
AWS
re:Invent
NOV. 27 – DEC. 1, 2023 | LAS VEGAS, NV

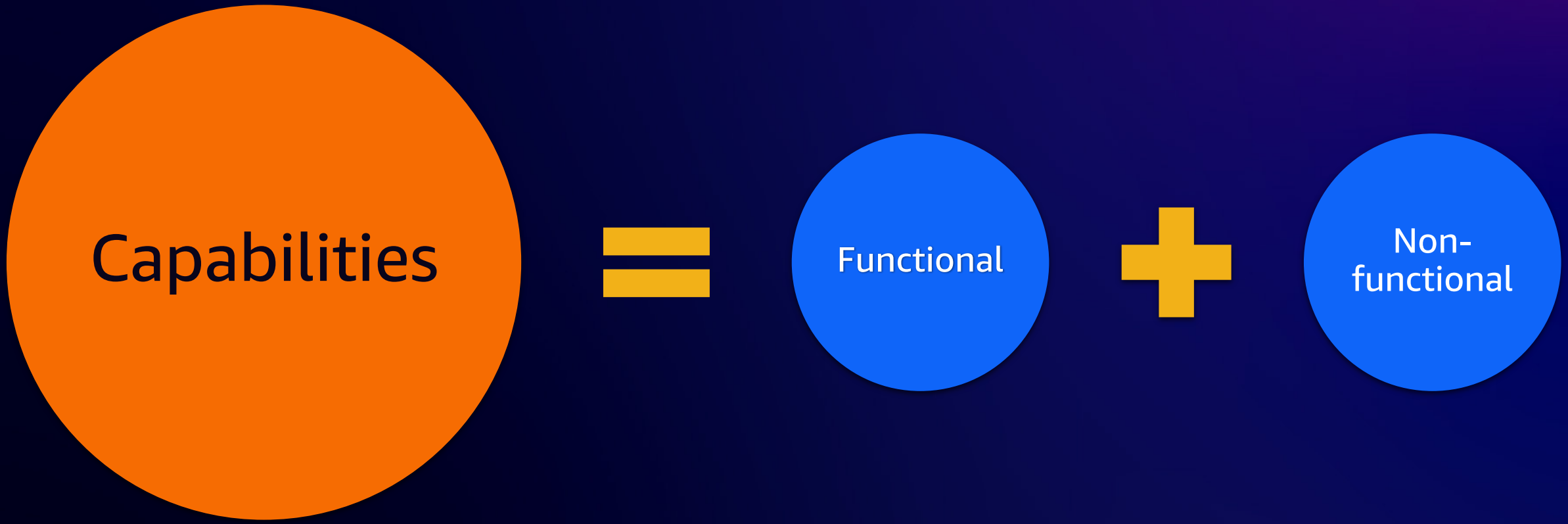
ARC210-R

A capability-oriented approach to defining your cloud architecture

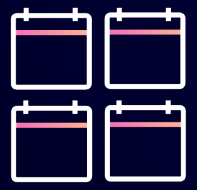
Sushanth Mangalore
Sr. Solutions Architect
Amazon Web Services

Mukund Rao
Sr. Solutions Architect
Amazon Web Services

Thinking in capabilities

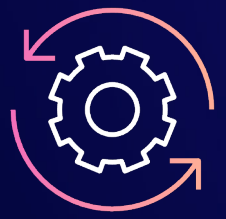


Why start with capabilities?

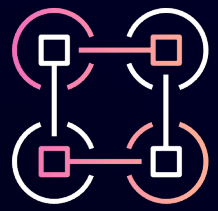


Decouple intent from implementation

Enumerate constraints



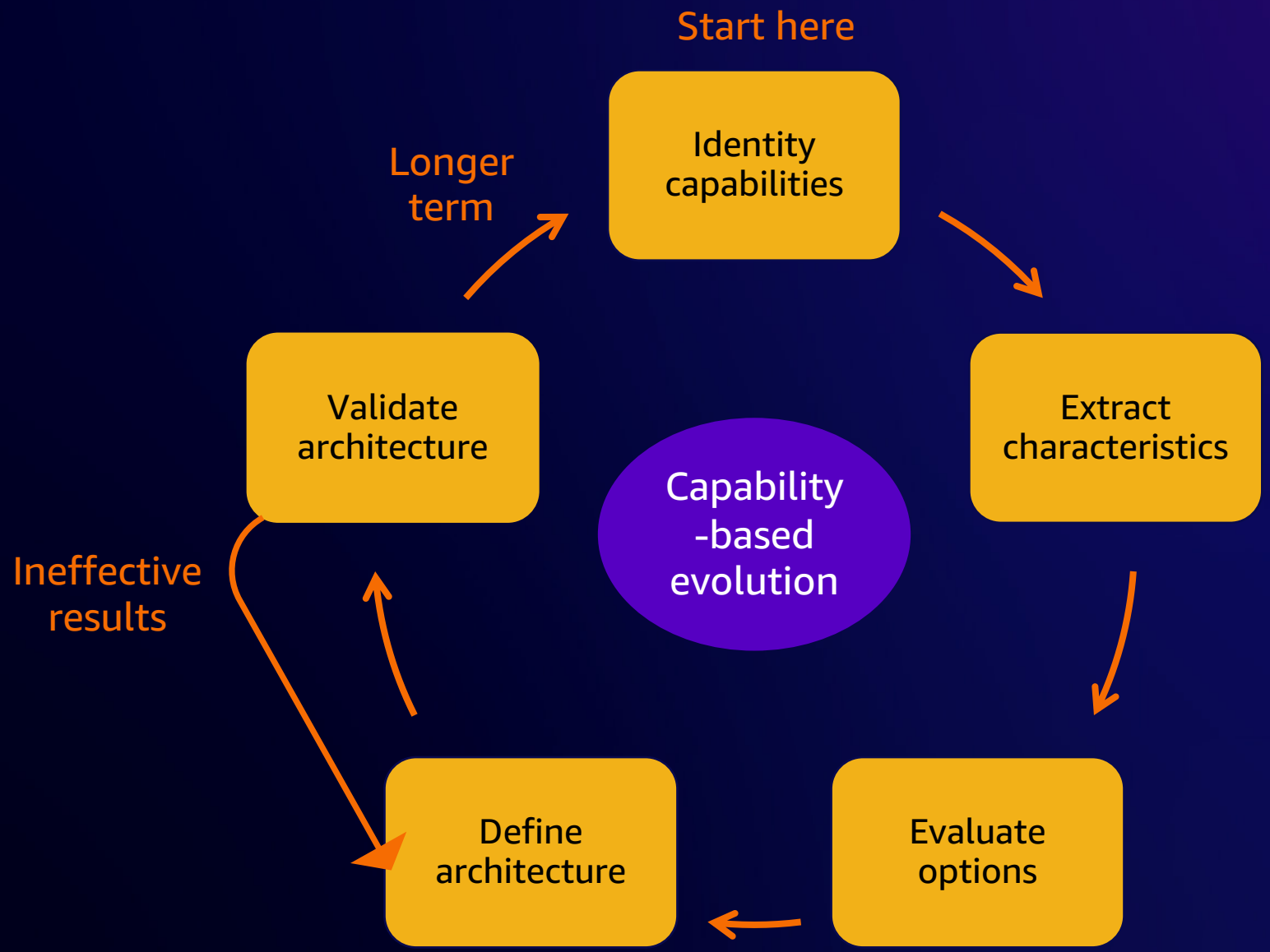
Iterative design

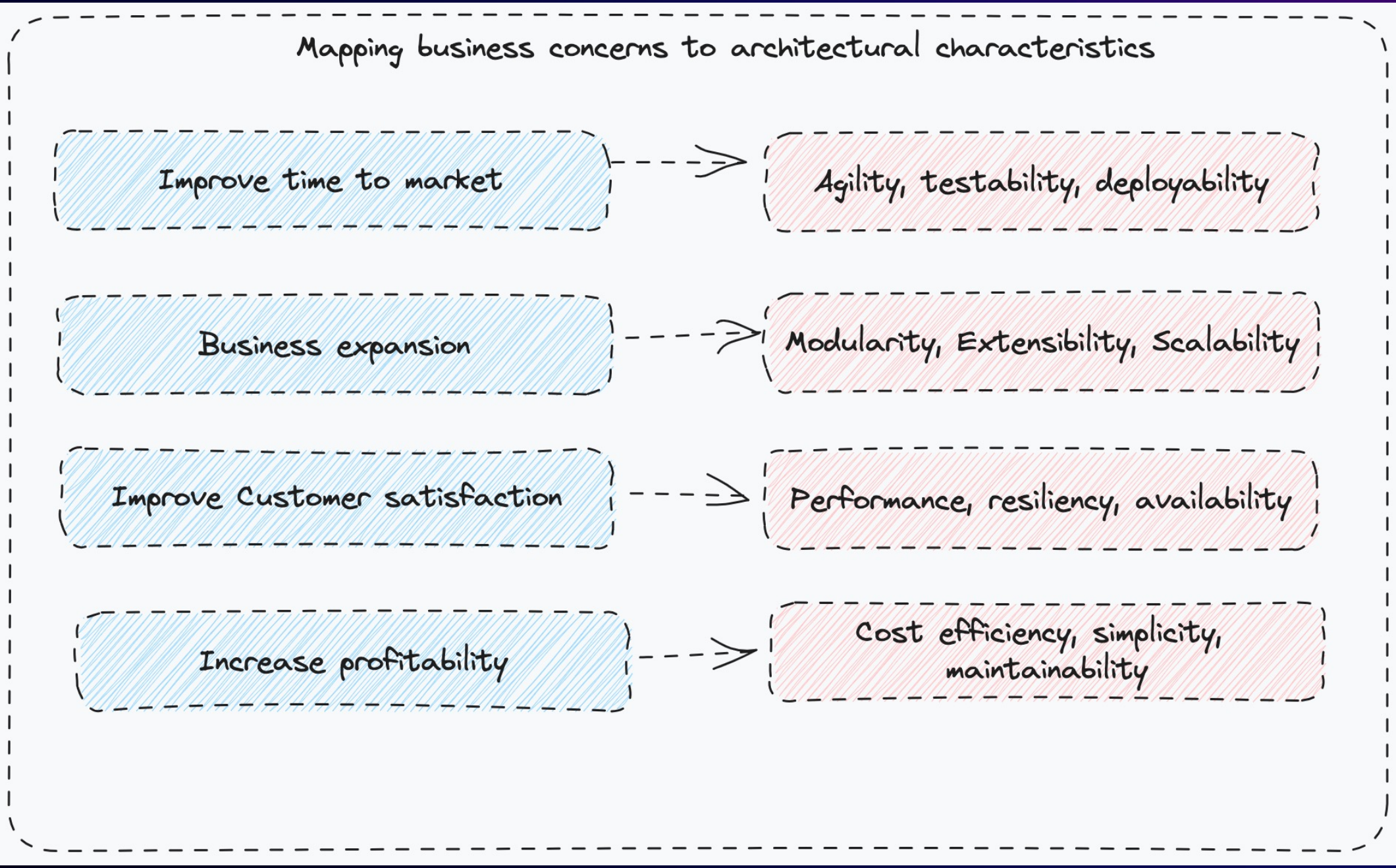


Derive architectural characteristics

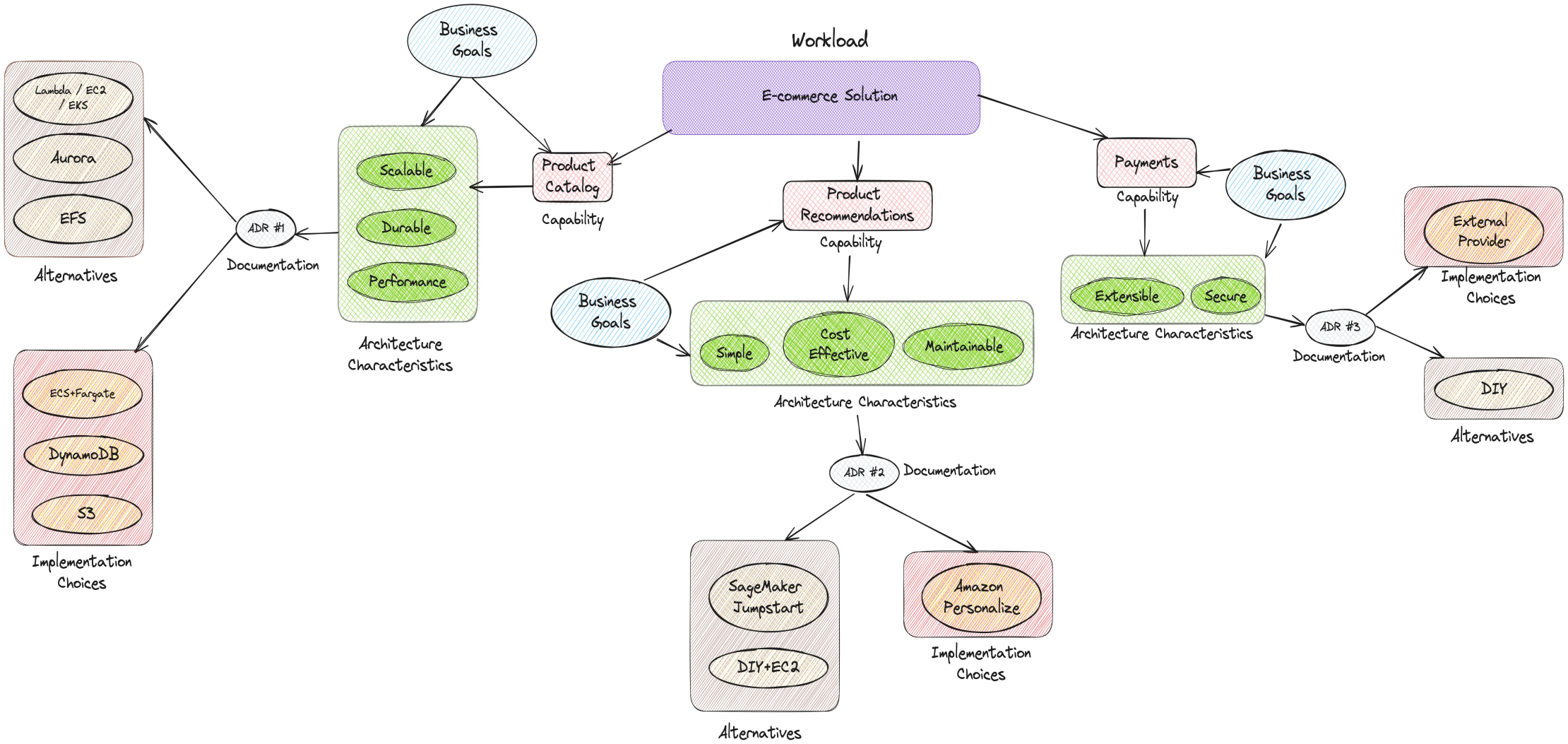
Evaluate your options

Mental Model





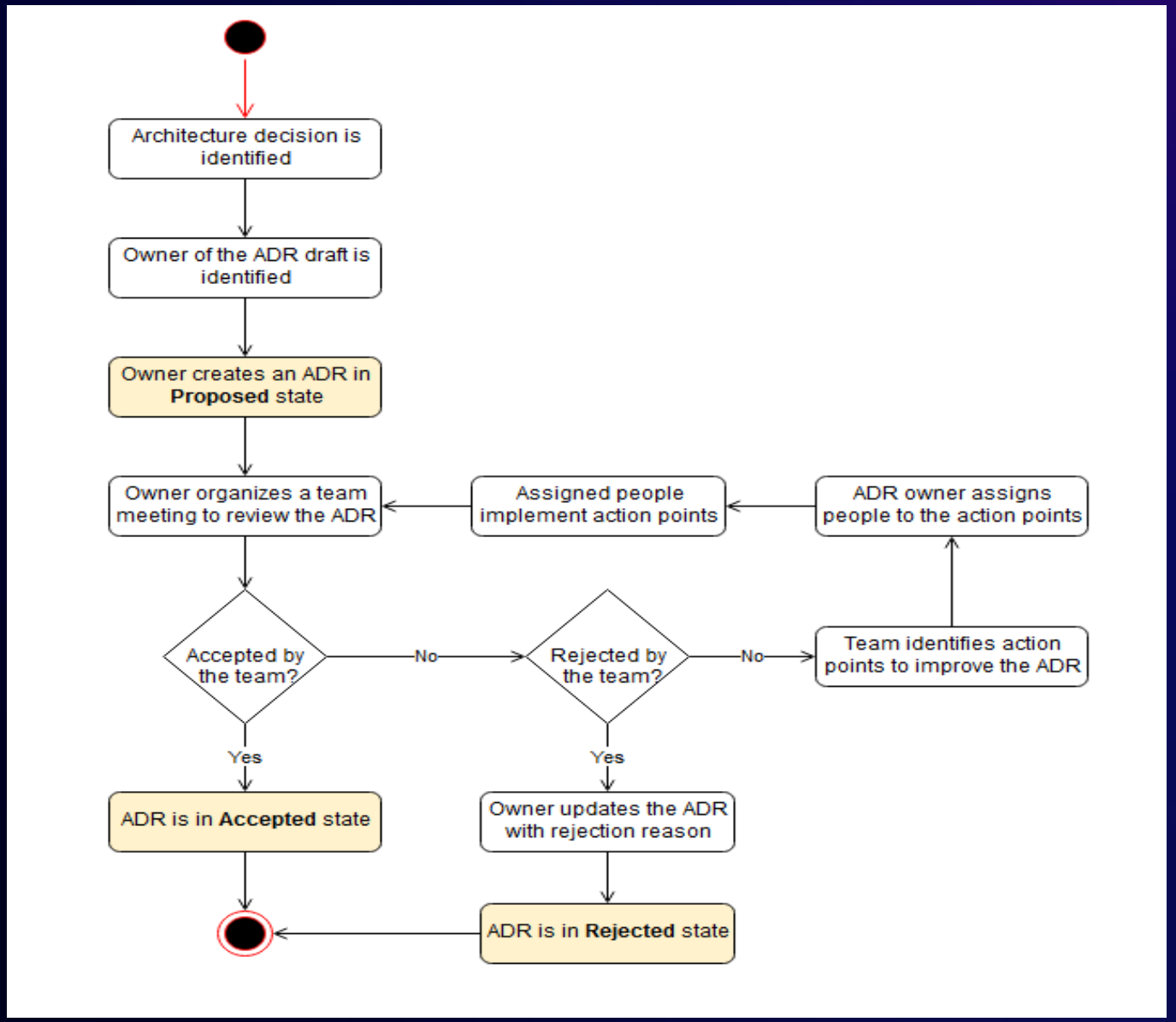
Capability to Architecture Trace



Architectural Decision Records (ADR)

An architectural decision record (ADR) is a document that describes a choice the team makes about a significant aspect of the software architecture they're planning to build. Each ADR describes the architectural decision, its context, and its consequences. ADRs have states and therefore follow a lifecycle.

ADR Process



ADR Template

Title: Architectural Decision Record (ADR #1)

Status: Proposed/ Under Review / Accepted

Context: Problem statement / Alternatives

<Stakeholders>: Sr. Architect, Lead Engineer, Product Manager - Names

Decision:

Consequences: Trade-offs / fall backs / 2-way doors

ADR Sample

Title: ADR #1 - Product Catalog

Status: Accepted

Context: We need to select the Compute, Database and Storage options to implement the Product Catalog that aligns with our business goals.

Stakeholders: Bob - Solutions Architect, Sandy - Lead Engineer, Ravi - Product Manager

Decision: We have decided to select Amazon ECS with Fargate as our compute option, DynamoDB as our database and Amazon S3 as the storage. This is in line with required architectural characteristics of scalability, durability and performance. These characteristics were chosen based on our business goals of rapid customer growth through wide vareity of product selection and excellent user experience at all times, including periodic flash sales

Alternatives Considered: EKS/EC2/Lambda - We expect consistent site usage, so Lambdas may not suit. We do not have Kubernetes expertise in-house, so we ruled out EKS. We prefer to use containers for better resource utilization, portability and dependency management, so we ruled out deploying on EC2 directly. With ECS also, Fargate was preferred for easier maintenance.
Aurora - Ruled out as DynamoDB will scale better for our needs and we want to allow flexible schema for products.
EFS - Ruled out as S3 offers better durability and is cost effective for static assets

Consequences: Using Fargate will mean that we will have always-on compute, but we will use autoscaling to scale out and scale in based on the demand. DynamoDB means we will need to ensure we understand our data access patterns well, so we can design our tables and indices for efficient and performant querying. Picking S3 means we are expecting product catalog to only need static assets. In the future, if we require file-system type of access, we may need to consider other options like EFS.