Ontology-Assisted Query of Graph Databases

David Silberberg, Wayne Bethea, Paul Frank, John Gersh, Dennis Patrone, David Patrone, Elisabeth Immer

david.silberberg@jhuapl.edu
443-778-6231

APL
The Johns Hopkins University
APPLIED PHYSICS LABORATORY

Goals of the Graph Query Language (GQL) Project

- To create a comprehensive query language that supports analysis of graph data
  - Unifies disparate graph query approaches into a single, seamless, and declarative language

- Types of analysis supported
  - Graph pattern matching
  - Traditional graph algorithms (e.g., shortest path, minimal spanning tree, cut-sets, finding cliques, etc.)
  - Metrics-oriented graphs algorithms (e.g., betweenness, centrality, etc.)
  - Special analysis methods (e.g., hypothesis expression, etc.)
  - <u>Ontology-assisted graph query</u>

Related Work

- **Four categories of graph query languages and examples**

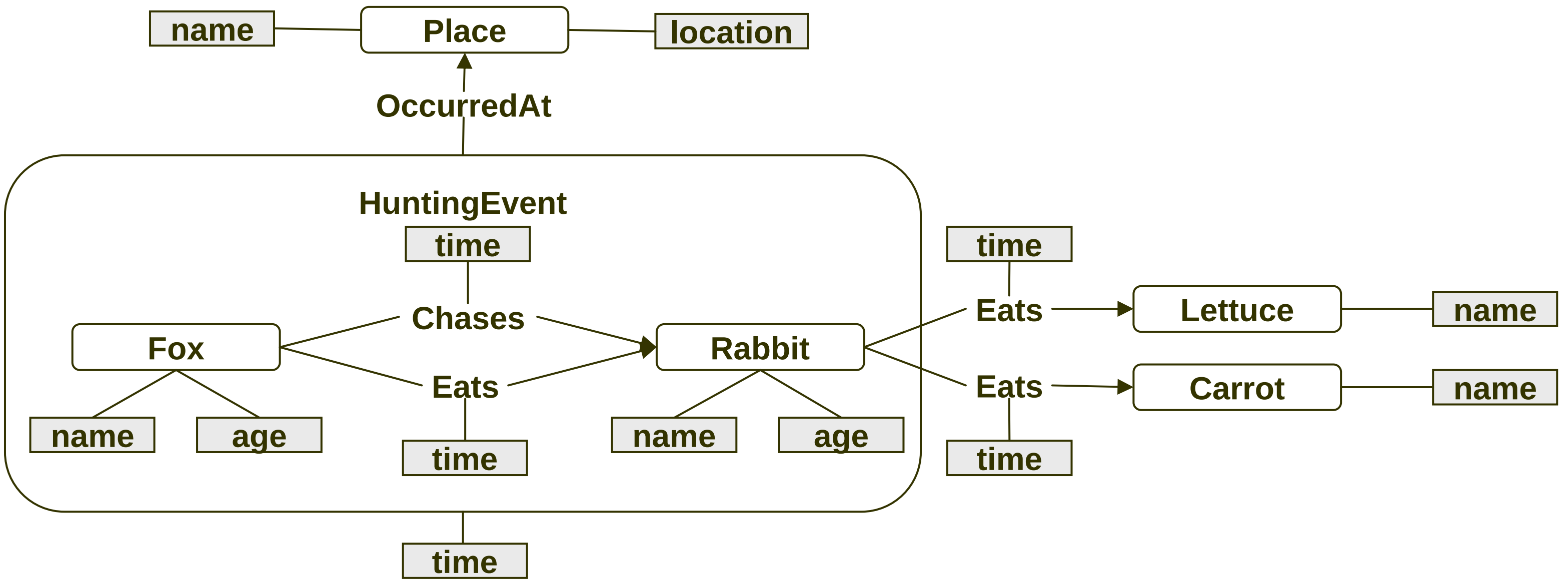
1. **Knowledge base (subject-predicate-object) query languages**
   - **SPARQL, RQL, RAL, RDF Query Language, …**
2. **Graph reasoning query languages**
   - **OWL-QL, GraphLog, Query and Inference Service for RDF, …**
3. **Query languages with graph operators**
   - **GOQL, GRAM, …**
4. **Graphical user interface query language**
   - **QGRAPH**

Features of GQL that Support Analysis

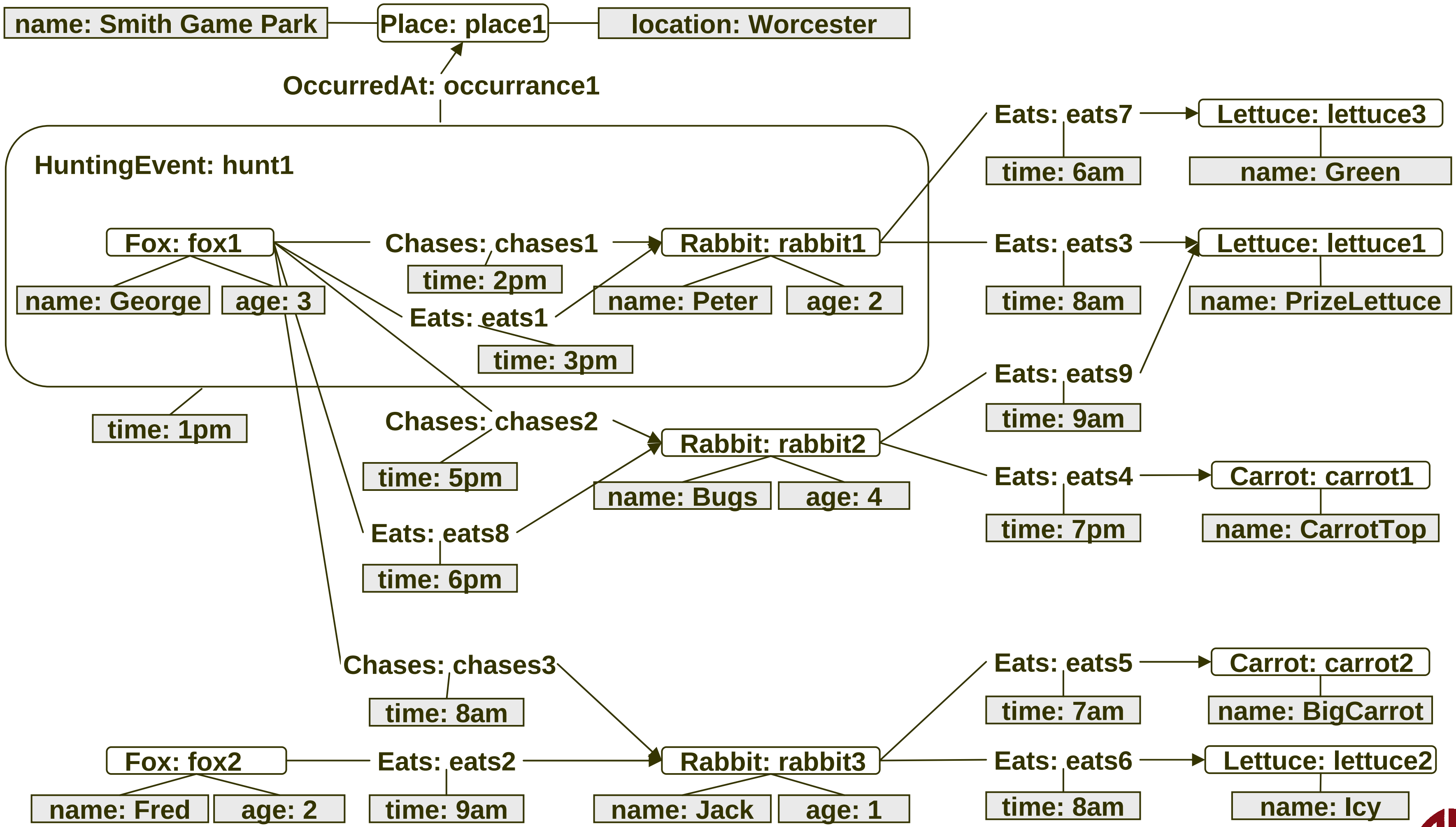
- Schema-based graph query
  - Returns a single graph or a set of graphs (not tables or XML files)
  - Aliasing
  - Graph exploration through wildcard search
  - Embedded queries (helps achieve first order logic expressiveness)
  - Creates new graph structures in query results
  - Query over defined patterns (of activity or behavior, for example)
- Special commands tailored to analysis
  - Hypothesis expressions
  - Composite vertices (of vertices and edges)
- Algorithms
  - External algorithms that return graphs (e.g., shortest path)
  - External algorithms that return metrics (e.g., social network analysis)
- Semantics
  - Ontology-assisted graph query

Example Graph Schema

- Typed *Vertices* and *Edges* that contain *Properties*
- Typed *Composite Vertices*
- Schema describes the <u>structure</u>, not the <u>semantics</u>
  - Labels imply semantics – perhaps different implications among users
  - Do not ensure consistency, consensus, etc.



Example Graph Database



Graph Query Paradigm

- **User interaction**
  - **Direct pattern matching queries to graphs**
    - ***Algorithm speeds are achieved, in part, by exploiting the graph model***
  - **Interpret results represented as a graph or set of graphs**
  - **Iteratively refine, expand, and manipulate graph results**
  - **Apply algorithms to discover interesting subgraphs and graph characteristics**
    - ***Algorithm speeds are achieved, in part, by exploiting the graph model***
  - **Evaluate hypotheses applied to graph data**

- **User emphasis is on graph manipulation operations and not necessarily on *performing inference* or *deriving entailments***

GQL Operators - Overview

- Basic Syntax
  - SUBGRAPH clause
    - Finds a subgraph in the source graph
  - CONSTRAINT clause
    - Filters the subgraph based on property constraints
  - RETURN clause
    - Describes the resulting graph or sets of graphs to return

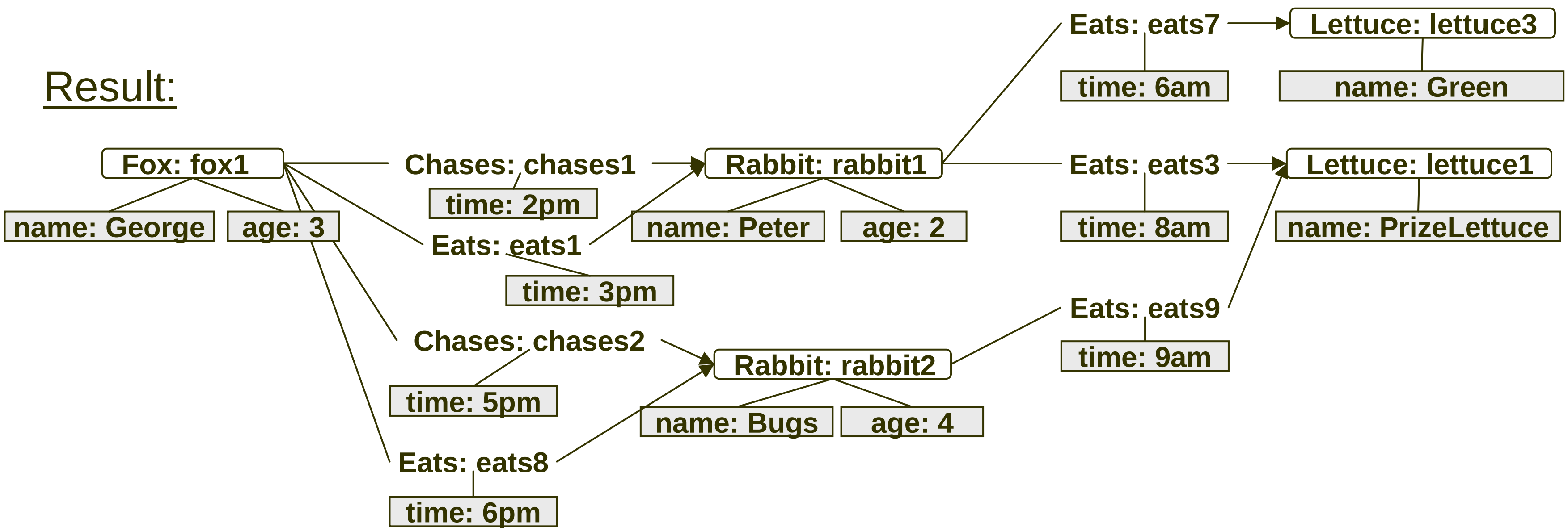
- Syntax for analysis
  - ASSUME clause
    - Supports hypothesis statements
  - PATTERN clause
    - Defines search patterns

BACK

Query that Returns a Single Graph

SUBGRAPH	**Fox Chases Rabbit AND Fox Eats Rabbit AND Rabbit Eats Lettuce**
CONSTRAINT	**Chases.Time < Eats.Time**
RETURN	**Fox Chases Rabbit AND Fox Eats Rabbit AND Rabbit Eats Lettuce**

Result:



Query that Returns a Set of Graphs

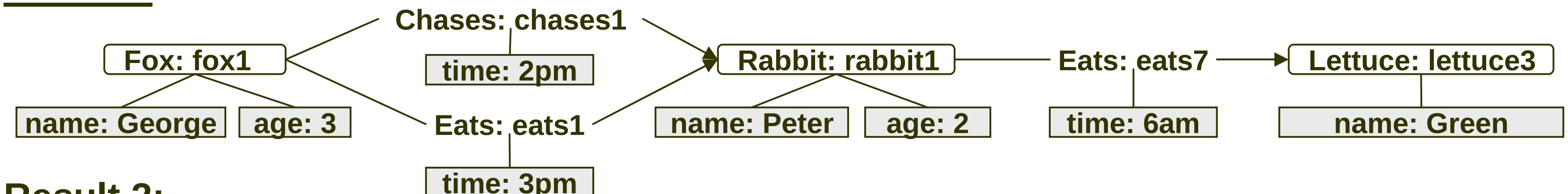
BACK

- Uses the edge expansion operator (#) in the RETURN clause

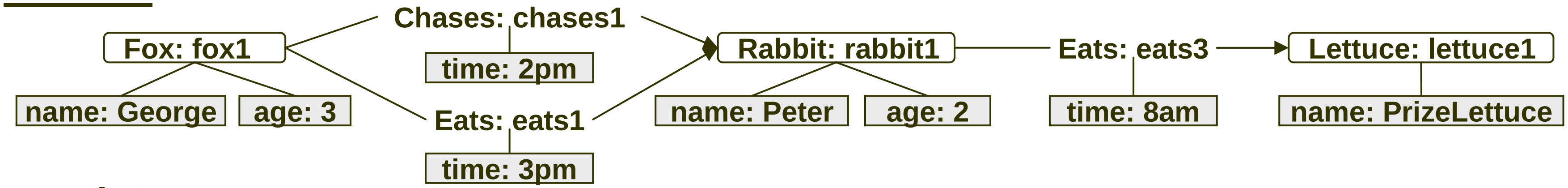
  SUBGRAPH Fox Chases Rabbit AND Fox Eats Rabbit
  AND Rabbit Eats Lettuce

  RETURN Fox Chases# Rabbit AND Fox Eats# Rabbit
  AND Rabbit Eats# Lettuce

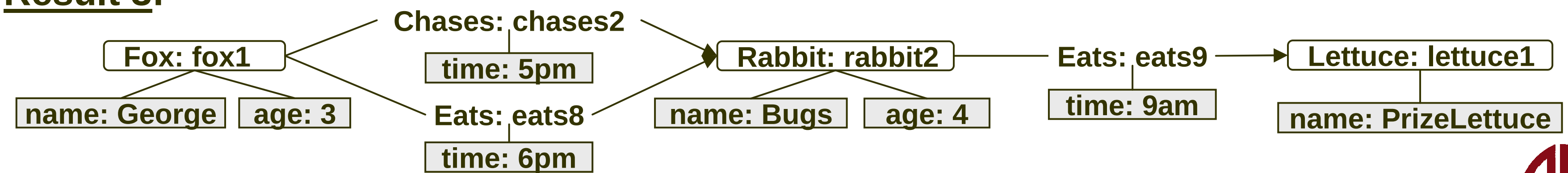
Result 1:



Result 2:



Result 3:



Ontology Assistance

- Goal
  - Maintain the graph query paradigm
  - Maintain separation between ontology and schema
    - Ontology and schema can be developed independently
    - Supports the use of personal ontologies
  - Exploit domain terminology and semantics

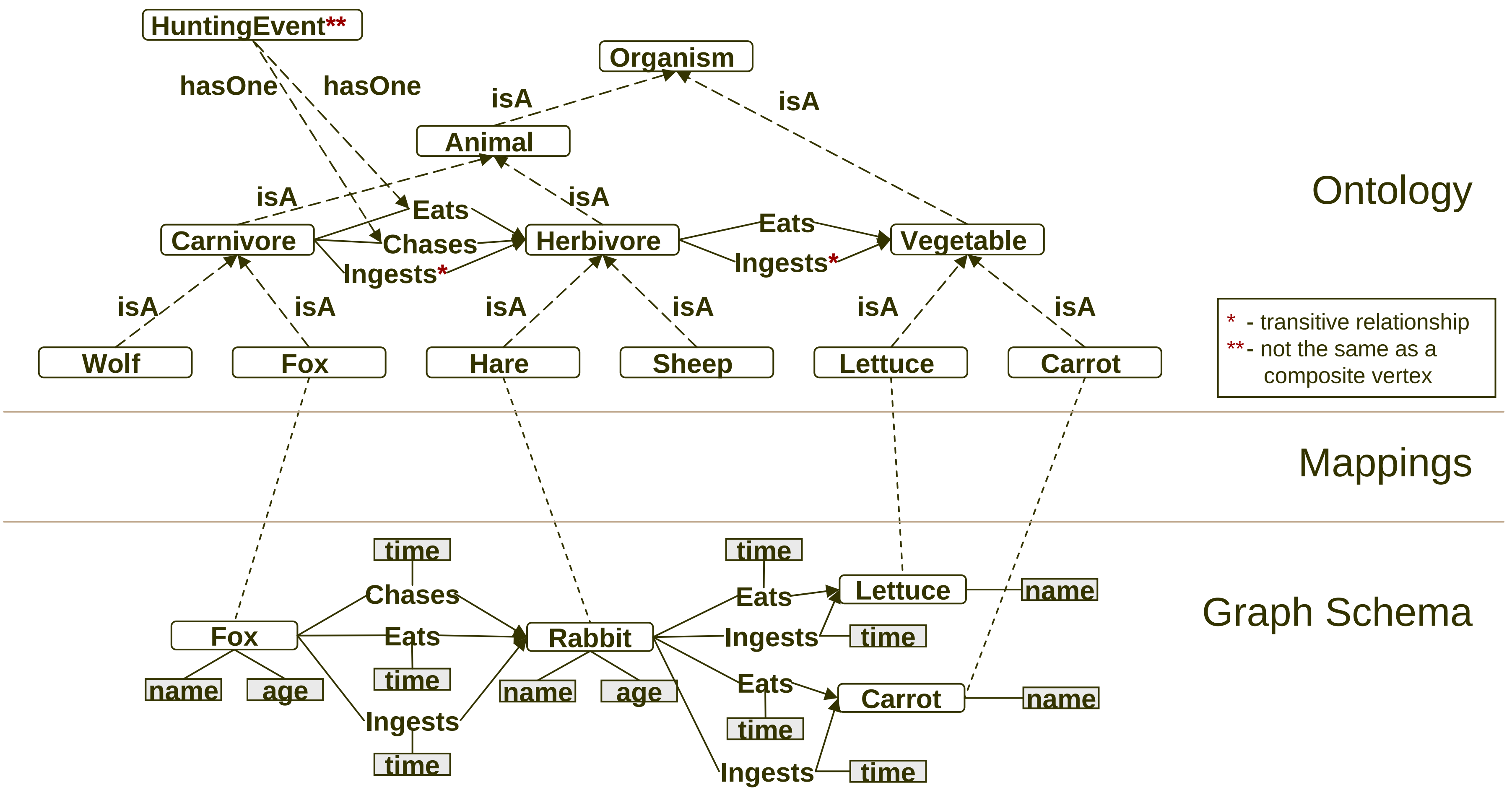
- Examples of ontology assistance
  - Subsumption
  - Transitivity
  - Composite Classes

Virtual Schema

- Terminology available to users
  - Schema elements
    - Vertex, edge and property names
  - Ontology elements
    - Class, relationship and attribute names

- Other virtual schema elements
  - Subclass relationships imply valid relationships among ontology elements
  - Ontology – Schema mappings imply equality between ontology and schema elements

Virtual Schema Example



Subsumption Example
(Ontology to Graph Terms)

SUBGRAPH Carnivore Eats Herbivore

SUBGRAPH   Wolf Eats Rabbit
UNION
SUBGRAPH   Wolf Eats Sheep
UNION
SUBGRAPH   Fox Eats Rabbit
UNION
SUBGRAPH   Fox Eats Sheep

Transitivity Example
(Ontology to Graph Terms)

SUBGRAPH Animal Ingests Organism

SUBGRAPH Wolf Ingests Rabbit
UNION
SUBGRAPH Wolf Ingests Sheep
UNION
SUBGRAPH Fox Ingests Rabbit
UNION
SUBGRAPH Fox Ingests Sheep
UNION
SUBGRAPH Rabbit Ingests Lettuce
UNION
SUBGRAPH Rabbit Ingests Carrot
UNION
SUBGRAPH Sheep Ingests Lettuce
UNION
SUBGRAPH Sheep Ingests Carrot
UNION

SUBGRAPH Wolf Ingests Rabbit AND Rabbit Ingests Lettuce
UNION
SUBGRAPH Wolf Ingests Rabbit AND Rabbit Ingests Carrot
UNION
SUBGRAPH Wolf Ingests Sheep AND Rabbit Ingests Lettuce
UNION
SUBGRAPH Wolf Ingests Sheep AND Rabbit Ingests Carrot
UNION
SUBGRAPH Fox Ingests Rabbit AND Rabbit Ingests Lettuce
UNION
SUBGRAPH Fox Ingests Rabbit AND Rabbit Ingests Carrot
UNION
SUBGRAPH Fox Ingests Rabbit AND Rabbit Ingests Lettuce
UNION
SUBGRAPH Fox Ingests Rabbit AND Rabbit Ingests Carrot

Composite Classes Example
(Ontology to Graph Terms)

SUBGRAPH HuntingEvent

SUBGRAPH	**Wolf Chases Rabbit AND Wolf Eats Rabbit**
UNION	
SUBGRAPH	**Wolf Chases Sheep AND Wolf Eats Sheep**
UNION	
SUBGRAPH	**Fox Chases Rabbit AND Fox Eats Rabbit**
UNION	
SUBGRAPH	**Fox Chases Sheep AND Fox Eats Sheep**

Conclusion

- Goal is to create a comprehensive graph query language
  - Unifies disparate graph query approaches
  - Maintains graph query paradigm
  - Performance gains through preservation of graph structure

- Provide ontology-assistance capabilities
  - Makes explicit the implicit graph schema semantics
  - Exploits domain semantics while maintaining the graph query paradigm

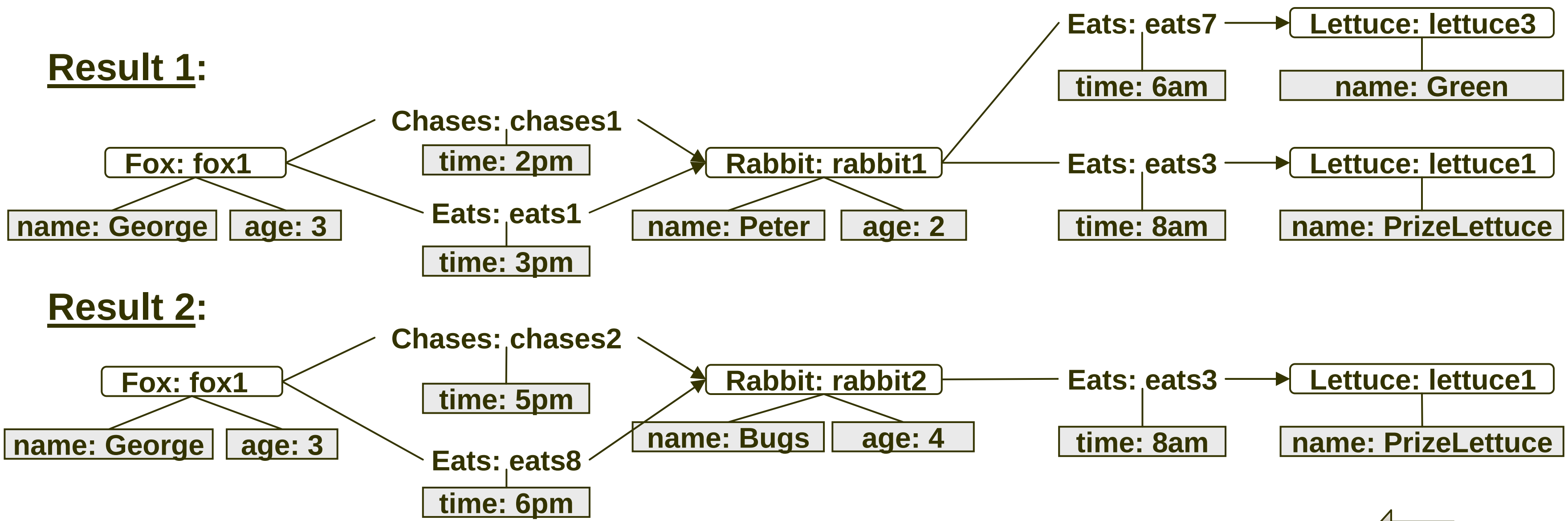
Backup Slides

Query that Returns Hybrid Results

- Some segments of the graph remain whole (Rabbit Eats Lettuce) and other segments are expanded (Fox Chases# Lettuce):

  SUBGRAPH Fox Chases Rabbit AND Fox Eats Rabbit AND Rabbit Eats Lettuce

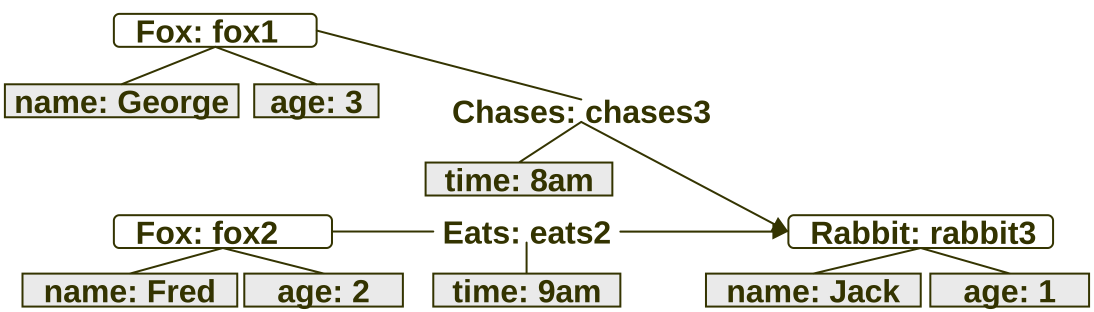
  RETURN Fox Chases# Rabbit AND Fox Eats# Rabbit AND Rabbit Eats Lettuce



BACK

Aliasing

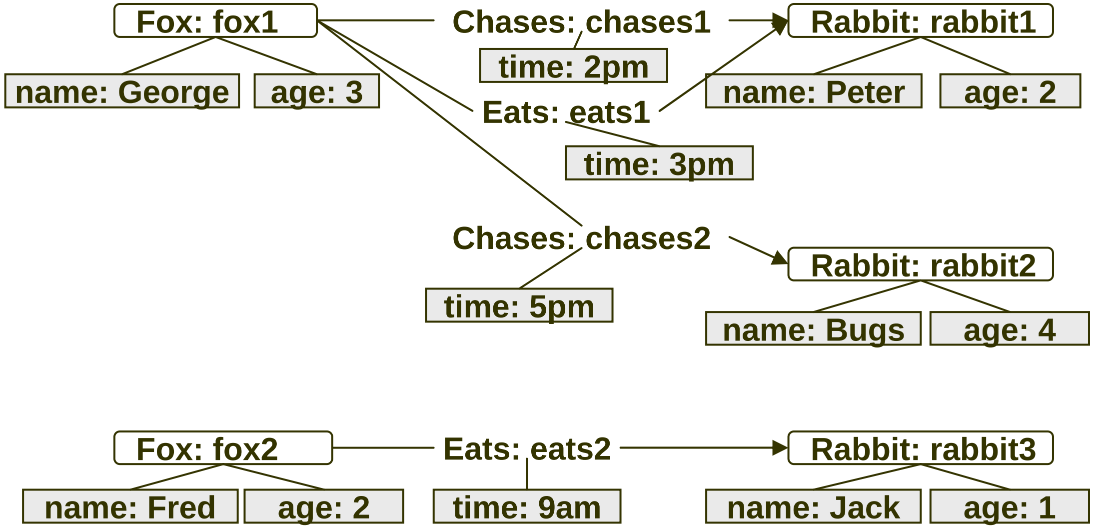
```
SUBGRAPH     Fox ALIAS ChasingFox Chases Rabbit AND
             Fox ALIAS EatingFox Eats Rabbit
CONSTRAINT   ChasingFox.name <> EatingFox.name
RETURN       ChasingFox Chases Rabbit AND
             EatingFox Eats Rabbit
```



BACK

Wildcard Queries

SUBGRAPH Fox * ALIAS InterestingEdge Rabbit
RETURN Fox InterestingEdge Rabbit



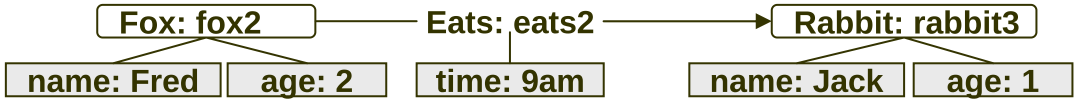
- One edge wildcard queries
- Multiple hops
  - May be computationally expensive in a graph
  - Can be handled by an external AllPath() algorithm

BACK

Embedded Queries

- Significant component of first order logic expressiveness
- To request the first fox that ate a rabbit, the following existential query is formulated:

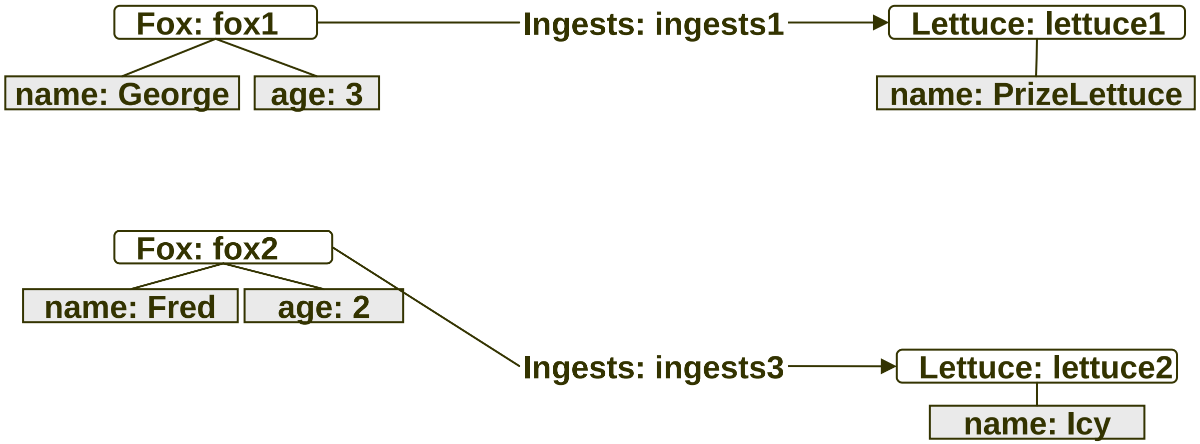
```
SUBGRAPH      Fox Eats ALIAS E1 Rabbit
CONSTRAINT    NOT EXISTS
                (SUBGRAPH    Fox Eats ALIAS E2 Rabbit
                 CONSTRAINT  E1.time > E2.time)
RETURN        Fox Eats Rabbit
```



BACK

New Result Graph Structure Query

SUBGRAPH	**Fox Eats Rabbit AND Rabbit Eats Lettuce**
RETURN	**Fox new(Ingests) Lettuce**



BACK

APL

Pattern Definition

- Assigns names to interesting graph patterns
- Can be reused in multiple queries

```
PATTERN Predator (Fox new(PreysUpon) Rabbit) =
        SUBGRAPH      Fox Chases Rabbit AND
                      Fox Eats Rabbit
        CONSTRAINT    Chases.time < Eats.time
        RETURN        Fox new(PreysUpon) Rabbit
```

Pattern Use

- **Query:**

  SUBGRAPH Predator(Fox PreysUpon Rabbit) AND Rabbit Eats Lettuce
  RETURN Fox new(Ingests) Lettuce

- **Is evaluated as if it were:**

  SUBGRAPH Fox Chases Rabbit AND Fox Eats Rabbit AND Rabbit Eats Lettuce
  CONSTRAINT Chases.time < Eats.time
  RETURN Fox new(Ingests) Lettuce

BACK

Hypothesis Expressions

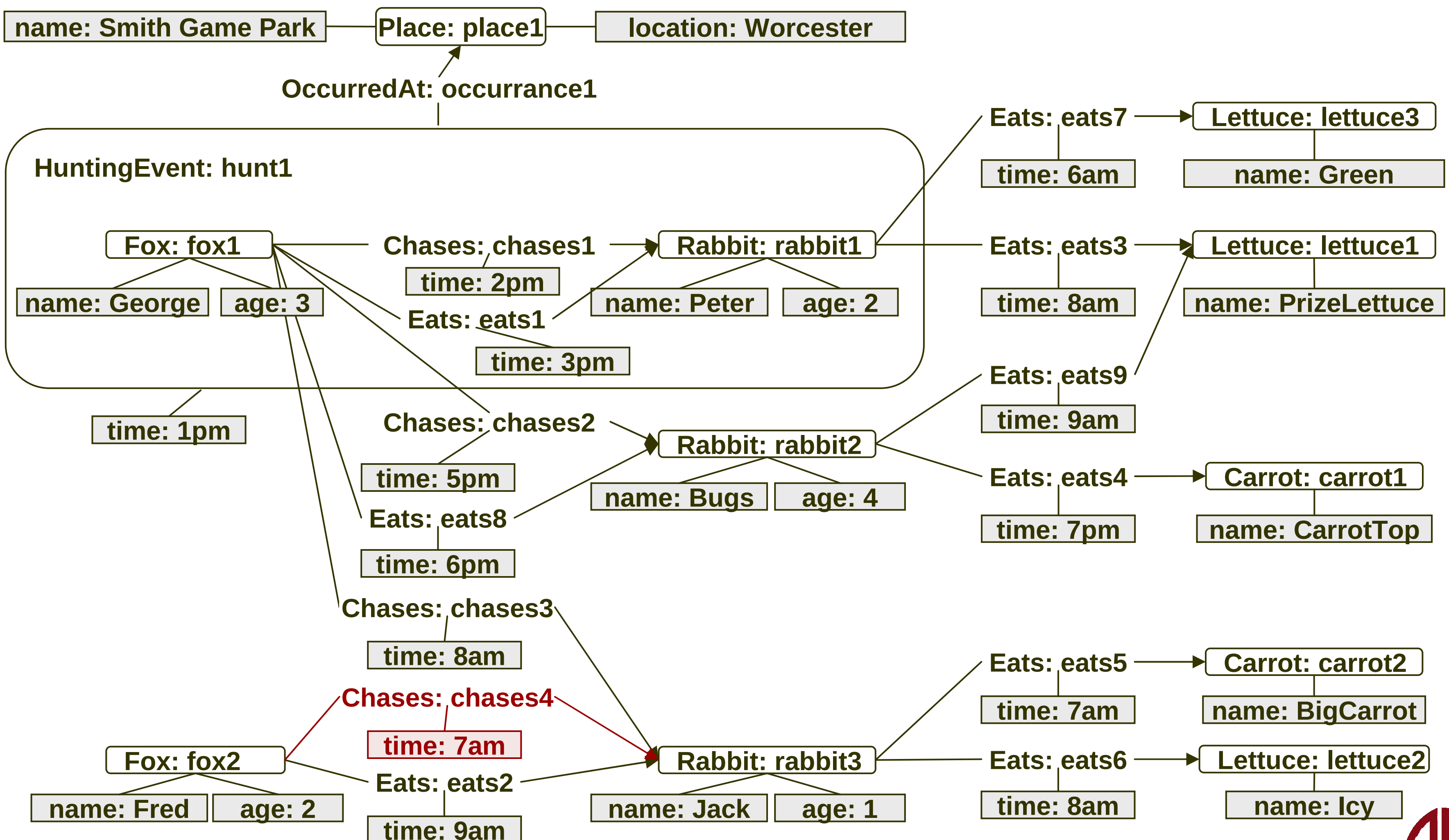
- Enables queries on hypothetical data

```
SUBGRAPH    Fox Chases Rabbit AND Fox Eats Rabbit AND
            Rabbit Eats Lettuce
CONSTRAINT  Chases.time < '8am'
RETURN      Fox Chases Rabbit AND Fox Eats Rabbit AND
            Rabbit Eats Lettuce
ASSUME      EDGE Chases [NEW time = '7am']
                 FROM Fox[CONSTRAINT name= 'Fred']
                 TO   Rabbit[CONSTRAINT name= 'Jack']
```

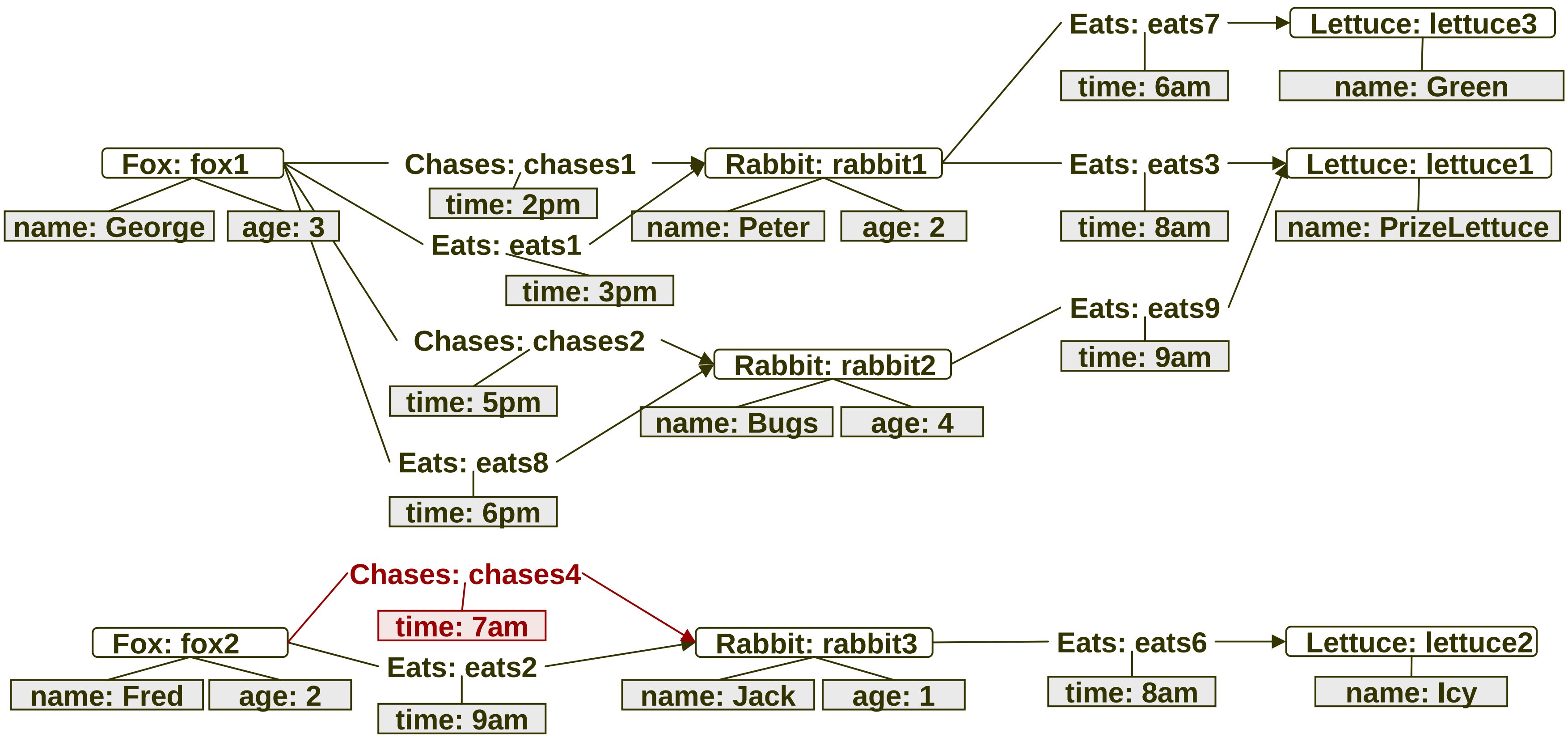
- Motivated by OWL-QL

BACK

Virtual Graph Element Insertion



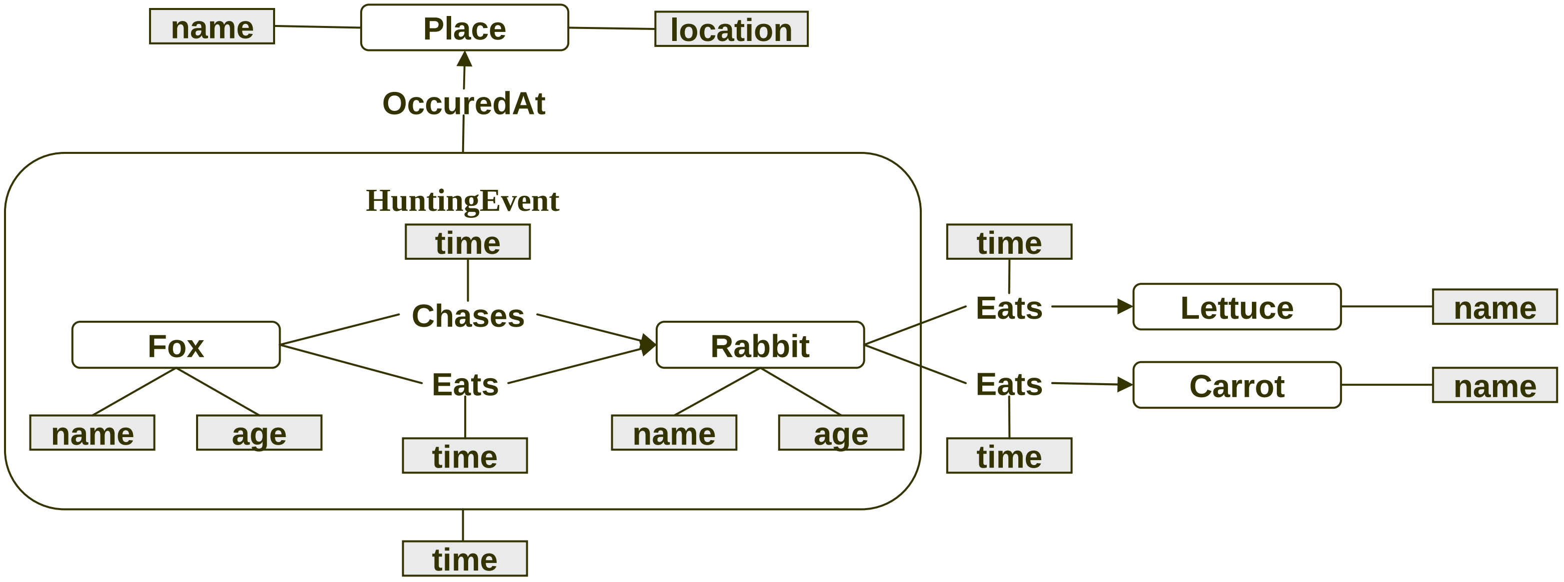
Hypothesis Expression Query Result



BACK

Composite Vertices

- Composite vertices
  - Composed of vertices and edges
  - Contained vertices can be composite as well

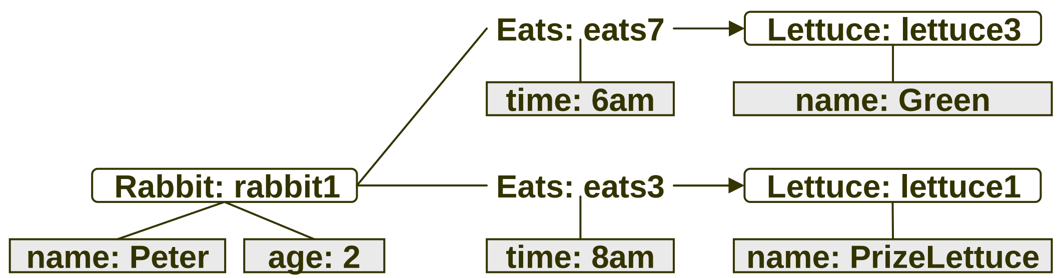


Composite Vertex Queries - continued

SUBGRAPH HuntingEvent OccuredAt Place AND
HuntingEvent DIRECTLY CONTAINS Rabbit AND
Rabbit Eats Lettuce

CONSTRAINT Place.name = ‘Smith Game Park’

RETURN Rabbit Eats Lettuce



- Addresses a subset of Harel's Higraphs
- Multiple hops
  - CONTAINS or IS-CONTAINED-BY
  - Feasible because of the hierarchy

BACK

External Graph Algorithms that Return Subgraphs

- Shortest Path

SUBGRAPH	GameWarden Chases Fox AND ShortestPath(Fox, Rabbit) ALIAS SP_alias AND Rabbit Eats Lettuce
RETURN	GameWarden Chases Fox AND SP_alias AND Rabbit Eats Lettuce

- Adjacent Vertices

SUBGRAPH	AdjacentVertices(Rabbit) ALIAS AV_alias
CONSTRAINT	count_edges(Rabbit) > 10
RETURN	AV_alias

BACK

External Graph Algorithms that Return Metrics

- **Centrality: Find the Foxes that eventually Eat the Rabbits, who play a central role in the garden activities**

SUBGRAPH	**Fox Eats Rabbit**
CONSTRAINT	**Centrality (Rabbit) > .8**
RETURN	**Fox Eats Rabbit**

- **Clustering Coefficient: Find the Foxes that are likely to work together when Chasing Rabbits**

SUBGRAPH	**Fox ALIAS Fox1 Chases Rabbit AND**
	Fox ALIAS Fox2 Chases Rabbit
CONSTRAINT	**ClusteringCoefficient (Fox1, Fox2) > .6**
	AND Fox1 <> Fox2
RETURN	**Fox Eats Rabbit**

BACK