

# Nexres

The next generation of ResilientDB:  
a distributed blockchain platform

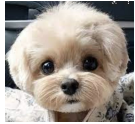
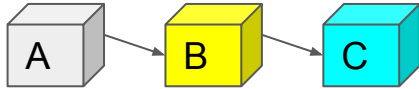
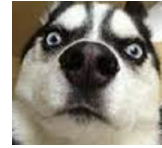
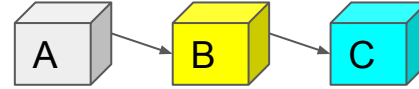
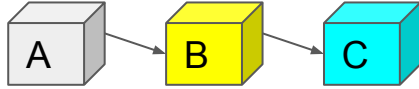
Junchao Chen  
jucchen@ucdavis.edu

 UC DAVIS

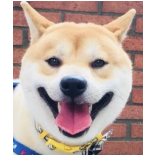
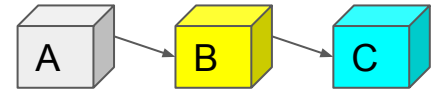
# Introduction

- What is Blockchain
- What is Nexres
- KV Server Example

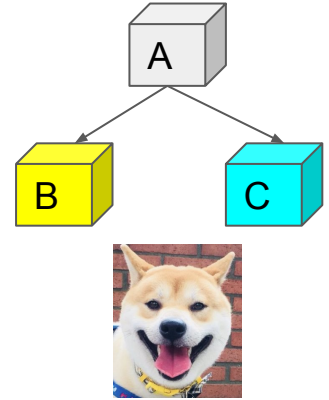
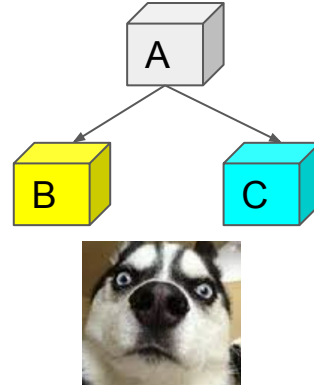
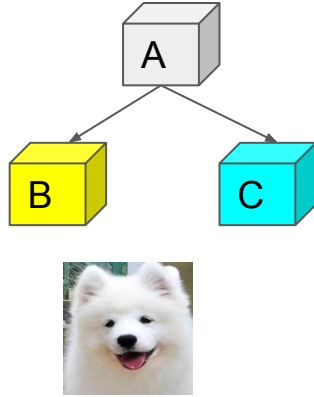
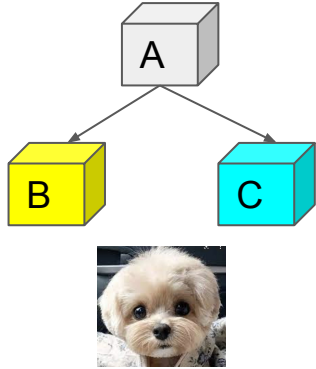
# What is blockchain



- A distributed ledger
- Users own their individual copies
- Consistent structure
- Idempotent (same input, same output)
- Can trace the histories
- Immutable
- Run in Byzantine Env

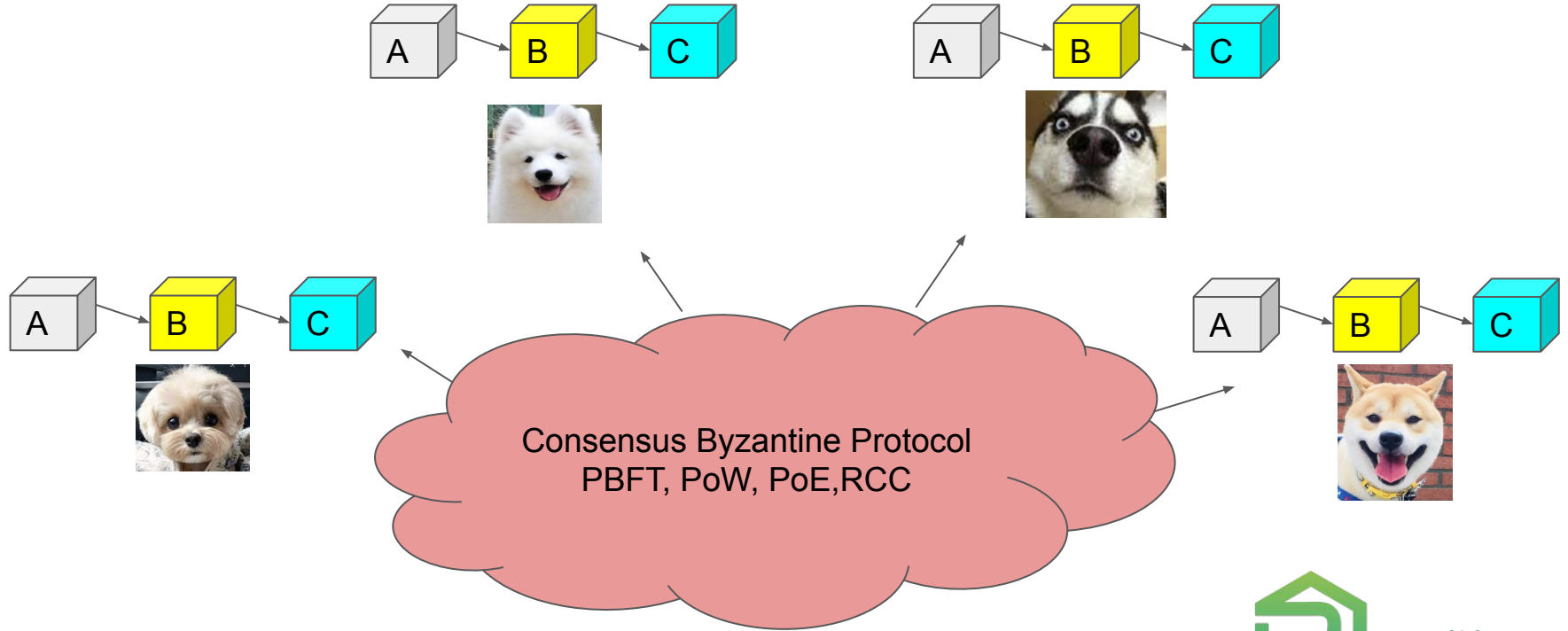


# Is blockchain a single chain?



- Can be any DAG graph!!!

# Behind blockchain



# Nexres

An open source high performance Remote Procedure Call (RPC) framework running in a byzantine environment.

A distributed platform helping people to develop an indivisual blockchain system.

- Easy to develop your own application or smart contract.
- Easy to develop your own consensus protocol

Using user kindly development suitekit.

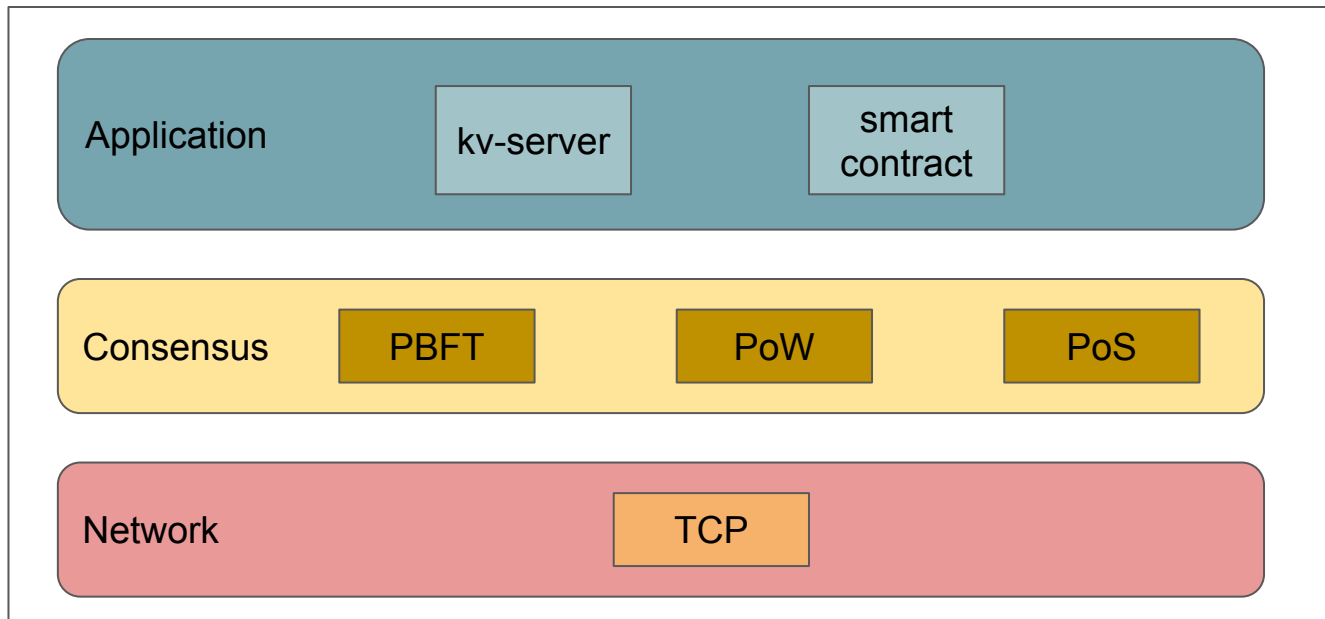
- Written in C++: easy to access SDKs.
- Protobuf Message: easy to define a serialabe message
- Bazel Build: easy to build the system



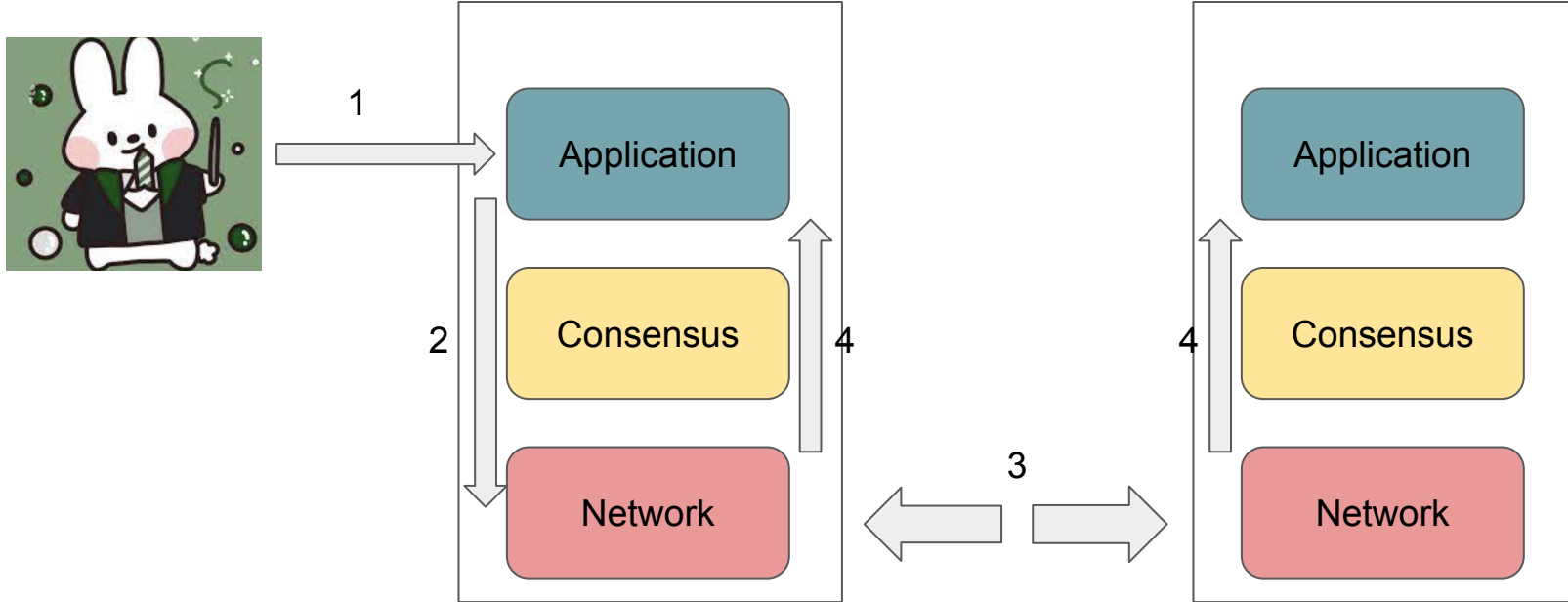
# Nexres

## Three Layers

1. Application
2. Consensus
3. Network

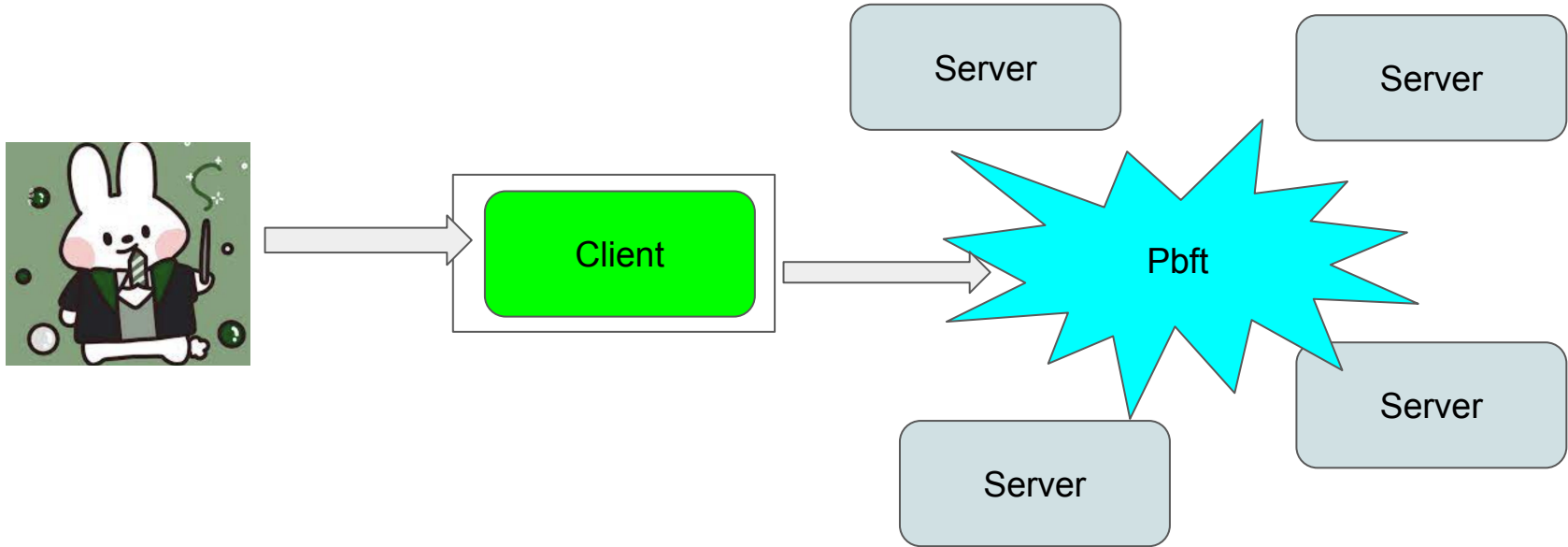


# Nexres: Interact with Application SDK

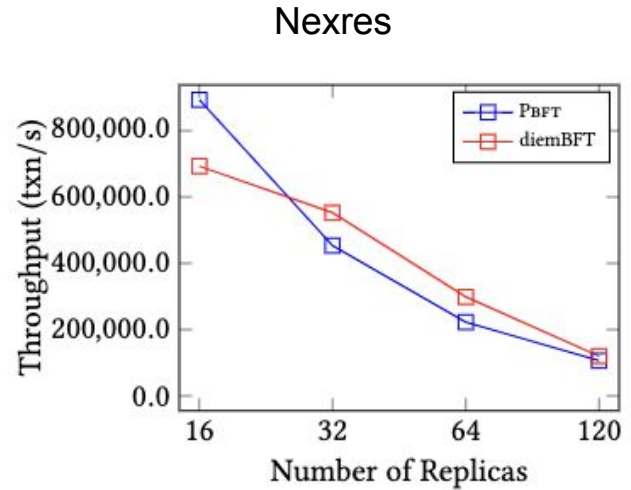
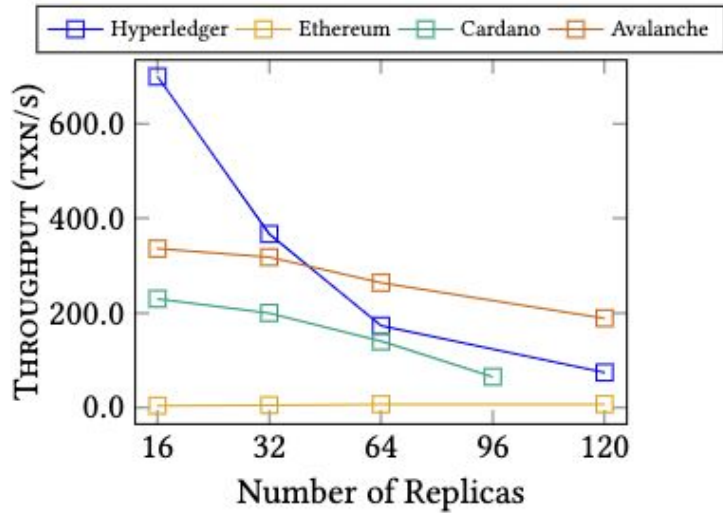




# Example: Nexres - KVServer with Pbft



# Performance



# How to run Nexres

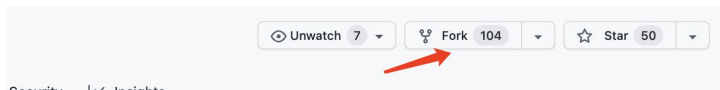
OS: Ubuntu 20.04

Github: <https://github.com/resilientdb/resilientdb/tree/nexres>



# How to run Nexres

1. Clone the code: `git clone https://github.com/resilientdb/resilientdb.git`
  - a. remember fork first.



2. Switch to nexres branch
  - a. `git checkout -b nexres remotes/origin/nexres`
3. Run the initialization script to set up the environment
  - a. `sh INSTALL.sh`
4. Start the kv server
  - a. `sh example/start_kv_server.sh` (a few minutes to compile)
5. Run client request
  - a. `bazel-bin/example/kv_server_tools example/kv_client_config.config set cs 265`
  - b. `bazel-bin/example/kv_server_tools example/kv_client_config.config get cs`

Thanks!



# GeoBFT

Wayne Wang  
[wjawang@ucdavis.edu](mailto:wjawang@ucdavis.edu)

Exploratory Systems Lab  
University of California, Davis



# Introduction

- Geo-scale Deployments
- What is Clustering
- GeoBFT Consensus

# Geo-scale Deployment

GeoBFT is designed to deal with geographical scale deployments in which many replicas are spread across the world.

	<i>Ping round-trip times (ms)</i>						<i>Bandwidth (Mbit/s)</i>					
	<i>O</i>	<i>I</i>	<i>M</i>	<i>B</i>	<i>T</i>	<i>S</i>	<i>O</i>	<i>I</i>	<i>M</i>	<i>B</i>	<i>T</i>	<i>S</i>
Oregon ( <i>O</i> )	≤ 1	38	65	136	118	161	7998	669	371	194	188	136
Iowa ( <i>I</i> )		≤ 1	33	98	153	172		10004	752	243	144	120
Montreal ( <i>M</i> )			≤ 1	82	186	202			7977	283	111	102
Belgium ( <i>B</i> )				≤ 1	252	270				9728	79	66
Taiwan ( <i>T</i> )					≤ 1	137					7998	160
Sydney ( <i>S</i> )						≤ 1						7977

Data from Google Cloud experiment



# Clustering

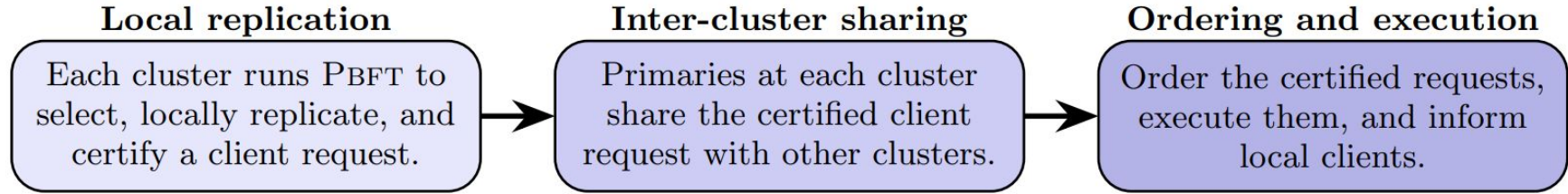
Clustering replicas in a region together, and favoring communication within such clusters over global inter-cluster communication.

# Clustering

Clustering replicas in a region together, and favoring communication within such clusters over global inter-cluster communication.

- This makes sure that the protocol is aware of the network topology.
- The protocol is also decentralized, no single replica or cluster should be responsible for coordinating decisions, because centralized design can limit the outgoing bandwidth and latency of that particular replica/cluster.

# GeoBFT Protocol

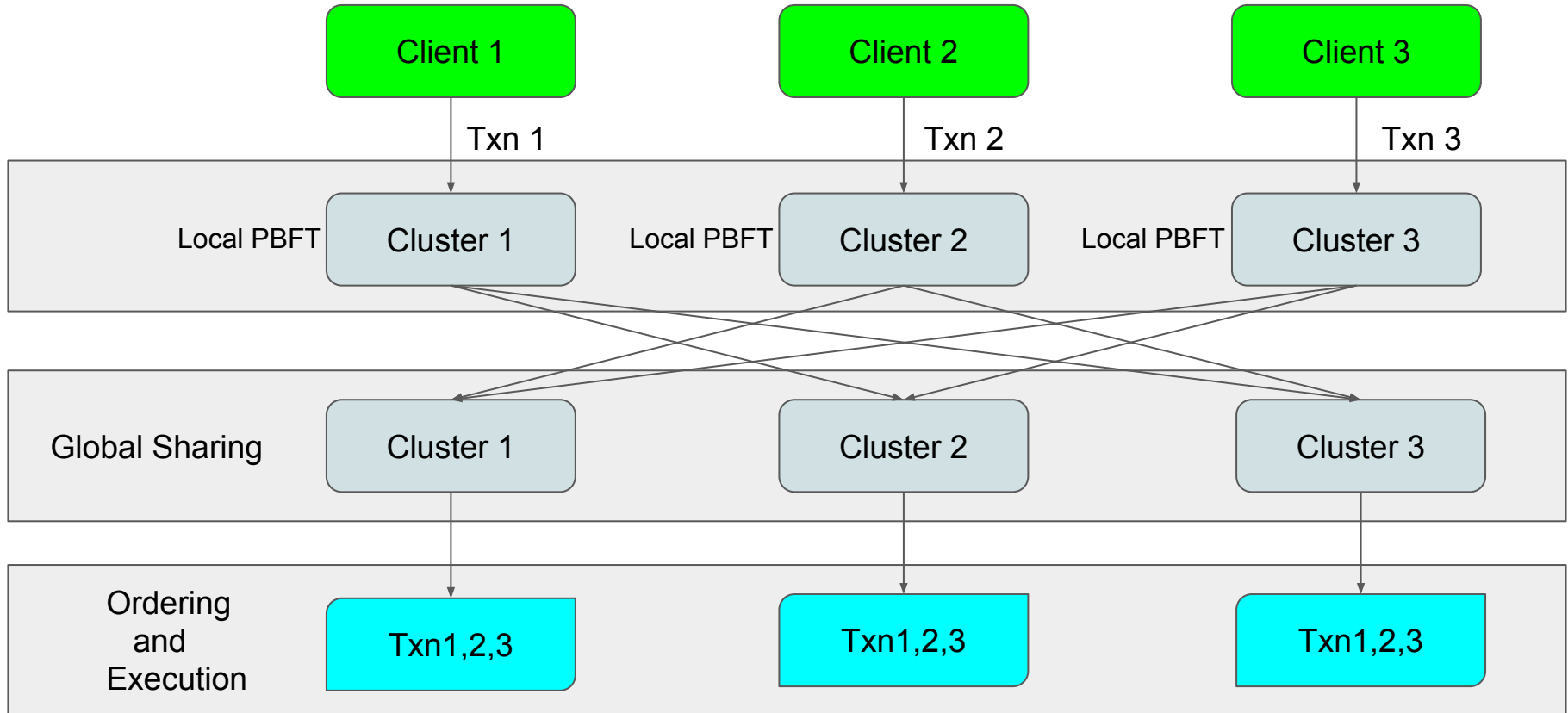


GeoBFT operates in rounds.

There are 3 main steps in each round:

1. Local replication
2. Inter-cluster sharing (Global sharing)
3. Ordering and execution

# GeoBFT Protocol



# Q & A

# NexRes In Action

@divjeet

Exploratory Systems Lab  
University of California, Davis



# NexRes Application

- Deployment Manager
- Block Explorer
- Monitoring Dashboard (Yet to be integrated with NexRes Webapp)
- NFT Marketplace

# Deployment Manager ( or Node Manager)

The screenshot shows the 'Deployment Configuration' page of the ResilientDB NexRes interface. The top navigation bar includes the ResilientDB NexRes logo and links for Home, Deployment, Monitoring, and Explorer. The main content area features a progress bar with three steps: Endpoints, Ordering, and Network. The 'Endpoints' step is currently active. Below the progress bar, a central card displays the IP address '127.0.0.1:3000' and shows '0 Replicas' and '0 Clients'. At the bottom, there is an input field containing '127.0.0.1:3000' and an 'Add Endpoint' button. Navigation buttons for '< Previous' and 'Next >' are also present.

**ResilientDB**  
NexRes

Home Deployment Monitoring Explorer

## Deployment Configuration

Endpoints Ordering Network

127.0.0.1:3000  
0 Replicas  
0 Clients

127.0.0.1:3000 Add Endpoint

< Previous Next >

Add Endpoints of  
Replicas and Clients



# Deployment Manager ( or Node Manager)



**ResilientDB**  
NexRes



Home



Deployment



Monitoring



Explorer

## Deployment Configuration

Endpoints

Ordering

Network

Protocol:

### Protocol Options

Num Commitment Threads:

Watermark width:

[< Previous](#)

[Next >](#)

Choose Desired  
Protocol (Currently  
only PBFT supported)  
and protocol options

# Deployment Manager ( or Node Manager)

- Create clusters and deploy.
- After deploying you can see each replica and client, and perform actions like pause or kill on them.

The screenshot shows the ResilientDB NexRes web interface. The top navigation bar includes the ResilientDB NexRes logo and four menu items: Home, Deployment (highlighted in green), Monitoring, and Explorer. The main content area is titled "Deployment Configuration" and features a progress bar with three steps: Endpoints, Ordering, and Network. Below the progress bar, a sad face icon and the text "No clusters added yet." are displayed. A green button labeled "Add new cluster" is positioned below the text. At the bottom, there are two buttons: "Previous" and "Deploy" (highlighted in green).

# Block Explorer



ResilientDB  
NexRes



Home



Deployment



Monitoring



Explorer

## NexRes BlockChain (Dummy Data)

Active Replica

11

Active Client

2

Batch Size (MB)

112

Checkpoint Window Size

12

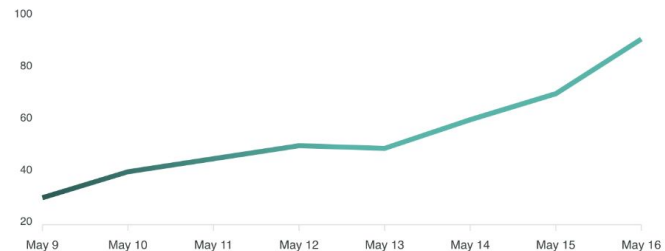
Client Timer

3

Proposal Timer

4

## NexRes Transaction History



## Latest Blocks

#	Block #	Hash	Size	Block Height	Transactions	Mined By	Gas Used	Commit Certificate	Created At
1	981230	993512	1248 bytes	21312	2 transactions	0x123123235	7455(99%)	commit certificate	1:29 AM Thursday, May 19, 2022 (PDT)

Gives an overview of  
the blockchain

# Block Explorer

- Shows history of blockchain.
- User can select the block# or transactions to dive deeper as shown.

Latest Blocks										
#	Block #	Hash	Size	Block Height	Transactions	Mined By	Gas Used	Commit Certificate	Created At	
1	<a href="#">981230</a>	993512	1248 bytes	21312	<a href="#">2 transactions</a>	0x123123235	7455(99%)	commit certificate	1:29 AM Thursday, May 19, 2022 (PDT)	
2	<a href="#">98123</a>	993513	1248 bytes	21312	<a href="#">1 transactions</a>	0x123123235	7455(99%)	commit certificate	4:29 AM Thursday, May 12, 2022 (PDT)	
3	<a href="#">98124</a>	993514	3248 bytes	21312	<a href="#">1 transactions</a>	0x123123235	7455(99%)	commit certificate	2:21 AM Thursday, May 29, 2022 (PDT)	
4	<a href="#">98125</a>	993515	2228 bytes	21312	<a href="#">1 transactions</a>	0x123123235	7455(99%)	commit certificate	2:49 AM Thursday, May 14, 2022 (PDT)	
5	<a href="#">98125</a>	993515	2228 bytes	21312	<a href="#">1 transactions</a>	0x123123235	7455(99%)	commit certificate	2:49 AM Thursday, May 14, 2022 (PDT)	

# Block Explorer



**ResilientDB**  
NexRes



Home



Deployment



Monitoring



Explorer

## Transactions for Block 981230

#	Transaction Hash	Block	Client Id	Transaction Data	Client Signature
1	0x123124123	993516	0x12423123	data	signature
2	0x12312999	993516	0x12423123	data	signature

Shows transactions  
stored inside a  
block.

# Block Explorer



**ResilientDB**  
NexRes


  
Home

  
Deployment

  
Monitoring

  
Explorer

## Block Info

Block #	981230	Hash	993512
Size	1248	Block Height	21312
Transactions	2 transactions	Mined By	0x123123235
Difficulty	4121142	Total Difficulty	5235325235
Gas Used	7455(99%) 	Parent Hash	0x8123812381290
State Root	0x9812649816389	Nounce	0x43627812f
Commit Certificate	commit certificate	Created At	1:29 AM Thursday, May 19, 2022 (PDT)

Shows complete  
information of a  
block.

# NFT Marketplace

## Home Page

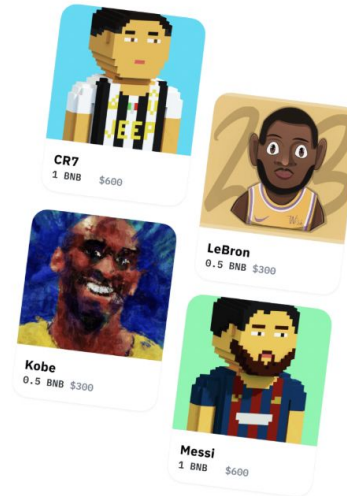
### NFT MARKETPLACE

[HOME](#)[PRODUCTS](#)[CONTACT](#)[TEAM](#)

DISCOVER, COLLECT, AND  
SELL EXTRAORDINARY

# NFTs

on the world's first & largest NFT marketplace

[EXPLORE](#)[CREATE](#)

User can either  
explore or  
create(mint) an NFT

# NFT Marketplace

User can upload a  
file to mint NFT


## Mint NFT

### NFT MARKETPLACE

HOMEPRODUCTSCONTACTTEAM

#### Sell your art!

Fill the required information and enter the world of NFTs



First name


Last name

Product name

About



Brief description for your NFT.

Upload Art

  
Upload a file or drag and drop  
PNG, JPG, GIF up to 10MB

Save

©2021 All rights reserved Privacy policy Terms of service





# NFT Marketplace

## Products List

**NFT MARKETPLACE**

[HOME](#)

[PRODUCTS](#)

[CONTACT](#)

[TEAM](#)

### COLLECTIONS



Cryptopunk #1

\$192



Cryptopunk #2

\$35



Cryptopunk #3

\$89



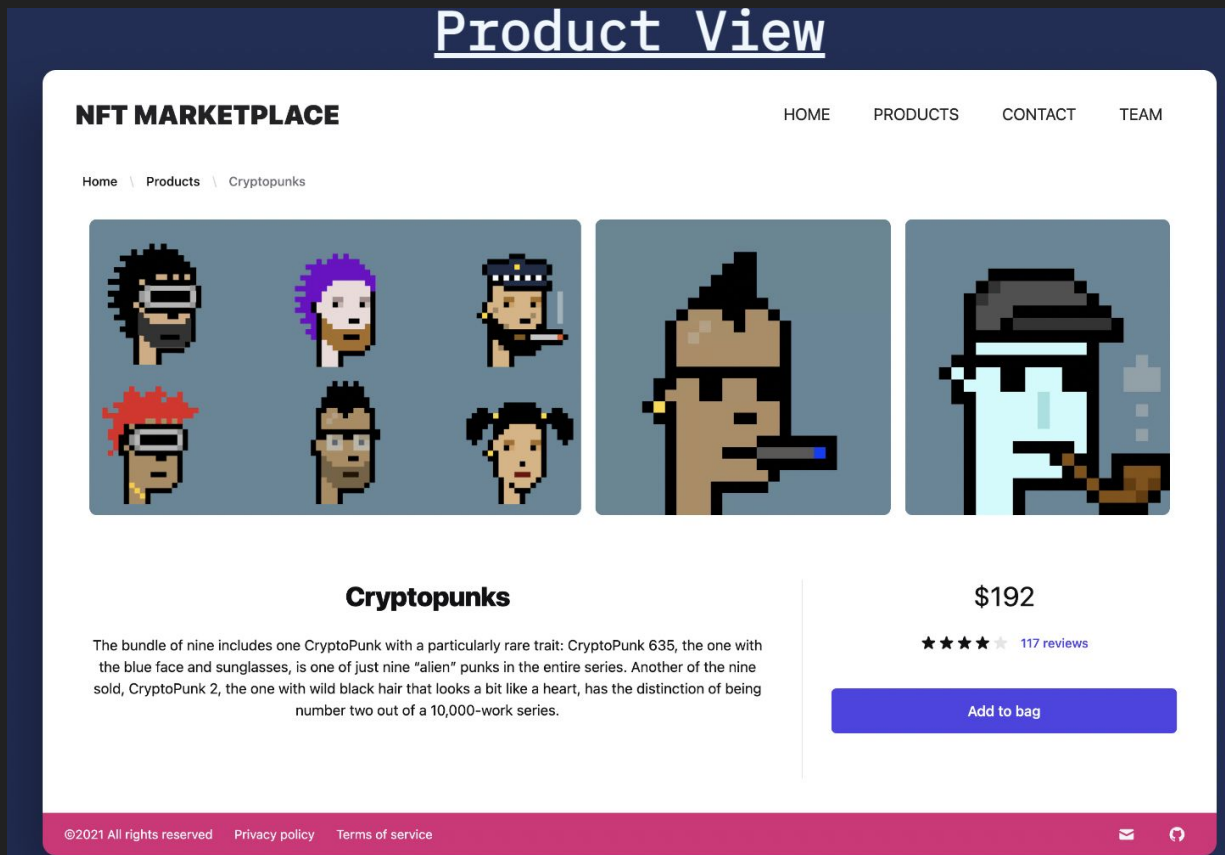
Cryptopunk #4

\$35

Shows all available  
NFTs



# NFT Marketplace



# Project Ideas

- Implement the missing links
  - Connecting Block Explorer to NexRes using the new SDK
  - Connecting Monitoring Dashboard to NexRes Webapp
  - Connecting NFT Marketplace to NexRes using the new SDK
- Implement a crypto wallet on NexRes
  - Add this wallet to NFT Marketplace
- Adding smart contract capabilities to NexRes or more capabilities to the SDK.

# Nexres Durability Layer

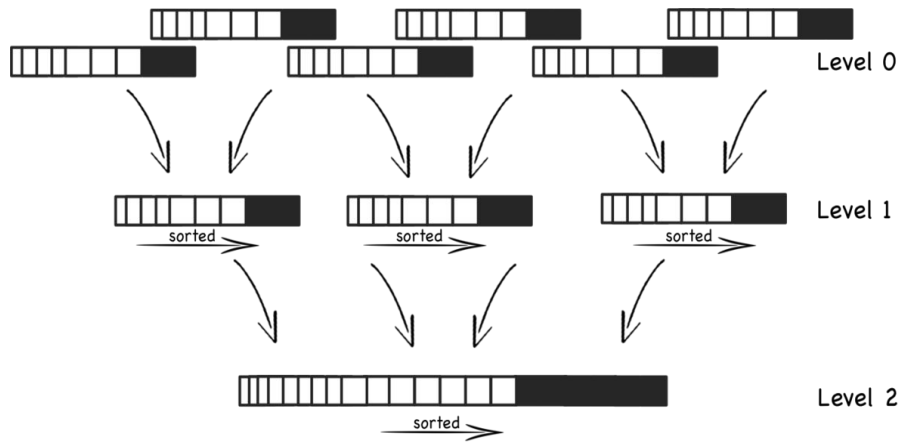
Julieta Duarte and Glenn Chen

Exploratory Systems Lab  
University of California, Davis





- key-value storage library written at google
- data is sorted in key and value pairs
  - order can be changed by callers with custom comparison functions
- storage architecture is a log-structured merge tree (LSM)
  - write-optimized B-tree variant
  - optimized more for large sequential writes compared to small random writes
  - mutable MemTables in memory and immutable SSTables on disk
    - made up of blocks



Compaction continues creating fewer, larger and larger files

[https://en.wikipedia.org/wiki/Log-structured\\_merge-tree](https://en.wikipedia.org/wiki/Log-structured_merge-tree)

# Standalone LevelDB Performance Benchmarks

**Keys:** 16 bytes each **Values:** 100 bytes each (50 bytes after compression) **Entries:** 1,000,000

## Write Performance

fillseq : 1.765 micros/op; 62.7 MB/s

fillsync : 268.409 micros/op; 0.4 MB/s

fillrandom : 2.460 micros/op; 45.0 MB/s (approx 400,000 writes per second)

overwrite : 2.380 micros/op; 46.5 MB/s

## Read Performance

readrandom : 16.677 micros/op; (approximately 60,000 reads per second)

readseq : 0.476 micros/op; 232.3 MB/s

readreverse : 0.724 micros/op; 152.9 MB/s

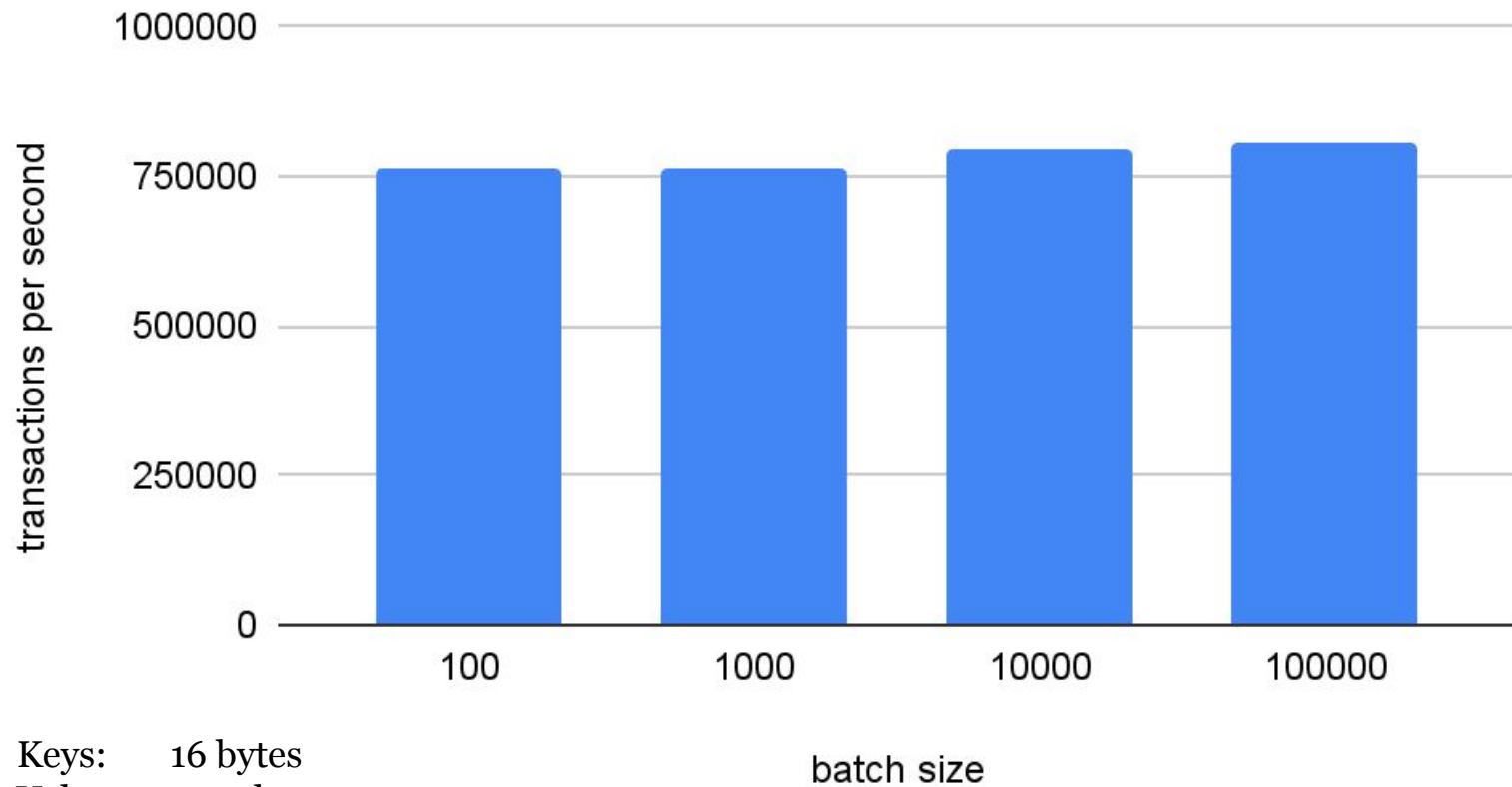
*After compactions:*

readrandom : 11.602 micros/op; (approximately 85,000 reads per second)

readseq : 0.423 micros/op; 261.8 MB/s

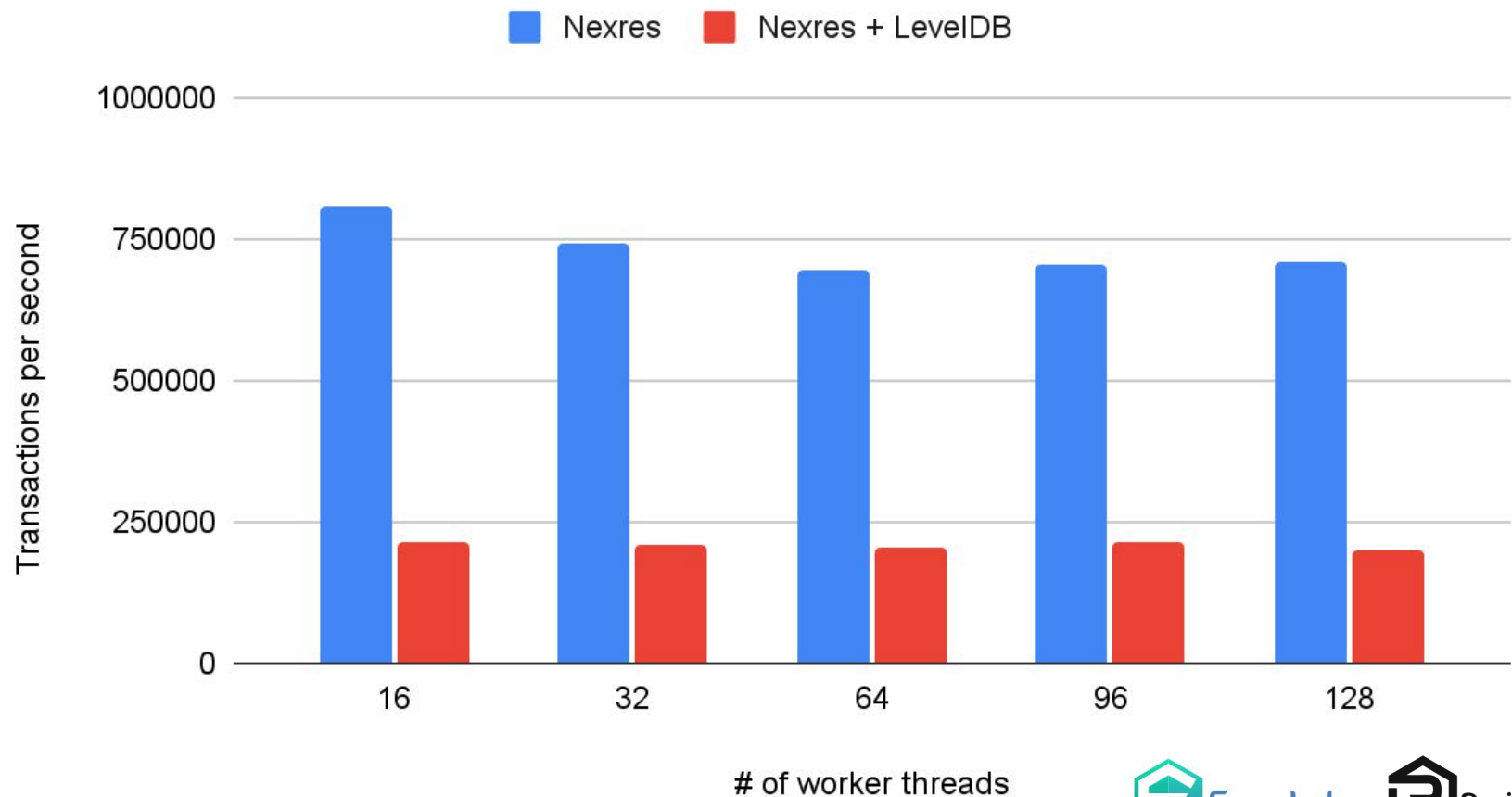
readreverse : 0.663 micros/op; 166.9 MB/s

# LevelDB Batched Writes



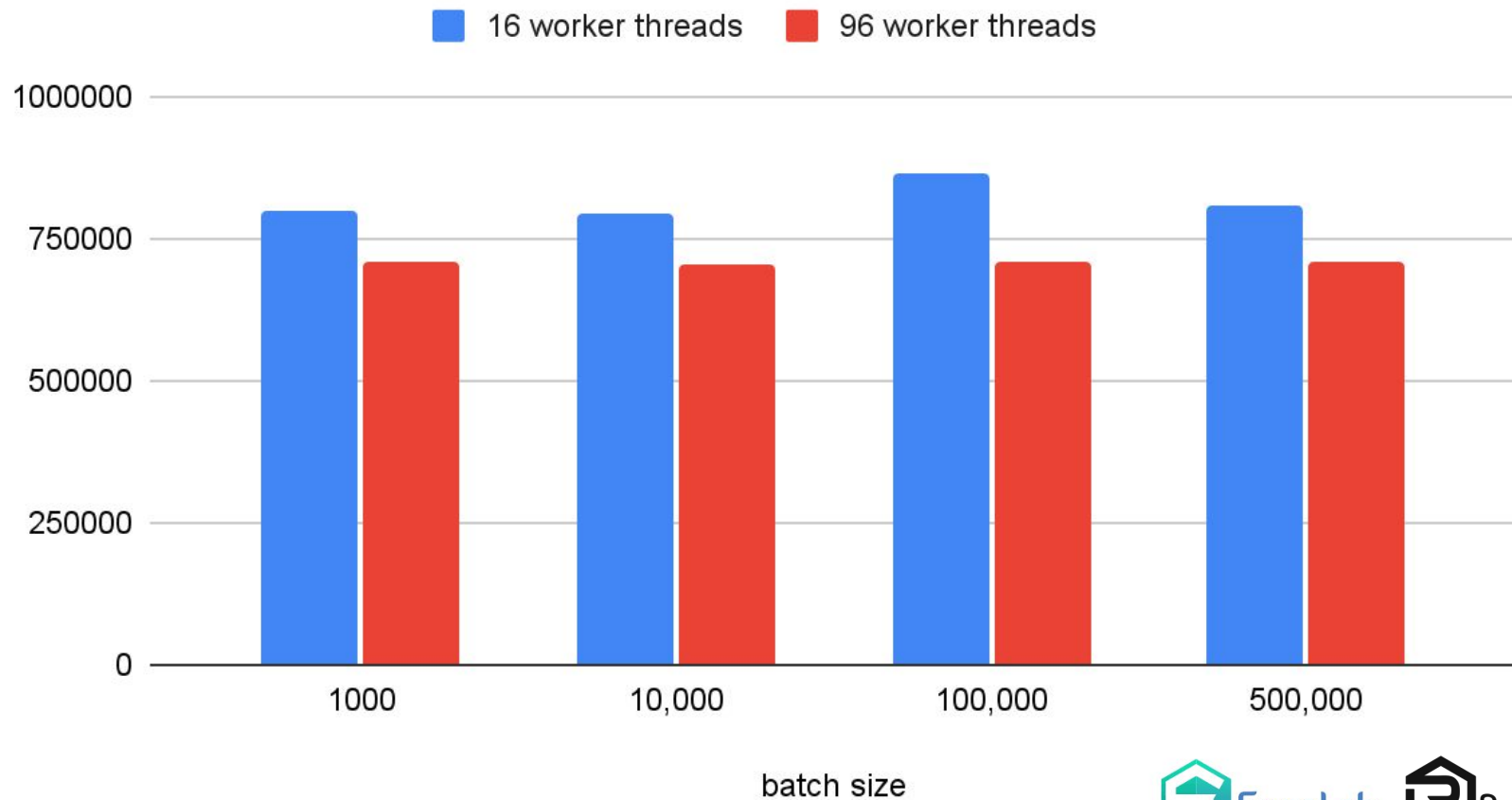
Keys: 16 bytes  
Values: 100 bytes  
Entries: 100,000,000

# Durability Enabled vs. Disabled Comparison





## Nexres + LevelDB batched writes





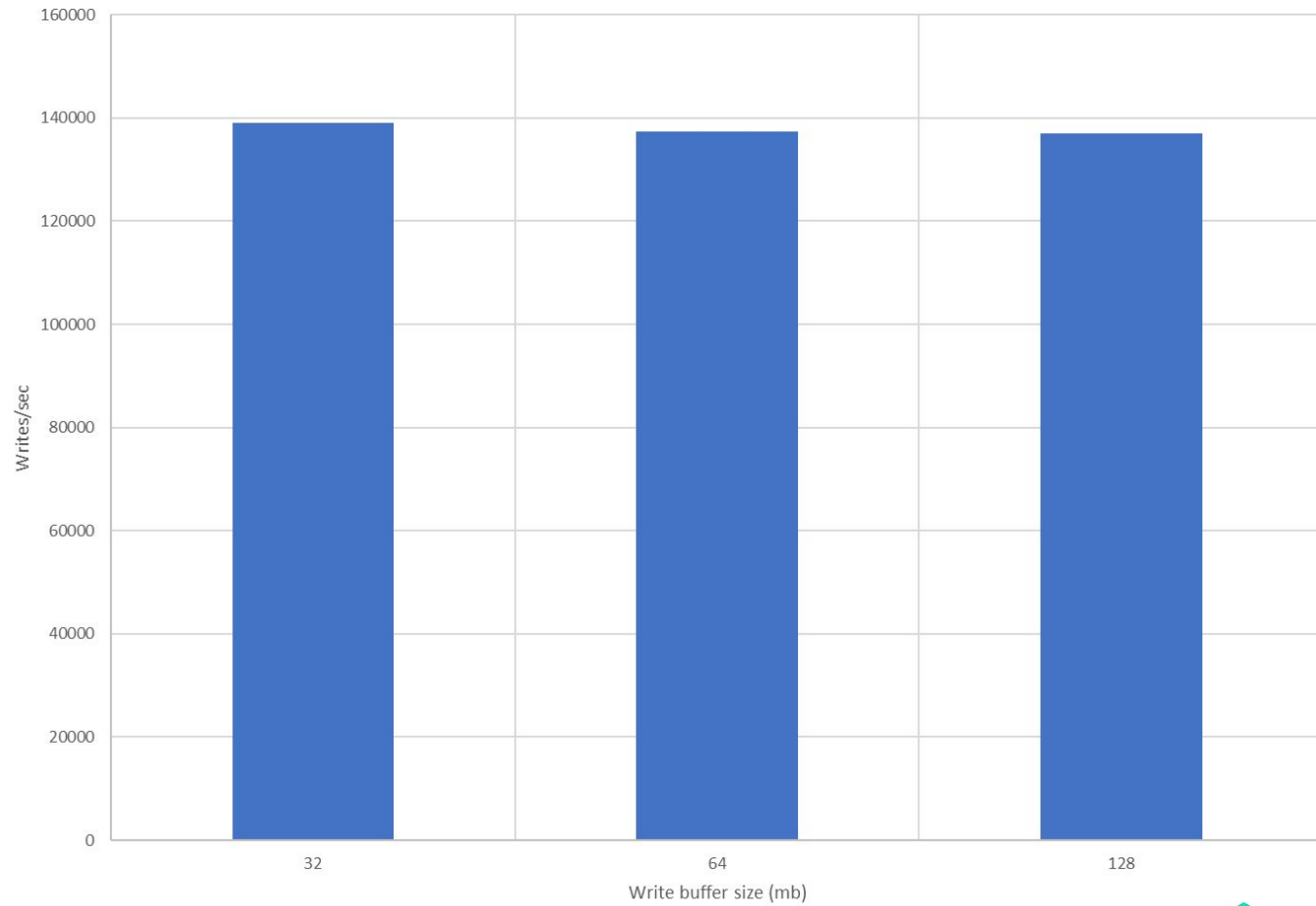
# RocksDB

- key-value storage library made by Meta
- optimized for storing small to medium size key-values on flash drives and RAM
- uses log-structured merge-trees
  - consists of a memtable and multiple levels of files on disk
- offers more features than LevelDB, including multiple compaction styles

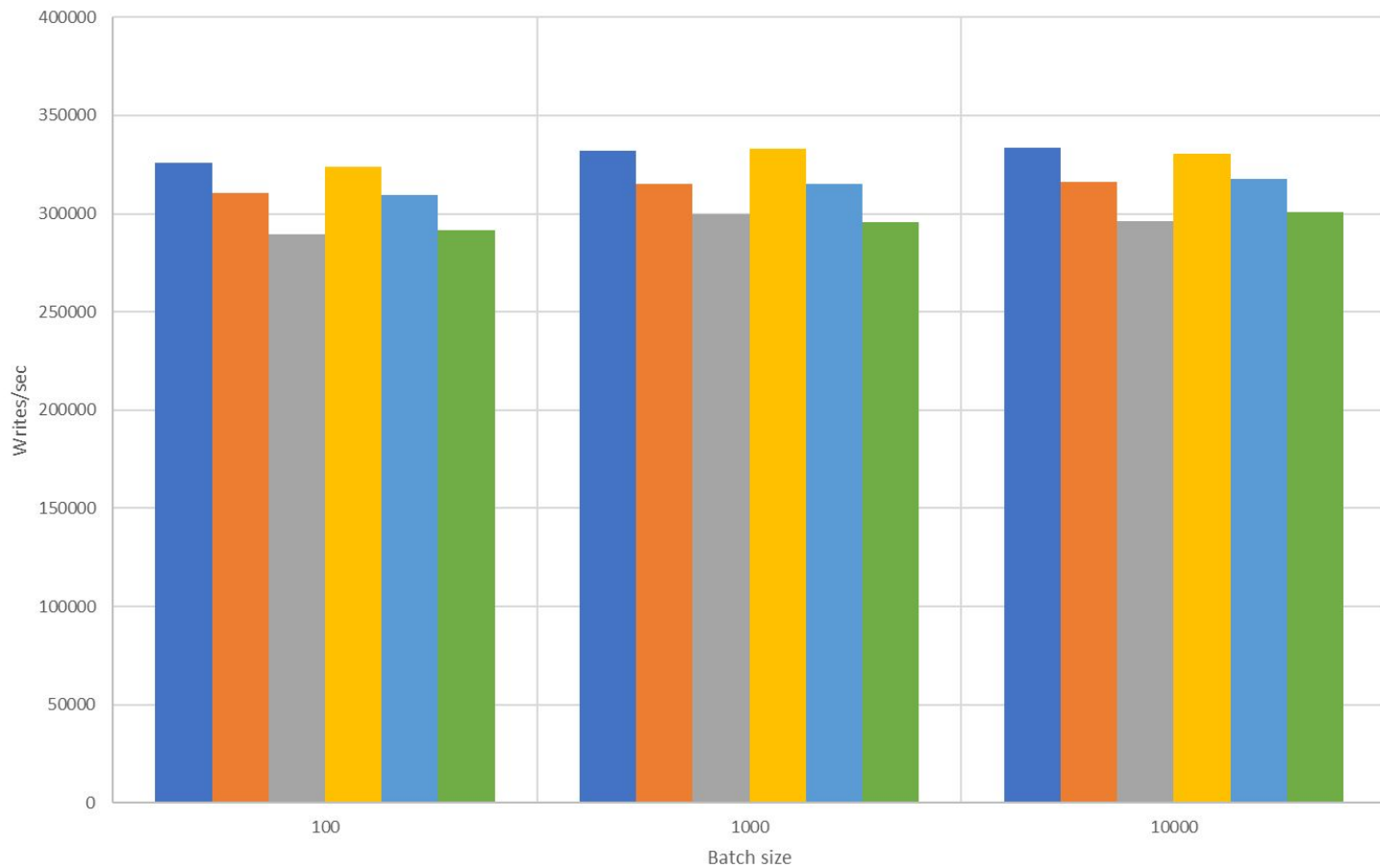
# RocksDB Performance Testing

- Key size = 2, value size = 10
  - same as what is used in the provided kv\_server\_performance tester
- 32 cores, 240 gb RAM
- 4 nodes + client for Nexres

Individual RocksDB Writes, 8 threads

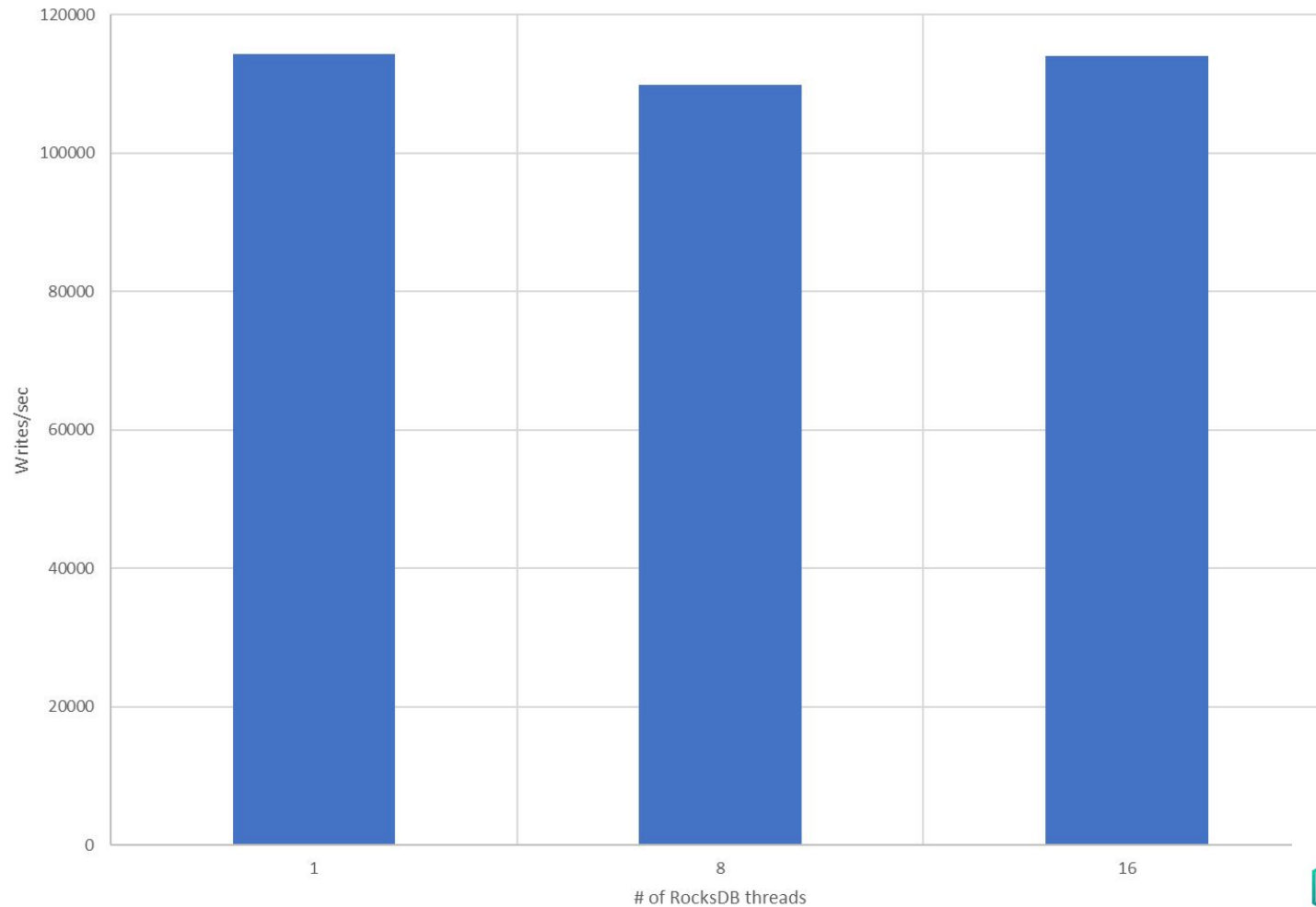


# Batched RocksDB Writes



■ write buffer 32, 8 rocksdb threads ■ write buffer 64, 8 rocksdb threads ■ write buffer 128, 8 rocksdb threads  
■ write buffer 32, 16 rocksdb threads ■ write buffer 64, 16 rocksdb threads ■ write buffer 128, 16 rocksdb threads

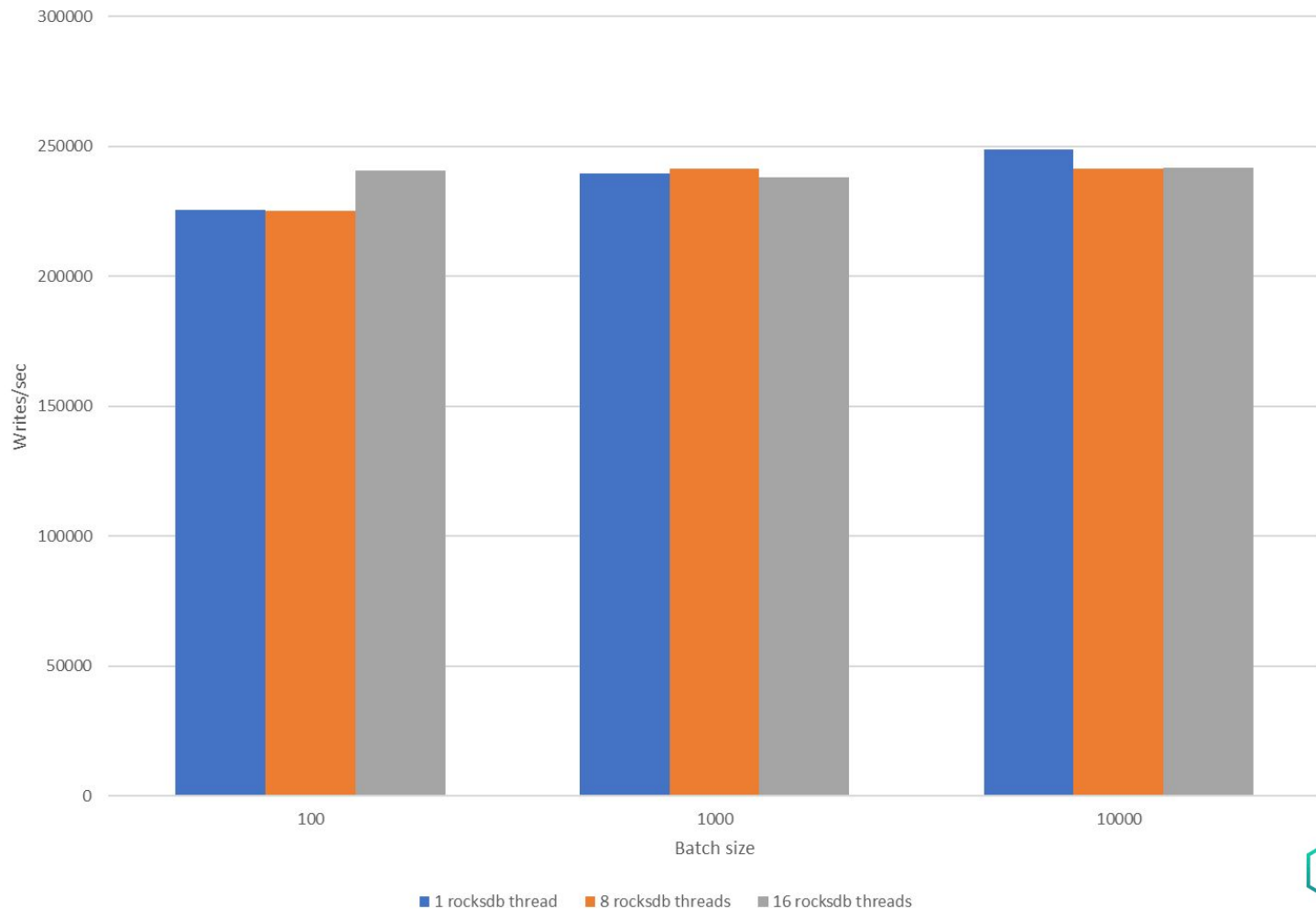
Nexres w/ Individual RocksDB Writes



32 MB write  
buffer size

16 Nexres  
worker threads

## Nexres w/ batched RocksDB writes



32 MB write  
buffer size

16 Nexres  
worker threads

# Using Durability Options in NexRes

- NexRes uses protobufs to specify format of configuration options
  - similar to JSON format
- rocksdb\_info and leveldb\_info are optional
  - if they are missing, their enable flags will be set to false by default
- detailed instructions in README of durability\_layer folder

Sample config settings that would be placed in a .config file

```
self_region_id:1,  
  rocksdb_info : {  
    enable_rocksdb:true,  
    num_threads:1,  
    write_buffer_size_mb:32,  
    write_batch_size:1,  
  },  
  leveldb_info : {  
    enable_leveldb:false,  
    write_buffer_size_mb:128,  
    write_batch_size:1,  
  },
```



# NexRes SDK

---

@aroy @juduarte @gjjchen

# Main goal of the sdk

- UTXO-based smart contracts
- Creation and Transfer of Assets

# Background

---

# Smart Contract

- Bitcoin script (Not Turing complete)
- Solidity (Turing complete)

# Pros and Cons of Turing Completeness

- Any Turing complete programming language create **loops**
- Loops creates a possibility of infinite loops
- Malicious actor can halt the network with endless operation

# Gas: Ethereum's Way of Avoiding Infinite Loops

- A fee you pay for any operation on ethereum
- once all the gas is used up, the network stops processing the contract

What is Crypto-Conditions  
Why do we need it

---

# Crypto-Conditions

- A mechanism that allows smart contracts to be developed on top of Bitcoin-protocol-based blockchains
- Basically makes building smart contracts easier for any blockchain
- <https://datatracker.ietf.org/doc/html/draft-thomas-crypto-conditions-04>



# Unspent Transaction Output (UTXO)

- Cryptocurrency transactions are made of **inputs** and **outputs**
- In a Tx, a user takes one or more UTXOs to serve as the input(s).
- The user provides their digital signature to confirm ownership over the inputs, which result in outputs
- The UTXOs consumed are now considered "spent," and can no longer be used
- Outputs from the transaction become new UTXOs
- Kind of similar to how cash works

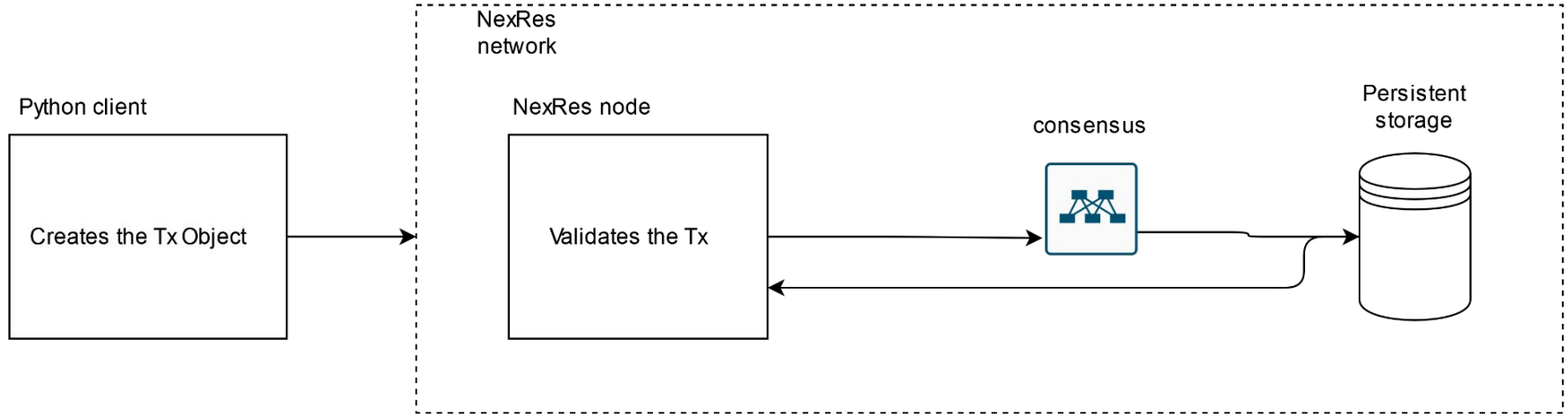
# UTXO-Based Smart Contracts

- Lock UTXO in a publicly-known address and prevent those UTXO from being spent until a certain set of conditions has been met
- conditions fulfilled → the UTXO are unlocked and sent to the appropriate address

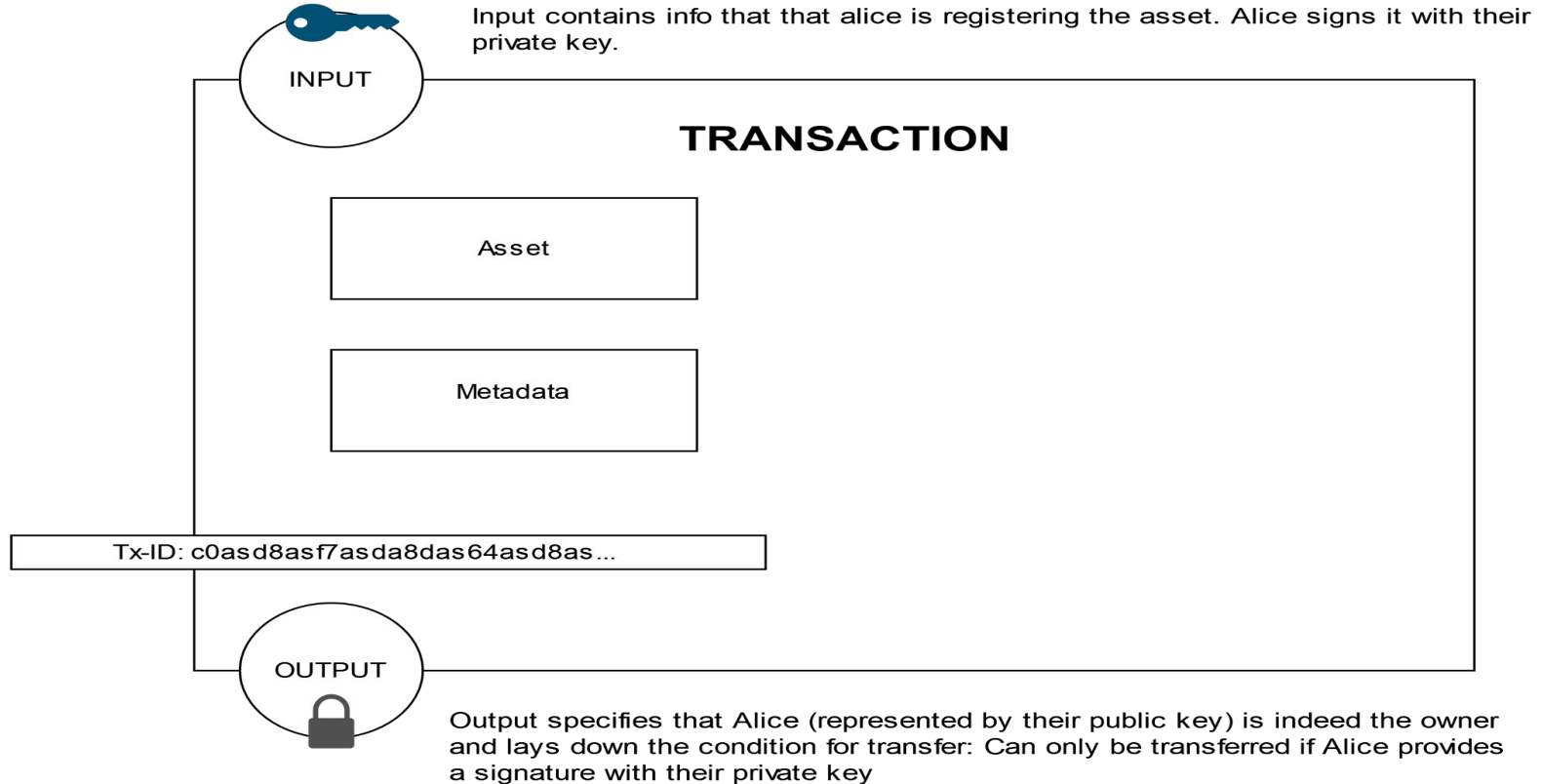
# High level overview of Tx

---

# Architecture



# Transaction per BEP-13 specification



# Endpoints

- `<ip>/v1/transactions/getAssets`
  - GET request
  - returns a list of all transactions stored in Nexres
- `<ip>/v1/transactions/getAssets/<txId>`
  - GET request
  - returns a transaction corresponding to a specific txId
- `<ip>/v1/transactions/commit`
  - POST request
  - accepts a JSON transaction object and writes it to Nexres
- Available on servers using either Crow or Pistache REST APIs

# Pistache vs Crow

HTTP and REST frameworks for C++

## Pistache

- Slower than Crow except on local environments with smaller requests
- Hasn't yet hit the 1.0 release
  - might be unstable but should still be production ready

## Crow

- Faster than Pistache in most situations, especially when clients and the server are on different machines
- Uses lambda expressions
  - may be slightly harder to learn if you're planning to modify the endpoints

# Demo

---

Creation and Transfer of assets