



Using DPDK on embedded RISC-V cores of a NIC

Dmitry Kozlyuk,  Mitigator Global

Programmable HW for anti-DDoS

- We want NIC to decide which packets to keep when CPU is overloaded.
- Offloading?
 - Performance limited by packet rate and rule structure (JUMPs)
 - Fragmented traffic, invalid packets, broken sessions
 - Limited support for protocol fields and checks
- Challenge-Response protections
- Mutable custom state tables

BlueField 3 Data Path Accelerator (DPA)

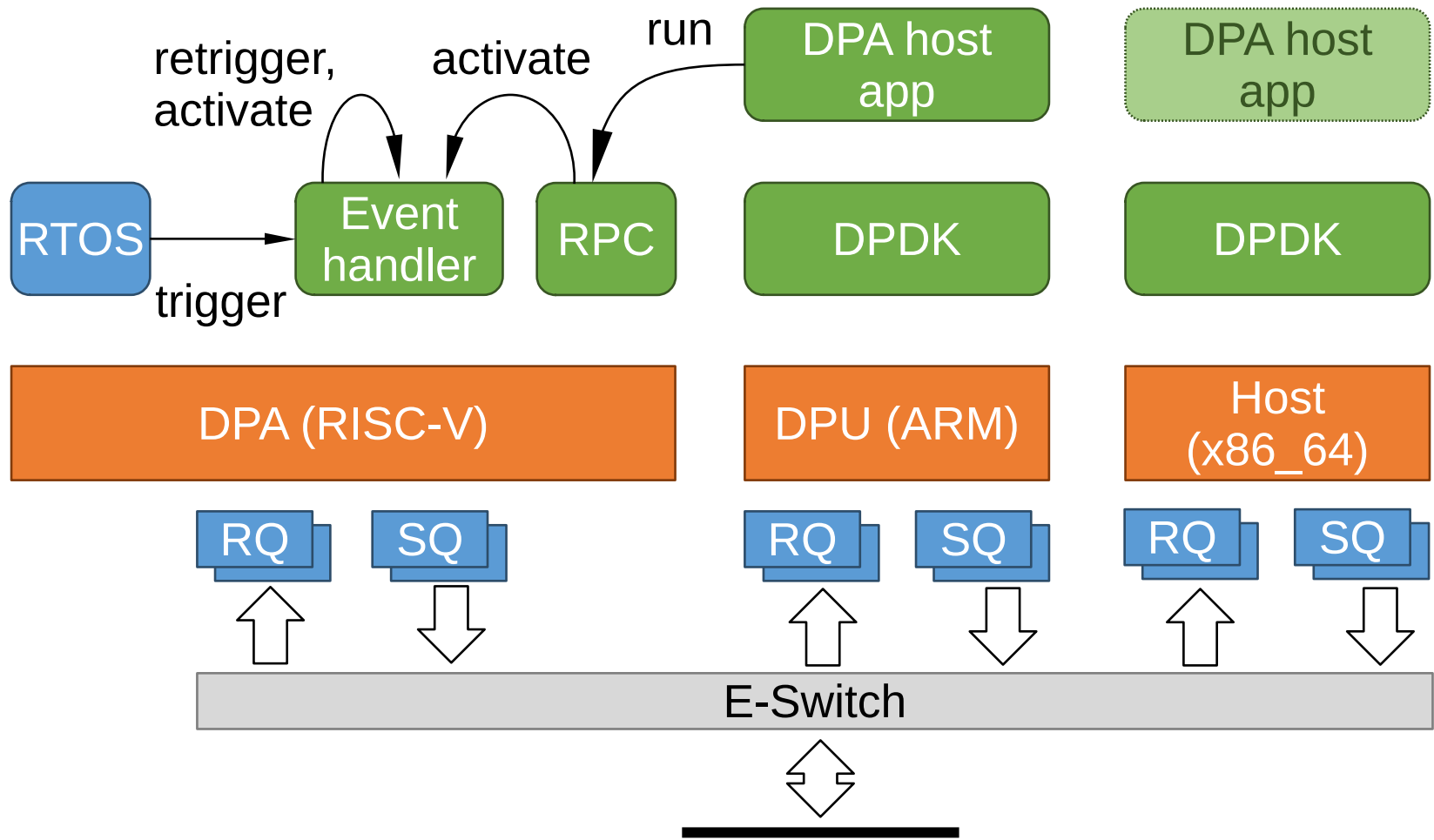
- BlueField 2: Data Processing Unit (DPU) = SmartNIC + ARM running Linux

- BlueField 3 adds DPA

- ARM for control plane, RISC-V for data plane
- DPA runs RTOS, apps on DPU or host spawn DPA processes
- DPA can access DPU and host memory with latency cost
- NIC can DMA packets to DPA RISC-V cache

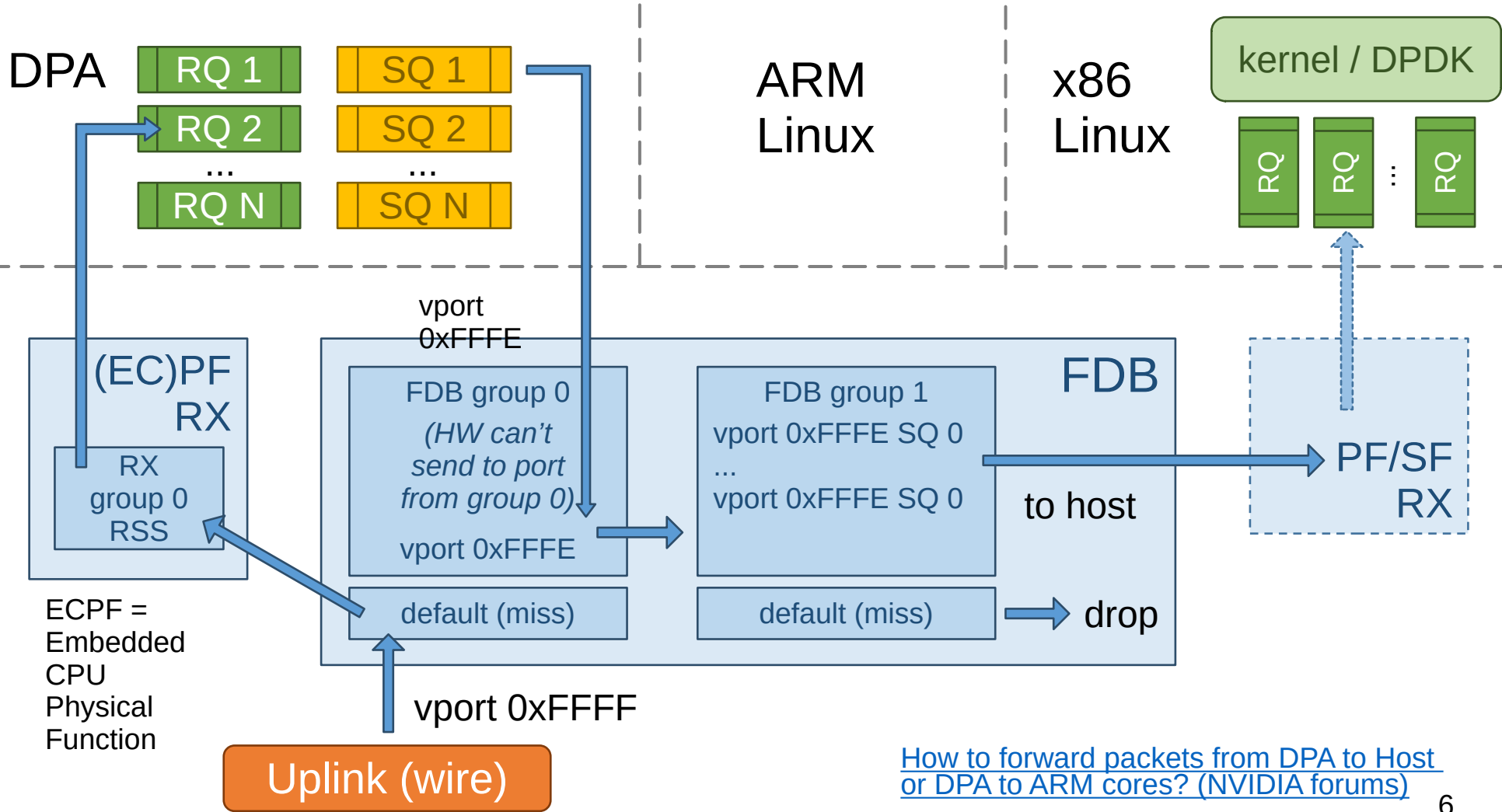
Resources	DPU (ARM)	DPA (RISC-V)
Processor	Cortex-A78AE	RV64IMAC
Cores*	16	256 (190 usable)
L1D\$	64K×16	1K×256
L1I\$	64K×16	1K×16
L2\$	0.5M×16	1.5M×1
L3\$	16M×1	3M×1
Frequency	2.133GHz	1.8GHz
RAM	16 GB	1 GB of 16 GB

[\[Demystifying Datapath Accelerator Enhanced Off-path SmartNIC\]](#)

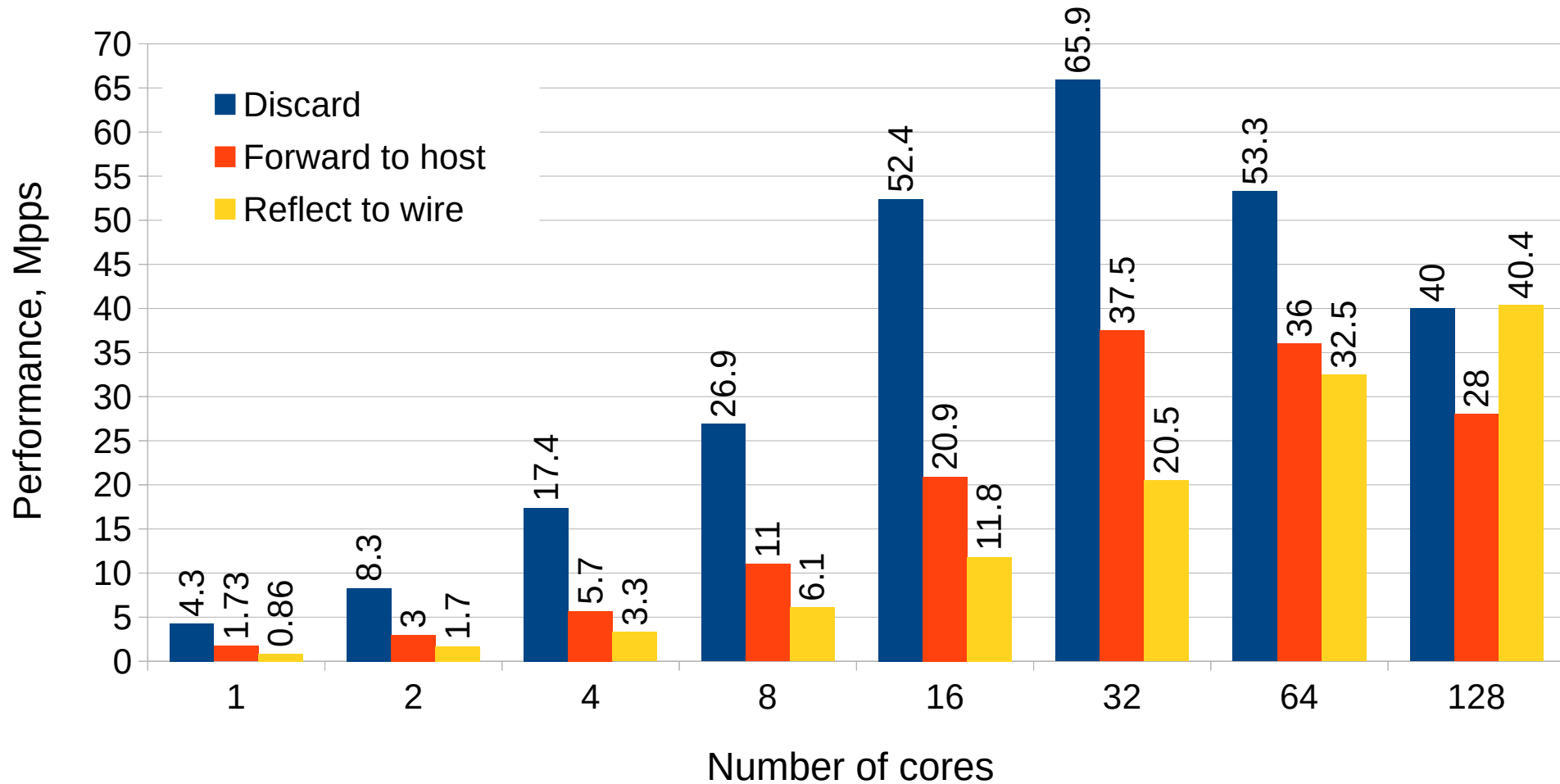


Technologies

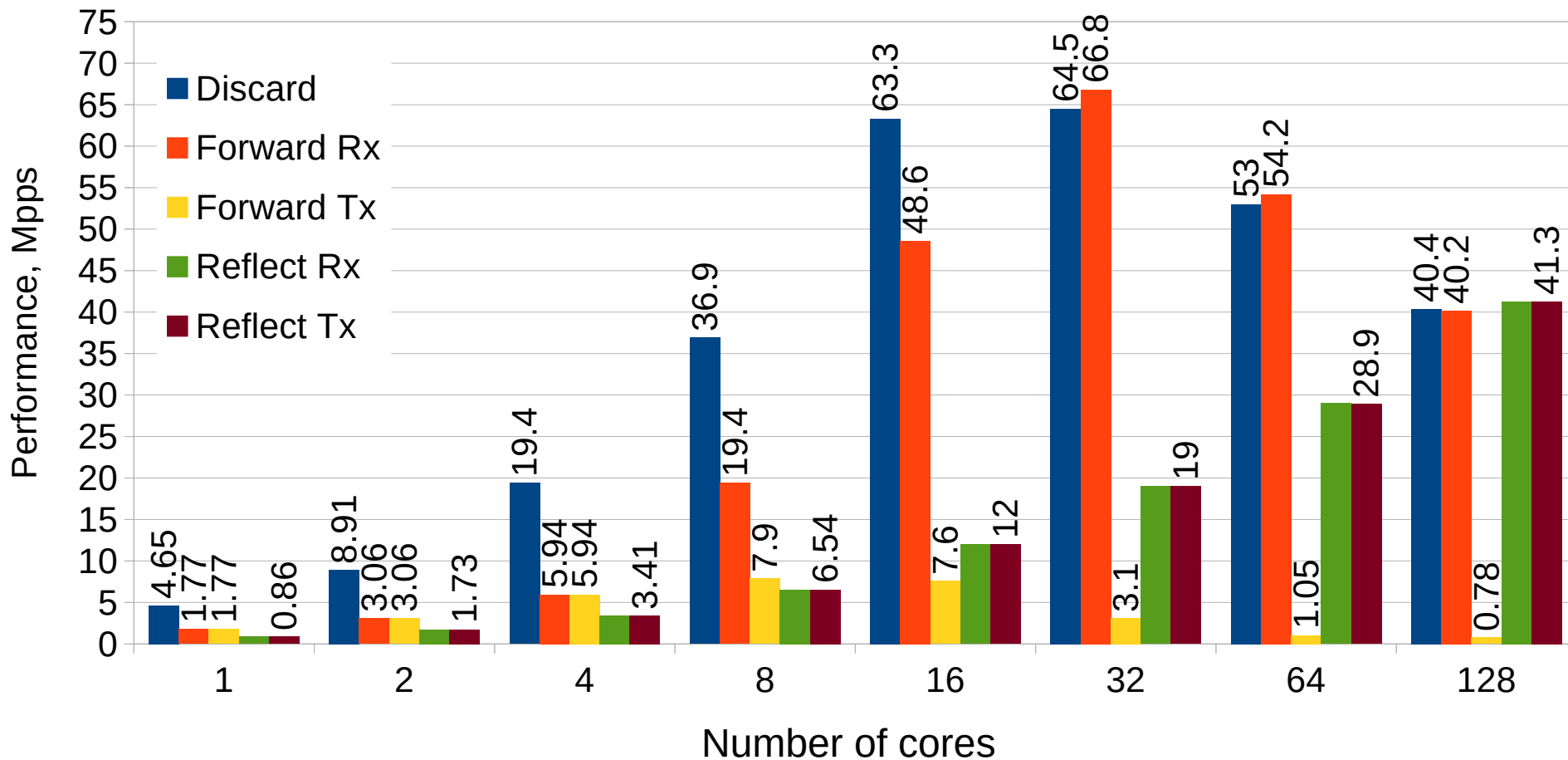
- Why not DOCA (Datacenter-on-a-Chip Architecture)?
 - Closed source!
 - only RDMA for DPA currently
- DPACC: a Clang fork producing ELF with DPA program and harness.
- FlexIO
 - Setup NIC objects for DPA (RxQ, TxQ, memory buffers)
 - Load DPA programs to the NIC
 - Communicate with DPA programs
- DPDK
 - Steering using `rte_flow` and `rte_pmd_mlx5_rx/tx_queue_setup()`.
 - **Dataplane code for DPA programs.**



Zero packet loss rates @ 64 Bpp



Rates @ 74.4 Mpps, 64 Bpp



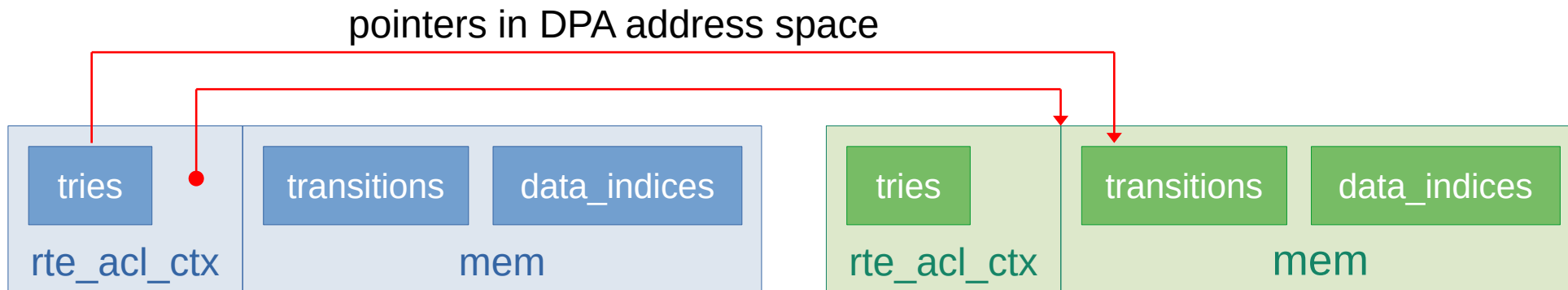
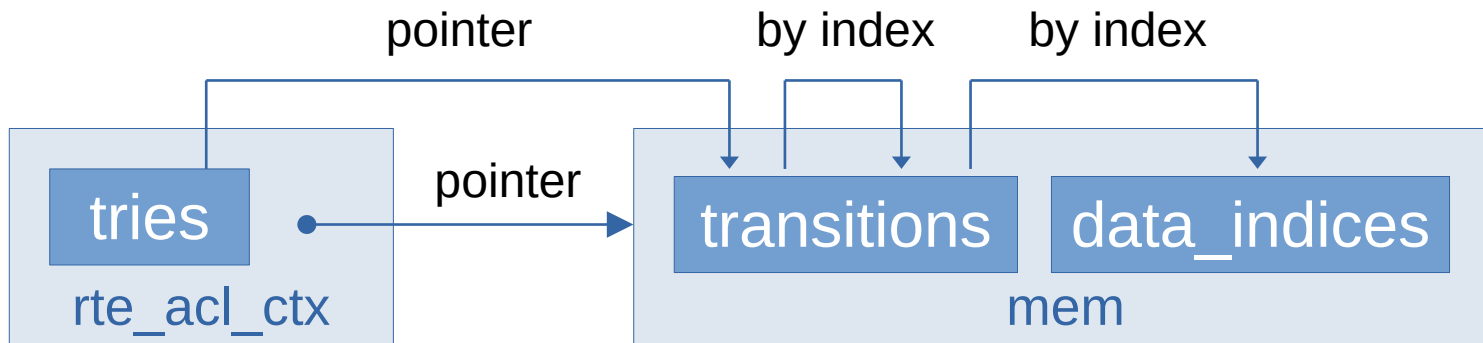
ACL library (recap)

- Classify packets by fields with prioritized rules.
 - Example: Access Control List
 - Example: select a policy to process traffic
- Build phase (control path): create search tries from rules.
 - CPU intensive, $O(\text{number of rules}^2)$, may take seconds
 - Memory: temporary — GB, resulting tries — KB to MB
- Run phase (fast path): traverse tries to match packets.

Build ACL on host, run on DPA

- Like Hyperscan does (regex library)
- Use cases beyond DPA:
 - Performance: distributed data plane
 - Viability: data plane on a SoC
 - Industrial IoT applications
 - Security: move complex build code from dataplane process.
 - Timeout and cancellation become trivial.
- Address space translation is sufficient for DPA.
- Universal solution would require some format discipline.

ACL address space translation



Temporary non-functional ACL

Functional ACL copy in DPA memory

Address space translation API

```
int
rte_acl_translate(const struct rte_acl_ctx *ctx,
                 uint64_t base_addr,
                 void *out, size_t *out_size);

ctx = rte_acl_create(...);
rte_acl_add_rule(ctx, ...);
rte_acl_build(ctx);

rte_acl_translate(ctx, 0, NULL, &size);

base_addr = /* allocate size bytes in device memory */
buffer = calloc(1, size);

rte_acl_translate(ctx, base_addr, buffer, &size);

/* copy size bytes from buffer to device memory at base_addr */
```

Testing ACL performance

- Number of rules:
 - Search complexity
 - Size in memory (vs cache size)
- Structure of rules:
 - 5-tuple
 - Each rule matches an equal fraction of traffic.
 - Match rate $\sim 1/8$
 - 1 category (no multiple ACL in one context)
- Priorities don't show much difference.
- DDoS-like traffic:
proto tcp src <random> dst <fixed> sport <random> dport <random>

Rules	Size
1	10 KB
16	18 KB
1024	441 KB
4096	2 MB
8192	4 MB
16384	7 MB
32768	14 MB
65536	27 MB

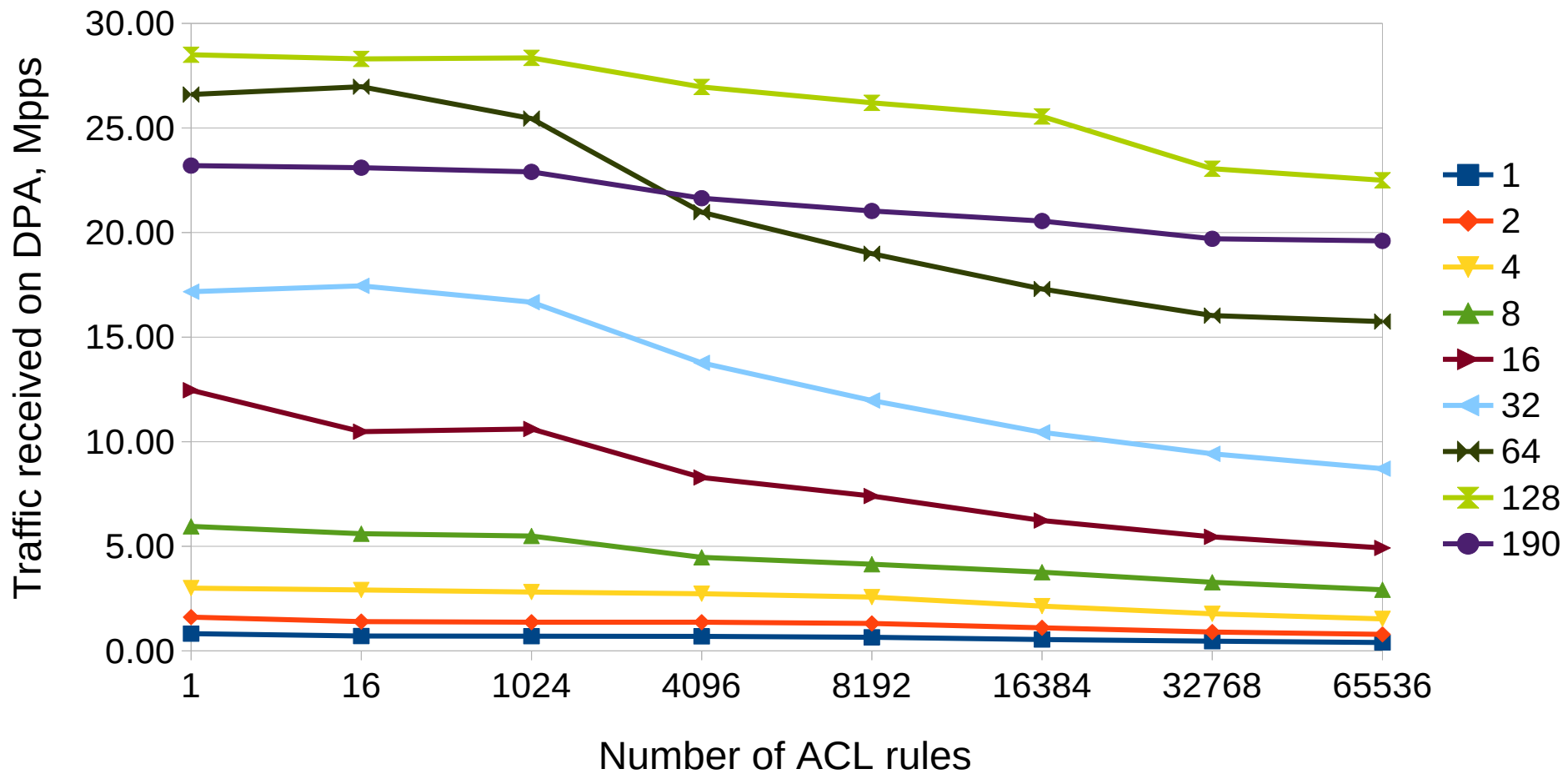
Tuning ACL scalar runtime

- **for each** packet:
 - **for each** of up to 8 tries:
 - **while** not matched:
 - transition to the next trie node
- Inner **while** loop is unrolled to process `MAX_SEARCHES_SCALAR = 2` tries at a time.
 - Unrolling is manual in upstream, rewritten to a loop for testing.
- Unrolling is suboptimal for DPA.

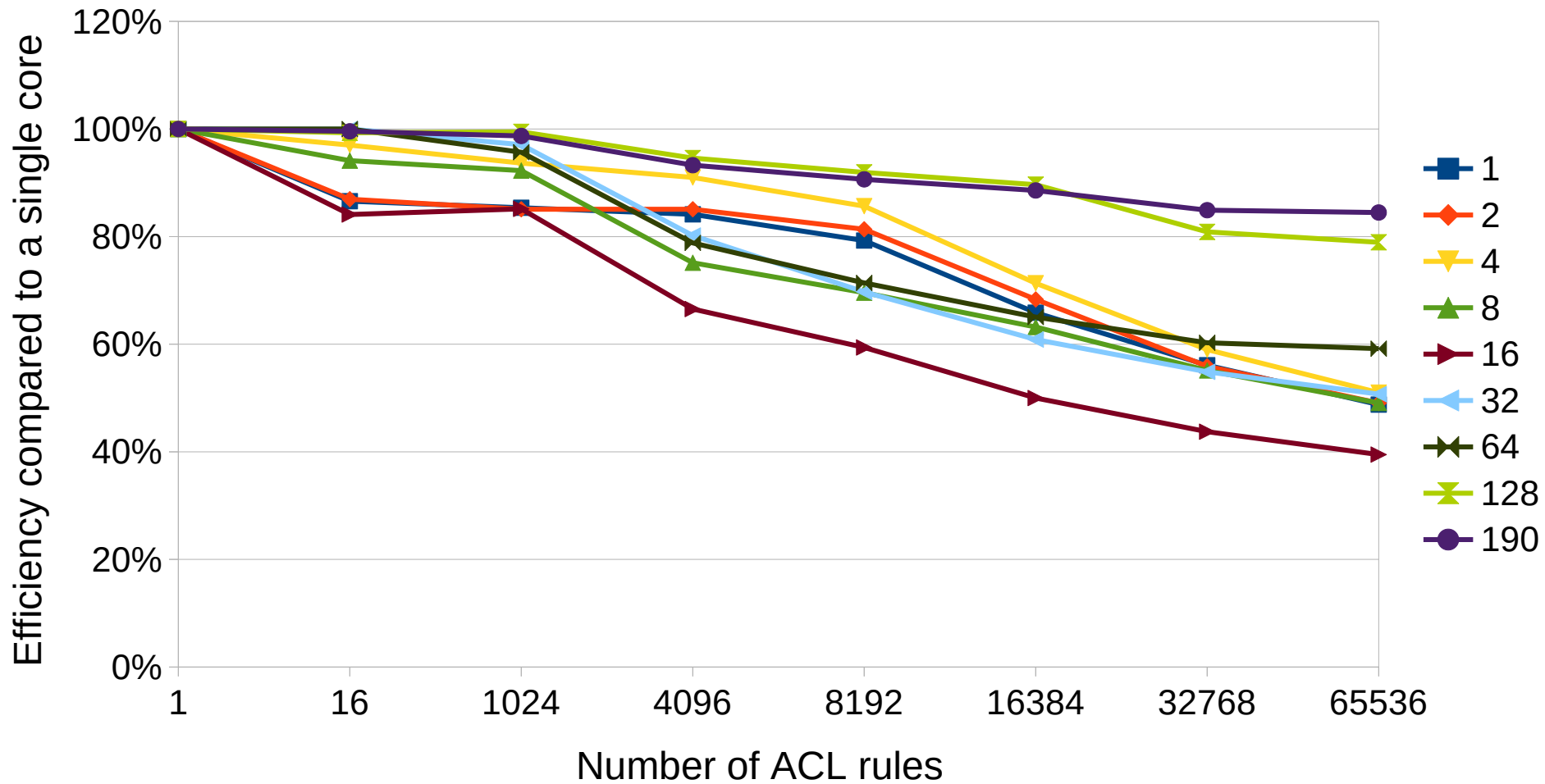
MAX_SEARCHES_SCALAR

Value	Mpps	Efficiency
1	0.700	109%
2	0.645	100%
4	0.510	78%
8	0.398	62%

ACL Performance



ACL scaling with DPA cores



BPF Library

- Load code at runtime with safety and security guarantees.
 - Verifier and JIT (control path)
 - Interpreter (fast path)
- Address space translation technique similar to ACL.
- Interpreter mode only:
 - (DPDK) No JIT for RISC-V.
 - (FlexIO) No way to add executable code to a DPA process.
- External symbols (a.k.a. BPF helpers) are in DPA address space.
 - DPA RPC to deliver addresses to the host.
 - Put helpers in ELF sections and parse ELF?
- Test program: SYN packet filter.

BPF interpreter performance and scaling

