Understanding Datasets

Seeing the Unseen through Graph Automations

September 2022

MITRE | SOLVING PROBLEMS FOR A SAFER WORLD™

The Need

The Problem To Solve

- Simplified mechanism for building and unifying disparate data sources into a normalize high performance Unified Data Fabric, which can be evaluated and understood for Analytics, Machine Learning, and Simulations
- Mechanisms to visually explore and understand both data entities and their relationships to one another
- Mechanisms to understand and traverse threads of augmented entities
- Mechanisms to project and understand through graph analytics the desired dataset

The Rules Simulation Engine is the Answer, One mechanism for all these things, and more.

Why Graphs?

Graphs are the Answer

- **High performance Entity<->Entity Relationship Management**
- **Any Entity can reference any other Entity. Graphs are highly Adaptable, and White Board Friendly (Schemaless)**
- **Graphs are the foundational element for Analytics, NLP, ML, and AI.**
- **Graphs are thousands of time faster than the best Relational Databases even on small datasets and the performance gap grows exponentially as dataset sizes increase**
- **Relationships are First Class Citizens**
- **Graphs are the cornerstone of our solution**

Find Reports and How Many are Managed Three Level Down

Recursive Cypher

```
match (boss)-[:Manages*0..3]->(mgr)
where boss.name="Dave Green"
and (mgr-[Manages]->())
```

Recursive SQL

Declarative

Describe what you want
Not of how to do it

Graphs are schemaless and adaptable to changing requirements.

Relationships are first class citizens

Comparing Graph and Relational Structures

Dataset of 1000 people, each with 50 friends. Find all the paths from one person to another in four or less hops.

	# of People	Query Time
RDBMS	1,000	2000ms
Neo4J	1,000	2ms
Neo4J	1,000,000	2ms

Dataset of 1,000,000 people each with 50 friends. Find friends of friends up to five levels deep.

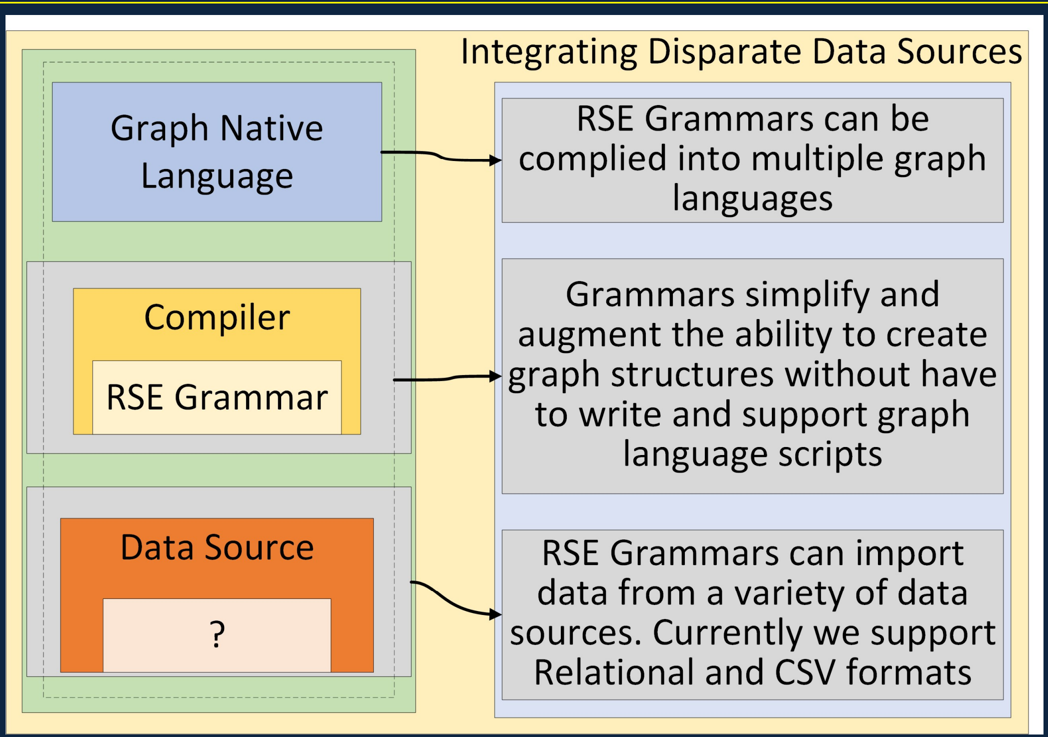
Depth	RDBMS (ms)	Neo4J (ms)	Records Returned
2	0.016	0.010	~2,500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	unfinished	2.132	~800,000

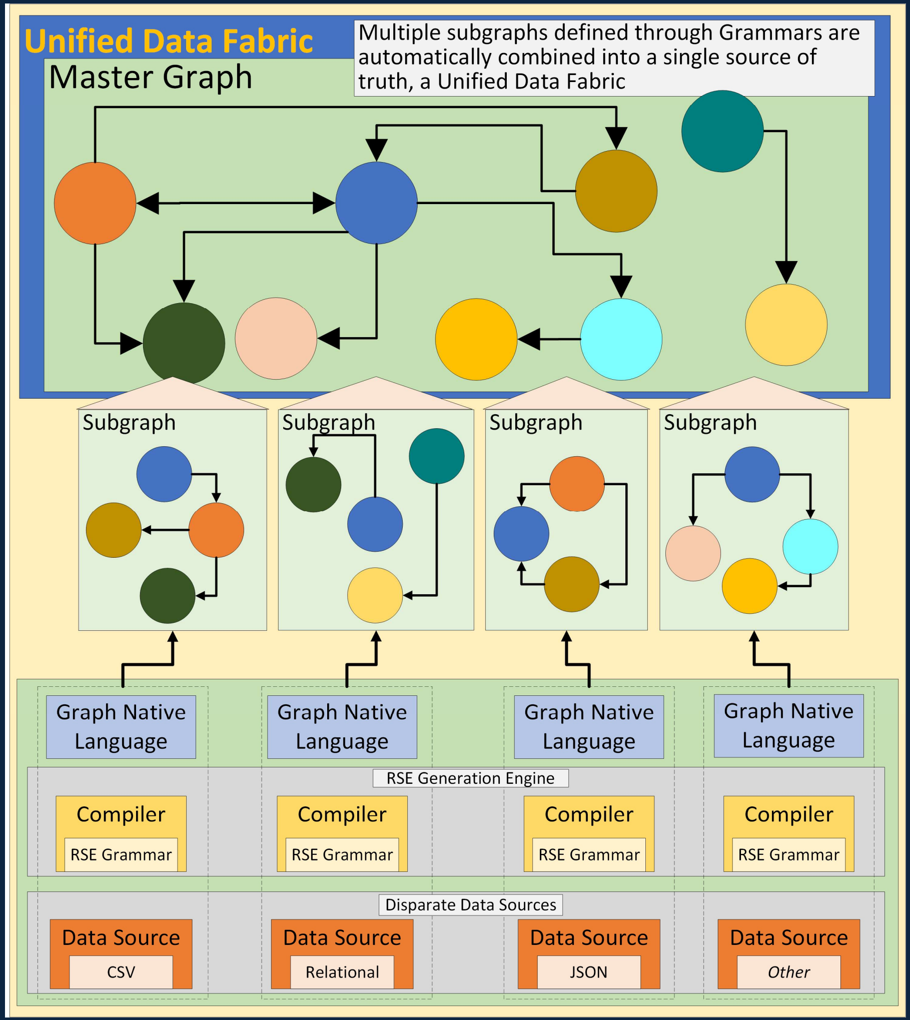
rlukas

Grammar Overview

Unified Data Fabric Creation Mechanism

Grammars are complied into native graph languages, currently Cypher a Neo4J open standard, but could also generate other Graph languages.
Grammars are able load data from an ever-growing set of data sources.

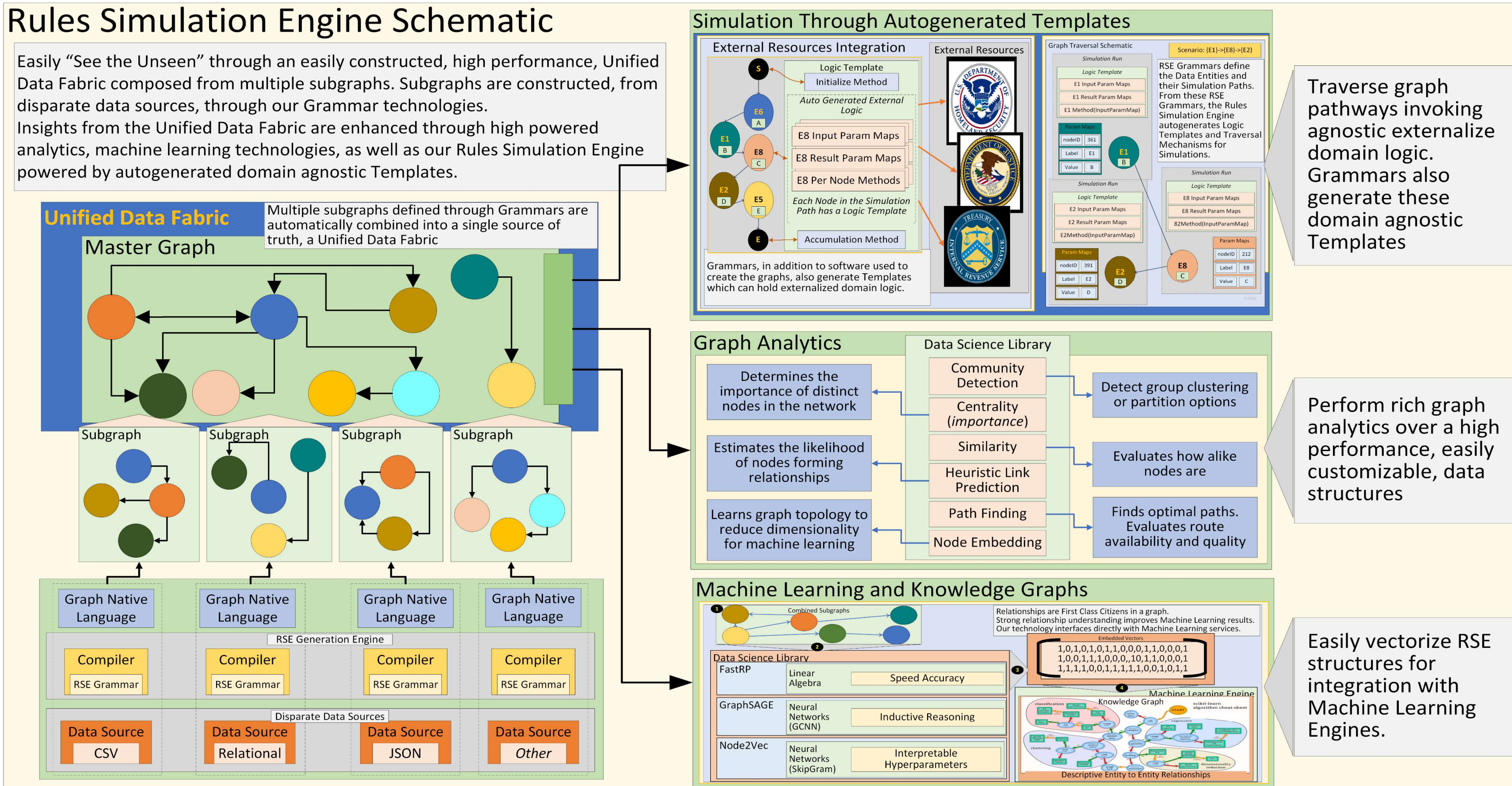




MITRE

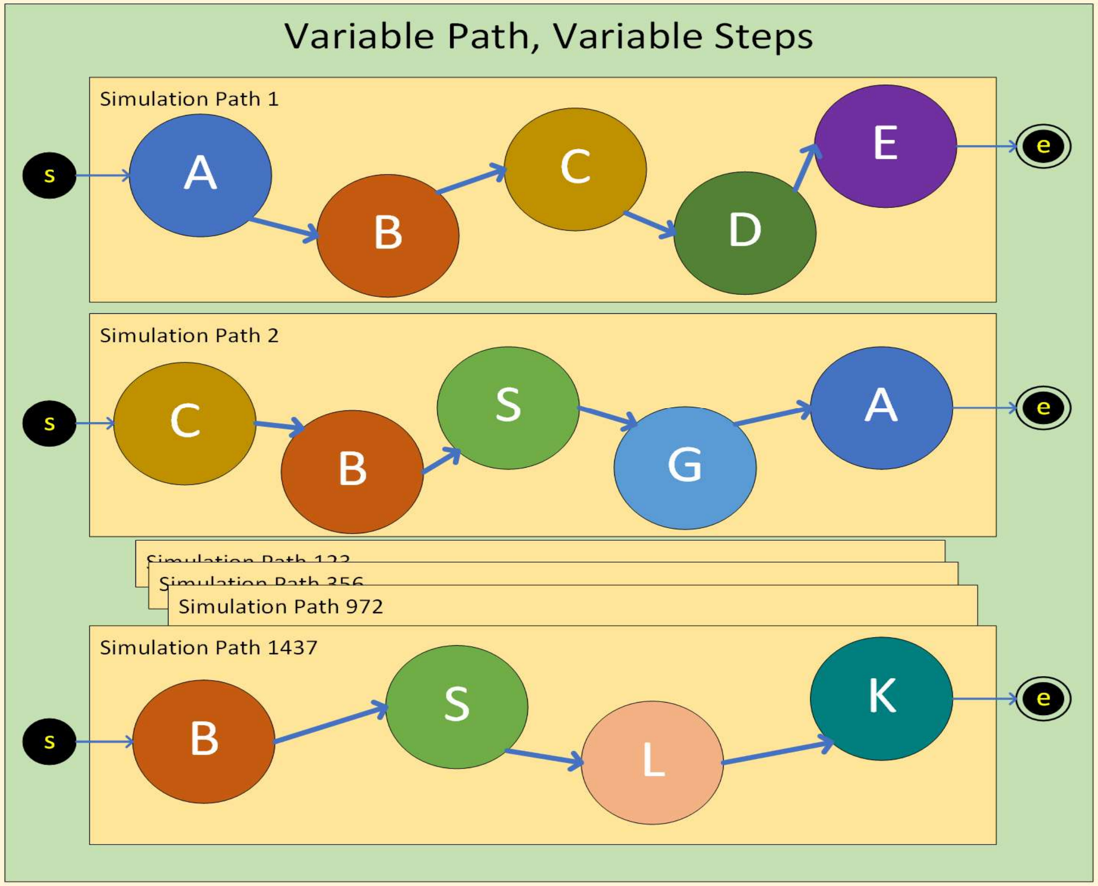
Approved for Public Release; Distribution Unlimited. Public Release Case Number 22-2914

Overview Schematic



Making Decisions Through Thread Pathways

Create & Accumulate Graphs



Most projects can identify *Data Entities* that might contribute to the eventual outcome, with new *Data Entities* constantly emerging.

Our Solution provides a simple mechanism to define scenarios. These scenarios select, and order, *Data Entities* which can be included into variable length *Simulation Paths*.

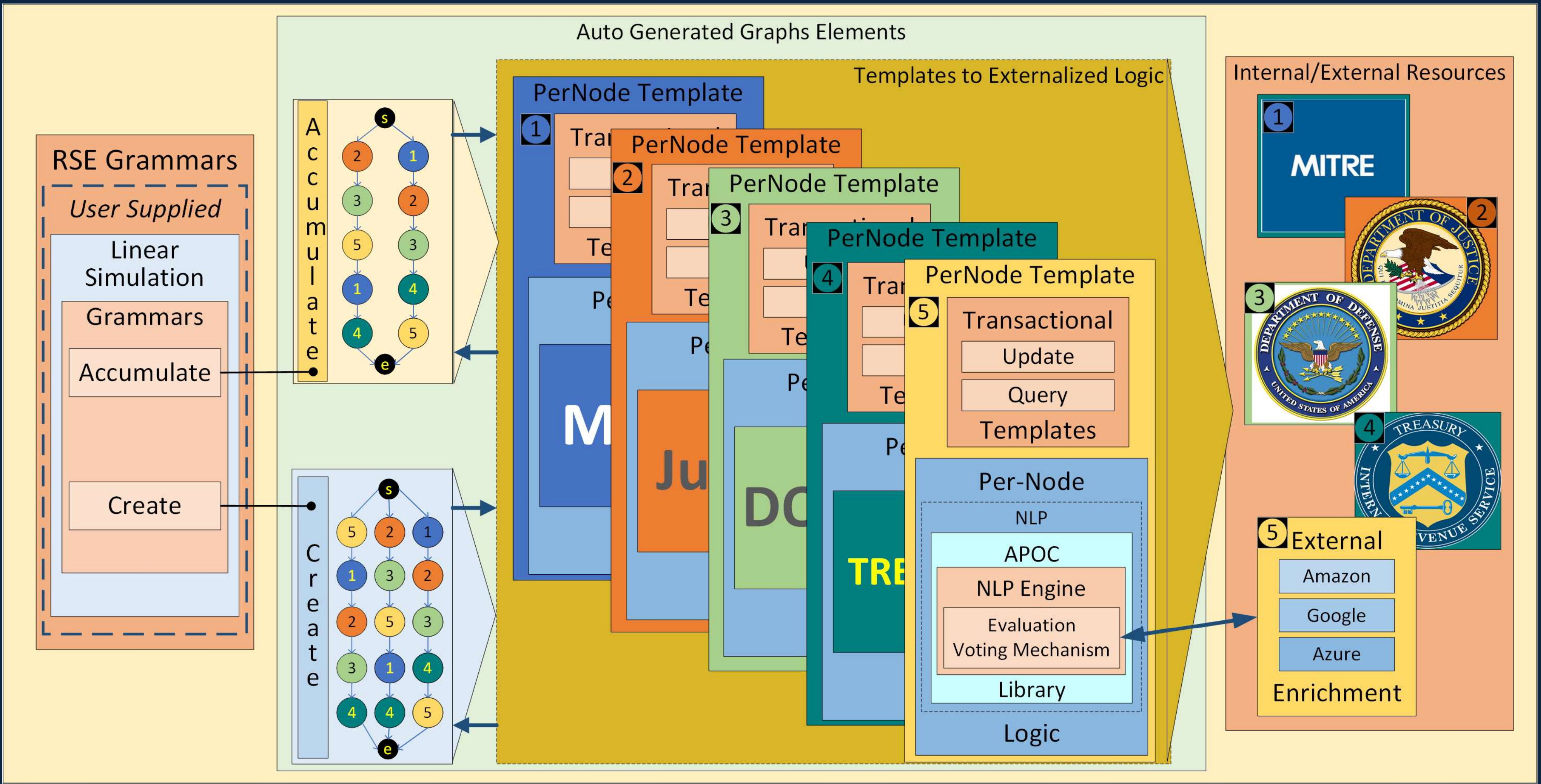
{A}->{B}->{C}->{D}->{E},
{C}->{B}->{S}->{G}->{A},
{B}->{S}->{L}->{K}, ...

Data Entity values may have multiple settings. For example, Income Levels may be described as a set of ranges.

Because of the dynamic nature of projects, *Simulation Paths* must be easily, and dynamically, defined from external sources.

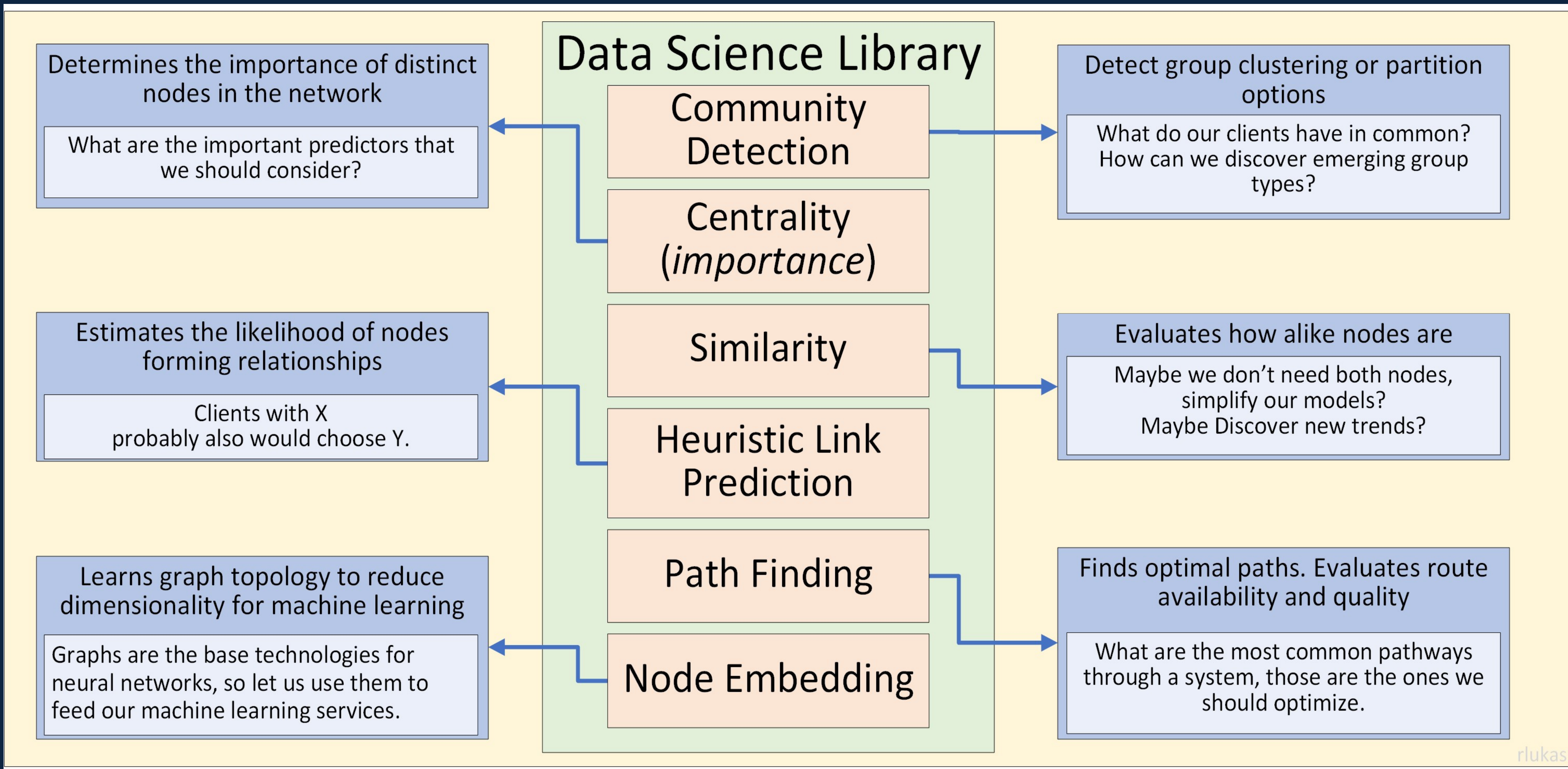
Thread Entities Receive Domain Logic

Domain Agnostic Simulations

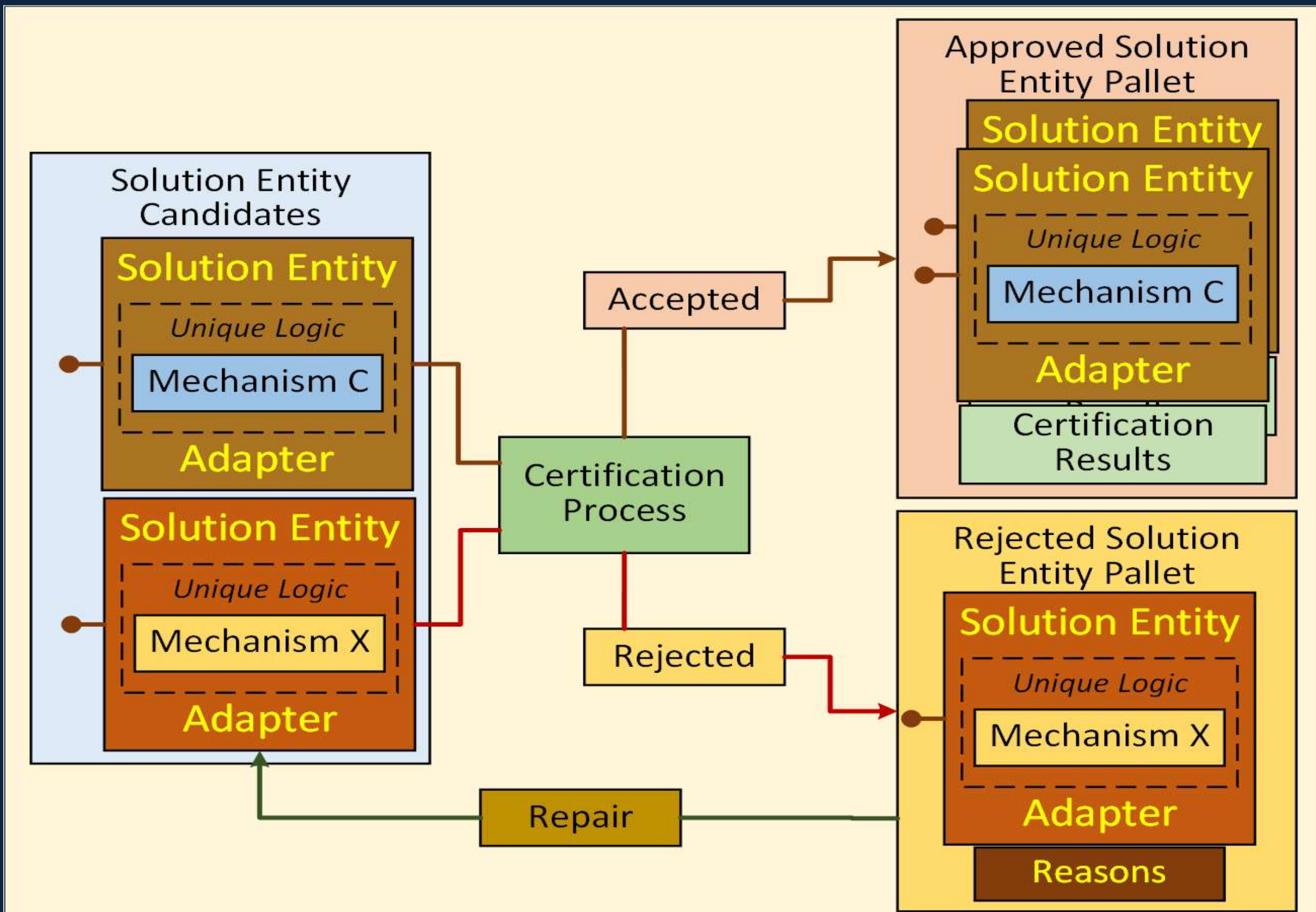


Data Science Library: A Global View

Graph Analytics



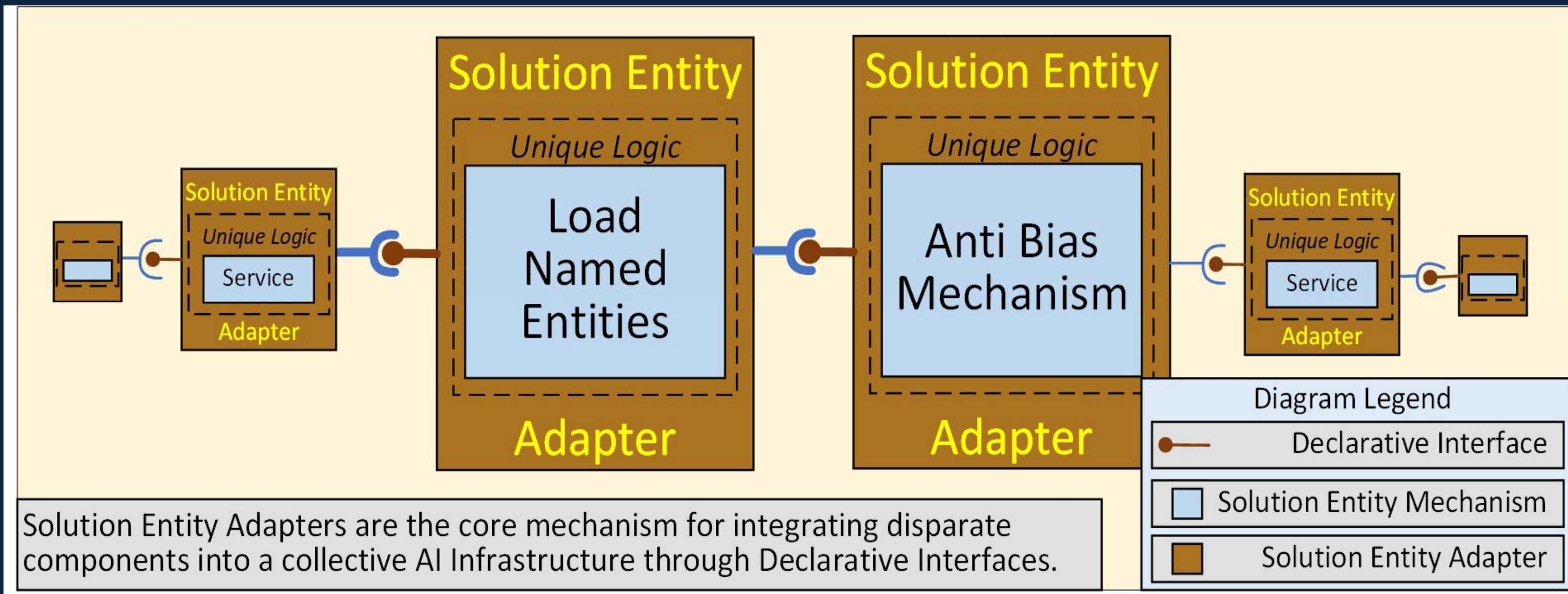
Solution Entity Pallet Mechanisms (CI/CD)



- Solution Entities are the cornerstone of our solution.
- Solution Entities undergo a Certification Process and normal CI/CD pipeline
- Solution Entities are combined provide perspectives into our high-performance datasets.

The AI Infrastructure Pipeline

Fundamental Building Blocks



Solution Entities are the next generation, domain agnostic Rule Engine Templates, residing in, and are selected from, a pallet library. Solution Entities expose and are connected through declarative interfaces

Interface Mechanisms

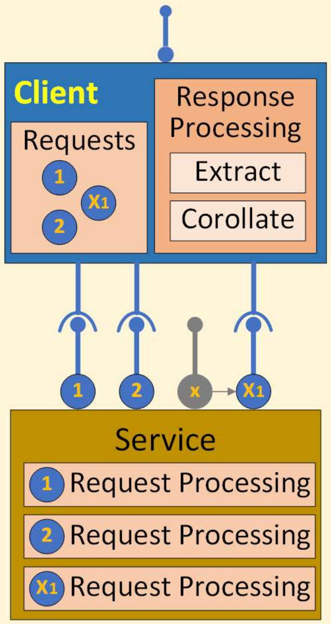
Current Interface Schema

Servers and Clients are Roles. A component often operates in both Roles

The need for an attribute set, generally requires acquisition through multiple interfaces and multiple requests from the Client.

The Client then has to parse the presented structures to obtain the desired results.

The Client is tightly bound to the structure of the Server. Changes to the Server interfaces (X -> X1) directly affects all X dependent Clients.



Each Service interface exposes one or more methods (functions with parameters and return types)

SOAP, REST, and Language Native, although different, are common interface types

Declarative Interfaces establish a common declarative request mechanism throughout the entire topology

Tightly bound components are contrary to Plug and Play topologies

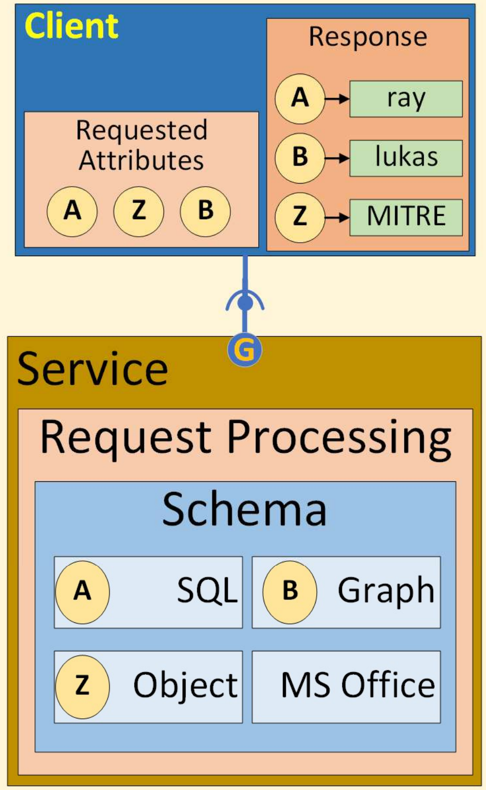
Interface versioning complicates the topology and the exposing Services

Transferring unrequested attributes harms performance on multiple levels

Declaritive Interface Schema

The Service Exposes a single interface to all Clients and Client versions.

Requests contain a the list of Requested Attributes. Services send only Requested Attributes simplifying Client processing.



The Schema contains pre-canned components which provide access to disparate data sources.

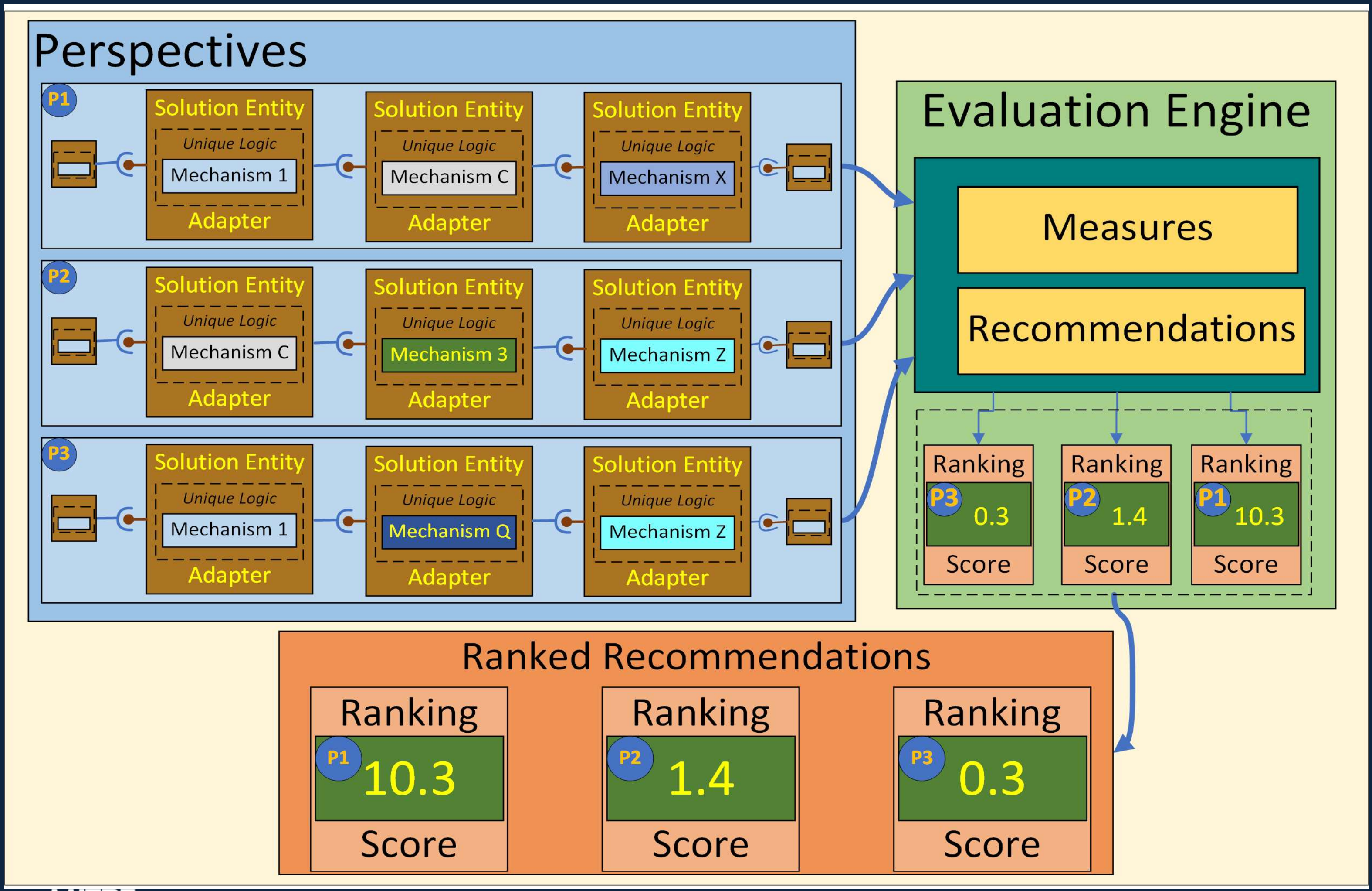
The Schema creates a standardized Key:Value Response object which is returned to the Requestor.

Declarative Interfaces simplifies both the micro and macro topologies

Declarative interfaces promote a Plug and Play Infrastructure

Gaining and Evaluating Different Perspectives

Fundamental Building Blocks

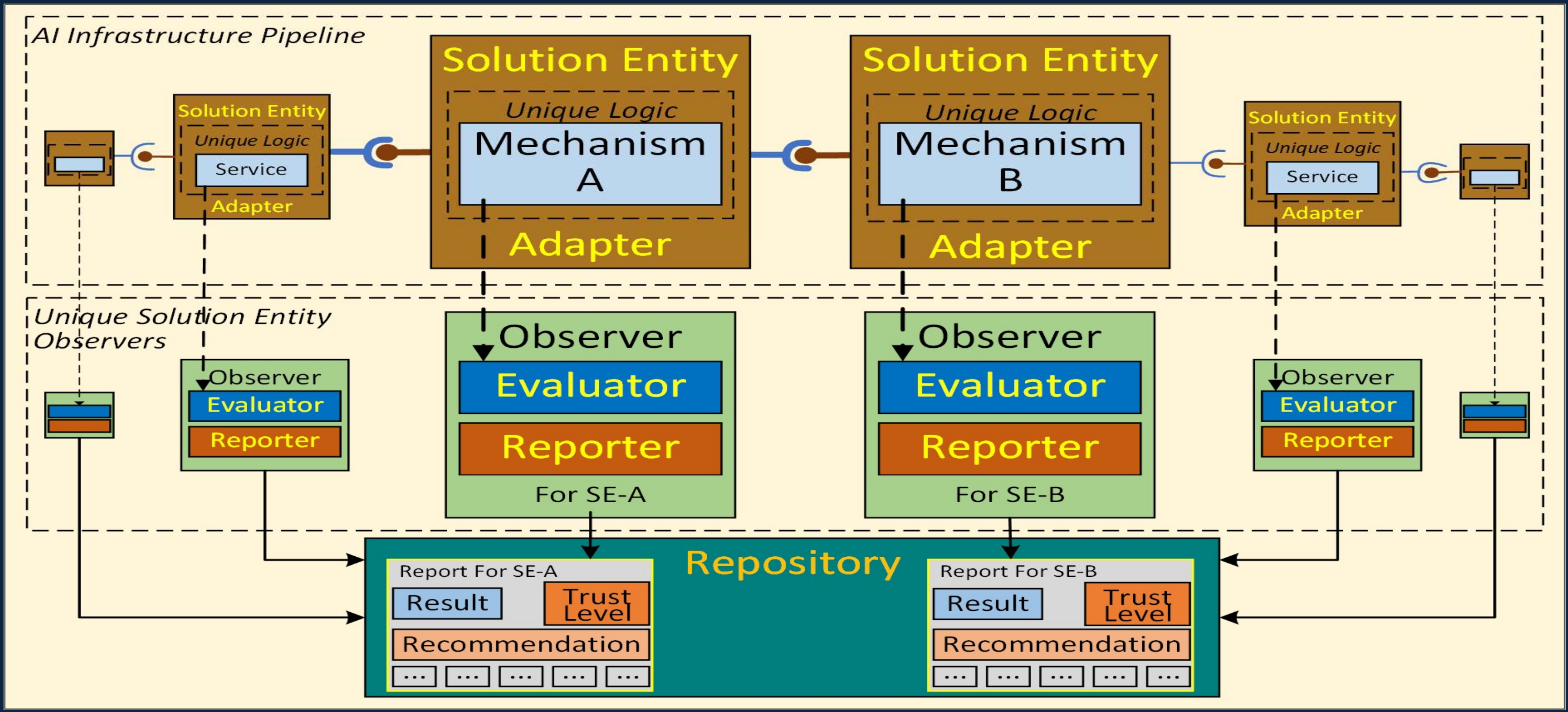


- Solution Entities are combined in any order for multiple perspectives
- Perspectives are then fed into the Evaluation Engine which produce Ranking Scores.

MITRE

Observers Provide a Global View

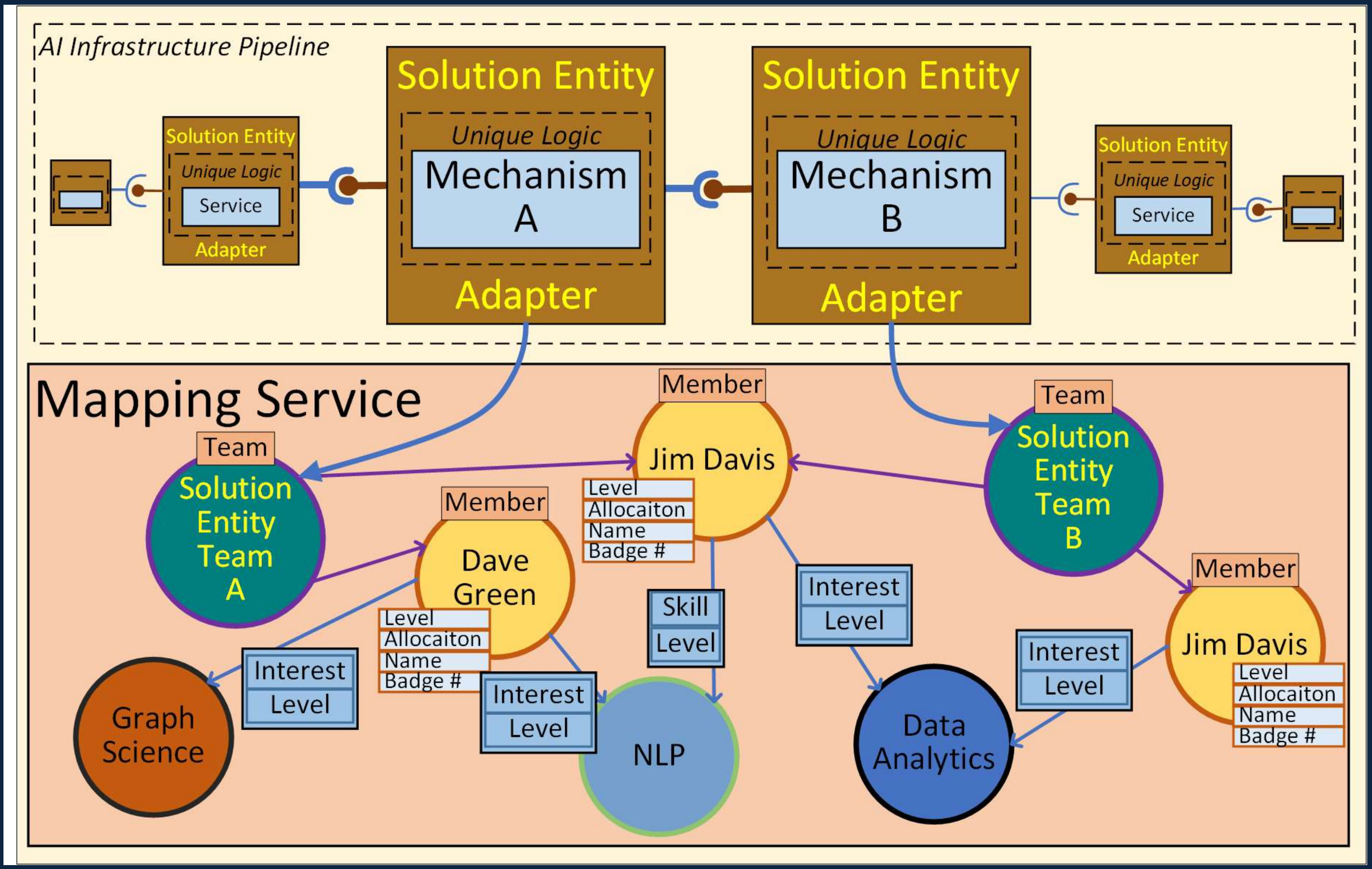
Fundamental Building Blocks



Each Solution Entity Adapter implements an Observer Pattern which reports Events are Relevant State into a common Mapping Service Repository

General Mapping Service

Fundamental Building Blocks



Solution Entity States are preserved directly through their Adapters or the Observers into a high-performance graph structure which are easily be explored visually or through graph analytics.

Wrap Up

The Problem To Solve

- **Simplified mechanism for building and unifying disparate data sources into a normalize high performance Unified Data Fabric, which can be evaluated and understood**

  MITRE Grammars/Compiler provide a simple mechanism to create new graph structures to address emerging explorations through multiple services

- **Mechanisms to visually explore and understand both data entities and their relationships to one another**

  Bloom: Perspectives, Share Discoveries within teams, Near Natural Language processing

  Browser: Cypher developers interface simplified through our Grammars

- **Mechanisms to understand and traverse threads of augmented entities**

  Templates and Solution Entities provide domain agnostic Rules Engine thread processing

- **Mechanisms to project and understand through graph analytics the desired dataset**

  Analytic Projections and Machine Learning integration through Data Science Library

The Rules Simulation Engine is the Answer, One mechanism for all these things, and more.

MITRE

MITRE Labs inspires breakthroughs in applied science and advanced technology to transform the future of U.S. scientific and economic leadership. Our goal: Deliver disruptive innovation to support our mission of solving problems for a safer world.

Please feel free to contact me with any questions

Thank You

Ray Lukas (rlukas@mitre.org)
Jim Lockett (jdlockett@mitre.org)

The Attached Appendix Supplies More Detailed Explanations of RSE Grammars and Subgraphs

Database Comparisons

Database	Purpose	Attributes
Relational		
Oracle, SQL Server	Table Based; Rows and Columns	Highly Refined; ACID Transactions; Schema Based
No SQL		
MongoDB ★, Cassandra, Spanner	Key-Value Storage; Document Storage	Eventual Consistency; Great for Web Interactions
Graph		
Anzo, Neo4J	Model Simplicity; High Speed Entity-Entity Traversals	White Board Friendly; ACID Transactions; Schemaless

★ Mongo DB contains ACID Transactions

Relational

Strict schema and data normalization separating data into tables. To preserve data consistency ACID transactions are supported. This imposes limitations on how relationships can be queried. Translations from OO to Rel are difficult and expensive.

Relational model and other NoSQL database models link the data by implicit connections

Relationships are reunified at query time

Graph

Whiteboard Friendly/Object Oriented: Native Graph databases have no pre-canned schema. Structures are directly mapped, any node can point to any other node. Unlimited query environment.

Relationships are first-class citizen in a graph database and can be labelled, directed, and assigned properties

★ Graphs and their Analytics Libraries are Very Scalable. Our graph technology can handle Trillions of nodes. Graphs with less than 10 million nodes and relationship are considered small.

But Which Is The Best? It Depends On What You Are Doing

Performance: Connections are made a creation time, not at query time

RSE General Terminology

Basic Graph Terminologies

Grammars: Grammars provide a simplified abstraction mechanism from the underlying graph technologies (in our case the Cypher Language) used to describe, and create graph, structures. Grammars are much like a high level programming language (C, C++, Java) which abstract the underlying CPU capabilities (assembly/machine code) mechanisms. *Grammars are compiled into the underlying Cypher codes.*

Pattern: Patterns are basic text strings (called ASCII Art) which describes a Node's Relationship to another Node. (Node)-[Relationship]->(Node). *See Appendix for details.*

Cypher: Cypher is the language of Graphs. Our Grammars are abstractions of Cypher and are compiled into Cypher which can be easily deployed to create new high performant graph structures. Cypher's compliment in the relational data world would be SQL. Cypher is basic *SQL for Graphs, but much better.*

Create and Merge Commands: Two basic and commonly used Cypher commands. Create creates the presented Pattern, even if is already exists. Merge uses the presented Pattern if it already exists, else it creates a new pattern in the database. *This is the underlying mechanism for auto connecting subgraphs and/or sharing threads in the Accumulation graphs.*

Rules Simulation Engine Terminologies

Rules Engine: A Rules Engine is a service that allows domain (system) logic to be defined and invoked externally from the engine. Since the logic of the system is externalized from the engine, this allows the engine to be easily adapted to, and used across, virtually any domain. *This is a cornerstone concept in our solution, and how we provide a domain agnostic solution.*

Simulation Path: A graph is a series of interconnected entities (nodes). Our Rules Simulation Engine is able to traverse these pathways and for each node invoke externalized domain logic living inside autogenerated templates. This is how our Rules Simulation Engine implements a high performance, domain agnostic, decision/simulation mechanism. *This technique is generally used with Create and Accumulate (threading type) graphs. Merge graphs are generally for graph analytics.*

RSE Grammar Node Types

RSE Grammar Node Types

Standard Element Types

Node (instanceVar:LabelList{AttributeNameValueList})

Relationship (node)-[instanceVar:RelationshipLabelList{AttributeNameValueList}]-(node)

Enhanced Element Types

Enhanced Visualization

Index Nodes: These are special autogenerated nodes which group like nodes together. For example, all the states in a country, all the counties in a state, all cities in a county all the data centers for each city. Each of these topology nodes are index nodes.

Graph Debugging

Data Source Nodes: Special nodes created for each data source that connect to all the nodes from that data source. Data Source nodes allow you to visualize the data sources for each node in a subgraph.

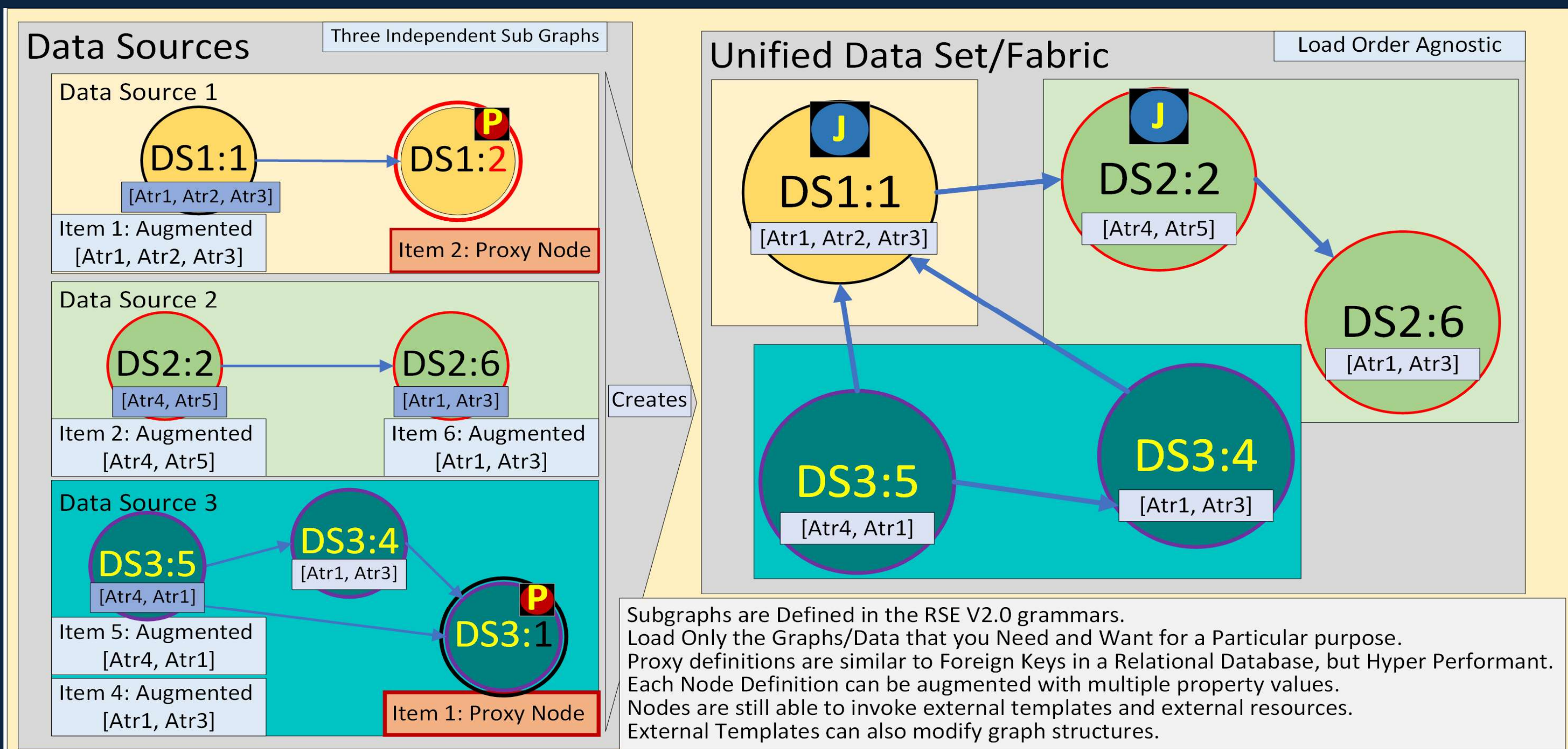
Error Graphs: ErrorType nodes are generated when errors are found in the underlying data source. Each ErrorType node points to nodes which describe each occurrence of such errors. Currently null data elements are supported, with additional error detection schemas under development

Unified Data Fabric

Combining Subgraphs

Subgraph Technology: Each Data Source can define one or more subgraphs. Subgraphs can be loaded in any order, or not a at all.

Proxy Nodes: Proxy Nodes (DS1:2 and DS3:1) are Place holder (Proxy) nodes which define connection points that will be mapped to by other subgraphs. Proxy Nodes hold only the state information defined in that Data Source.



Proxy Node Mappings

- **DS1:2 is defined in DS2:2. When DS2 is loaded DS1:2 is mapped and augmented with Attribute 4 and Attribute 5**
- **DS3:1 is defined in DS1:1. Loading the Data Source 1 subgraph maps to DS3:1 and load Attribute 1, Attribute 2, and Attribute 3**

Visual Exploration Mechanisms

Bloom

GPU Accelerated Visualization

High performance physics and rendering

- Easily discover patterns yielding more questions and explorations.
- Near natural language visual explorations of graph structures without knowing SQL or Cypher. Bloom even can suggests entities to include in your visualizations.
- Provides mechanisms to easily update graph entities without knowing Cypher
- Graph customizations (millions of colors, property based styles, icons, auto sizing, etc.)
- Easily augmented capabilities through embedded Cypher enrichments
- Provides defined perspectives which are tailored for a specific role or context
- Subgraphs, Perspectives, Scenes, and other enrichments can be shared

Browser

- Developer Tool requires Cypher, the Language of Graphs, Knowledge
- Limited display (colors, sizing of entities, no node icons, etc.) capabilities

Node Definition

Node Definition Grammar

Grammar Components

DS1	DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})
NodeName	DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})
Identifier	DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})
{Aug1, Aug2}	DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})

MultiFields

MultiFields contain multiple delimited data elements embedded into a single column. In this example these are separated by the semicolon character.

Multifield Definition Semicolon Delimiter

Node Definition Grammar

DataSource

Unique Node Identifier. Attribute Name will be "value" in the subgraph

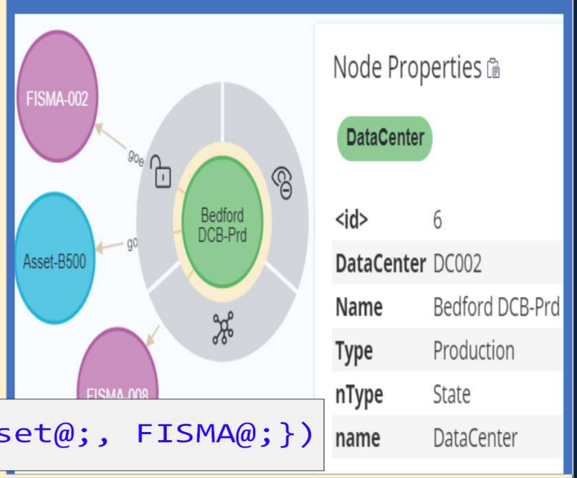
DS1: NodeName(Identifier {Aug1, Aug2})

The Name of the Node in the Subgraph

Additional Data Source Resident Augmentations for this Node

```
DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})
```

Data Center Node



Data Source
Defines where this node should obtain its attributes. Current supported data sources include CSV files and Relational Database Tables.

Node Name
Node names are no longer restricted to the column name.

Identifier
List of the unique identifiers (primary key concept) for this node.

Augmentations
Defines additional attributes from the data source which should be included into this Node.

Improved Autogenerated Graph Structures
Accumulate, Create, and Merge graph

Subgraph Definition

Node Interconnections

Subgraph Definition

Root Node -> Destination Nodes

Root Node

DS1: NodeName(Identifier {Aug1, Aug2})

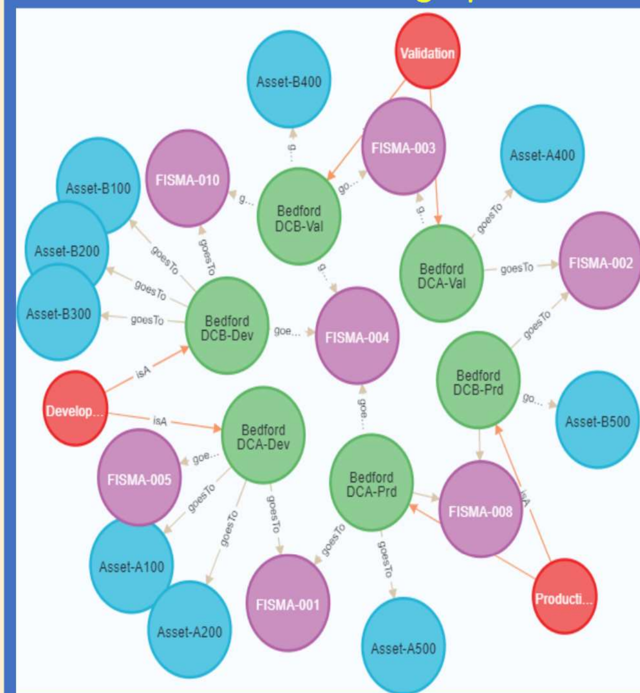
>

Destination Node List

[DS1: NodeName(Identifier {Aug1, Aug2}) ,

DS1: NodeName(Identifier {Aug1, Aug2})]

Data Center Subgraph



Graph Definition Example

```
DataCntDS:Type@I(Type) > [DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})]
```

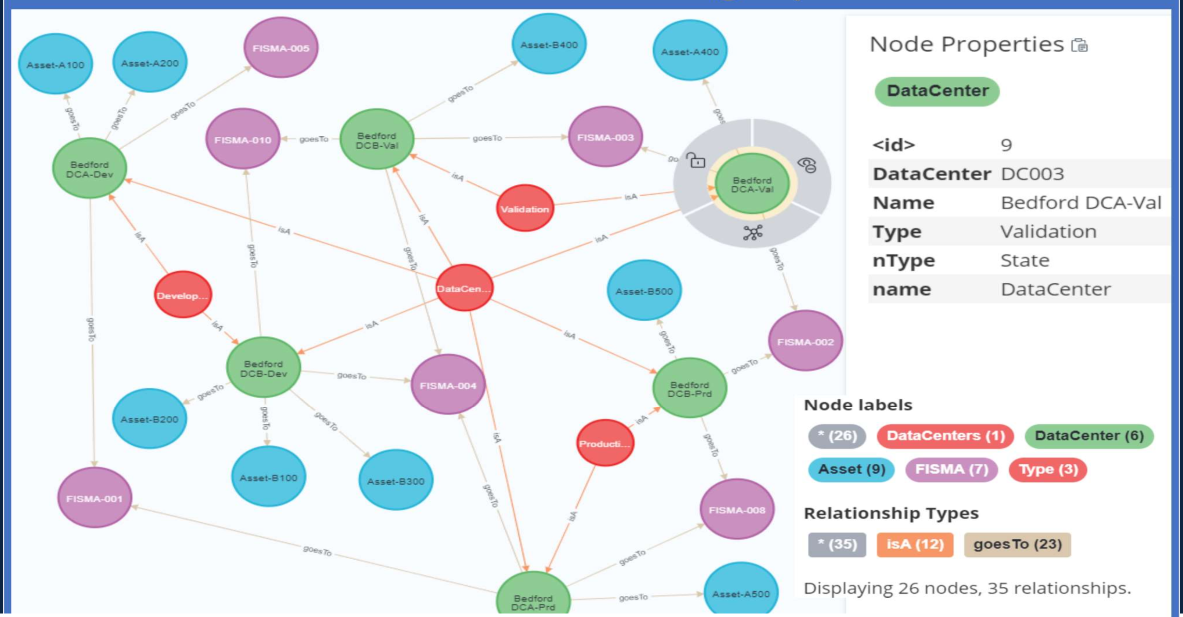
Subgraph Technology:

- Each Data Source defines can a subgraph. The DataCenter data source defines the following Data Center subgraph.
- Multiple subgraphs can, through our simplifies grammars, be defined each from separate or shared data sources and types.
- Subgraphs can be loaded in any order, or not at all. Loading a subgraph automatically connects like nodes together into a Master Graph
- Only load the data you need for your specific explorations

Data Center Subgraph

Constructing the Data Center Subgraph

Data Center Subgraph



We can see that the Data Center Data Source contains two multifield elements, Assets and supported FISMA elements

There are two types of Index nodes, both shown in Red. DataCenter Index node in the center points to all Data Center nodes. The Type Index node (Production, Validation, and Development) connect to their respective Data Center types.

Asset nodes are Proxy Nodes, containing on the Asset ID.

Data Center Grammars

Define Data Center Index Node

```
DataCentersDS:DataCenters@I()
```

Define Data Center Node with Asset and FISMA Multifields

```
DataCntDS:DataCenter(Name{DataCenter, Type, Asset@;, FISMA@;})
```

Define Data Center Type Index Node

```
DataCentersDS:Type@I(Type)
```

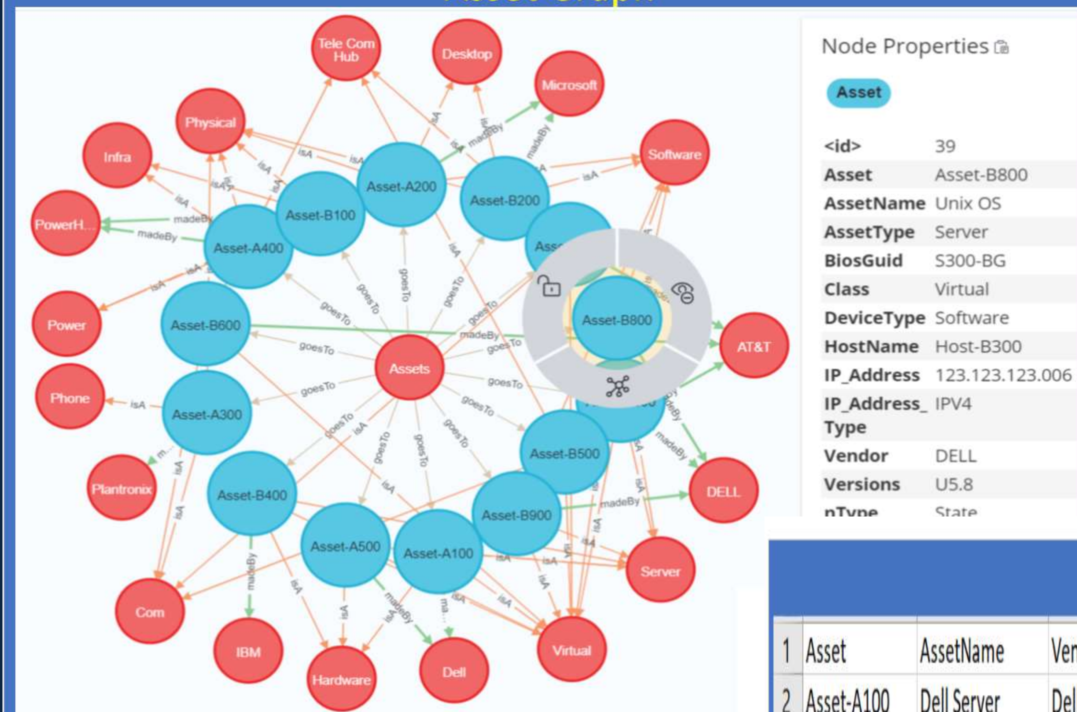
Data Center Data

1	DataCenter	Name	Type	Asset	FISMA
2	DC001	Bedford DCA-Prd	Production	Asset-A500	FISMA-001; FISMA-004; FISMA-008
3	DC002	Bedford DCB-Prd	Production	Asset-B500	FISMA-002;FISMA-008
4	DC003	Bedford DCA-Val	Validation	Asset-A400	FISMA-003;FISMA-003;FISMA-002
5	DC004	Bedford DCB-Val	Validation	Asset-B400	FISMA-004;FISMA-010;FISMA-003
6	DC005	Bedford DCA-Dev	Development	Asset-A100;Asset-A200	FISMA-005;FISMA-001
7	DC006	Bedford DCB-Dev	Development	Asset-B100;Asset-B200;Asset-B300	FISMA-004;FISMA-010

Asset Subgraph

Constructing the Asset Subgraph

Asset Graph



Asset Subgraph

Define Asset Index Node

```
AssetsDS:Assets@I()
```

Define Vendor, AssetType, ClassType, DeviceType Index Nodes

```
AssetsDS:Vendor@I(Vendor)
AssetsDS:AssetType@I(AssetType)
AssetsDS:ClassType@I(Class)
AssetsDS:DeviceType@I(DeviceType)
```

Define Asset Node

```
AssetsDS:Asset(Asset{AssetName, Vendor, Class, AssetType, DeviceType,
        BiosGuid, Versions, HostName, IP_Address_Type, IP_Address})
```

Asset Data

1	Asset	AssetName	Vendor	AssetType	Class	DeviceType	BiosGuid	Versions	HostName	IP_Address_Type	IP_Address
2	Asset-A100	Dell Server	Dell	Server	Virtual	Hardware	S100-BG	V3.2.4	Host-A100	IPV6	123.123.123.001
3	Asset-A200	Windows Office	Microsoft	Desktop	Physical	Software	D200-BG	C-2-5	Host-A200	IPV4	123.123.123.002
4	Asset-A300	Phone Systems	Plantronix	Phone	Physical	Com	PH100-BG	V8.3.1	Host-A300	IPV6	123.123.123.003
5	Asset-A400	Power Generator	PowerHouse	Power	Physical	Infra	PW100-BG	C2	Host-A400	IPV4	123.123.123.004
6	Asset-A500	Dell Server	Dell	Server	Virtual	Hardware	S200-BG	V3.4	Host-A500	IPV6	123.123.123.005
7	Asset-B100	Power Generator	PowerHouse	Power	Physical	Infra	PW200-BG	C2	Host-B100	IPV4	123.123.123.006

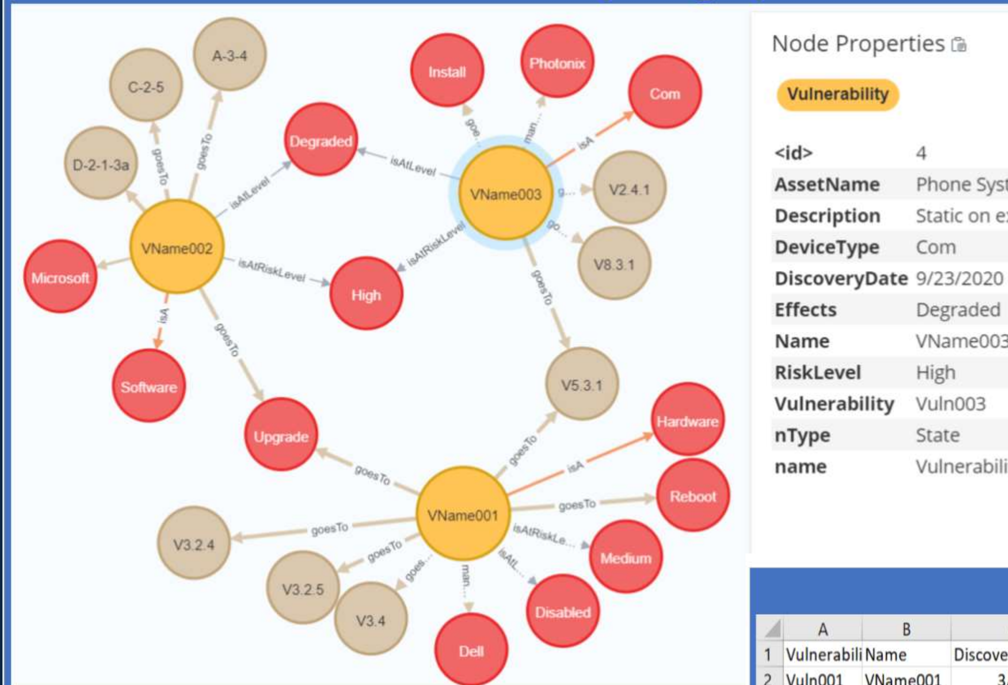
Assets have a single Asset Index node pointing to all Assets

Assets have multiple Index nodes each describing various Asset characteristics

Vulnerability Subgraph

Constructing the Vulnerability Subgraph

Vulnerabilty Subgraph



Vulnerability 003 and 002 are at High Risk

Vulnerability 001 and 002 require an Upgrade. 001 also requires a Reboot, etc.

Vulnerability Grammars

Define Vulnerability Index Node

```
VulnerabilitiesDS:Vulnerabilities@I()
```

Define RiskLevel, DeviceType, Effects, Vendor Index Nodes

```
VulnerabilitiesDS:RiskLevel@I(RiskLevel)
VulnerabilitiesDS:DeviceType@I(DeviceType)
VulnerabilitiesDS:Effects@I(Effects)
VulnerabilitiesDS:Vendor@I(Vendor)
```

Define Vulnerabilty Node

```
VulnerabilitiesDS:Vulnerability(Vulnerability{Name, DiscoveryDate,
     Description, DeviceType, AssetName, Effects, RiskLevel,
     DeviceType, Versions@;, ImplemImpact@;})
```

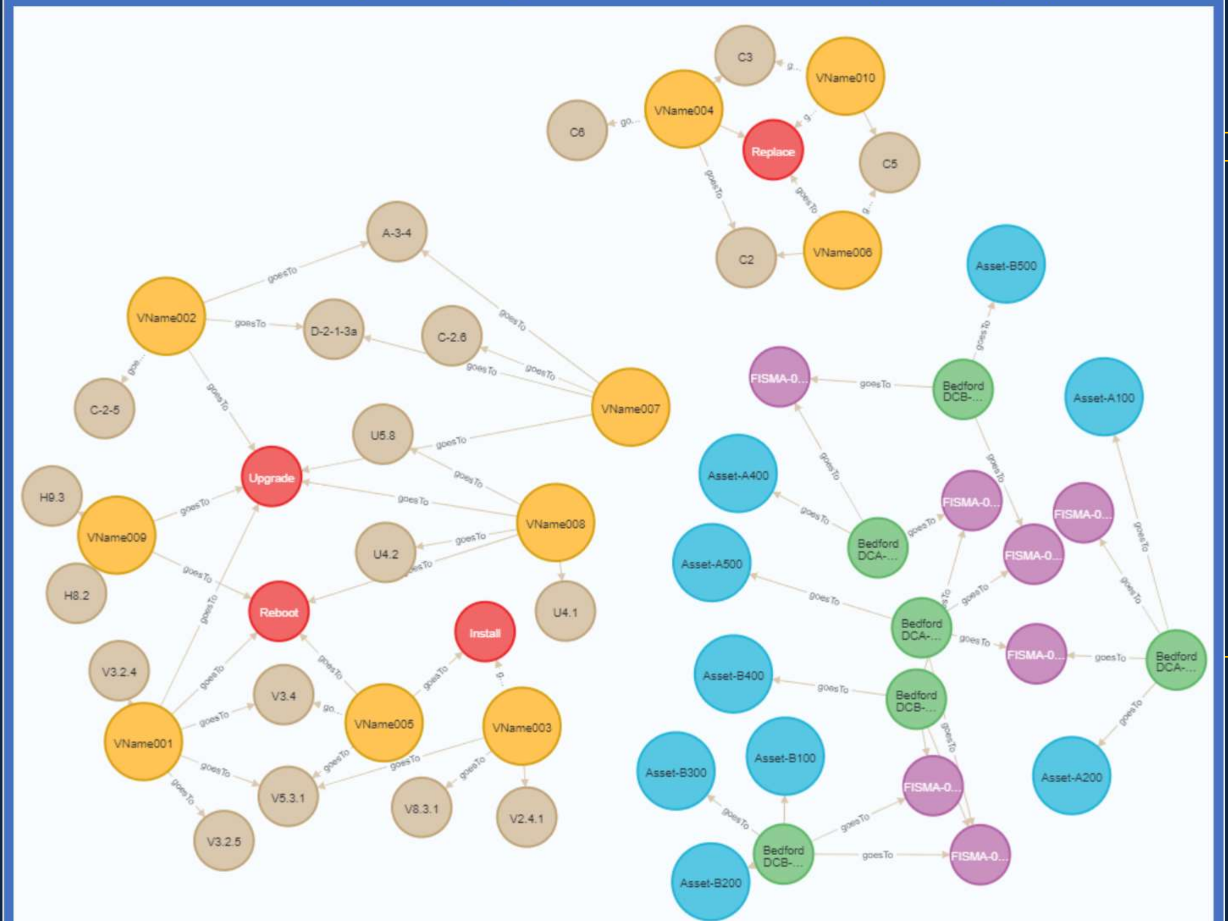
Vulnerability Data

	A	B	C	D	E	F	G	H	I	J	K
1	Vulnerabili	Name	DiscoveryDate	Description	DeviceType	AssetName	Effects	RiskLevel	Vendor	ImplemImpact	Versions
2	Vuln001	VName001	3/15/1998	Network Failures	Hardware	Dell Server	Disabled	High	Dell	Upgrade;Reboo	V3.2.4; V3.2.5; V3.4;V5.3.1
3	Vuln002	VName002	4/5/2021	Performance Impact	Software	Windows Office	Degraded	Medium	Microsoft	Upgrade	A-3-4; C-2-5; D-2-1-3a
4	Vuln003	VName003	9/23/2020	Static on external conn	Com	Phone Systems	Degraded	Low	Photonix	Install	V5.3.1;V2.4.1;V8.3.1
5	Vuln004	VName004	3/6/2001	Reduced Power Outpu	Infra	Power Generator	Degraded	Medium	Powerwerx	Replace	C6;C2;C3
6	Vuln005	VName005	5/10/2020	Device access disabled	Hardware	Dell Server	Disabled	High	Dell	Install;Reboot	V3.4;V5.3.1
7	Vuln006	VName006	8/12/2019	Faulty oputput voltage	Infra	Power Generator	Disabled	High	Powerwerx	Replace	C2;C5
8	Vuln007	VName007	7/25/2004	Network Failures	Software	Windows Office	Disabled	Medium	Microsoft	Upgrade	A-3-4; C-2.6; D-2-1-3a
9	Vuln008	VName008	7/10/2005	Performance Impact	Software	Unix OS	Degraded	Low	IBM	Upgrade;Reboo	U4.2;U4.1;U5.8
10	Vuln009	VName009	9/3/2010	Network Failures	Hardware	IBM Server	Disabled	Medium	IBM	Upgrade;Reboo	H9.3;H8.2
11	Vuln010	VName010	10/4/2014	Voice stream interrupt	Com	Phone Systems	Degraded	Low	Photonix	Replace	C3;C5

Combined Master Graph

Putting it All Together

Data Center, Assets, and Vulnerability Subgraphs



The Master Graph contains all three subgraphs, and describes the Data Centers, Assets, and Vulnerabilities.

Assets (in Blue) are not connected to Vulnerabilities (in Gold) as shown in the Master Graph and the following Cypher Query.

Show Asset-Vulnerability Connections

```
neo4j$ match (a:Asset)-[]→(b:Vulnerability) return a, b
```

Table

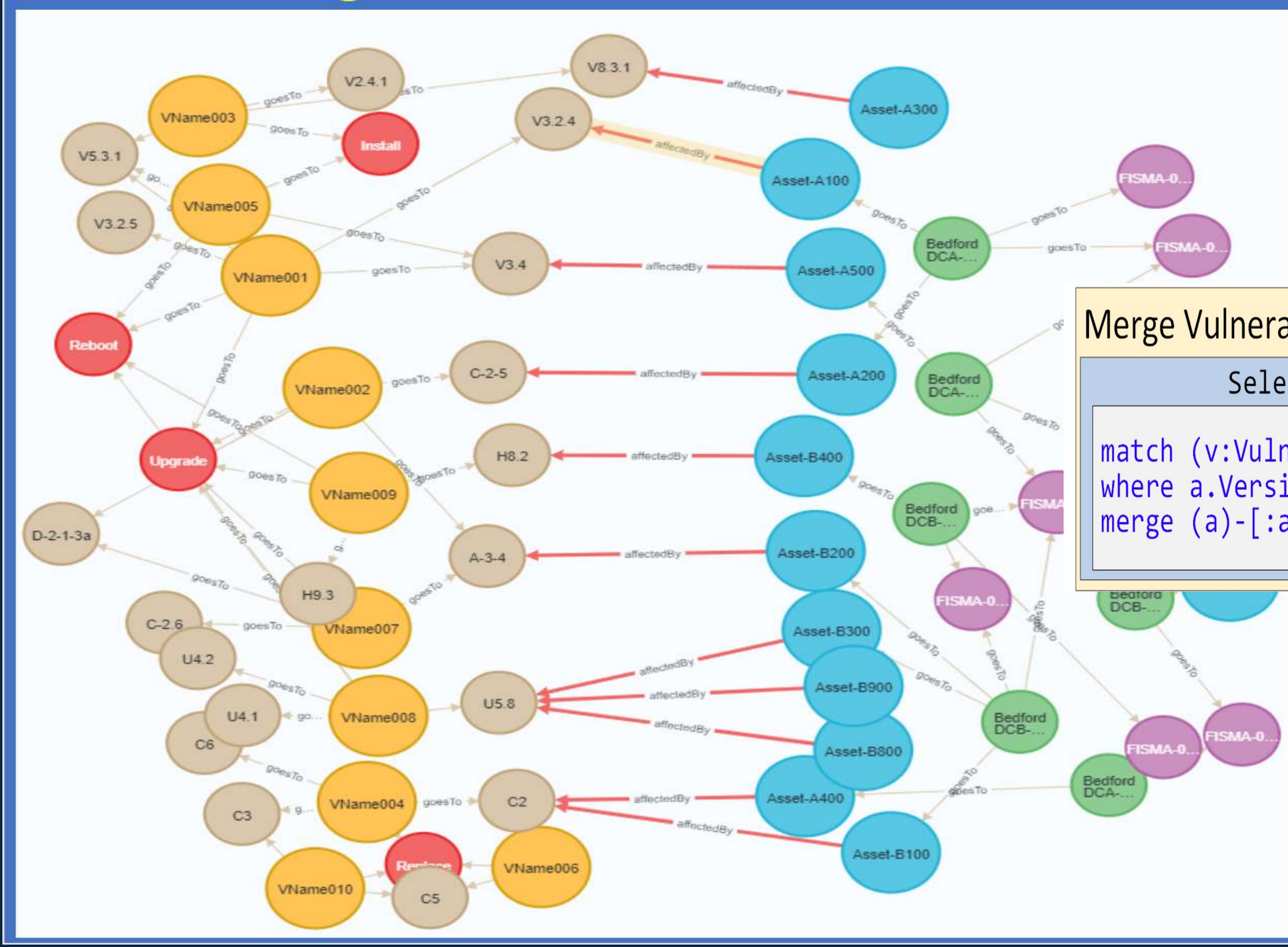
(no changes, no records)

The underlying data sources have no such connections.

Merging Vulnerability and Assets

Building New Relationships

Merged Vulnerabilities and Assets



Standard Cypher, the language of graphs, as shown below, can be used to bind these entities together. The Bloom interface also provides this capability.

Merge Vulnerabilities and Affected Assets Together

Select matching nodes and build relationships

```
match (v:Vulnerability)-[gt:goesTo]->(ve:Versions), (a:Asset)
where a.Versions = ve.Versions and v.DeviceType = a.DeviceType
merge (a)-[:affectedBy]->(ve)
```

We can also see that some Assets, A330, B800, and B900 for example, are not bound in our Data Sets to a Data Center.
This rogue assets might require further exploration.

Introduction to Cypher Patterns

The Heart of Cypher Syntax

Understanding Patterns

ASCII Art

1. A Node has a Relationship to Another Node

2. (node) – [relationship] – (node)

   (node) - [Relationship] - (node)

3. (:Label {Property}) - [Relationship] - (:Label {Property})

4. (e1:Employee{name:"ray"}) - [r1:worksWith] - (e2:Employee{name:"steve"})

5. (e1:Employee{name:"ray"}) - [r1:worksWith] -> (e2:Employee{name:"steve"})

   (e1:Employee{name:"ray"}) - [r2:isAMemberOf] -> (g1:Group{name:"research"})

rlukas

MITRE

Generated Cypher Code

Load Nodes and Linkages Mechanism

```
1                                    2                         3                       4
:auto USING PERIODIC COMMIT 300  load csv with headers from "file:///dataSet.csv" as dataSetRow
    with dataSetRow 5

6 ForEach(_ In Case When ((dataSetRow.dataCenterColName Is  not  Null)) Then [1] Else [] End|

   Merge (dataCenterInstance:DataCenter{name:"DataCenter",
7                                    dataCenterID:trim(dataSetRow.datacenter_id_derived)})

   set dataCenterInstance.nType = trim("State") 8

  ForEach(_ In Case When ((dataSetRow.ServerID Is  not  Null)) Then [1] Else [] End|
     Merge (serverInstance:Server{name:"Server", ServerID:trim(dataSetRow.ServerID)})
     set serverInstance.macaddress = trim(dataSetRow.macAddress)
9    set serverInstance.dnsname = trim(dataSetRow.dnsname)
     set serverInstance.nType = trim("State")

     merge (dataCenterInstance)-[dataCenterToServerLink:contains]->(serverInstance)

  ) //end For ((dataSetRow.ServerID Is  not  Null)) command

6 ) //end For ((dataSetRow.dataCenterColName Is  not  Null)) command
```

Generated Cypher Code

Multifield

Multifield Mechanism

```
:auto USING PERIODIC COMMIT 300  load csv with headers from "file:///newVersion.csv" as osRow
    with osRow

(1)  ForEach(_ In Case When ((osRow.ServerID Is  not  Null)) Then [1] Else [] End|
      Merge (serverInstance:Server{name:"Server", ServerID:trim(dataSetRow.ServerID)})

      ForEach(_ In Case When (osRow.os is not null) Then [1] Else [] End|
(2)     ForEach(osItem in split(osRow.os, ';') Then [1] Else [] End|
(3)       ForEach(_ IN CASE WHEN (osItem  <> '') Then [1] Else [] End|
(4)
            merge (osInstance:OS{name:"OS", operSys:trim(osItem)})
(5)         set osInstance.nType = trim("Index")
            merge (serverInstance)-[:isA]->(osInstance)
          )
        )
      )
    )
```

MITRE

MITRE Labs inspires breakthroughs in applied science and advanced technology to transform the future of U.S. scientific and economic leadership. Our goal: Deliver disruptive innovation to support our mission of solving problems for a safer world.

Please feel free to contact me with any questions

Thank You

Ray Lukas (rlukas@mitre.org)
Jim Lockett (jdlockett@mitre.org)

rlukas@MITRE.org