NALEJ® Technologies.

MOSAIC Enterprise Open-Source Software Platform.

Infrastructure 3.0:
Hyperscale Hybrid Cloud.
Intelligent & Programmable Edge Software, Services, Systems and Networks.

DEVELOPER.2024 TREATISE

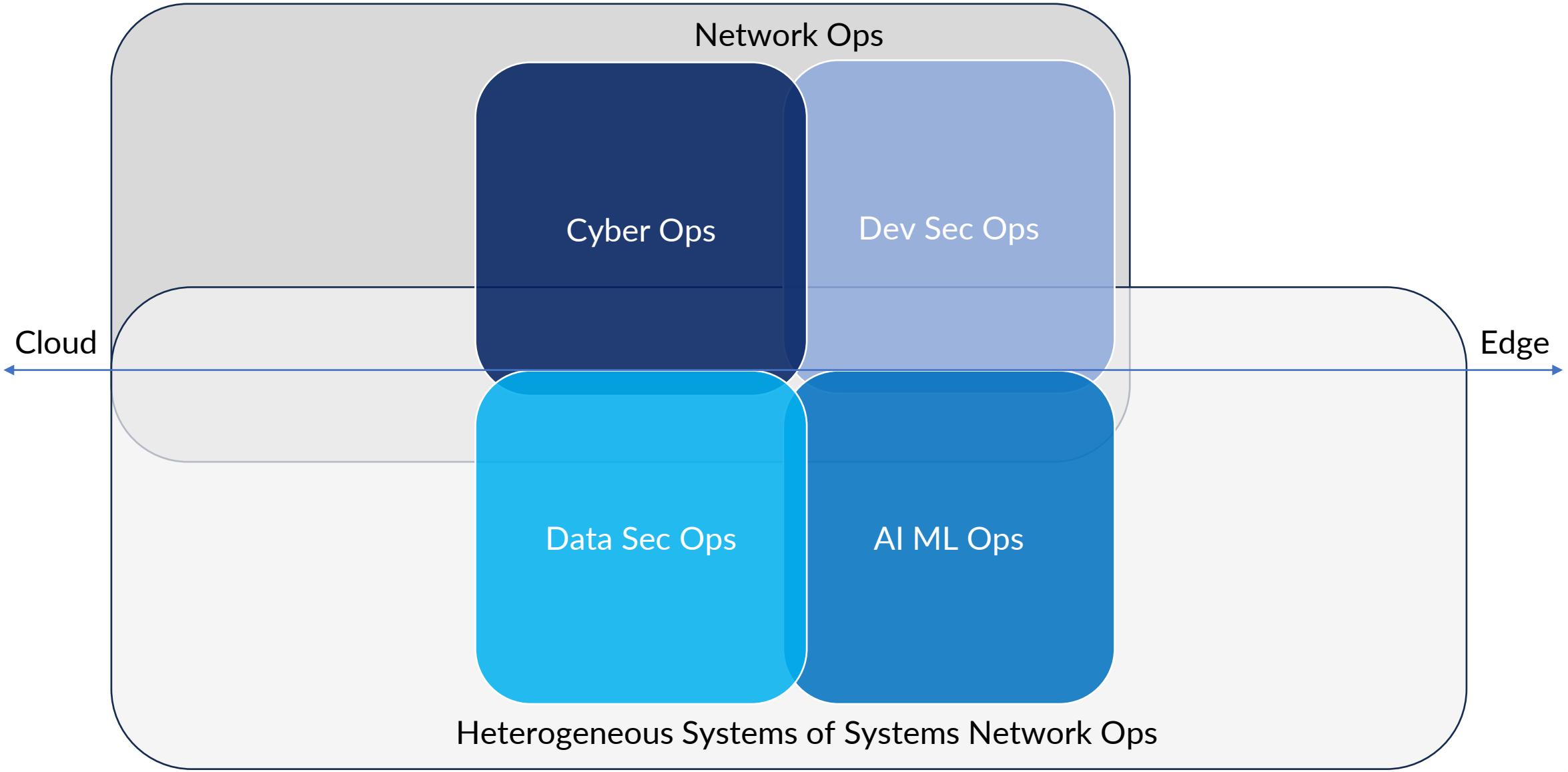
1. Fiercely Devoted to Enterprise Open-Source
2. Data-First Mentality
3. Infrastructure as Code 2.0+ (and defining together what comes next)
4. AI Optimization
5. Fully Automated & Multi-Modal Capabilities
6. Prototype to Innovation:
   (R&D Operations, Rapid-Prototyping, Hackathons & Exercises)

PROBLEM.CLOUD.PUBLIC

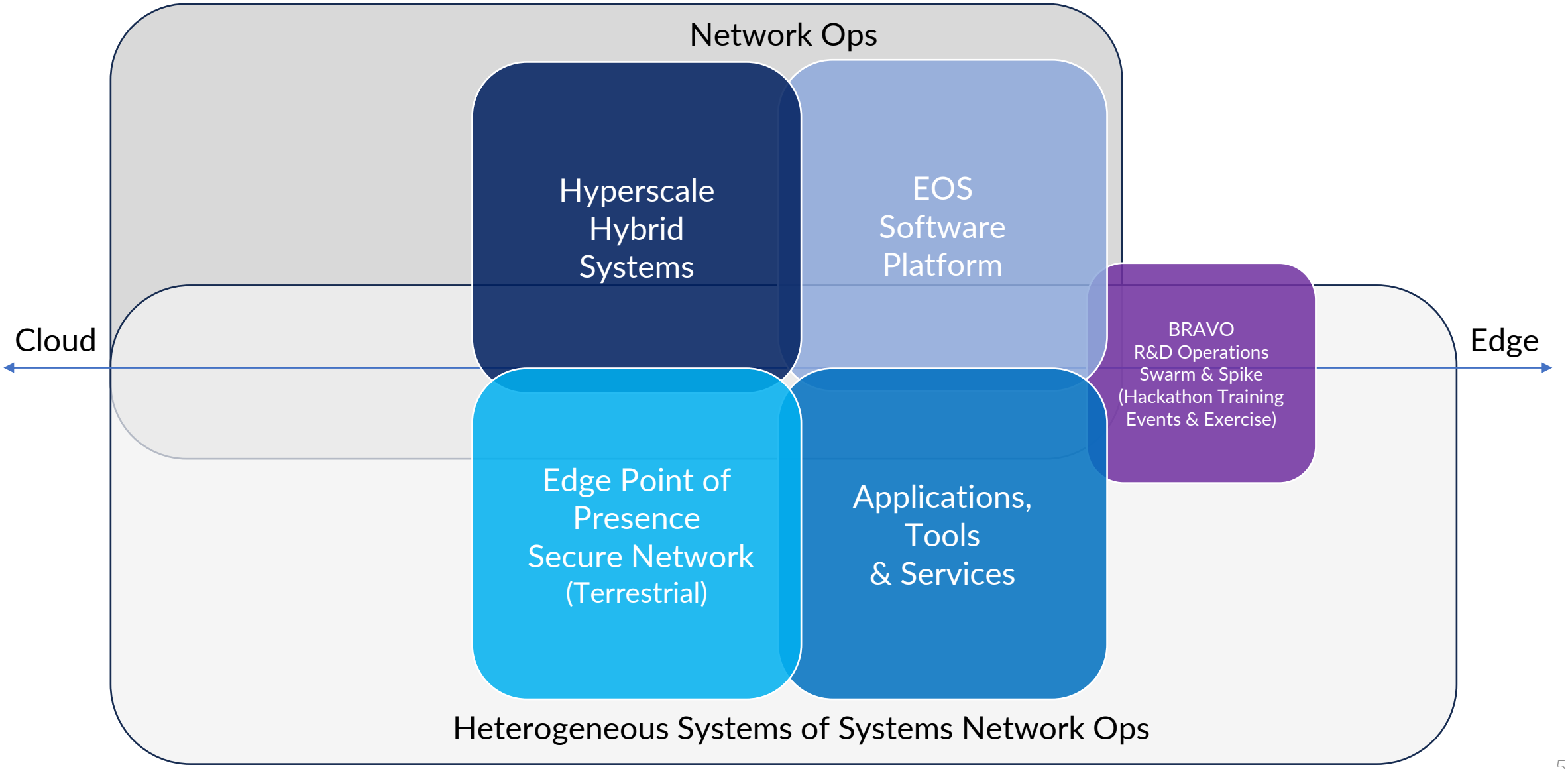
Distributed Cloud and the Adoption of Hyperscale Hybrid Cloud Capabilities are due to the following Public Cloud issues;

High Costs,
Security Vulnerabilities,
Performance,
Lack of Resilience,
Data Workload Shifts to the Distributed Cloud-Edge.

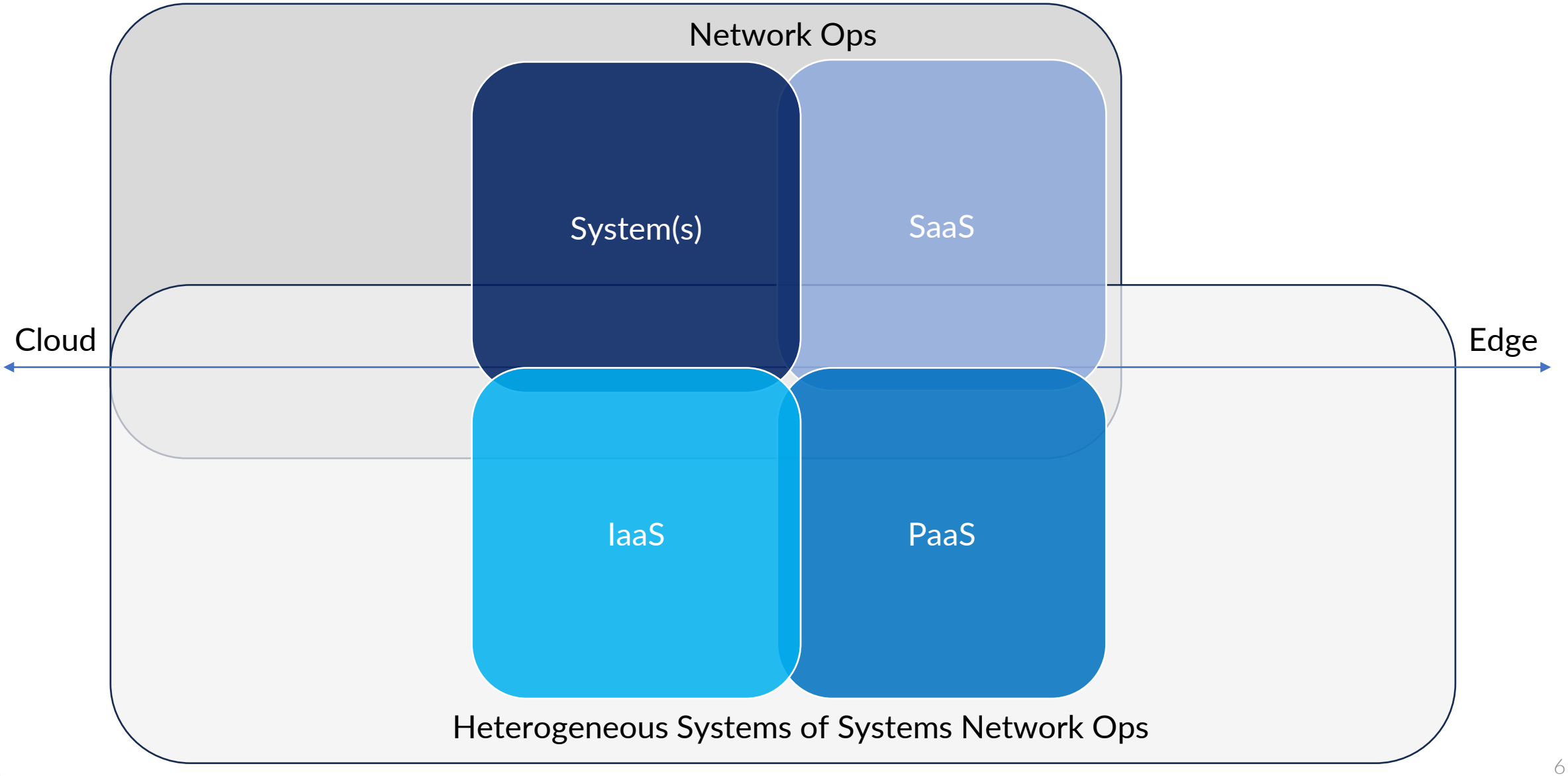
MOSAIC.PLATFORM



SOLUTIONS



MODALITIES

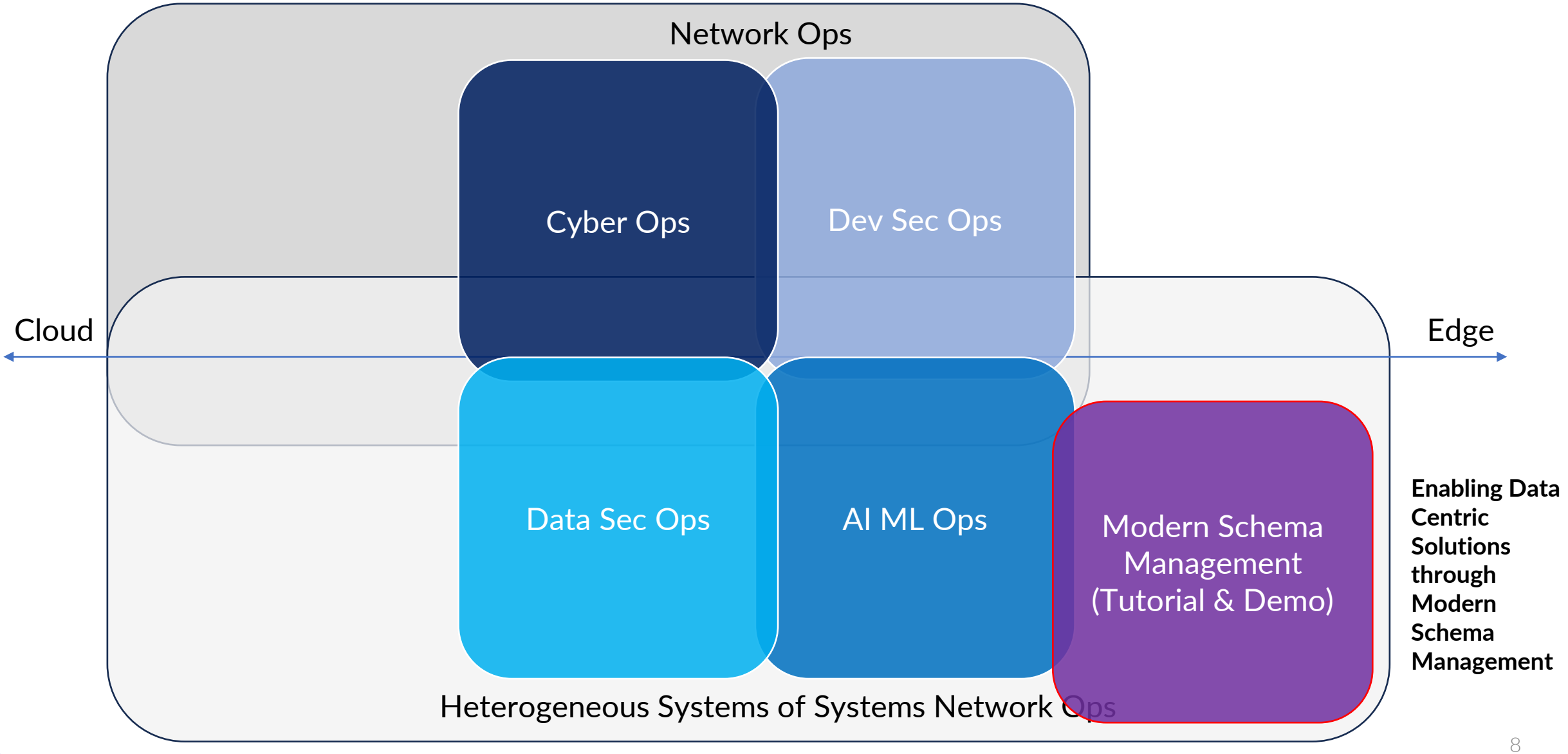


NALEJ®

Developers.

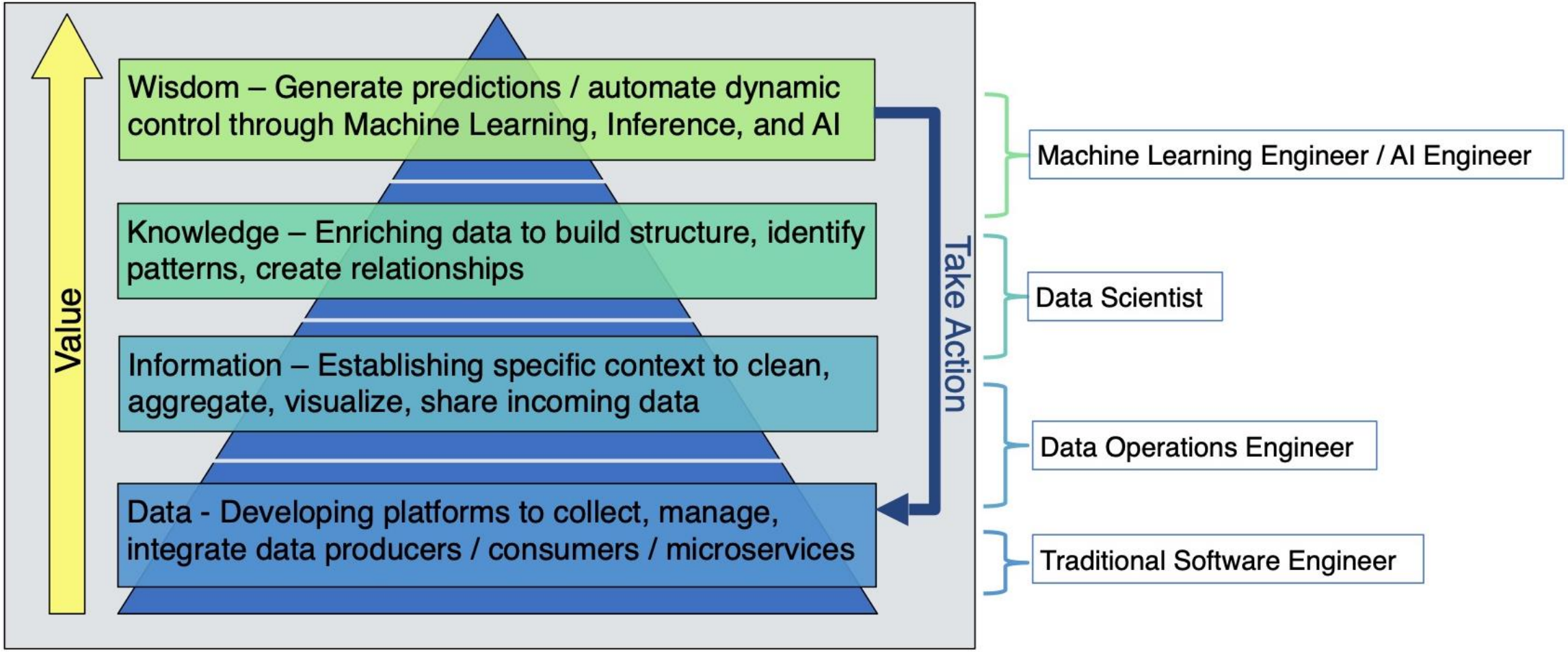
NALEJ Presenter:
Aaron Taylor, Senior Lead Architect

DEVELOPER



Enabling Data Centric Solutions through Modern Schema Management

DEVELOPER
Hierarchical Data Solution Needs



AVRO Schema.
Why Do We Need a Schema w/Kafka?

- Without a schema, what happens if:
  - A producer sends bad data?
  - A field gets renamed?
  - The data format changes?
- The Consumers Break!

- So, data needs to be:
  - Self describable
  - Able to evolve without breaking downstream consumers
- Solution:
  - Schemas and Schema Registry are required!!!

Avro VS Traditional Data Schemas: What is Avro?

Technological Advances in Data Schemas

Avro	• Data is fully typed • Data is compressed automatically (less CPU) • JSON defined schema comes along with data • Documentation embedded in schema • Data can be read across any language (binary) • Schema can safely evolve over time	• Avro support for some languages may be lacking • Can't print data without Avro tools (it's compressed & serialized)
JSON	• Data can take any form, e.g., arrays, nested elements • Widely accepted format on web • Can be read by most languages • Easily shared over a network	• Data has no schema enforcing • Repeated keys can cause massive JSON Objects
Relational Tables	• Data is fully typed • Data fits in a table	• Data has to be flat • Data is stored in database; data definition will be different for each database
CSV	• Easy to parse • Easy to read • Easy to make sense of	• Data types of elements must be inferred • Parsing becomes tricky when data contains comma • Column names may or may not be there

Avro Data Schema Evolution

- Based on how applications leverage data, various configurations of *Schema Evolution* can be managed with a Schema Registry
- Types of Schema Evolution:
  - none - new schema can be any valid Avro schema
  - backward - new schema can read data produced by latest registered schema
  - backward_transitive - new schema can read data produced by all previously registered schemas
  - forward - latest registered schema can read data produced by new schema
  - forward_transitive - all previously registered schemas can read data produced by new schema
  - full - new schema is backward/forward compatible with latest registered schema
  - full_transitive - new schema is backward/forward compatible with all previously registered schemas

Avro in Java
Constructing Avro Records: GenericRecord

- A **GenericRecord** is used to create an Avro object from a Schema
  - References schema as a file or as a string
  - Simplest way to create Avro objects in Java
  - Not recommended for production due to potential to cause runtime errors

Avro in Java
Constructing Avro Records: SpecificRecord

- A **SpecificRecord** is an Avro object created by using *code generation* from a Schema
  - Code generation plugins exist for multiple build tools
    - e.g., Gradle, Maven, SBT
  - Official "Avro Code Generation Tool" is in Maven
  - Example:

```
{   "namespace" : "io.astral.avro",
   "name" : "DataFeature",
   "type" : "record",
   "fields" : [ {"name": "id", "type" : ["null","int"], "default": null, "docs" : "The numeric ID of the data feature"},
      {"name": "name", "type" : ["null","string"], "default": null, "docs" : "The name of the data feature"},
      {"name": "description", "type" : ["null","string"], "default": null, "docs" : "The description of the data feature"},
      {"name": "value", "type": ["null","long"], "default": null, "docs" : "The value of the data feature"},
      {"name": "unit", "type": ["null","string"], "default": null, "docs" : "The unit of the data feature"},
      {"name": "host_url","type": ["null","string"],"default": null,"docs": "Originating Host URL"},
      {"name": "time", "type" : ["null","string"], "default": null, "docs" : "String formatted timestamp of data feature"}],
   "docs": "Value and unit of a data feature"}
```

Overview:

- Apache Avro Data Schemas
- **Confluent Schema Registry and REST Proxy**
- Data Schema Analogies and Learning Resources

Confluent Schema Registry

- Store/retrieve schemas for Producers / Consumers
- Enforce Backward / Forward / Full compatibility on topics
- Decrease size of payload of data sent to Kafka
- Operations through REST API:
  - Add schemas
  - Retrieve a schema
  - Update a schema
  - Delete a schema

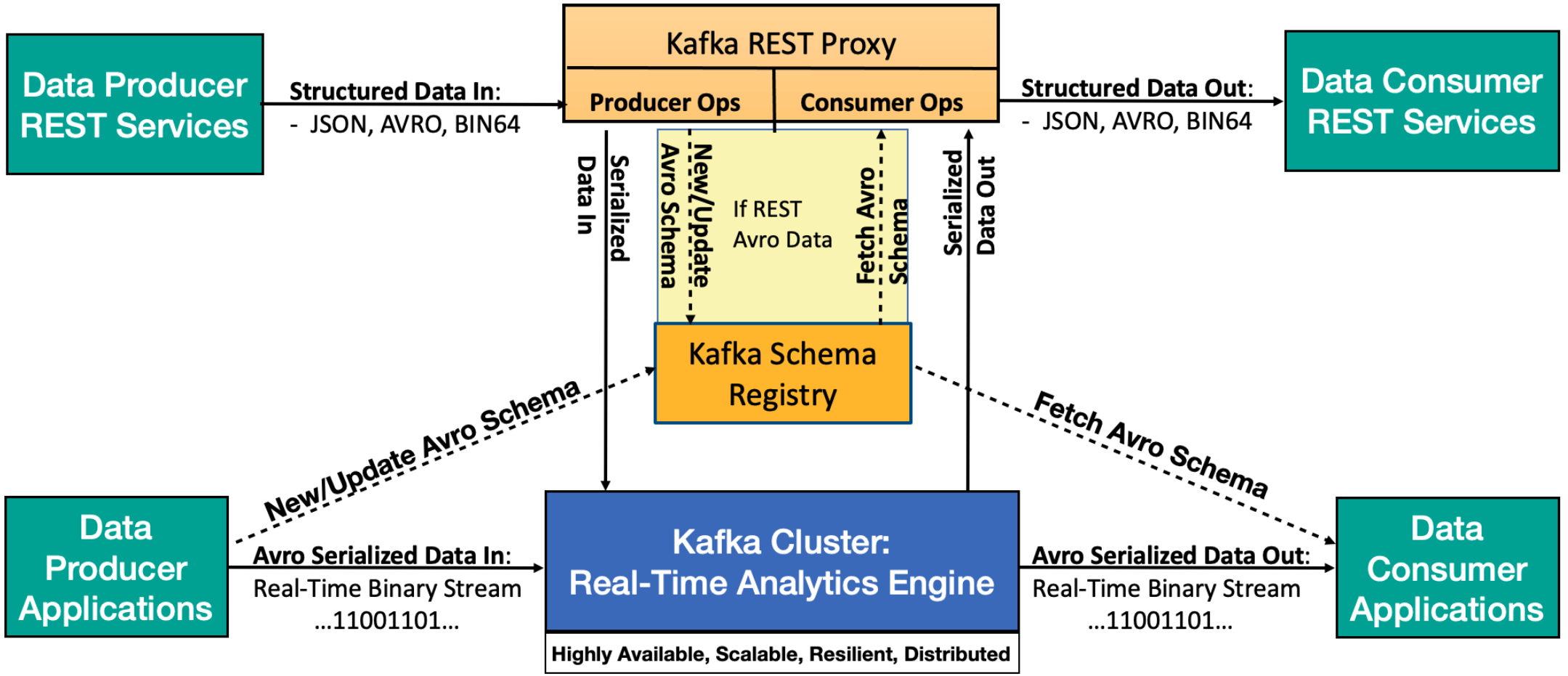
- Schemas can be applied to key and/or values

Confluent Rest Proxy

- Integrated with schema registry:
  - Enables Services to easily read/write to Avro serialized topics

- Note:
  - Performance hit to using HTTP instead of Kafka's native protocol
    - Throughput decrease 3-4x

Kafka Ecosystem Architecture: Schema Registry & REST Proxy



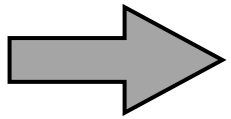
Overview:

- Apache Avro Data Schemas
- Confluent Schema Registry and REST Proxy
- **Data Schema Analogies and Learning Resources**

Analogies to Electrical Engineering:

- Data Schemas provide a Generalized Signal Context that:
  - Define discrete data sources of any context, not just power sources
  - Translates physical/meta needs at the edge into a software contex
  - Enforces signal integrity across Software Defined Infrastructure
  - Fundamental to developing closed-loop AI systems and solutions
  - Example: Ohms law modeled as Avro schema

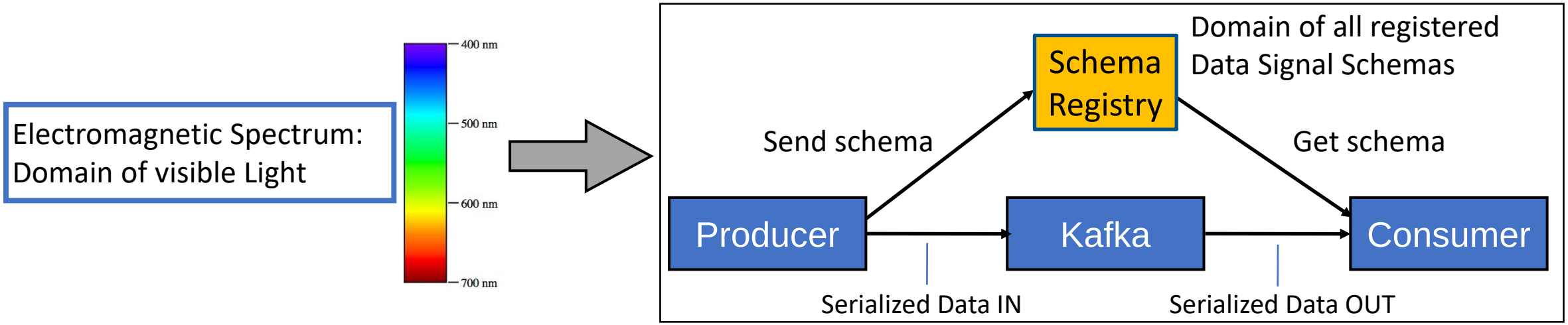
V=I*Z



```
{
   “namespace” : “example.avro”,
   “type” : “record”,
   “name” : “OhmsLaw”,
   “fields” : [
      {“name” :  “SourceId”, “type” : “string”},
      {“name” :  “voltage”, “type” : “double”},
      {“name” :  “current”, “type” : “double”},
      {“name” :  “impedance”, “type” : “double”}]
}
```

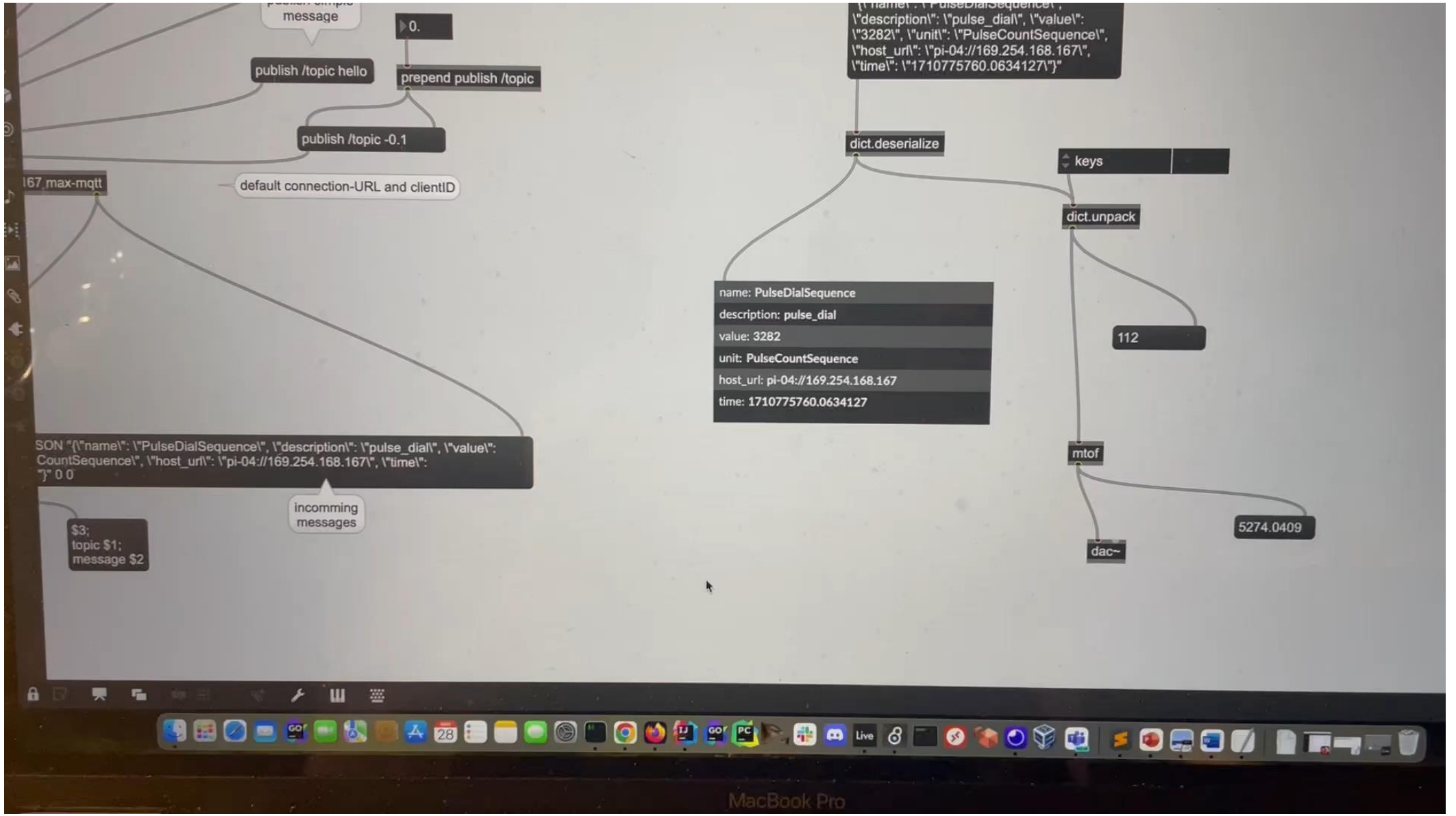
Analogies to Electrical Engineering:

- Schema Registry provides domain of all defined signals:
  - A discrete data signal spectrum, rather than electromagnetic spectrum
  - Repository to all versions of a specific schema
  - Provides means to serialize/deserialize data in distributed systems
  - Ensures data propagation continues, even when schema “breaks
  - Example:



Demo Video 1: Extreme Edge Real-Time Data

Demo Video 2: Centralized Data Edge-to-Cloud



Learning Resources / References:

- Maarek, S. (2017). *Introduction to Schemas in Apache Kafka with the Confluent Schema Registry*. [online] Medium. Available at: https://medium.com/@stephane.maarek/introduction-to-schemas-in-apache-kafka-with-the-confluent-schema-registry-3bf55e401321 [Accessed 15 Feb. 2019].
- Docs.confluent.io. (2019). *Schema Registry — Confluent Platform*. [online] Available at: https://docs.confluent.io/current/schema-registry/docs/index.html [Accessed 16 Feb. 2019].
- GitHub. (2019). *confluentinc/schema-registry*. [online] Available at: https://github.com/confluentinc/schema-registry [Accessed 16 Feb. 2019].
- Maarek, S. (2017). *How to use Apache Kafka to transform a batch pipeline into a real-time one*. [online] Medium. Available at: https://medium.com/@stephane.maarek/how-to-use-apache-kafka-to-transform-a-batch-pipeline-into-a-real-time-one-831b48a6ad85 [Accessed 16 Feb. 2019].
- Maarek, S. (2019). *Apache Kafka Series - Confluent Schema Registry and REST Proxy*. [online] Available at: https://www.udemy.com/confluent-schema-registry/ [Accessed 15 Feb. 2019]
- Medium. (2019). *Using Kafka Streams API for predictive budgeting – Pinterest Engineering – Medium*. [online] Available at: https://medium.com/@Pinterest_Engineering/using-kafka-streams-api-for-predictive-budgeting-9f58d206c996 [Accessed 16 Feb. 2019].