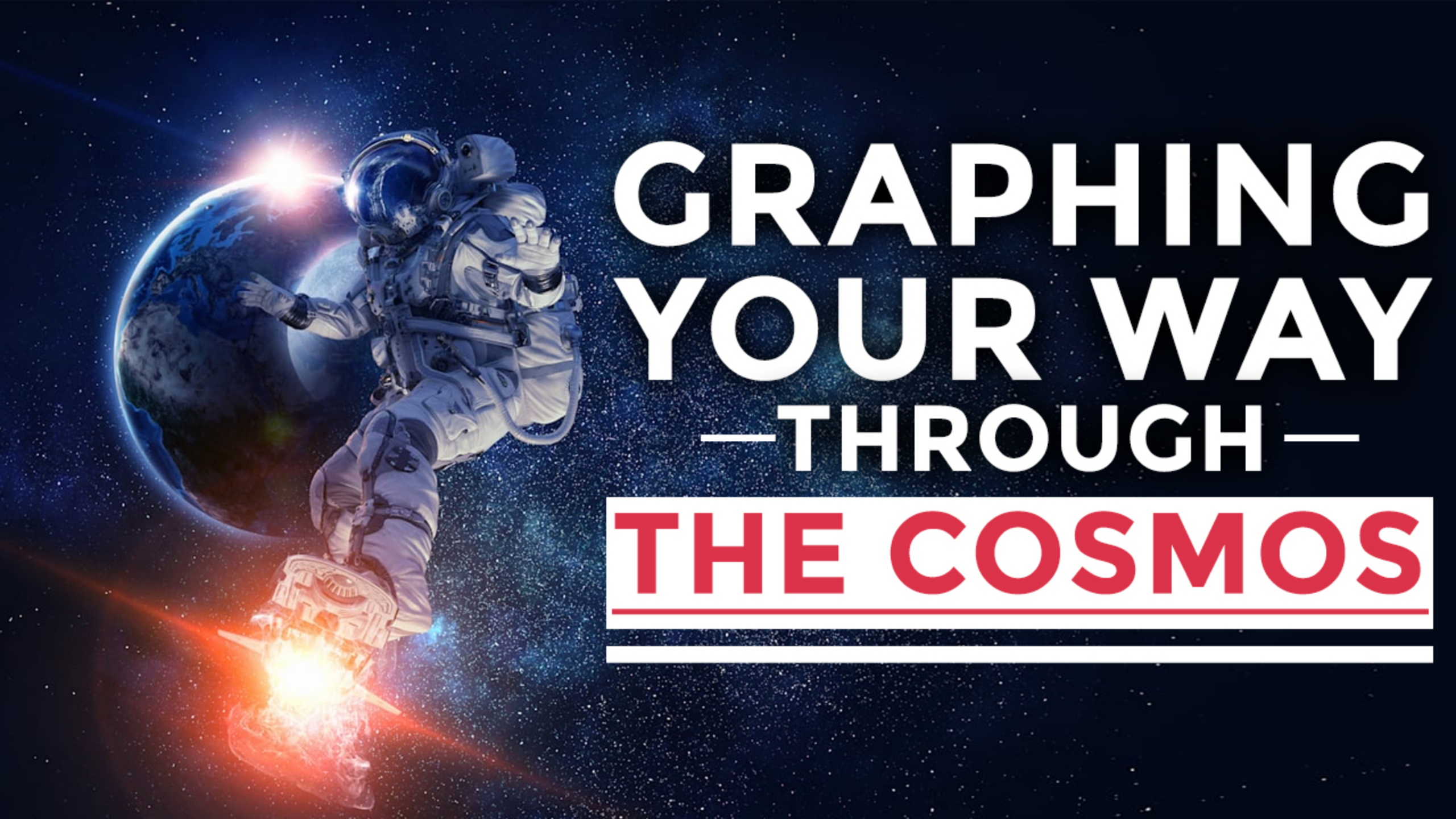
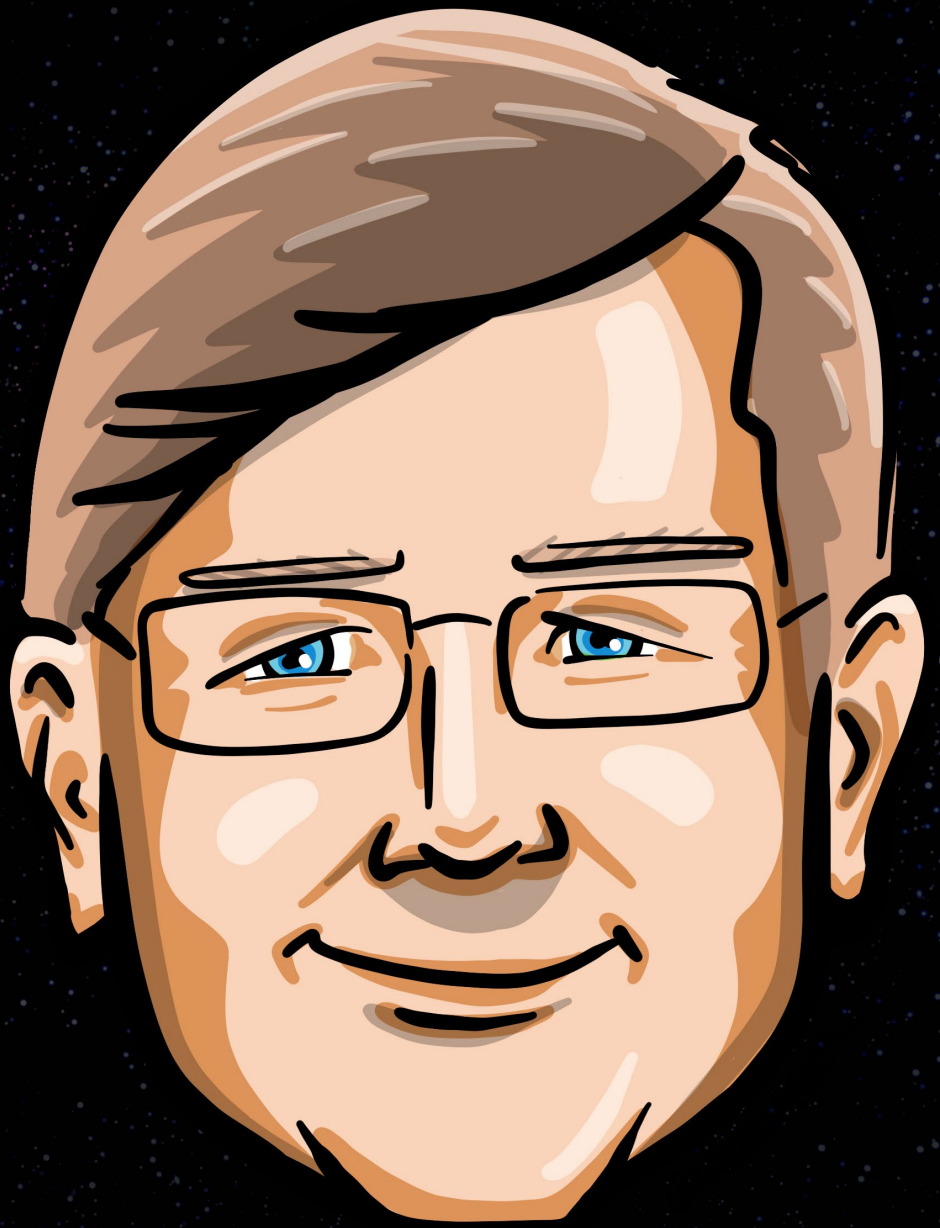
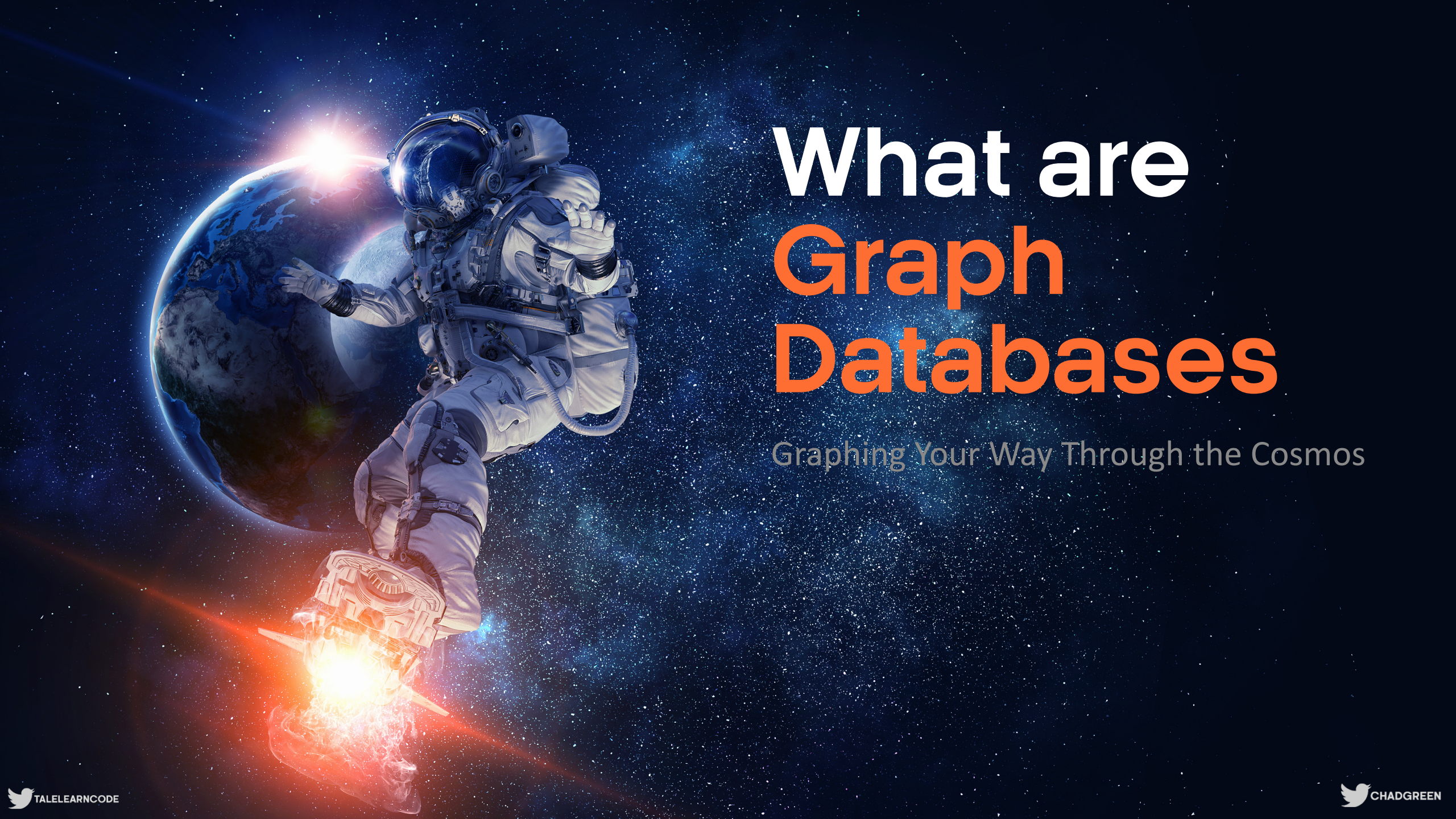

GRAPHING
YOUR WAY
—THROUGH—
THE COSMOS

Who is Chad Green



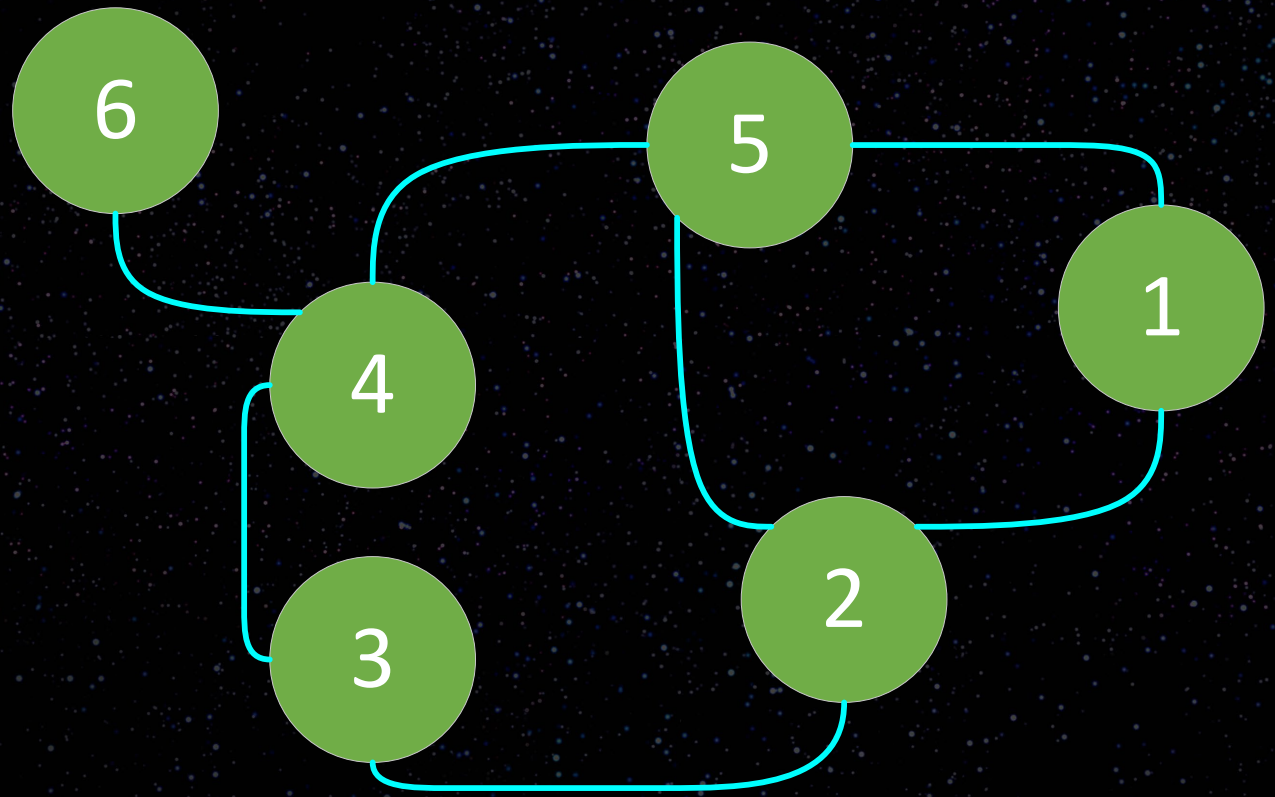
- chadgreen@chadgreen.com
- TaleLearnCode
- TaleLearnCode.com
- ChadGreen & TaleLearnCode
- ChadwickEGreen


What are
Graph
Databases
Graphing Your Way Through the Cosmos
TALELEARNCODE
CHADGREEN

What is a Graph

Structure amounting to set of objects in which some pares of the objects are in some sense related

What is a Graph

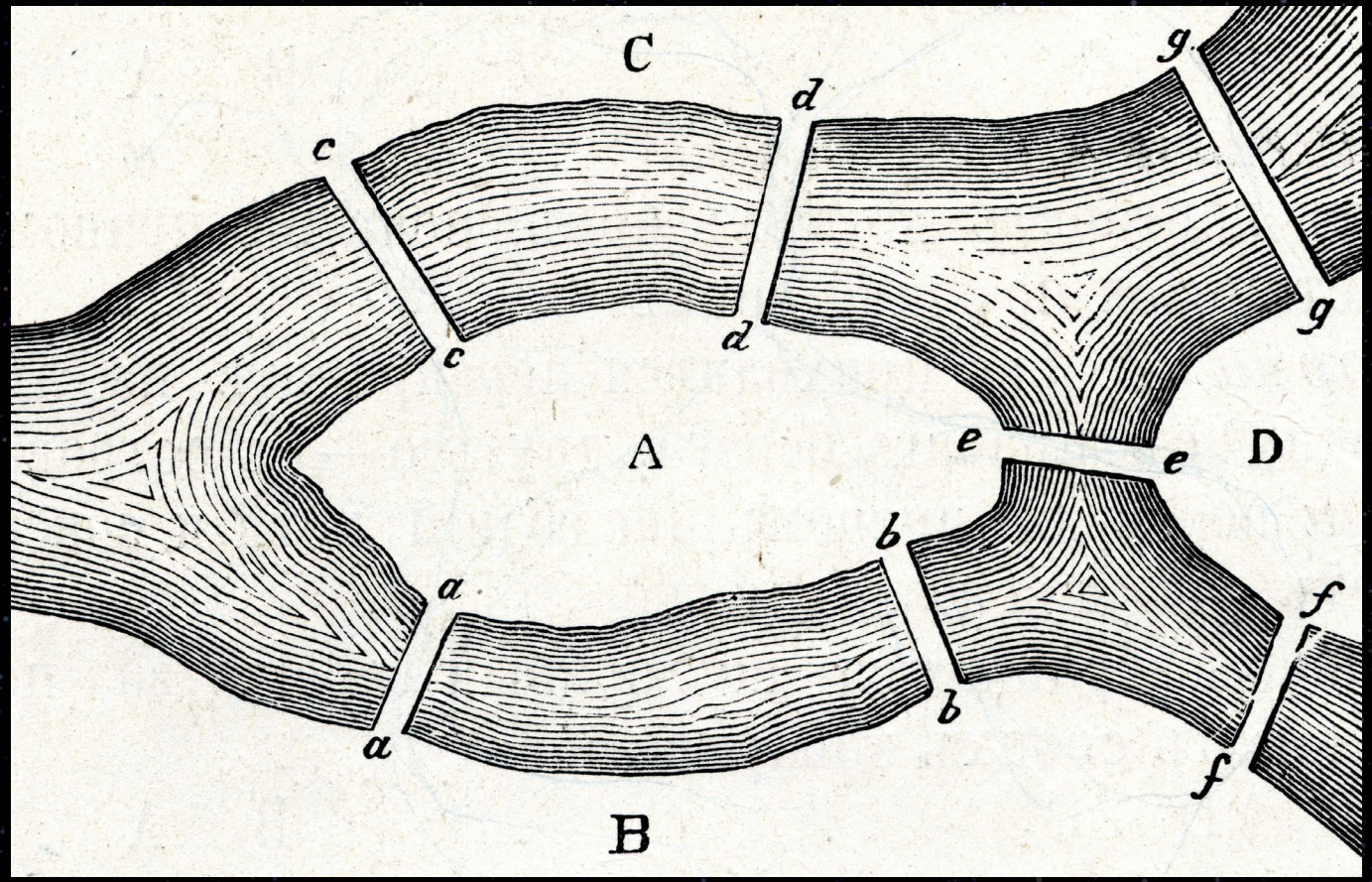


What is a Graph

$$G = (V, E)$$

History Graph Theory

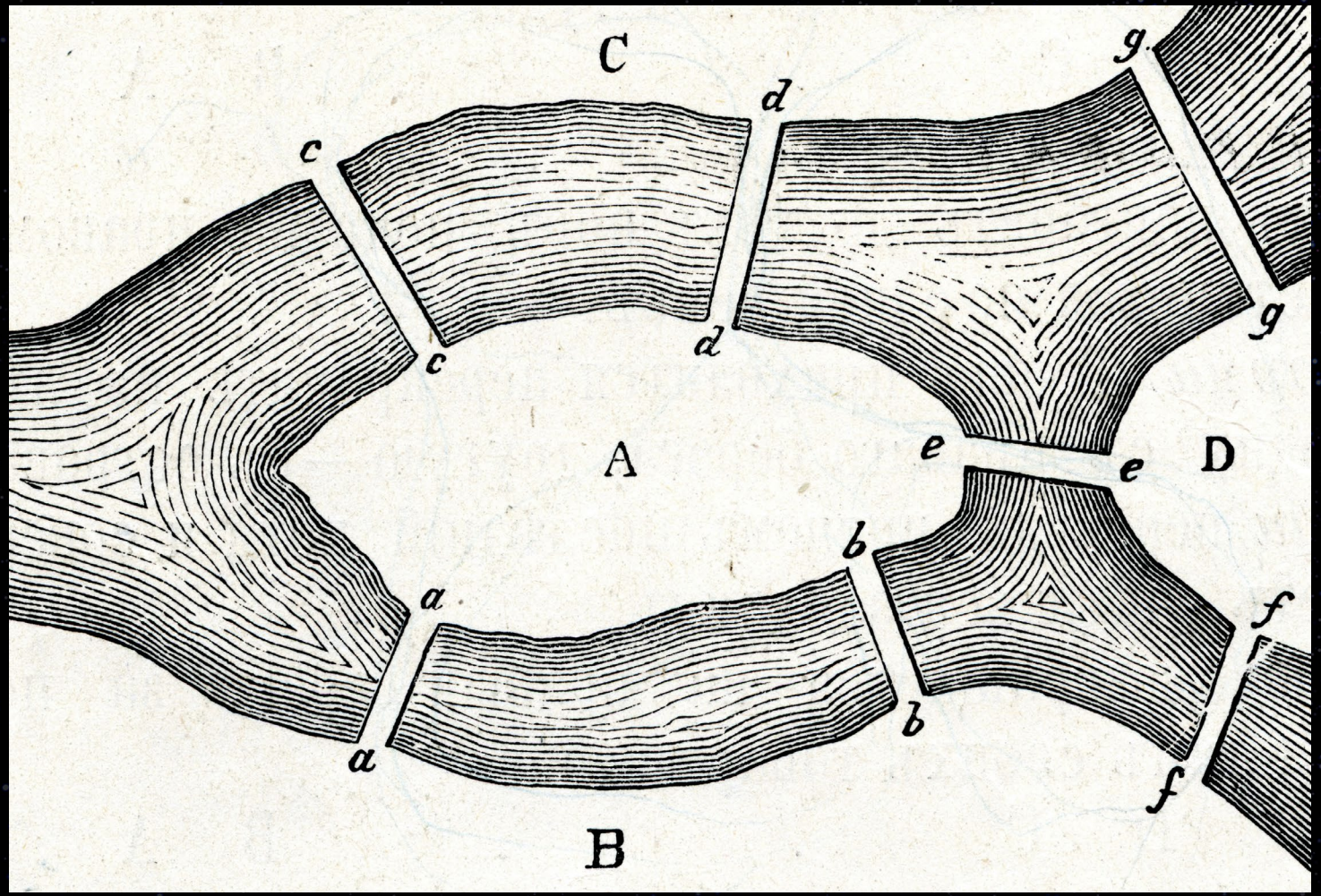
History Graph Theory



History Graph Theory

Leonard Euler

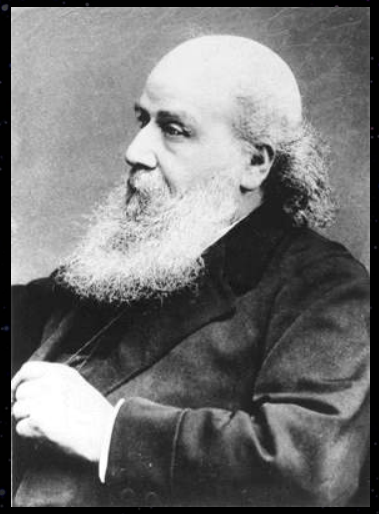
History Graph Theory



Solutio problematis ad geometriam situs pertinentis

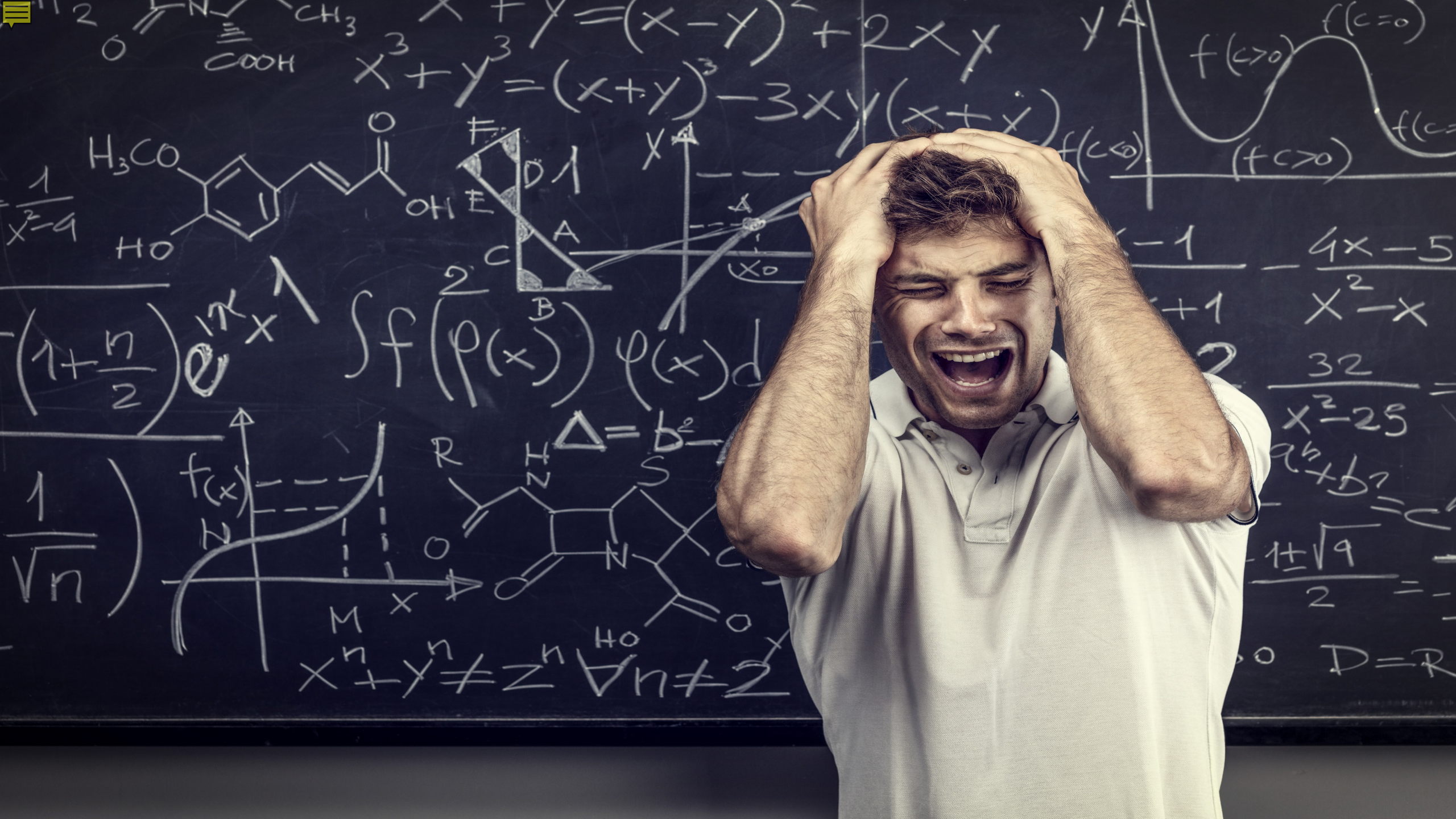
The solution of a problem relating to the geometry of position

History Graph Theory



Application of Graph Theory

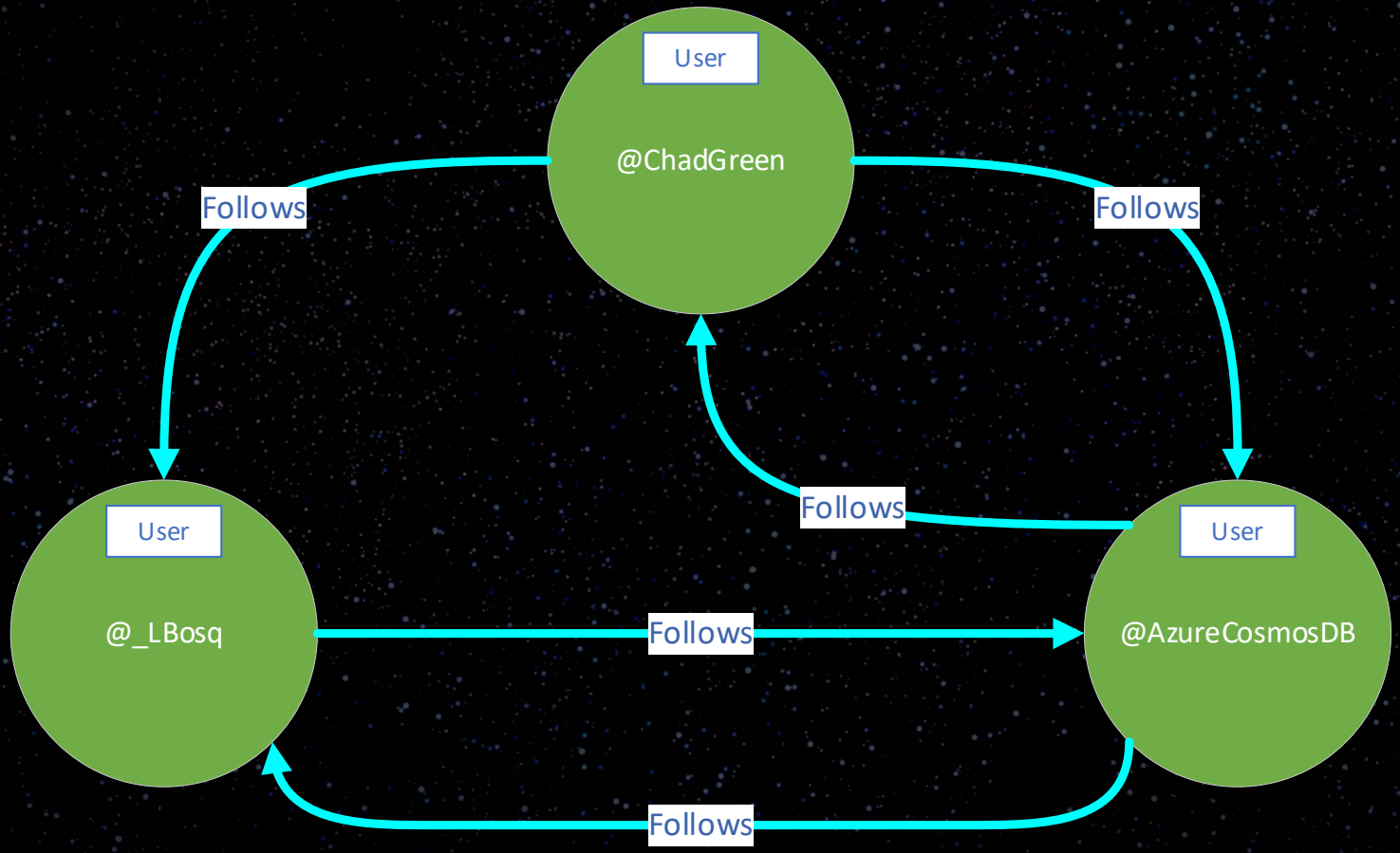
- Linguistics
- Physics and Chemistry
- Social Sciences
- Biology
- Computer Science


H_3CO
HO
OH
COOH
$x^3+y^3=(x+y)^3-3xy(x+y)$
$\Delta=b^2-$
$x^n+y^n\neq z^n \quad \forall n\neq 2$
f(c=0
f(c>0
f(c<0
D=R

What is a Graph

Allows us to model all kinds of scenarios

What is a Graph



What is a Graph Database

Database that uses graph structures to represent and store data

What is a Graph Database

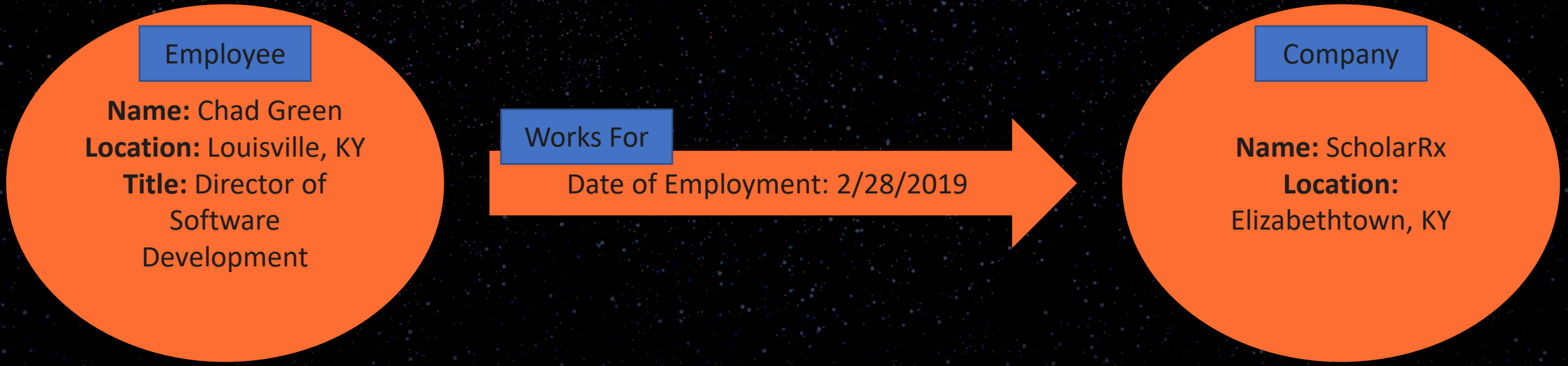
Represents data as it exits in the real world as they are naturally connected

Property Graph Model

Contains **nodes** (vertices) and **relationships** (edges)

Nodes and relationships contain **properties**

Relationships are **named** and **directed** with a **start** and **end** node



The Power of Graph Databases

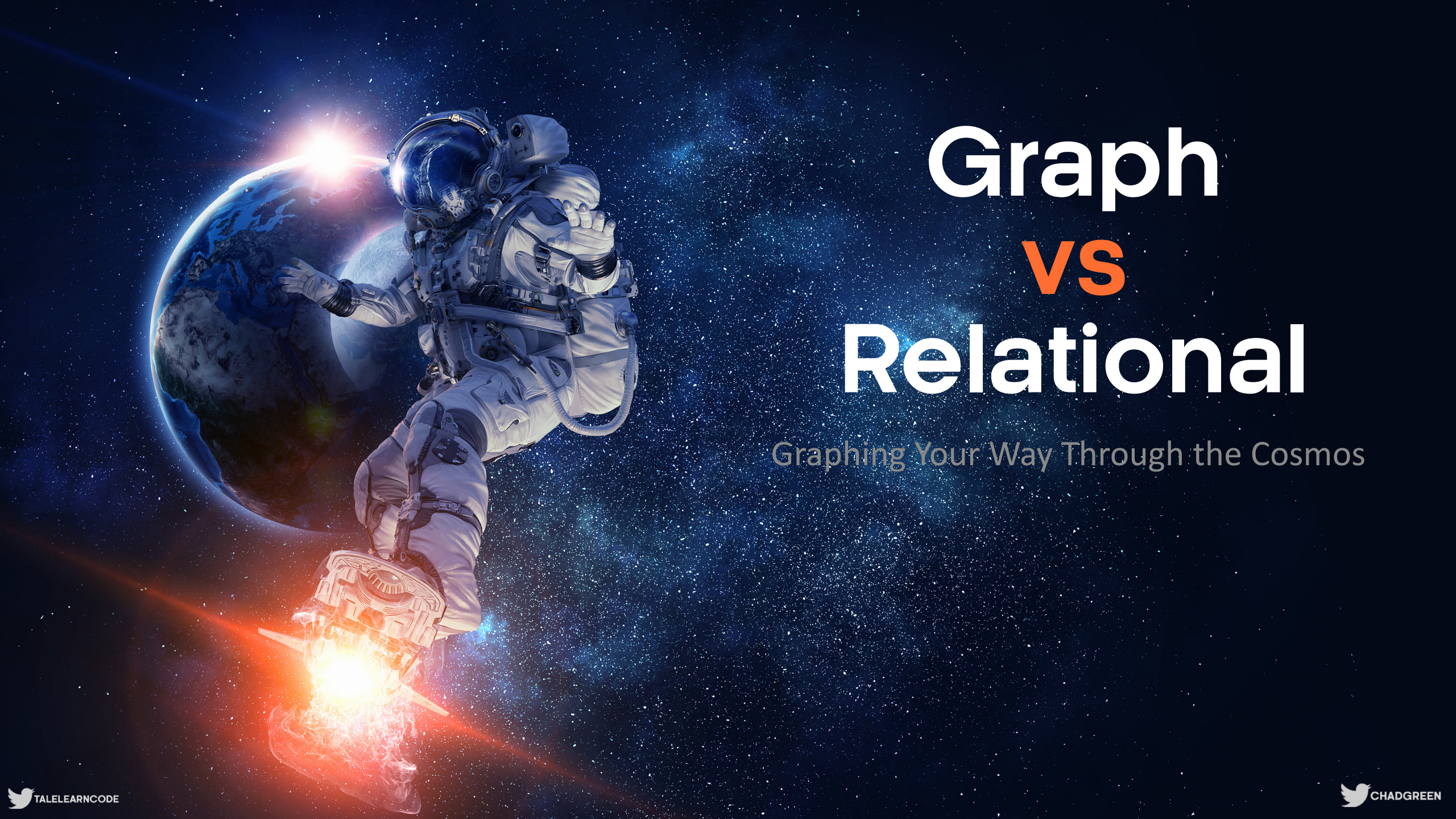
Performance

Flexibility

Agility

Common Graph Use Cases

- Internet of Things
- Asset Management
- Recommendations
- Fraud Detection
- Data Integration
- Identity and Access Management
- Social Networks
- Communication Networks
- Genomics
- Epidemiology
- Sematic Web
- Search


Graph
vs
Relational
Graphing Your Way Through the Cosmos
TALELEARNCODE
CHADGREEN

Graph vs Relational

Relational	Graph
Tables	Vertices (Nodes)
Schema with nullables	No schema
Relations with foreign keys	Relation is first class citizen
Related data fetched with joins	Related data fetched with a pattern

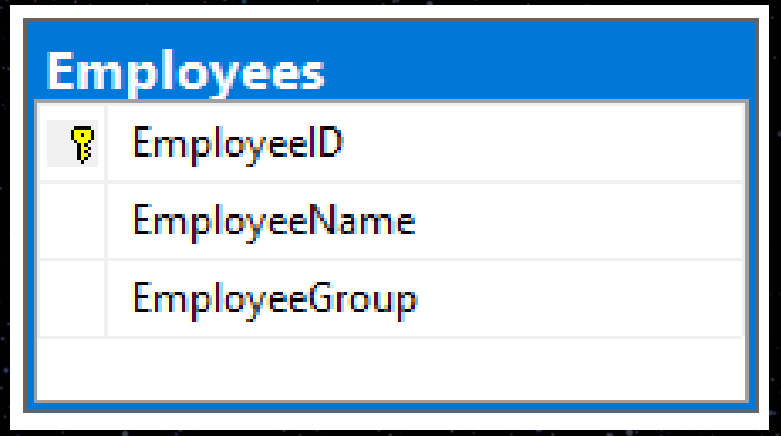
Human Resource Data

Graph Databases vs Relational Databases

Human Resource Data

Graph vs Relational

EmployeeId	EmployeeName	EmployeeGroup
1	Willis B. Hawkins	Sales
2	Neil S. Vega	Sales
3	Ada C. Lavigne	Engineering



Human Resource Data

Graph vs Relational

```
-- Create the Employee Table
CREATE TABLE Employees
(
    EmployeeID    INT          IDENTITY(1,1),
    EmployeeName  VARCHAR(64),
    EmployeeGroup VARCHAR(32),
    CONSTRAINT pkcEmployees PRIMARY KEY CLUSTERED (EmployeeId)
)
GO

-- Populate the Employee Table
INSERT INTO Employees (EmployeeName, EmployeeGroup)
             VALUES ('Willis B. Hawkins', 'Sales'),
                    ('Neil S. Vega',      'Sales'),
                    ('Ada C. Lavigne',    'Engineering');
GO
```

Human Resource Data

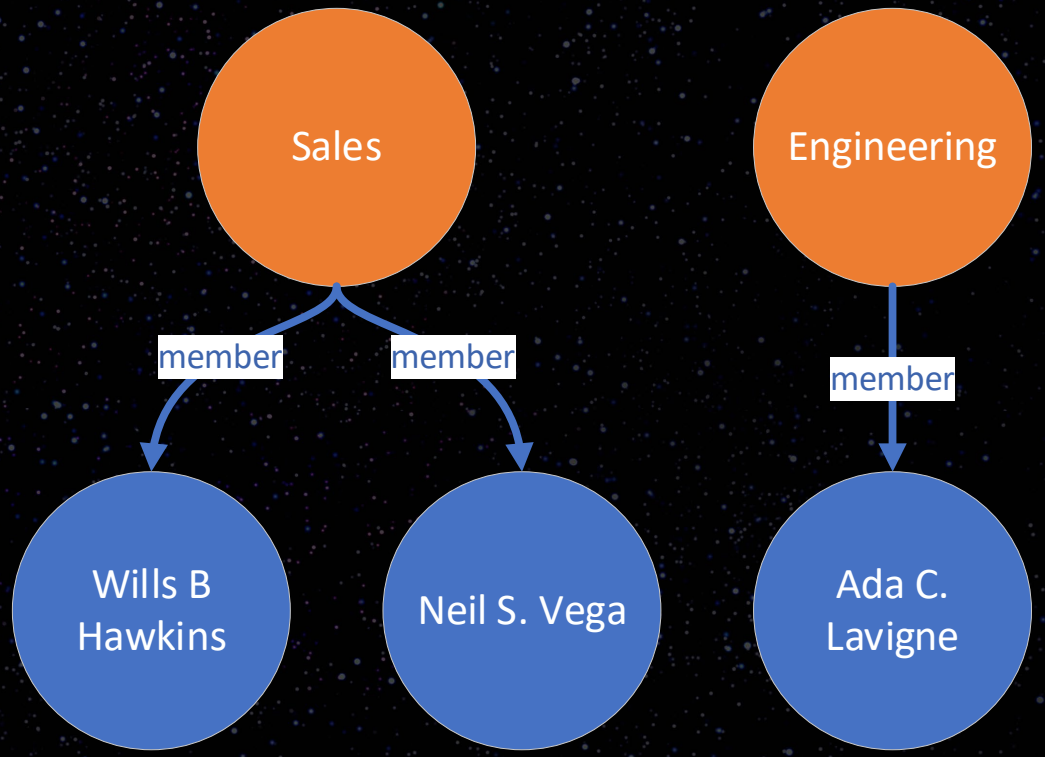
Graph vs Relational

```
// Create group nodes
g.addV('group').property('id', 'Sales’)
g.addV('group').property('id', 'Engineering’)

// Create employee nodes
g.addV('employee').property('id', 'Willis B. Hawkins’)
g.addV('employee').property('id', 'Neil S. Vega’)
g.addV('employee').property('id', 'Ada C. Lavigne’)

// Create relationships between groups and employees
g.V('Sales').addE('member').to(g.V('Willis B. Hawkins’))
g.V('Sales').addE('member').to(g.V('Neil S. Vega’))
g.V('Engineering').addE('member').to(g.V('Ada C. Lavignee'))
```

Human Resource Data

Graph vs Relational

EmployeeId	EmployeeName	EmployeeGroup
1	Willis B. Hawkins	Sales
2	Neil S. Vega	Sales
3	Ada C. Lavigne	Engineering



3 rows, 3 columns

8 documents (vertices and edges)

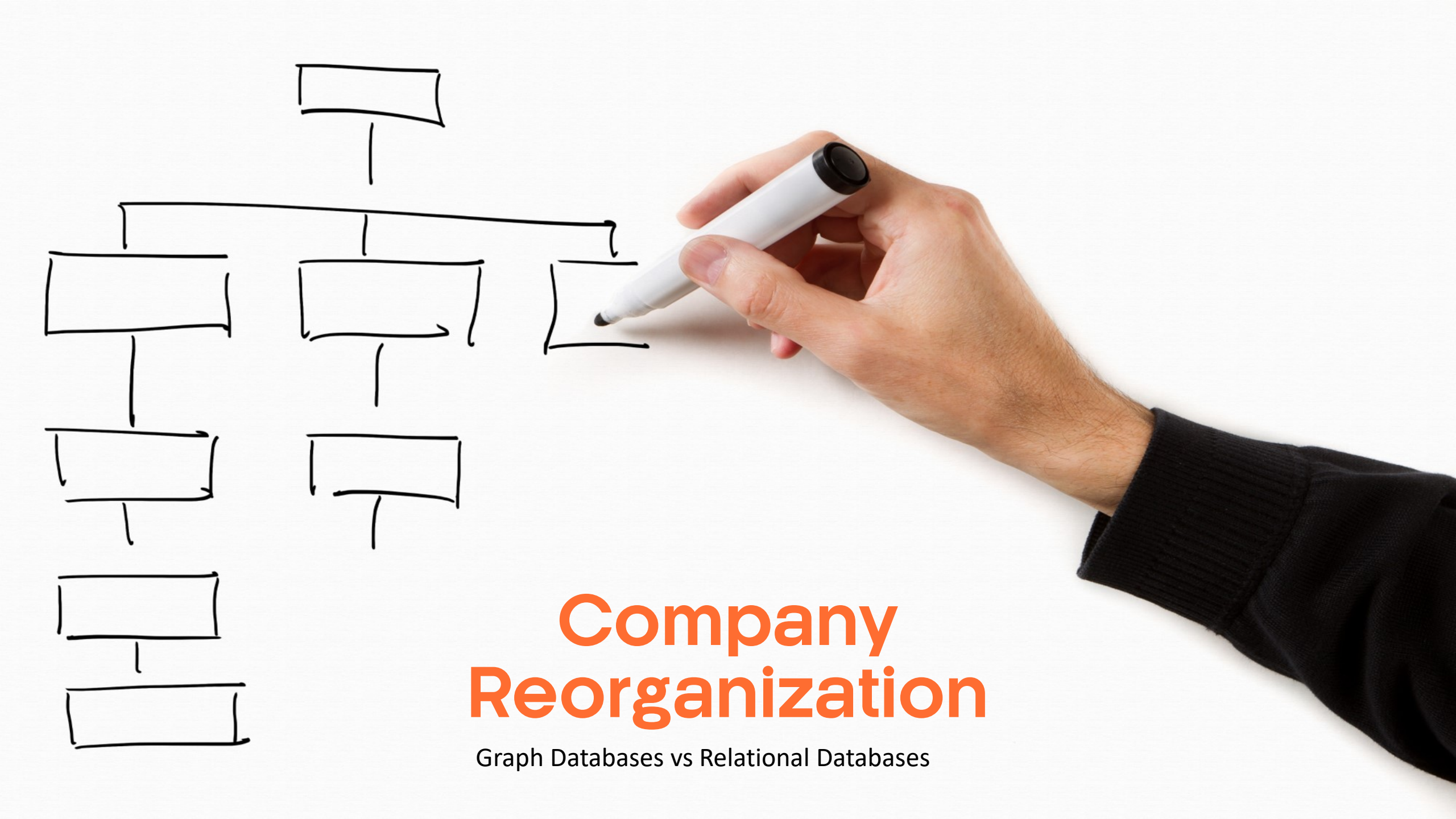
Human Resource Data

Graph vs Relational

EmployeeId	EmployeeName	EmployeeGroup
1	Willis B. Hawkins	Sales
2	Neil S. Vega	Sales
3	Ada C. Lavigne	Engineering


```
SELECT * FROM Employees;
```

```
g.V().hasLabel('employee')
```



Company Reorganization

Graph Databases vs Relational Databases

Employees can now belong to multiple groups

Graph vs Relational

```
-- Create the Groups table
CREATE TABLE Groups
(
  GroupId   INT          IDENTITY(1,1),
  GroupName VARCHAR(64),
  CONSTRAINT pkcGroups PRIMARY KEY CLUSTERED (GroupId)
)
```

Employees can now belong to multiple groups

Graph vs Relational

```
-- Create the Employee_Group join table
CREATE TABLE Employee_Group
(
  GroupId INT,
  EmployeeId INT,
  CONSTRAINT pkcEmployeeGroup PRIMARY KEY CLUSTERED (GroupId, EmployeeId),
  CONSTRAINT fkEmployeeGroup_Groups    FOREIGN KEY (GroupId)    REFERENCES Groups(GroupId),
  CONSTRAINT fkEmployeeGroup_Employees FOREIGN KEY (EmployeeId) REFERENCES Employees(EmployeeId)
)
```

Employees can now belong to multiple groups

Graph vs Relational

```
-- Populate the Employee_Group table from Employees and Groups
INSERT INTO Employee_Group (GroupId, EmployeeId)
SELECT Groups.GroupId,
       Employees.EmployeeId
  FROM Employees,
       Groups
 WHERE Groups.GroupName = Employees.EmployeeGroup
```

Employees can now belong to multiple groups

Graph vs Relational

```
-- Drop the Employees.EmployeeGroup column that is no longer valid
ALTER TABLE Employees DROP COLUMN EmployeeGroup
```

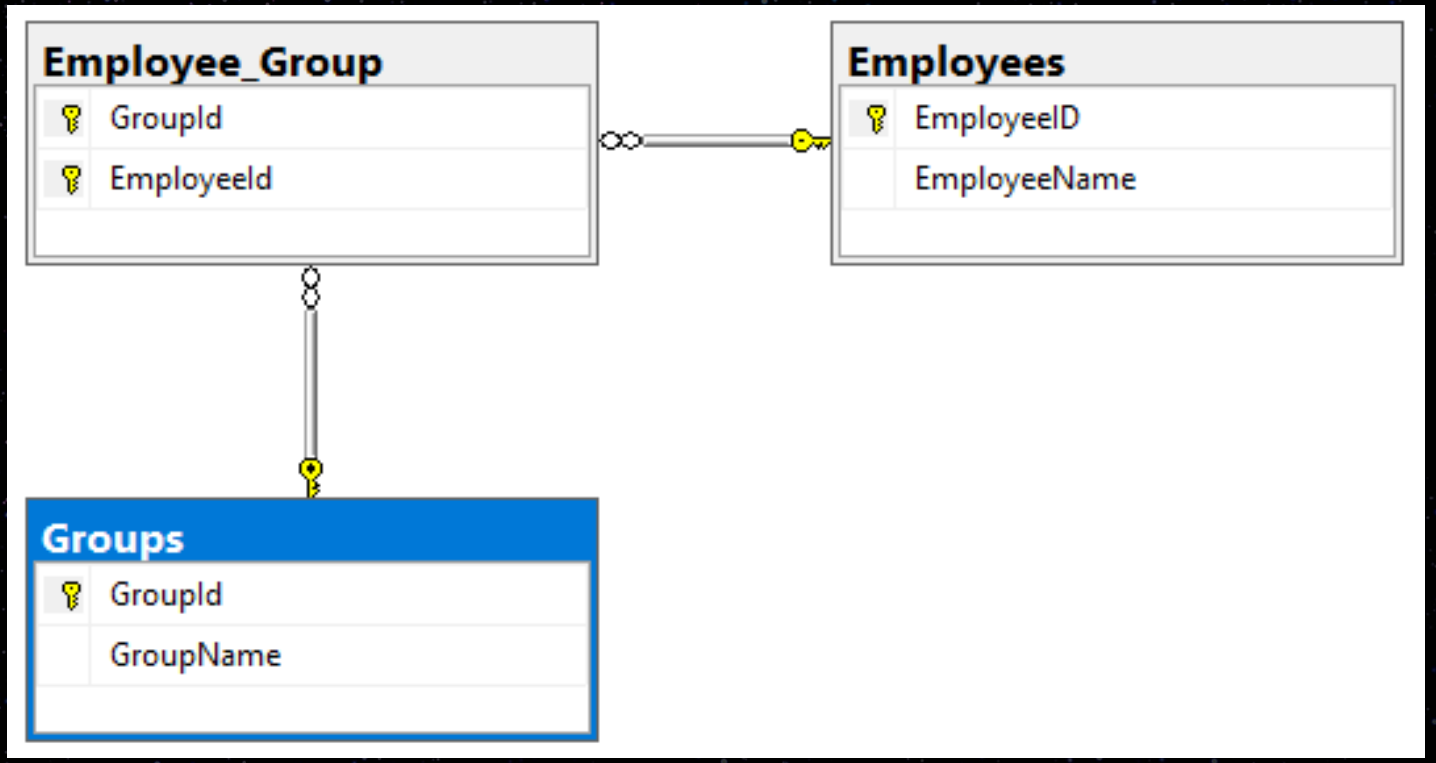
Employees can now belong to multiple groups

Graph vs Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales

GroupId	EmployeeId
1	3
2	1
2	2



Employees can now belong to multiple groups

Graph vs Relational

```
// Add link to existing node
g.V('Sales').addE('member').to(g.V('Ada C. Lavigne’))
```

Employees can now belong to multiple groups

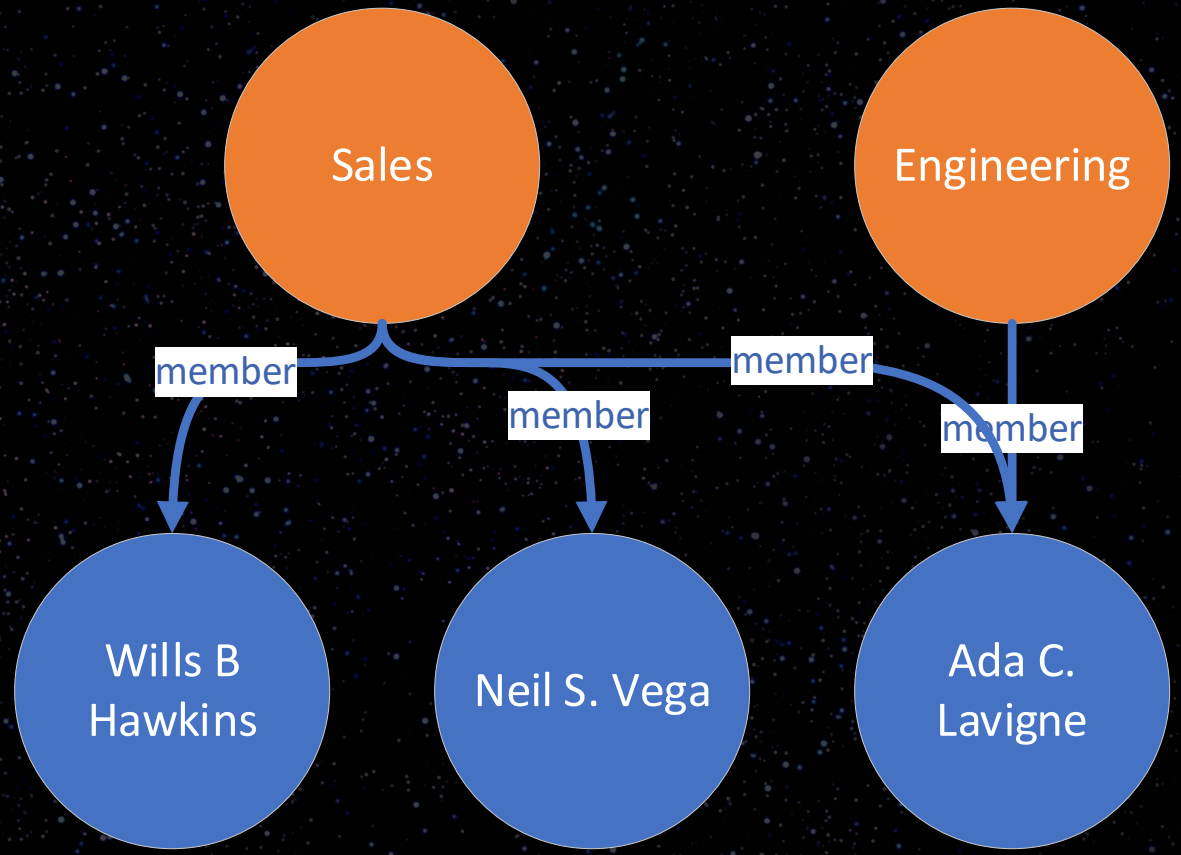
Graph vs Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales

GroupId	EmployeeId
1	3
2	1
2	2

Added 2 tables; 6 rows; 4 new columns
Removed a column



+1 document

Employees can now belong to multiple groups

Graph vs Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne


```
SELECT Employees.EmployeeId,
       Employees.EmployeeName
  FROM Employees
 INNER JOIN Employee_Group
         ON Employee_Group.EmployeeId = Employees.EmployeeId
 INNER JOIN Groups
         ON Groups.GroupId = Employee_Group.GroupId
 WHERE Groups.GroupName = 'Sales'
```

```
g.V('Sales').outE('member’).inV()
```

Corporate Merger

Graph Databases vs Relational Databases

Nested Groups

Graph vs Relational

```
-- Create the new Product Group
INSERT INTO Groups (GroupName) VALUES ('Product Group')
```

Nested Groups

Graph vs Relational

```
-- Associate everyone to the new Product Group
INSERT INTO Employee_Group (GroupId, EmployeeId)
SELECT Groups.GroupId,
       Employees.EmployeeId
  FROM Groups,
       Employees
 WHERE Groups.GroupName = 'Product Group
```

Nested Groups

Graph vs Relational

```
-- Create the Group/Group union table
CREATE TABLE Group_Group
(
  ParentGroupId INT,
  ChildGroupId  INT,
  CONSTRAINT pkcGroup_Group PRIMARY KEY CLUSTERED (ParentGroupId, ParentGroupId),
  CONSTRAINT fkGroupGroup_Groups_Parent FOREIGN KEY (ParentGroupId) REFERNCES Groups(GroupId),
  CONSTRAINT fkGroupGroup_Groups_Child  FOREIGN KEY (ChildGroupId)  REFERNCES Groups(GroupId)
)
```

Nested Groups

Graph vs Relational

```
-- Relate the child groups to the parent group
INSERT INTO Group_Group (ParentGroupId, ChildGroupId)
SELECT (SELECT GroupId FROM Groups WHERE GroupName = 'Product Group'),
       Groups.GroupId
  FROM Groups
 WHERE Groups.GroupName <> 'Product Group'
```

Nested Groups

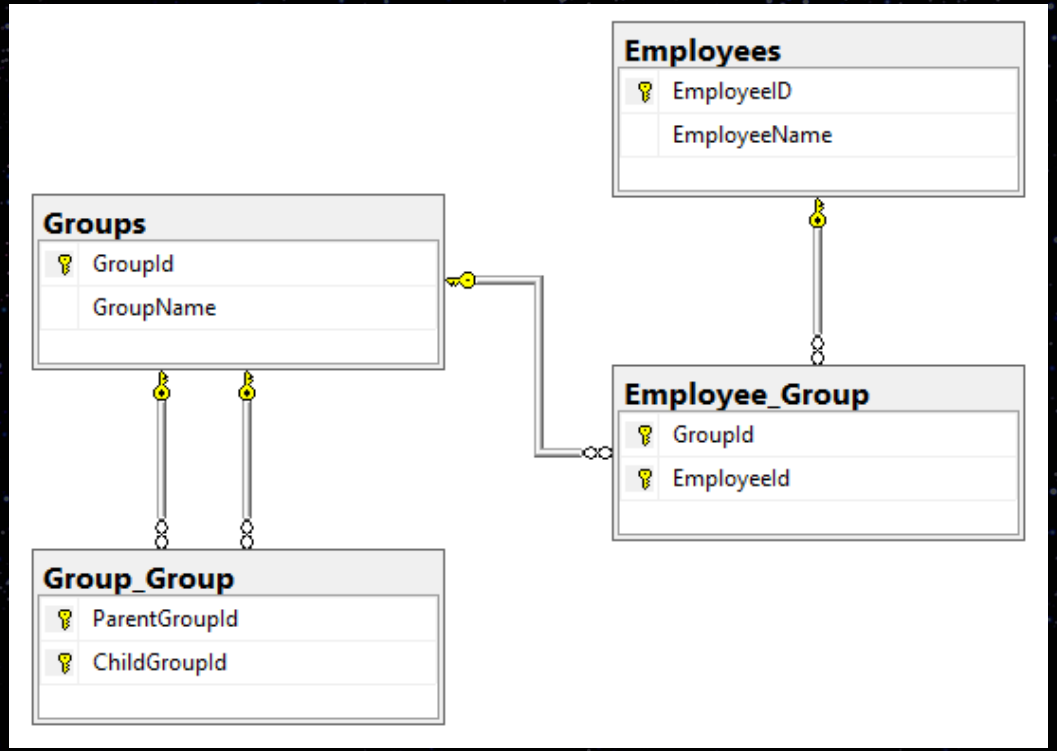
Graph vs Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales
3	Product Group

ParentGroupId	ChildGroupId
3	1
3	2

GroupId	EmployeeId
1	3
2	1
2	2
2	3
3	1
3	2
3	3



Nested Groups

Graph vs Relational

```
// Add supergroup node
g.addV('group').property('id', 'Product Group')

// Link to adjacent nodes
g.V('Product Group').addE('contains_subgroup').to(g.V('Engineering’))
g.V('Product Group').addE('contains_subgroup').to(g.V('Sales'))
```

Nested Groups

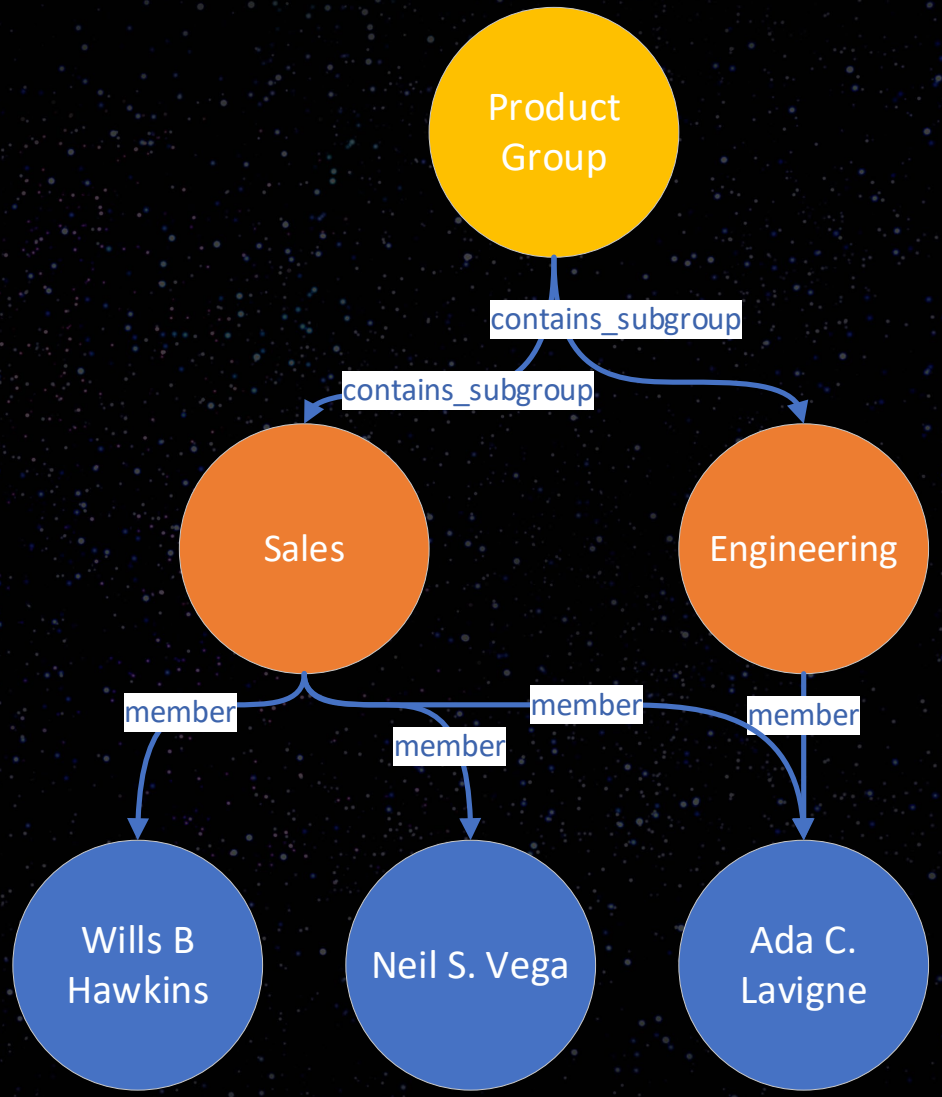
Graph vs Relational

EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales
3	Product Group

ParentGroupId	ChildGroupId
3	1
3	2

GroupId	EmployeeId
1	3
2	1
2	2
2	3
3	1
3	2
3	3



Added 1 table; 6 rows; 2 new columns

+3 documents

Nested Groups

Graph vs Relational

GroupId	GroupName
1	Engineering
2	Sales


```
SELECT Groups.GroupId,
       Groups.GroupName
  FROM Groups
 INNER JOIN Group_Group
       ON Group_Group.ChildGroupId = Groups.GroupId
 WHERE Group_Group.ParentGroupId =
       (SELECT GroupId
              FROM Groups
             WHERE GroupName = 'Product Group')
```

```
g.V('Product Group’)
.outE('contains_subgroup’)
.inV()
```

Management

Graph Databases vs Relational Databases

Additional Hierarchies

Graph vs Relational

```
-- Create the Employee/Employee join table
CREATE TABLE Employee_Employee
(
  ParentEmployeeId INT,
  ChildEmployeeId  INT,
  CONSTRAINT pkcEmployeeEmployee PRIMARY KEY CLUSTERED (ParentEmployeeId, ChildEmployeeId),
  CONSTRAINT fkEmployeeEmployee_Employee_Parent FOREIGN KEY (ParentEmployeeId) REFERENCES Employees(EmployeeId),
  CONSTRAINT fkEmployeeEmployee_Employee_Child  FOREIGN KEY (ChildEmployeeId)  REFERENCES Employees(EmployeeId)
)
```

Additional Hierarchies

Graph vs Relational

```
-- Make Ada the boss
INSERT INTO Employee_Employee (ParentEmployeeId, ChildEmployeeId)
SELECT (SELECT EmployeeId FROM Employees WHERE EmployeeName = 'Ada C. Lavigne'),
       EmployeeId
  FROM Employees
 WHERE EmployeeId IN (SELECT EmployeeId
                        FROM Employee_Group
                       WHERE Employee_Group.GroupId = (SELECT GroupId
                                                         FROM Groups
                                                        WHERE GroupName = 'Sales'))
```

Additional Hierarchies

Graph vs Relational

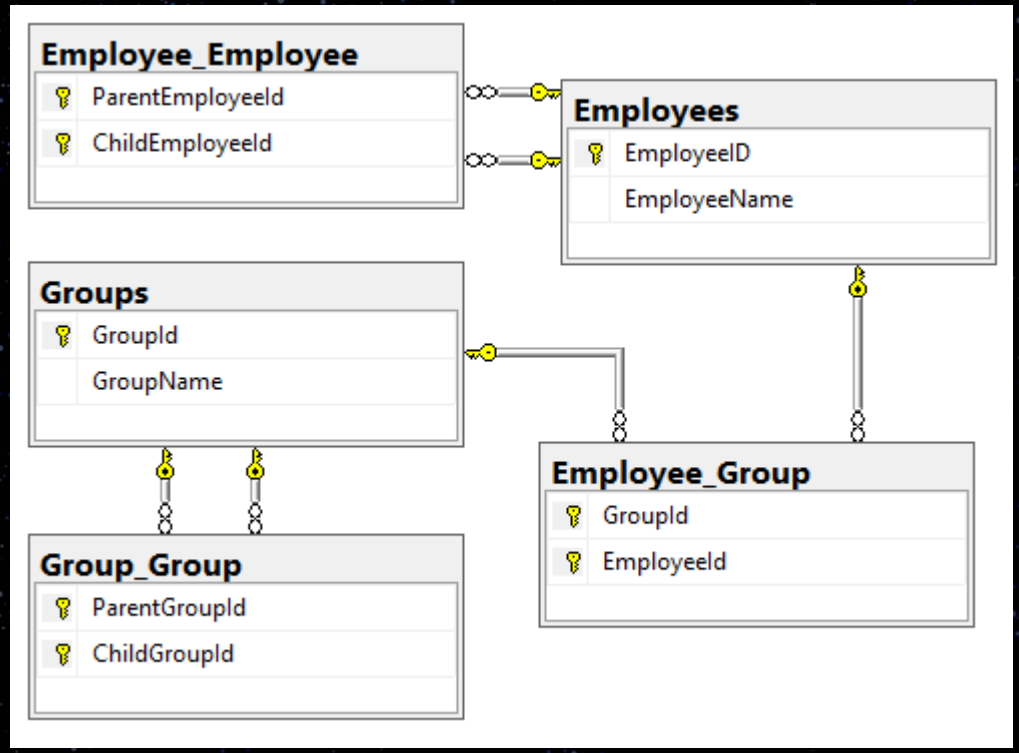
EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales
3	Product Group

ParentGroupId	ChildGroupId
3	1
3	2

GroupId	EmployeeId
1	3
2	1
2	2
2	3
3	1
3	2
3	3

ParentEmployeeId	ChildEmployeeId
3	1
3	2
3	3



Additional Hierarchies

Graph vs Relational

```
// Add relationships
g.V('Ada C. Lavigne').addE('has_report').to(g.V('Willis B. Hawkins'))
g.V('Ada C. Lavigne').addE('has_report').to(g.V('Neil S. Vega'))
```

Additional Hierarchies

Graph vs Relational

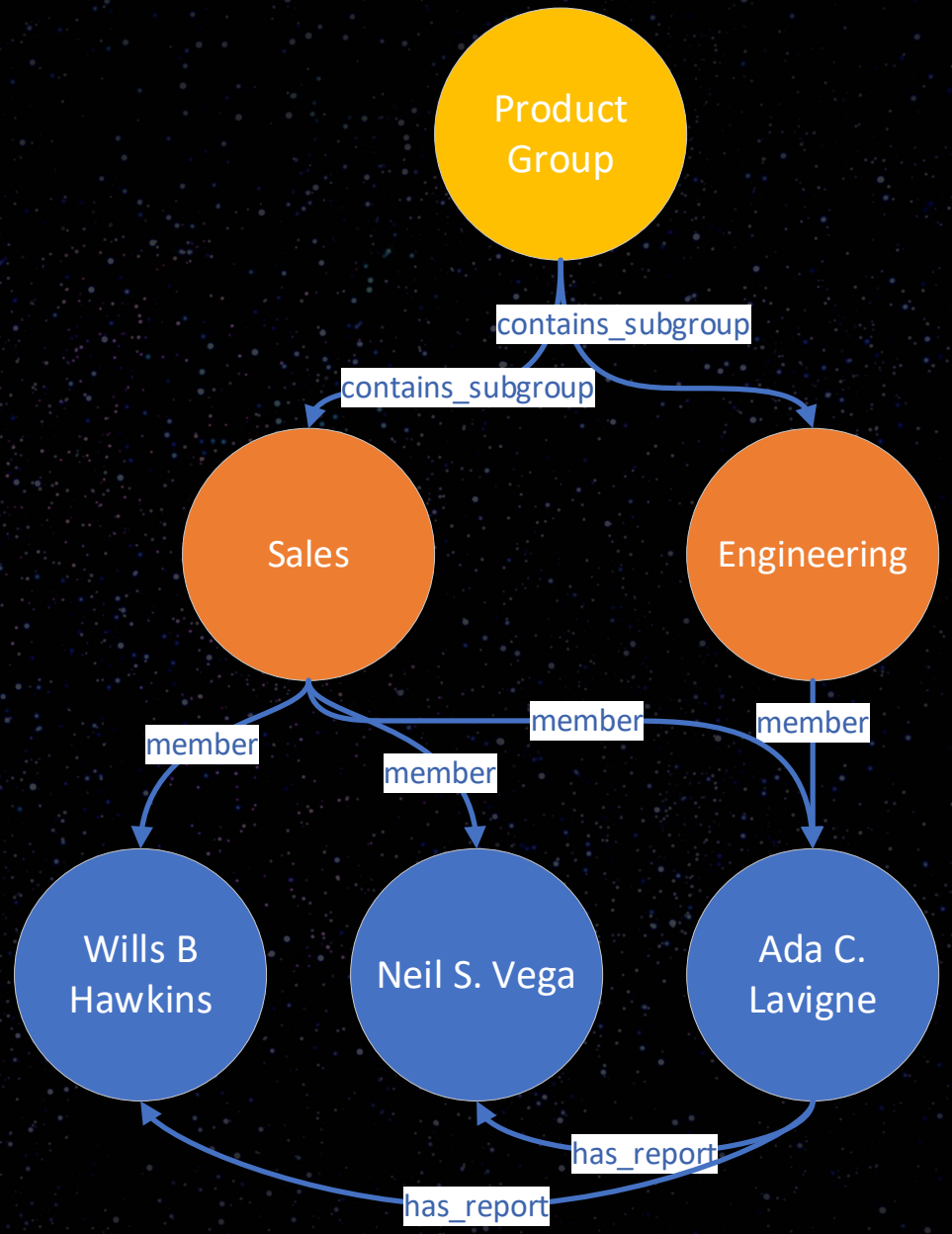
EmployeeId	EmployeeName
1	Willis B. Hawkins
2	Neil S. Vega
3	Ada C. Lavigne

GroupId	GroupName
1	Engineering
2	Sales
3	Product Group

ParentGroupId	ChildGroupId
3	1
3	2

GroupId	EmployeeId
1	3
2	1
2	2
2	3
3	1
3	2
3	3

ParentEmployeeId	ChildEmployeeId
3	1
3	2
3	3



Added 1 table; 2 rows; 2 new columns

+2 documents

Additional Hierarchies

Graph vs Relational

EmployeeName
Ada C. Lavigne

```
SELECT DISTINCT EmployeeName
  FROM Employees
 INNER JOIN Employee_Group
    ON Employee_Group_EmployeeId
       = Employes.EmployeeId
 INNER JOIN Employee_Employee
    ON Employee_Employee.ParentEmployeeId
       = Employees.EmployeeId
 WHERE Employee_Group.GroupId =
       (SELECT GroupId
             FROM Groups
            WHERE GroupName = 'Engineering')
```

Ada C. Lavigne

```
g.V('Engineering')
.outE('member')
.inV()
.outE('has_report')
.values('id')
```

Challenges of Relational Databases

- Schema Management
- Table Alternations
- Costly Writes Against Multiple Tables
- Multiple JOIN Operations
- Complex Read Queries


What is
Gremlin
Graphing Your Way Through the Cosmos
TALELEARNCODE
CHADGREEN

What is TinkerPop

Open source, vendor-agnostic, graph computing framework

What is TinkerPop

Apace2 license

ndor-agnostic, graph computing framework

What is TinkerPop

Apace2 license

Model domain as graph and analyze using Gremlin

TinkerPop-enabled systems integrate with one another

Open source, vendor-agnostic, graph computing framework

What is TinkerPop

Gremlin

What is TinkerPop

Gremlin

Gremlin Console

What is TinkerPop

- Gremlin
- Gremlin Console
- Gremlin Server

What is TinkerPop

- Gremlin
- Gremlin Console
- Gremlin Server
- TinkerGraph

What is TinkerPop

- Gremlin
- Gremlin Console
- Gremlin Server
- TinkerGraph

Programming Interfaces

What is TinkerPop

- Gremlin
- Gremlin Console
- Gremlin Server
- TinkerGraph

- Programming Interfaces
- Documentation

What is TinkerPop

- Gremlin
- Gremlin Console
- Gremlin Server
- TinkerGraph
- Programming Interfaces
- Documentation
- Useful Recipes

What is Gremlin

- Graph traversal language and virtual machine
- Supports OLTP and OLAP
- Supports imperative and declarative querying
- Supports user-defined domain specified languages


What is
Cosmos DB
Graphing Your Way Through the Cosmos
TALELEARNCODE
CHADGREEN

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Turnkey global distribution

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Turnkey global distribution

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Elastic scale out of storage & throughput

Turnkey global distribution

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Elastic scale out of storage & throughput

Turnkey global distribution

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Guaranteed low latency at the 99th percentile

Elastic scale out
of storage & throughput

Turnkey global
distribution

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Guaranteed low latency at the 99th percentile

Elastic scale out
of storage & throughput

Turnkey global
distribution

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Five well-defined consistency models

Guaranteed low latency at the 99th percentile

Elastic scale out of storage & throughput

Turnkey global distribution

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Comprehensive SLAs

Guaranteed low latency at the 99th percentile

Elastic scale out of storage & throughput

Five well-defined consistency models

Turnkey global distribution

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Comprehensive SLAs

Guaranteed low latency at the 99th percentile

Elastic scale out of storage & throughput

Five well-defined consistency models

Turnkey global distribution

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Battle Tested

XBOX LIVE

Office 365

skype™

Guaranteed low latency at the 99th percentile

Elastic scale out of storage & throughput

Five well-defined consistency models

Turnkey global distribution

Comprehensive SLAS

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Battle Tested

Coca-Cola · AVEVA Schneider Electric · Albertsons · Safeway · Johnson Controls · Siemens Healthineers · BMT · Liberty Mutual Insurance · jet · Rolls-Royce · ExxonMobil · Cincinnati Children's

Guaranteed low latency at the 99th percentile

Elastic scale out of storage & throughput

Five well-defined consistency models

Turnkey global distribution

Comprehensive SLAS

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Ubiquitous Regional Presence

Turnkey globa
distribution

Comprehensive
SLAS

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

Secure by default and enterprise ready

Guaranteed low latency at the 99th percentile

Elastic scale out of storage & throughput

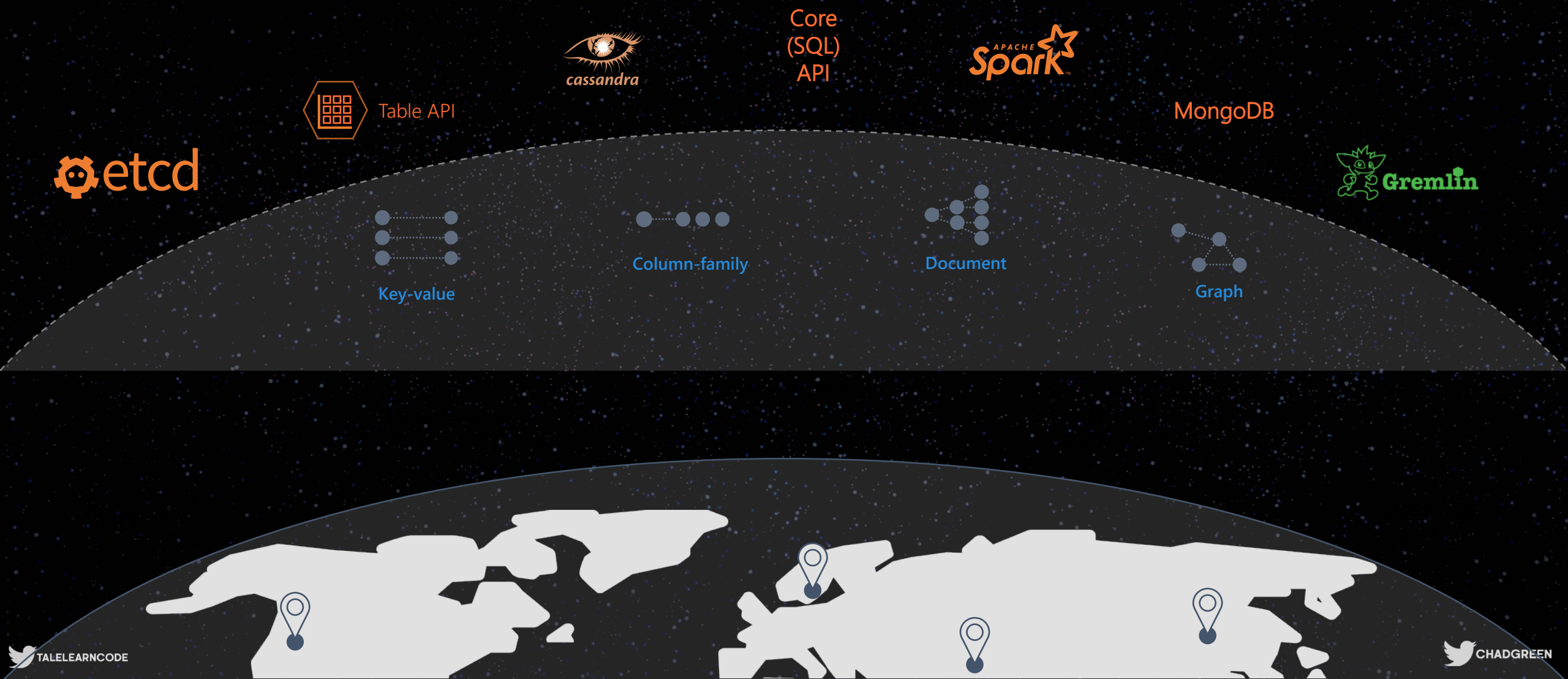
Five well-defined consistency models

Turnkey global distribution

Comprehensive SLAS

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service

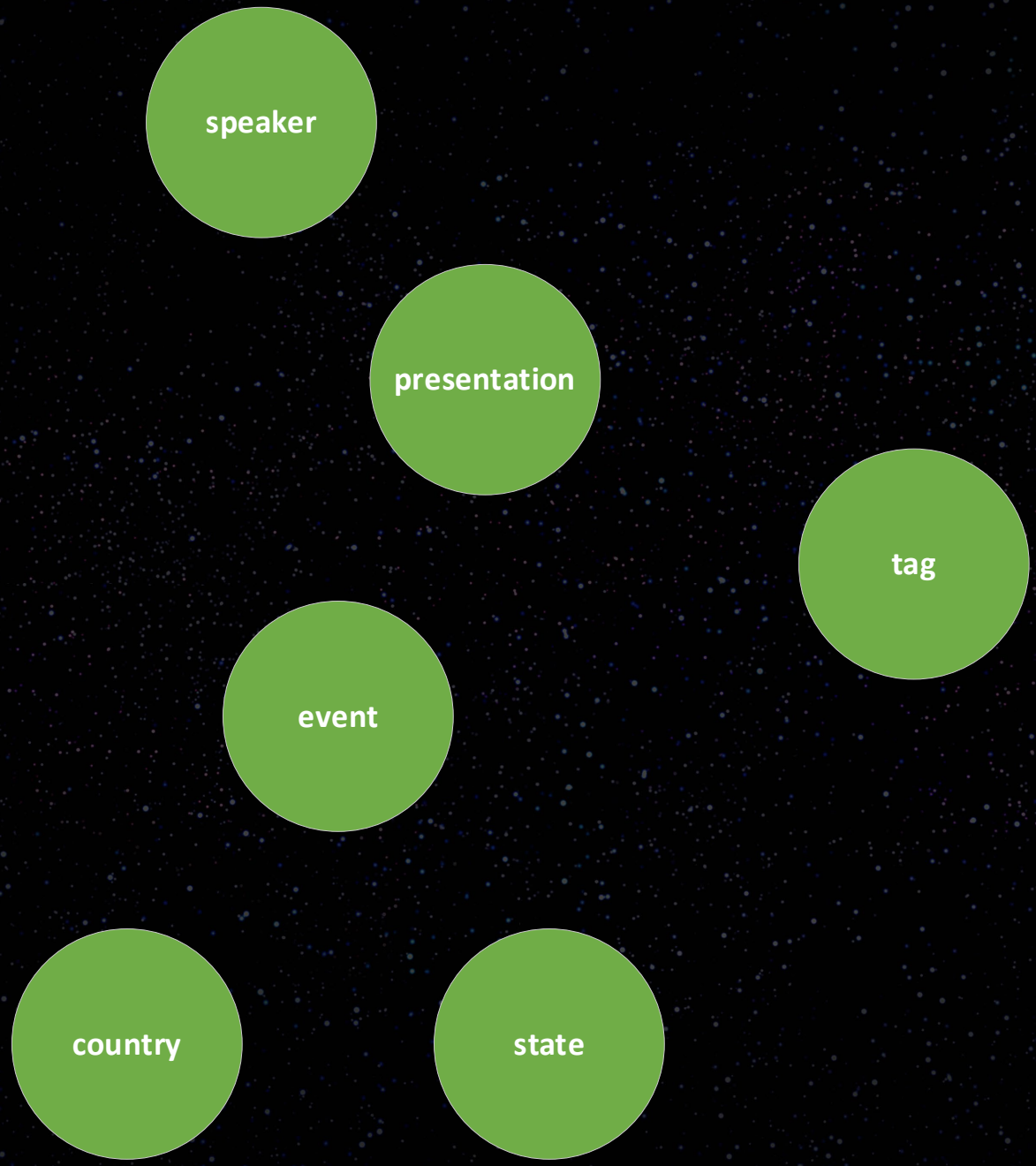



Exploring
Graph
Traversals
Graphing Your Way Through the Cosmos
TALELEARNCODE
CHADGREEN

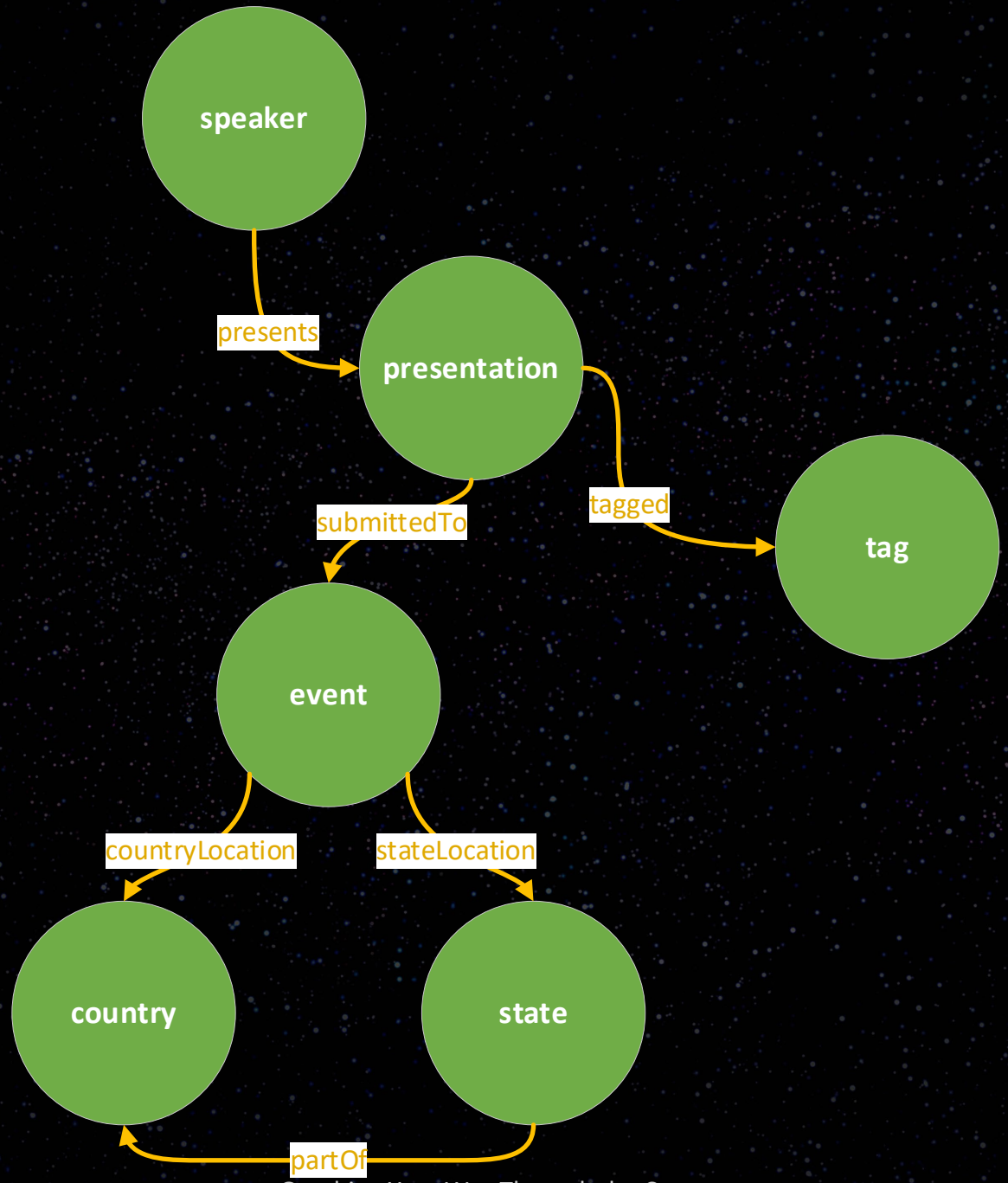
Requirements for Speaking Engagement Management

- Where has a presentation been submitted?
- What presentations are tagged with a particular tag?
- Where have presentations tagged with a particular tag been accepted?
- What events has a speaker been accepted at?
- Where were the events a speaker has been accepted to?

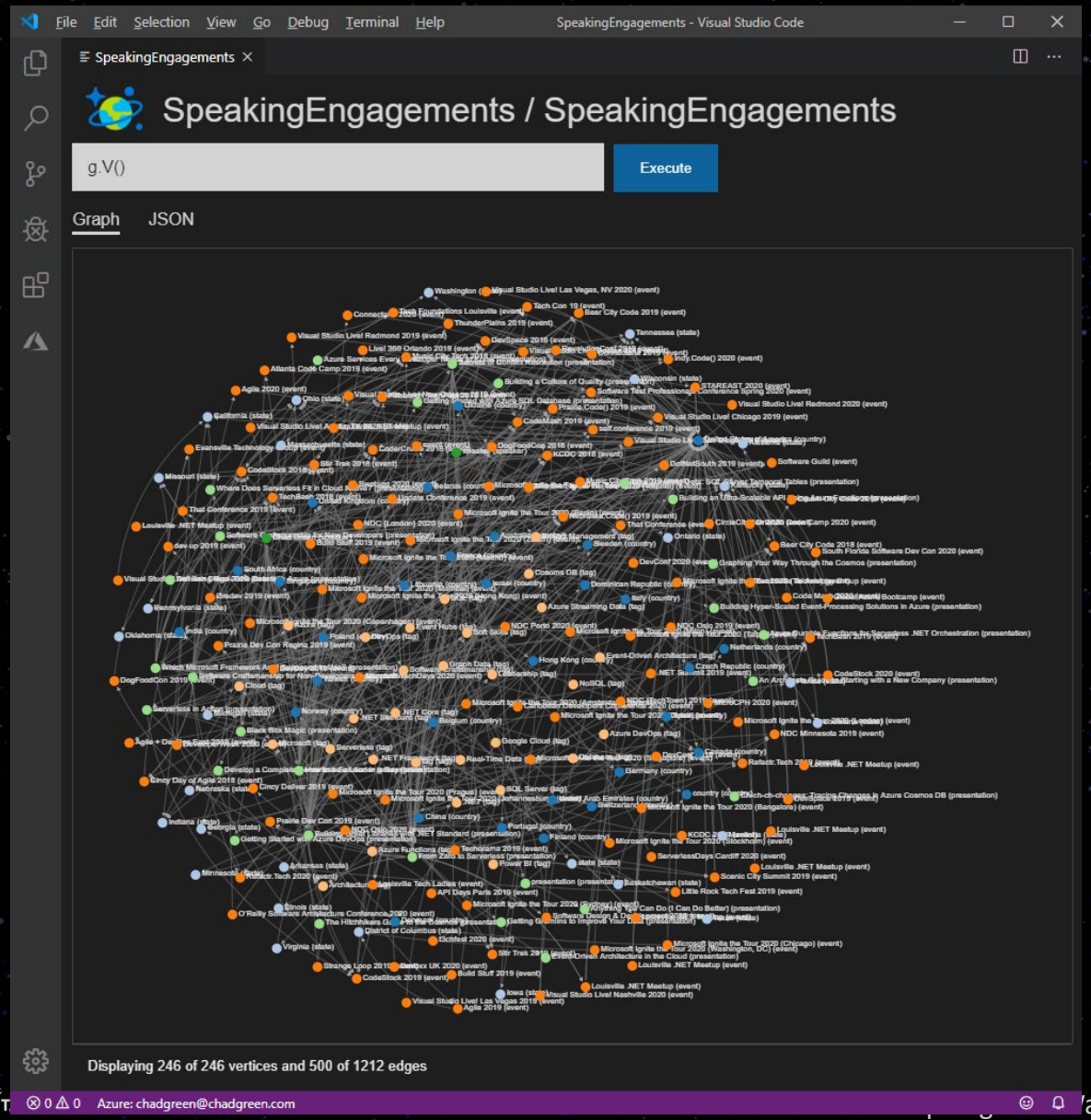
“Schema”



“Schema”

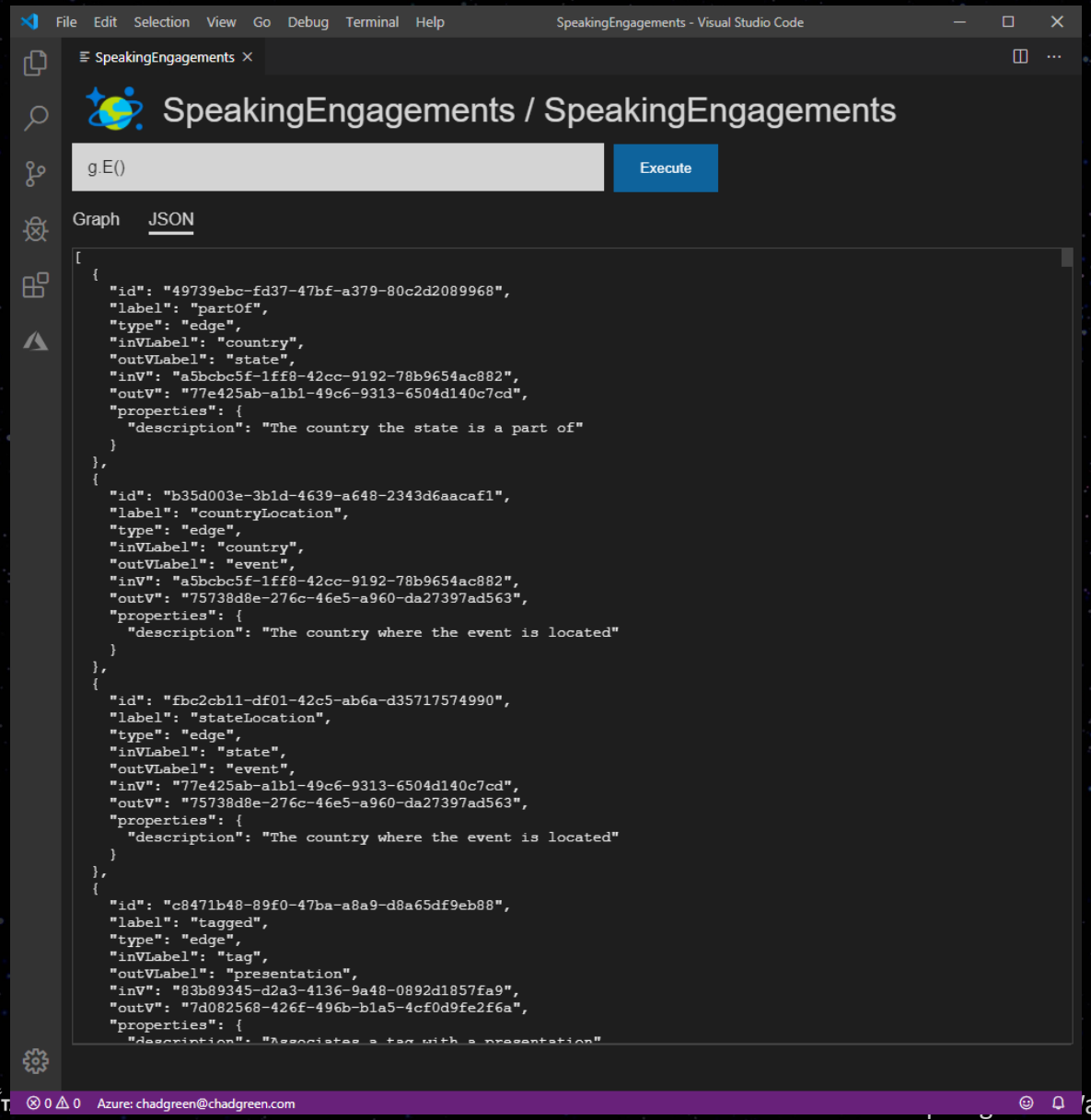

speaker
presents
presentation
tagged
tag
submittedTo
event
countryLocation
stateLocation
country
state
partOf

View the whole graph



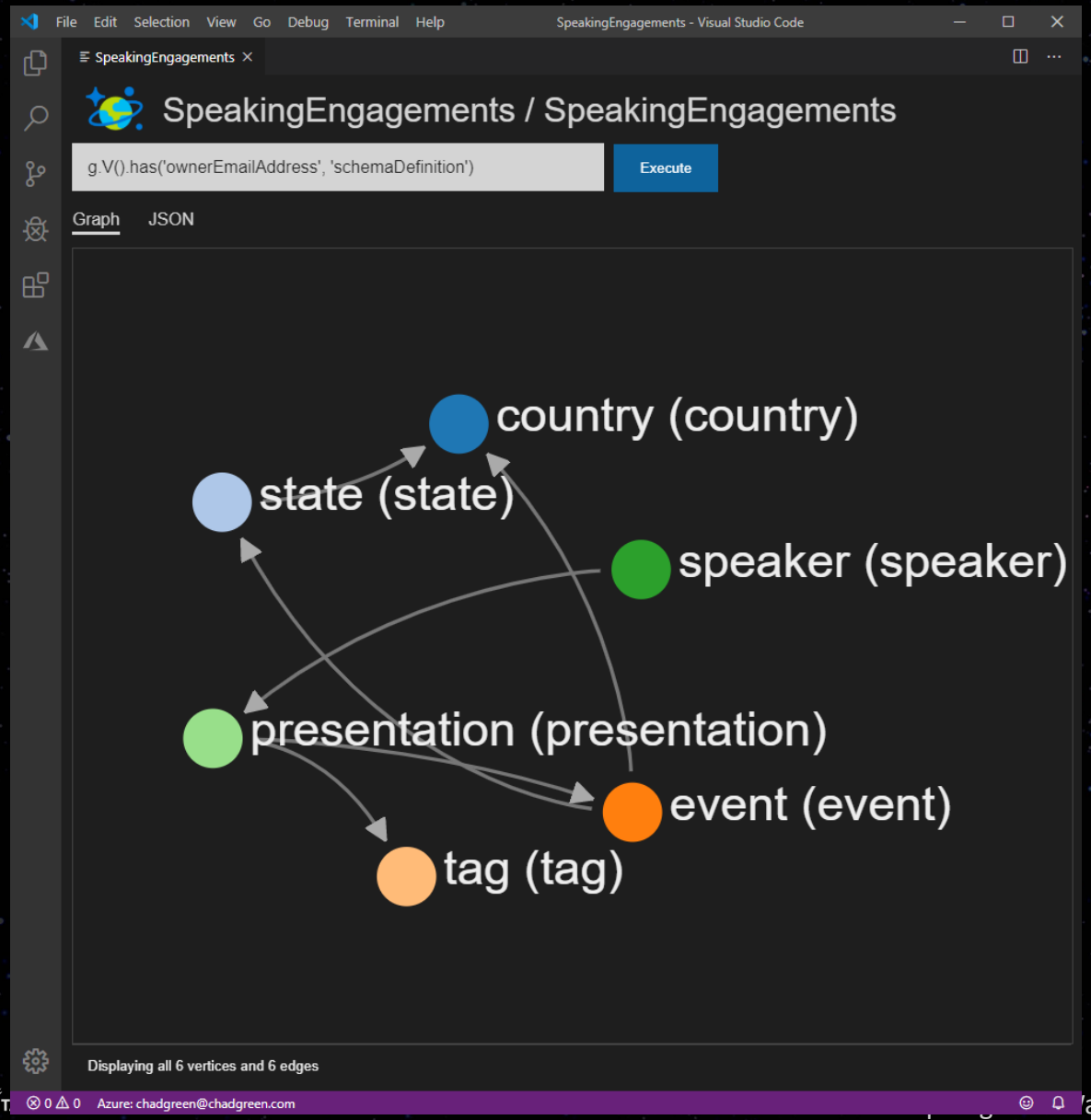
g.V()

View all of the edges



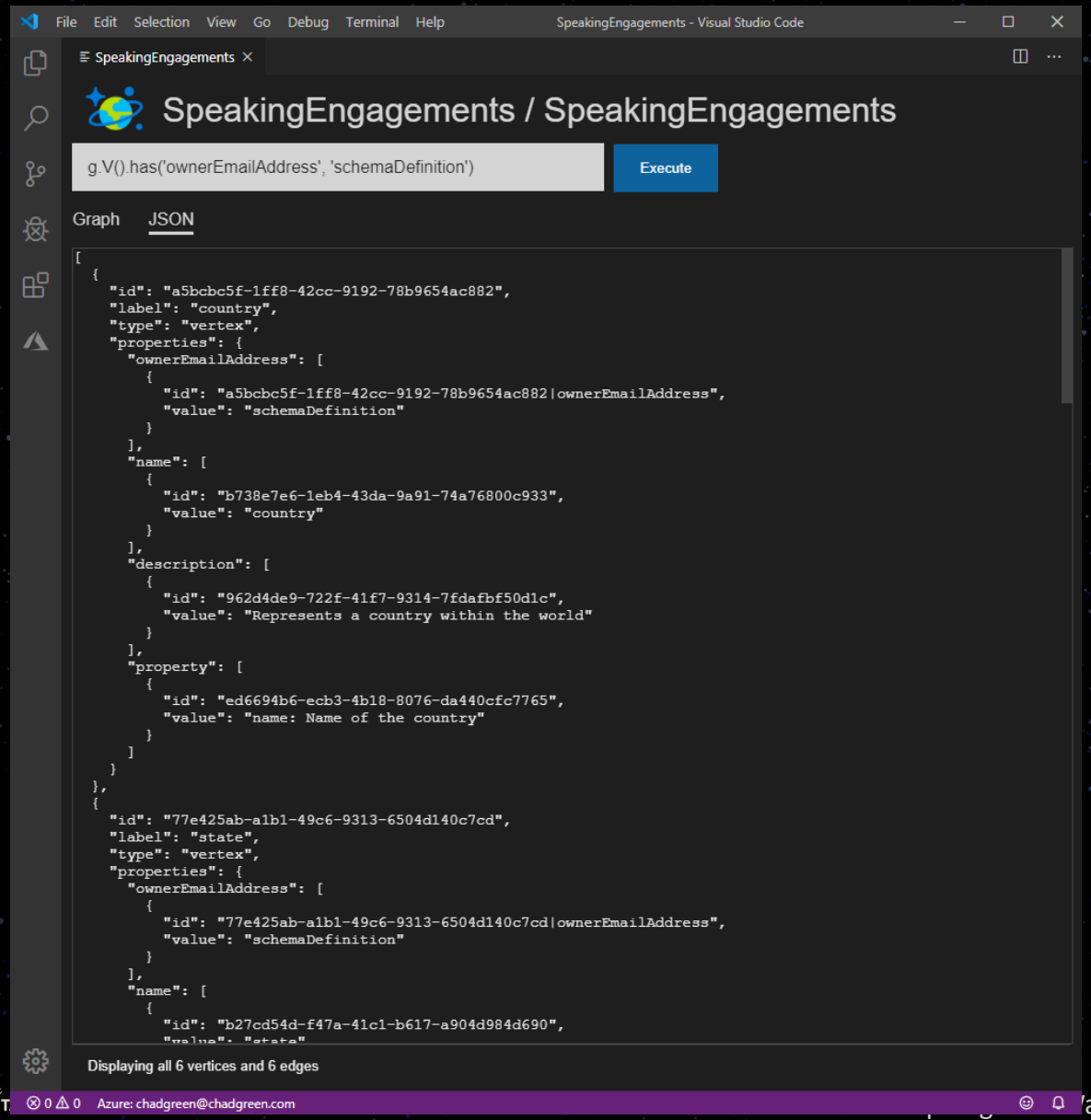
g.E()

View the schema definition



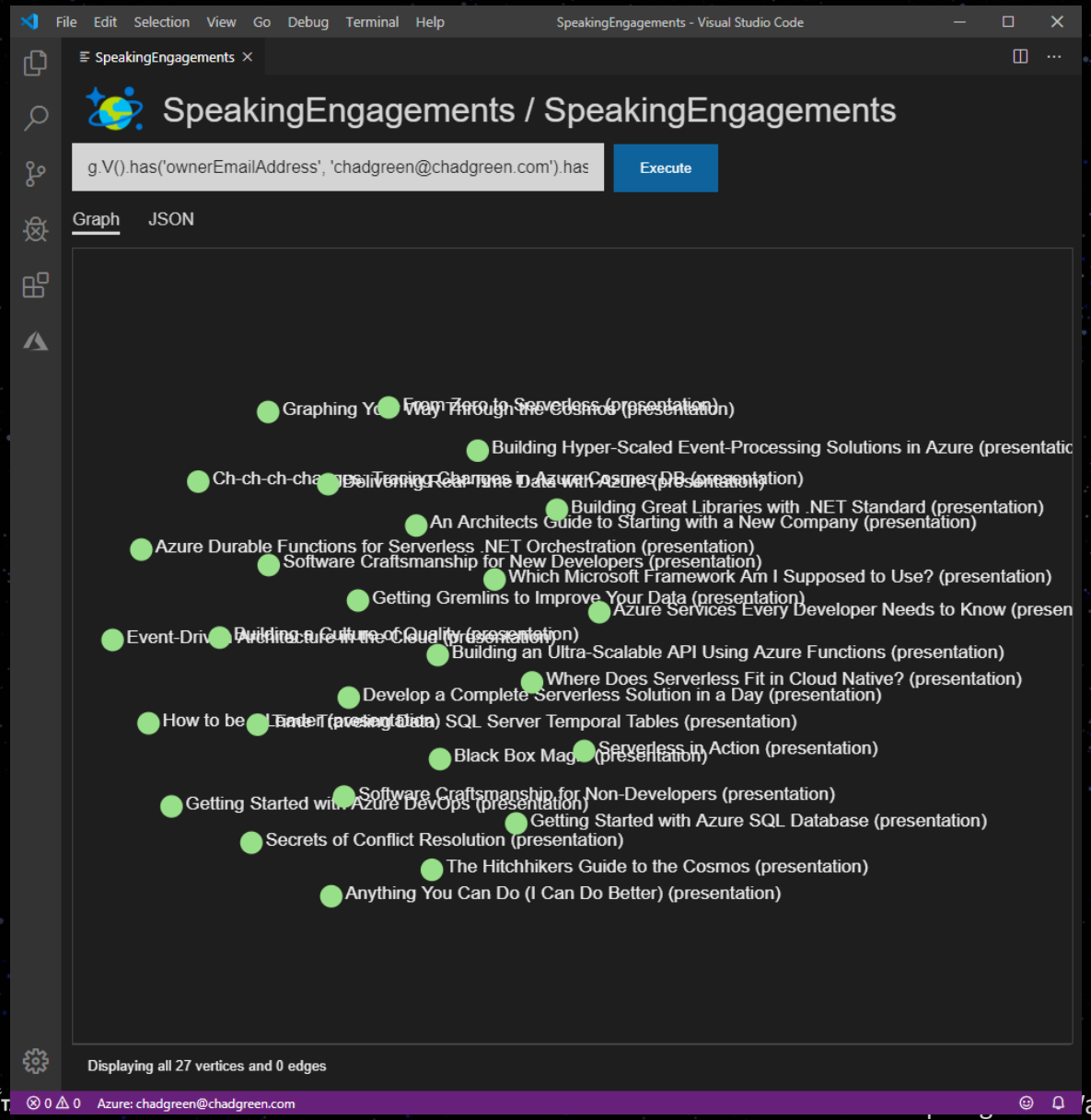

```
g.V()
.has('ownerEmailAddress',
'schemaDefinition')
```

View the schema definition



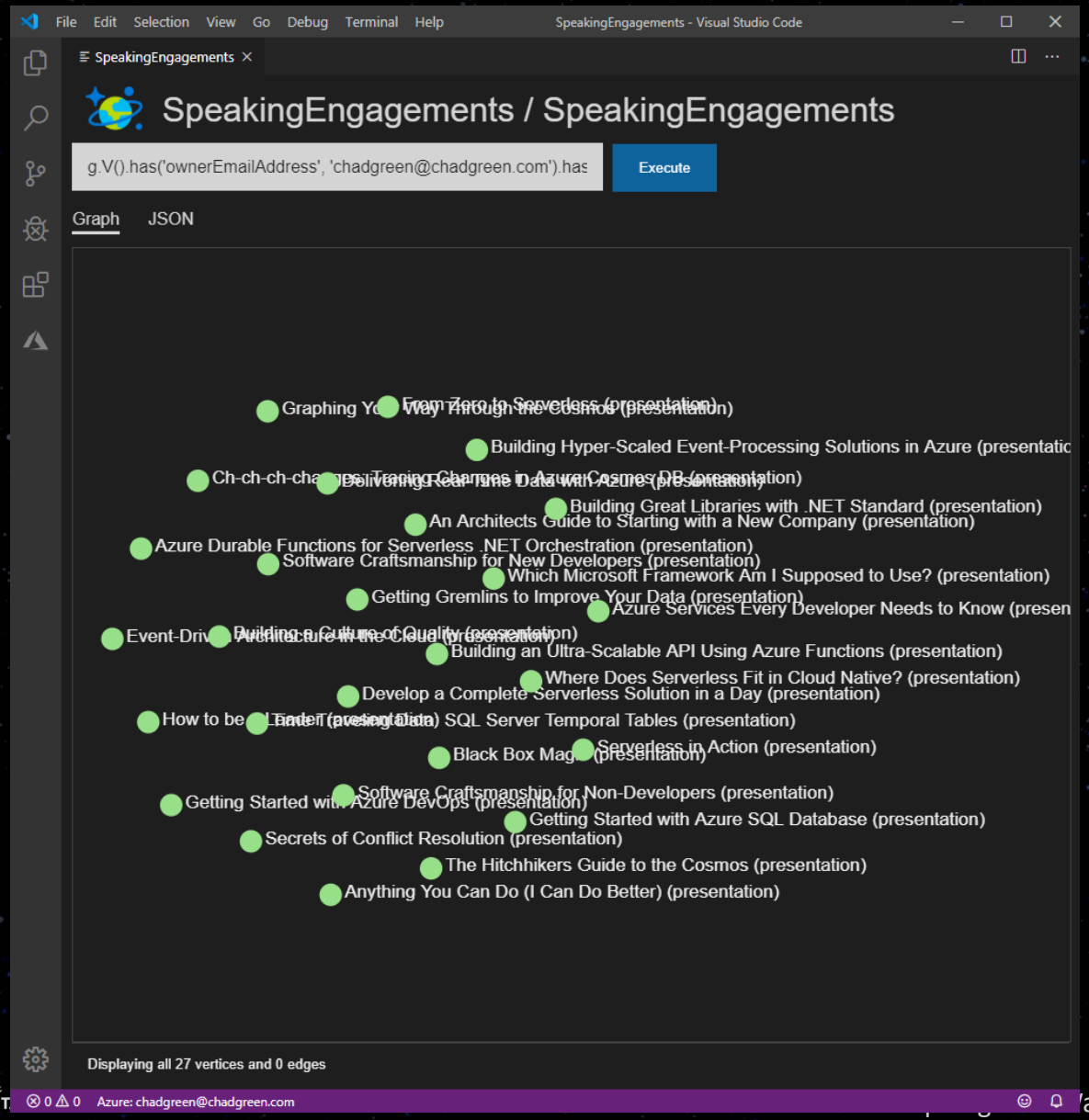
g.V()
.has('ownerEmailAddress',
'schemaDefinition')

What presentations are in my repertoire?



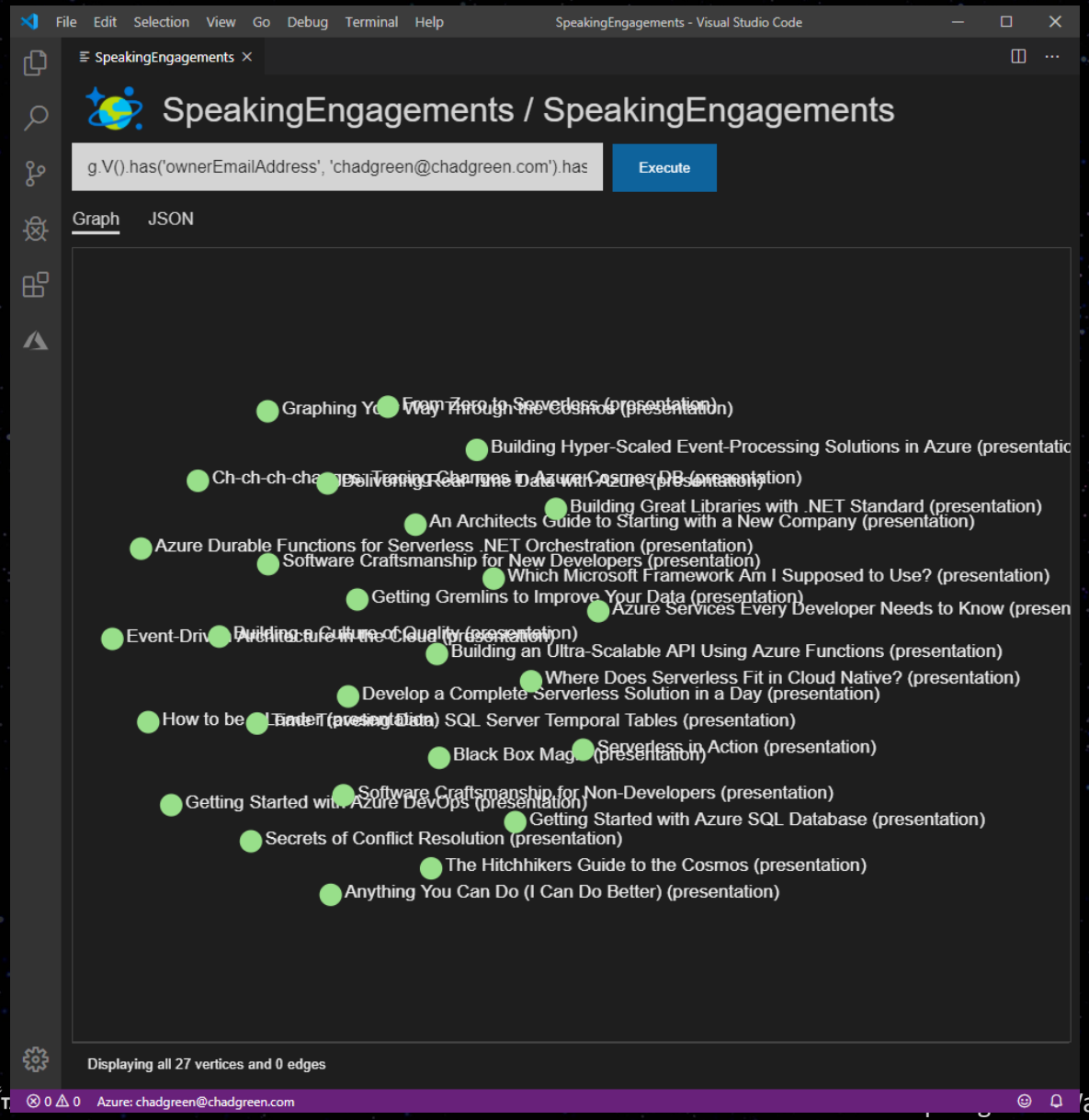

```
g.V()
.has('ownerEmailAddress',
'chadgreen@chadgreen.com')
.hasLabel('presentation')
```

What presentations are in my repertoire?



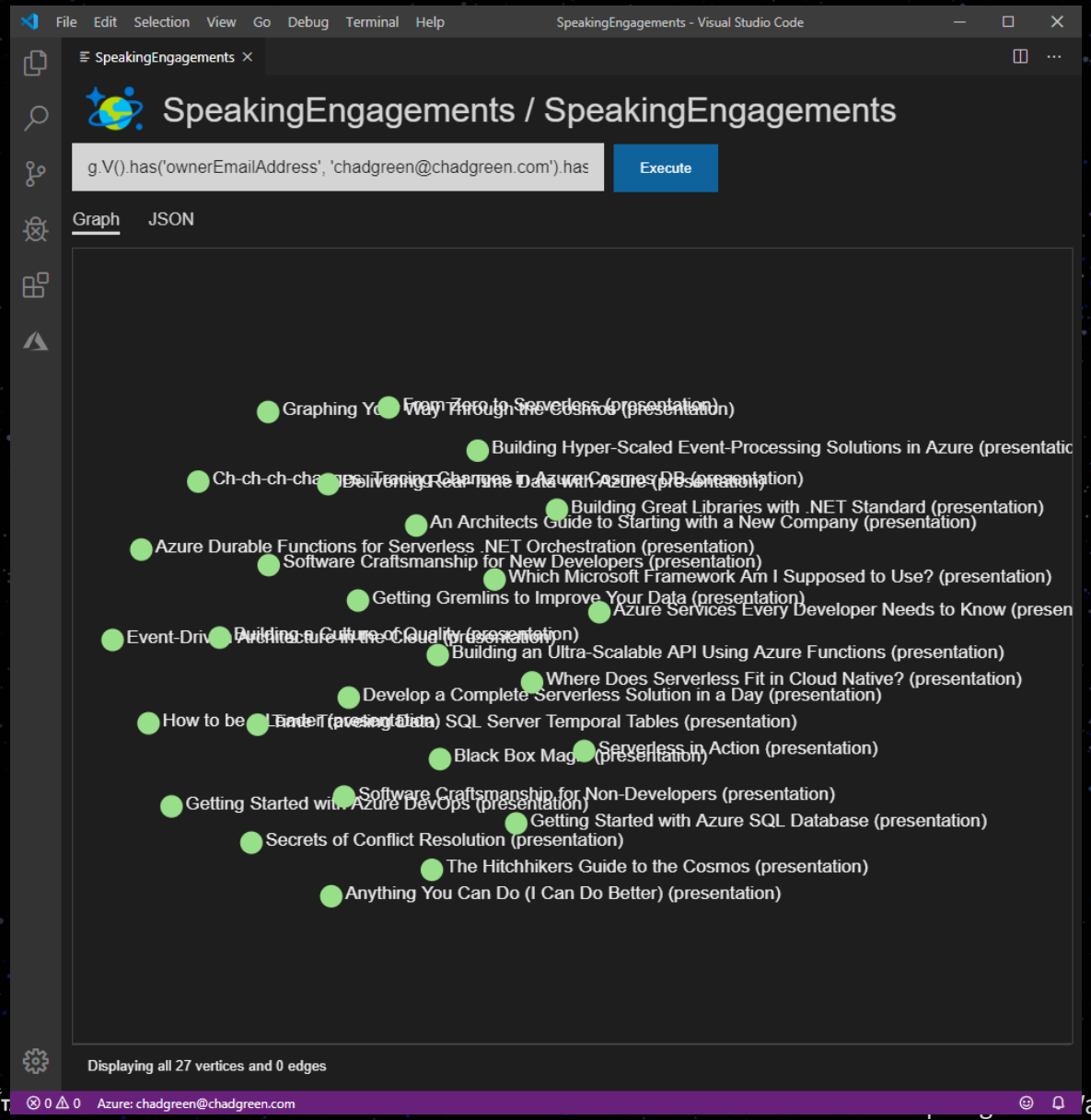

```
g.V()
.has('ownerEmailAddress',
'chadgreen@chadgreen.com')
.hasLabel('presentation')
```

What presentations are in my repertoire?



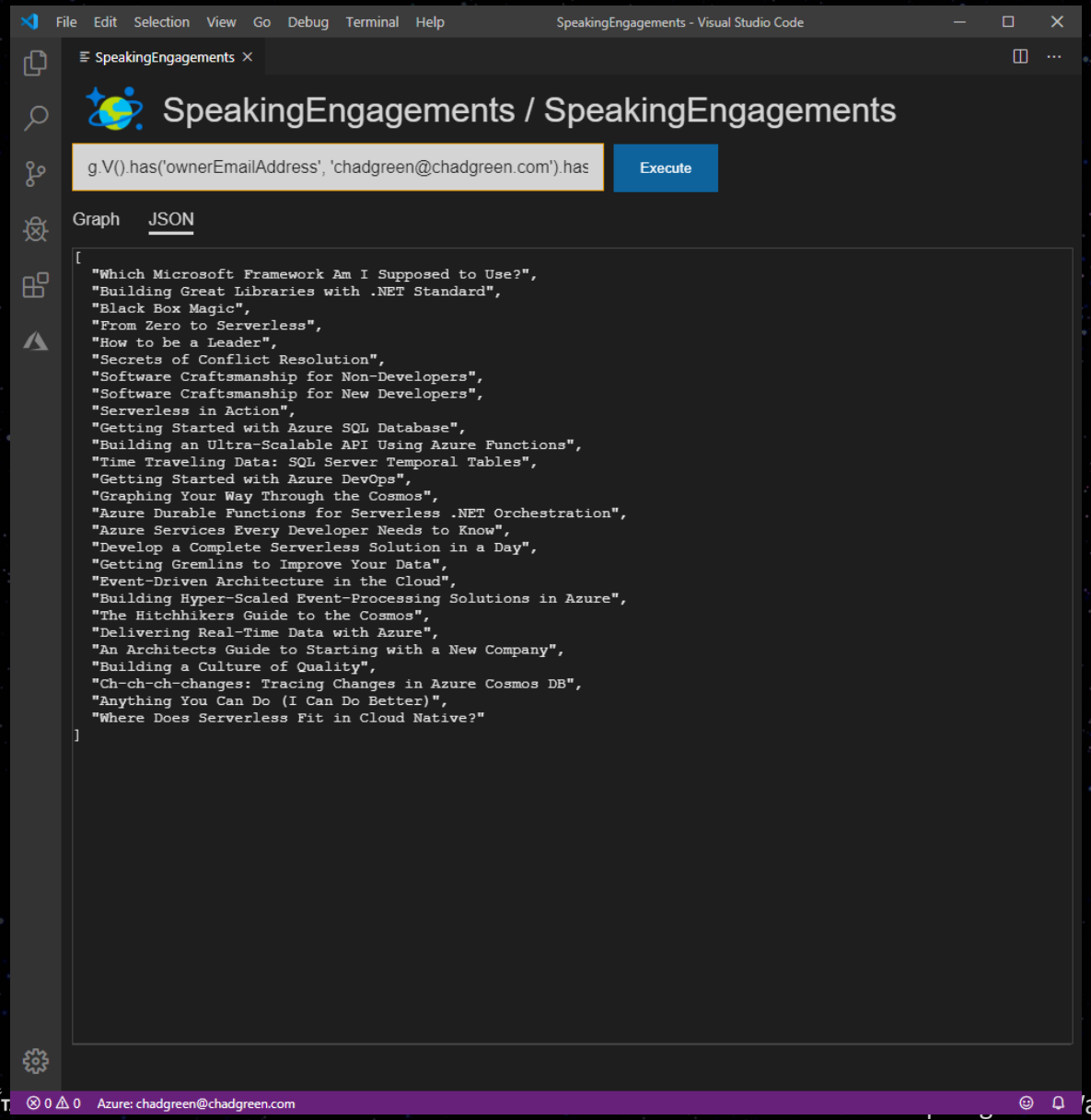

```
g.V()
.has('ownerEmailAddress',
'chadgreen@chadgreen.com')
.hasLabel('presentation')
```

What presentations are in my repertoire?



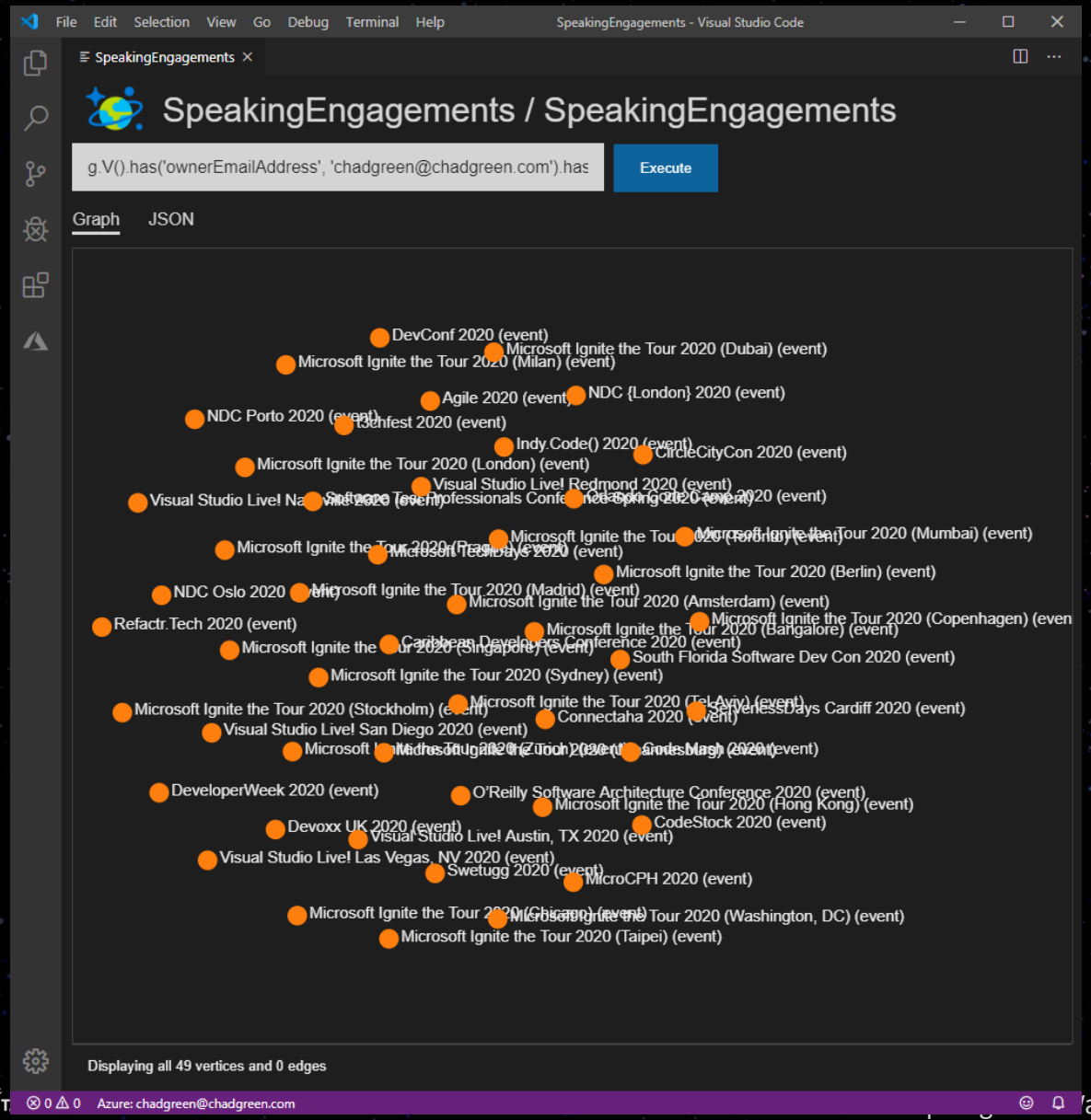

```
g.V()
.hasLabel('presentation')
```

What presentations are in my repertoire?



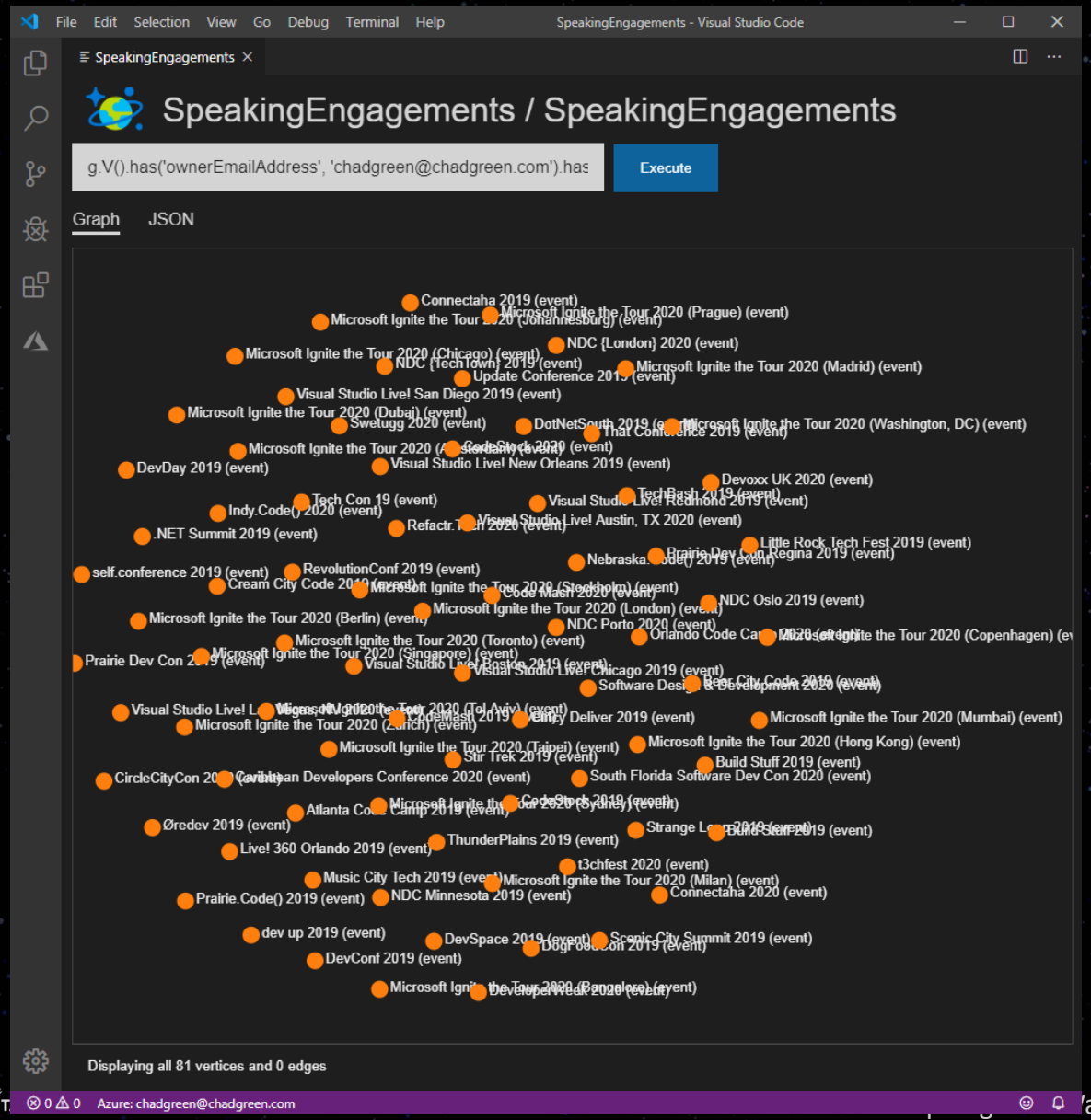

```
g.V()
.hasLabel('presentation')
.values('name')
```

What are the events that I have submitted to?



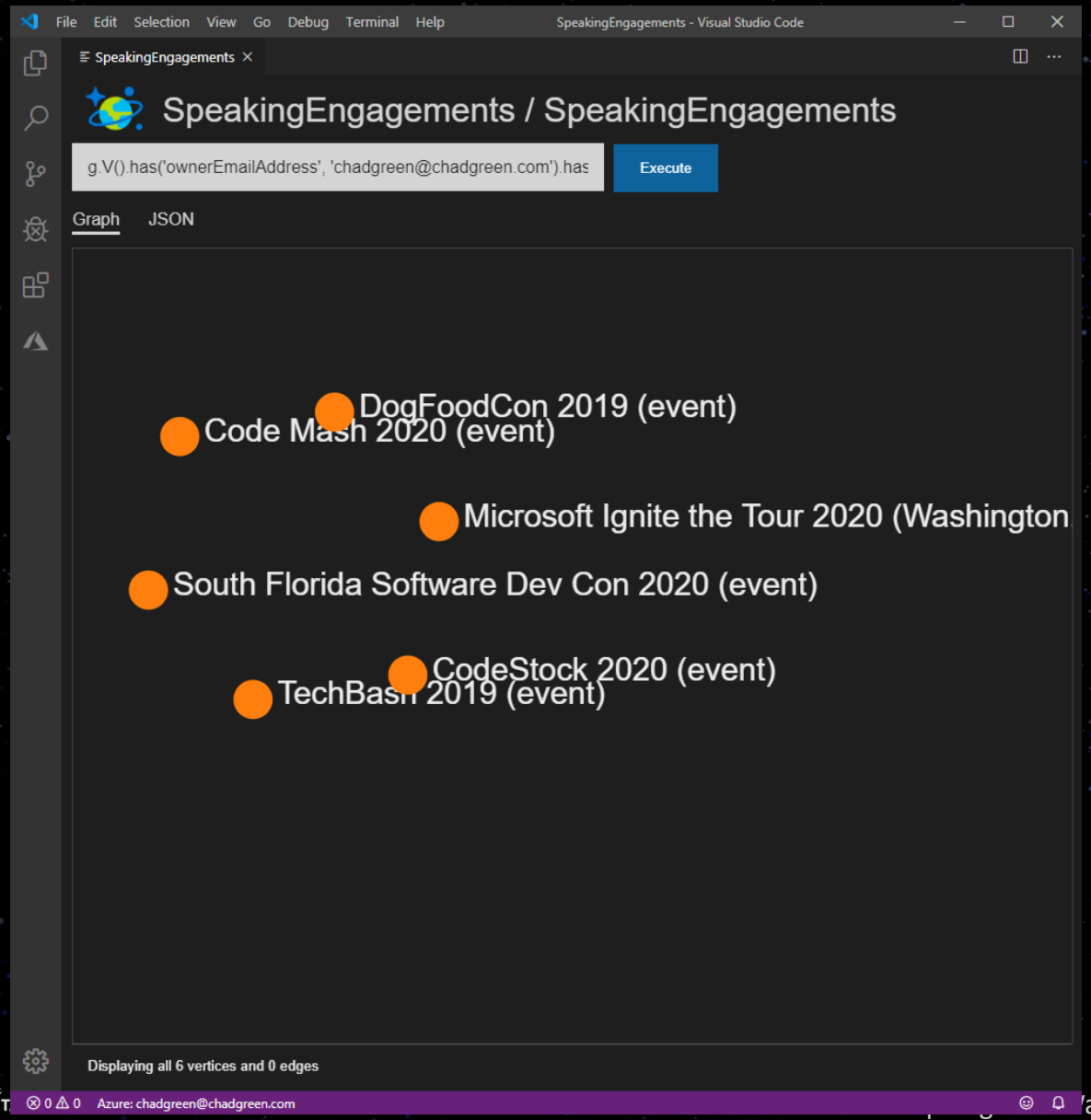

```
g.V()
.hasLabel('event')
.has('year', '2020')
```

Where has *Graphing Your Way Through the Cosmos* been submitted to?



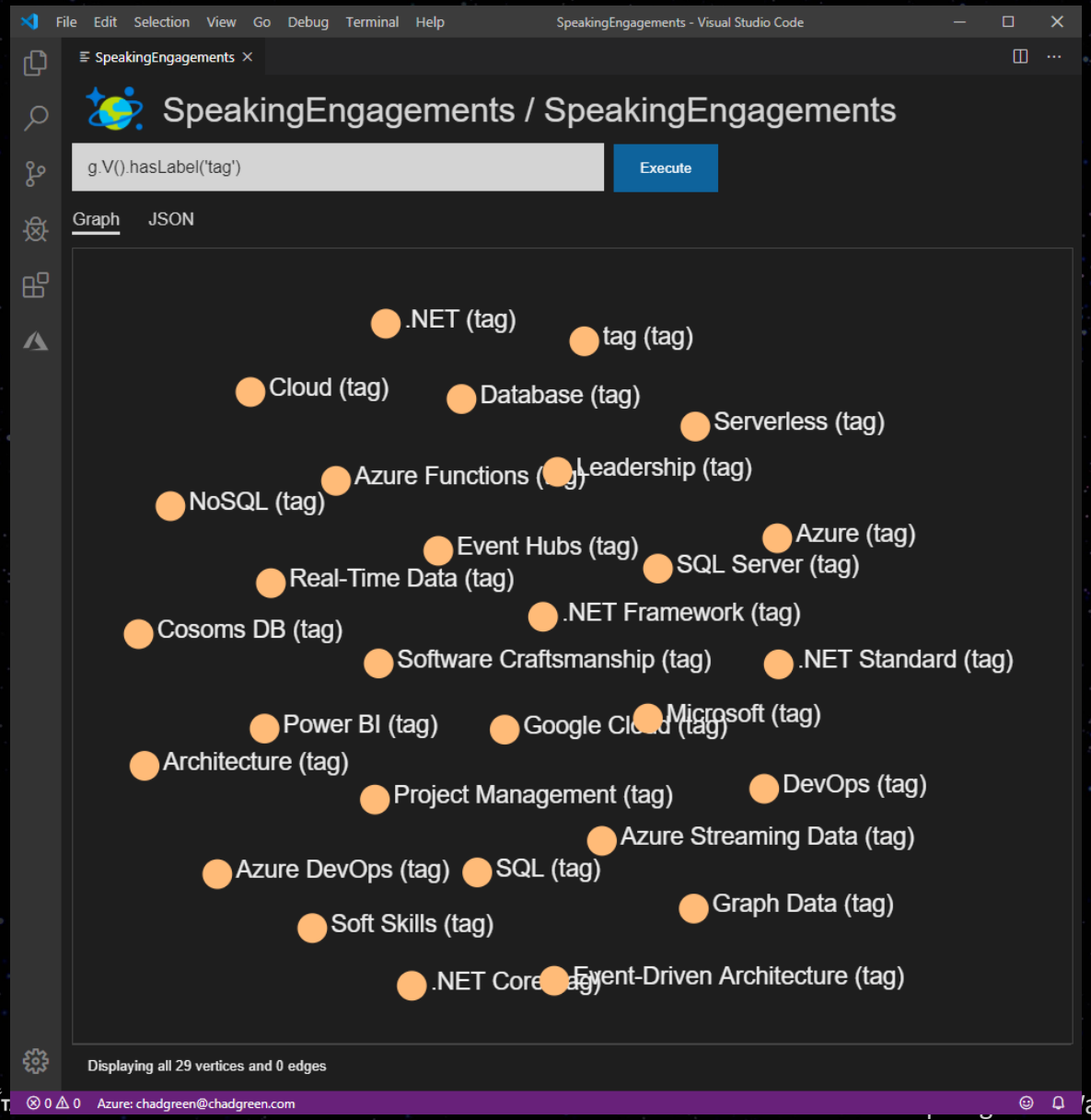

```
g.V()
.hasLabel('presentation')
.has('name', 'Graphing Your Way Through the Cosmos')
.outE('submittedTo')
.inV()
```

Where has *Graphing Your Way Through the Cosmos* been scheduled?



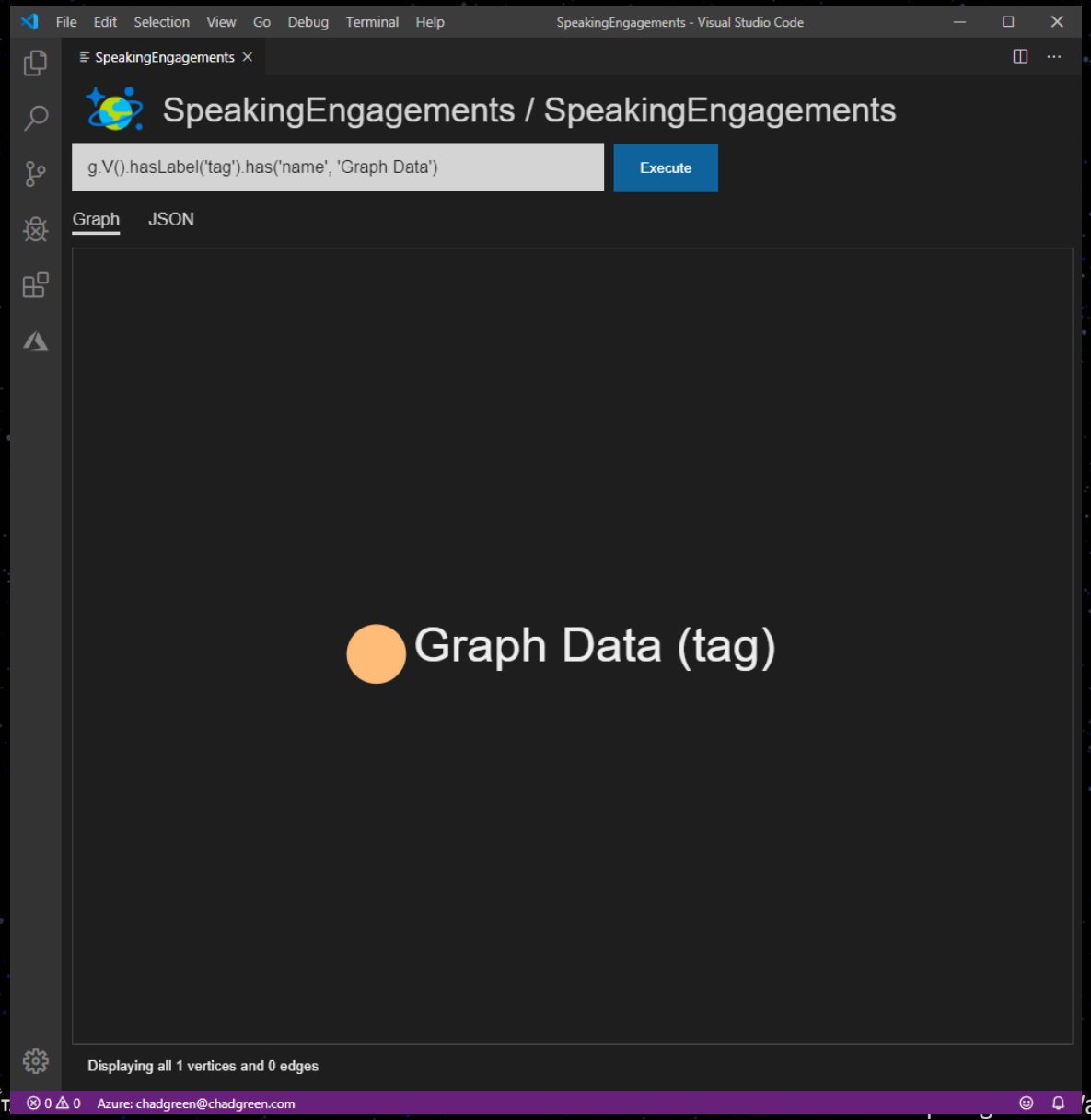

```
g.V()
.hasLabel('presentation')
.has('name', 'Graphing Your
Way Through the Cosmos')
.outE('submittedTo')
.has('status', 'Confirmed')
.inV()
```

View all of the tags



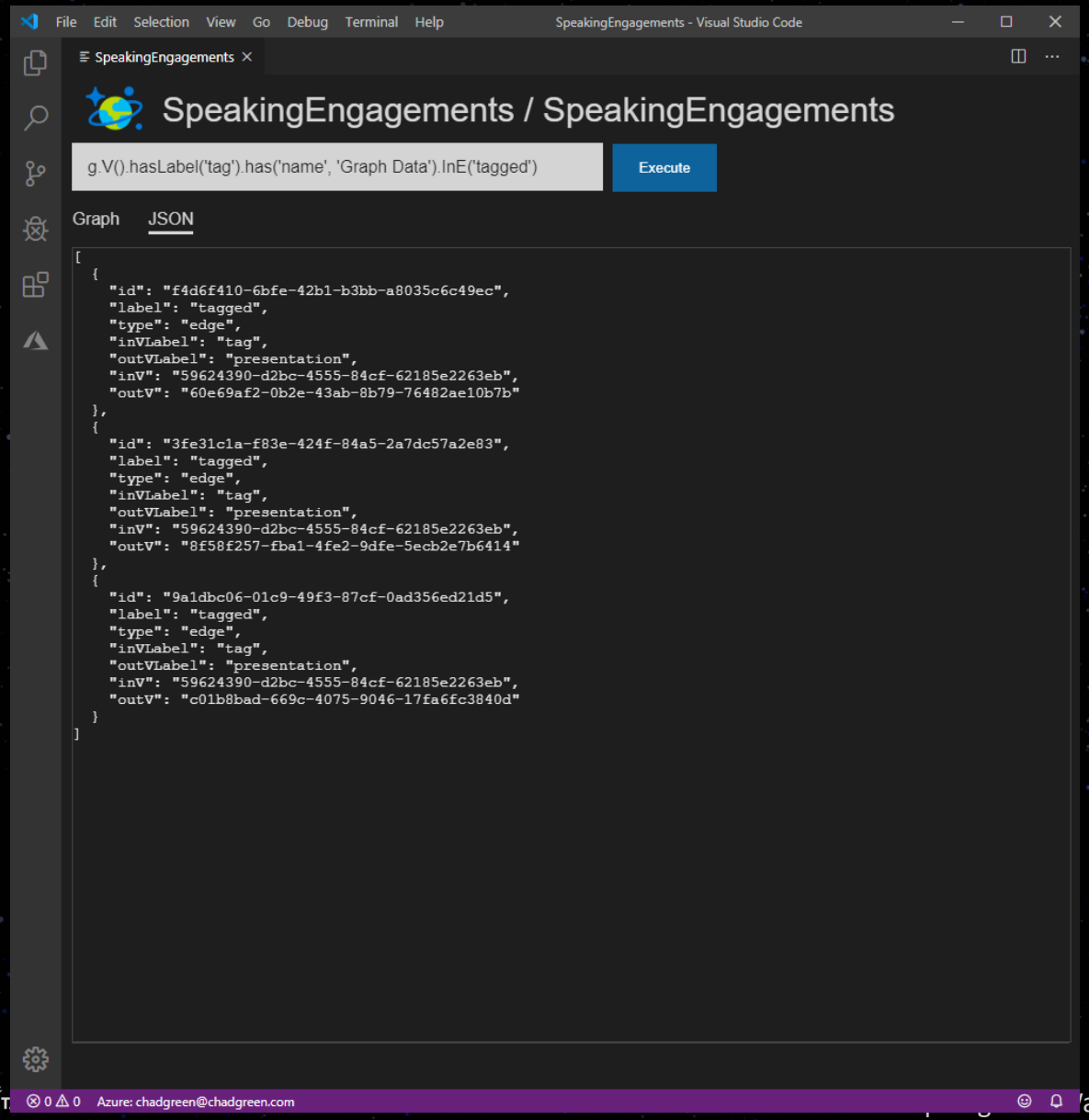

```
g.V()
.hasLabel('tag')
```

Focus on the *Graph Data* tag



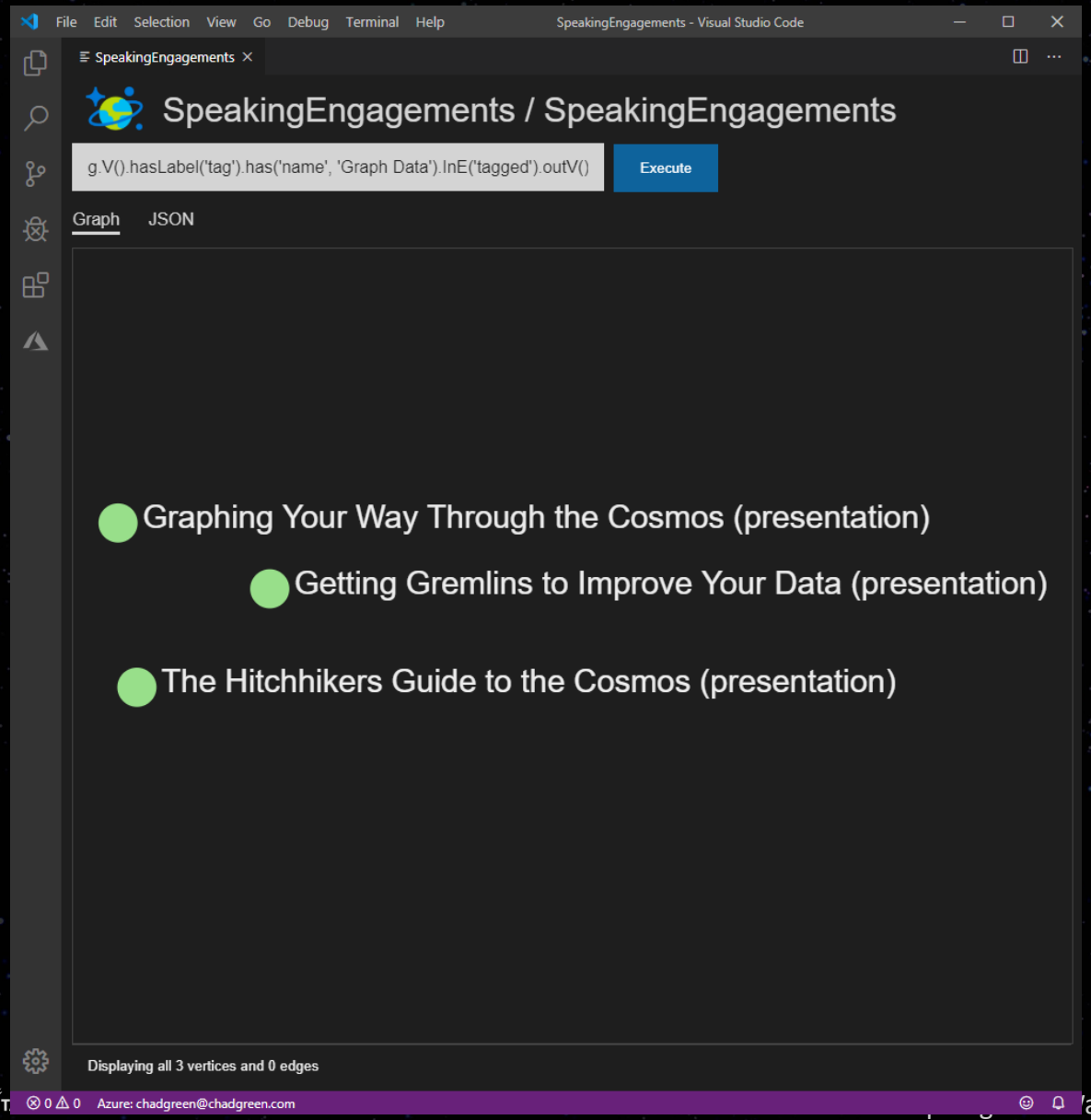

```
g.V()
.hasLabel('tag')
.has('name', 'Graph Data')
```

What presentations are tagged with *Graph Data*?



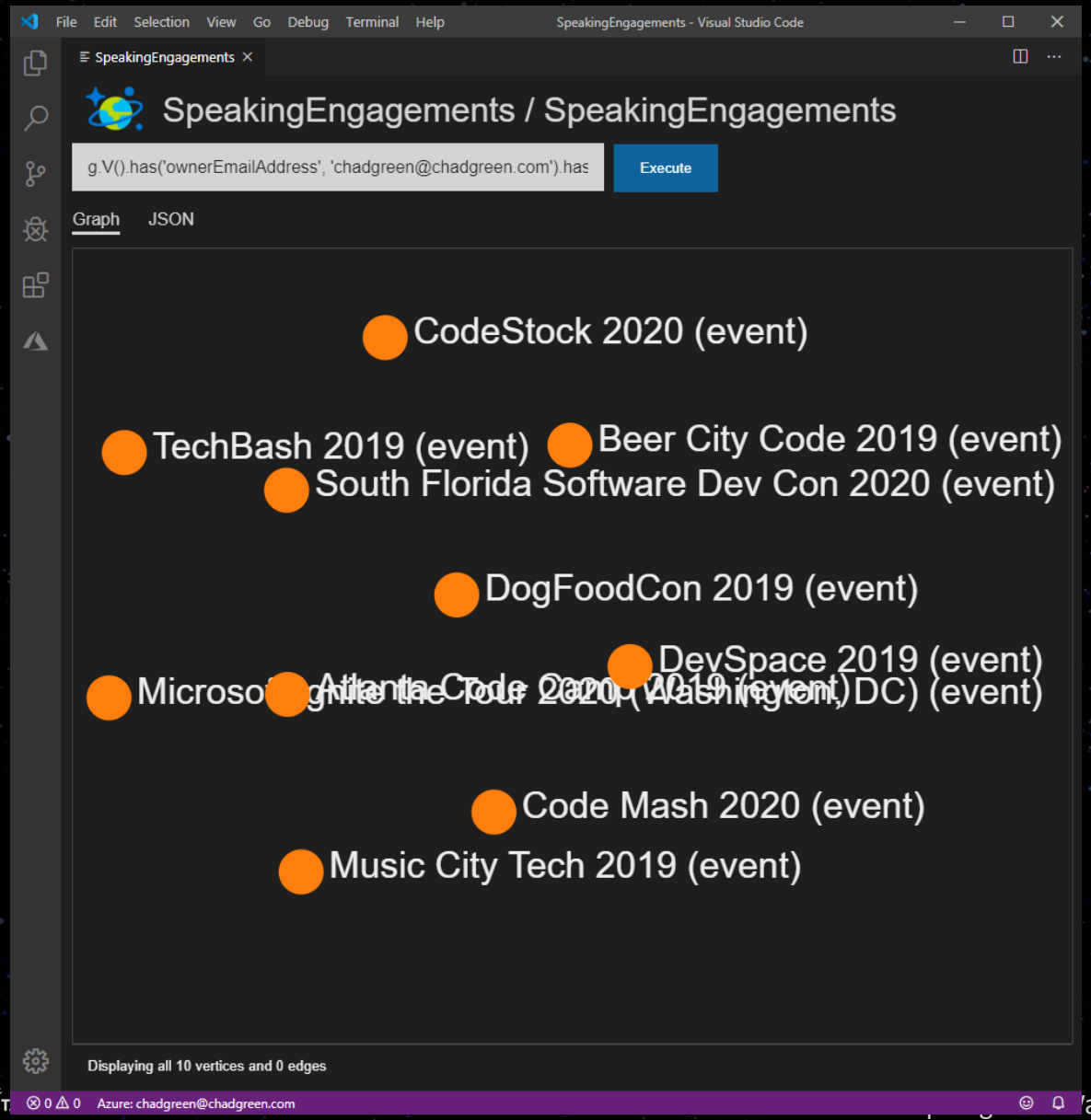

```
g.V()
.hasLabel('tag')
.has('name', 'Graph Data')
.inE('tagged')
```

What presentations are tagged with *Graph Data*?



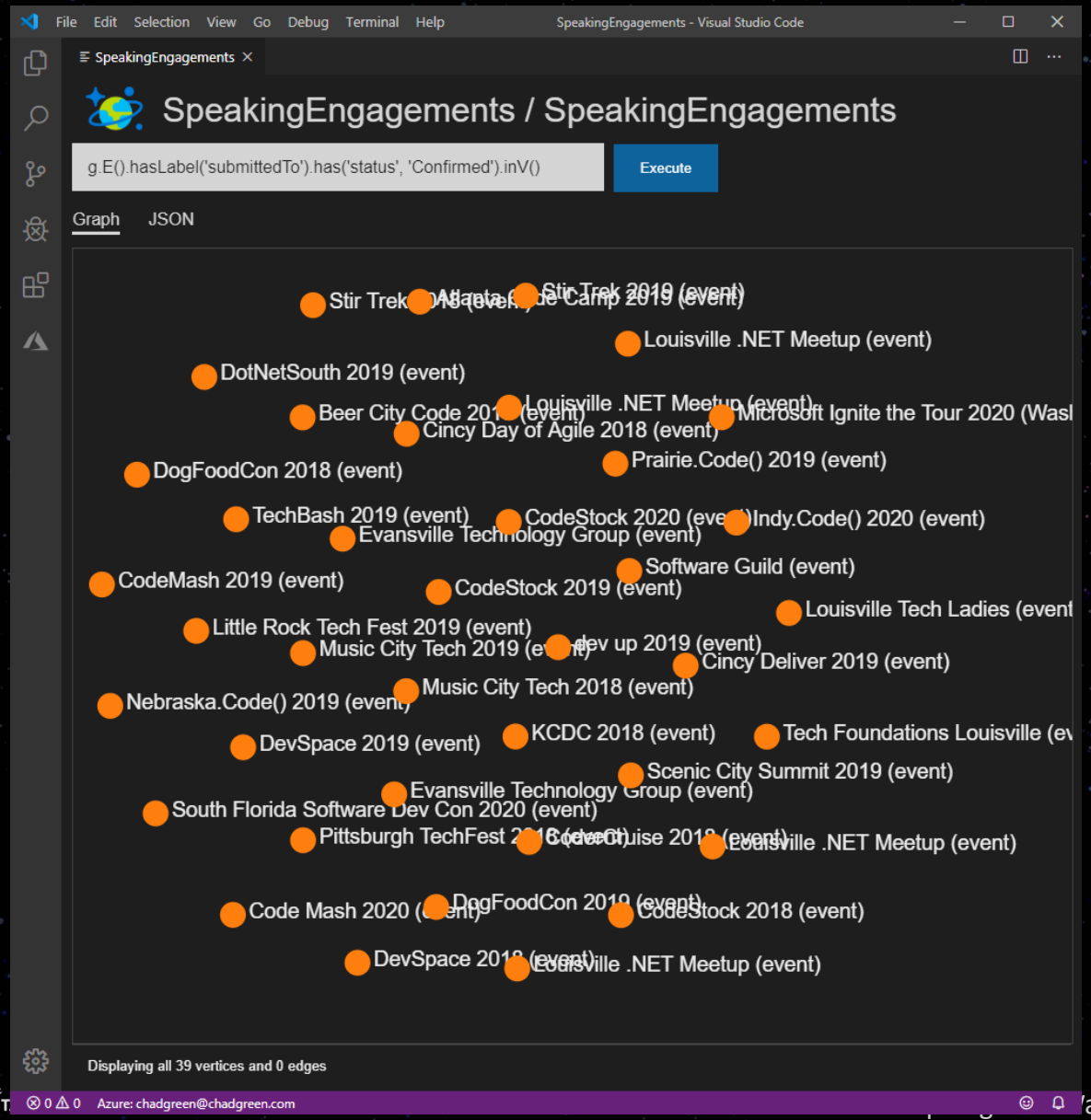

```
g.V()
.hasLabel('tag')
.has('name', 'Graph Data')
.inE('tagged')
.outV()
```

Where have presentations tagged *Graph Data* been scheduled?



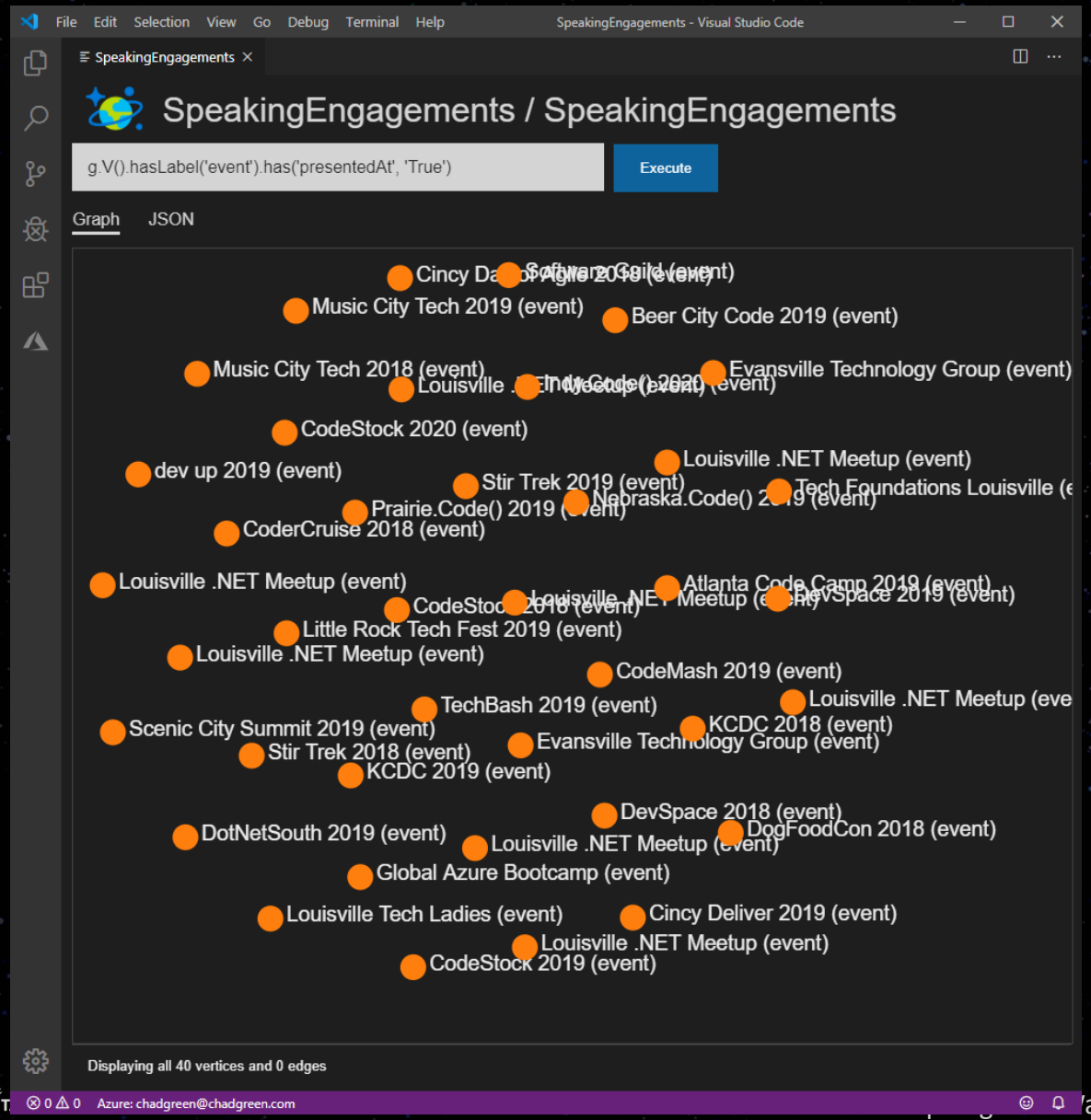

```
g.V()
.hasLabel('tag')
.has('name', 'Graph Data')
.inE('tagged')
.outV()
.outE('submittedTo')
.has('status', 'Confirmed')
.inV()
```

What events have I been scheduled for?




```
g.E()
.hasLabel('submittedTo')
.has('status', 'Confirmed')
.inV()
```

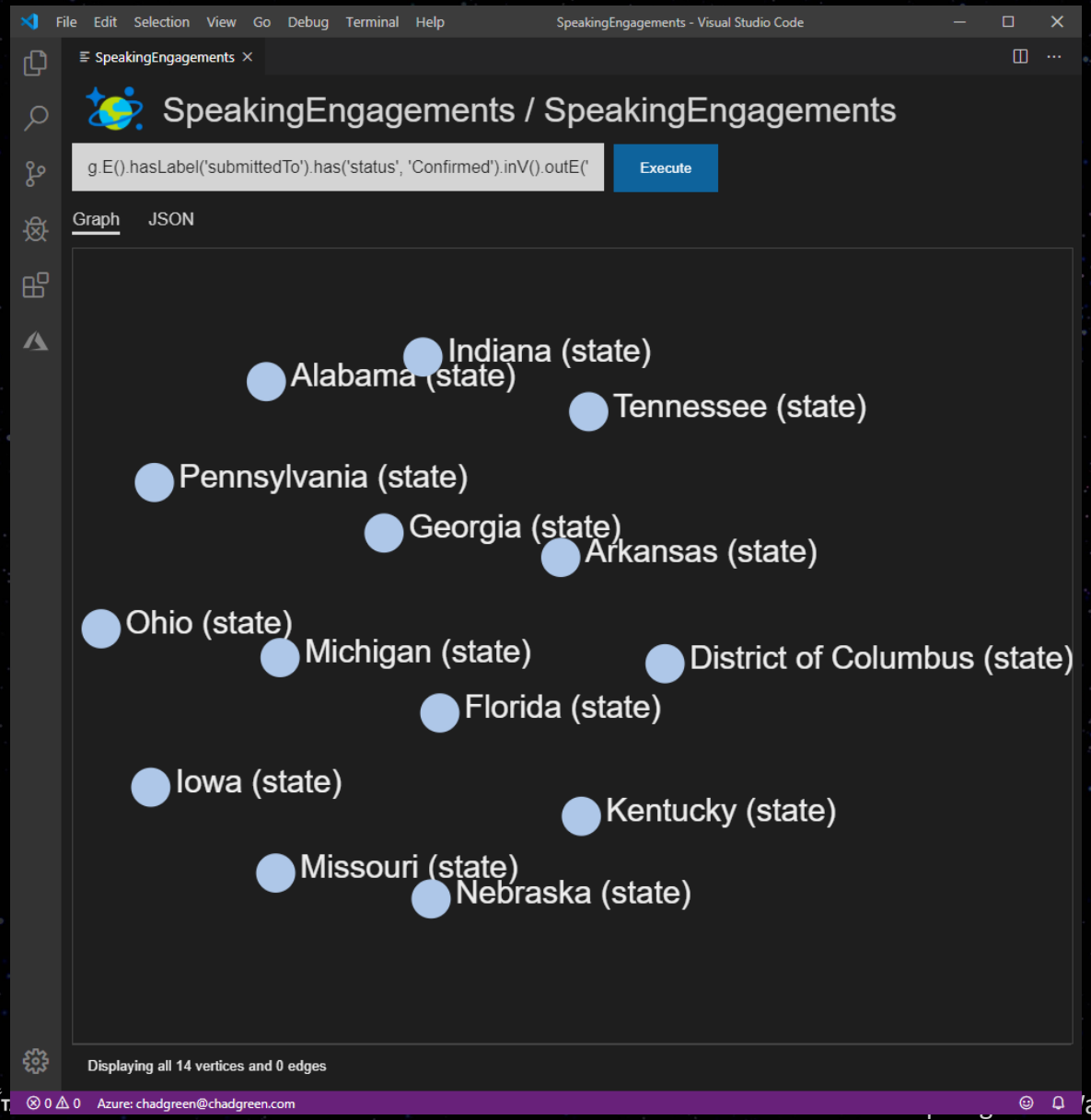
What events have I been scheduled for?




```
g.E()
.hasLabel('submittedTo')
.has('status', 'Confirmed')
.inV()
```

```
g.V()
.hasLabel('event')
.has('presentedAt', 'True')
```

What states have I been scheduled in?




```
g.E()
.hasLabel('submittedTo’)
.has('status', 'Confirmed’)
.inV()
.outE('stateLocation’)
.inV()
```

Wrapping Up

- Graphs – set of objects in which pairs are in some sense related
- Graph Theory – Starts with the 7 bridges of Königsberg
- Graph databases – use graph structure to represent and store data
- Azure Cosmos DB – globally distributed, multi-model database service
- Graph vs Relational – lots of benefits that make graph database worth a look
- Graph Traversal – Navigating graph data using patterns


Thank You!
Graphing Your Way Through the Cosmos
chadgreen@chadgreen.com
TaleLearnCode
TaleLearnCode.com
ChadGreen & TaleLearnCode
ChadwickEGreen
TALELEARNCODE
CHADGREEN