Love your Data

用户DB架构部

by Keithlan

Agenda


Enterprise
MySQL
mysqldump
Snapshot
RTO
Disaster
Methods
incremental
instances
engines
Warm
product
physical
Backing
compressed
data
NFS
files
Backup
features
Hot
databases
using
FULL
Restore
storage
much
use
summarizes
copied
Customers
techniques
mechanism
selected
Database
InnoDB
backups
netcat
faster
command
Copy
Xtrabackup
Binary
Zmanda
tables
logical
Logs
substantial
RPO
Write
Recovery
Binlog
Cold

Backups

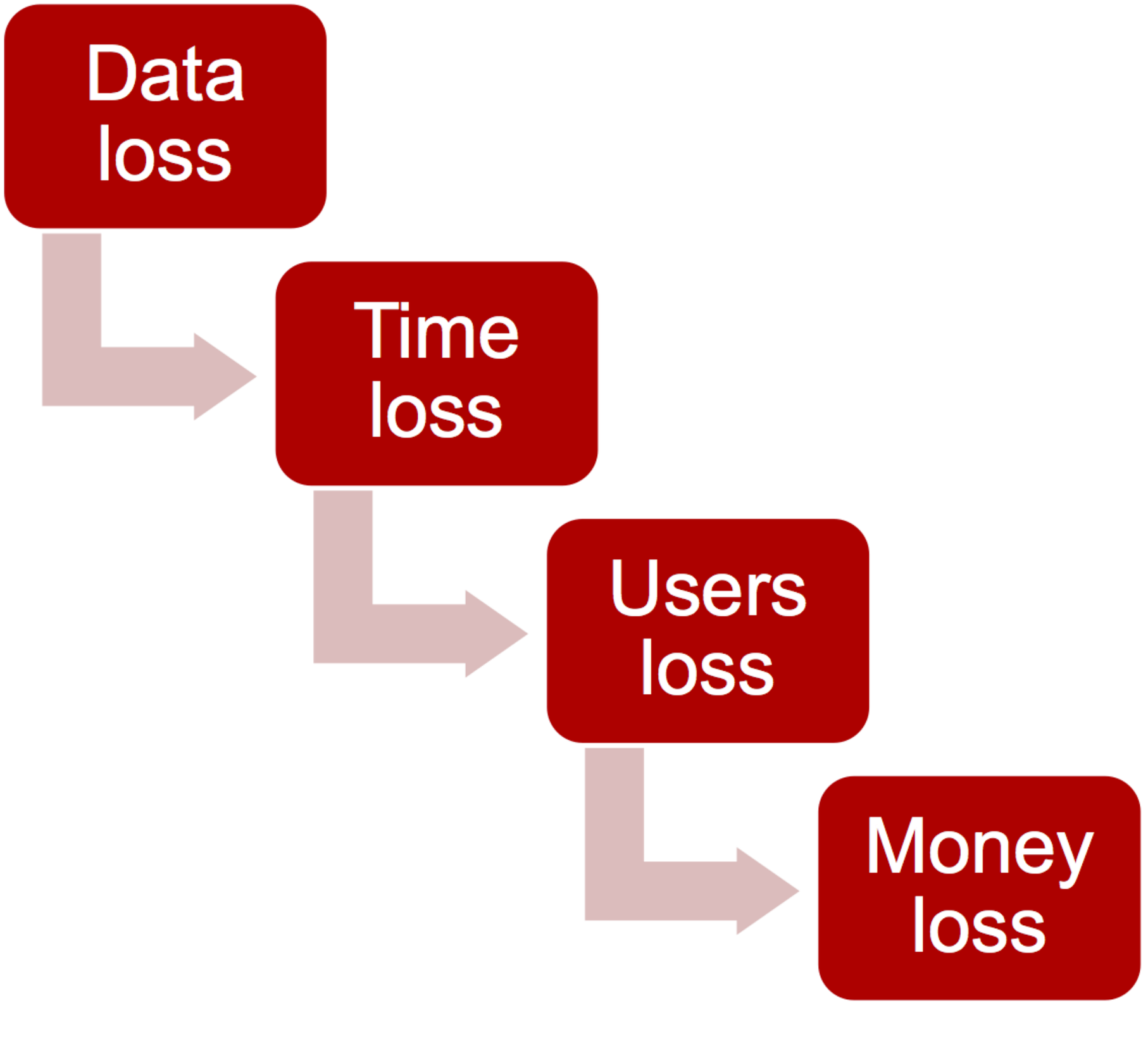
Backups **do not** give you

More users

More money

Increase productivity

Not having backup



MySQL Backup Types

Hot vs. Warm vs. Cold

Logical vs. Physical

Local vs. Remote

Full vs. Point in Time

MySQL Backup Types

Hot vs. Warm vs. Cold

Can you take your MySQL server offline to make a backup?

Do you have a replica where your backup will not impact the master?

Logical vs. Physical

Backup the files or dump out the data so that you can recreate your server

Local vs. Remote

Can you afford the latency of a network round-trip

Full vs. Point in Time

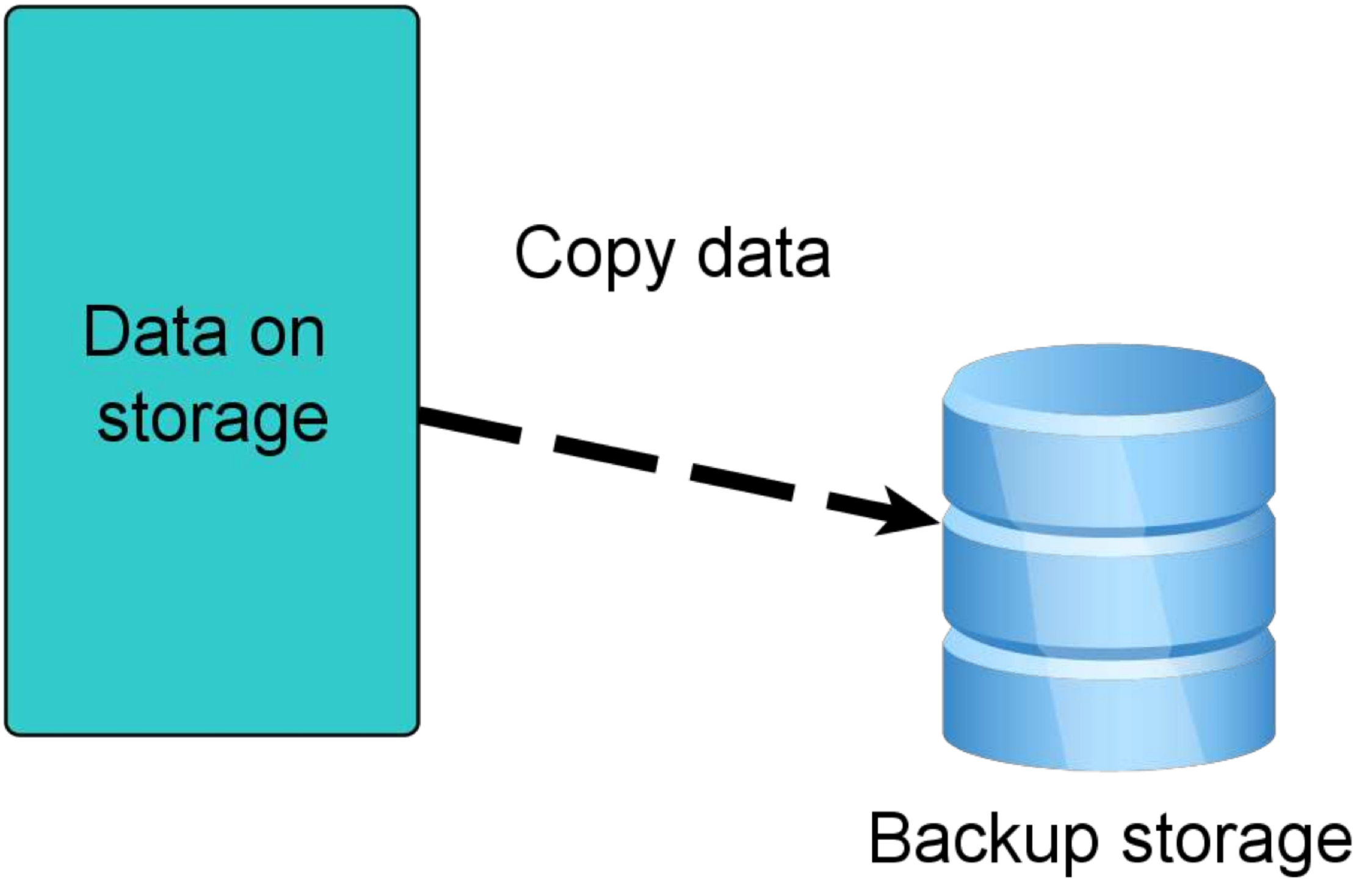
Linked to your RPO, how granular should you go?

Backup Repository Model

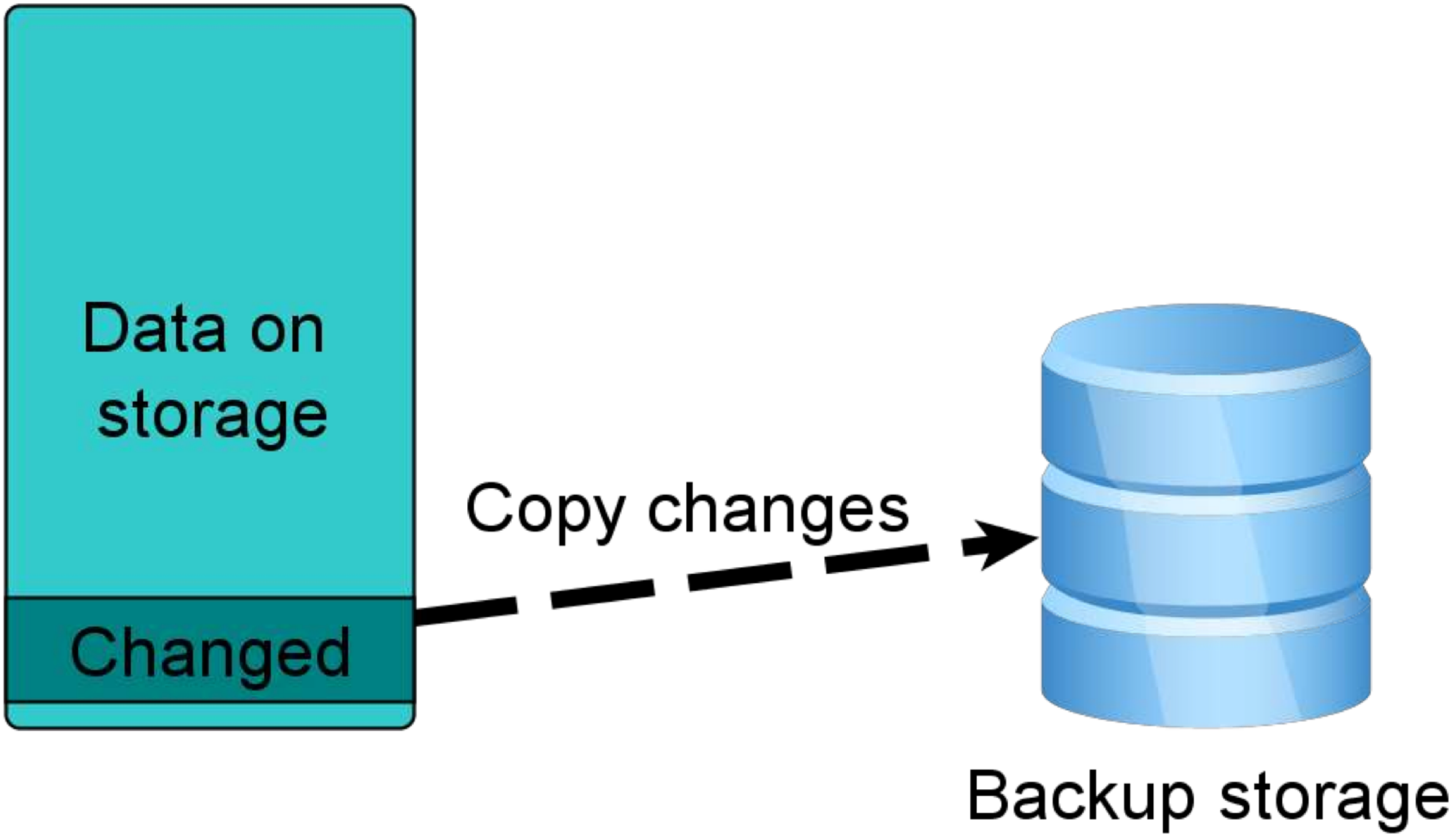
Backups need to be stored and organized

FULL: Complete system images
DIFFERENTIAL: Changes since last full backup
INCREMENTAL: Changes between two points in time.

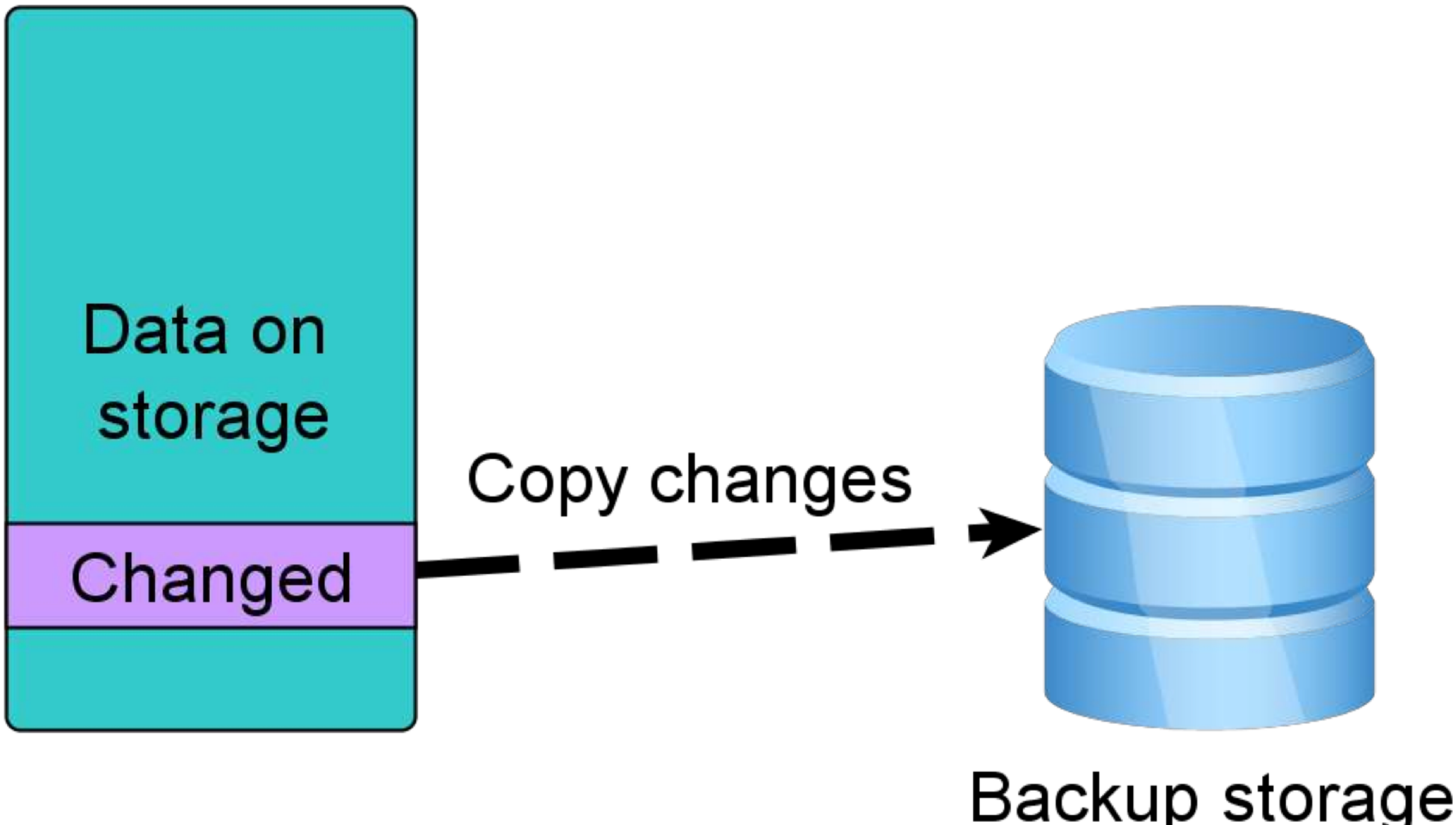
Full copy



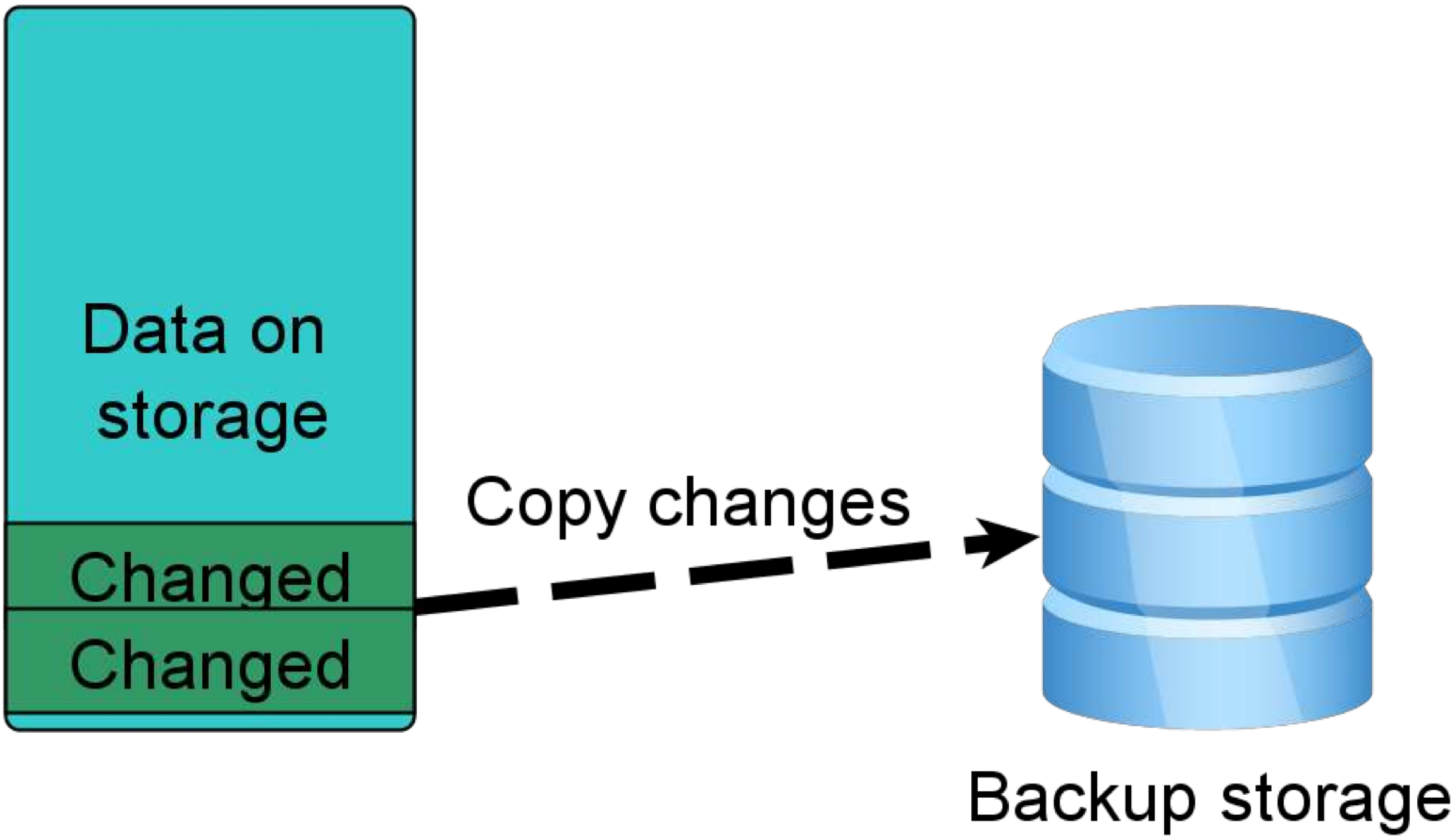
Incremental. Day 1 – changes since full



Incremental. Day 2 – changes since Incremental



Differential. Changes since Full



MySQL Backup Tools

Logical

- mysqldump
- mydumper

Physical

- Cold Backup
- MySQL Enterprise Backup
- Xtrabackup

Snapshot

- SAN
- LVM
- ZFS

Frameworks

- Zmanda
- Holland
- Xtrabackup Manager

Flush tables Vs Flush tables with read lock

1. lock_global_read_lock
 --> mdl_request.init(MDL_key::GLOBAL, "", "", MDL_SHARED, MDL_EXPLICIT);

2. close_cached_tables
 （FLUSH TABLES只做这一步）
 -- 关闭表前，需要等待所有查询都结束后才能关闭

3. make_global_ock_block_commit
--> mdl_request.init(MDL_key::COMMIT, "", "", MDL_SHARED, MDL_EXPLICIT);

for upate情况

```
Qestion1:
Session 1 : select * from ac_balance_log_balance for update ;
Session 2 : flush tables
Session 3 : select * from ac_balance_log_balance limit 1 ;
Session 4 : show processlist;
dbadmin:crm_sale_db> show processlist;
+----------+---------+-----------+-------------+---------+------+------------------+-----------------------------------------------+
| Id       | User    | Host      | db          | Command | Time | State            | Info                                          |
+----------+---------+-----------+-------------+---------+------+------------------+-----------------------------------------------+
| 77744131 | dbadmin | localhost | crm_sale_db | Query   |   11 | Sending data     | select * from ac_balance_log_balance for update |
| 77744148 | dbadmin | localhost | crm_sale_db | Query   |    8 | Flushing tables  | flush tables                                  |
| 77744181 | dbadmin | localhost | crm_sale_db | Query   |    5 | Waiting for table | select * from ac_balance_log_balance limit 1  |
| 77744231 | dbadmin | localhost | crm_sale_db | Query   |    0 | NULL             | show processlist                              |
+----------+---------+-----------+-------------+---------+------+------------------+-----------------------------------------------+
4 rows in set (0.00 sec)
```

```
Session 1 : select * from ac_balance_log_balance for update ;
Session 2 : flush tables with read lock;
Session 3 : select * from ac_balance_log_balance limit 1 ;
Session 4 : show processlist;

dbadmin:crm_sale_db> show processlist;
+----------+---------+-----------+-------------+---------+------+-------------------------+-----------------------------------------------+
| Id       | User    | Host      | db          | Command | Time | State                   | Info                                          |
+----------+---------+-----------+-------------+---------+------+-------------------------+-----------------------------------------------+
| 77744131 | dbadmin | localhost | crm_sale_db | Query   |   35 | Sending data            | select * from ac_balance_log_balance for update |
| 77744148 | dbadmin | localhost | crm_sale_db | Query   |   15 | Waiting to get readlock | flush tables with read lock                   |
| 77744181 | dbadmin | localhost | crm_sale_db | Sleep   |   12 |                         | NULL                                          |
| 77744231 | dbadmin | localhost | crm_sale_db | Query   |    0 | NULL                    | show processlist                              |
+----------+---------+-----------+-------------+---------+------+-------------------------+-----------------------------------------------+
4 rows in set (0.00 sec)
总结： 对于X锁，flush tables 会阻塞当前表的查询，但是flush tables with read lock不会，因为它还处在wating to get read lock阶段，还没进行到加全局读锁，flush tables阶段。
```

select 时间很长的情况

Qestion 2:
Session 1 : select * from ac_balance_log_balance;
Session 2 : flush tables
Session 3 : select * from ac_balance_log_balance limit 1 ;
Session 4 : show processlist;
dbadmin:crm_sale_db> show processlist;

Id	User	Host	db	Command	Time	State	Info
77744131	dbadmin	localhost	crm_sale_db	Query	9	Writing to net	select * from ac_balance_log_balance
77744148	dbadmin	localhost	crm_sale_db	Query	5	Flushing tables	flush tables
77744181	dbadmin	localhost	crm_sale_db	Query	2	Waiting for table	select * from ac_balance_log_balance limit 1
77744231	dbadmin	localhost	crm_sale_db	Query	0	NULL	show processlist

4 rows in set (0.00 sec)

Session 1 : select * from ac_balance_log_balance;
Session 2 : flush tables with read lock;
Session 3 : select * from ac_balance_log_balance limit 1 ;
Session 4 : show processlist;
dbadmin:crm_sale_db> show processlist;
dbadmin:crm_sale_db> show processlist;

Id	User	Host	db	Command	Time	State	Info
77744131	dbadmin	localhost	crm_sale_db	Query	13	Sending data	select * from ac_balance_log_balance
77744148	dbadmin	localhost	crm_sale_db	Query	7	Flushing tables	flush tables with read lock
77744181	dbadmin	localhost	crm_sale_db	Query	4	Waiting for table	select * from ac_balance_log_balance limit 1
77744231	dbadmin	localhost	crm_sale_db	Query	0	NULL	show processlist

4 rows in set (0.00 sec)

总结：对于普通的Select 扫描（执行时间很长），flush tables 和 flush tables with read lock 都是一样的，除了会锁住被扫描表的查询外（因为都要关闭表），flush tables with read lock 还会加上全局读锁，也就是所有的写都会被锁住。

mysqldump

Type: *Logical* ***Heat:*** *Hot[innodb only]:Warm[myisam]* ***Impact:*** *Medium* ***Speed:*** *Slow*

The command line utility to create logical dumps of your schema, database objects and data Good solution for small to medium datasets (0G>20G)

Pros

- Packaged with MySQL
- Broad compatibility (engines)
- Flexible use with pipelines (gzip, sed, awk, pv, ssh, nc)
- No locking in --single-transaction with innodb only tables

Cons

- Single threaded
- Locking by default
- Can be hard to troubleshoot errors (*syntax error on line 14917212938*)
- Slow to reload data
- Be wary of foreign keys and triggers when restoring.

mysqldump

Backup Examples

Backup all tables

```
mysqldump -u user -p pass --all-databases > backup.sql
```

Backup all tables compressed

```
mysqldump -u user -p pass --all-databases | gzip -5 > backup.sql.gz
```

Backup with database objects

```
mysqldump -u user -p pass --routines --triggers --events > backup.sql
```

Backup with no data

```
mysqldump -u user -p pass --no-data --triggers -events > backup.sql
```

mysqldump

Restore Examples

Restore mysqldump

```
mysql -u user -p < backup.sql
```

Restore from within

```
source backup.sql;
```

Restore a compressed dump binlog off

```
(echo "set session sql_log_bin=0;" ; zcat dump.sql.gz) | mysql –u user
```

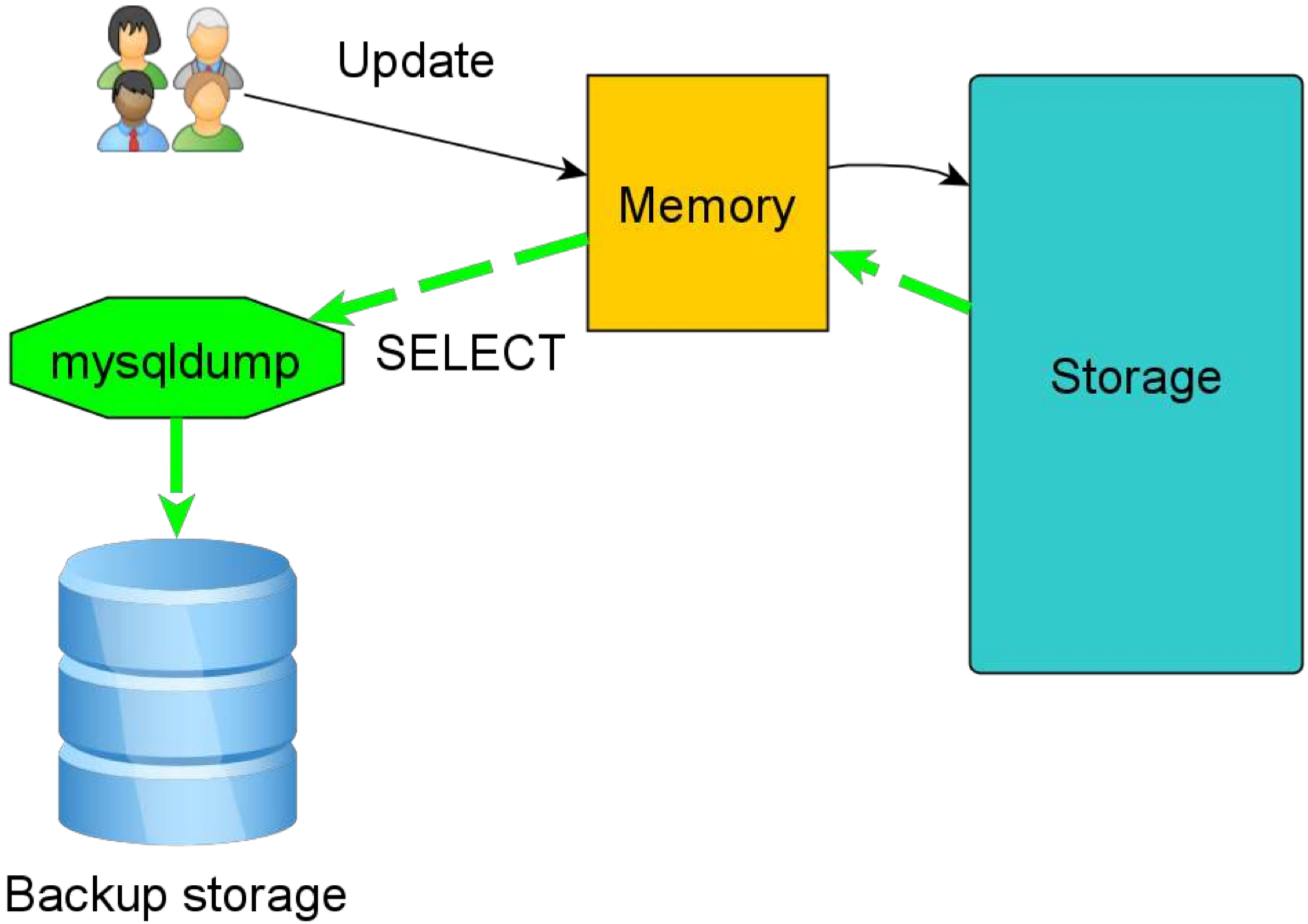
Restore table from dumpfile using sed

```
cat dump.sql | sed -n '/^-- Table structure for table `t1`/,/^UNLOCK TABLES;/p' \
| mysql -u user
```

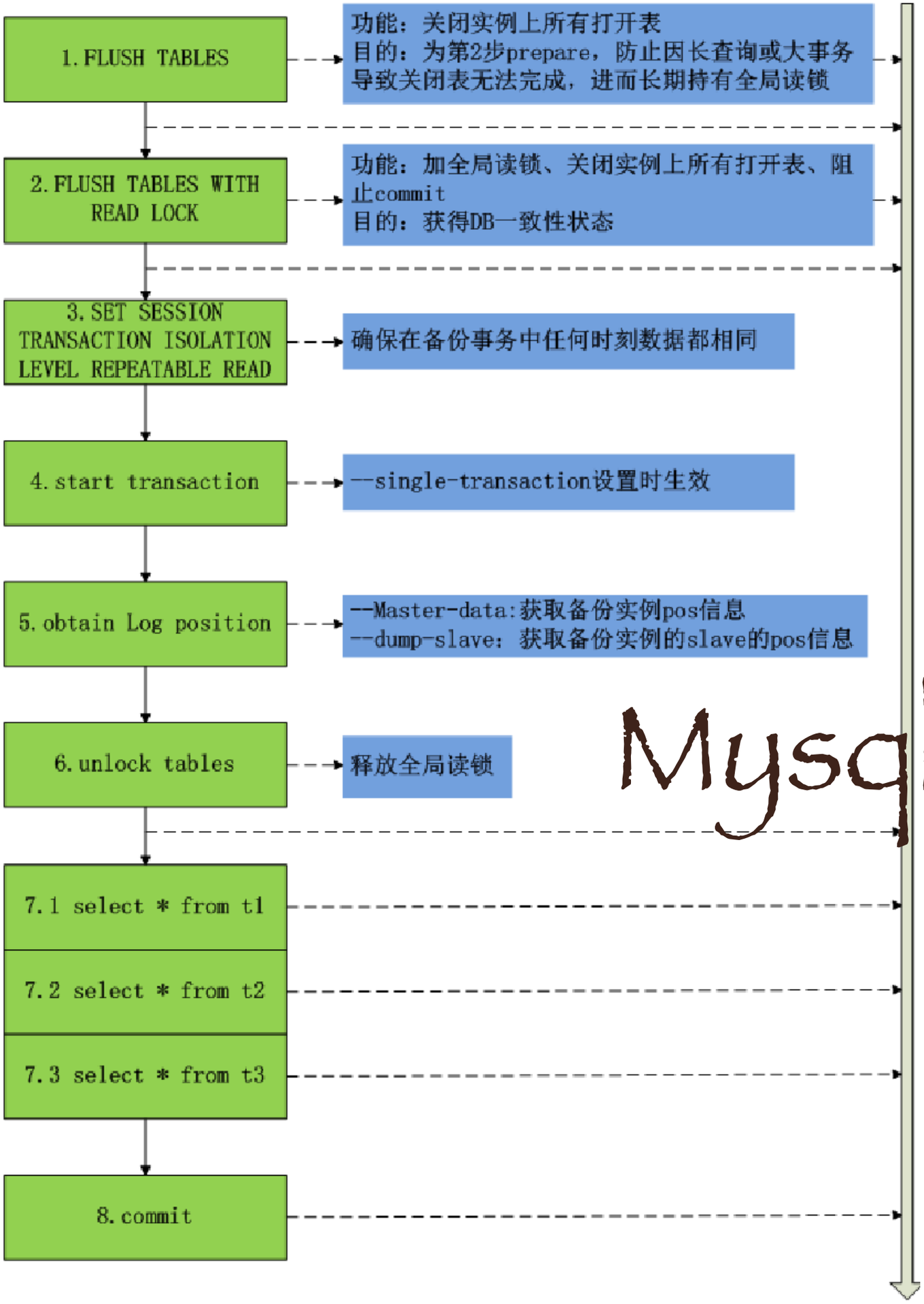
Backup & restore one liner

```
mysqldump db_one | ssh moore@myslave mysql -u user db_one
```

mysqldump



Mysqldump流程图



Mysqldump之Q&A

Q1：再mysqldump—single-transaction之前，有一条很大的update事务在执行，那么dump数据需要等待update事务执行完毕么？如果不是，怎么保证导出的数据是一致?

A1：

1) --single-transaction 是不会锁任何事务的。因为，只要开启set RR模式即可备份，并且通过MVCC保证数据一致性。

2) --single-transaction --master-data 会flush tables && flush tables with read lock； 所以，会等这条大事务执行完毕才会备份。

Q2：如果在mysqldump的过程中，其他的transaction对备份备的表做了DDL，那么备份会成功么?

A2：

1) 对于mysql5.1来说， mysqldump和mydumer都会遭到ddl的破坏，而导致数据不一致备份。

2) 对于mysql5.5来说，由于引入了meta-data lock的概念，所以从某种程度来说会好一些，但是还是会被DDL破坏。

mydumper

https://launchpad.net/mydumper

Type: *Logical* ***Heat:*** *Warm* ***Impact:*** *Medium* ***Speed:*** *Fast*

A parallel logical dumper for MySQL developed and maintained by ex-MySQL employees.

Pros

- It's fast! Multithreded
- Human readable output
- Good solution for larger datasets (multi-threaded)
- Native compression
- Dump remote host
- Compatible with drizzle
- Nearly hot if using only innodb tables

Cons

- No official binaries shipped
- Slower restore then physical backups but faster then mysqldump
- Caveats with restoration
- Relies on mysqldump for database objects (routines, events, etc)

Mydumper

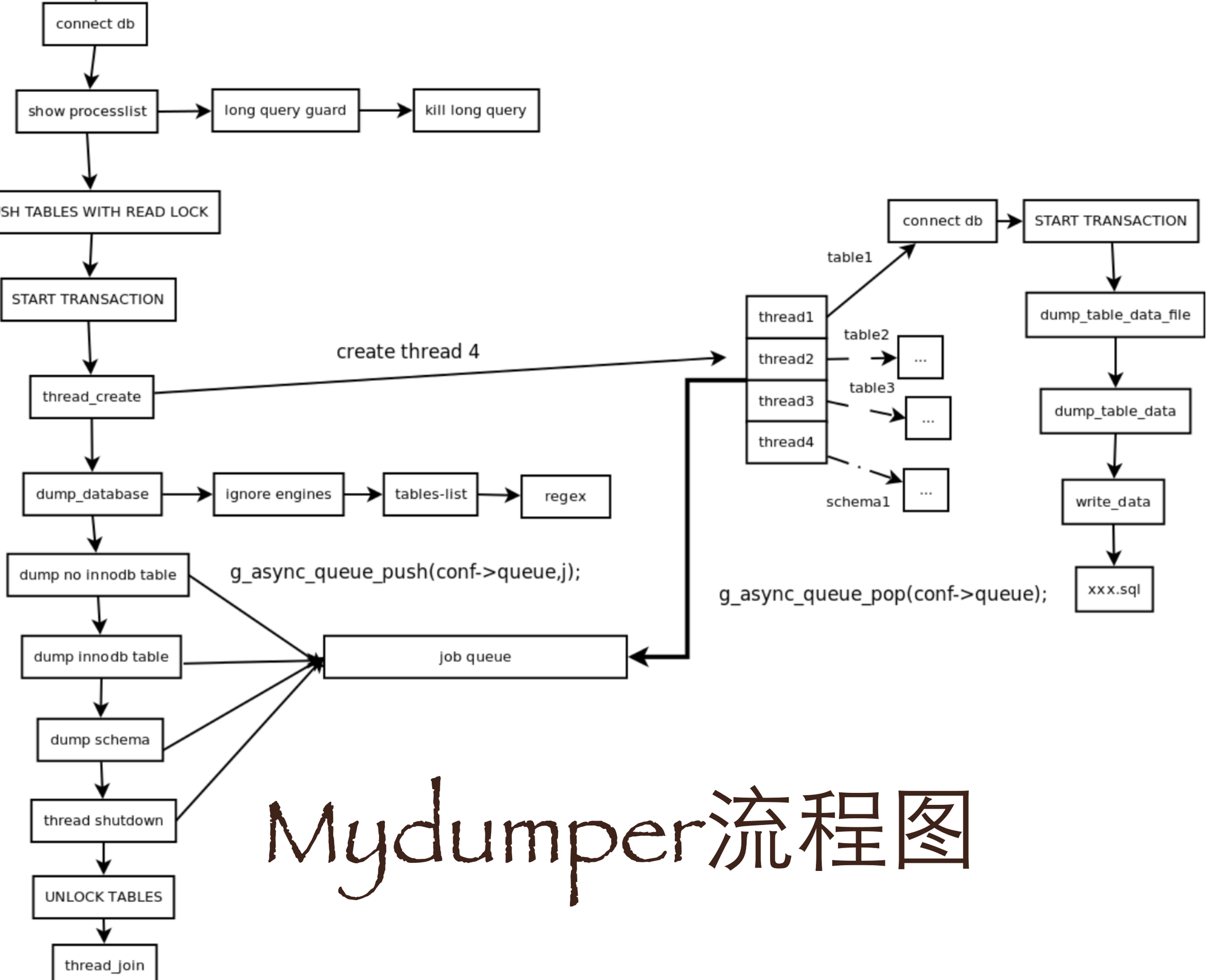
Usage:
[mydumper]
mydumper -B ark_db -o /data/dbbackup/mydumper/ -b --host='ip' --user='secret' --password='xx' --port='3307'

mydumper -x "(ark_db.*)|(test.*)|(mysql.*)" -o /data/dbbackup/mydumper/ --host='ip' --user='secret' --password='xx' --port='3306' --logfile='/data/dbbackup/mydumper/mydumper.log' --threads=6 -v 3 --导出部分库，还有innoDB和Myisam

mydumper -o /data/dbbackup/mydumper/ --host='ip' --user='secret' --password='xx' --port='3306' --logfile='/data/dbbackup/mydumper/mydumper.log' --threads=6 -v 3 --所有库都导出

[myloader]
myloader -o -t 6 -u secret -p xxx -S /tmp/mysql.sock --directory='/data/dbbackup/mydumper' --将/data/dbbackup/mydumper上的所有库都恢复

myloader -o -t 6 -u secret -p xxx -S /tmp/mysql.sock -B ark_db --directory='/data/dbbackup/mydumper' --将/data/dbbackup/mydumper中的ark_db恢复


connect db
show processlist
long query guard
kill long query
FLUSH TABLES WITH READ LOCK
START TRANSACTION
thread_create
create thread 4
dump_database
ignore engines
tables-list
regex
dump no innodb table
dump innodb table
dump schema
thread shutdown
UNLOCK TABLES
thread_join
g_async_queue_push(conf->queue,j);
job queue
thread1
thread2
thread3
thread4
table1
table2
table3
schema1
...
connect db
START TRANSACTION
dump_table_data_file
dump_table_data
write_data
xxx.sql
g_async_queue_pop(conf->queue);
Mydumper流程图

Mydumper

OUtLook

1） start transaction 1，先进行flush tables with read lock

2）开启多个线程，并且将表分为3个队列，innoDB，non-innoDB，结构

3）start transaction 2，先处理non-innoDB表，处理完后unlock tables

4）start transaction 3，设置RR级别，开始分多线程dump表和表结构

总结：由此可见，mydumper对于DDL还是无能为力

Snapshot Backups

***Type**: Physical* ***Heat**: Warm* ***Impact**: Varies* ***Speed**: Fast*

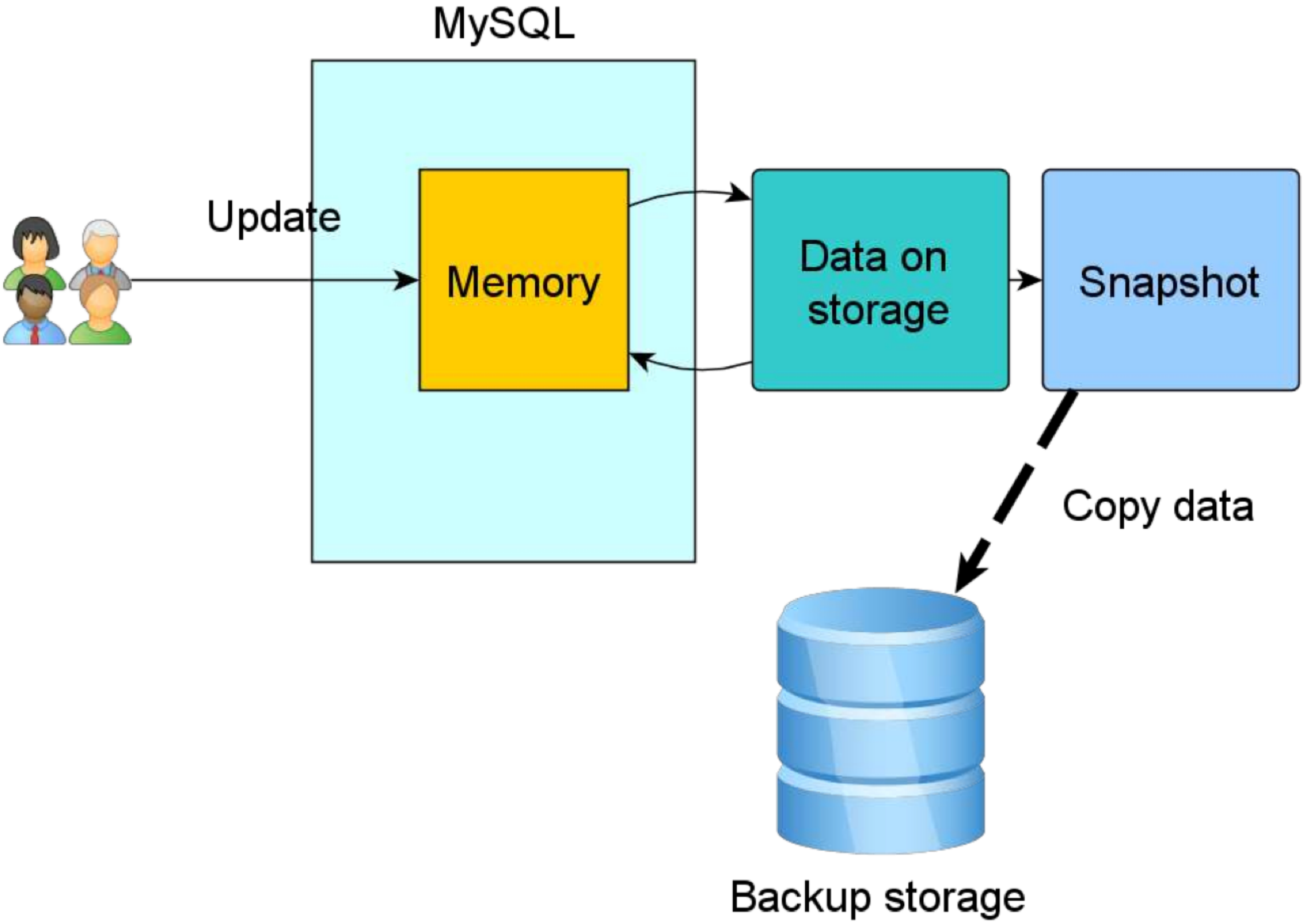
Techniques as seen in **mylvmbackup** show that snapshots can be used to make the backup online using copy on write technologies. Using a snapshot capable filesystem such as **LVM**, **ZFS, VSS** or **SAN** storage with the same ability can afford you a storage checkpoint where changes can be simply rolled back if issues arise

Pros

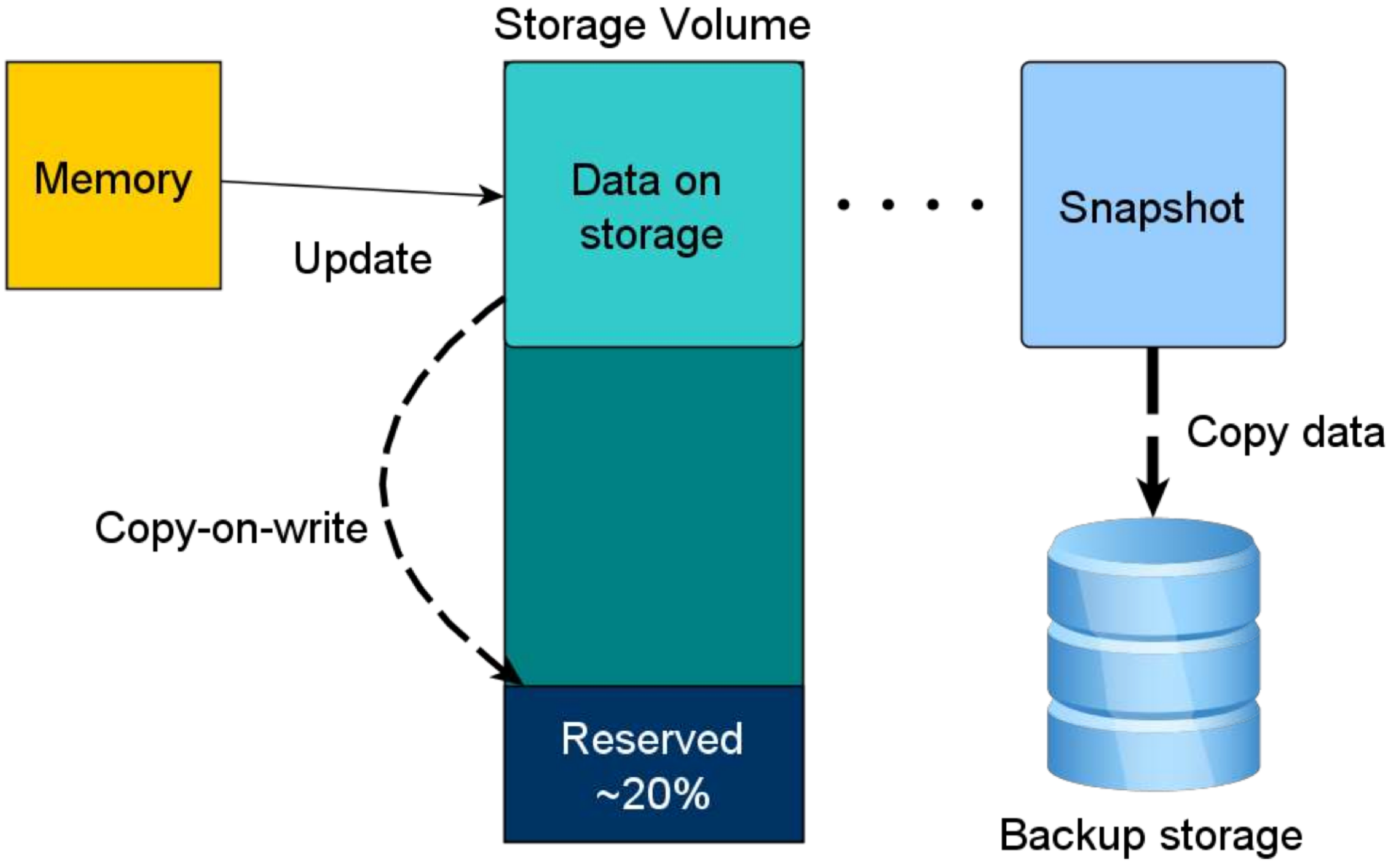
- Fast
- Warm backups
- Familiar commands as with storage tools

Cons

- Crash recovery needed for InnoDB to apply redo log
- The 'copy on write' overhead could impact performance from the point the snapshot is created until it's destroyed.


MySQL
Update
Memory
Data on storage
Snapshot
Copy data
Backup storage

LVM snapshot



Percona Xtrabackup 2.x

***Type**: Physical* ***Heat**: Hot* ***Impact**: Low* ***Speed**: Fast*

One of the strongest and widely used solutions for consistently backing up MySQL files focused on Xtradb/Innodb but also support for non-transactional tables too. Xtrabackup makes use of the XtraDB/ InnoDB crash recovery cycle to apply the redo logs to the data when preparing for a restore. This prepare phase can happen at the end of the backup or as part of the recovery phase.

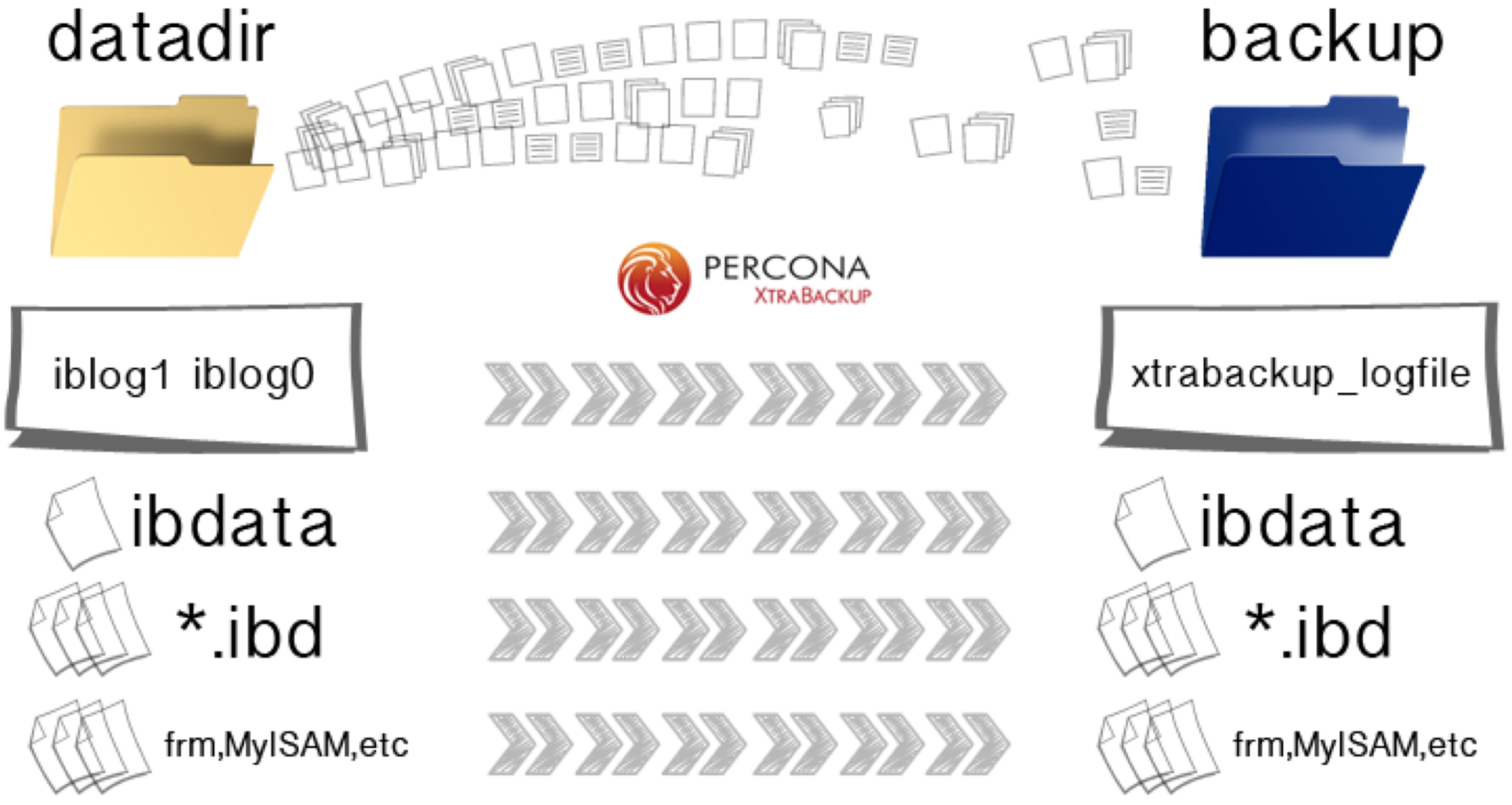
Pros

- Free, GPL Licensed
- Hot & Physical
- Throttles to keep load low
- Native compression (qpress)
- Export tables
- Parallel backup
- Wide OS compatibility
- Consistent backup of MyISAM
- Great documentation and recipes
- Compatible with XtraDB Cluster

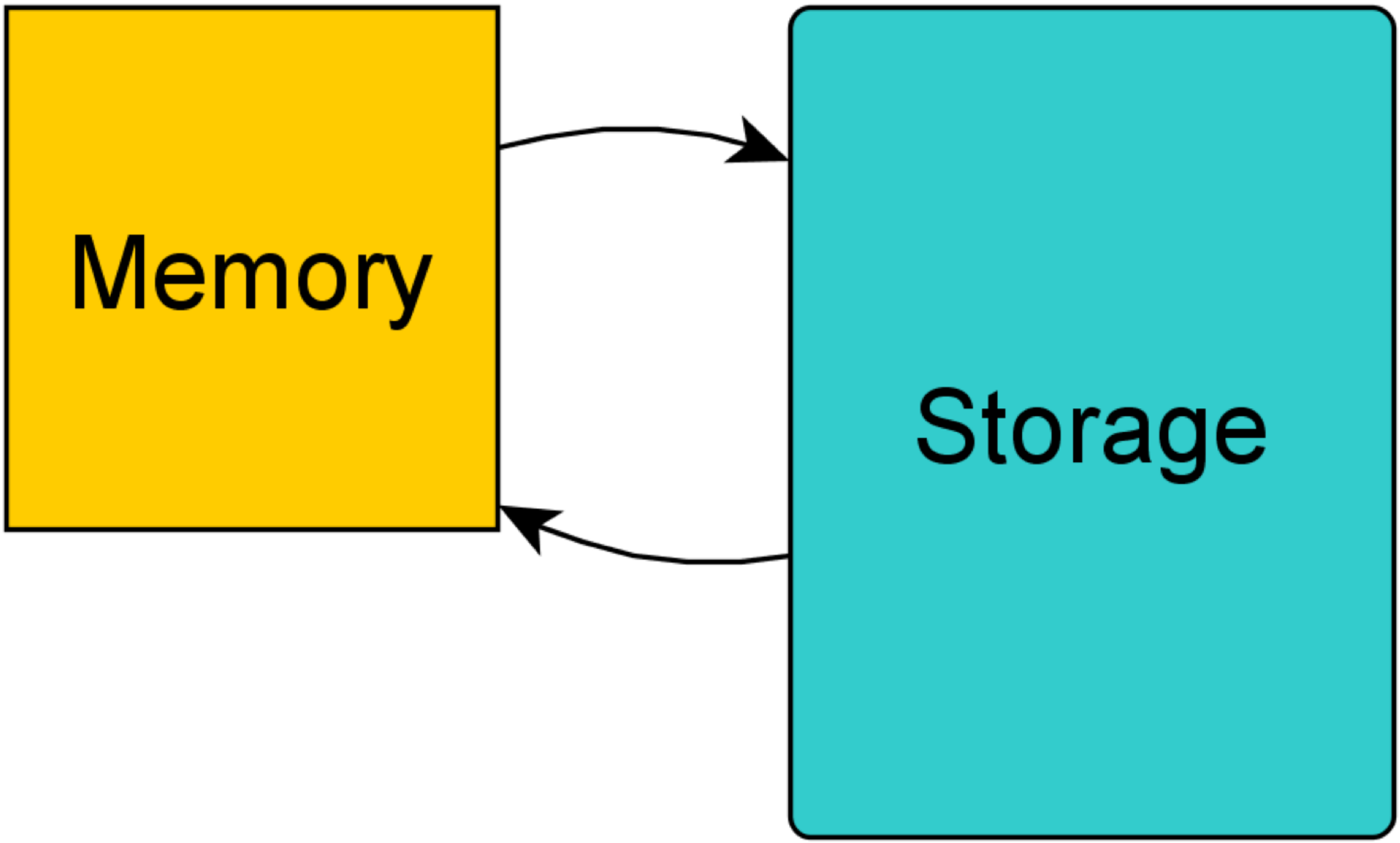
Cons

- Windows version in Alpha
- Multiple stage restore
- Cannot prepare compressed (on the fly) backups
- Qpress compression < gzip/pigz

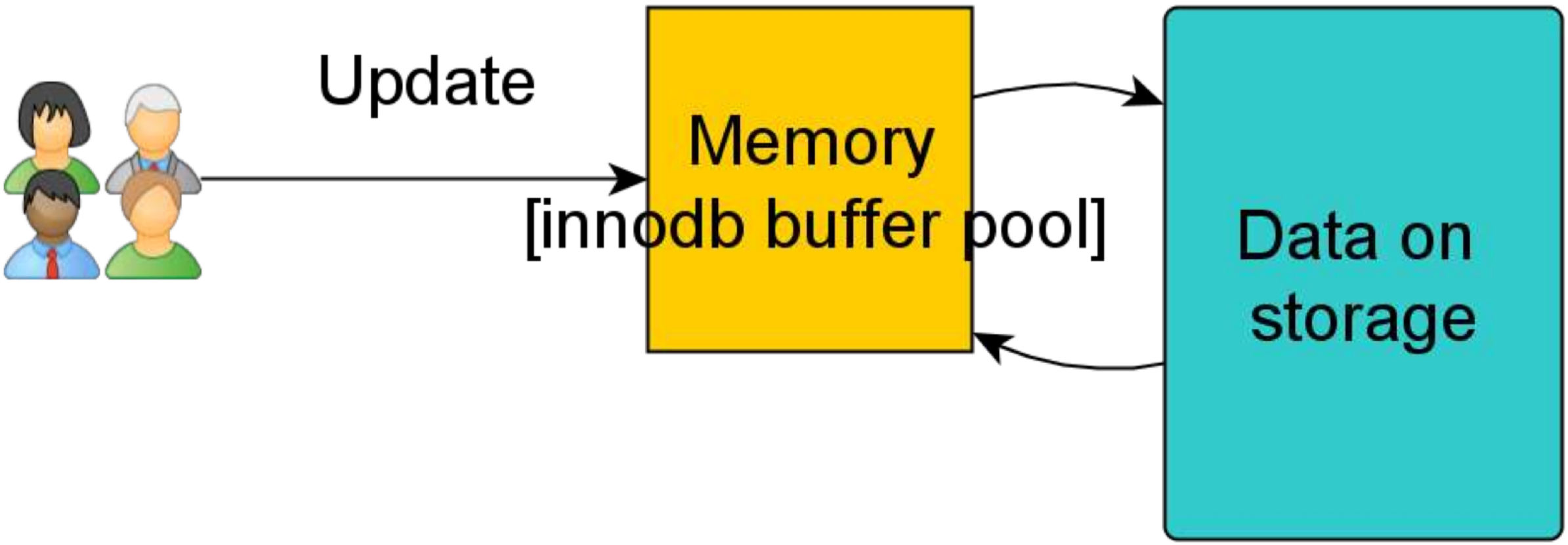
Percona Xtrabackup 2.x



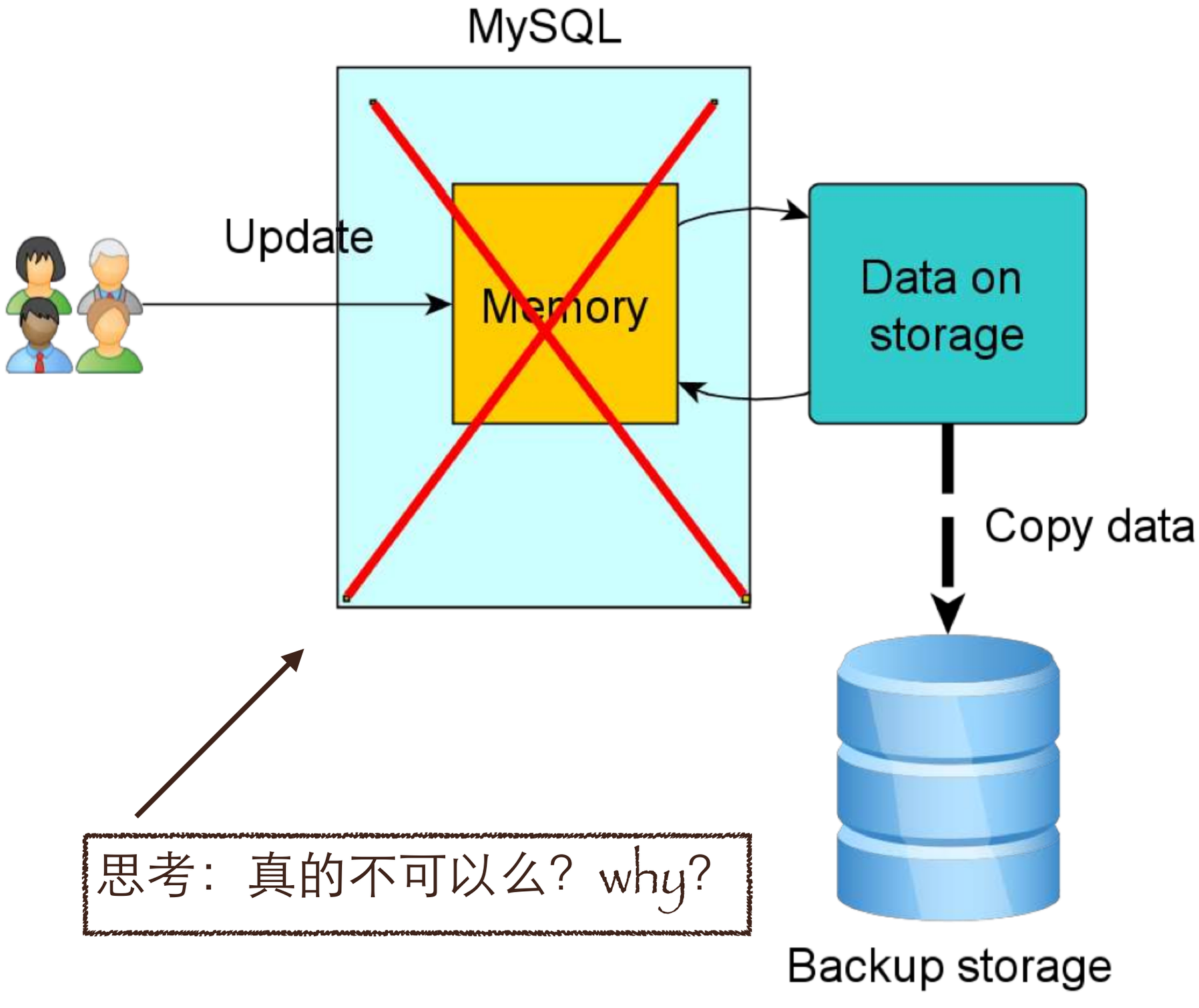
Database



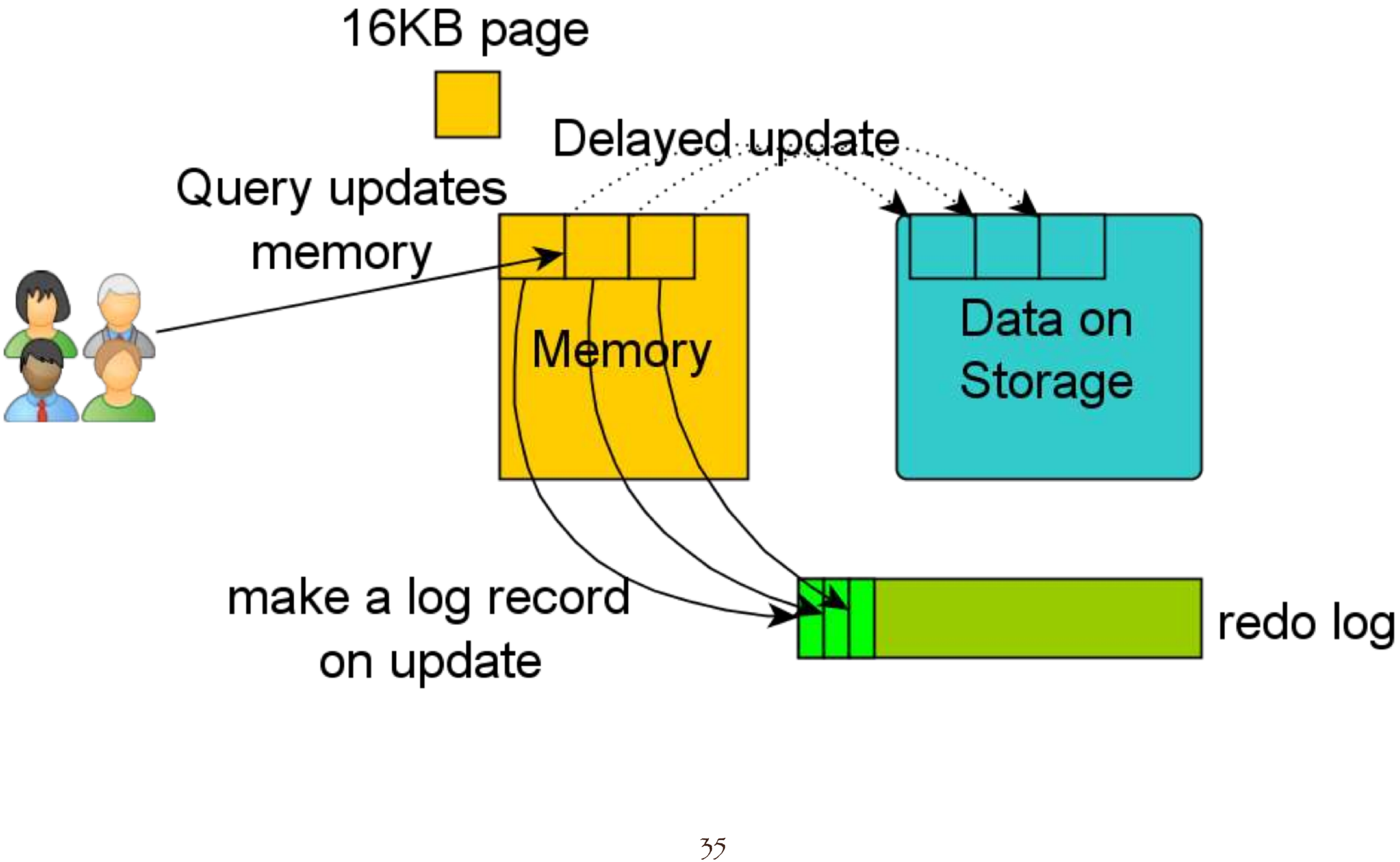
InnoDB updates



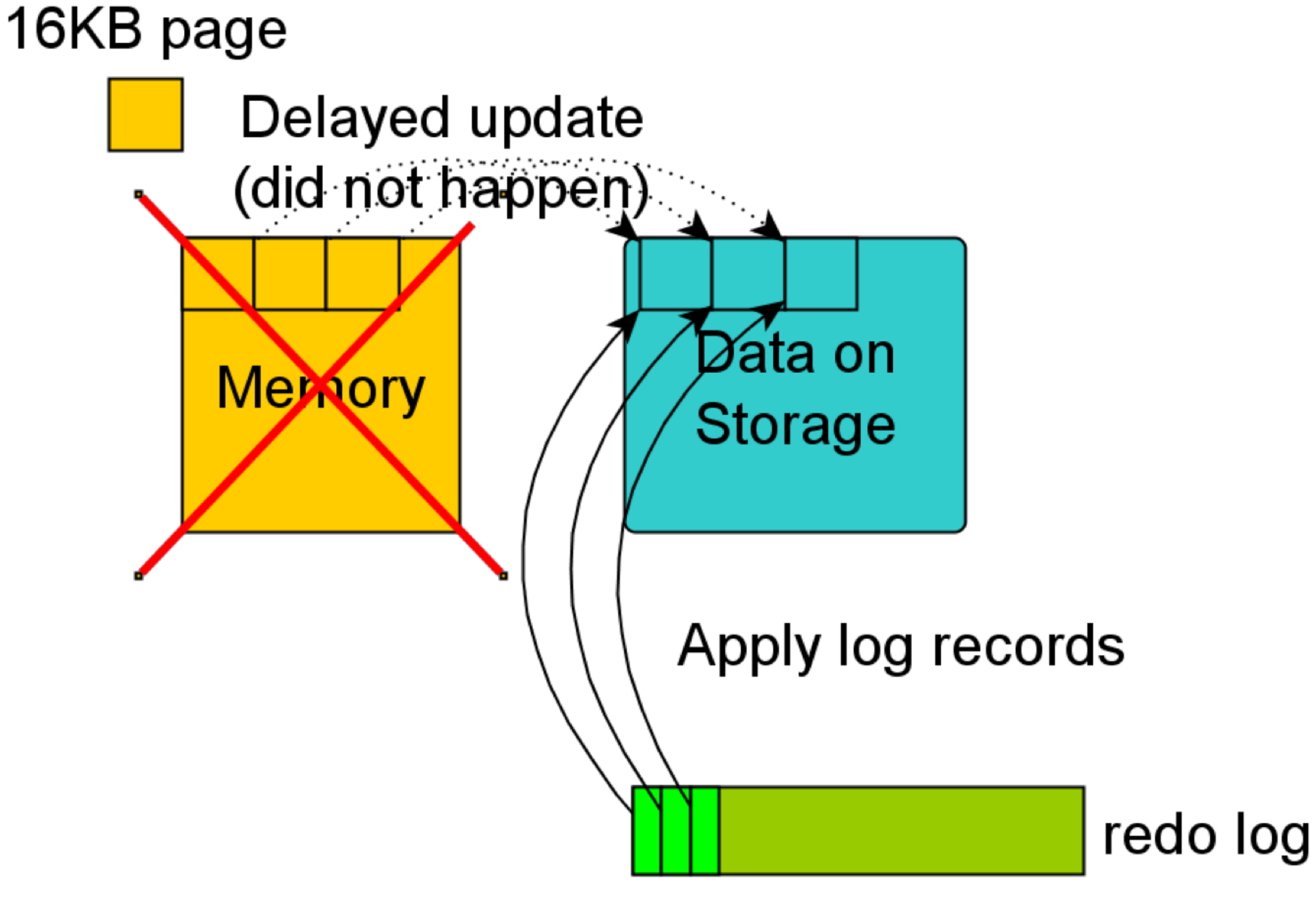
Cold copy

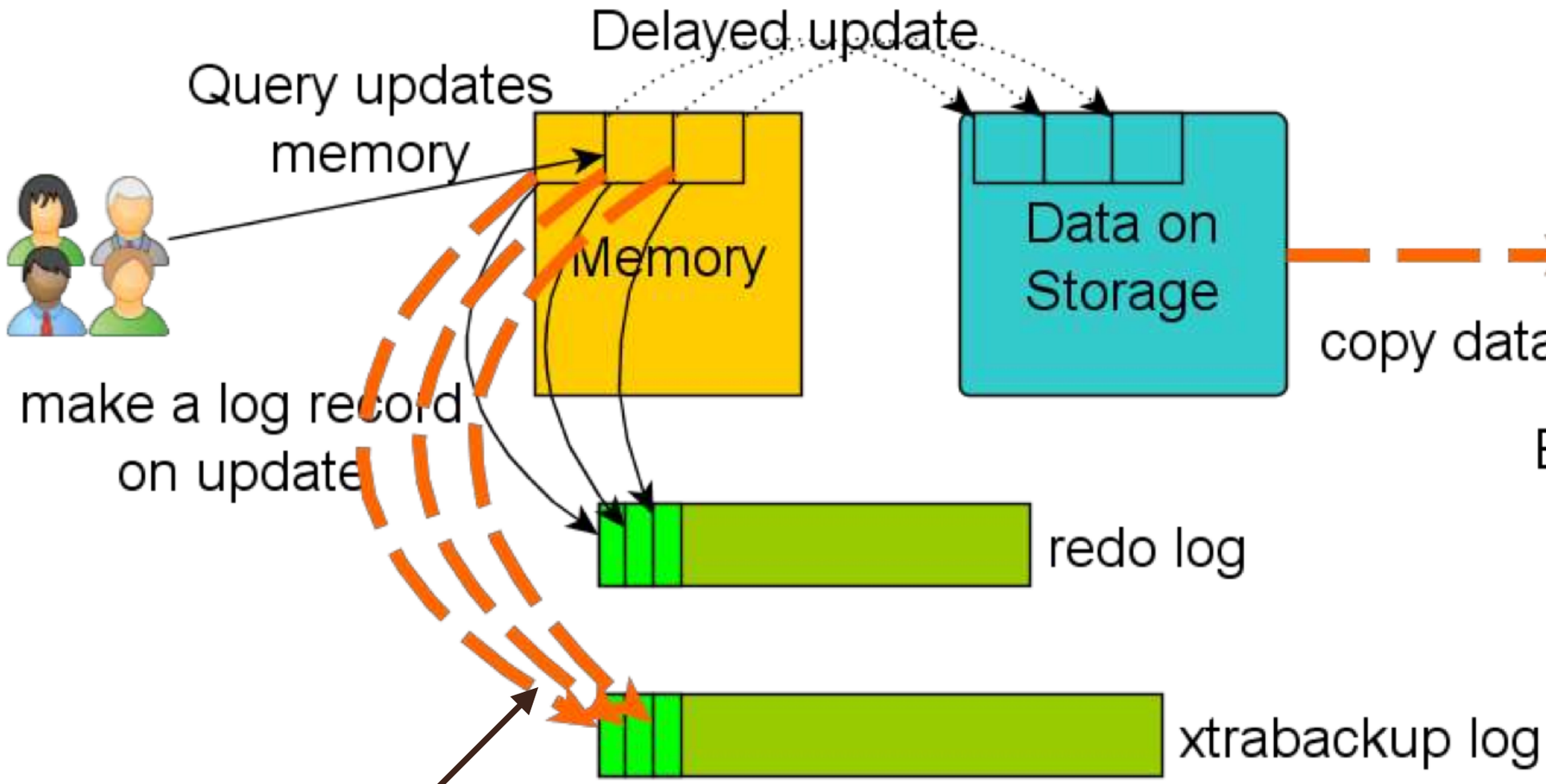

MySQL
Update
Memory
Data on storage
Copy data
Backup storage
思考：真的不可以么？why?

InnoDB internals



InnoDB recovery

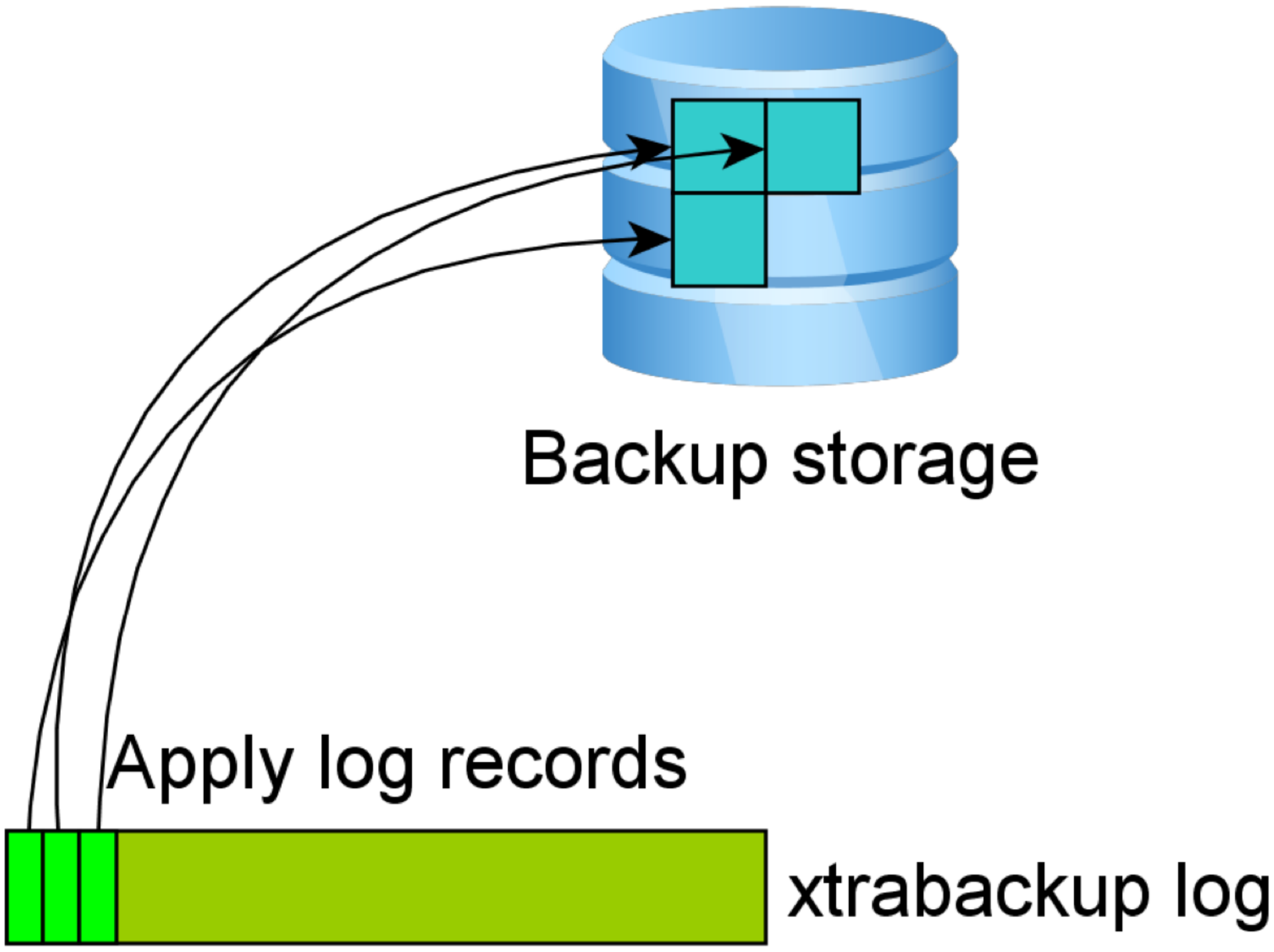


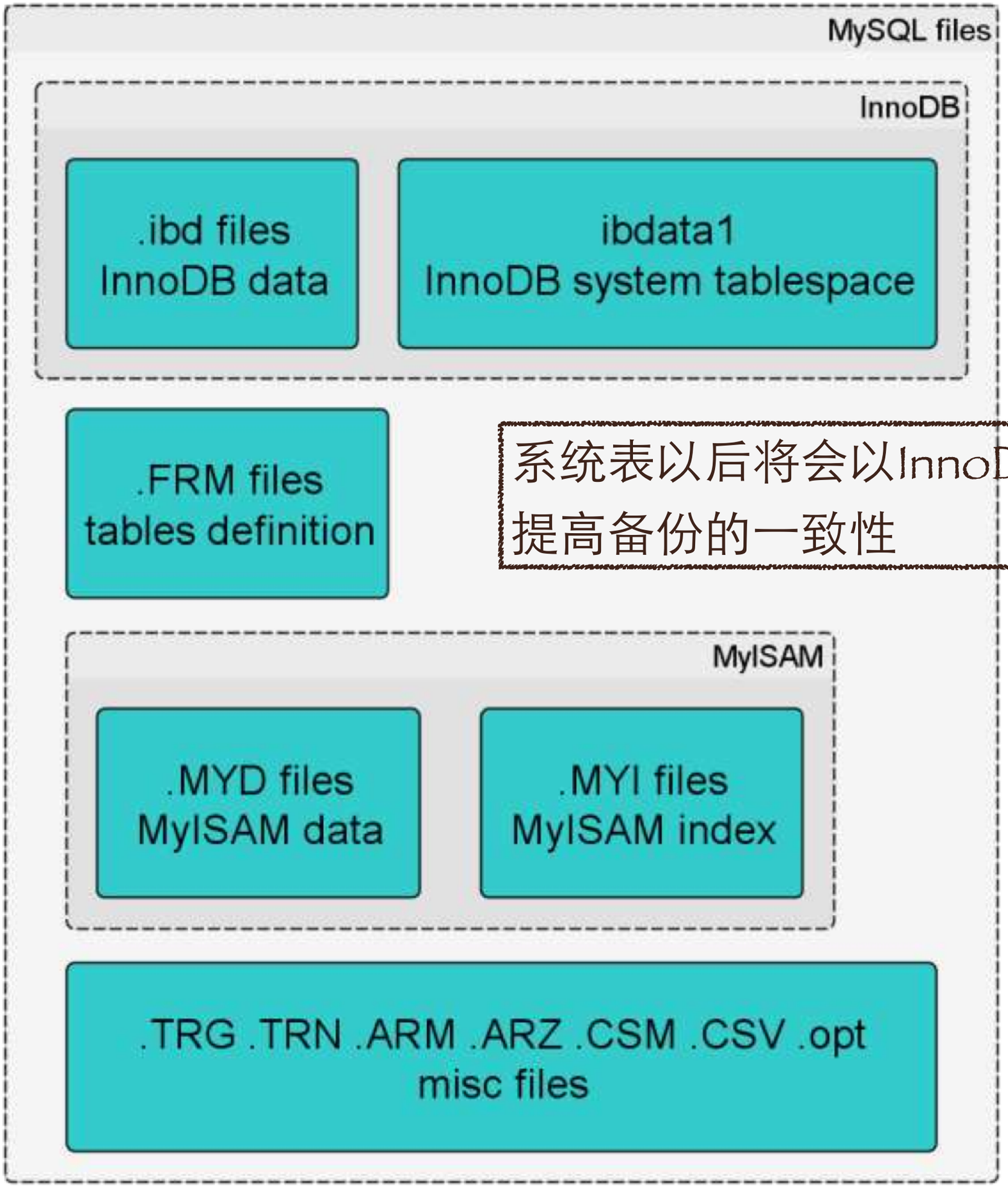


如果xtrabackup log-Monitor来不及拷贝日志怎么办?

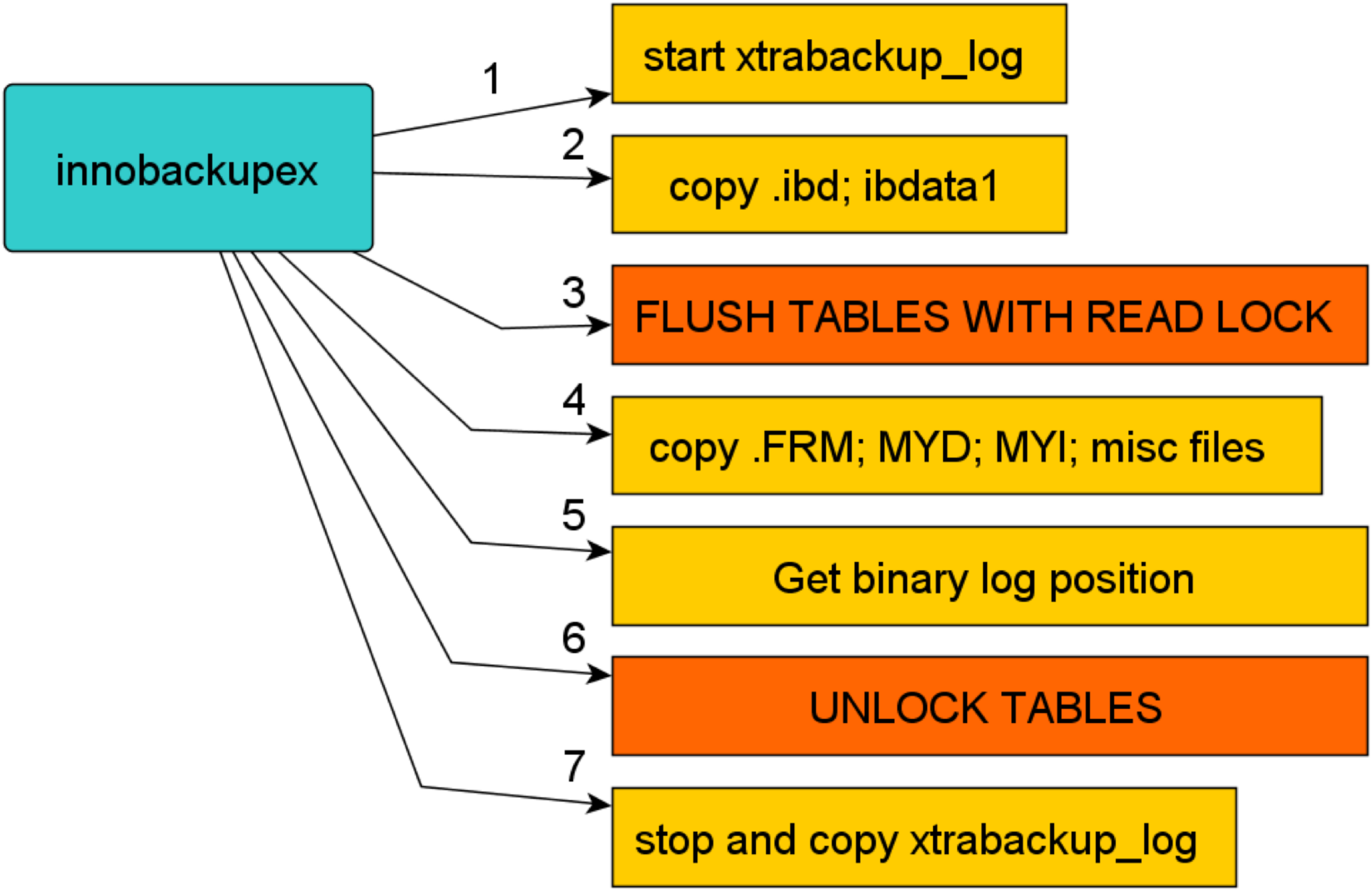
官方解释：1）调整redo 大小 2）在业务低峰期备份

Backup 2nd stage

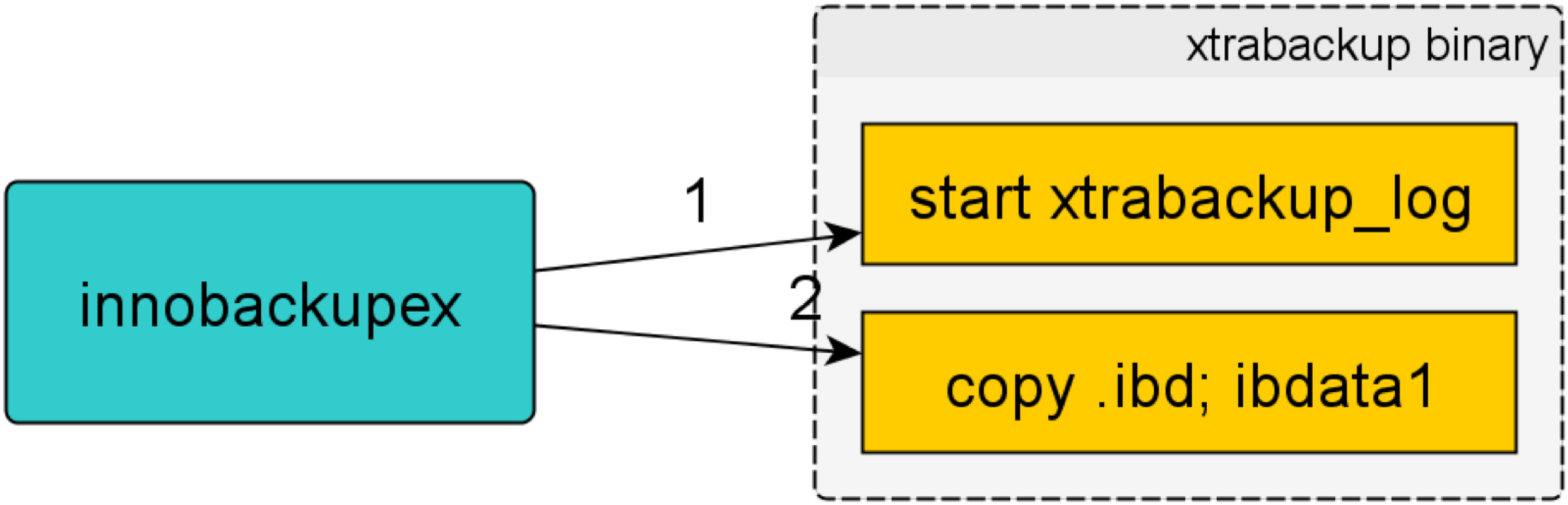




系统表以后将会以InnoDB格式存储，提高备份的一致性

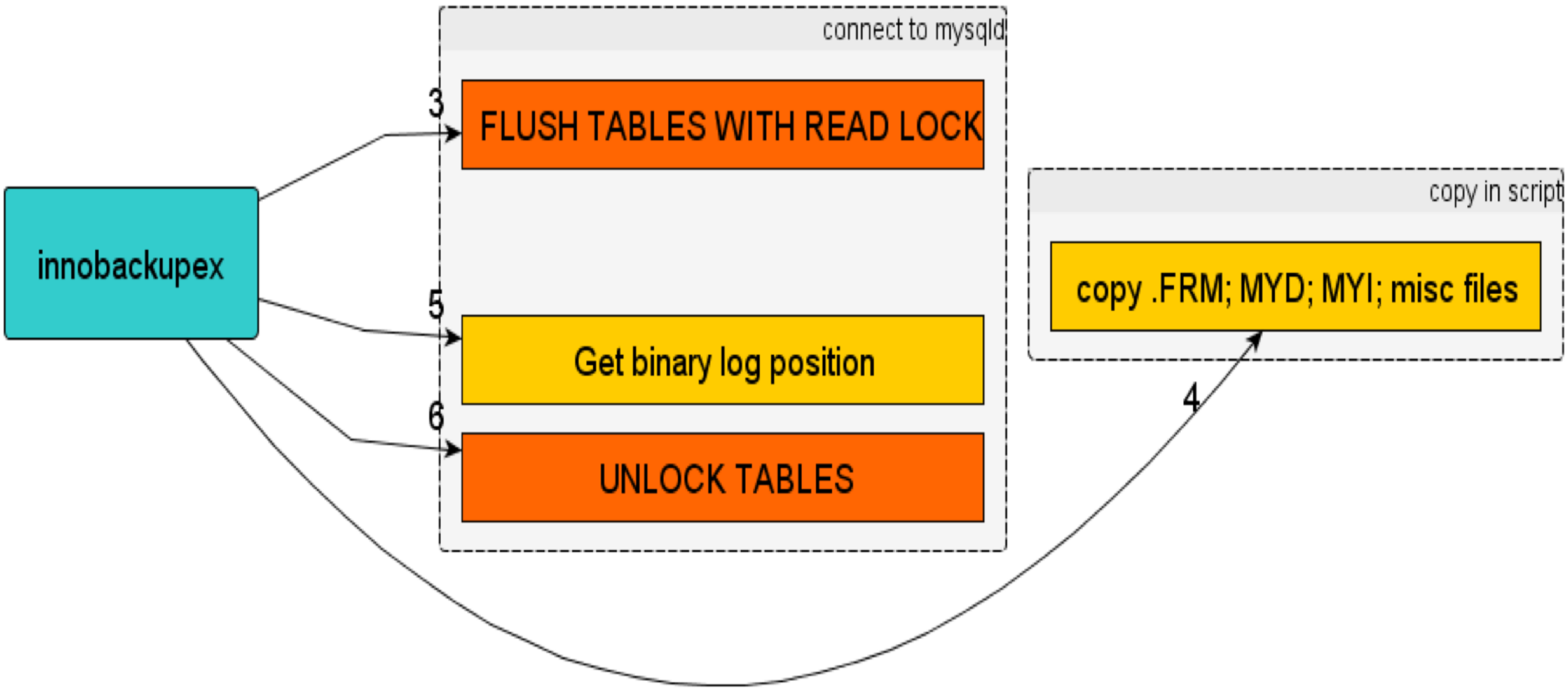

innobackupex
1
start xtrabackup_log
2
copy .ibd; ibdata1
3
FLUSH TABLES WITH READ LOCK
4
copy .FRM; MYD; MYI; misc files
5
Get binary log position
6
UNLOCK TABLES
7
stop and copy xtrabackup_log

1）记录最新的LSN，并以此作为恢复的起始点

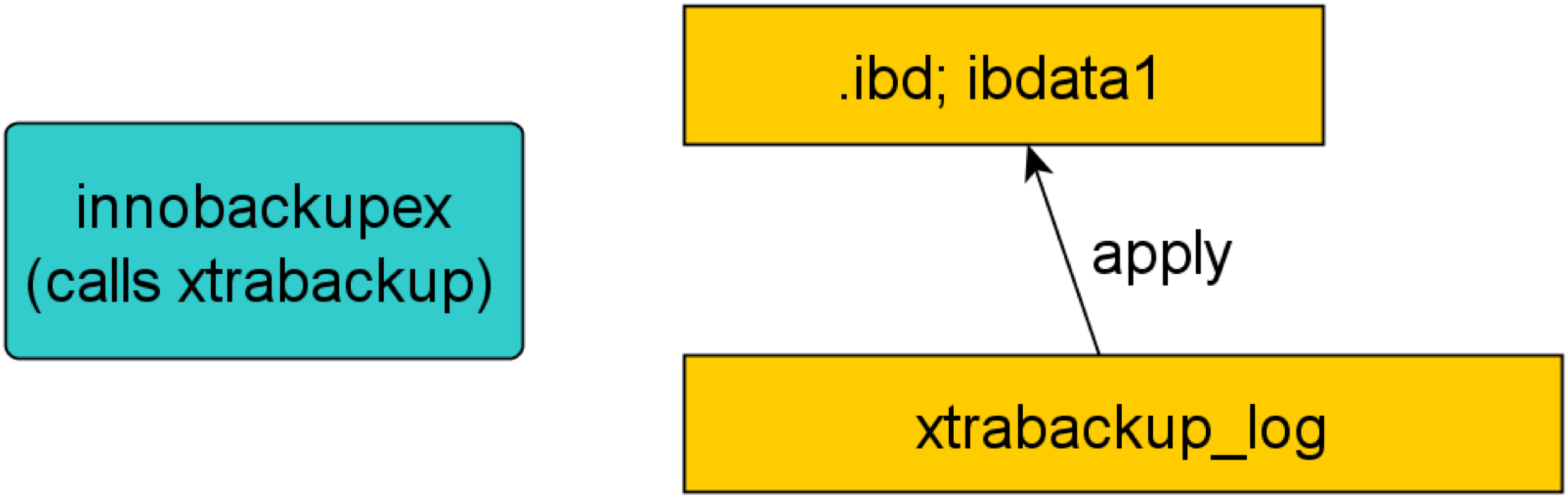


2）这时候拷贝的数据是不一致的，不过通过redo，最终是一致的

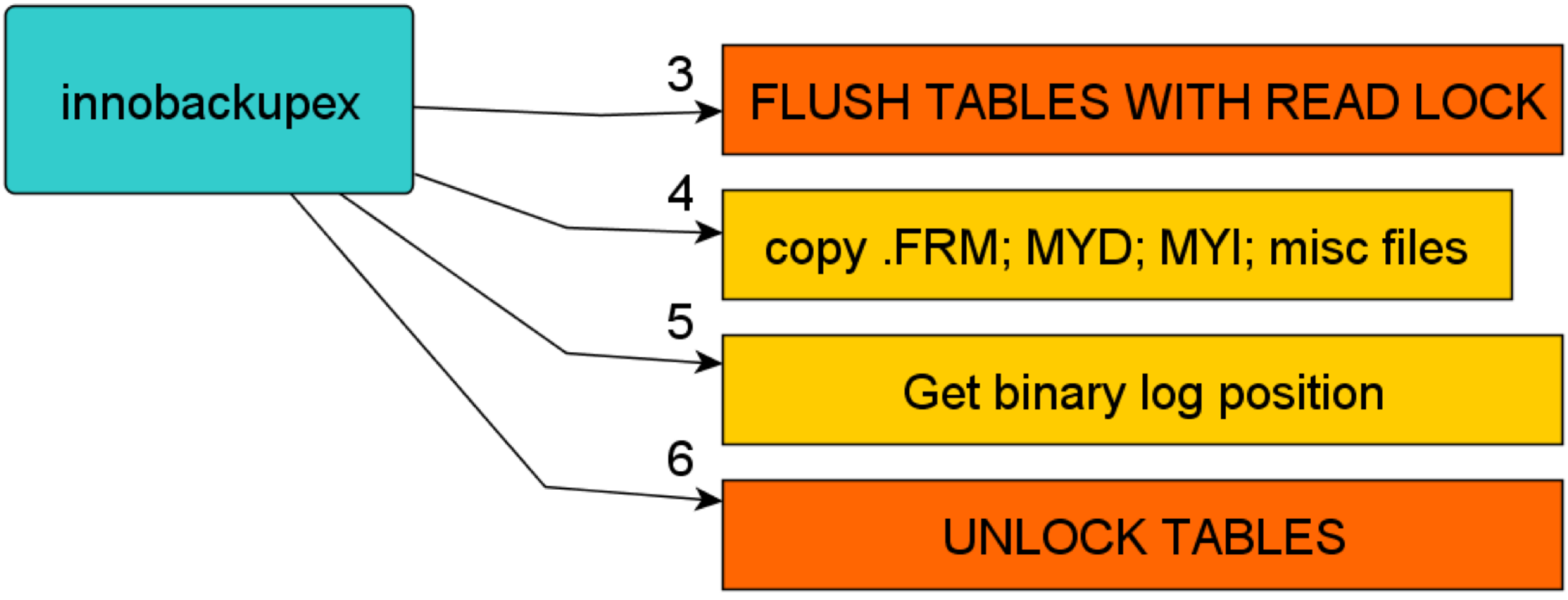
3）注意：这里没有拷贝.frm文件。思考，这是为什么？DDL。。。


connect to mysqld
3
FLUSH TABLES WITH READ LOCK
innobackupex
5
Get binary log position
6
UNLOCK TABLES
copy in script
copy .FRM; MYD; MYI; misc files
4

Stage 2

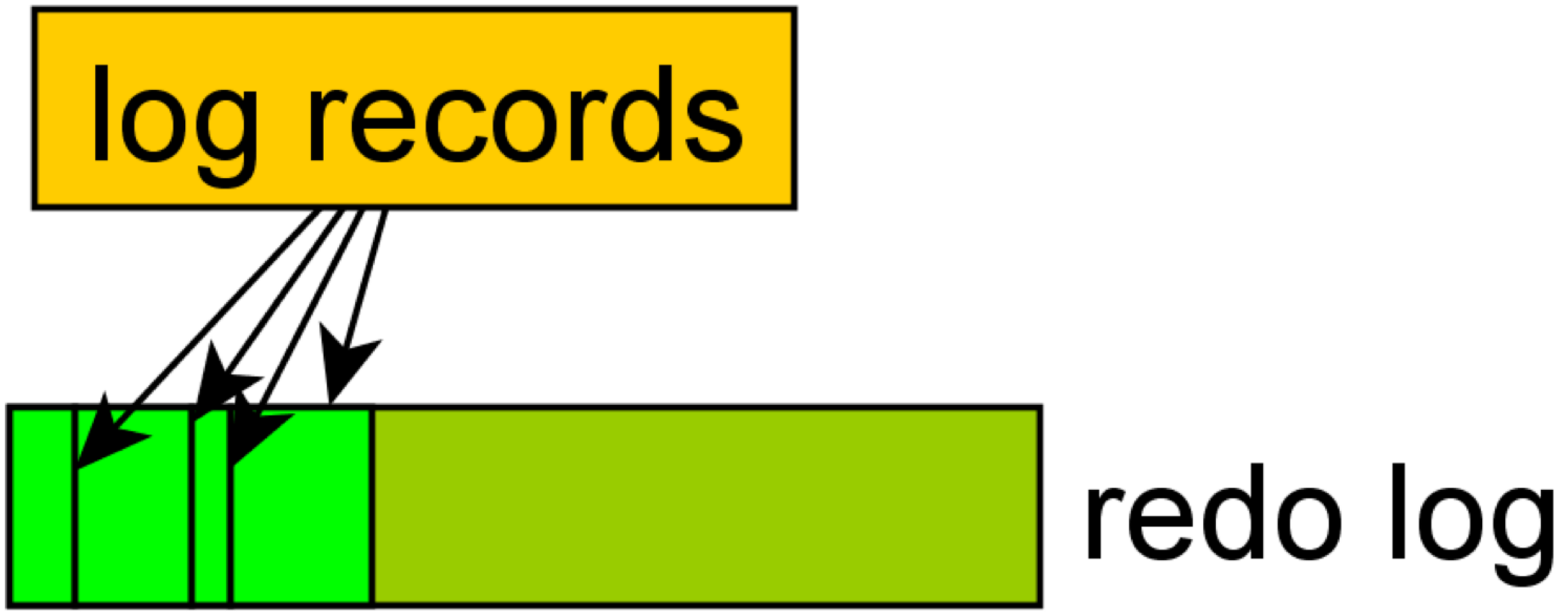

innobackupex
(calls xtrabackup)
.ibd; ibdata1
apply
xtrabackup_log

FLUSH TABLES WITH READ LOCK



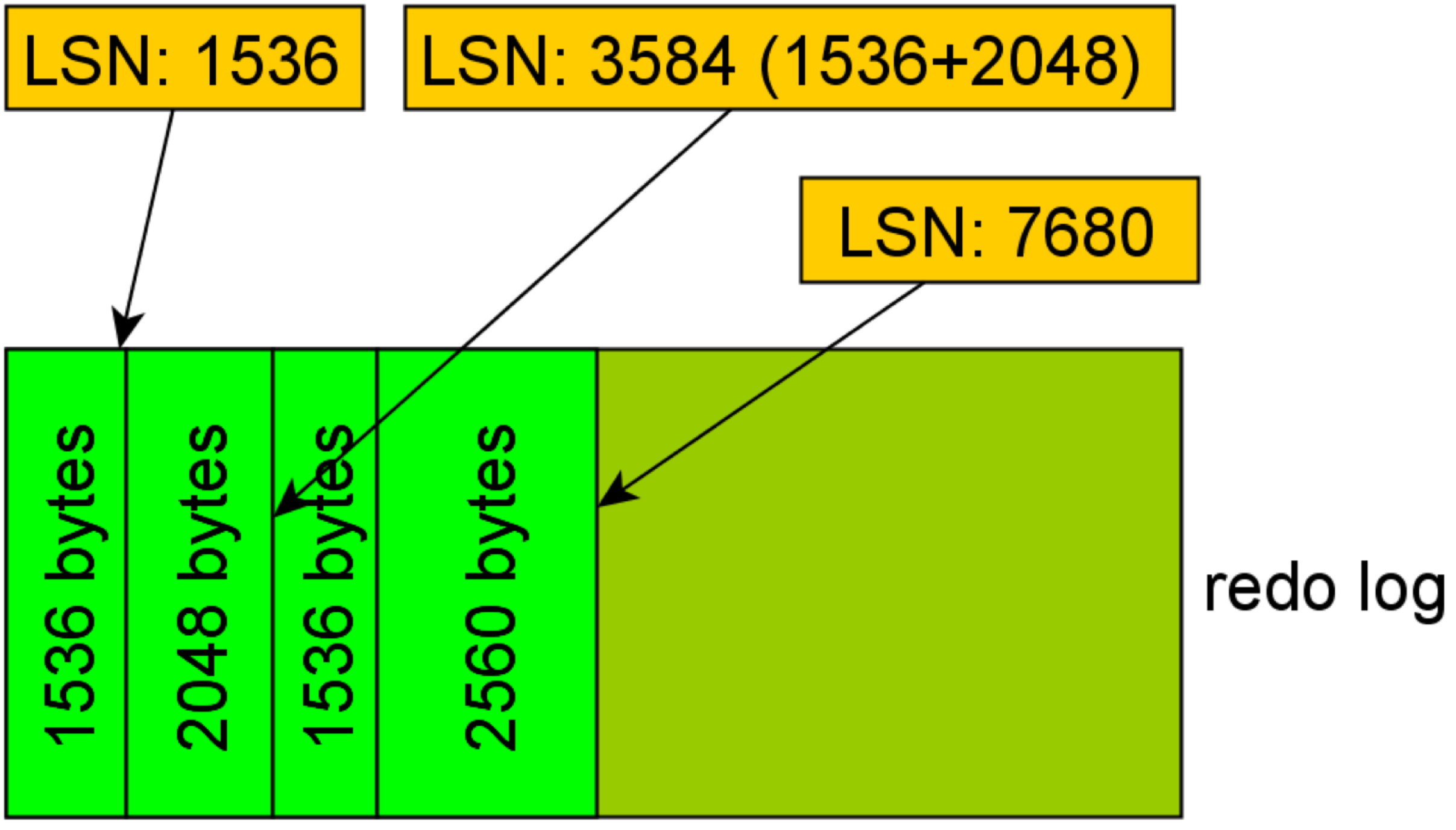
Why? Consider:

- Copy `table1.frm`
- Copy `table2.frm`
- Copy `table3.frm`
- Copy `table4.frm`
  - `..... Meantine ALTER TABLE table1 started ……..`
- Copy `table5.frm`
- Copy `table6.frm`
- Copy `table7.frm`



重点知识：redo不会记录frm的变更。

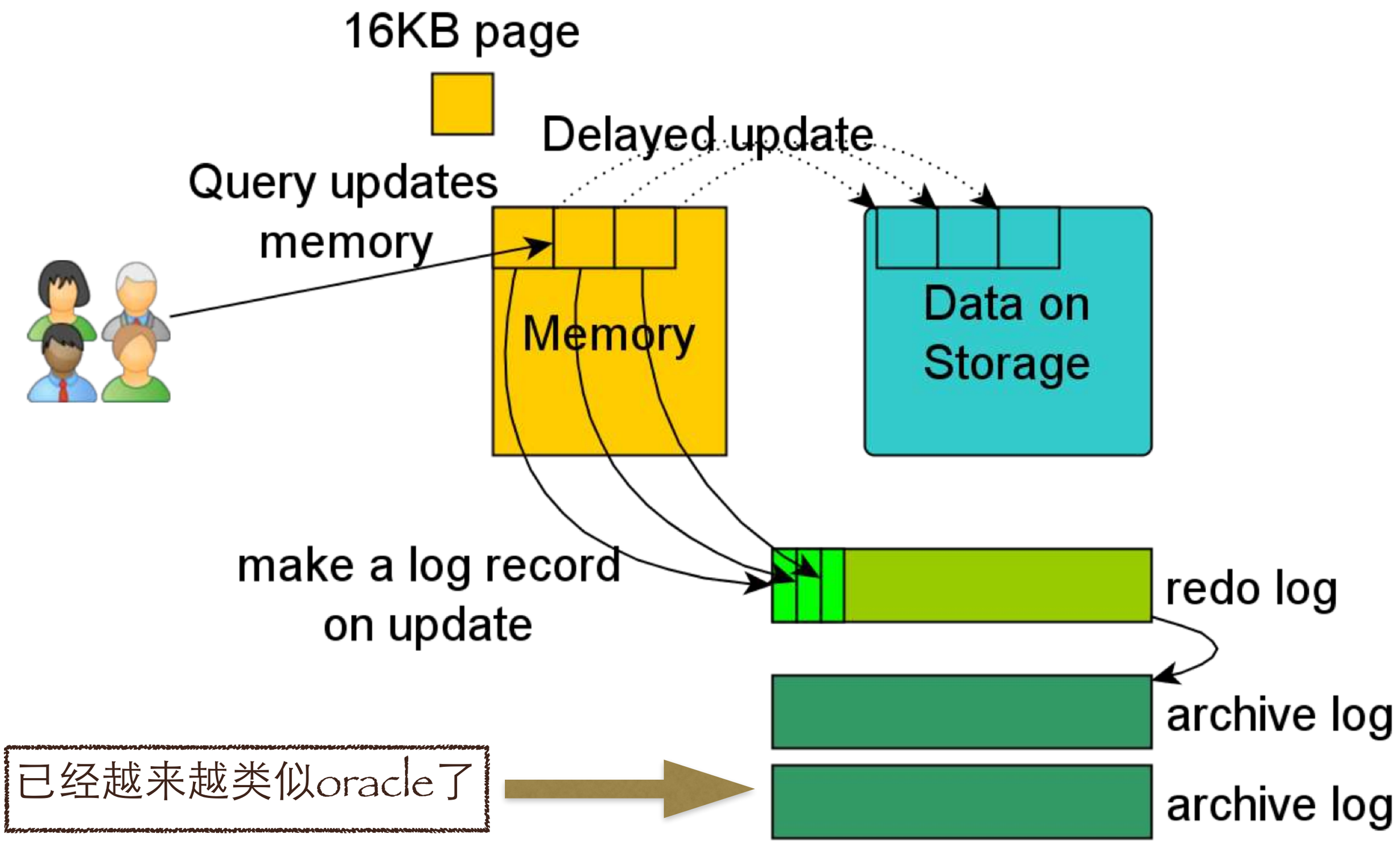
只会记录datafile的变更，并且通过比对datafile和redo的lsn来恢复数据


LSN: 1536
LSN: 3584 (1536+2048)
LSN: 7680
1536 bytes
2048 bytes
1536 bytes
2560 bytes
redo log

With FTWRL

- FLUSH TABLES WITH READ LOCK
- Copy `table1.frm`
- Copy `table2.frm`
- Copy `table3.frm`
- Copy `table4.frm`
  - `.....` `ALTER TABLE table1 --->` **`LOCKED`** `.....`
- Copy `table5.frm`
- Copy `table6.frm`
- Copy `table7.frm`
- `UNLOCK TABLES`

Archive logs idea



已经越来越类似oracle了

recover

Disaster Recovery Plan

Time or data?

RTO & RPO

Disaster Recovery Plan

Recovery Time Objective

"the duration of time and service level within which a business process must be restored after a disaster or disruption"

http://en.wikipedia.org/wiki/Recovery_Time_Objective

Disaster Recovery Plan

Recovery Time Objective

Includes

- Time allowed to troubleshoot (without recovery/fix)
- The recovery time itself
- Time for communication to stakeholders

Disaster Recovery Plan

Recovery Point Objective

"the maximum tolerable period in which data might be lost from an IT service due to a major incident"

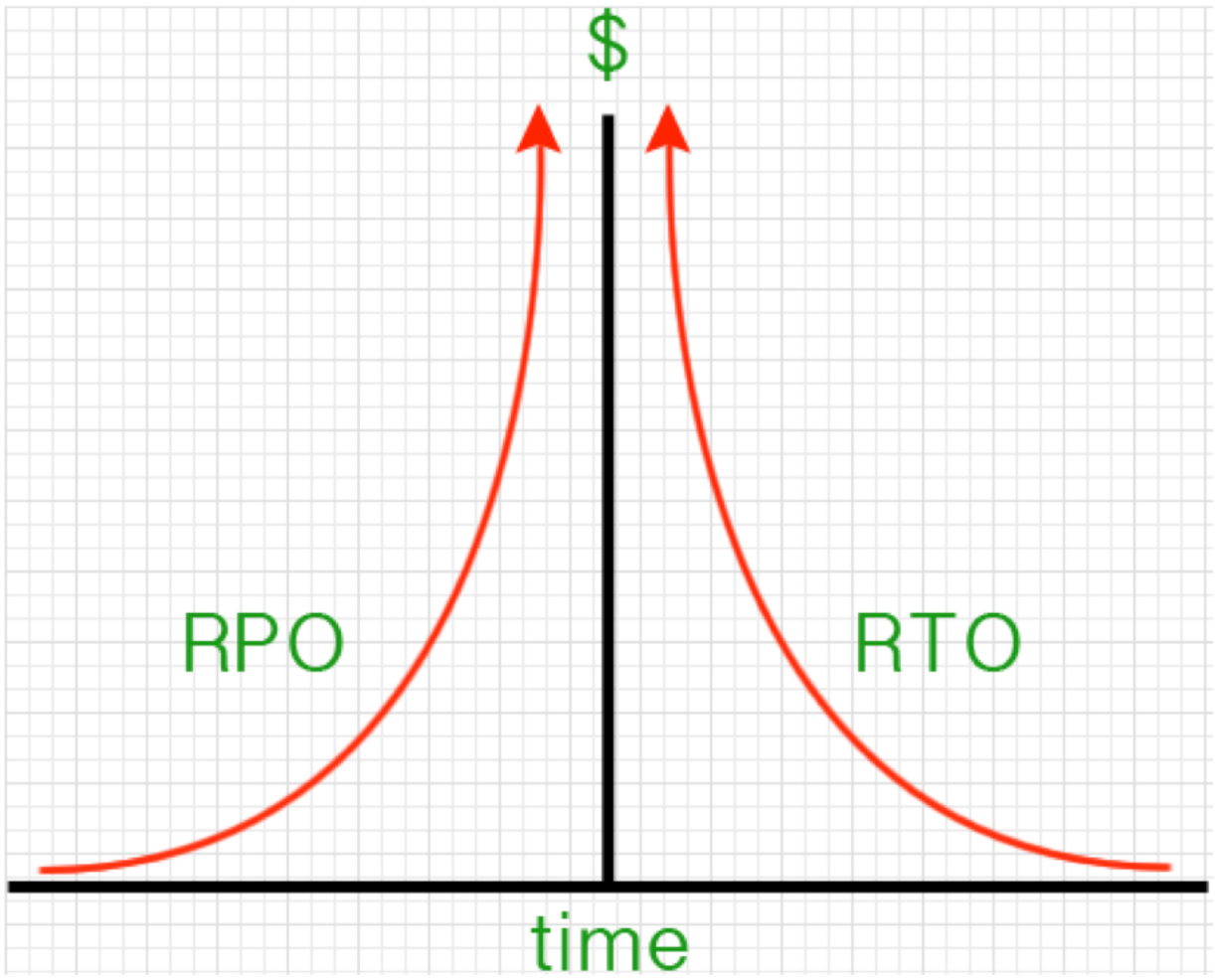
http://en.wikipedia.org/wiki/Recovery_Point_Objective

Disaster Recovery Plan

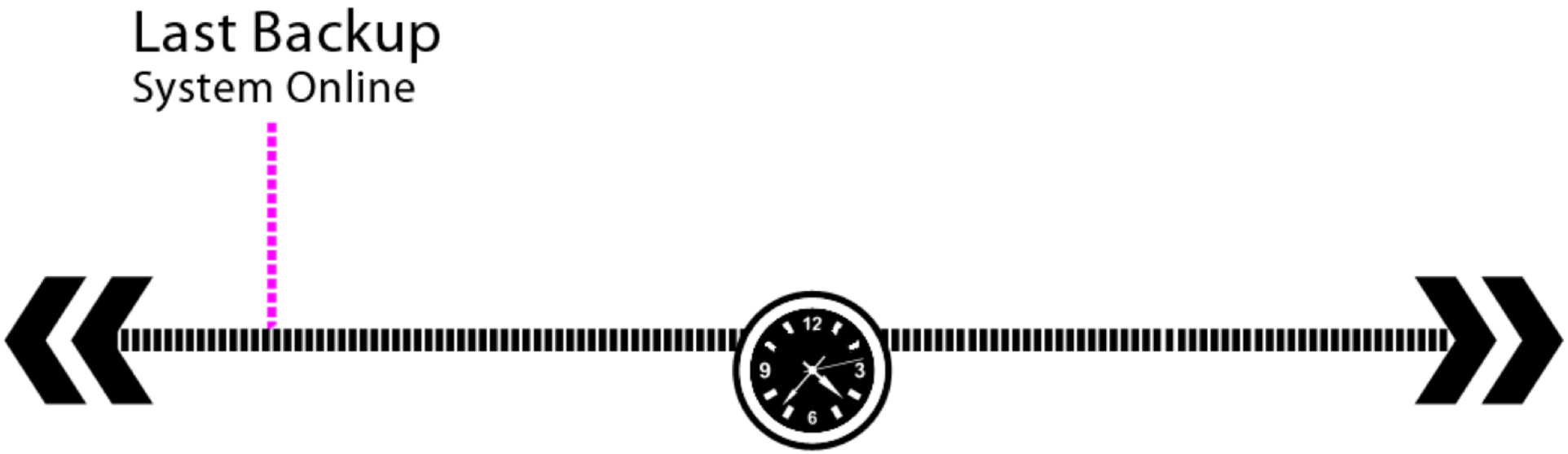
Can you afford…

- Downtime?
- Data loss?

Generally it costs too much $ to say no to both

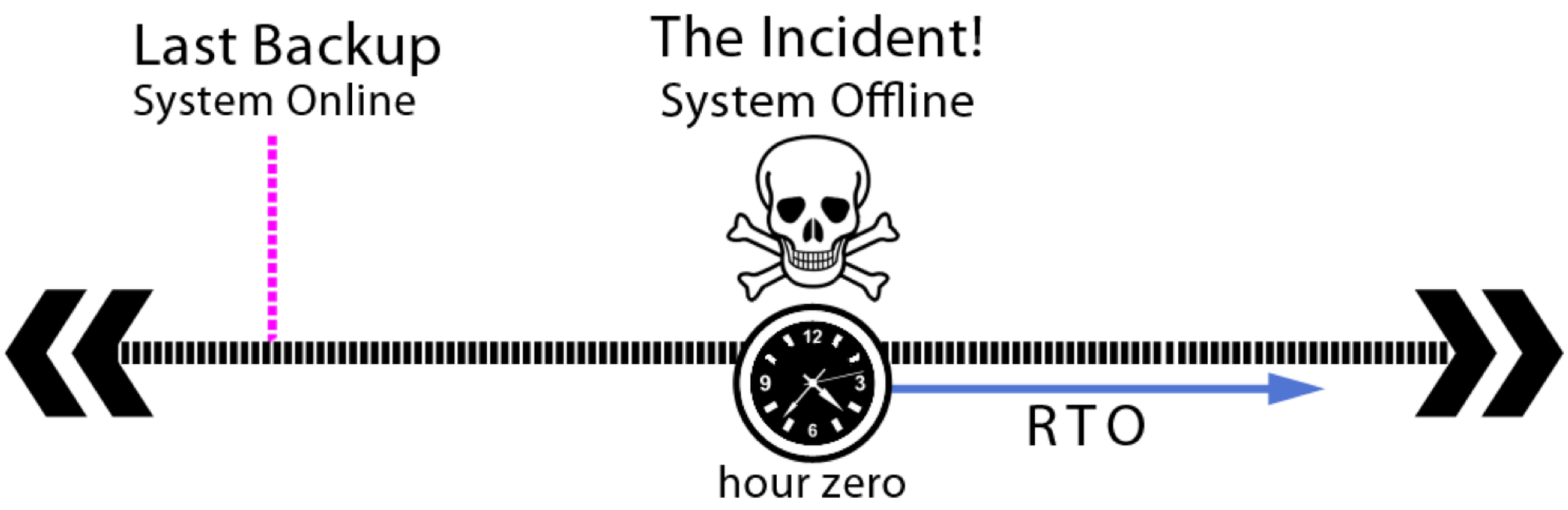


Disaster Recovery Plan

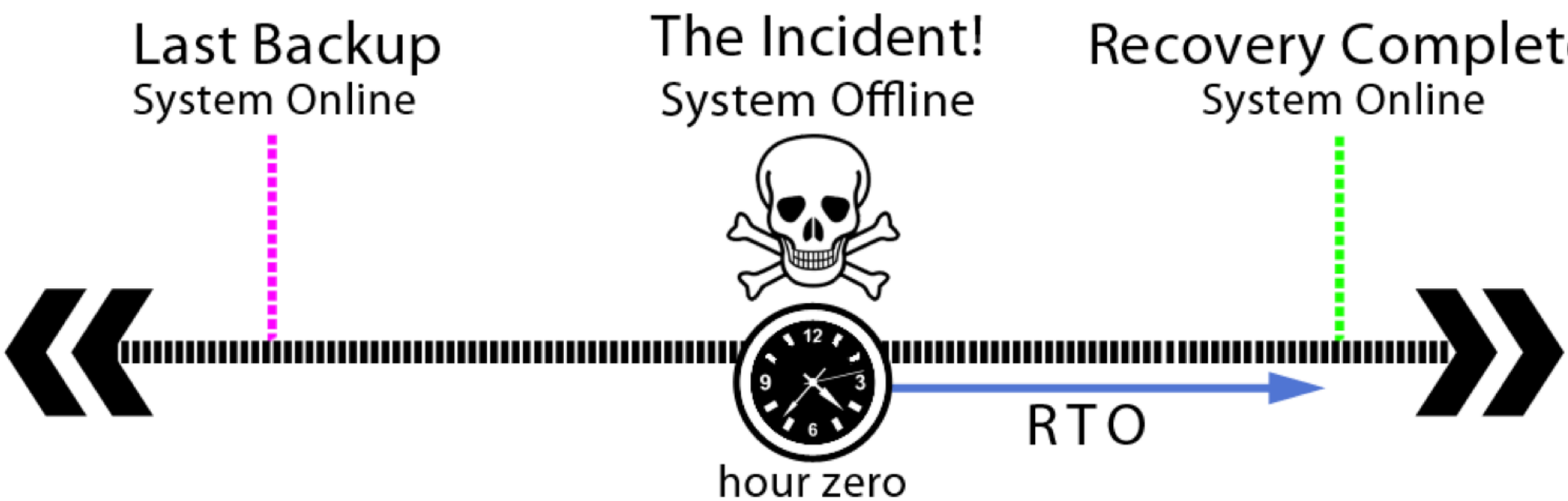


Disaster Recovery Plan

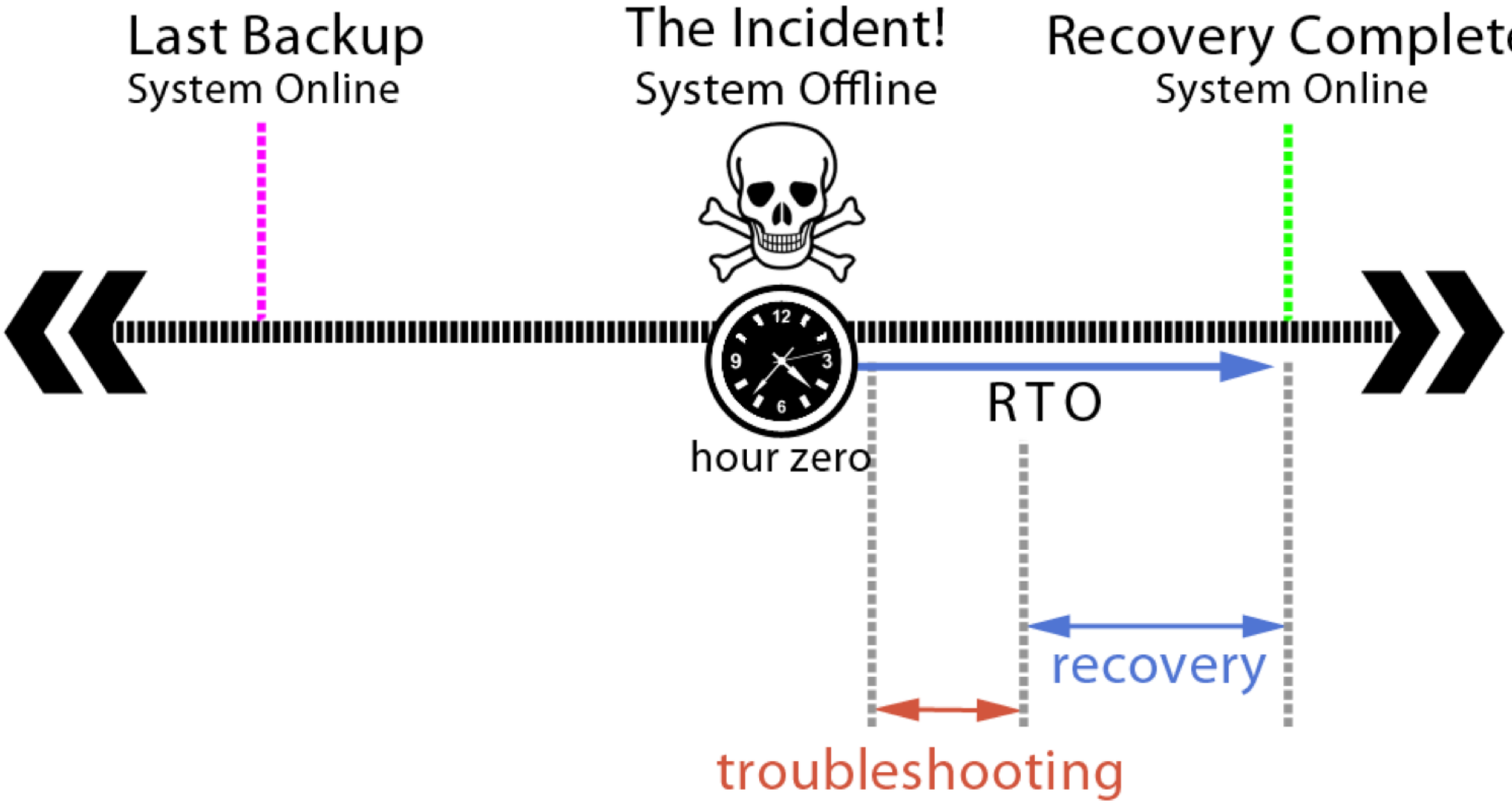
Disaster Recovery Plan



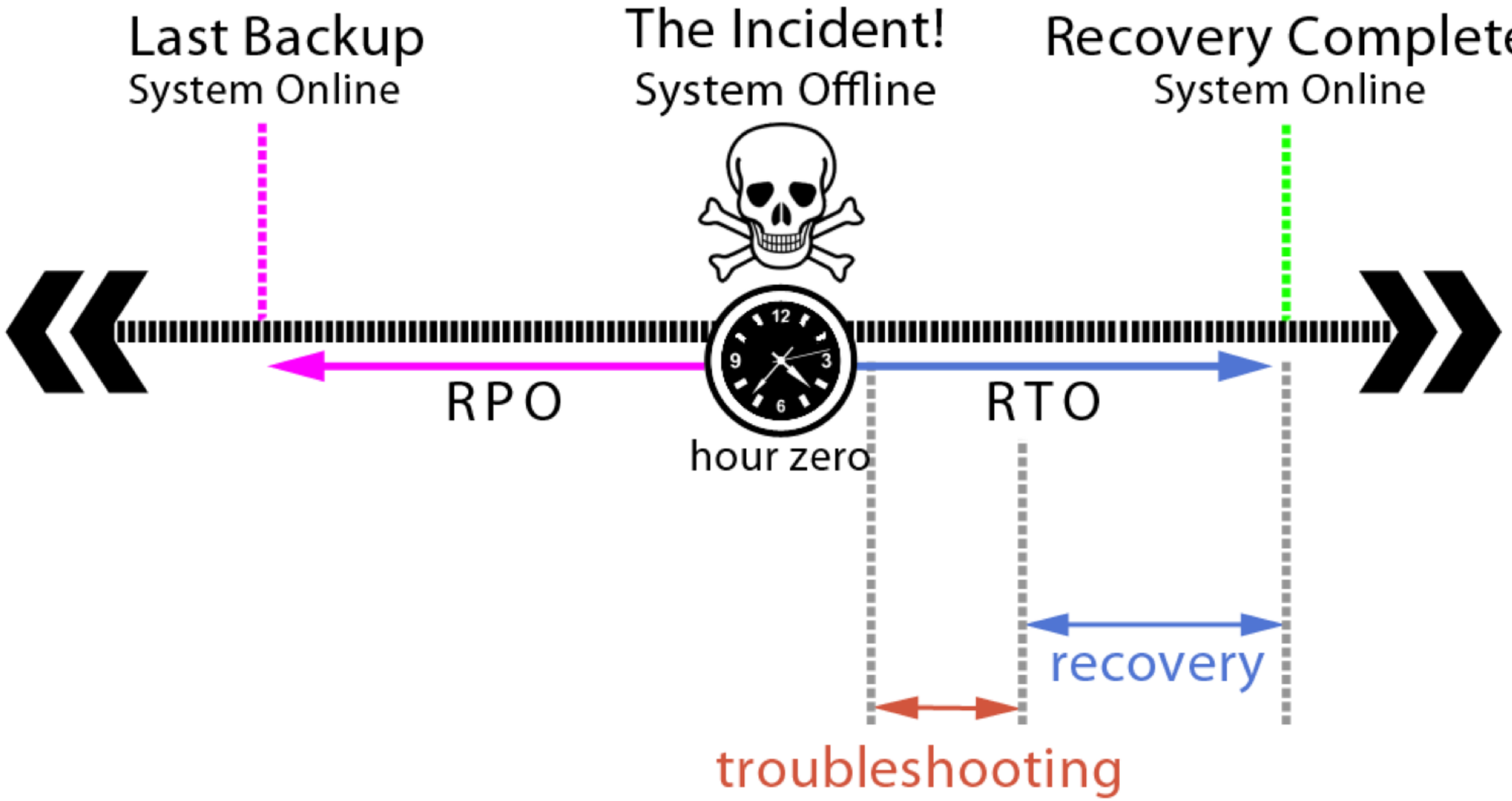
Disaster Recovery Plan



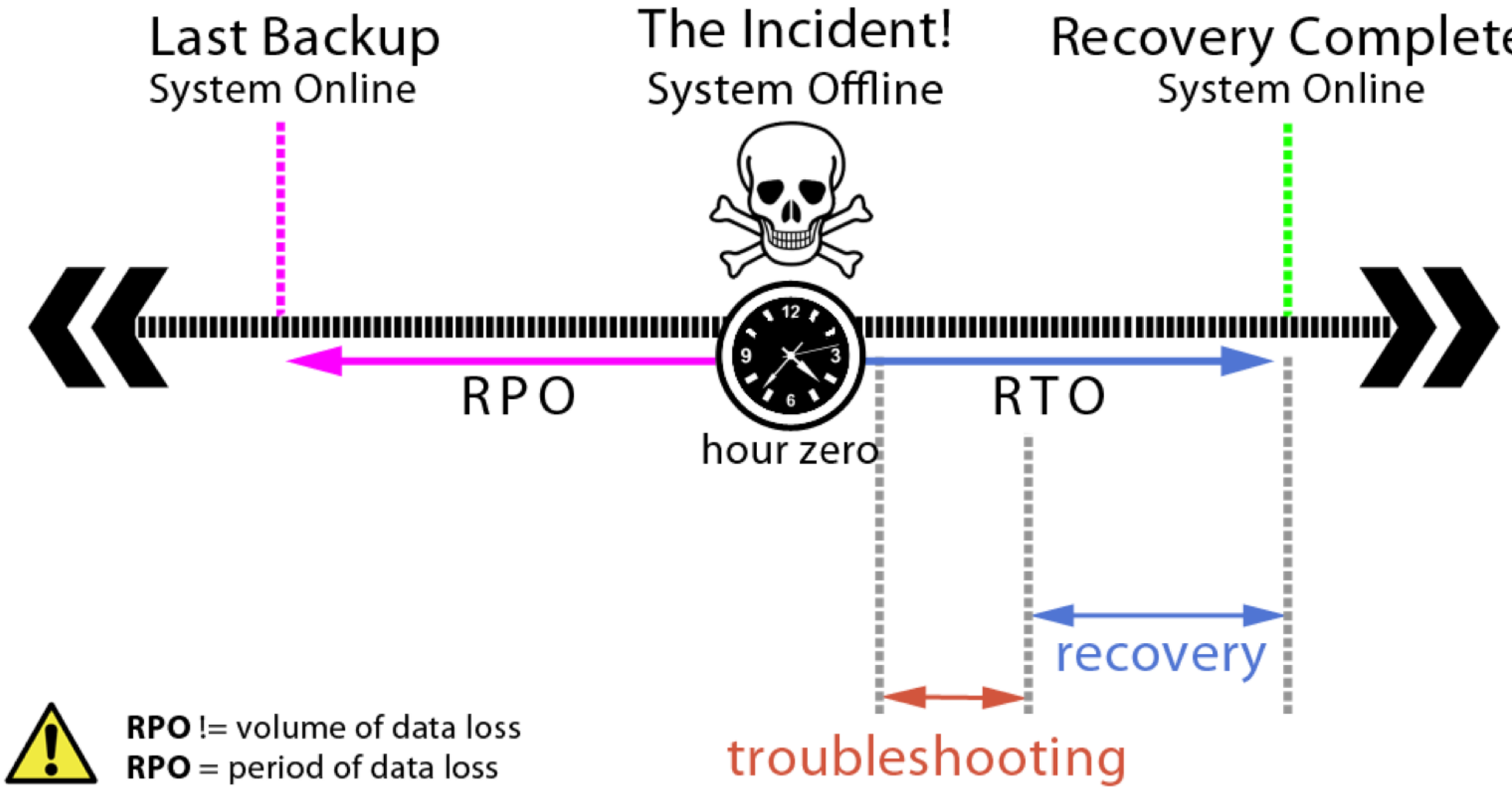
Disaster Recovery Plan



Disaster Recovery Plan



Disaster Recovery Plan



RPO != volume of data loss
RPO = period of data loss

传统恢复方式vs新恢复方式

传统：全备.gz+binlog恢复

```
传统恢复之基于时间点恢复
1）解压全备
2）从binlog中提取错误SQL以及时间点A。
3）从binlog重释放SQL
4）将全备copy到恢复服务器，并prepare好。
5）将binlog中释放的SQL copy到恢复服务器上。
6）应用SQL到时间点A

传统恢复之基于时间点的单表恢复
1）解压全备
2）从binlog中提取错误SQL以及时间点A
3）从binlog重释放SQL，  并且只提取单表的操作，然后验证提取的SQL的正确性~~
4）将全备copy到恢复服务器，并prepare好
5）将binlog中释放的SQL copy到恢复服务器上。
6）应用SQL到时间点A
```

改进：全备+日志服务器恢复（利用自带的replication 重演日志）

```
新恢复之基于时间点恢复
1）在binlog机器上Server1搭建日志服务器，让日志服务器认领需要恢复的binlog
2）在Server2上恢复全备，并prepare好。   --重点，无须解压
3）从binlog中提取错误SQL以及时间点A
4）让Server2同步Server1，change master从relay-log.info 开始，时间点A结束（start slave until position）。

新恢复之基于时间点的单表恢复
1）在binlog机器上Server1搭建日志服务器，让日志服务器认领需要恢复的binlog
2）在Server2上恢复全备，并prepare好。   --重点，无须解压
3）从binlog中提取错误SQL以及时间点A
4）让Server2同步Server1，change master从relay-log.info 开始并且设置 replica-do-table，时间点A结束（start slave until position）。
```

日志服务器恢复?

思路:

1）利用IO-Thread恢复

1.1）在关闭mysql的情况下修改mysql-bin.index,让mysql认领拷贝过来的binlog为自己的binlog

1.2）然后让需要恢复的机器change master 过来，完成恢复

2）利用SQL-Thread恢复

2.1） 在关闭mysql的情况下，修改relay-log.info && relay-bin.index ，让其认领拷贝过来的binlog为自己的relay-log

2.2) 然后开启SQL-thead来完成恢复

为什么选择复制来完成恢复?

a） 复制是最可靠的日志恢复方式，这无可厚非。

b） mysqlbinlog可能无法生成二进制日志中得数据跟新

c） 复制的速度更快速，因为无需将sql从日志中导出并传送给mysql

d） 可以很容易的观察到复制的过程

e） 能够更方便的处理错误。例如：skip slave counter

f） 更方便的过滤复制事件。

g） mysqlbinlog可能会因为日志记录格式更改而无法读取二进制日志。

h） 避免了编写复杂脚本的难度和出错性，更加易于维护和使用。

i） 最主要的还有速度，新恢复是原来的4倍速度以上。

j） 如果是多线程复制，速度将是指数级别的提高。

Thanks All