CLOUDFLARE

Packet Metadata – Where are thee?

Jakub Sitnicki
Willem Ferguson
Tiago Lam
Cloudflare

Linux Plumbers Conference
December 2025
Tokyo, Japan

Agenda

0 Recap

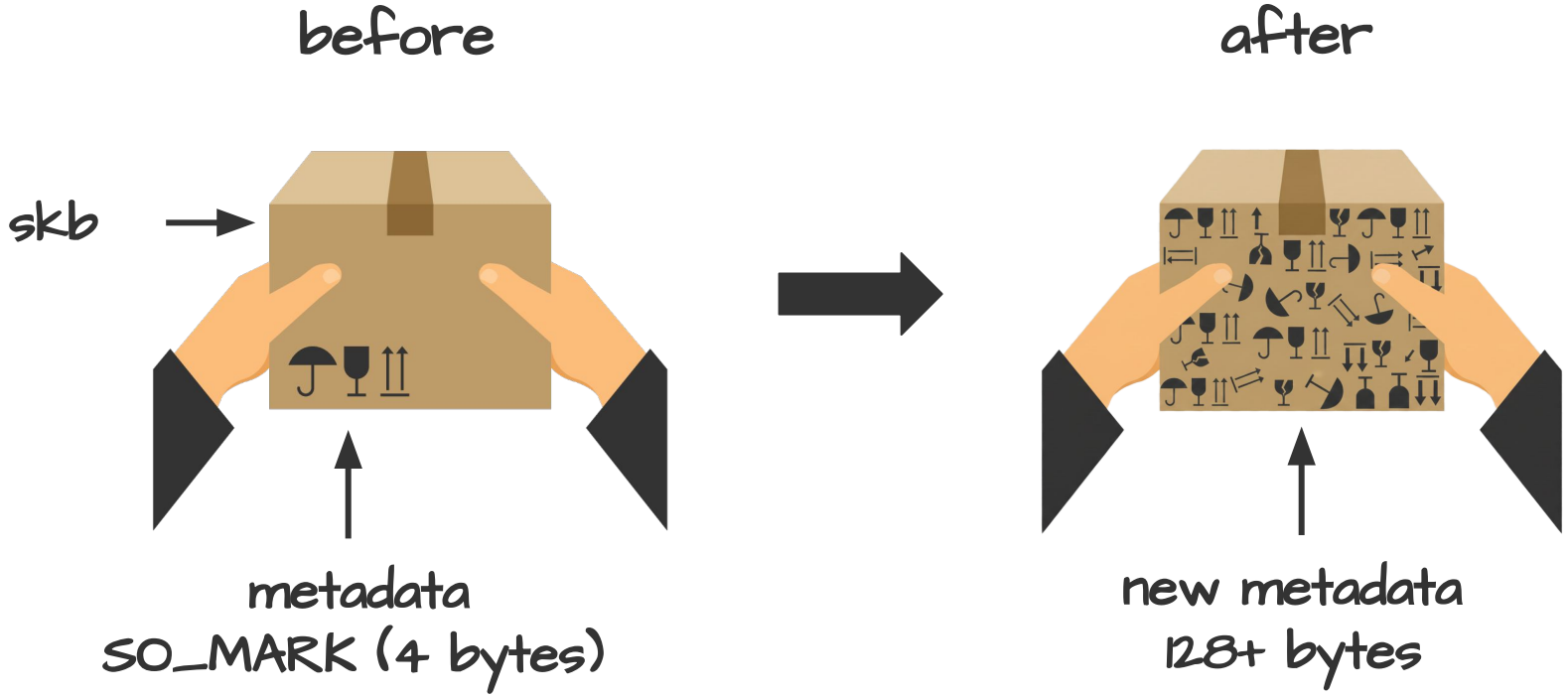
1 Project update

2 Lessons from production

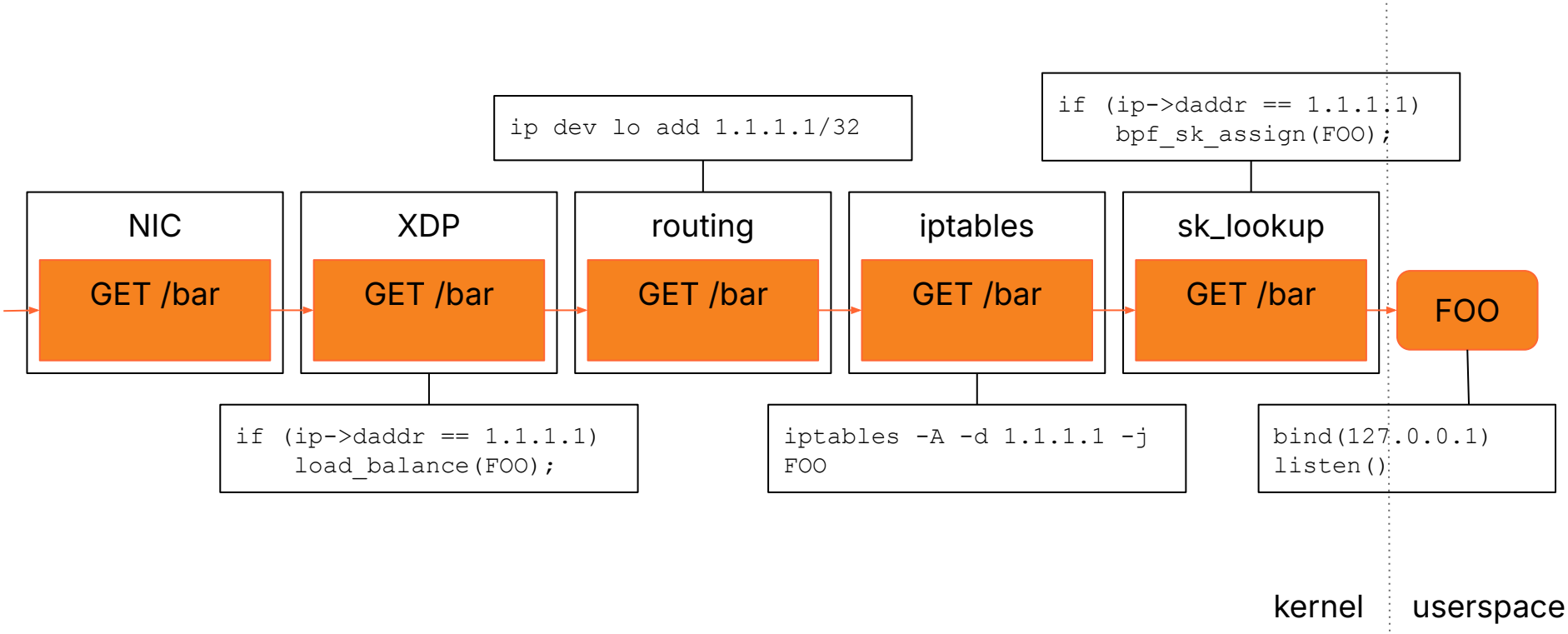
3 Open challenges

Motivation & use cases

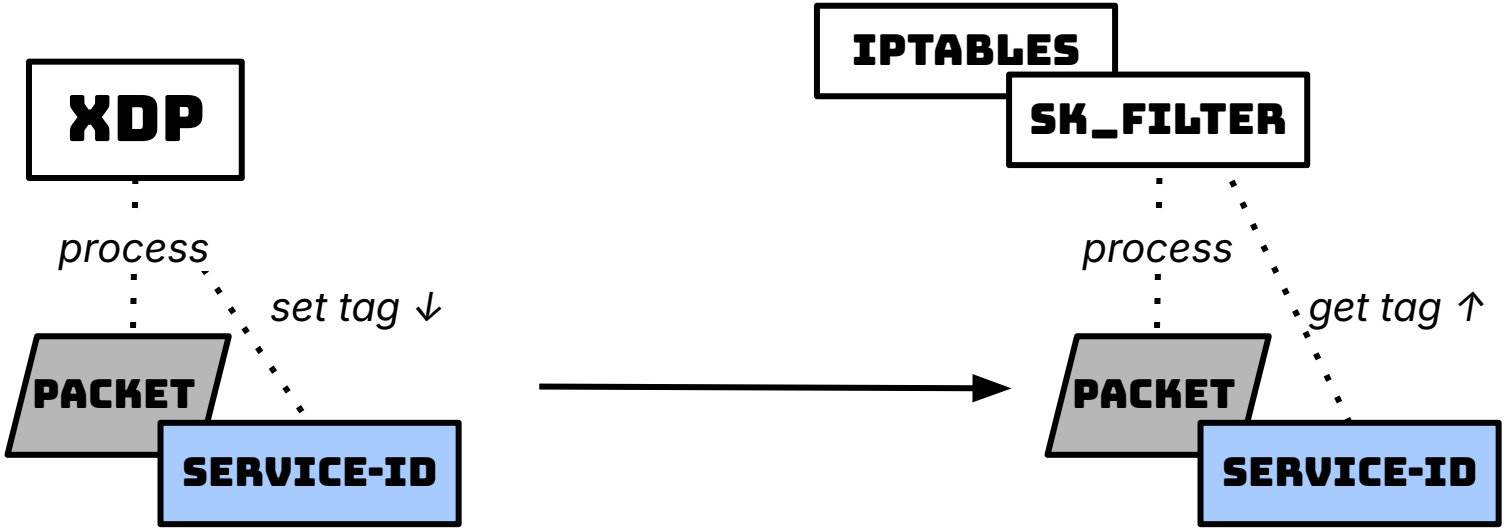
Motivation – Support tens of bytes of packet metadata



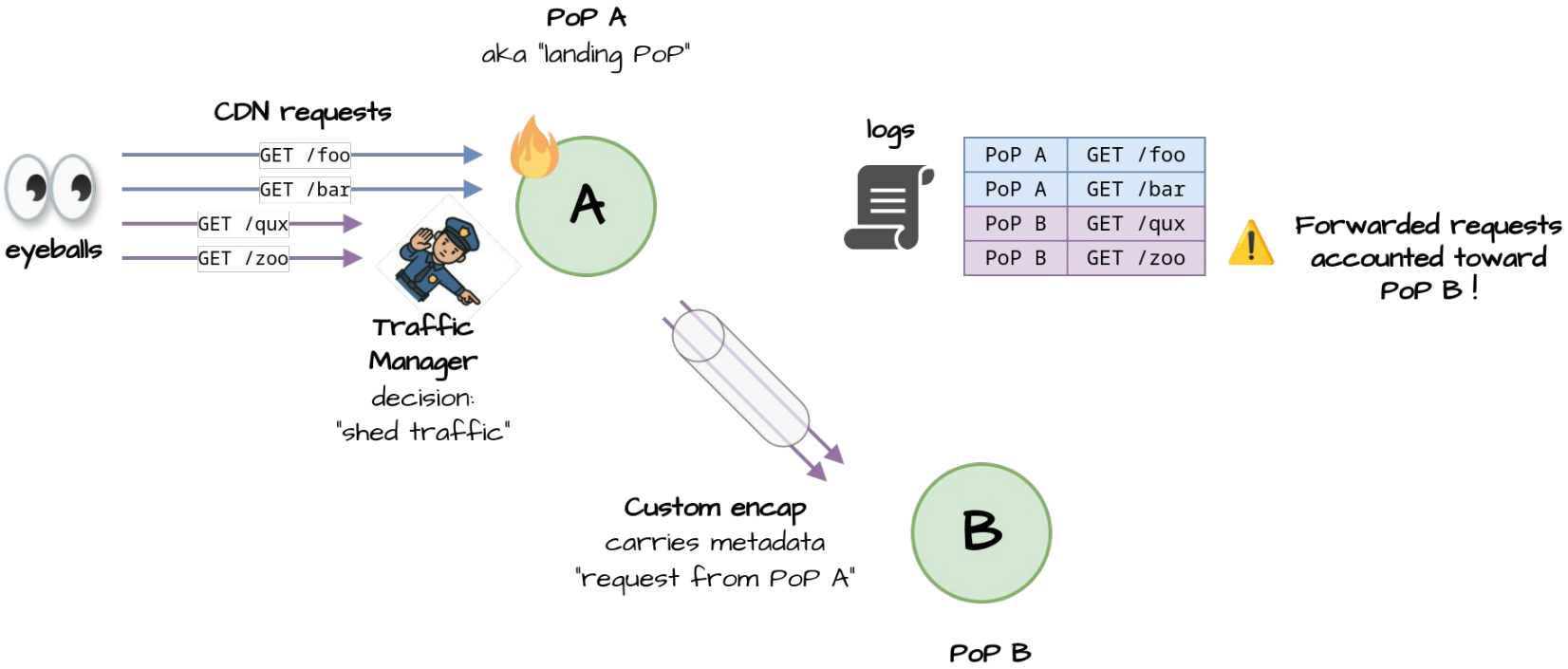
Motivation – Use case #1
Configuration everywhere – Inconsistency everywhere



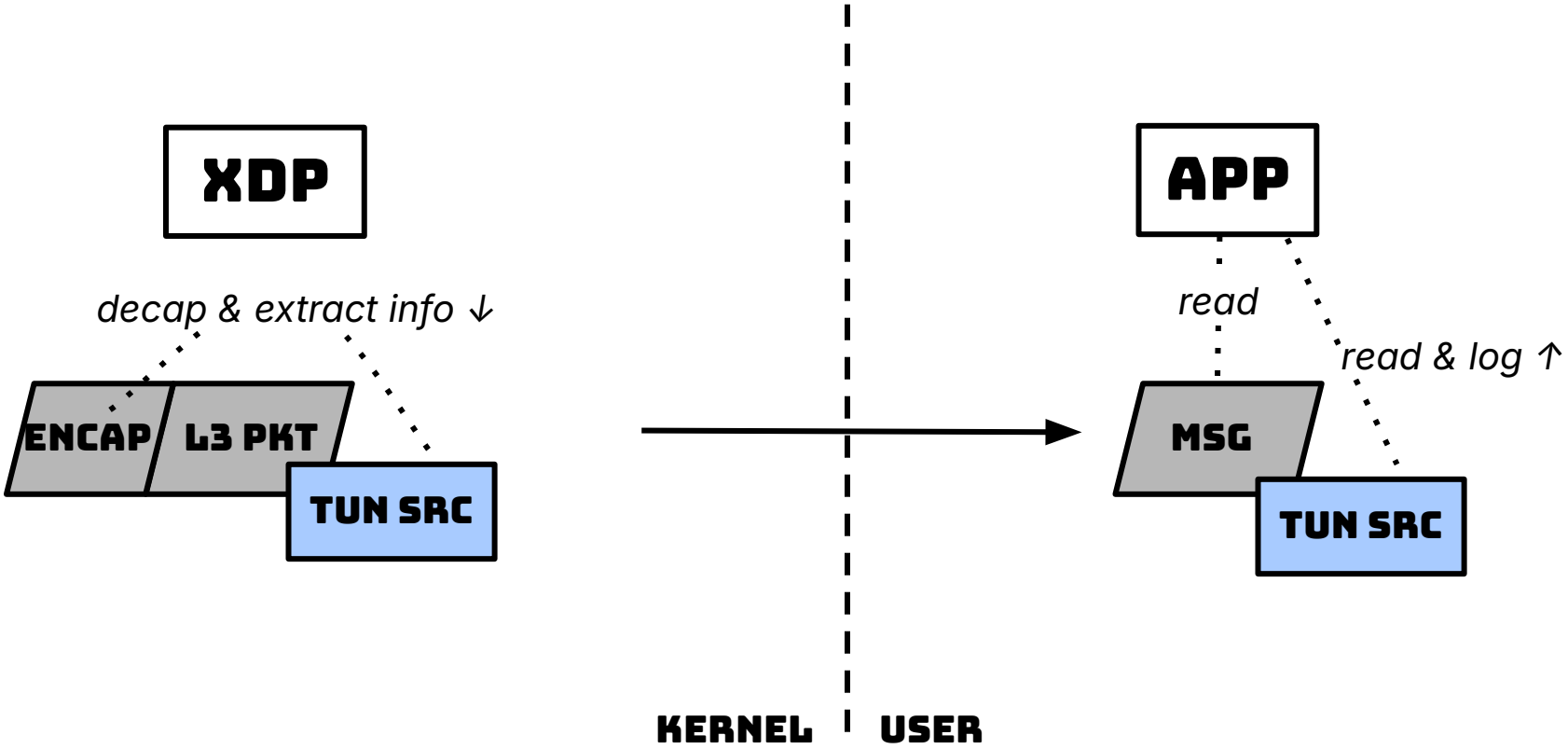
Motivation – Use case #1 – Classify packets in XDP



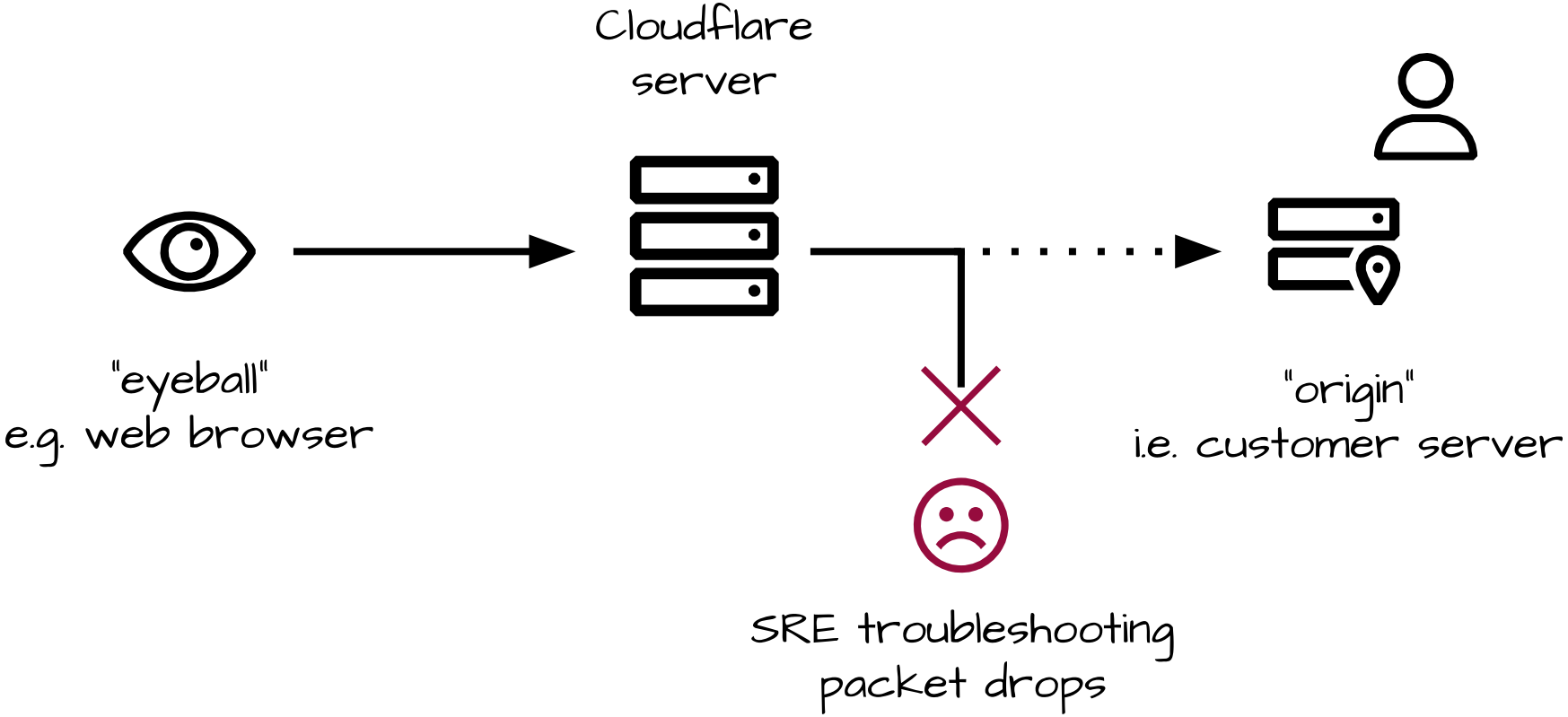
Motivation – Use case #2
Where did this request come from?



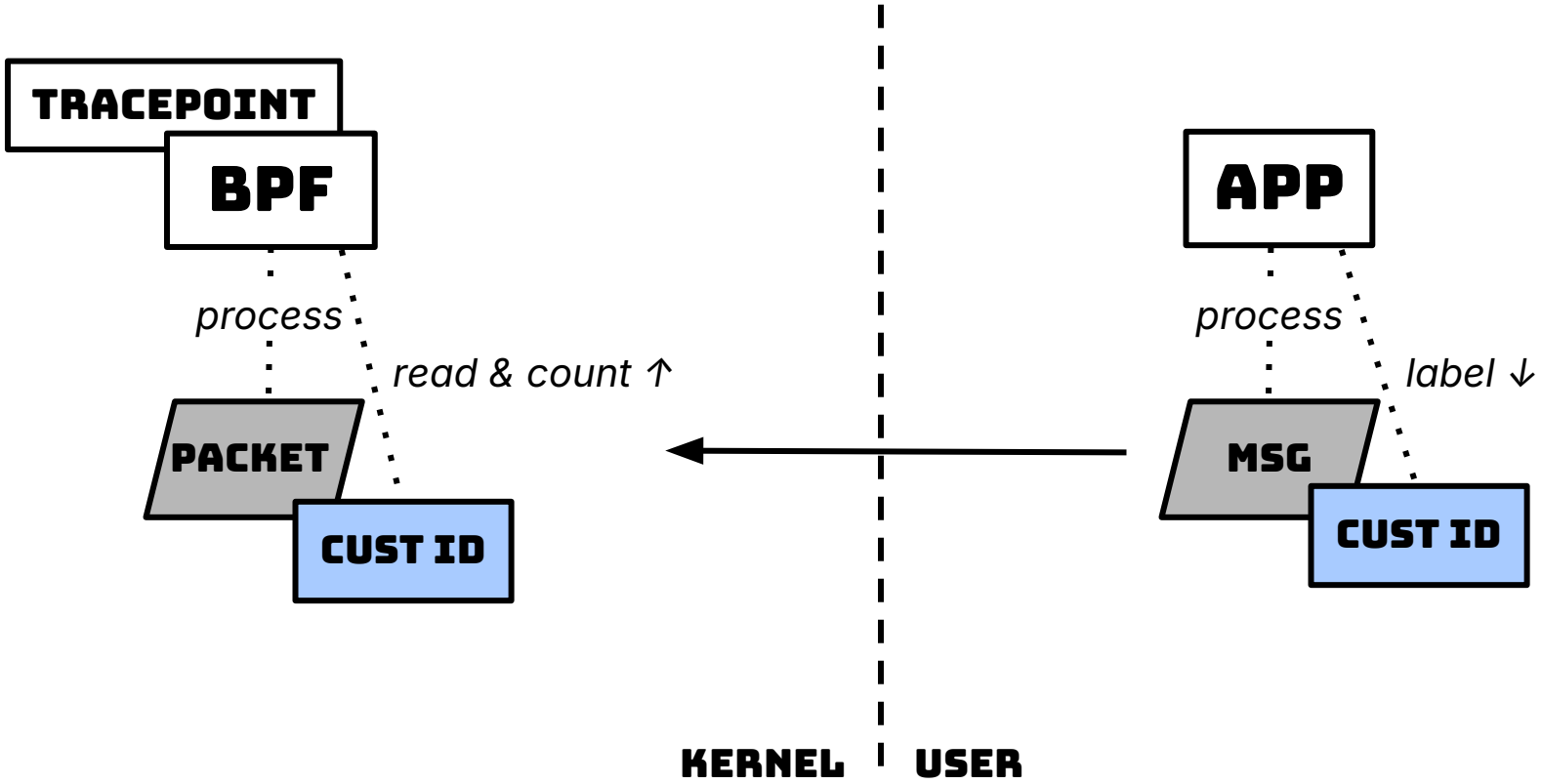
Motivation – Use case #2 – Pass info from encap headers to user-space



Motivation – Use case #3 – Attribute packet drops

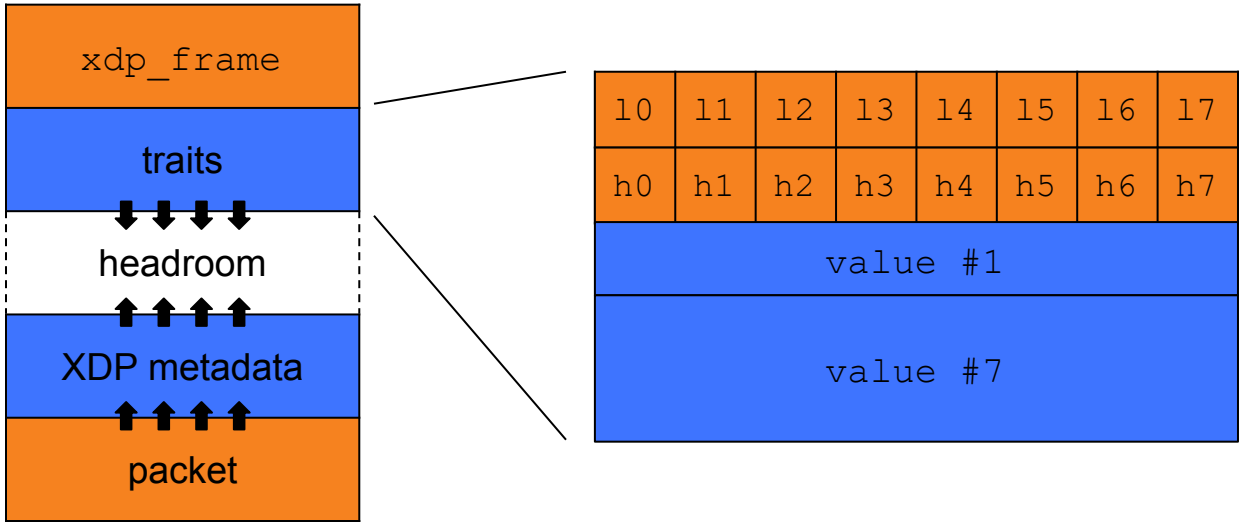


Motivation – Use case #3 – Attribute packet drops



Last time at Netdev...

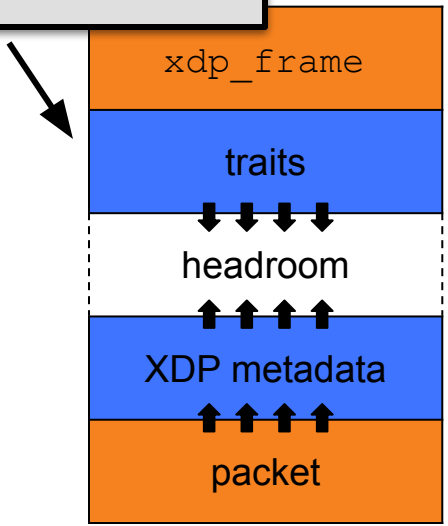
Traits - proposal from Netdev, March ‘25

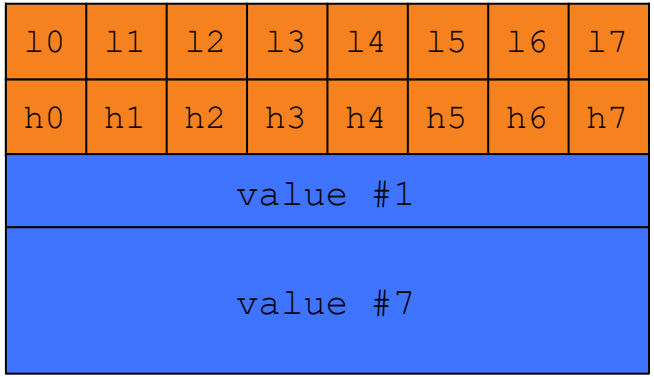


Traits - proposal from Netdev, March ‘25

CLOUDFLARE

new area designated for metadata in skb headroom

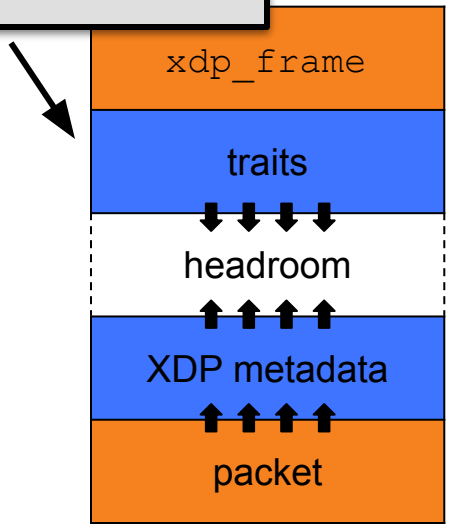


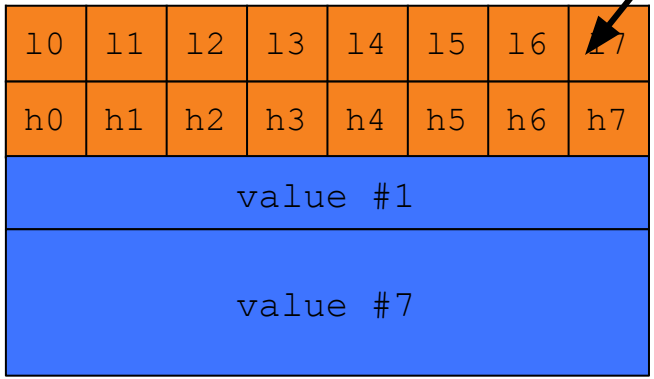


Traits - proposal from Netdev, March ‘25

new area designated for metadata in skb headroom

uses lean K/V storage format baked into the kernel

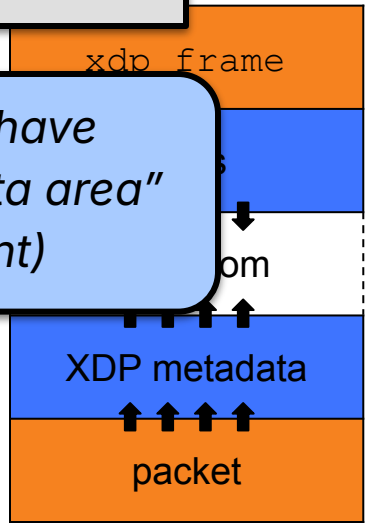




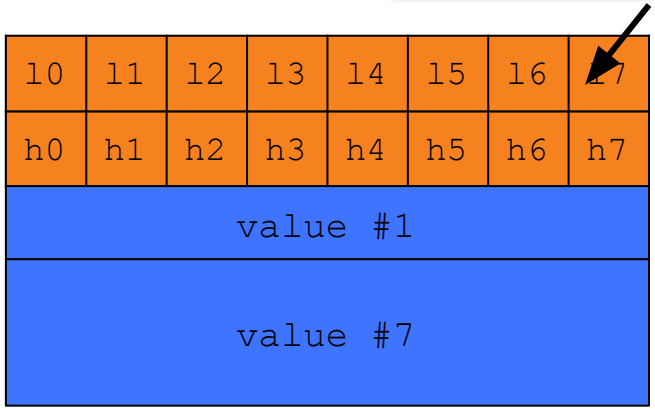
Traits - proposal from Netdev, March ‘25

new area designated for metadata in skb headroom

“we already have XDP metadata area” (it’s redundant)



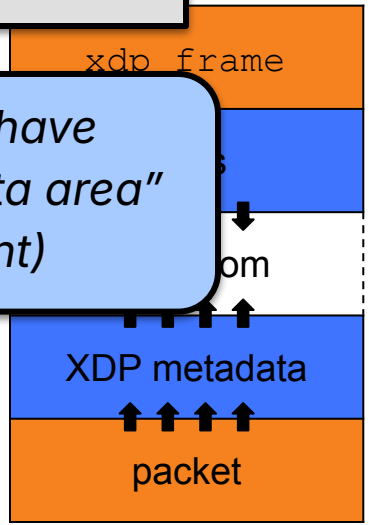
uses lean K/V storage format baked into the kernel

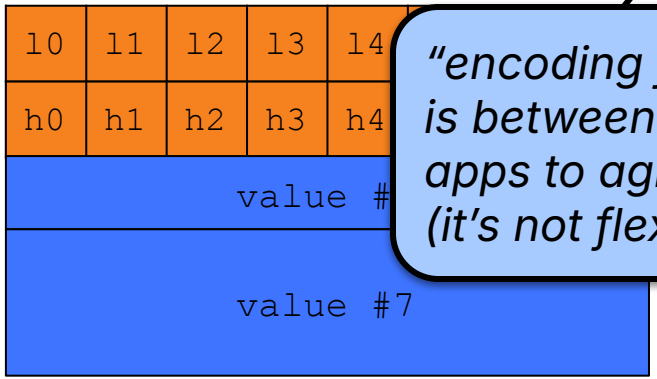


Traits - proposal from Netdev, March ‘25

new area designated for metadata in skb headroom

“we already have XDP metadata area” (it’s redundant)





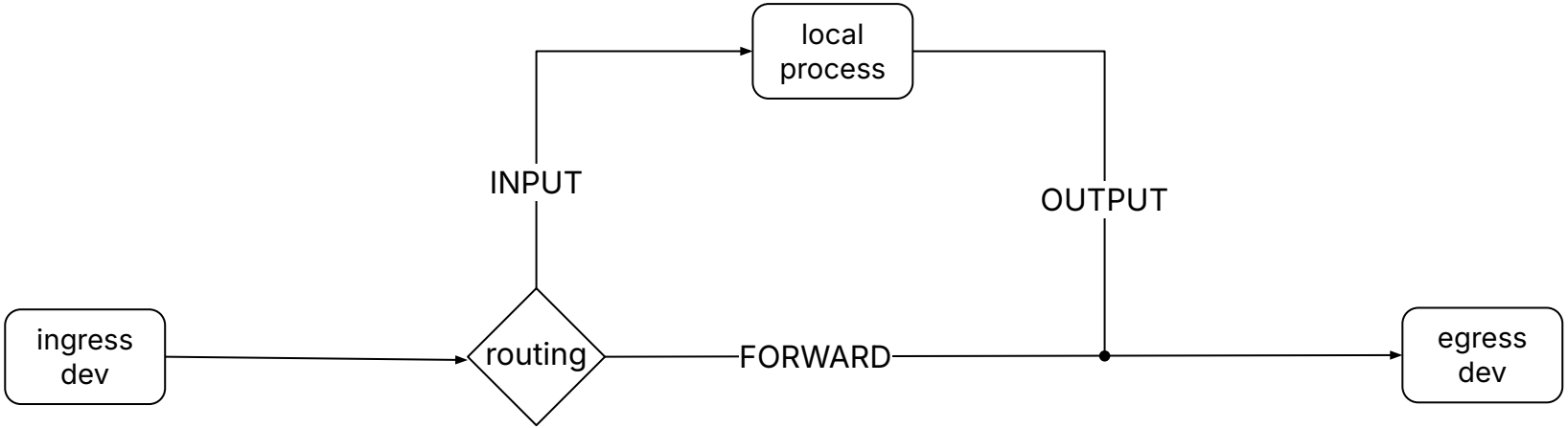
uses lean K/V storage format baked into the kernel

“encoding format is between BPF and apps to agree on” (it’s not flexible)

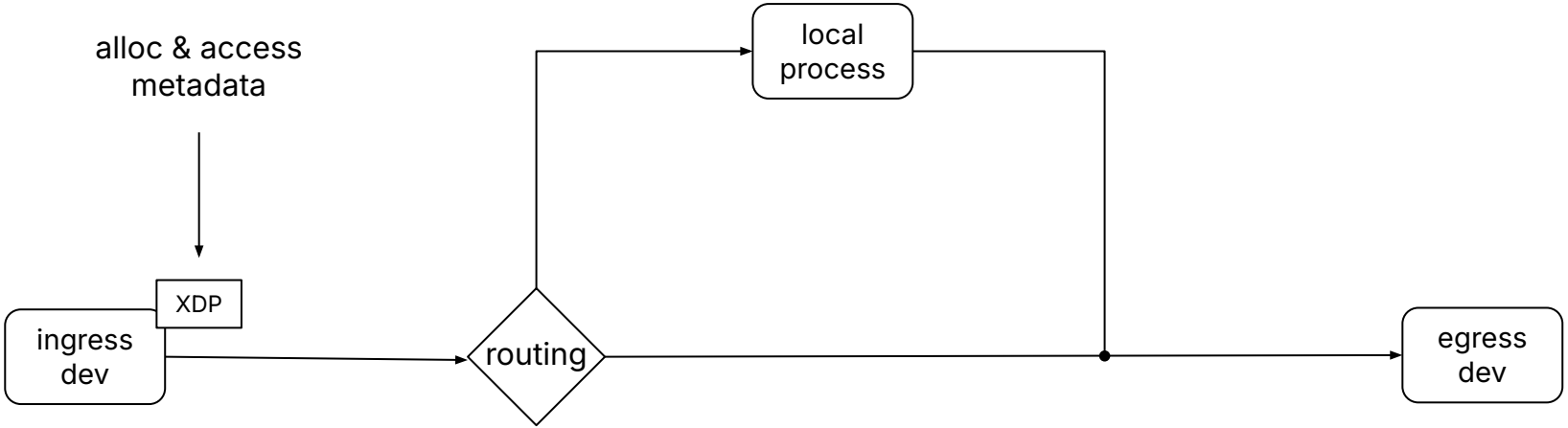

SKB TRAITS
NOWHERE
XDP METADATA

Can we make XDP metadata work?

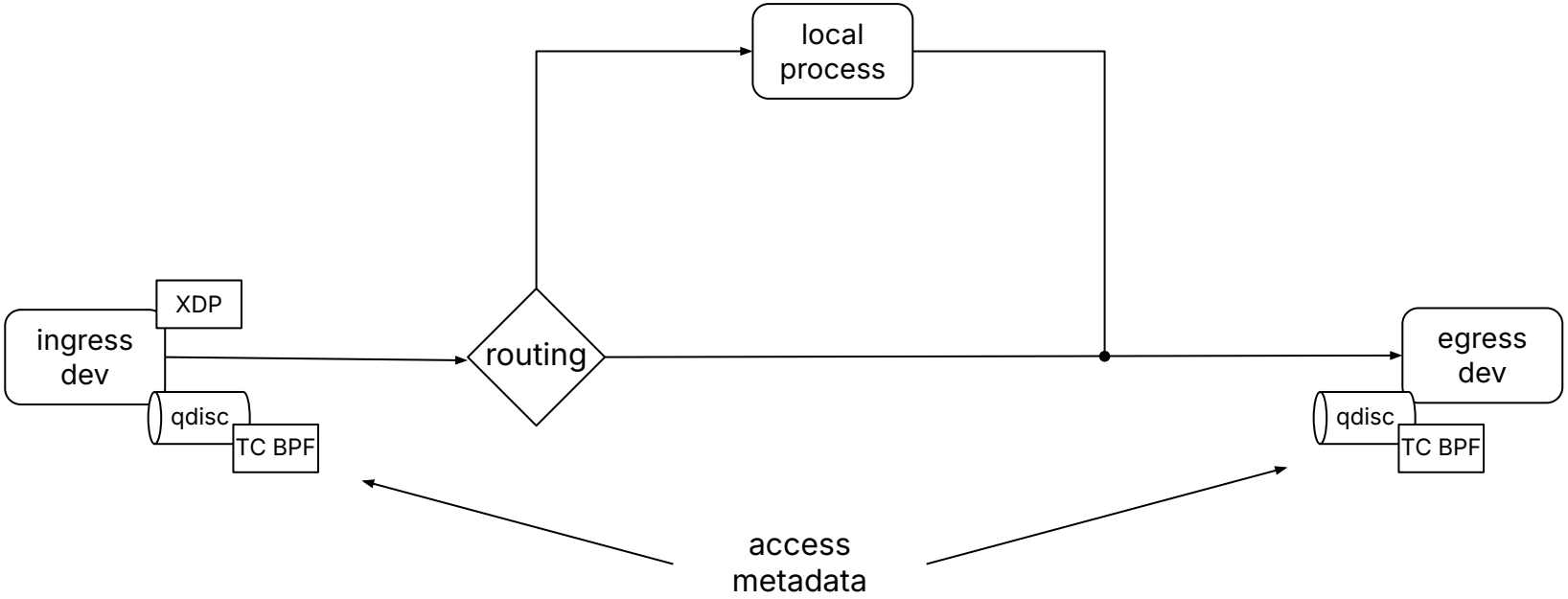
What can we do with XDP metadata today?



What can we do with XDP metadata today?

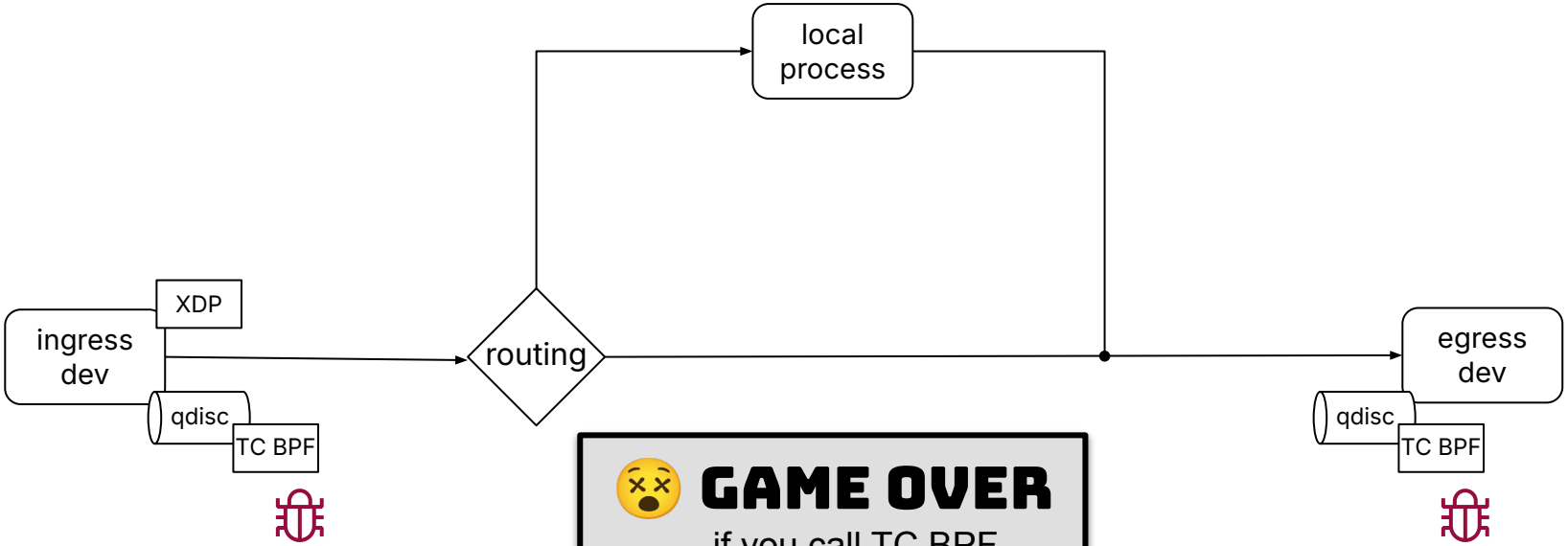


What can we do with XDP metadata today?



Terms & Conditions apply 🙈 🐵 🙊

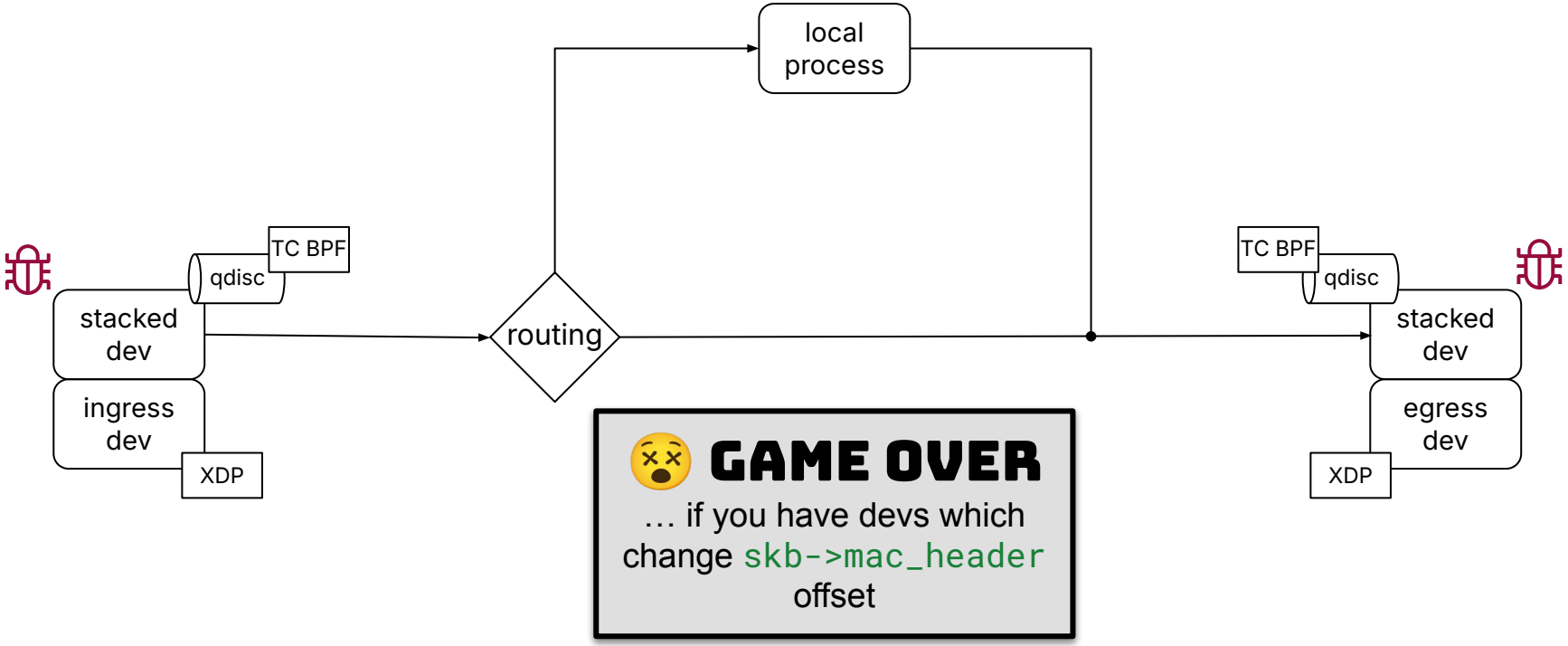
Some BPF helpers corrupt metadata



😵 **GAME OVER**

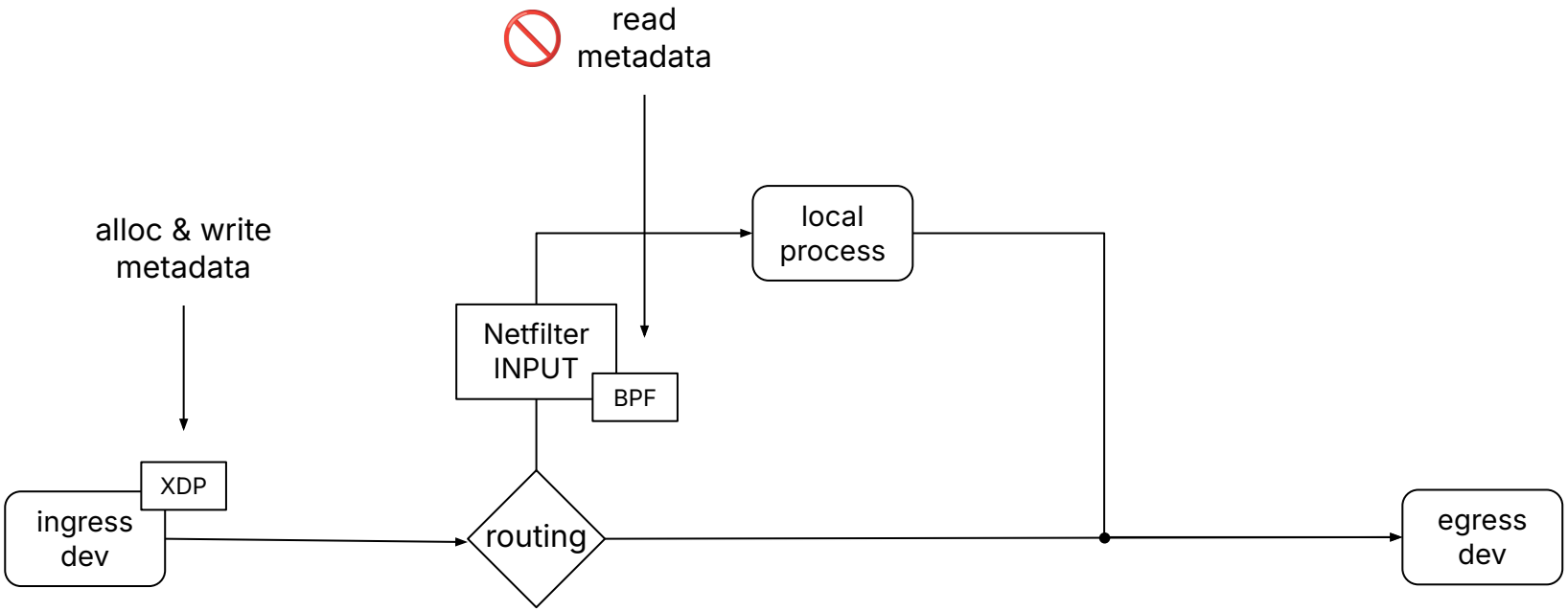
… if you call TC BPF `bpf_skb_*()` helpers that operate on `skb->head`

Stacked L2 (VLAN, GRE) devices corrupt metadata

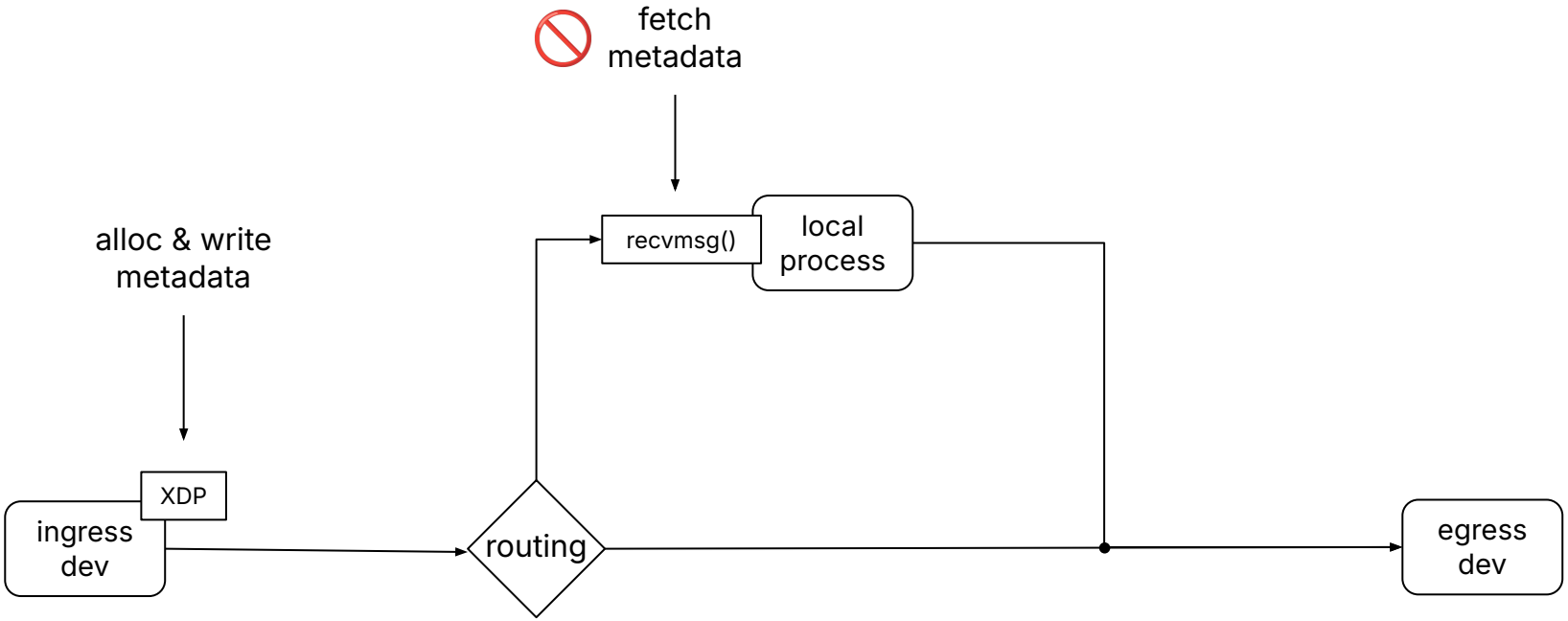


Some things are not possible today...

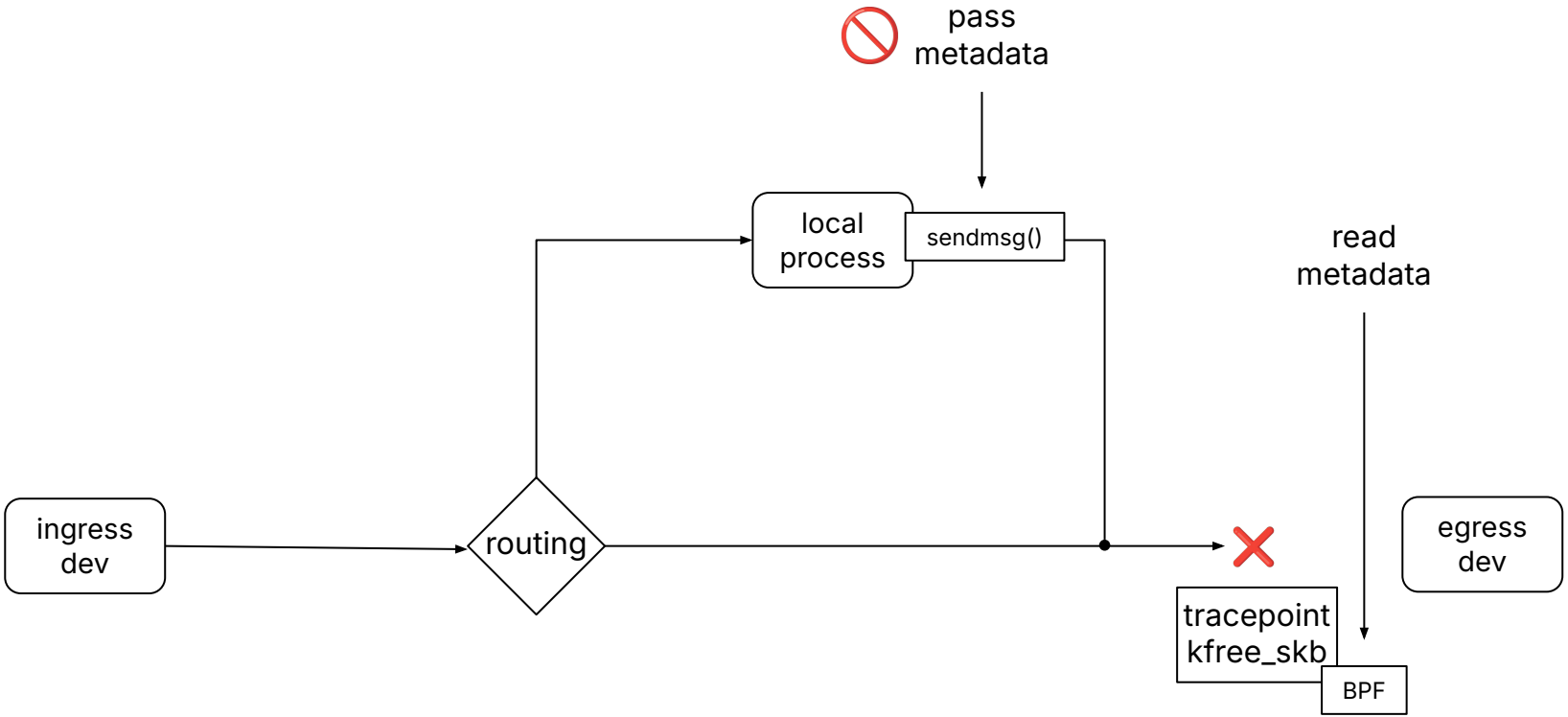
Access to metadata outside of XDP/TC



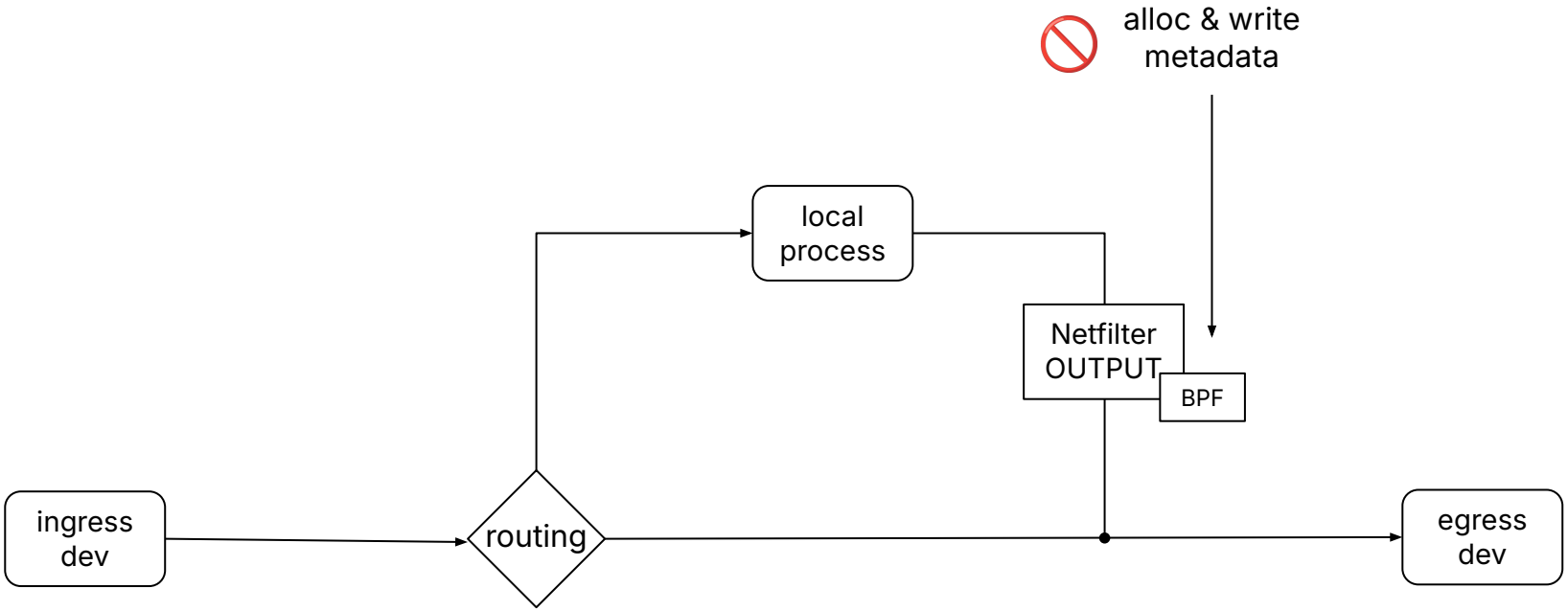
Consume metadata from user-space



Produce metadata from user-space



Allocate metadata from BPF other than XDP



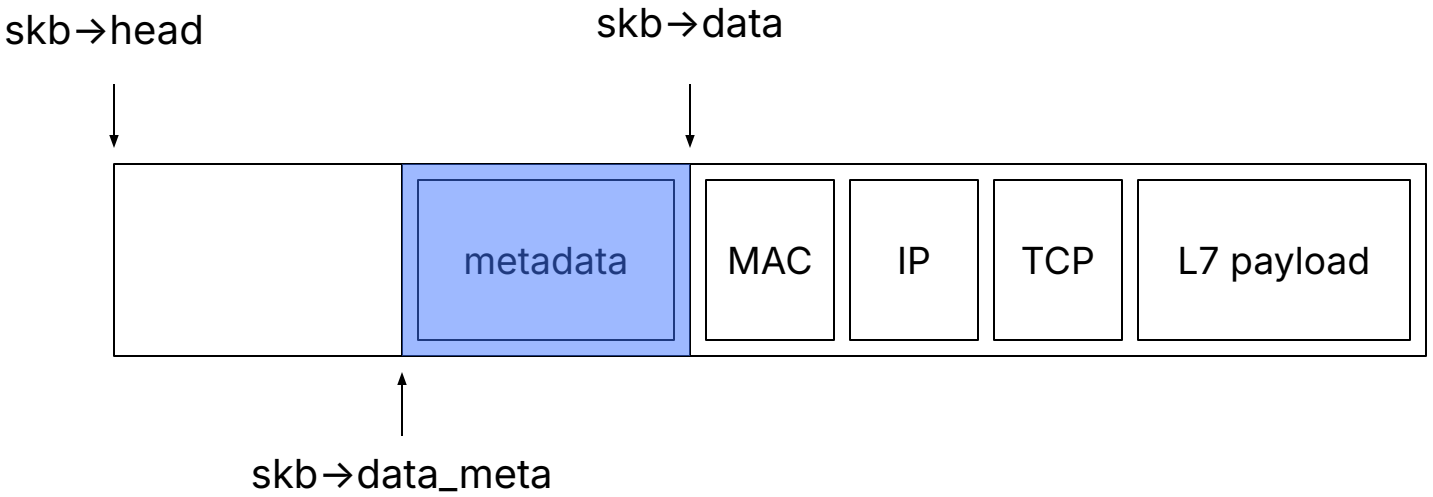
How do we fix XDP/skb metadata?

CLOUDFLARE

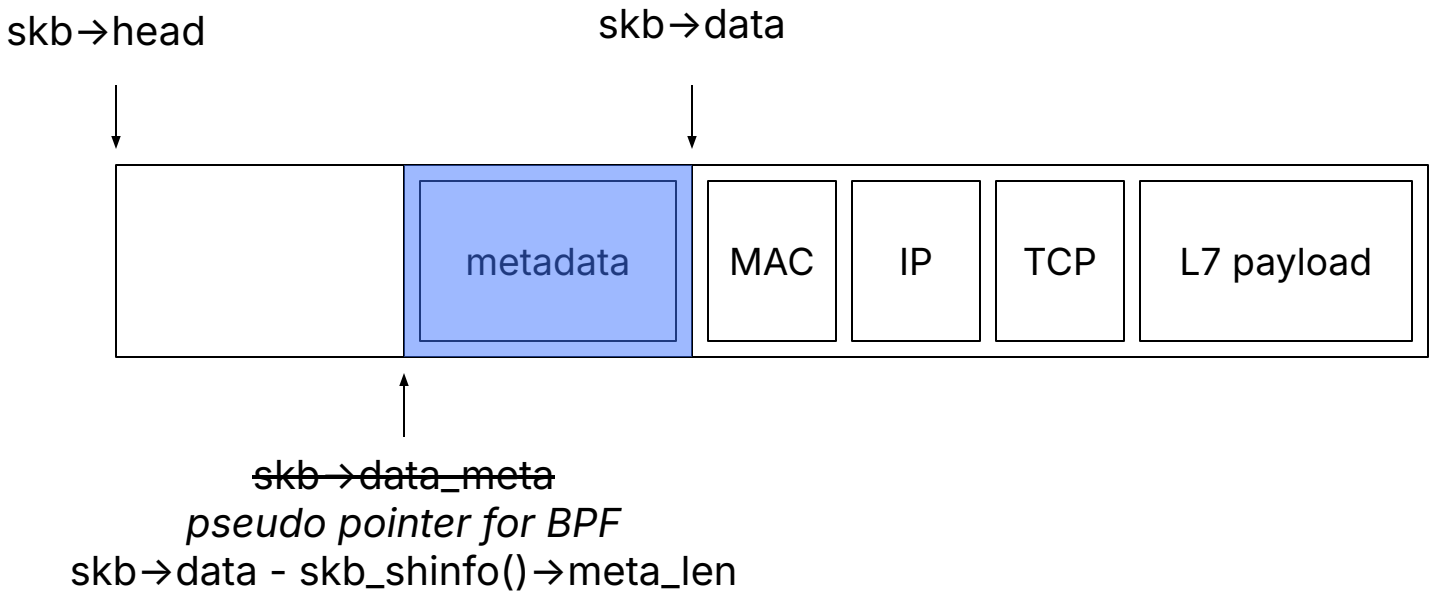
How do we fix XDP/skb metadata?*

* and stay backwards compatible

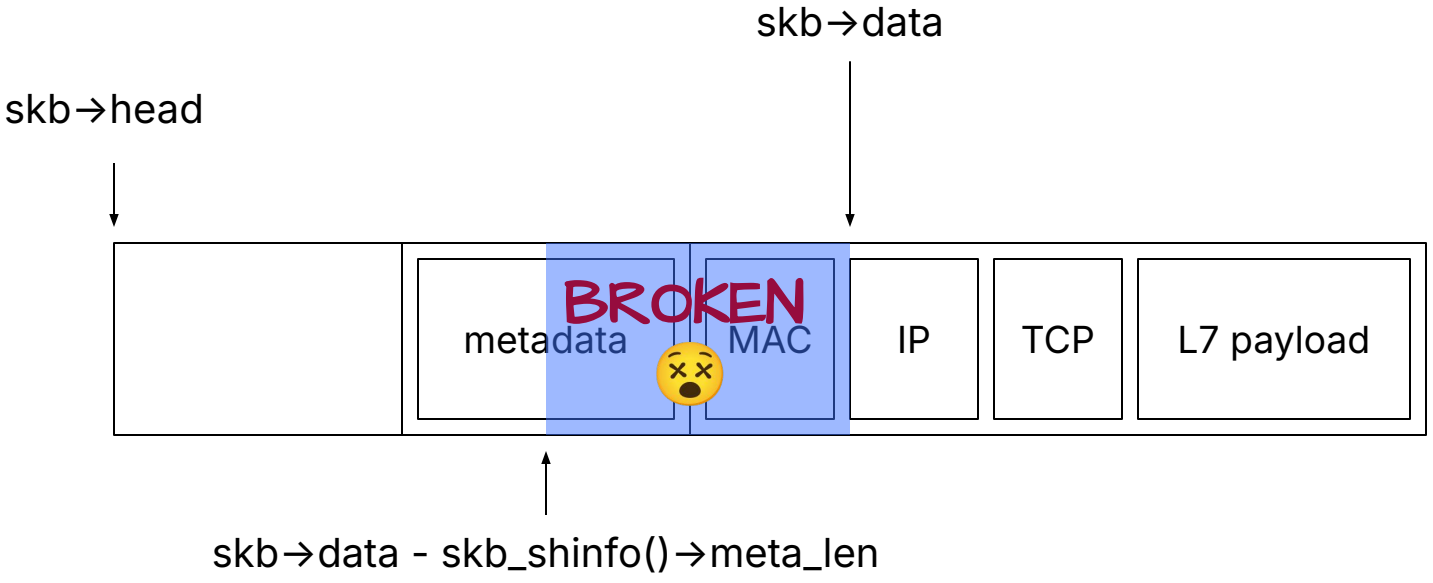
What do we need to make `skb->data_meta` work?



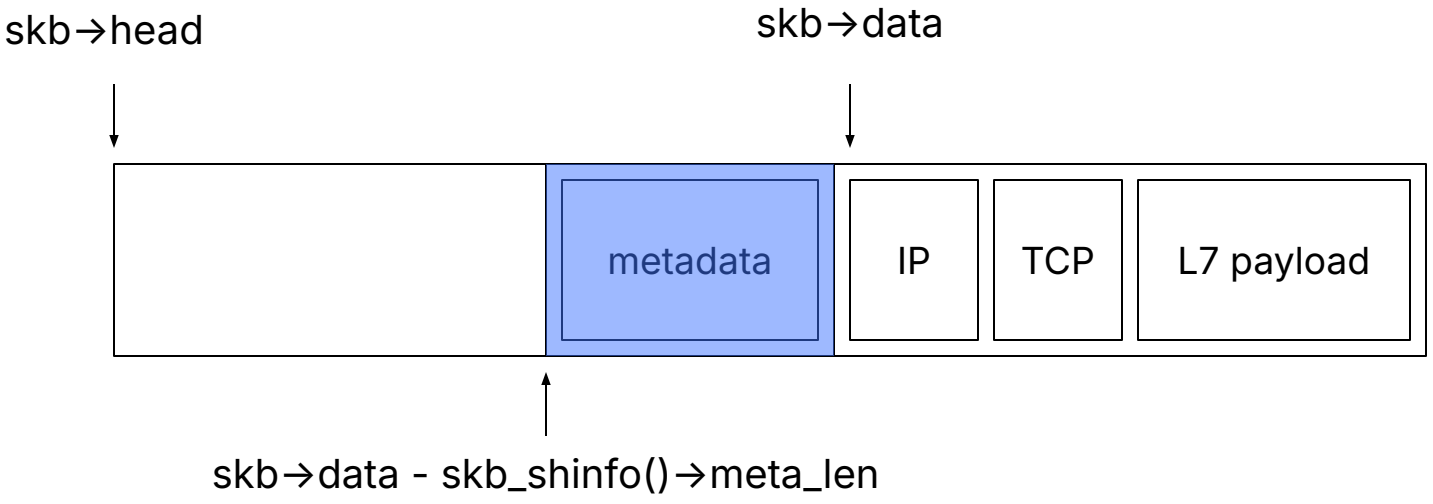
What do we need to make `skb->data_meta` work?



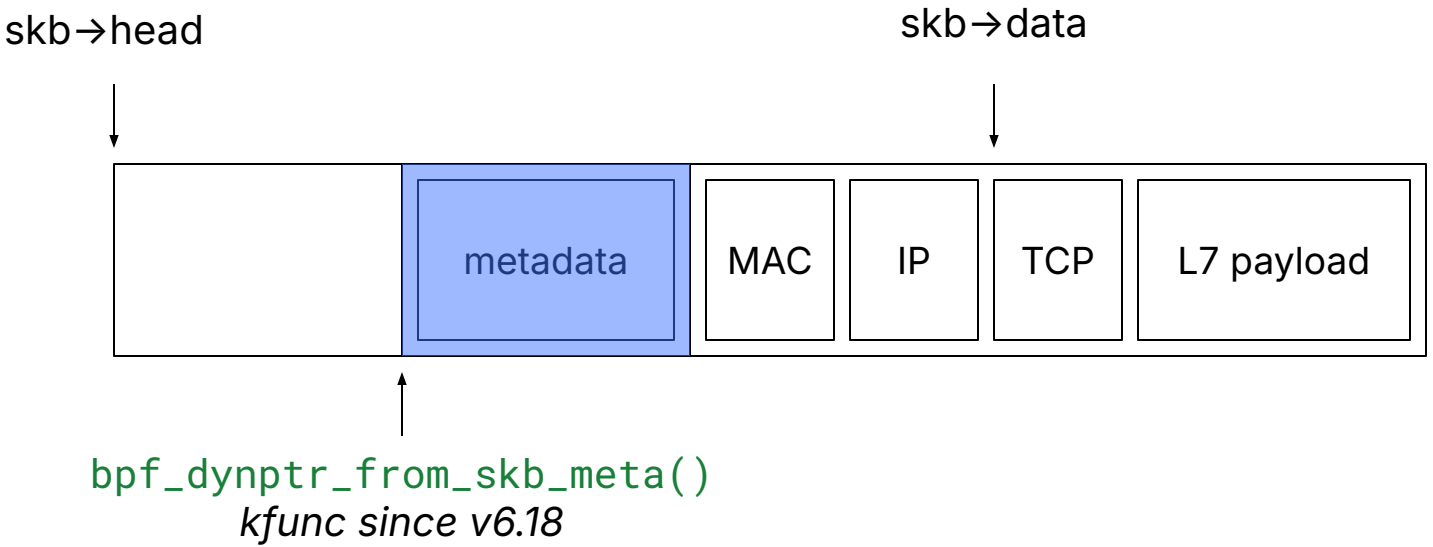
As we pull packet headers...



… we'd need to keep moving the metadata

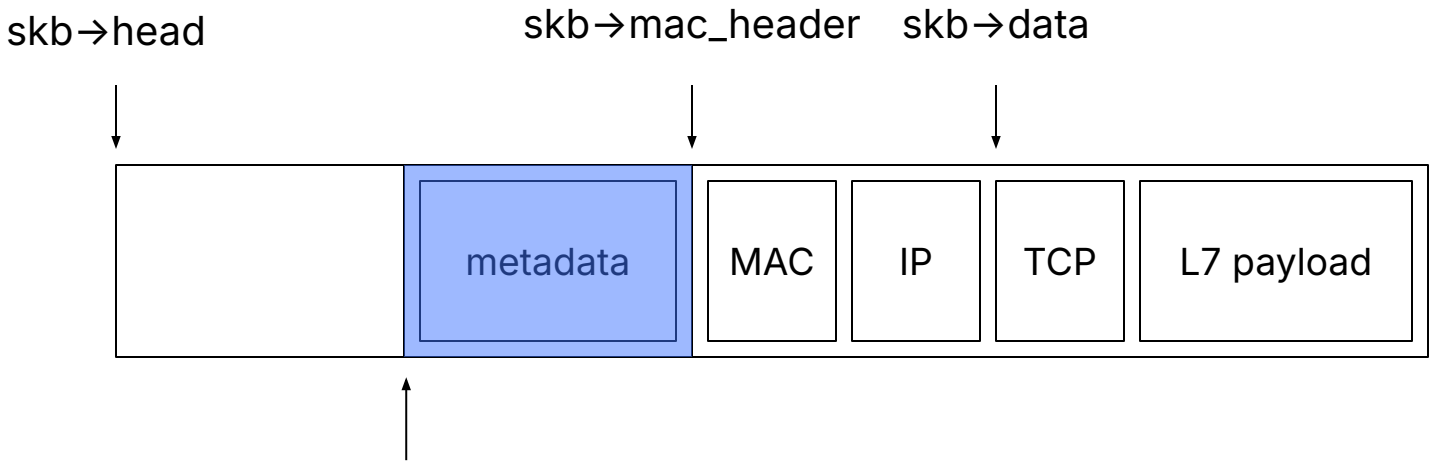


Alternative?
Another layer of indirection

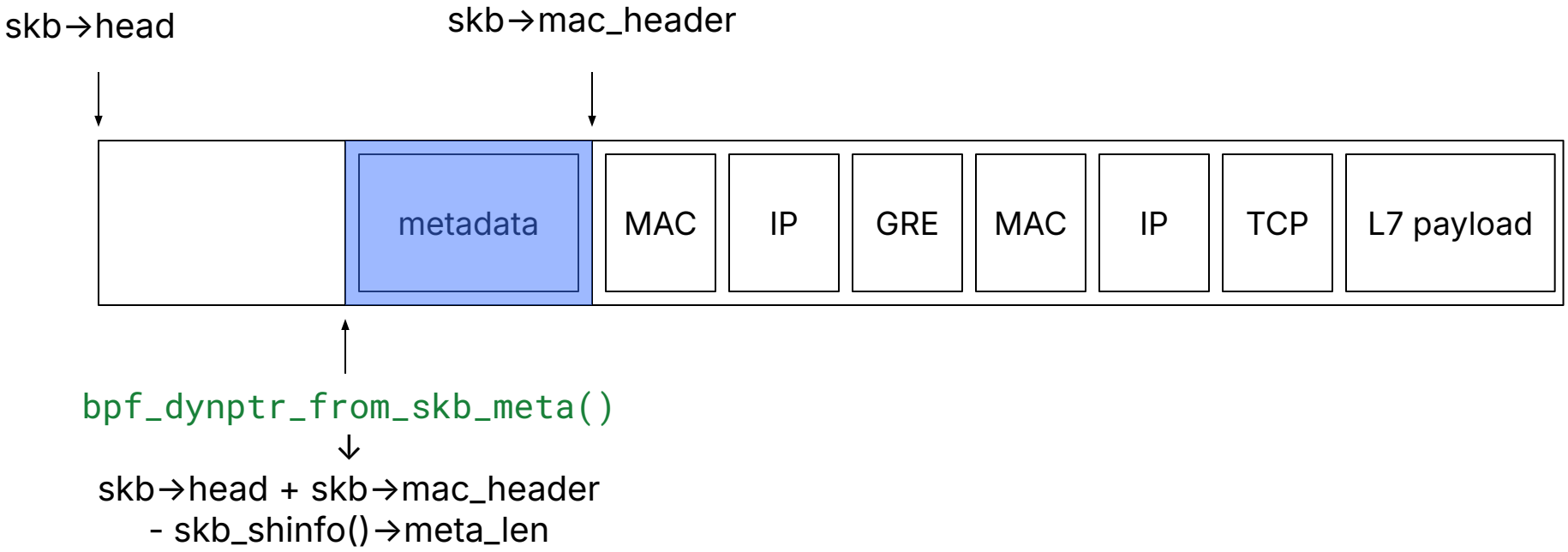


https://msgid.link/20250814-skb-metadata-thru-dynptr-v7-0-8a39e636e0fb@cloudflare.com/

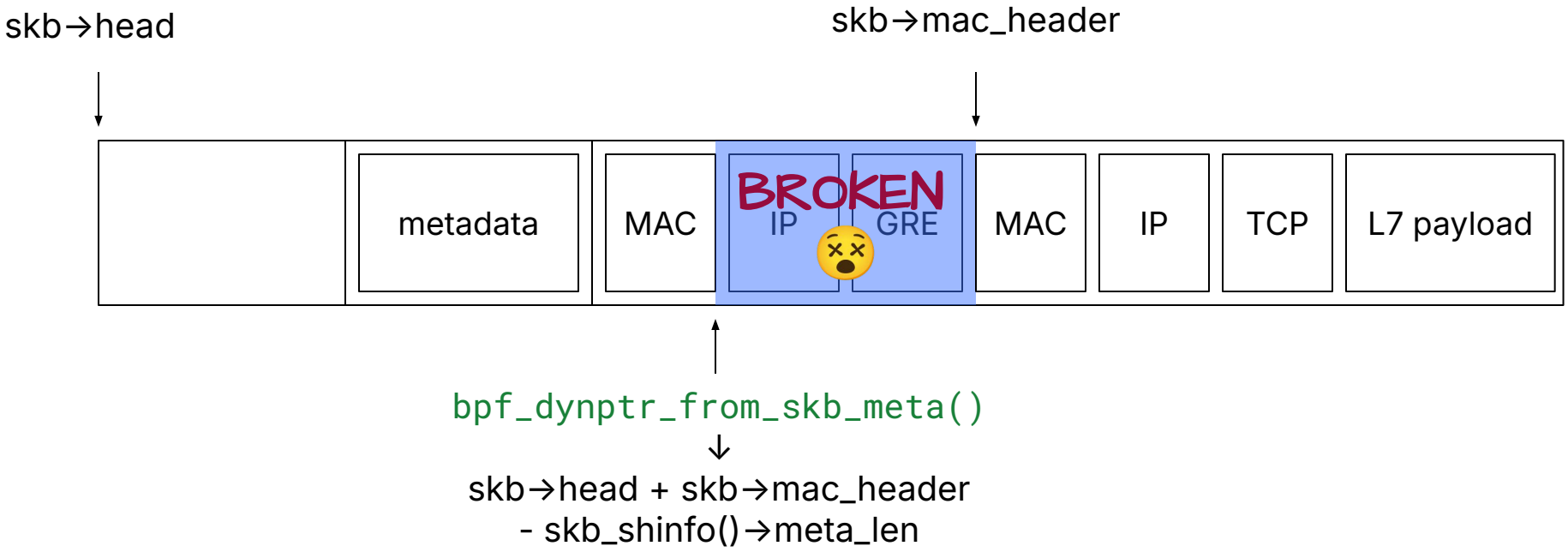
It does not rely on `skb->data` ...



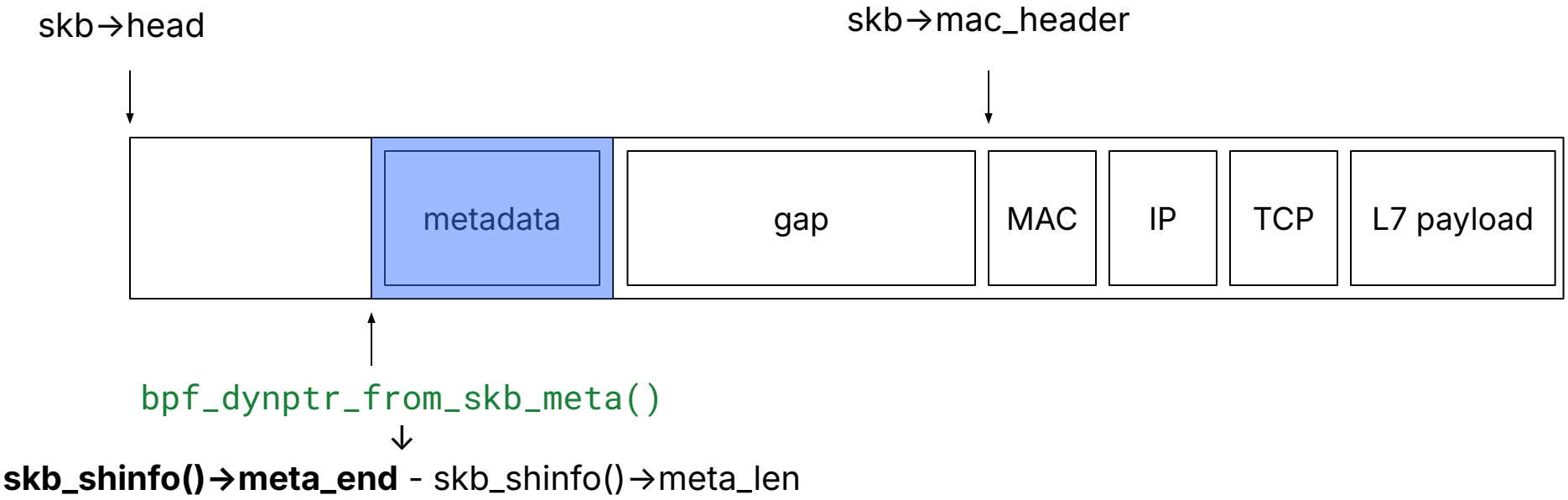
But if we have L2 tunnels...



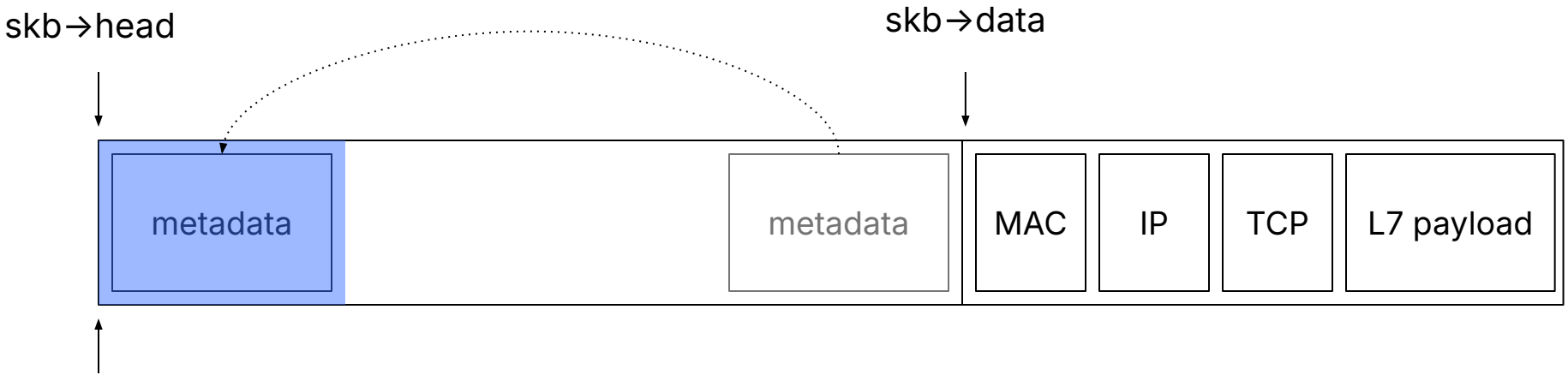
… it breaks down when you reset the MAC header



Solution?
Track metadata offset separately from MAC



Can move the metadata out of the way when pushing headers (TX path)

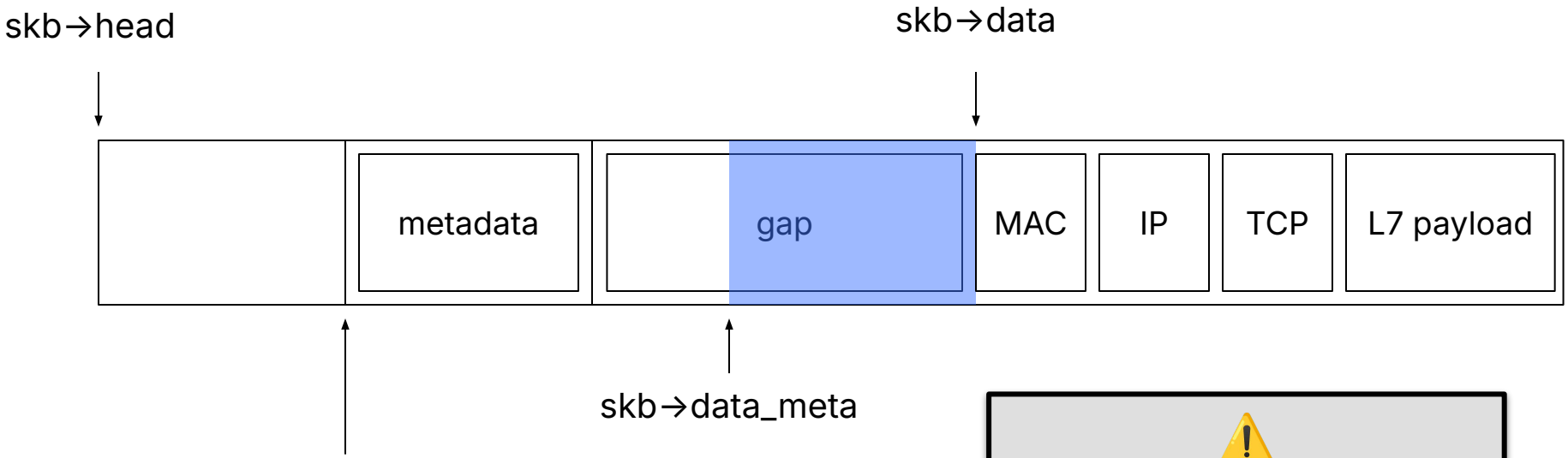


`bpf_dynptr_from_skb_meta()`

↓

skb_shinfo()→meta_end - skb_shinfo()→meta_len

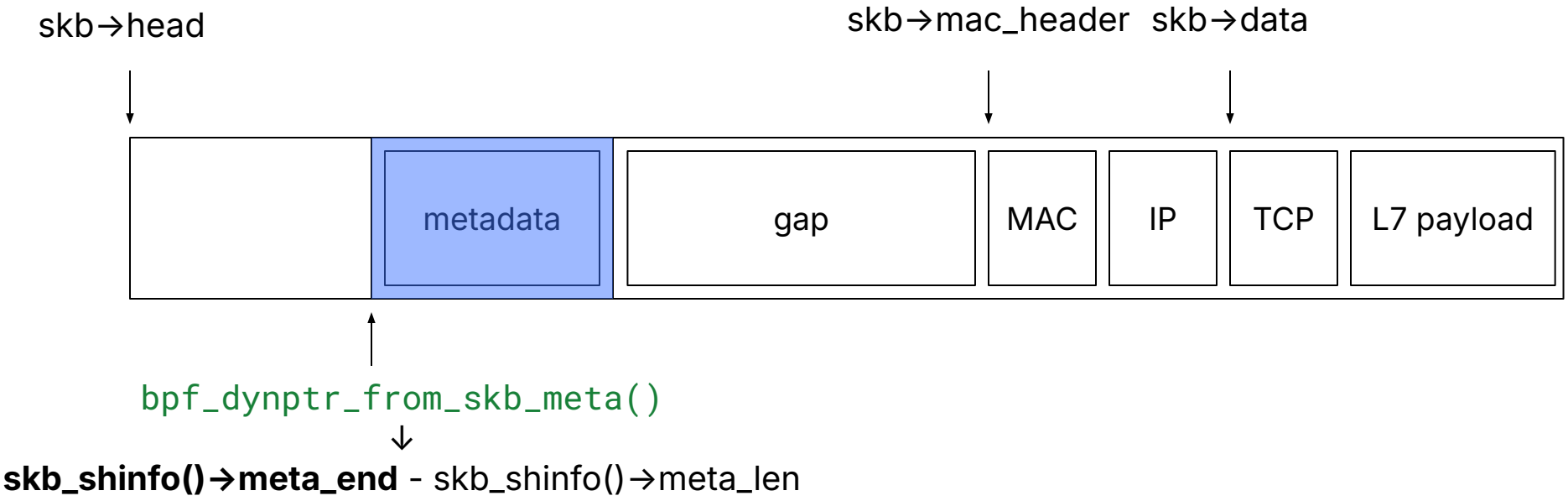
The curse of `skb->data_meta`



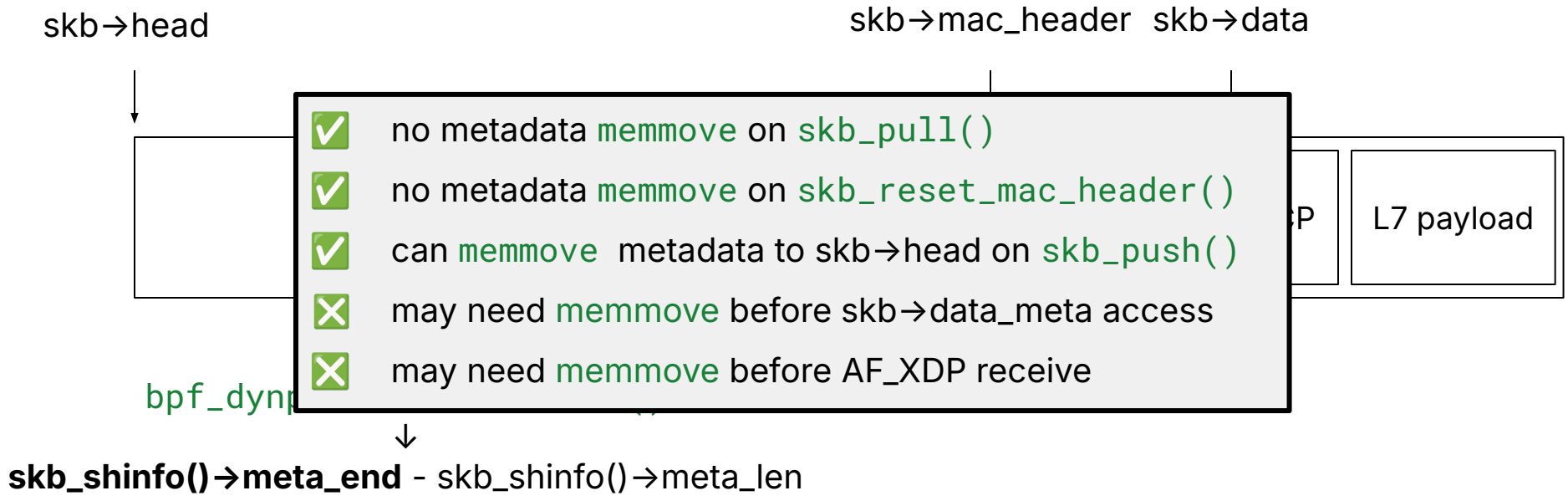
⚠️

We need to keep moving the metadata when using `skb->data_meta` in TC BPF

Proposed layout –
Metadata anywhere within the headroom



Proposed layout –
Metadata anywhere within the headroom



CLOUDFLARE

(The unexciting...)

Plan for uAPI

uAPI metadata access » Stream sockets

CLOUDFLARE

Incoming connection – reading metadata

```
// Opt in - Save metadata from TCP SYN packet
setsockopt(listener_fd, SOL_TCP, TCP_SAVE_SYN_METADATA,
           &(int){ 1 /* enable */ }, …);

conn_fd = accept(listener_fd, …);

// Retrieve metadata from TCP SYN
meta_len = sizeof(metadata);
getsockopt(conn_fd, SOL_TCP, TCP_SYN_METADATA,
           metadata, &meta_len);
```

Follow an existing API pattern – TCP_SAVE_SYN

uAPI metadata access » Stream sockets

CLOUDFLARE

Incoming connection – reading metadata

```
// Opt in - Save metadata from TCP SYN packet
setsockopt(listener_fd, SOL_TCP, TCP_SAVE_SYN_METADATA,
           &(int){ 1 /* enable */ }, …);

conn_fd = accept(listener_fd, …);

// Retrieve metadata from TCP SYN
meta_len = sizeof(metadata);
getsockopt(conn_fd, SOL_TCP, TCP_SYN_METADATA,
           metadata, &meta_len);
```

`metadata` is an
up to **252B**-long blob,
format not specified

uAPI metadata access » Stream sockets

CLOUDFLARE

Outgoing connection – writing metadata

```
conn_fd = socket(AF_INET, SOCK_STREAM, 0);

// Set metadata for TCP SYN packet
setsockopt(conn_fd, SOL_TCP, TCP_SYN_METADATA,
           metadata, sizeof(metadata));
// … and then initiate connection
connect(conn_fd, …);
```

uAPI metadata access » Datagram sockets

CLOUDFLARE

Receiving metadata

```
/* Opt in to receive packet traits */
setsockopt(sock_fd, SOL_SOCKET, SO_PKT_METADATA, &one, …);

/* Receive packet with ancillary message */
recvmsg(sock_fd, &msg, 0);

/* Retrieve traits */
for (cm = CMSG_FIRSTHDR(msg); cm; cm = CMSG_NXTHDR(msg, cmsg)) {
        if (cm->cmsg_level == SOL_SOCKET &&
            cm->cmsg_type == SCM_PKT_METADATA)
                memcpy(metadata, CMSG_DATA(cm), sizeof(metadata));
}
```

Follow an existing API pattern – SO_TIMESTAMPING

Metadata access » Datagram sockets

CLOUDFLARE

Sending metadata

```
char *metadata[] = { … };

struct msghdr *msg;
// …
cmsg              = CMSG_FIRSTHDR(msg);
cmsg->cmsg_level  = SOL_SOCKET;
cmsg->cmsg_type   = SCM_PKT_METADATA;
cmsg->cmsg_len    = CMSG_LEN(sizeof(metadata));
memcpy(CMSG_DATA(cmsg), metadata, sizeof(metadata));

sendmsg(fd, msg, 0);
```

📋 (Tentative) Roadmap

CLOUDFLARE

[x] Add bpf_dynptr_from_skb_meta()

[x] Make TC bpf_skb_*() helpers preserve metadata (bug fix)

[-] Make L2 decap preserve metadata (bug fix) [work in progress]

[ ] Allow-list bpf_dynptr_from_skb_meta() for other BPF prog types

[ ] uAPI read access to metadata

-=-=- Milestone: Metadata on RX path -=-=-

[ ] [FWD path] Make L2 encap preserve metadata (bug fix)

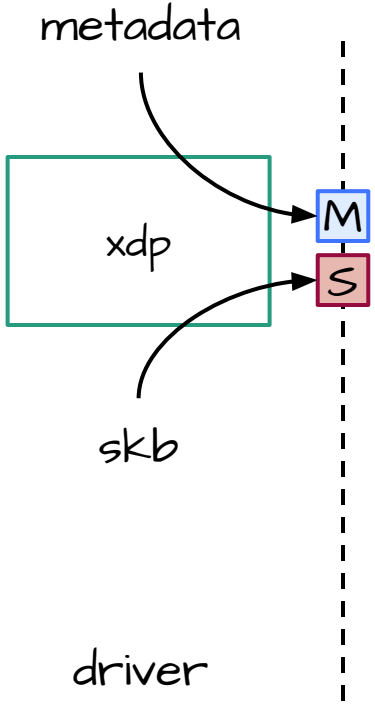
[ ] [TX path] uAPI write access to metadata

[ ] [TX path] Alloc metadata from other BPF prog types

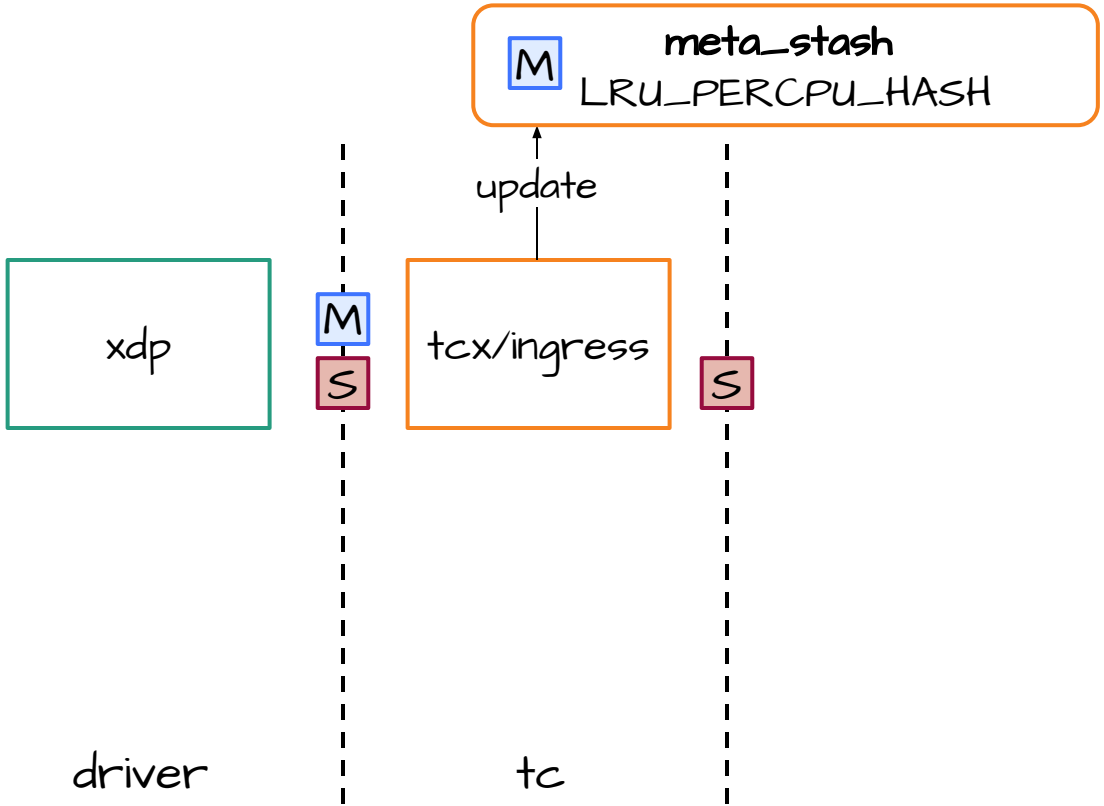
Lessons from production

How to pass metadata to user-space when net stack doesn’t support it yet?

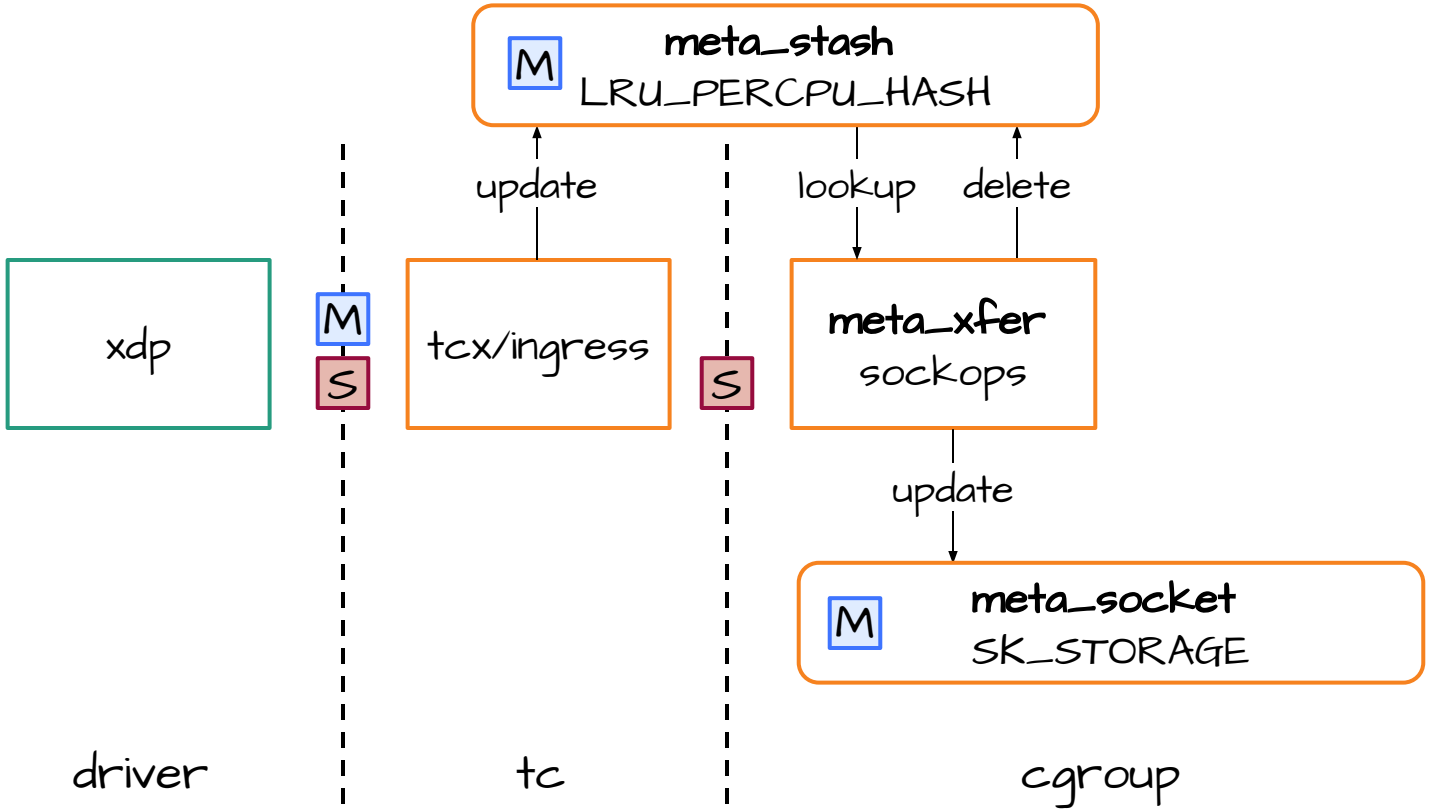
Recipe for TCP - custom BPF `getsockopt`



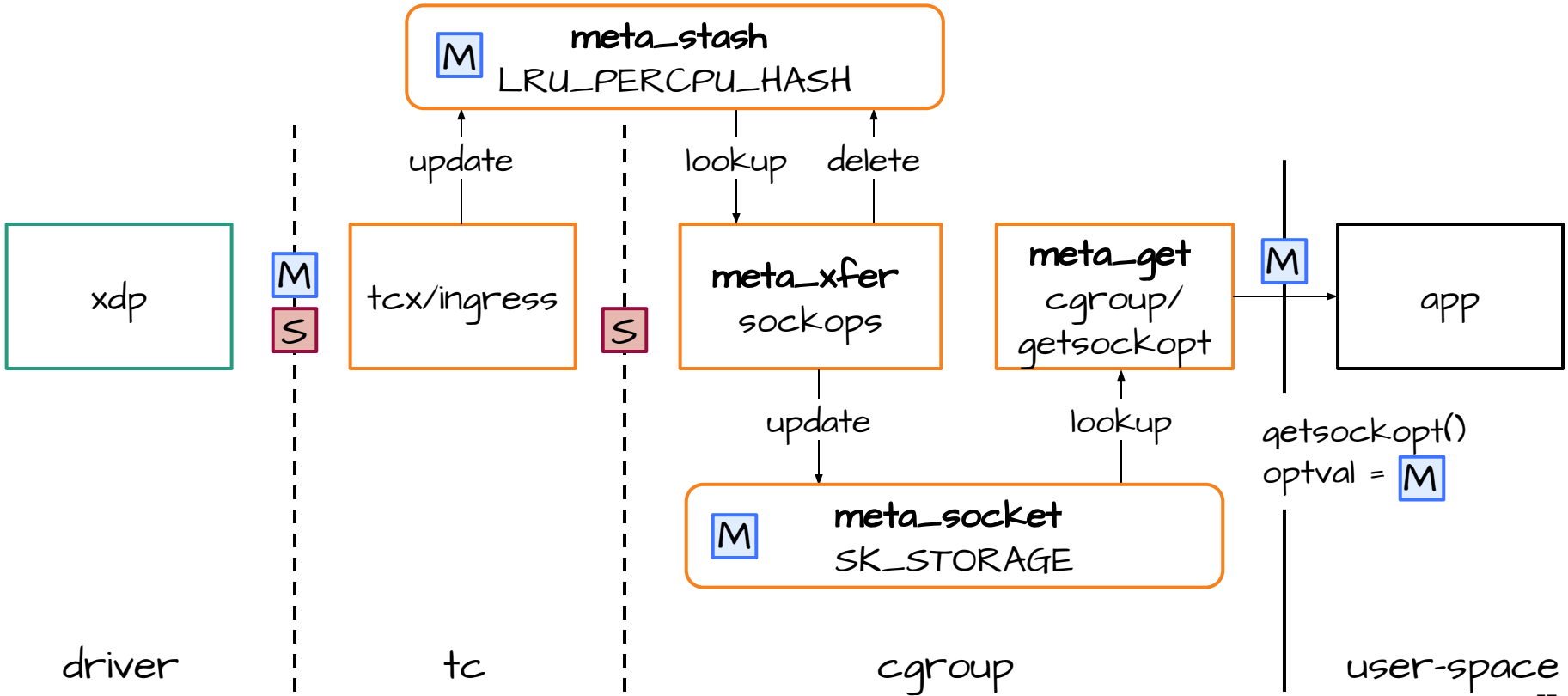
Recipe for TCP - custom BPF `getsockopt`



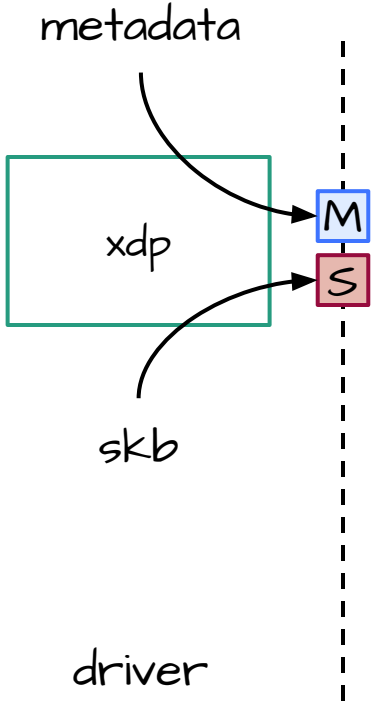
Recipe for TCP - custom BPF `getsockopt`



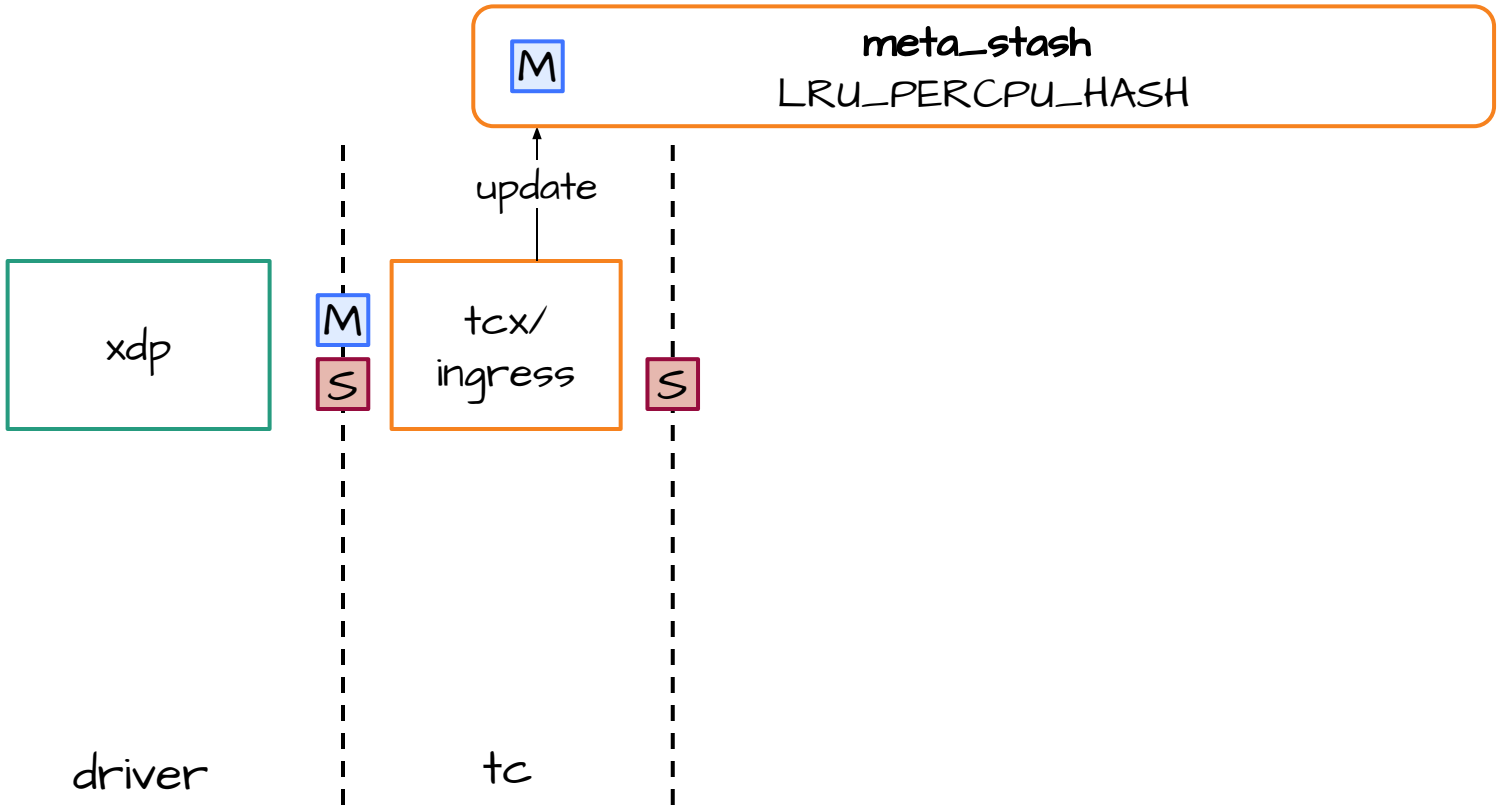
Recipe for TCP - custom BPF `getsockopt`



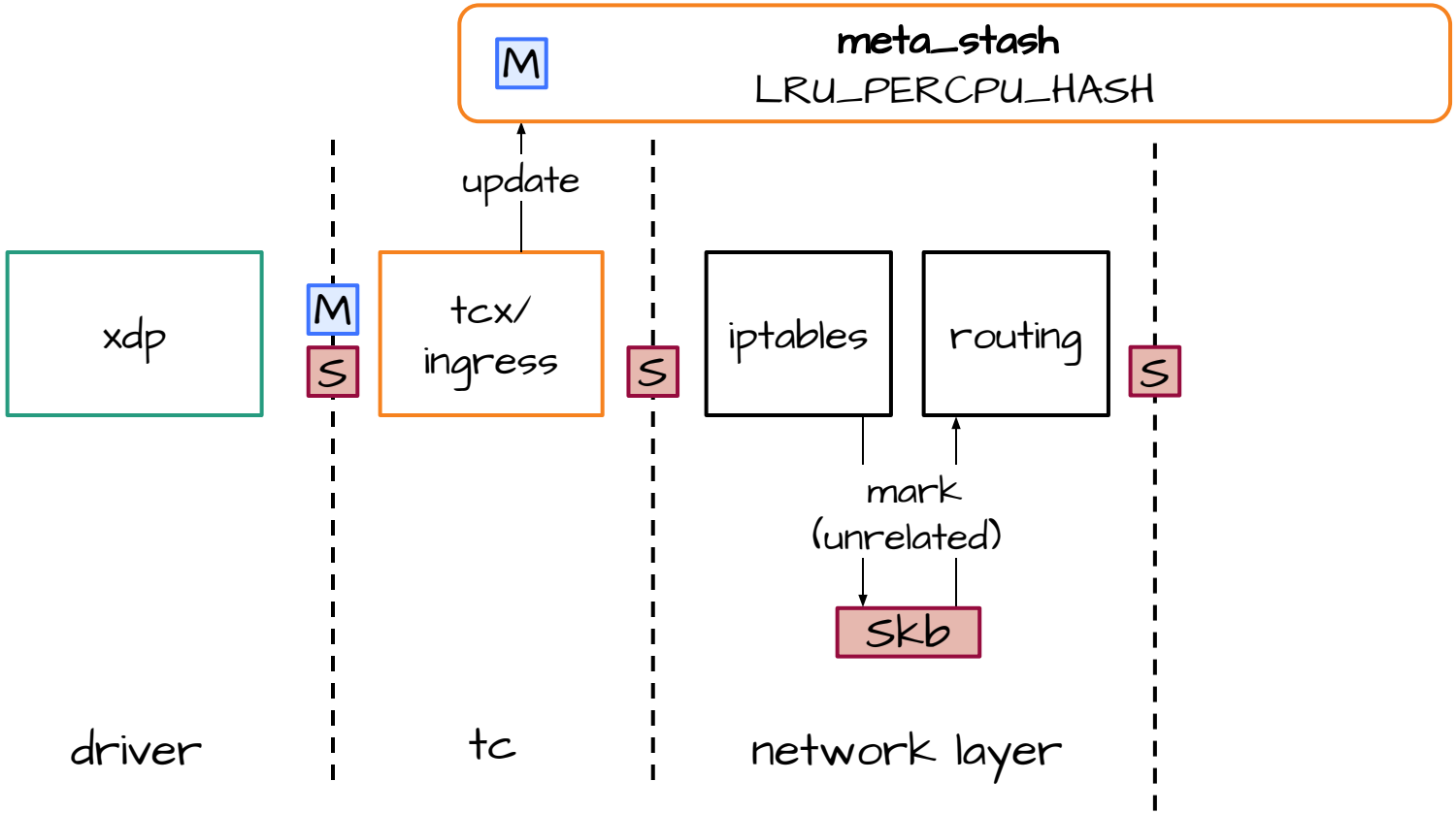
Recipe for UDP - repurpose SO_RCVMARK / SO_MARK



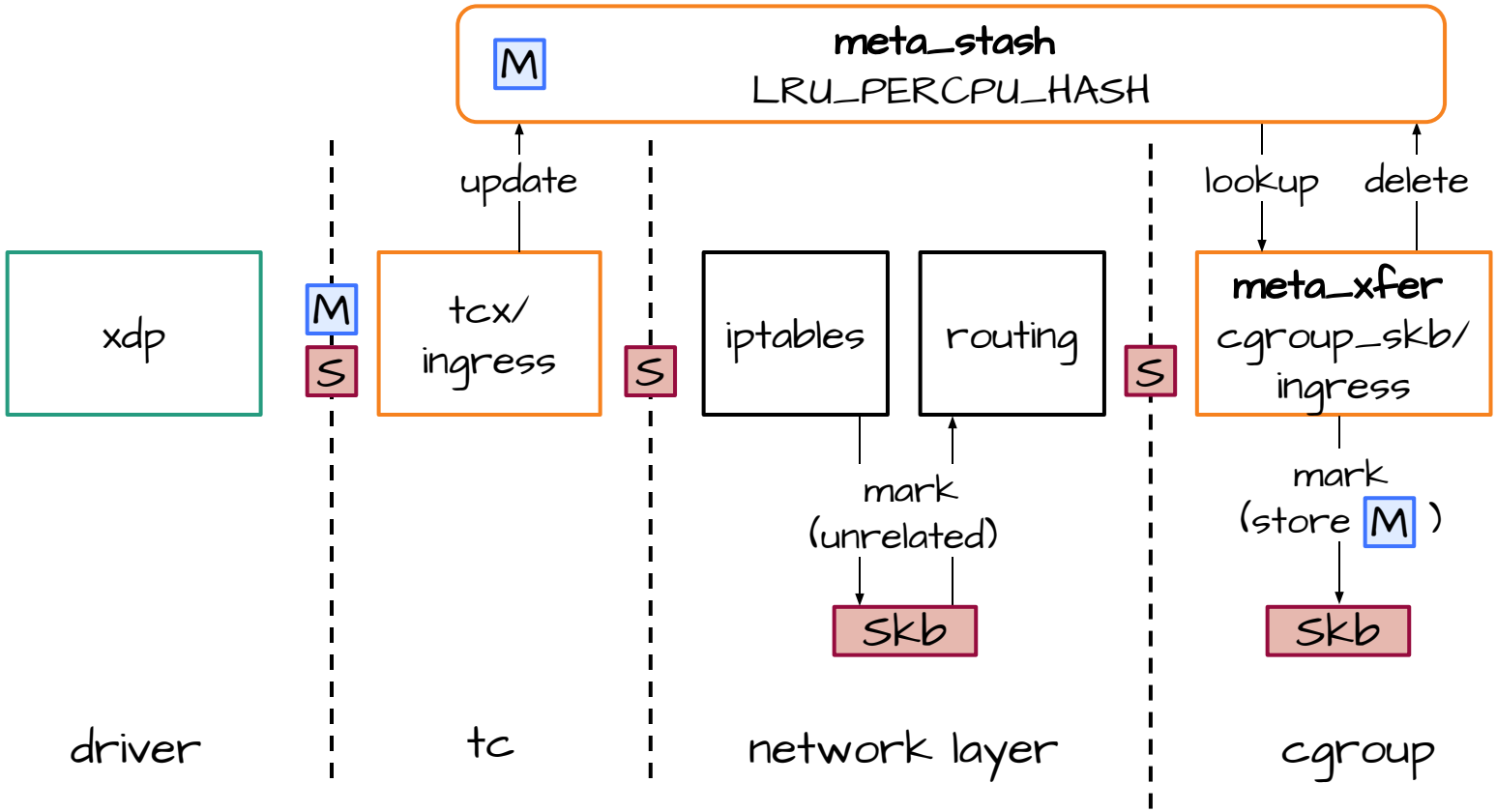
Recipe for UDP - repurpose SO_RCVMARK / SO_MARK



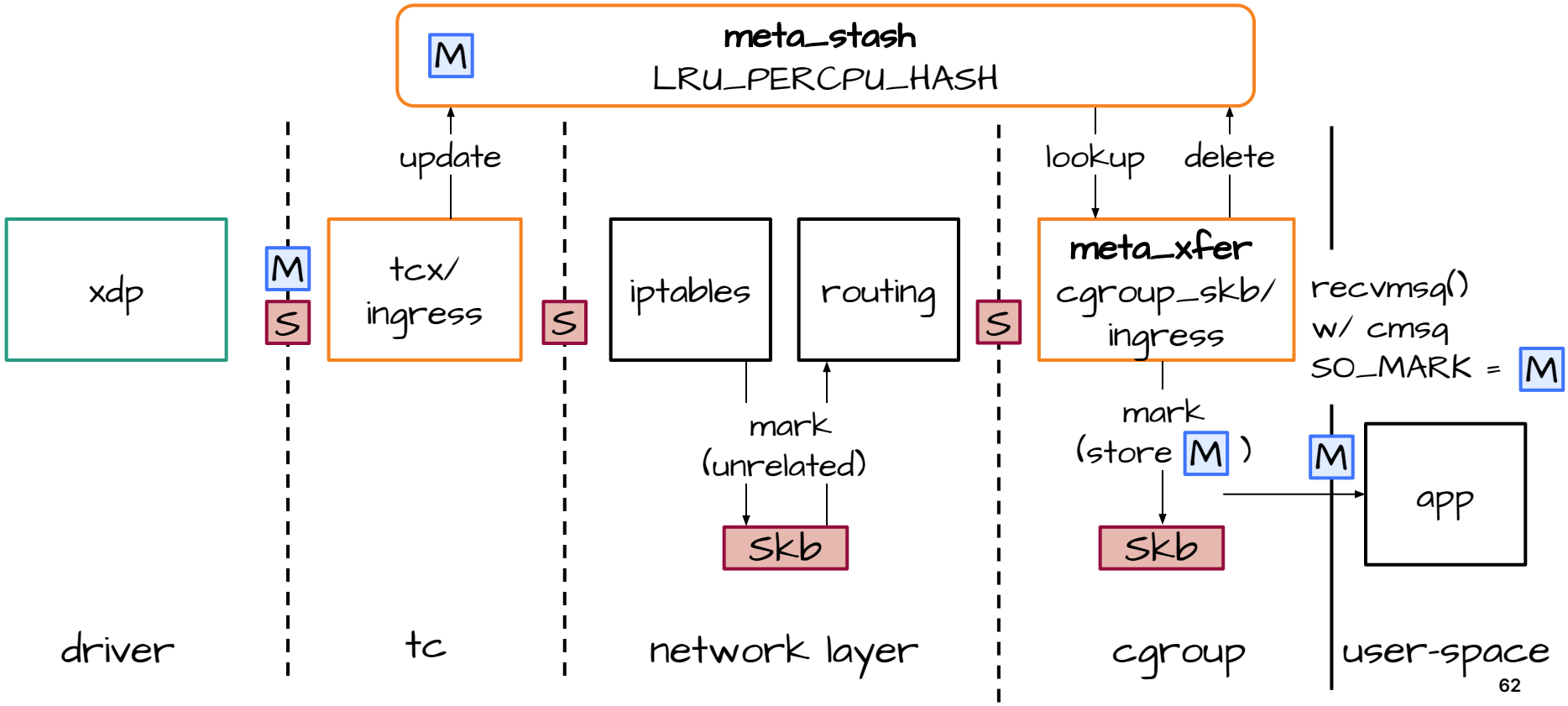
Recipe for UDP - repurpose SO_RCVMARK / SO_MARK



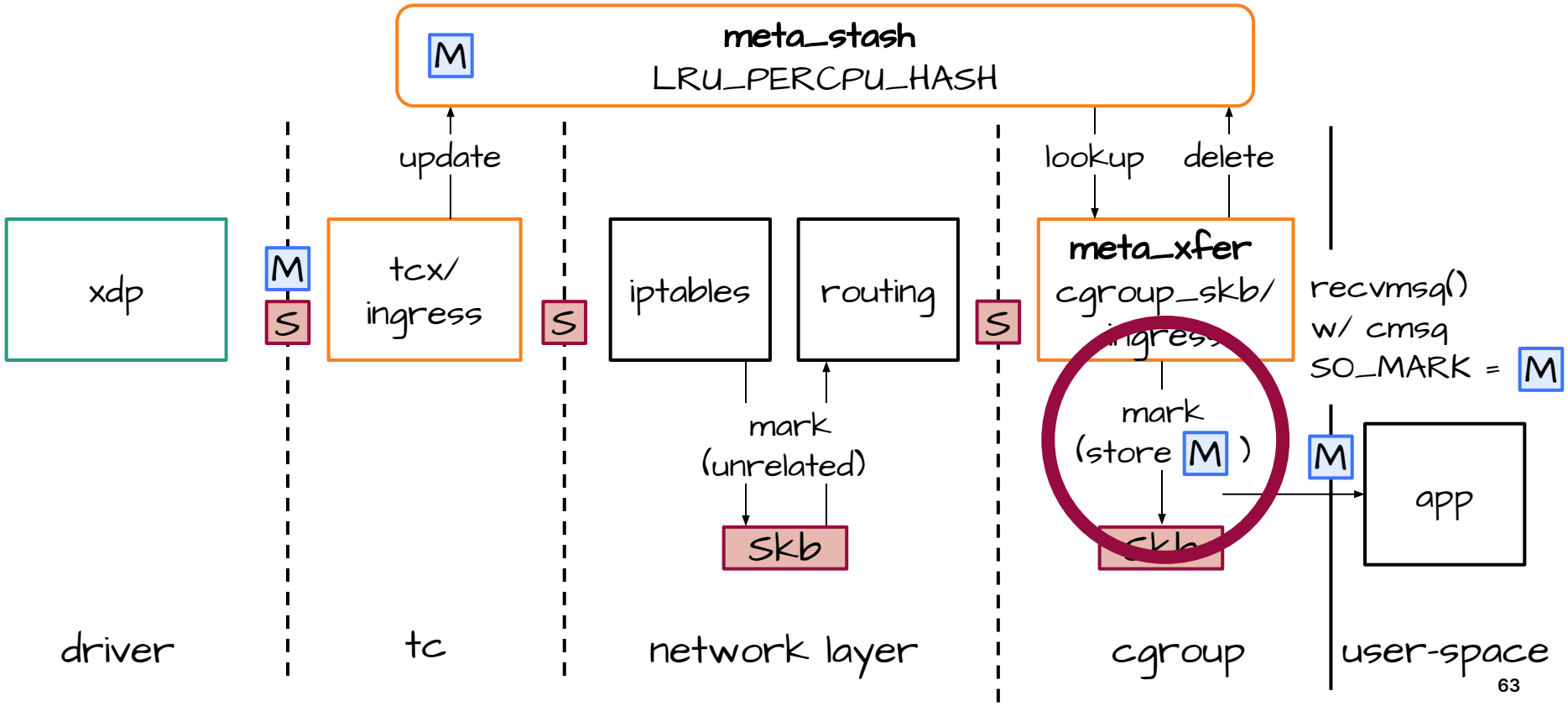
Recipe for UDP - repurpose SO_RCVMARK / SO_MARK



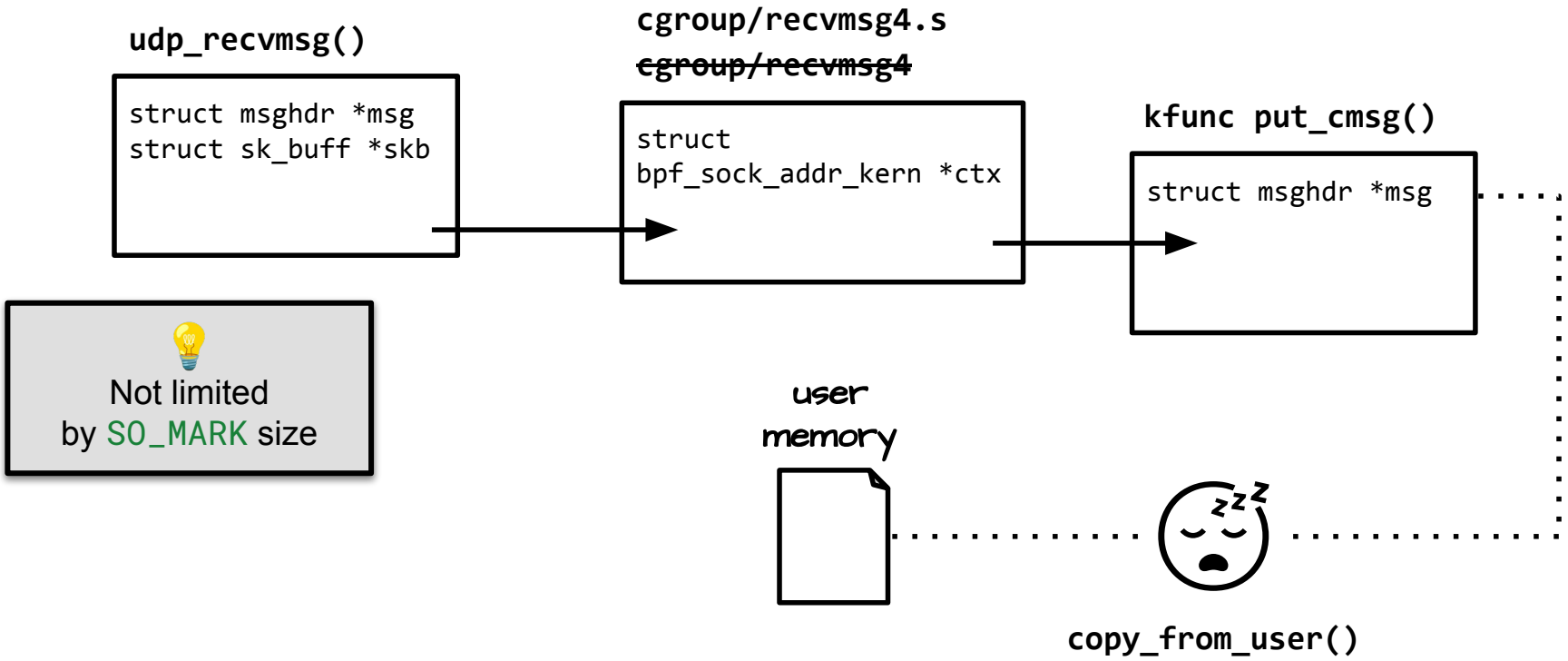
Recipe for UDP - repurpose SO_RCVMARK / SO_MARK



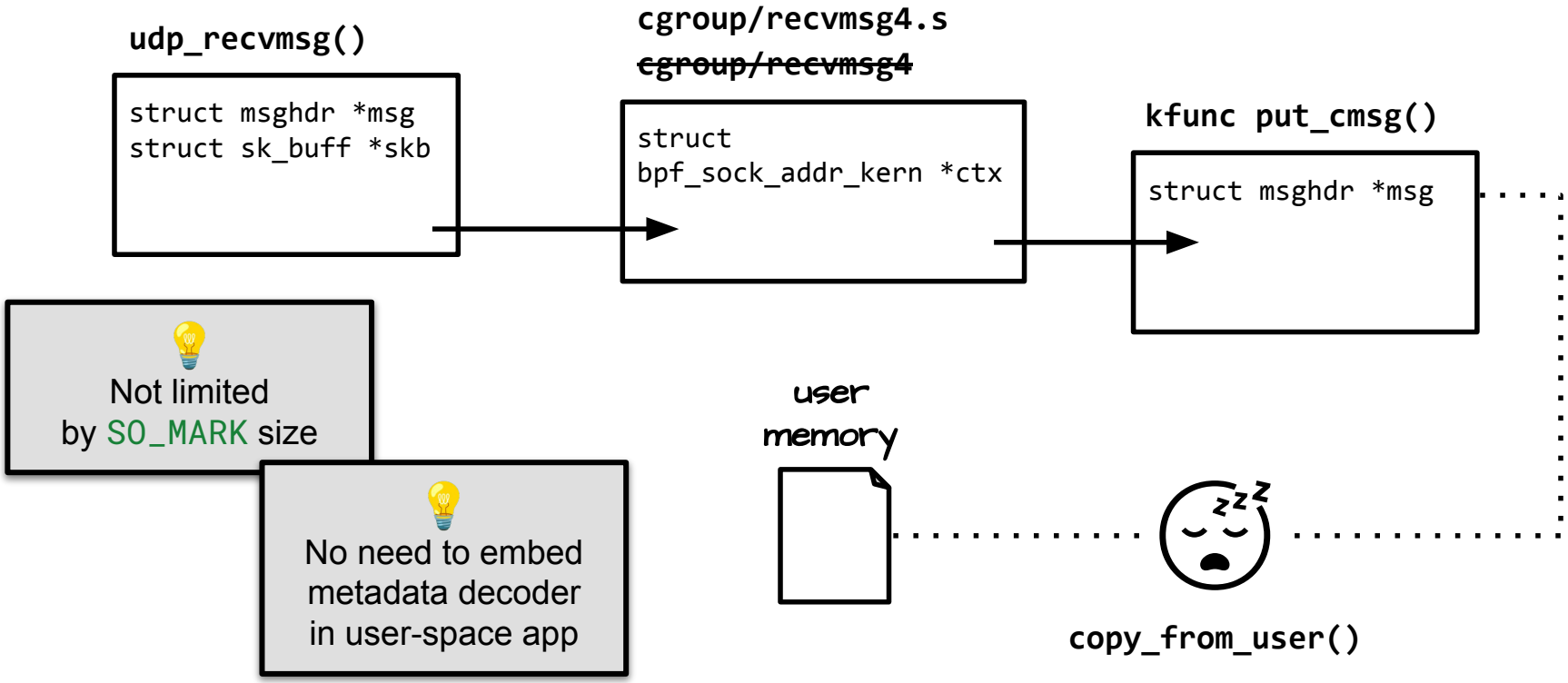
Recipe for UDP - repurpose SO_RCVMARK / SO_MARK



Blast from the past (LPC 2024) – Idea: Produce `cmsg`'s from BPF

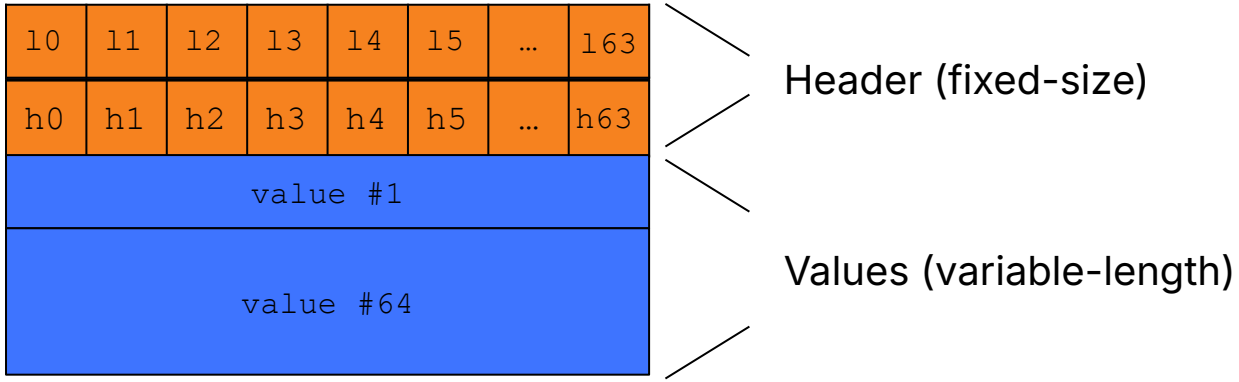


Blast from the past (LPC 2024) – Idea: Produce `cmsg`'s from BPF



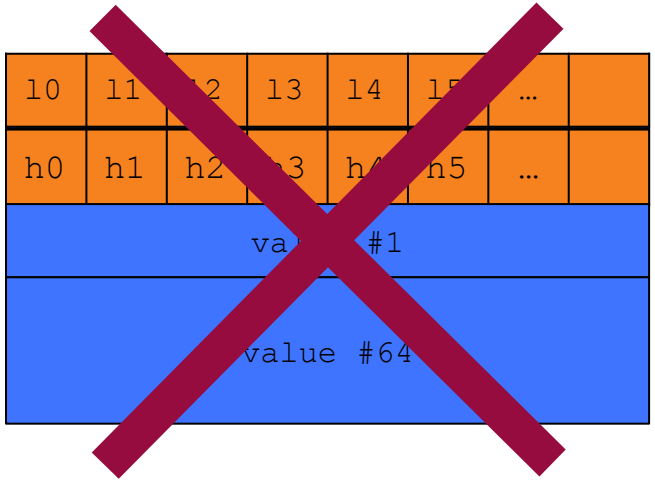
Encoding format – Pivot from K/V to TLV

Initial approach – Space-efficient K/V format



- Up to 64 entries
- Supported value size:
  - `0` (flag), `4B`, `8B`

Initial approach – Didn't meet real-life needs



Needed a data structure that:

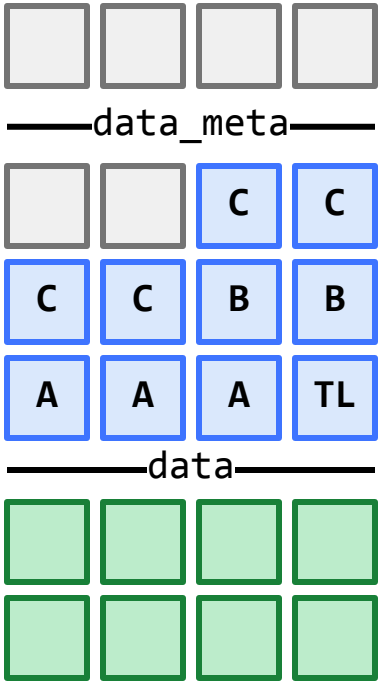
1. Can store ordered, repeated items

- multiple timestamps
- multiple tracing IDs
- info from multiple layers of encap

2. Can serve as scratch space

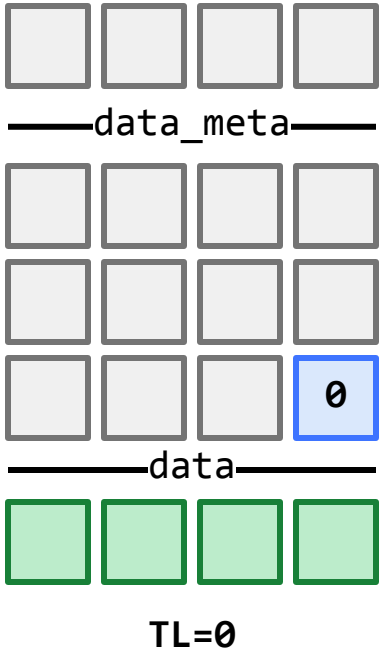
- pass info between `bpf_tailcall`'ed programs

Current approach – TLV tuples + total length

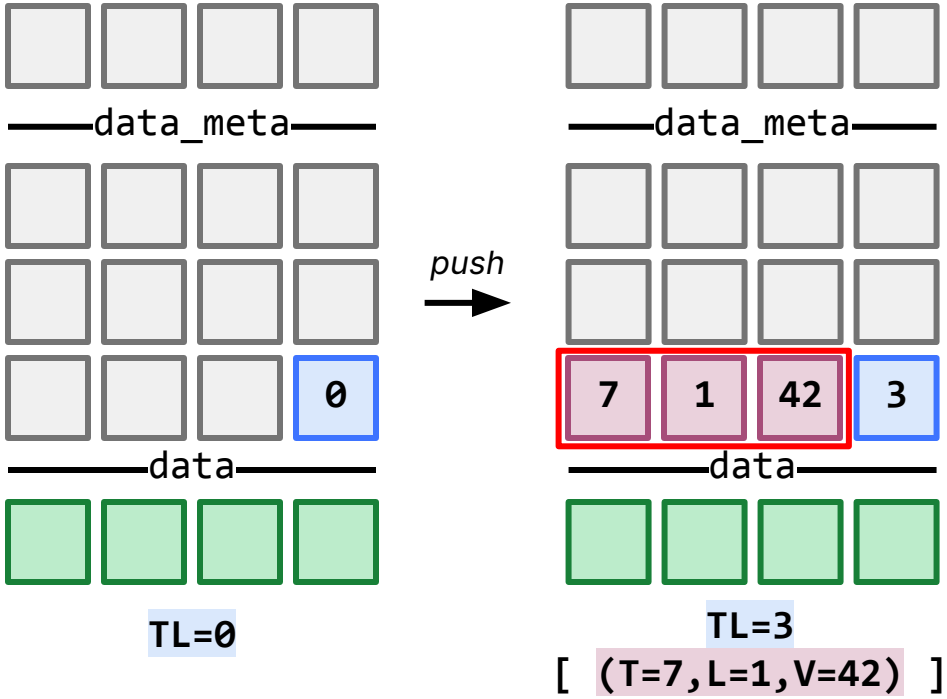


- Grows bottom-up
  No `memmove` on `bpf_xdp_adjust_meta`
- Total length (`TL`) as first byte
  No need to initialize whole metadata area
  Popping most recent item is cheap
- Arbitrary value lengths
  Flags modelled as `0`-length entries
  Self-imposed limit of `16B` max value length
- Can hold ordered, repeated entries

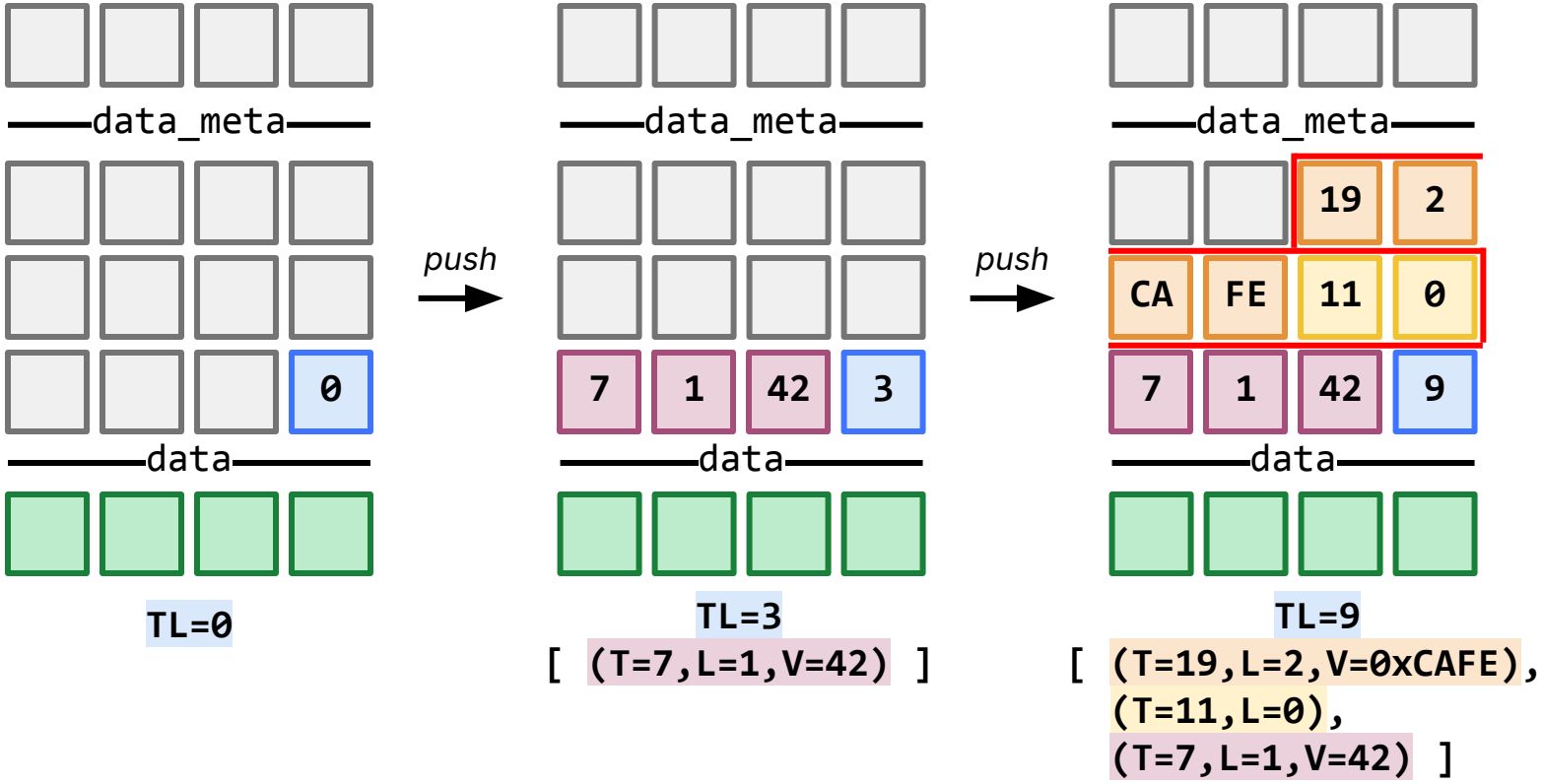
Current approach –
Example – push 3 pieces of metadata, pop one



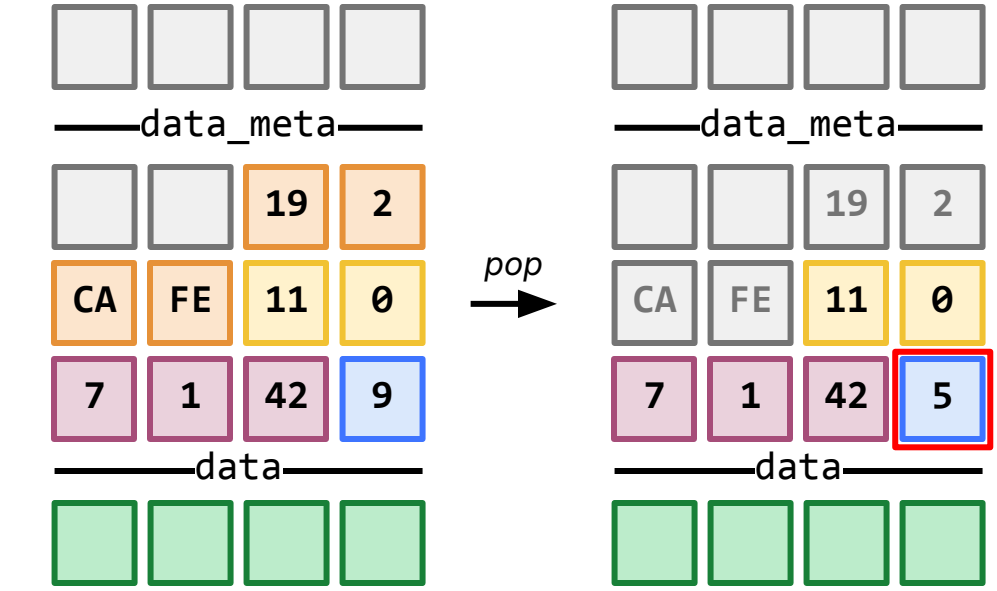
Current approach –
Example – push 3 pieces of metadata, pop one



Current approach –
Example – push 3 pieces of metadata, pop one



Current approach –
Example – push 3 pieces of metadata, pop one



TL=9
[(T=19,L=2,V=0xCAFE),
(T=11,L=0),
(T=7,L=1,V=42)]

TL=5
[(T=11,L=0),
(T=7,L=1,V=42)]

Testing pains ☹

BPF_PROG_TEST_RUN with metadata...

CLOUDFLARE

```
void test_xdp_metadata(int prog_fd)

{
    struct xdp_md ctx_in = {};

    /* … */
    ctx_in.data_meta = <meta end offset>;
    ctx_in.data = <meta end offset>;
    ctx_in.data_end = <data end offset>;
    opts.ctx_in = &ctx_in;
    opts.ctx_size_in = sizeof(ctx_in);
    err = bpf_prog_test_run_opts(prog_fd, &opts);
    /* … */
}
```

BPF_PROG_TEST_RUN with metadata...
... or the lack thereof for __sk_buff context

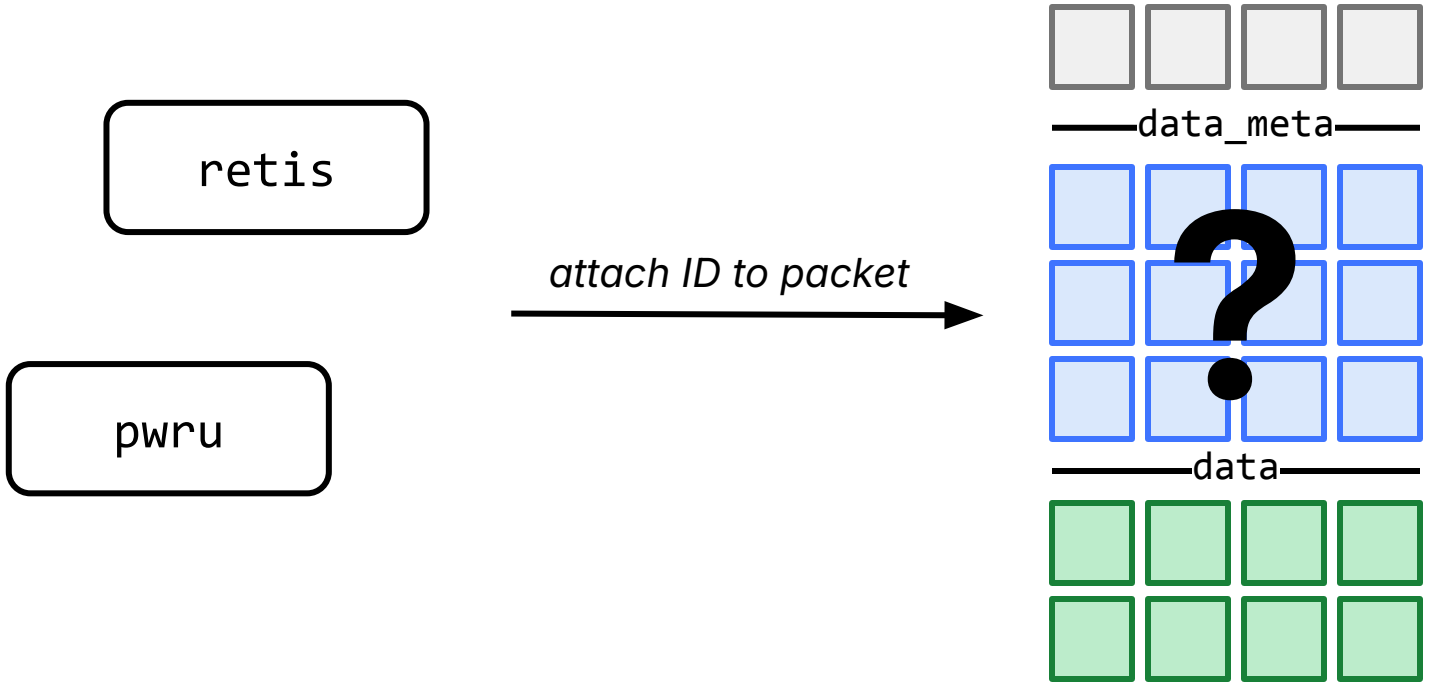
```
void test_xdp_metadata(int prog_fd)

{
    struct xdp_md ctx_in = {};

    /* … */
    ctx_in.data_meta = <meta end offset>;
    ctx_in.data = <meta end offset>;
    ctx_in.data_end = <data end offset>;
                      &ctx_in;
                  _in = sizeof(ct
                  g_test_run_opts
}
```

supported by
Bpf_prog_test_run_xdp
(XDP progs)

missing for
bpf_prog_test_run_skb
(TC BPF progs)

Problems without (good) answers...

Integration with 3rd party tools

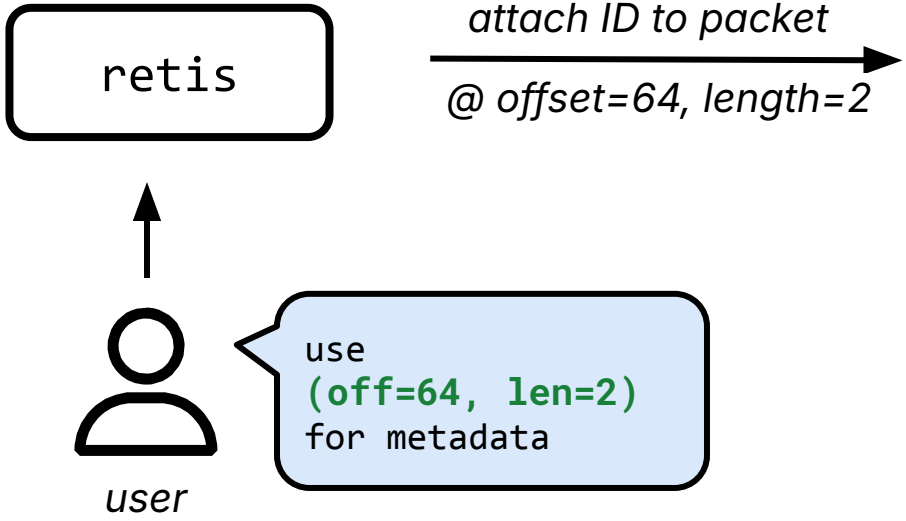


https://github.com/retis-org/retis
https://github.com/cilium/pwru

Integration with 3rd party tools

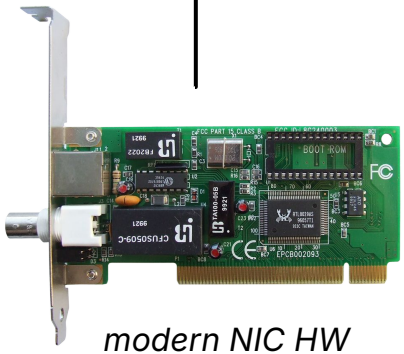
CLOUDFLARE

🤔 Make user specify where to place metadata?

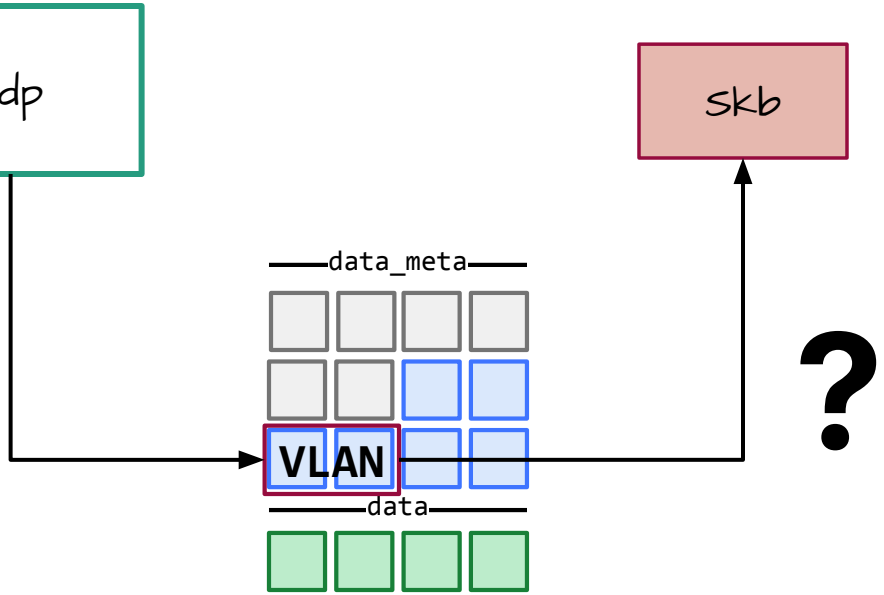


Accessing metadata from the network stack – Use case: Consuming NIC metadata

```
bpf_xdp_metadata_rx_timestamp()
bpf_xdp_metadata_rx_hash()
bpf_xdp_metadata_rx_vlan_tag()
```



modern NIC HW



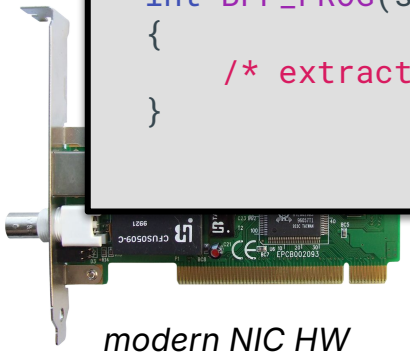
Accessing metadata from the network stack – Use case: Consuming NIC metadata

CLOUDFLARE

bpf_xdp_metadata_rx_timestamp()
bpf_
bpf_

🤔 struct_ops callbacks to the rescue?

```
SEC("struct_ops/set_rx_hash")
int BPF_PROG(set_rx_hash_from_metadata, struct sk_buff *skb)
{
    /* extract RX hash from skb metadata and set skb->hash */
}
```



modern NIC HW

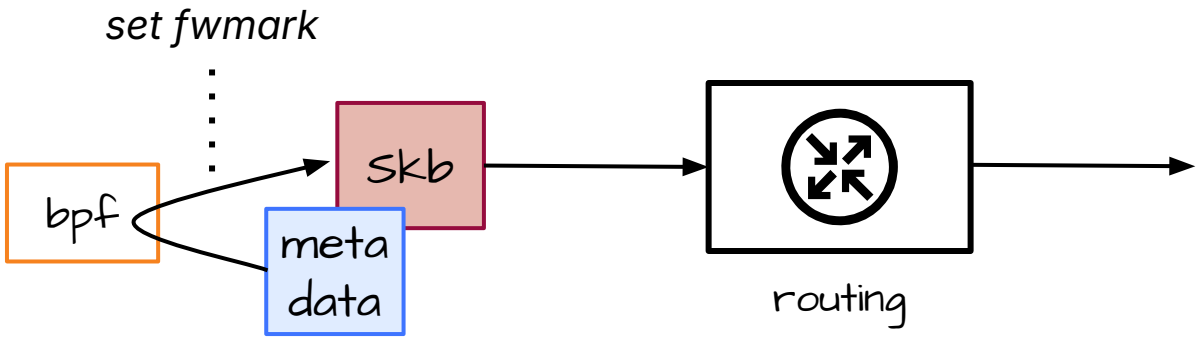
kb

VLAN

data

?

Accessing metadata from the network stack – Use case: Policy routing by metadata

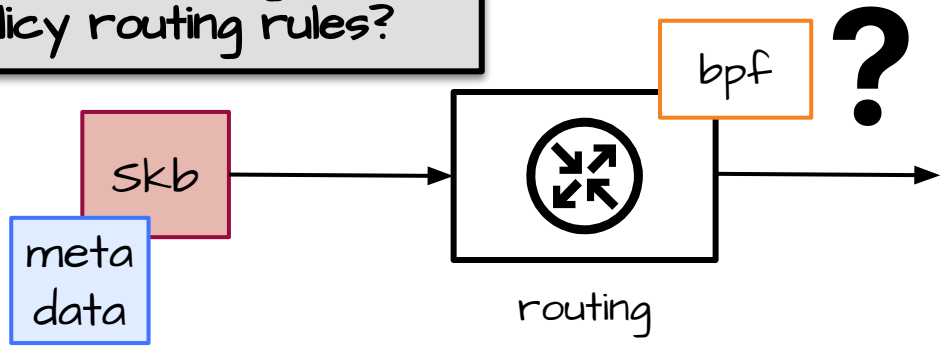



```
$ ip rule
0:      from all lookup local
32765:  not from all fwmark 0x100cf lookup 65743
32766:  from all lookup main
32767:  from all lookup default
```

Accessing metadata from the network stack – Use case: Policy routing by metadata

CLOUDFLARE

🤔 Have `sk_filter` programs attached to policy routing rules?




```
$ ip rule
0:      from all lookup local
32765:  not from all /sys/fs/bpf/sk_filter lookup 65743
32766:  from all lookup main
32767:  from all lookup default
```

Resources (newest to oldest)

CLOUDFLARE

Patch series ✉

1. [PATCH RFC bpf-next 00/15] Decouple skb metadata tracking from MAC header offset 🔗
2. [PATCH bpf-next v4 00/16] Make TC BPF helpers preserve skb metadata 🔗
3. [PATCH bpf-next v7 0/9] Add a dynptr type for skb metadata for TC BPF 🔗
4. [PATCH RFC bpf-next v2 00/17] traits: Per packet metadata KV store 🔗

Talks 📣

1. Netdev 0x19 2025: Traits: Rich Packet Metadata 🔗
2. Linux Plumbers Conference 2024: Marking Packets With Rich Metadata 🔗

Thank you

jakub@cloudflare.com

wferguson@cloudflare.com

tiagolam@cloudflare.com