Churn

GÉANT's Next Iteration in Network

Monitoring and Reporting

Erik Reid

TNC26, Helsinki

10 June 2026

tnc26

Co-funded by the European Union

whoami

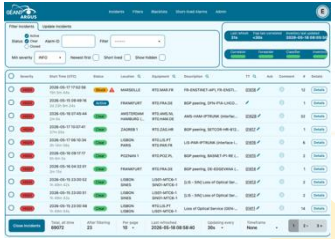
```
% id
 uid = erik.reid
 gid = GÉANT

% jq metadata.json
{
  "years": 9,
  "job": "Head, Software Engineering Team",
  "GN5-2": "Task Lead, WP9 T4",
  "email": "erik.reid@geant.org"
}
```

What is Churn and why have we made it?

Churn is GÉANT's latest iteration in improving how we monitor and report on network utilization and performance

… it's a deliberate and structured response to some pressures and problems GÉANT sees approaching with our observability and reporting stack

Growing Duplicate and Confusing Network Configuration



credentials

prefix-list

NETCONF

firewall

gRPC

ssh

prefix-list

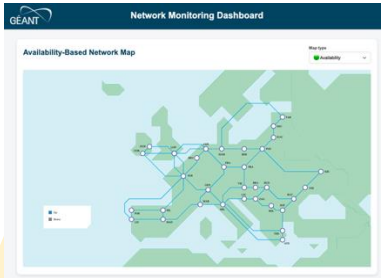
credentials

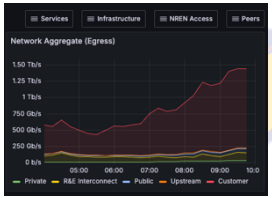
firewall

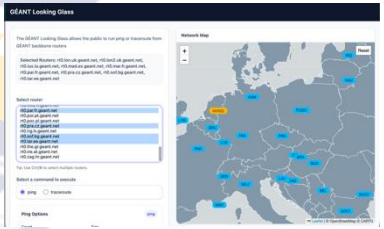
prefix-list

SNMP

credentials



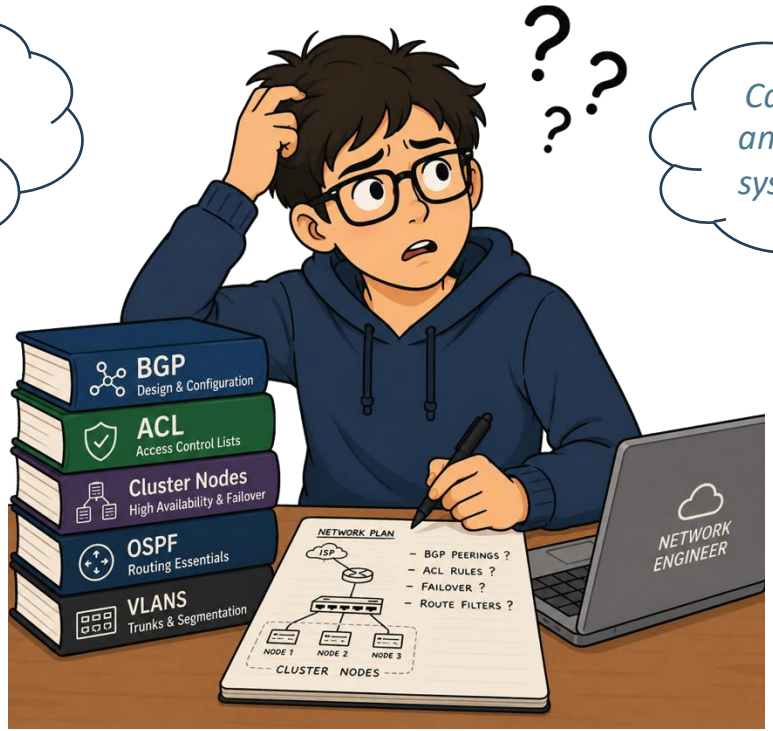




No “Big Picture” - Confusing Network Management

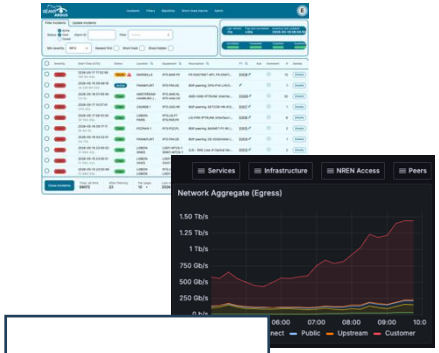
Can you give me the list of addresses that need access to …?

Can you tell me if any of these X/Y/Z systems still exist?



Why aren’t we seeing any monitoring data for …?

Redundancy and Overlap: Logical and Load



TBD...

TBD...

NETCONF

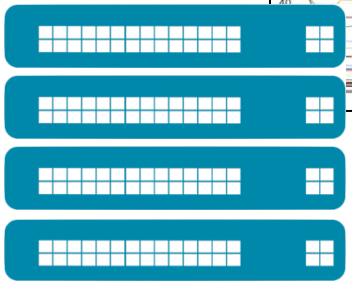
SNMP

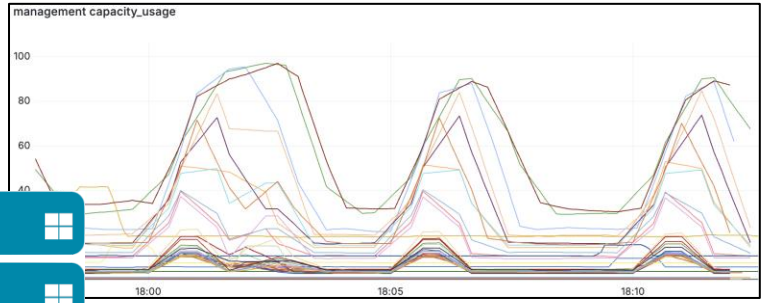
ST

NETCONF

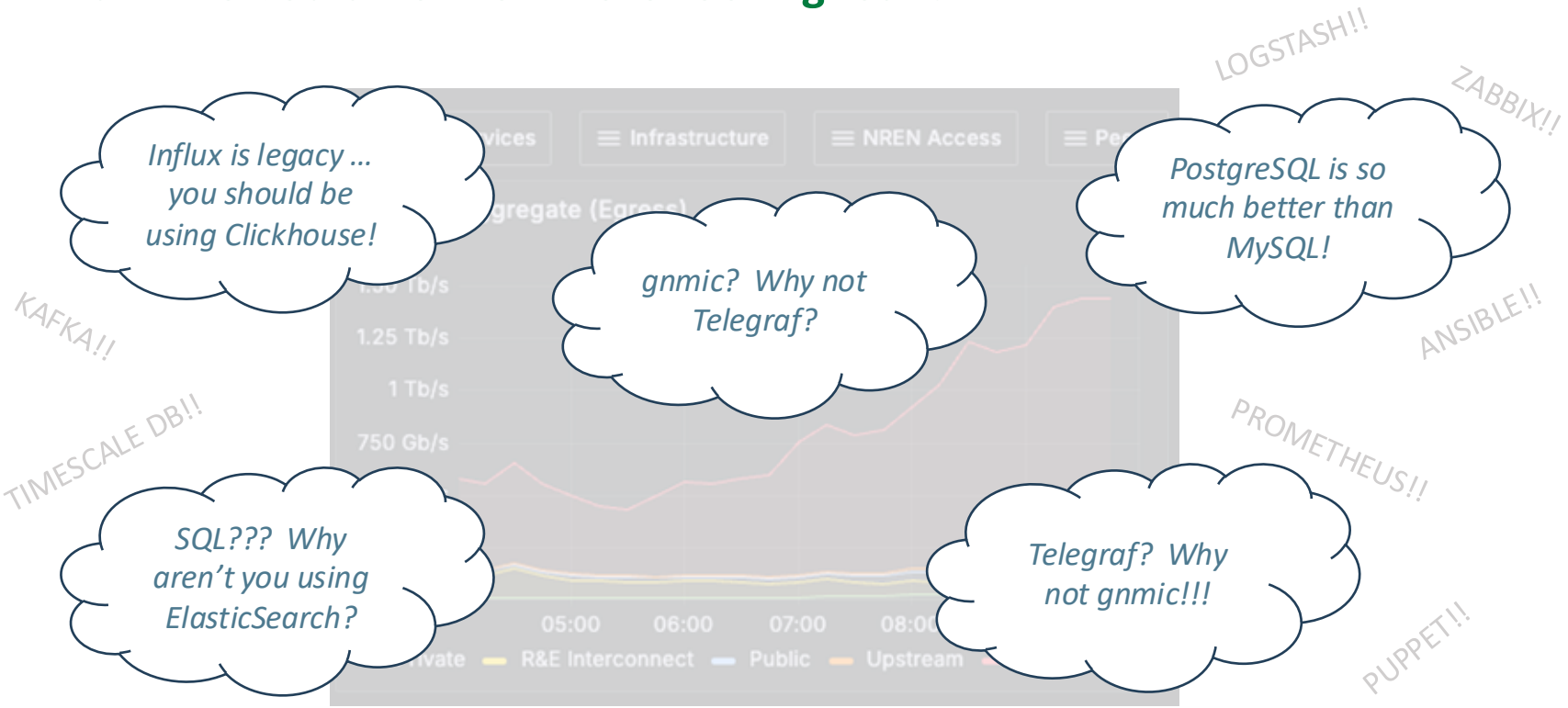
SNMP

ST





And … We Could Be Even More Tool-Agnostic



Summary: Problems To Requirements

Unmanageable Network Configuration

Duplicated/Overlapping Effort

Too Tight Tech Coupling

Maintenance Scalability Wall

System-Level Requirements

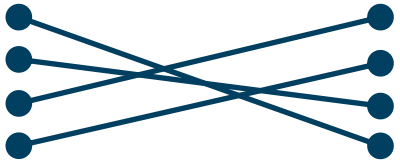
- Small ingest footprint
- Sharable processing pipeline
- Tech Stack dependencies at processing level, not architecture level
- Complexity doesn't scale linearly with new projects

Design Approach: Issues With Some Extremes We Keep Repeating

Framework Approach

- generality assumed too early
- breaks with the next use case
- “metadata” trap
- changes are difficult
- performance?

Service-Specific Pipelines



- New pipelines add too much effort
- Limited reuse
- Shared processing not leveraged
- Stack drift & dependencies

Over-Generalization ⟷ Fragmentation

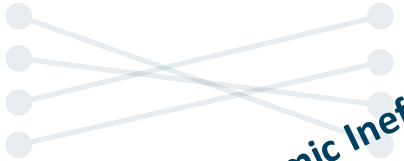
Design Approach: Issues With Some Extremes We Keep Repeating

Framework Approach

False Generality -> Hidden Complexity

- generality assumed too early
- breaks with the next use case
- "metadata" trap
- changes are difficult
- performance?

Service-Specific Pipelines



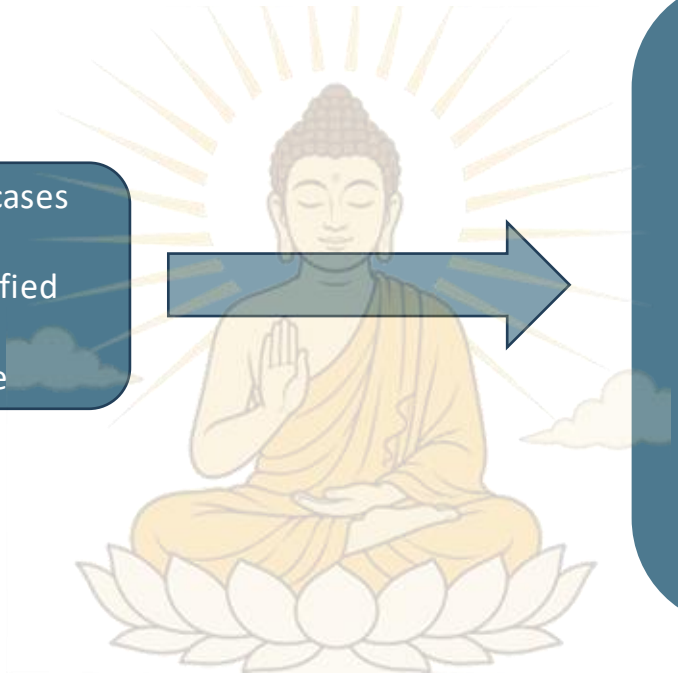
Local Optimization -> Systemic Inefficiency

- New pipelines add too much effort
- Limited reuse
- Shared processing not leveraged
- Stack drift and dependencies

Over-Generalization ⟷ Fragmentation

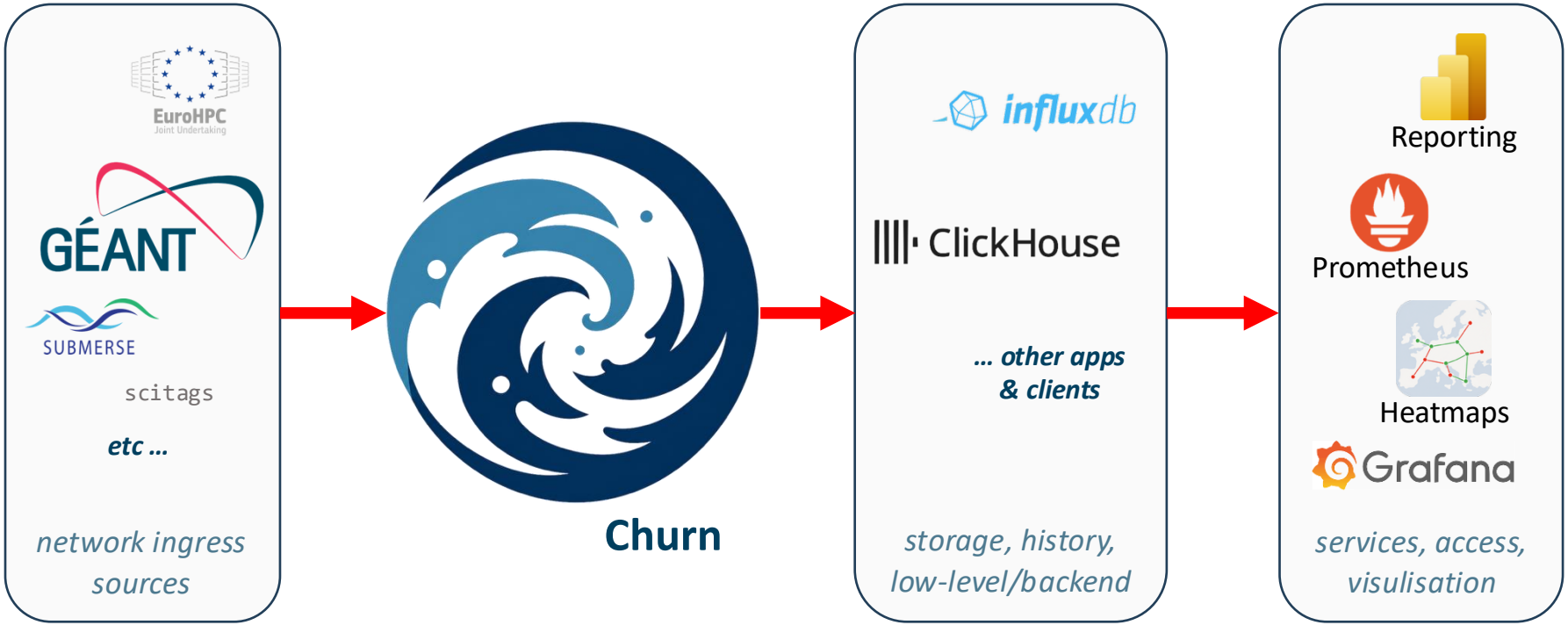
Pragmatic Approach: Middle Path

- Start from concrete use cases
- Abstract only where justified
- Agile, continuously refine



- Iteration, not a Framework
- Share what actually converges
- Follow guidelines, but don’t enforce uniformity
- Internally Explicit (no “metadata”)
- Accept some development effort for new integrations and pipelines, but keep it minimal

Where Does Churn Fit in the Big Picture?

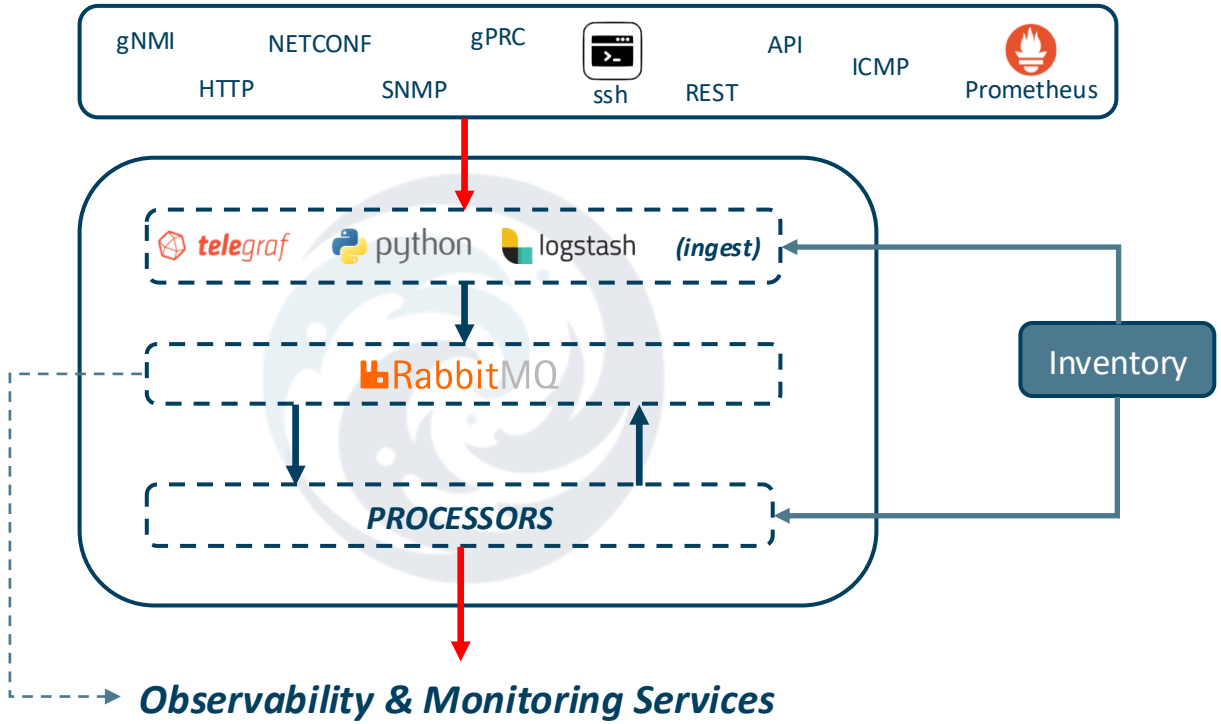


What Churn Is and Is Not

- Is
  - A common way to consolidate required and runtime access to network devices
  - A way to maximize visibility of monitoring communication with network devices, and to keep this as simple and secure as possible

- Is not
  - Another monitoring tool (Zabbix, Telegraf, Prometheus, ELK, Kentik)
  - Another data store (InfluxDB, ClickhouseDB, ElasticSearch, …)
  - A final and complete solution to all possible network monitoring needs

A Look Inside …



Development & Maintenance

Strict Internal consistency:

- Exchange and Queue names are part of the System Design
- Internal Process Communication: heavy use of Pydantic and validation
- Internal Churn models don't leave Churn

Processors can be very simple and follow a repeatable pattern:

Development & Maintenance: Processor Code Example

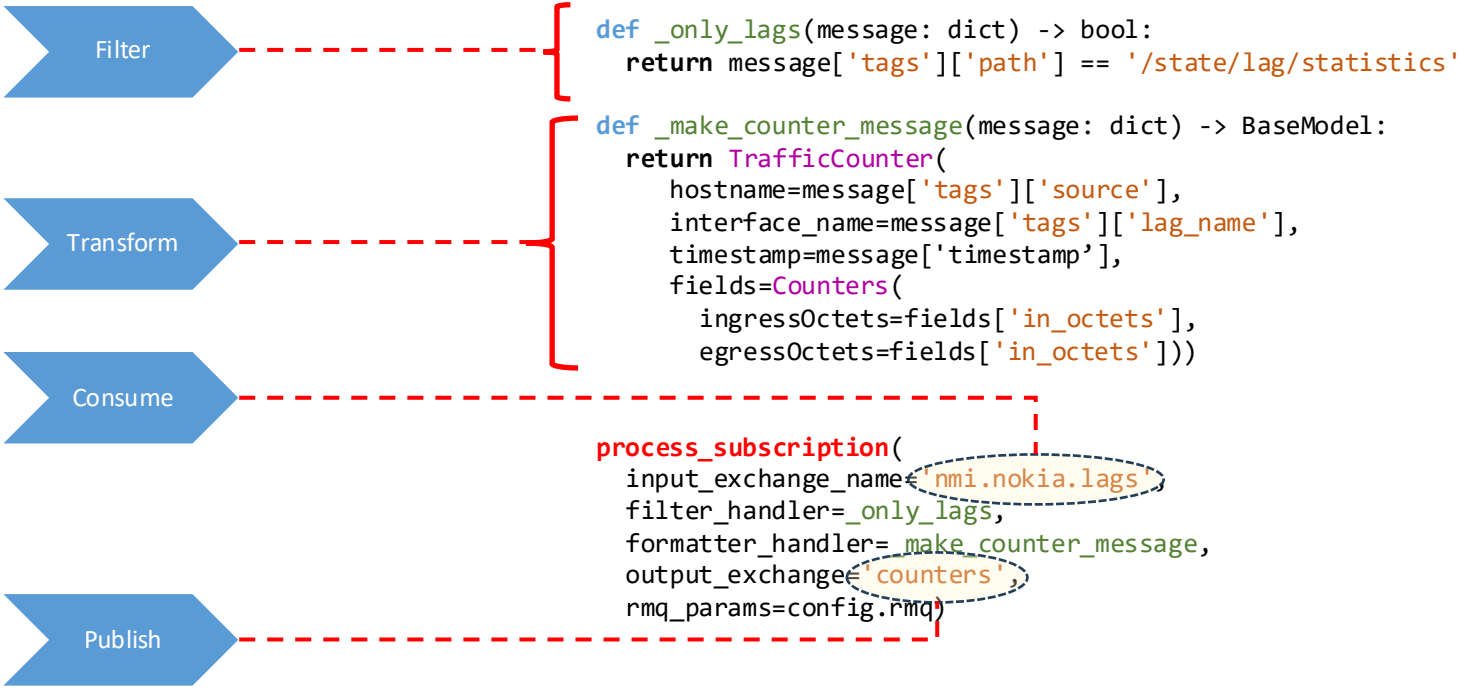
Most processors can be small, follow repeatable patterns, and share simple helper utilities

Input: gnmi/OpenConfig

Output: Churn Pydantic

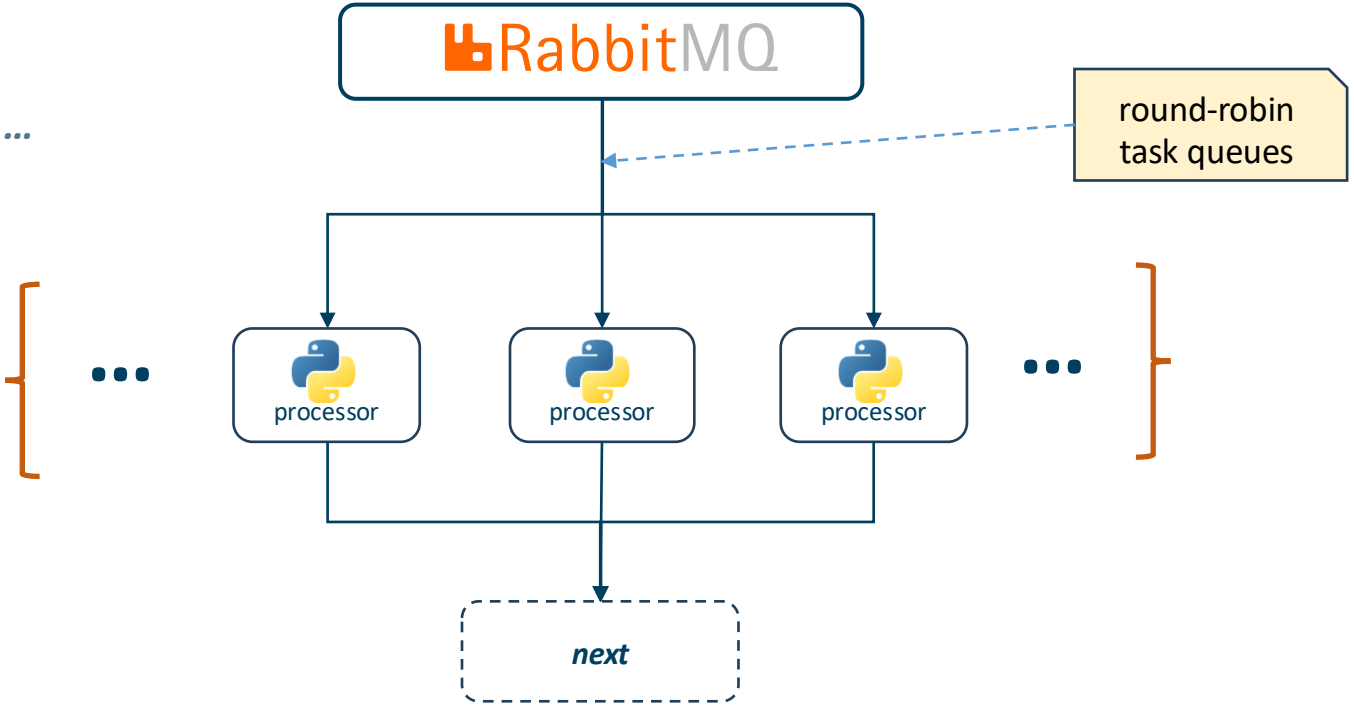
```
def _only_lags(message: dict) -> bool:
  return message['tags']['path'] == '/state/lag/statistics'

def _make_counter_message(message: dict) -> BaseModel:
  return TrafficCounter(
     hostname=message['tags']['source'],
     interface_name=message['tags']['lag_name'],
     timestamp=message['timestamp'],
     fields=Counters(
       ingressOctets=fields['in_octets'],
       egressOctets=fields['in_octets']))


process_subscription(
  input_exchange_name='nmi.nokia.lags',
  filter_handler=_only_lags,
  formatter_handler=_make_counter_message,
  output_exchange='counters',
  rmq_params=config.rmq)
```

Development & Maintenance : Processor Code Example



Processing Scales Nicely

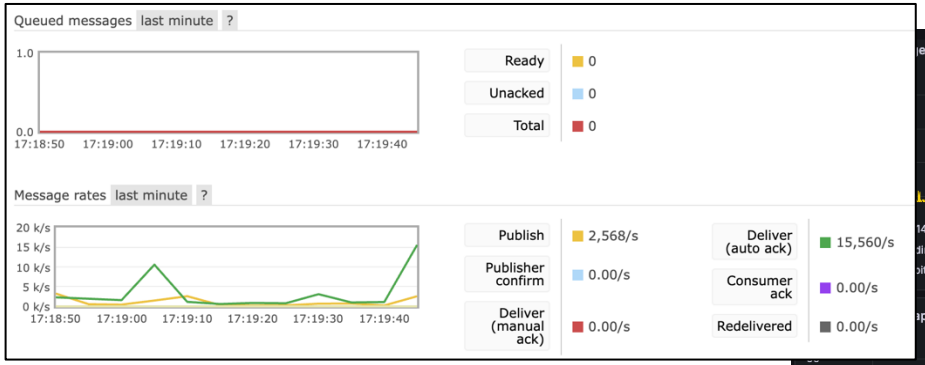
And in case you thought we forgot …



Stories

A few interesting things that have happened along the way …

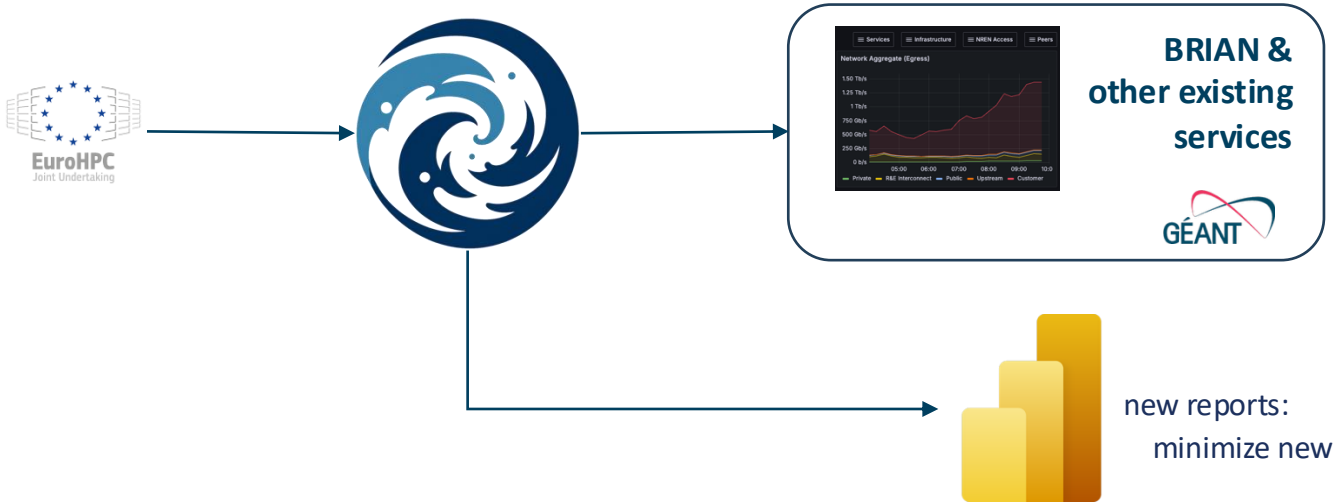
System Observability



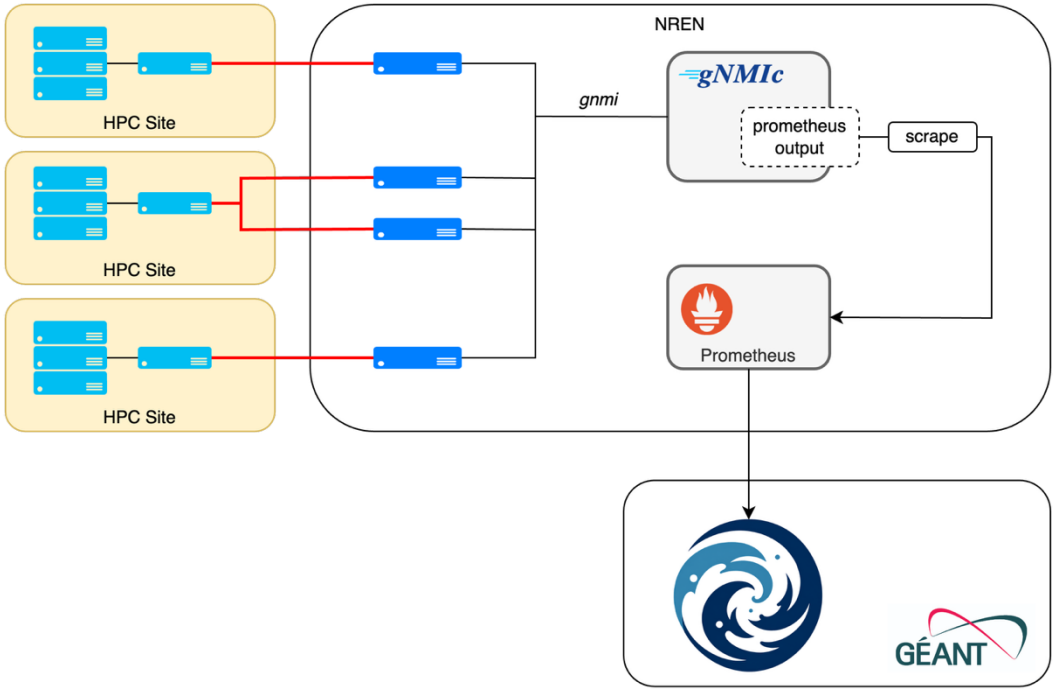
More observability, for free
(even more than we’e using for now …)

EuroHPC Is A Big Deal

Link Availability and Utilization can leverage most of the same processing pipelines as for the existing network



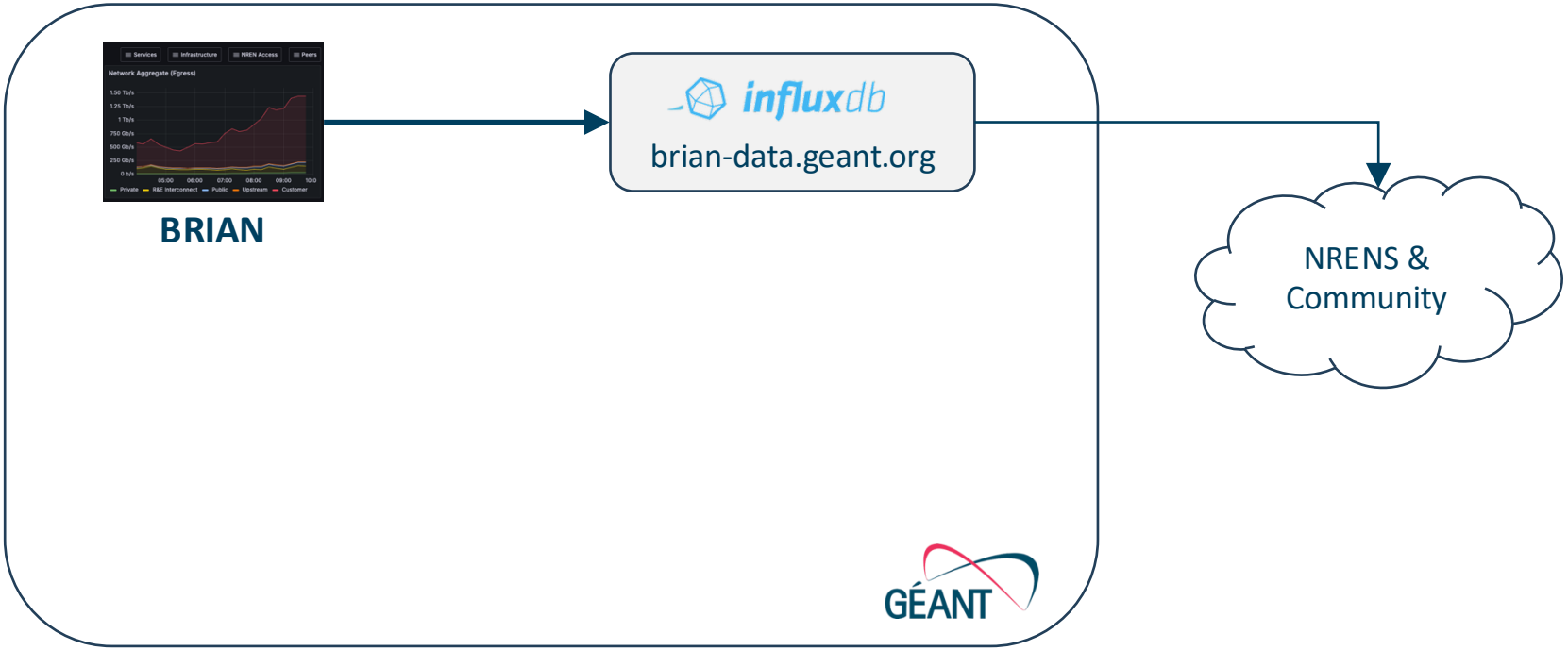
EuroHPC: Quick Coalescence/Nice Sharing Model



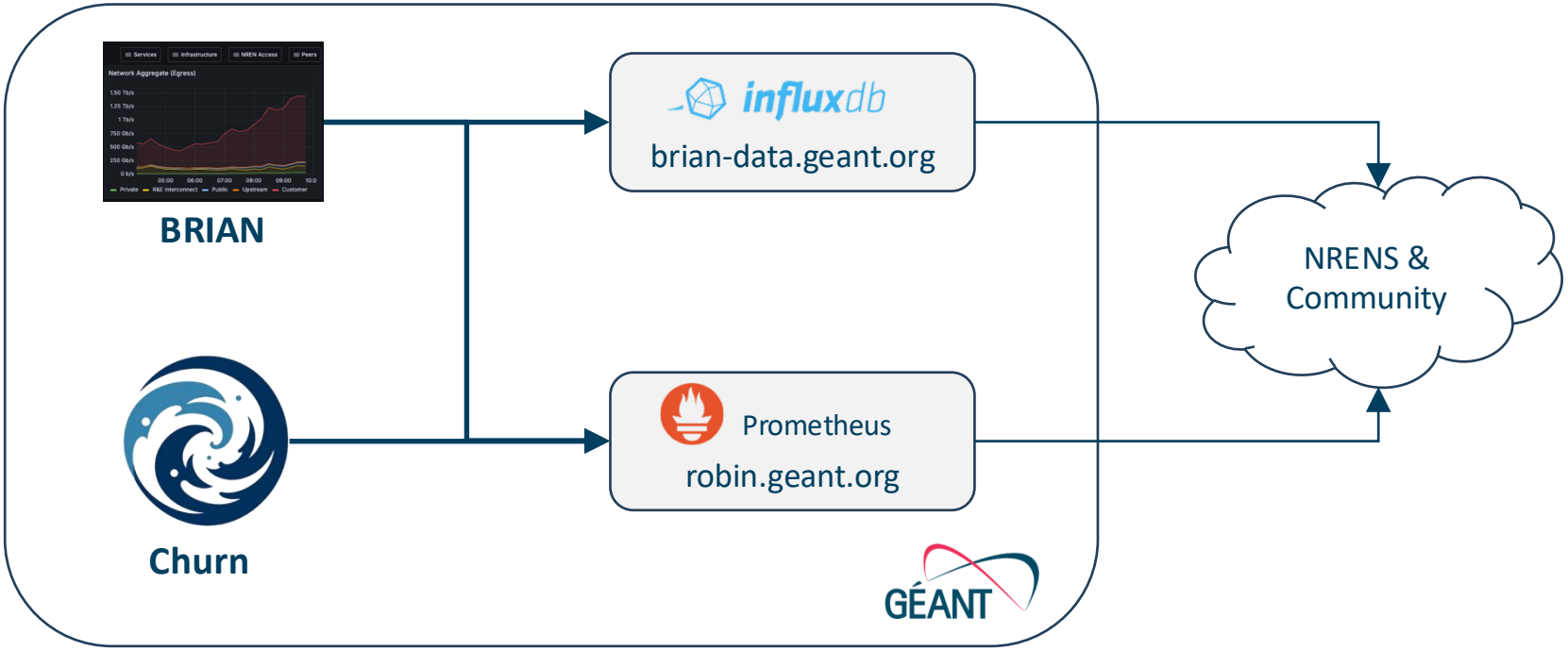
It's helpful to think of using Prometheus as a lightweight data sharing tool, rather than a monitoring platform

- Easy to Provision & Run
- Easy to Understand
- Broad built-in compatibility
- Minimizes access to NREN networks

Stories: Improved ROBIN (Data Sharing)



Stories: Improved ROBIN (Data Sharing)



More Info

DOCUMENTATION:
https://swd-documentation.geant.org/churn/develop/

SOURCE CODE:
https://gitlab.software.geant.org/geant-swd/churn/churn

CONTACT: erik.reid@geant.org

Thank you

Any questions?

erik.reid@geant.org

Co-funded by the European Union