Monitoring with Prometheus

Agenda

Observability meaning

Difference between (logs,traces,metrics)

Metrics types

What is Prometheus

Prometheus architecture

Prometheus installing (prometheus, node exporter, cient library)

Configuring prometheus.yaml

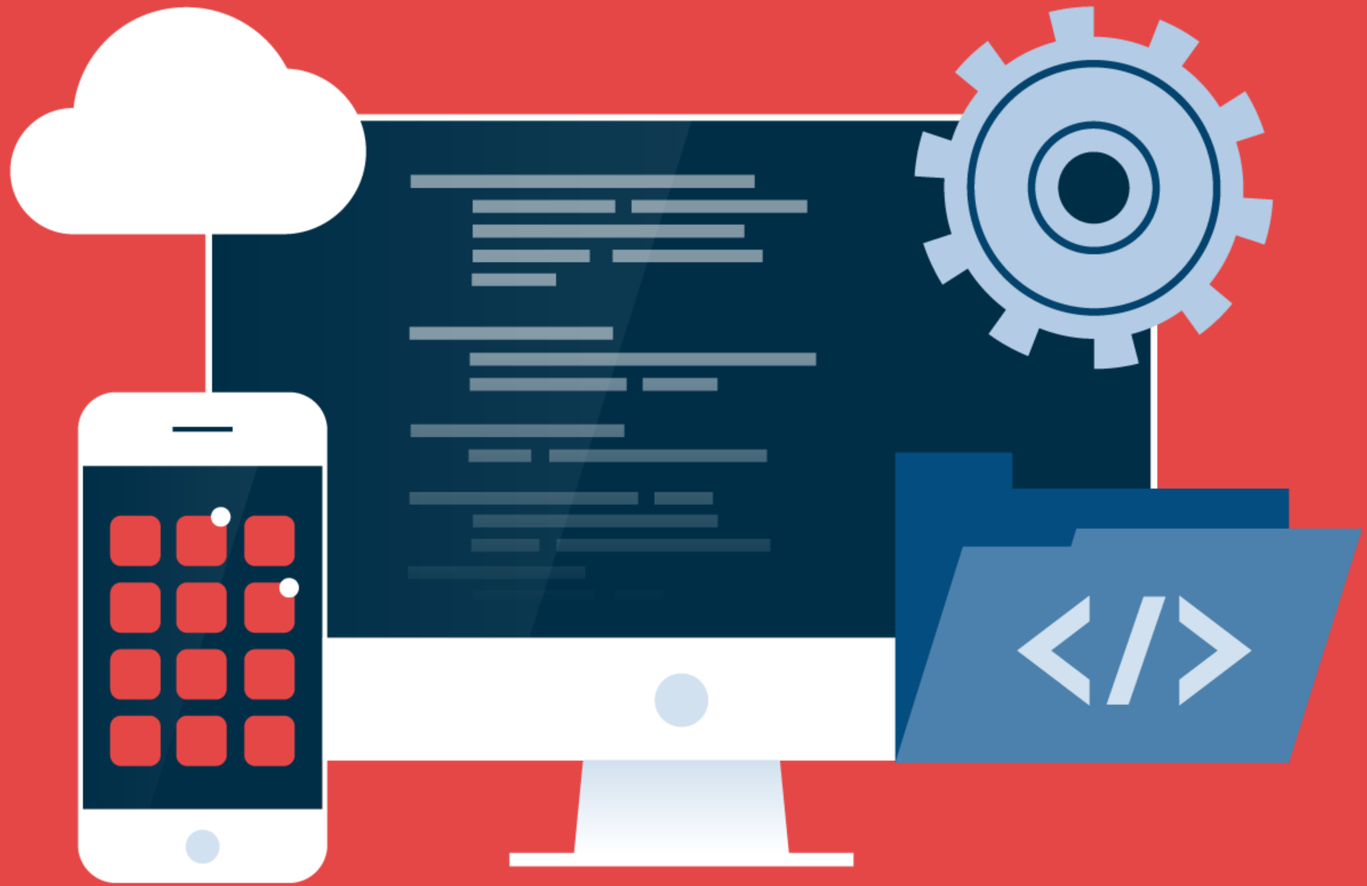
PromQl

What is Observability

What is Observability

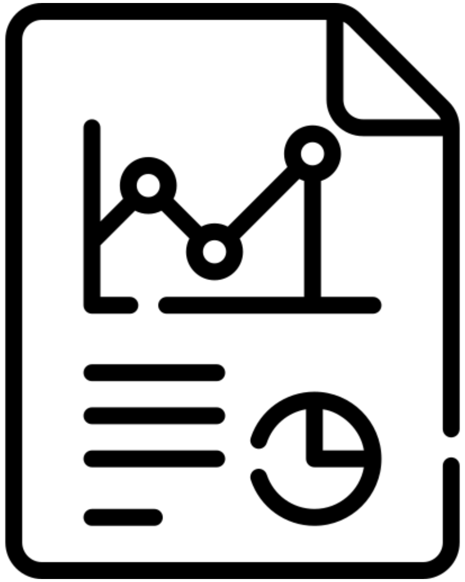
- **Observability**- the ability to understand and measure the state of a system based upon data generated by the system
- Observability allows you to generate actionable outputs from **unexpected** scenarios
- Observability will help:
  - Give better insight into the internal working of a system/application
  - Speed up troubleshooting
  - Detect hard to catch problems
  - Monitor performance of an application
  - Improve cross-team collaboration

The main purpose of observability is to better understand the internals of your system



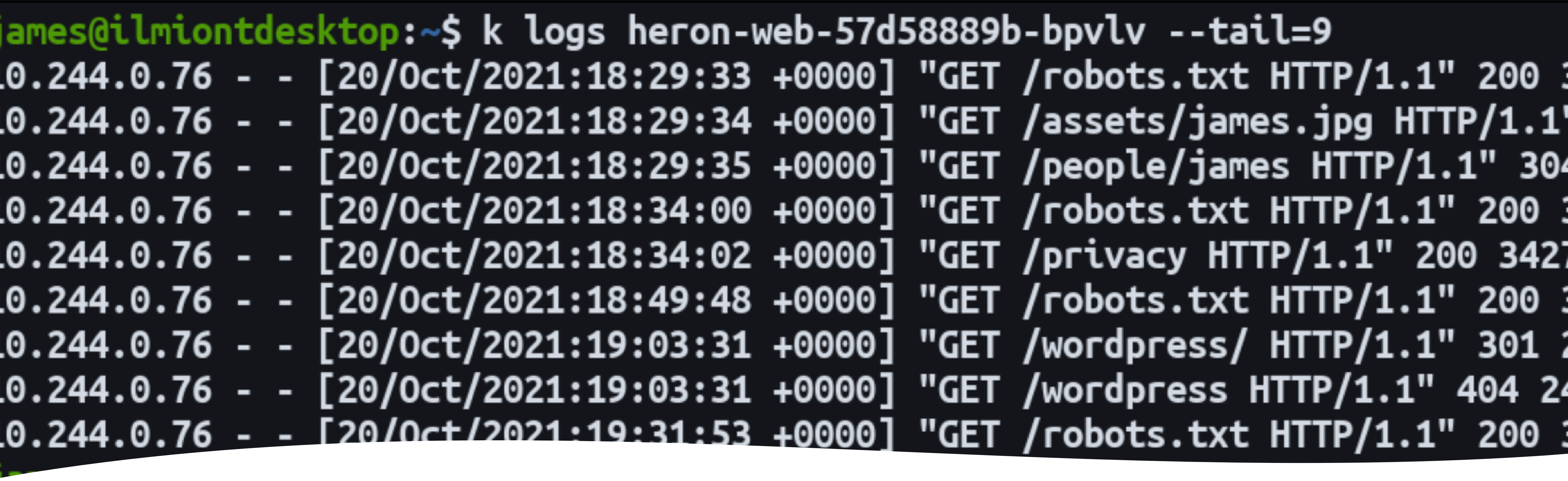
How do we accomplish observability ?

logging



Metrics

traces



```
james@ilmiontdesktop:~$ k logs heron-web-57d58889b-bpvlv --tail=9
0.244.0.76 - - [20/Oct/2021:18:29:33 +0000] "GET /robots.txt HTTP/1.1" 200
0.244.0.76 - - [20/Oct/2021:18:29:34 +0000] "GET /assets/james.jpg HTTP/1.1"
0.244.0.76 - - [20/Oct/2021:18:29:35 +0000] "GET /people/james HTTP/1.1" 30
0.244.0.76 - - [20/Oct/2021:18:34:00 +0000] "GET /robots.txt HTTP/1.1" 200
0.244.0.76 - - [20/Oct/2021:18:34:02 +0000] "GET /privacy HTTP/1.1" 200 342
0.244.0.76 - - [20/Oct/2021:18:49:48 +0000] "GET /robots.txt HTTP/1.1" 200
0.244.0.76 - - [20/Oct/2021:19:03:31 +0000] "GET /wordpress/ HTTP/1.1" 301
0.244.0.76 - - [20/Oct/2021:19:03:31 +0000] "GET /wordpress HTTP/1.1" 404 2
0.244.0.76 - - [20/Oct/2021:19:31:53 +0000] "GET /robots.txt HTTP/1.1" 200
```

logging

- Logs are records of events that have occurred and encapsulate information about the specific event
- Logs are comprised of :
  - Timestamp of when the log occurred
  - Message containing information

Traces

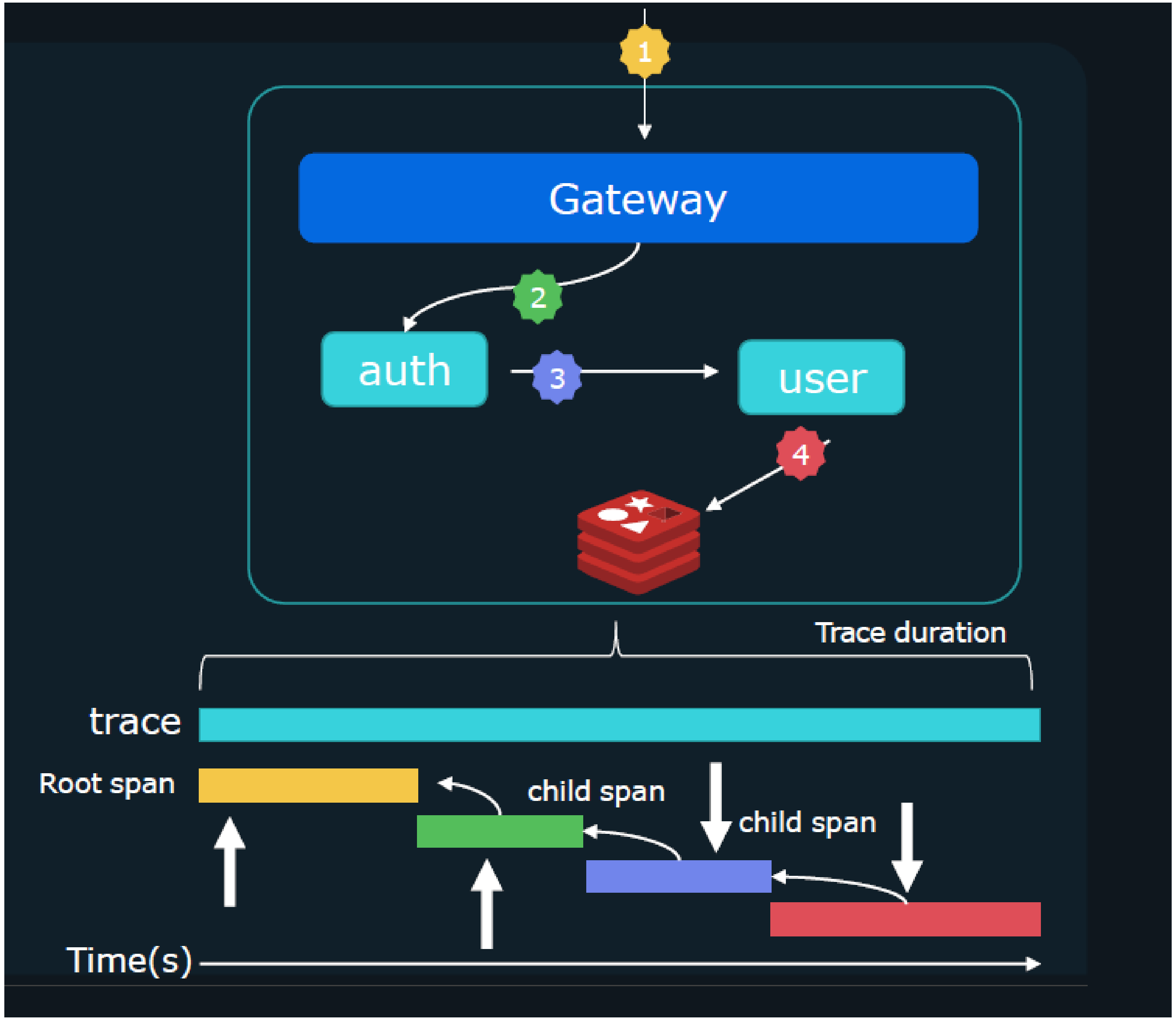
Traces allow you to follow operations as they traverse through various systems & services

So we can follow an individual request and see it flow through our system hop by hop

Traces help us connect the dots on how processes and services work together

traces

- Each trace has a trace-id that can be used to identify a request as It traverses the system
- Individual events forming a trace are called spans
- Each span tracks the following:
  - Start time
  - Duration
  - Parent –id



Metrics

- Metrics provide information about the state of a system using numerical values
    - CPU Load
    - Number of open files
    - HTTP response times
    - Number of errors

The data collected can be aggregated over time and graphed using visualization tools to identify trends over time

Prometheus

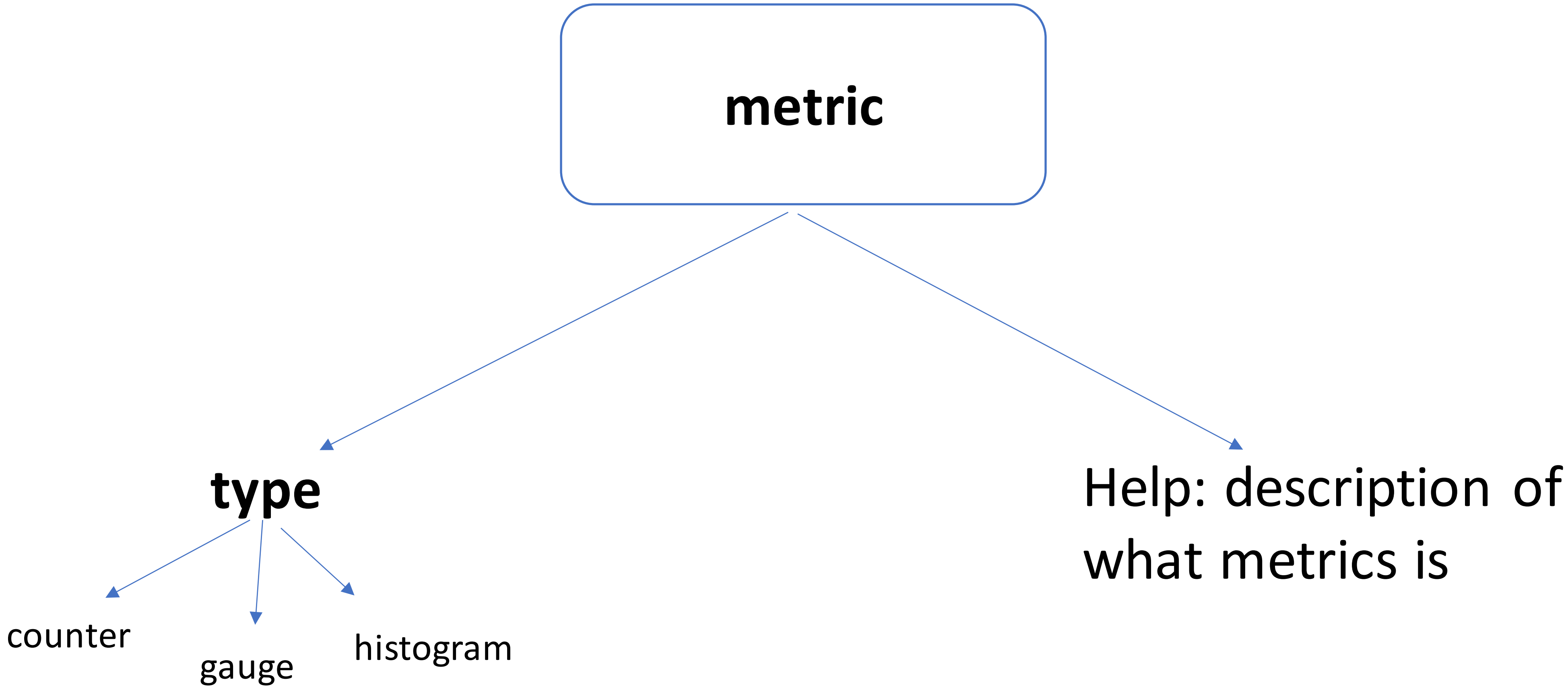
logs

Metrics

Traces

Prometheus is a monitoring solution that is responsible for collecting and aggregating **Metrics**

Metric types



metric

type

counter

gauge

histogram

Help: description of what metrics is

Metric type: Counter

```
http_requests_total 10000000

cpu_seconds_total 3000

                                        @ 22:00
```

```
http_requests_total 970000

cpu_seconds_total 3000

                                        @ 21:00
```

```
http_requests_total - http_requests_total offset 1h

= 300000
```

Metric type: Gauge

```
http_requests_active 2000
memory_allocated_bytes 4.832e+09
                                              @ 22:00
```

```
http_requests_active 900
memory_allocated_bytes 3.642e+09
                                              @ 21:00
```

```
memory_allocated_bytes / (1024*1024*1024)
= 4.5    # gigabytes
```

Metric type: Histogram

```
calculation_seconds_bucket{le="1"}  0
calculation_seconds_bucket{le="5"}  3
calculation_seconds_bucket{le="10"} 6
calculation_seconds_bucket{le="20"} 9
calculation_seconds_bucket{le="60"} 10
                                        @ 21:00
```

```
calculation_seconds_bucket{le="20"} /
calculation_seconds_bucket{le="+Inf"}    # SLA
```

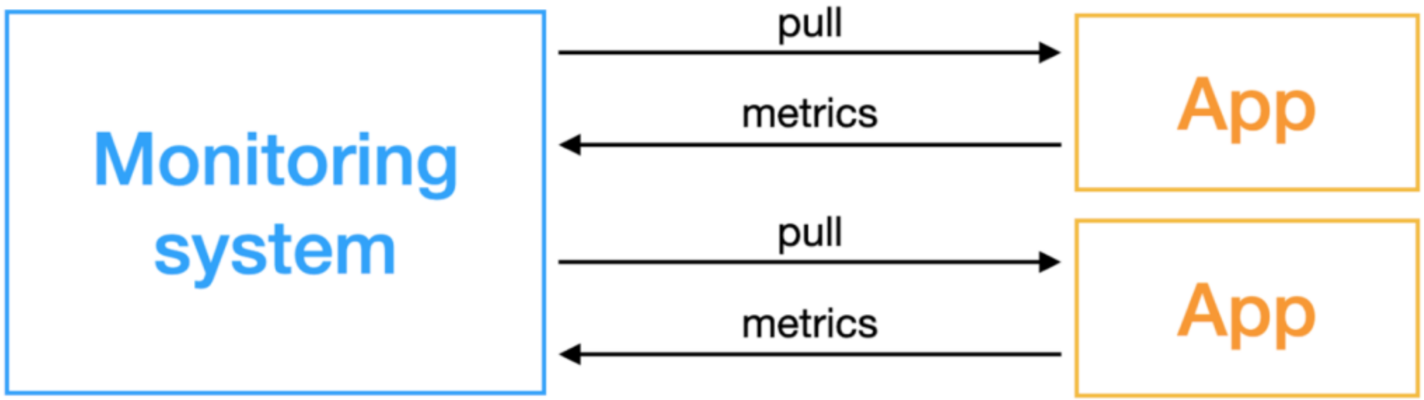
What is Promethus

- Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud.
- Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community.
- It is now a standalone open source project and maintained independently of any company. To emphasize this, and to clarify the project's governance structure, Prometheus joined the Cloud Native Computing Foundation in 2016 as the second hosted project, after Kubernetes.
- Prometheus collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels.

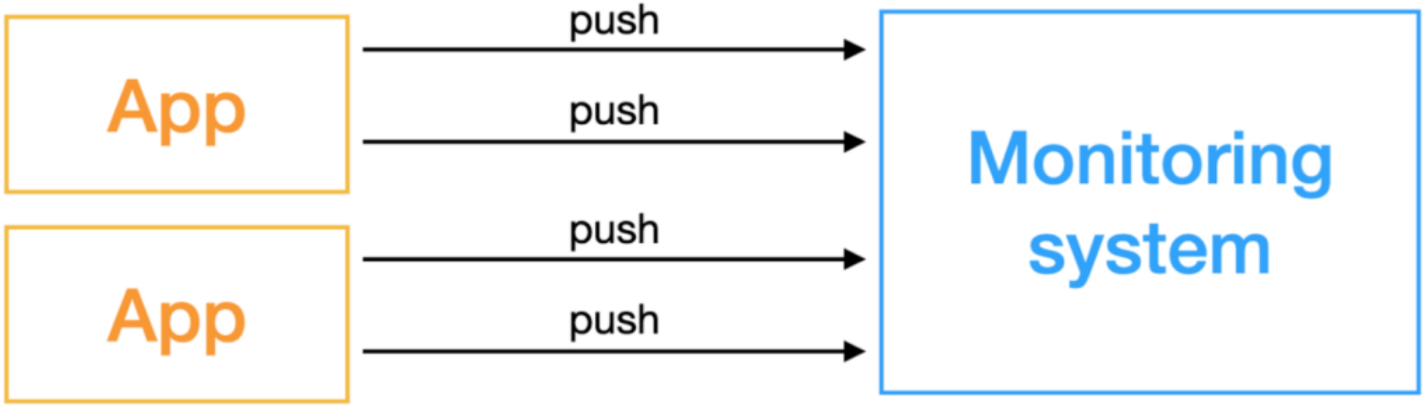
Why using Prometheus

- **Multidimensional data model –** Using time-series data, which is identified by metric name and key-value pairs.
- **PromQL –** A flexible querying language that can leverage the multi-dimensional data model.
- **No reliance on distributed storage –** All single server nodes remain autonomous.
- **Pull model –** Prometheus can collect time-series data by actively “pulling” data over HTTP.
- **Pushing time-series data –** Available through the use of an intermediary gateway.
- **Monitoring target discovery –** Available through static configuration or service discovery.
- **Visualization –** Prometheus offers multiple types of graphs and dashboards.

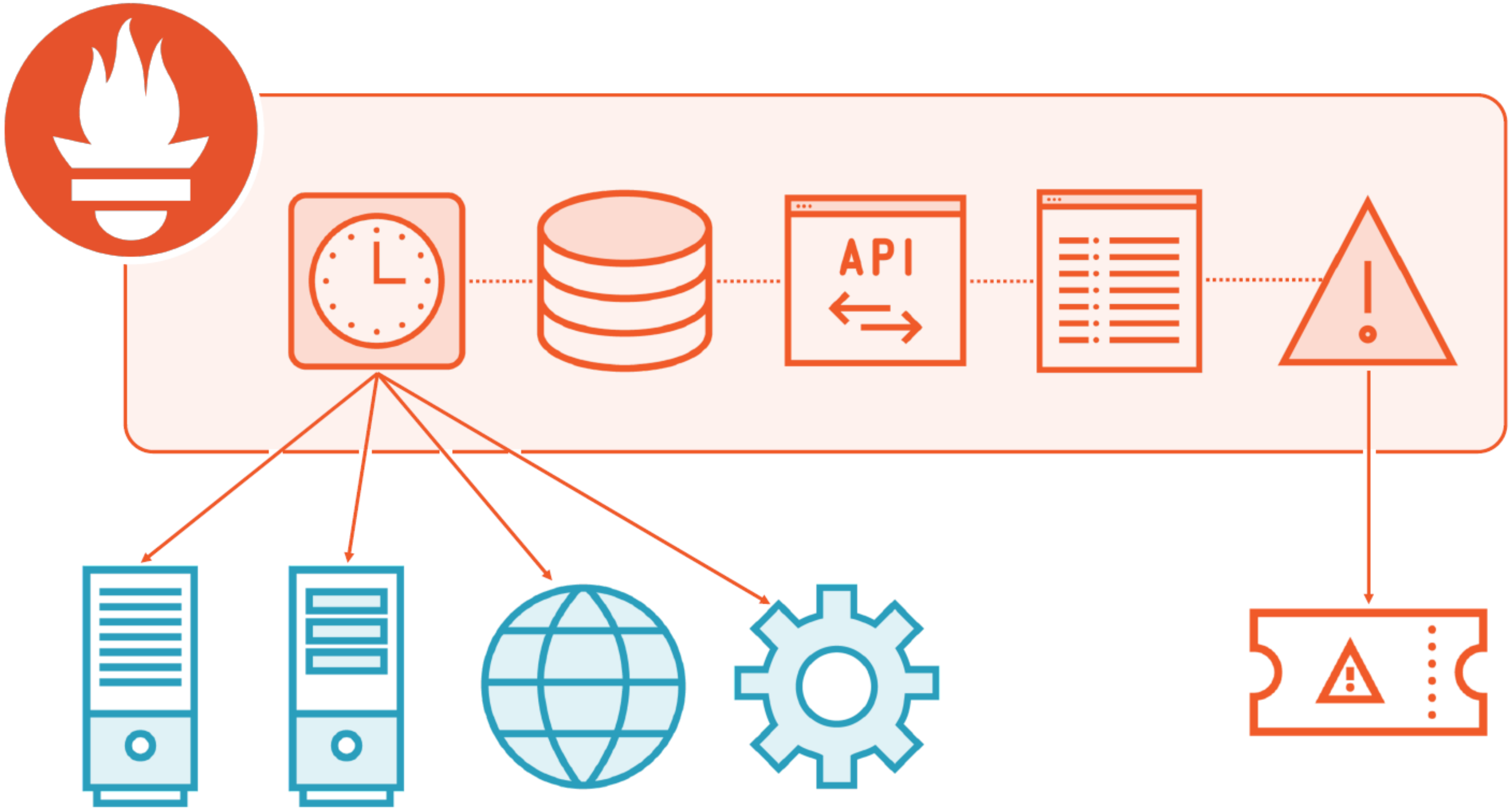
Pull-based system



Push-based monitoring system

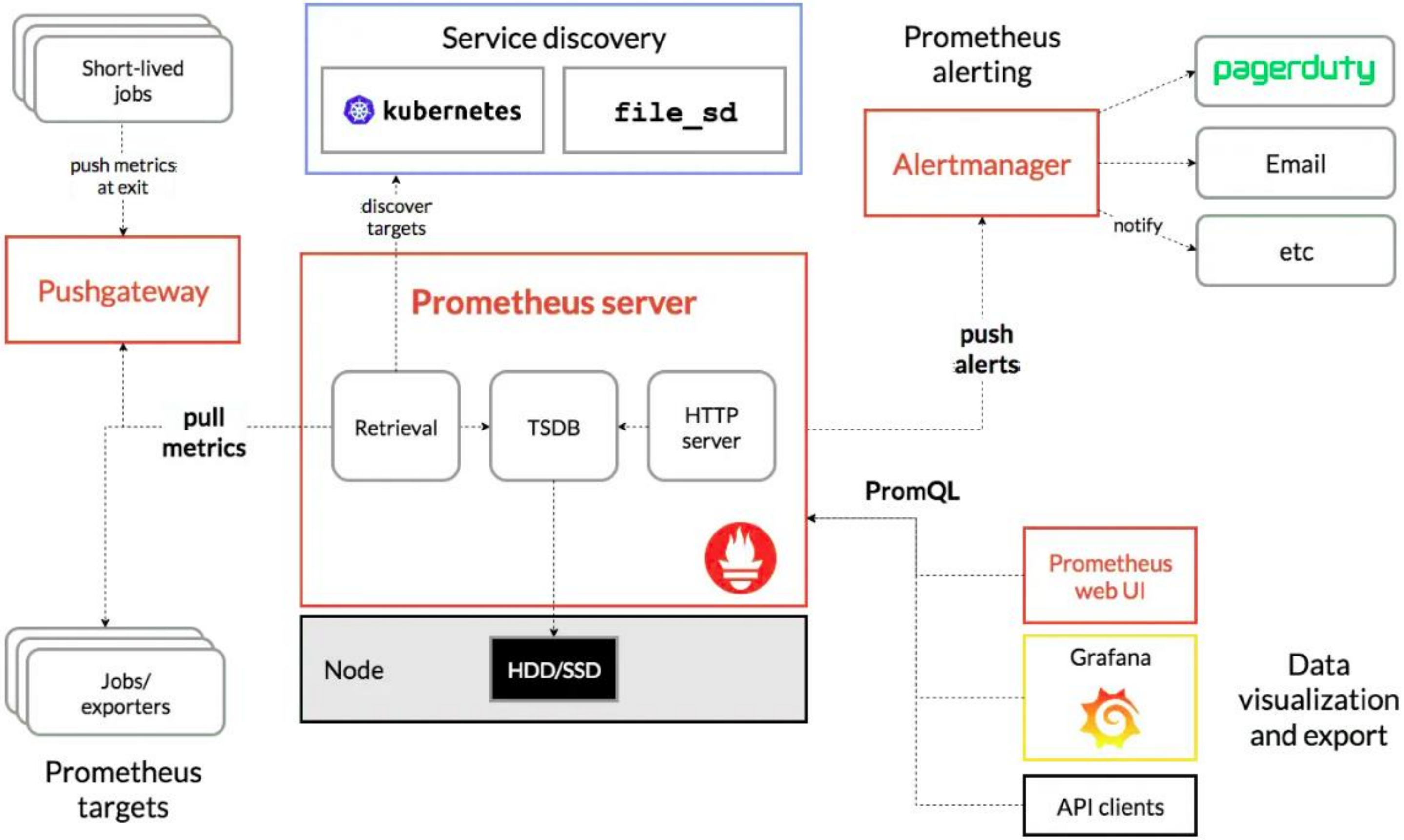


Prometheus Architecture

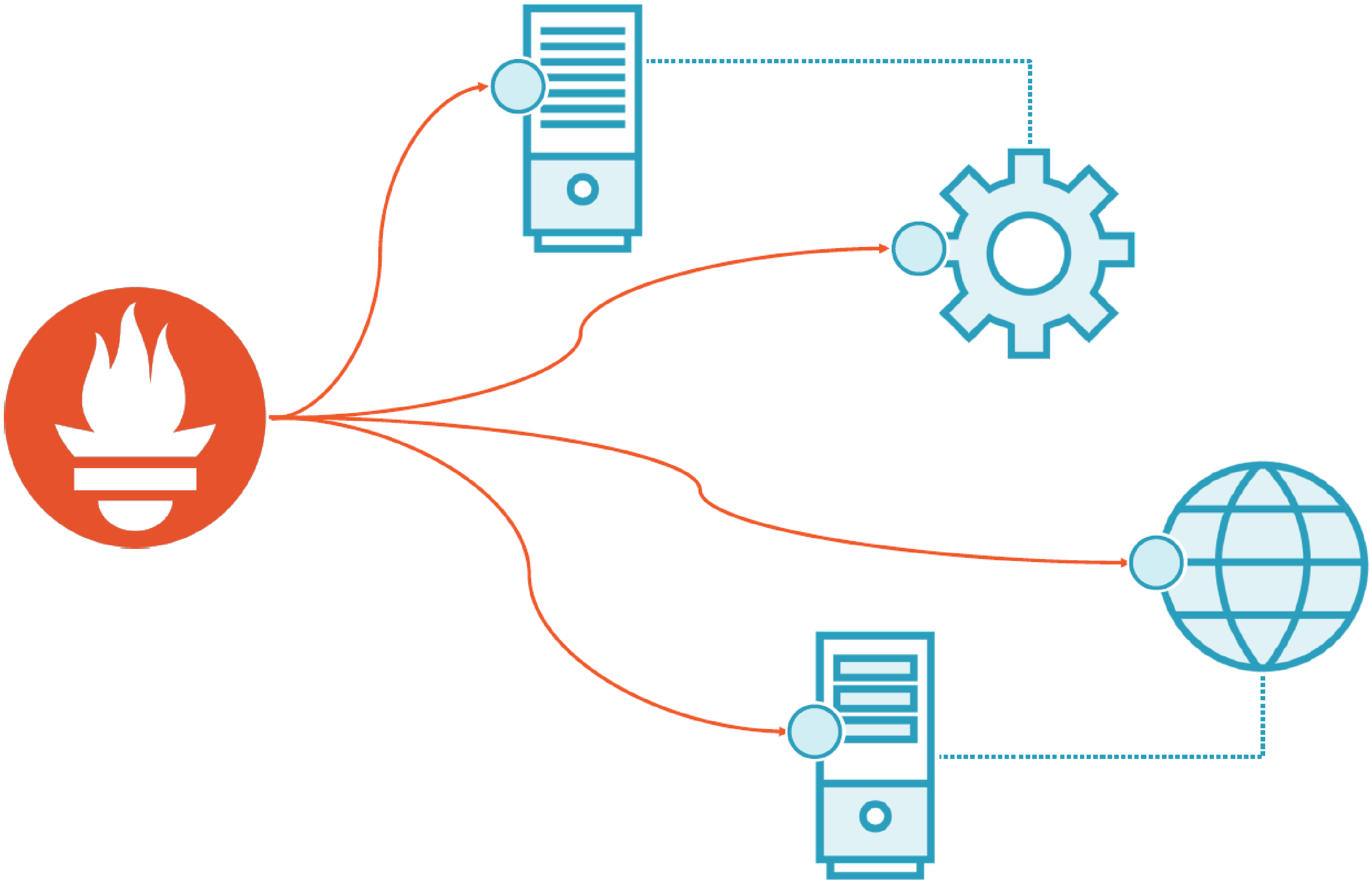


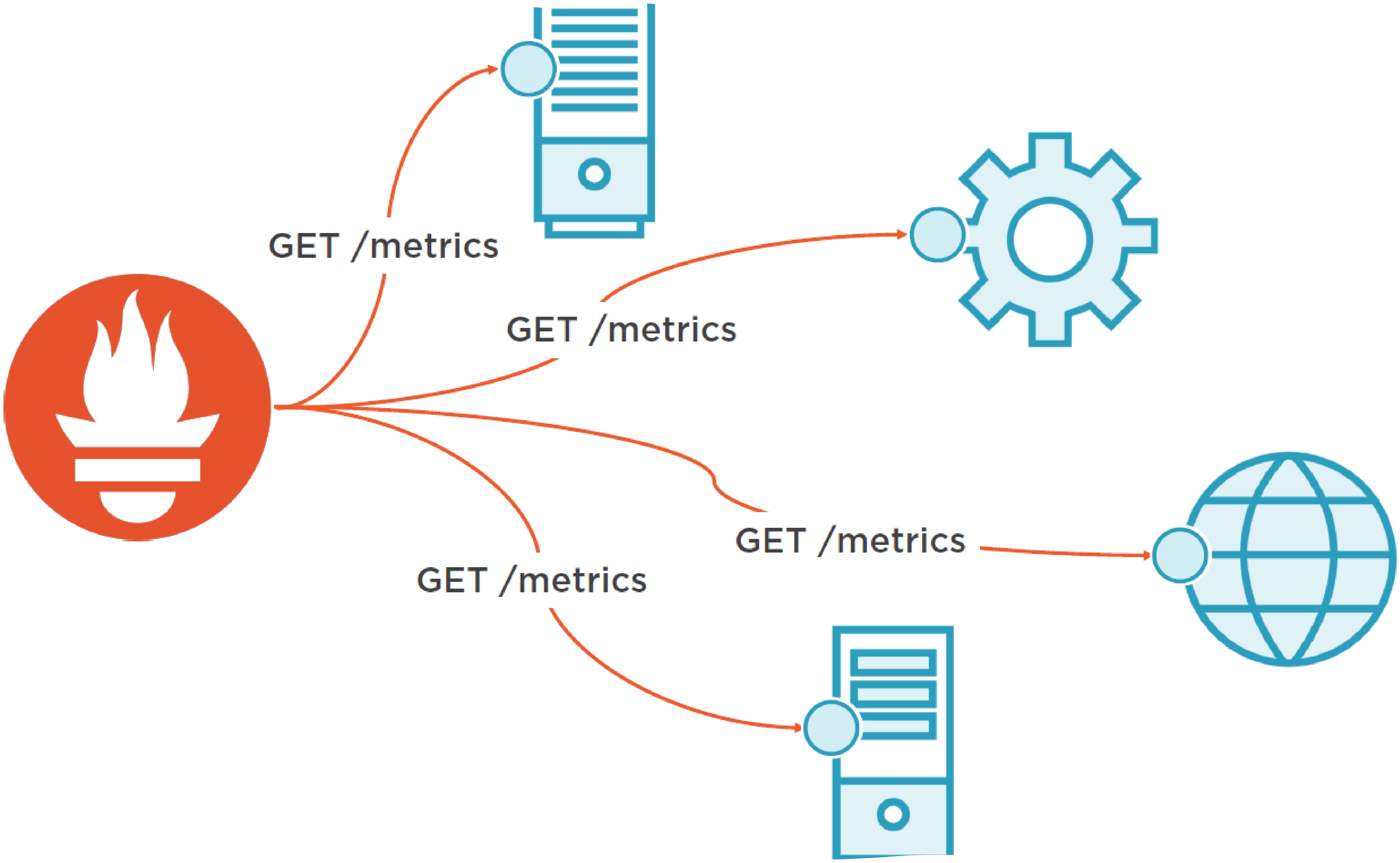
Prometheus Architecture

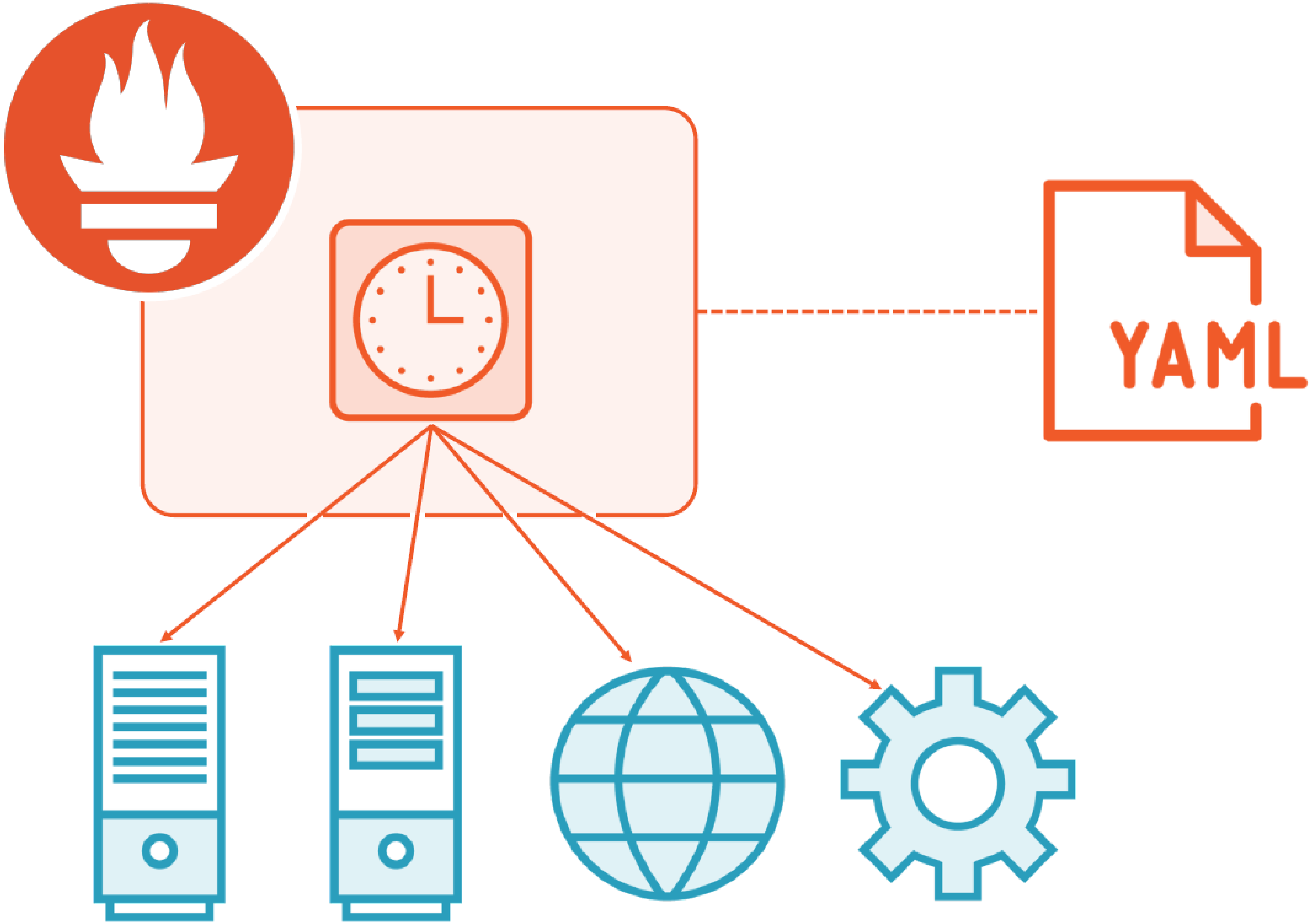
Here is the high-level **architecture of Prometheus.**



source: prometheus.io

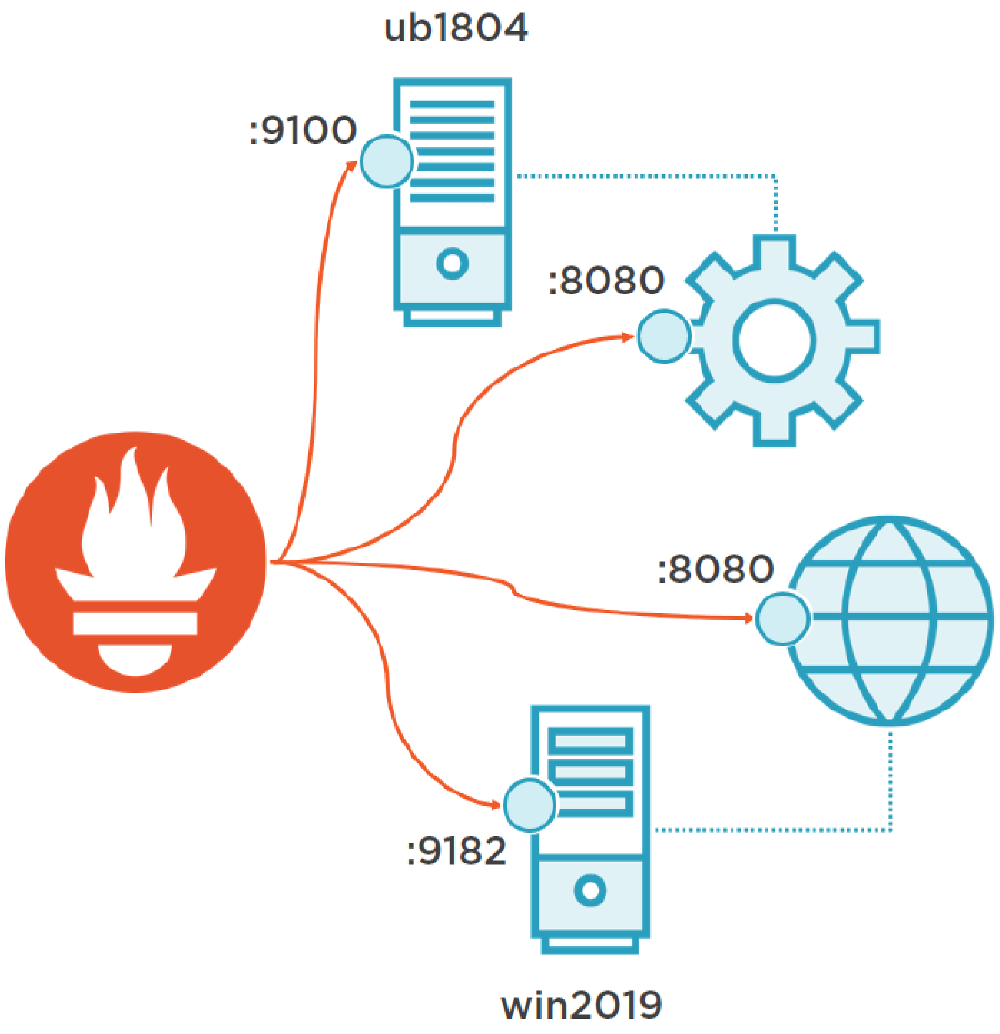



GET /metrics
GET /metrics
GET /metrics
GET /metrics



- Targets
- Scrape interval
- Metric updates

```
scrape_configs:

  - job_name: 'linux'
    static_configs:
      - targets: ['ub1804:9100']

  - job_name: 'batch'
    static_configs:
      - targets: ['ub1804:8080']

  - job_name: 'windows'
    static_configs:
      - targets: ['win2019:9182']

  - job_name: 'web'
    static_configs:
      - targets: ['win2019:8080']
```



Demo

Running Prometheus

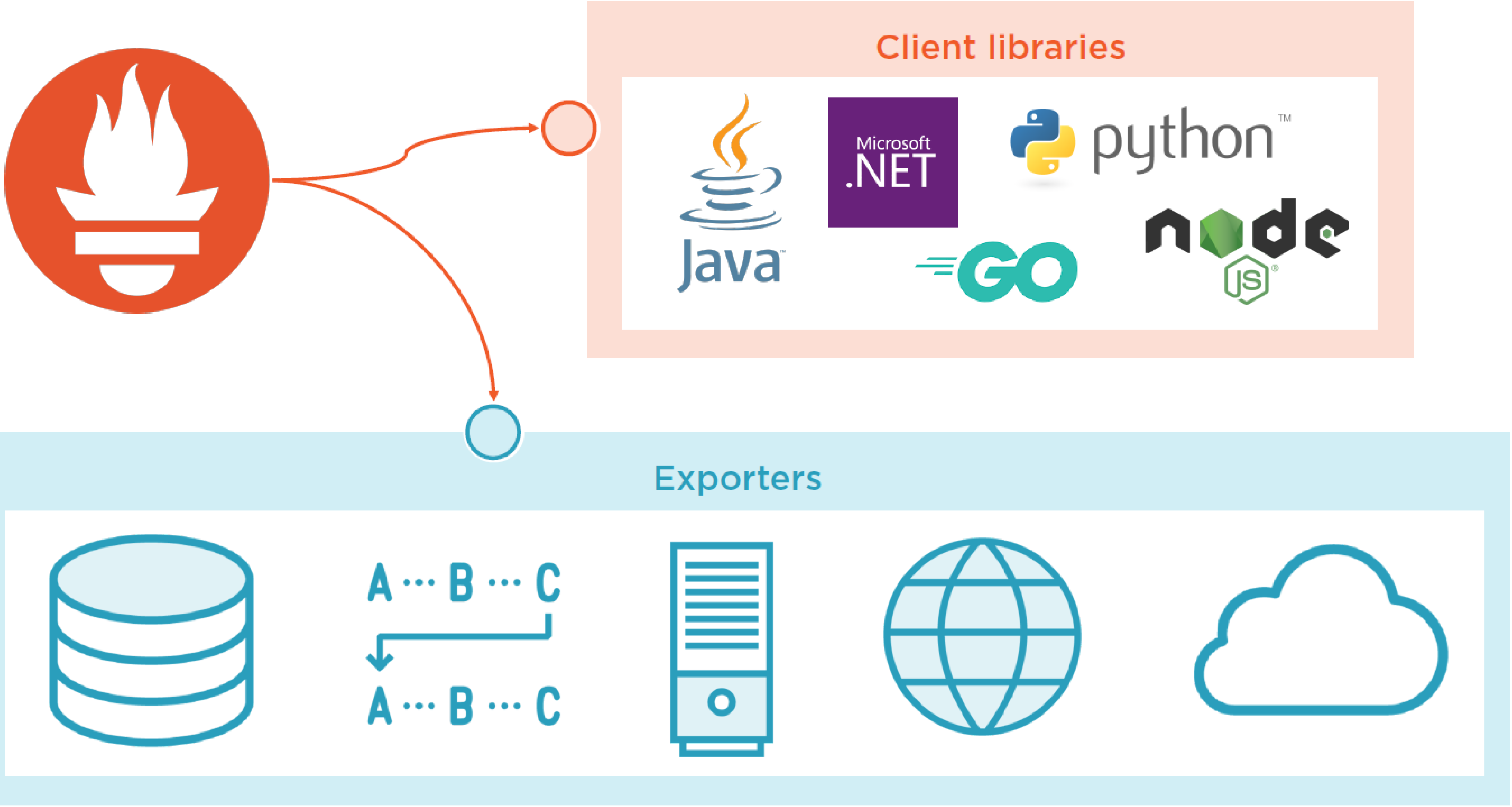
- Download options
- Running the server
- Exploring the UI

Labels

```
http_requests_total{code="200",path="/"} 800
http_requests_total{code="500",path="/p1"} 12980
http_requests_total{code="500",path="/p2"} 1064
http_requests_total{code="404",path="/p3"} 36
```

```
sum without(code, path) (http_requests_total)
= 14880     # all requests
```

```
sum without(path) (http_requests_total{code="500"})
= 14044     # all errors
```



Client libraries

Exporters

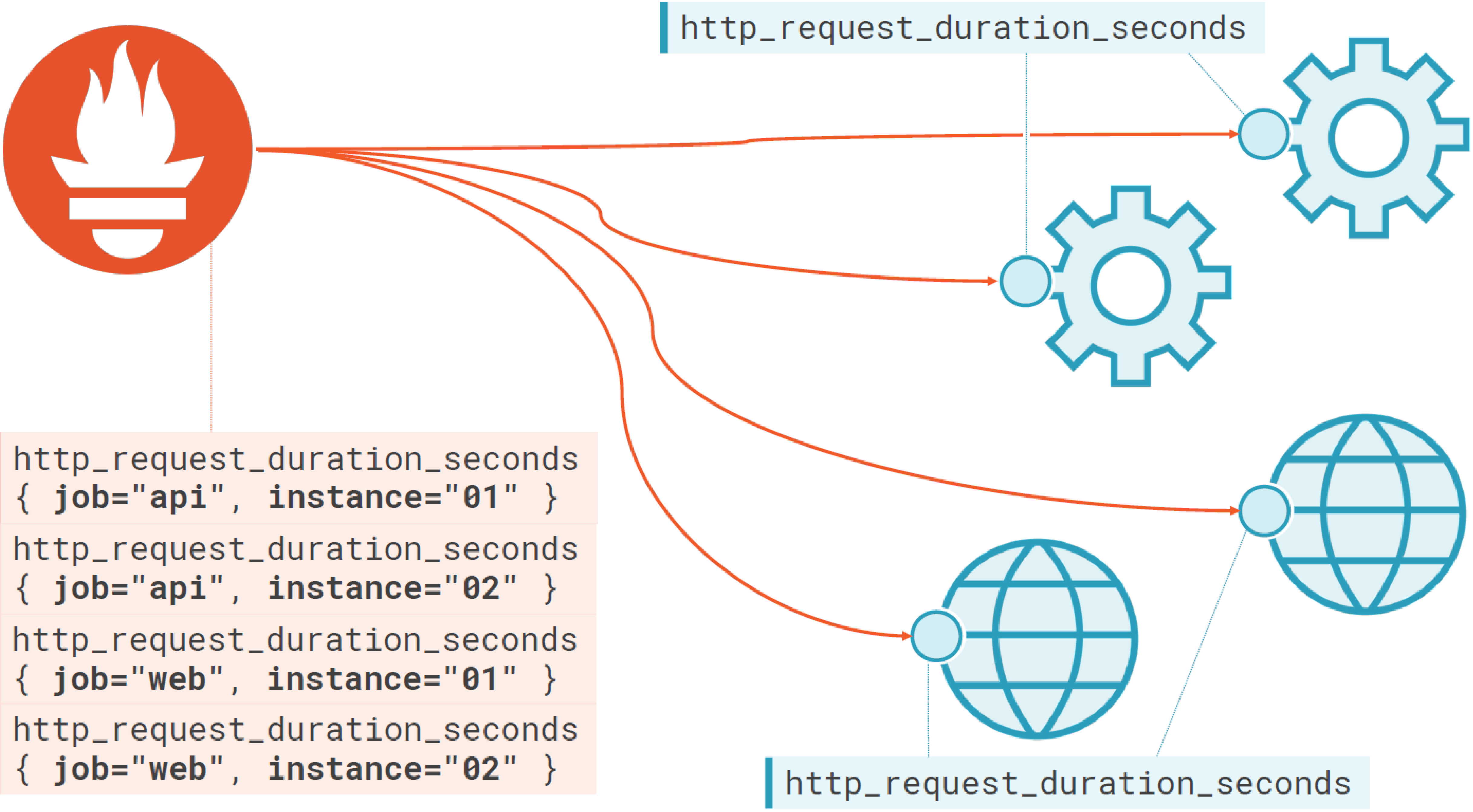
file-sd.yml

```yaml
global:
  scrape_interval:     15s

scrape_configs:
  - job_name: prometheus
    static_configs:
    - targets: [localhost:9090]

  - job_name: web-file
    file_sd_configs:
      - files:
         - 'web.json'
```

web.json

```json
[
    {
      "targets": [ "win2019:9182" ]
    },
    {
      "targets": [ "win2019:8080" ]
    }
]
```

Exploring PromQL Syntax



http_request_duration_seconds

```
http_request_duration_seconds
{ job="api", instance="01" }
http_request_duration_seconds
{ job="api", instance="02" }
http_request_duration_seconds
{ job="web", instance="01" }
http_request_duration_seconds
{ job="web", instance="02" }
```

http_request_duration_seconds

```
worker_jobs_total
{instance="i1",
 status="processed} 150
```

```
sum
without(instance, status)
(worker_jobs_total)
```

instance	job_status	job_count
i1	processed	150

```
SELECT
SUM(job_count) FROM
job_summaries
```

instance	job_status	job_count	timestamp
i1	processed	150	1592210327

170K rows /24hr

- per status
- per instance

~ 50b per row

```
worker_jobs_total
{instance="i1",
 status="processed}
```

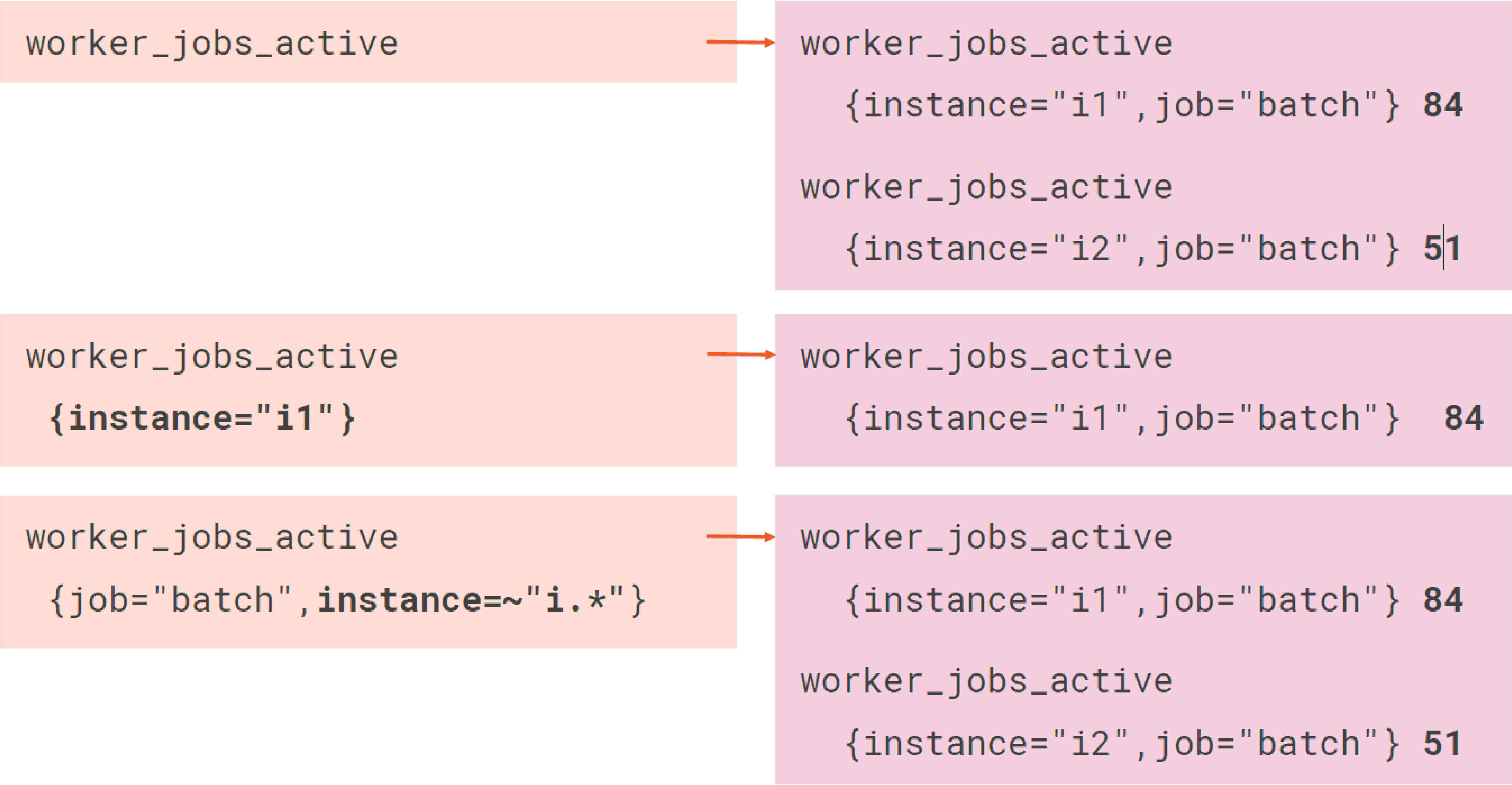
```
150 @ 1592210327
158 @ 1592210357
210 @ 1592210387
235 @ 1592210417
```

One time series
- per status
- per instance

< 2b per sample

```
histogram_quantile(
  0.90,
  sum without(code,instance)(
   rate(http_request_seconds[5m]
)))
```

```
select top (1) percentile_cont(0.90)
 within group (order by avg_duration)
 over () as percentile_90
from (select avg(duration) as avg_duration,
      percentile_cont(0.90)
        over (order by avg(duration))
       as percentile_90
      from t
      group by status_code, instance
     ) t;
```



```
worker_jobs_active
```

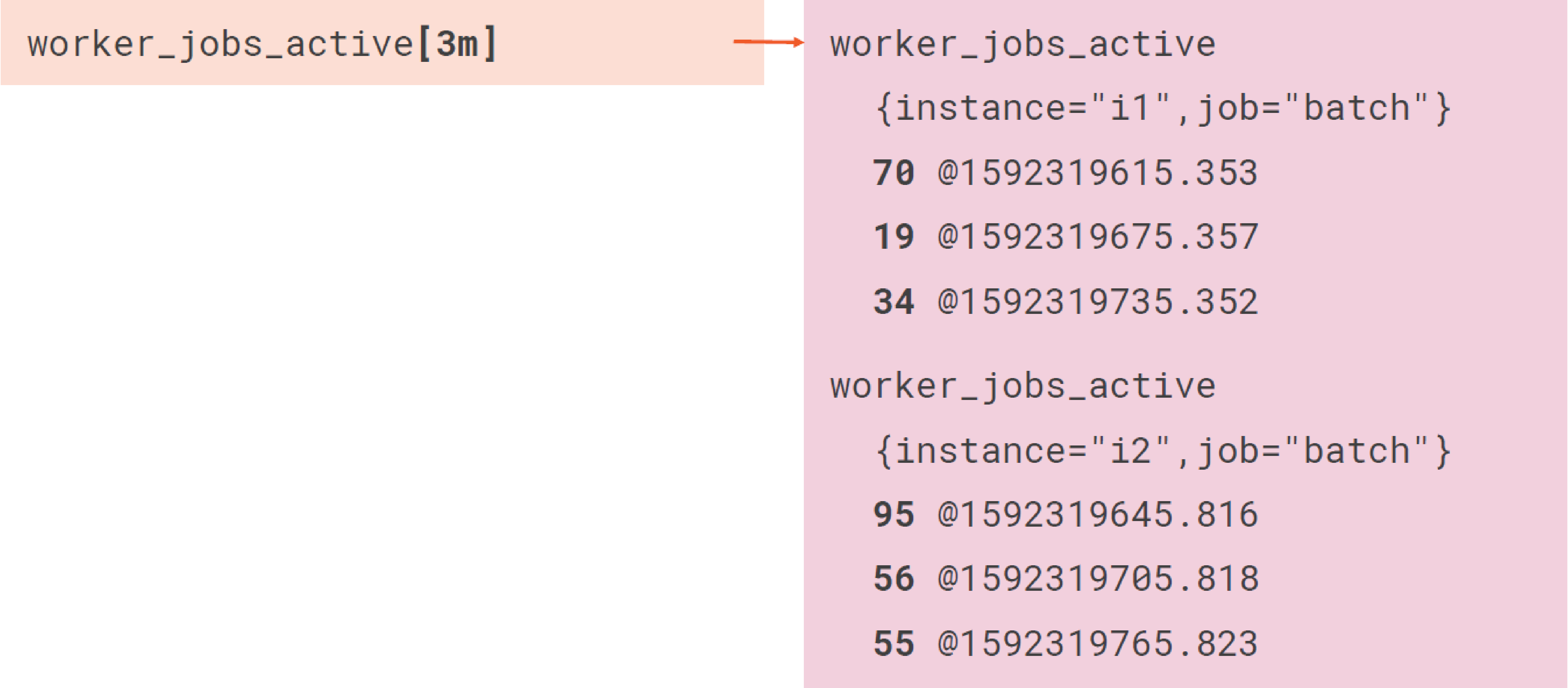
```
worker_jobs_active
  {instance="i1",job="batch"} 84

worker_jobs_active
  {instance="i2",job="batch"} 51
```

```
worker_jobs_active
 {instance="i1"}
```

```
worker_jobs_active
  {instance="i1",job="batch"}  84
```

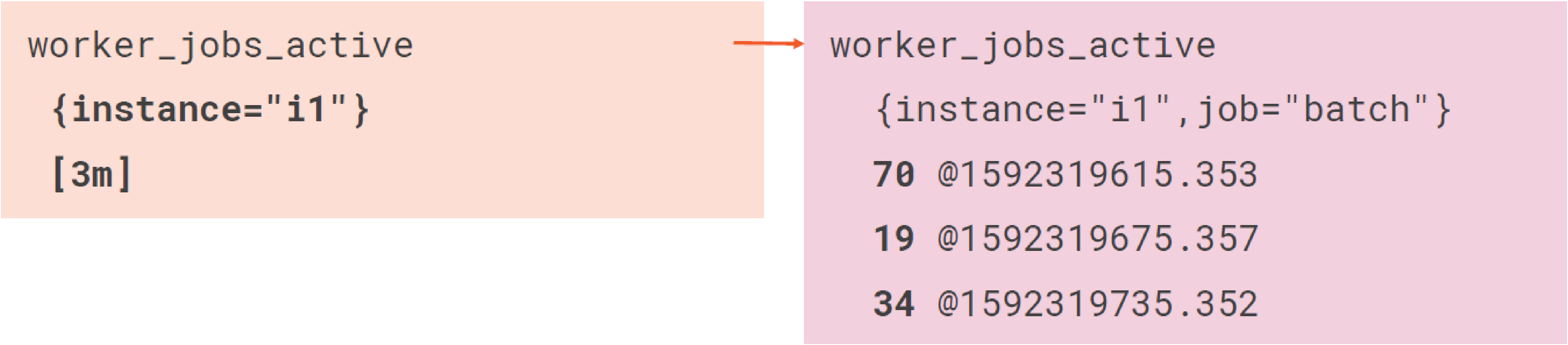
```
worker_jobs_active
 {job="batch",instance=~"i.*"}
```

```
worker_jobs_active
  {instance="i1",job="batch"} 84

worker_jobs_active
  {instance="i2",job="batch"} 51
```



```
worker_jobs_active[3m]
```

```
worker_jobs_active
  {instance="i1",job="batch"}
  70 @1592319615.353
  19 @1592319675.357
  34 @1592319735.352

worker_jobs_active
  {instance="i2",job="batch"}
  95 @1592319645.816
  56 @1592319705.818
  55 @1592319765.823
```



```
worker_jobs_active
 {instance="i1"}
 [3m]
```

```
worker_jobs_active
  {instance="i1",job="batch"}
  70 @1592319615.353
  19 @1592319675.357
  34 @1592319735.352
```

```
sum(worker_jobs_active)
```

```
135
```

```
sum without(job)
(worker_jobs_active)
```

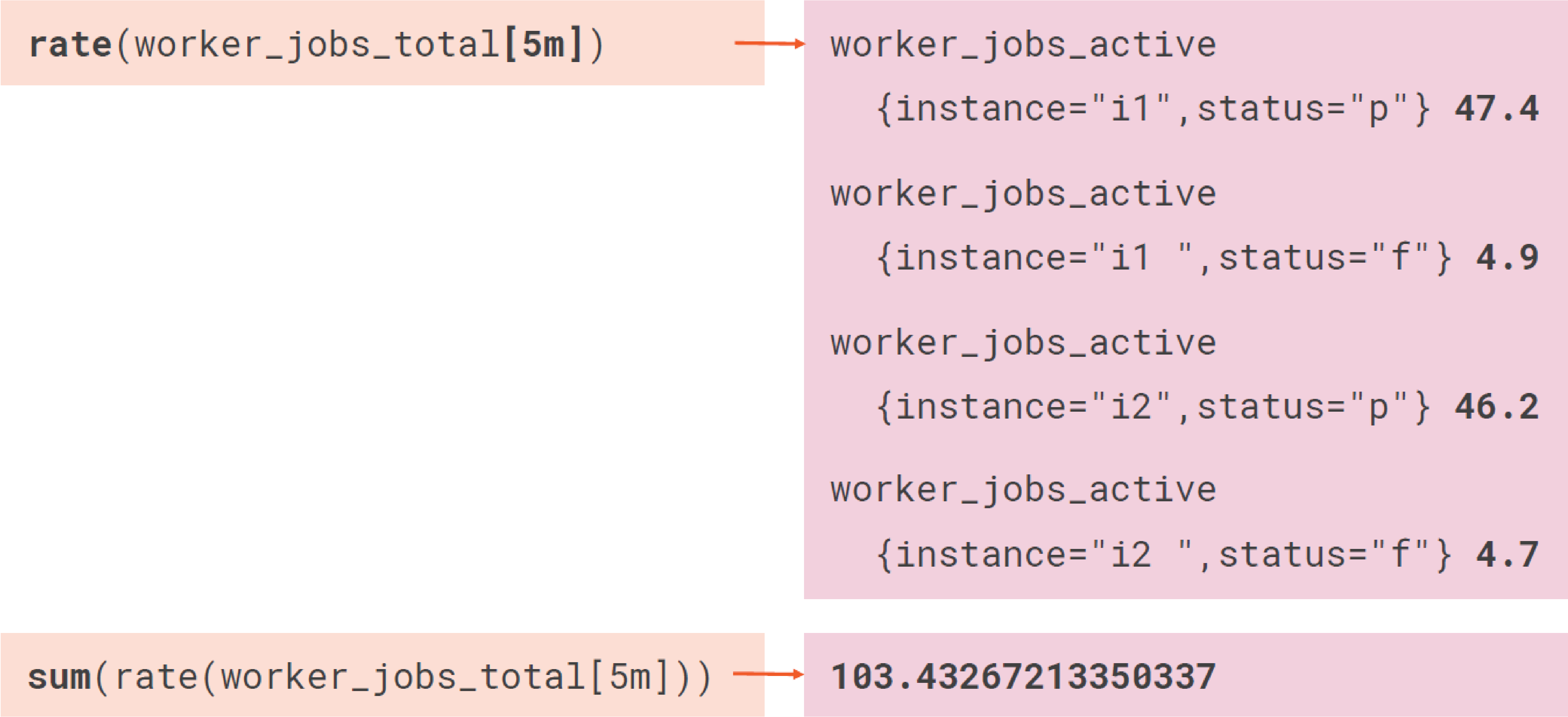
```
worker_jobs_active
  {instance="i1"} 80
worker_jobs_active
  {instance="i2"} 55
```

```
delta(worker_jobs_active[1h])
```

```
worker_jobs_active
  {instance="i1"} 18.305058891159

worker_jobs_active
  {instance="i2"} 8.1355748348591
```

```
avg(delta(worker_jobs_active[1h]))
```

```
13.220316863009444
```

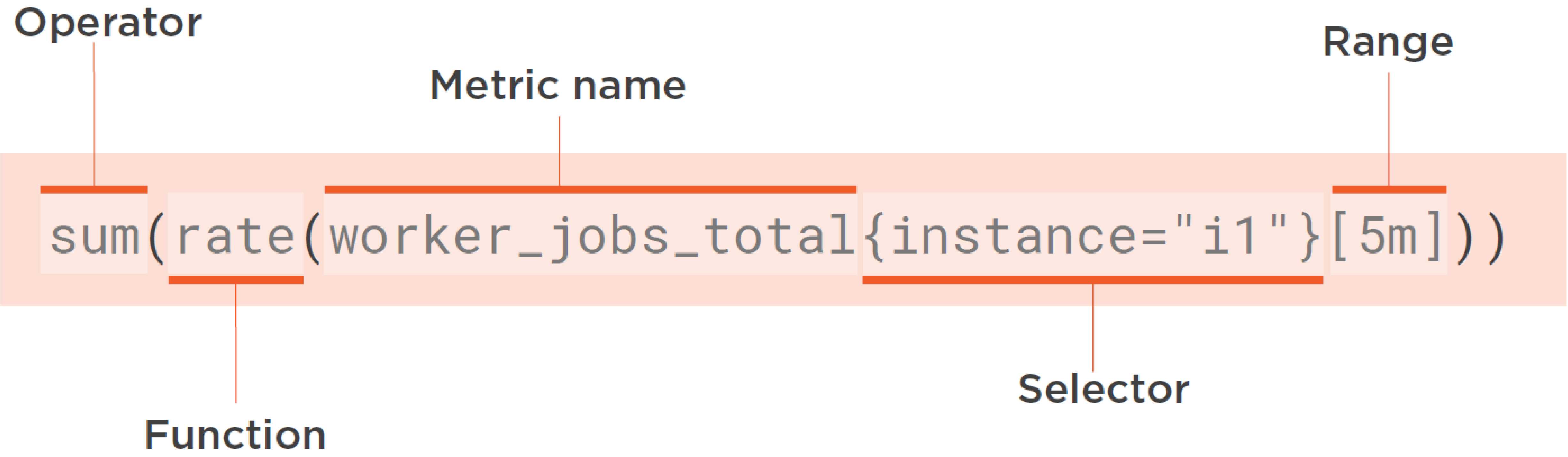


```
rate(worker_jobs_total[5m])
```

```
worker_jobs_active
  {instance="i1",status="p"} 47.4

worker_jobs_active
  {instance="i1 ",status="f"} 4.9

worker_jobs_active
  {instance="i2",status="p"} 46.2

worker_jobs_active
  {instance="i2 ",status="f"} 4.7
```

```
sum(rate(worker_jobs_total[5m]))
```

```
103.4326721350337
```



Operator

Metric name

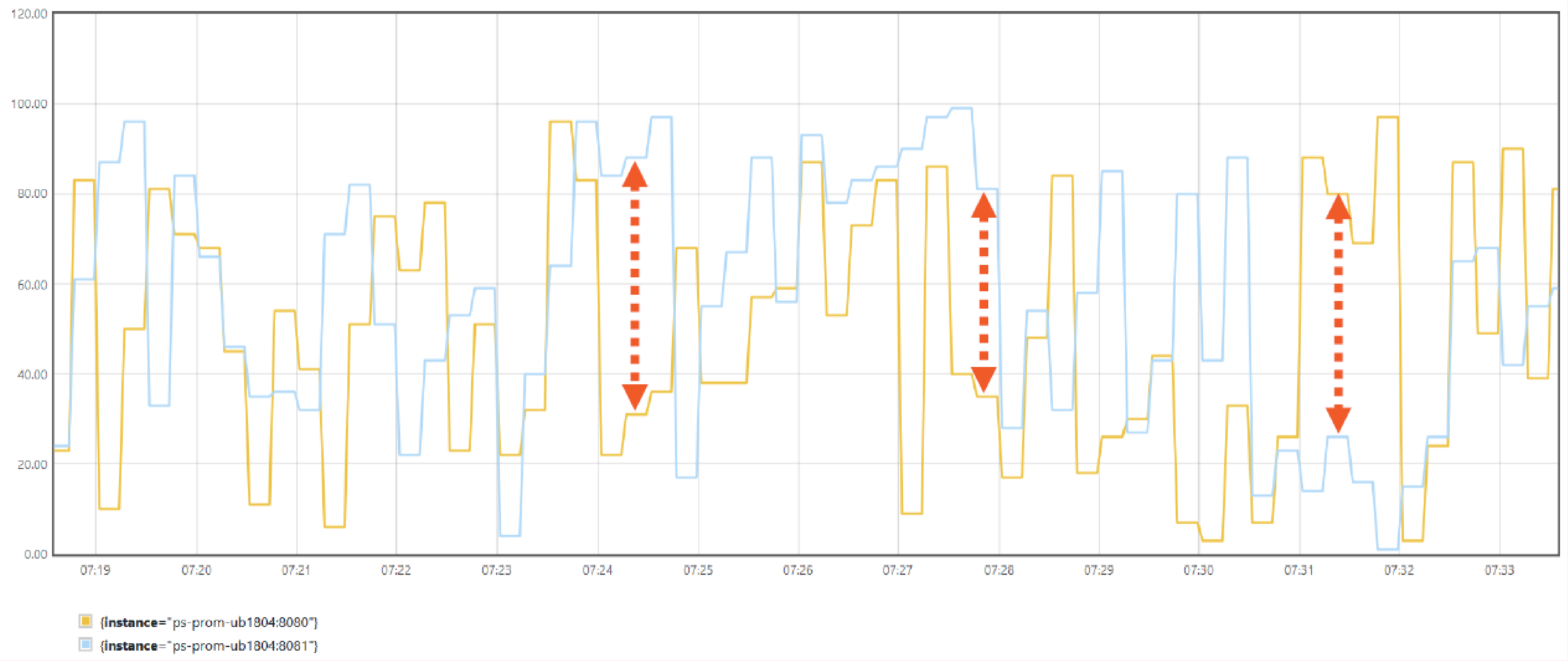
Range

```
sum(rate(worker_jobs_total{instance="i1"}[5m]))
```

Function

Selector

sum without(job, os, runtime) (worker_jobs_active)



Summary

- Prometheus architecture
- Monitoring systems
- Metric format
- Running Prometheus
- Configuring Prometheus
- Target labels
- PromQL
- Typical expressions

Q&A

Thank you