RDMA

Basic knowledge

Kernel bypassing network: DPDK

- Userspace, driver lib 来使用网卡
- 优势
  - bypass kernel
  - 减少memcpy overhead
  - 用户可以定制化网络，不需要使用TCP/IP协议
- 不足：
  - 扔就需要软件栈来实现general-purpose networking, 比如用户态协议栈

CPU bypassing:

- CPU很难变快了 ——power wall
- NIC网卡持续变快， 200Gbps
- 数据中心需要节能，polling会导致能耗高，中断会导致性能差(超过100X)
- 方案：单边RDMA，可以bypass CPU

RDMA目前的特定

- 价格更低
  - $19/Gbps (RDMA 40Gbps) vs. $60/60Gbps (Ethernet 10Gbps)
- 向下兼容性(downward compatibility)
  - RDMA atop RoCE supports fast ethernet (compatible with existing datacenter apps)
- 在现代数据中心中应用广泛
  - Even avaliable in the public cloud

Case study: dsitributed kv store

Farm-KV @NSDI'14

Goal: 使用单边RDMA来建立分布式KVS

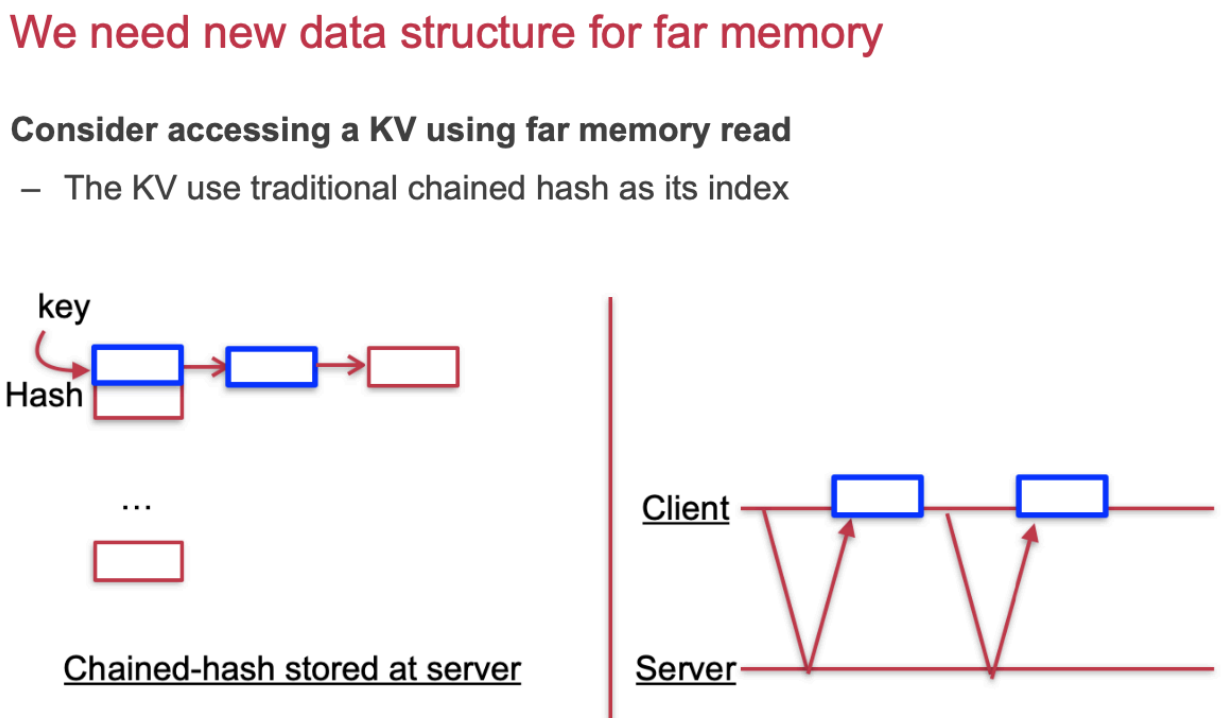
Question:

- 我们使用的是single node KVS
- 我们是否可以将内存访问替换为远端内存访问?

Problem:

- 单边RDMA速度还是比本地内存访问慢很多 （2 us vs 100ns）

传统HashIndex会增加更多的network roundtrips



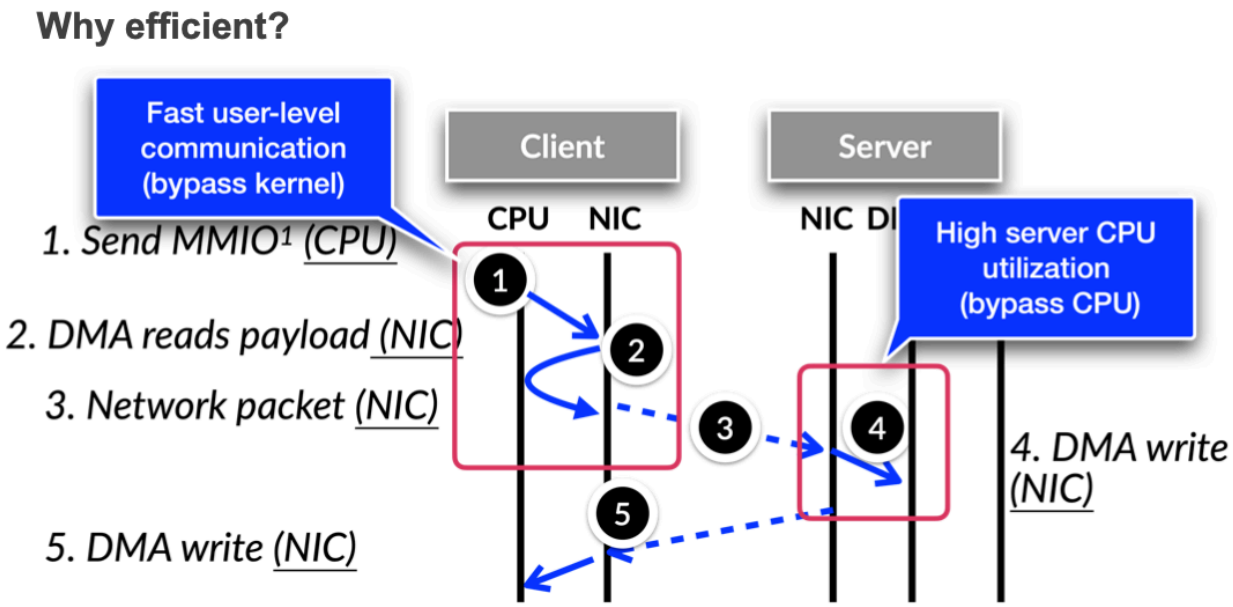
Network amplification: 远端内存在网卡端只有非常简单的抽象；实现high level的KV抽象需要更多NIC网卡操作

因此，FaRM对数据结构(hopscotch hashing)进行了重新设计。Hash之后的key对应的bucket，能够保证目标entry一定在有限个bucket offset中（相当于进行了一个限制）。这样就可以通过一次RDMA Read来把offset个bucket一次性读回。

但是Update会变得更加复杂（需要更多rehash操作）

RDMA实现细节

Life cycle of an RDMA one-sided WRITE



但是RDMA的cost还是很高：PCIe latency, network latency.

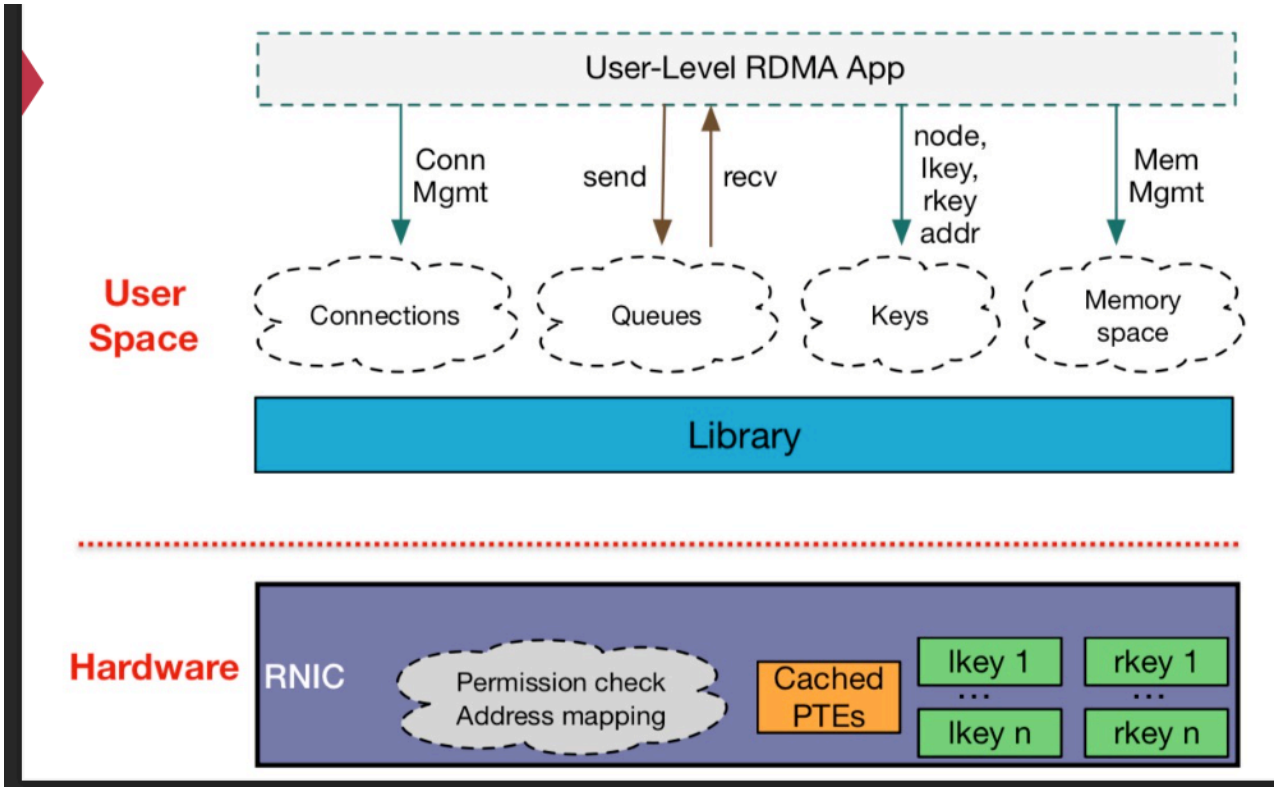
Outstanding request & coroutine

- Outstanding requests means keep multiple RDMA requests on-the-fly
- Coroutine is the same as traditional asynchronous I/O programming
- Goal: hide polling latency

doorbell batching

- One MMIO to post a batch of requests, NIC uses DMA to read them

RDMA: out memory design



Connection, translation, permissions, etc. information are stored on the host memory, NIC only caches them in its SRAM, but with a limited space(~2MB). When cache miss, use PCIe read to fecth them.

RDMA: out memory design makes cache miss costly

Driver stores these at the host memory

– NIC caches a portion of such information

Fetch miss items using PCIe read

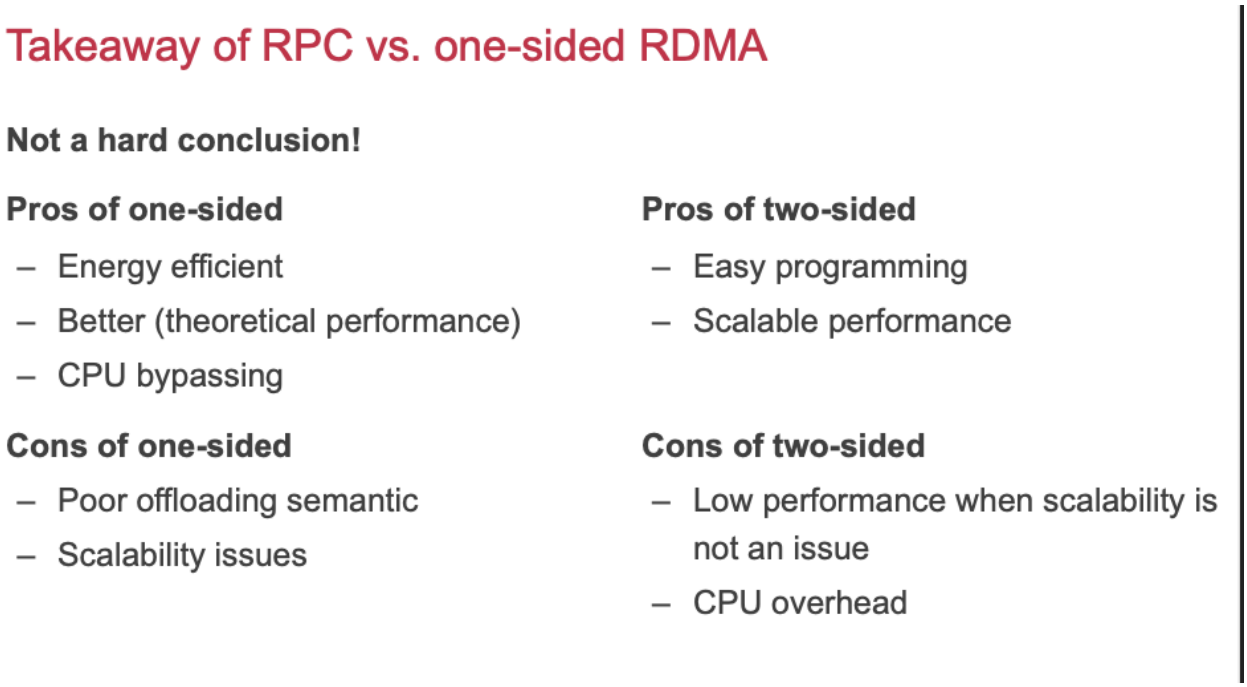
– ~ 1us additional latency, fetch from DRAM using PCIe read

– For comparison, RDMA one-sided read takes ~2us

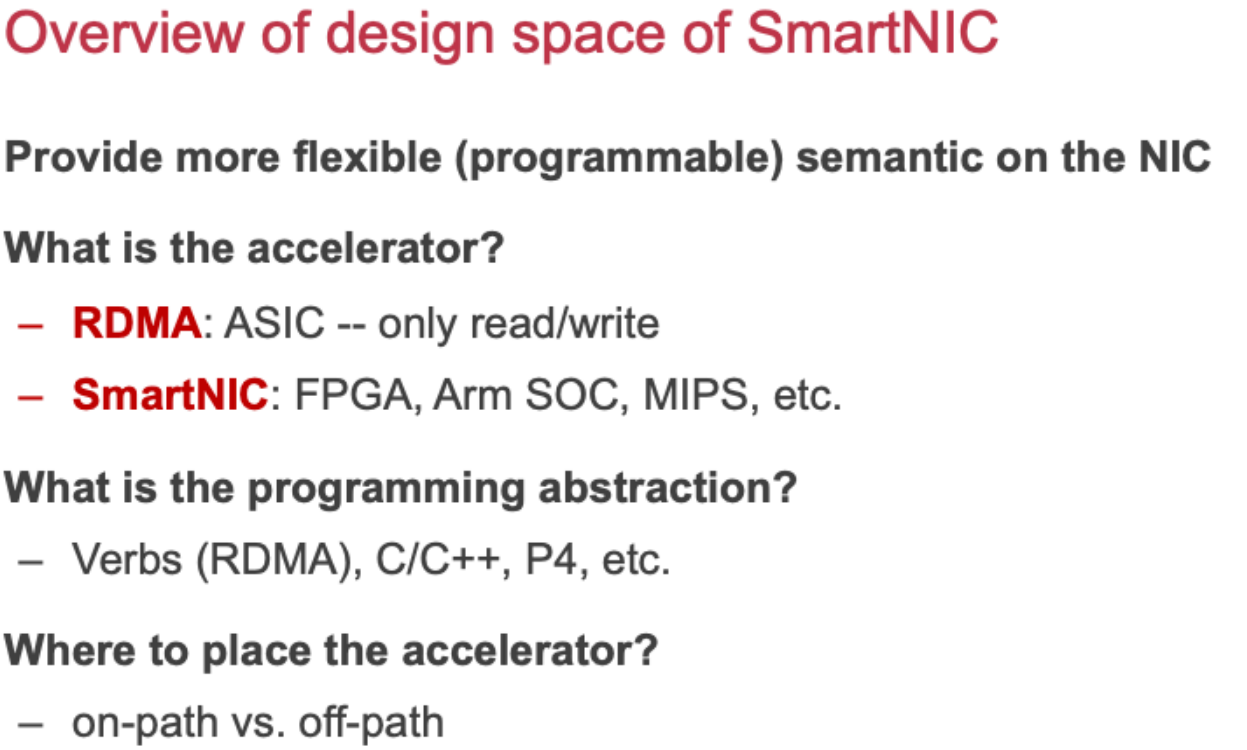
一些缓解方案：

- FaRM@NSDI'14: use huge page
  - Reduce #PTE entries
- LITE@SOSP'17 takes to another extreme; it directly physical memory, and uses kernel for security check
  - No PTE and keys need to be cached at the NIC!
- FaSST@OSDI16: 使用UD

- 减少QP，没有connection信息

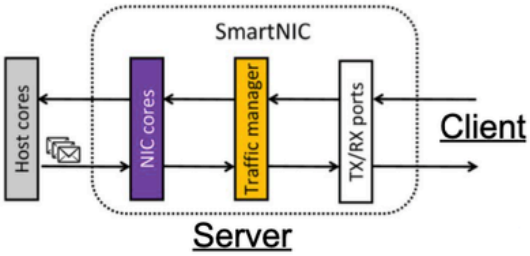


SmartNIC



On-path vs. off-path

On-path vs. off-path

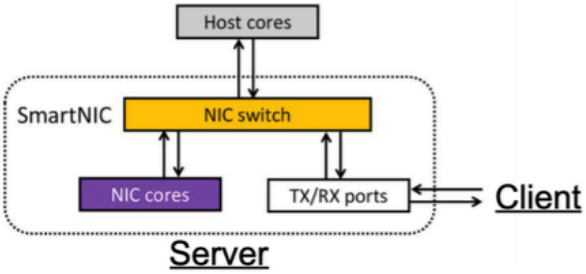


Pros

- Lower latency

Cons

- No resource isolations



Pros

- Clear resource isolation

Cons

- Latencies added due to NIC switch

92

FPGA vs. Processors

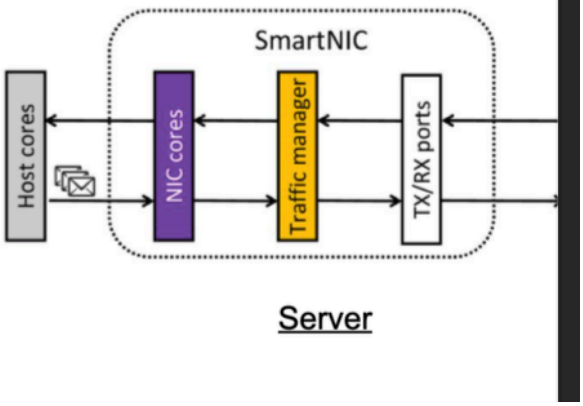
FPGA vs. Processors

The NIC cores can be

- FPGA
- ARM processors [1]

In the case of ARM, the NIC cores run a full OS

- E.g., Linux
- Represented RNICs: Bluefield, Stingray

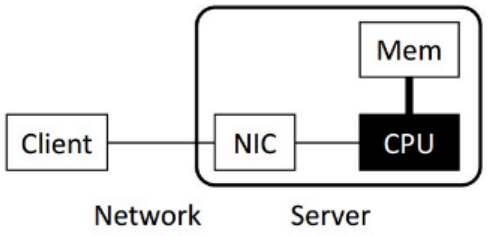


为什么智能网卡慢：

- 功耗限制
- 读取Host DRAM需要进过PCIe

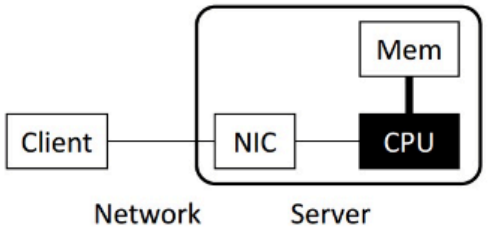
KV-Direct:

Distributed key-value architecture



Software (Kernel TCP/IP)

Bottleneck: Network stack in OS
(~300 Kops per core)

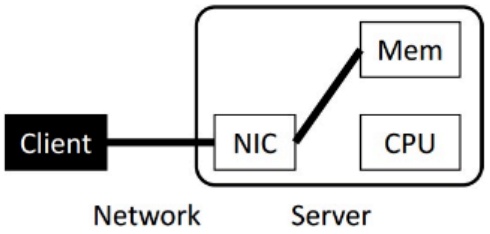


Software (Kernel Bypass)

e.g. DPDK, mtcp, libvma, two-sided RDMA

Bottlenecks: CPU random memory access and KV operation computation

(~5 Mops per core)

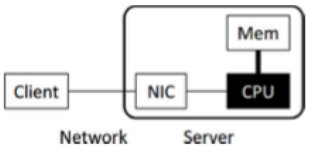


One sided RDMA

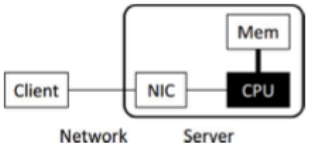
Communication overhead: multiple round-trips per KV operation (fetch index, data)

Synchronization overhead: write operations

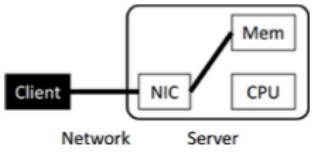
Distributed key-value architecture



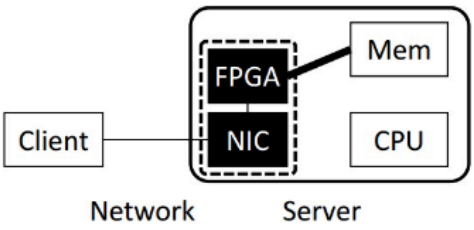
Software (Kernel TCP/IP)



Software (Kernel Bypass)



One sided RDMA



KV Direct FPGA + NIC

Offload KV processing on CPU to Programmable NIC

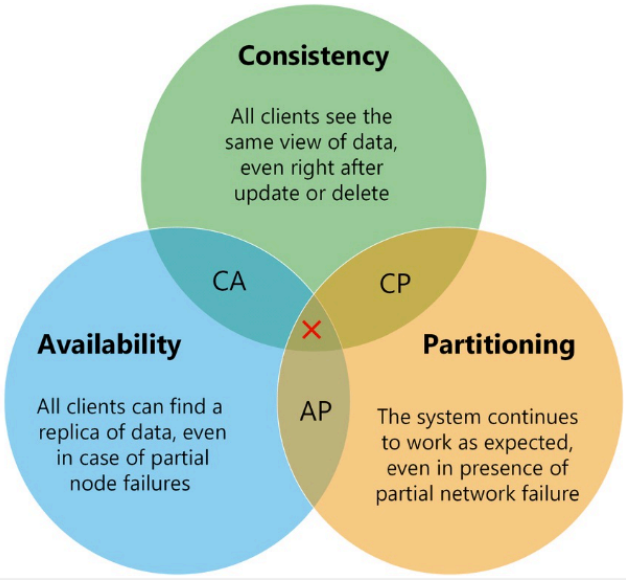
Fault Tolerance

Fault, Error, Failure

- **Fault can be latent or active**
  - If active, get wrong data or control signals
- **Error is the results of active fault**
  - E.g. violation of assertion or invariant of spec
  - Discovery of errors is ad hoc (formal specification?)
- **Failure happens if an error is not detected and masked**
  - Not producing the intended result at an interface

CAP theorem

- **CAP: pick two & trade-off**
- **CA**
  - Single machine
- **CP**
  - Hbase, Redis
- **AP**
  - Cassandra, CouchDB



Recovery恢复

Goal: 从crash 或者网络failure中恢复状态

正确性保证：未commit的continue or abort；已经commit的数据可以持久化

Solution: **Logging**

Write-ahead logging (WAL)：在更新数据状态之前，把log先写入磁盘；并且log是append-only的;

Structure of WAL Record

- **LSN**: unique log sequence number
- **Type**: operation (insert/delete/update)
- **After image**: new state
- **Before image**: old state
  - Insert location, delete location

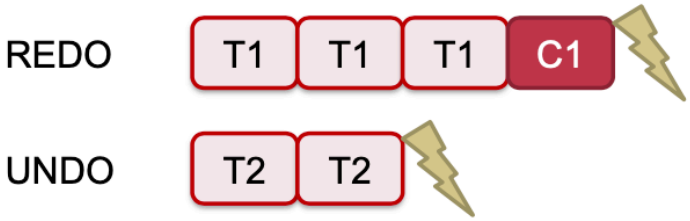
Checksum	LSN	Log Record Type	Transaction Commit Timestamp	Table Id	Insert Location	Delete Location	After Image

在Commit的时候，添加一个特殊的Log：CMT (Commit Log)

- Commit log (CMT)

Recovery Rules:

- **Recovery rules**
  - Travel from end to start
  - Mark all transaction's log record w/o CMT log and append ABORT log
  - REDO: committed
  - UNDO: uncommitted/aborted



日志与并发事务：

- 单个log queue
- 多个log queue

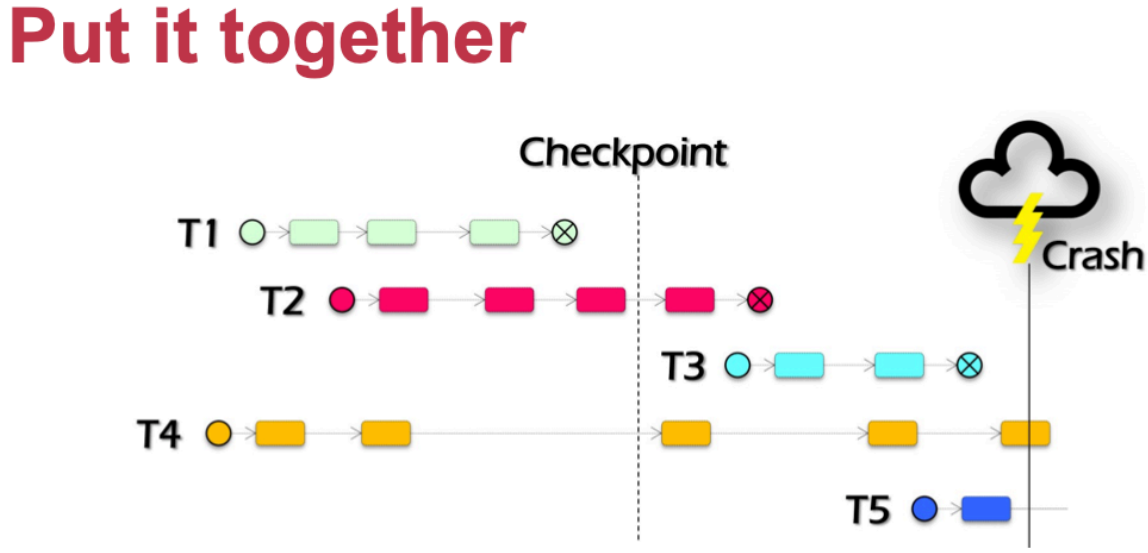
性能考虑：

- 局部性（log序列化考虑Sequential IO）
- 并发控制（会增加tail latency）

我们为什么需要flush disk:

- No need to recovery from a blank state
- Avoid aborting long transaction

刷磁盘的频率：不是只在最后的时候把log刷入磁盘——checkpoint



T1: do nothing; T2: redo; T3: redo; T4: undo; T5: undo

WBL (write-behind logging)

WBL: write-behind logging

- **Reduce data duplication**
  - flushing changes to the database in NVM during regular transaction processing
- **When insert a tuple**
  - Write data *before* meta-data in the log
- **Log is behind the contents of the database**

Primary-backup 主从备份

Fault tolerance机制：

- Replicated machins
  - Fully replication of data
- Log
  - Synchronize data updating