パケット処理の独自実装や高速化手法の比較と実践

Survey and Implementation examples of
custom packet processing and acceleration methods

JANOG45 ＠札幌

海老澤健太郎
トヨタ自動車株式会社

日下部雄也
BBSakura Networks株式会社

本日のお題

パケット処理とは？

（ハードウェア＆ソフトウェア処理の違い）

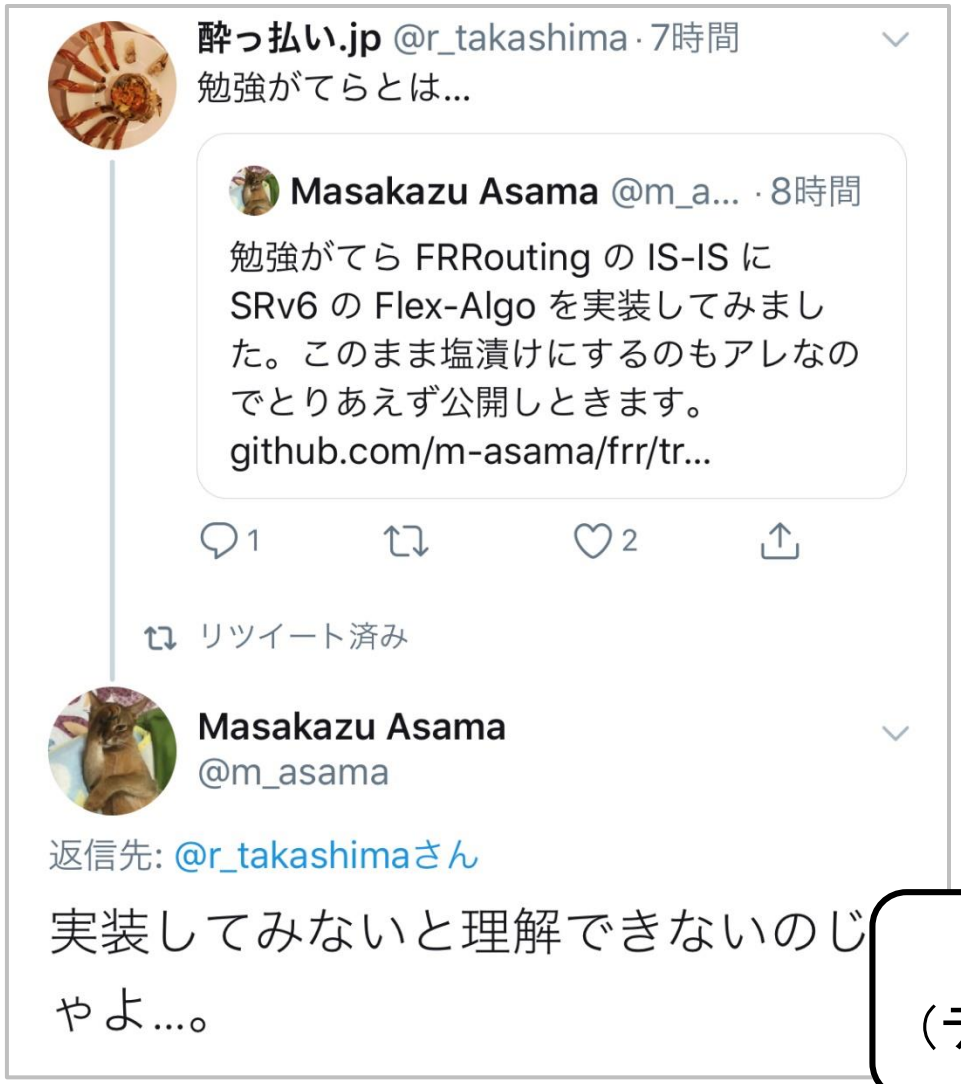
パケット処理の独自実装や高速化手法の紹介

それぞれの特徴や適用領域の違いについて比較

独自パケット処理の実装例（P4/XDP）

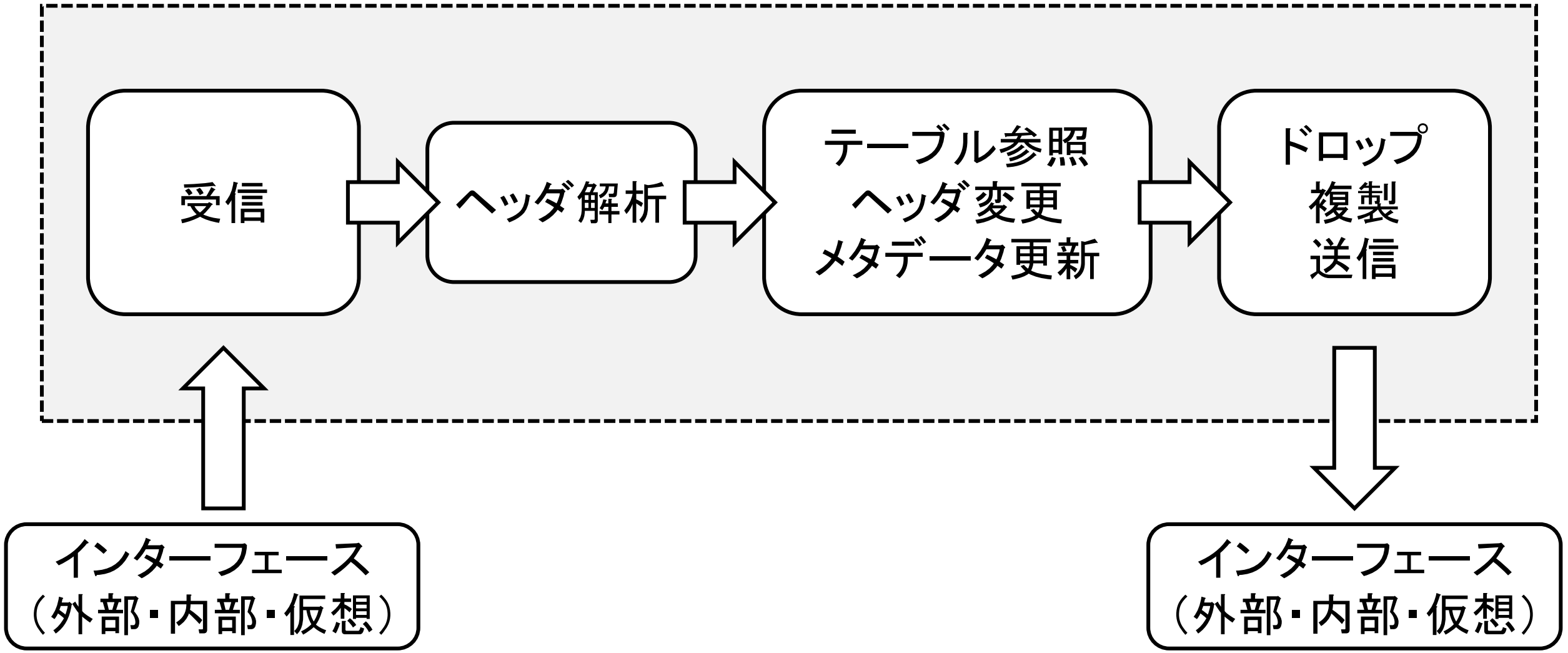
（サンプルコードを元に具体的な実装方法や実行環境の解説）

パケット処理を独自実装する “もうひとつの” 理由



実装する事でパケット処理
（データプレーン）への理解が深まる

パケット処理とは？


受信
ヘッダ解析
テーブル参照
ヘッダ変更
メタデータ更新
ドロップ
複製
送信
インターフェース
（外部・内部・仮想）
インターフェース
（外部・内部・仮想）

実装の種類

ハードウェア

専用回路を用いた処理

ASIC

FPGA

ソフトウェア

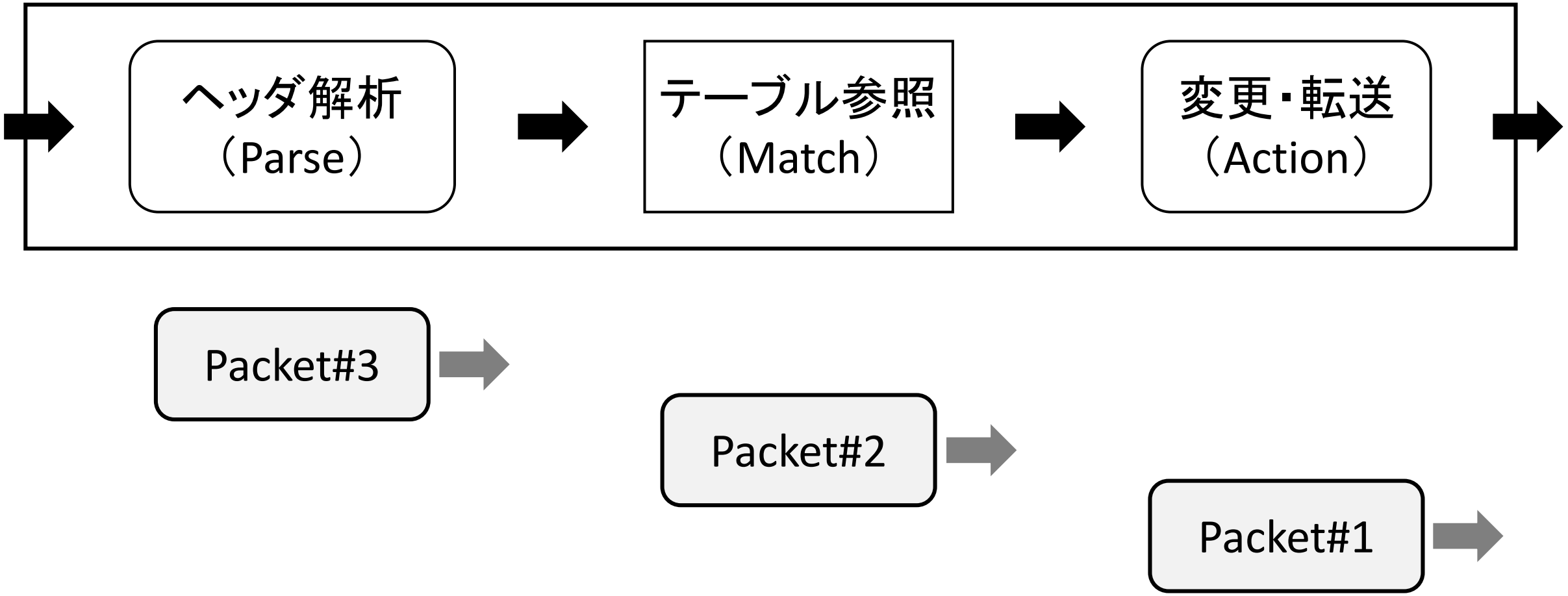
CPUを用いた処理

ソケット通信
カーネルモジュール
XDP（Express Data Path）
DPDK（Data Plane Development Kit）

ハードウェア実装によるパケット処理

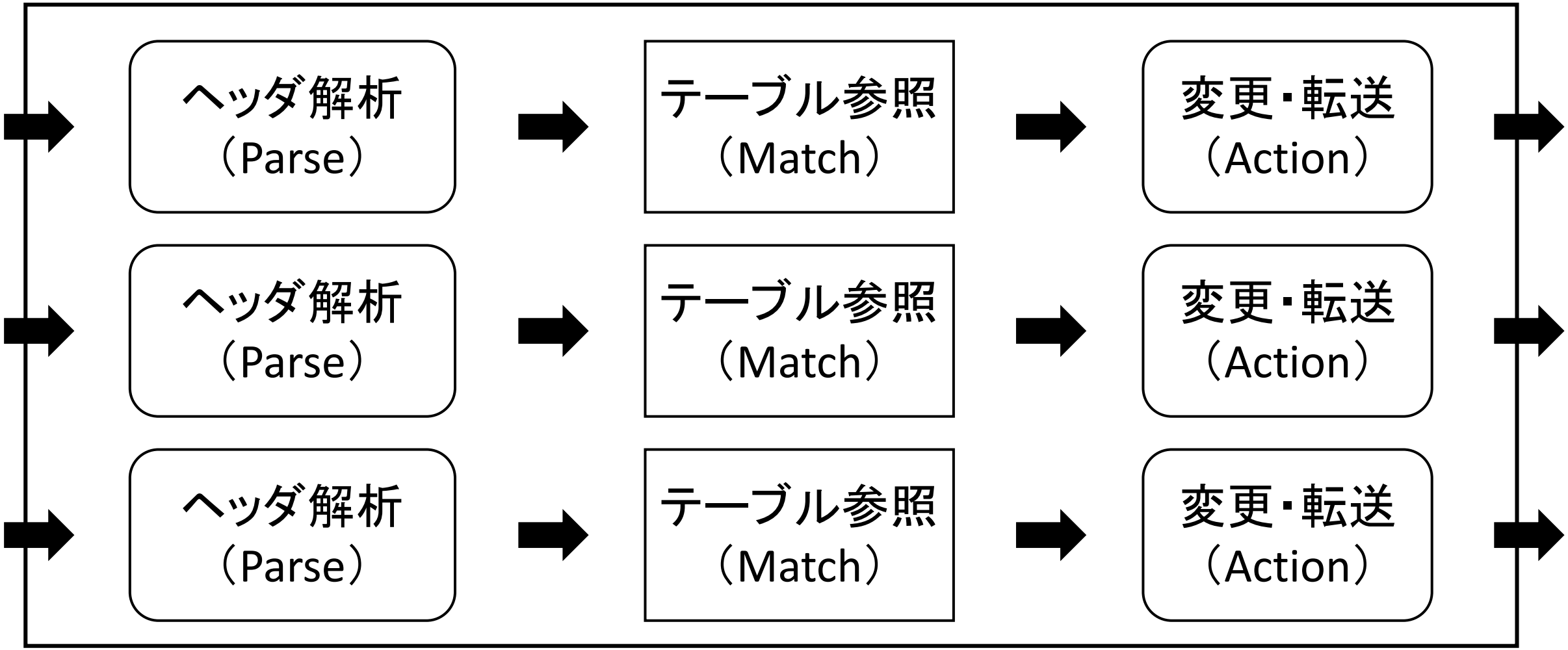
「パイプライン」がハードウェアによるパケット処理の基本

複数パケットの並列処理



「パイプライン」がハードウェアによるパケット処理の基本

パイプラインの並列化により性能がxx倍



ハードウェアのリソース ≒ 回路の量（チップ上の面積）

「配線」が多くなる処理は苦手

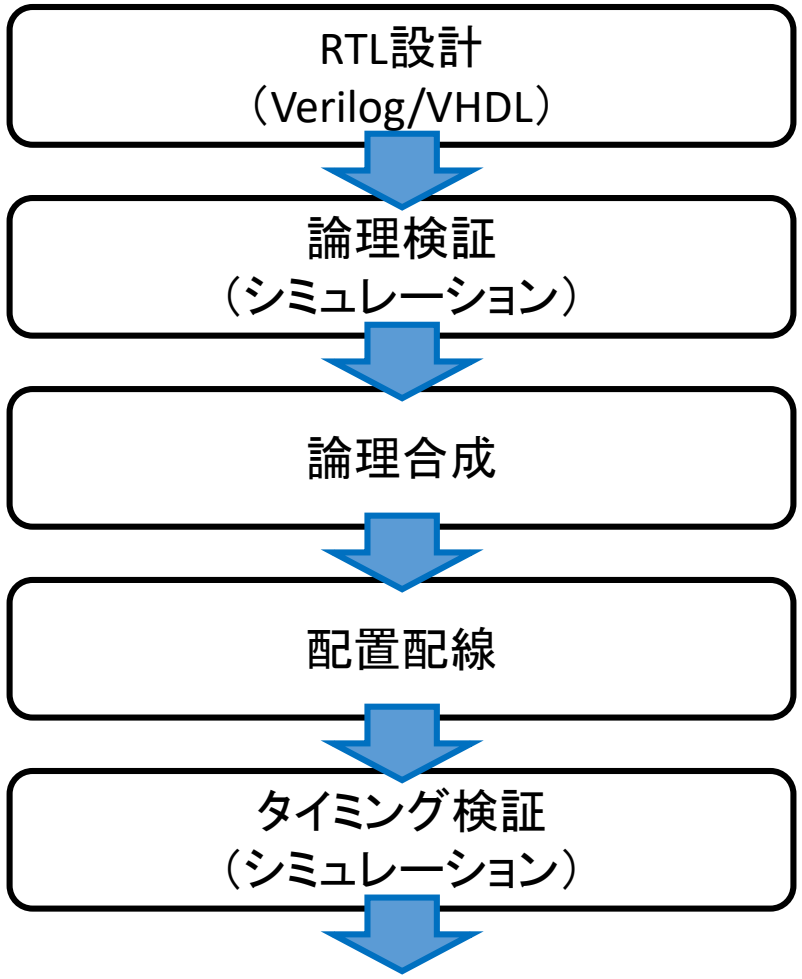
⇒ マッチテーブル・キーの長さ
⇒ パースするヘッダの長さ
⇒「パケット全体を変更する」という処理は苦手

「配線の変更」を可能とするにはより多くのリソースが必要

⇒ レジスタ（メモリ）やそれに合わせて動作が変わる回路たち
⇒ カスタマイズ（独自処理）可能な箇所は最小限にしたい

FPGAの特徴

一般的なFPGA開発

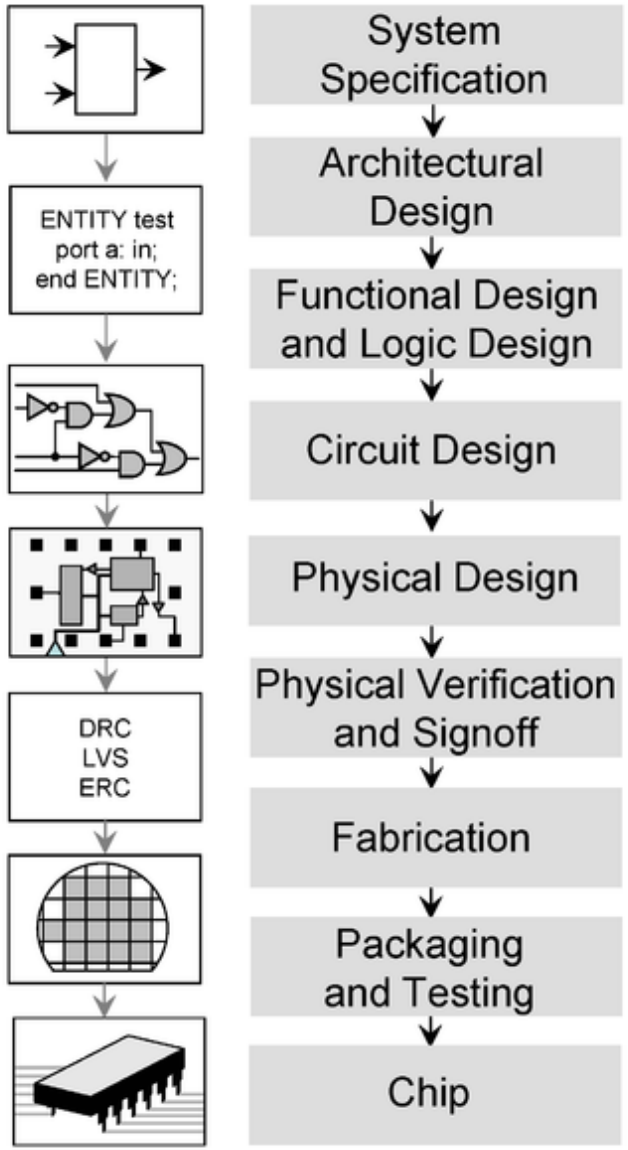


回路をすべて独自処理実装可能
（任意の変換ロジックを実装可能）

中速・高消費電力

回路を意識したプログラミングが必要

プログラミングの難易度が高い



ASIC の特徴

（基本）回路は固定

高速・低消費電力

回路変更にはチップの焼き直しが必要

（レジスタなどを利用した動作変更も可能だが最小限：バグ対応のため etc.）

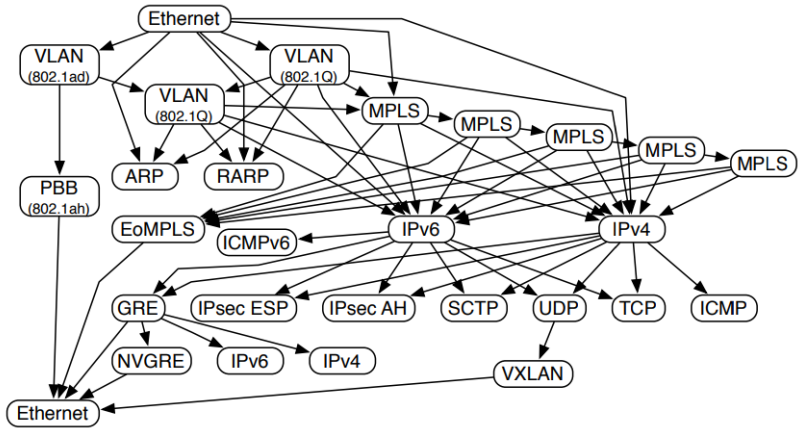
変更に必要な期間が月・年単位

https://commons.wikimedia.org/wiki/File:Integrated_circuit_design.png

（ハードウェア視点での）パケット処理の独自実装とは？

レジスタなどを利用した “特定回路” のカスタマイズ

ヘッダ フォーマットの定義　マッチフィールドの定義　アクションの定義
パースグラフの構築　テーブルタイプの定義　フィールド操作ロジック



- MAC address
- IPv4 address
- proto + TCP ports
- (any header fields)

- drop
- forward
- copy
- push / pop
- add(+) sub(-) multiple(*)
- bit shift (<<) (>>)

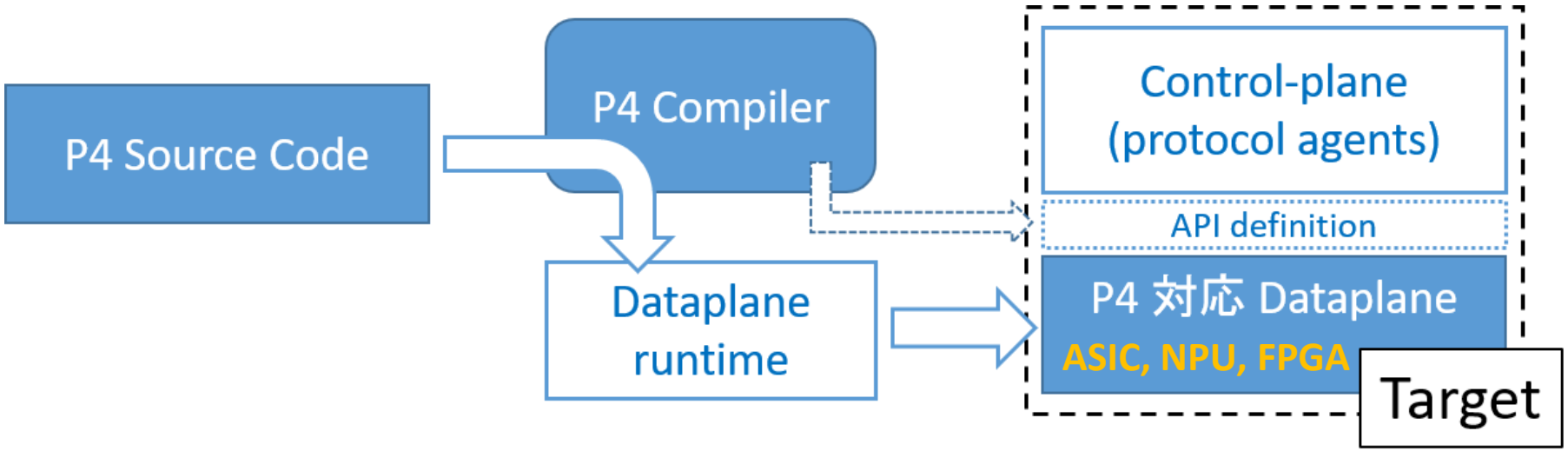
→ ヘッダ解析（Parse） → テーブル参照（Match） → 変更・転送（Action） →

P4 … パケット処理に特化したプログラミング言語

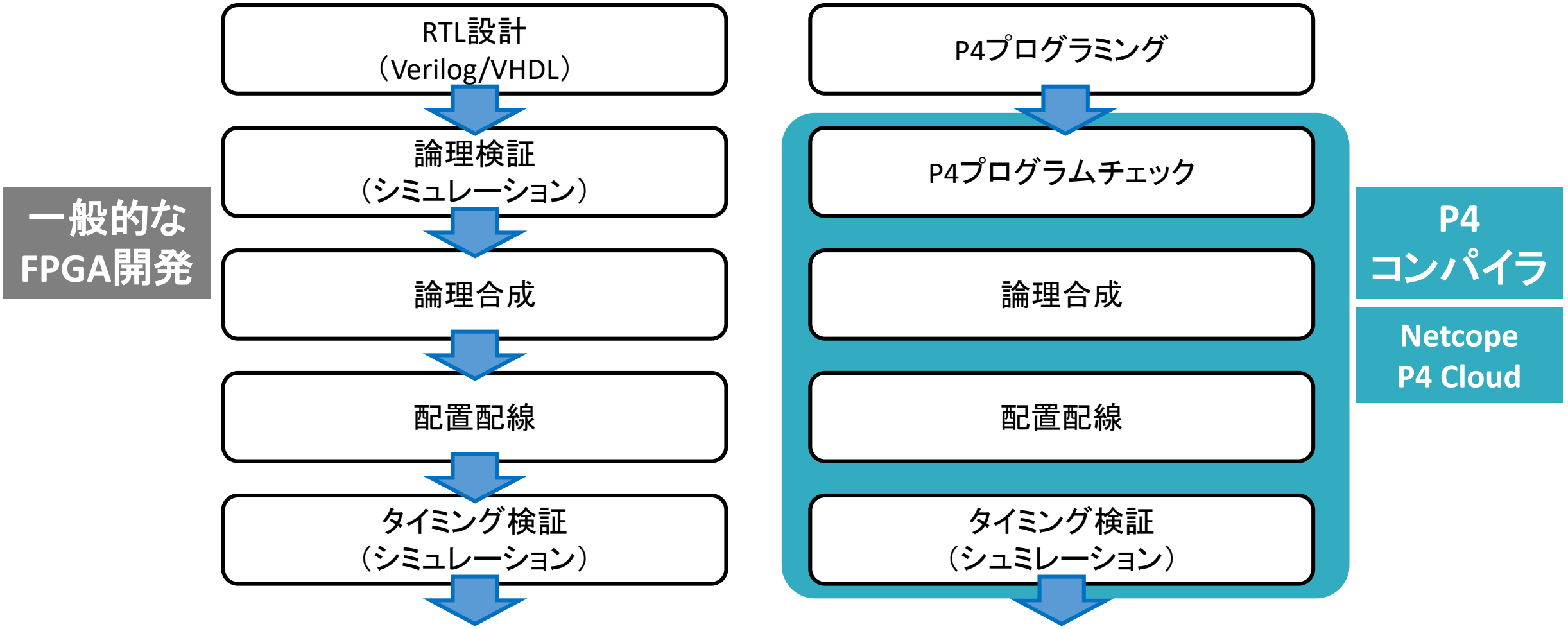
"Programming Protocol-Independent Packet Processors"

P4 Source Code	パケット処理パイプラインの定義 パーサーやテーブル、アクション、など
P4 Compiler	P4をTarget上で実行可能な形式にコンパイル Target毎に提供される
Target (P4対応Dataplane)	P4 Dataplane runtime に従いパケットを処理 Hardware: ASIC, NPU, FPGA | Software: CPU

https://p4.org/



P4を用いたFPGAの開発フロー



P4を用いた独自パケット処理の実装方法の解説は後半で …

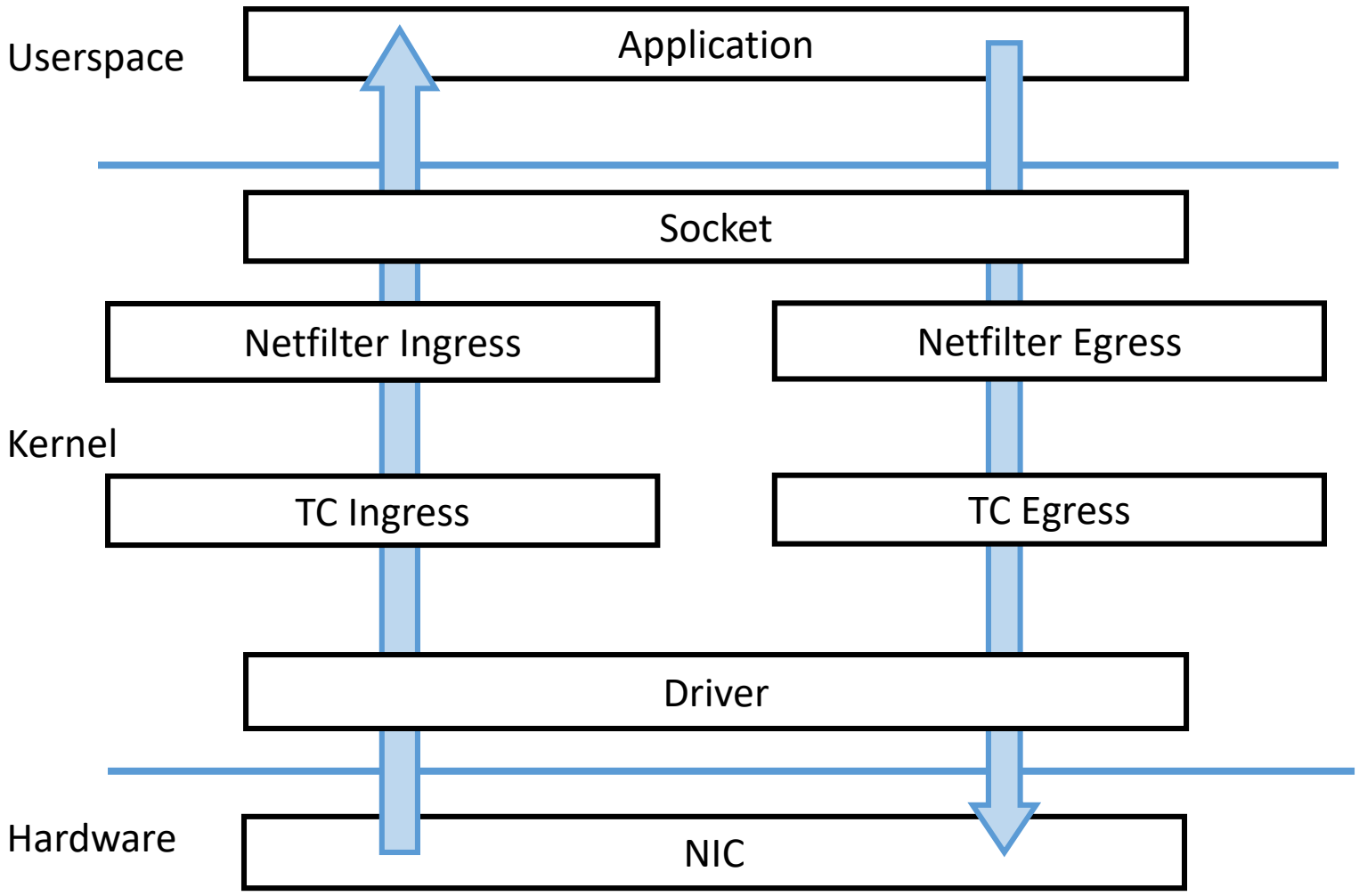
ソフトウェア実装によるパケット処理

ソフトウェア処理

- ソケット通信
- カーネルモジュール
- XDP（Express Data Path）
- DPDK（Data Plane Development Kit）

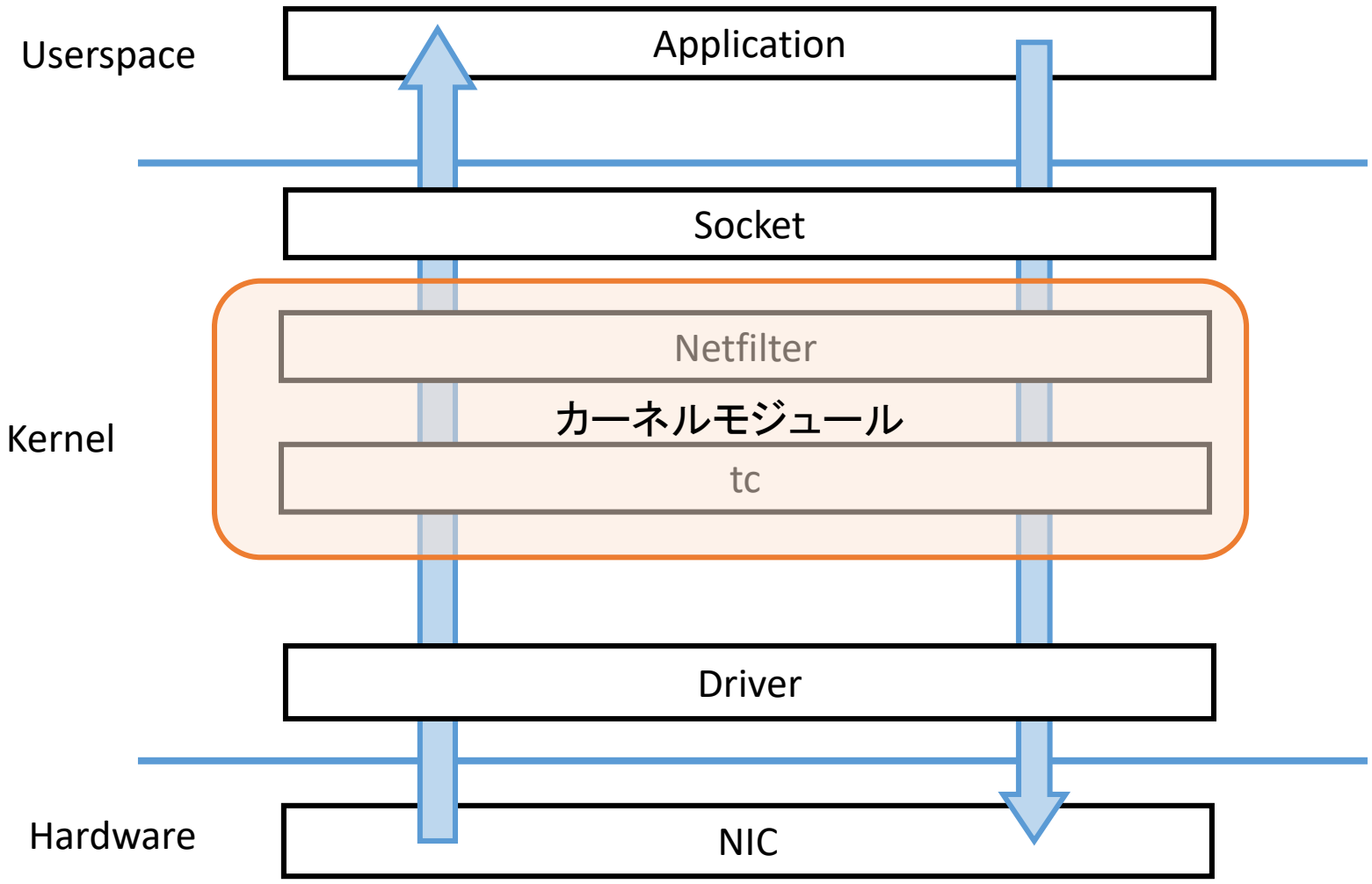
ソケット通信

- 普通のアプリケーションで通信したい場合はこの方式
- パケット送受信時にsyscallが必要で、Linuxカーネル内でも様々な処理を行っているため、他の方式と比べると性能は低い
- tun/tapなどを使ってトンネリングするアプリケーションを実装することも可能



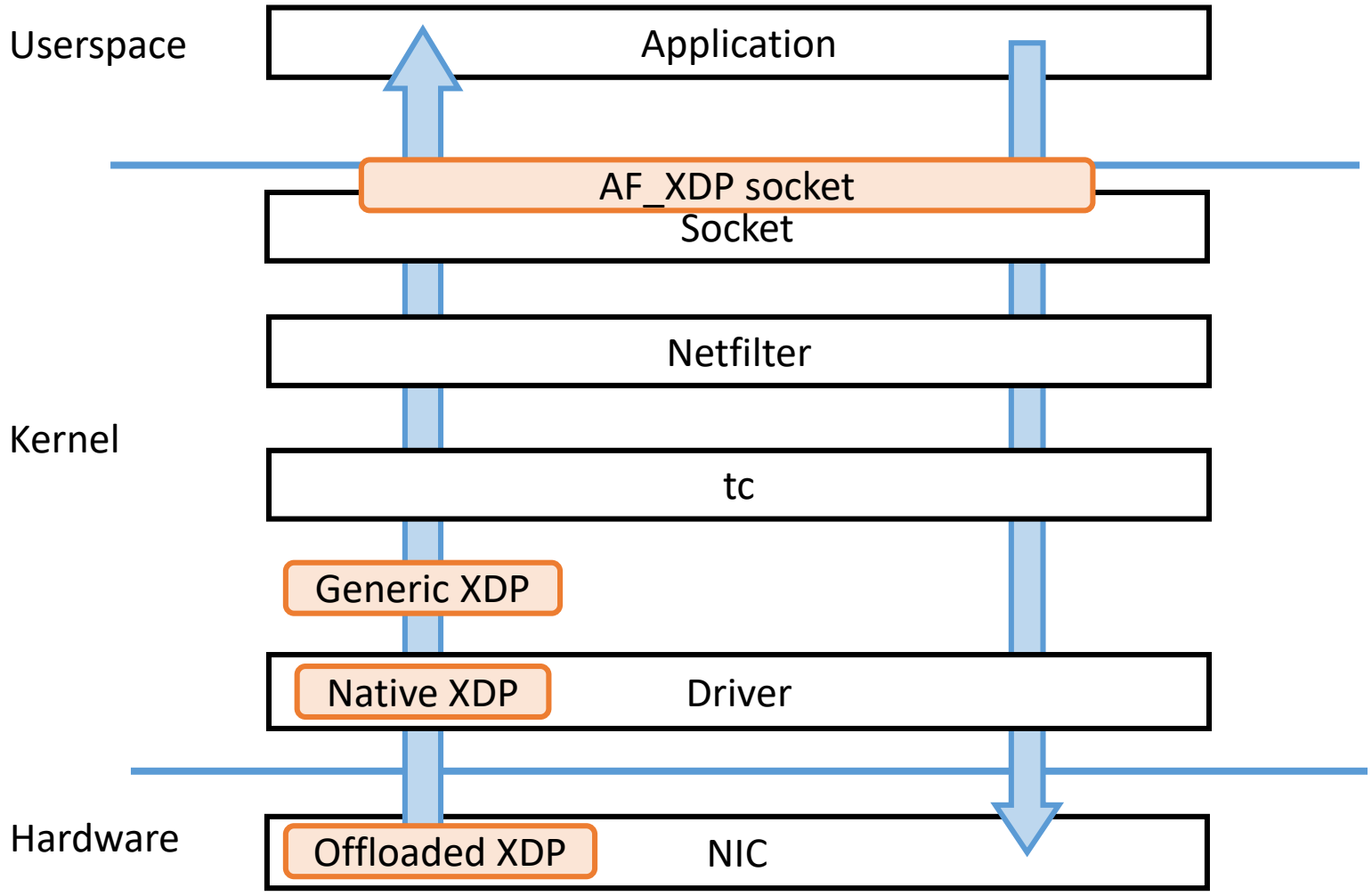
カーネルモジュール

- 新しいプロトコルやネットワークの機能をLinux自体に機能追加したいときに使う
- Netfilterやtcもカーネルモジュール
- 開発、運用の難易度が高く、実装をミスった場合、カーネル毎落ちるため危険
- ユーザスペースのアプリケーションと比較すると高性能



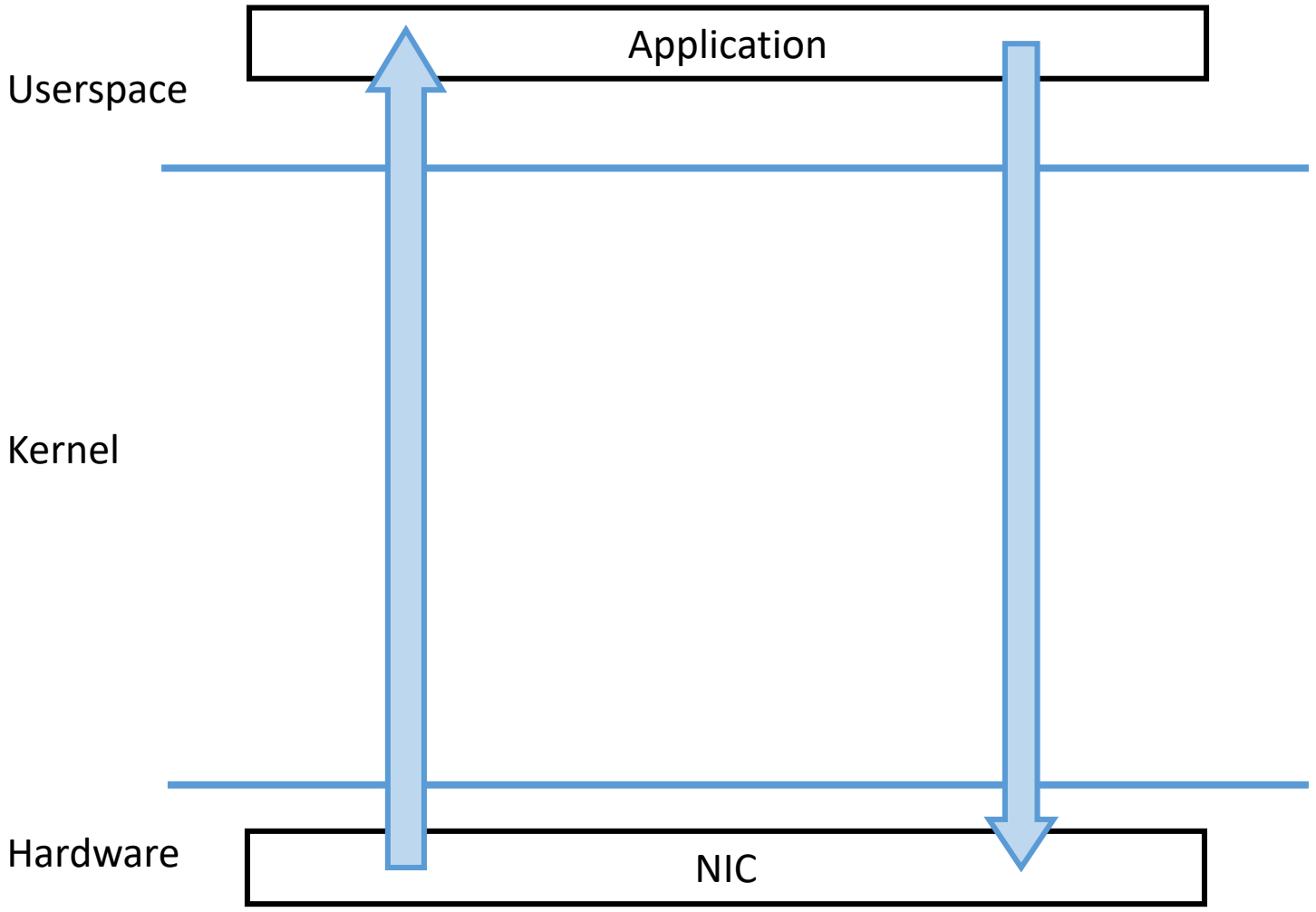
XDP

- 高速にパケットを処理したいが、Linuxの機能を使ったり、ユーザスペースのアプリケーションと同居させたいときに使う
- ロード時にVerifierによるチェックがあるためカーネルモジュールよりは安全
- 対応NIC（ドライバ）が限られている
  - 15種類のドライバが対応（5.4時点）
  - XDP_SETUP_PROGでdriversフォルダをgrepするとわかる
- ハードウェアオフロードに対応しているのはNetronomeのNICのみ（5.4時点）
- 使用例
  - bpfilter
  - OvS-AF_XDP
  - Cloudflare(DDoS Mitigation)
  - Facebook(Katran)
  - Cilium
  - XFLAG(Static NATなど)
  - LINE(L4LB, SRv6)
  - BBSakura(Mobile Core)



DPDK

- UIO(User space IO)を利用して、直接NICを制御する
- Linuxカーネル内の処理をスルーし、必要な処理だけを実行できるため高速
- 専用のCPUコアを割り当てないといけないため、他のアプリケーションとの同居がしづらい
- 必要な処理は全部DPDKのアプリケーションでやらないといけない
- 対応NICが限られている
- AF_XDPの上で動作することも可能
- 使用例
  - OvS-DPDK
  - NTT(Lagopus)
  - NTT Com(Kamuee)
  - Cisco TRex



ソフトウェア処理

- シングルコア性能は
- DPDK > XDP >>>> カーネルモジュール >> ソケット通信
- というイメージ
- マルチコアではXDPがDPDKより性能が良いデータがある
- 実際の性能は使うハードウェアや作ったプログラムによってかなり異なるので、測ってみないとわかりませんが、
- 10G以上のNICで効率的にパケットを処理したい場合はXDPかDPDKというイメージ

パケット処理実装のまとめ（ハードウェア＆ソフトウェア）

ハードウェアとソフトウェア処理の比較

ハードウェア

専用回路を用いた処理

並列化による高速処理
安定＝遅延・性能の揺らぎが少

限られた範囲でのプログラミング
アプリケーションとは離れた場所
パケット処理専用ハードが必要
（スイッチやFPGA搭載サーバー）

ソフトウェア

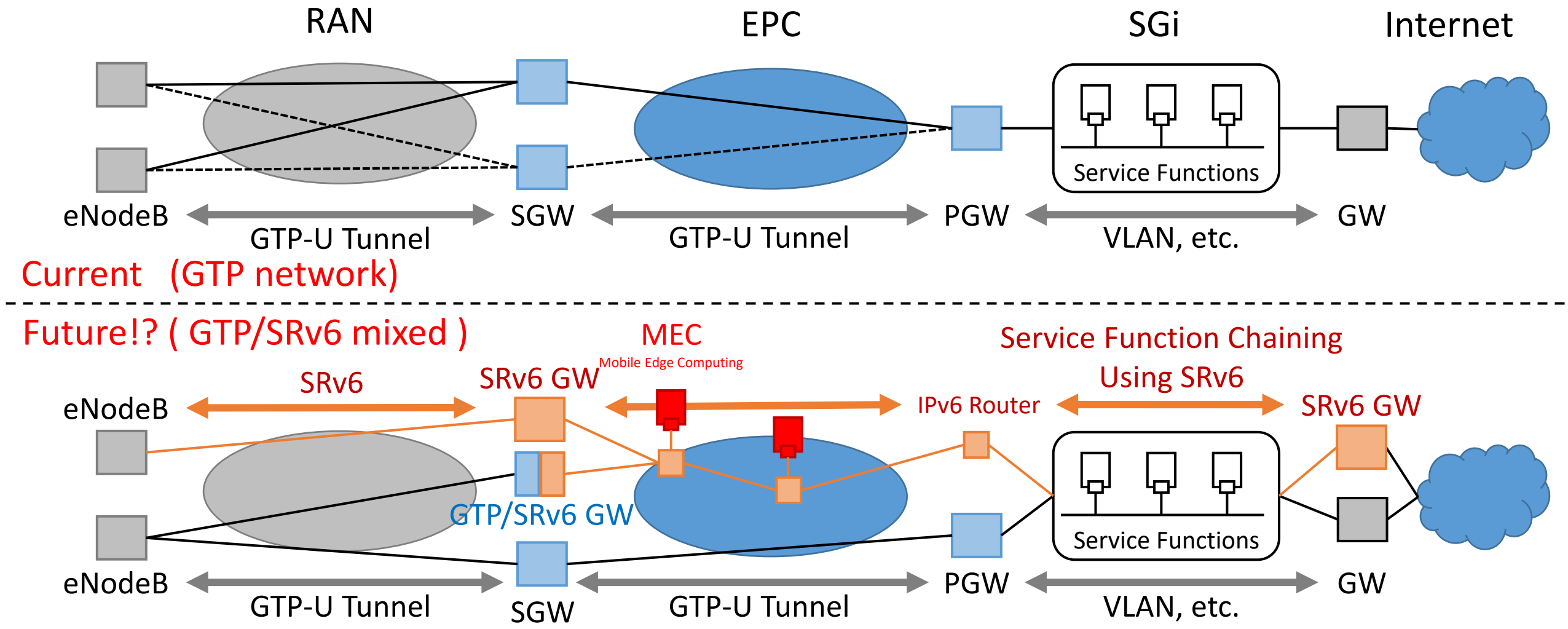
CPUを用いた処理

マルチコアを使った並列処理
開発の難易度は方式によって異なる

DPDK以外は他のアプリケーションとの同居がしやすい
XDP、DPDKでは対応NICが必要だが、大抵の物理サーバや仮想サーバで動作可能

独自パケット処理の実装例

Segment Routing IPv6 for Mobile User Plane



GTP-U Echo ⇔ SRv6 変換ルール（Internet-Draft）

https://datatracker.ietf.org/doc/draft-ietf-dmm-srv6-mobile-uplane/

DMM Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

S. Matsushima
SoftBank
C. Filsfils
M. Kohno
P. Camarillo
Cisco Systems, Inc.
D. Voyer
Bell Canada
C. Perkins
Futurewei
November 4, 2019

Segment Routing IPv6 for Mobile User Plane
draft-ietf-dmm-srv6-mobile-uplane-07

Abstract

This document shows the applicability of SRv6 (Segment Routing IPv6) to the user-plane of mobile networks. The network programming nature of SRv6 accomplish mobile user-plane functions in a simple manner. The statelessness of SRv6 and its ability to control both service layer path and underlying transport can be beneficial to the mobile user-plane, providing flexibility, end-to-end network slicing and SLA control for various applications. This document describes the SRv6 mobile user plane.

https://datatracker.ietf.org/doc/draft-murakami-dmm-user-plane-message-encoding/

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 7, 2020

T. Murakami, Ed.
Arrcus, Inc
S. Matsushima
SoftBank
K. Ebisawa
Toyota Motor Corporation
P. Garvia
R. Shekhar
Cisco Systems, Inc.
November 4, 2019

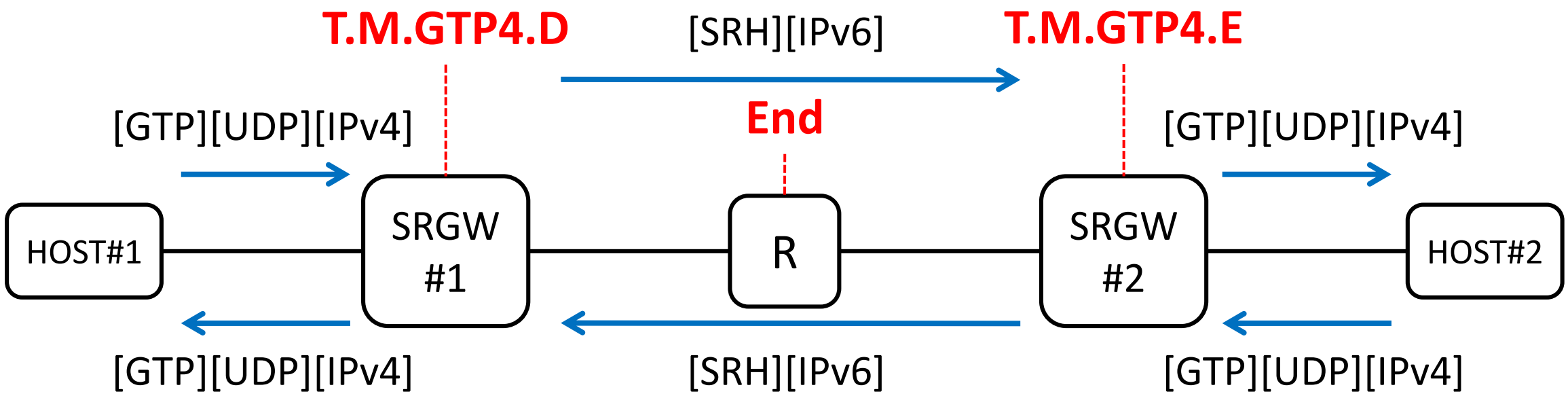
User Plane Message Encoding
draft-murakami-dmm-user-plane-message-encoding-00

Abstract

This document defines the encoding of User Plane messages into Segment Routing Header (SRH). The SRH carries the User Plane messages over SRv6 Network.

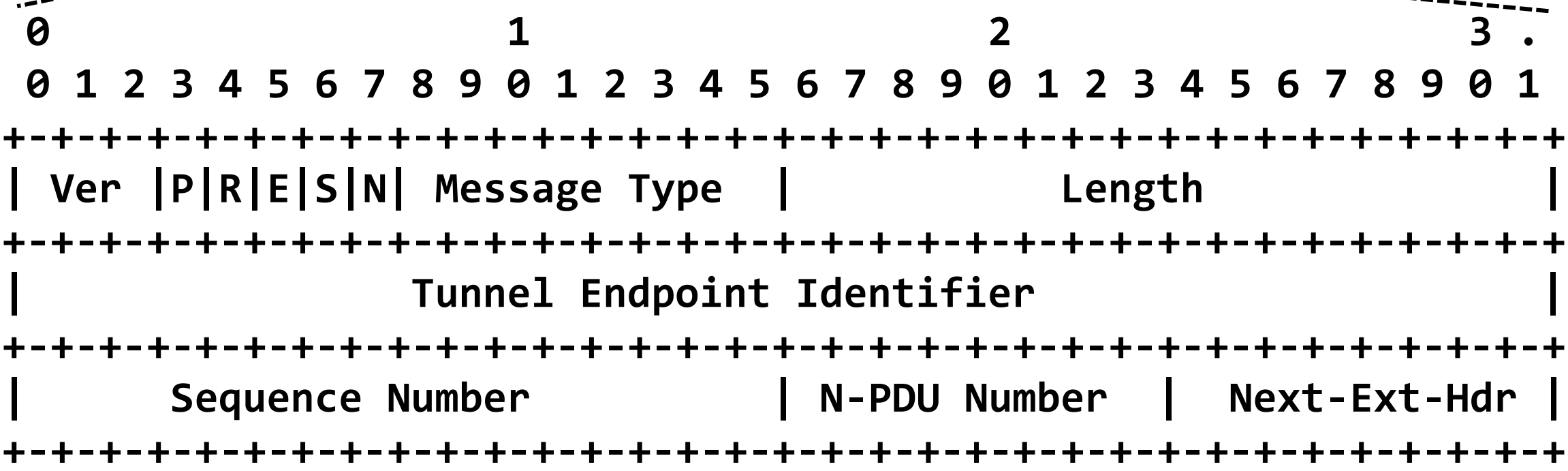
注：Sequence番号は次のバージョンで追加予定

GTP-U Echo ⇔ SRv6 変換処理



GTP-U Header format

[GTP][UDP][IPv4]



GTP-U Echo Request / Reply Message

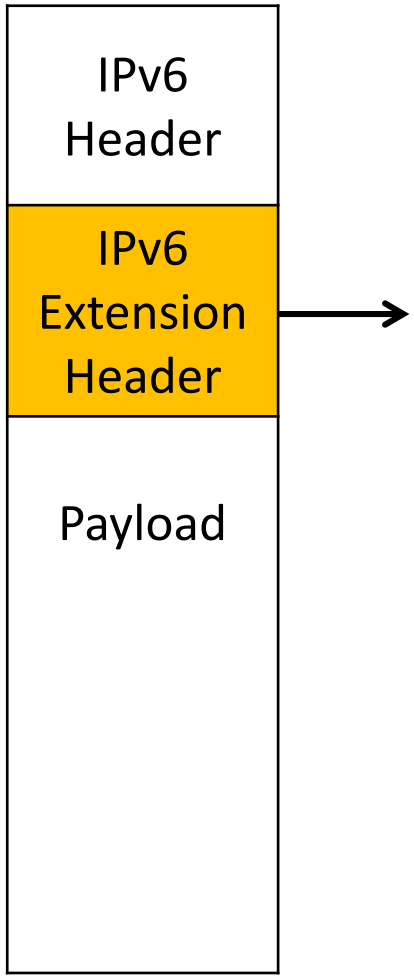
GTP Echo request

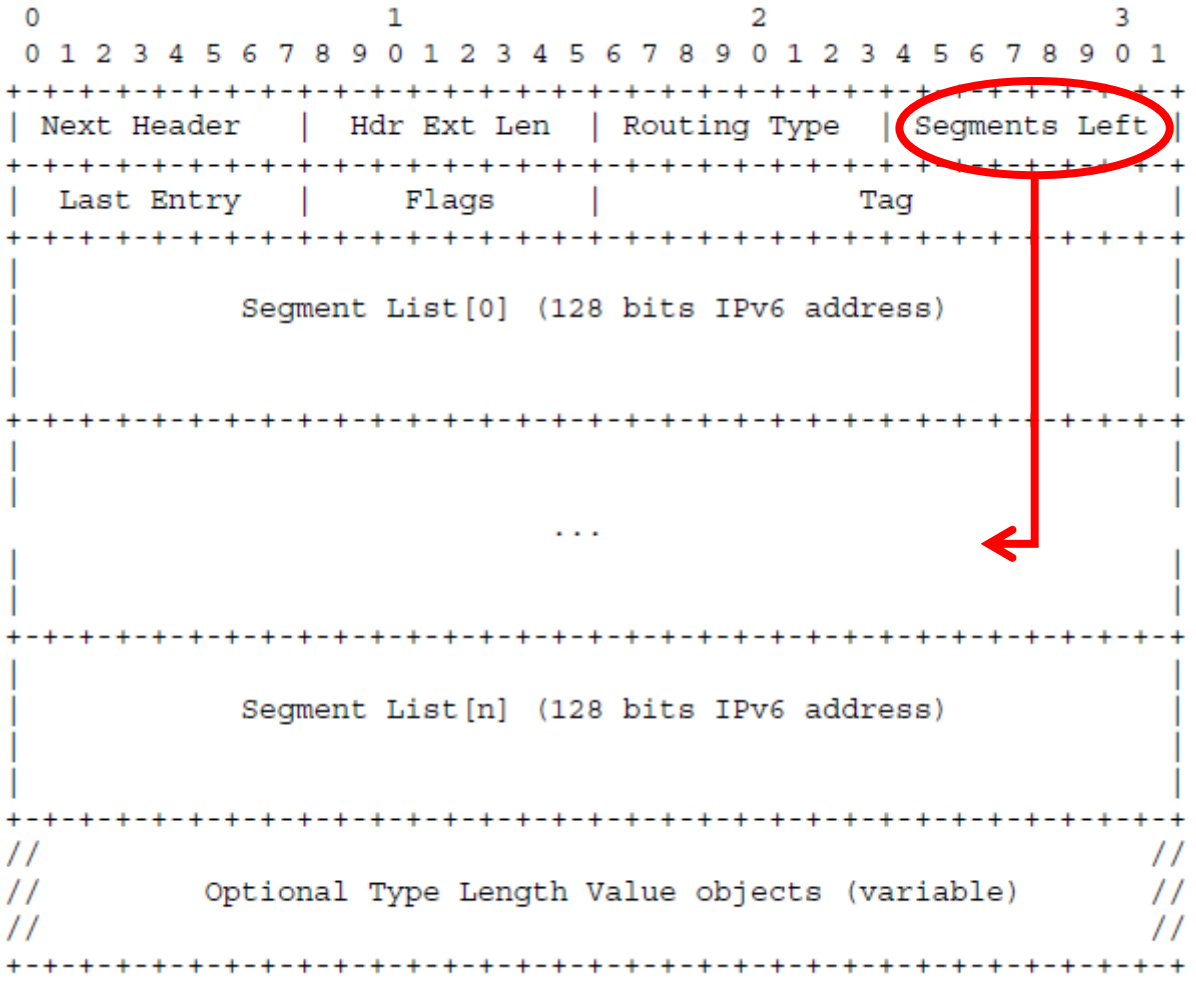
```
> Ethernet II, Src: 32:b6:b9:6c:54:4b (32:b6:b9:6c:54:4b), Dst:
> Internet Protocol Version 4, Src: 172.20.0.1, Dst: 172.20.0.2
> User Datagram Protocol, Src Port: 2123, Dst Port: 2123
v GPRS Tunneling Protocol
  v Flags: 0x32
      001. .... = Version: GTP release 99 version (1)
      ...1 .... = Protocol type: GTP (1)
      .... 0... = Reserved: 0
      .... .0.. = Is Next Extension Header present?: No
      .... ..1. = Is Sequence Number present?: Yes
      .... ...0 = Is N-PDU number present?: No
    Message Type: Echo request (0x01)
    Length: 4
    TEID: 0x00000000 (0)
    Sequence number: 0x0002 (2)
```

GTP Echo response

```
> Ethernet II, Src: c6:fc:d3:27:73:05 (c6:fc:d3:27:73:05), Dst:
> Internet Protocol Version 4, Src: 172.20.0.2, Dst: 172.20.0.1
> User Datagram Protocol, Src Port: 2123, Dst Port: 2123
v GPRS Tunneling Protocol
  v Flags: 0x32
      001. .... = Version: GTP release 99 version (1)
      ...1 .... = Protocol type: GTP (1)
      .... 0... = Reserved: 0
      .... .0.. = Is Next Extension Header present?: No
      .... ..1. = Is Sequence Number present?: Yes
      .... ...0 = Is N-PDU number present?: No
    Message Type: Echo response (0x02)
    Length: 4
    TEID: 0x00000000 (0)
    Sequence number: 0x0002 (2)
```

SRv6 (SRH) Header format





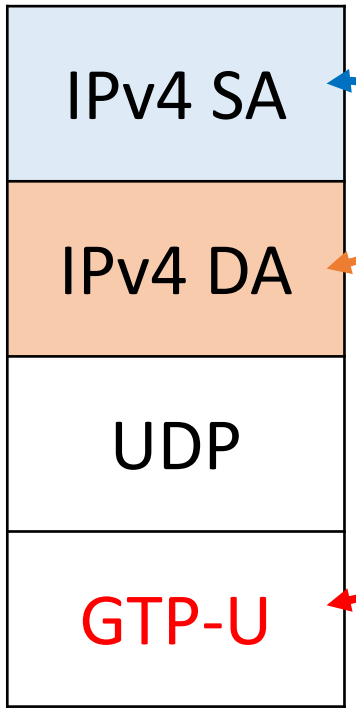
- Routing Type
  - 4 (Segment Routing)
- Segments Left
  - Index to the next segment in the Segment List
  - Decremented on Endpoint node
- Last Entry
  - Index to the first segment in the Segment List
- Segment List
  - Encoded starting from the last segment of the path (Segment List [0] contains the last segment)

Reference: draft-ietf-6man-segment-routing-header

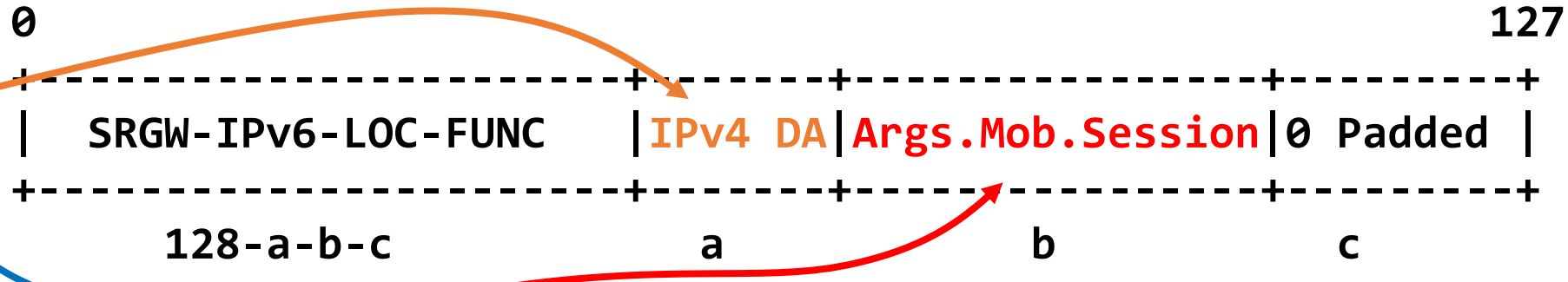
GTP IPv4 Addr ⇔ SRv6 Segment ID & Src Address

T.M.GTP4.D →

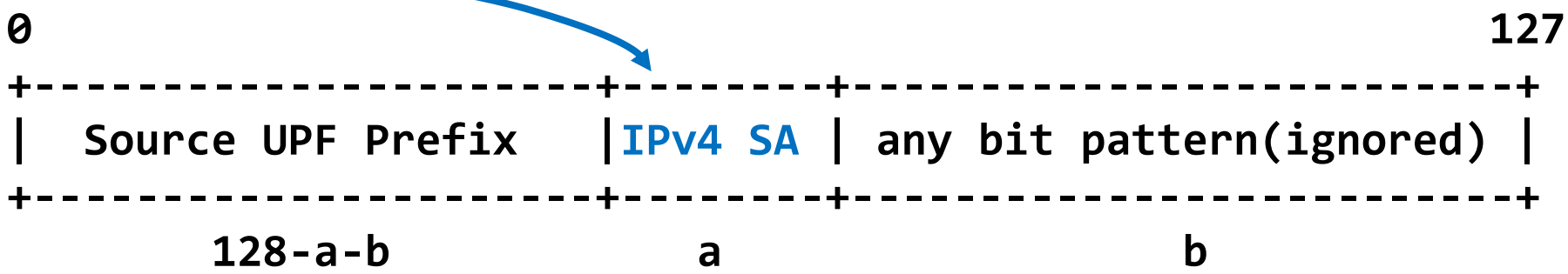
← **T.M.GTP4.E**



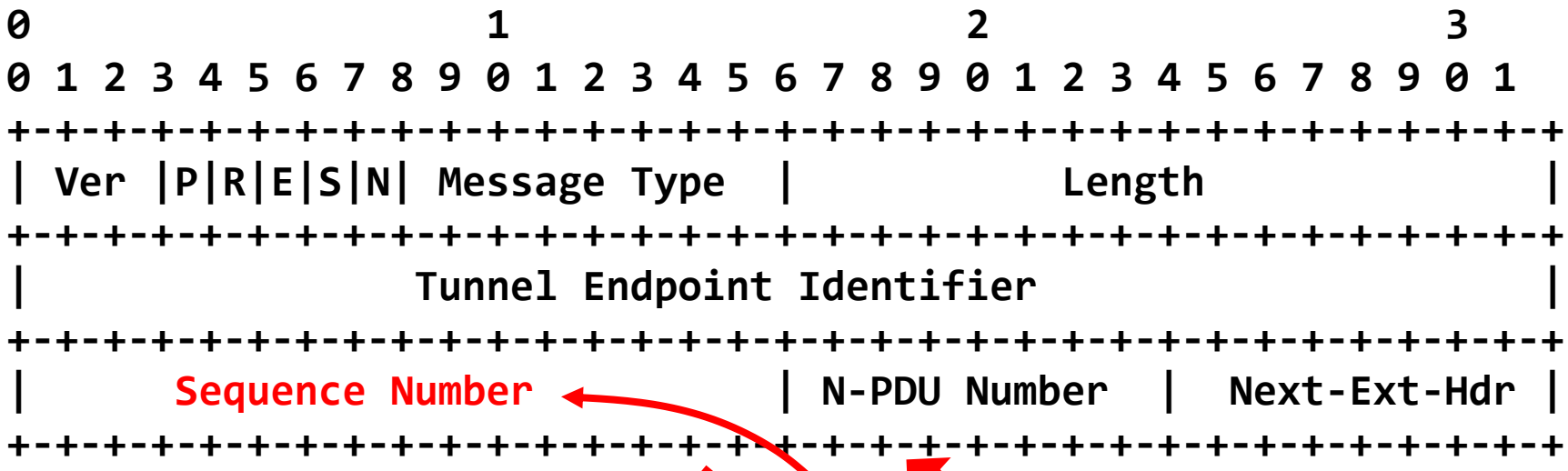
Destination Segment ID (Last Destination IPv6 Addr)



Source IPv6 Address



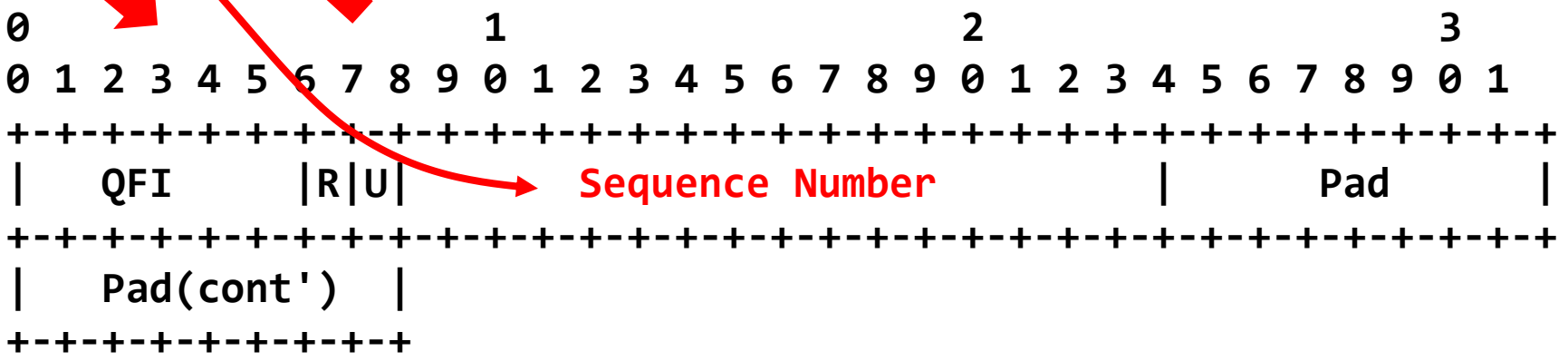
GTP-U Header ⇔ SRv6 Segment ID (Args.Mob.Session)



Echo Request/Response の
TEID==0 のため、Seq を記憶

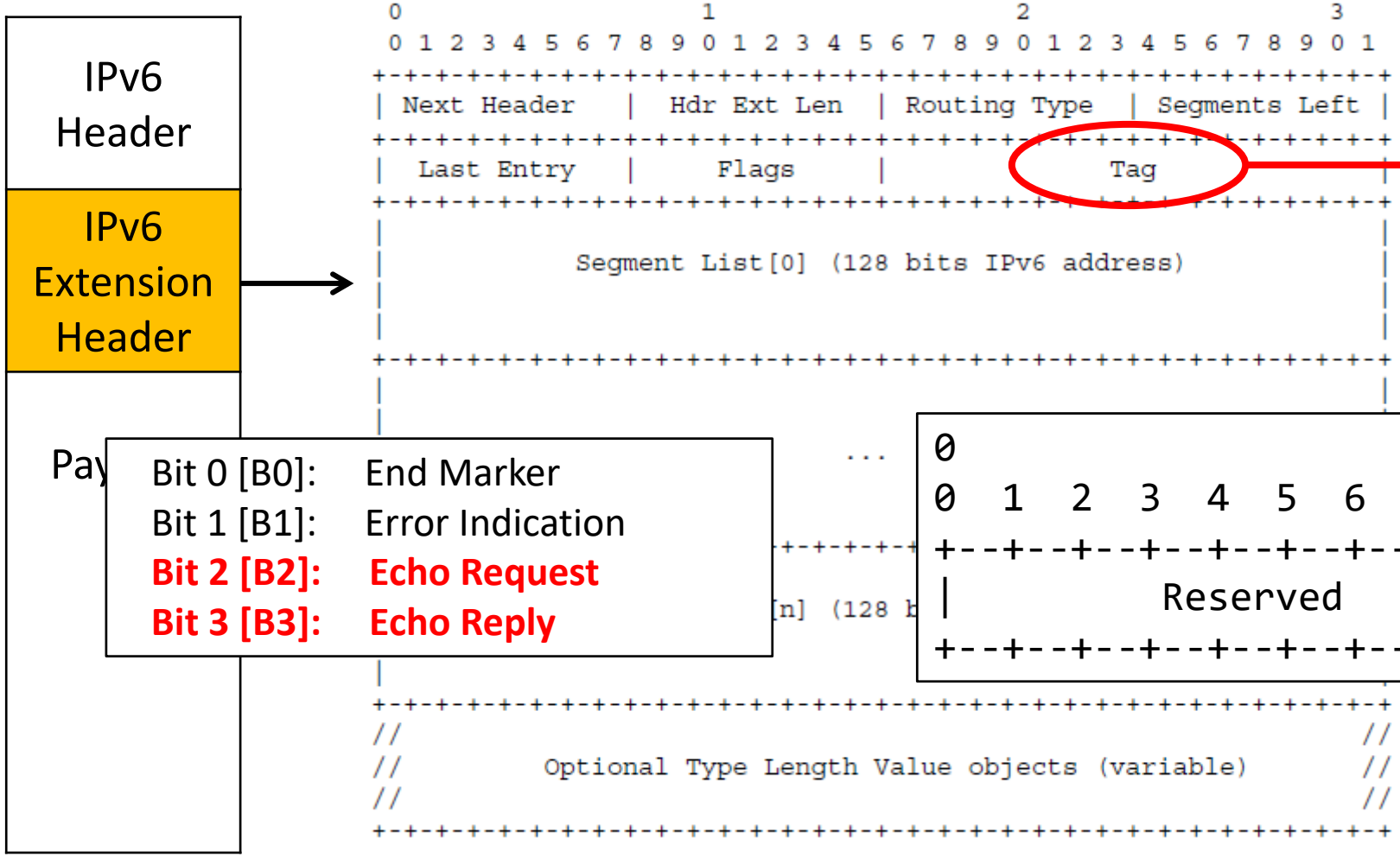
T.M.GTP4.D

T.M.GTP4.E



Args.Mob.Session format for Echo Request, Echo Reply and Error Indication

GTP-U Message Type をエンコード



- Routing Type
  - 4 (Segment Routing)
- Segments Left
  - Index to the next segment in the Segment List
  - Decremented on Endpoint node
- Last Entry

0 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6

| Reserved | B3 | B2 | B1 | B0 |

segment of the path (Segment List [0] contains the last segment)

Bit 0 [B0]: End Marker
Bit 1 [B1]: Error Indication
Bit 2 [B2]: Echo Request
Bit 3 [B3]: Echo Reply

Reference: draft-ietf-6man-segment-routing-header