Prometheus
Best Practices and Beastly Pitfalls

Julius Volz, April 20, 2018

Areas

- Instrumentation
- Alerting
- Querying
- Monitoring Topology

Instrumentation

What to Instrument

- "USE Method" (for resources like queues, CPUs, disks...)
  Utilization, **S**aturation, **E**rrors
  http://www.brendangregg.com/usemethod.html

- "RED Method" (for request-handling services)
  Request rate, **E**rror rate, **D**uration
  https://www.slideshare.net/weaveworks/monitoring-microservices

- Spread metrics liberally (like log lines)
- Instrument every component (including libraries)

Metric and Label Naming

- Prometheus server does not enforce typing and units
- BUT! **Conventions:**
  - Unit suffixes
  - Base units (`_seconds` vs. `_milliseconds`)
  - `_total` counter suffixes
  - either `sum()` or `avg()` over metric should make sense
  - See https://prometheus.io/docs/practices/naming/

Label Cardinality

- Every unique label set: one series
- Unbounded label values will blow up Prometheus:
  - public IP addresses
  - user IDs
  - SoundCloud track IDs (*ehem*)

Label Cardinality

- Keep label values well-bounded
- Cardinalities are multiplicative
- What ultimately matters:
  - **Ingestion:** total of a couple million series
  - **Queries:** limit to 100s or 1000s of series
- Choose metrics, labels, and #targets accordingly

Errors, Successes, and Totals

Consider two counters:

- failures_total
- successes_total

What do you actually want to do with them?
Often: **error rate ratios**!

Now complicated:

```
  rate(failures_total[5m])
/
  (rate(successes_total[5m]) + rate(failures_total[5m]))
```

Errors, Successes, and Totals

- Track **failures and total requests**, not **failures and successes**.

  - failures_total
  - requests_total

Ratios are now simpler:

```
rate(failures_total[5m]) / rate(requests_total[5m])
```

Missing Series

Consider a labeled metric:

```
ops_total{optype="<type>"}
```

Series for a given "type" will only appear once something happens for it.

Missing Series

Query trouble:

- `sum(rate(ops_total[5m]))`
  - ⇨ empty result when **no** op has happened yet

- `sum(rate(ops_total{optype="create"}[5m]))`
  - ⇨ empty result when no "`create`" op has happened yet

Can break alerts and dashboards!

Missing Series

If feasible:

Initialize known label values to 0. In Go:

```
for _, val := range opLabelValues {
    // Note: No ".Inc()" at the end.
    ops.WithLabelValues(val)
}
```

Client libs automatically initialize label-less metrics to 0.

Missing Series

Initializing not always feasible. Consider:

```
http_requests_total{status="<status>"}
```

A `status="500"` filter will break if no `500` has occurred.

Either:

- Be aware of this
- Add missing label sets via `or` based on metric that exists (like `up`):

  ```
  <expression> or up{job="myjob"} * 0
  ```

  See https://www.robustperception.io/existential-issues-with-metrics/

Metric Normalization

- Avoid non-identifying extra-info labels
  Example:
  ```
  cpu_seconds_used_total{role="db-server"}
  disk_usage_bytes{role="db-server"}
  ```
- Breaks series continuity when role changes
- Instead, join in extra info from separate metric:
  https://www.robustperception.io/how-to-have-labels-for-machine-roles/

Alerting

General Alerting Guidelines

Rob Ewaschuk's "My Philosophy on Alerting" (Google it)

Some points:

- Page on user-visible symptoms, not on causes
  - ...and on immediate risks ("disk full in 4h")
- Err on the side of fewer pages
- Use causal metrics to answer **why** something is broken

Unhealthy or Missing Targets

Consider:

alert: HighErrorRate
expr: rate(errors_total{job="myjob"}[5m]) > 10
for: 5m

Congrats, amazing alert!

But what if **your targets are down** or **absent in SD**?

⇨ empty expression result, no alert!

Unhealthy or Missing Targets

- Always have an up-ness and presence alert per job:

```
# (Or alert on up ratio or minimum up count).
alert: MyJobInstanceDown
expr: up{job="myjob"} == 0
for: 5m

alert: MyJobAbsent
expr: absent(up{job="myjob"})
for: 5m
```

"for" Duration

Don't make it too short or missing!

alert: InstanceDown
expr: up == 0

Single failed scrape causes alert!

"for" Duration

Don't make it too short or missing!

alert: InstanceDown
expr: up == 0
for: 5m

"for" Duration

Don't make it too short or missing!

alert: MyJobMissing
expr: absent(up{job="myjob"})

Fresh (or long down) server may immediately alert!

"for" Duration

Don't make it too short or missing!

alert: MyJobMissing
expr: absent(up{job="myjob"})
for: 5m

"for" Duration

- Make this at least 5m (usually)

"for" Duration

Don't make it too long!

alert: InstanceDown
expr: up == 0
for: 1d

No **for** persistence across restarts! (#422)

"for" Duration

- Make this at most 1h (usually)

Preserve Common / Useful Labels

Don't:

alert: HighErrorRate
expr: sum(rate(...)) > x

Do (at least):

alert: HighErrorRate
expr: sum by(job) (rate(...)) > x

Useful for later routing/silencing/...

Querying

Scope Selectors to Jobs

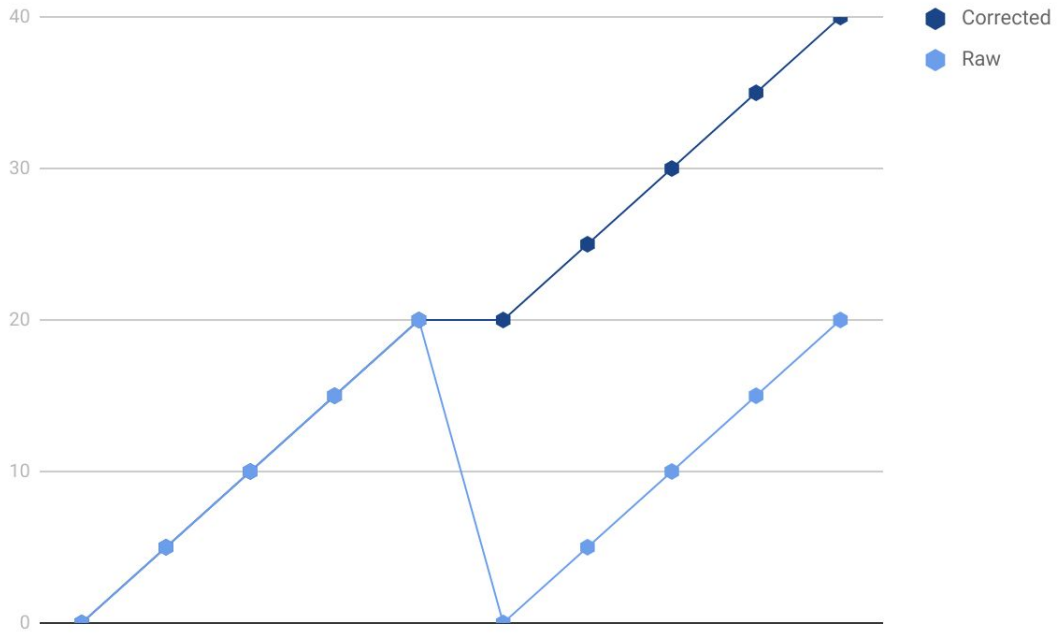
- Metric name has single meaning only within one binary (job).
- Guard against metric name collisions between jobs.
- ⇨ Scope metric selectors to jobs (or equivalent):

Don't: rate(http_request_errors_total[5m])

Do: rate(http_request_errors_total**{job="api"}**[5m])

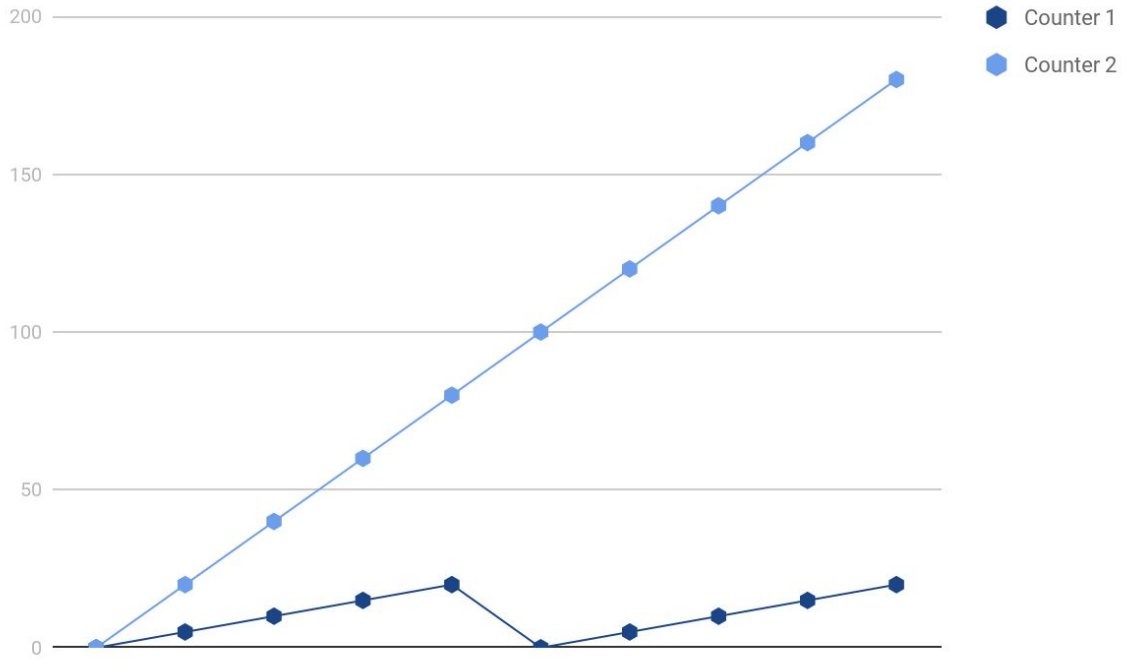
Order of rate() and sum()

Counters can reset. `rate()` corrects for this:



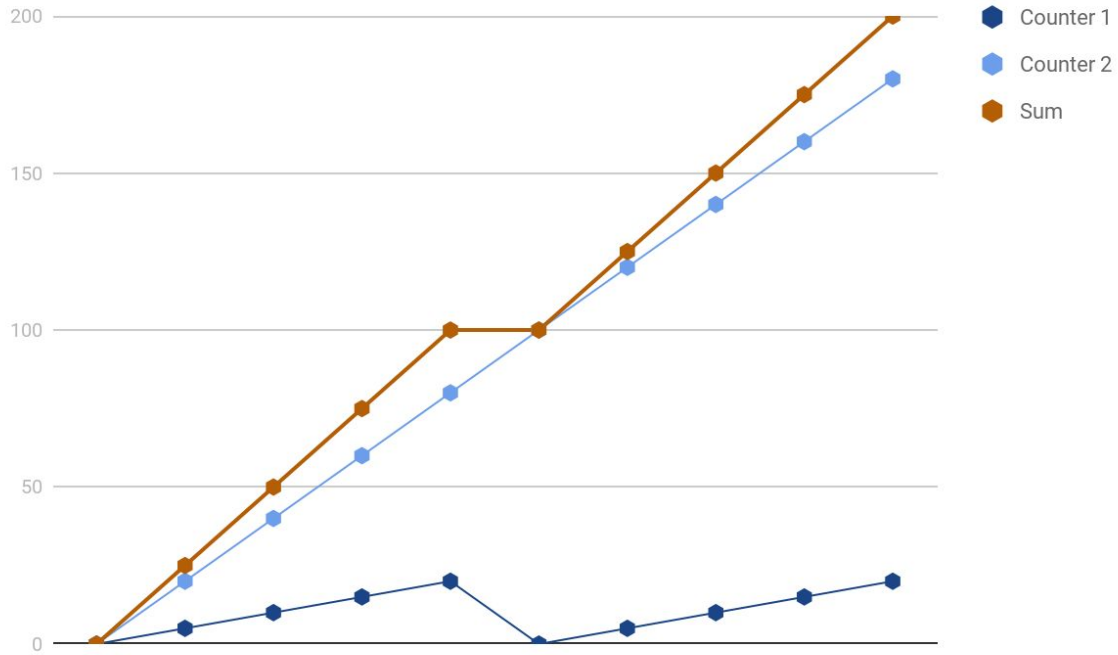
Order of rate() and sum()

`sum()` before `rate()` masks resets!



Order of rate() and sum()

`sum()` before `rate()` masks resets!



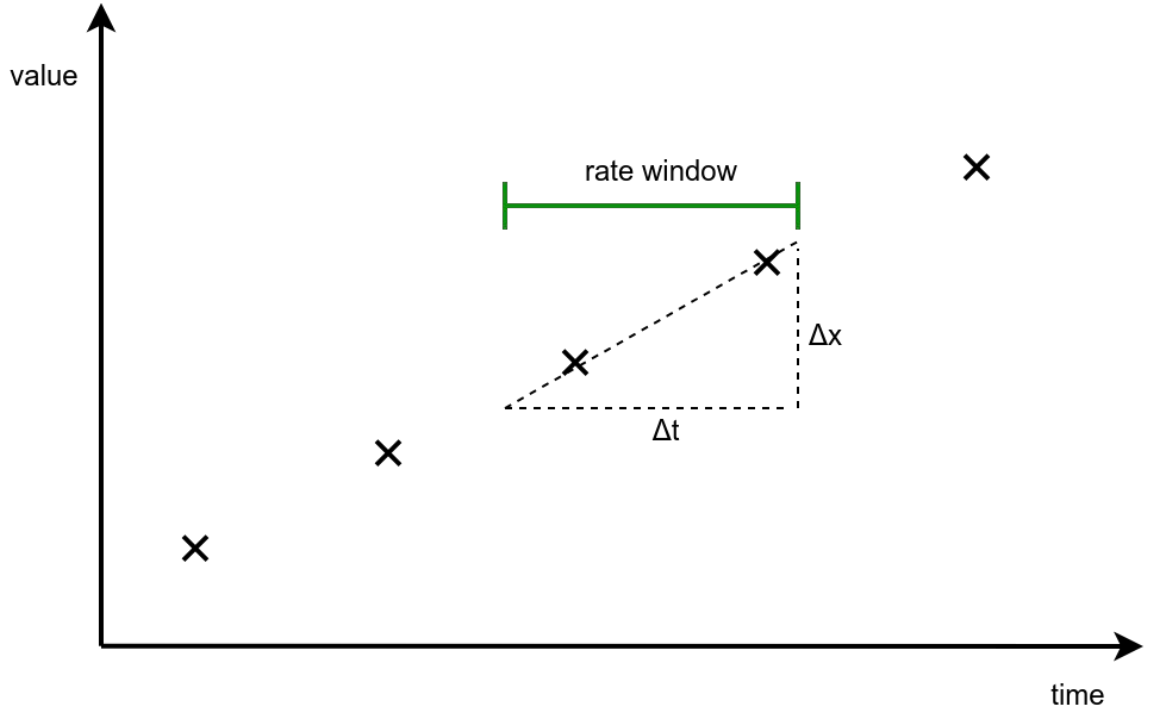
Order of rate() and sum()

⇨ Take the sum of the rates, not the rate of the sums!

(PromQL makes it hard to get wrong.)

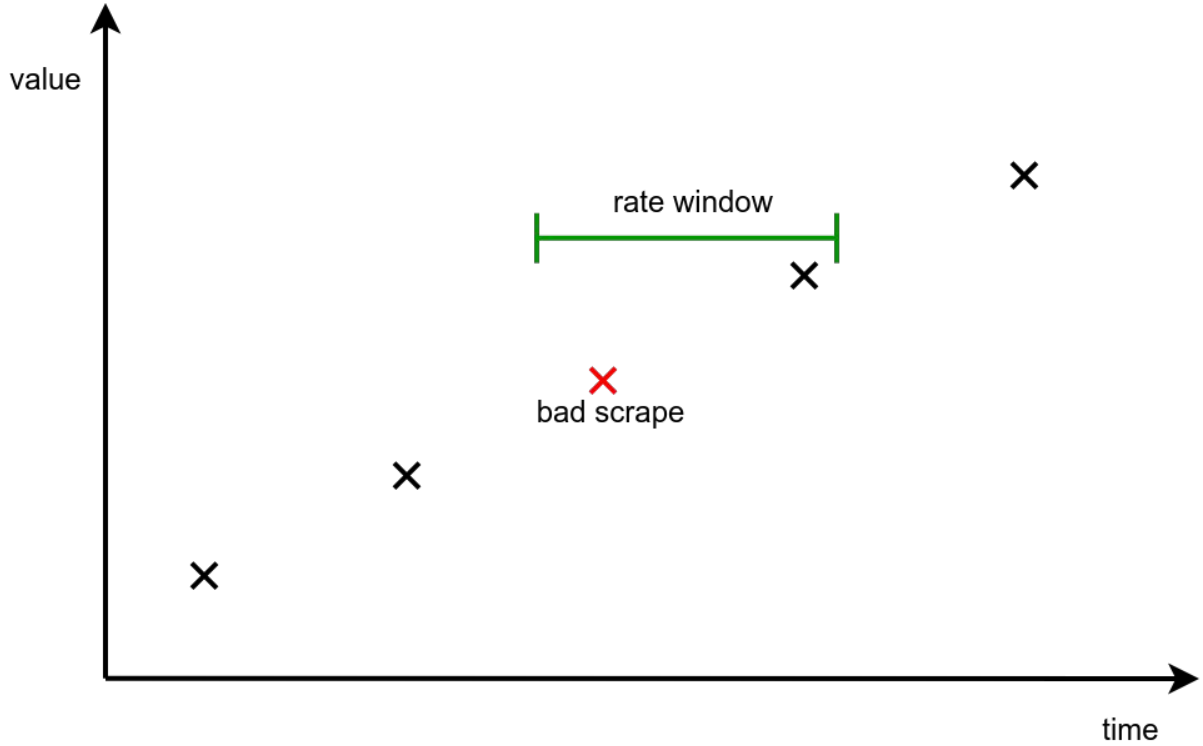
rate() Time Windows

`rate()` needs at least two points under window:



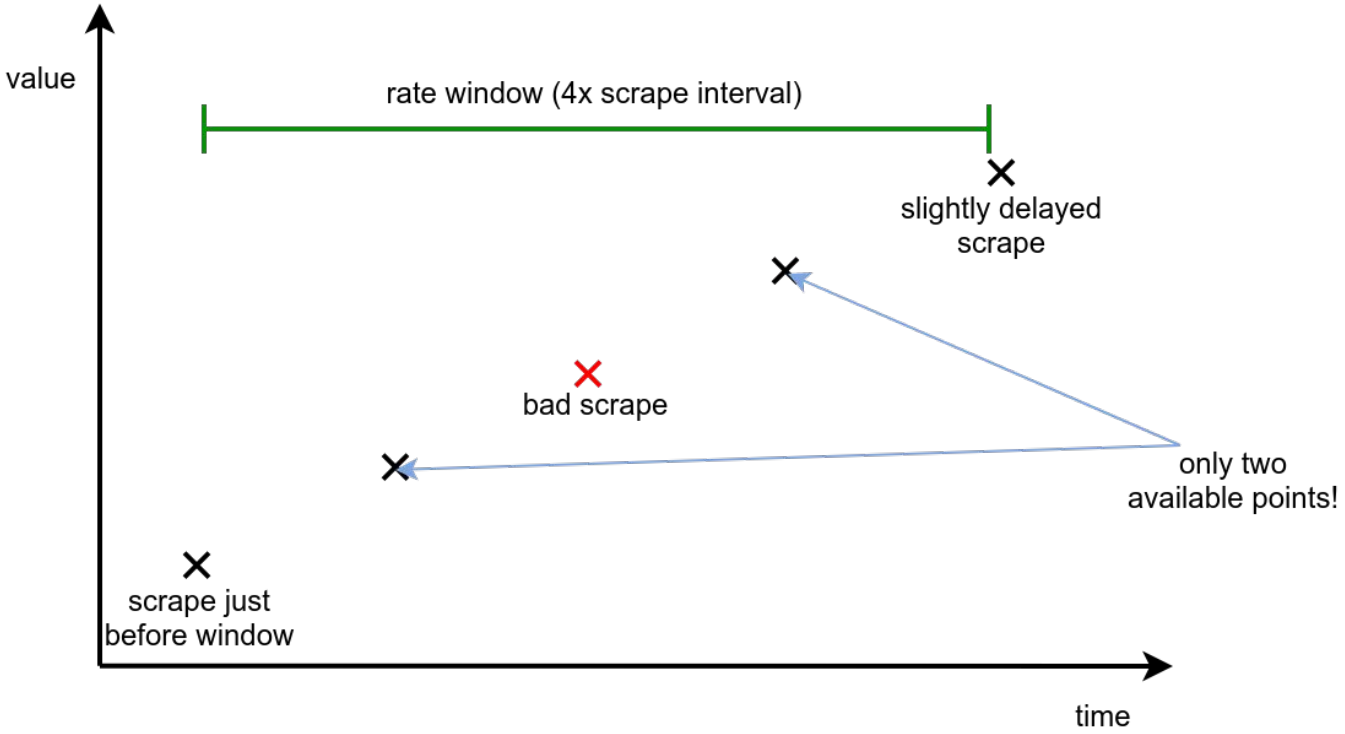
rate() Time Windows

failed scrape + short window = empty `rate()` result:



rate() Time Windows

Also: window alignment issues, delayed scrapes



rate() Time Windows

- To be robust, use a rate() window of at least 4x the scrape interval!

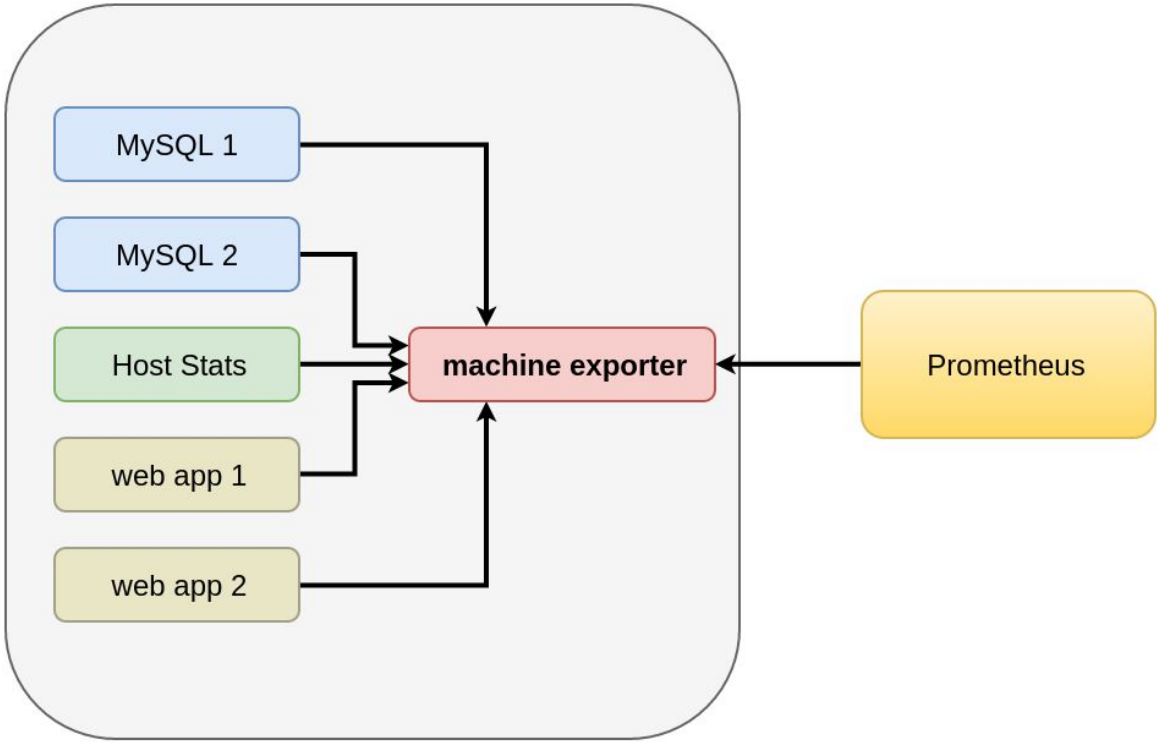
Monitoring Topology

Uber-Exporters

or...

Per-Process Exporters?

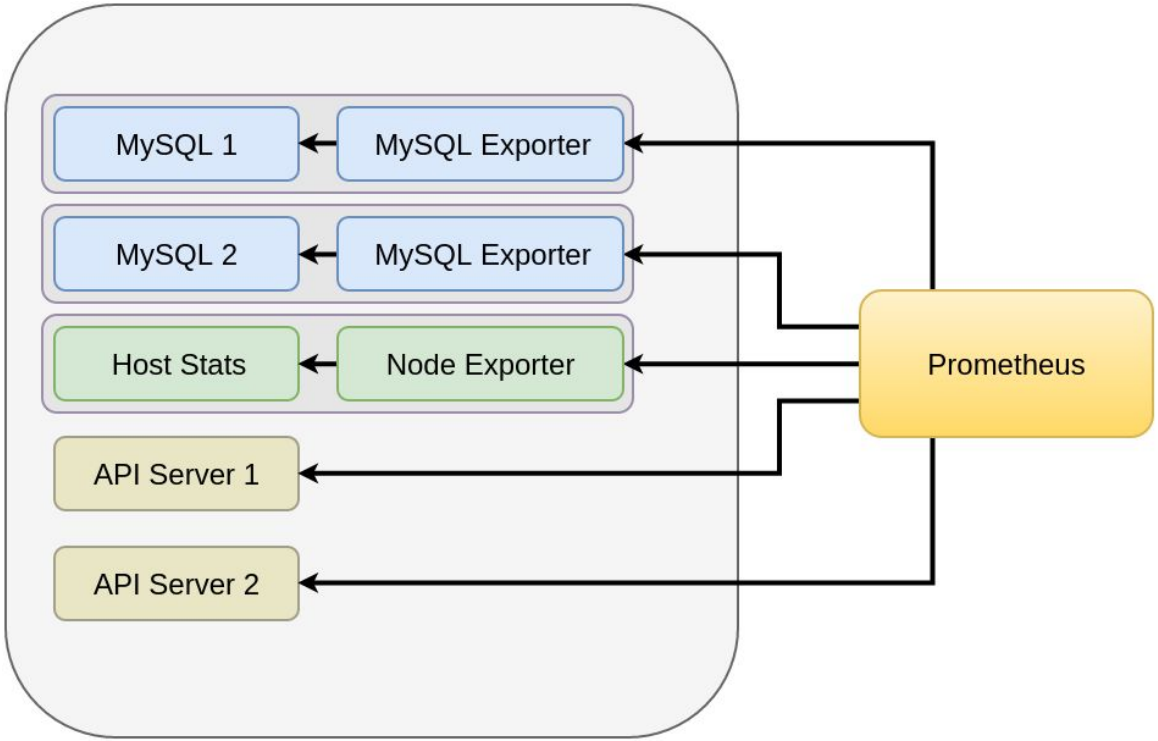
Per-Machine Uber-Exporters



BAD:

- operational bottleneck
- SPOF, no isolation
- can't scrape selectively
- harder up-ness monitoring
- harder to associate metadata

One Exporter per Process



BETTER!

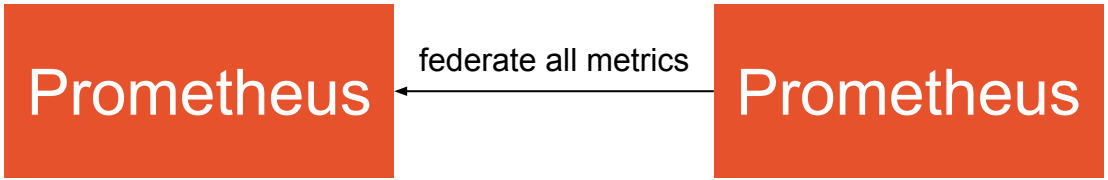
- no bottleneck
- isolation between apps
- allows selective scraping
- integrated up-ness monitoring
- automatic metadata association

Similar Problem: Abusing the Pushgateway

See https://prometheus.io/docs/practices/pushing/

Abusing Federation

Don't use federation to fully sync one Prometheus server into another: inefficient and pointless (scrape targets directly instead).



Use federation for:

- Pulling selected metrics from other team's Prometheus
- Hierarchical federation for scaling. See: https://www.robustperception.io/scaling-and-federating-prometheus/

Thanks!