PingCAP

Go in TiDB

Yao Wei | PingCAP

About me

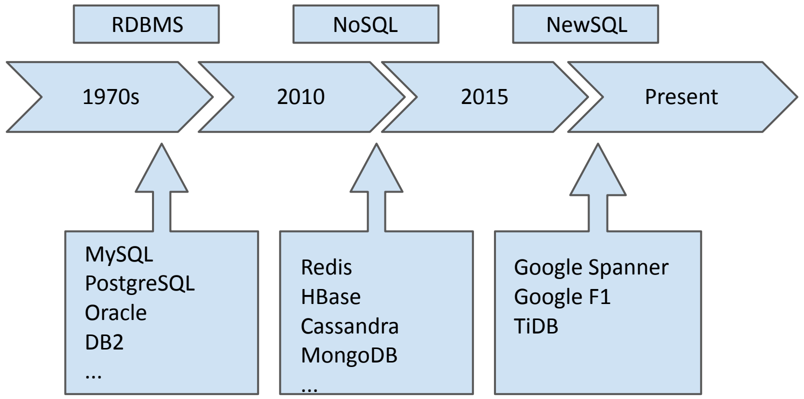
- Yao Wei (姚维)
- TiDB Kernel Expert, General Manager of South Region, China
- 360 Infra team / Alibaba-UC / PingCAP
- Atlas/MySQL-Sniffer
- Infrastructure software engineer

PingCAP

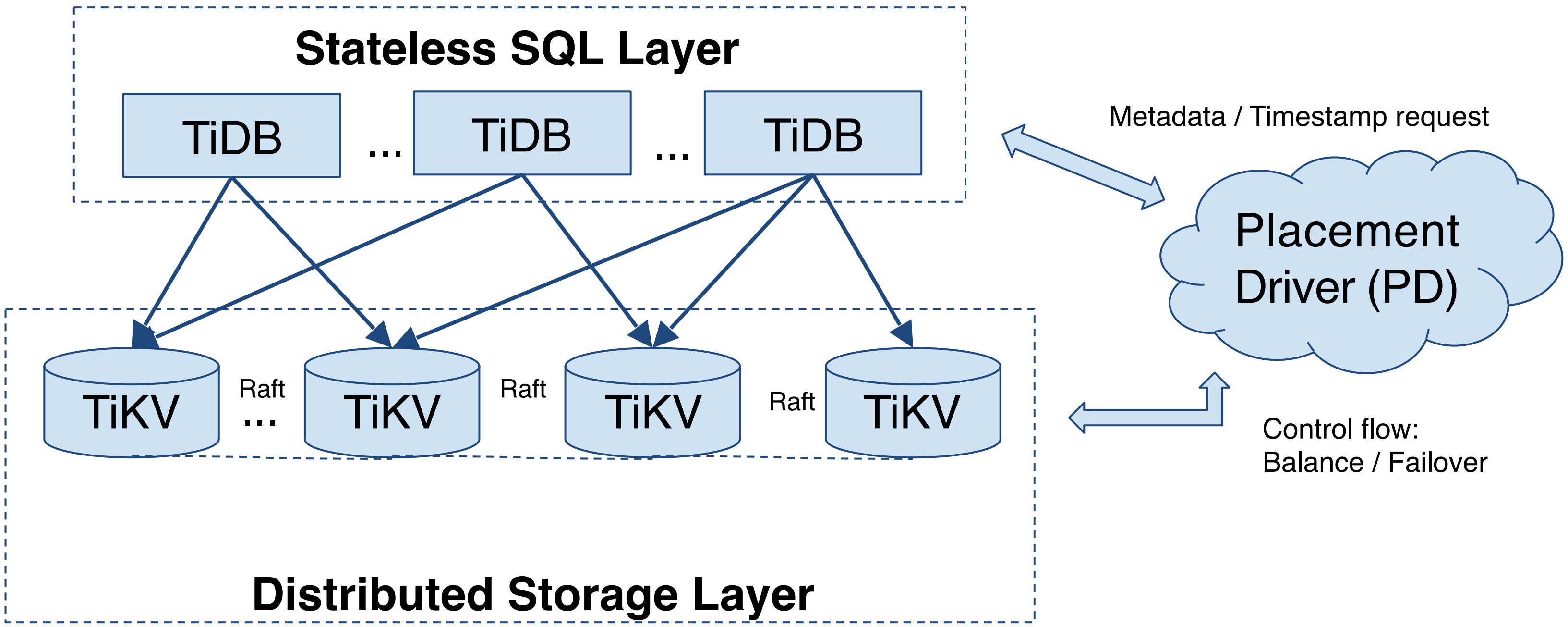
Why a new database?

Brief History

- Standalone RDBMS
- NoSQL
- Middleware & Proxy
- NewSQL

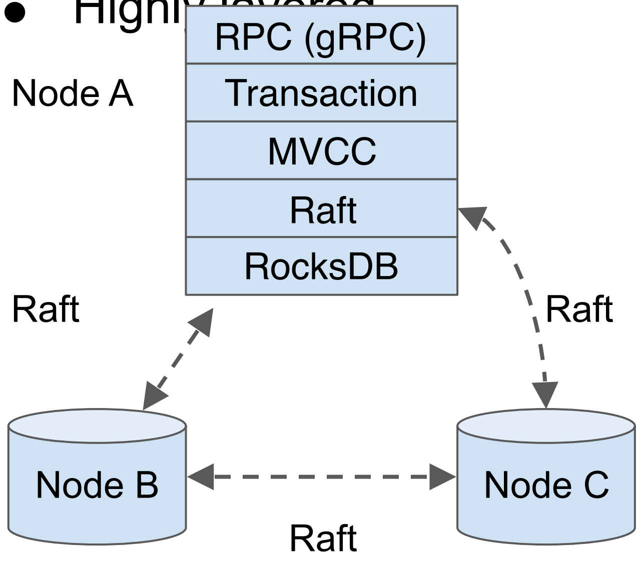


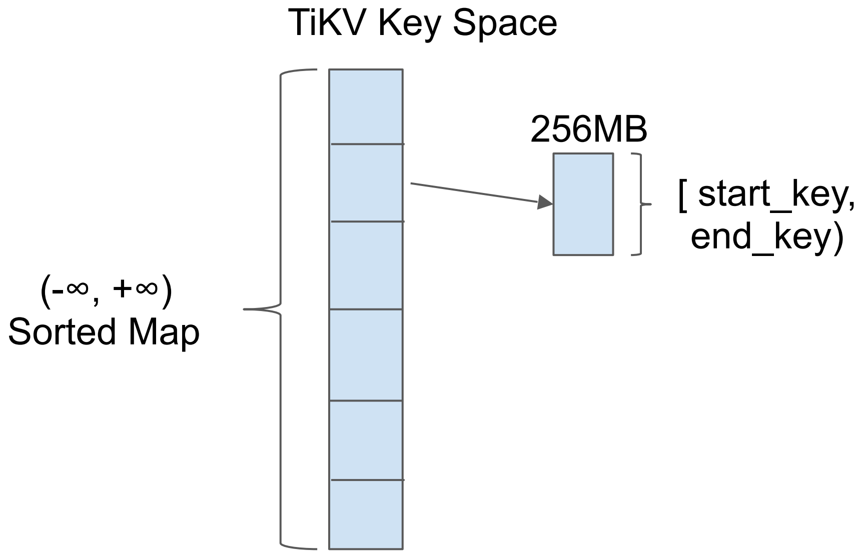
Architecture



TiKV - Overview

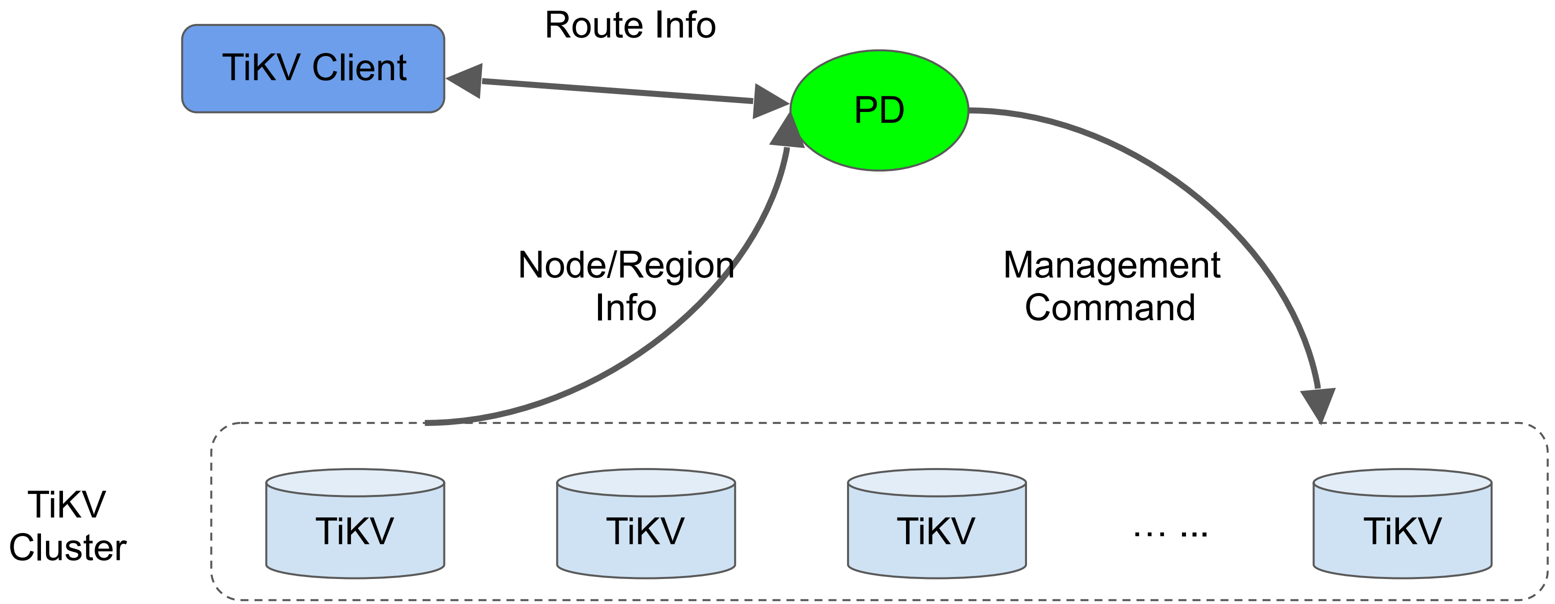
- Region: a set of continuous key-value pairs
- Data is organized/stored/replicated by Regions
- Highly layered





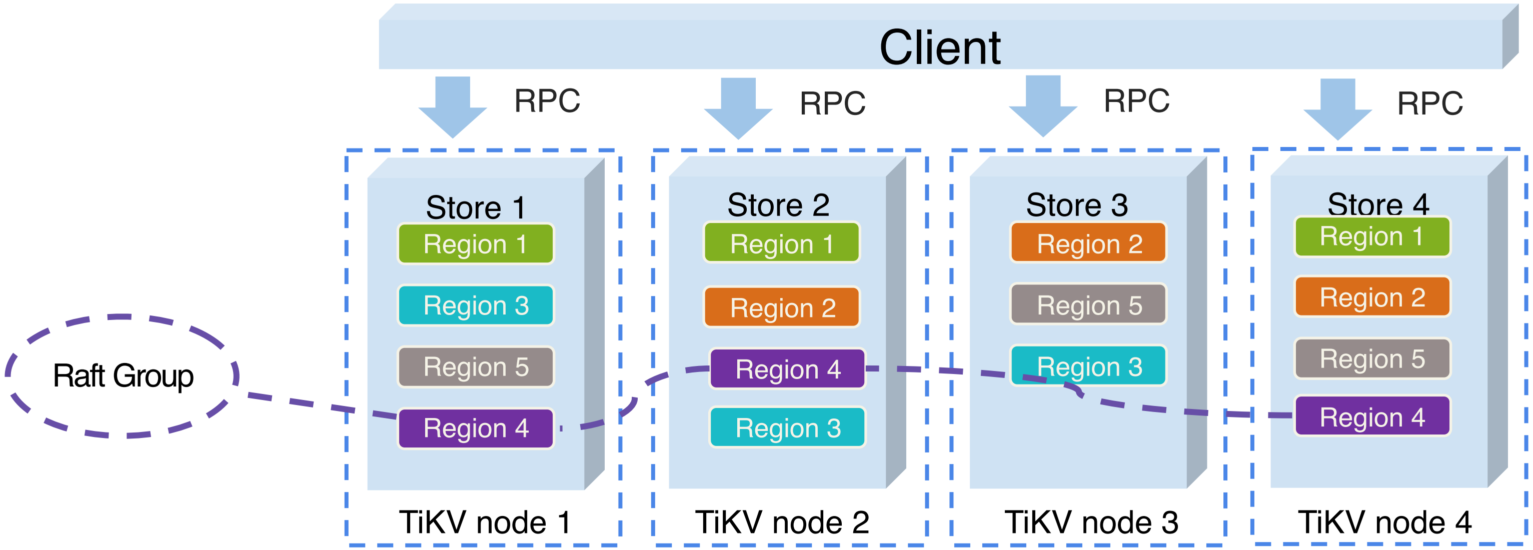
PD - Overview

- Meta data management
- Load balance management

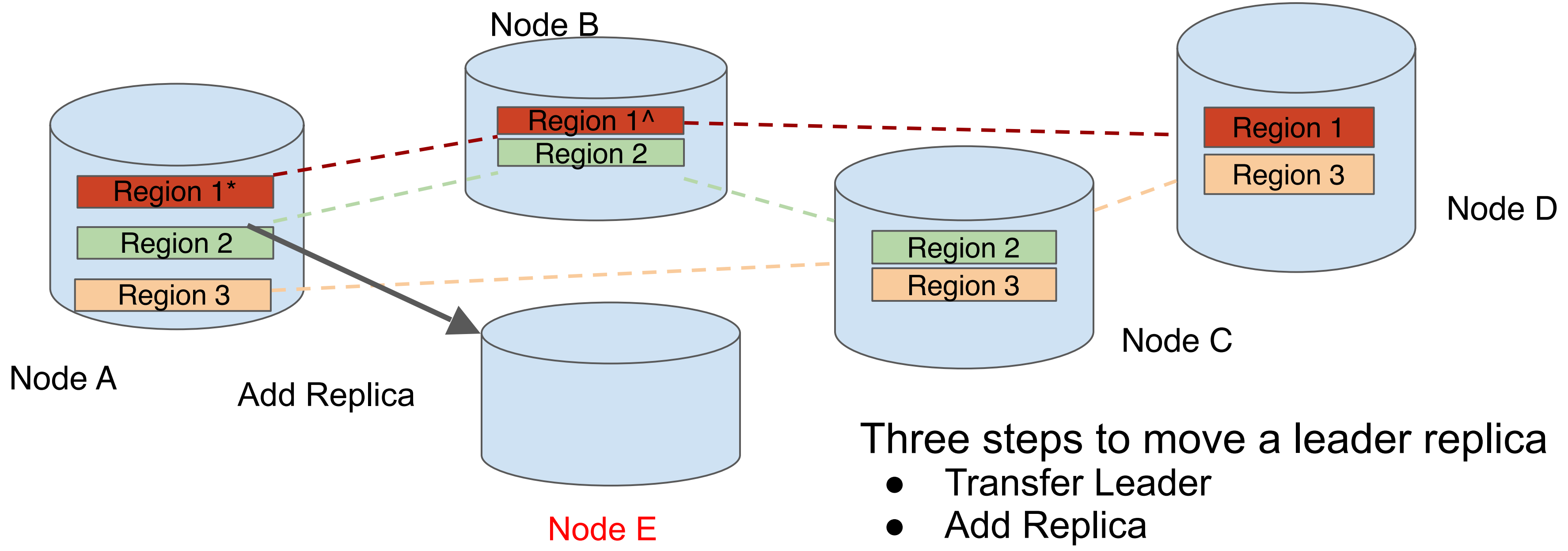


TiKV - Multi-Raft

Multiple raft groups in the cluster, one group for each region.



TiKV - Horizontal Scale

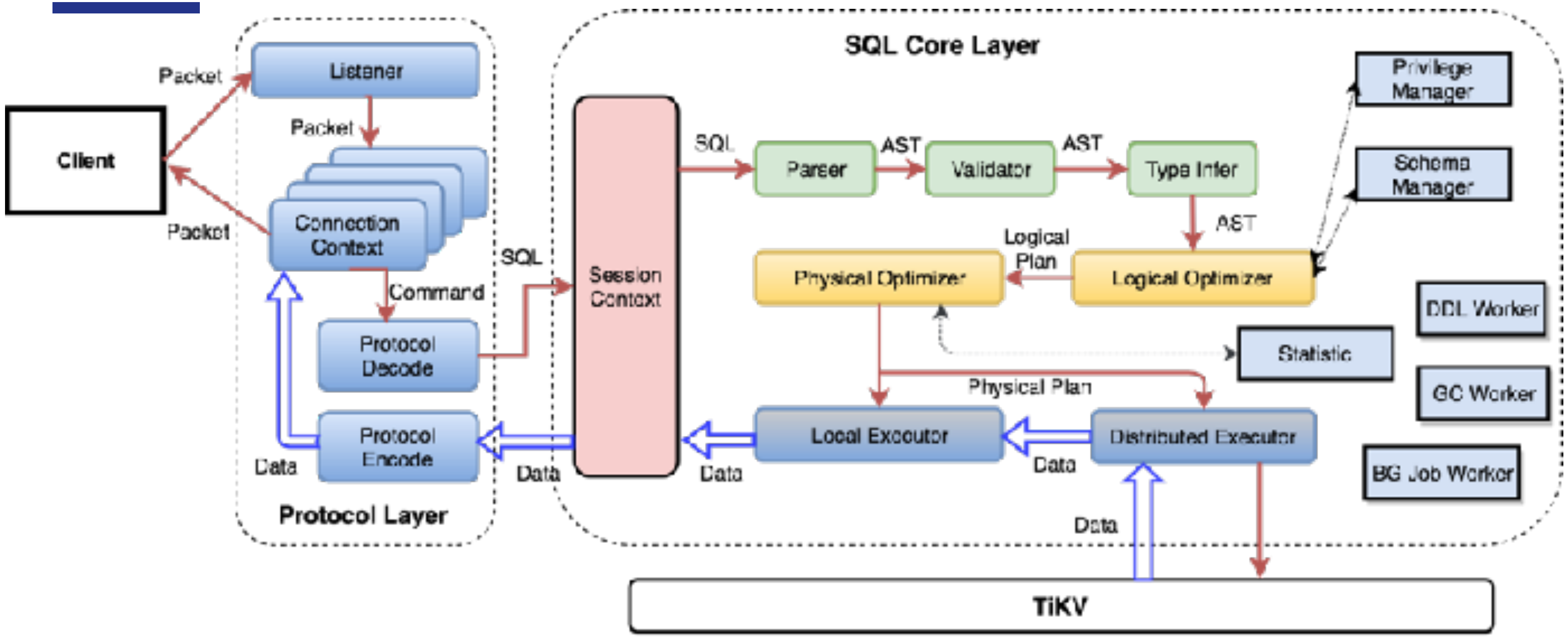


Three steps to move a leader replica

- Transfer Leader
- Add Replica
- Remove Replica

PingCAP

SQL Layer

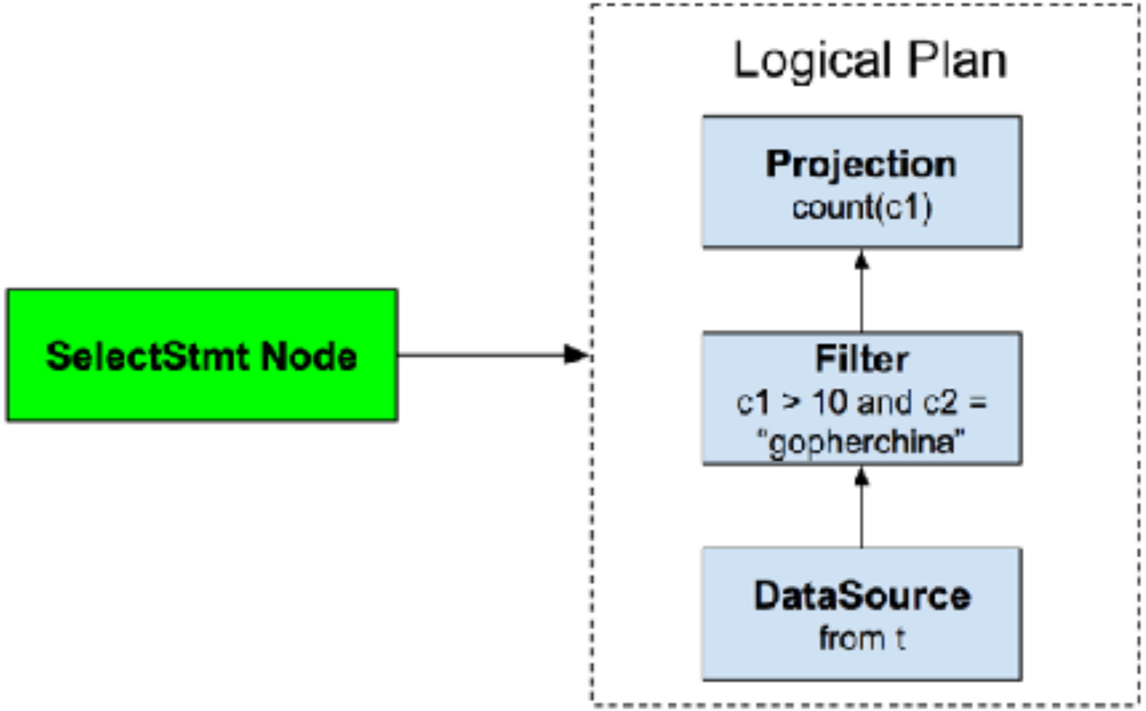


Example - SQL

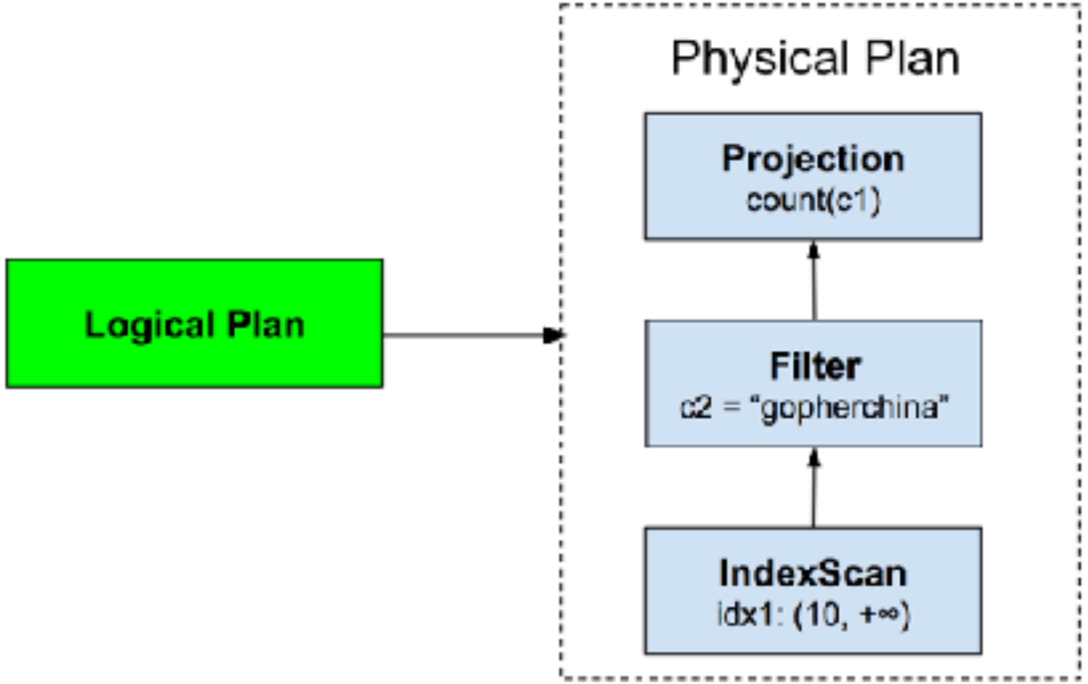
CREATE TABLE t (c1 INT, c2 VARCHAR(32), INDEX idx1 (c1));

SELECT COUNT(c1) FROM t WHERE c1 > 10 AND c2 =
“gopherchina”;

Example - Logical Plan



Example - Physical Plan



Challenges of distributed ACID database?

- Distributed Database is very complex
- Lots of RPC work
- Keep high performance
- Tons of data
- Huge amount of OLTP queries
- Very complex OLAP queries
- External Consistency
- SQL is much more complex than KV

PingCAP

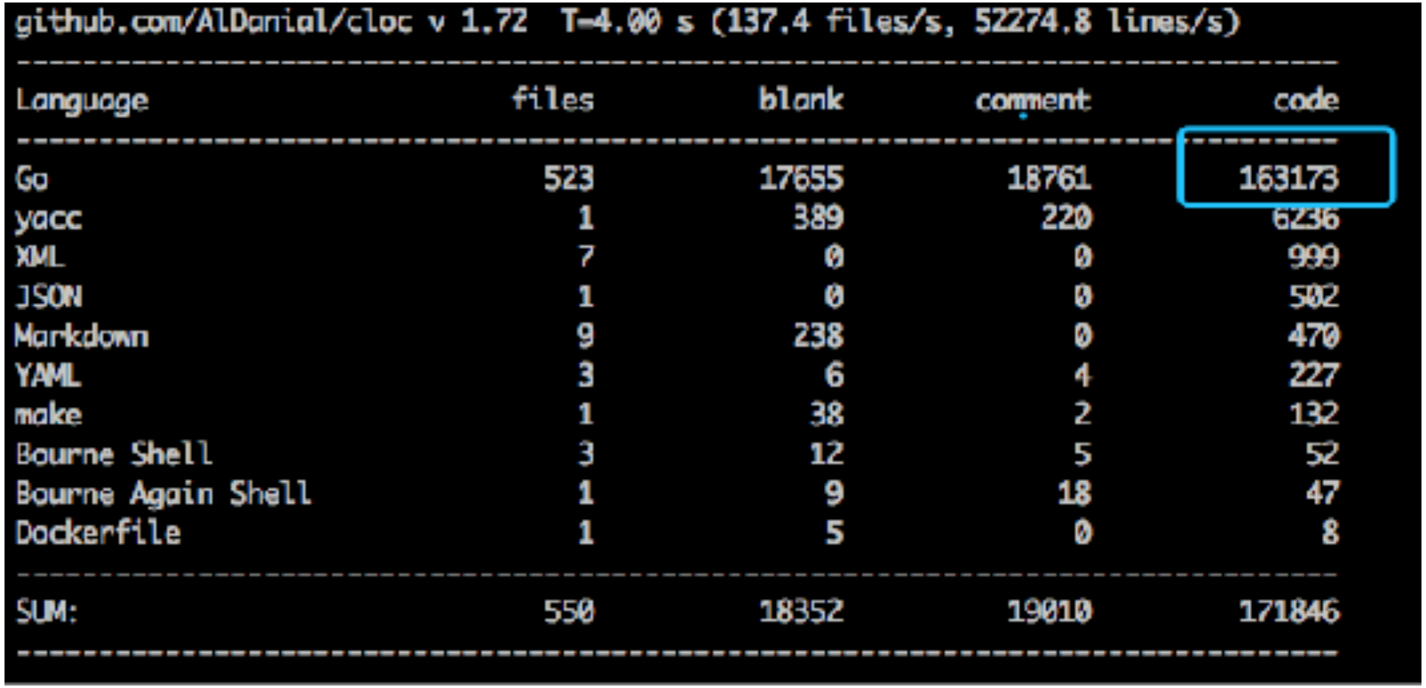
Why TiDB choose Golang?

- Easy-learning
- Productivity
- Concurrency
- Easy to trace bugs and profile
- Standard libraries and tools
- Tolerant GC latency
- Good performance
- Quick improvement

PingCAP

Go in TiDB

- More than 160K lines of Go code and 138 contributors.



```
github.com/AlDanial/cloc v 1.72  T=4.00 s (137.4 files/s, 52274.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Go                             523          17655          18761         163173
yacc                             1            389            220           6236
XML                              7              0              0            999
JSON                             1              0              0            502
Markdown                         9            238              0            470
YAML                             3              6              4            227
make                             1             38              2            132
Bourne Shell                     3             12              5             52
Bourne Again Shell               1              9             18             47
Dockerfile                       1              5              0              8
-------------------------------------------------------------------------------
SUM:                           550          18352          19010         171846
-------------------------------------------------------------------------------
```

Memory && GC

- Query may touch a huge number of data.
- Memory allocation may cost a lot of time.
- Put pressure on GC worker.
- Degrade the performance of SQL.
- OOM sucks!
- runtime.morestack

Reduce the Number of Allocation

- Get enough memory in one allocation operation

```
a := []int{1, 2, 3, 4, 5}
b := []int{}
// a much better way:
// b := make([]int, 0, len(a))
for _, i := range a {
	b = append(b, i)
}
```

Reuse Object

- Share a stack for all queries in one session
- Introduce a cache in goyacc
- Resource pool

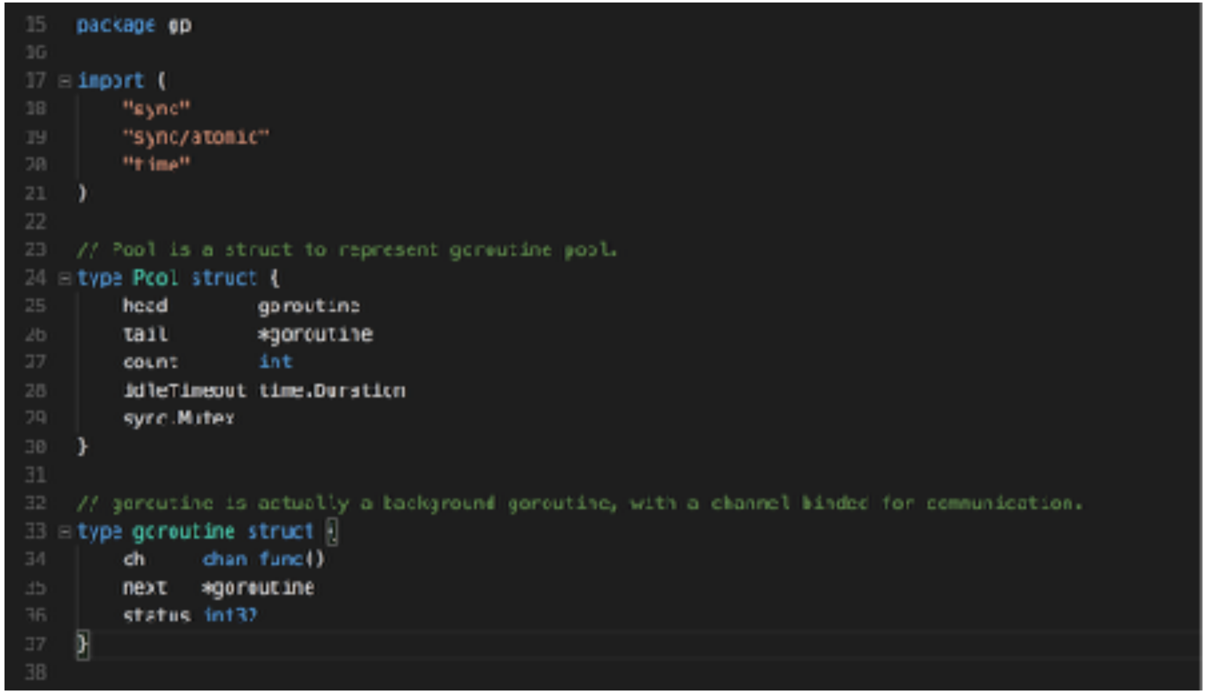
Reduce runtime.morestack

```
var growStack = false

func growStack() {
    var buf [16 << 10] /* 16 KB */ byte
    if growStack {

        for i := range buf {
            buf[i] = byte(i)
        }
        groupStack = true
    }
}
```

Not enough!

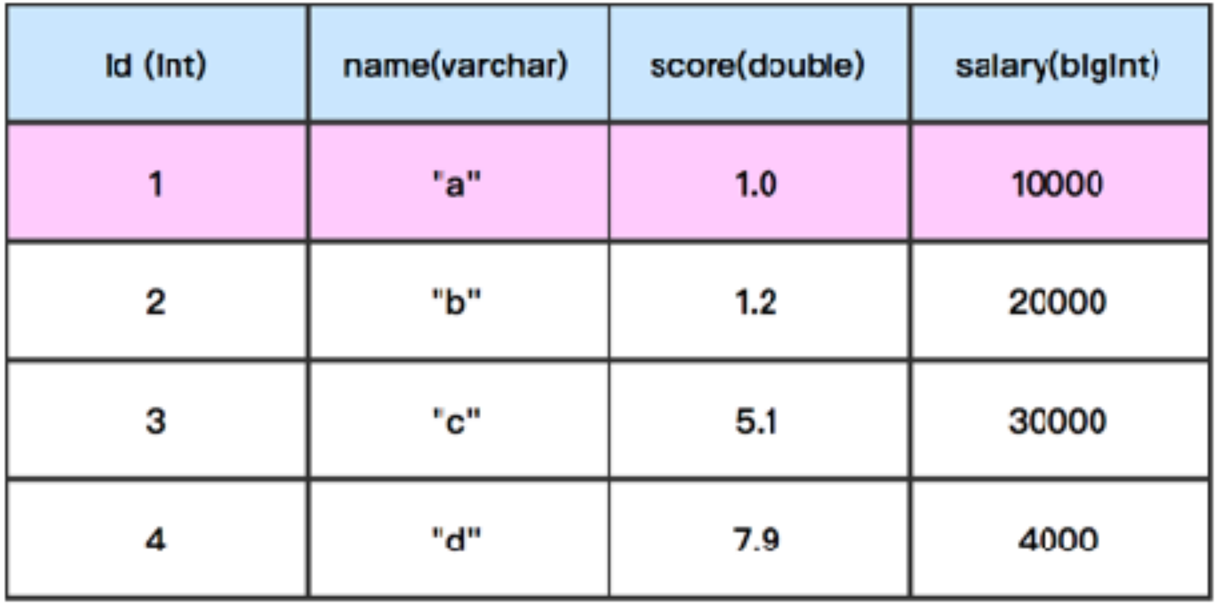
Goroutine Pool

tidb/util/goroutine_pool/gp.go:



```go
package gp

import (
	"sync"
	"sync/atomic"
	"time"
)

// Pool is a struct to represent goroutine pool.
type Pool struct {
	head        goroutine
	tail        *goroutine
	count       int
	idleTimeout time.Duration
	sync.Mutex
}

// goroutine is actually a background goroutine, with a channel binded for communication.
type goroutine struct {
	ch     chan func()
	next   *goroutine
	status int32
}
```

Row Memory Format



Id (Int)	name(varchar)	score(double)	salary(bigInt)
1	"a"	1.0	10000
2	"b"	1.2	20000
3	"c"	5.1	30000
4	"d"	7.9	4000

Row Memory Format

- How to store the row in memory?
- Union? Json? Protobuf?
- Use Datum to store one Column.

```
type Datum struct {
 k         byte          // datum kind.
 collation uint8         // collation can hold uint8 values.
 decimal   uint16        // decimal can hold uint16 values.
 length    uint32        // length can hold uint32 values.
 i         int64         // i can hold int64 uint64 float64 values.
 b         []byte        // b can hold string or []byte values.
 x         interface{}   // x hold all other types.
 }
```

Row Memory Format

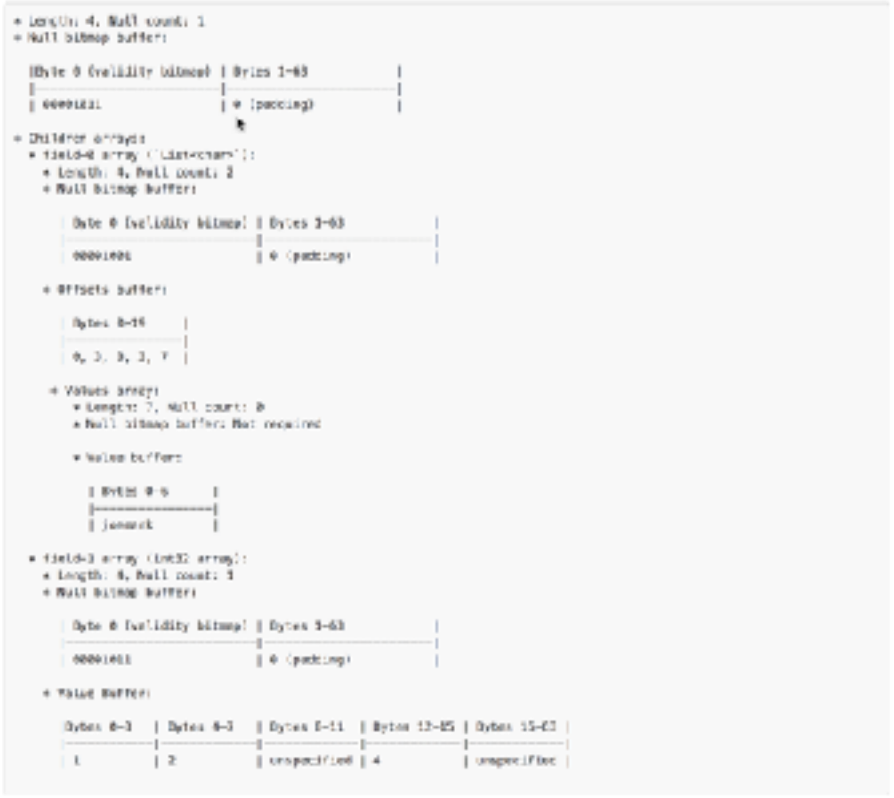
- **What is the disadvantage about Datum?**
  - Use unnecessary memory in every column.
  - Must use assert to get complex types:

```
func (d *Datum) GetMysqlDecimal() *MyDecimal {
		return d.x.(*MyDecimal)
}
```

  - Impossible to do vectorizable serial computation
- So how to store multi types data in golang?

Apache Arrow

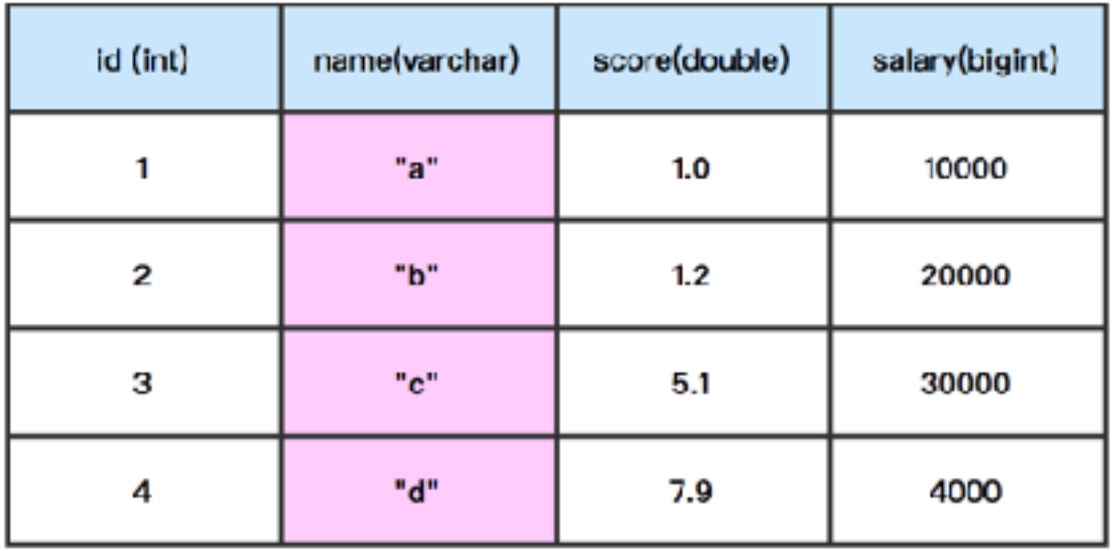
- binary data format
- Array lengths
- Null count
- Null bitmaps
- Offsets buffer
- Values Array
- more details plz see the docs



```
* Length: 4, Null count: 1
* Null bitmap buffer:

  |Byte 0 (validity bitmap) | Bytes 1-63            |
  |-------------------------|-----------------------|
  | 00001011                | 0 (padding)           |

* Children arrays:
  * field-0 array (`List<char>`):
    * Length: 4, Null count: 2
    * Null bitmap buffer:

      | Byte 0 (validity bitmap) | Bytes 1-63            |
      |--------------------------|-----------------------|
      | 00001001                 | 0 (padding)           |

    * Offsets buffer:

      | Bytes 0-19     |
      |----------------|
      | 0, 3, 3, 3, 7  |

     * Values array:
        * Length: 7, Null count: 0
        * Null bitmap buffer: Not required

        * Value buffer:

          | Bytes 0-6      |
          |----------------|
          | joemark        |

  * field-1 array (int32 array):
    * Length: 4, Null count: 1
    * Null bitmap buffer:

      | Byte 0 (validity bitmap) | Bytes 1-63            |
      |--------------------------|-----------------------|
      | 00001011                 | 0 (padding)           |

    * Value Buffer:

      |Bytes 0-3   | Bytes 4-7   | Bytes 8-11  | Bytes 12-15 | Bytes 16-63 |
      |------------|-------------|-------------|-------------|-------------|
      | 1          | 2           | unspecified | 4           | unspecified |
```

Row Memory Format

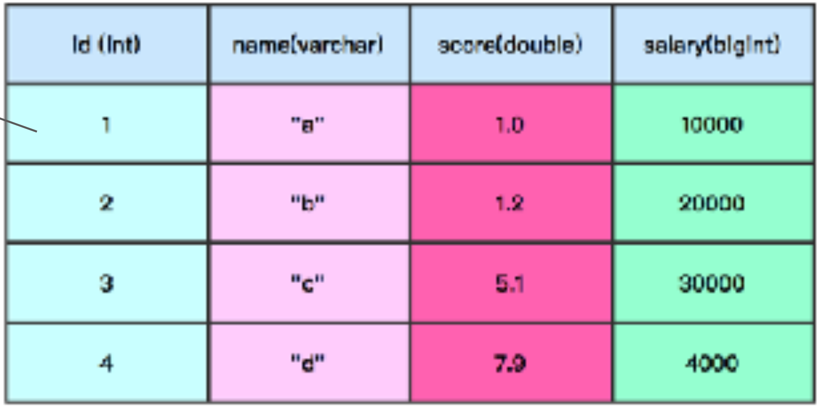
- We can eliminate the arrow’s offsets array.
- Column store VS Row store.



id (int)	name(varchar)	score(double)	salary(bigint)
1	"a"	1.0	10000
2	"b"	1.2	20000
3	"c"	5.1	30000
4	"d"	7.9	4000

Chunk

```
type Chunk struct {
        columns []*column
}

type column struct {
        length     int
        nullCount  int
        nullBitmap []byte
        offsets    []int32
        data       []byte
        elemBuf    []byte
        ifaces     []interface{}
}
```



Id (Int)	name(varchar)	score(double)	salary(bigInt)
1	"a"	1.0	10000
2	"b"	1.2	20000
3	"c"	5.1	30000
4	"d"	7.9	4000

Chunk

- **Use unsafe.pointers to get complex types:**

```
func (r Row) GetMyDecimal(colIdx int) *types.MyDecimal {
	col := r.c.columns[colIdx]
	return (*types.MyDecimal)(unsafe.Pointer(&col.data[r.idx*types.MyDecimalStructSize]))
}

func (r Row) GetUint64(colIdx int) uint64 {
	col := r.c.columns[colIdx]
	return *(*uint64)(unsafe.Pointer(&col.data[r.idx*8]))
}
```

PingCAP

Chunk

- **Vectorized Execute expressions:**

```
func VectorizedExecute(ctx context.Context, exprs []Expression, input, output *chunk.Chunk) error {
	sc := ctx.GetSessionVars().StmtCtx
	for colID, expr := range exprs {
			err := evalOneColumn(sc, expr, input, output, colID)
			if err != nil {
					return errors.Trace(err)
			}
	}
	return nil
}

func evalOneColumn(sc *stmtctx.StatementContext, expr Expression, input, output *chunk.Chunk, colID int) (err error)
{
	switch fieldType, evalType := expr.GetType(), expr.GetType().EvalType(); evalType {
	case types.ETInt:
			for row := input.Begin(); err == nil && row != input.End(); row = row.Next() {
					err = executeToInt(sc, expr, fieldType, row, output, colID)
			}

	.....
}
```

PingCAP

Thanks!

Contact me:

wink@pingcap.com
www.pingcap.com
Wechat: 82091309

12.16 深圳| Gopher meetup

该二维码7天内(12月22日前)有效，重新进入将更新