MySQL主从同步

——原理、问题、解决方案和应用

@淘宝丁奇

一．MySQL主从同步基本流程

二．延迟的原因

三．解决方案一

四．解决方案二 ——Transfer

五．应用场景和业务限制

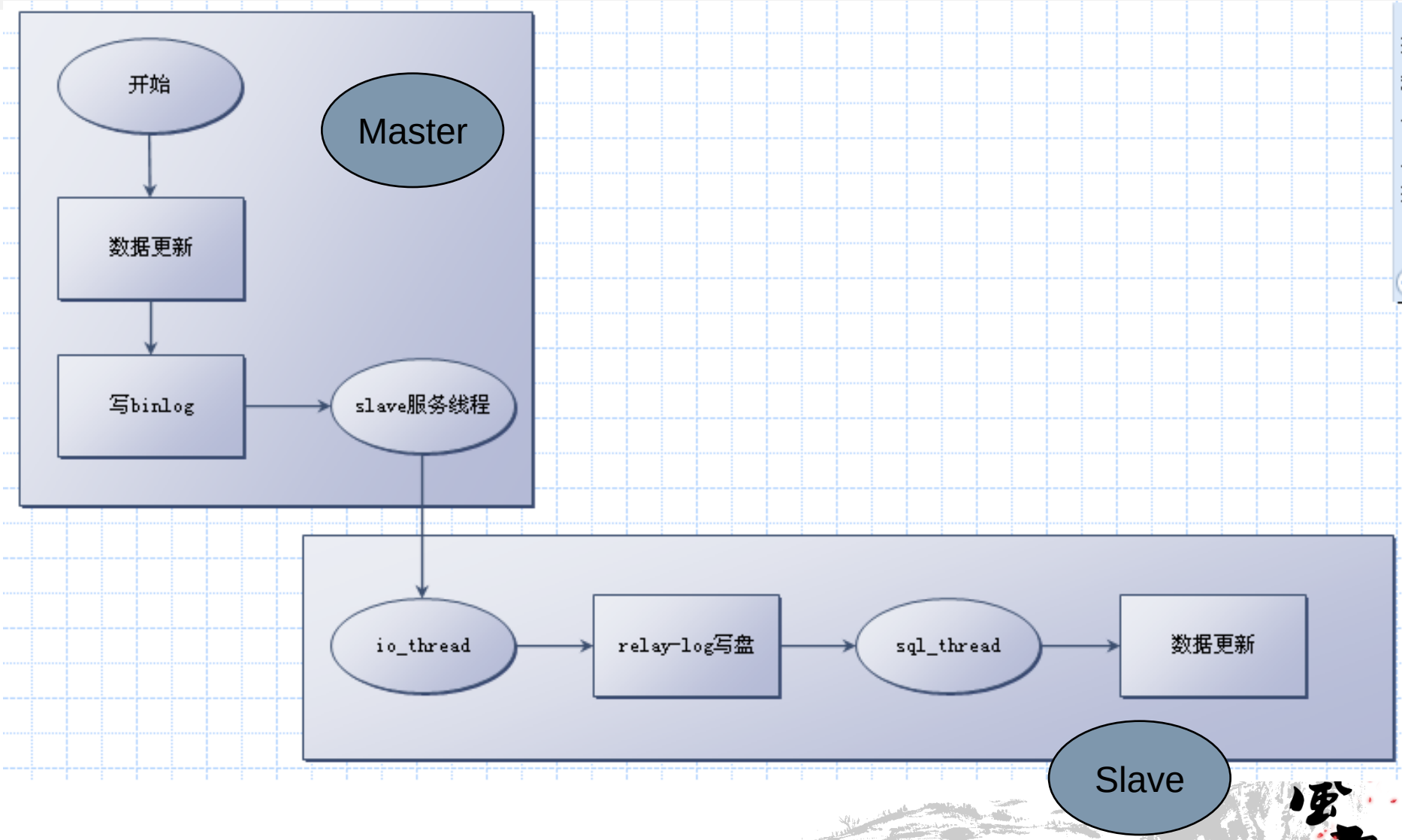
六．保障和退化

七．在多主同步的应用

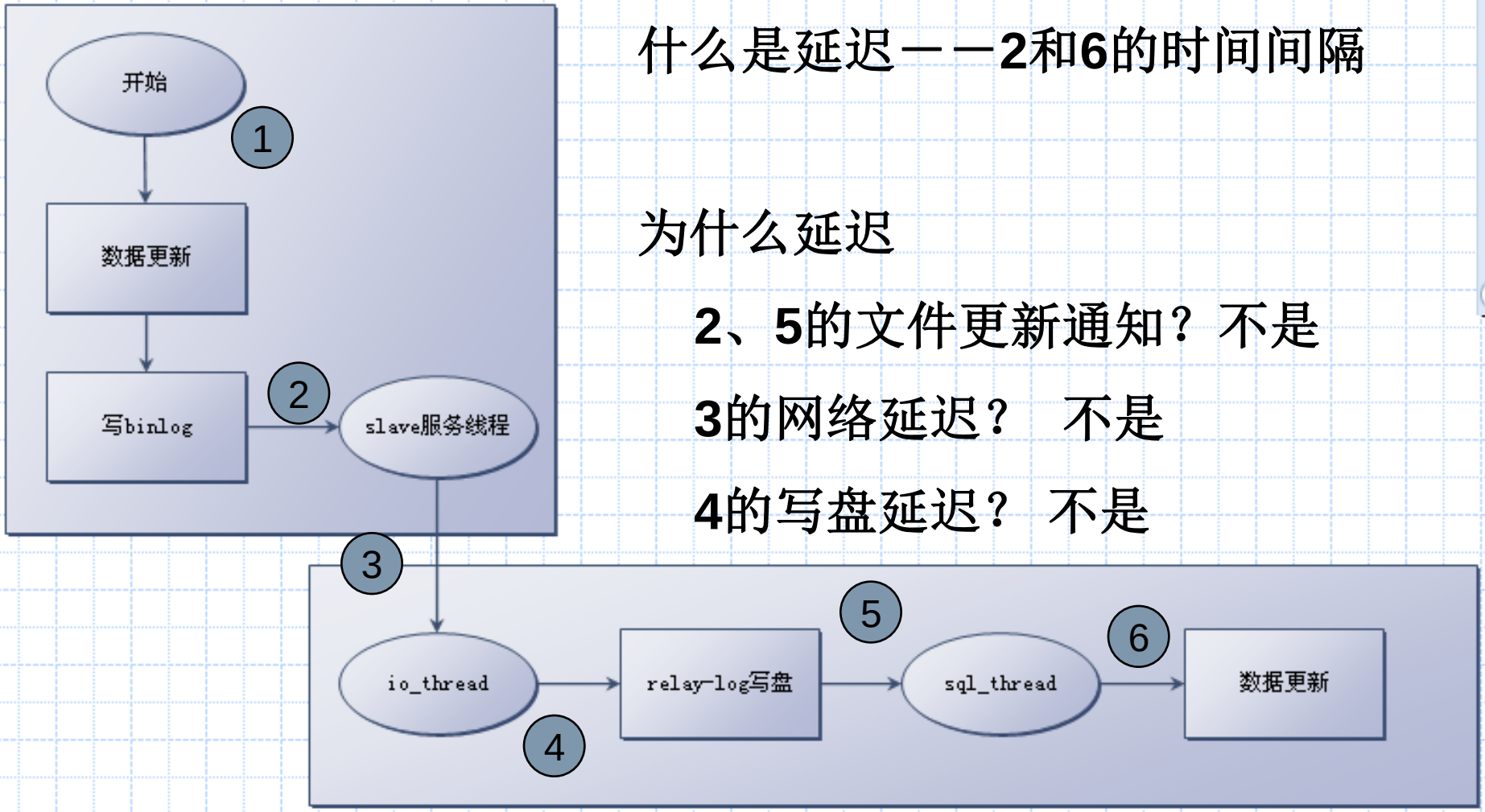
八．不能解决的光速问题

九．不能解决的更新延迟

MySQL主从同步基本流程



MySQL主从同步延迟原因



什么是延迟——2和6的时间间隔

为什么延迟

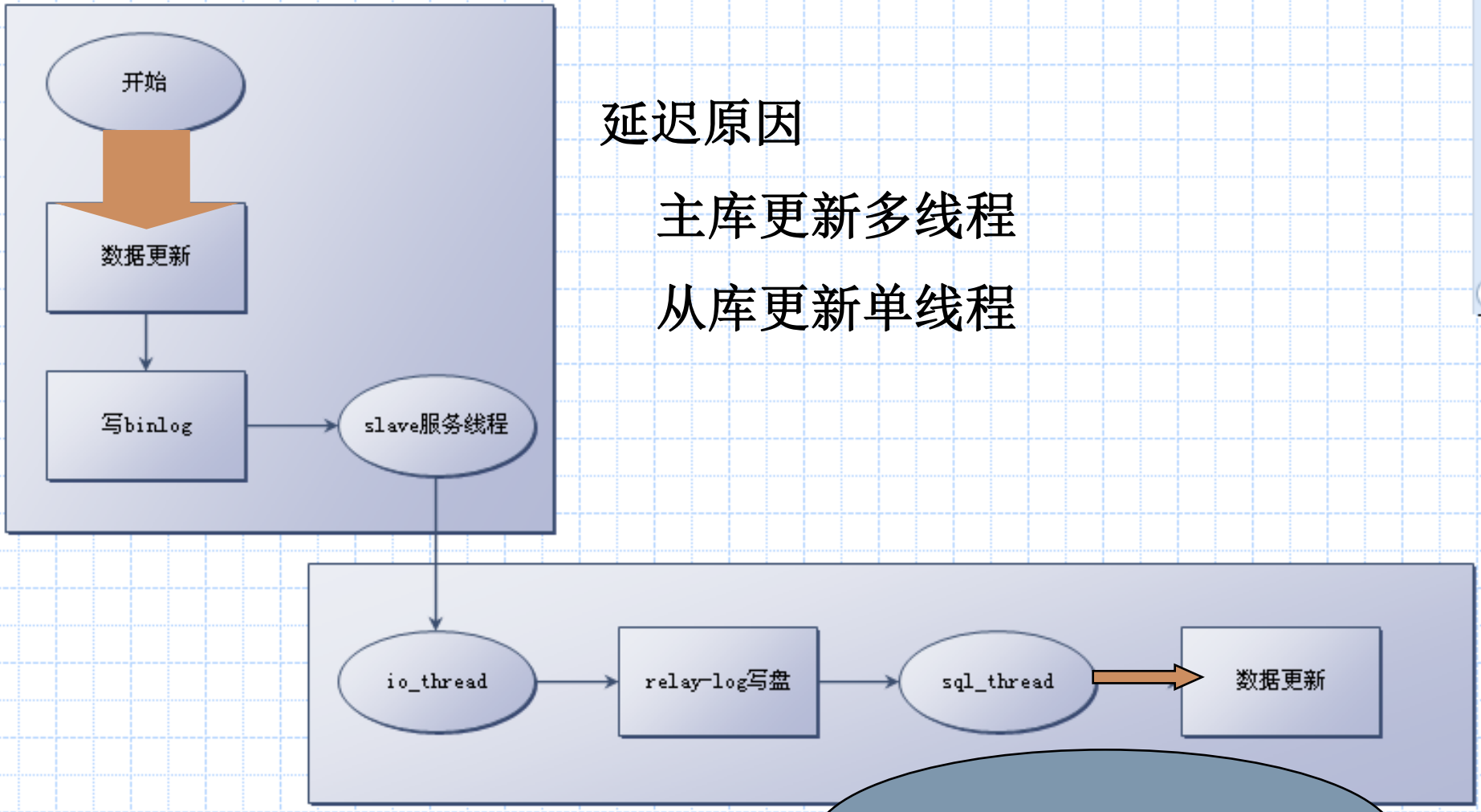
2、5的文件更新通知？不是

3的网络延迟？ 不是

4的写盘延迟？ 不是

等等。。。1和2之间那个箭头怎么不画出来——我们不关心

MySQL主从同步延迟原因



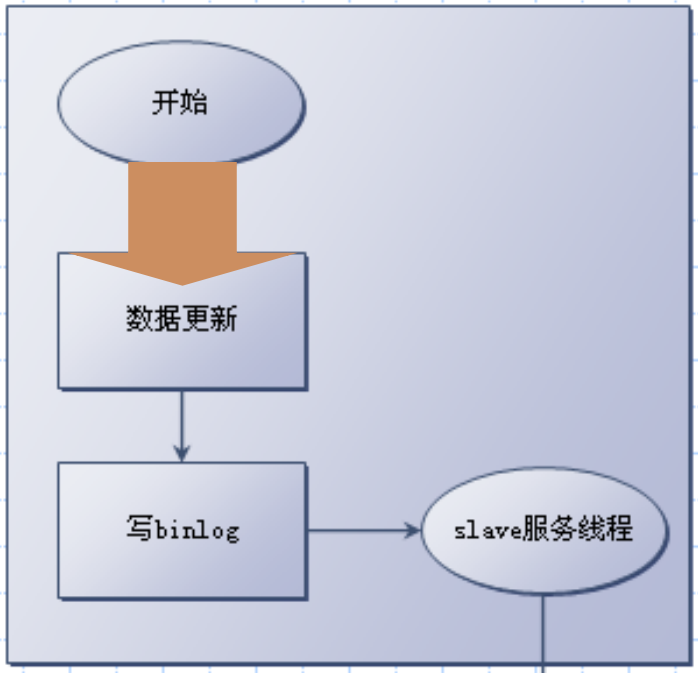
延迟原因

主库更新多线程

从库更新单线程

都是箭头，你咋这么苗条呢？

MySQL主从同步解决方案

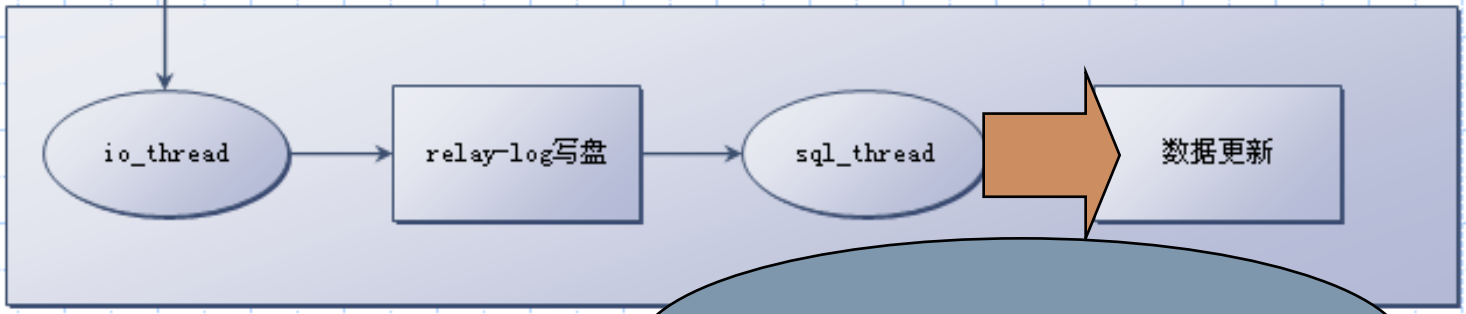


解决方案：

从库变成多线程更新

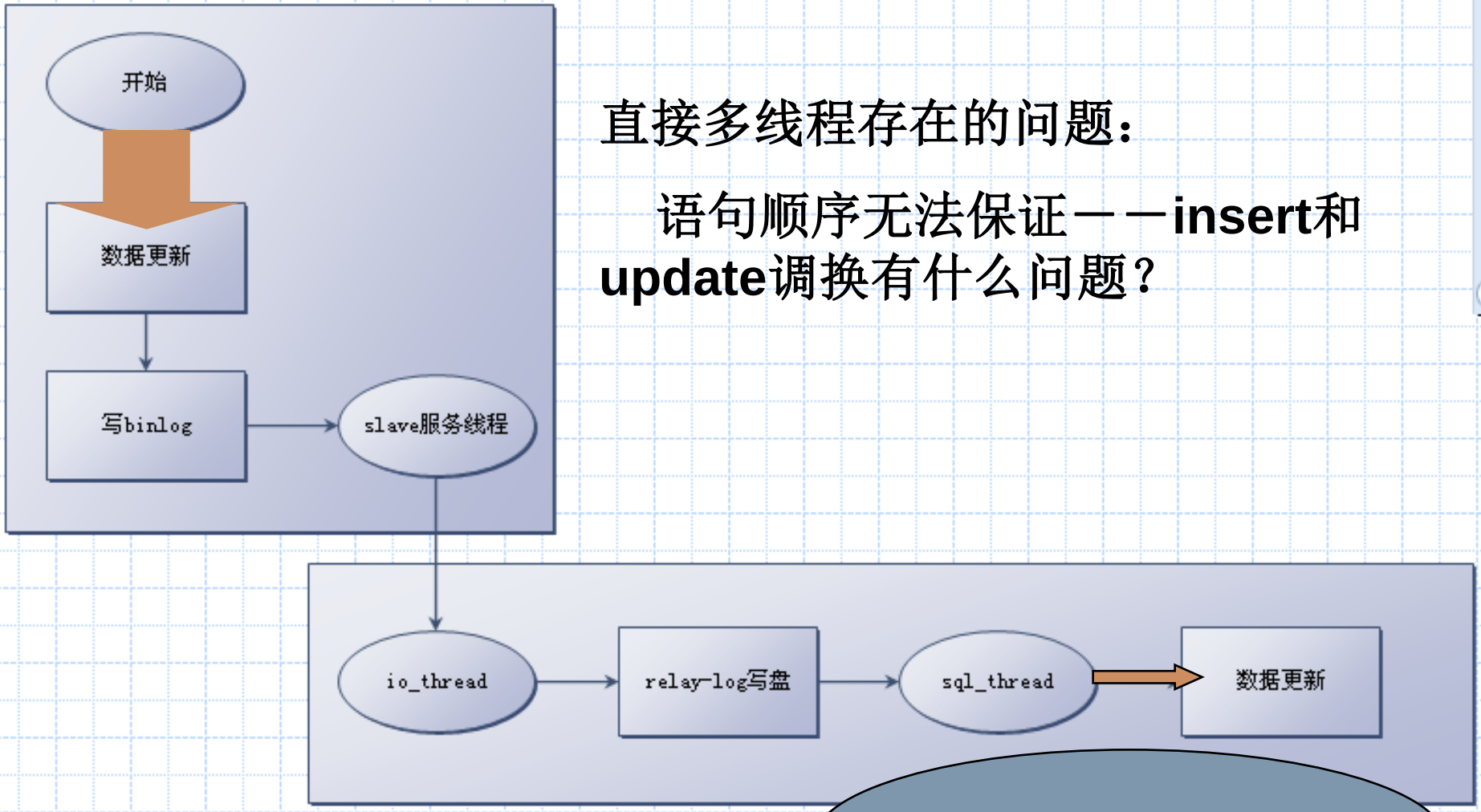
反问一句：

三秒钟变格格么。有那么好MySQL为什么不支持？



说胖就胖了啊。。。

MySQL主从同步解决方案

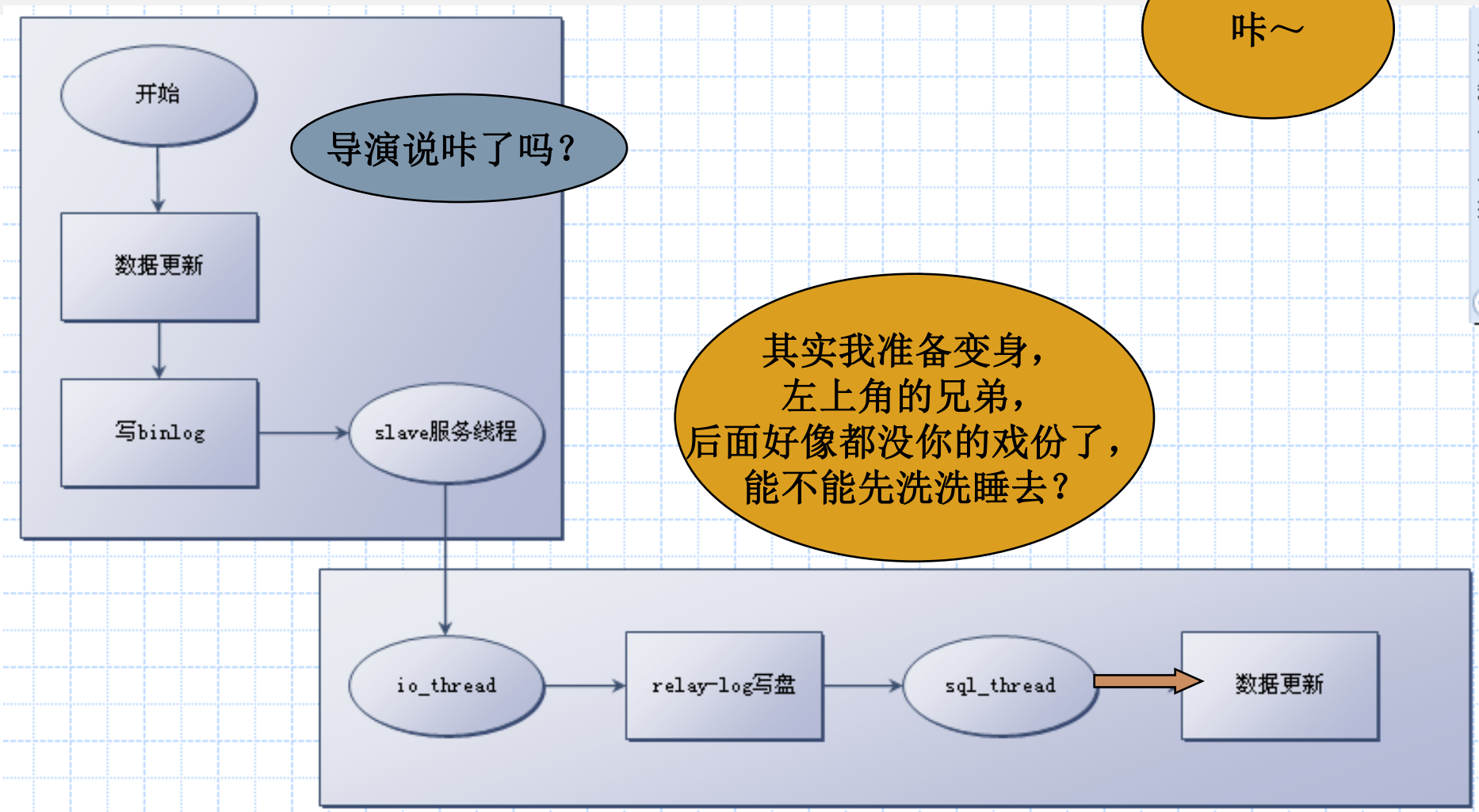


直接多线程存在的问题：

语句顺序无法保证——insert和update调换有什么问题？

又瘦回去了，怂了。。。

MySQL主从同步解决方案



MySQL主从同步解决方案

解决方案分析：

1、一定要多线程！为什么？

2、多线程又会打乱顺序

3、总是有些没那么严格的，是吧？

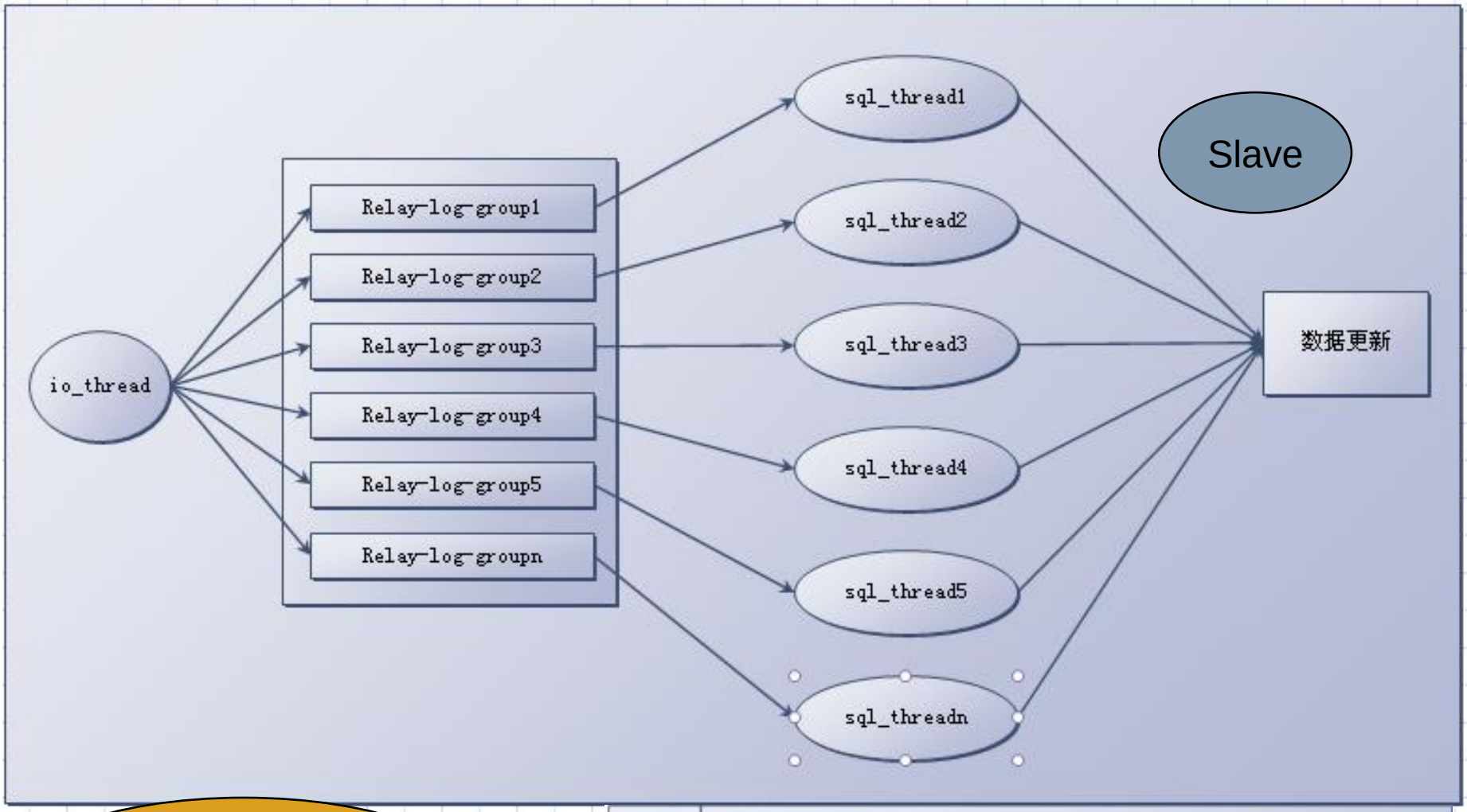
4、同一个表的更新必须按照顺序

5、不同表呢？

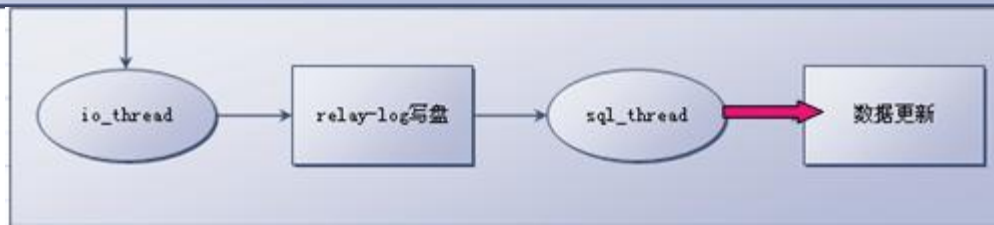
6、先作个不同表之间并行的，线上一个库都有很多表

过渡太久了吧，变身的那位呢？

MySQL主从同步解决方案



认不出来了，来个对比照



MySQL主从同步解决方案

应该是解决了

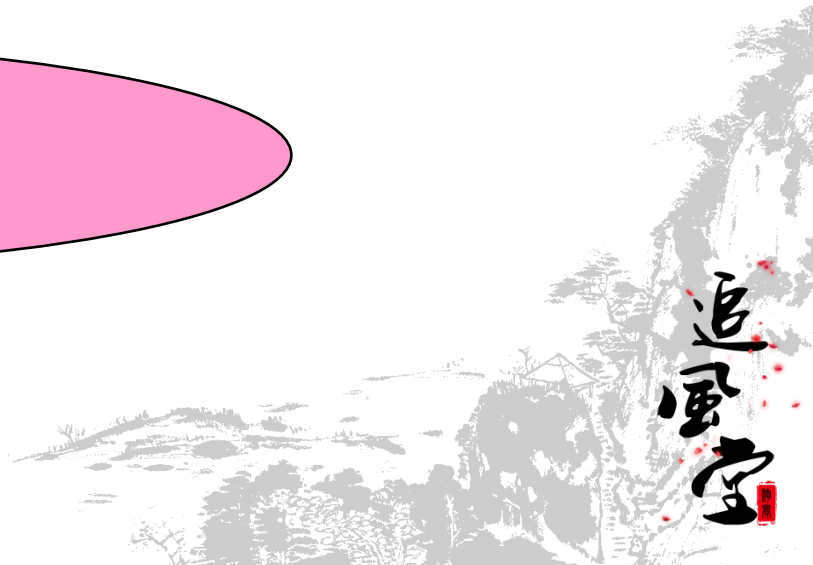
从此Master和Slave过着幸福的生活？

太naïve了。。。

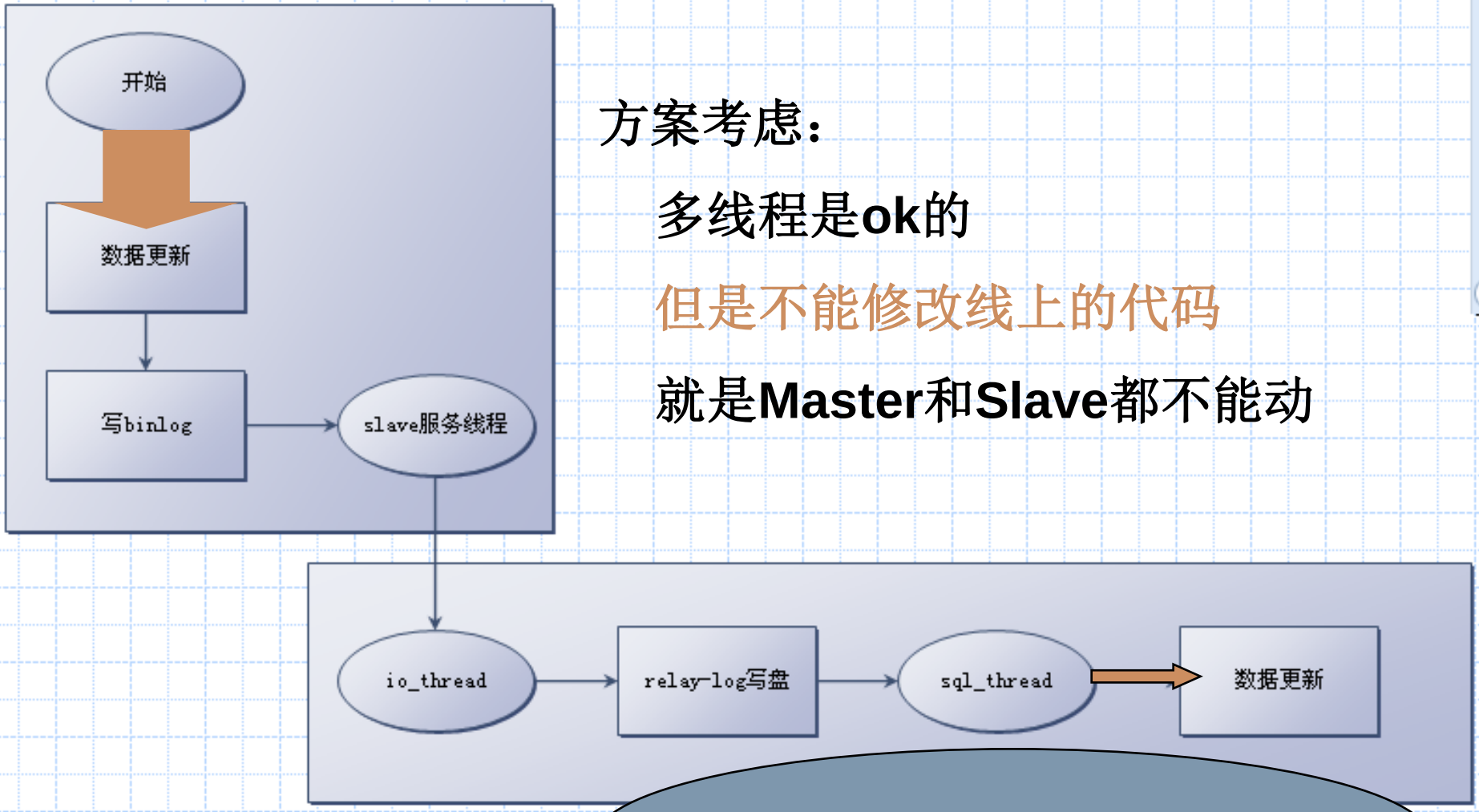
实际上，刚才那个是副导演
导演回来了，说：

咱这剧本不允许主角变身！

未完待续



MySQL主从同步解决方案



方案考虑：

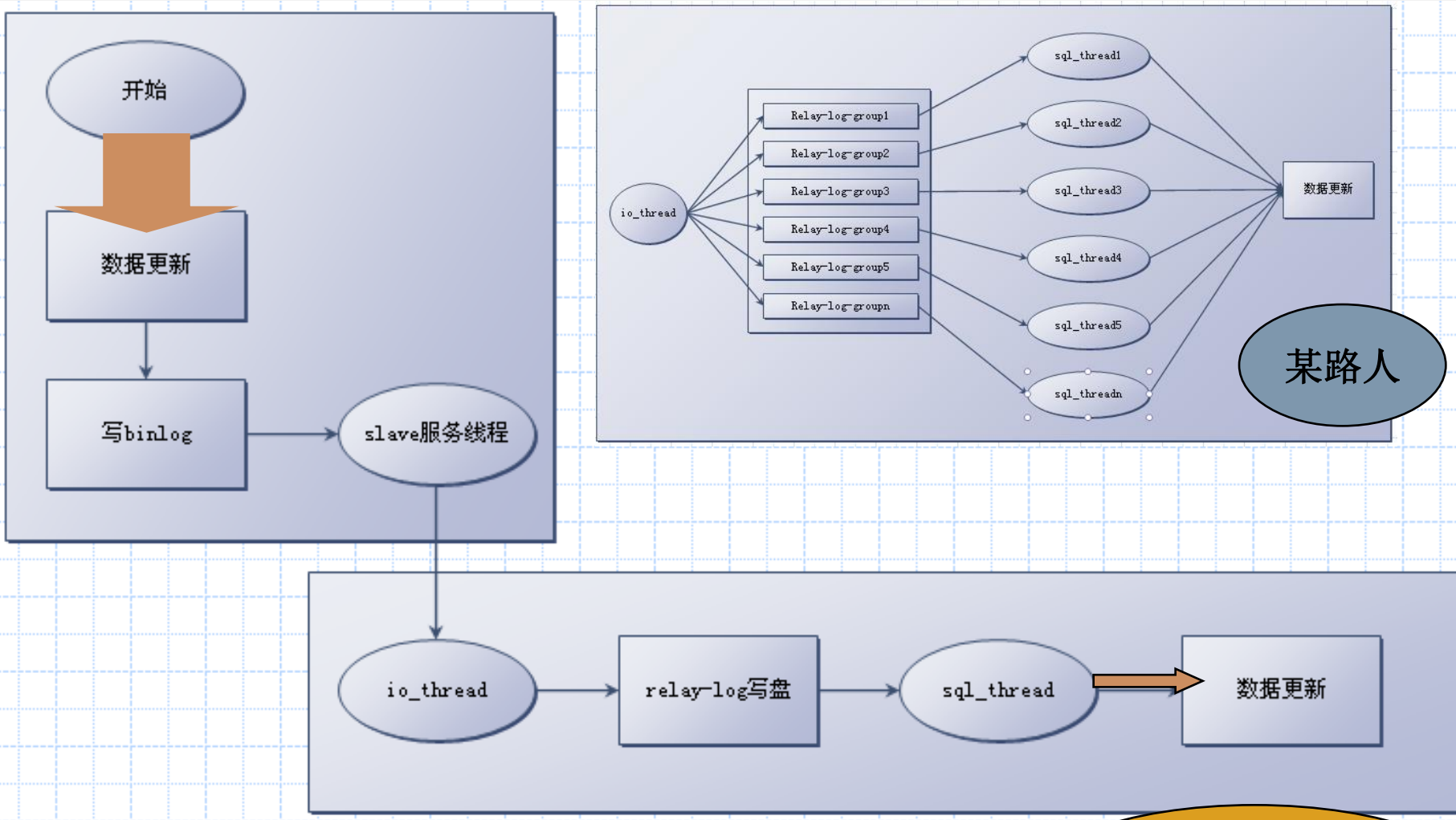
多线程是ok的

但是不能修改线上的代码

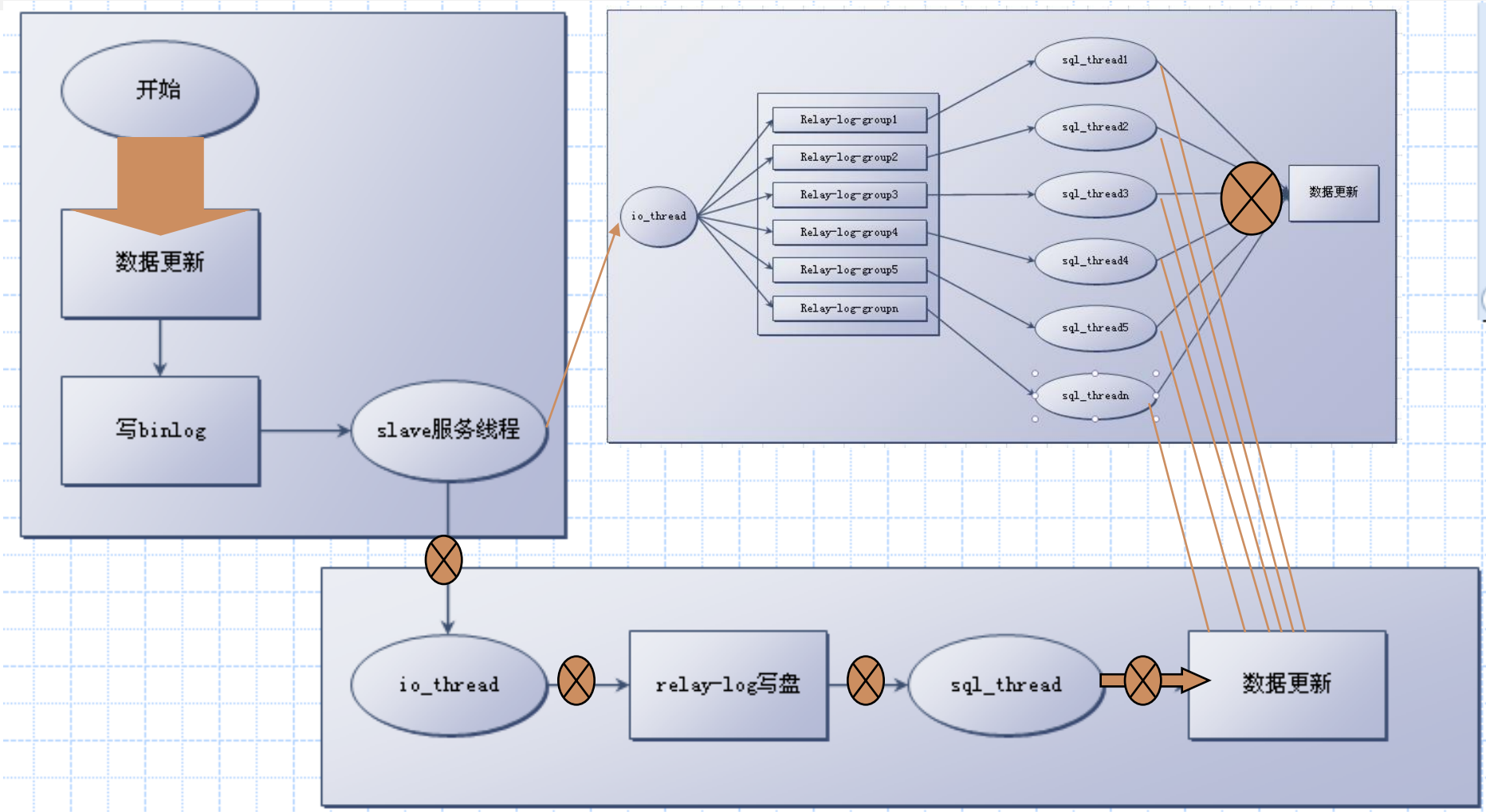
就是Master和Slave都不能动

变回来了，导演管饭，听导演的

MySQL主从同步解决方案



MySQL主从同步解决方案



MySQL主从同步解决方案

以上为前传，介绍MySQL多线程同步工具(Transfer)的设计思路

以下为文字解释版

1. MySQL的主从同步延迟，是指从库的更新性能低于主库的更新性能。
2. 相同的机器配置，出现性能差异的原因，是从库上单线程更新。

主库QPS

Com_insert	18619
Com_insert	16932
Com_insert	18146
Com_insert	18367
Com_insert	19144
Com_insert	18644
Com_insert	18201
Com_insert	19423
Com_insert	17976

从库QPS

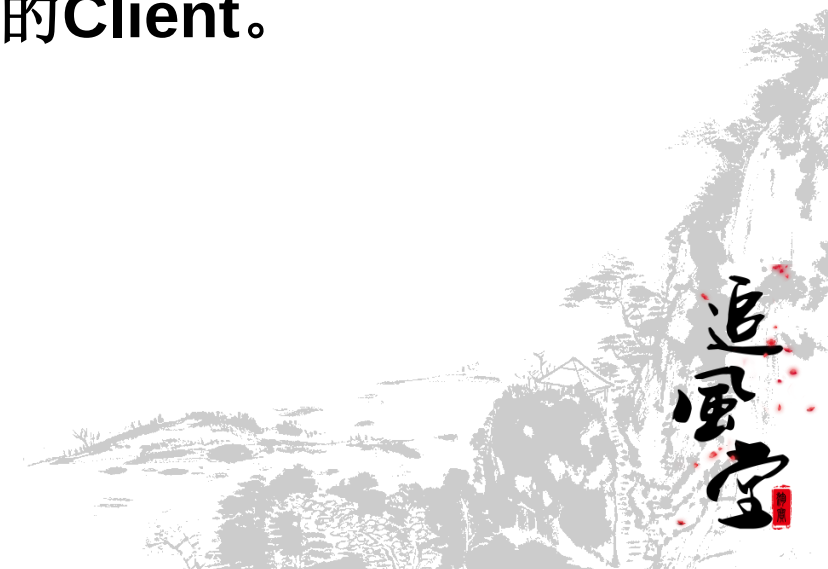
Com_insert	2738
Com_insert	2257
Com_insert	2044
Com_insert	2163
Com_insert	2332
Com_insert	2086
Com_insert	2477
Com_insert	1847
Com_insert	2303

从库vmstat

-cpu-				
id	wa	st		
0	100		0	0
5	4	91	0	0
4	3	92	0	0
6	3	91	0	0
5	4	91	0	0
6	3	90	0	0

MySQL主从同步解决方案

3. 一种方案是将从库的单线程apply改成多线程，但需要修改slave的代码。
4. 安全起见，以工具的形式提供多线程同步功能。
5. Transfer也是一个MySQL，DBA一般部署在slave同一个机器上，放到/u01/mysql2
6. Transfer设置为Master的从库，接收日志后更新Slave
7. 从Slave来看，Transfer是一个普通的Client。



一．MySQL主从同步基本流程

二．延迟的原因

三．解决方案一

四．解决方案二 ——Transfer

五．应用场景和业务限制

六．保障和退化

七．在多主同步的应用

八．不能解决的光速问题

九．不能解决的更新延迟

Transfer的应用场景和业务限制

1. 多表业务
   - Transfer的策略是在io_thread接收主库日志后，分成16份不同的relay-log存放
   - 再用16个sql_thread分别读取日志分发
   - 确保同一个表的更新语句顺序与主库binlog相同
2. 对Master的限制
   - 主库设置binlog为row模式（不支持Statement的原因）
   - 主库单个语句的binlog不能超过1G (原因说明）
   - 尽量减少一个语句更新两个表

Transfer的应用场景和业务限制

3. 对Slave的限制

- 设置max_allowed_packet = 1G
- 需要一个root权限账号提供给Transfer

4. 对DDL语句的处理

- 0号线程的作用

Transfer的保障和退化

1. 保障

- Transfer本身挂了数据不丢（持久化的数据队列）
- Slave出错重启后，继续同步直接start slave
- Master重启后自动重新同步
- 维护方便。
  - stop slave; change master; slave_skip_errors
  - 直接接入现成监控系统

2. 退化

- Statement模式下某些语句不支持。 支持的语句性能也不提升
- 事务打散
- 从库上不再支持rollback (什么时候从库会收到rollback?)

效果对比

主库QPS

Com_insert	18619
Com_insert	16932
Com_insert	18146
Com_insert	18367
Com_insert	19144
Com_insert	18644
Com_insert	18201
Com_insert	19423
Com_insert	17976

从库QPS

Com_insert	2738
Com_insert	2257
Com_insert	2044
Com_insert	2163
Com_insert	2332
Com_insert	2086
Com_insert	2477
Com_insert	1847
Com_insert	2303

从库vmstat

-cpu------				
id	wa	st		
0	100	0	0	
5	4	91	0	0
4	3	92	0	0
6	3	91	0	0
5	4	91	0	0
6	3	90	0	0

主库QPS

Com_insert	15944
Com_insert	15793
Com_insert	16076
Com_insert	16435
Com_insert	17988
Com_insert	19879
Com_insert	18374
Com_insert	18159
Com_insert	18786

从库QPS

Com_insert	17020
Com_insert	16071
Com_insert	15469
Com_insert	16760
Com_insert	16770
Com_insert	17991
Com_insert	19526
Com_insert	19385
Com_insert	18411

从库vmstat

u------			
wa	st		
100	0	0	
3	79	8	0
3	80	8	0
5	77	4	0
4	83	2	0
4	79	6	0

一．MySQL主从同步基本流程

二．延迟的原因

三．解决方案一

四．解决方案二 ——Transfer

五．应用场景和业务限制

六．保障和退化

七．在多主同步的应用

八．不能解决的光速问题

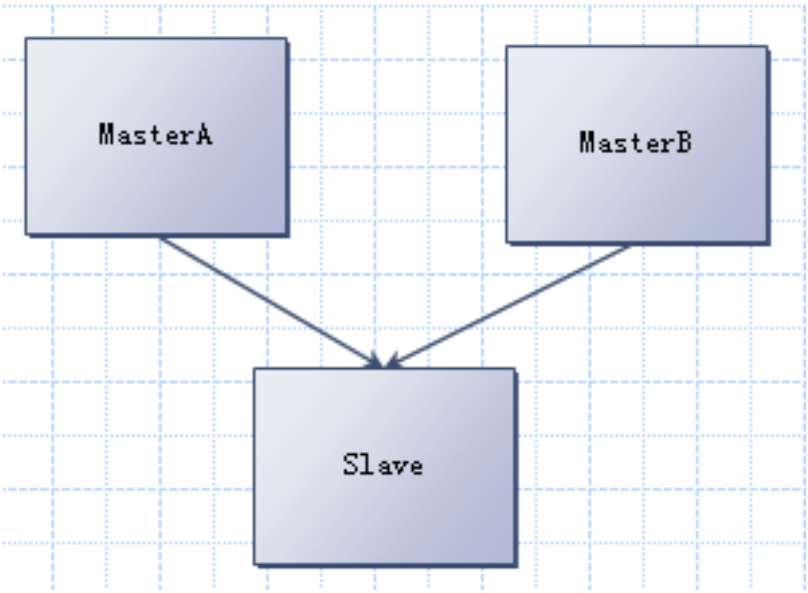
九．不能解决的更新延迟

Transfer在多主同步的应用

多主复制的需求来源

- 备份节约机器
- 数据聚集分析

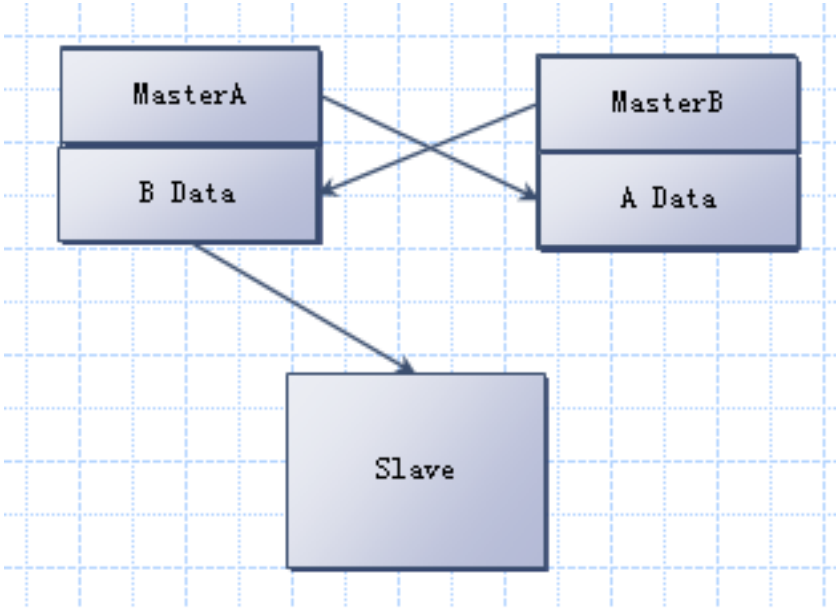
理想方案



MySQL不支持

Transfer在多主同步的应用

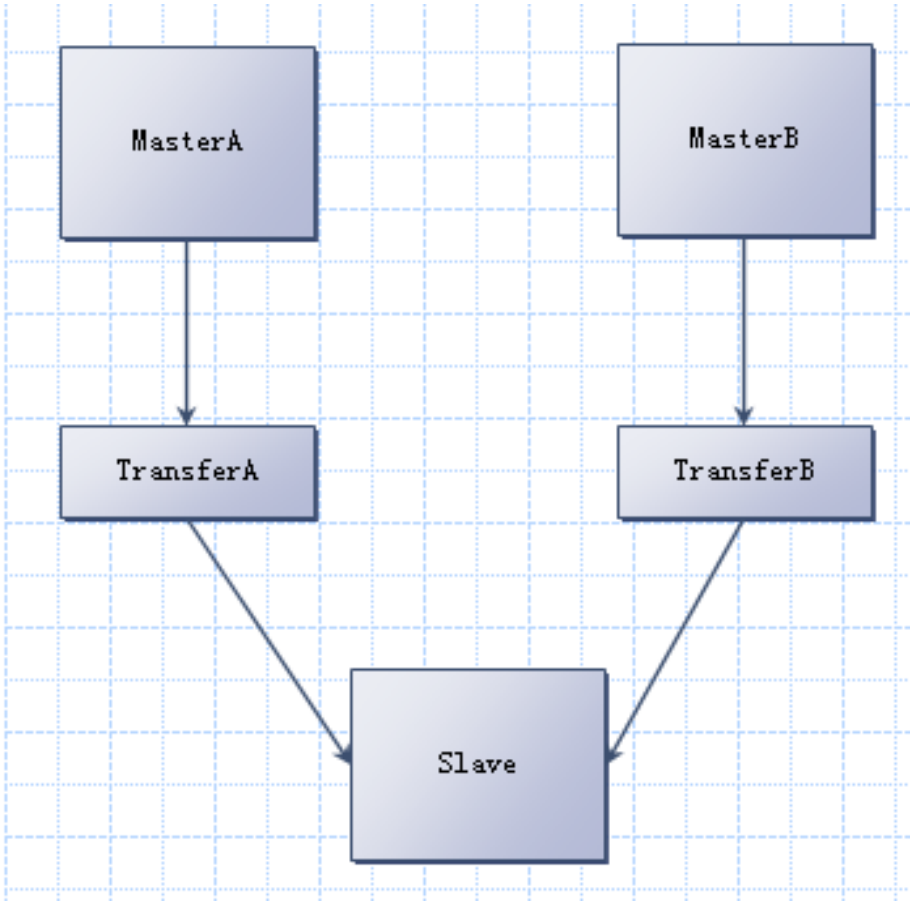
现在方案



- 浪费硬盘空间
- 增加额外更新
- 更大的延迟

Transfer在多主同步的应用

Transfer方案



一．MySQL主从同步基本流程

二．延迟的原因

三．解决方案一

四．解决方案二 ——Transfer

五．应用场景和业务限制

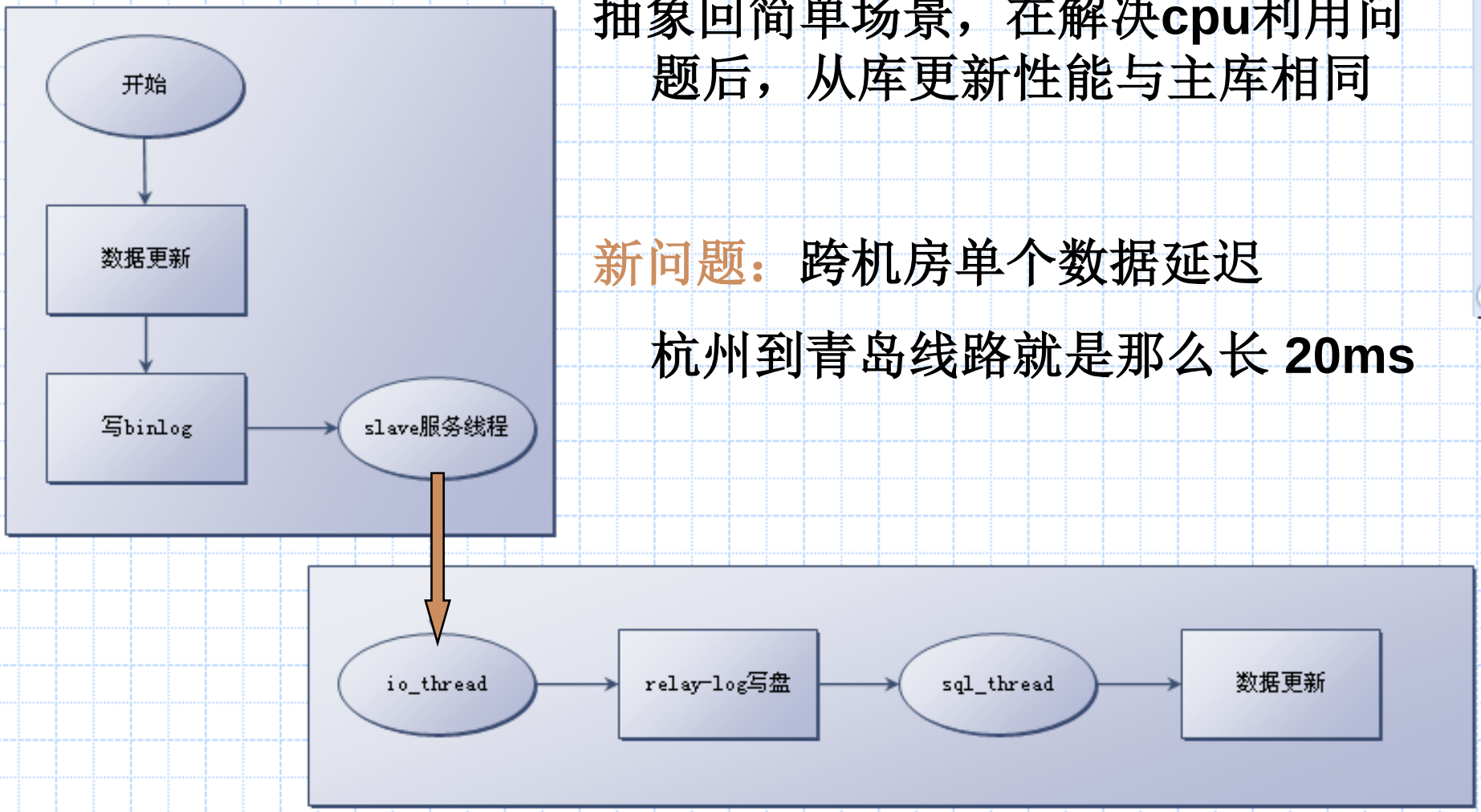
六．保障和退化

七．在多主同步的应用

八．不能解决的光速问题

九．不能解决的更新延迟

无法解决的光速问题

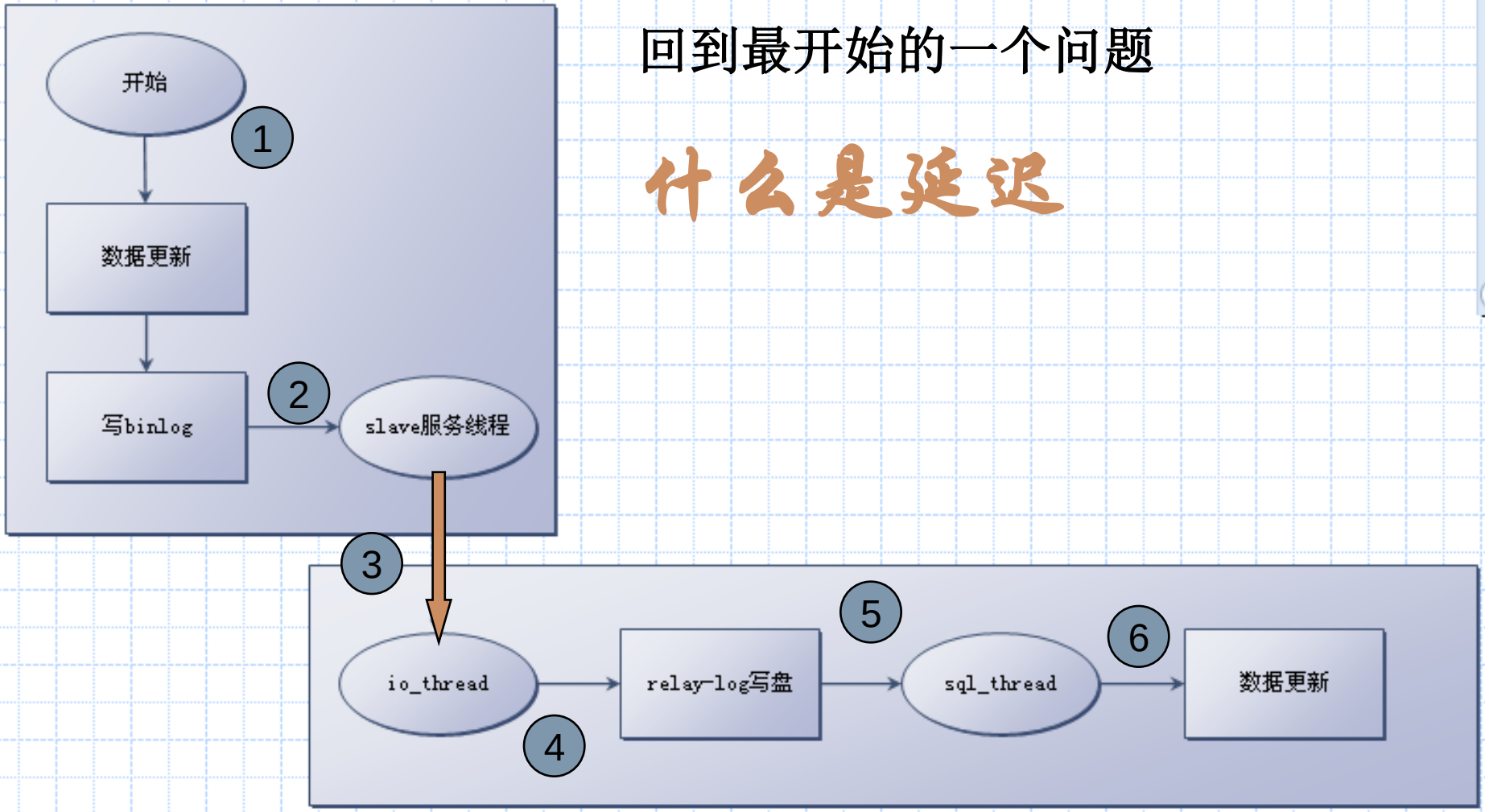


抽象回简单场景，在解决cpu利用问题后，从库更新性能与主库相同

新问题：跨机房单个数据延迟

杭州到青岛线路就是那么长 20ms

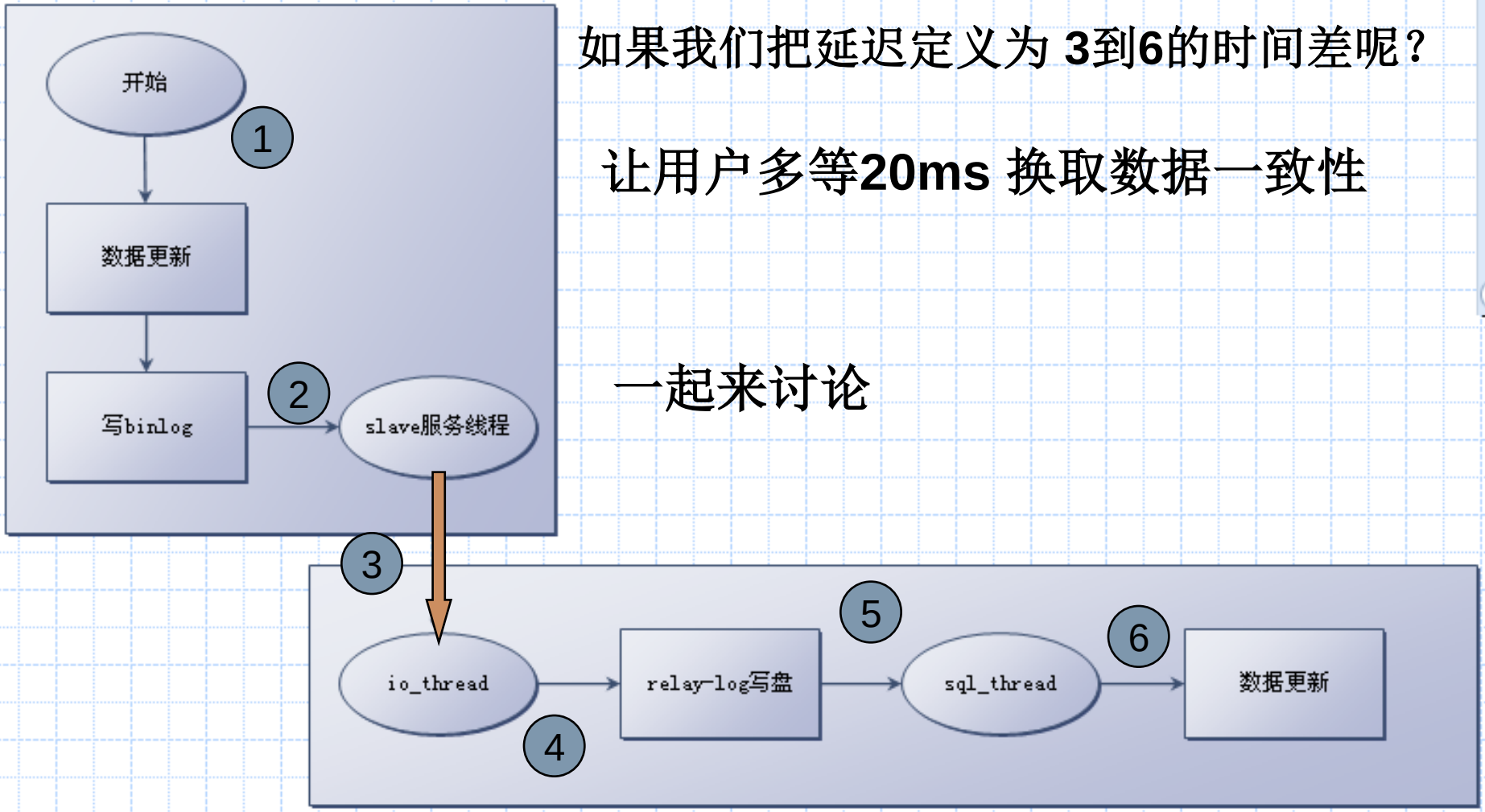
无法解决的光速问题



回到最开始的一个问题

什么是延迟

无法解决的光速问题



如果我们把延迟定义为 3到6的时间差呢？

让用户多等20ms 换取数据一致性

一起来讨论

一．MySQL主从同步基本流程

二．延迟的原因

三．解决方案一

四．解决方案二 ——Transfer

五．应用场景和业务限制

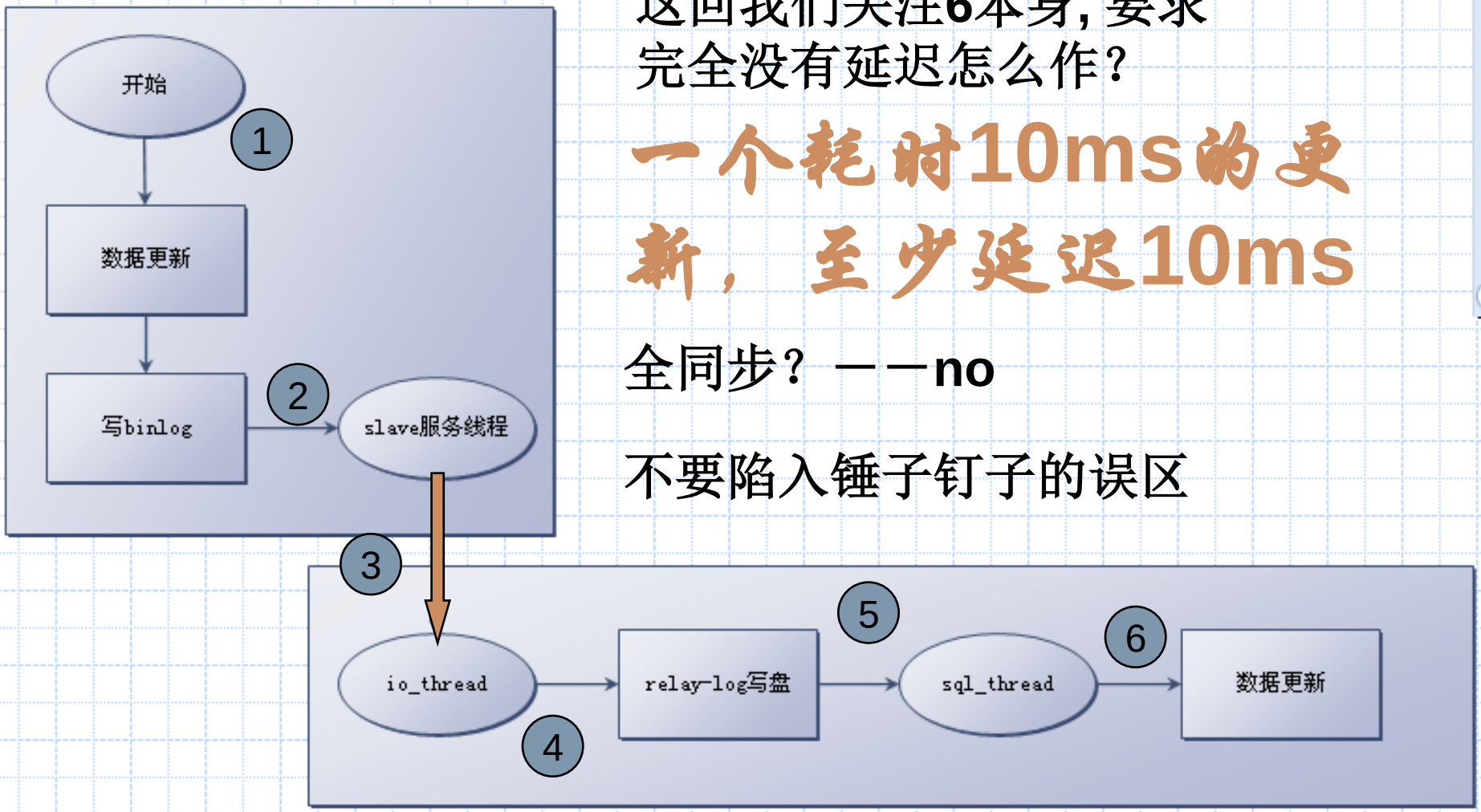
六．保障和退化

七．在多主同步的应用

八．不能解决的光速问题

九．不能解决的更新延迟

不能解决的更新延迟



这回我们关注6本身，要求完全没有延迟怎么作？

一个耗时10ms的更新，至少延迟10ms

全同步？——no

不要陷入锤子钉子的误区

放弃这方案，用双写

谢谢