Linux Kernel Extensions
for Databases

Alexander Krizhanovsky

Tempesta Technologies, Inc.

ak@tempesta-tech.com

Who am I?

- CEO & CTO at **NatSys Lab** & **Tempesta Technologies**
- **Tempesta Technologies** *(Seattle, WA)*
  - Subsidiary of NatSys Lab. developing **Tempesta FW** – a first and only hybrid of HTTP accelerator and firewall for DDoS mitigation & WAF
- **NatSys Lab** *(Moscow, Russia)*
  - Custom software development in:
    - high performance network traffic processing
    - databases

Tempesta
Technologies

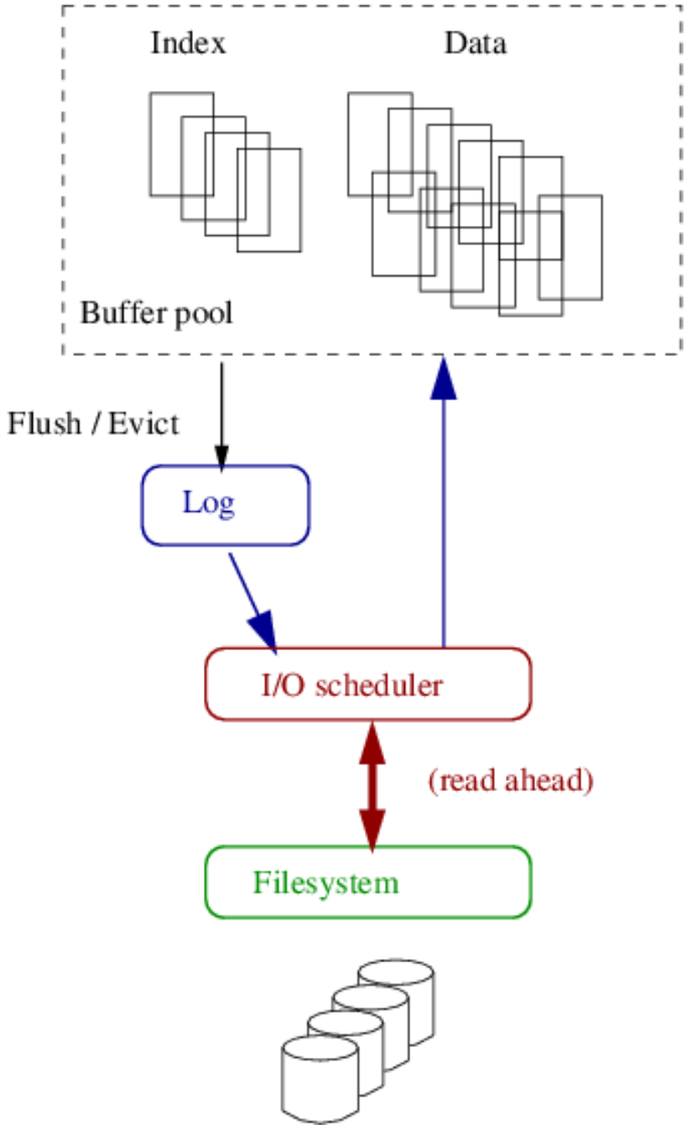
The begin

(many years ago)

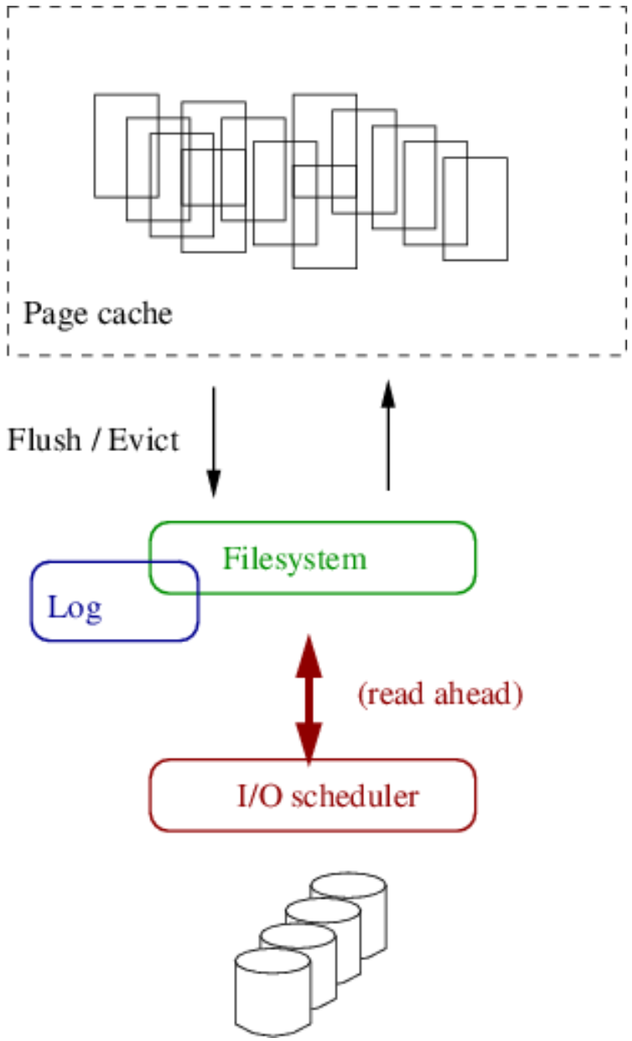
- **Database** to store Instant messenger's history
- **Plenty of data** (NoSQL, 3-touple key)
- High **performance**
- Some **consistency** (no transactions)
- 2-3 months *(quick prototype)*

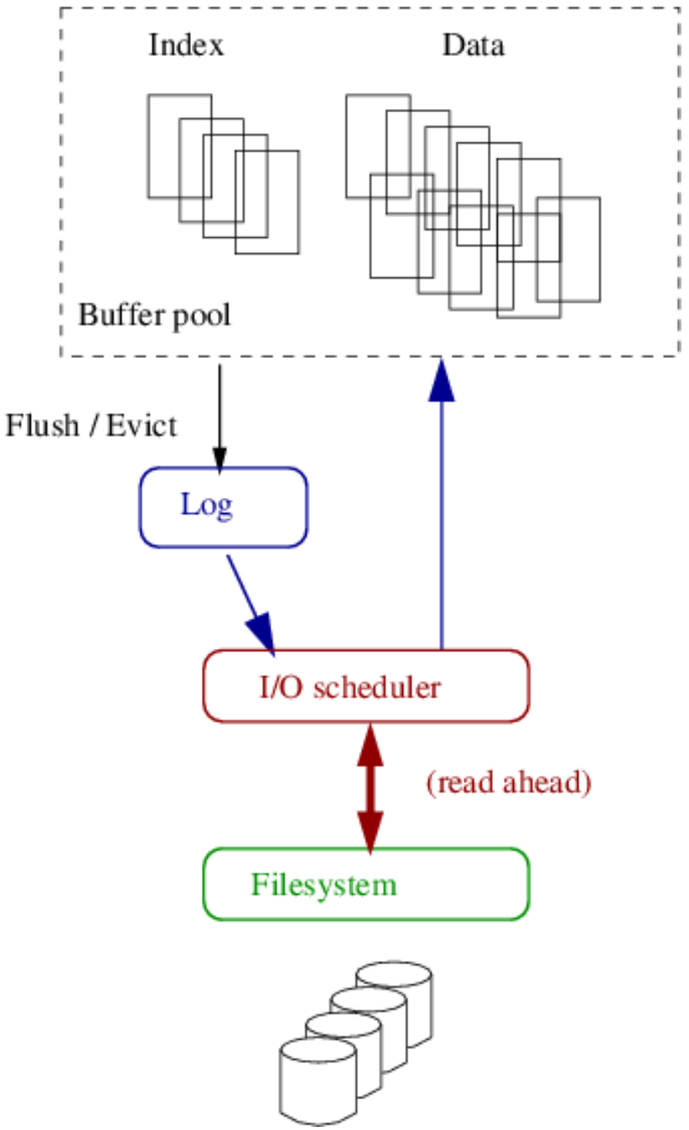
Simple DBMS

- *Disclamer:*
  memory & I/O only,
  no index,
  no locking,
  no queries
- *“DBMS” means*
  InnoDB



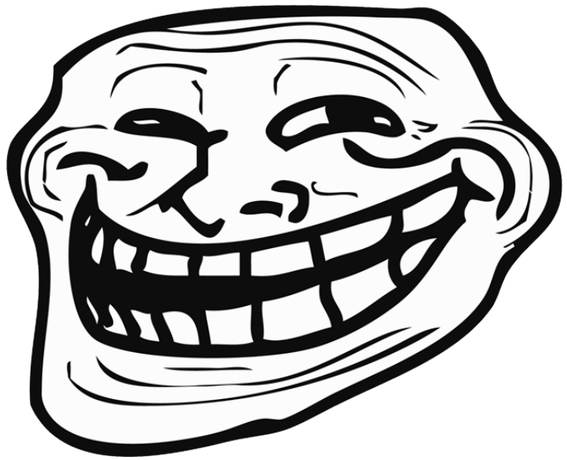
Linux VMM?


Page cache
Flush / Evict
Filesystem
Log
(read ahead)
I/O scheduler


Index
Data
Buffer pool
Flush / Evict
Log
I/O scheduler
(read ahead)
Filesystem

open(O_DIRECT): OS kernel bypass

«In short, the whole "let's bypass the OS" notion is just fundamentally broken. It sounds simple, but it sounds simple only to an idiot who writes databases and doesn't even UNDERSTAND what an OS is meant to do.»



Linus Torvalds

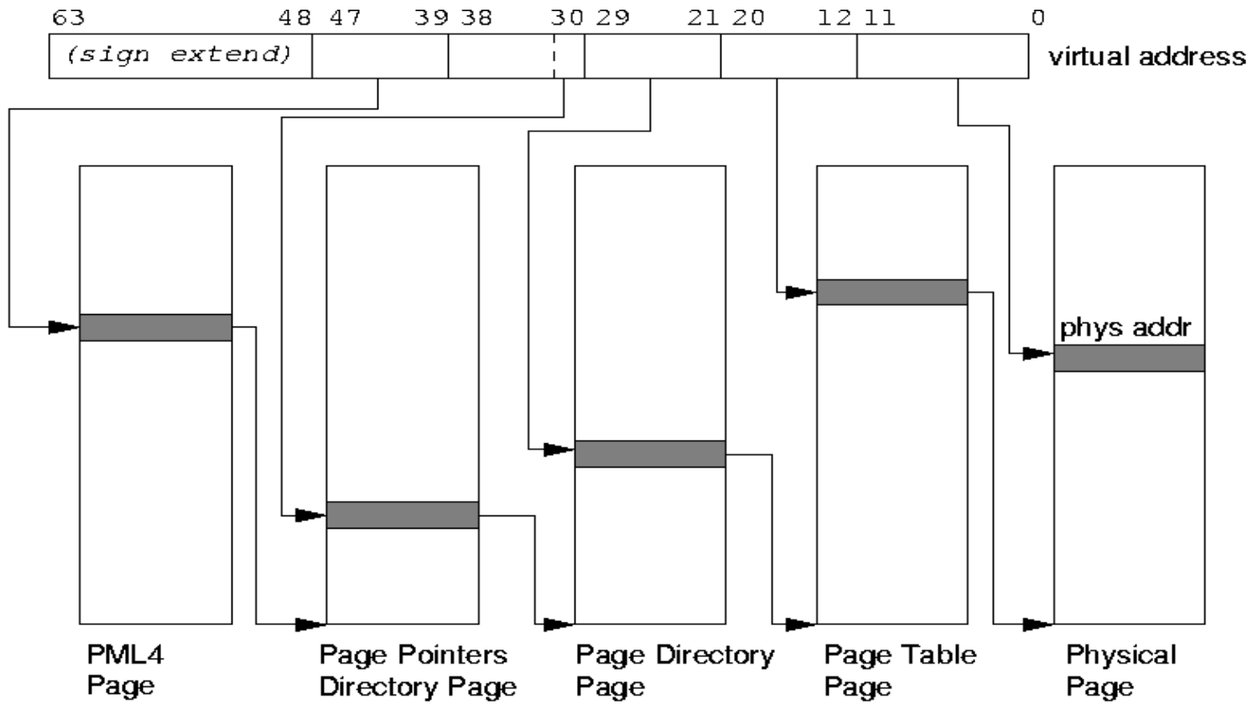
«Re: O_DIRECT question»

https://lkml.org/lkml/2007/1/11/129

mmap(2)!

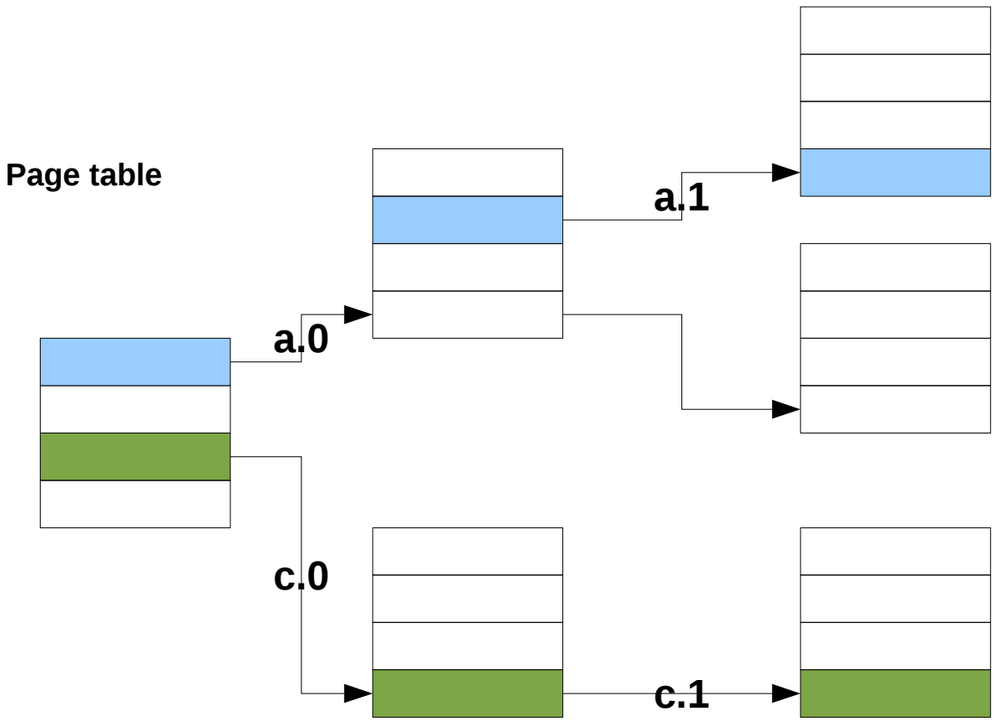
- Automatic page eviction
- Transparrent persistency
- I/O is managed by OS
- ...and ever **radix tree** index for free!

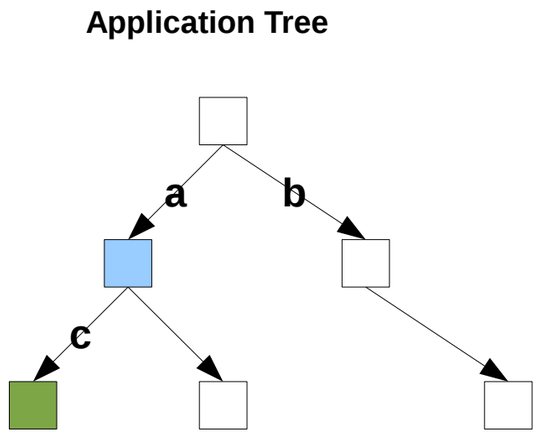
x86-64 page table (radix tree)



Tempesta Technologies

A tree in the tree





mmap(2): index for free

```
$ grep 6f0000000000 /proc/[0-9]*/maps
$
```

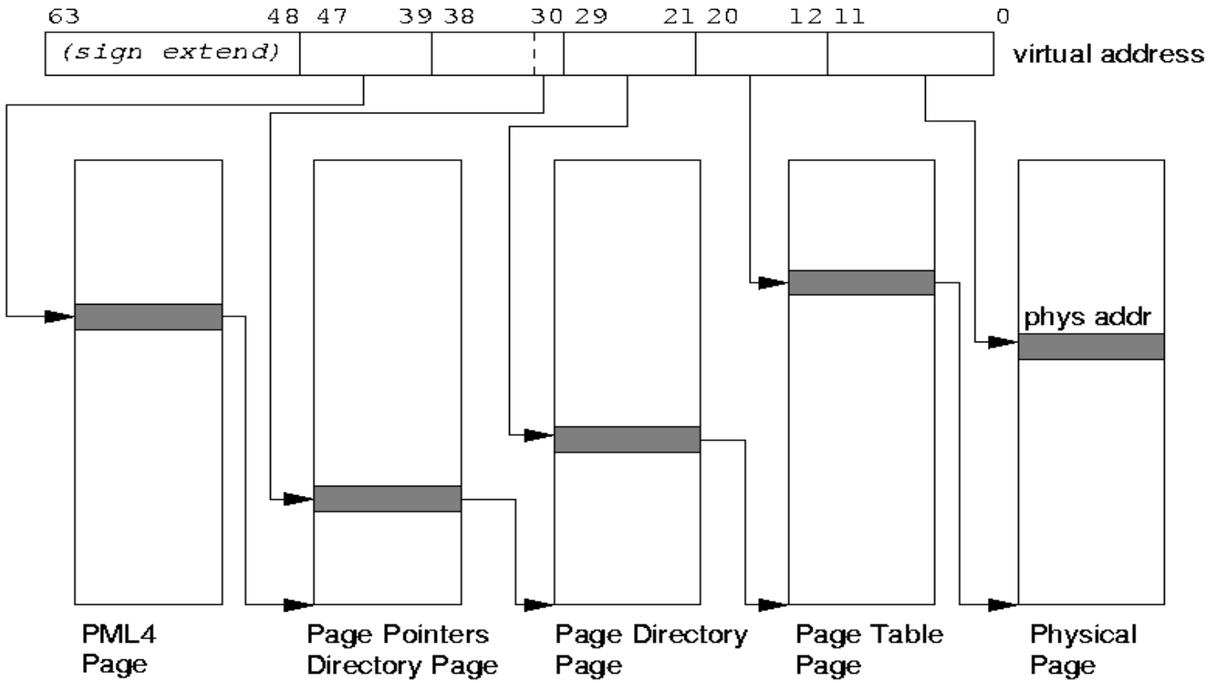
```
DbItem *db = mmap(0x6f0000000000, 0x40000000 /* 1GB */, ...);
DbItem *x = (DbItem *)(0x6f0000000000 + key);
```

...or just an array

```
DbItem *db = mmap(0, 0x40000000 /* 1GB */, ...);
DbItem *x = &db[key];
```

Virtual memory isn't for free

- TLB cache **is small** (~1024 entries, i.e. 4MB)
- TLB **cache miss** is up to 4 memory transfers
- **Spacial locality** is crucial: 1 address outlier is up to 12KB
  …but Linux VMM coalesces memory areas
- **Context switch** of *user-space processes* invalidates TLB
  ...but threads and user/kernel context switches are cheap



Tempesta Technologies

Lesson 1

- Large mmap()'s are expensive
- Spacial locality is your friend
- Kernel mappings are resistant to context switches

DBMS vs OS

- Stonebreaker, *"Operating System Support for Database Management"*, **1981**
  - OS buffers (pages) force out with **controlled order**
  - Record-oriented FS (**block != record**)
  - Data consistency control (**transactions**)
  - FS blocks **physical contiguity**
  - Tree structured FS: **a tree in a tree**
  - Scheduling, process management and IPC

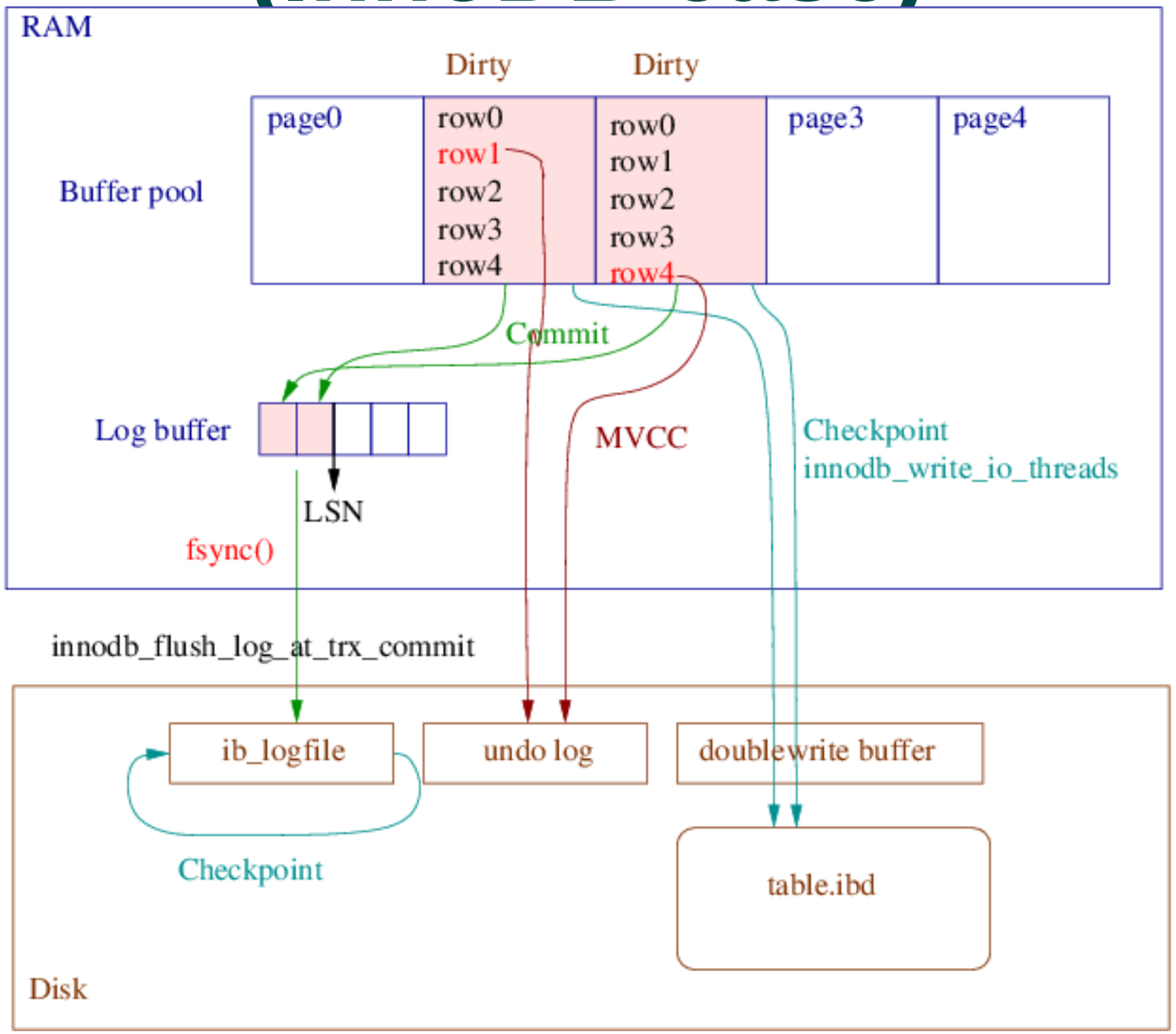
Filesystem: extents

- Modern Linux filesystems: BtrFS, EXT4, XFS
- **Large contigous** space is allocated at once
- **Per-extent addressing**

Lesson 2

- There are no (or small) file blocks fragmentation
- There are no trees inside extent
- *fallocate(2)* became my new friend

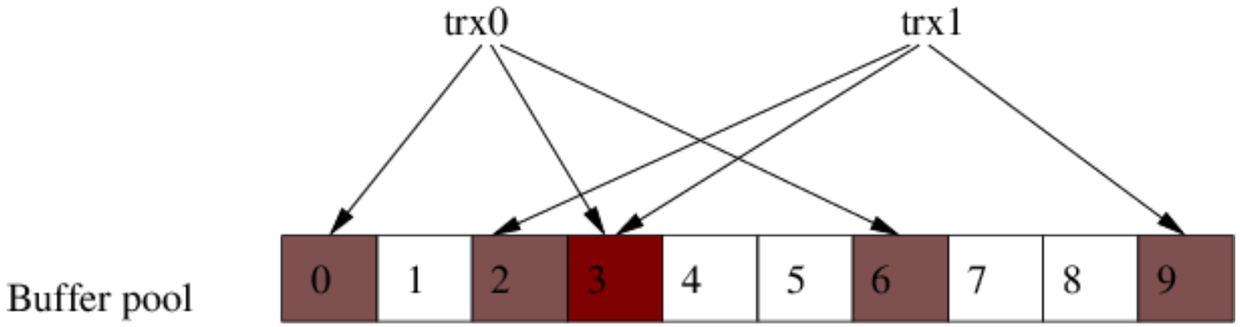
Transactions and consistency control (InnoDB case)



Tempesta Technologies

Lesson 3

- **Atomicity: *which* pages and *when* are written**
- **Database operates on record granularity**



Q: Can modern filesystem do this for us?

Log-enhanced filesystems

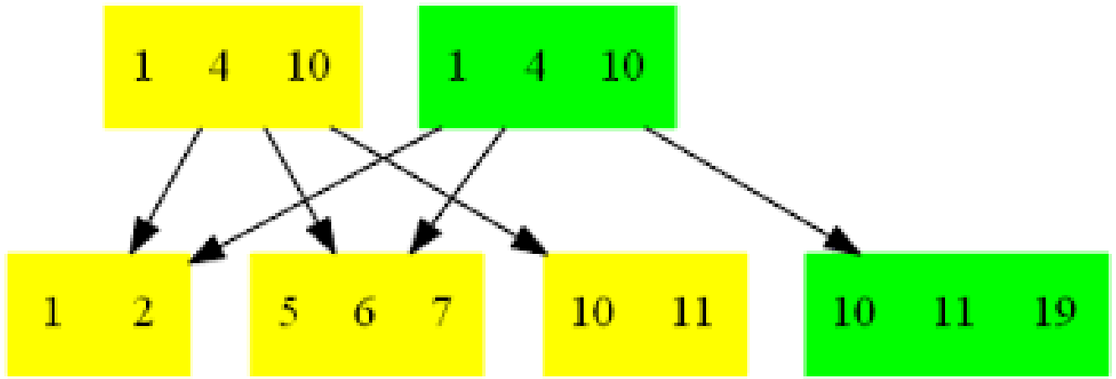
- **XFS** – metadata only
- **EXT4** – metadata and **data**
- Log writes are **sequential**, data updates are **batched**
- **Double writes** on data updates

Log-structured filesystems

- BSD LFS, Nilfs2
- **Dirty data blocks** are written to **next available segment**
- Changed **metadata** is also written to **new location**
- ...so **poor performance** on **data updates**
- Inodes aren't at fixed location → **inode map**
- **Garbage collection** of dead blocks (with significant overhead)
- **Poor fragmentation** on large files → slow updates

Copy-On-Write filesystems

- BtrFS, ZFS
- **Whole** tree **branches** are **COW**'ed
- **Constant** root place
- Still **fragmentation** issues and heavy write loads
- Very poor at **random writes** (OLTP), better for OLAP

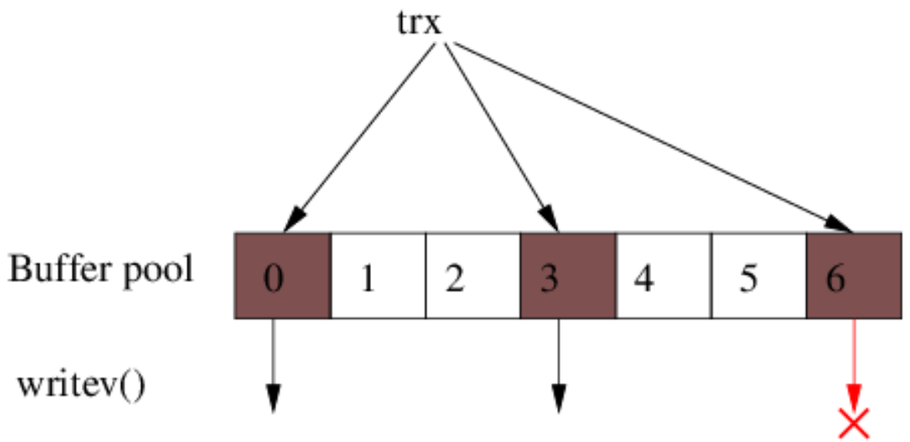


Soft Updates

- **BSD UFS2**
- **Proper ordering** to keep filesystem structure consistent *(metadata)*
- **Garbage collection** to gather lost data blocks
- Knows about filesystem metadata, not about stored data

Lesson 4:
Data consistency control

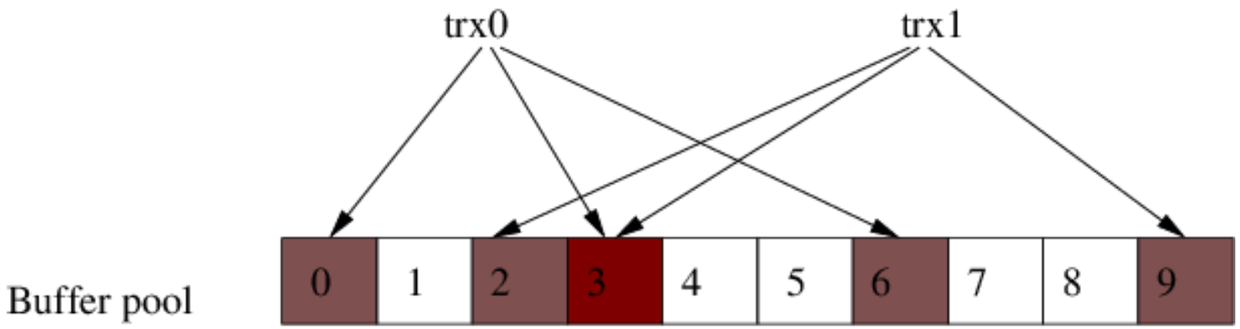
- Can log-enhanced data journaling FS replace doublewrite buffer?
  https://www.percona.com/blog/2015/06/17/update-on-the-innodb-double-write-buffer-and-ext4-transactions/
  , by Yves Trudeau, Percona.
- **NOT!**
  - Filesystem gurantees data block consistency, not group of blocks!



Tempesta Technologies

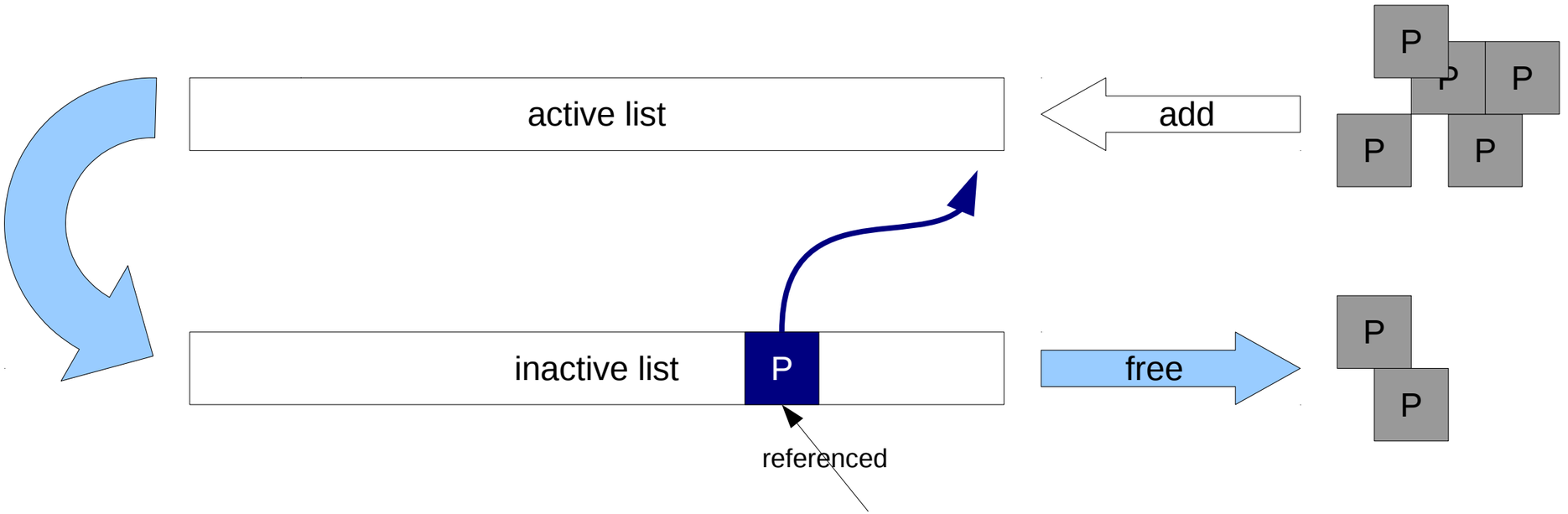
Lesson 5

- **Modern Linux filesystems are *unstructured***



Page eviction

- Typically current process reclaims memory
- ***kswapd*** – *alloc_pages()* slow path
- **OOM**



File synchronization syscalls

- `open(.., O_SYNC | O_DSYNC)`
- `fsync(int fd)`
- `fdatasync(int fd)`
- `msync(int *addr, size_t len,..)`
- `sync_file_range(int fd, off64_t off, off64_t nbytes,..)`

File synchronization syscalls

- `open(.., O_SYNC | O_DSYNC)`
- `fsync(int fd)`
- `fdatasync(int fd)`
- `msync(int *addr, size_t len,..)`
- `sync_file_range(int fd, off64_t off, off64_t nbytes,..)`

- No page subset syncrhonization
- *write(fd, buf, **1GB**)* – isn't atomic against system failure
- Some pages can be flushed before synchronization

Flush out advises

- `posix_fadvise(int fd, off_t offset, off_t len, int advice)`
  - POSIX_FADV_DONTNEED – invalidate specified pages

```
int invalidate_inode_page(struct page *page) {
    if (PageDirty(page) || PageWriteback(page))
        return 0;
```

- `madvise(void *addr, size_t length, int advice)`
  - MADV_DONTNEED – unmap page table entries, initializes dirty pages flushing

Lesson 6:
Linux VMM as DBMS engine?

- **Linux VMM**
  - *evicts* dirty pages
  - it doesn't know *exactly* whether they're still needed (DONTNEED!)
  - nobody knows *when* the pages are synced
  - checkpoint is typically full database file sync
  - performance: **locked scans** for free/clean pages by **timeouts** and **no-memory**
- **Don't use mmap() if you want consistency!**

Transactional filesystems: Reiser4

- Hybrid TM: Journaling or Write-Anywhere (Copy-On-Write)
- Only small data block writes are transactional
- Full transaction support for large writes isn't implemented

Transactional filesystems: BtrFS

- Uses **log-trees**, so [probaly] can be used instead of doublewrite buffer
- **ioctl()**: BTRFS_IOC_TRANS_START and BTRFS_IOC_TRANS_END

Transactional filesystems: others

- **Valor**
  R.P.Spillane et al, “Enabling Transactional File Access via Lightweight Kernel Extensions”, FAST'09
  - Transactions: kernel module betweem VFS and filesystem
  - New transactional syscalls (*log begin*, *log append*, *log resolve*, *transaction sync*, locking)
  - patched *pdflush* for eviction in *proper order*
- **Windows TxF**
  - deprecated

Transactional operating systems

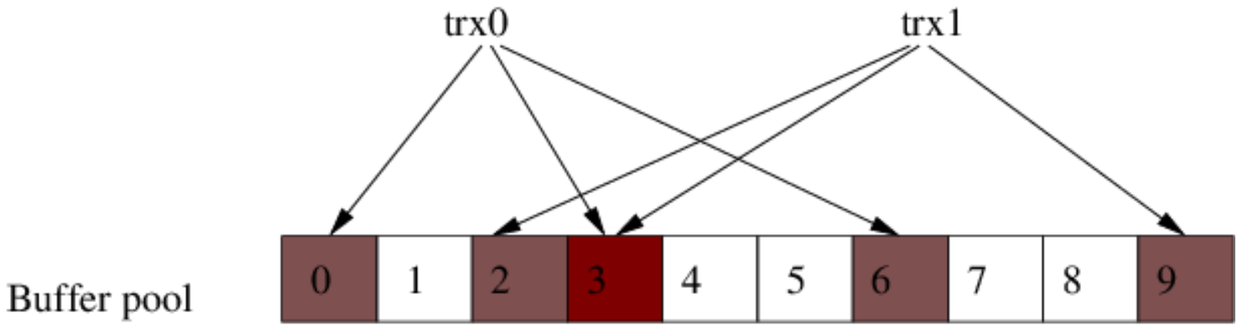
- **TxOS**
  D.E.Porter et al., “Operating System Transactions”, SOSP'09
  - Almost any sequence of syscalls can run in transactional context
  - New transactional syscalls (*sys_xbegin*, *sys_xend*, *sys_xabort*)
  - Alters kernel data structures by transactional headers
  - Shadow-copies consistent data structures
  - Properly resolves conflicts between transactional and non-transactional calls

The same OS does the right job

- **Failure-atomic msync()**
  S.Park et al., “Failure-Atomic msync(): A Simple and Efficient Mechanism for Preserving the Integrity of Durable Data”, Eurosys'13.
  - No voluntary page writebacks: *MAP_ATOMIC* for *mmap()*
  - Jounaled writeback
    - *msync()*
    - REDO logging
    - page writebacks

Record-oriented filesystem

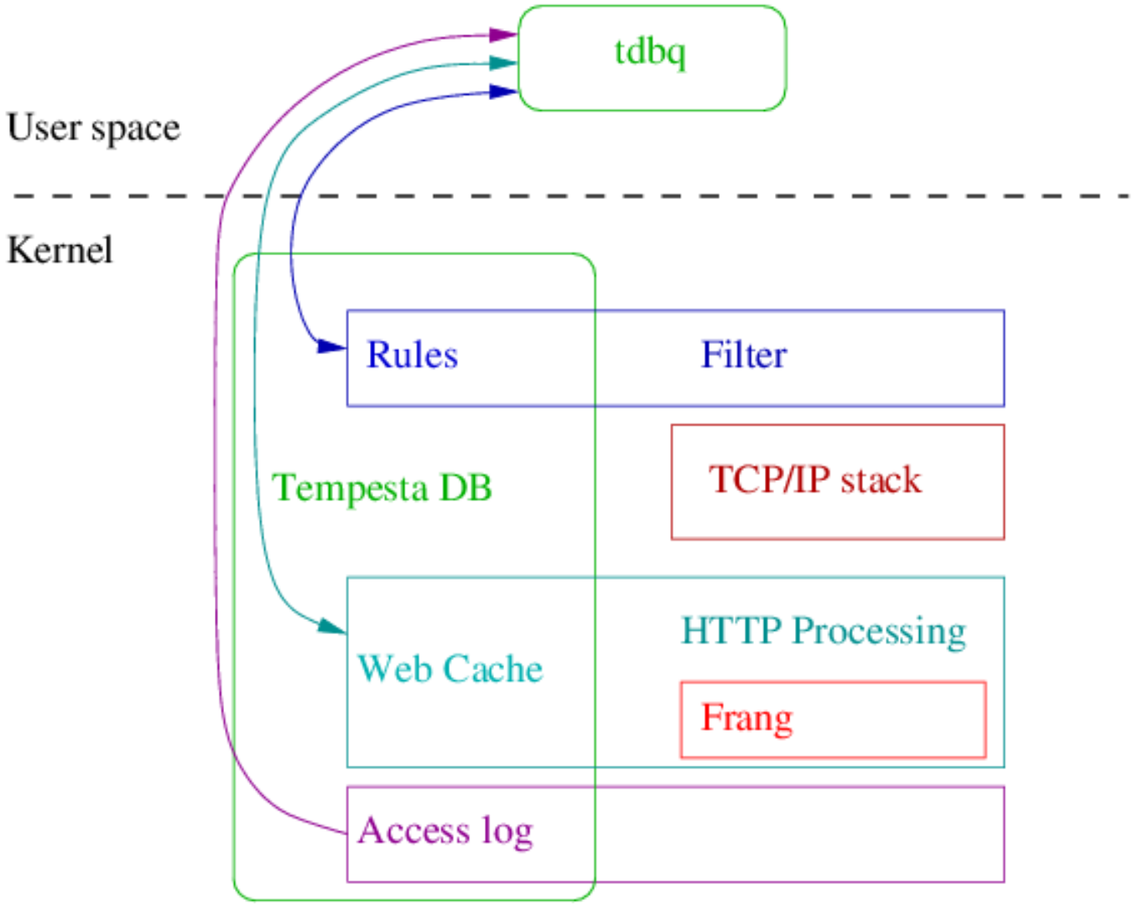
- **OpenVMS Record Management Service (RMS)**
  - Record formats: *fixed length*, *variable length*, *stream*
  - Access methods: *sequential*, *relative record number*, *record address*, *index*
  - *sys$get() & sys$put()* instead of *read()* and *write()*



TempestaDB

- Is part of TempestaFW (a hybrid of **firewall** and **Web-accelerator**)
- **In-memory** database for Web-cache and firewall rules (*must be fast!*)
  Stonebreaker's "The Traditional RDBMS Wisdom is All Wrong"
- Accessed from **kernel space** (*softirq!*) as well as **user space**
- Can be **concurrently accessed by many processes**
- *In-progress development*

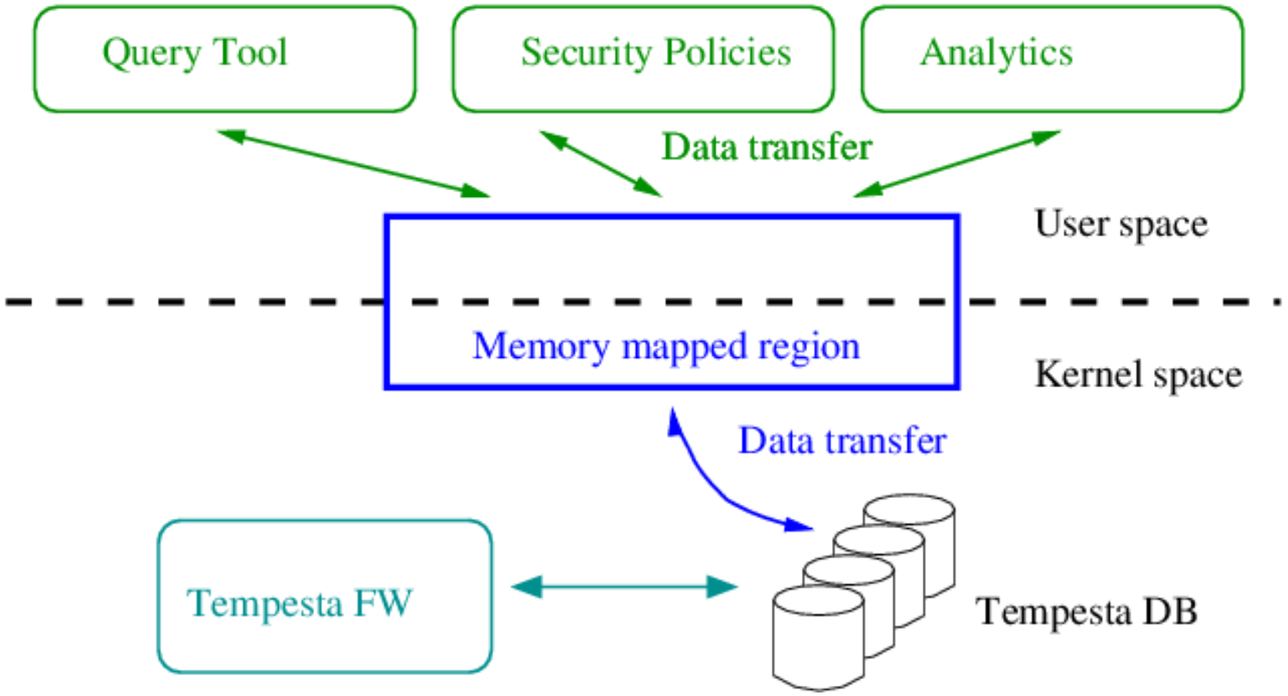
Kernel database for Web-accelerator?



Transport

http://natsys-lab.blogspot.ru/2015/03/linux-netlink-mmap-bulk-data-transfer.html

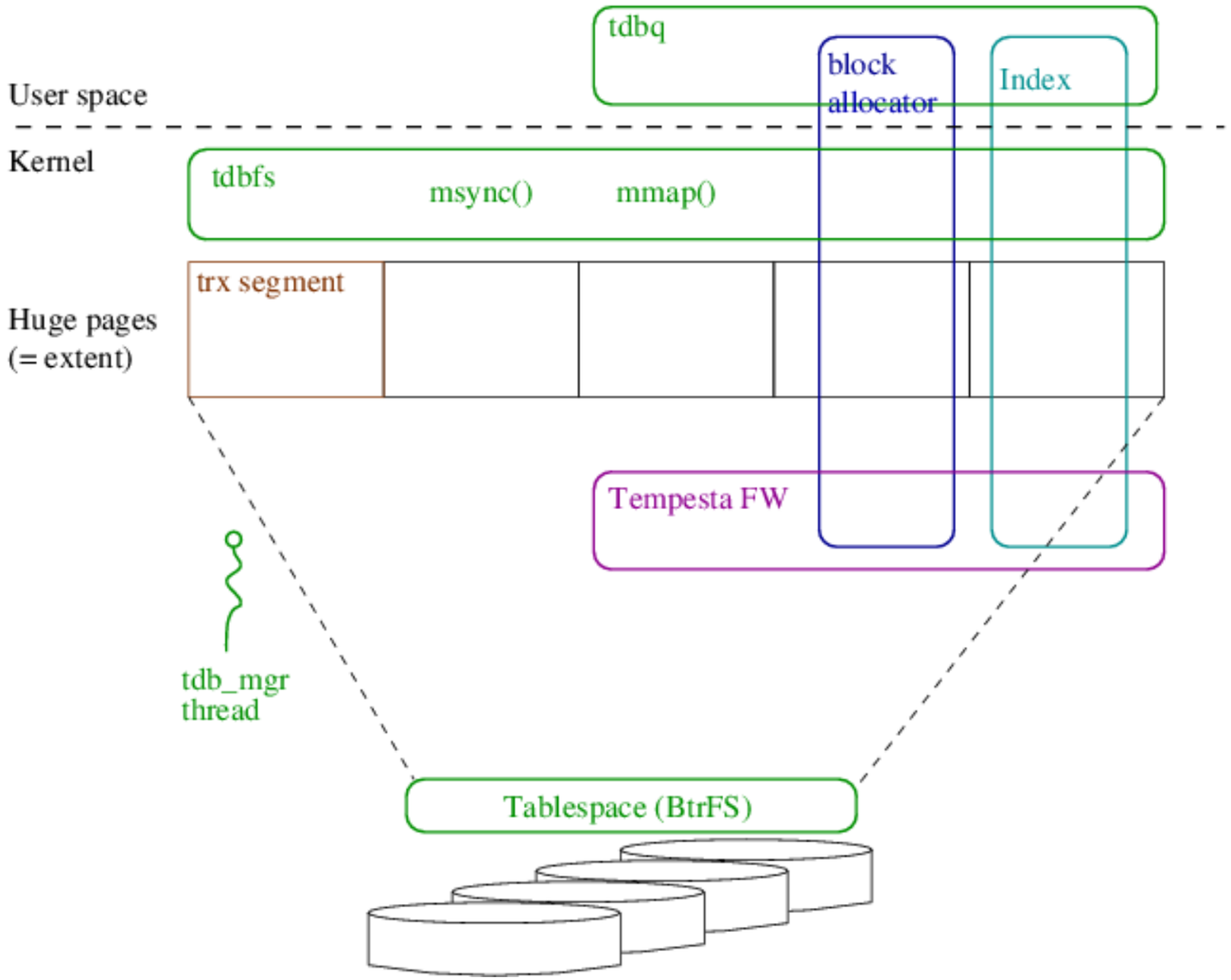
- Collect query results → copy to some buffer
- Zero-copy mmap() to user-space
- Show to user



TempestaDB internals

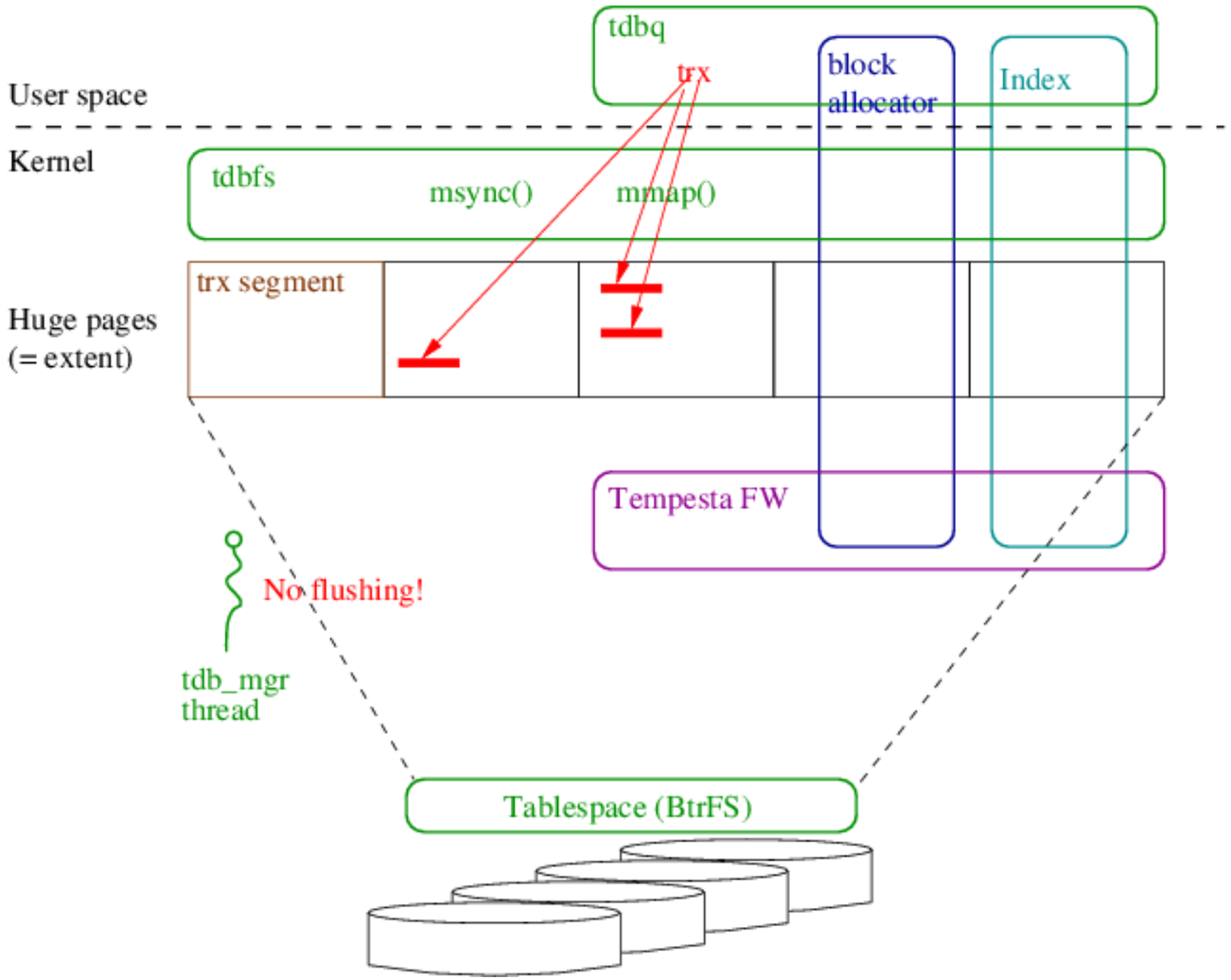
- **Preallocates** large pool of **huge pages** at boot time
  - so **full DB file mmap()** is compensated by huge pages
  - 2MB extent = huge page
- **Tdbfs** is used for custom *mmap()* and *msync()* for persistency
- *mmap()* => record-orientation out of the box
- **No-steal force** or **no-force** buffer management
  - no need for doublewrite buffer
  - undo and redo logging is **up to application**
- Automatic **cache eviction**

TempestaDB internals



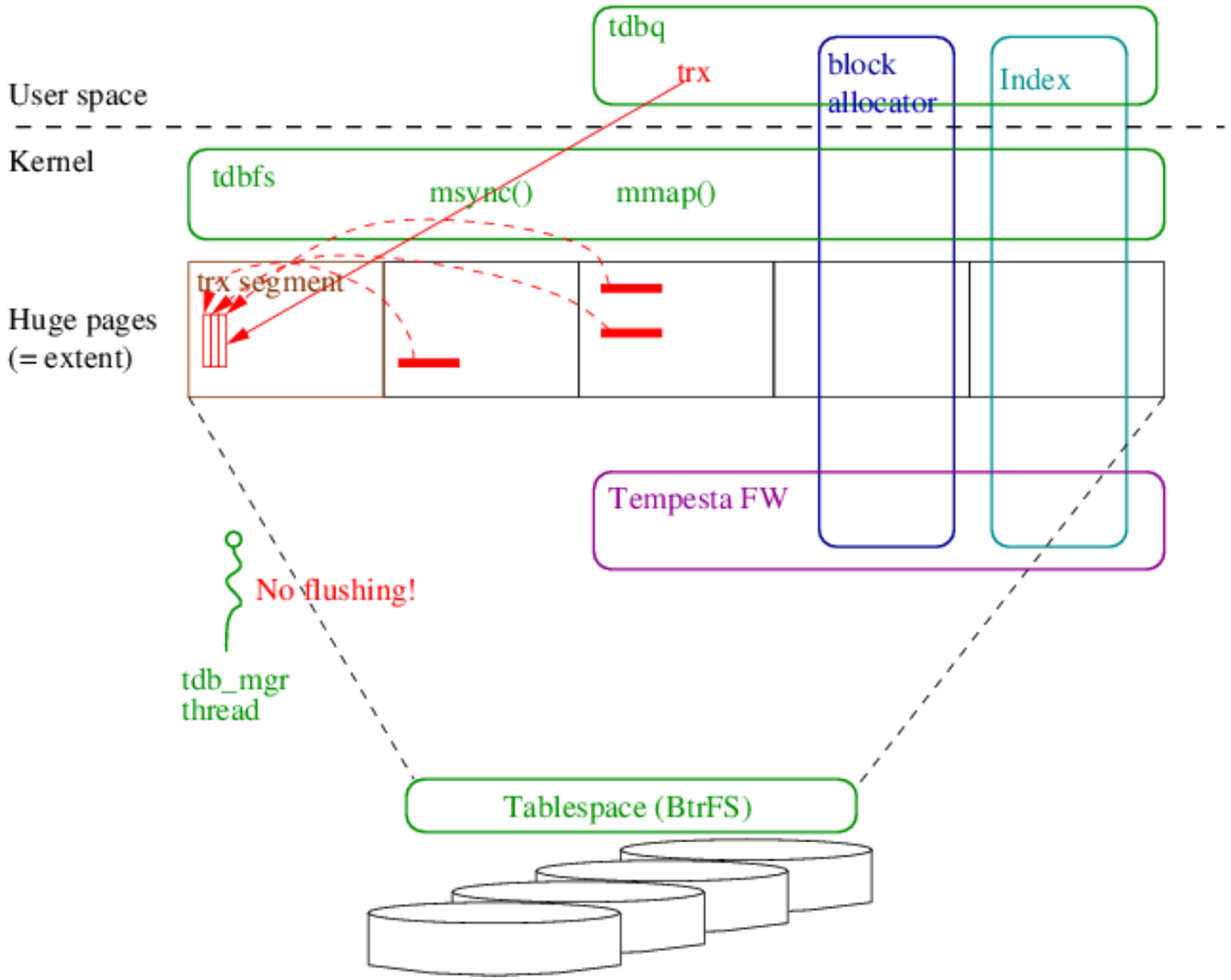
Tempesta Technologies

TempestaDB: trx write (no-steal)



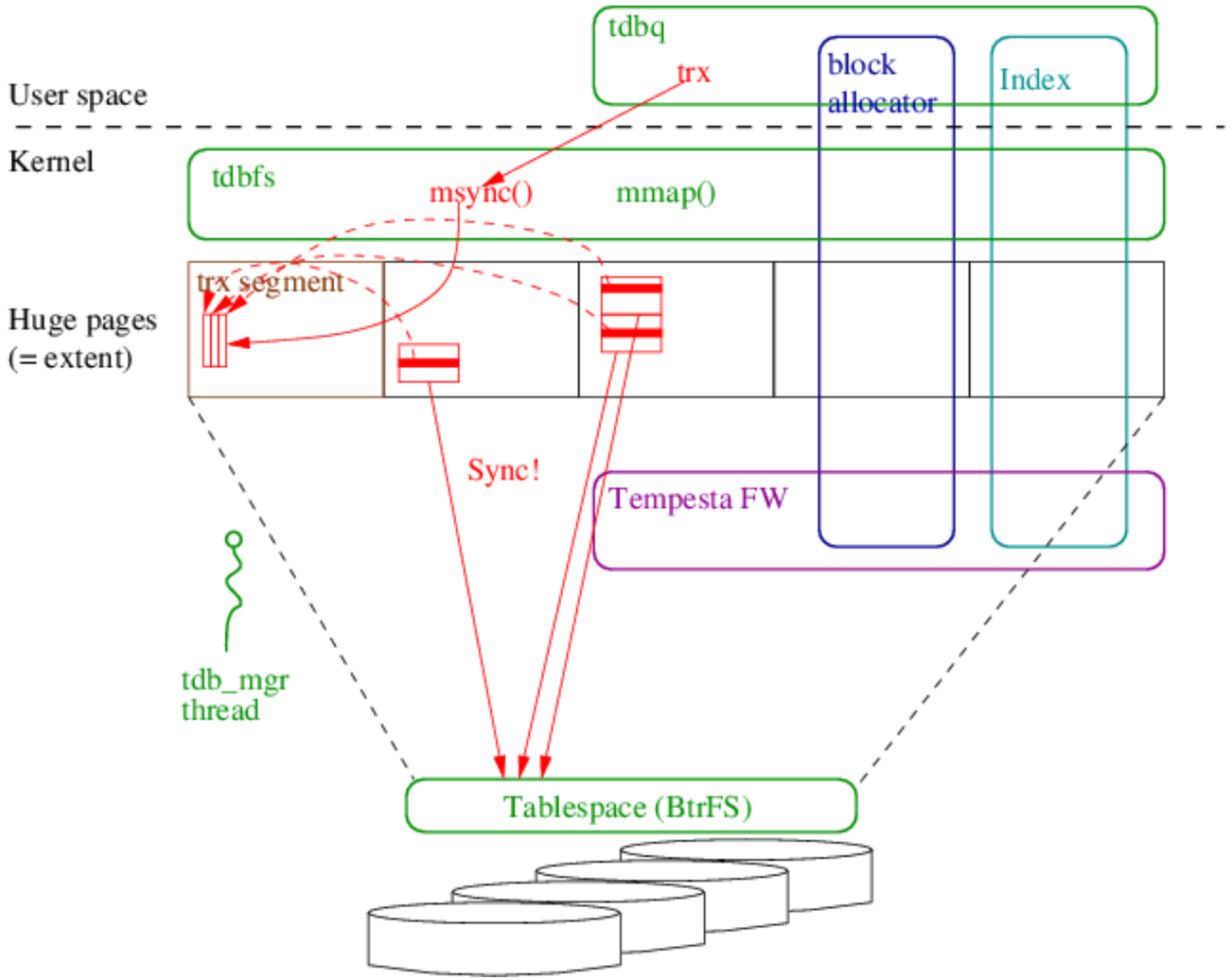
Tempesta Technologies

TempestaDB: commit



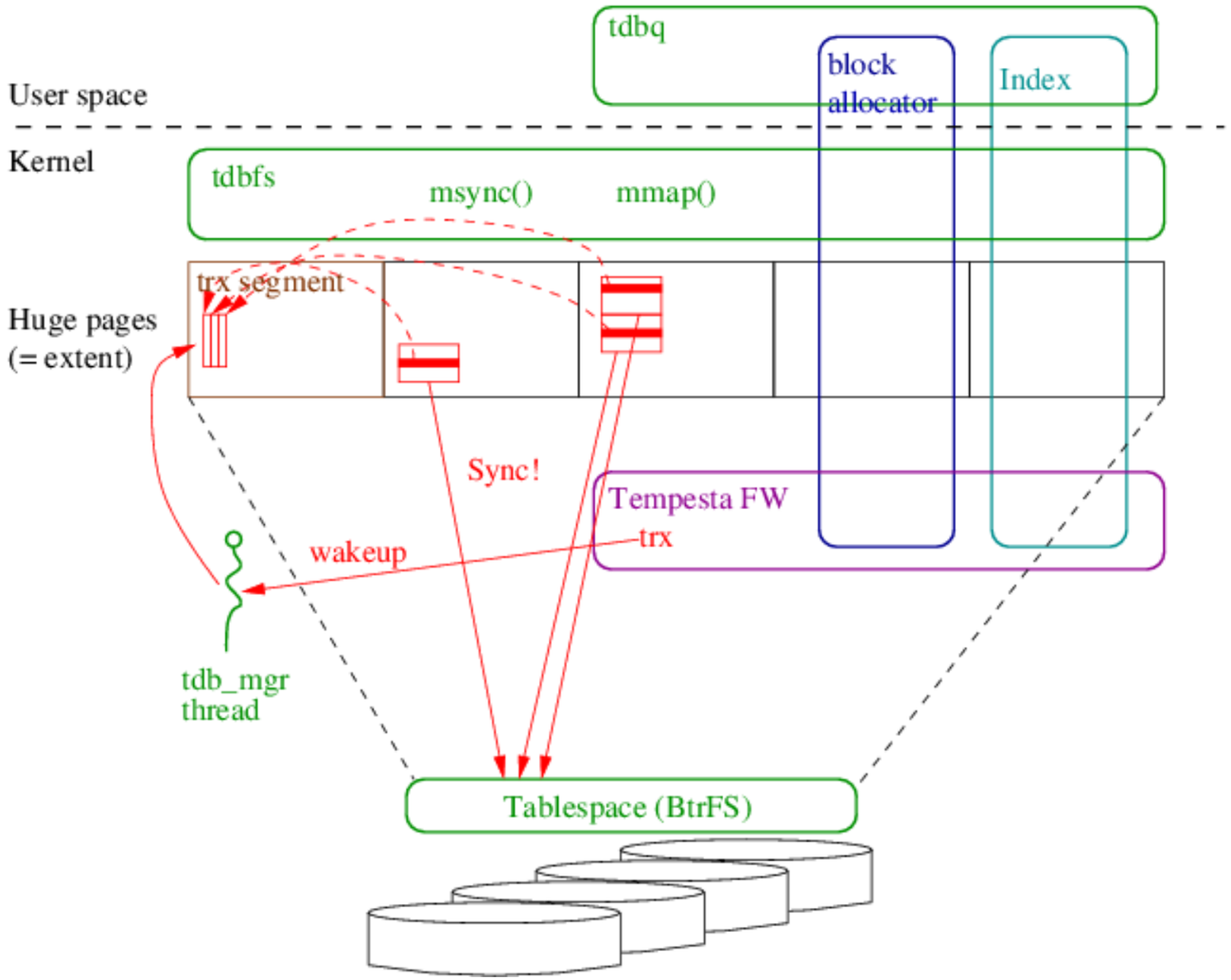
Tempesta Technologies

TempestaDB: commit (force)



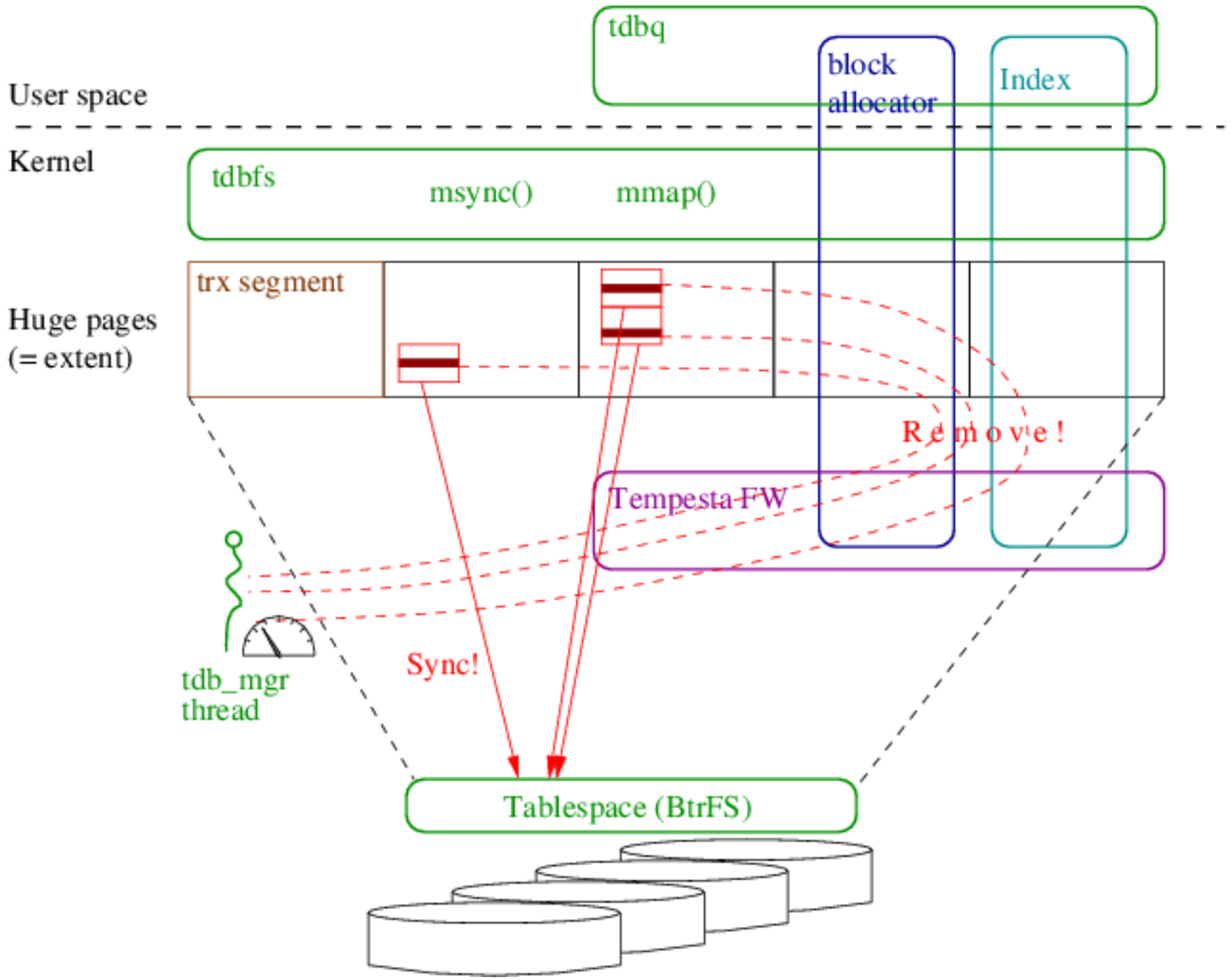
Tempesta Technologies

TempestaDB: commit (no-force)



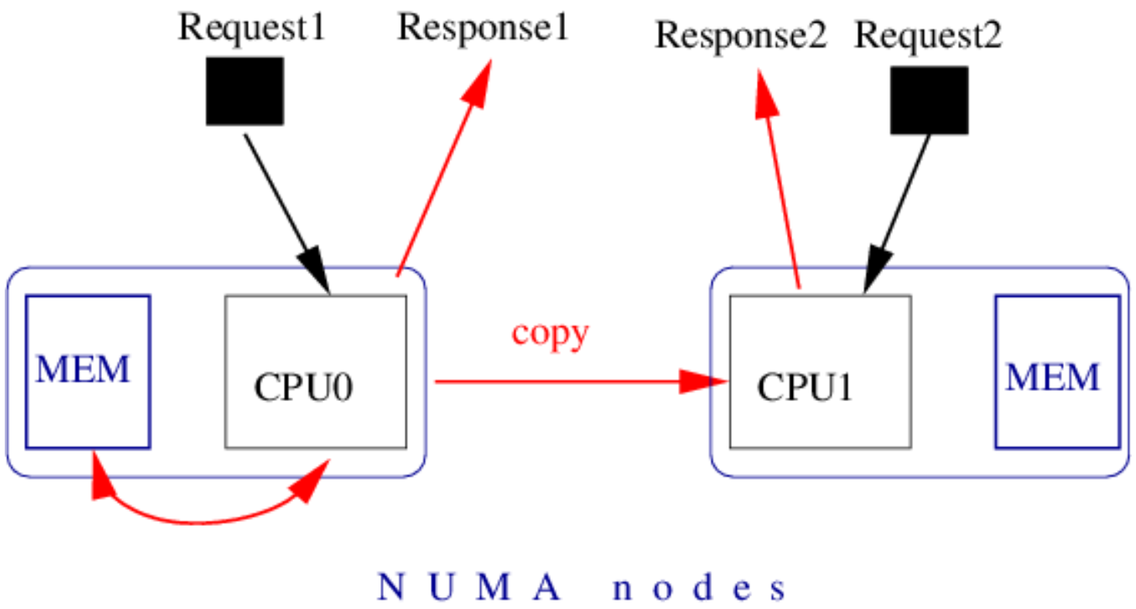
Tempesta Technologies

TempestaDB: cache eviction

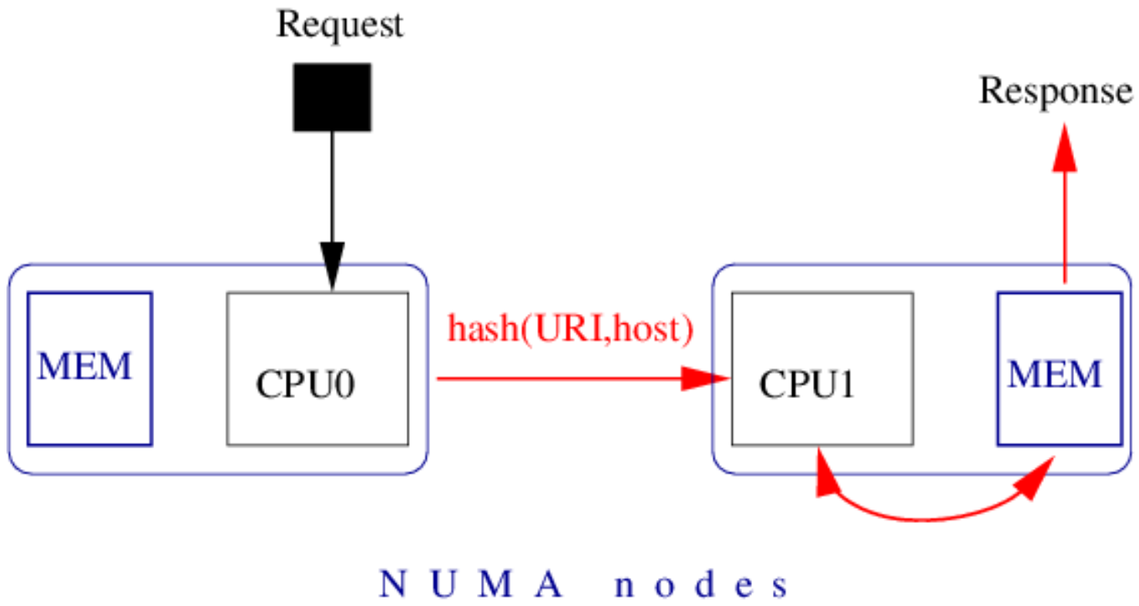


Tempesta Technologies

NUMA replication



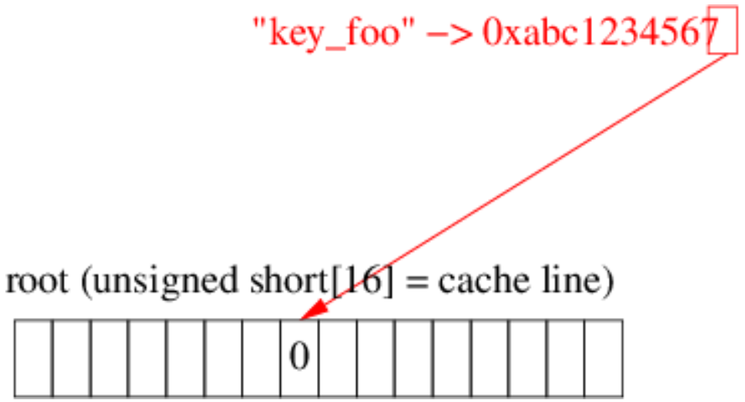
NUMA sharding



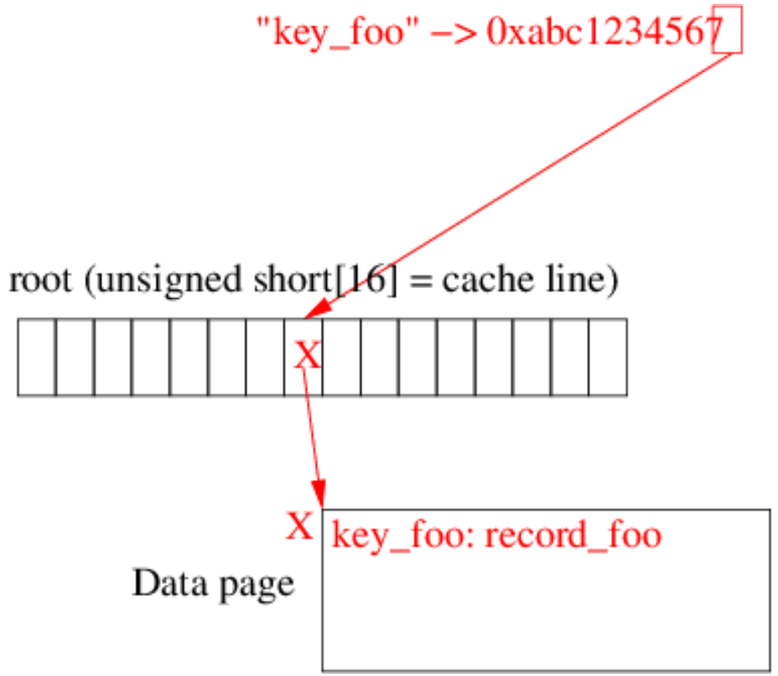
Memory optimized

- **Cache conscious Burst Hash Trie**
  - **short offsets** instead of pointers
  - (almost) **lock-free**
- **lock-free block allocator** for virtually contiguous memory

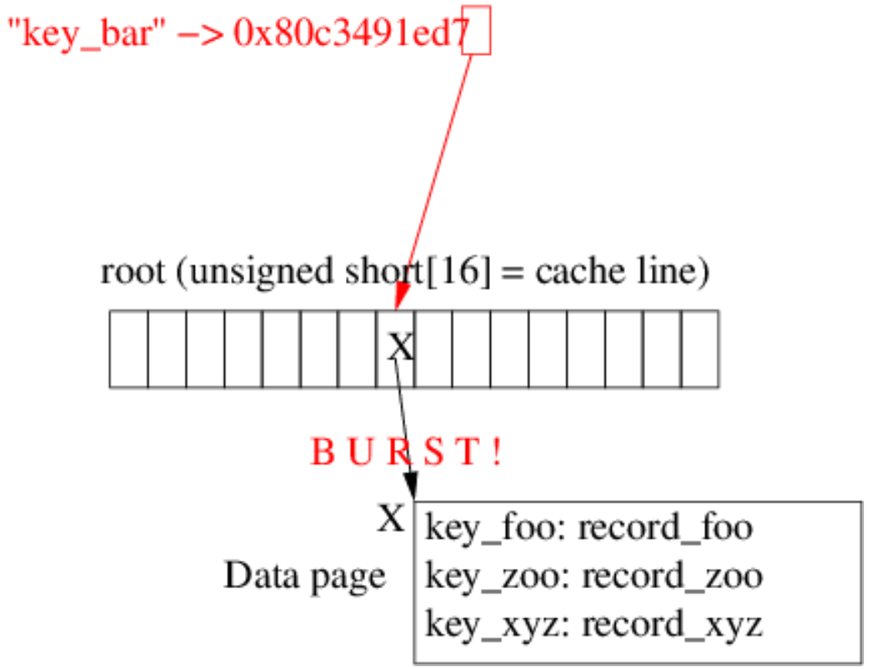
Burst Hash Trie



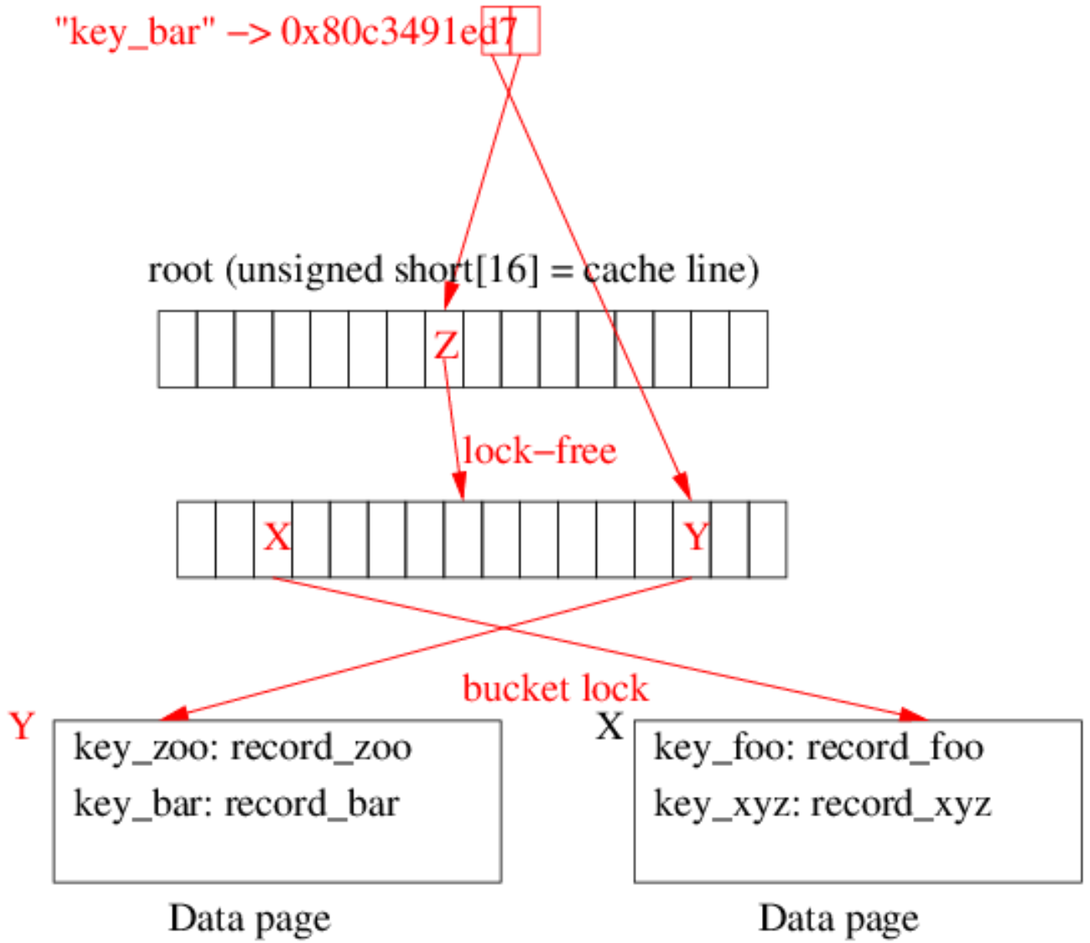
Burst Hash Trie



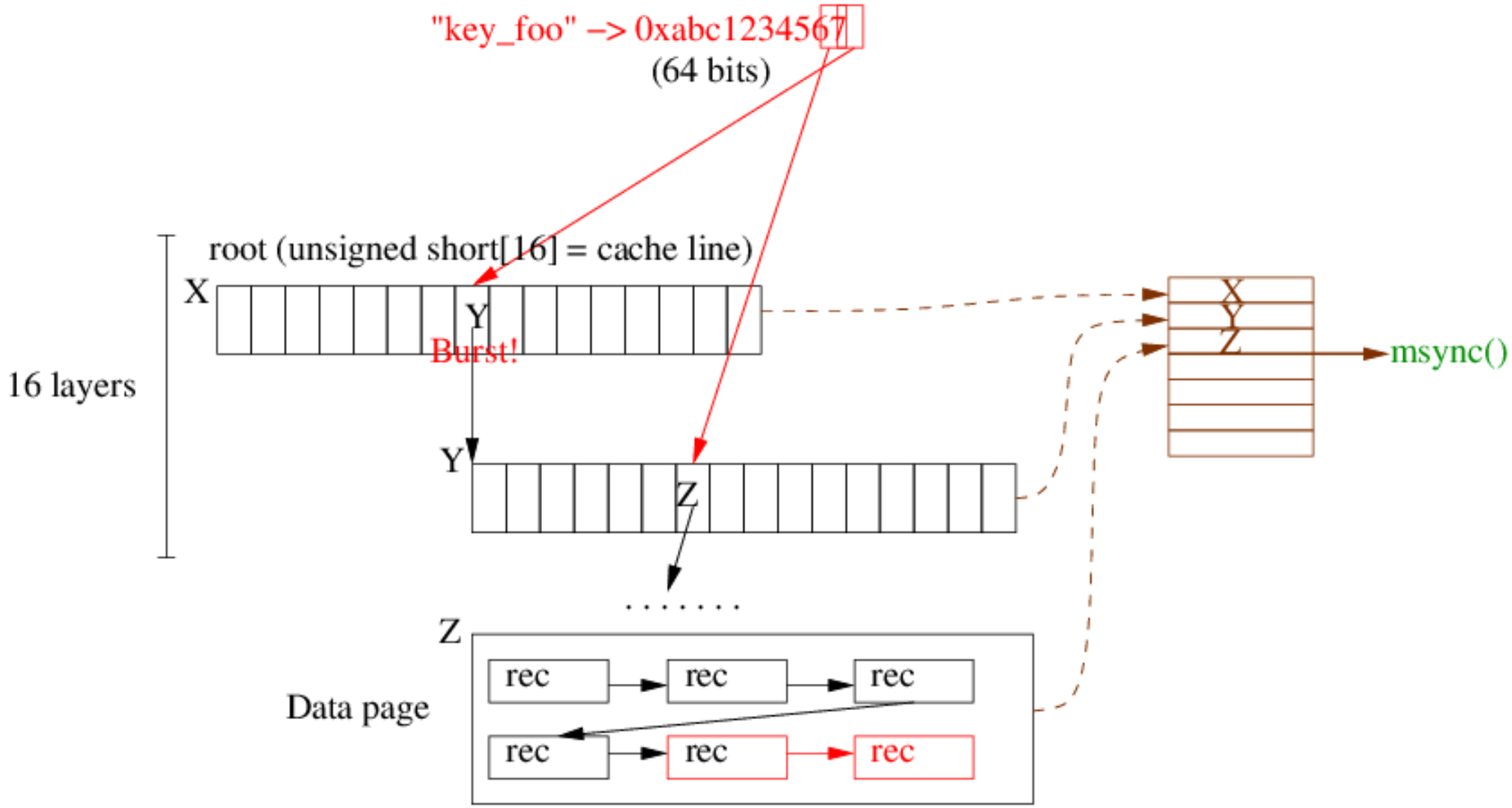
Burst Hash Trie



Burst Hash Trie



Burst Hash Trie: transactions



Thanks!

- Availability: *https://github.com/tempesta-tech/tempesta*
- Blog: *http://natsys-lab.blogspot.com*
- E-mail: *ak@tempesta-tech.com*

We are hiring!

Tempesta Technologies