架构融合
云化共建

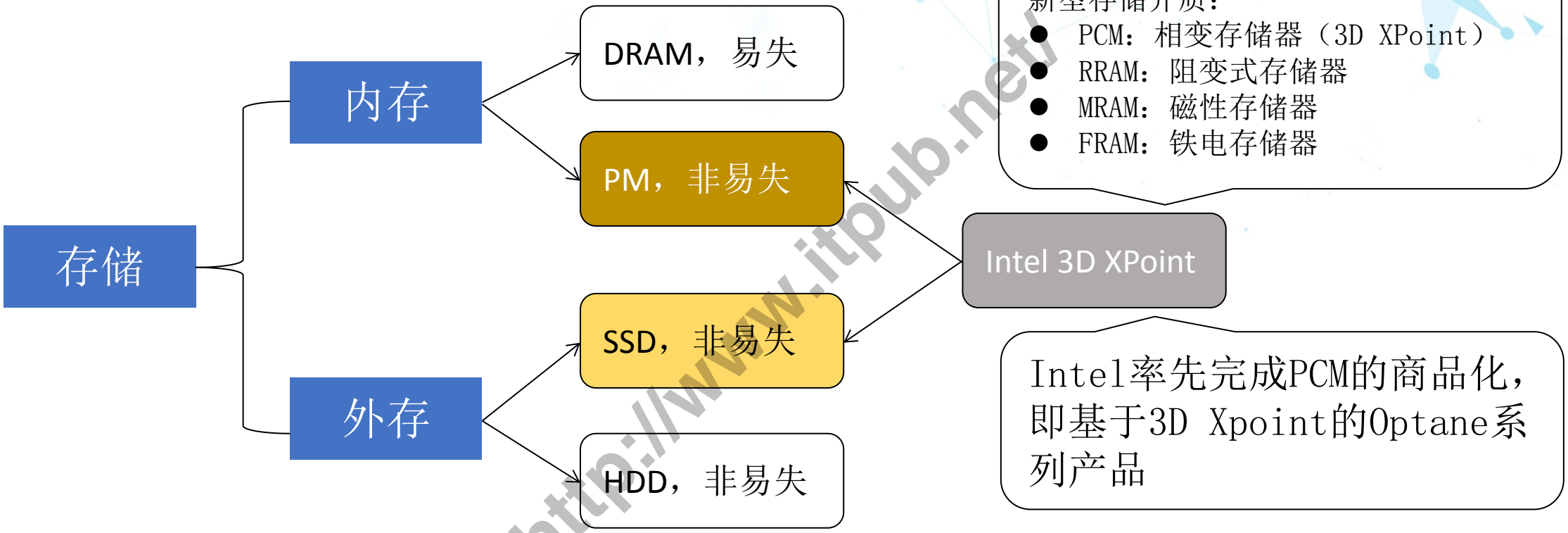
持久内存在分布式存储中的应用

韩银俊

2020.10

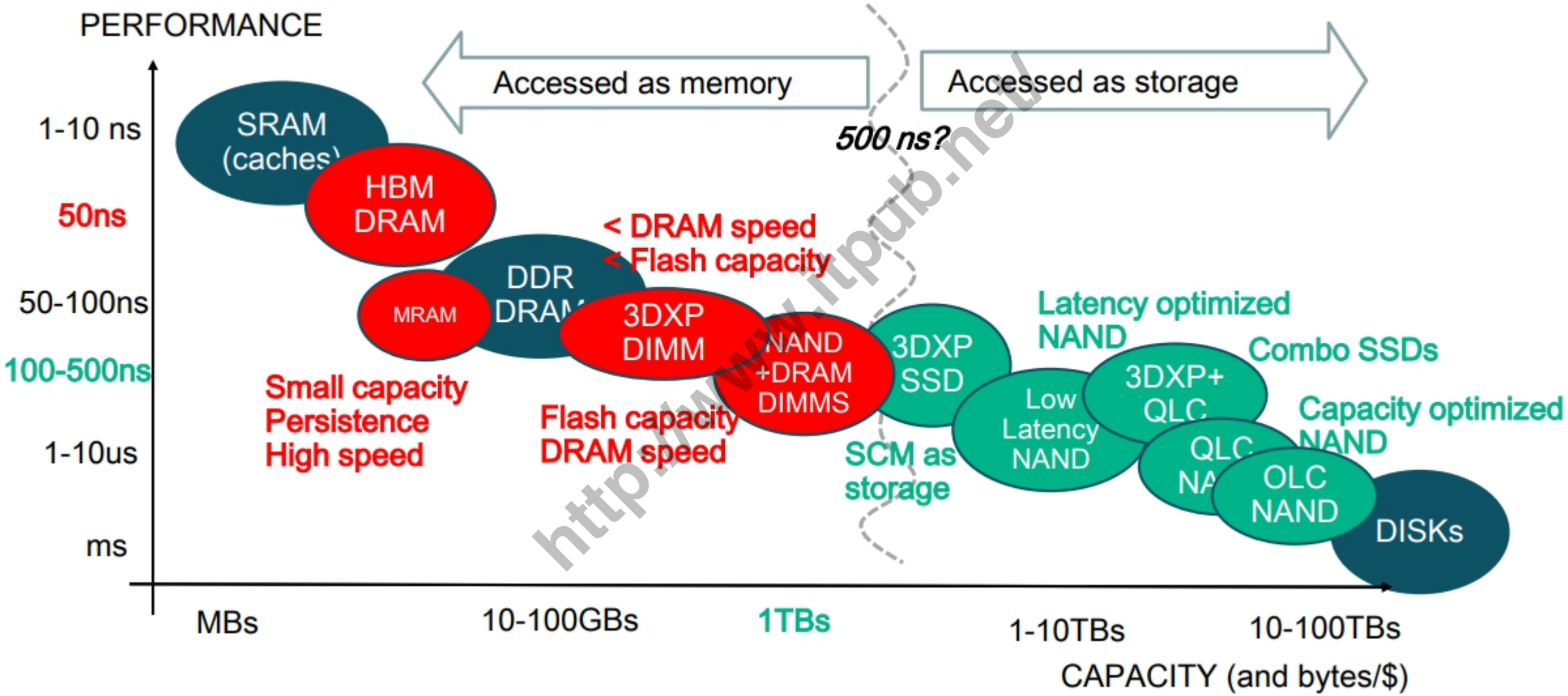
目录

背景



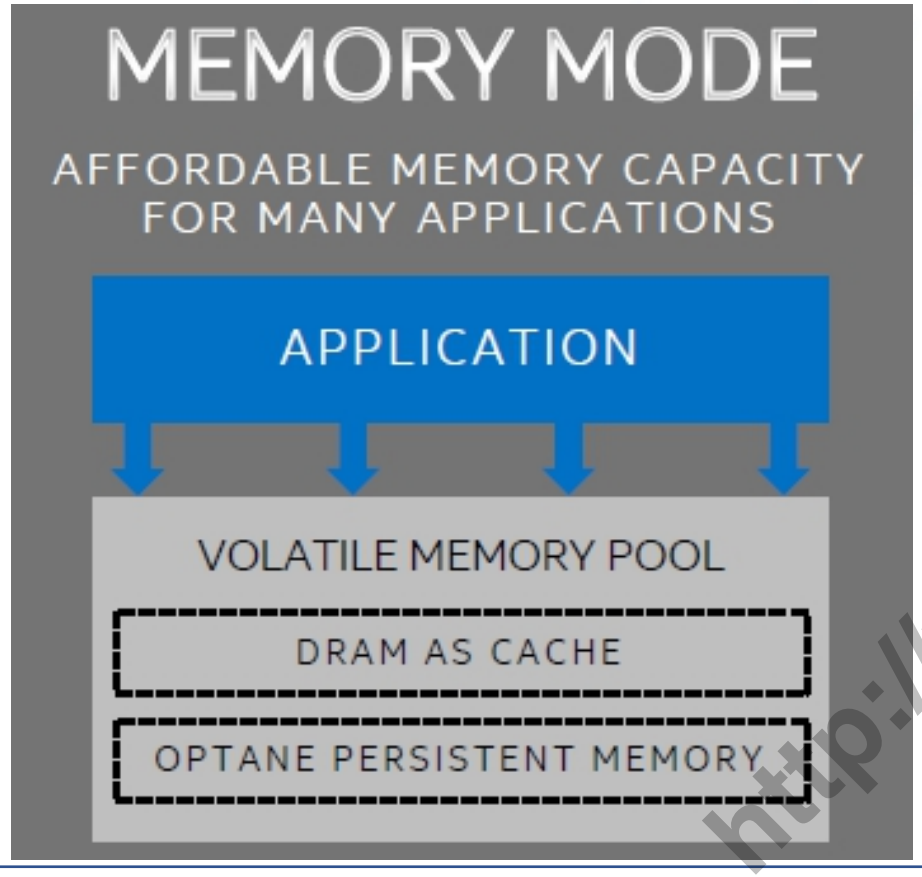
PM持久内存是一种新的存储介质，模糊了内存和外存的界限，将对上层软件架构和存储生态产生重大影响。

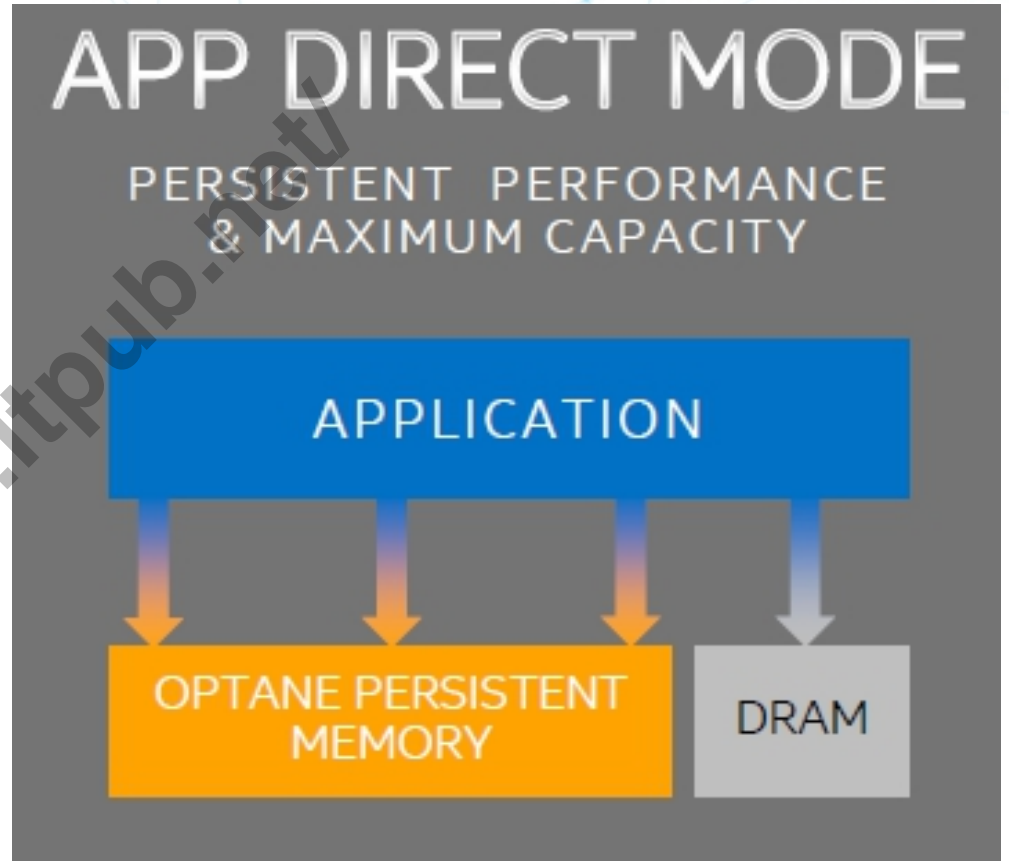
存储介质层次结构



Source: P. Faraboschi, HPE, “The Data Access Continuum”, SC’19 MCHPC

pmem使用模式

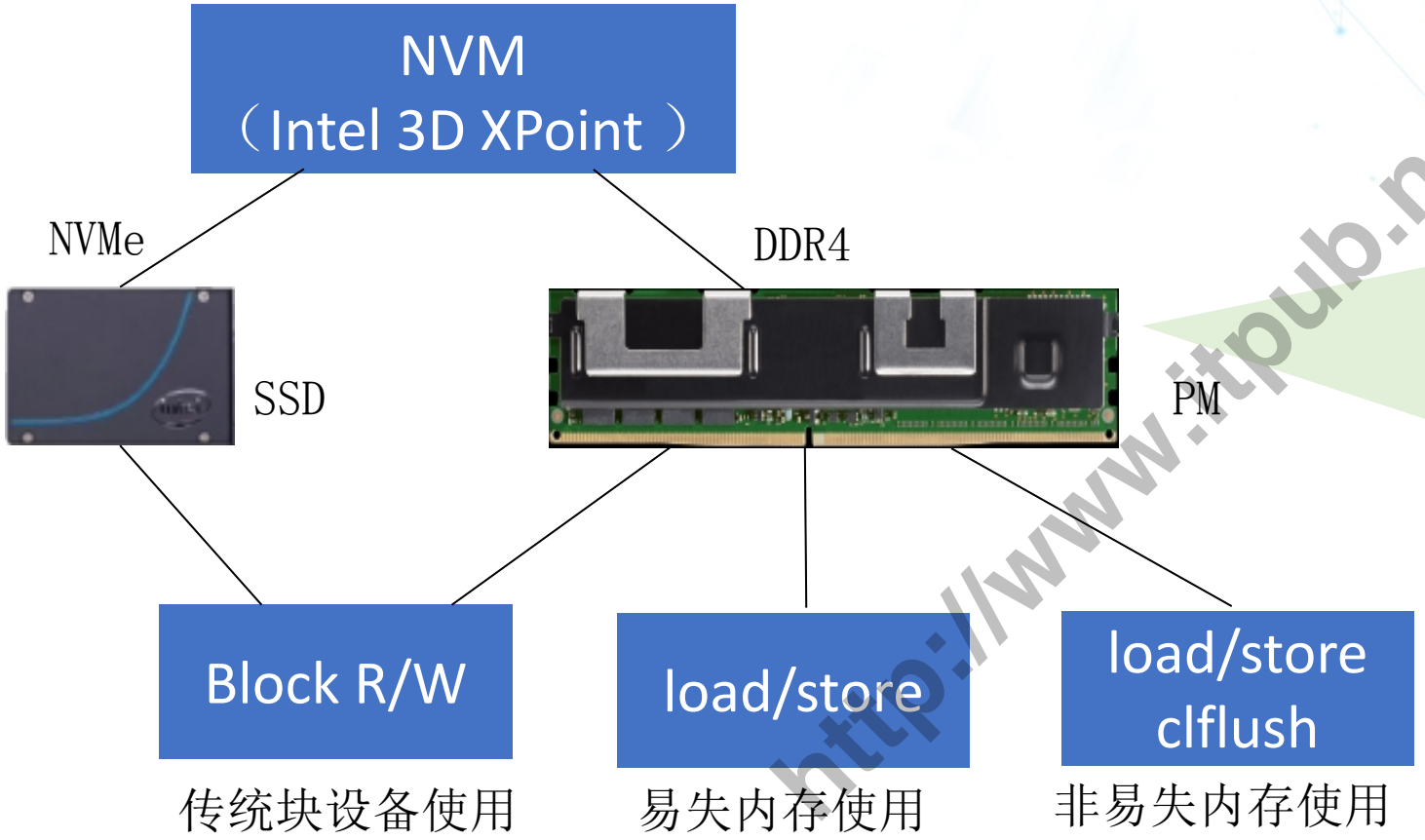




内存模式：PM作为易失内存使用，DRAM作为PM的高速缓存，内存有效容量为PM的容量，应用不需要做任何修改。

APP Direct模式：PM作为非易失或易失内存使用，DRAM为易失内存，内存有效容量为PM和DRAM容量的总和，应用需要做定制优化。

NVM使用形态



一：PM特性：

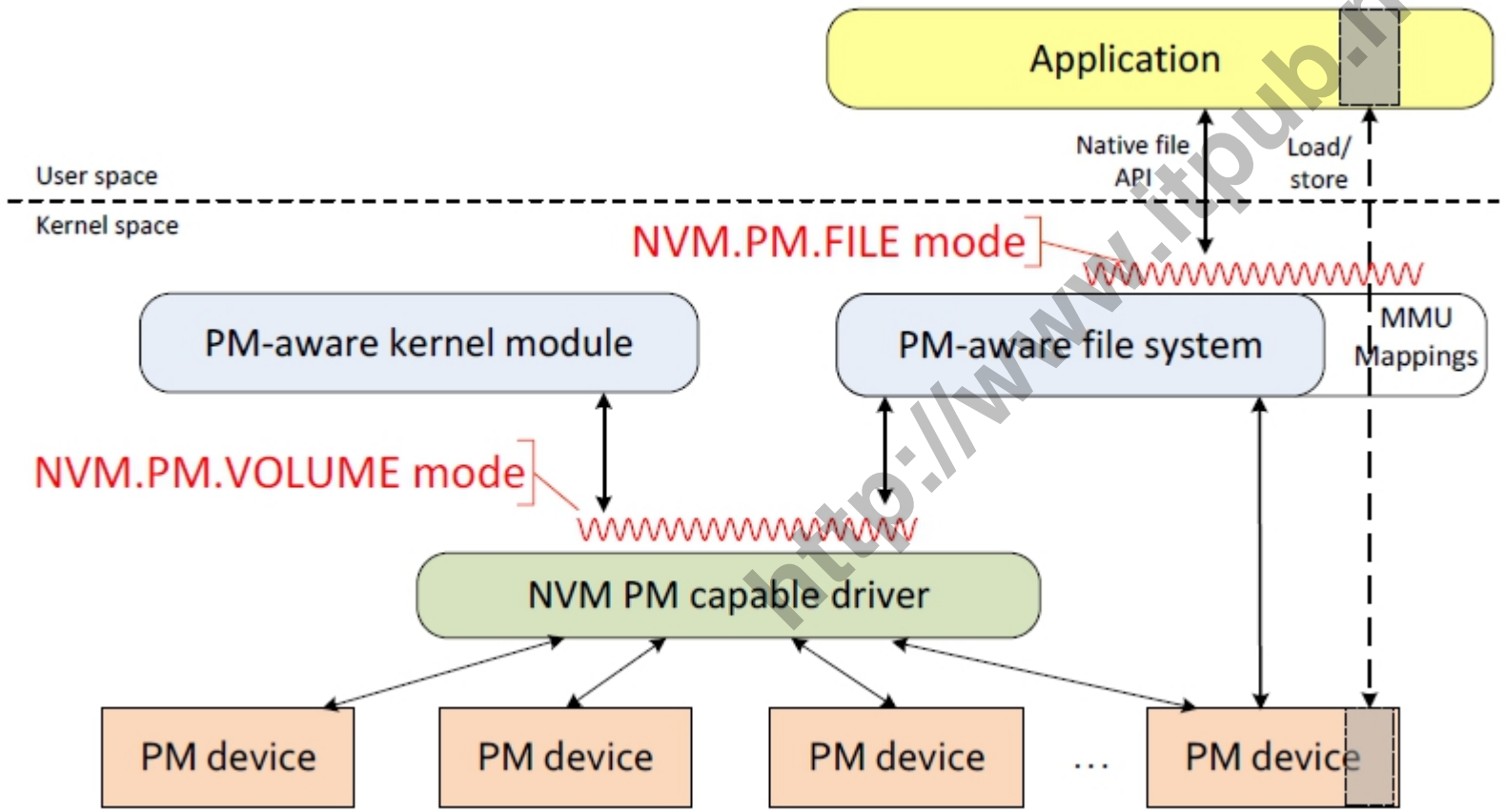
- 性能接近DRAM
- 像SSD一样掉电不丢数据
- 内存接口，字节寻址
- 容量比DRAM大一个数量级，单条达512G

二：相似概念：

- NVM：Non-volatile memory ，非易失内存
- SCM：Storage Class Memory，存储级内存

SNIA NVM编程模型

- 存储网络工业协会（SNIA）2017.06发布了NVM编程模型V1.2版本，里面详细描述了NVM编程模型。

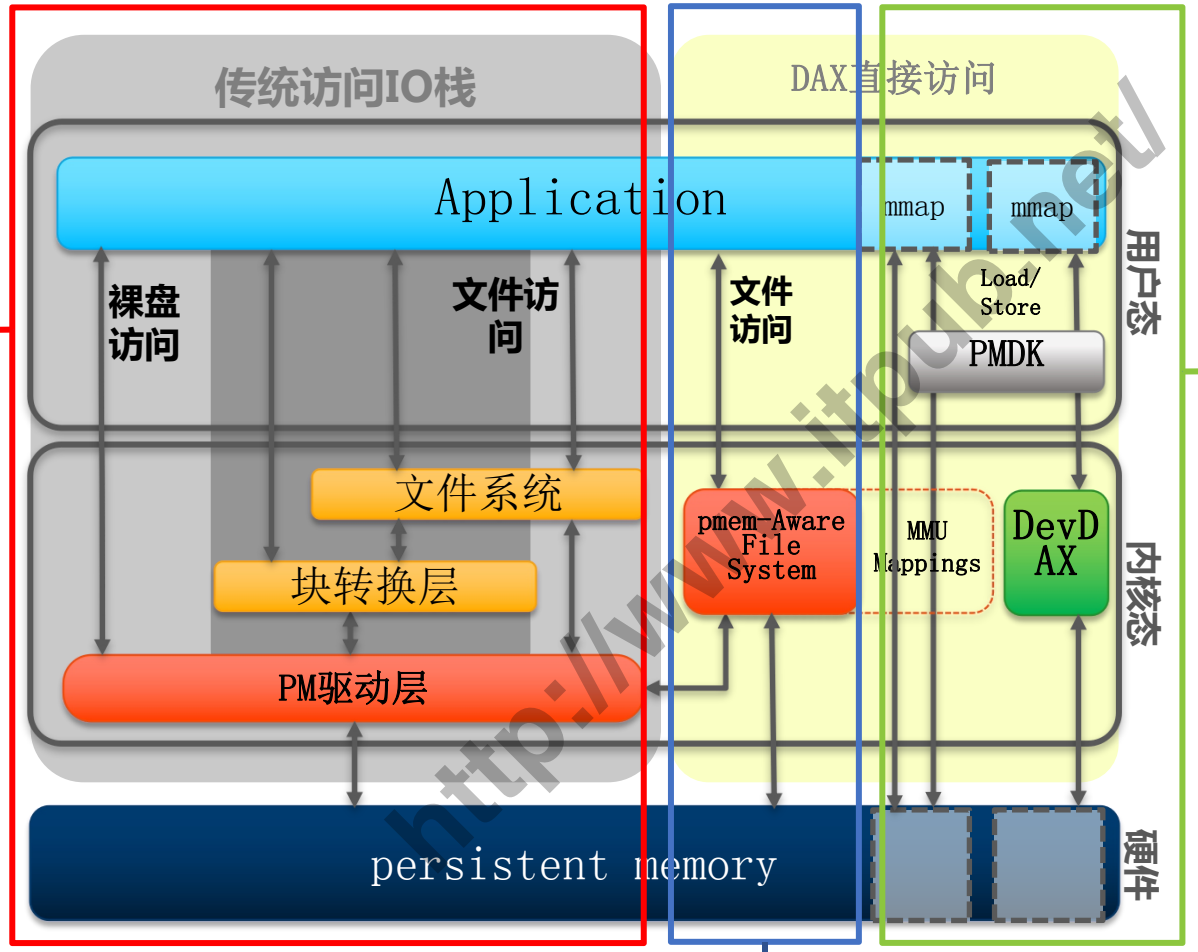


- 该规范定义了各种用户空间和支持NVM的操作系统（OS）内核组件之间的建议行为。
- 使常见的NVM行为能够由多个操作系统特定的接口公开。

Source: https://www.snia.org/tech_activities/standards/curr_standards/npm

pmem 应用的IO栈

PM当作SSD使用，应用不需要做修改



应用针对PM做定制优化，作为持久内存使用。
基于CPU持久化指令或基于PMDK进行开发，这种方式充分能发挥PM特性，性能最优。

PM当作SSD使用，基于支持DAX的文件系统，应用不需要做修改

持久内存使用面临的挑战

数据持久性

由于CPU缓存的存在，数据在flush到PM之前，存储不保证是持久性的

故障原子性

x86架构上只保证8个字节是原子的，需要通过事务保证故障原子性

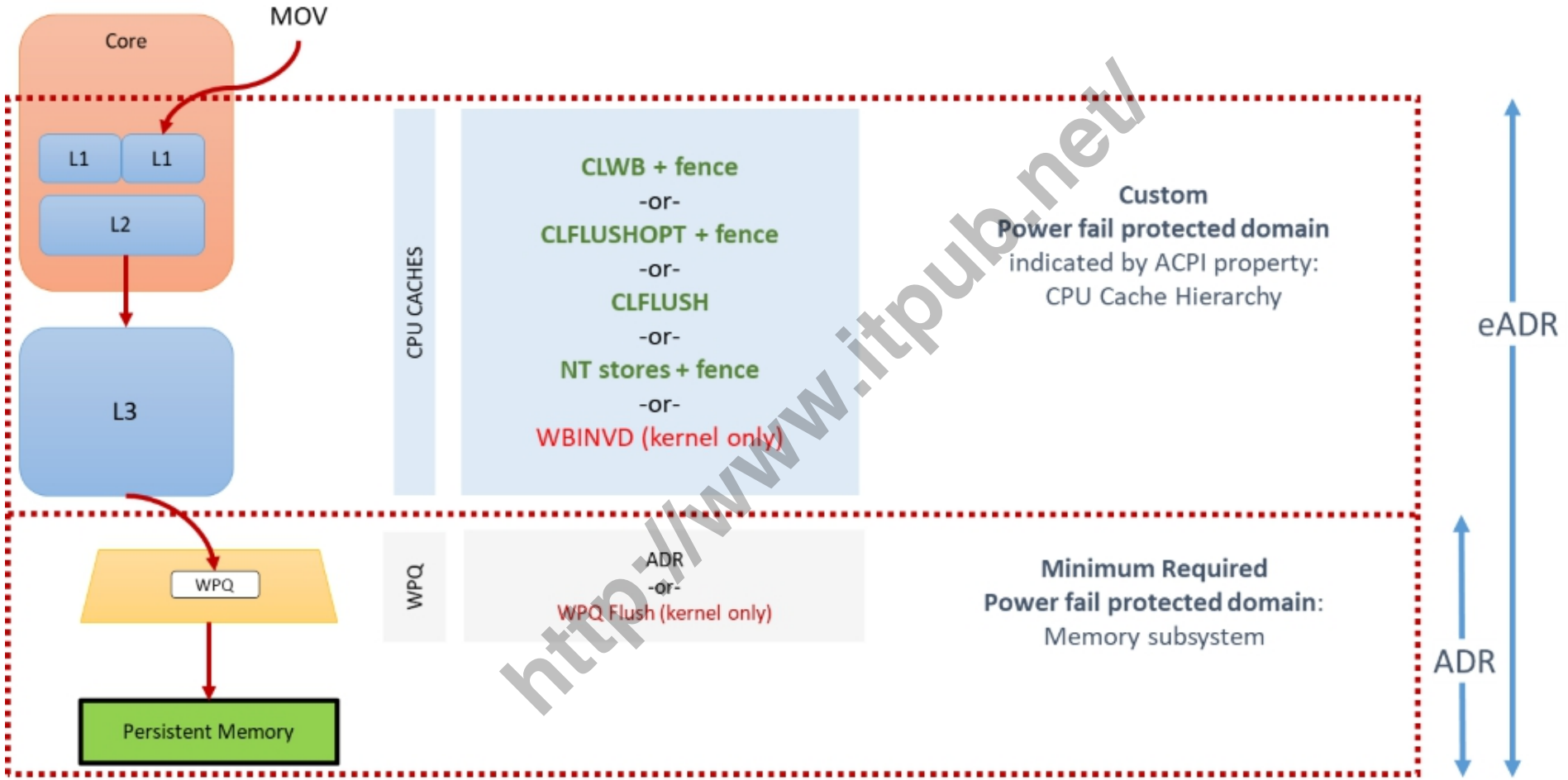
内存泄漏

内存泄漏到PM是持久的，重启无法恢复

与DRAM配合

PM性能与DRAM存在差距，但是具有非易失性，需要跟DRAM合理配合使用

ADR(Asynchronous DRAM Refresh)



ADR和eADR电源故障保护域

用于持久内存的x86缓存刷新指令

指令	说明
CLFLUSH	刷新单个Cache Line，串行执行，缺少并发性
CLFLUSHOPT (followed by an SFENCE)	为支持持久内存而新引入的，与CLFLUSH类似，但支持并发执行
CLWB (followed by an SFENCE)	与CLFLUSHOPT类似，但是CPU缓存中还保留，下次访问时可以命中
NT stores (followed by an SFENCE)	绕过CPU缓存，直接写内存

8字节故障原子性

```
uint64_t v = 0;
...
v = 1024;
pmem_persist(&v, 8);
```

v = ?
- 0
- 1024

```
strcpy(pmem, “Hello,World!”);
pmem_persist(pmem, 14);
```

pmem =
- “\0\0\0\0\0\0\0\0\0\0...”
- “Hello,W\0\0\0\0\0\0...”
- “\0\0\0\0\0\0\0\0ord!\0”
- “Hello,World!\0”

...

可见性与持久性

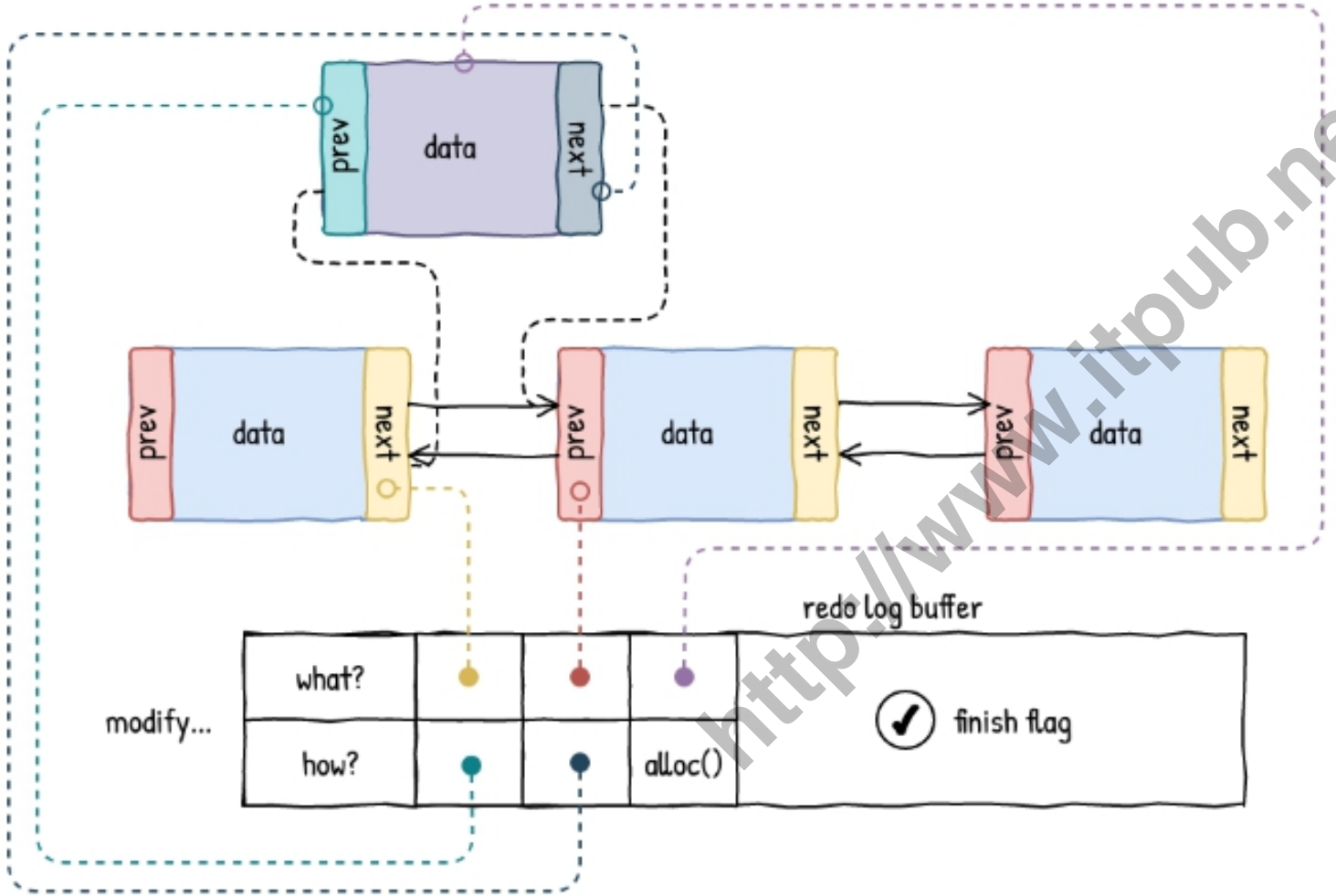
```
//thread 1
//initial value of pmem->a is 0
atomic_store(&pmem->a,1);   //visible=1, persistent=?
pmem_persist(&pmem->a);     //visible=1, persistent=1

//thread 2
//initial value of pmem->b is 0
if (atomic_load(&pmem->a) == 1)
{
    pmem->b=1;                      // visible=1, persistent=?
    pmem_persist(&pmem->b);  // visible=1, persistent=1
}
```

Possible persistent values of (a,b) = (0,0), (1,0), (1,1), (0,1)

逻辑错误

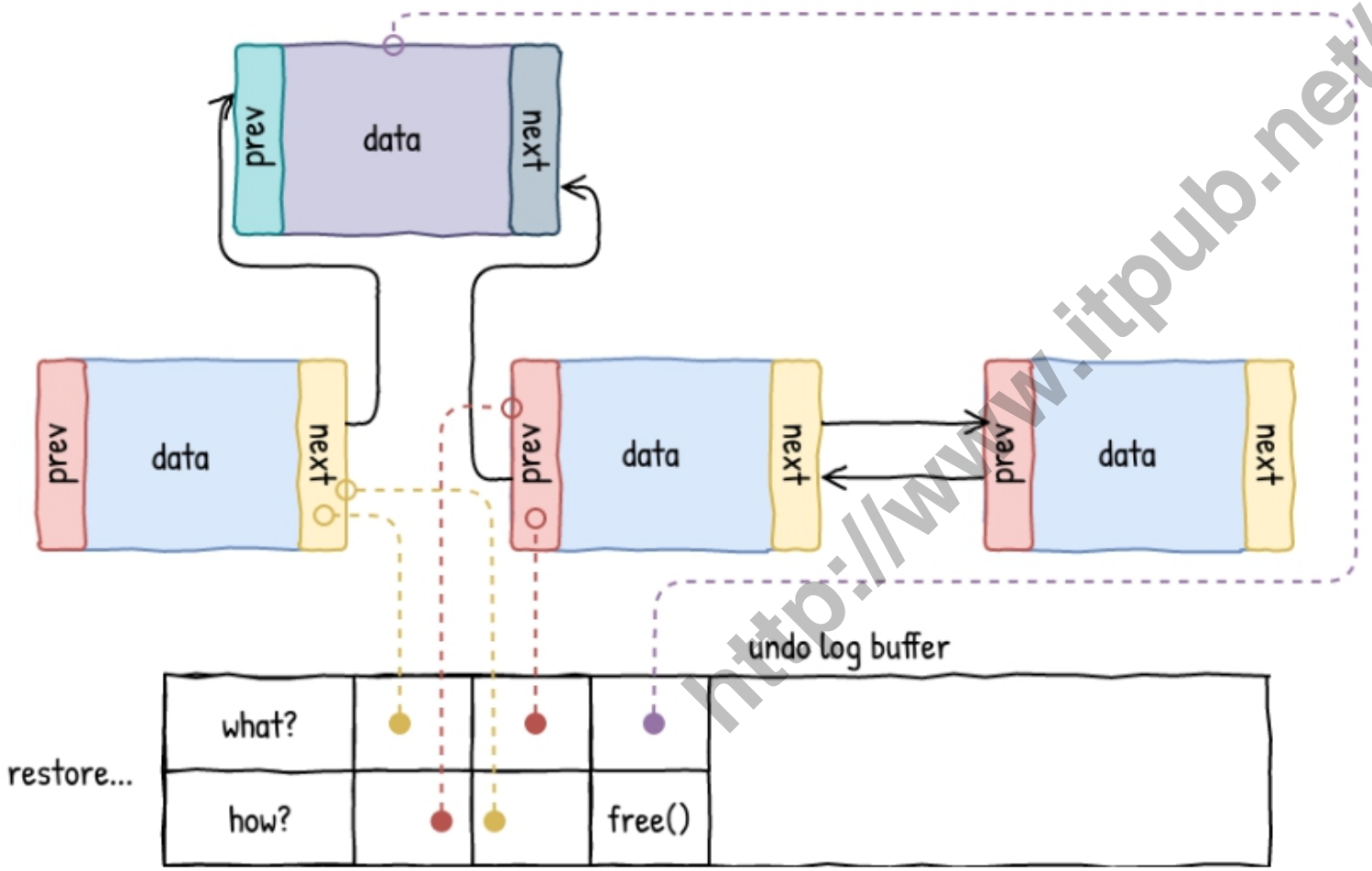
ACID事务保障-REDO



1）分配新的Node，填充数据和修改前后指针
2）修改前一节点next指向新节点
3）修改后一节点prev指向新节点

修改数据结构之前，先写REDO

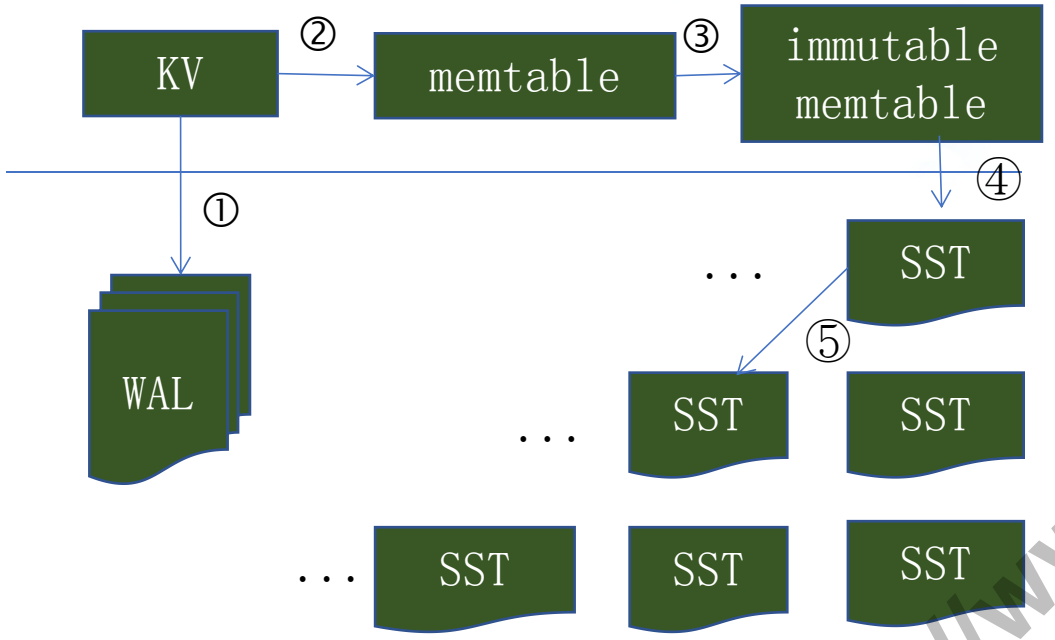
ACID事务保障-UNDO

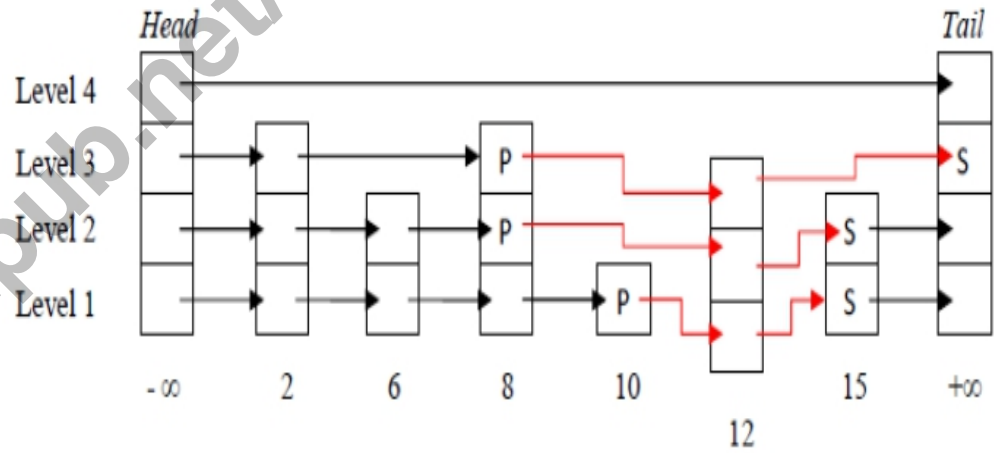


数据结构修改之前，先创建内存区域快照，事务异常终止时，恢复原先的数据

基于UNDO 可以立即看到数据的修改
基于REDO 数据修改不会立即可见

存储架构适配





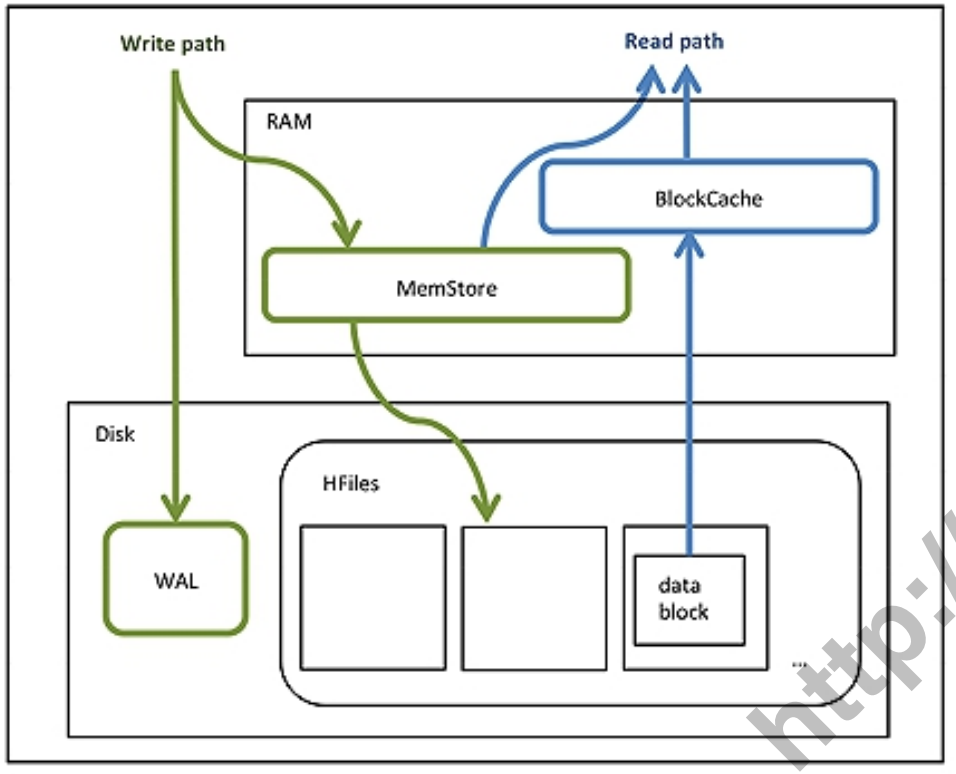
传统方案：先写顺序WAL日志，再随机持久化数据；随机转顺序。

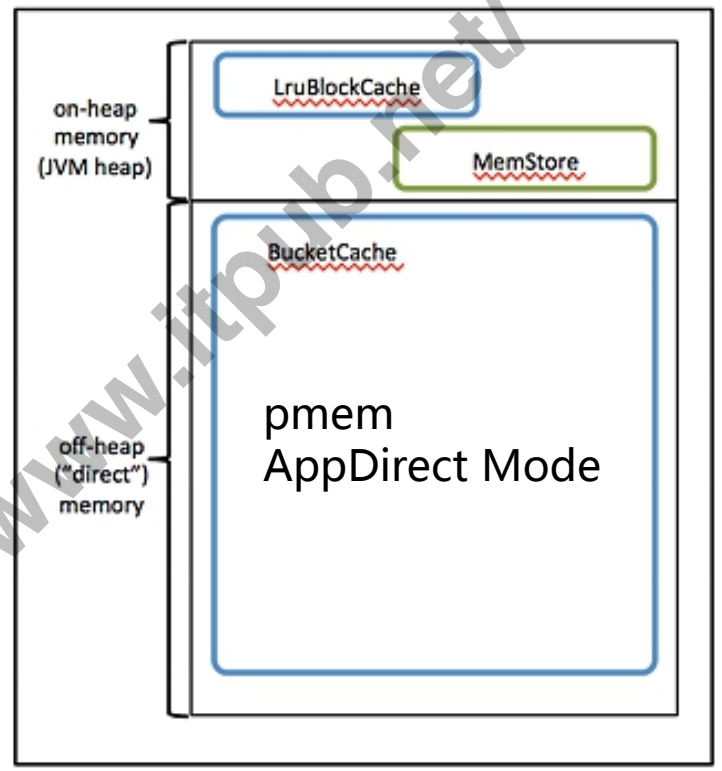
背后假设：1）内存快而易失，外存慢而持久；
2）外存顺序写性能远好于随机写。

pmem特性：1、PM内存本身是非易失的；
2、PM上随机写和顺序写性能差异很小

改进方案：使用无锁并发持久内存数据结构，或仅记录少量元数据事务日志

把pmem当作大内使用(Hbase BucketCache)





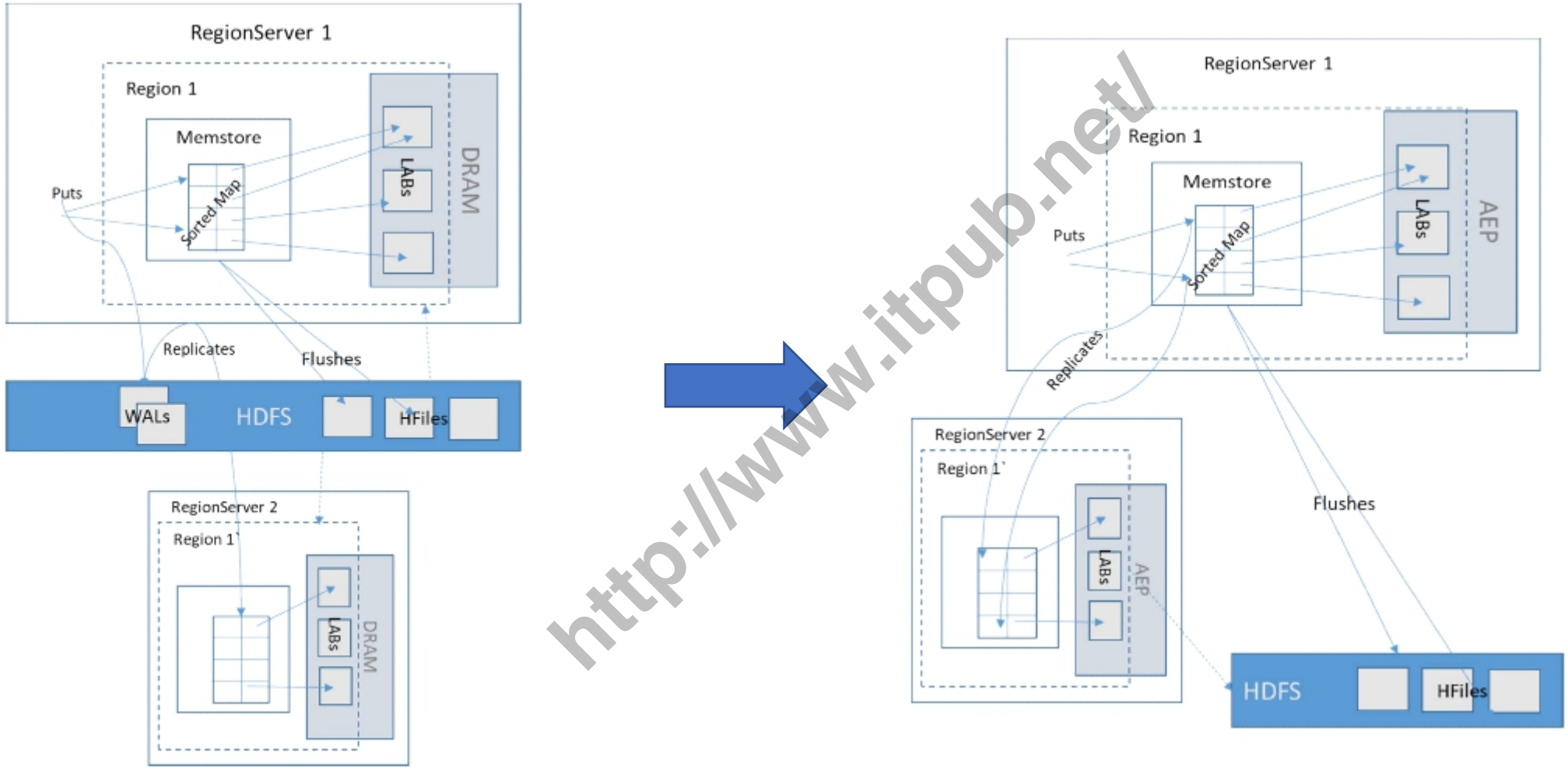
并不利用pmem的持久性，只是利用大内存特性

在内存命中的情况下，pmem性能达到DRAM性能的94%

source: https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.1.4/hbase-data-access/content/overview-hbase-io.html

issue：https://issues.apache.org/jira/browse/HBASE-21874　Status：Resolved

使用pmem移除HBase的WAL



Source: https://yq.aliyun.com/download/2919

Issue: https://issues.apache.org/jira/browse/HBASE-20003 Status: Open

数据存放策略

类型	DRAM	PMEM	NVMe SSD	HDD/SATA SSD
特性	低时延，<100ns 易失，字节寻址 随机访问性能好	低时延，<1us 非易失，字节寻址 随机访问性能好	时延 <10us 块接口 适合批量更新	时延 <2ms 块接口 适合顺序访问
使用方式	索引及数据的读写缓存	存放分布性存储元数据信息或事务日志的缓存，并异步更新到SSD	存放元数据，事务日志数据，热点数据	存放冷数据
根据PMEM，NVMe SSD和HDD不同的存储介质的不同物理特性，因地制宜的组织数据存放				

pmem在分布式存储中应用实践

pmem应用：

1）元数据：数据分布，对象索引，对象属性，空闲空间管理等

2）事务日志：Redo，Undo

3）数据缓存：小数据存储，数据合并

4）数据分层：热点数据存储

5）远程复制：PMoF

总结

- pmem性能比DRAM差，时延达300ns是DRAM的3倍，写带宽实测不足2GB,对性能敏感的应用，不能直接拿来当内存使用
- pmem由于存在缓存丢失问题，所以开发人员需要关注数据的可见性和持久性问题
- pmem提供了字节寻址和持久性，所以在传统存储为了保证事务特性而引入的写放大等优化不适用于pmem
- pmem在分布式存储中更适合用于存储元数据信息，事务日志信息，和热点数据的缓存

Thanks