{mthree}

Service Level Objectives

Site Reliability Engineering

Objectives

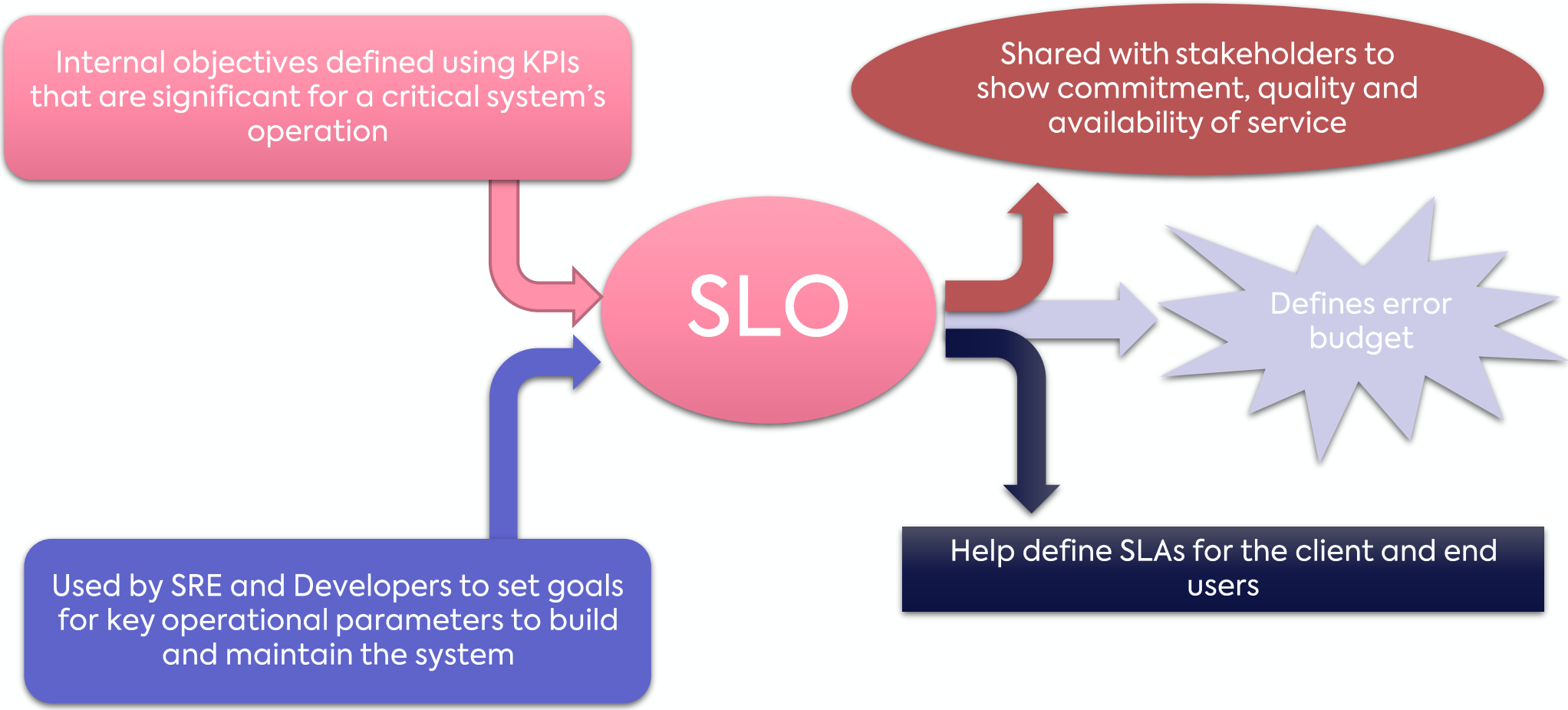
In this module, we will look at service level objectives (SLOs) and their importance in driving a reliable system and working environment.

Learning Objectives

- Define an SLO
- Identify useful SLIs
- Determine if SLO ranges are realistic and relevant
- Create calculations to determine availability
- List ways to improve SLOs to improve toil and keep users happy

{mthree}

SLO in Brief



{mthree}

SLA = Service Level Agree

Importance of SLOs

- Provide clear goals
- Set end-user expectations
- Powerful tool for ensuring the quality of product and service
- Define realistic goals
  - >>> Critical systems can and will fail
  - >>> Set acceptable levels of failure frequency and severity of errors
  - >>> This defines the error budget
- Number and frequency of errors < error budget
  - >>> This is not just operational issues
  - >>> Deployment failures also included

Error Budget

- Metric that determines how unreliable the service is allowed to be
- Set at an acceptable working frequency for the metric
  - >>> Monthly
  - >>> Quarterly
- Removes politics between teams, deciding how much risk allowed
- Calculation
  - >>> 1 - SLO = Error Budget for that service
- Building your Error Budget in detail
  - >>> Error Budgets Explained

{mthree}

Google's Error Budget Practices

 Product management defines an SLO	Sets an expectation of how much uptime for the service per period
 Actual uptime is measured by a neutral third party	Our monitoring system (SLIs)
 Difference between these two numbers is the **budget**	How much "unreliability" is remaining for the period
 As long as there is error budget remaining	New releases/changes can be pushed

{mthree}

Google's Example

A service's SLO is to successfully serve 99.999% of all queries per quarter

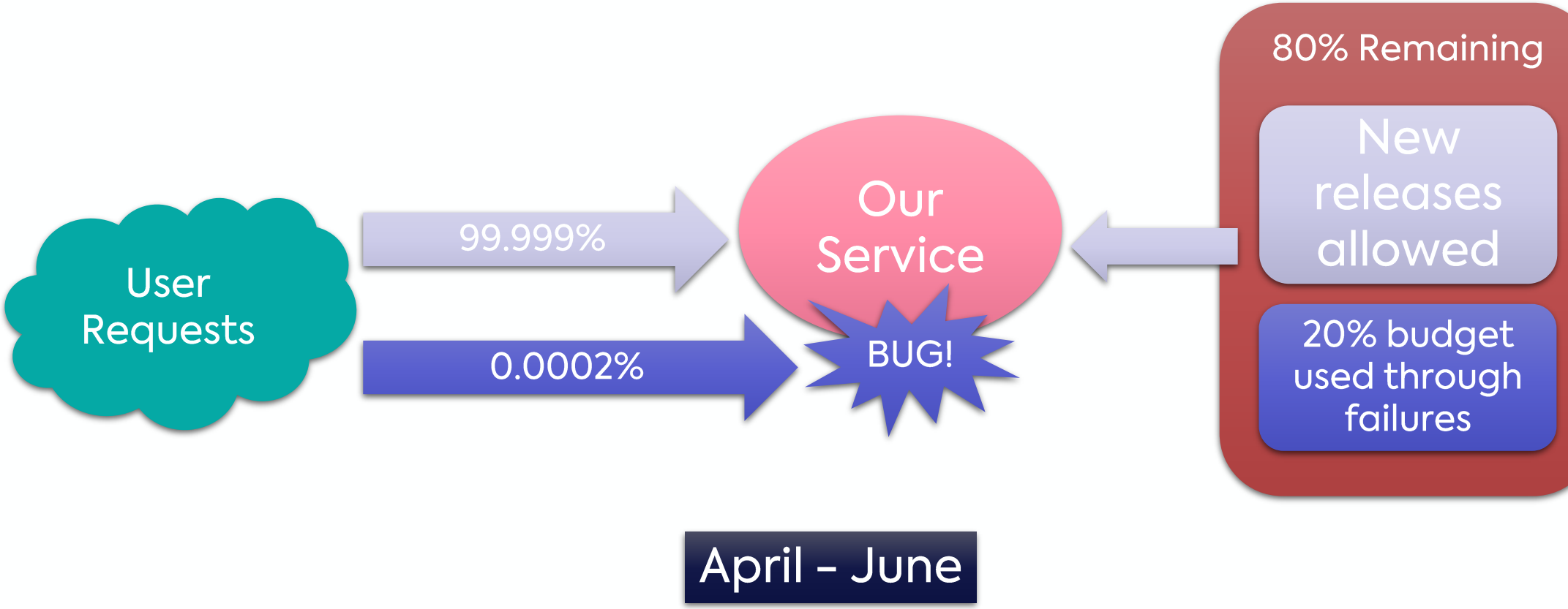
The service's error budget is a failure rate of 0.001% for a given quarter

If a problem causes us to fail 0.0002% of the expected queries per quarter

The problem spends 20% of the service's quarterly error budget

{mthree}

Google's Example: Diagram



Calculating SLOs

- We discover a server that crashes for 6 minutes every hour
  - >>> 90% availability
  - >>> Less than 1 hour of reliability!
- Calculating availability
  - >>> Good for
    - ~ Infrastructure
    - ~ Platforms
    - ~ Application components
- Calculation measured for each area
  - >>> Gives us a map of the site
- SLOs can be per silo
- SLA overall

$$Availability\ (\%) = \left(\frac{Active\ time}{Expected\ operating\ time}\right) \times 100\%$$

$$Reliability\ (MTBF) = \frac{Total\ uptime}{Number\ of\ breakdowns}$$

MTBF: Mean time between fa

{mthree}

Latency SLOs Calculations

- Where to measure the latency?
- From the user to the backend and return
  - >>> True end-to-end
  - >>> But are we responsible for the users' ISP deficiencies?
- From the point it enters our system and returns to the entry point
  - >>> Everything in here is what we can manage and control
  - >>> We should note entry times and return times to users
- Is it per user or aggregated across a number of user requests?
  - >>> Over a number of requests
- Do I react immediately or over a time period?
  - >>> Over a time period
- Systems may respond differently at different times

{mthree}

Meaningful Metrics

- Identifying the right metrics to set SLOs
  - >>> Too many metrics and calculation becomes complex
  - >>> Reduce noise – don't alert unless it affects SLO
- What metrics really matter?
  - >>> Latency, Traffic, Errors, Saturation
  - >>> Changes in components such as version numbers, options, configuration
    - ~ Historical trend comparison to see if we are improving with new features
- What metrics can we ignore that are not affecting users
  - >>> For example, a disk failure may not affect the users if we have resilience
  - >>> A long running database query might increase latency, but within budget
  - >>> The user's Internet connection
- What metrics tell us if the end- is being affected?
  - >>> Slow disk or network acces that pushes latency above expected values
  - >>> Request/Response times v our system, point of entry point of return
  - >>> Wait time for service avail (waiting for microservice t available)
- An iterative process
  - >>> Constant review and monitoring of metrics as th system evolves

{mthree}

What SLIs Do I Use?

Application metrics

What data is being served

These are internal metrics – not what the user experiences

Can help if we can measure application latency (poor SQL queries, etc)

Server logs

Internal metrics that can help identify potential issues within the syster

Front-end infrastructure metrics

Information from load balancers

Correct URL requests, latency issues, services not available

Synthetic clients or data

Fabricating client requests to get an understanding of the client view

Validating responses

Real view of end-to-end reliability

{mthree}

What SLIs Do I Use? (cont.)

Data processing pipelines

Measure latency on internal services to see how well data is transferred

Client-side instrumentation

Add features to the client that can provide metrics

For example:

- Time and time zone request was sent from their system and reached u
- IP address to determine ISP, geographic region, bandwidth

Performance and capacity testing

Generally related to infrastructure and system components

Typical metrics provide

Saturation/Availability

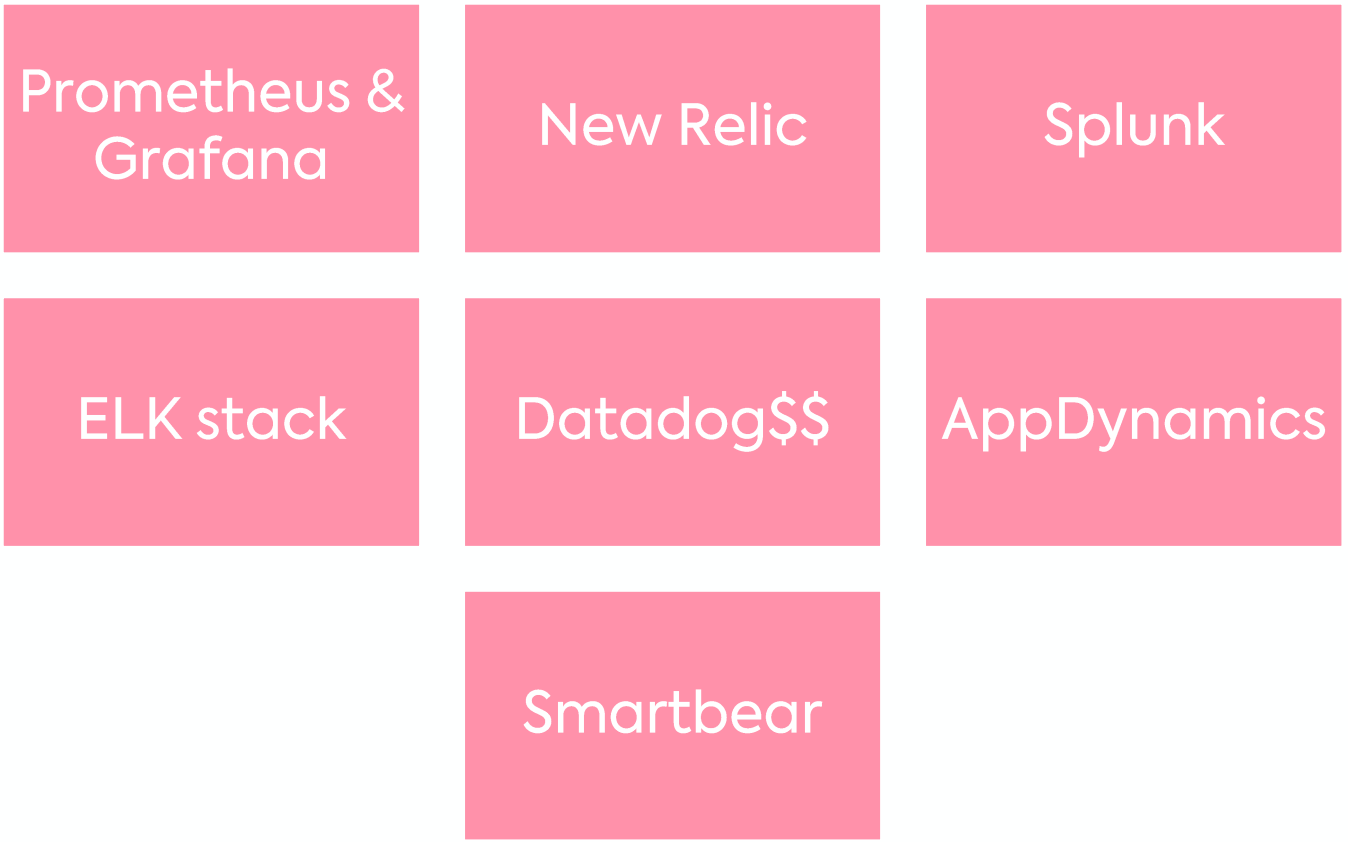
Latency

Throughput

Error rate

{mthree}

Tools for SLO



{mthree}

Improvements

- Historical data
- If we are always under our error budget, do we
  - >>> Increase our SLA, for example, from 99.999% to 99.9999%?
    - ~ Eventually we will be able to increase
  - >>> Do we keep the SLA and increase the rate of change/release?
    - ~ Fewer errors, more development time and improvement time
- A trade-off based on how your project progresses
- Discuss with the client
  - >>> Do they want more reliability or fast rate of change?

Activity: SLO in the Lab – Step-by-Step

- Here we will look at creating an effective SLO for user experience
  - >>> This will be based on the following metrics
    - ~ Time required for the API service request to be processed < 300ms
    - ~ We will aim for a 95% reliability of all buy requests
    - ~ In any 1-hour period
- We will need to have some effective SLIs
  - >>> Focus is on the application
  - >>> We could record latency in the application, the database, and the network between them
    - ~ This would introduce complexity and is better served in understanding performance
    - ~ It would be a separate internal SLO that would be introduced if we start to exceed

Activity: SLO in the Lab

- The SLIs

$$\left(\frac{count(rate(http\_server\_requests\_seconds\_bucket\{namespace = "orderbook - dev", uri = "/buy", le = \sim"0.[012].* "\}[1h]))}{count(rate(http\_server\_requests\_seconds\_bucket\{namespace = "orderbook - dev", uri = "/buy"\}[1h]))}\right) \times 1$$

- Grafana alert

```
(count(rate(http_server_requests_seconds_bucket{namespace="orderbook-
dev",uri="/buy",le=~"0.[012].*"}[1h]))/count(rate(http_server_requests_seconds_bucket{namesp
e="orderbook-dev",uri="/buy"}[1h])) * 100) < 95
```

{mthree}

Activity: Adding to Grafana

- Create a new panel in your dashboard
- Panel title: Latency SLO > 300ms
- Query = Prometheus
  - >>> Metric
- Scroll to legend and under Legend mode change to Hidden
- Scroll to Thresholds
  - >>> Change the base color to Red by clicking the circle
  - >>> Add Threshold and type 95 as the value in the box
  - >>> Click Show thresholds and select As filled regions

```
(count(rate(http_server_requests_seconds_bucket{namespac
"orderbook-
dev",uri="/buy",le=~"0.[012].*"}[1h]))/count(rate(http_s
ver_requests_seconds_bucket{namespace="orderbook-
dev",uri="/buy"}[1h])) * 100) < 95
```

- Alert
  - >>> Click the Alert tab under the graph
  - >>> Click Create alert rule from this panel
  - >>> In conditions set IS BELOW to 95
- Apply
- Save your dashboard
- This is an SLO so make it stand out on the dashboard

{mthree}

If We Exceed Our SLO?

- We will have many SLOs to identify client experience
  - >>> We may also have SLOs for toil being measured from Jira/ServiceNow tickets
- Ideally, we should take action before SLO reaches 0%
- Decisions to make
  - >>> Generally, we would stop any changes in this area for the failing SLO
    - ~ Only well-proven or low/low impact changes should be made
  - >>> Focus becomes reliability and resilience
    - ~ Reflect on what has been causing our SLOs to reduce
  - >>> Areas where SLO has not reached critical can still make changes
- Below 0% then we should ensure that we fix the current issue
  - >>> No further changes until the SLO time period resets

{mthree}

Summary Q&A

{mthree}

References

- Best practices for setting SLO/SLIs
  - >>> SLOs and SLIs best practices for systems
  - >>> Putting customers first with SLIs and SLOs
  - >>> Setting SLOs: a step-by-step guide
  - >>> Ways of measuring SLIs
  - >>> Service level objectives 101: Establishing effective SLOs
- How available
  - >>> SLOs, SLIs, SLAs, oh my—CRE life lessons
- Why we need SLOs
  - >>> Implementing SLOs
  - >>> Service Level Objectives
  - >>> A Guide To Service Level Objectives, Part 1: SLOs & You
  - >>> Managing reliability with SLOs and Error Budgets
  - >>> Site Reliability: How to Improve the Quality and Reliability of Services Using SLI / SLO / SLA
  - >>> Understanding Service Level Objectives

{mthree}