Part 2: The Prolog Web, etc.

Torbjörn Lager

Department of Philosophy, Linguistics and
Theory of Science, University of Gothenburg

Outline

- The Prolog Web
- Two kinds of Web APIs
- The Prolog Web and the Semantic Web
- Voice UIs for talking to the Prolog Web
- Statecharts, Web Prolog and the Prolog Web
- The crisis of Prolog
- Rebranding Prolog
- Standardising Web Prolog
- Celebrating Prolog


Web Prolog
and the programmable
Prolog Web

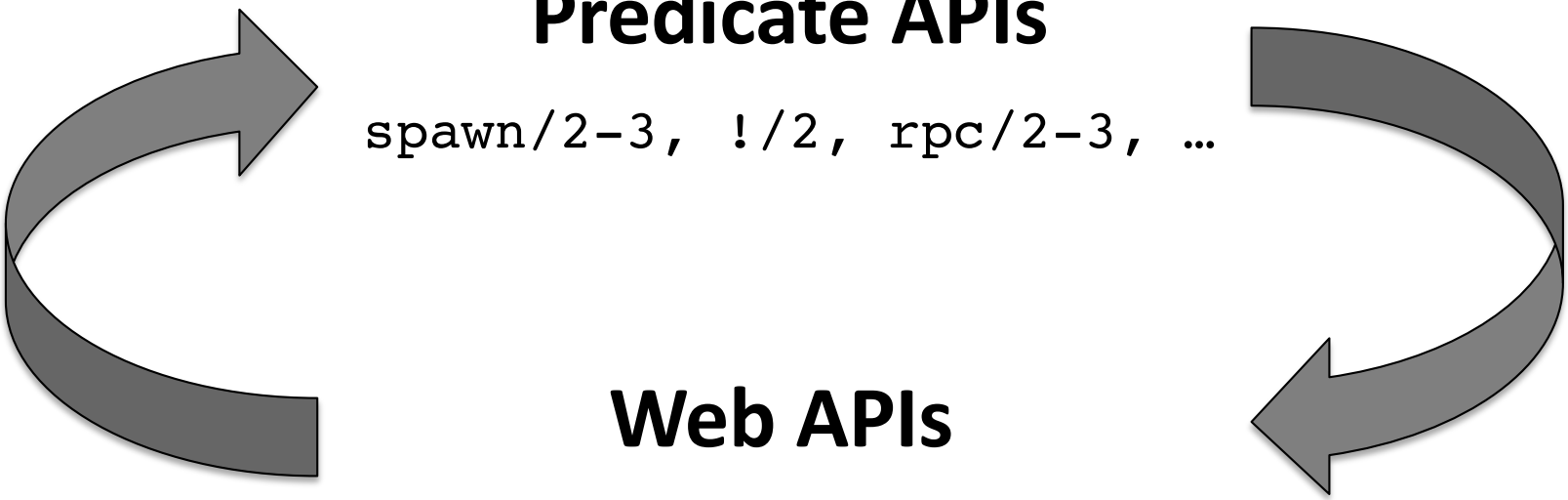
The Prolog Web – the elevator pitch

Imagine **the Web wrapped in Prolog**, running on a distributed architecture comprising a network of **nodes** supporting **RESTful HTTP APIs** and **WebSocket APIs**, as well as web formats such as **JSON**. Think of *parts* of it as declarative – a web of **pure Horn clause logic** – while other parts stay procedural only. Think of it as **a high-level Web**, capable of **serving answers to queries** – answers that follow from what the Prolog Web "knows". Moreover, imagine it being **programmable**, allowing **Web Prolog** source code to flow in either direction, from the client to the node or from the node to the client. This is what the **Prolog Web** is all about.

Two kinds of web APIs

- A WebSocket API for stateful, asynchronous and bi-directional communication
- A RESTful HTTP API for stateless, synchronous communication

Mutual dependency



Using the WebSocket API

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8"/>
    <title>WS example</title>
    <script src="/example.js">
    </script>
  </head>
  <body>
    <div id="output"></div>
  </body>
</html>
```

In the JavaScript console:

```
["X":"a"]
["X":”b"]
["X":”c"]
```

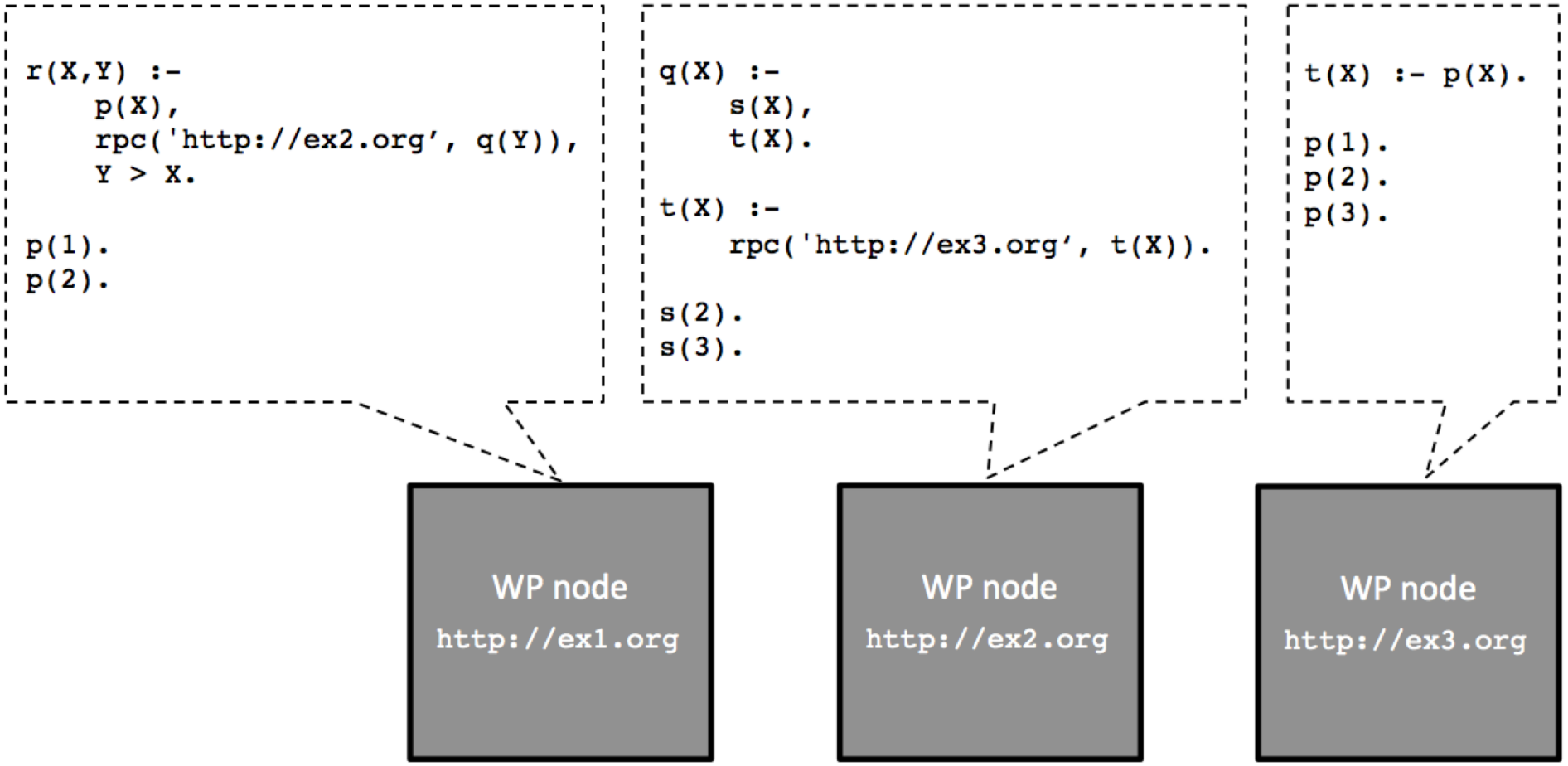
```
var ws = new WebSocket('ws://ex.org/ws','pcp-0.2');
ws.onopen = function (message) {
  ws.send(JSON.stringify({
    command: 'pengine_spawn',
    options: '[format(json)]'
  }));
};
ws.onmessage = function (message) {
  var event = JSON.parse(message.data);
  if (event.type == 'spawned') {
    ws.send(JSON.stringify({
      command: 'pengine_ask',
      pid: event.pid,
      query: 'p(X)'
    }));
  } else if (event.type == 'success') {
    console.log(event.data);
    if (event.more) {
      ws.send(JSON.stringify({
        command: 'pengine_next',
        pid: event.pid
      }));
    }
  }
};
```

Using the RESTful HTTP API

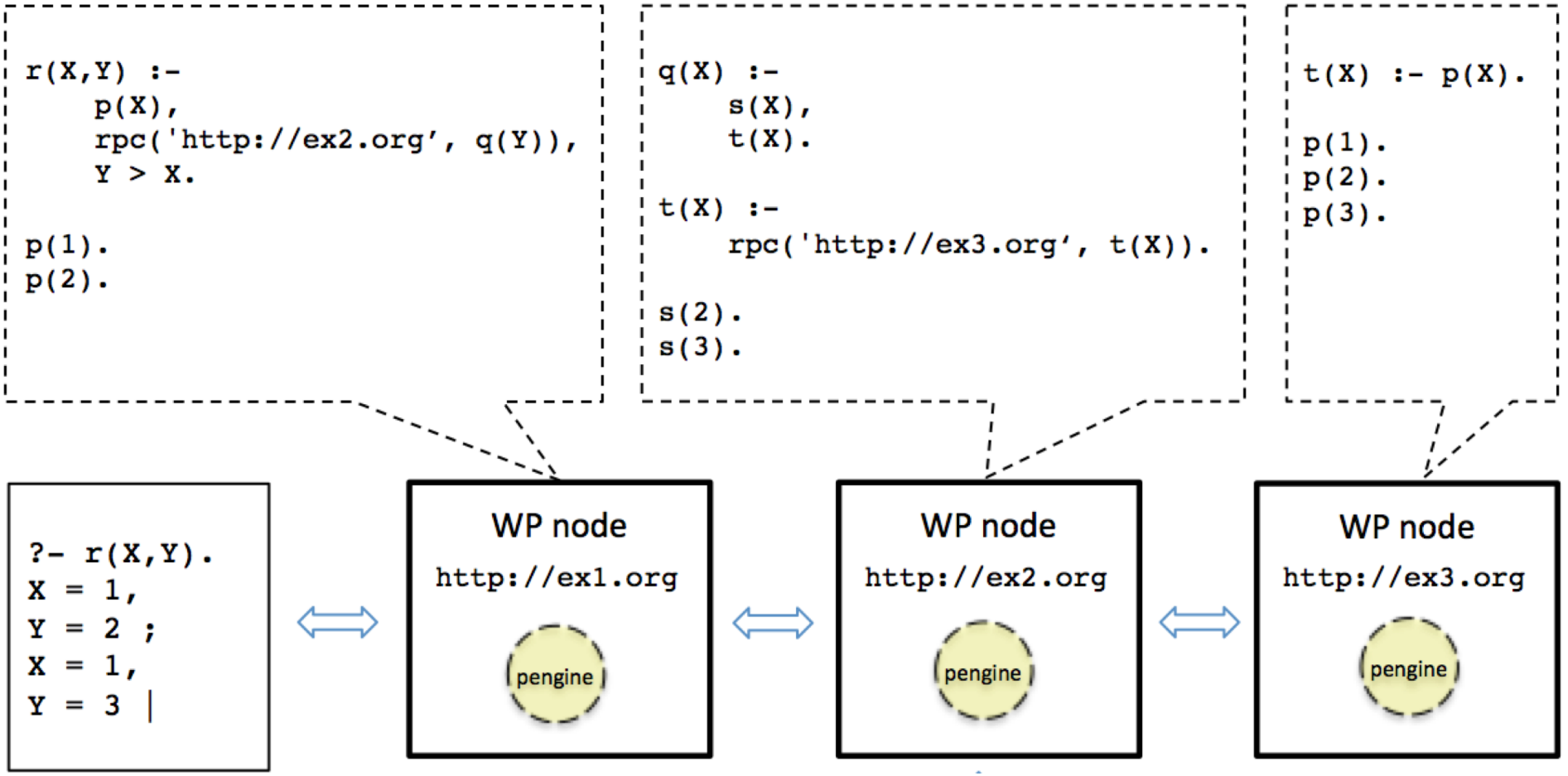
```
GET http://remote.org/ask?query=append(Xs,Ys,[a,b,c])&limit=2
{ "type":"success",
  "pid":"anonymous",
  "data":[{"Xs":[], "Ys":["a","b","c"]},{"Xs":["a"], "Ys":["b","c"]}],
  "more":true
}

GET http://remote.org/ask?query=append(Xs,Ys,[a,b,c])&offset=2&limit=2
{ "type":"success",
  "pid":"anonymous",
  "data":[{"Xs":["a","b"], "Ys":["c"]},{"Xs":["a","b","c"], "Ys":[]}],
  "more":false
}
```

- But how, in the general case, can we make this efficient?

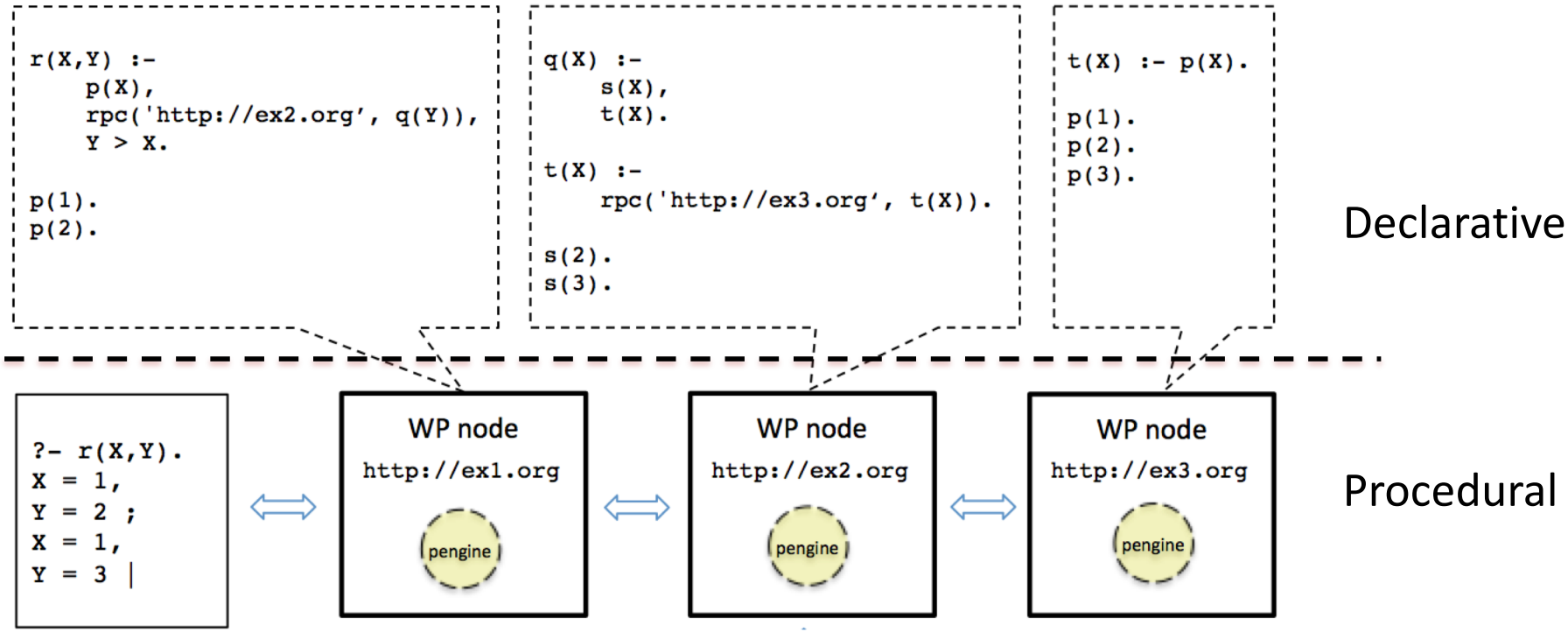
Pure Web Prolog and the pure Prolog Web



Querying the pure Prolog Web



Two levels of abstraction

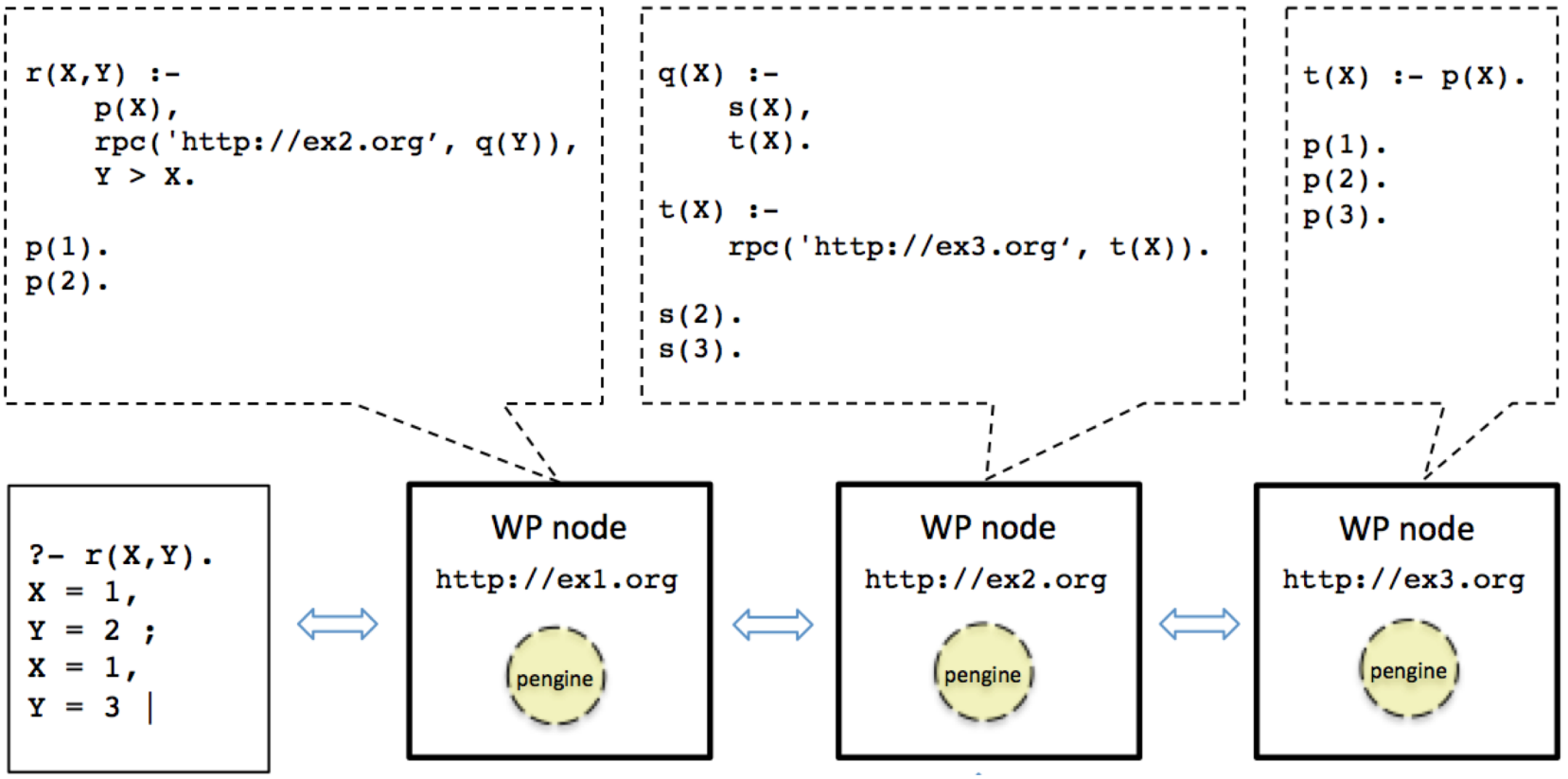



Web Prolog
and the programmable
Prolog Web

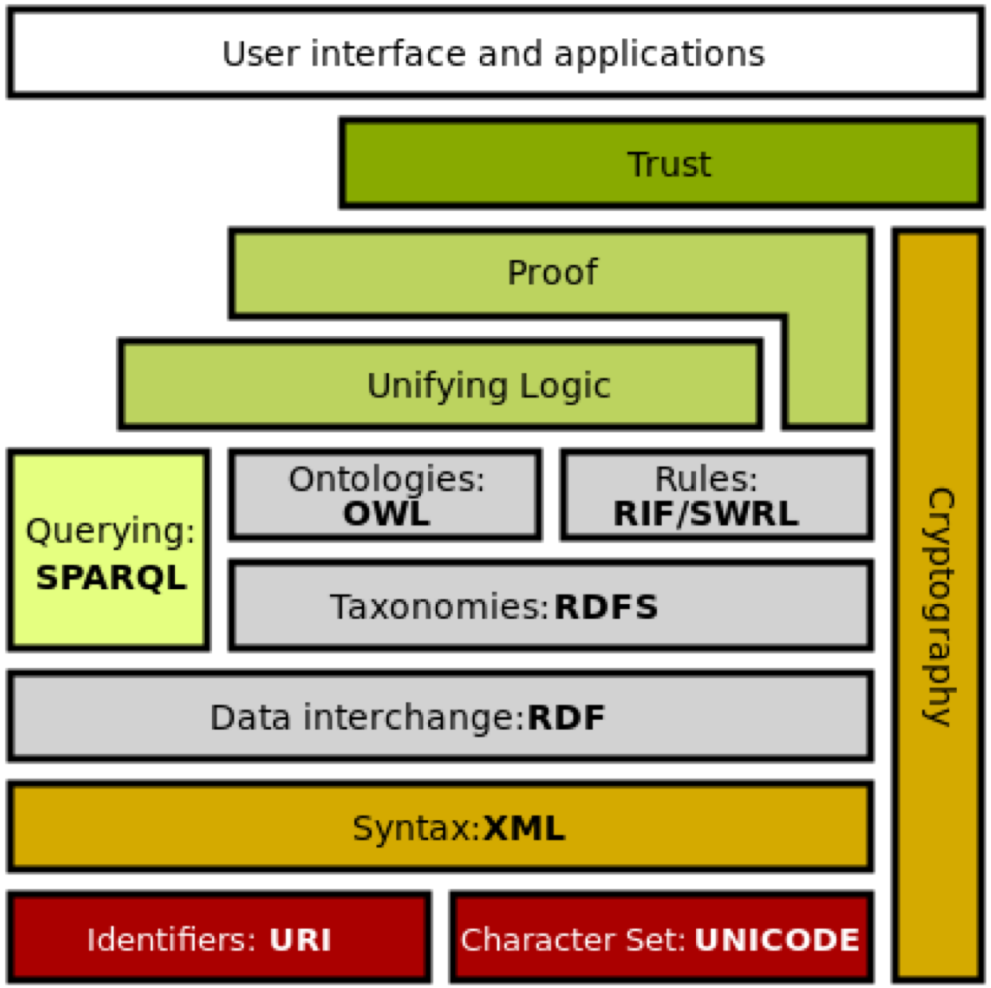
Two meanings of “programmable”

- Node-resident program
- Program injected by client

Querying the Prolog Web



Compare with the Semantic Web



The approach of Wielemaker *et al*

N-Triples

```
<http://example.org/bob#me> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/bob#me> <http://xmlns.com/foaf/0.1/knows> <http://example.org/alice#me> .
<http://example.org/bob#me> <http://schema.org/birthDate> "1990-07-04"^^<http://www.w3.org/2001/XMLSchema#date> .
```

rdf_load/2

```
rdf('http://example.org/bob#me',
    'http://...rdf-syntax-ns#type',
    'http://xmlns.com/foaf/0.1/Person').
rdf('http://example.org/bob#me',
    'http://xmlns.com/foaf/0.1/knows',
    'http://example.org/alice#me').
rdf('http://example.org/bob#me',
    'http://schema.org/birthDate',
    literal(type('http://...#date','1990-07-04'))).
```

Define

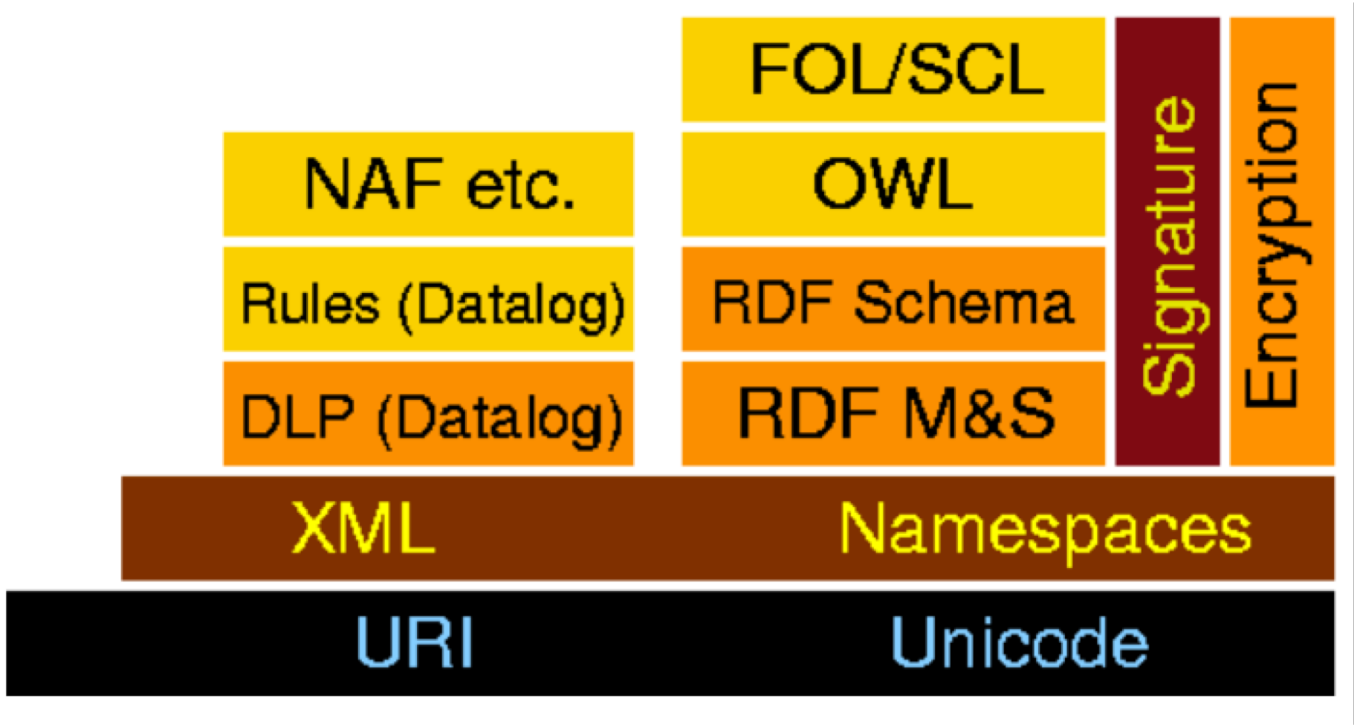
```
known_with_birthdate(Me, He, Date):-
    rdf(Me, foaf:knows, He),
    rdf(He, schema:birthDate, Date).
```

Query

```
?- known_with_birthdate(Me, He, Date).
Me = 'http://example.org/bob#me'
He = 'http://example.org/alice#me'
Date = literal(type(xsd:date,'1990-07-04') ;
Me = 'http://example.org/bob#me'
He = 'http://example.org/john#me'
Date = literal(type(xsd:date,'1992-09-06') ;
...
```

Compare with the Semantic Web

The two-tower architecture of (Horrocks et al 2005)



Logic Programming > Prolog
Web Logic Programming > Web Prolog

- Constraint Logic Programming
- Probabilistic Logic Programming
- Abductive Logic Programming
- Inductive Logic Programming
- Disjunctive Logic Programming
- Probabilistic Inductive Logic Programming
- Probabilistic Abductive Logic Programming
- Probabilistic Disjunctive Logic Programming
- Probabilistic Constraint Logic Programming
- Abductive Constraint Logic Programming
- Inductive Constraint Logic Programming
- …

Can be made available over the Prolog Web

Compare with the Semantic Web

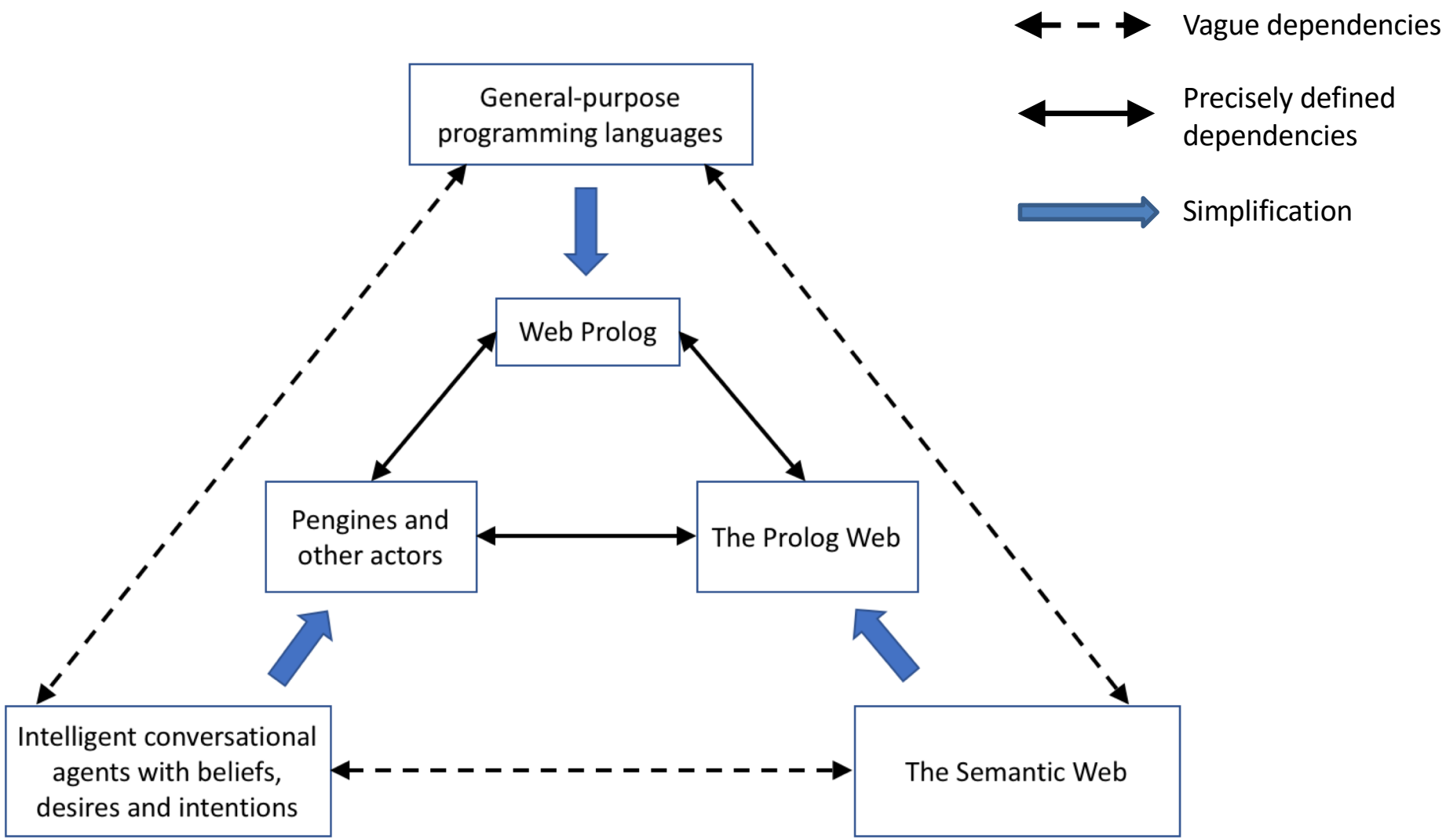
The Semantic Web

- Different languages for know-ledge representation and querying
- Semantic Web languages are not programming languages
- The Semantic Web doesn't come with an architecture

The Prolog Web

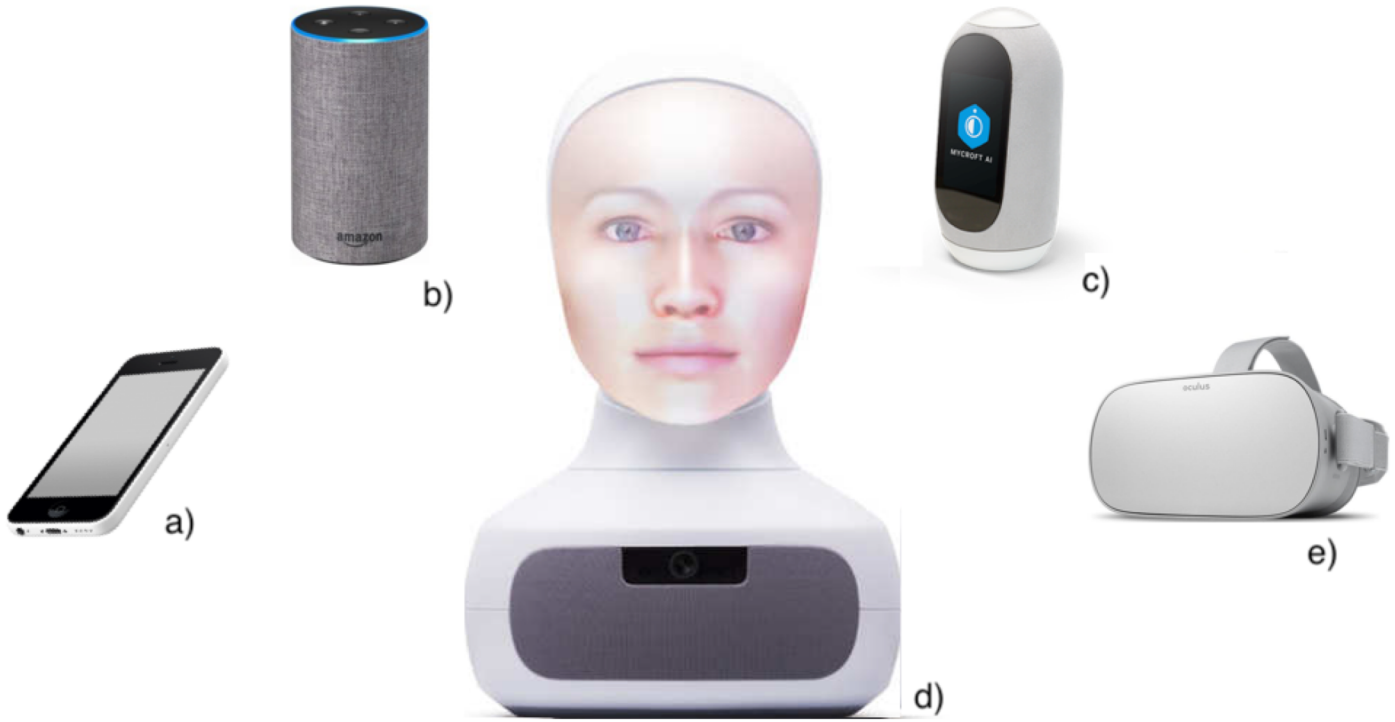
- Same language for know-ledge representation and querying
- Web Prolog is a programming language
- The Prolog Web comes with an architecture

The big picture

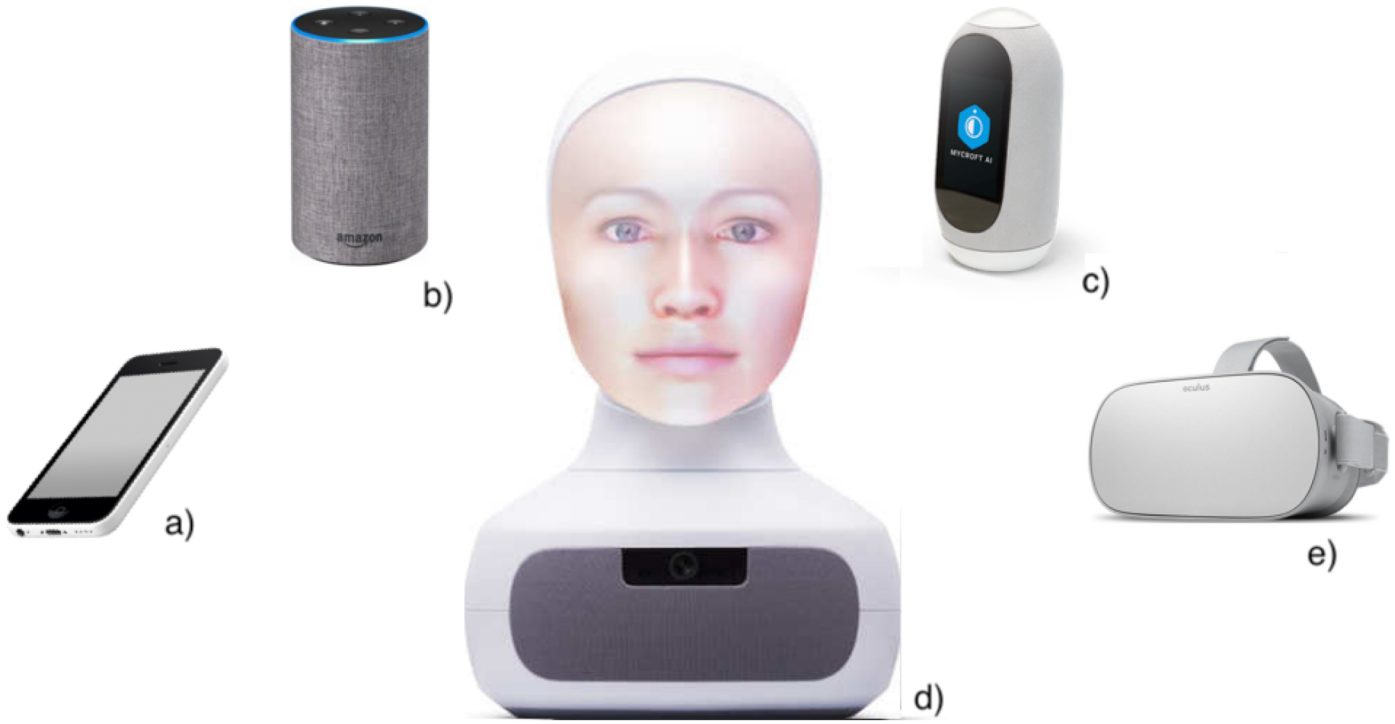


Talking to the Prolog Web

Conversational systems – how to program things like these?



Conversational systems – how to program things like these?



They seem to need both **knowledge representation** and **reasoning** as well as **means for real-time interaction** …

Statechart actors

- Statecharts (Harel, 1987) is a graphical notation for specifying reactive systems in great detail. In Harel's own words, a “fully executable visual formalism intended for capturing the behaviour of complex real-world systems”

- Popular among user interface designers – for graphical UIs as well as voice UIs.

- A good way to manage state, and great for orchestration of processes.

- **Statechart actors** are programmed in **SCXML** and uses Web Prolog as a data model and scripting language

SCXML

W3C Recommendation

State Chart XML (SCXML): State Machine Notation for Control Abstraction

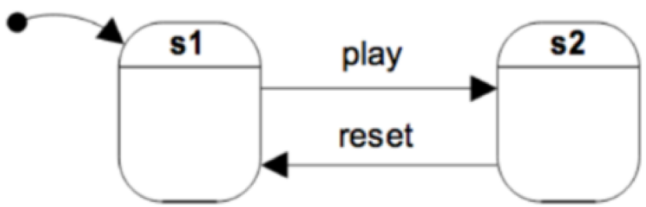
W3C Recommendation 1 September 2015

This version:
http://www.w3.org/TR/2015/REC-scxml-20150901/
Latest version:
http://www.w3.org/TR/scxml/
Previous version:
http://www.w3.org/TR/2015/PR-scxml-20150430/
Editors:
Jim Barnett, Genesys (Editor-in-Chief)
Rahul Akolkar, IBM
RJ Auburn, Voxeo
Michael Bodell, (until 2012, when at Microsoft)
Daniel C. Burnett, Voxeo
Jerry Carter, (until 2008, when at Nuance)
Scott McGlashan, (until 2011, when at HP)
Torbjörn Lager, Invited Expert
Mark Helbing, (until 2006, when at Nuance)
Rafah Hosn, (until 2008, when at IBM)
T.V. Raman, (until 2005, when at IBM)
Klaus Reifenrath, (until 2006, when at Nuance)
No'am Rosenthal, (until 2009, when at Nokia)
Johan Roxendal, Invited Expert

Statechart actors

- Statechart (and SCXML) features event-driven finite-state automata with :
  - Hierarchy and history states
  - Parallelism (a.k.a. orthogonality)
  - Broadcast communication
  - Process invocation
  - Inter-process communication
  - Data model and scripting language

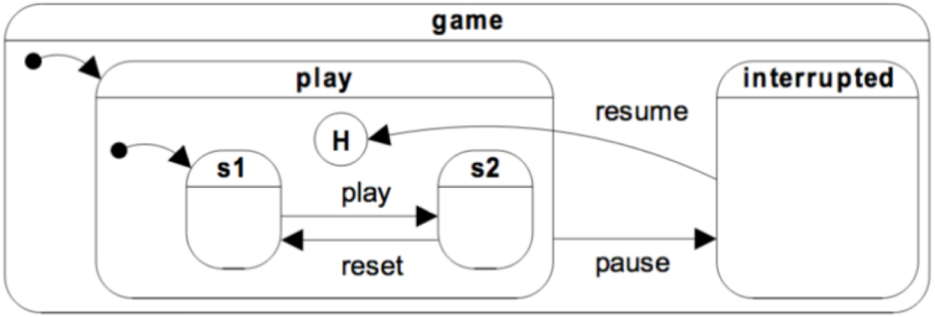
Running a statechart actor



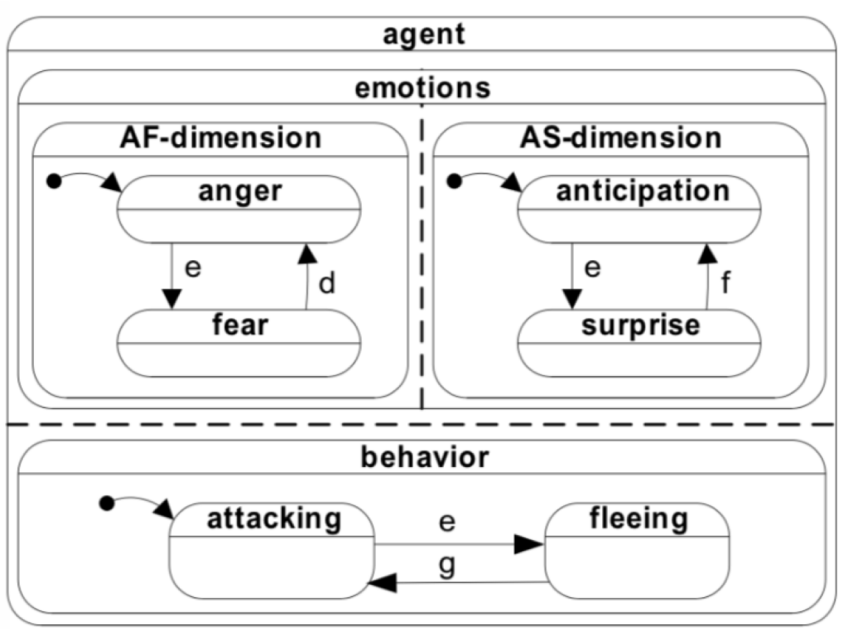

```
<scxml datamodel="web-prolog" initial="s1">
    <state id="s1">
        <onentry>return('IDLE')</onentry>
        <transition event="play" target="s2"/>
    </state>
    <state id="s2">
       <onentry>return('PLAYING')</onentry>
       <transition event="reset" target="s1"/>
    </state>
</scxml>
```

```
?- spawn(run([]), Pid, [
        type(scxml),
        src_uri('http://src.org/game.scxml'),
        monitor(true)
   ]).
Pid = '8dca3c82'.

?- flush.
Shell got 'IDLE'
true.

?- $Pid ! play.
true.

?- receive({What -> true}).
What = 'PLAYING.

?- $Pid ! reset.
true.

?- receive({What -> true}).
What = 'IDLE'.
?-
```

Hierarchy and parallelism




```
<scxml datamodel="web-prolog" initial="play">
    <state id="play" initial="s1">
        <history id="h">
            <transition target="s1"/>
        </history>
        <state id="s1">
            <transition event="play" target="s2"/>
        </state>
        <state id="s2">
            <transition event="reset" target="s1"/>
        </state>
        <transition event="pause" target="interrupted"/>
    </state>
    <state id="interrupted">
        <transition event="resume" target="h"/>
    </state>
</scxml>
```




```
<scxml datamodel="web-prolog" initial="agent">
    <parallel id="agent">
        <parallel id="emotions">
            <state id="AF-dimension">
                <state id="anger">
                    <transition event="e" target="f
                </state>
                <state id="fear">
                    <transition event="d" target="a
                </state>
            </state>
            <state id="AS-dimension">
```

Invoking a pengine and running a query

```
<scxml datamodel="web-prolog" initial="spawn-ask-collect">
    <state id="spawn-ask-collect" initial="ask">
        <spawn type="pengine"
               node="http://remote.org"
               exit="false">
            p(a). p(b). p(c).
        </spawn>
        <state id="ask">
            <transition event="spawned(Pid)" target="collect">
                pengine_ask(Pid, p(X))
            </transition>
        </state>
        <state id="collect">
            <transition event="success(Pid,Data,true)" target="collect">
                return(Data),
                pengine_next(Pid)
            </transition>
            <transition event="success{_,Data,false)" target="f"/>
                return(Data)
            </transition>
        </state>
    </state>
    <final id="f"/>
</scxml>
```

Euclid's algorithm for calculating the greatest common divisor of a set of numbers

```
<scxml datamodel="web-prolog" initial="run">
    <datamodel>
        number(25).
        number(10).
        number(15).
        number(30).
    </datamodel>
    <state id="run">
        <transition cond="number(X), number(Y), X > Y">
            Z is X-Y,
            retract(number(X)),
            assert(number(Z))
        </transition>
        <transition cond="number(X)" target="stop">
            return(X)
        </transition>
    </state>
    <final id="stop"/>
</scxml>
```


Web Prolog
and the programmable
Prolog Web

Prolog is no longer popular

- Used to be fairly popular 1980 – 2000.
- Applications in artificial intelligence, knowledge representation, expert systems, computational linguistics, etc.
- Some really good text books were written – many of them are now given away for free.

The crisis of Prolog

- *A Wake Up Call for the Logic Programming Community*

  ”In the six years that I have been doing research in the Logic Programming community, I have met a lot of nice people and heard about a lot of interesting research ideas. However, the community itself isn't exactly thriving and LP has a serious PR problem. It's my fear that if we happily continue along our current path, then there will be little left of LP and its flagship Prolog in a couple of years.”

  Tom Schrijvers in the December 2007 issue of the ALP Newsletter

- Things have improved since then, but not by very much…

More about the crisis of Prolog

- At ICLP 2008, the self assessment continues:

  [One] big problem is that the Prolog community has been too fragmented. This is true at several levels: at the system side one notes that there are too many incompatible Prolog systems around, their structure and libraries are different, their implementation technology is very diverse, and their aims are difficult to reconcile (research or industrial deployment).

A plethora of Prolog systems

- B-Prolog
- CxProlog
- ECLiPSe Prolog
- GNU Prolog
- JIProlog
- Lean Prolog
- Qu-Prolog
- Quintus Prolog
- SICStus Prolog
- SWI Prolog
- XSB Prolog
- YAP Prolog
- Ciao Prolog
- Jekejeke Prolog
- BinProlog
- LPA Prolog
- Strawberry Prolog
- …

Different Prolog systems are good at different things…

Failed standard

- The core ISO standard for Prolog, while a significant achievement, is limited in its scope and does not address a number of features that real applications need.

- The ISO Prolog standard has thus failed both in the sense that few existing Prolog systems of today are conformant, and that most systems go far beyond the standard in some respects.

- In 2009, Paulo Moura stepped down as chair for the ISO Prolog standardization effort, writing on his blog:

> *I still believe that standardization is vital for the future of Prolog as a programming language. But the current ISO process is the wrong way to do it.*

An Attempt to Make Prolog Great Again

- Lack of popularity, a failed standardisation effort, a fragmented community, …

What to do?

Rebranding Prolog

- *Rebranding* is a marketing strategy in which a new name, term, symbol, design, or combination thereof is created for an established brand with the intention of developing a new, differentiated identity in the minds of consumers, investors, competitors, and other stakeholders.

  Wikipedia

Compare LISP -> Clojure rebranding

LISP

rebranded into

Clojure

runs on top of

The Java VM

Prolog

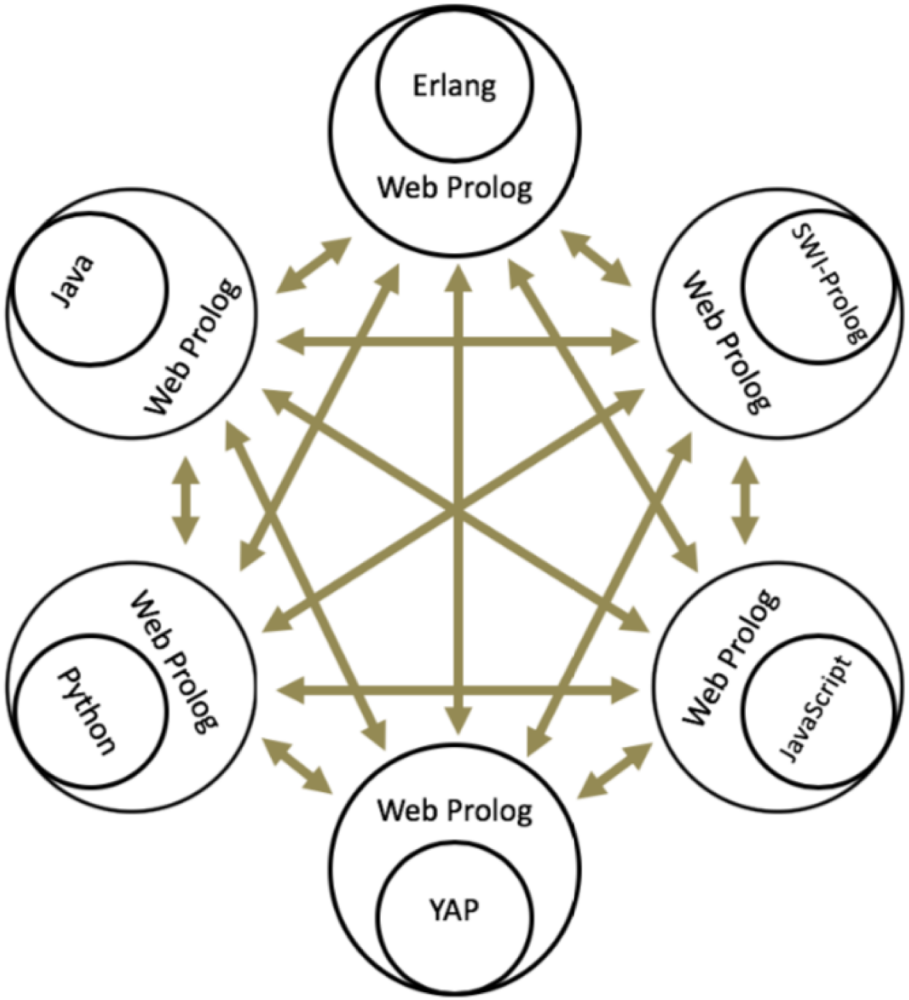
rebranded into

Web Prolog

runs on top of

The Web

Web Prolog as an interlingua



Standardisation of Web Prolog

- Prolog and Erlang are both very mature languages. With Web Prolog, not much is novel, except in the change of focus and in the ways things are combined. This can be seen as an advantage when standardising.

- Let’s standardise under the auspices of the W3C instead of (and/or as a complement to) ISO!

- Imagine:

W3C Community Grops

Two relevant W3C Working Groups

The Semantic Web Programming Languages Community Group

A community focused on the adoption of Semantic Web concepts within contemporary and new programming languages. These will incorporate W3C Semantic Web standards for Ontology, Linked data and representations as integral parts of the development tool chains. Particularly the group will aim to 1. Develop new semantically-aware programming languages, **2. Modify existing languages to be semantically-aware** 3. Develop design patterns for semantically-aware programming. 4. Develop Ontologies for computer programming concepts to allow inter-lingual sharing of basic and domain-specific algorithms.

The Voice Interaction Community Group

Existing W3C voice interaction standards such as VoiceXML are based on use cases centered around telephony-based voice systems. The typical interaction style that these standards support is system-initiated directed dialog using grammars to constrain the speech recognizer. In recent years, interaction with voice applications has become much more flexible, with a user-initiated dialog style and significantly fewer constraints on spoken input. Many of these new applications take the form of "virtual assistants". These include general-purpose assistants (for example, Siri, Cortana, Google Now and Alexa) as well as virtual assistants with specialized domain expertise. The proposed Community Group will collect new use cases for voice interaction, develop requirements for applications such as virtual assistants and explore areas for possible standardization, possibly producing specifications if appropriate. Depending on interest, this exploration could include such topics as (1) discovery of virtual assistants with specific expertise, for example a way to find a virtual assistant that can supply weather information (2) standard formats for statistical language models for speech recognizers (3) standard representations for references to common concepts such as time (4) interoperability for conversational interfaces and **(5) work on dialogue management or ‘workflow' languages.** New functionality for existing voice standards can also be a topic of discussion. Speech application developers and voice user interface designers should be particularly interested in this group.

Reviving Prolog

- by rebranding it as a **web logic programming language**
- a language serving as an **interlingua** allowing different Prolog platforms running on the Web to **communicate**, **interoperate** and **cooperate** in the service of human users of the **Prolog Web**
- a language which gives the Prolog community **a Prolog dialect in common,**
- a language that might **appeal to the logic programming community at large**,
- a dialect based on **a subset of the ISO Prolog core extended with primitives inspired by Erlang**
- with a **new standard** under the auspices of the **W3C,**
- the use of Web Prolog as a **scripting language in SCXML,**
- and targeting a **new application area – spoken (and possibly multi-modal) conversational systems.**

A proposal for the Logic Programming and Prolog community

2022

Celebrate Prolog's 50th anniversary in 2022 –
Marry Prolog with Erlang and spawn Web Prolog!

Support **Web Prolog** 2022 – an interlingua
to celebrate Prolog's 50th anniversary!

Thank you!

(And don’t forget, I’ll be happy to demonstrate the proof-of-concept implementation while I’m here!)