Gdevops

全球敏捷运维峰会

新硬件环境下的数据库技术

演讲人：朱阅岸

内容大纲

1. 现代处理器及新型存储的发展
2. 现代处理器下的数据库技术
3. 面向新型存储的数据库系统
4. 总结

Do not go gentle into that good night,

.....

Though wise men at their end know dark is right,

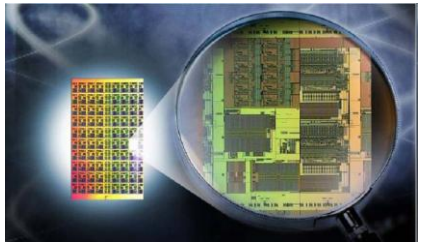
.....

内容大纲

1. 现代处理器及新型存储的发展
2. 现代处理器下的数据库技术
3. 面向新型存储的数据库系统
4. 总结

现代处理器

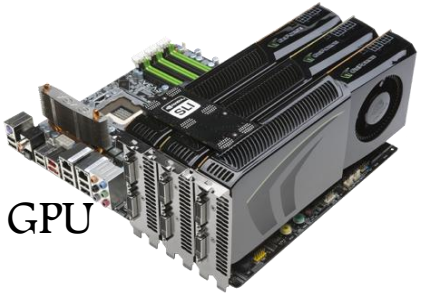
- 功耗与制造工艺的原因使得处理器制造商不再追求高频率，而是转向片上多处理器技术



Intel Haswell Ex
144 Cores

Xeon Phi
72 cores

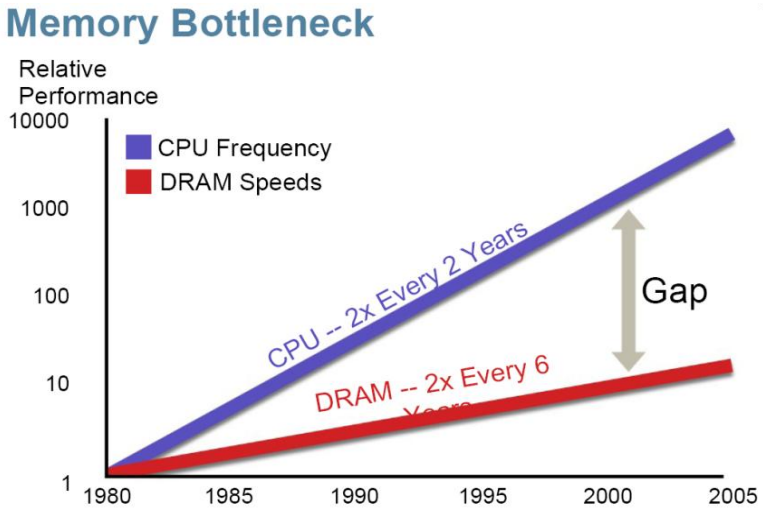
nVIDIA GeForce GPU
216 cores



Triple GeForce GPU
648 cores

现代处理器

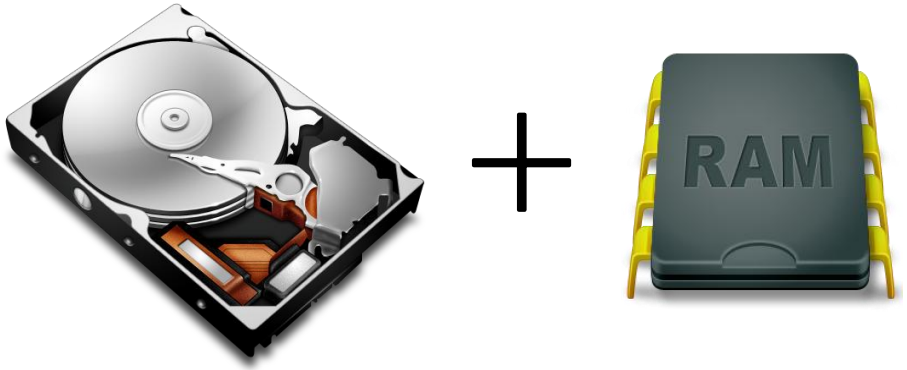
- CPU与内存的速度日益扩大，以前访问内存只需要一个CPU周期的时间，现在访问内存都成了昂贵的操作，出现了 "memory wall" 效应



CPU与Memory的速度日益增大

新型存储设备

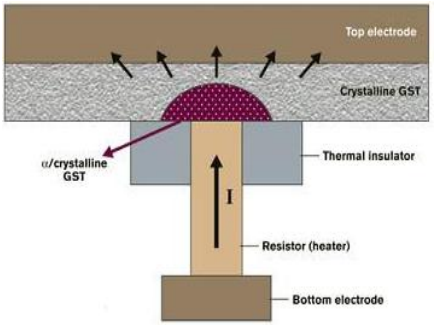
- 非易失性内存从实验室开始走向工业界，例如Intel傲腾
- 具有磁盘的持久存储特性与接近内存的访问速度
- 主要特性是非易失、低延迟、高密度和读写不对称



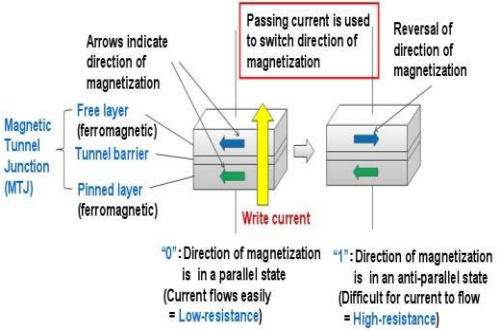
Non-Volatile Memory兼具磁盘与内存特征

新型存储设备原理介绍

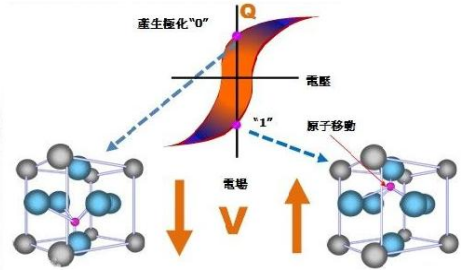
- 非易失性内存的技术主要有如下几种：
  - a) 相变存储器：材料可以在结晶状态与非结晶状态转变
  - b) 自旋磁矩：改变两层磁性材料磁矩方向
  - c) 铁电材料：材料所形成的电荷高低，二元状态
  - d) 忆阻器：是一种有记忆功能的非线性电阻



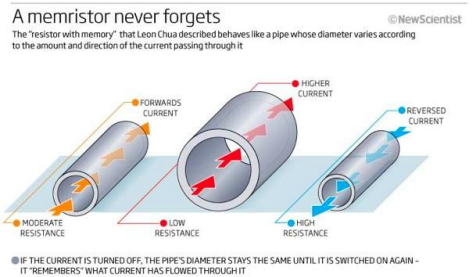
1. 相变技术(phase change memory)



2.自旋磁矩



3. 铁电材质



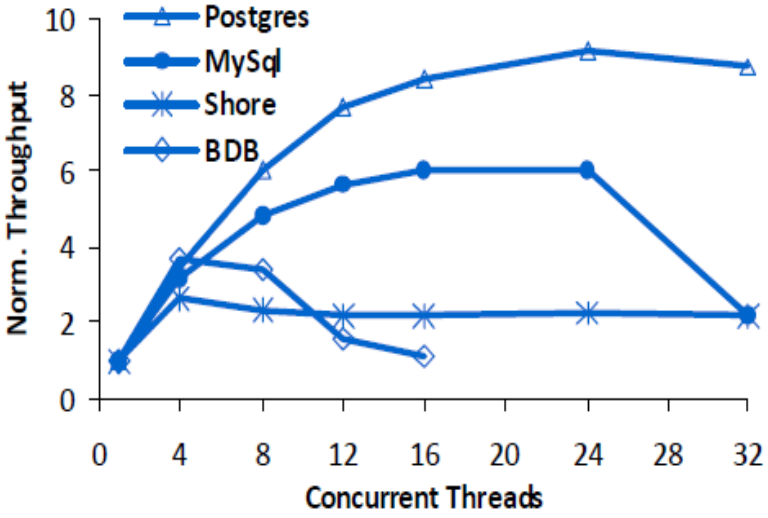
4. 忆阻器

相关参数

	DRAM	NAND Flash	PCM
页面大小 读延迟 写延迟 带宽 擦除延迟	64B ~100ns ~100ns ~GB/s per die N/A	4KB ~25 µs ~500 µs 5-40 MB/s per die 2 ms	64B ~100ns ~1 µs 50-100 MB/s per die N/A
寿命	∞	$10^4 - 10^5$	$10^6 - 10^8$
读耗能 写耗能 空闲耗能	0.8 J/GB 1.2 J/GB ~100 mW/GB	1.5 J/GB [28] 17.5 J/GB [28] 1–10 mW/GB	1 J/GB 6 J/GB ~1 mW/GB
密度	1×	4×	2 – 4×

DBMS的设计

- 底层硬件决定上层软件的设计
- 关系数据系统的开发始于上世纪70年代末，磁盘I/O是那个时期系统性能的主要瓶颈
- 数据库系统落后的设计观念与现代硬件的矛盾日益突出



不同并发下，各个系统的扩展性

ref: Shore-MT: A Scalable Storage Manager for the Multicore Era

时间都去哪儿了

- [SIGMOD 08]：OLTP through the looking glass, and what we found there
- 研究表明，传统的磁盘数据库只有约12%的CPU时间用于实际事务处理，
- 其它时间都耗费在缓存池管理、并发控制和恢复子系统等辅助性模块上

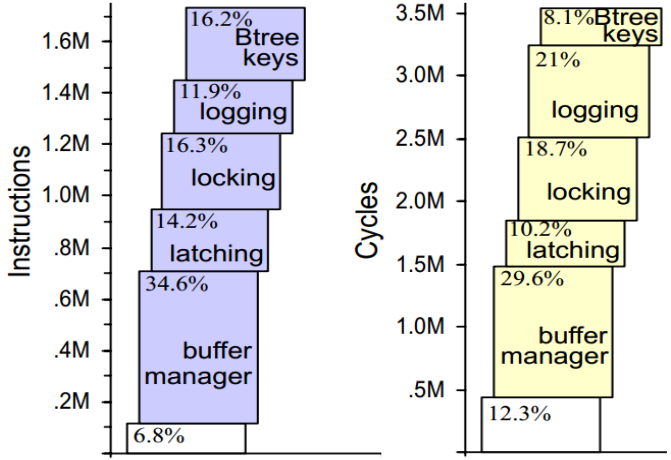
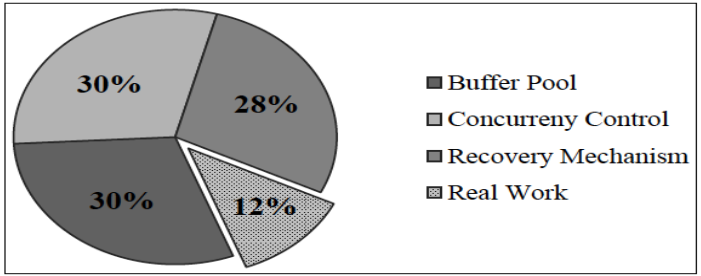


Figure 8. Instructions (left) vs. Cycles (right) for New Order.



传统的数据库系统的设计主要为优化磁盘IO与复用CPU而设计，并且这些模块中存在中大量的临界区阻碍系统的扩展性

内容大纲

1 现代处理器及新型存储的发展

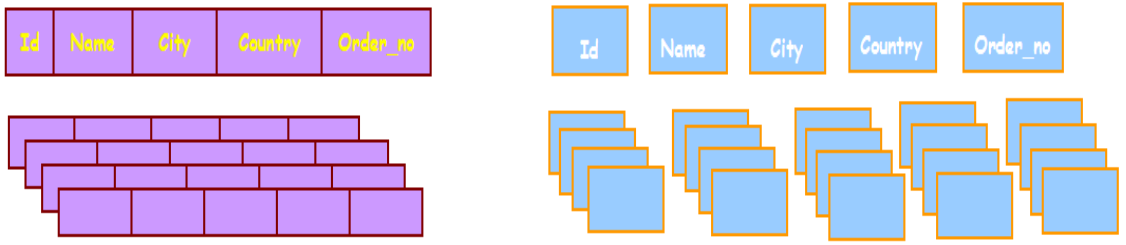
2 现代处理器下的数据库技术

3 面向新型存储的数据库系统

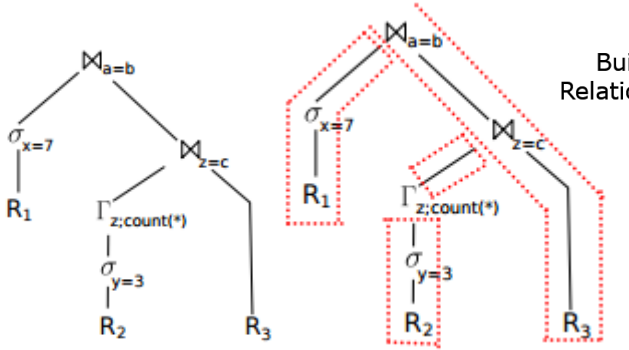
4 总结

RAM-Locality设计原则

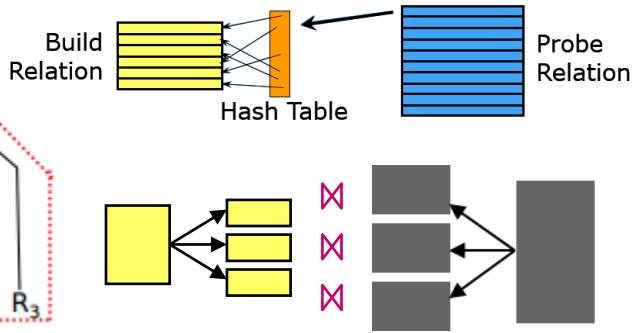
- 列存储技术：这主要应用在OLAP领域，例如列存储
  先驱MonetDB，为高效挖掘CPU的效率针对cache进行了大量的优化
- 设计高速缓存友好的数据结构与算法：例如cache line对齐、cache友好的表连接算法、向量化查询执行引擎等



row store Vs. colum store



向量化查询执行



cache友好的表连接算法

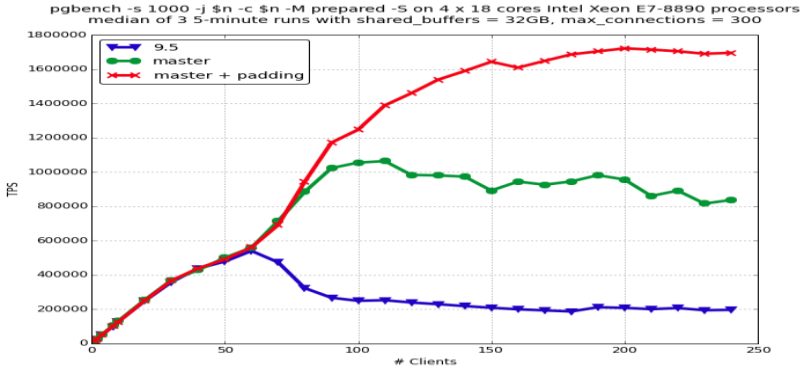
一个例子

```
struct PGPROC
{
...
TransactionId        xid;
TransactionId        xmin;
int                  pid;
BackendId            backendId;
Oid                  databaseId;
Oid                  roleId;
bool                 inCommit;
uint8                vacuumFlags;
...
};
```

25个成员

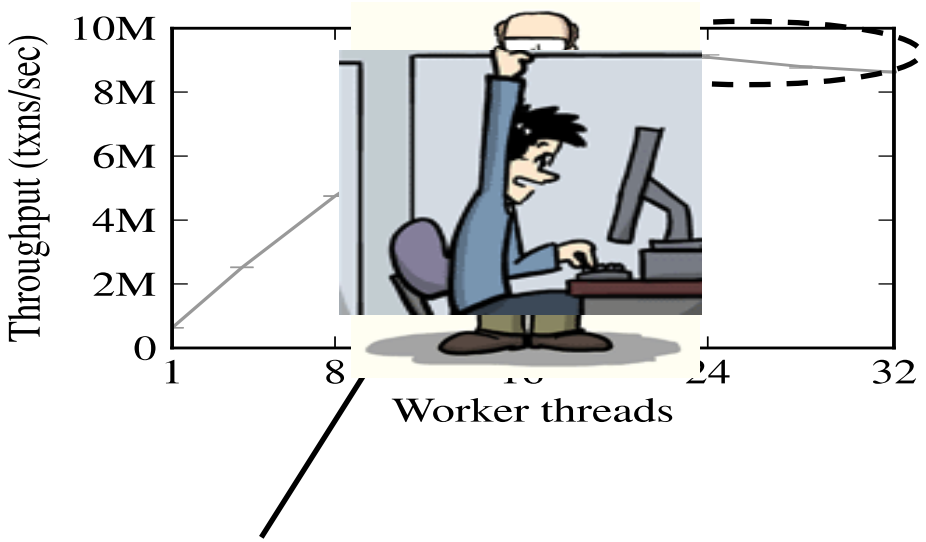
```
typedef struct PGXACT
{
TransactionId xid;
TransactionId xmin;
...
} PGXACT;
```

5个成员




```
for (i = 0; i < arrayP->numProcs; i++)
{
        volatile PGPROC *proc = arrayP->procs[i];
        TransactionId pxid = proc->xid;
        if (!TransactionIdIsValid(pxid))
                continue;
        ...
        if (TransactionIdEquals(pxid, xid)){
                result = true;
                break;
        }
}
```

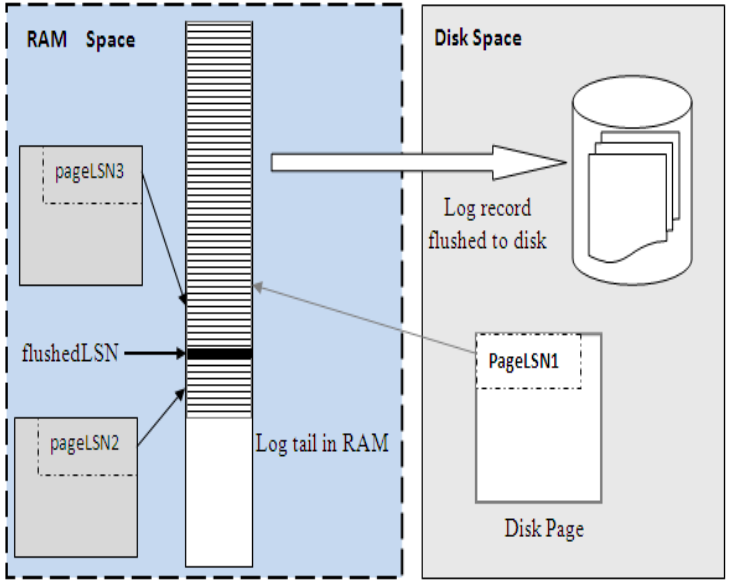
避免热点与简化临界区




```
txn_commit()
{
   // prepare commit
   // […]
   commit_tid = atomic_fetch_and_add(&global_tid);
   // quickly serialize transactions a la Hekaton
}
```

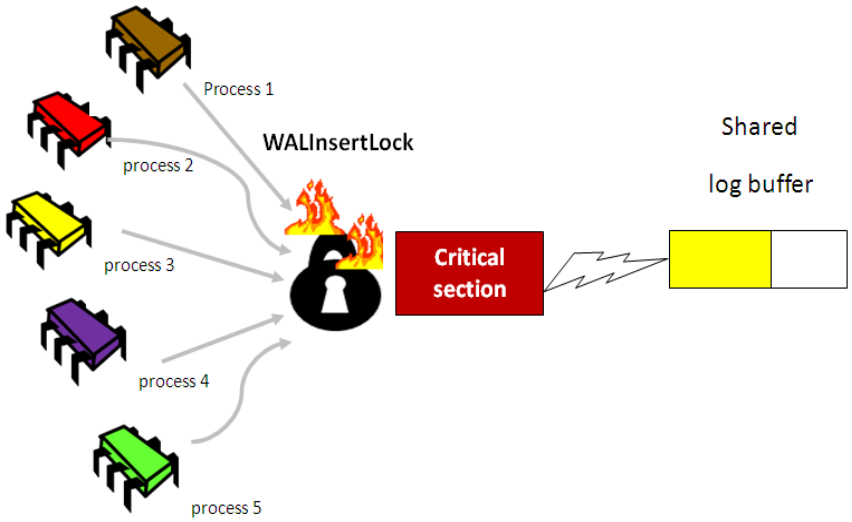
避免热点与简化临界区

- InnoDB的性能瓶颈目前主要集中在锁管理器与日志模块；PostgreSQL的性能瓶颈体现在缓冲区管理模块与日志模块
- 先写日志。数据库通常采用的一种事务日志实现方法



避免热点与简化临界区

- 传统的日志提交算法主要有三个步骤：
  1. 首先获取写日志缓冲区上的排他锁
  2. 线程将日志记录拷贝到相应的日志缓冲区
  3. 释放缓冲区上的锁

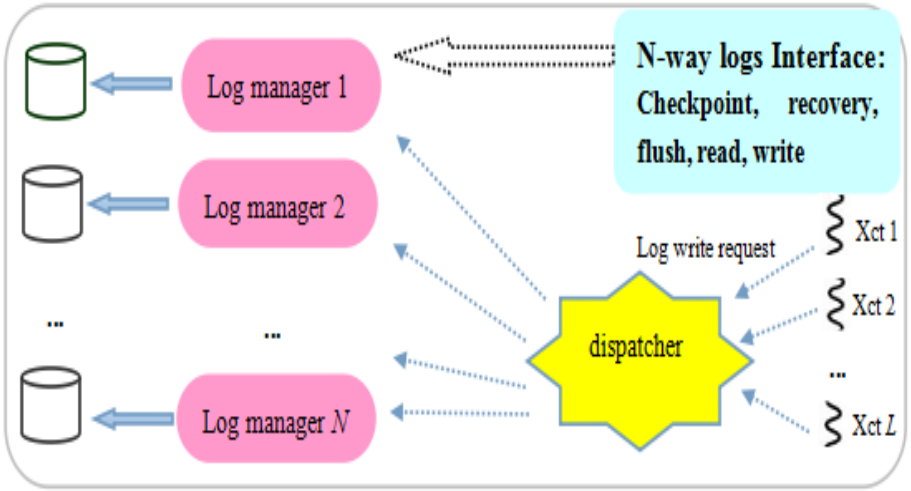


WAL的实现引发的热点问题

避免热点

- 分布式日志

1. 日志管理器上竞争激烈的根本原因是系统强制给不相关的更新事务规定顺序。实际上，系统大部分事务没有冲突，并行地执行
2. 将负载分散到N个不同的日志管理器上，提高写日志的并行性



简化临界区

```
START_CRIT_SECTION();
/* Now wait to get insert lock */
#933  LWLockAcquire(
        WALInsertLock,
        LW_EXCLUSIVE);

      ......
      copy the redo record

#1220 LWLockRelease(
        WALInsertLock);
END_CRIT_SECTION();
```

Pg9.4之前的WAL实现

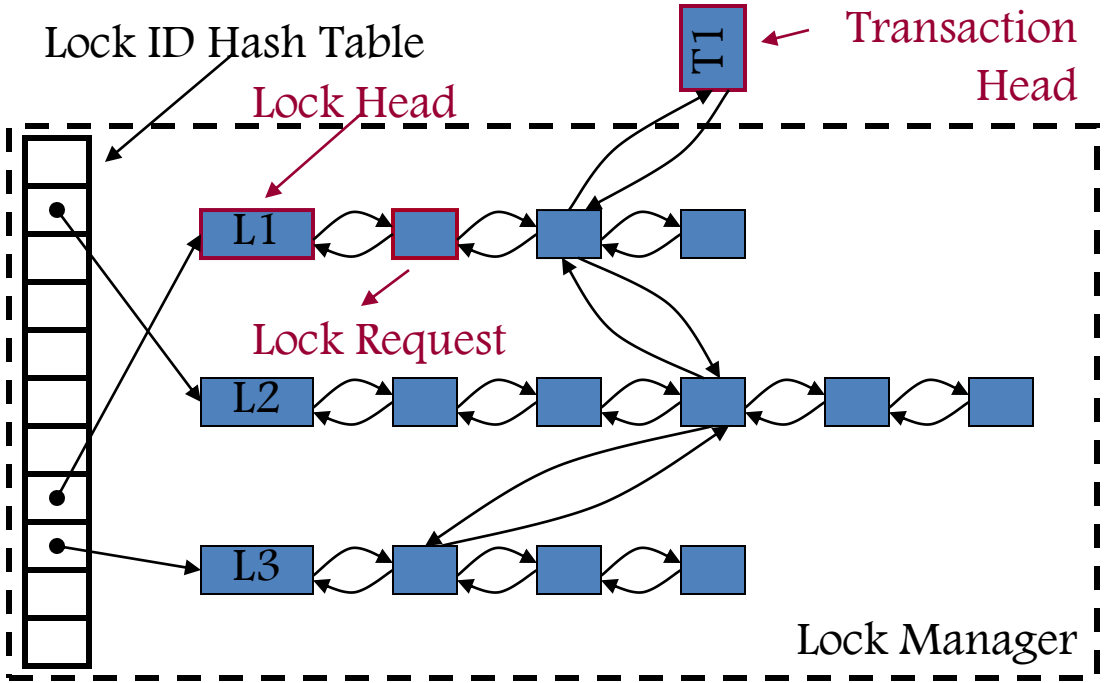
```
SpinLockAcquire(
       &Insert->insertpos_lck);

startbytepos         = Insert->CurrBytePos;
endbytepos           = startbytepos + size;
prevbytepos          = Insert->PrevBytePos;
Insert->CurrBytePos = endbytepos;
Insert->PrevBytePos = startbytepos;

SpinLockRelease(
       &Insert->insertpos_lck)
```

优化的WAL实现

锁管理器（锁申请）



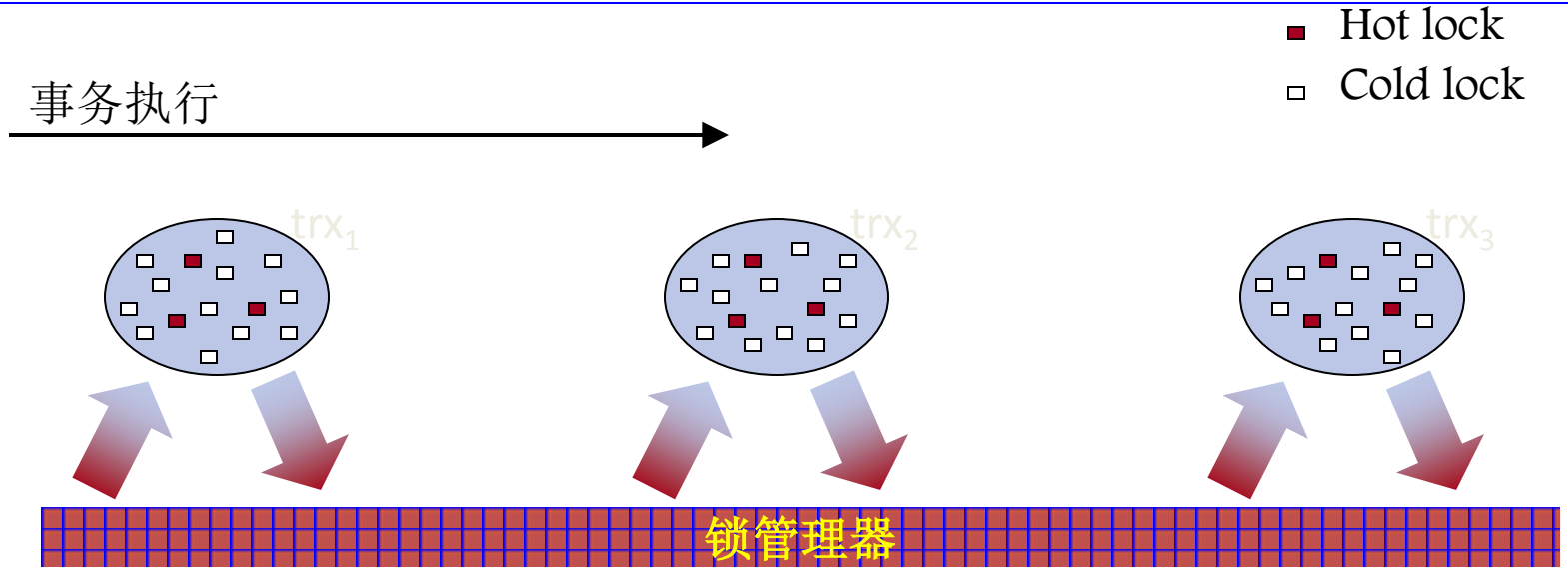
Requirements

⇒ 并发地查找或者创建锁

⇒ 每个锁追踪多个请求

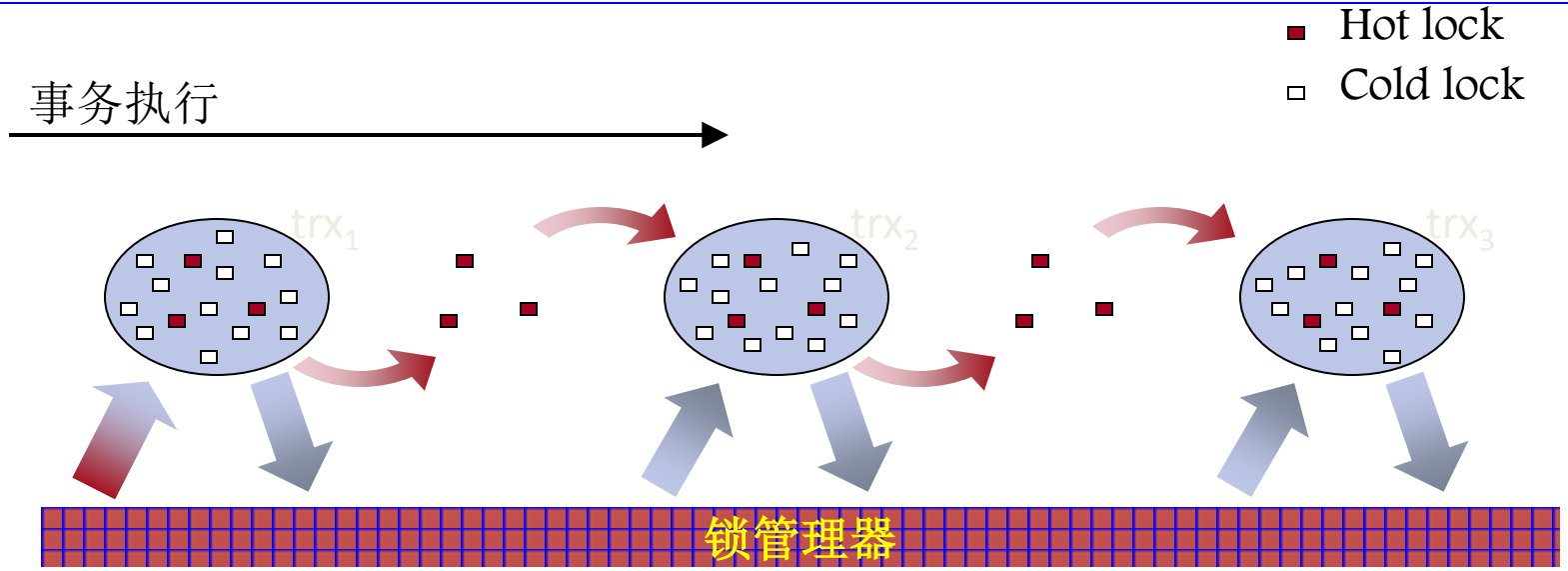
⇒ 每个事务需要保存申请的锁

热点问题



频繁地修改共享数据结构，引发热点

减少与临界区的交互



代理线程将锁保留在本地，传递给下一个事务

内容大纲

1 现代处理器及新型存储的发展

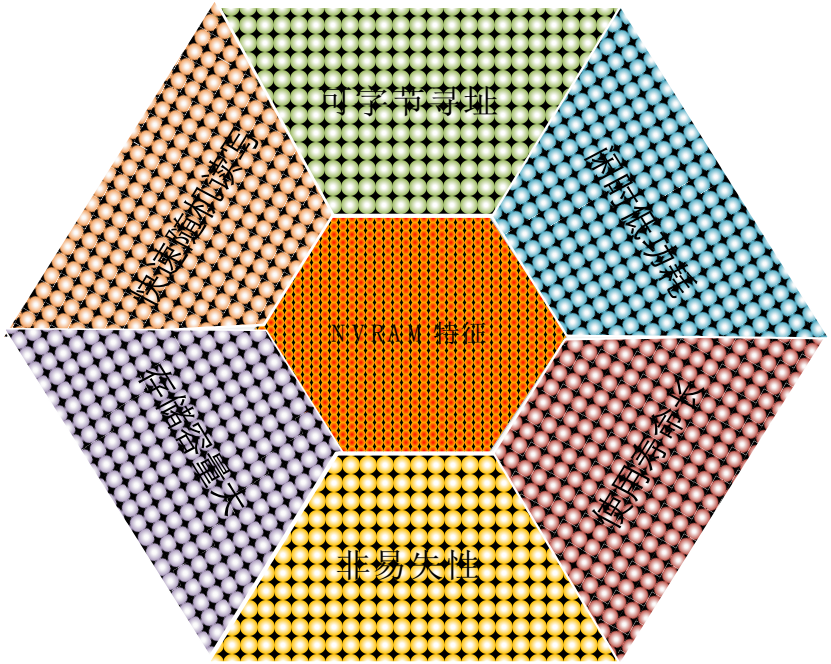
2 现代处理器下的数据库技术

3 面向新型存储的数据库系统

4 总结

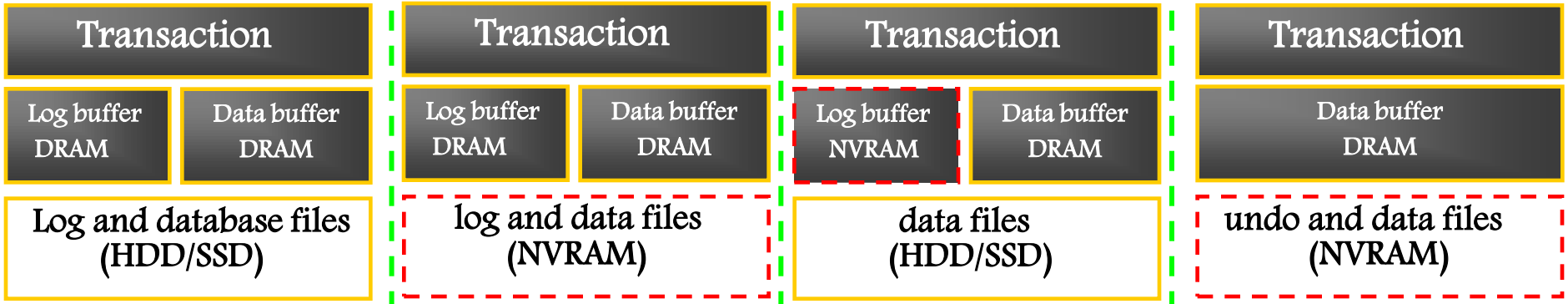
面向新型存储的数据库系统

- NVRAM之“六脉神剑”



面向新型存储的数据库系统

- NVRAM接入DBMS的三种方式



图a. 传统的设计　图b. 作为磁盘替代　图c. 作为日志存储　图d. 全系统接入

面向新型存储的数据库系统

- write-behind logging

1. 运行时只需追踪脏页，无需构造redo日志
2. 事务按时间片成组提交，提交时先写脏页，然后写日志。日志记录包含提交时间区间(Cp,Cd)，无需构造after-image

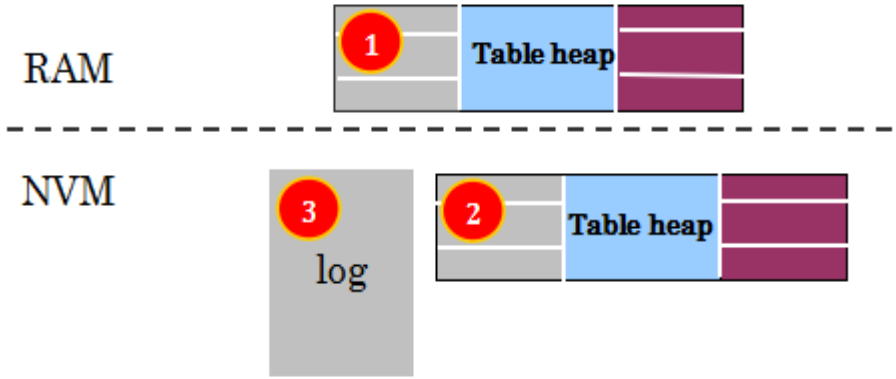


图. 刷盘次序

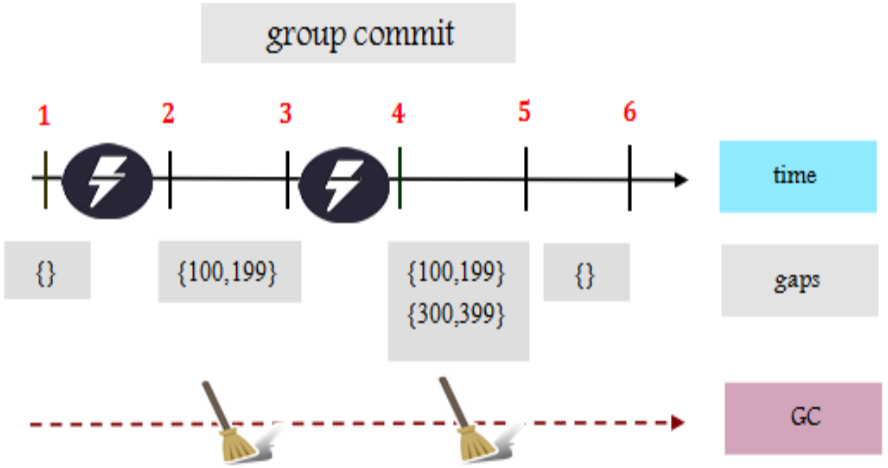
checksum	LSN	optype	persisted Commit timestamp (C_p)	dirty Commit timestamp(C_d)

图. 日志记录格式

面向新型存储的数据库系统

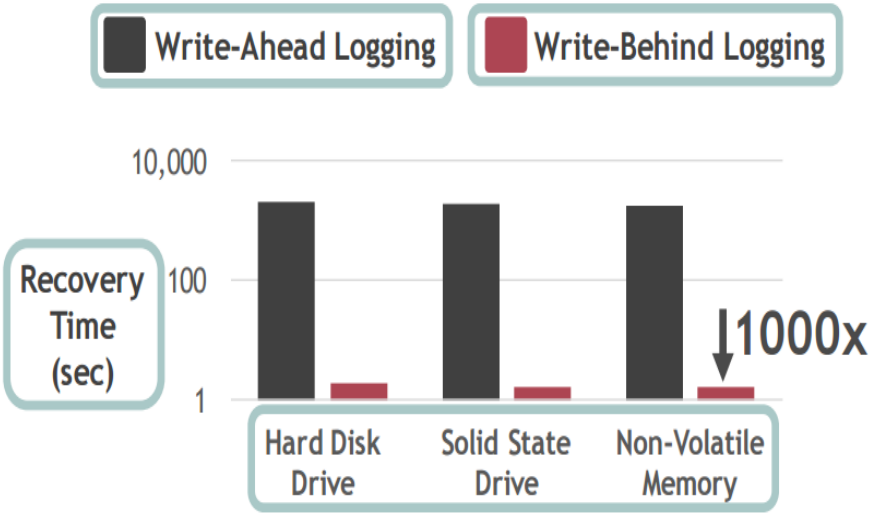
- write-behind logging

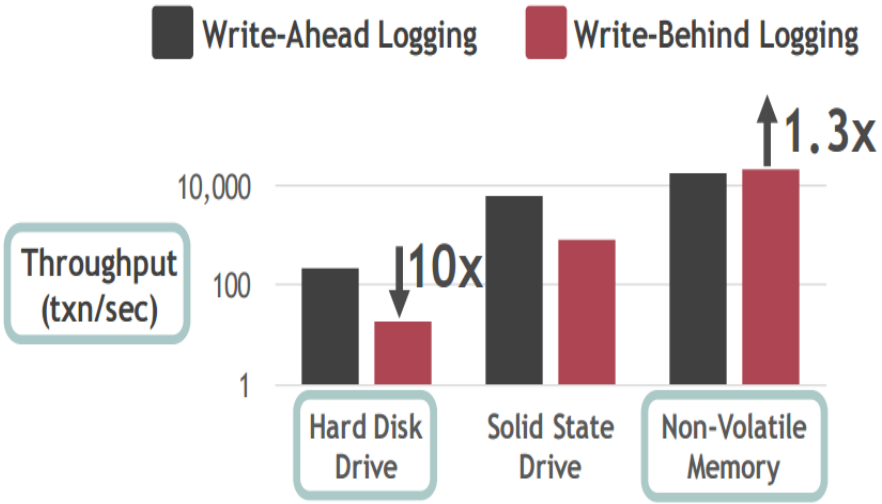
1. 系统崩溃恢复时刻，需要扫描日志，建立崩溃时间区间
2. 为了减少系统崩溃恢复阶段需要扫描的日志量，需要定期建立检查点
3. GC进程会清理崩溃时间区间



面向新型存储的数据库系统

- TPC-C benchmark





内容大纲

1. 现代处理器及新型存储的发展
2. 现代处理器下的数据库技术
3. 面向新型存储的数据库系统
4. 总结

总结

- 应用需求、数据以及计算机硬件是拉动数据库系统发展的“三驾马车”
- 多核与内存计算时代下，系统设计人员应当将更多的精力放在系统扩展性，更应注重数据访问的局部性
- NVM的出现使得系统设计人员可以将注意力完全地从I/O上移除，专注系统扩展性设计

这是最坏的时代，

这是最好的时代

全球敏捷运维峰会

THANK YOU!