# MySQL/Percona 5.6/5.7 秒杀场景优化

杨德华@PHPCON 2017

# 个人介绍

- 杨德华
- 曾供职于阿里集团数据库技术团队
- 经历5年双十一MySQL大考
- 目前供职 杭州数列科技
- 微信号:whitepoplar

# 压测场景

- 压测工具 sysbench0.5（官方认可的压测工具）
- 事务隔离级别均为READ-COMMITTED. CPU 32cores. BP15G。数据放在SSD，日志放在SAS盘。
- thread_pool_size=16;thread_pool_oversubscribe=1; innodb_thread_concurrency=32;
- CREATE TABLE `test_update` (
- `id` int(11) NOT NULL AUTO_INCREMENT,
- `stock` int(11) DEFAULT NULL,
- PRIMARY KEY (`id`)
- ) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8
- insert into item.test_update (id,stock) value(1,500000);

# 压测场景

- 更新语句update test_update set stock =stock -1  where id = 1 and stock >3;

# 5.6版本的压测数据

| 并发连接数 | 版本5.6.19 | 版本5.6.19+关闭死锁 | Percona5.6+线程池 | Percona5.6+线程池+关闭死锁 |
|---|---|---|---|---|
| 256 | 2271.91 | 9198.25 | 7925.89 | 8229.47 |
| 512 | 646.53 | 6225.06 | 7587.56 | 7608.73 |
| 1024 | 30 | 3924.82 | 6717.44 | 6925.37 |
| | | | | |

- 更新语句update test_update set stock =stock -1  where id = 1 and stock >3;

# 5.7版本压测数据

| 并发 | MySQL5.7.18死锁检测打开 | MySQL5.7.18死锁检测关闭 | Percona5.1.17-15死锁检测打开 | Percona5.1.17-15死锁检测关闭 |
|---|---|---|---|---|
| 256 | 2000 | 6000 | 4000 | 4200 |
| 512 | 500 | 2500 | 4300 | 4350 |
| 1024 | 30 | 1500 | 5000 | 5200 |

- 更新语句update test_update set stock =stock -1  where id = 1 and stock >3;

# 秒杀场景解读

- 秒杀/热点更新场景
  - 互联网电商常见场景
    - 活动预热->商品聚集大量人气->瞬间下单

# 生活中的场景

- 思考:生活中有哪些场景和秒杀类似?

# 场景抽象

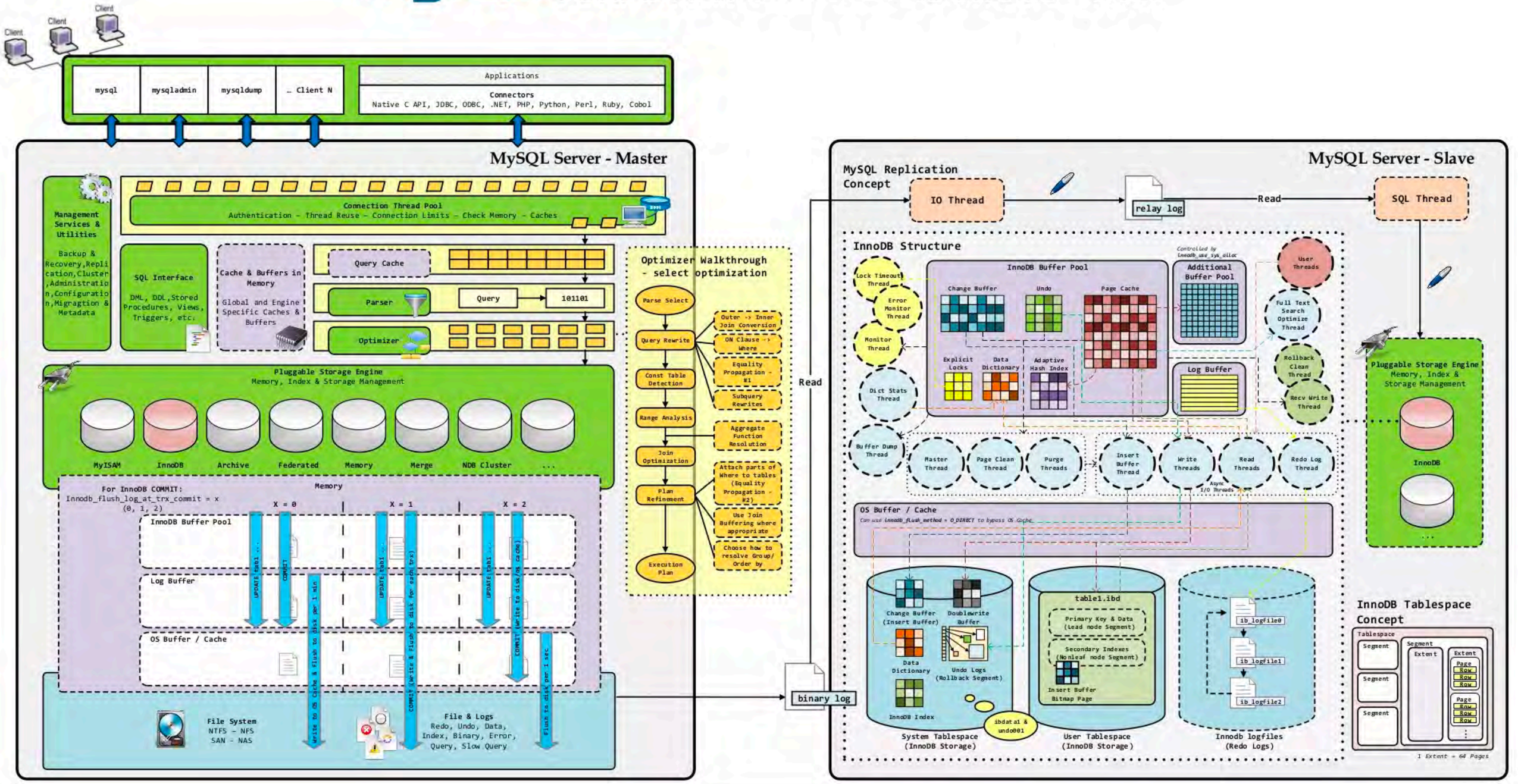
- 处理请求是并行，DB节点最后变成了串行
- 多个应用节点向一个DB节点的一行发送数据处理请求

- DB要保证数据的ACID以及高性能

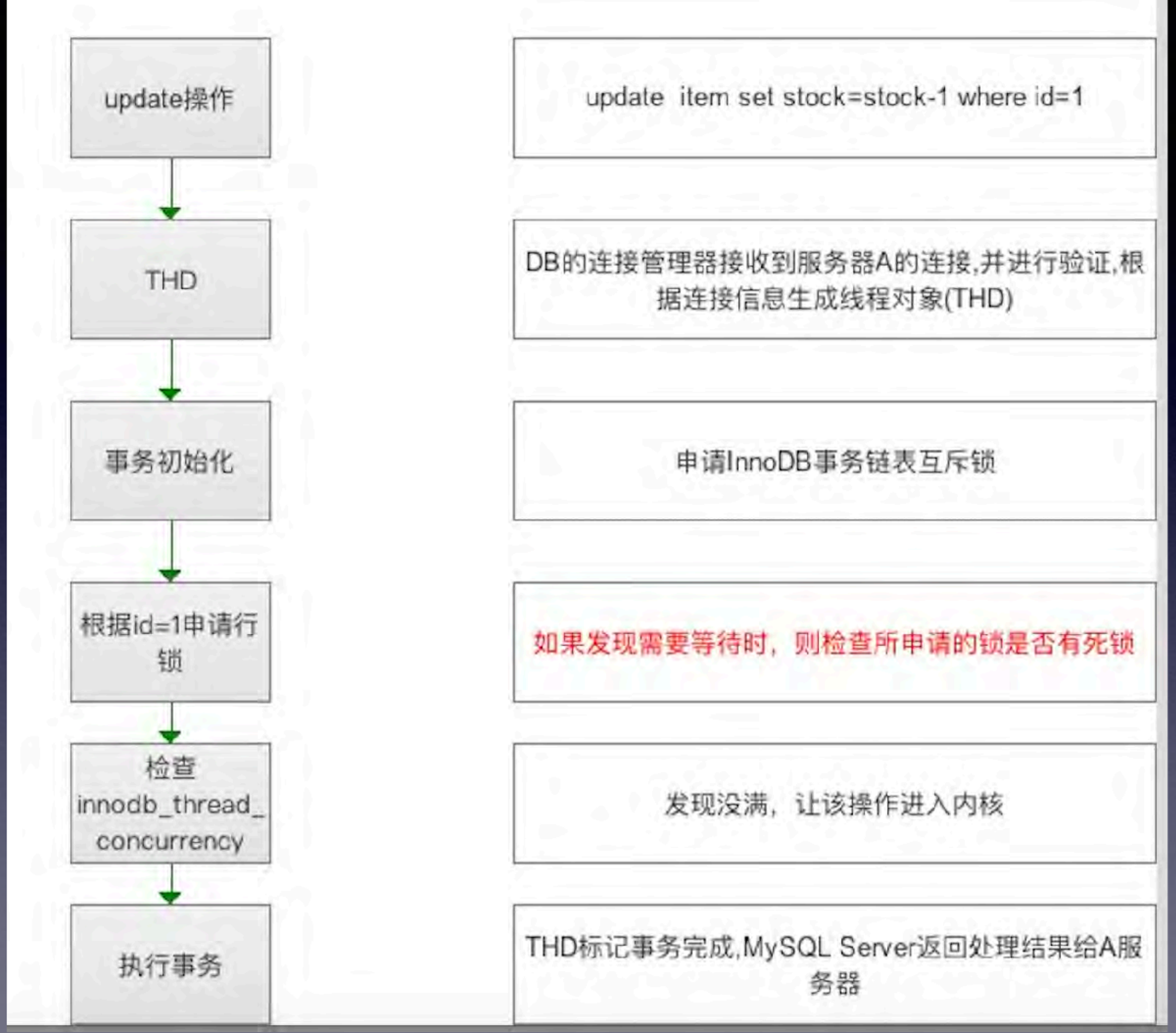
# 目录

- 一个事务之旅(简化版)
- DB端:线程池、事务、死锁检测基础概念
- DB端:性能压测数据对比、结论
- 应用端如何优化?

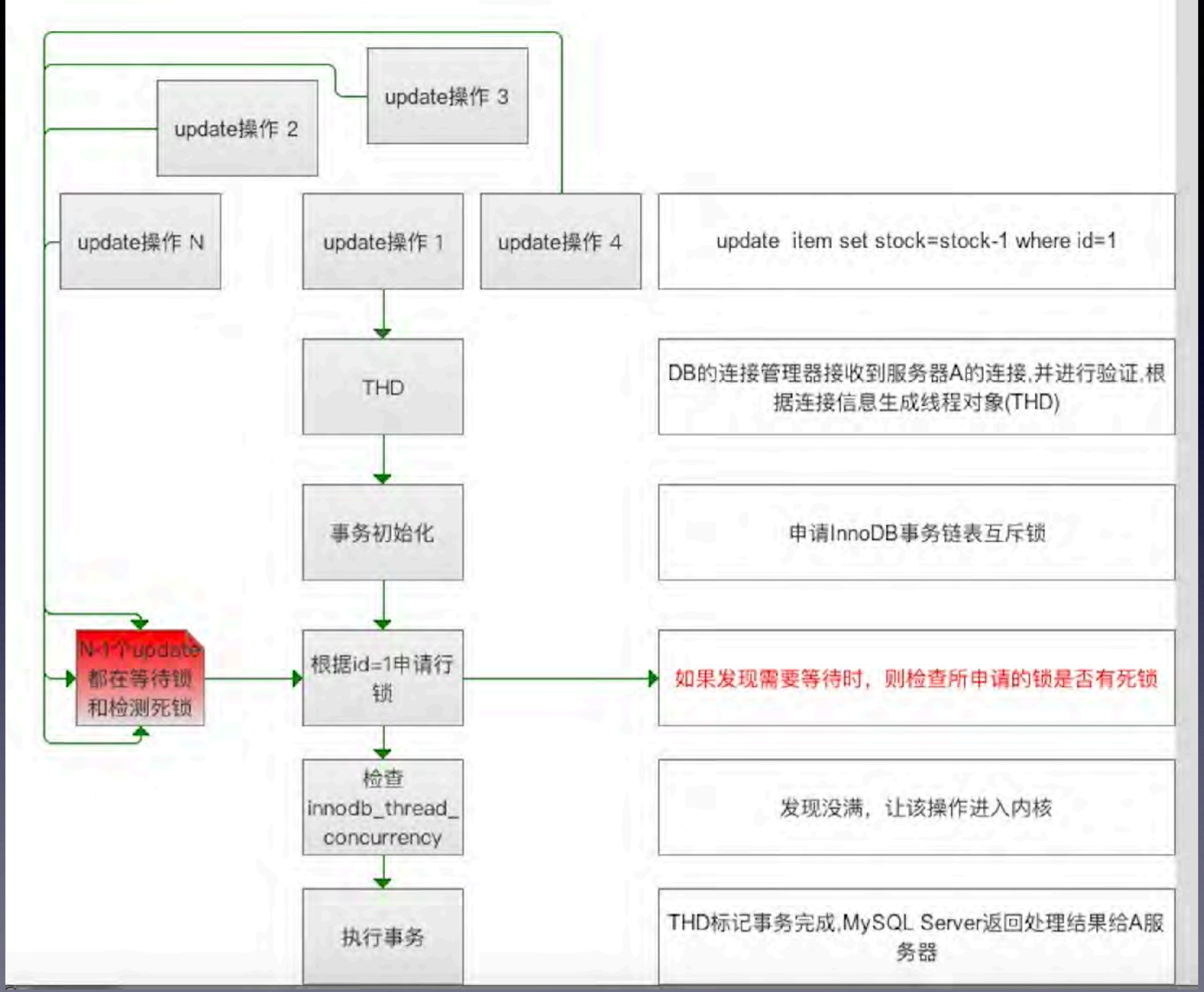
# MySQL 内部模块及InnoDB存储引擎架构



# 一个update事务之旅(简化)



# 高并发的update事务热点库存扣减情况



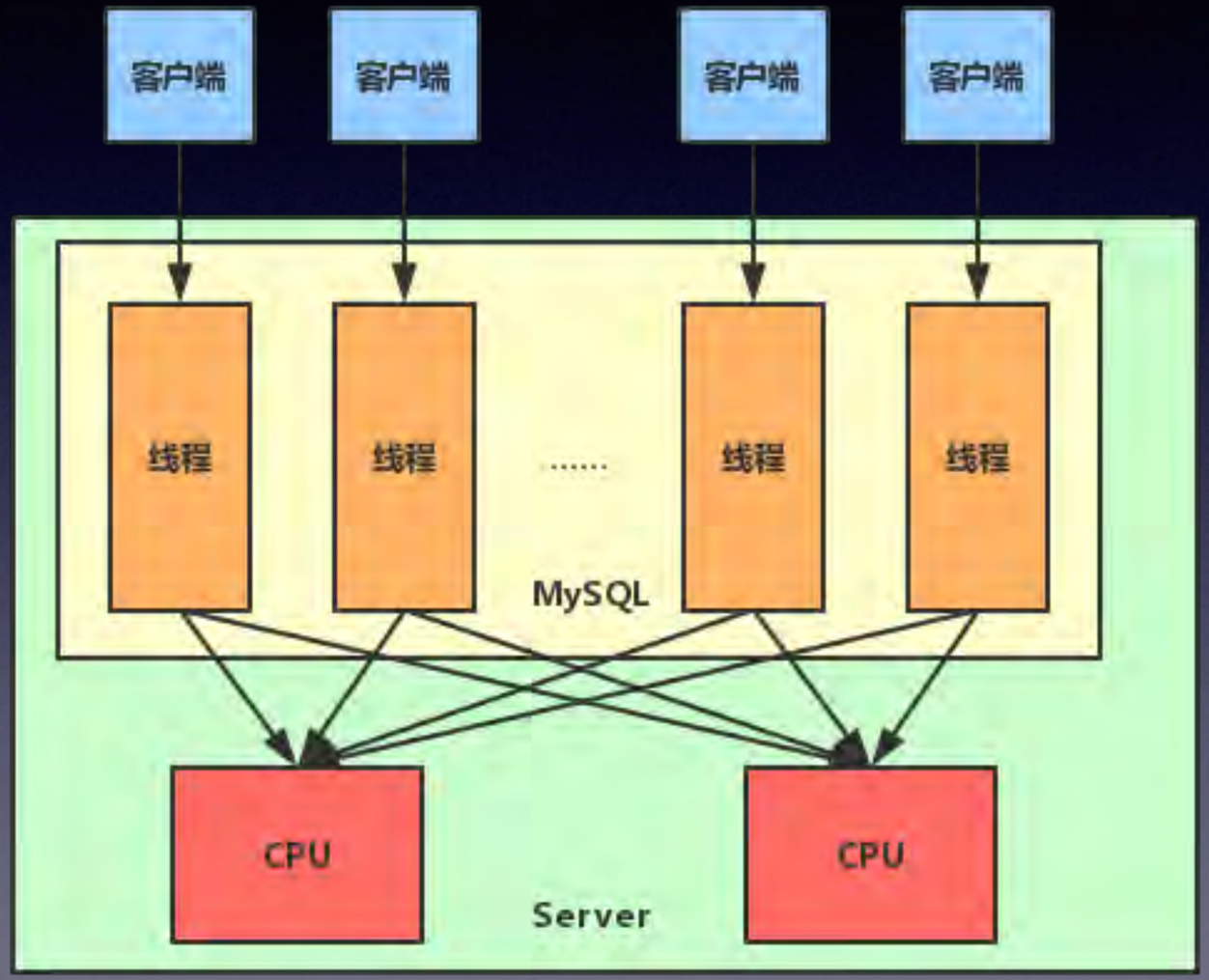
# 高并发访问秒杀场景下

- 数千个老王进行了点击购买，这时候DB会出现什么情况?
- 连接数增加，MySQL Server的线程切换成本增加
- 每个update都要等待前面的update释放锁，都会进行死锁检测，请求越多，检测等待越久

# 目录

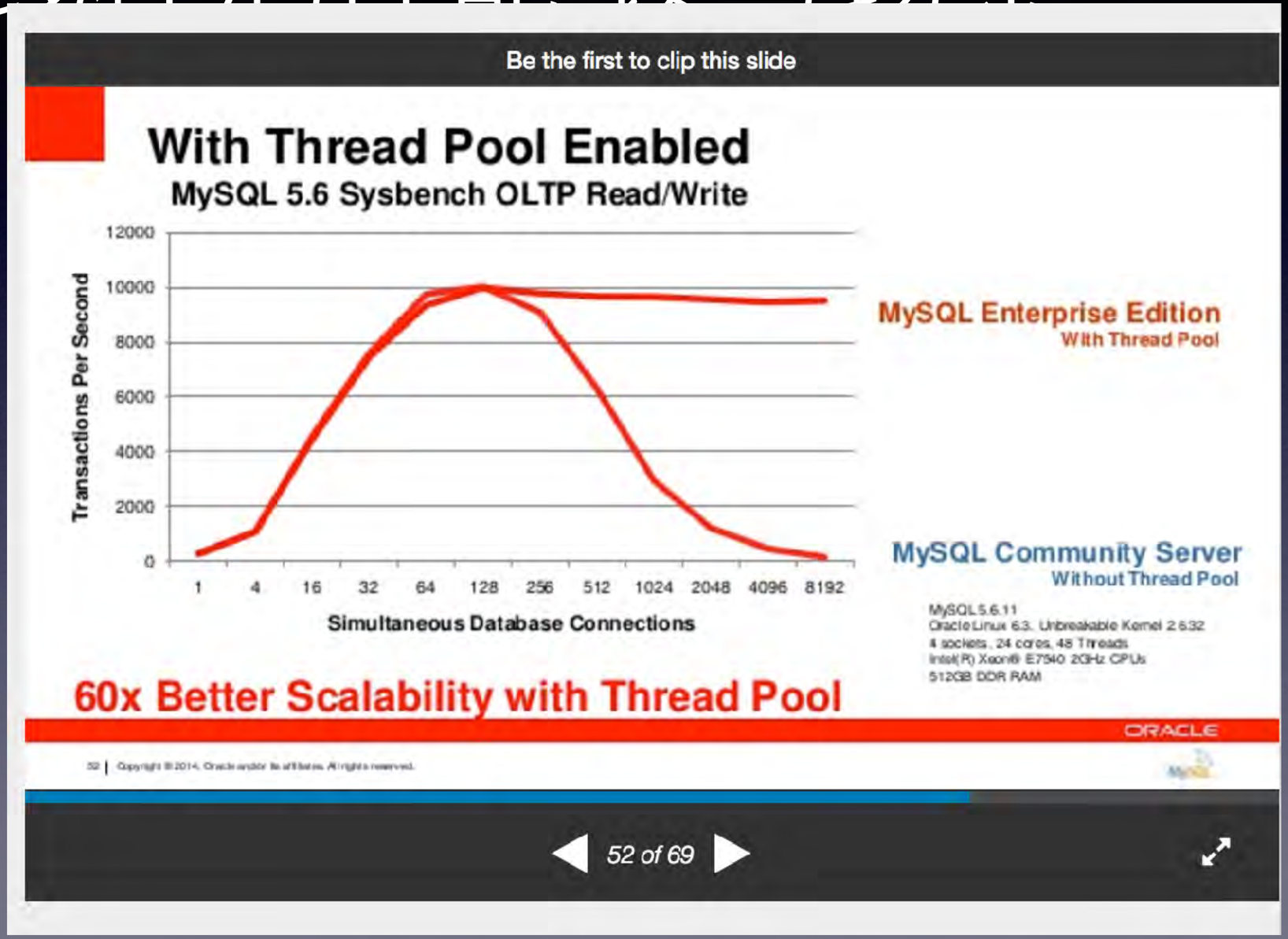
- DB端:一个InnoDB事务之旅
- DB端:线程池、事务、死锁检测基础概念
- DB端:性能压测数据对比、结论
- 应用端如何优化?

# MySQL Server默认线程模型



# 线程池性能 读写场景

128个线程并发
原生版本性能
降低

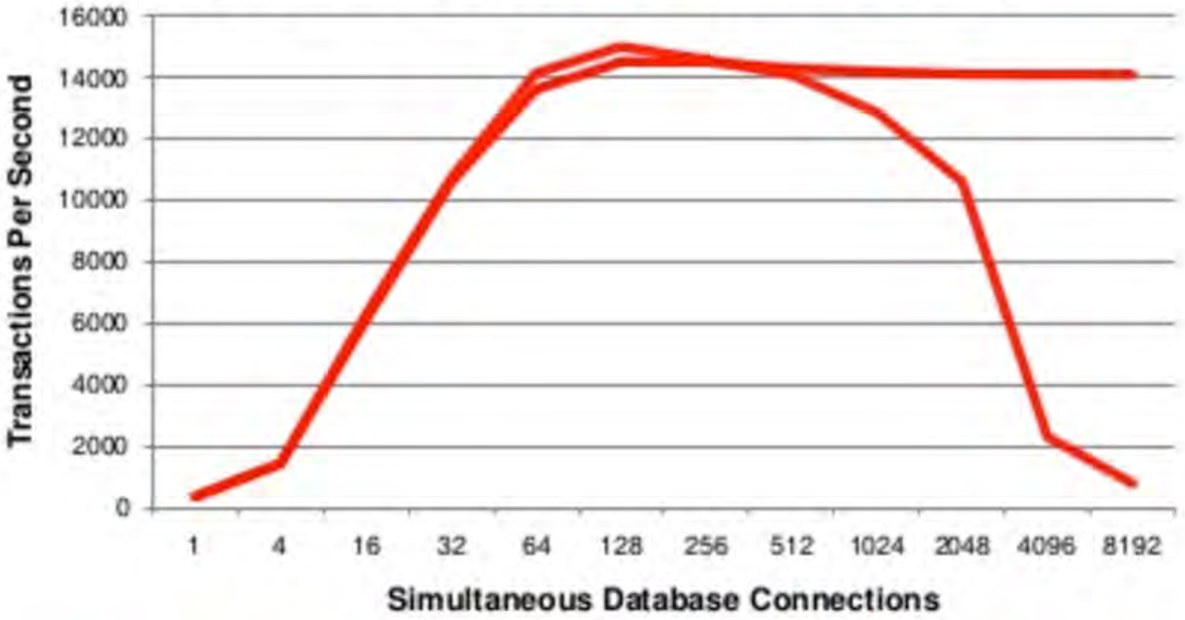


# 线程池性能 只读场景



## With Thread Pool Enabled

### MySQL 5.6 Sysbench OLTP Read Only




**MySQL Enterprise Edition**
With Thread Pool

**MySQL Community Server**
Without Thread Pool

MySQL 5.6.11
Oracle Linux 6.3, Unbreakable Kernel 2.6.32
4 sockets, 24 cores, 48 Threads
Intel(R) Xeon(R) E7540 2GHz CPUs
512GB DDR RAM

**18x Better Scalability with Thread Pool**

ORACLE

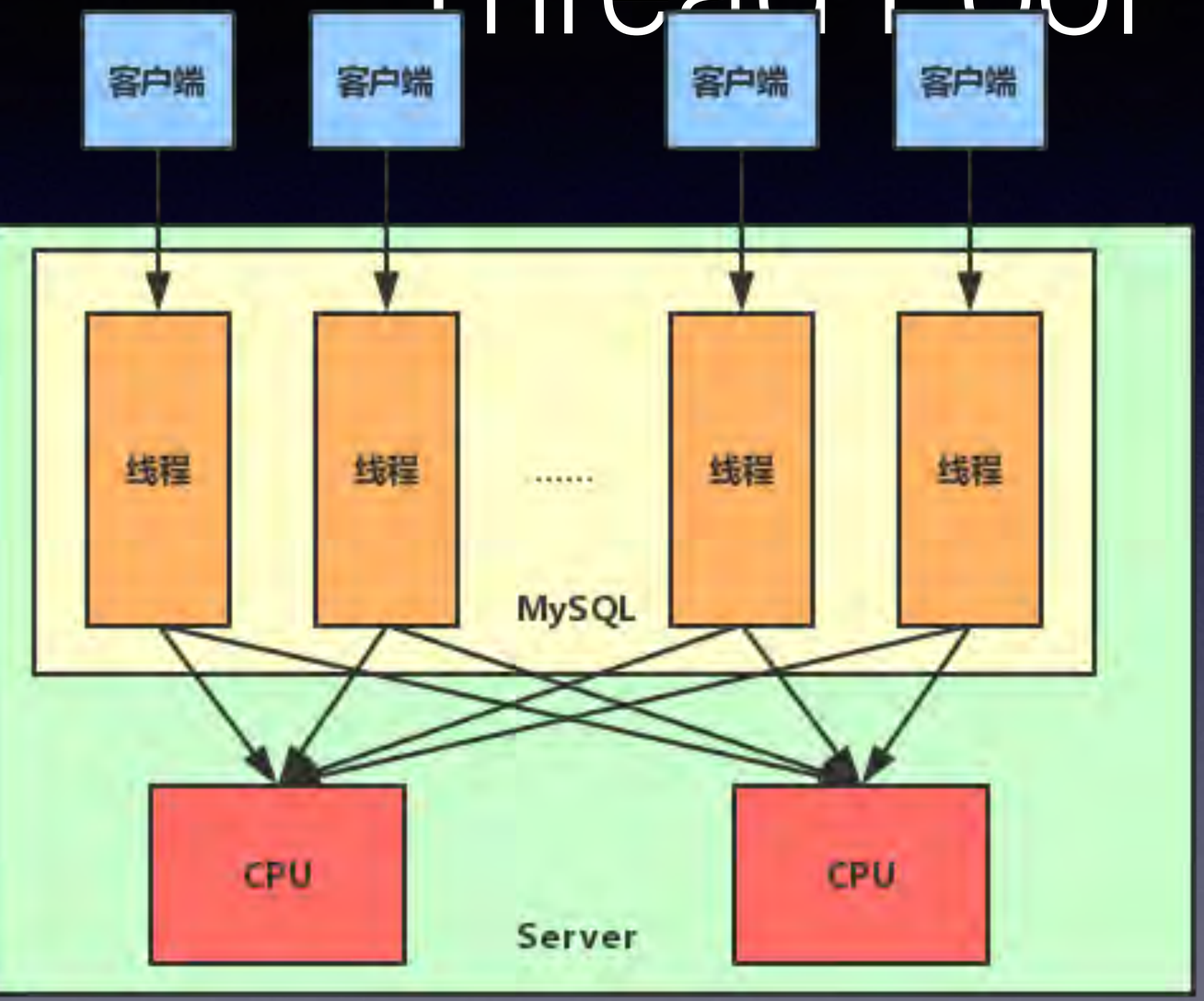
# MySQL线程池的发展路径

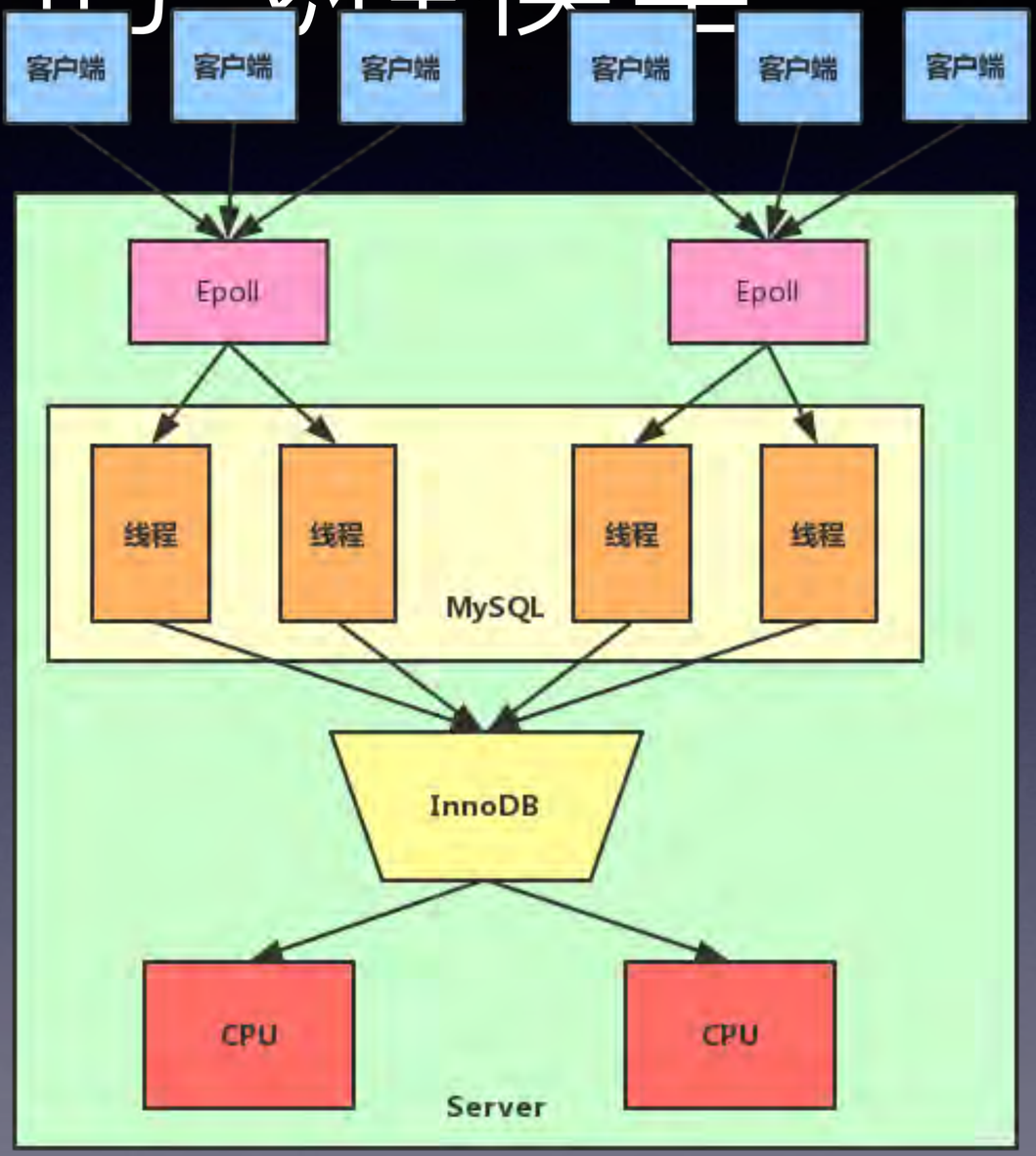
- MySQL处理客户端对应的参数是thread_handling
  - no-threads:debug模式，DB只创建一个链接和线程
  - one-thread-per-connection 每一个客户端链接，服务端都会创建对应的一个线程来服务，然后根据thread_cache来进行缓存线程信息以便重复使用
  - dynamically-loaded 线程池模式 服务端预先根据设置的线程池大小，创建好线程，等客户端创建connections到Server后，调度线程池里面的线程来进行排队服务，可以理解为线程池帮助服务端做了一下缓冲，减少对后端的影响，后端可以更专注于本身的事情。

# MySQL线程池的发展路径

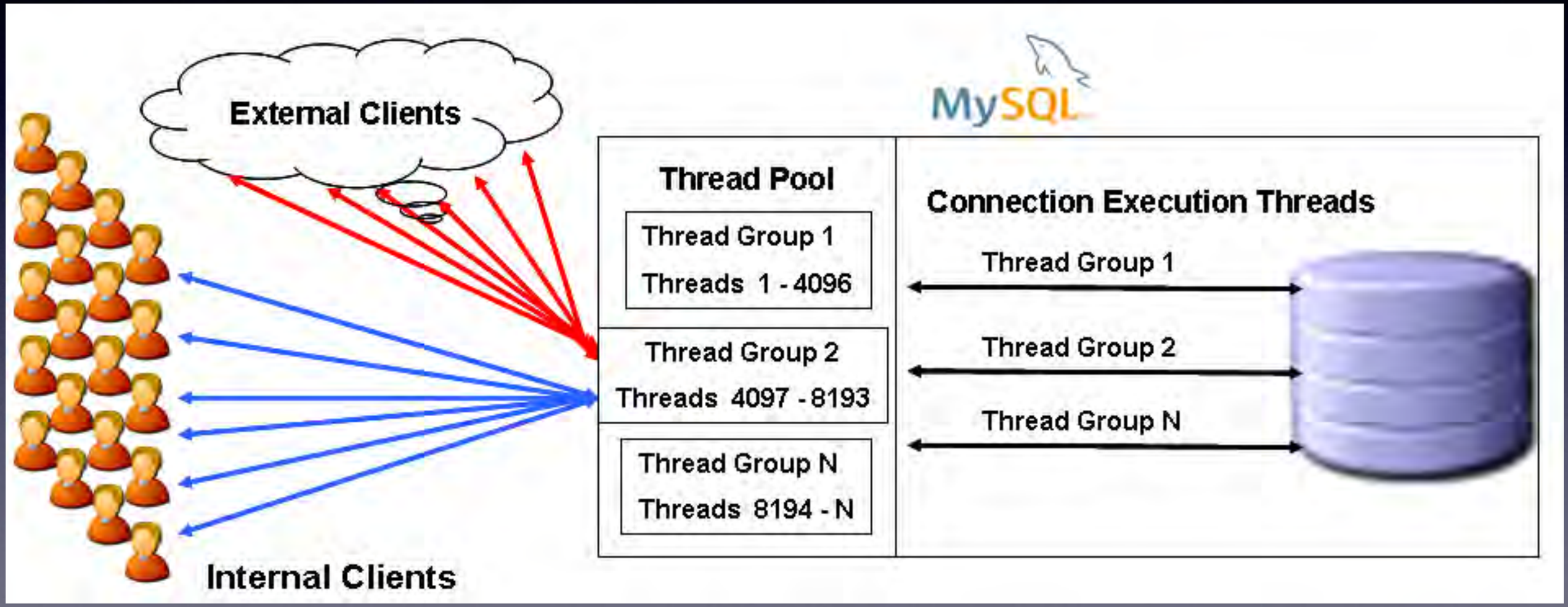
- 线程池模式在MySQL 5.5 企业版，提供了一个线程池的Plugin，专门针对MySQL的活跃线程超过一定数量性能会大幅下跌的场景。线程池通过限制了随后一些开销较小的线程，提高了MySQL性能的稳定。但没开源。
- MariaDB从MariaDB 5.1开始支持线程池功能
- 2013年MariaDB对线程池进行了开源
- Percona基于MariaDB的线程池，也增加了线程池功能，我们压测的版本Percona-5.6.25默认带有线程池功能

# Thread Pool下的线程模型



# MySQL线程池的实现

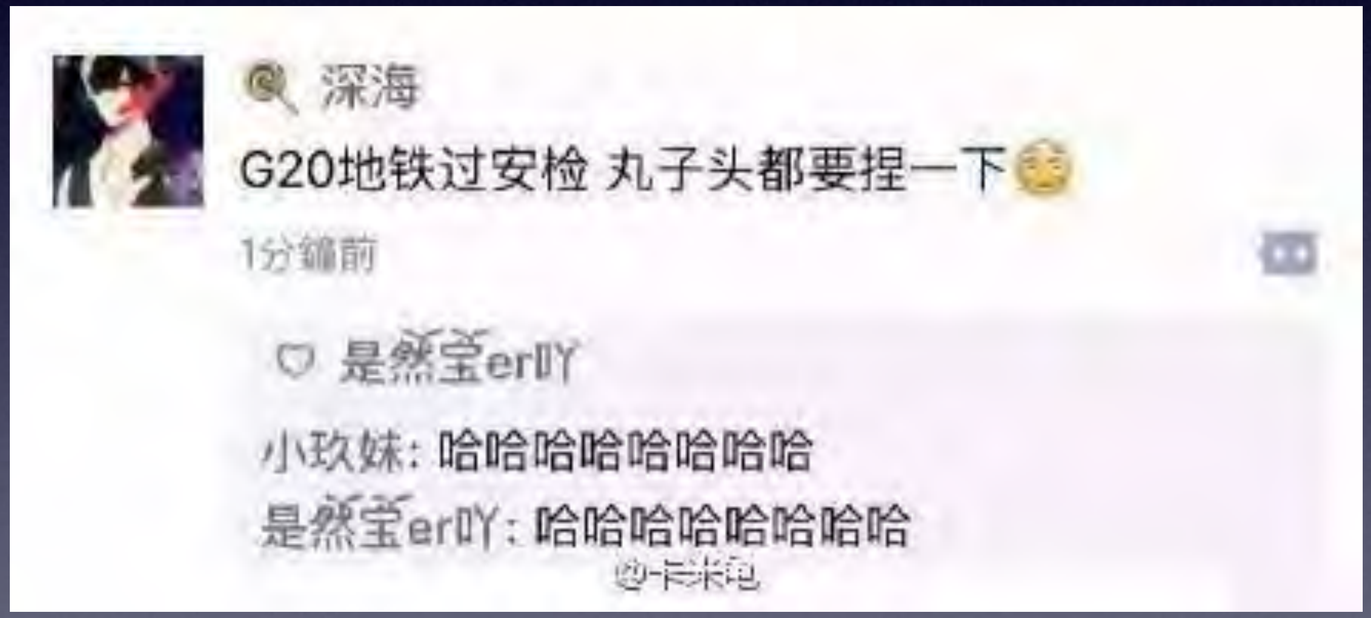


# 线程池总结

- 每个线程都消耗额外的内存，而且每次线程间的切换都会消耗CPU周期并丢弃CPU高速缓存中的数据。
- 线程池减少了连接和线程切换和维护的成本
- Percona和MariaDB比官方社区版增加了线程池支持

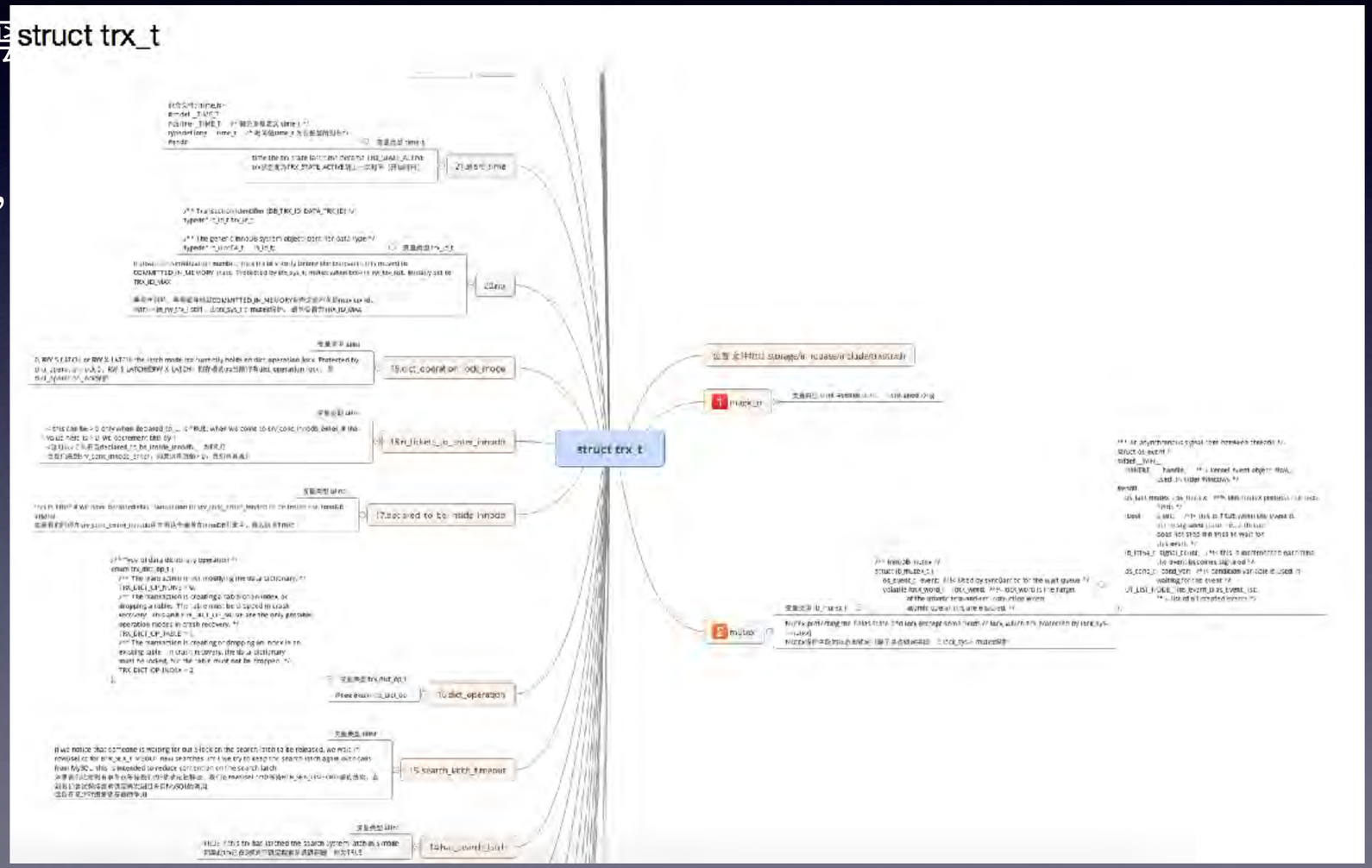
# InnoDB死锁检测

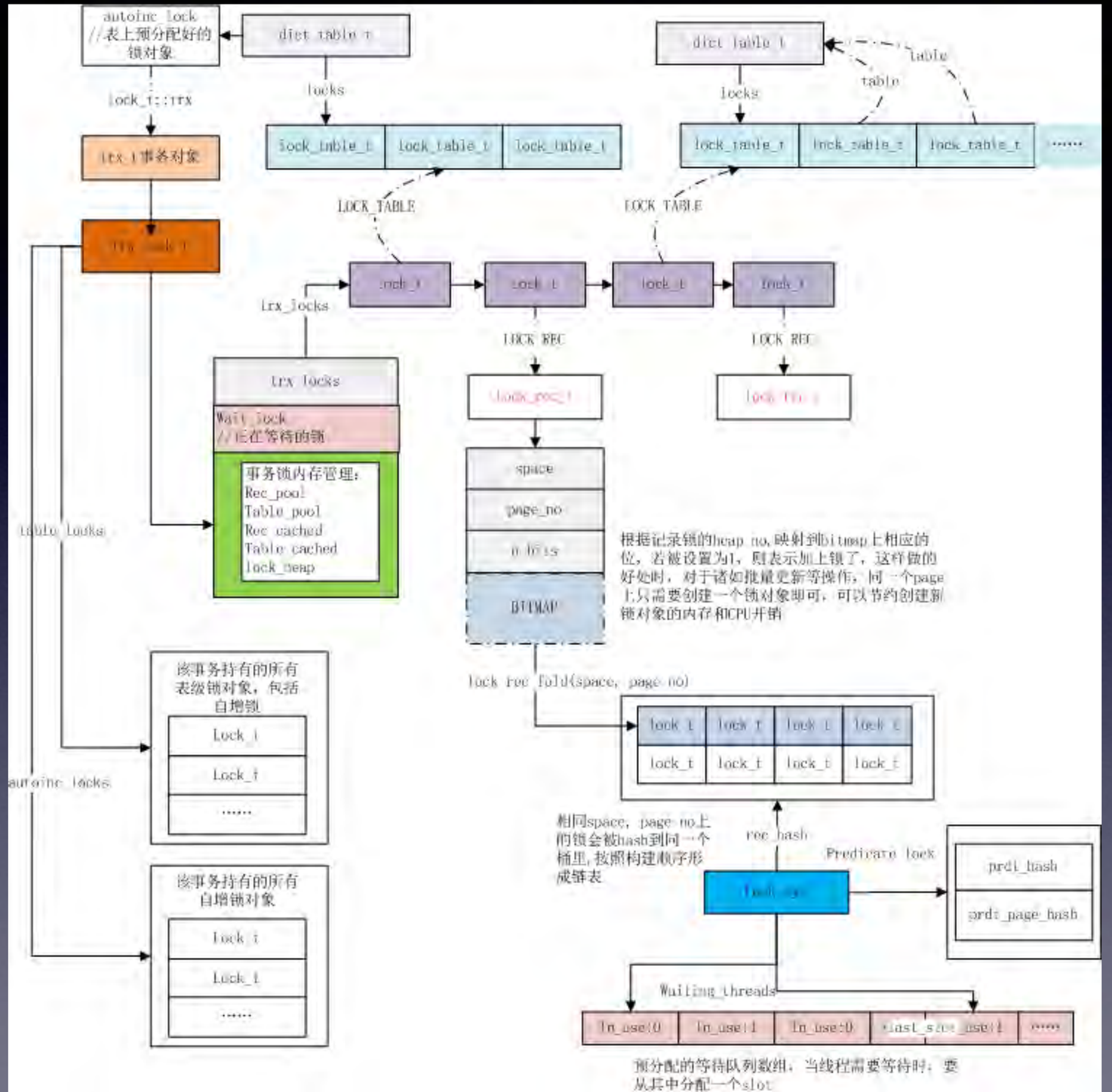

杭州戒備森嚴
中視
杭州
台南
27-32
西湖景區封閉 車證+身分證才能通關
中視新聞
愛心公益
<樂公益平台>幫助單親媽媽 自力更生 搜尋:lecoin.cc


深海
G20地铁过安检 丸子头都要捏一下
1分鐘前
是然宝er吖
小玖妹: 哈哈哈哈哈哈哈哈哈
是然宝er吖: 哈哈哈哈哈哈哈哈哈

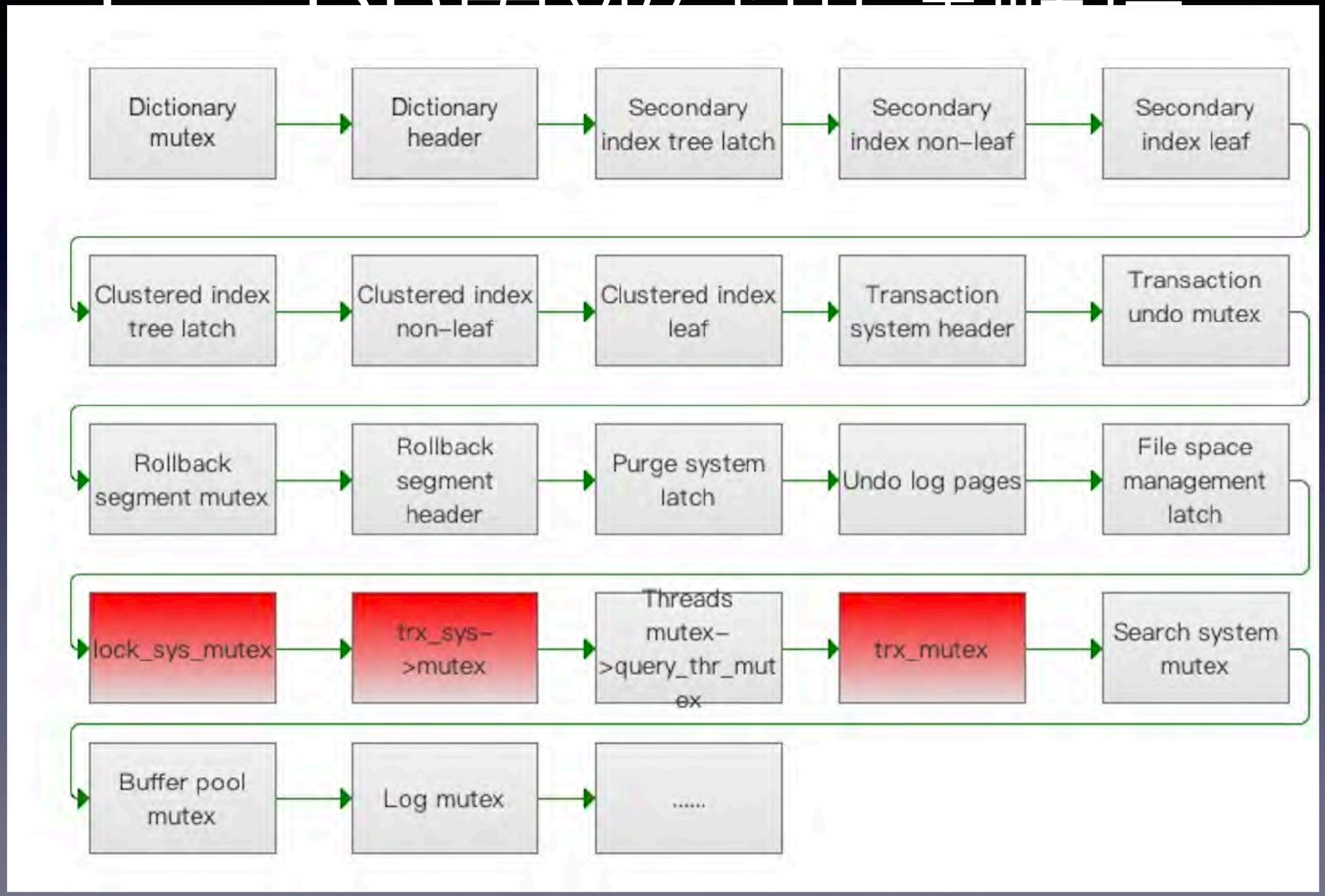
# InnoDB事务

- ACID(原子性,一致性,隔离
- InnoDB事务源码结构体,
- 需要关注的几个锁结构
  - 事务锁结构 trx_locks
  - 锁结构 lock_t
  - 事务结构 trx_t




autoinc_lock
//表上预分配好的锁对象
dict_table_t
lock_t::trx
trx_t事务对象
trx_lock_t
locks
lock_table_t
lock_table_t
lock_table_t
dict_table_t
table
table
locks
lock_table_t
lock_table_t
lock_table_t
.....
LOCK_TABLE
LOCK_TABLE
trx_locks
lock_t
lock_t
lock_t
lock_t
LOCK_REC
LOCK_REC
trx_locks
Wait_lock
//正在等待的锁
事务锁内存管理：
Rec_pool
Table_pool
Rec_cached
Table_cached
lock_heap
lock_rec_t
lock_rec_t
table_locks
space
page_no
n_bits
BITMAP
根据记录锁的heap_no,映射到bitmap上相应的位，若被设置为1，则表示加上锁了，这样做的好处时，对于诸如批量更新等操作，同一个page上只需要创建一个锁对象即可，可以节约创建新锁对象的内存和CPU开销
lock_rec_fold(space, page_no)
该事务持有的所有表级锁对象，包括自增锁
Lock_t
Lock_t
.....
autoinc_locks
该事务持有的所有自增锁对象
Lock_t
Lock_t
.....
lock_t
lock_t
lock_t
lock_t
lock_t
lock_t
lock_t
lock_t
相同space，page_no上的锁会被hash到同一个桶里，按照构建顺序形成链表
rec_hash
Predicate lock
lock_sys
prdt_hash
prdt_page_hash
Waiting_threads
In_use:0
In_use:1
In_use:0
last_slot_use:1
.....
预分配的等待队列数组，当线程需要等待时，要从其中分配一个slot

# InnoDB内部锁争用顺序



# InnoDB死锁检测

- 1.什么是死锁?
- 2.InnoDB什么时候会检测死锁?
- 3.数据库系统如何处理死锁?
- 4.InnoDB的死锁检测过程?

# 什么是死锁

- 《数据库系统实现》第八章第二节这样定义死锁

  - 并发执行的事务由于竞争资源而到达一个存在死锁的状态:
  - 若干事务的每一个事务都在等待被其他事务占用的资源，因而每个事务都不能取得进展。

# 死锁的形象认识

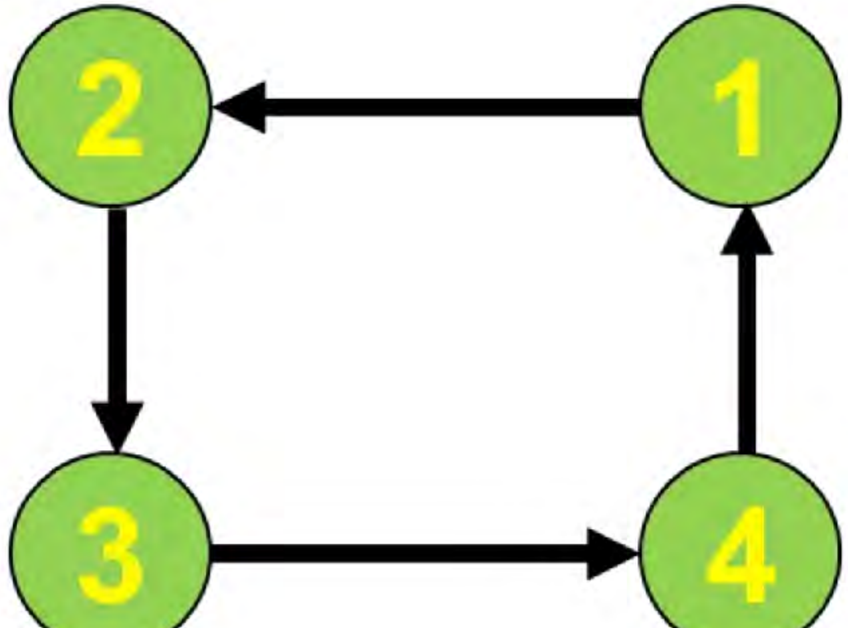
- 堵车现象
- 两位木匠钉地板

子

# 数据库系统如何处理死锁

死锁处理机制

- 超时死锁检测
  - 当存在死锁时，想所有事务都能同时继续执行通常是不可能的，因此，至少一个事务必须中止并重新开始。超时是最直接的办法，对超出活跃时间的事务进行限制和回滚
- 等待图
  - 等待图的实现，是可以表明哪些事务在等待其他事务持有的锁，可以在数据库的死锁检测里面加上这个机制来进行检测是否有环的形成
- 通过元素排序预防死锁
  - 这个想法很美好，但
  办法解决死锁的原因
- 通过时间戳检测死锁
  - 对每个事务都分配一



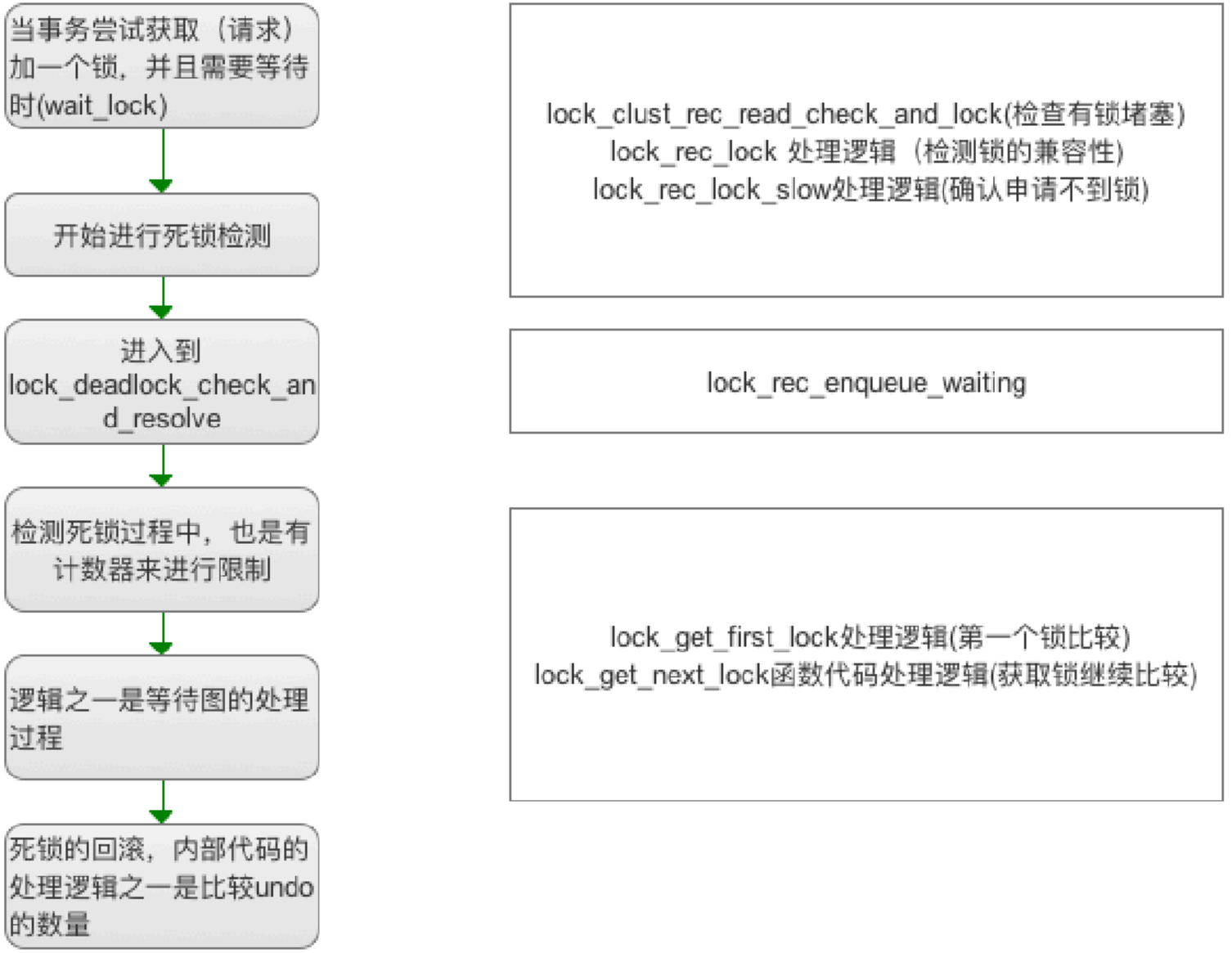
# 涉及死锁检测的例子

- create table
  engine=inn
- insert into t
- innodb_loc

场景

| 时间 | T1 | GDB | T2 |
| --- | --- | --- | --- |
| 先开始T1 | begin;<br>update<br>test_lock set<br>stock=stock-1<br>where id=1; | | |
| 接着开始T2 | | b<br>lock_get_first_lock<br>c | |
| | | | begin;<br>update test_lock set stock=stock-1 where id=1; |
| | | bt(输出了25行堆栈)<br>c | |
| 50秒后 | | | ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction |

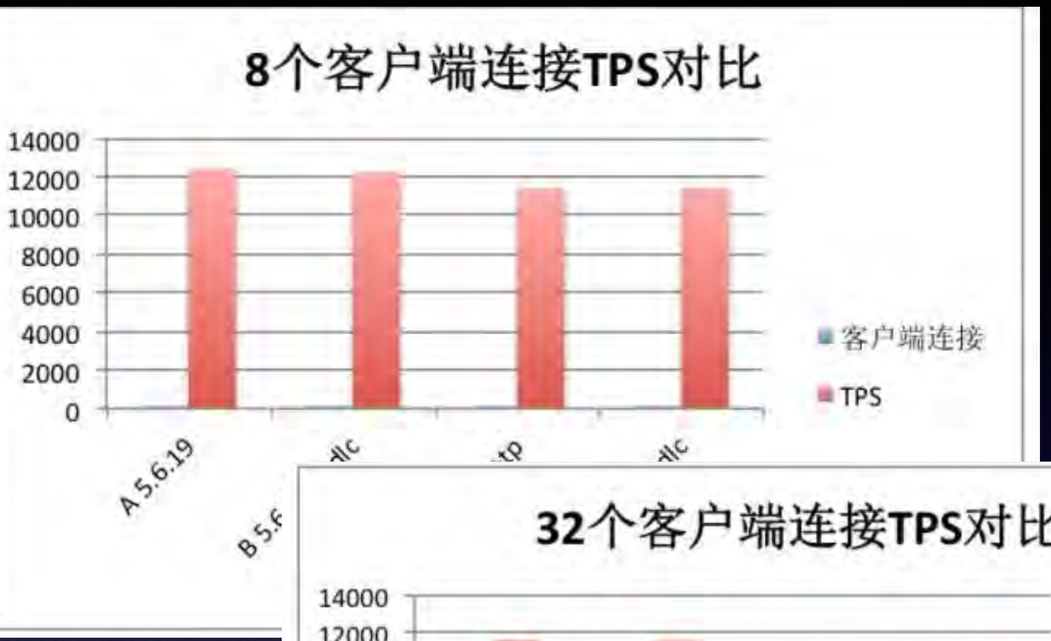
# MySQL 死锁检测和回滚

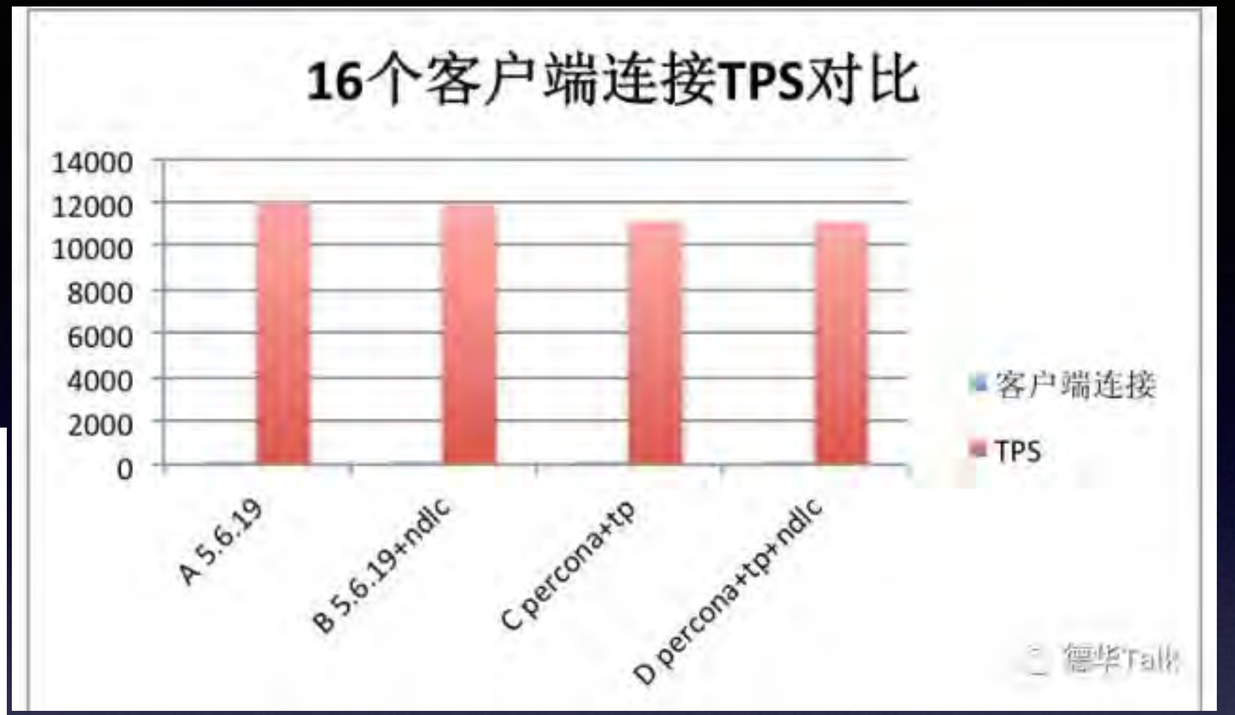


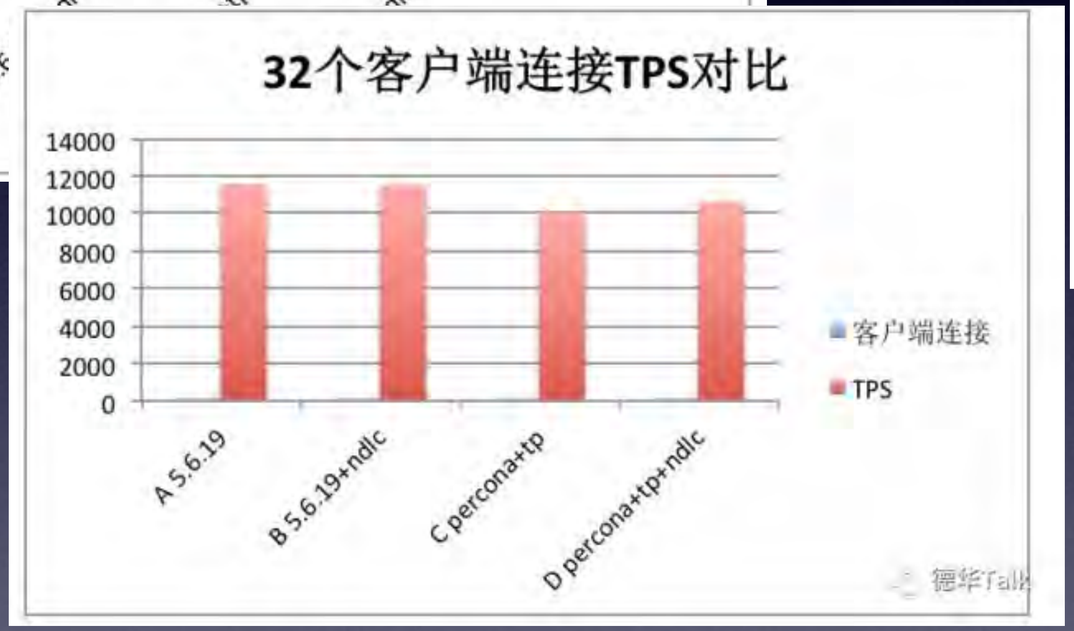
# 关闭死锁检测的一些故事..

# 性能压测对比

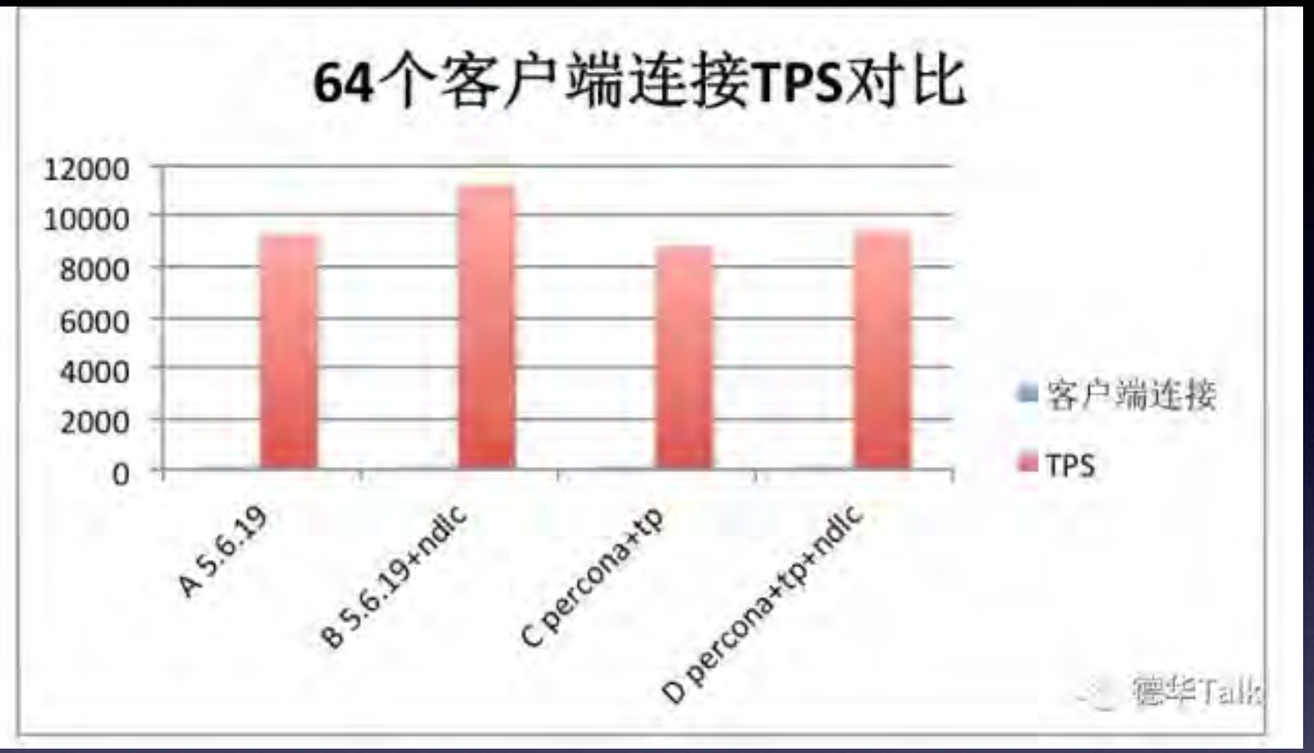
- 对着四个版本进行了8，16，32，64，128，512，1024个并发线程的压测
- A.MySQL5.6.19 原生版本
- B.MySQL5.6.19 增加死锁检测关闭 （5.6.1.9+no deadlock check ndlc)
- C.Percona5.6.25 原生版本 (percona+tp)
- D.Percona5.6.25增加死锁检测关闭 (percona+tp+nldc)

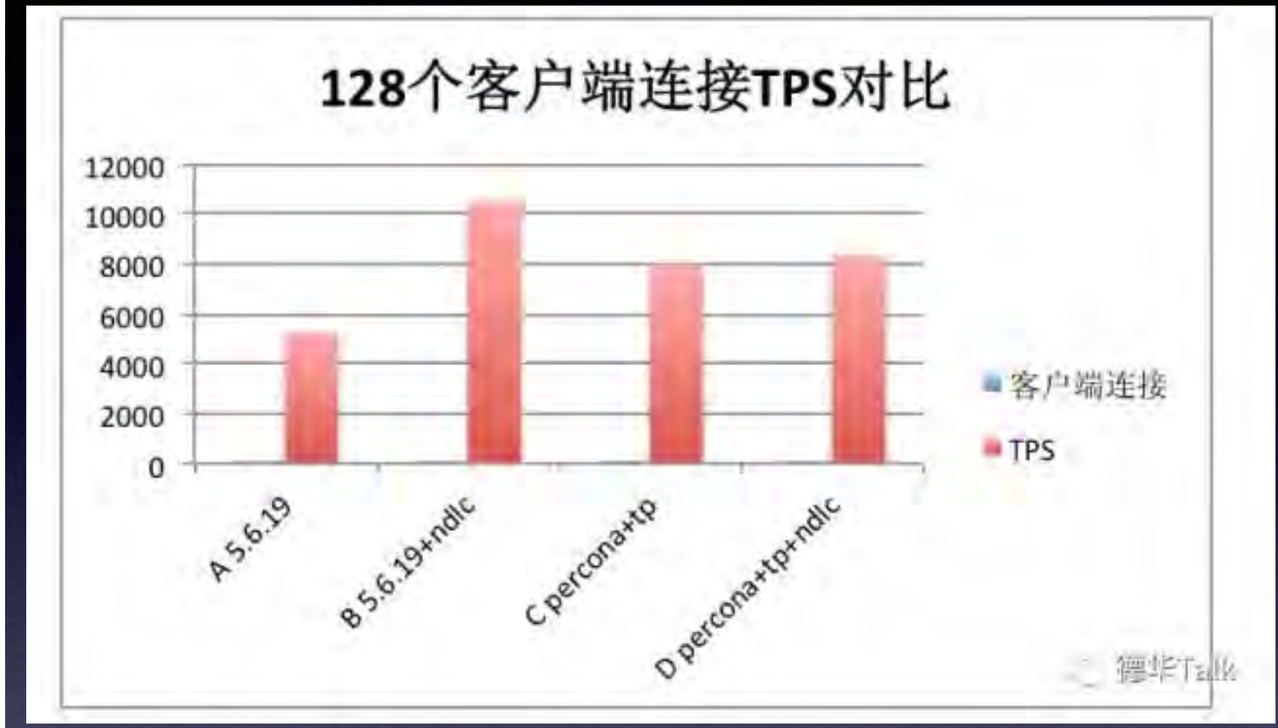


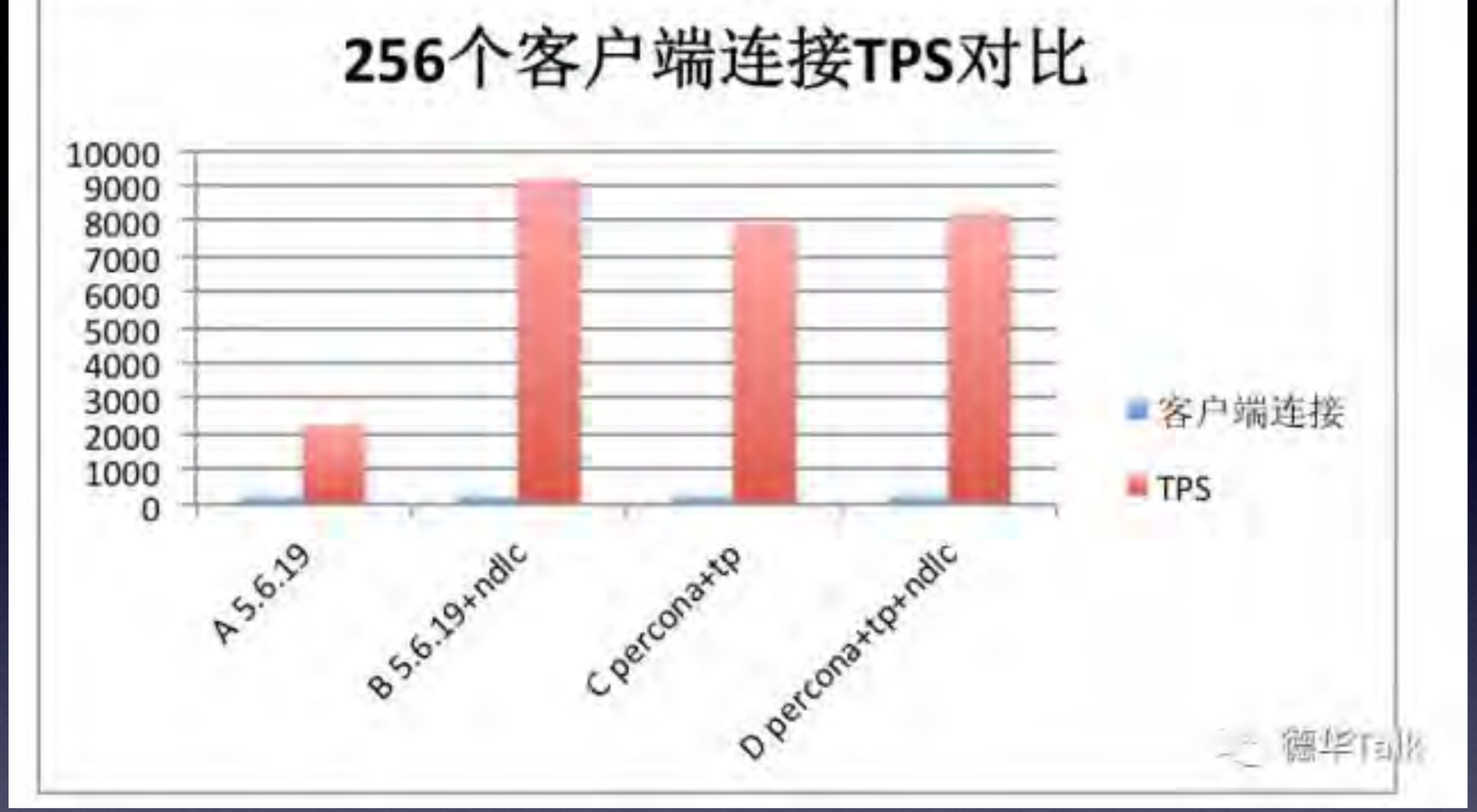




- 8，16，32个客户端下，原生版本的TPS比后面三个场景的TPS要高.但这么少的连接，业务很难会只给这么少连接

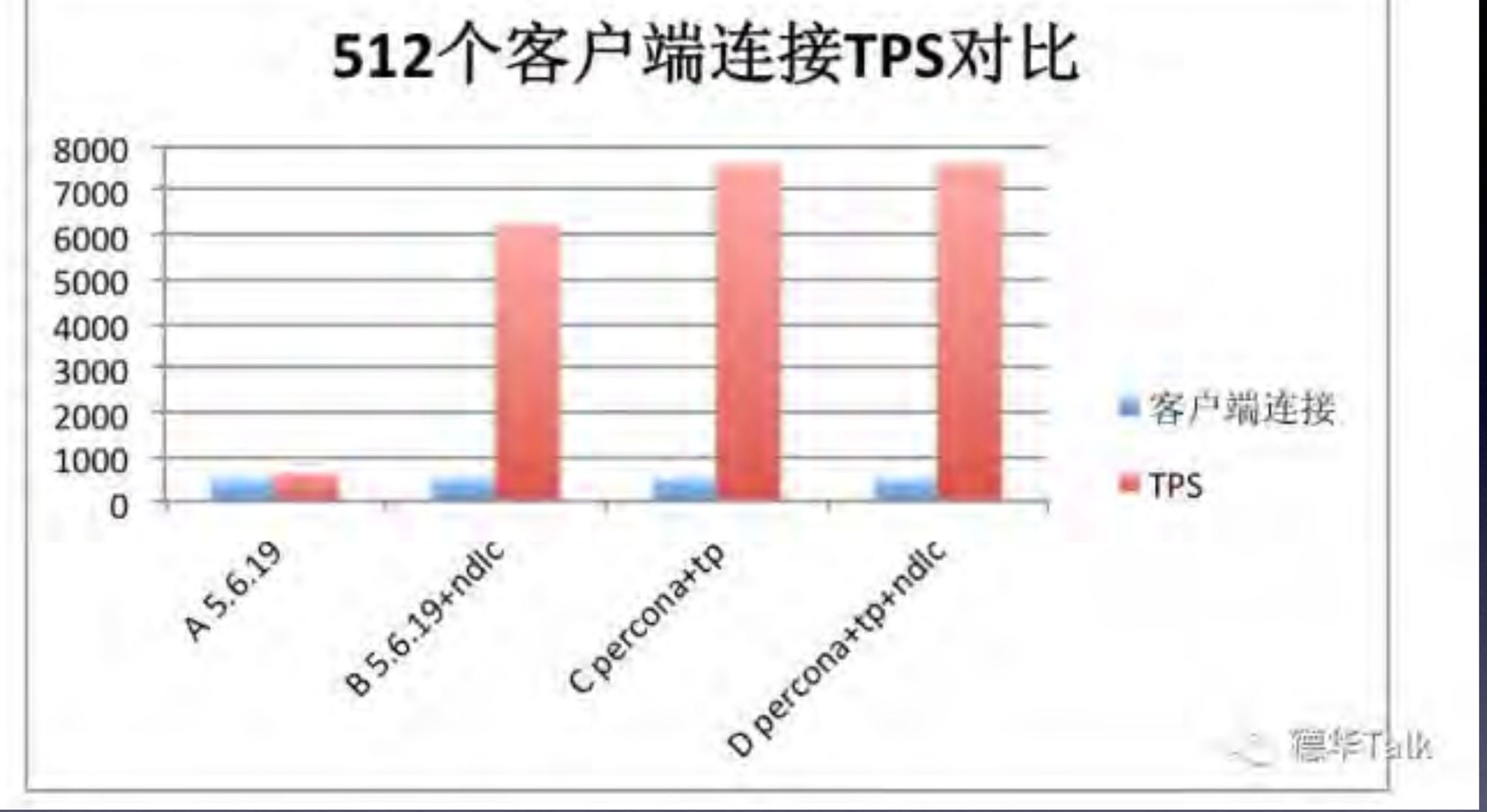


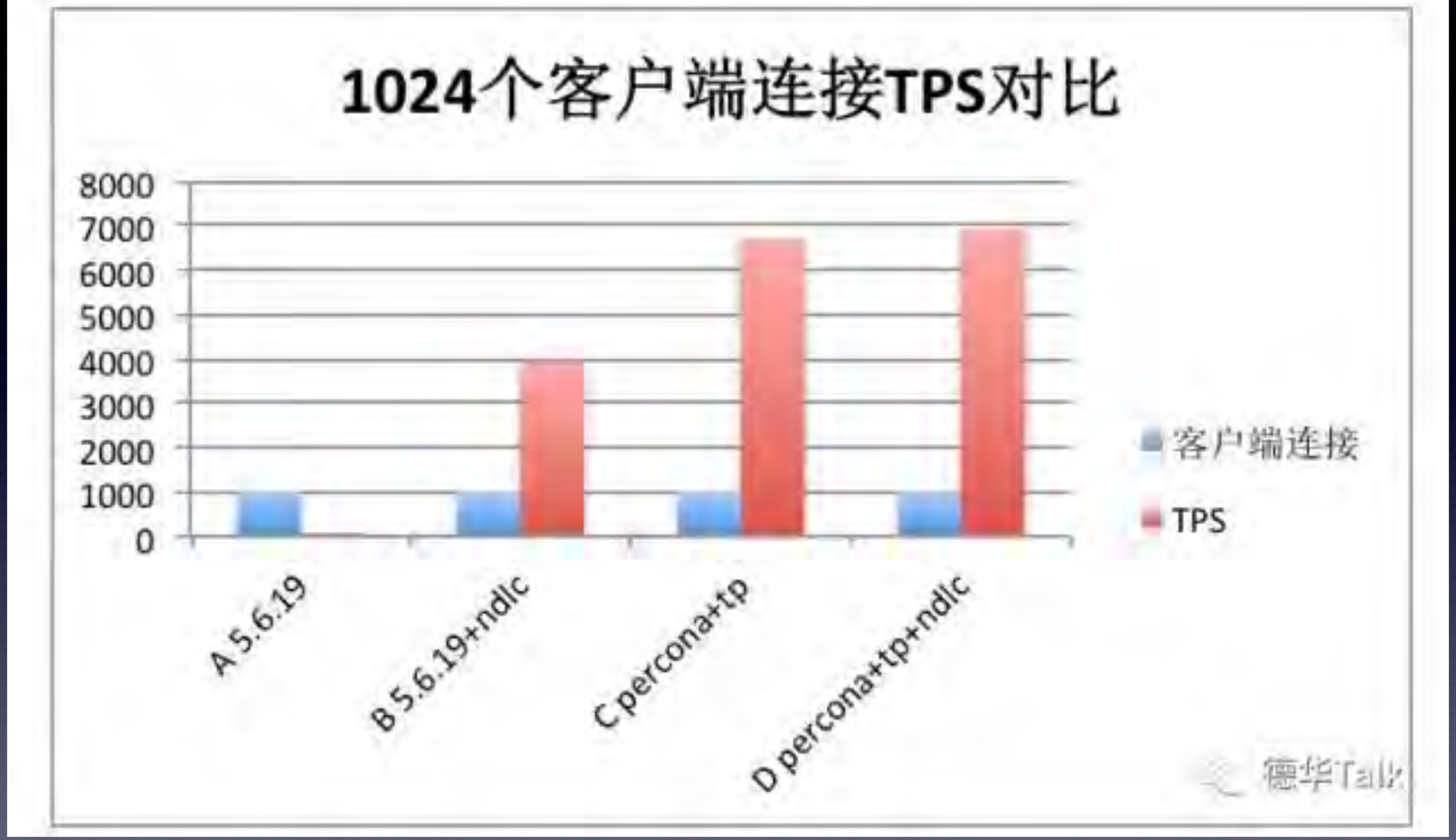

- 64个和128个客户端连接
- 原生版本+关闭死锁检测 TPS最高




- 256个客户端连接下，原生版本+关闭死锁检测 TPS最高，原生版本的TPS已经降低到2000+




- 512个客户端连接下，5.6.19TPS降低到600+




- 1024个客户端连接下，5.6.19TPS降低到30

# 5.6版本的压测数据

| 并发连接数 | 版本5.6.19 | 版本5.6.19+关闭死锁 | Percona5.6+线程池 | Percona5.6+线程池+关闭死锁 |
|---|---|---|---|---|
| 256 | 2271.91 | 9198.25 | 7925.89 | 8229.47 |
| 512 | 646.53 | 6225.06 | 7587.56 | 7608.73 |
| 1024 | 30 | 3924.82 | 6717.44 | 6925.37 |
| | | | | |

# 5.7版本压测数据

| 并发 | MySQL5.7.18死锁检测打开 | MySQL5.7.18死锁检测关闭 | Percona5.1.17-15死锁检测打开 | Percona5.1.17-15死锁检测关闭 |
|---|---|---|---|---|
| **256** | 2000 | 6000 | 4000 | 4200 |
| **512** | 500 | 2500 | 4300 | 4350 |
| **1024** | 30 | 1500 | 5000 | 5200 |

# 关闭死锁检测-压测结论

- 关闭死锁检测后，5.6.19/5.7.19版本在1024个客户端并发连接情况下可以从30TPS提升到3924
- 应用程序无须更改代码，减库存性能提升120倍；

# 线程池—压测结论

- Percona版本由于有线程池做了一层连接的并发控制，间接减少了数据库内部的线程上下文切换和锁竞争
- 在1024个客户端并发连接的情况下，TPS是5.6.19+关闭死锁检测的接近1倍；

# 总体策略—压测结论

- Percona版本+线程池调优+关闭死锁检测的情况下，在1024个客户端并发连接下，性能有小幅度提升

# 落地措施-压测结论

- 5.6.19版本，谨慎考虑可以只增加关闭死锁检测的补丁代码，实现最小风险升级,关闭死锁检测后，依赖DBA审核SQL和锁等待超时回滚事务.
- 5.7.19有官方支持的参数关闭死锁检测
- 但在微服务架构下，客户端对DB的链接越来越多，建议最终升级成Percona版本，利用线程池的优势来提升DB的稳定性

# 问题回顾

- 1.MySQL线程池机制减少了内部线程竞争、CPU上下文切换

- 2.简化业务逻辑，key-value式的SQL场景可以关闭死锁检测来提升DB性能，如果真有死锁,通过锁等待超时来回滚事务

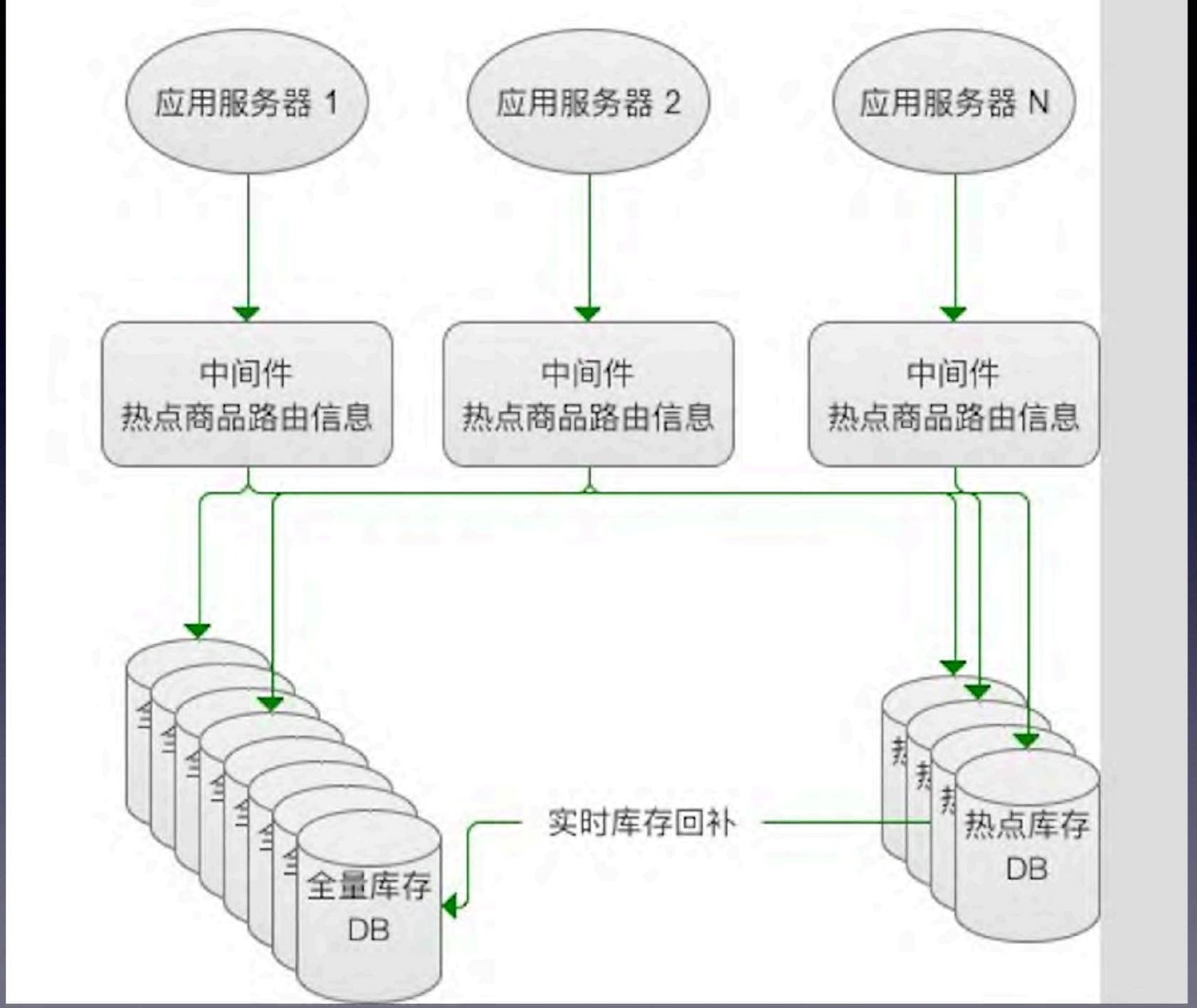
# 目录

- DB端:一个InnoDB事务之旅
- DB端:线程池、事务、死锁检测基础概念
- DB端:性能压测数据对比、结论
- 应用端如何优化?

# 其他讨论

- DB端还可以做的优化
  - 根据事务信息做预排队
- 应用端可以做的优化
  - 根据连接数据信息预排队，和上一条类似

# 热点库存架构


应用服务器 1
应用服务器 2
应用服务器 N
中间件
热点商品路由信息
中间件
热点商品路由信息
中间件
热点商品路由信息
实时库存回补
全量库存
DB
热点库存
DB

# 场景抽象

- 多个应用节点向一个DB节点的一行发送数据处理请求
- 应用处理请求是并行，DB节点最后变成了串行
- DB要保证数据的ACID以及高并发连接下的高性能

# 结束

- 谢谢大家