Query Languages for Graph Databases

Susan B. Davidson

CIS 700: Advanced Topics in Databases

MW 1:30-3

Towne 309

http://www.cis.upenn.edu/~susan/cis700/homepage.html

Graph Databases

- Offer a more intuitive representation for many modern applications
  - Social networks
  - Transportation networks
  - Biological pathways
  - Citation networks
  - …
- A number of graph database engines, data models and query languages have been released over the past few years.

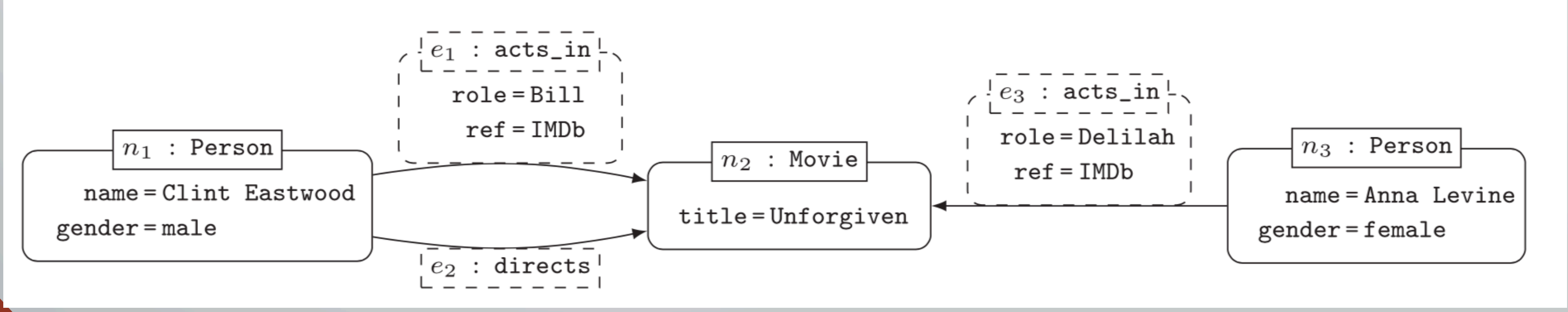
Outline

- **Graph data models: edge-labeled and property graphs**
- Graph patterns
  - Basic and complex
  - How expressed in SPARQL and Cypher
- Navigational queries
  - Regular path queries, integrated into graph pattern queries
  - How expressed in SPARQL and Cypher

Graph data models

- Edge-labeled graph, e.g. used in RDF

- Property graph, e.g. used in Neo4j



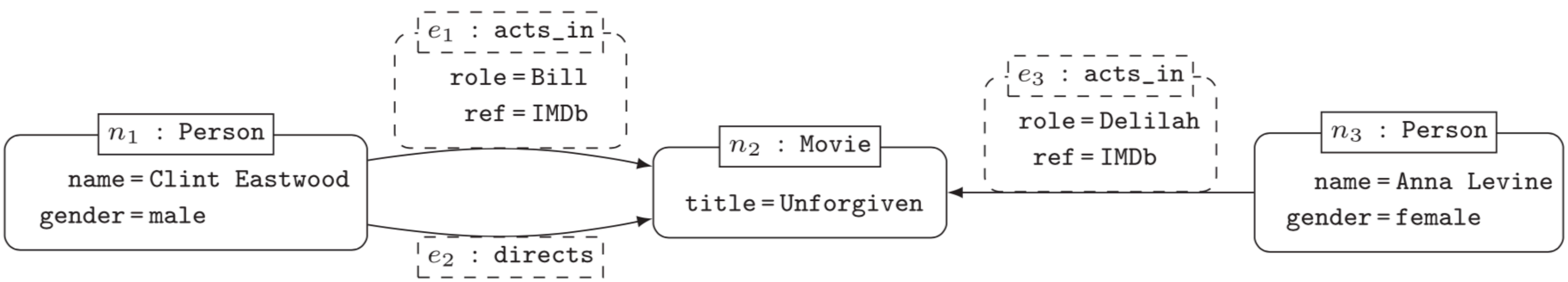
Components of property graphs

$$V = \{n_1, n_2, n_3\} \qquad E = \{e_1, e_2, e_3\}$$

$$\rho(e_1) = (n_1, n_2) \qquad \rho(e_2) = (n_1, n_2)$$
$$\rho(e_3) = (n_3, n_2)$$

$$\lambda(n_1) = \texttt{Person} \qquad \lambda(n_2) = \texttt{Movie}$$
$$\lambda(n_3) = \texttt{Person} \qquad \lambda(e_1) = \texttt{acts\_in}$$
$$\lambda(e_2) = \texttt{directs} \qquad \lambda(e_3) = \texttt{acts\_in}$$

$$\sigma(n_1, \texttt{name}) = \texttt{Clint Eastwood}$$
$$\sigma(n_1, \texttt{gender}) = \texttt{male}$$
$$\sigma(n_2, \texttt{title}) = \texttt{Unforgiven}$$
$$\sigma(n_3, \texttt{name}) = \texttt{Anna Levine}$$
$$\sigma(n_3, \texttt{gender}) = \texttt{female}$$
$$\sigma(e_1, \texttt{role}) = \texttt{Bill}$$
$$\sigma(e_1, \texttt{ref}) = \texttt{IMDb}$$
$$\sigma(e_3, \texttt{role}) = \texttt{Delilah}$$
$$\sigma(e_3, \texttt{ref}) = \texttt{IMDb}$$



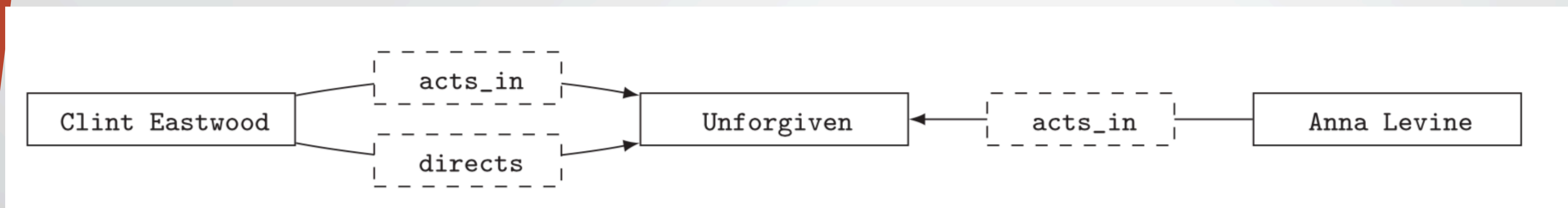
Graph query languages: Core features

- Pattern matching
  - Basic graph patterns (bgp)
  - Complex graph patterns (cgp): bgp extended to include operators such as projection, union, options, etc.
- Navigation
  - Use paths as a core, e.g. regular path queries (RPQs)
  - Navigational graph patterns (ngps): paths incorporated into bgps
  - Complex navigational graph patterns (cngps): ngps extended with operators

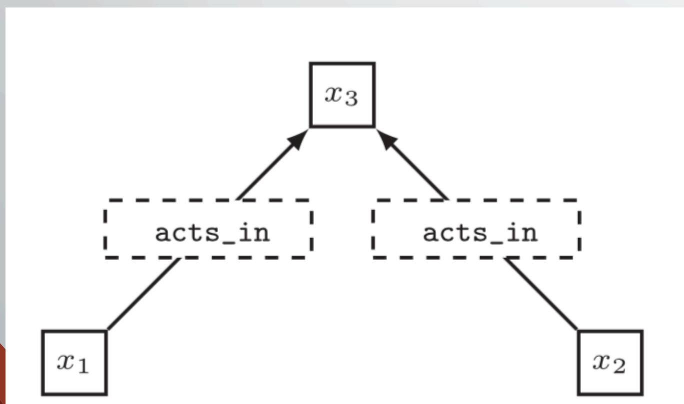
Outline

- Graph data models: edge-labeled and property graphs
- **Graph patterns**
  - Basic and complex
  - How expressed in SPARQL and Cypher
- Navigational queries
  - Regular path queries, integrated into graph pattern queries
  - How expressed in SPARQL and Cypher

Basic graph patterns (edge-labeled graph)

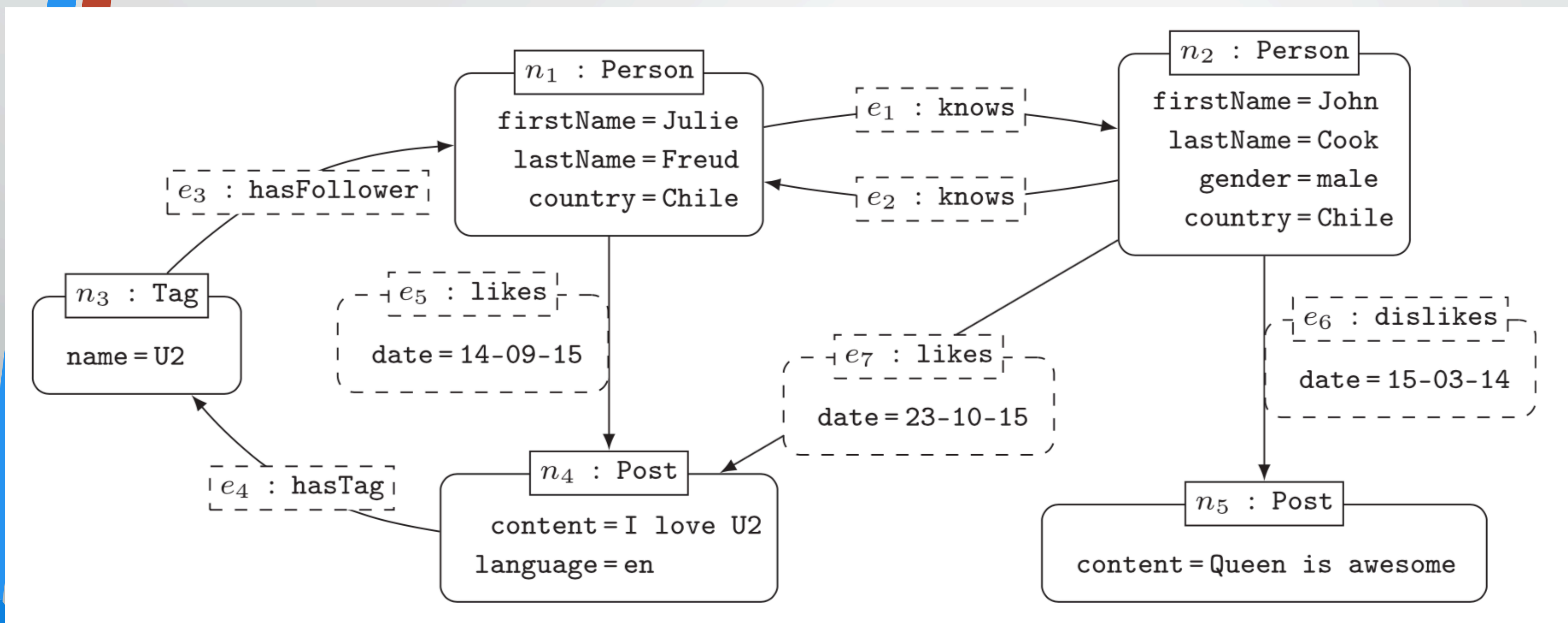


- Find all co-stars.



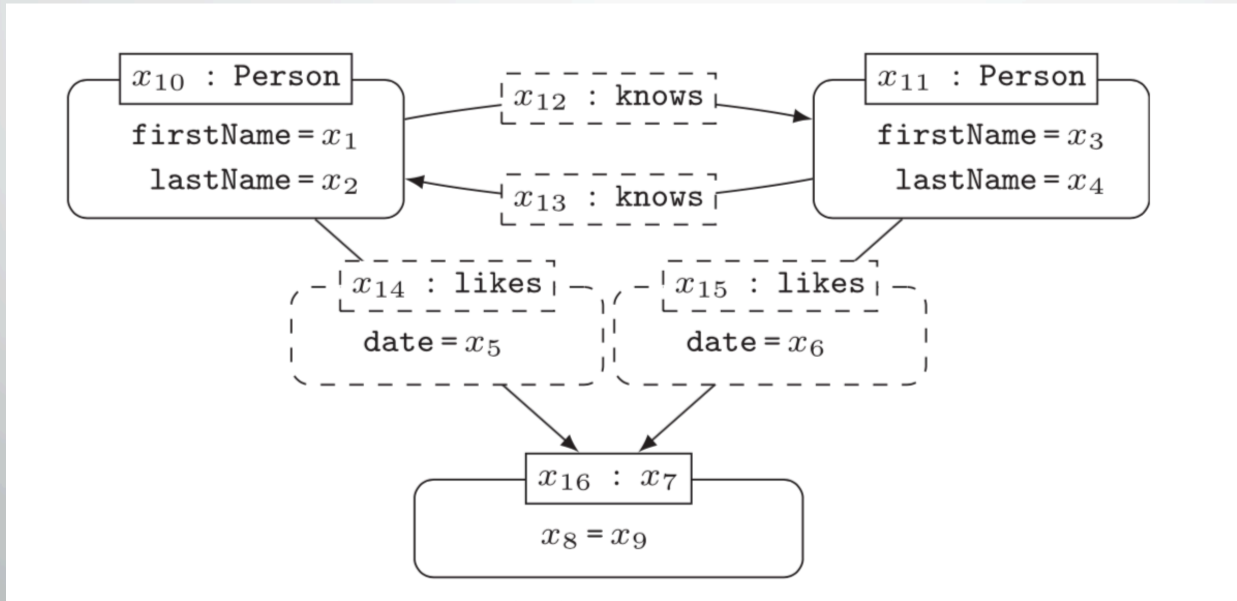
x_1	x_2	x_3
Clint Eastwood	Anna Levine	Unforgiven
Anna Levine	Clint Eastwood	Unforgiven
Clint Eastwood	Clint Eastwood	Unforgiven
Anna Levine	Anna Levine	Unforgiven

Social Network Property Graph

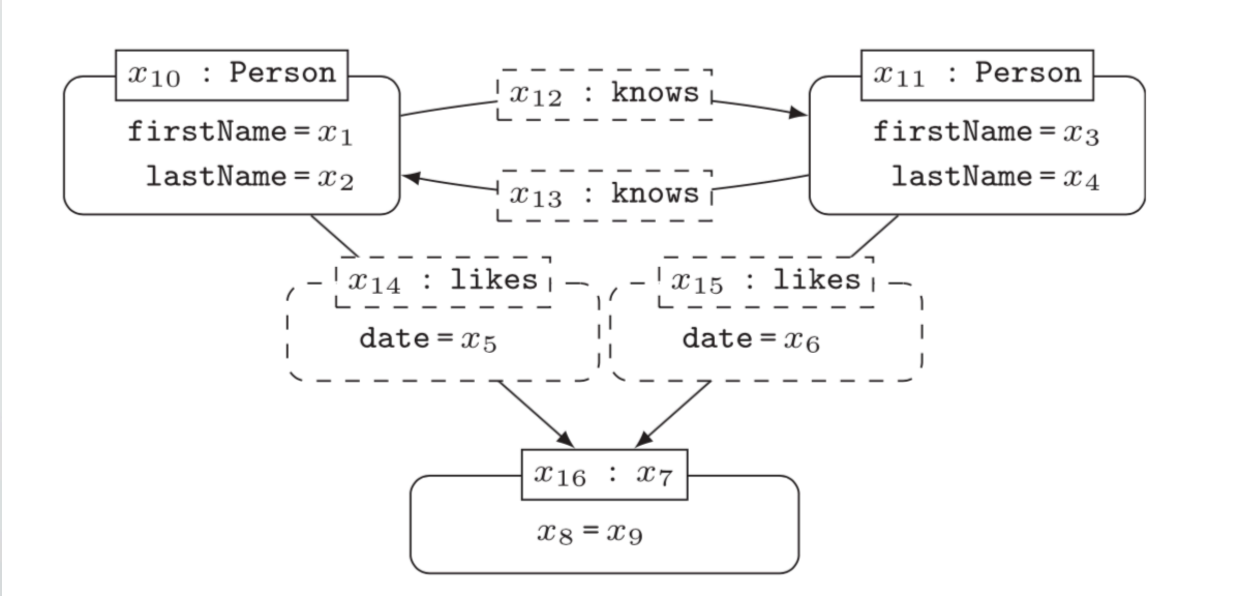


Basic graph patterns (property graph)

- Things that (mutual) friends in the social network both like. Return first and last names, all details of the items they both like, and the date on which they both like the items.



Query result

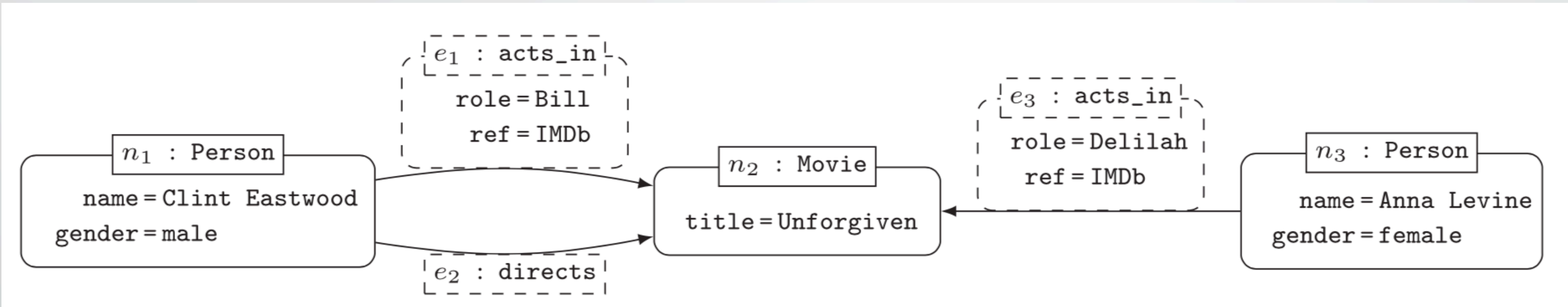


x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	...
Julie	Freud	John	Cook	14-09-15	23-10-15	Post	content	I love U2	n_1	...
John	Cook	Julie	Freud	23-10-15	14-09-15	Post	content	I love U2	n_2	...
Julie	Freud	John	Cook	14-09-15	23-10-15	Post	language	en	n_1	...
John	Cook	Julie	Freud	23-10-15	14-09-15	Post	language	en	n_2	...

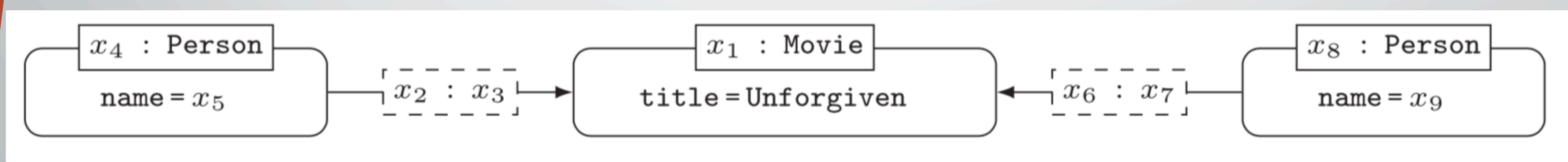
Evaluation

- **Def. 3.5 (Match)**: Given an edge-labeled graph G= (V, E) and a bgp Q= (V', E') a *match* of Q in G is a mapping from the set of constants and variables in Q to contants in G such that:
  - Constants are mapped to themselves: h(a)= a
  - Each edge of Q is mapped to an edge of G which preserves the structure of Q in its image under h in G: for each (b, l, c) in E' it holds that (h(b), h(l), h(c)) is in E
- This leads to three different semantics for evaluation:
  - Homomorphism-based semantics – currently used in SPARQL (RDF)
  - Isomorphism-based semantics
    - No-repeated anything: no two variables can be bound to the same term
    - No-repeated node
    - No-repeated edge -– currently used in Cypher (Neo4j)

Effect of semantics: sample query



- Consider the following query:



Effect of semantics: query result

- Unrestricted semantics:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
n_2	e_2	directs	n_1	Clint Eastwood	e_3	acts_in	n_3	Anna Levine
n_2	e_3	acts_in	n_3	Anna Levine	e_2	directs	n_1	Clint Eastwood
n_2	e_1	acts_in	n_1	Clint Eastwood	e_3	acts_in	n_3	Anna Levine
n_2	e_3	acts_in	n_3	Anna Levine	e_1	acts_in	n_1	Clint Eastwood
n_2	e_2	directs	n_1	Clint Eastwood	e_1	acts_in	n_1	Clint Eastwood
n_2	e_1	acts_in	n_1	Clint Eastwood	e_2	directs	n_1	Clint Eastwood
n_2	e_1	acts_in	n_1	Clint Eastwood	e_1	acts_in	n_1	Clint Eastwood
n_2	e_2	directs	n_1	Clint Eastwood	e_2	directs	n_1	Clint Eastwood
n_2	e_3	acts_in	n_1	Anna Levine	e_3	acts_in	n_1	Anna Levine

Effect of semantics: query result

- No-repeated anything:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
n_2	e_2	directs	n_1	Clint Eastwood	e_3	acts_in	n_3	Anna Levine
n_2	e_3	acts_in	n_3	Anna Levine	e_2	directs	n_1	Clint Eastwood
~~n_2~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~	~~e_3~~	~~acts_in~~	~~n_3~~	~~Anna Levine~~
~~n_2~~	~~e_3~~	~~acts_in~~	~~n_3~~	~~Anna Levine~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~
~~n_2~~	~~e_2~~	~~directs~~	~~n_1~~	~~Clint Eastwood~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~
~~n_2~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~	~~e_2~~	~~directs~~	~~n_1~~	~~Clint Eastwood~~
~~n_2~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~
~~n_2~~	~~e_2~~	~~directs~~	~~n_1~~	~~Clint Eastwood~~	~~e_2~~	~~directs~~	~~n_1~~	~~Clint Eastwood~~
~~n_2~~	~~e_3~~	~~acts_in~~	~~n_1~~	~~Anna Levine~~	~~e_3~~	~~acts_in~~	~~n_1~~	~~Anna Levine~~

Effect of semantics: query result

- No-repeated node:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
n_2	e_2	directs	n_1	Clint Eastwood	e_3	acts_in	n_3	Anna Levine
n_2	e_3	acts_in	n_3	Anna Levine	e_2	directs	n_1	Clint Eastwood
n_2	e_1	acts_in	n_1	Clint Eastwood	e_3	acts_in	n_3	Anna Levine
n_2	e_3	acts_in	n_3	Anna Levine	e_1	acts_in	n_1	Clint Eastwood
~~n_2~~	~~e_2~~	~~directs~~	~~n_1~~	~~Clint Eastwood~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~
~~n_2~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~	~~e_2~~	~~directs~~	~~n_1~~	~~Clint Eastwood~~
~~n_2~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~	~~e_1~~	~~acts_in~~	~~n_1~~	~~Clint Eastwood~~
~~n_2~~	~~e_2~~	~~directs~~	~~n_1~~	~~Clint Eastwood~~	~~e_2~~	~~directs~~	~~n_1~~	~~Clint Eastwood~~
~~n_2~~	~~e_3~~	~~acts_in~~	~~n_1~~	~~Anna Levine~~	~~e_3~~	~~acts_in~~	~~n_1~~	~~Anna Levine~~

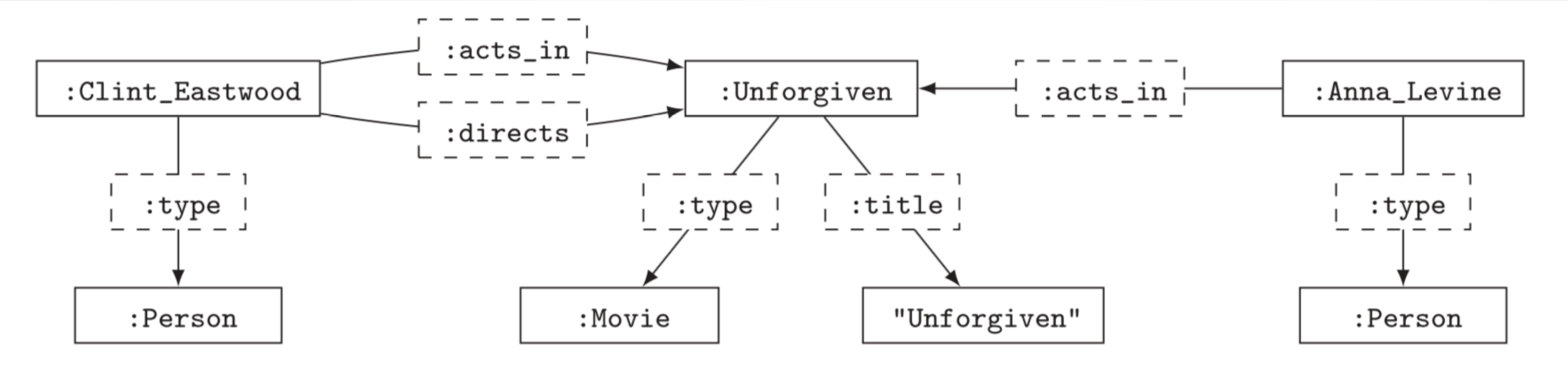
Complex graph patterns

- Basic graph patterns cover natural join and selection based on equality
- Complex graph patters add further traditional relational operators:
  - Projection, union, difference, optional (aka left-outer-join), filter

SPARQL

- W3C standard for query RDF graphs
- Based on triple patterns (subject, predicate, object), where variables are indicated by “?”
- Supports all complex graph pattern features
- Uses a homomorphism-based semantics

Sample RDF graph



SPARQL query: projection and filter

```
SELECT ?x1 ?x2
WHERE {
      ?x1 :acts_in ?x3 . ?x1 :type :Person .
      ?x2 :acts_in ?x3 . ?x2 :type :Person .
      ?x3 :title "Unforgiven" . ?x3 :type :Movie .
      FILTER(?x1 != ?x2)
 }
```

- **Result:**

?x1	?x2
:Clint_Eastwood	:Anna_Levine
:Anna_Levine	:Clint_Eastwood

SPARQL queries: union, difference

```
SELECT ?x
WHERE {{ :Clint_Eastwood :acts_in ?x . } UNION { :Clint_Eastwood :directs ?x . }}
```

- **Result:** :Unforgiven

```
SELECT ?x
WHERE {{ ?x :acts_in :Unforgiven . } MINUS { ?x :directs :Unforgiven . }}
```

- **Result:** :Anna_Levine

SPARQL query: optional

```
SELECT ?x1 ?x2 ?x3
WHERE {{ ?x1 :acts_in ?x2 .} OPTIONAL { ?x1 ?x3 ?x2 . FILTER(?x3 != :acts_in) }}
```

- **Result:**

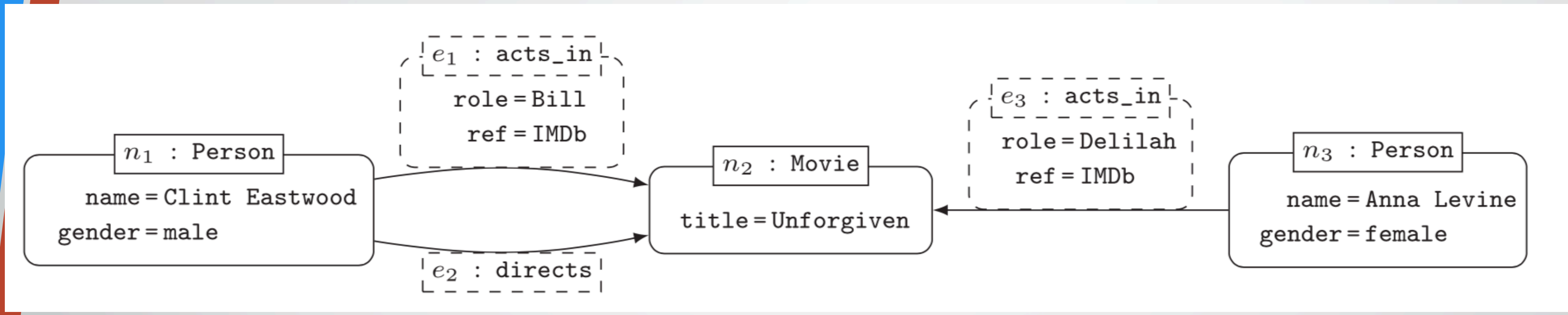
?x1	?x2	?x3
:Clint_Eastwood	:Unforgiven	:directs
:Anna_Levine	:Unforgiven	

Cypher

- Query language for Neo4j, based on patterns
- Semantics: Isomorphism-based no-repeated edges
- Syntax:
  - Nodes are written inside "( )" and edges inside of "[ ]".
  - Filters for labels specified using ":"
  - Values for properties specified using "{ }"
  - Return clause projects output variables

```
MATCH (x1:Person) -[:acts_in]-> (:Movie {title:"Unforgiven"})
                   <-[:acts_in]- (x2:Person)}
RETURN x1,x2
```

Sample graph



Isomorphism-based semantics

```
MATCH (x1:Person) -[:acts_in]-> (:Movie {title:"Unforgiven"})
                  <-[:acts_in]- (x2:Person)}
RETURN x1,x2
```

```
MATCH (x1:Person) -[:acts_in]-> (x3:Movie {title:"Unforgiven"})
MATCH (x2:Person) -[:acts_in]-> (x3)
RETURN x1,x2
```

Cypher queries: union, difference

```
MATCH (:Person {name:"Clint Eastwood"}) -[:acts_in]-> (x3:Movie)
RETURN x3.title
UNION ALL MATCH (:Person {name:"Clint Eastwood"}) -[:directs]-> (x3:Movie)
RETURN x3.title
```

- **Result:** {"Unforgiven", "Unforgiven"}

```
MATCH (x1:Person) -[:acts_in]-> (x3:Movie {title:"Unforgiven"})
WHERE NOT (x1) -[:directs]-> (x3)
RETURN x1.name
```

- **Result:** {"Anna Levine"}

Cypher queries: optional

```
SELECT ?x1 ?x2 ?x3
WHERE {{ ?x1 :acts_in ?x2 .} OPTIONAL { ?x1 ?x3 ?x2 . FILTER(?x3 != :acts_in) }}
```

- **Result:**

?x1	?x2	?x3
:Clint_Eastwood	:Unforgiven	:directs
:Anna_Levine	:Unforgiven	

Outline

- Graph data models: edge-labeled and property graphs
- Graph patterns
  - Basic and complex
  - How expressed in SPARQL and Cypher
- **Navigational queries**
  - Regular path queries, integrated into graph pattern queries
  - How expressed in SPARQL and Cypher

Navigational graph queries

- Path queries are added to basic graph queries: $x \xrightarrow{\alpha} y$
- α is a regular expression over the set of edge labels
  - knows+
  - knows+.likes
  - knows+.(likes | dislikes)
  - *
- Inverse operator to allow backward edge traversal
  - acts_in. acts_in $^{-}$

Evaluation of path query

- All paths in G whose label satisfies α
- But there may be an infinite number of such paths (cycles), so in practice:
  - Arbitrary path semantics: test existence of such a path, or pairs of nodes connected by such paths (finite)
  - Shortest path semantics
  - No-repeated node semantics (simple paths)
  - No-repeated edge semantics (used in Cypher)
- Output:
  - Boolean
  - Nodes
  - Paths
  - Graphs (compact representation)

Navigational graph patterns (ngps)

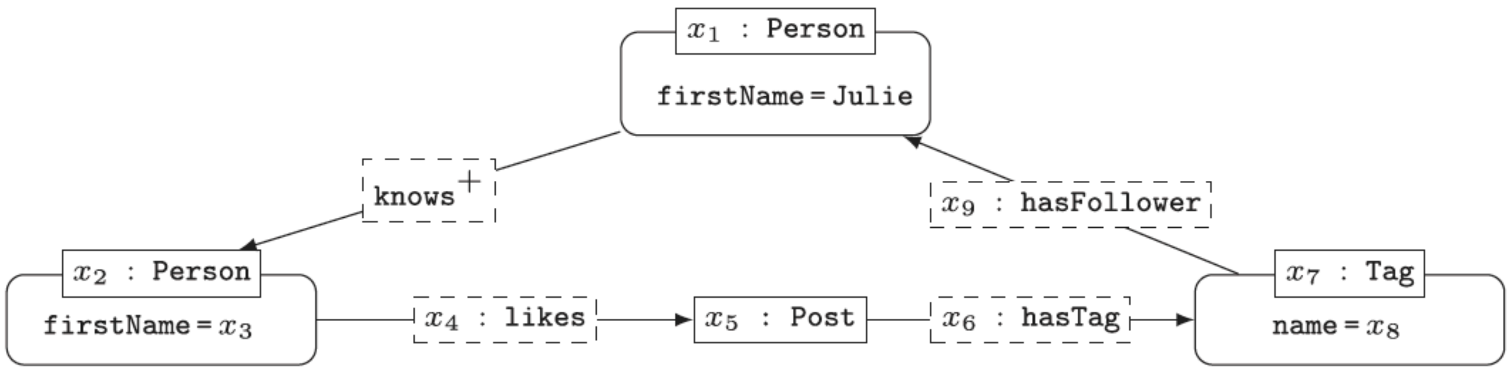


Fig. 10. A navigational graph pattern that characterises the friends of friends of Julie that like a post with a tag she that she follows.

- Complex navigational graph patterns (cngps)
  - Project x5: recommended posts for Julie
  - Union: recommended posts for John or Julie
  - Intersection: recommended posts for both John and Julie
  - Difference: recommended posts for Julie but not John

SPARQL examples

- All pairs of actors who have finite collaboration distance

```
SELECT ?x ?y
WHERE  ?x (:acts_in/^:acts_in)* ?y
```

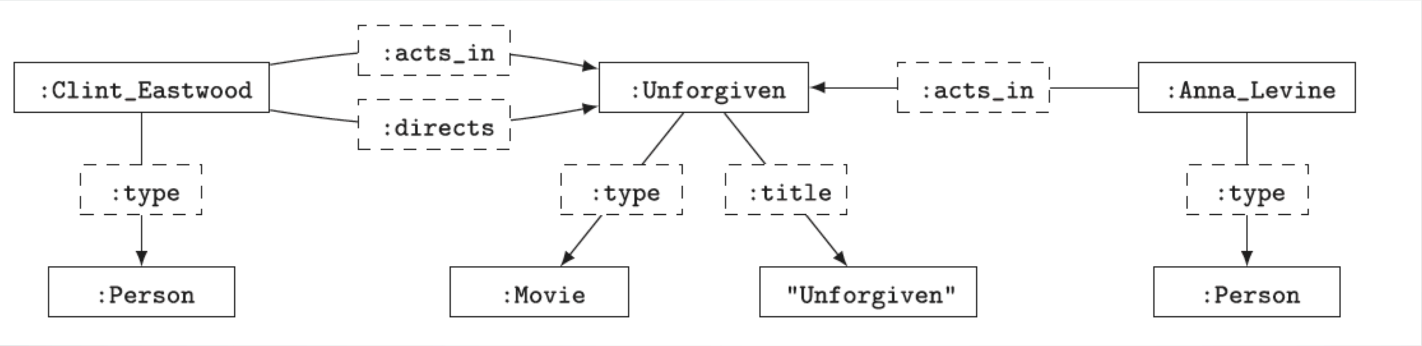
- All people with a finite Erdos-Bacon number

```
SELECT ?x
WHERE { ?x (:acts_in/^:acts_in) * :Kevin_Bacon . ?x (:author/^:author)* :Paul_Erdos . }
```

- Posts recommended to Julie but not to John

```
SELECT ?x
WHERE {
  {
    { :Julie :knows+/:likes ?x . ?x :hasTag/:hasFollower :Julie . }
     MINUS
    { :John :knows+/:likes ?x . ?x :hasTag/:hasFollower :John . }
}
```

SPARQL: limited form of negation




```
SELECT ?y
WHERE { :Clint_Eastwood (!{:type,:directs})* ?y }
```

- Will match :Unforgiven (IRI), “Unforgiven” (string) and :Movie (IRI)
- Called “negated property sets”

Cypher

- Does not support full regular expressions, but allows transitive closure over a single edge label or edge property/value pair

```
MATCH (x1:Person) -[:knows*]-> (x2:Person)
RETURN x1,x2
```

- Uses no-repeated-edge semantics for cngps and *bag* semantics

```
MATCH (x1) -[*]-> (x2)
RETURN x1,x2
```

What would this return?

Cypher: shortest and bounded paths

- Returns a single shortest witnessing path (could also use allShortestPaths to get all shortest paths)

```
MATCH ( julie:Person {firstname:"Julie"} ),
p = shortestPath( (julie) -[:knows*]-> (x:Person) )
RETURN p
```

- Star operator on edge label - can also specify bounds on the length (e.g. [:knows*2..7]

```
MATCH (x1:Person firstName:"Julie") -[:knows*]-> (x2:Person)
MATCH (x2) -[:likes]-> () -> [:hasTag] -> (x3)
MATCH (x3) -[:hasFollower]-> (x1)
RETURN x2
```

Ok, but what about aggregation?

We'll take a look at Cypher.

Cypher and aggregation

Common aggregation functions are supported: count, sum, avg, min, and max

```
MATCH (p:Person)
RETURN count(*) as headcount;
```

"headcount"
"145"

```
MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)<-[:DIRECTED]-(director:Person)
RETURN actor,director,count(*) AS collaborations
```

"actor"	"director"	"collaborations"
{"born":"1946","name":"Susan Sarandon"}	{"born":"1965","name":"Lana Wachowski"}	"1"
{"born":"1960","name":"Annabella Sciorra"}	{"born":"1956","name":"Vincent Ward"}	"1"
{"born":"1956","name":"Tom Hanks"}	{"born":"1951","name":"Robert Zemeckis"}	"2"
{"born":"1953","name":"David Morse"}	{"born":"1959","name":"Frank Darabont"}	"1"

Cypher: beyond min, max, sum, count

- ***Collect()*** function collects all aggregated values into a list

```
MATCH (m:Movie)<-[:ACTED_IN]-(a:Person)
RETURN m.title AS movie, collect(a.name) AS cast, count(*) AS actors
```

"movie"	"cast"	"actors"
"You've Got Mail"	["Dave Chappelle","Parker Posey","Steve Zahn","Meg Ryan","Tom Hanks","Greg Kinnear"]	"6"
"Apollo 13"	["Tom Hanks","Kevin Bacon","Ed Harris","Bill Paxton","Gary Sinise"]	"5"
"Johnny Mnemonic"	["Dina Meyer","Takeshi Kitano","Ice-T","Keanu Reeves"]	"4"
"Stand By Me"	["Marshall Bell","Kiefer Sutherland","John Cusack","Corey Feldman","Jerry O'Connell","River Phoenix","Wil Wheaton"]	"7"
"The Polar Express"	["Tom Hanks"]	"1"

Cyper: WITH Example

```
MATCH (person:Person)-[:ACTED_IN]->(m:Movie)
WITH person, count(*) AS appearances, collect(m.title) AS movies
WHERE appearances > 1
RETURN person.name, appearances, movies
```

"person.name"	"appearances"	"movies"
"Cuba Gooding Jr."	"4"	["A Few Good Men","Jerry Maguire","As Good as It Gets","What Dreams May Come"]
"Oliver Platt"	"2"	["Frost/Nixon","Bicentennial Man"]
"Philip Seymour Hoffman"	"2"	["Twister","Charlie Wilson's War"]
"Sam Rockwell"	"2"	["The Green Mile","Frost/Nixon"]
"Greg Kinnear"	"2"	["As Good as It Gets","You've Got Mail"]
"Zach Grenier"	"2"	["RescueDawn","Twister"]
"Rosie O'Donnell"	"2"	["A League of Their Own","Sleepless in Seattle"]

Source: https://neo4j.com/developer/cypher-query-language/

Summary

- SPARQL is the W3C recommended language for querying RDF graphs, and is based on an edge-labeled graph model.
  - Supports all complex graph patterns
  - Homomorphism-based bag semantics
  - Allows more than regular path queries (e.g. inverse)
  - Output can be boolean or nodes
- Cypher is the query language for Neo4j, and is based on a property graph model.
  - Supports all complex graph patterns
  - No-repeated-edge bag semantics
  - Allows only a fragment of regular path queries (repeated label)
  - Output can be boolean/nodes/paths/graphs