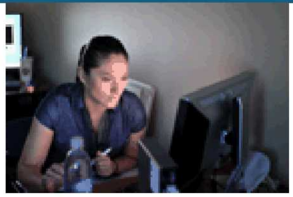
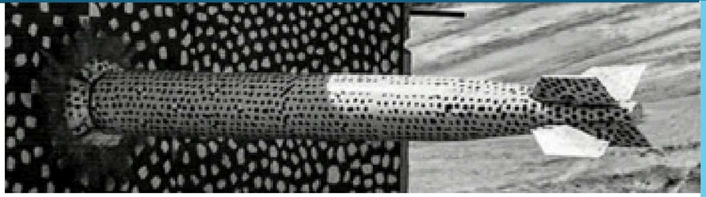
SAND2019-1812PE

GMS Architecture Overview

PRESENTED BY

Ryan Prescott

SAND2019-AAAAAA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

Approach

Overview

- Software
- Platform

Challenges

Architecture Approach

SAFe Context

The GMS project is using the Scaled Agile Framework (SAFe)

- Provides an approach to scale agile development practices for multi-team projects such as GMS
- Adopted at the start of development (PI 1)

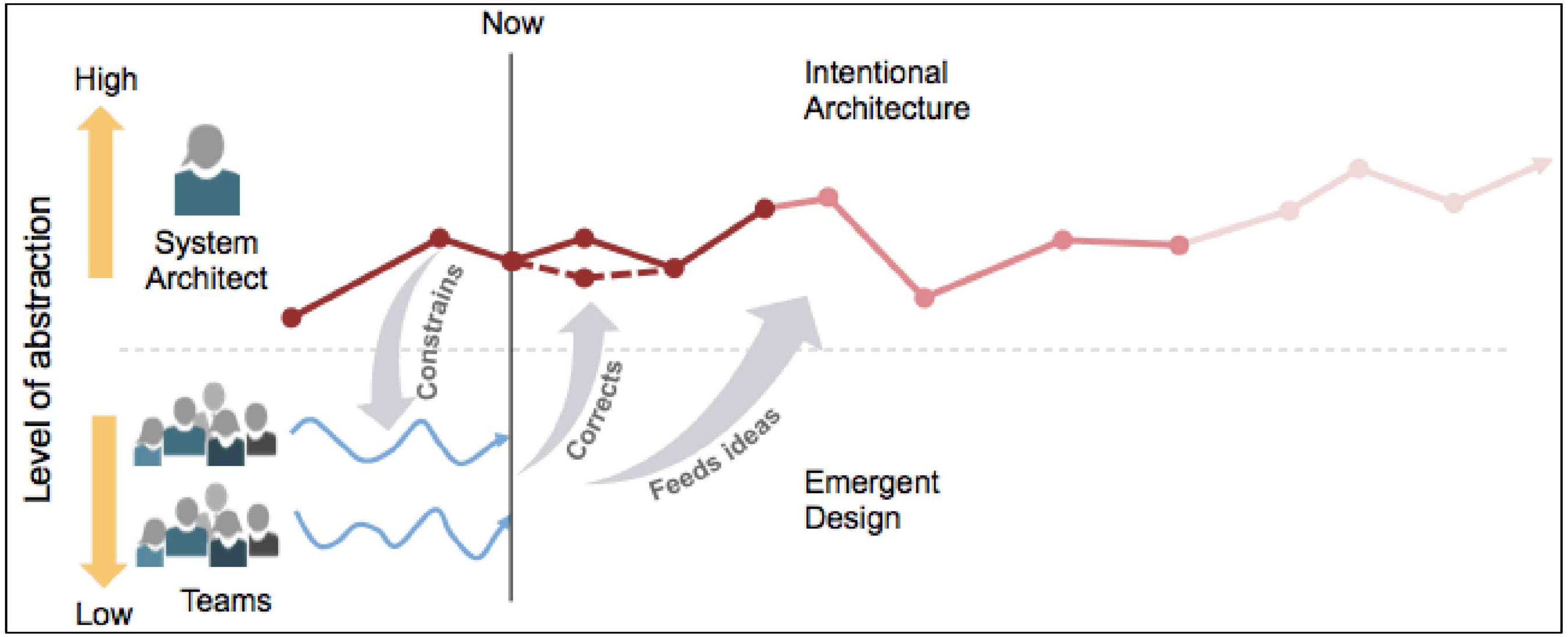
Agile Architecture principles and practices are part of SAFe

- **Goal**: active evolution of system architecture & design while business features are developed
- Intentional Architecture & Emergent Design
- Architecture Runway

Architecture in SAFe

The system evolves over the course of development through a balance between

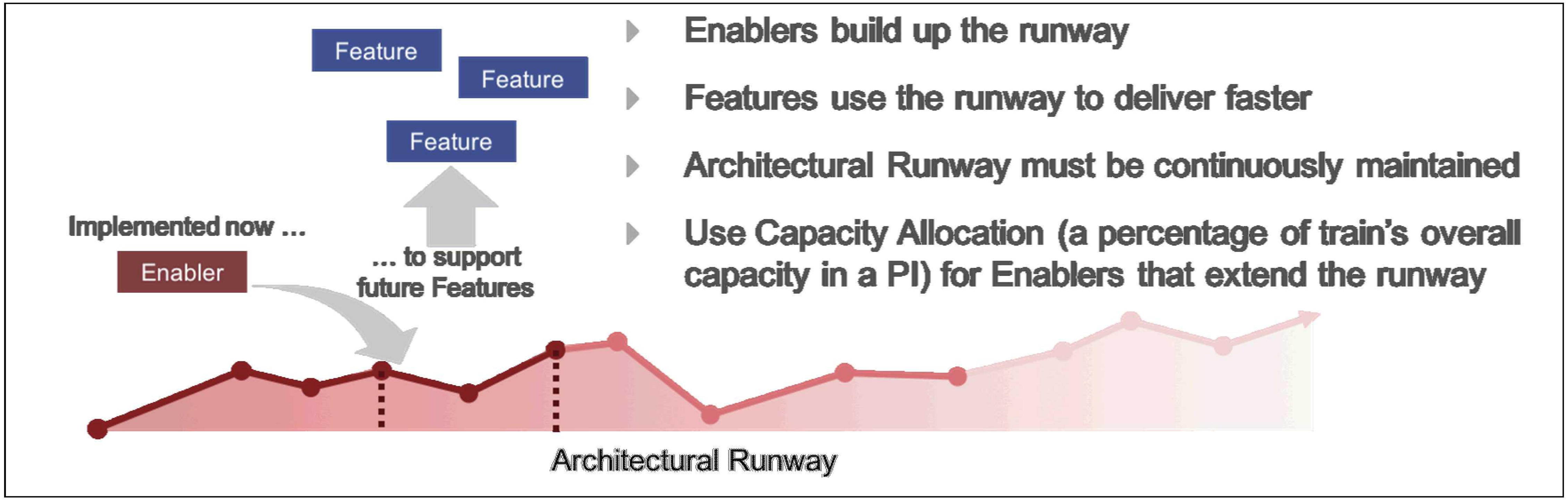
- **Intentional Architecture** – Planned initiatives to guide the overall System design across teams & features
  - Avoid redundant and conflicting designs
- **Emergent Design** – Development teams drive the design of solutions as part of feature development
  - Based on the Agile Manifesto principle: *The best architectures, requirements, and designs emerge from self-organizing teams*[1]



Architecture in SAFe

Architecture Runway: components, infrastructure, etc. needed to develop near-term features

- Developed by teams
- Guided by architects



- Enablers build up the runway
- Features use the runway to deliver faster
- Architectural Runway must be continuously maintained
- Use Capacity Allocation (a percentage of train's overall capacity in a PI) for Enablers that extend the runway

Architecture Specification

The GMS architecture is developed incrementally each PI to support feature implementation

The architecture is documented using a small set of views

- Representations of the system focusing on specific concerns
- Adapted from the *4+1 View Model*[1]

Logical View

- Focused on the high-level functionality provided by the System
- Overview diagrams (freeform), UML Class Diagrams and textual descriptions

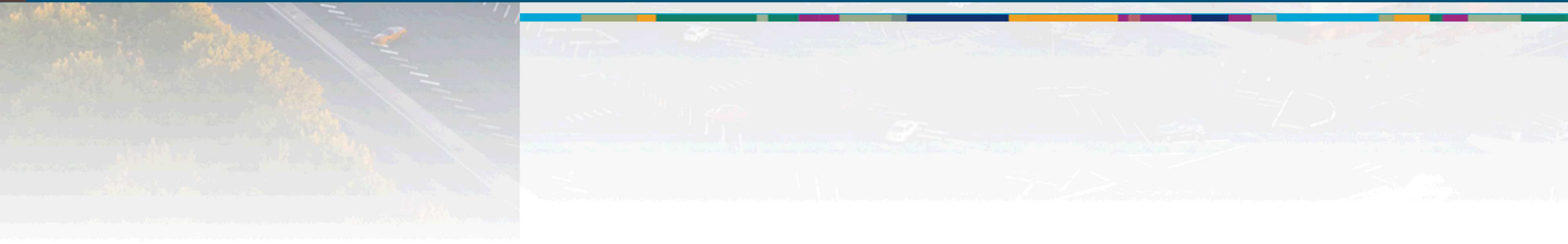
Deployment View

- Focused on the design of the physical/virtual platform and deployment of software components

Scenario View

- Use Case Realization model developed during the Elaboration Phase
  - Used as a reference, but not actively updated
  - Likely to be replaced with a small set of light-weight scenarios

Architecture Overview - Software

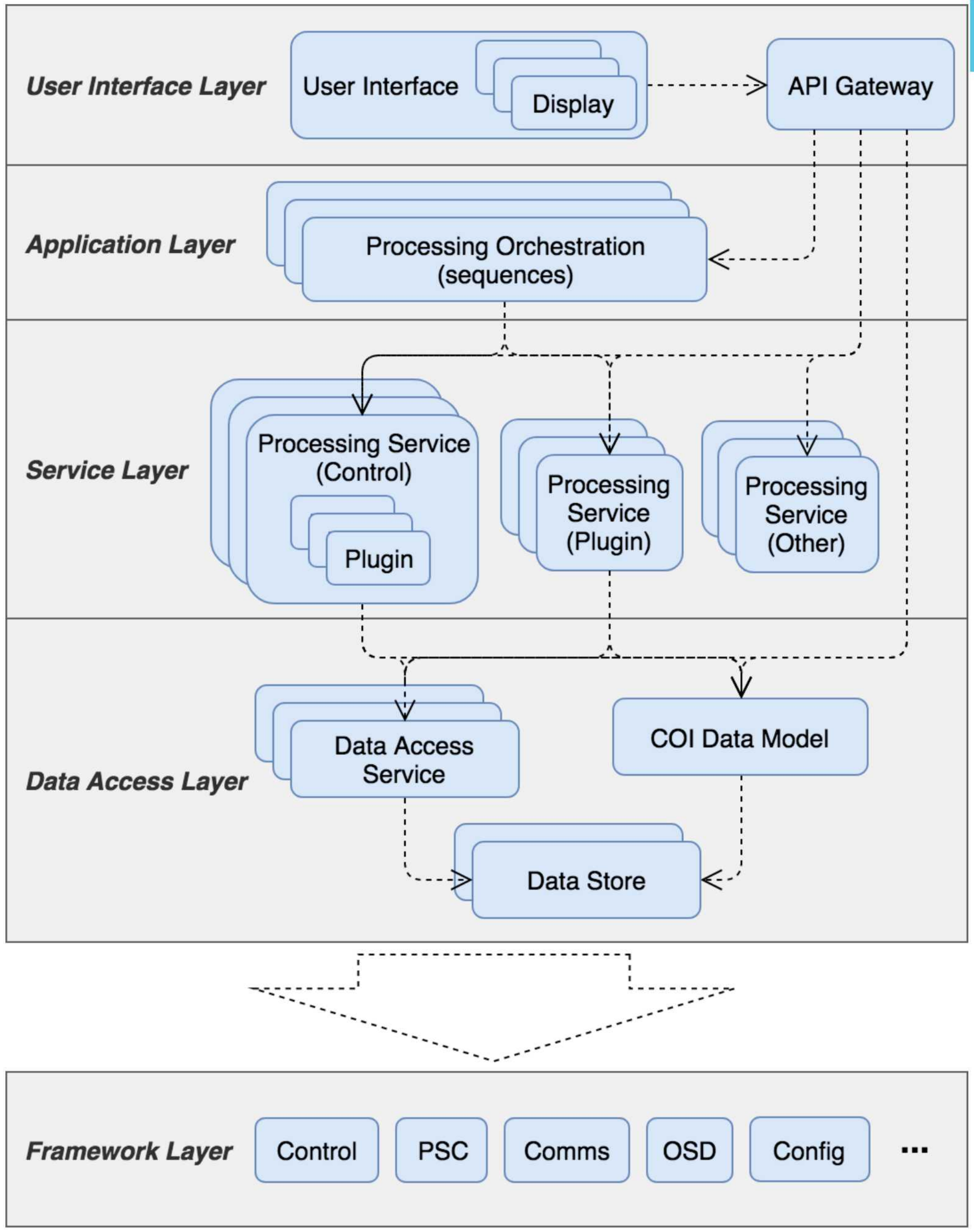


Logical View Layers

The GMS software architecture can be organized into a set of layers

- **User Interface** – The User Displays, organized into User Interfaces, together with supporting components
- **Application** - The orchestration of services into processing workflows that implement the core mission logic
- **Service** – Software functions supporting automatic processing workflows and User Interface interactions
- **Data Access** – Interface software providing access to persistent data while encapsulating the underlying storage implementation
- **Framework** - Shared project software providing common support functions and implementing standard patterns

Higher layers depend on lower layers



Framework Layer – Object Storage & Distribution

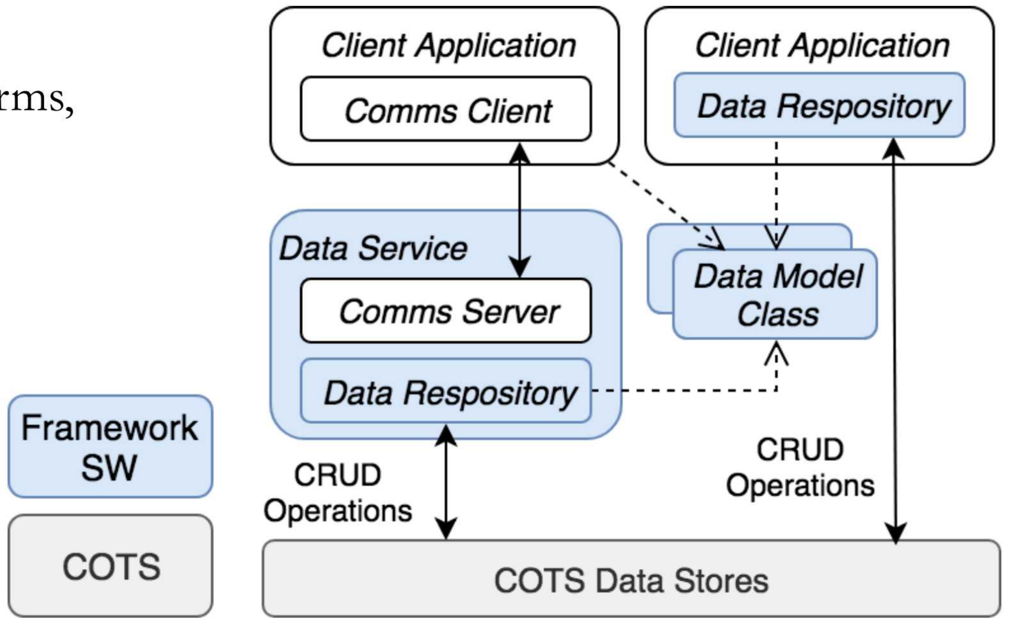
Supports access to persistent data in the System, while encapsulating the underlying storage technologies

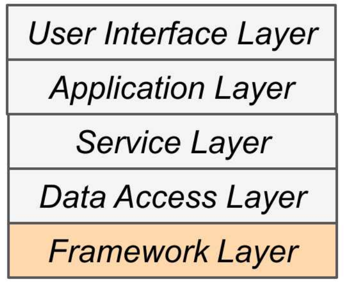
Includes

- **Data Model** – Object model for persistent GMS data
- **Data Services** – Access methods exposed via the GMS Comms framework (JSON/MessagePack over HTTP)
- **Data Repositories** – Access methods exposed as a Java API
- **Storage layer** – Underlying Storage technologies and data model mappings

Storage Layer Technologies

- **Postgres** – Station reference, data acquisition and processing results
- **Apache Cassandra** – Channel Segment data (raw & filtered waveforms, beams, FK timeseries)





Framework Layer – Comms

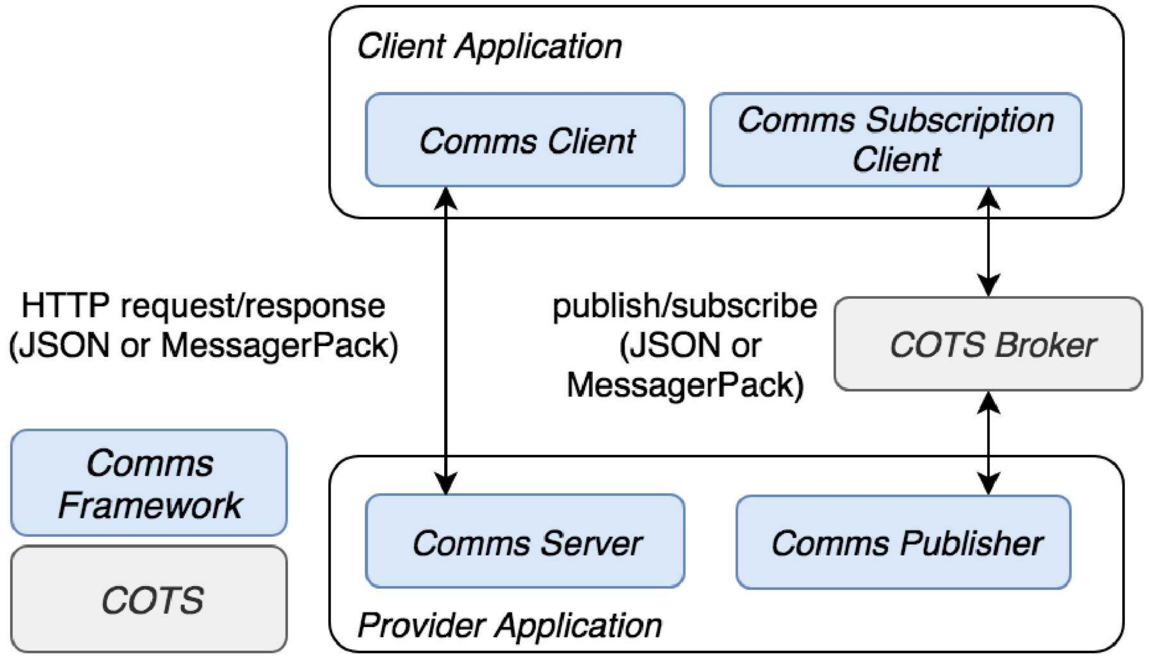
Supports network communication among components within and across layers of the System

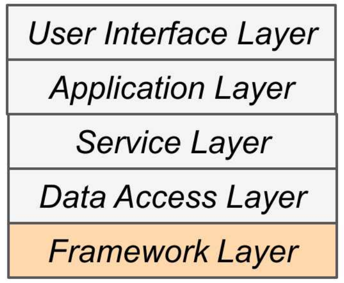
Includes
- Network protocols and data encodings supported by COTS
- Java client & server encapsulating protocols and encodings

Supported communication patterns
- Request/Response
- Publish/Subscribe

Technologies
- Request/Response: JSON & MessagePack over HTTP
- Publish/Subscribe: TBD

Enables integration of components implemented in multiple languages


Client Application
Comms Client
Comms Subscription Client
HTTP request/response (JSON or MessagerPack)
publish/subscribe (JSON or MessagerPack)
COTS Broker
Comms Framework
COTS
Comms Server
Comms Publisher
Provider Application



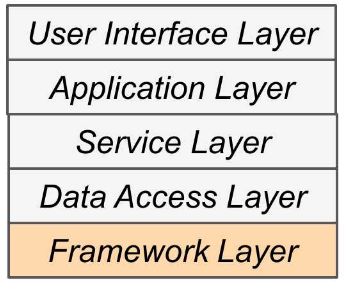
Framework Layer - Configuration

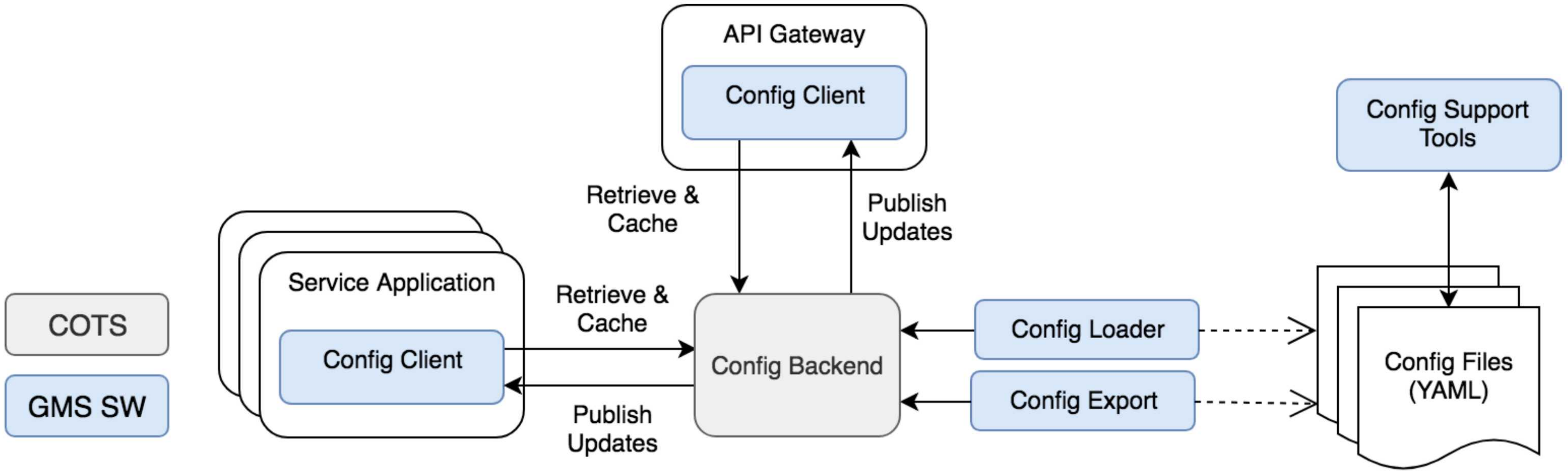
Supports definition, loading, and runtime access to configuration settings

- Offline definition in YAML files, supported by config tools and versioned as part of the GMS release process
- Loaded into persistent data stores as part of the runtime System
- Modifiable within the runtime System via the User Interface

Two types of configuration

1. **System Configuration** - Settings used for general application setup (e.g. DB connection parameters, logging levels)
2. **Processing Configuration** - parameters used to configure processing operations such as filtering, beaming, FK, detection, association. Parameters can be defined and accessed for combinations of selection criteria (e.g. station, channel, phase, source region, time of year, etc.)





Framework Layer - Control

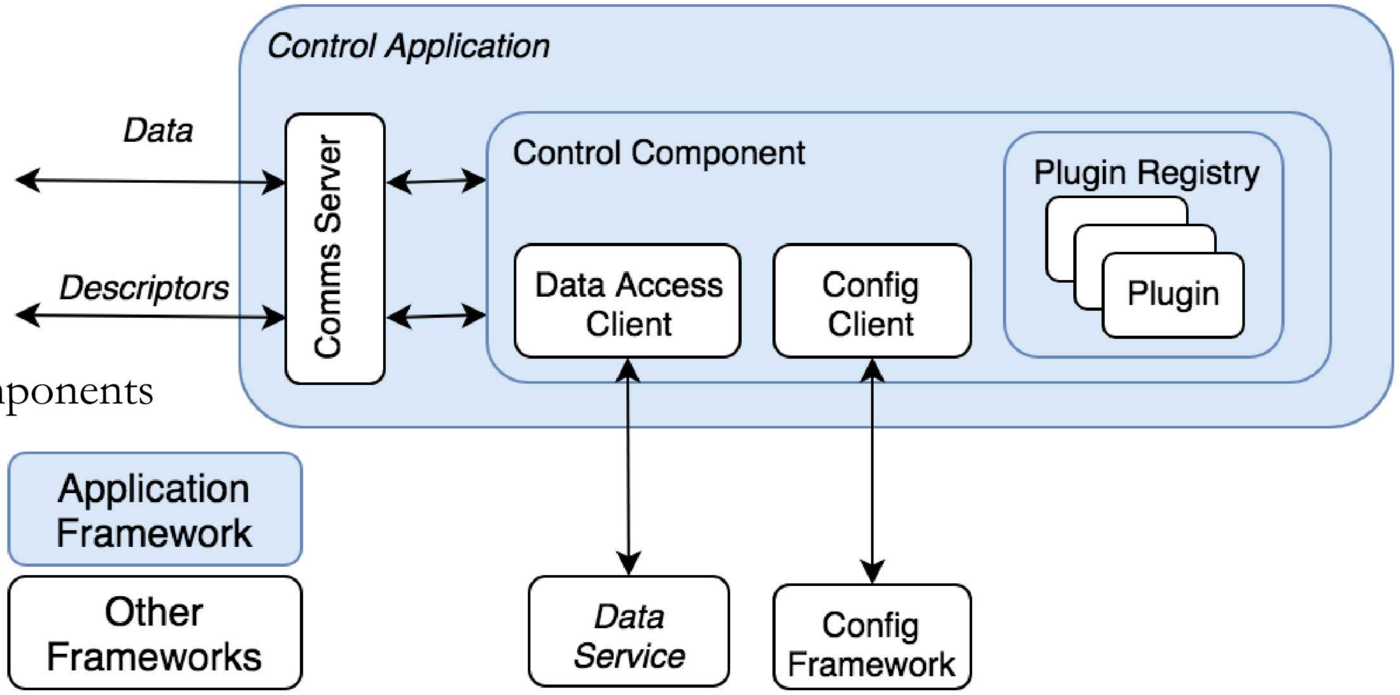
Provides standard patterns and Java software supporting development of extensible components deployed either as libraries within an application (e.g. processing workflow), or as standalone applications accessed via service interfaces

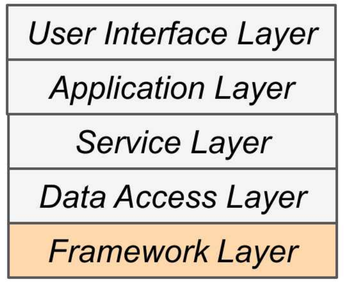
Control Components

- Entry point for automatic processing business logic
  - Filtering, beaming, FK, detection, association, location, magnitude, etc.
  - Accessed from automatic processing sequences and UI
- Independent of other Control components
  - Develop and replace in insolation

Plugins

- Implement algorithms
- Extension point for new algorithm implementations
- Deployed as selectable algorithm within Control components





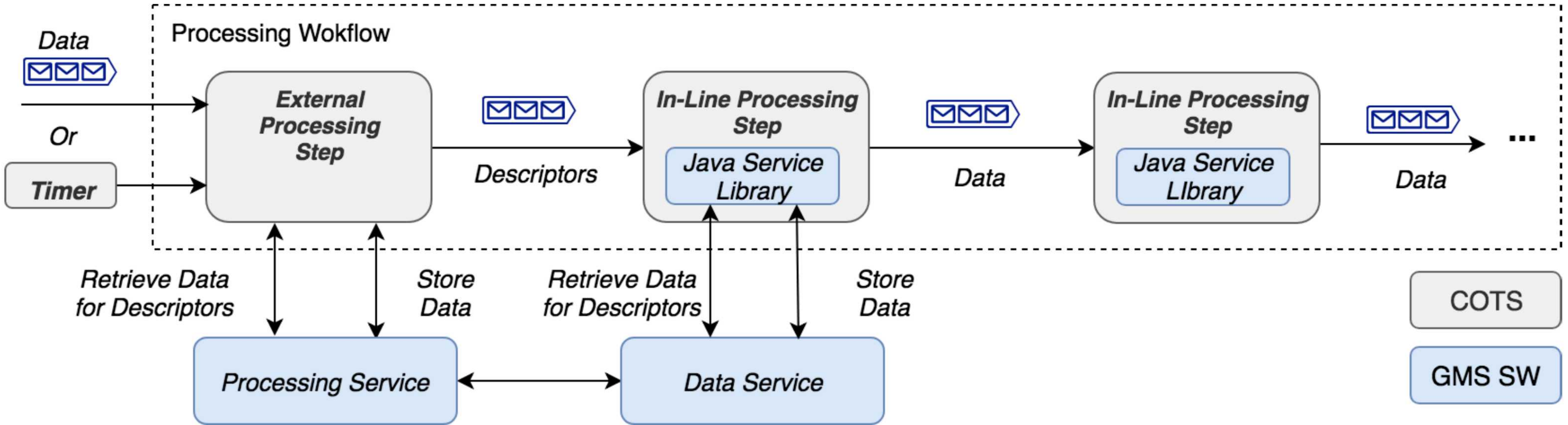
Framework Layer – Processing Sequence Controller

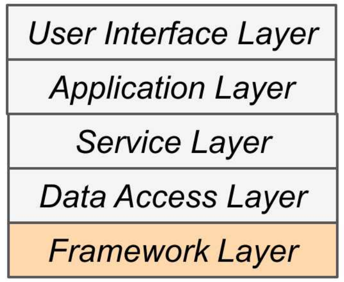
Supports orchestration of services into processing flows (e.g. station processing, network processing)

- Key extension point
- Processing flows support both automatic processing and User Interface interactions
- Supports multiple flow types, deployment models and data passing models
  - Data-driven and interval-driven flows
  - In-line and external service models
  - Direct data passing (in-line or via network service) and Descriptor passing (similar to claimcheck pattern)

Implemented using Apache NiFi

- Open-source orchestration framework for automated data processing flows
- Designed for scaling, fault tolerance, extensibility, security





Data Access Layer

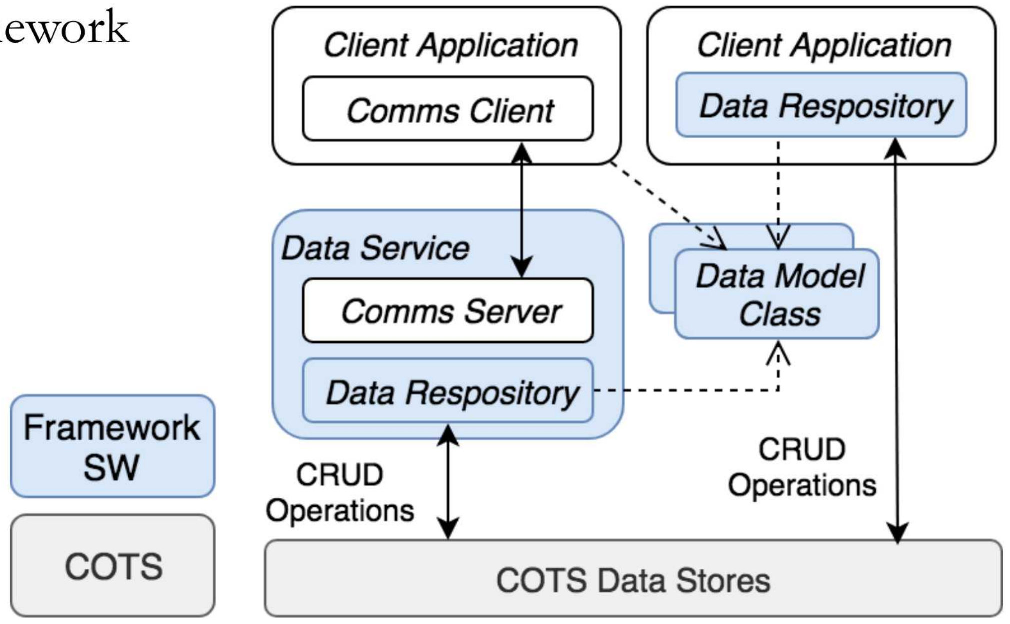
Provides access to persistent data in the System via a Common Object Interface (COI) consisting of:

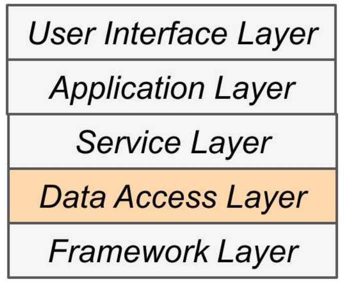
- Shared data model (Java classes, JSON or MessagPack)
- Data services for access via the GMS Comms framework (HTTP)
  - Enables cross-language integration
- Java clients direct access (native protocols)

Encapsulates the underlying storage technologies, including physical schemas, query languages, etc.

Supports access from the User Interface, Application, and Service layers

Implemented via the Object Storage & Distribution (OSD) framework





Service Layer

Processing functions available via well-defined interfaces

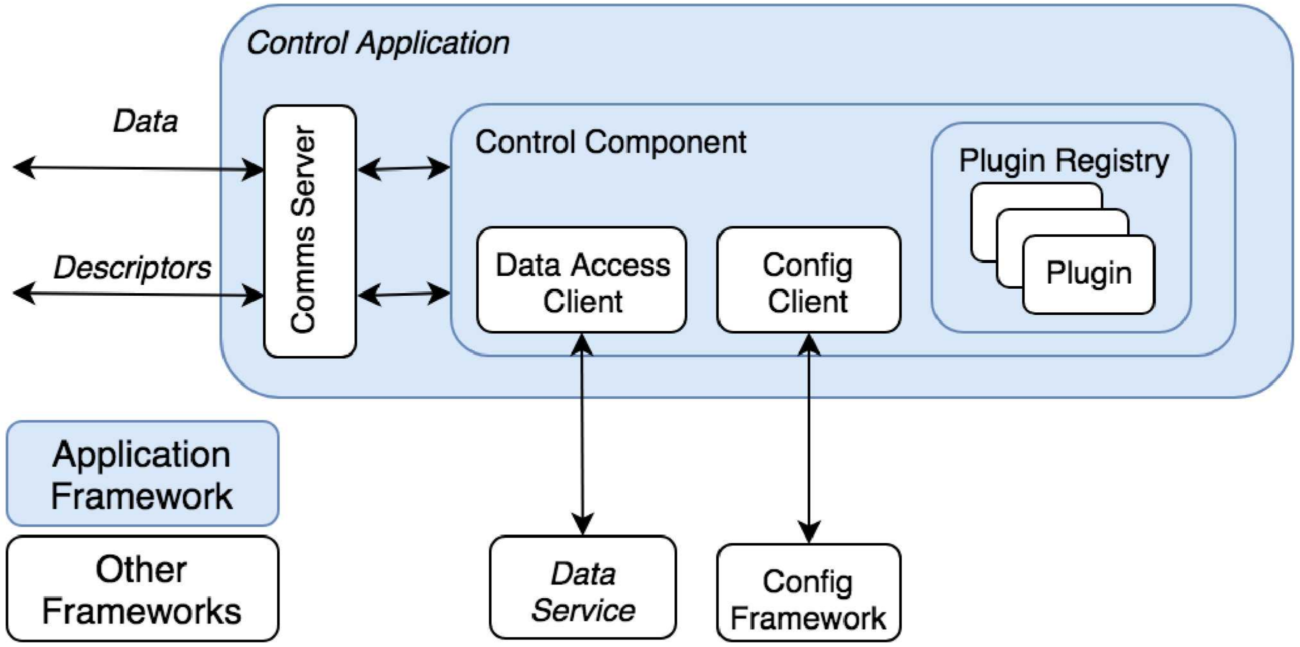
- Key extension points
- Orchestrated into processing workflows in the Application layer via the PSC framework
- Invoked from the User interface layer to support user interactions
- Implemented via the GMS Control framework

Network Service Interfaces

- Support direct data passing and descriptor pattern (JSON or MessagePack over HTTP)

Examples

- Waveform QC, Filtering, Beaming, Signal Detection, Signal Detection Association, Event Location, etc.



User Interface Layer
Application Layer
Service Layer
Data Access Layer
Framework Layer

Application Layer

Orchestration of services into processing flows (e.g. station and network processing)

- Primarily Control components from the Service Layer

Supports automatic processing and User Interface interactions

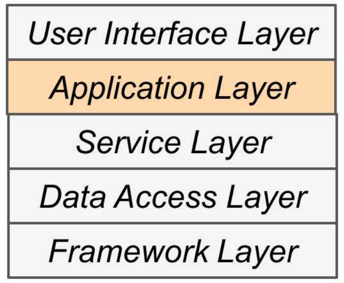
Components may be embedded within the processing flow runtime or deployed as external applications invoked via network interfaces

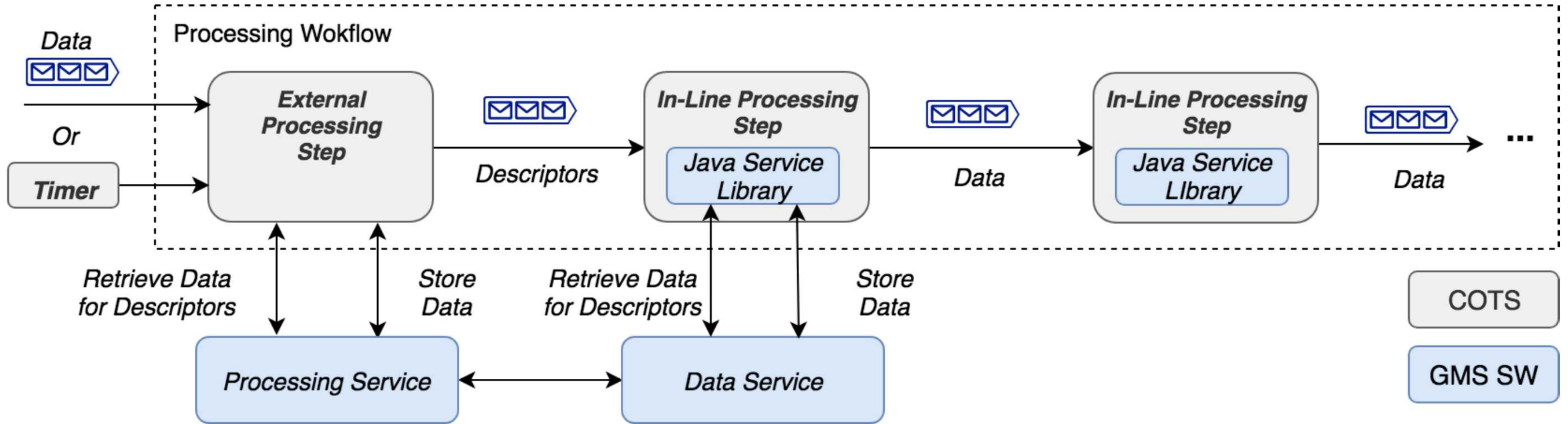
- External application model enables cross-language integration

Supports multiple flow types, deployment models and data-passing models

- Data-driven and interval-driven flows
- In-line and external service models
- Direct data passing (in-line or via network service) and descriptor passing (similar to claimcheck pattern)

Implemented via the Processing Sequence Controller framework (PSC)





UI Layer – Overview

GMS User Interface composed of Displays, supported by an API Gateway backend

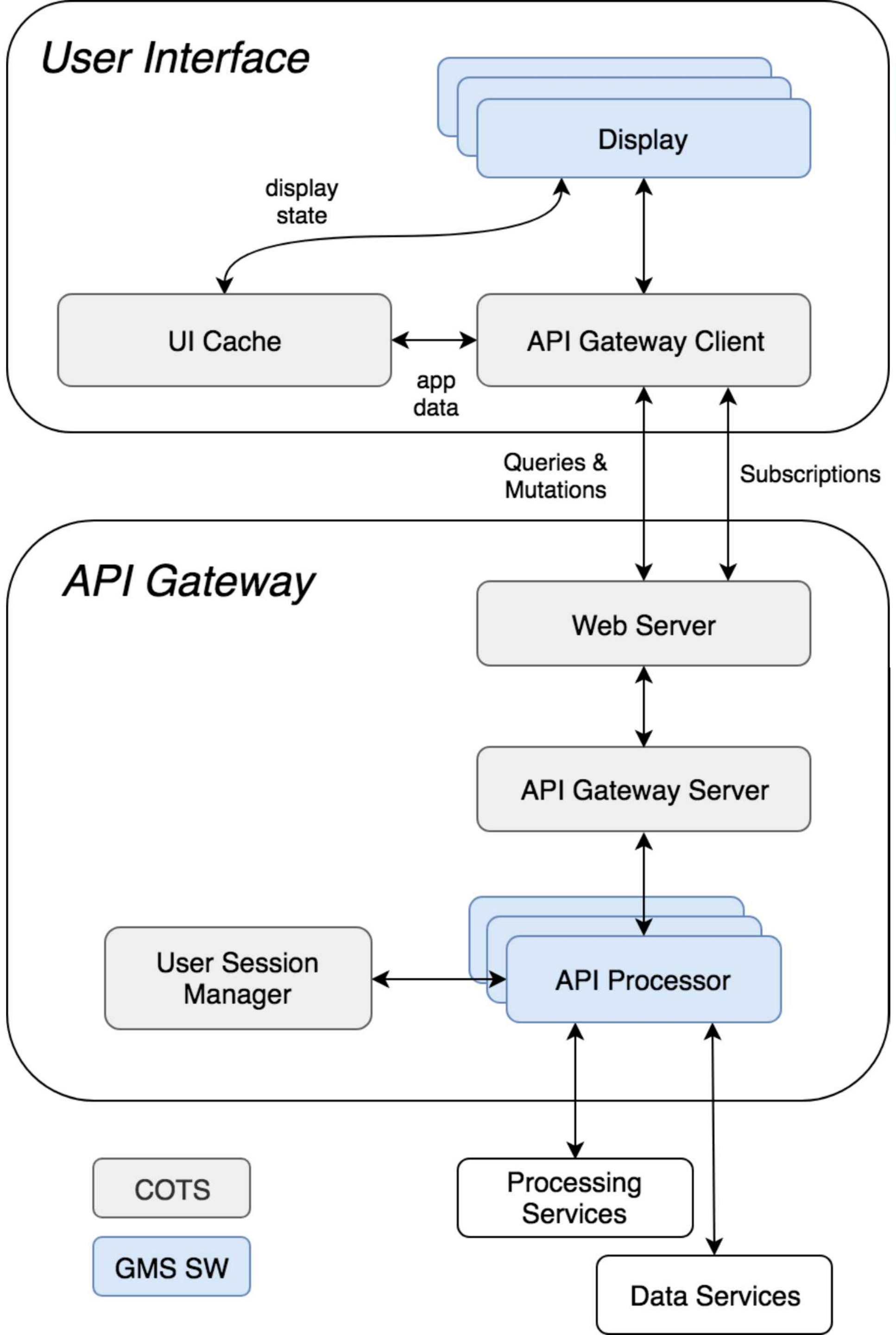
- Current focus: Analyst User Interface

Displays support User interactions with the System

- E.g. Waveform Display, Signal Detection List, Event List, Map Display, etc.

The API Gateway mediates interactions with lower layers (Application, Service, Data Access) and provides UI support functions

- Standard pattern popularized as part of Microservices architectures



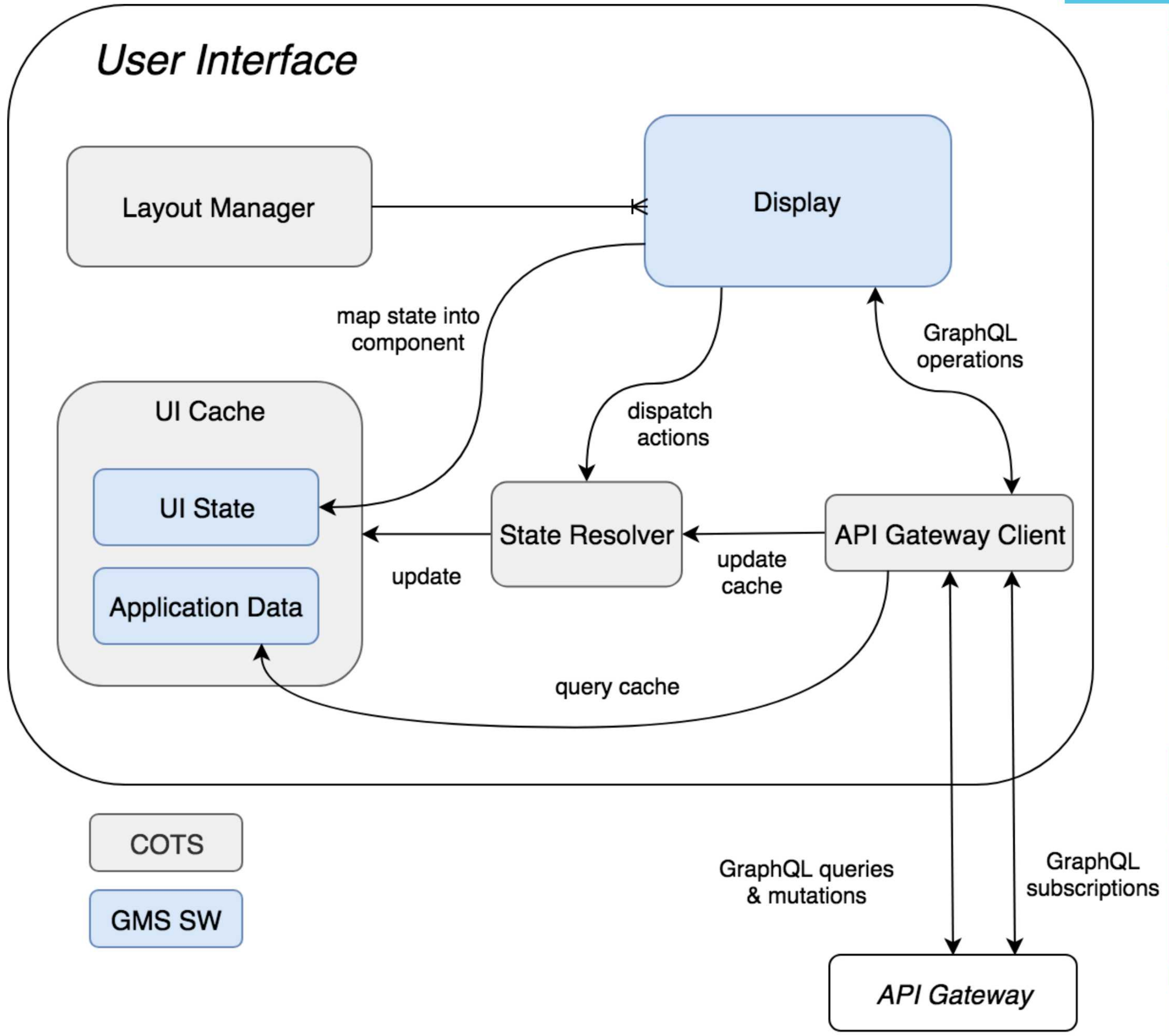
User Interface Layer
Application Layer
Service Layer
Data Access Layer
Framework Layer

UI Layer - Displays

Displays implement a shared design pattern
- Layout controlled via a Layout Manager
- Access to shared UI state and application data via a UI Cache
- Access to processing functions (Application & Service layers) and persistent data (Data Access Layer) via the API Gateway

Technologies
- Language: Typescript
- Web UI Framework: React
- Layout Manager: Golden Layout
- UI Cache: Redux
- API Gateway Client: Apollo GraphQL client

User Interface Layer
Application Layer
Service Layer
Data Access Layer
Framework Layer



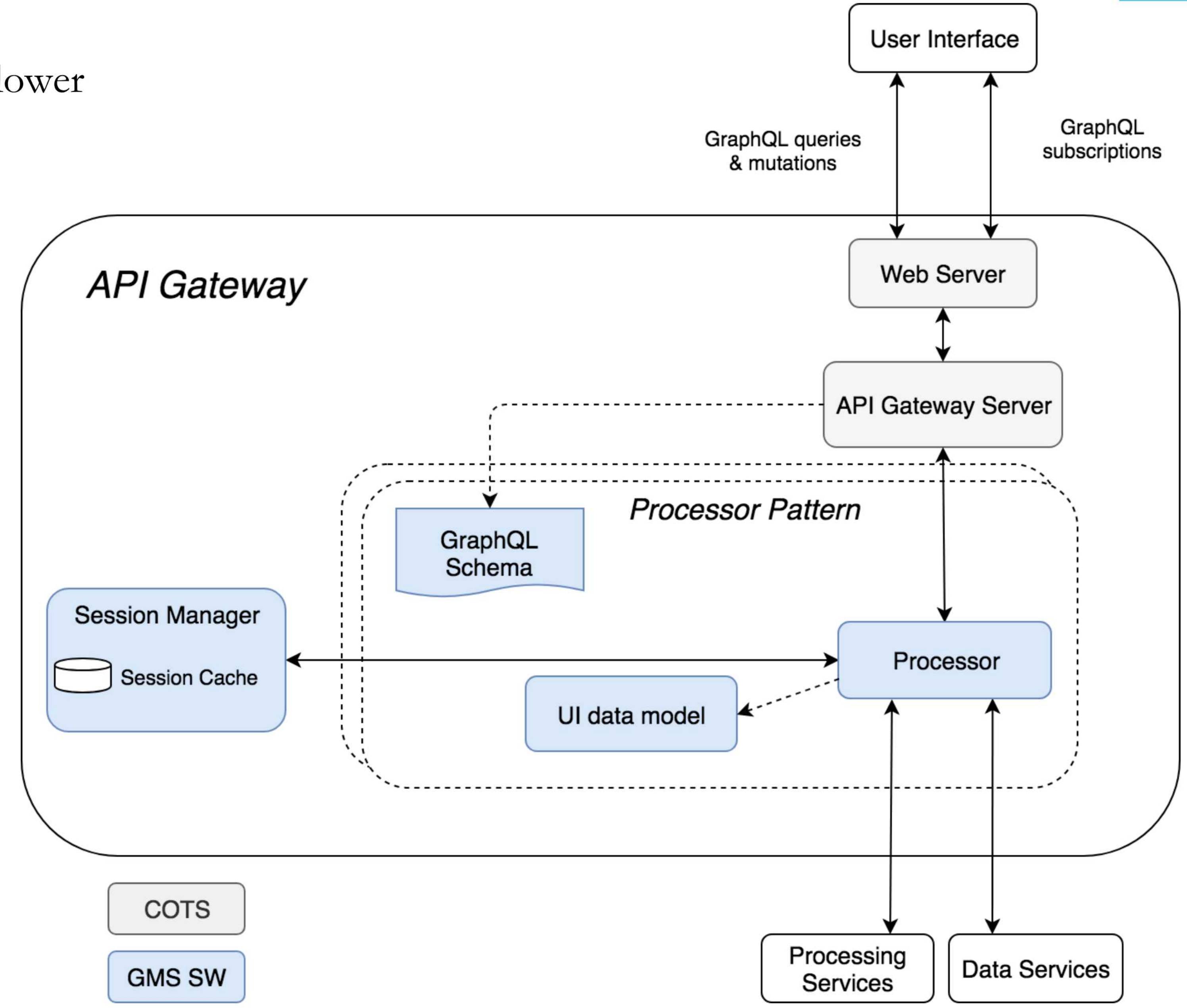
UI Layer – API Gateway

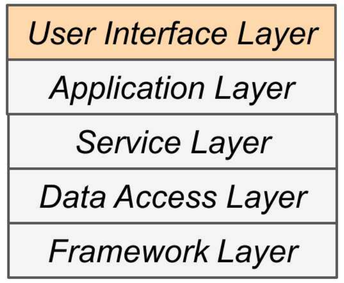
Intermediary between the User Interface and lower layers (Application, Service, Data Access), providing:

- API consolidation, routing & composition
- Protocol translation
- Data model views
- User authentication & session management
- Mocked backend

Technologies

- Language: Typescript
- Application framework: Node.js
- API: GraphQL (Apollo)
  - GraphQL is an open-source data access language





Architecture Overview - Platform

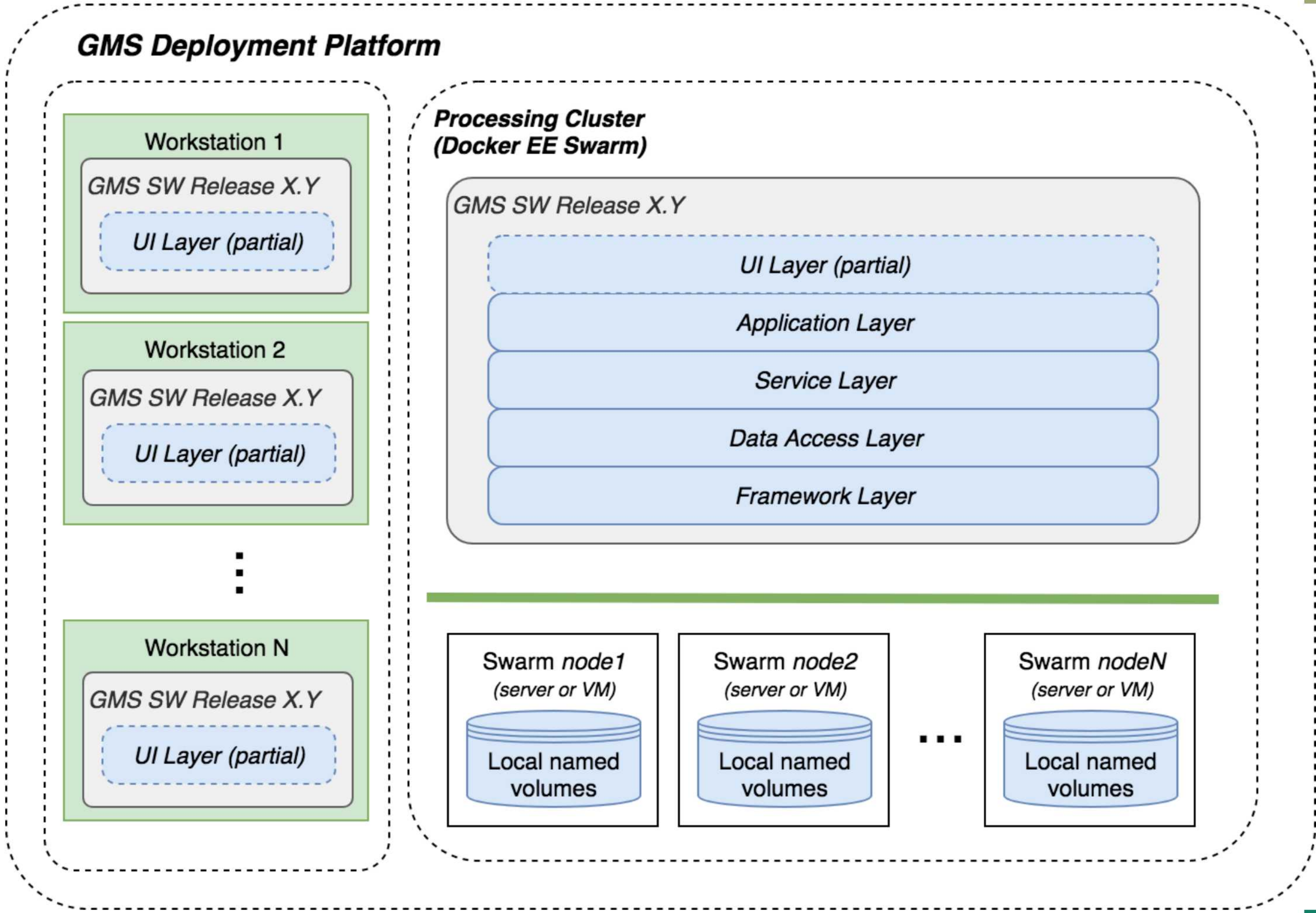
GMS Software Deployment

Docker containers provide the standard deployment unit for GMS software components

- Docker is a framework for packaging and deploying application components
- Docker Compose/Swarm is a framework for defining, deploying, scaling and monitoring distributed container-based applications
- Docker EE is a commercial distribution of Docker frameworks with licensed support

Each GMS deployment includes two primary elements:

- A Processing Cluster (VM or server-based) built using Docker Swarm
- A set of UI workstations

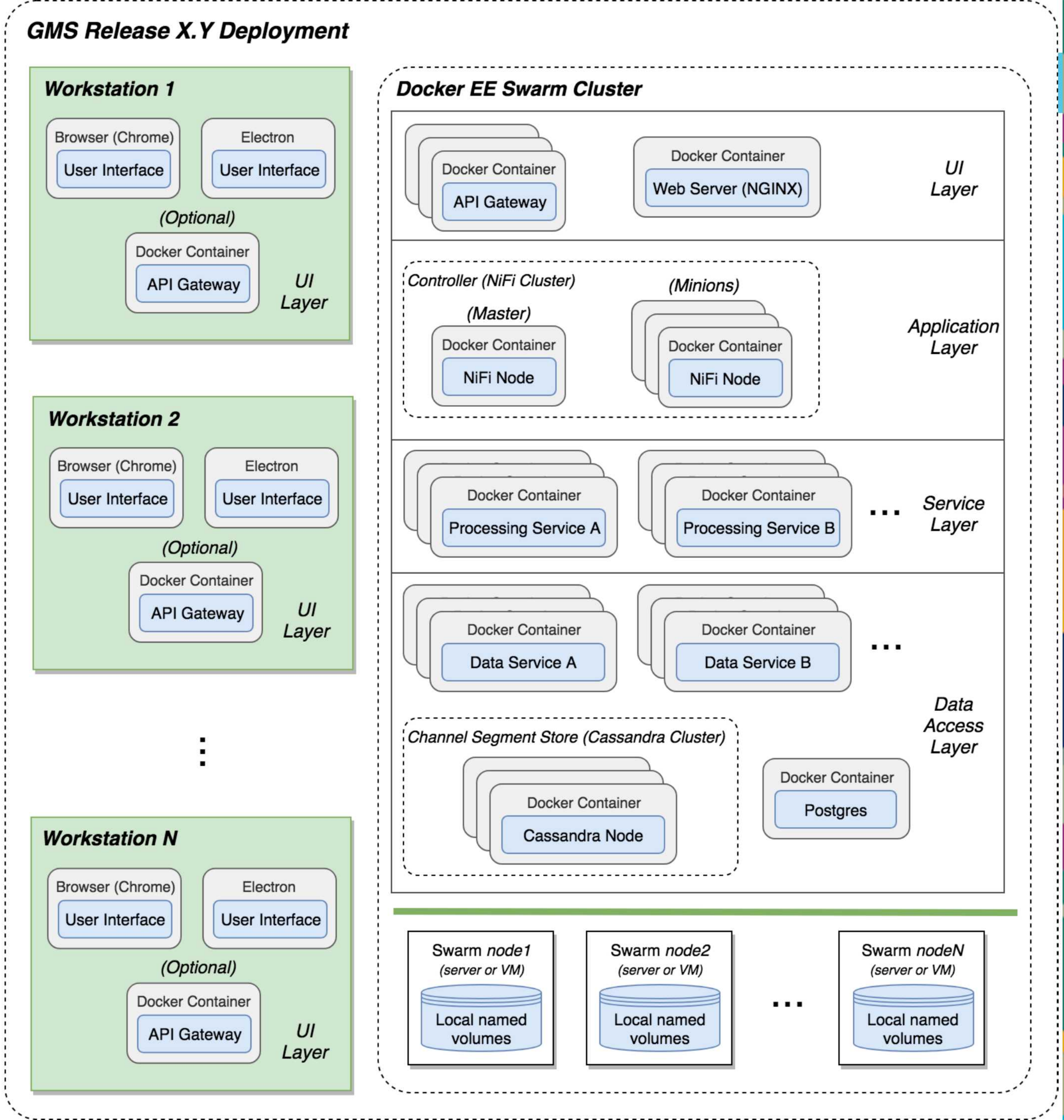


GMS Software Deployment

The Processing Cluster hosts the following layers of the application:

- **UI** (partial) – Includes the API Gateway and web server
- **Application** – The controller NiFi cluster with deployed processing workflows
- **Service** – The set of replicated services implementing GMS processing functions
- **Data Access** – The data stores (Postgres and Cassandra), together with data services & clients supporting access from the layers above

Workstations host part of the **UI** layer, including the browser & Desktop clients, together with an optional copy of the API Gateway



Platform Evolution

GMS platform development started with the Red Hat OpenShift container platform

Problems uncovered during early development caused the project switch to the Docker EE platform in PI 6

- OpenShift is an enterprise-scale product that is difficult to deploy and support for a single project
  - High license cost
  - Substantial support effort required (System team, consultants)
- Misalignment of core features
  - Source-to-image designed for single container per service per repository

Docker EE experience so far

- Significantly reduced platform complexity
- Simplified deployment model

Transition

- PI 6: Single-node testbed deployment (Docker Compose)
- PI 7+: Clustered testbed deployment (Docker Compose, Docker Swarm)

Architecture Challenges

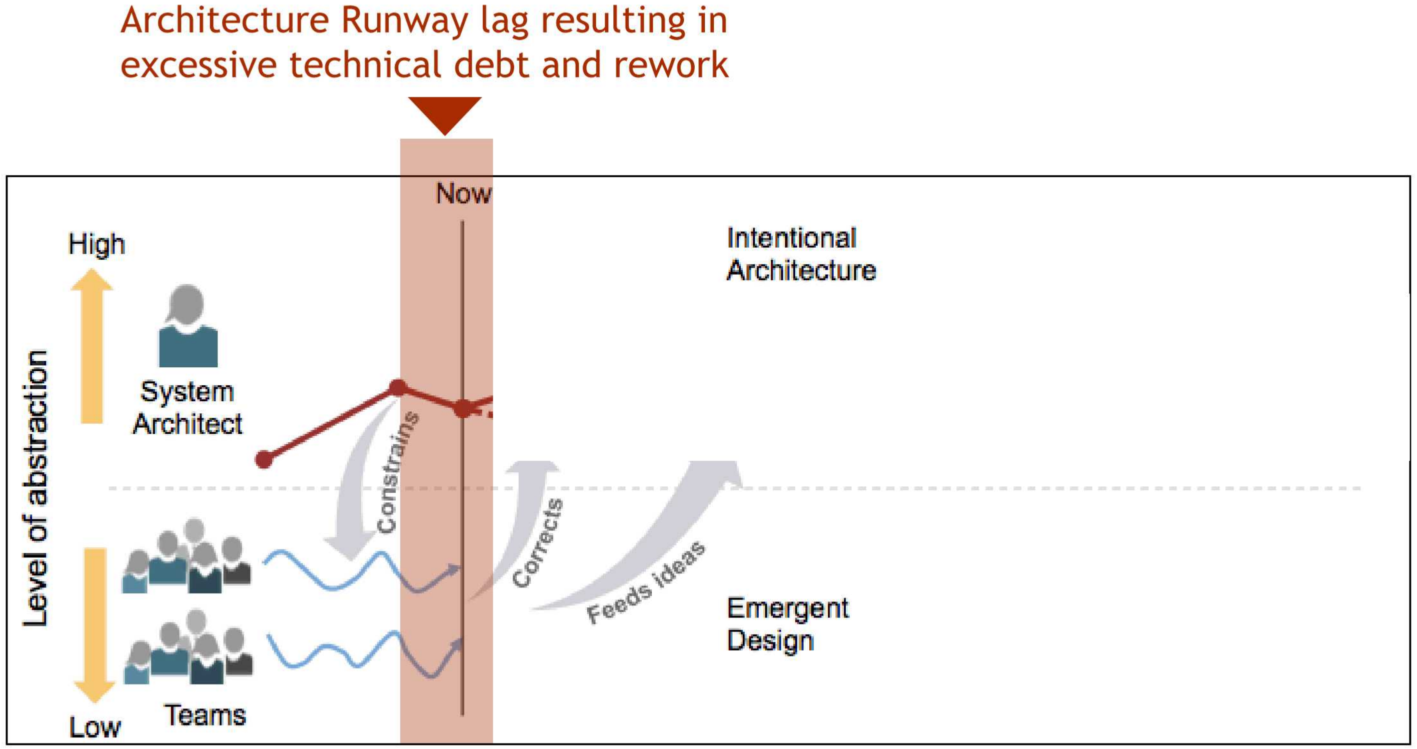
Architecture Challenges

Architecture Runway

- The project has struggled to establish and maintain an architecture runway
- Lack of shared software frameworks & standardization has hindered development

Intentional Architecture vs. Emergent Design

- Over-reliance on emergent design early in development
- Insufficient architecture guidance has hindered development



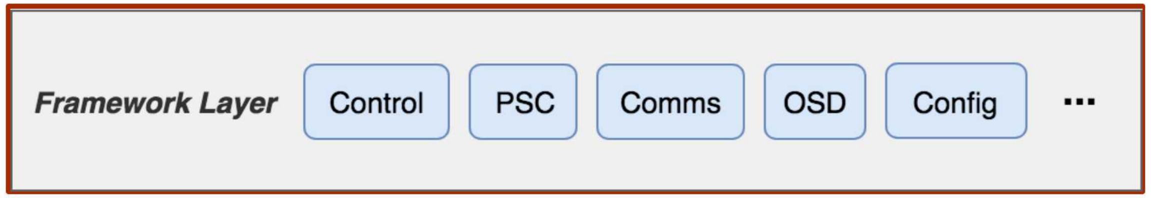
Project Response

Established a dedicated Architecture team in PI 5 focused on providing more rigorous architecture guidance further in advance

- Higher-formality documentation and team hand-offs

Established a dedicated Frameworks team in PI 7 to accelerate development of the architecture runway

- Shared frameworks improve reduce complexity and effort to develop application components



Questions

References

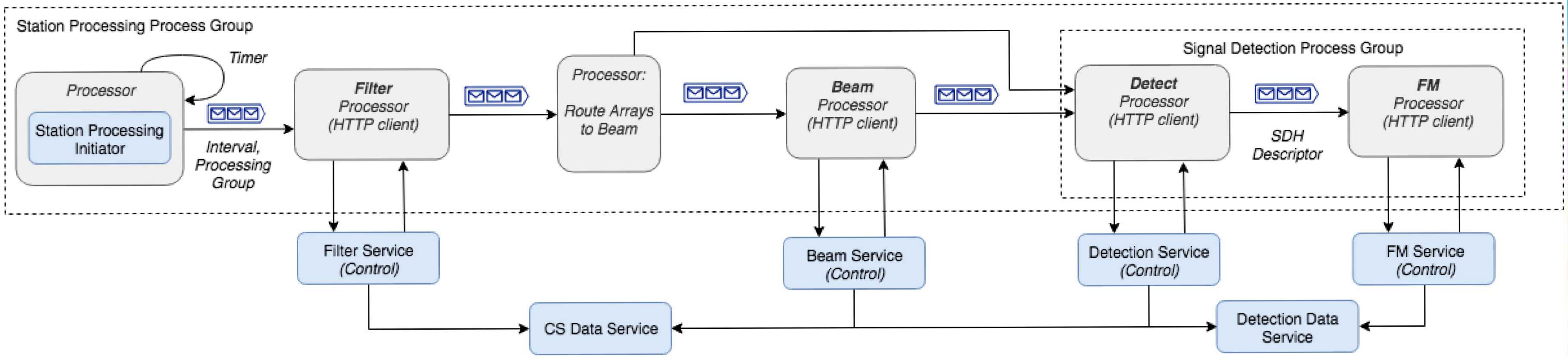
[1] Kruchten, Philippe (1995, November). Architectural Blueprints — The “4+1” View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50.

Backup

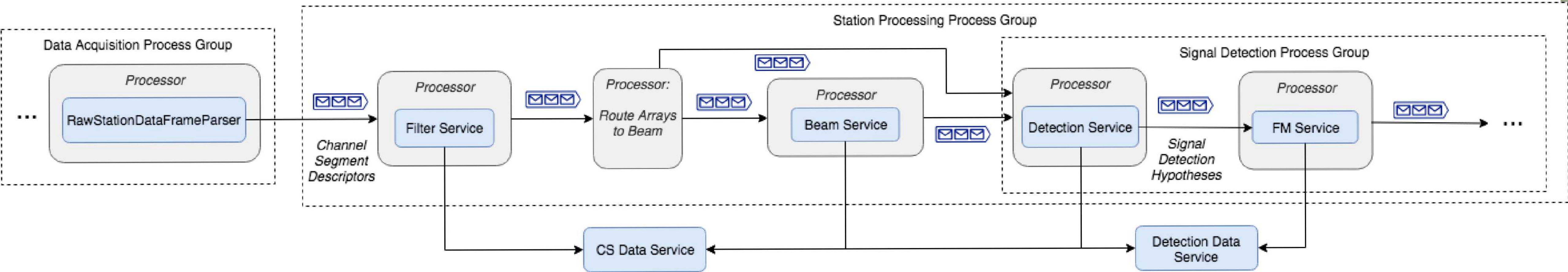
Application Layer – Station Processing

Basic Station Processing flow under development

Starting design: interval driven with external services


Station Processing Process Group
Timer
Processor
Station Processing Initiator
Interval, Processing Group
Filter Processor (HTTP client)
Processor: Route Arrays to Beam
Beam Processor (HTTP client)
Signal Detection Process Group
Detect Processor (HTTP client)
SDH Descriptor
FM Processor (HTTP client)
Filter Service (Control)
Beam Service (Control)
Detection Service (Control)
FM Service (Control)
CS Data Service
Detection Data Service

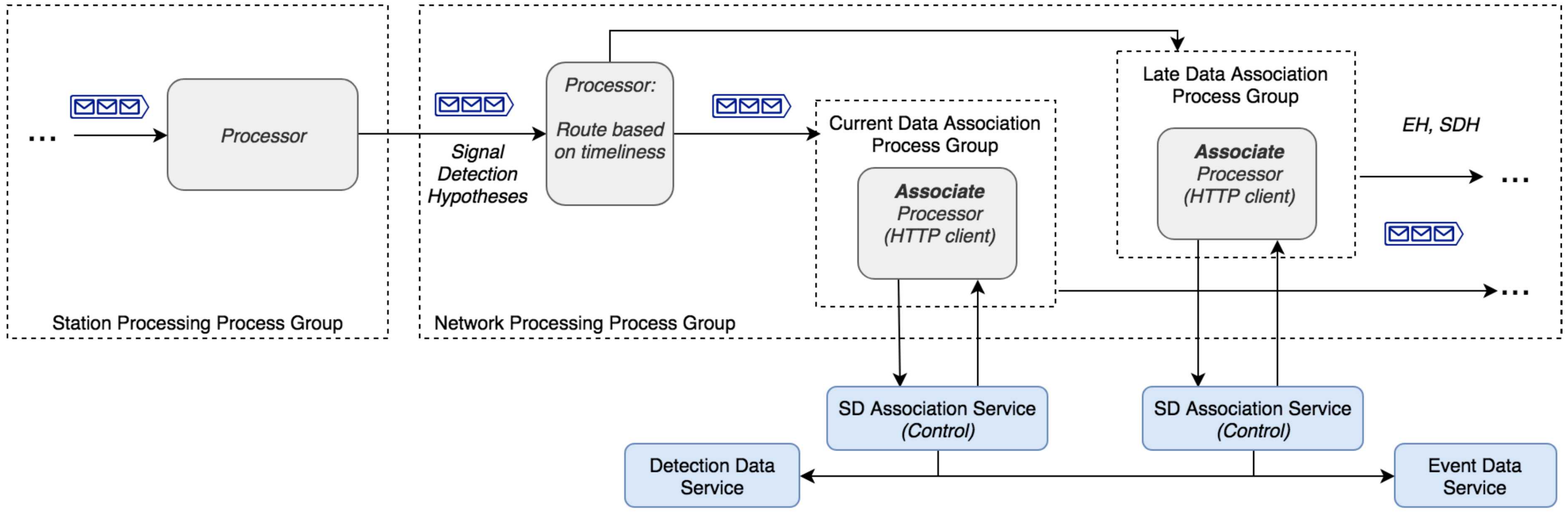
Investigating data-driven alternative with in-line processing services


...
Data Acquisition Process Group
Processor
RawStationDataFrameParser
Channel Segment Descriptors
Station Processing Process Group
Processor
Filter Service
Processor: Route Arrays to Beam
Processor
Beam Service
Signal Detection Process Group
Processor
Detection Service
Signal Detection Hypotheses
Processor
FM Service
...
CS Data Service
Detection Data Service

Application Layer – Network Processing

Initial Signal Detection Association flow development starting PI 7

- Investigation of data-driven vs interval-driven associator


...
Processor
Station Processing Process Group
Signal Detection Hypotheses
Processor:
Route based on timeliness
Network Processing Process Group
Current Data Association Process Group
Associate Processor (HTTP client)
Late Data Association Process Group
Associate Processor (HTTP client)
EH, SDH
...
...
SD Association Service (Control)
SD Association Service (Control)
Detection Data Service
Event Data Service