The Kernel Report

Namhyung Kim
LG Electronics

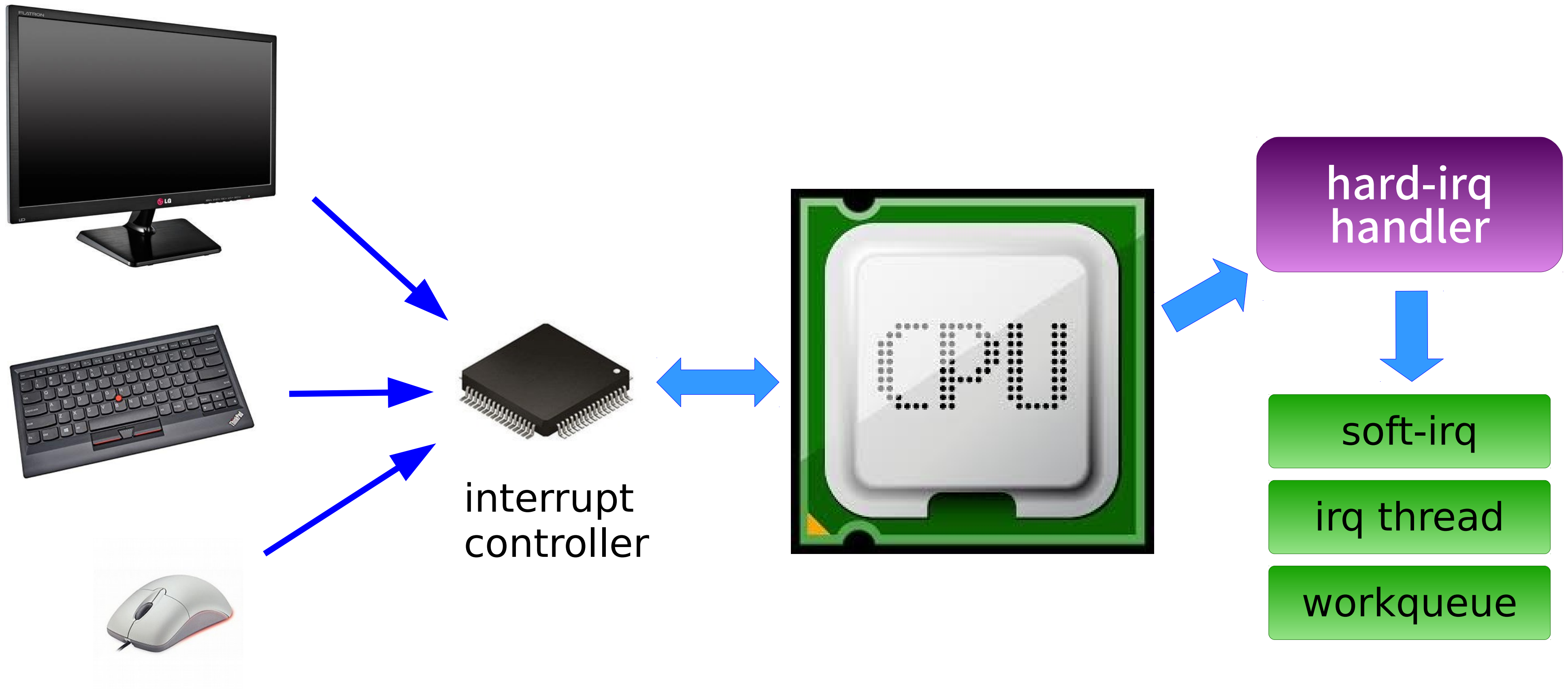
Kernel versions

- **4.16**
  - HRtimer in softirq, printk() lockup prevention, Usercopy whitelisting
- **4.17**
  - Inter-event tracing histogram, scheduler load tracking improvements (+ 4.19)
- **4.18**
  - Restartable sequence, bpfilter, dm-writecache
- **4.19**
  - Async IO polling, block IO latency controller, L1TF

HRtimer in softirq

- **Running (high-resolution) timer function in a soft-irq context instead of hard-irq**
  - `HRTIMER_MODE_{ABS,REL}_SOFT`
- **Hard-irq vs Soft-irq**
  - Hardirq handler runs with irq-disabled
  - Most of work deferred to the bottom-half
    - softirq, tasklet or workqueue
    - Or threaded-irq handler (rt-task)

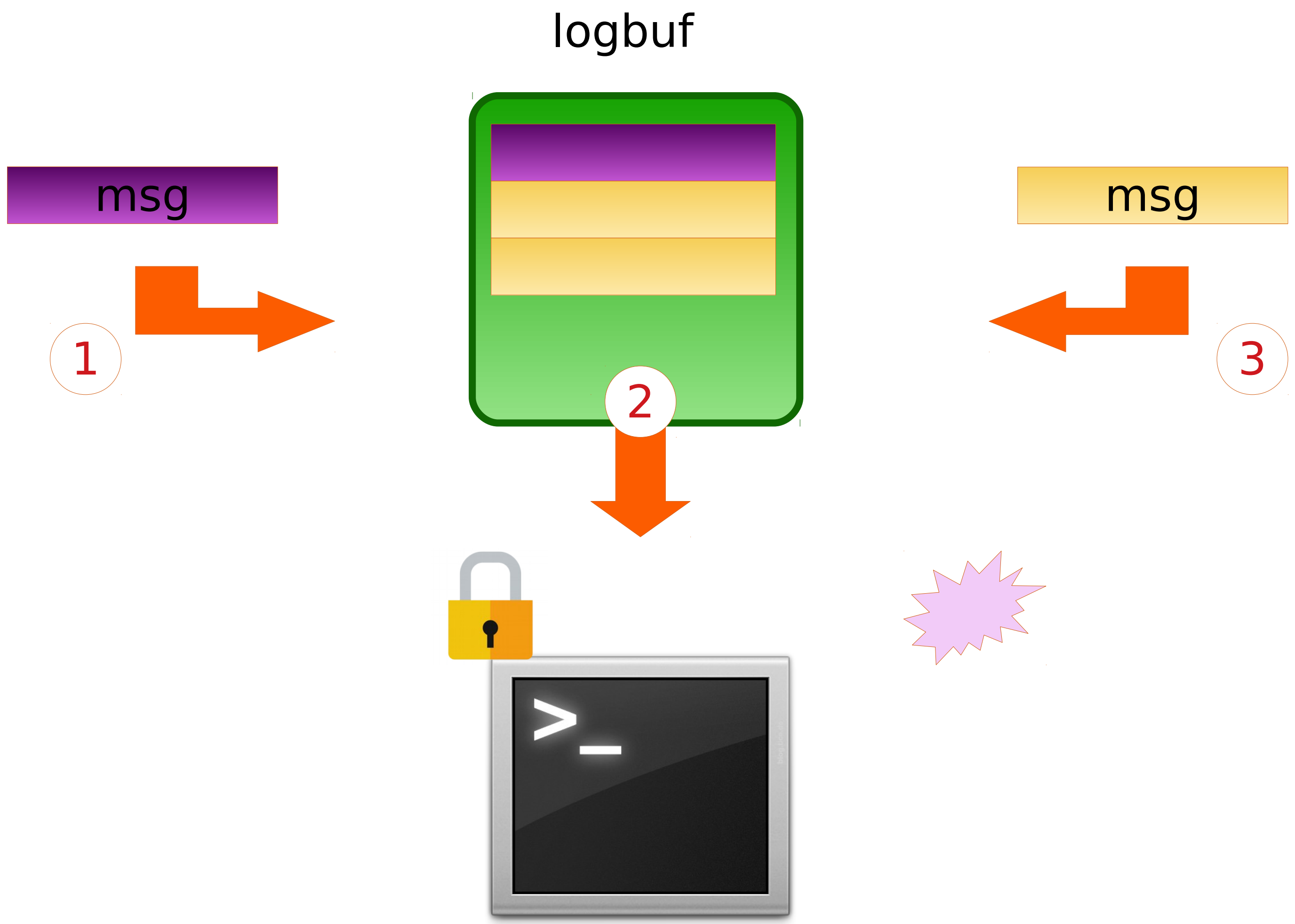
Interrupt handling



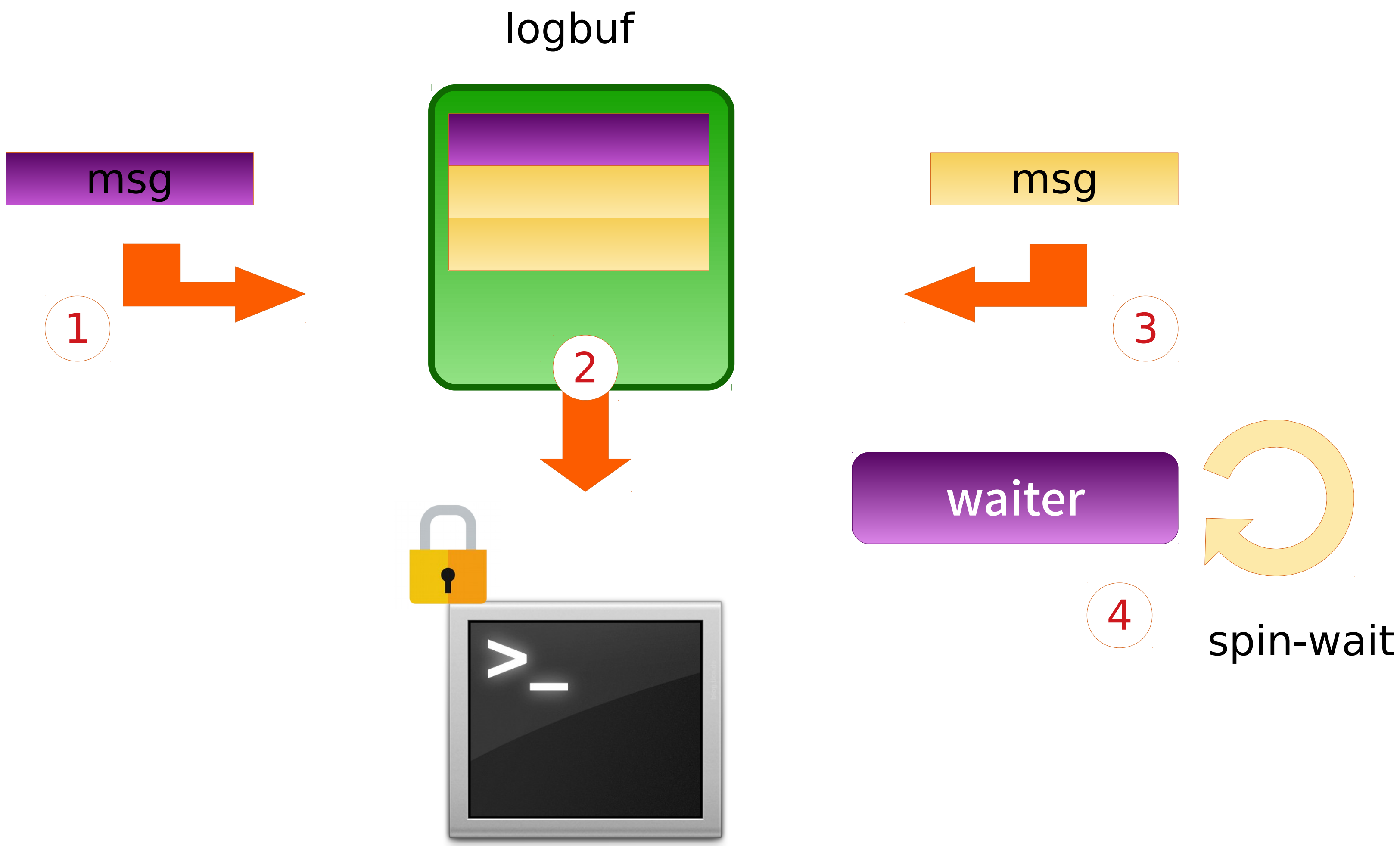
printk() lockup prevention

- **Lockup senario**
  - First printk() will output all contents of logbuf
  - A burst of printk() will soft-lockup the first caller if it’s faster than console ouput
- **Solution**
  - New code will change responsibility of output to next caller

printk: before



printk: After



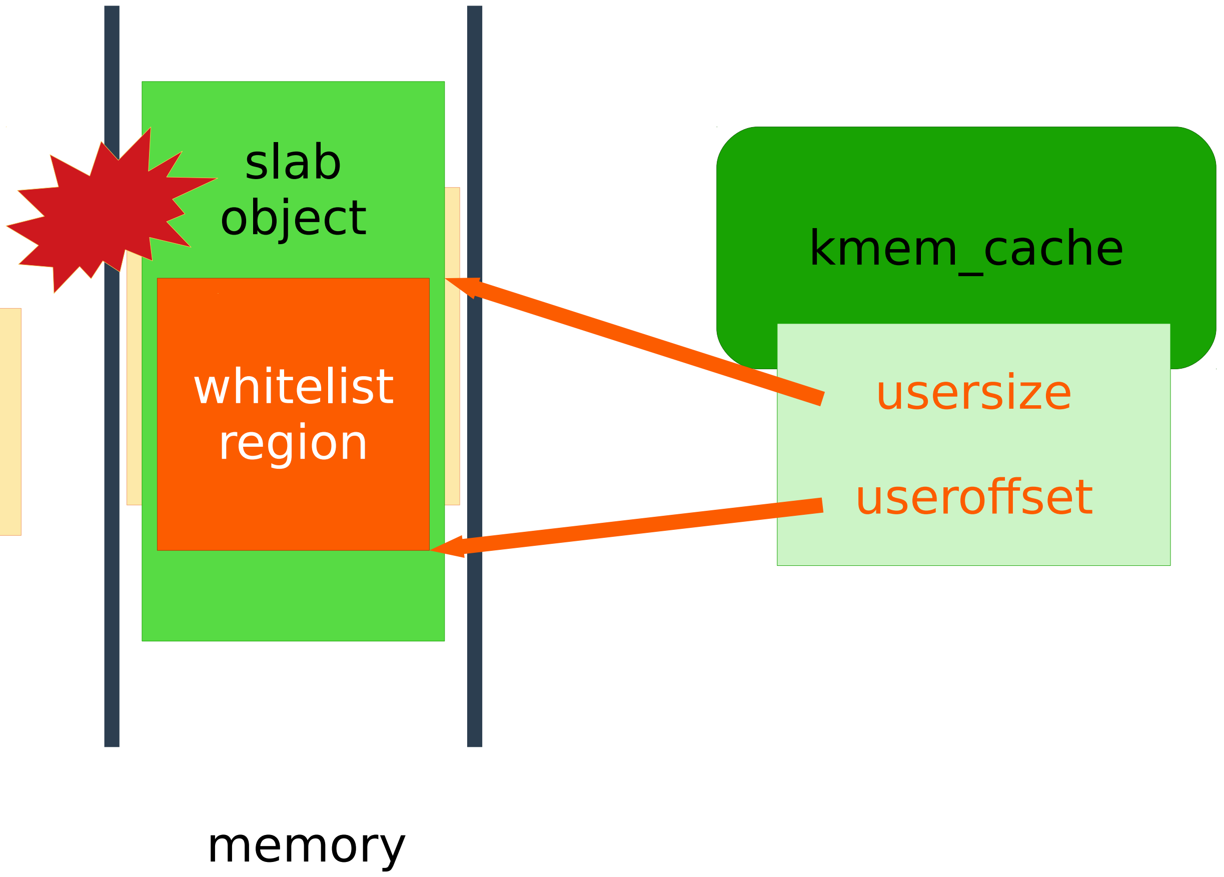
Usercopy whitelisting

- **Security feature**
  - Limit range of interaction between user and kernel in a slab object

```
struct kmem_cache *
kmem_cache_create_usercopy(const char *name,
             unsigned int size, unsigned int align,
             slab_flags_t flags,
             unsigned int useroffset, unsigned int usersize,
             void (*ctor)(void *));
```

Usercopy whitelisting

```
copy_to_user()
copy_from_user()
```



Scheduler load tracking

- **PELT**
  - Per-Entity Load Tracking
  - Keep (cpu) util for each task (sched-entity)
- **UTIL_EST (4.17)**
  - Save util value when task goes to sleep
- **IRQ/Real-time utilization (4.19)**
  - For schedutil cpufreq governor

Load Tracking

- **Load = CPU utilization**
- **Task util**
  - RUNNING / time (moving average)
  - Task placement decision
- **CPU util**
  - Sum of task utils
  - Load balancing decision
  - cpufreq decision

Util Estimation

- **UTIL_EST**
  - Enqueued : task util at the time of dequeue
  - Ewma : exp. Weighted moving average of ‘enqueued’. (task only)
- **Use max value of task util**
  - Insensitive to transient changes
  - Better estimate a long slept big task
  - FAIR tasks only

Inter-event tracing histogram

- **Inter-event tracing**
  - Tracing two or more events and calculate value from those events
  - Create new (synthesized) event
- **Tracing histogram**
  - don’t save all trace record
  - Aggregate result using given keys

Tracing histogram

- **To know distribution of values**
  - Just count the event (using keys)
  - don’t waste buffer and no need to copy

`* kmalloc event histrogram`

Key: pid	hitcount	Value: bytes_alloc
1234	1	128
5678	3	768
13579	100	6400

Inter-event tracing

- **Inter-event tracing**
  - Variable support : save last value
  - Trigger action: do something if event match
  - Synthesized event: create new event runtime
- **Wakeup latency**
  - Time between task wake and schedule
  - Using inter-event tracing
  - Histogram on the synthesized event

Wakeup Latency example

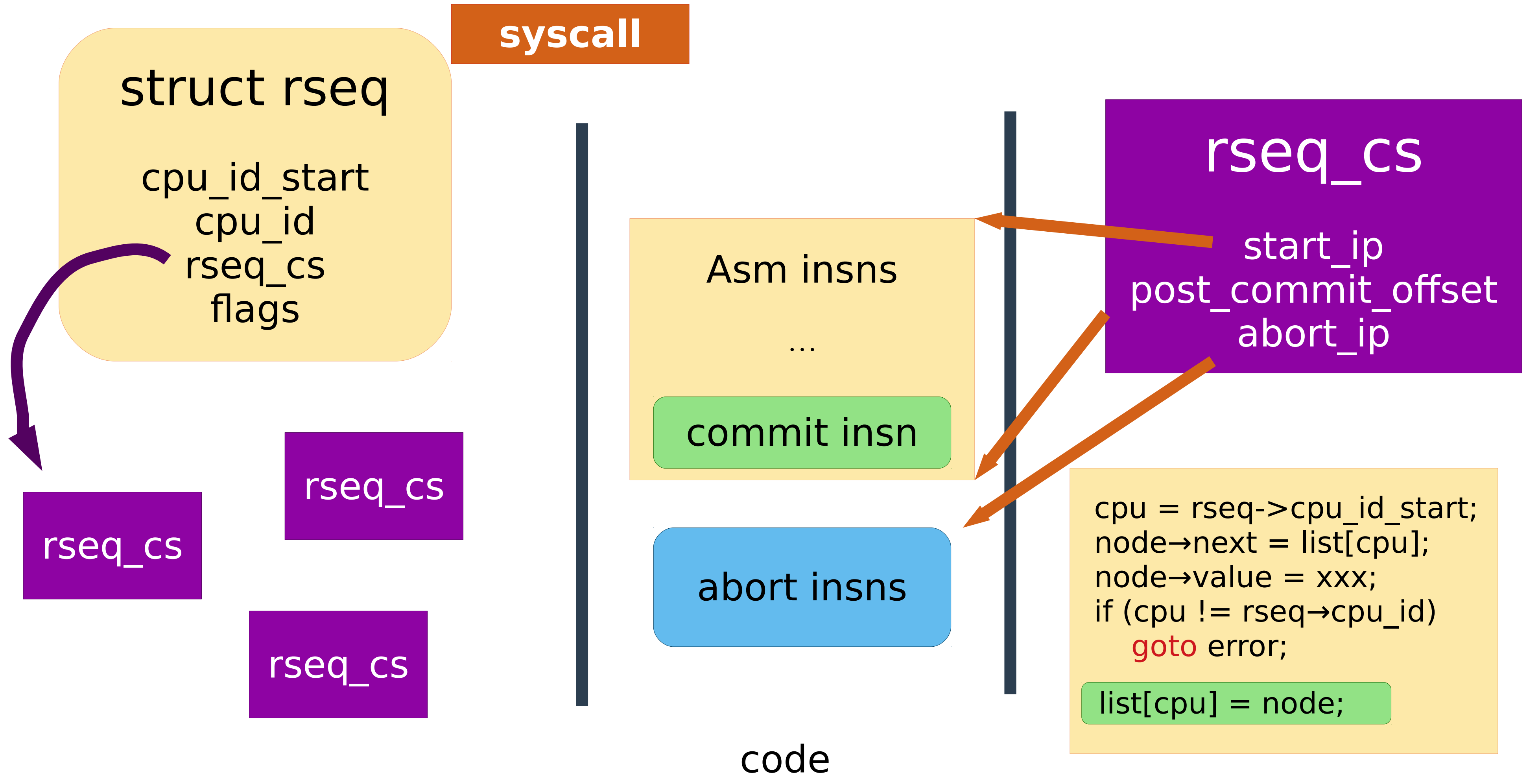
```
# cd /sys/kernel/debug/tracing

# echo ‘wakeup_latency u64 lat; pid_t pid; int prio’ \
      >> synthetic_events

# echo ‘hist:key=lat.log2’ \
      >> events/synthetic/wakeup_latency/trigger

# echo ‘hist:key=pid:t0=common_timestamp.usecs if comm==”cyclictest”’ \
      >> events/sched/sched_waking/trigger

# echo ‘hist:key=next_pid:lat=common_timestamp.usecs-$t0: \
        onmatch(sched.sched_waking). \
        wakeup_latency($lat,next_pid,next_prio) \
        if next_comm==”cyclictest” \
      >> events/sched/sched_switch/trigger

# cat events/synthetic/wakeup_latency/hist
```

Restartable Sequence

- **Fast Lockless Synchronization**
  - Make fast-path faster
  - Avoid atomic instruction
  - Speed up by using per-cpu data
- **New syscall**

```
long rseq(struct rseq *rseq, uint32_t len,
          int flags, uint32_t sig);
```

Restartable Sequence

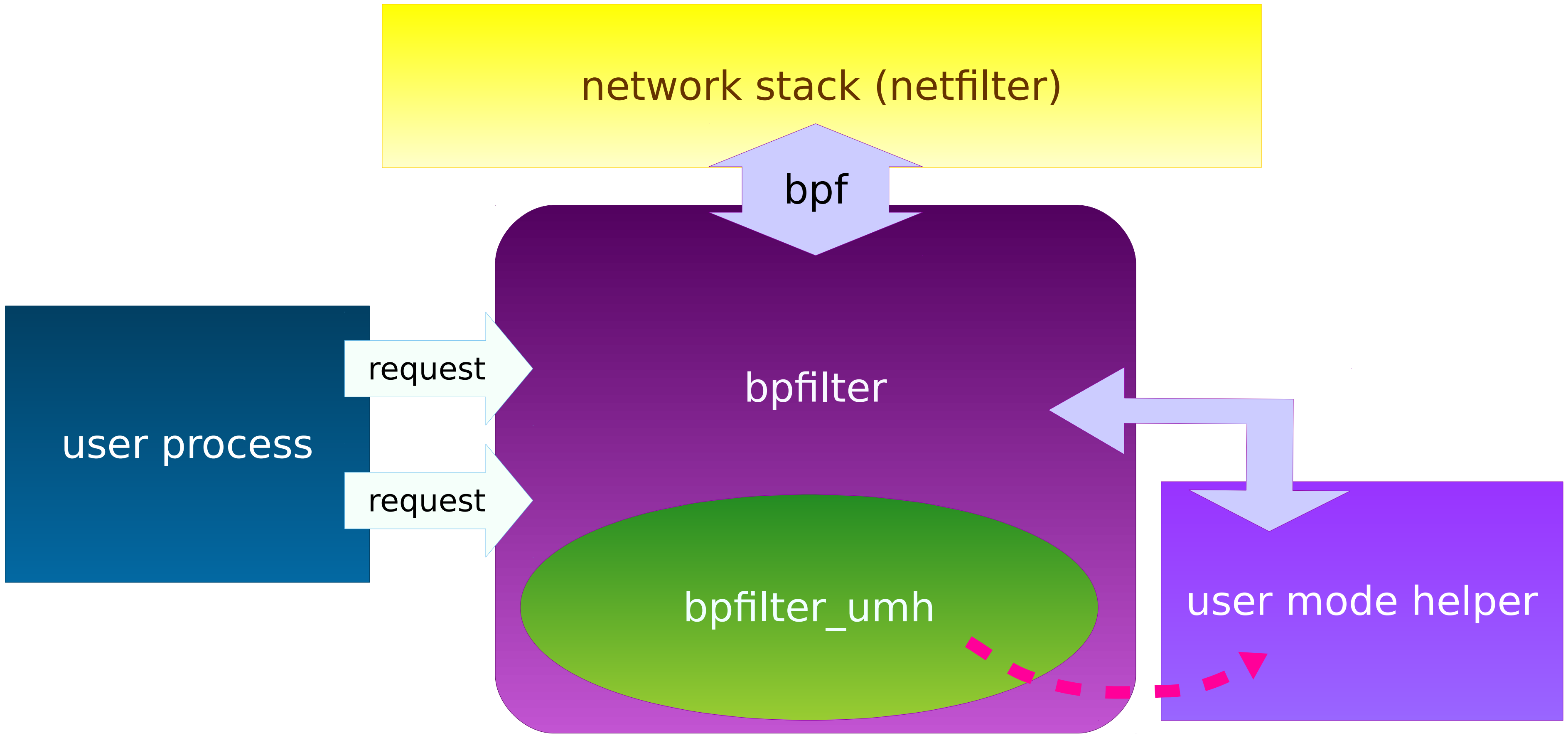


```
cpu = rseq->cpu_id_start;
node→next = list[cpu];
node→value = xxx;
if (cpu != rseq→cpu_id)
    goto error;
list[cpu] = node;
```

bpfilter

- **Netfilter with BPF**
  - Replace iptables, nftables
  - Fast, safe vm in kernel
  - Initial work just started
- **User-mode helper**
  - Do complex jobs in user space
  - Kernel module include an ELF binary
  - Communicate via pipe

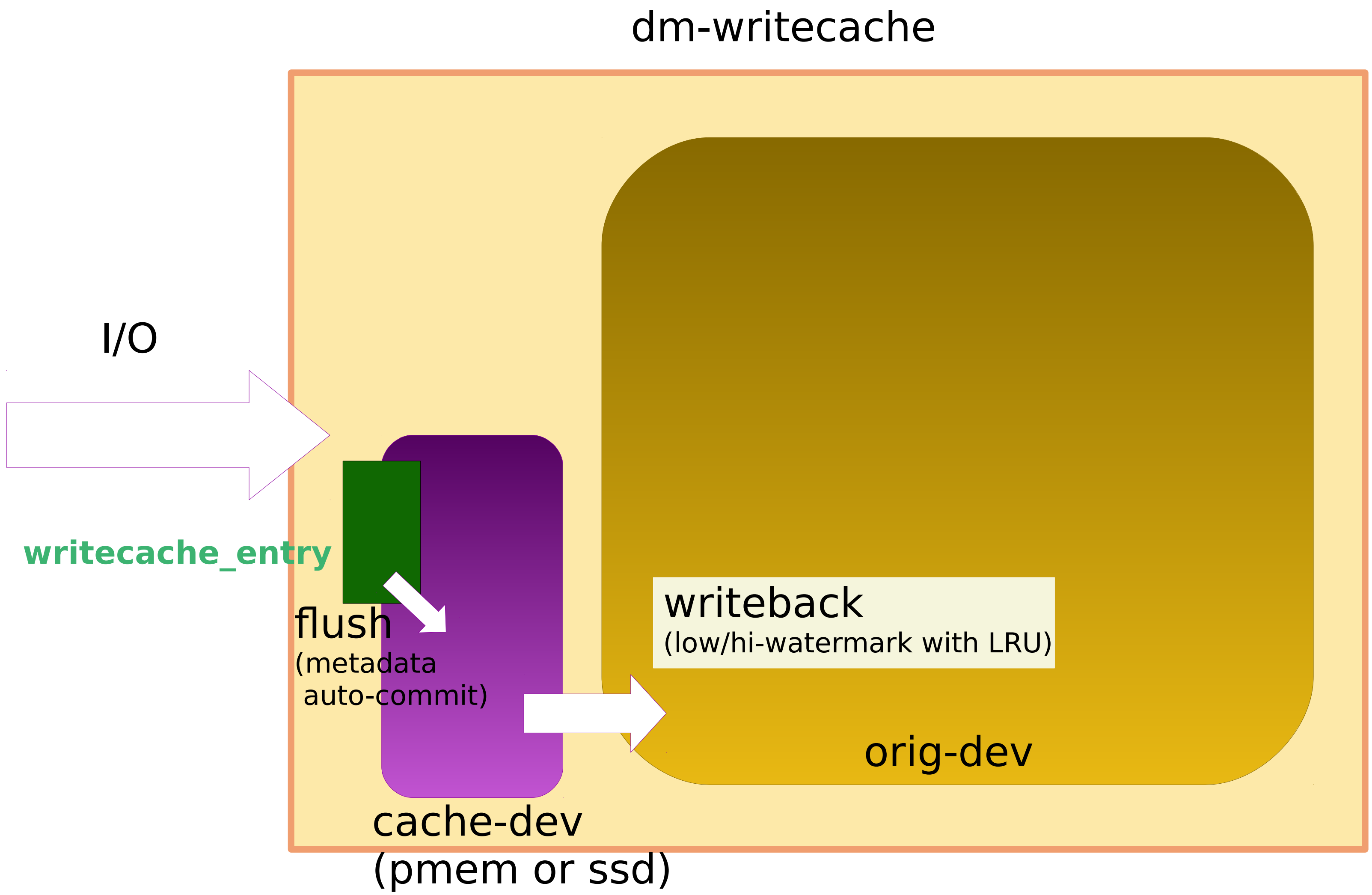
bpfilter


network stack (netfilter)
bpf
user process
request
request
bpfilter
bpfilter_umh
user mode helper

dm-writecache

- **Device mapper target**
  - Writeback caching to Pmem or SSD
    - Read will use page-cache (in RAM)
- **Operations**
  - Flush (commit) – metadata sync
  - Writeback : save to original dev

dm-writecache



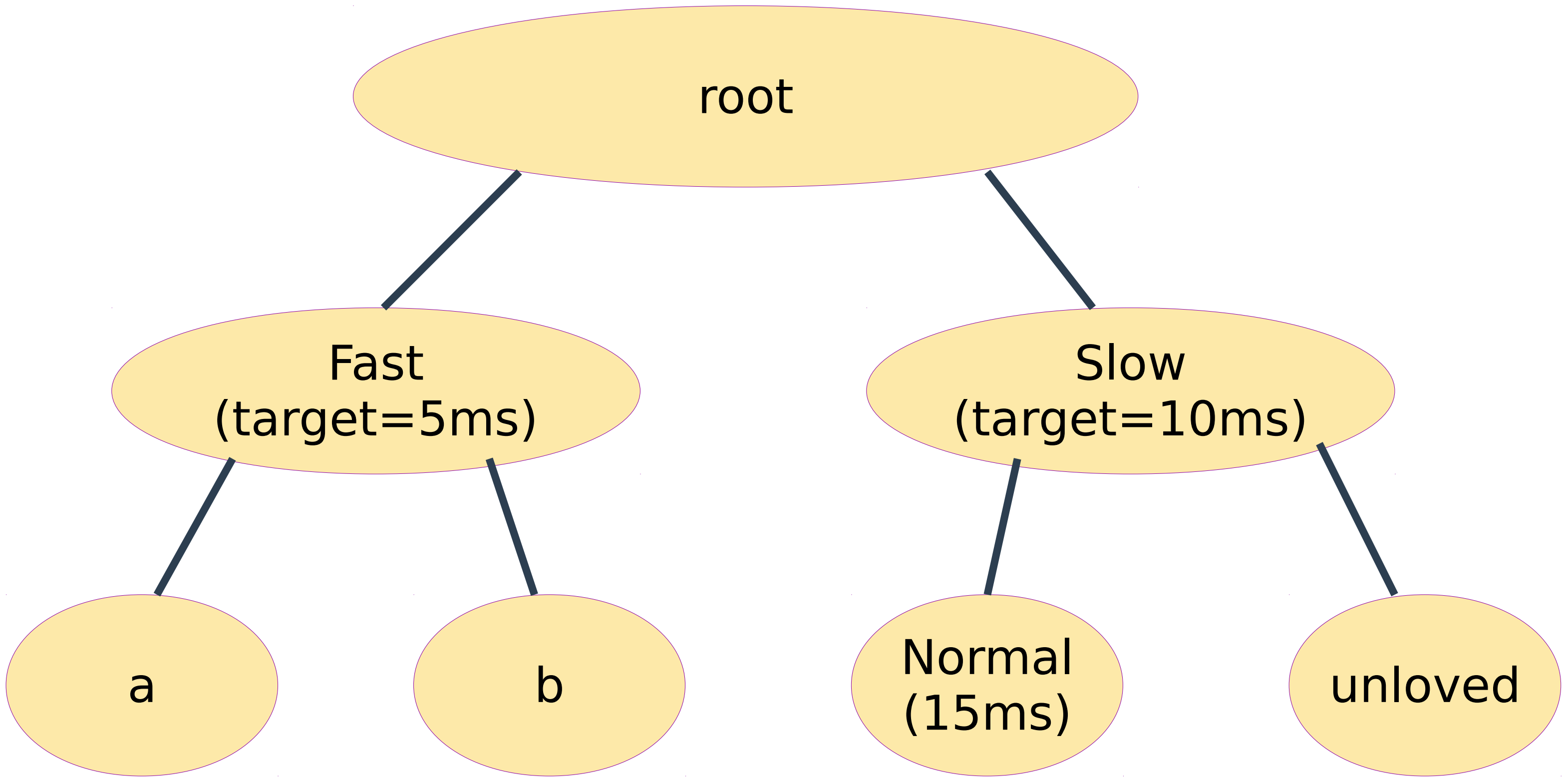
Async I/O polling

- **New polling API**
  - select()
  - poll()
  - epoll_wait()
  - io_submit() (opcode = IOCB_CMD_POLL)
- **Kernel Async-IO API**
  - No need to wait/block
  - use ring-buffer for communication

Block I/O latency controller

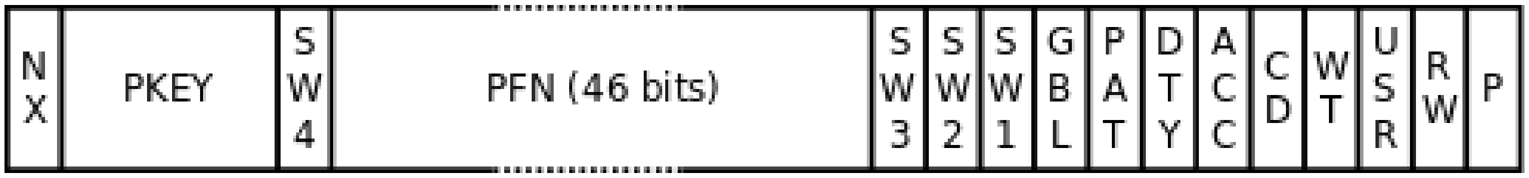
- **Bandwidth vs Latency**
  - Throttle I/O using cgroups
- **I/O latency controller**
  - echo <*major:minor*> target=100 > io.latency
  - <*major:minor*> = device ID
  - target = <*max latency in usec*>
- **Exceptions**
  - Metadata I/O (started by filesystem)
  - Swap I/O (started by reclaim)

Block I/O latency controller



L1TF Mitigations

- **L1 Terminal Fault (Foreshadow)**
  - Affects Intel CPUs only
  - Allow access to any physical memory
- **Bypass “Present” bit in PTE**
  - Only in L1 cache with speculation



https://lwn.net/Articles/762570/

L1TF Mitigations

- **Mitigations**
  - Invert all bits in PTE (for non-present page)
  - Flush cache before return to user
  - Disable SMT (hyper-threading)
- **Virtualization**
  - Guest can run any kernel
  - Disable EPT (Extended Page Table)

Any question?