Tabular: Efficiently Building Efficient Indexes

Ziyi Yan
Mohamed Farouk Drira
Tianxun Hu
Tianzheng Wang

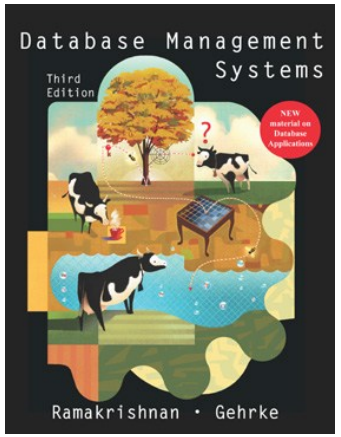
SFU SIMON FRASER UNIVERSITY

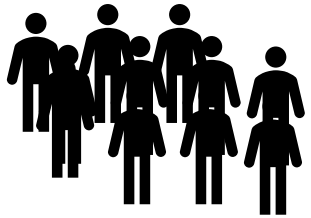
SFU Data-Intensive Systems Lab
https://github.com/sfu-dis

Building DBMSs: a Black Art

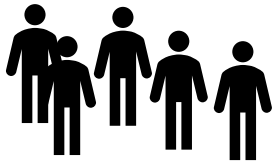
- Hand-crafted data structures
  - Hard to code up, hard to debug
  - Pretend correctness until it crashes

Know the basics





Know the hardware



+ DBMS research

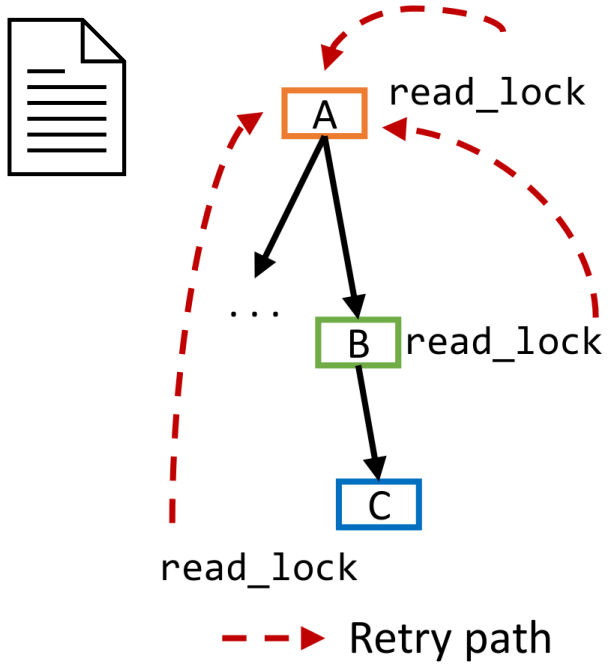
+ Neighbouring areas

Need to know multiple areas very well; rare

Let's hand-craft a concurrent B+-tree

- Memory-friendly layout + optimistic lock coupling

VLDB/SIGMOD papers




```
1 bool BTree::Insert(Key k, Value v):
2 restart:
3 epoch_enter()
4 retry = false
5   n = root
6   ver = n.read_lock(retry)
7   if retry or n != root: goto restart
8   while n is inner:
9     next = n.children[findChild(n, k)]
10    n.verify_read(ver, retry)
11    if retry: goto restart
12    ver_next = next.read_lock(retry)
13    if retry: goto restart
14    if next is inner node: ...
15    else: ...
16 ...
17 epoch_exit()
```

Epoch manager implementation...

PPoPP/SOSP papers

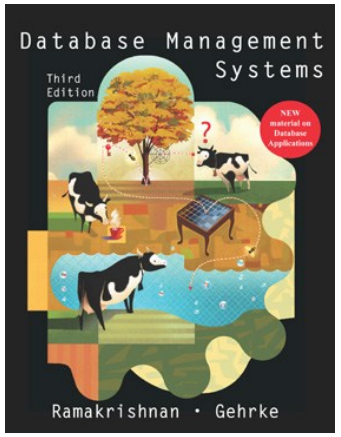
PPoPP/SIGMOD/VLDB papers

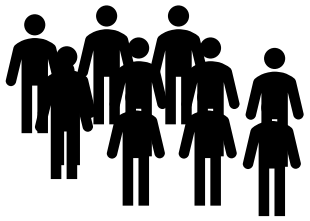
Optimistic lock implementation...

Complex stuff!

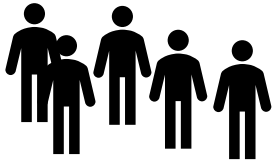
Building DBMSs: a Black Art

Know the basics





Know the hardware



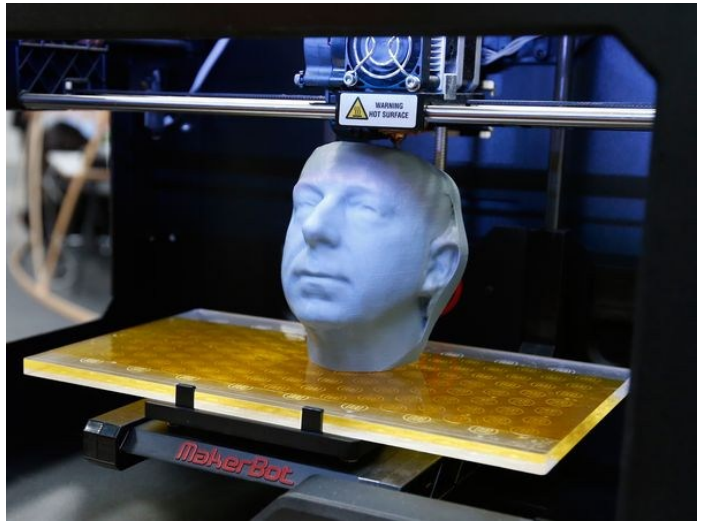
But many more know how to *use* a DBMS!
+ DBMS internal basics

+ DBMS research

+ Neighbouring areas

- Low-level programming
  - Hard to code up, hard to debug
  - Pretend correctness until it crashes

DB Transactions vs. Manual Parallel Programming



- Transactions
- Tables and records
- Transparent write-ahead-logging
- Buffered tables

- Locks/lock-free algorithms
- Explicit memory management
- Manual persistence, I/O
- Manual caching solutions

Can we do the same for parallel programming?

* To Lock, Swap or Elide: On the Interplay of Hardware Transactional Memory and Lock-free Indexing, VLDB 2015

Modelling Data Structures as Relational Tables

```
struct BTreeNode {
  int n_keys;
  bool is_leaf;
  int lock;
  BTreeNode *right_child;
  KVPair kvs[16];
};
```

A "B-tree node table"

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

- Struct/class → Table
  - Member variables → Table columns
  - Defined by a schema, just like a DB app
- Instances of struct/class → Table records
- Index operations → Transactions
  - An OLTP engine takes care of concurrency and persistence

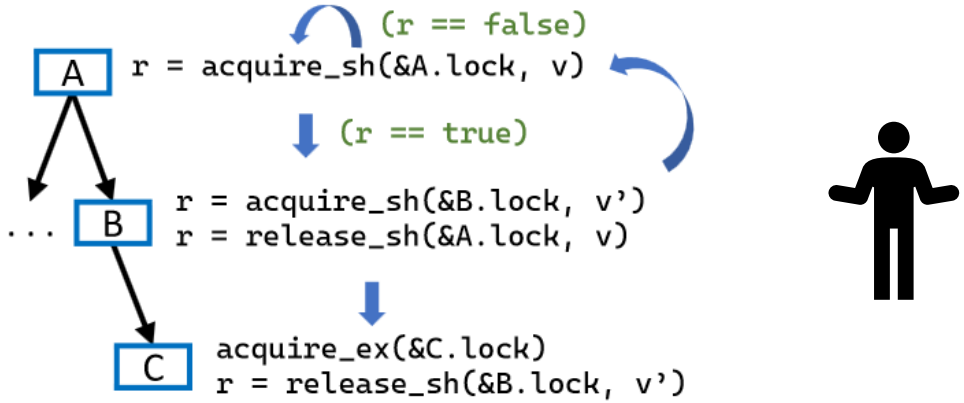
Modelling Data Structures as Relational Tables

Before

Allocation:

```
struct *mynode = (Node *)malloc(sizeof(Node));
```

Concurrency:



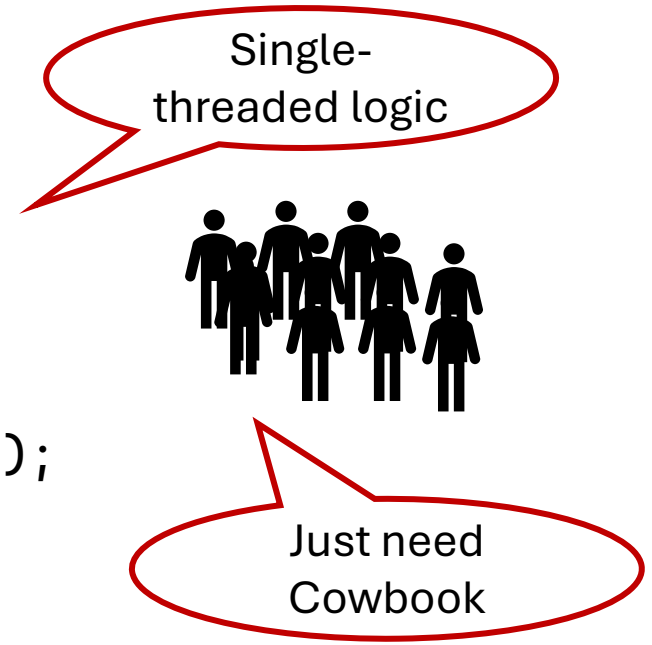
After

Allocation:

```
struct *mynode = table.insert(…)
```

Concurrency:

```
Transaction {
  table.read(A);
  table.read(B);
  table.read(C);
  …
  table.update(C);
}
```

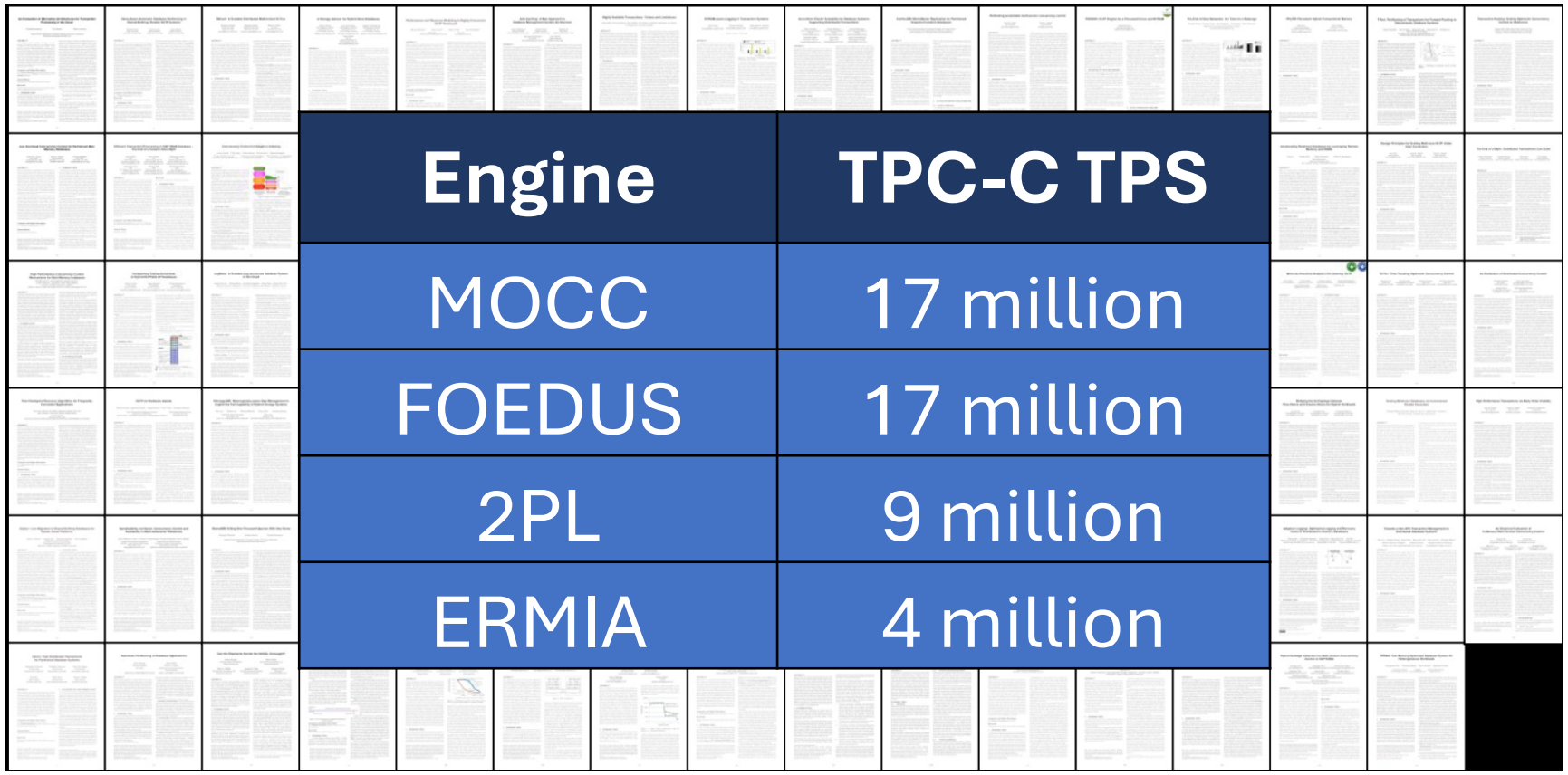


"Wasn't this tried and failed?"

Past Failed Aspirations

- Object-on-DB / Object-oriented DBMSs
  - High overhead over full RDBMSs
- Hardware transactional memory (HTM)
  - No persistence
  - Hard to use – “weird” aborts
- Software transactional memory (STM)
  - High overhead
  - No persistence
  - “Research toy”

It's different this time: in-memory OLTP since 2010+



Engine	TPC-C TPS
MOCC	17 million
FOEDUS	17 million
2PL	9 million
ERMIA	4 million

On a 16-socket, 288-core HPE server

"Who need these?"

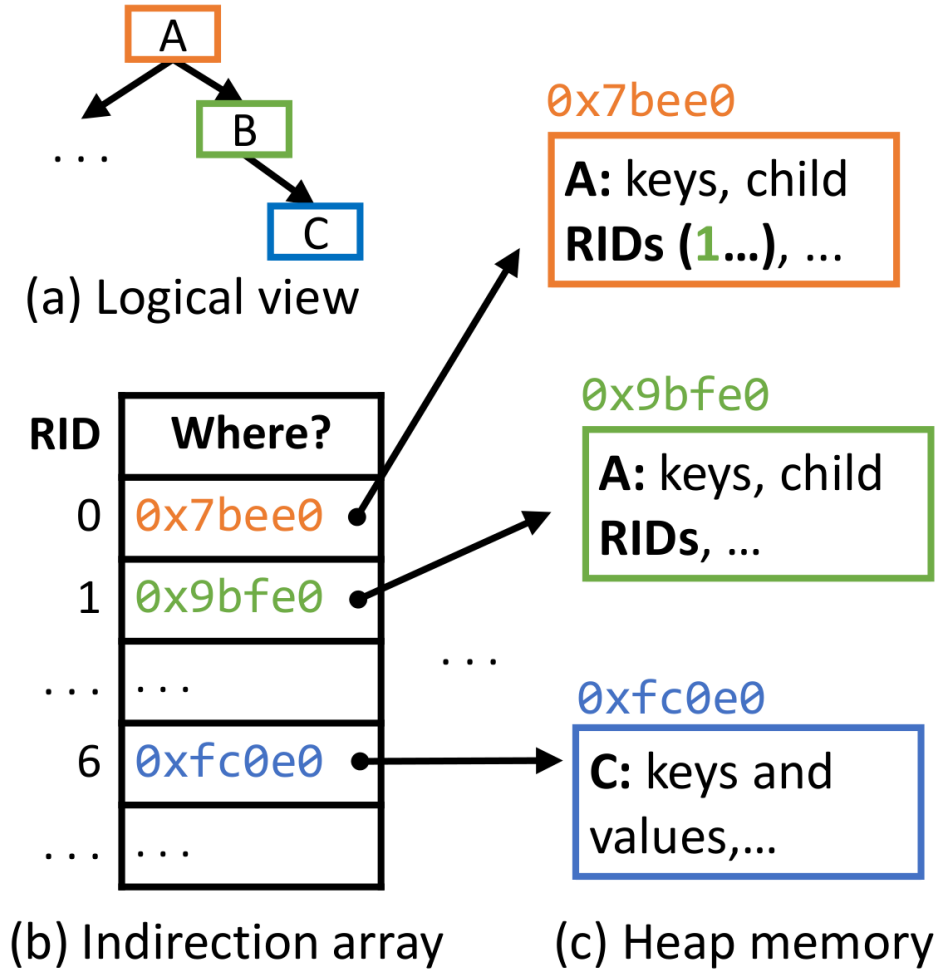
"Very few!"

Much more headroom for objects-over-DB

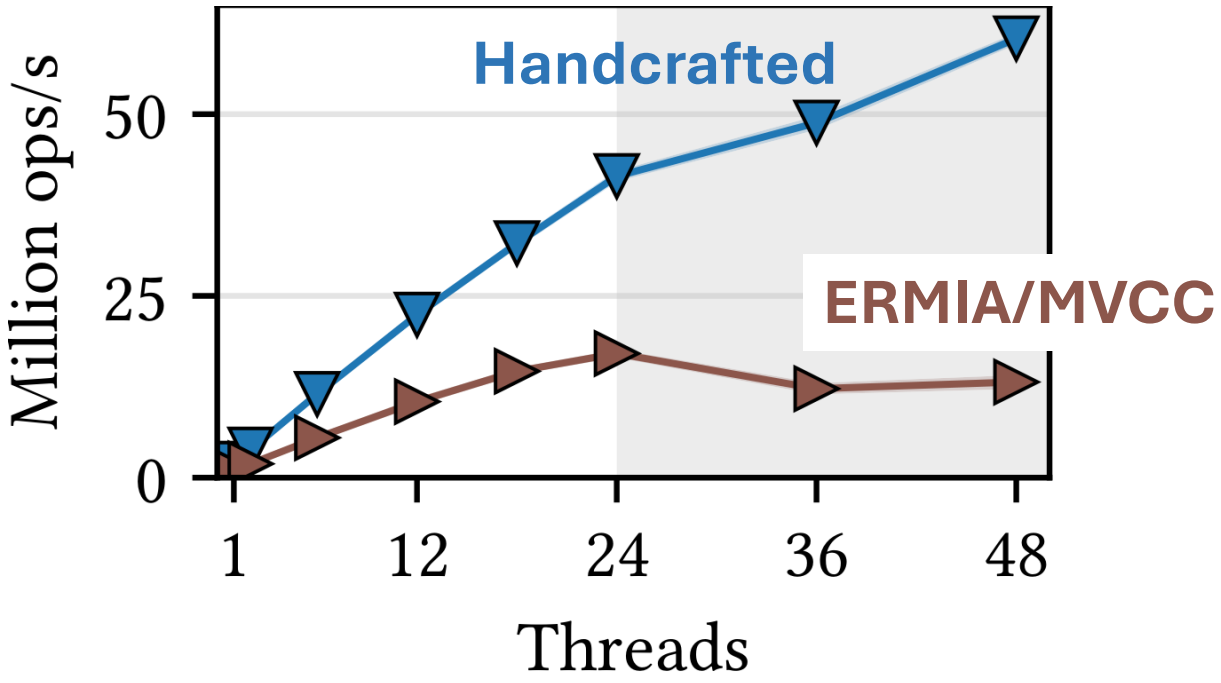
*What Are We Doing With Our Lives?: Nobody Cares About Our Concurrency Control Research, SIGMOD 2017 Keynote

Tabular: Efficiently Building Efficient Indexes

First try: B+-tree over ERMIA*



Pointer chasing dominates: 80% slower

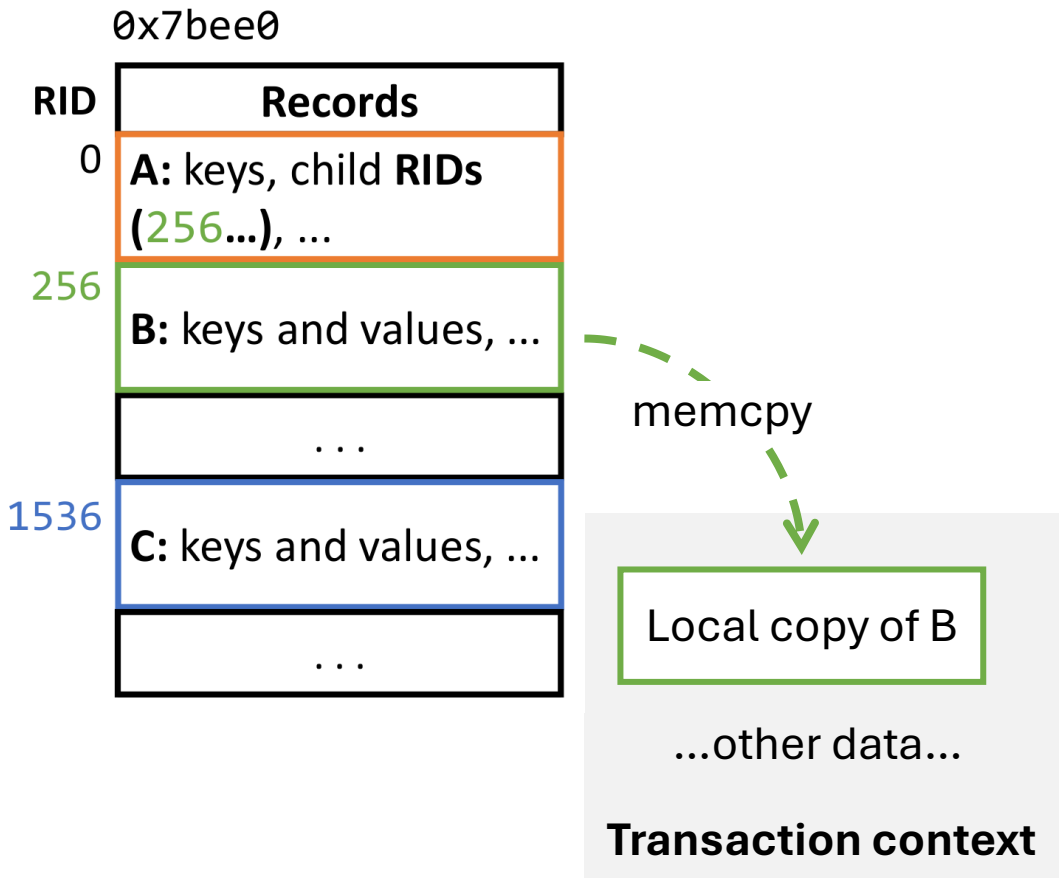


Multi-versioning considered harmful

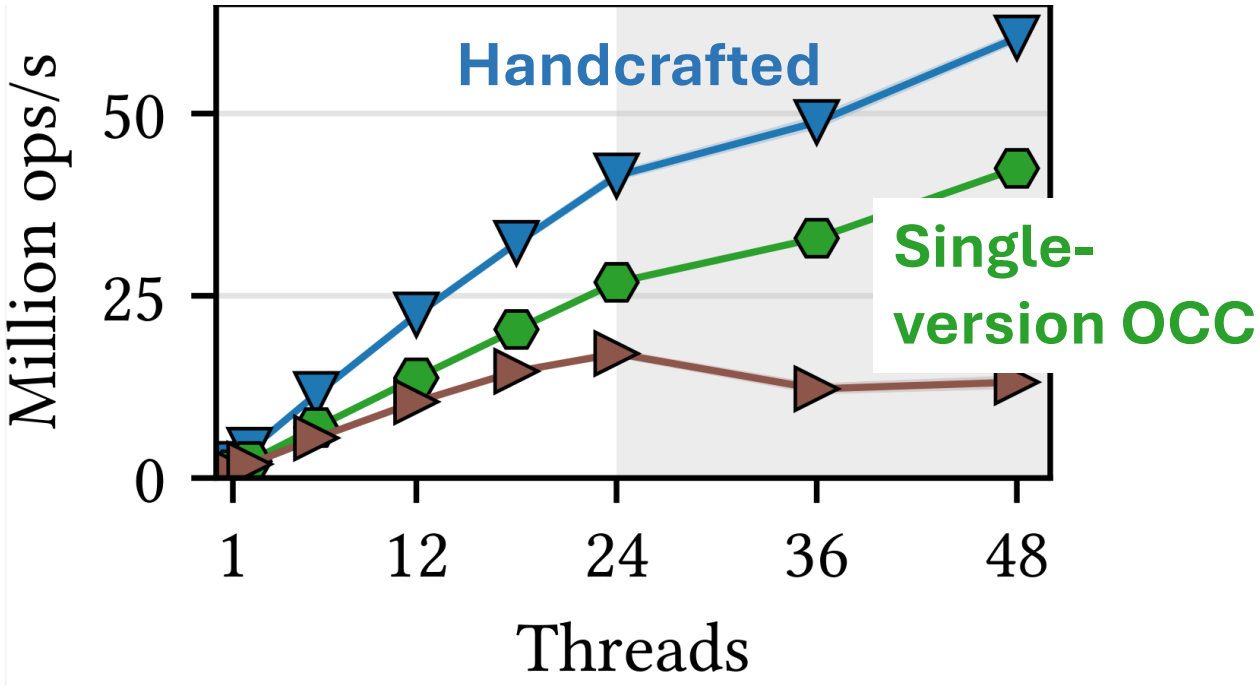
* ERMIA: Fast Memory-Optimized Database System for Heterogeneous Workloads, SIGMOD 2016

One more time: B+-tree over Single-Versioned OCC

Just plain flat space



~50% cycles on memcpy, still a ~30% gap:



Memory copy considered harmful

Tabular: Single-Version + (Near) Zero Memory Copy

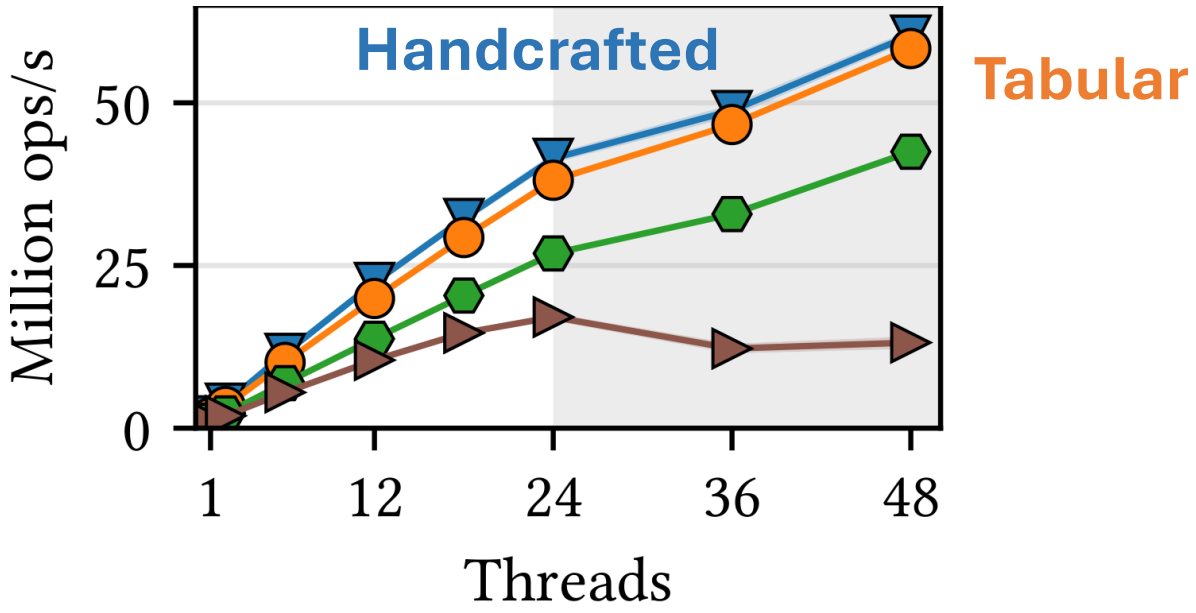
- Flat table space to avoid versioning/pointer chasing overhead
- An old trick – ship the function, not data – to remove unnecessary memcpy

Before:

```
. . .
Node n = table.read(rid);
k, v = search(&n)…
. . .
```

After:

```
. . .
k, v = table.read(rid, search);
. . .
```

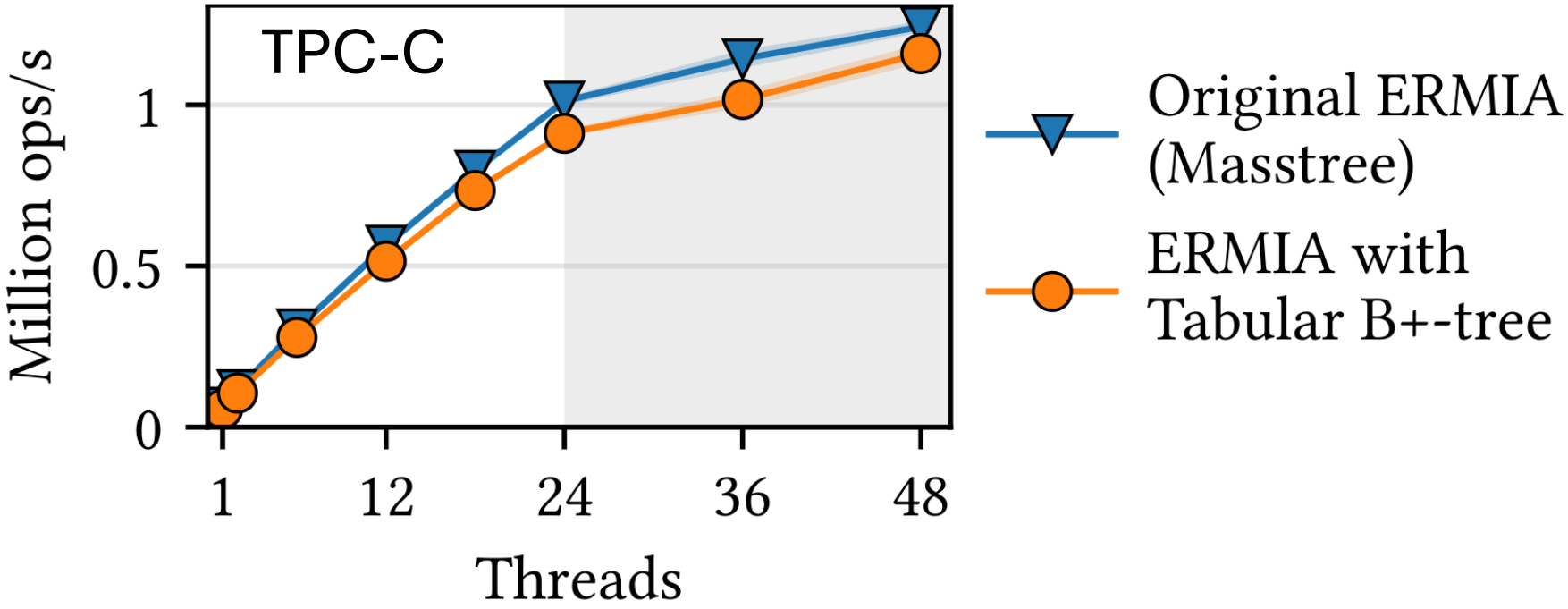


Applies to writes, too

Only need simple tweaks in OCC protocol to make this work

As a Drop-in Replacement

- Masstree in ERMIA ➔ Tabular B+-tree



Over 95% of hand-crafted

Summary

- Building a DBMS is hard
  - Have to know much more beyond DBMS itself: Parallel programming, hardware…
- Tabular: Objects-on-DB via modern OLTP techniques
  - Single-versioning + OCC + zero-copy transactions
  - Database concurrency control for data structures
  - Various benefits – easier programming, debugging, migration, transparent persistence…

More in our paper and code repo:

https://github.com/sfu-dis/tabular

Thank you!