Tabular: Efficiently Building Efficient Indexes

Ziyi Yan, Mohammed Farouk Drira, Tianxun Hu, Tianzheng Wang
Simon Fraser University
https://github.com/sfu-dis/tabular

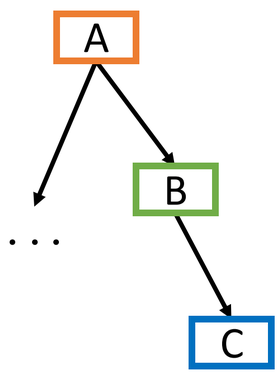
SFU SIMON FRASER UNIVERSITY

What? Concurrent indexes are hard to build, debug and maintain in varying compute/storage hierarchies.

Why? Error-prone synchronization primitives and locking protocol that overcomplicates memory management.

How? Re-use DBMS concurrency control to provide easy-to-use transactions to build concurrent indexes.

Consider B+-tree on transactional tables



(a) Logical view

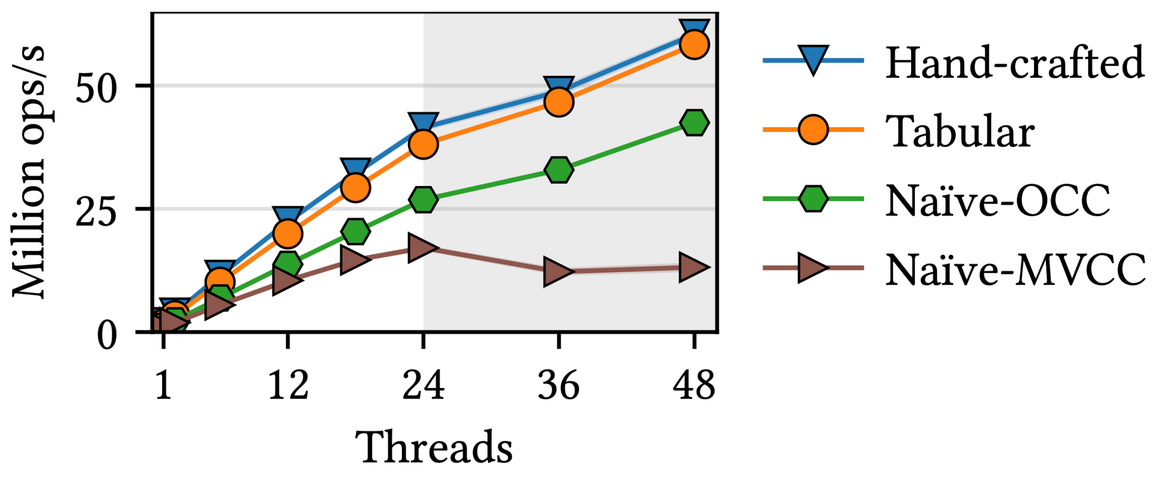
```
struct BTreeNode {
  int capacity;
  int n_keys;
  bool is_leaf;
  int lock;
  BTreeNode
      *right_child;
  KVPair kvs[16];
};
```
(b) C/C++ struct/class definition

Map struct field to column

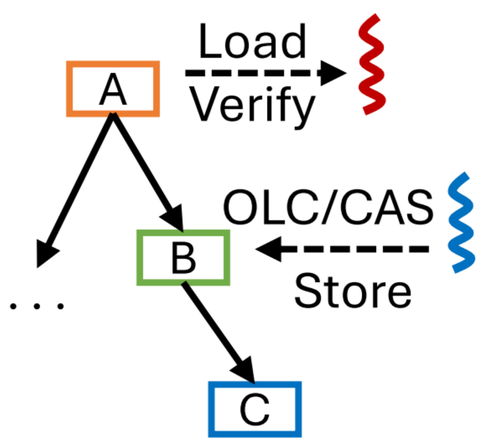
RID	n_keys	is_leaf	right_child	kvs
0	10	False	1	...
1	8	False	2	...
2	4	True	INVALID_RID	...
...				

(c) Table storing BTreeNode objects

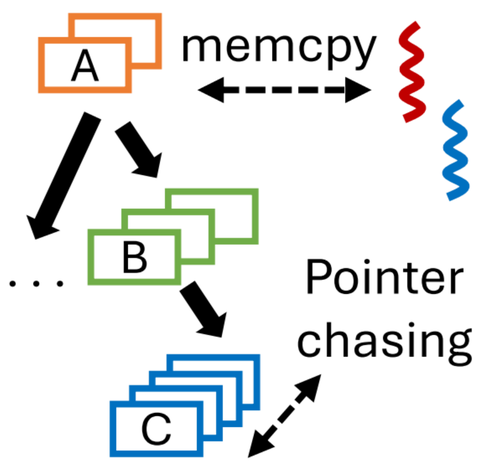
Naïve approaches are too slow



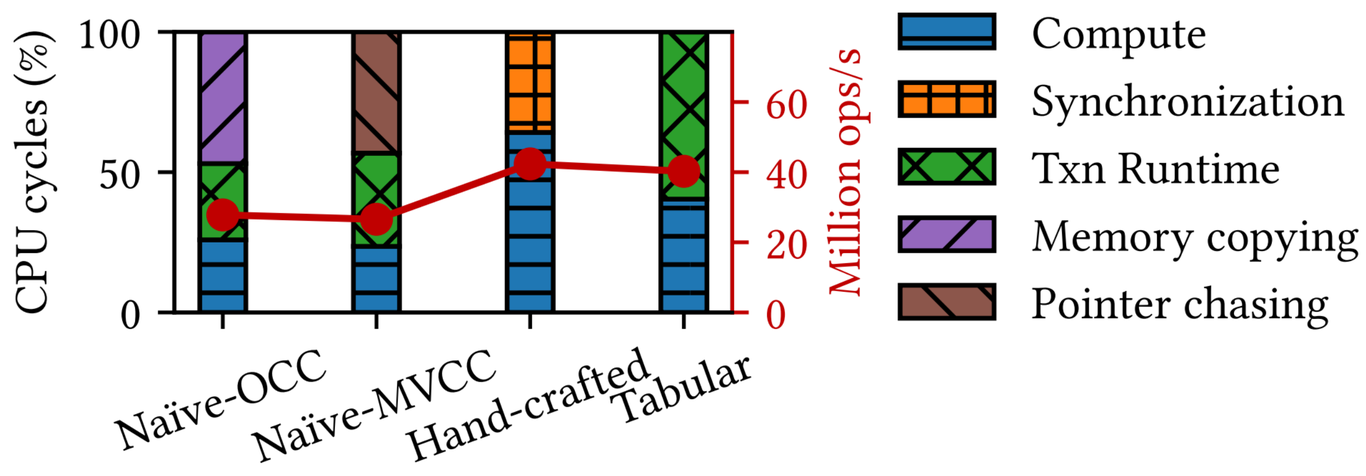
Major overheads in traditional CC: Pointer chasing and Memory Copying



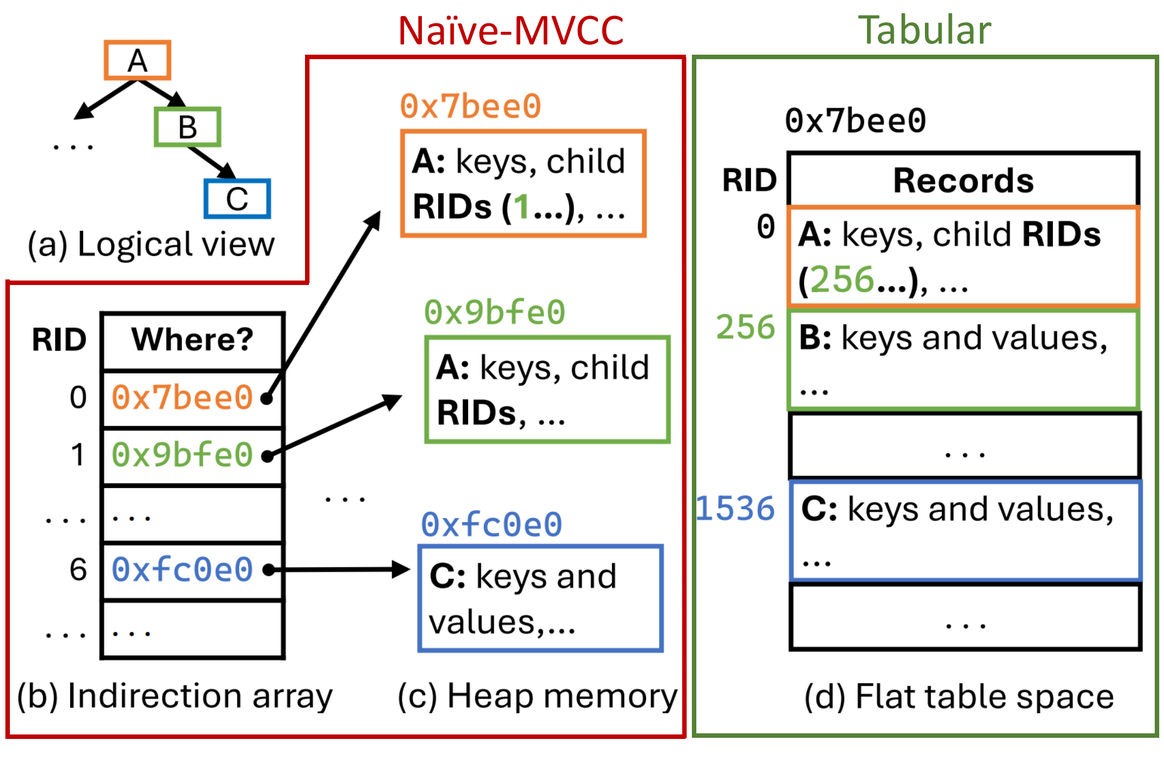
(a) Hand-crafted



(b) Naïve



Tabular #1: Direct-access 1V-OCC



Tabular #2: Zero-copy Read/Update

```
1  def Transaction::ReadCallback(table, rid, callback):
2    # Take a reference (pointer) to the record
3    Record &record = table.GetRecord(rid);
4  retry:
5    tid = record.get_consistent_tid()
6    callback(record) # Execute the callback in-place
7
8    # Verify the record did not change
9    tid_now = record.get_consistent_tid()
10   if (tid_now == tid):
11     read_set.Add(record, tid);
12   else:
13     goto retry
```

Evaluation: YCSB-like index benchmark and End-to-end TPC-C benchmark

Dual-socket server:
- 2 x 24-core Intel Xeon Gold 6252 @ 2.10GHz
- 384GB DRAM

YCSB-like index benchmark:
- 100M 8-byte keys and 8-byte values
- Indexes: B+-tree, Hash-table and ART (not shown here)

End-to-end TPC-C benchmark:
- Reach ~90% throughput of original ERMIA