Verarbeitung von XML-Dateien

Jörn Clausen
joern@TechFak.Uni-Bielefeld.DE

XML – Was geht mich das an?

- effektives Datenaustauschformat
- von Mensch und Maschine gut verarbeitbar
- selbstdokumentierend
- fortschreitende Verbreitung
- RNAML, PROXIML, MAGE-ML, Blast, . . .

XML verarbeiten – aber richtig

- Wahl der Qual: einfach, schnell oder speicherschonend
- einfach, aber hoher Speicherverbrauch
- schnell und klein, aber schlechte Programmstruktur
- ungünstige Kombinationen möglich

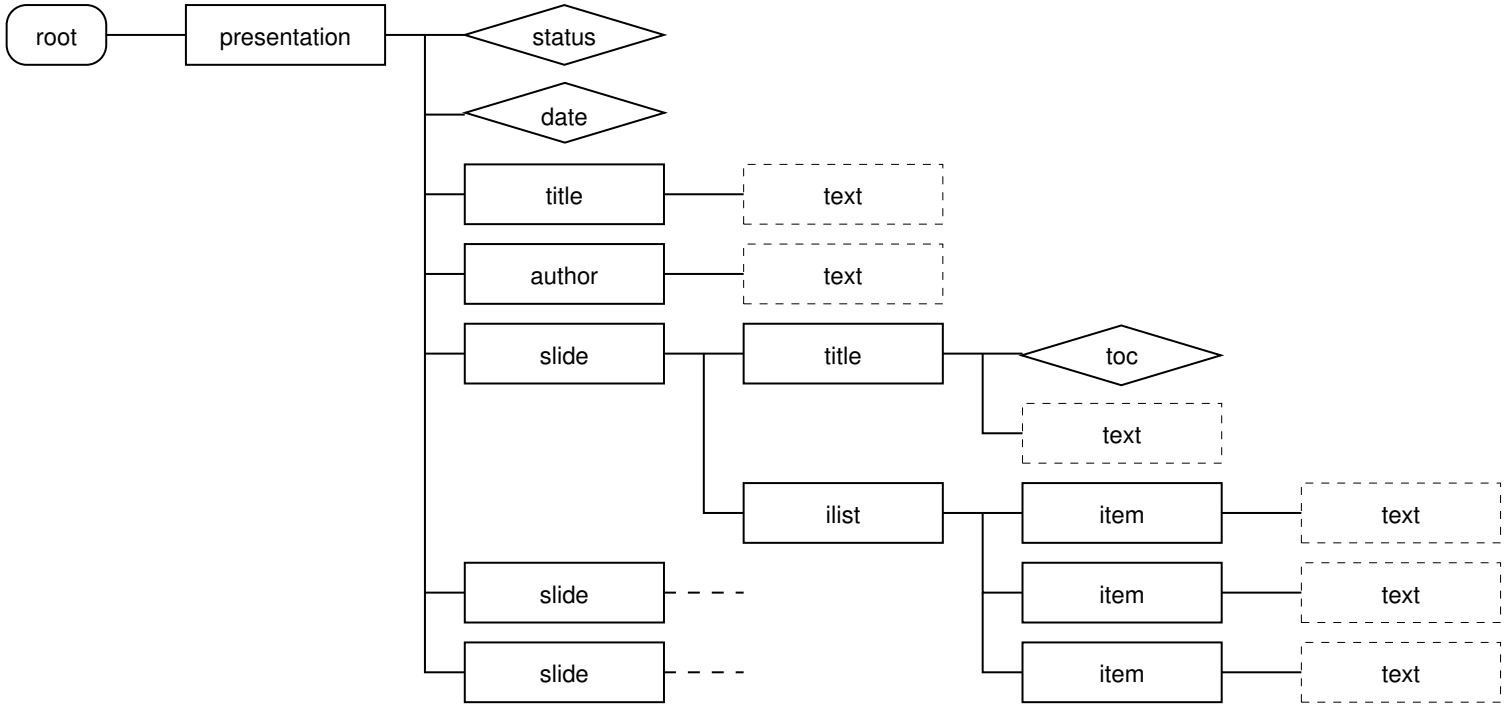
XML verarbeiten – aber richtig, cont.

- zwei Verfahren
  1. Baum-basiert:
     - einfache Benutzung
     - freier Zugriff auf alle Daten in konstanter Zeit
     - hoher Speicherverbrauch
     - hohe startup-Zeit
  2. Event-basiert:
     - geringer Speicherverbrauch
     - Ergebnisse bereits während der Verarbeitung ausgeben
     - kein look-ahead/look-behind
- . . . soweit die Theorie

ein Beispiel

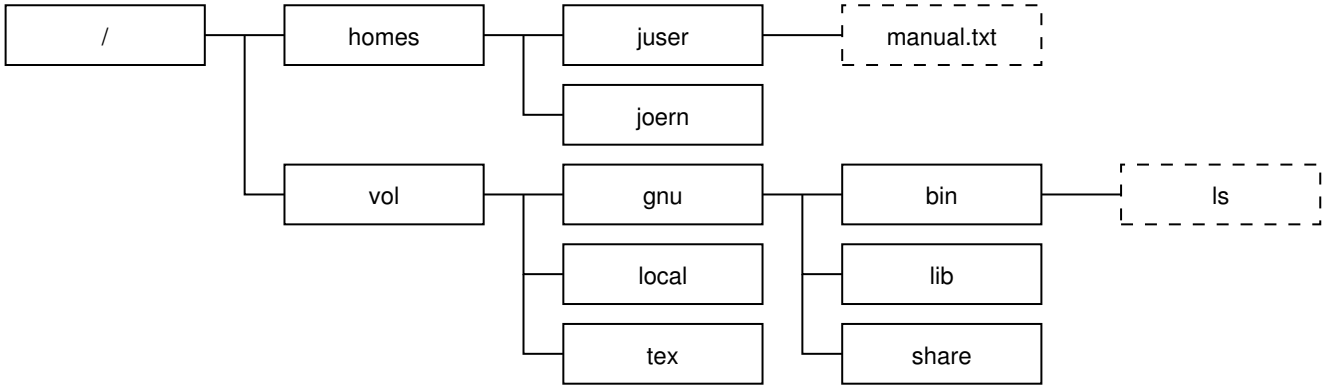
```
<?xml version="1.0"?>
<presentation status="draft" date="2002-10-04">
  <title>XML &amp; Friends for Dummies</title>
  <author>Joe User</author>
  <slide>
    <title toc="yes">What is XML?</title>
    <ilist>
      <item>XML is not a markup language (unlike HTML)</item>
      <item>XML instances can be <emph>well formed</emph> or
        even <emph>validating</emph></item>
      <item>XML stands for &xml;</item>
    </ilist>
  </slide>
  <slide>...</slide>
</presentation>
```

XML-Dokument als Baum



- weitere Text-Knoten durch whitespace
- Aufgabe: lokalisiere einen (oder mehrere) Knoten

Analogie Dateisystem



- absoluter Pfad: Wegbeschreibung vom Wurzelverzeichnis

  `/vol/gnu/bin/ls`

- relativer Pfad: Wegbeschreibung vom „aktuellen“ Verzeichnis

  `../juser/manual.txt`

XPath

- XPath beschreibt Pfade im XML-Baum
- Unterschiede gegenüber Dateisystem:
  - gleichnamige Kindknoten
  - Knotentypen: Elementknoten, Attributknoten, Textknoten, ...
- Beispiele:

```
slide
/presentation/slide/title
/presentation/@status
//slide/title
```

- Bedingungen in XPath-Ausdrücken:

```
/presentation/slide[position()=2]
/presentation/slide/[title/@toc="yes"]/ilist
```

biologische Daten in XML

- Organismus: Sinorhizobium meliloti
- XML-Version via Entrez

Chromosom	41.4 MByte
Plasmid pSymA	15.5 MByte
Plasmid pSymB	18.9 MByte

- Aufgabe: erzeuge HTML-Tabelle aller identifizierten Gene
- (scheinbar) redundante Elemente
- XML automatisch aus ASN.1 erzeugt

```
<Seq-feat>
  <Seq-feat_data>
    <SeqFeatData>
      <SeqFeatData_gene>
        <Gene-ref>
          <Gene-ref_locus>lacE</Gene-ref_locus>
            <Gene-ref_syn>
              <Gene-ref_syn_E>SMb21652</Gene-ref_syn_E>
            </Gene-ref_syn>
          </Gene-ref>
      ....
  </Seq-feat_data>
  <Seq-feat_location>
      ....
          <Seq-interval_from>0</Seq-interval_from>
          <Seq-interval_to>1274</Seq-interval_to>
          <Seq-interval_strand>
            <Na-strand value="plus"/>
          </Seq-interval_strand>
```

1. Verfahren: XSLT

- Extensible Stylesheet Language Transformations
- Transformation XML $\rightarrow$ XML oder XML $\rightarrow$ HTML
- templates, die auf XPath-Ausdrücke matchen
- zwei XSLT-Prozessoren:
  - xsltproc (C, Gnome-Projekt)
  - Saxon (Java, Michael Kay)

```
<xsl:template match="/">
  <html>
    <head>
      <title>Genes</title>
    </head>
    <body>
      <table width="100%">
        <tr>
          <th align="left">gene</th>
          <th align="left">reference</th>
          <th align="left">location</th>
        </tr>
        <xsl:apply-templates select=".//Seq-feat"/>
      </table>
    </body>
  </html>
</xsl:template>
```

```
<xsl:template match="Seq-feat">
  <xsl:if test=".//Gene-ref_locus">
    <tr>
      <td>
        <xsl:value-of select=".//Gene-ref_locus"/>
      </td>
      <td>
        <xsl:value-of select=".//Gene-ref_syn_E"/>
      </td>
      <td>
        <xsl:value-of select=".//Seq-interval_from"/>
        <xsl:text> - </xsl:text>
        <xsl:value-of select=".//Seq-interval_to"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select=".//Na-strand/@value"/>
      </td>
    </tr>
  </xsl:if>
</xsl:template>
```

2. Verfahren: XPath in Perl

- Knoten im XML-Baum als Objekte
- Zugriff auf Knoten über XPath-Ausdrücke
- z.B. Iteration über Array von Knoten-Objekten
- zwei Implementierungen:
  - XML::XPath (basiert auf XML::Parser)
  - XML::LibXML

```
my @featurenodes = $doc->find('//Seq-feat')->get_nodelist;

foreach $node (@featurenodes) {
    my $gene = $node->findvalue('.//Gene-ref_locus');
    if ($gene) {
        my $ref = $node->findvalue('.//Gene-ref_syn_E');
        my $from = $node->findvalue('.//Seq-interval_from');
        my $to = $node->findvalue('.//Seq-interval_to');
        my $strand = $node->findvalue('.//Na-strand/@value');
        print "<tr>\n";
        print "  <td>$gene</td>\n";
        print "  <td>$ref</td>\n";
        print "  <td>$from - $to $strand</td>\n";
        print "</tr>\n";
    }
}
```

3. Verfahren: SAX

- Simple API for XML
- Strom von XML-Daten triggert *Events*:
  - `start_document`, `end_document`, `start_element`, `end_element`, `characters`, ...
- Trennung in XML-Parser und Event-Handler
- beliebige Kombination von Parsern und Handlern
- diverse Parser:
  - XML::Parser::PerlSAX, XML::SAX::Expat
  - XML::LibXML::SAX
  - XML::SAX::PurePerl

SAX

- Problem: Wie programmiert man den Event-Handler?
- viele Beschreibungen der API, kaum ausführliche Beispiele
- Zuordnung von Textdaten zu Elementen
- zwei Möglichkeiten:
  - endlicher Automat (Zustandsänderung bei `start_element`)
  - Stack (`characters` $\hat{=}$ push, `end_element` $\hat{=}$ pop)
- Verarbeitung eines Elements über Programm verstreut
- vereinfachter Beispielcode

```
sub start_element {
    my ($self, $element) = @_;
    if ($element->{Name} eq 'Seq-feat') {
        $tabrow = "<tr>\n";
    } elsif ($element->{Name} eq 'Gene-ref_locus') {
        $tabrow .= "  <td>";

sub end_element {
    my ($self, $element) = @_;
    if ($element->{Name} eq 'Seq-feat') {
        $tabrow .= "</tr>\n";
        print $tabrow;
    } elsif ($element->{Name} eq 'Gene-ref_locus') {
        $tabrow .= $recenttext."</td>\n";

sub characters {
    my ($self, $chars) = @_;
    $recenttext = $chars->{Data};
}
```

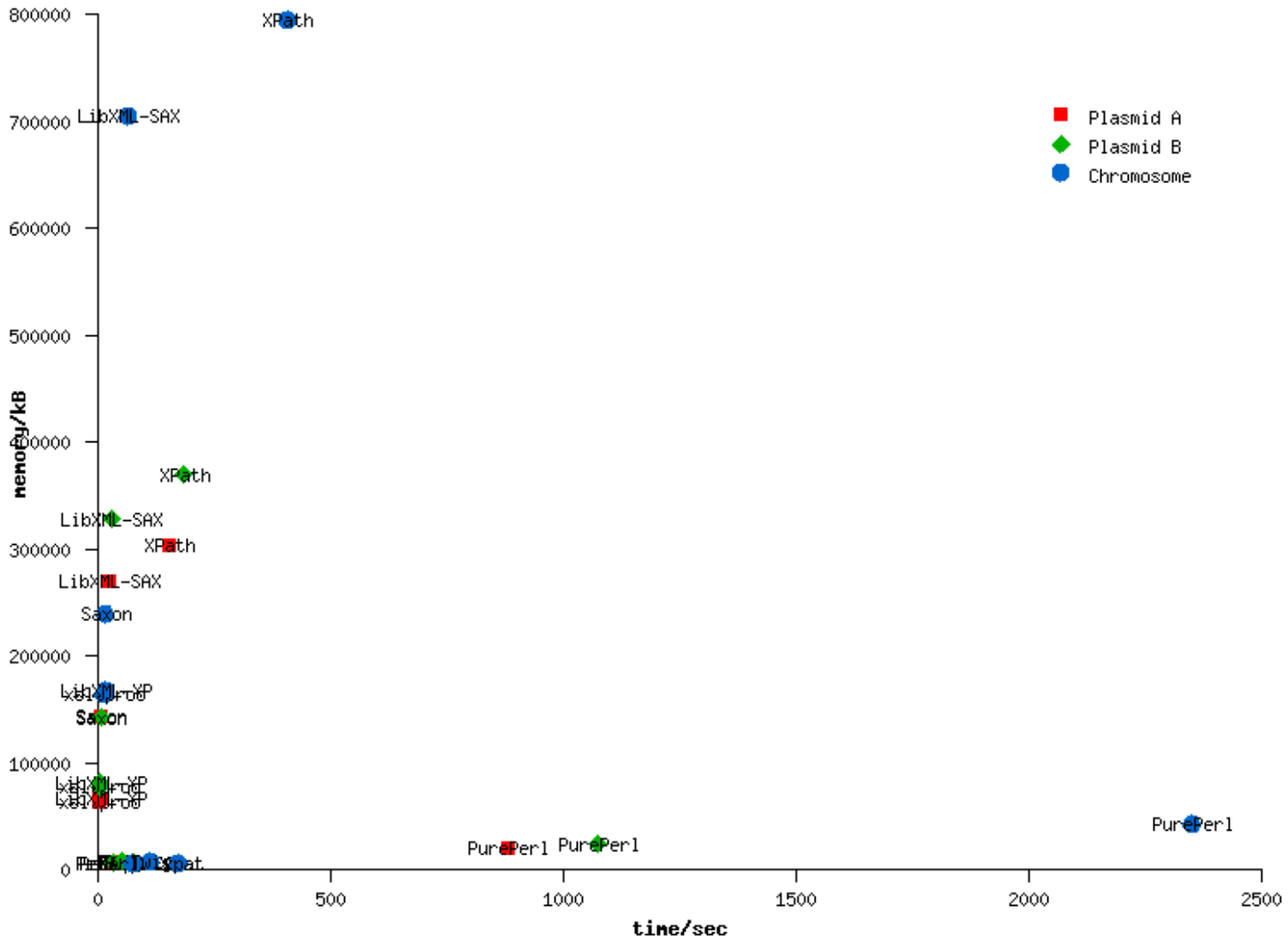
Hybride Ansätze

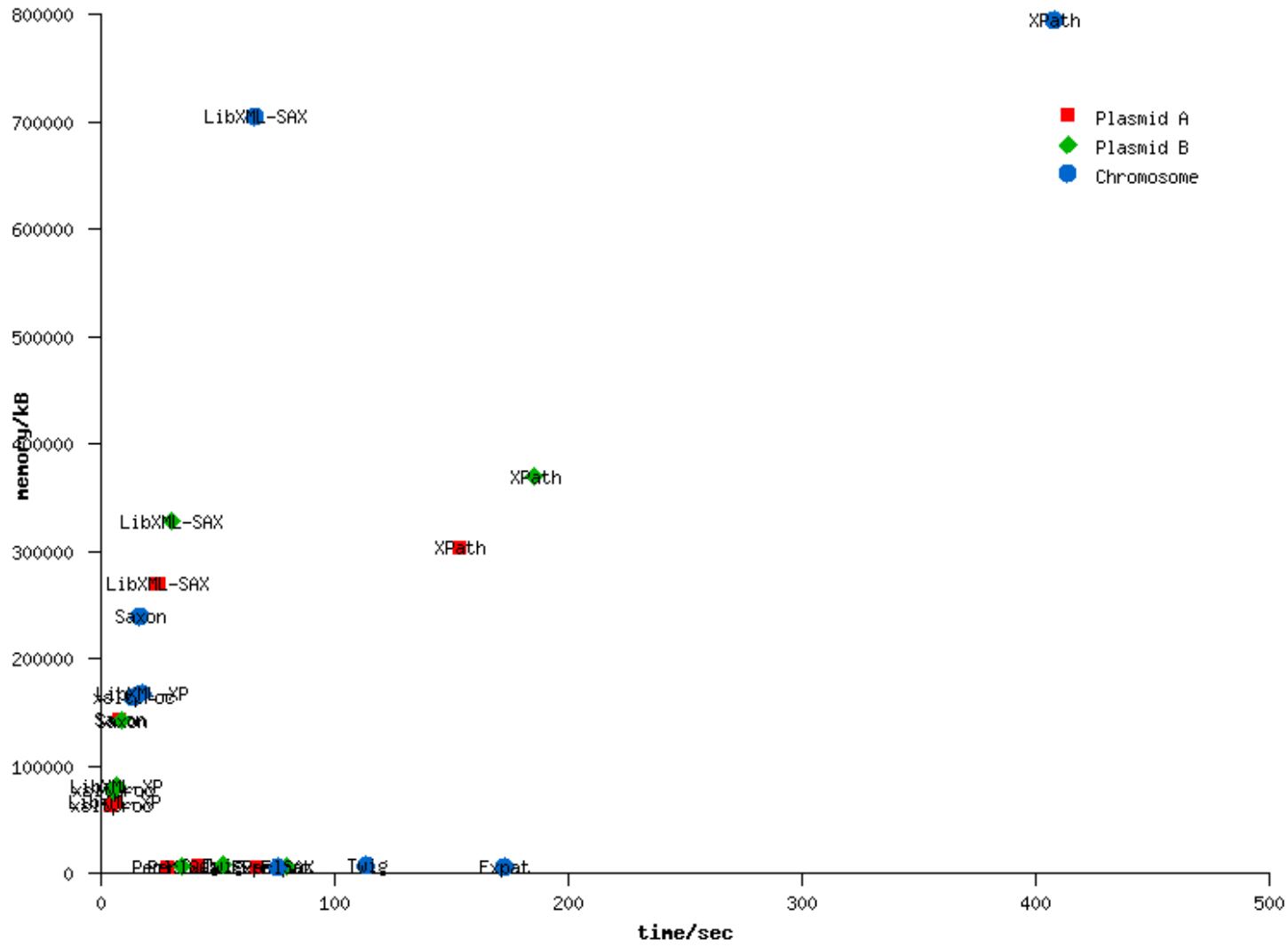
- schneide per SAX die „interessanten“ Zweige heraus
- verarbeite die Zweige mit XPath
- XML::Twig
- gute Idee, passable Resultate, miserable API

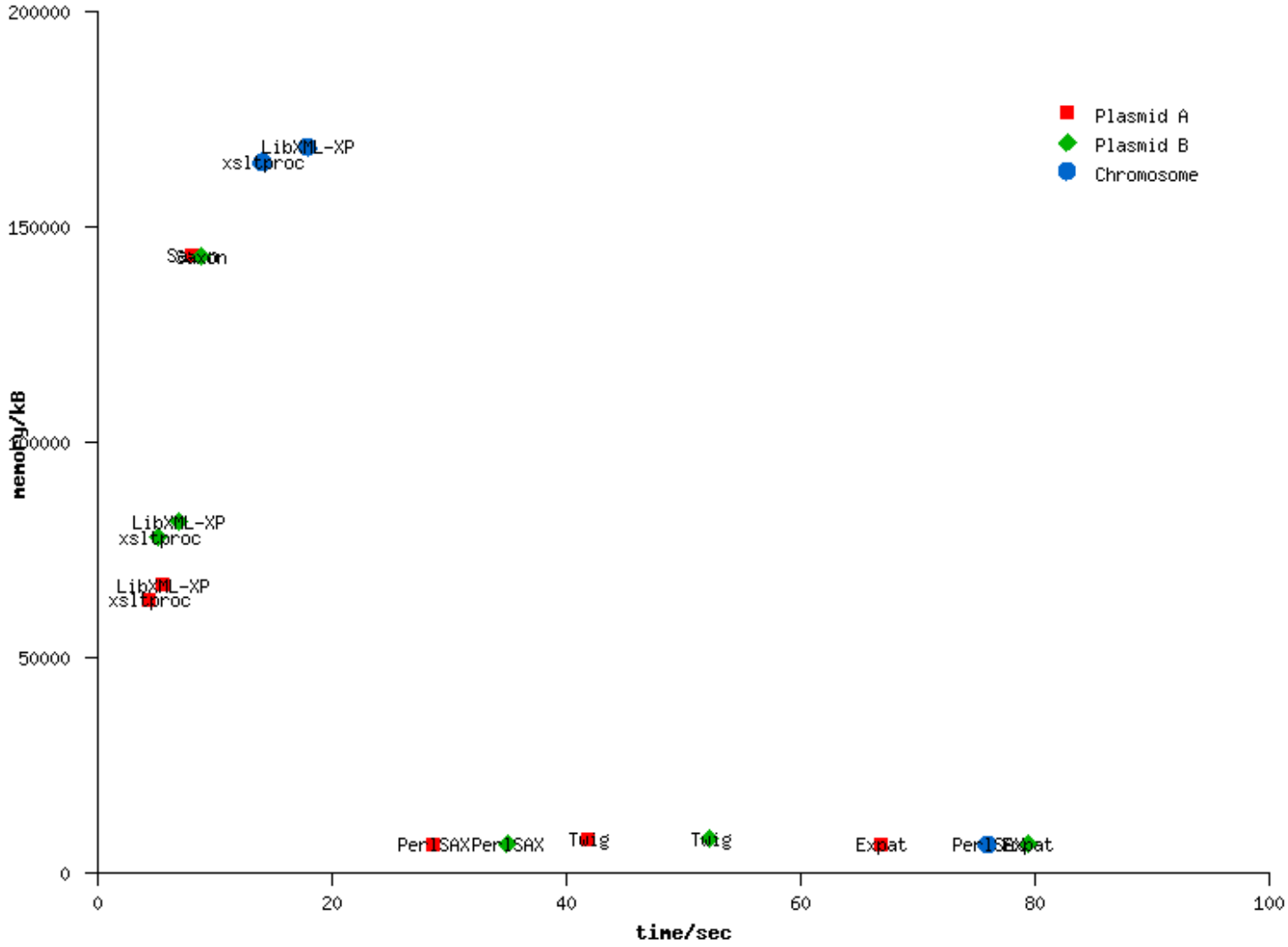
Ergebnisse

		sm_psyma		sm_psymb		sm_chr	
XPath	xsltproc rec	4.36	63528	5.28	78160	14.10	165216
	xsltproc exp	3.17	61040	3.90	73200	9.98	155584
	Saxon rec	8.08	143584	8.84	143344	16.76	239632
	Saxon exp	7.17	143576	7.86	143304	14.82	239640
	LibXML-XP	4.26	64800	5.28	76944	13.33	159400
	XPath	153.83	303944	185.68	370320	408.10	795416
SAX	PerlSAX	28.63	6792	35.00	6840	76.01	6800
	Expat	66.88	6856	79.46	6872	172.86	6848
	LibXML-SAX	24.63	270320	30.24	328416	65.94	705536
	PurePerl	882.82	21432	1076.06	24320	2351.33	43568
	Twig	41.90	8144	52.35	8192	113.68	8264

Laufzeiten in Sekunden, Speicher in kByte, Ultra Sparc III+ 900 MHz


800000
700000
600000
500000
400000
300000
200000
100000
0
memory/kB
XPath
LibXML-SAX
XPath
LibXML-SAX
XPath
LibXML-SAX
Saxon
Saxon
PurePerl
PurePerl
PurePerl
Plasmid A
Plasmid B
Chromosome
0
500
1000
1500
2000
2500
time/sec


800000
700000
600000
500000
400000
300000
200000
100000
0
memory/kB
0
100
200
300
400
500
time/sec
Plasmid A
Plasmid B
Chromosome
XPath
LibXML-SAX
XPath
LibXML-SAX
XPath
LibXML-SAX
Saxon
Saxon
Twig
Expat


200000
150000
100000
50000
0
0
20
40
60
80
100
time/sec
Plasmid A
Plasmid B
Chromosome
LibXML-XP
xsltproc
LibXML-XP
xsltproc
LibXML-XP
xsltproc
PerlSAX
PerlSAX
Twig
Twig
Expat

Fazit

- es gibt nicht **die** Lösung
- Fragen nach der Art der Daten:
  - Wie groß ist die Datenmenge?
  - Wie sind die Daten strukturiert?
- Fragen nach der Art der Verarbeitung:
  - In welcher Form benötige ich die Ergebnisse?
  - Wieviel Zeit habe ich?
  - Wieviel Speicher habe ich?
- verschiedene Verfahren ausprobieren