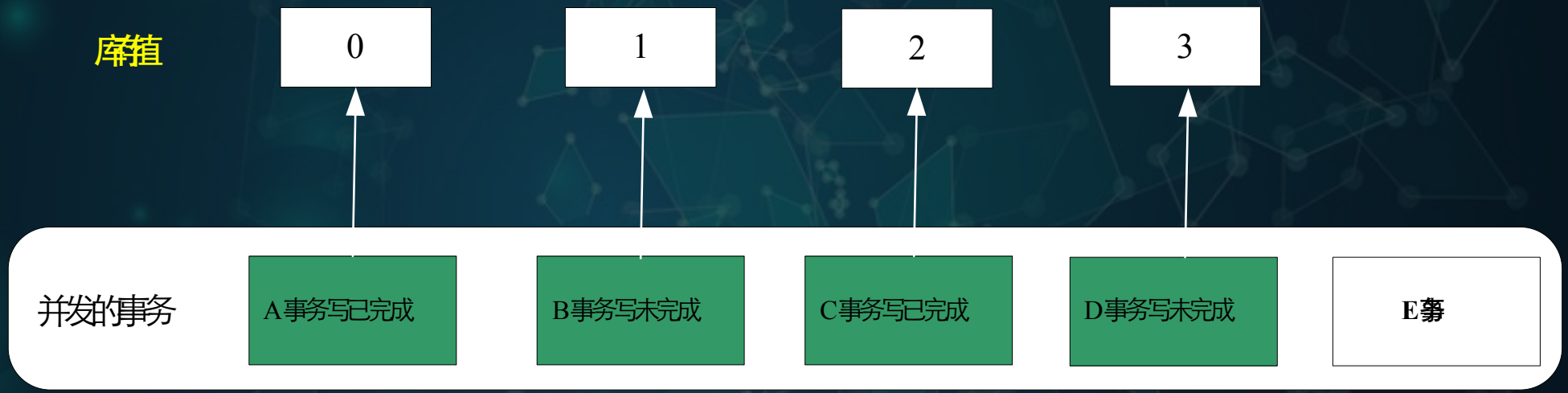
WiredTiger存储引擎

目录
CONTENTS

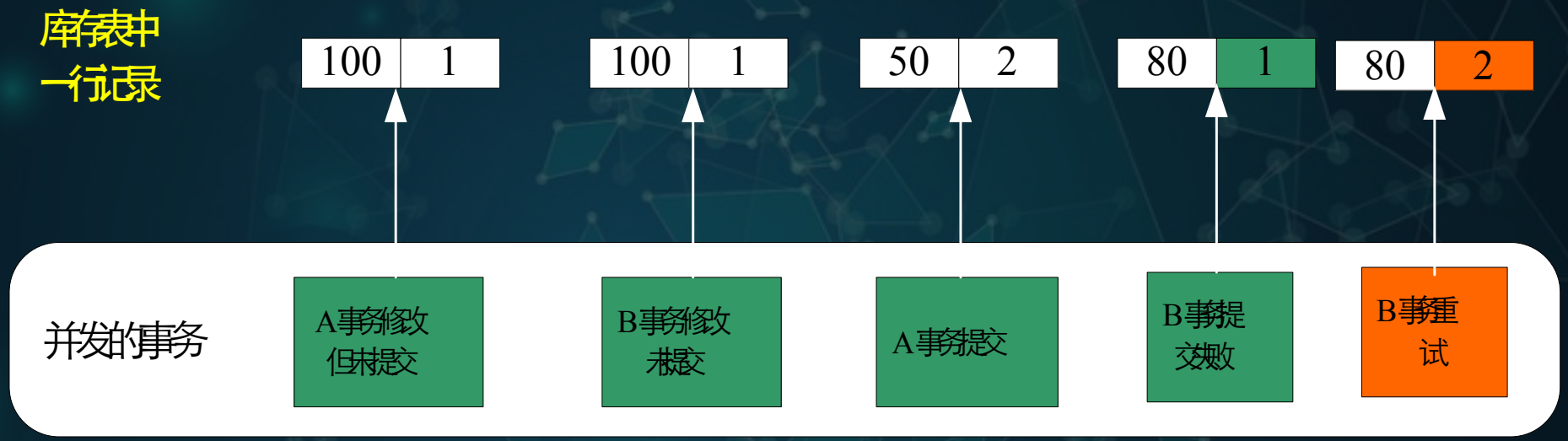
插件式存储引擎架构

事务特性与快照隔离级别



1. E事务开始时，产生一个快照，快照里面包含B和D两个未完成的写事务。
2. 如果E事务是读事务，则读取到库存值是在C事务完成后的值2，即使后面D事务提交完成了，E事务读取到的值也不会改变。
3. 如果E事务为写事务，对库存值进行修改，则会发生冲突（通过**MVCC**实现冲突检测），这样能防止对过期数据的修改，保证数据的一致性。

MVCC实现事务的冲突检测与并发



1. A事务首先从表中读取到要修改的行数据，读取到库存值为100，行记录的版本号为1

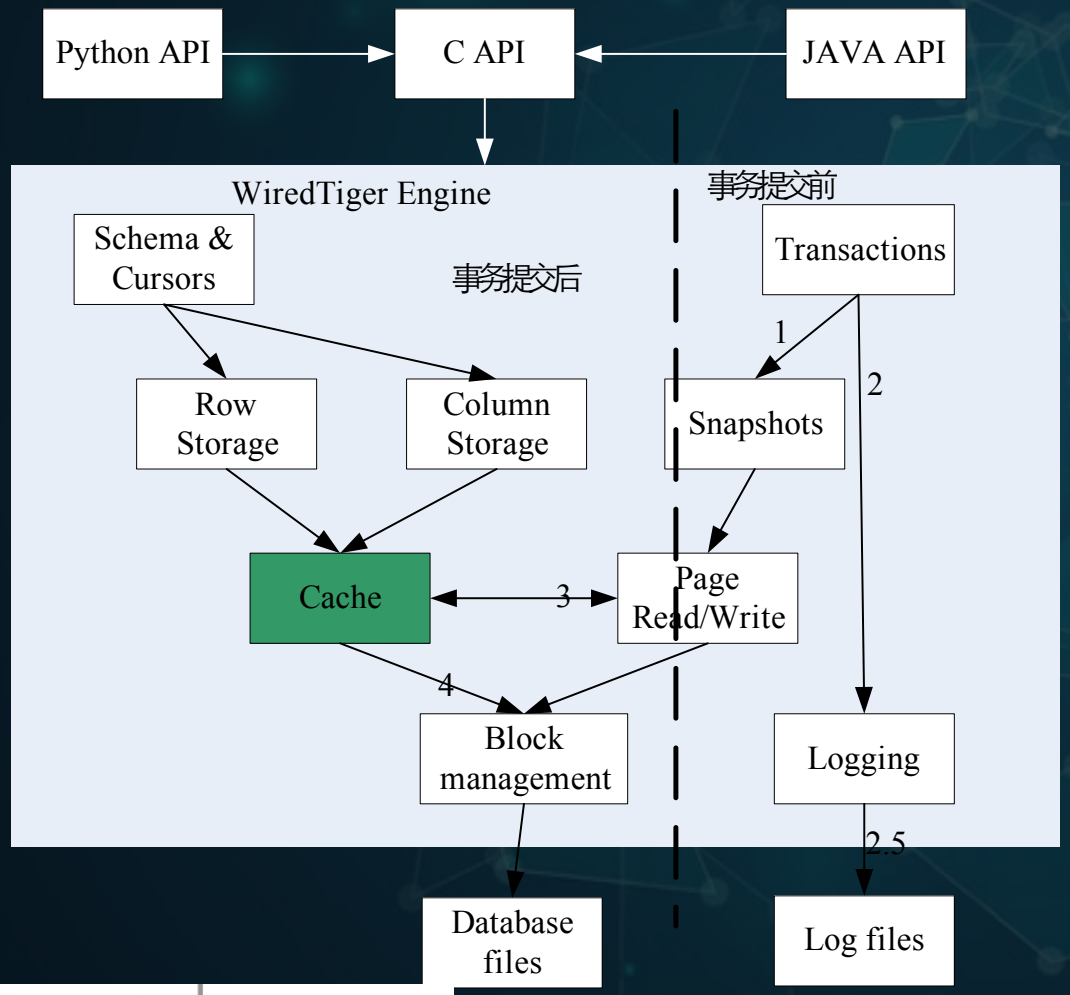
2. B事务也从中读取到要修改的相同行数据，读取库存值为100，行记录版本号为1。

3. A事务修改库存值后提交，同时行记录版本[illegible]1，即变为2，大于一开始读取到的版本，A事务可以提交。

4. 但B事务提交时发现此时行记录版本号已经变为2，产生冲突，B事务提交失败。

5. B事务尝试重新提交，此时读取到的版本号为2，加1，即变为3，不会产生冲突正常提交。

典型的写操作事务流程



1. 写操作事务开始执行之前，对所有正在执行的还未提交事务进行快照。

2. 将本次写操作的动作保存到operation_array中，可以从中提取出动作进行回滚；其次将修改的数据以日志形式记录下来，记录到日志缓冲区。

3. 写操作事务提交，首先将日志缓冲区中的数据刷到磁盘上，写入到log文件，数据库意外宕机恢复时需要读取这个文件，重演文件里面的动作。

4. 写操作引起的数据变化，首先写入到WiredTiger存储引擎的cache中，cache中的数据以Btree结构组织，Btree的叶子节点是真正存放数据的page，当数据发生更改时page就变为“脏页”（在内存中）；
存储引擎默认每60s将“脏页”中的数据写到物理磁盘上进行持久化。

引入checkpoint机制后的典型写操作流程

刷盘周期

内存中被修改的数据按一定周期(默认60s)或一定条件（如果开启了Journal，则日志文件大小达到2G）时能刷到磁盘上进行持久化

触发Checkpoint

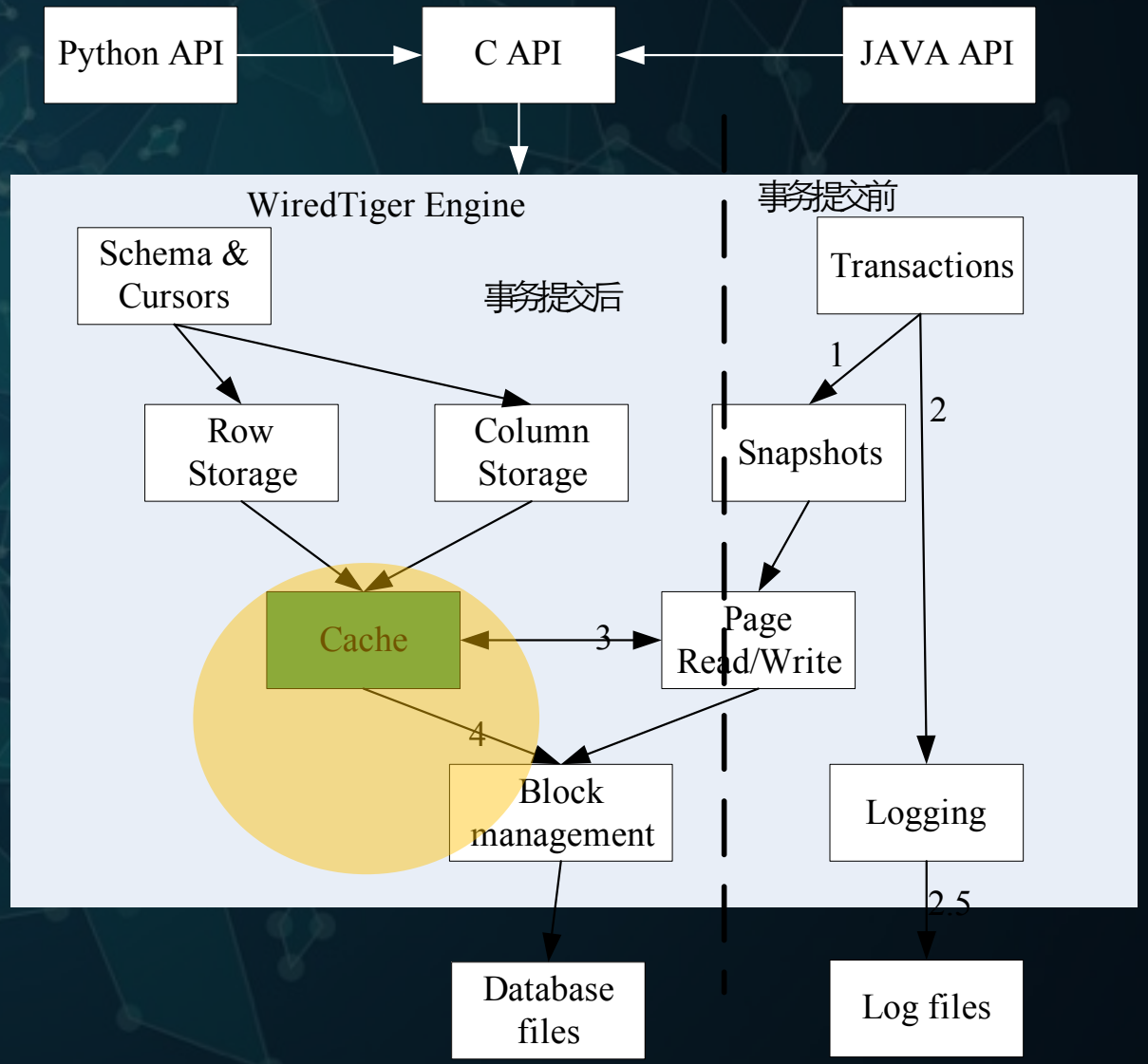
当刷盘条件触发时，持久化内存中的修改page，就会产生一个checkpoint点在数据文件上

数据恢复

当数据库意外宕机恢复时，只需要从最新的checkpoint点进行恢复，这样节省恢复时间

清理内存

一旦新的checkpoint完成后，内存中旧的checkpoint点到新的checkpoint点间的数据占用的page可以释放掉



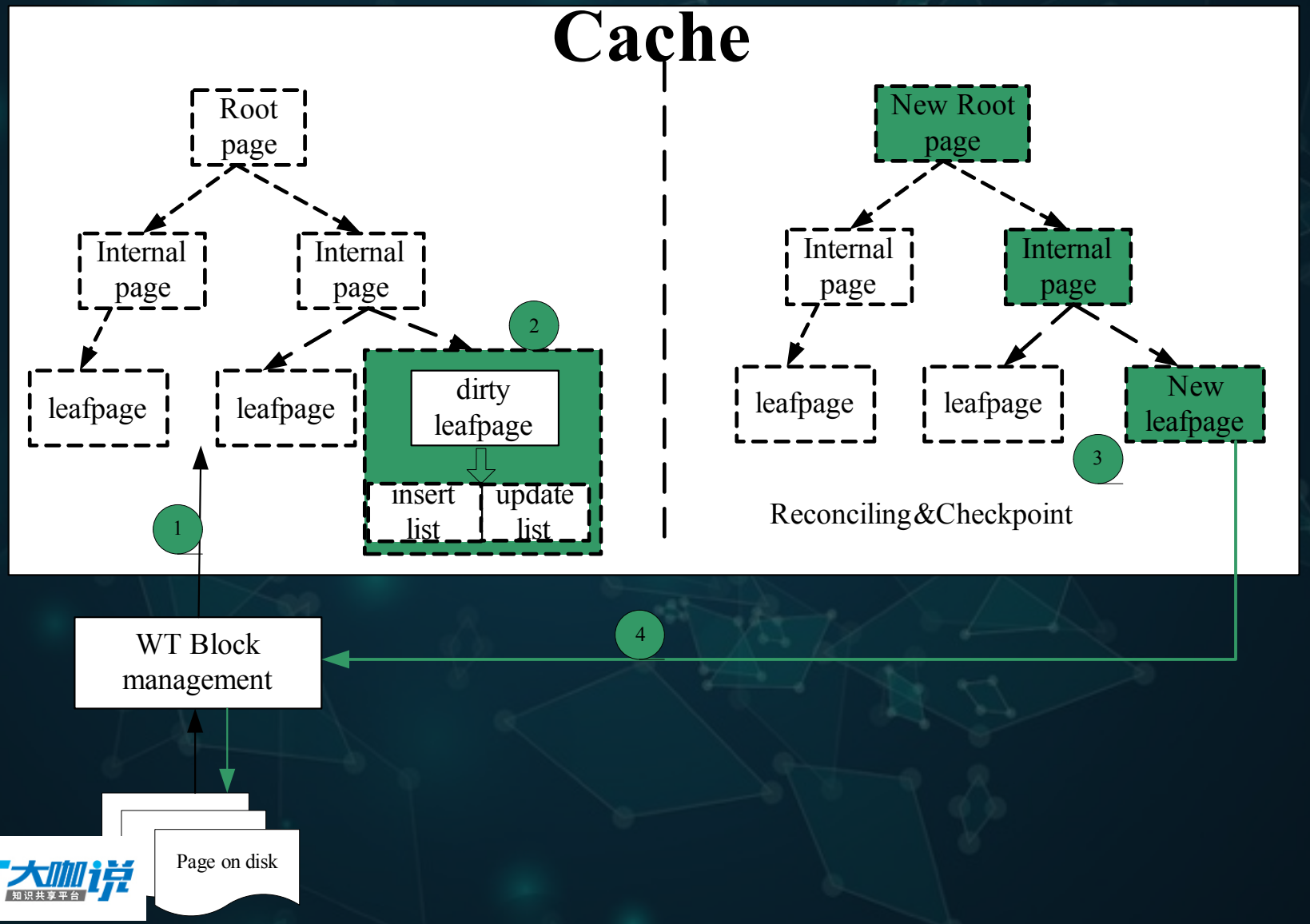
WiredTiger对内存的使用情况

内部内存
Internal
Cache

文件系统缓存
File System
Cache

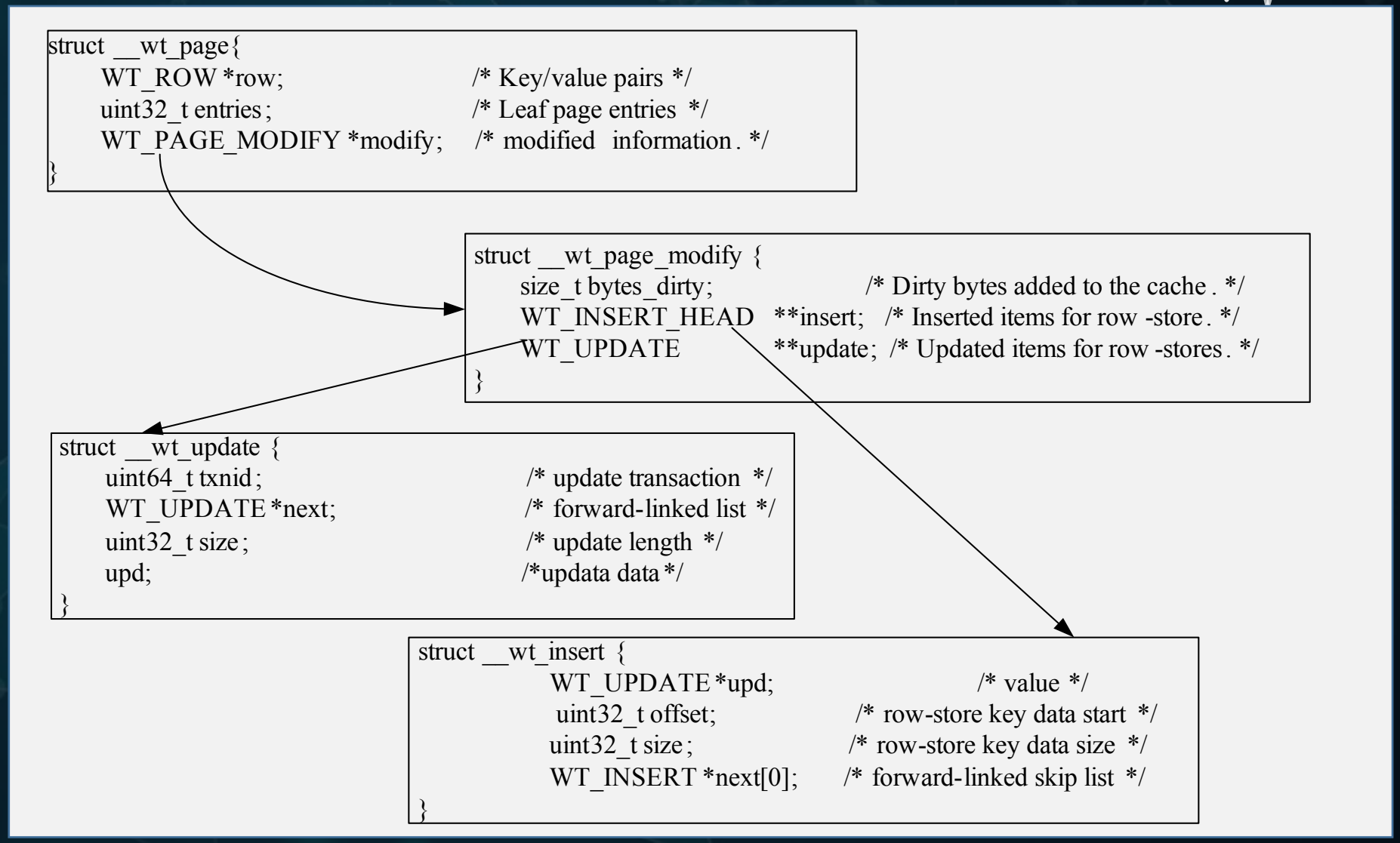
- 内部内存默认取值：50% of (RAM - 1 GB), or256 MB.
- 索引和集合的数据都被加载到内部内存，索引是被压缩的放在内部内存，集合是没由被压缩的放在内部内存.
- 通过文件系统缓存，自动使用其它所有空闲内存.
- 放在文件系统缓存里面的数据，与磁盘上的数据格式一致，可以减少磁盘I/O

Internal Cache的内部结构



与Internal Cache相关的几个数据结构

leafpage



压缩特性

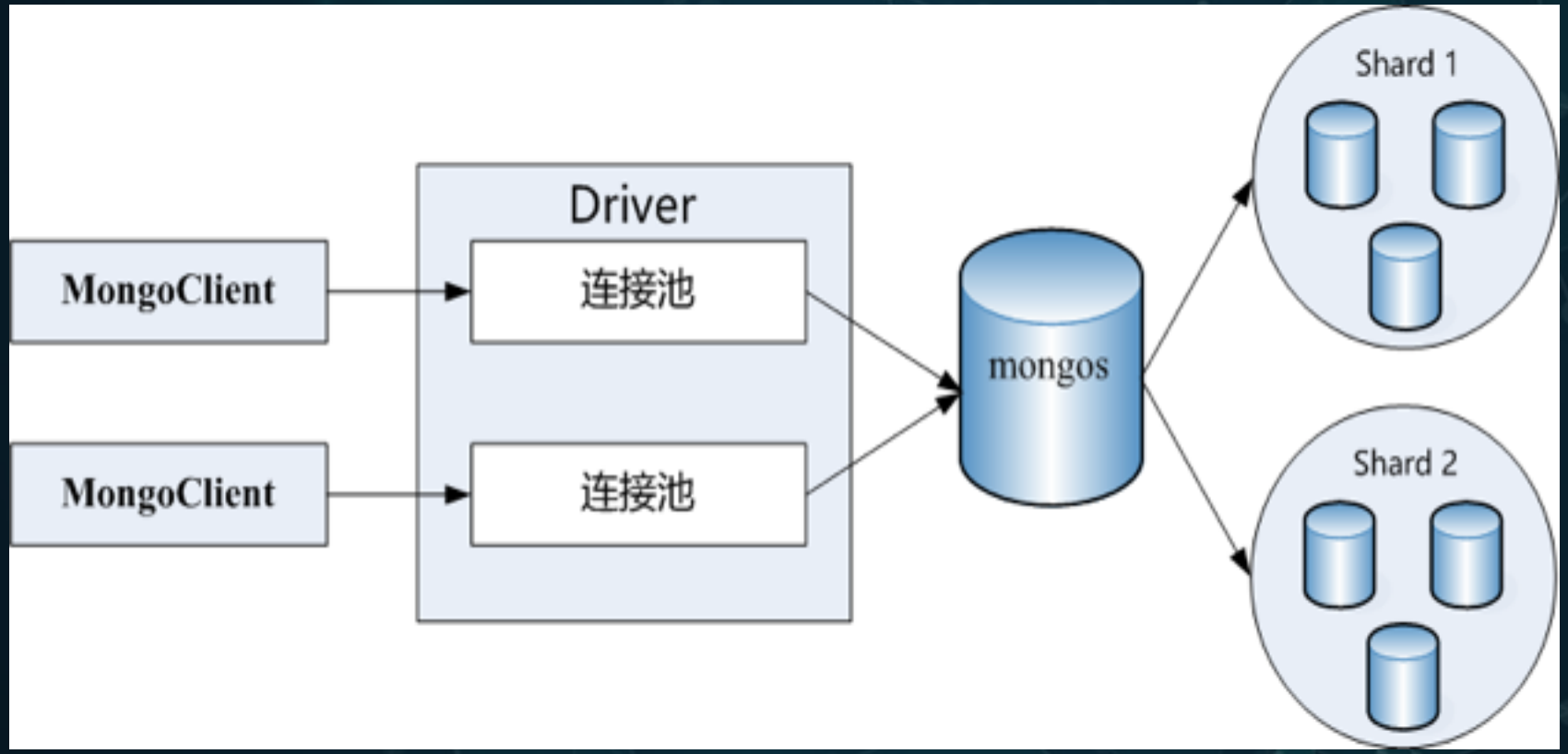
MMAPV 1

```
> db.customers.stats()
{
        "ns" : "customers.customers",
        "count" : 999998,
        "size" : 111999776,
        "avgObjSize" : 112,
        "numExtents" : 12,
        "storageSize" : 174735360,
```

Wired Tiger

```
> db.customers.stats()
{
        "ns" : "customers.customers",
        "count" : 999998,
        "size" : 63999872,
        "avgObjSize" : 64,
        "storageSize" : 20430848,
        "capped" : false,
        "wiredTiger" : {
```

案例：应用程序开发连接池问题



1. 驱动程序连接池的默认大小
2. 服务器端能够支撑的最大连接数

MongoDB集群随着访问量的增加，业务繁忙时出现连接拒绝的错误

案例：在代码里面修改驱动程序默认连接池大小

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:40000')
print(client.max_pool_size)
```

默认值是100

client = MongoClient(‘mongodb://localhost:40000/?maxPoolSize=200’) //传入参数修改默认值

案例：修改启动参数配置服务器端能支撑的并发数

通过命令db.serverStatus().connections查看

```
> db.serverStatus().connections
{ "current" : 4, "available" : 815, "totalCreated" : 150 }
```

最大连接数由maxIncomingConnections (4.0版本里面参数名改为maxConn，默认为65536)和操作系统单个进程能打开的最大文件描述符数总量的80%决定的(/etc/security/limits.conf)，取两个之间的最小值。

```
root@ubuntu:/home/guoyw# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 3643
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 3643
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

```
dbpath = /usr/local/mongodb-4.0/data
logpath = /usr/local/mongodb-4.0/logs/123.log
journal = true
bind_ip = localhost,192.168.248.130
port = 50000
auth = true
maxConns = 10
```

感谢聆听