Software contribution from MIPT: Development of Event Metadata System and Monitoring & High-Availability Service

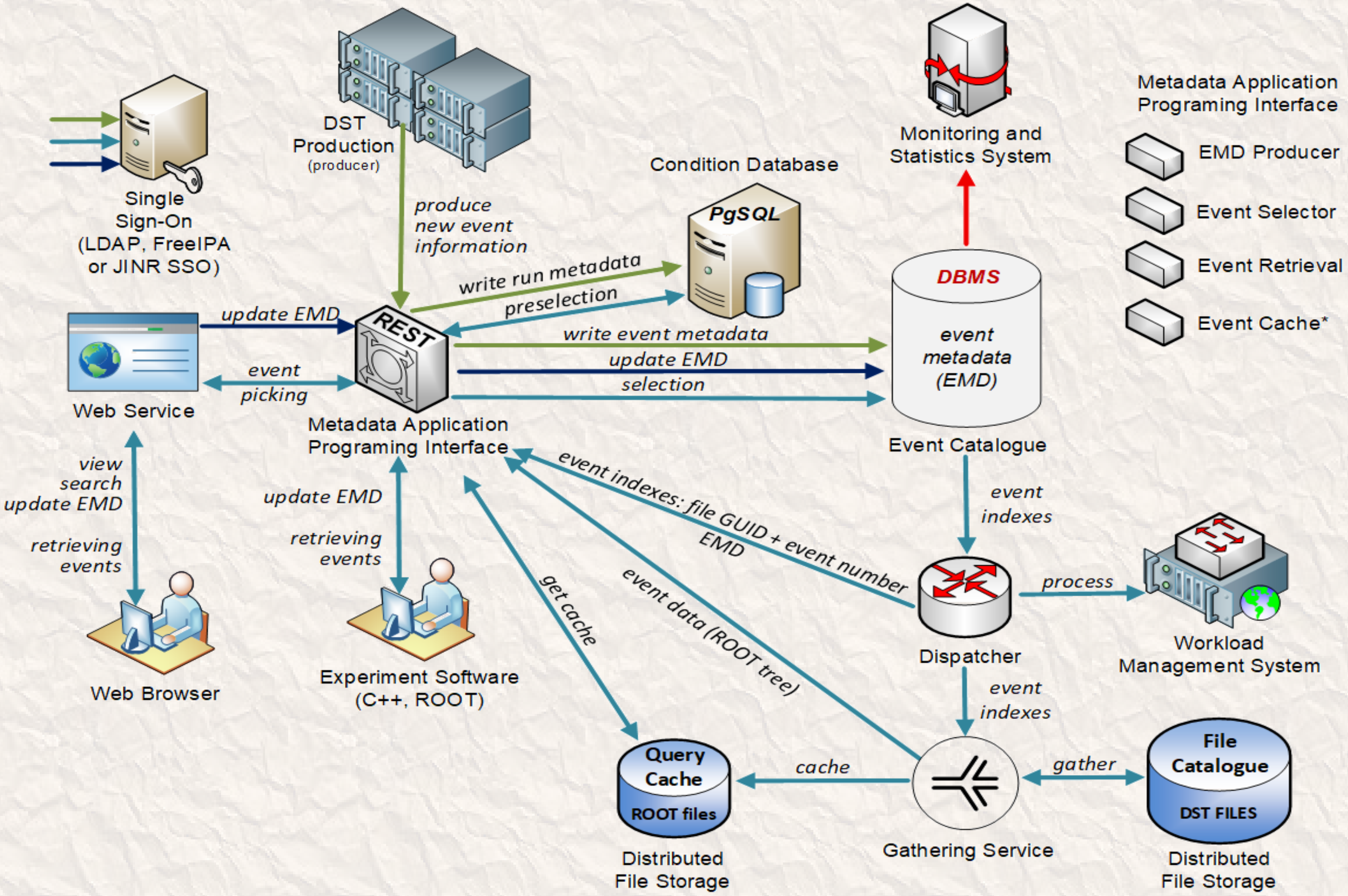
Peter Klimai

Current Projects Summary

Project	URL	Notes
Event Metadata System	https://git.jinr.ru/nica_db/emd	First version deployed; updates discussed in this talk
Deployment scripts for EMS		This talk
High Availability design for EMS		This talk
Statistics collection and visualization for EMS		WIP
Next-generation event display	https://github.com/SciProgCentre/visionforge	See talk by A. Nozik
Monitoring service	https://mon-service.jinr.ru https://git.jinr.ru/nica/bmnroot/-/tree/dev/services/is_monitor	In production; updates planned. This talk
Slow control system viewer	https://bmn-tango.jinr.ru	Needs update to match new SCS and its database

Event Metadata System – Update

EMS Architecture and Features



- Event Metadata System
  - Event Catalogue is based on PostgreSQL
  - Integrates with BM@N Condition database
  - REST API and Web UI developed based on Kotlin multiplatform
  - Configurable to support different metadata
  - ROOT macro to write BM@N events in the catalogue
  - Role-based access control implemented
  - Monitoring

For more details:

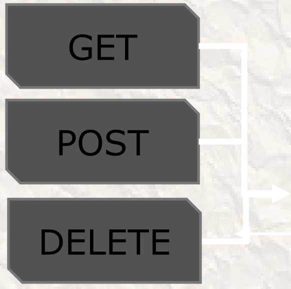
E. Alexandrov, I. Alexandrov, A. Chebotov, A. Degtyarev, I. Filozova, K. Gertsenberger, P. Klimai and A. Yakovlev, “Implementation of the Event Metadata System for physics analysis in the NICA experiments”, J. Phys.: Conf. Ser. 2438, 012046 (2023).

EMS Updates

- Recent EMS Updates (discussed next):
  - New unified REST API scheme
  - Simplified to support only one metadata table per EMS instance
  - OpenAPI documentation (aka Swagger) now available
  - Database performance improvement studies (indexes)
  - High Availability solution
  - Deployment scripts (Ansible based)

New scheme for REST API

- The new scheme is unified for different BM@N Information Systems



GET

POST

DELETE

https://bmn-event.jinr.ru/event_api/v1/event?

run_number=3950:4000&beam_particle=Ar&target_particle=Al

energy=3.16:3.18&target_particle=SRC%20Lead

HOSTNAME / SERVICE / VERSION / ENTITY?parameter_set

HOSTNAME=https://bmn-[SYSNAME].jinr.ru

SERVICE=[SYSNAME]_api

VERSION=v1 (v2…)

ENTITY=tablename without last '_' (if present)

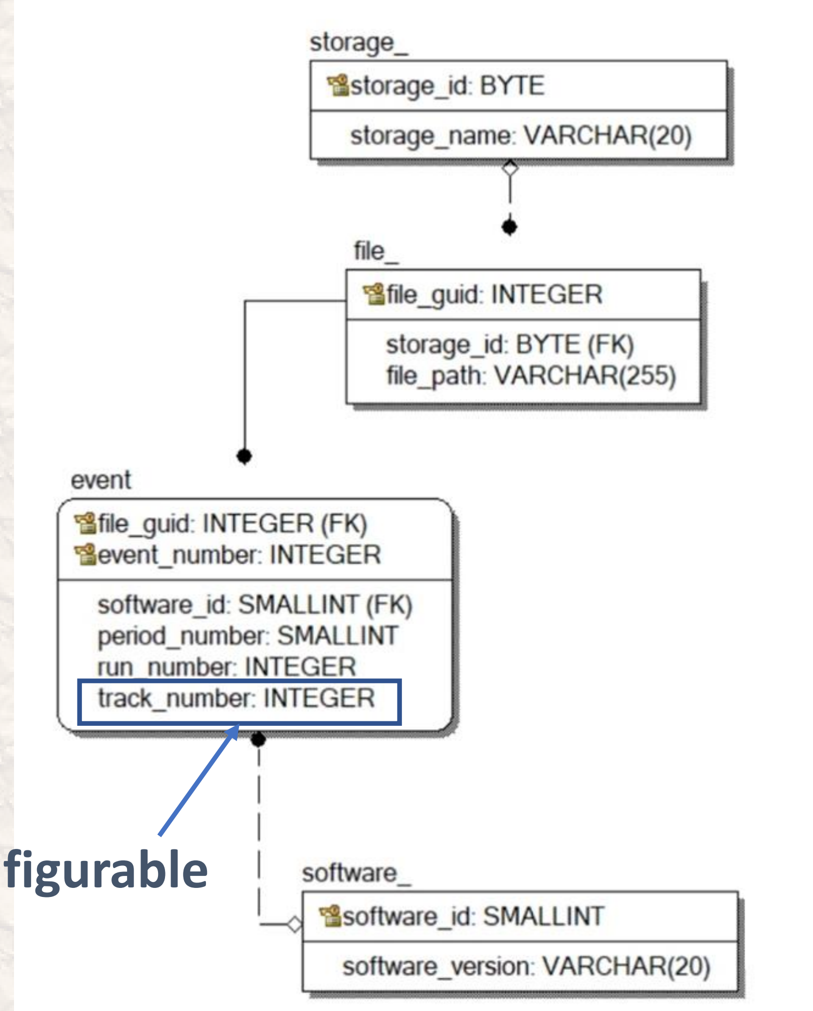
parameters are separated by '&'
ranges: min:max → >=min AND <=max
min: → >=min :max → <=max

For the Unified Condition Database (UniConDa), SYSNAME = uniconda

For the Event Metadata System (EMS), SYSNAME = event

Current BM@N Database Schema

EMS Database:



storage_
- storage_id: BYTE
- storage_name: VARCHAR(20)

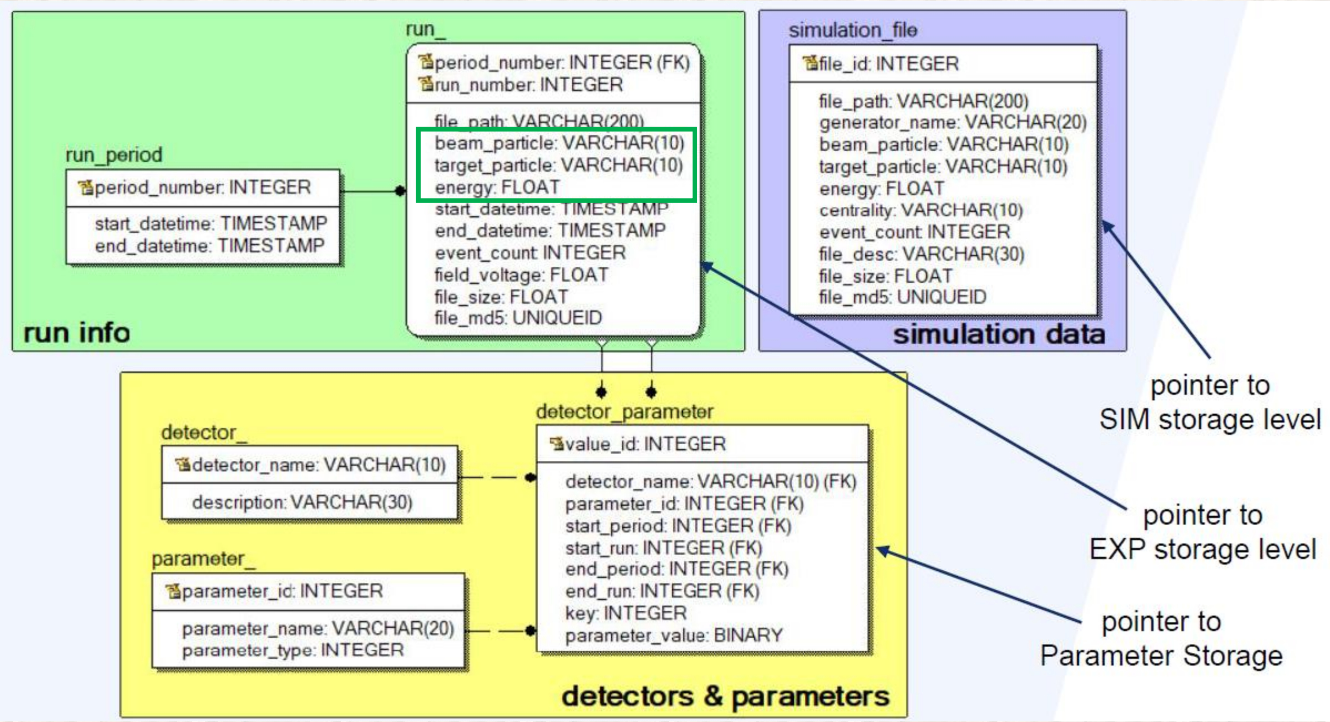
file_
- file_guid: INTEGER
- storage_id: BYTE (FK)
- file_path: VARCHAR(255)

event
- file_guid: INTEGER (FK)
- event_number: INTEGER
- software_id: SMALLINT (FK)
- period_number: SMALLINT
- run_number: INTEGER
- track_number: INTEGER

software_
- software_id: SMALLINT
- software_version: VARCHAR(20)

Configurable

Condition Database:



run_period
- period_number: INTEGER
- start_datetime: TIMESTAMP
- end_datetime: TIMESTAMP

run_
- period_number: INTEGER (FK)
- run_number: INTEGER
- file_path: VARCHAR(200)
- beam_particle: VARCHAR(10)
- target_particle: VARCHAR(10)
- energy: FLOAT
- start_datetime: TIMESTAMP
- end_datetime: TIMESTAMP
- event_count: INTEGER
- field_voltage: FLOAT
- file_size: FLOAT
- file_md5: UNIQUEID

run info

simulation_file
- file_id: INTEGER
- file_path: VARCHAR(200)
- generator_name: VARCHAR(20)
- beam_particle: VARCHAR(10)
- target_particle: VARCHAR(10)
- energy: FLOAT
- centrality: VARCHAR(10)
- event_count: INTEGER
- file_desc: VARCHAR(30)
- file_size: FLOAT
- file_md5: UNIQUEID

simulation data

detector_
- detector_name: VARCHAR(10)
- description: VARCHAR(30)

parameter_
- parameter_id: INTEGER
- parameter_name: VARCHAR(20)
- parameter_type: INTEGER

detector_parameter
- value_id: INTEGER
- detector_name: VARCHAR(10) (FK)
- parameter_id: INTEGER (FK)
- start_period: INTEGER (FK)
- start_run: INTEGER (FK)
- end_period: INTEGER (FK)
- end_run: INTEGER (FK)
- key: INTEGER
- parameter_value: BINARY

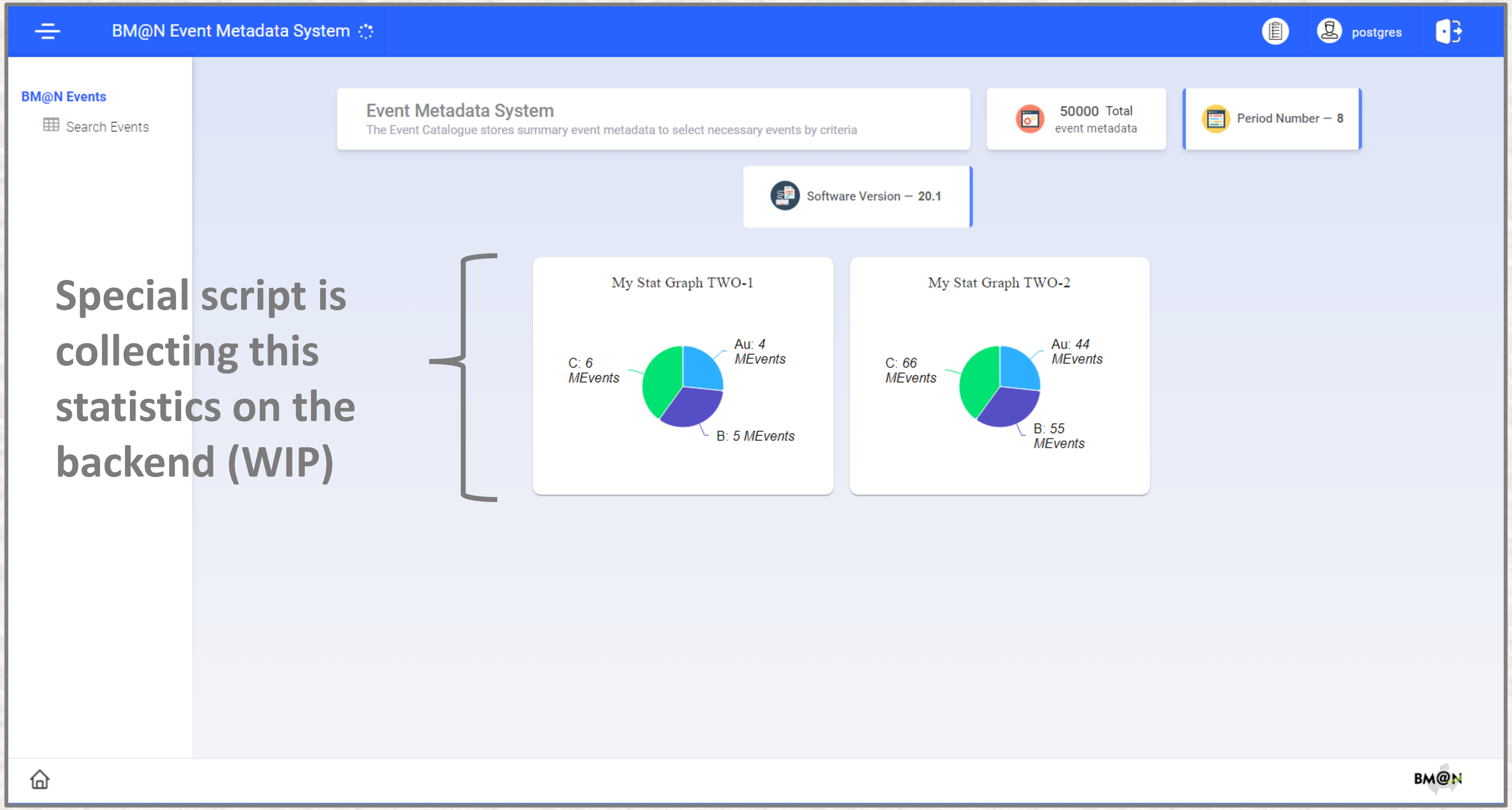
detectors & parameters

pointer to SIM storage level

pointer to EXP storage level

pointer to Parameter Storage

Web UI Main Page



BM@N Event Metadata System

postgres

BM@N Events

Search Events

Event Metadata System

The Event Catalogue stores summary event metadata to select necessary events by criteria

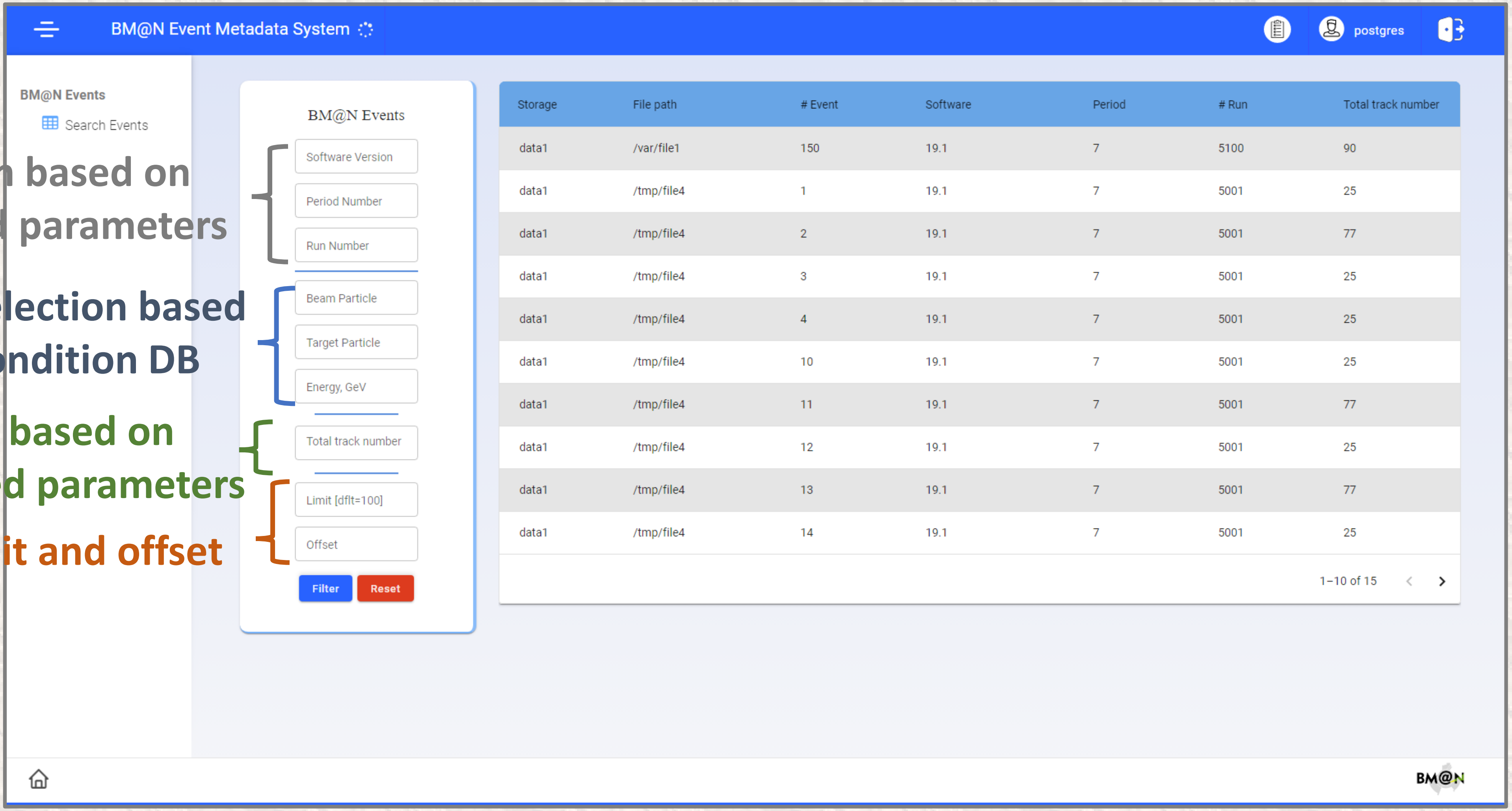
50000 Total event metadata

Period Number – 8

Software Version – 20.1

Special script is collecting this statistics on the backend (WIP)

Main search page



Storage	File path	# Event	Software	Period	# Run	Total track number
data1	/var/file1	150	19.1	7	5100	90
data1	/tmp/file4	1	19.1	7	5001	25
data1	/tmp/file4	2	19.1	7	5001	77
data1	/tmp/file4	3	19.1	7	5001	25
data1	/tmp/file4	4	19.1	7	5001	25
data1	/tmp/file4	10	19.1	7	5001	25
data1	/tmp/file4	11	19.1	7	5001	77
data1	/tmp/file4	12	19.1	7	5001	25
data1	/tmp/file4	13	19.1	7	5001	77
data1	/tmp/file4	14	19.1	7	5001	25

1–10 of 15

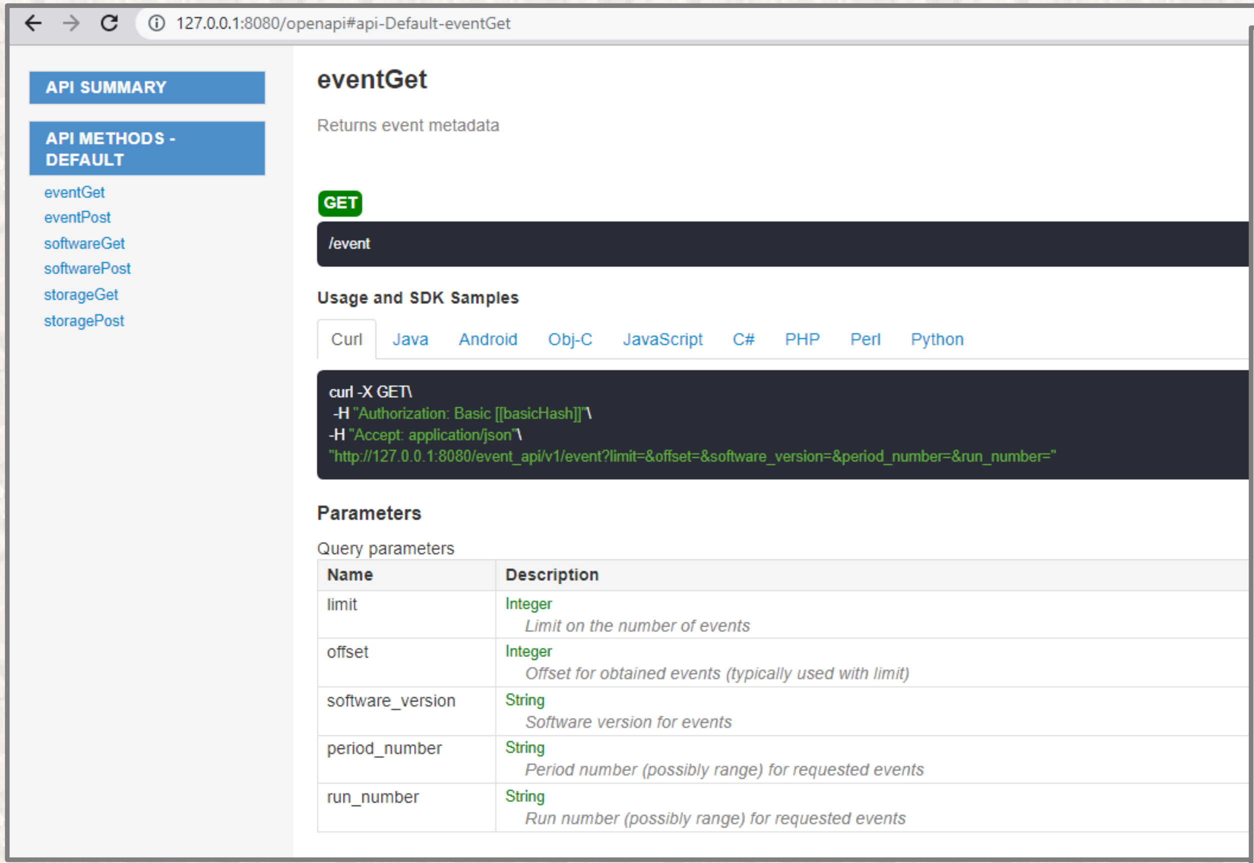
Selection based on standard parameters

Preselection based on Condition DB

Selection based on configured parameters

Limit and offset

OpenAPI pages for EMS



eventGet

Returns event metadata

GET

/event

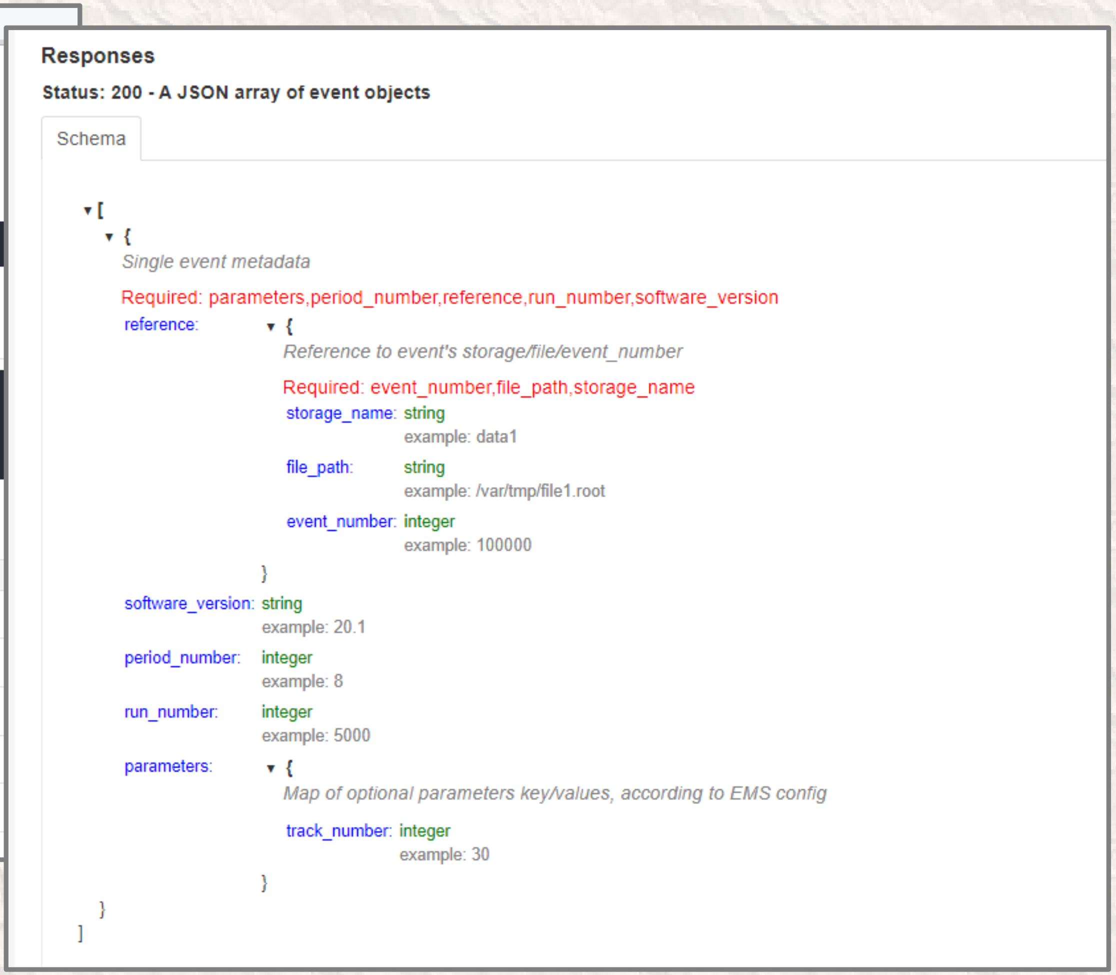
Usage and SDK Samples

```
curl -X GET\
 -H "Authorization: Basic [[basicHash]]"\
-H "Accept: application/json"\
"http://127.0.0.1:8080/event_api/v1/event?limit=&offset=&software_version=&period_number=&run_number="
```

Parameters

Query parameters

Name	Description
limit	Integer *Limit on the number of events*
offset	Integer *Offset for obtained events (typically used with limit)*
software_version	String *Software version for events*
period_number	String *Period number (possibly range) for requested events*
run_number	String *Run number (possibly range) for requested events*



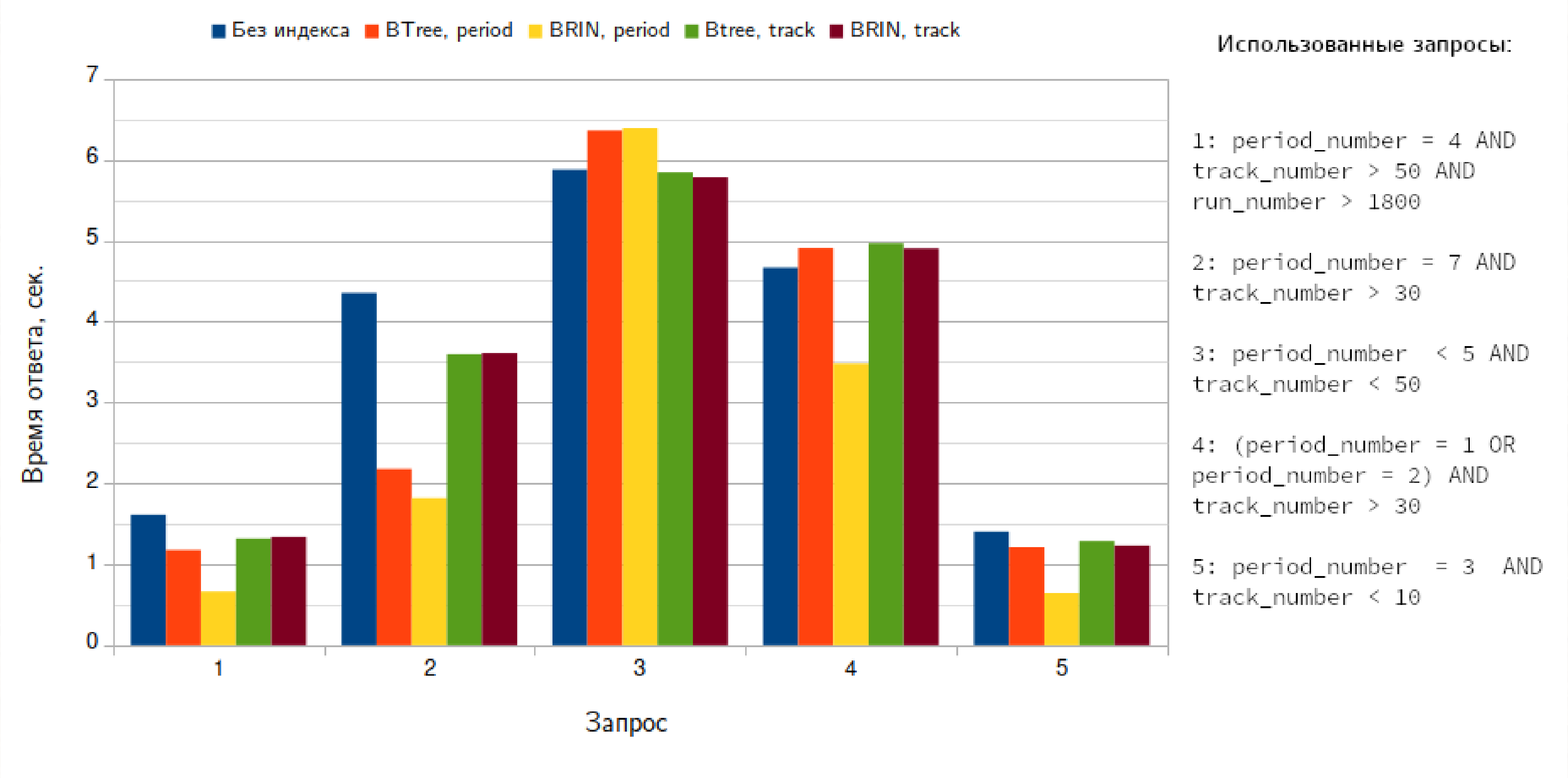
Responses

Status: 200 - A JSON array of event objects

```
▾[
  ▾{
    Single event metadata
    Required: parameters,period_number,reference,run_number,software_version
    reference:         ▾{
                        Reference to event's storage/file/event_number
                        Required: event_number,file_path,storage_name
                        storage_name: string
                                      example: data1
                        file_path:    string
                                      example: /var/tmp/file1.root
                        event_number: integer
                                      example: 100000
                       }
    software_version: string
                      example: 20.1
    period_number:    integer
                      example: 8
    run_number:       integer
                      example: 5000
    parameters:        ▾{
                        Map of optional parameters key/values, according to EMS config
                        track_number: integer
                                      example: 30
                       }
  }
]
```

Index Selection (Type and Columns)

- Measurements with test database instance are shown (50M events)

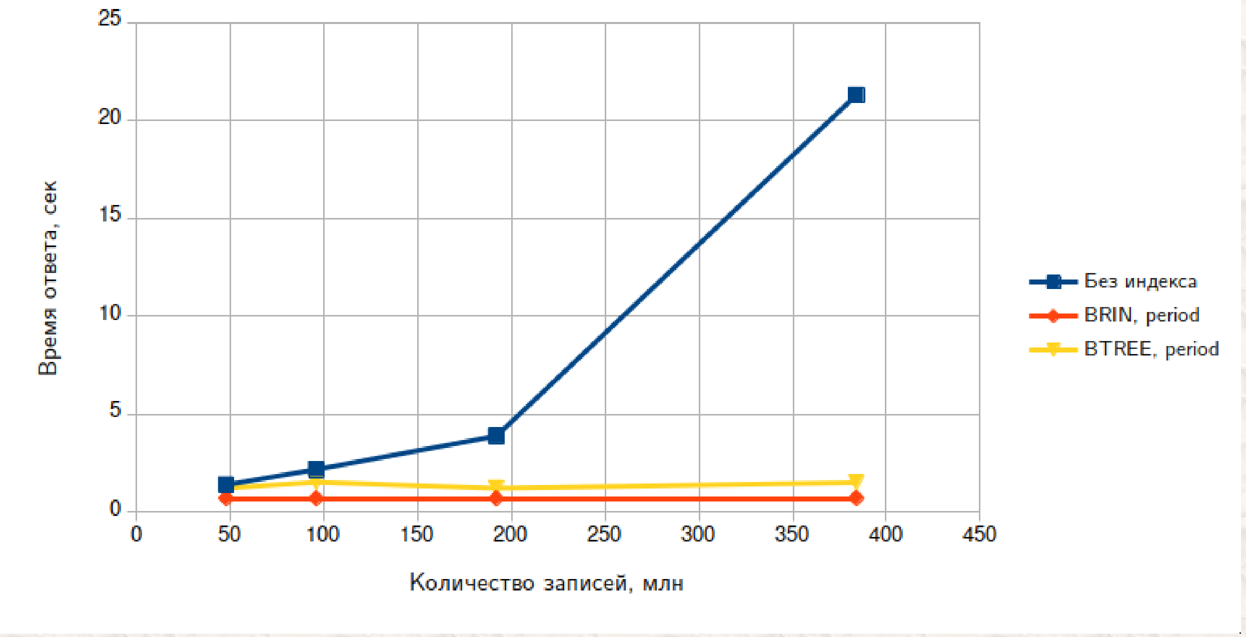


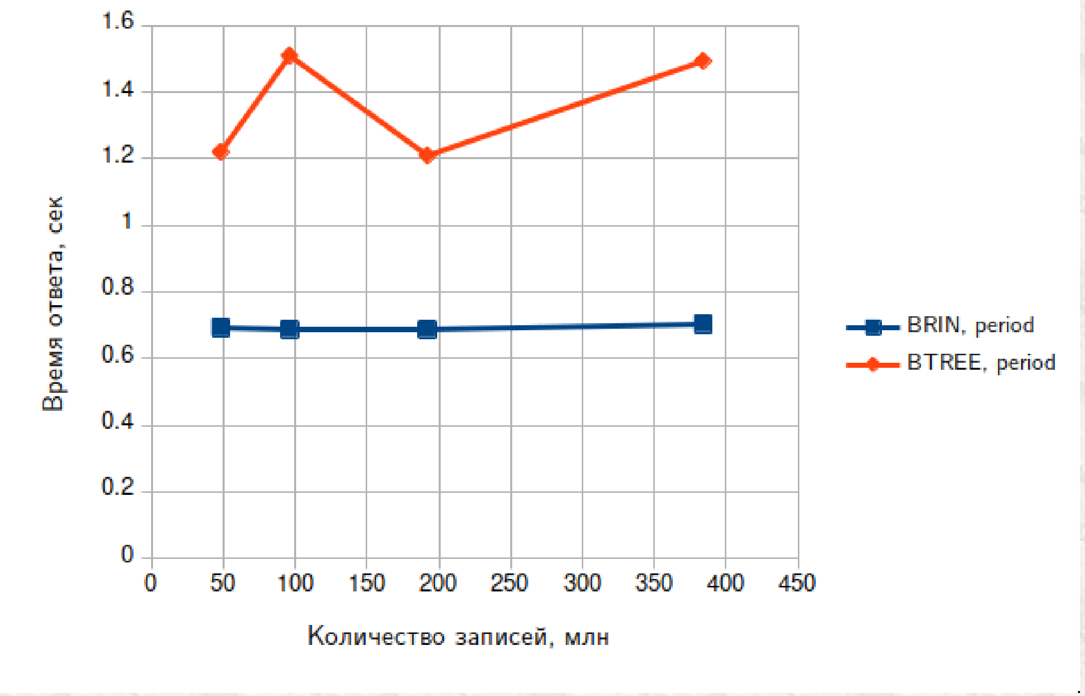
Использованные запросы:

```
1: period_number = 4 AND
track_number > 50 AND
run_number > 1800

2: period_number = 7 AND
track_number > 30

3: period_number  < 5 AND
track_number < 50

4: (period_number = 1 OR
period_number = 2) AND
track_number > 30

5: period_number  = 3  AND
track_number < 10
```

Response time

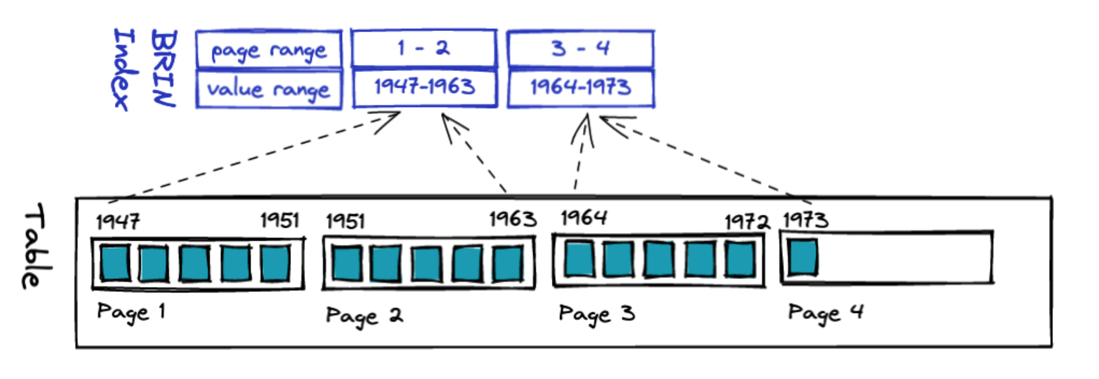
- Adding more periods to test database



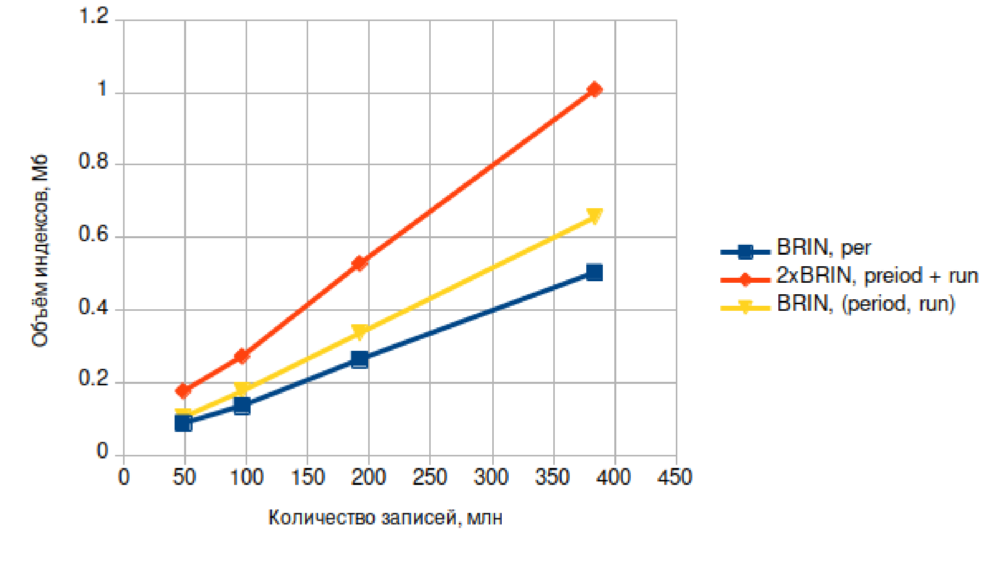


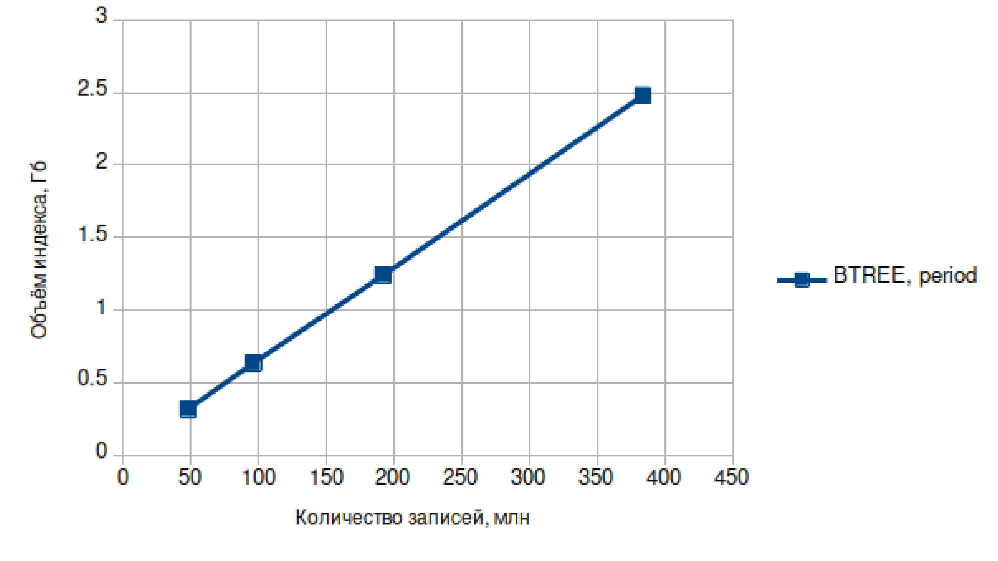
Index size on disk

- BRIN vs. BTREE
  - Overall, BRIN (Block Range Index) works better for indexing columns having some natural correlation with their physical location within the table



https://www.crunchydata.com/blog/postgres-indexing-when-does-brin-win
https://www.postgresql.org/docs/current/brin-intro.html



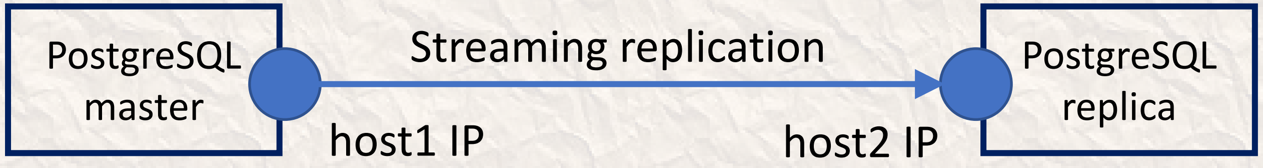


High Availability – Task

- Need for HA
  - EMS as well as other IS are essential for timely obtaining physical results of the experiment
  - From client point of view, connection must be initiated to single IP / domain name
    - We do not want to ask client to keep several addresses like primary/secondary ones
  - Considering 2 to 1, active/passive redundancy
  - Need to avoid split brain and no brain scenarios

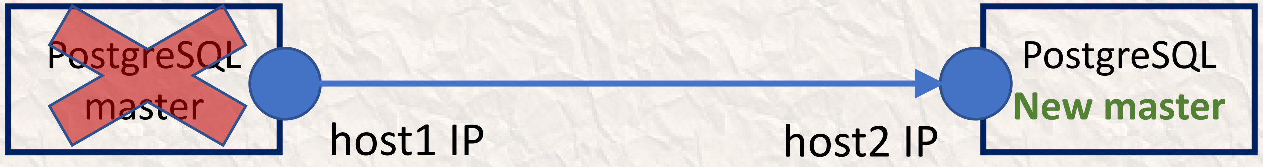
High Availability – Solution

- Base for HA solution
  - PostgreSQL supports streaming replication out of the box (one master to one/many replica servers)
    - https://www.postgresql.org/docs/current/warm-standby.html#STREAMING-REPLICATION
  - Completely synchronous replication is also available (at a performance price)
    - https://www.postgresql.org/docs/current/warm-standby.html#SYNCHRONOUS-REPLICATION



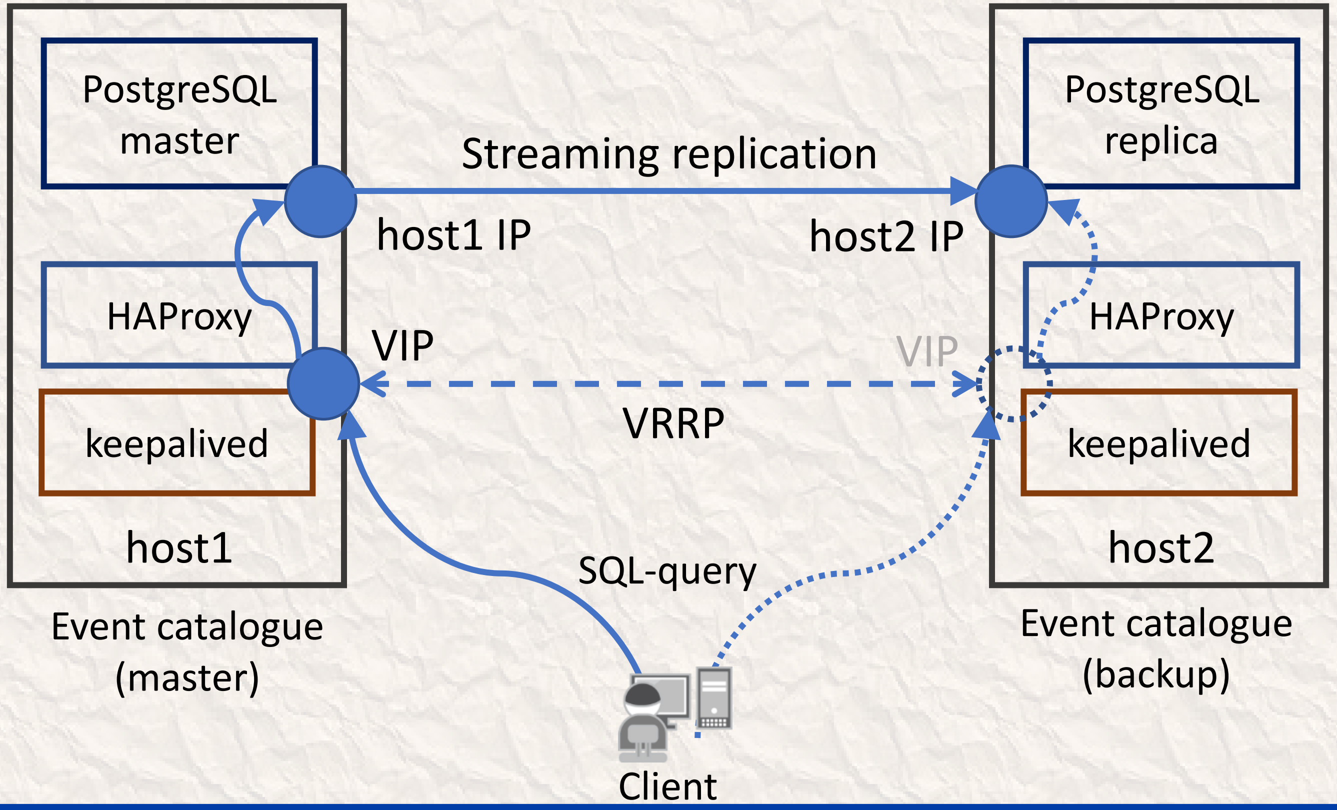
Switchover to new master

- Switchover
  - One command on replica - `pg_ctl promote`
  - Old master must be turned off to avoid split brain
  - Monitoring system can perform switchover (WIP), or it can be done manually
  - It works, but the big question is – where does a client connect?



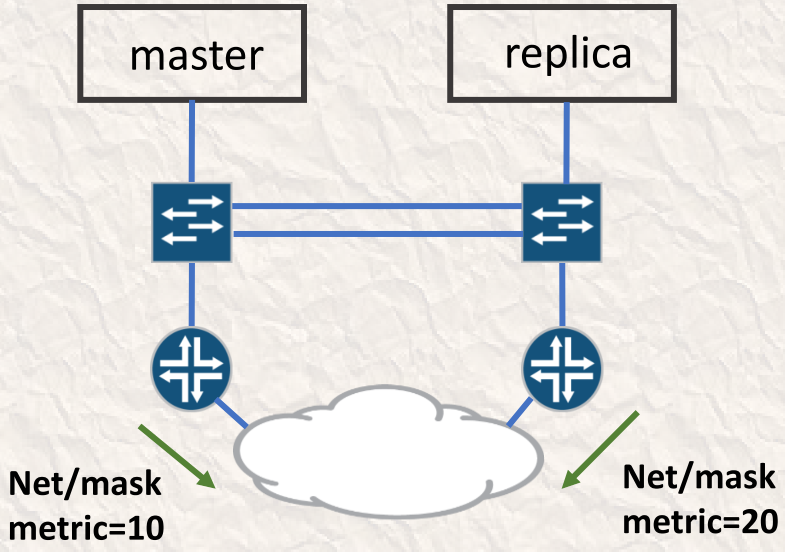
Solution based on VRRP (single L2 domain)

- Keepalived provides virtual IP address for client connection
- This works when both servers are in the same L2 (broadcast) domain



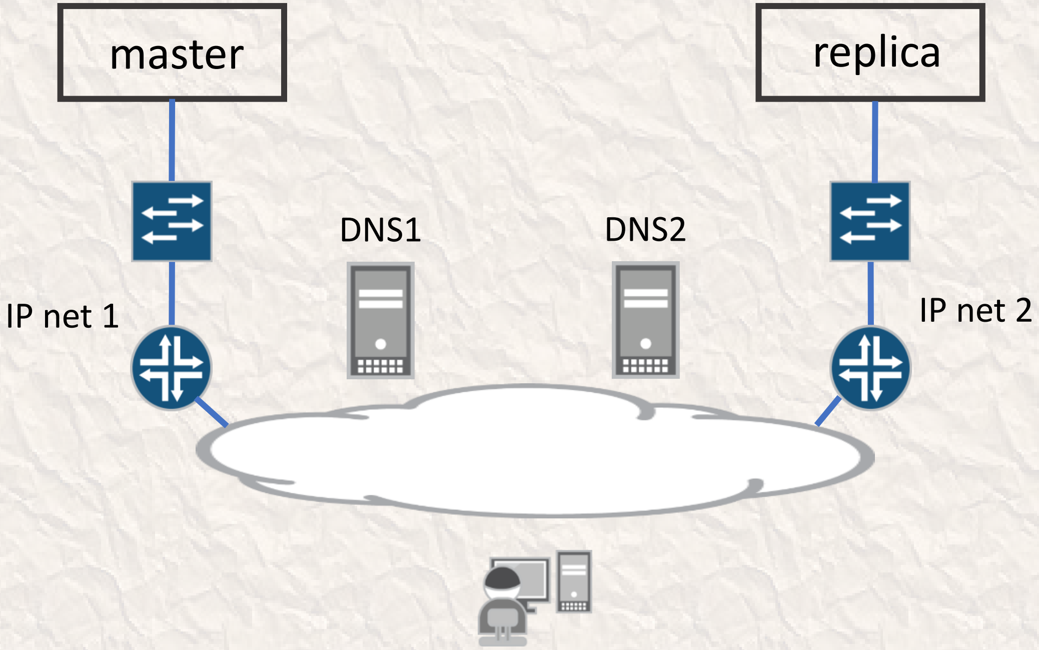
Avoiding single point of failure

- VRRP-based solution can be considered final if:
  - L2-segment is built with redundancy (both for links and switches)
  - VIP’s network is announced from at least two routers
  - Not possible to implement without access to network infrastructure



Solution based on DNS

- Solution details (WIP)
  - PostgreSQL replication unmodified
  - Client connection to host/domain name (needs DNS settings)
  - Monitoring system performs switchover
    - Change DNS record
    - Perform `pg_ctl promote`
  - Switchover time determined by DNS TTL settings



EMS Automated Deployment

- Why automated deployment?
  - Manual deployment of a distributed system is slow and error-prone
  - Automation increases speed and predictability
  - Avoids issue of “forgotten step” in documentation
  - EMS instance may be deployed by other NICA experiments
- Main components of solution
  - Ansible
  - Docker
- Inputs
  - EMS configuration as YAML template
  - Deployment configuration as Ansible variables in hosts file
    - To be replaced by unified JSON config (WIP)

Ansible Playbook example (abbreviated)

```
(env) [lab@alma1 ems-deploy]$ cat deploy-pgsql.pb.yaml
---
- name: Deploy PostgreSQL on Event Catalogue hosts
  hosts: event_catalogue
  become: yes

  tasks:

  - name: Install packages
    dnf: "name={{ item }} state=present"
    with_items:
      - postgresql
      - postgresql-server

  - name: Install Python packages
    pip: "name={{ item }}  state=present"
    with_items:
      - psycopg2-binary

…
```

…

```
- name: Check if PostgreSQL is initialized
   ansible.builtin.stat:
     path: "/var/lib/pgsql/data/pg_hba.conf"
   register: postgres_data

 - name: Initialize PostgreSQL
   command: "postgresql-setup initdb"
   when: not postgres_data.stat.exists

 - name: Start and enable services
   service: "name={{ item }} state=started enabled=yes"
   with_items:
     - postgresql
```

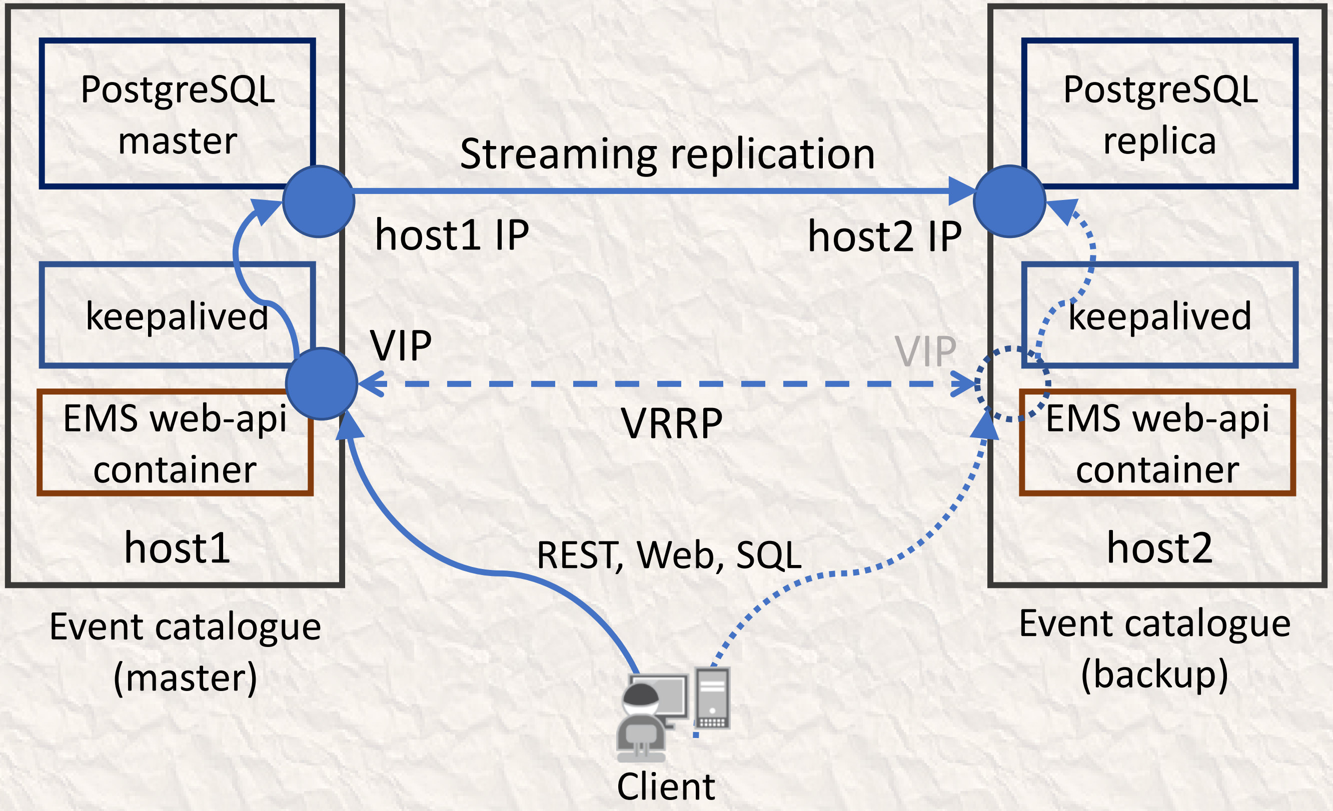
…

Deployment example (abbreviated)

```
[lab@alma1 ems-deploy]$ source env/bin/activate
(env) [lab@alma1 ems-deploy]$ ansible-playbook deploy-pgsql.pb.yaml

PLAY [Deploy PostgreSQL on Event Catalogue hosts] ****************************************

TASK [Gathering Facts] *******************************************************************
ok: [ems2]
ok: [ems1]

TASK [Install packages] ******************************************************************
ok: [ems1] => (item=postgresql)
ok: [ems2] => (item=postgresql)
ok: [ems1] => (item=postgresql-server)
ok: [ems2] => (item=postgresql-server)

…

TASK [Apply SQL schema file] *************************************************************
changed: [ems1]

PLAY RECAP *******************************************************************************
ems1                       : ok=13   changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ems2                       : ok=16   changed=4    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0

(env) [lab@alma1 ems-deploy]$ ansible-playbook deploy-vrrp.pb.yaml
(env) [lab@alma1 ems-deploy]$ ansible-playbook deploy-web-api-docker.pb.yaml
```

The Result

- After running the three playbooks:

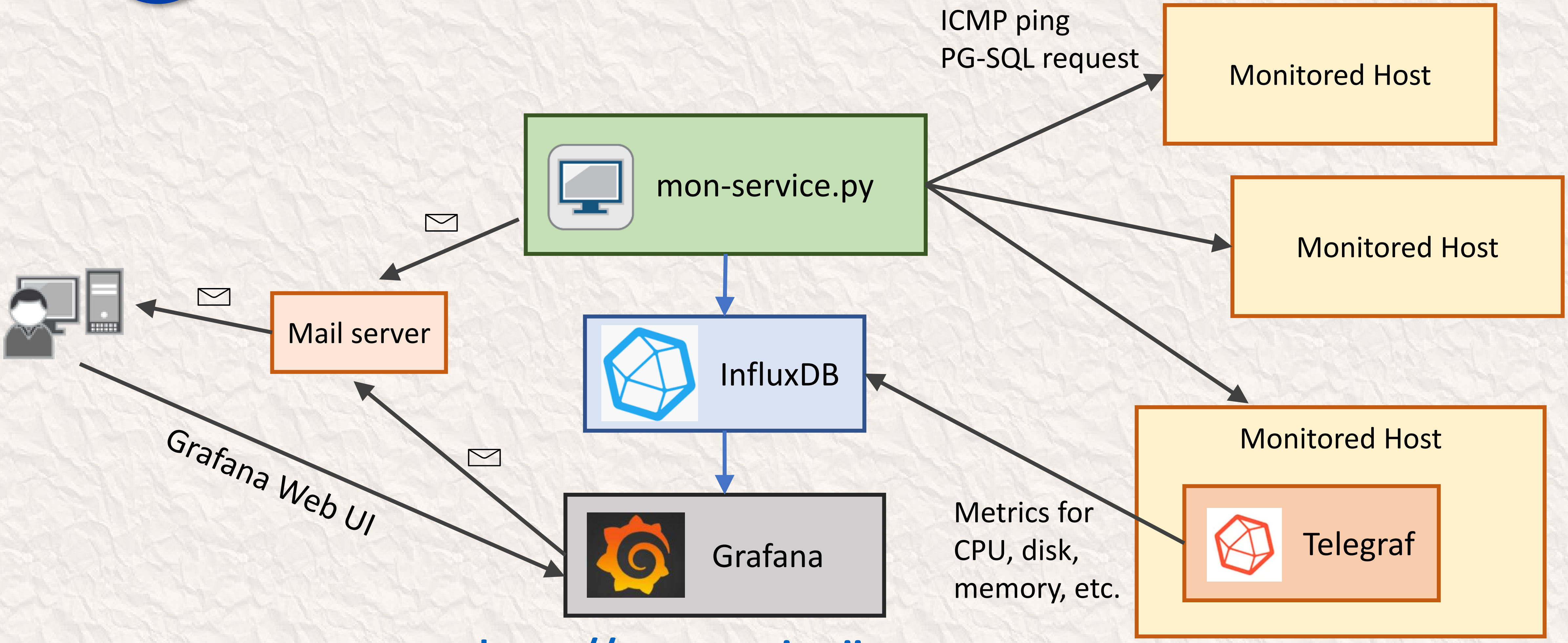


Monitoring Service Overview

Monitoring Service - Task

- Monitoring Service Features
  - Ping and PG-SQL request to check database server status
  - Configurable via JSON file
  - Email notifications
  - Response time stored in InfluxDB
  - Use Grafana for visualization and additional alerting
  - Monitor server parameters such as Disk, CPU, Memory, etc.
- Planned new features:
  - Web-services monitoring
  - API endpoint monitoring
  - HA switchover functionality

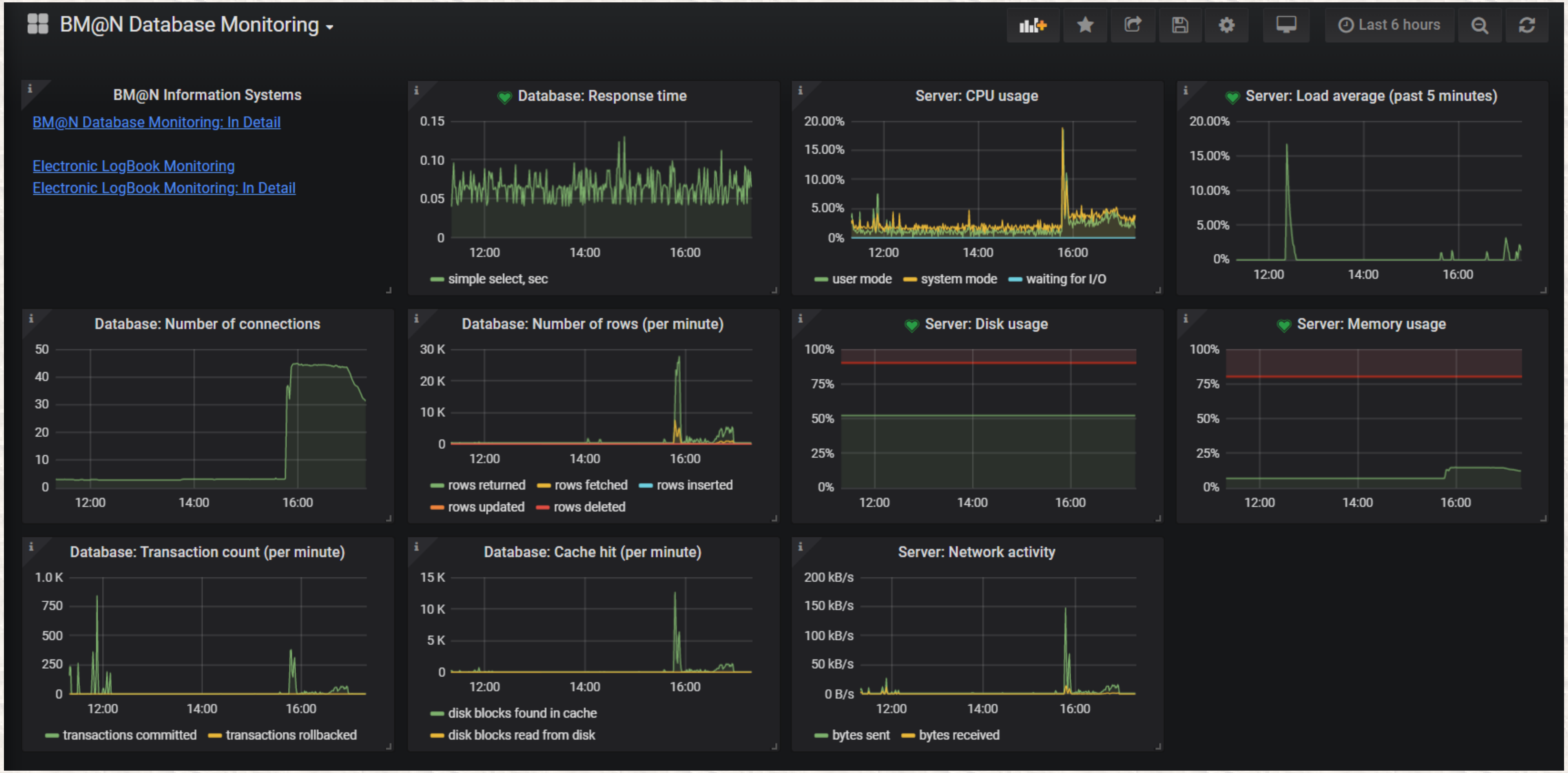
Monitoring Service - Components



https://mon-service.jinr.ru

Monitoring Service View Example

https://mon-service.jinr.ru



Thank You!