slalom_build

Reference Architectures

Introducing BlackSlope

The Origination

Reference architectures are useful to deliver reliable code consistently. They consist of a prescribed pattern for laying out an N-tier software architecture that a team can easily learn to adopt. As a group, we started to work towards an architecture which is:

- Simple
- Portable
- Maintainable
- Testable

The goal is to rapidly speed up development of new projects or, additions to existing projects without compromising the code quality and reliability.

BlackSlope Advantages

Using BlackSlope as reference architecture has some benefits out of the box. It provides guidelines regarding component design, naming conventions, folder structure, and a lot more.

Reliable solution

BlackSlope proposed architecture has been successfully utilized in implementing solid software solutions for Slalom's clients.

Speed up development

By having a collection of ready to use tools and features, BlackSlope improves development speed. Using it reduces time and effort to implement a solution and as a result the cost of development.

Flexibility and modularity

BlackSlope is designed in a way that makes is easy to replace any component or plug in new one as needed.

Cross-Platform

BlackSlope is targeting ASP.NET Core which is a Cross-Platform framework.

Latest .NET version

BlackSlope is getting updated according to the latest version of .NET 5.

A Reference Architecture

Slalom uses an internally developed reference architecture called BlackSlope which was created from experience in delivering successful products.

It focuses on highly scalable API layers and includes standard patterns for modularity and extensibility.

Slalom released this reference architecture to the public and it is being maintained by internal and the opensource communities. This allows us to improve quality and performance as well as adopt the latest updates and technologies.

Multi-tier Architecture

BlackSlope is providing a baseline for a reference architecture to implement an API app. Each layer of an API application is equipped with a set of components to implement required functionalities for that layer. Even though each component plays a specific role it still could be utilized at multiple layers of the application.

Operation

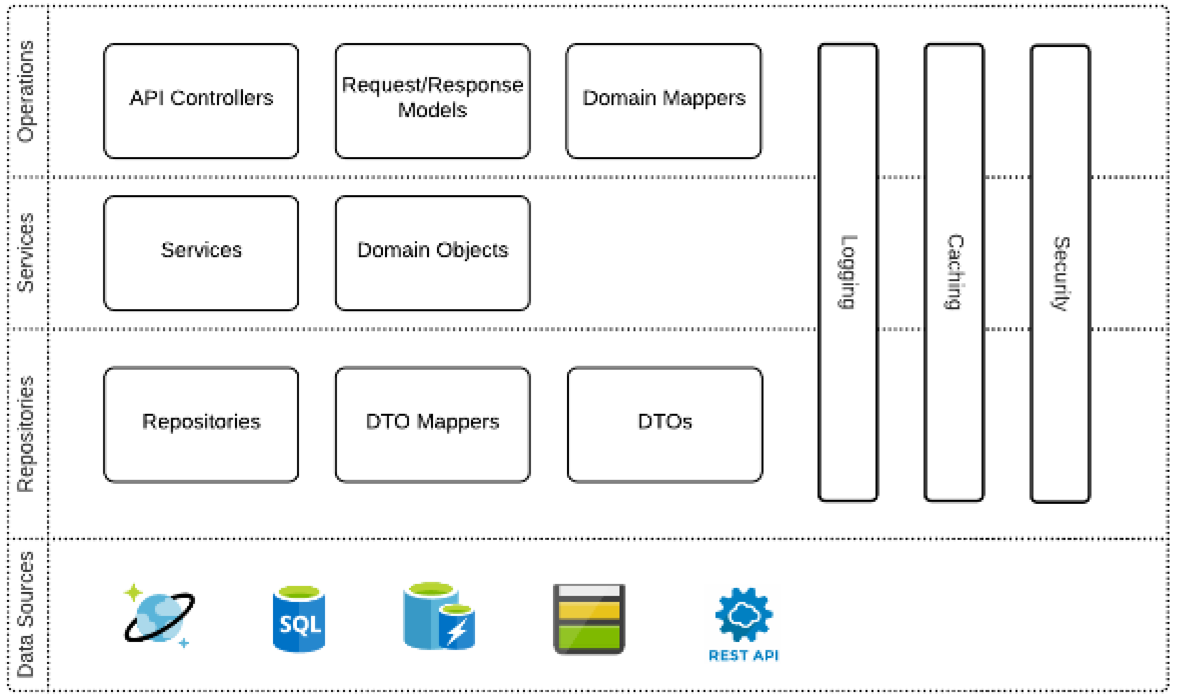
This layer contains the controllers and Request/View Models. Also, the validation process for requests should happen at this layer. BlackSlope validator uses Fluent Validation underneath but like any other components you could plug in your preferred validation tool.

Services

This layer handles the business and domain logic. It contains Domain Models and knows about repository layer and DTOs . When a service needs to access data layer it uses a repository.

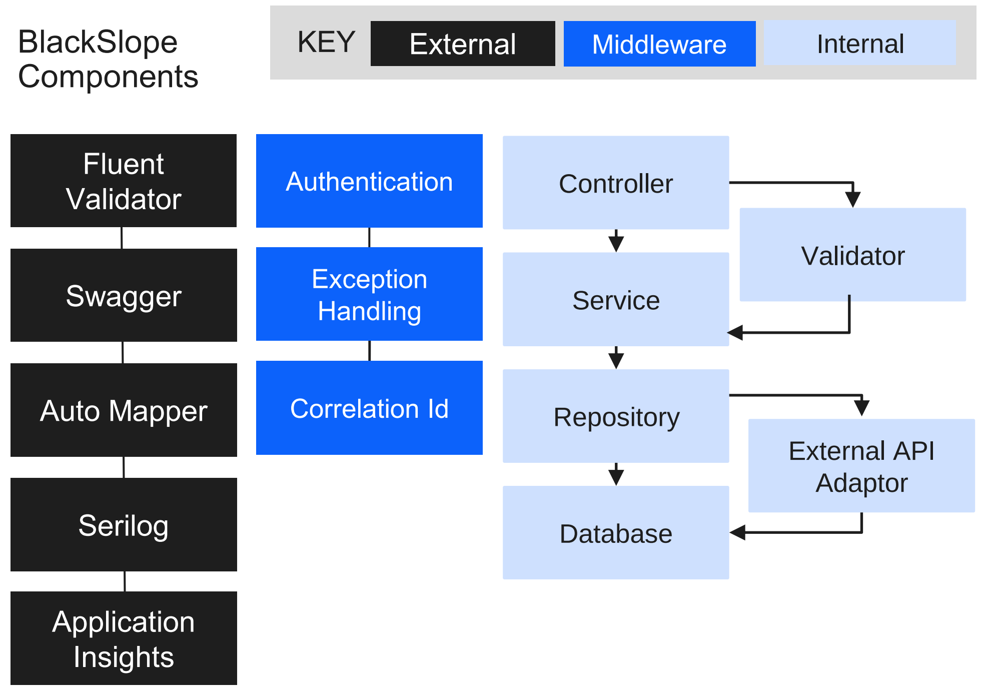
Repositories

Repositories can be grouped into two major categories. One type provides access to data storage, and the other type manages the calls to external APIs and services which are not part of the current application.



BlackSlope Components

BlackSlope utilizes some internal and some third-party packages to handle specific responsibilities. Each of these packages can be customized through configuration settings.



Code Quality

It is important to have a clean codebase. It helps in maintainability of the application as well as future improvement. Code analyzers could play important role in automating some level of quality check.

Microsoft .NET analyzers

BlackSlope utilize Microsoft .NET analyzers to check code for security and performance issues.

StyleCop analyzers

Consistency in code style is always a challenge BlackSlope utilize StyleCop to automate some level of style checks.

CI/CD integration

Code analysis could be done as part of the build process and being enforced through the CI/CD pipeline.

Defining the 'next generation' tech stack

Define next generation technology Stack

- Create a holistic architectural vision
- Review industry and community standards for libraries and toolsets
- Review DevSecOps methodologies and CI/CD pipelines
- Invest in POCs to validate choices

Abstract tightly coupled architectures into layers that can be evolved independently

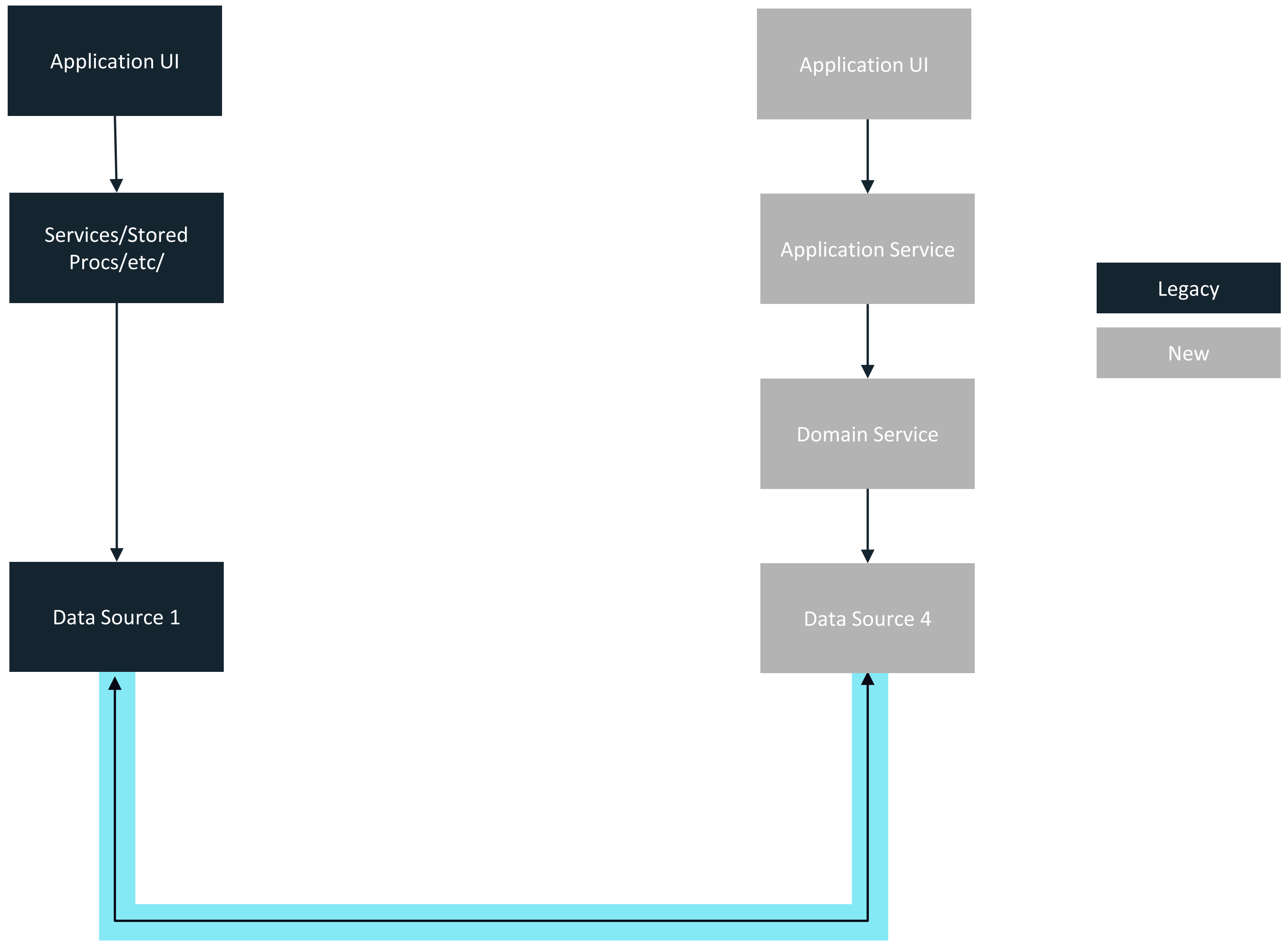
- Anti-Corruption Layers: https://docs.microsoft.com/en-us/azure/architecture/patterns/anti-corruption-layer
- Use Doman Driven methods to define bounded contexts that can evolve independently

Integrate new applications to existing ecosystems to replace legacy codebases over time

- Strangler pattern: https://martinfowler.com/bliki/StranglerFigApplication.html
- New UI for existing data sources (Facades and Adapters)
- One way back fed data for legacy read only support
- Two way data synchronization for legacy read/write support

Sample Strangler application 1:

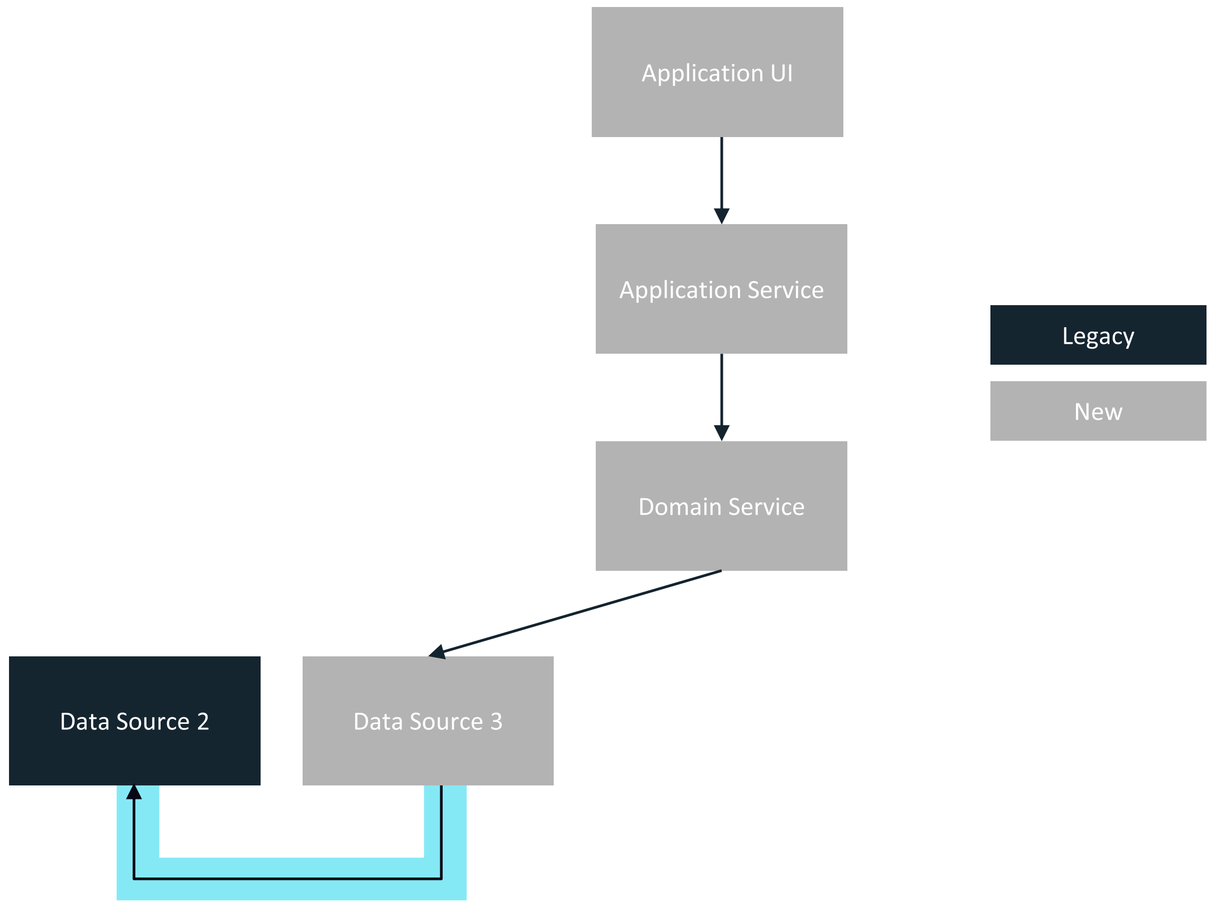
Legacy Read/Write Support



Two-way data sync for read-write legacy support

Sample Strangler application 2:

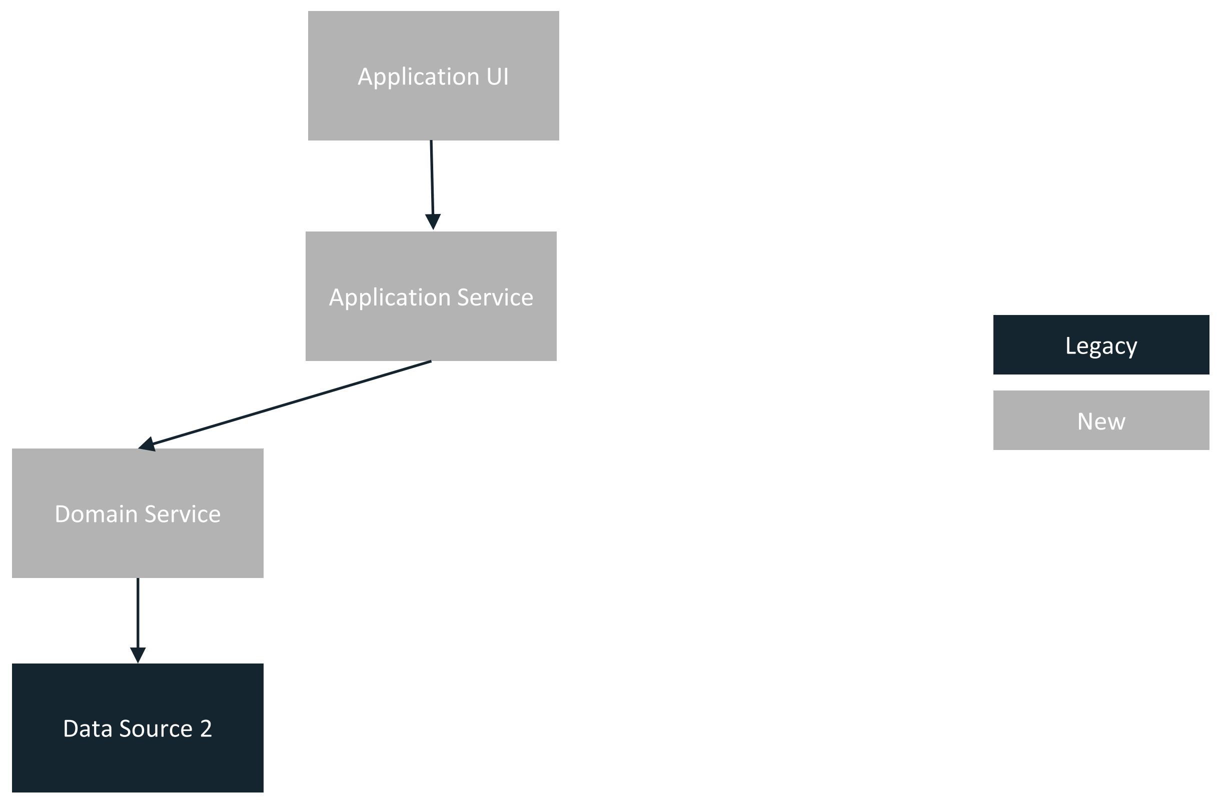
Read-Only Legacy Support



One-way data back feed for read only legacy support

Facades:

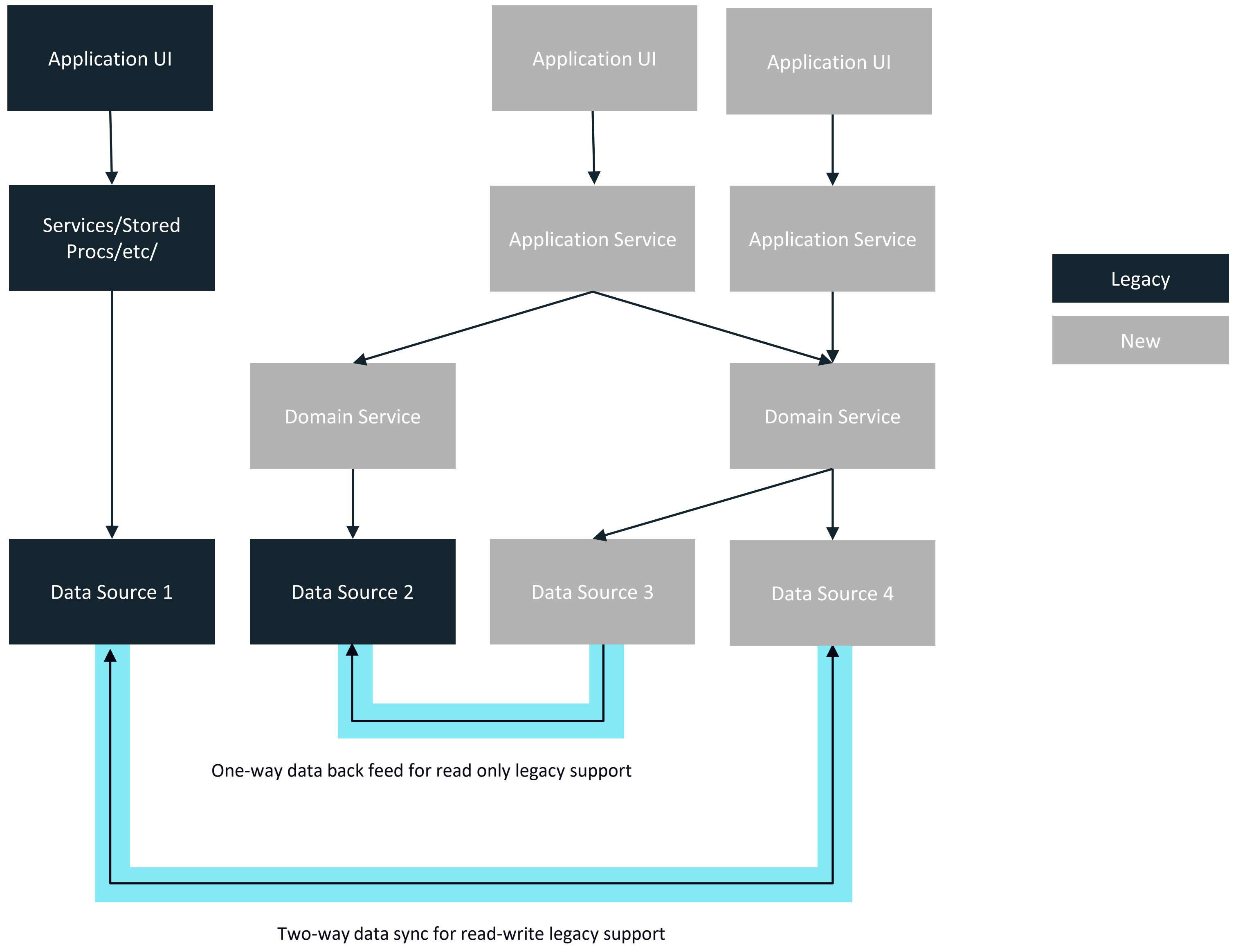
New consumers of existing logic as proofs of concepts



Sample Strangler ecosystem:

Combination based on applications individual needs

"We never told the users that they must use the new system. Nor did we remove access to the old system. We relied on making the system so compelling that there was no reason to use the old. This also meant that we stayed focused on the users real requirements"



slalombuild.com