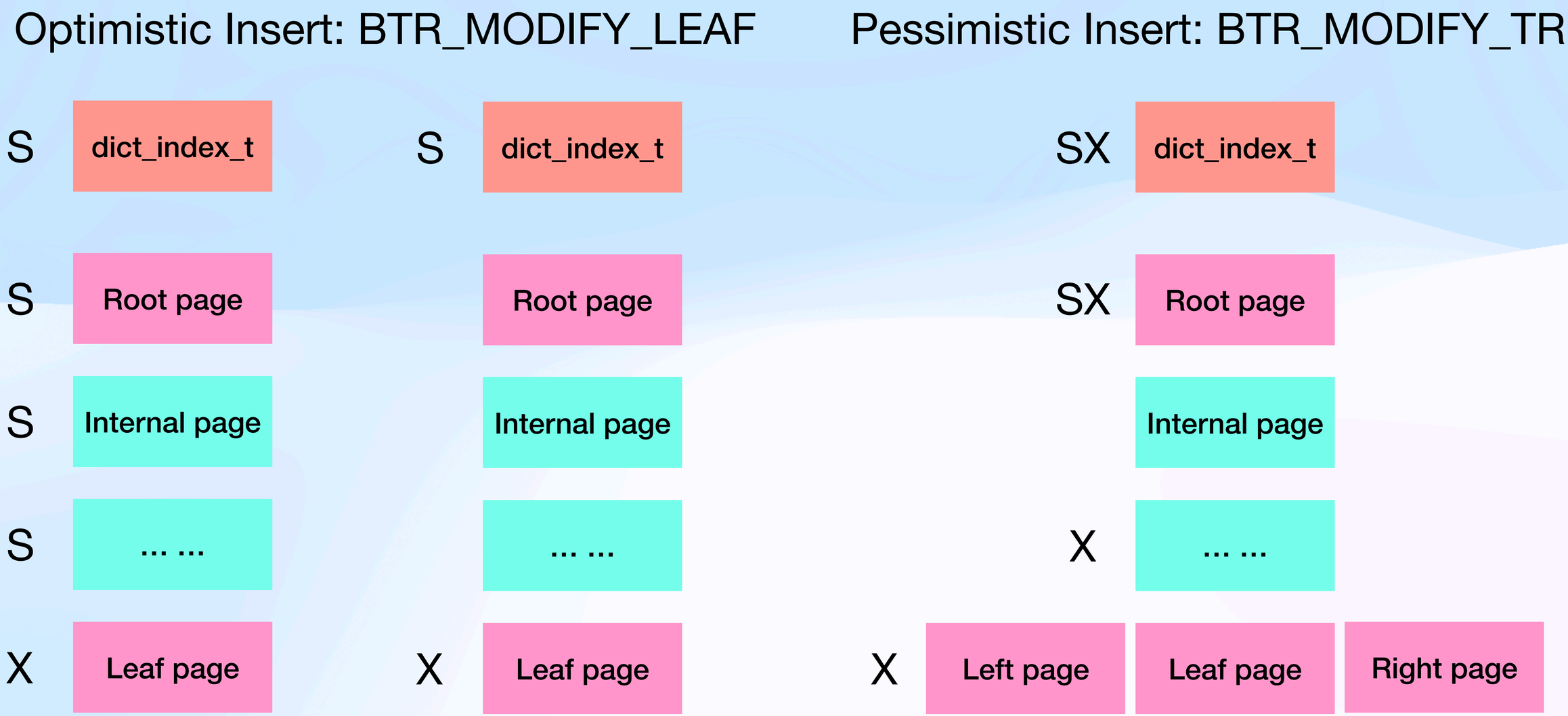
Database nerd

2023.10 - Present		Hopsworks AB	RonDB (MySQL NDB Cluster)
2018.03 - 2023.10		Alibaba	PolarDB (MySQL InnoDB)
2014.07 - 2018.03		Qihoo360	Pika (Redis, RocksDB)

Current InnoDB B+Tree Bottlenecks

Bottleneck	Root Cause	Effect	Serverity
Serialized SMOs	BTR_MODIFY_TREE acquires dict_index_t::lock SX	No parallel page splits	☹☹☹
Long SMO latch duration	BTR_MODIFY_TREE holds X-latches on the pages in the affected subtree	No concurrent reads/writes within the subtree	☹☹☹
Repeated B+Tree descents	Optimistic insert descends the B+Tree once Pessimistic insert descends it at least twice	Unnecessary tree descents	☹

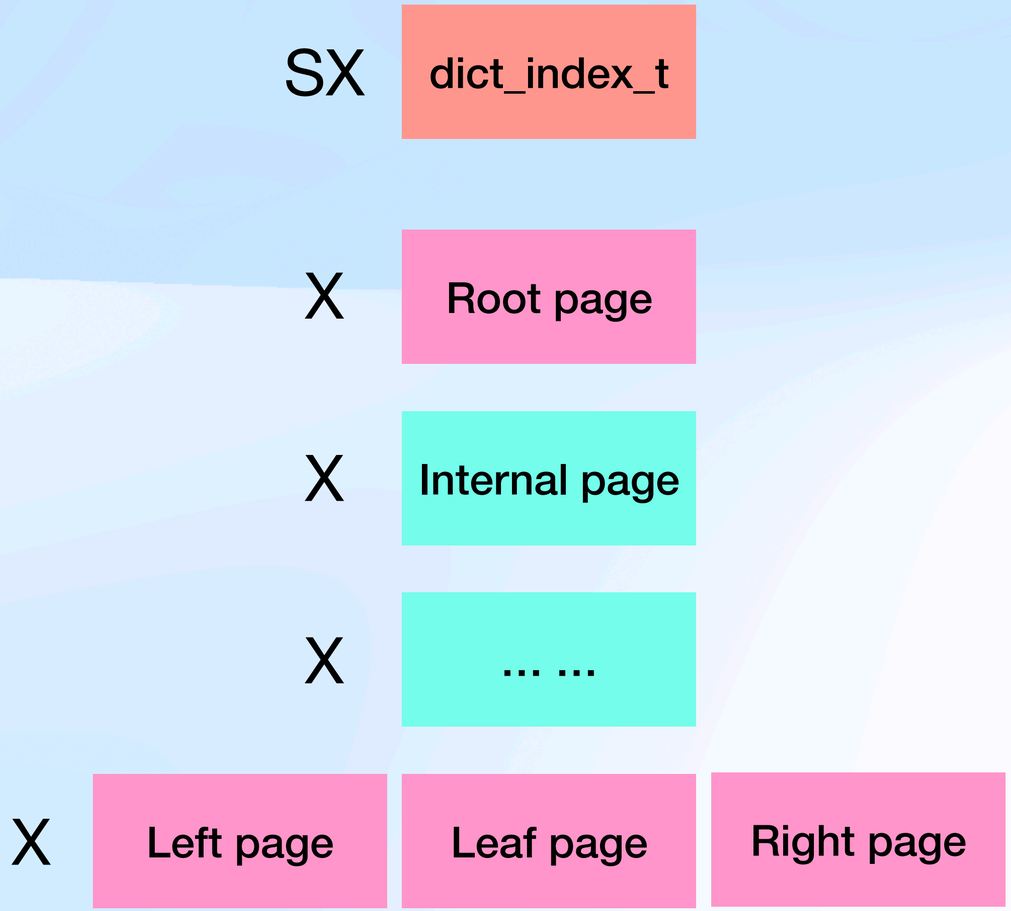
Current InnoDB B+Tree Insert Flow



Current InnoDB B+Tree Insert Flow

Impact of pessimistic insert:

Worst case:



The entire subtree is pre-X-latched during tree descent until cascading splits finish:

1. No concurrent splits on the B+Tree
2. No concurrent reads/writes on the subtree

Proposal: Improve B+Tree Insert Performance

By addressing these bottlenecks

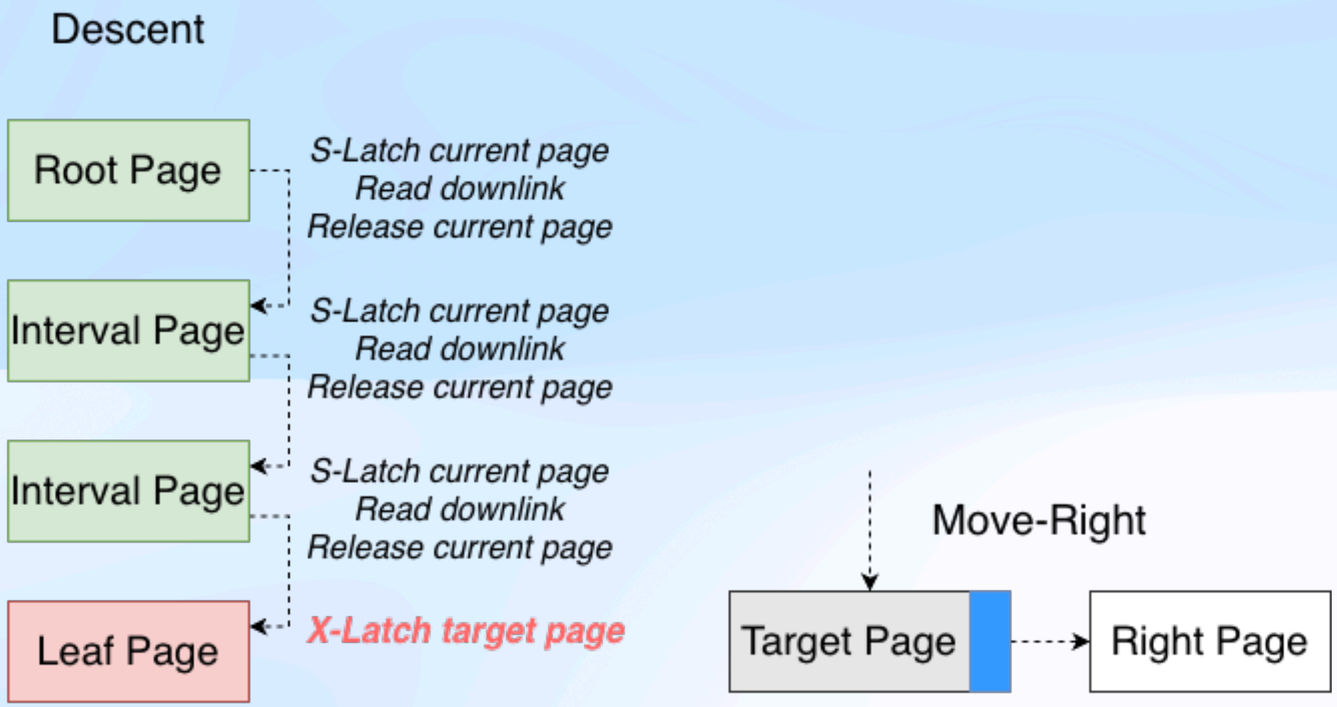
Bottleneck	Goal	Solution
Serialized SMOs	Enable parallel SMOs	B-link-style split protocol
Long SMO latch duration	Reduce SMO latch scope	
Repeated B+Tree descents	Avoid unnecessary tree descents	Smooth transition from optimistic to pessimistic insert

B-link-style split protocol

- Each page stores a **high key** and a **right link**
- A page split first creates a new right sibling and updates the key range at the current level
- The new sibling becomes reachable through the right link before its downlink is fully installed in the parent
- Searches that land on an old page can move right using the high key
- Parent updates are propagated upward using the same protocol

High-level design

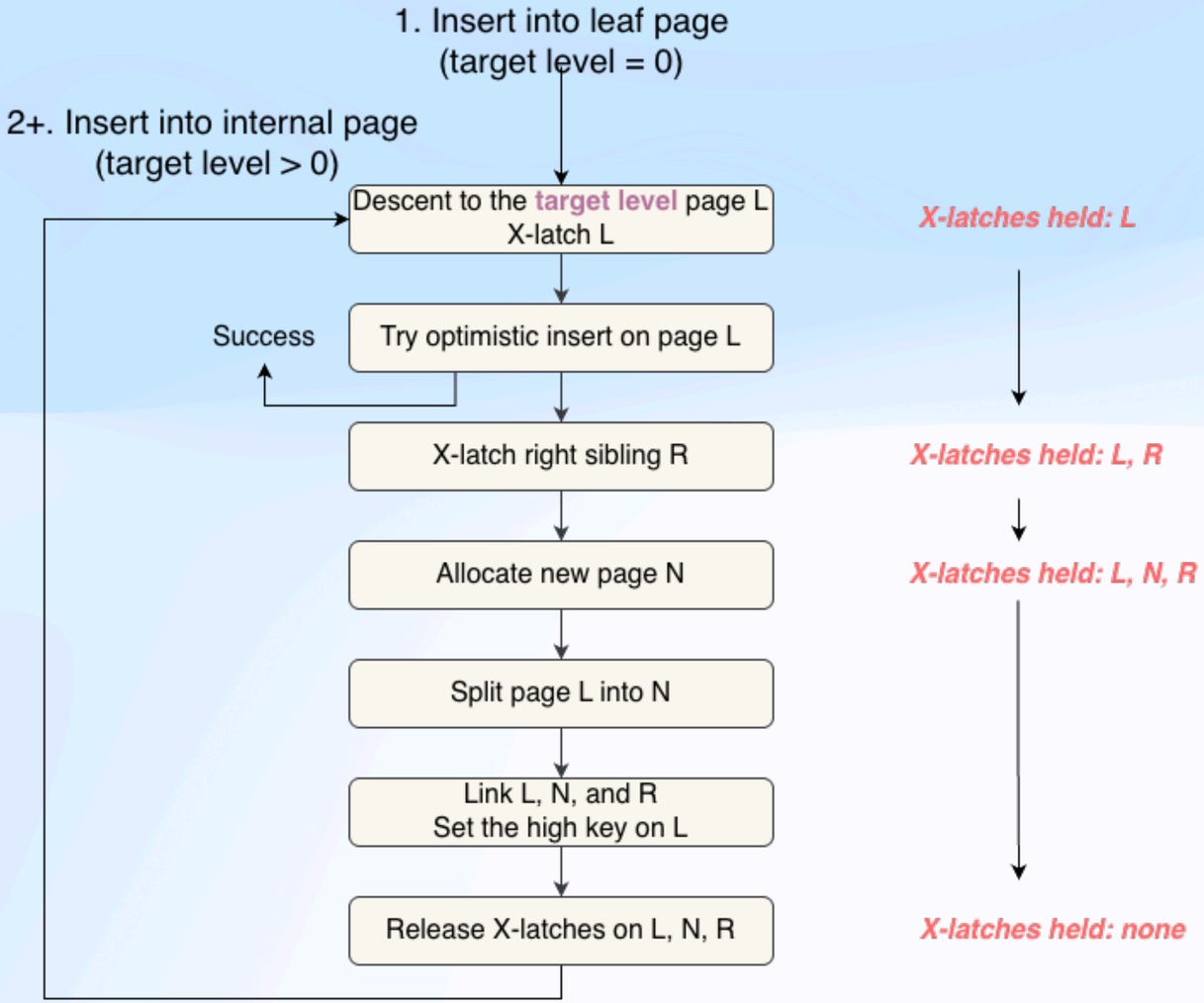
1. Tree descent and Move-Right



If search key >= **high key**, move right

High-level design

2. Insert Flow



Low-level design

How to implement this in InnoDB?

The devil is in the details…

Low-level design

1. How to Implement the High Key?

- Avoid compatibility issues
  - PAGE_IS_BLINK: highest bit of the 2-byte PAGE_DIRECTION field
  - PAGE_INCOMPLECT_SPLIT: second-highest bit of …
  - Store high key as the previous record of supremum record
- Avoid correctness issues
  - Skip the high key in most record-iteration paths
  - Skip the high key when setting record locks
  - … …

Low-level design

2. How to break down the SMO mini-transaction(mtr)?

- 1 mtr : one-level processing
  - mtr_1:
    - Descend to leaf page L
    - Try optimistic record insert, If it fails, split L, set the INCOMPLETE_SPLIT flag to L
  - mtr_2:
    - Opt X-latch / descend to parent page P
    - Try optimistic node_ptr insert, If it fails, split P
    - X-latch L and clear the flag
  - mtr_3…: Repeat mtr_2 at the next upper level

- What breaks:
  - No consistent index-lock ownership across the SMO
- How to solve it:
  - Add a new interface: mtr_t::transfer_to(...)
  - mtr_2.start()

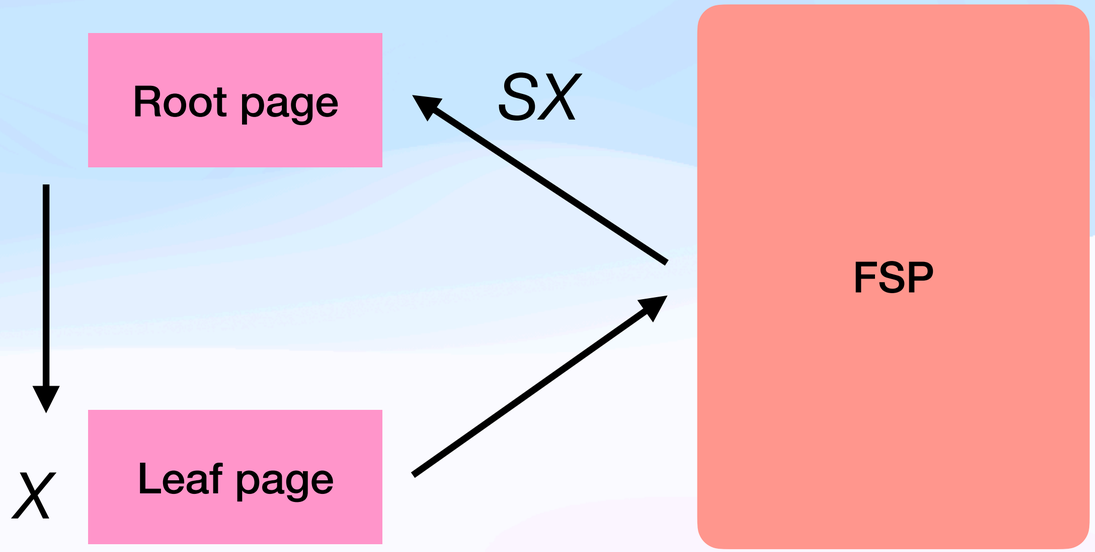
    mtr_1.transfer_to (mtr_2, index S)

    mtr_1.commit()

Low-level design

3. InnoDB Historical Constraint: FSP

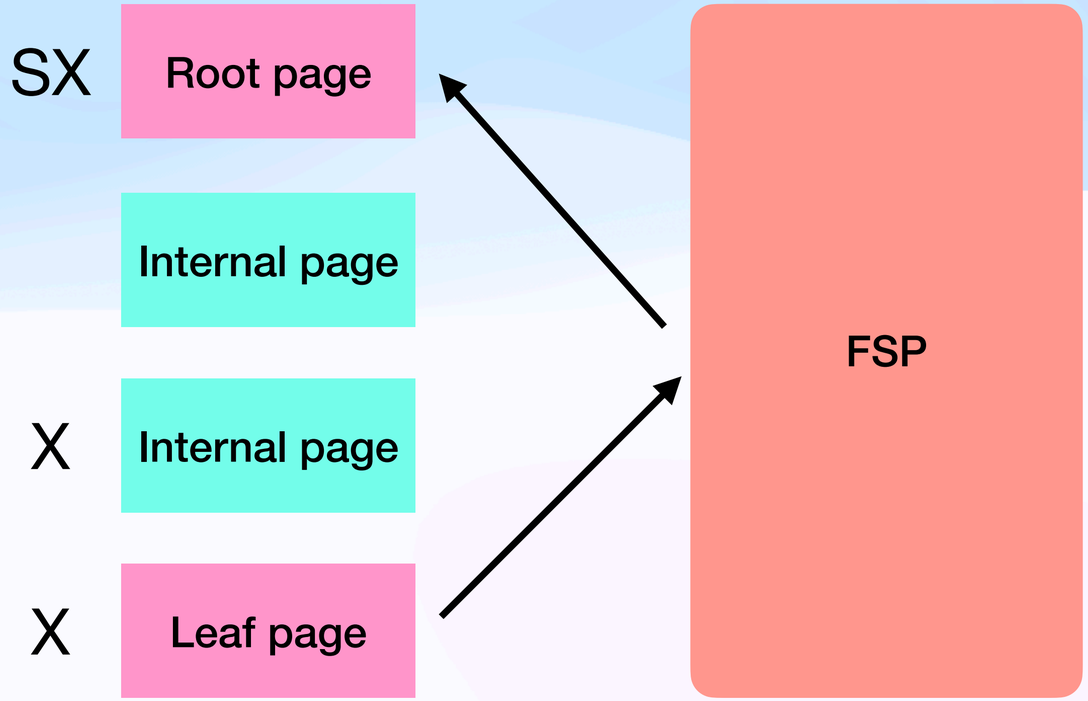
- SMOs need to allocate pages from FSP while holding page latches
- FSP needs to acquire the root page's SX-latch to operate
- Root page latch becomes involved in an external component: FSP
- Down-locking + up-locking = deadlock risk



Low-level design

3. InnoDB Historical Constraint: FSP

- InnoDB handles this by holding the root page's SX-latch throughout the SMO



Low-level design

3. InnoDB Historical Constraint: FSP

- Choices for B-link-style SMOs:

Hold root SX-latch throughout the SMO	Still no parallel SMOs	No
Remove the root page's role from FSP	Compatibility issue	No
Use an up-locking order	1. Clearing the child page flag and inserting the node_ptr into the parent must be atomic, which still requires down-locking 2. Root SX-latch is still held longer than necessary	No
Reduce page allocation mtr scope	Page leak issue	50% YES
Async FSP preallocator	?	75% YES

Low-level design

3. InnoDB Historical Constraint: FSP

- Async FSP preallocator
  - Removes page allocation from the insert path through on-demand preallocation
  - May help prevent page leaks with a more comprehensive design
  - B-link needs a window between removing a page from the tree and actually freeing it in FSP, a well-designed async maintainer can take over this responsibility

Low-level design

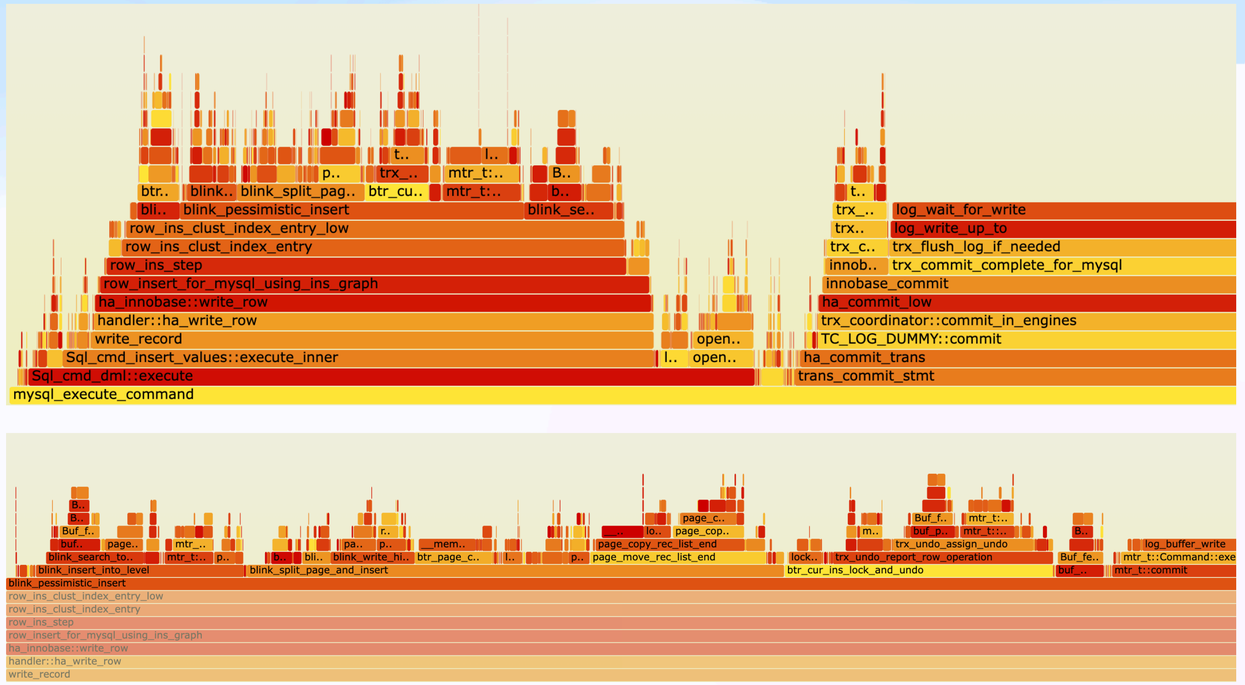
4. Crash Recovery

- A crash may leave an SMO in a state where the child page has been split, but the new page’s node_ptr has not yet been inserted into the parent page
  - No correctness issue
  - After recovery, scan the buffer pool and help pages with the INCOMPLETE_SPLIT flag finish the remaining SMO work
  - Future operations can also help complete in-flight SMOs

Benchmark 1

POC Based on MySQL 9.7.0

- **Split-heavy insert workload**
  - Populate 2.5M rows ~2.5KB per row, so each leaf page is close to the split threshold
  - Insert 400K evenly distributed rows to trigger frequent page splits, and **measure this phase**



	TPS	Avg latency	P95 latency
MySQL 9.7.0	5,666.22	5.65 ms	17.01 ms
Poc	91,523.92	0.35 ms	0.40 ms
	16.1x ↑		42.6x ↓

Benchmark 2

POC Based on MySQL 9.7.0

- **sysbench oltp_insert**
  - 1M insert events into an empty table

auto_inc ON

	TPS	P95 latency
MySQL 9.7.0	80223	1.64 ms
Poc	104,776	0.90 ms
	30.6% ↑	45.1% ↓

auto_inc OFF

	TPS	P95 latency
MySQL 9.7.0	194,912	0.19 ms
Poc	245,161	0.19 ms
	25.8% ↑	

Adoption Plan

Follow the MySQL Proposal Guidelines

- “Low risk”
  - Do not modify the existing B+Tree code path directly
  - Implement the new protocol in separate new functions
  - Dispatch to the new protocol from row_ins_clust_index_entry_low() and row_ins_sec_index_entry_low() with using feature switches and feature preconditions
- Easy adoption
  - Multiple iterations: 1. Split; 2. Merge; 3 …
  - V1: Split: S(index->lock); S(index->smo); Merge: SX(index->lock); X(index->smo)

Thank you