Topics in Database Systems: Main/in-memory Database Systems

CS848 Spring 2016

David Toman

Wednesday 3:30-6:20pm in DC 2568

cs.uwaterloo.ca/~david/cs848/

Proloferation of NEW DB(-like) Implementations

Quick sample:

. . . and dozens of others

In contrast to. . .

. . . before Y∼2000 it was pretty much divided between

the big four (ORALE, IBM/DB2, Sybase, and MS Server)

and (later, with the advent of the WEB) Postgress, MySQL, etc.

Proloferation of NEW DB(-like) Implementations

Quick sample:

. . . and dozens of others

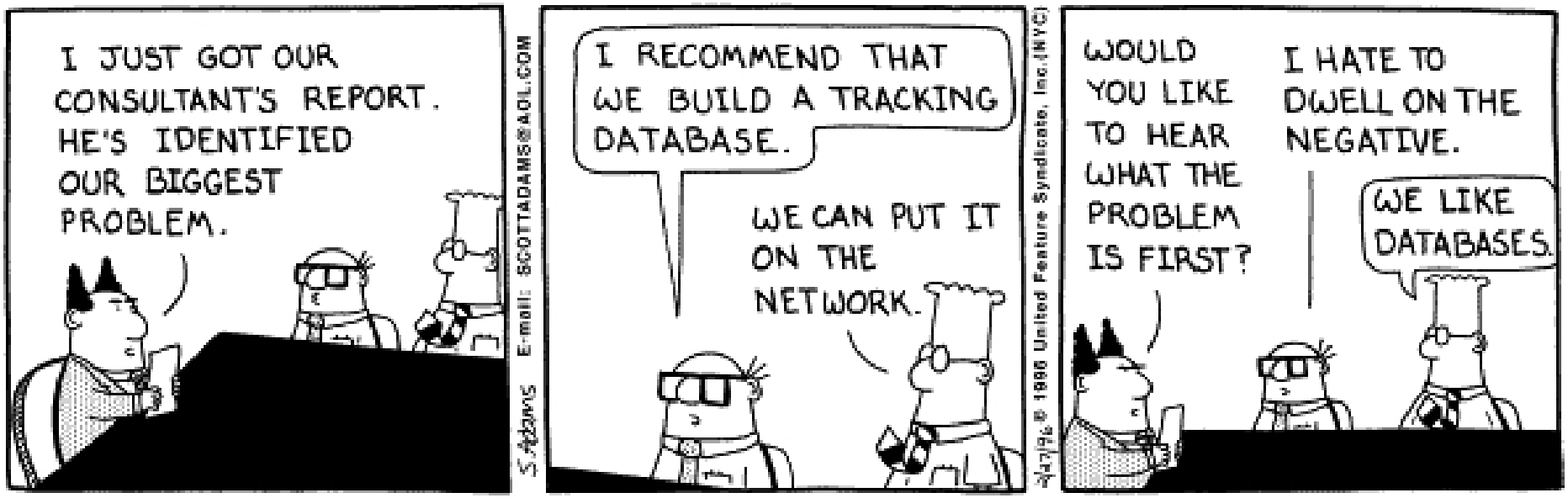
In contrast to. . .

. . . before Y∼2000 it was pretty much divided between

the big four (ORALE, IBM/DB2, Sybase, and MS Server)

and (later, with the advent of the WEB) Postgress, MySQL, etc.

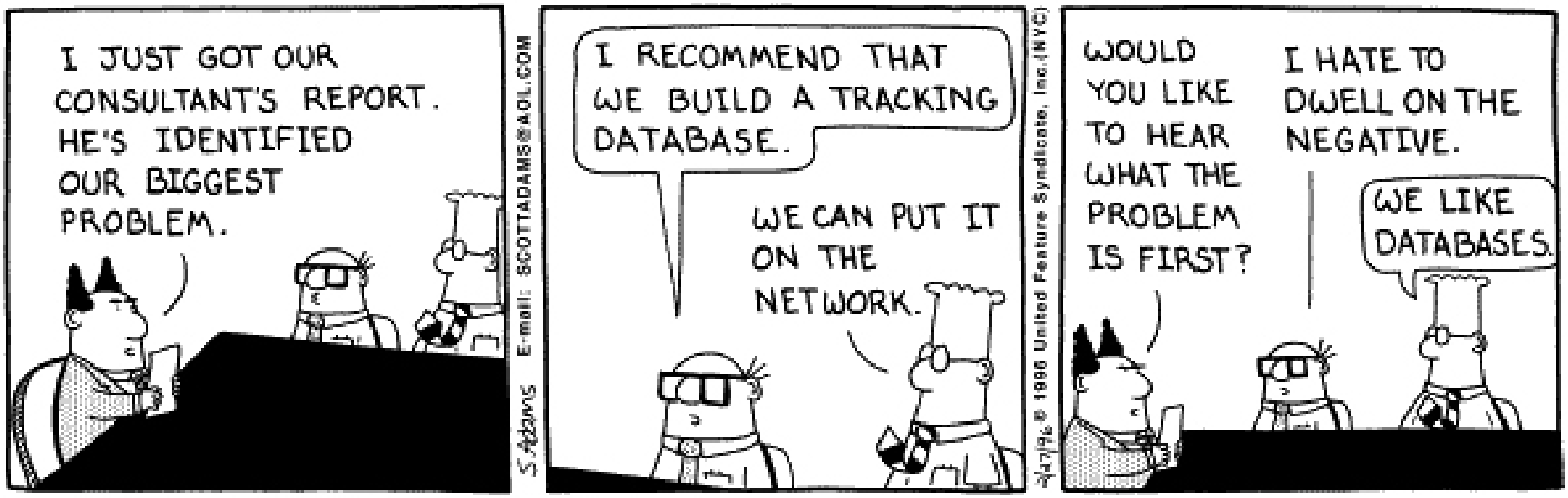
Why so many? And why Main/In-Memory? (M/IMDB)



New Circumstances

1. cheap and abundant hardware (incl. Main Memory)
2. changes in applications/workloads
3. cost (we won't focus on this though)

Why so many? And why Main/In-Memory? (M/IMDB)



New Circumstances

1. cheap and abundant hardware (incl. Main Memory)
2. changes in applications/workloads
3. cost (we won't focus on this though)

Topics of Interest

1. What are the main differences between managing Main/In-memory data v.s. data in external storage?
   - impact on query/update processing
     how many *instructions* does it take to answer simple queries?
   - what happens to ACID (and can we afford durability at all)?
2. What is the impact on *programming interface* to the I/MMDB?
   - declarative (SQL-like) vs. procedural (C++-like)
   - query optimization?
3. What is the impact of multi-core/CPU hardware
   - data partitioning and query compilation/allocation
   - communication/synchronization between parallel operations
     dependency on architecture (Multicore, NUMA, Shared-nothing)?
4. UDTs (user-defined topics)

Topics of Interest

1. What are the main differences between managing Main/In-memory data v.s. data in external storage?
   - impact on query/update processing
     how many *instructions* does it take to answer simple queries?
   - what happens to ACID (and can we afford durability at all)?
2. What is the impact on *programming interface* to the I/MMDB?
   - declarative (SQL-like) vs. procedural (C++-like)
   - query optimization?
3. What is the impact of multi-core/CPU hardware
   - data partitioning and query compilation/allocation
   - communication/synchronization between parallel operations
     dependency on architecture (Multicore, NUMA, Shared-nothing)?
4. UDTs (user-defined topics)

Outline&Organization

- Organization:
  - $\Rightarrow$ Lectures (2-3),
  - $\Rightarrow$ Presentations of papers (reading list), and
  - $\Rightarrow$ Projects
- First meeting: **Wed May 4, 2016 at 3:30 in DC 2568**
- Prerequisites:
  - $\Rightarrow$ *Intro to Databases* (CS348-like), and
  - $\Rightarrow$ standard programming skills
    (although this is not necessarily an implementation class)
- Class web site: `cs.uwaterloo.ca/~david/cs848/`
  reading list, schedule of classes/presentations, policies, etc.

Organization (ii)

Week 1:	Organization, Introduction to DB implementation: Classical Approaches
Weeks 2-3:	More on I/MMDB Assignment of in class presentations.
Weeks 4-7:	In class Presentations
Week 8:	In class Presentatios and Summary of Presentations
Weeks 9-11:	Project Presentations
Week 12:	Summary and Wrap-up

Reading List

1. NoSQL
   - NoSQL Databases (warning: this is rather big document)
   - Scalable SQL and NoSQL Data Stores
   - History Repeats Itself: Sensible and NonsenSQL Aspects of the NoSQL Hoopla
2. Systems
   - Special Issue on Main-Memory Database Systems IEEE June 2013
3. Architecture, Data models, and Algorithmic Issues:
   - The End of an Architectural Era
   - Anti-Caching: A New Approach to Database Management System Architecture
   - MonetDBX100: Hyper-Pipelining Query Execution
   - Improved Query Performance with Variant Indexes bitmap indices
     (not really Main Memory, but relevant)
   - Locality-Sensitive Operators for Parallel Main-Memory Database Clusters
4. Column Stores:
   - C-Store: A Column-oriented DBMS
   - Column-Stores vs. Row-Stores: How Different Are They Really
   - Integrating compression and execution in column-oriented database systems

Reading List (cont.)

5. Concurrency at al.:
   - High-Performance Concurrency Control Mechanisms for Main-Memory Databases
   - HyPer: A Hybrid OLTP and OLAP Main Memory Database System Based on Virtual Memory Snapshots
   - Rethinking Main Memory OLTP Recovery
   - An Integrated Approach to Recovery and High Availability in an Updatable, Distributed Data Warehouse (not really MM)
6. Partitioning of Data:
   - Skew-Aware Automatic Database Partitioning in Shared-Nothing, Parallel OLTP Systems
   - Bigtable: A Distributed Storage System for Structured Data (not really MM)
   - Cassandra - A Decentralized Structured Storage System (not really MM)
   - Life beyond Distributed Transactions (really about partitioning)
   - Dynamo-amazons highly available key-value store

In-class Presentations

2-3 presentations a week

$\Rightarrow$ 40-50 minutes presenttion

$\Rightarrow$ 20+ minutes discussion

1. Presenter:
   - Summary of paper/topic: Explanation of main ideas (in your own words), Examples
   - Critique: Is this a real problem? Is the solution comprehensive? exceptions?
   - Competition: What do other researchers/products do?
   - Future: How can the approach be extended/new ideas
2. Everyone else:
   - submit 2-3 questions on the paper(s) to be presented in advance
     $\Rightarrow$ **by Tuesday midnight before class to `david@uwatelroo.ca`**
   - participate in discussion

Assessment

1. class participation, including submitting questions (20%)
2. in class presentation of a topic/paper from the reading list (35%)
3. project presentation and delivery (45%)

NB: I'll discuss projects in next class—
please focus on chosing your *in-class* presentation first!

Database Implementation

(overview of standard techniques)

Requirements (user point of view)

Goal of a DBMS

Execute user queries/Updates (as fast as possible)

(typical) Requirements:

1. Stores all of your Data (scalability)
2. Physical Data Independence (SQL vs. B-trees et al.)
3. Durability (the idea of a *transaction*)
4. Isolation (sharing/concurrency)

$\Rightarrow$ do we need all of the above all the time?

Requirements (user point of view)

Goal of a DBMS

Execute user queries/Updates (as fast as possible)

(typical) Requirements:

1. Stores all of your Data (scalability)
2. Physical Data Independence (SQL vs. B-trees et al.)
3. Durability (the idea of a *transaction*)
4. Isolation (sharing/concurrency)

$\Rightarrow$ do we need all of the above all the time?

Requirements (user point of view)

Goal of a DBMS

Execute user queries/Updates (as fast as possible)

(typical) Requirements:

1. Stores all of your Data (scalability)
2. Physical Data Independence (SQL vs. B-trees et al.)
3. Durability (the idea of a *transaction*)
4. Isolation (sharing/concurrency)

$\Rightarrow$ do we need all of the above all the time?

Standard Architecture: Client-Server "System"

Query/Update Compiler

$\Rightarrow$ compiles a *logical expression* to a *plan*

Query/Update Execution Engine:

$\Rightarrow$ executes a *prepared plan*

1. Query processor (access paths)
2. Transaction Manager
3. Recovery Manager
4. Buffer Pool

Where does the Time go? (a case study)

- SHORE (Scalable Heterogeneous Object Repository, Wisconsin '90s)
  $\Rightarrow$ the whole database is preloaded in main memory
- TPC-C (OLTP) benchmark: "new order" and "payment" transactions
  $\Rightarrow$ 50/50 mix of the transactions in experiments
- Experiments show performance gain by removing/simplifying:
  1. B-Tree keys (no prefix compression)
  2. no logging (no durability)
  3. no locks (no concurrency)
  4. no latches (no transactions: begin/commit/. . .)
  5. no buffer manager (remember DB preloaded!)

Where does the Time go? (a case study)

- SHORE (Scalable Heterogeneous Object Repository, Wisconsin '90s)
  $\Rightarrow$ the whole database is preloaded in main memory
- TPC-C (OLTP) benchmark: "new order" and "payment" transactions

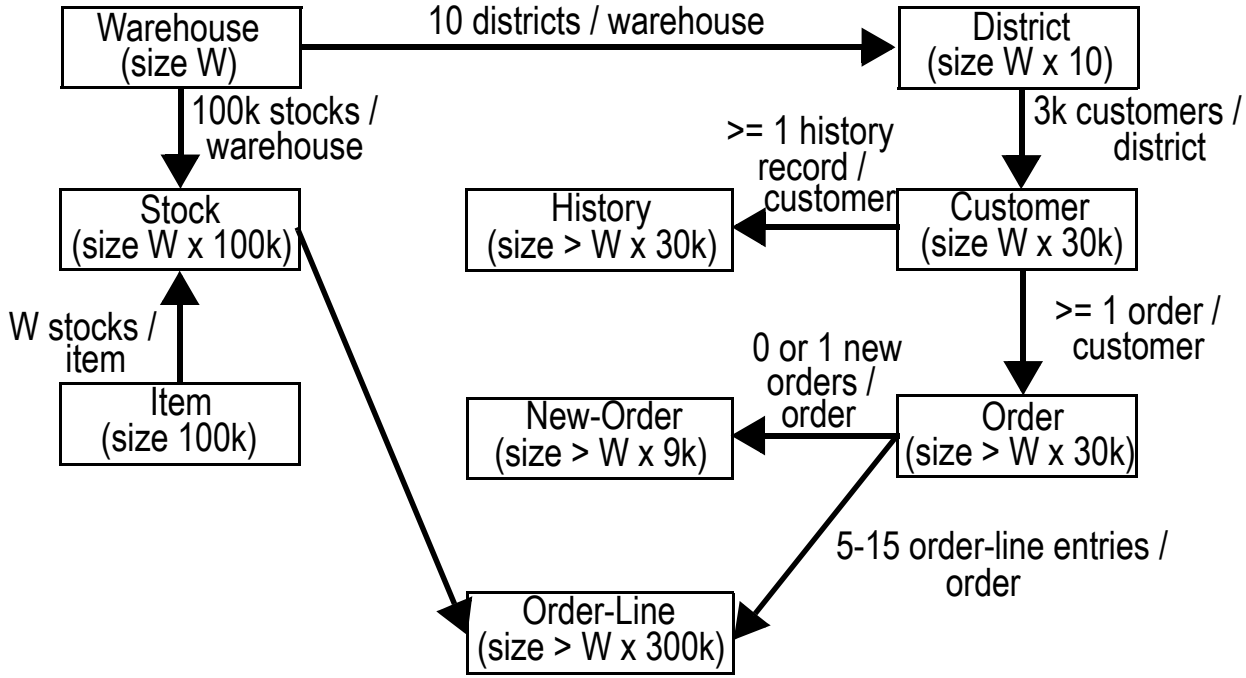


Figure 3. TPC-C Schema.

Where does the Time go? (a case study)

- SHORE (Scalable Heterogeneous Object Repository, Wisconsin '90s)
  $\Rightarrow$ the whole database is preloaded in main memory
- TPC-C (OLTP) benchmark: "new order" and "payment" transactions
  $\Rightarrow$ 50/50 mix of the transactions in experiments
- Experiments show performance gain by removing/simplifying:
  1. B-Tree keys (no prefix compression)
  2. no logging (no durability)
  3. no locks (no concurrency)
  4. no latches (no transactions: begin/commit/...)
  5. no buffer manager (remember DB preloaded!)

Where does the Time go? (setup)

Assumptions:

1. all data preloaded into main memory
2. transactions compiled and linked against SHORE
3. 50-50 mix
4. 40k transaction runs

New Order	Payment
begin	begin
for loop(10)	Btree lookup(D), pin
.....Btree lookup(I), pin	Btree lookup (W), pin
Btree lookup(D), pin	Btree lookup (C), pin
Btree lookup (W), pin	update rec (C)
Btree lookup (C), pin	update rec (D)
update rec (D)	update rec (W)
for loop (10)	create rec (H)
.....Btree lookup(S), pin	commit
.....update rec (S)	
.....create rec (O-L)	
.....insert Btree (O-L)	
create rec (O)	
insert Btree (O)	
create rec (N-O)	
insert Btree (N-O)	
insert Btree 2ndary(N-O)	
commit	

Figure 4. Calls to Shore's methods for New Order and Payment transactions.

Where does the Time go?

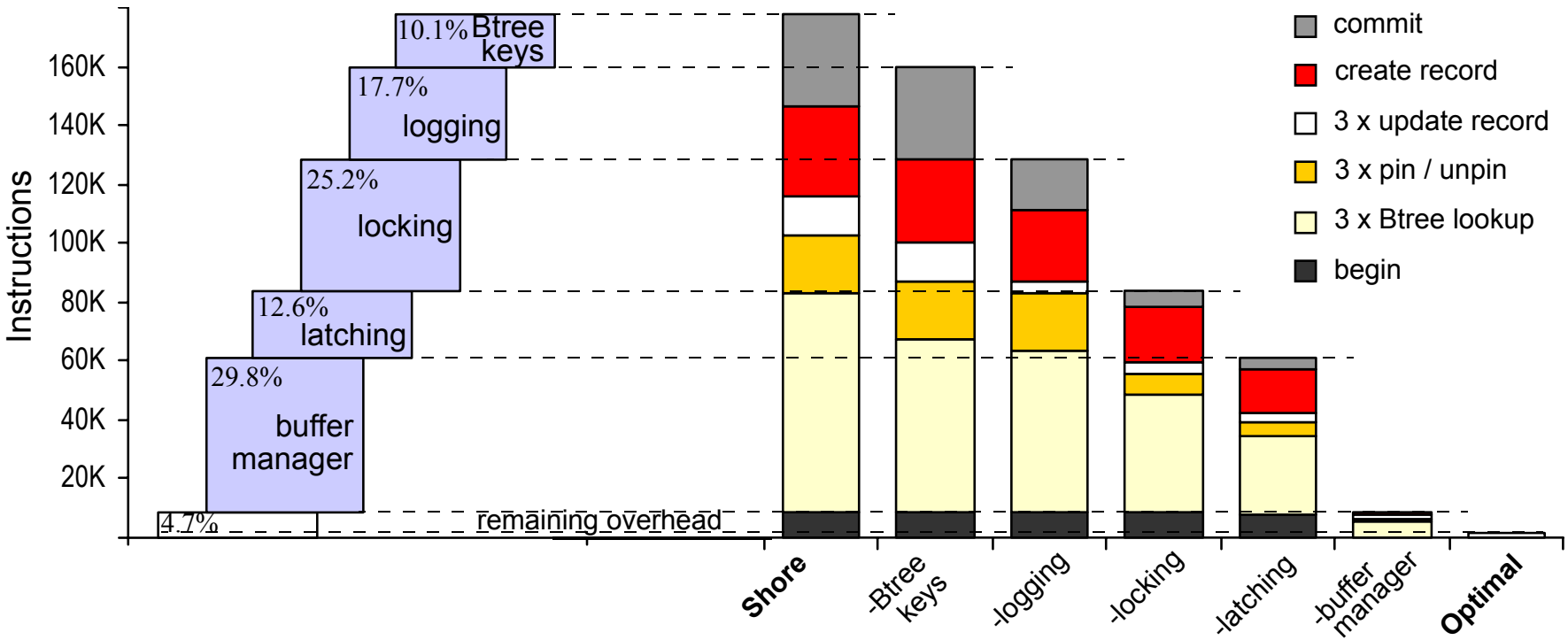


Figure 5. Detailed instruction count breakdown for Payment transaction.

Where does the Time go?

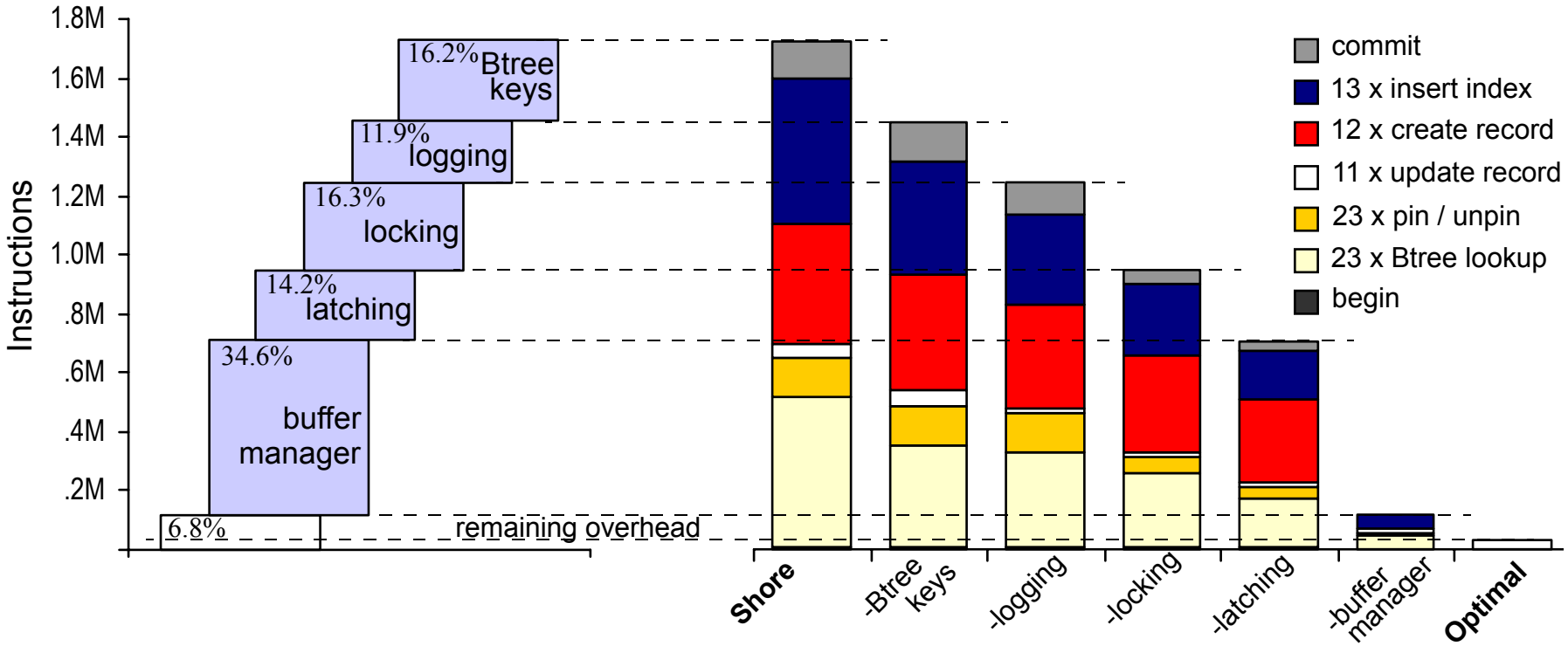


Figure 6. Detailed instruction count breakdown for New Order transaction.

Where does the Time go?

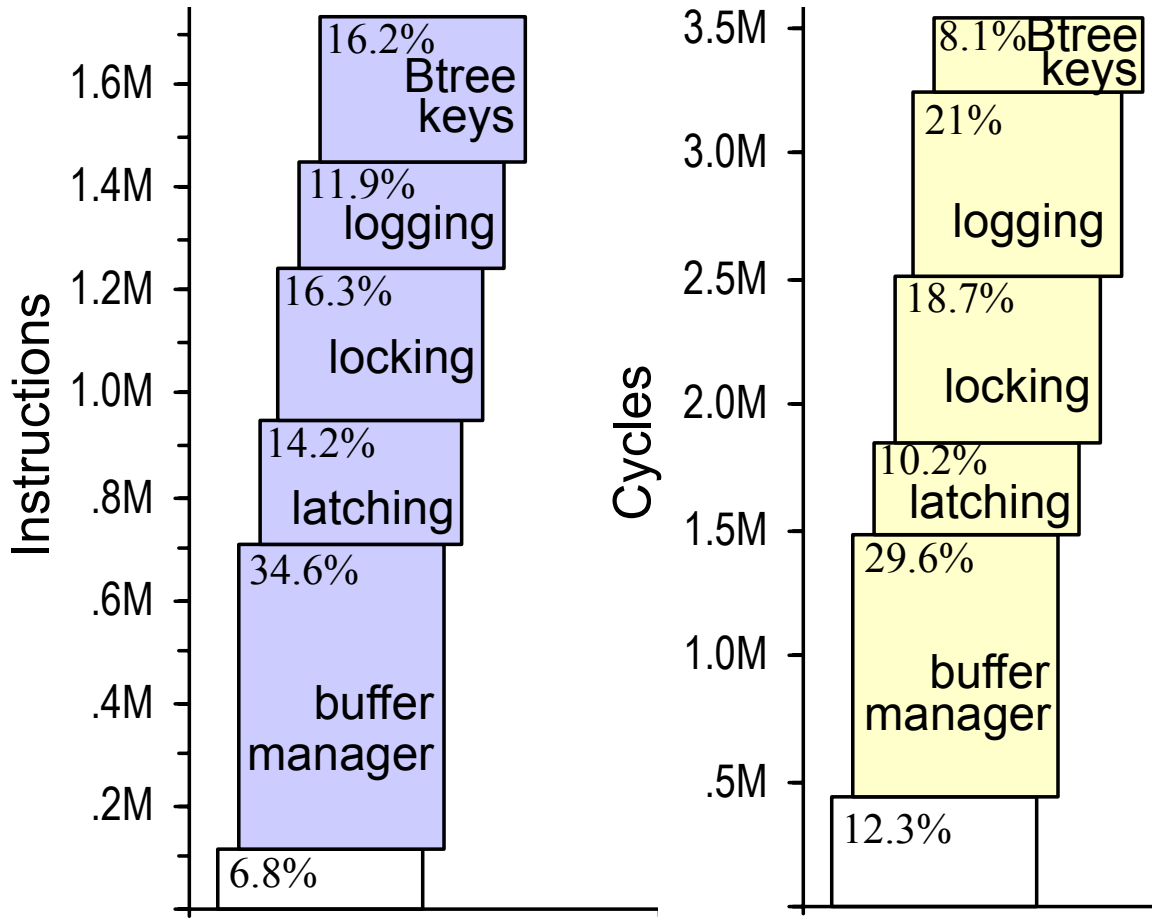


Figure 8. Instructions (left) vs. Cycles (right) for New Order.

Where does the Time go? (conclusions)

Having a giant buffer cache to fit the whole dataset

doesn't seem to solve all problems (90+% OVERHEAD!)

However...

...the savings in experiments at cost of functionality

$\Rightarrow$ can MMDBs be engineered to mittigate the overhead

without sacrifising functionality?

- Single threading vs. multicore
- Availability (replication) vs. logging
- Variations on isolation
- Cache-conscious data structures

Where does the Time go? (conclusions)

Having a giant buffer cache to fit the whole dataset

doesn't seem to solve all problems (90+% OVERHEAD!)

However...

...the savings in experiments at cost of functionality

$\Rightarrow$ can MMDBs be engineered to mittigate the overhead without sacrifising functionality?

- Single threading vs. multicore
- Availability (replication) vs. logging
- Variations on isolation
- Cache-conscious data structures

Where does the Time go? (conclusions)

Having a giant buffer cache to fit the whole dataset

doesn't seem to solve all problems (90+% OVERHEAD!)

However...

...the savings in experiments at cost of functionality

$\Rightarrow$ can MMDBs be engineered to mittigate the overhead without sacrifising functionality?

- Single threading vs. multicore
- Availability (replication) vs. logging
- Variations on isolation
- Cache-conscious data structures

Take Home

Lots of open issues:

1. DB engine vs. Compilation aproaches
2. Main memory data organization (multilevel memory)
3. Taking advantage of parallelism (many levels)
4. . . .

Assignment (beyond thinking about the above issues)

Look through the *reading list* and pick topic/paper for your in-class presentation (and email me at david@uwaterloo.ca)

$\Rightarrow$ **DEADLINE: 05/16** (Monday in two weeks)

$\Rightarrow$ I'll have to cluster presentations on similar topics together

Take Home

Lots of open issues:

1. DB engine vs. Compilation aproaches
2. Main memory data organization (multilevel memory)
3. Taking advantage of parallelism (many levels)
4. . . .

Assignment (beyond thinking about the above issues)

Look through the *reading list* and pick topic/paper for your in-class presentation (and email me at david@uwaterloo.ca)

$\Rightarrow$ **DEADLINE: 05/16** (Monday in two weeks)

$\Rightarrow$ I'll have to cluster presentations on similar topics together

Take Home

Lots of open issues:

1. DB engine vs. Compilation aproaches
2. Main memory data organization (multilevel memory)
3. Taking advantage of parallelism (many levels)
4. ...

Assignment (beyond thinking about the above issues)

Look through the *reading list* and pick topic/paper for your in-class presentation (and email me at `david@uwaterloo.ca`)

$\Rightarrow$ **DEADLINE: 05/16** (Monday in two weeks)

$\Rightarrow$ I'll have to cluster presentations on similar topics together