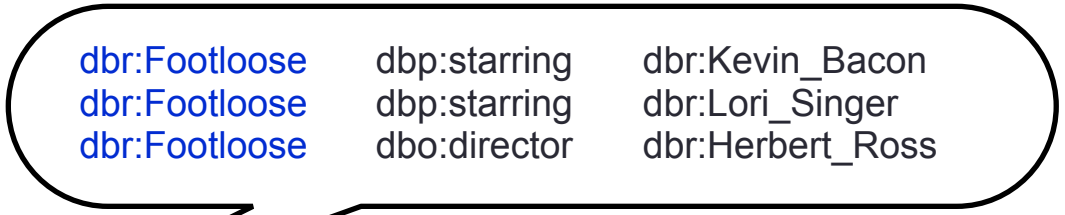
Linked Data

dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Linked Data

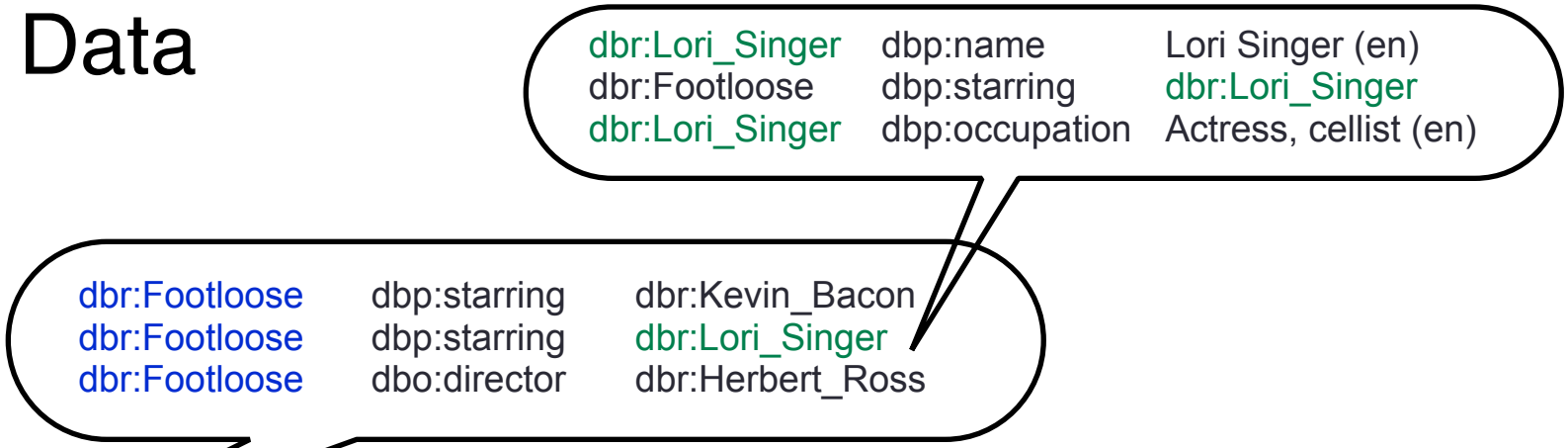
dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Linked Data



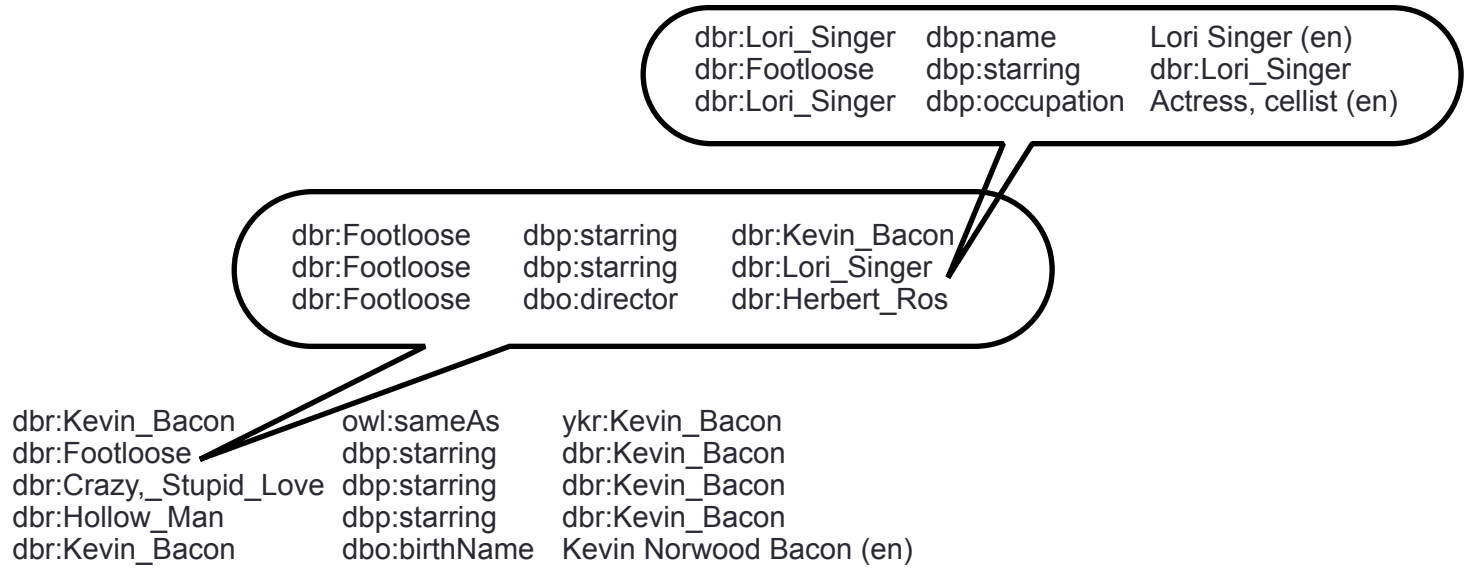
dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Linked Data



dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Linked Data + endpoints

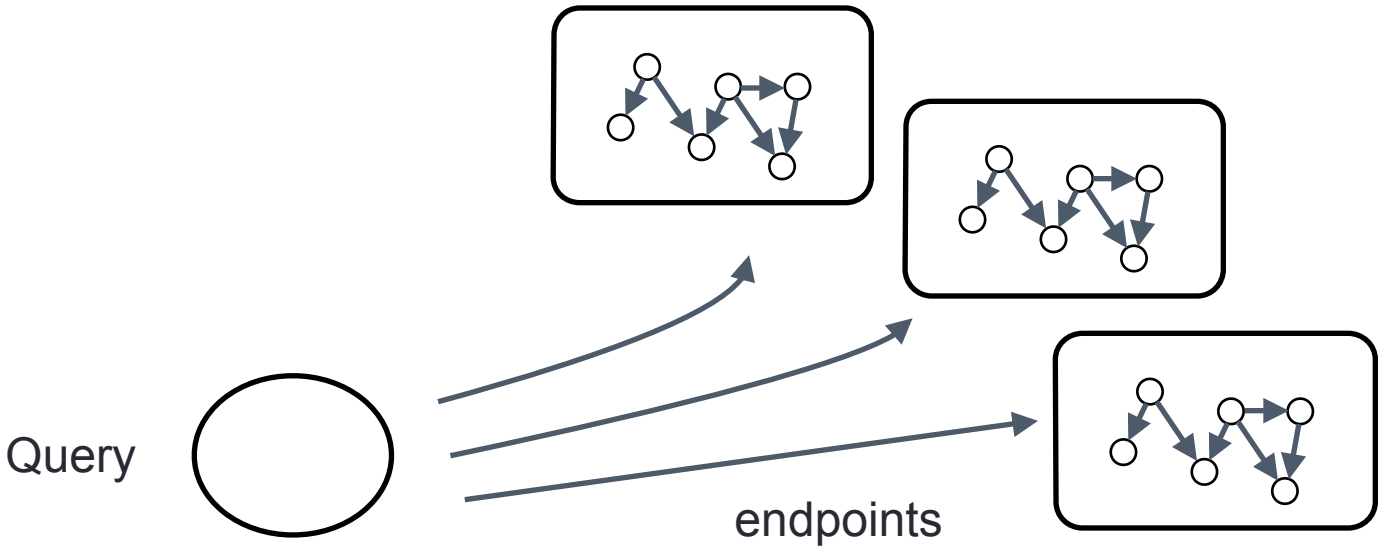


Linked Data + endpoints

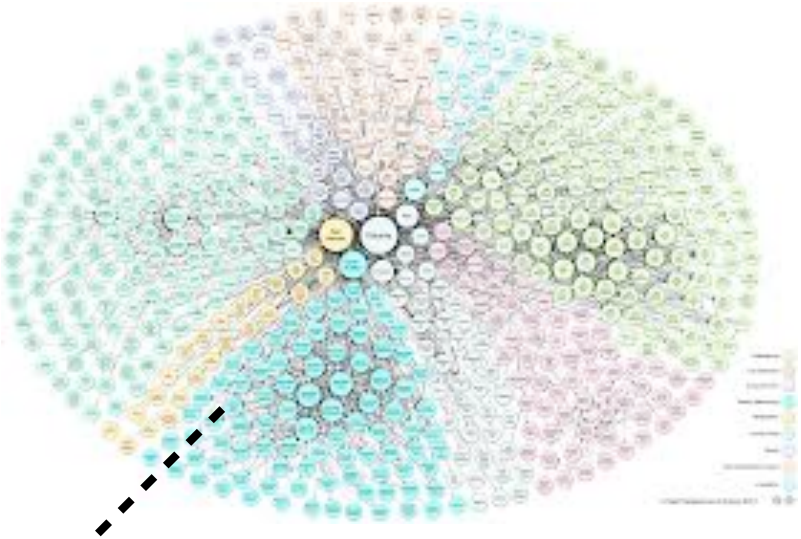
dbr:Lori_Singer	dbp:name	Lori Singer (en)
dbr:Footloose	dbp:starring	dbr:Lori_Singer
dbr:Lori_Singer	dbp:occupation	Actress, cellist (en)

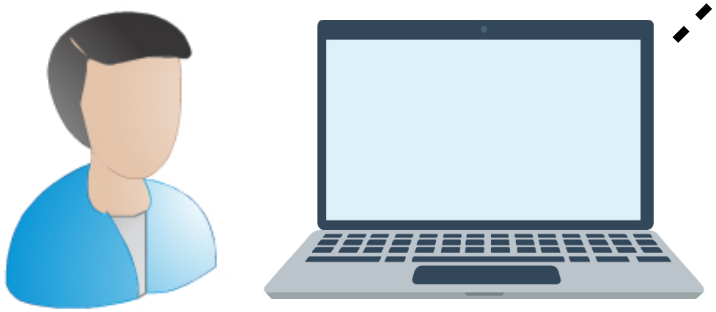
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Lori_Singer
dbr:Footloose	dbo:director	dbr:Herbert_Ros

dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)



The Dream:
Query the web as if it was a database





(still far from it)

Realising the dream?

What they say:

Why work on this?
The dream is too difficult to realise

What they don’t say:

It will remain impossible if we don’t work on it!
This is what we are supposed to do!
Pick challenging problems and make them reality

Realising the dream?

This talk: we focus on the challenge

Understand how to query distributed, ungoverned data

The Fascinating World of Querying Linked Data

Juan L. Reutter

CIWS

Center for **Semantic Web** Research

PUC Chile

The Fascinating World of Querying Linked Data

This talk:
3 specific problems (of course there are much more)

- Identify the correct semantics for queries over Linked Data
- How to compute queries over Linked Data
- Make the SERVICE operator work on Endpoints

Outline

What does it mean to query the web (semantics)

How to actually do it (algorithms)

SERVICE on Endpoints

Outline

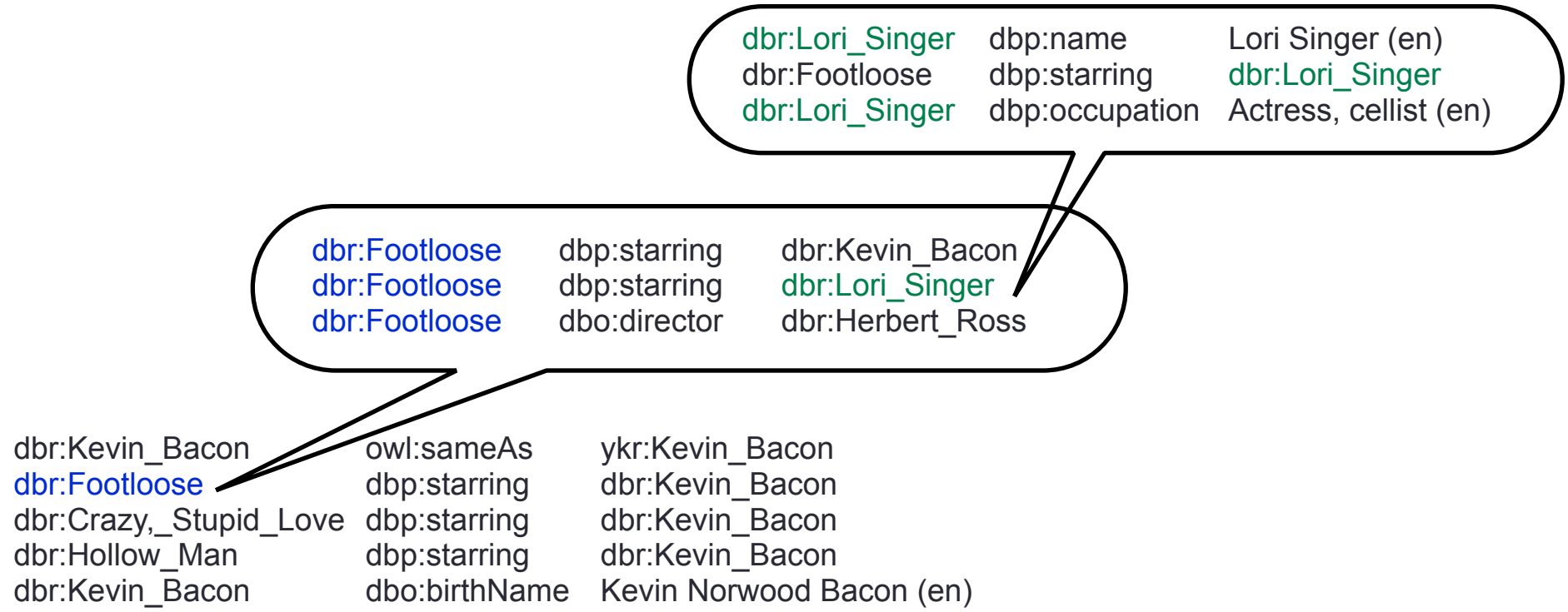
What does it mean to query the web (semantics)

Basics

Querying as an approximation

Other possible semantics

RDF dereference



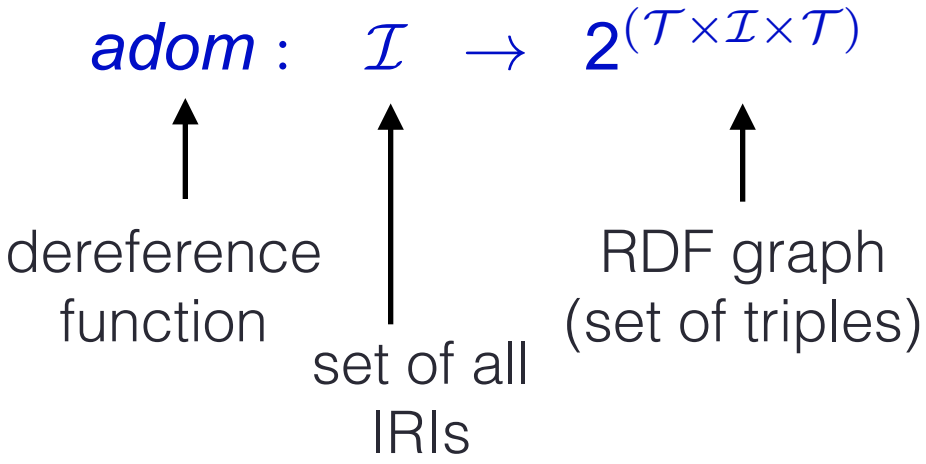
Web of Linked Data

Each IRI produces a (possibly empty) RDF graph

$$\mathit{adom}: \quad \mathcal{I} \quad \rightarrow \quad 2^{(\mathcal{T} \times \mathcal{I} \times \mathcal{T})}$$

Web of Linked Data

Each IRI produces a (possibly empty) RDF graph



Web of Linked Data

Each IRI produces a (possibly empty) RDF graph

$$adom : \quad \mathcal{I} \quad \rightarrow \quad 2^{(\mathcal{T} \times \mathcal{I} \times \mathcal{T})}$$

adom(dbr:Footloose)

dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Lori_Singer
dbr:Footloose	dbo:director	dbr:Herbert_Ross

dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Graph of Linked Data

Each IRI produces a (possibly empty) RDF graph

$$adom: \quad \mathcal{I} \quad \rightarrow \quad 2^{(\mathcal{T} \times \mathcal{I} \times \mathcal{T})}$$

Graph of Linked Data: Union of all these RDF graphs

(just one huge graph - everything that is out there)

Querying the Web of Linked Data

Database Perspective:
Query the entire graph of Linked Data:

$$\bigcup_{I \in \mathcal{I}} adom(I)$$

Querying the Web of Linked Data

Database Perspective:
Query the entire graph of Linked Data:

- the union of all RDF graphs found via dereferencing
- everything that's out there
- several domains, huge amount of triples

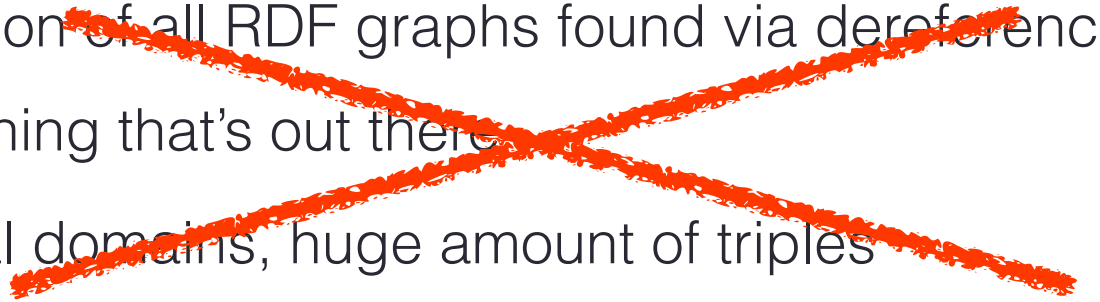
Querying the Web of Linked Data

Database Perspective:
Query the entire graph of Linked Data:

the union of all RDF graphs found via dereferencing

everything that's out there

several domains, huge amount of triples



How are we even supposed to do this?

Querying the Web of Linked Data (the only realistic approach)

Linked Data Perspective:

Querying the Web of Linked Data (the only realistic approach)

Linked Data Perspective:
Start with a bunch of IRIs or triples

- crawl more data at the same time we query
- ideally, don't crawl everything, only what we need

Example: querying for co-actors of Kevin Bacon (dbpedia)

{?mov dbp:starring dbr:Kevin_Bacon } .
{?mov dbp:starring ?act }

?mov	?act
dbr:Footloose	dbr:Lori_Singer
dbr:Footloose	dbr:Kevin_Bacon

adom(dbr:Footloose)

dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Lori_Singer
dbr:Footloose	dbo:director	dbr:Herbert_Ross
…	…	…

dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Example: querying for co-actors of Kevin Bacon (dbpedia)

{?mov dbp:starring dbr:Kevin_Bacon } .
{?mov dbp:starring ?act }

?mov	?act
dbr:Footloose	dbr:Lori_Singer
dbr:Footloose	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbr:Ryan_Goslin
dbr:Crazy,_Stupid_Love	dbr:Julianne_Moore

adom(dbr:Crazy,_Stupid_Love)

dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Ryan_Gosling
dbr:Crazy,_Stupid_Love	dbo:director	dbr:Julianne_Moore
…	…	…

dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Example: querying for co-actors of Kevin Bacon (dbpedia)

{?mov dbp:starring dbr:Kevin_Bacon } .
{?mov dbp:starring ?act }

?mov	?act
dbr:Footloose	dbr:Lori_Singer
dbr:Footloose	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbr:Ryan_Goslin
dbr:Crazy,_Stupid_Love	dbr:Julianne_Moore
…	…
…	…
…	…

adom(dbr:Hollow_Man)

dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Key Issue:

Need to understand this crawling.

How to do it

What to expect form it

How to work with it

Outline

What does it mean to query the web (semantics)

Basics

Querying as an approximation

Other possible semantics

Querying linked data as an approximation

Previous work generally understands
crawling as a way of getting partial answers

answers while crawling the web $\subseteq$ answers if we could download the entire web

Works very well for monotone queries

Monotone Queries

A query Q is monotone if

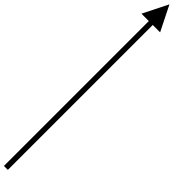
$$Q(G) \subseteq Q(G')$$

whenever the graph G is a subset of the graph G'

Monotone Queries

A query Q is monotone if

$$Q(G) \subseteq Q(G')$$



evaluation of query Q over graph G

whenever the graph G is a subset of the graph G'

Monotone Queries

evaluation of query Q over graph G

A query Q is monotone if

$$Q(G) \subseteq Q(G')$$

whenever the graph G is a subset of the graph G'

If we put more triples in the graph,
the answer is either the same, or bigger

answers while crawling the web $\subseteq$ answers if we could download the entire web

Works very well for monotone queries:

answers while crawling the web $\subseteq$ answers if we could download the entire web

Works very well for monotone queries:

if Q is a monotone query,
then any crawling strategy can be seen as an approximation

answers while crawling the web $\subseteq$ answers if we could download the entire web

Works very well for monotone queries:

if Q is a monotone query,
then any crawling strategy can be seen as an approximation

As we crawl we increase the size of the graph,
so we may find more answers.

Eventually we'll download the entire web,
delivering all answers.

Example: querying for co-actors of Kevin Bacon (dbpedia)

{?mov dbp:starring dbr:Kevin_Bacon } .
{?mov dbp:starring ?act }

?mov	?act
dbr:Footloose	dbr:Lori_Singer
dbr:Footloose	dbr:Kevin_Bacon

subset of all answers

adom(dbr:Footloose)

dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Lori_Singer
dbr:Footloose	dbo:director	dbr:Herbert_Ross
…	…	…

dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Example: querying for co-actors of Kevin Bacon (dbpedia)

```
{?mov  dbp:starring  dbr:Kevin_Bacon } .
{?mov  dbp:starring  ?act }
```

?mov	?act
dbr:Footloose	dbr:Lori_Singer
dbr:Footloose	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbr:Ryan_Goslin
dbr:Crazy,_Stupid_Love	dbr:Julianne_Moore

more data,
more answers

dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Ryan_Gosling
dbr:Crazy,_Stupid_Love	dbo:director	dbr:Julianne_Moore

dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Lori_Singer
dbr:Footloose	dbo:director	dbr:Herbert_Ross

answers while crawling the web $\subseteq$ answers if we could download the entire web

However, this approach has its drawbacks

- when queries are not monotone

- when queries do not return mappings

Drawback: non monotone queries

{?mov dbp:starring dbr:Kevin_Bacon } . { ?mov dbp:starring ?act } .

OPTIONAL {?act rdf:label ?name . FILTER (?name != “Lori Singer”) }

Drawback: non monotone queries

{?mov dbp:starring dbr:Kevin_Bacon } . { ?mov dbp:starring ?act } .
OPTIONAL {?act rdf:label ?name . FILTER (?name != “Lori Singer”) }

?mov	?act	?name
dbr:Footloose	dbr:Lori_Singer	
dbr:Footloose	dbr:Kevin_Bacon	

dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Lori_Singer
dbr:Footloose	dbo:director	dbr:Herbert_Ross
…	…	…

once we discover

adom(
dbr:Lori_Singer)

we’ll need to delete this answer

dbr:Kevin_Bacon	owl:sameAs	ykr:Kevin_Bacon
dbr:Footloose	dbp:starring	dbr:Kevin_Bacon
dbr:Crazy,_Stupid_Love	dbp:starring	dbr:Kevin_Bacon
dbr:Hollow_Man	dbp:starring	dbr:Kevin_Bacon
dbr:Kevin_Bacon	dbo:birthName	Kevin Norwood Bacon (en)

Drawback: non monotone queries

```
{?mov dbp:starring  dbr:Kevin_Bacon } . { ?mov  dbp:starring ?act } .
OPTIONAL {?act rdf:label ?name . FILTER (?name != “Lori Singer”) }
```

we cannot guarantee that the partial answers are answers over the entire web

answers while crawling the web

answers if we could download the entire web

some answers here

may not be answers here

Drawback: queries returning values

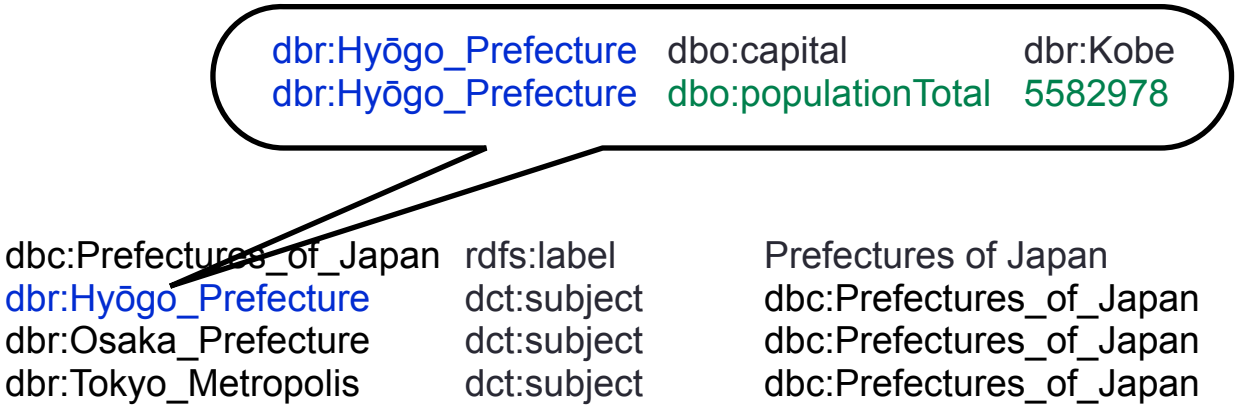
SELECT MAX(?pop) WHERE
{?dis dct:subject dbc:Prefectures_of_Japan} . {?dis dbo:populationTotal ?pop }

dbc:Prefectures_of_Japan	rdfs:label	Prefectures of Japan
dbr:Hyōgo_Prefecture	dct:subject	dbc:Prefectures_of_Japan
dbr:Osaka_Prefecture	dct:subject	dbc:Prefectures_of_Japan
dbr:Tokyo_Metropolis	dct:subject	dbc:Prefectures_of_Japan

Drawback: queries returning values

SELECT MAX(?pop) WHERE
{?dis dct:subject dbc:Prefectures_of_Japan} . {?dis dbo:populationTotal ?pop }

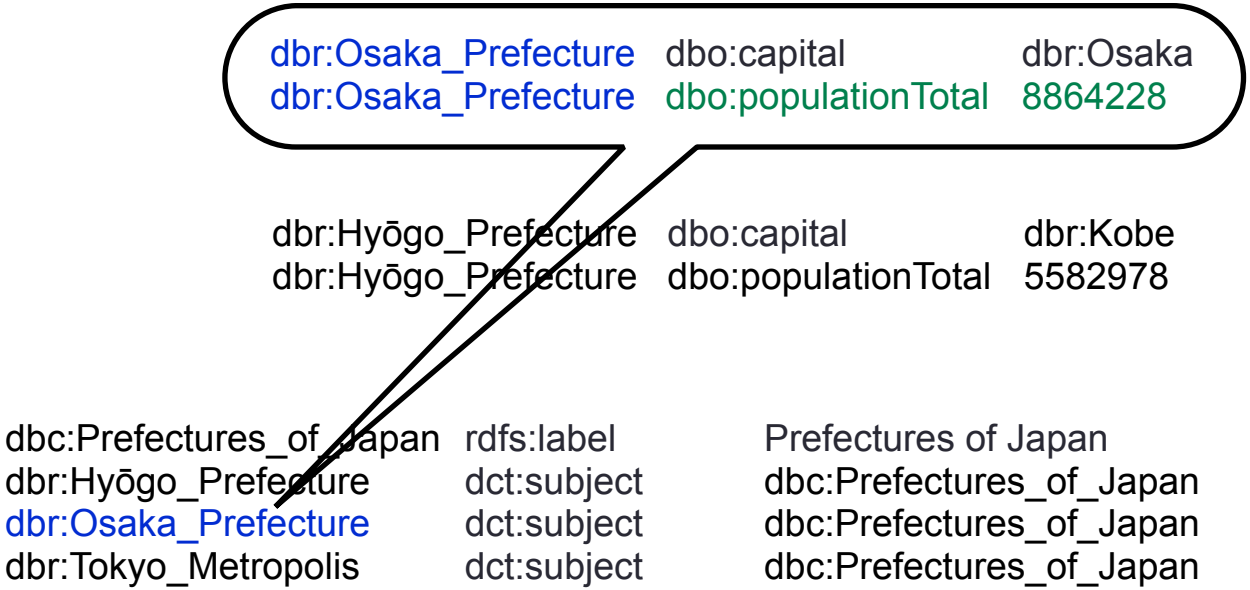
Answer: 5.582.978



Drawback: queries returning values

SELECT MAX(?pop) WHERE
{?dis dct:subject dbc:Prefectures_of_Japan} . {?dis dbo:populationTotal ?pop }

Answer: 5.582.978
Was wrong!
New answer: 8.864.228

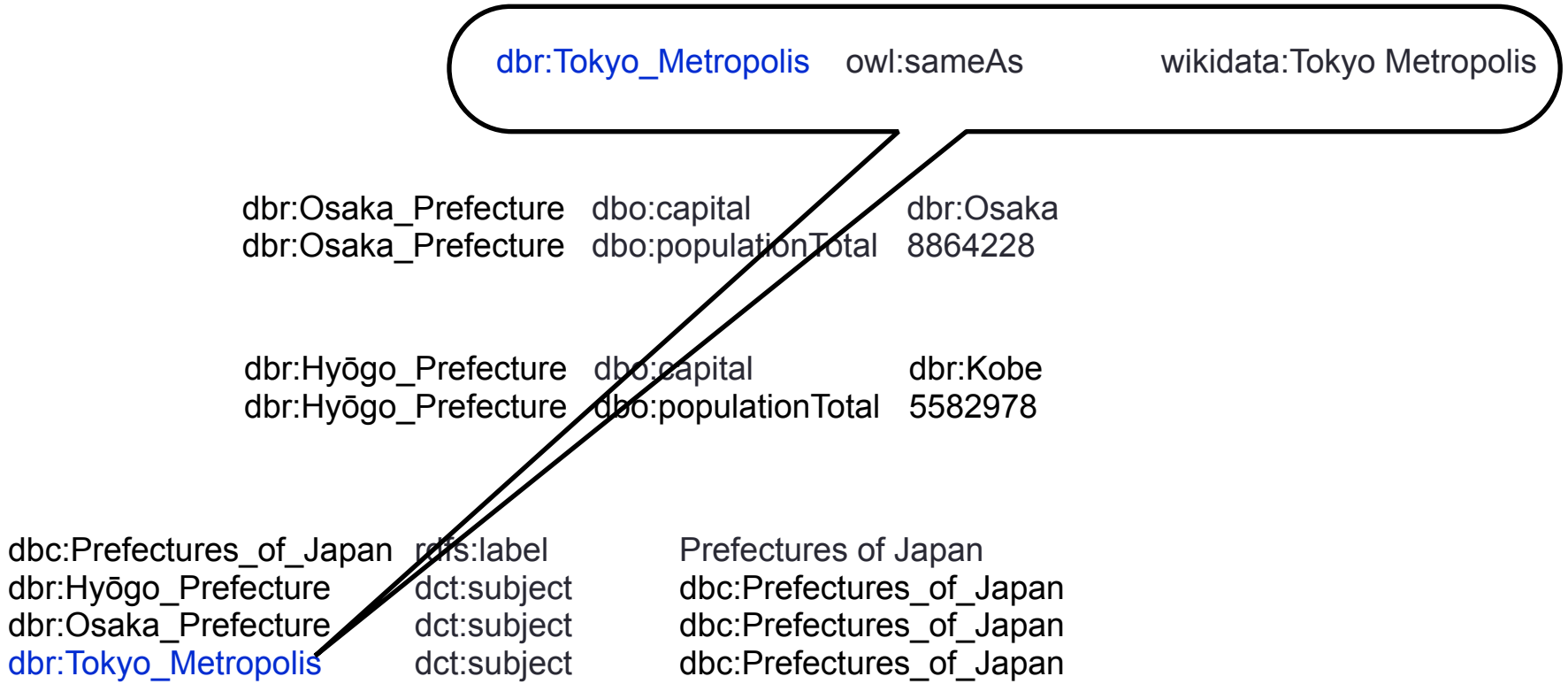


Drawback: queries returning values

SELECT MAX(?pop) WHERE

{?dis dct:subject dbc:Prefectures_of_Japan} . {?dis dbo:populationTotal ?pop }

Answer: 8.864.228
Is it correct?



Drawback: queries returning values

```
SELECT MAX(?pop) WHERE
{?dis dct:subject dbc:Prefectures_of_Japan} . {?dis dbo:populationTotal ?pop }
```

Cannot guarantee answer is correct until crawl has finished.

But how do we know it has finished?

answers while crawling the web ⊈ answers if we could download the entire web

this is a number

may be different to the correct one

Outline

What does it mean to query the web (semantics)

Basics

Querying as an approximation

Other possible semantics

Key Issue:

Understand the behaviour of web queries

What information could we get from the web, if we knew exactly what we were doing?

First attempt: all-powerful crawler

Imagine an almighty user, knows the entire Web of Linked Data

On a given query and a given starting IRI,
the user knows exactly what to dereference to
eventually answer this query

Theorem

Any algorithm that can simulate an all-powerful crawler
must sometimes download the entire Web of Linked Data

First attempt: all-powerful crawler

Theorem

Any algorithm that can simulate an all-powerful crawler must sometimes download the entire Web of Linked Data

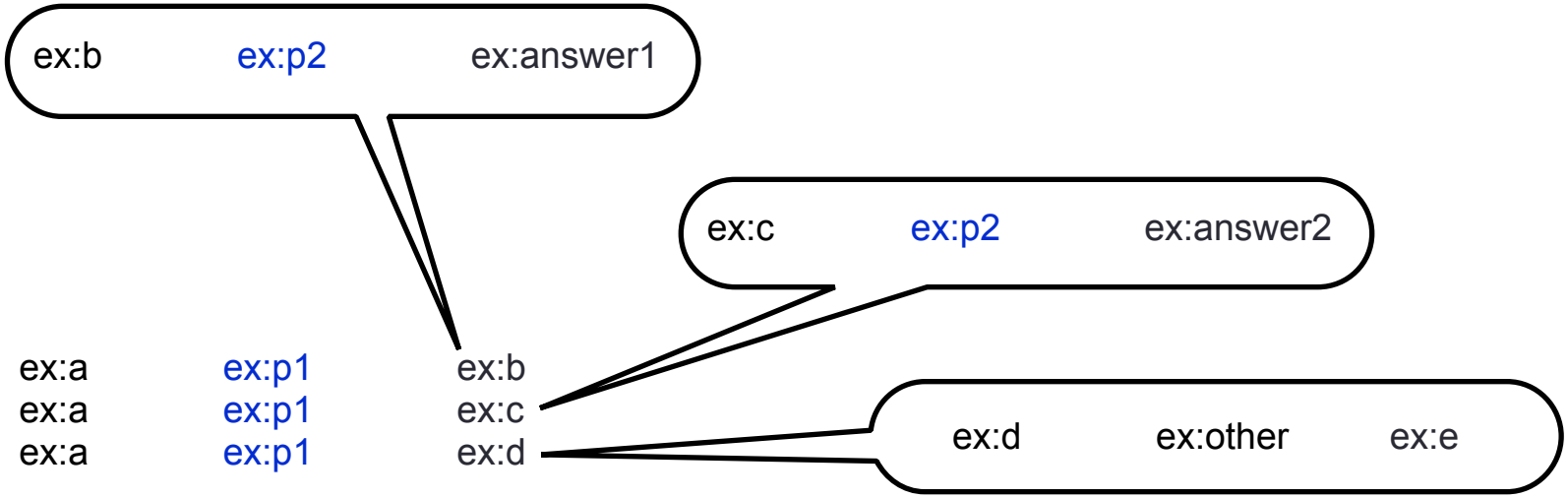
Basically:

The only way in which we can simulate an all-powerful crawler is to download the entire Web of Linked Data

All-powerful crawler (Intuition)

```
SELECT ?ans
WHERE  {ex:a  ex:p1 ?y } . { ?y  ex:p2 ?ans }
```

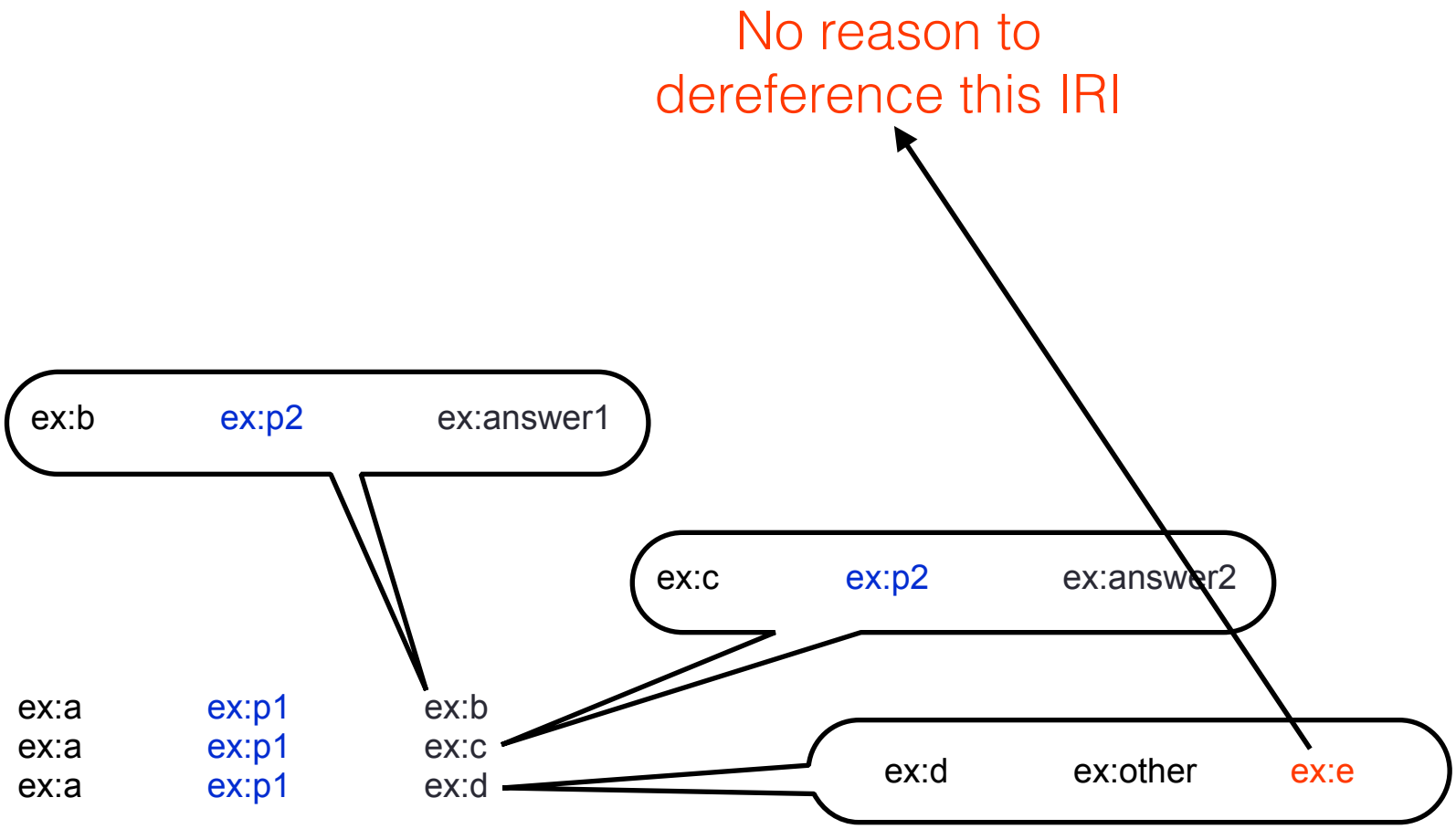
?ans
ex:answer1
ex:answer2



All-powerful crawler (Intuition)

```
SELECT ?ans
WHERE  {ex:a  ex:p1 ?y } . { ?y  ex:p2 ?ans }
```

?ans
ex:answer1
ex:answer2



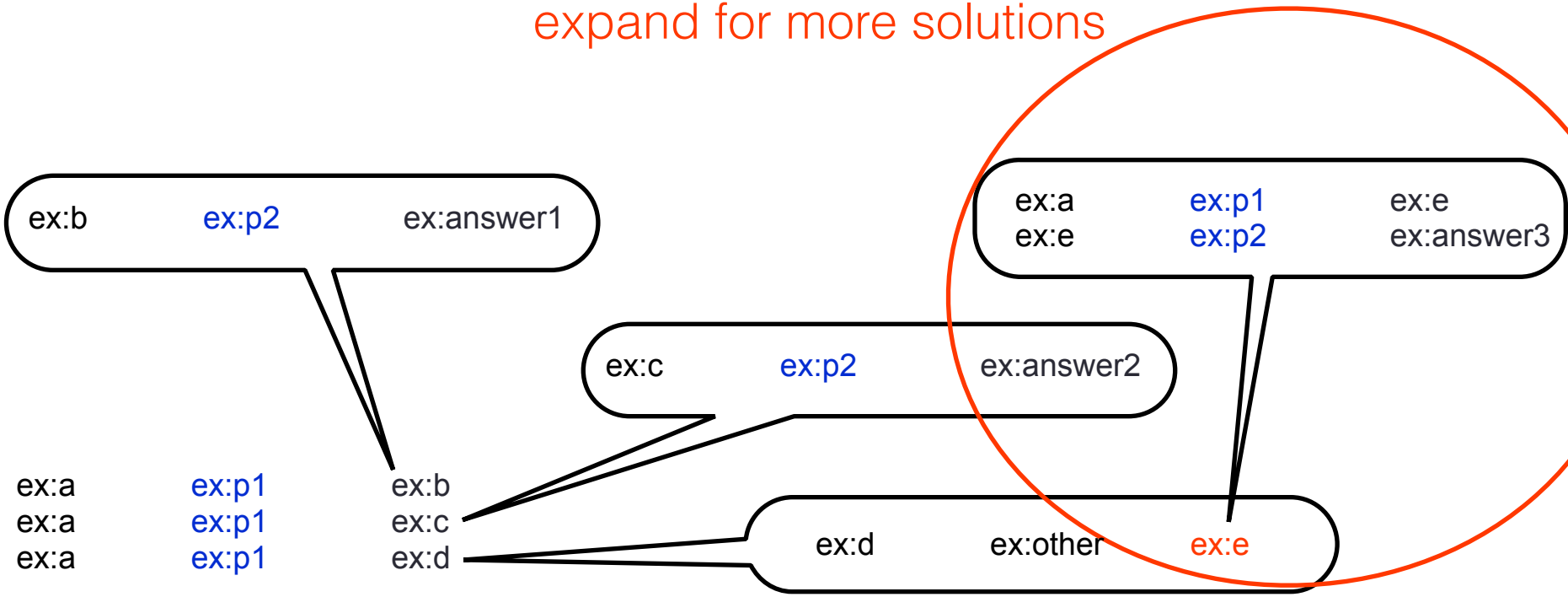
All-powerful crawler (Intuition)

```
SELECT ?ans
WHERE  {ex:a  ex:p1 ?y } . { ?y  ex:p2 ?ans }
```

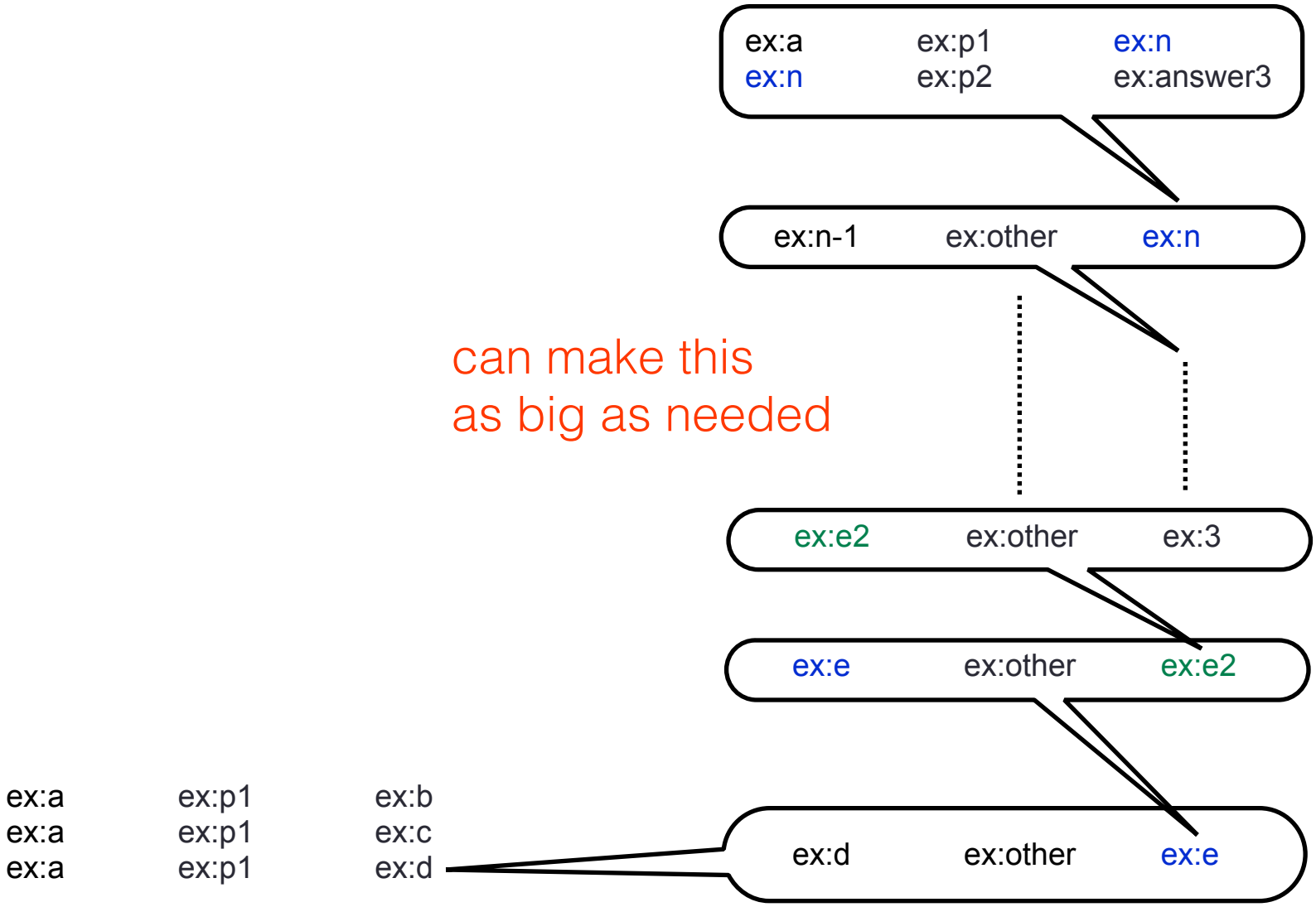
?ans
ex:answer1
ex:answer2
ex:answer3

No reason to
dereference this IRI

Crawler guess:
expand for more solutions



```
SELECT ?ans
WHERE  {ex:a  ex:p1 ?y } . { ?y  ex:p2 ?ans }
```



Second attempt: All-powerful crawler, but must justify her moves

Same almighty user, knows the entire Web of Linked Data

However, in order to dereference a new document,
the crawler must guarantee that this document may lead to answers

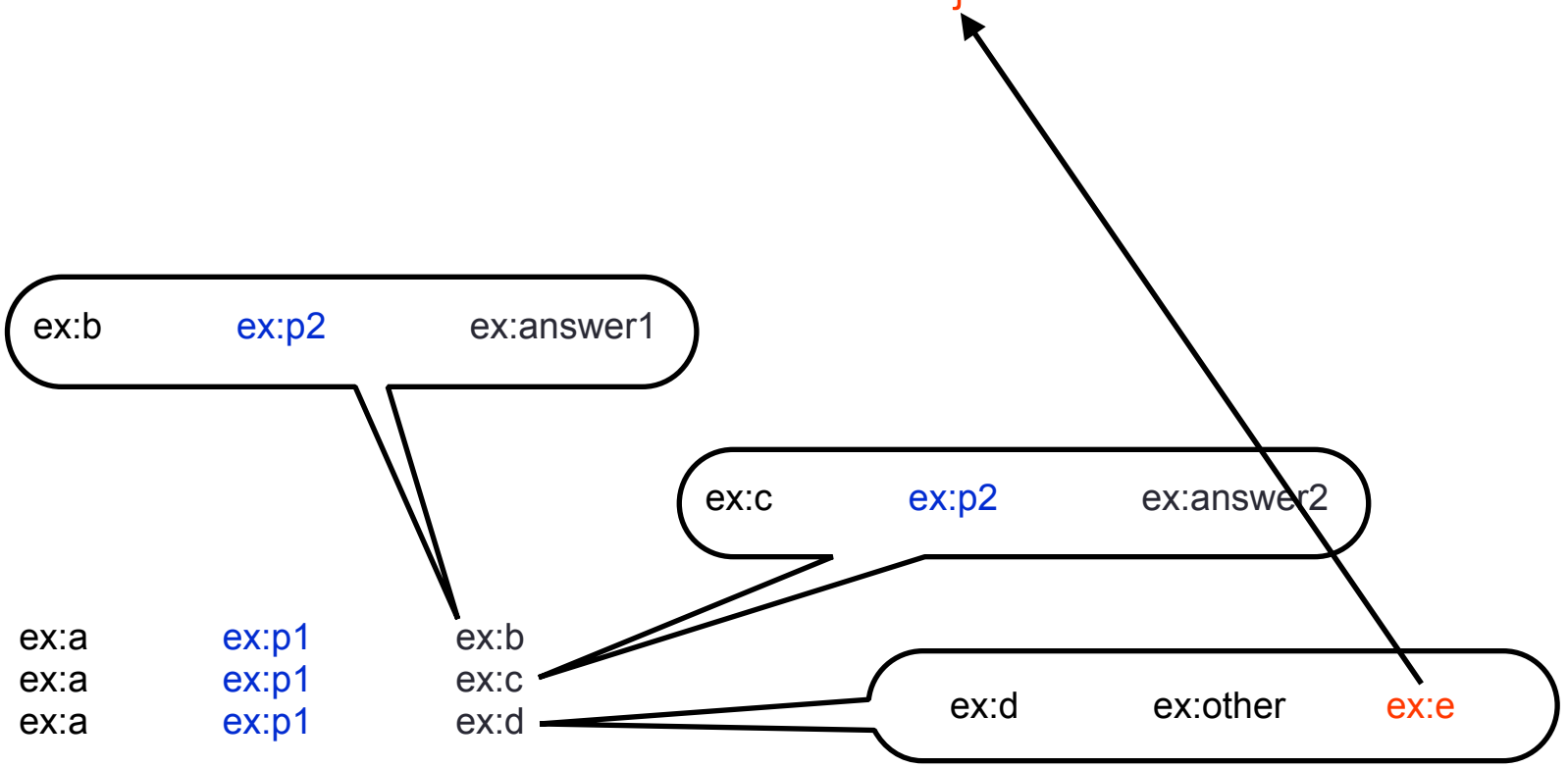
(also known as a reachability criterion [Hartig 12])

Justifications when dereferencing (Intuition)

```
SELECT ?ans
WHERE  {ex:a  ex:p1 ?y } . { ?y  ex:p2 ?ans }
```

?ans
ex:answer1
ex:answer2

Can't dereference ex:e, there is no justification

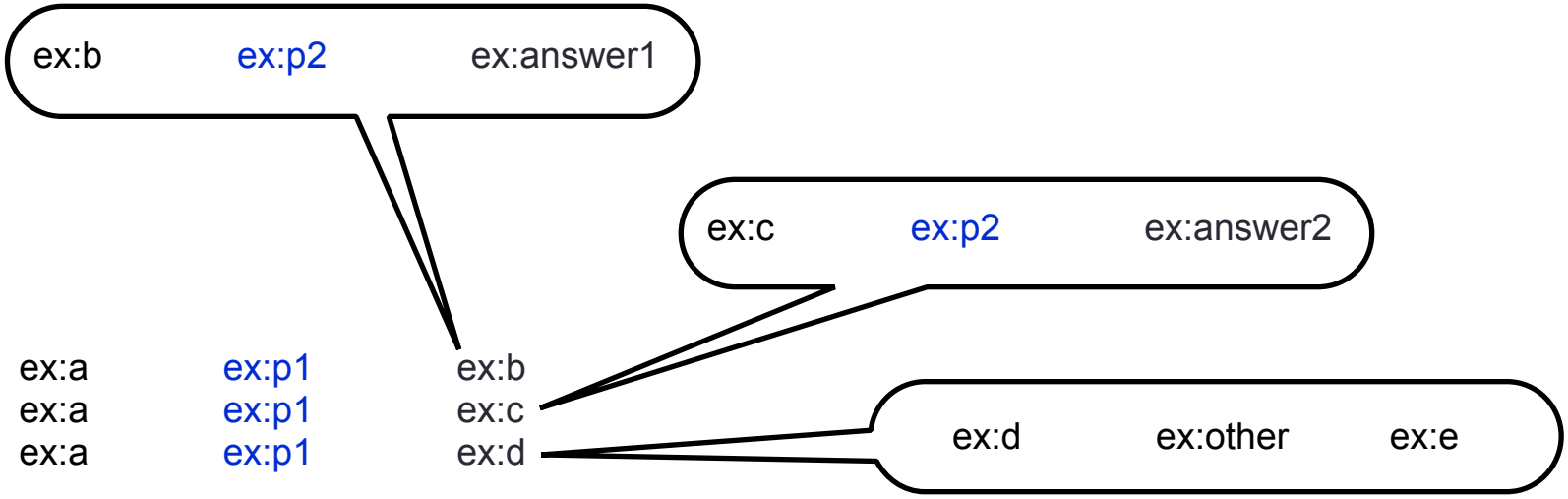


Justifications when dereferencing (Intuition)

```
SELECT ?ans
WHERE  {ex:a  ex:p1 ?y } . { ?y  ex:p2 ?ans }
```

?ans
ex:answer1
ex:answer2

This is what the crawler can do.
This is the final answer



Second attempt: All-powerful crawler, but must justify her moves

Important:

The answers produced may be incomplete or even wrong!

But there was probably no realistic way of getting the correct answers

A query language L is complete for justified crawling
if there is an algorithm that, for each query Q in L,

retrieves the same answers for Q
as an all-powerful crawler that needs to justify her moves

Second attempt: All-powerful crawler, but must justify her moves

A query language L is complete for justified crawling
if there is an algorithm that, for each query Q in L,
retrieves the same answers for Q
as an all-powerful crawler that needs to justify her moves

Is this the correct approach? are there other approaches?

Need more research!

Second attempt: All-powerful crawler, but must justify her moves

A query language L is complete for justified crawling
if there is an algorithm that, for each query Q in L,
retrieves the same answers for Q
as an all-powerful crawler that needs to justify her moves

Is this the correct approach? are there other approaches?

Need more research!

Two examples:
- BGPs
- Property Paths

All-powerful crawler with justification, Basic Graph Patterns

Given an IRI *I* and a BGP *Q*,

I is justified (in our local temporal graph)
if there is a partial match for *Q* using a triple with *I*

Theorem: [Hartig, Bizer, Freytag 09]

BGPs are complete for justified crawling

All-powerful crawler with justification, Property Paths (intuition)

Can extend this to Property Paths.

An IRI is now justified if it participates in the partial evaluation of a property path

Theorem: [Hartig, Pirrò 15]

PPs are complete for justified crawling

All-powerful crawler with justification, Property Paths (intuition)

Can extend this to Property Paths.

An IRI is now justified if it participates in the partial evaluation of a property path

Can we extend this further?
does it make sense?

Theorem: [Hartig, Pirrò 15]

PPs are complete for justified crawling

Main Takeaway: we have no idea what to do

Main Takeaway: we have no idea what to do

- Query answers as approximation:
  good for monotone queries,
  does not generalise to full SPARQL

- Another option:

retrieve the same as an all-powerful crawler with justification
  Is this intuitive?
  Need a formal definition for full SPARQL.

- Other options? what to expect from non-monotone queries?

- What about SPARQL Entailment Regimes?

Outline

What does it mean to query the web (semantics)

How to actually do it (algorithms)

SERVICE on Endpoints

Outline

How to actually do it (algorithms)

Example:
Property Paths

Comparing Algorithms

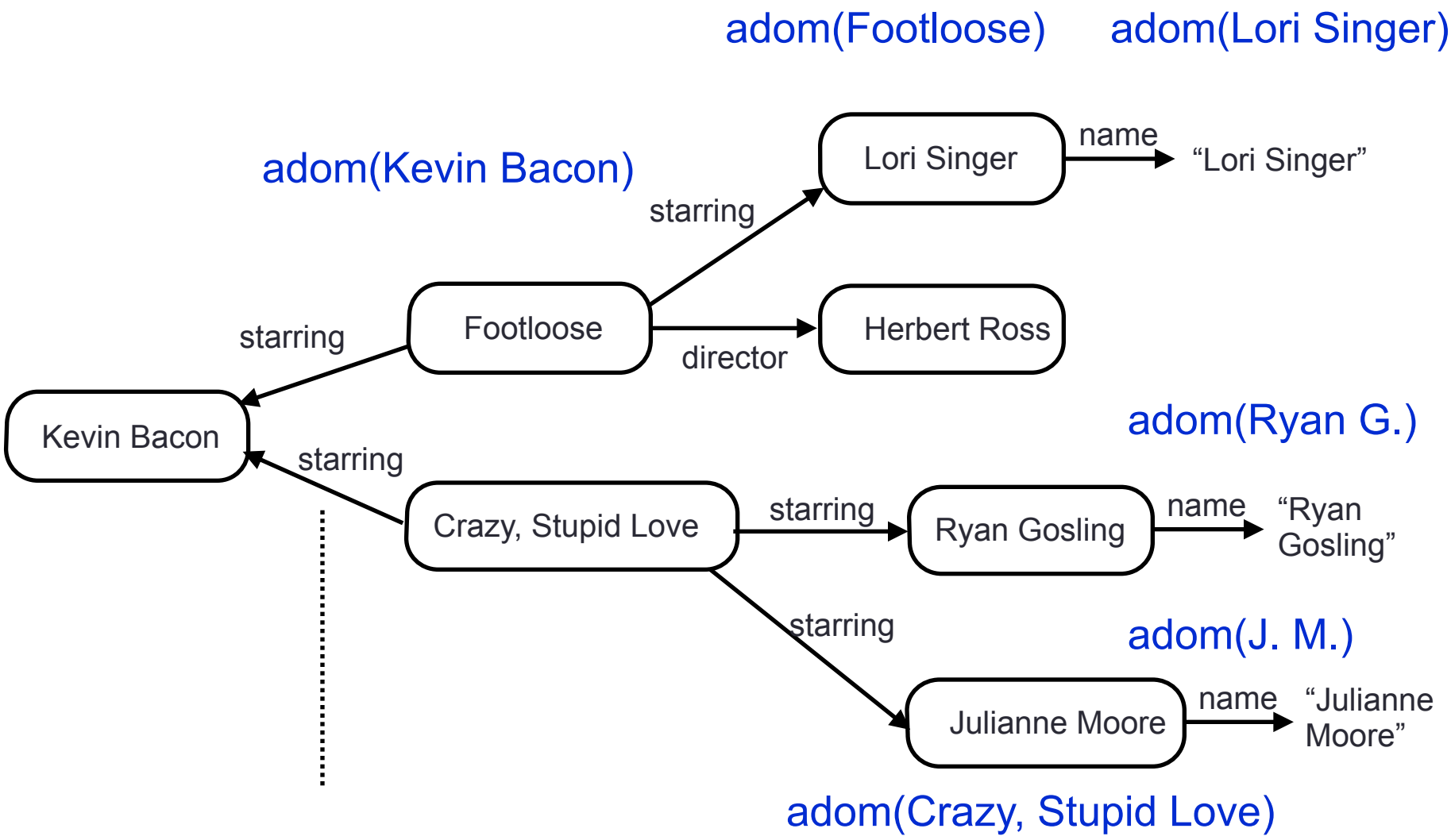
Even if we know what to do, there is the issue of how to do it

Some amount of work done for BGPs already
(go to Thursday's afternoon Search(II) session)

We'll see some work in progress for Property Paths
(here, even if we have an algorithm, it may not terminate)

Computing the answers of property paths: example

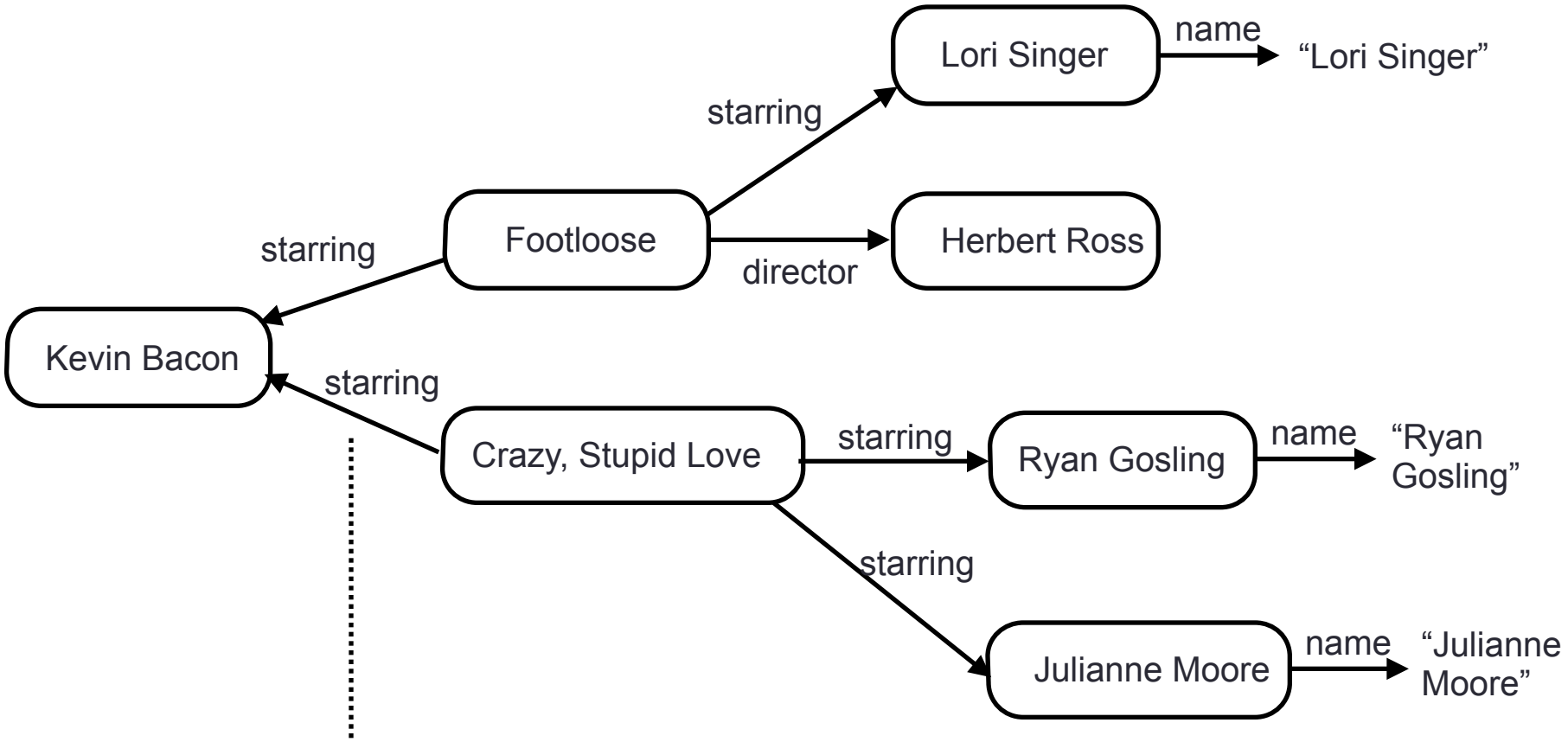
{Bacon ^starring/starring/name ?x }

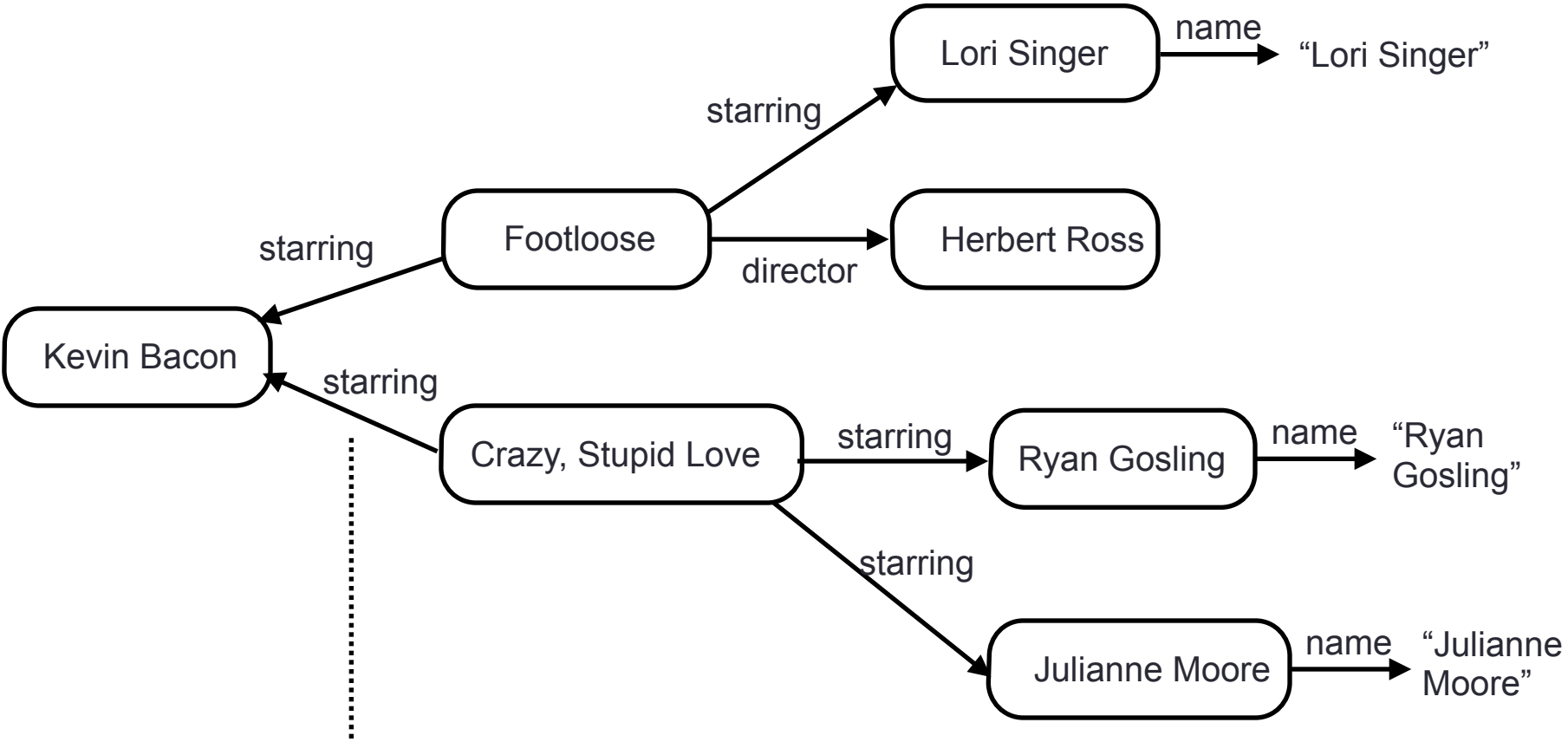


Computing the answers of property paths: example

{Bacon ^starring/starring/name ?x }

Justification: can dereference as long as IRI appears in path





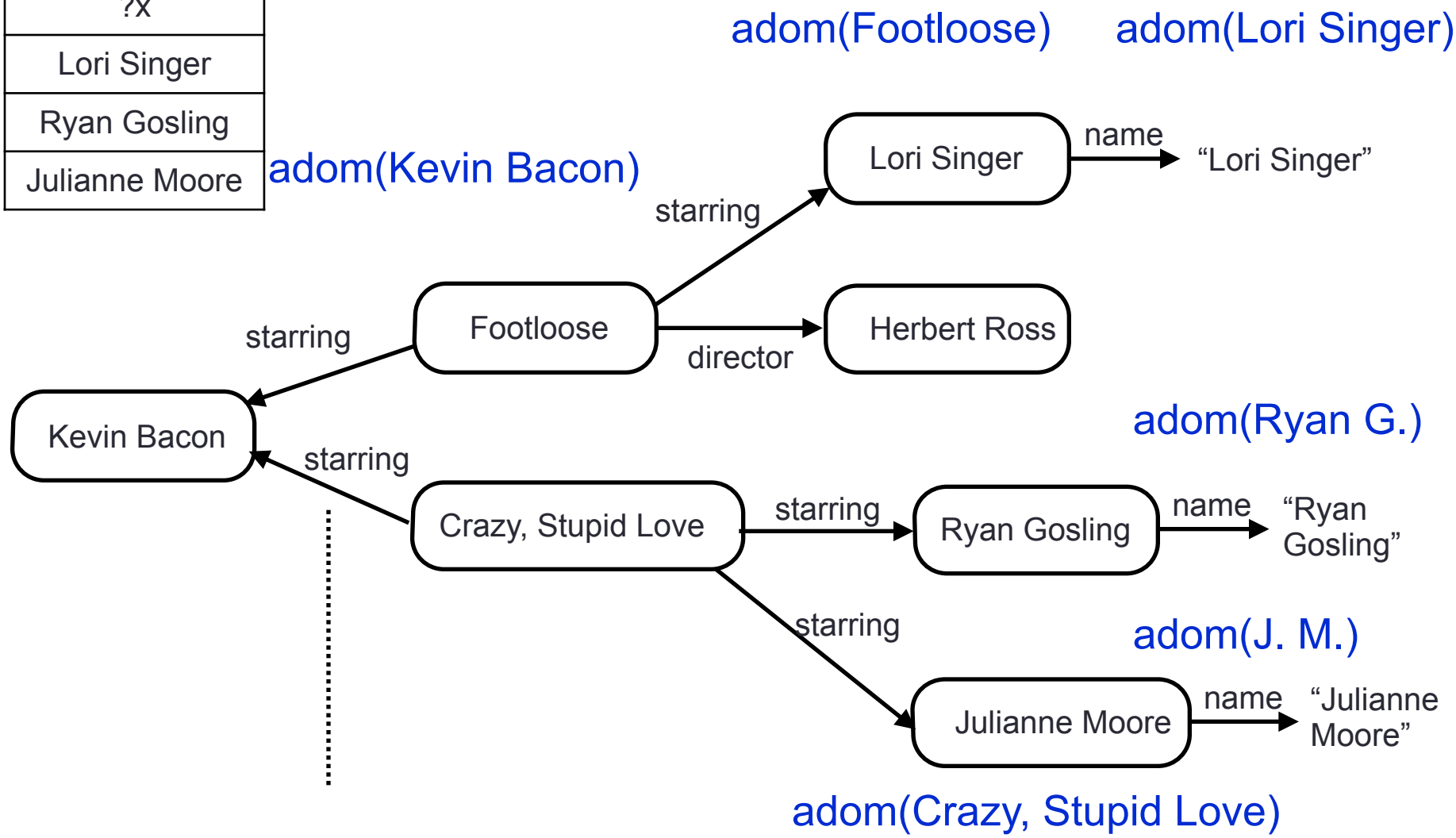
{Bacon ^starring/starring/name ?x }

Answers to this query are paths
^ starring / starring / name

How do we discover this graph?

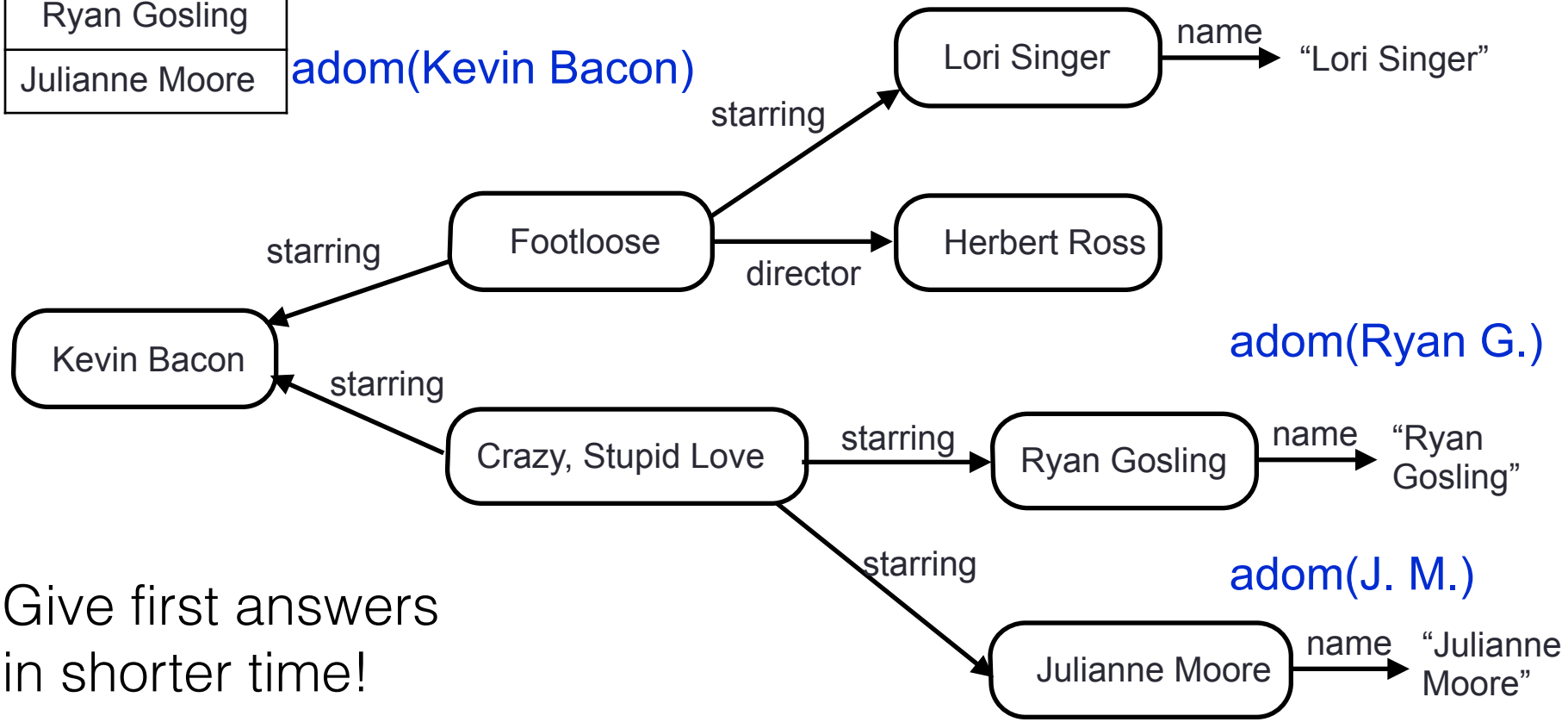
BFS crawling for {Bacon ^starring/starring/name ?x }

?x
Lori Singer
Ryan Gosling
Julianne Moore



DFS crawling for {Bacon ^starring/starring/name ?x }

?x
Lori Singer
Ryan Gosling
Julianne Moore

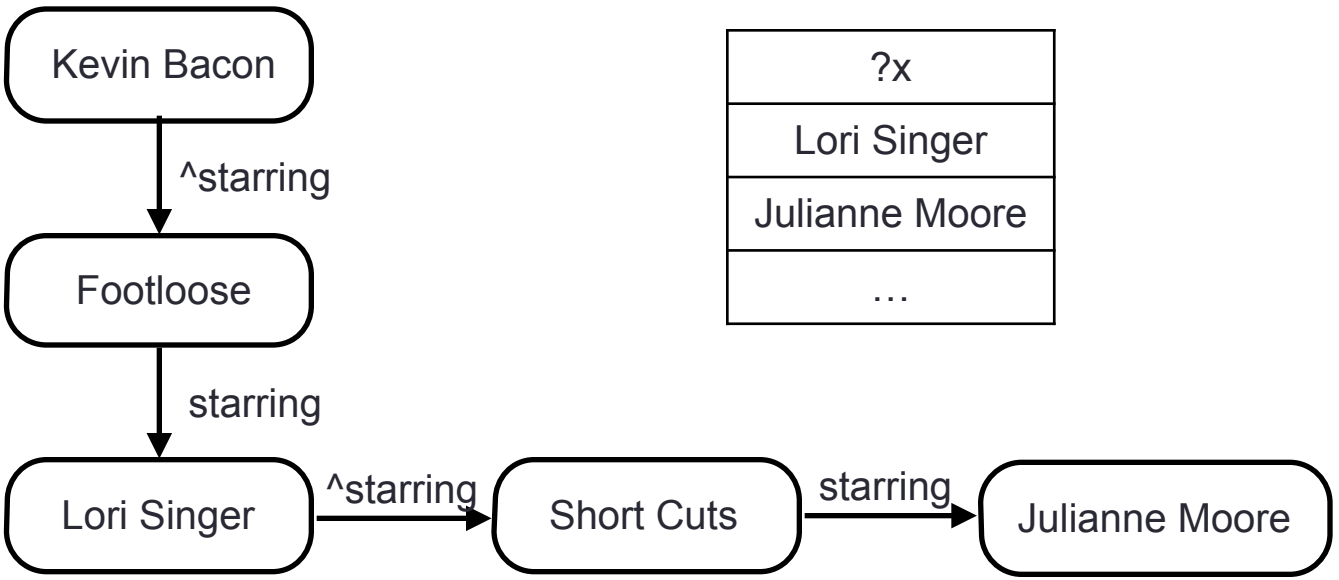


Give first answers in shorter time!

Better option, even better if we just want K answers

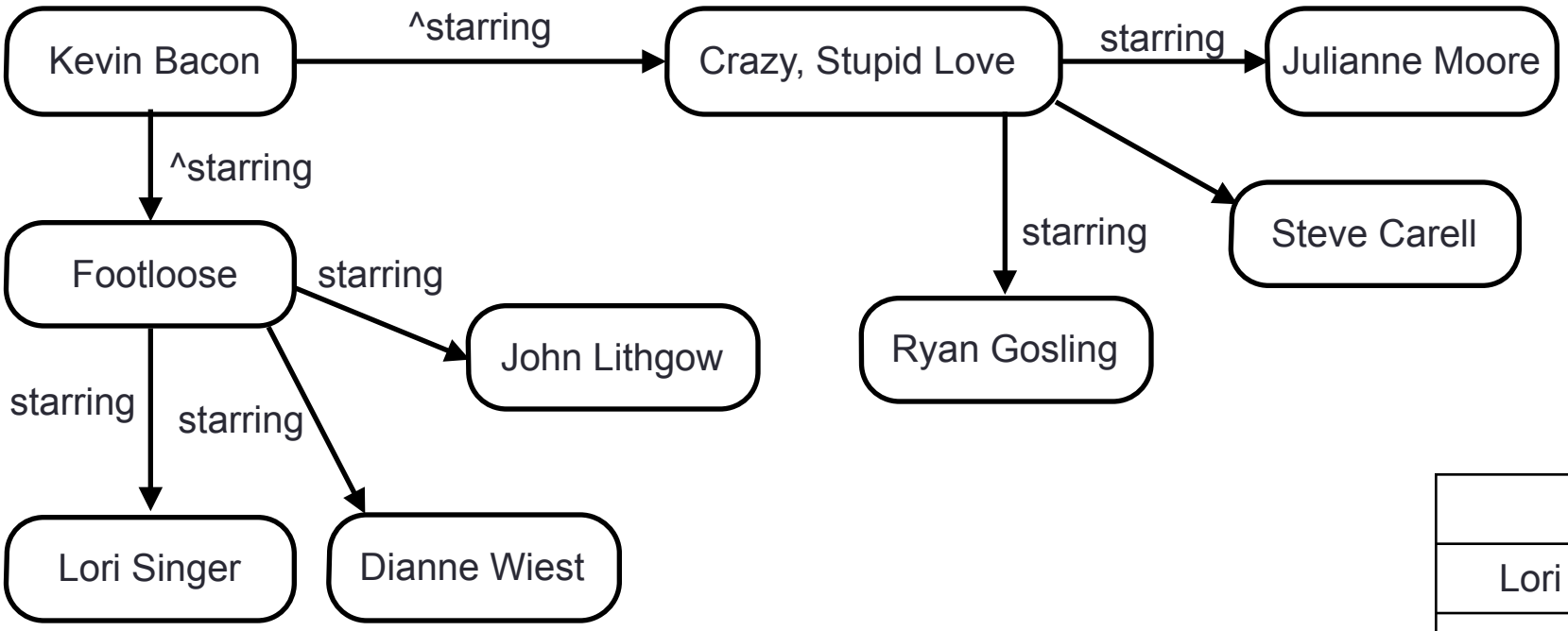
adom(Crazy, Stupid Love)

DFS crawling for {Bacon (^starring/starring)* ?x }



Each new answer requires 2 new requests.
Not optimal

IDS crawling for {Bacon (^starring/starring)* ?x }



?x
Lori Singer
Dianne Wiest
John Lithgow
Ryan Gosling
Steve Carell
Julianne Moore

Exhaust all actors from a movie,
then move to the next one,
repeat

Best option, minimal amount of time to give answers

Generalising IDS for arbitrary property paths

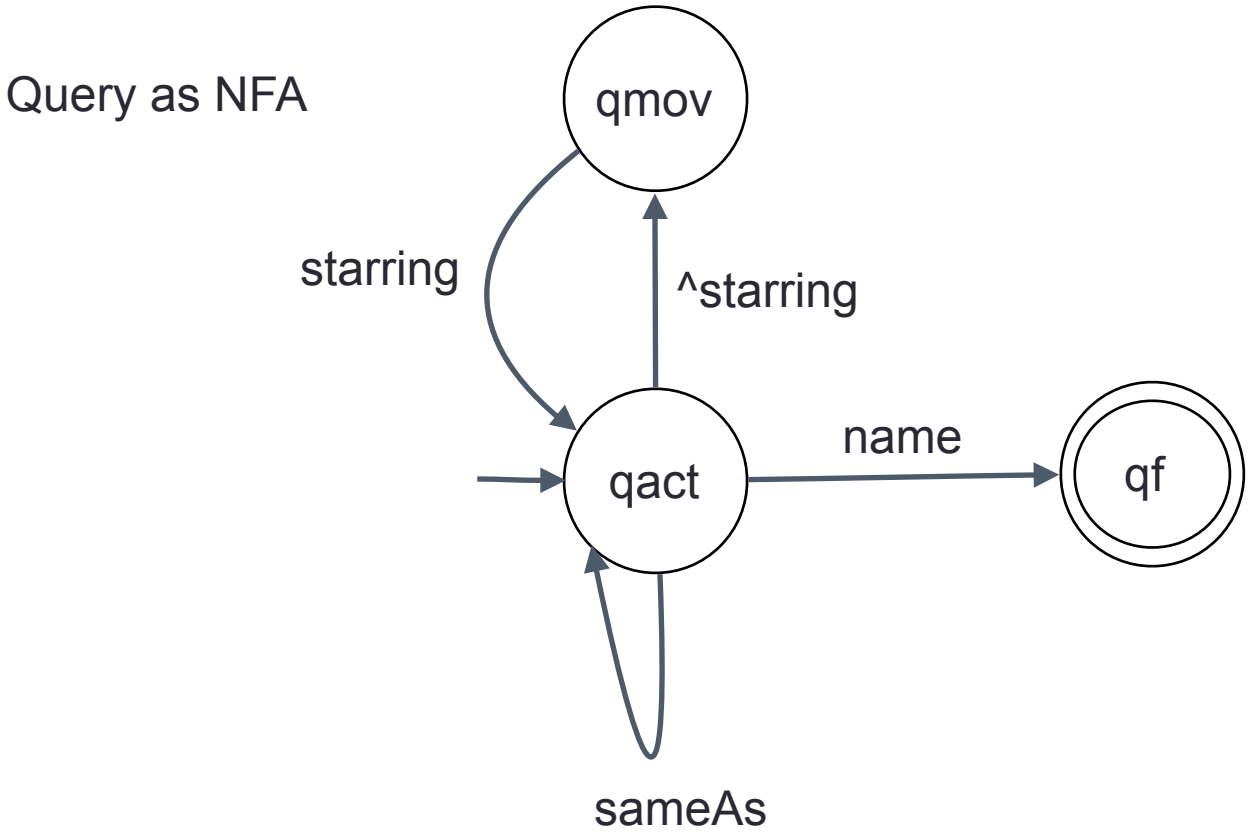
1: Transform Property Path into an automata
2: Assign a state of the automata to each IRI we retrieve
3: Fetch the IRI that would take us closer to a final state

This approach can be formalised as an A* search, over the graph of linked data.

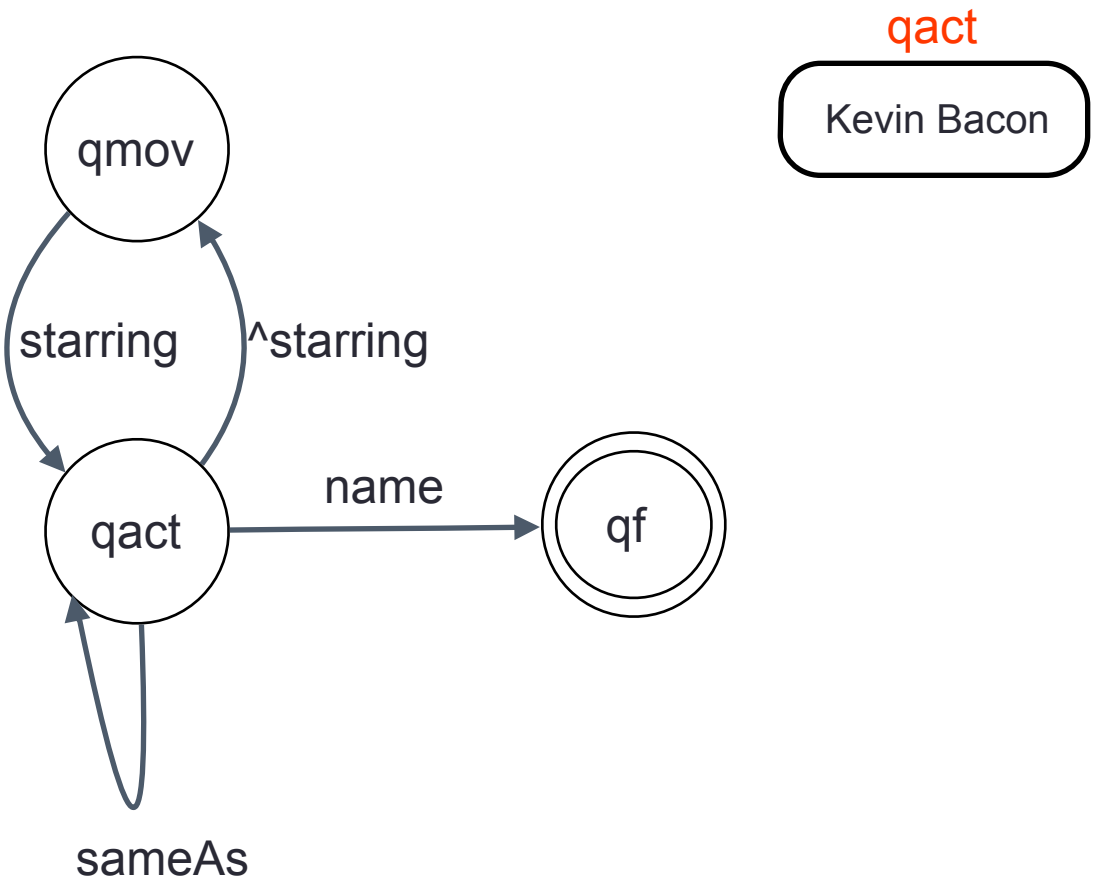
Work in progress!

A* search for crawling the web (example)

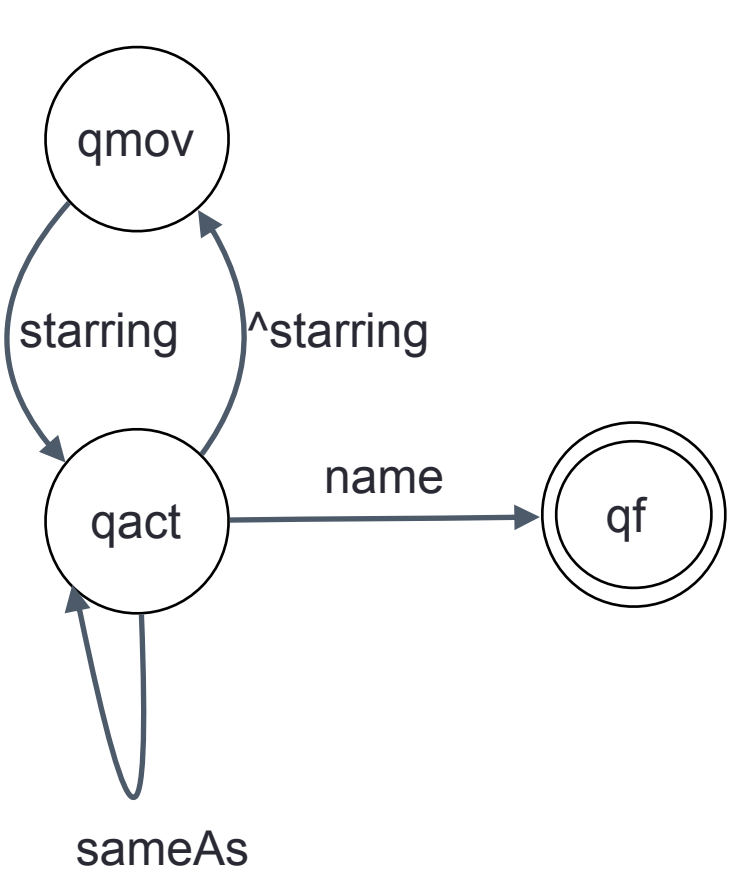
{Bacon (^starring/starring | sameAs)*/name ?x }

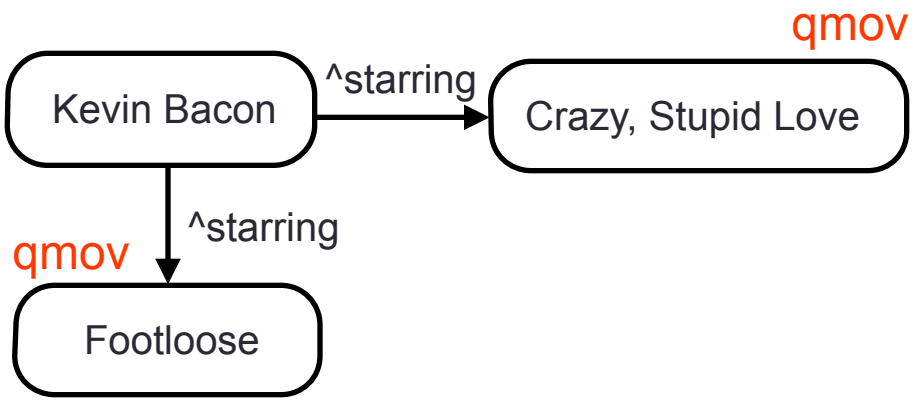


A* search for crawling the web (example)

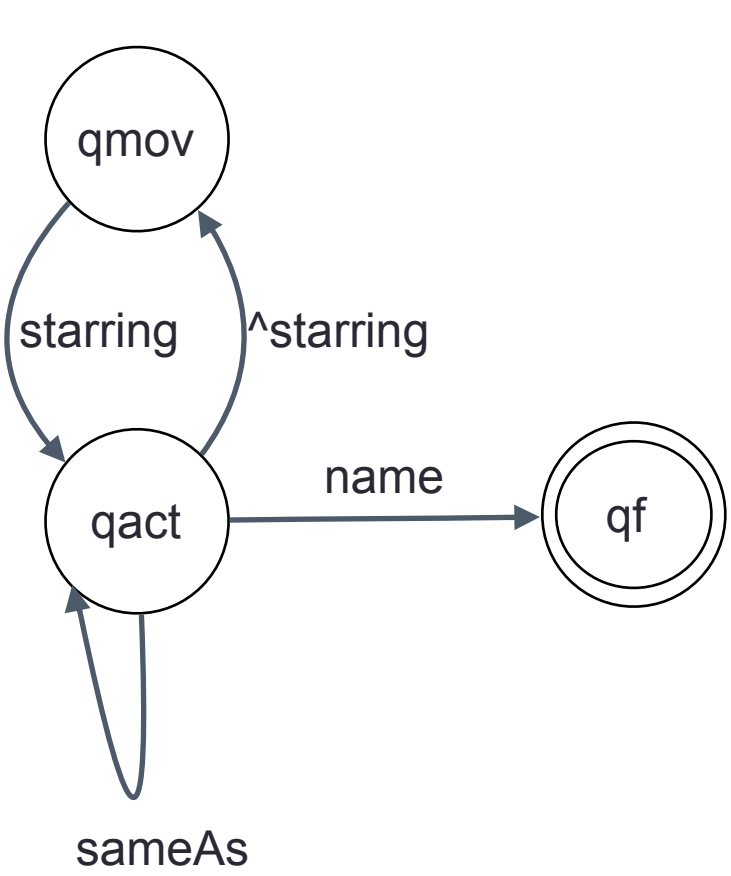


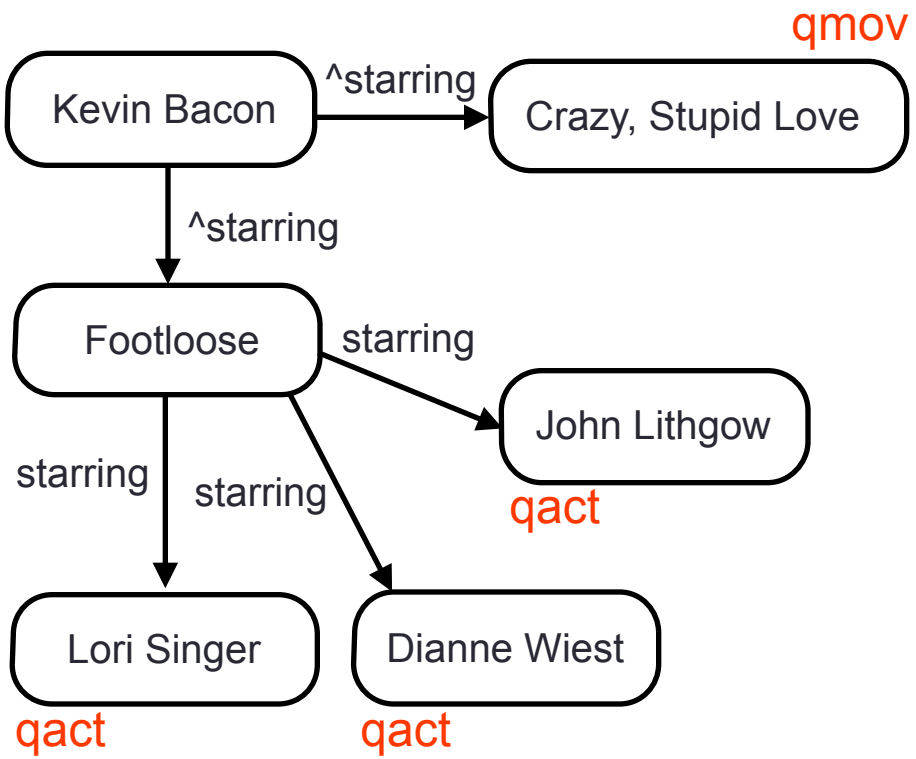
A* search for crawling the web (example)



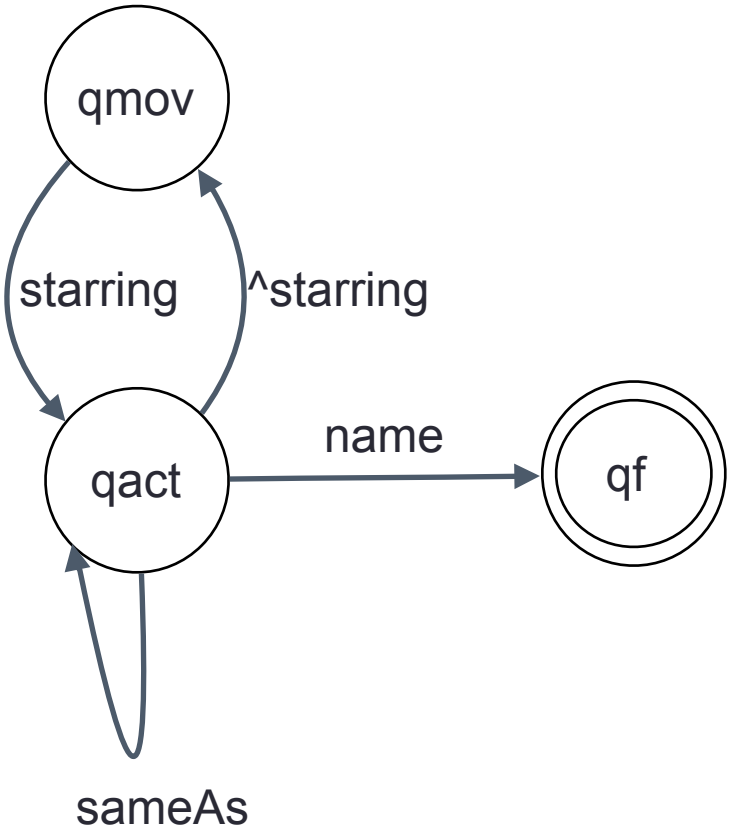


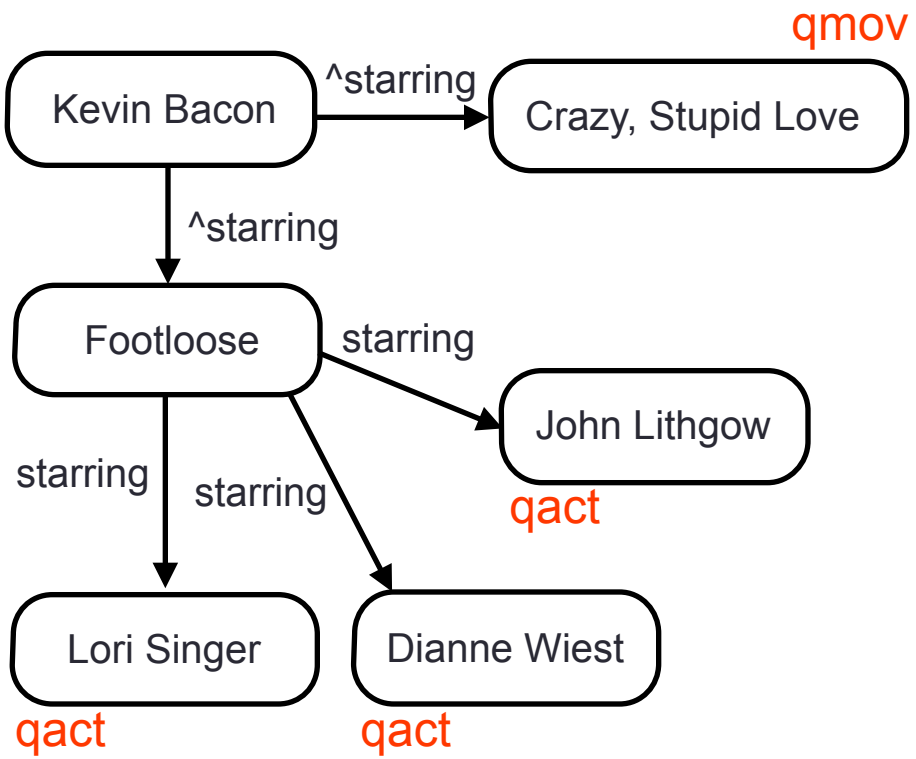
A* search for crawling the web (example)





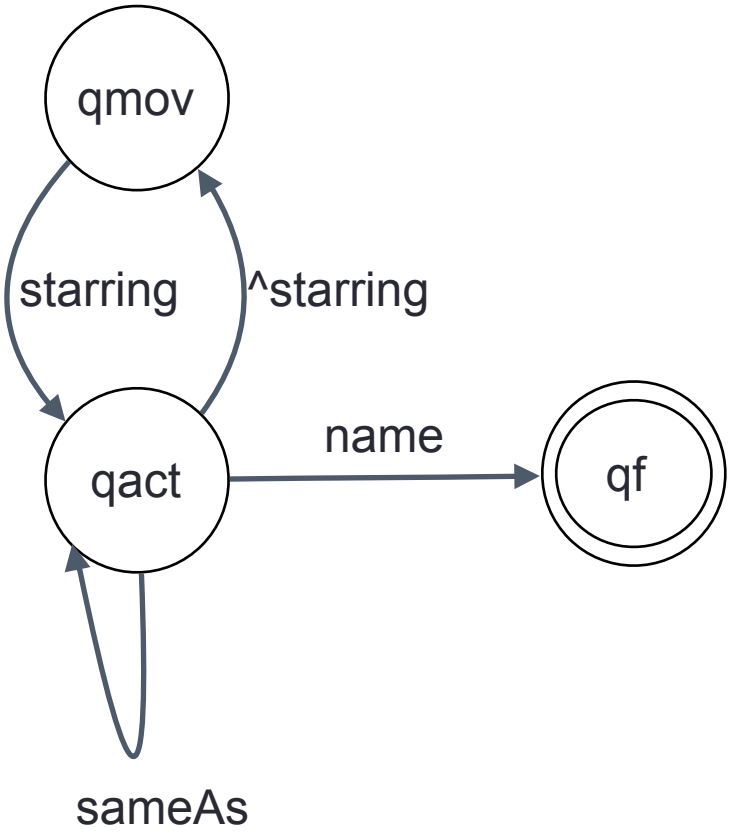
A* search for crawling the web (example)

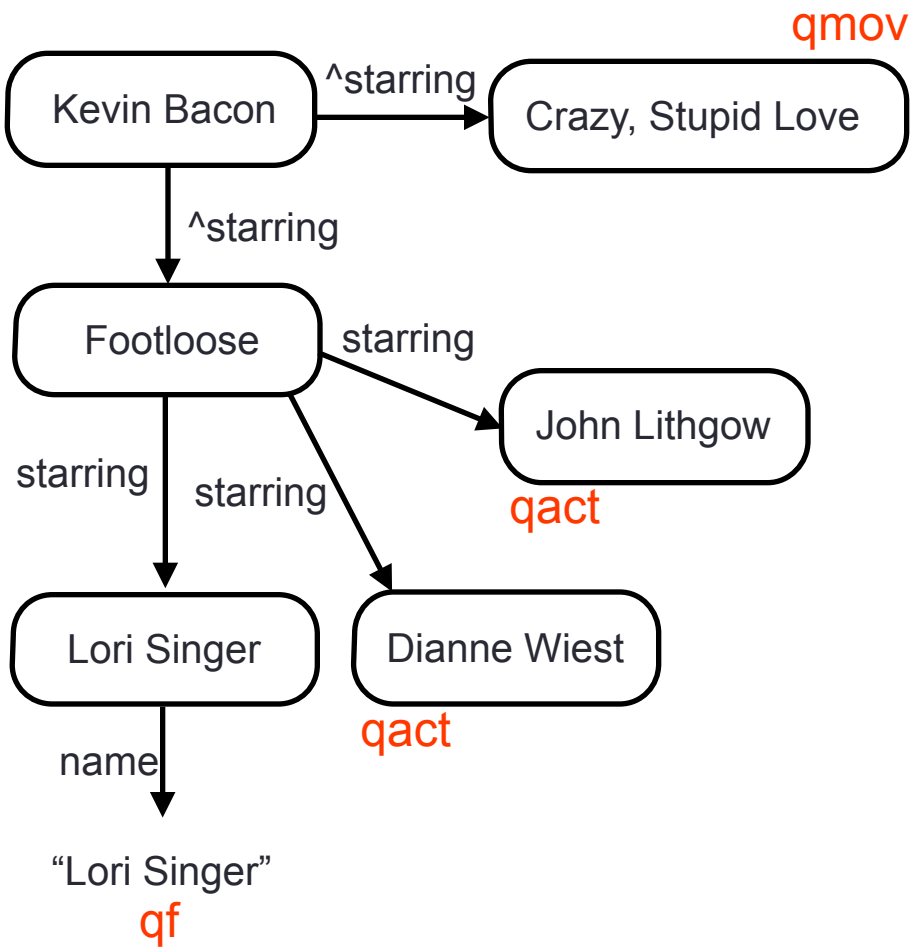




next fetch?
qact is closer to final state!

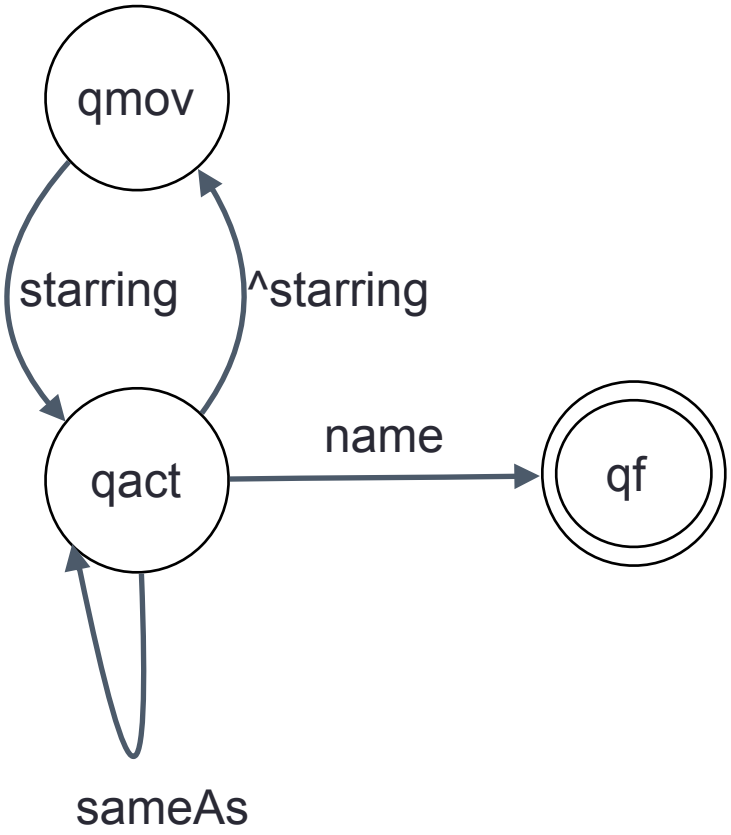
A* search for crawling the web (example)

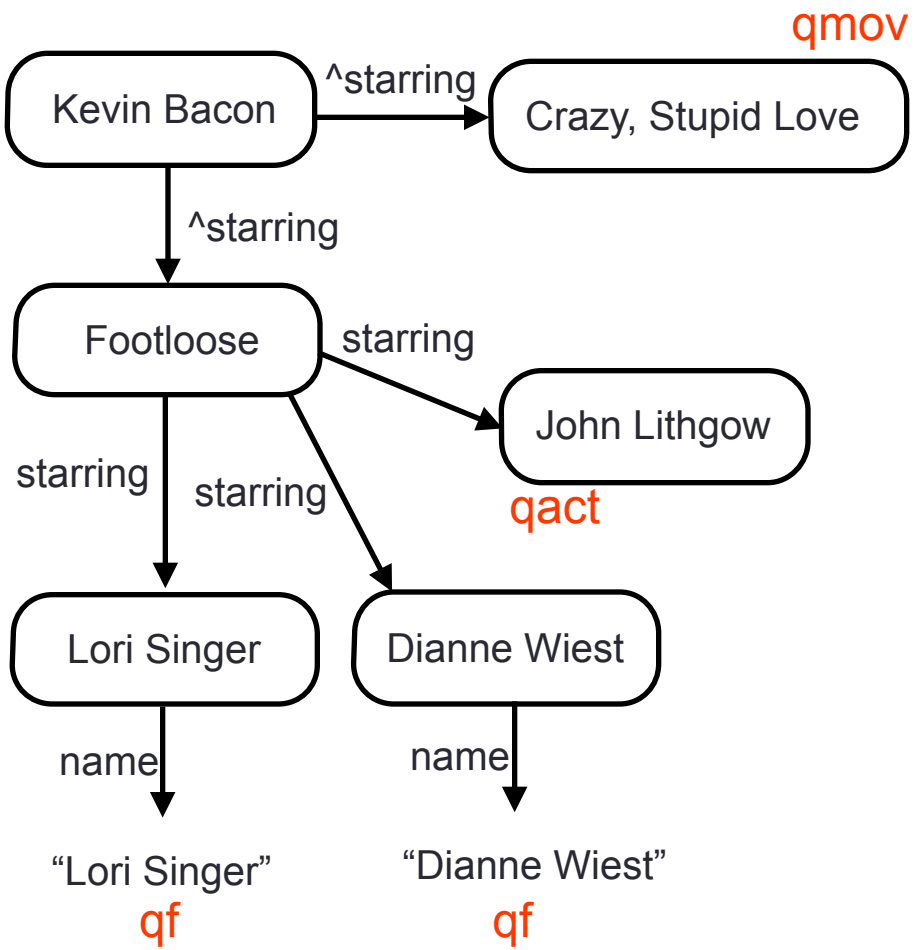




next fetch?
qact is closer to final state!

A* search for crawling the web (example)

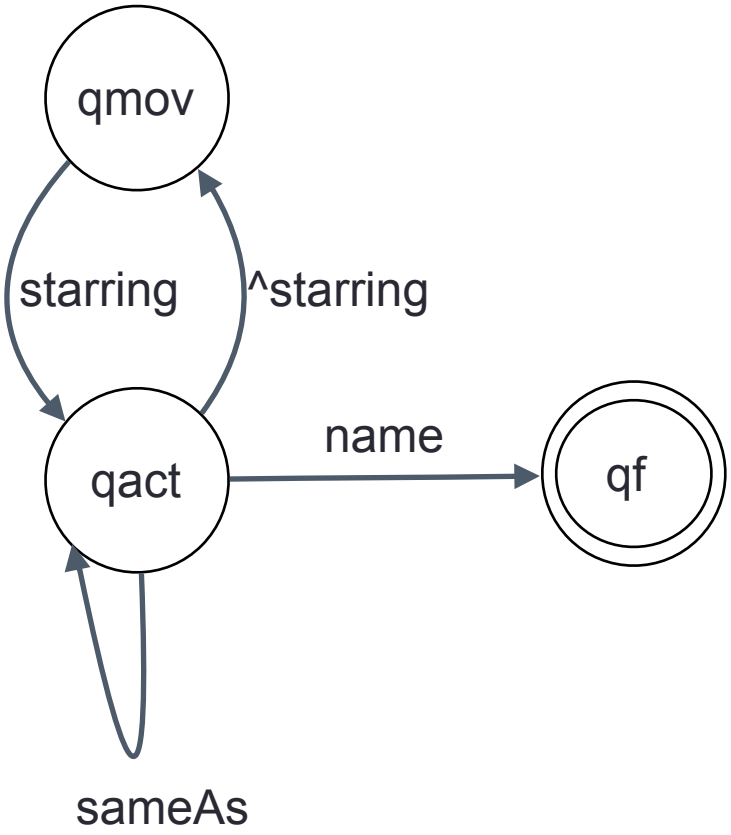


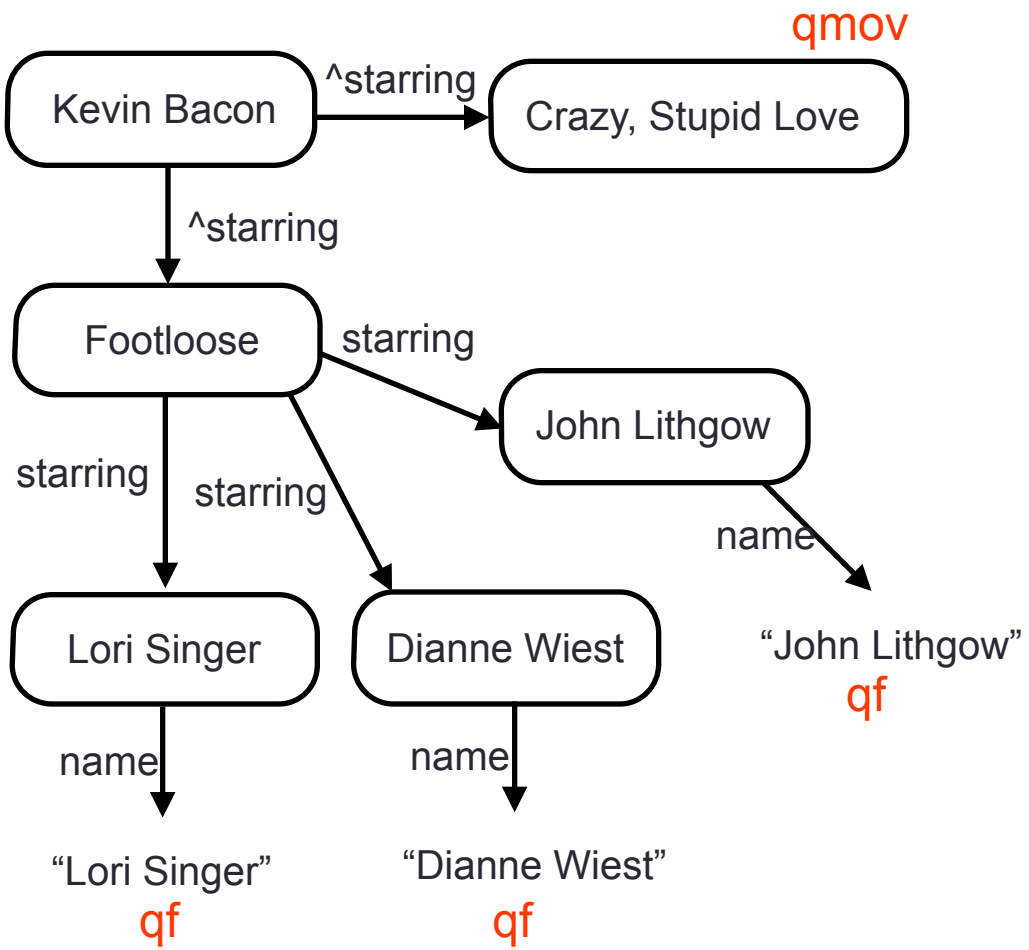


next fetch?

qact is closer to final state!

A* search for crawling the web (example)

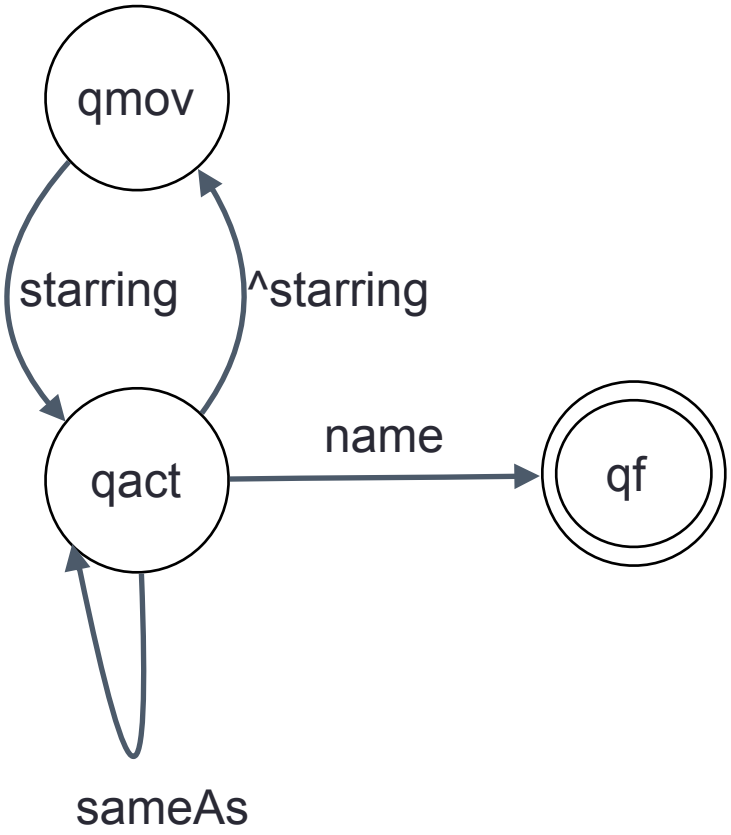


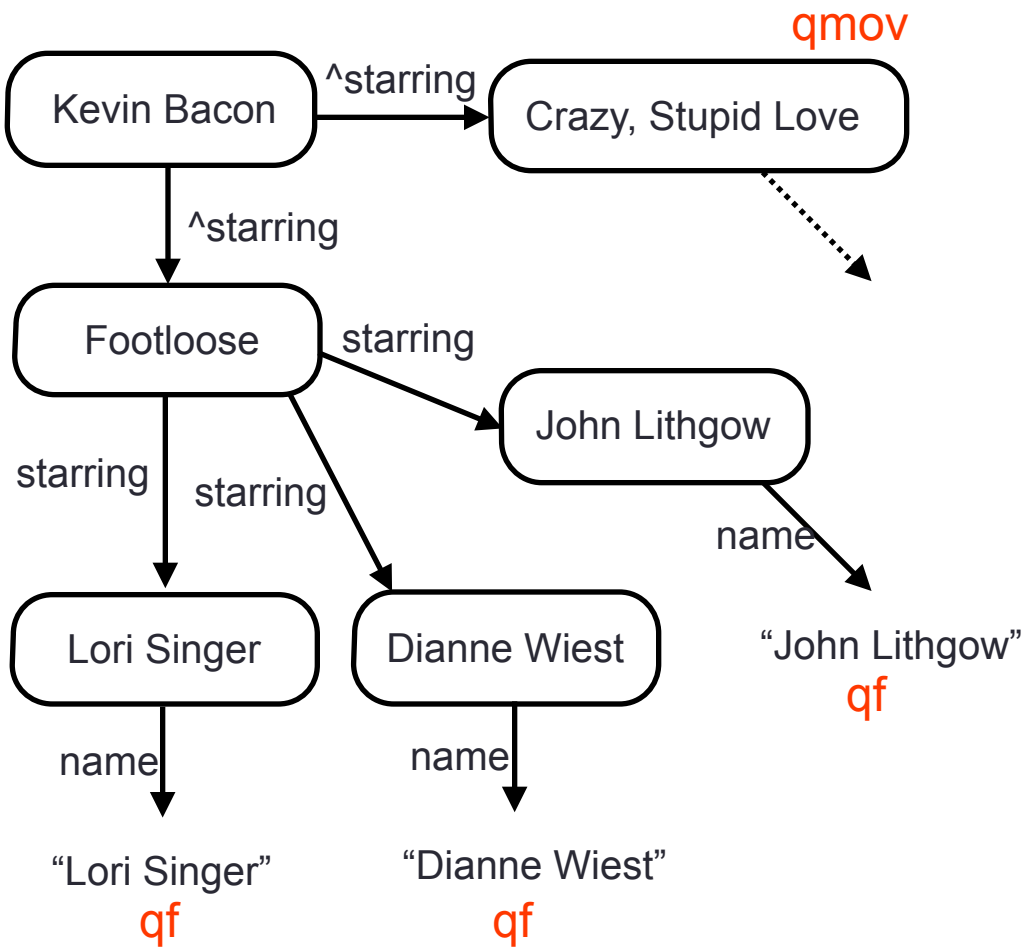


next fetch?

qact is closer to final state!

A* search for crawling the web (example)





now we fetch other movies

A* search for crawling the web

- Complete for justified crawling

- Every new answer is the next shortest path

- Can be shown to beat BFS/DFS/IDS in practice

A* search for crawling the web

- Complete for justified crawling

- Every new answer is the closer to our starting point

- Can be shown to beat BFS/DFS/IDS in practice

how does one compare these algorithms?

Outline

How to actually do it (algorithms)

Example:
Property Paths

Comparing Algorithms

Comparing algorithms

Algorithm A and algorithm B.
Which one is better?

Comparing algorithms

Algorithm A and algorithm B.
Which one is better?

Easy case: both algorithms terminate,
(so both give the same answers)

Algorithm A and algorithm B.
Which one is better?

Easy case: both algorithms terminate,
(so both give the same answers)

Better algorithm:

Uses less resources to give the same # of answers

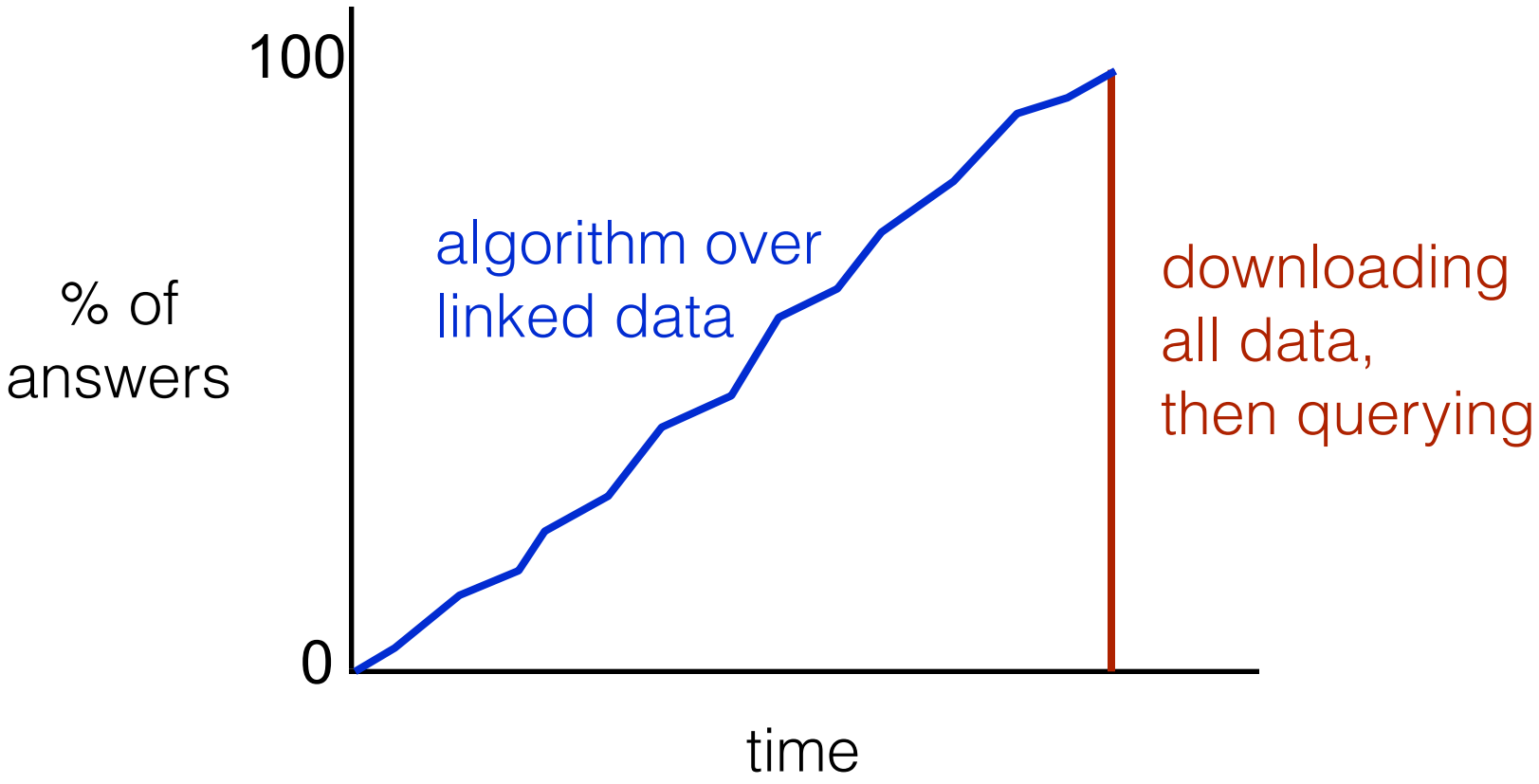
Algorithm A and algorithm B.
Which one is better?

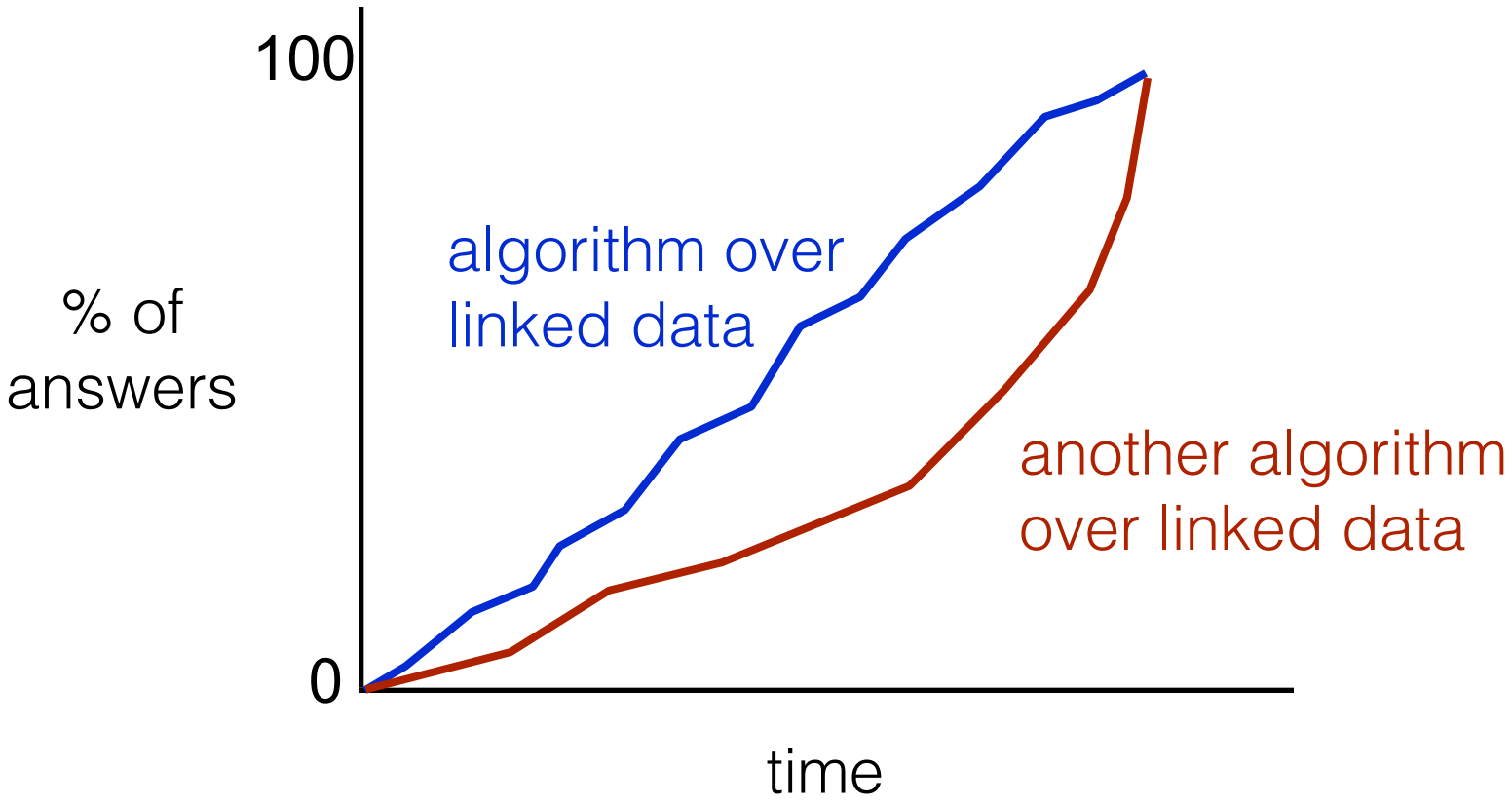
Easy case: both algorithms terminate,
(so both give the same answers)

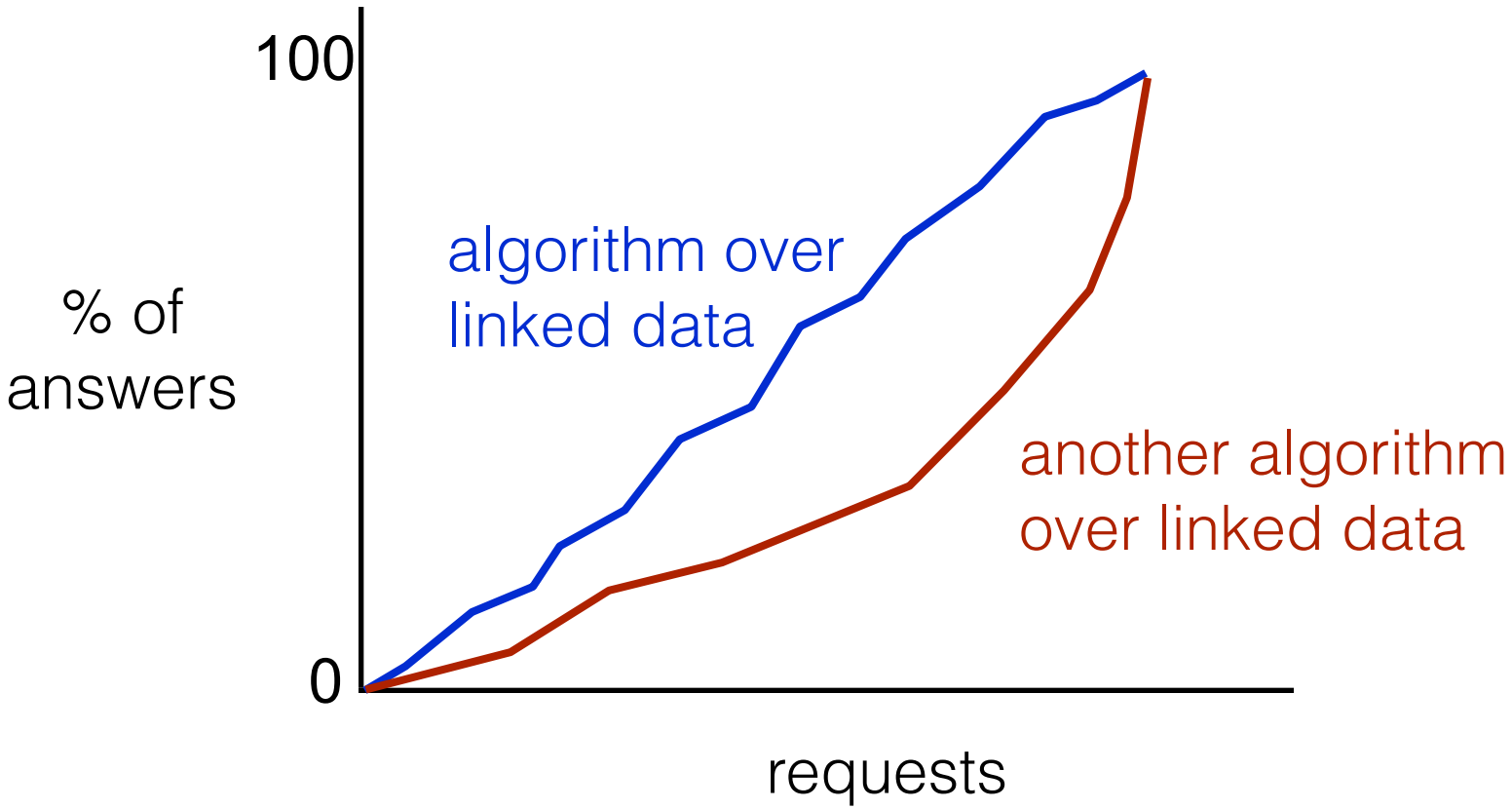
Better algorithm:

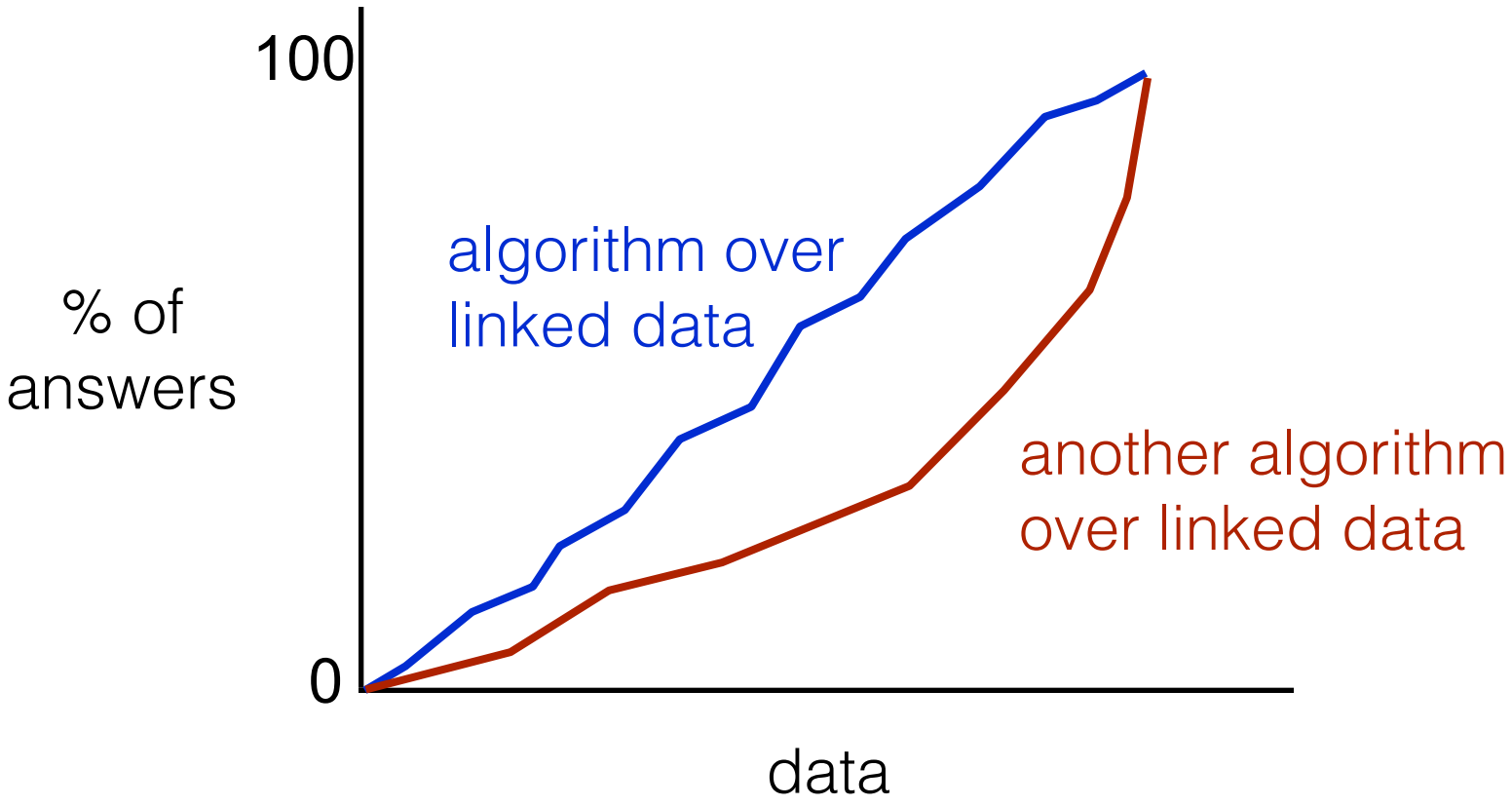
Uses less resources to give the same # of answers

time data transmitted #of requests


100
0
% of answers
time
algorithm over linked data
downloading all data, then querying


100
0
% of answers
time
algorithm over linked data
another algorithm over linked data

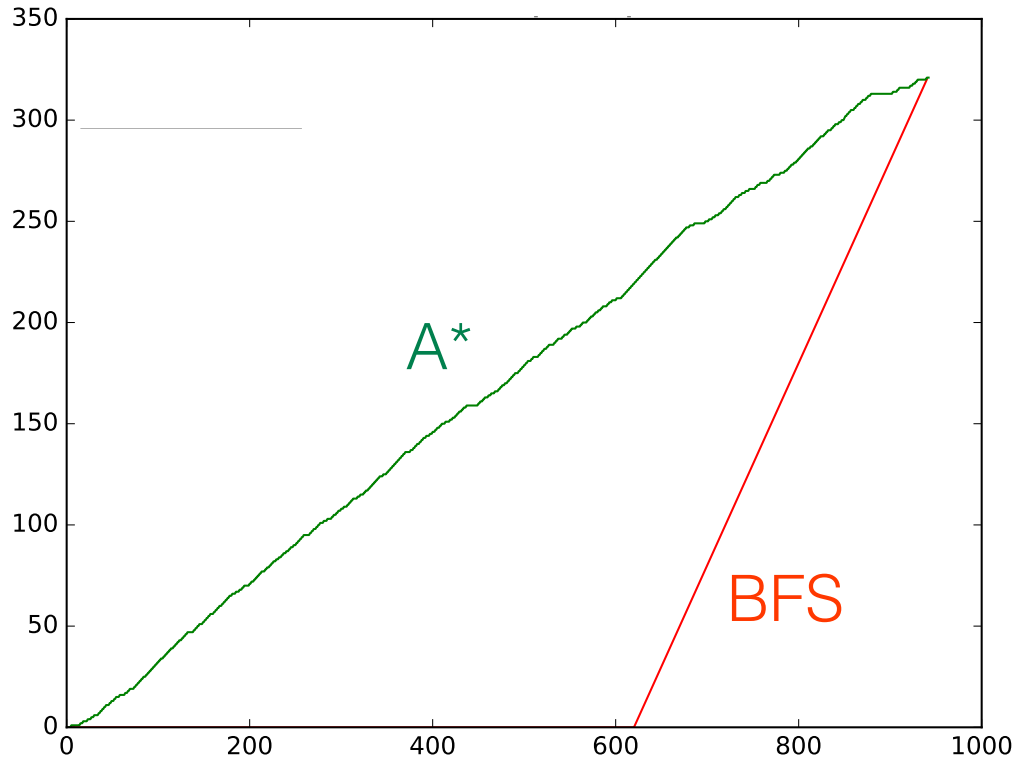

100
0
% of answers
requests
algorithm over linked data
another algorithm over linked data


100
% of answers
0
algorithm over linked data
another algorithm over linked data
data

So… coming back to A* search for crawling the web

- Can be shown to beat BFS/DFS/IDS in practice

answers

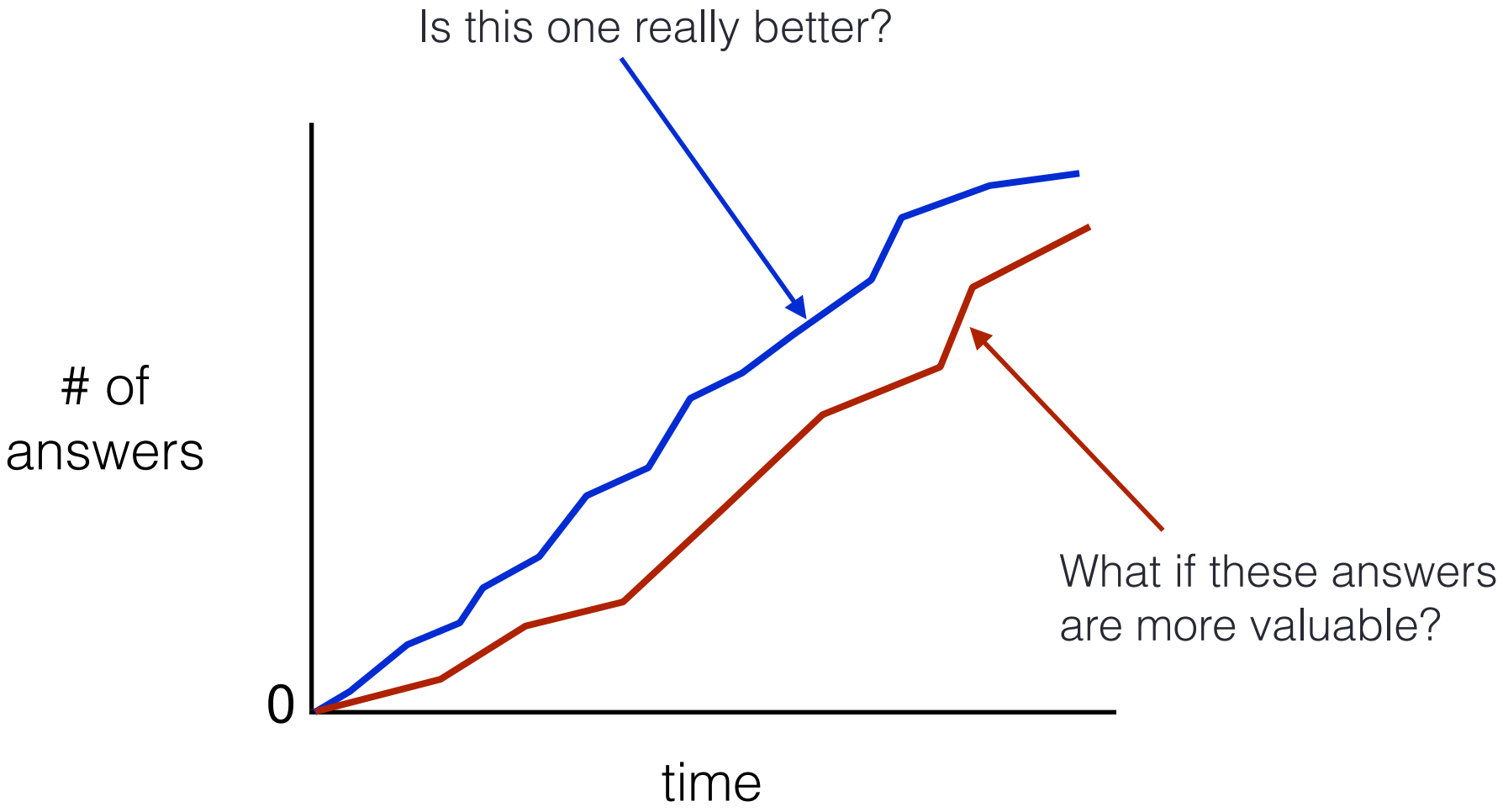


requests

coauthors in DBLP over linked data

Algorithm A and algorithm B.
Which one is better?

Not so easy case: algorithms do not terminate,
(so they don't give the same answers!)



How do I measure which answer is more important?

Leverage the use of ontologies? meta-information?

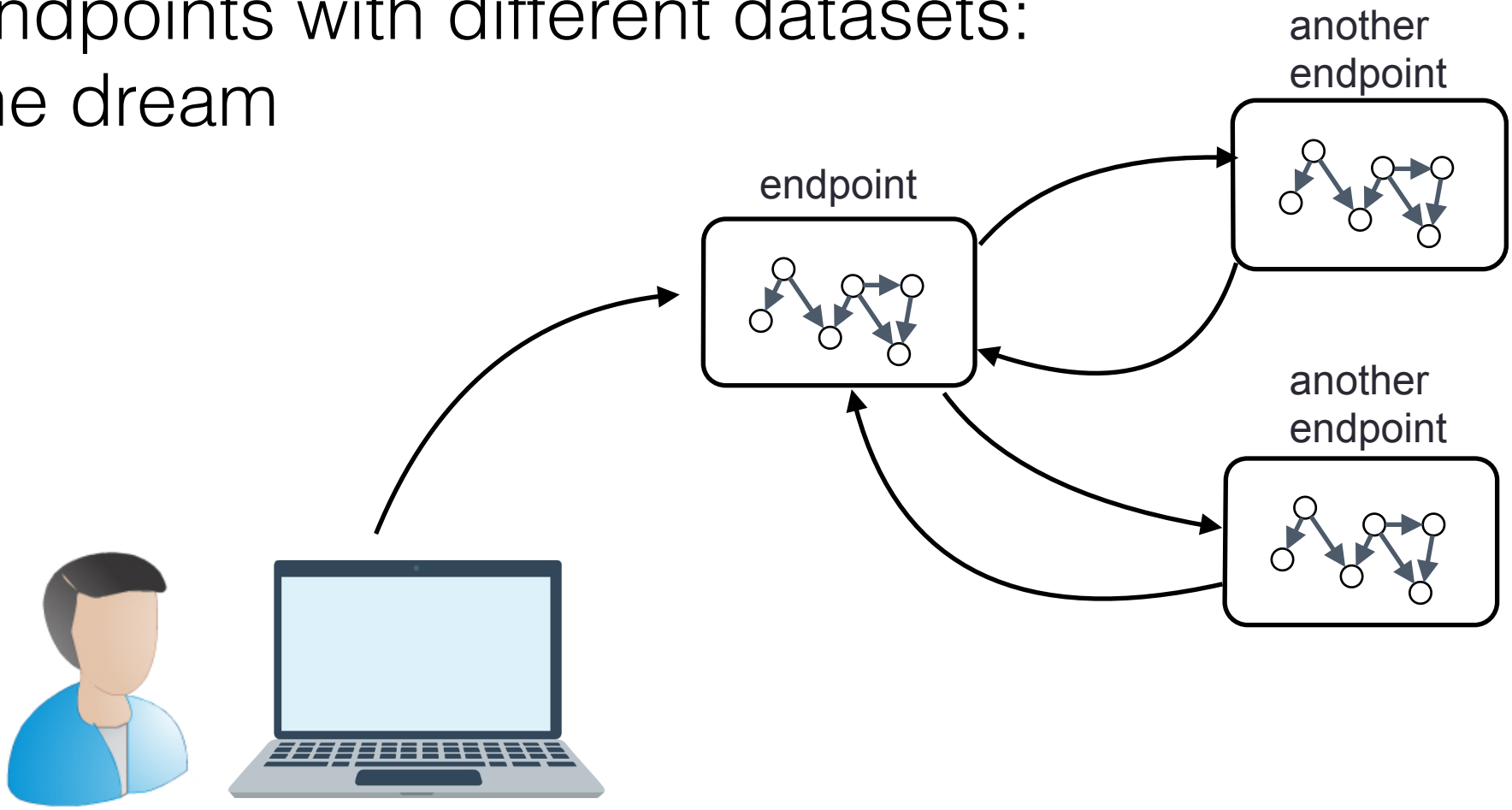
Outline

What does it mean to query the web (semantics)

How to actually do it (algorithms)

SERVICE on Endpoints

Endpoints with different datasets: the dream



Need to query these 3 endpoints.

SPARQL standard: use SERVICE operator!

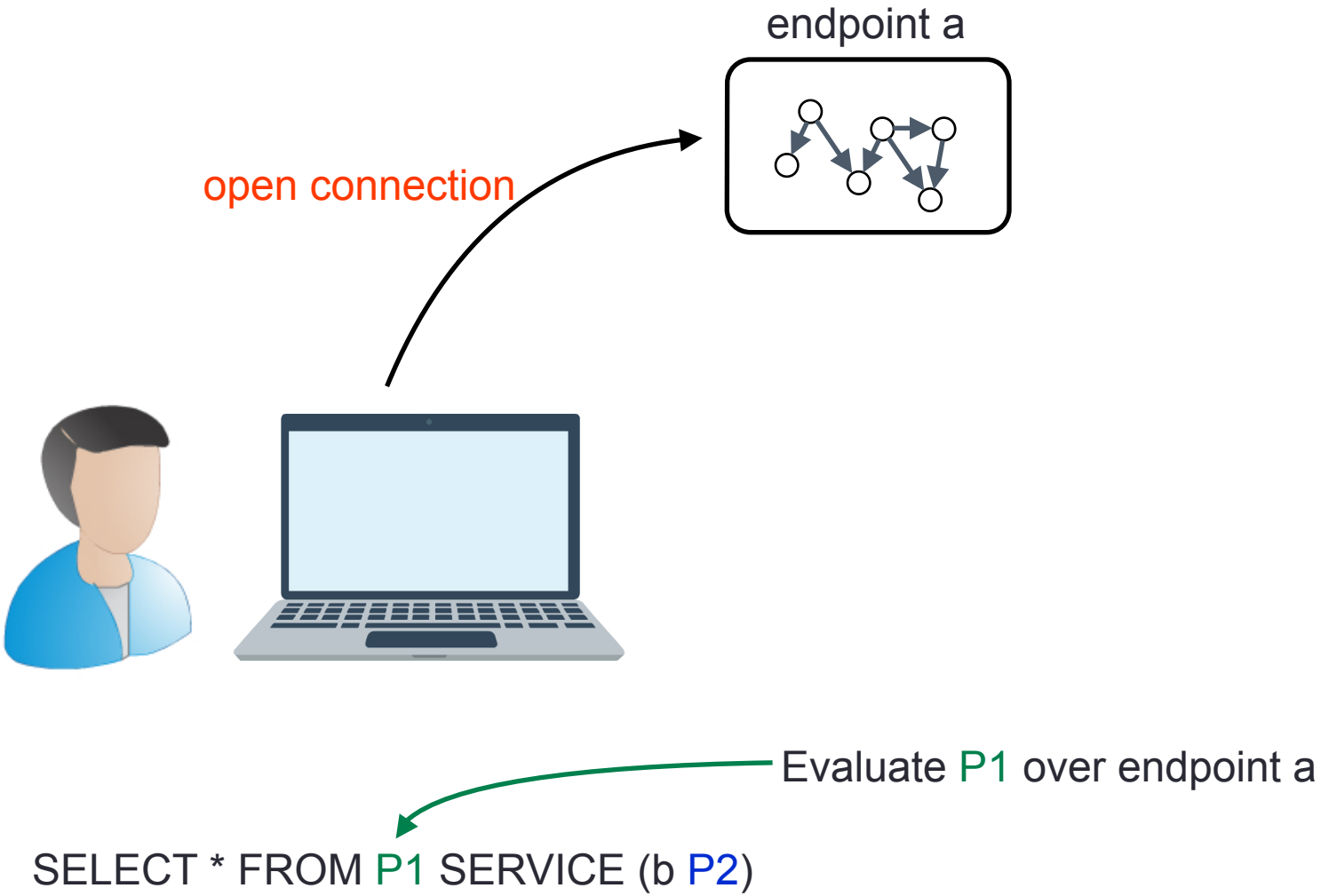
In practice, almost none (none?) endpoints implement SERVICE operator

Why?
What have we done about it?

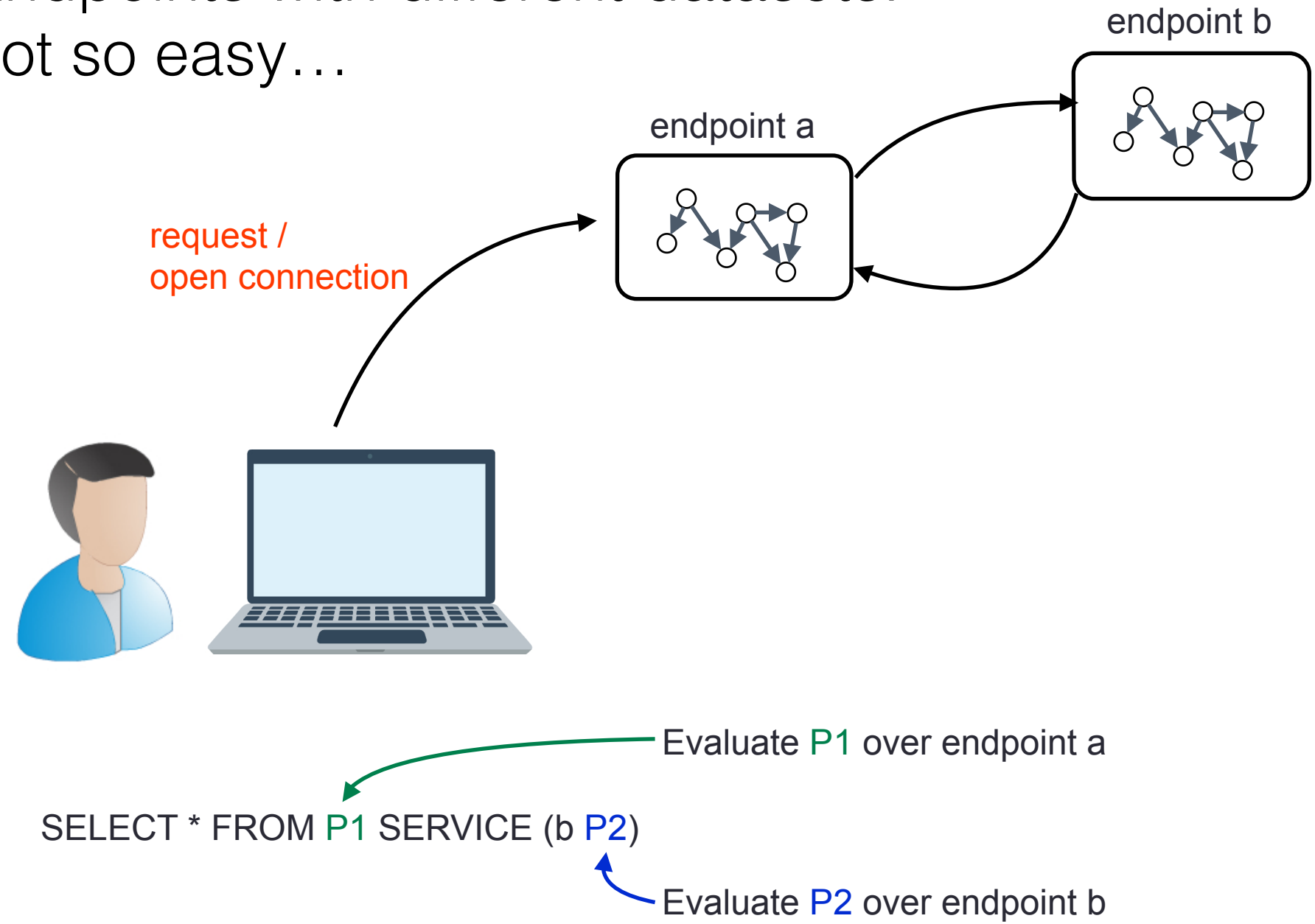
Endpoints with different datasets: not so easy…

SELECT * FROM P1 SERVICE (b P2)

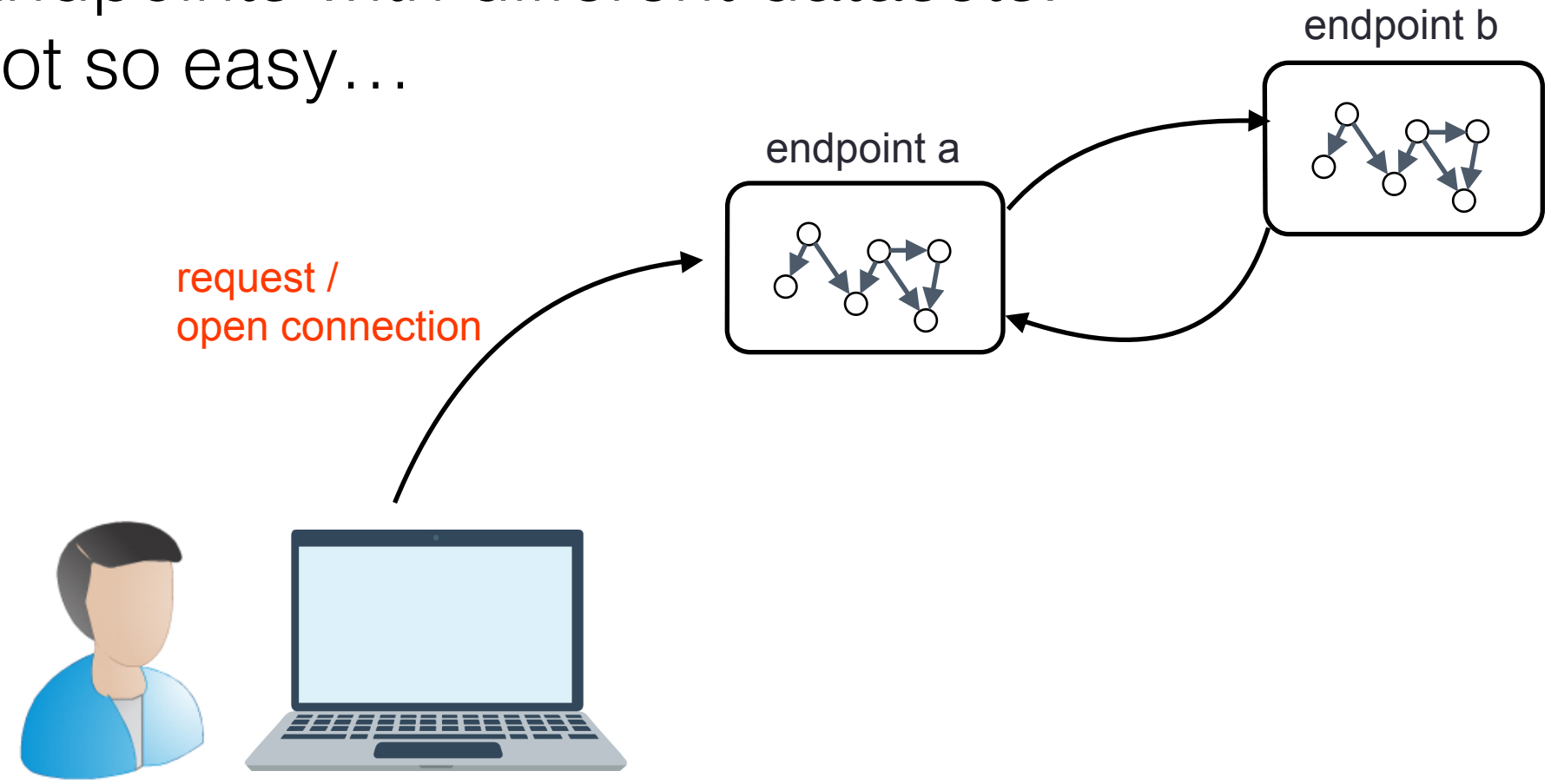
Endpoints with different datasets: not so easy…



Endpoints with different datasets: not so easy…



Endpoints with different datasets: not so easy…



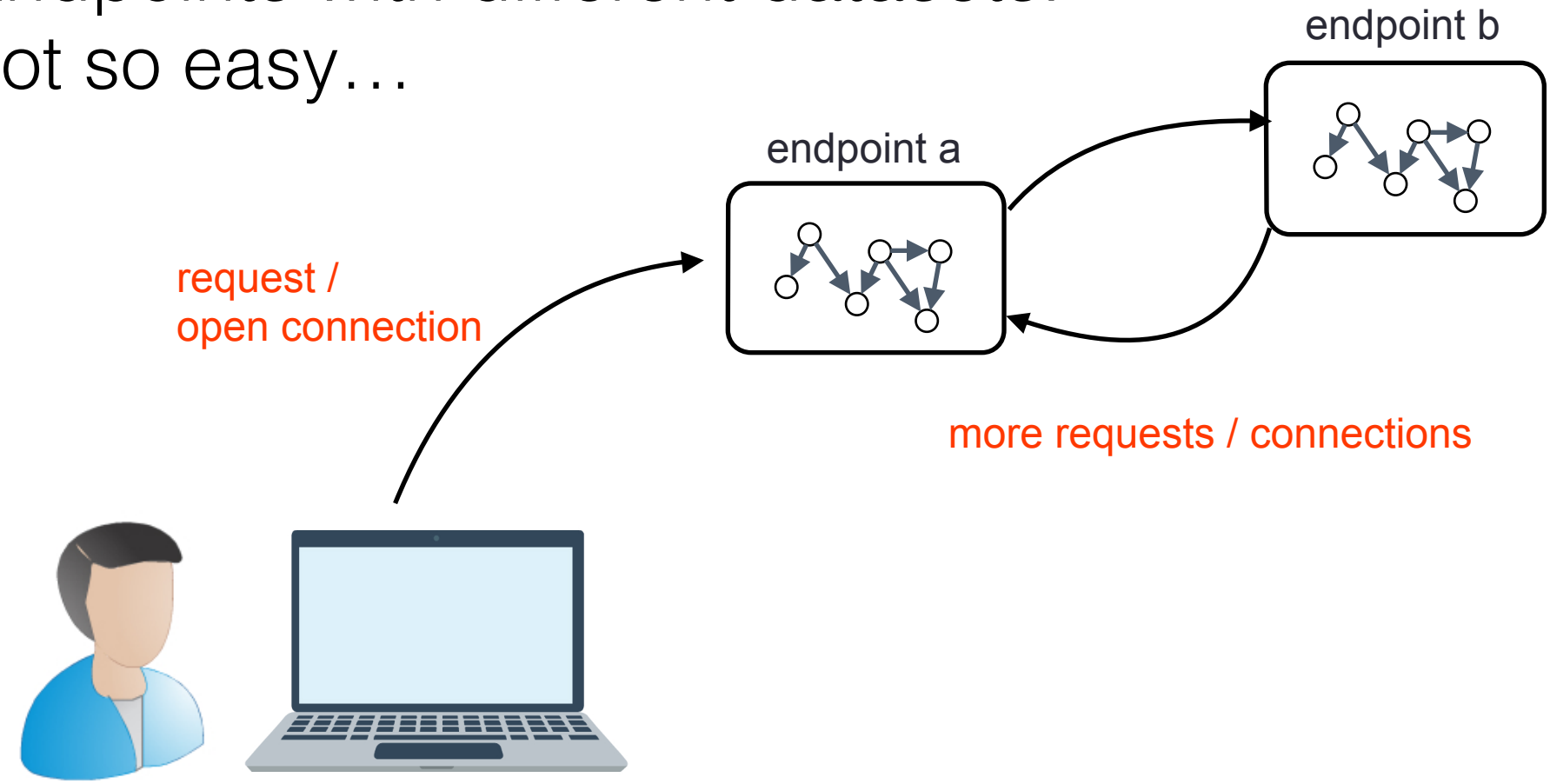
Evaluate P1 over endpoint a

SELECT * FROM P1 SERVICE (b P2)

compute P1 ⋈ P2

Evaluate P2 over endpoint b

Endpoints with different datasets: not so easy…



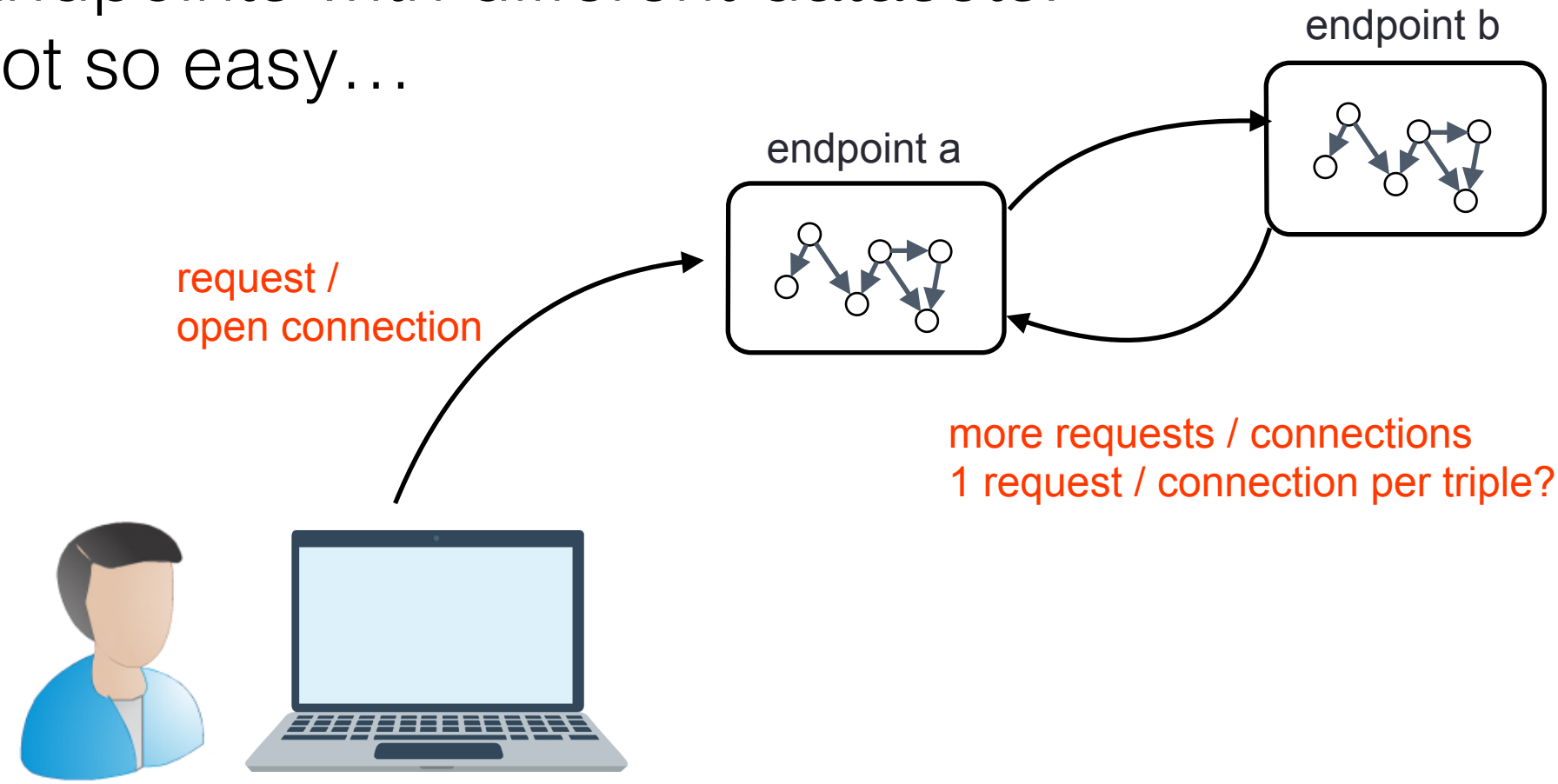
Evaluate P1 over endpoint a

SELECT * FROM P1 SERVICE (b P2)

compute P1 ⋈ P2

Evaluate P2 over endpoint b

Endpoints with different datasets: not so easy…



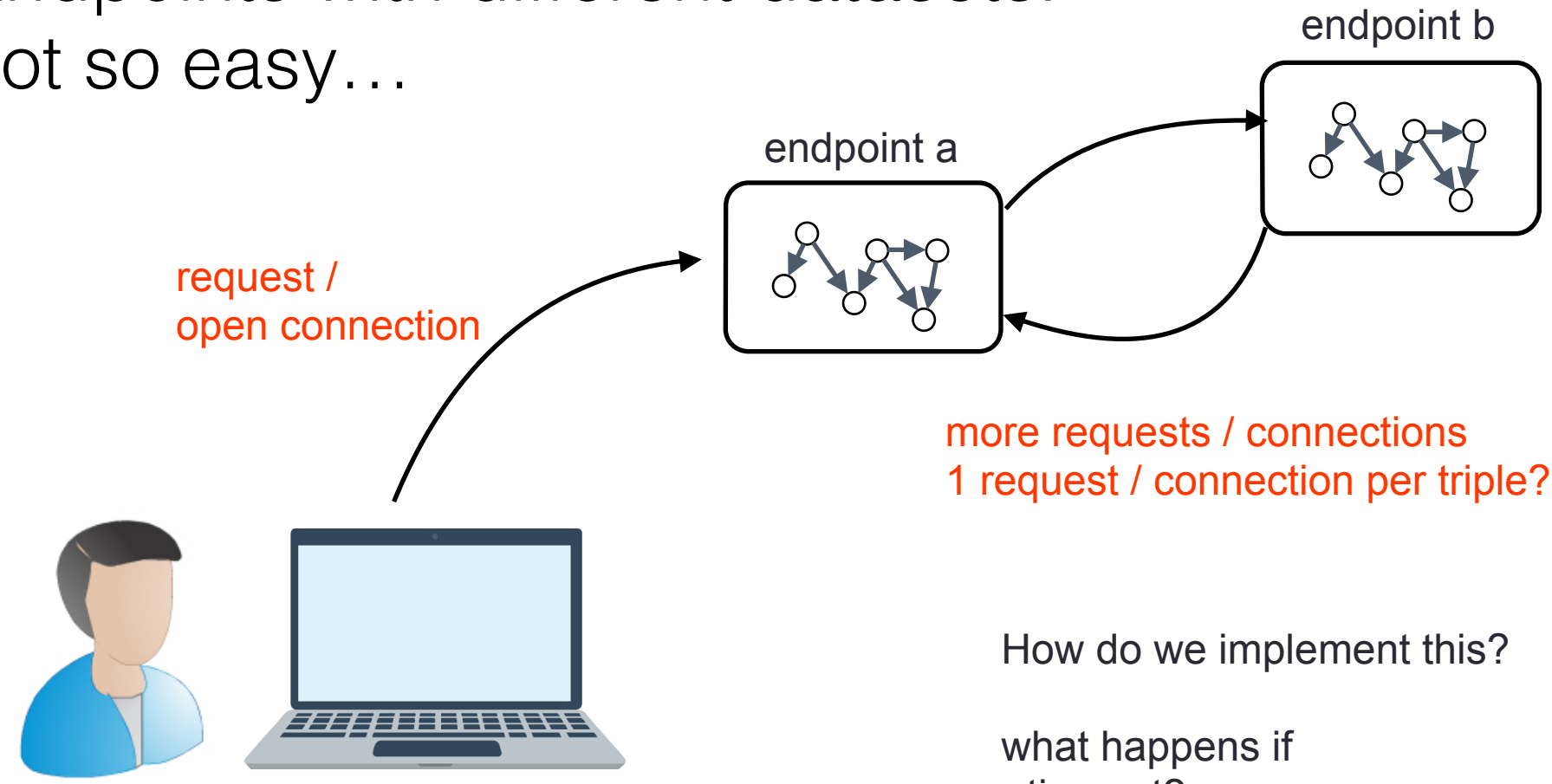
Evaluate P1 over endpoint a

SELECT * FROM P1 SERVICE (b P2)

compute P1 ⋈ P2

Evaluate P2 over endpoint b

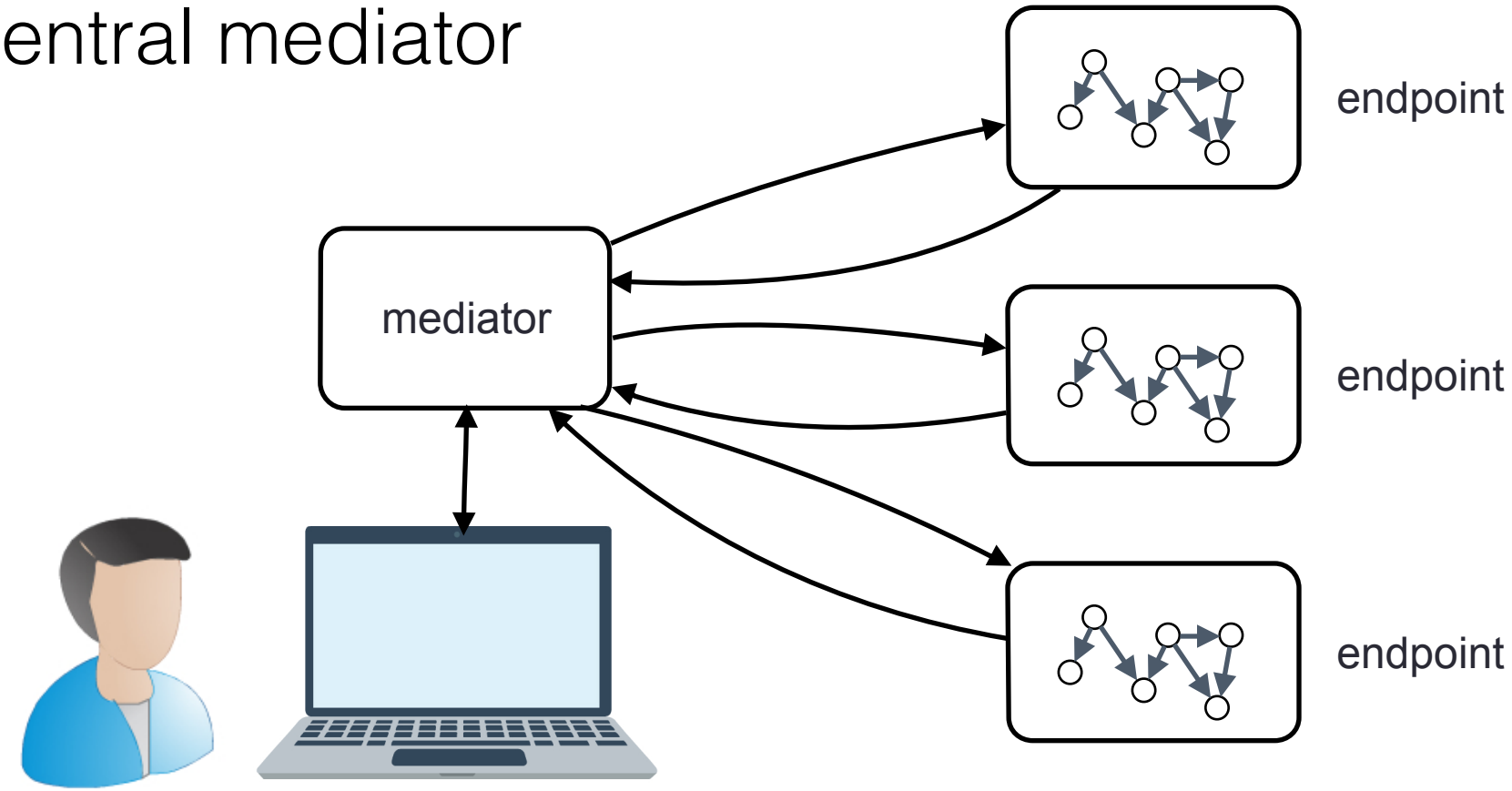
Endpoints with different datasets: not so easy…



How do we implement this?

what happens if
- timeout?
- refuse?
- endpoint is down?
- query takes too long?

Typical solution: central mediator

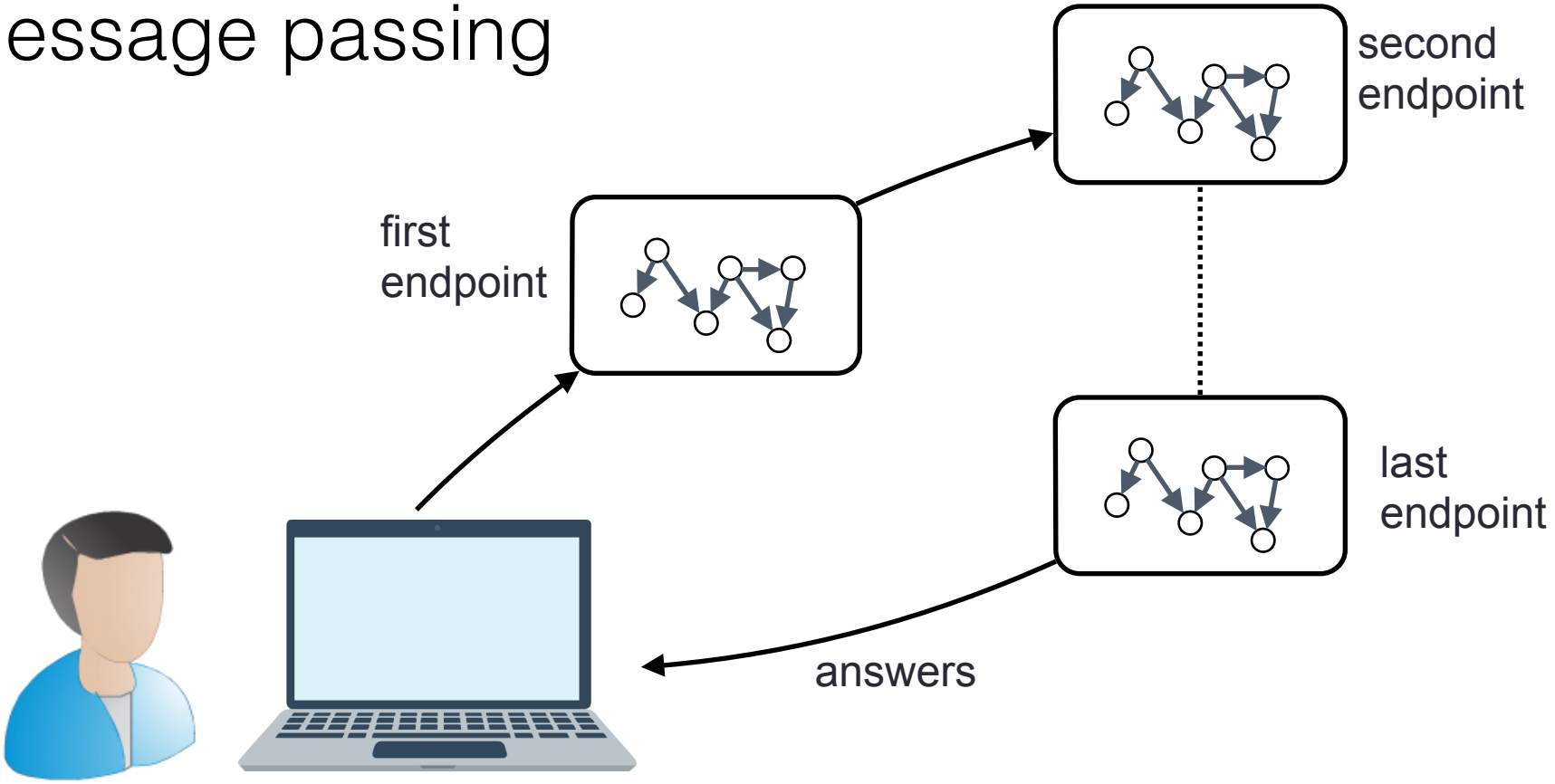


Go to Querying/SPARQL(I) Session in Thursday for a good survey of the techniques

However…

How far are we from running SERVICE queries in endpoints?

Other possible solution: Message passing



Problems:
- query planning
- query optimisation
- tolerance over failure
- any publicly available implementation?

Looking Forward

We should…

take risks,
stop worrying about the future of linked data
work on realising the dream

We should in particular

- Understand the fundamentals behind querying linked data

We should in particular

- Understand the fundamentals behind querying linked data
  even if there are not many datasets
  even if it is still difficult to write these queries

We should in particular

- Understand the fundamentals behind querying linked data

- Develop, implement and compare algorithms that can do this

We should in particular

- Understand the fundamentals behind querying linked data

- Develop, implement and compare algorithms that can do this
  even if for now it is just a research exercise
  won't find out about users if there is nothing to show

We should in particular

- Understand the fundamentals behind querying linked data

- Develop, implement and compare algorithms that can do this

- Implement SERVICE in endpoints

We should in particular

- Understand the fundamentals behind querying linked data

- Develop, implement and compare algorithms that can do this

- Implement SERVICE in endpoints
  even if it demands a lot of resources from providers
  even if most endpoints are not stable

Some solutions may be even used in other areas!

case in point:

Our algorithm for computing property paths in linked data can be used to compute path queries in graph databases

Some solutions may be even used in other areas!

case in point:

Our algorithm for computing property paths in linked data can be used to compute path queries in graph databases

- we are currently working on a system that implements “linked data crawling” on local databases

- when we just want a few answers, this approach beats graph databases (Neo4j, SPARQL DBs, TitanDB…)

The Fascinating World of Querying Linked Data

Questions?