Linux高性能服务调优实践

Gang Deng From Alibaba Cloud

Agenda

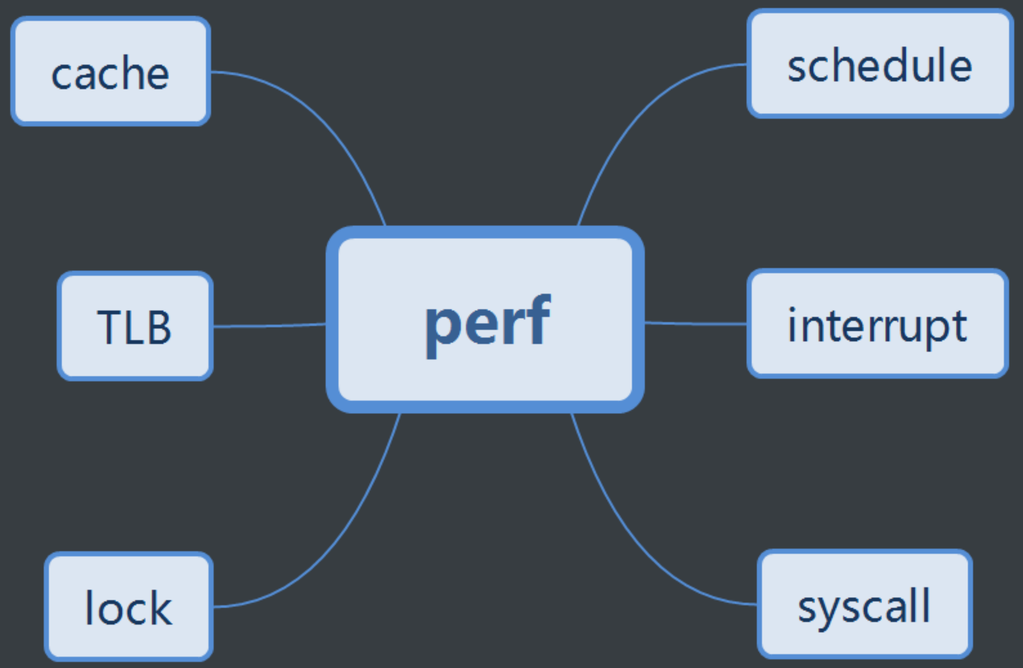
- Introduction
- Cache
- TLB
- Lockless
- Schedule
- Summarize

Agenda

- Introduction
- Cache
- TLB
- Lockless
- Schedule
- Summarize

Introduction

- ESSD
  - Enhanced SSD Cloud Disk
  - 1,000,000 iops & 100us
  - Challenge & solutions ?

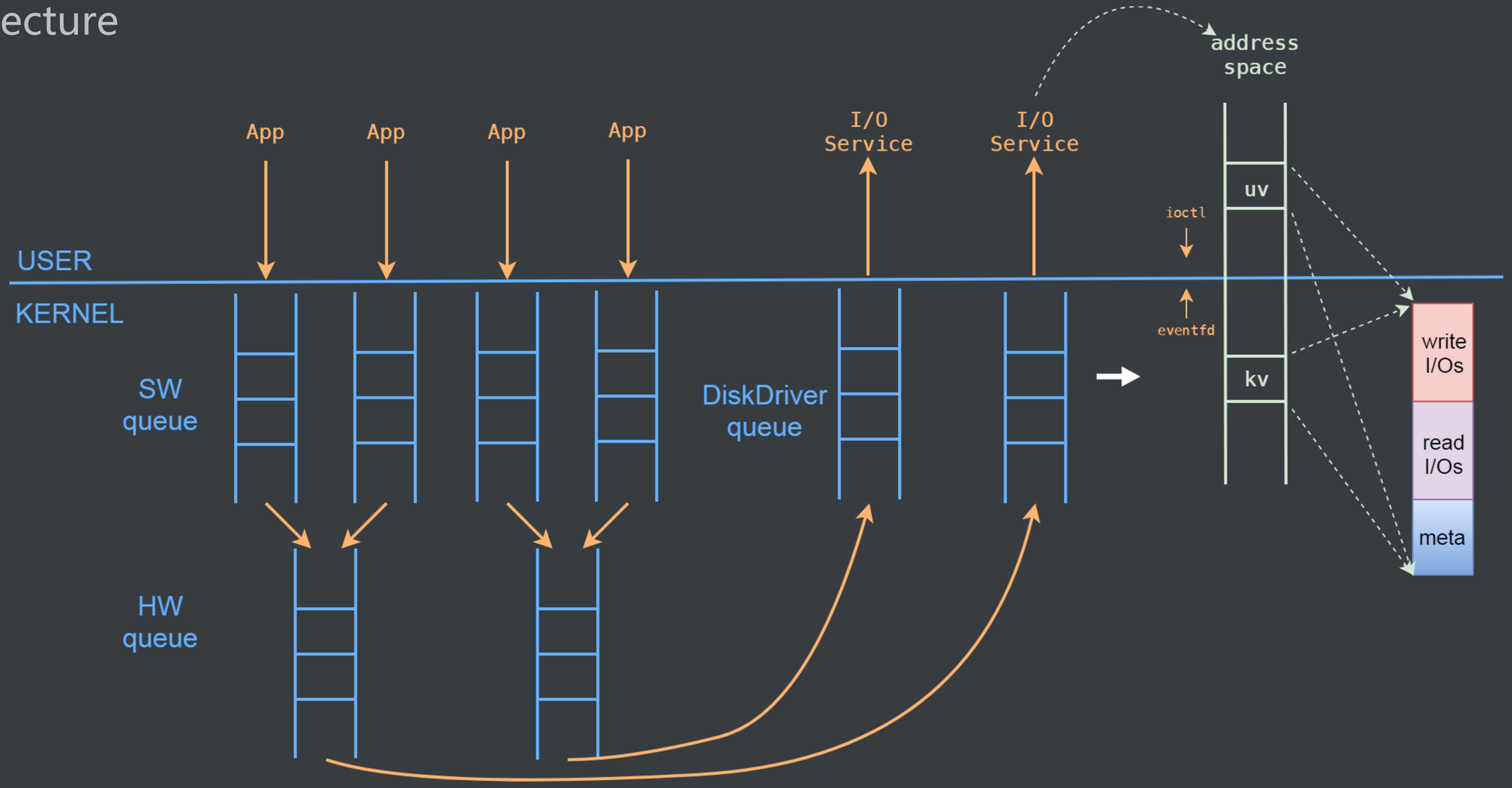


Agenda

- Introduction
- Cache
- TLB
- Lockless
- Schedule
- Summarize

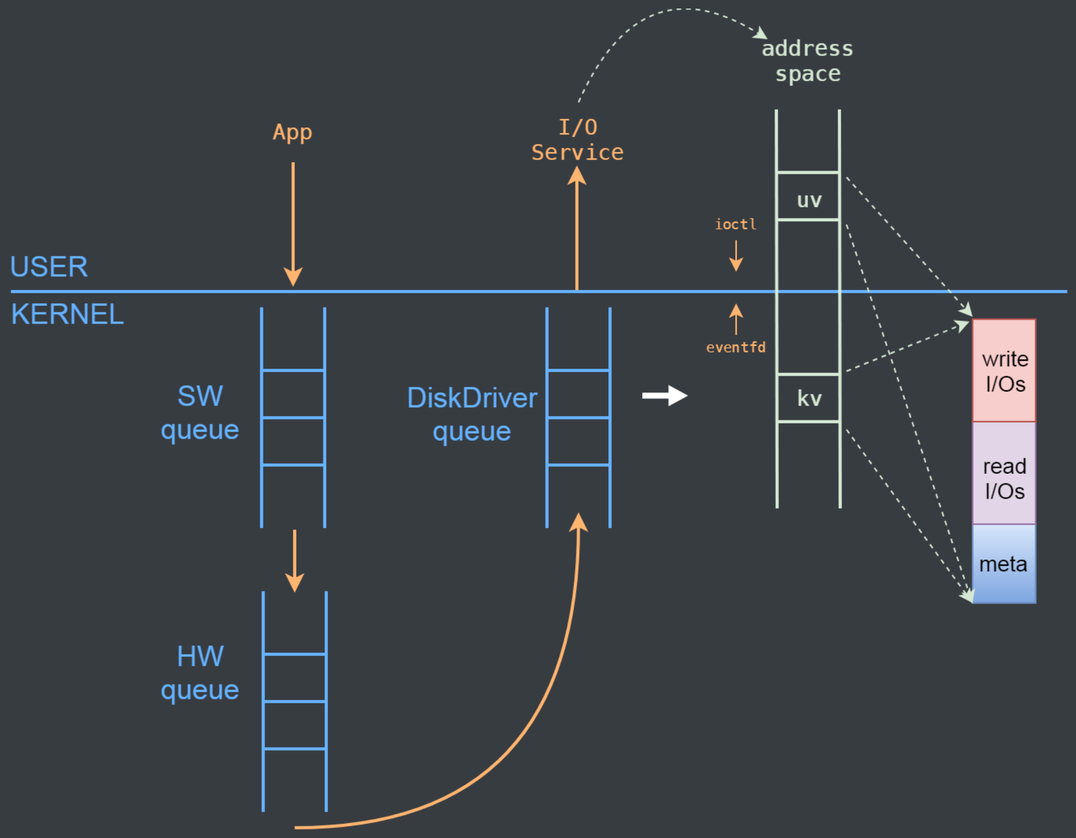
Cache

- Architecture

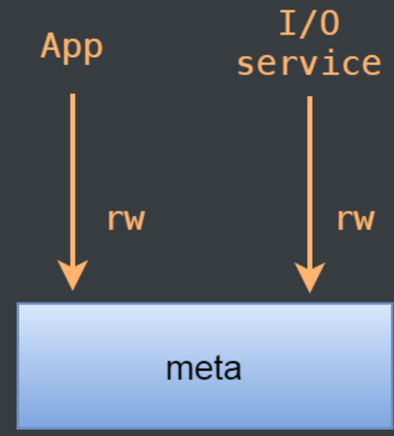


Cache

- Why need cache optimization?



Simplifed



Cache contention in IPC

Cache

- cache false sharing

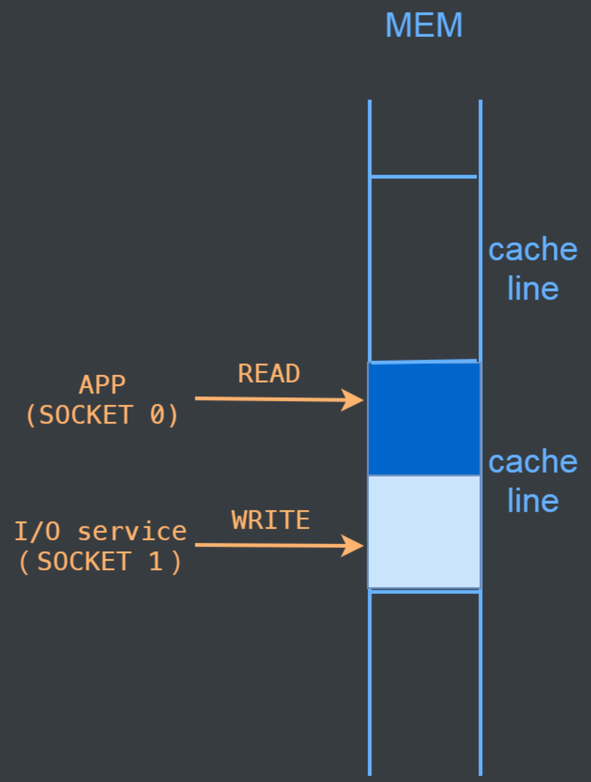


Fig 1. different sockets R/W the same cacheline

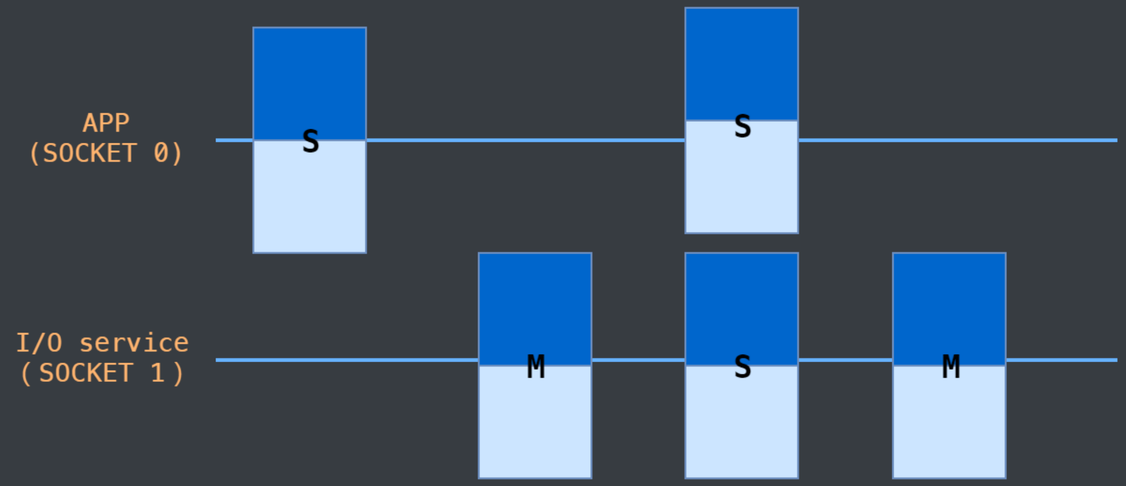


Fig2. cache contention

Cache

- How to eliminate it ?

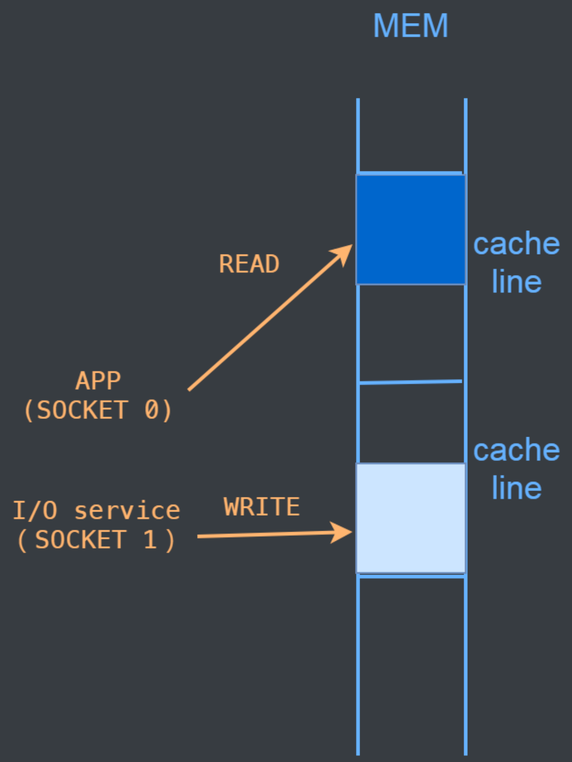


Fig 1. different sockets R/W different cacheline

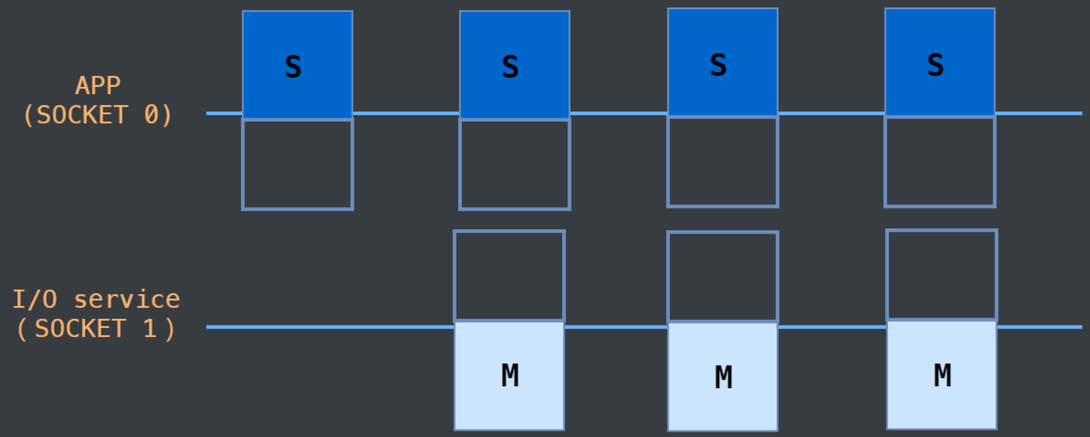


Fig2. no cache contention

Cache

- perf c2c

```
=================================================
            Shared Data Cache Line Table
=================================================
#
#                            Total      Tot  ----- LLC Load Hitm -----  ---- Store Reference ----  ---
# Index           Cacheline  records     Hitm    Total      Lcl      Rmt    Total    L1Hit   L1Miss  ...
# .....  ..................  .......  .......  .......  .......  .......  .......  .......  .......  ...
#
      0  0xffffc90041261080      466   25.79%      172        0      172       99       99        0
      1  0xffff887e7f355440       55    4.20%       28        0       28        2        2        0
      2  0xffff887e7f2d5440       46    3.75%       25        0       25        3        3        0
      3  0xffff887e7f255440       46    3.45%       23        0       23        1        1        0
      4  0xffff887e7f215440       37    3.15%       21        0       21        3        3        0
      5  0xffff887e7f335440       47    3.15%       21        0       21        4        4        0
      6  0xffff887e7f2b5440       41    2.70%       18        0       18        5        5        0
      7  0xffff887e7f375440       31    1.80%       12        0       12        3        3        0
      8  0xffff887e7f3b5440       31    1.35%        9        0        9        2        2        0
      9      0x7f8263ff2800       10    1.05%        7        0        7        0        0        0
     10  0xffff887bcd597080       67    0.75%        5        0        5        0        0        0
     11  0xffff887713d257c0        9    0.60%        4        0        4        3        3        0
     12  0xffff88771e84f400       11    0.60%        4        0        4        0        0        0
     13  0xffff88775872a5c0        8    0.60%        4        0        4        4        4        0
     14  0xffff887782a5cd00        6    0.60%        4        0        4        0        0        0
     15  0xffff887e7f2f5440        6    0.60%        4        0        4        0        0        0
     16      0x7f8263fdec00        4    0.60%        4        0        4        0        0        0
     17      0x7f8263feac00        7    0.60%        4        0        4        0        0        0
     18      0x7f8263ff1400        5    0.60%        4        0        4        0        0        0
     19  0xffff887713d27540        6    0.45%        3        0        3        0        0        0
```

hot spot

Fig1. hot cacheline statistic

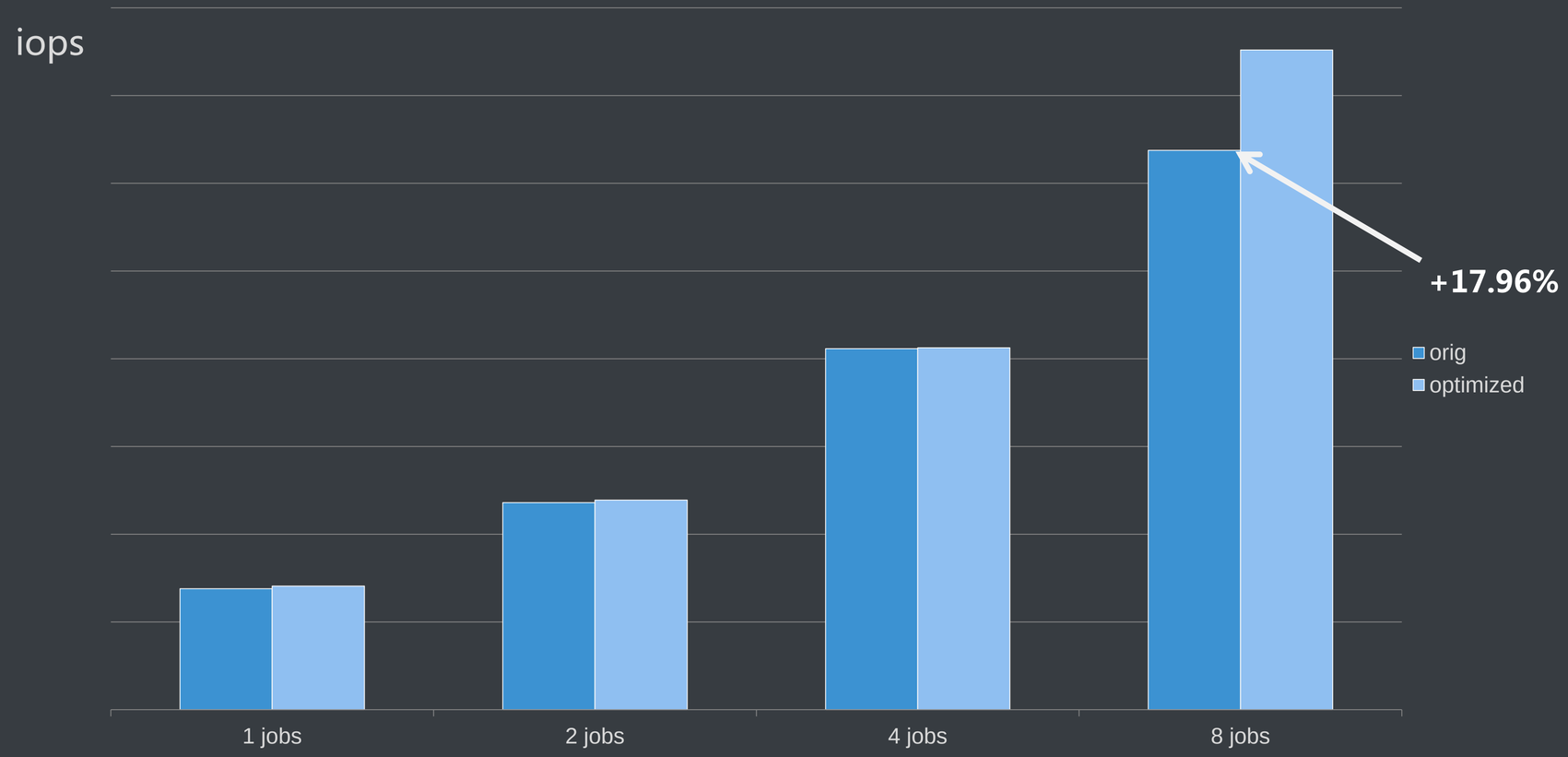
```
95.35%    0.00%    0.00%    0.00%                  0x1c    62900  0xffffffffa05bbeb7
   |
   |--0.34%--0xffffffffffffffff
   |          |
   |          |--0.31%--__GI___ioctl
   |          |          system_call
   |          |          sys_ioctl
   |          |          do_vfs_ioctl
   |          |          0x5f8
   |          |          0x429b
   |          |          0xeb7
   |          |
   |           --0.03%--__clone
   |                     start_thread
   |                     vnic_polling_run
   |                     |
   |                     |--0.03%--vnic_null_do_req
   |                     |          |
   |                     |          |--0.02%--vnic_req_done
   |                     |          |          __GI___ioctl
   |                     |          |          system_call
   |                     |          |          sys_ioctl
   |                     |          |          do_vfs_ioctl
   |                     |          |          0x5f8
   |                     |          |          0x429b
   |                     |          |          0xeb7
   |                     |          |
```

code addr

Fig2. locate code address

Cache

- optimized performance



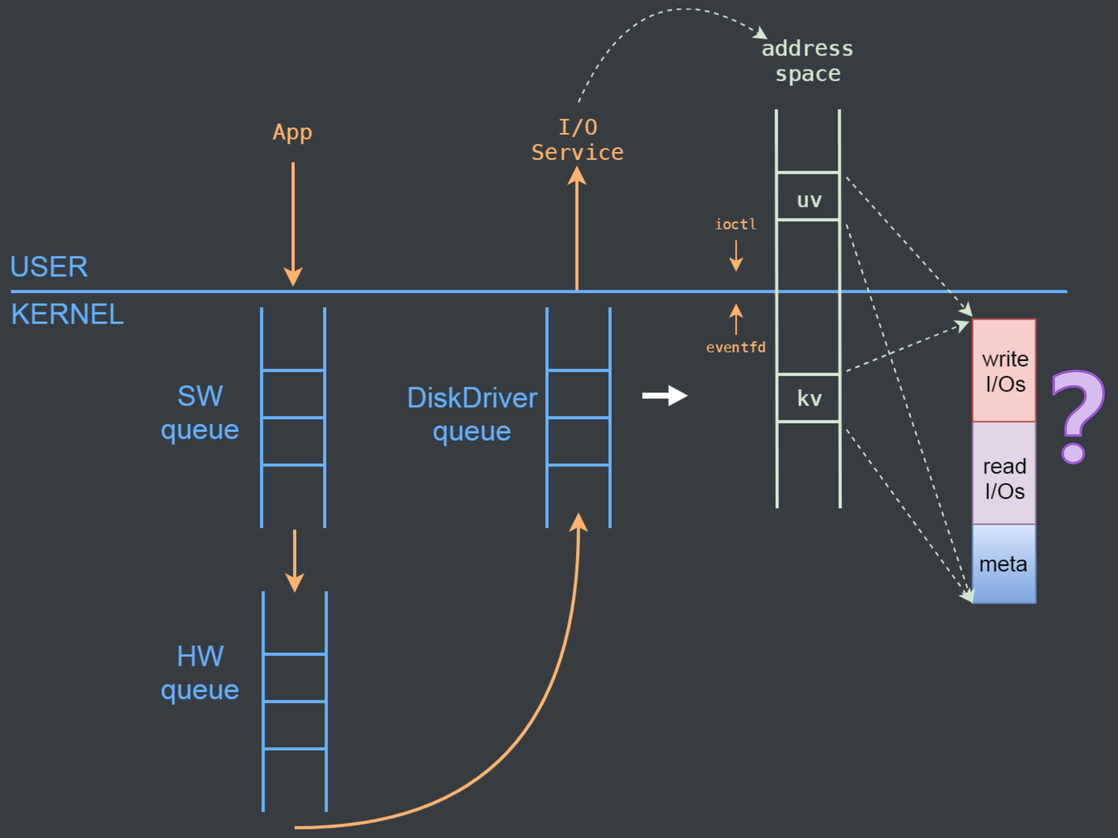
bs=4k，iodepth=8，meta perf

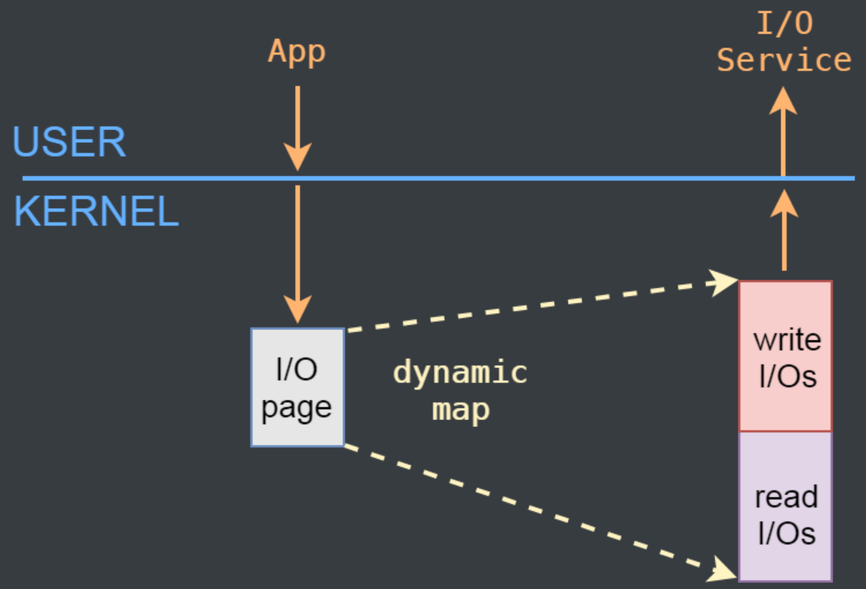
Agenda

- Introduction
- Cache
- TLB
- Lockless
- Schedule
- Summarize

TLB

- zero copy

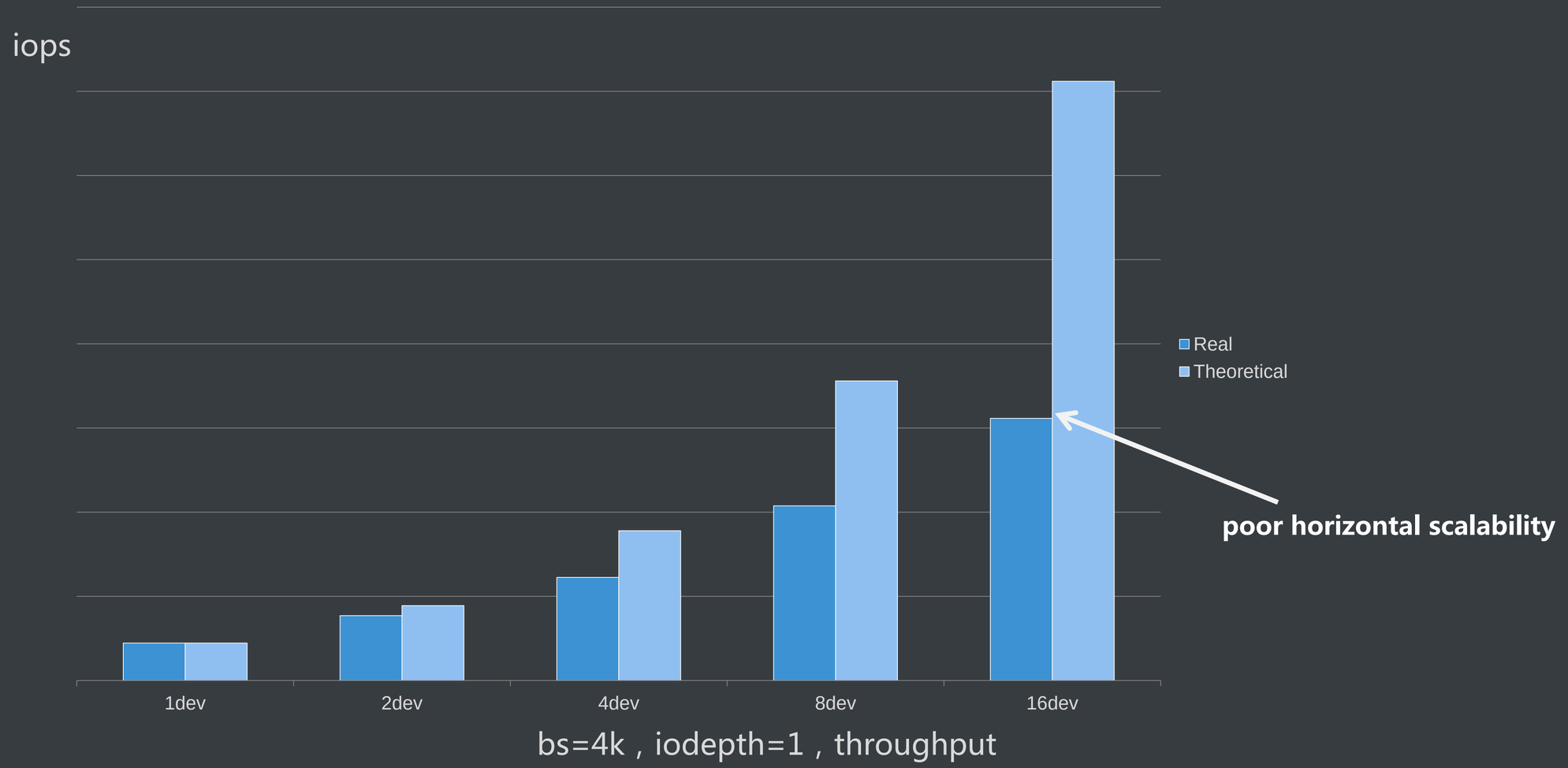




apply_to_page_range( )

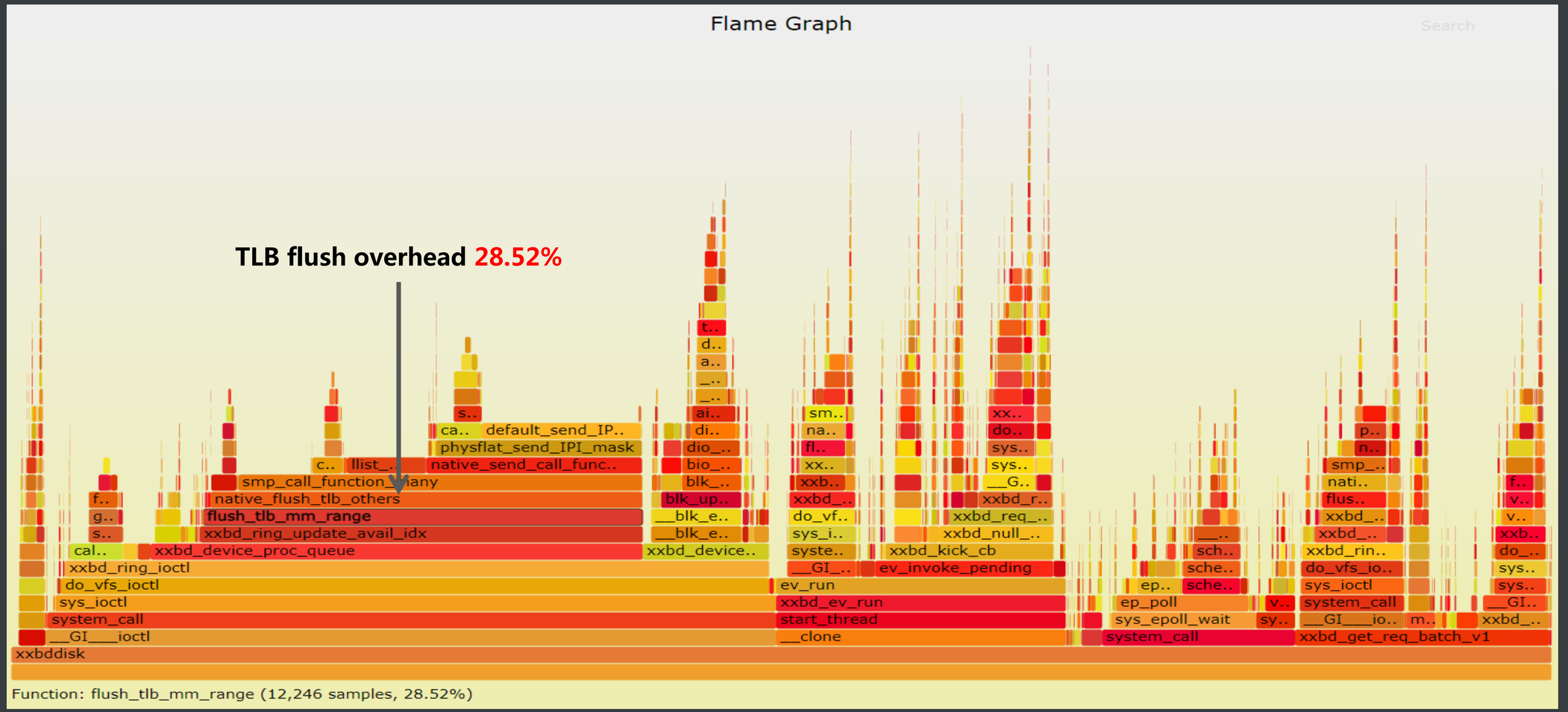
TLB

- performance of zero copy



TLB

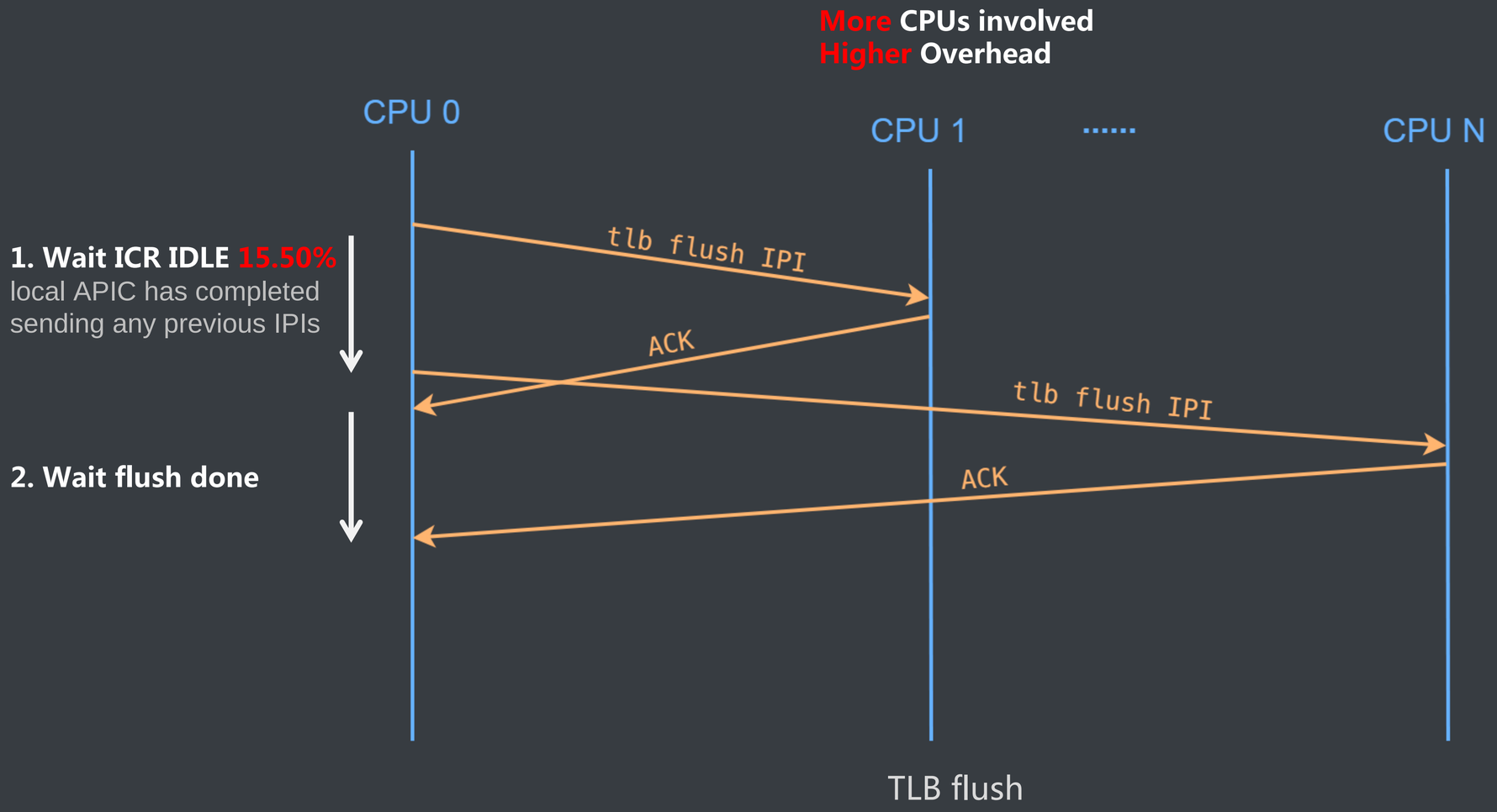
- performance analysis



16 devs, I/O service

TLB

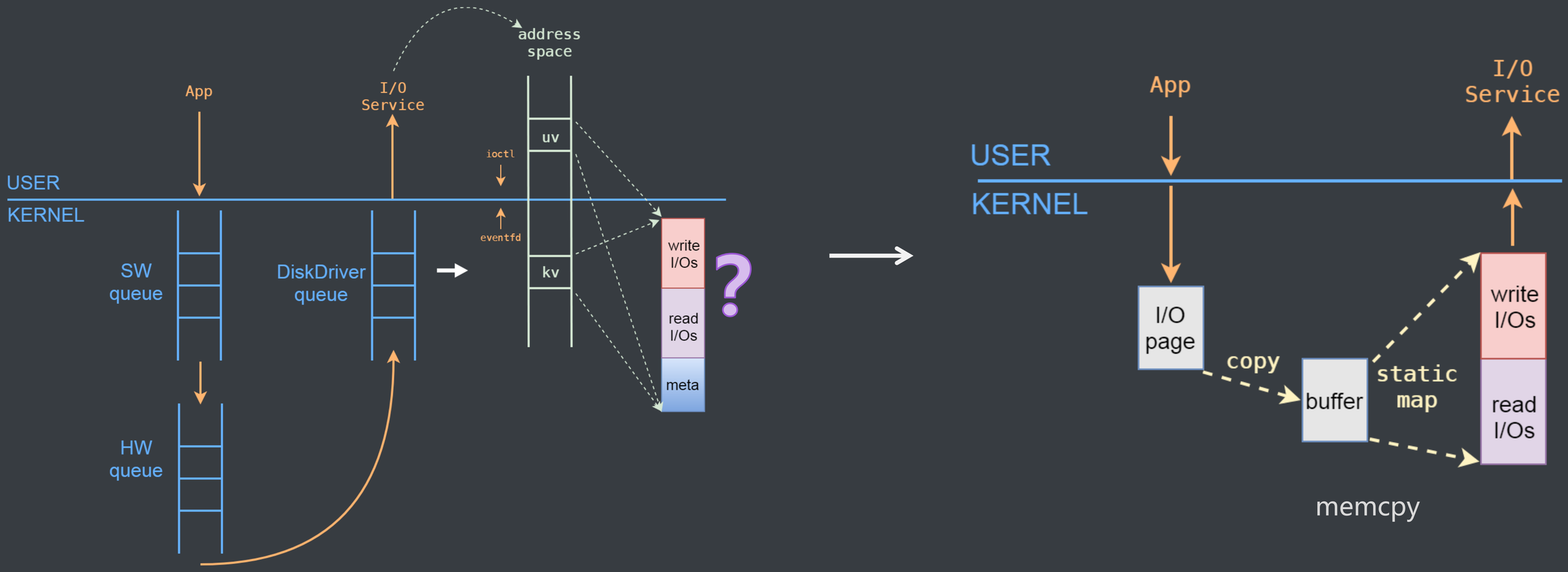
- How does TLB flush work?



TLB flush

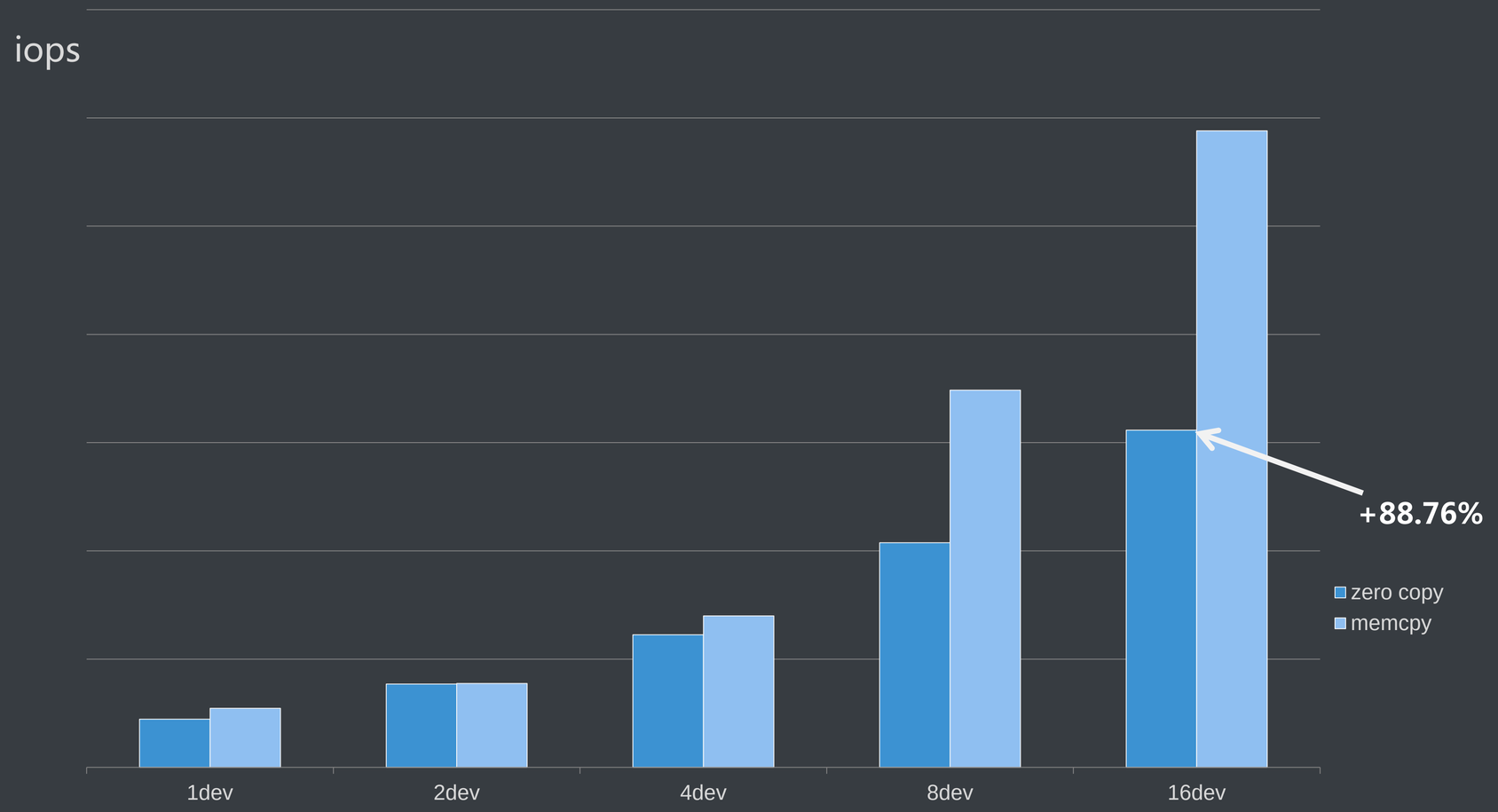
TLB

- TLB flush elimination



TLB

- zero copy V.S. memcpy



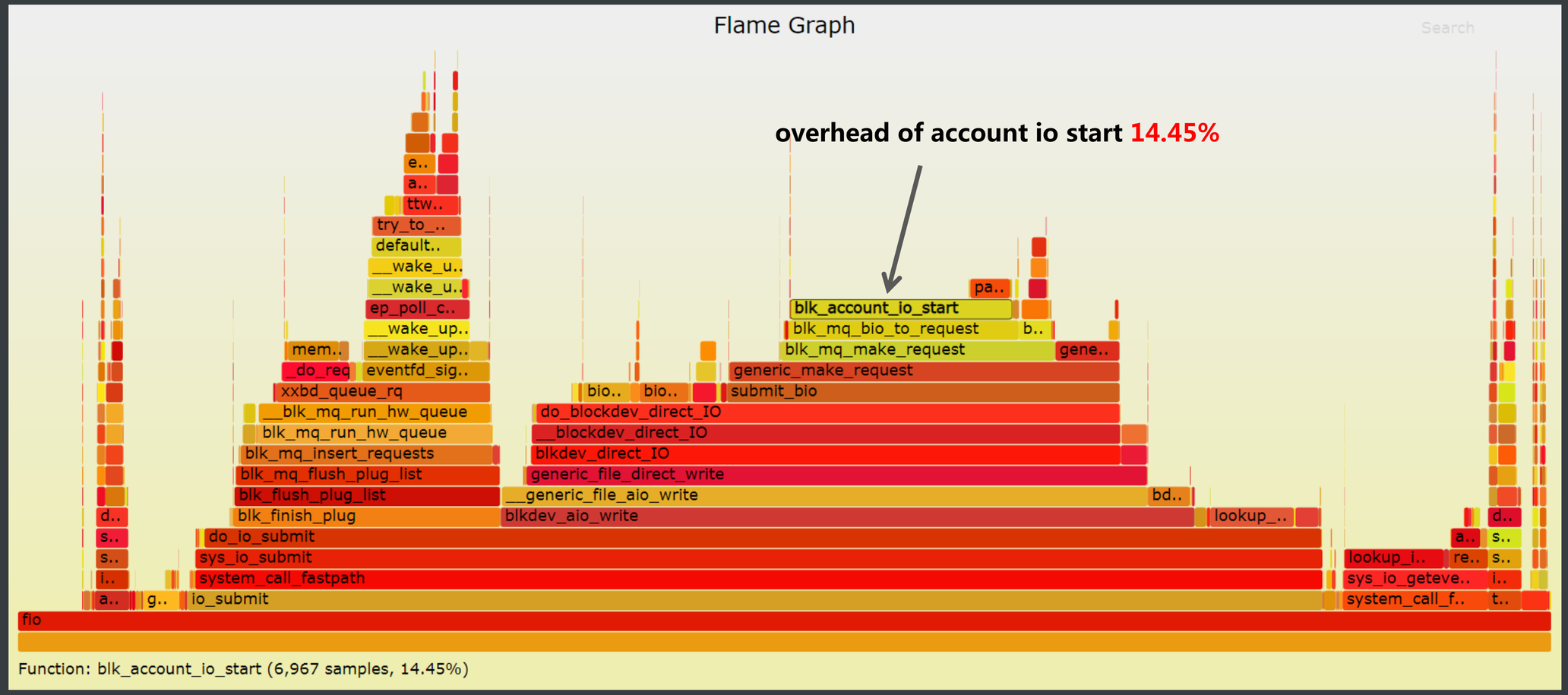
bs=4k，iodepth=1，throughput of multi devices

Agenda

- Introduction
- Cache
- TLB
- Lockless
- Schedule
- Summarize

Lockless

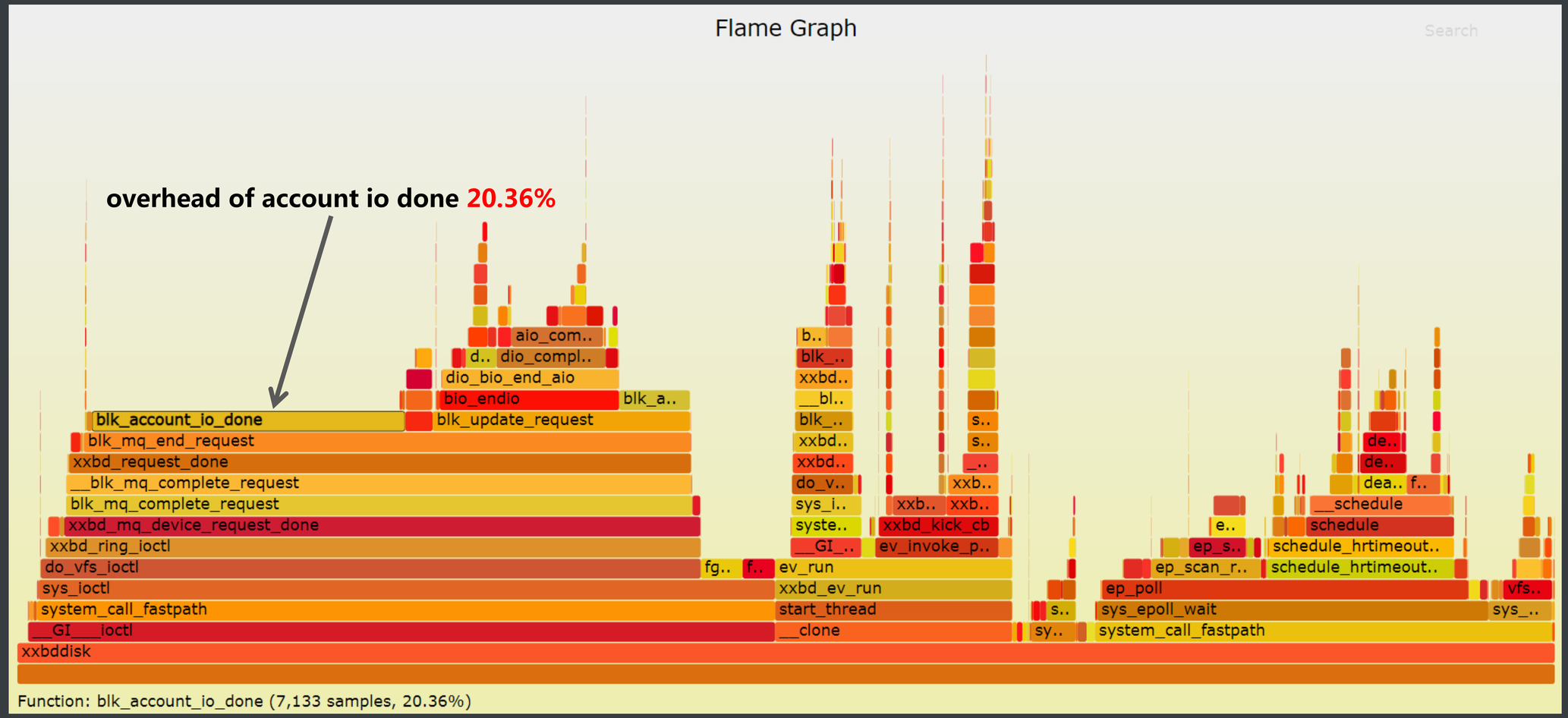
- Overhead of io accounting when high throughput



1,760,000 iops , fio

Lockless

- Overhead of io accounting when high throughput



1,760,000 iops , I/O service

Lockless

- Overhead of io accounting when high throughput
  - CPU overhead: 14.45%(20.36%)

```
                static inline void part_dec_in_flight(struct hd_struct *part, int rw)
                {
                        atomic_dec(&part->in_flight[rw]);
                  lea    0xb8(%r12),%rax
                atomic_dec():
                 *
                 * Atomically decrements @v by 1.
                 */
                static inline void atomic_dec(atomic_t *v)
                {
                        asm volatile(LOCK_PREFIX "decl %0"
                  lock   decl   0x8(%rbx,%rax,4)
                part_dec_in_flight():
                        if (part->partno)
 36.42            mov    0x2d4(%rbx),%eax
  7.48            test   %eax,%eax
                ↓ je     129
                part_to_disk():
                        int node_id;
                };
```

Overhead of single atomic instruction = 20.36% * (36.42+7.48)% = 8.94%

Fig1. cpu-cycles sampling of blk_account_io_done()

```
                static inline void part_dec_in_flight(struct hd_struct *part, int rw)
                {
                        atomic_dec(&part->in_flight[rw]);
                  lea    0xb8(%r12),%rax
                atomic_dec():
                 *
                 * Atomically decrements @v by 1.
                 */
                static inline void atomic_dec(atomic_t *v)
                {
                        asm volatile(LOCK_PREFIX "decl %0"
                  lock   decl   0x8(%rbx,%rax,4)
                part_dec_in_flight():
                        if (part->partno)
 50.31            mov    0x2d4(%rbx),%eax
 16.45            test   %eax,%eax
                ↓ je     129
                part_to_disk():
                        int node_id;
                };
```

Fig2. cache-misses sampling of blk_account_io_done()

Atomic inflight io accounting was deleted in commit f299b7c7a9de:
blk-mq: provide internal in-flight variant

Lockless

- Overhead of io accounting when high throughput
  - CPU overhead: 14.45%(20.36%)
  - CPU overhead of single atomic instruction: 8.94%
  - Disable io accounting, we achieved 2,500,000 iops (+42.05%)

Lockless I/O accounting

no spinlock, no atomic, avoid cache contention

Lockless

- Implementation
  - per-queue accounting separately
  - store data in share memory
  - must be cache line size aligned
  - iostat tool shows aggregate output

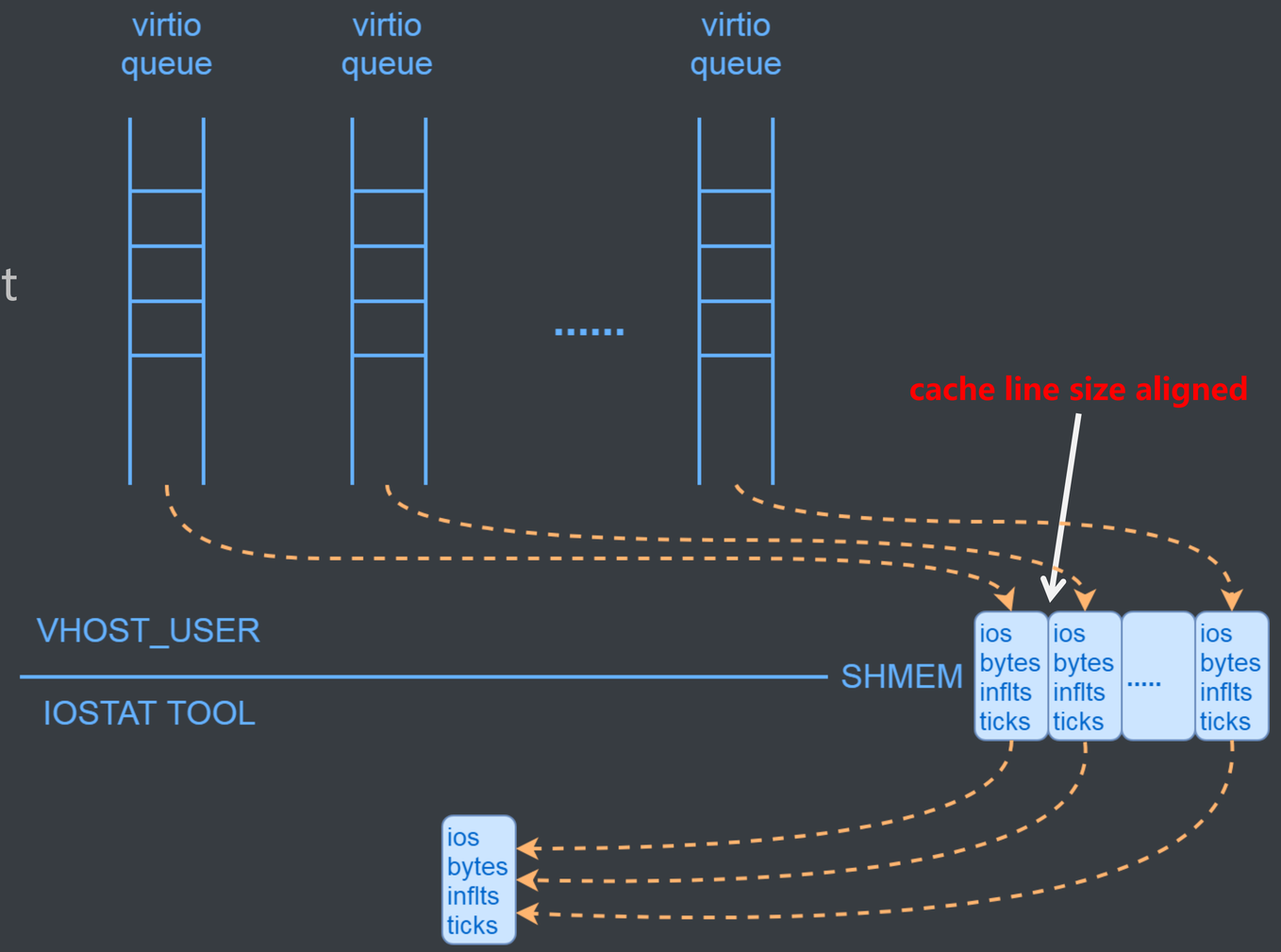


Fig1. lockless accounting

Agenda

- Introduction
- Cache
- TLB
- Lockless
- Schedule
- Summarize

Schedule

- latency analysis

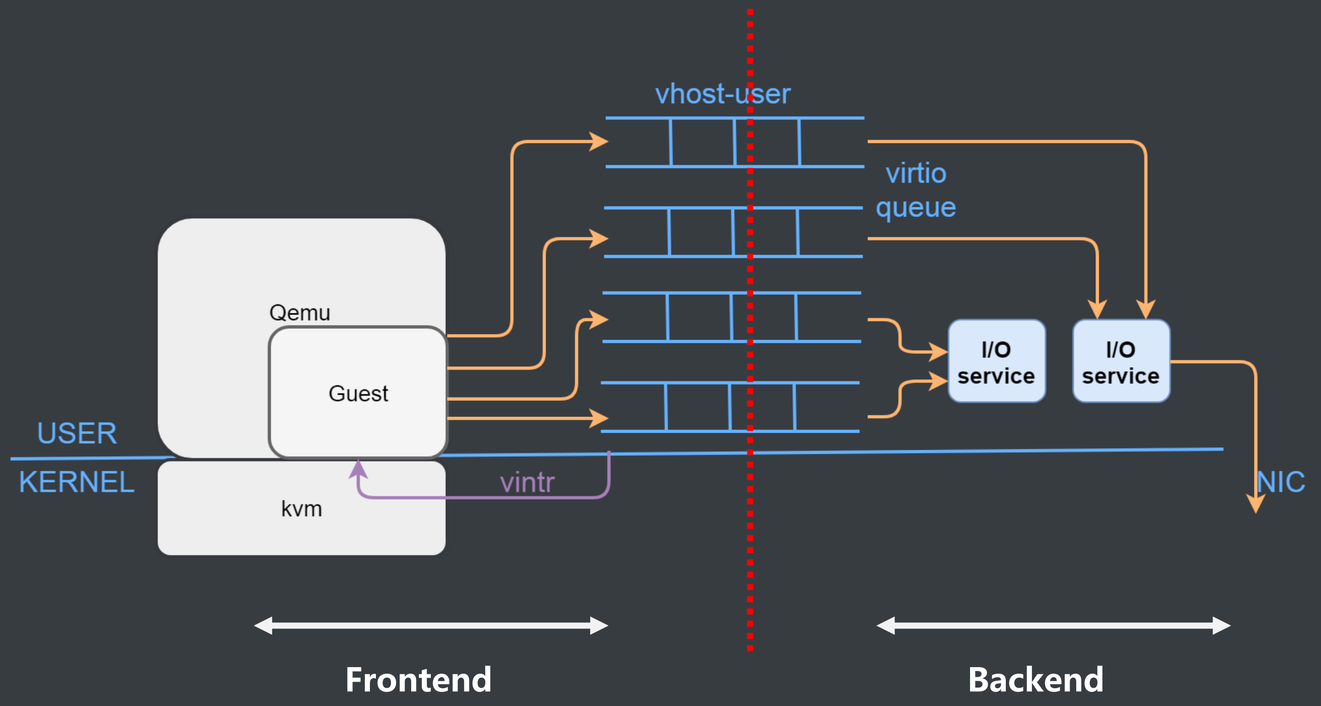


Fig1. Frontend & Backend

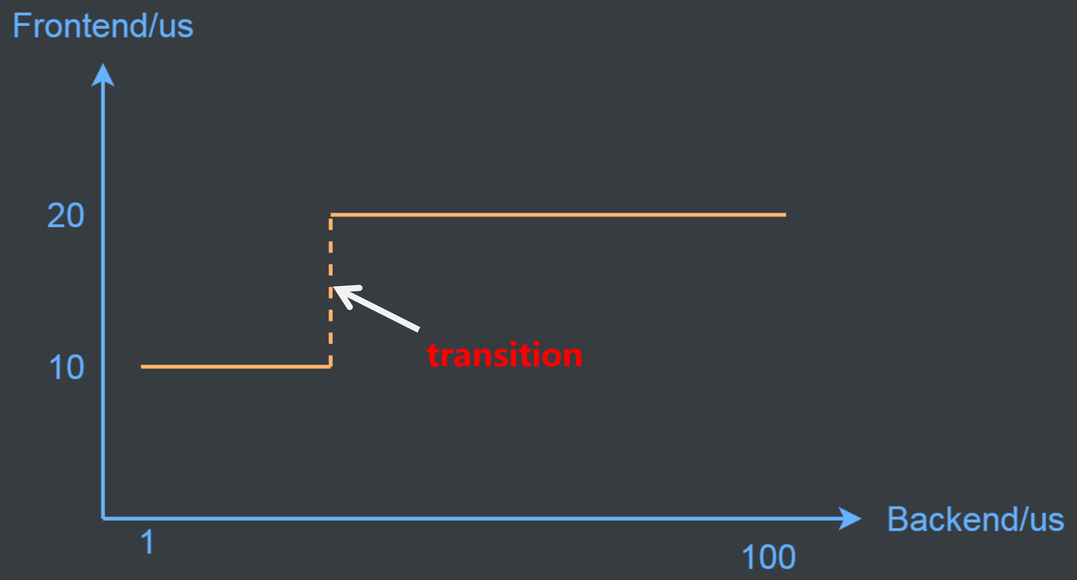


Fig2. transition of latency

Schedule

- event tracing
  - kvm_msi_set_irq
  - kvm_exit
  - kmv_entry

Schedule

- event tracing

```
qemu-system-x86-8548  [013] d... 15668.476655: kvm_entry: vcpu 0
qemu-system-x86-8548  [013] d... 15668.476660: kvm_exit: reason MSR_WRITE rip 0xffffffff81046208 info 0 0
qemu-system-x86-8548  [013] .... 15668.476660: kvm_msr: msr_write 6e0 = 0x58808bb017e
qemu-system-x86-8548  [013] d... 15668.476661: kvm_entry: vcpu 0
         <...>-145549 [001] d... 15668.476668: kvm_msi_set_irq: dst 0 vec 51 (Fixed|physical|edge)
qemu-system-x86-8548  [013] d... 15668.476669: kvm_exit: reason MSR_WRITE rip 0xffffffff81046208 info 0 0
qemu-system-x86-8548  [013] .... 15668.476670: kvm_msr: msr_write 6e0 = 0x588bb41917e
qemu-system-x86-8548  [013] d... 15668.476671: kvm_entry: vcpu 0
```

inject vintr while vm running

Fig1. backend = 1us

```
qemu-system-x86-8548  [015] d... 16033.974610: kvm_entry: vcpu 0
qemu-system-x86-8548  [015] d... 16033.974620: kvm_exit: reason MSR_WRITE rip 0xffffffff81046208 info 0 0
qemu-system-x86-8548  [015] .... 16033.974622: kvm_msr: msr_write 6e0 = 0x665ae332d88
qemu-system-x86-8548  [015] d... 16033.974622: kvm_entry: vcpu 0
qemu-system-x86-8548  [015] d... 16033.974623: kvm_exit: reason HLT rip 0xffffffff81046345 info 0 0
         <...>-152787 [001] d... 16033.974628: kvm_msi_set_irq: dst 0 vec 51 (Fixed|physical|edge)
qemu-system-x86-8548  [015] d... 16033.974636: kvm_entry: vcpu 0
qemu-system-x86-8548  [015] d... 16033.974642: kvm_exit: reason MSR_WRITE rip 0xffffffff81046208 info 0 0
qemu-system-x86-8548  [015] d... 16033.974643: kvm_msr: msr_write 6e0 = 0x665348c5f68
qemu-system-x86-8548  [015] d... 16033.974643: kvm_entry: vcpu 0
```

inject vintr while vm exited

8us

Fig2. backend = 10us

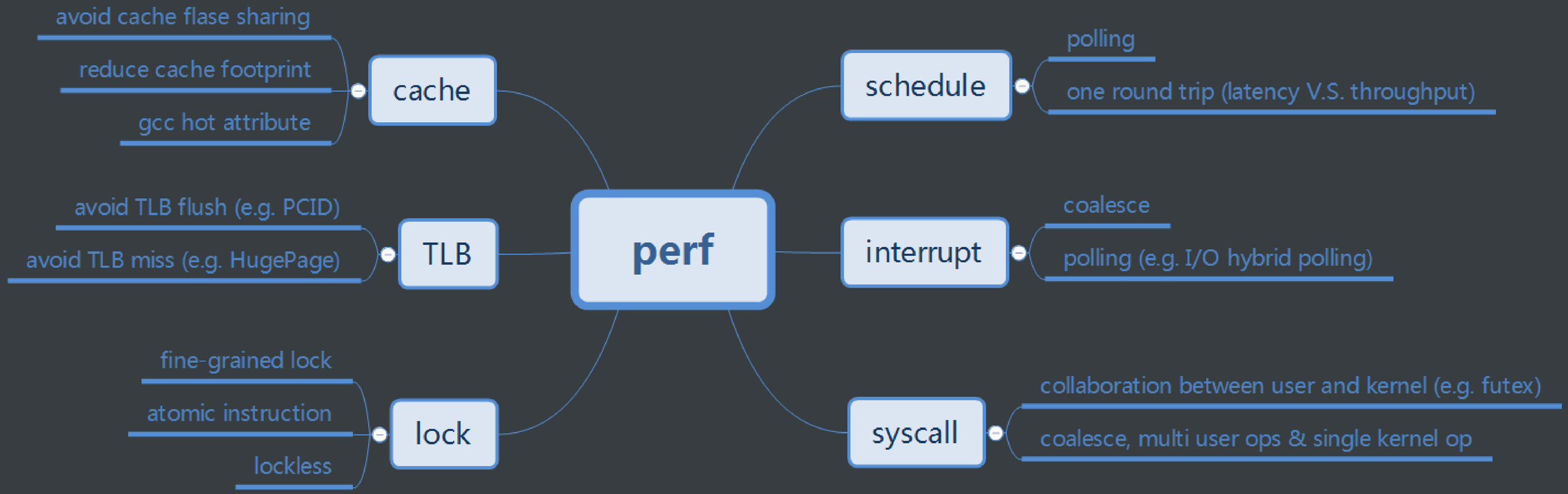
Schedule

- solutions
  - host kvm halt_poll
  - guest kernel cmdline: idle=poll
  - custom guest cpu idle driver

Agenda

- Introduction
- Cache
- TLB
- Lockless
- Schedule
- Summarize

Summarize


perf
cache
avoid cache flase sharing
reduce cache footprint
gcc hot attribute
TLB
avoid TLB flush (e.g. PCID)
avoid TLB miss (e.g. HugePage)
lock
fine-grained lock
atomic instruction
lockless
schedule
polling
one round trip (latency V.S. throughput)
interrupt
coalesce
polling (e.g. I/O hybrid polling)
syscall
collaboration between user and kernel (e.g. futex)
coalesce, multi user ops & single kernel op

Q & A

MORE THAN JUST CLOUD | Alibaba Cloud