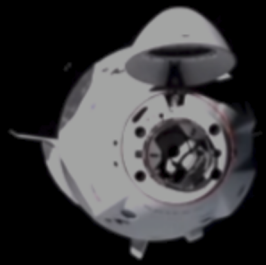
The automation engineers guide to SRE



The Quest

To better understand the concepts, principles and the relationship between DevOps, SRE and different testing approaches

And how as automation engineers we can leverage these to our advantage

Agenda

DevOps & SRE
- The 10,000ft view

Performance
- Shift left and moving right

Chaos
- Cultural change to when things fail

Automated checks
- Using real world examples

Security
- Surfacing potential vulnerabilities

Observability/Metrics
- Understanding behaviour

Testing approaches

DevOps ==
Development + Operations

DevOps is a set of practices that combines software development and IT operations

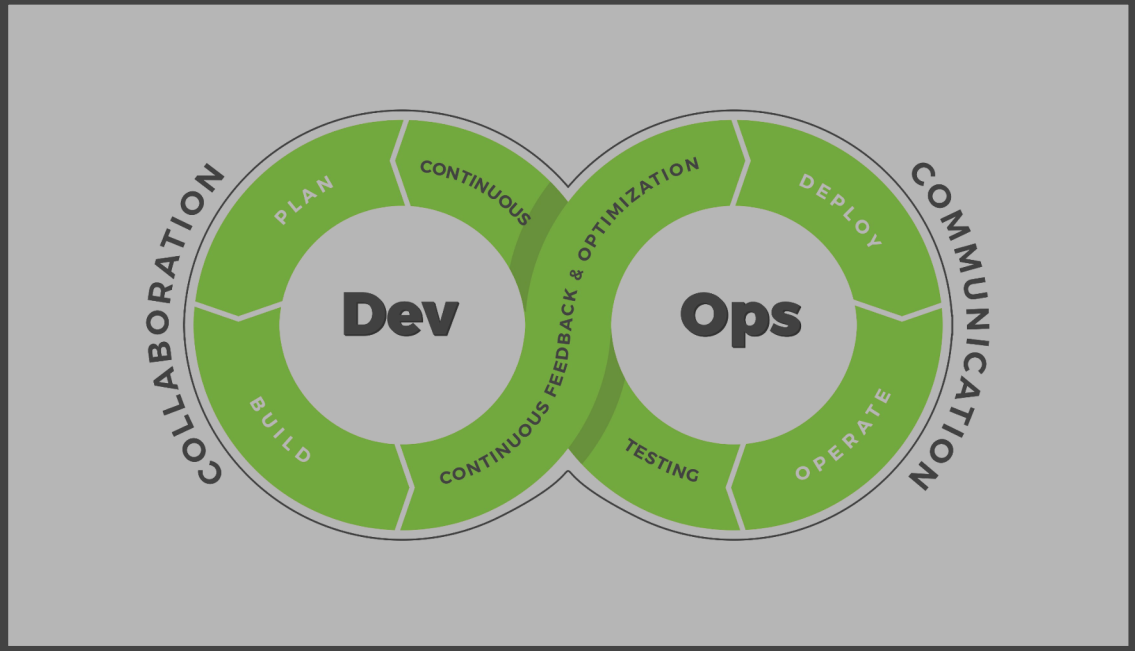
DevOps in a nutshell

DevOps is about meeting software users ever-increasing demand for frequent, innovative new features combined with uninterrupted performance and availability

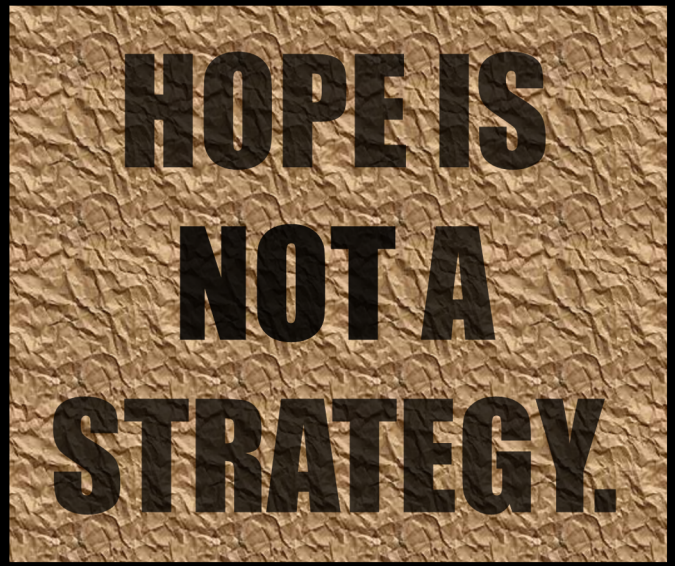
It's about:

- Continuous communication & collaboration
- Shared responsibility
- A commitment to automation
- Better work management
- Reduced lead time
- Deploying more frequently

DevOps


COLLABORATION
PLAN
CONTINUOUS
BUILD
CONTINUOUS FEEDBACK & OPTIMIZATION
Dev
Ops
DEPLOY
TESTING
OPERATE
COMMUNICATION

SRE
(site reliability engineering)

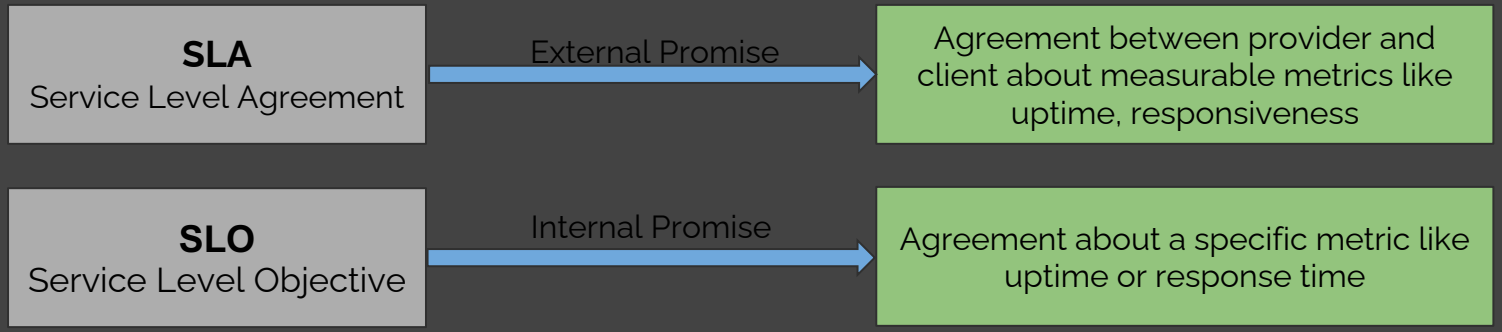


SRE is about

SRE is the practice of applying software engineering principles to operations and infrastructure processes to help organizations create highly reliable and scalable software systems

- Embracing risk
- Blameless culture
- Remove TOIL
- Apply automation where possible
- Define the problem space (SLO/SLI/SLA's)

Vocab



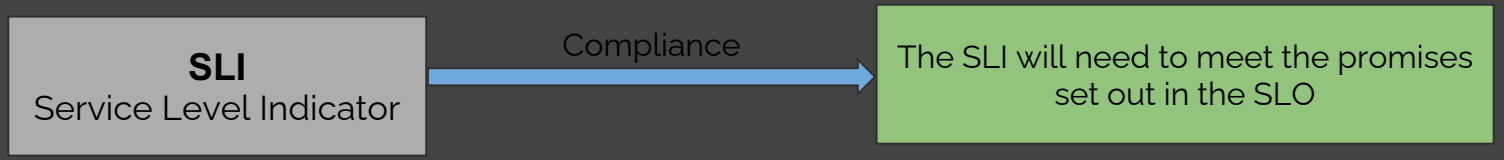
Error budget

- How much headroom is allowed

TOIL

- Operational work that can be automated

Measurement/Awareness



User interfacing SLIs

- Availability
- Latency
- Quality

Data processing SLIs

- Throughput
- Data freshness
- Data correctness
- Coverage

Responsibility

SRE teams are responsible for the availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning of their services.

Consider things like

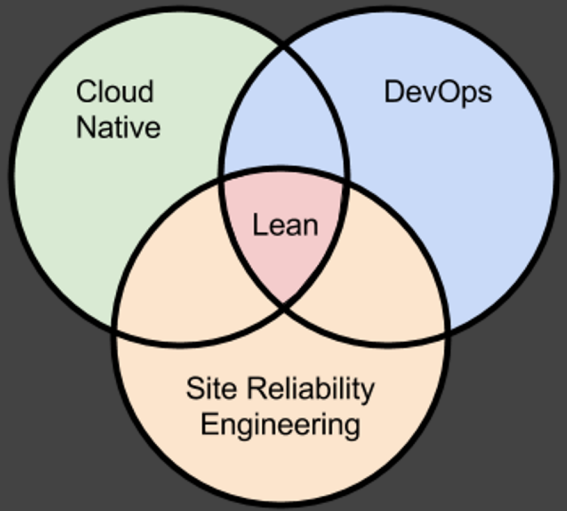
- Morale
- employee experience,
- human wellness
- organization structure
- tool stack
- hardware and software

SRE + DevOps

Principles combined with idealisms that are backed up with measurement

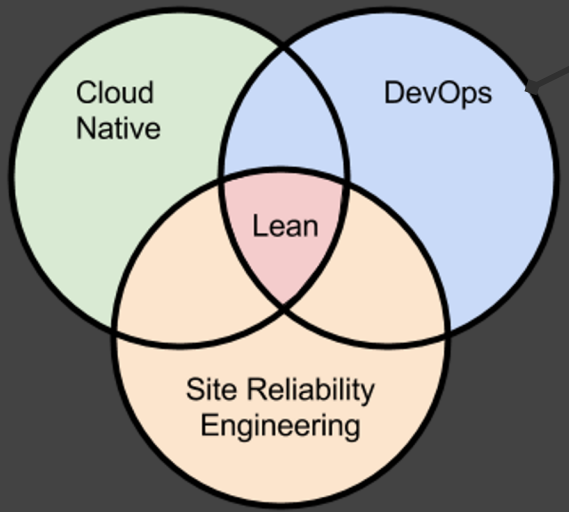
Combining forces

Both SRE and DevOps work to bridge the gap between development and operations teams to deliver services faster.

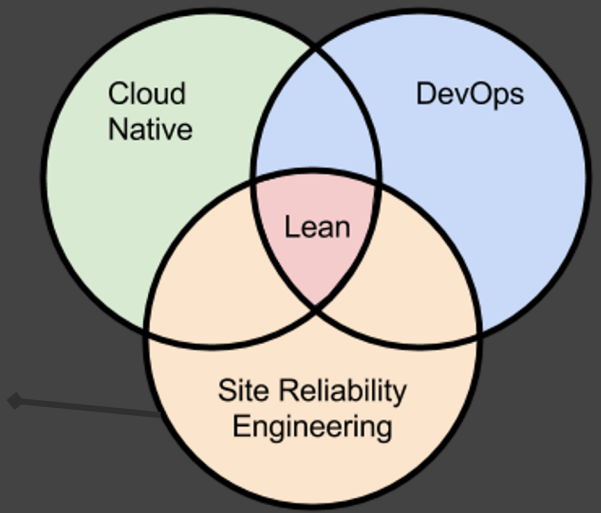


DevOps

- Continuous and rapid development
- Promotes Open communication
- Ability to make mistake
- Shared Ops responsibilities



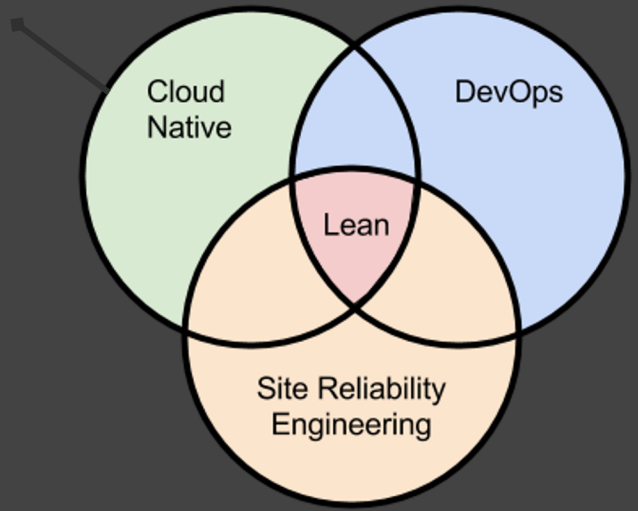
SRE



- Embracing risk
- Eliminating toil
- Balancing velocity and reliability
- Reducing operational tasks
- Managing operational risk
- DevOps bridge

Cloud Native

- It's about services not servers
- SaaS, PaaS, IaaS
- Savings, Scalability & Availability
- Containerisation, Automation and orchestration



DevOps Pillars

DevOps

- Implement gradual change
- Accept failure as normal
- Reduce organization silos
- Leverage tooling & automation
- Measure everything

DevOps / SRE relationship

DevOps	SRE
Implement gradual change	Encourage moving quickly by reducing costs of failure
Accept failure as normal	Have a formula for balancing accidents and failures against new releases
Reduce organization silos	Shared ownership with developers by using the same tools and techniques across the stack
Leverage tooling & automation	Encourages “Automating all the things”, minimises manual systems work to focus on efforts that bring long-term system value
Measure everything	Believes that operations is a software problem, and defines prescriptive ways for measuring availability, uptime, outages, toil

DevOps / SRE relationship

2

DevOps	SRE
Implement gradual change	Encourage moving quickly by reducing costs of failure
Accept failure as normal	Have a formula for balancing accidents and failures against new releases
Reduce organization silos	Shared ownership with developers by using the same tools and techniques across the stack
Leverage tooling & automation	Encourages “Automating all the things”, minimises manual systems work to focus on efforts that bring long-term system value
Measure everything	Believes that operations is a software problem, and defines prescriptive ways for measuring availability, uptime, outages, toil

DevOps / SRE relationship

DevOps	SRE
Implement gradual change	Encourage moving quickly by reducing costs of failure
Accept failure as normal	Have a formula for balancing accidents and failures against new releases
Reduce organization silos	Shared ownership with developers by using the same tools and techniques across the stack
Leverage tooling & automation	Encourages “Automating all the things”, minimises manual systems work to focus on efforts that bring long-term system value
Measure everything	Believes that operations is a software problem, and defines prescriptive ways for measuring availability, uptime, outages, toil

3

DevOps / SRE relationship

DevOps	SRE
Implement gradual change	Encourage moving quickly by reducing costs of failure
Accept failure as normal	Have a formula for balancing accidents and failures against new releases
Reduce organization silos	Shared ownership with developers by using the same tools and techniques across the stack
Leverage tooling & automation	Encourages “Automating all the things”, minimises manual systems work to focus on efforts that bring long-term system value
Measure everything	Believes that operations is a software problem, and defines prescriptive ways for measuring availability, uptime, outages, toil

4

DevOps / SRE relationship

DevOps	SRE
Implement gradual change	Encourage moving quickly by reducing costs of failure
Accept failure as normal	Have a formula for balancing accidents and failures against new releases
Reduce organization silos	Shared ownership with developers by using the same tools and techniques across the stack
Leverage tooling & automation	Encourages “Automating all the things”, minimises manual systems work to focus on efforts that bring long-term system value
Measure everything	Believes that operations is a software problem, and defines prescriptive ways for measuring availability, uptime, outages, toil

Relationships
SRE and Testing

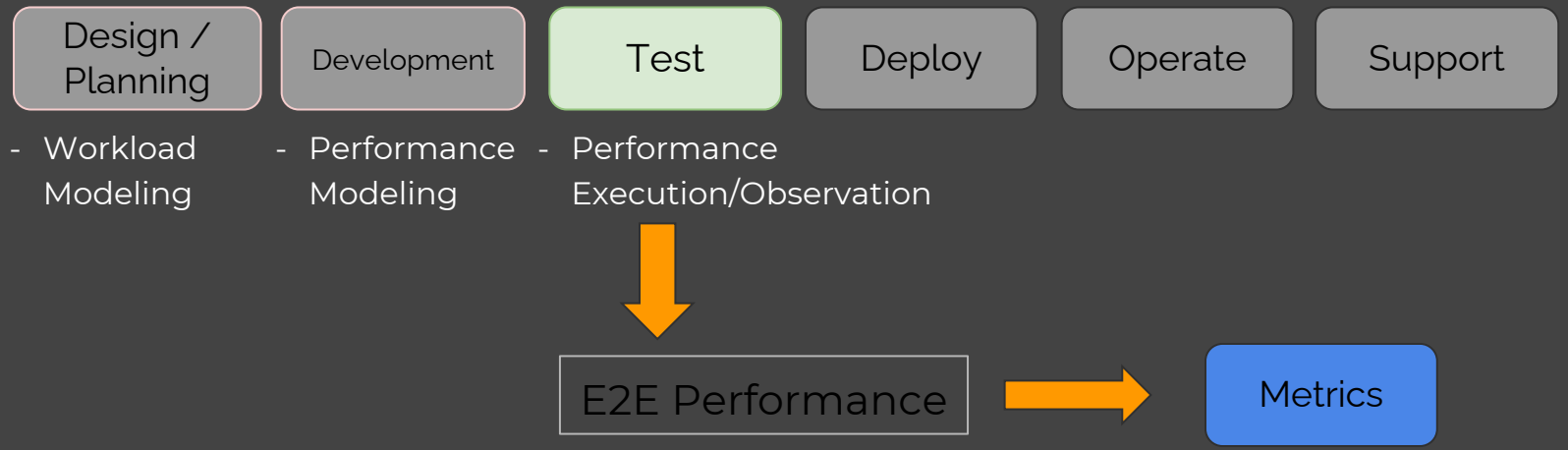
- Performance
- Chaos
- Security
- Functional automation

Performance Engineering

Planning and building the application
with performance in mind

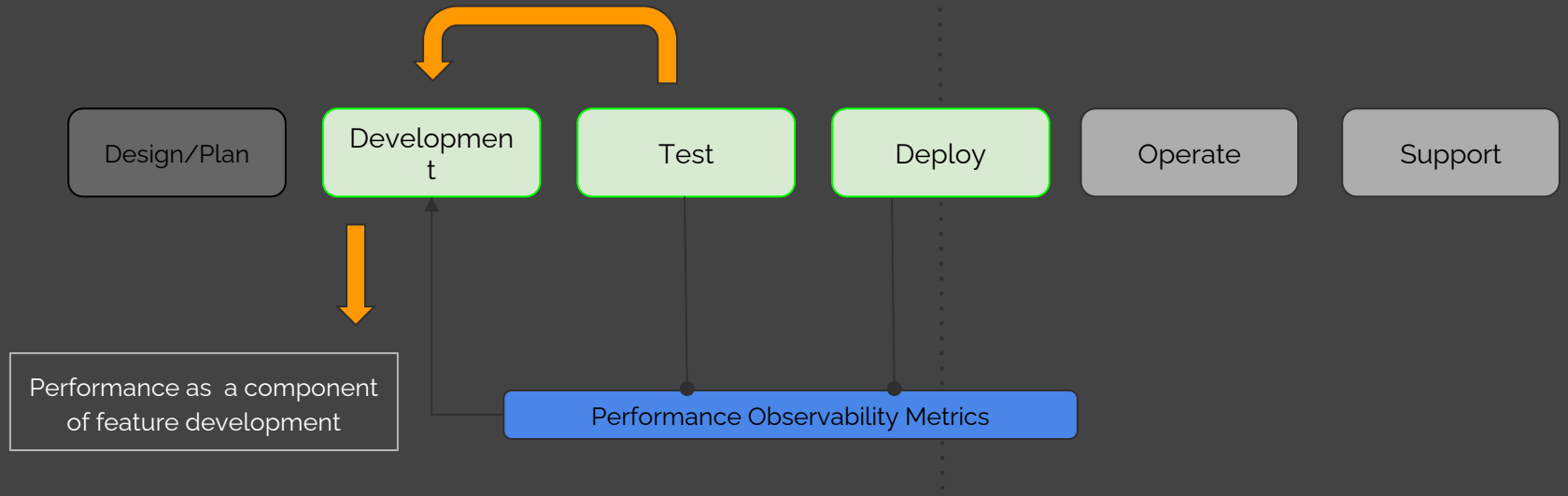
Traditional E2E Performance Testing

- Very waterfall orientated
- Testing late in the cycle leads to a larger feedback loop and in turn increasing cost, resources, energy and delivery time



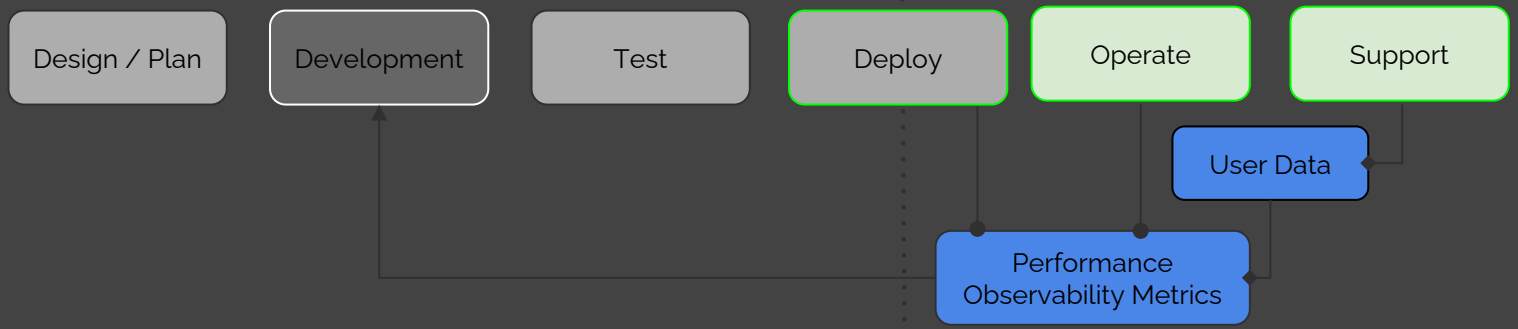
Performance Engineering
Shift Left

- Moving part of the performance testing process into the development phase



Performance Engineering
Move Right

- Using production data to verify performance assumptions
- Using reported and collected user data in conjunction with operational metrics to drive development priorities



Performance + SRE

People and process

- Understanding performance early
- Including user data
- Enhanced Traceability
- Reduced fix time
- Increased Transparency
- Opening communication channels
- Accountability (SLO's)

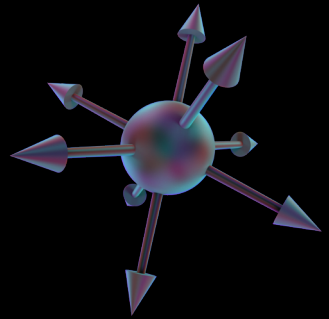
Measure

- Are we performing as expected (SLI's)

Implement gradual change + **Reduce organization silos**

Chaos Engineering(CE)

Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production



Chaos Engineering is NOT...

About breaking the system

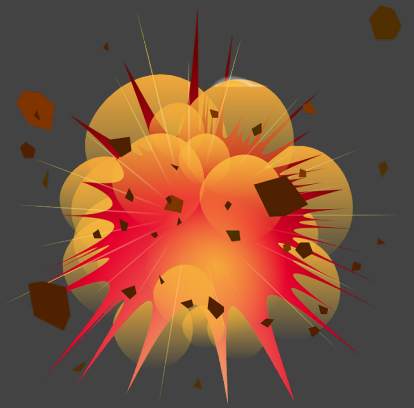
Chaos is about building a culture of resilience in the presence of unexpected system outcomes

It's all about understanding the end user experience and understanding how we are building tolerant systems

Chaos is about experimentation

The Approach

- ➔ Start By Defining the Baseline (Steady-State)
- ➔ Hypothesize the Steady-State Will Endure
- ➔ Introduce Variables/Experiments
- ➔ Try to Disprove the Hypothesis



The Pillars

- ➔ Adequate coverage
- ➔ Run often and in Prod (or similar)
- ➔ Minimising the blast radius

Chaos + SRE

People and Process

- Behaviour under varying conditions
- Cultural shift to what happens when things fail
- Improves how infrastructure is built and how services are consumed
- Builds internal trust and empathy

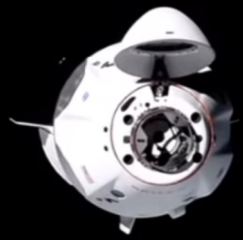
Measure

- Throughput when services disabled

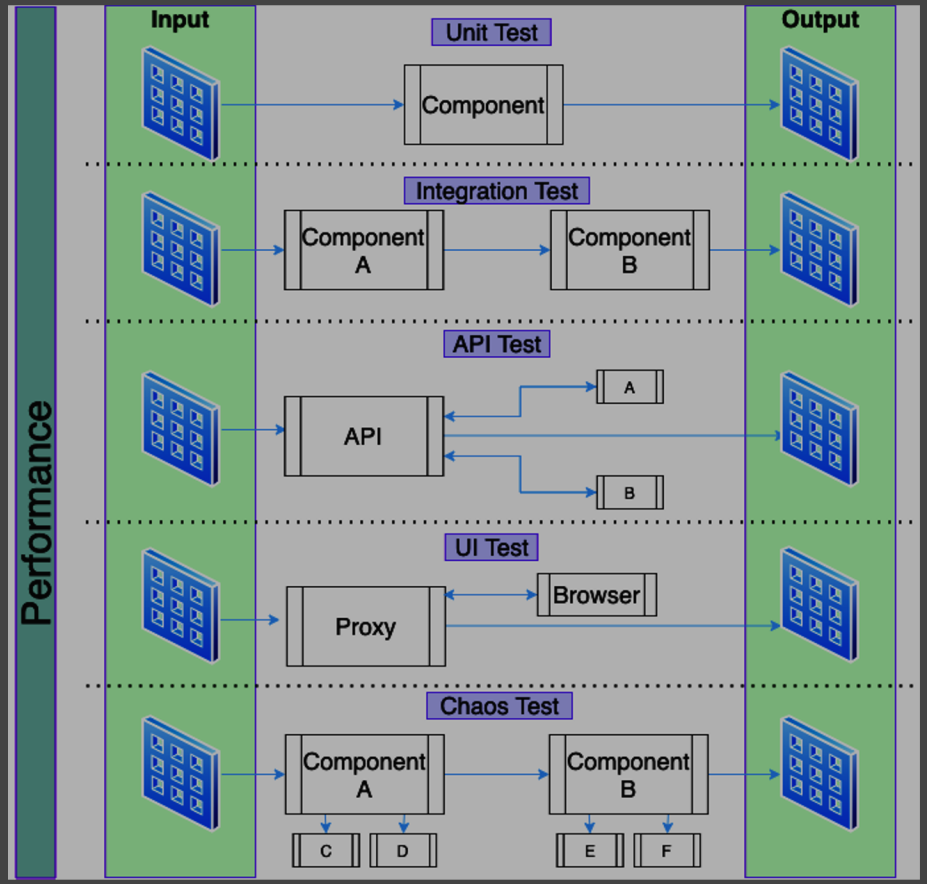
Accept failure as normal  **Reduce organization silos**

Functional Automation

Testing Resilience and reliability



The comparison



We take inputs and produce outputs we then compare them to what's expected in the way of assertions

Performance spans all boundaries as each approach can and should have a performance consideration

SRE + F Automation

People & Process

- Confirming expectations
- Understanding/representing logical flow
- Enhanced communication and socialisation
- Fast feedback cycles
- Reduced double handling
- Unbiased opinions

Measure

- Failing tests, Disabled tests
- Test presence ratio (Unit/Int/API/UI)

Leverage tooling & automation + **Reduce organization silos**


Security

Vulnerability testing

SAST

- Inside out approach
- Can be run early on (feature branches)
- Can be run against all code bases (app, services, apis)
- Easily automatable

DAST

- Outside in approach
- Used later on in the SDLC
- Only used for web app and services
- Uses fault injection techniques

IAST

- Scalable
- Reduced false positives

SRE + Security

People & Process

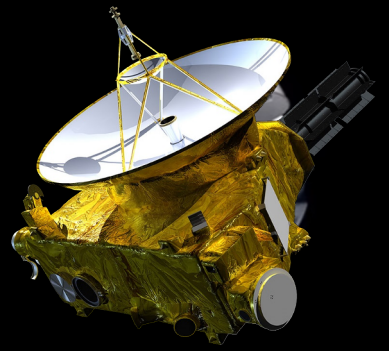
- Improved reliability / predictability
- Open discussions about sec
- Unbiased feedback
- Lowered costs

Measure

- Passwords found in code

Leverage tooling & automation + **Reduce organization silos**

Observability/Measurement

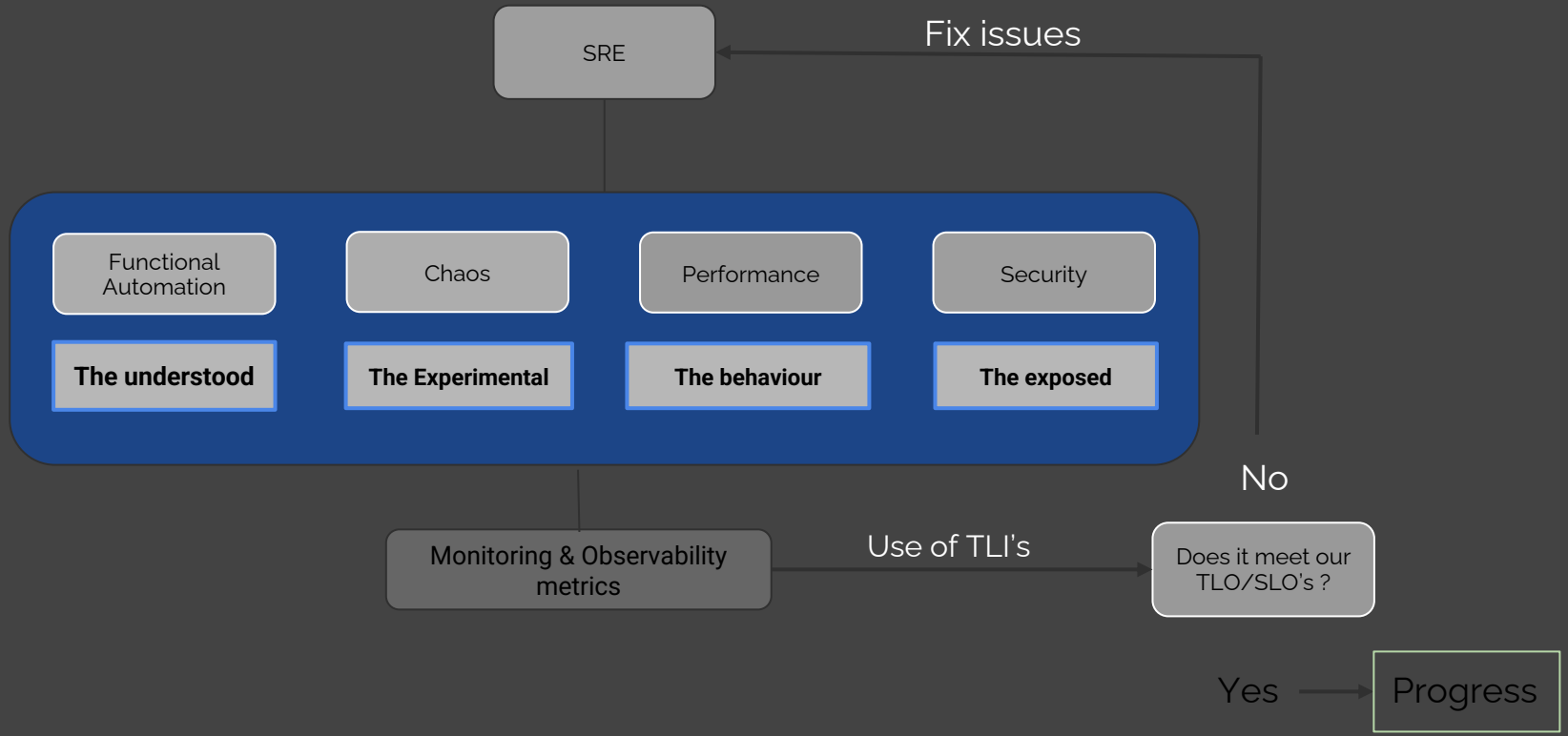


Understanding ~~test~~ state

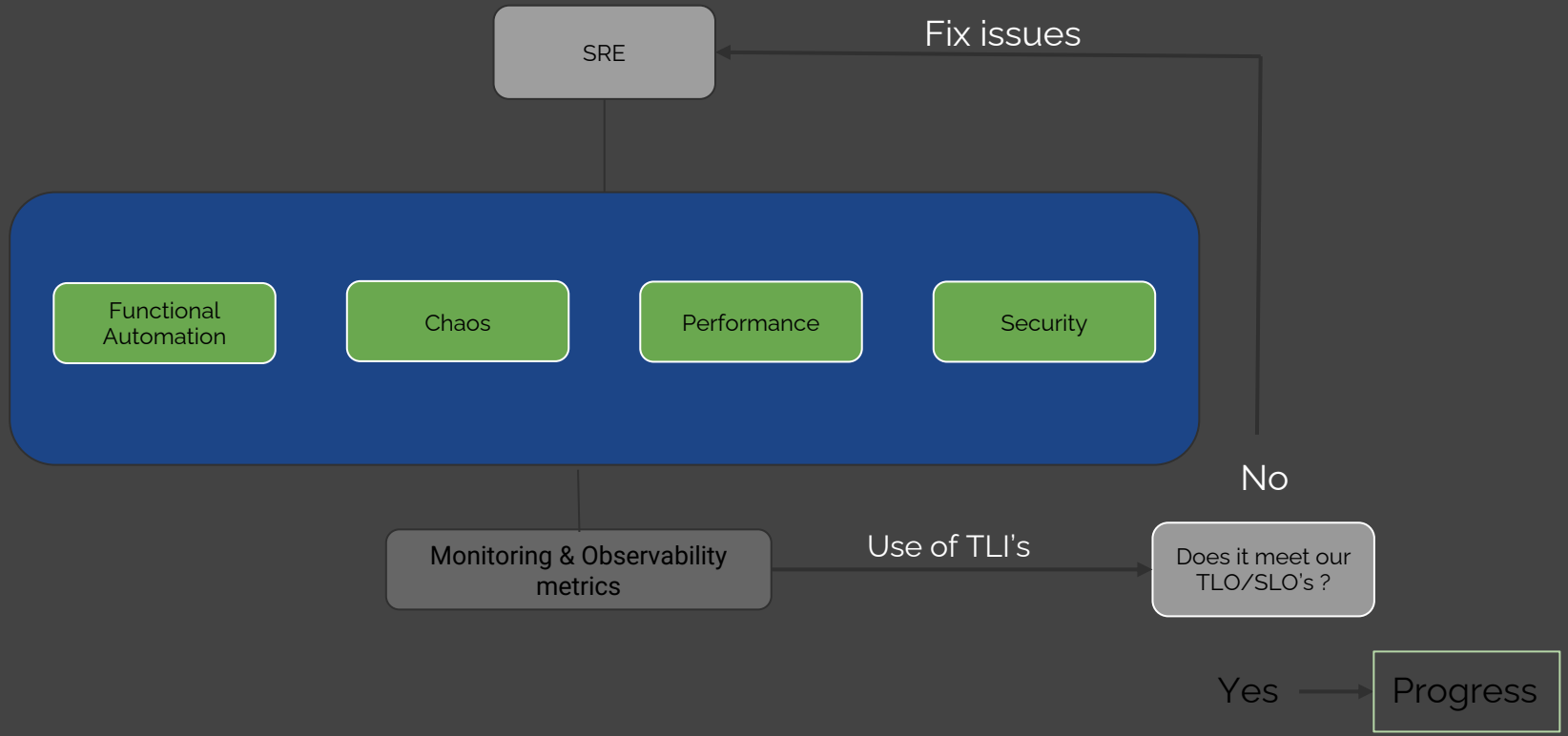
We use tests to influence and verify load, look for vulnerabilities, induce erratic behaviour and confirm the expected

We need a way to make sure that the results that are generated are processed, understood and acted upon

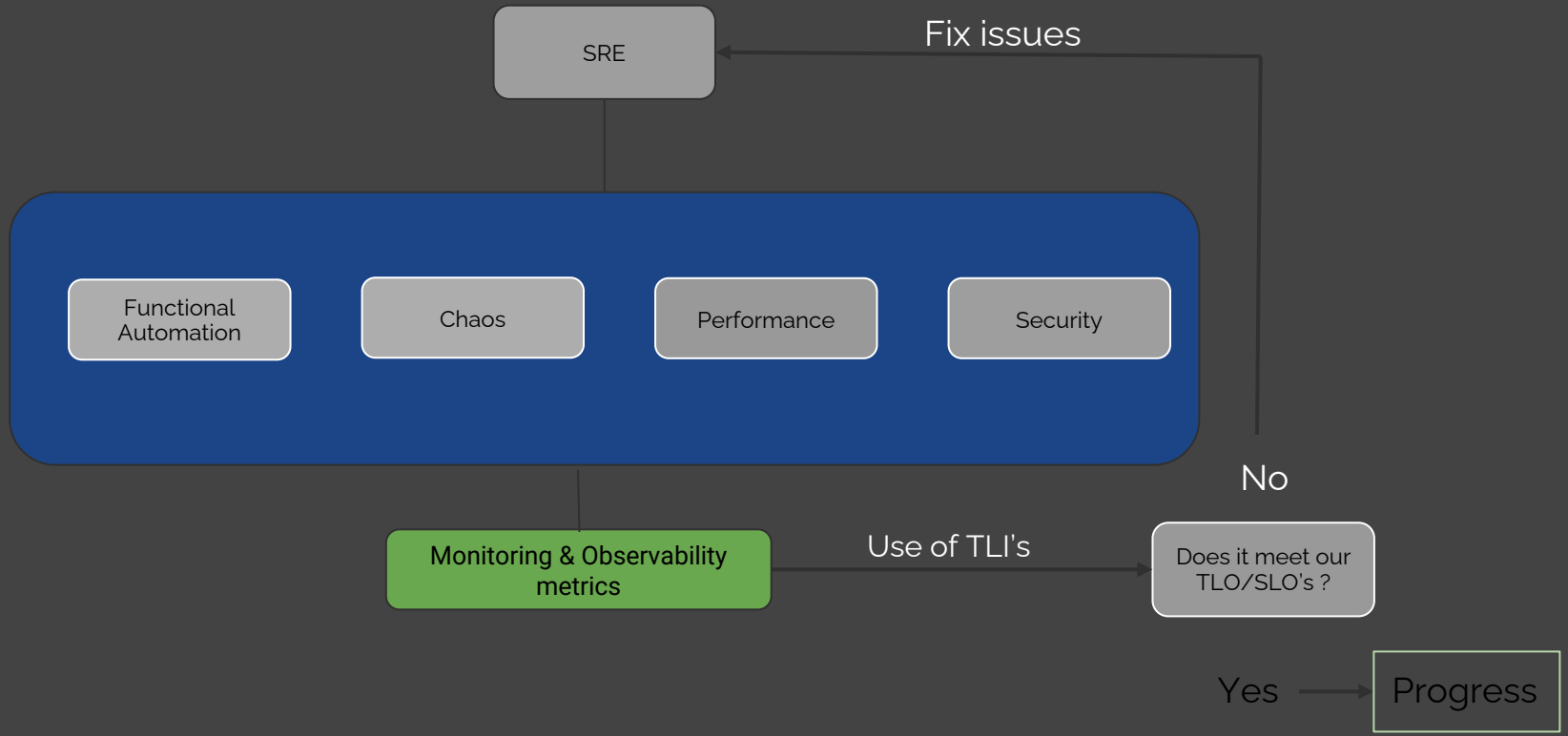
Collecting the data



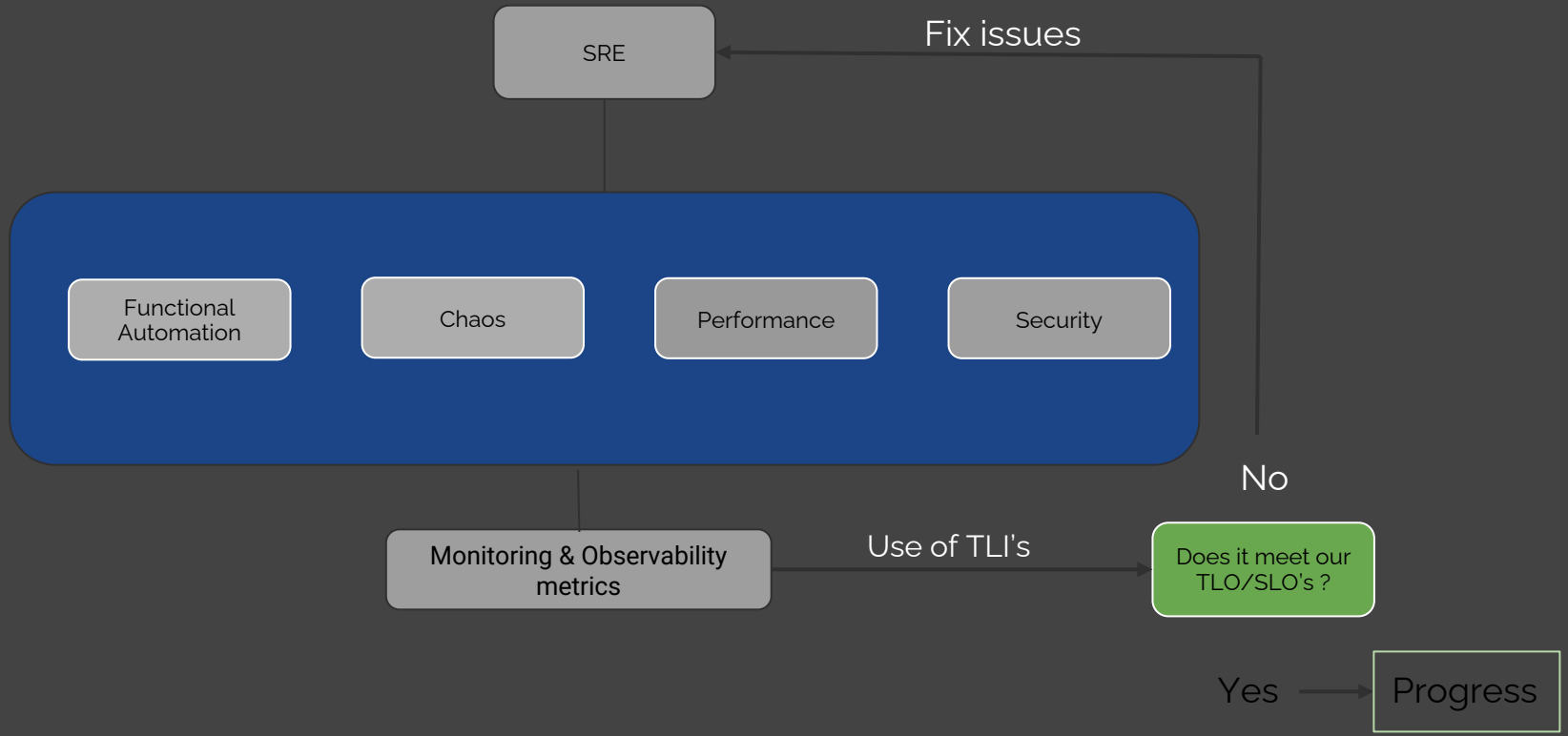
Collating the data



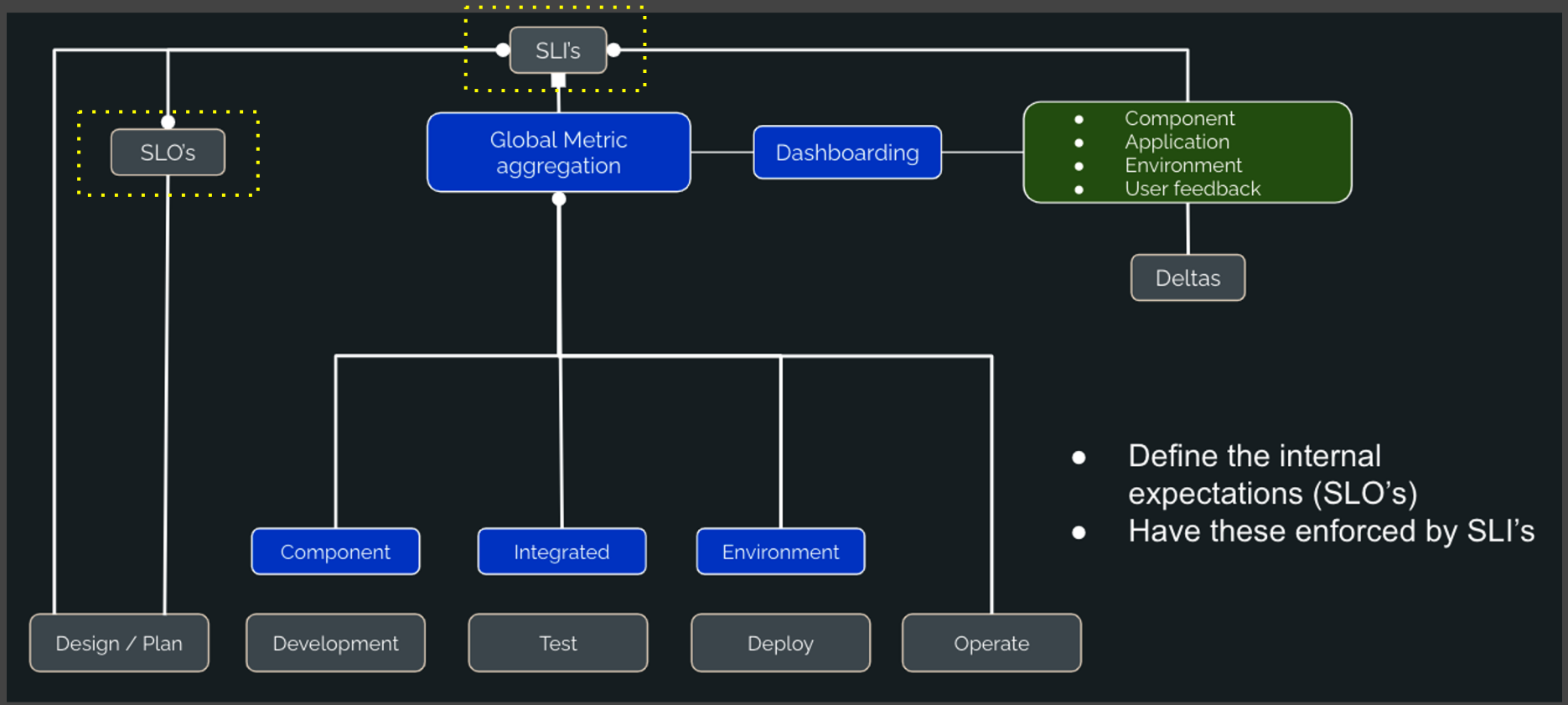
Processing the data



Evaluating the data



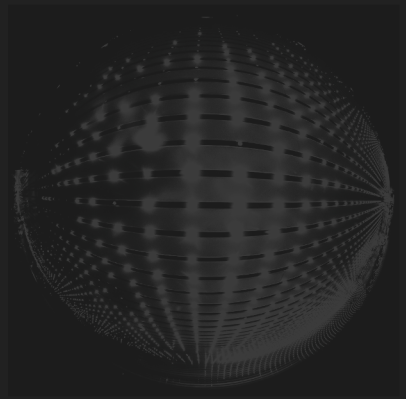
Doing the comparisons



- Define the internal expectations (SLO's)
- Have these enforced by SLI's

Bringing it all together

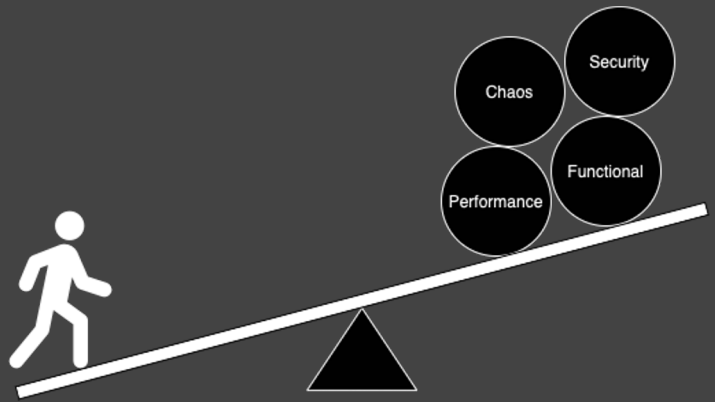
SRE +
Chaos, Performance, Test Automation, Security



Finding Balance

What's the right combination of tools and at what point in the development chain will they return the most benefit

- The right tool
- For the right domain
- At the right time
- Aligns to the teams maturity and skillset
- Operate open contribution model



Re-Cap

- **Measurement**
  - Understanding your system, applications and their behaviour
  - Determining what ‘ideal’ state looks like
- **Chaos**
  - Viewed as experiments to help understand behaviour
- **Security**
  - Vulnerabilities can cause long lasting business and series commercial implications
- **Performance**
  - Is broad and covers all other testing areas and disciplines and its important to include component and E2E strategies

Automation

- Finding the right balance of testing types can help increase communication, trust and in turn produce a better product

Useful links

Some Reading:
Principles of chaos engineering : https://principlesofchaos.org/

Tooling:
SRE/DevOps performance : https://artillery.io/
Keptn - event based control plane for cloud native : https://github.com/keptn/keptn
Chaos tool : Gremlin https://gremlin.com

Recent conferences
https://www.sloconf.com/
https://www.cloud-native-sre.wtf/

Contact

Web : https://scottgriffiths.me
Linkedin : https://www.linkedin.com/in/scgriffiths/