Linux VMM
for Database Developers

Alexander Krizhanovsky
Tempesta Technologies, NatSys Lab.
ak@natsys-lab.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

Tempesta Technologies

The begin

(many years ago)

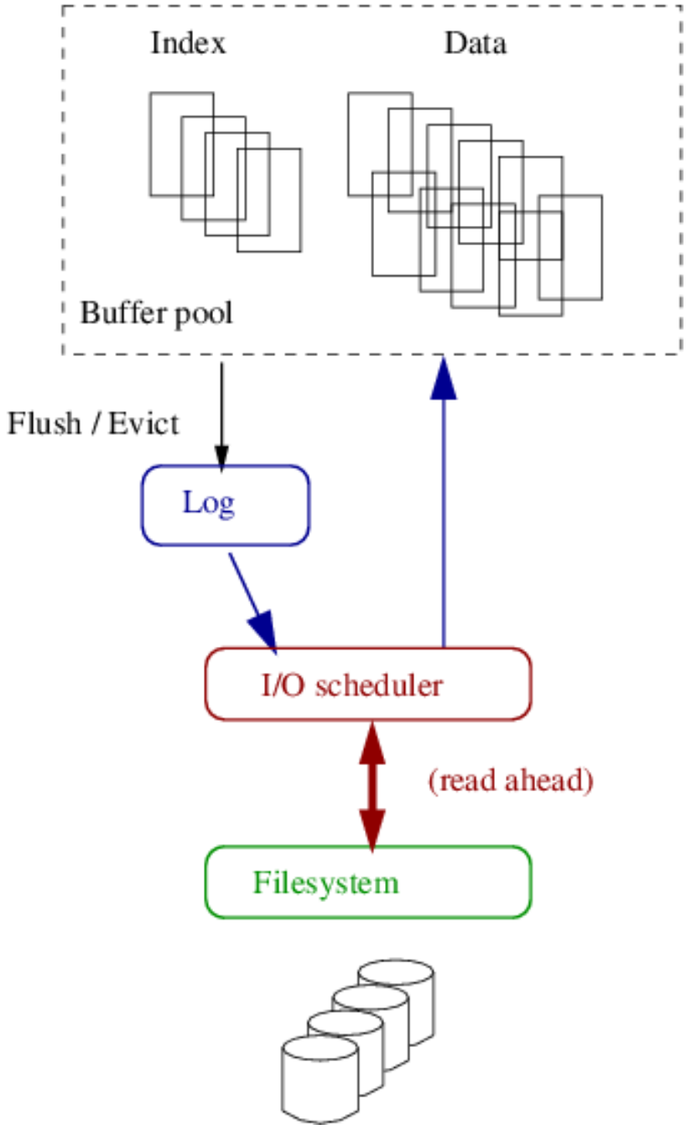
- **Database** to store Instant messenger's history
- **Plenty of data** (NoSQL, 3-touple key)
- High **performance**
- Weak **consistency** (no transactions)

The begin
(many years ago)

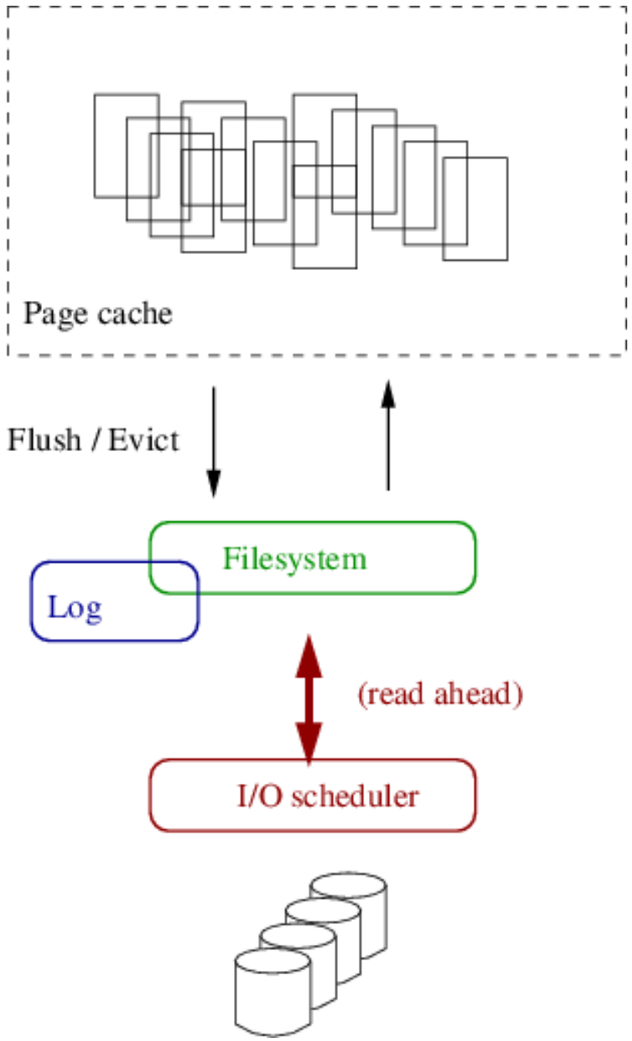
- **Database** to store Instant messenger's history
- **Plenty of data** (NoSQL, 3-touple key)
- High **performance**
- Weak **consistency** (no transactions)
- 2-3 months *(quick prototype)*

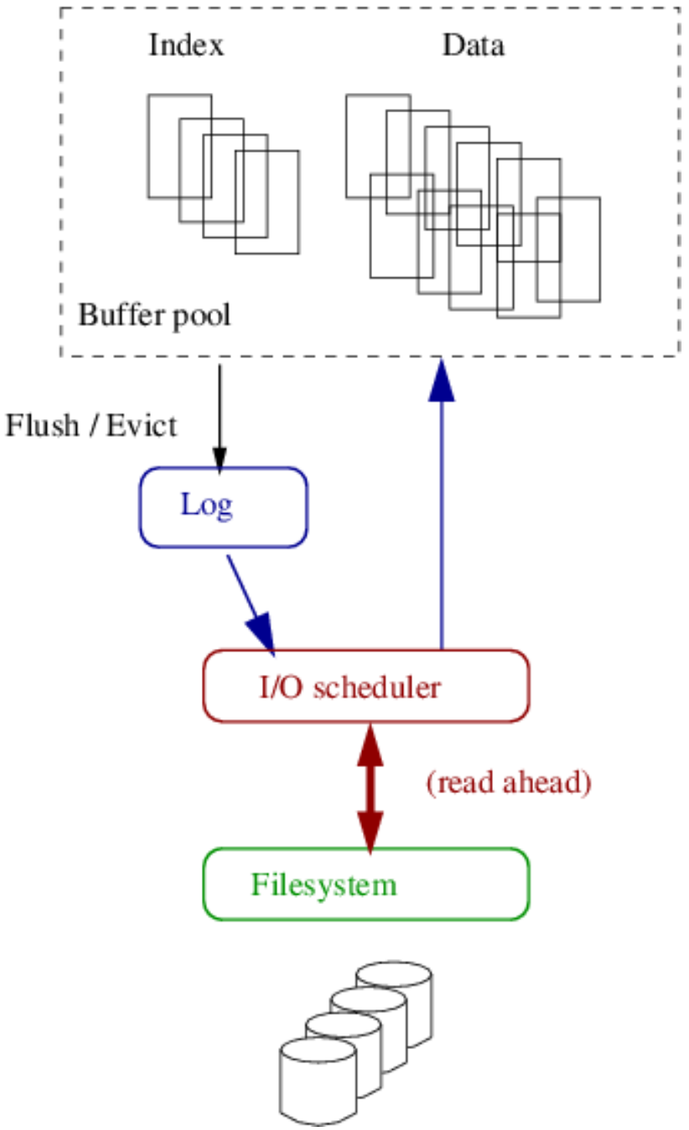
Simple DBMS

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



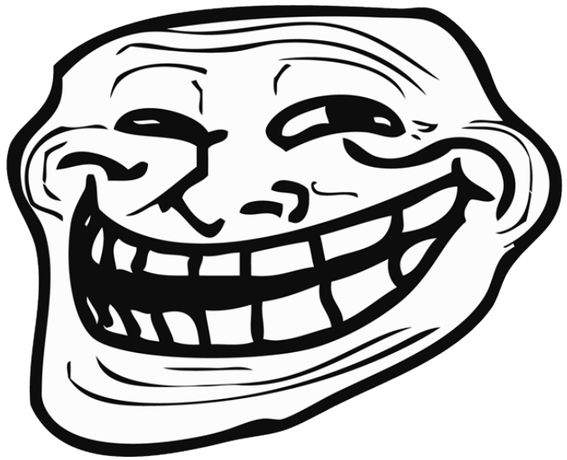
Linux VMM?


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

mmap(2)!

- Automatic page eviction
- Transparrent persistency

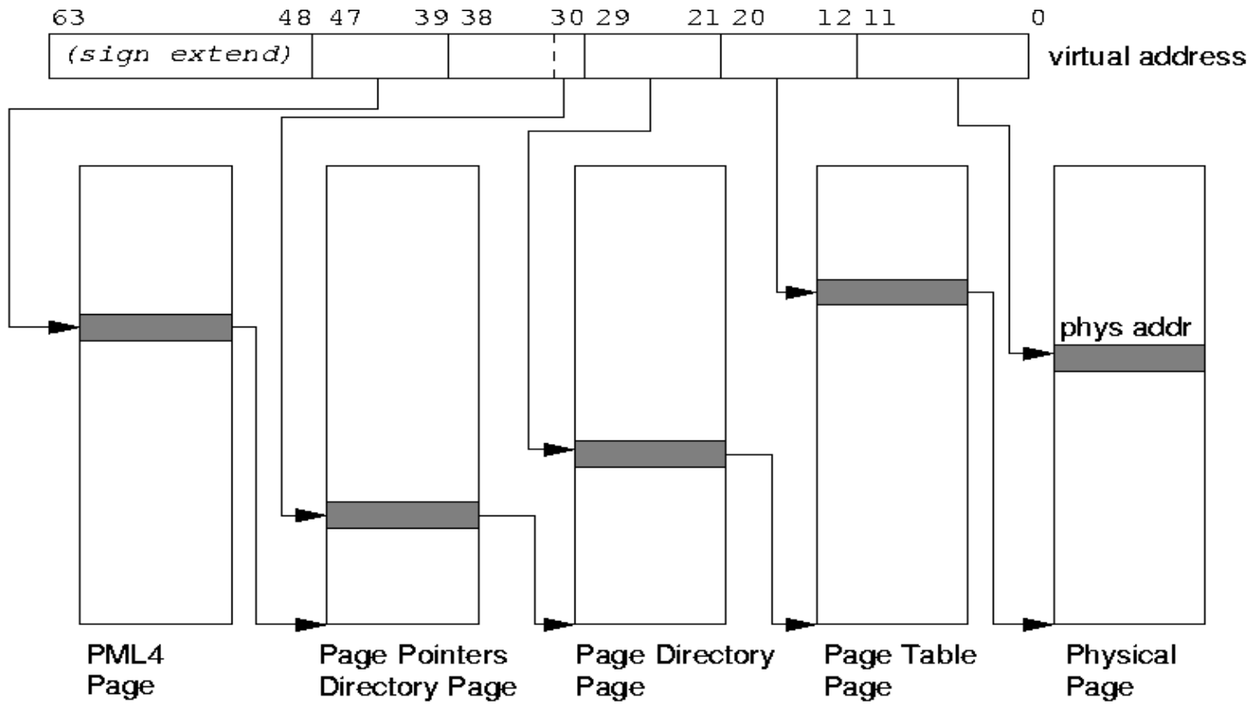
mmap(2)!

- Automatic page eviction
- Transparrent persistency
- I/O is managed by OS

mmap(2)!

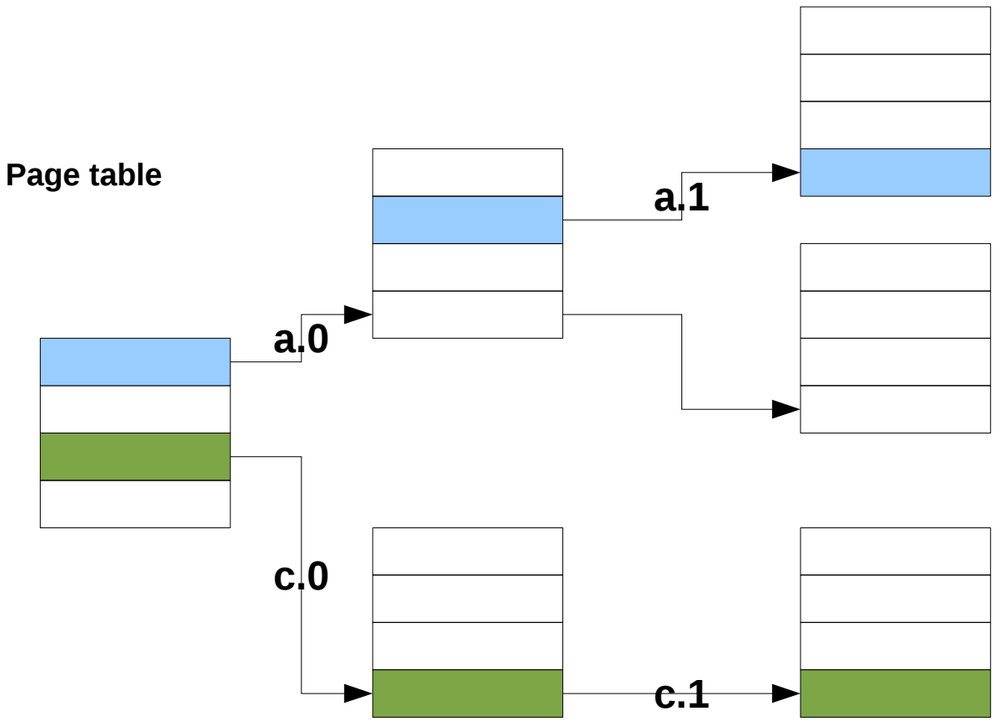
- Automatic page eviction
- Transparrent persistency
- I/O is managed by OS
- ...and ever **radix tree** index for free!

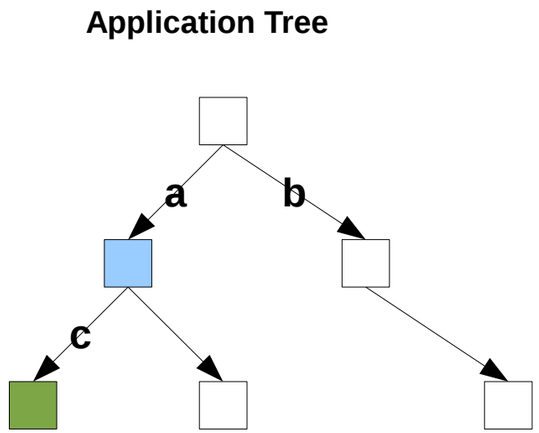
x86-64 page table (radix tree)



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);
```

mmap(2): index for free

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

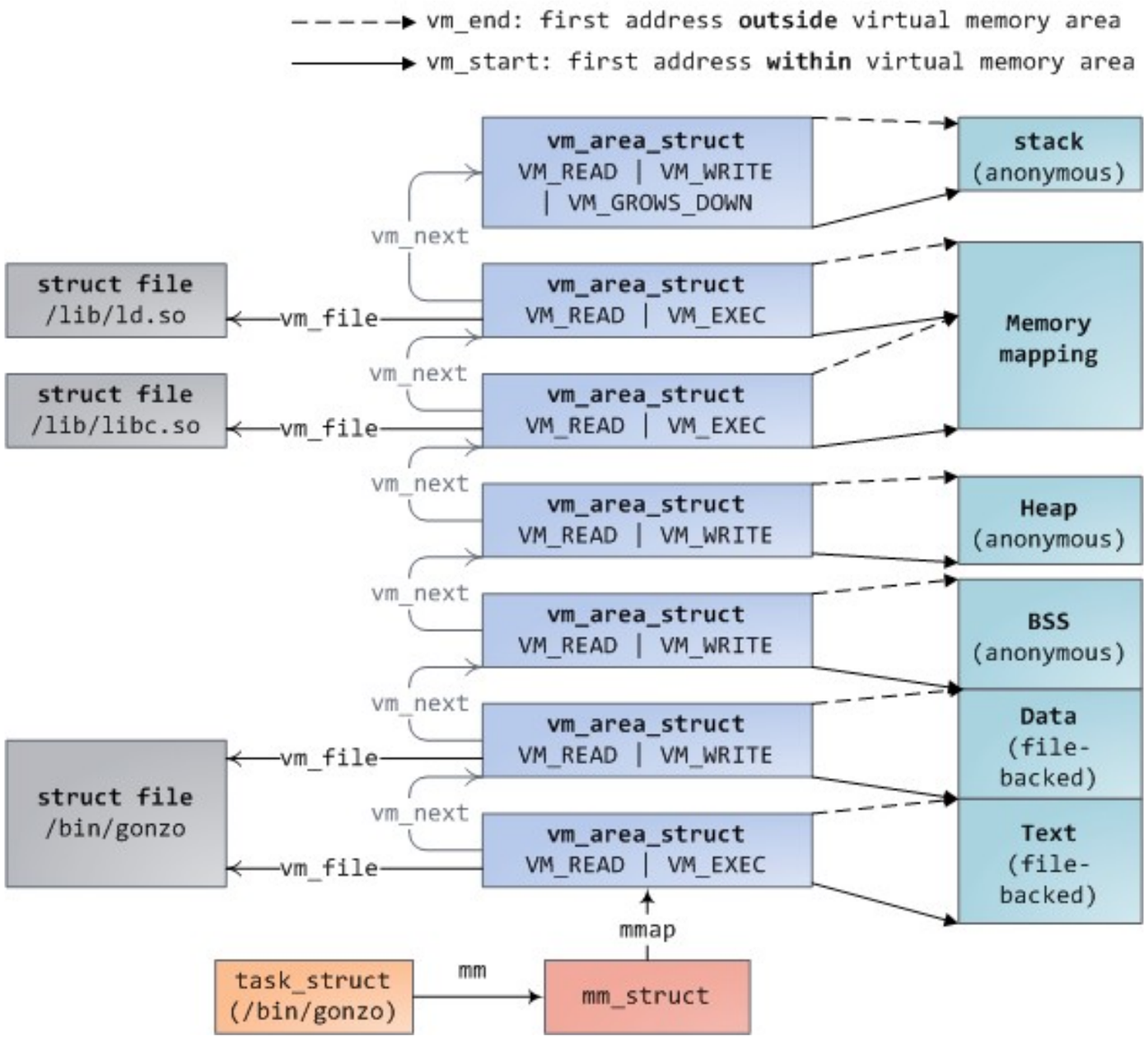
```
DbItem *db = mmap(0x6f0000000000, 0x40000000 /* 1GB */, ...);
DbItem *x = (DbItem *)(0x6f0000000000 + key);
```

- `0x6f0000000000 + key` – virtual address
- Data is stored in ***physical*** *page*
- Keys density is crucial

...or just an array

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

Virtual Memory Area (VMA)



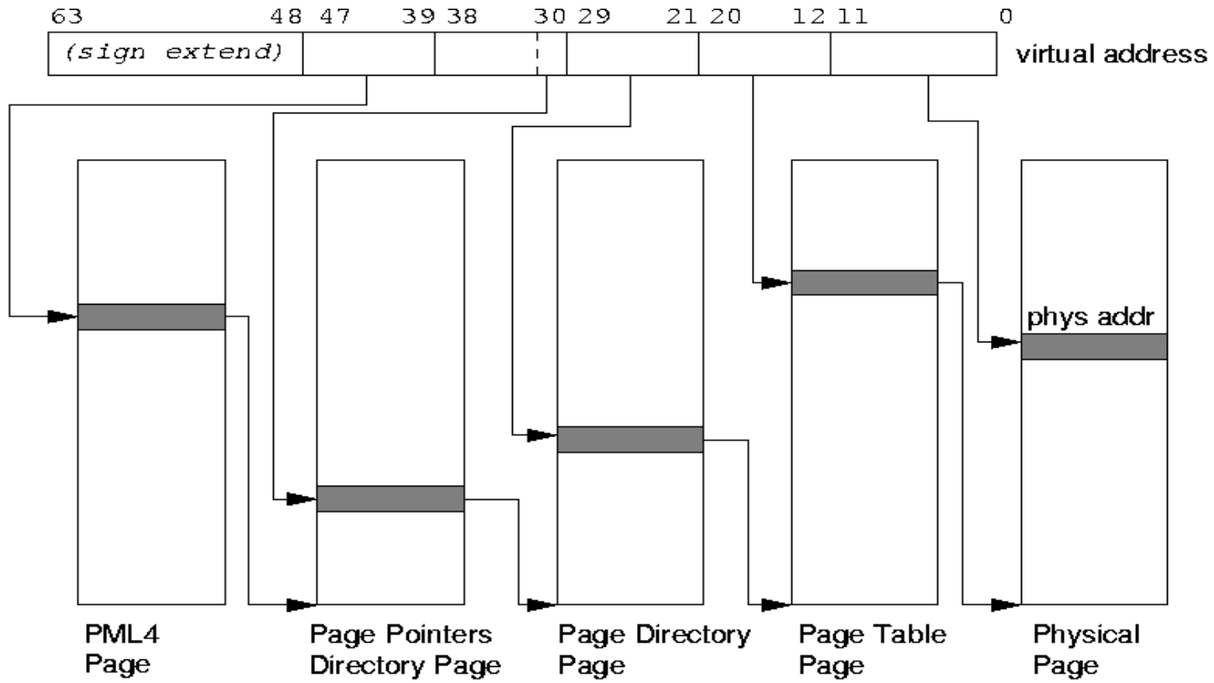
Tempesta Technologies

Virtual memory isn't for free

- TLB cache **is small** (~1024 entries, i.e. 4MB)

Virtual memory isn't for free

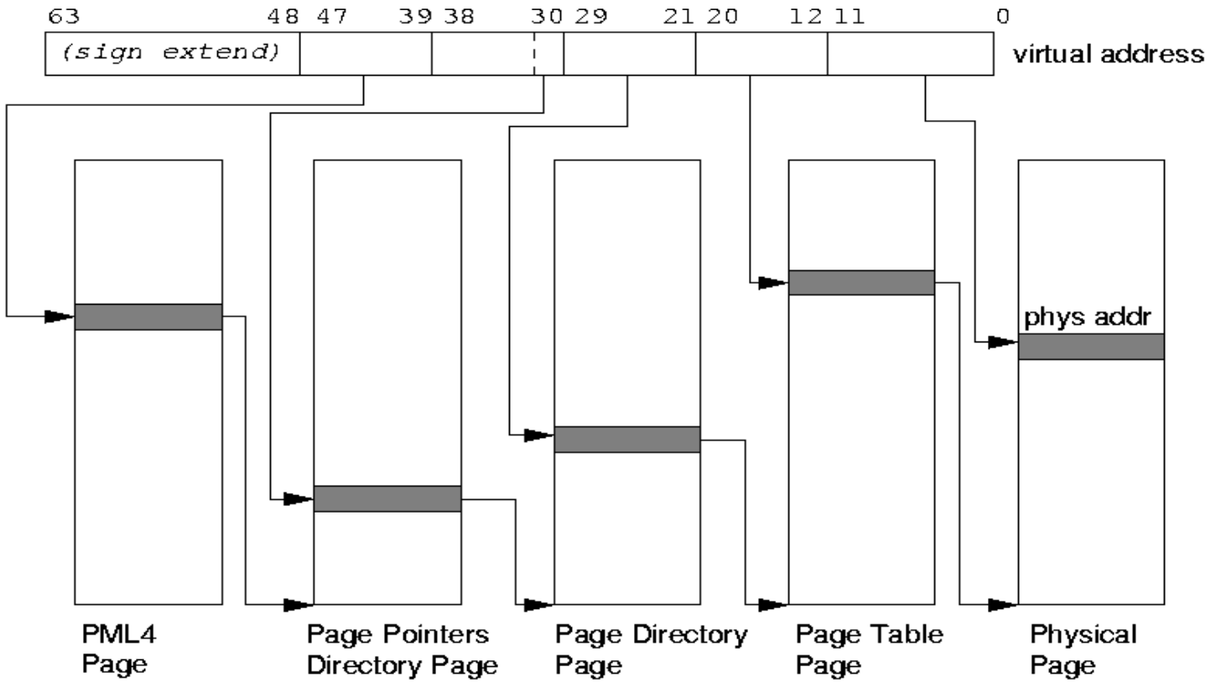
- TLB cache **is small** (~1024 entries, i.e. 4MB)
- TLB **cache miss** is up to 4 memory transfers



Tempesta
Technologies

Virtual memory isn't for free

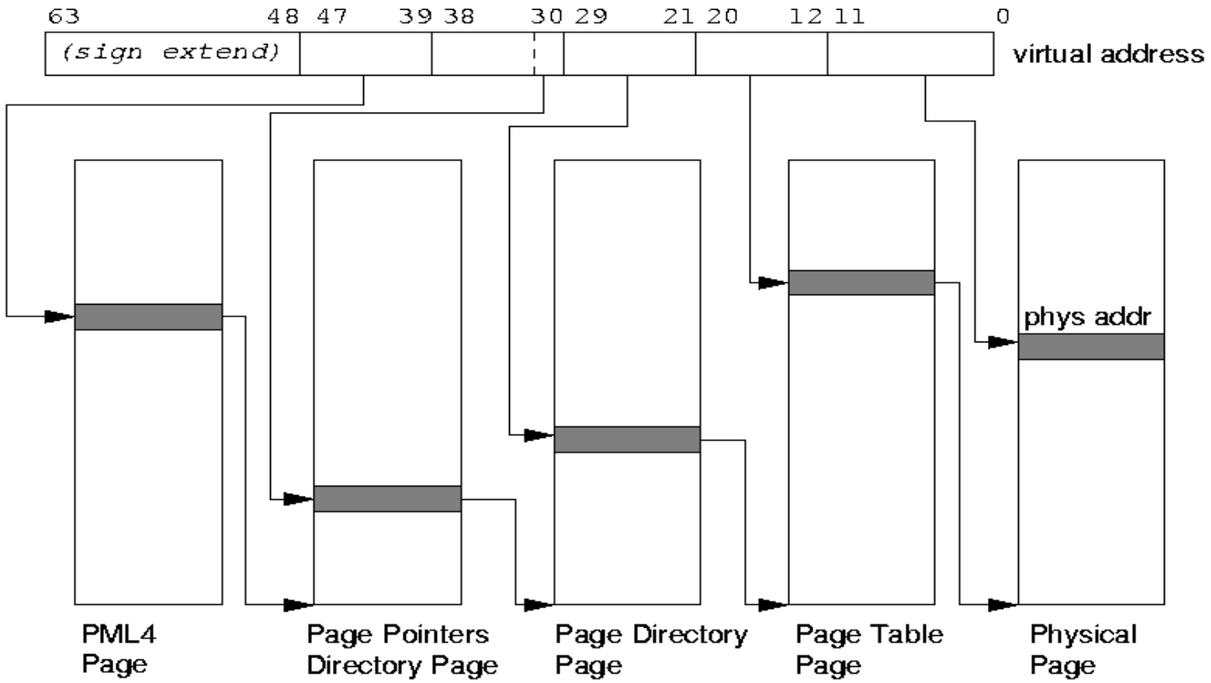
- TLB cache **is small** (~1024 entries, i.e. 4MB)
- TLB **cache miss** is up to 4 memory transfers
- **Spacial locality** is crucial: 1 address outlier is up to 12KB



Tempesta
Technologies

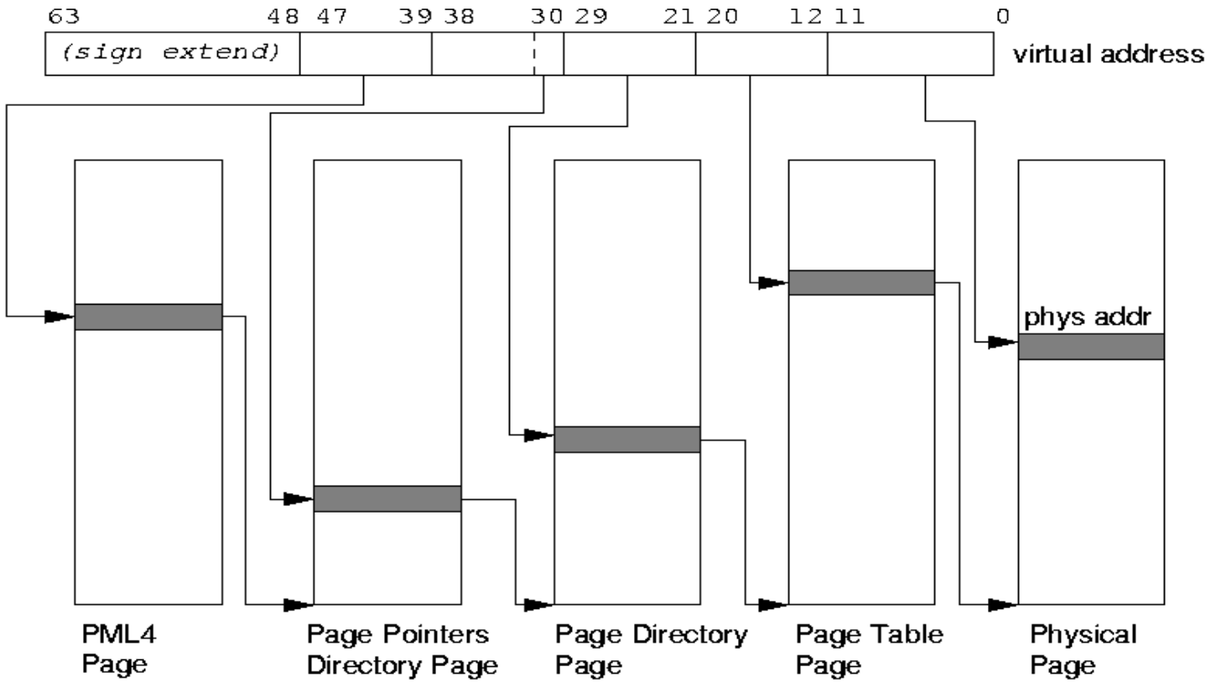
Virtual memory isn't for free

- 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



Virtual memory isn't for free

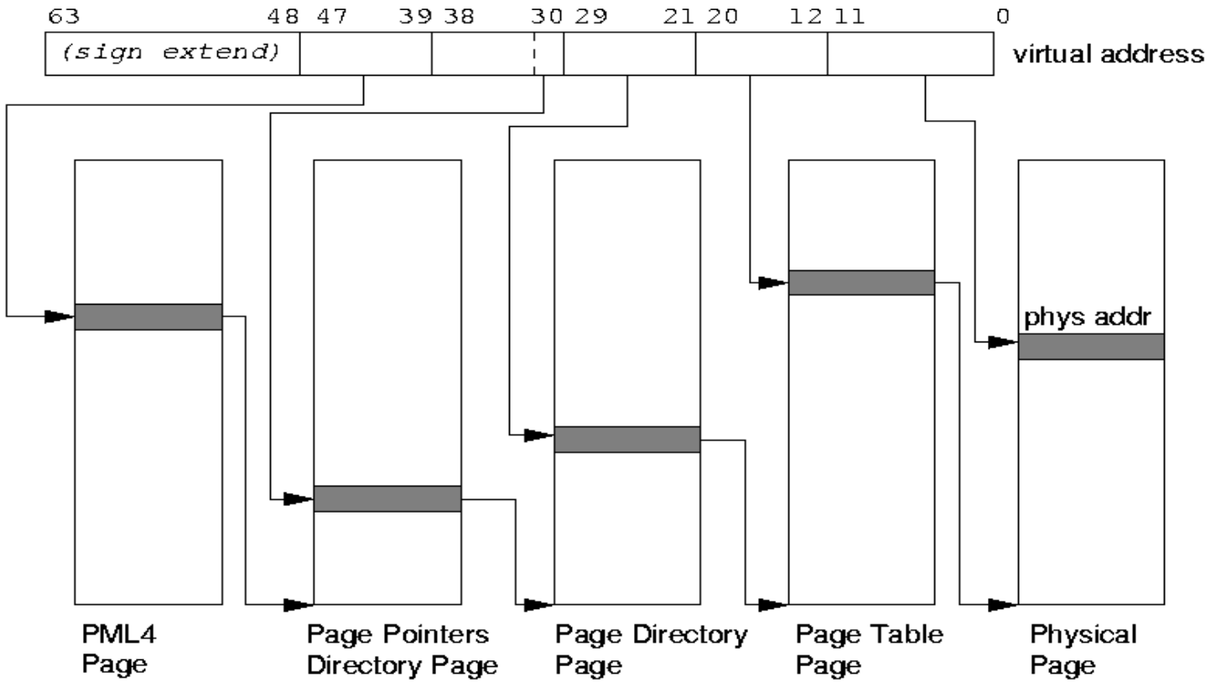
- TLB cache **is small** (~1024 entries, i.e. 4MB)
- TLB **cache miss** is up to 4 memory transfers
- **Spacial locality** is crucial: 1 address outlier is up to 12KB
  …but Linux VMM coalesces memory areas
- **Context switch** of *user-space processes* invalidates TLB



Tempesta Technologies

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

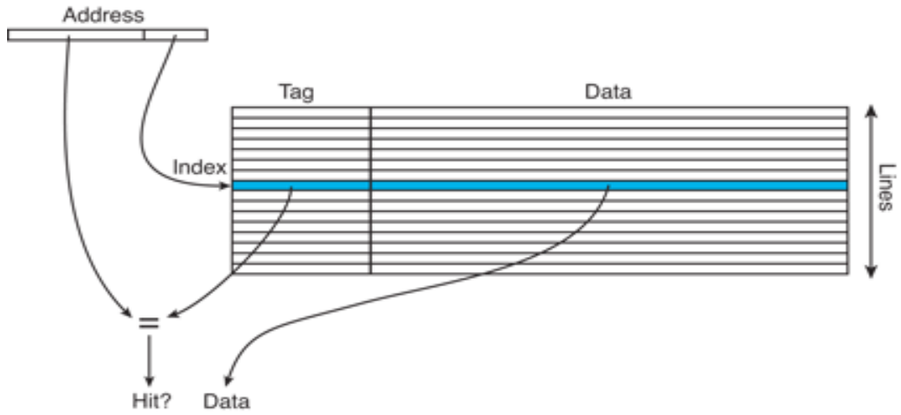


Tempesta Technologies

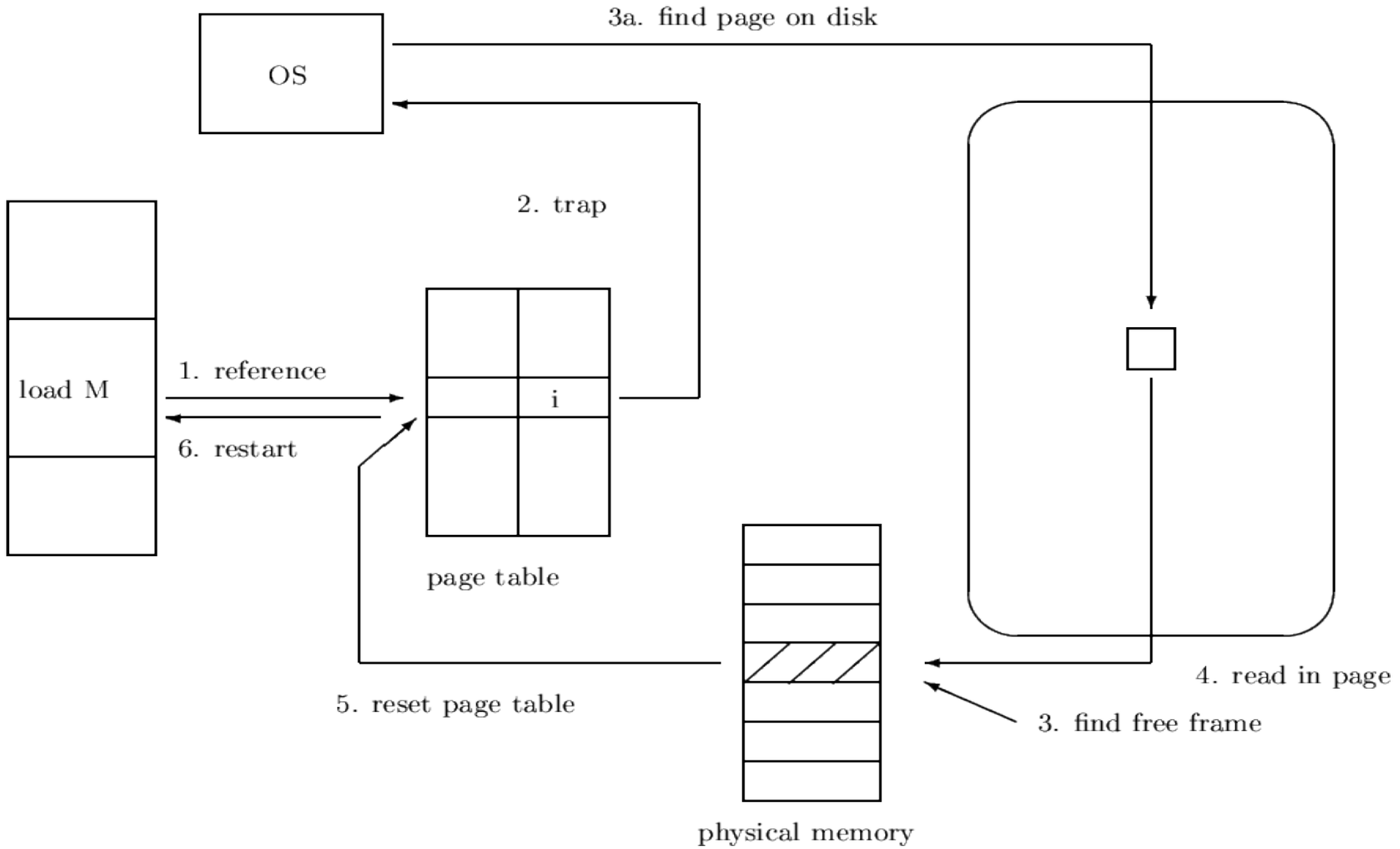
Cache Lookup (x86-64)

- **L1**: **VIPT** (Virtually Indexed Physically Tagged)
- **L2**, **L3**: **PIPT** (Physically Indexed Physically Tagged)

VIPT: L1 cache invalidation on context switch



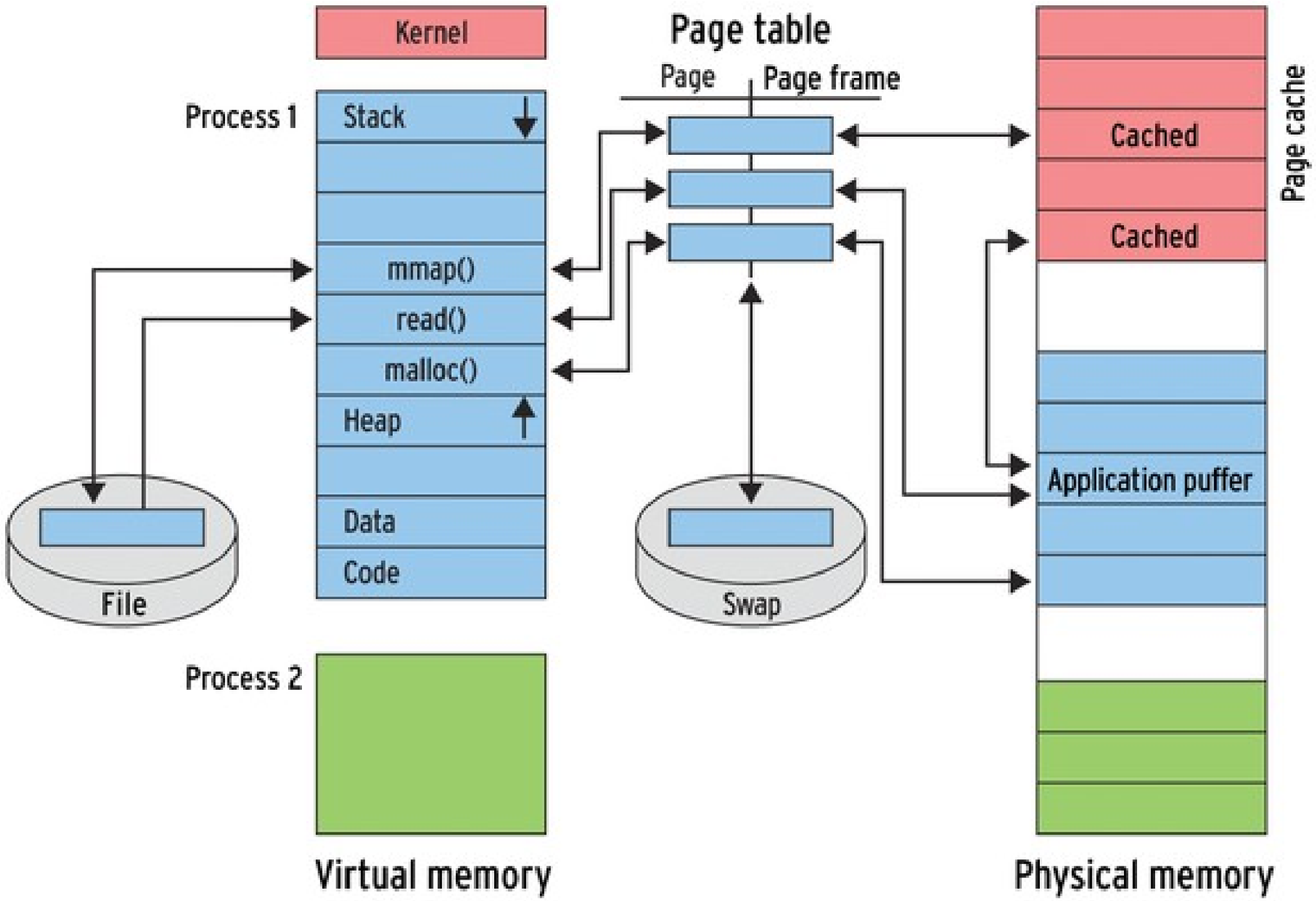
Page fault


3a. find page on disk
OS
2. trap
load M
1. reference
6. restart
i
page table
5. reset page table
4. read in page
3. find free frame
physical memory

Hello world page fault

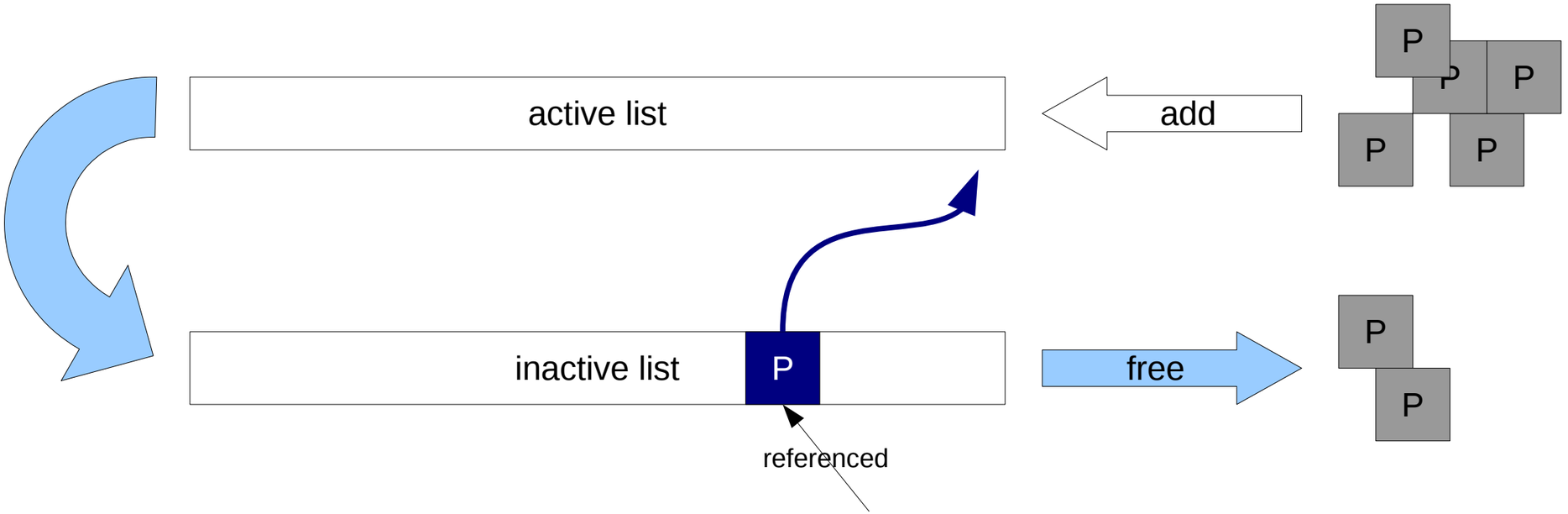
```
int main(int argc, char *argv[])
{
    printf(“Hello world!\n”);
    return 0;
}
```

Linux paging



Page eviction

- Typically current process reclaims memory
- ***kswapd*** – *alloc_pages()* slow path
- **OOM**: usually shrinking active list



VMM lists state

```
$ grep -i active /proc/meminfo
Active:          5447208 kB
Inactive:        2030052 kB
Active(anon):    4465572 kB
Inactive(anon):  1388500 kB
Active(file):     981636 kB
Inactive(file):   641552 kB
```

Managing VMM lists

- Sysctls:
  - *vm.dirty_background_{ratio,bytes}* – dirty page makers are **throttled** and **writeback workers** are started
  - *vm.dirty_ratio* – dirty page maker starts writing **itself**
  - *vm.dirty_expire_centisecs* – when a page becomes **inactive**
  - *vm.dirty_writeback_centisecs* – interval between writebacks by flushers
  - *vm.vfs_cache_pressure* – seems unused now…
  - ...and more in *linux/Documentation/sysctl/vm.txt*

msync(2)

- **man msync**: *“msync - synchronize a file with a memory map”*

msync(2)

- **man msync**: *“msync - synchronize a file with a memory map”*
- *linux/mm/msync.c:*

```
/*
 * MS_SYNC syncs the entire file - including mappings.
 *
 * MS_ASYNC does not start I/O (it used to, up to 2.5.67).
 * Nor does it marks the relevant pages dirty (it used to up to
2.6.17).
 * Now it doesn't do anything, since dirty pages are properly tracked.
 */
```

fsync(2), posix_fadvise(2), madvise(2)

- `fsync(int fd)` - synchronizes the whole file

fsync(2), posix_fadvise(2), madvise(2)

- `fsync(int fd)` - synchronizes the whole file
- `posix_fadvise(int fd, off_t offset, off_t len, int advice)`
  - POSIX_FADV_DONTNEED – invalidate specified pages

fsync(2), posix_fadvise(2), madvise(2)

- `fsync(int fd)` - synchronizes the whole file
- `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;
```

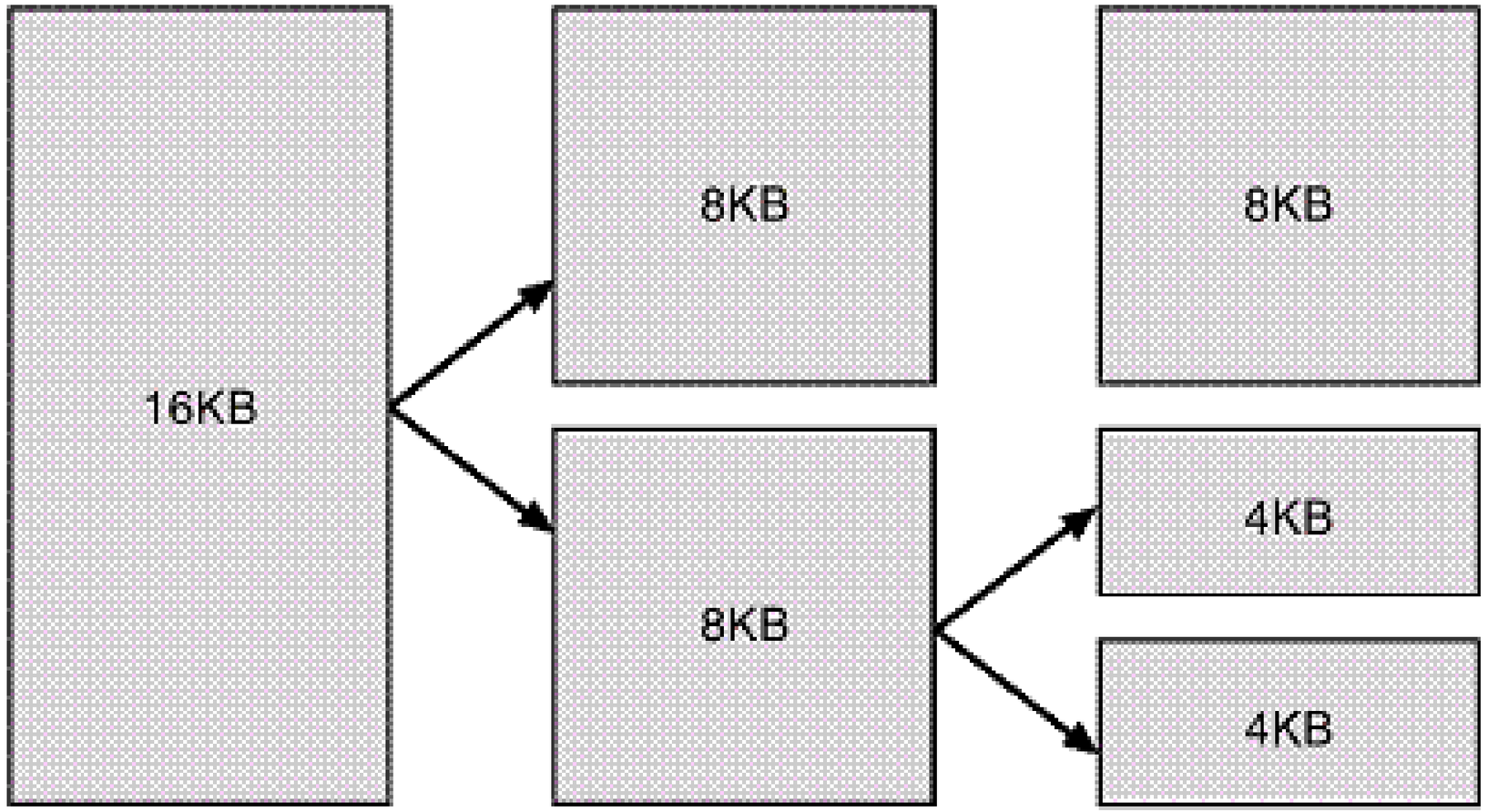
fsync(2), posix_fadvise(2), madvise(2)

- `fsync(int fd)` - synchronizes the whole file
- `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

Buddy allocator



Huge pages

- Huge pages – **2MB** (4KB * 512), gigantic pages – **1GB** (4KB * 512 ^2)

Huge pages

- Huge pages – **2MB** (4KB * 512), gigantic pages – **1GB** (4KB * 512 ^2)
- **Smaller TLB** (~1024 for 4KB, ~32 for 2MB, ~4 for 1GB)

Huge pages

- Huge pages – **2MB** (4KB * 512), gigantic pages – **1GB** (4KB * 512 ^2)
- **Smaller TLB** (~1024 for 4KB, ~32 for 2MB, ~4 for 1GB)
- **Cheaper TLB cache miss** (3-level or 2-level)

Huge pages

- Huge pages – **2MB** (4KB * 512), gigantic pages – **1GB** (4KB * 512 ^2)
- **Smaller TLB** (~1024 for 4KB, ~32 for 2MB, ~4 for 1GB)
- **Cheaper TLB cache miss** (3-level or 2-level)
- Smaller **page table**

Huge pages

- Huge pages – **2MB** (4KB * 512), gigantic pages – **1GB** (4KB * 512 ^2)
- **Smaller TLB** (~1024 for 4KB, ~32 for 2MB, ~4 for 1GB)
- **Cheaper TLB cache miss** (3-level or 2-level)
- Smaller **page table**
- Less number of **page faults**

Huge pages

- Huge pages – **2MB** (4KB * 512), gigantic pages – **1GB** (4KB * 512 ^2)
- **Smaller TLB** (~1024 for 4KB, ~32 for 2MB, ~4 for 1GB)
- **Cheaper TLB cache miss** (3-level or 2-level)
- Smaller **page table**
- Less number of **page faults**
- Poorly supported in VMs (especially 1GB)

```
$ grep 'pse\|pdpe1g' /proc/cpuinfo
```

Compound pages

- Huge pages aren't real pages (***compound*** *pages*):

```
struct page *p, *page = alloc_pages(HUGETLB_PAGE_ORDER);
__SetPageHead(page);
for (p = page + 1; p < page + (1 << HUGETLB_PAGE_ORDER); ++p) {
    p->first_page = page;
    __SetPageTail(p);
}
```

Transparrent huge pages vs hugetlbfs

- **Hugetlbfs** reserves huge pages at *system startup*
  - need page – get page

Transparrent huge pages vs hugetlbfs

- **Hugetlbfs** reserves huge pages at *system startup*
  - need page – get page
- **THS** allocates huge pages in *runtime*
  - VMM does more work on defragmentation
  - Page fault can do more work on trying to allocate huge page

Linux VMM as DBMS engine?

- **Linux VMM**
  - *evicts* dirty pages

Linux VMM as DBMS engine?

- **Linux VMM**
  - *evicts* dirty pages
  - it doesn't know *exactly* whether they're still needed (DONTNEED!)

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

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
    ...but it will be somewhen soon

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
    ...but it will be somewhen soon
  - checkpoint is full database file sync

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
    ...but it will be somewhen soon
  - checkpoint is full database file sync
    ...typically DONTNEED'ed pages are already synced

Thanks!

- E-mail: *ak@natsys-lab.com*
- Blog: *http://natsys-lab.blogspot.com*