Extracting data from XML

- Wednesday
  DTL

Parsing - XML package

- 2 basic models - DOM & SAX
  - Document Object Model (DOM)
    Tree stored internally as C, or as regular R objects
    - Use XPath to query nodes of interest, extract info.
    - Write recursive functions to "visit" nodes, extracting information as it descends tree
    - extract information to R data structures via handler functions that are called for particular XML elements by matching XML name
  - For processing very large XML files with low-level state machine via R handler functions - closures.

Preferred Approach

- DOM (with internal C representation and XPath)
- Given a node, several operations
  - xmlName() - element name (w/w.o. namespace prefix)
    xmlNamespace()
  - xmlAttrs() - all attributes
    xmlGetAttr() - particular value
  - xmlValue() - get text content.
  - xmlChildren(), node[[i]], node [["el-name"]]
  - xmlSApply()
  - xmlNamespaceDefinitions()

Examples

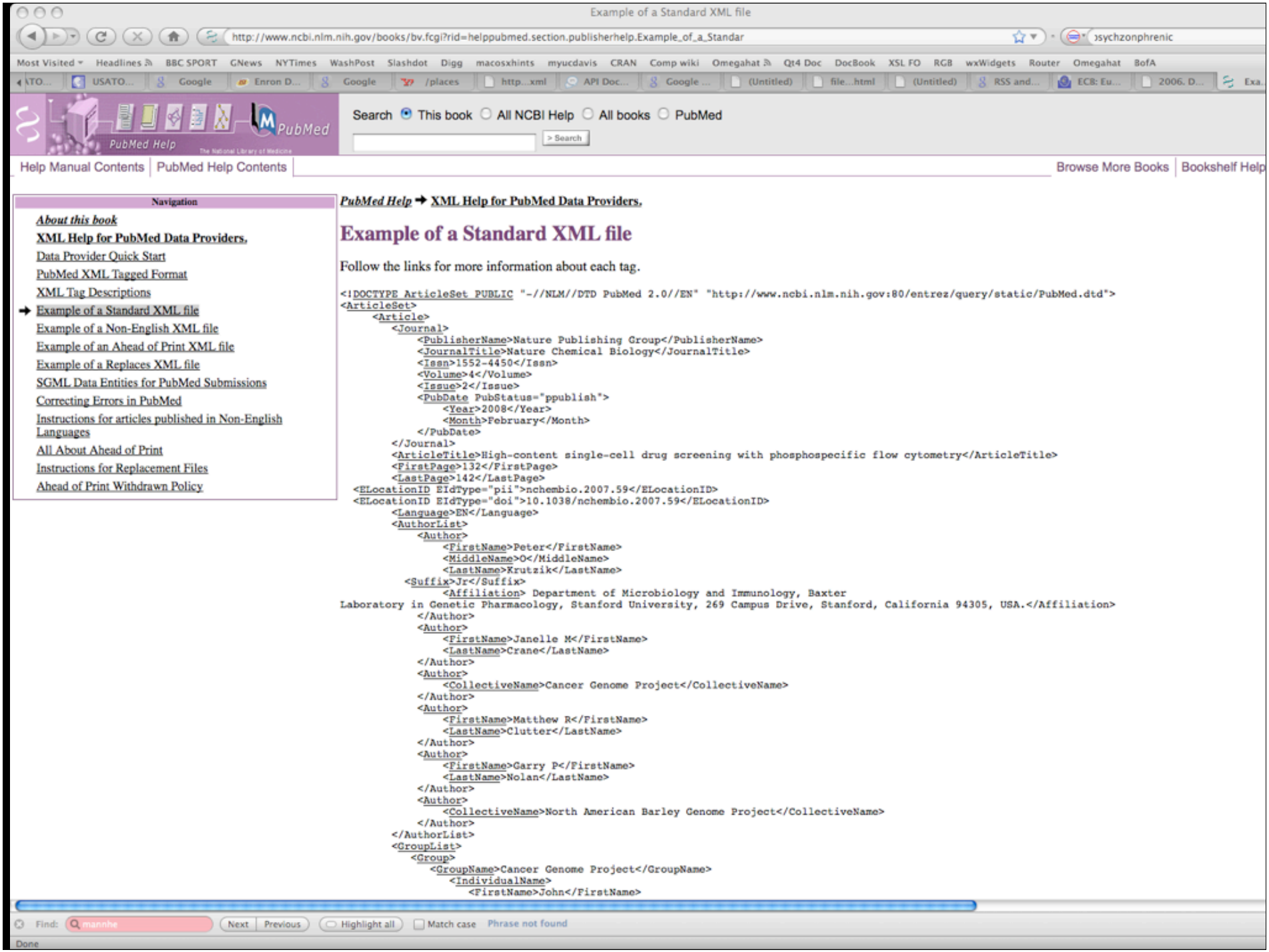
- Scraping HTML - (you name it!)
- zillow - house price estimates
- PubMed articles/abstracts
- European Bank exchange rates
- itunes - CDs, tracks, play lists, ...
- PMML - predictive modeling markup language
- CIS - Current Index of Statistics/Google Scholar
- Google - Page Rank, Natural Language Processing
- Wikipedia - History of changes, ...
- SBML - Systems biology markup language
- Books - Docbook
- SOAP - eBay, KEGG, ...
- Yahoo Geo/places - given name, get most likely location

PubMed

- Professionally archived collection of "medically-related" articles.
- Vast collection of information, including
  - article abstracts
  - submission, acceptance and publication date
  - authors
  - ...

PubMed

- We'll use a sample PubMed example article for simplicity.
  Can get very large, rich <ArticleSet> with many articles via an HTTP query done from within R/XML package directly.
- Take a look at the data, see what is available or read the documentation
  Or explore the contents.
- http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=helppubmed.section.publisherhelp.XML_Tag_Descriptions


Example of a Standard XML file
http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=helppubmed.section.publisherhelp.Example_of_a_Standar
Search This book All NCBI Help All books PubMed
> Search
Help Manual Contents
PubMed Help Contents
Browse More Books
Bookshelf Help
Navigation
About this book
XML Help for PubMed Data Providers.
Data Provider Quick Start
PubMed XML Tagged Format
XML Tag Descriptions
Example of a Standard XML file
Example of a Non-English XML file
Example of an Ahead of Print XML file
Example of a Replaces XML file
SGML Data Entities for PubMed Submissions
Correcting Errors in PubMed
Instructions for articles published in Non-English Languages
All About Ahead of Print
Instructions for Replacement Files
Ahead of Print Withdrawn Policy
PubMed Help ➜ XML Help for PubMed Data Providers.
Example of a Standard XML file
Follow the links for more information about each tag.
<!DOCTYPE ArticleSet PUBLIC "-//NLM//DTD PubMed 2.0//EN" "http://www.ncbi.nlm.nih.gov:80/entrez/query/static/PubMed.dtd">
<ArticleSet>
<Article>
<Journal>
<PublisherName>Nature Publishing Group</PublisherName>
<JournalTitle>Nature Chemical Biology</JournalTitle>
<Issn>1552-4450</Issn>
<Volume>4</Volume>
<Issue>2</Issue>
<PubDate PubStatus="ppublish">
<Year>2008</Year>
<Month>February</Month>
</PubDate>
</Journal>
<ArticleTitle>High-content single-cell drug screening with phosphospecific flow cytometry</ArticleTitle>
<FirstPage>132</FirstPage>
<LastPage>142</LastPage>
<ELocationID EIdType="pii">nchembio.2007.59</ELocationID>
<ELocationID EIdType="doi">10.1038/nchembio.2007.59</ELocationID>
<Language>EN</Language>
<AuthorList>
<Author>
<FirstName>Peter</FirstName>
<MiddleName>O</MiddleName>
<LastName>Krutzik</LastName>
<Suffix>Jr</Suffix>
<Affiliation> Department of Microbiology and Immunology, Baxter Laboratory in Genetic Pharmacology, Stanford University, 269 Campus Drive, Stanford, California 94305, USA.</Affiliation>
</Author>
<Author>
<FirstName>Janelle M</FirstName>
<LastName>Crane</LastName>
</Author>
<Author>
<CollectiveName>Cancer Genome Project</CollectiveName>
</Author>
<Author>
<FirstName>Matthew R</FirstName>
<LastName>Clutter</LastName>
</Author>
<Author>
<FirstName>Garry P</FirstName>
<LastName>Nolan</LastName>
</Author>
<Author>
<CollectiveName>North American Barley Genome Project</CollectiveName>
</Author>
</AuthorList>
<GroupList>
<Group>
<GroupName>Cancer Genome Project</GroupName>
<IndividualName>
<FirstName>John</FirstName>
Find: mannhe Next Previous Highlight all Match case Phrase not found
Done

- doc = xmlTreeParse("pubmed.xml", useInternal = TRUE)
- top = xmlRoot(doc)
- xmlName(top)
  [1] "ArticleSet"
- names(top) - child nodes of this root
  [1] "Article" "Article" - so 2 articles in this set.

- Let's fetch the author list for each article.
  Do it first for just one and then use "apply" to iterate

- names(top[[1]])

```
          Journal     ArticleTitle        FirstPage
        "Journal"  "ArticleTitle"     "FirstPage"
         LastPage     ELocationID     ELocationID
       "LastPage"   "ELocationID"  "ELocationID"
         Language      AuthorList      GroupList
       "Language"    "AuthorList"    "GroupList"
    ArticleIdList          History        Abstract
  "ArticleIdList"        "History"     "Abstract"
      ObjectList
    "ObjectList"
```

- art = top[[1]] [["AuthorList"]]
  what we want

- names(art)
  [1] "Author" "Author" "Author" "Author" "Author"
  "Author"
- names(art[[1]])
  [1] "FirstName" "MiddleName" "LastName" "Suffix"
  [5] "Affiliation"
- So how do we get these values, e.g. to put in a data frame.
- Each element is a node with text content.

- So loop over the nodes and get the content as a string

  xmlSApply(art[[1]], xmlValue)

- To do this for all authors of the article

  xmlSApply(art, function(x) xmlSApply(x, xmlValue))

- How do we deal with the different types of fields in the names?
  e.g. First, Middle, Last, Affiliation
  CollectiveName
  data representation/analysis question from here.

Pubmed Dates

- In the <History> element, have date
  received, accepted, aheadofprint
- May want to look at time publication lag (i.e. received to publication time) for different journals.
- So get these dates for all the articles

```
<History>
 <PubDate PubStatus="received">
   <year>...</year><Month>06</Month><Day>15</Day>
 <PubDate>
 <PubDate PubStatus="accepted">
  <year>.....</day>
 </PubDate>
```

- Find the element PubDate within History which has an attribute whose value is "received"
- Can use art[["History"]][["PubDate"]] to get all 3 elements.
- But what if we want to access the 'received' dates for all the articles in a single operation, then the accepted, ...
- Need a language to identify nodes with a particular characteristic/condition

XPath

- XPath is a language for expressing such node subsetting with rich semantics for identifying nodes
  - by name
  - with specific attributes present
  - with attributes with particular values
  - with parents, ancestors, children
- XPath = YALTL (Yet another language to learn)

XPath language

- /node - top-level node
- //node - node at any level
- node[@attr-name] - node that has an attribute named "attr-name"
- node[@attr-name='bob'] - node that has attribute named attr-name with value 'bob'
- node/@x - value of attribute x in node with such attr.
- Returns a collection of nodes, attributes, etc.

- Let's find the date when the articles were received
- nodes = getNodeSet(top,
  "//History/PubDate[@PubStatus='received']")
- 2 nodes - 1 per article
- Extract year, month, day
  lapply(nodes, function(x) xmlSApply(x, xmlValue))
- Easy to get date "accepted" and "aheadofprint"

Text mining of abstract

- Content of abstract as words
- abstracts = xpathApply(top, "//Abstract", xmlValue)
- Now, break up into words, stem the words, remove the stop-words,
- abstractWords = lapply(abstracts, strsplit, "[[:space:]]")
- library(Rstem)
  abstractWords = lapply(abstractWords,
  function(x) wordStem[[1]])
- Remove stop words
  lapply(abstractWords, function(x) x[x %in% stopWords])

Zillow - house prices

- Thanks to Roger, yesterday evening I found the Zillow XML API - (Application Programming Interface)
- Can register with Zillow, make queries to find estimated house prices for a given house, comparables, demographics, ...
- Put address, city-state-zip & Zillow login in URL request
- Can put this at the end of a URL within xmlTreeParse() "http://www.zillow.com/...../...?zws-id=...&address=1029%20Bob's%20Way&citstatezip=Berkeley"
- But spaces are problematic, as are other characters.

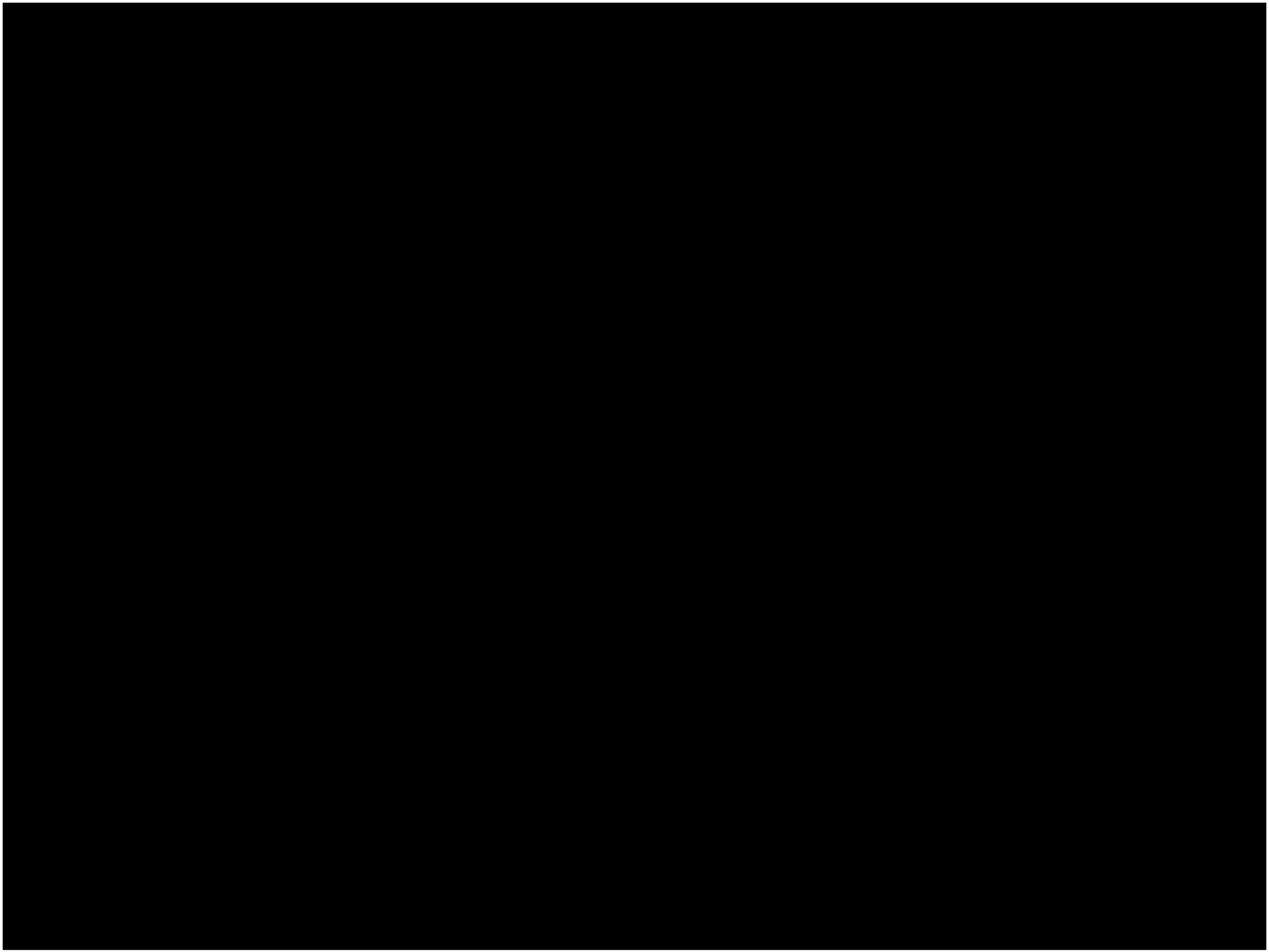
- So I use library(RCurl)
- reply = getForm("http://www.zillow.com/webservice/GetSearchResults.htm",
  'zws-id' = "AB-XXXXXXXXXXX_10312q",
  address = "1093 Zuchini Way",
  citystatezip = "Berkeley, CA, 94212")
- reply is text from the Web server containing XML

<?xml version=\"1.0\" encoding=\"utf-8\"?>\n<SearchResults:searchresults xsi:schemaLocation=\"http://www.zillow.com/static/xsd/SearchResults.xsd /vstatic/71a179109333d30cfb3b2de866d9add9/static/xsd/SearchResults.xsd\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:SearchResults=\"http://www.zillow.com/static/xsd/SearchResults.xsd\">\n\n   <request>\n<address>112 Bob's Way Avenue</address>\n   <citystatezip>Berkeley, CA, 94212</citystatezip>\n   </request>\n   \n   <message>\n   <text>Request successfully processed</text>\n   <code>0</code>\n\t\t\n   </message>\n\n\n   <response>\n\t\t<results>\n\t\t\t\n\t\t\t<result>\n\t\t\t\t\t<zpid>24842792</zpid>\n\t<links>\n\t\t<homedetails>http://www.zillow.com/HomeDetails.htm?city=Berkeley&state=CA&zprop=24842792&s_cid=Pa-Cv-X1-CLz1carc3c49ms_htxqb&partner=X1-CLz1carc3c49ms_htxqb</homedetails>\n\t\t<graphsanddata>http://www.zillow.com/Charts.htm?chartDuration=5years&zpid=24842792&cbt=8965965681136447050%7E1%7E43-17yrvL7nIj-Y5pqbsoqb_nh1QW4CVIhubJRAXIOkwbPosbEGChw**&s_cid=Pa-Cv-X1-CLz1carc3c49ms_htxqb&partner=X1-CLz1carc3c49ms_htxqb</graphsanddata>\n\t\t<mapthishome>http://www.zillow.com/search/RealEstateSearch.htm?zpid=24842792#src=url&s_cid=Pa-Cv-X1-CLz1carc3c49ms_htxqb&partner=X1-CLz1carc3c49ms_htxqb</mapthishome>\n\t\t<myestimator>http://www.zillow.com/myestimator/Edit.htm?zprop=24842792&s_cid=Pa-Cv-X1-CLz1carc3c49ms_htxqb&partner=X1-CLz1carc3c49ms_htxqb</myestimator>\n\t\t<myzestimator deprecated=\"true\">http://www.zillow.com/myestimator/Edit.htm?zprop=24842792&s_cid=Pa-Cv-X1-CLz1carc3c49ms_htxqb&partner=X1-CLz1carc3c49ms_htxqb</myzestimator>\n\t</links>\n\t<address>\n\t\t<street>1292 Bob's way</street>\n\t\t<zipcode>94</zipcode>\n\t\t<city>Berkeley</city>\n\t\t<state>CA</state>\n\t\t<latitude>34.882544</latitude>\n\t\t<longitude>-123.11111</longitude>\n\t</address>\n\t\n\t\n\t<zestimate>\n\t\t<amount currency=\"USD\">803000</amount>\n\t\t<last-updated>07/14/2008</last-updated>\n\t\t\n\t\t\n\t\t<oneWeekChange deprecated=\"true\"></oneWeekChange>\n\t\t\n\t\t\n\t\t<valueChange currency=\"USD\" duration=\"31\">-33500</valueChange>\n\t\t\n\t\t\n\t\t<valuationRange>\n\t\t\t<low currency=\"USD\">650430</low>\n\t\t\t

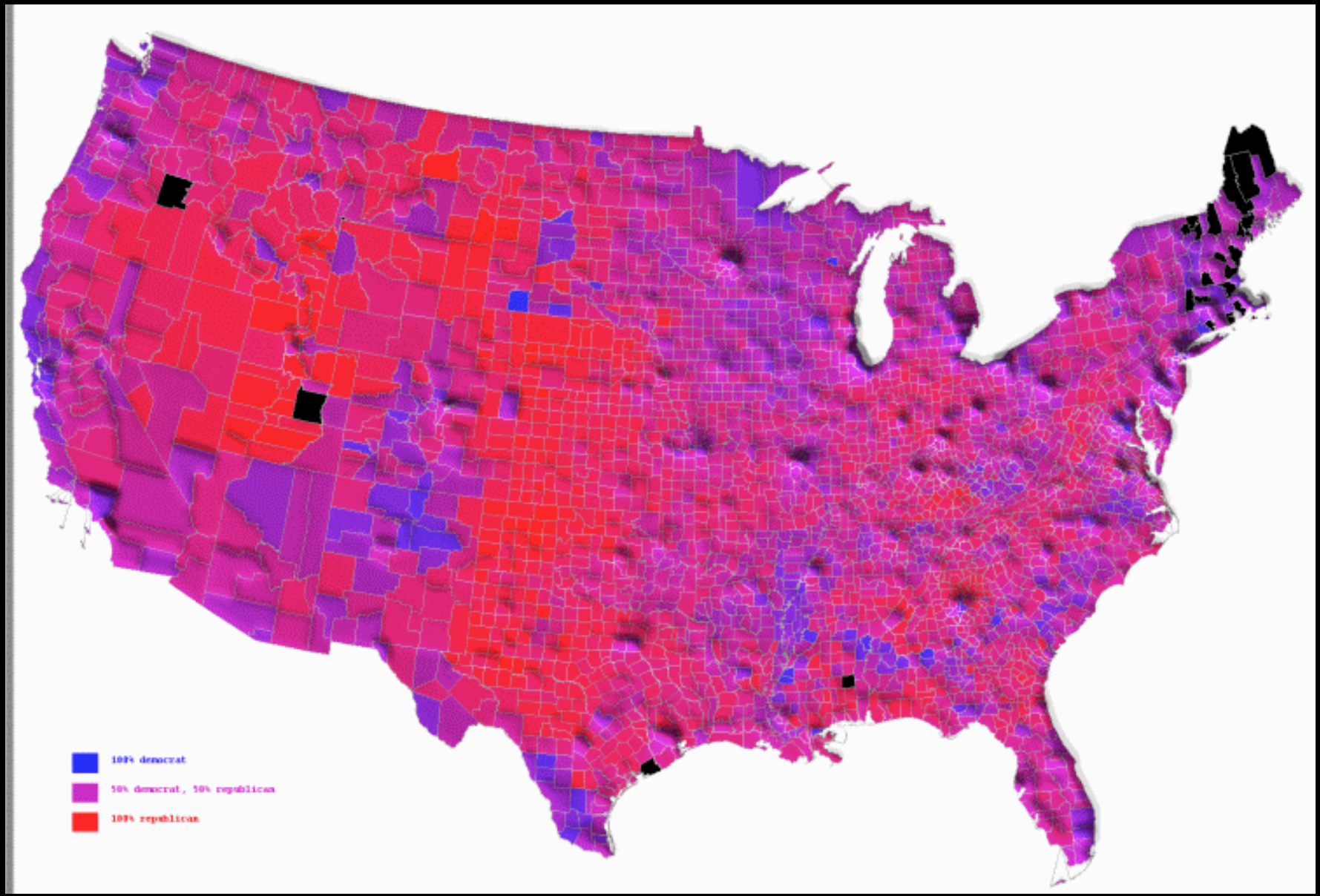
```
<?xml version="1.0" encoding="utf-8"?>
<SearchResults:searchresults xsi:schemaLocation="http://
www.zillow.com/static/xsd/SearchResults.xsd /vstatic/
71a179109333d30cfb3b2de866d9add9/static/xsd/SearchResults.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SearchResults="http://www.zillow.com/static/xsd/
SearchResults.xsd">

    <request>
        <address>123 Bob's Way</address>
        <citystatezip>Berkeley, CA, 94217</citystatezip>
    </request>

    <message>
        <text>Request successfully processed</text>
        <code>0</code>

    </message>


    <response>
     <results>

        <result>
            <zpid>1111111</zpid>
  <links>
```

Processing the result

- We want to get the value of the element
  <amount>803000</amount
- doc =
  xmlTreeParse(reply, asText = TRUE, useInternal = TRUE)
- xmlValue(doc[["//amount"]])
  [1] "803000"
- Other information too



2004 Election Results



http://www.princeton.edu/~rvdb/JAVA/election2004/

Where are the data?

- Within days of the election ?
  USA Today, CNN, ...
- http://www.usatoday.com/news/politicselections/vote2004/results.htm
- By state, by county, by senate/house, ...

Election 2004

Presidential vote by county - New Jersey

Presidential Results - By County

County	Total Precincts	Precincts Reporting	Bush	Kerry	Nader
Atlantic	158	158	46,197	52,181	535
Bergen	557	557	178,304	192,827	1,891
Burlington	359	359	90,112	103,971	945
Camden	331	331	76,925	129,918	894
Cape May	131	131	26,316	19,614	241
Cumberland	93	93	23,186	26,410	154
Essex	567	566	80,822	191,999	1,052
Gloucester	237	237	59,760	66,476	734
Hudson	452	416	55,530	113,603	831
Hunterdon	113	113	39,449	25,727	484
Mercer	265	264	53,469	85,682	771
Middlesex	597	597	119,436	156,168	1,701
Monmouth	437	437	161,693	131,808	2,005
Morris	395	395	126,761	90,476	1,154
Ocean	346	346	143,797	92,621	1,571
Passaic	288	288	73,568	91,939	833
Salem	45	45	15,635	13,650	165
Somerset	277	277	67,505	61,550	739
Sussex	107	106	42,085	22,282	480
Union	443	443	77,621	112,542	980
Warren	87	87	29,323	17,876	454

Updated: 11/11/2004 2:31 PM ET

Vote returns will appear shortly after polls close in each state or locality (click here for times) and will update automatically. Click refresh button for latest results. Winners of some races may be

- read.table ?
- Within the noise/ads, look for a table whose first cell is "County"
- Actually a
  <td>County</td>
- How do we know this? Look at one or two HTML files out of the 50. Verify the rest.
- Then, given the associated <table> element, we can extract the values row by row and get a data.frame/....

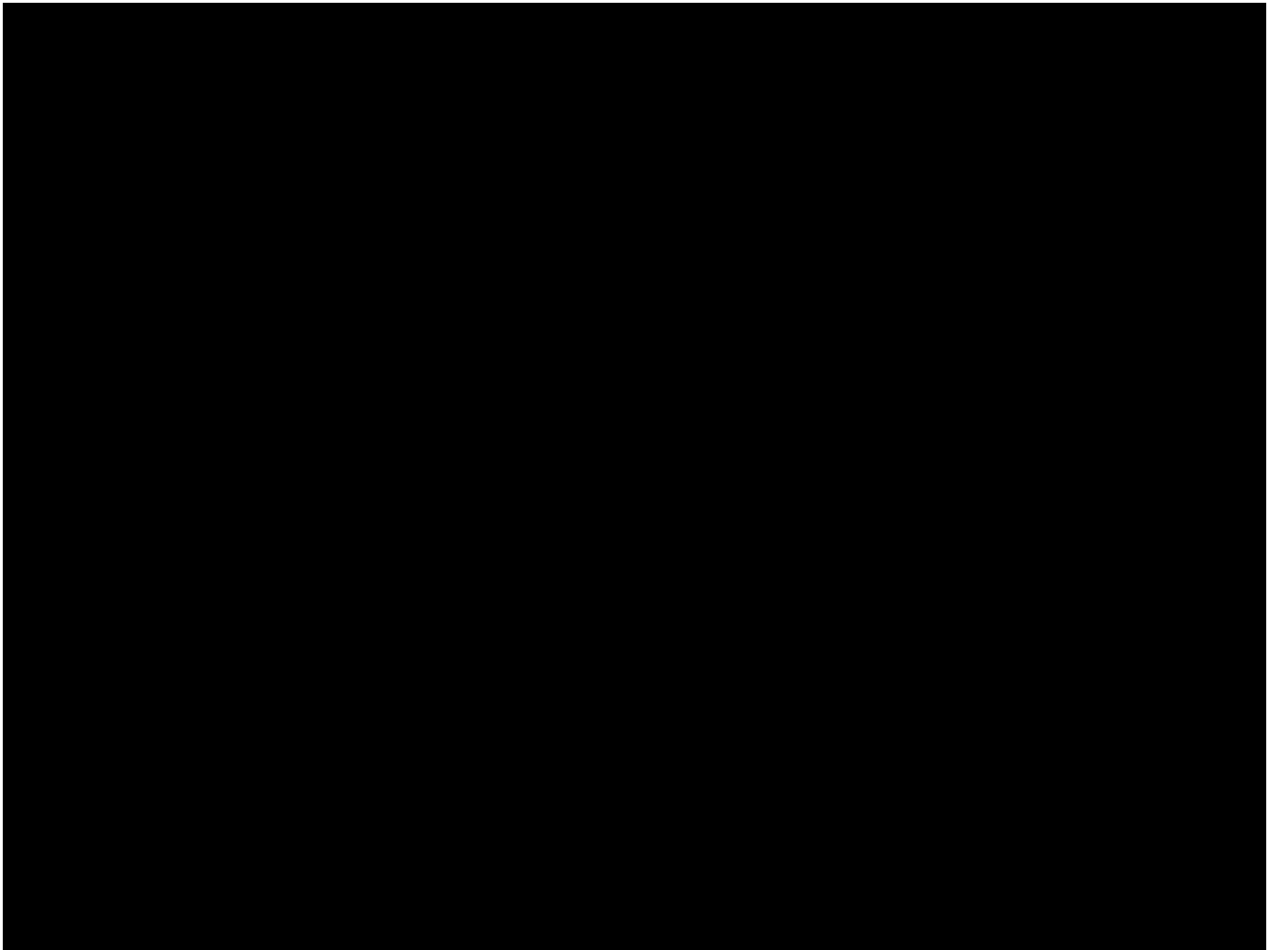
XPath expression

```
<table>........<tr>
          <td class="notch_medium" width="153"><b>County</
b></td><td class="notch_medium" align="Right"
width="65"><b>Total Precincts</b></td><td
class="notch_medium" align="Right" width="70"><b>Precincts
Reporting</b></td><td class="notch_medium" align="Right"
width="60"><b>Bush</b></td><td class="notch_medium"
align="Right" width="60"><b>Kerry</b></td><td
class="notch_medium" align="Right" width="60"><b>Nader</
b></td>

        </tr><
```

- Little bit of trial and error
- getNodeSet(nj, "//table[tr/td/b/text()='Total Precincts']")
- Could be more specific, e.g. tr[1] - first row

- Now that we have the <table> node, read the data into an R data structure
- rows = xmlApply(v[[1]],
  function(x)
  xmlSApply(x, xmlValue))
- i.e. for each row, loop over the <td> and get its value.
- Got some "\n\t\t\t" and last row is "Updated...."
  first row is the County, Total Precincts, ...
- So discard the rows without 7 entries
  then remove the 7th entry ("\n\t\t\t")

```
v = getNodeSet(nj, "//table[tr/td/b/text()='Total Precincts']")
rows = xmlApply(v[[1]], function(x) xmlSApply(x, xmlValue))

 # only the rows with 7 elements
rows = rows[sapply(rows, length) == 7]
# Remove the 7th element, and transpose to put back into
# counties as rows, precinct, candidates, ... as columns.
# So get a  matrix of  # counties by 6 matrix of character
# vectors.
rows = t(sapply(rows, "[", -7))
```



Learning XPath

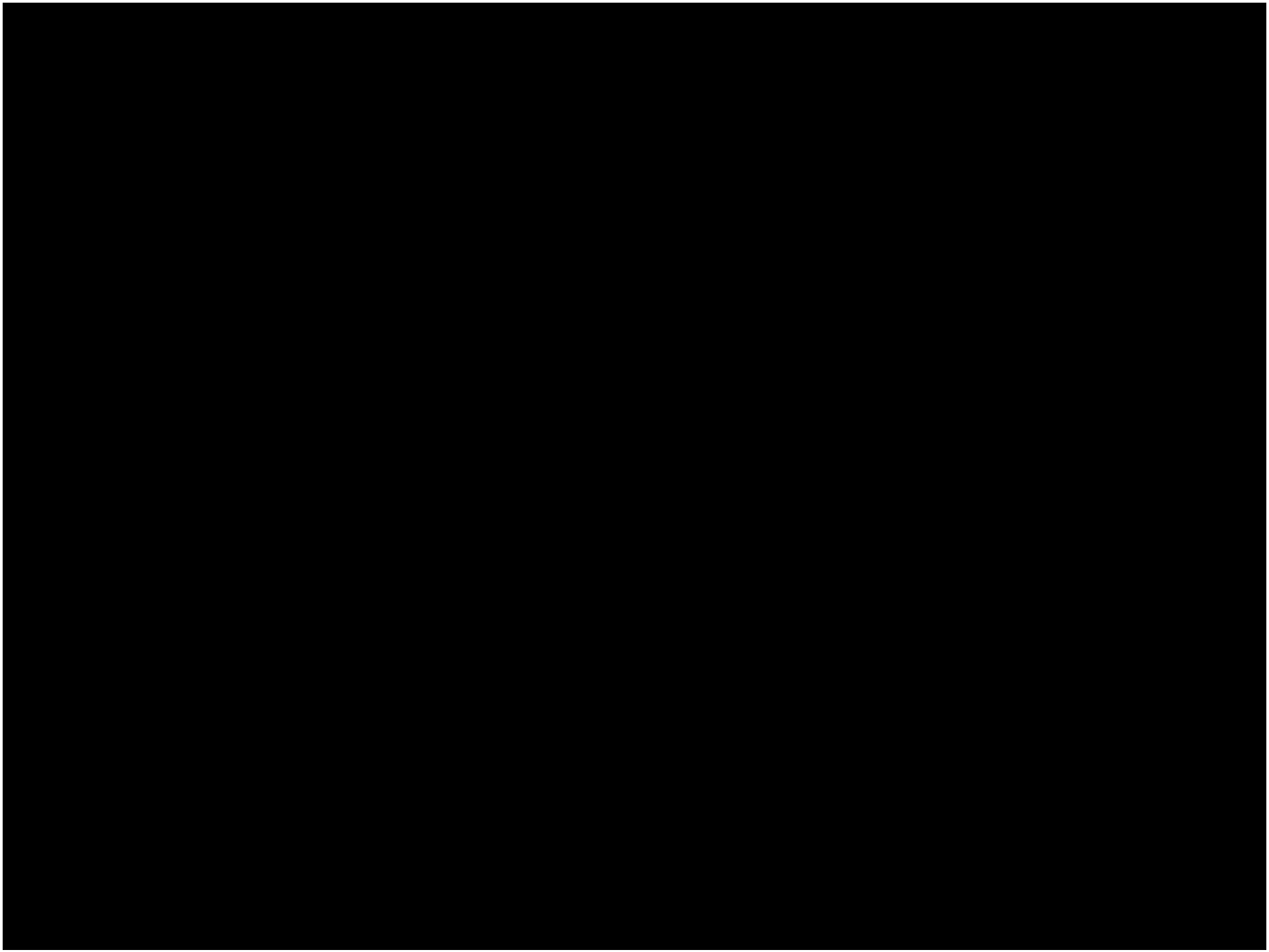
- XPath is another language
- part of the XML technologies
  - XInclude
  - XPointer
  - XSL
  - XQuery
- Can't we extract the data from the XML tree/DOM (Document Object Model) without it and just use R programming - Yes

- doc = xmlTreeParse("pubmed.xml")
- Now have a tree in R
  - recursive - list of children which are lists of children
  - or recursive tree of C-level nodes
- Write an R function which "visits" each node and extracts and stores the data from those nodes that are relevant
  - e.g. the <Author>, <PubDate> nodes

- Recursive functions are sometimes difficult to write
- Have to store the results "globally"/non-locally
  leads to closures/lexical scoping - "advanced R"
- Have to traverse the entire tree via R code - SLOW!

Handlers

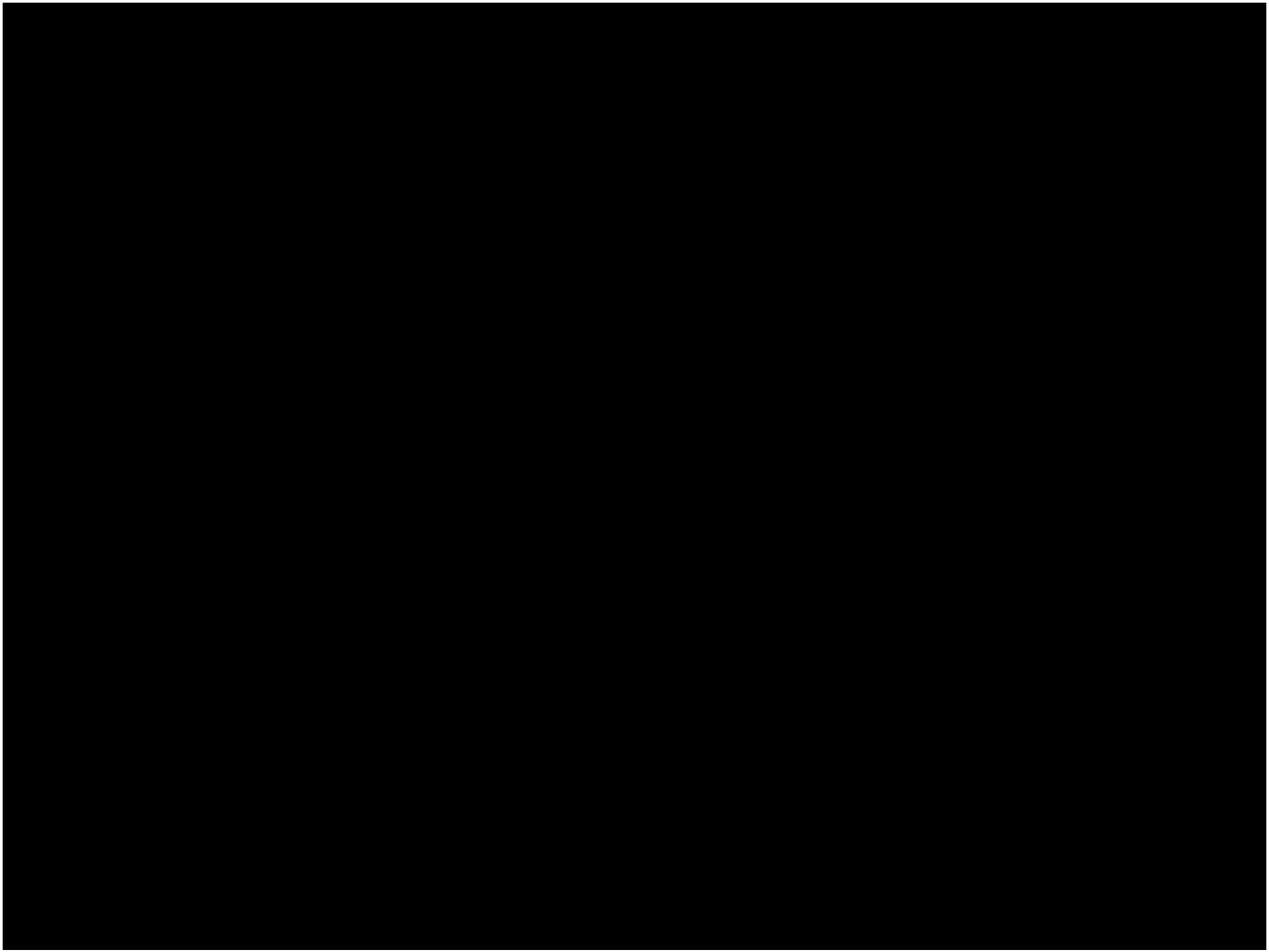
- Alternative approach
  - when we read the XML tree into R and convert it to a list of lists of children ...
  - when convert each C-level node, see if caller has a function registered corresponding to the name/type of node
    - if so call it and allow it to extract and store the data.



Efficient Parsing

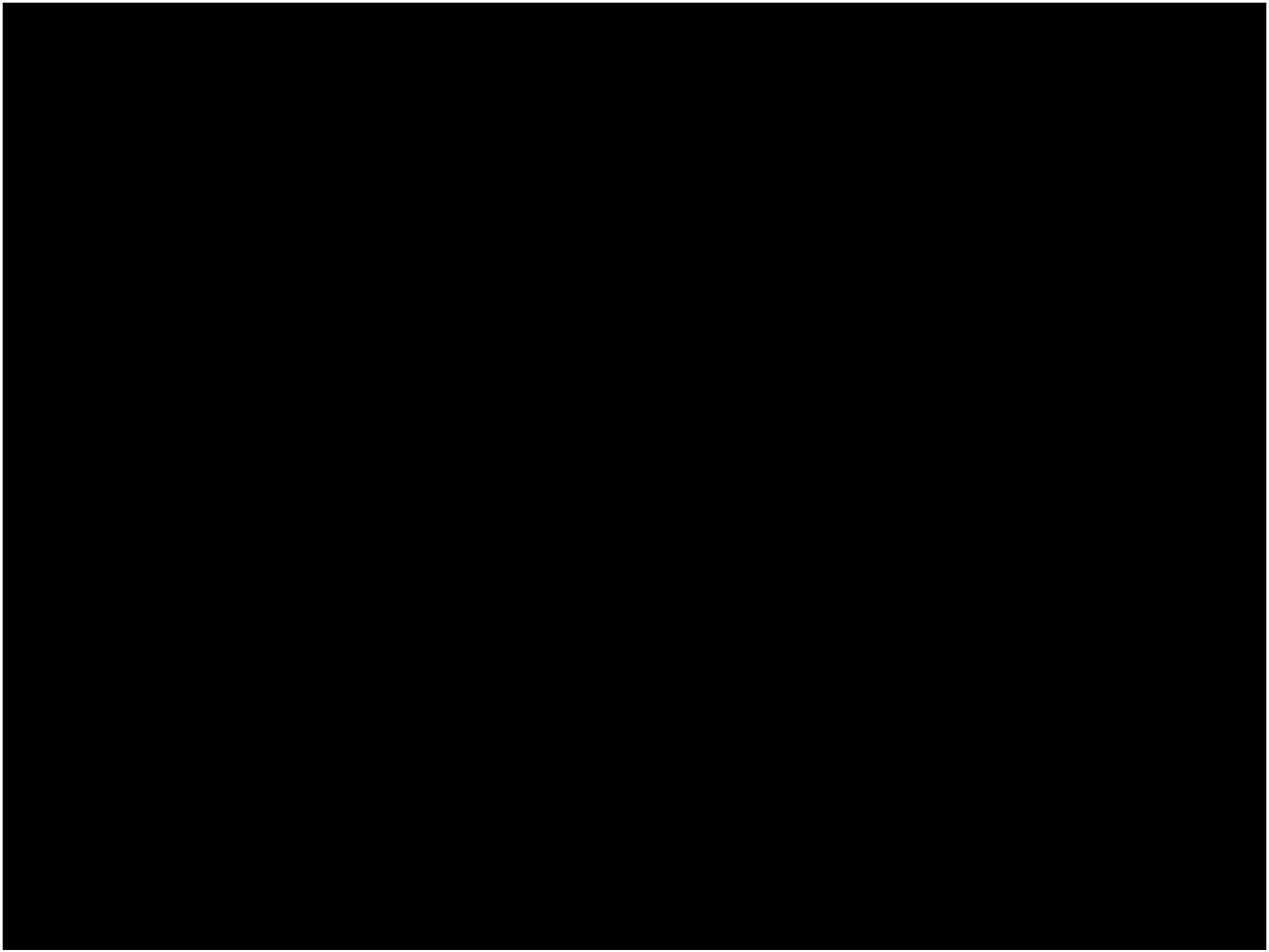
- Problem with previous styles is we have the entire tree in memory and then extract the data
  => 2 times the data in memory at the end
- Bad news for large datasets
  - All of Wikipedia pages - 11Gigabytes
- Need to read the XML as it passes as a stream, extracting and storing the contents and discarding the XML.
- SAX parsing - "Simple API for XML"!

- xmlEventParse(content,
  list(startElement = function(node, ...)....,
  endElement = function(node, ...) ...,
  text = function(x) ...,
  comment = function(x) ... , ...))
- Whenever XML parser sees start/end/text/comment node, calls R function which maintains state.
- Awkward to write, but there to handle very large data.



Schema....

- Just like a database has a schema describing the characteristics of columns in all tables within a database, XML documents often have an XML Schema (or Document Type Definition - DTD) describing the "template" tree and what elements can/must go where, attributes, etc.
- The XML Schema is written in XML, so we can read it!
- And we can actually create R data types to represent the same elements in XML directly in R.
- So we can automate some of the reading of XML elements into useful, meaning R objects harder to programmatically flatten into data frames.



RCurl

- xmlTreeParse() & xmlEventParse() can read from files, compressed files, URLs, direct text - but limited connection support.
- RCurl package provides very rich ways that extend R's ability to access content from URLs, etc. over the Internet.
- HTTPS - encrypted/secure HTTP
  passwords/authentication
  efficient, persistent connections
  multiplexing
  different protocols
- Pass results to XML parser or other consumers.

Exceptions/Conditions