Communication System Design Projects

PROFESSOR DEJAN KOSTIC

PRESENTER: KIRILL BOGDANOV

KTH-DB Geo Distributed Key Value Store

DESIGN AND DEVELOP GEO DISTRIBUTED KEY VALUE STORE. DEPLOY AND TEST IT ON A NETWORK EMULATOR.

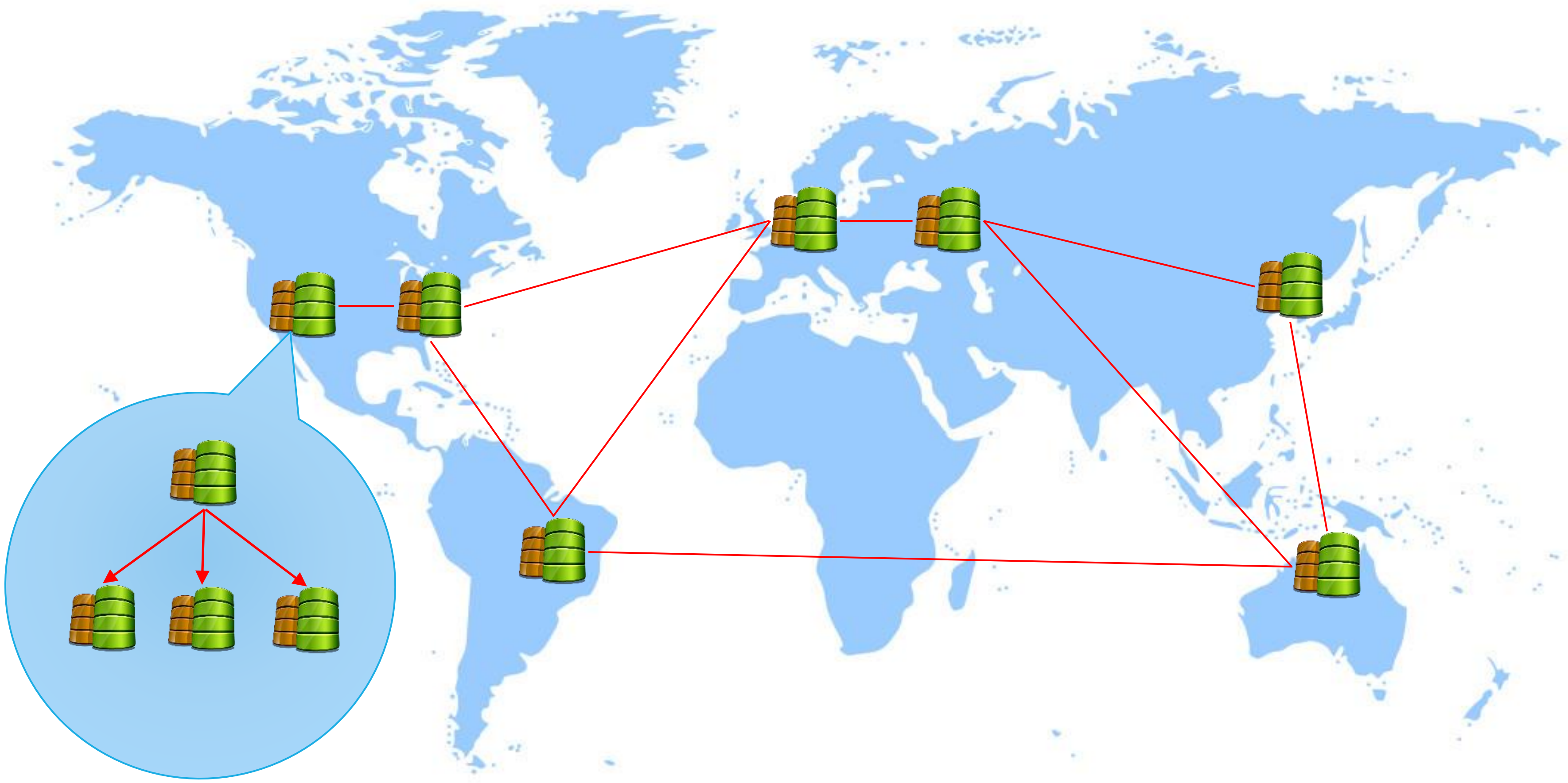
Project #5 KTH-DB

What is a Key Value Store?

- Is a NoSQL database
  - Does not support SQL queries
- Not a relationship database (Oracle, DB2, MySQL)
- It is a map<key, value>
- Simple API
  - `Put(key, value)`
  - `value = Get(key)`
  - `Delete(key)`

Key	Value (age, telephone, address...)
John Doe	{20 , 777333222, ”London”}
Robert Green	{54, 123456, ”New York”}
Alex Murphy	{45 , 31323311, ”Detroit”}

What is Geo-Distributed?



NoSQL Databases

- Google **BigTable**
- Facebook **Cassandra**
- Amazon **Dynamo**
- Yahoo **PNUTS**
- Many others…

Properties

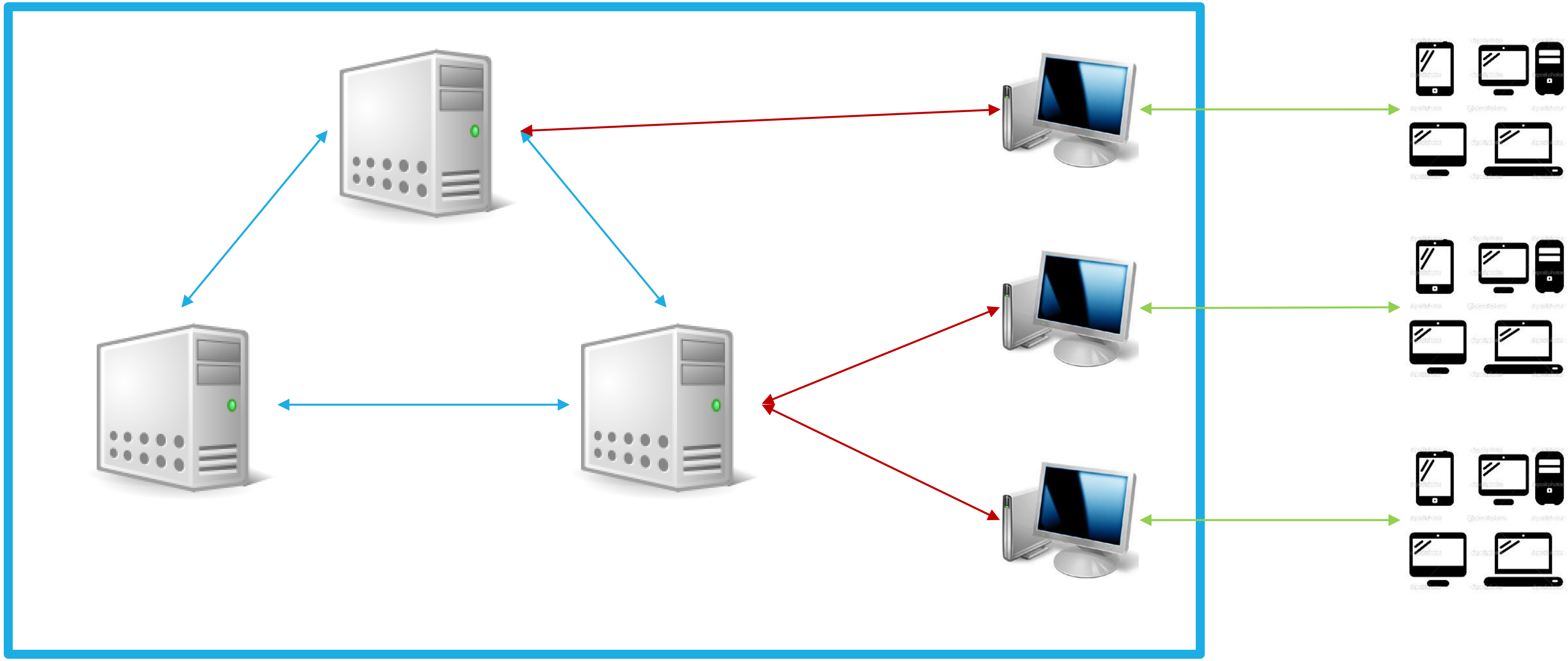
Geo Distributed Key Value Stores

- Outstanding pieces of engineering
- High Availability
- Disaster Recovery
- Data management
- Highly Scalable

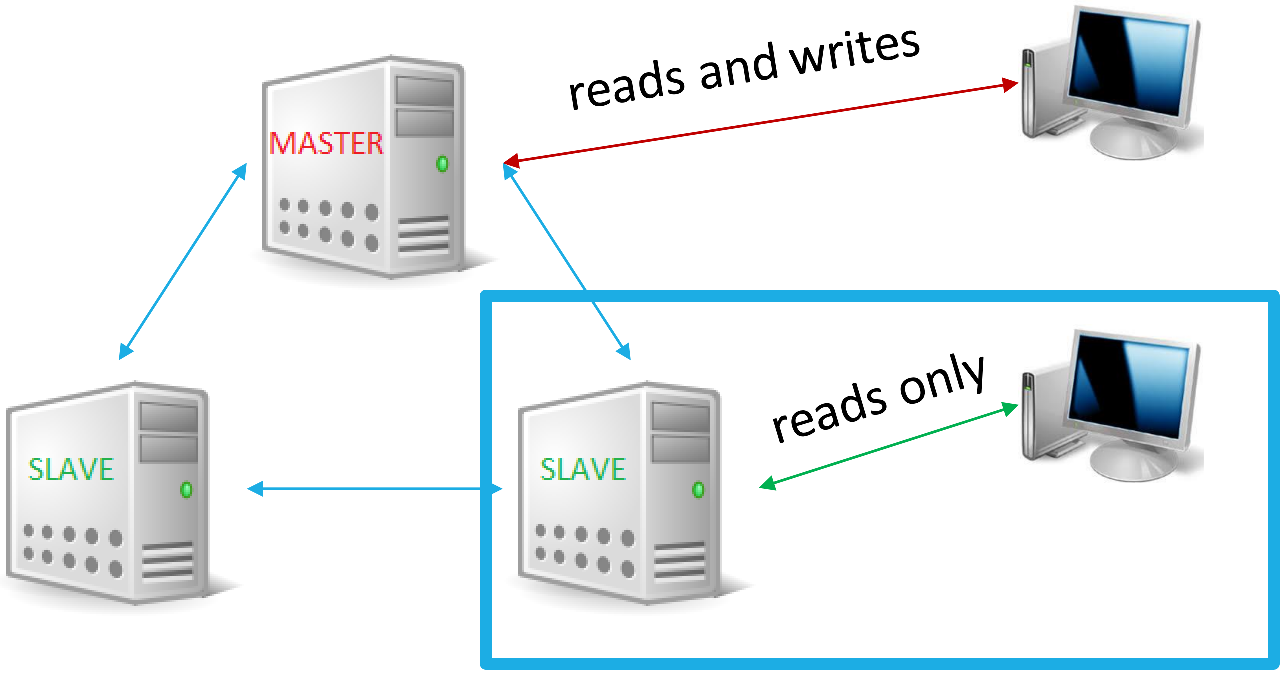
Why Implement a Key Value Store

- Hands on experience with NoSQL databases
- Object Oriented Design
- Understanding client/server model
- Algorithms, Data Structures and Networking
- Compare performance to a production industry systems
- Challanging back end engineering project!

Distributed Data Bases



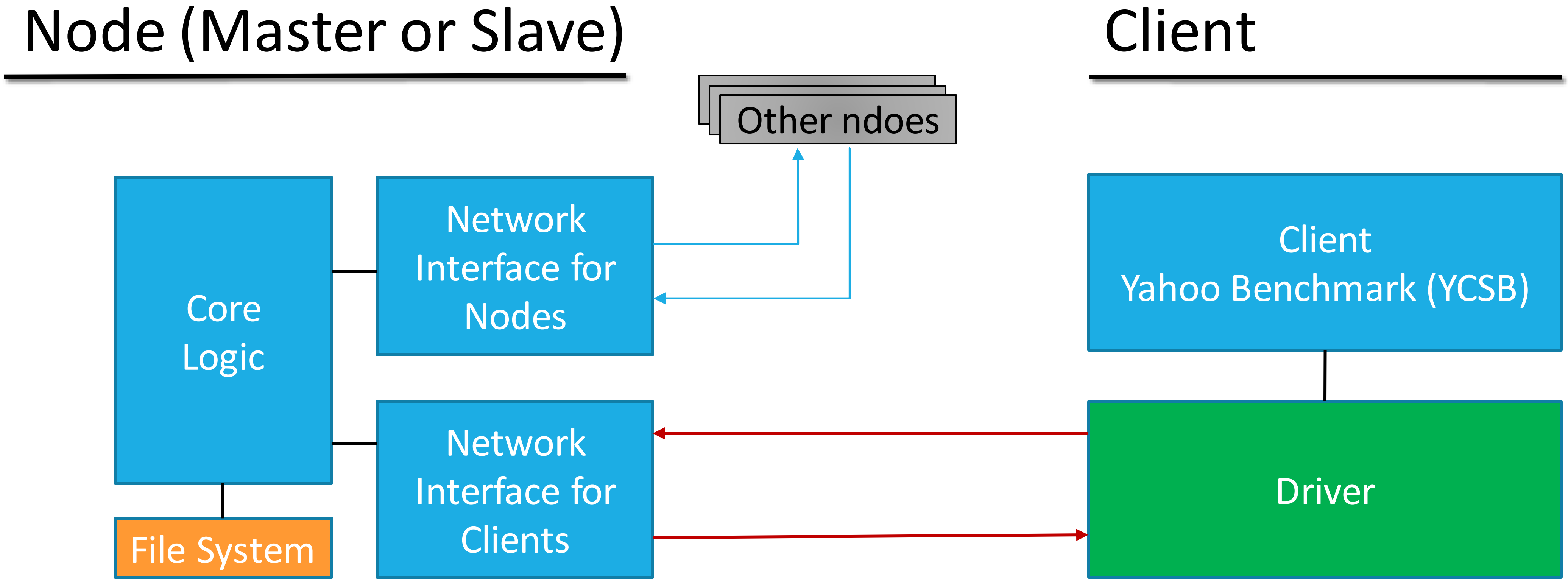
Master/Slave



Master:
- ❑ Accepts connections from clients
- ❑ Performs read/write (get/put) operations
- ❑ Sends updates to all Slaves

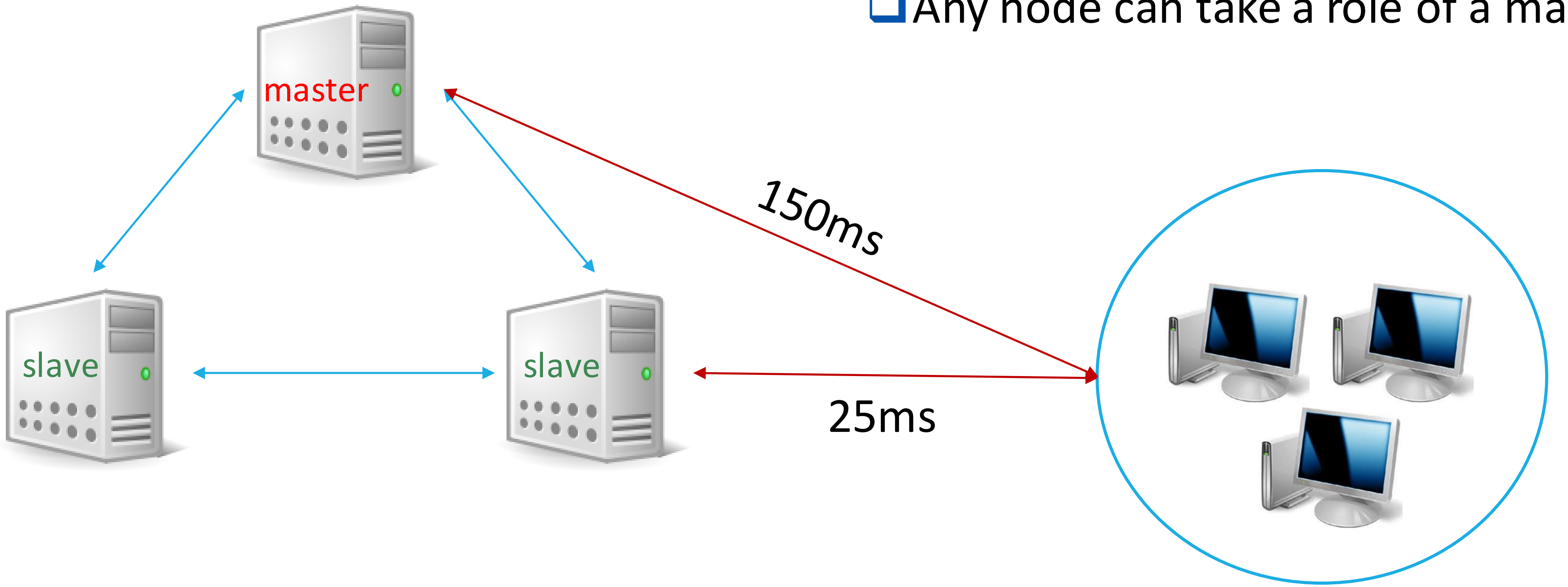
Slaves:
- ❑ Accepts connections from clients
- ❑ Performs reads only (get) operations
- ❑ Receives updates from Master

Scope - Architecture



Dynamic System

- Any node can take a role of a master



Requirements

1. Using MiniNet HiFi (network emulator) deploy KTH-DB on 10 nodes and run YCSB benchmark
    1. Use MiniNet to simulate network latencies
    2. Demonstrate performance of 500 writes/second and 2000 reads/second, store 10M keys and handle 100 clients accessing the data store in parallel.
2. Adopt Master/Slave architecture where writes executed only on Master node and reads can be done on any node
3. Create self aware system. Any node in the system can take role of the Master. Measure network latency from Master to Clients and assign ”master” status to maximize write performance.
4. C/C++/C11/Java
5. Use network libraries, for example zeromq, POCO etc

Reading Material

- Douglas Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, and Marcos K. Aguilera, ”**Transactions with Consistency Choices on Geo-Replicated Cloud Storage**”, no. MSR-TR-2013-82, September 2013
- Ghemawat, S., Gobioff, H., and Leung, S.-T. 2003. “**The Google file system**”. In 19th Symposium on Operating Systems Principles. Lake George, NY. 29-43.
- Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. 2006. “**Bigtable: a distributed storage system for structured data**”. In roceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation - Volume 7 (Seattle, WA, November 06 - 08, 2006). USENIX Association, Berkeley, CA, 15-15.
- B.F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. **“Benchmarking Cloud Serving Systems with YCSB”**. In ACM Symposium on Cloud Computing, 2010.

Global Distributed Snapshot

DEVELOP DISTRIBUTED TRANSACTION ENGINE, COLLECT ITS GLOBAL SNAPSHOT, USE THIS SNAPSHOT TO DETECT AND RESOLVE DEADLOCKING IN THE SYSTEM

Project #6 Snapshot

Distributed Systems

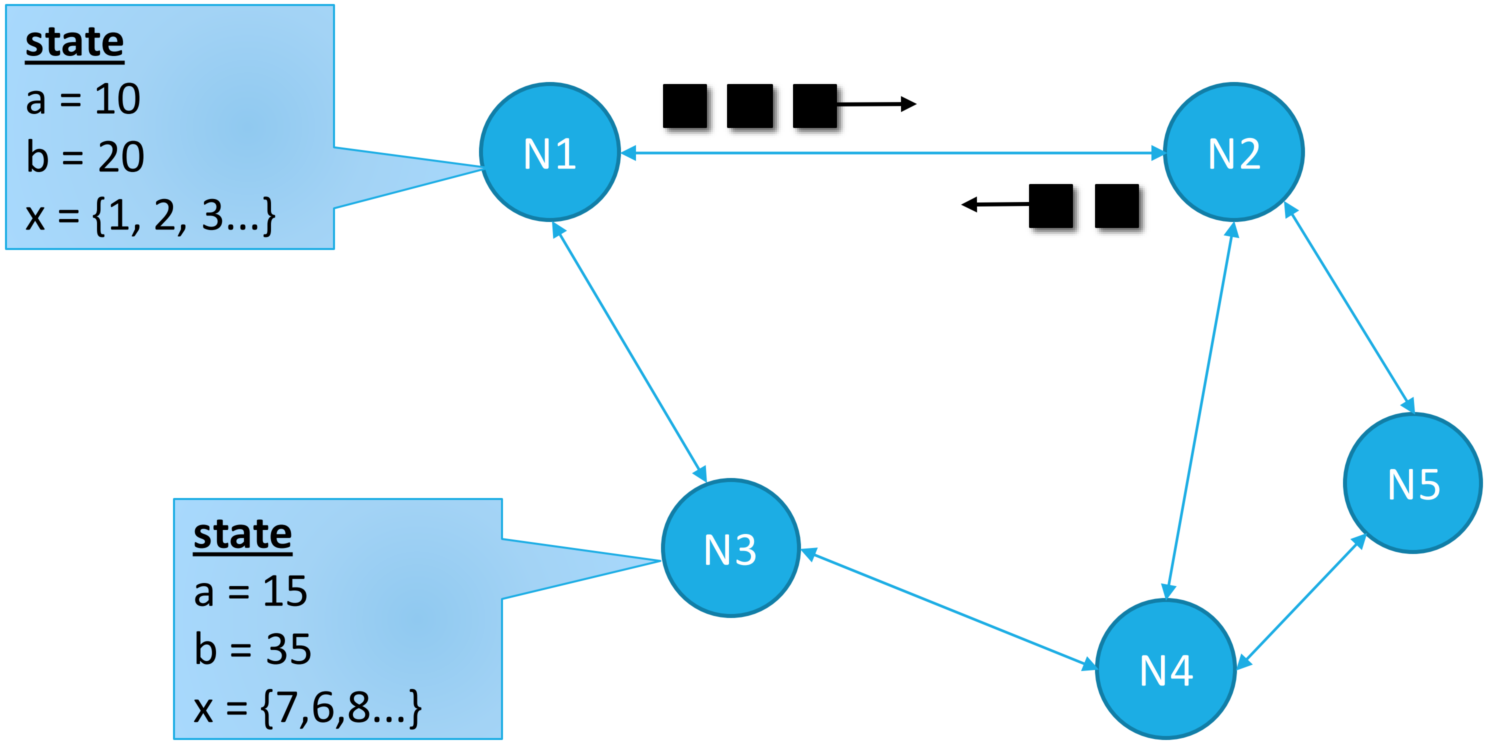
- Parallel and High Performance Computing
- Internet protocols
- Distributed Databases
- P2P networks
- Critical Systems that require high fault tolerance
- Wireless sensors
- Many more ...

Properties

- Collection of independant processes
- Coordinate their work by sending messages over network medium
- No shared memory
- No global synchronized clock
- Numerous event orderings due to nature of network, failures and external events

- Hard to design develop and manage

Global Distributed Snapshot



Used for:

- ✓ Distributed Debugging
- ✓ Failure recovery
- ✓ Future state prediction
- ✓ Deadlock detection

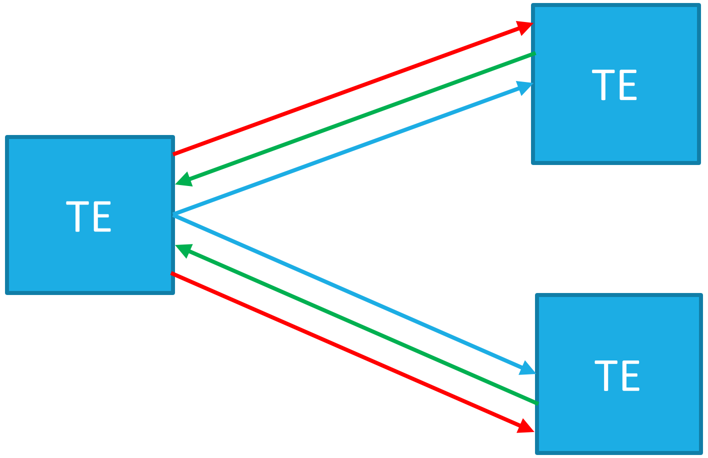
Scope

Transaction Engine	Simple Distributed Database capable of performing Atomic Transactions
Sample Workload	Randomly generated workloads to simulate human behavior
Collect Global Snapshots	Periodically collect global system states
Detect Deadlocks	Analyse each snapshot and check for Deadlocks
Resolve Deadlocks	Apply Deadlock resolution mechanism

Scope – Transaction Engine

- **Transaction Engine**
- Sample Workload
- Collect Global Snapshots
- Detect Deadlocks
- Resolve Deadlocks

Account Number	Value
100100	2 000 kr
100123	100 000 kr



Atomic Transactions

- ❑ Obtain lock on a row
- ❑ Receive confirmation
- ❑ Send out new value

Scope - Workload

- Transaction Engine
- **Sample Workload**
- Collect Global Snapshots
- Detect Deadlocks
- Resolve Deadlocks

- Simulate user behavior by generating random transactions
- Periodically each node in the system selects two random accounts and transfers random amount of money from one to another
- Each transaction is Atomic and each change should be consistent across all databases

Scope – Global Snapshot

- Transaction Engine
- Sample Workload
- **Collect Global Snapshots**
- Detect Deadlocks
- Resolve Deadlocks

- ❑ No Global Clock –not possible to order everyone to record their state at a time ”*T*”
- ❑ Investigate this problem and choose an algorithm for obtaining global distributed snapshots
- ❑ Periodically collect snapshots

Scope – Deadlocks

Transaction Engine

Sample Workload

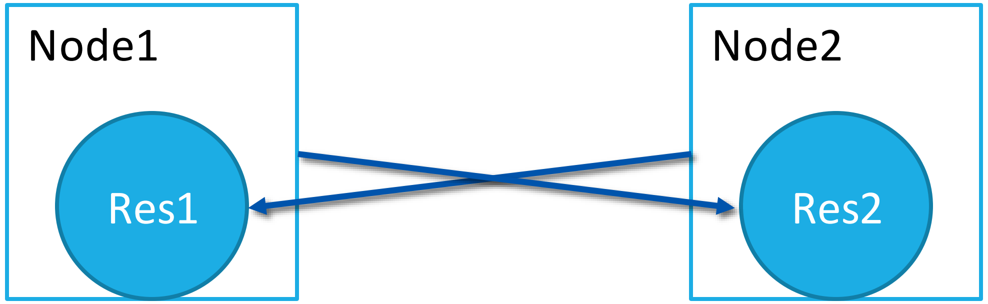
Collect Global Snapshots

Detect Deadlocks

Resolve Deadlocks

Deadlocking - occure when two or more processes compete for a shared resource and waiting for each other to finish, thus neither ever does

- ❑ Analyse global snapshot and detect dead locks



Scope – Deadlock Resolution

- Transaction Engine
- Sample Workload
- Collect Global Snapshots
- Detect Deadlocks
- **Resolve Deadlocks**

- ❑ Decide how deadlock will be resolved
- ❑ Which transaction will be allowed to succeed?

Requirements

1. Using MiniNet HiFi (network emulator) deploy Transaction Engine on 5 nodes
2. Demonstrate that system can collect Distributed Global Snapshots
3. Show that all snapshots are consistent
4. Detect deadlocks at a runtime and resolve them using algorithm of choice
5. C/C++/C11/Java
6. Use network libraries, for example zeromq, POCO etc

Reading Material

- LAMPORT, L. “**Time, clocks, and the ordering of events in a distributed system**”. Commun. ACM 21,7 (July 1978), 558-565.
- K. Mani Chandy and Leslie Lamport. “**Distributed snapshots: Determining global states of distributed systems**”. ACM Transactions on Computer Systems, 3(1):63–75, February 1985.
- A.D. Kshemkalyani, M. Raynal, and M. Singhal, “**An Introduction to Snapshot Algorithms in Distributed Computing**”, Distributed Systems Eng. J., vol. 2, no. 4, pp. 224-233, Dec. 1995.
- CHANDY, K.M., AND MISRA, J. “**A distributed algorithm for detecting resource deadlocks in distributed systems**”. In Proc. A CM SIGA CT-SIGOPS Syrup. Principles of Distributed Computing ACM, New York, 1982, pp. 157-164.
- CHANDY, K. M., MISRA, J., AND HAAS, L. “**Distributed deadlock detection**”. ACM Trans. Comput. Syst. I,2 (May 1983), 144-156.

Q&A

Global Distributed Snapshot	KTH-DB
Distributed Transaction Engine	Geo-Distributed Key Value Store
Atomic Transactions	Master-Slave Architecture
Global Distributed Snapshot	Strong Consistency
Deadlock Detection/Resolution	Yahoo benchmark
Deployemnt on a Network Emulator	Deployment on a Network Emulator