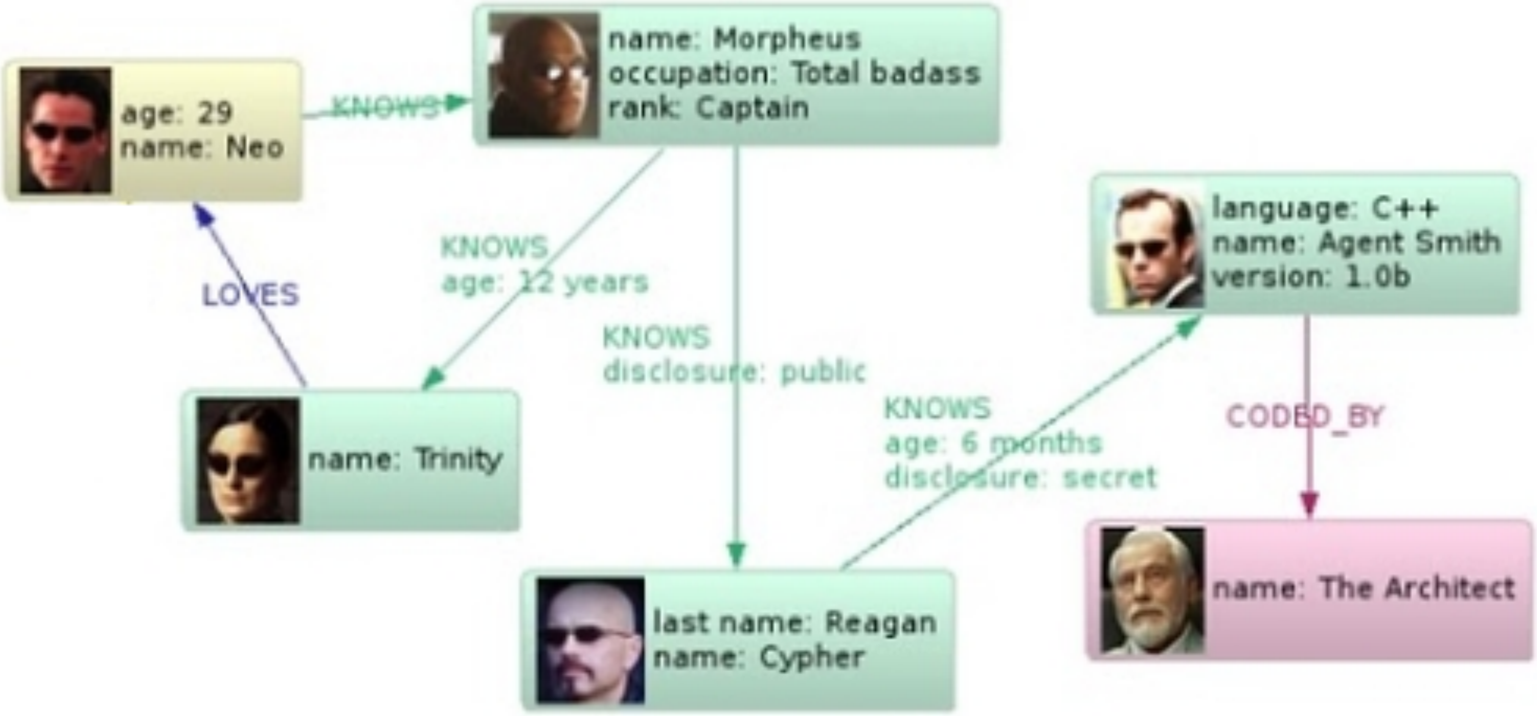
# Non-Standard-Datenbanken

## Graphdatenbanken

Prof. Dr. Ralf Möller
Universität zu Lübeck
Institut für Informationssysteme

Dennis Heinrich (Übungen)

# Graph Database, think different!




- Nodes
- Edges (directed or not)
- Properties

# A general query structure

MATCH [Nodes and relationships]

WHERE [Boolean filter statement]

RETURN [DISTINCT] [statements [AS alias]]

ORDER BY [Properties] [ASC\DESC]

SKIP [Number] LIMIT [Number]

# First query

Get all nodes of type *Program* that have the name *Hello World!*:

MATCH (a : Program)

WHERE a.name = 'Hello World!'

RETURN a

# Query relationships

Get all relationships of type *Author* connecting *Programmers* and *Programs*:


MATCH (a : Programmer)-[r : Author]->(b : Program)

RETURN r

# Matching nodes and relationships

- Nodes:

  (a), (), (:Ntype), (a:Ntype), (a { prop:'value' } ) ,
  (a:Ntype { prop:'value' } )

- Relationships:

  (a)--(b), (a)-->(b), (a)<--(b), (a)-->(), (a)-[r]->(b),
  (a)-[:Rtype]->(b), (a)-[:R1|:R2]->(b),
  (a)-[r:Rtype]->(b)

- May have more then 2 nodes:

  (a)-->(b)<--(c), (a)-->(b)-->(c)

- Path:

  p = (a)-->(b)

# More options:

- Relationship distance:

  (a)-[:Rtype*2]->(b) – 2 hops of type Rtype.

  (a)-[:Rtype* ]->(b) – any number of hops of type Rtype.

  (a)-[:Rtype*2..10]-> (b) – 2-10 hops of Rtype.

  (a)-[:Rtype* ..10]-> (b) – 1-10 hops of Rtype.

  (a)-[:Rtype*2.. ]-> (b) – at least 2 hops of Rtype.

  Could be used also as:

  (a)-[r*2]->(b) – r gets a sequence of relationships

  (a)-[*{prop:val}]->(b)

# Operators

- Mathematical

  +, -, *, /,%, ^ (power, not XOR)
- Comparison

  =,<>,<,>,>=,<=, =~ (Regex), IS NULL , IS NOT NULL
- Boolean

  AND, OR, XOR, NOT
- String

  Concatenation through +
- Collection

  Concatenation through +

  IN to check is an element exists in a collection.

# More WHERE options

- WHERE others.name IN ['Andres', 'Peter']
- WHERE user.age IN range (18,30)
- WHERE n.name =~ 'Tob.*‘
- WHERE n.name =~ '(?i)ANDR.*‘ - (case insensitive)
- WHERE (tobias)-->()
- WHERE NOT (tobias)-->()
- WHERE has(b.name)
- WHERE b.name? = 'Bob'

  (Returns all nodes where name = 'Bob' plus all nodes without a name property)

# Functions:

- On paths:
  - MATCH shortestPath( (a)-[*]-(b) )
  - MATCH allShorestPath( (a)-[*]-(b) )
  - Length(path) – The path length or 0 if not exists.
  - RETURN relationships(p) - Returns all relationships in a path.

- On collections:
  - RETURN a.array, filter(x IN a.array WHERE length(x)= 3)

    FILTER - returns the elements in a collection that comply to a predicate.
  - WHERE ANY (x IN a.array WHERE x = "one“ ) – at least one
  - WHERE ALL (x IN nodes(p) WHERE x.age > 30) – all elements
  - WHERE SINGLE (x IN nodes(p) WHERE var.eyes = "blue") – Only one

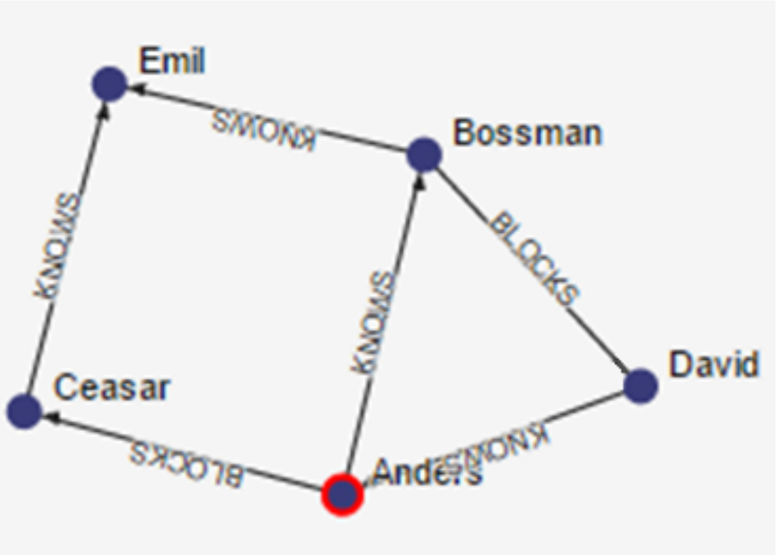
  * nodes(p) – nodes of the path p

# With

- Manipulate the result sequence before it is passed on to the following query parts.
- Usage of WITH :
  - Limit the number of entries that are then passed on to other MATCH clauses.
  - Introduce aggregates which can then be used in predicates in WHERE.
  - Separate reading from updating of the graph. Every part of a query must be either read-only or write-only.

# With

```
MATCH (david { name: "David" })--(otherPerson)-->()
WITH otherPerson, count(*) AS foaf
WHERE foaf > 1
RETURN otherPerson
```




What will be returned?

The person connected to David with the at least one outgoing relationship.

(2 {name:"Anders"})

* foaf = 2

# More collections options

- MATCH (user)

  RETURN count(user)
- MATCH (user)

  RETURN count(DISTINCT user.name)
- MATCH (user)

  RETURN collect(user.name)

  Collection from the values, ignores NULL.
- MATCH (user)

  RETURN avg(user.age)

  Average numerical values. Similar functions are sum, min, max.

# Additional Functionality: Mining the link structure

- **Centrality** (who are the most important nodes?)
- **Reach/Influence:** coverage of one or more nodes (facility location, influence maximization)
- **Similarity** of node pairs (link prediction, targeted ads, friend/product recommendations, attribute completion)
- **Communities**: set of nodes that are more tightly related to each other than to others

# Computing on Very Large Graphs

- Many applications, platforms, algorithms
- **Clusters** (Map Reduce, Hadoop) when applicable
- **iGraph/Pregel** better with edge traversals
  - http://igraph.org Pregel is from Google (see also *Apache Giraph)*
- (Semi-)**streaming** : pass(es), keep small info (per-node)

- **General algorithm design principles** :
  - settle for approximations
  - keep total computation/ communication/ storage "linear" in the size of the data
  - Parallelize (minimize chains of dependencies)
  - Localize dependencies



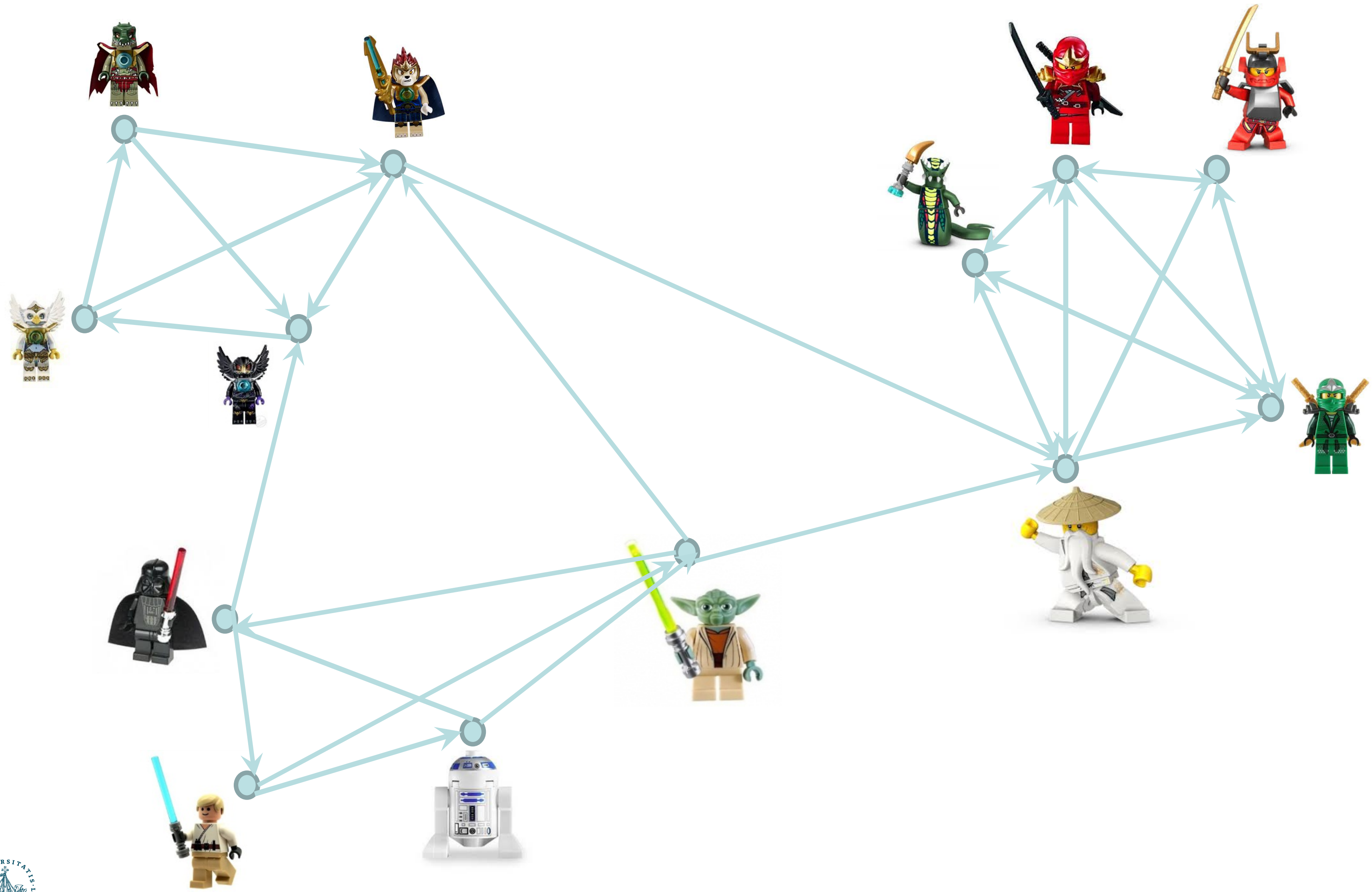
Grzegorz Malewicz et al., Pregel: A System for Large-Scale Graph Processing, Google, **2010**

# Node sketches

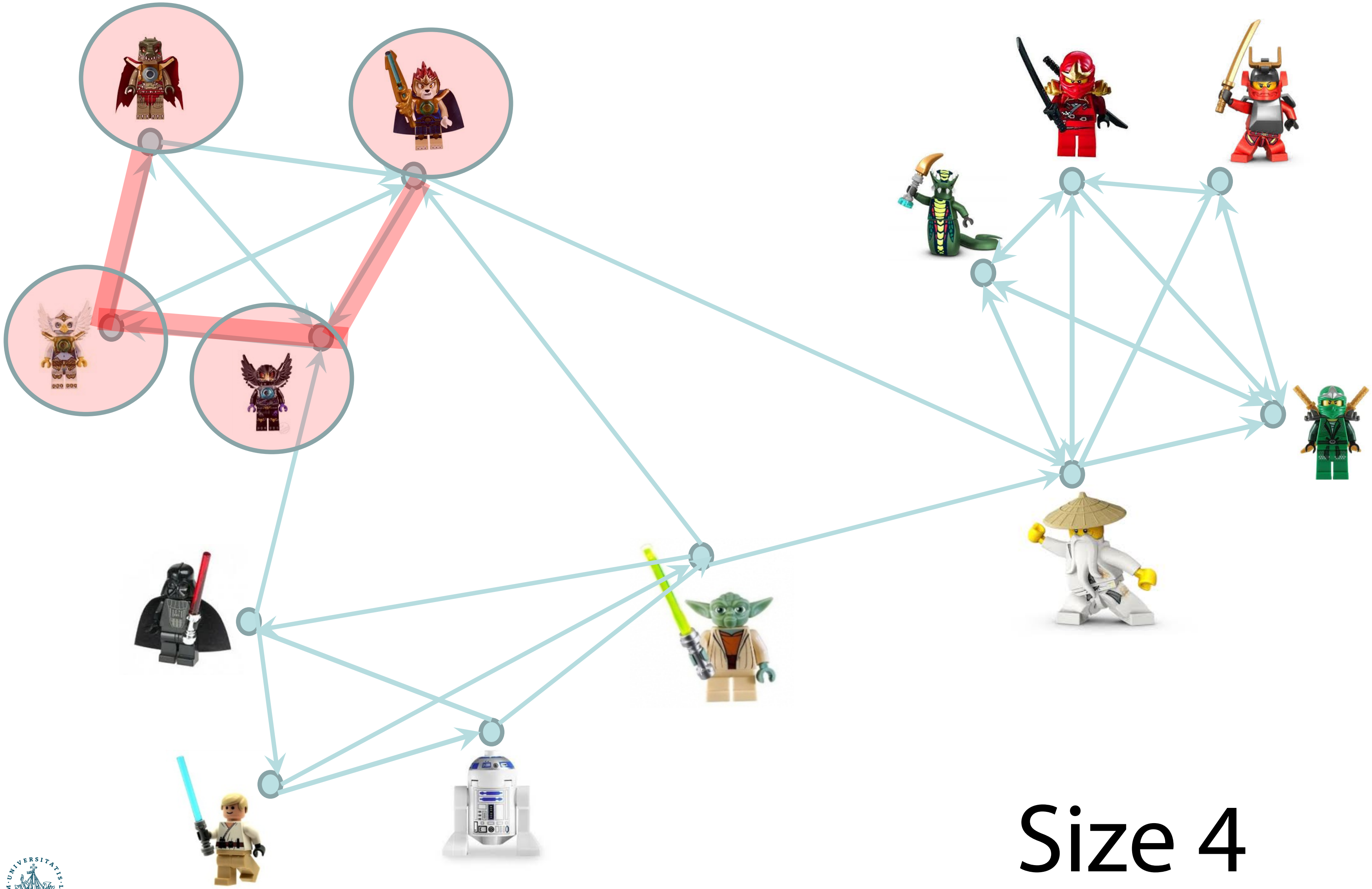
## MinHash sketches of reachability sets

- Compute a sketch for each node, <u>efficiently</u>
- From sketch(es) can estimate properties that are "harder" to compute exactly

# Sketching Reachability Sets

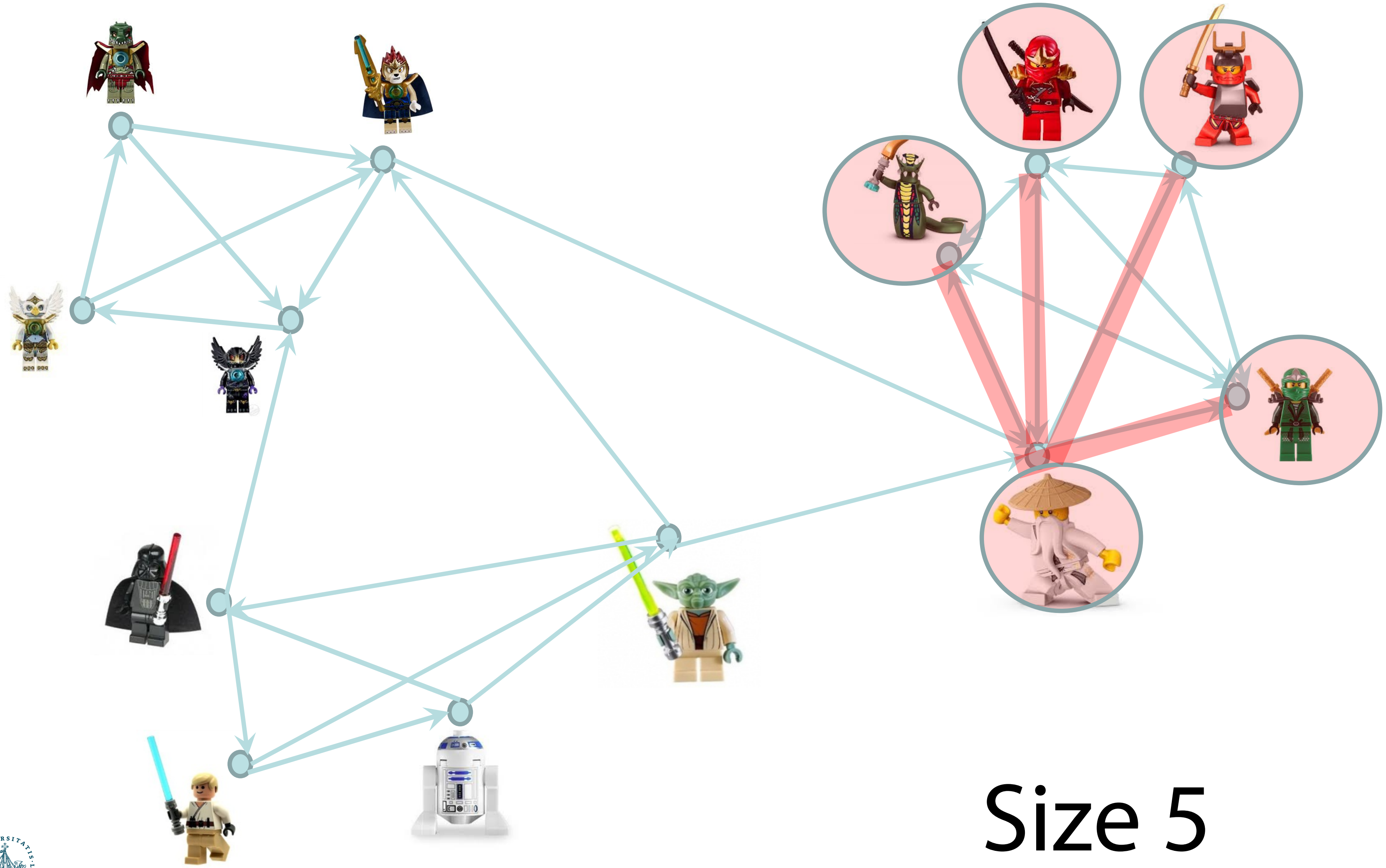


# Reachability Set of 



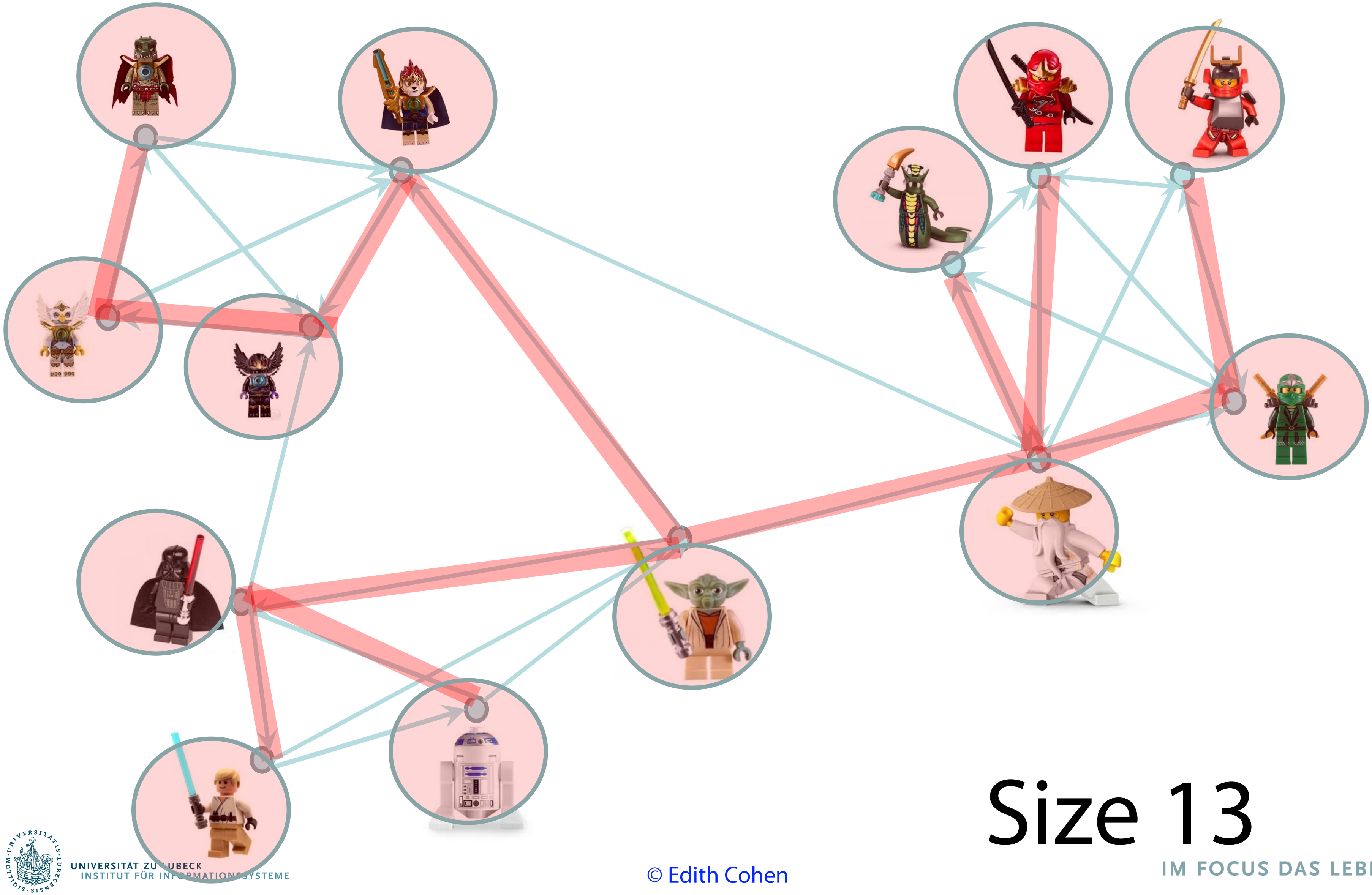
Size 4

# Reachability Set of 



Size 5

# Reachability Set of



Size 13

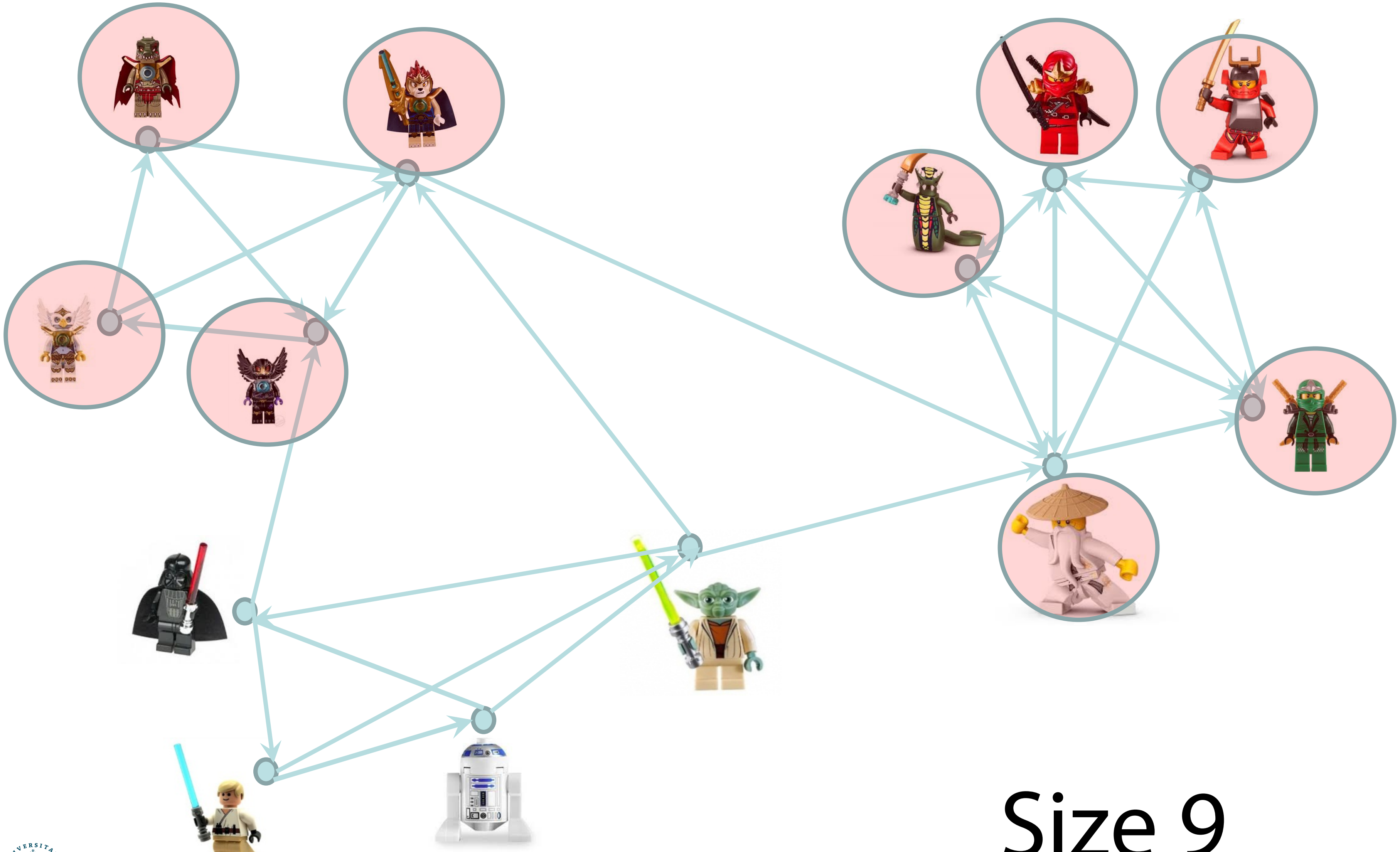
# Why sketch reachability sets ?

From reachability sketch(es) we can:

- Estimate cardinality of reachability/influence set of one or more nodes
- Get a uniform sample of the reachable nodes
- Estimate similarity of nodes by relations between reachability sets (e.g., Jaccard similarity)

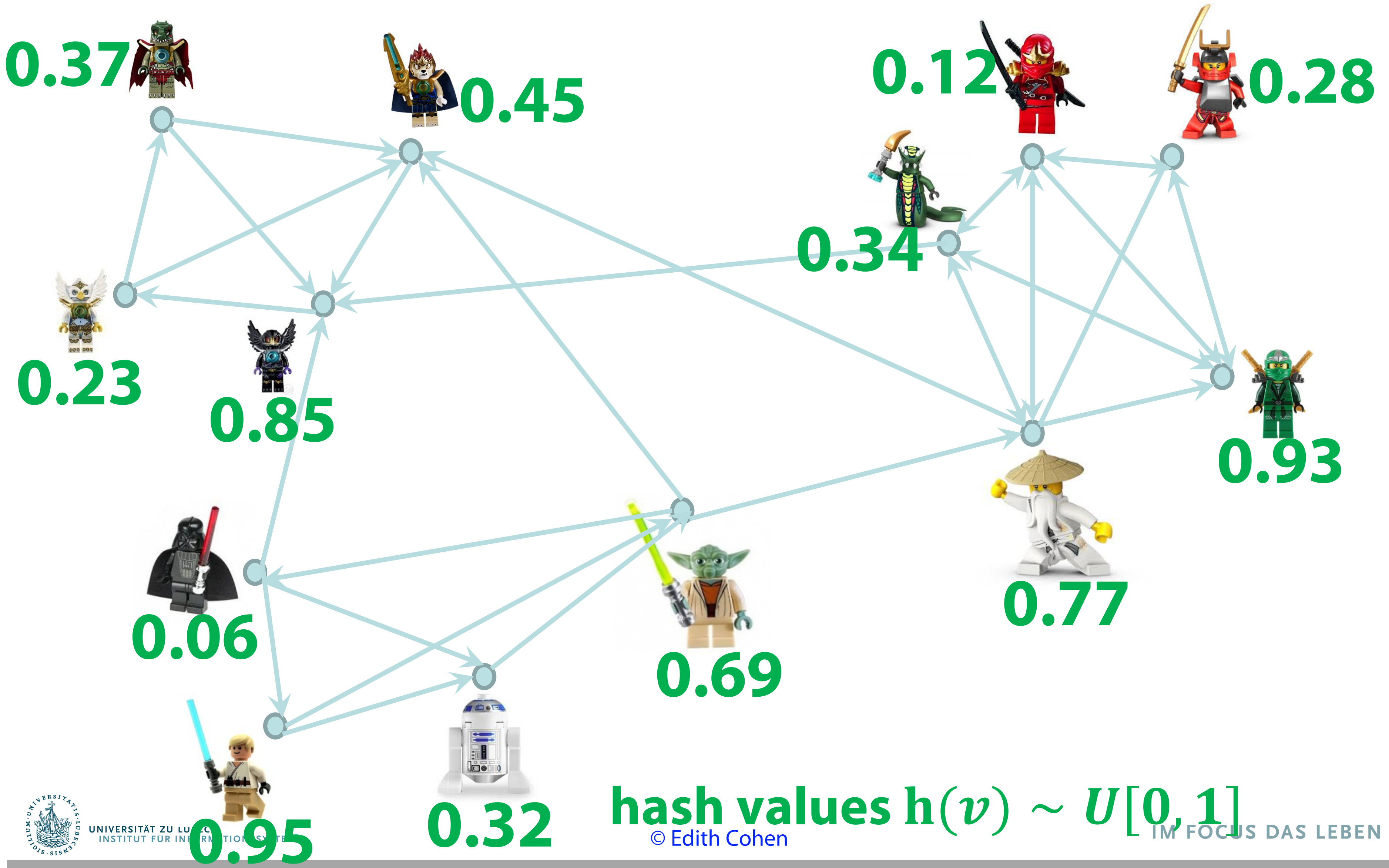
➢ Exact **computation** is *costly*: $O(mn)$ with $n$ nodes and $m$ edges, **representation size** is massive: does not scale to large networks!

# Influence of ∪



Size 9

# MinHash sketches of all Reachability sets




**hash values $h(v) \sim U[0,1]$**

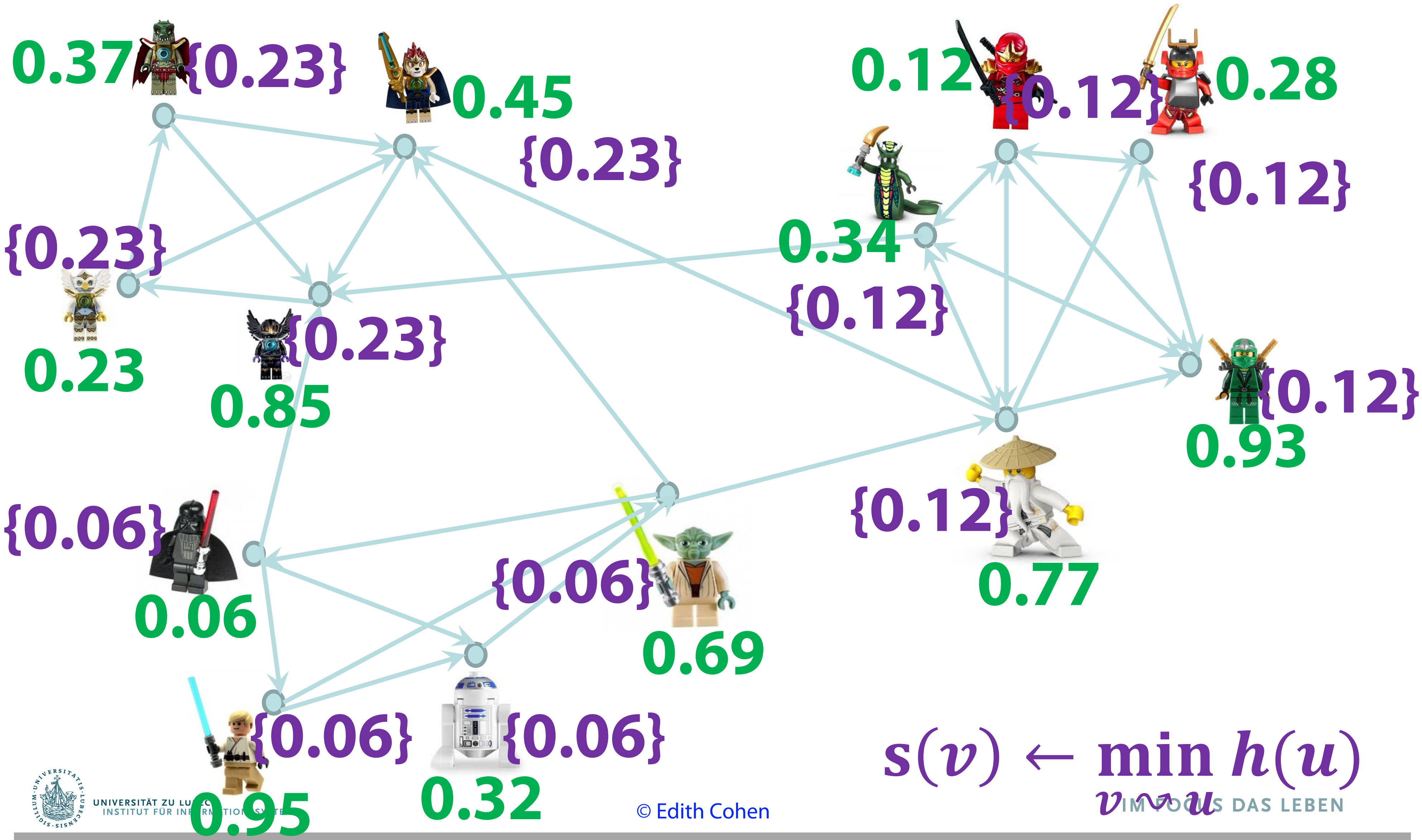
# MinHash sketches of all Reachability Sets: $k = 1$

For each $v$: $\mathbf{s}(v) \leftarrow \min_{v \rightsquigarrow u} h(u)$

Depending on application, may also want to include node ID in sketch:

$$\operatorname{argmin}_{v \rightsquigarrow u} h(u)$$

# MinHash sketches of all Reachability Sets: $k = 1$




$$s(v) \leftarrow \min_{v \rightsquigarrow u} h(u)$$

# Computing MinHash Sketches of all Reachability Sets: $k=1$ BFS method

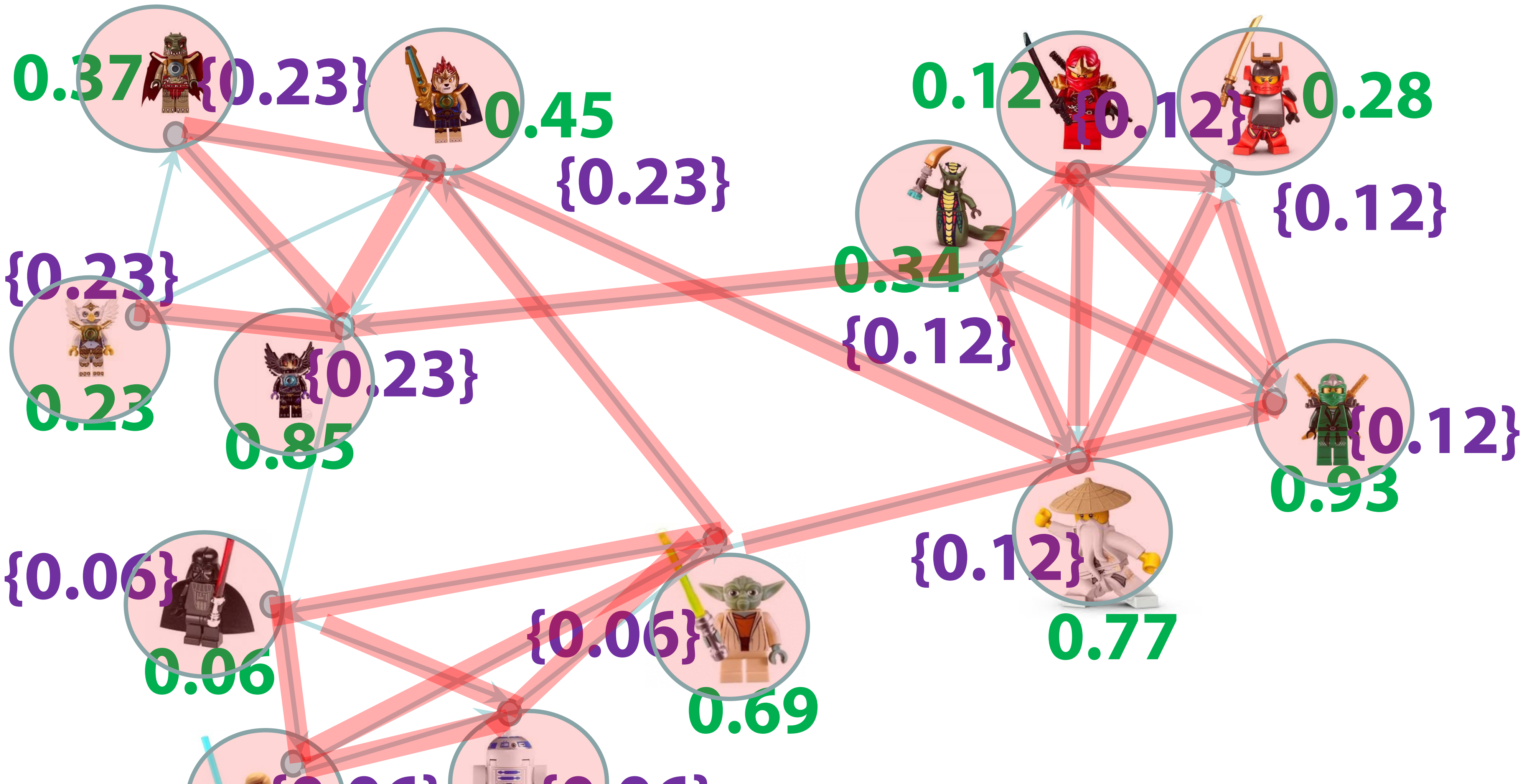
$$\mathbf{s}(v) \leftarrow \min_{v \rightsquigarrow u} h(u)$$

Iterate over nodes $u$ by increasing $h(u)$:

Visit nodes $v$ through a reverse search from $u$:

- **IF** $s(v) = \emptyset$,
  - $s(v) \leftarrow h(u)$
  - Continue search on $\text{inNeighbors}(v)$
- **ELSE**, truncate search at $v$

# Compute MinHash sketches of all Reachability Sets: $k = 1$, BFS




$$s(v) \leftarrow \min_{v \rightsquigarrow u} h(u)$$

# Computing MinHash sketches of all reachability sets: $k = 1$ BFS method

Analysis:

Each arc is used exactly once $O(m)$, where $m$ is the number of arcs

Note: sorting of nodes is not needed.

Random permutation of $n$ nodes can be generated in $O(n)$ time