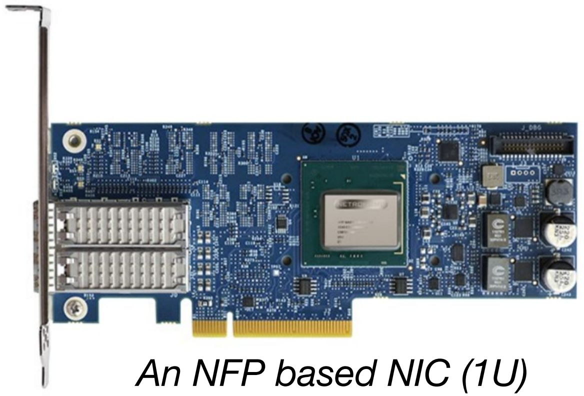
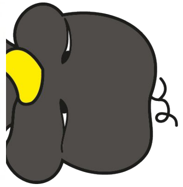
Comprehensive BPF offload

Nic Viljoen & Jakub Kicinski
Netronome Systems


An NFP based NIC (1U)



netdev 2.2, Seoul, South Korea
November 8-10, 2017

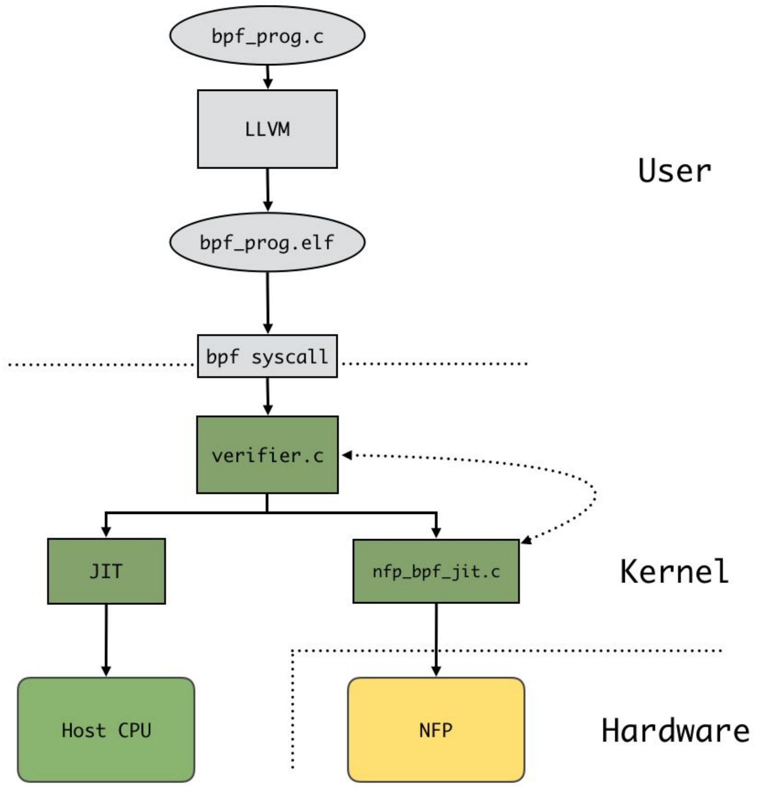
Agenda

- Refresher
  - Programming Model
  - Architecture
- Performance & Optimization
- Requirements for Production Offload
  - bpftool
  - Verifier restructuring

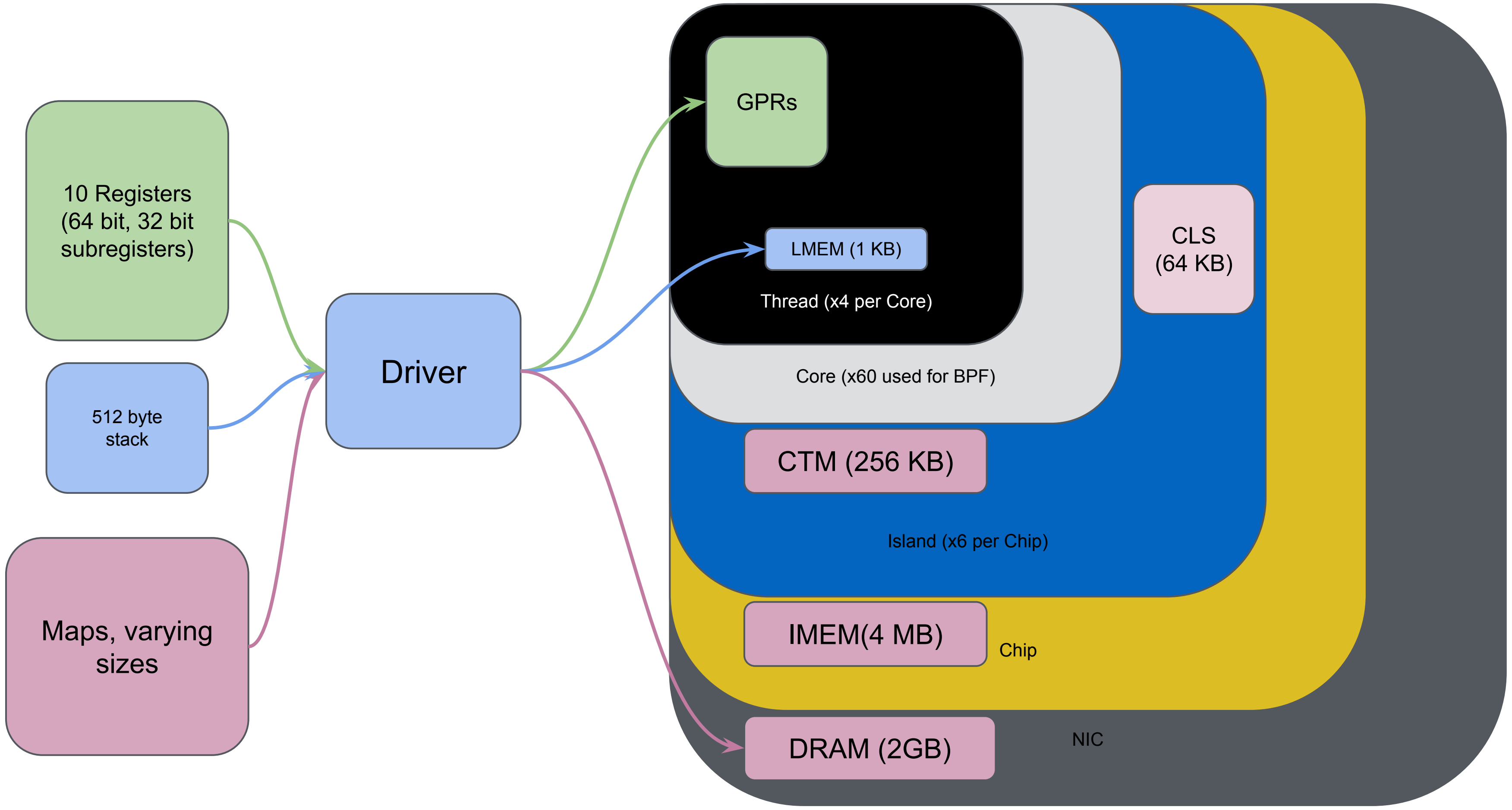
Programming Model *(refresher)*

NETRONOME

- Program is written in standard manner
- LLVM compiled as normal
- iproute/tc loads the program requesting offload
- The *nfp_bpf_jit.c* converts the eBPF bytecode to NFP machine code (and we mean the actual machine code :))
- Translation reuses a significant amount of verifier infrastructure



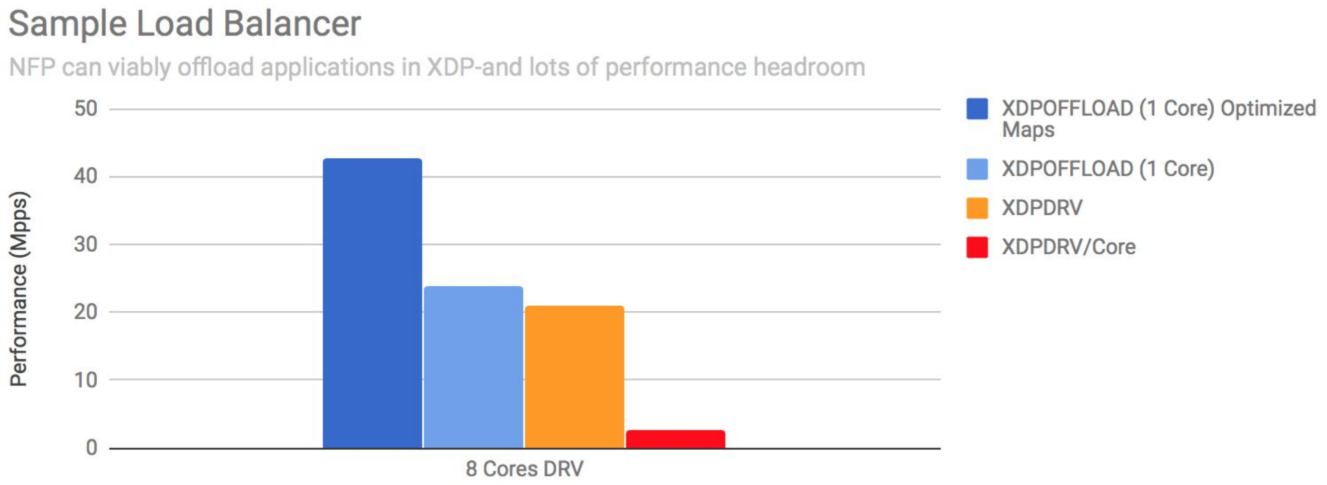
Refresher-BPF Offload Mapping



Performance

NETRONOME

- Simple XDP load balancer (~ 800 BPF insns, 4 lookups)
  - Based on the TC example in kernel - **selftests/bpf/l4lb.c**
  - Combined with **samples/bpf/xdp_tx_iptunnel_kern.c**
- Per CPU array changed to standard array to run offloaded
  - There is no nice equivalent for per CPU at the moment on the NIC
- Not optimised-big health warning :)



Future Optimizations

- Map placement/caching-as shown on previous page
- Using Packet Cache-reduce latency of packet accesses from ~50 cycles to ~3 cycles
- 32 bit ALU from LLVM where possible-reduce ALUs from ~ 4 machine code insns to 1
- Remove FW locks-double memory bandwidth

Requirements For Production Readiness

Multi-stage processing:

- Reliable manner to run some programs in host if not possible/desirable in offload

Debug:

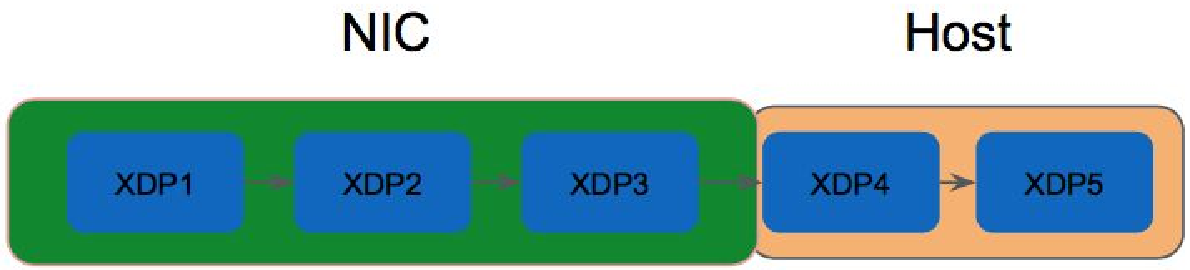
- Usable verifier error messages
- Introspection-both of maps and programs

JIT:

- Translation before optimization

Multi-Stage Processing

- Offload some programs
  - This can be managed by the driver (implicitly) or explicitly
  - Use `data_meta` to inject programs into the correct BPF program to run next
    - Important for edge case where the next program to run is not fixed
  - Allows offload to be used for beneficial cases only
    - Can be explicitly via flags



Progress made - kernel

Upstream:

- new instructions (Daniel, Jiong, I);
- direct packet access;
- stack support;
- adjust head helper;
- add 32-bit subregister support to LLVM (Jiong).

Prototyped/PoC:

- map offload support (hash and array maps);
- atomic add operation;
- memcpy optimizations (Jiong);
- initiate work on register state tracking (Jiong).

Progress made - tooling

bpftool

- in kernel tree for Linux 4.15 (in the `tools/` directory);
- iproute2-like syntax;
- list and pin objects;
- programs:
  - show type, name, tag, id, memory usage, load time, used maps;
  - dump JITed and translated images (to file or print instructions);
- maps:
  - show type, name, id, key/value size, number of elements, flags;
  - lookup, update, delete, etc.
- JSON output (Quentin);
- BPF FS integration (Quentin, Prashant).

Progress made - tooling

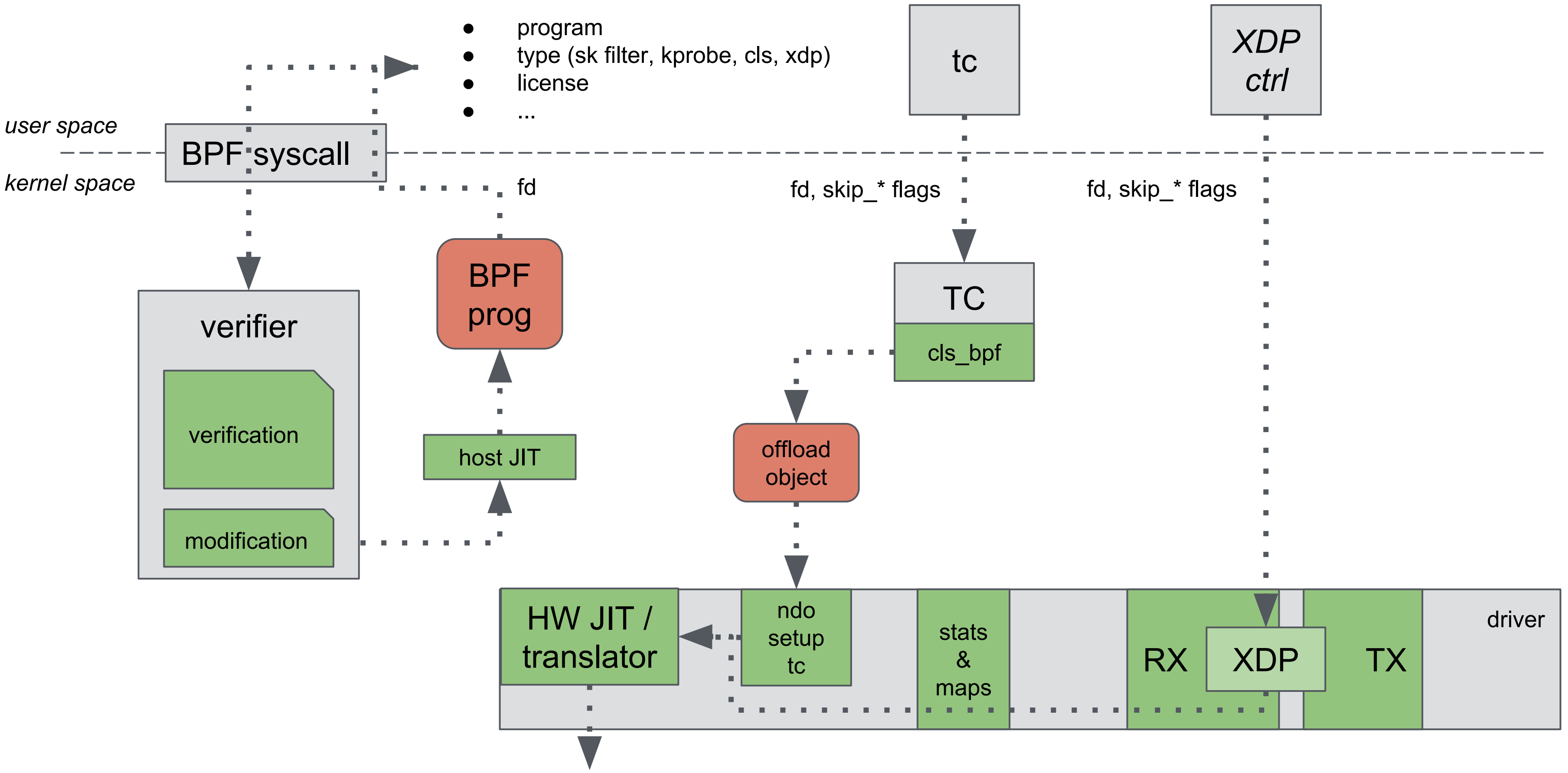
llvm-mc (Jiong)

- upstream for LLVM 6.0;
- LLVM's macro assembler;
- verifier-style syntax:

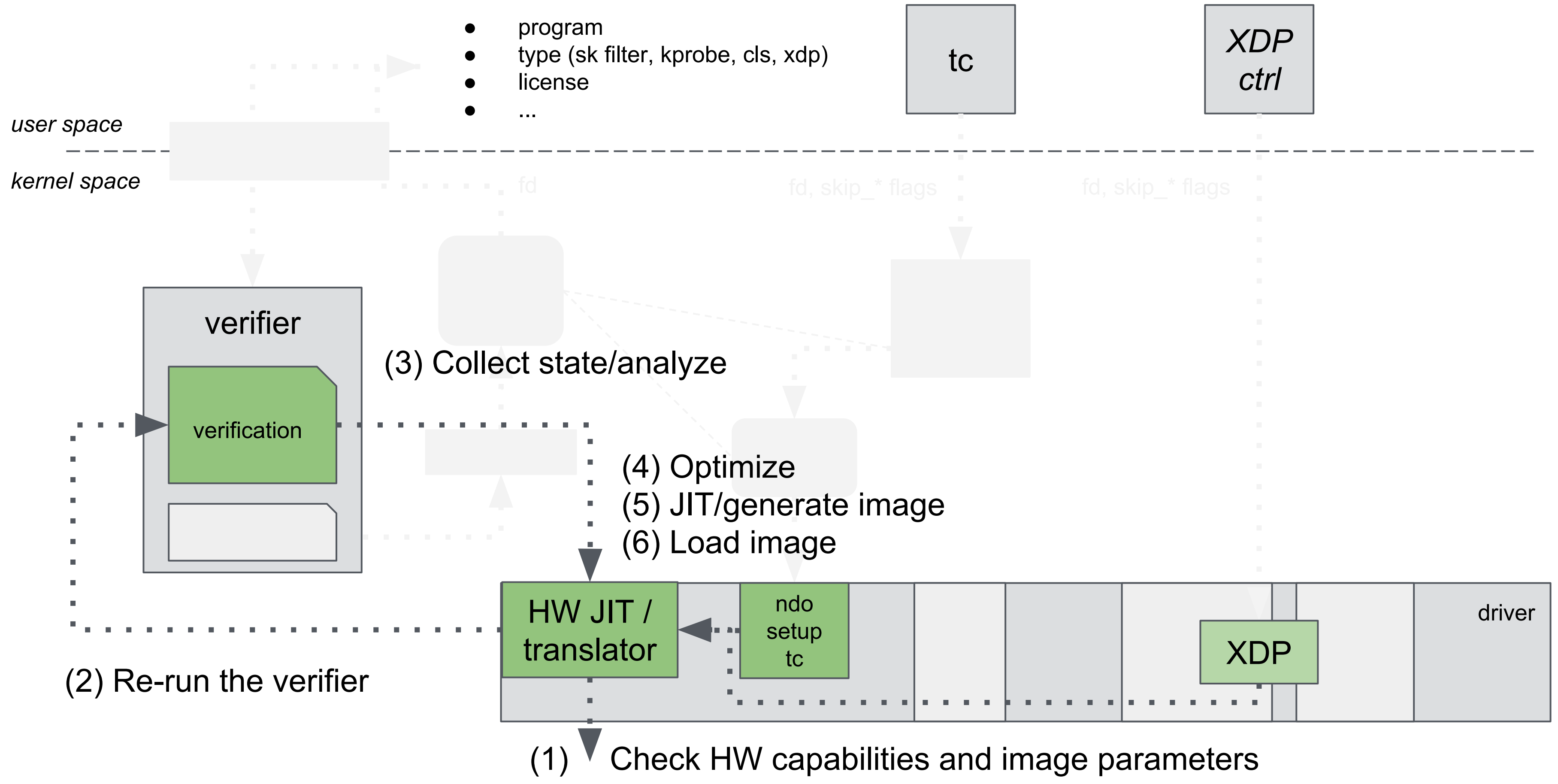
```
r1 = r6
r2 = 0xff000000 ll
call 12
r0 = 0
exit
```

- allows hand-crafting precise BPF programs (or compiling C code into assembly and modifying it);
- opens way for BPF inline assembly;
- very useful for testing particular instruction sequences.

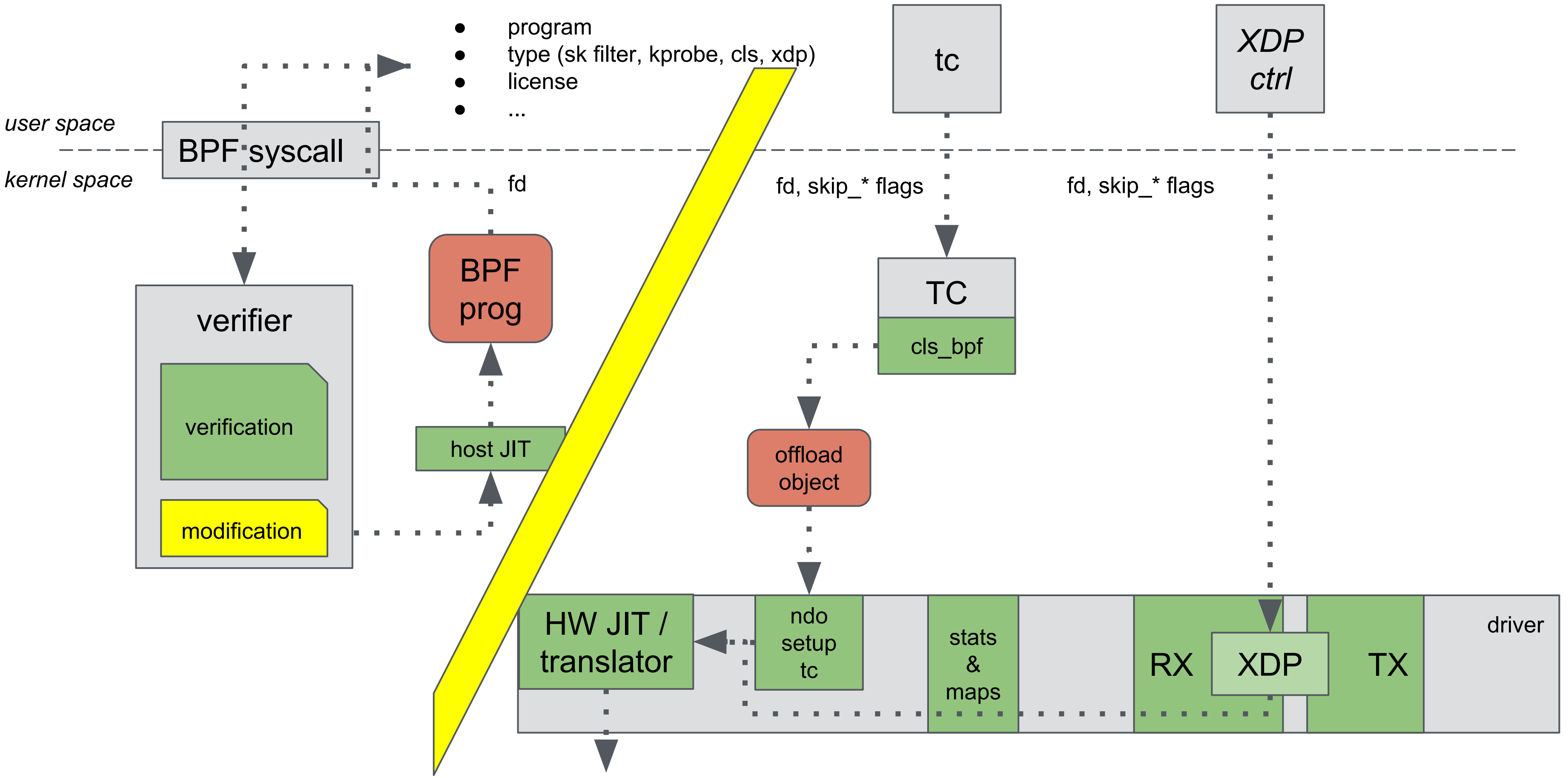
Kernel basics (*refresher*)



Translation and loading *(refresher)*

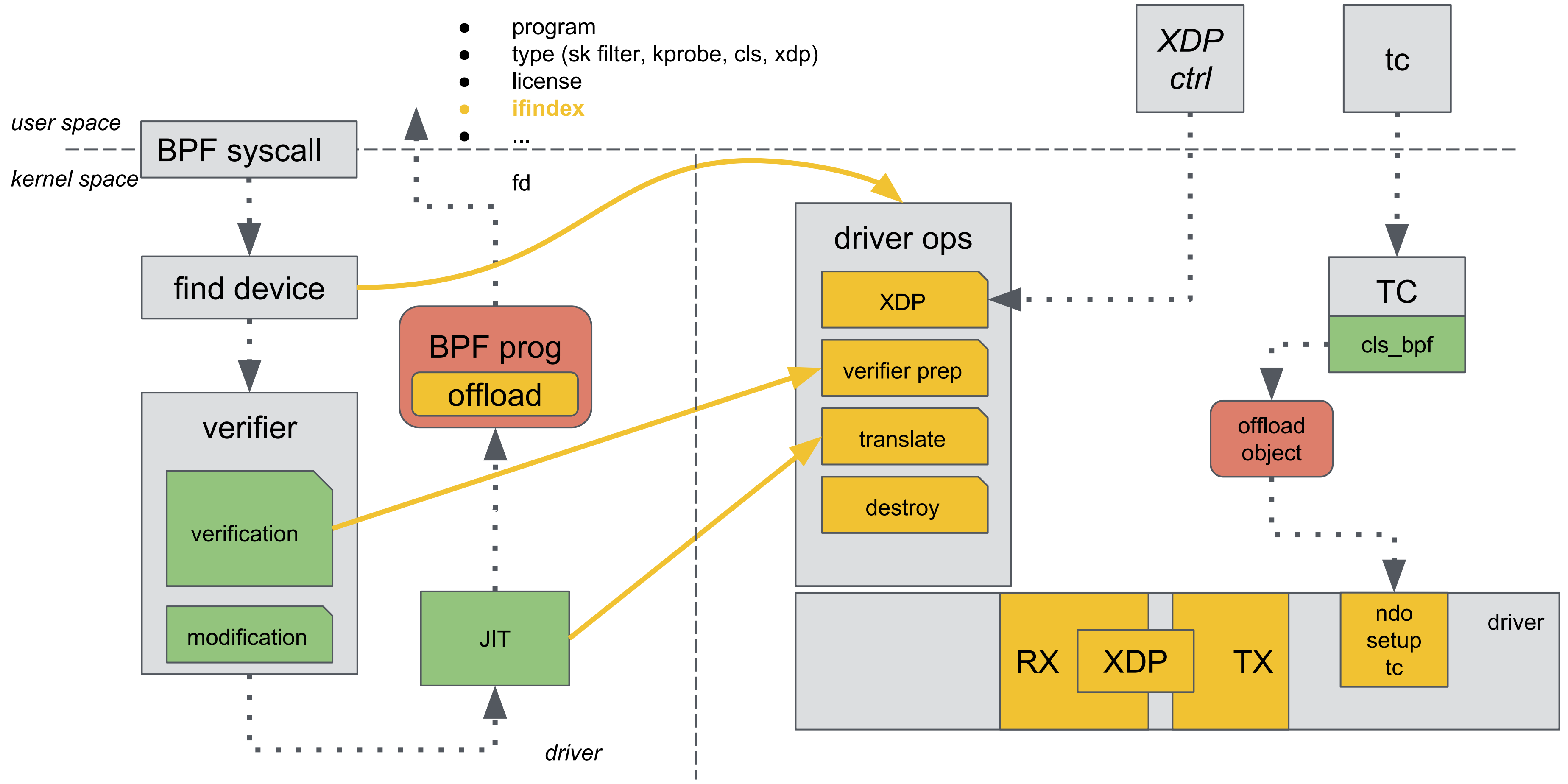


Kernel basics (*refresher*)



Progress made - kernel

NETRONOME



Rationale for recent kernel changes

- allow device translator to access the loaded program as-is:
  - IDs/offsets not translated:
    - structure field offsets;
    - functions;
    - map IDs.
  - no prolog/epilogue injected;
  - no optimizations made;
- output errors at program load and map creation time;
- make use of access to the verifier log;
- include device information in introspection APIs (bpftool);
- dump translated image:
  - similar to host “JITed image”;
  - BPF core already has access to offload state (no longer driver black box);
  - need to report machine info?

Debug and tooling APIs

- netlink extack support in cls_bpf/TC offloads:
  - XDP already carries extack for use by the drivers;
  - allows easier error reporting at attachment time;
- bpf perf event output - output samples for debugging the datapath;
- simple API for enabling/disabling optimizations:
  - verifier/kernel already has some simple optimizations (e.g. lookup inlining);
  - nfp translator already has a few and we expect to add more;
  - need to report, enable/disable optimizations with nice granularity;
- maps:
  - create maps on the device from the start;
  - simplify map load/eviction and locking greatly;
  - report errors/resource exhaustion at map creation time.

The end

Thank you!