GARNET | Ghanaian Academic and Research Network

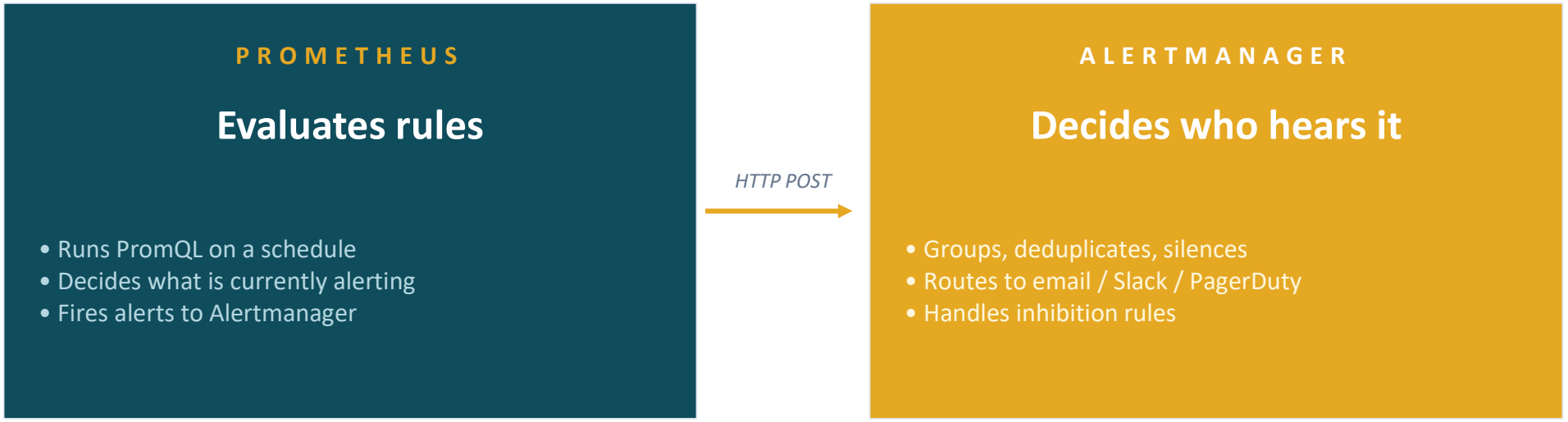
DAY 3 · SESSION 2

Alerting with Prometheus & Alertmanager

Rules, routing, and waking up the right person

ARCHITECTURE

Alerting is a two-stage pipeline



Rules belong to Prometheus. Routing belongs to Alertmanager. They are different jobs.

RULES

Anatomy of an alert rule

```yaml
groups:
  - name: network
    rules:
      - alert: InterfaceDown
        expr: ifOperStatus == 2
        for: 5m
        labels:
          severity: warning
          team: noc
        annotations:
          summary: "Interface {{ $labels.ifName }} on {{ $labels.instance }} is down"
          description: "Has been down for 5+ minutes."
```

Rule = expr (when) · for (how long) · labels (routing hints) · annotations (human text)

STATES

The lifecycle of one alert

Why 'for' matters

Without it, every momentary blip pages someone. With it, transient noise is filtered out — only sustained problems fire.

DESIGN PRINCIPLES

Alert discipline: what makes a good alert

✓	**Symptoms, not causes**	*✗ CPU > 80%*	*High CPU isn't a problem if users are happy.*
✓	**Actionable**	*✗ FYI: disk getting full*	*If no one acts, why send it?*
✓	**Has a clear runbook**	*✗ InterfaceDown — figure it out*	*Pages without runbooks waste on-call time.*
✓	**Right severity / right channel**	*✗ Everything to email*	*Mixing critical and informational dulls response.*

ROUTING

Alertmanager routing tree

Alerts walk down a tree of routes. The first match wins by default — or routes can be 'continue: true' for fan-out.

```
route:
  receiver: default-email
  group_by: [alertname, instance]
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  routes:
    - matchers: [severity = critical]
      receiver: oncall-pagerduty
    - matchers: [team = noc]
      receiver: noc-slack
```

FEATURES

Three Alertmanager features that save your sanity

Grouping

Bundle related alerts into one notification — '40 interfaces down on switch X' instead of 40 emails.

Silencing

Mute alerts during maintenance windows or known issues. Set via UI or API. Time-bound.

Inhibition

When alert A fires, suppress alert B. 'Switch down' inhibits 'all interfaces down' on that switch.

RECEIVERS

Where alerts actually go

Email	Default fallback. Reliable but easy to ignore at scale.
Slack / Teams	Best for team channels. Threaded, browseable, low-friction.
PagerDuty / Opsgenie	When you need someone awake. Escalation policies built-in.
Webhook	Integrate with anything: ticketing, custom bots, internal tools.
SMS	Last-resort criticals. Many providers via webhook → SMS gateway.

Match channel to severity. Don't send daily-digest informational alerts to phone-buzzing pagers.

BONUS

Recording rules: pre-compute expensive queries

A recording rule runs a PromQL expression on a schedule and saves the result as a new time series.

```
groups:
  - name: precomputed
    interval: 30s
    rules:
      - record: instance:cpu_busy:rate5m
        expr: 1 - avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m]))
```

Use them for

Dashboards that re-query the same expensive expression · Alert rules built on aggregated metrics · Sharing pre-computed series across many panels

EXAMPLE

End-to-end: one alert from query to email

1	**Rule evaluates**	Every 1m: ifOperStatus == 2 returns 1 series for GigE0/1 on core-rt-1
2	**Pending starts**	Series first appears at 14:02. Alert enters PENDING state.
3	**Fires after 5m**	At 14:07 the 'for: 5m' window has elapsed. Alert FIRES.
4	**Sent to Alertmanager**	Prometheus POSTs alert with severity=warning, team=noc labels.
5	**Routing decides**	team=noc matcher → noc-slack receiver. Grouping waits 30s for siblings.
6	**Slack message**	At 14:07:30, Slack channel #noc-alerts gets the formatted notification.

ON-CALL DISCIPLINE

When the page fires: the playbook

1	**Acknowledge**	Click the ack within 5 minutes. Stops escalation. Tells the team you're on it.
2	**Assess severity**	Read the alert summary. Check the dashboard linked in the runbook. Is this user-impacting now?
3	**Communicate**	Drop a one-liner in the incident channel even if you have no answer yet. Silence is worse than uncertainty.
4	**Mitigate first**	Restore service before finding root cause. Failover, rollback, restart — buy time, then investigate.
5	**Resolve & document**	Resolve the alert in Alertmanager once stable. Capture timeline + cause for the post-mortem.

ALERTING IN PLACE

From watching to being told

Rules say what's broken · Alertmanager decides who hears · Discipline beats tooling