SONiC + P4によるマルチテナント SRv6サービスチェイニングの実現

桑田　斉

hitoshi.kuwata.gt@apresiasystems.co.jp

JANOG46

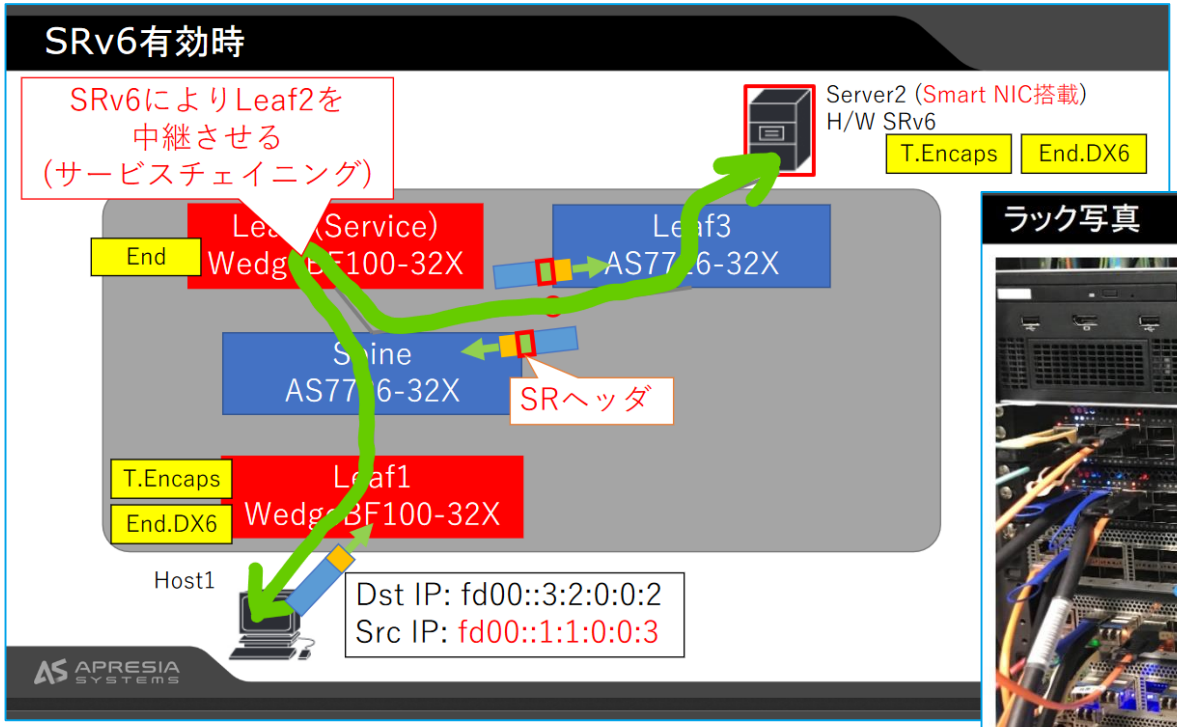
自己紹介

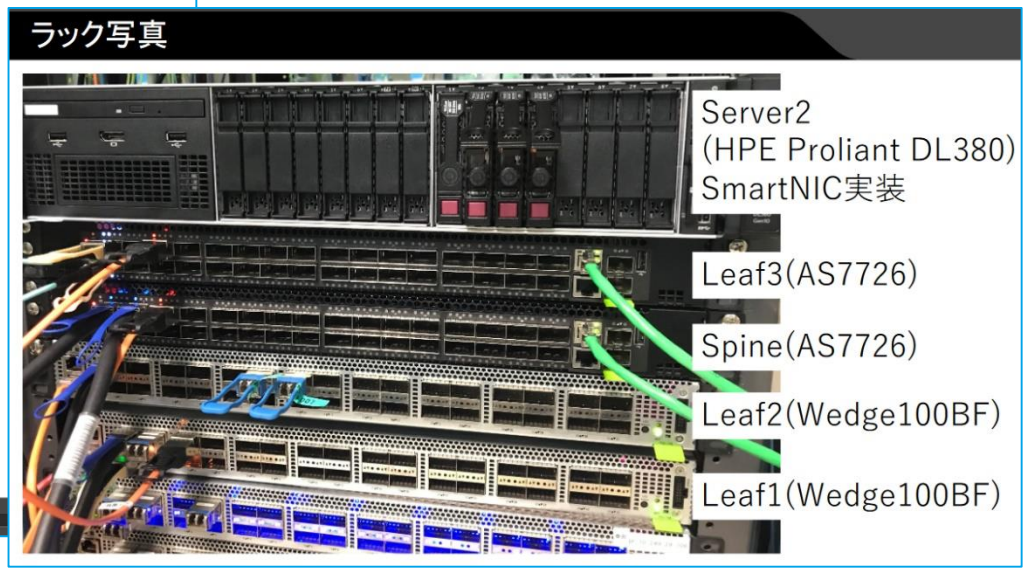
- APRESIA Systems 株式会社　次世代技術本部　技術開発部
  - 桑田　斉（くわた　ひとし）
- 2003年 日立電線株式会社入社
  - 以来、APRESIAスイッチシリーズの製品ソフトウェア開発に従事
  - 2016年にAPRESIA Systems株式会社の独立により現職
  - 最近の活動
    - ホワイトボックススイッチ向けのNOS（SONiCなど）
    - P4プログラミング（日本P4ユーザ会※ 運営委員）

      ※https://p4users.org/

前回のJANOG45の発表

- SONiCとP4ハードウェアにて実現するSRv6サービスチェイニング（発表：熊谷）
  - SONiCとCumulusの組み合わせで、IP CLOSファブリックを構築
  - P4にてSRv6を実現





https://www.janog.gr.jp/meeting/janog45/program/srv6sfc

今回の発表の目的

- マルチテナントを意識するなど、運用に近いSRv6ネットワークを構成すること
- クラウドネイティブに求められるネットワーク要件を視野に入れること
  - EVPN/VXLANによるマルチテナントネットワークとの違い

※クラウドネイティブ定義

- https://github.com/cncf/toc/blob/master/DEFINITION.md#%E6%97%A5%E6%9C%AC%E8%AA%9E%E7%89%88

> クラウドネイティブ技術は、パブリッククラウド、プライベートクラウド、ハイブリッドクラウドなどの近代的でダイナミックな環境において、スケーラブルなアプリケーションを構築および実行するための能力を組織にもたらします。 このアプローチの代表例に、コンテナ、サービスメッシュ、マイクロサービス、イミュータブルインフラストラクチャ、および宣言型APIがあります。
>
> これらの手法により、回復性、管理力、および可観測性のある疎結合システムが実現します。 これらを堅牢な自動化と組み合わせることで、エンジニアはインパクトのある変更を最小限の労力で頻繁かつ予測どおりに行うことができます。
>
> Cloud Native Computing Foundationは、オープンソースでベンダー中立プロジェクトのエコシステムを育成・維持して、このパラダイムの採用を促進したいと考えてます。 私たちは最先端のパターンを民主化し、これらのイノベーションを誰もが利用できるようにします。

◆マイクロサービス

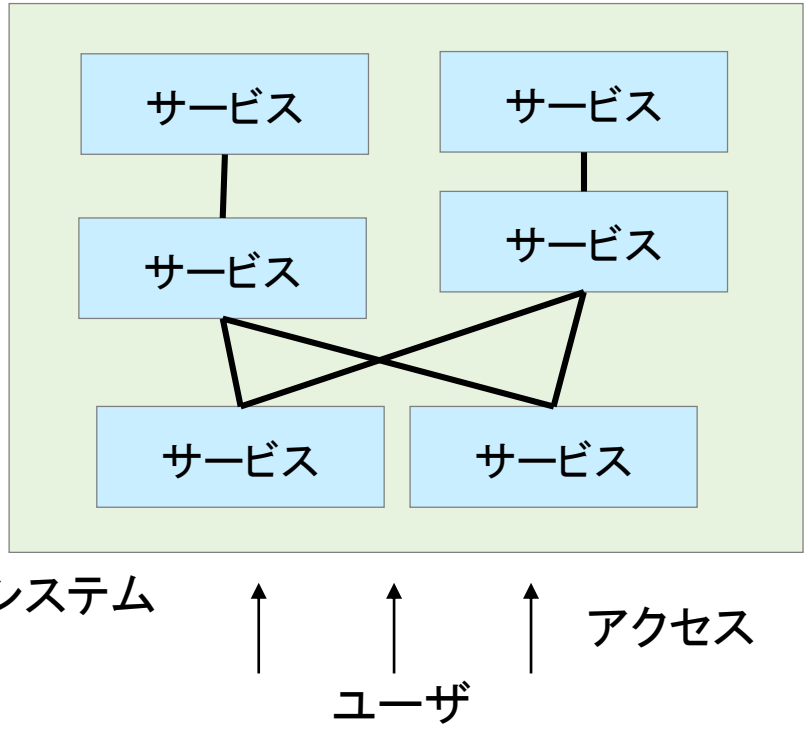
◇ 分割されたサービスを組み合わせてシステムを構築

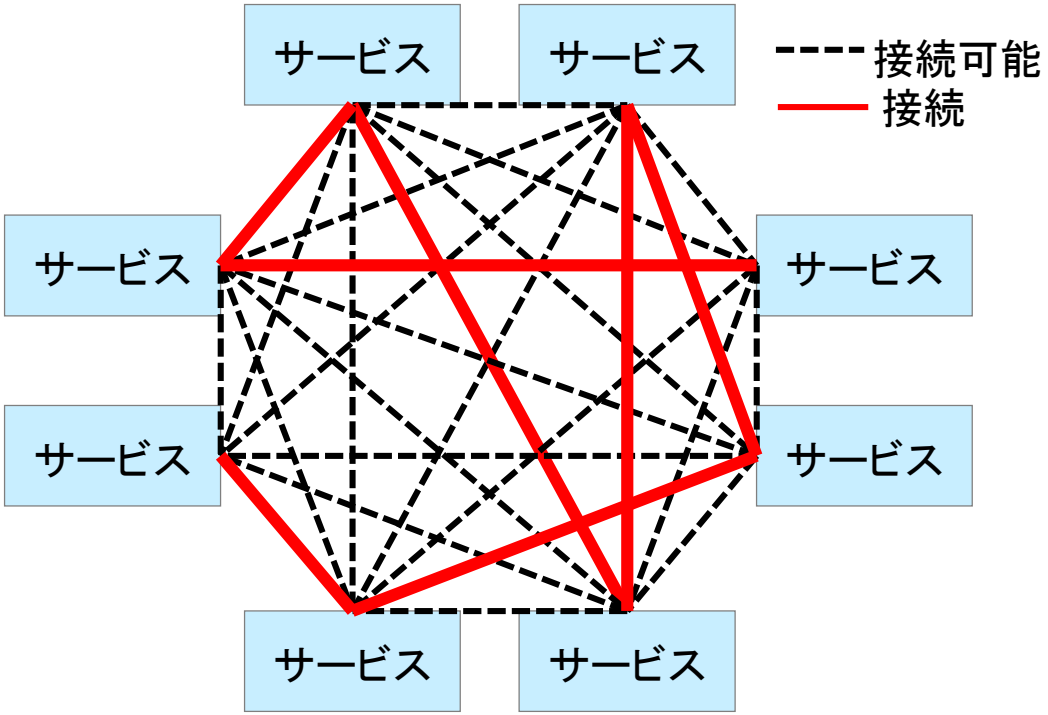
→ 各サービスの影響範囲を限定可能

◆サービスメッシュ

◇ 分割されたサービスを任意のパスで接続可能

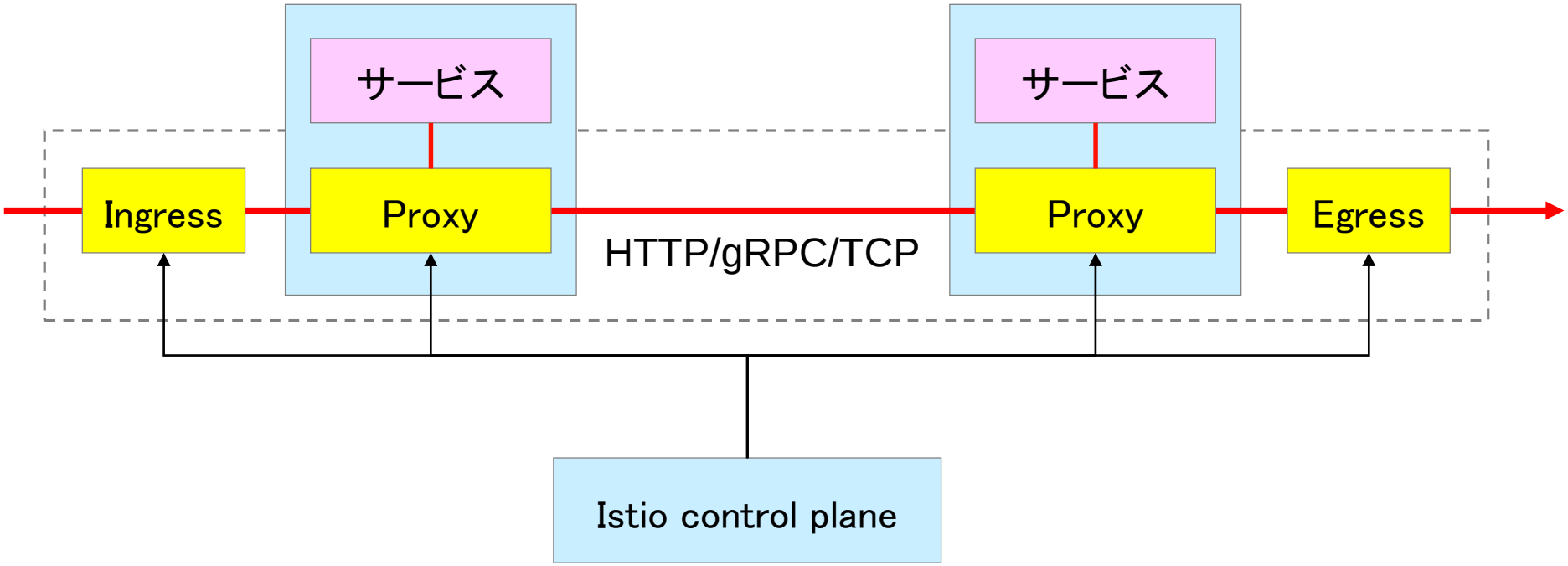
→ サービスは接続形態を意識しない





サービスメッシュの実現方法

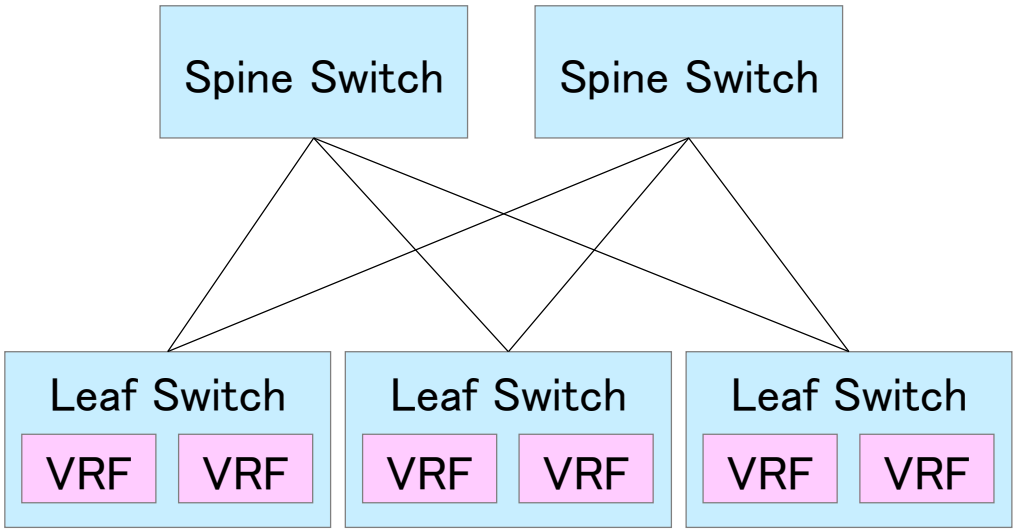
- ソフトウェアレイヤーにてサービスメッシュを実現する場合（例：Istio）
  - サービス間の通信の接続をproxyにて肩代わりして、メッシュ接続を実現



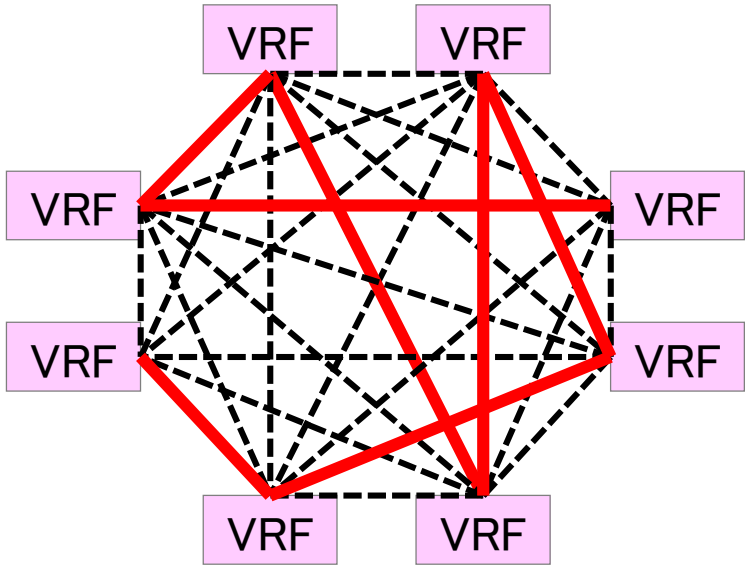
参照： https://istio.io/latest/docs/concepts/security/

◆ マイクロサービスと同様に、VRFのメッシュ接続をSRv6サービスチェインにて実現

物理ネットワーク
IP CLOSファブリック



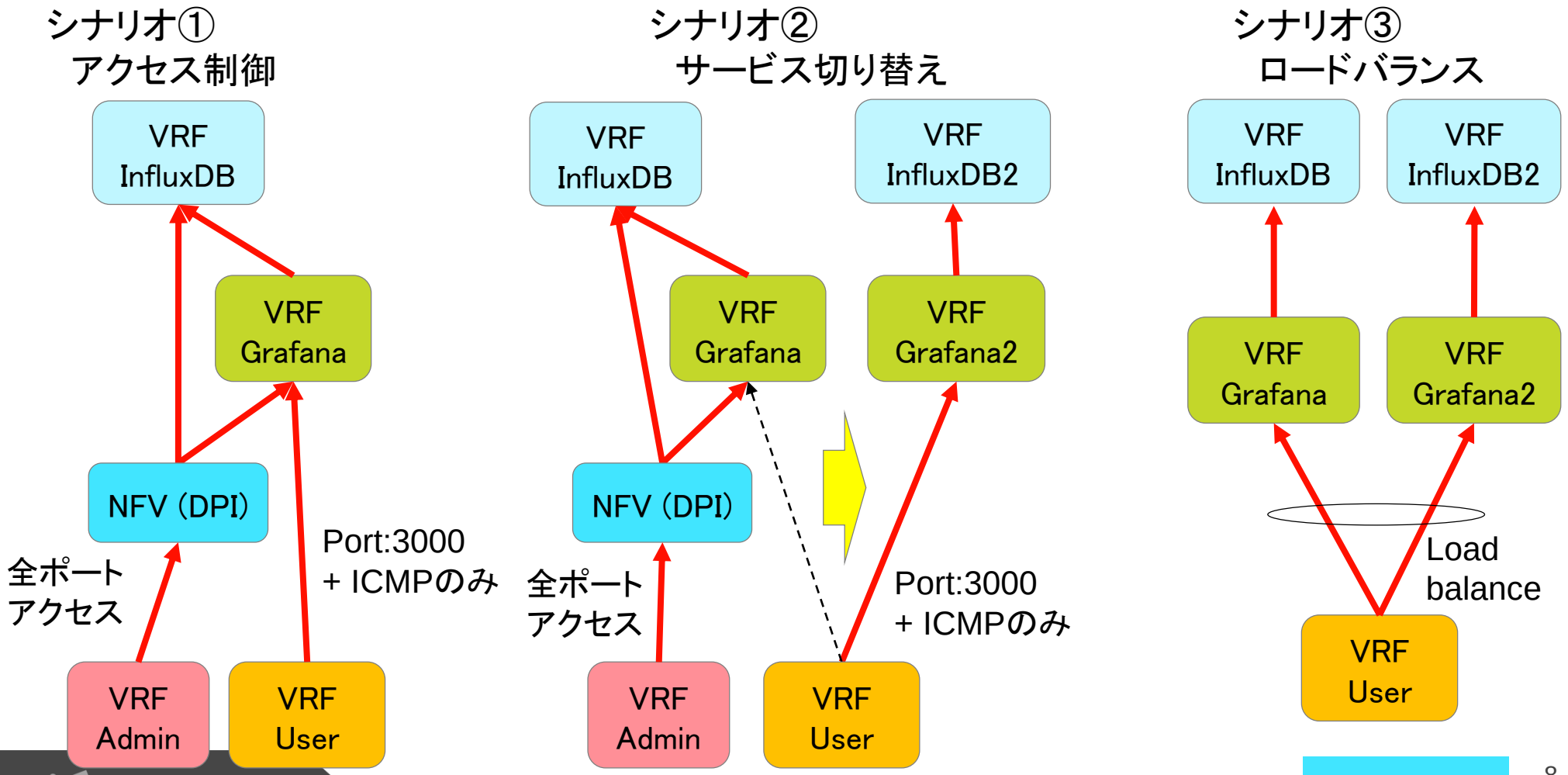
論理ネットワーク
メッシュ接続



SRv6サービス
チェインで接続

SRv6にて構築するネットワークのデモシナリオ

- サービスの例として、InfluxDBとGrafanaによるデータの可視化環境を使用
- アクセス側は、Admin(管理者)とUser(ユーザ)に対して、異なるセキュリティポリシーを適用



デモの物理環境

10.202.1.11/24 NFV (DPI) 10.203.1.11/24

VLAN100
VRF Admin

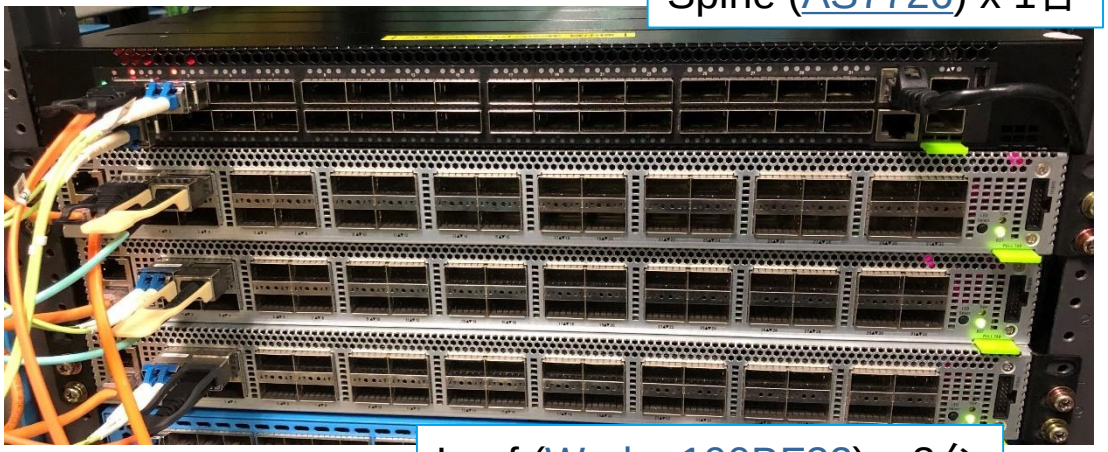
VLAN200
VRF Grafana

Leaf3 (Wedge100BF32)
SONiC + P4 + SRv6

Spine (AS7726)
SONiC on Edgecore or
Cumulus + Broadcom

実験環境外観

Spine (AS7726) x 1台



Leaf (Wedge100BF32) x 3台

Leaf1 (Wedge100BF32)
SONiC + P4 + SRv6

VLAN100
VRF User

VLAN200
VRF Admin

VM
10.11.1.11/24

VM
10.201.1.11/24

Leaf2 (Wedge100BF32)
SONiC + P4 + SRv6

VLAN100
VRF Grafana

VLAN200
VRF InfluxDB

VLAN300
VRF Grafana2

VLAN400
VRF InfluxDB2

VM
10.101.1.11/24

VM
10.102.1.11/24

VM
10.101.1.11/24

VM
10.102.1.11/24

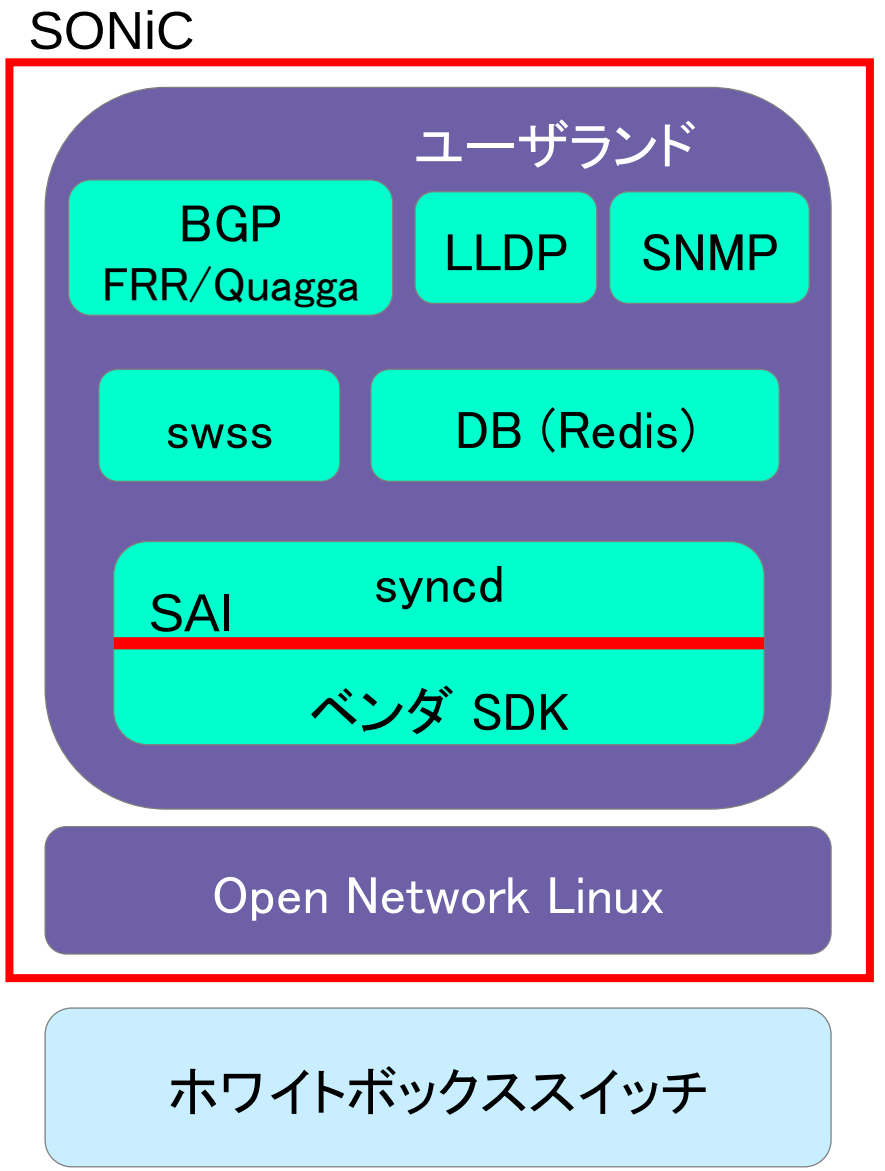
構成要素の説明

- SONiC
- P4
- SRv6

APRESIA®

SONiC（Software for Open Networking in the Cloud）とは？

- ホワイトボックススイッチ用のOSSのNOS
- Microsoftが公開したソースコードが母体
- SAI (Switch Abstraction Interface)を定義し、スイッチチップの差分を隠蔽したことで、マルチベンダ対応を実現
  - Broadcom, Barefoot, Mellanoxなど、複数のチップベンダのスイッチをサポート
- BGPやLLDP、データベースなどのアプリケーションはコンテナ化
  - L3スタックとしてFRRoutingを採用
- BGPを使ってIP CLOSファブリックを構築可能



SONiCの主な機能の拡張状況

◆ IP CLOS Fabric構築に必要な機能が202012ブランチでカバーされる予定

本デモで使用

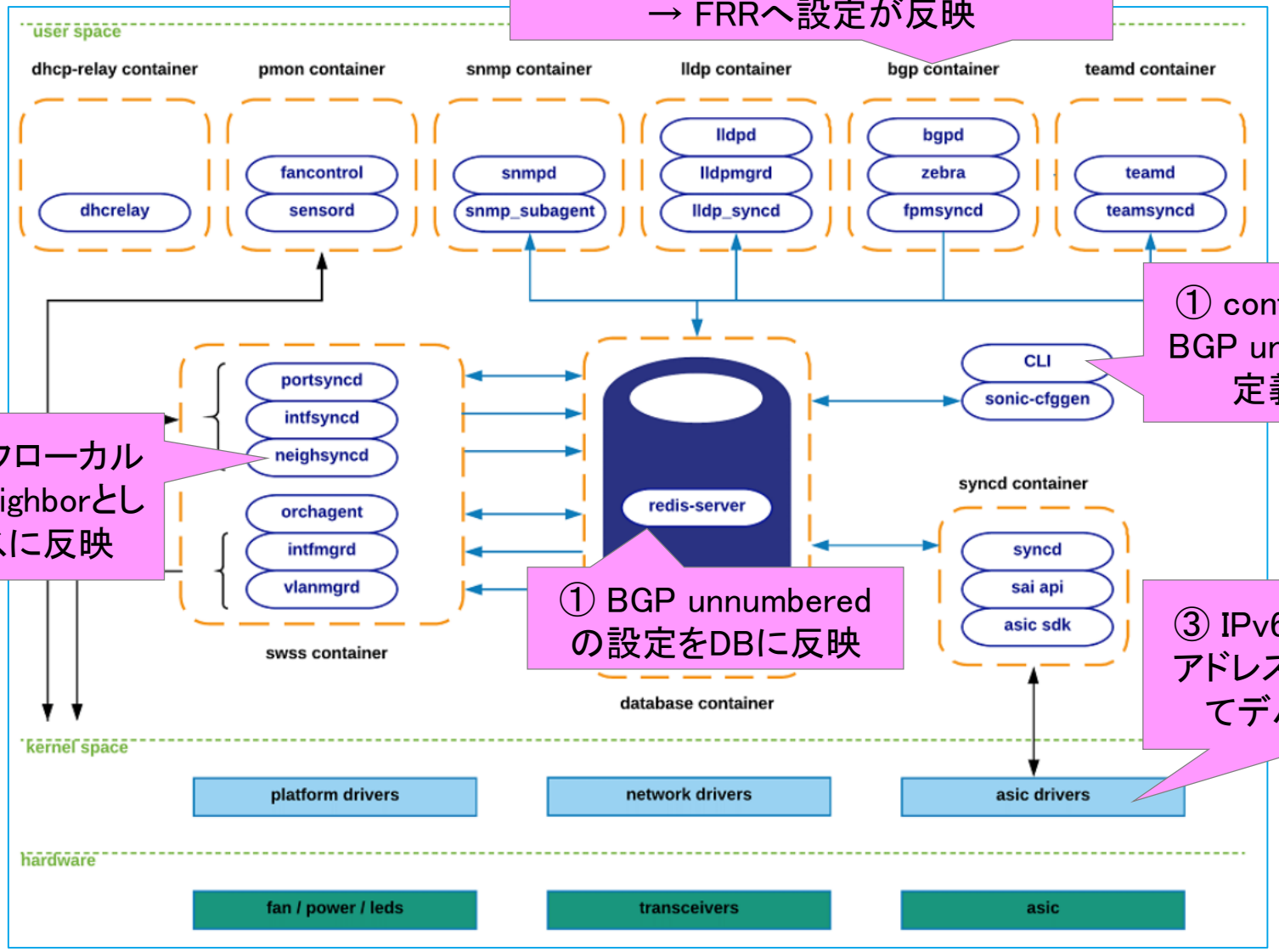
機能	201911 （安定化中）	202006 （開発中）	202012 （次期開発）	当社改造 Barefoot 向け ベース：201911	SONiC on Edgecore Broadcom向け ベース：201911
FRR採用	●	●	●	●	●
BGP EVPN (type 5)	●	●	●	●	●
ZTP	●	●	●	●	●
sFlow	●	●	●	●	●
VRF	●	●	●	●	●
Sub Port	●	●	●	●	●
NAT	●	●	●	●	●
BFD	–	●	●	–	–
EVPN/VXLAN	–	延期	●	–	●
FRR BGP NBI (North Bound Interface)	–	延期	●	–	–
IPv6 Link Local (RFC5549)	–	延期	●	● ※改造	●
BGP Unnumbered	–	延期	●	● ※改造	●
MC-LAG (L2)	–	延期	●	–	●

SRv6サービスチェイニングにて使用

SONiCのロードマップは以下参照
https://github.com/Azure/SONiC/wiki/Sonic-Roadmap-Planning

BGP unnumbered 改造内容

② BGPコンテナのテンプレート変更
→ FRRへ設定が反映



① config_db.jsonに
BGP unnumberedの
定義を追加

③ IPv6リンクローカル
アドレスをneighborとし
てデバイスに反映

① BGP unnumbered
の設定をDBに反映

③ IPv6リンクローカル
アドレスをneighborとし
てデバイスに反映

https://github.com/Azure/SONiC/wiki/Architecture

BGP unnumbered 改造内容① config_db.jsonへの追加

従来のconfig_db.json

```
"INTERFACE": {
        "Ethernet0|10.0.0.0/31": {},
        "Ethernet4|10.0.0.2/31": {},
        "Ethernet8|10.0.0.4/31": {},
 },
"BGP_NEIGHBOR": {
        "10.0.0.1": {
            "asn": "65200",
            "holdtime": "180",
            "keepalive": "60",
            "local_addr": "10.0.0.0",
            "nhopself": 0,
            "rrclient": 0
        },
        "10.0.0.3": {
            "asn": "65200",
            "holdtime": "180",
            "keepalive": "60",
            "local_addr": "10.0.0.2",
            "nhopself": 0,
            "rrclient": 0
        },
        "10.0.0.5": {
            "asn": "65200",
            "holdtime": "180",
            "keepalive": "60",
            "local_addr": "10.0.0.4",
            "nhopself": 0,
            "rrclient": 0
        },
```

BGP unnumberedを使った場合、以下に簡略可能

```
"INTERFACE": {
     "Ethernet0": {},
     "Ethernet4": {},
     "Ethernet8": {}
 },
"BGP_UNNUMBERED": {
     "FABRIC": {
          "interfaces": [
              "Ethernet0",
              "Ethernet4",
              "Ethernet8 “
          ]
      }
 },
```

※当社独自の改造のため、
SONiCコミュニティにて開発中の
記述仕様とは異なります

Redis(DB)への反映

```
admin@Spine1:~$ redis-cli
127.0.0.1:6379> select 4
OK
127.0.0.1:6379[4]> keys *
 1) "LOOPBACK_INTERFACE|Loopback0|11.11.11.11/32"
 2) "FEATURE|telemetry"
 3) "CONTAINER_FEATURE|sflow"
 4) "BGP_UNNUMBERED|FABRIC"
(省略)

127.0.0.1:6379[4]> HVALS "BGP_UNNUMBERED|FABRIC"
1) "Ethernet0,Ethernet4,Ethernet8"
127.0.0.1:6379[4]>
```

BGP unnumbered 改造内容②

- BGPコンテナ内のテンプレートに以下を追加
  - /usr/share/sonic/templates/bgpd/bgpd.main.conf.j2
- 右のようにFRRの設定に展開される

```
{% block bgp_unnumbered %}
{% if BGP_UNNUMBERED %}
{% for peer_name, peer_interfaces in BGP_UNNUMBERED.iteritems() %}
  neighbor {{ peer_name }} peer-group
  neighbor {{ peer_name }} remote-as external
  address-family ipv4 unicast
    neighbor {{ peer_name }} activate
    neighbor FABRIC soft-reconfiguration inbound
  exit-address-family
  address-family ipv6 unicast
    neighbor {{ peer_name }} activate
    neighbor FABRIC soft-reconfiguration inbound
  exit-address-family
{% for peer_interface in peer_interfaces['interfaces'] %}
  neighbor {{ peer_interface }} interface peer-group FABRIC
{% endfor %}
{% endfor %}
{% endif %}
{% endblock bgp_unnumbered %}
```

```
Leaf1# show running-config
(中略)
router bgp 65110
 bgp router-id 1.1.1.1
 bgp log-neighbor-changes
 no bgp default ipv4-unicast
 bgp graceful-restart restart-time 240
 bgp graceful-restart
 bgp graceful-restart preserve-fw-state
 bgp bestpath as-path multipath-relax
 neighbor FABRIC peer-group
 neighbor FABRIC remote-as external
 neighbor Ethernet0 interface peer-group FABRIC
 neighbor Ethernet4 interface peer-group FABRIC
 !
 address-family ipv4 unicast
  network 1.1.1.1/32
  neighbor FABRIC activate
  neighbor FABRIC soft-reconfiguration inbound
  neighbor FABRIC route-map FROM_BGP_PEER_V4 in
  neighbor FABRIC route-map TO_BGP_PEER_V4 out
  maximum-paths 64
 exit-address-family
 !
 address-family ipv6 unicast
  network fd00:ffff:0:1::/64
  neighbor FABRIC activate
  neighbor FABRIC soft-reconfiguration inbound
  neighbor FABRIC route-map FROM_BGP_PEER_V6 in
  neighbor FABRIC route-map TO_BGP_PEER_V6 out
  maximum-paths 64
 exit-address-family!
```

BGP unnumbered 改造内容③ IPv6リンクローカルのneighbor登録

```
diff --git a/neighsyncd/neighsync.cpp b/neighsyncd/neighsync.cpp
index 1af9445..e269bfa 100644
--- a/neighsyncd/neighsync.cpp
+++ b/neighsyncd/neighsync.cpp
@@ -73,9 +73,12 @@ void NeighSync::onMsg(int nlmsg_type, struct nl_object *obj)
     key+= ":";

     nl_addr2str(rtnl_neigh_get_dst(neigh), ipStr, MAX_ADDR_SIZE);
-    /* Ignore IPv6 link-local addresses as neighbors */
-    if (family == IPV6_NAME && IN6_IS_ADDR_LINKLOCAL(nl_addr_get_binary_addr(rtnl_neigh_get_dst(neigh))))
-        return;
+    if (family == IPV6_NAME && IN6_IS_ADDR_LINKLOCAL(nl_addr_get_binary_addr(rtnl_neigh_get_dst(neigh)))) {
+            if (key.find("Ethernet") == string::npos)  {
+            return;
+        }
+    }
+
     /* Ignore IPv6 multicast link-local addresses as neighbors */
     if (family == IPV6_NAME && IN6_IS_ADDR_MC_LINKLOCAL(nl_addr_get_binary_addr(rtnl_neigh_get_dst(neigh))))
         return;
```

Link localであってもEthernetインタフェースの場合は次の処理に進む

上記の修正によって、Linuxが持っているNeighbor情報がデバイスドライバにも反映される

```
bf_switch:0> show neighbor all
  ================================================================================
  oid      handle  dest_ip                  mac_address          nexthop_handle
  ================================================================================
  1        rif.5   0xfe80::290:fbff:fe00:2 00:90:FB:00:00:02    nexthop.3
  2        rif.6   0xfe80::290:fbff:fe00:3 00:90:FB:00:00:03    nexthop.4
  3        rif.5   169.254.0.1             00:90:FB:00:00:02    nexthop.5
  4        rif.3   169.254.0.1             00:90:FB:00:00:01    nexthop.6
  5        rif.3   0xfe80::290:fbff:fe00:1 00:90:FB:00:00:01    nexthop.7
  6        rif.6   169.254.0.1             00:90:FB:00:00:03    nexthop.8
bf_switch:0>
```

- P4: Programming Protocol-independent Packet Processors
  - パケットのパイプライン処理（パーサー、マッチアクションテーブル、デパーサー）を記述するためのプログラミング言語
  - Barefoot Tofinoチップ（スイッチ用）、Netcope P4コンパイラ（FPGA NIC用)など、P4でプログラミングしたデータプレーンをハードウェアにて動作させることが可能

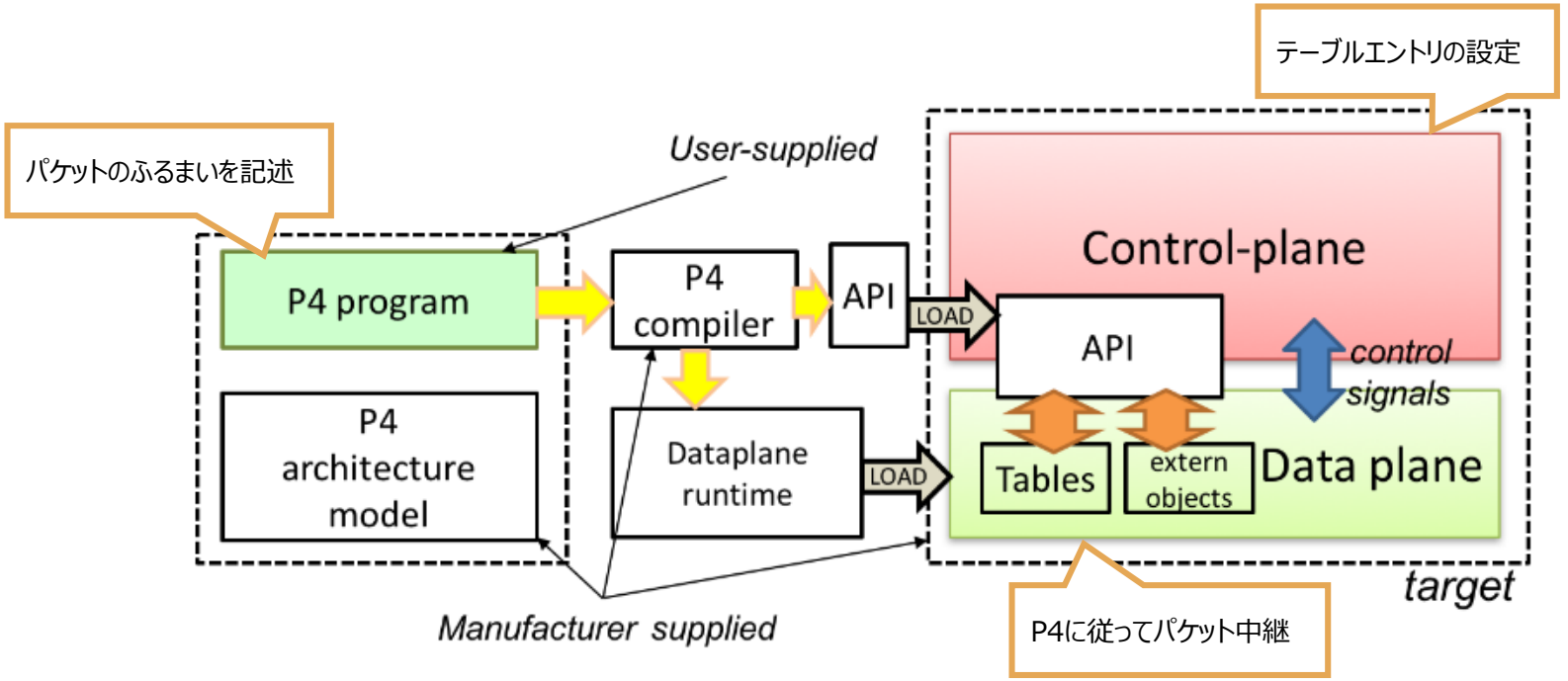
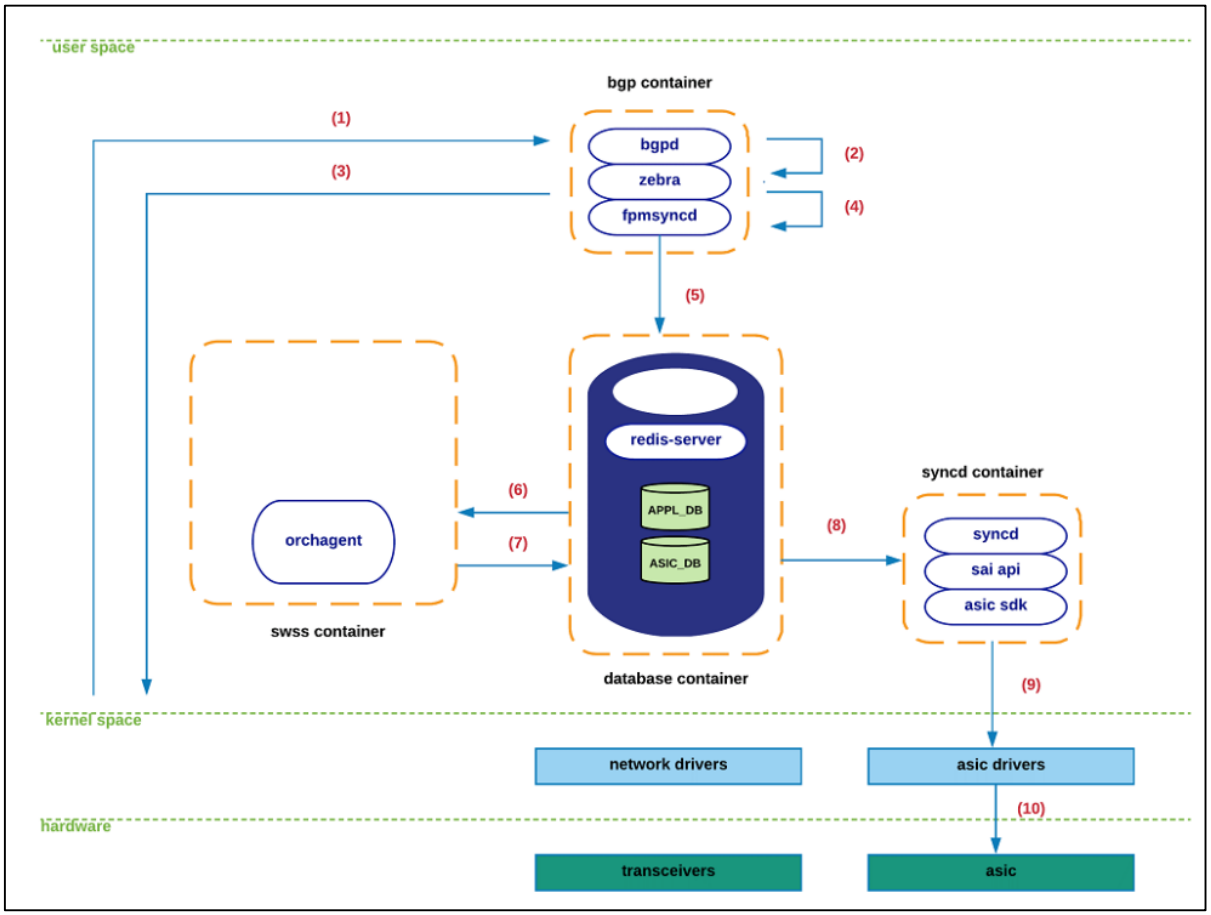


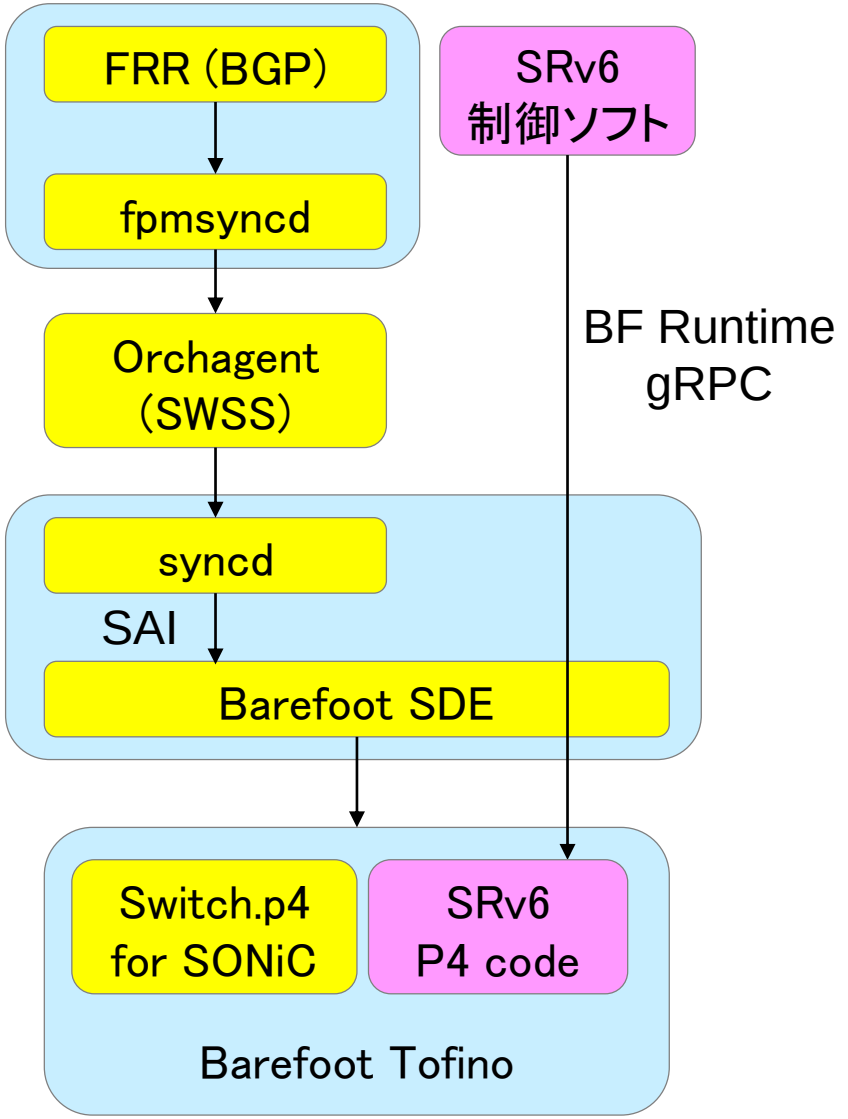
Figure 2. Programming a target with P4.

https://p4.org/p4-spec/docs/P4-16-v1.2.1.pdf

SONiC Architecture

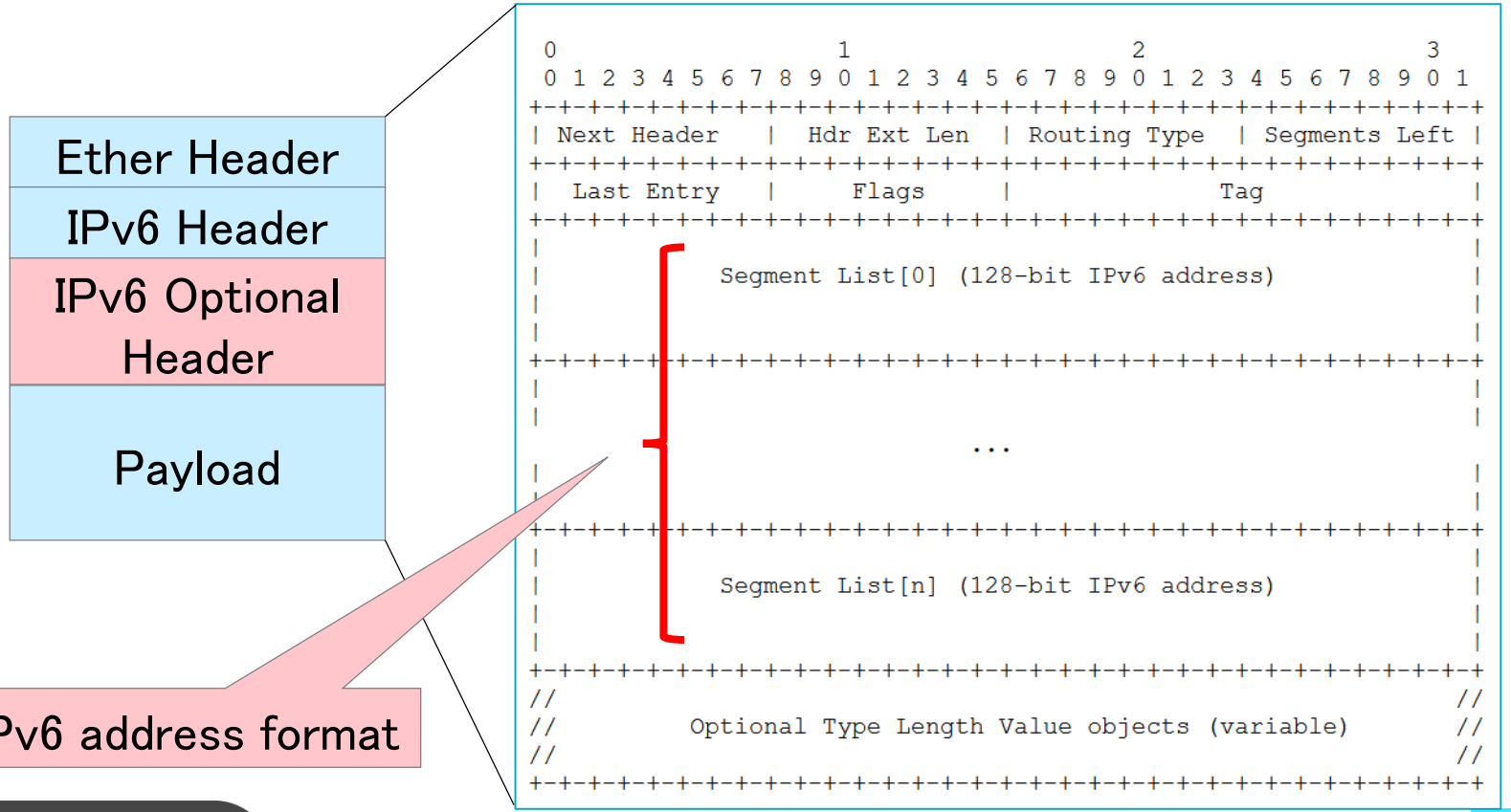
https://github.com/Azure/SONiC/wiki/Architecture





SRv6 header format

- IPv6版のSegment Routing
  - SID (Segment ID)をIPv6アドレスにて表記
  - 参照: IETF RFC8754: IPv6 Segment Routing Header (SRH)
    - https://tools.ietf.org/html/rfc8754

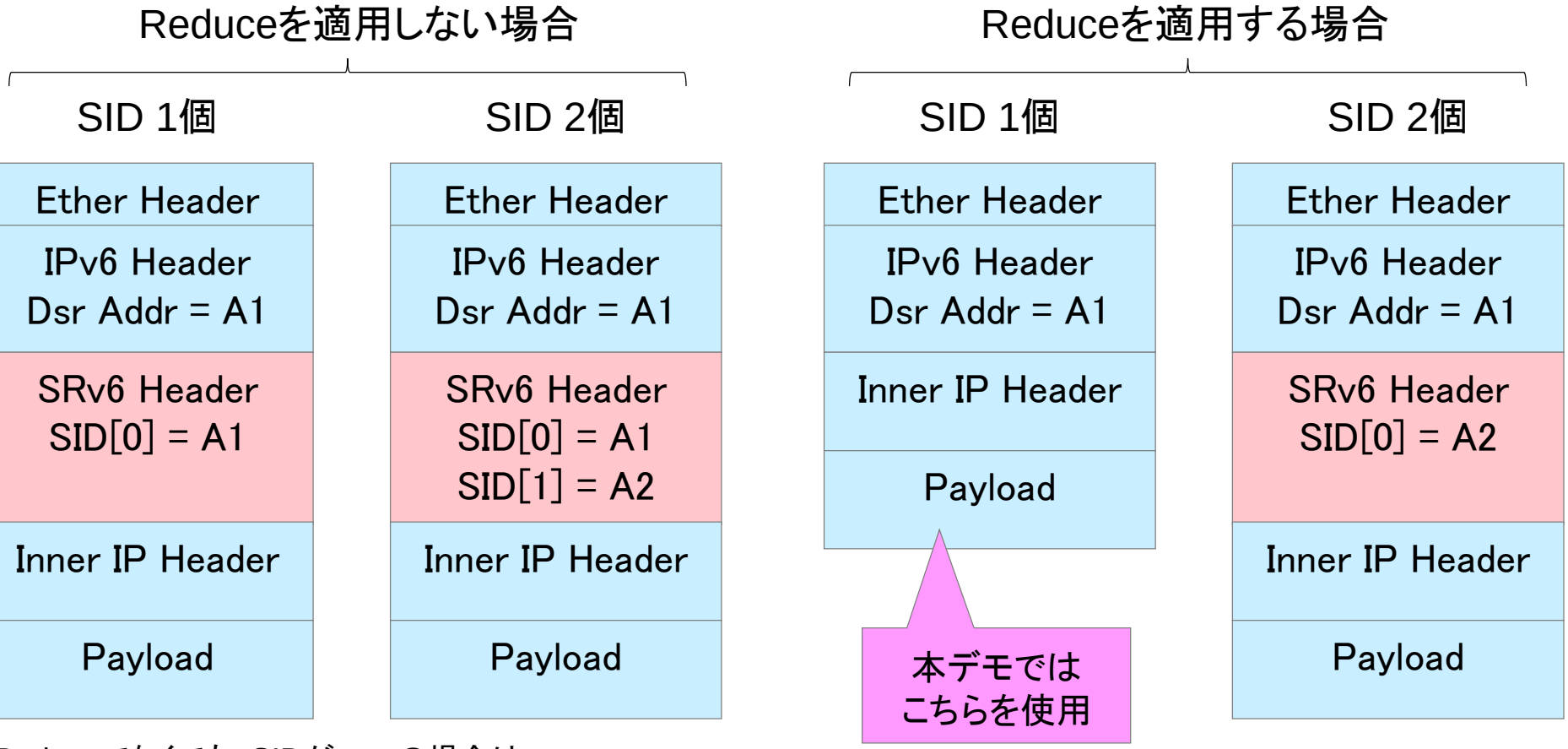


SRv6 Function

- SRv6 Function = SRv6のフレームを制御する機能の種別
  - 参照：IETF Draft: SRv6 Network Programming
    - https://www.ietf.org/id/draft-ietf-spring-srv6-network-programming-17.txt
- SR Headend
  - オリジナルフレームをSRv6に変換
    - H.Encaps, H.Encaps.Red, H.Encaps.L2, H.Encaps.L2.Red
- SR Endpoint
  - SRv6 HeaderのSIDリストをもとにOuter IPv6の宛先アドレスを変換
    - End,,,
  - SRv6からオリジナルフレームに変換
    - End.DX6, End.DX4, End.DT6, End.DT4,,,

※ 本デモでは、H.Encaps.Red（Reduce）とEnd.DT4を使用

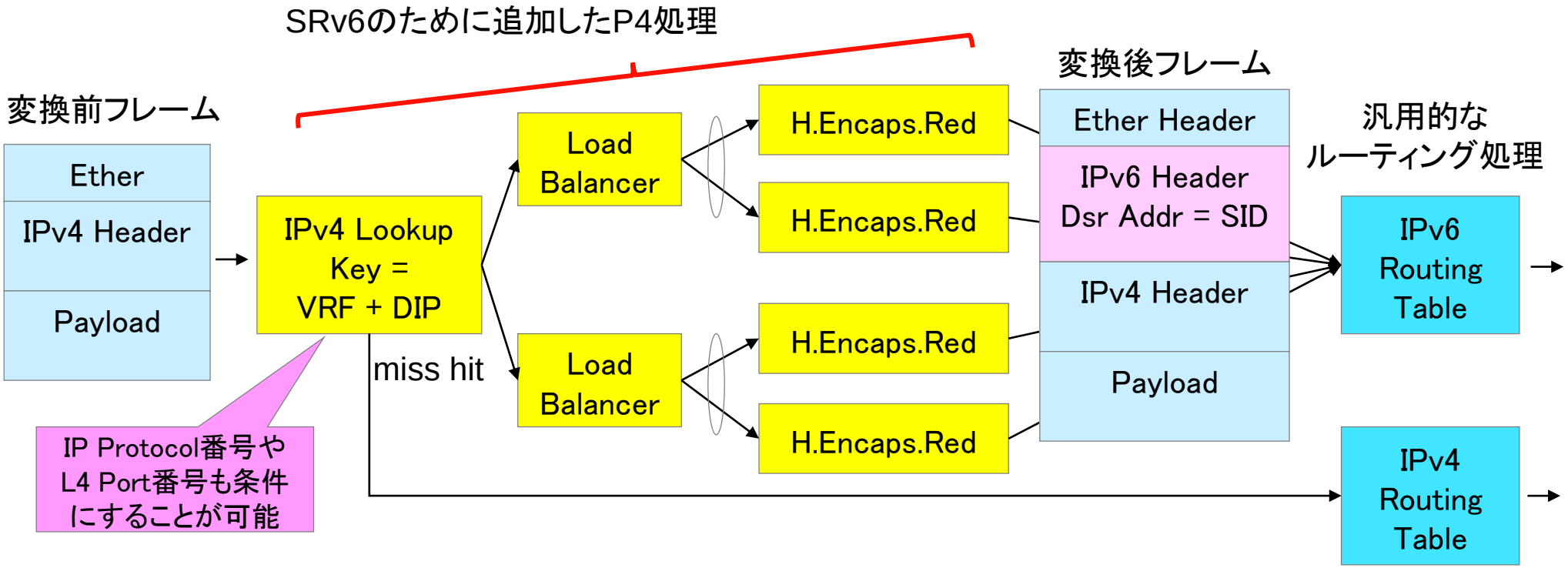
- H.Encaps.Red (Reduce)
  - SID Listの最初のSIDをSRv6ヘッダに含めない



※Reduceでなくても、SIDが一つの場合は
SRHを省略してよい

H.Encaps.Redの実装方法

- 処理の流れ
  - VRF + DIPをキーにしてSRv6への変換対象かを判断
  - SRv6への変換が必要な場合、Load balancerを指定
  - Load balancerからActionを指定（SRv6変換）
  - SRv6変換後のIPv6ヘッダをもとにルーティング処理



本デモのEnd.DT4用のSIDの定義

SID (128bit)	Locator 64bit	Function ID 16bit	Argument 48bit

各フィールドのbit長は任意

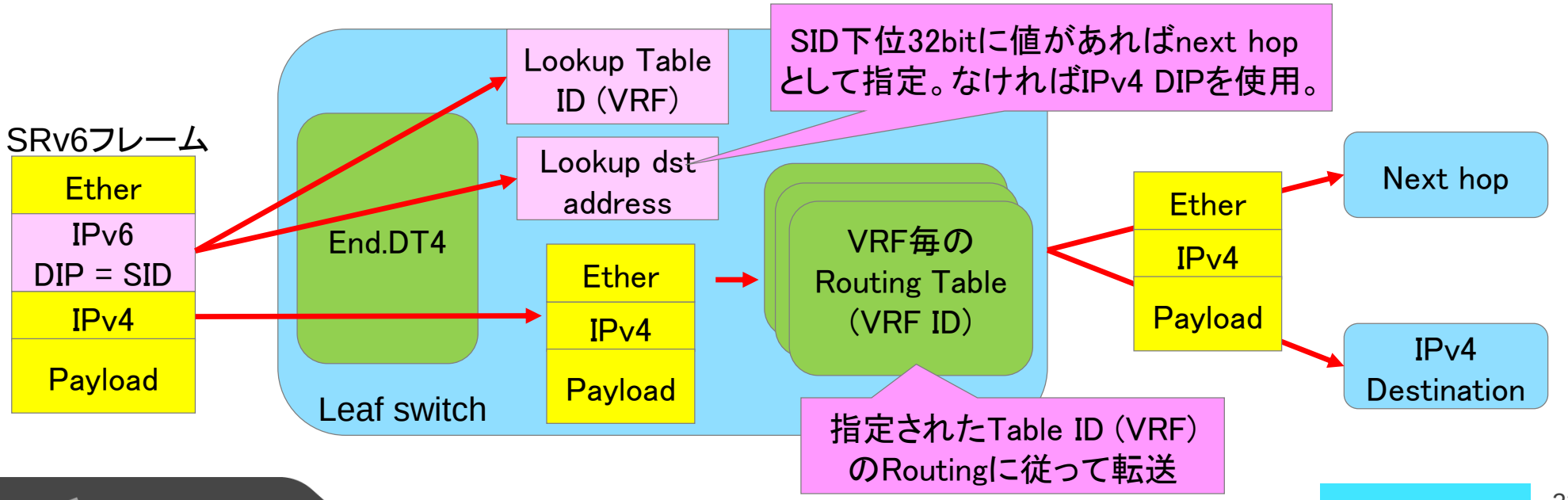
- End.DT4の例
  - FD00:FFFF:0:2:11:1006:0:0
  - FD00:FFFF:0:3:11:1006:a03:20b※

※1006 → VRF ID,
0a03020b → IPv4: 10.3.2.11

Locator
Leafスイッチ

Function
End.DT4

Argument
VRF ID (16bit)
Next hop IPv4 (32bit)



実際に変換されたSRv6のフレームキャプチャ

- Leaf1のVRFからLeaf3配下のNFV宛ての通信
  - 対象のVRFのindexが0x1004、Next hopのIPが10.202.1.11(0x0aca010b)
  - IPv6の送信先アドレス = FD00:FFFF:0:3:11:1004:aca:10b

```
> Frame 1: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface \\.\pipe\07-014-03-08-2020-10-35-37, id 0
> Ethernet II, Src: Portwell_62:c5:14 (00:90:fb:62:c5:14), Dst: Edgecore_f8:ea:41 (3c:2c:99:f8:ea:41)
v Internet Protocol Version 6, Src: fd00:ffff:0:1:0:1004:ac9:10b, Dst: fd00:ffff:0:3:11:1004:aca:10b
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 64
    Next Header: IPIP (4)
    Hop Limit: 64
    Source: fd00:ffff:0:1:0:1004:ac9:10b
    Destination: fd00:ffff:0:3:11:1004:aca:10b
v Internet Protocol Version 4, Src: 10.201.1.11, Dst: 10.101.1.11
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 64
    Identification: 0x10b2 (4274)
  > Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live: 63
    Protocol: TCP (6)
    Header checksum: 0x13c3 [correct]
    [Header checksum status: Good]
    [Calculated Checksum: 0x13c3]
    Source: 10.201.1.11
    Destination: 10.101.1.11
> Transmission Control Protocol, Src Port: 51108 (51108), Dst Port: hbci (3000), Seq: 1, Ack: 1, Len: 0
```

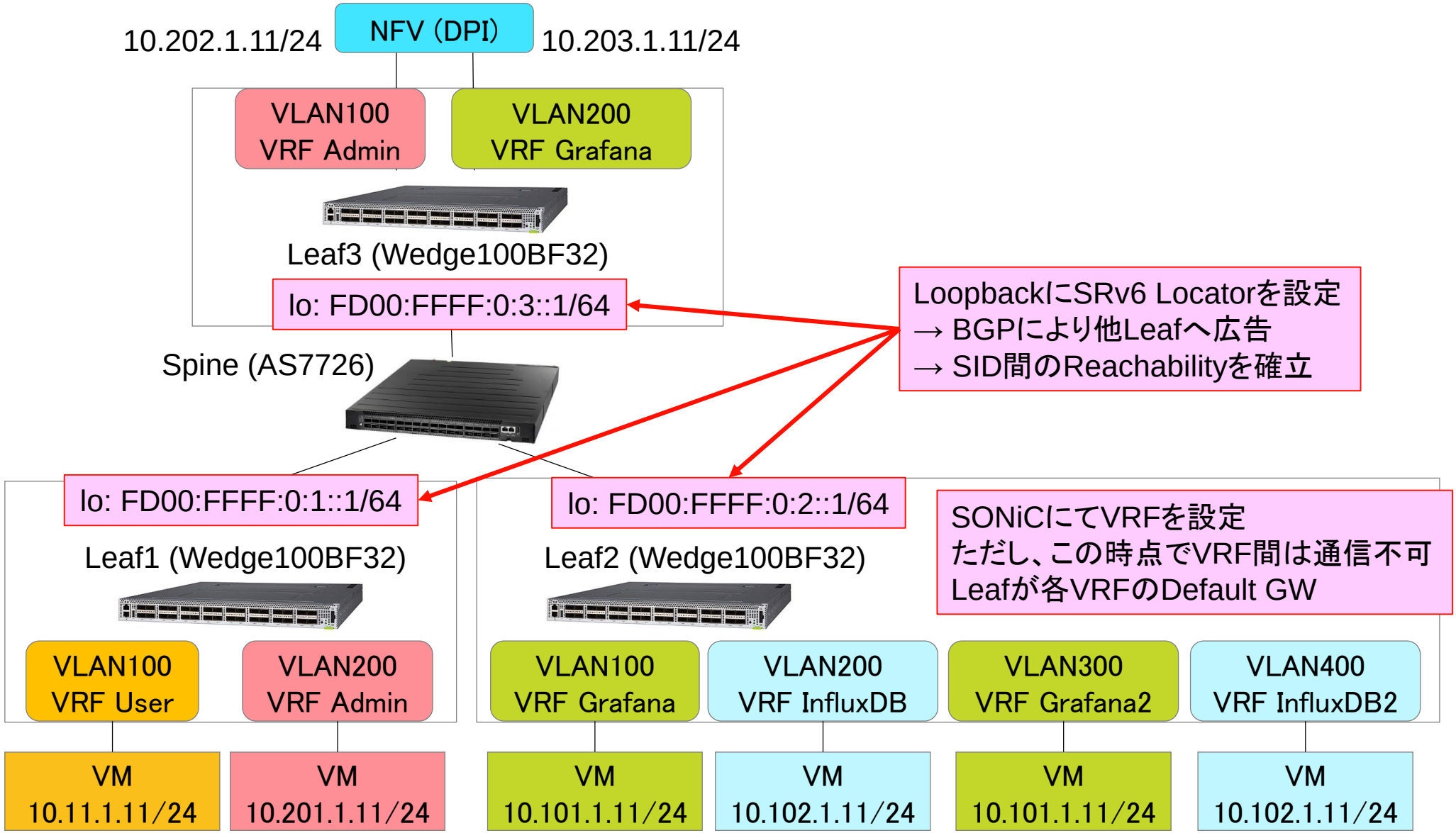
SRv6サービスチェイニング

実機デモ

APRESIA®

JANOG46

初期状態 SONiCの設定



SRv6 Locatorの広告

◆ LoopbackアドレスにSRv6 Locatorを設定

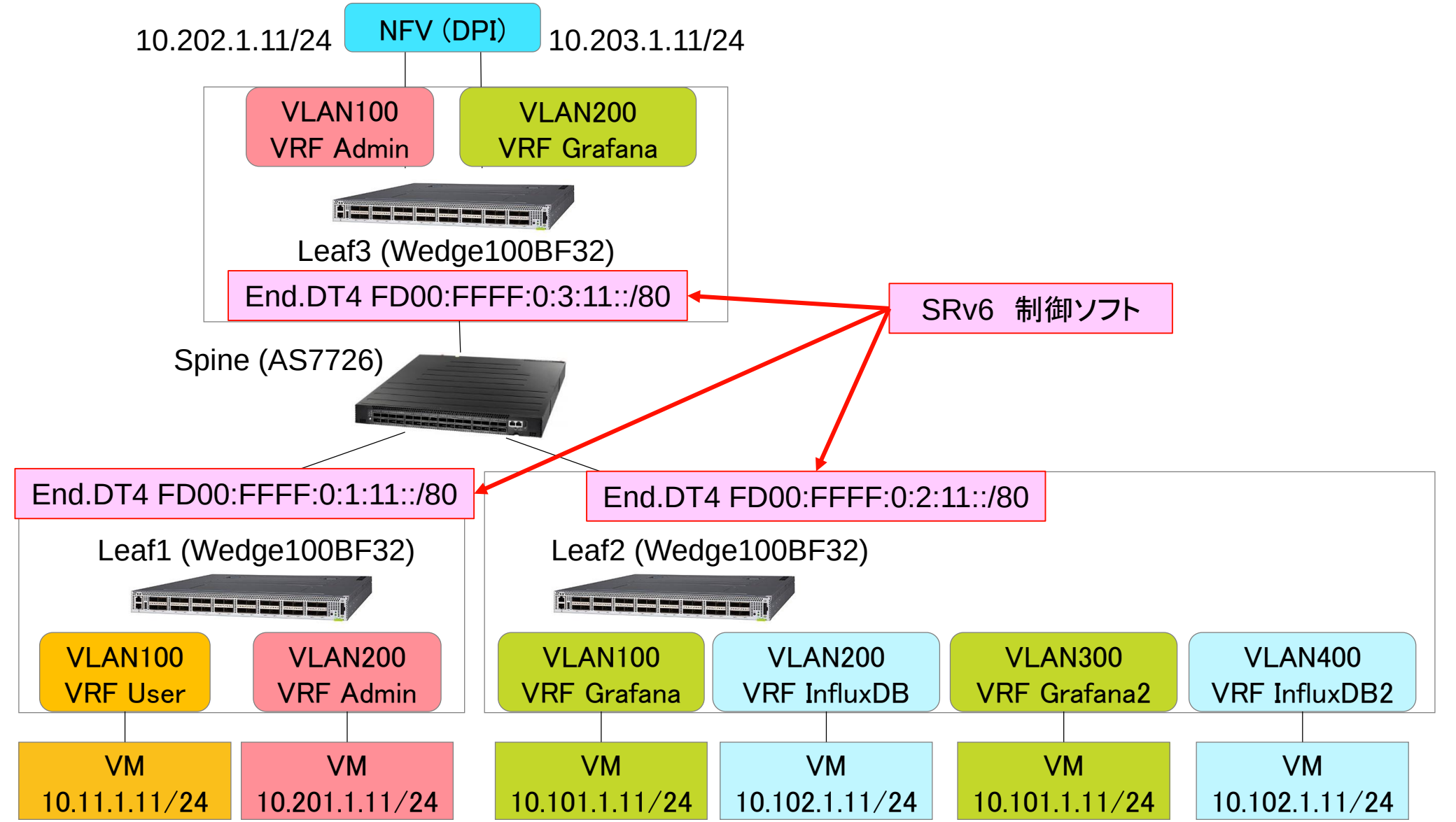
→ BGPにより他Leafに広告

```
Leaf1# show ipv6 route
Codes: K - kernel route, C - connected, S - static, R - RIPng,
       O - OSPFv3, I - IS-IS, B - BGP, N - NHRP, T - Table,
       v - VNC, V - VNC-Direct, A - Babel, D - SHARP, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route

C>* fd00:ffff:0:1::/64 is directly connected, Loopback0, 1d21h08m
B>* fd00:ffff:0:2::/64 [20/0] via fe80::3e2c:99ff:fef8:ea41, Ethernet0, 00:28:26
B>* fd00:ffff:0:3::/64 [20/0] via fe80::3e2c:99ff:fef8:ea41, Ethernet0, 00:28:26
C * fe80::/64 is directly connected, Bridge, 1d21h08m
C * fe80::/64 is directly connected, Ethernet8, 1d21h08m
C * fe80::/64 is directly connected, Ethernet4, 1d21h08m
C * fe80::/64 is directly connected, Ethernet0, 1d21h08m
C * fe80::/64 is directly connected, Loopback0, 1d21h08m
C * fe80::/64 is directly connected, usb0, 1d21h08m
C>* fe80::/64 is directly connected, eth0, 1d21h08m
```

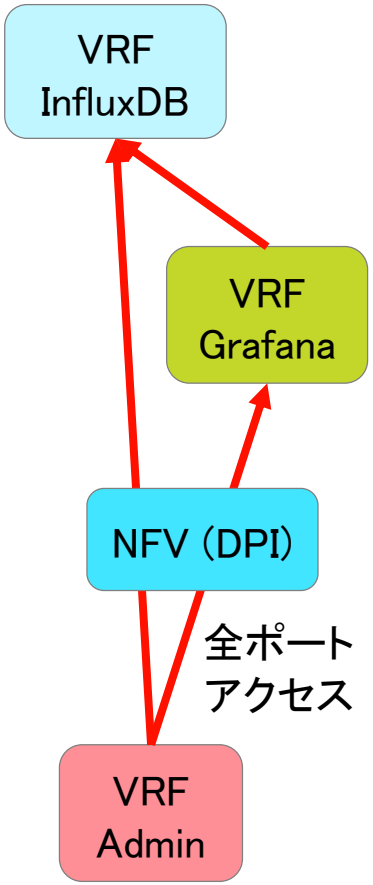
Leaf2, Leaf3のSRv6のLocator
がBGPによって同期されている

初期設定 SRv6のEnd.DT4の設定

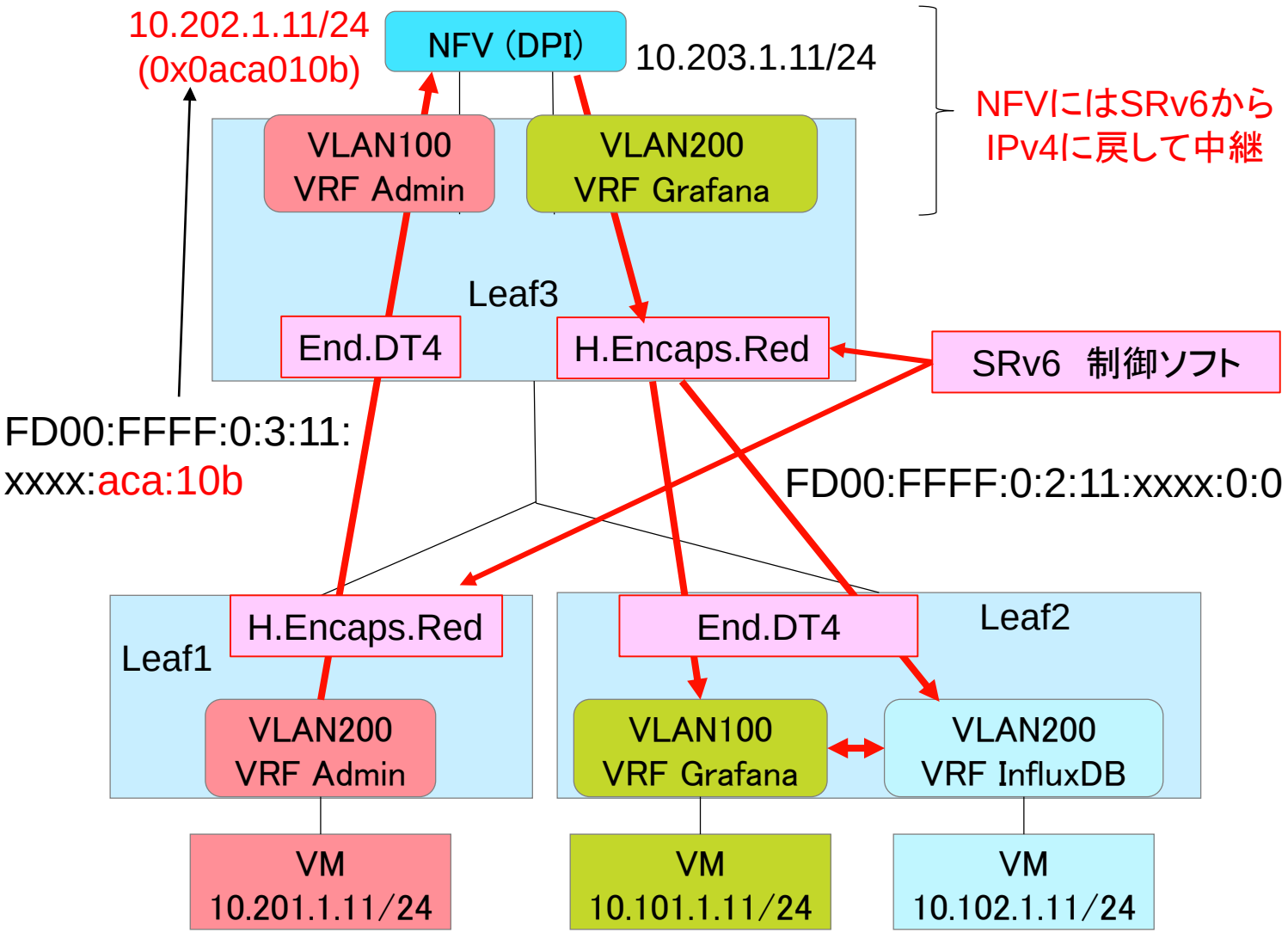


デモシナリオ① Admin VRF （アクセス制限無し＋NFV）

VRF間の通信



物理的な通信経路（例：上り方向）

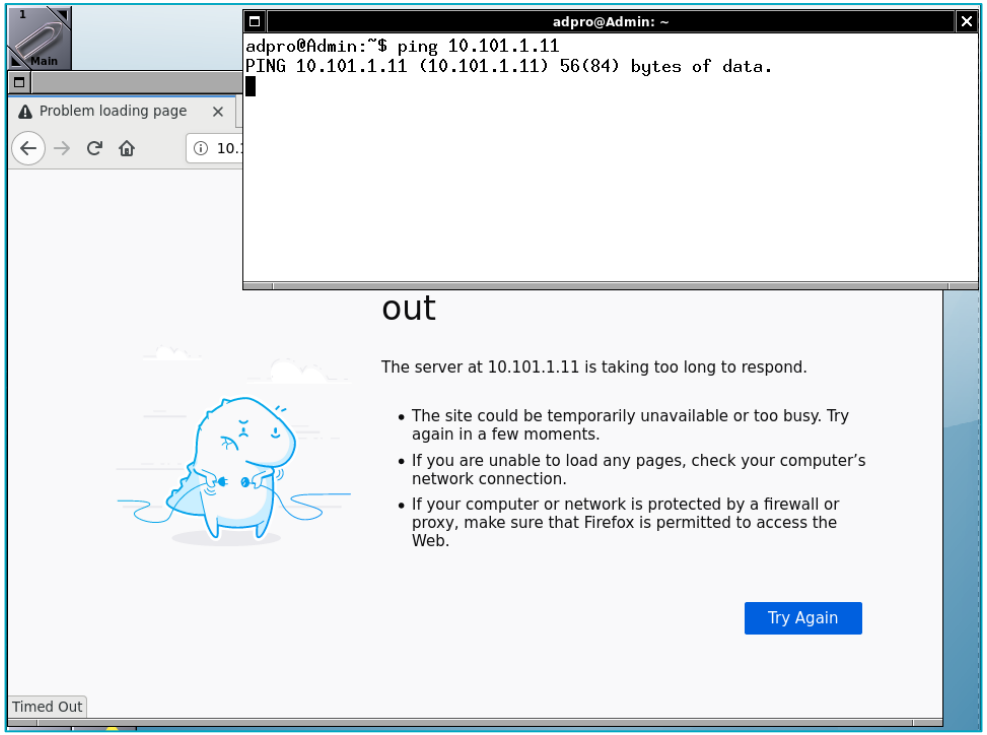


デモシナリオ① Admin VRF （アクセス制限無し＋NFV）

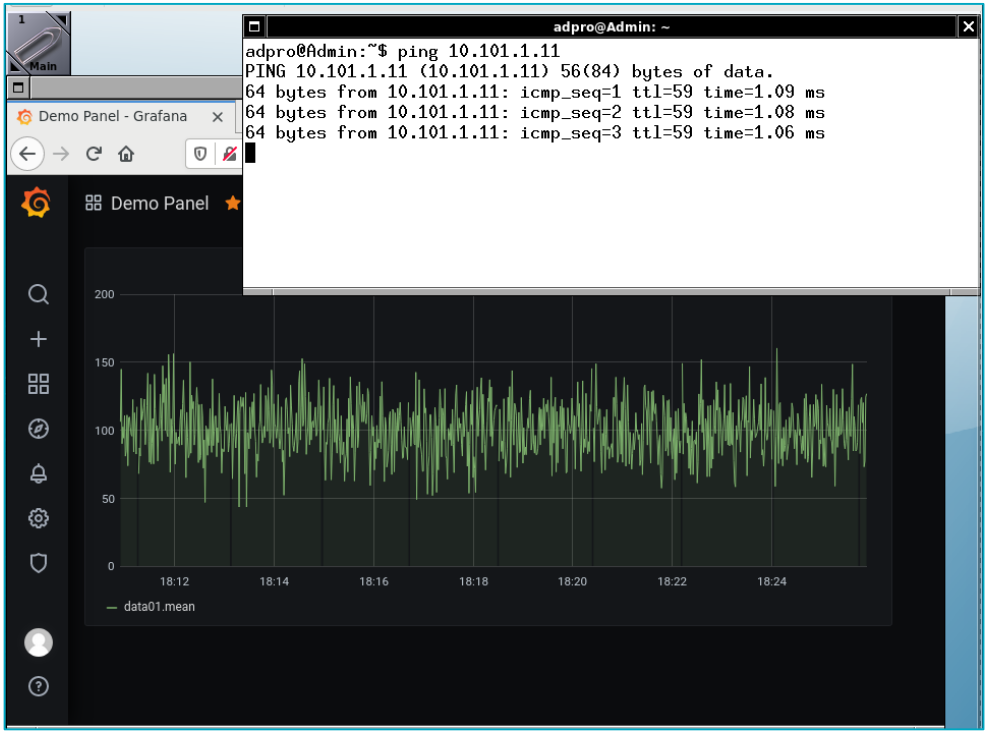
◆ 以下を実行

```
def T01_admin():
    set_service_chain_nfv(vrf['admin'], nfv['admin'], nfv['grafana'], vrf['grafana'])
    set_service_chain_nfv(vrf['admin'], nfv['admin'], nfv['grafana'], vrf['influxdb'])
    set_service_chain_nfv(vrf['grafana'], nfv['grafana'], nfv['admin'], vrf['admin'])
    set_service_chain_nfv(vrf['influxdb'], nfv['grafana'], nfv['admin'], vrf['admin'])
```

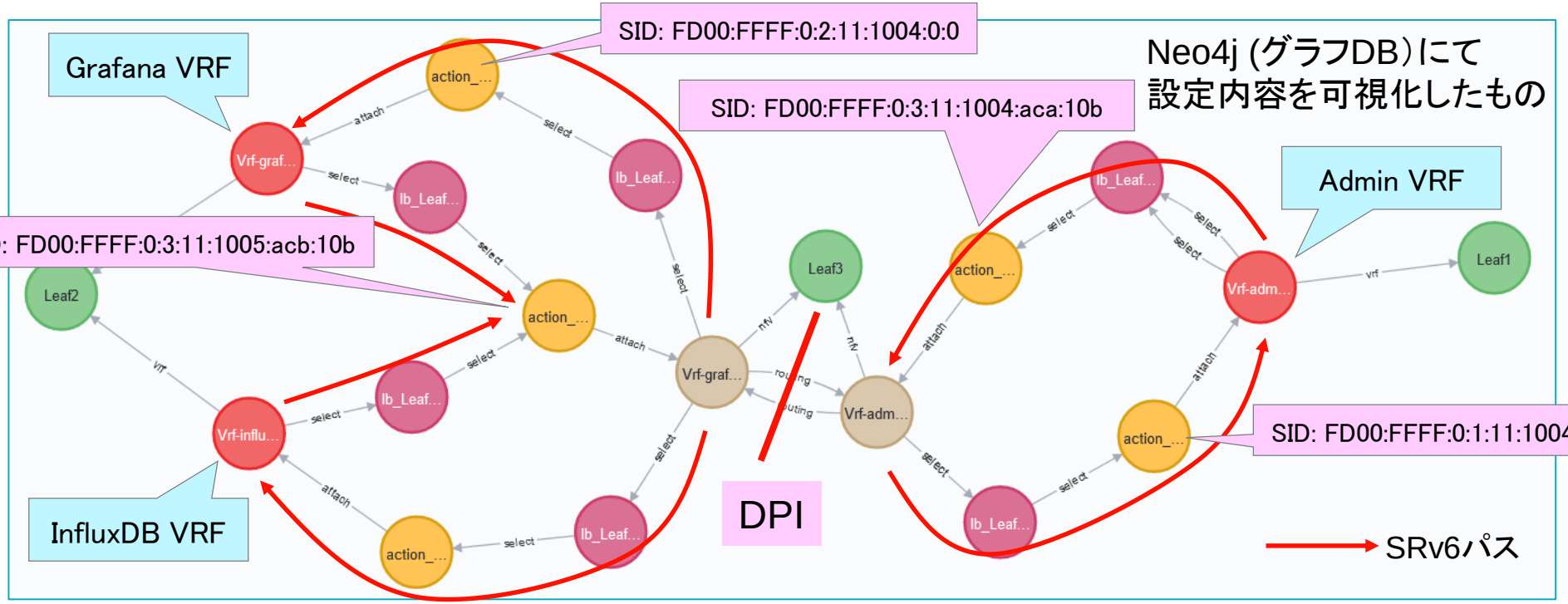
Ping不可、Grafanaアクセス不可



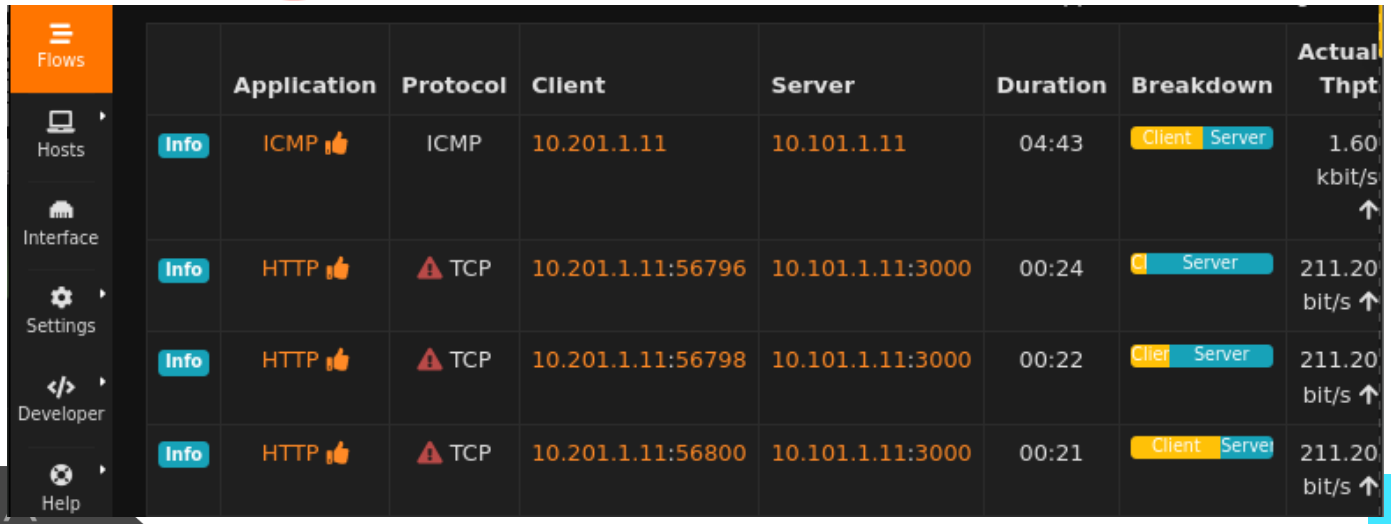
Ping成功、Grafanaアクセス成功



デモシナリオ① Admin VRF （アクセス制限無し＋NFV）

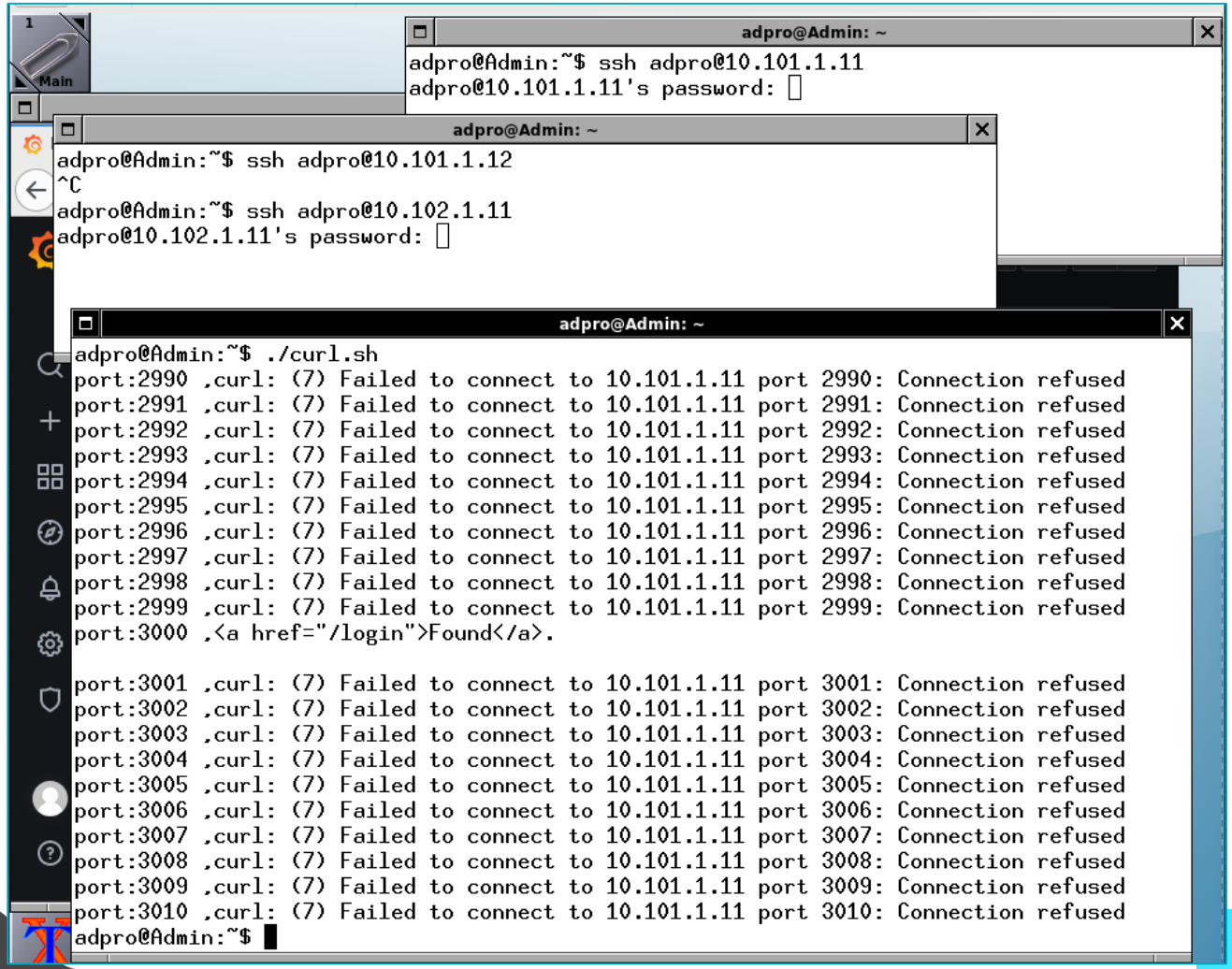


DPIとして
ntopngを使用

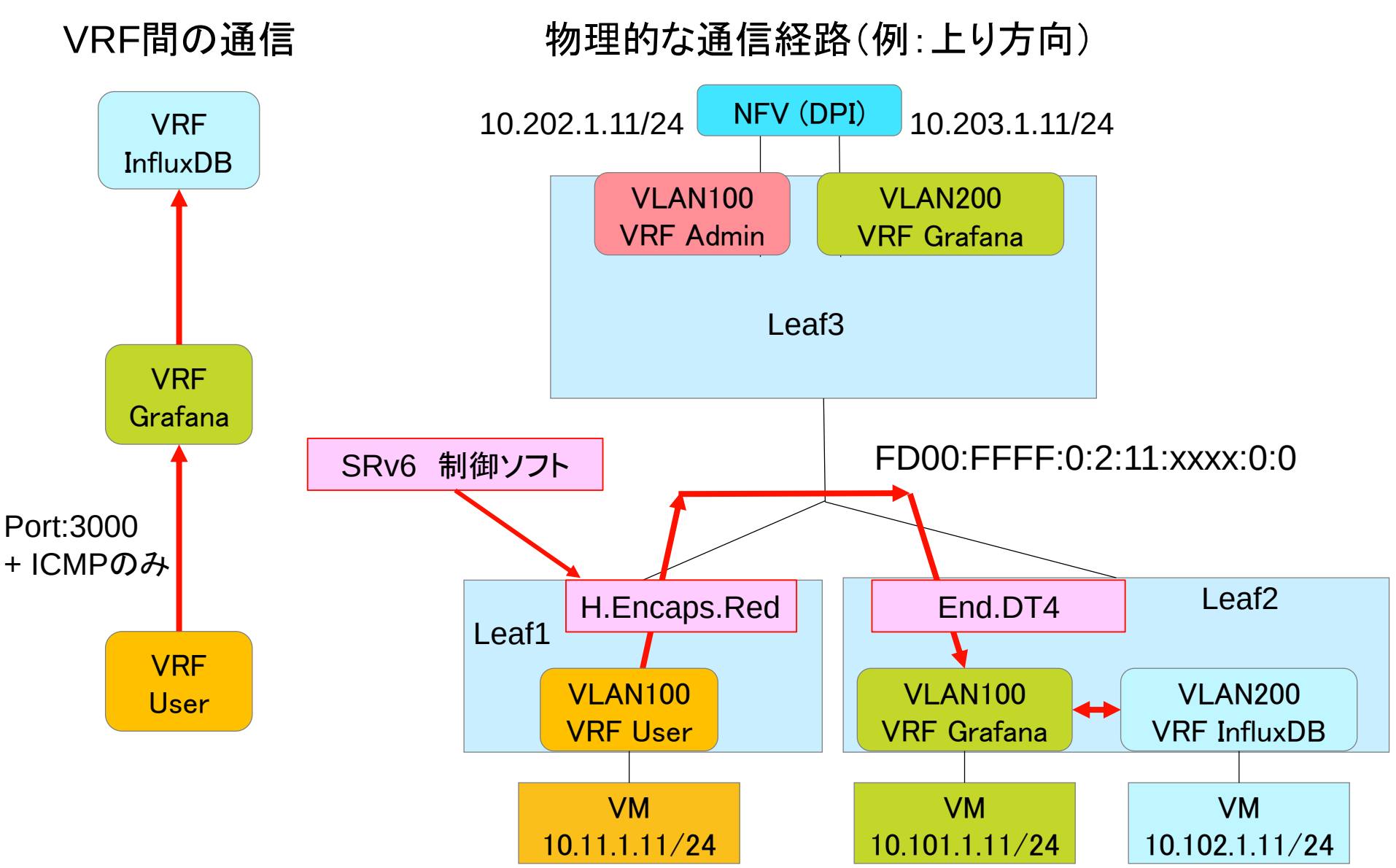


デモシナリオ① Admin VRF （アクセス制限無し＋NFV）

- Grafana, InfluxdbともにICMP/SSHのアクセス可能
- GrafanaへはPort 3000以外も通信は到達可能
  - connection refusedはGrafanaが応答を返している



デモシナリオ① User VRF （アクセス制限有り + NFV無し）



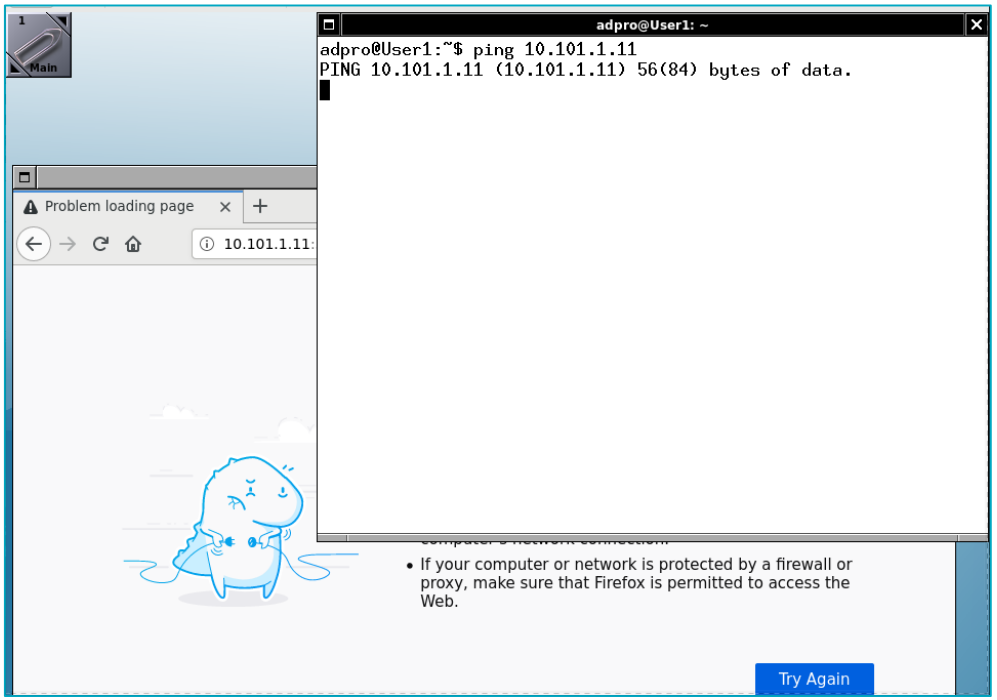
デモシナリオ① User VRF （アクセス制限有り + NFV無し）

◆ 以下を実行

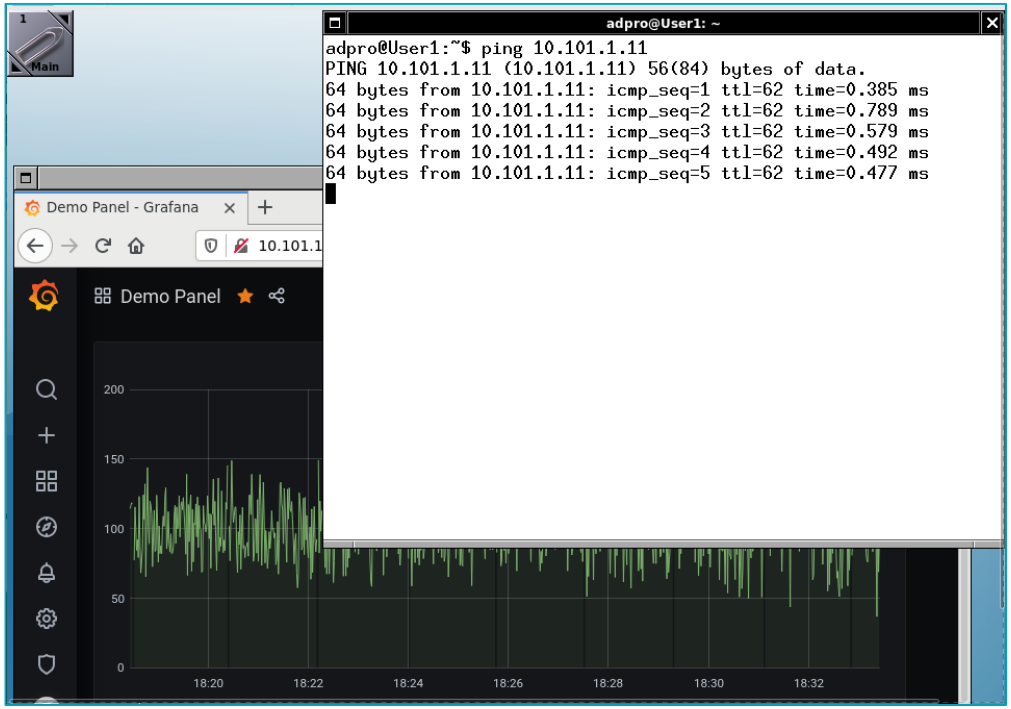
ICMPとTCP Port 3000のみの
アクセスに限定したapp_idを指定

```
def T02_user():
    set_service_chain(vrf['user'], [vrf['grafana']], app_id=USER_APP_ID)
    set_service_chain(vrf['grafana'], [vrf['user']], app_id=USER_APP_ID)
```

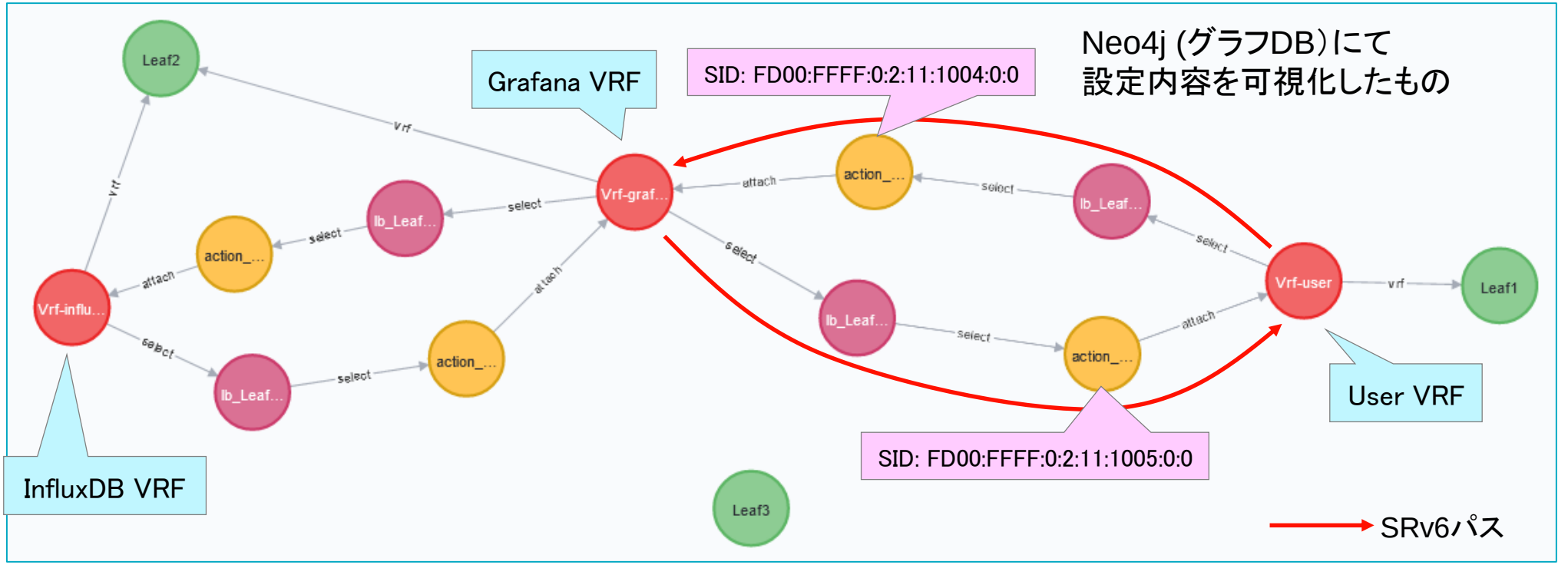
Ping不可、Grafanaアクセス不可



Ping成功、Grafanaアクセス成功

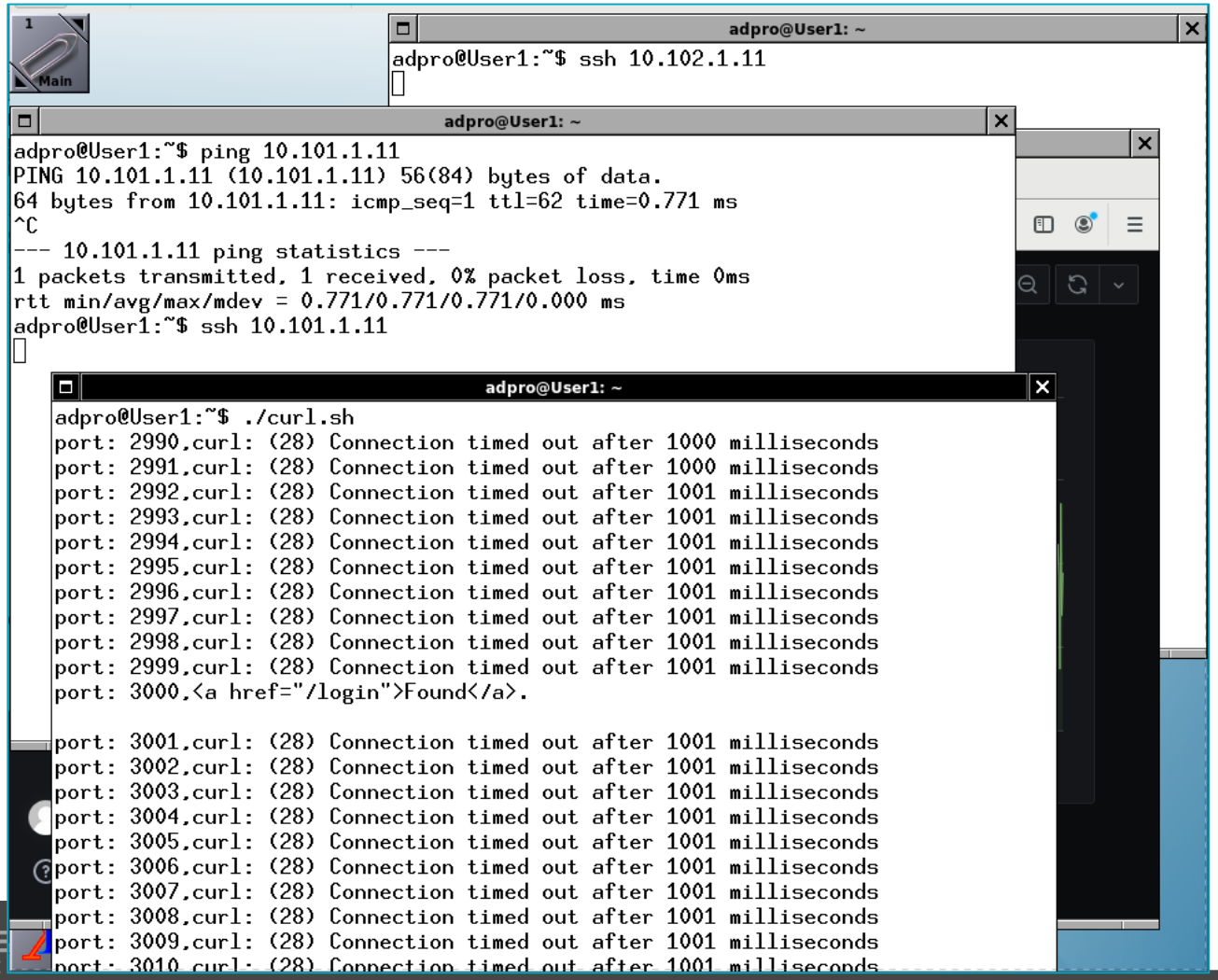


デモシナリオ① User VRF （アクセス制限有り + NFV無し）

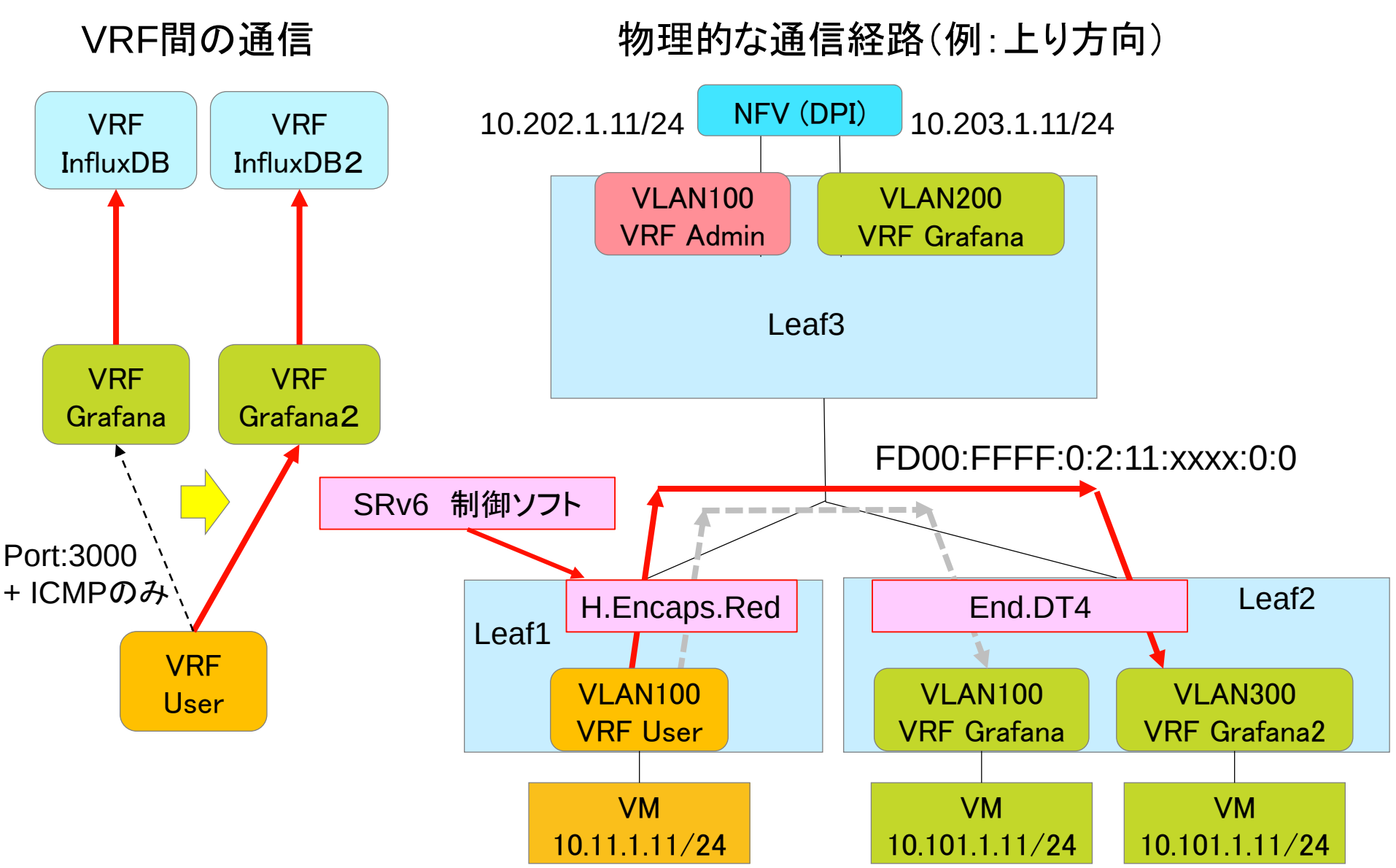


デモシナリオ① User VRF （アクセス制限有り + NFV無し）

- SSHの通信不可
- ICMPはGrafanaのみアクセス可能
- GrafanaへはPort 3000のみ到達可能（それ以外のPortはGrafanaへ到達できずタイムアウト）



デモシナリオ②　User VRF　サービス切り替え

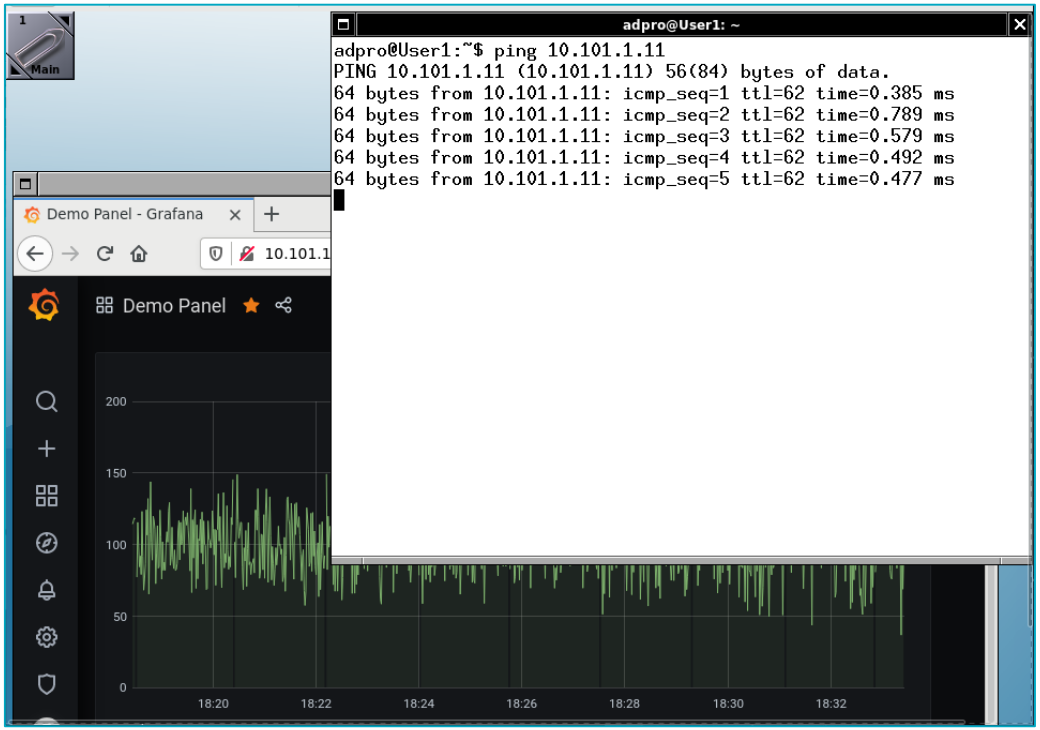


デモシナリオ②　User VRF　サービス切り替え

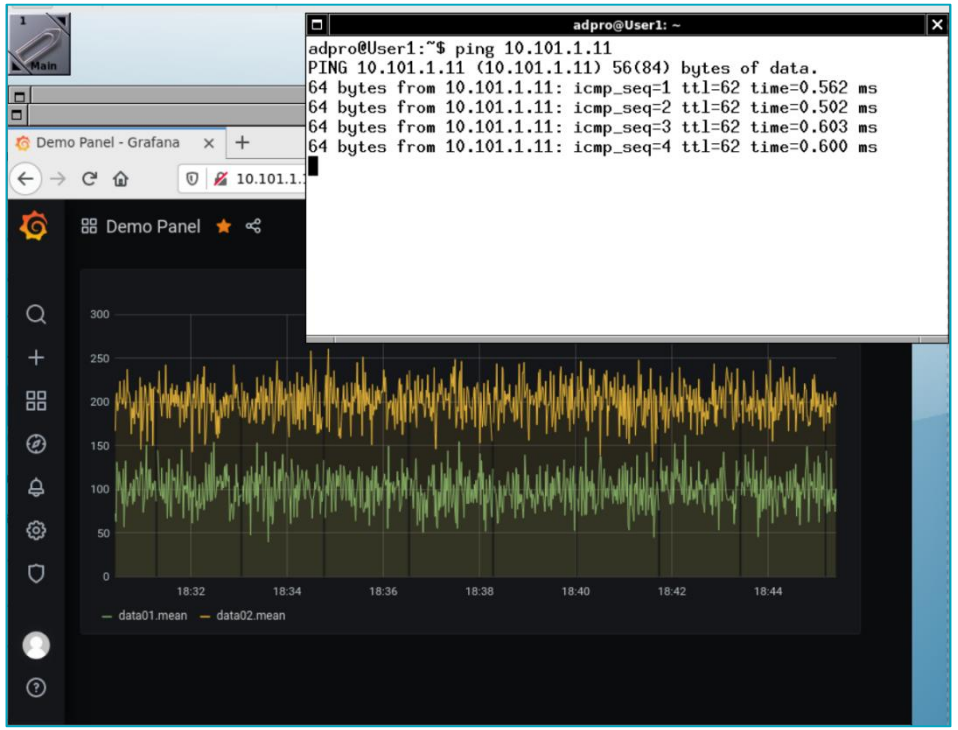
◆ 以下を実行

```
def T03_user_change():
    set_service_chain(vrf['user'], [vrf['grafana2']], app_id=USER_APP_ID)
    set_service_chain(vrf['grafana2'], [vrf['user']], app_id=USER_APP_ID)
```

Ping成功、Grafanaアクセス不可

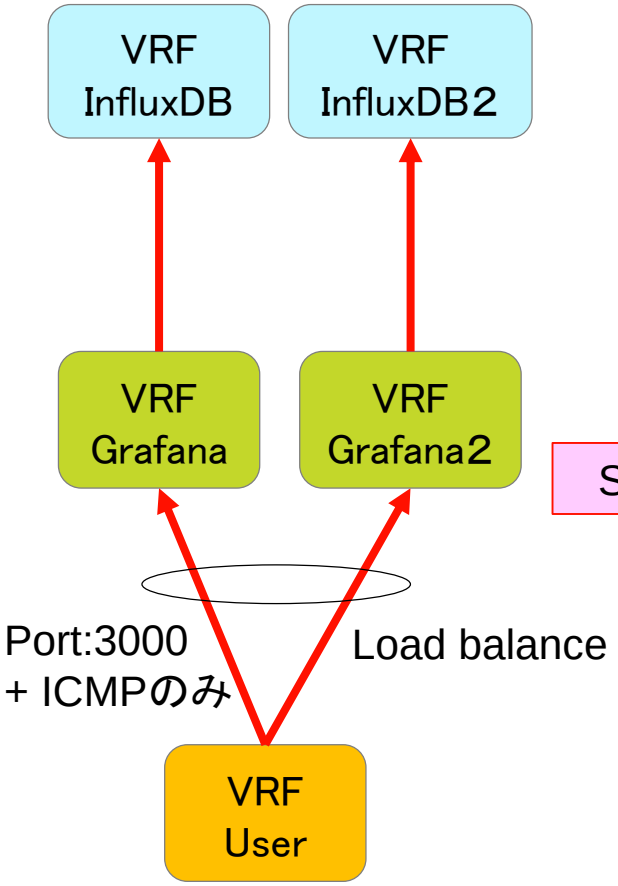


Ping成功、Grafanaの画面切り替え

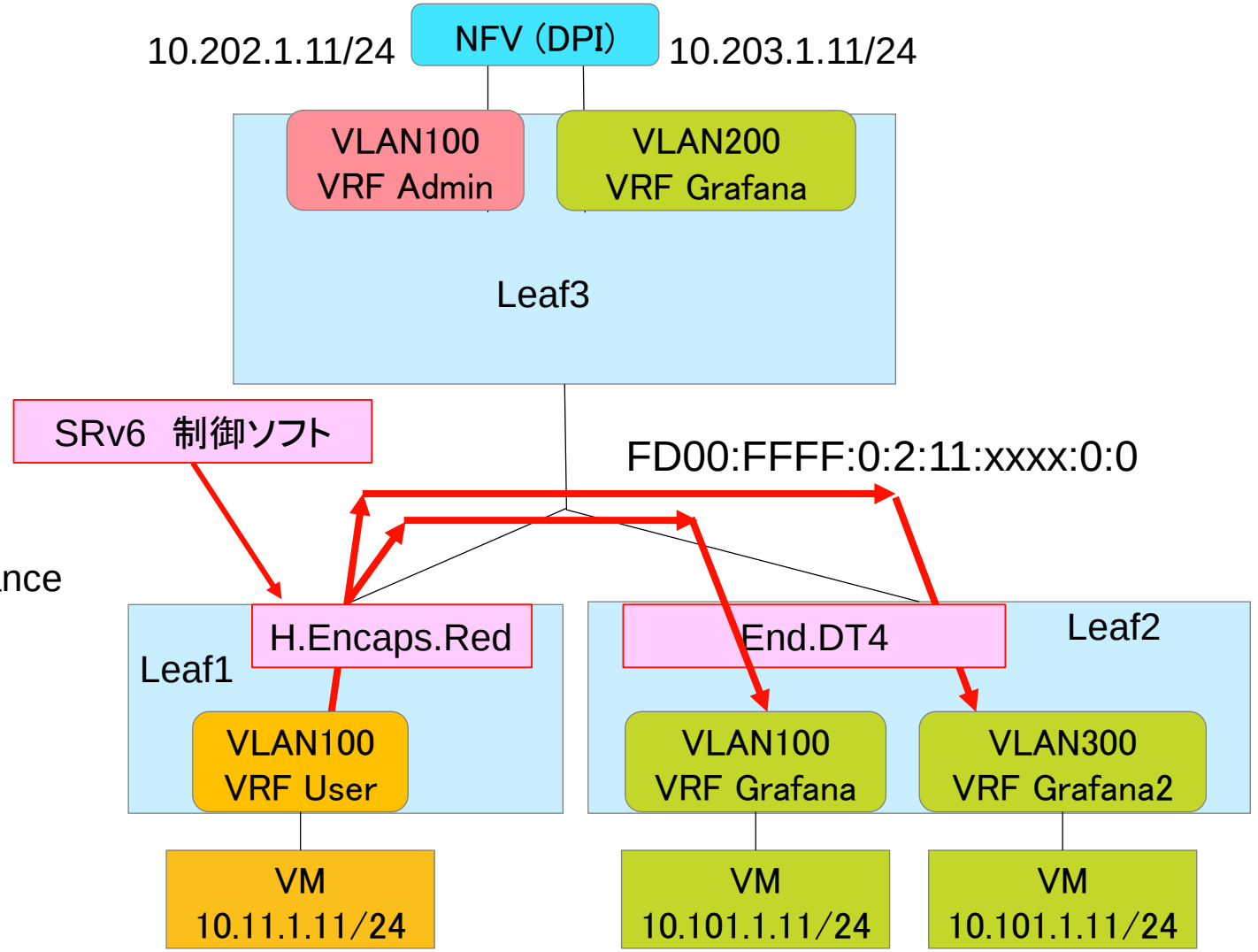


デモシナリオ③ Grafana VRFのLoad balancer

VRF間の通信



物理的な通信経路（例：上り方向）

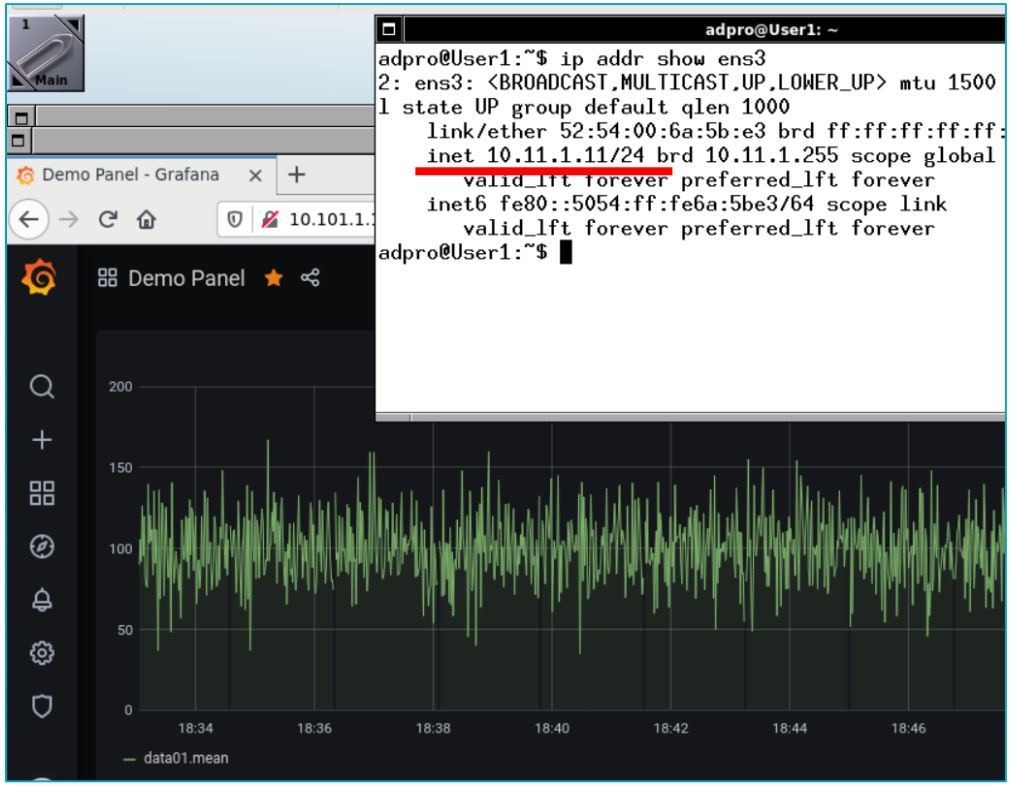


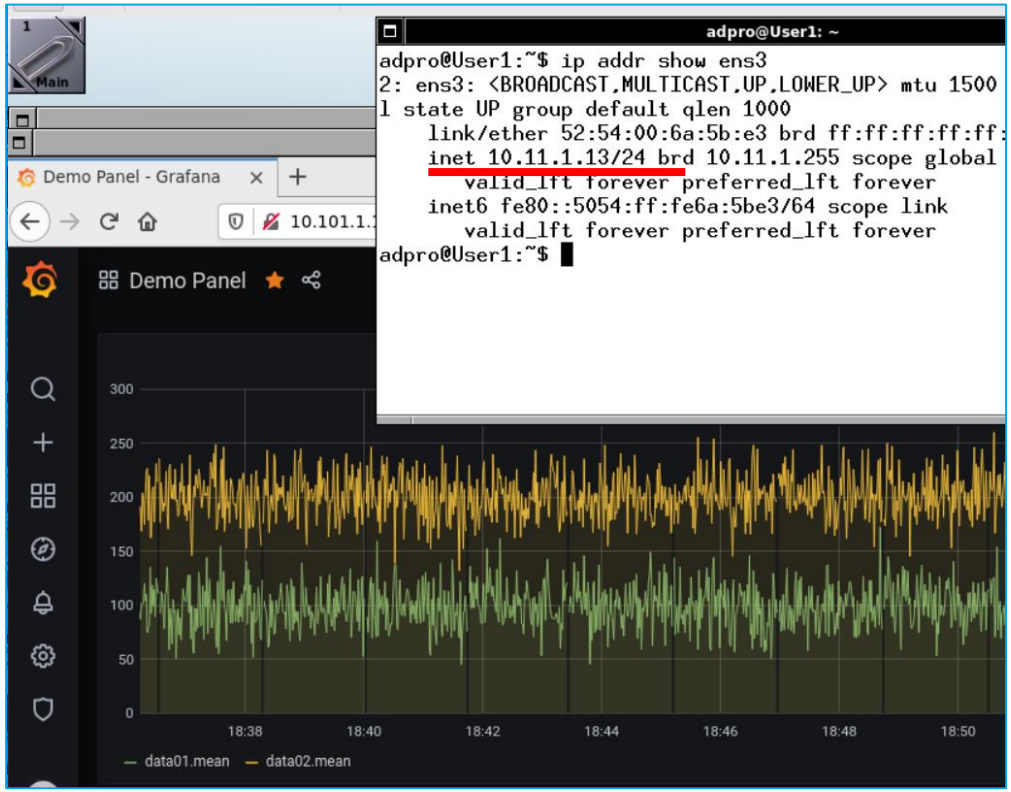
デモシナリオ③ Grafana VRFのLoad balancer

- 以下を実行

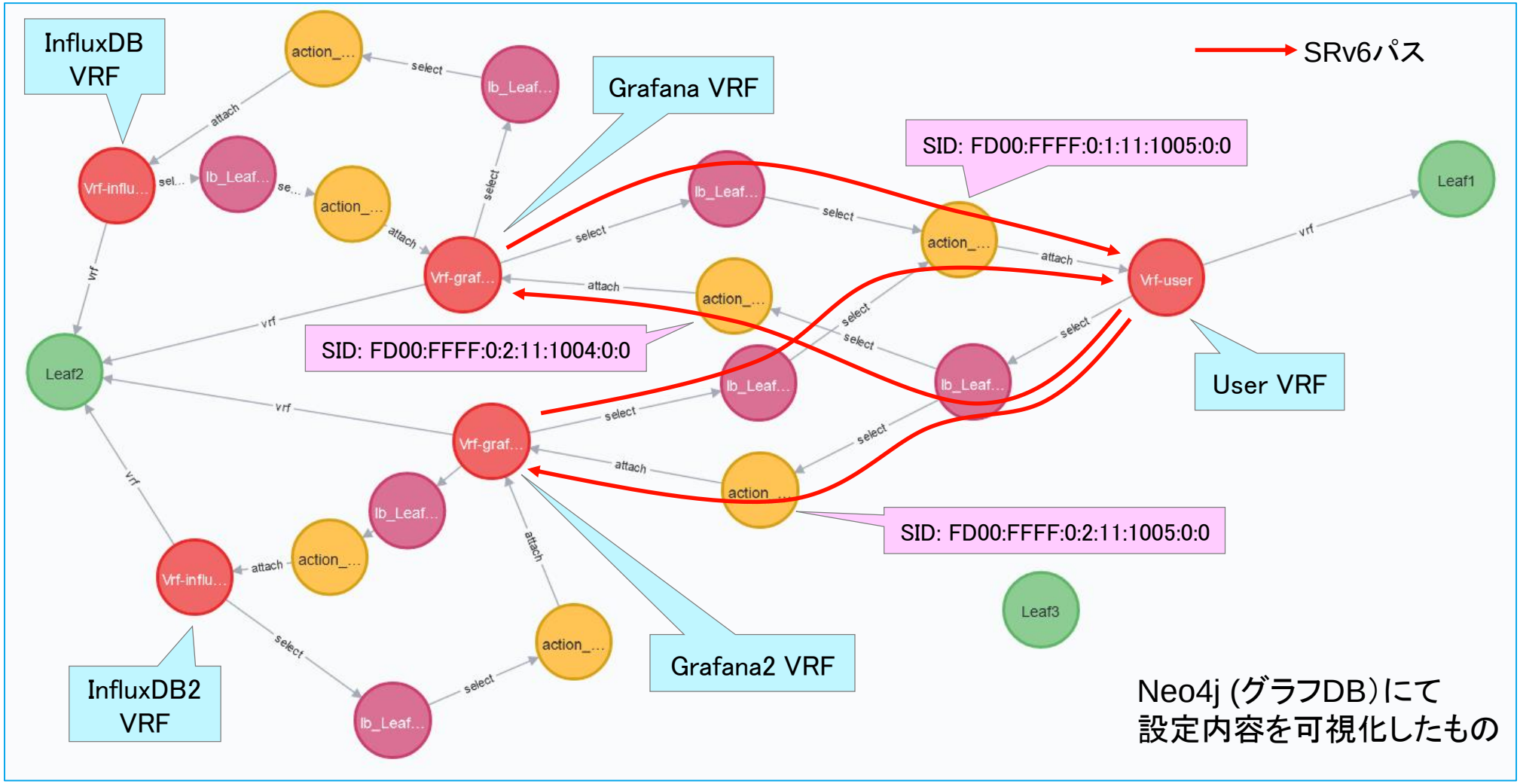
```
def T04_user_load_balancer():
    set_service_chain(vrf['user'], [vrf['grafana'], vrf['grafana2']], app_id=USER_APP_ID)
    set_service_chain(vrf['grafana'], [vrf['user']], app_id=USER_APP_ID)
    set_service_chain(vrf['grafana2'], [vrf['user']], app_id=USER_APP_ID)
```

- ユーザ側のIPアドレスを変更するとアクセス先のGrafana VRFが切り替わる（Load balance）





デモシナリオ③ Grafana VRFのLoad balancer

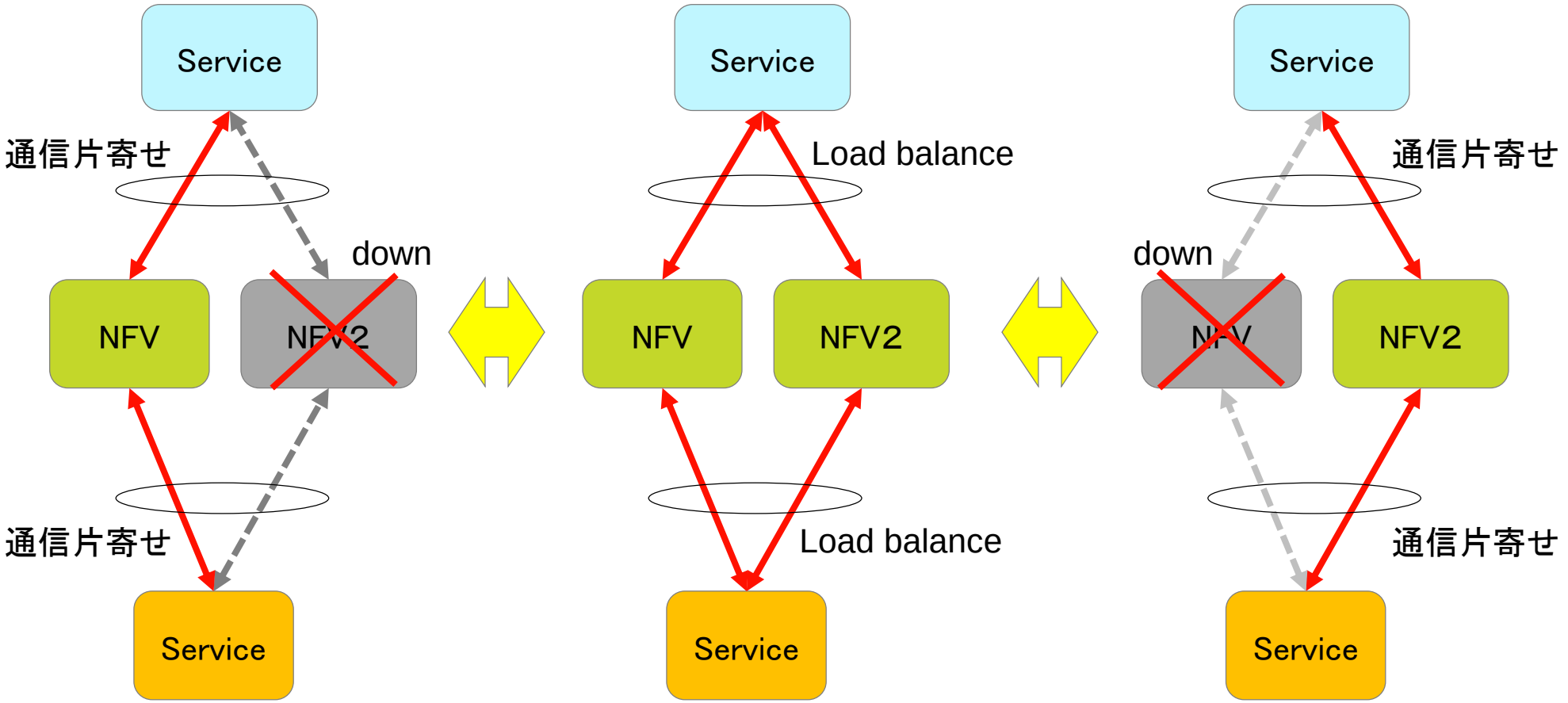


応用 + まとめ

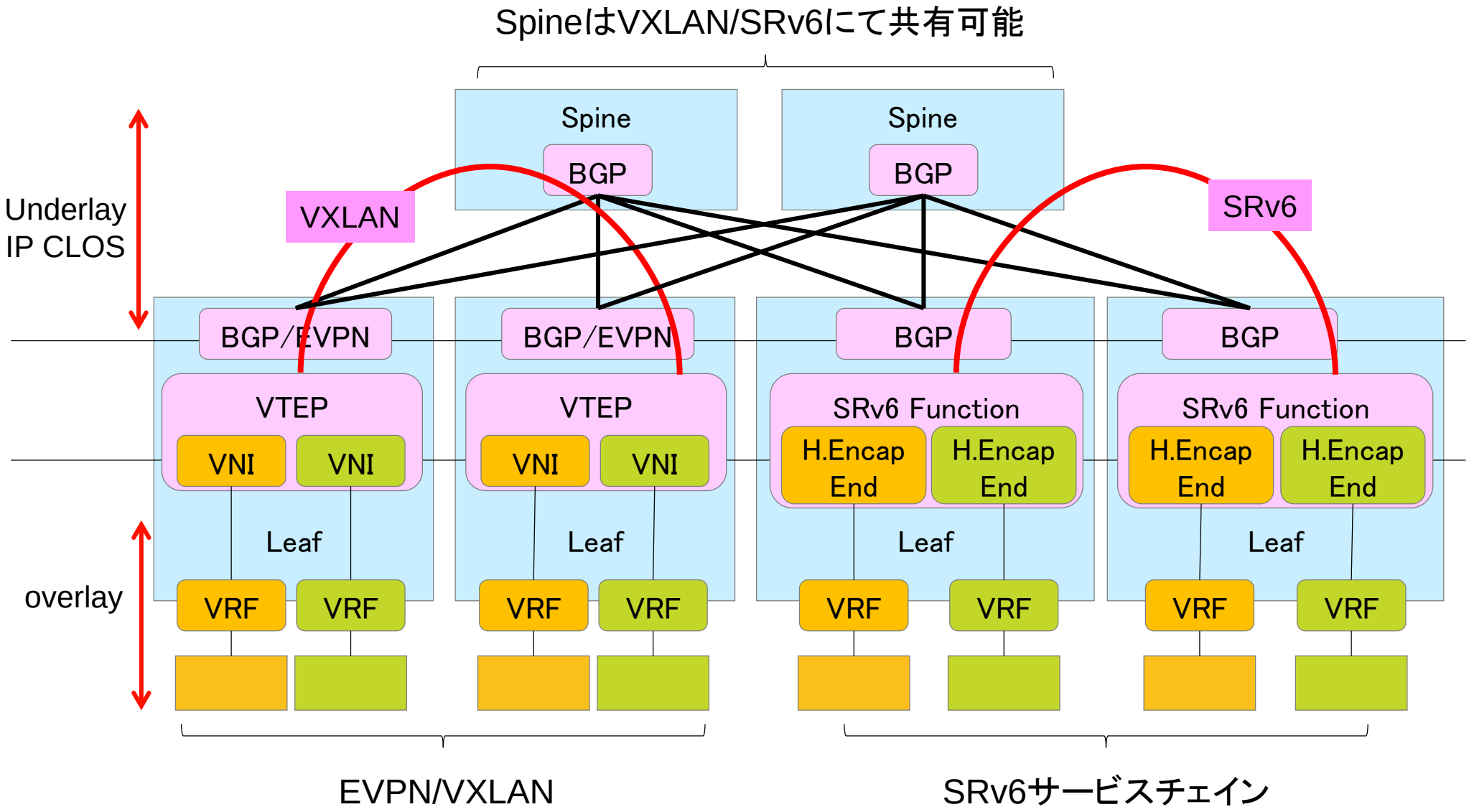
APRESIA®

JANOG46

Load balancerの応用：NFVの冗長



EVPN/VXLAN環境とSRv6サービスチェインの共存



まとめ

- 本日お話したこと
  - SONiCとSRv6(P4)の組み合わせによって、マルチテナントネットワークを実現
    - SONiC: IP CLOSファブリック（BGP）、VRF、SRv6 Locatorの広告
    - SRv6: VRF間の中継を制御（サービスチェイン）
      - サービスメッシュと同様なネットワーク制御をVRFに対して実現
      - セキュリティポリシーの適用（ポート制御）、Load balancer

- 議論いただきたいこと
  - DC内におけるマルチテナントネットワークの中継経路の管理に関する問題・課題
  - 今回ご紹介した中継経路の制御によって解決できる問題