

Embedded Linux Discussions

- Idle task implementation issue and others -

TOSHIBA CORPORATION
Core Technology Center
Embedded System Core Technology Development Dept.
KUMGAI Hiroki
Jul 4, 2008

Contents

- **Topics**

- Idle task implementation for MIPS
- XFS log problem / D-Cache aliasing issue
- Timeout handling at thread synchronization
- Kernel priority inversion in futex

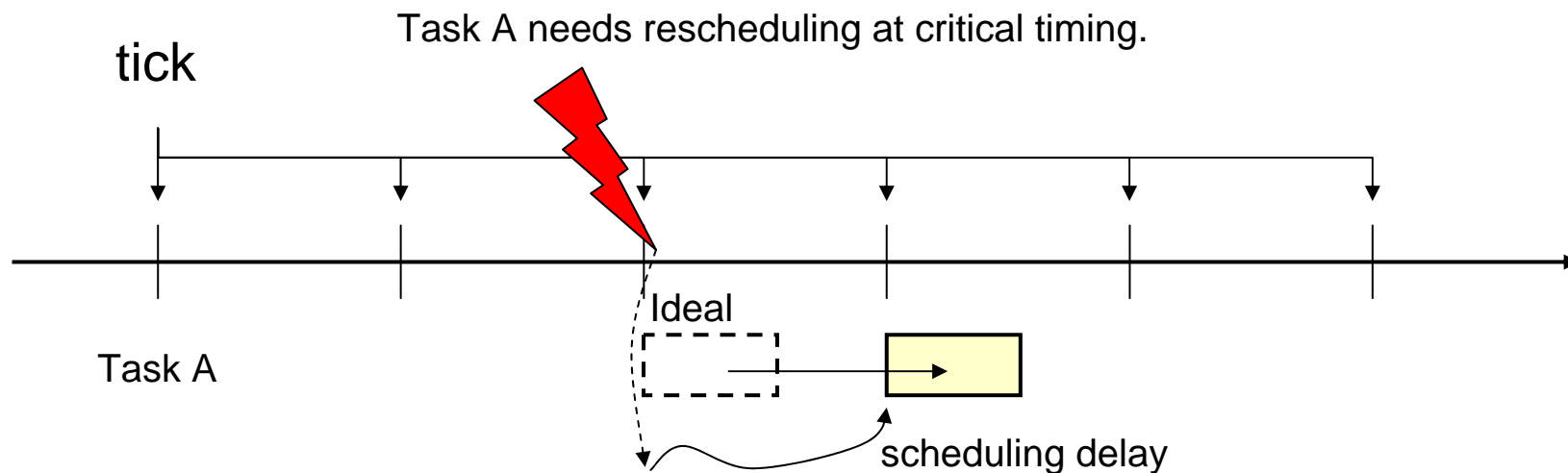
Idle task implementation for MIPS

- **Background**

- In 2.6.15 kernel preemption has been disabled in the idle task.
(<http://lwn.net/Articles/152547/> [sched: disable preempt in idle tasks])

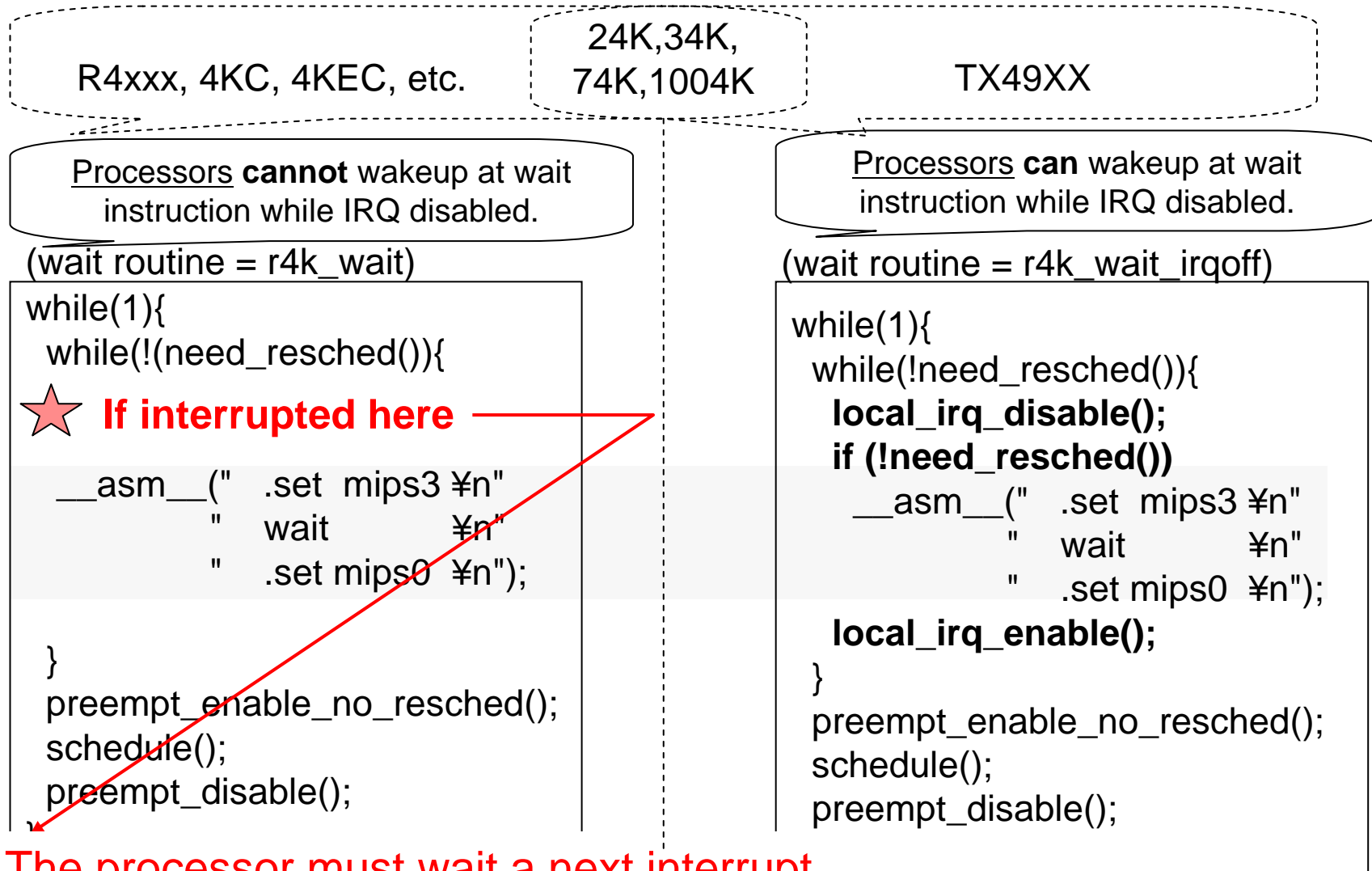
- **Problem**

- This change increases scheduling latency to 1 tick on some MIPS processors.



Idle task implementation for MIPS

- MIPS wait instruction and interrupt handling.**



The processor must wait a next interrupt.

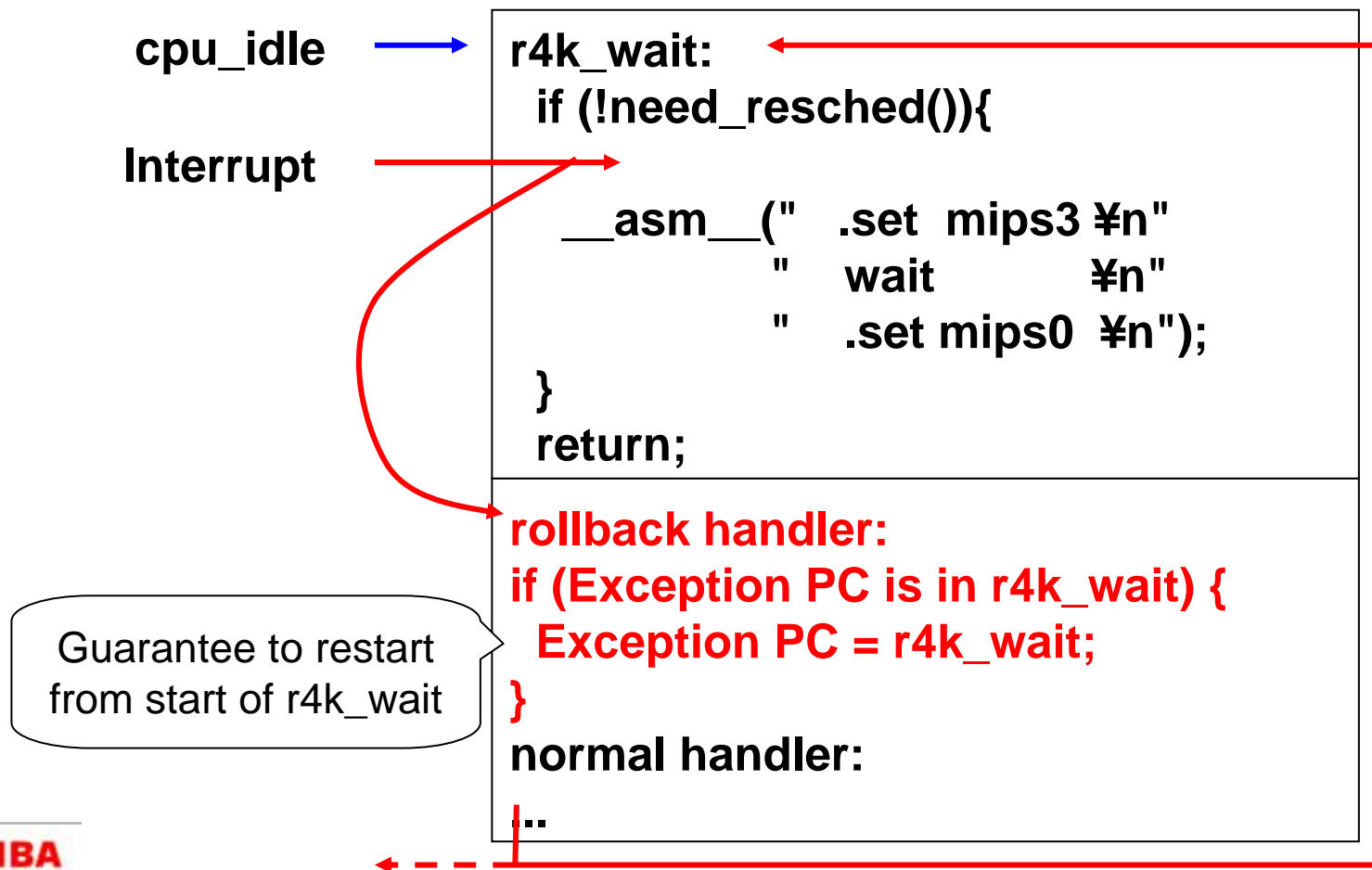
(Ordinary next interrupt would be a tick interrupt.)

Idle task implementation for MIPS

- **Solution**

[Re: WAIT vs. tickless kernel] (not mainlined)

<http://www.linux-mips.org/archives/linux-mips/2007-11/msg00033.html>



Idle task implementation for MIPS

- **Summary**

- If you need resolve the scheduling latency issue, or need new features of tickless kernel and/or hrtimer...

Kernel	Processors	Condition	Status
ver < 2.6.15	all		No problem (*)
ver >= 2.6.15	R4xxx, 4KC, 4KEC, etc.		Patch is required
	24K, 34K, 1004K	(c0 config7 & MIPS_CONF7_WII) == 0	Patch is required
		(c0 config7 & MIPS_CONF7_WII) == 1	No problem (*)
	74K	(c0 prid & 0xff) < 0x44	Patch is required
		(c0 prid & 0xff) >= 0x44	No problem (*)
	TX49XX		No problem (*)

(*) Kernel preemption must be enabled.

XFS log problem

XFS log problem (1)

- **Encountered mount errors after many operations.**

(XFS is a journaling filesystem developed by Silicon Graphics Inc.)

It has been found a simpler problem and it's easy to reproduce.

```
# mkfs.xfs -f /dev/sda1
```

```
# mount -t xfs /dev/sda1 /mnt
```

```
# xfs_logprint /dev/sda1
```

```
xfs_logprint:
```

```
(--- snipped ---)
```

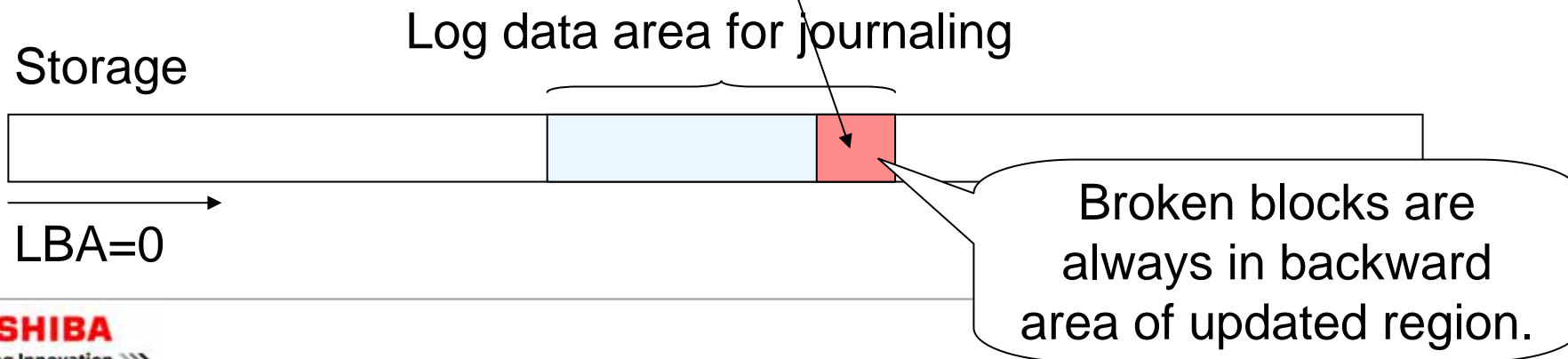
```
*****
```

```
* ERROR: found data after zeroed blocks block=488 *
```

```
*****
```

```
Bad log - data after zeroed blocks
```

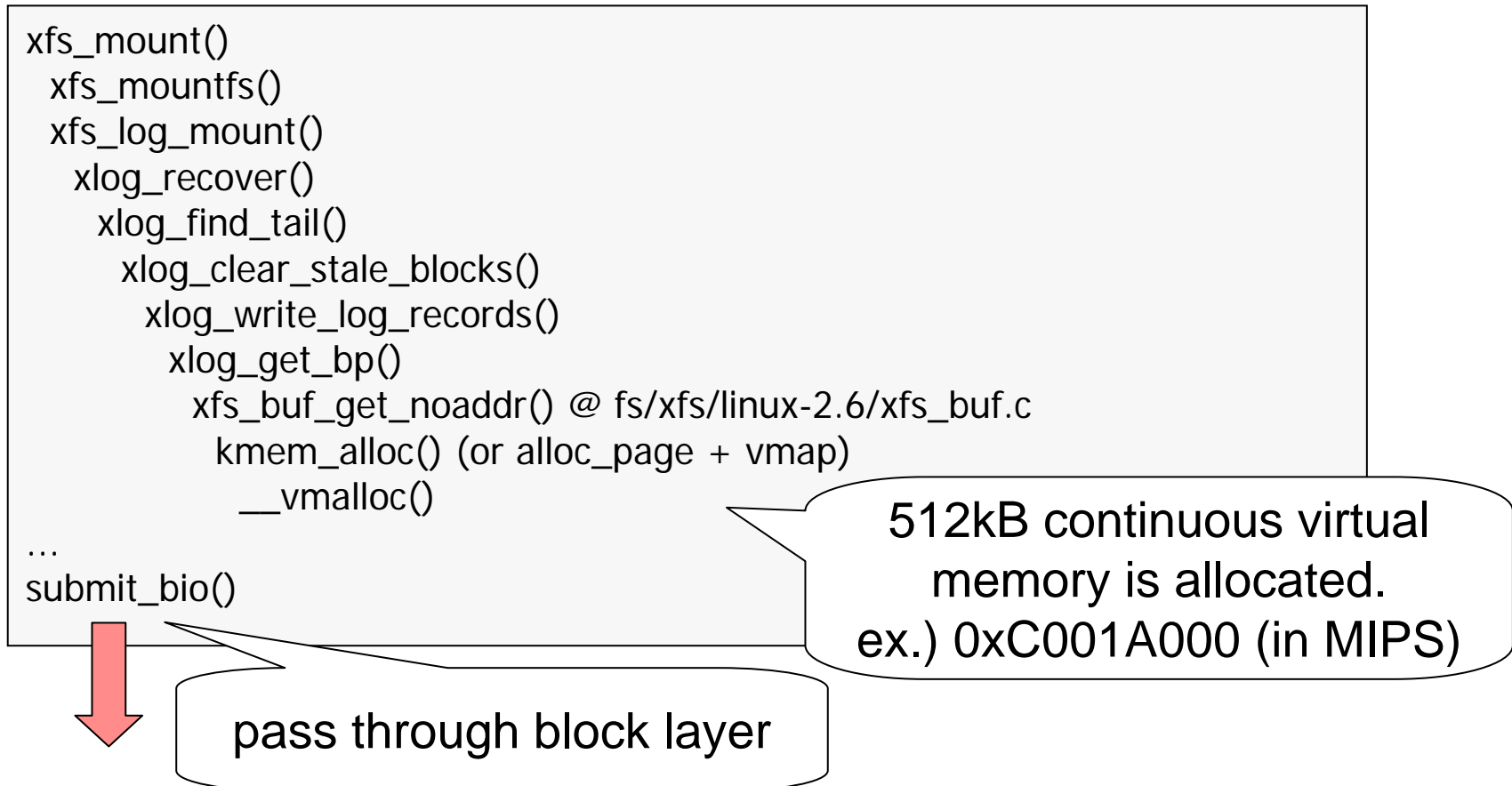
It can be mounted successfully, but error is reported by log dump tool.



XFS log problem (2)

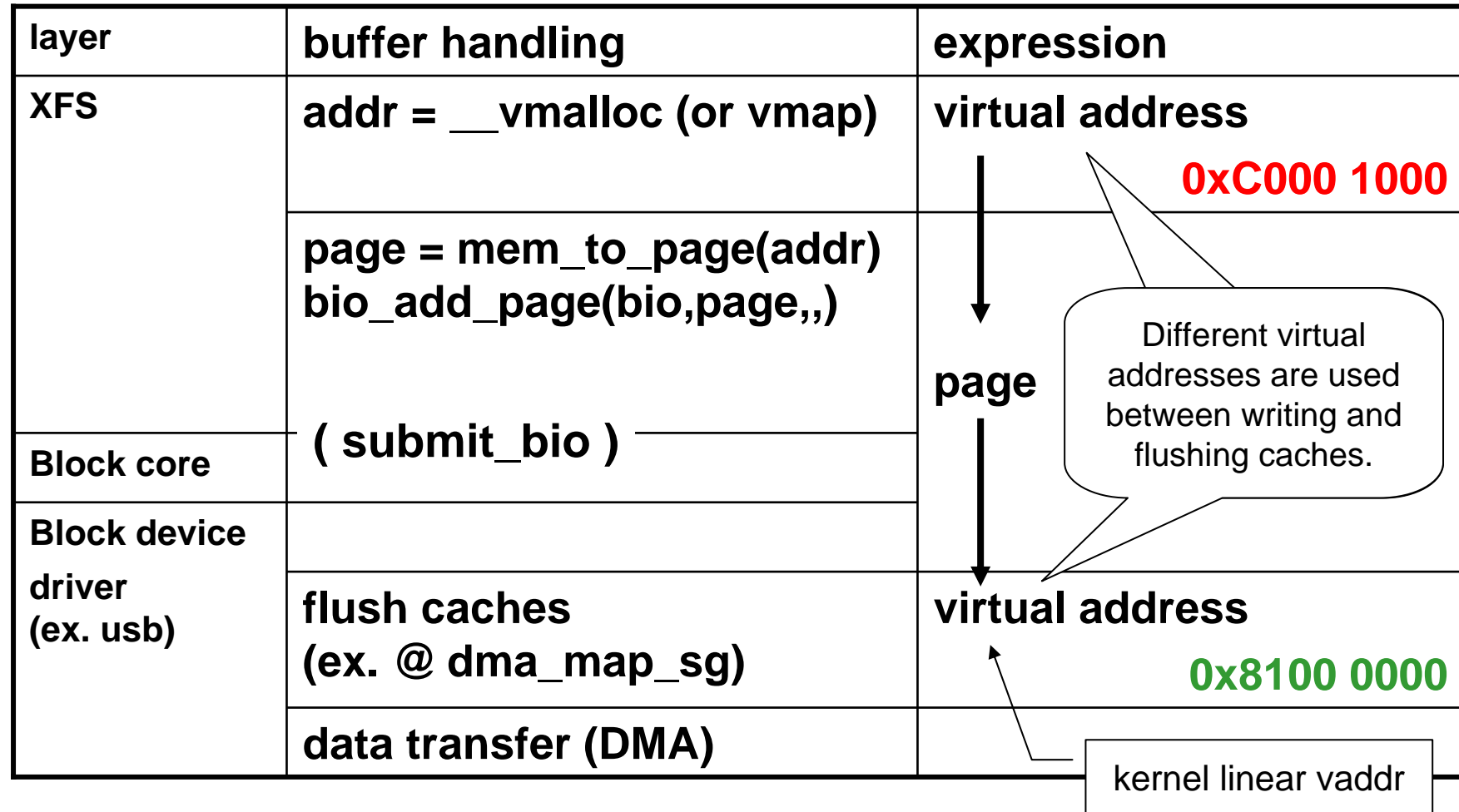
- **Write at mount time**

- Clean logs are written.



XFS log problem (3)

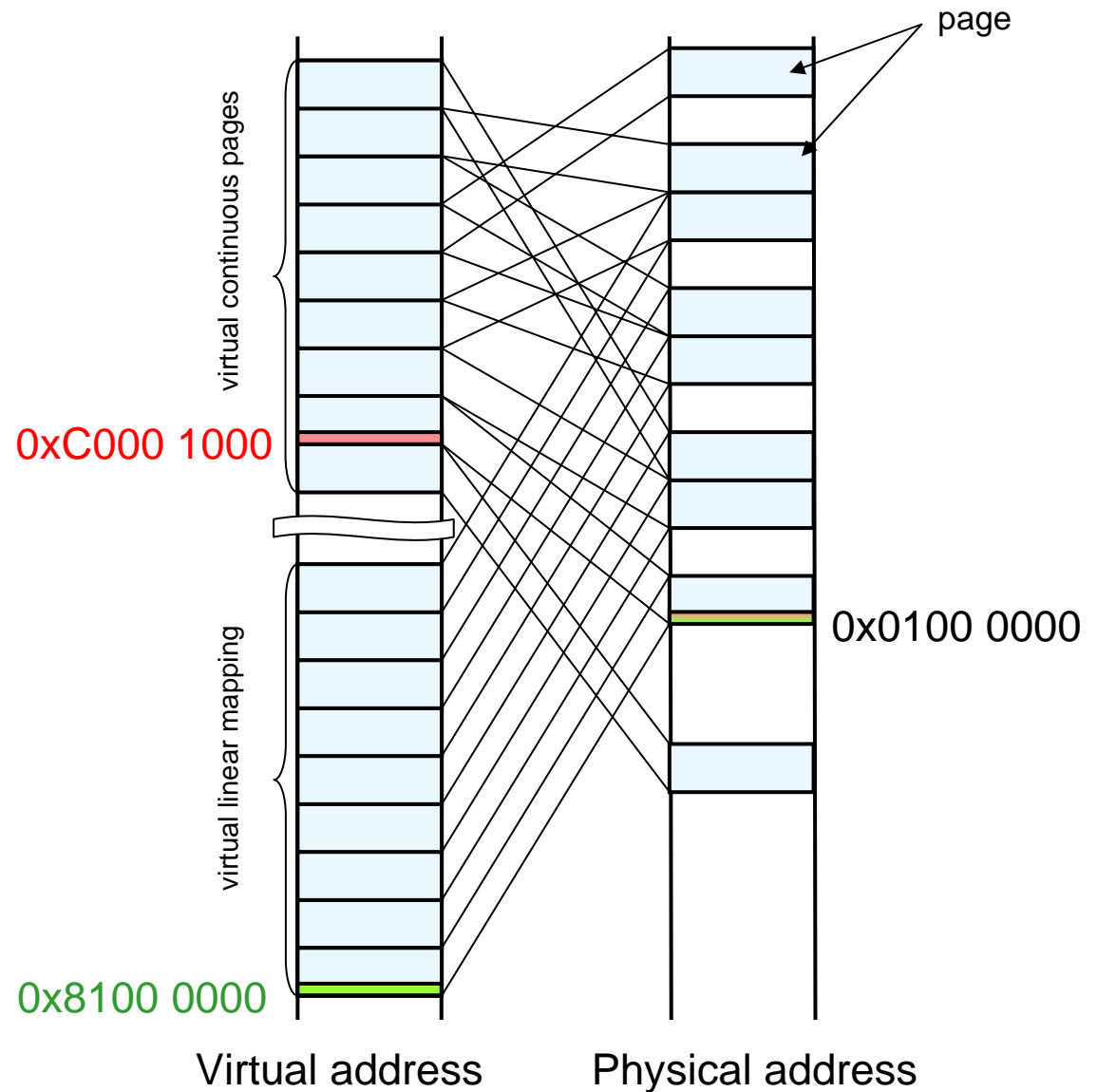
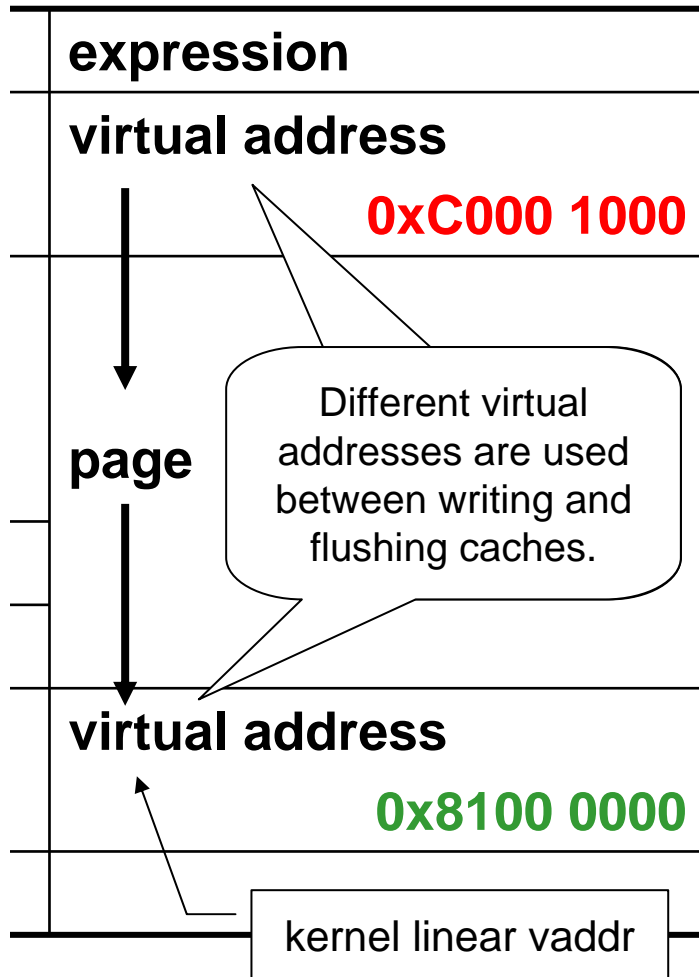
- **Data flow**



➡ **D-cache aliasing issue**

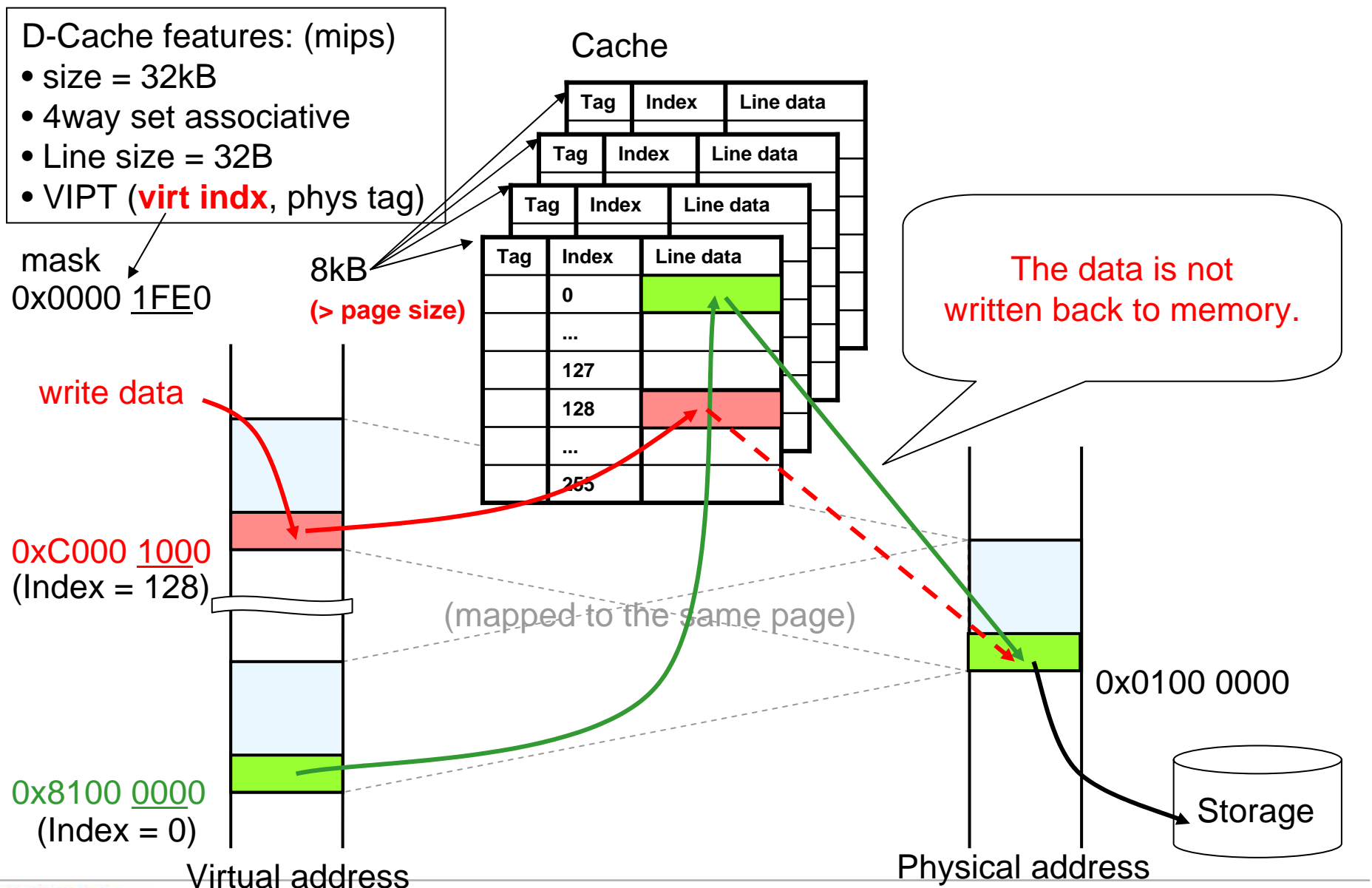
XFS log problem (4)

- Page mapping




D-Cache aliasing issue

(see: Documents/cachetlb.txt)



XFS log problem (5)

- Solution**

layer	buffer handling	expression
XFS	addr = __vmalloc (or vmap)	virtual address 0xC000 1000
	page = mem_to_page(addr)	
	__flush_anon_page(page,addr)	page  add flush routine
	bio_add_page(bio,page,,)	
Block core	(submit_bio)	
Block device (ex. usb)		
	flush caches (ex. @ dma_map_sg)	virtual address 0x8100 0000
	data transfer (DMA)	

TO * __flush_anon_page is available for mips above 2.6.21
Leading Innovation >>>

XFS log problem (6)

- **Cache aliasing summary** (only for relevant processors)

XFS	
All version	<p>Need to flush at XFS.</p> <p>It has been reported on linux-arch ML. [virtual alias problem with vmap and I/O (xfs)] http://www.spinics.net/lists/linux-arch/msg04301.html</p>
FUSE/DirectIO	
<2.6.20(arm) <2.6.21(mips)	<p>Need fixes, see the following. [fuse,get_user_pages,flush_anon_page,aliasing cahces and all that again] http://www.ussg.iu.edu/hypermail/linux/kernel/0612.2/1571.html</p>
>=2.6.20(arm) >=2.6.21(mips)	<p>The above patch has been applied.</p>

Timeout handling at thread synchronization

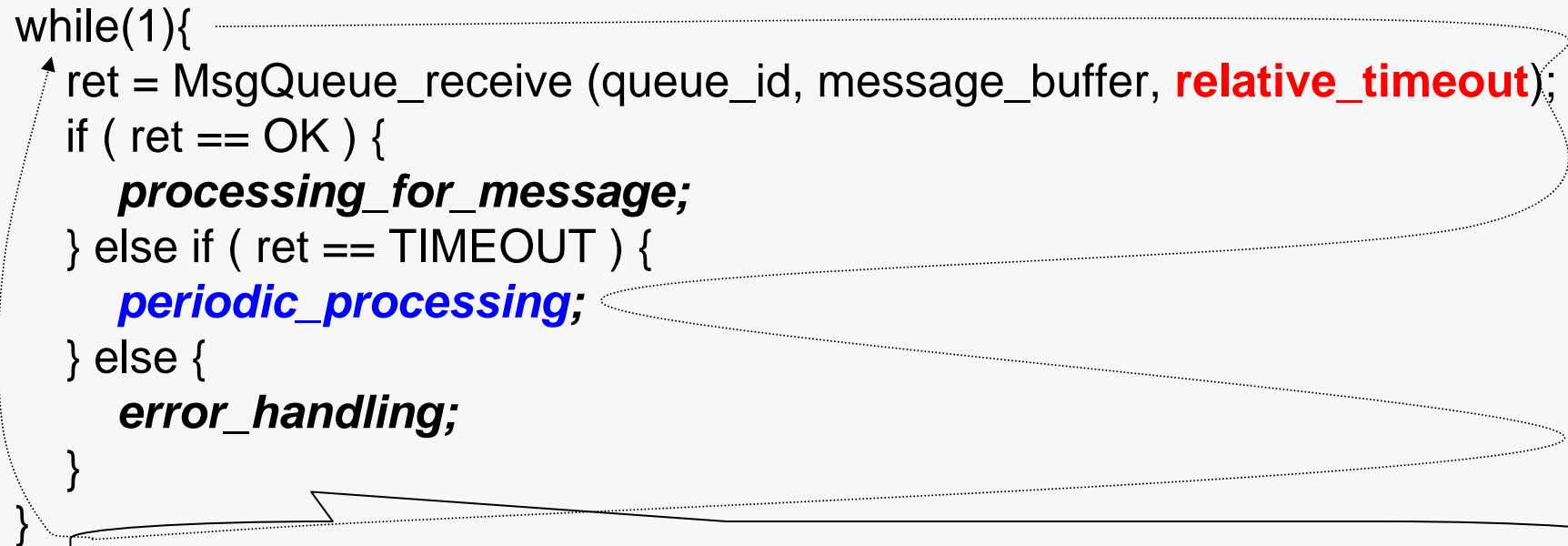
Timeout handling at thread synchronization (1)

- **Background**

- How to realize user-level synchronization methods with relative timeouts on Linux/glibc?
 - And it's maybe required for the reuse of the legacy RTOS application.

For example:

```
while(1){  
    ret = MsgQueue_receive (queue_id, message_buffer, relative_timeout);  
    if ( ret == OK ) {  
        processing_for_message;  
    } else if ( ret == TIMEOUT ) {  
        periodic_processing;  
    } else {  
        error_handling;  
    }  
}
```

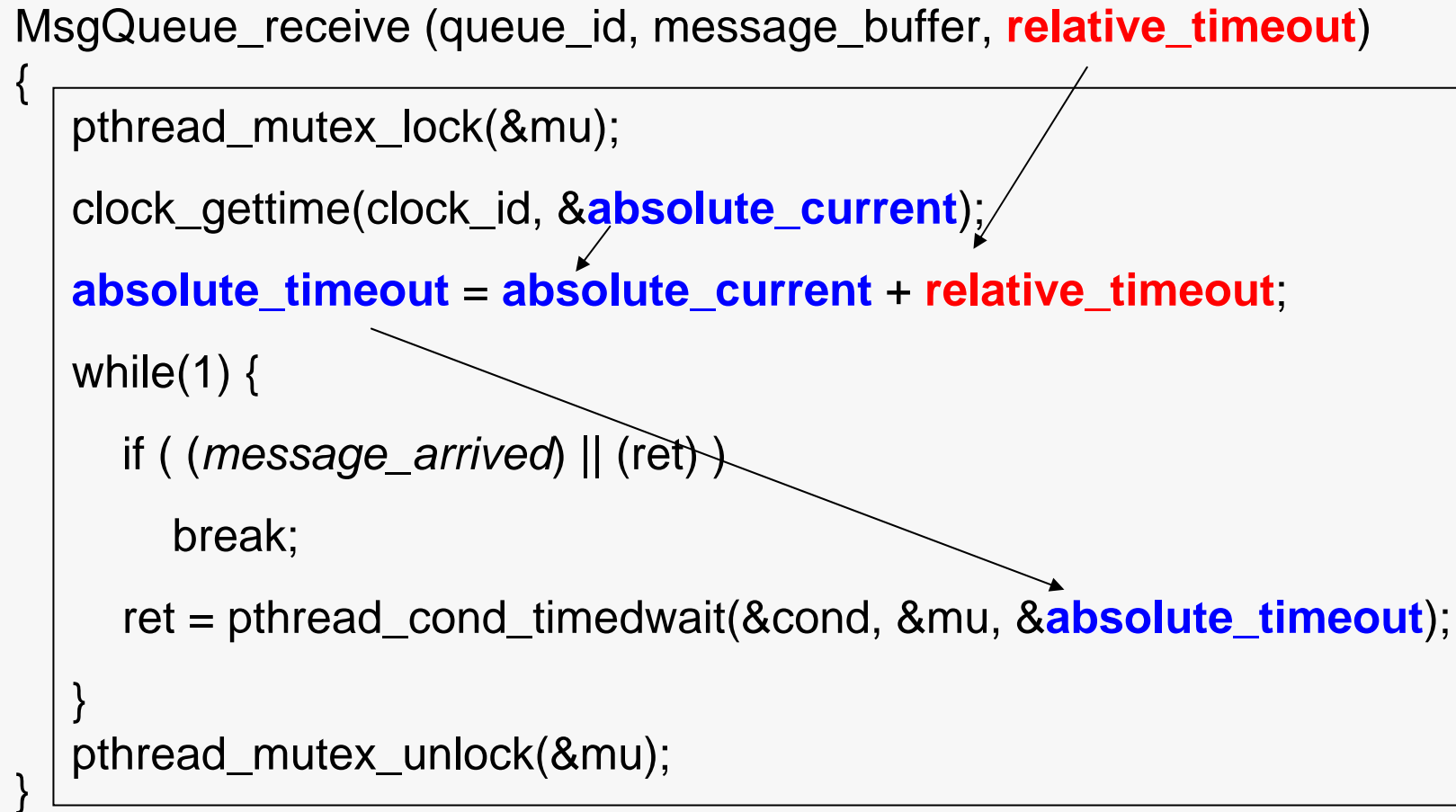


Sometime relative timeout is used for normal processing flow.

Timeout handling at thread synchronization (2)

- **Implement with POSIX interface (with condvar)**

```
MsgQueue_receive (queue_id, message_buffer, relative_timeout)
{
    pthread_mutex_lock(&mu);
    clock_gettime(clock_id, &absolute_current);
    absolute_timeout = absolute_current + relative_timeout;
    while(1) {
        if ( (message_arrived) || (ret) )
            break;
        ret = pthread_cond_timedwait(&cond, &mu, &absolute_timeout);
    }
    pthread_mutex_unlock(&mu);
}
```



Timeout handling at thread synchronization (3)

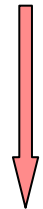
- Expand more...

```
MsgQueue_receive (queue_id, message_buffer, relative_timeout)
{
    clock_gettime(clock_id, &absolute_current);
    absolute_timeout = absolute_current + relative_timeout;
    ret = pthread_cond_timedwait(&cond, &mu, &absolute_timeout);

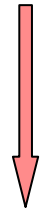
    pthread_cond_timedwait(cond, mu, absolute_timeout)
    {
        clock_gettime(clock_id, &absolute_current);
        relative_timeout = absolute_timeout - absolute_current;
        futex(FUTEX_WAIT, relative_timeout);
    }
}
```

axis

rel



abs

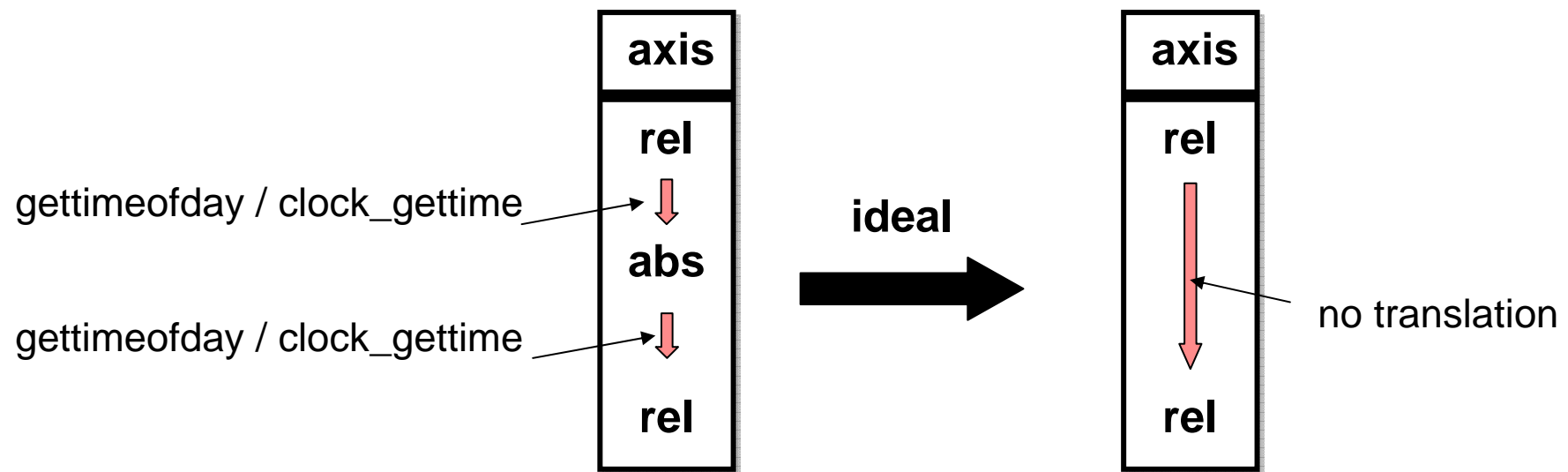


rel

Timeout handling at thread synchronization (4)

- **Problem**

- Wasteful translations between time axes.
- Disturbed by `settimeofday` / `clock_settime`.
 - This leads to incorrect timeout.



ref.) Rationale for the Monotonic Clock

http://www.opengroup.org/onlinepubs/009695399/xrat/xsh_chap02.html#tag_03_02_08_19

Timeout handling at thread synchronization (5)

- **Solution**

A) Use `clock_id = CLOCK_MONOTONIC`.

B) Implement non-portable "re timed_np" APIs with relative timeout values.

Timeout handling at thread synchronization (6)

A) Use `clock_id = CLOCK_MONOTONIC`.

This resolves disturbances by `settimeofday` / `clock_settime`.
However it's applicable only for `pthread_cond_timedwait`.

```
clock_id = CLOCK_MONOTONIC;  
pthread_condattr_setclock(&condattr, clock_id);  
pthread_cond_init(&cond, &cond_attr)
```

```
pthread_mutex_lock(&mu);  
clock_gettime(clock_id, &absolute_current);  
absolute_timeout = absolute_current + relative_timeout;  
while(1) {  
    pthread_cond_timedwait(&cond, &mu, &absolute_timeout);  
}  
pthread_mutex_unlock(&mu);
```

* `CLOCK_MONOTONIC` clock is never changed by `settimeofday` or `clock_settime` - because it's a spec.

Timeout handling at thread synchronization (7)

B) Implement non-portable "reltimed_np" APIs with relative timeout values.

(There are in Solaris,

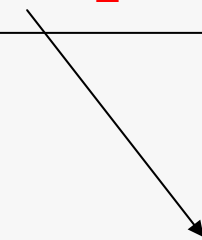
<http://docs.sun.com/app/docs/doc/816-5168/6mbb3hr05?l=ja&a=expand>)

ex.)

- sem_reltimedwait_np
- pthread_cond_reltimedwait_np
- pthread_mutex_reltimedlock_np

```
MsgQueue_receive (queue_id, message_buffer, relative_timeout)
{
    pthread_mutex_lock(&mu);
    while(1) {
        if ( message_arrived || (ret) )
            break;
        ret = pthread_cond_reltimedwait_np(&cond, &mu, &relative_timeout);
    }
    pthread_mutex_unlock(&mu);
}
```

(returns remaining time)



Timeout handling at thread synchronization (8)

- **Summary**

	CLOCK_MONOTONIC	reltimed_np
Support APIs	only pthread_cond_timedwait	Implementation depend
Additional implementation	None	glibc/NPTL, kernel/futex
Performance		Less overhead (no translation)
Settimeofday, clock_settime disturbance	No problem	No problem

Kernel priority inversion in futex

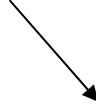
Kernel priority inversion in futex (1)

- **Problem**

- It has been observed deadlocks on the same workload in NPTL (@2.6.20.19) but not in linuxthreads (@2.6.10).

The contention is on task->mm->mmap_sem in kernel.

futex operations

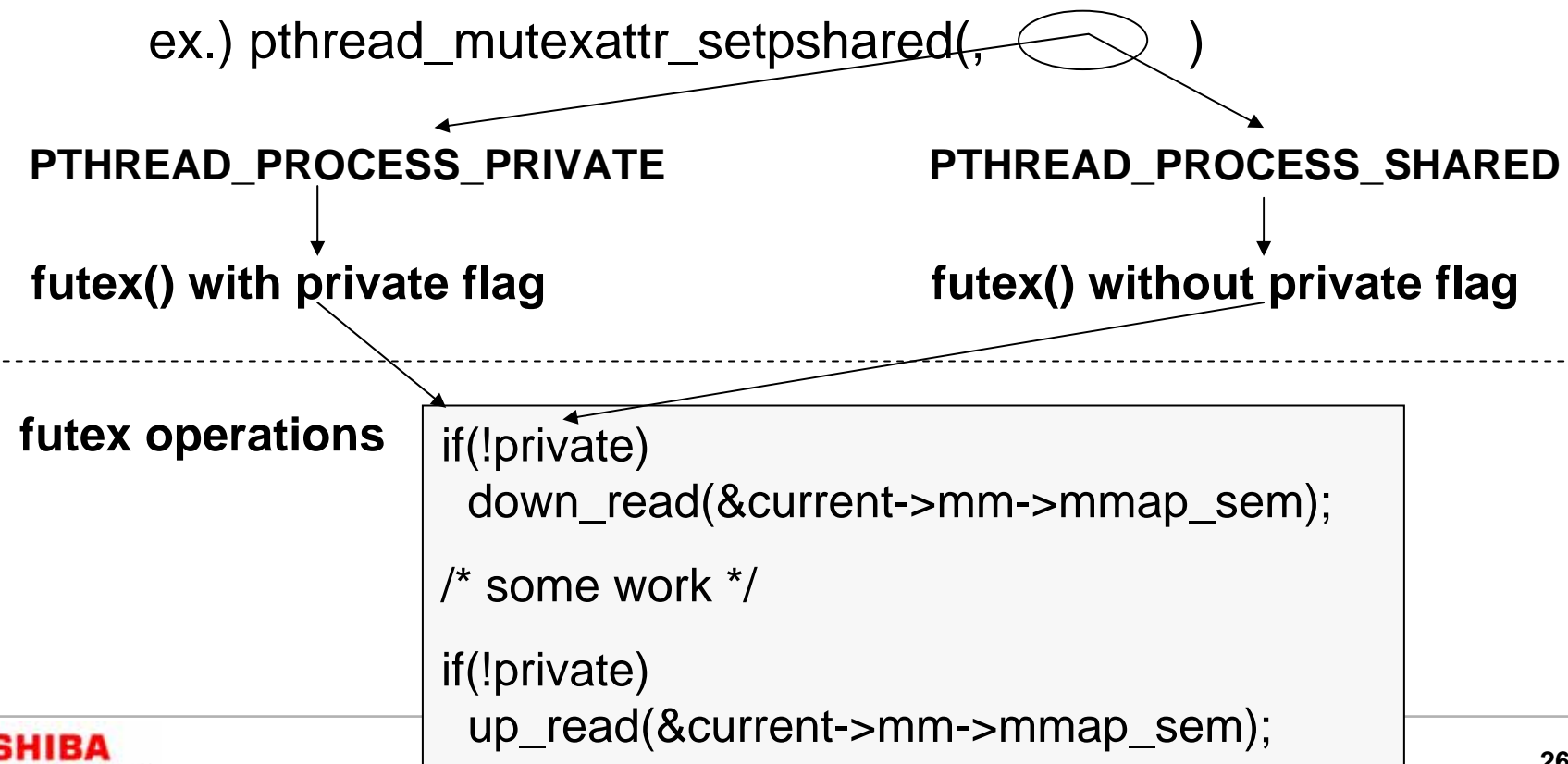


```
down_read(&current->mm->mmap_sem);  
/* some work */  
up_read(&current->mm->mmap_sem);
```

Kernel priority inversion in futex (2)

- **Changes in later kernel**

- This semaphore has been changed to lock conditionally at <http://lkml.org/lkml/2007/4/11/100> (merged in 2.6.22). If the futex is not used on multi processes, the lock can be omitted.



Kernel priority inversion in futex (3)

- **Summary**

version	use case of futex	solution
~2.6.21	only private	Apply the patch of http://lkml.org/lkml/2007/4/11/100 , and force to set FUTEX_PRIVATE_FLAG.
	process shared	No solution (RT patch!?)
2.6.22~ glibc 2.7~	only private	No problem
	process shared	No solution (RT patch!?)

Summary

- **Some not fully resolved issues are explained with some solutions.**
- **Topics**
 - Idle task implementation for MIPS
 - XFS log problem / D-Cache aliasing issue
 - Timeout handling at thread synchronization
 - Kernel priority inversion in futex

TOSHIBA

Leading Innovation >>>