



Collecting and analyzing  
routing information with

# ROTONDA

NLNOG Day 2025

September 30, 2025, Amsterdam

Luuk Hendriks

# BMP?

# BMP, oversimplified



AS 1

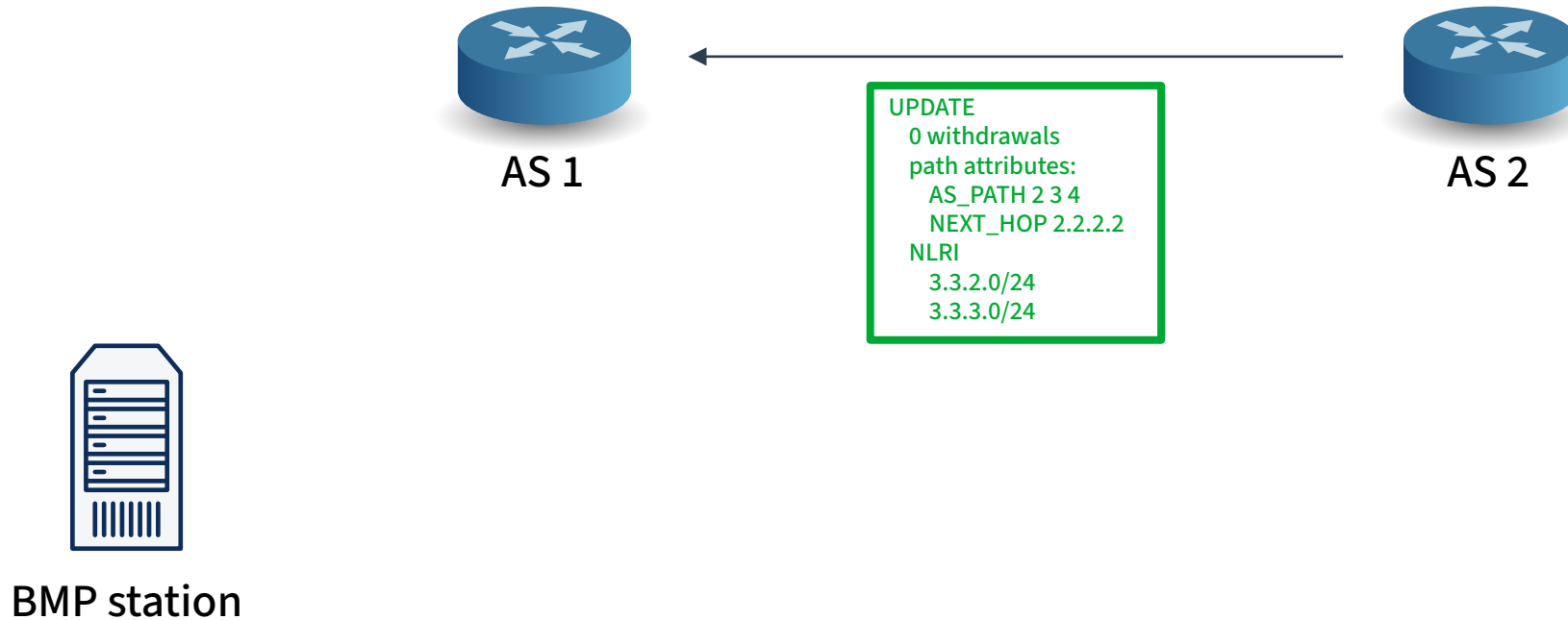


AS 2

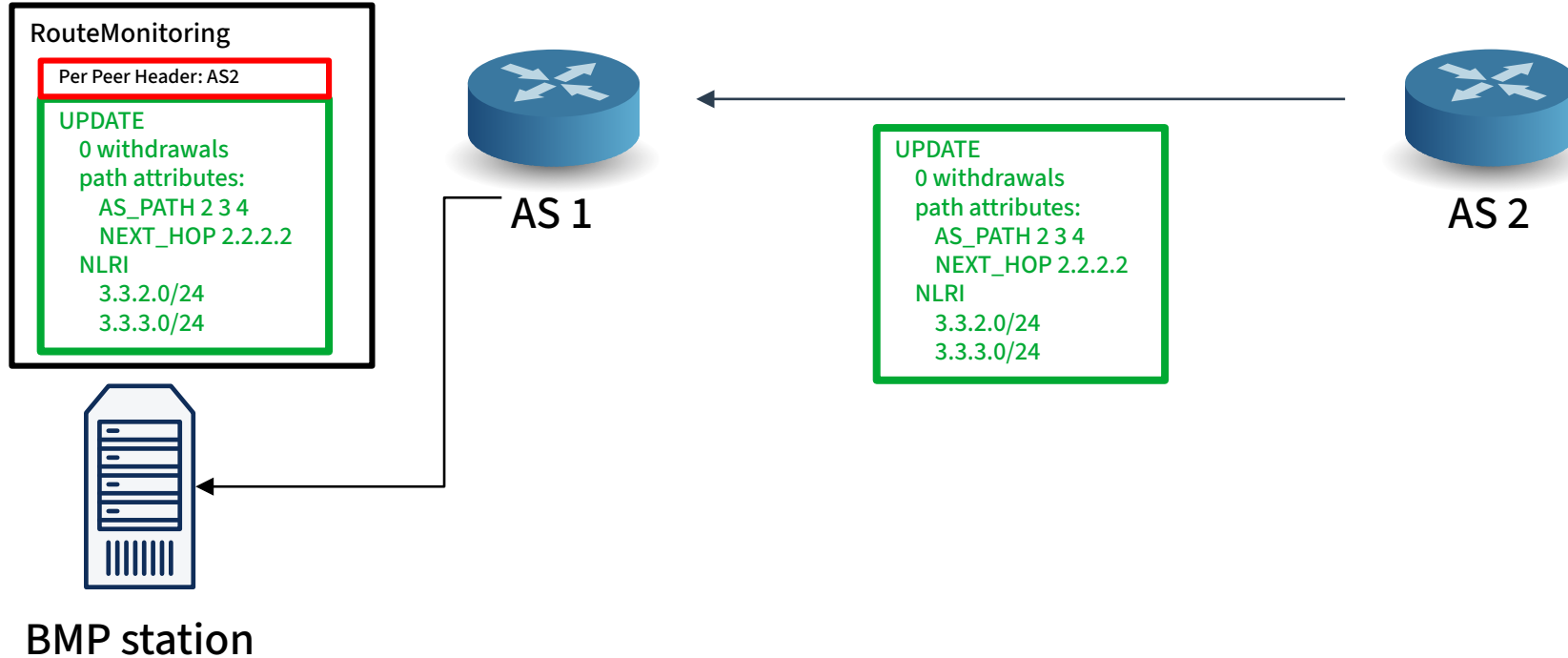


BMP station

# BMP, oversimplified



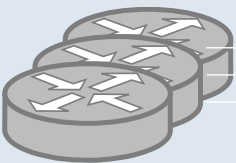
# BMP, oversimplified





Store



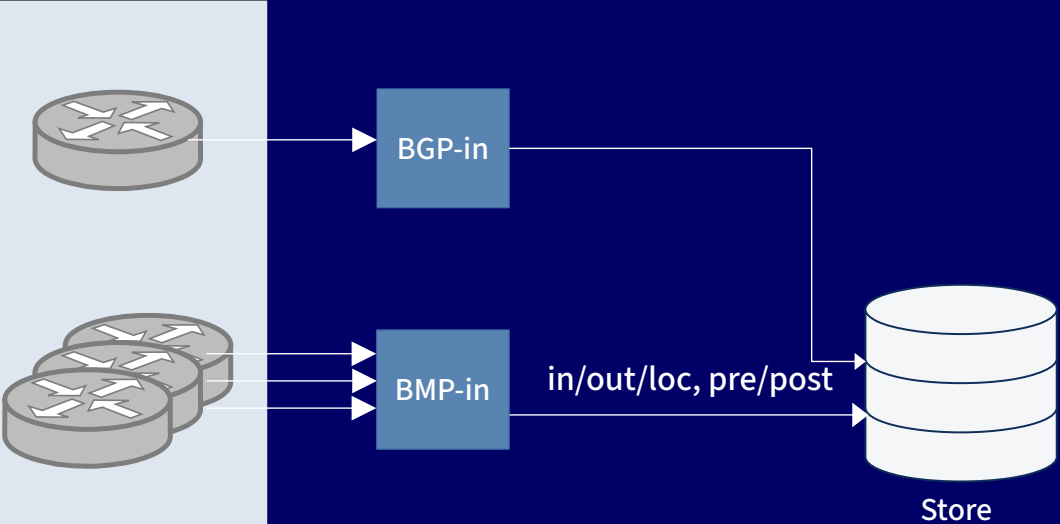


in/out/loc, pre/post

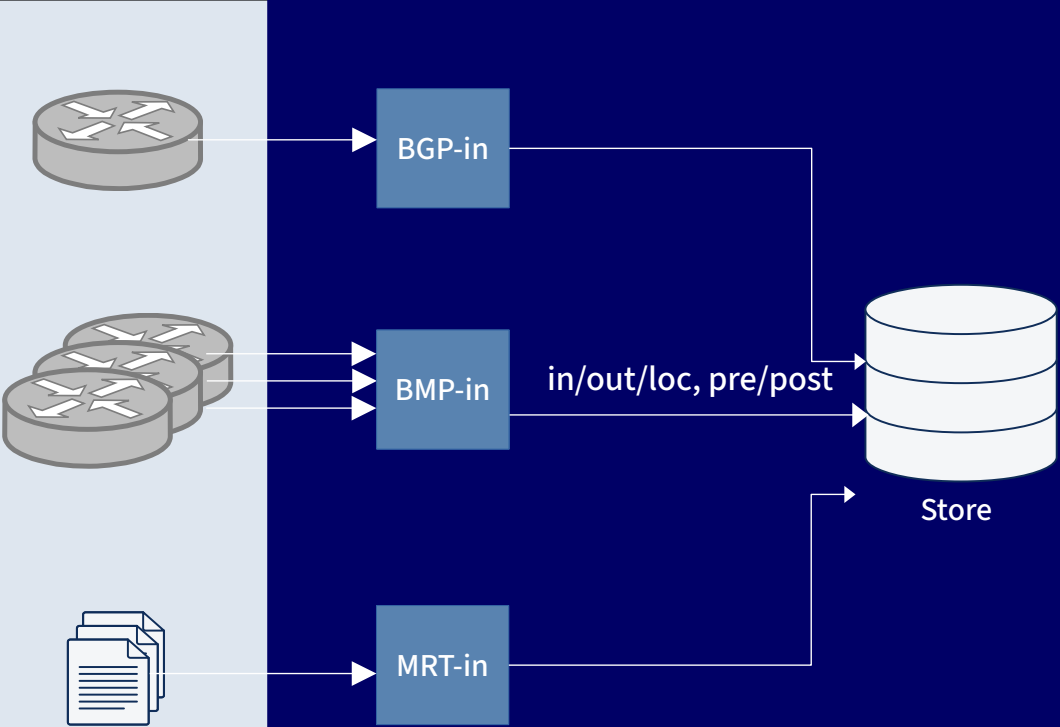


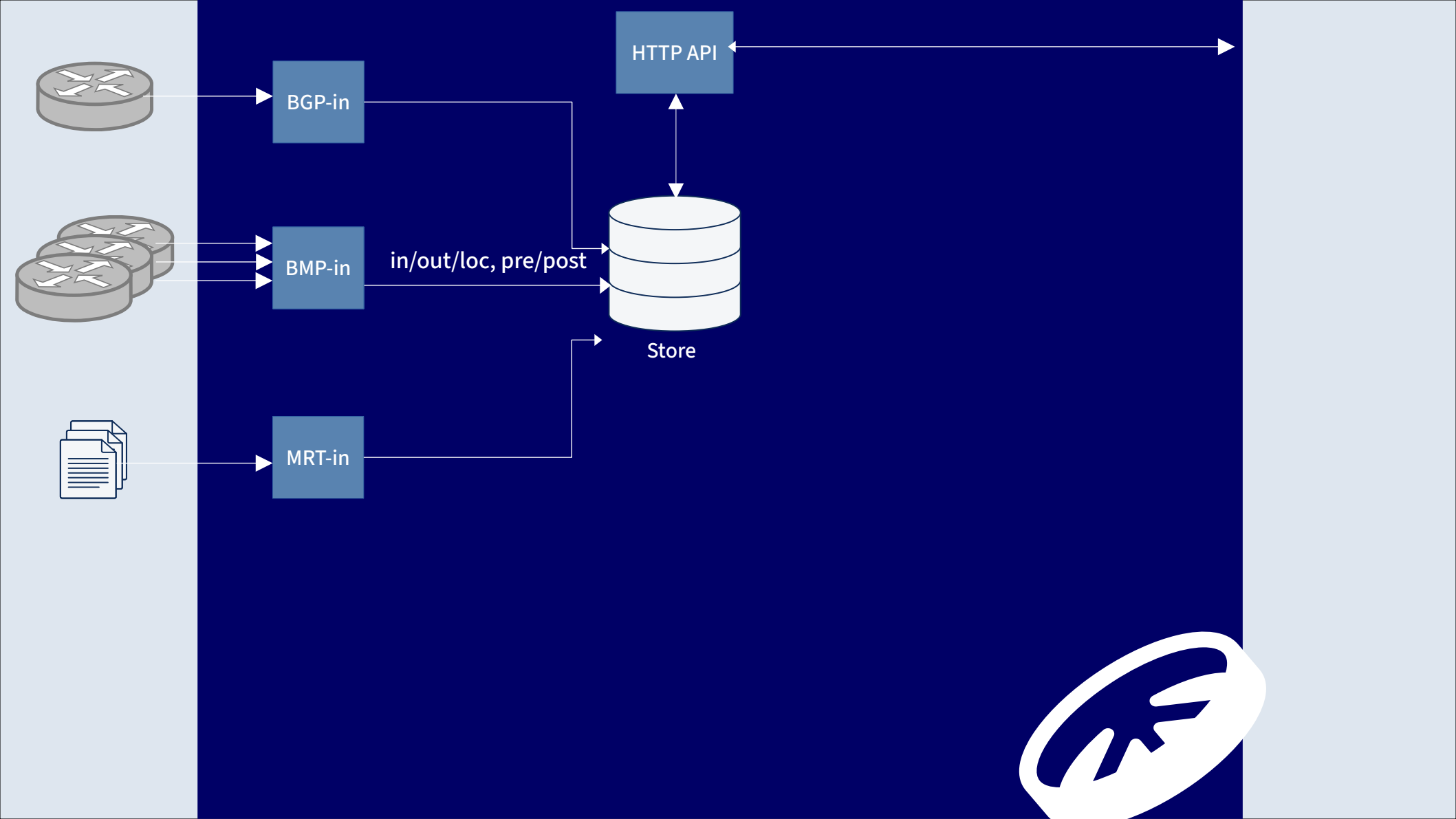
Store

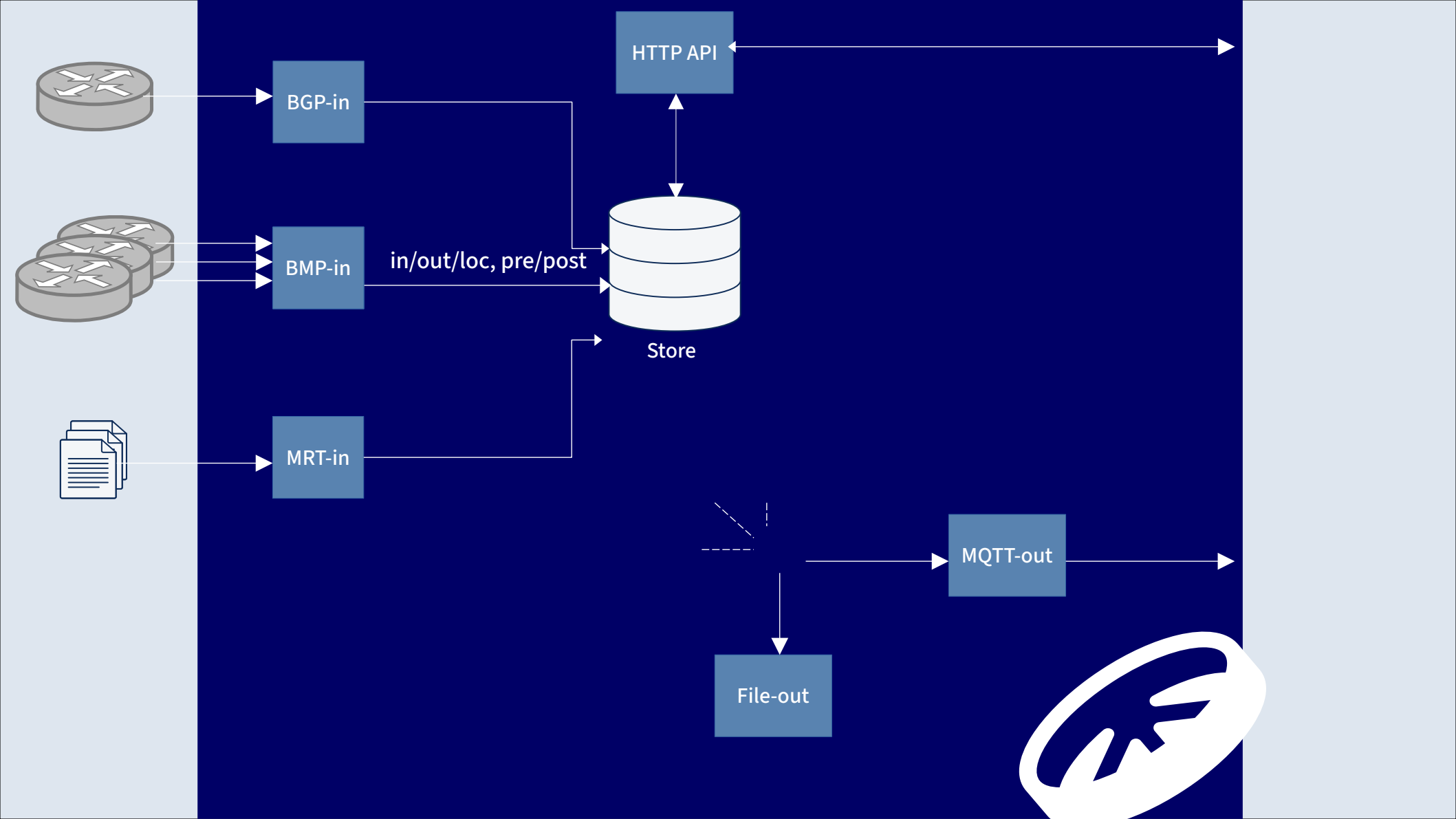


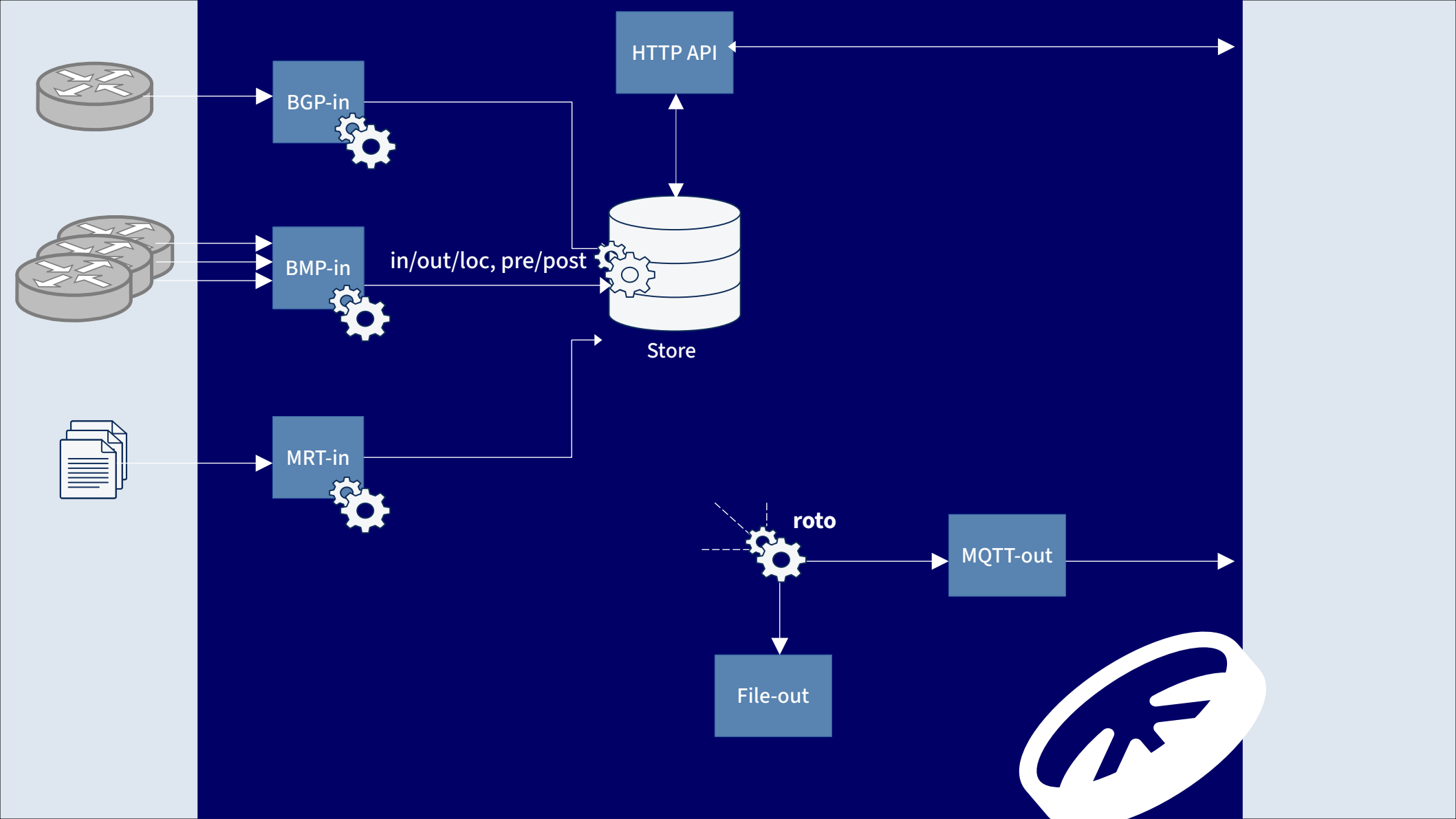












# Introducing Roto

# Introducing Roto



# Roto: high-level feel, strongly typed, compiled

## Syntax highlighting!

```
filter bmp_in (bmp_msg: BmpMsg, ingress_info: IngressInfo) {  
    let my_asn = AS221321;  
  
    if bmp_msg.is_ibgp(my_asn) {  
        # Don't store iBGP  
        reject  
    } else {  
        accept  
    }  
  
}
```

# Roto: high-level feel, strongly typed, compiled

## Type checking and useful error messages!

```
filter bmp_in(bmp_msg: BmpMsg, ingress_info: IngressInfo) {  
    let my_asn = 185.49.140.0/23; # <-- oops  
  
    if bmp_msg.is_ibgp(my_asn) {  
        reject  
    } else {  
        accept  
    }  
  
}
```



# Roto: high-level feel, strongly typed, compiled

## Type checking and useful error messages!

```
filter bmp_in(bmp_msg: BmpMsg, ingress_info: IngressInfo) {  
    let my_asn = 185.49.140.0/23; # <-- oops  
  
    if bmp_msg.is_ibgp(my_asn) {  
        reject  
    } else {  
        accept  
    }  
}
```

```
rotonda::manager: Unable to load main Roto script: Error: Type error: mismatched types  
[ /home/luuk/code/rotonda/axum-api/etc/filters.roto:81:24 ]  
  
81     if bmp_msg.is_ibgp(my_asn) {  
                                                 
                               expected `Asn`, found `Prefix`  
.
```

# Roto: high-level feel, strongly typed, compiled

## Compiled!

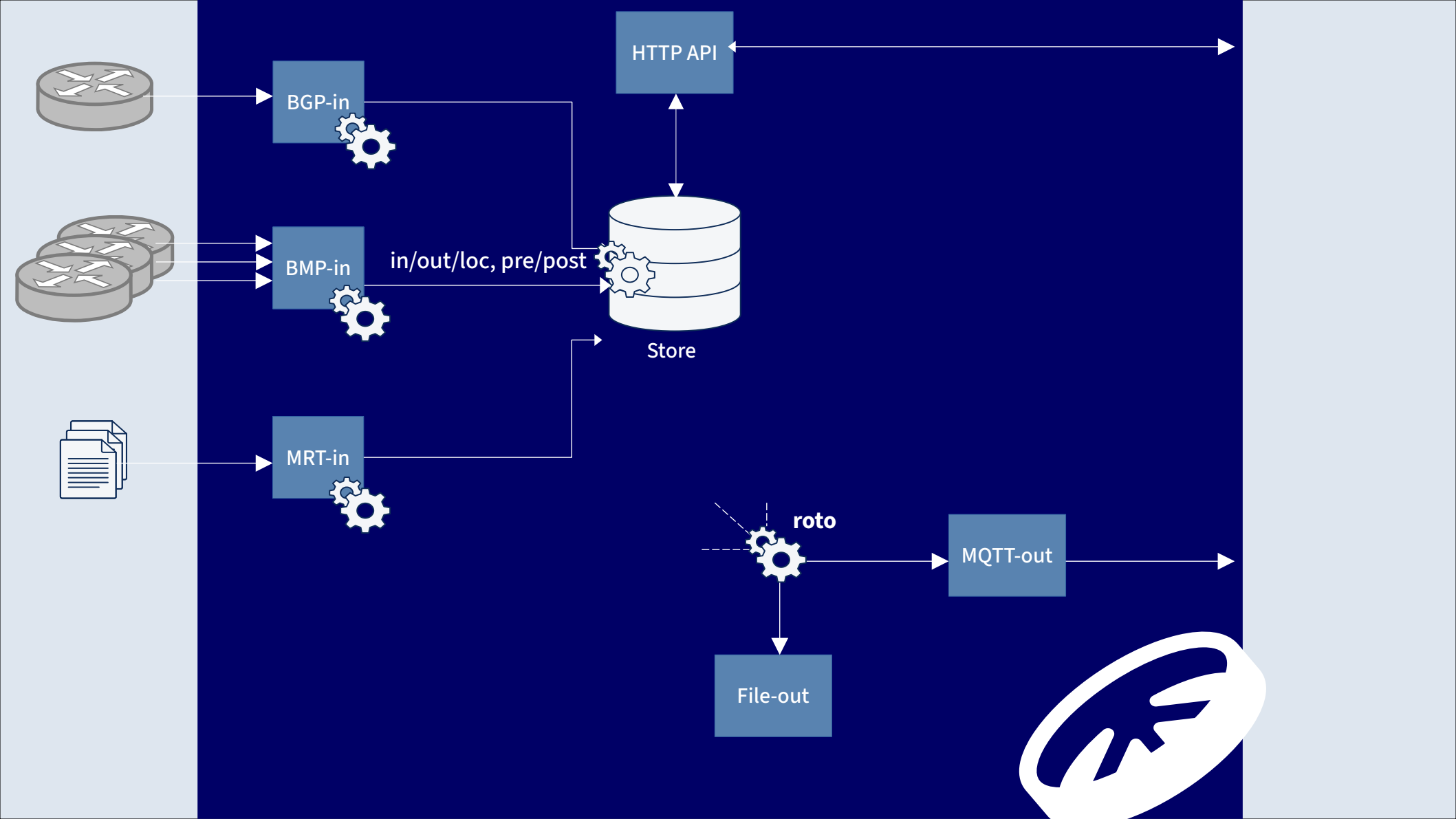
```
filter bmp_in (bmp_msg: BmpMsg, ingress_info: IngressInfo) {  
    let my_asn = AS221321;  
  
    if bmp_msg.is_ibgp(my_asn) {  
        # Don't store iBGP  
        reject  
    } else {  
        accept  
    }  
  
}
```

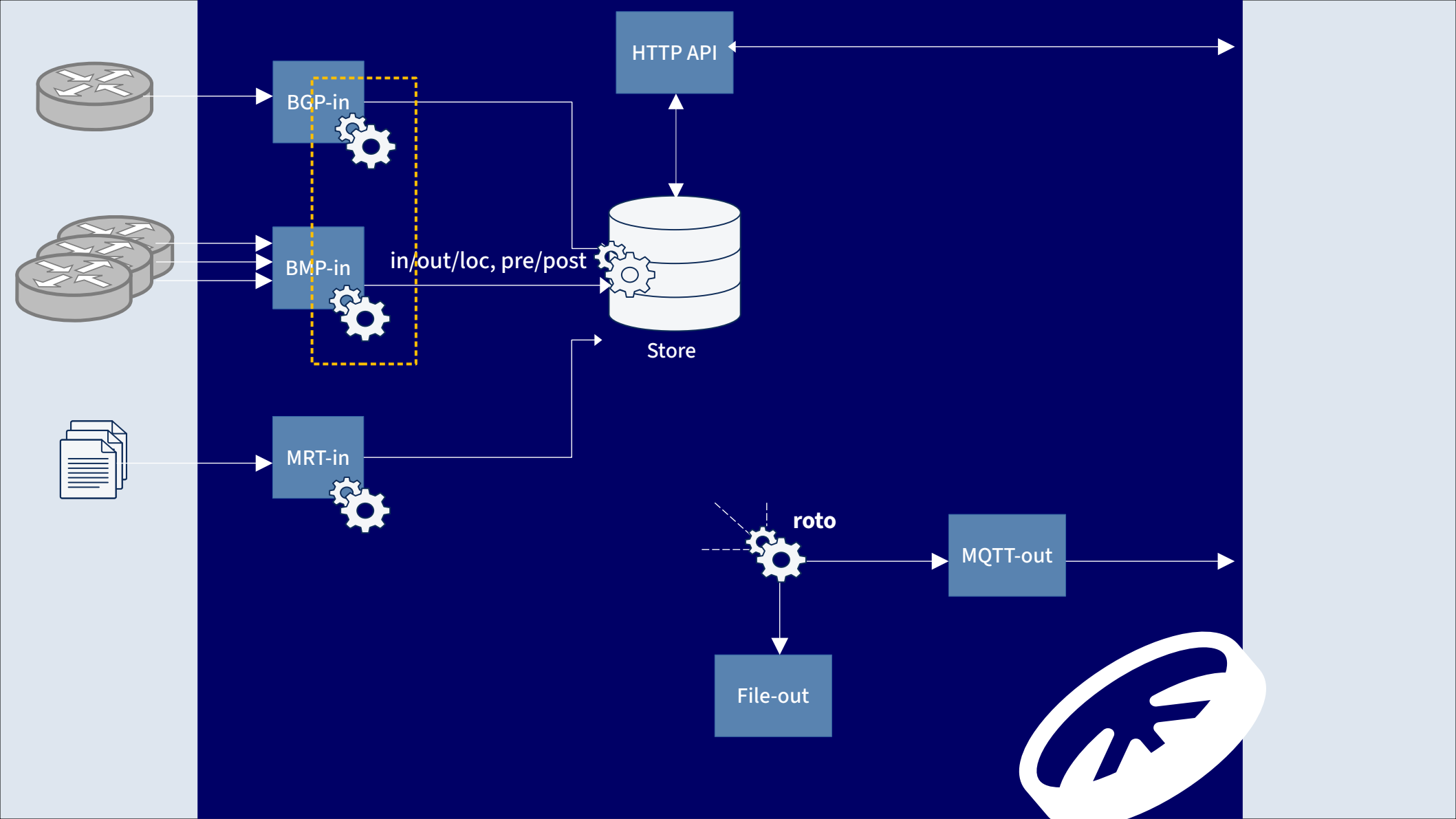
# Roto: high-level feel, strongly typed, compiled

## Compiled! From Roto to Cranelift IR,

```
filter bmp_in (bmp_msg: BmpMsg, ingress_info: IngressInfo) {  
  block5:  
    l v28 = iadd_imm.i64 v27, 4  
    v29 = load.i32 aligned v28  
    i v47 -> v29  
    jump block6  
  block6:  
    v32 = iconst.i64 0x558f_2cfd_0ab0  
    call_indirect sig1, v32(v31, v30) ; v32 = 0x558f_2cfd_  
    call fn1(v33, v30)  
    v35 = load.i64 aligned v33  
    v36 = load.i64 aligned v33+8  
    v37 = load.i64 aligned v33+16  
    v38 = load.i64 aligned v33+24  
    store aligned v35, v34  
    store aligned v36, v34+8  
    store aligned v37, v34+16  
    store aligned v38, v34+24  
}
```







```
filter bgp_in(  
    bgp_msg: BgpMsg,  
    ingress_info: IngressInfo,  
) {  
  
    let origin_to_log = AS65536;  
    let community_to_log = NO_PEER;  
  
    if bgp_msg.match_aspath_origin(origin_to_log) {  
        output.log_matched_origin(origin_to_log);  
    }  
    if bgp_msg.contains_community(community_to_log) {  
        output.log_matched_community(community_to_log)  
    }  
  
    accept  
}
```

```
filter bmp_in(  
    bmp_msg: BmpMsg,  
    ingress_info: IngressInfo,  
) {  
    let my_asn = AS12345;  
    let asn_to_log = AS65536;  
    let community_to_log = community(0xffff029a);  
  
    if bmp_msg.is_peer_down() {  
        output.log_peer_down()  
    }  
  
    if bmp_msg.is_ibgp(my_asn) {  
        reject  
    } else {  
        if bmp_msg.aspath_contains(asn_to_log) {  
            output.log_matched_asn(asn_to_log);  
        }  
    }  
}
```

```
filter bgp_in {
  bgp_msg: BgpMsg,
  ingress_info: IngressInfo,
} {

  let origin_to_log = AS65536;
  let community_to_log = NO_PEER;

  if bgp_msg.match_aspath_origin(origin_to_log) {
    output.log_matched_origin(origin_to_log);
  }
  if bgp_msg.contains_community(community_to_log) {
    output.log_matched_community(community_to_log)
  }

  accept
}
```

```
filter bmp_in {
  bmp_msg: BmpMsg,
  ingress_info: IngressInfo,
} {

  let my_asn = AS12345;
  let asn_to_log = AS65536;
  let community_to_log = community(0xffff029a);

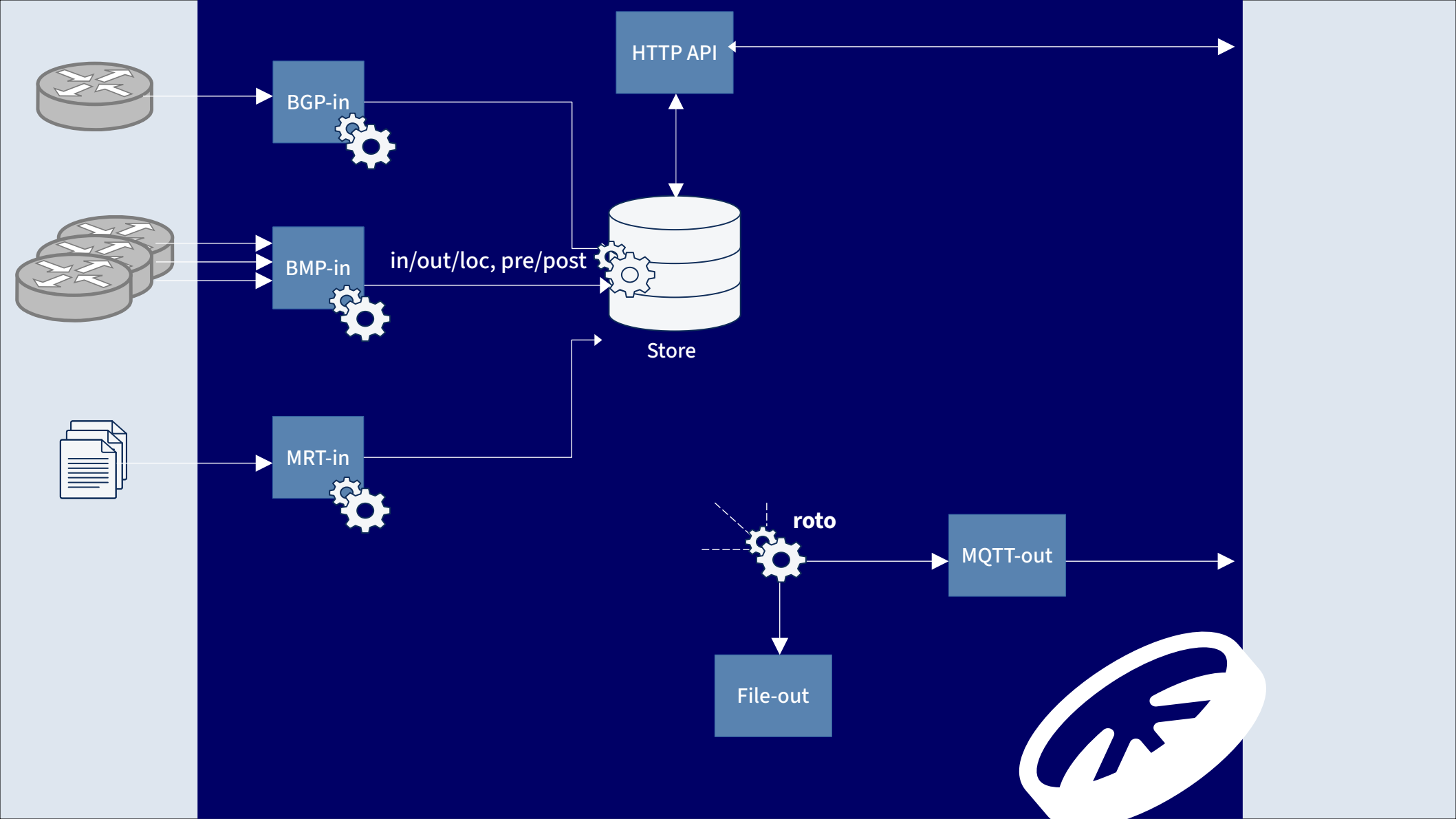
  if bmp_msg.is_peer_down() {
    output.log_peer_down()
  }

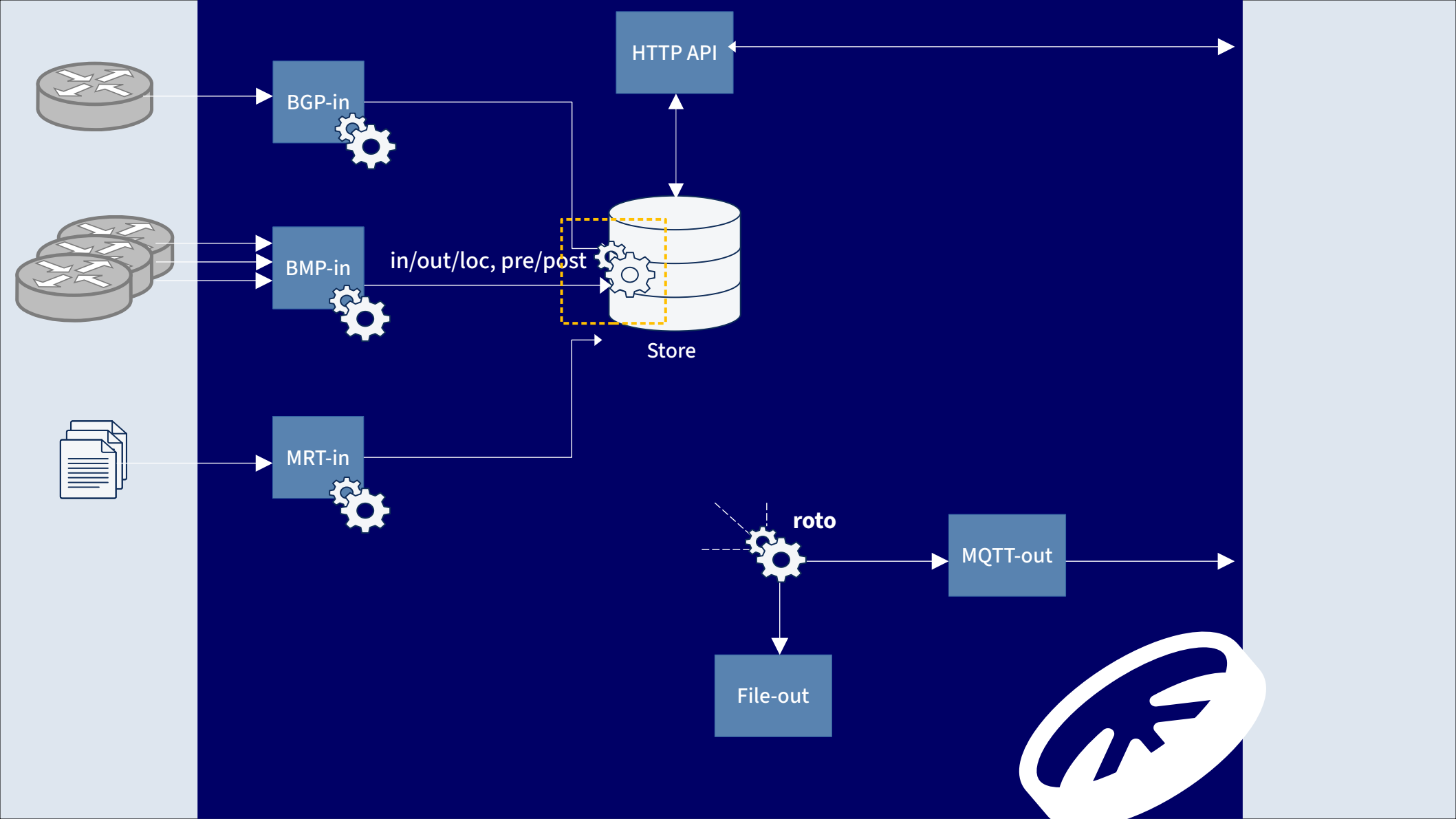
  if bmp_msg.is_ibgp(my_asn) {
    reject
  } else {
    if bmp_msg.aspath_contains(asn_to_log) {
      output.log_matched_asn(asn_to_log);
    }
  }
}
```



```
filter bgp_in(  
  bgp_msg: BgpMsg,  
  ingress_info: IngressInfo,  
) {  
  
  let origin_to_log = AS65536;  
  let community_to_log = NO_PEER;  
  
  if bgp_msg.match_aspath_origin(origin_to_log) {  
    output.log_matched_origin(origin_to_log);  
  }  
  if bgp_msg.contains_community(community_to_log) {  
    output.log_matched_community(community_to_log)  
  }  
  
  accept  
}
```

```
filter bmp_in(  
  bmp_msg: BmpMsg,  
  ingress_info: IngressInfo,  
) {  
  let my_asn = AS12345;  
  let asn_to_log = AS65536;  
  let community_to_log = community(0xffff029a);  
  
  if bmp_msg.is_peer_down() {  
    output.log_peer_down()  
  }  
  
  if bmp_msg.is_ibgp(my_asn) {  
    reject  
  } else {  
    if bmp_msg.aspath_contains(asn_to_log) {  
      output.log_matched_asn(asn_to_log);  
    }  
  }  
}
```

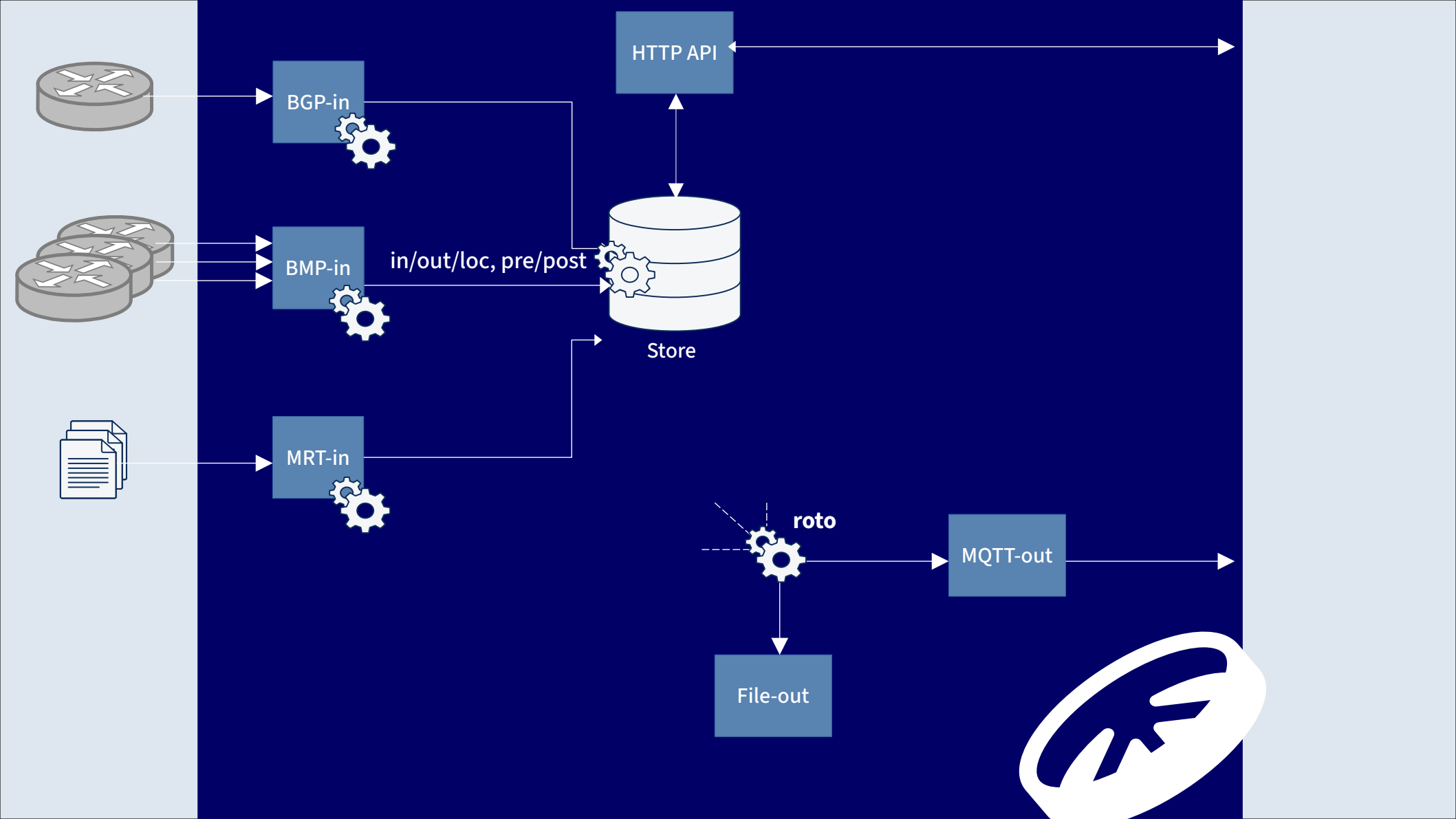


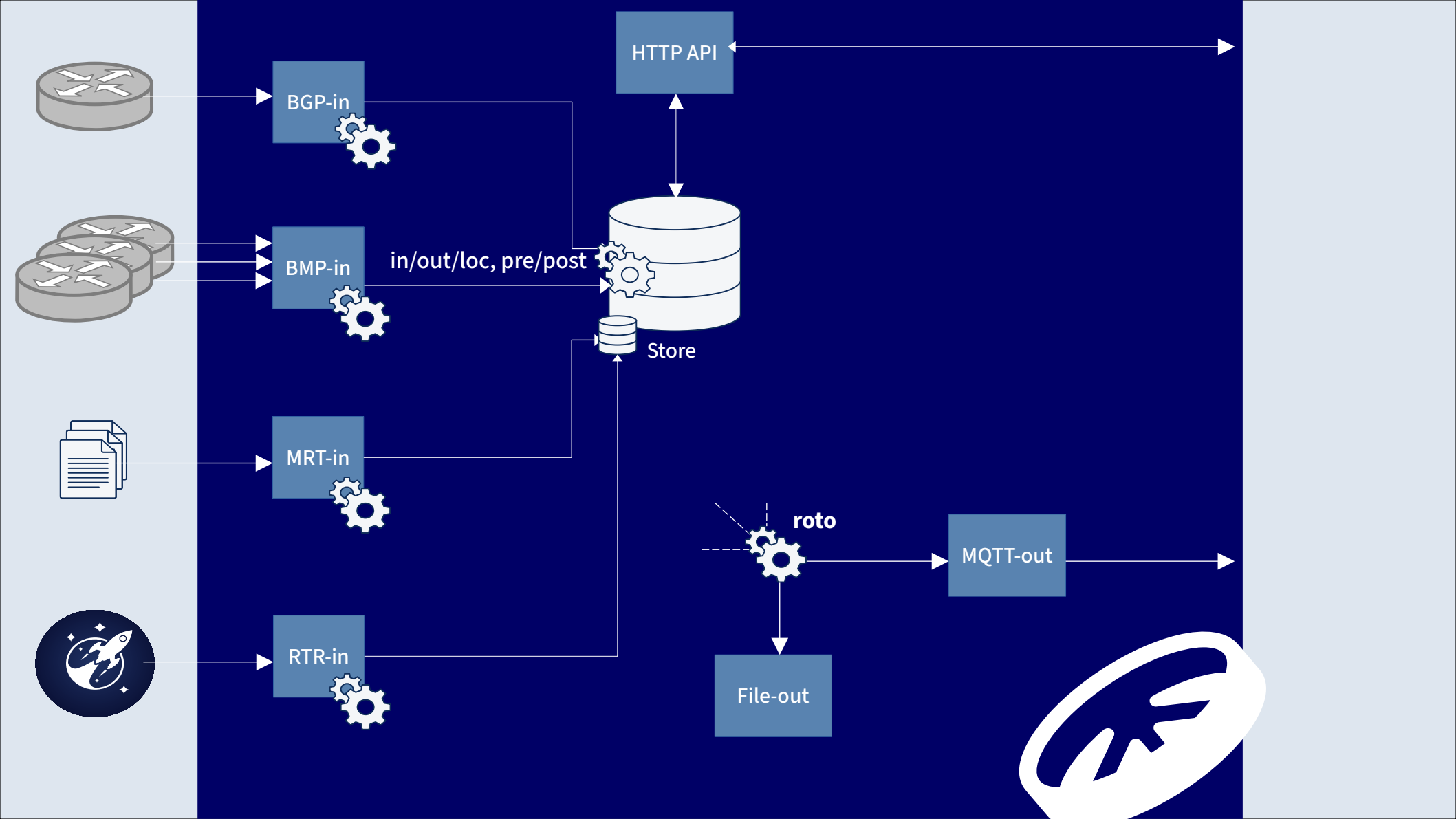


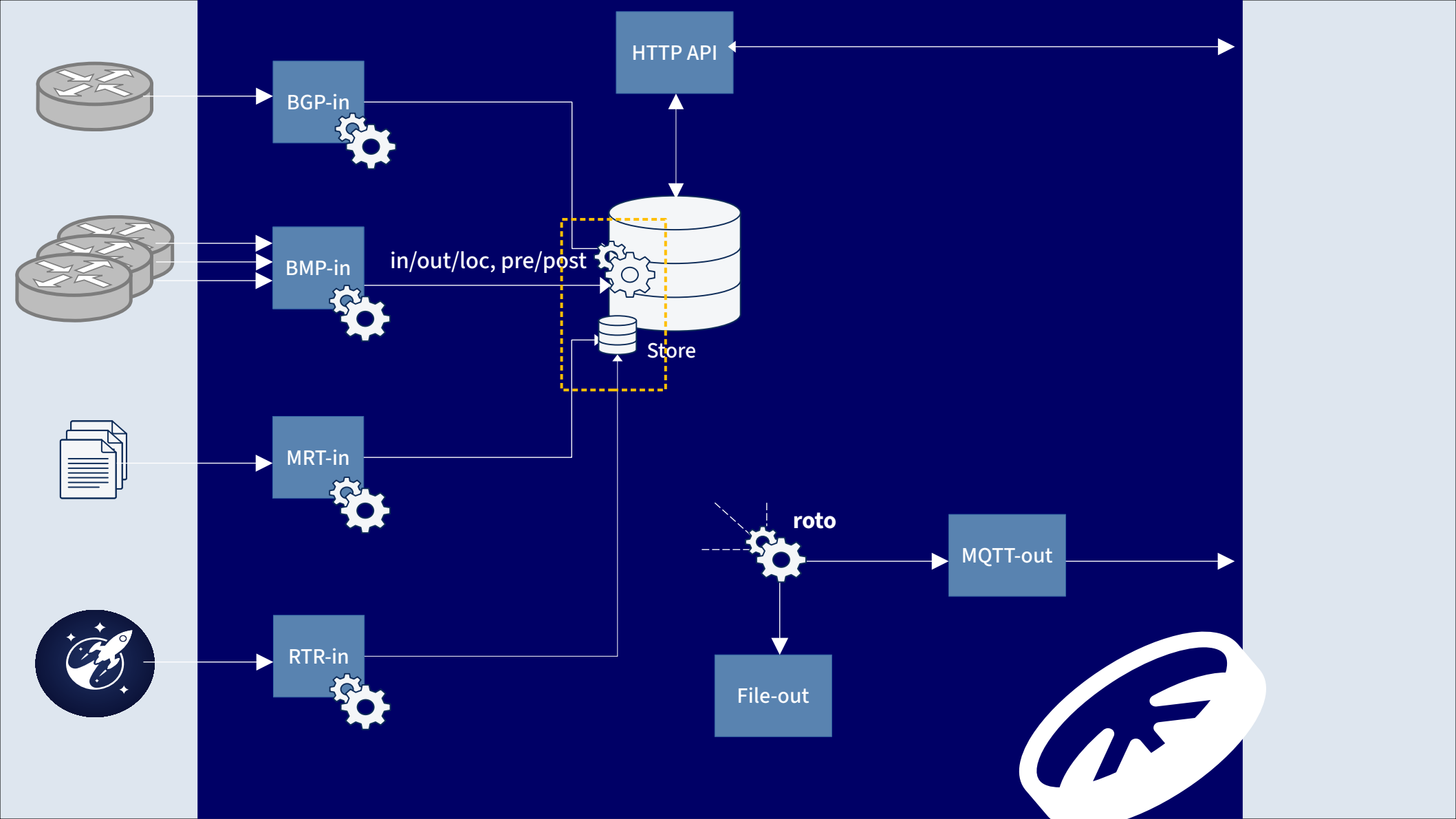
```
filter rib_in_pre(  
    route: Route,  
    ingress_info: IngressInfo,  
) {  
  
    if prefix_lists.contains("my_prefixes", route.prefix()) {  
        output.log_prefix(route.prefix());  
    }  
  
    accept  
}
```

```
filter rib_in_pre(
    route: Route,
    ingress_info: IngressInfo,
) {
    if prefix_lists.contains("my_prefixes", route.prefix()) {
        output.log_prefix(route.prefix());
    }

    accept
}
```









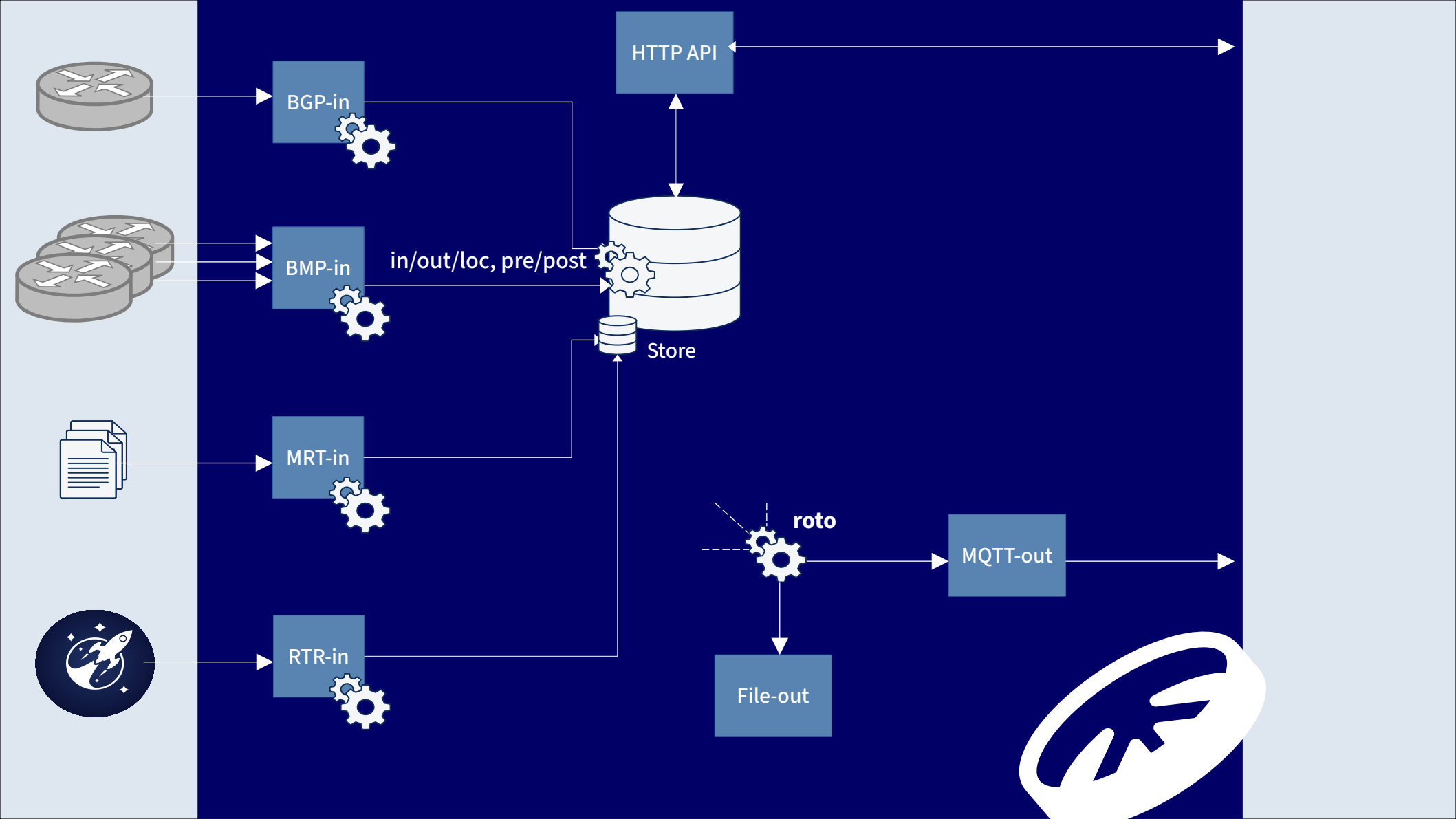
```
filter rib_in_pre(
    route: Route,
    ingress_info: IngressInfo,
) {
    if prefix_lists.contains("my_prefixes", route.prefix()) {
        output.log_prefix(route.prefix());
    }

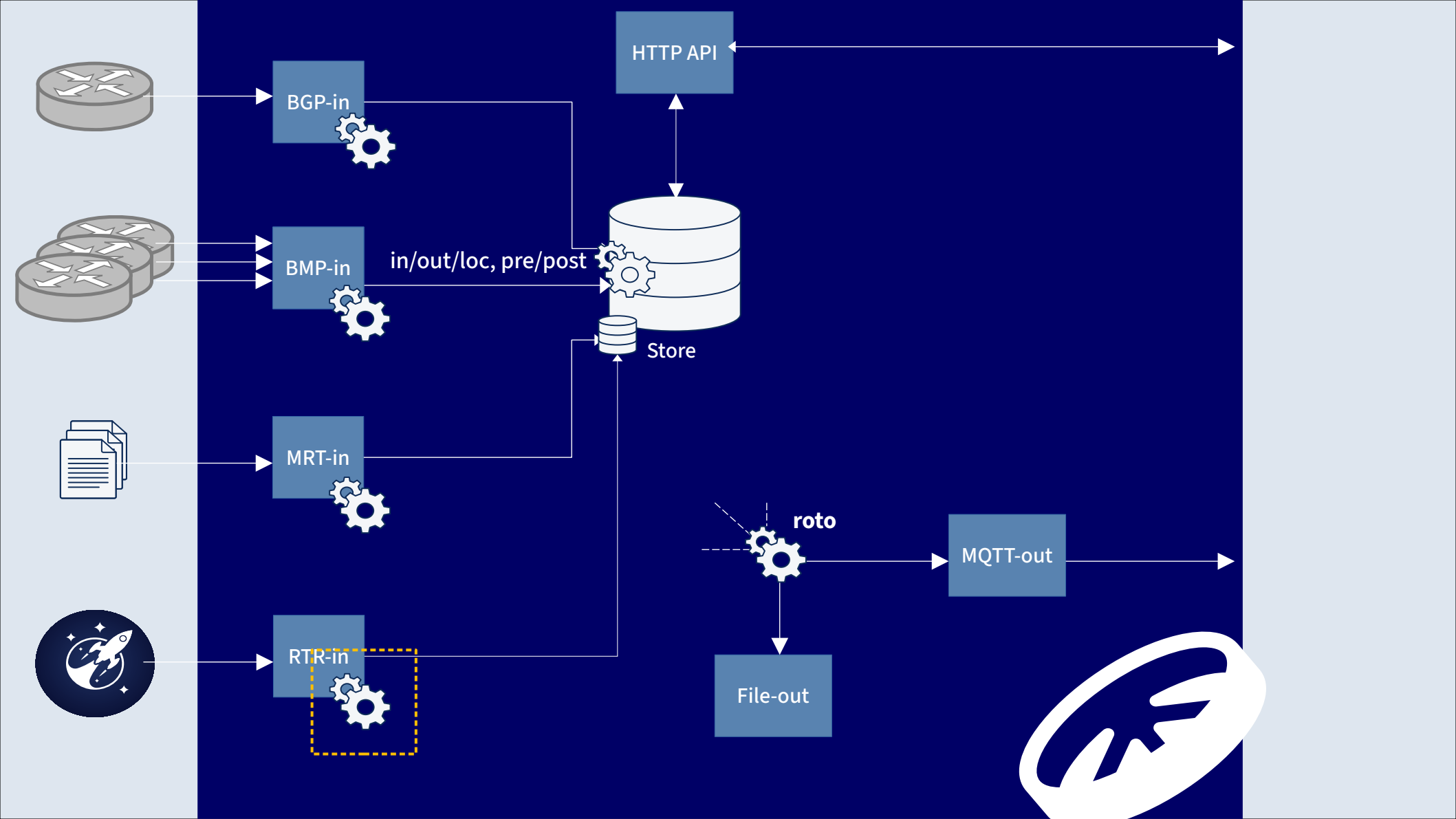
    accept
}
```

```
filter rib_in_pre(
    route: Route,
    ingress_info: IngressInfo,
) {
    if prefix_lists.contains("my_prefixes", route.prefix()) {
        output.log_prefix(route.prefix());
    }

    # Only has effect when an RTR unit is configured in rotonda.conf
    rpki.check_rov(route);

    accept
}
```





```
filter vrp_update(vrp_update: VrpUpdate) {  
    if asn_lists.contains("my_asns", vrp_update.asn()) {  
        output.entry().timestamped_custom("[RTR] (my asns): " + vrp_update.fmt());  
        output.write_entry();  
    }  
  
    if prefix_lists.covers("my_prefixes", vrp_update.prefix()) {  
        output.timestamped_print("[RTR] (my prefixes): " + vrp_update.fmt())  
    }  
  
    accept  
}
```

<billy\_mays.jpg>

```
fn rib_in_rov_status_update(rov_update: RovStatusUpdate) {  
    if rov_update.has_changed() {  
        # Only log whenever something changed to Invalid:  
        if rov_update.current_status().is_invalid() {  
            output.timestamped_print("[RTR]" + rov_update.fmt())  
        }  
    }  
    # When it concerns any of 'my_prefixes', always log:  
    if prefix_lists.covers("my_prefixes", rov_update.prefix()) {  
        output.timestamped_print("[RTR] (my prefixes): " + rov_update.fmt())  
    }  
}
```

```
fn rib_in_rov_status_update(rov_update: RovStatusUpdate) {  
    if rov_update.has_changed() {  
        # Only log whenever something changed to Invalid:  
        if rov_update.current_status().is_invalid() {  
            output.timestamped_print("[RTR]" + rov_update.fmt())  
        }  
    }  
    # When it concerns any of 'my_prefixes', always log:  
    if prefix_lists.covers("my_prefixes", rov_update.prefix()) {  
        output.timestamped_print("[RTR] (my prefixes): " + rov_update.fmt())  
    }  
}
```



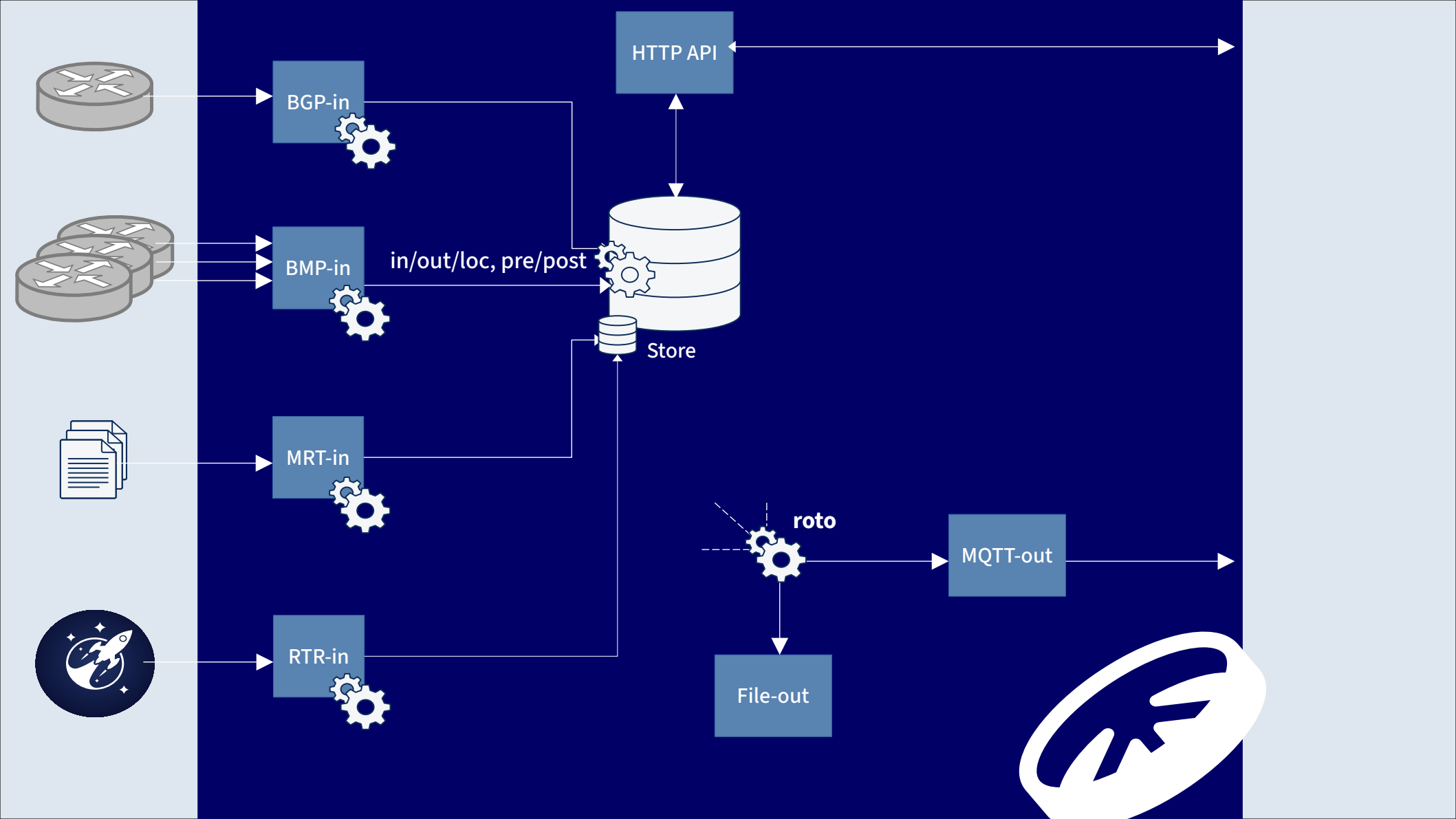
```
fn rib_in_rov_status_update(rov_update: RovStatusUpdate) {  
    if rov_update.has_changed() {  
        # Only log whenever something changed to Invalid:  
        if rov_update.current_status().is_invalid() {  
            output.timestamped_print("[RTR]" + rov_update.fmt())  
        }  
    }  
    # When it concerns any of 'my_prefixes', always log:  
    if prefix_lists.covers("my_prefixes", rov_update.prefix()) {  
        output.timestamped_print("[RTR] (my prefixes): " + rov_update.fmt())  
    }  
}
```

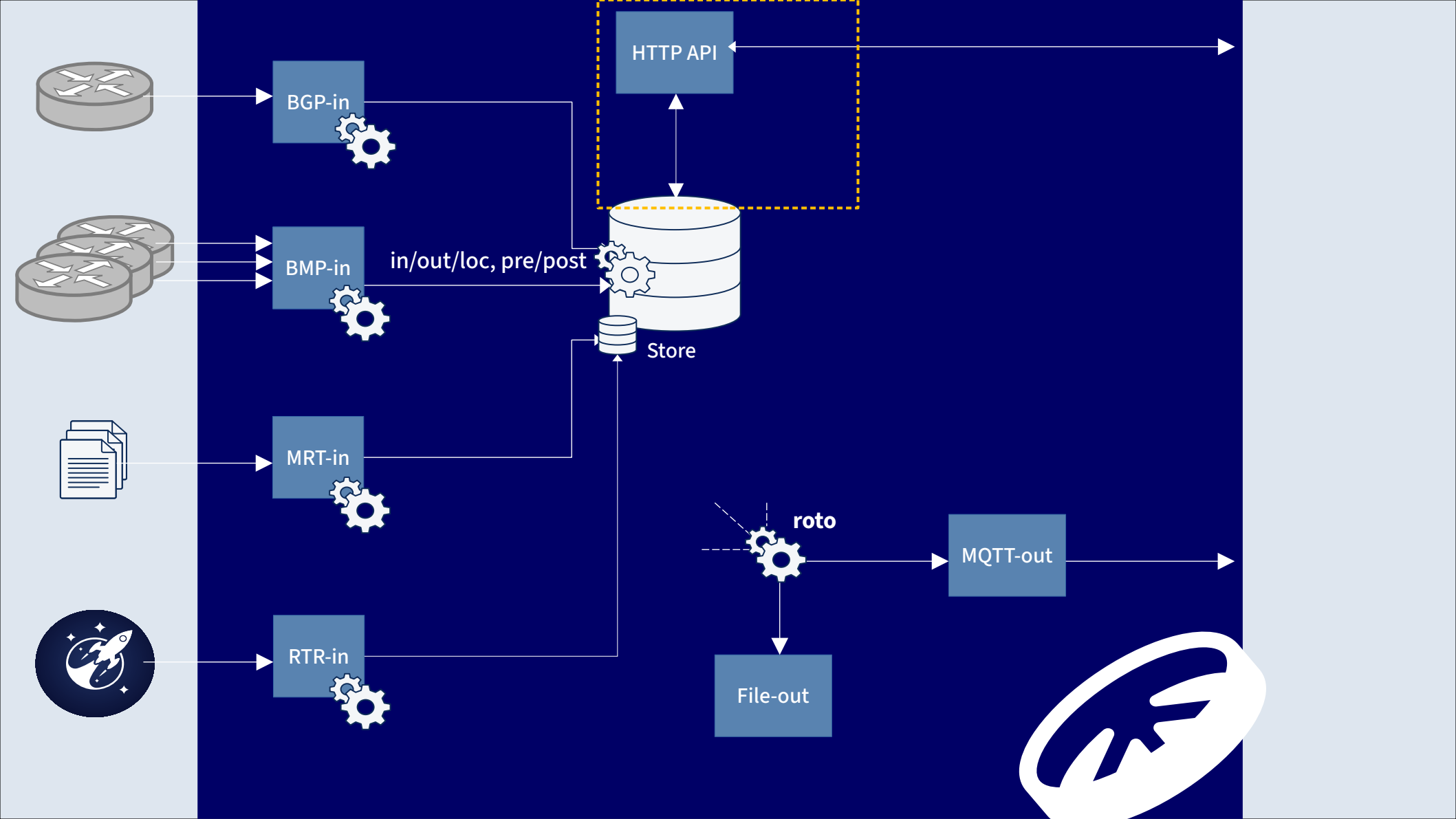
```
fn rib_in_rov_status_update(rov_update: RovStatusUpdate) {  
    if rov_update.has_changed() {  
        # Only log whenever something changed to Invalid:  
        if rov_update.current_status().is_invalid() {  
            output.timestamped_print("[RTR]" + rov_update.fmt())  
        }  
    }  
    # When it concerns any of 'my_prefixes', always log:  
    if prefix_lists.covers("my_prefixes", rov_update.prefix()) {  
        output.timestamped_print("[RTR] (my prefixes): " + rov_update.fmt())  
    }  
}
```

```
fn rib_in_rov_status_update(rovd_update: RovStatusUpdate) {  
    if rovd_update.has_changed() {  
        # Only log whenever something changed to Invalid:  
        if rovd_update.current_status().is_invalid() {  
            output.timestamped_print("[RTR]" + rovd_update.fmt())  
        }  
    }  
    # When it concerns any of 'my_prefixes', always log:  
    if prefix_lists.covers("my_prefixes", rovd_update.prefix()) {  
        output.timestamped_print("[RTR] (my prefixes): " + rovd_update.fmt())  
    }  
}
```

```
fn rib_in_rov_status_update(rovd_update: RovStatusUpdate) {  
    if rovd_update.has_changed() {  
        # Only log whenever something changed to Invalid:  
        if rovd_update.current_status().is_invalid() {
```

```
rotonda::units::rib_unit::unit: got RTR Serial update  
rotonda::units::rib_unit::unit: pushed VRP for 40.223.13.0/24-24 from AS214025  
rotonda::units::rib_unit::unit: removed VRP for 40.223.12.0/22-24 from AS834  
rotonda::units::rib_unit::unit: pushed VRP for 66.93.41.0/24-24 from AS214025  
rotonda::units::rib_unit::unit: pushed VRP for 66.212.31.0/24-24 from AS214025  
[2025-09-27T07:51:22Z] [RTR][NotFound] -> [Invalid] 66.212.31.0/24 originated by AS834, learned from AS200242  
[2025-09-27T07:51:22Z] [RTR][NotFound] -> [Invalid] 66.212.31.0/24 originated by AS834, learned from AS213151  
[2025-09-27T07:51:22Z] [RTR][NotFound] -> [Invalid] 66.212.31.0/24 originated by AS834, learned from AS3214  
[2025-09-27T07:51:22Z] [RTR][NotFound] -> [Invalid] 66.212.31.0/24 originated by AS834, learned from AS8888  
[2025-09-27T07:51:22Z] [RTR][NotFound] -> [Invalid] 66.212.31.0/24 originated by AS834, learned from AS60150  
[2025-09-27T07:51:22Z] [RTR][NotFound] -> [Invalid] 66.212.31.0/24 originated by AS834, learned from AS51019  
[2025-09-27T07:51:22Z] [RTR][NotFound] -> [Invalid] 66.212.31.0/24 originated by AS834, learned from AS52025
```





# Getting JSON out of Rotonda

- `/api/v1/ingresses[?filter[type]=bgp|bmp|...]`  
Query for *ingresses* and their session information
- `/api/v1/ribs/ipv4unicast/routes`  
`/api/v1/ribs/ipv4unicast/routes/185.49.140.0/23`  
Query for stored routes per address family

# Thank you RouteViews!

← → ↻ 🔒 🖨️ www.routeviews.org/routeviews/ 📄 📶 ☆



[HOME](#) [ABOUT](#) [FAQ](#) [PEERING](#) [TOOLS](#) [COLLECTORS](#) [MAP](#)  
[SUPPORTERS](#) [BLOG](#) [CONTACT](#)

## University of Oregon RouteViews Project

The University's RouteViews project was originally conceived as a tool for Internet operators to obtain real-time BGP information about the global routing system from the perspectives of several different backbones and locations around the Internet. Although other tools handle related tasks, such as the various Looking Glass Collections (see e.g. [TRACEROUTE.ORG](#)), they typically either provide only a constrained view of the routing system (e.g., either a single provider, or the route server) or they do not provide real-time access to routing data.

While the RouteViews project was originally motivated by interest on the part of operators in determining how the global routing system viewed *their* prefixes and/or AS space, there have been many other interesting uses of this RouteViews data. For example, NLNR has used RouteViews data for AS path visualization and to study IPv4 address space utilization ([archive](#)). Others have used RouteViews data to map IP addresses to origin AS for various topological studies. CAIDA has used it in conjunction with the [NetGeo](#) database in generating geographic locations for hosts, functionality that both [CoralReef](#) and the [Skitter](#) project support.





```
{
  "meta": null,
  "data": {
    "nlri": "185.49.140.0/23",
    "routes": [
      {
        "status": "active",
        "ingress": {
          "id": 39,
          "ingress_type": "bgpViaBmp",
          "parent_ingress": 3,
          "remote_addr": "185.1.167.88",
          "remote_asn": 51019,
          "rib_type": "AdjRibIn",
          "peer_rib_type": "inPost",
          "peer_type": "GlobalInstance"
        },
        "rpki": {
          "rov": "valid"
        },
        "pathAttributes": [
          {
            "origin": "Igp"
          },
          {
            "asPath": [
              "AS6447",
              "AS51019",
              "AS34927",
              "AS50673",
              "AS8587"
            ]
          },
          {
            "conventionalNextHop": "185.1.167.88"
          },
          {
            "standardCommunities": [
              "AS34927:930",
              "AS51019:1101"
            ]
          }
        ]
      }
    ]
  }
}
```



```
{
  "meta": null,
  "data": {
    "nlri": "185.49.140.0/23",
    "routes": [
      {
        "status": "active",
        "ingress": {
          "id": 39,
          "ingress_type": "bgpViaBmp",
          "parent_ingress": 3,
          "remote_addr": "185.1.167.88",
          "remote_asn": 51019,
          "rib_type": "AdjRibIn",
          "peer_rib_type": "inPost",
          "peer_type": "GlobalInstance"
        },
        "rpki": {
          "rov": "valid"
        },
        "pathAttributes": [
          {
            "origin": "Igp"
          },
          {
            "asPath": [
              "AS6447",
              "AS51019",
              "AS34927",
              "AS50673",
              "AS8587"
            ]
          },
          {
            "conventionalNextHop": "185.1.167.88"
          },
          {
            "standardCommunities": [
              "AS34927:930",
              "AS51019:1101"
            ]
          }
        ]
      }
    ]
  }
}
```



```
{
  "meta": null,
  "data": {
    "nlri": "185.49.140.0/23",
    "routes": [
      {
        "status": "active",
        "ingress": {
          "id": 39,
          "ingress_type": "bgpViaBmp",
          "parent_ingress": 3,
          "remote_addr": "185.1.167.88",
          "remote_asn": 51019,
          "rib_type": "AdjRibIn",
          "peer_rib_type": "inPost",
          "peer_type": "GlobalInstance"
        },
        "rpki": {
          "rov": "valid"
        },
        "pathAttributes": [
          {
            "origin": "Igp"
          },
          {
            "asPath": [
              "AS6447",
              "AS51019",
              "AS34927",
              "AS50673",
              "AS8587"
            ]
          },
          {
            "conventionalNextHop": "185.1.167.88"
          },
          {
            "standardCommunities": [
              "AS34927:930",
              "AS51019:1101"
            ]
          }
        ]
      }
    ]
  }
}
```



```
{
  "meta": null,
  "data": {
    "nlri": "185.49.140.0/23",
    "routes": [
      {
        "status": "active",
        "ingress": {
          "id": 39,
          "ingress_type": "bgpVia8mp",
          "parent_ingress": 3,
          "remote_addr": "185.1.167.88",
          "remote_asn": 51019,
          "rib_type": "AdjRibIn",
          "peer_rib_type": "inPost",
          "peer_type": "GlobalInstance"
        },
        "rpki": {
          "rov": "valid"
        },
        "pathAttributes": [
          {
            "origin": "Igp"
          },
          {
            "asPath": [
              "AS6447",
              "AS51019",
              "AS34927",
              "AS50673",
              "AS8587"
            ]
          }
        ],
        "conventionalNextHop": "185.1.167.88"
      },
      {
        "standardCommunities": [
          "AS34927:930",
          "AS51019:1101"
        ]
      }
    ]
  }
}
```



```
{
  "meta": null,
  "data": {
    "nlri": "185.49.140.0/23",
    "routes": [
      {
        "status": "active",
        "ingress": {
          "id": 39,
          "ingress_type": "bgpVia8mp",
          "parent_ingress": 3,
          "remote_addr": "185.1.167.88",
          "remote_asn": 51019,
          "rib_type": "AdjRibIn",
          "peer_rib_type": "inPost",
          "peer_type": "GlobalInstance"
        },
        "rpki": {
          "rov": "valid"
        },
        "pathAttributes": [
          {
            "origin": "Igp"
          },
          {
            "asPath": [
              "AS6447",
              "AS51019",
              "AS34927",
              "AS50673",
              "AS8587"
            ]
          },
          {
            "conventionalNextHop": "185.1.167.88"
          },
          {
            "standardCommunities": [
              "AS34927:930",
              "AS51019:1101"
            ]
          }
        ]
      }
    ]
  }
}
```

# Getting JSON out of Rotonda: simple filtering

- `/api/v1/ribs/ipv4unicast/routes/145.100.0.0/15`  
  
`?include=moreSpecifics, # include, you know, more specifics`  
  
`&filter[ribType]=inPre # routes in AdjRibIn pre-policy`  
`&filter[rovStatus]=valid # && only ROV valid`  
`&filter[originAsn]=1133 # && originated by AS1133`  
`&filter[community]=AS1103:9876 # && and contains this community`

# Getting JSON out of Rotonda: simple filtering

- `/api/v1/ribs/ipv4unicast/routes/145.100.0.0/15`  
  
`?include=moreSpecifics, # include, you know, more specifics`  
  

|                                                 |                                                       |
|-------------------------------------------------|-------------------------------------------------------|
| <code>&amp;filter[ribType]=inPre</code>         | <code># routes in AdjRibIn pre-policy</code>          |
| <code>&amp;filter[rovStatus]=valid</code>       | <code># &amp;&amp; only ROV valid</code>              |
| <code>&amp;filter[originAsn]=1133</code>        | <code># &amp;&amp; originated by AS1133</code>        |
| <code>&amp;filter[community]=AS1103:9876</code> | <code># &amp;&amp; and contains this community</code> |

All filters are boolean AND'd

(More filters, includes and other options are described in the docs)

# Filtering JSON responses using Roto

- `/api/v1/ribs/ipv4unicast/routes`

`?include=moreSpecifics, # on /routes, this is default`

`&function[roto]=hidden_hop # use a roto function to filter`



```
filter hidden_hop(attr: PathAttributes) {  
  match attr.otc() {  
    Some(otc) -> {  
      match attr.aspath() {  
        Some(asp) -> {  
          if not aspath.contains(otc) {  
            accept  
          } else {  
            reject  
          }  
        },  
        None -> reject,  
      }  
    }  
    None -> { reject }  
  }  
}
```

```

filter hidden_hop(attr: PathAttributes)
  match attr.otc() {
    Some(otc) -> {
      match attr.aspath() {
        Some(asp) -> {
          if not aspath.contains(otc)
            accept
          } else {
            reject
          }
        },
        None -> reject,
      }
    }
    None -> { reject }
  }
}

```

```

{
  "nlri": "186.251.12.0/22",
  "routes": [
    {
      "status": "active",
      "ingress": {
        "id": 39,
        "ingress_type": "bgpViaBmp",
        "parent_ingress": 3,
        "remote_addr": "185.1.167.88",
        "remote_asn": 51019,
        "rib_type": "AdjRibIn",
        "peer_rib_type": "inPost",
        "peer_type": "GlobalInstance"
      },
      "rpki": {
        "rov": "notFound"
      },
      "pathAttributes": [
        {
          "asPath": [
            "AS6447",
            "AS51019",
            "AS34927",
            "AS267613",
            "AS262645",
            "AS263561"
          ]
        },
        {
          "otc": "AS6777"
        }
      ]
    }
  ]
}

```

```

filter hidden_hop(attr: PathAttributes)
  match attr.otc() {
    Some(otc) -> {
      match attr.aspath() {
        Some(asp) -> {
          if not aspath.contains(otc)
            accept
          } else {
            reject
          }
        },
        None -> reject,
      }
    }
    None -> { reject }
  }
}

```

```

{
  "nlri": "186.251.12.0/22",
  "routes": [
    {
      "status": "active",
      "ingress": {
        "id": 39,
        "ingress_type": "bgpViaBmp",
        "parent_ingress": 3,
        "remote_addr": "185.1.167.88",
        "remote_asn": 51019,
        "rib_type": "AdjRibIn",
        "peer_rib_type": "inPost",
        "peer_type": "GlobalInstance"
      },
      "rpki": {
        "rov": "notFound"
      },
      "pathAttributes": [
        {
          "asPath": [
            "AS6447",
            "AS51019",
            "AS34927",
            "AS267613",
            "AS262645",
            "AS263561"
          ]
        },
        {
          "otc": "AS6777"
        }
      ]
    }
  ]
}

```

- Prometheus style metrics, enabling for monitoring and alerting
- Built-in Rotonda-related metrics
  - Soon to be overhauled
  - Some of the metrics might currently be off, don't trust these blindly
- and of course ...

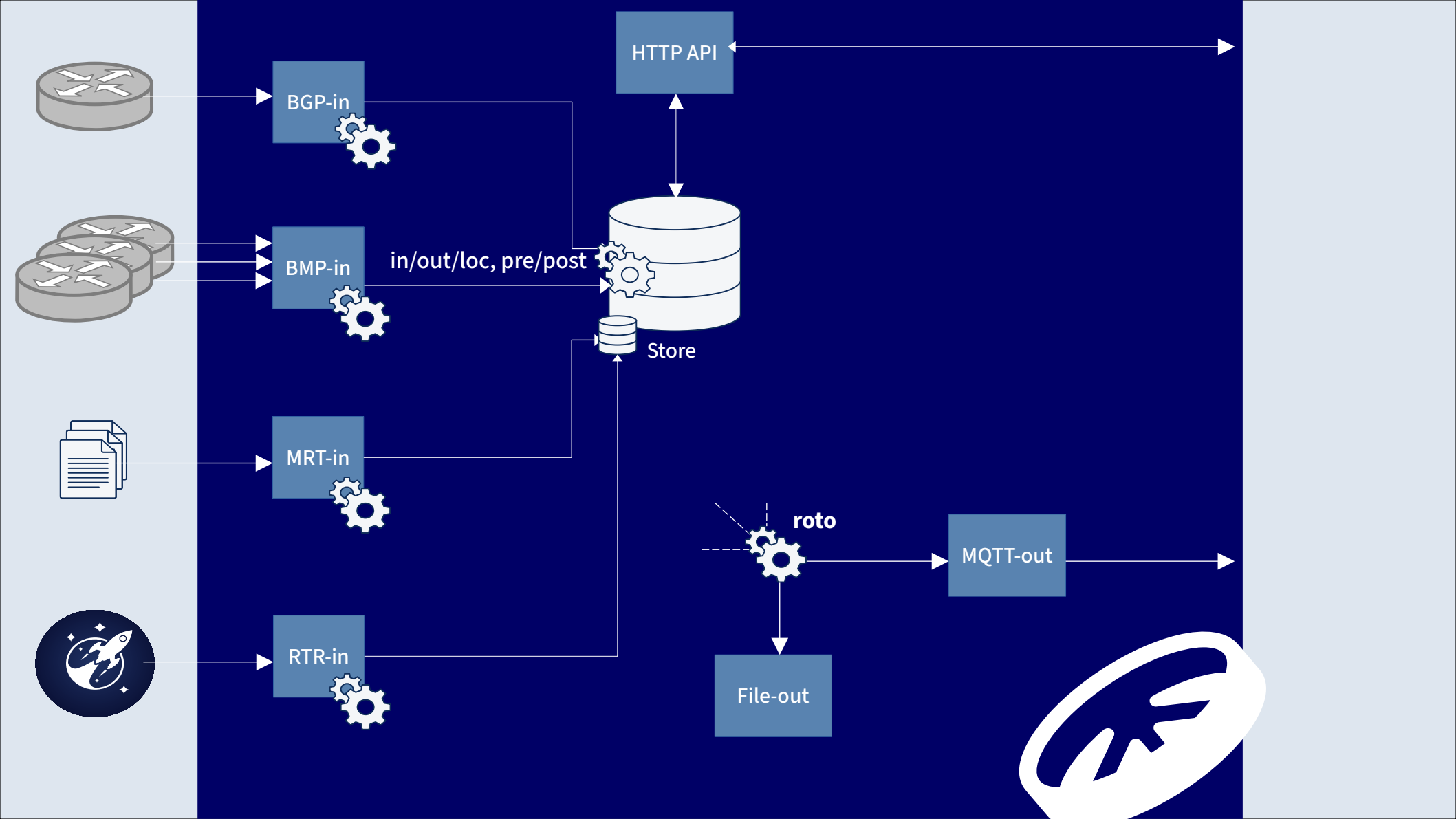
# User-defined metrics via Roto!

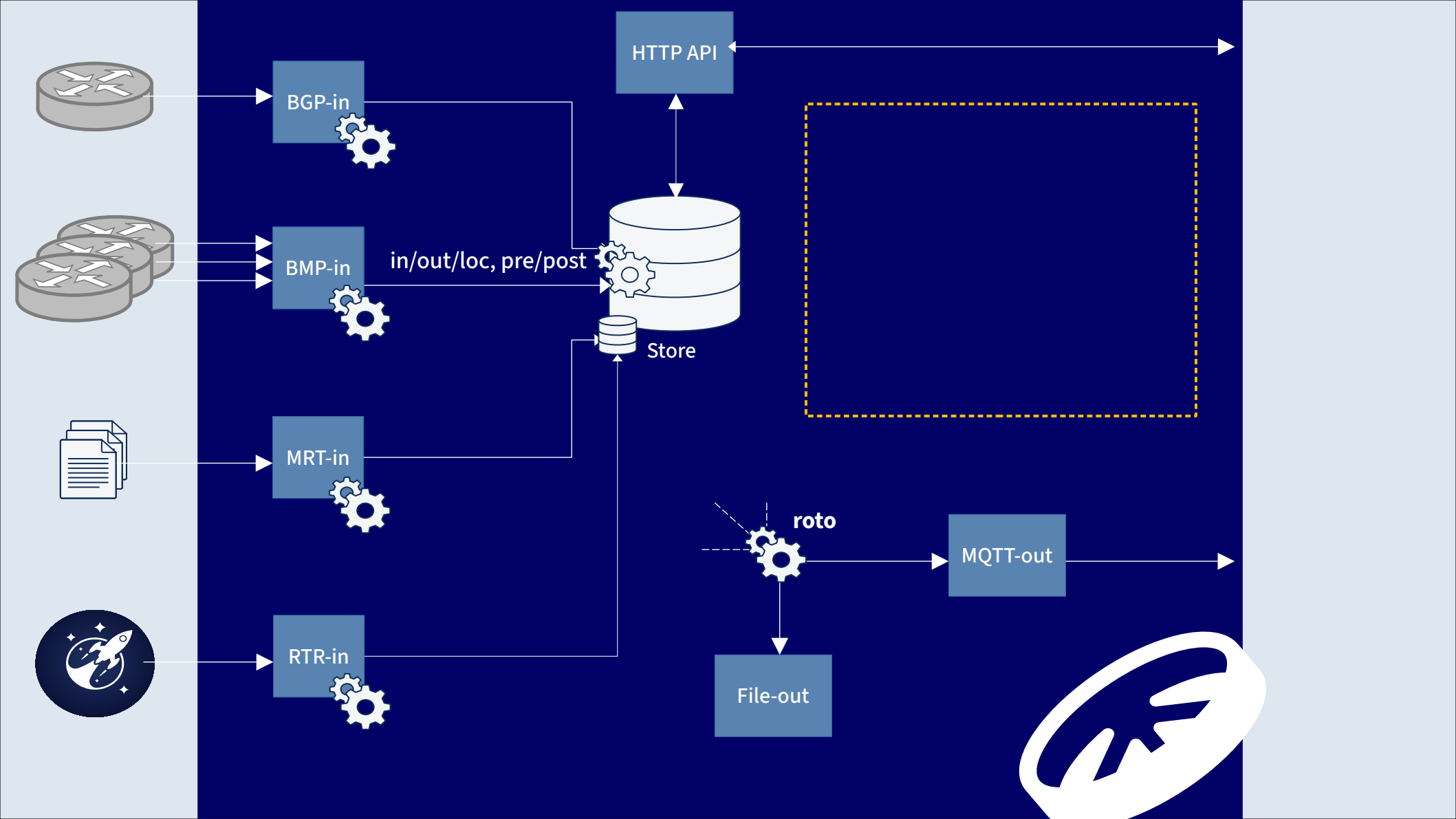
```
filter bmp_in(bmp_msg: BmpMsg, ingress_info: IngressInfo) {  
    if bmp_msg.withdrawals_count() >= 5 {  
        metrics.increase_counter(  
            f"packed_withdrawals{{peer_address='{ingress_info.peer_address()}',\  
            peer_asn='{ingress_info.peer_asn()}'}}",  
            1  
        );  
    }  
    accept  
}
```

# User-defined metrics via Roto!

```
filter bmp_in(bmp_msg: BmpMsg, ingress_info: IngressInfo) {  
    if bmp_msg.withdrawals_count() >= 5 {  
        metrics.increase_counter(  
            f"packed_withdrawals{{peer_address='{ingress_info.peer_address()}',\n              peer_asn='{ingress_info.peer_asn()}'}}",  
            1  
        );  
    }  
    accept  
}
```

```
$ curl -s "http://localhost:8081/metrics" | grep packed_withdrawals  
# TYPE packed_withdrawals counter  
roto_user_defined_packed_withdrawals{peer_address='185.49.140.10', peer_asn='AS211321'} 1
```







# Open-ish question

- **Streaming out at high volume**
  - Kafka, though what should the schema look like, exactly?
  - Any other alternatives? What would be a good fit for you?

## Concluding, Rotonda today

- Ingest and store high volumes of routing info in memory
- IPv6/IPv4 unicast (and multicast)
- Interaction via HTTP JSON API and minimal web UI


# Rotonda \$tomorrow

- **Non-unicast address families**
  - e.g. L3VPN, FlowSpec, (BGP-LS?)
  - ADDPATH
- **Expose ASPA to Roto**
- **Persistent storage to disk**
- **Interaction via CLI**
- **YANG-based configuration**

# Give it a go now

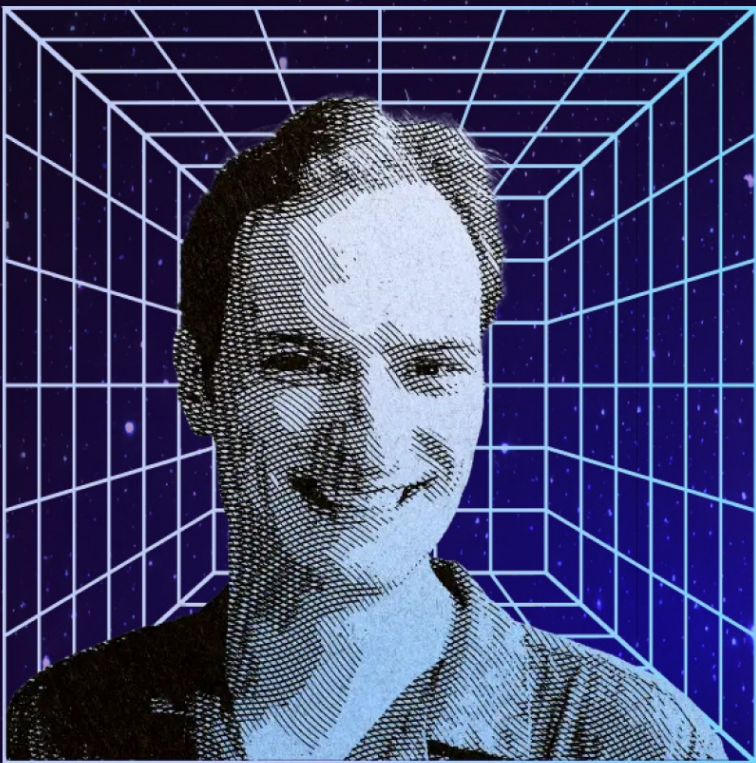
- Freely available open source software
- Take a .deb, .rpm or docker image, or compile using cargo
- <https://rotonda.docs.nlnetlabs.nl>
- <https://github.com/NLnetLabs/rotonda/>

# Give it a go

- Freely available open source software
- Take a .deb, .rpm or docker image or compile using cargo
- <https://rotonda.ch> Releases 9
- <https://github.com>  **0.5.0 'Mosaïque Public'** Latest  
Today

# and help us out!

- **We are not operators**
  - we don't run a network
  - we don't have any routing hardware
- **Publicly available non-unicast (test)data is hard to come by**
  - .pcaps welcome!
- **BMP export implementations all have their surprises**
  - Moreover, BMP as a protocol is still evolving. Any input/data is useful for us.



Talk

# ROTO: a FAST and SAFE SCRIPTING Language

Terts Diepraam  
software engineer  
at NInet Labs

EURO  
RUST

by Mainmatter

October 9-10, 2025  
Paris and online  
[eurorust.eu](http://eurorust.eu)



Collecting and analyzing  
routing information with

# ROTONDA

NLNOG Day 2025

[luuk@nlnetlabs.nl](mailto:luuk@nlnetlabs.nl)

