Ajax at Work: A Case Study

Peter-Paul Koch (ppk)
http://www.quirksmode.org

Why Ajax?

Enhancing customer expectations

Why Ajax?

why

Because it's popular!

Why Ajax?

why

Because it's the future!

Why Ajax?

Because clients want to score!

Why ~~Ajax~~ JavaScript?

The purpose of JavaScript is adding usability to a web page.

Ideally the page should remain accessible, though.

Why ~~Ajax~~ JavaScript?

Use JavaScript only if it gives a significant usability benefit over a non-scripted page.

So what do we use Ajax for?

Why ...?

The Ajax Gods

Click on the Ajax God of your choice to view more information!

The Ajax Gods

Lug - ancient fertility God

Rate Lug

Made the switch to Ajax relatively recently.

(Note the traditional gesture of propitiation.)

So what do we use Ajax for?

Why

Emulating frames

```
<frameset rows="150,*">
   <frame src="header.html" />
   <frame src="thumbs.html" />
</frameset>
```

So what do we use Ajax for?

Why

Emulating frames

```
<frameset rows="50,*">
  <frame src="header.html" />
  <frame src="thumbs.html" />
</frameset>
```

Is this what Ajax is all about?

The Ajax Gods

Lug - ancient fertility God

Rate Lug

Made the switch to Ajax relatively recently.

(Note the traditional gesture of propitiation.)

The Ajax Gods

Lug - ancient fertilit

Rate Lug

Made the switch to Ajax relati

(Note the traditional gesture of propitiation.)

Rate Lug

Rate Lug

Rate Lug

(response
from server)

Rate Lug

Insufficient rating.
Try propitiating
Lug again.

(response
from server)

So what do we use Ajax for?

Why

Sending queries and receiving responses. Pretty standard nowadays.

Still, we could do this with frames, too, if we really wanted.

(Personal opinion warning)

Why

The less our application emulates frames, the more Ajaxy it becomes.

(Personal opinion warning)

Why

Whenever you conceive an Ajax-application, ask yourself if frames could do the job, too.

(Personal opinion warning)

Why

If the answer is “Yes, easily”

ask yourself whether you really need Ajax .

Today's case study

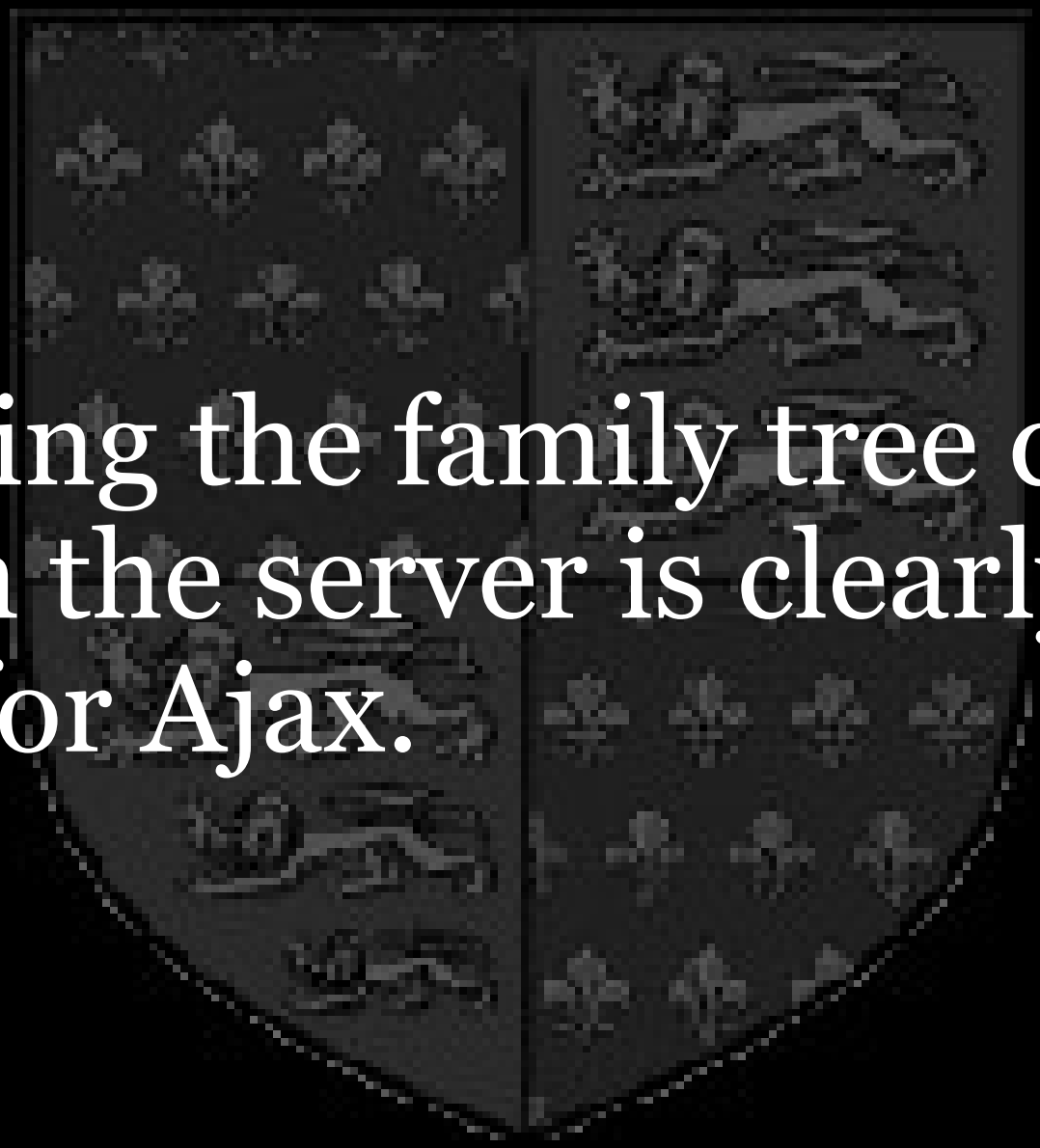
An interactive family tree of the Plantagenets & Tudors (1216-1603)

Ajax is used for fetching new data
(The real trouble lies in the display of the data. That's not today's subject, though.)

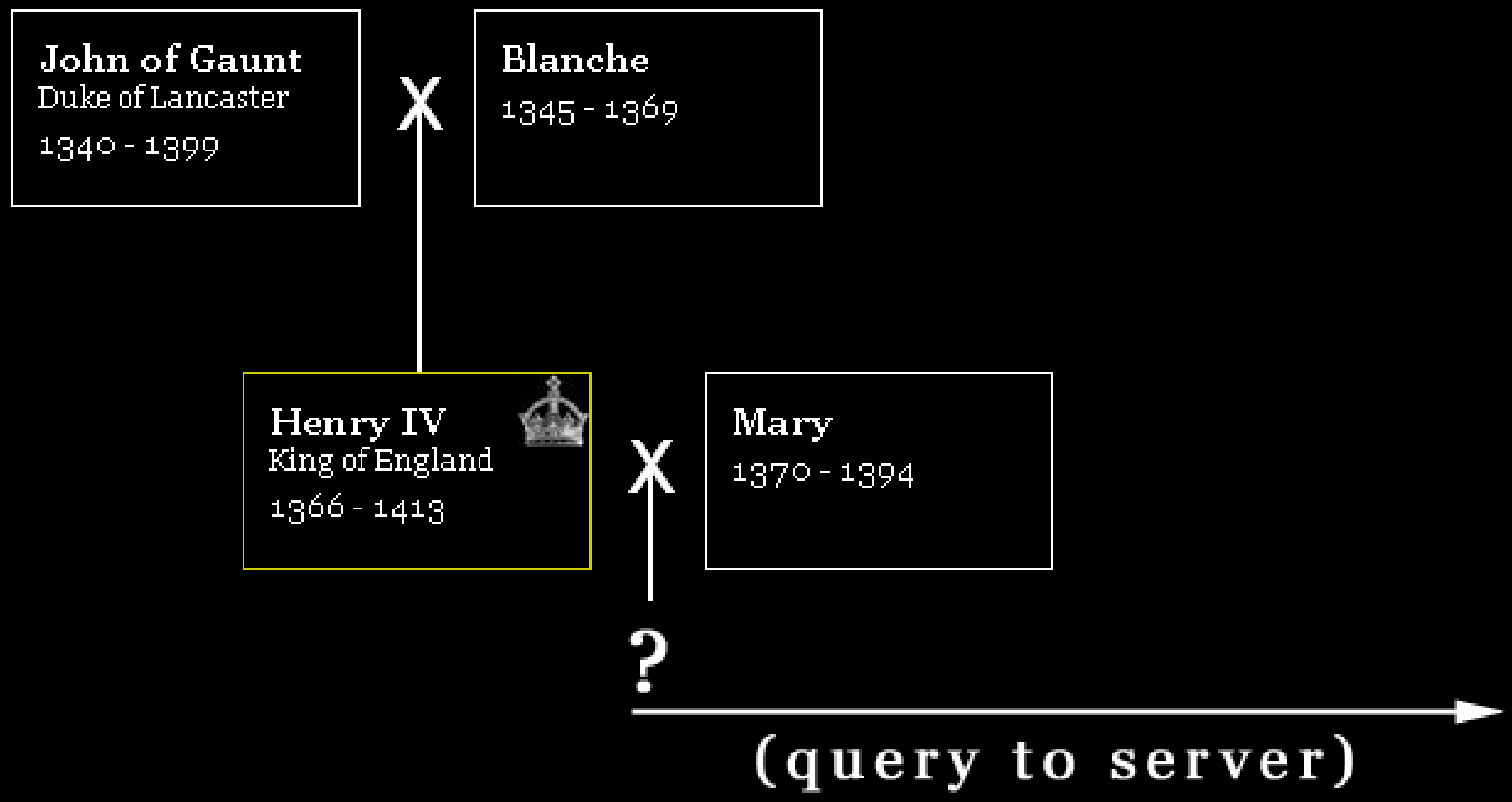
An interactive family tree of the Plantagenets & Tudors (1216-1603)

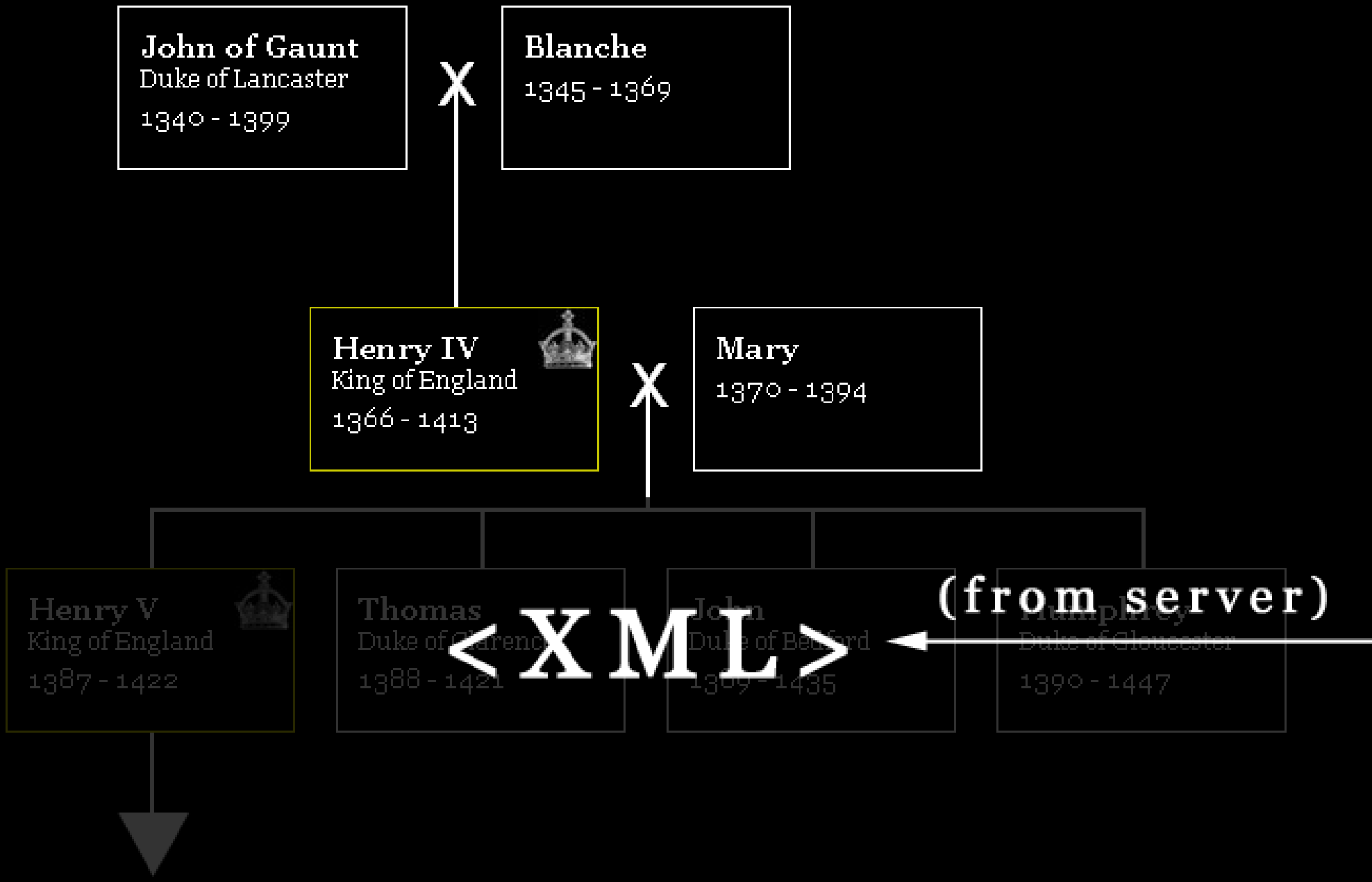
See it live at http://quirksmode.org/familytree/

Firefox and Safari only – for now



Getting the family tree data from the server is clearly a job for Ajax.


John of Gaunt
Duke of Lancaster
1340 - 1399
X
Blanche
1345 - 1369
Henry IV
King of England
1366 - 1413
X
Mary
1370 - 1394
?
(query to server)


John of Gaunt
Duke of Lancaster
1340 - 1399
X
Blanche
1345 - 1369
Henry IV
King of England
1366 - 1413
X
Mary
1370 - 1394
Henry V
King of England
1387 - 1422
Thomas
Duke of Clarence
1388 - 1421
John
Duke of Bedford
1389 - 1435
Humphrey
Duke of Gloucester
1390 - 1447
<XML>
(from server)

John of Gaunt
Duke of Lancaster
1340 - 1399

X

Blanche
1345 - 1369

Henry IV
King of England
1366 - 1413

X

Mary
1370 - 1394

Henry V
King of England
1387 - 1422

Thomas
Duke of Clarence
1388 - 1421

John
Duke of Bedford
1389 - 1435

Humphrey
Duke of Gloucester
1390 - 1447

We can't really do this with frames.

Ajax is the only solution.

So what do we use Ajax for?

Data mining

Getting data from server is a job for Ajax

Displaying the results needs some DOM scripting (which isn't Ajax, strictly speaking), as well as a lot of CSS

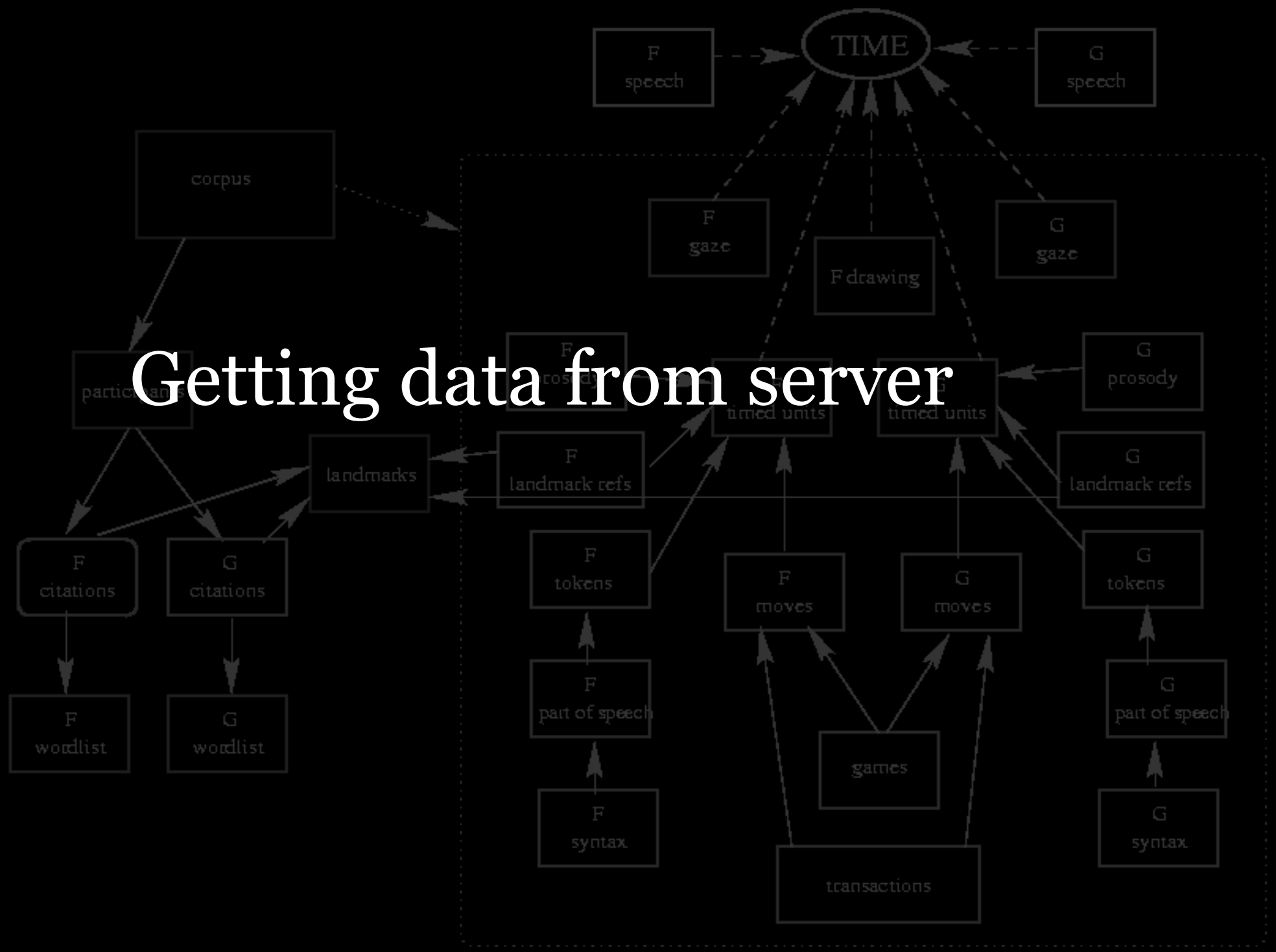
So what do we use Ajax for?

Data mining

Getting data from server is a job for Ajax

~~Displaying the results needs some DOM scripting (which isn't Ajax, strictly speaking), as well as a lot of CSS~~

Getting data from server



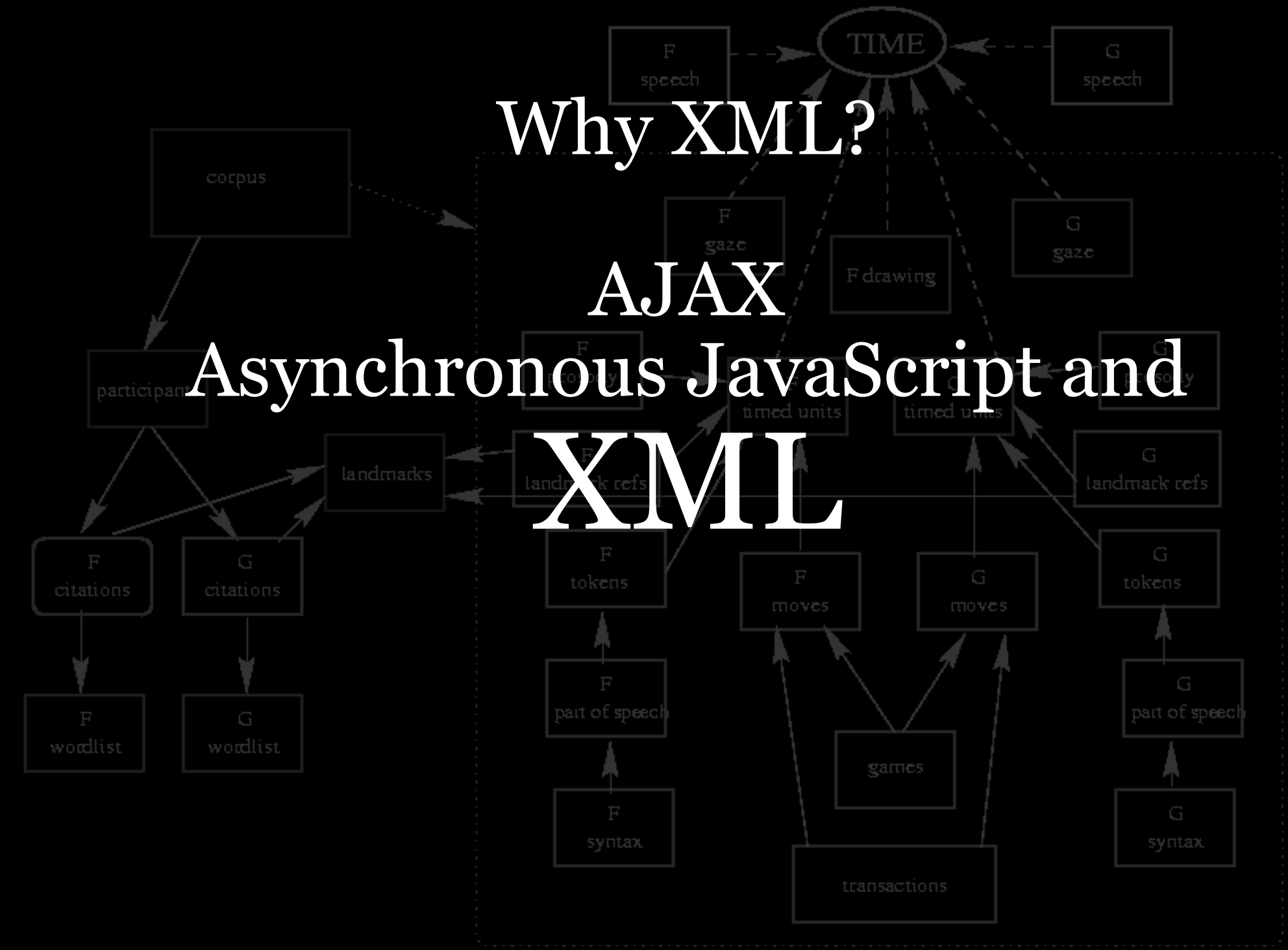
```
<person id="15">
    <name>
        <short>Richard II</short>
    </name>
    <birth>1365</birth><death>1400</death>
    <father idref="3">Edward</father>
    <mother idref="8">Joan</mother>
    <ranks>
        <rank>
            <predecessor idref="1">Edward III</predecessor>
            <title>King of England</title>
            <start>1377</start>
            <end>1399</end>
            <successor idref="17">Henry IV</successor>
        </rank>
    </ranks>
</person>
```

Why XML?

Facilitating existing technologies

Why XML?

AJAX
Asynchronous JavaScript and
XML



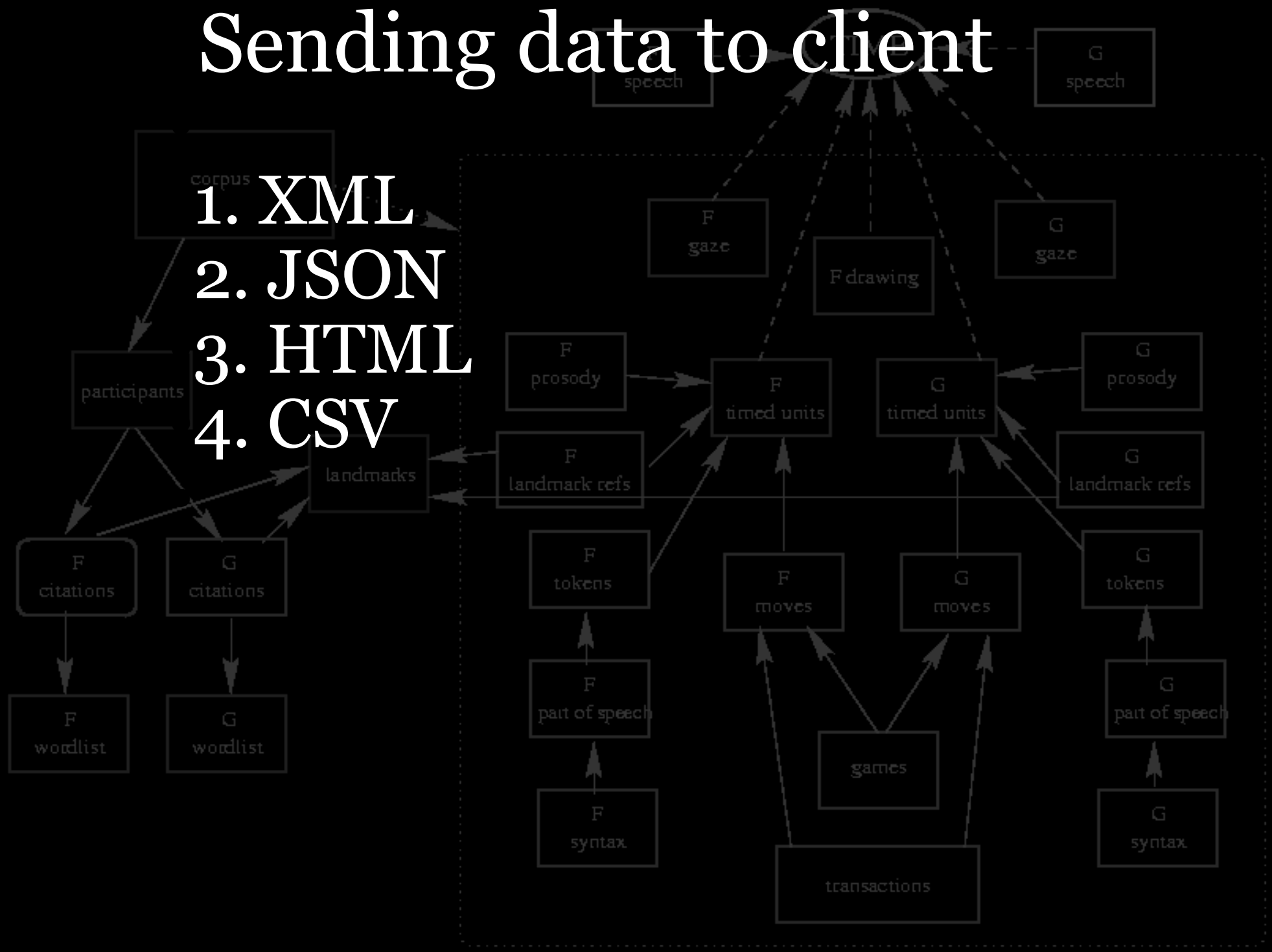
Why XML?

Ajax doesn't need XML.

XML is just one possibility.

Sending data to client

1. XML
2. JSON
3. HTML
4. CSV



XML

```
<person id="15">
    <name>
        <short>Richard II</short>
    </name>
    <birth>1365</birth><death>1400</death>
    <father idref="3">Edward</father>
    <mother idref="8">Joan</mother>
    <ranks>
        <rank>
            <predecessor idref="1">Edward III</predecessor>
            <title>King of England</title>
            <start>1377</start>
            <end>1399</end>
            <successor idref="17">Henry IV</successor>
        </rank>
    </ranks>
</person>
```

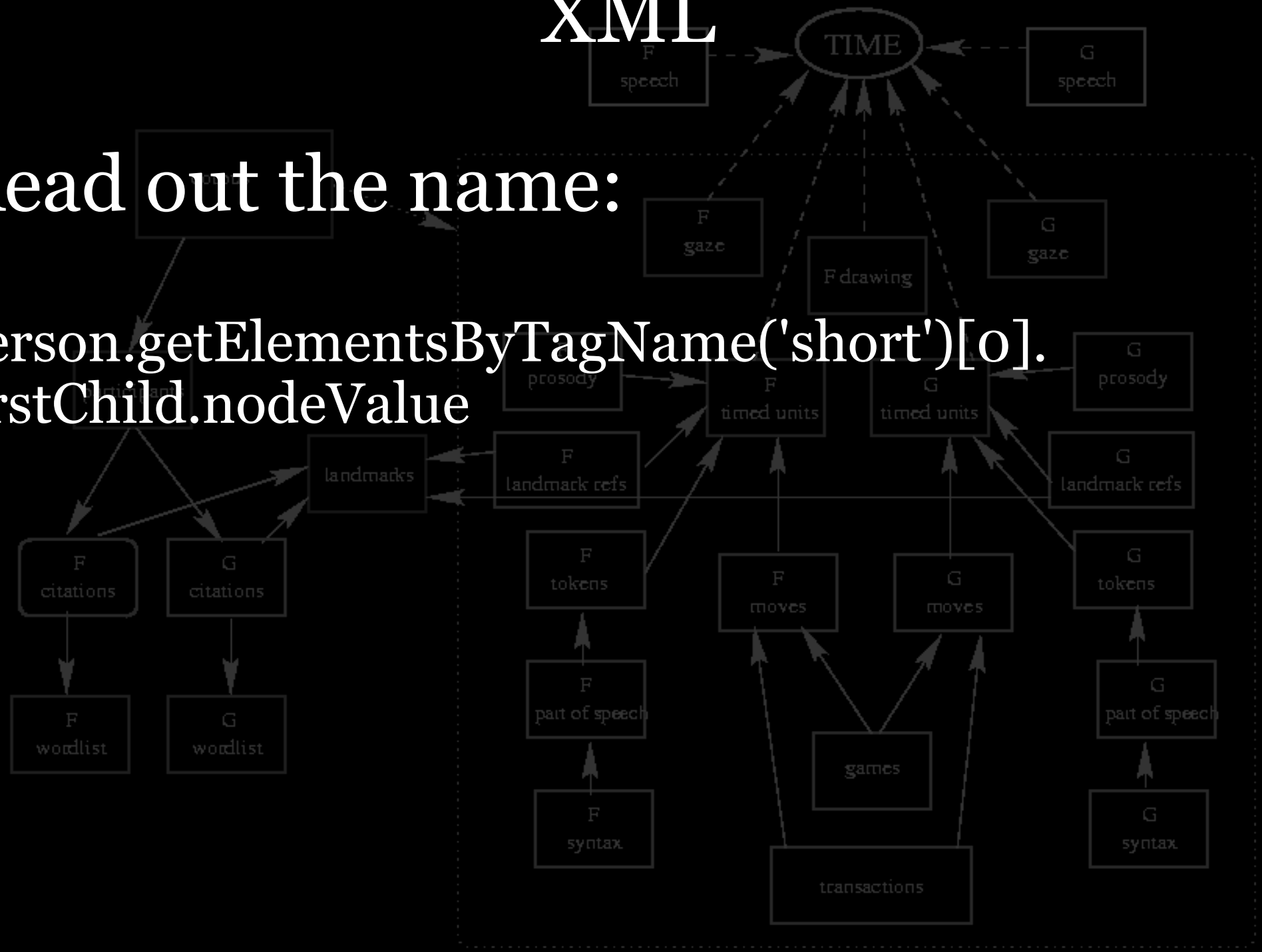
XML

```
<person id="15">
    <name>
        <short>Richard II</short>
    </name>
    <birth>1365</birth><death>1400</death>
    <father idref="3">Edward</father>
    <mother idref="8">Joan</mother>
    <ranks>
        <rank>
            <predecessor idref="1">Edward III</predecessor>
            <title>King of England</title>
            <start>1377</start>
            <end>1399</end>
            <successor idref="17">Henry IV</successor>
        </rank>
    </ranks>
</person>
```

XML

Read out the name:

person.getElementsByTagName('short')[0].
firstChild.nodeValue



XML

Advantages

* Humans can read XML
* Many languages have XML support and use the W3C DOM

Disadvantages

* Verbose and clunky

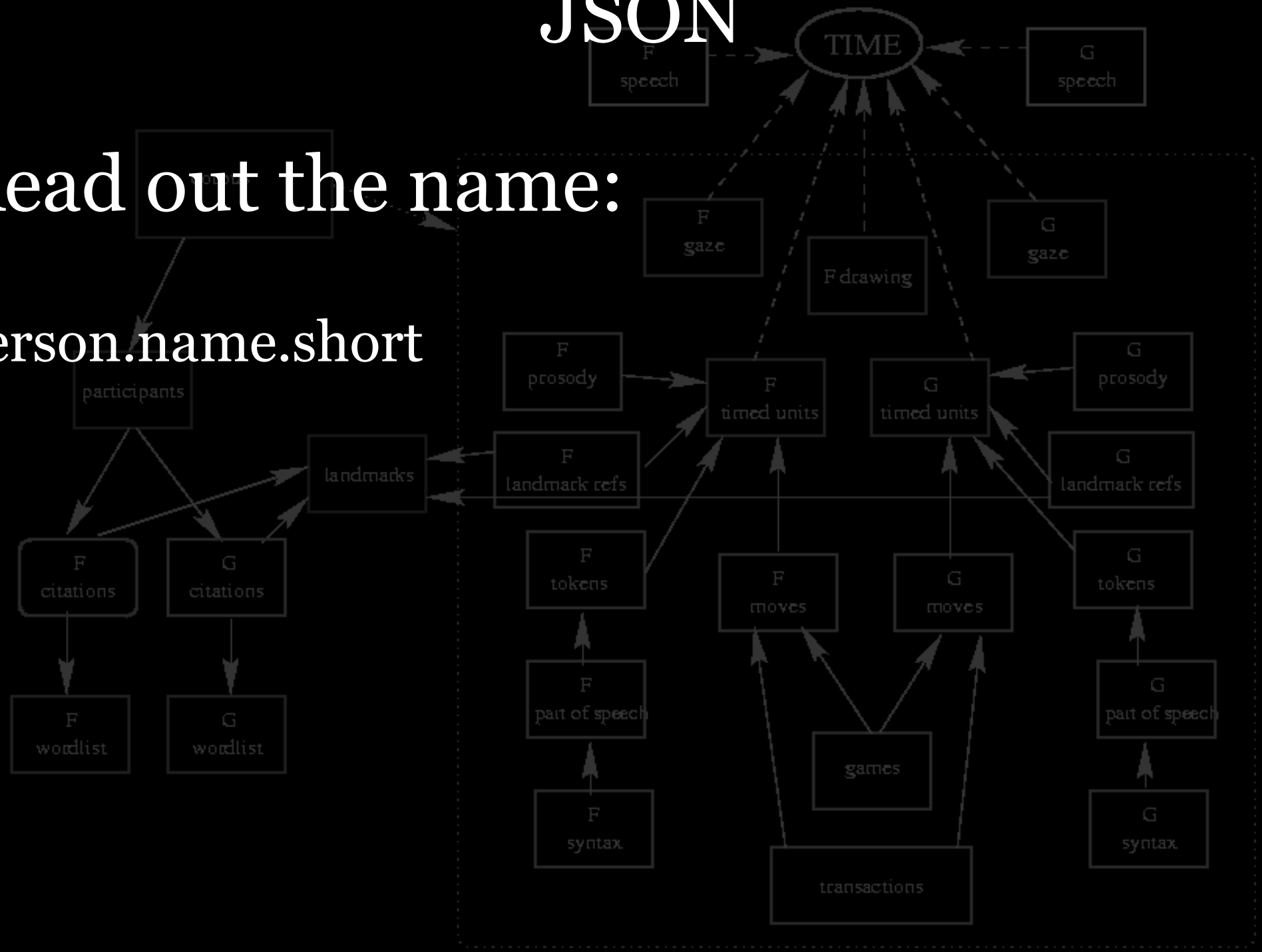
JSON


```
{
    "name": {"short": "Richard II"},
    "birth": "1365",
    "death": "1400",
    "father": {"idref": "3","name": "Edward"},
    "mother": {"idref": "8","name": "Joan"},
    "ranks": {
        {
            "title": "King of England",
            "start": "1377",
            "end": "1399",
            "predecessor": {"name": "Edward III","idref": "1"},
            "successor": {"name": "Henry IV","idref": "17"}
        },
    }
}
```

JSON

Read out the name:

person.name.short



JSON


```
{
    "name": {"short": "Richard II"},
    "birth": "1365",
    "death": "1400",
    "father": {"idref": "3","name": "Edward"},
    "mother": {"idref": "8","name": "Joan"},
    "ranks": {
        {
            "title": "King of England",
            "start": "1377",
            "end": "1399",
            "predecessor": {"name": "Edward III","idref": "1"},
            "successor": {"name": "Henry IV","idref": "17"}
        },
    }
}
```

JSON

```
{
    "name": {"short": "Richard II"},
    "birth": "1365",
    "death": "1400",
    "father": {"idref": "3","name": "Edward"},
    "mother": {"idref": "8","name": "Joan"},
    "ranks": {
        {
            "title": "King of England",
            "start": "1377",
            "end": "1399",
            "predecessor":{"name": "Edward III","idref": "1"},
            "successor": {"name": "Henry IV","idref": "17"}
        },
    }
}
```

Can you spot the error?

JSON


```
{
    "name": {"short": "Richard II"},
    "birth": "1365",
    "death": "1400",
    "father": {"idref": "3","name": "Edward"},
    "mother": {"idref": "8","name": "Joan"},
    "ranks": {
        {
            "title": "King of England",
            "start": "1377",
            "end": "1399",
            "predecessor": {"name": "Edward III","idref": "1"},
            "successor": {"name": "Henry IV","idref": "17"}
        },
    }
}
```

JSON


```
{
    "name": {"short": "Richard II"},
    "birth": "1365",
    "death": "1400",
    "father": {"idref": "3","name": "Edward"},
    "mother": {"idref": "8","name": "Joan"},
    "ranks": {
        {
            "title": "King of England",
            "start": "1377",
            "end": "1399",
            "predecessor":{"name": "Edward III","idref": "1"},
            "successor": {"name": "Henry IV","idref": "17"}
        },
    }
}
```

JSON

Advantages

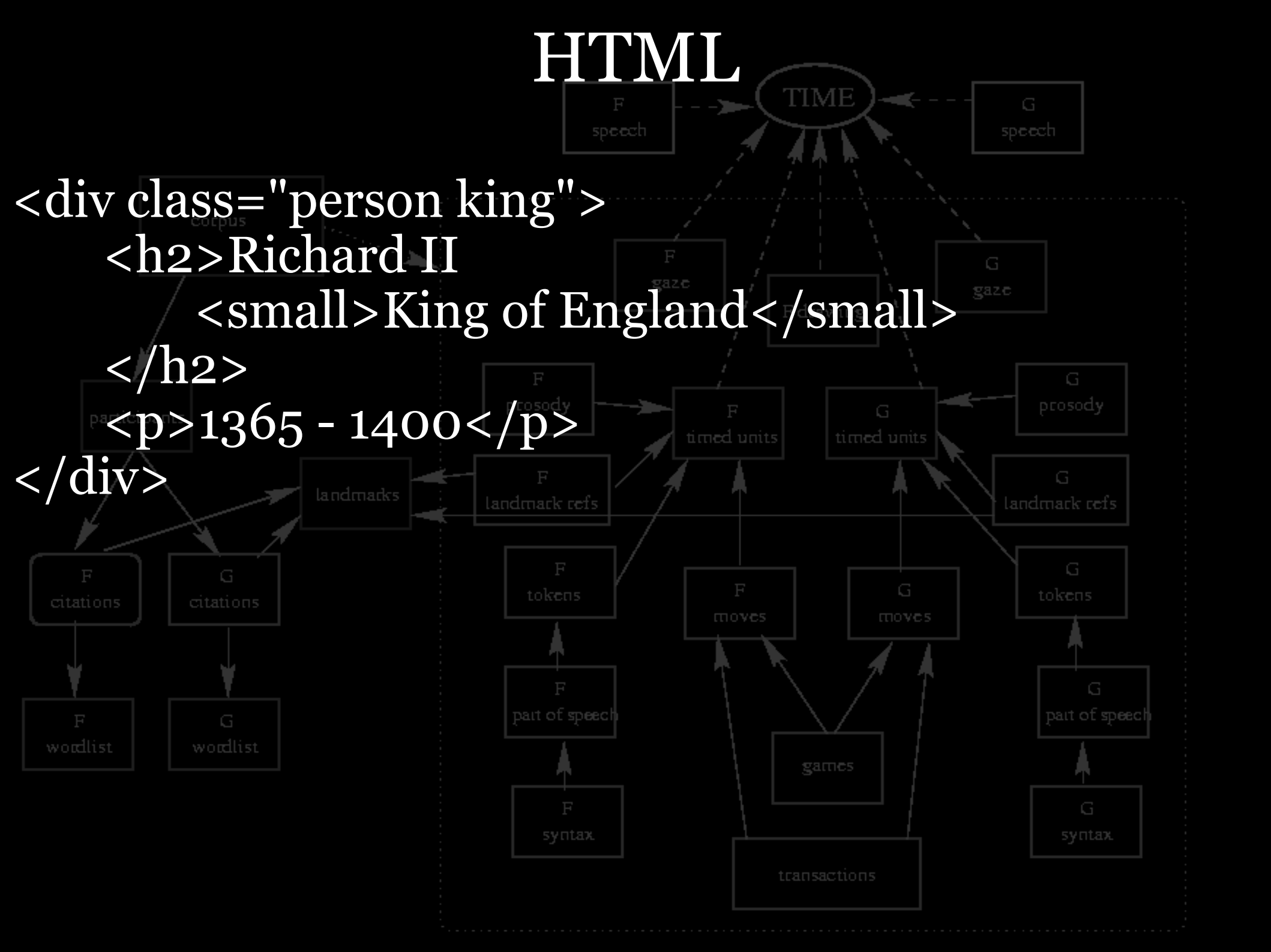
* Same functionality as XML, but lighter

Disadvantages

* Relatively unknown outside geek circles (as yet)
* Less human-readable; hard to check by eye alone

HTML

```
<div class="person king">
    <h2>Richard II
        <small>King of England</small>
    </h2>
    <p>1365 - 1400</p>
</div>
```

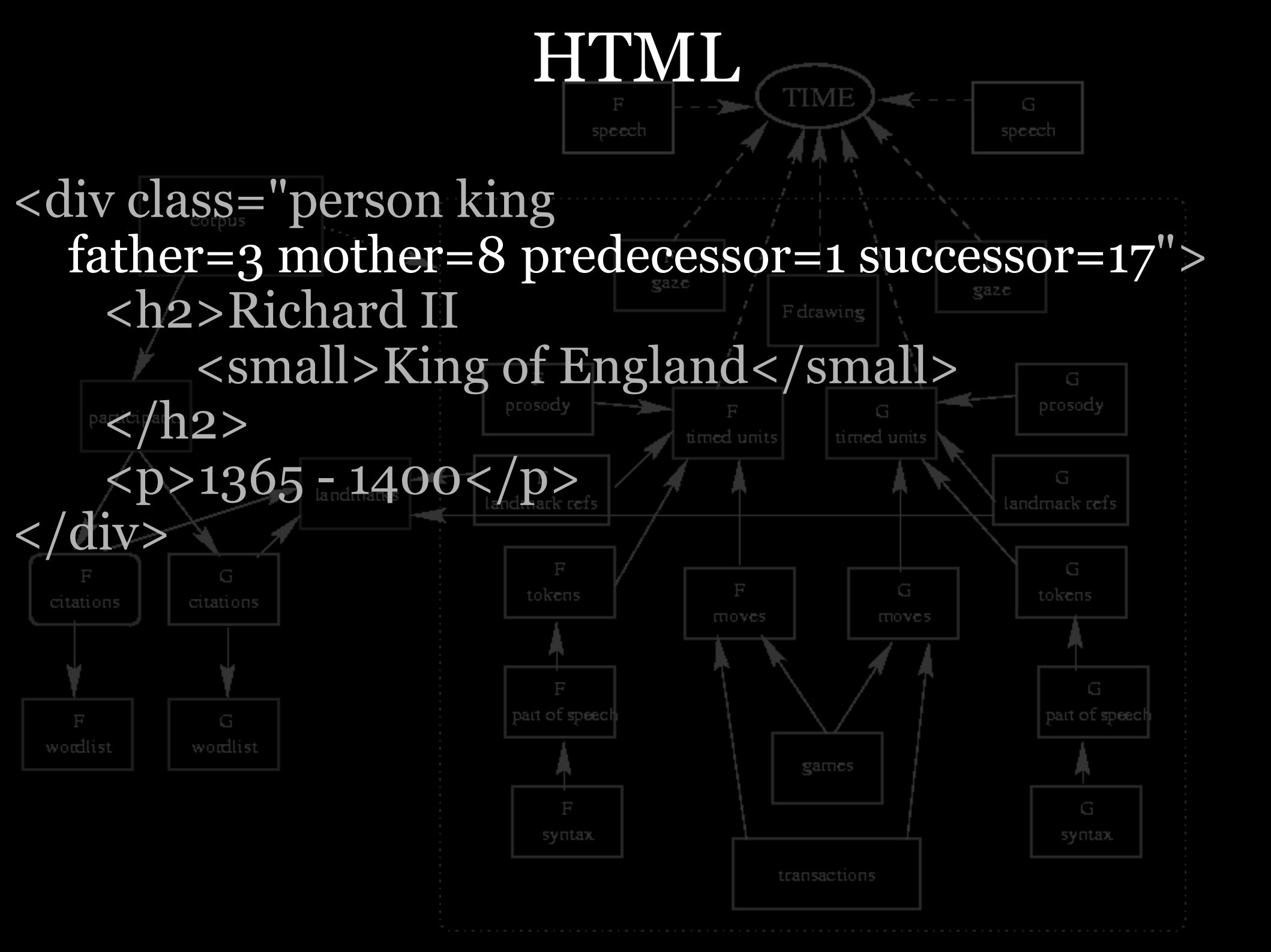


HTML

```
<div class="person king"
    father="3" mother="8"
    predecessor="1" successor="17">
    <h2>Richard II
        <small>King of England</small>
    </h2>
    <p>1365 - 1400</p>
</div>
```

HTML

```
<div class="person king
   father=3 mother=8 predecessor=1 successor=17">
     <h2>Richard II
          <small>King of England</small>
     </h2>
     <p>1365 - 1400</p>
</div>
```



HTML

Advantages

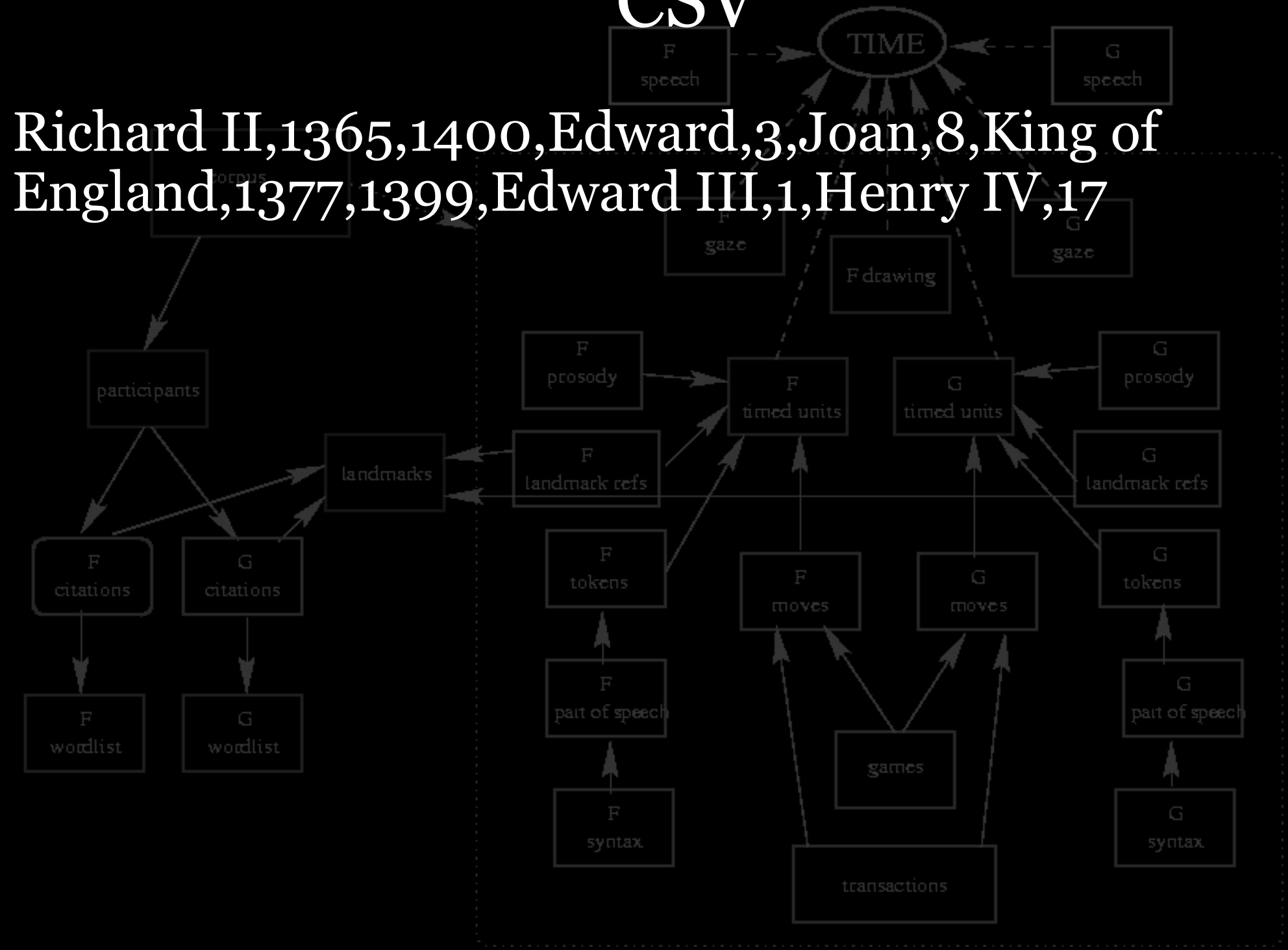
* Really simple
* Best for accessibility

Disadvantages

* Relational data difficult to incorporate

CSV

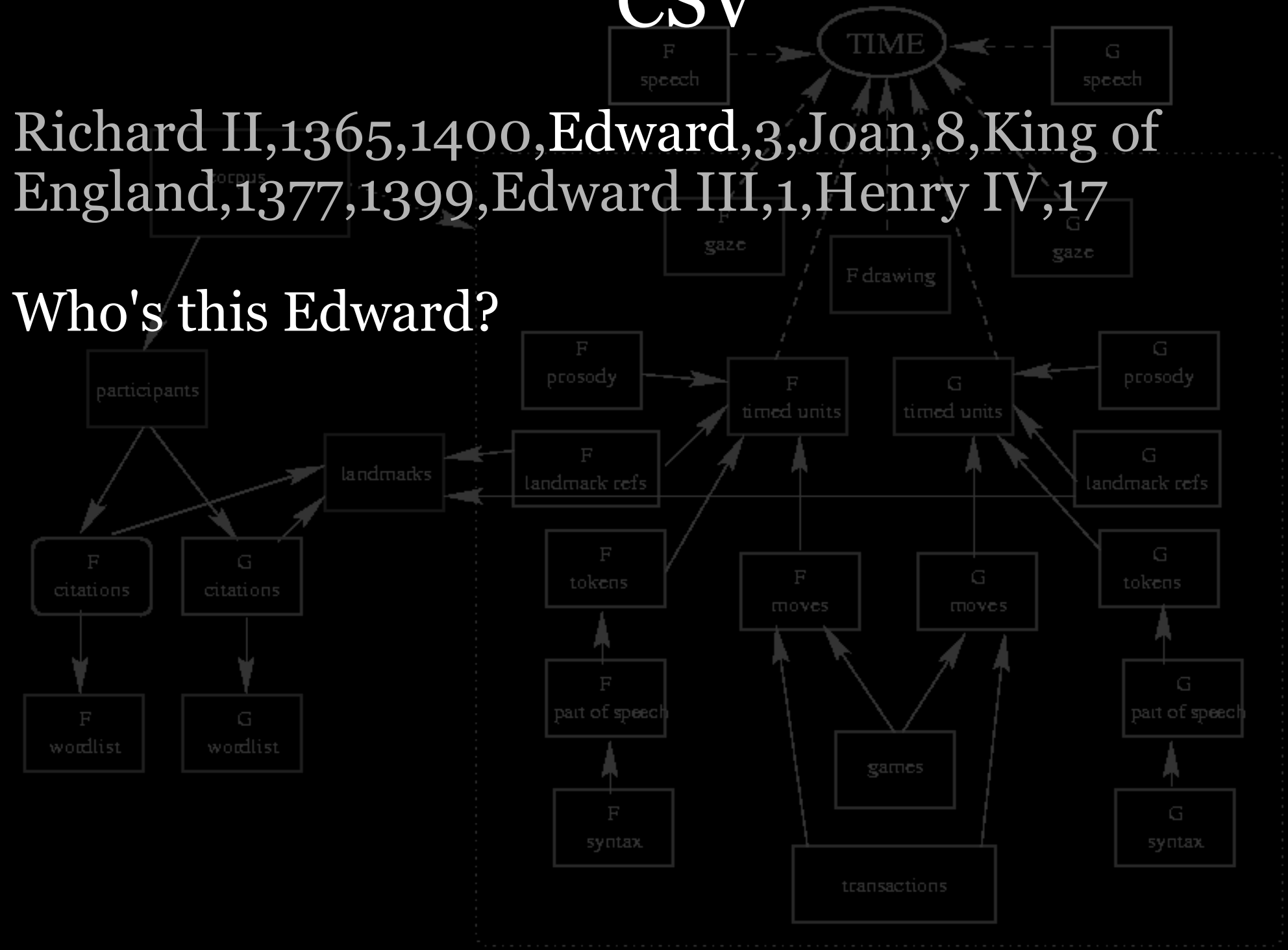
Richard II,1365,1400,Edward,3,Joan,8,King of England,1377,1399,Edward III,1,Henry IV,17



CSV

Richard II,1365,1400,Edward,3,Joan,8,King of England,1377,1399,Edward III,1,Henry IV,17

Who's this Edward?



CSV

Advantages

* Easy to parse for programming languages

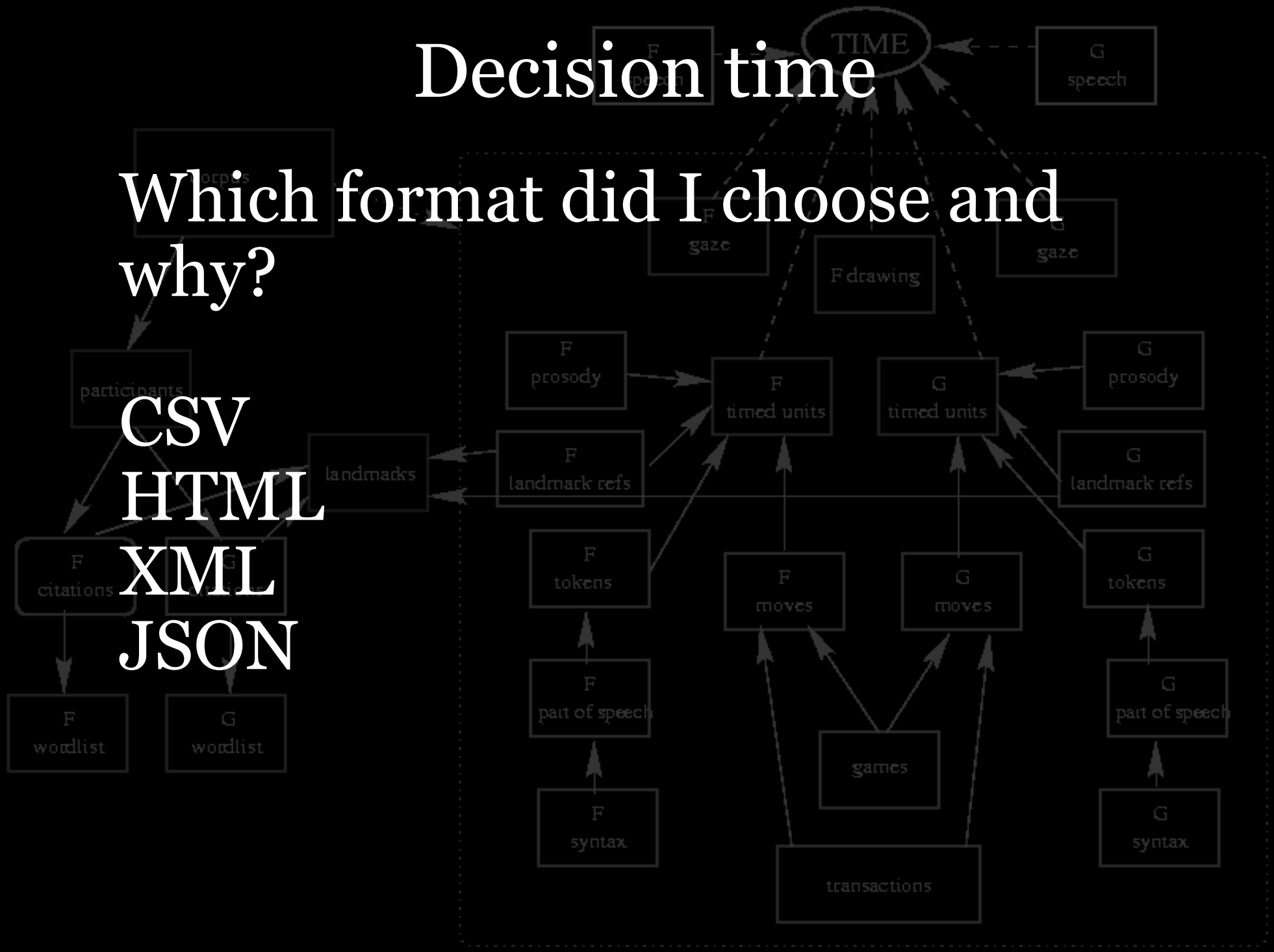
Disadvantages

* No meta-data
* Not human-readable

Decision time

Which format did I choose and why?

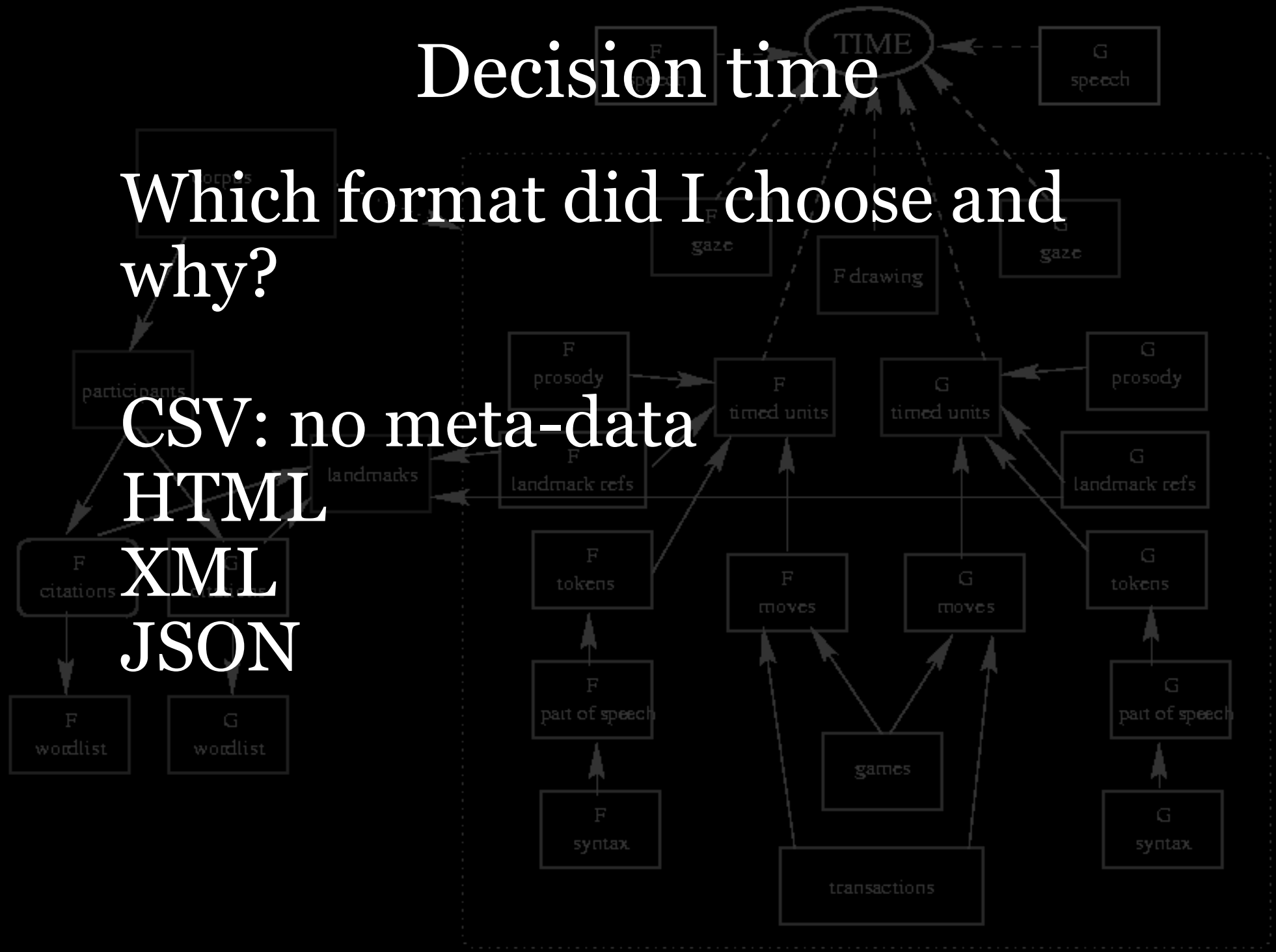
CSV
HTML
XML
JSON



Decision time

Which format did I choose and why?

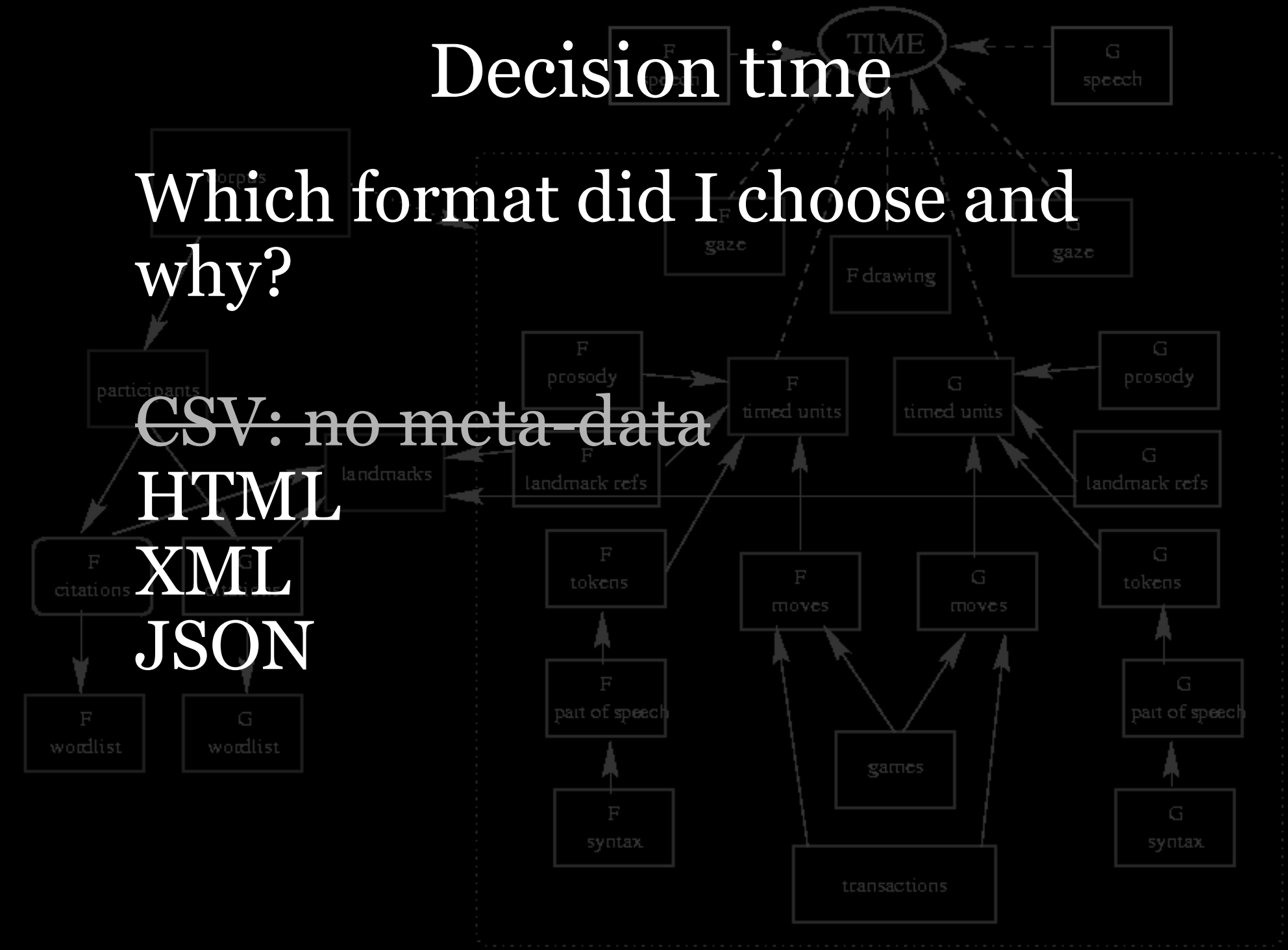
CSV: no meta-data
HTML
XML
JSON



Decision time

Which format did I choose and why?

~~CSV: no meta-data~~
HTML
XML
JSON



Decision time

Which format did I choose and why?

~~CSV: no meta-data~~

HTML:

* unsuited for relational data
* there are no pages, so no accessibility benefit

XML

JSON

Decision time

Which format did I choose and why?

~~CSV: no meta-data~~
~~HTML:~~
~~* unsuited for relational data~~
~~* there are no pages, so no accessibility benefit~~
XML
JSON

Decision time

Which format did I choose and why?

~~CSV: no meta-data~~
~~HTML:~~
~~* unsuited for relational data~~
~~* there are no pages, so no accessibility benefit~~
XML
JSON: social factors

Decision time

Which format did I choose and why?

~~CSV: no meta-data~~
~~HTML:~~
~~* unsuited for relational data~~
~~* there are no pages, so no accessibility benefit~~
XML
~~JSON: social factors~~

Social factors

JSON is relatively unknown (as yet).

Therefore, company X, which is going to produce the database, may not know how to work with JSON.

Social factors - XML

Me:
“Hello, I'd like you to send me my data in XML."

Them:
“Yes, of course! We just bought this £60,000 component that does exactly that.”

Social factors - XML

Me:
"Hello, I'd like you to send me my data in XML."

Them:
"Yes, of course! We just bought this £60,000 component that does exactly that."

Social factors - JSON

Me:
“Hello, I'd like you to send me my data in JSON"

Them [confused]:
“In what?”

Communication enters danger zone.

Social factors - JSON

Me:
“JSON, you know, the light-weight data interchange format invented by Douglas Crockford."

Them:
“Erm ... well ... we're focusing on enterprise-wide leveraging software right now, so I'm not sure this is going to work.”

Social factors - JSON

Me:
"JSON, the light-weight data interchange format invented by Douglas Crockford."

Them:
"Erm ... well ... we're focusing on enterprise-wide leveraging software right now, so I'm not sure this relationship is going to work."

Total communication breakdown

Social factors

You cannot assume that every server-side party knows JSON.

You **can** assume that every server-side party knows XML.

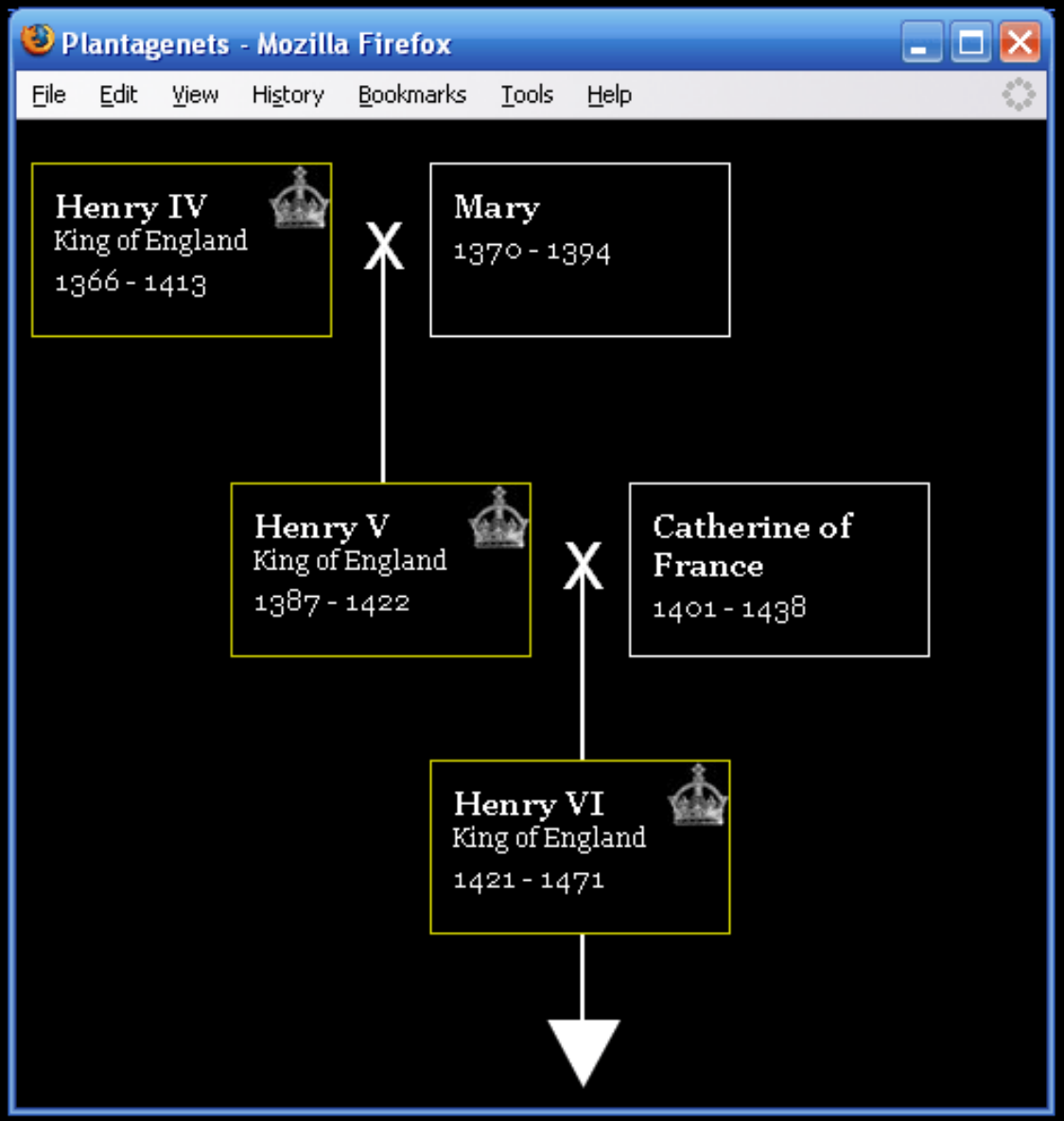
For the moment XML is the safer choice when working with third parties.

Loading

Integrating timeless experiences

Situation

Right now I load all XML at once.
120 Plantagenets, 46K.



XML


```
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>
<father idref="3">Edward
</father><mother idref="8">
Joan</mother><ranks><rank>
<predecessor idref="1">
Edward III</predecessor>
<title>King of England</title>
<start>1377</start><end>1399
</end><successor idref="17">
Henry IV</successor></rank>
</ranks></person>
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>
<father idref="3">Edward
</father><mother idref="8">
Joan</mother><ranks><rank>
<predecessor idref="1">
Edward III</predecessor>
<title>King of England</title>
<start>1377</start><end>1399
```

Situation

Right now I load all XML at once.
120 Plantagenets, 46K.

Eventually the applications could contain all royal houses of Europe; thousands of persons.

We need a more sophisticated load strategy.

Load strategy

1) Store all data you receive, so that you never have to request it again. (Rather obvious.)

2) Define the problem: loading cascade.

Loading cascade – the situation

1) User clicks on Richard of York. The new view needs Richard's children and grandchildren.

2) Richard's XML contains his children. Load these from server

```
request('62','63','64','65','66');
```

Loading cascade – the situation

3) Once we have Richard's children, we need their children.

4) Parse newly received XML and extract their child IDs.

5) Load grandchildren.
   request('lots','of','ids');

Loading cascade – the situation

6) But what about more complicated situations? Suppose the view needs the parents-in-law of Richard's children?

7) Load children, then spouses of children, then parents of spouses.

Loading cascade – the situation

8) General problem: you don't know which XML to load before other XML has been parsed.

Load strategy

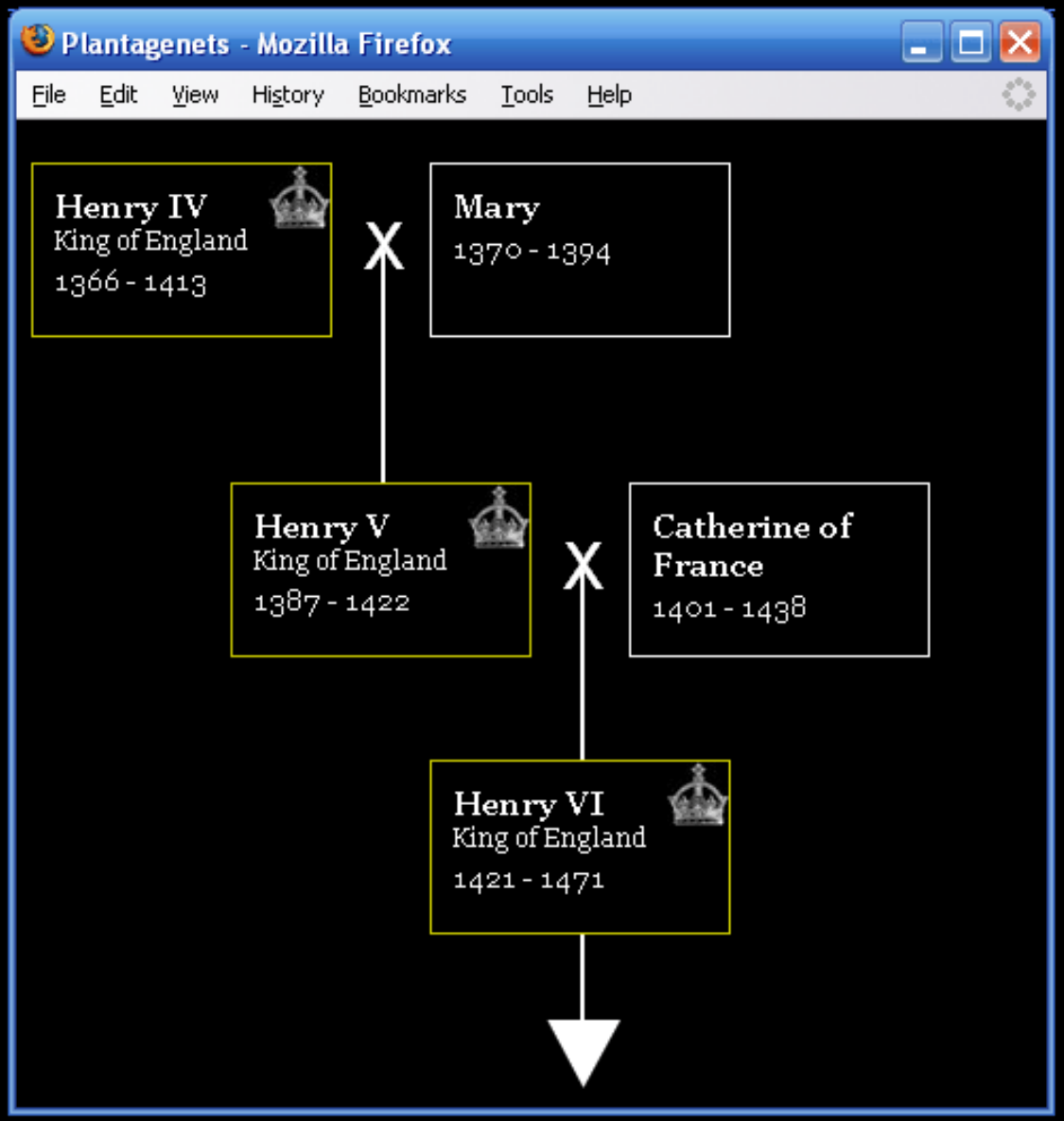
1) Store all data you receive, so that you never have to request it again. (Rather obvious.)

2) Define the problem: loading cascade.

3) Decide who will do the work: JavaScript, or PHP.

Doing the work - JavaScript?

Receive XML, parse it for the IDs we need, and send out a new request.



XML

```
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>
```

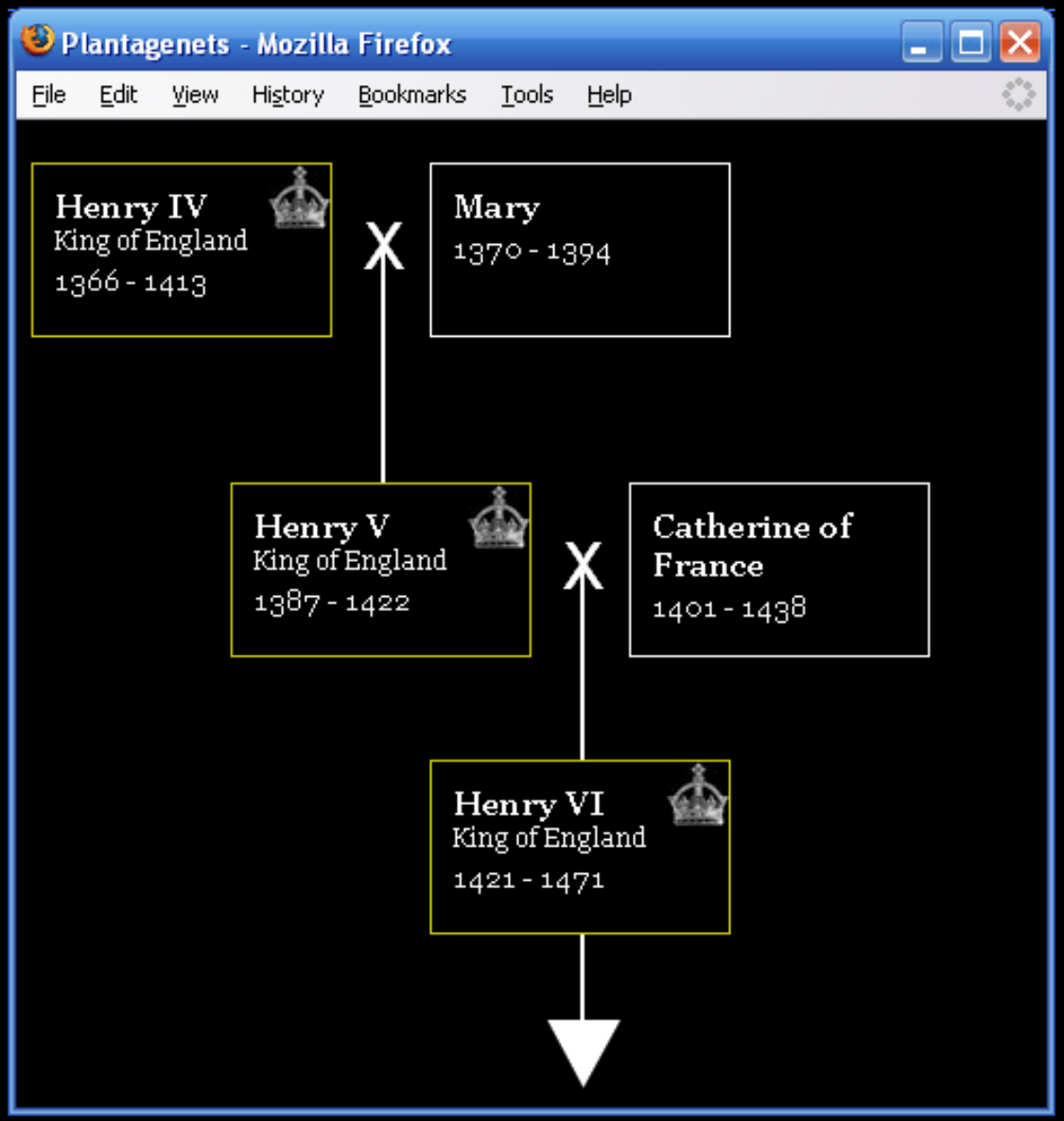
XML

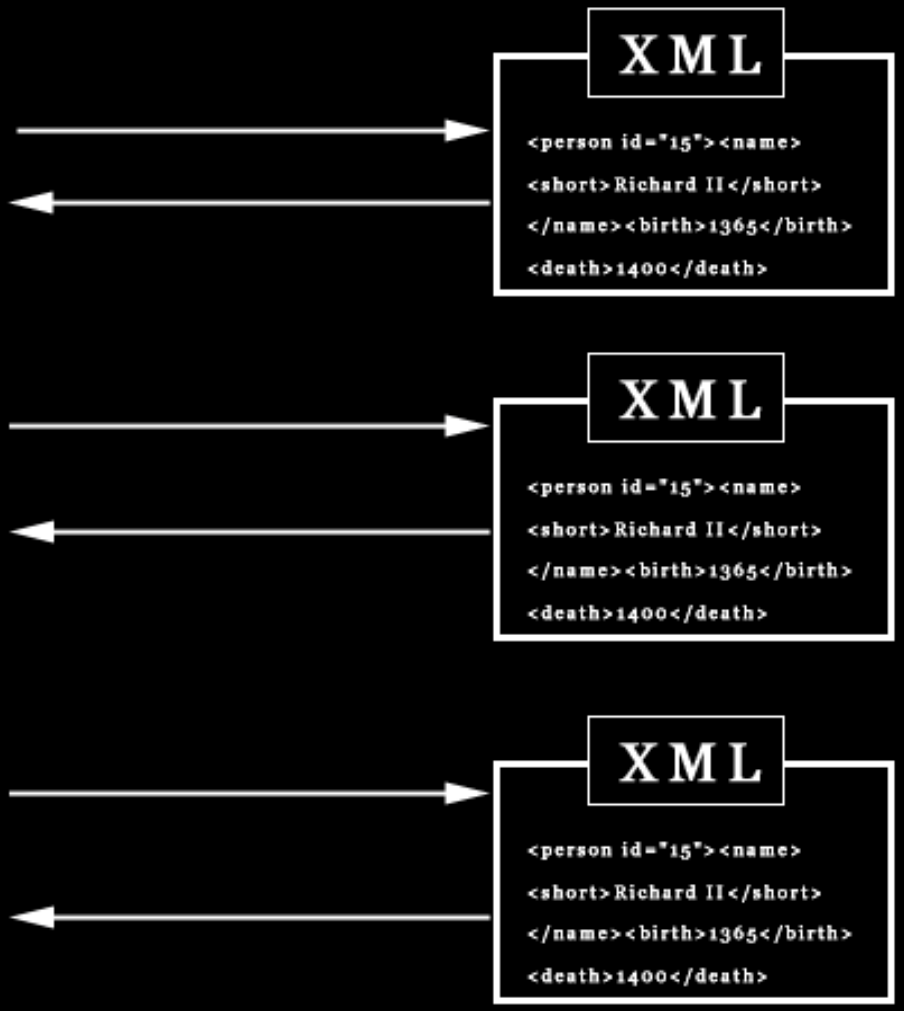
```
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>
<father idref="3">Edward
</father><mother idref="8">
Joan</mother><ranks><rank>
<predecessor idref="1">
Edward III</predecessor>
<title>King of England</title>
<start>1377</start><end>1399
</end><successor idref="17">
Henry IV</successor></rank>
</ranks></person>
```

Doing the work - JavaScript?

Receive XML, parse it for the IDs we need, and send out a new request.

Feasible, but in complicated situations you might need a few requests before you can show the data.

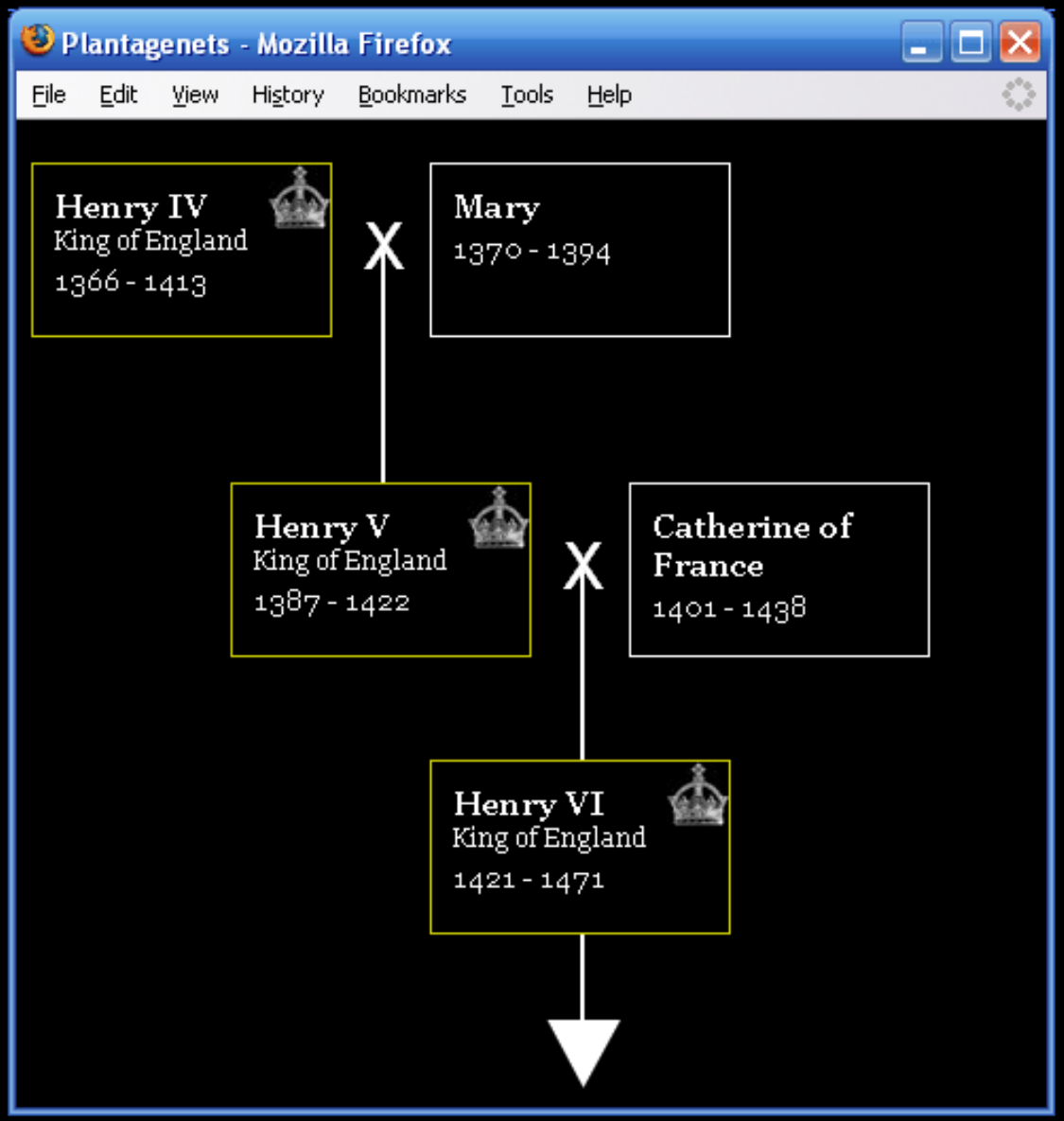

Plantagenets - Mozilla Firefox
File
Edit
View
History
Bookmarks
Tools
Help
Henry IV
King of England
1366 - 1413
X
Mary
1370 - 1394
Henry V
King of England
1387 - 1422
X
Catherine of France
1401 - 1438
Henry VI
King of England
1421 - 1471

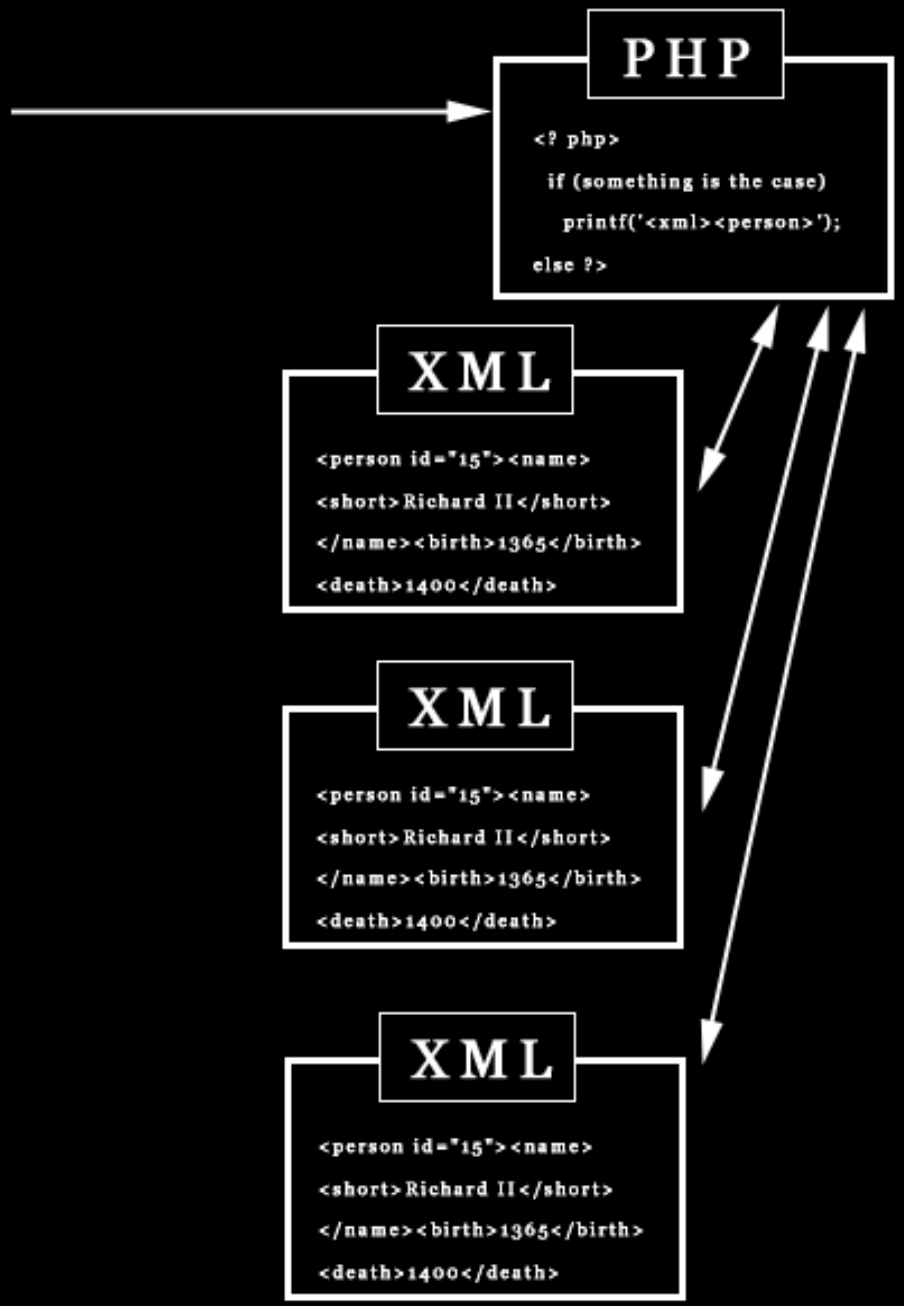

XML
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>
XML
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>
XML
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>

Doing the work - PHP?

Send request for something like
“/children/spouses/parents”

PHP interprets this as
1) Find children of selected person
2) Find their spouses
3) Find their parents


Plantagenets - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Henry IV
King of England
1366 - 1413
X
Mary
1370 - 1394
Henry V
King of England
1387 - 1422
X
Catherine of France
1401 - 1438
Henry VI
King of England
1421 - 1471

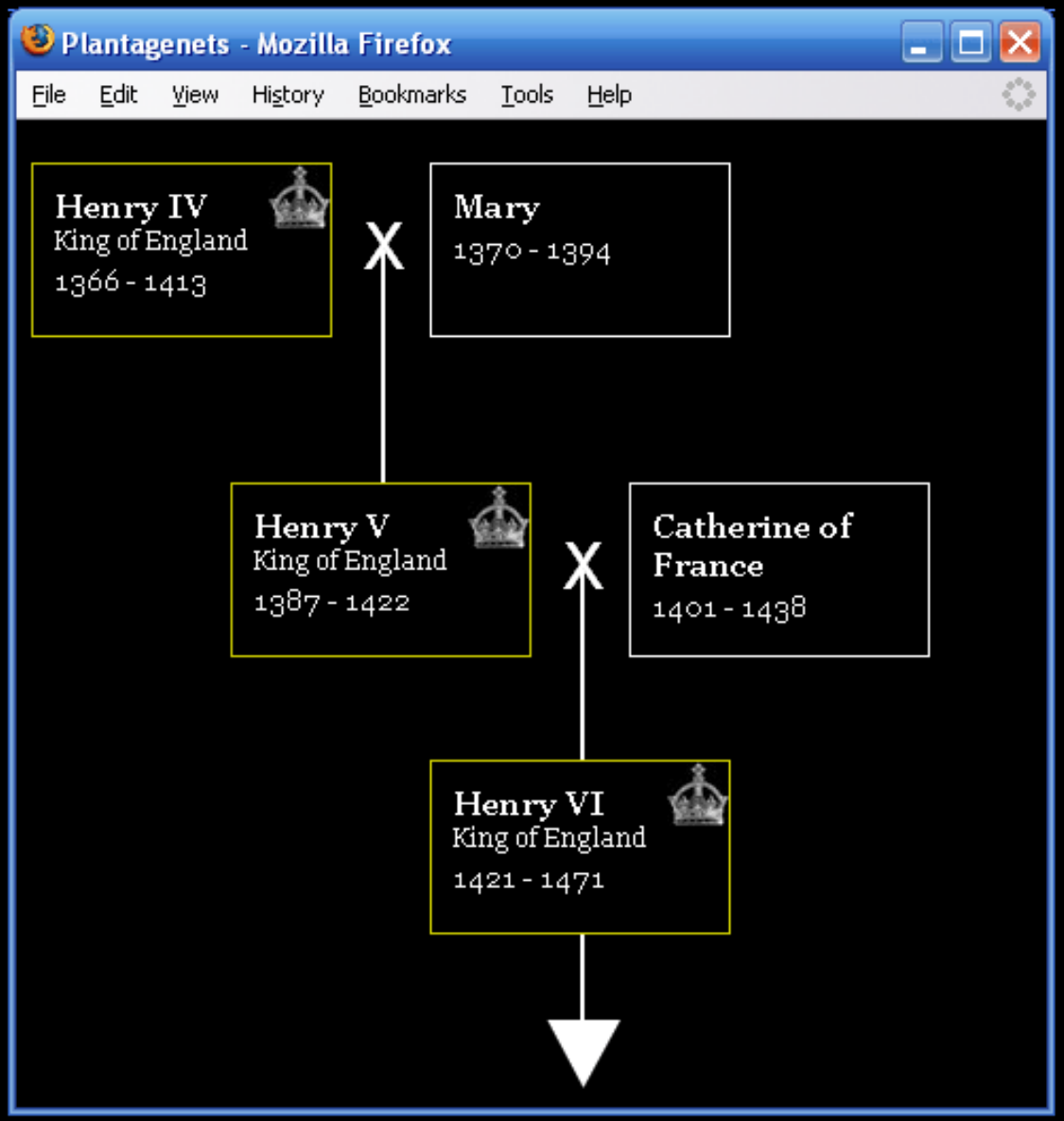

PHP
<? php>
if (something is the case)
printf('<xml><person>');
else ?>
XML
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>
XML
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>
XML
<person id="15"><name>
<short>Richard II</short>
</name><birth>1365</birth>
<death>1400</death>

Doing the work - PHP?

Send request for something like
“/children/spouses/parents”

PHP interprets this as
1) Find children of selected person
2) Find their spouses
3) Find their parents

Then send back all this info in one XML file.



XML

```
<person id="15">
<name>
<short>Richard II</short>
</name>
<birth>1365</birth>
<death>1400</death>
<father idref="3">Edward
</father>
<mother idref="8">
Joan</mother>
<ranks>
<rank>
<predecessor idref="1">
Edward III</predecessor>
<title>King of England</title>
<start>1377</start>
<end>1399
</end>
<successor idref="17">
Henry IV</successor>
</rank>
</ranks>
</person>
```

Decision time

Who will search for the data: JavaScript or PHP?

Partly depends on programming skills.

Decision time

General rule:

Assume the server is faster than the browser.

(Source: Yahoo!; see especially http://yuiblog.com/blog/2006/11/28/performance-research-part-1/)

Decision time

So it's best to gather all XML files in PHP and send them to the browser in one batch.

One other possibility: preloading

Preloading

Preload data while the user is busy studying other data.

Hardly ever discussed; found only one article (http://particletree.com/features/preloading-data-with-ajax-and-json/)

Preloading

Problem: how do we know which data the user wants to see next?

We don't.

Especially not in a dynamic environment such as family trees.

Preloading

So preloading cannot be used for the time being.

We're left with the PHP solution.

Conclusions

When an Ajax solution is proposed, always wonder if the same effect can be obtained by using frames.

If “Yes”, ask yourself whether Ajax is really needed.

Conclusions

Despite JSON being the better format in the long run, right now XML is the best way of communicating with the server.

NOT because of the “X” in Ajax but because the average third party will have heard of it.

Conclusions

Why

My family tree application needs a sophisticated load strategy that allows for a loading cascade.

This topic is underreported.

Conclusions

Why

The server should do most of the data-collection work, because it's faster than the client.

Conclusions

Why

For the time being, there are more questions than answers when working with Ajax.

If you find answers, write them down and publish them!

Why

Thank you