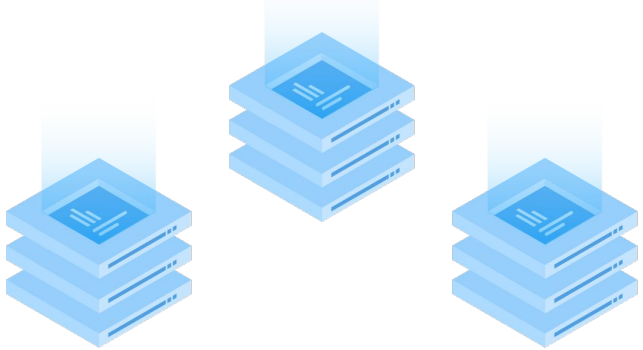
# Building a database startup in China

## Trends in Tech and Business

By Dongxu Huang

PingCAP TiDB

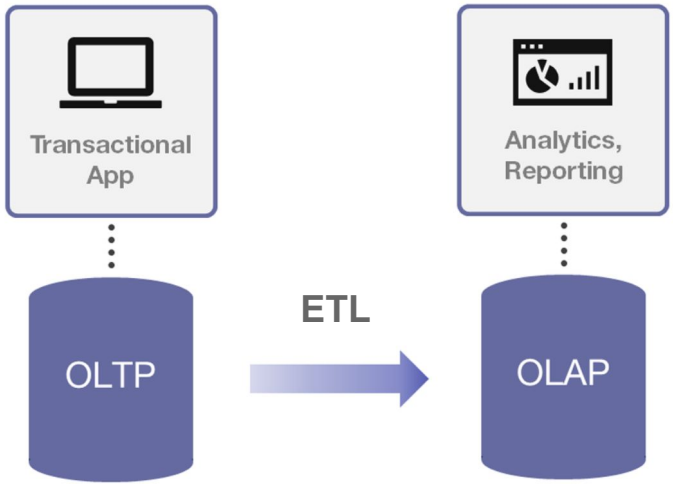


PingCAP

# Part I - Intro to PingCAP

# OLTP & OLAP

- OLTP (Online transaction processing)
  - Oracle / SQL Server / IBM DB2
  - MySQL / PostgreSQL
- OLAP (Online analytical processing)
  - SAP HANA / Apache Spark / Pivotal Greenplum
  - Hadoop Hive / Apache Impala / Apache Kudu / Presto / Druid
  - ...



# What are the pain points?

- Data is growing at a faster rate than ever before
  - Trend: AI / Data mining
  - Distributed systems become mainstream
- Traditional OLTP databases are no longer sufficient to meet the needs of companies in the big data era
  - Scalability with strong consistency
- OLTP and OLAP are separate to each other
  - You can’t do real-time analyze because of the existence of ETL
  - And writing & maintaining ETL jobs is very hard and boring
- People are moving to cloud, maintaining infrastructure is painful
- SQL never dies
  - Legacy codes / applications, they already depend on SQL databases.
  - Everybody knows SQL, and it’s developer-friendly.

# PingCAP: A Chinese-born Database Company

- Founded in 2015 by 3 infrastructure engineers
- **What's the story?**

PingCAP

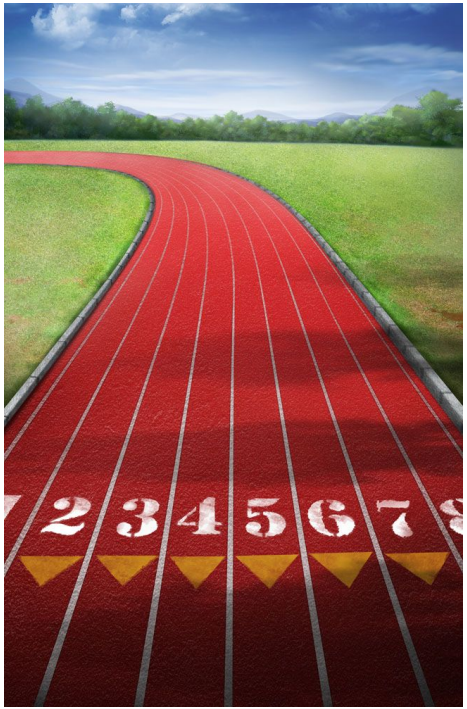
# Part II - What's TiDB?

# The Wishlist

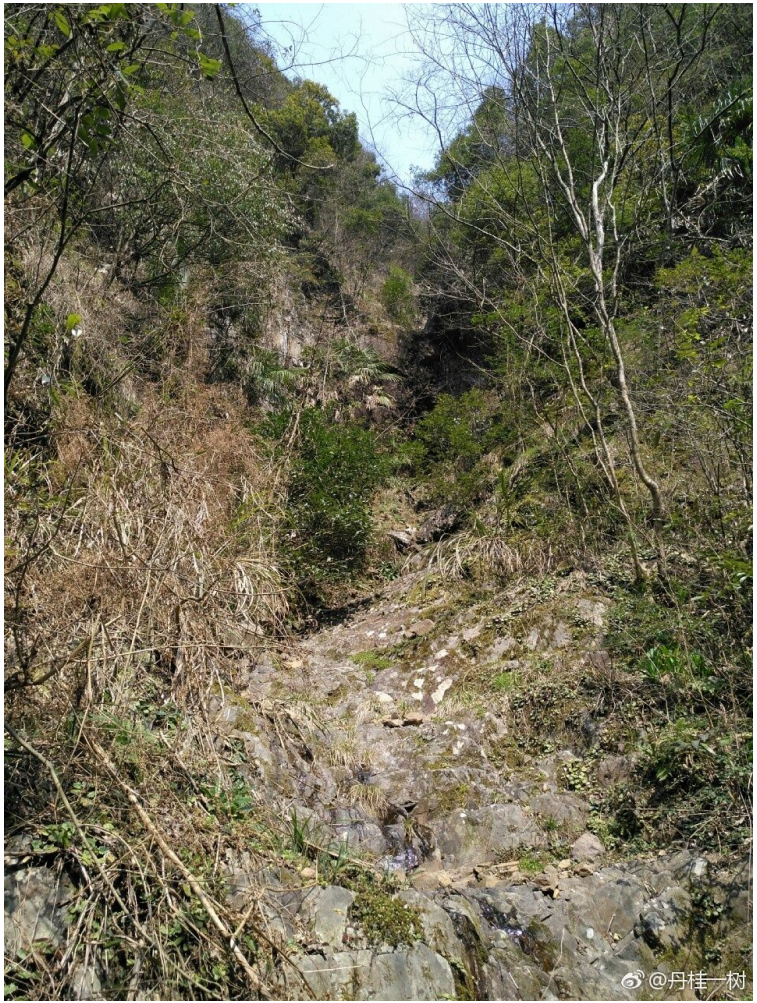
- Scalability
  - Scaling the capacity/thoughput with the cluster size
  - Elasticly scaling out for **hyper growth**
- ACID semantics with steady transaction latency
- High Availability
- OLAP without interfering the OLTP workload

ONE
RING
TO
RULE
THEM
ALL

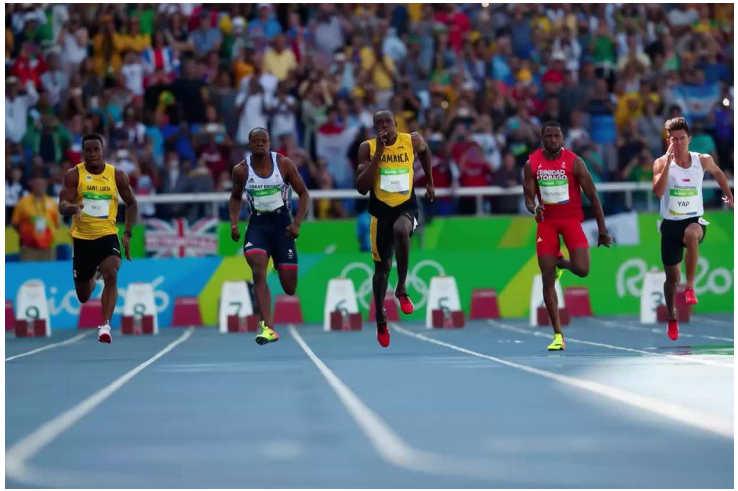
# Expectation



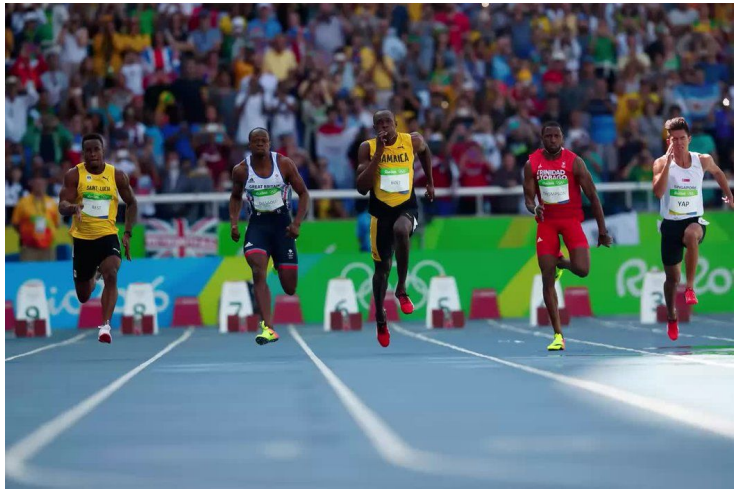
# Reality



VS

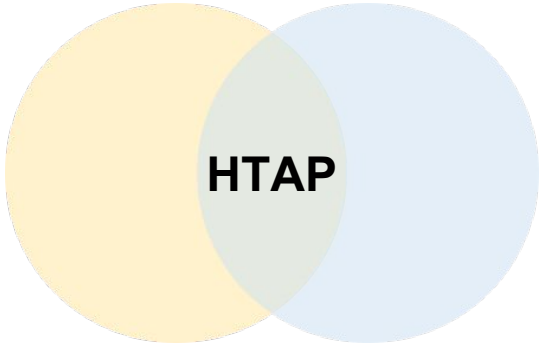


VS

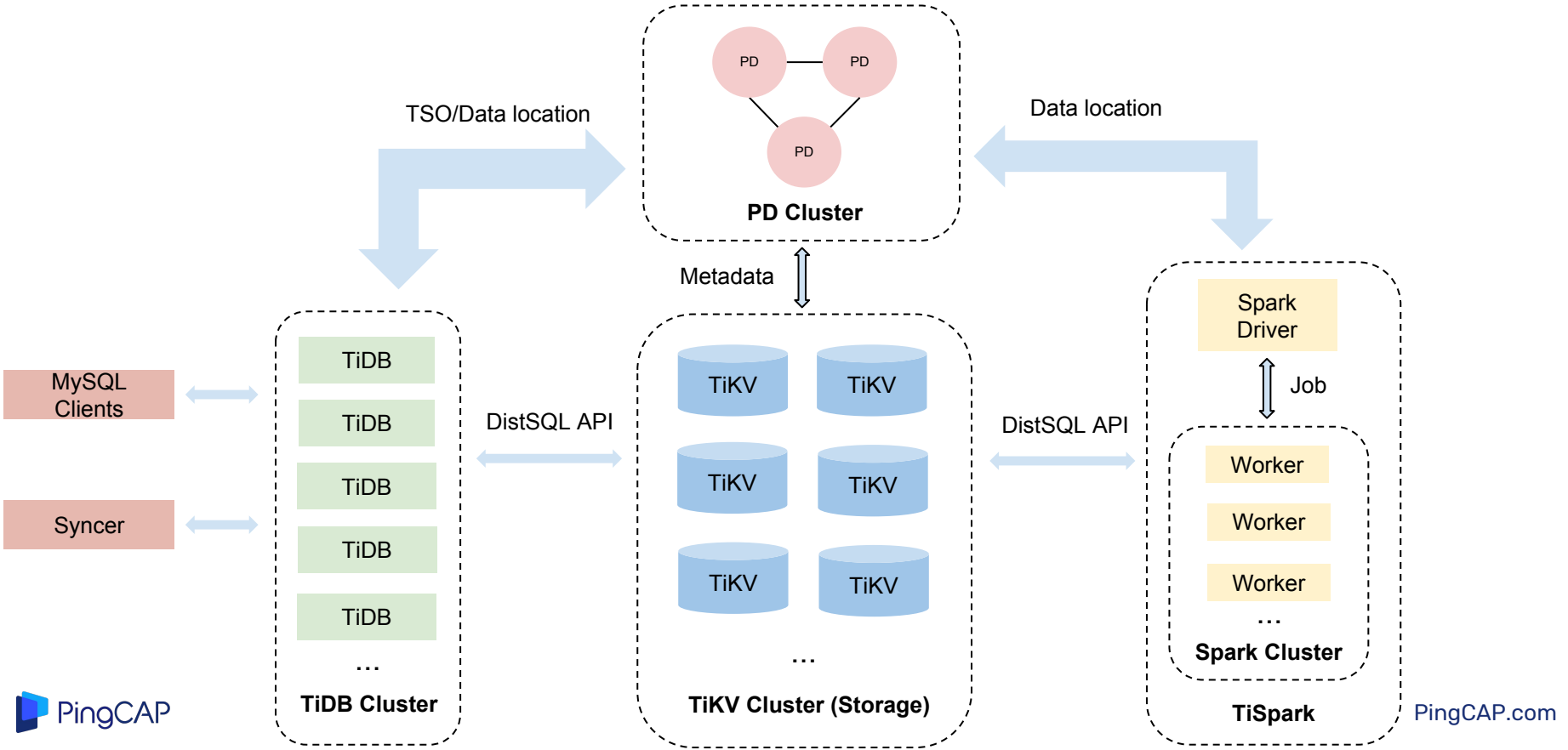


# TiDB platform

- NewSQL: the best features of both RDBMS and NoSQL
  - Full-featured SQL
    - MySQL compatibility
  - ACID compliance
  - HA with strong consistency
  - Elastic scalability
- HTAP
  - Serve both OLTP & Real-time OLAP



# TiDB Architecture




PingCAP

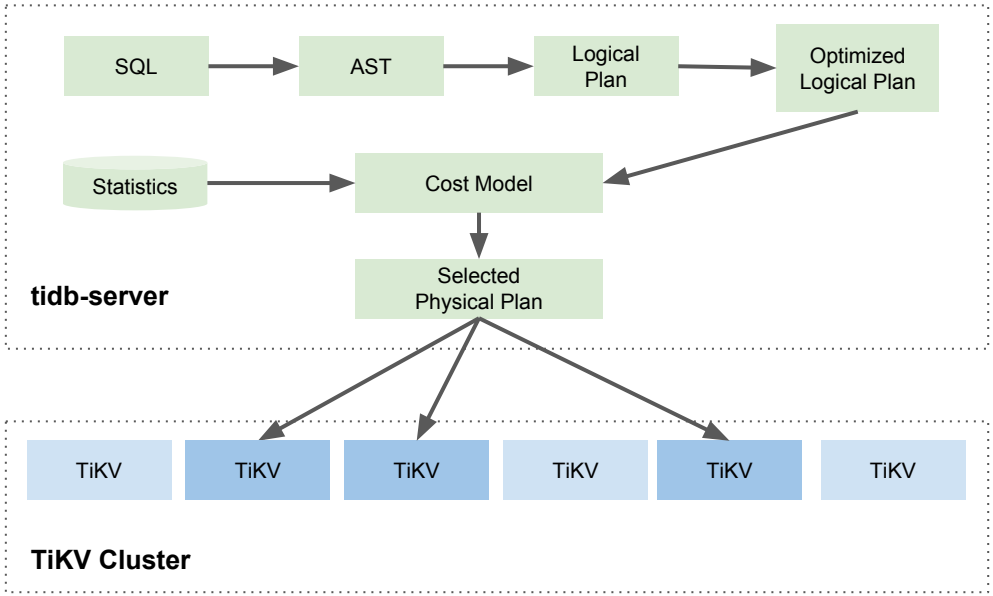
PingCAP.com

# Components

- TiDB (tidb-server)
- TiKV (tikv-server)
- Placement Driver (PD)
- TiSpark
- Tools (syncer / TiDB-Lightning / {tikv,pd}-ctl)
- TiDB-operator for Kubernetes

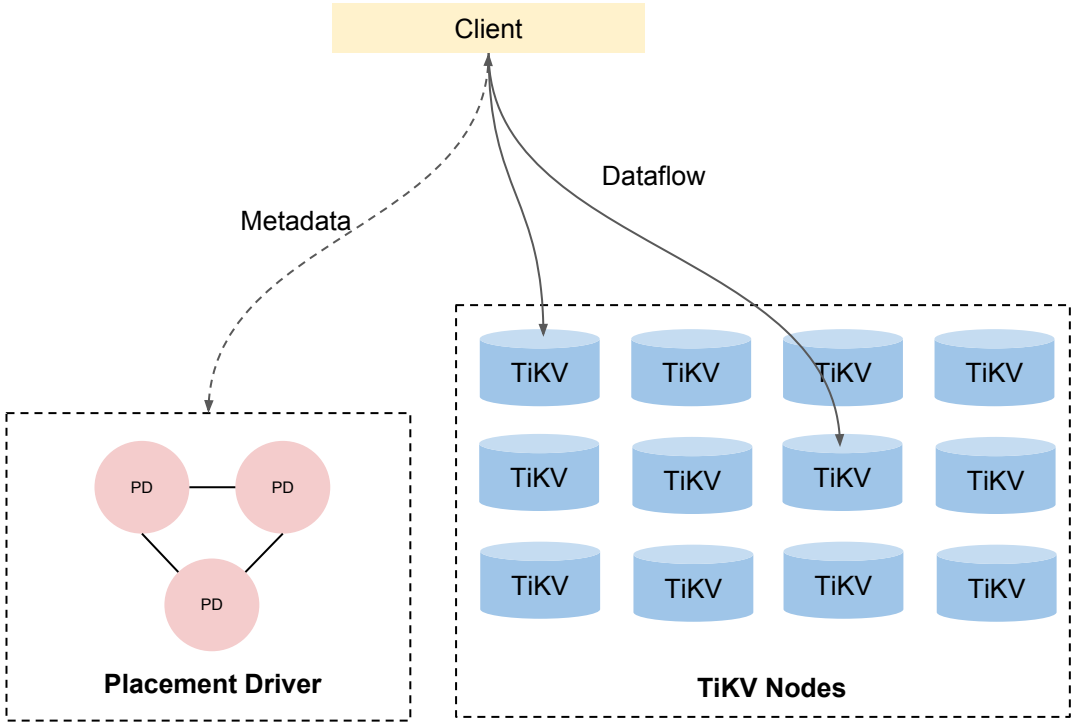
# TiDB (tidb-server)

- Stateless SQL layer
  - Client can connect to any existing tidb-server instance
  - TiDB *__will not__* re-shuffle the data across different tidb-servers
- Full-featured SQL Layer
  - Speak MySQL wire protocol
    - Why not reusing MySQL?
  - Homemade parser  & lexer
  - RBO & CBO
  - Secondary index support
  - DML & DDL



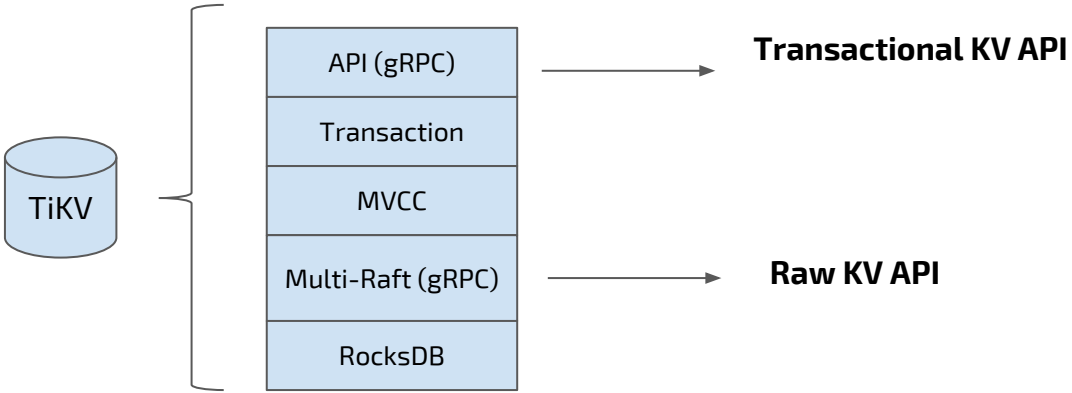
# TiKV (tikv-server)

- The **storage layer** for TiDB
- Distributed **Key-Value** store
  - **Support ACID Transactions**
  - Replicate logs by **Raft**
  - **Range** partitioning
    - Split / merge **dynamically**
  - Support coprocessor for **SQL operators pushdown**



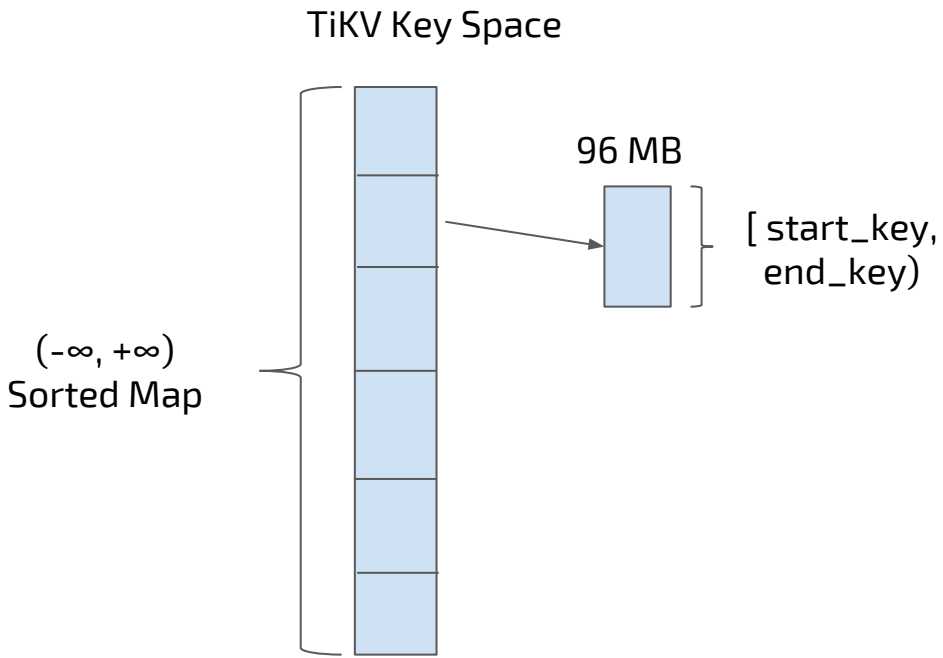
# TiKV (tikv-server) - Physical stack

Highly layered

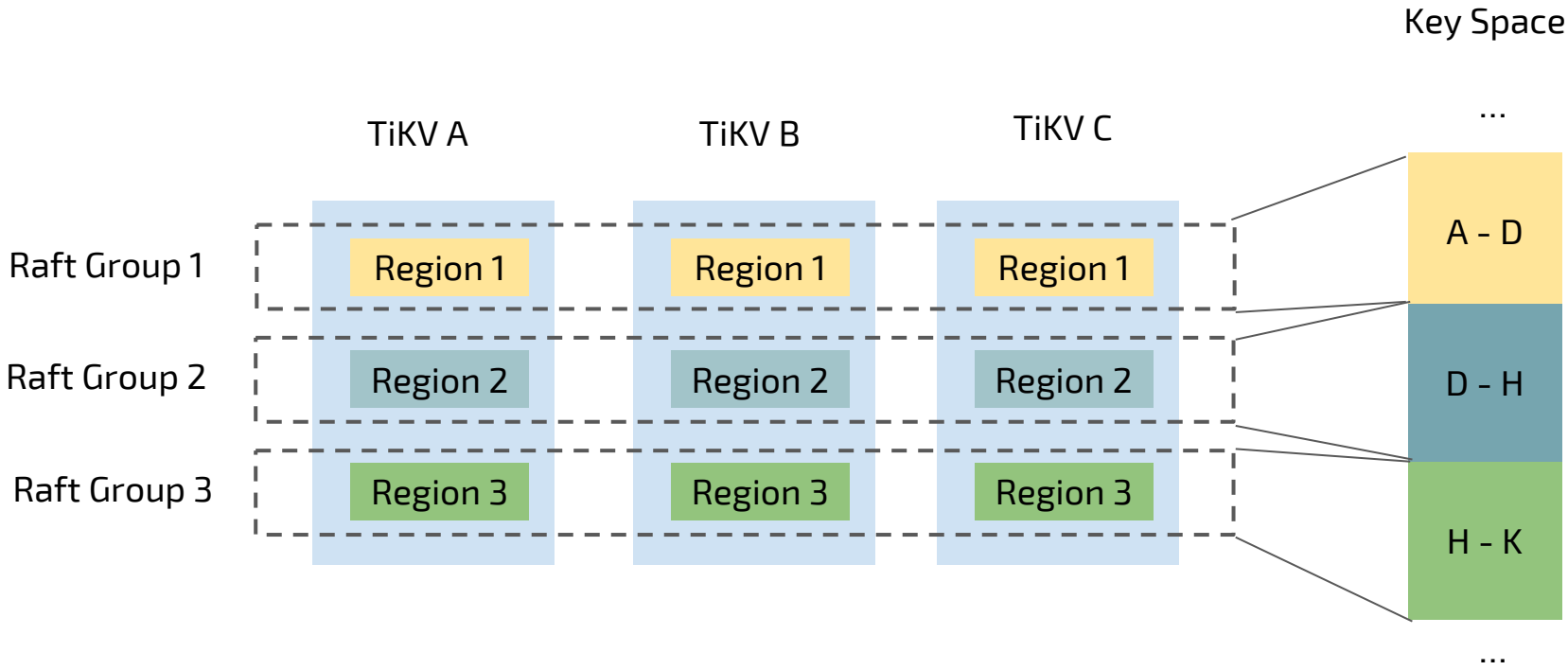


# TiKV (tikv-server) - Logical view (1/2)

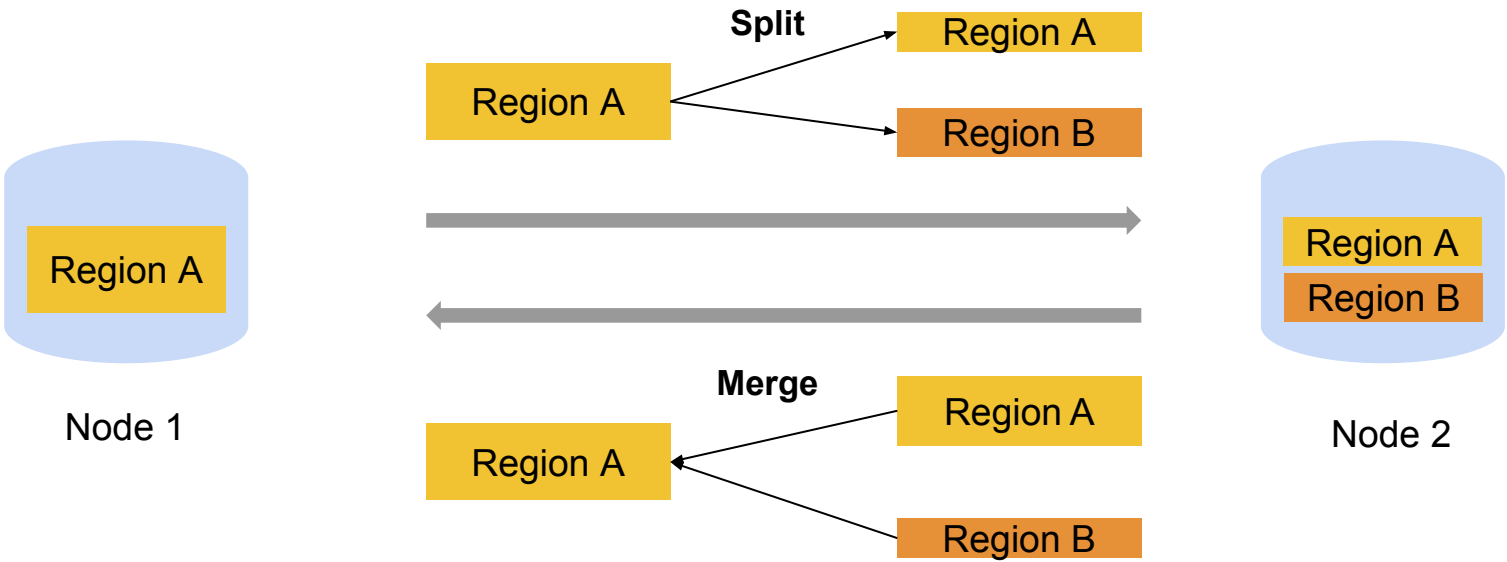
- Stores Key-Value pairs
- Infinite sorted (in byte-order) Key-Value map
- Key space is split into regions (Range-based) **dynamically**, like HBase
- Metadata: [start_key, end_key)
- Each region has multiple replicas (default 3) across different physical nodes, data is replicated by **Raft**
- All regions in the same node share the same RocksDB instance



# TiKV (tikv-server) - Logical view (2/2)

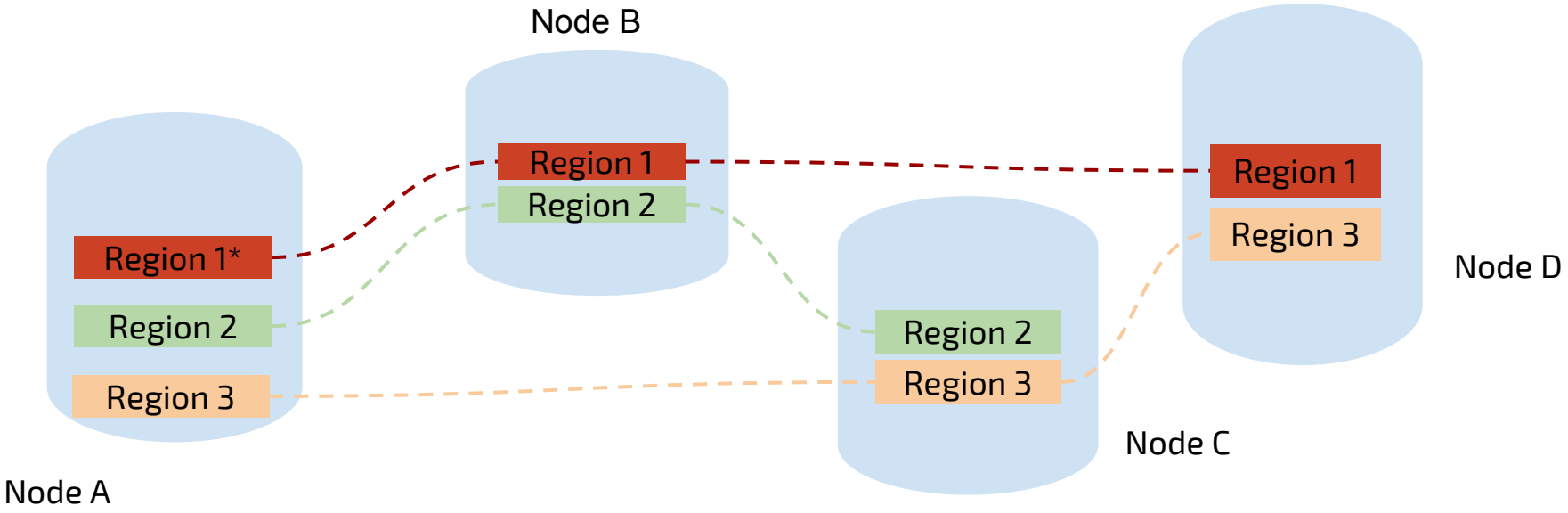


# TiKV (tikv-server) - Region split & merge

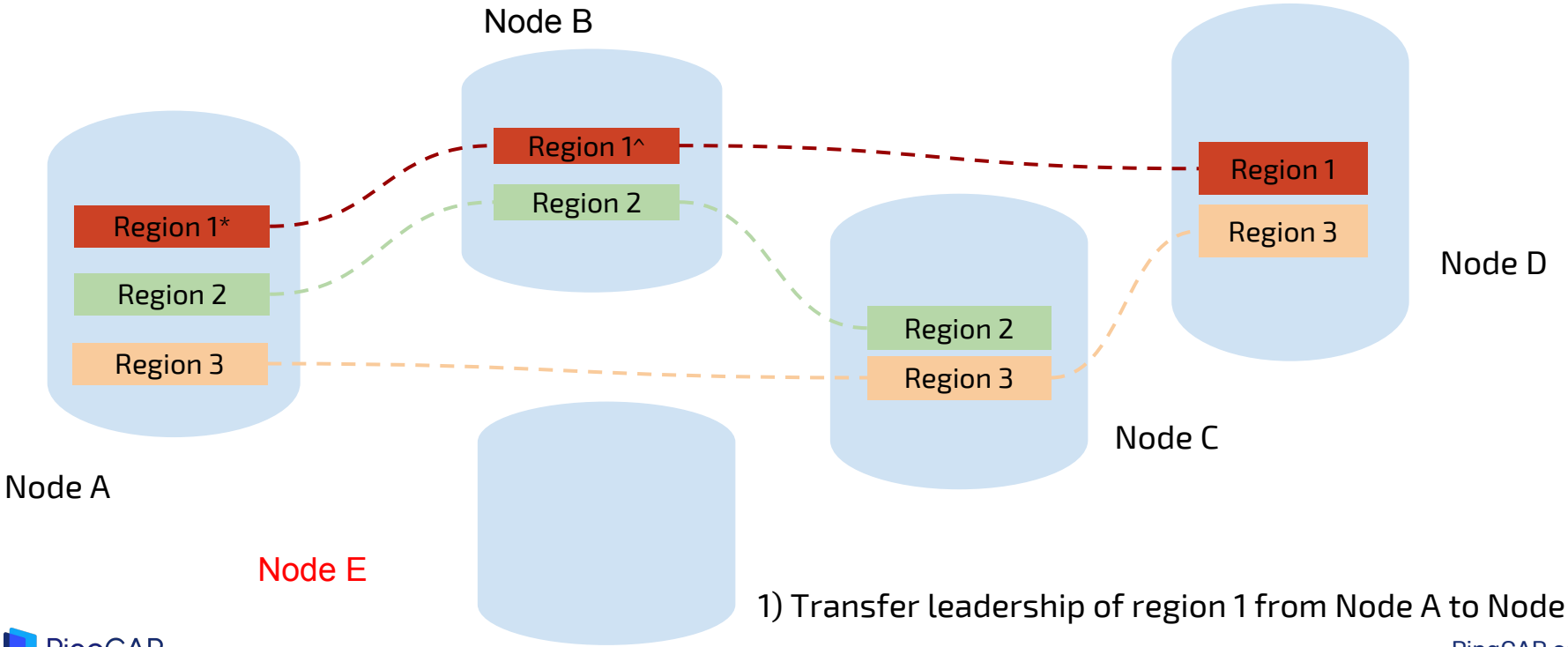



Region splitting and merging affect all replicas of one region.
The correctness and consistency are guaranteed by **Raft**.

# TiKV (tikv-server) - Scaling & Rebalancing

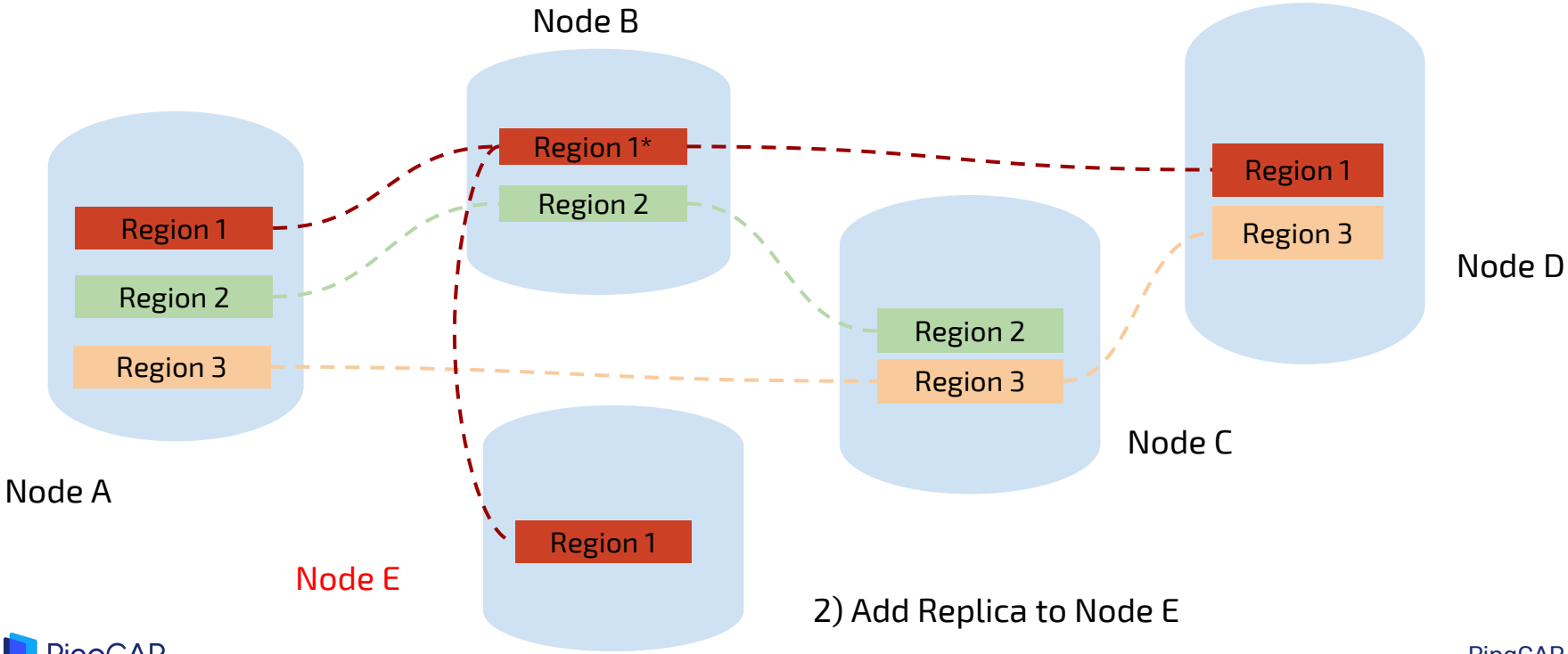


# TiKV (tikv-server) - Scaling & Rebalancing

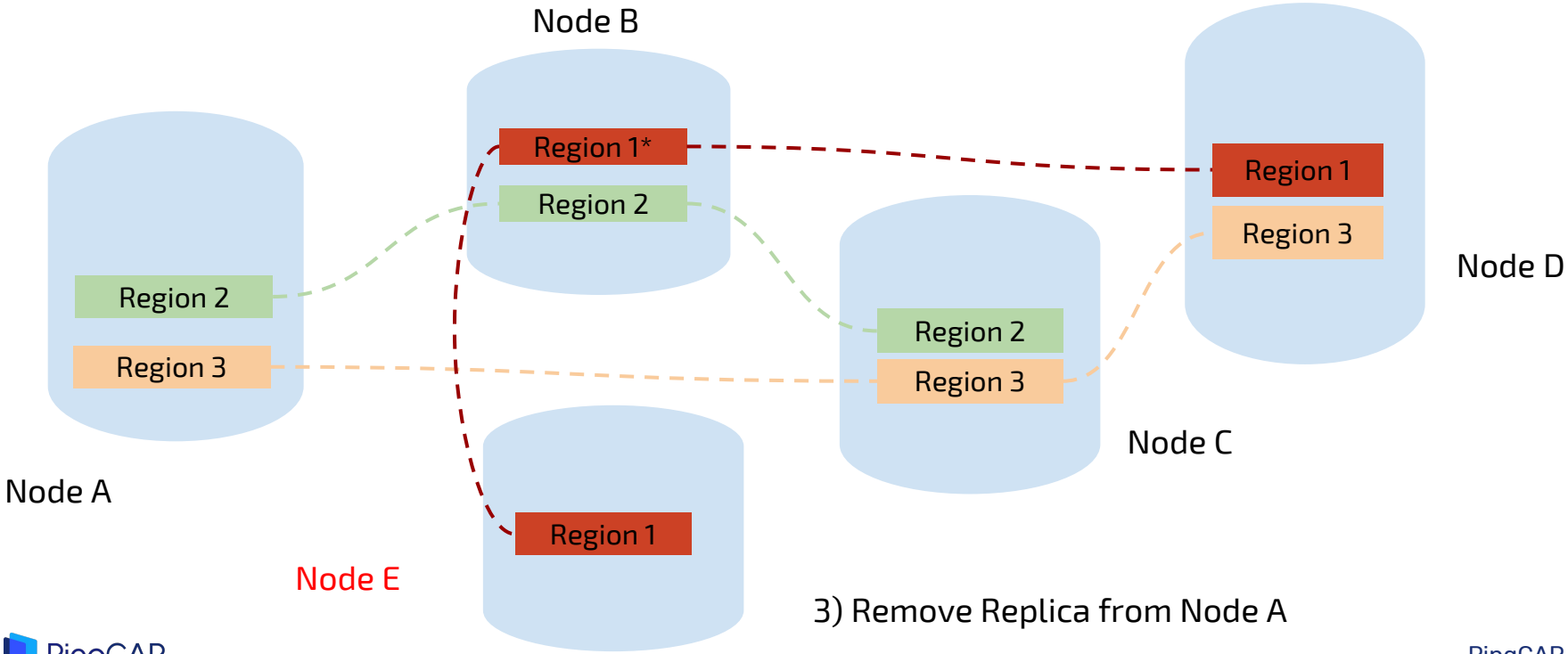



1) Transfer leadership of region 1 from Node A to Node B

# TiKV (tikv-server) - Scaling & Rebalancing



# TiKV (tikv-server) - Scaling & Rebalancing



PingCAP

# Part III - What has changed?

# What has changed?

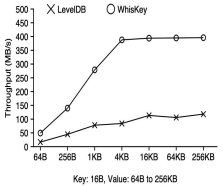



Figure 7: **Sequential-load Performance.** *This figure shows the sequential-load throughput of LevelDB and WiscKey for different value sizes for a 100-GB dataset. Key size is 16 B.*

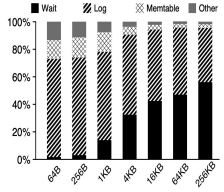



Figure 8: **Sequential-load Time Breakup of LevelDB.** *This figure shows the percentage of time incurred in different components during sequential load in LevelDB.*

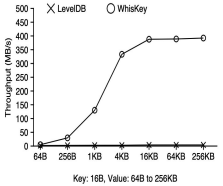



Figure 9: **Random-load Performance.** *This figure shows the random-load throughput of LevelDB and WiscKey for different value sizes for a 100-GB dataset. Key size is 16 B.*

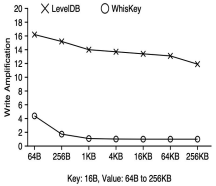
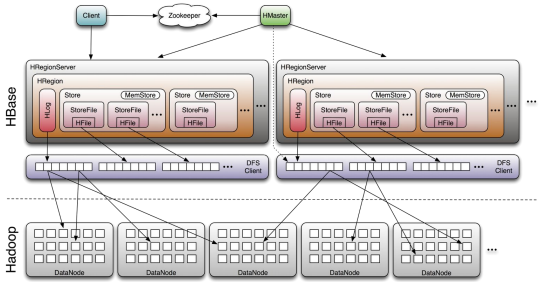
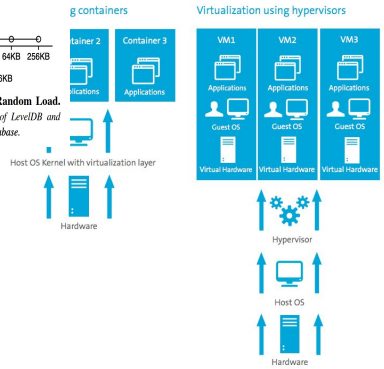
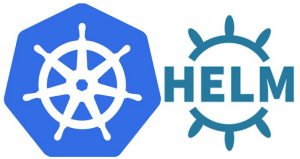


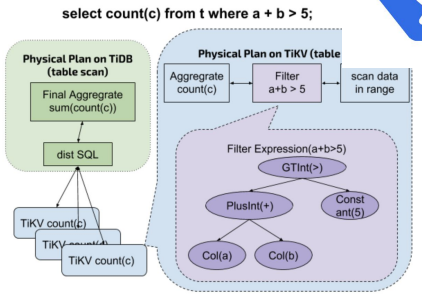

Figure 10: **Write Amplification of Random Load.** *This figure shows the write amplification of LevelDB and WiscKey for randomly loading a 100-GB database.*



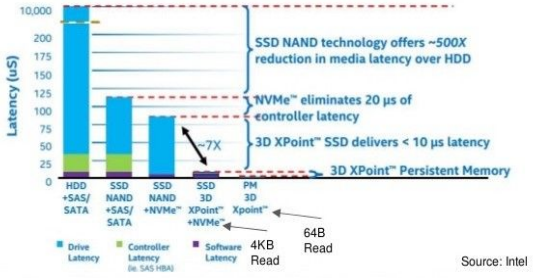


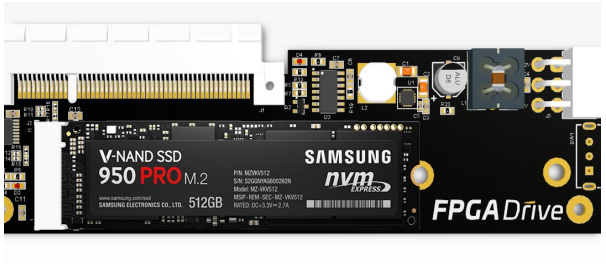

## Software!



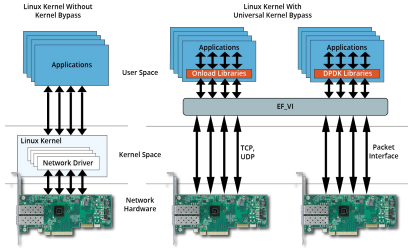



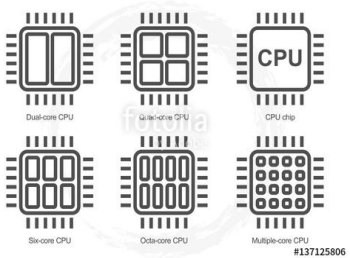
# What has changed?



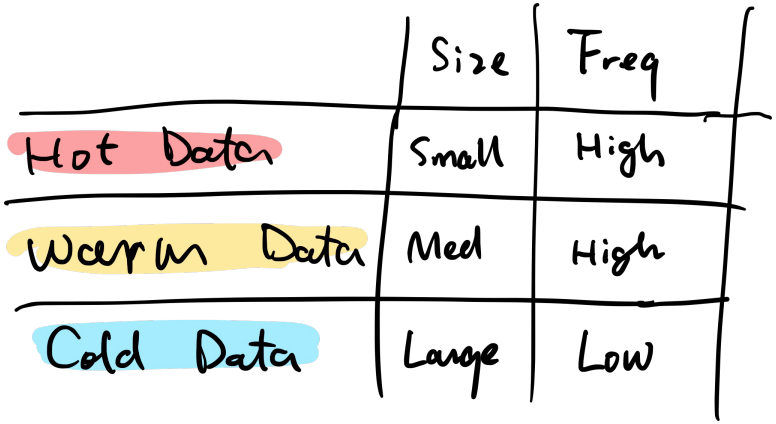




## Hardware!



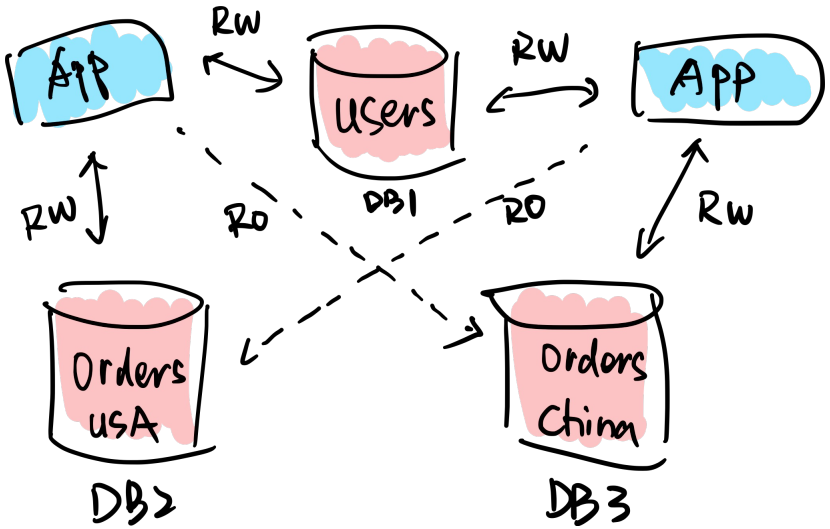



# What has changed?



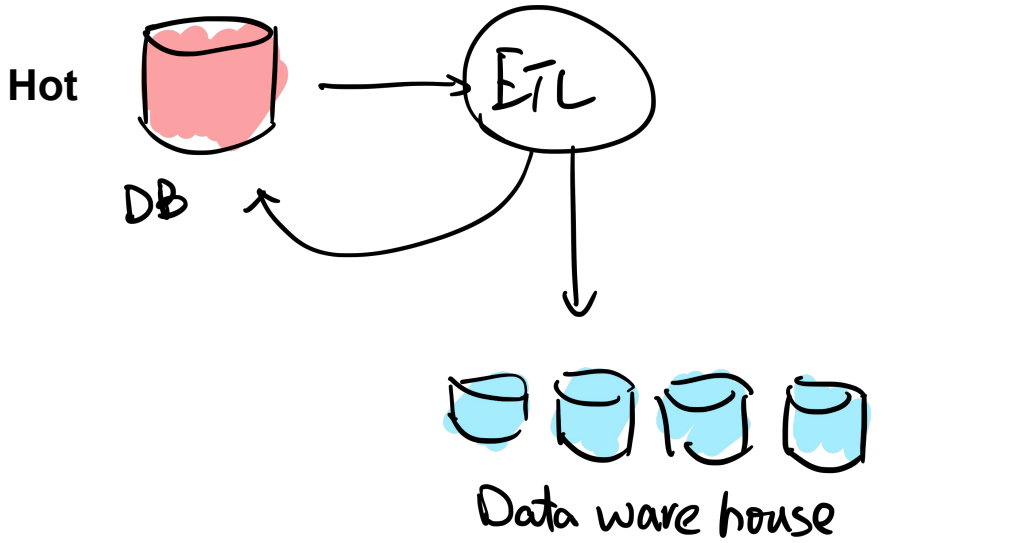

- Data type
  - Hot data: Need for speed!
  - Warm data: **The source of truth**
  - Cold data: Archive
- Warm data architecture
  - the missing part of modern data processing stack

# Let's say we have an application like this...

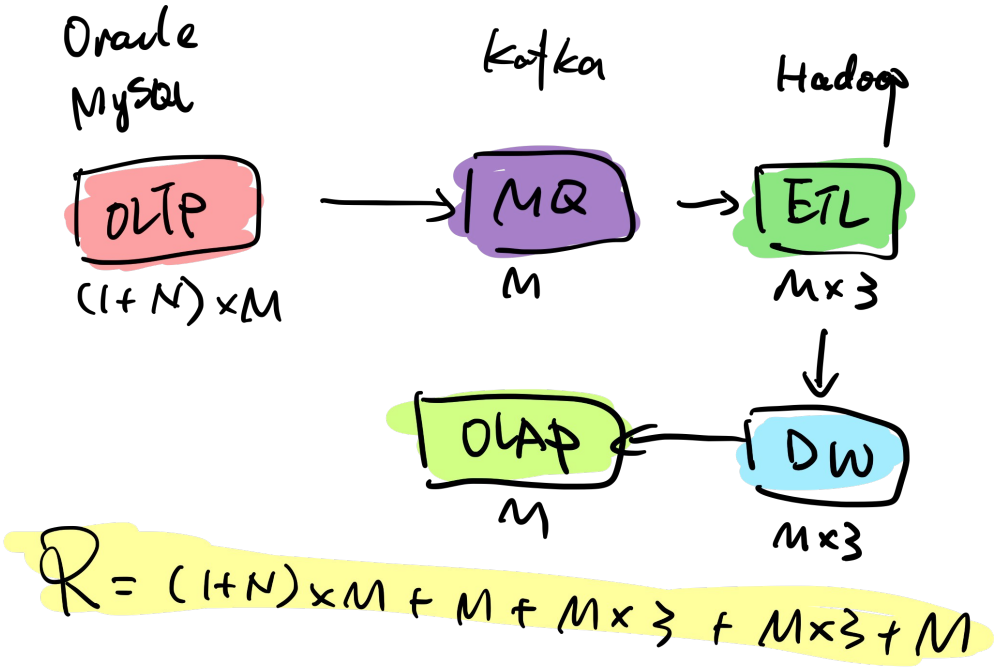



```
SELECT COUNT(DISTINCT t1.BuyerID)
FROM  Orders_USA t1, Orders_China
t2
WHERE  t1.BuyerID = t2.BuyerID;
```

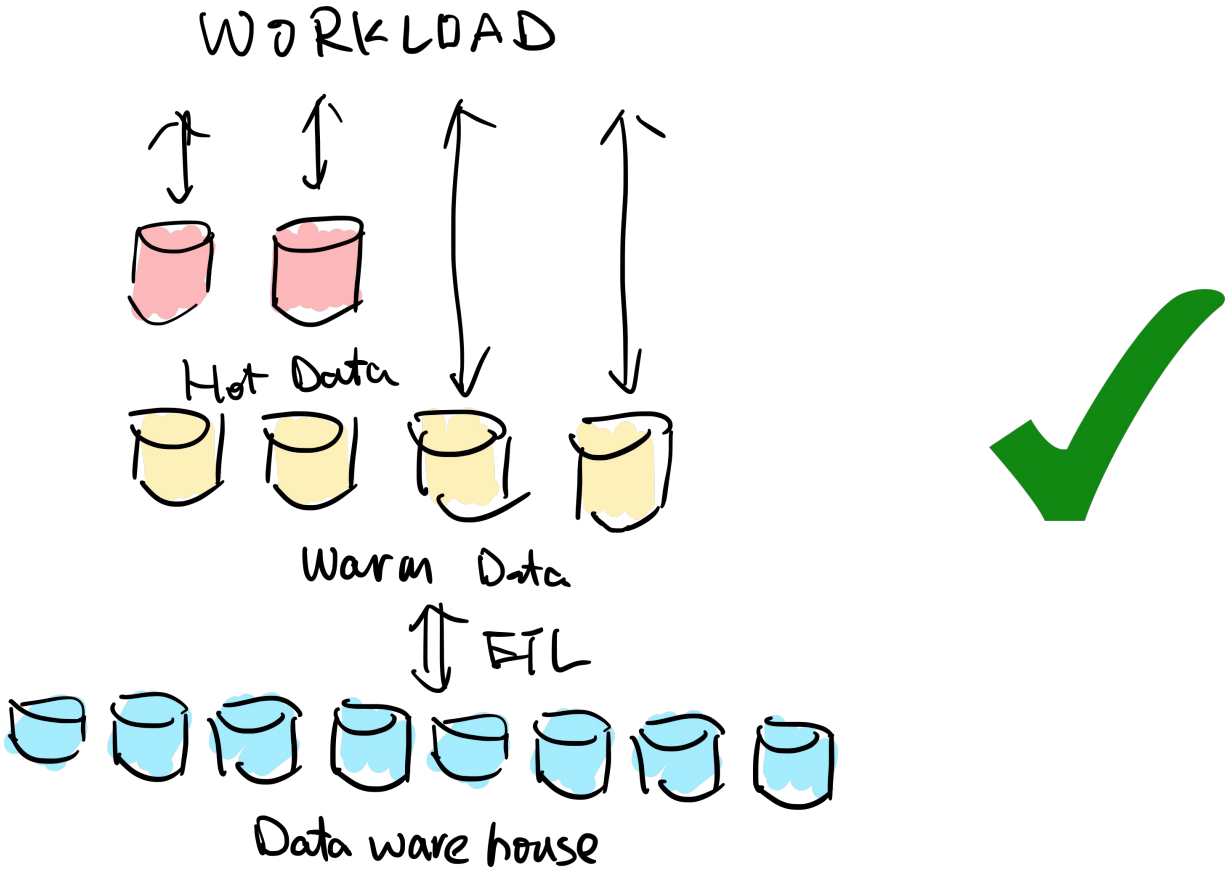
# In the old days...



# How many replicas do we need?




M = number of Applications.
N = Number of Replicas

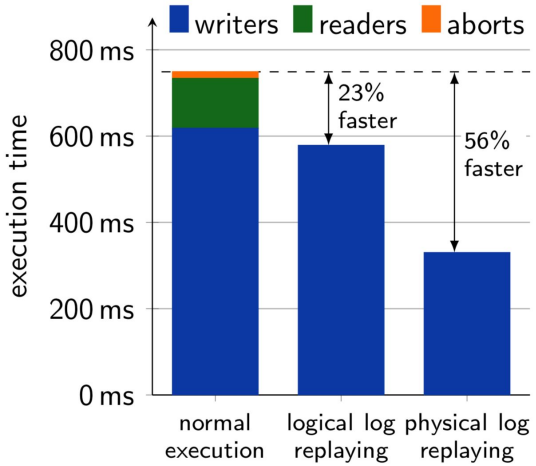

WORKLOAD
Hot Data
Warm Data
ETL
Data ware house

PingCAP

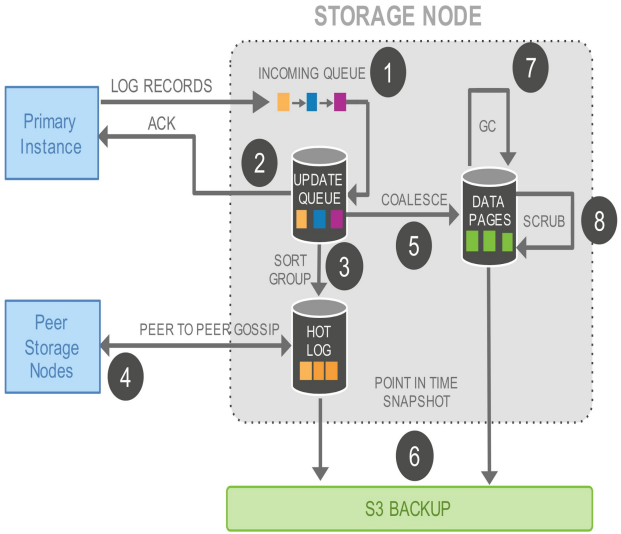
# Part IV - Tech trends

# Let’s say we want to build a new database in 2010s...

# Log is the new database




*HyPer*

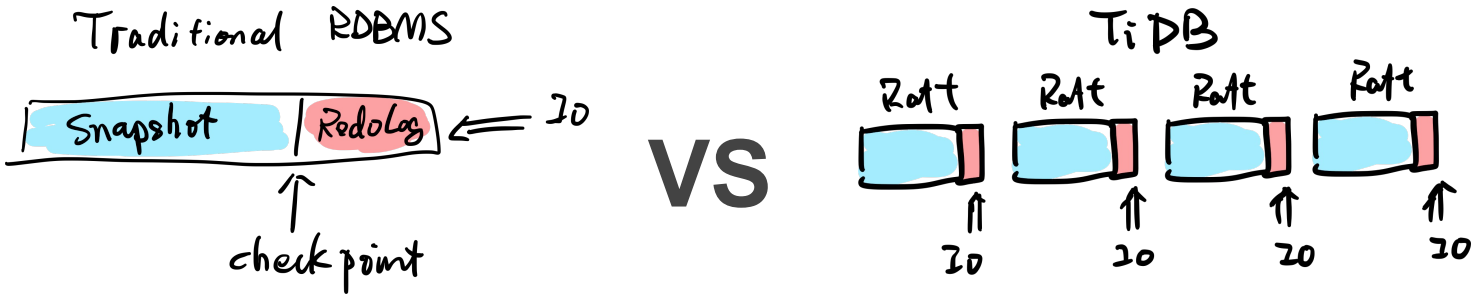



*AWS Aurora*

- Fewer I/O
- Smaller network packets

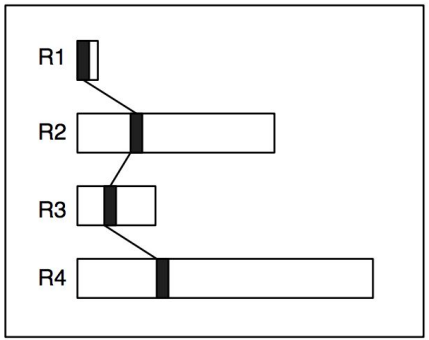
# Log is the new database

## Traditional RDBMS vs TiDB

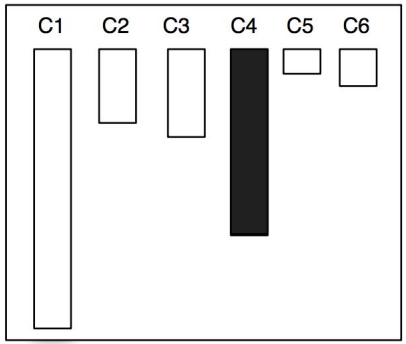


# Vectorized

**SELECT** SUM(C4) **FROM** R;

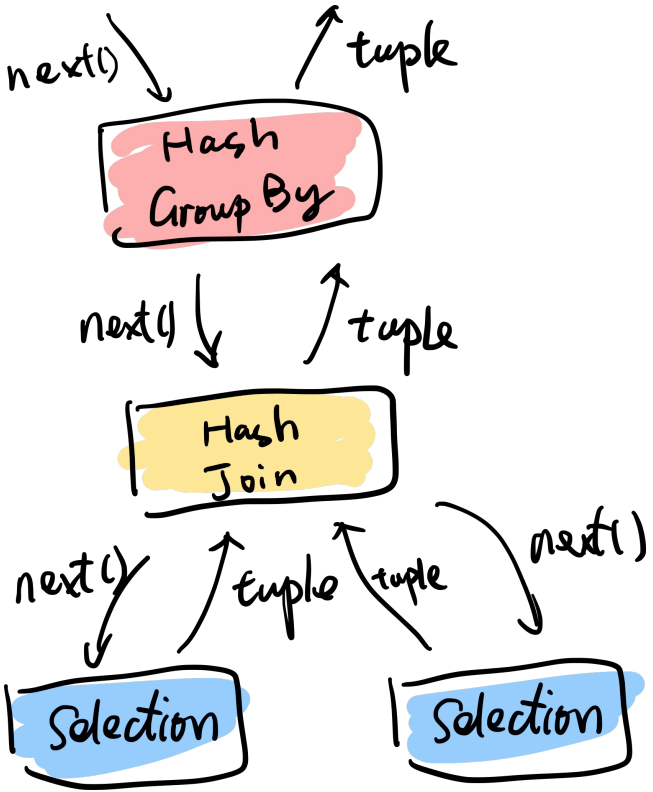



VS



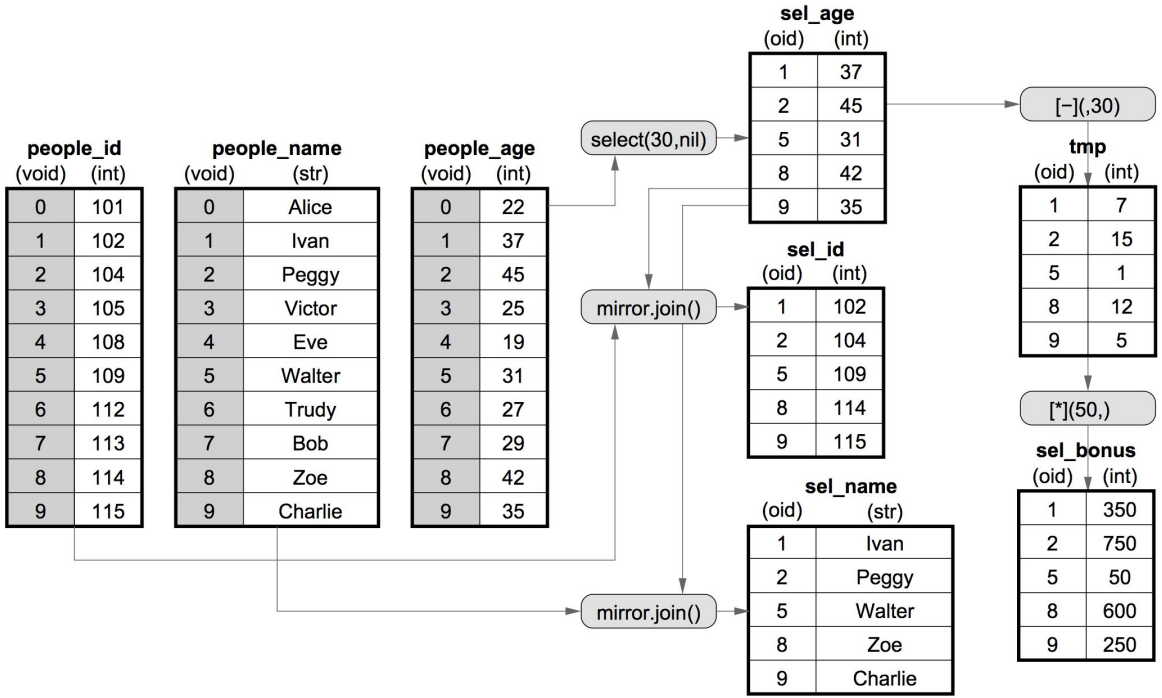
# Vectorized: Challenges

- Limitation of the Volcano Model
  - Tuple at a time
- Poor cache utilization
- Virtual function call overhead
  - next()
- **How to keep data fresh?**



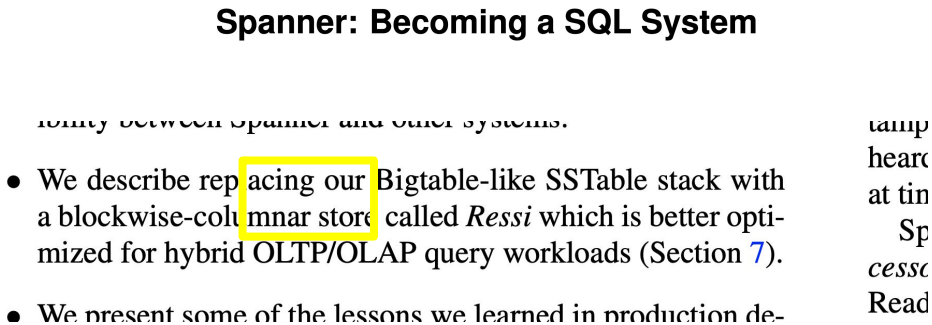
# Vectorized: From Tuple to Chunk

```
SELECT
      Id,
      Name,
      Age,
      (Age-30)*50 AS
Bonus
FROM People
WHERE Age > 30
```



# Workload Isolation

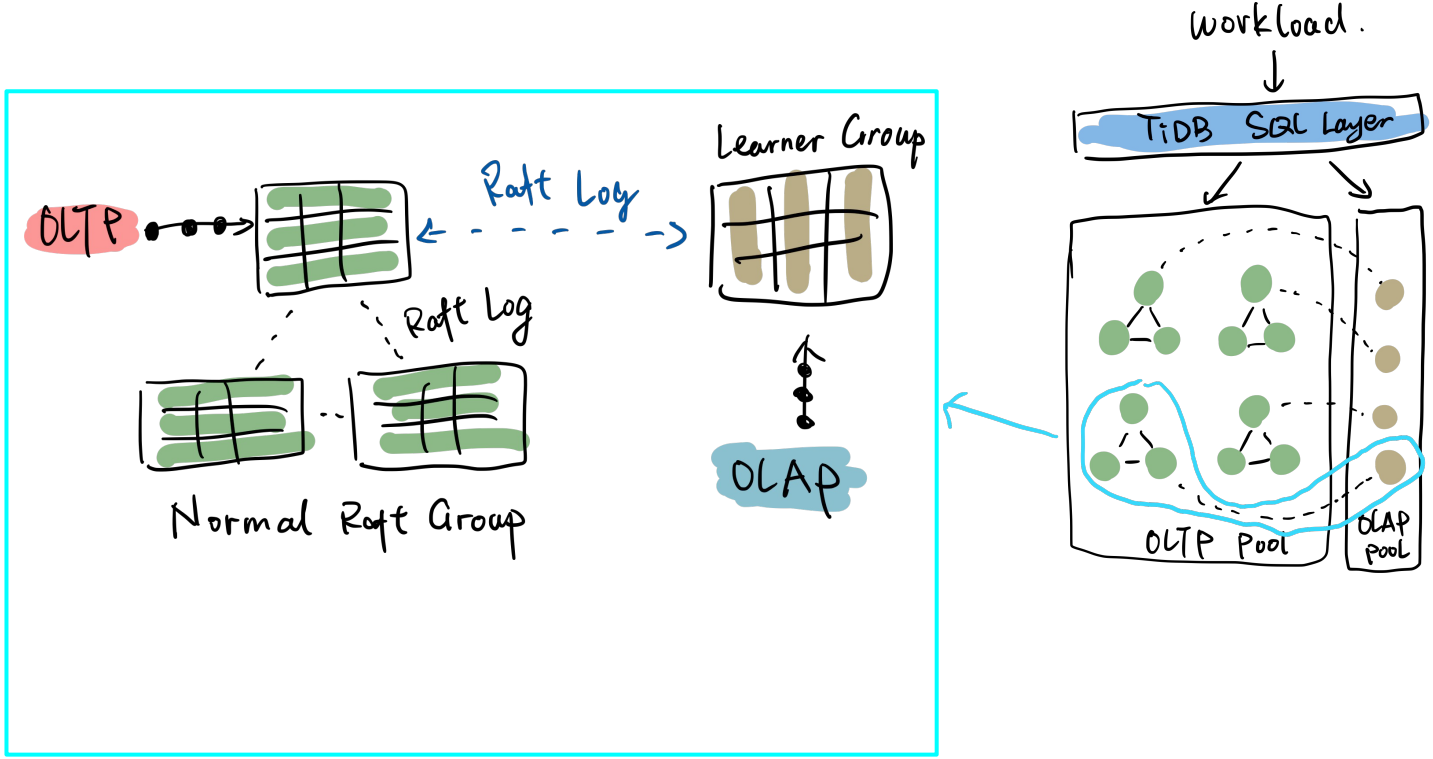
- What's the real trade-off?
- How to keep data fresh?
  - Raft
  - MVCC
  - MemStore
- TiFlash



**Spanner: Becoming a SQL System**

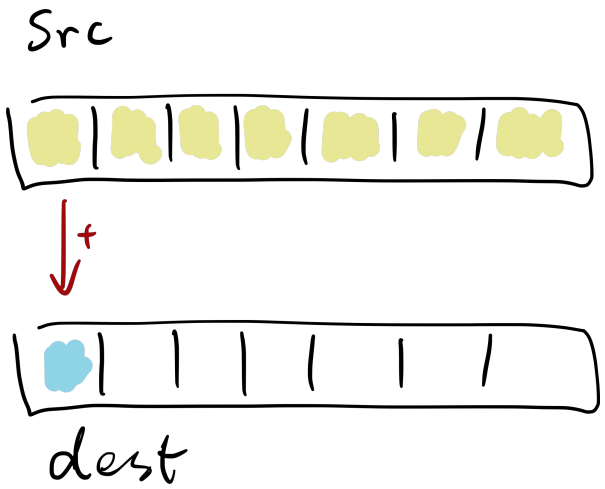
- We describe replacing our Bigtable-like SSTable stack with a blockwise-columnar store called *Ressi* which is better optimized for hybrid OLTP/OLAP query workloads (Section 7).
- We present some of the lessons we learned in production de-

# Workload Isolation



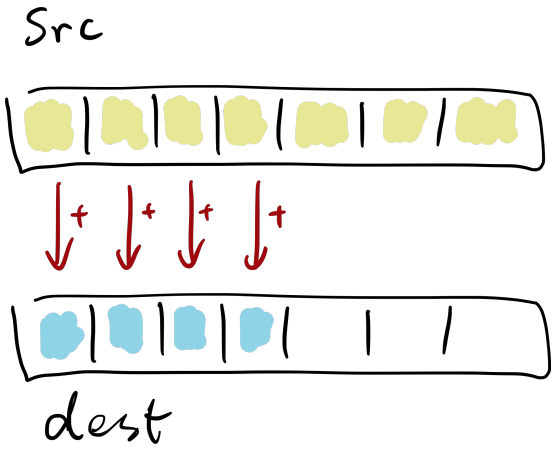
# SIMD

```
void plus(
  uint32_t * dest,
  uint32_t * src,
  size_t n
) {
  for i in 0..n {
    dest[i] += src[i];
  }
}
```

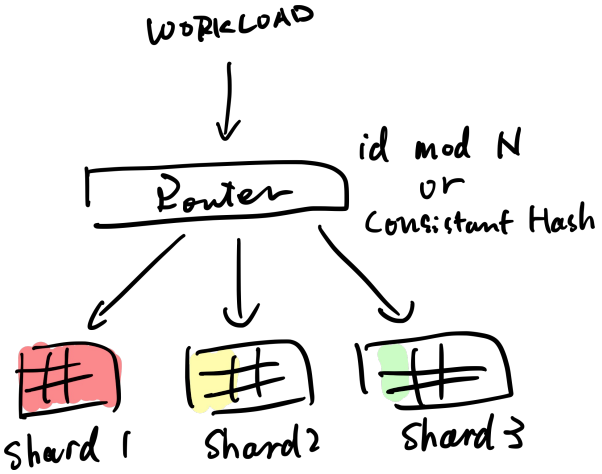


# SIMD

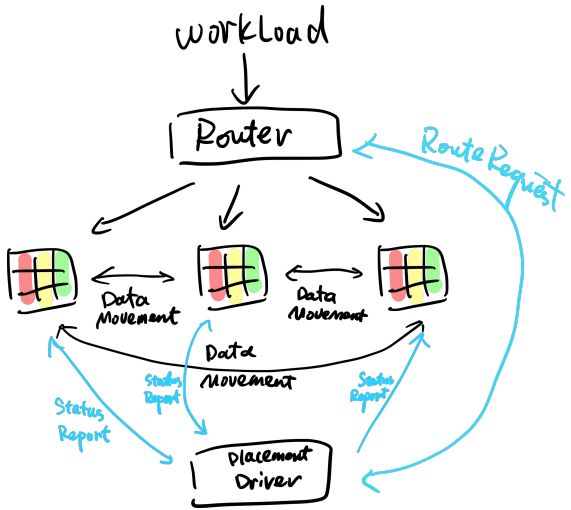
```
void plus(
  uint32_t * dest,
  uint32_t * src,
  size_t n
) {
  while ... {
    _mm_add_epi32(&dest[i],
&src[i]);
    i += 4;
  }
}
```



# Dynamic Data placement




VS



# Dynamic Data placement

- Flexible, no need to expose sharding details to users. Application development becomes simple and flexible
- Aware of the workload changes and response in real-time
- Logical partitioning based on business trait is more intuitive
- Challenges:
  - How to find hot spots in time
  - How to adjust the replica strategy more flexibly
    - Number of replicas, replica data placement, data structure of replica
  - Could AI help us?

# Storage and Computing Seperation

- **What are we talking about when we talk about Storage Computing Separation?**
  - Q: Is TiDB a Storage-Compute-Separation architecture?
- Pros:
  - The physical resources required by the storage and computing layers are different, separation is good for resource scheduling.
    - Stateless access layer (Like TiDB-Server instances, handling connections) is more convenient to expand on demand.
    - Operation and maintenance friendly, components can be upgraded on demand.

# Everything is Pluggable

- Computing
  - TiDB SQL
  - Spark SQL
- Storage
  - Local storage
    - TiFlash
    - ...
  - Multi-model data source
    - Unistore
    - ...

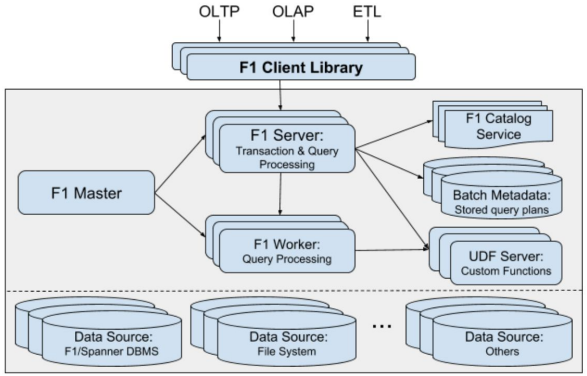
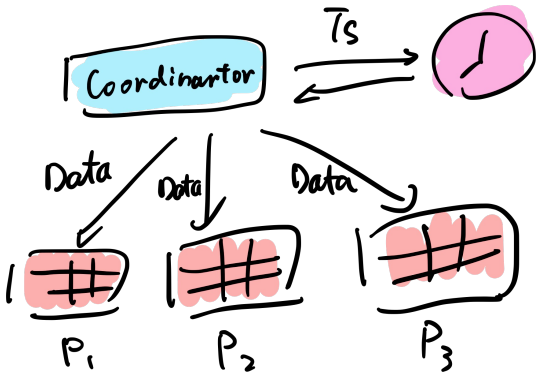


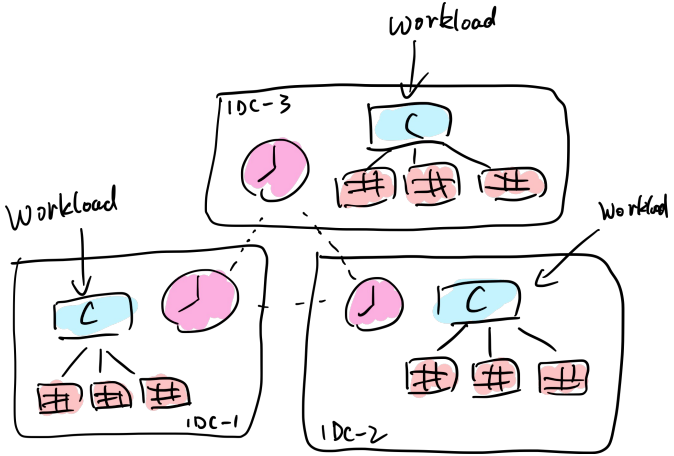

**Figure 1: Overview of F1 Federated Query Processing Platform**

# Distributed Transaction




- 2PC is still the only option
  - Timestamp is the best thing we got for now
- Challenges:
  - Reduce round-trips
    - Is it necessary to assign 2 timestamps for each transaction?
    - Is PD the only place to get timestamp?

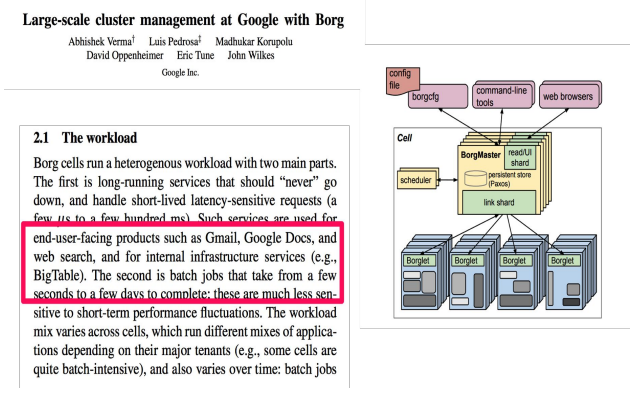
# Distributed Transaction: What we can do




- Follower Read (WIP)
  - Scale-out read performance without sacrificing consistency
- Optimizing Percolator model (DONE)
  - OptimizedCommitTS.tla
- Use wall clock like HLC?

# Cloud-Native Architecture

- DB is DB, NOTHING MORE, NOTHING LESS
- Putting cluster scheduling, resource allocation, and tenant isolation outside of the database kernel
- Integrating with the user's infrastructure
  - Kubernetes is winning



Large-scale cluster management at Google with Borg

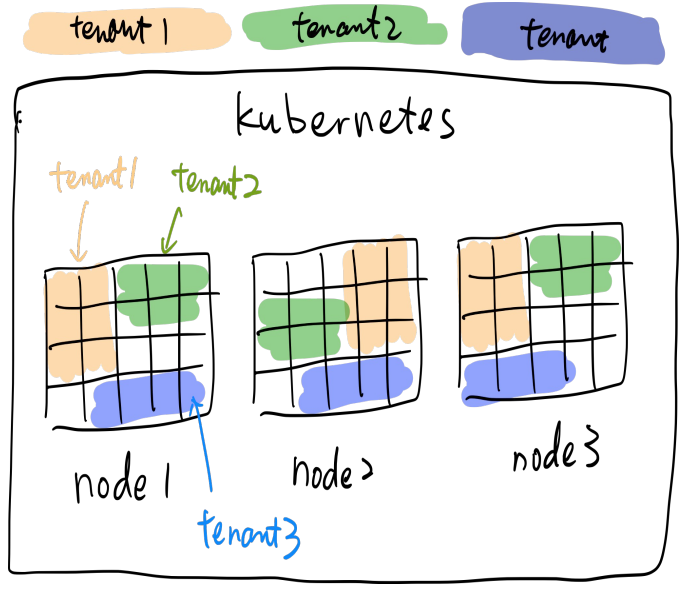
Abhishek Verma[†] Luis Pedrosa[‡] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes

Google Inc.

2.1 The workload

Borg cells run a heterogenous workload with two main parts. The first is long-running services that should "never" go down, and handle short-lived latency-sensitive requests (a few µs to a few hundred ms). Such services are used for end-user-facing products such as Gmail, Google Docs, and web search, and for internal infrastructure services (e.g., BigTable). The second is batch jobs that take from a few seconds to a few days to complete; these are much less sensitive to short-term performance fluctuations. The workload mix varies across cells, which run different mixes of applications depending on their major tenants (e.g., some cells are quite batch-intensive), and also varies over time: batch jobs



PingCAP

# Part V - Business trends

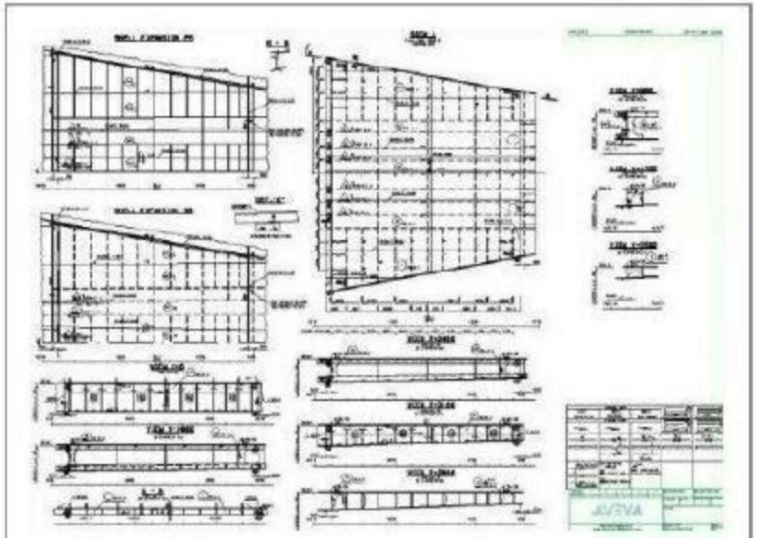
# What’s happening in China

野蛮
生长

# What’s happening in China

客户需求：造一条船，能过河就好

▽技术评估：这个项目起码需要1.5年



# What’s happening in China

▽老板发话：市场不等人，先上线再迭代，给技术团队1个月时间搞定！！！

▽测试团队：提前进入单元测试阶段

# What’s happening in China

▽测试团队：突击进入集成测试阶段

▽项目终于正式上线了

## What’s happening in China

# 对新技术赋能业务的期望更高

真实的两个故事，来自：
TiDB 的一个在某二线城市的客户
TiDB 的一个行业巨头客户

# What’s happening in China

传统行业互联网转型过程中的阵痛带来的机会

GIFAK.NET

# What’s happening in China

基础软件人才储备逐渐变强

# What’s happening in China

一些核心场景(银行核心系统)敢于使用国产技术

# 北京银行企业级NewSQL数据库赋能金融科技建设

发布时间：2018-03-27 16:37:49 | 来源：搜狐 | 作者： | 责任编辑：科学频道

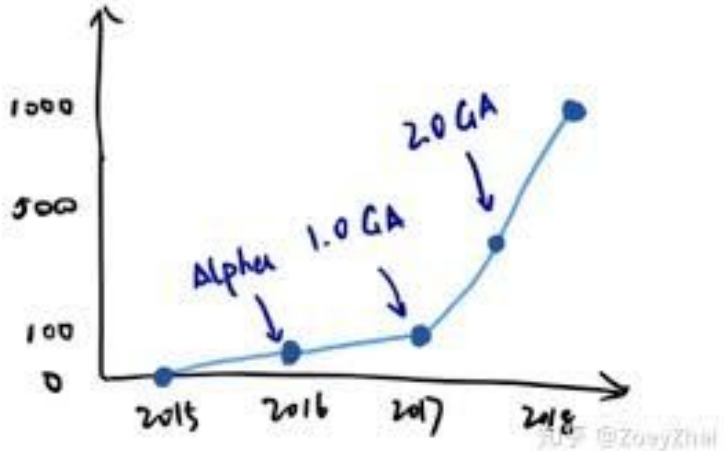
近年来，国家不断提高对信息技术安全可控的战略要求，银行业希望在快速发展业务的同时不断降低经营成本。这不仅促使商业银行积极提升自主掌控能力，也促使商业银行对基础软件的服务能力、软硬件升级成本控制提出新的要求。与此同时，面对互联网金融带来的交易复杂度及交易频次的大幅提升，商业银行信息系统采用的传统数据库一体化解决方案，在应对此类场景时遇到了明显的性能瓶颈，而提升系统性能只靠替换式的硬件升级，成本昂贵。在这种背景下，引入一种高性能、可弹性扩展、能够支持OLTP场景的数据库成为我行系统建设的优先选择方案。

## 一、分布式数据库的价值与应用场景

分布式事务数据库采用多种模式实现数据的分散存储，将数据库压力分散到不同服务器上，与集中式数

# What's happening in China

PingCAP 路径：
开源（互联网/社区）<-> 商业化



PingCAP

# Thank You !