Scheduling at Scale: Using BPF Schedulers with sched_ext

Daniel Hodges

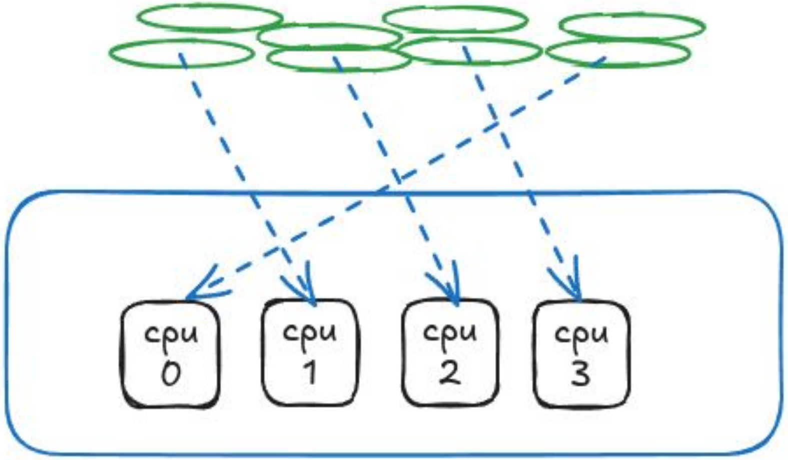
Meta

What is scheduling?

What - kernel (cfs/eevdf)

Where - What CPU

How long - time slice



Why?

```
            tot=    650 local=80.62 wake/exp/reenq=16.77/ 2.62/ 0.00
            xlayer_wake=17.69 xlayer_rewake=15.85
            keep/max/busy= 0.15/ 0.00/ 2.92 kick= 0.00 yield/ign= 0.15/    0
            open_idle= 0.00 mig= 9.38 xnuma_mig= 0.00 xllc_mig= 0.00 affn_viol= 0.31
            preempt/first/xllc/xnuma/idle/fail= 0.00/ 0.00/ 0.00/ 0.00/ 0.00/ 0.00 min_exec=97.69/ 392.91ms, slice=20ms
            cpus=  2 [  2,  2] 00000003
^CEXIT: unregistered from user space
thread 'l14411 /home/daniel #
```

```
  HDRTEST usr/include/linux/dvb/dmx.h
  HDRTEST usr/include/linux/dvb/net.h
  HDRTEST usr/include/linux/dvb/osd.h
  CC      crypto/compress.o
  HDRTEST usr/include/linux/dvb/video.h
  CC      fs/super.o
  CC      block/bio.o
  HDRTEST usr/include/linux/dvb/version.h
  HDRTEST usr/include/linux/dvb/frontend.h
  CC      security/keys/keyring.o
  HDRTEST usr/include/linux/genwqe/genwqe_card.h
  CC      init/do_mounts_initrd.o
  HDRTEST usr/include/linux/hdlc/ioctl.h
  HDRTEST usr/include/linux/hsi/cs-protocol.h
  HDRTEST usr/include/linux/hsi/hsi_char.h
  HDRTEST usr/include/linux/iio/events.h
  HDRTEST usr/include/linux/iio/buffer.h
  CC      arch/x86/lib/cmdline.o
  HDRTEST usr/include/linux/iio/types.h
  HDRTEST usr/include/linux/isdn/capicmd.h
^Z
[1]+  Stopped                 make -j `nproc`
l14411 /home/daniel/git/hodgesds-linux #
```

```
    1[|||||                    4.3% 1397MHz N/A]    9[|                       0.6% 1400MHz N/A]
    2[||||                     3.1% 1400MHz N/A]   10[|||||||||               8.5% 1397MHz N/A]
    3[||||                     4.3% 1400MHz N/A]   11[|||||||||               8.6% 1397MHz N/A]
    4[|                        0.6% 1400MHz N/A]   12[                        0.0% 1397MHz N/A]
    5[|||||                    3.7% 1400MHz N/A]   13[|                       0.6% 1400MHz N/A]
    6[||||                     3.1% 1400MHz N/A]   14[                        0.0% 1397MHz N/A]
    7[|||||||                  6.9% 1400MHz N/A]   15[||||                    2.5% 1400MHz N/A]
    8[||                       1.2% 1397MHz N/A]   16[                        0.0% 1400MHz N/A]
  Uptime: 01:22:33                                 Battery: 90.3% (Running on A/C)
  Mem:46.4G used:3.40G shared:82.7M compressed:0K buffers:2.09M cache:6.01G available:42.5G
                                                   Tasks: 161, 588 thr, 301 kthr; 2 running
  zrm:0K used:0K uncompressed:0K                   Load average: 5.68 5.02 4.26
  Swp:0K used:0K cache:0K frontswap:0K             PSI some CPU:     4.09% 12.44%  9.20%
  Disk IO: 0.0% read: 0KiB/s write: 0KiB/s         PSI full IO:      0.00%  0.03%  0.00%
  Network: rx: 0KiB/s tx: 0KiB/s (1/0 pkts/s)      PSI full memory:  0.00%  0.00%  0.00%
COMM            CPU SCHED  NI  DISK READ  DISK WRITE START   PID USER      PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
supertuxkart      5 OTHER   0   0.00 B/s   0.00 B/s 16:23 18894 daniel     20   0 2951M  285M 93160 S  25.3  0.6  4:37.82 supertuxkart
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

```
[0]                                                          1:sudo |2:bash*|3:nvim- 6.11.0-rc1-x86_64+
1 2 3                    / 353.1 GiB 62.1% | ... | ⚡ 69.25% | ... | ♪ 0% 2024-10-29 16:47:35
```

Why

Modern Hardware

Stacked workloads

Scheduling is hard

Overview

How to build schedulers with BPF

sched_ext schedulers

Scheduling at Scale

What is sched_ext?

Scheduler (kernel 6.12+)

BPF as a scheduler

Safety - automatic exit on task stalls

Rapid iteration time, no reboots!

Extending the kernel with BPF

High level overview- READ CODE

BPF- Some knowledge required

Goal- Understand building blocks

BPF Building Blocks

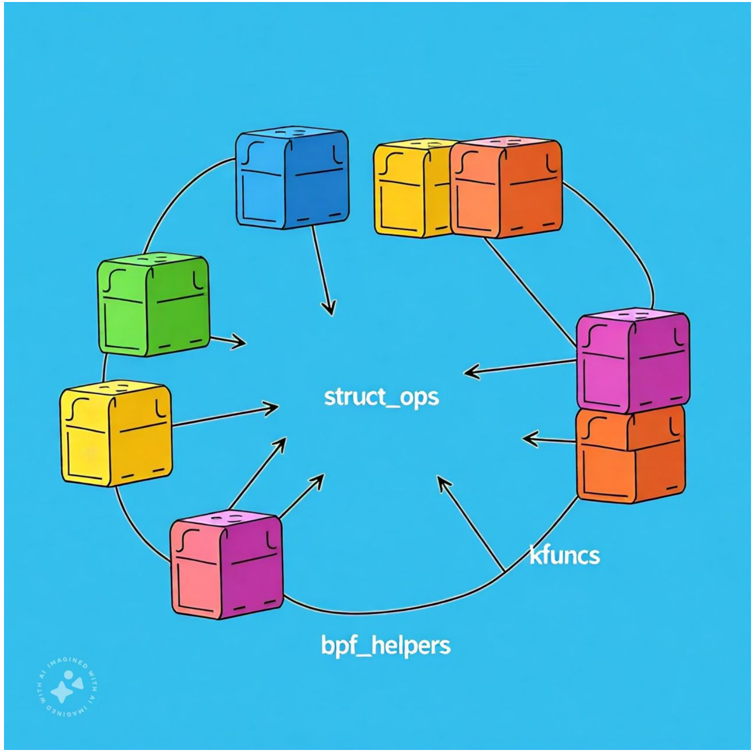
bpf_helpers

kfuncs

maps

bpf_cpumasks

struct_ops

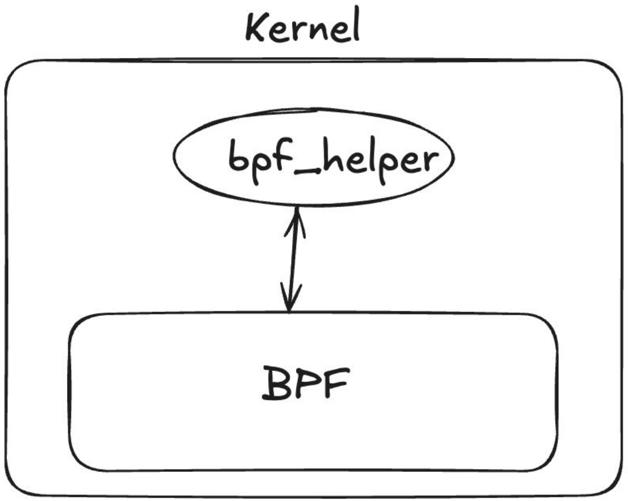


bpf_helpers

Found in all types of bpf programs

Tracing, networking, security, etc

Example: stack trace



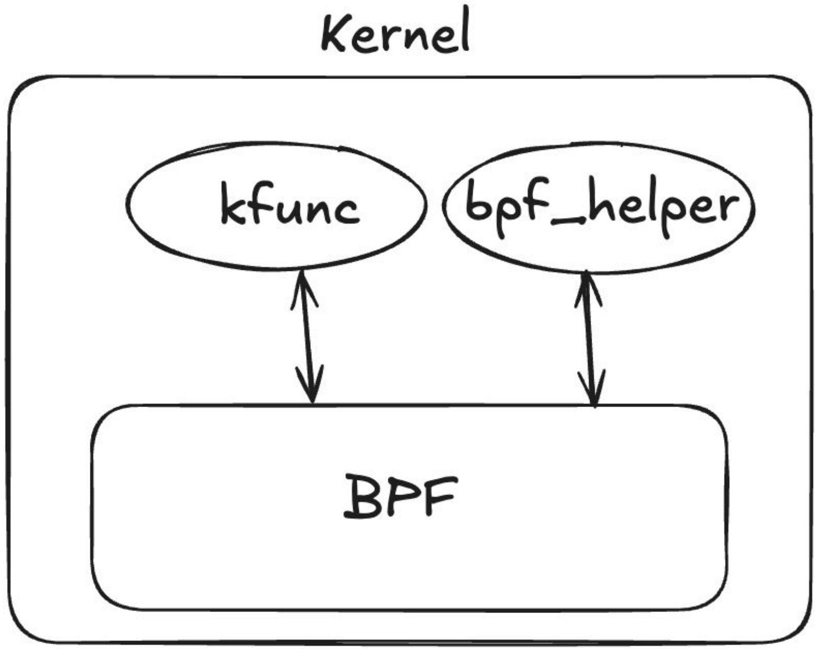

```
long bpf_get_stack(void *ctx, void *buf, u32 size, u64 flags)
```

kfuncs

Not a stable interface

Restrictions per program type

See kernel docs for details



BPF Maps

BPF_MAP_TYPE_TASK_STORAGE

BPF_MAP_TYPE_PERCPU_ARRAY

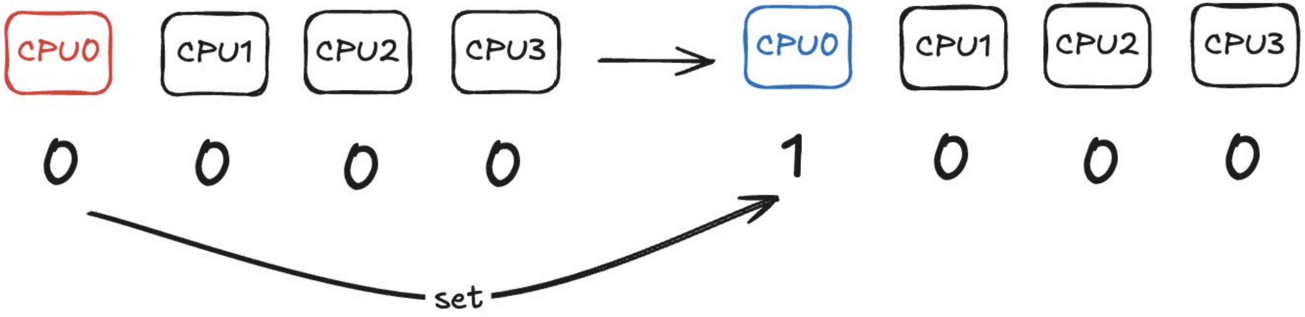
BPF_MAP_TYPE_HASH

Many more!

bpf_cpumasks

BPF version of `struct_cpumask`

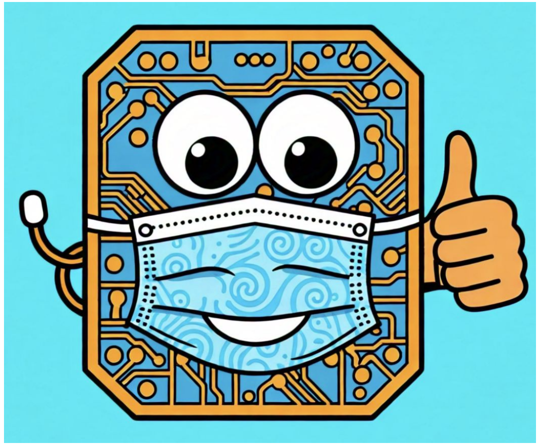
Idle CPU tracking, Topology, Soft Affinity



bpf_cpumasks

```
// Example bpf_cpumask
my_cpumask = bpf_cpumask_create();
if (!cpumask)
        return -ENOMEM;

bpf_cpumask_set_cpu(0, cpumask);
bpf_cpumask_and(
    my_cpumask, other_cpumask, my_cpumask);
```



Implementation: kfuncs

Bitwise operations: and, or, xor

struct_ops

Interface

Implementation

Loading

sched_ext_ops

```
struct bpf_example_ops {
        int (*foo)(void);
        int (*bar)(int a, int b);
};
```

struct_ops

Interface

Implementation

Loading

sched_ext_ops

```
SEC("struct_ops/foo")
int BPF_PROG(foo_impl) {
    return 42;
}

SEC("struct_ops/bar")
int BPF_PROG(bar_impl, int a, int b) {
    return a + b;
}

SEC(".struct_ops.link")
struct bpf_example_ops example_1 = {
    .foo = (void *)foo_impl,
    .bar = (void *)bar_impl,
};
```

struct_ops

- Interface
- Implementation
- **Loading**
- sched_ext_ops

```
struct struct_ops_example *skel;
int err;

skel = struct_ops_example__open();
if (!ASSERT_OK_PTR(skel, "struct_ops_example_open"))
    return;

// do any setup
err = struct_ops_example__load(skel);
ASSERT_OK(err, "struct_ops_example_load");
```

struct_ops

Interface

Implementation

Loading

sched_ext_ops

```
// Macros included!
SCX_OPS_DEFINE(simple_ops,
    .select_cpu   = (void *)simple_select_cpu,
    .enqueue      = (void *)simple_enqueue,
    .dispatch     = (void *)simple_dispatch,
    .running      = (void *)simple_running,
    .stopping     = (void *)simple_stopping,
    .enable       = (void *)simple_enable,
    .init         = (void *)simple_init,
    .exit         = (void *)simple_exit,
    .name         = "simple");
```

DSQs: BPF Dispatch Queues

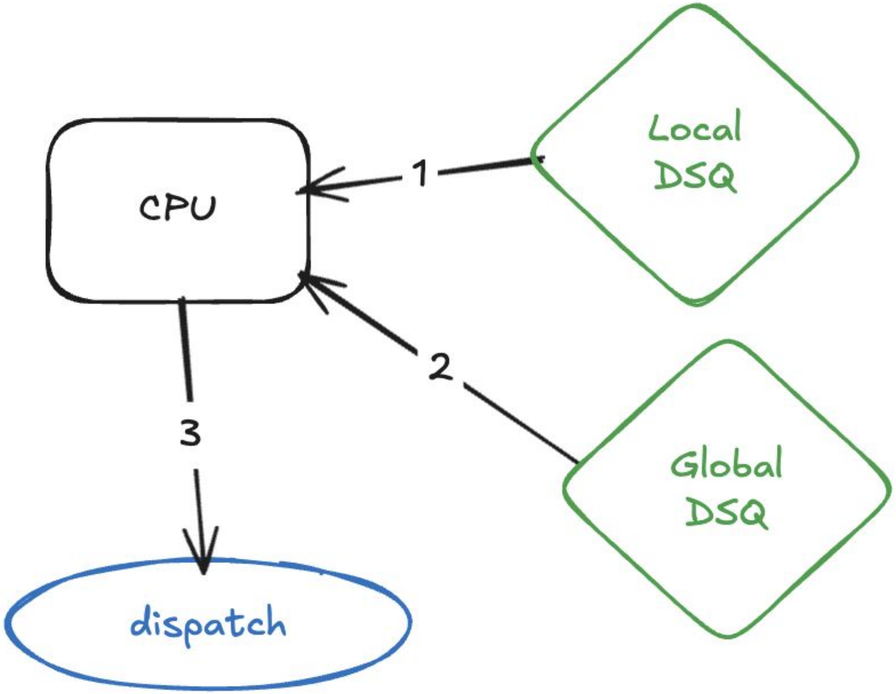
How to run a task?

Custom DSQs: `scx_bpf_create_dsq`

DSQs: BPF Dispatch Queues

A CPU always executes a task from its local DSQ fist

Scheduler can dispatch/ into local/global



How scheduling works: Initializing a Scheduler

scx_simple

init/exit for scheduler setup

Kernel docs/source for complete set

```
SCX_OPS_DEFINE(simple_ops,
           .select_cpu     = (void *)simple_select_cpu,
           .enqueue        = (void *)simple_enqueue,
           .dispatch       = (void *)simple_dispatch,
           .running        = (void *)simple_running,
           .stopping       = (void *)simple_stopping,
           .enable         = (void *)simple_enable,
           .init           = (void *)simple_init,
           .exit           = (void *)simple_exit,
           .name           = "simple");
```

https://github.com/sched-ext/scx/blob/main/scheds/c/scx_simple.c

How scheduling works: Task Execution

Callbacks on task states

Uses: stats, state tracking, etc

```
SCX_OPS_DEFINE(simple_ops,
           .select_cpu    = (void *)simple_select_cpu,
           .enqueue       = (void *)simple_enqueue,
           .dispatch      = (void *)simple_dispatch,
           .running       = (void *)simple_running,
           .stopping      = (void *)simple_stopping,
           .enable        = (void *)simple_enable,
           .init          = (void *)simple_init,
           .exit          = (void *)simple_exit,
           .name          = "simple");
```

How scheduling works: select_cpu

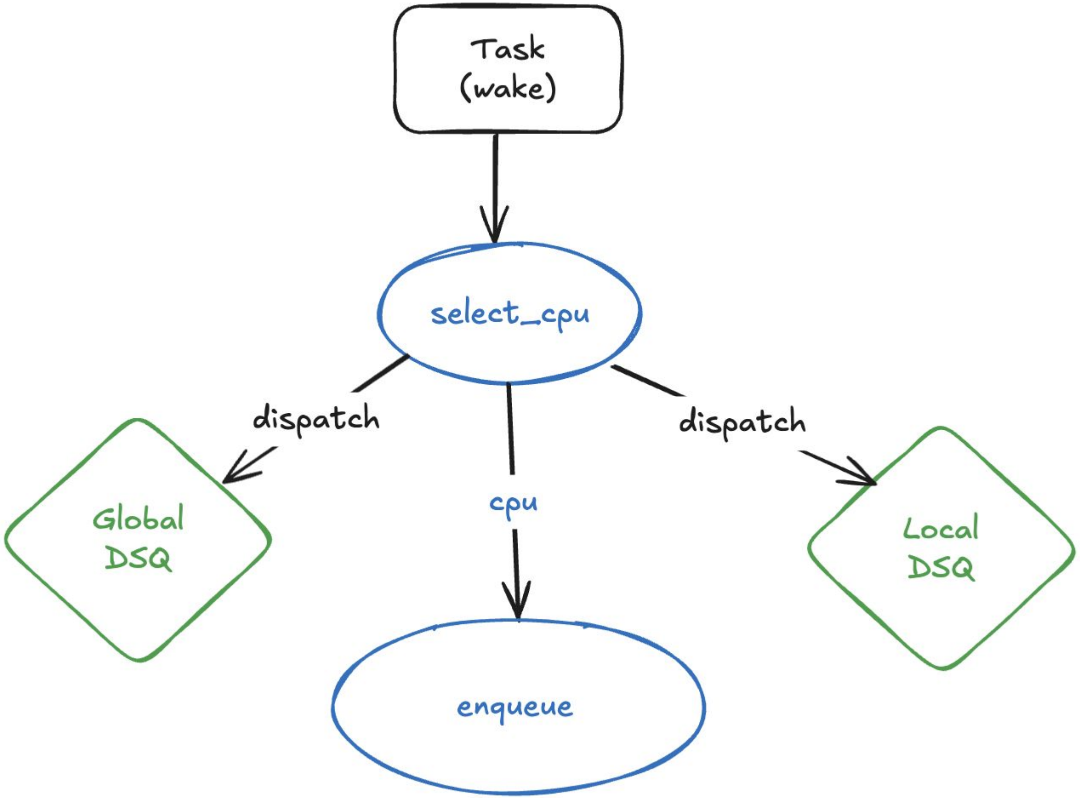
Dispatch to a DSQ

Select a CPU as a hint

Next step enqueue

```
SCX_OPS_DEFINE(simple_ops,
               .select_cpu     = (void *)simple_select_cpu,
               .enqueue        = (void *)simple_enqueue,
               .dispatch       = (void *)simple_dispatch,
               .running        = (void *)simple_running,
               .stopping       = (void *)simple_stopping,
               .enable         = (void *)simple_enable,
               .init           = (void *)simple_init,
               .exit           = (void *)simple_exit,
               .name           = "simple");
```

Task


Task
(wake)
select_cpu
dispatch
dispatch
cpu
Global
DSQ
Local
DSQ
enqueue

select_cpu

select_cpu

enqueue

dispatch

```
s32 BPF_STRUCT_OPS(simple_select_cpu,
struct task_struct *p, s32 prev_cpu, u64 wake_flags)
{
    bool is_idle = false;
    s32 cpu;

    cpu = scx_bpf_select_cpu_dfl(
            p, prev_cpu, wake_flags, &is_idle);
    if (is_idle) {
        stat_inc(0); /* count local queueing */
        scx_bpf_dispatch(p, SCX_DSQ_LOCAL,
                            SCX_SLICE_DFL, 0);
    }

    return cpu;
}
```

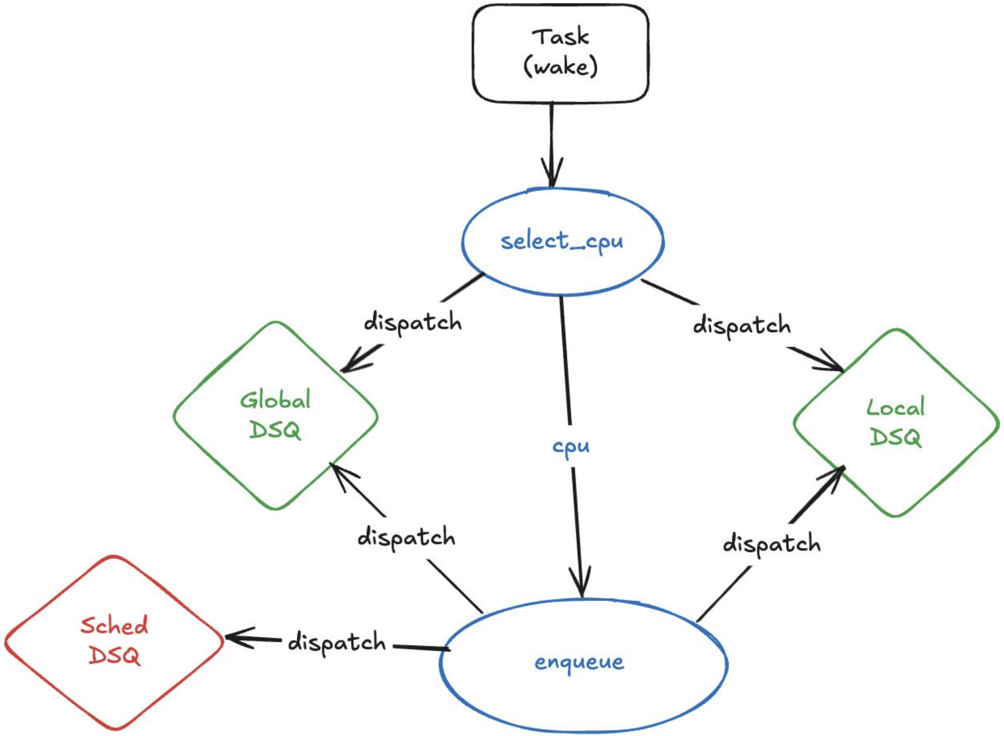
How scheduling works: Enqueue

Dispatch to a DSQ

Enqueue to BPF scheduler

```
SCX_OPS_DEFINE(simple_ops,
           .select_cpu     = (void *)simple_select_cpu,
           .enqueue        = (void *)simple_enqueue,
           .dispatch       = (void *)simple_dispatch,
           .running        = (void *)simple_running,
           .stopping       = (void *)simple_stopping,
           .enable         = (void *)simple_enable,
           .init           = (void *)simple_init,
           .exit           = (void *)simple_exit,
           .name           = "simple");
```

sched_ext_ops: enqueue



enqueue

select_cpu

enqueue

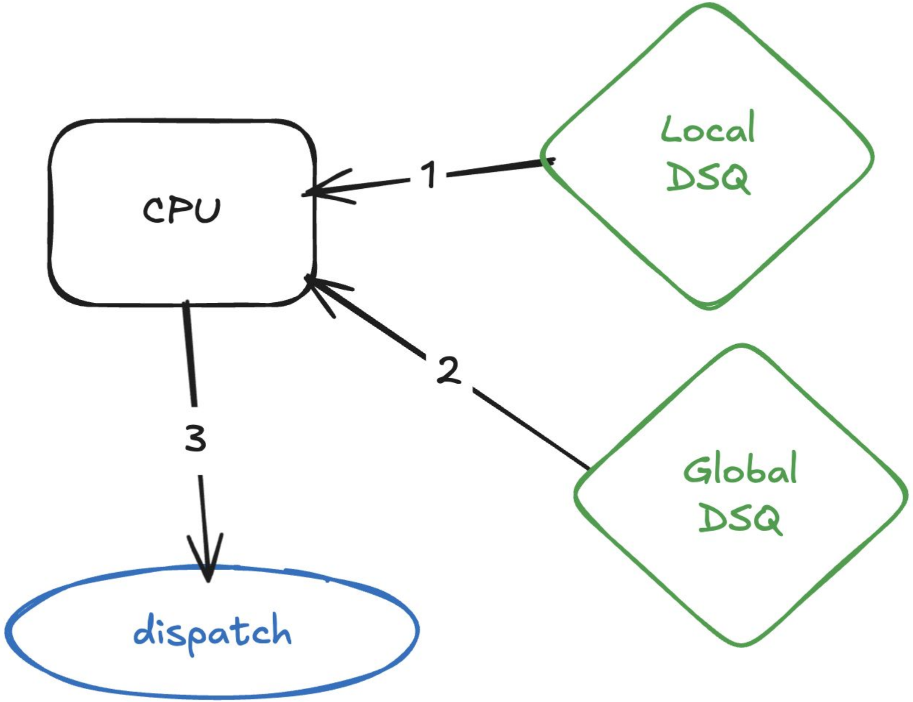
dispatch

```
void BPF_STRUCT_OPS(simple_enqueue, struct task_struct *p,
                    u64 enq_flags)
{
        u64 vtime = p->scx.dsq_vtime;

        // Limit the budget to one slice.
        if (vtime_before(vtime, vtime_now -
            SCX_SLICE_DFL))
            vtime = vtime_now - SCX_SLICE_DFL;

        scx_bpf_dispatch_vtime(p, SHARED_DSQ,
                               SCX_SLICE_DFL, vtime, enq_flags);
}
```

CPU ready to schedule



How scheduling works: Dispatch

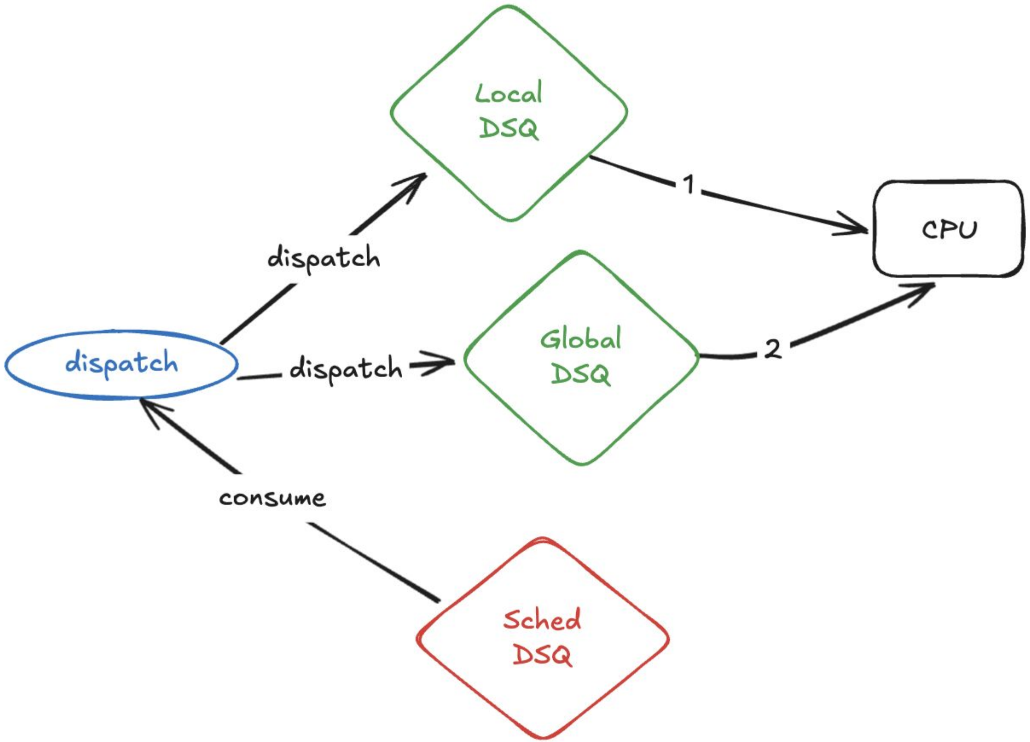
Local DSQ empty

Dispatch to DSQs

Enqueue to BPF scheduler

```
SCX_OPS_DEFINE(simple_ops,
           .select_cpu    = (void *)simple_select_cpu,
           .enqueue       = (void *)simple_enqueue,
           .dispatch      = (void *)simple_dispatch,
           .running       = (void *)simple_running,
           .stopping      = (void *)simple_stopping,
           .enable        = (void *)simple_enable,
           .init          = (void *)simple_init,
           .exit          = (void *)simple_exit,
           .name          = "simple");
```

sched_ext_ops: dispatch



dispatch

select_cpu

enqueue

dispatch

```
void BPF_STRUCT_OPS(simple_dispatch, s32 cpu,
                    struct task_struct *prev)
{
    scx_bpf_consume(SHARED_DSQ);
}
```

What have we learned

Basics of how to build a scheduler

kfuncs, bpf_cpumasks, struct_ops

scx_simple.bpf.c for complete example

Schedulers: Active Development

scx_rustland/scx_bpfland

scx_lavd

scx_rusty

scx_layered

scx_bpfland/scx_rustland

Interactive workloads

Scheduling in userspace!

Context switches

scx_lavd: Latency-criticality Aware Virtual Deadline

Portable gaming

Task graph

Core compaction

```
  1[||                                   0.7%  840MHz 44°C]   7[                                   0.0%  896MHz 44°C]
  2[|                                    0.0%  855MHz N/A]    8[                                   0.0%  858MHz N/A]
  3[||                                   0.7%  831MHz N/A]    9[                                   0.0%  876MHz N/A]
  4[|                                    0.0% 1010MHz N/A]   10[                                   0.0%  859MHz N/A]
  5[                                     0.0%  830MHz 47°C]  11[                                   0.0%  900MHz 47°C]
  6[                                     0.0%  858MHz N/A]   12[                                   0.0%  846MHz N/A]
Uptime: 14 days, 11:11:22                                     Battery: N/A
Mem:46.6G used:1.82G shared:11.9M compressed:0K buffers:2.61M cache:17.4G available:44.3G    Tasks: 40, 77 thr, 184 kthr; 1 running
zrm:0K used:0K uncompressed:0K                                Load average: 0.05 0.03 0.00
Swp:0K used:0K cache:0K frontswap:0K                          PSI some CPU:    99.00% 99.00% 99.00%
Disk IO: 0.0% read: 0KiB/s write: 0KiB/s                      PSI full IO:      0.00%  0.00%  0.00%
Network: rx: 1KiB/s tx: 39KiB/s (13/23 pkts/s)                PSI full memory:  0.00%  0.00%  0.00%
```

scx_rusty

NUMA/Multi CCX

Load balancing

General Purpose

scx_layered

Scheduling policy per layer

Widely deployed across the fleet Meta

Soft affinity domains

scx_layered

JSON!?

Confined,
Grouped,
Open

Matches

```
[{
  "name":"simple",
  "comment":"it's easy",
  "matches":[[]],
  "kind":{
    "Open": {
      "preempt": false,
      "slice_us": 800
    }
  }
}]
```

scx_layered

Soft Affinity (Grouped)

Frequency control (schedutil)

Complex config

```
[{
  "name":"hodgesd",
  "comment":"hodgesd user",
  "matches":[
     [{"UIDEquals":12345}]
  ],
  "kind": {
    "Grouped":{
      "util_range": [0.05, 0.6],
      "slice_us": 1000,
      "preempt": true,
      "preempt_first": true,
      "perf": 1024
     }
  }
},
{
  "name":"normal",
  "comment":"the rest",
  "matches":[[]],
  "kind":{
    "Confined": {
      "util_range": [0.25, 0.6],
      "preempt": false,
      "slice_us": 500,
      "perf": 512
    }
  }
}]
```

Schedulers: Performance

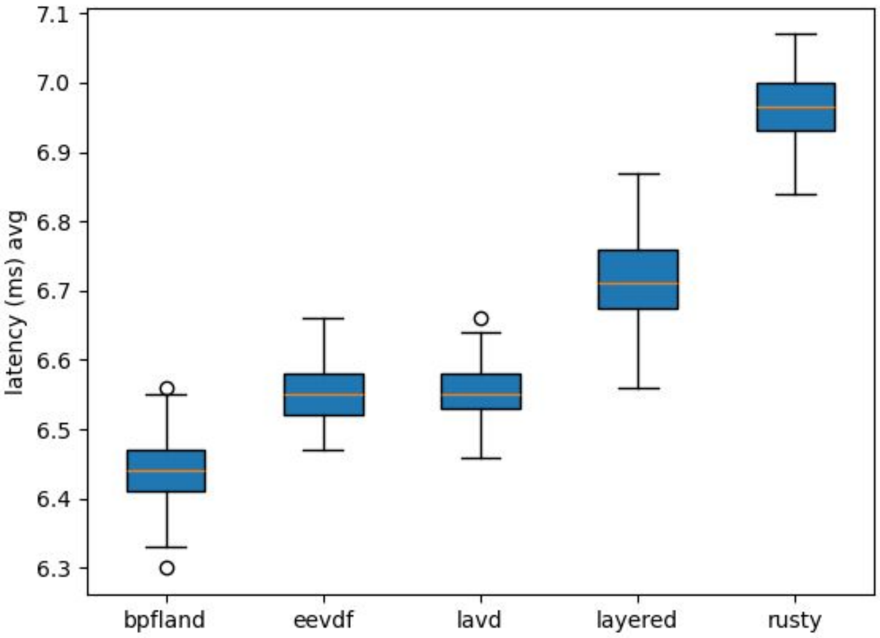
Can BPF schedulers perform as well as in kernel? Yes!

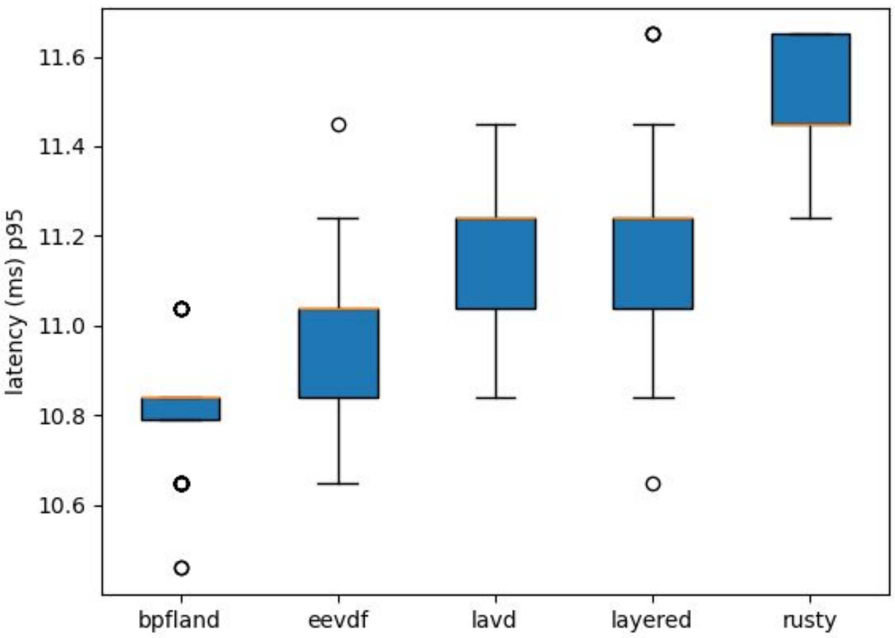
Complex workloads/architectures require scheduler tuning

Benchmarks -> default settings, no tuning

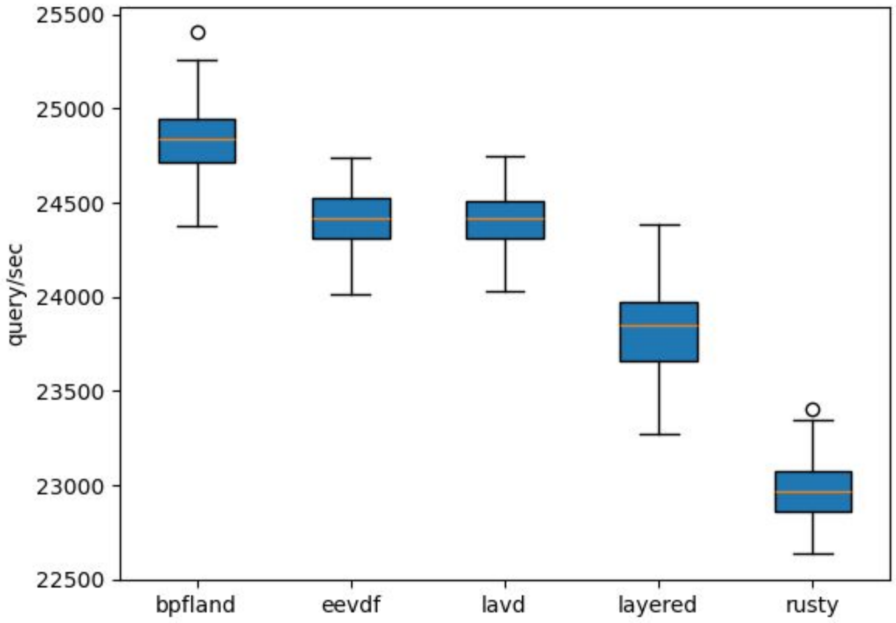
Sysbench: 1.0.20 MySQL benchmarks
System: Gentoo 6.12.0-rc3
Processor: AMD 7940HX
MySQL: 8.0.36
100 consecutive iterations

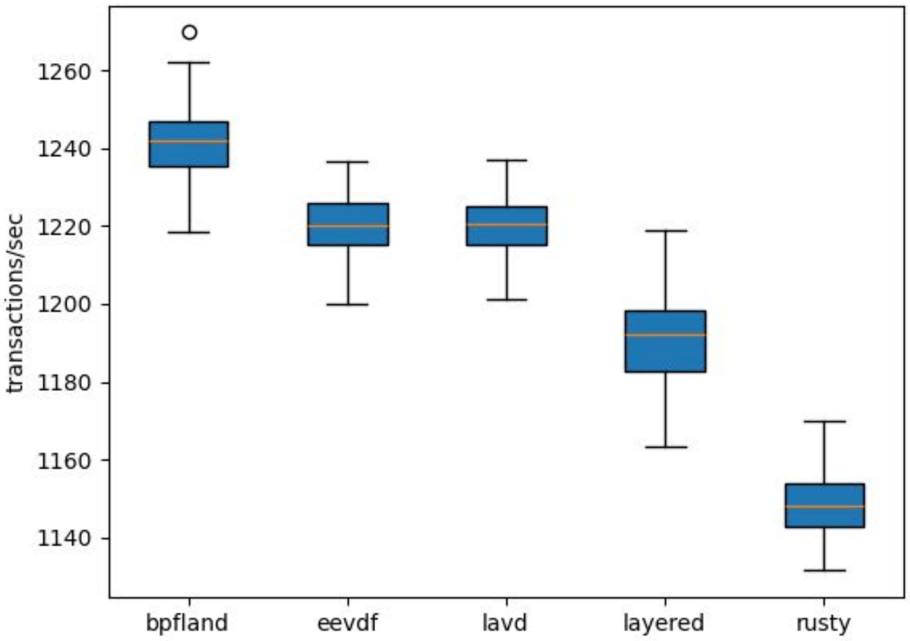
Schedulers: Sysbench MySQL





Schedulers: Sysbench MySQL





Challenges

Backporting

Complex Architectures

Testing

Scheduler Observability

BPF

Kernel Backports

Have a strategy for kernel upgrades!

Backporting/testing can be a full time job, avoid it!

Alternative: wait 2 years for kernel upgrades

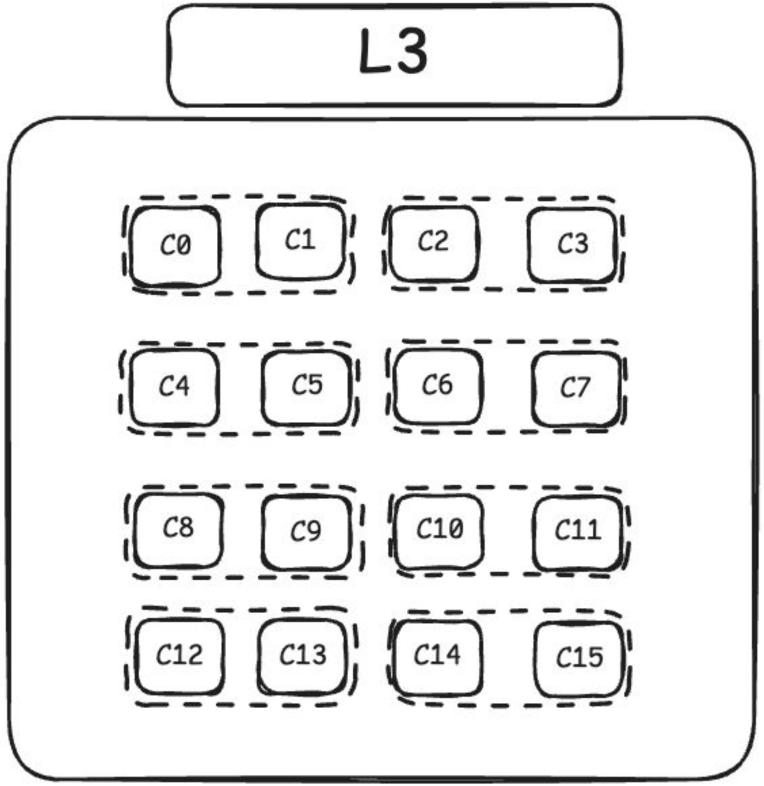
Simple Architectures

When scheduling was "easy"

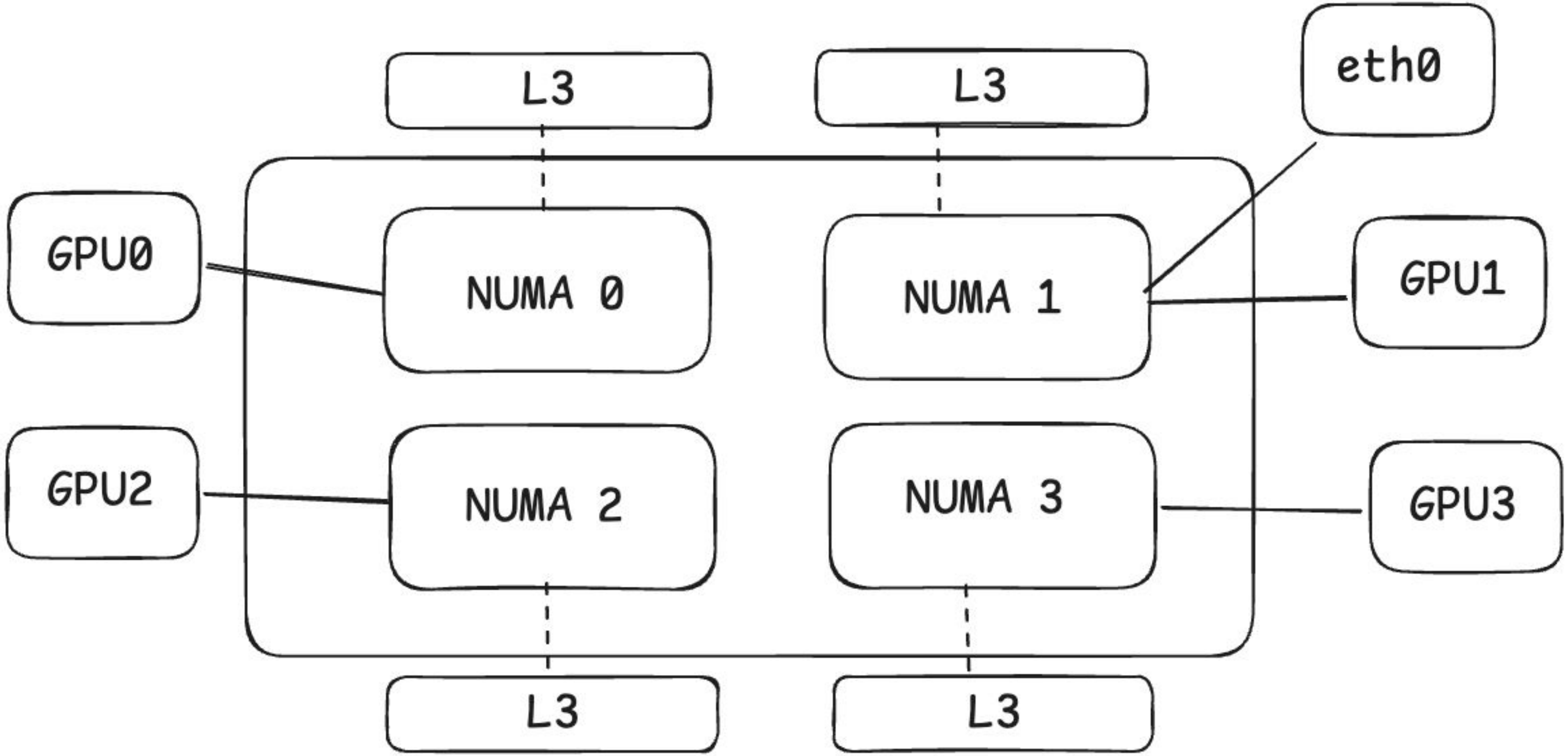
Unified cache

SMT?

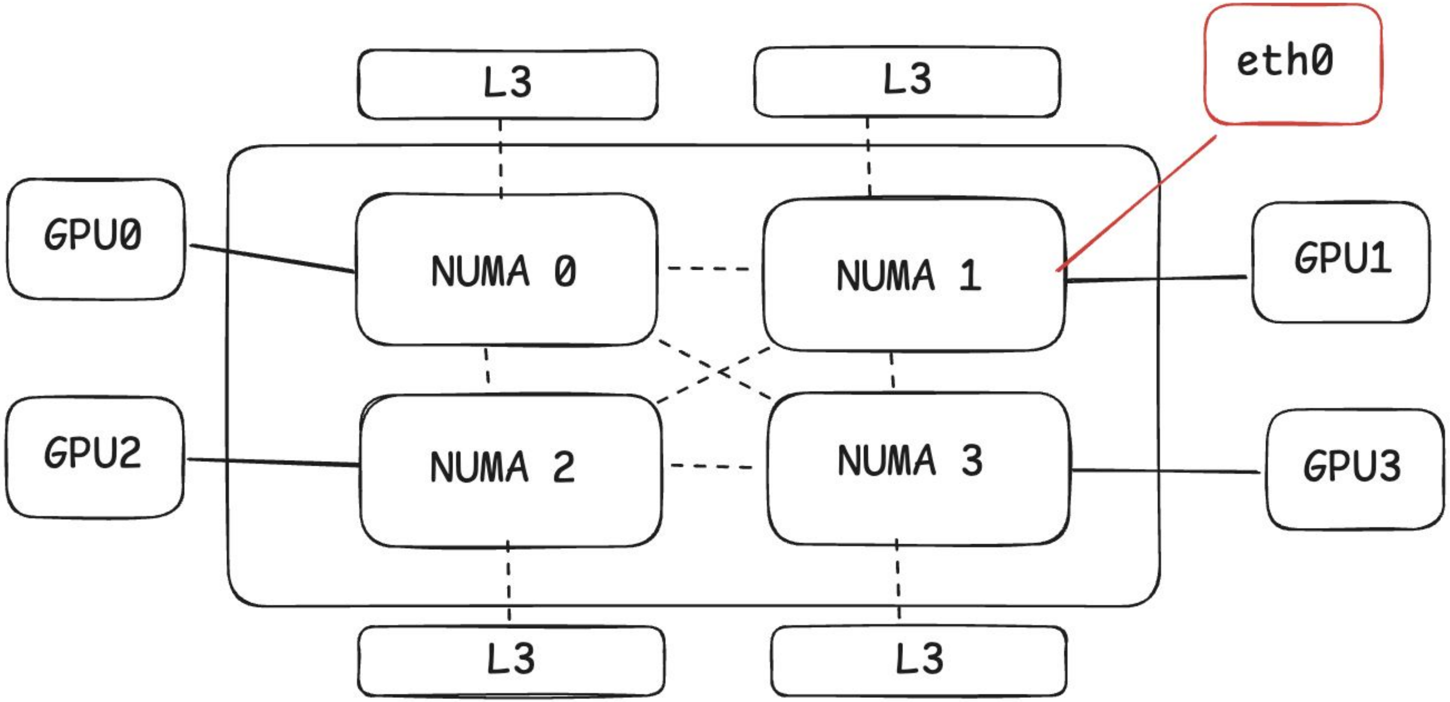
No big/little cores



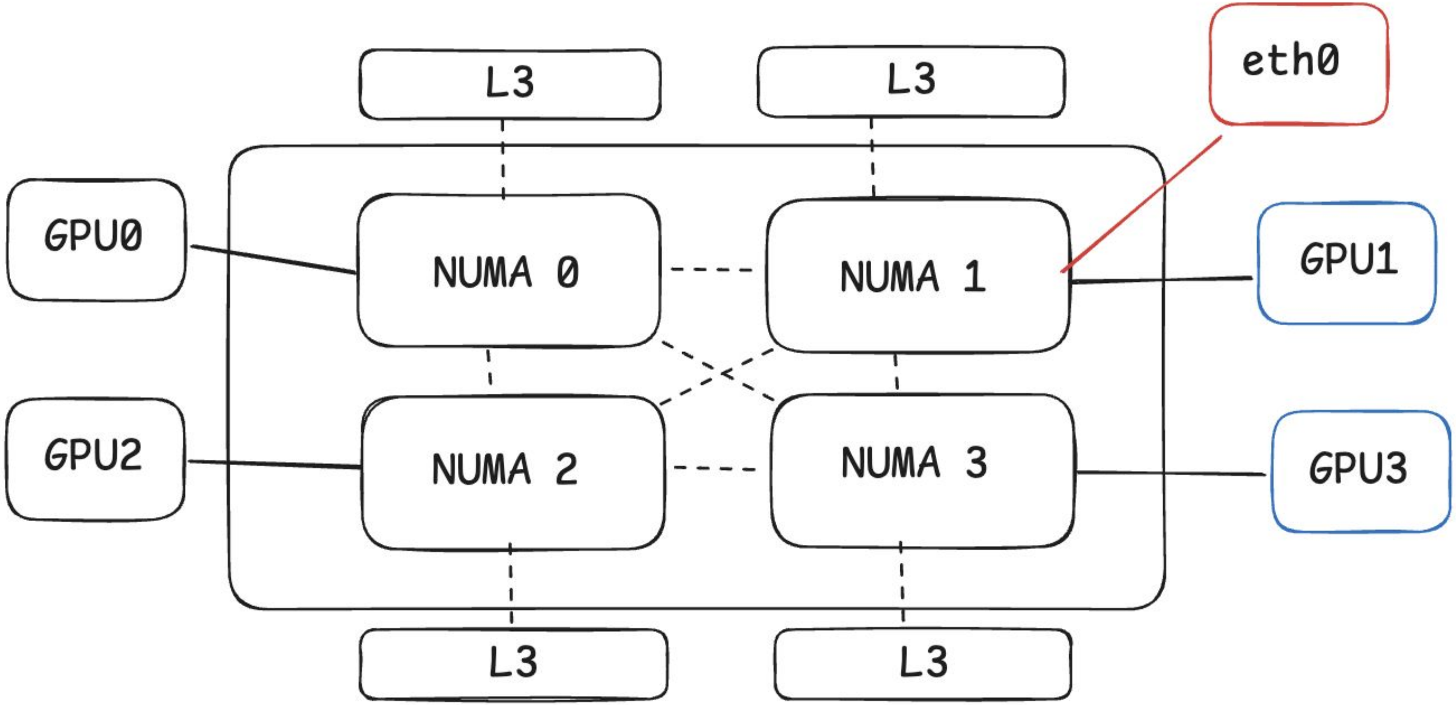
Complex Architectures



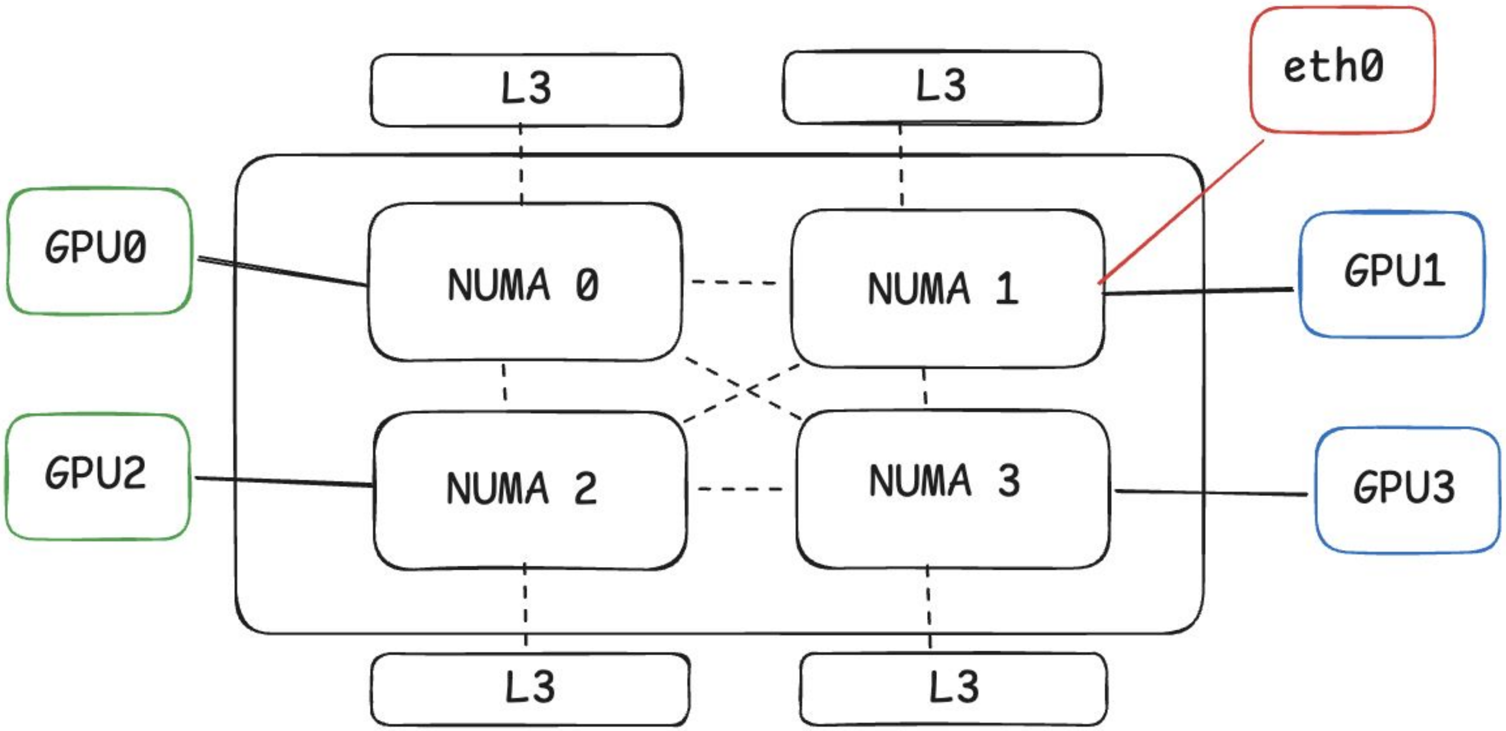
Complex Architectures: Networking



Complex Architectures: GPU Inference



Complex Architectures: GPU Training



Testing: Correctness

Testing scheduler changes is not easy

Did the scheduler make the right decision?

Testing in production (bad)

bpftrace for testing (sched_switch)

Testing: Performance

stress-ng: general purpose scheduler benchmarks

Workload specific!

```
 stress-ng: util/dcycle/frac/3003.3/ 93.1/   93.1 load/load_frac_adj/frac=    30.00/83.34/ 24.1 tasks=    31
            tot=     30 local= 0.00 wake/exp/reenq= 0.00/100.0/ 0.00
            xlayer_wake= 0.00 xlayer_rewake= 0.00
            keep/max/busy=32010.0/173.3/ 0.00 kick=100.0 yield/ign= 0.00/    0
            open_idle=66.67 mig=86.67 xnuma_mig=16.67 xllc_mig=16.67 affn_viol= 0.00
            preempt/first/xllc/xnuma/idle/fail= 0.00/ 0.00/ 0.00/ 0.00/ 0.00/ 0.00 min_exec= 0.00/   0.00ms, slice=2ms
            cpus= 32 [ 32, 32] fff00000 f000000f 00000fff

stress-ng: info:  [1314132] skipped: 0
stress-ng: info:  [1314132] passed: 30: cpu (30)
stress-ng: info:  [1314132] failed: 0
stress-ng: info:  [1314132] metrics untrustworthy: 0
stress-ng: info:  [1314132] successful run completed in 20.01 secs
$ sudo stress-ng -c 30 -t 20 -M
stress-ng: info:  [1320195] setting to a 20 secs run per stressor
stress-ng: info:  [1320195] dispatching hogs: 30 cpu
stress-ng: metrc: [1320195] stressor       bogo ops real time  usr time  sys time   bogo ops/s     bogo ops/s CPU used per       RSS Max
stress-ng: metrc: [1320195]                           (secs)    (secs)    (secs)   (real time) (usr+sys time) instance (%)          (KB)
stress-ng: metrc: [1320195] cpu              710265     20.00    597.71      0.25     35510.06        1187.82        99.65          5120
stress-ng: info:  [1320195] skipped: 0
stress-ng: info:  [1320195] passed: 30: cpu (30)
stress-ng: info:  [1320195] failed: 0
stress-ng: info:  [1320195] metrics untrustworthy: 0
```

Scheduler Observability

Scheduler stats -> metrics/monitoring

Scheduler tracing

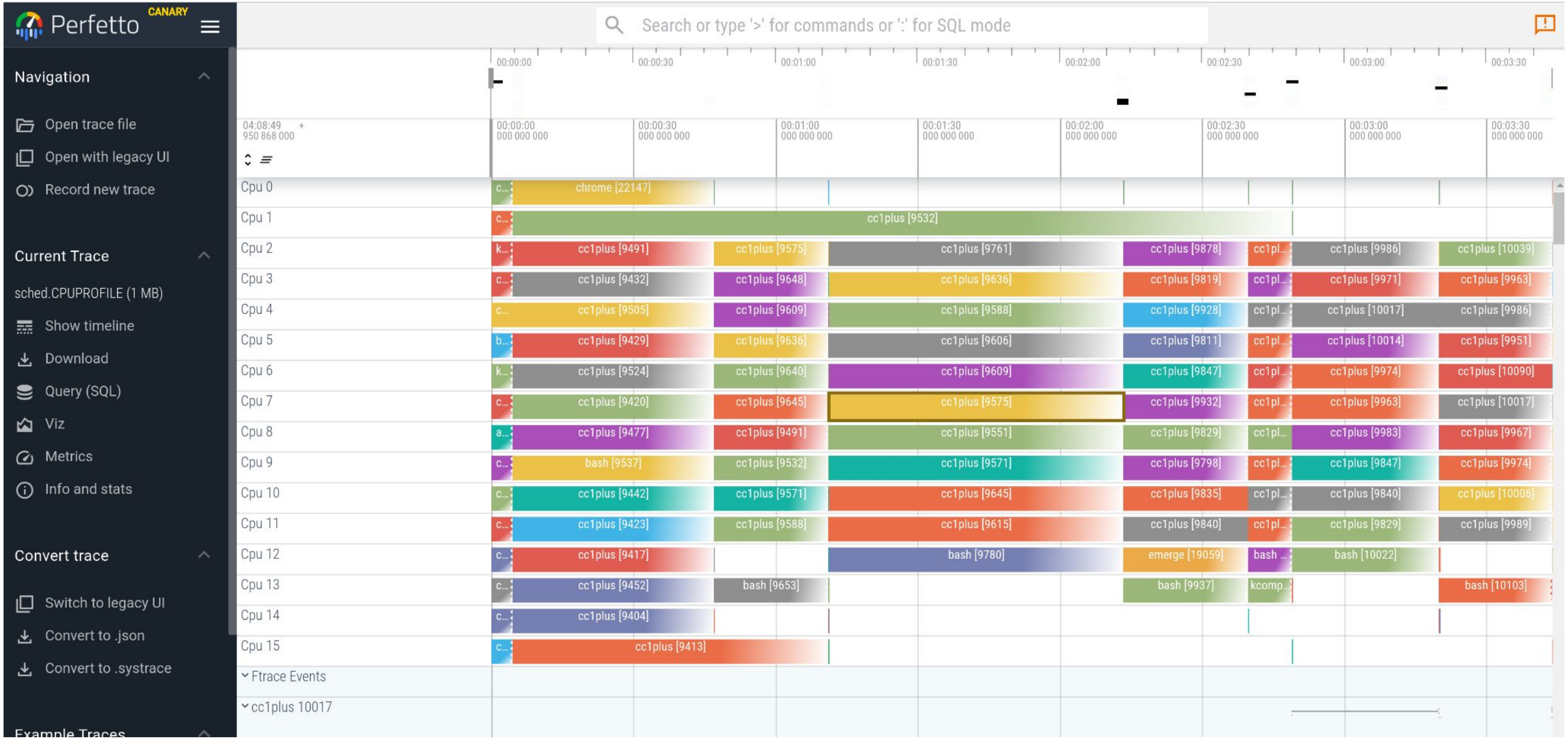
Bpftrace- custom scripts

```
scheduler: layered
@avg_lat: 50

@usec_hist:
[0]                67809 |@@@@@@@@@@@@@@@@@                                   |
[1]               204104 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[2, 4)            162973 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           |
[4, 8)            161962 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           |
[8, 16)           103171 |@@@@@@@@@@@@@@@@@@@@@@@@@@                          |
[16, 32)           40731 |@@@@@@@@@@                                          |
[32, 64)           14619 |@@@                                                 |
[64, 128)           5573 |@                                                   |
[128, 256)          2459 |                                                    |
[256, 512)          1128 |                                                    |
[512, 1K)            484 |                                                    |
[1K, 2K)             161 |                                                    |
[2K, 4K)             127 |                                                    |
[4K, 8K)             103 |                                                    |
[8K, 16K)             63 |                                                    |
[16K, 32K)            60 |                                                    |
[32K, 64K)            41 |                                                    |
[64K, 128K)           34 |                                                    |
[128K, 256K)          11 |                                                    |
[256K, 512K)           2 |                                                    |
[512K, 1M)             4 |                                                    |
[1M, 2M)               4 |                                                    |
[2M, 4M)               4 |                                                    |

@dsq_lat[17]: 7
@dsq_lat[1026]: 12
@dsq_lat[14]: 14
@dsq_lat[38]: 15
@dsq_lat[34]: 16
@dsq_lat[13]: 21
@dsq_lat[20]: 26
@dsq_lat[40]: 26
@dsq_lat[33]: 26
@dsq_lat[41]: 27
@dsq_lat[19]: 30
@dsq_lat[11]: 31
@dsq_lat[42]: 33
@dsq_lat[39]: 35
@dsq_lat[37]: 48
@dsq_lat[0]: 50
@dsq_lat[21]: 63
@dsq_lat[43]: 97
@dsq_lat[35]: 125
@dsq_lat[36]: 138
@dsq_lat[16]: 481
```

Perfetto: Visualizing Scheduler Traces



System metrics

Context switches

RQ delay

LLC hit ratio

IPC

Future Direction

Composable schedulers

Concurrent scheduler support

Schedulers in JVM/bpftrace?

Conclusion

How to build a scheduler

Challenges

Schedulers- We need you!

sched_ext

Meta