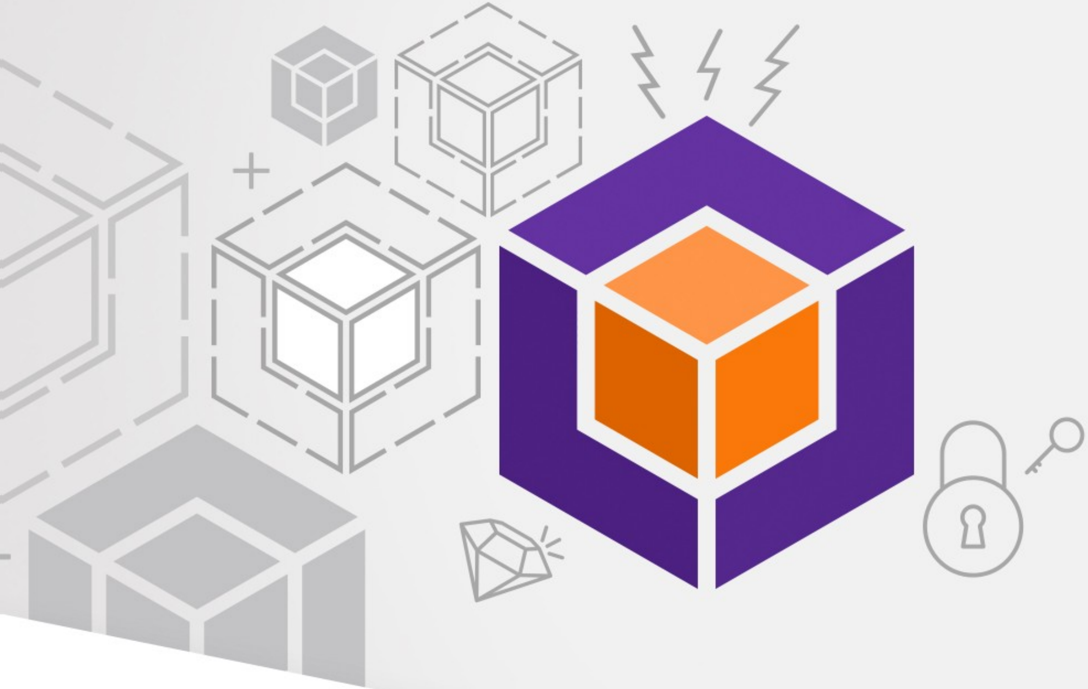
YottaDB



Rock Solid.
Lightning Fast.
Secure.

Pick any three

Outline

- What is it?
- Business Model
- Applications
- Ecosystem Examples
- Today’s Demos
- More Information

What is it?

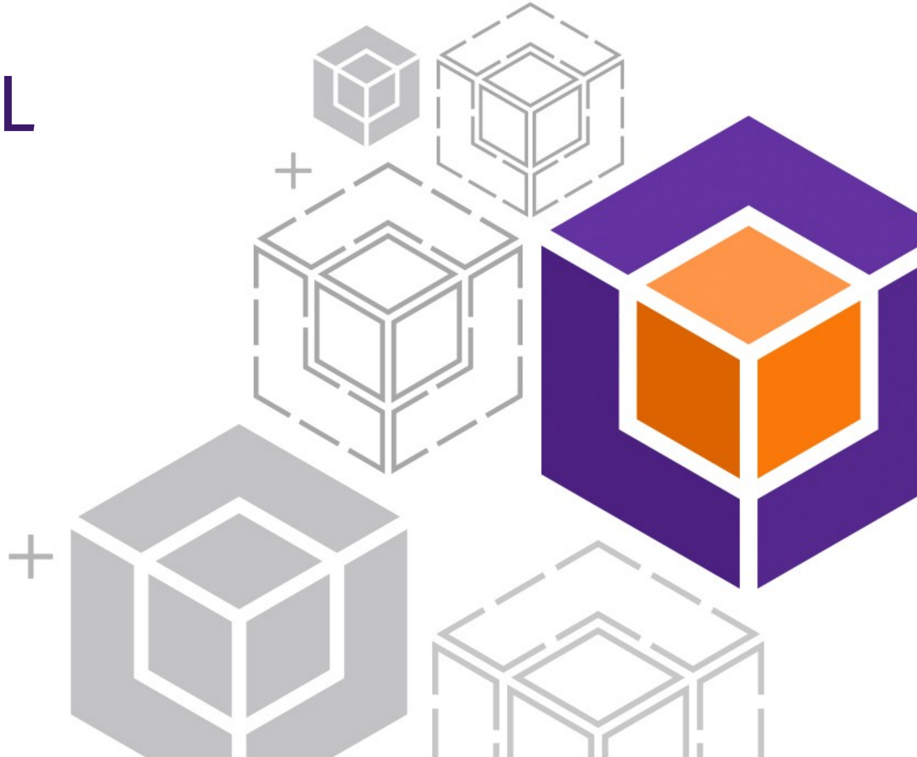
What is it?

- Hierarchical Key-value NoSQL
- Daemonless Database Engine
- Transaction Processing
- Optimistic Concurrency Control
- Data-centric Architecture
- Wrappers & Plugins
- Mission Critical Availability
- Platforms
- Octo

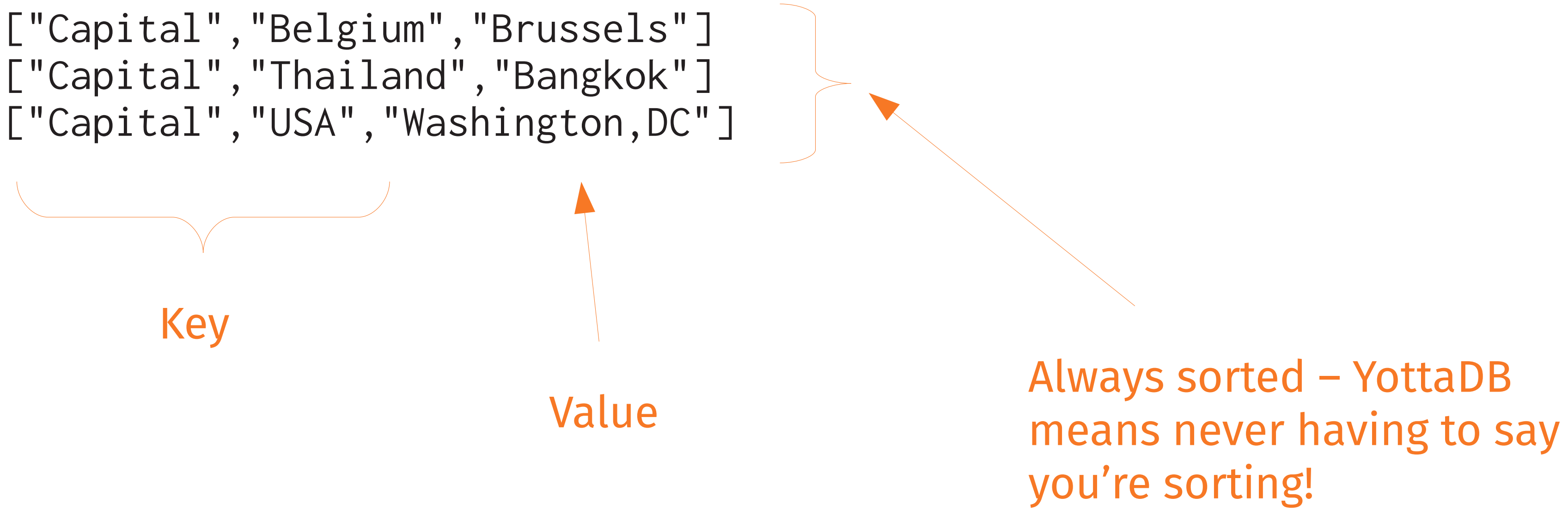
Core Database Technology

- Mature, high performance, hierarchical key-value, *language-agnostic*, NoSQL database whose code base scales up to mission-critical applications like large real-time core-banking and electronic health records, and also *scales down* to run on platforms like the Raspberry Pi Zero, as well as *everything in-between*.
- *Rock Solid. Lightning Fast. Secure. Pick any three.*

Hierarchical Key-Value NoSQL



Key-Value Tuples



```
["Capital","Belgium","Brussels"]
["Capital","Thailand","Bangkok"]
["Capital","USA","Washington,DC"]
```

Key

Value

Always sorted – YottaDB means never having to say you’re sorting!

Schemaless

```
["Capital","Belgium","Brussels"]
["Capital","Thailand","Bangkok"]
["Capital","USA","Washington,DC"]
["Population","Belgium",13670000]
["Population","Thailand",84140000]
["Population","USA",325737000]
```

Default order for each key:

- Empty string ("")
- Canonical numbers in numeric order
- Strings (blobs) in lexical order

Schema determined entirely by application – YottaDB assigns no meaning

Numbers and strings (blobs) can be freely intermixed in values and keys except first

Mixed Key Sizes

```
["Capital","Belgium","Brussels"]
["Capital","Thailand","Bangkok"]
["Capital","USA","Washington,DC"]
["Population","Belgium",13670000]
["Population","Thailand",84140000]
["Population","USA",325737000]
["Population","USA",17900802,3929326]
["Population","USA",18000804,5308483]
…
["Population","USA",20100401,308745538]
```

"Population" + 1 more key means value is latest population

"Population" + 2 more keys means value is population on date represented by last key

yyyymmdd

Keys, Array References, (Sub)Trees

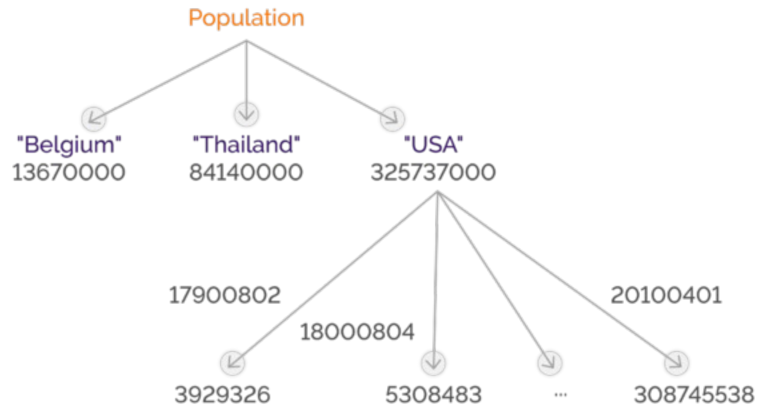
```
Population("Belgium")=13670000
Population("Thailand")=84140000
Population("USA")=325737000
Population("USA",17900802)=3929326
Population("USA",18000804)=5308483
…
Population("USA",20100401)=308745538
```

First key is variable name

Other keys are subscripts

Array references are a familiar programming paradigm

POPULATION TREE



Any JSON structure is representable as a tree, but not vice versa

Sharing and Persistence – Database Access

- Process private, available only for lifetime of process

```
Population("Belgium")
Population("Thailand")
Population("USA")
```

“local” variables

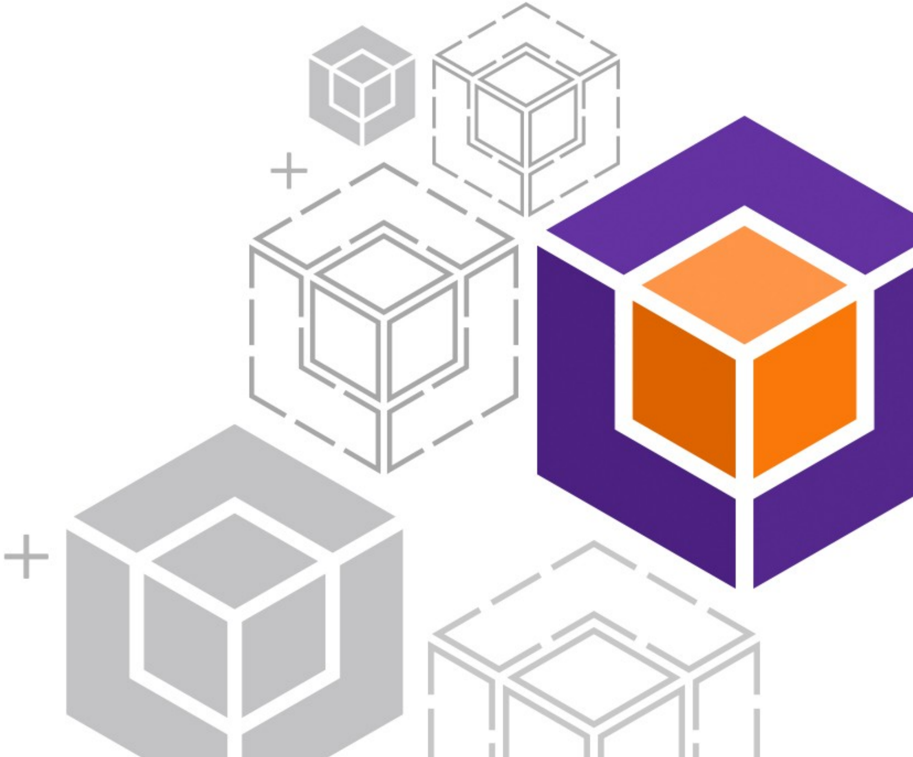
- Shared across processes, persistent beyond lifetime of any process

```
^Population("Belgium")
^Population("Thailand")
^Population("USA")
```

“global” variables

Spot the difference?

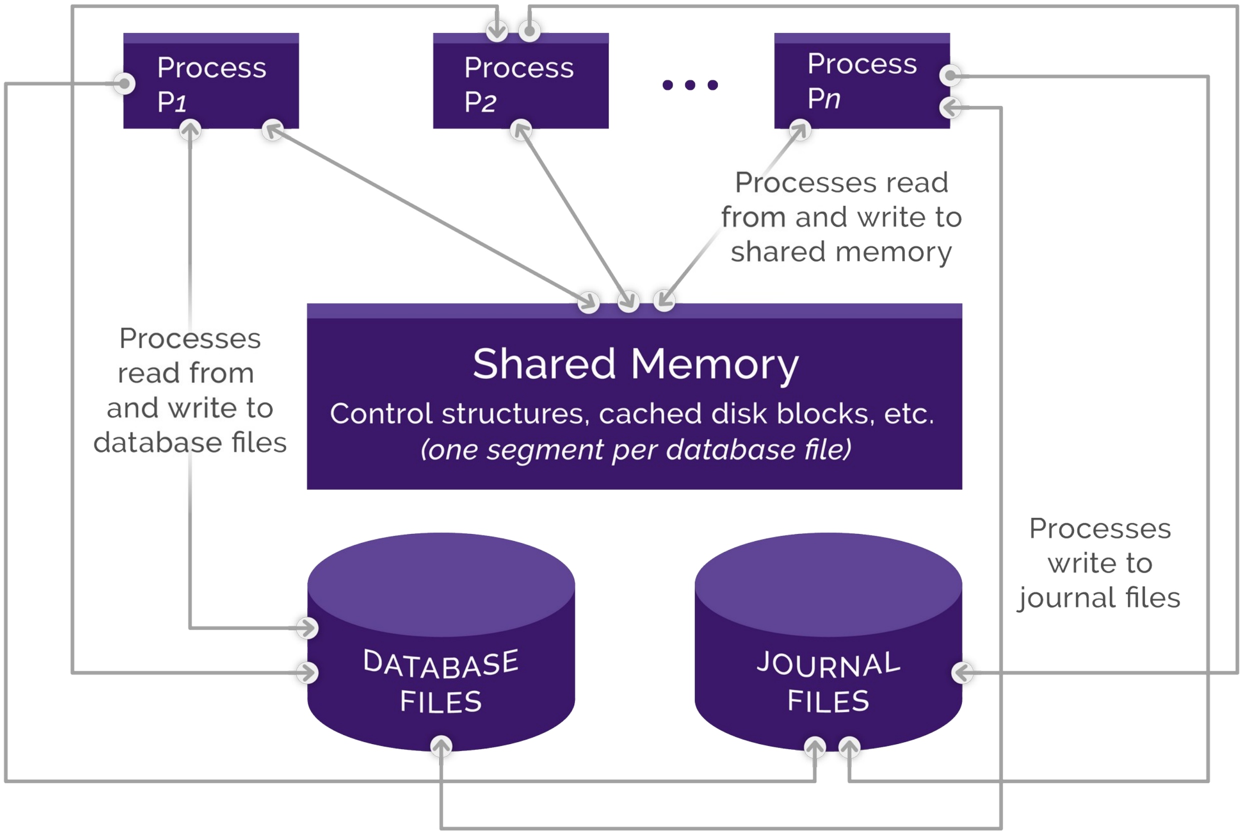
Daemonless Database Engine



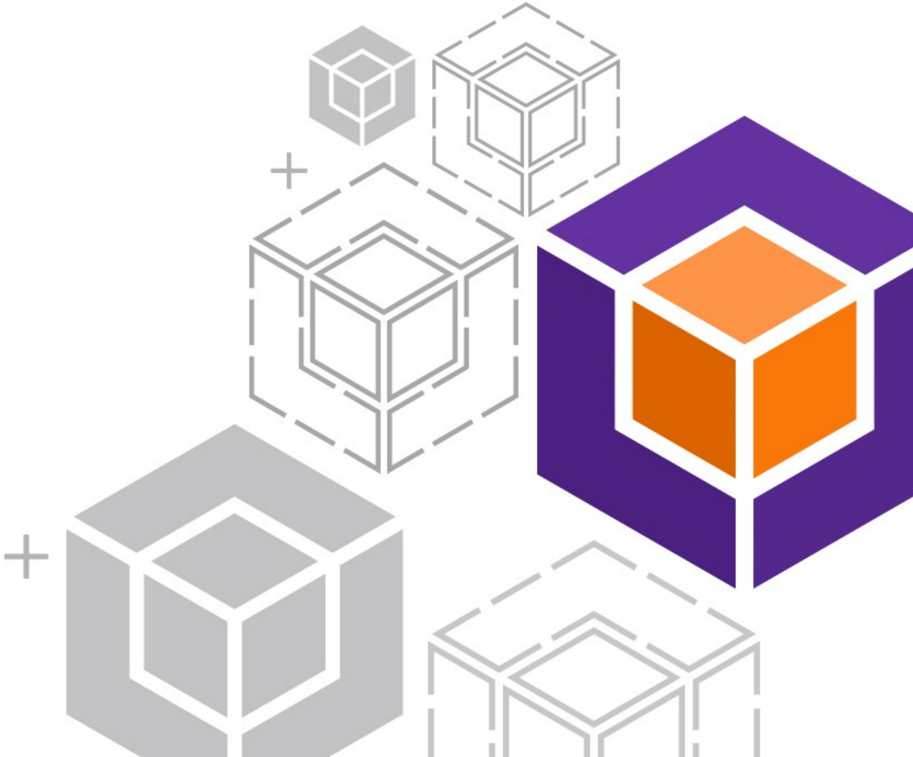
Daemonless Database Engine

- Database engine runs in application process
- Processes cooperate to manage database using shared memory control structures and buffers
- Simple security model
  - *Complexity is the enemy of security*
- No single point of failure
- Eliminates potential performance bottleneck

DAEMON-LESS DATABASE ENGINE


Process P1
Process P2
• • •
Process Pn
Processes read from and write to shared memory
Processes read from and write to database files
Shared Memory
Control structures, cached disk blocks, etc.
(one segment per database file)
DATABASE FILES
JOURNAL FILES
Processes write to journal files

Transaction Processing



Balance Transfer Example

- Validate that accounts exist, requested transfer permitted, sufficient balance, request authenticated
- Subtract $100 from savings account
- Add $100 to checking account
- Compute and debit any applicable service charges
- Log the transaction

Balance Transfer Example – Atomic

- Validate that accounts exist, requested transfer permitted, sufficient balance, request authenticated
- Subtract $100 from savings account
- Add $100 to checking account
- Compute and debit any applicable service charges
- Log the transaction

All of it happens or none of it happens

Balance Transfer Example – Consistent

YottaDB

- Validate that accounts exist, requested transfer permitted, sufficient balance, request authenticated
- Subtract $100 from savings account
- Add $100 to checking account
- Compute and debit any applicable service charges
- Log the transaction

No other process can see this intermediate state

Balance Transfer Example – Isolated

- Validate that accounts exist, requested transfer permitted, sufficient balance, request authenticated
- Subtract $100 from savings account
- Add $100 to checking account
- Compute and debit any applicable service charges
- Log the transaction

This logic sees no database state change except its own

Balance Transfer Example – Durable

- Validate that accounts exist, requested transfer permitted, sufficient balance, request authenticated
- Subtract $100 from savings account
- Add $100 to checking account
- Compute and debit any applicable service charges
- Log the transaction

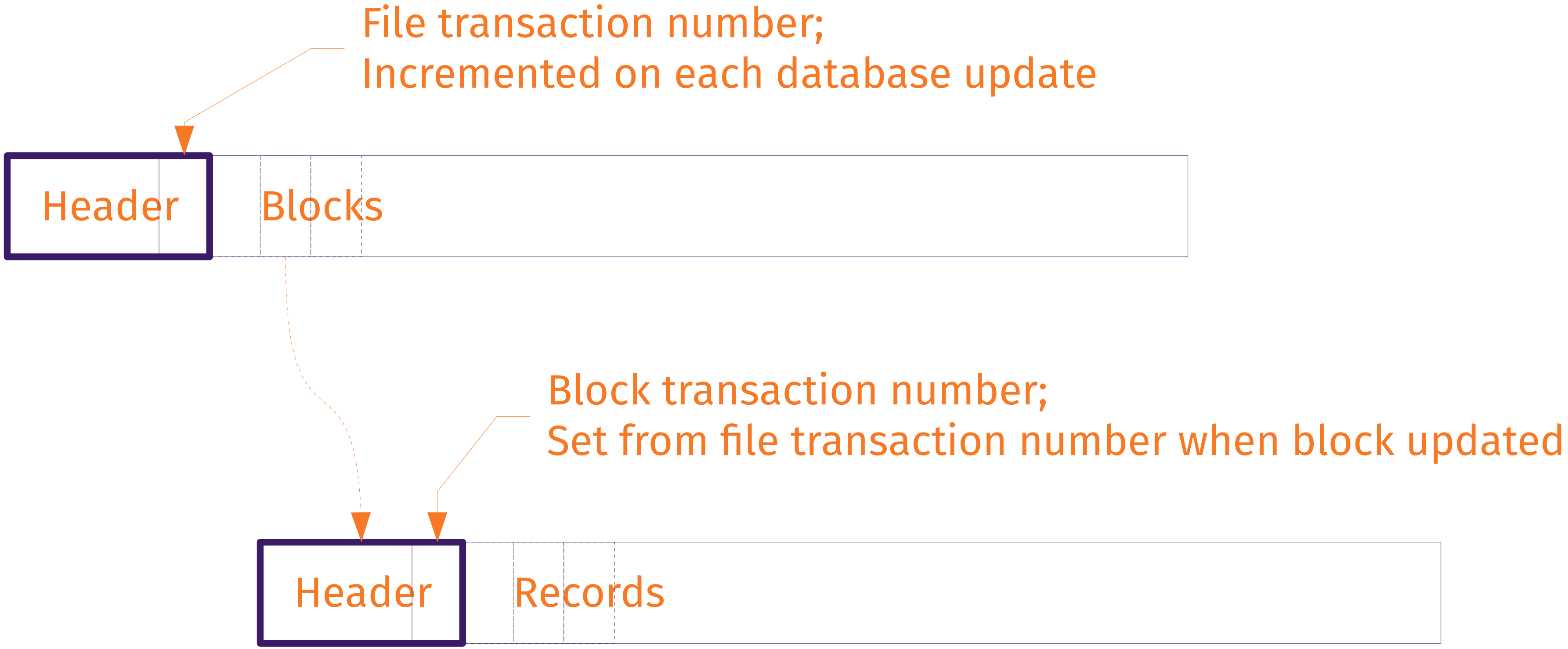
Committed

Permanent once committed

Optimistic Concurrency Control (OCC)

http://www.eecs.harvard.edu/~htk/publication/1981-tods-kung-robinson.pdf

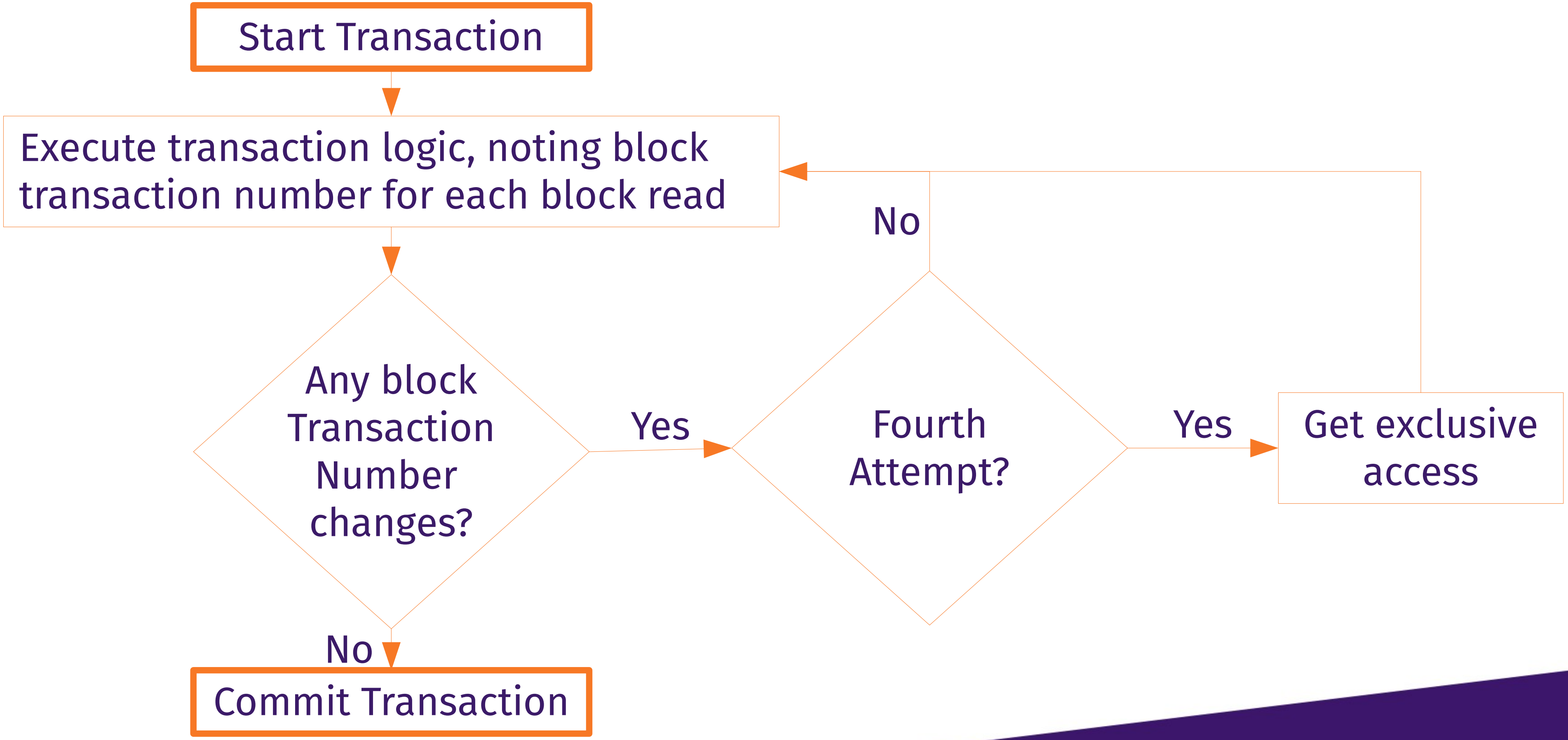
Transaction Numbers



File transaction number;
Incremented on each database update

Block transaction number;
Set from file transaction number when block updated

OCC Implementation



Start Transaction

Execute transaction logic, noting block transaction number for each block read

Any block Transaction Number changes?

Yes

Fourth Attempt?

Yes

Get exclusive access

No

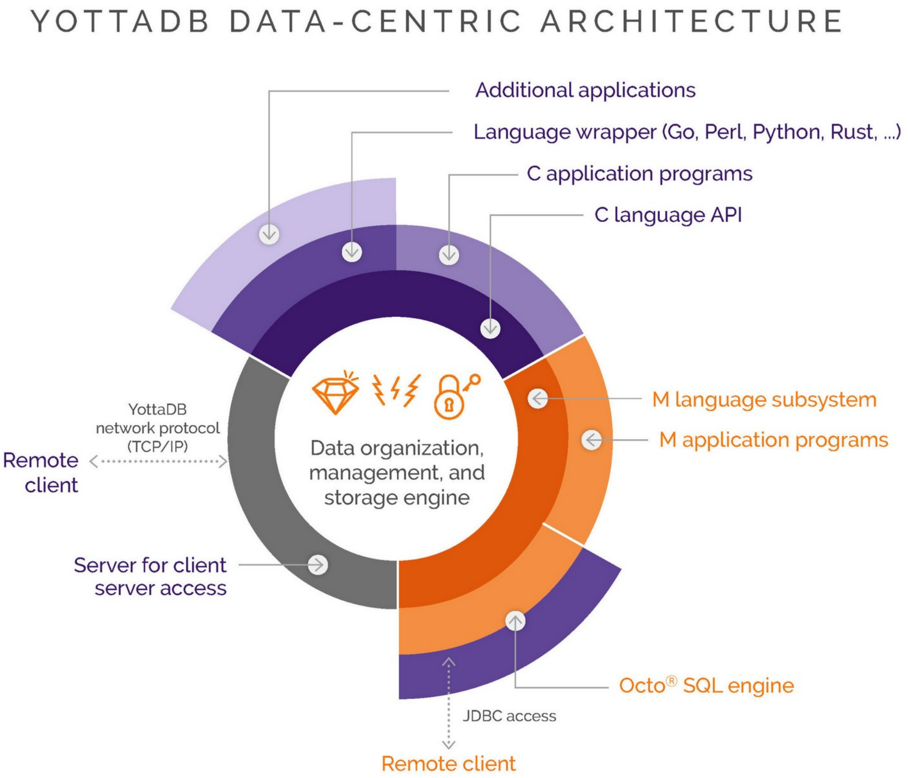
No

Commit Transaction

YottaDB OCC Benefits

- Scalabililty, throughput
- Strong ACID properties
  - e.g., even checking for absence of data

Data-Centric Architecture


YOTTADB DATA-CENTRIC ARCHITECTURE
Additional applications
Language wrapper (Go, Perl, Python, Rust, ...)
C application programs
C language API
M language subsystem
M application programs
Data organization, management, and storage engine
YottaDB network protocol (TCP/IP)
Remote client
Server for client server access
Octo® SQL engine
JDBC access
Remote client

Wrappers & Plugins

- Wrappers provide language APIs
  - “Native” to each language
  - Installed where expected by each implementation
- Plugins extend core functionality (e.g., Octo, AIM)

Hello world – C

```
#include "libyottadb.h"
int main()
{
     ydb_buffer_t  lang[1], value, varname;
     YDB_LITERAL_TO_BUFFER("^hello", &varname);
     YDB_LITERAL_TO_BUFFER("C", &lang[0]);
     YDB_LITERAL_TO_BUFFER("Hello, world!", &value)
     return ydb_set_s(&varname, 1, &lang[0], &value);
}
```

Hello world – Go

```go
package main
import (
    "lang.yottadb.com/go/yottadb"
)

func main() {

    defer yottadb.Exit()

    _ = yottadb.SetValE(yottadb.NOTTP, nil, "สวัสดีชาวโลก",
            "^hello", []string{"Go"})
}
```

Hello world – Lua

```lua
local ydb = require('yottadb')

ydb.set('^hello', {'Lua'}, 'Hallo Wereld')
```

Hello world – M

```
^hello("M")="Привіт Світ"
```

Hello world – Node.js

```
const ydb = require('nodem').Ydb();

ydb.open();

ydb.set('^hello', 'Node.js', 'مرحبا بالعالم');

ydb.close();
```

Hello world – Perl

```
#! /usr/bin/perl

use YottaDB qw(:all);

y_set "^hello", "Perl", "Grüß Gott Welt";
```

Hello world – Python

```
import yottadb

if __name__ == "__main__":
    yottadb.set("^hello", ("Python",), value="नमस्ते दुनिया")
```

Hello world – Rust

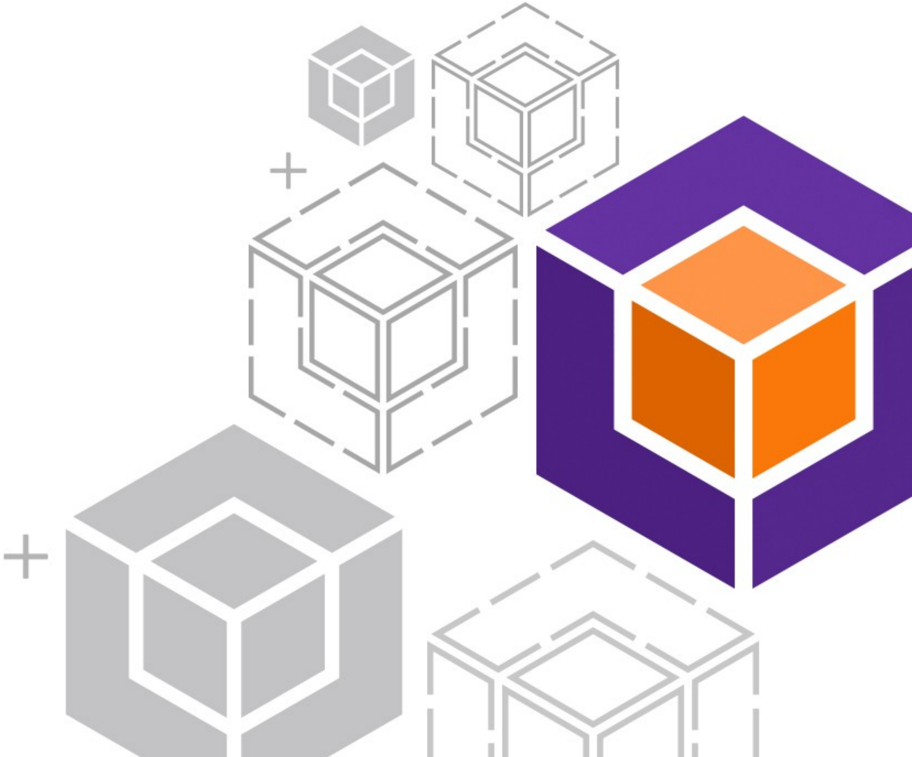
```
#![allow(non_snake_case)]


use yottadb::simple_api::Key;
use yottadb::craw::YDB_NOTTP;

fn main() {
    let err_buffer = Vec::new();
    let mut hello = Key::new("^hello", &["Rust"]);
    hello.set_st(YDB_NOTTP, err_buffer,
            "こんにちは世界".as_bytes()).unwrap();
}
```

Many Languages, One Database

```
$ mupip extract -format=zwr -select=hello -stdout
YottaDB MUPIP EXTRACT /usr/local/lib/yottadb/r134/mupip extract -format=zwr -select=hello -stdout
UTF-8
13-JUN-2022  11:38:17 ZWR
^hello("C")="Hello, world!"
^hello("Go")="สวัสดีชาวโลก"
^hello("Lua")="Hallo Wereld"
^hello("M")="Привіт Світ"
^hello("Node.js")="مرحبا بالعالم"
^hello("Perl")="Grüß Gott Welt"
^hello("Python")="नमस्ते दुनिया"
^hello("Rust")="こんにちは世界"
%YDB-I-RECORDSTAT, ^hello:        Key cnt: 8  max subsc len: 16  max rec len: 37  max node len: 48
%YDB-I-RECORDSTAT, TOTAL:         Key cnt: 8  max subsc len: 16  max rec len: 37  max node len: 48
$
```

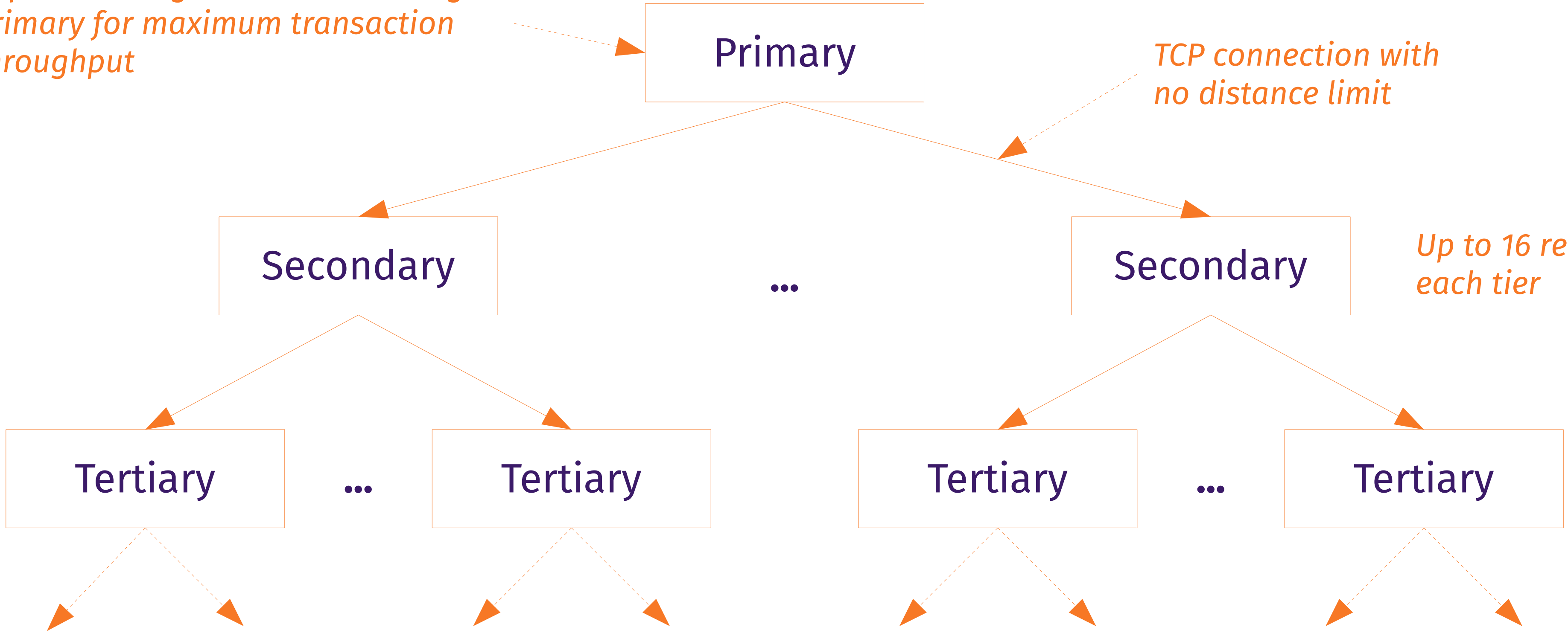
Mission Critical Availability

“Five nines”



Replication

Application logic executes on single primary for maximum transaction throughput



TCP connection with no distance limit

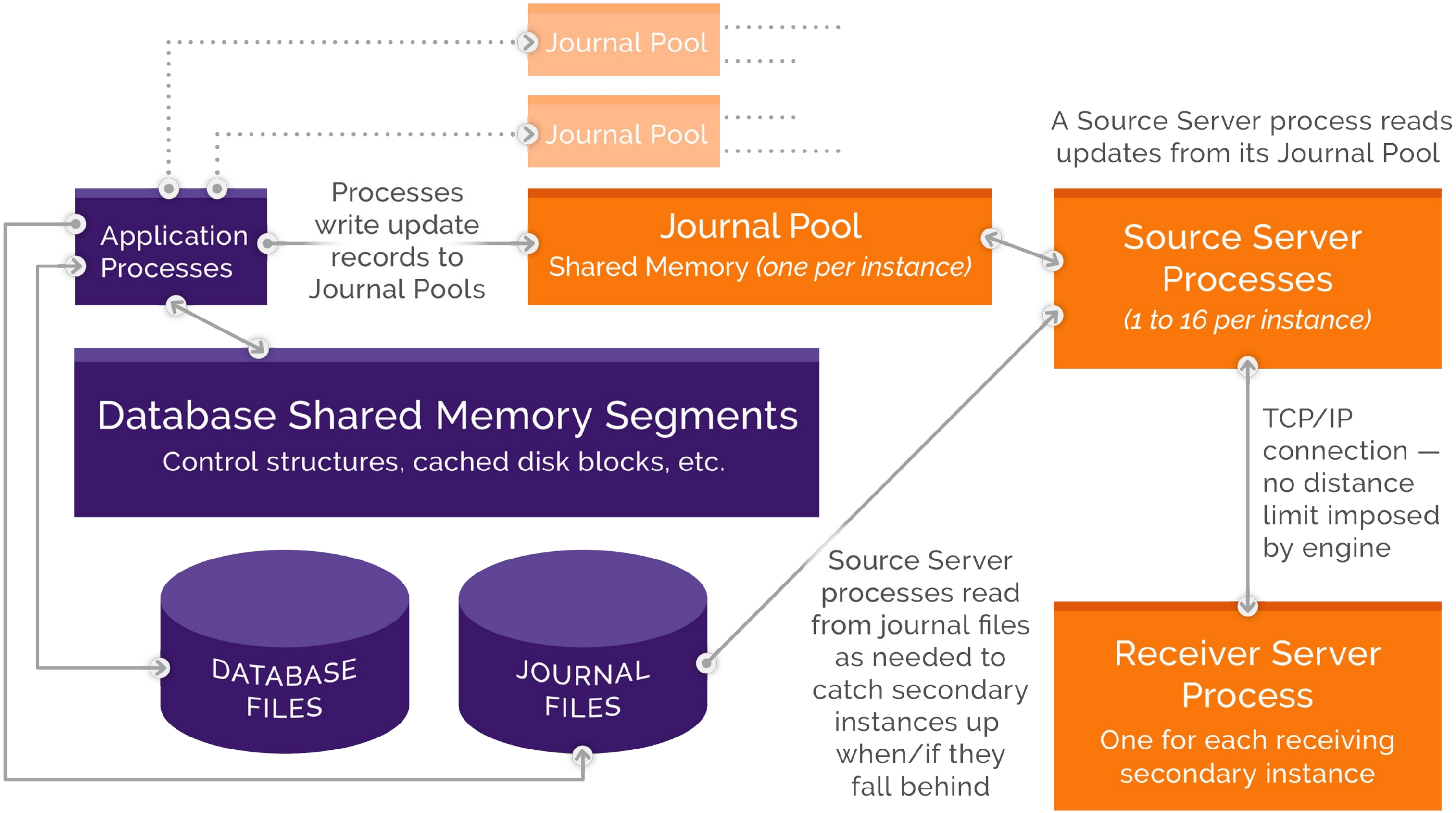
Up to 16 replicas each tier

No limit on number of tiers

How Replication Works

- All business logic on single primary instance to maximize ACID transaction serialization throughput
- Asynchronous *Logical* replication upto 16 instances
  - Minimizes network bandwidth usage
  - AP system per CAP Theorem
- Tools to help application restore eventual (CAP) Consistency while maintaining (ACID) Consistency

DATABASE REPLICATION



Journal Pool
Journal Pool
A Source Server process reads updates from its Journal Pool
Application Processes
Processes write update records to Journal Pools
Journal Pool
Shared Memory *(one per instance)*
Source Server Processes
(1 to 16 per instance)
Database Shared Memory Segments
Control structures, cached disk blocks, etc.
TCP/IP connection — no distance limit imposed by engine
DATABASE FILES
JOURNAL FILES
Source Server processes read from journal files as needed to catch secondary instances up when/if they fall behind
Receiver Server Process
One for each receiving secondary instance

Near-Zero Down Time Switchover

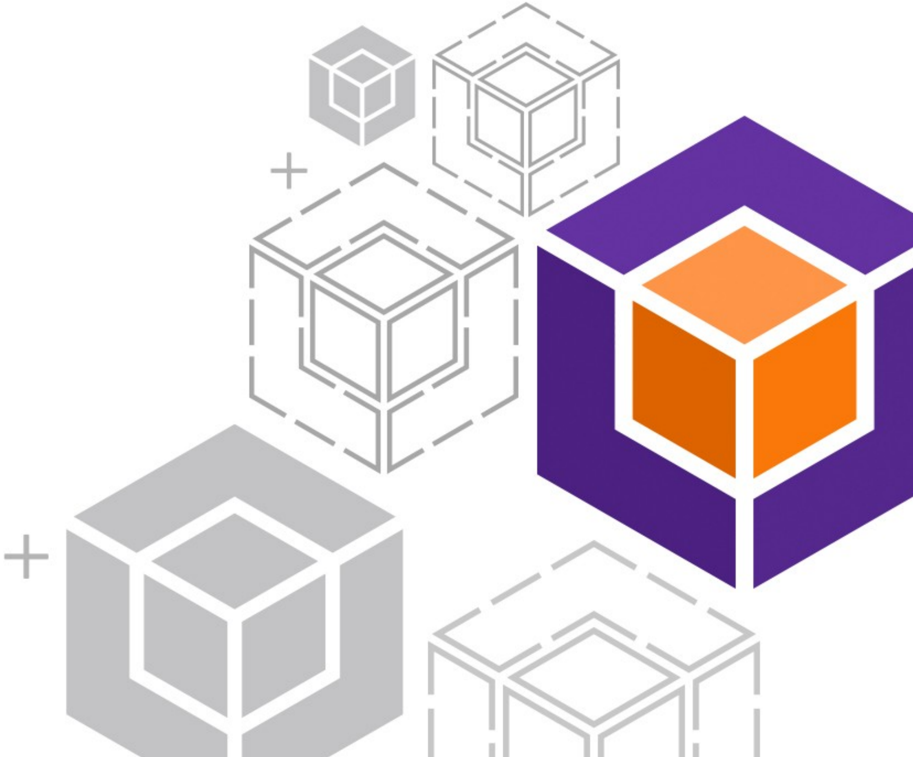
State / Event	Instance A	Instance B
Normal Operation (e..g, network latency backlog)	P:100	S:98
A goes down; B switched to primary role	X	P:98
B keeps application available; A serviced	X	P:98→120
A comes up as secondary; rolls back 2 transactions that are sent to B	S:100→98	P:120→125
A catches up as B operates as primary	S:98→130	P:125→130
B reprocesses rolled back transactions	S:130→132	P:130→132
Normal operation	S:132→140	P:132→140

Near-Zero Down Time Rolling Upgrade

State / Event	Instance A	Instance B
Normal Operation	P:100	S:100
A goes down; B switched to primary role	X	P:100
B keeps application available; A upgraded to A^+	X	P:100→120
A^+ comes up as secondary[†]; catches up to B	S:100→125	P:120→125
Switch A to primary role[†]; upgrade B	P:125	X
A^+ keeps application available; B upgraded to B^+	P:125→140	X
B^+ comes up as secondary; catches up to A^+	P:140→150	P:125→150

[†] Optional replication schema change filters to maintain application availability even when upgrade involves schema change

Platforms



Supported & Supportable Platforms

	x86_64	AARCH64 (ARM v8)	ARM-HF (ARM v7)
Debian	✓	✓	✓
Ubuntu	✓		
RHEL	✓		
SUSE	✓		

Supportable Platforms

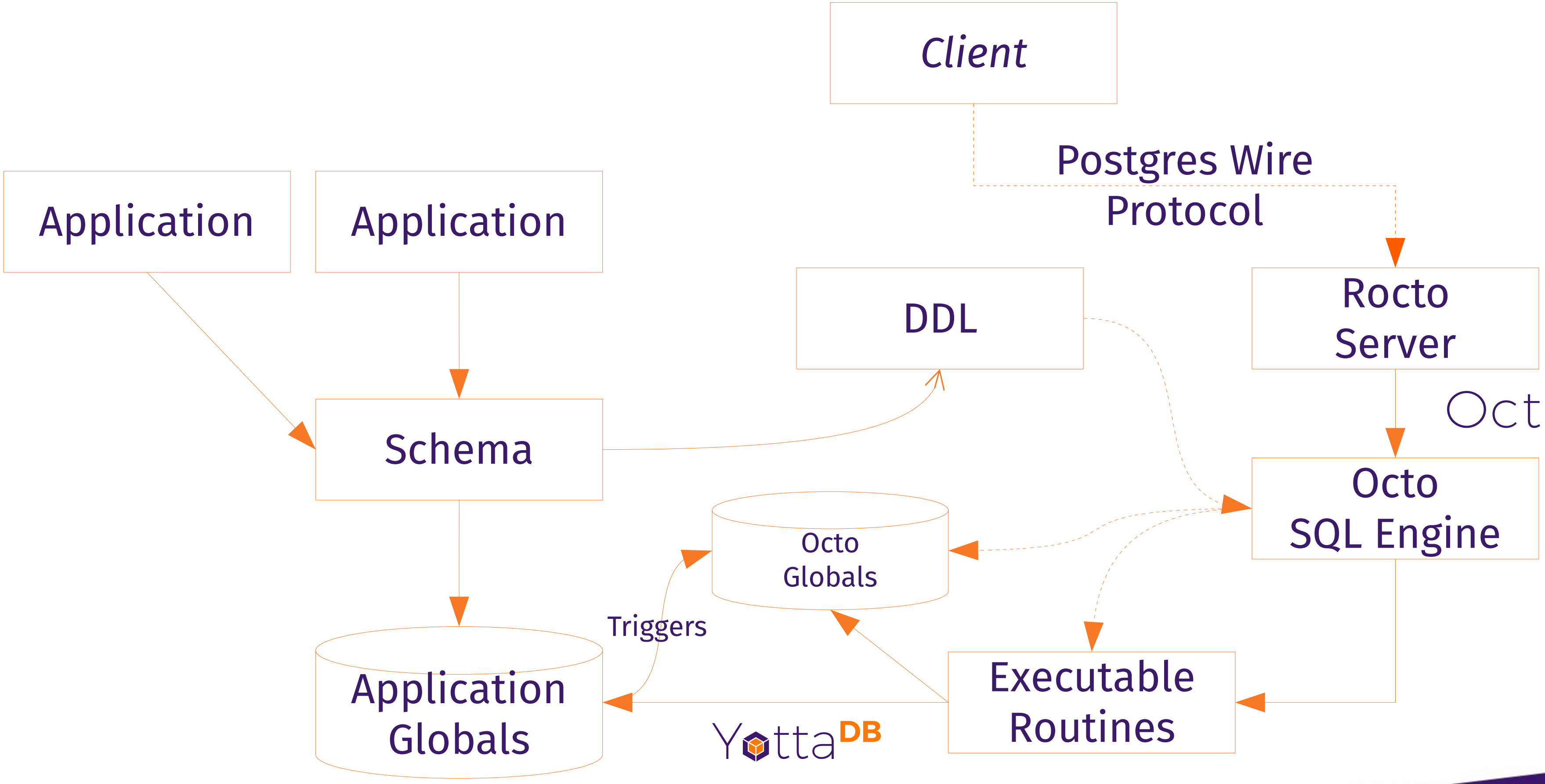
- Debian derivatives: All CPU architectures
- RHEL & SUSE derivatives and other: x86_64
- Build from Source: All CPU architectures on contemporary Linux distributions

Octo – SQL too

- Octo is a SQL database engine whose tables are mapped to YottaDB hierarchical key-value nodes
- Octo runs on YottaDB on 64-bit platforms

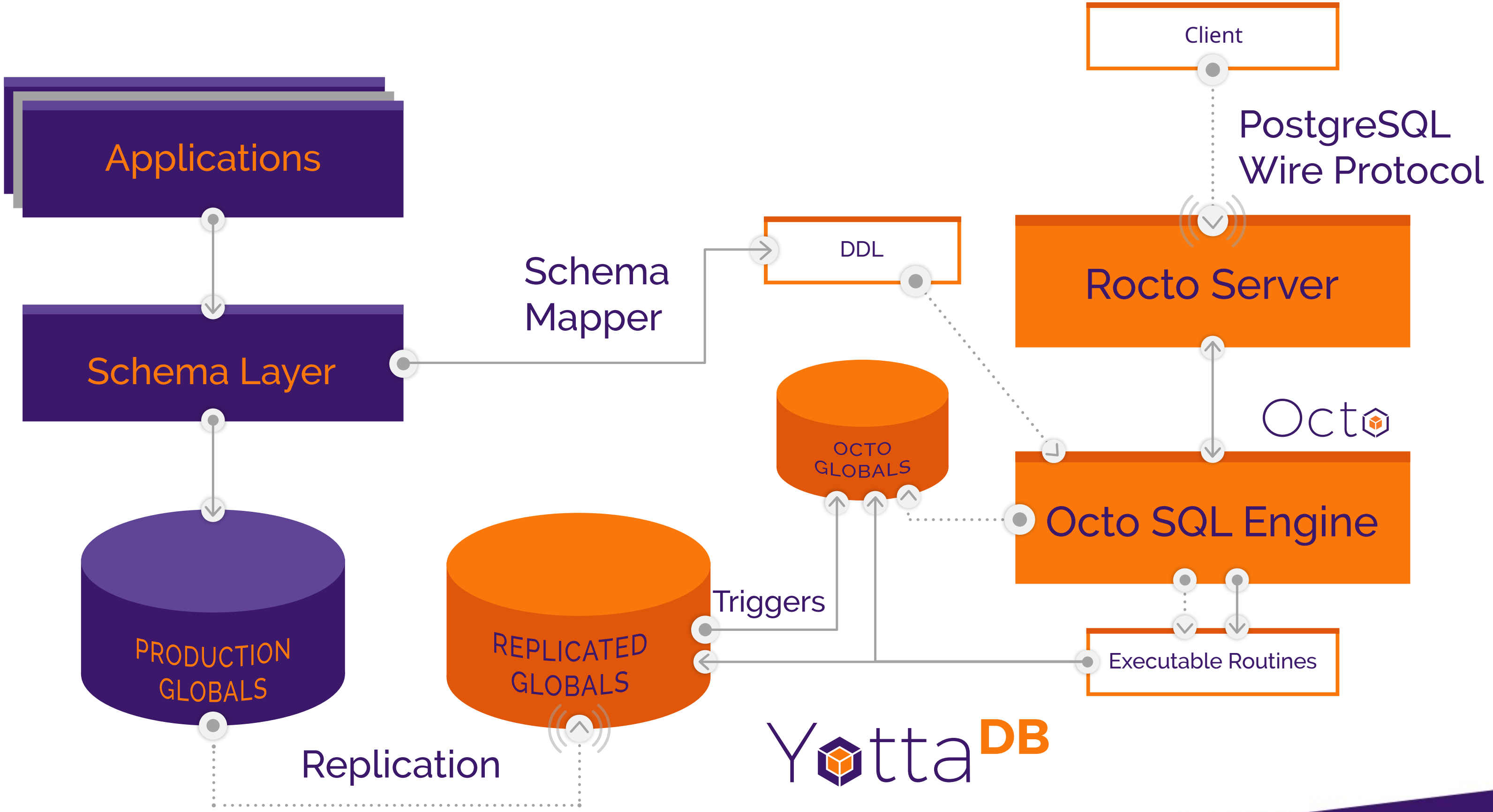
Octo is a registered trademark of YottaDB LLC

Octo Architecture

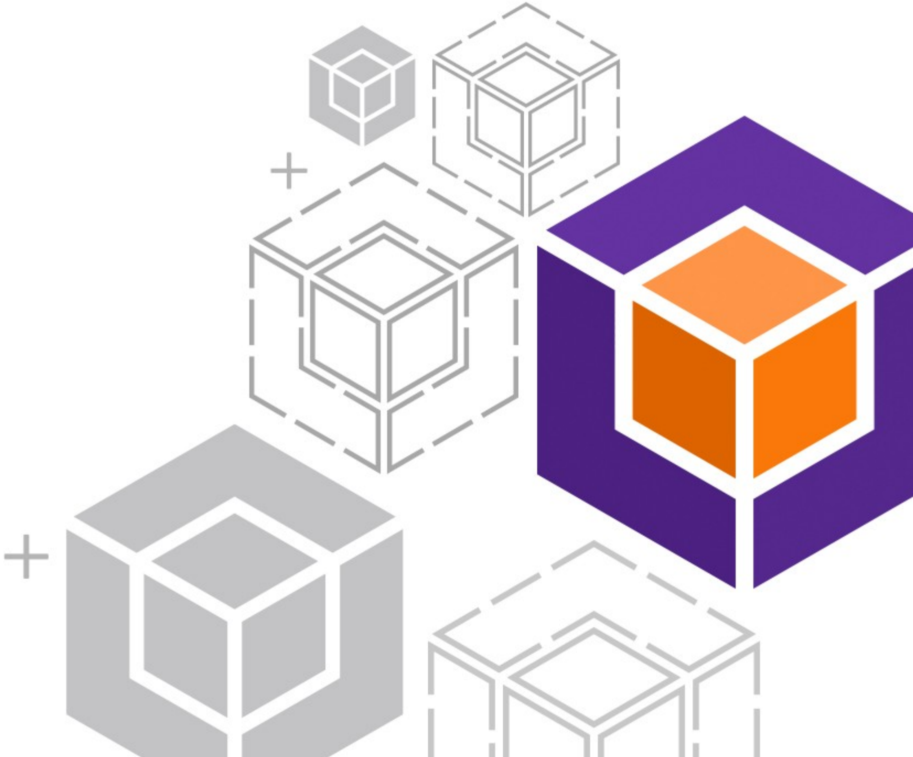


Global = Key-value node

Octo – Query / Analytics Example



Business Model

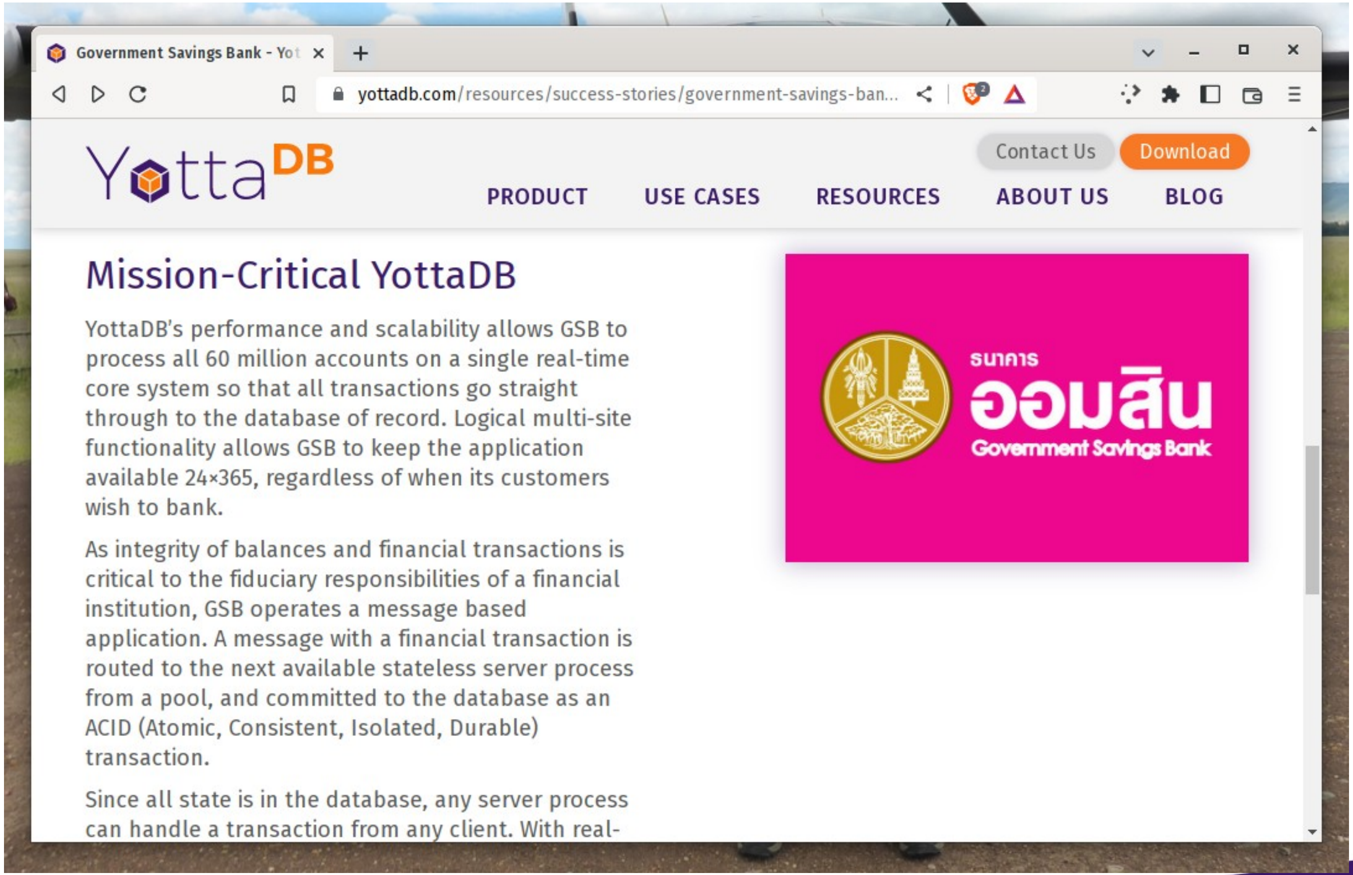


Support Contracts Fund Development

- Software is free
  - *100% Free / Open Source Software (FOSS)*
  - All work at https://gitlab.com/YottaDB
- People are not free
  - Support services with SLAs on commercial terms
  - Support options and tiers, including 24×7 support
- Worldwide support from USA

Applications

Real Time Large Scale Banking



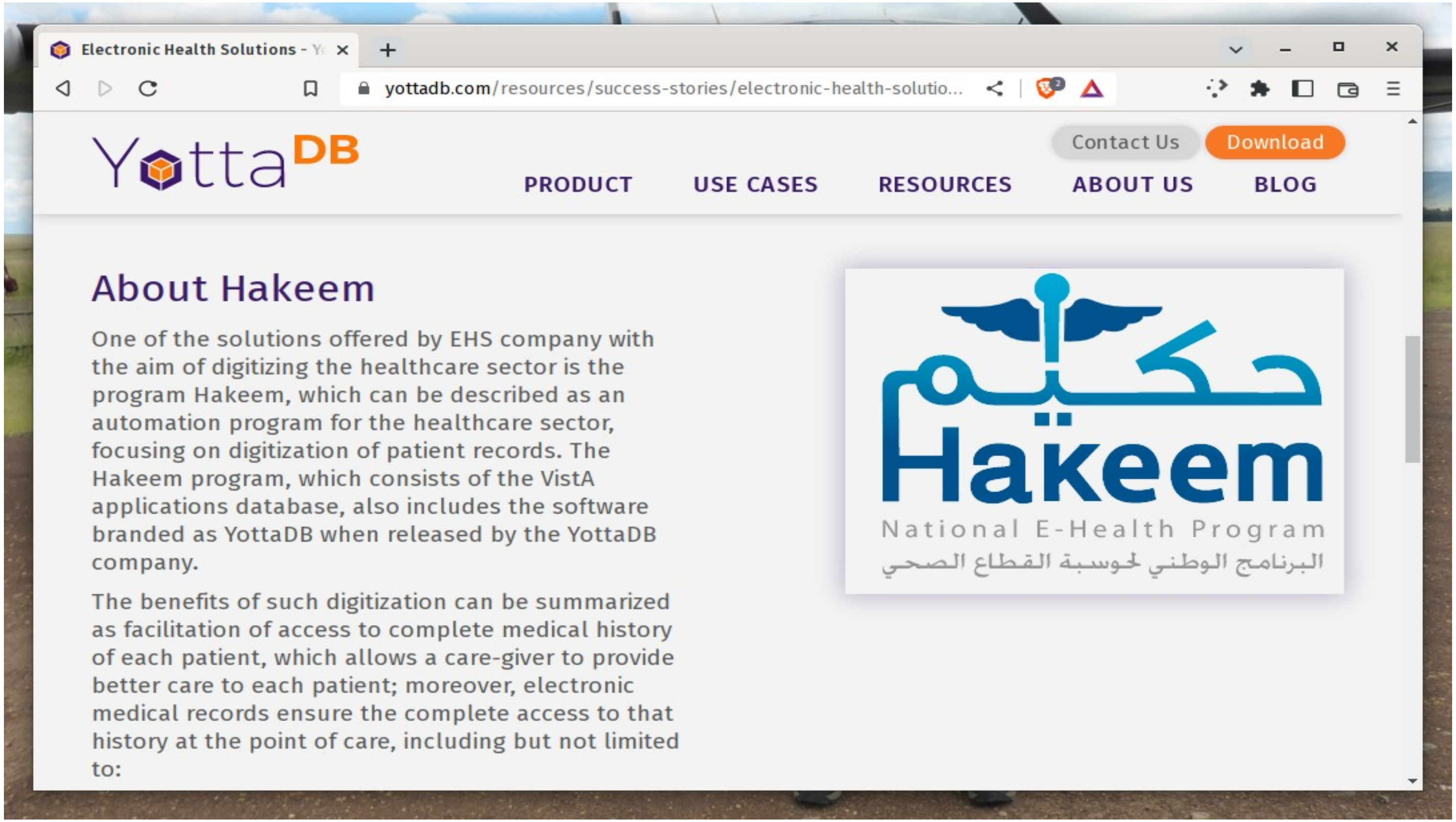
Mission-Critical YottaDB

YottaDB's performance and scalability allows GSB to process all 60 million accounts on a single real-time core system so that all transactions go straight through to the database of record. Logical multi-site functionality allows GSB to keep the application available 24×365, regardless of when its customers wish to bank.

As integrity of balances and financial transactions is critical to the fiduciary responsibilities of a financial institution, GSB operates a message based application. A message with a financial transaction is routed to the next available stateless server process from a pool, and committed to the database as an ACID (Atomic, Consistent, Isolated, Durable) transaction.

Since all state is in the database, any server process can handle a transaction from any client. With real-

Nation-scale Electronic Health Records

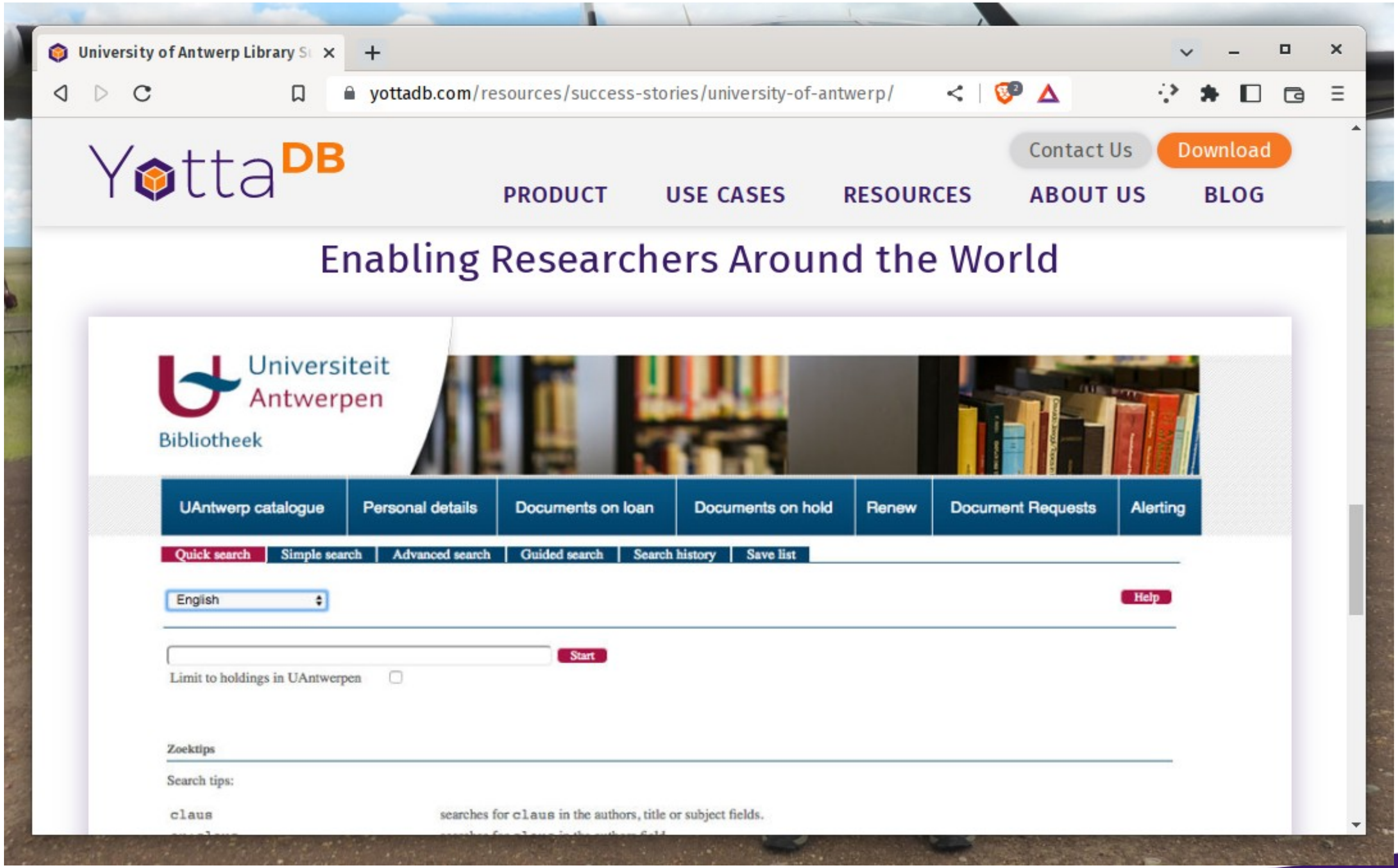

Electronic Health Solutions - Y
yottadb.com/resources/success-stories/electronic-health-solutio...
Contact Us
Download
PRODUCT
USE CASES
RESOURCES
ABOUT US
BLOG
حكيم
Hakeem
National E-Health Program
البرنامج الوطني لحوسبة القطاع الصحي

About Hakeem

One of the solutions offered by EHS company with the aim of digitizing the healthcare sector is the program Hakeem, which can be described as an automation program for the healthcare sector, focusing on digitization of patient records. The Hakeem program, which consists of the VistA applications database, also includes the software branded as YottaDB when released by the YottaDB company.

The benefits of such digitization can be summarized as facilitation of access to complete medical history of each patient, which allows a care-giver to provide better care to each patient; moreover, electronic medical records ensure the complete access to that history at the point of care, including but not limited to:

Major University Library Catalog



Enabling Researchers Around the World

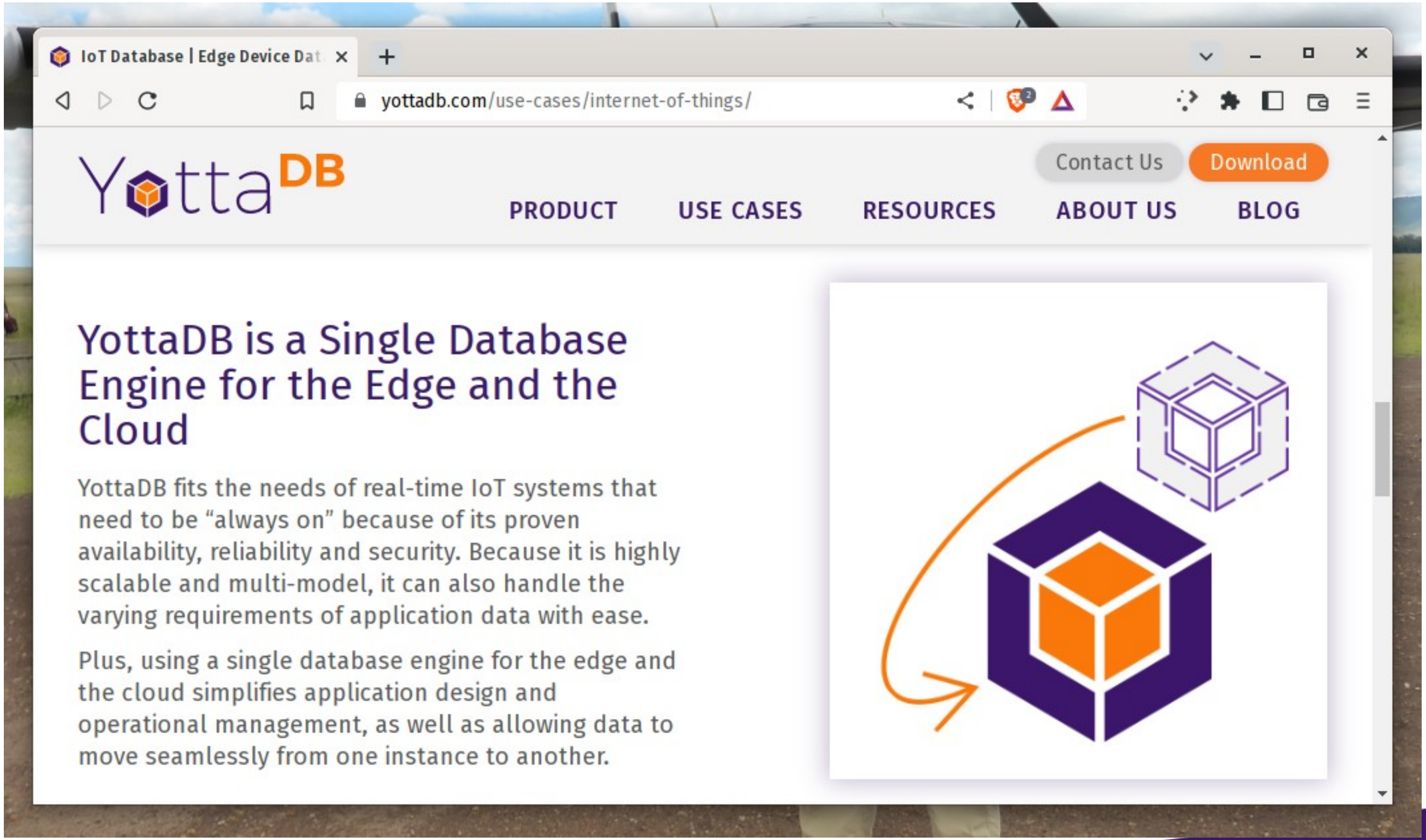
Limit to holdings in UAntwerpen □

Zoektips

Search tips:

`claus` searches for `claus` in the authors, title or subject fields.

Industrial Internet of Things


IoT Database | Edge Device Dat
yottadb.com/use-cases/internet-of-things/
YottaDB
Contact Us
Download
PRODUCT
USE CASES
RESOURCES
ABOUT US
BLOG

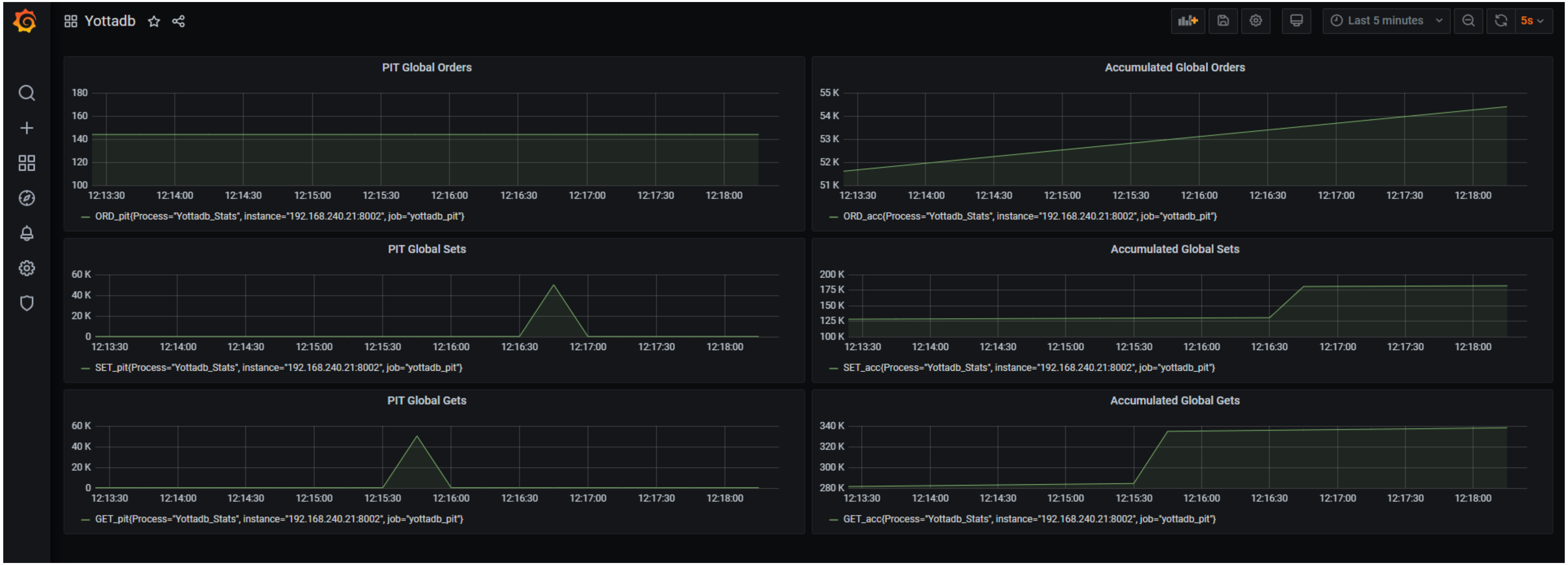
YottaDB is a Single Database Engine for the Edge and the Cloud

YottaDB fits the needs of real-time IoT systems that need to be “always on” because of its proven availability, reliability and security. Because it is highly scalable and multi-model, it can also handle the varying requirements of application data with ease.

Plus, using a single database engine for the edge and the cloud simplifies application design and operational management, as well as allowing data to move seamlessly from one instance to another.

Ecosystem Examples

Grafana Dashboard



— ORD_pit{Process="Yottadb_Stats", instance="192.168.240.21:8002", job="yottadb_pit"}

— ORD_acc{Process="Yottadb_Stats", instance="192.168.240.21:8002", job="yottadb_pit"}

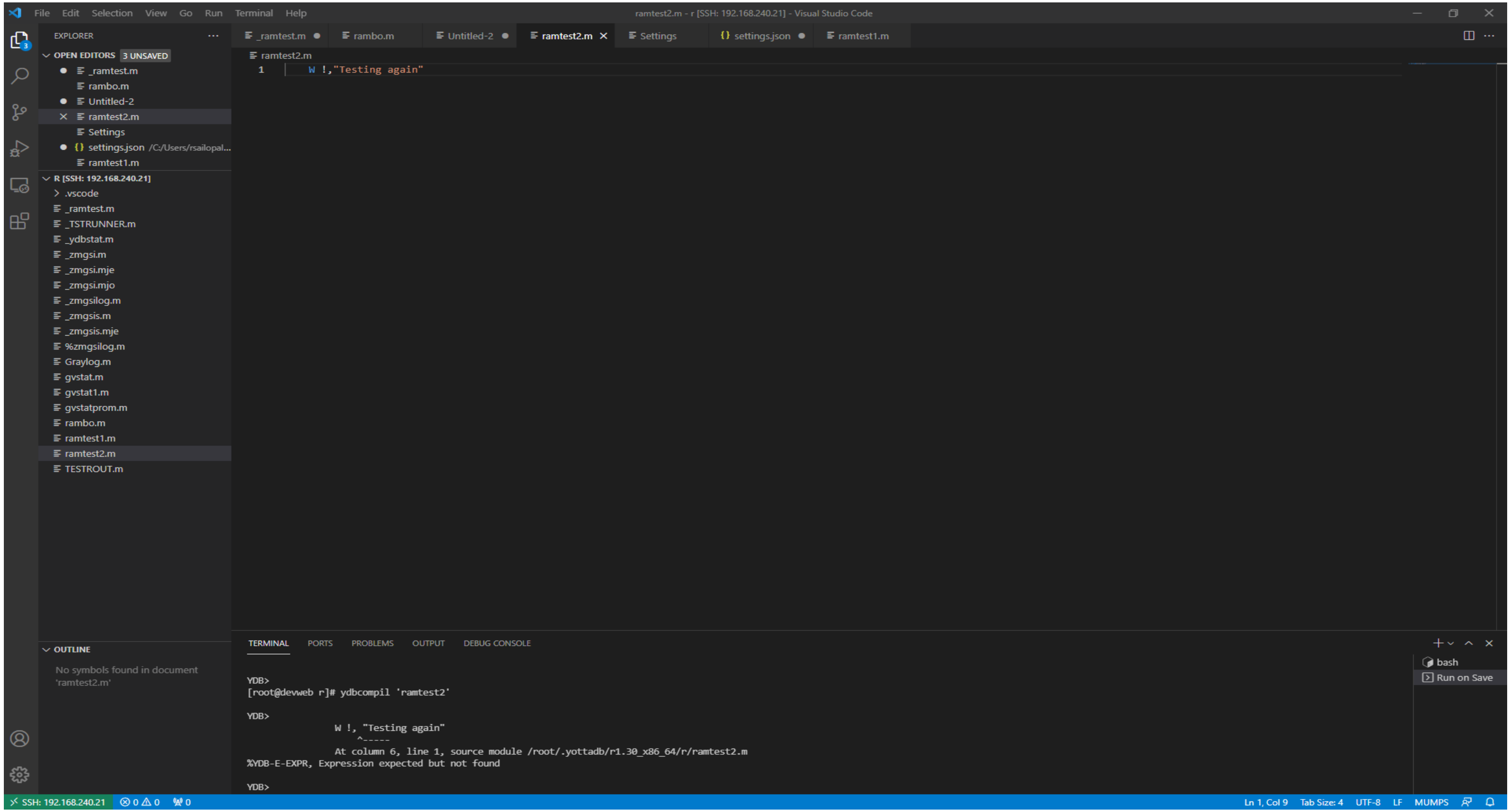
— SET_pit{Process="Yottadb_Stats", instance="192.168.240.21:8002", job="yottadb_pit"}

— SET_acc{Process="Yottadb_Stats", instance="192.168.240.21:8002", job="yottadb_pit"}

— GET_pit{Process="Yottadb_Stats", instance="192.168.240.21:8002", job="yottadb_pit"}

— GET_acc{Process="Yottadb_Stats", instance="192.168.240.21:8002", job="yottadb_pit"}

Editing in Visual Studio



ramtest2.m

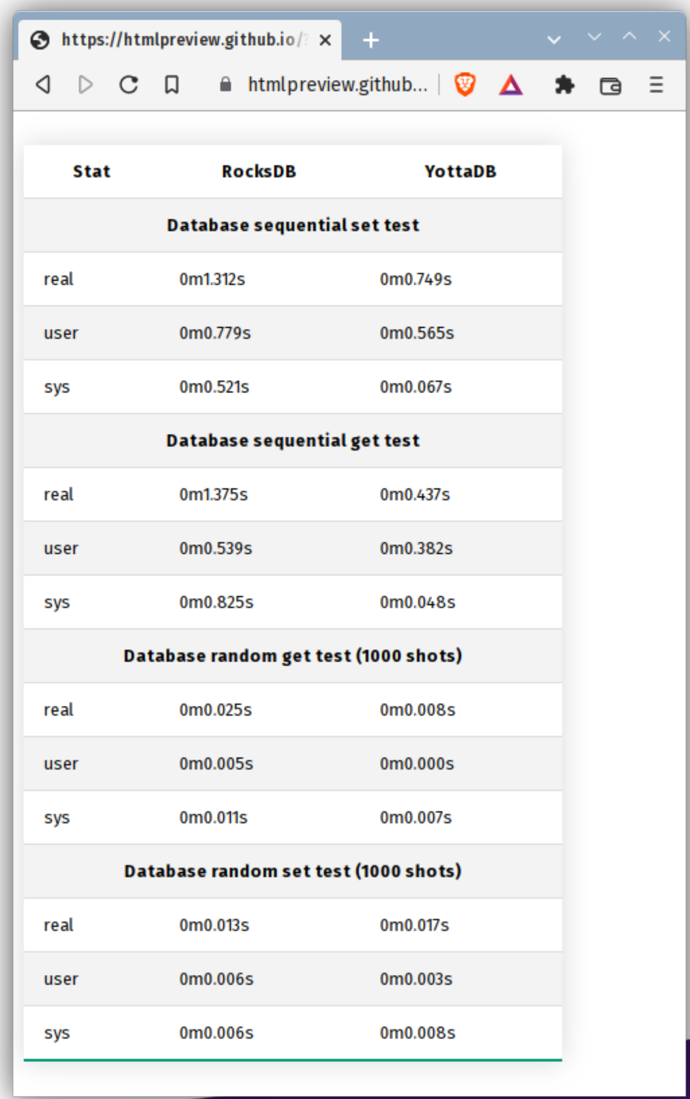
```
	W !,"Testing again"
```

No symbols found in document 'ramtest2.m'

```
YDB>
[root@devweb r]# ydbcompil 'ramtest2'

YDB>
		W !, "Testing again"
		    ^-----
		At column 6, line 1, source module /root/.yottadb/r1.30_x86_64/r/ramtest2.m
%YDB-E-EXPR, Expression expected but not found

YDB>
```

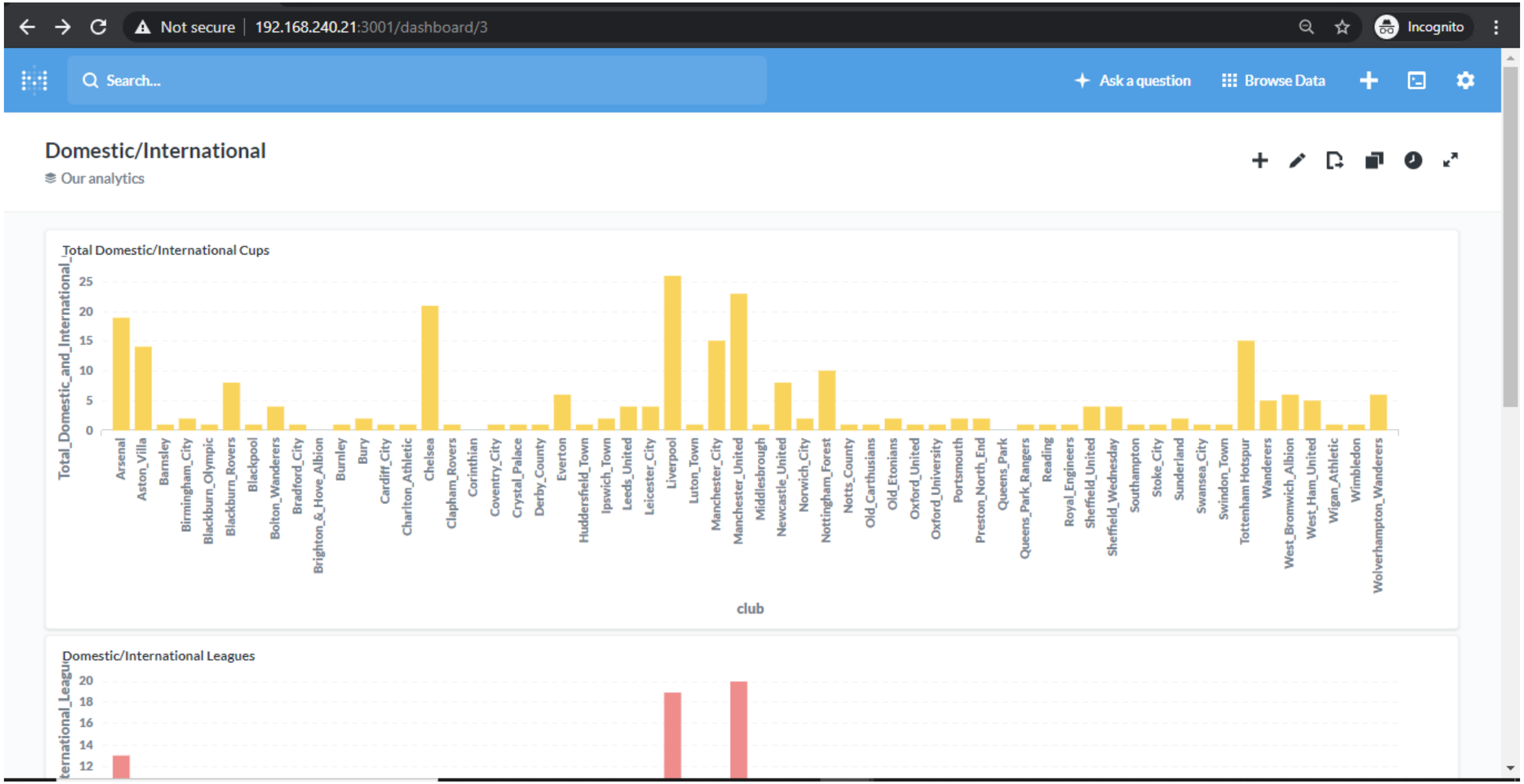
YottaDB vs RocksDB

https://github.com/RamSailopal/YottaDBvsRocksDB



Stat	RocksDB	YottaDB
Database sequential set test		
real	0m1.312s	0m0.749s
user	0m0.779s	0m0.565s
sys	0m0.521s	0m0.067s
Database sequential get test		
real	0m1.375s	0m0.437s
user	0m0.539s	0m0.382s
sys	0m0.825s	0m0.048s
Database random get test (1000 shots)		
real	0m0.025s	0m0.008s
user	0m0.005s	0m0.000s
sys	0m0.011s	0m0.007s
Database random set test (1000 shots)		
real	0m0.013s	0m0.017s
user	0m0.006s	0m0.003s
sys	0m0.006s	0m0.008s

Integration with Business Intelligence



Domestic/International

Our analytics

Total Domestic/International Cups

Domestic/International Leagues

jsonHIVES vs. MongoDB* … 1

- Caveats
  - jsonHIVES is still in development
  - Both were compared “out of the box” with no tuning or optimization

* courtesy Stefano Lalli

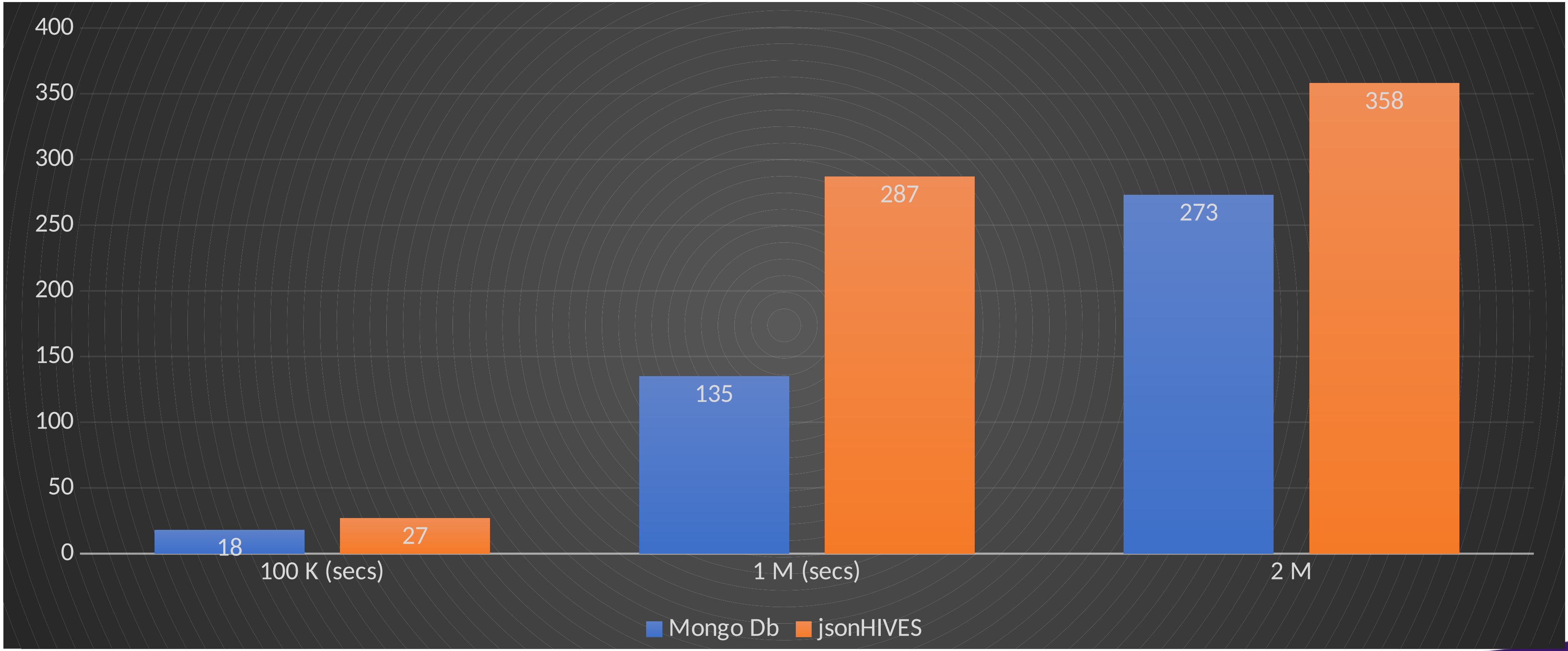
jsonHIVES vs. MongoDB … 2

- Identical data
  - 100 million identical records
  - Each record contains 15 nodes, i.e., 1.5 billion nodes
  - Indexes on searched fields
- Docker containers had same number of CPUs & RAM

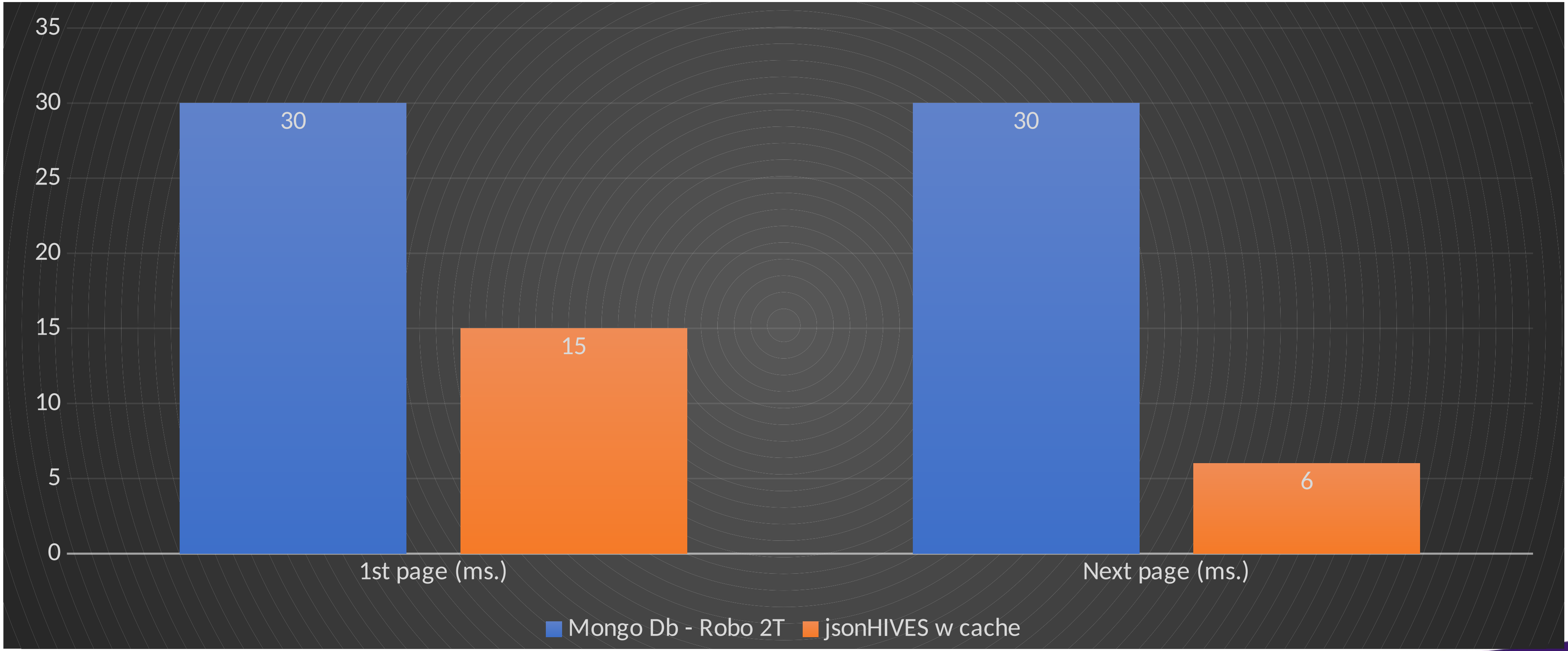
jsonHIVES vs. MongoDB … 3

- Clients
  - JsonHIVES – node.js driver
  - MongoDB – Robo2T (now called bongo)
- Page fetches with 50 records (750 nodes)

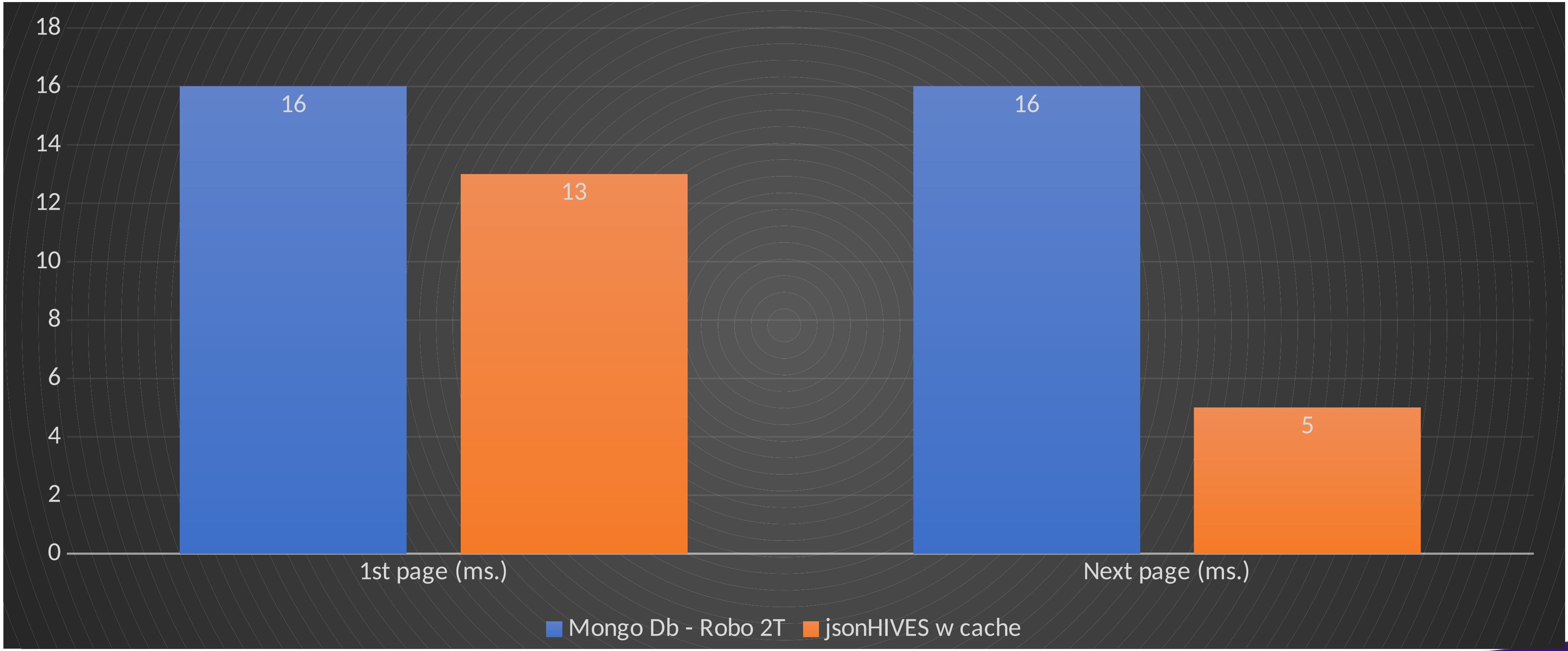
Bulk Insert Records



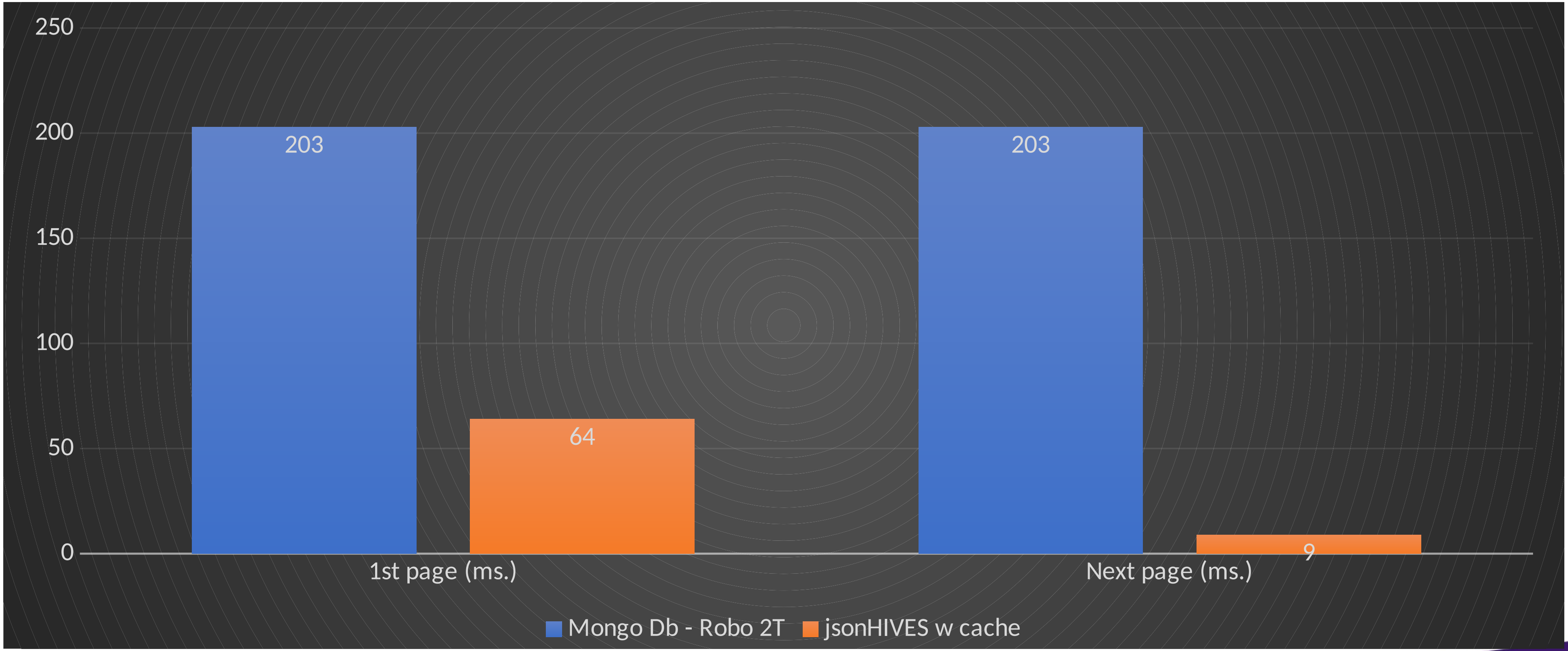
SELECT * WHERE name.last = "RAMOS"



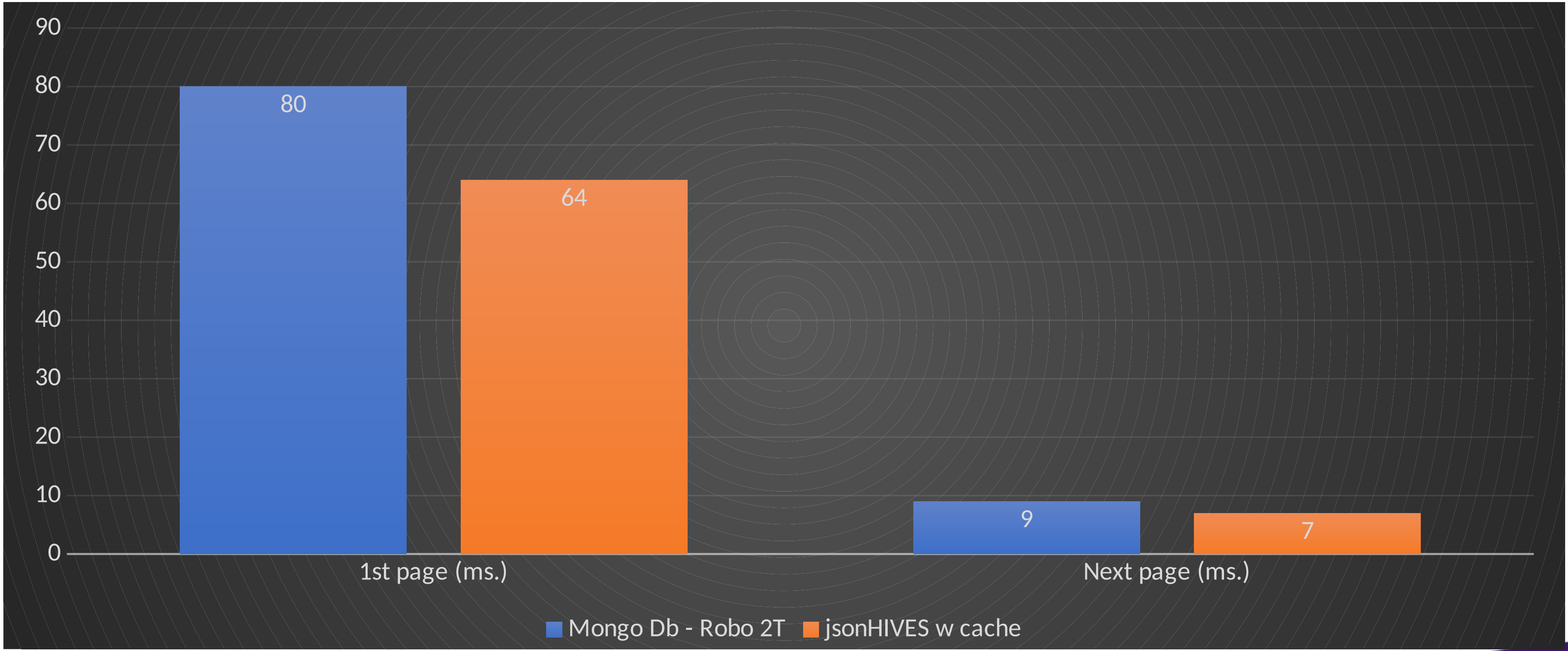
SELECT * WHERE zip = 90210



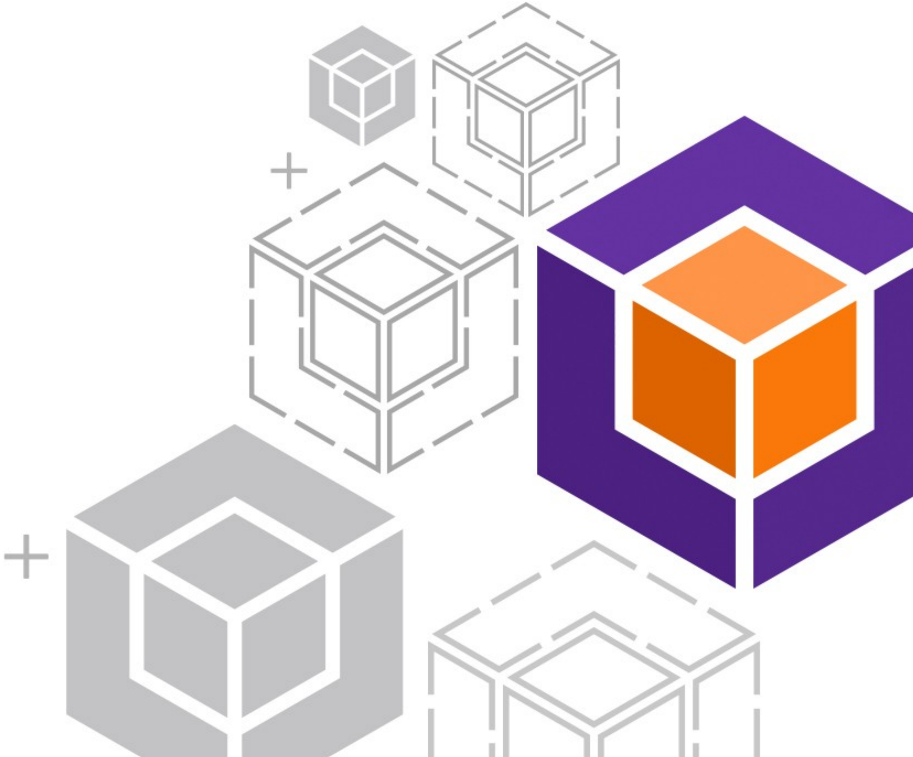
SELECT * WHERE city = "San Francisco" AND state = "California"



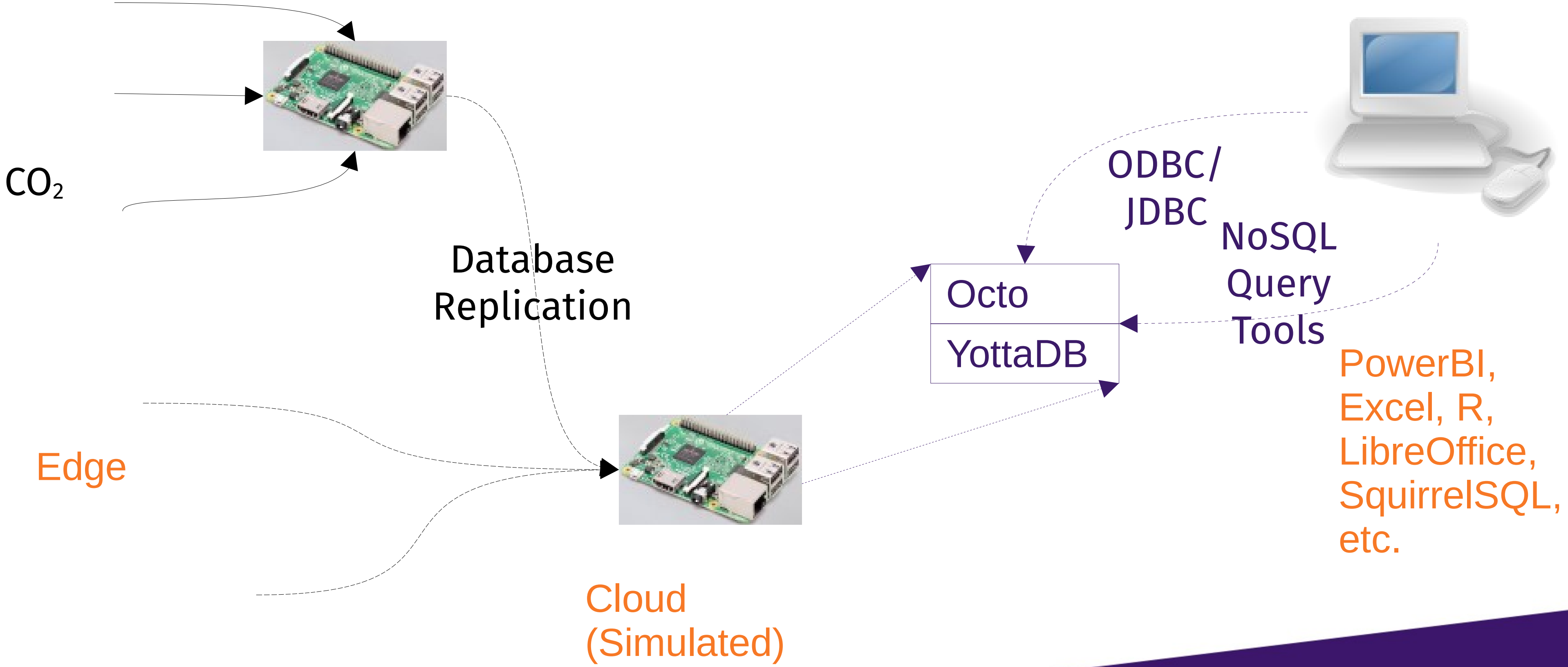
SELECT * WHERE state LIKE "%as" (/as$/)



Today’s Demos



Edge to Cloud – YottaDB Everywhere



CO_2

Database Replication

ODBC/ JDBC

NoSQL Query Tools

Octo

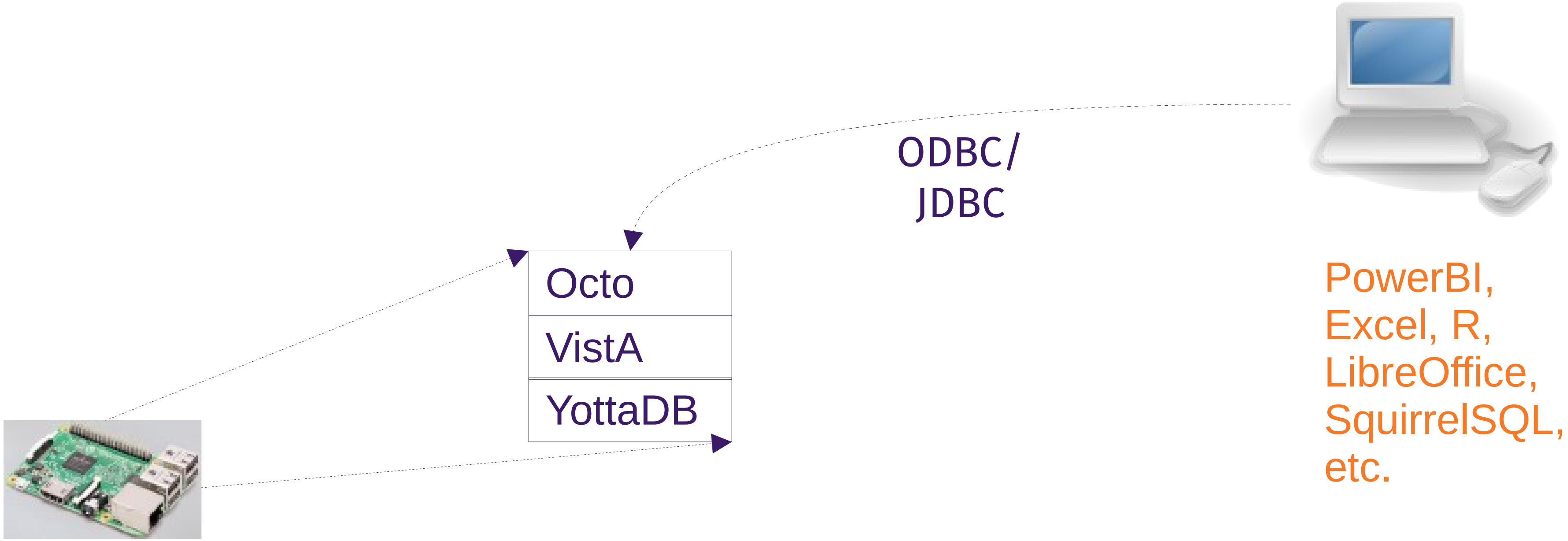
YottaDB

PowerBI, Excel, R, LibreOffice, SquirrelSQL, etc.

Edge

Cloud (Simulated)

VistA Electronic Health Records



ODBC/
JDBC

Octo

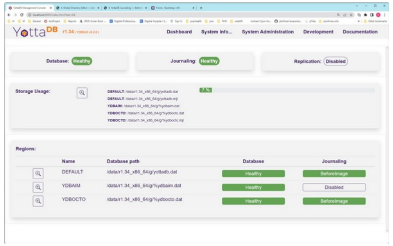
VistA

YottaDB

PowerBI,
Excel, R,
LibreOffice,
SquirrelSQL,
etc.

VistA
(Simulated
Patients)

GUI – Local or Remote



Database: Healthy

Journaling: Healthy

Replication: Disabled

Storage Usage:

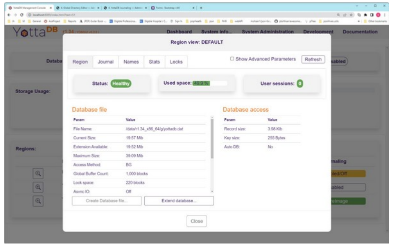
DEFAULT: /data/r1.34_x86_64/g/yottadb.dat
DEFAULT: /data/r1.34_x86_64/g/yottadb.mjl
YDBAIM: /data/r1.34_x86_64/g/%ydbaim.dat
YDBOCTO: /data/r1.34_x86_64/g/%ydbocto.dat
YDBOCTO: /data/r1.34_x86_64/g/%ydbocto.mjl

7 %

Regions:

Name	Database path	Database	Journaling
DEFAULT	/data/r1.34_x86_64/g/yottadb.dat	Healthy	BeforeImage
YDBAIM	/data/r1.34_x86_64/g/%ydbaim.dat	Healthy	Disabled
YDBOCTO	/data/r1.34_x86_64/g/%ydbocto.dat	Healthy	BeforeImage

VNC (or https)



Region view: DEFAULT

Status: Healthy

Used space: 49.9 %

User sessions: 0

Database file

Param	Value
File Name:	/data/r1.34_x86_64/g/yottadb.dat
Current Size:	19.57 Mib
Extension Available:	19.52 Mib
Maximum Size:	39.09 Mib
Access Method:	BG
Global Buffer Count:	1,000 blocks
Lock space:	220 blocks
Async IO:	Off

Database access

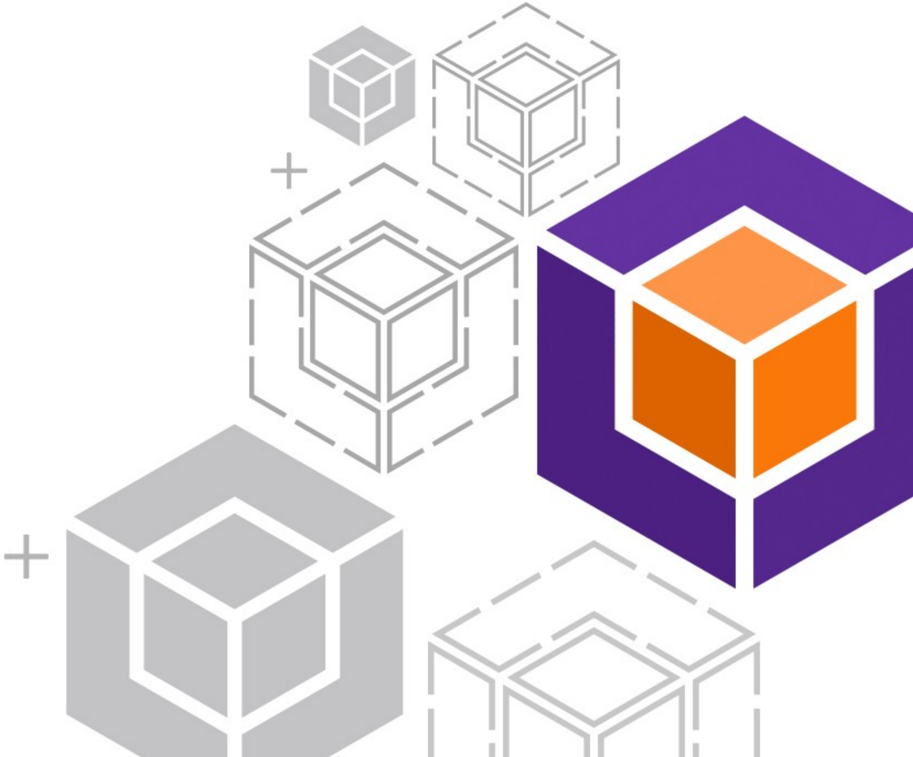
Param	Value
Record size:	3.98 Kib
Key size:	255 Bytes
Auto DB:	No

GUI

YottaDB

YottaDB Hands On – Ask Us For A Demo!

More Information



Links

- https://yottadb.com
- https://gitlab.com/YottaDB
- https://docs.yottadb.com

YottaDB

Thank You!

K.S. Bhaskar
bhaskar@yottadb.com