basware
Simplify Operations, Spend Smarter.

ROLE OF OBSERVABILITY
in DevOps Transformations

Alistair Gilbert
VP, Product Development • R&D_DevOps
Basware

AGENDA

01 What is DevOps

02 Importance of nonfunctional requirement management for larger enterprises

03 How to conduct a DevOps transformation

04 What is Observability

05 What makes for a highly observable system

06 What makes for a good monitoring system

07 How to achieve great observability

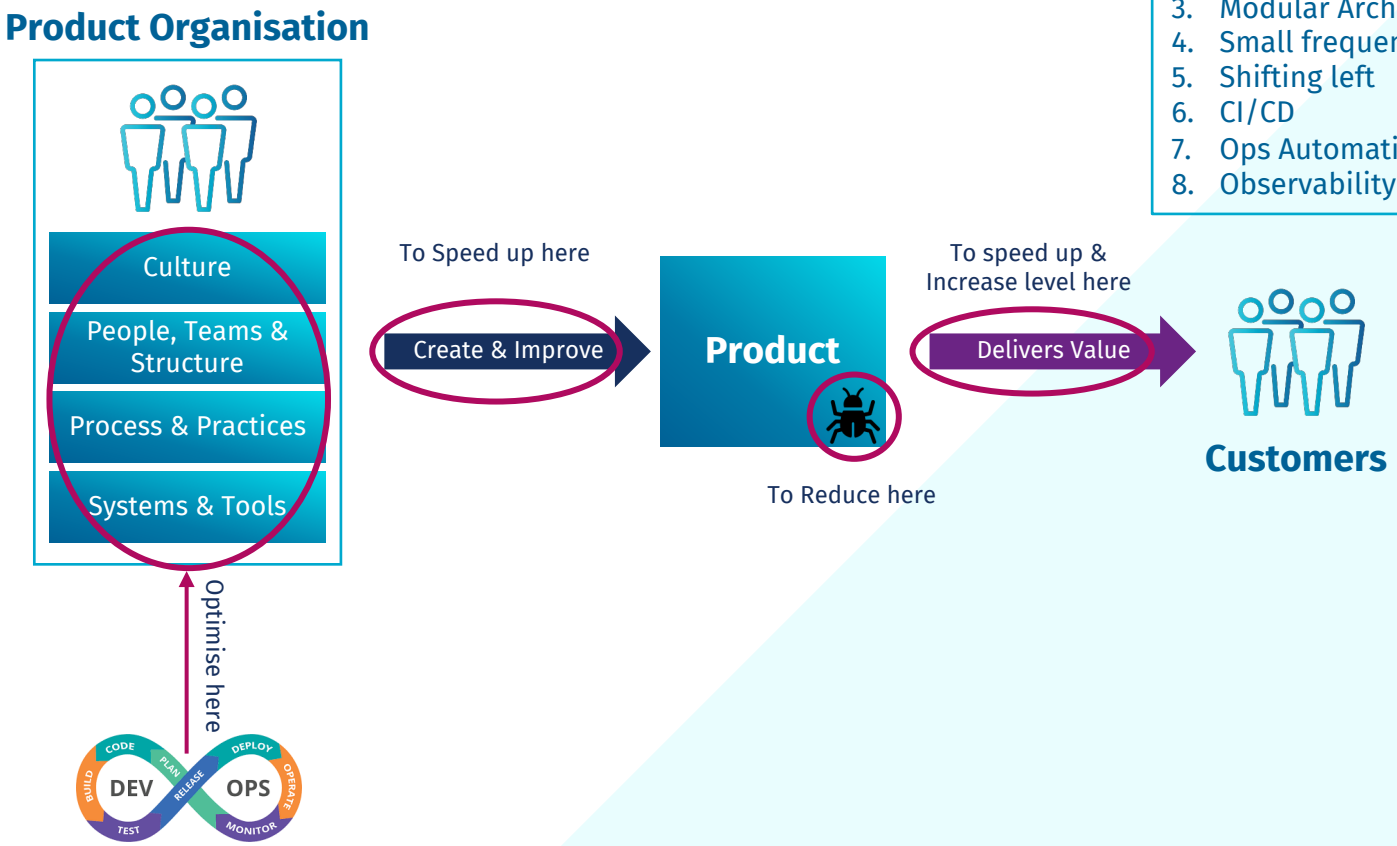
08 What makes observability so important to DevOps

09 Increasing observability through dynatrace

WHAT IS DEVOPS

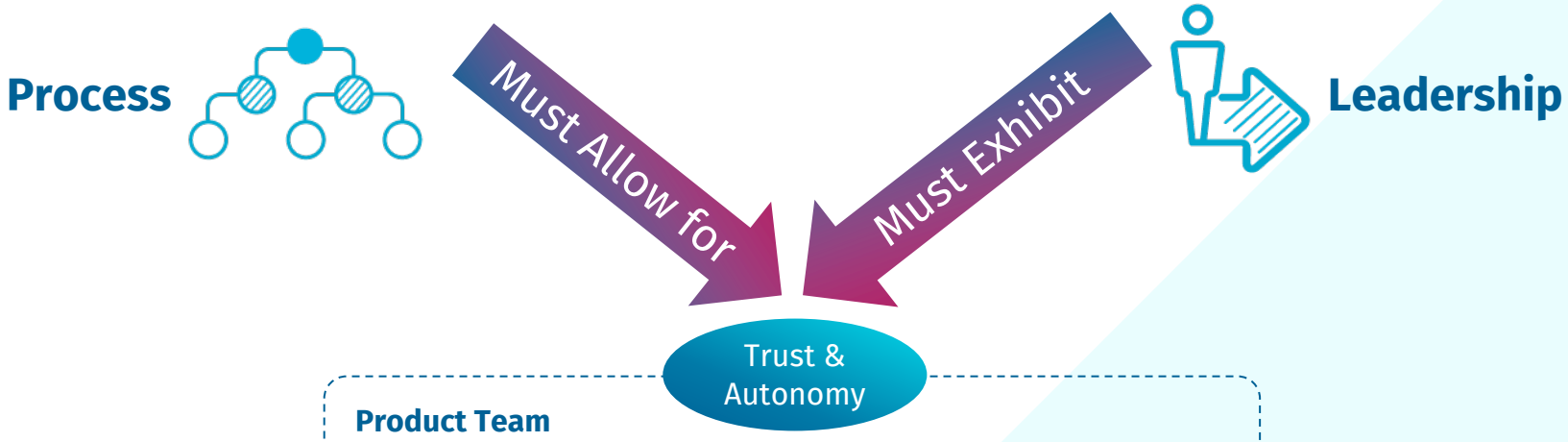
WHAT IS DEVOPS

Domains of Change:
1. Process & Leadership
2. Team Accountability
3. Modular Architecture
4. Small frequent changes
5. Shifting left
6. CI/CD
7. Ops Automation & NoOps
8. Observability



PROCESS & LEADERSHIP

Enables team trust and autonomy

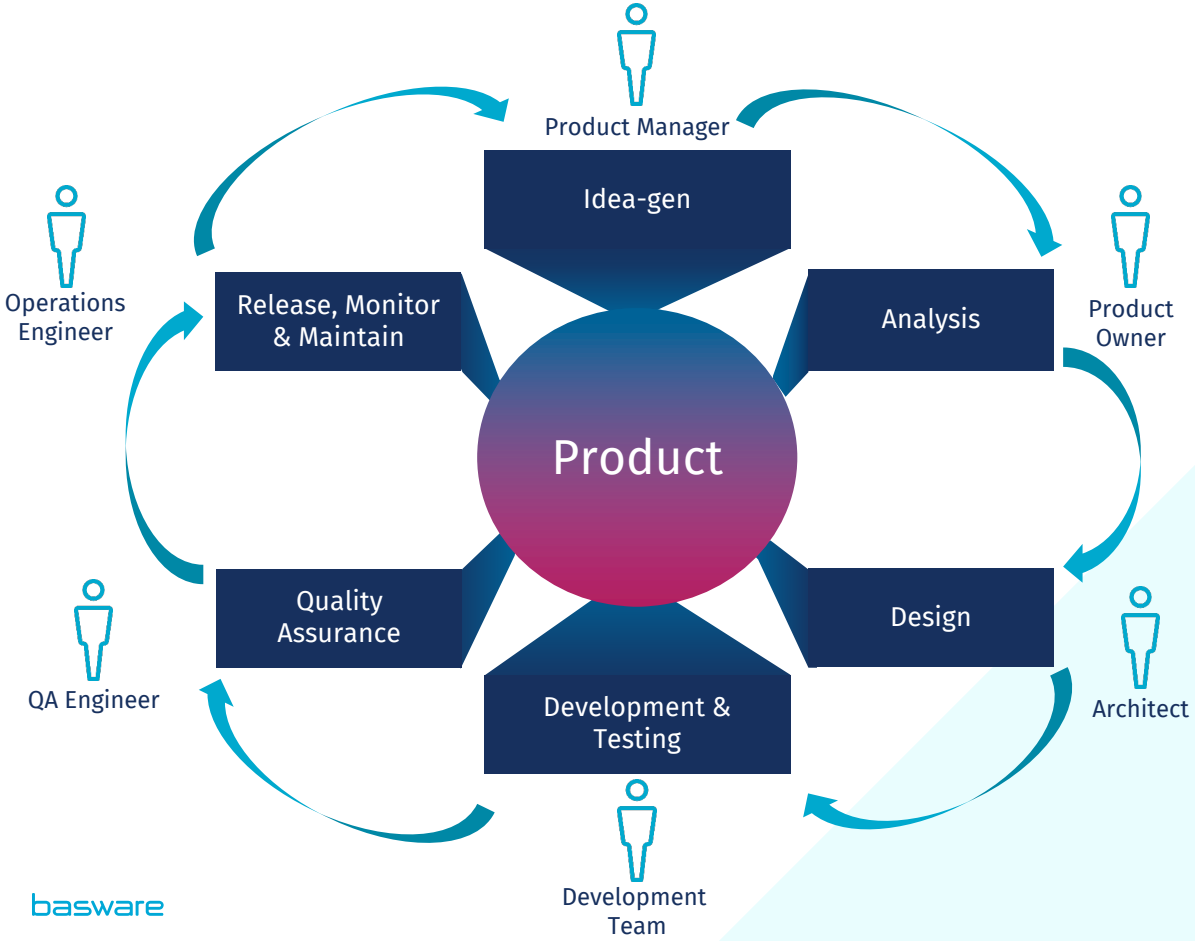


Product Team

Trusted and enabled to define (at least to some degree) their:

- Development model & practices
- Technology choices
- Testing approach and strategy
- CI/CD practices
- Observability and Monitoring needs and approach
- Release approach and cadence
- etc

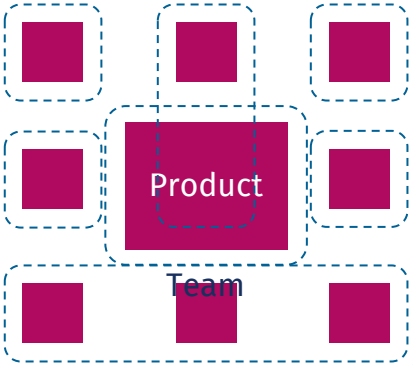
TEAM ACCOUNTABILITY



Single team responsible for all aspects of the product means there are no handovers and the team as a whole has complete accountability for every aspects of the product.

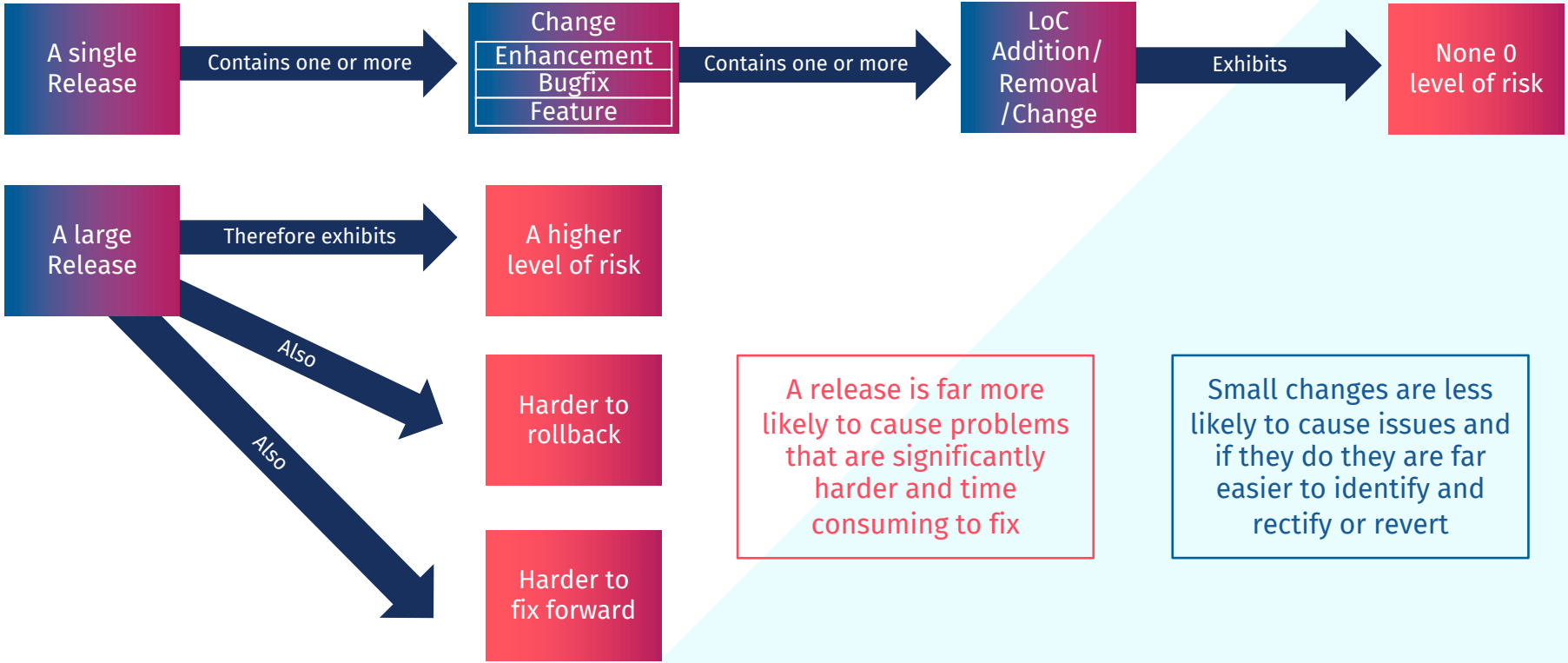
MODULAR ARCHITECTURE

Without a clear architectural plan the product code base WILL grow beyond what a single team is capable of managing. Specialised sub teams will emerge and handovers will be introduced.



A solid architectural strategy must be in place that sets a clear definition for functional context bounding that can drive appropriate modularisation of the code base, allowing for teams to take full accountability for specific areas of the product.

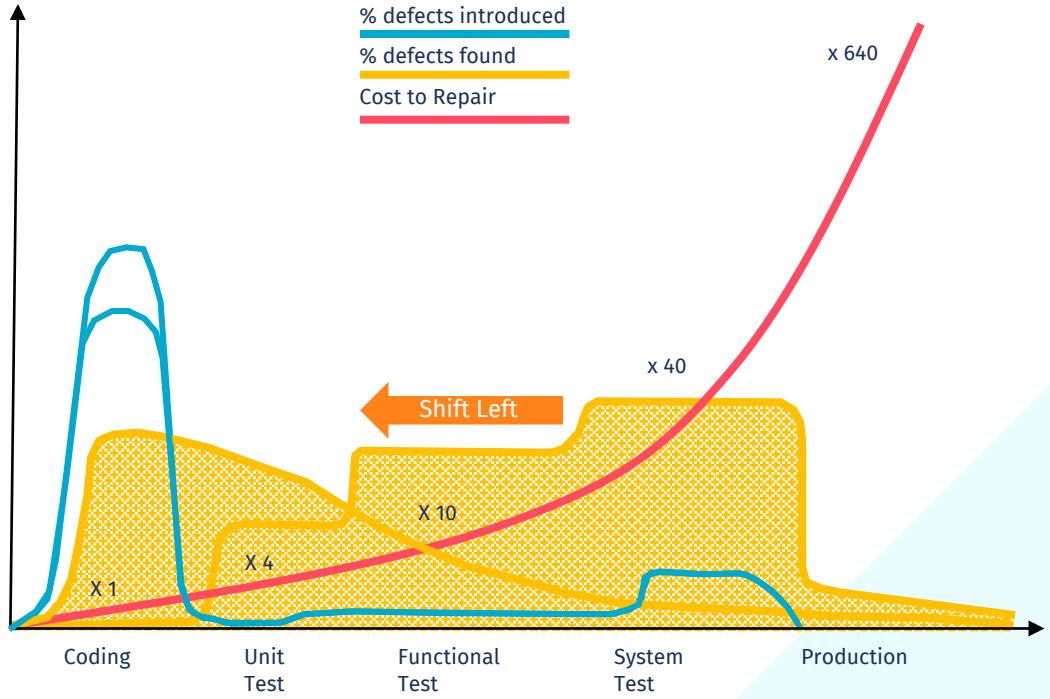
SMALL FREQUENT CHANGES



A release is far more likely to cause problems that are significantly harder and time consuming to fix

Small changes are less likely to cause issues and if they do they are far easier to identify and rectify or revert

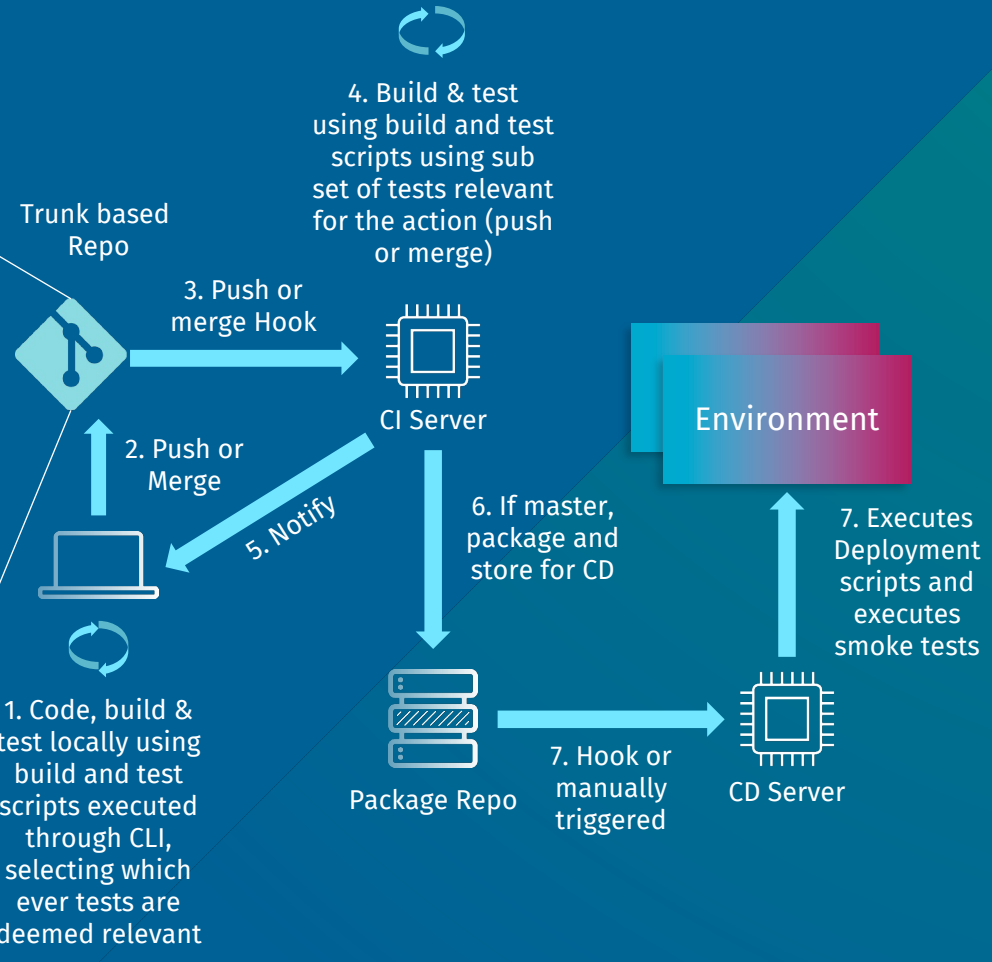
SHIFTING QUALITY LEFT



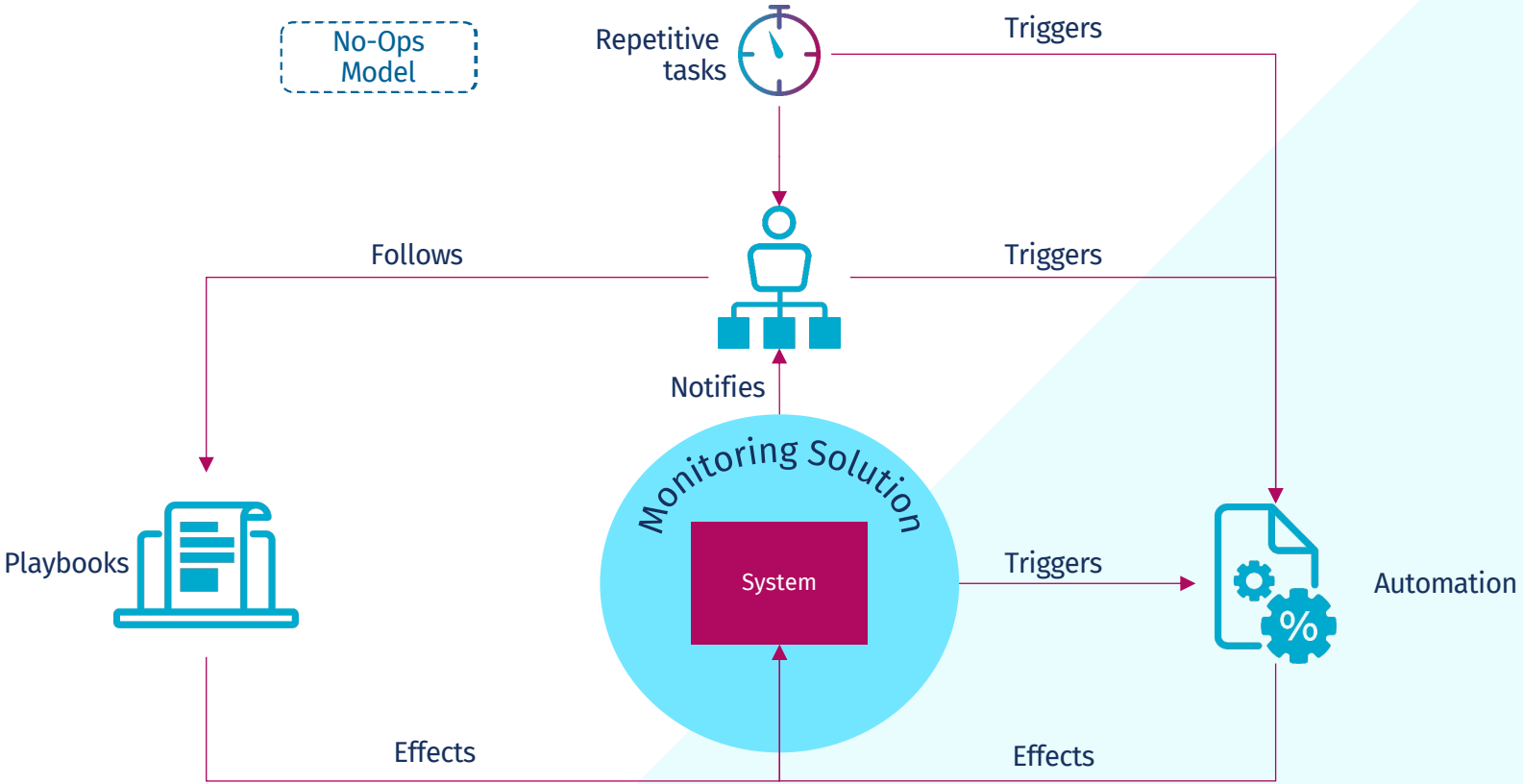
- Employ TDD Practices
- Design and Code for testability
- Track test coverage and gate according to testing pyramid
- Ensure local testability at every opportunity
- Build developer tooling to support local testability
- Adopt Code Quality / Complexity tooling into pipeline and IDEs
- Adopt OSS license compliance tooling into pipeline and IDEs
- Adopt 3rd Party library vulnerability tooling into pipeline and IDEs
- Adopt static code analysis tooling for security into pipeline and IDEs

CI/CD

Includes:
- *Product Code*
- *Full Stack as Code*
  - *Infra as Code*
  - *Platform as code*
  - *Config as code*
  - *Monitoring as code*
  - *etc*
- *Build Scripts*
- *Deployment Scripts (and env config)*
- *Test Automation & Orchestration scripts*



OPERATIONAL AUTOMATION & NO-OPS



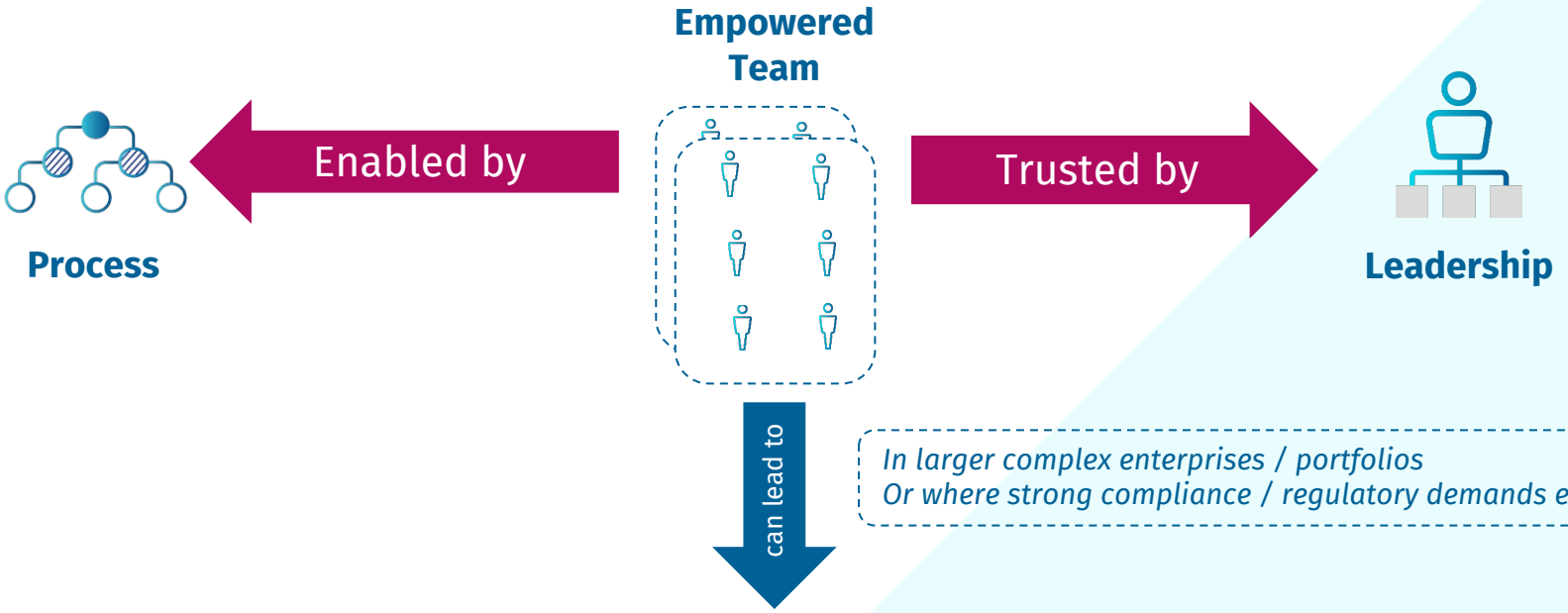
OBSERVABILITY

Covered in more detail in subsequent slides

Observability *– A measure of a how effectively a* ***product*** *and it's surrounding* ***monitoring solutions*** *have been* ***integrated together*** *so as to ensure* ***rapid*** *detection of problems (****without configuration****) together with a detailed understanding of* ***actionable causal relationships.***

IMPORTANCE OF NONFUNCTIONAL GOVERNANCE

UNCHECKED TEAMS



CENTRES OF EXCELLENCE -> STANDARDS

Process

Central CI/CD *

Leadership

Enforces

COE Teams

- Architecture
- DevOps
- Development
- Testing

Define

Define

Define

Define

- Non-functional Requirements
- Standard Approaches
- Patterns
- Templates
- etc

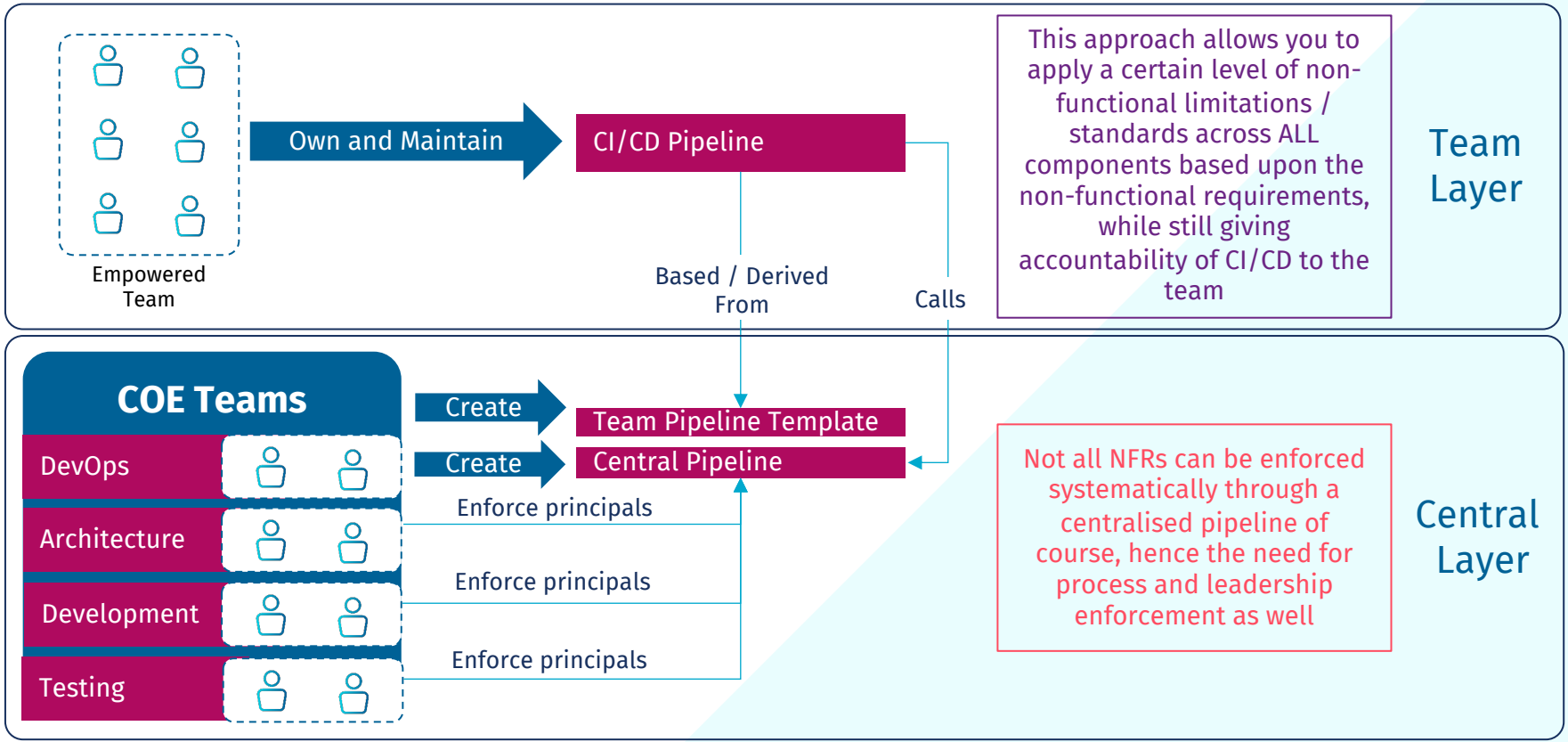
Examples:

1. *Dynatrace is the only observability tools to be used*
2. *All logs must contain customer id*
3. *80% UT/IT test coverage required*
4. *All data must be owned by a single component and only that component can access the data directly*

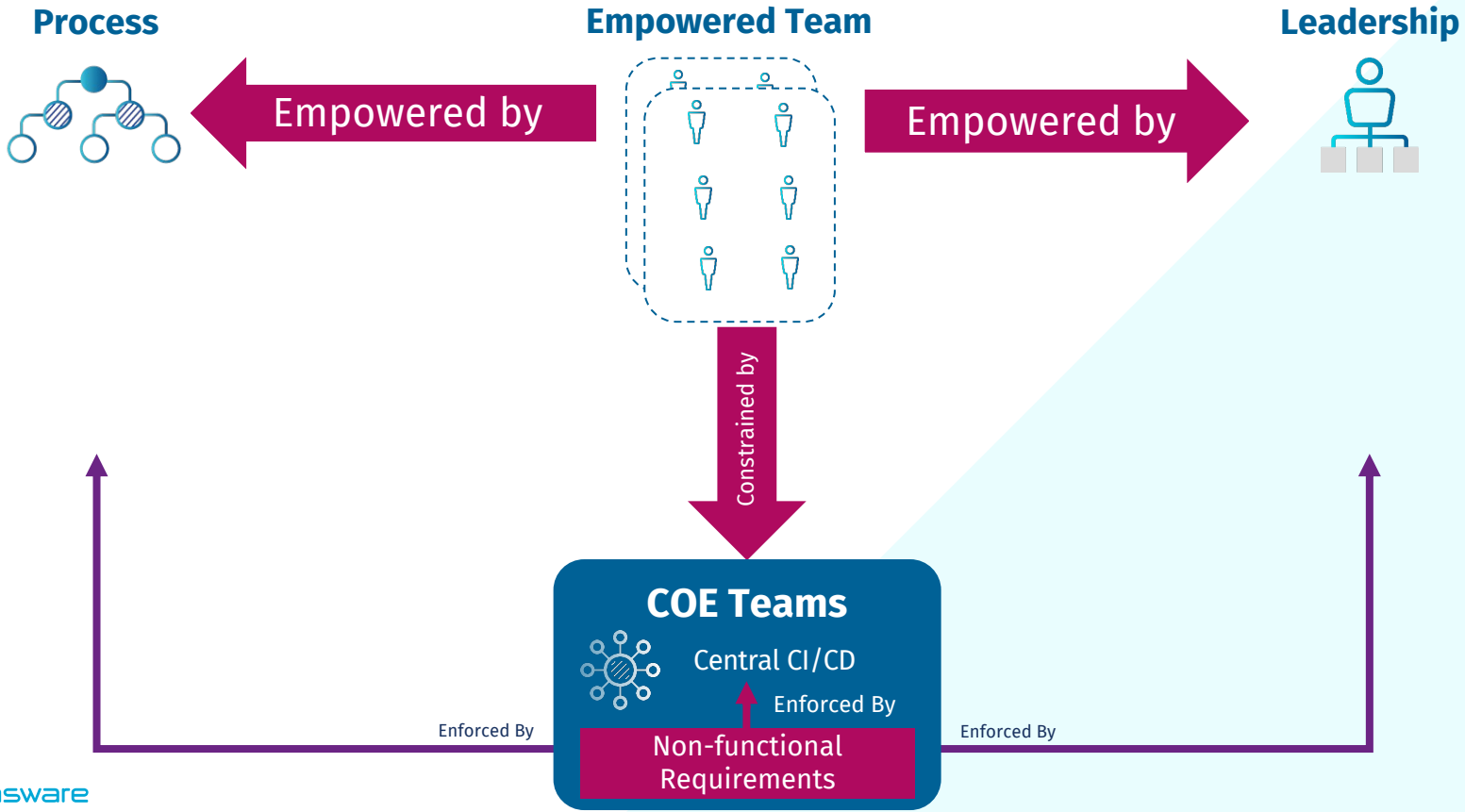
basware

* Explained on next slide

CI/CD - ENFORCING STANDARDS



CONTROLLED EMPOWERMENT



DEVOPS TRANSFORMATIONS

WHAT IS A DEVOPS TRANSFORMATION

- Simply the process of making all the changes described on previous slides within an organisation

- **BUT** - Each of the individual changes can be very complex in their own right, especially in large, complex, multi product organisations

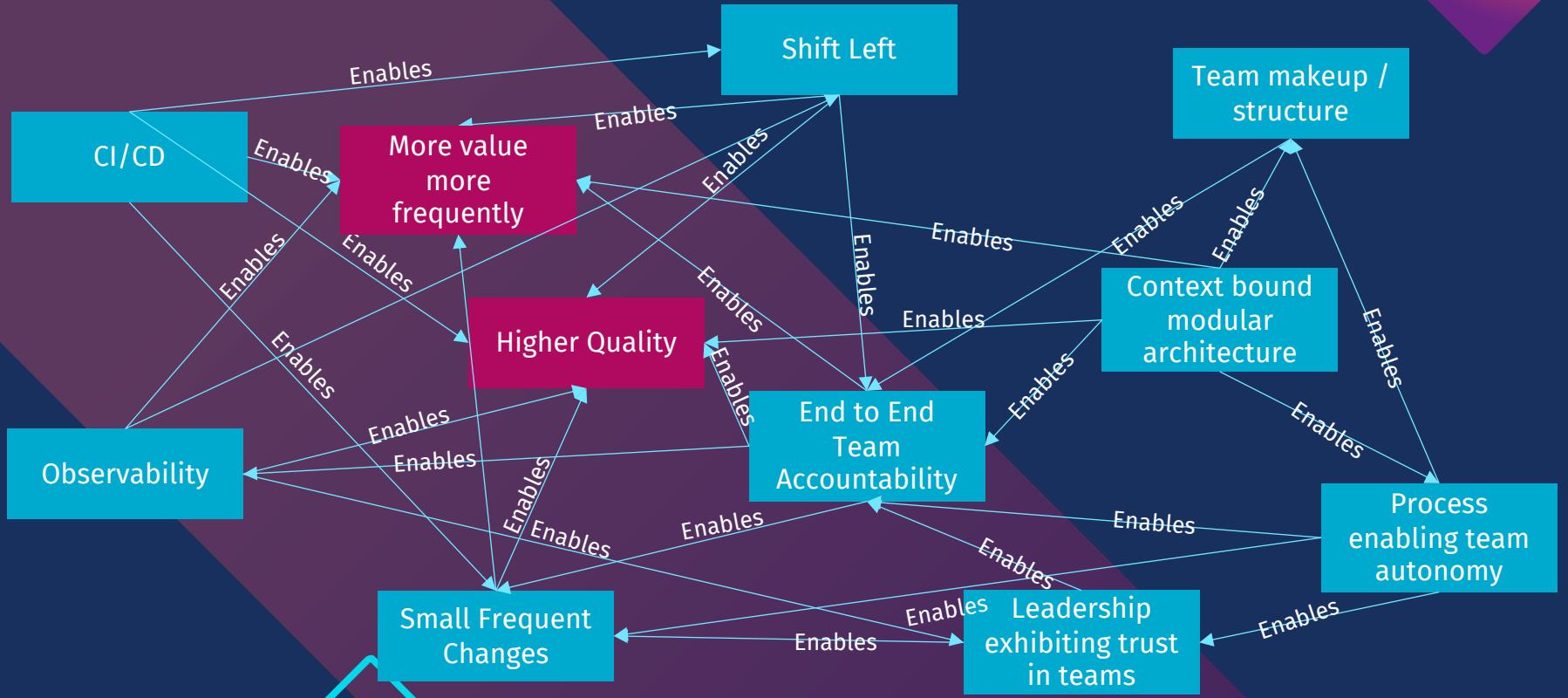
- **AND** – all of the domains of improvement depending on one another, so gradual, iterative change across all the domains in parallel is required

NOTE: This point above is critical to this presentation as it suggests two things

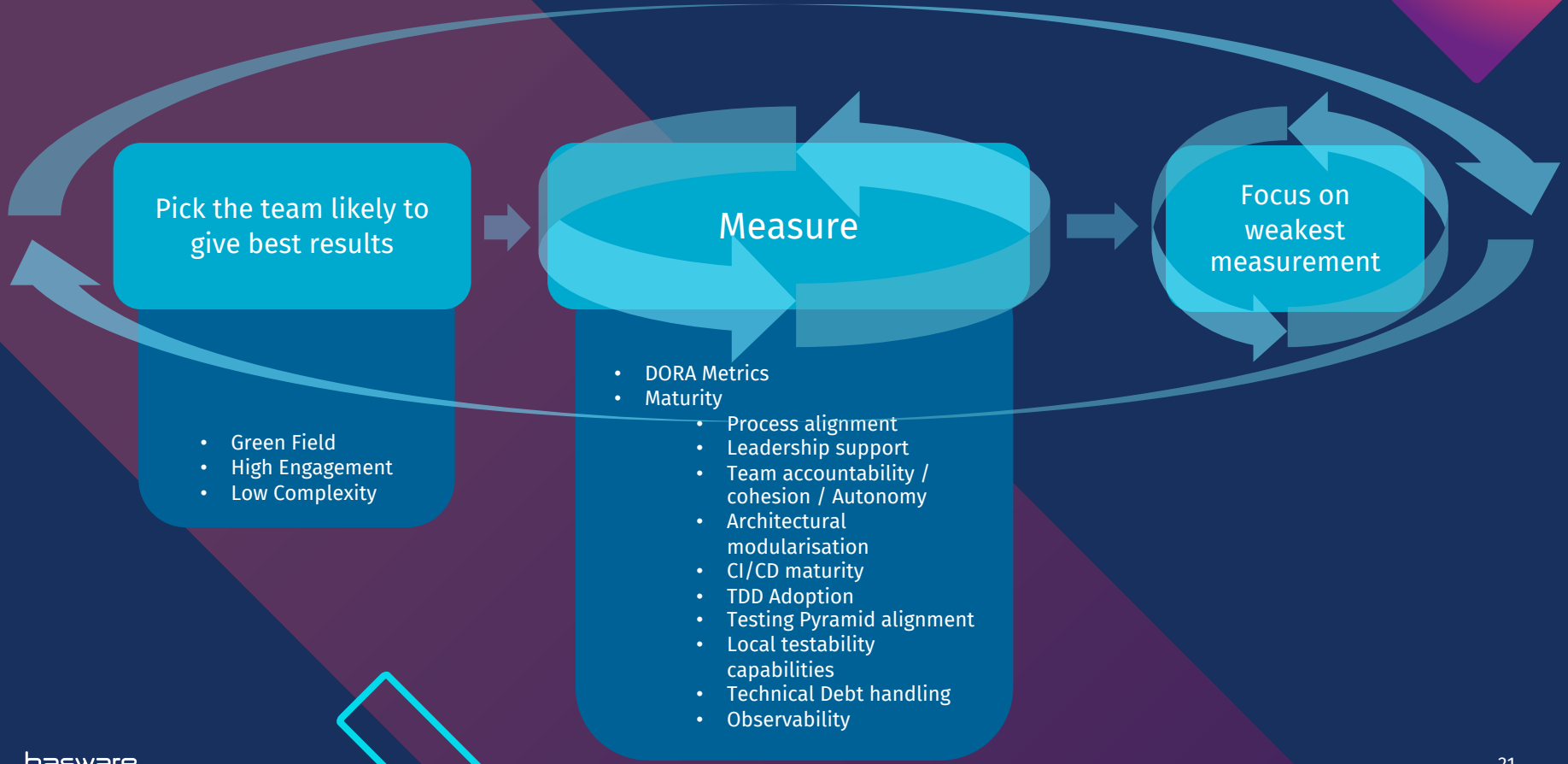
One must deliver improvements to Observability in order to succeed with a DevOps Transformation

*One **cannot** achieve improvements in Observability without improvements in other domains in parallel*

DOMAIN INTERDEPENDENCE



KEY TO SUCCESS

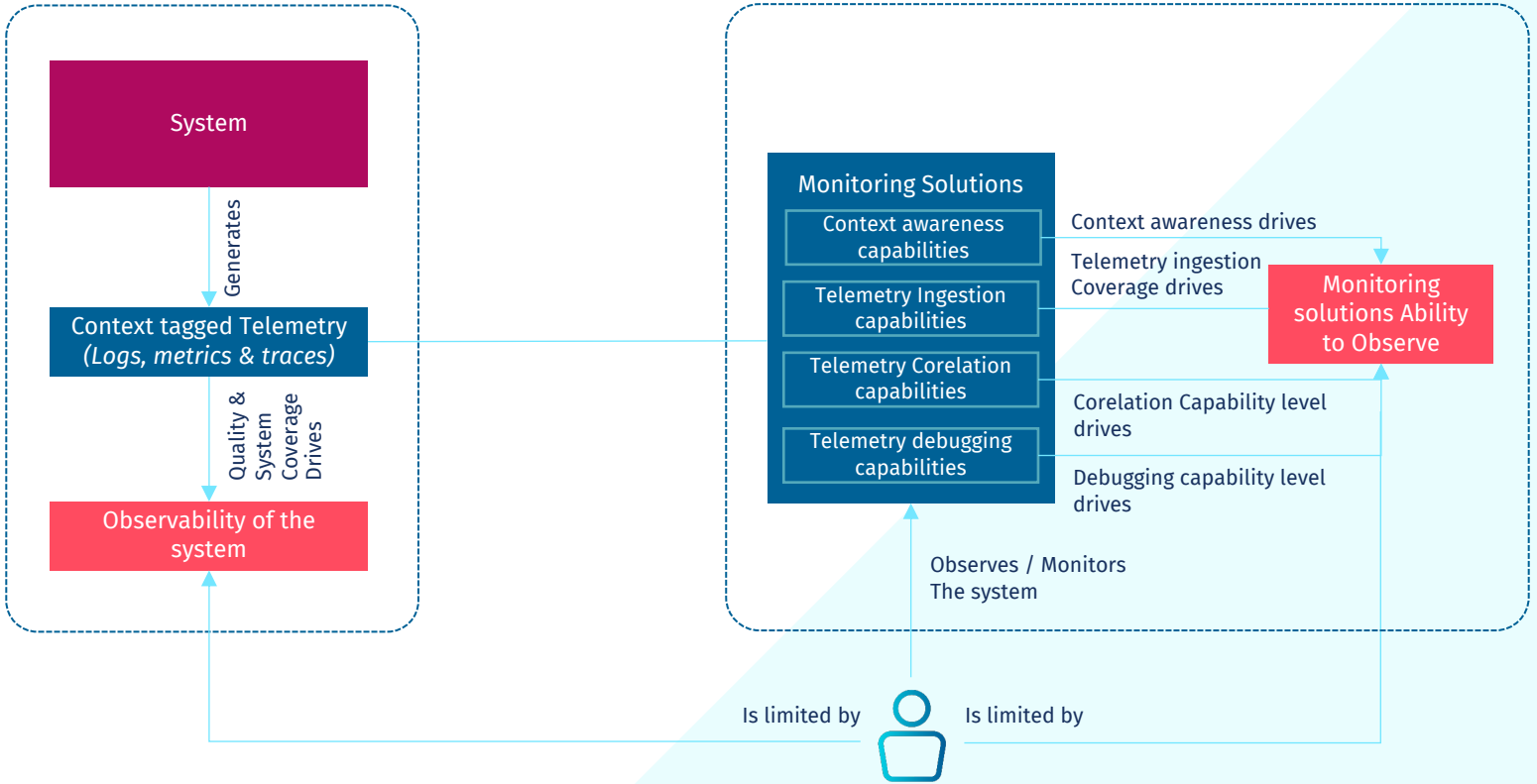


OBSERVABILITY

TERMINOLOGY

What makes a highly **OBSERVABLE SYSTEM**

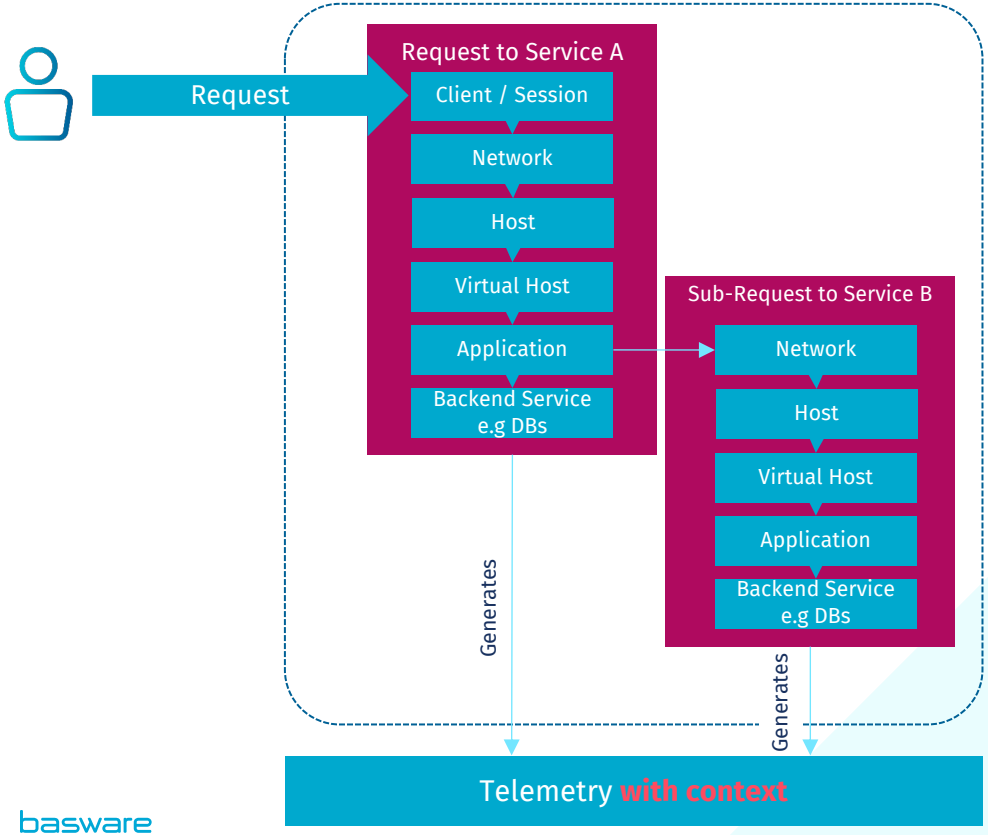
What makes a good **MONITORING SYSTEM**



WHAT MAKES A HIGHLY OBSERVABLE SYSTEM

WHAT IS OBSERVABILITY?

Context + telemetry is king



Telemetry

Telemetry is the atomic "unit/ building block" of observability.

It is the log entry, metric or application trace.

As such how well the available telemetry covers the **complete system** drives **good observability**

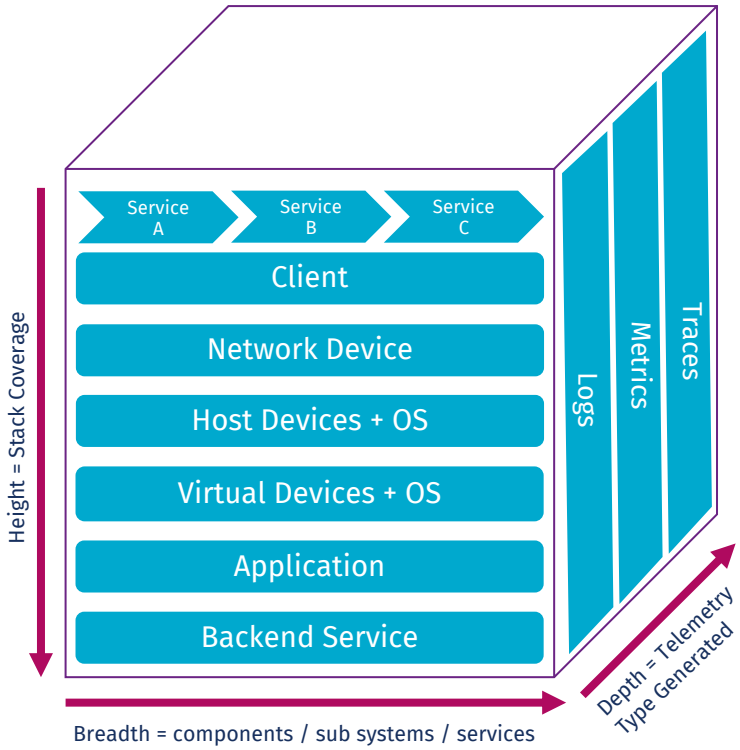
Context

Context is the glue that gives telemetry meaning in the context of what the system is intended to do and allows for relationships between telemetry to be derived

Context is therefore critical to **good observability**. It allows the system to be observed **within the context of the systems / process function**.

"If observability was a vector, telemetry would be the magnitude, context would be the direction"

TELEMETRY COVERAGE



- Good Telemetry Coverage drives good observability
- To have good coverage you need
  - Breadth
  - Height
  - Depth
- **Breadth** is governed entirely by the owning organisation of the service, they control what services generate what telemetry
- **Height & Depth** is ultimately governed by the owning organisation as well, but is done so in two different ways.
  - Application and Client layer coverage is governed by how well the organisation decide to generate logs, metrics and traces information within the code of the product
  - The other layers coverage is governed by the technology choices made and configuration parameters that are set against these layers.
    - E.g Network devices typically will only generate logs and metrics and will do so only according to how the device is configured

TELEMETRY CONTEXT

Service A

- Client / Session
- Network
- Host
- Virtual Host
- Application
- Backend Service

Can't typically inject context

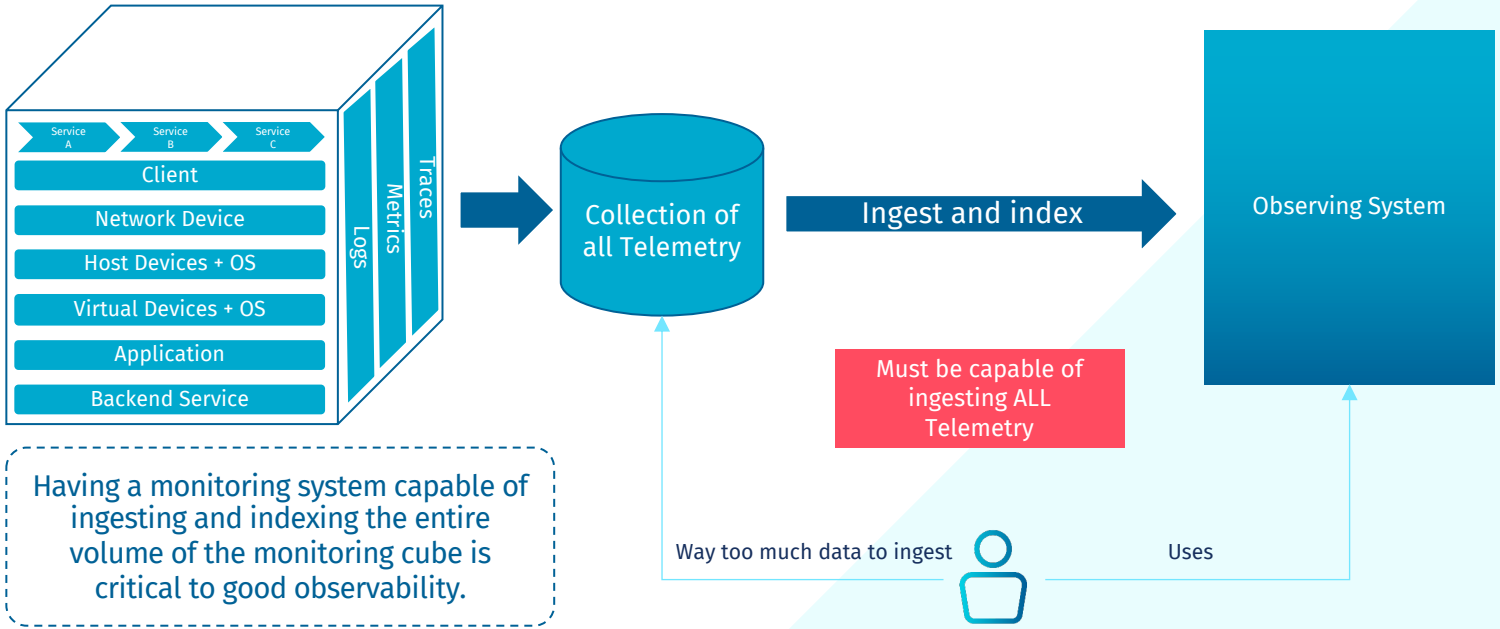
These layers do indeed generate telemetry and the telemetry will also have context. However the context will be relevant to the component itself. It is not typically possible to inject custom "system" context into the telemetry as components in these layers are typically facilitated through black box / off the shelf technologies.

Can inject Context

These layers are the layers for which application code is created by the owning organisation. Organisation can therefore inject rich system context to the telemetry through a mechanism known as instrumentation.

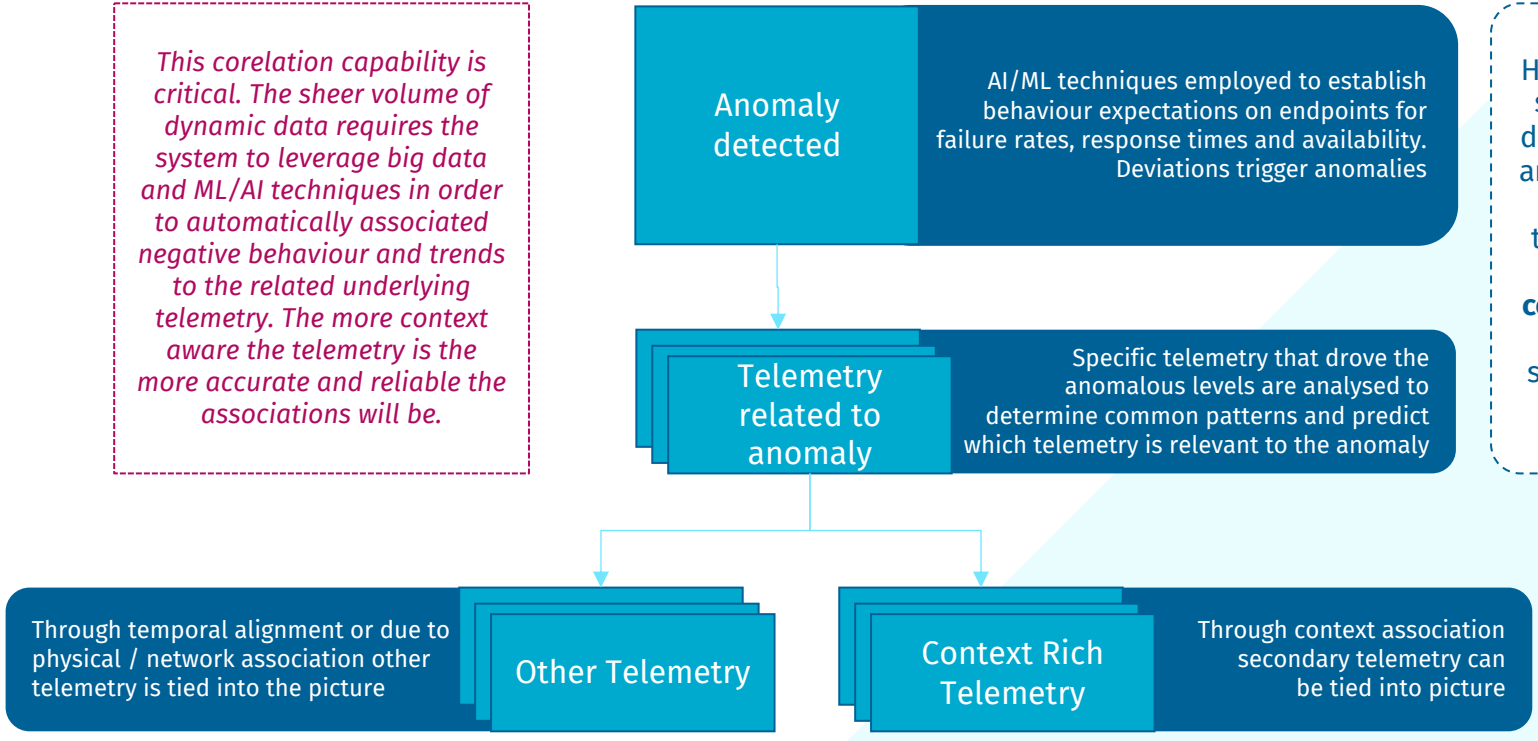
WHAT MAKES A GOOD MONITORING SYSTEM

TELEMETRY INGESTION



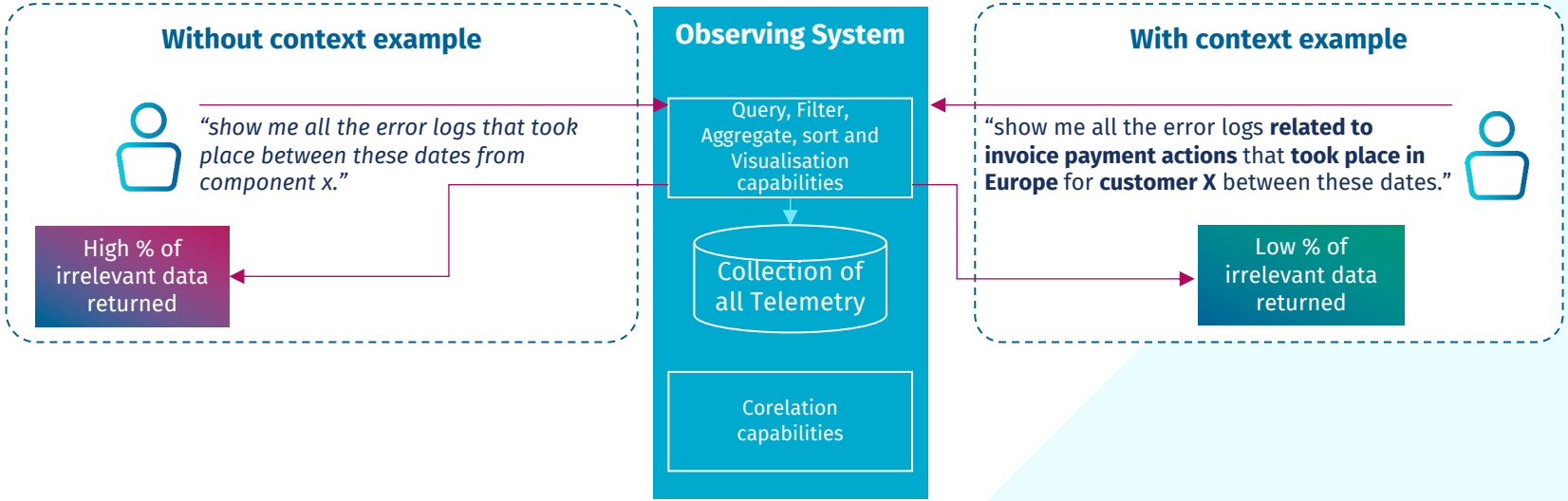
TELEMETRY CORELATION

This corelation capability is critical. The sheer volume of dynamic data requires the system to leverage big data and ML/AI techniques in order to automatically associated negative behaviour and trends to the related underlying telemetry. The more context aware the telemetry is the more accurate and reliable the associations will be.



Having a monitoring system capable of detecting anomalies and corelating these to truly related telemetry **using an understanding of context** provided by the observable system is critical to observability.

DEBUGGING CAPABILITIES

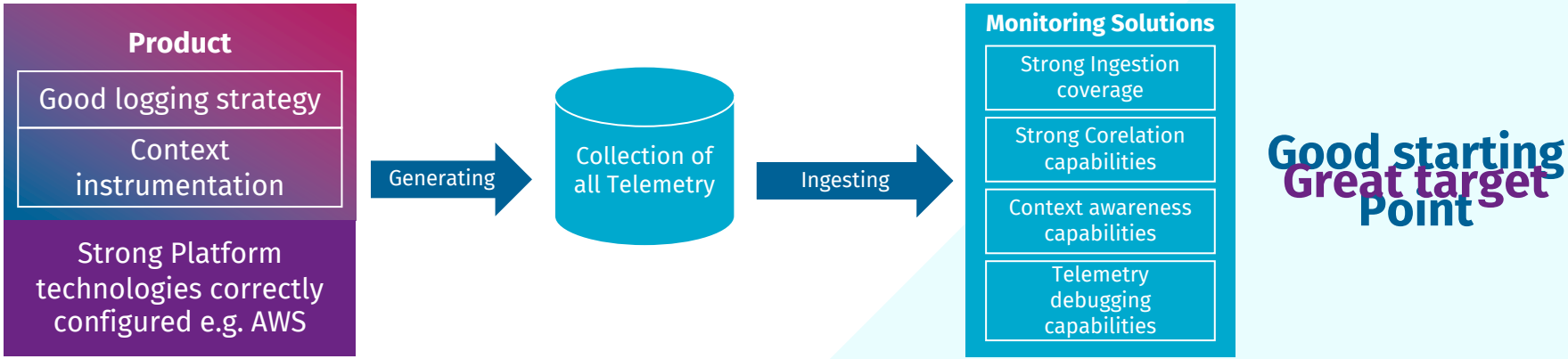


Having a monitoring system capable of being asked on the fly questions **with rich system context included in the question** and getting relevant and actionable responses is critical to good observability.

"Debugging is the act of asking questions about the behaviour of the system on the fly and getting answers that allow you to derive actions to improve / correct."

HOW TO ACHIEVE GREAT OBSERVABILITY

HOW TO ACHIEVE GREAT OBSERVABILITY



WHAT MAKES OBSERVABILITY SO IMPORTANT TO DEVOPS

DORA METRICS

A measure of devops success

Software Development

Software Deployment

Service Operations

Lead Time

Change Failure Rate

Availability

Deployment Frequency

Time to Restore

FOUR KEY METRICS

DORA METRICS EXPLAINED

Lead Time

Lead time is the time it takes to go from a need being recognised and accepted to the need being addressed in production.

Deployment Frequency

Proxy for batch size. The more frequently you deploy the smaller the size of the change. Small batches reduce cycle time, reduce risk and overhead, improve efficiency, increase motivation and urgency and reduce cost and schedule growth.

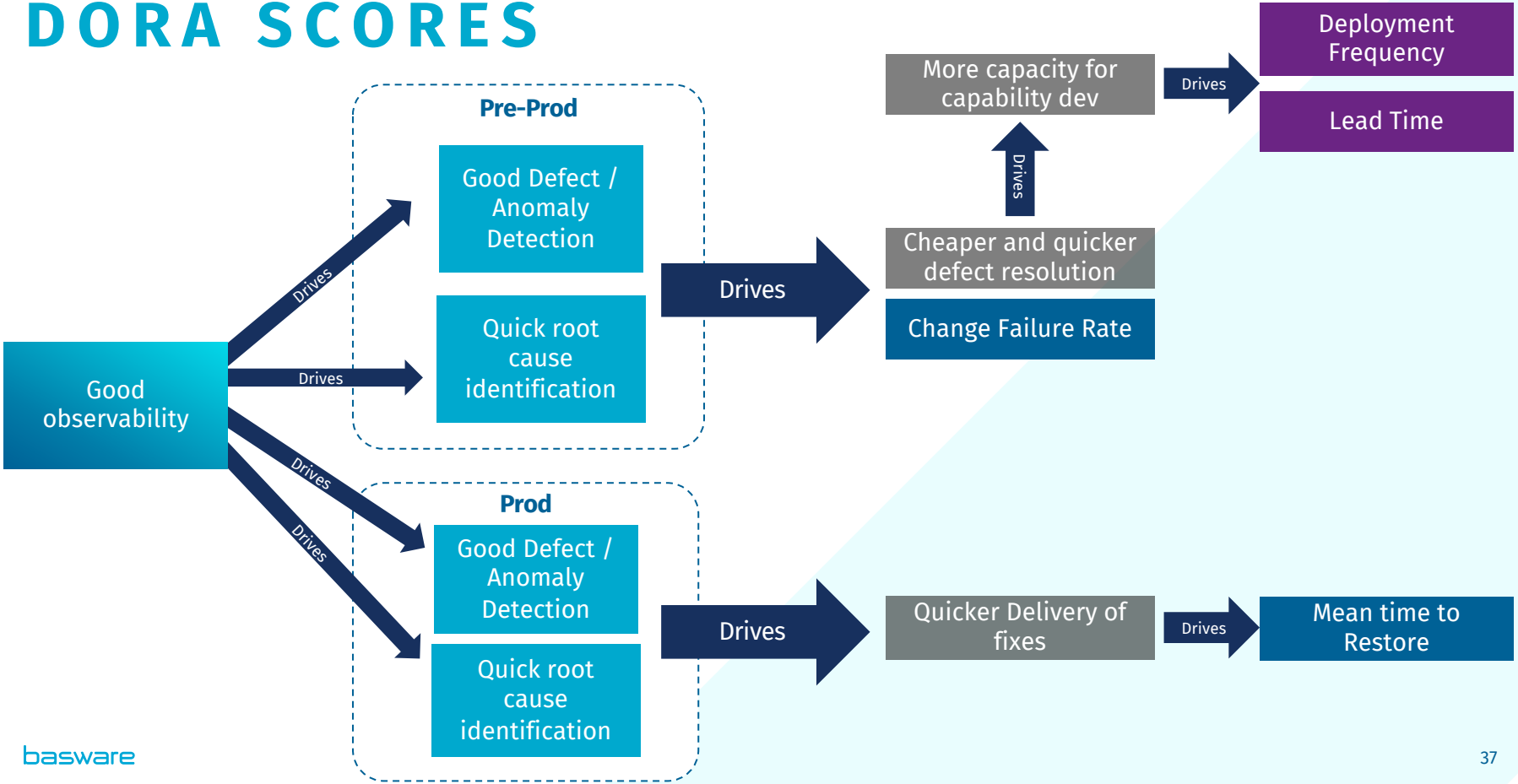
Mean time to restore

Reliability is traditionally measured as the time between failures, but in a modern software organisation failure inevitable. Thus, reliability is measured by how long it takes to restore a service when it fails.

Change failure percentage

This metric looks at the percentage of changes made to production that fail.

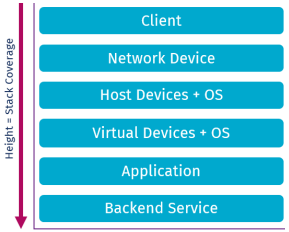
OBSERVABILITIES ROLE IN GOOD DORA SCORES



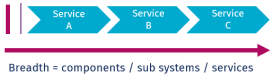
INCREASING OBSERVABILITY THROUGH DYNATRACE

INCREASING TELEMETRY COVERAGE

in dynatrace



- **Height**
  - Use OneAgent for servers / VMs
  - Serverless - Monitor Lambdas / Azure Functions / Google Cloud Functions
  - AWS / Azure / etc Integrations to capture cloud metrics

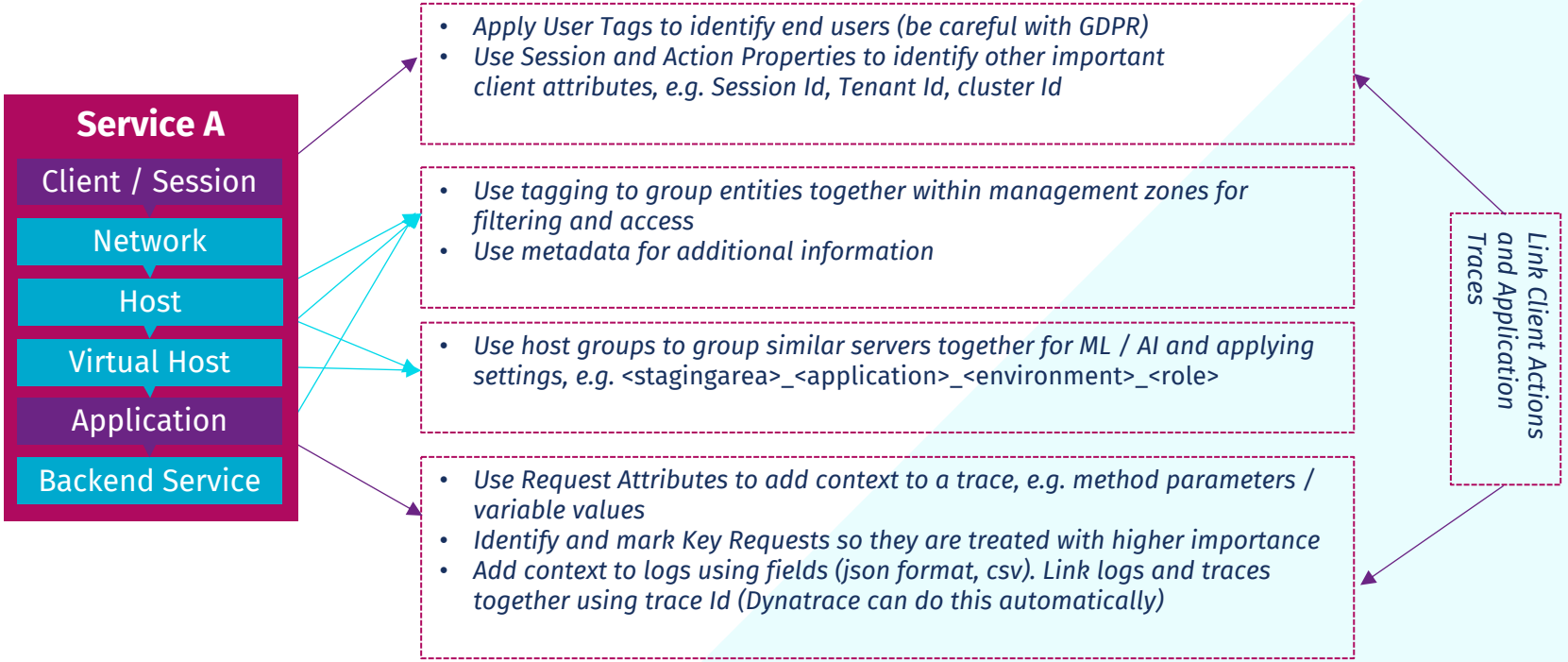


- **Breadth**
  - Monitor everything (or as much as you can) - A Service / microservice may rely on multiple other services. It is hard to identify the actual root cause if data from some services are missing

- **Depth**
  - Add manual instrumentation using OpenTelemetry / OpenTracing to capture spans / details that are not captured automatically
  - Use OneAgent / Activegate plugins or other Ingestion methods to add additional data
  - Use "custom service detection" to monitor services that are not exposed via standard communication technologies
  - Use "calculated service metrics" to capture additional important business or technical metrics

TELEMETRY CONTEXT

in dynatrace



basware
Simplify Operations, Spend Smarter.

THANK YOU