



Robert Haken

software & cloud architect, HAVIT, s.r.o.

haken@havit.cz, @RobertHaken, <https://knowledge-base.havit.cz> + .eu

Microsoft MVP: Development, MCT, MCPD: Web, MCSE: Cloud

Cloud Design Patterns

Cloud Challenges

Availability

Data Management

Design and Implementation

Messaging

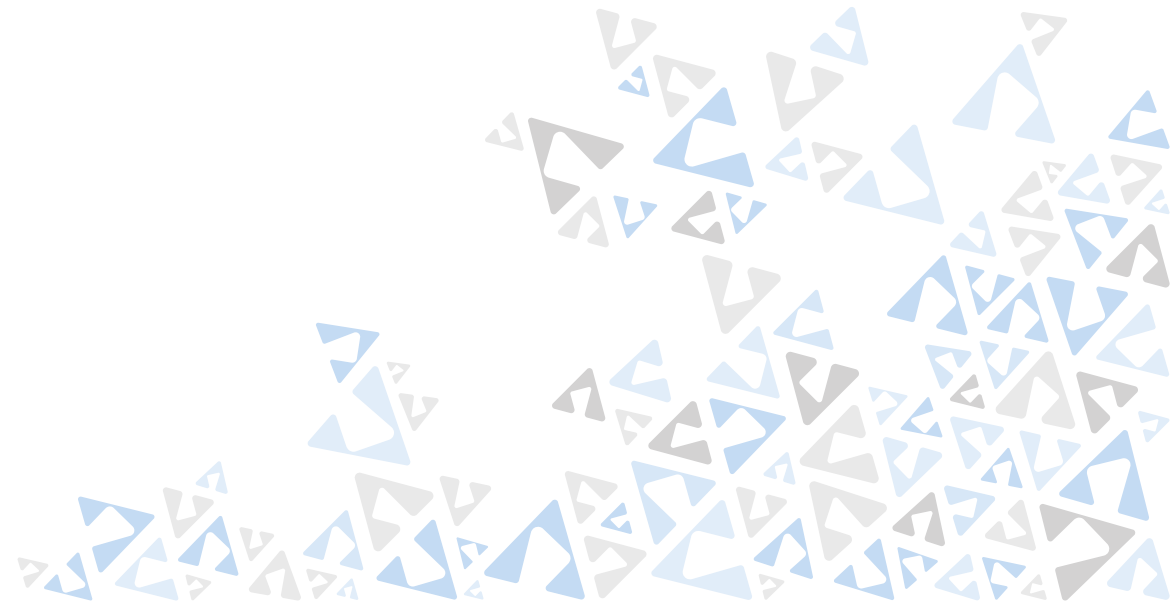
Management and Monitoring

Performance and Scalability

Resiliency

Security

Credits: <https://docs.microsoft.com/en-us/azure/architecture/patterns/>



Management and Monitoring patterns

Ambassador

Anti-Corruption Layer

External Configuration Store

Gateway Aggregation

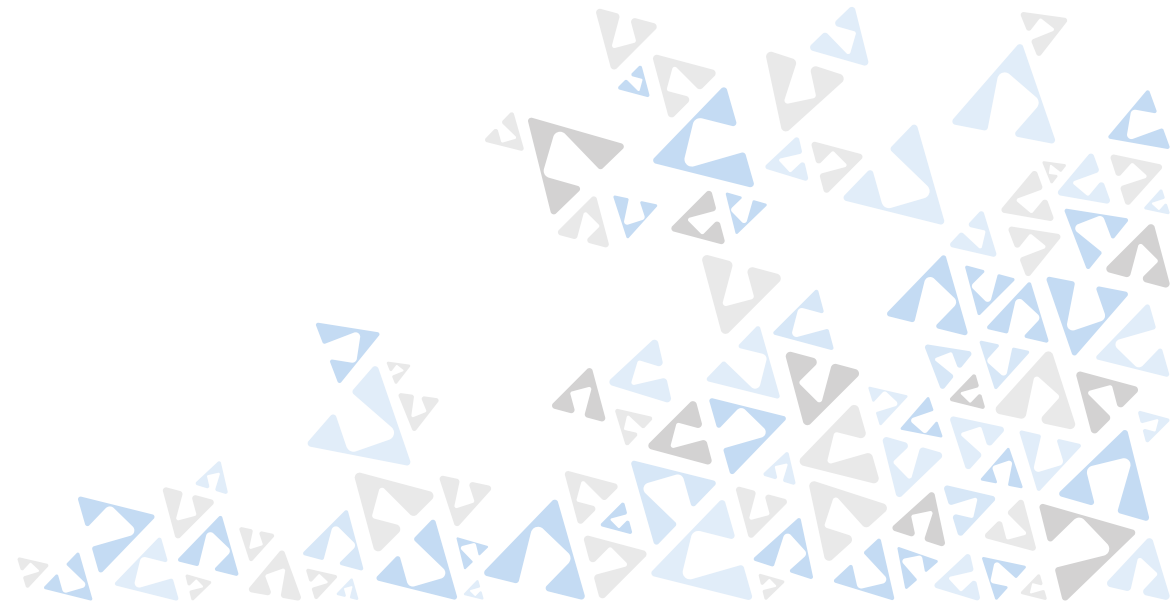
Gateway Offloading

Gateway Routing

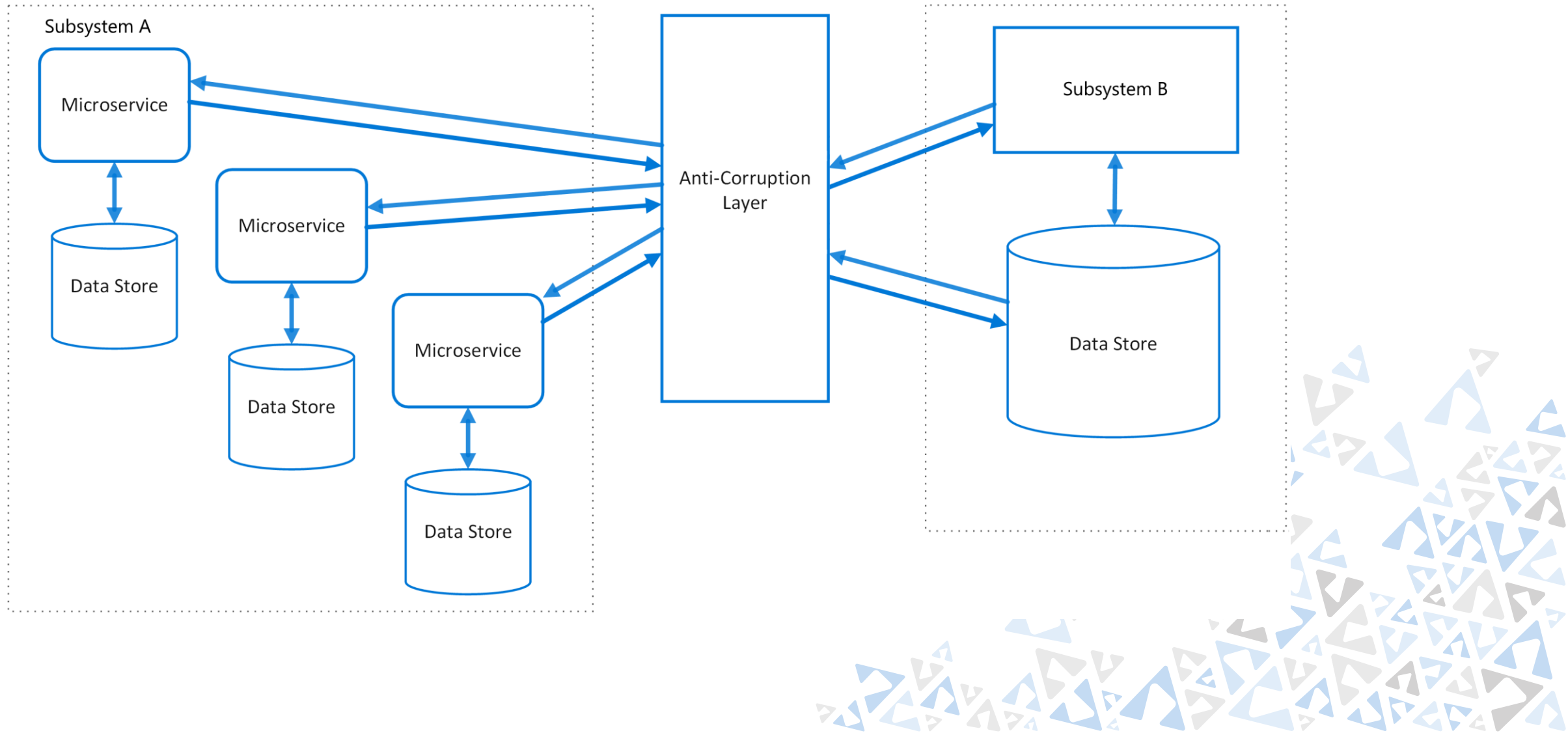
Health Endpoint Monitoring

Sidecar

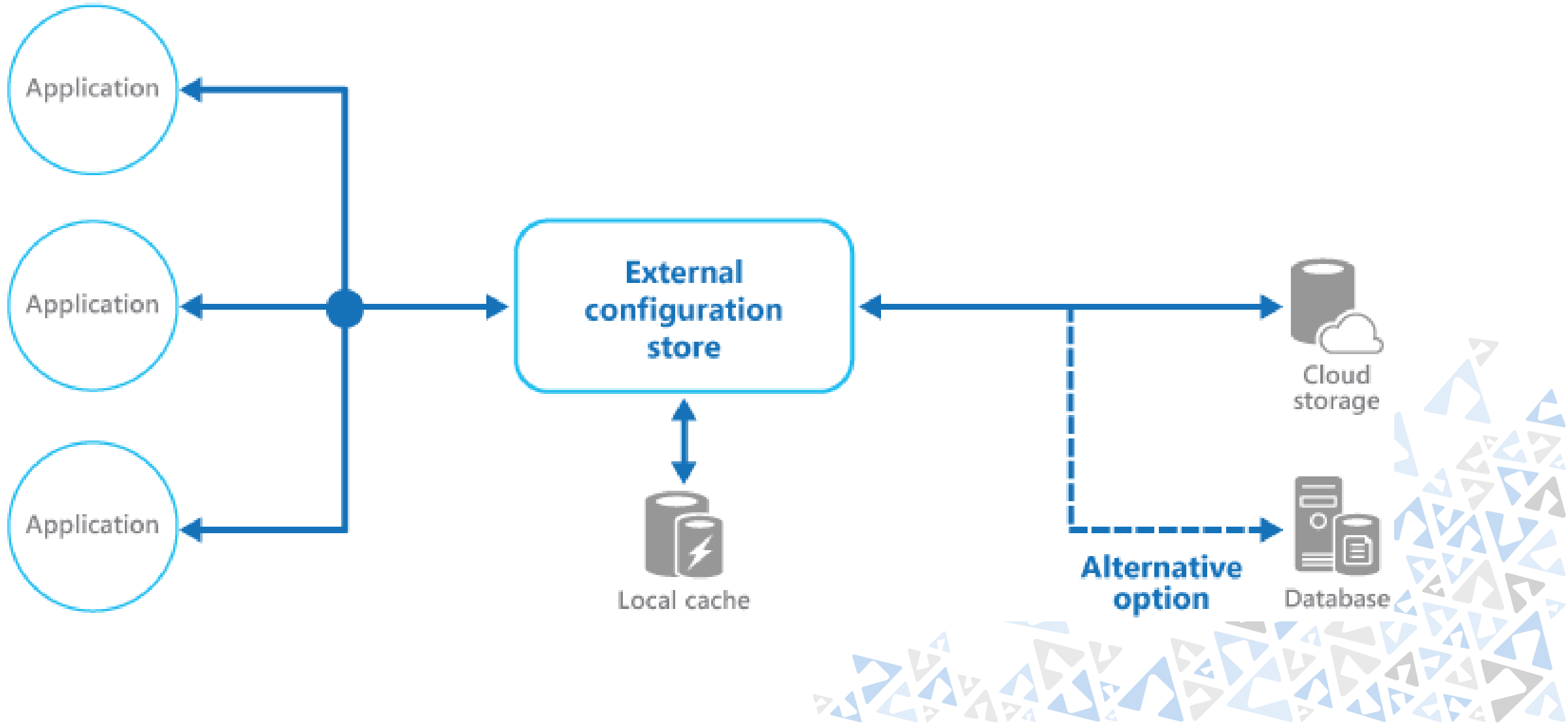
Strangler



Anti-Corruption Layer



External Configuration Store



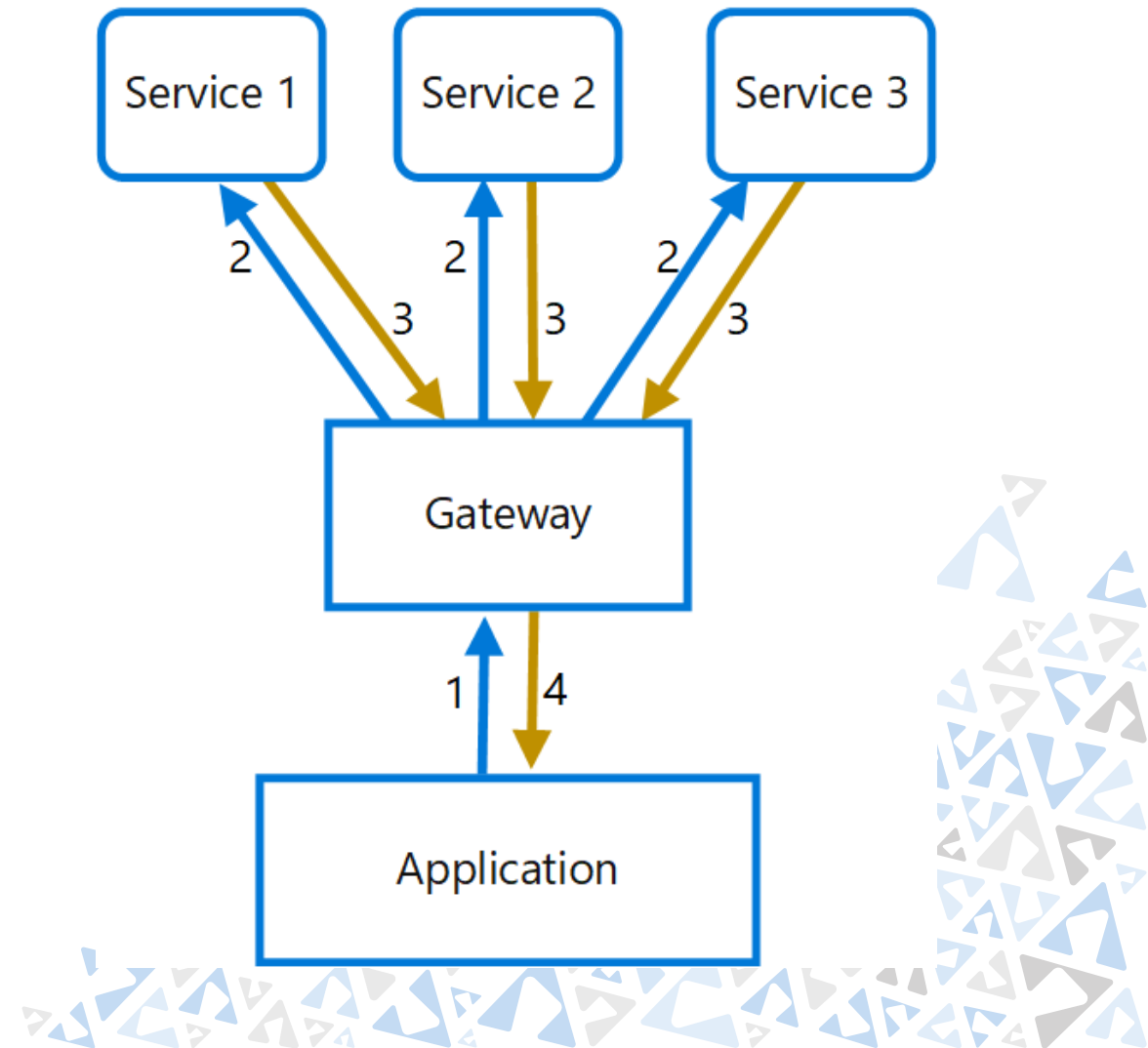
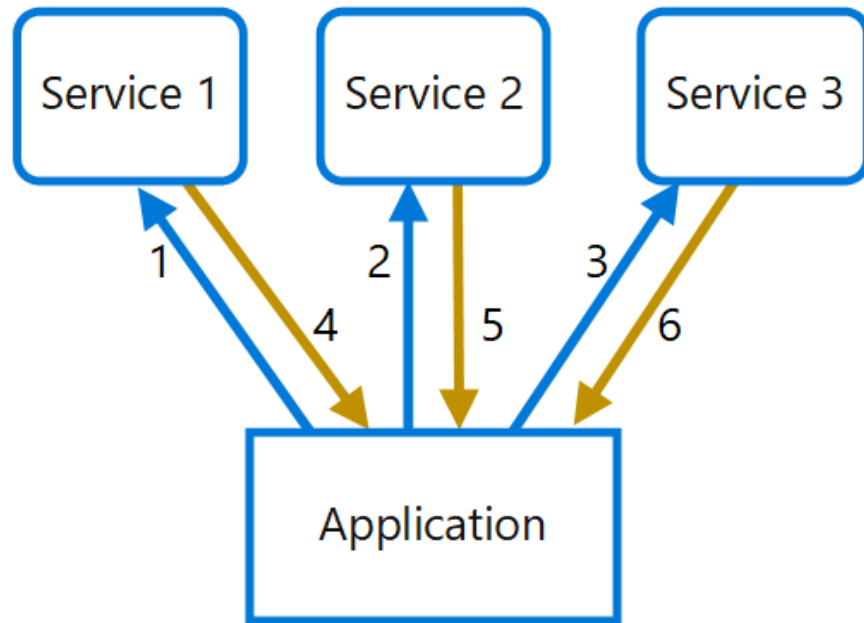
.NET 4.7.1 - ConfigurationBuilders

```
<configBuilders>
  <builders>
    <add name="KeyVault" mode="Strict" prefix="conn_" stripPrefix="true"
        clientId="MyId" clientSecret="mySecret" vaultName="MyVault"
        type="Microsoft.Configuration.ConfigurationBuilders.AzureKeyVaultConfigBuilder, ..." />
    <add name="MyOtherConfigBuilder" type="CustomConfigBuilders.MyOtherConfigBuilder, ..." />
  </builders>
</configBuilders>
<appSettings configBuilders="KeyVault,MyOtherConfigBuilder">
  <add key="Setting1" value="May Be Replaced" />
  <add key="Setting2" value="May Be Removed" />
  <!-- Setting3 could be added by a builder without even being declared here. -->
</appSettings>
```

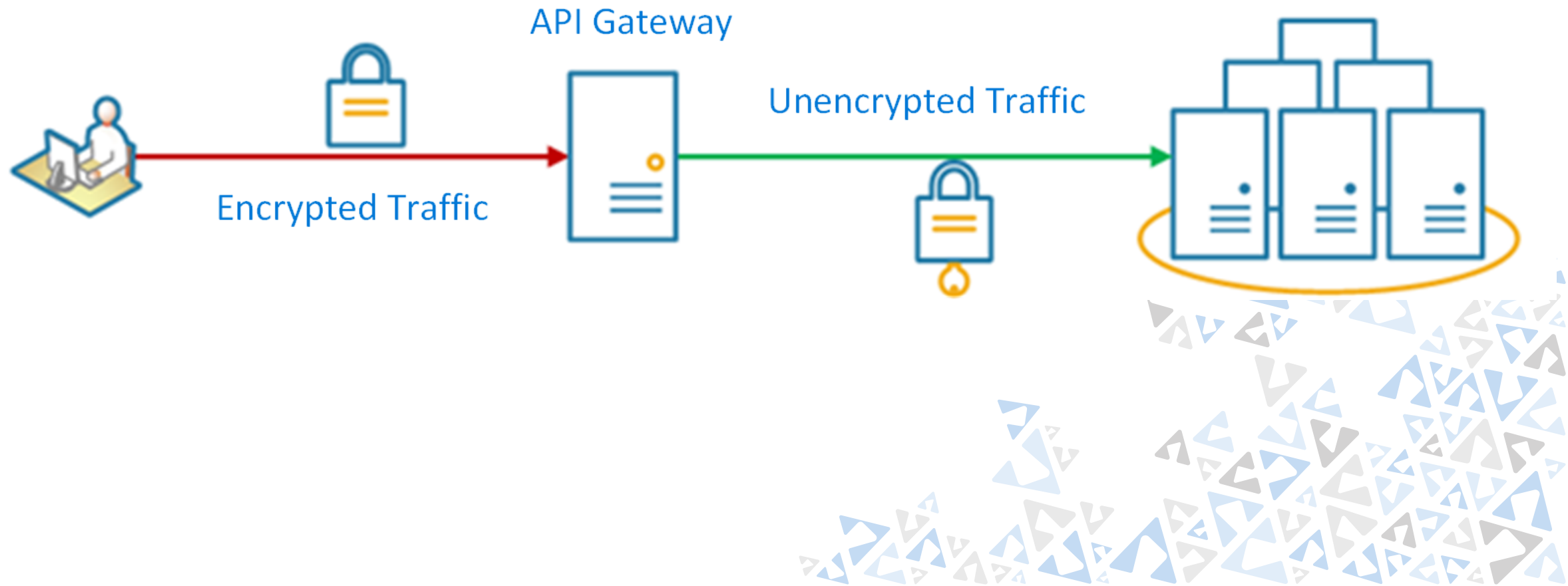
- [EnvironmentConfigBuilder](#) – Read from environment variables
- [AzureKeyVaultConfigBuilder](#) – Read from Azure Key Vault
- [UserSecretsConfigBuilder](#) – Read from a usersecrets file on disk
- [SimpleJsonConfigBuilder](#) – Read from a JSON file



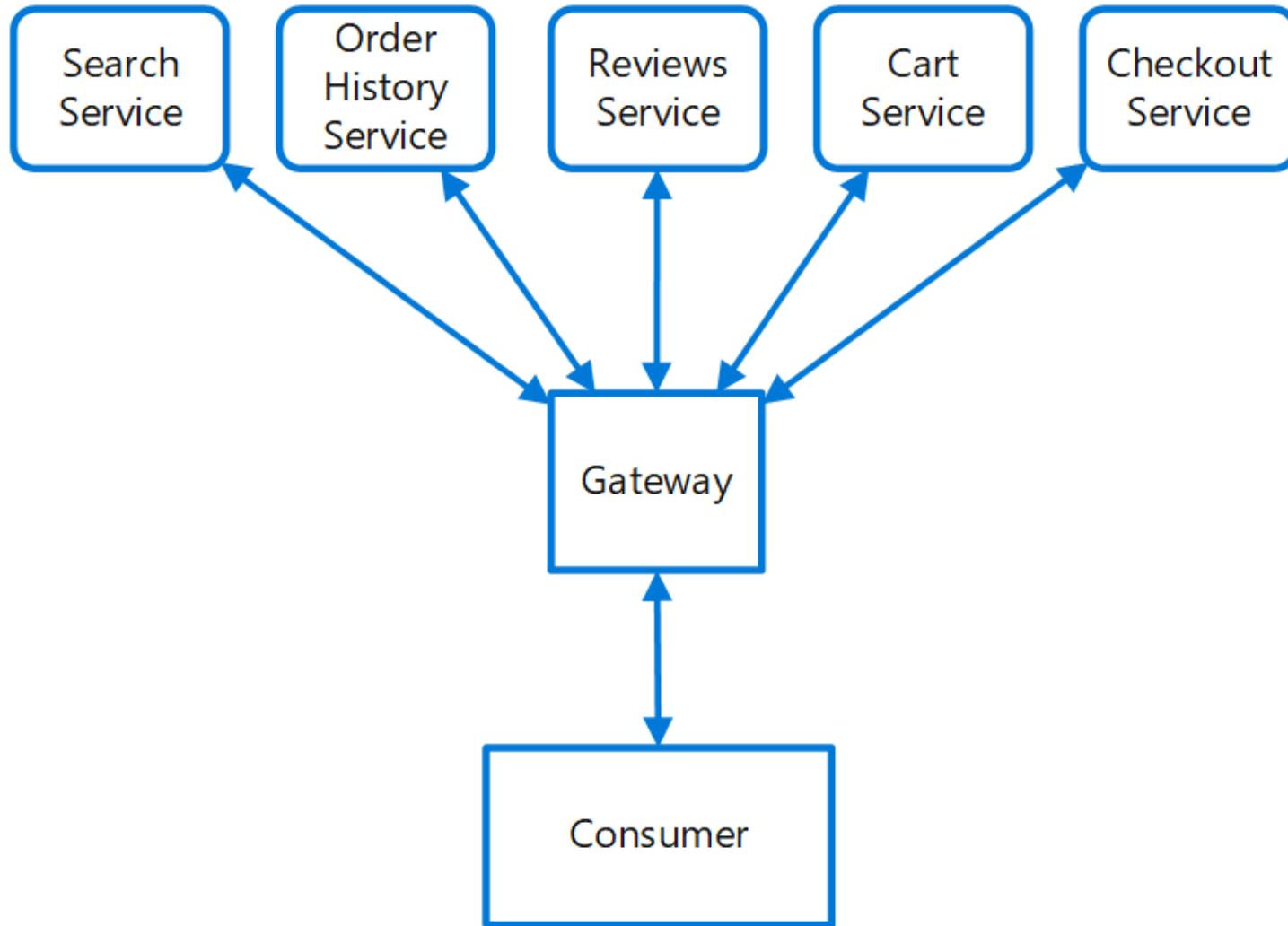
Gateway Aggregation



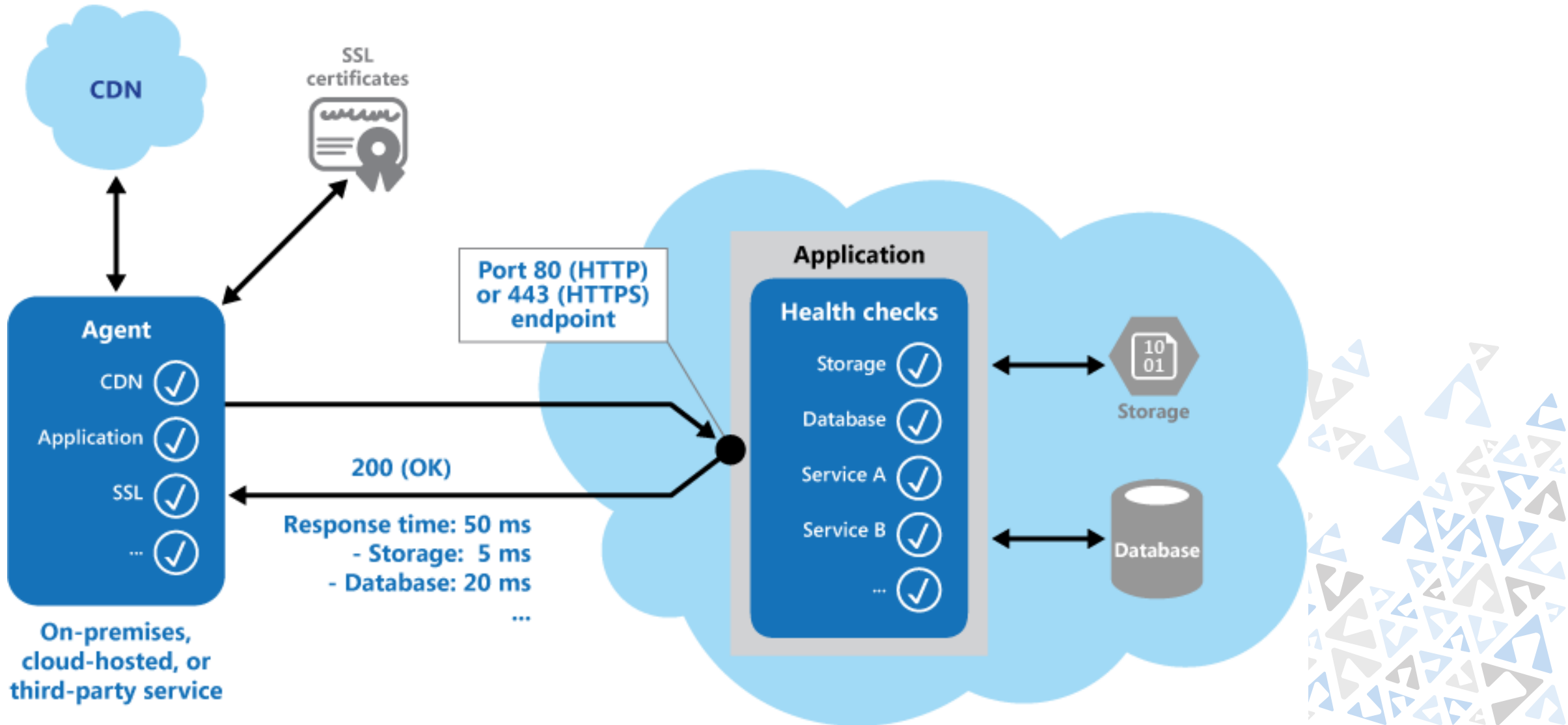
Gateway Offloading




Gateway Routing



Health Endpoint Monitoring



Azure Traffic Manager

 AACSDPFrontendWebTM - Configuration ✦ ✕

Traffic Manager profile

<< Save ✕ Discard

Routing method ⓘ

* DNS time to live (TTL) ⓘ
 seconds

Endpoint monitor settings ⓘ

Protocol

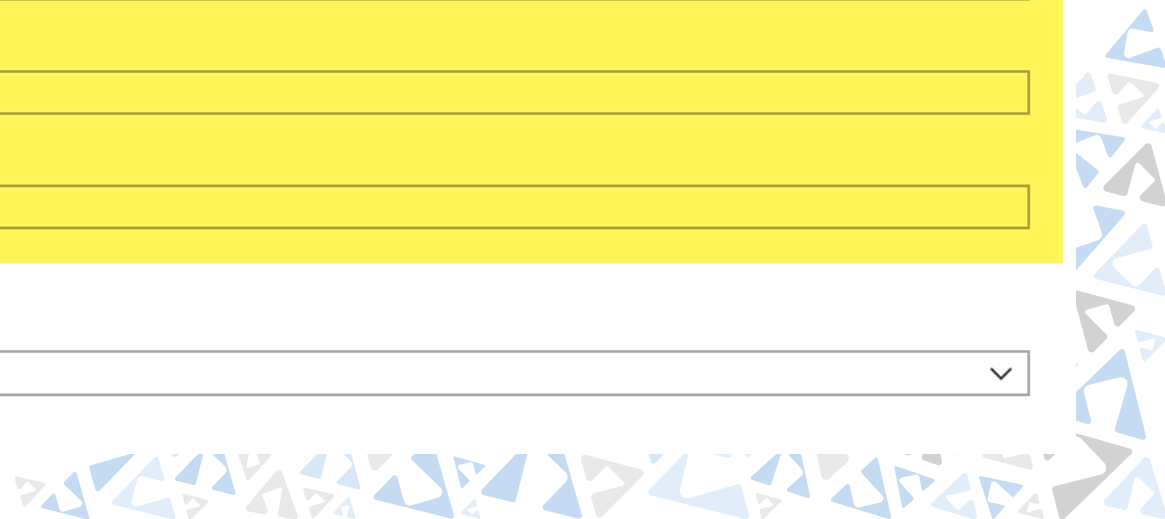
* Port

* Path

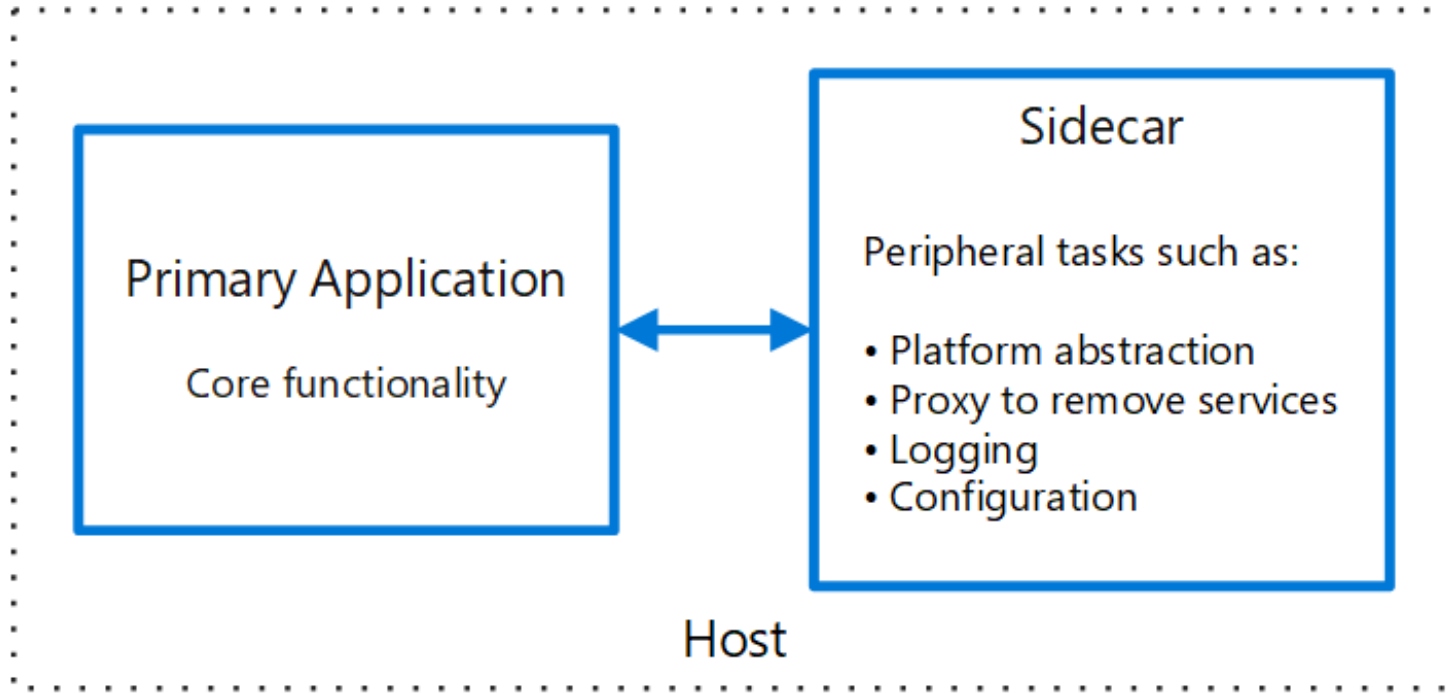
Fast endpoint failover settings

Probing interval ⓘ

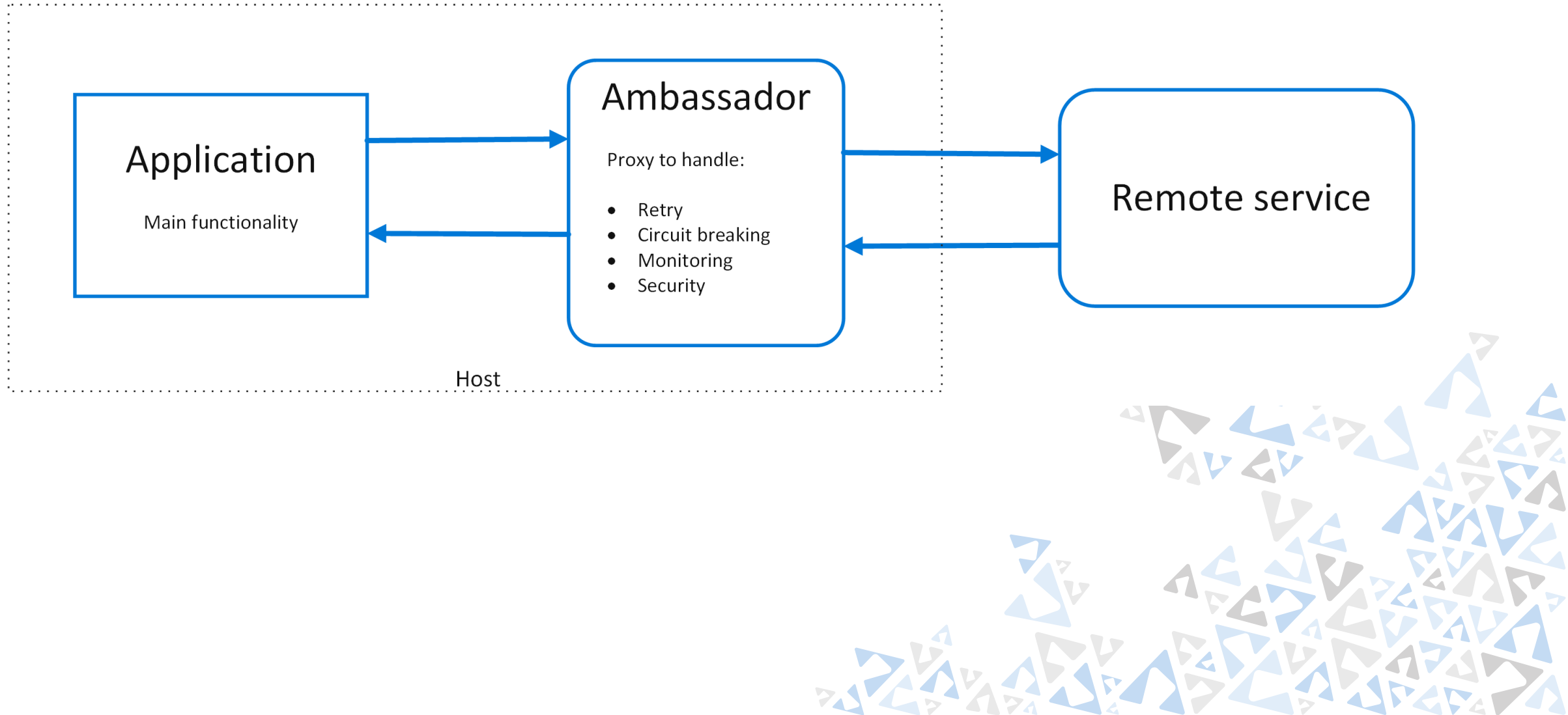
* Tolerated number of failures ⓘ



Sidecar



Ambassador



Performance and Scalability

Cache-Aside

CQRS

Event Sourcing

Index Table

Materialized View

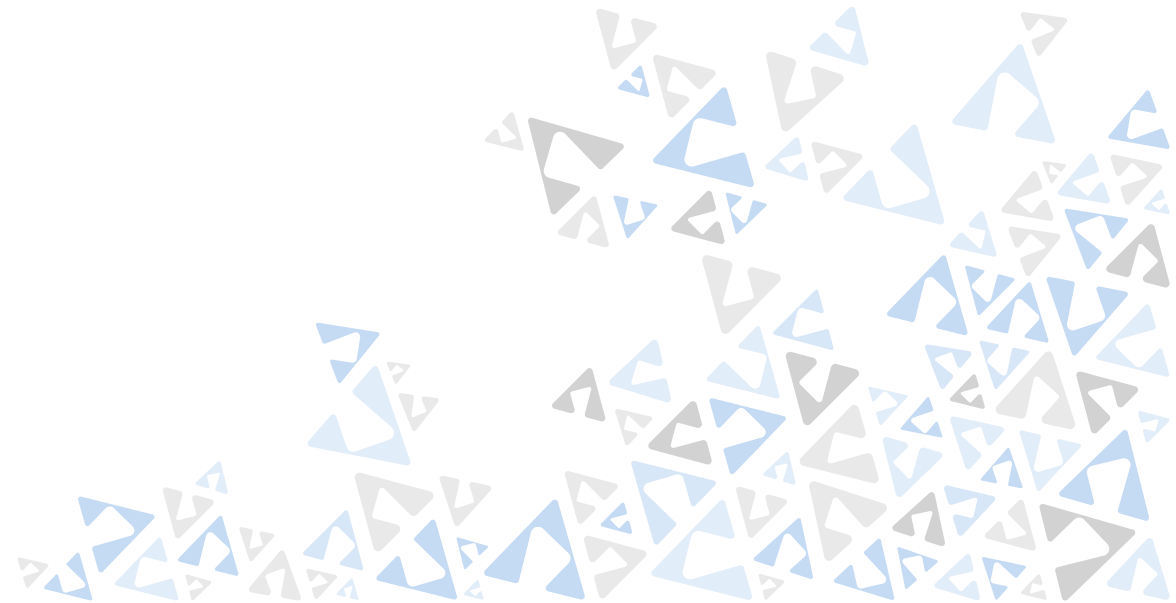
Priority Queue

Queue-Based Load Leveling

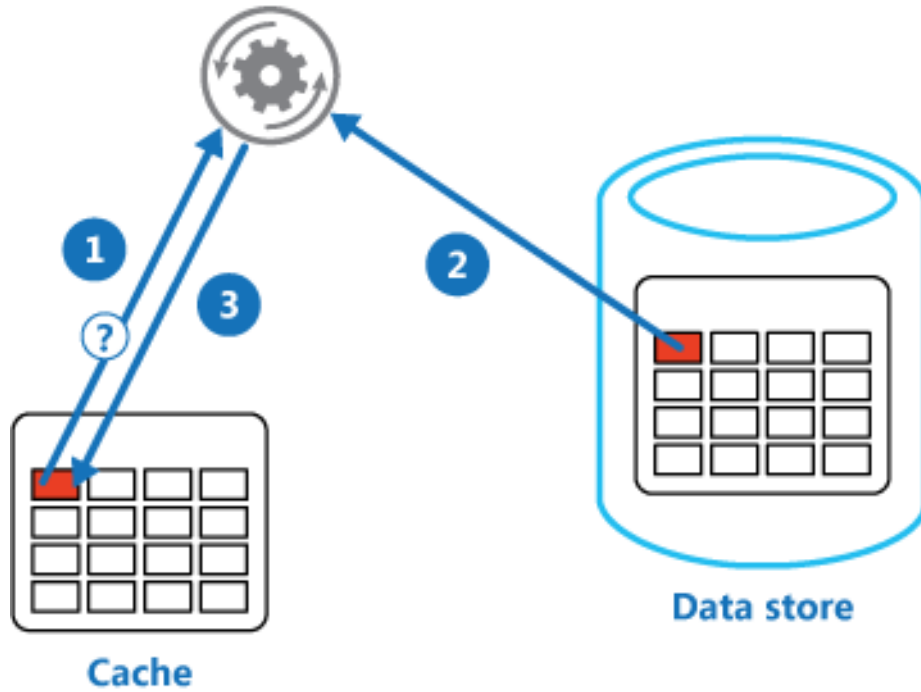
Sharding

Static Content Hosting

Throttling



Cache-Aside



- 1: Determine whether the item is currently held in the cache.
- 2: If the item is not currently in the cache, read the item from the data store.
- 3: Store a copy of the item in the cache.



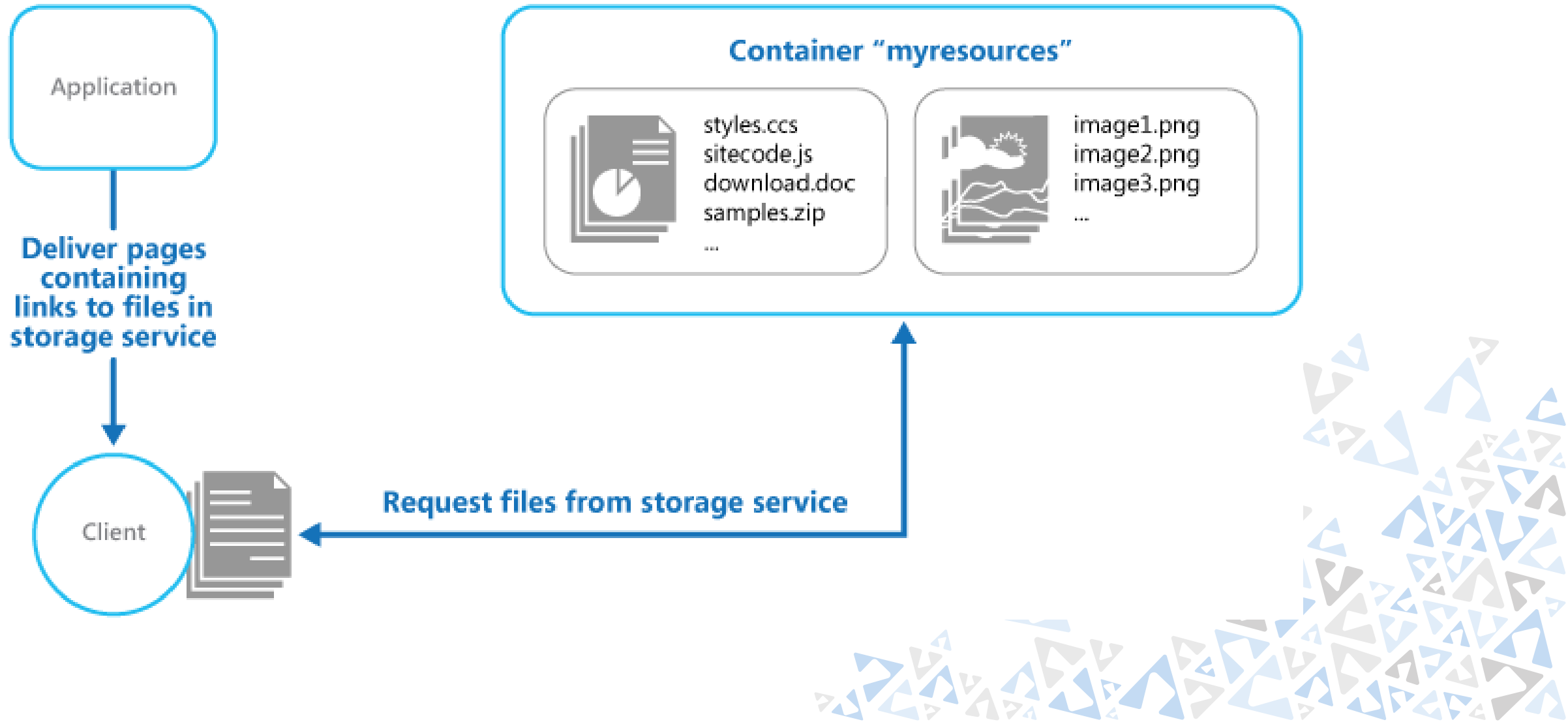
```
public string GetData()
{
    const string cacheKey = "MyData";

    string result;
    if (!memoryCache.TryGetValue(cacheKey, out result))
    {
        lock (dataLock)
        {
            if (!memoryCache.TryGetValue(cacheKey, out result))
            {
                {
                    result = FetchDataFromDb();

                    memoryCache.Set(
                        key: cacheKey,
                        value: result,
                        options: new MemoryCacheEntryOptions()
                        {
                            AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(5), Priority = CacheItemPriority.Normal
                        });
                }
            }
        }
    }
    return result;
}
private readonly object dataLock = new object();
```



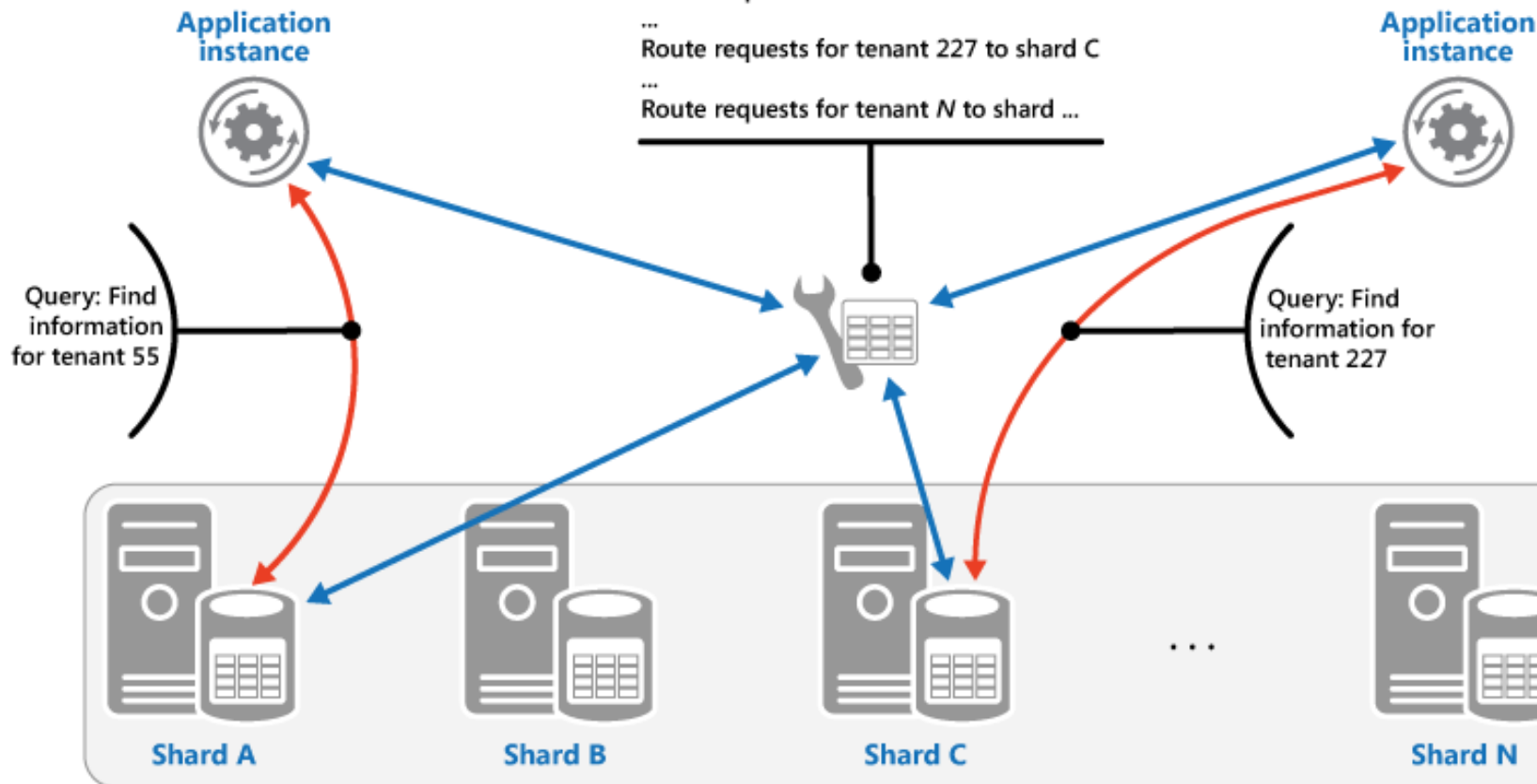
Static Content Hosting



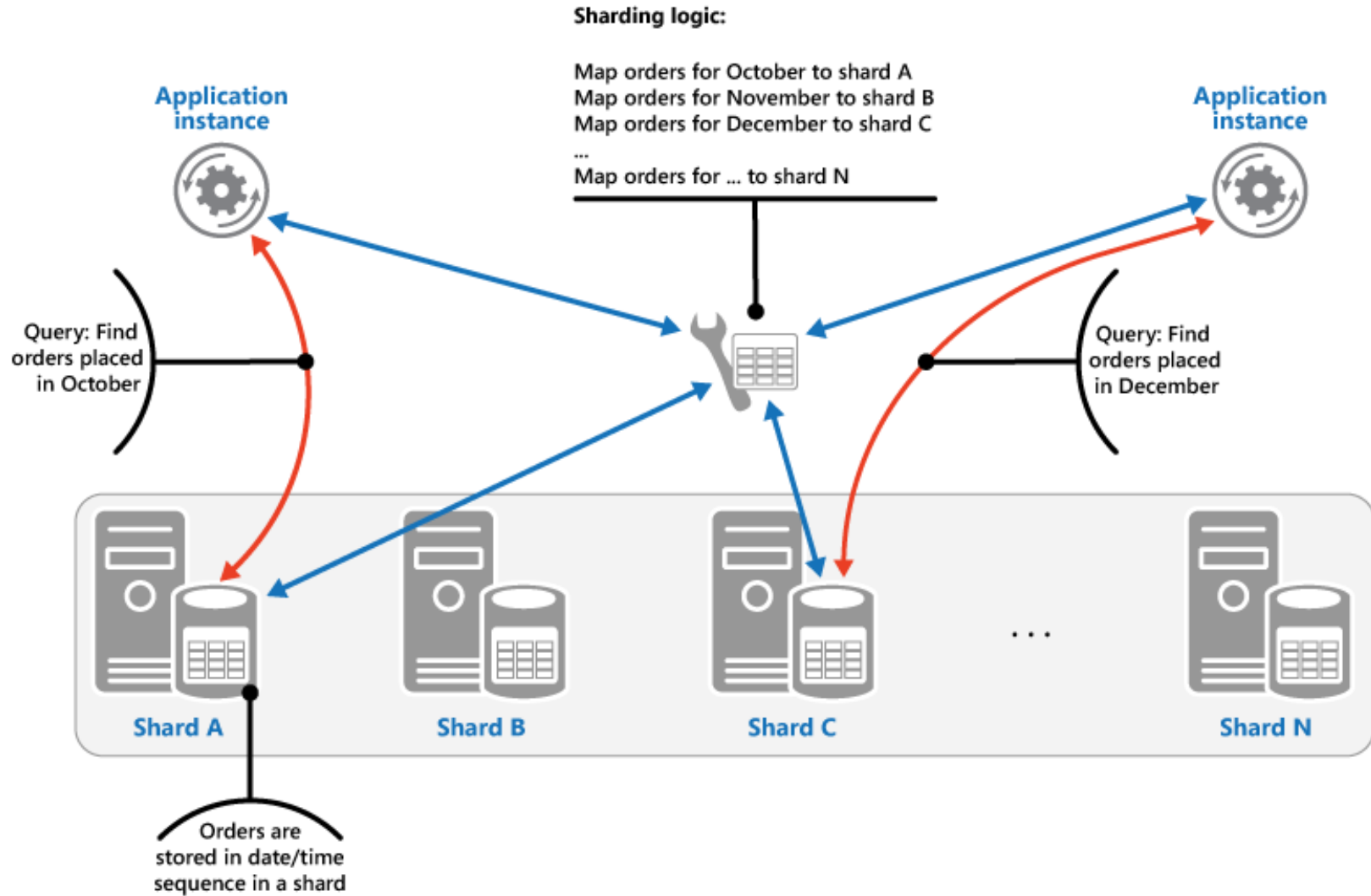
Sharding – Lookup Strategy

Sharding logic:

Route requests for tenant 1 to shard ...
...
Route requests for tenant 55 to shard A
...
Route requests for tenant 227 to shard C
...
Route requests for tenant N to shard ...



Sharding – Range Strategy



Sharding – Hash Strategy

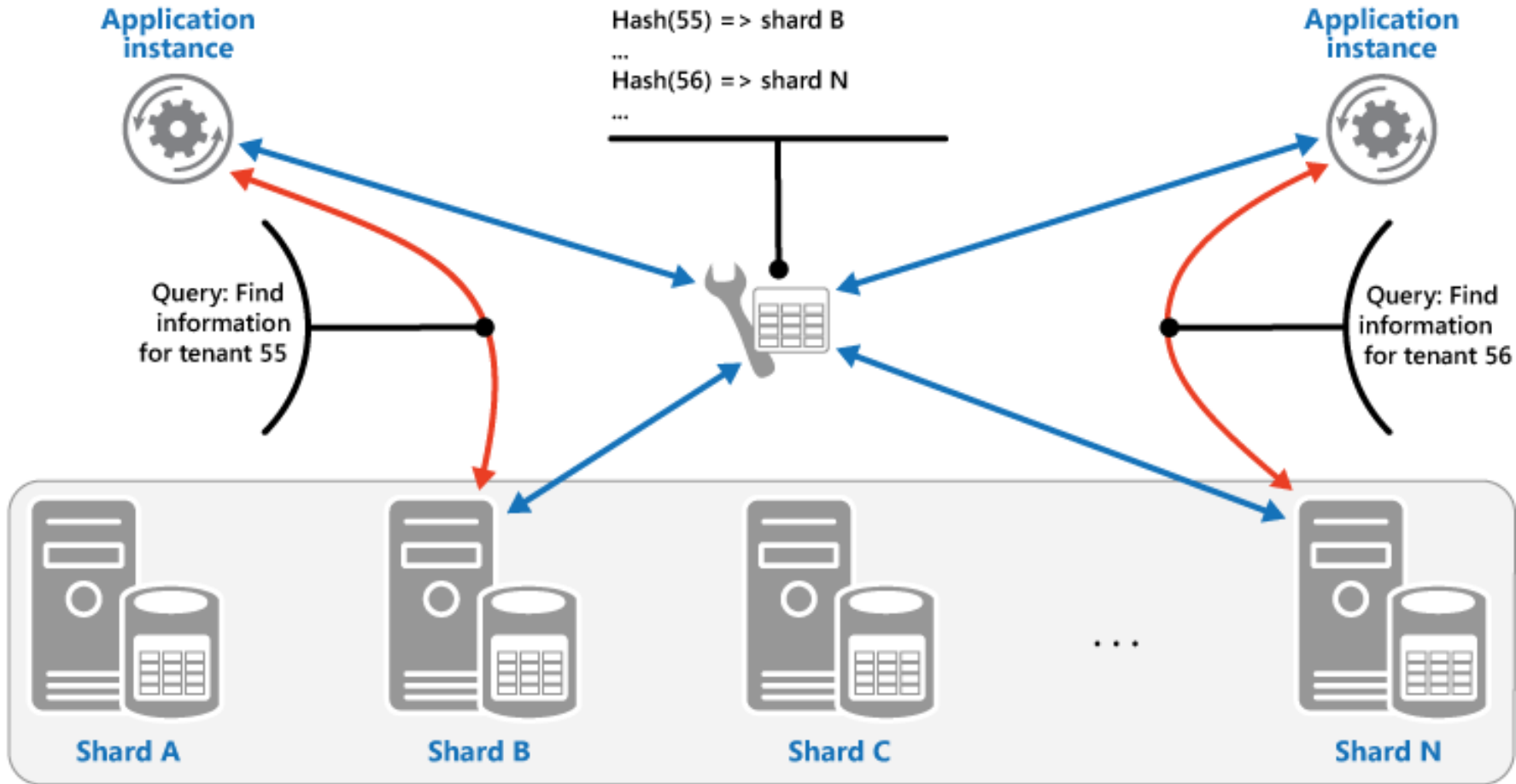
Sharding logic uses hashing:

Hash(55) => shard B

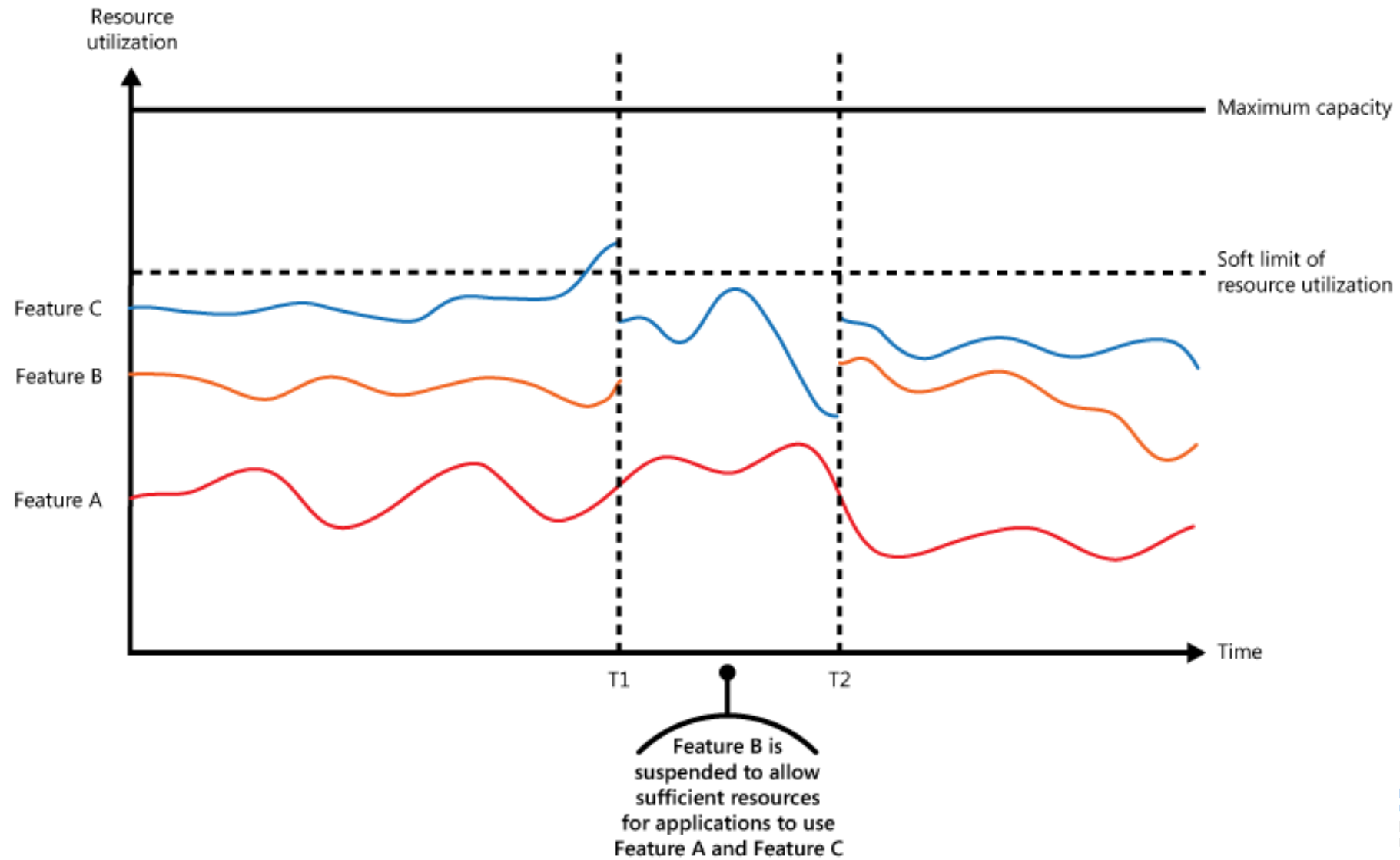
...

Hash(56) => shard N

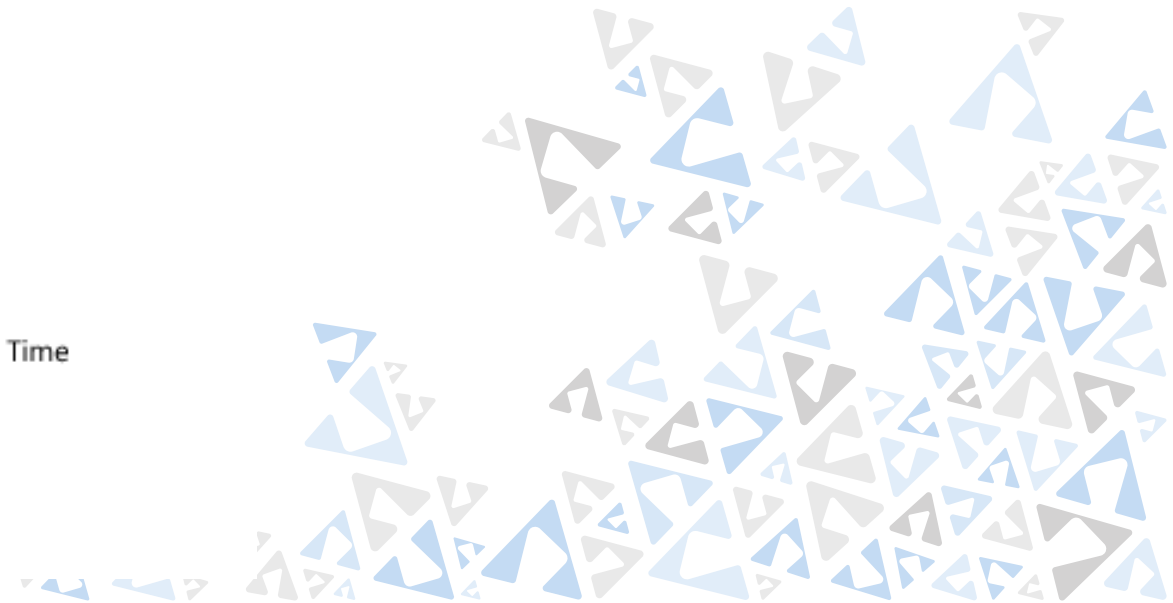
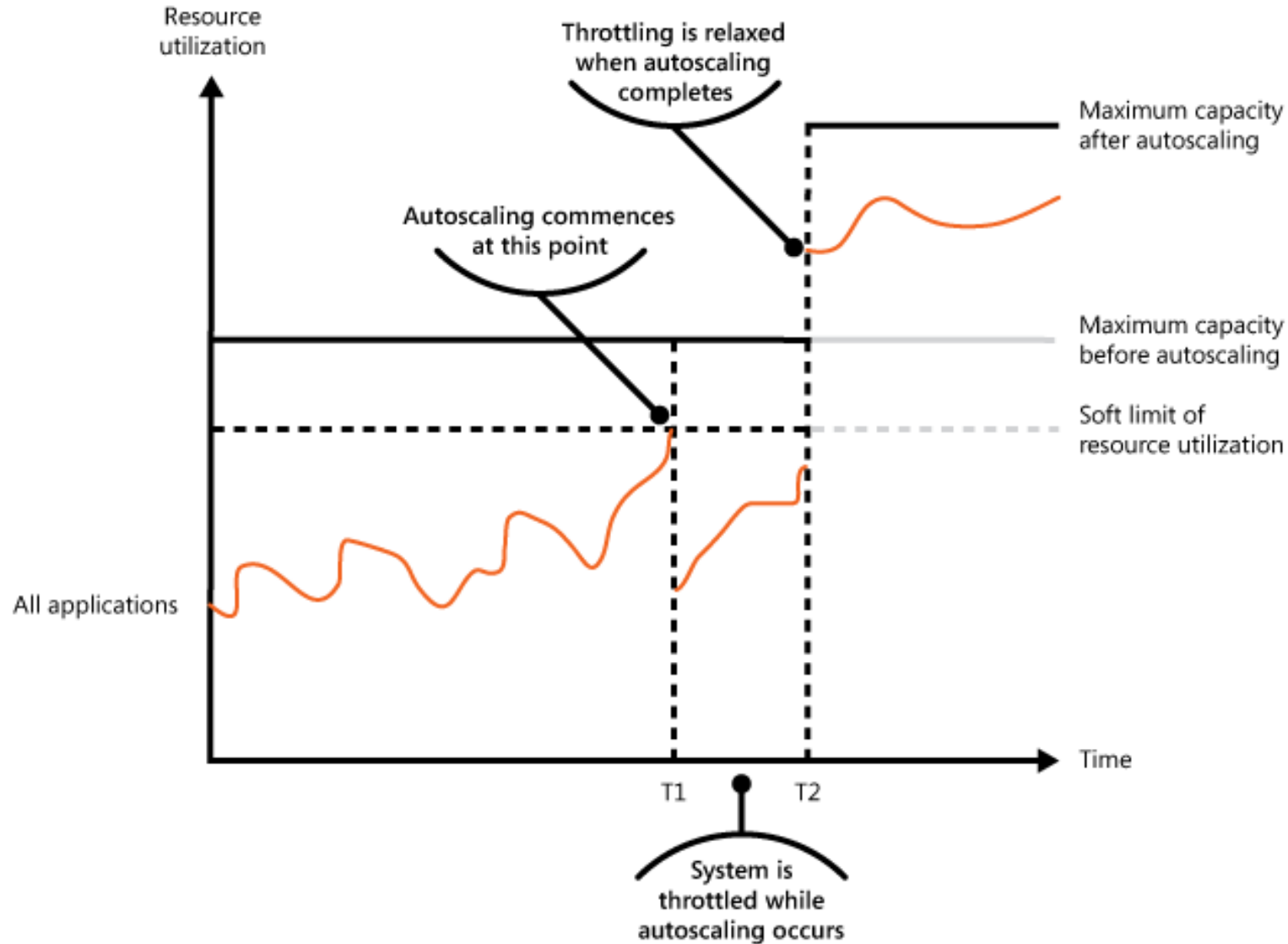
...



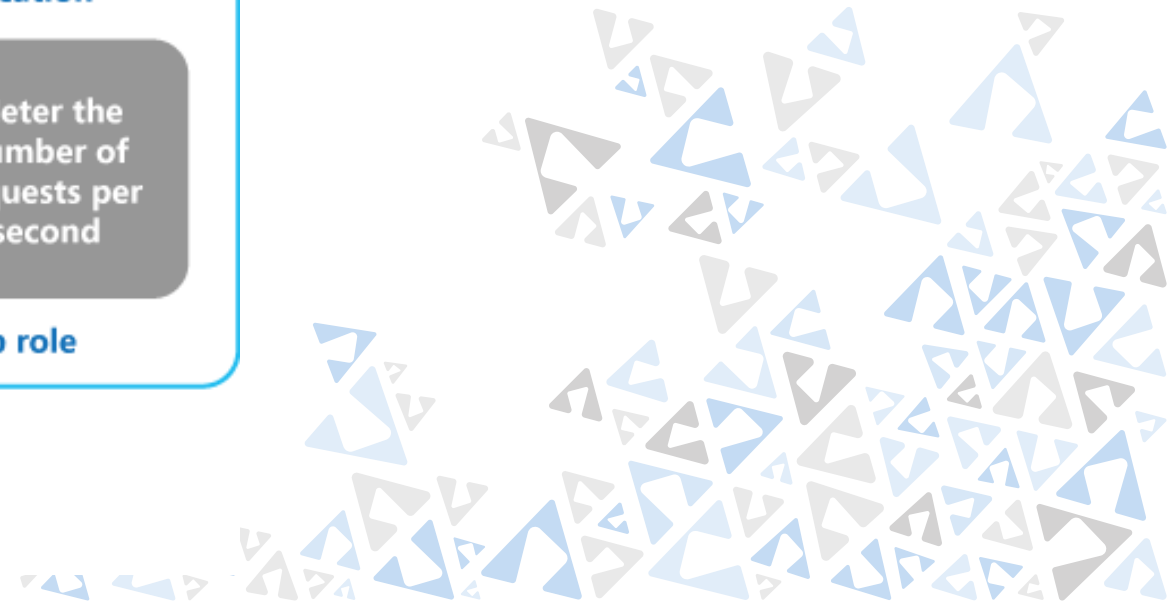
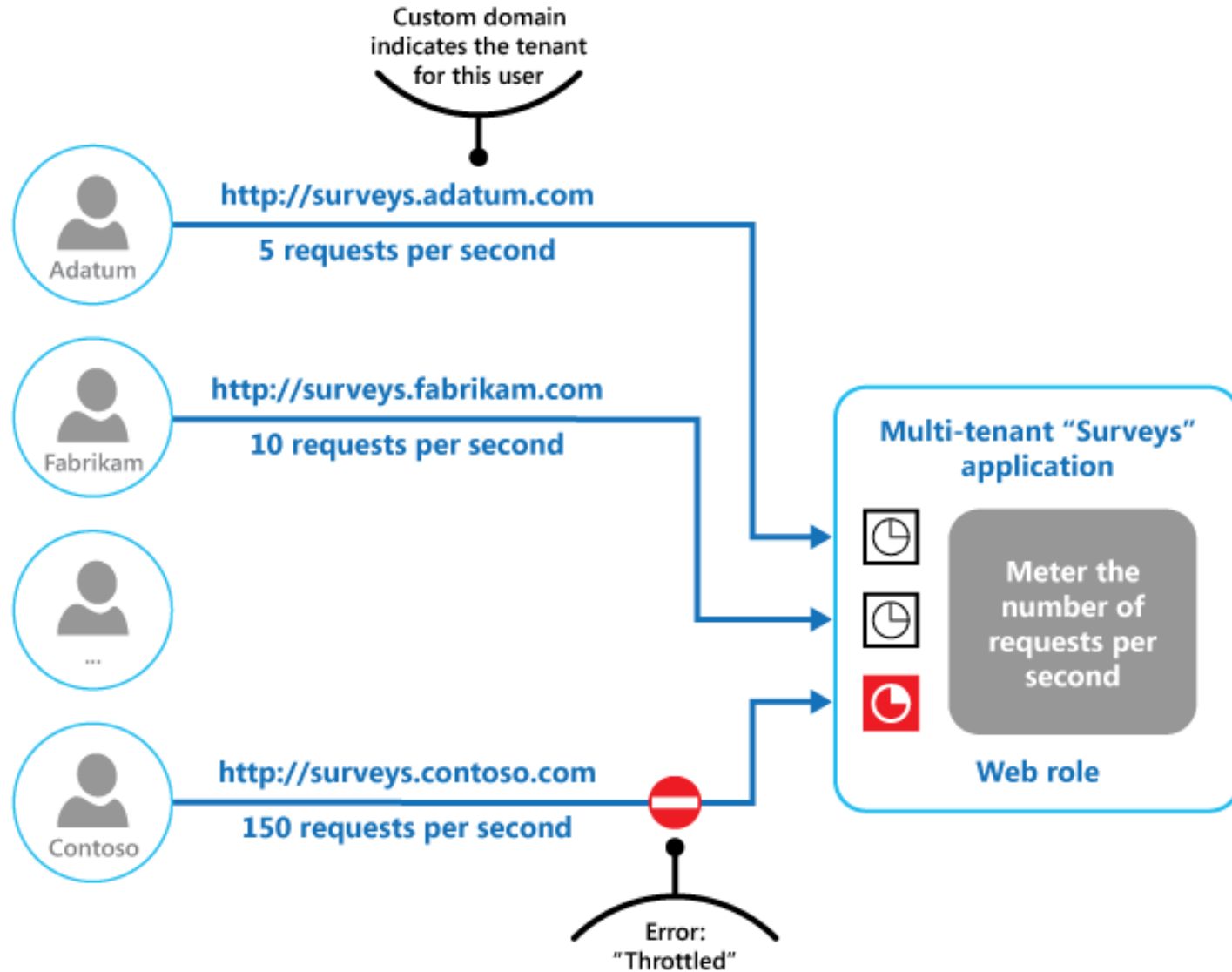
Throttling 1/3



Throttling – combined with Auto-Scaling 2/3



Throttling – Example 3/3



Resiliency

Bulkhead

Circuit Breaker

Compensating Transaction

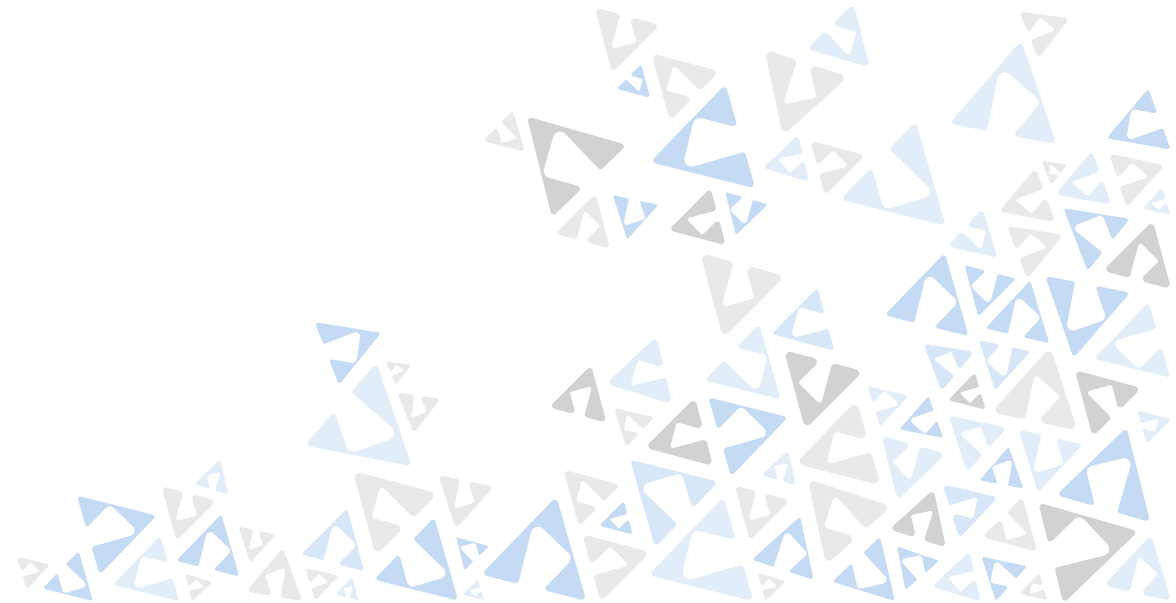
Health Endpoint Monitoring

Leader Election

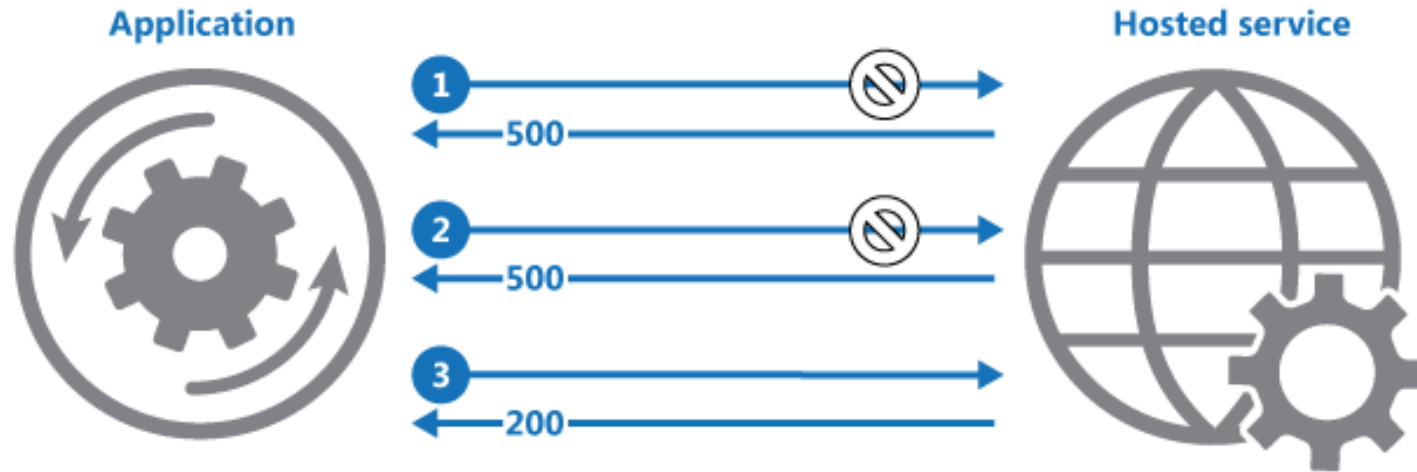
Queue-Based Load Leveling

Retry

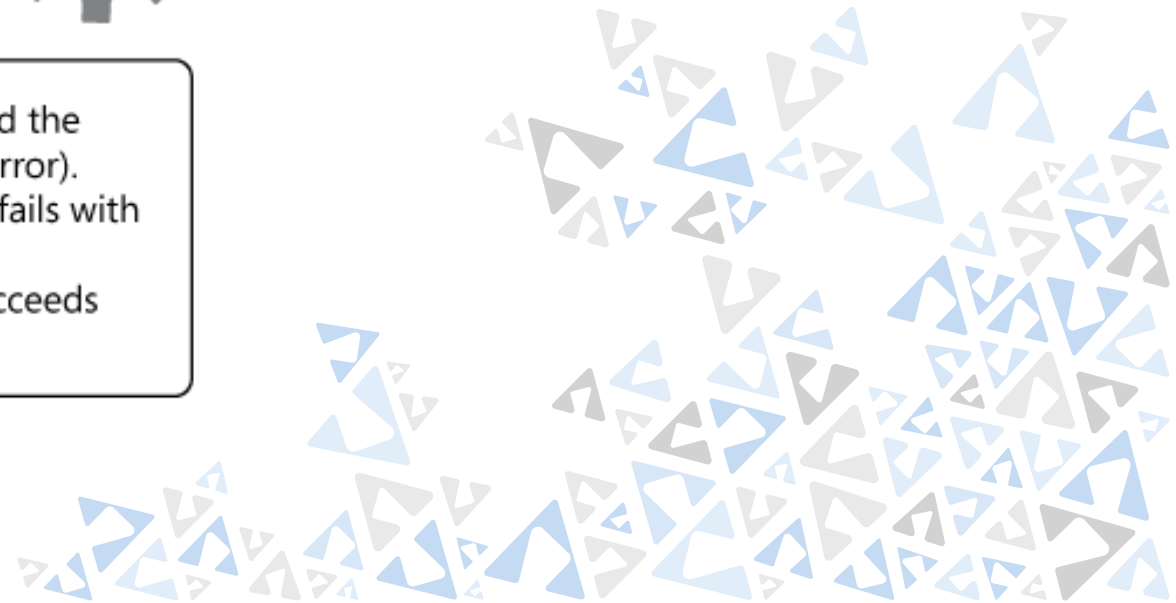
Scheduler Agent Supervisor



Retry



- 1: Application invokes operation on hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).
- 2: Application waits for a short interval and tries again. The request still fails with HTTP response code 500.
- 3: Application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).

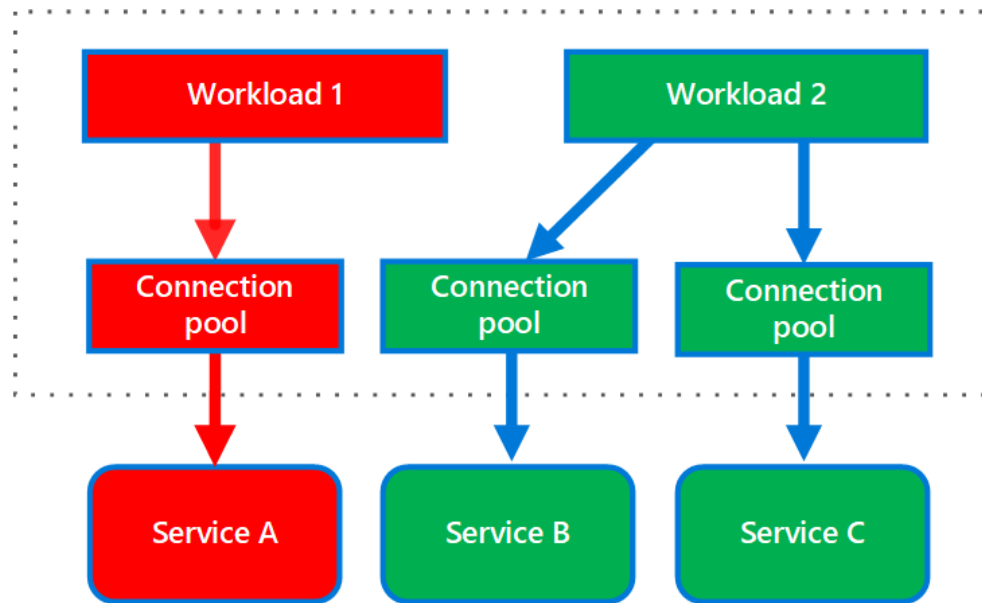


Retry – Azure Services

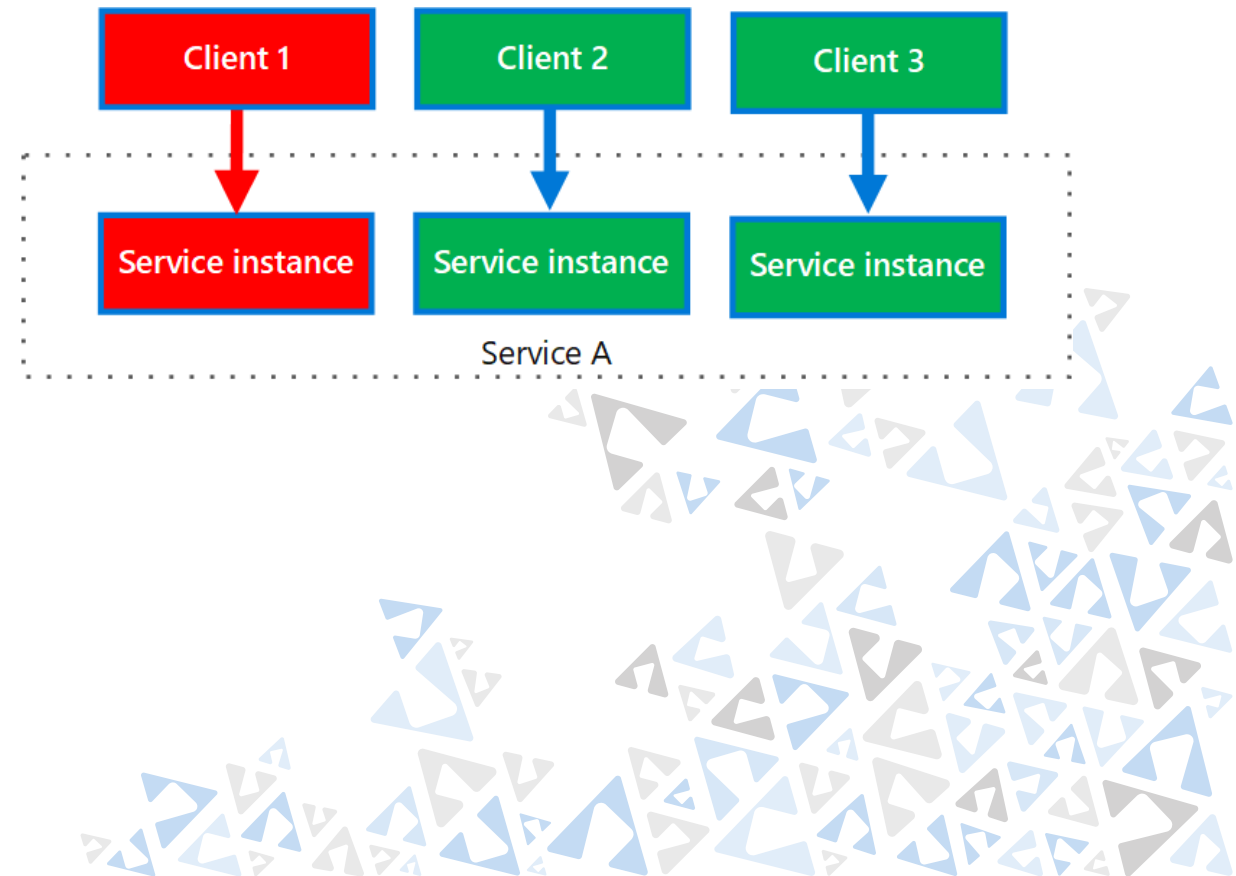
Service	Retry capabilities	Policy configuration	Scope	Telemetry features
<u>Azure Storage</u>	Native in client	Programmatic	Client and individual operations	TraceSource
<u>SQL Database with Entity Framework</u>	Native in client	Programmatic	Global per AppDomain	None
<u>SQL Database with Entity Framework Core</u>	Native in client	Programmatic	Global per AppDomain	None
<u>SQL Database with ADO.NET</u>	<u>Polly</u>	Declarative and programmatic	Single statements or blocks of code	Custom
<u>Service Bus</u>	Native in client	Programmatic	Namespace Manager, Messaging Factory, and Client	ETW
<u>Azure Redis Cache</u>	Native in client	Programmatic	Client	TextWriter
<u>Cosmos DB</u>	Native in service	Non-configurable	Global	TraceSource
<u>Azure Search</u>	Native in client	Programmatic	Client	ETW or Custom
<u>Azure Active Directory</u>	Native in ADAL library	Embedded into ADAL library	Internal	None
<u>Service Fabric</u>	Native in client	Programmatic	Client	None
<u>Azure Event Hubs</u>	Native in client	Programmatic	Client	None

Bulkhead [přepážka]

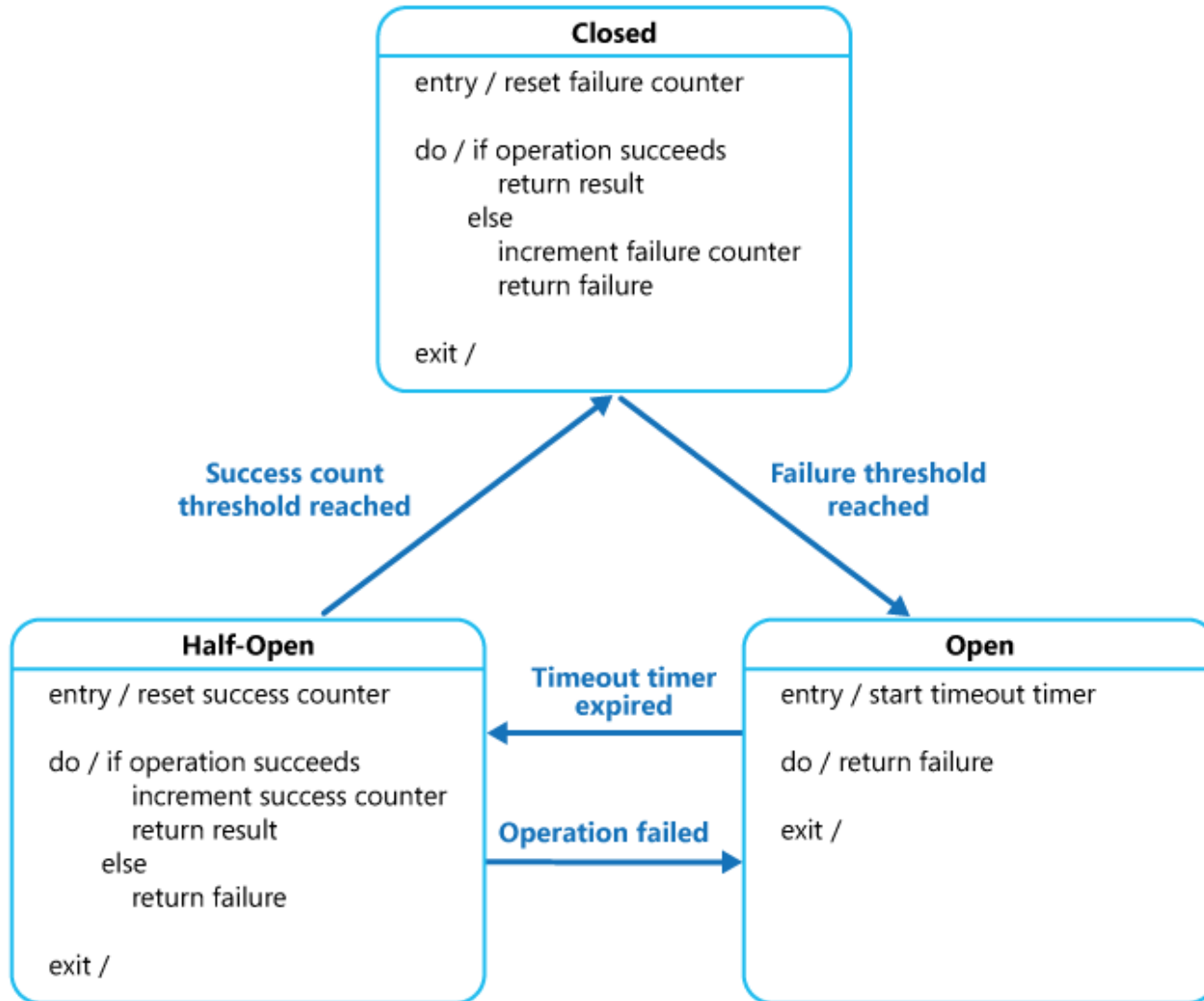
connection pools that call individual services



multiple clients calling a single service



Circuit Breaker



<https://github.com/App-vNext/Polly>



Data Management

Cache-Aside

CQRS

Event Sourcing

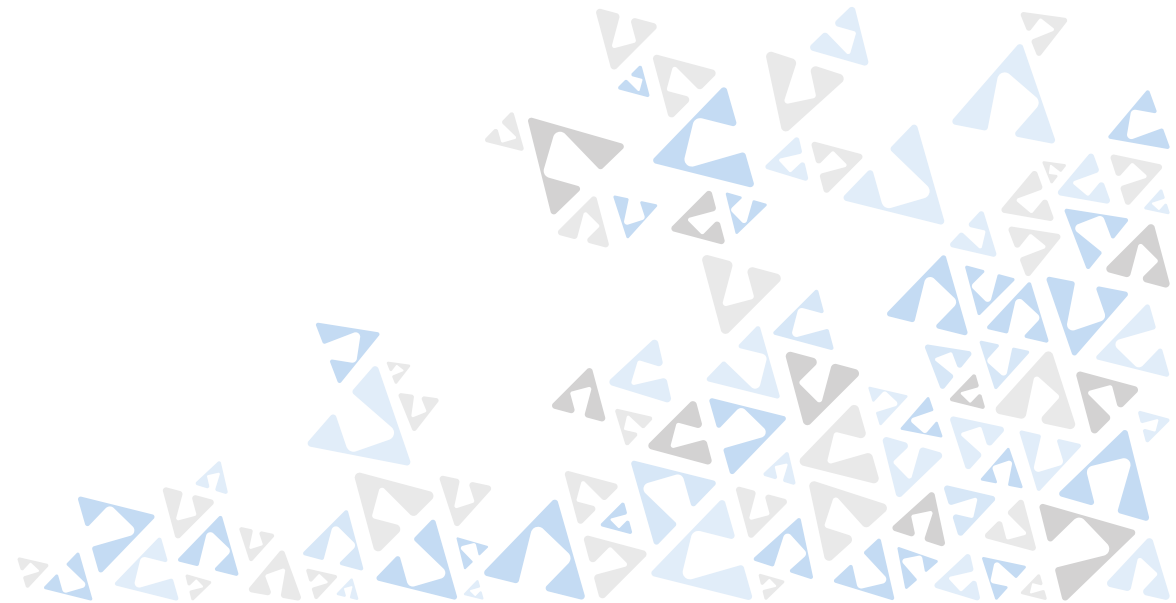
Index Table

Materialized View

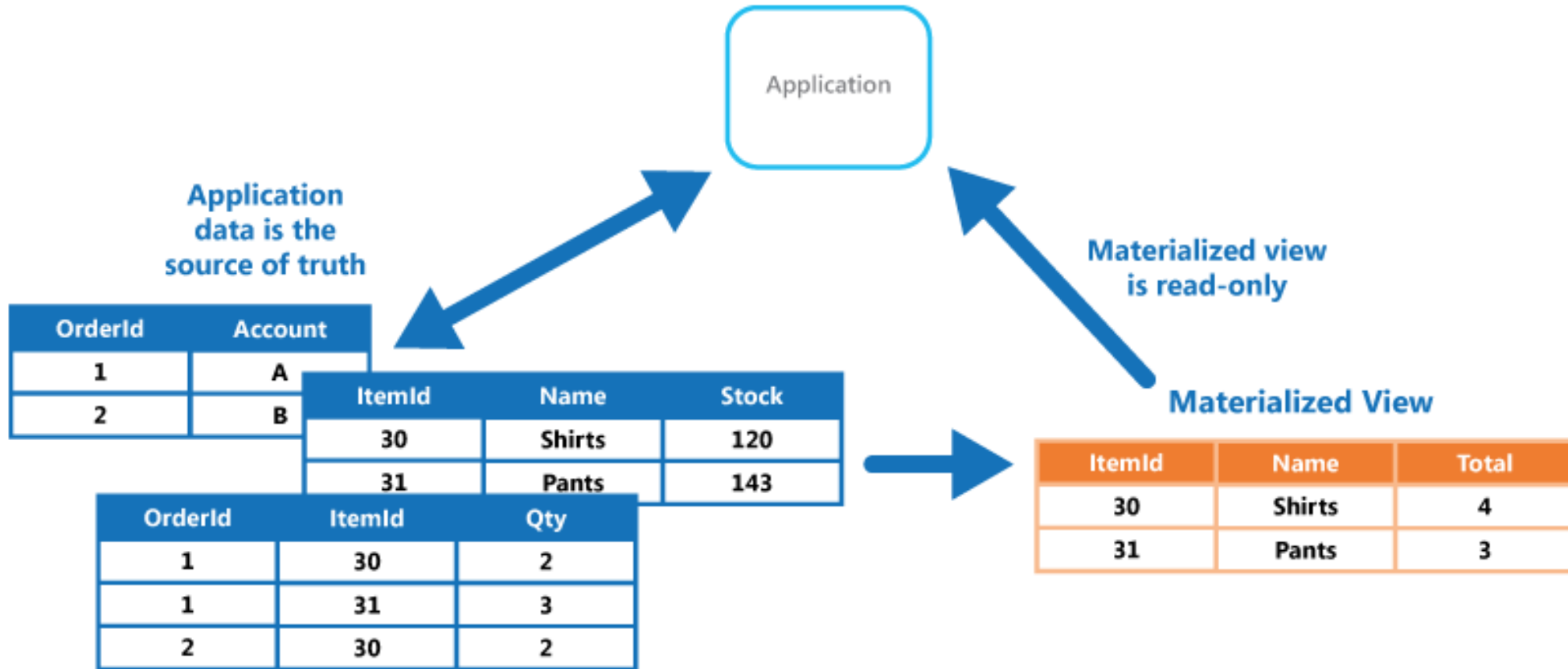
Sharding

Static Content Hosting

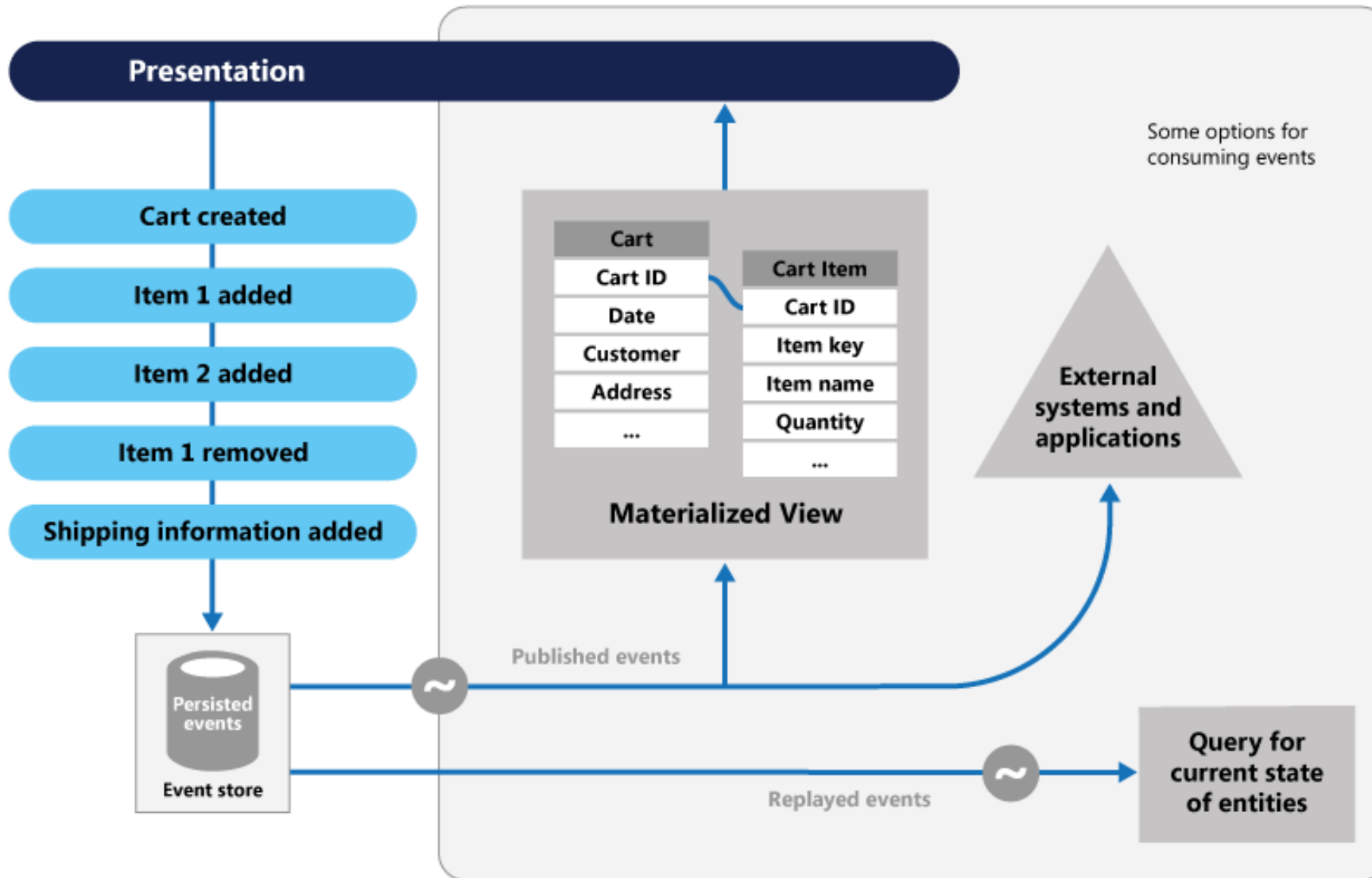
Valet Key



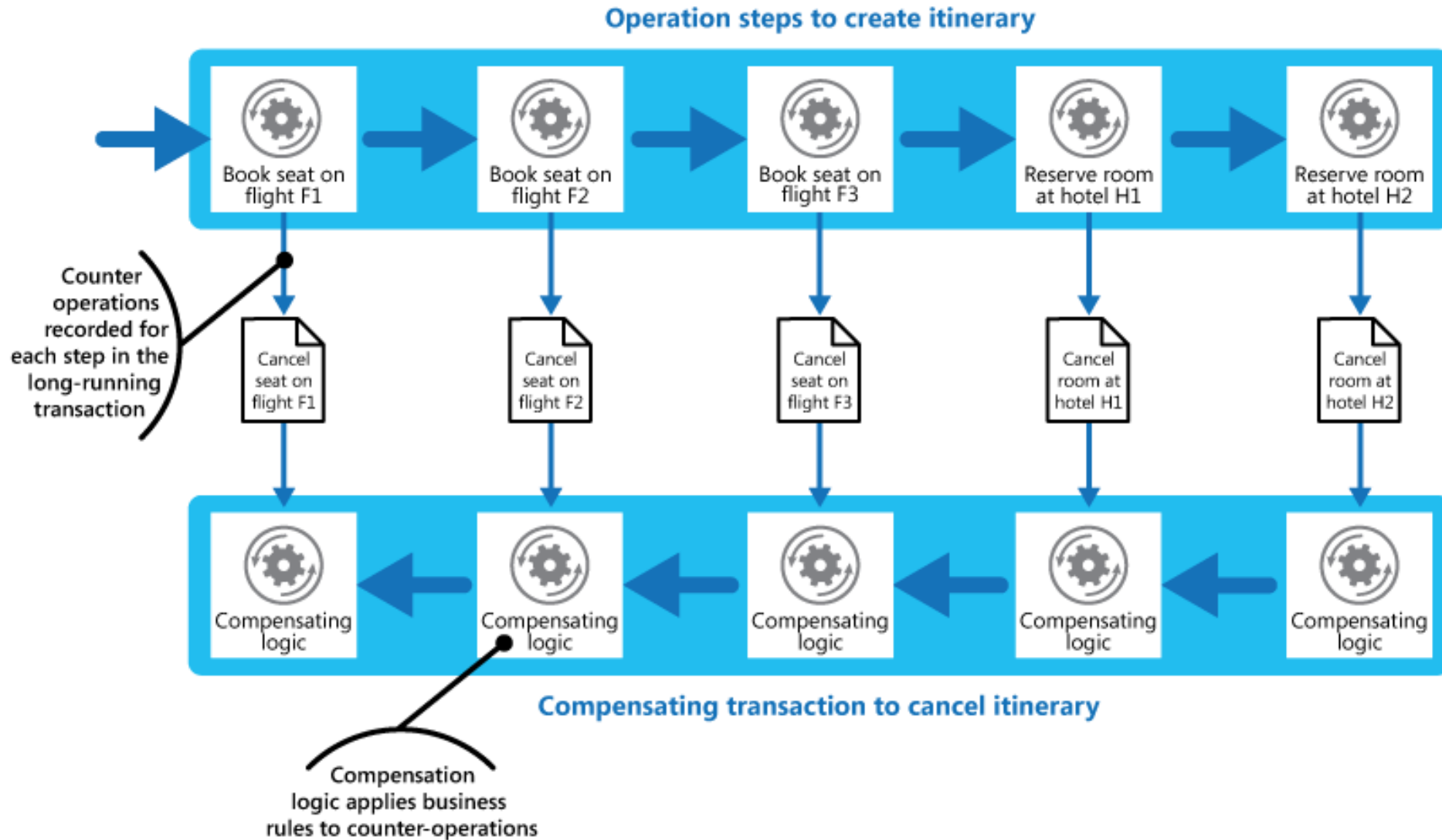
Materialized View



Event Sourcing

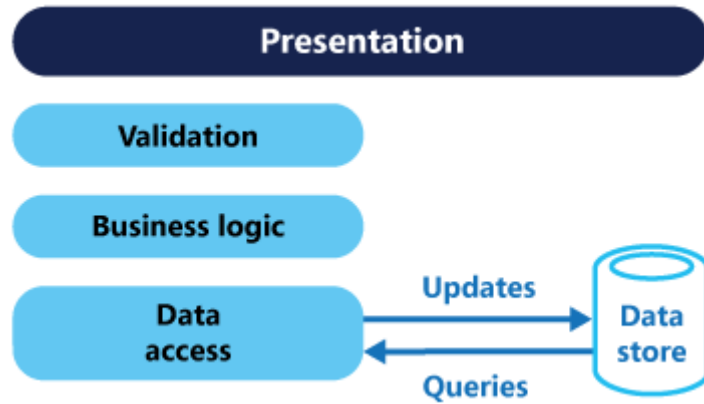


Compensating Transaction

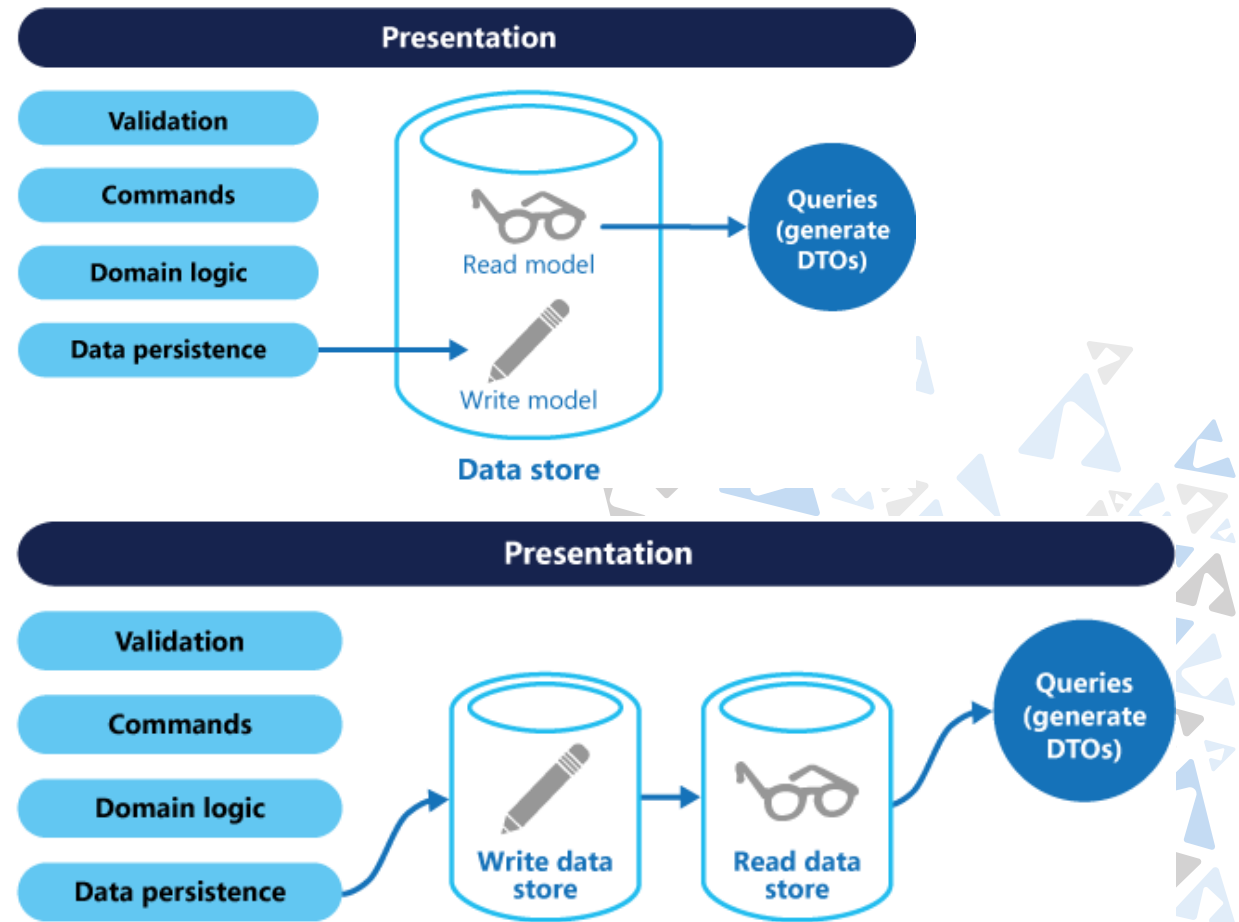


CQRS - Command and Query Responsibility Segregation

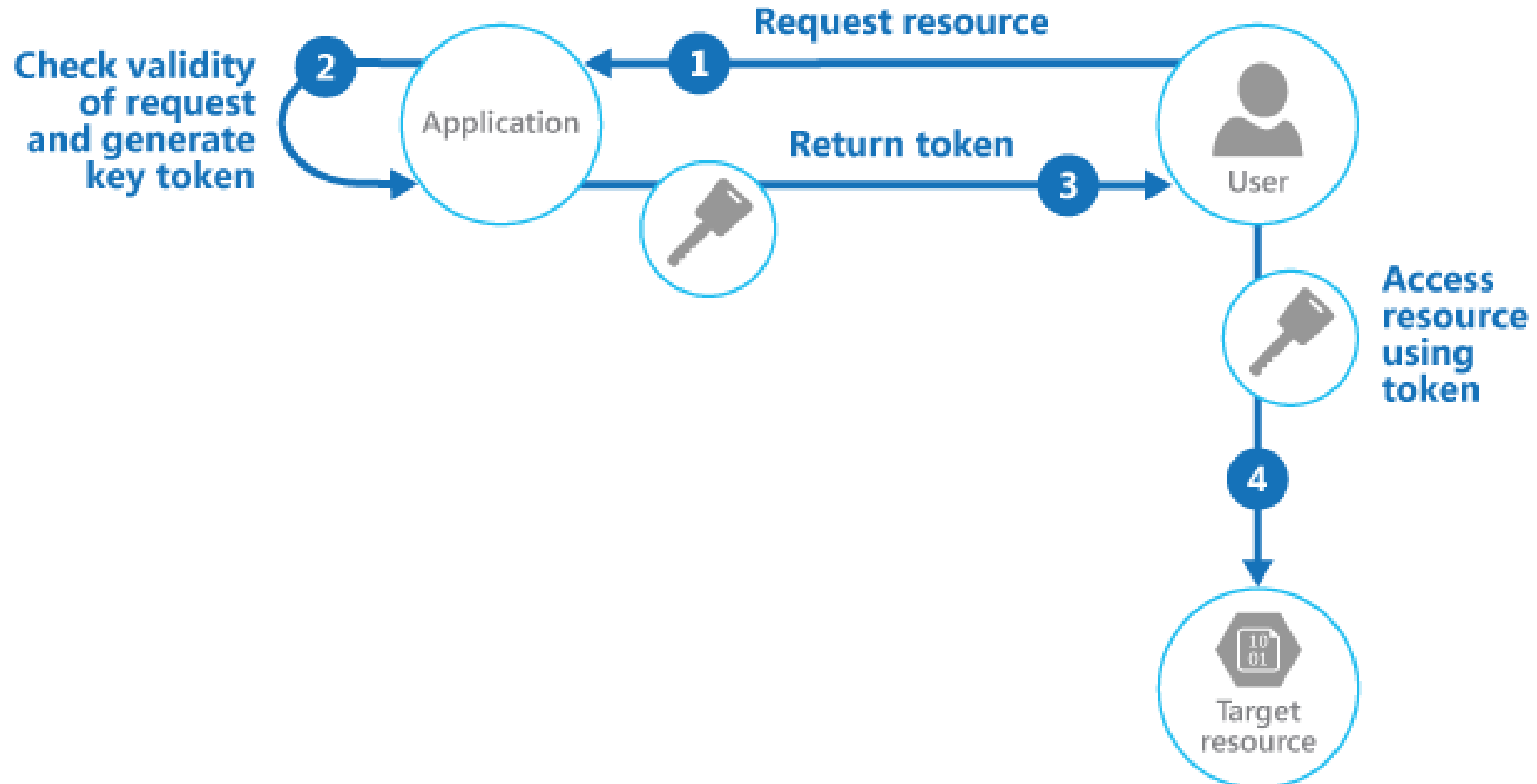
CRUD



CQRS



Valet Key





- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve proble...
- SETTINGS
- Access keys
- Configuration
- Encryption
- Shared access signature
- Firewalls and virtual networ...
- Properties
- Locks
- Automation script
- BLOB SERVICE
- Containers
- CORS
- Custom domain

A shared access signature (SAS) is a URI that grants restricted access rights to Azure Storage resources. You can provide a shared access signature to clients who should not be trusted with your storage account key but whom you wish to delegate access to certain storage account resources. By distributing a shared access signature URI to these clients, you grant them access to a resource for a specified period of time.

An account-level SAS can delegate access to multiple storage services (i.e. blob, file, queue, table). Note that stored access policies are currently not supported for an account-level SAS.

[Learn more](#)

Allowed services ?

- Blob File Queue Table

Allowed resource types ?

- Service Container Object

Allowed permissions ?

- Read Write Delete List Add Create Update Process

Start and expiry date/time ?

Start

2018-04-08

End

2018-04-08

(UTC+02:00) --- Current Timezone --- ▼

Allowed IP addresses ?

for example, 168.1.5.65 or 168.1.5.65-168.1.5.70

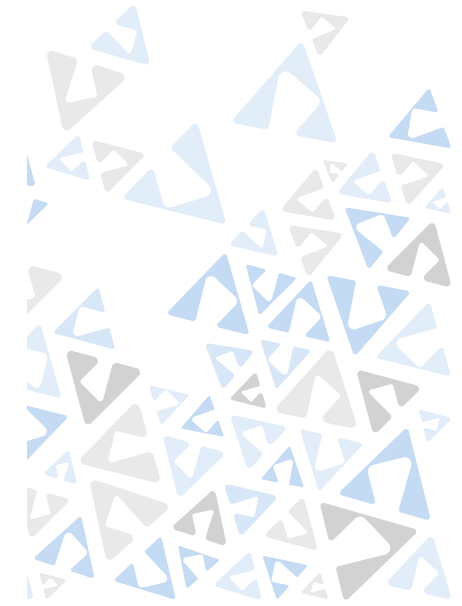
Allowed protocols ?

- HTTPS only HTTPS and HTTP

Signing key ?

key1 ▼

[Generate SAS and connection string](#)



Messaging

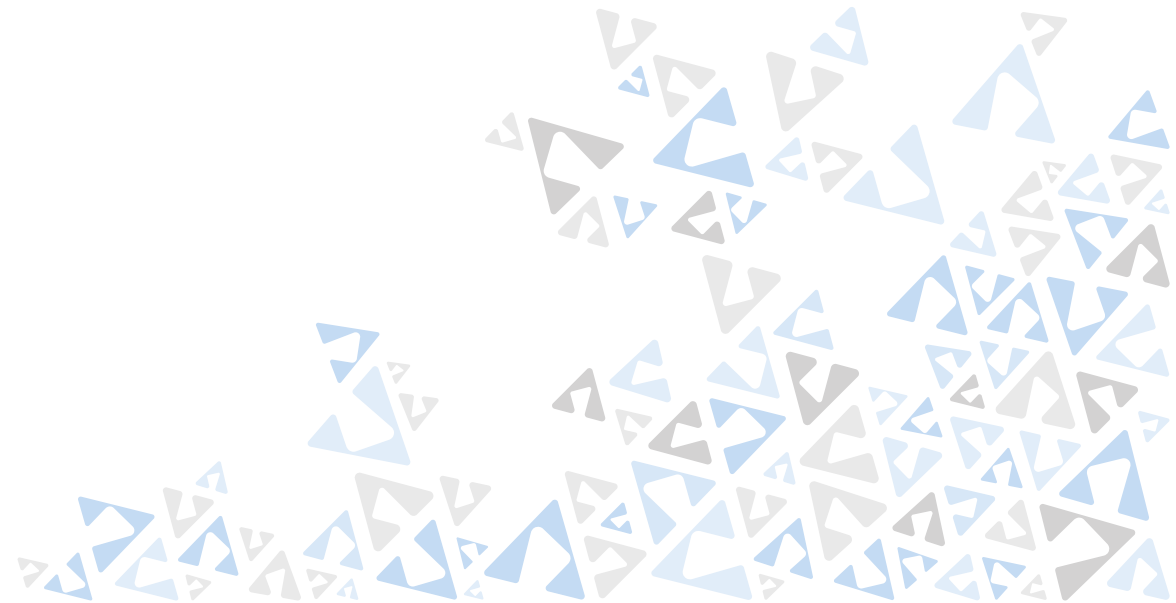
Competing Consumers

Pipes and Filters

Priority Queue

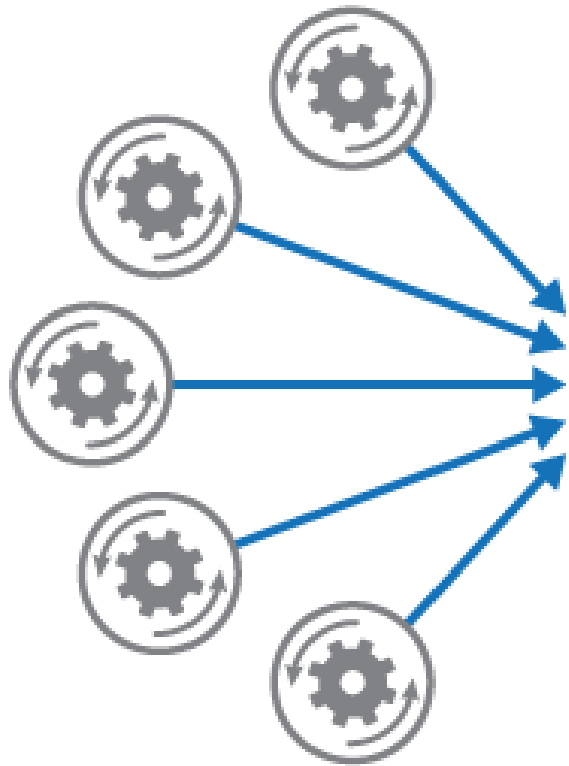
Queue-Based Load Leveling

Scheduler Agent Supervisor



Competing Consumers

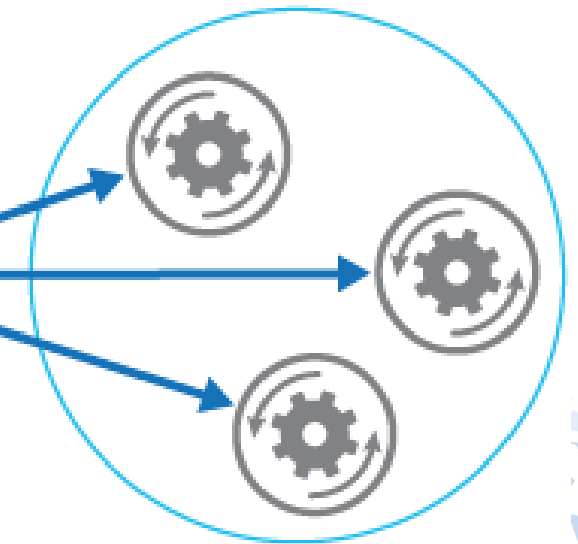
**Application instances -
generating messages**



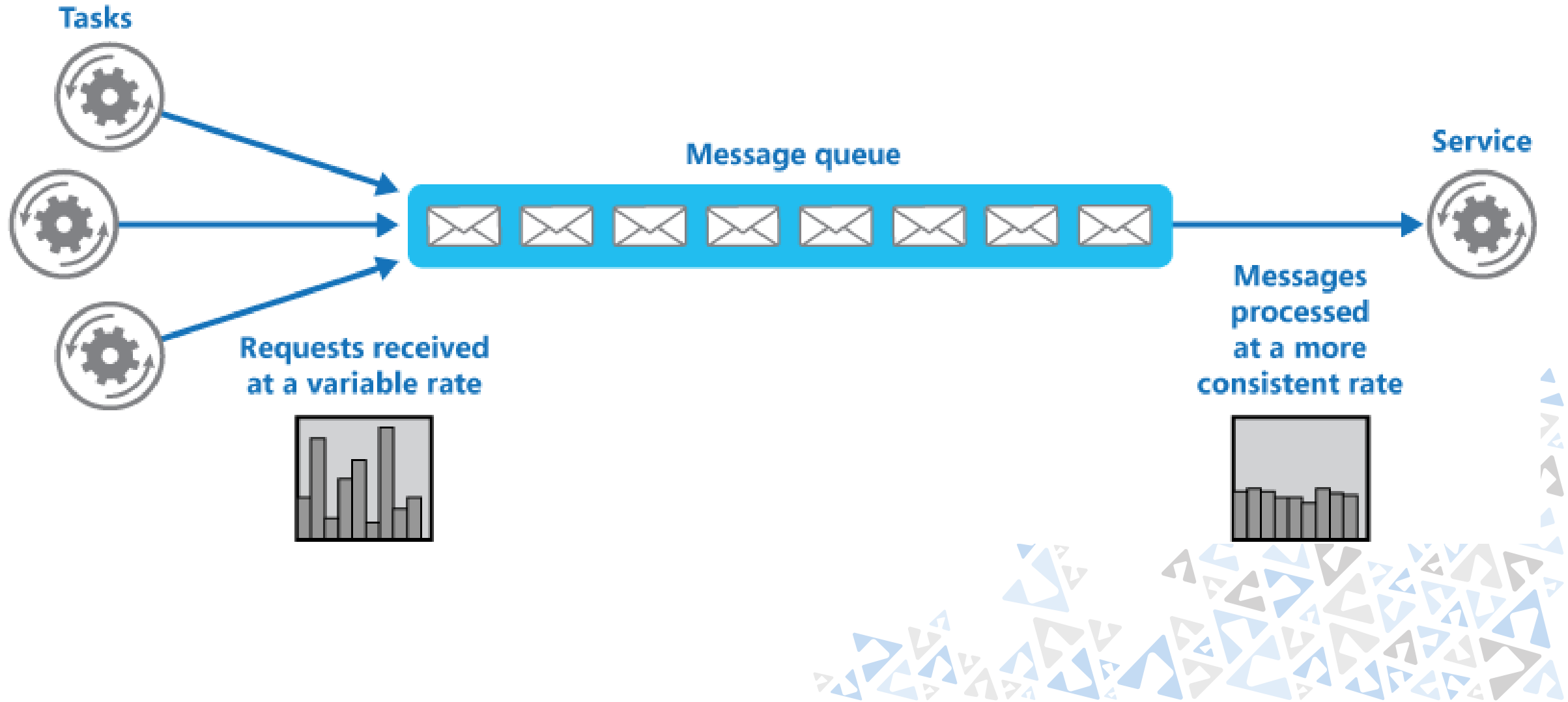
Message queue



**Consumer service
instance pool -
processing messages**

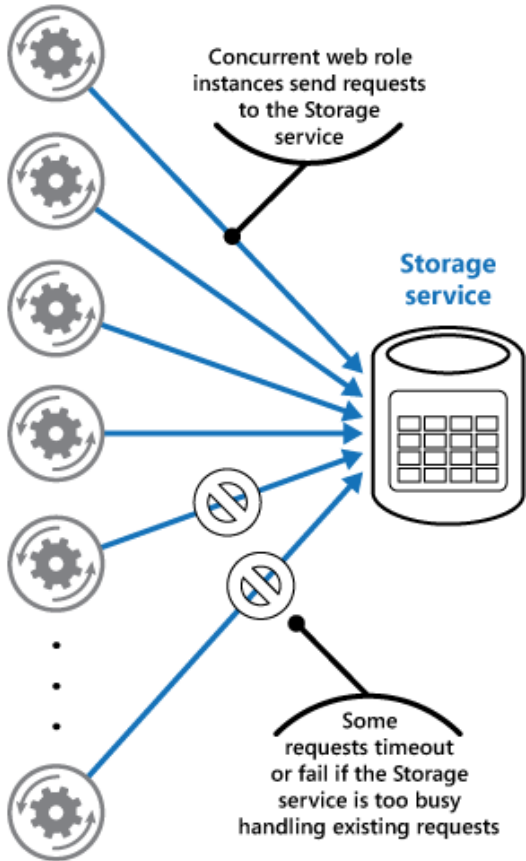


Queue-Based Load Leveling 1/2

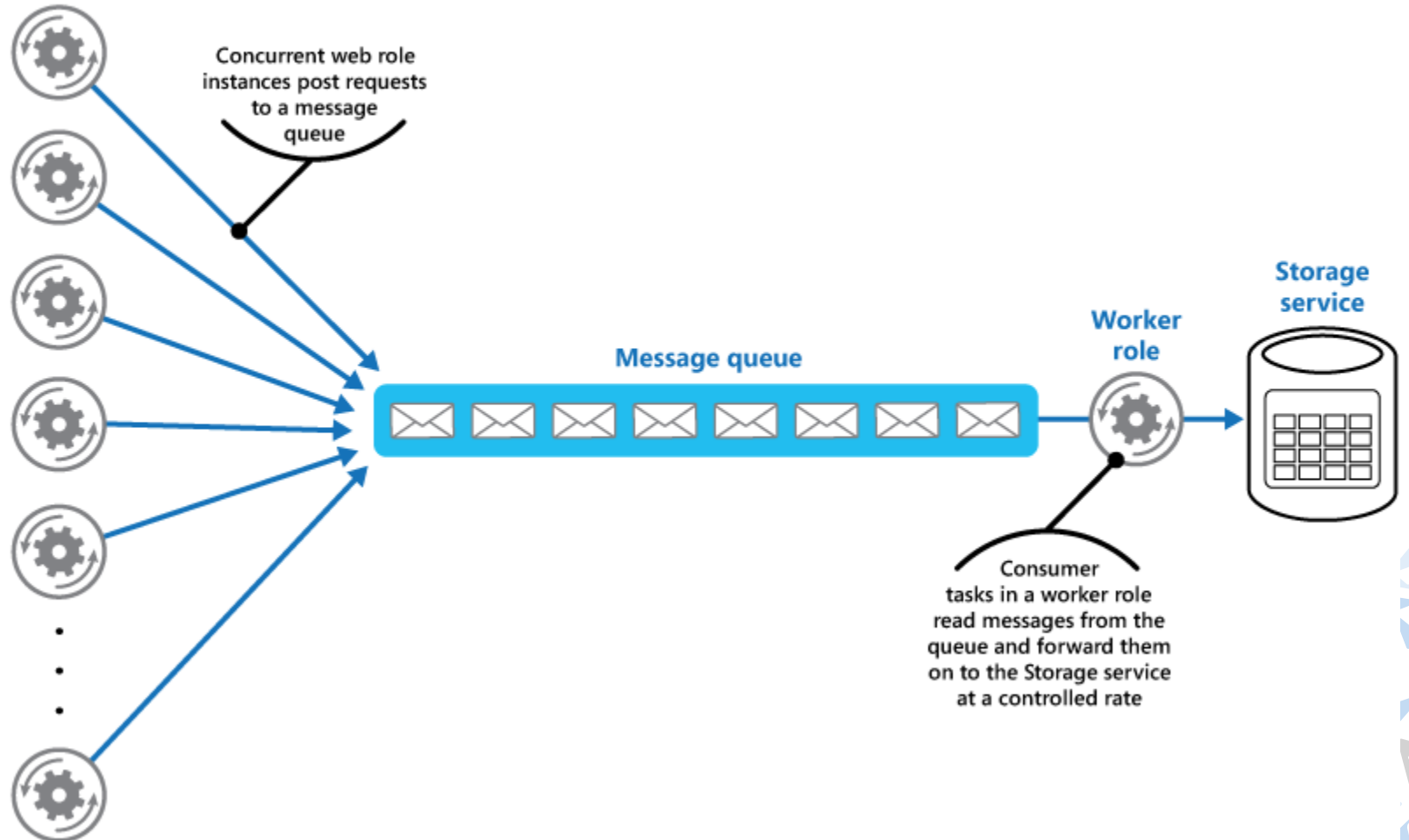


Queue-Based Load Leveling 2/2 - Example

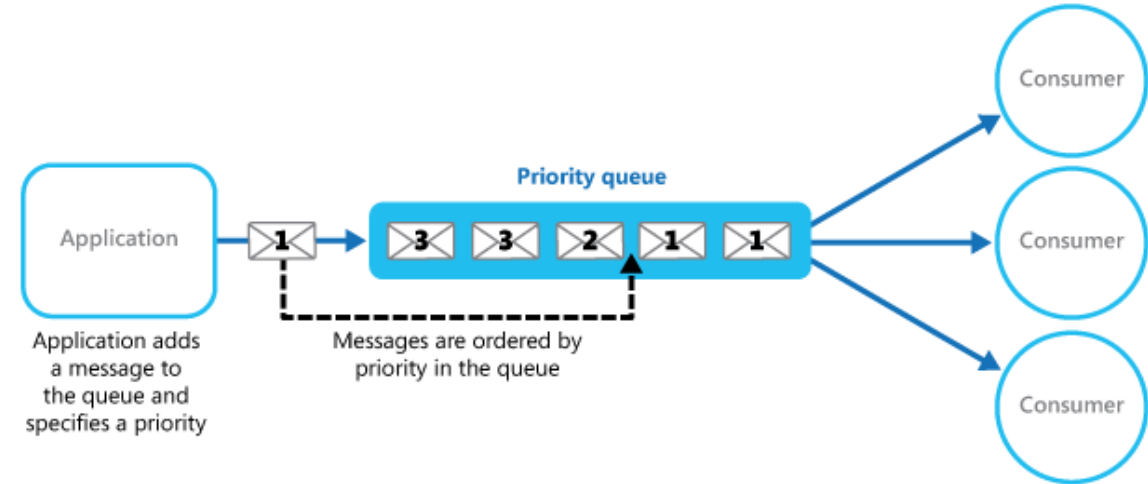
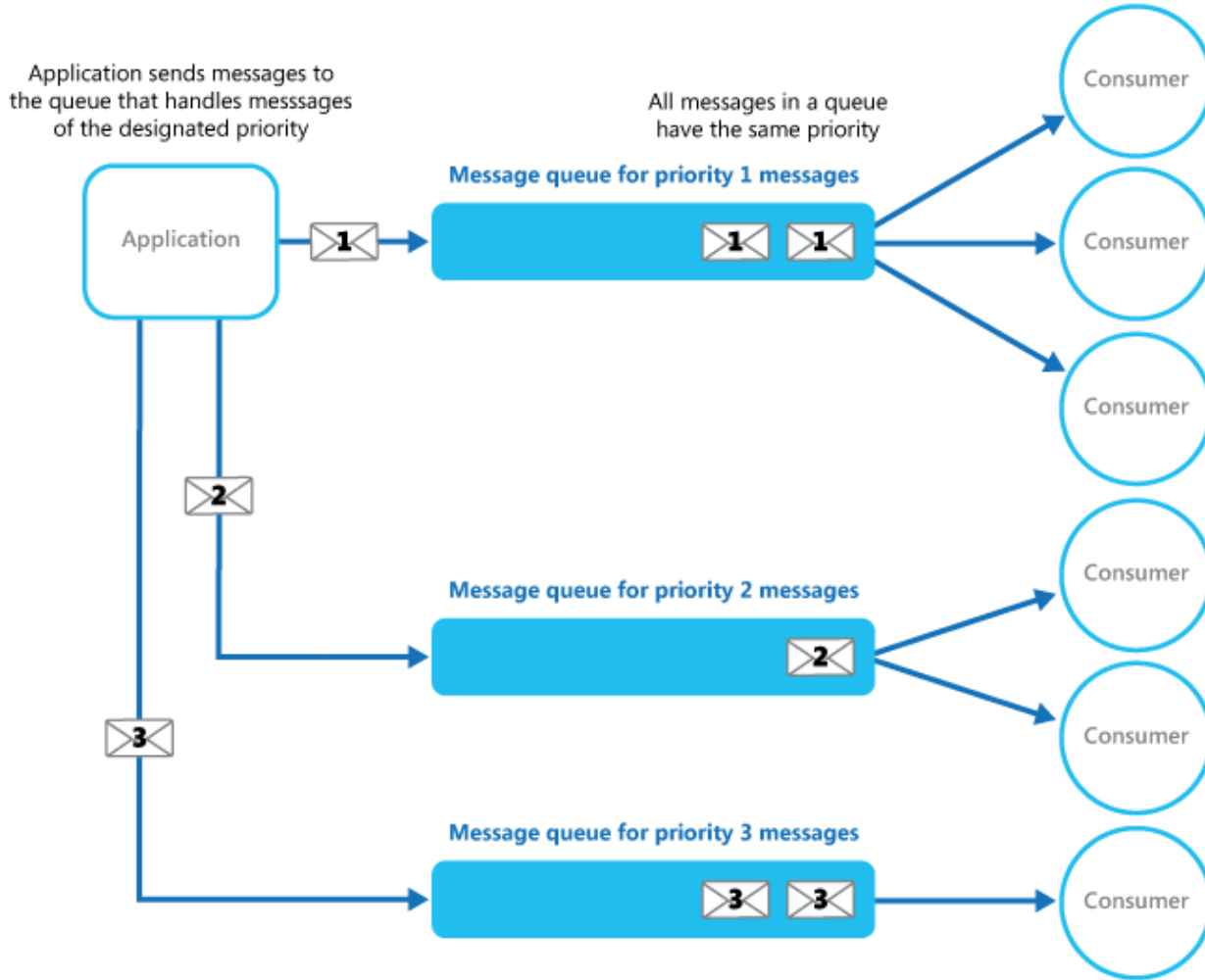
Web role instances



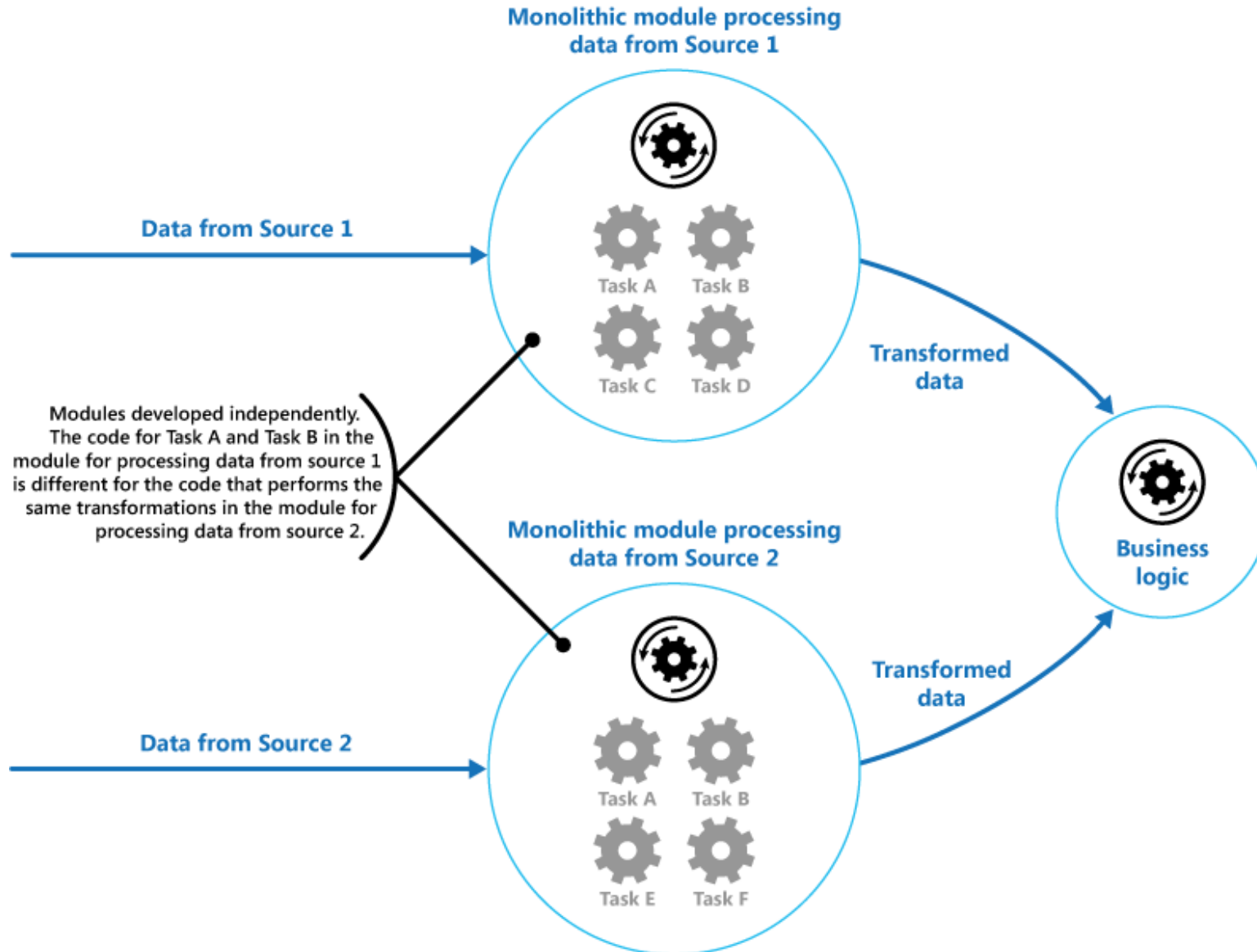
Web role instances



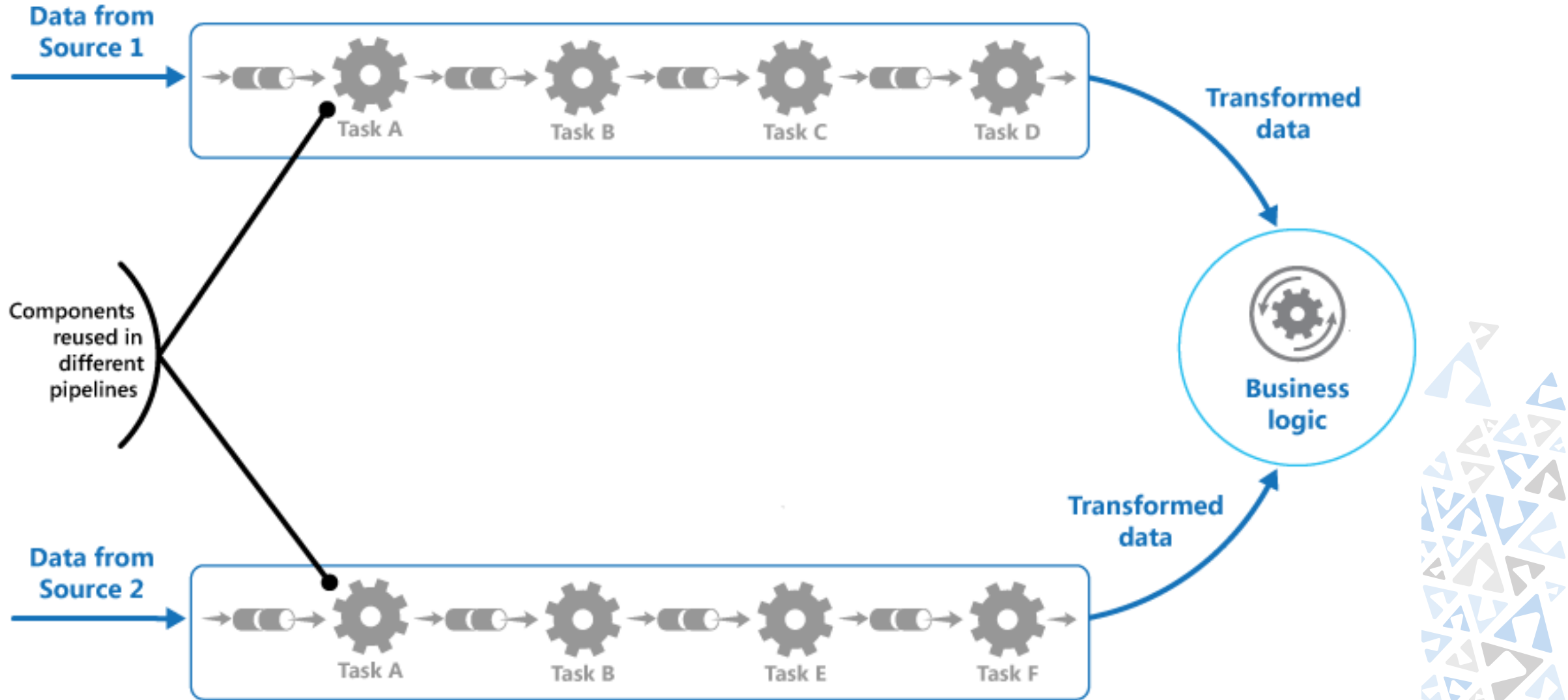
Priority Queue



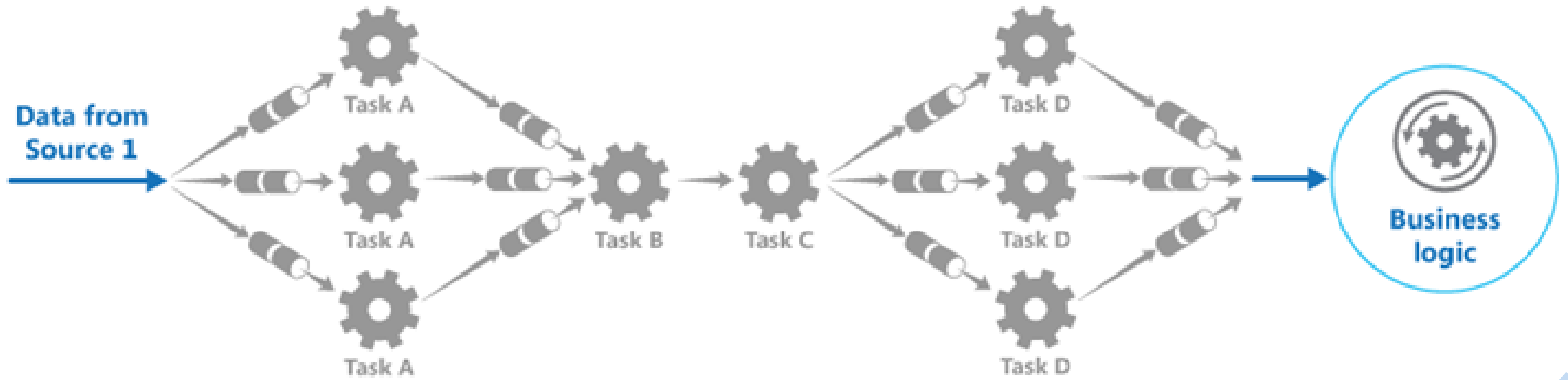
Pipes and Filters 1/3



Pipes and Filters 2/3



Pipes and Filters 3/3



Design and Implementation

Ambassador

Anti-Corruption Layer

Backends for Frontends

CQRS

Compute Resource Consolidation

External Configuration Store

Gateway Aggregation

Gateway Offloading

Gateway Routing

Leader Election

Pipes and Filters

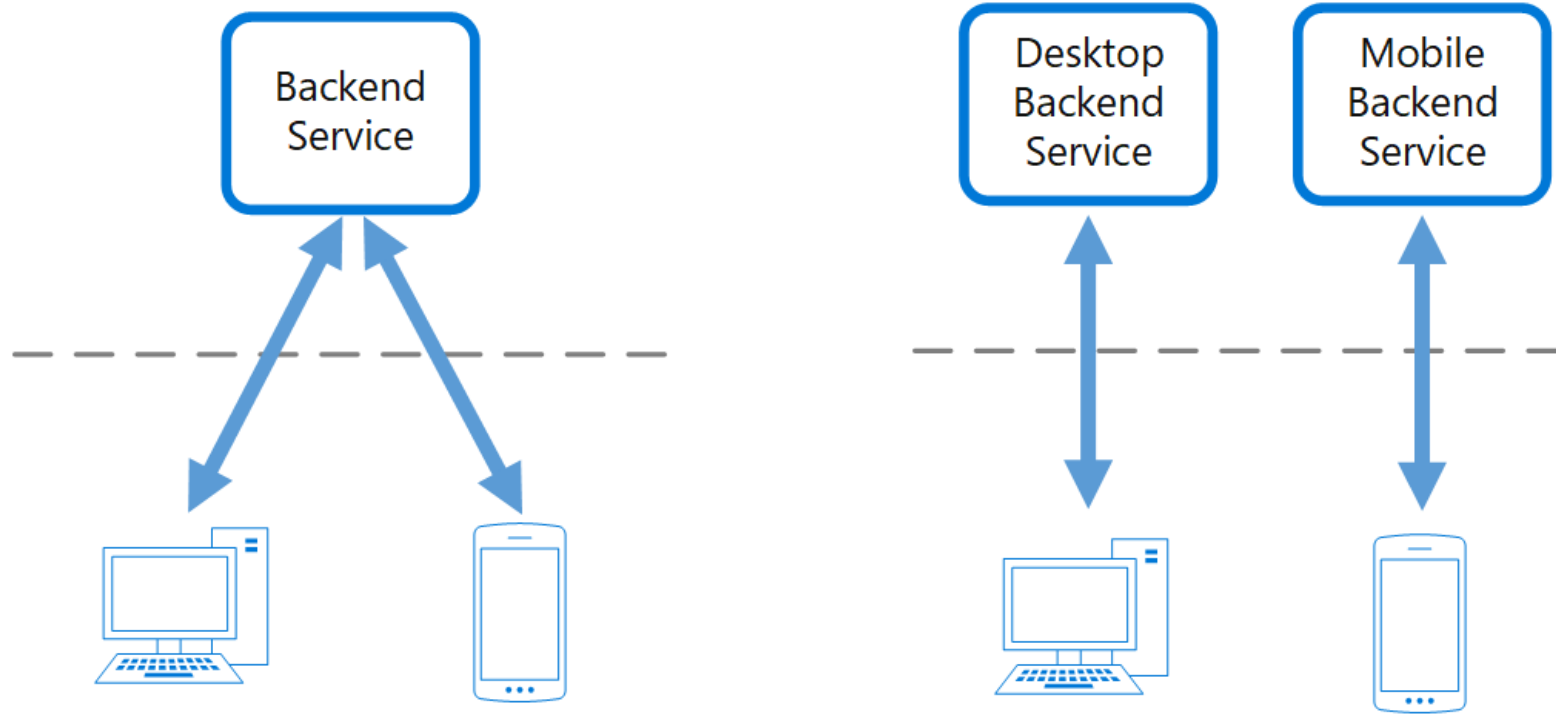
Sidecar

Static Content Hosting

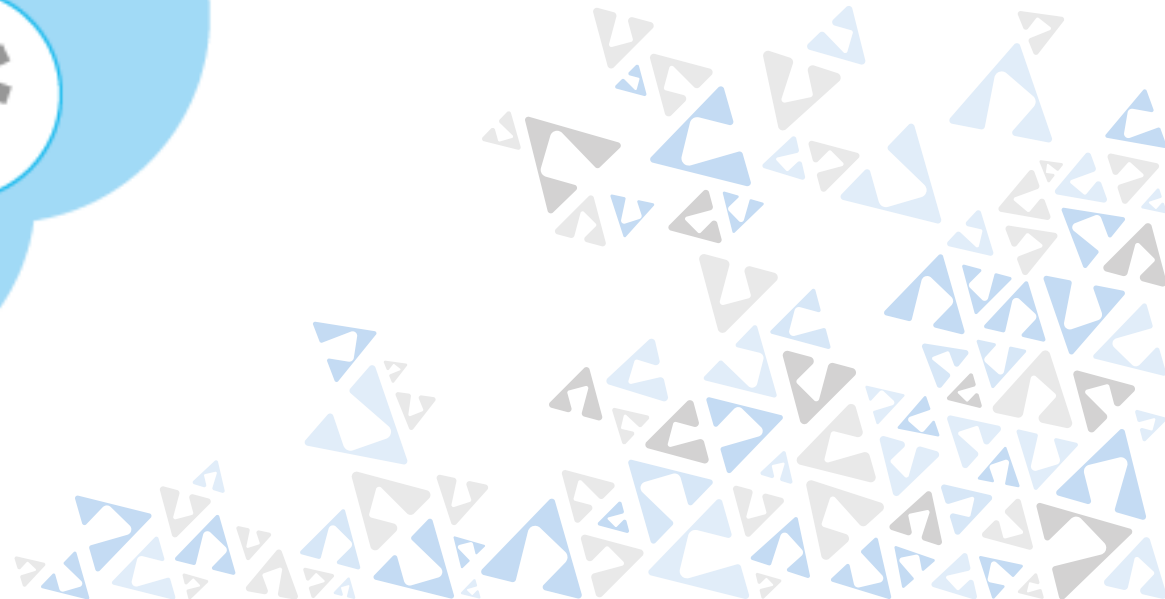
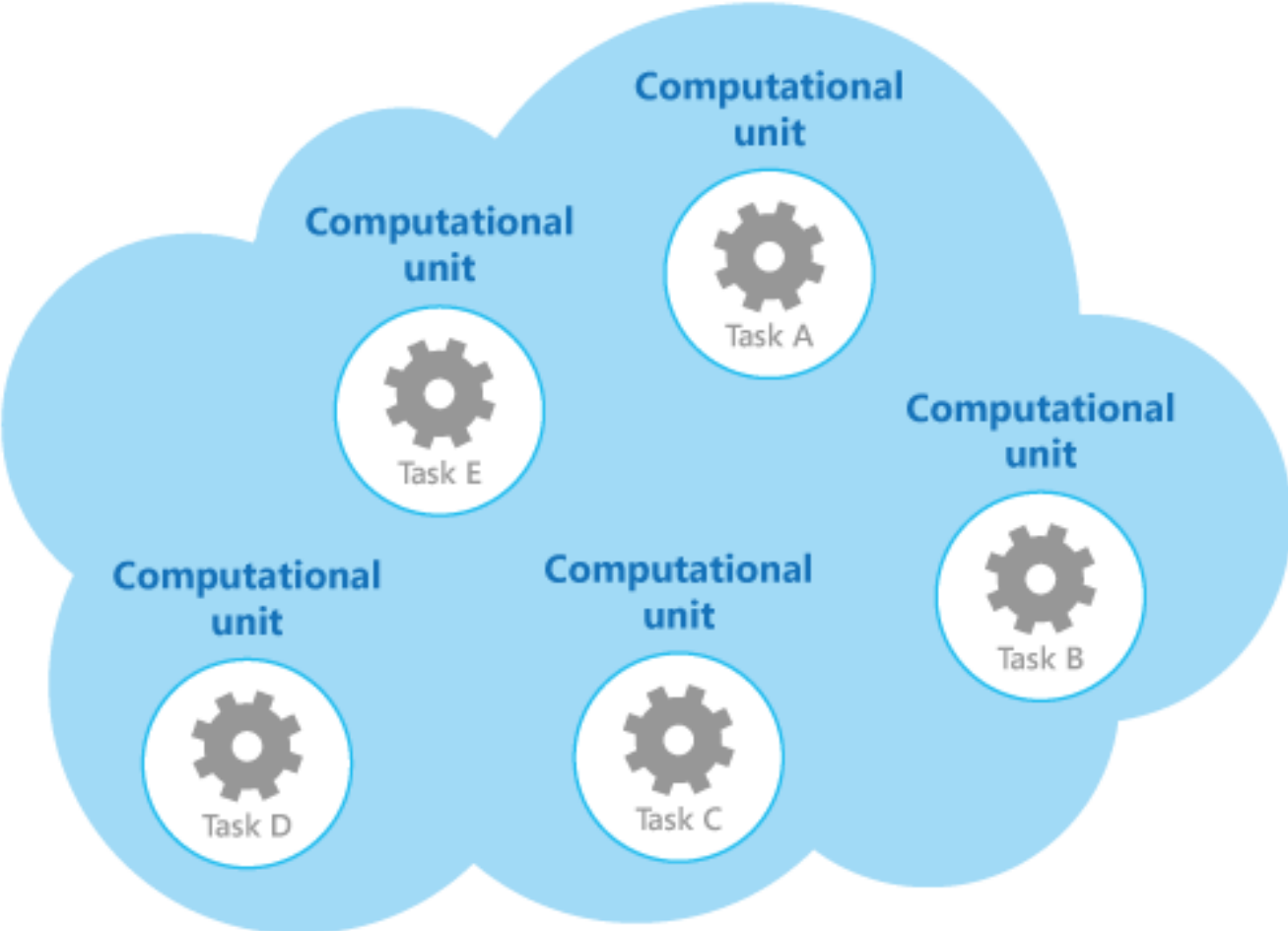
Strangler



Backends for Frontends

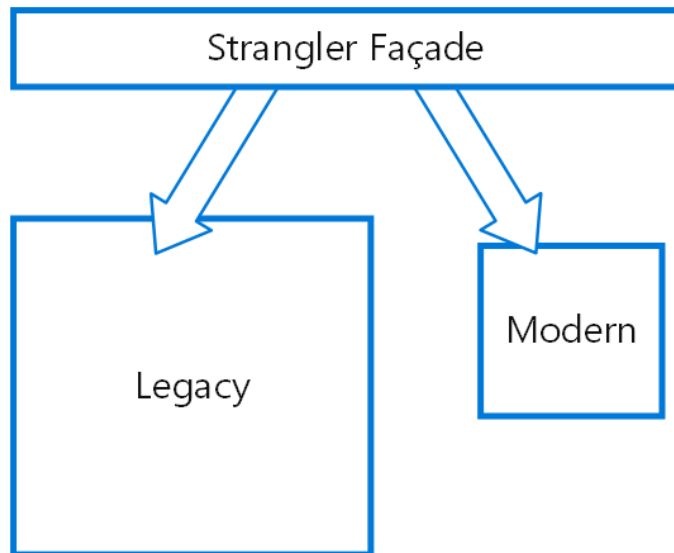


Compute Resource Consolidation

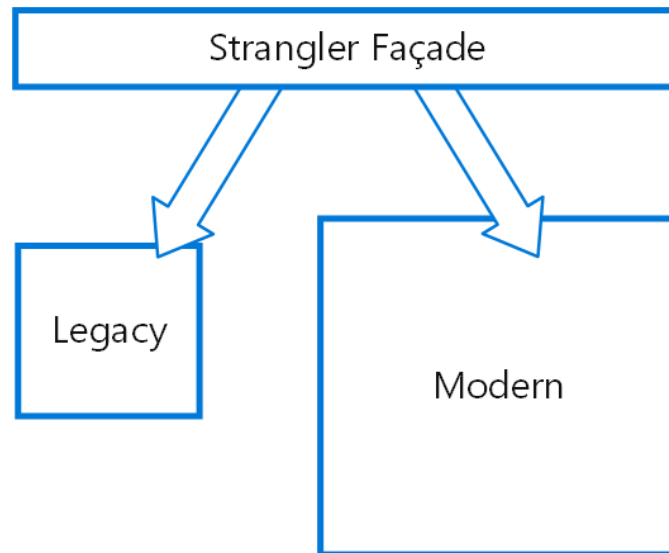


Strangler

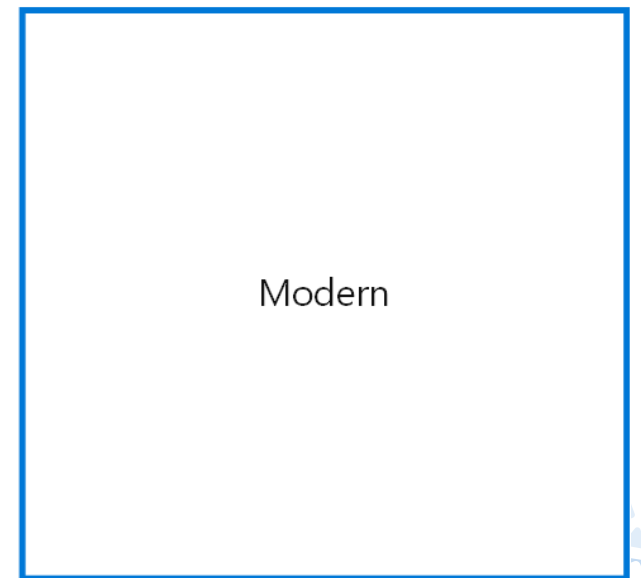
Early migration



Later migration



Migration complete

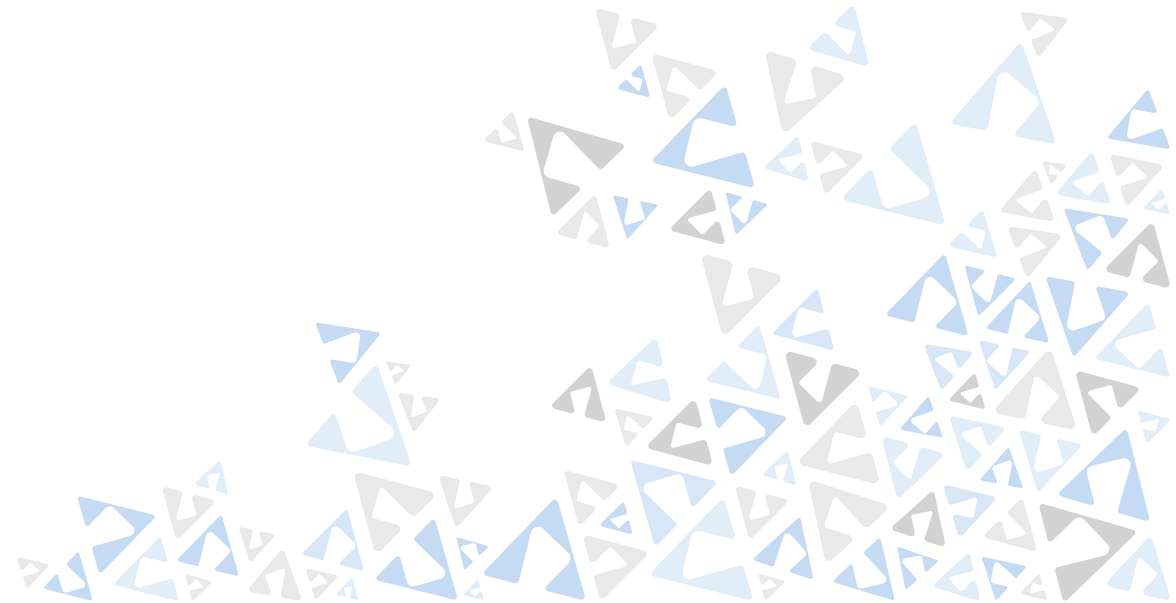


Security

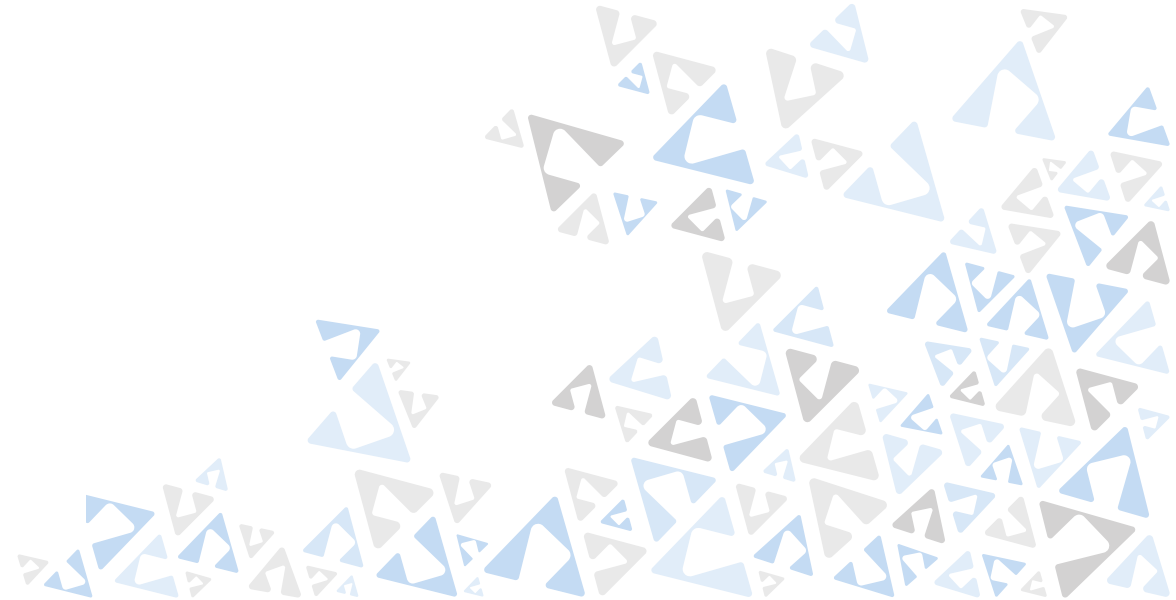
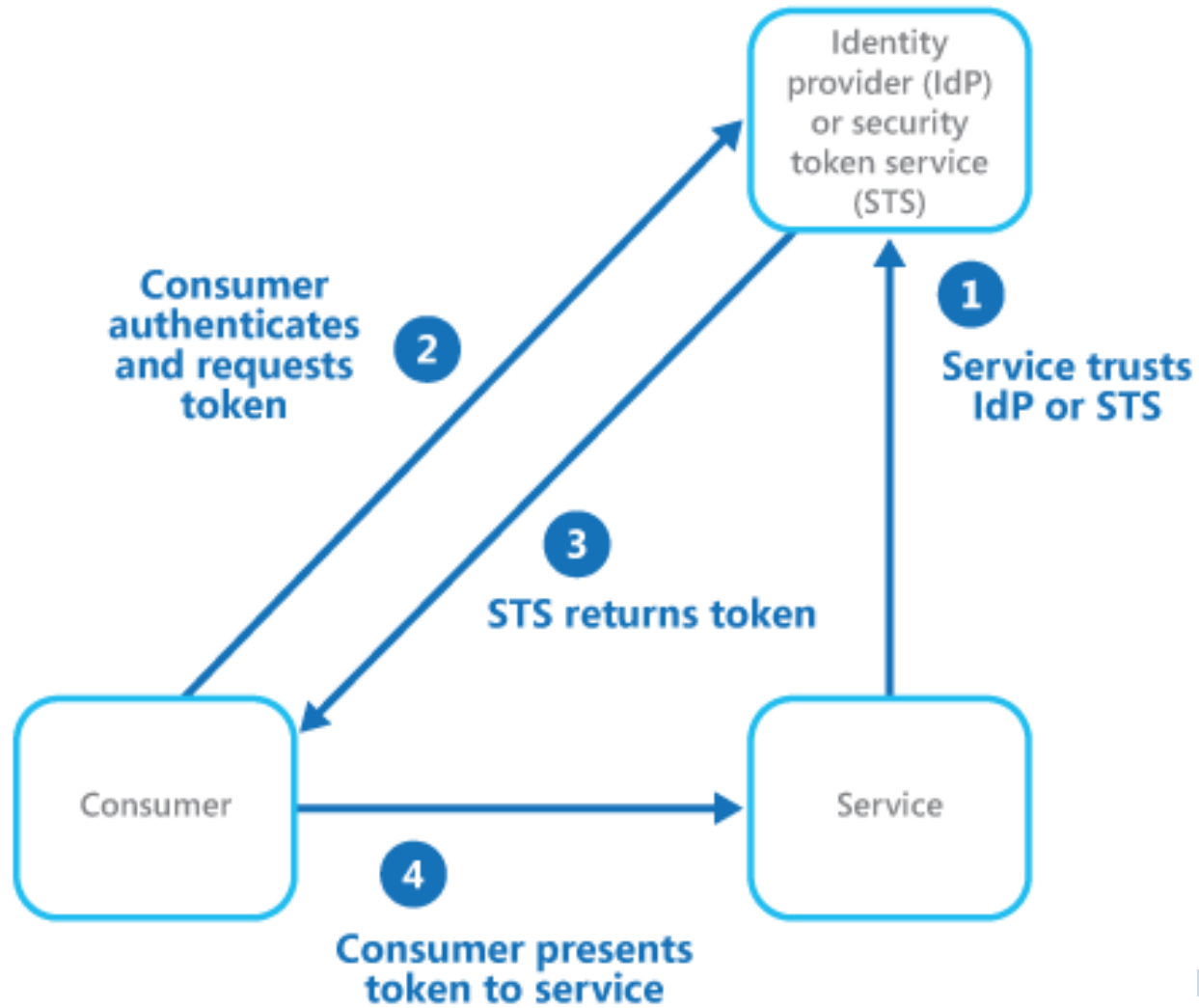
Federated Identity

Gatekeeper

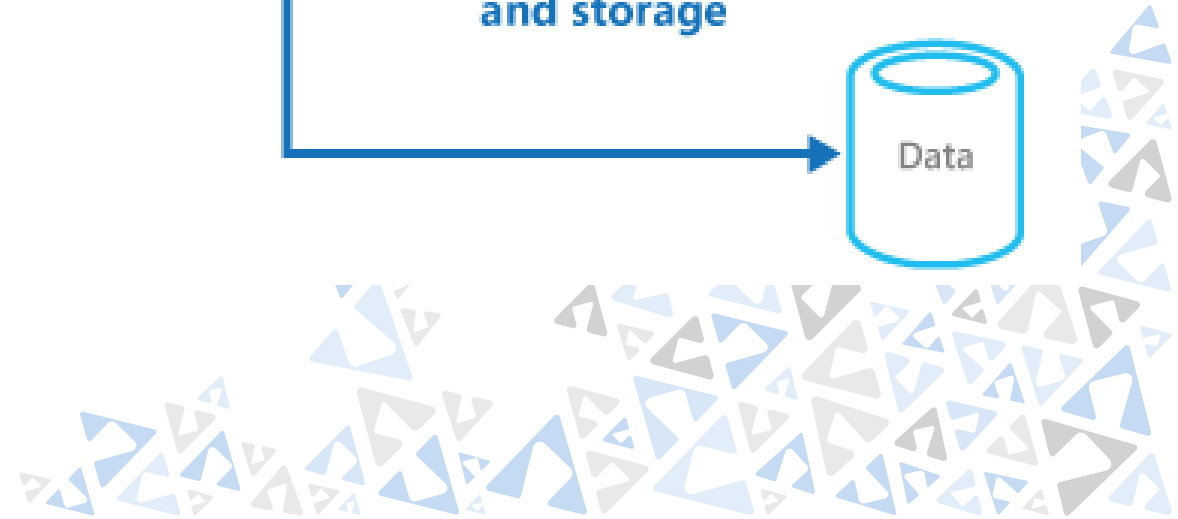
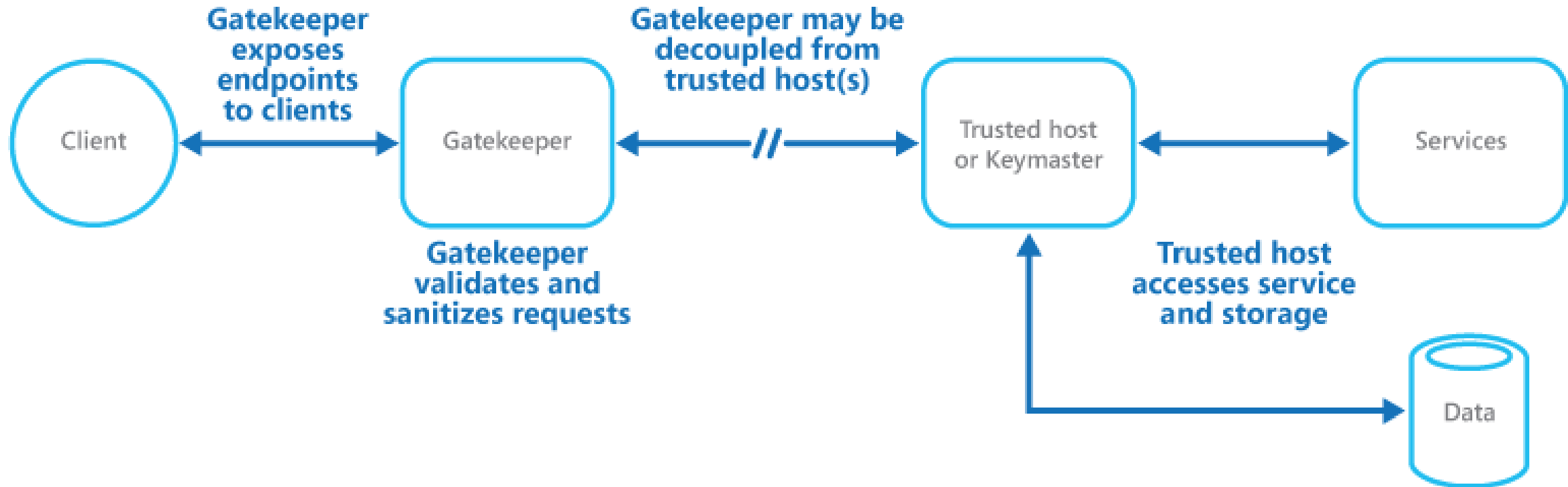
Valet Key



Federated Identity



Gatekeeper



Reference

Blog – HAVIT Knowledge Base

<http://knowledge-base.havit.cz/>

Twitter

[@RobertHaken](https://twitter.com/RobertHaken)

YouTube

<https://www.youtube.com/user/HAVITcz>

