Jans Aasman, ISWC 2007

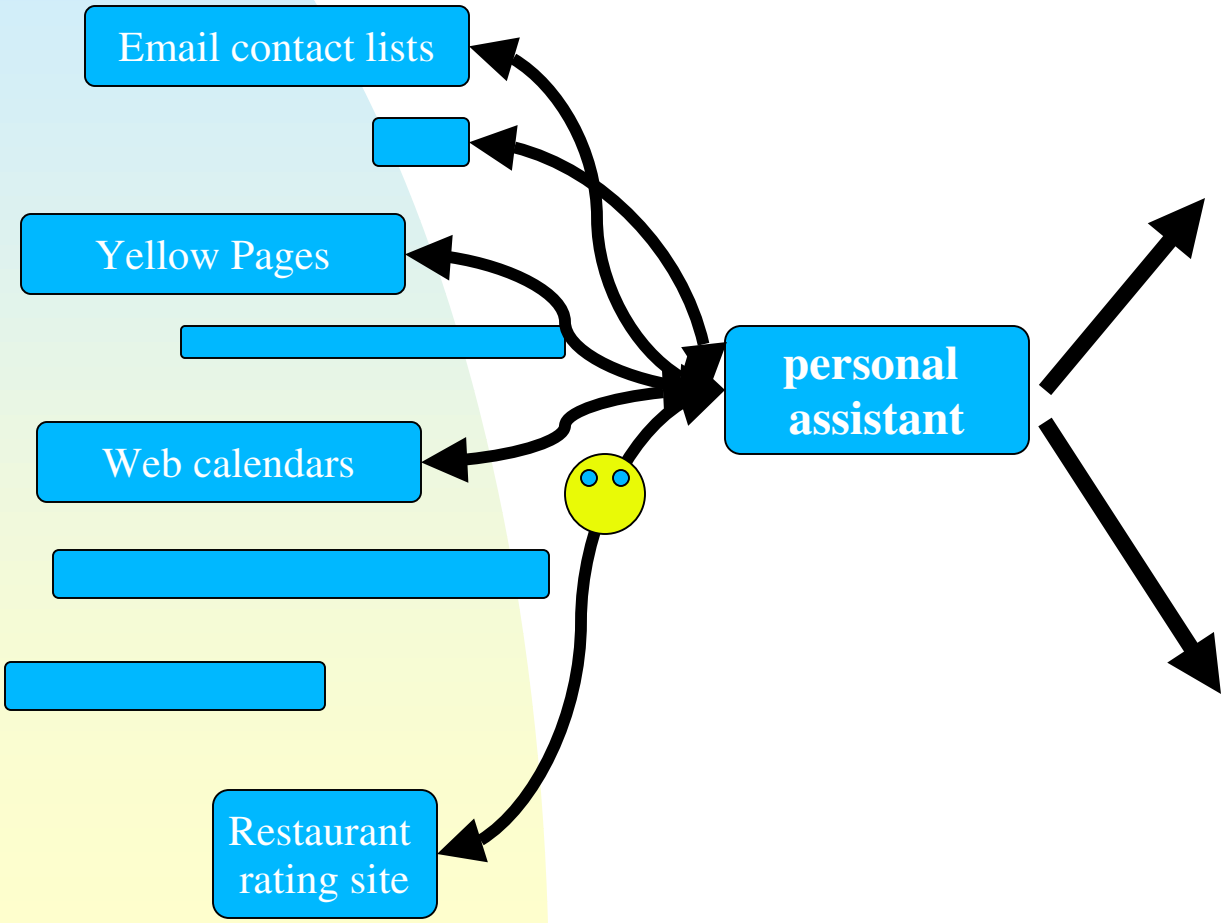
Using Social Network Analysis, Geotemporal Reasoning, and RDFS++ Reasoning for Business Intelligence

A tutorial covering Social Network Analytics, Geospatial and temporal analytics and Reasoning with a general graph database

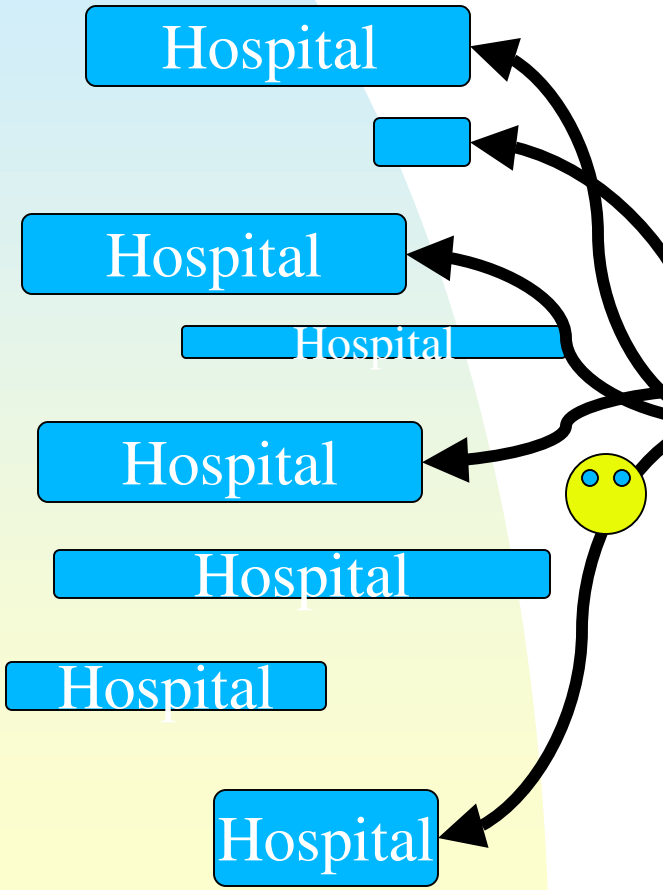
Purpose of the Semantic Web

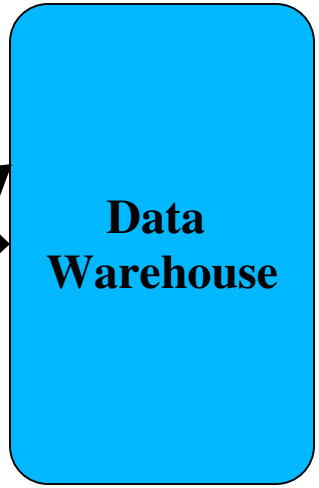
- Use distributed knowledge to create new types of analytics and build new services
  - One view is about having agents combine many small knowledge stores into useful new services
  - Another view is about having huge data stores with semi unstructured and networked data

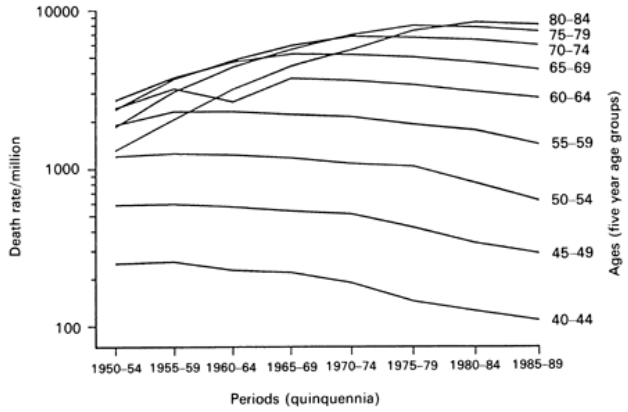
View 1: Your personal assistant will use distributed knowledge stores to plan a party for a group of friends according to their preferences and availability



View 2: The University of Texas unifies data from many hospitals for advanced data analytics and new services for the Government and the public

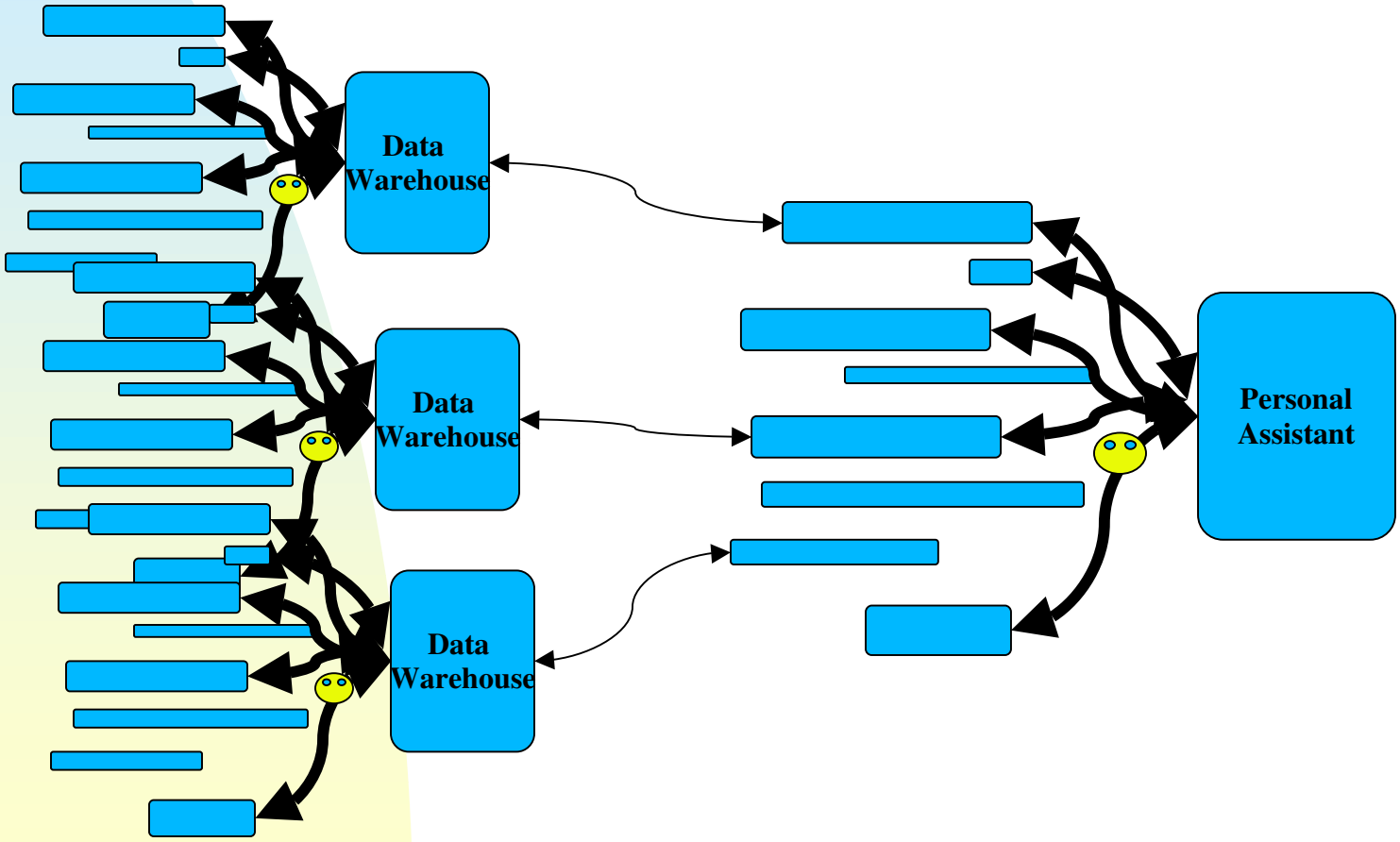






November, 2007

But those views are not really different, they live in the same eco system of services and knowledge stores



Our customers are in the data warehouse mode. Questions they ask

- Where did the people with this class of diseases come from, what kind of events happened before that in that area, what do we know about causal connections?
- With whom does this person/company have relationships? How is this person/company related to another person/company through a set of relationships? Can we distinguish subgroups that he lives in? How do these subgroups relate to each other, etc.?

The questions revolve around

- Class hierarchy reasoning
  - Classes of entities: people, companies, products, processes, documents, events, social phenomena.
- Events in time and space
  - Using basic temporal primitives
  - Using basic geospatial primitives
- Transactions between entities
  - From, to, entities involved, type, place, time
- Relationships between entities
  - Using social network analytic primitives
- Using information in natural text to augment all of the above: -> more structured RDF

Contents of this tutorial

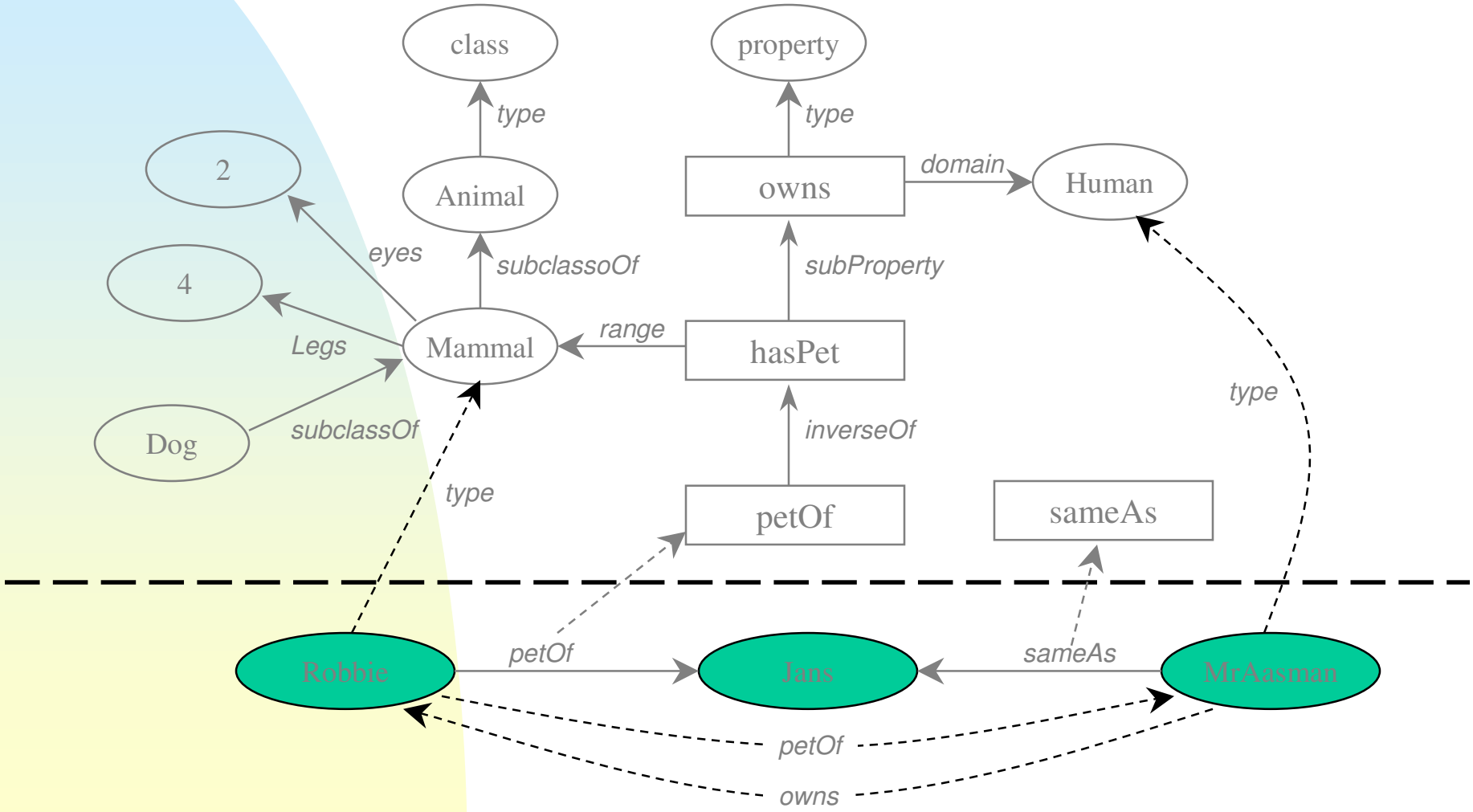
- RDF & triple stores in a few pages
- Difference between triple stores and RDBMS
- Triple Store Requirements
- Basic Social Network Analytics primitives
- Basic Geospatial primitives
- Temporal reasoning primitives
- Text indexing
- Combining it all in SPARQL and Prolog
- Hands on & demos

RDF: Resource Description Framework

- W3C's knowledge representation standard for the semantic web
- Semantic web is basically the Web 3.0
  - Metadata for content (web pages, multimedia contents, versioning) allows machines to help people search information and organize their lives
- Quickly became standard for metadata in general

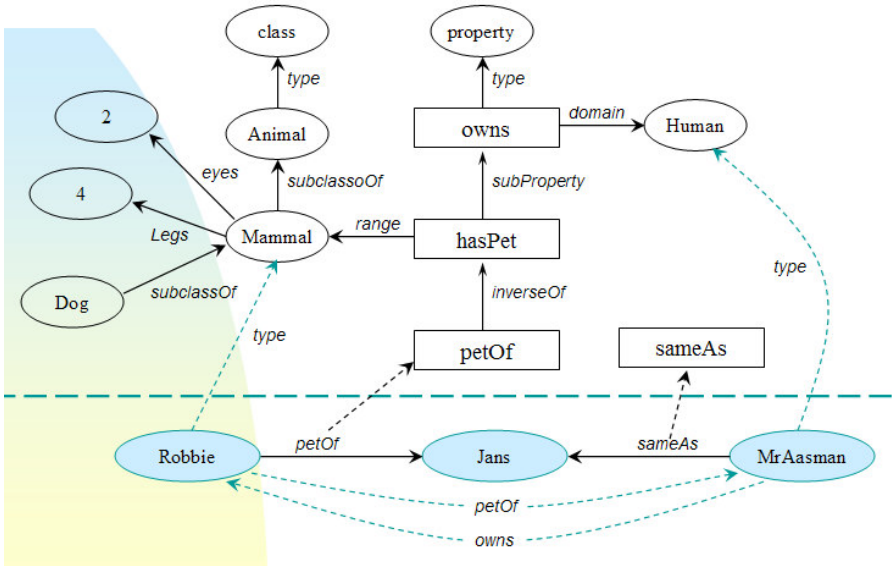
- But: nothing more than a way to serialize old-fashioned semantic networks

A typical semantic network from the late sixties



The same network serialized into triples

Subject	Predicate	Object
Animal	type	class
Mammal	subclassOf	Animal
Mammal	eyes	2
Mammal	legs	4
Dog	subclassOf	Mammal
owns	type	Property
owns	domain	Human
hasPet	subproperty	owns
hasPet	range	Mammal
hasPet	inverseOf	petOf
Robbie	petOf	Jans
MrAasman	sameAs	Jans



RDF: Subject, predicate, object turned into Resources (URIs) and literals

```
<http://www.franz.com/simple#Animal>
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   <http://www.w3.org/2002/07/owl#Class>.

<http://www.franz.com/simple#Mammal>
   <http://www.w3.org/2000/01/rdf-schema#subClassOf>
   <http://www.franz.com/simple#Animal> .

<http://www.franz.com/simple#Mammal>
   <http://www.franz.com/simple#eyes> "two" .

<http://www.franz.com/simple#Mamma>
   <http://www.franz.com/simple#legs> "four" .

<http://www.franz.com/simple#Dog> <http://www.w3.org/1999/02/22-
   rdf-syntax-ns#type> <http://www.franz.com/simple#Mammal> .

<http://www.franz.com/simple#owns>
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   <http://www.franz.com/simple#property> .

<http://www.franz.com/simple#owns>
   <http://www.w3.org/2000/01/rdf-schema#domain>
   <http://www.franz.com/simple#Human> .

<http://www.franz.com/simple#haspet>
   <http://www.franz.com/simple#subproperty>
   <http://www.franz.com/simple#owns> .
```

Resource

literal

Triples in a triple store

Triples are number vectors in memory and on disk.

```
#(2 4 5 6)
#(7 9 2 10)
#(7 11 12 13)
#(14 15 16 17)
#(18 4 7 19)
#(20 4 21 22)
#(20 23 24 25)
#(26 27 20 28)
#(26 29 7 30)
#(31 32 33 34)

.....
```

Dictionary

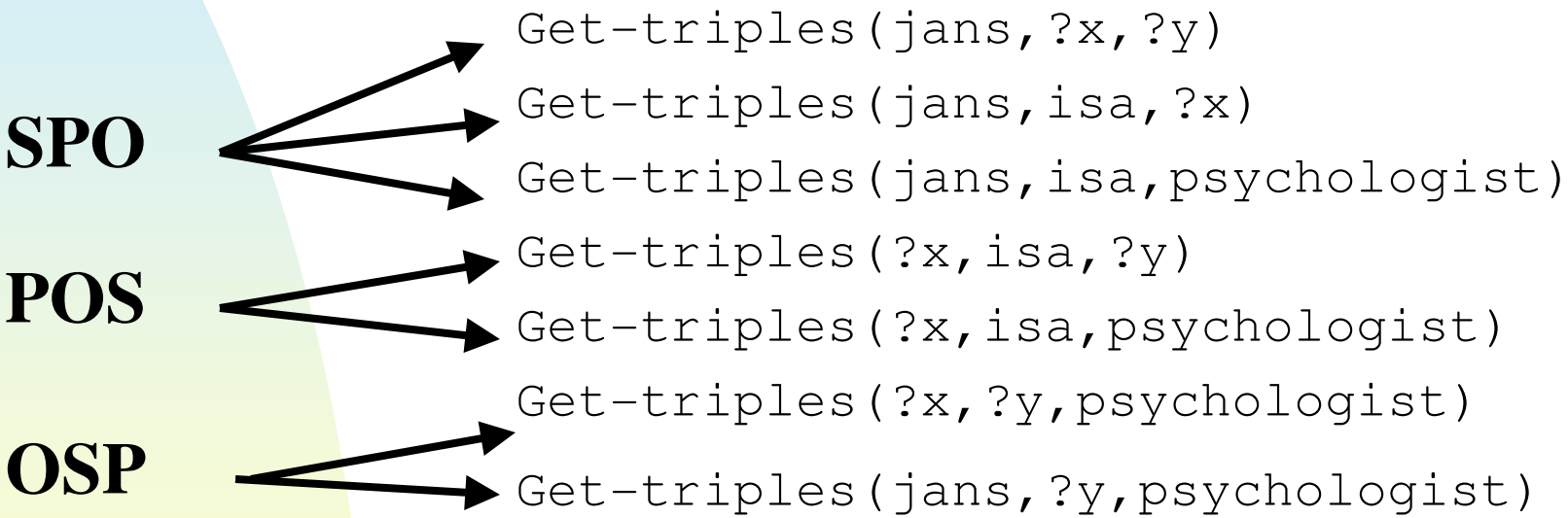
```
franz:Animal = 2
rdf:type = 4
owl:Class = 5
franz:Eyes = 9
.....
```

Reverse Dictionary

```
2 = franz:Animal
4 = rdf:Type
5 = owl:class
6 = Triple-id
7 = franz:Mammal

.....
```

Triples are indexed in three ways



And six ways with named graphs

Contents of this tutorial

- RDF & triple stores in a few pages
- Difference between triple stores and RDBMS
- Triple Store Requirements
- Basic Social Network Analytics primitives
- Basic Geospatial primitives
- Temporal reasoning primitives
- Text indexing
- Combining it all in SPARQL and Prolog
- Hands on & demos

The difference with a relational database?

```
<triple 32: "person2" "type" "person">
<triple 33: "person2" "first-name" "Rose">
<triple 34: "person2" "middle-name" "Elizabeth">
<triple 35: "person2" "last-name" "Fitzgerald">
<triple 36: "person2" "suffix" "none">
<triple 37: "person2" "alma-mater" "Sacred-Heart-Convent">
<triple 38: "person2" "birth-year" "1890">
<triple 39: "person2" "death-year" "1995">
<triple 40: "person2" "sex" "female">
<triple 41: "person2" "spouse" "person1">
<triple 58: "person2" "has-child" "person17">
<triple 56: "person2" "has-child" "person15">
<triple 54: "person2" "has-child" "person13">
<triple 52: "person2" "has-child" "person11">
<triple 50: "person2" "has-child" "person9">
<triple 48: "person2" "has-child" "person7">
<triple 46: "person2" "has-child" "person6">
<triple 44: "person2" "has-child" "person4">
<triple 42: "person2" "has-child" "person3">
<triple 60: "person2" "profession" "home-maker">
```

An artist's impression of the same info in an RDBMS

Table Person

ID	First-Name	Last-Name	Middle-In.	DOB	DOD	PlaceOB	Sex
2	Rose	Fitzgerald	Elizabeth	1890	1995	1	F

Table Spouses

ID1	ID2
2	1

Table to-schools

ID1	SchoolID
2	3

Table Schools

ID	Name
3	Sacred-Heart

Table has-profession

ID1	ProfID
2	3

Table Professions

ID	Name
3	Home-maker

Table Has-Child

ID1	ID2
2	17
2	15
2	14
2	13

Table Place

ID	Name	State	Longitude	Latitude
1	Boston	MA	42.3	-71.4

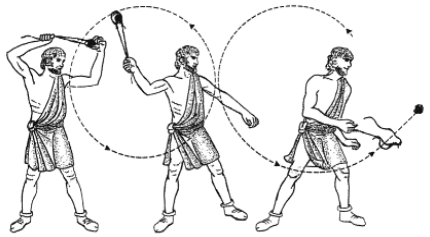
With a graph database

- You add new predicates without changing any schema
- One-to-many relations are directly encoded without the indirection of tables
- You never think about what to index because all the predicates are indexed

Contents of this tutorial

- RDF & triple stores in a few pages
- Difference between triple stores and RDBMS
- Triple Store Requirements
- Basic Social Network Analytics primitives
- Basic Geospatial primitives
- Temporal reasoning primitives
- Text indexing
- Combining it all in SPARQL and Prolog
- Hands on & demos

Triple Store Requirements



- Scalable and persistent
  - > Billion triples in < a day on affordable hardware
  - In memory too limiting, must persist to disk
- Compliant on standards
  - RDF, RDFS, OWL, SPARQL, Named Graphs
- Two modes of working
  - Standalone for analytics, Client/Server for real time services
- Accessible from any language
  - Java: Sesame and Jena remote repository APIs
  - .Net, Python, Ruby, Lisp, C through REST interface
- Reasoning
  - OWL subsets and full Description Logics
- GUI & Ontology Management

Triple Stores, Additional Requirements

- Triple referencing
  - Triples can point to triples, use for reification
- Named Graphs fully supported
  - Slot can also be used for weights, trust factors, provenance, distance, etc.
- Native data types and efficient range queries
  - With triple stores that store all data as strings, range queries are inefficient
  - Must support most xml schema types (dates, times, longitudes, latitudes, durations, telephone numbers, etc)
- Basic geospatial and temporal primitives
- Social Network Analytics library
- Text indexing
- Federated queries
- Combine it all with Prolog & SPARQL

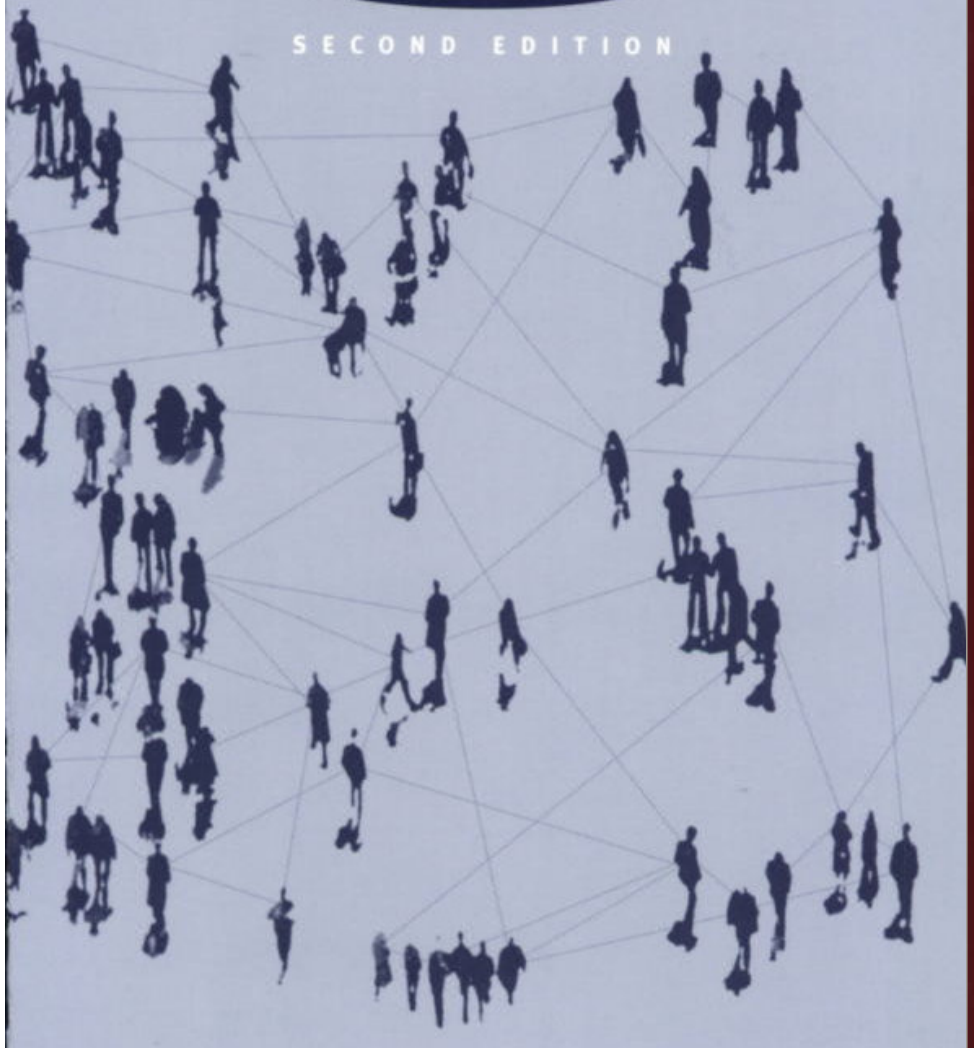
Contents of this tutorial

- RDF & triple stores in a few pages
- Difference between triple stores and RDBMS
- Triple Store Requirements
- Basic Social Network Analytics primitives
- Basic Geospatial primitives
- Temporal reasoning primitives
- Text indexing
- Combining it all in SPARQL and Prolog
- Hands on & demos

Social Network Analysis

a handbook

SECOND EDITION

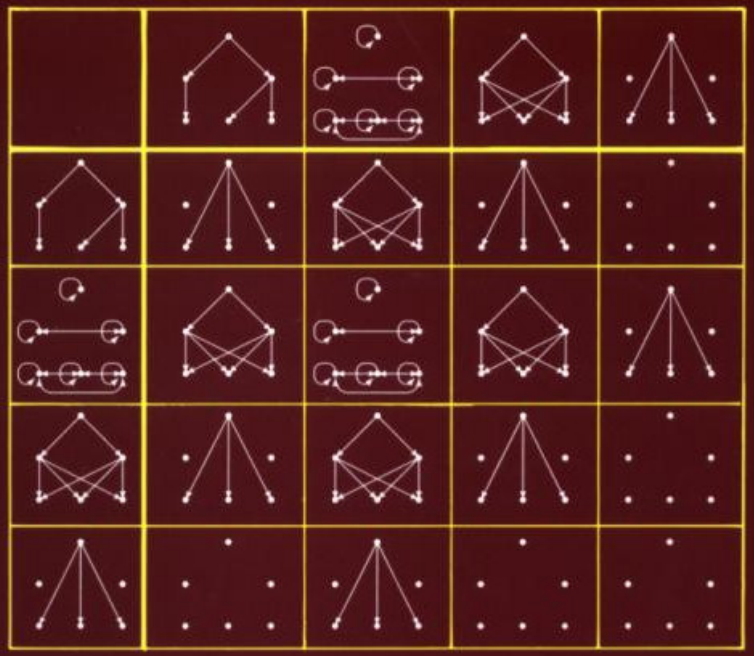


STRUCTURAL ANALYSIS IN THE SOCIAL SCIENCES 8

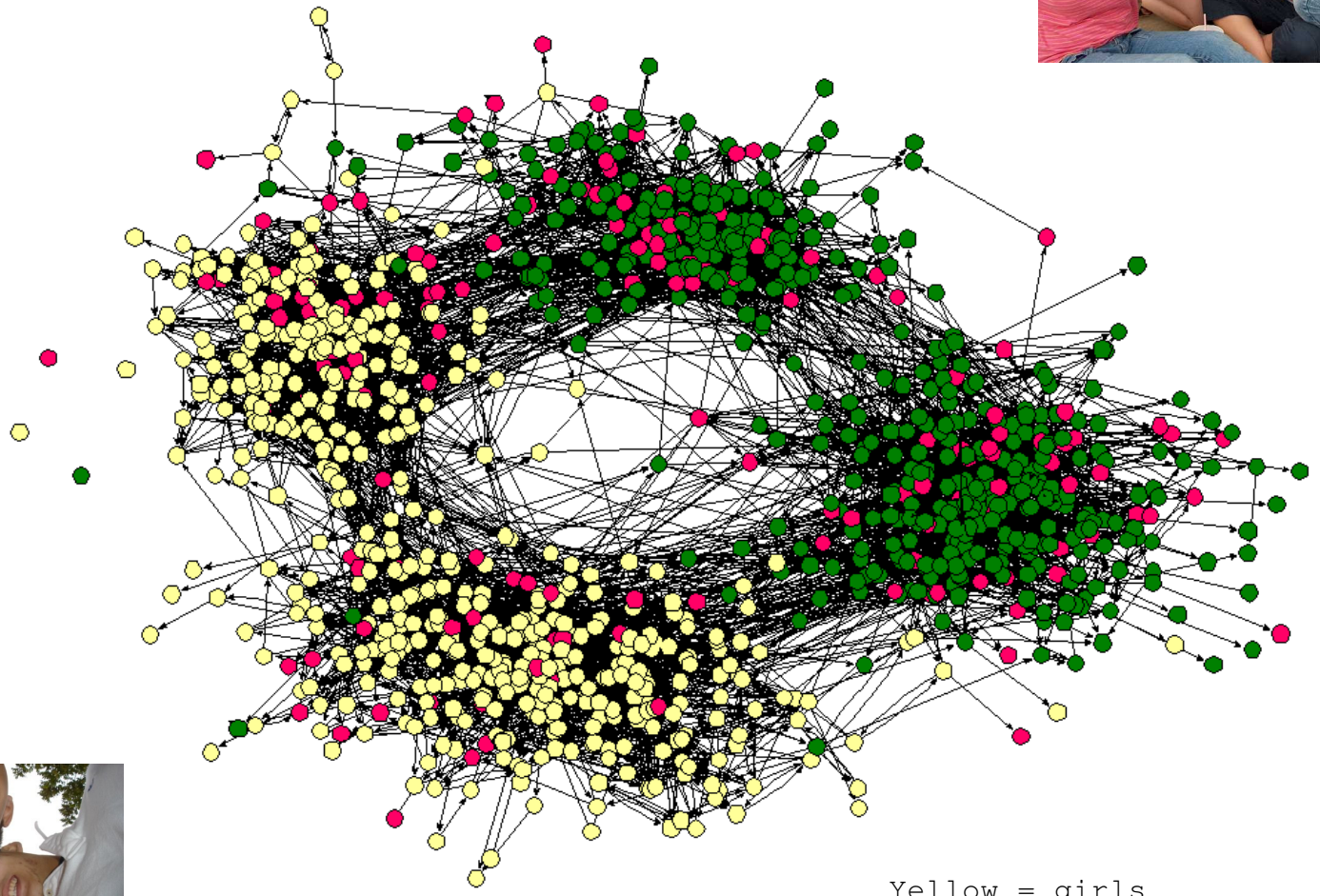
Social Network Analysis

Methods and Applications

Stanley Wasserman and Katherine Faust

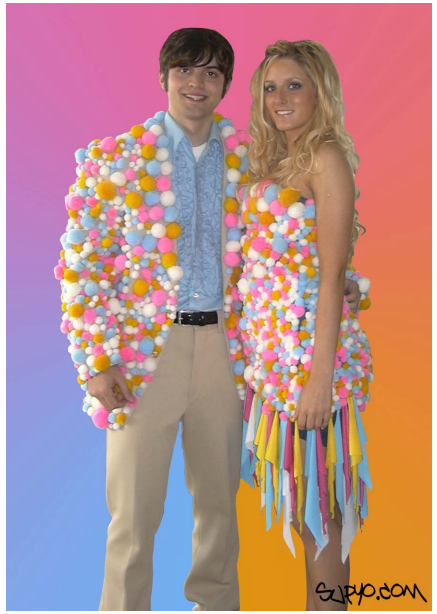


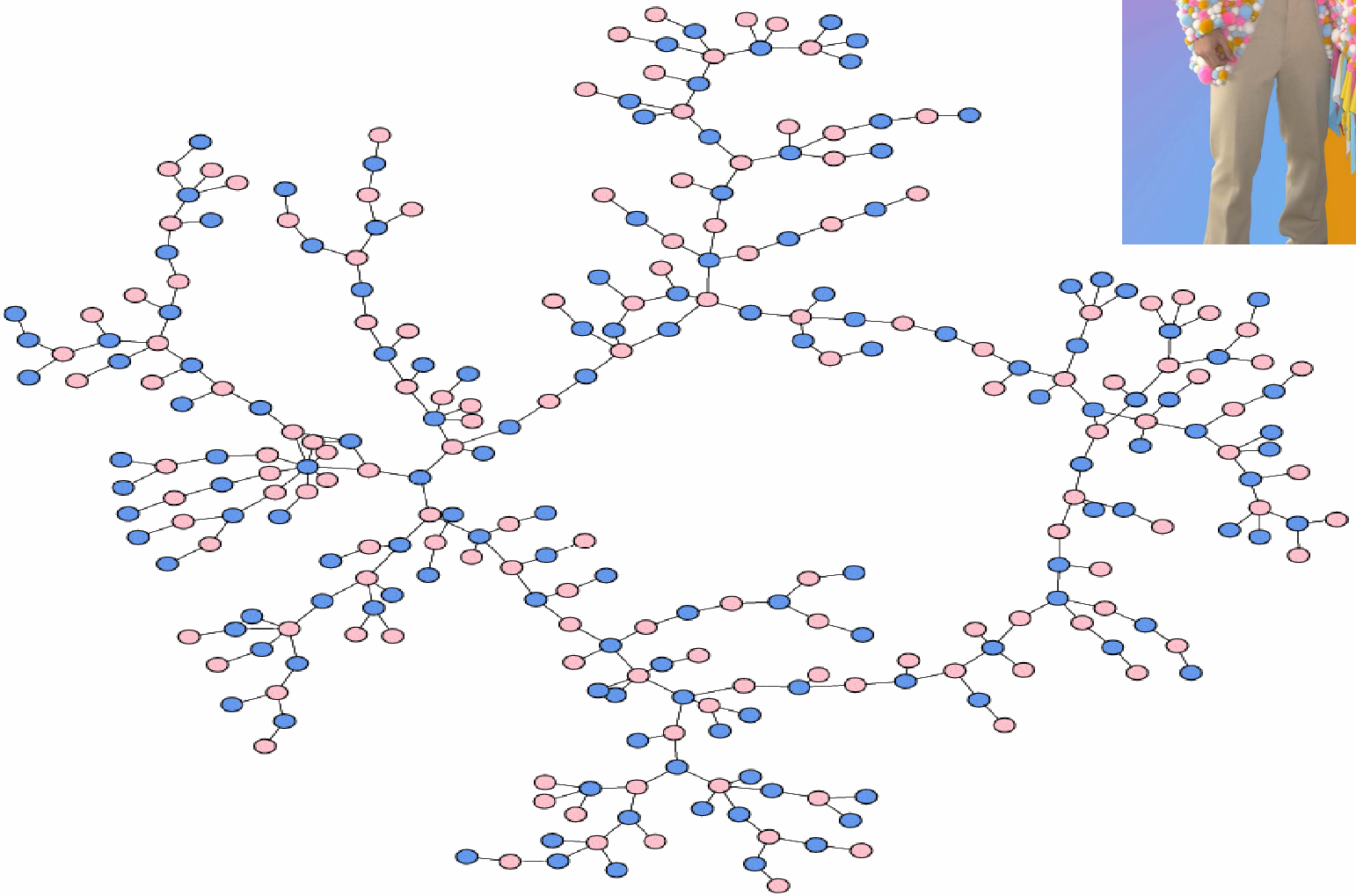
High school friends



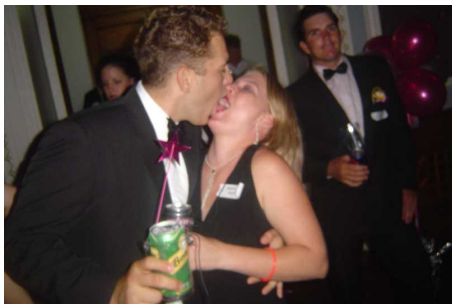
Yellow = girls
Green = boys
Red = sexually active

High-school dating

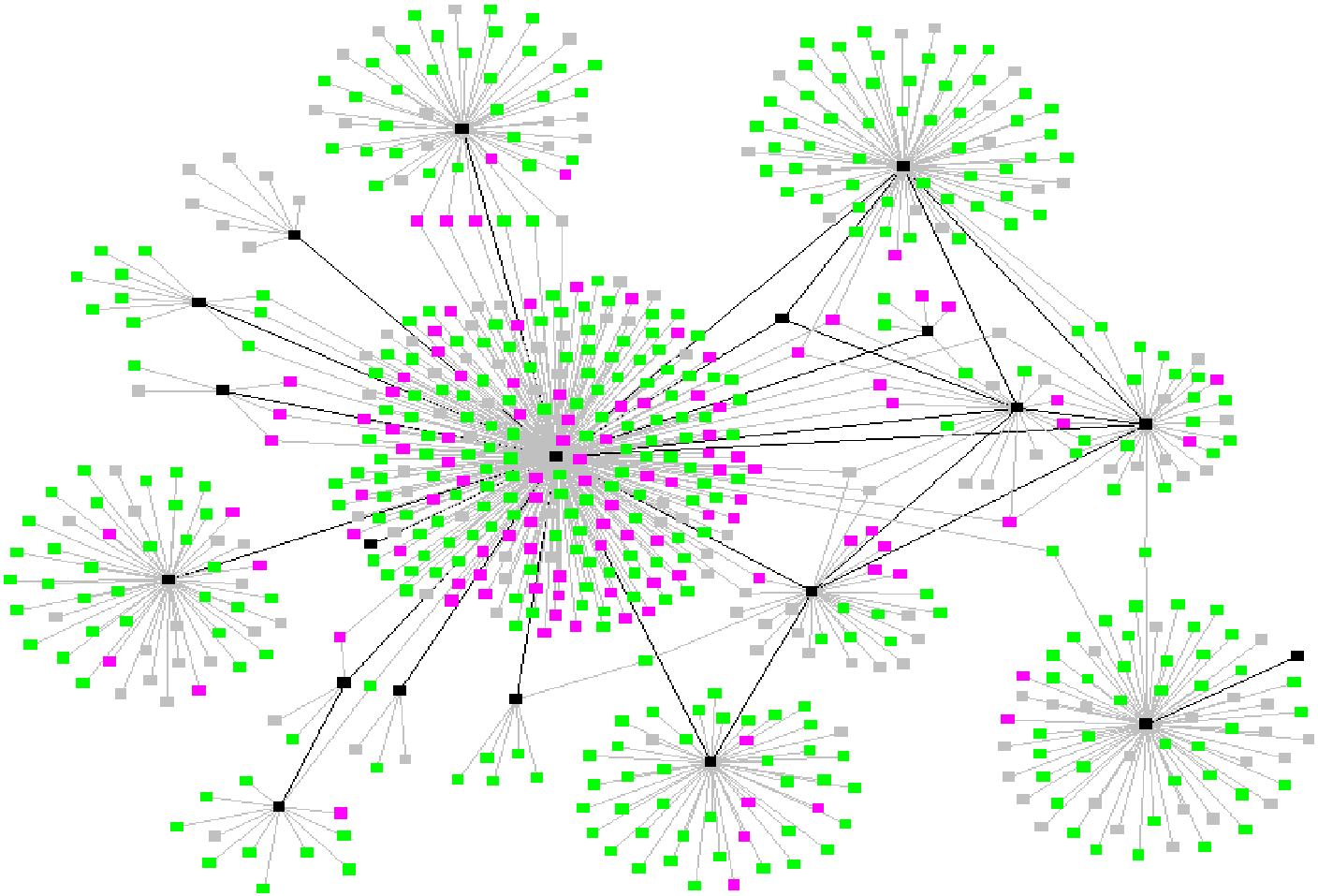




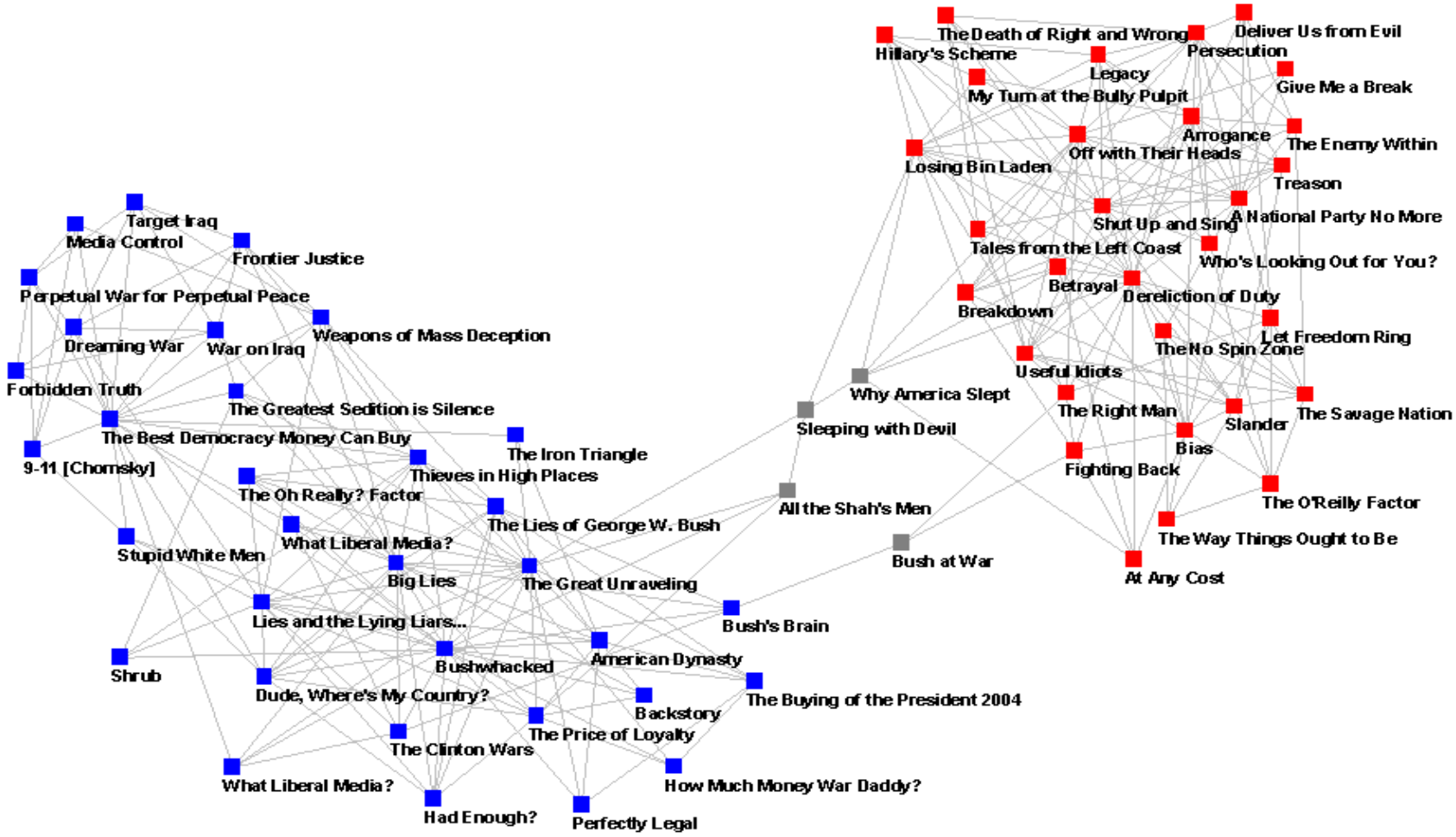
Sexual activity in high schools



Contagion of Tuberculosis



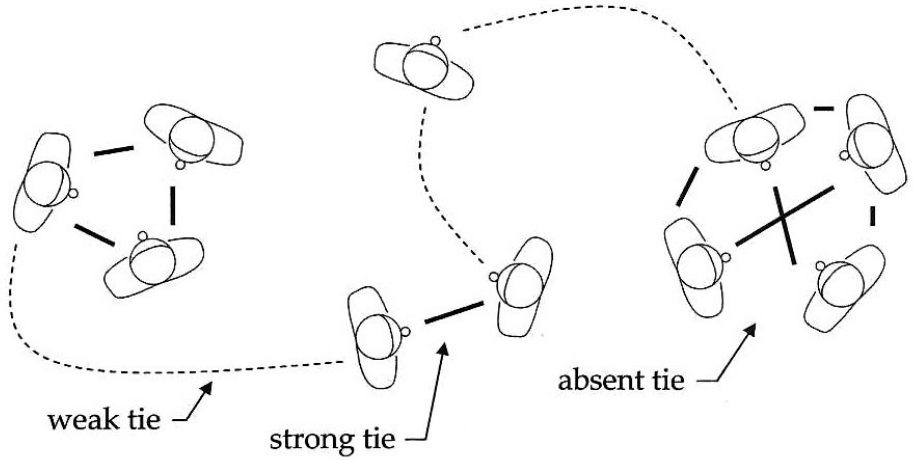
Political books



Goals of social network analysis

- **Goals of social network analysis**
  - Find degrees of separation through various connections: A -> B -> C -> D
  - Compute the connection strength from A to B (even if they never met) over a variety of types of connections
  - Cluster people in groups based on connection strengths
  - Find connection strengths between G1 and G2
  - Compute the centrality of a person in a group, or the centrality of a subgroup in a bigger group (also social capital*)
  - How does information flow from person to person through social networks
  - What is the chance that A and B are in the same virtual group

* Social capital is a core concept in business, organizational behavior and political science, defined as the advantage created by a person's location in a structure of relationships



Classical Search Techniques basis for SNA

All techniques

- Work on cyclic graphs
- Use generator functions to go from node to children of node
- Use evaluator functions for heuristic search
- Provide choice of maximum depth and whether to stop at first solution.

Example techniques

- `Depth-first (node1, node2, generator, maxdepth, exhaustive)`
- `Breadth-first (node1, node2, generator, maxdepth, exhaustive)`
- `Bi-directional (node1, node2, generator, maxdepth, exhaustive)`
- `Best-first (node1, node2, generator, maxdepth,exh, evaluator)`
- `A* (node1, node2, generator, maxdepth, exhaustive, evaluator)`

A sample of SNA techniques

- `In-degree (actor, generator)`
- `out-degree (actor, generator)`
- `Nodal-degree (actor, generator)`
- `Ego-group-select (group, generator, depth)`
- `Density (subgraph, generator)`
- `Actor-degree-centralization (actor, generator, group)`
- `Group-degree-centralization (group, generator)`
- `Actor-closeness-centrality (actor, group, generator)`
- `Group-closeness-centrality (group, generator)`
- `Actor-betweenness-centrality (actor, group, generator)`
- `Group-betweenness-centrality (group, generator)`
- `Clique (subgraph, graph, generator)`

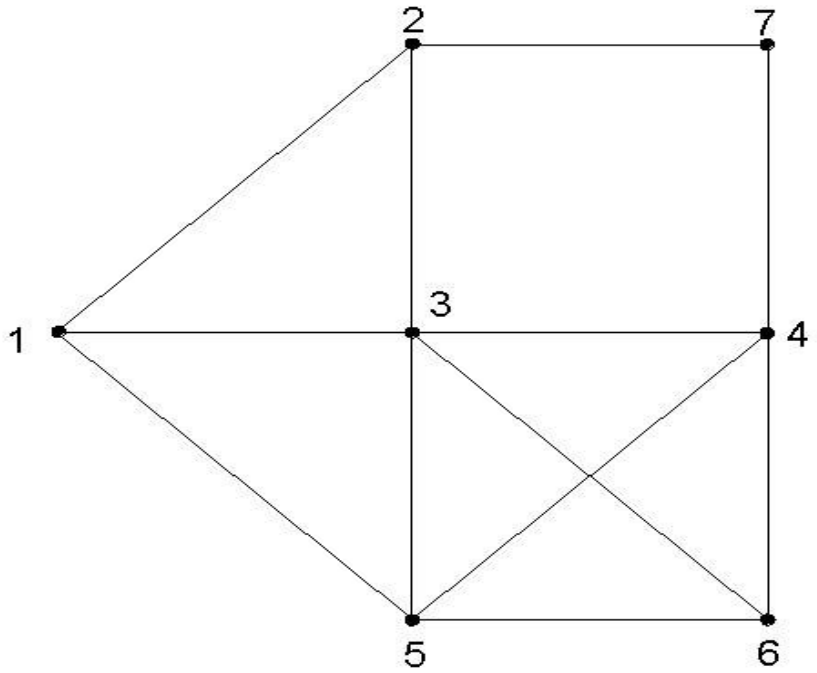
Ego-network (actor, generator, depth)

- Select a group of actors by applying generator until depth is reached
- Useful in some cases to avoid catastrophic complexity

Clique (actor, generator, minimum-depth)

- Find all the fully connected graphs around ‘actor’

7.2 Subgroups Based on Complete Mutuality



cliques: {1,2,3}, {1,3,5}, and {3,4,5,6}

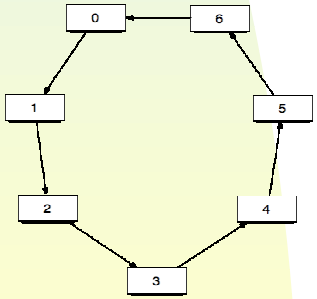
Fig. 7.1. A graph and its cliques

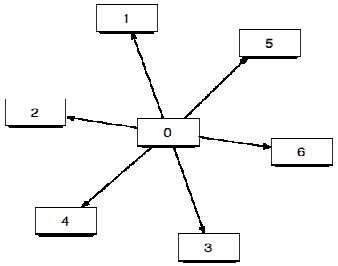
Basic degrees of actors

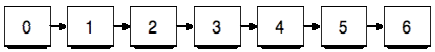
- In-degree (actor, generator)
  - How many nodes point to me according to generator
- Out-degree (actor, generator)
  - How many nodes do I point to
- Nodal-degree (actor, generator)
  - Add previous two.

Density (subgraph, generator)

- The normalized average degree of the actors in the graph. Roughly, how many of the possible connections between actors are realized.

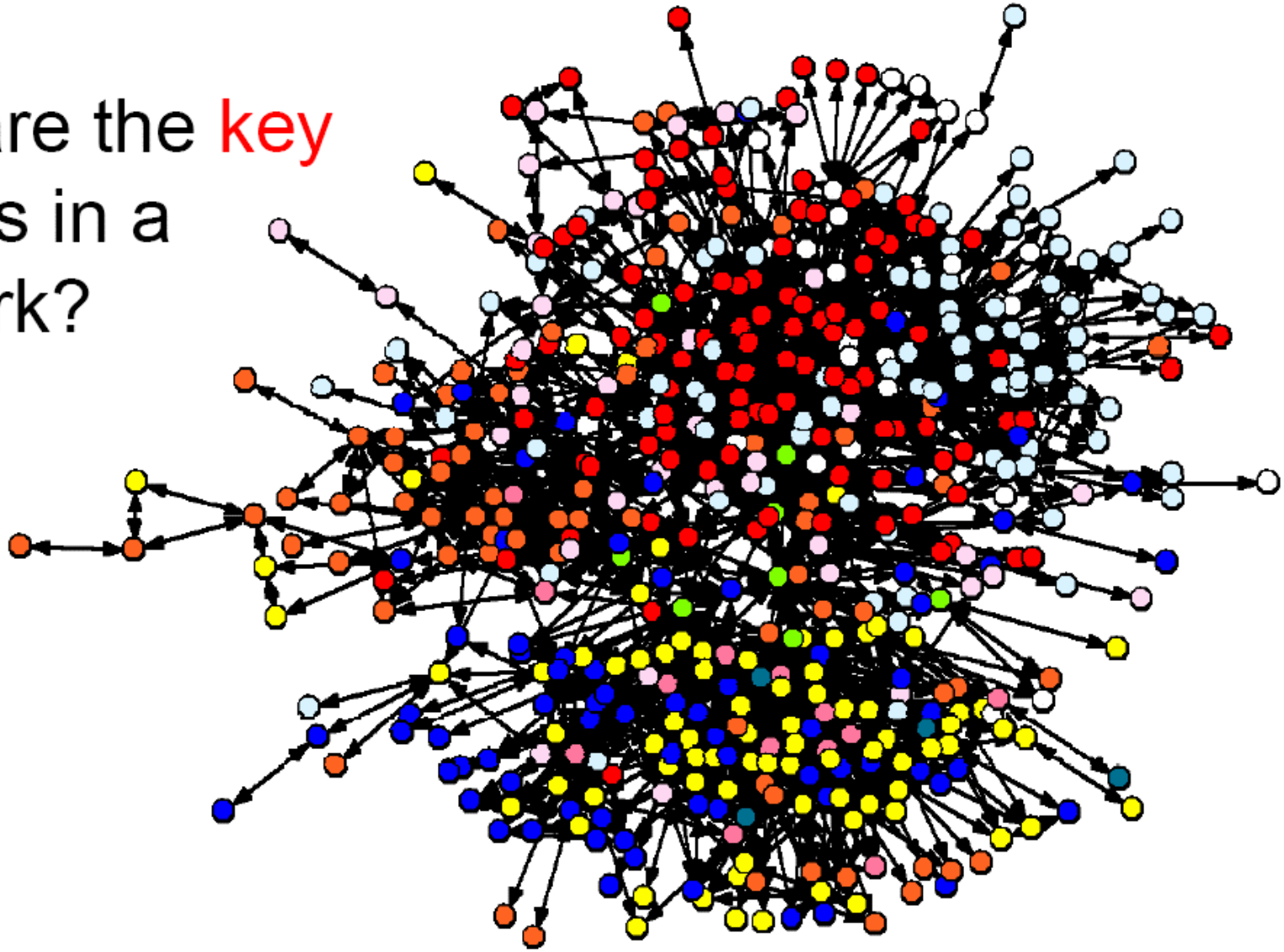






Research Question

- Who are the key players in a network?

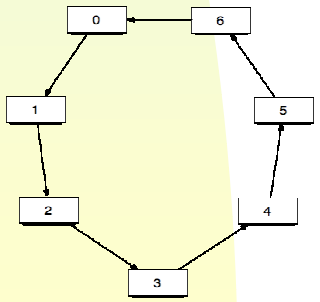


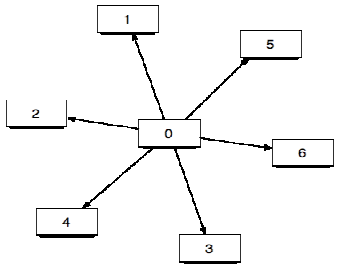
Actor-centrality

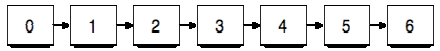
- How can we measure the importance of a single actor in his network
  - Actor degree centrality
  - Actor closeness centrality
  - Actor betweenness centrality

Actor-degree-centrality (actor, subgroup, generator)

- **Importance ==**
  - **"has high degree"**
- Examples
  - star: high for the middle, low for the ends
  - line: depends where you are in the line, lower at the end
  - circle: all degrees are the same

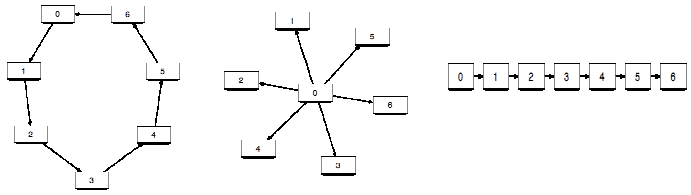






Actor-closeness-centrality (actor, subgroup, generator)

- **Importance == "can interact with many others" or "is close to many other actors"**
- **The (normalized) inverse average path length of all the shortest paths between the actor and every other member of the group. (Inverse so that higher values indicate more central actors).**
- Examples
  - star: highest for the middle
  - line: a little bit higher in the middle
  - circle: all the same



actor-betweenness-centrality (actor, subgroup, generator)

- Importance == "controls many paths in the graph"
- The actor-betweenness-centrality of actor _i_ is computed by counting the number of shortest paths between all pairs of actors (not including _i_) that pass through actor _i_. The assumption being that this is the chance that actor _i_ can control the interaction between_j_ and _k_.
- Example
  - star: the middle one is super important (1)
  - line: lower (at the end are zero)
  - circle: every one the same again

Group centrality:

- The overall centralization of the group using some actor characteristic as a measure. In general, sum the differences of the actor measure from the maximum actor measure and normalize.
- group-degree-centrality
  - Use actor-degree as the measure. It reaches its maximum in star-networks where one member is completely central and everyone else is on the periphery.
- group-closeness-centrality
  - Use actor-closeness as the measure.
- group-betweenness-centrality
  - Use actor-betweenness as the measure.

Contents of this tutorial

- RDF & triple stores in a few pages
- Difference between triple stores and RDBMS
- Triple Store Requirements
- Basic Social Network Analytics primitives
- Basic Geospatial primitives
- Temporal reasoning primitives
- Text indexing
- Combining it all in SPARQL and Prolog
- Hands on & demos

Geospatial Primitives

- If you want to adhere to the convention to define points with the predicates geo:latitude, geo:longitude then we provide special encoding for efficient spatial computations that compete with specialize spatial databases

```
<http://www.census.gov/ca/counties/central_contra_costa/moraga>
   rdf:type usgovt:Village ;
   dc:title "Moraga" ;
   dcterms:isPartOf <http://www.census.gov/ca/central_contra_costa> ;
   geo:lat 37.843776 ;
   geo:long -122.1245 ;
   census:population     16290 ;
   census:households      5760 ;
   census:landArea "24014073 m^2" ;
   census:waterArea "25388 m^2" ;
   census:details
    <http://www.census.gov/ca/central_contra_costa/moraga/censustables>
```

Sample Geospatial Primitives

- (geo-bouding-box ?x minlat maxlat minlon maxlon)
  - ?x unbound -> find all subjects ?x that are in bounding box
  - ?x bound -> is ?x in bouding box?
- (geo-box-around ?x ?y miles)
  - ?x bound, ?y unbound -> find all ?y in the bounding box going 'miles' left, right, up, down
- (geo-distance ?x ?y ?dist)
  - ?x and ?y need to be bound, ?dist is computed.
- (geo-distance< ?x ?y miles)
  - When ?y unbound, find all ?y in radius (miles).

Contents of this tutorial

- RDF & triple stores in a few pages
- Difference between triple stores and RDBMS
- Triple Store Requirements
- Basic Social Network Analytics primitives
- Basic Geospatial primitives
- Temporal reasoning primitives
- Text indexing
- Combining it all in SPARQL and Prolog
- Hands on & demos

Temporal Primitives

- Based on Allen's temporal logic and SNARK's interpretation of it: see
  `http://www.ai.sri.com/snark/tutorial/tutorial.html#htoc46`
- If you adhere to the convention to encode startimes and endtimes with franz:start and franz:end then we provide efficient temporal primitives

```
add-triple (event1, franz:start, "Thu Sep 27 14:21:43 2007")
add-triple (event1, franz:end,  "Thu Sep 27 14:21:43 2007")
```

- The primitives make using time more user friendly
- And are coded for efficiency

The point primitives

```
(point-before ?e1 ?e2)
(point-after ?e1 ?e2)
(point-simultaneous ?e1 ?e2)
(point-increasing ?e1 ?e2 ?e3)
(point-before ?p1 ?e1 ?p2 ?e2)
(point-after ?p1 ?e1 ?p2 ?e2)
(point-simultaneous ?p1 ?e1 ?p2 ?e2)
(point-increasing ?p1 ?e1 ?p2 ?e2 ?p3 ?e3)
```

Where ?e1 or ?e2 is a event resource and
?p1 and ?p2 are or !fr:start or !fr:end

Point primitives with specified date-times

```
(point-before-value ?p1 ?x ?date)
(point-after-value ?p1 ?x ?date)
(point-simultaneous-value ?p1 ?x ?date)
(point-increasing-value ?date1 ?p1 ?y ?date2)

where
   ?p1 is !fr:start or !fr:end (must be bound)
   ?x is resource (with start and/or end)
   ?date is a date-time in iso 8601 notation or as an
   integer
```

Interval primitives

```
(interval-before ?e1 ?e2)
(interval-meets ?e1 ?e2)
(interval-overlaps ?e1 ?e2)
(interval-starts ?e1 ?e2)
(interval-during ?e1 ?e2)
(inverval-finishes ?e2 ?e2)
(interval-after ?e1 ?e2)
(interval-met-by ?e1 ?e2)
(interval-overlapped-by ?e1 ?e2)
(interval-started-by ?e1 ?e2)
(interval-contains ?e1 ?e2)
(interval-finished-by ?e1 ?e2)
(interval-cotemporal ?e1 ?e2)
```

Interval primitives by value

```
(interval-before-value ?e1 ?min ?max)
(interval-meets-value ?e1 ?min ?max)
(interval-overlaps-value ?e1 ?min ?max)
(interval-starts-value ?e1 ?min ?max)
(interval-during-value ?e1 ?min ?max)
(inverval-finishes ?e1 ?min ?max)
(interval-after-value ?e1 ?min ?max)
(interval-met-by-value ?e1 ?min ?max)
(interval-overlapped-by-value ?e1 ?min ?max)
(interval-started-by-value ?e1 ?min ?max)
(interval-contains-value ?e1 ?min ?max)
(interval-finished-by-value ?e1 ?min ?max)
(interval-cotemporal-value ?e1 ?min ?max)
```

Where ?e1 is event resource (bound or unbound) and ?min and ?max are specified date-times

Contents of this tutorial

- RDF & triple stores in a few pages
- Difference between triple stores and RDBMS
- Triple Store Requirements
- Basic Social Network Analytics primitives
- Basic Geospatial primitives
- Temporal reasoning primitives
- Text indexing
- Combining it all in SPARQL and Prolog
- Hands on & demos

Text indexing

- Full text indexing on literals that contain natural language provides enormous additional value
- A basic example is the Enron email database with 500,000 email messages from 150 employees

```
(select (?x)
   (hasword ?x (and "losers" "california"))
   (q ?x enron:sender "Ken Lay"))
```

Contents of this tutorial

- RDF & triple stores in a few pages
- Difference between triple stores and RDBMS
- Triple Store Requirements
- Basic Social Network Analytics primitives
- Basic Geospatial primitives
- Temporal reasoning primitives
- Text indexing
- Combining it all in SPARQL and Prolog
- Hands on & demos

Combining it all with Prolog and SPARQL

(we are working on getting SPARQL to this phase)

```
(select (?x ?y)
   (qs OsamaBinLaden controls ?x ? ?triple-id) ➔ RDFS++ inference
   (q CIA beliefs ?triple-id (> .8)) ➔ Triple Referencing, Range Query
   (q ?x is-at ?p1 ?time1)  ➔Direct triple look up, time is named G
   (after ?time1 "2001-07-28T0:0:0") ➔Temporal primitive
   (ego-group ?x 2 ?group) ➔ Social networking analysis primitive
   (member-of  ?y ?group)➔  Plain prolog
   (q ?y is-at ?p2 ?time2)  ➔ Direct triple look up, time is named G
   (geodist-less ?p1 ?p2 12 kilometers)➔  Geospatial primitive
   (tempdist-less ?time1 ?time2 24 hours)) ➔Temporal primitive
```

Contents of this tutorial

- RDF & triple stores in a few pages
- Difference between triple stores and RDBMS
- Triple Store Requirements
- Basic Social Network Analytics primitives
- Basic Geospatial primitives
- Temporal reasoning primitives
- Text indexing
- Combining it all in SPARQL and Prolog
- Hands on & demos

Tutorial: hands-on

- Going through the CD, looking at the Java API
  - Look at: agraph.franz.com/iswc.lhmtl
- The basics of a triple store
- SPARQL
- Prolog & Select
- RDFS++ Reasoning
- Text indexing
- The Geospatial primitives
- The Temporal primitives
- The Social networking primitives
- Remaining time: playing with the examples from the learning center.

Going through the CD

- See agraph.franz.com/iswc.lhtml
- Eclipse + Java SDK
- Free version of Agraph
  - up to 50,000,000 triples
  - call us for unlimited eval version
- Example file from the learning centre
  - See http://agraph.franz.com/support/learning/
- A full Allegro Common Lisp
- Gruff: a graph visualizer for triple stores.