Lecture #11

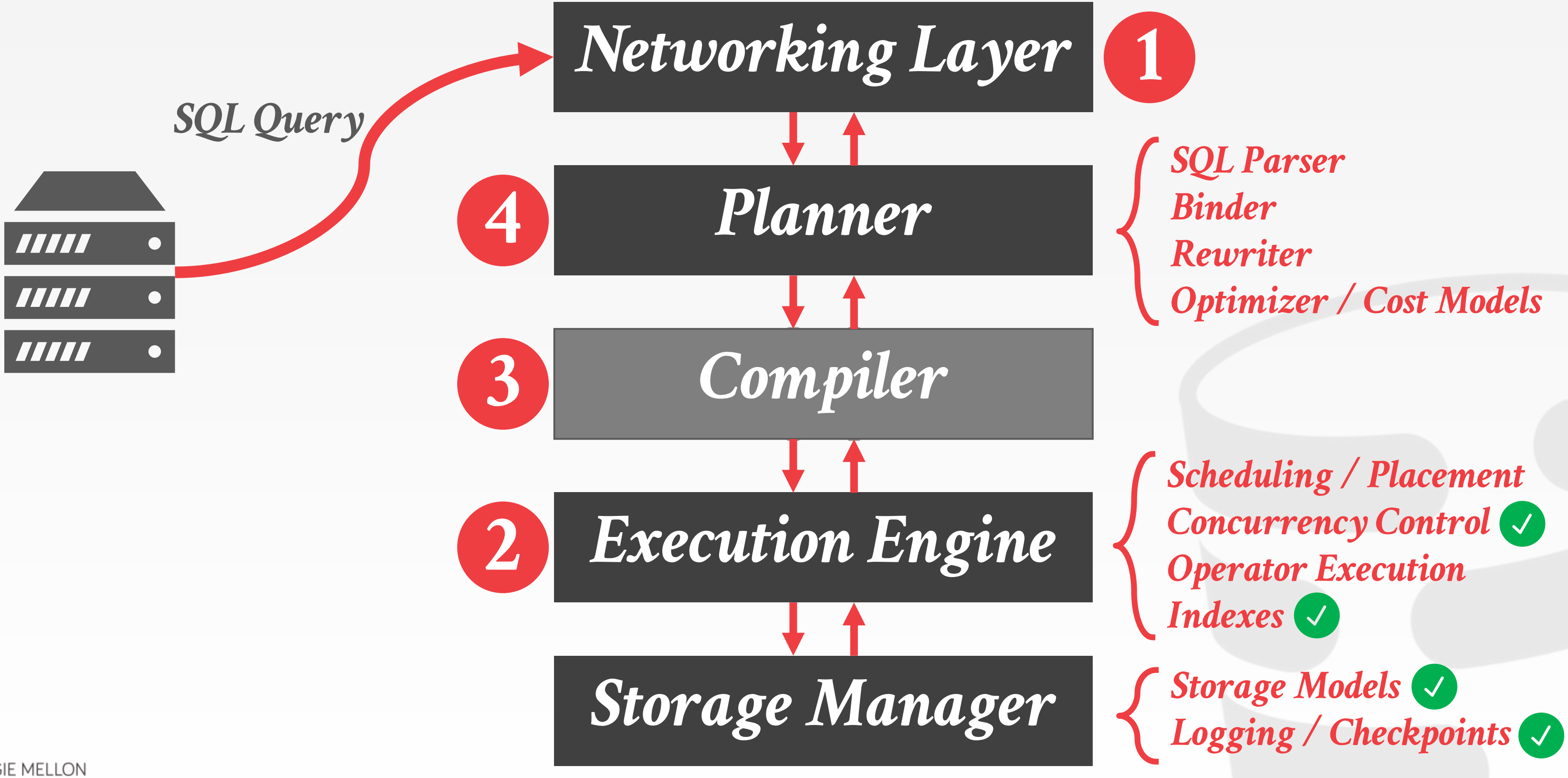
Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Networking

@Andy_Pavlo // 15-721 // Spring 2020

ARCHITECTURE OVERVIEW



TODAY'S AGENDA

Database Access APIs

Database Network Protocols

Replication Protocols

Kernel Bypass Methods

Project #2

DATABASE ACCESS

All the demos in the class have been through a terminal client.

→ SQL queries are written by hand.
→ Results are printed to the terminal.

Real programs access a database through an API:

→ Direct Access (DBMS-specific)
→ Open Database Connectivity (ODBC)
→ Java Database Connectivity (JDBC)

OPEN DATABASE CONNECTIVITY

Standard API for accessing a DBMS. Designed to be independent of the DBMS and OS.

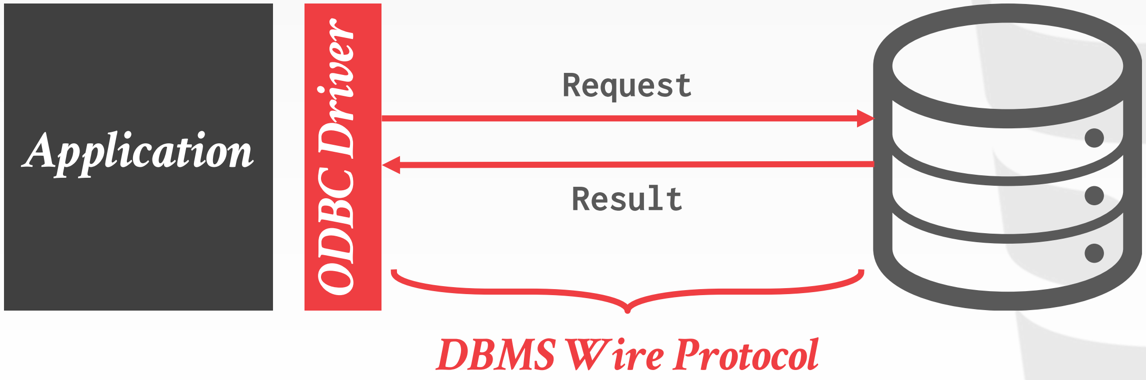
Originally developed in the early 1990s by Microsoft and Simba Technologies.

Every major relational DBMS now has an ODBC implementation.

OPEN DATABASE CONNECTIVITY

ODBC is based on the "device driver" model.

The **driver** encapsulates the logic needed to convert a standard set of commands into the DBMS-specific calls.



JAVA DATABASE CONNECTIVITY

Developed by Sun Microsystems in 1997 to provide a standard API for connecting a Java program with a DBMS.

JDBC can be considered a version of ODBC for the programming language Java instead of C.

JAVA DATABASE CONNECTIVITY

Approach #1: JDBC-ODBC Bridge
→ Convert JDBC method calls into ODBC function calls.

Approach #2: Native-API Driver
→ Convert JDBC method calls into native calls of the target DBMS API.

Approach #3: Network-Protocol Driver
→ Driver connects to a middleware that converts JDBC calls into a vendor-specific DBMS protocol.

Approach #4: Database-Protocol Driver
→ Pure Java implementation that converts JDBC calls directly into a vendor-specific DBMS protocol.

DATABASE NETWORKING PROTOCOLS

All major DBMSs implement their own proprietary wire protocol over TCP/IP.

A typical client/server interaction:

- → Client connects to DBMS and begins authentication process. There may be an SSL handshake.
- → Client then sends a query.
- → DBMS executes the query, then serializes the results and sends it back to the client.

EXISTING PROTOCOLS

Most newer systems implement one of the open-source DBMS wire protocols. This allows them to reuse the client drivers without having to develop and support them.

Just because on DBMS "speaks" another DBMS's wire protocol does not mean that it is compatible.

→ Need to also support catalogs, SQL dialect, and other functionality.

EXISTING PROTOCOLS

MySQL

memSQL

Clustrix

actorDB

TiDB

DOLT

CLEARDB

Bedrock

Amazon Aurora

PostgreSQL

amazon REDSHIFT

Greenplum

HyPer

VERTICA

Cockroach LABS

YugaByte

Peloton

CrateDB

Amazon Aurora

UMBRA

HIVE

APACHE Spark

PROTOCOL DESIGN SPACE

Row vs. Column Layout

Compression

Data Serialization

String Handling

DON'T HOLD MY DATA HOSTAGE: A CASE FOR CLIENT PROTOCOL REDESIGN
VLDB 2017

ROW VS. COLUMN LAYOUT

ODBC/JDBC are inherently row-oriented APIs.

→ Server packages tuples into messages one tuple at a time.
→ Client must deserialize data one tuple at a time.

But modern data analysis software operates on matrices and columns.

One potential solution is to send data in vectors.

→ Batch of rows organized in a column-oriented layout.

COMPRESSION

Approach #1: Naïve Compression

Approach #2: Columnar-Specific Encoding

More heavyweight compression is better when the network is slow.

Better compression ratios for larger message chunk sizes.

DATA SERIALIZATION

Approach #1: Binary Encoding

→ Client handles endian conversion.

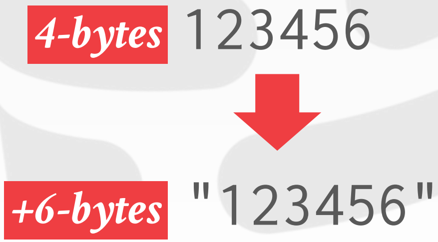
→ The closer the serialized format is to the DBMS's binary format, then the lower the overhead to serialize.

→ DBMS can implement its own format or rely on existing libraries (ProtoBuffers, Thrift, FlatBuffers).

Approach #2: Text Encoding

→ Convert all binary values into strings (`atoi`).

→ Do not have to worry about endianness.



STRING HANDLING

Approach #1: Null Termination

→ Store a null byte (`'\0'`) to denote the end of a string.
→ Client scans the entire string to find end.

Approach #2: Length-Prefixes

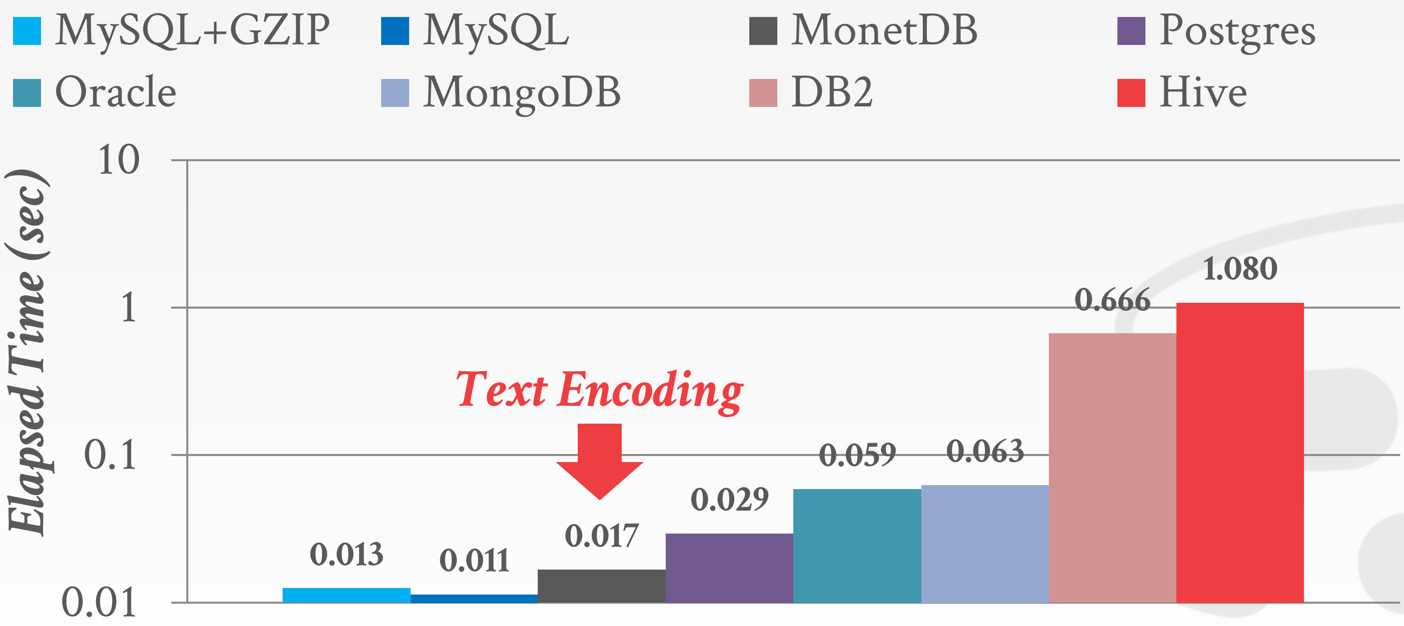
→ Add the length of the string at the beginning of the bytes.

Approach #3: Fixed Width

→ Pad every string to be the max size of that attribute.

NETWORK PROTOCOL PERFORMANCE

Transfer One Tuple from TCP-H LINEITEM

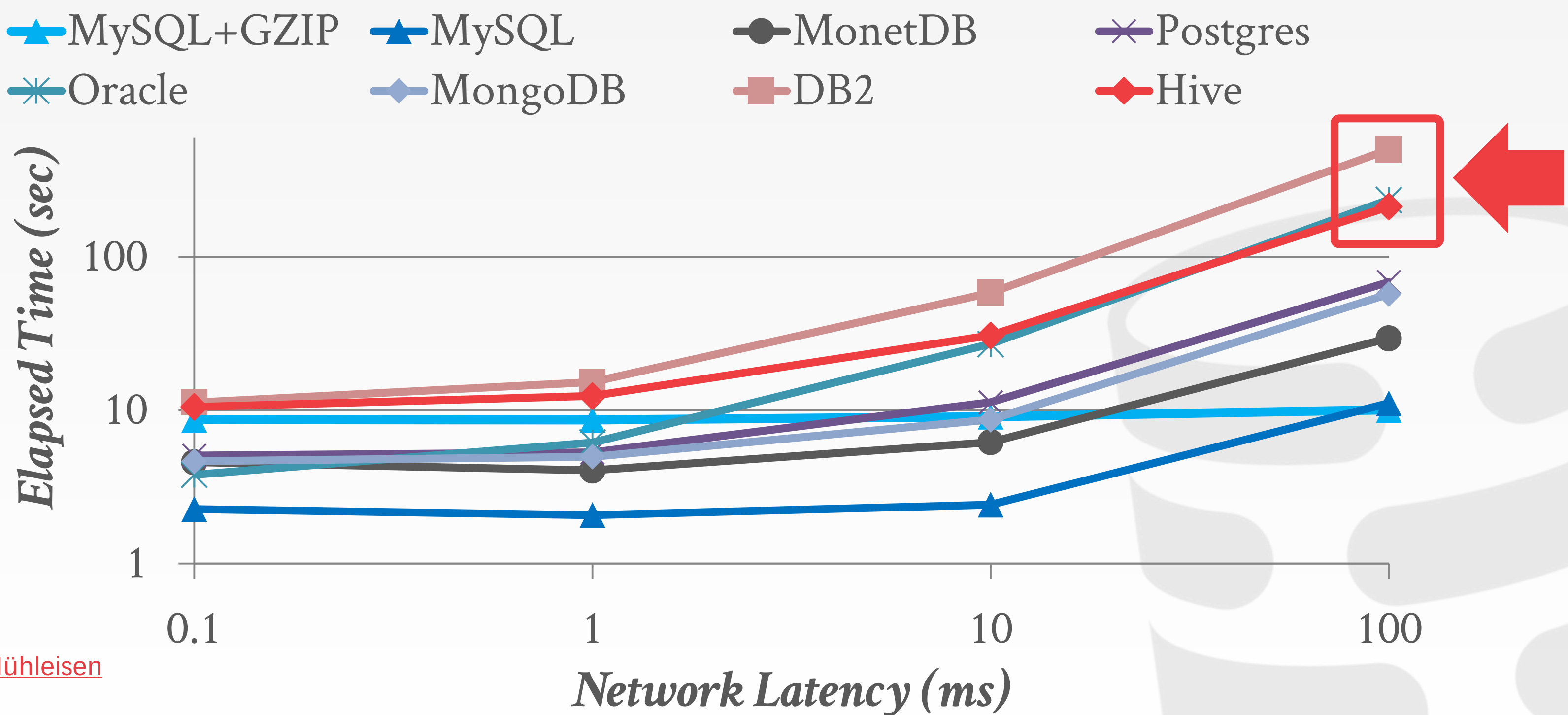


All Other Protocols Use Binary Encoding

Source: Hannes Mühleisen

NETWORK PROTOCOL PERFORMANCE

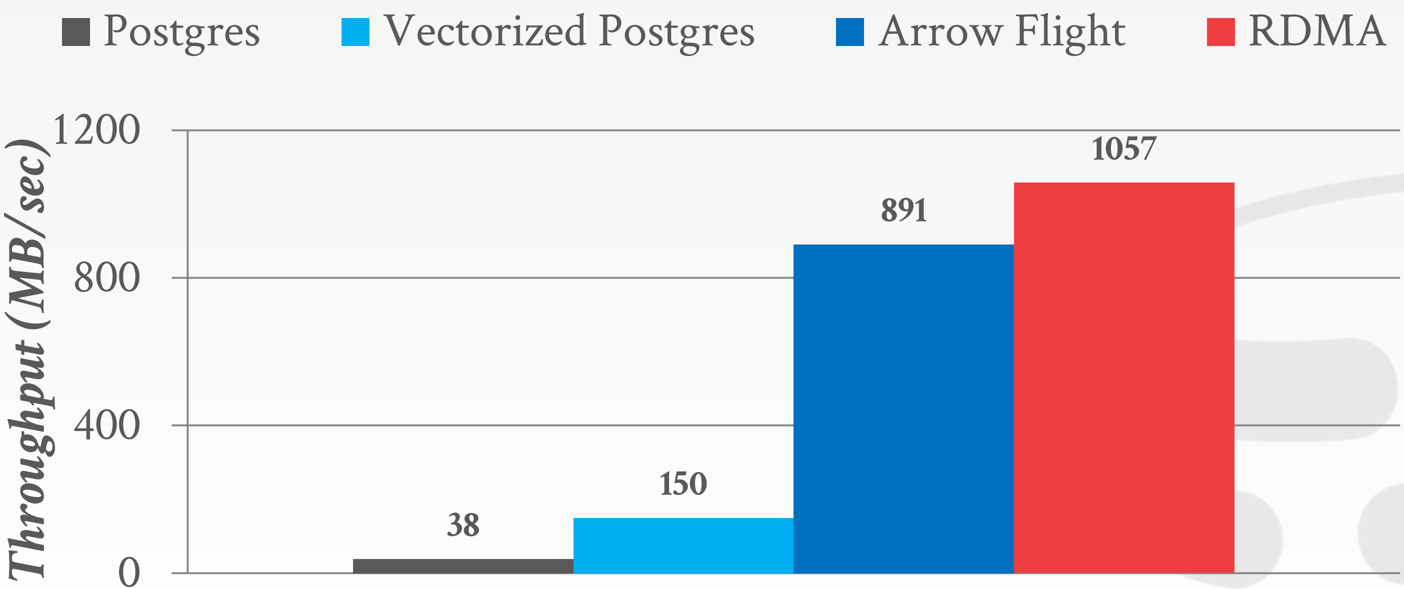
Transfer 1m Tuples from TCP-H LINEITEM



Source: Hannes Mühleisen

DATA EXPORT PERFORMANCE

Transfer 7GB of Tuples from TCP-C ORDER_LINE



Source: Tianyu Li

REPLICATION PROTOCOLS

DBMSs will propagate changes over the network to other nodes to increase availability.

→ Send either physical or logical log records.
→ Granularity of log record can differ from WAL.

Design Decisions:

→ Replica Configuration
→ Propagation Scheme

REPLICA CONFIGURATIONS

Approach #1: Master-Replica

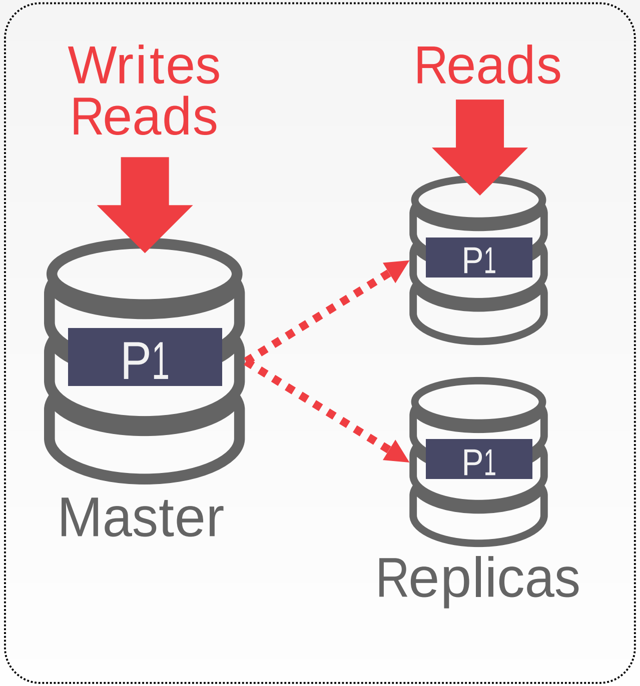
- → All updates go to a designated master for each object.
- → The master propagates updates to its replicas without an atomic commit protocol.
- → Read-only txns may be allowed to access replicas.
- → If the master goes down, then hold an election to select a new master.

Approach #2: Multi-Master

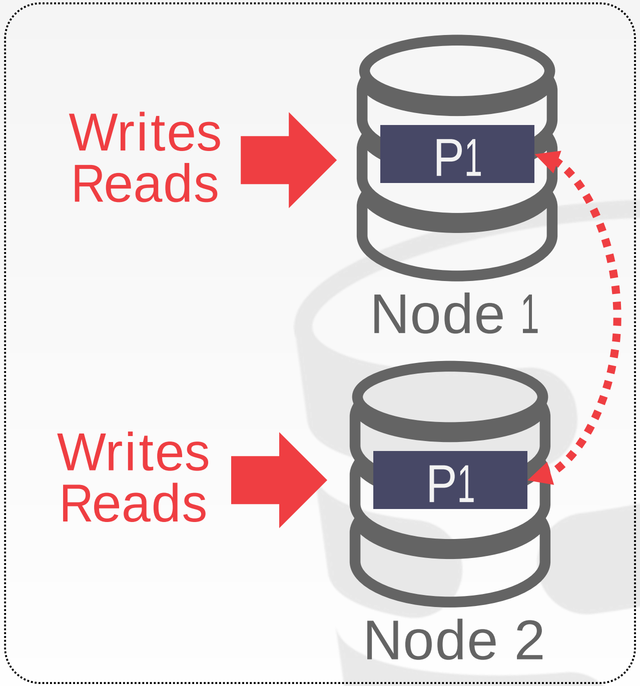
- → Txns can update data objects at any replica.
- → Replicas must synchronize with each other using an atomic commit protocol.

REPLICA CONFIGURATIONS

Master-Replica



Multi-Master



PROPAGATION SCHEME

When a txn commits on a replicated database, the DBMS decides whether it must wait for that txn's changes to propagate to other nodes before it can send the acknowledgement to application.

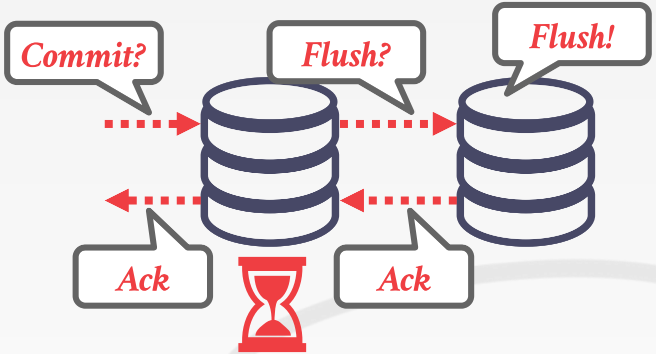
Propagation levels:

- → Synchronous (*Strong Consistency*)
- → Asynchronous (*Eventual Consistency*)

PROPAGATION SCHEME

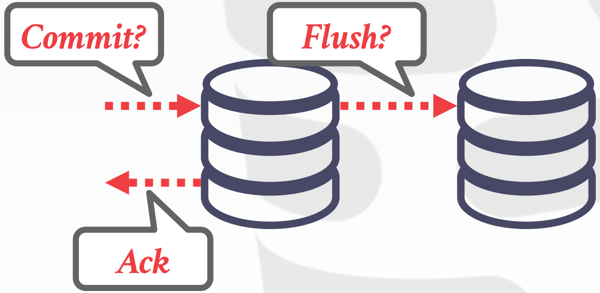
Approach #1: Synchronous

→ The master sends updates to replicas and then waits for them to acknowledge that they fully applied (i.e., logged) the changes.



Approach #2: Asynchronous

→ The master immediately returns the acknowledgement to the client without waiting for replicas to apply the changes.



OBSERVATION

The DBMS's network protocol implementation is not the only source of slowdown.

The OS's TCP/IP stack is slow…

→ Expensive context switches / interrupts
→ Data copying
→ Lots of latches in the kernel

KERNEL BYPASS METHODS

Allows the system to get data directly from the NIC into the DBMS address space.

→ No unnecessary data copying.
→ No OS TCP/IP stack.

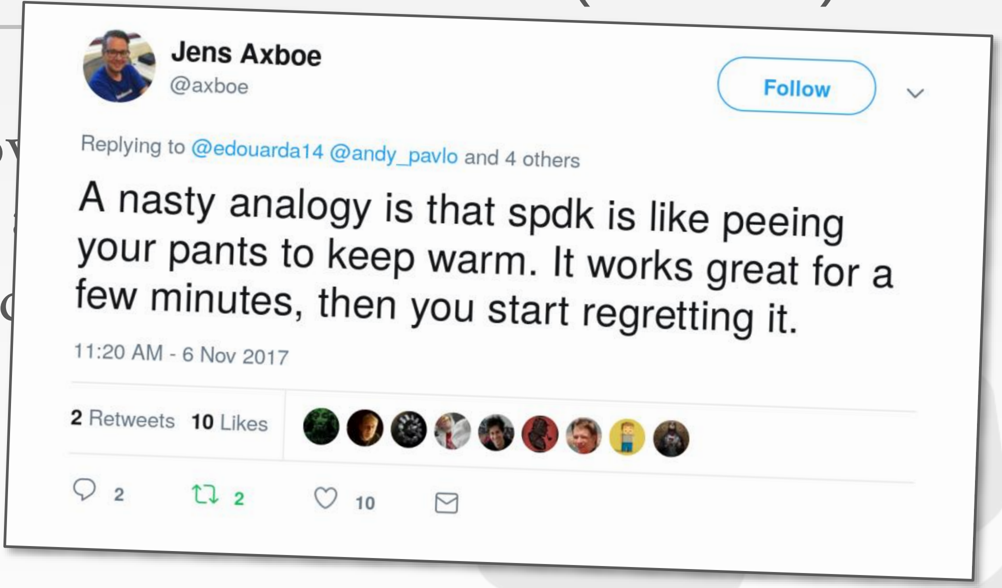
Approach #1: Data Plane Development Kit

Approach #2: Remote Direct Memory Access

DATA PLANE DEVELOPMENT KIT (DPDK)

Set of libraries that allo
directly. Treat the NIC

Requires the DBMS co
memory and buffers.
→ No data copying.
→ No system calls.



Example: ScyllaDB

REMOTE DIRECT MEMORY ACCESS

Read and write memory directly on a remote host without going through OS.

→ The client needs to know the correct address of the data that it wants to access.

→ The server is unaware that memory is being accessed remotely (i.e., no callbacks).

Example: Oracle RAC, Microsoft FaRM

PARTING THOUGHTS

A DBMS's networking protocol is an often overlooked bottleneck for performance.

Kernel bypass methods greatly improve performance but require more bookkeeping.

→ Probably more useful for internal DBMS communication.

PROJECT #2

Implement an in-memory B+Tree in the DBMS.

Must support the following features:
- → Insert / Get / Delete / Range Scan
- → Forward / Reverse Range Scans
- → Unique + Non-Unique Keys.

Other than implementing our API, you are free to do any optimization that you want.

PROJECT #2 – DESIGN

We will provide you with a header file with the index API that you have to implement.

→ Data serialization and predicate evaluation will be taken care of for you.

There are several design decisions that you are going to have to make.

→ There is no right answer.

→ Do not expect us to guide you at every step of the development process.

PROJECT #2 – TESTING

We are providing you with C++ unit tests for you to check your implementation.

We also have a Bw-Tree implementation to compare against.

We **strongly** encourage you to do your own additional testing.

PROJECT #2 – DOCUMENTATION

You must write documentation and comments in your code to explain what you are doing in all different parts.

We will inspect the submissions manually.

PROJECT #2 – GRADING

We will run additional tests beyond what we provided you for grading.

→ Bonus points will be given to the groups with the fastest implementation.

→ We will use ASAN when testing your code.

All source code must pass formatting and linter checks.

→ See documentation for formatting guidelines.

PROJECT #2 – GROUPS

This is a group project.

→ Everyone should contribute equally.

→ I will review commit history.

Email me if you do not have a group.

PROJECT #2

Due Date: March 15th @ 11:59pm
Projects will be turned in using Gradescope.

Full description and instructions:
https://15721.courses.cs.cmu.edu/spring2020/project2.html

NEXT CLASS

Let's start to talk about how to execute queries!