Vitess

Scaling MySQL with Go

{mike|sougou}@youtube

code.google.com/p/vitess

Briefly

- motivation & vision
- implementation strategy
- state of the system
- Go experience
- questions, answers?

Begin at the end

- MySQL-esque*
- Self-managing without magic**
- Increased efficiency***
  - memory usage, throughput
- External replication

Labor vs Management

- Automated reparenting
- Online schema apply*
  - alters, rebuilds
- Auto-sharding
  - incremental provisioning

Assumptions/Constraints

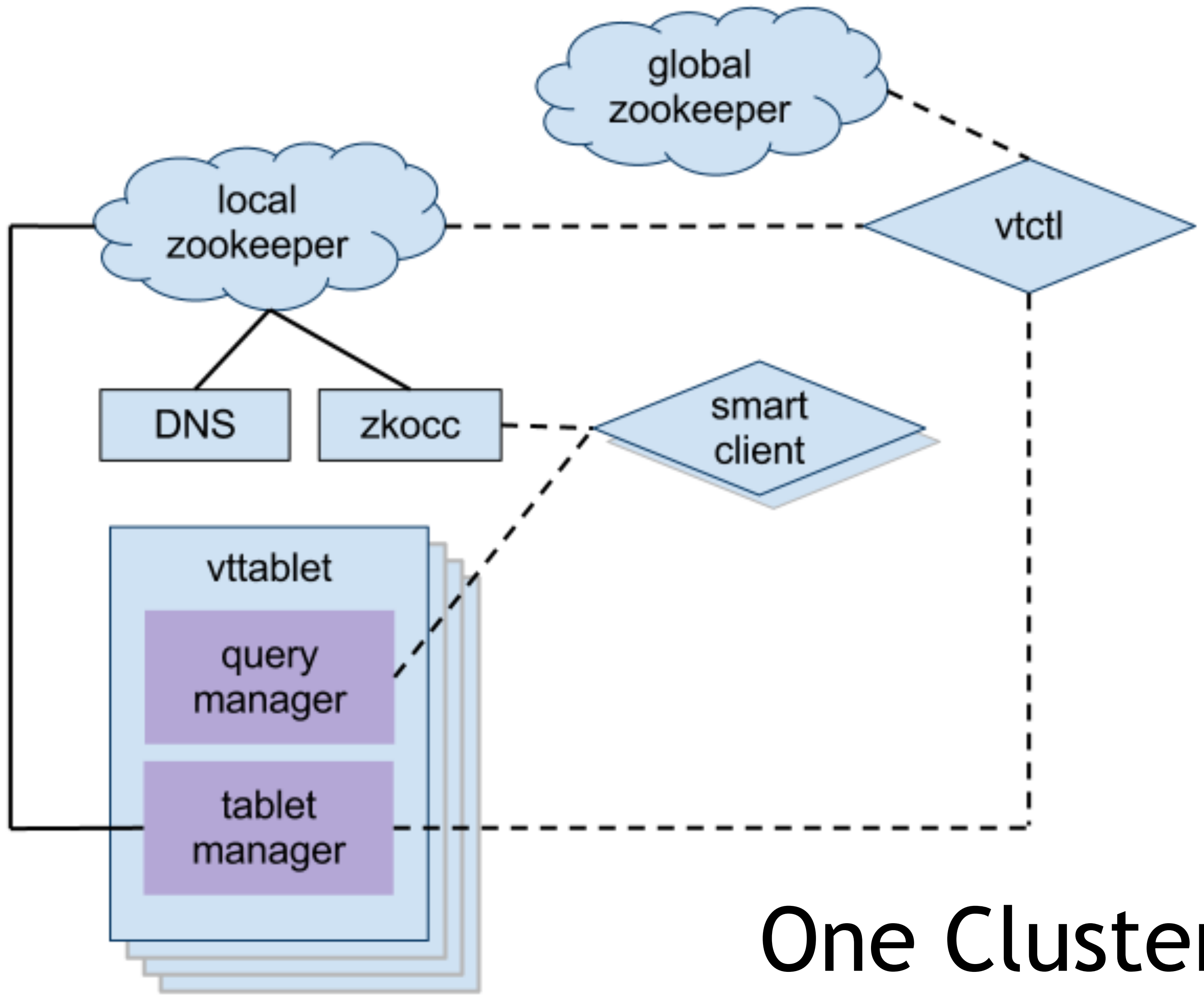
- keyspaces / keyspace_id
- range-based shards
- "shards x replicas" structure
- no cross-shard transactions*
- eventual consistency

The Tao of Vitess*

"choose the simplest solution with the loosest guarantees that are practical"

Implementation Strategy

- minimal changes to MySQL
- external query shaper
- external tablet manager
- coordinate in Zookeeper*

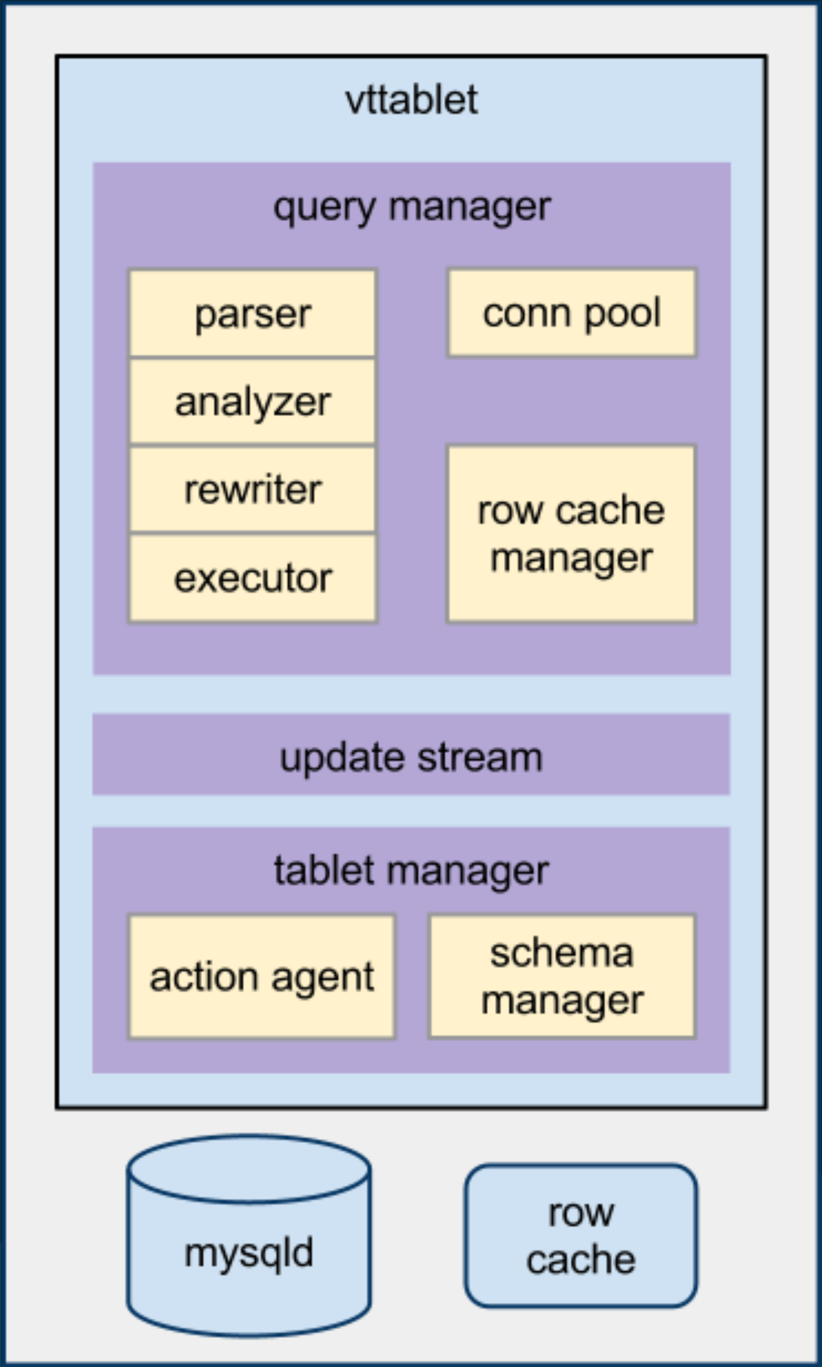

global zookeeper
local zookeeper
vtctl
DNS
zkocc
smart client
vttablet
query manager
tablet manager

One Cluster

Zookeeper

- local vs global
  - sparing use of global
- zkocc (insulation)
  - cell hiccups
  - zk protocol

One Machine


vttablet
query manager
parser
analyzer
rewriter
executor
conn pool
row cache manager
update stream
tablet manager
action agent
schema manager
mysqld
row cache

vttablet

vttablet

- Full query service
  - +streaming
- Tablet management
  - Stored directly in zookeeper
- Update stream

Tablet Mgmt

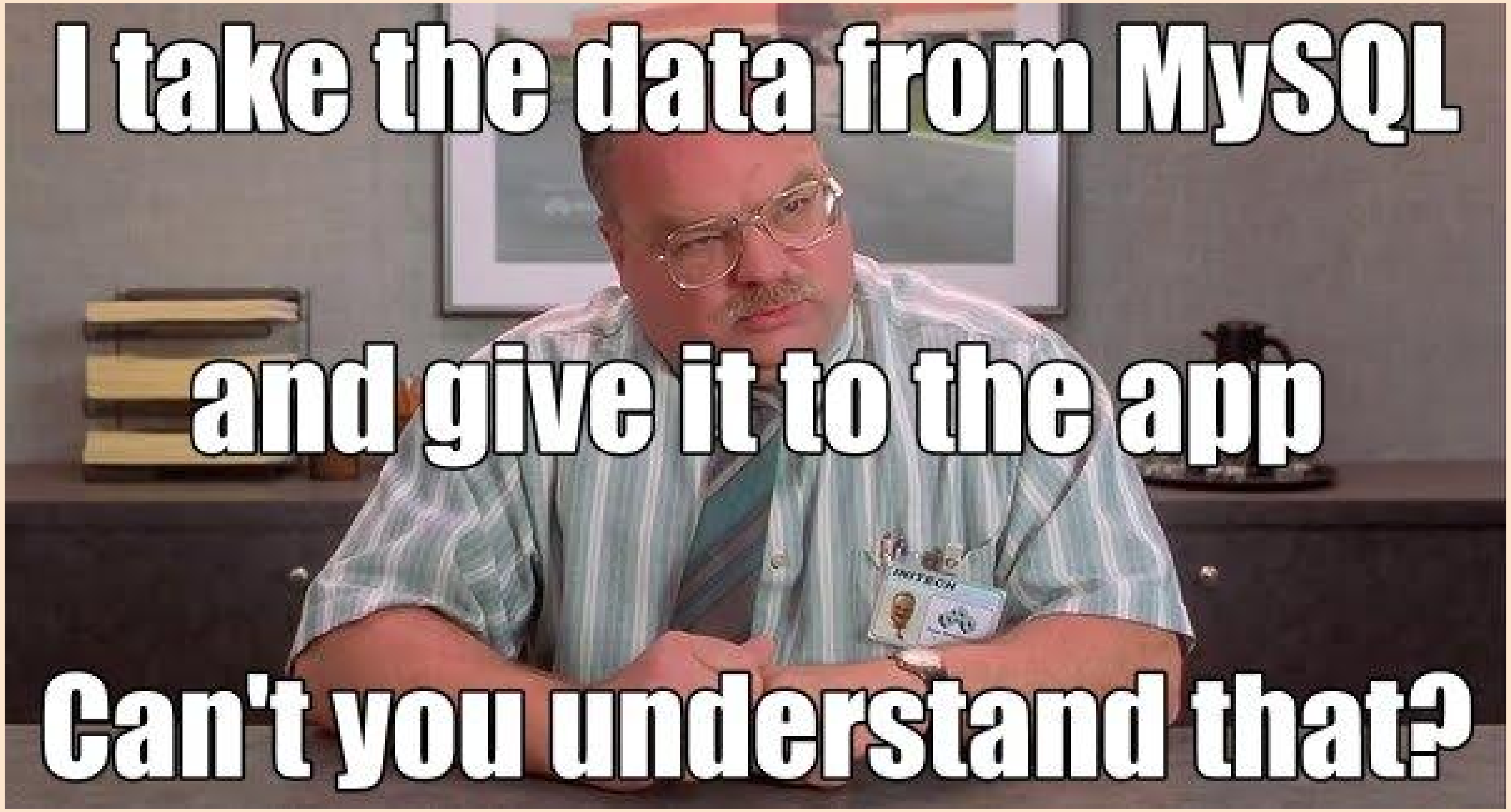
- Actions
  - reliably queued
  - communicated via zookeeper
  - performed by vtaction*
- "manholes"

Update Stream

- primary key change notifications
- derived from binlog*
- eventual consistency
    - out-of-order

Query Server (vtocc)

- RPC front-end to MySQL
- QoS Connection pooling
- Transaction management
- DML annotations
- SQL parser


I take the data from MySQL
and give it to the app
Can't you understand that?

In production

- Serves all of YouTube's MySQL queries
- Months of crash-free & leak-free operation
- Zero downtime restarts
- Configuration, Logging & Statistics

Fail-safes

- Query consolidation
- Row count limit
- Transaction limit
- Query and transaction timeouts
- Easy to add more

Row Cache

- vs Buffer Cache
- CPU usage
- Primary Key fetches
- Results merging using subqueries
- Benchmarks

Upcoming Features

- Online Resharding
- Schema Management
- Joins for rowcache
- Query Router
- Configurable QRewriter*

Go Experience

- Highlights
- Lowlights
- Deploy

General Productivity

- Falls between C and Python
- Fast compile/test cycle
- Batteries included and fully charged

Expressive

- auto-rotating logfile: 105 lines
- connection pooler: 227 lines
- memcache client: 250 lines*

Language Features

- interfaces
- first class concurrency
- defers and closures
- CGO
- channels* (select)

Lowlights

- []byte vs string
- error vs panic
- GC for large footprints
- scheduler overhead

Pitfalls - nil interface

```
func x() error {
  var err *MyError
  return err
}
```

Is x() == nil?

Pitfalls - range vars

```
for _, url := range urls {
  go func() {
    fmt.Println(url)
  }()
}
```

How many urls get printed?

Deploying

- statically linked
  - until you use CGO
  - handy for prod debugging
- debug web server / profiling
- SIGABRT
- relatively strace friendly

New Committers

- Alain Jobart
- Shruti Patil
- Ric Szopa

We're hiring!

Questions?

- Vitess: http://code.google.com/p/vitess
- Go: http://golang.org