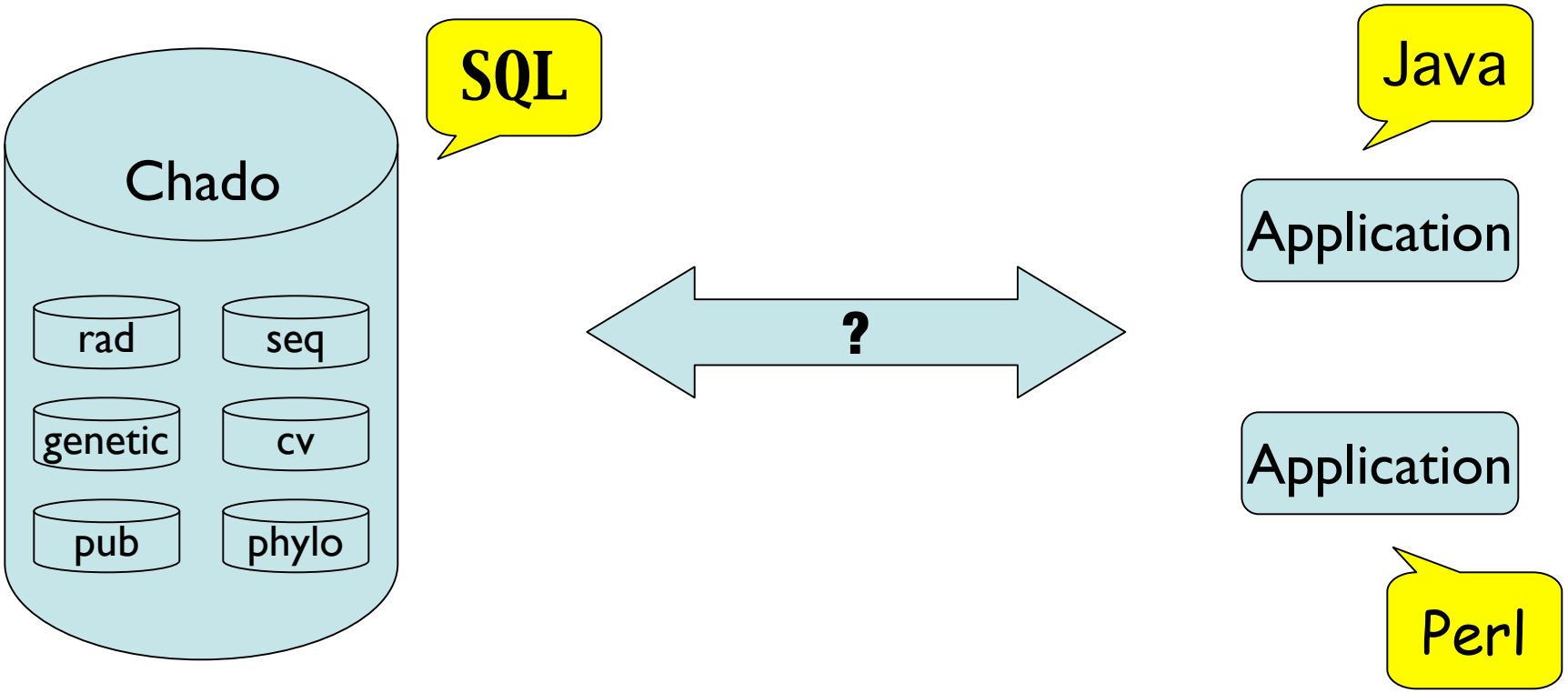
Chado and interoperability

Chris Mungall, BDGP

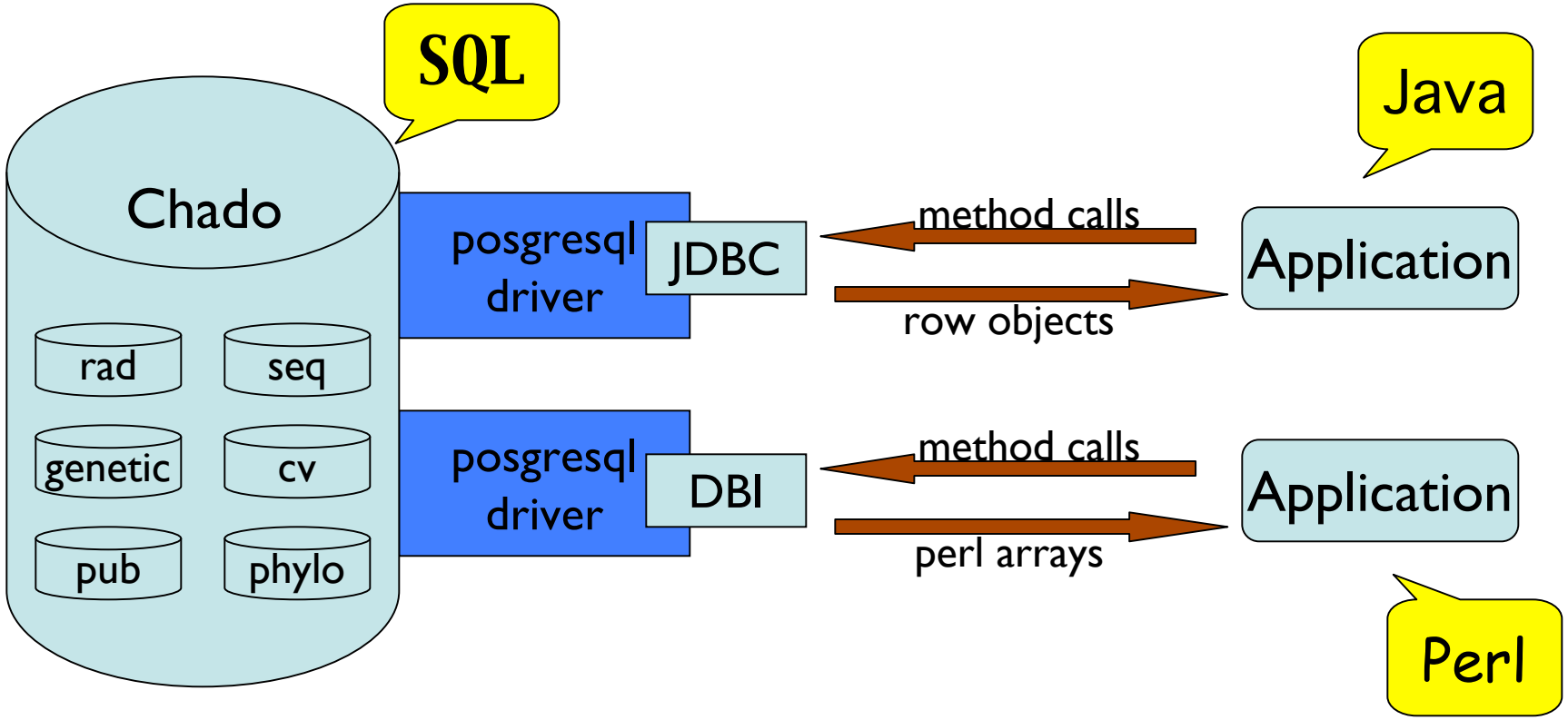
Pinglei Zhou, FlyBase-Harvard

Databases and applications



How do we get databases and applications speaking to one another?

Databases and applications



Generic database interfaces only solve part of the problem

They let us *embed* SQL inside application code

Why this alone isn't enough

- Interfacing applications to databases is a tricky business…

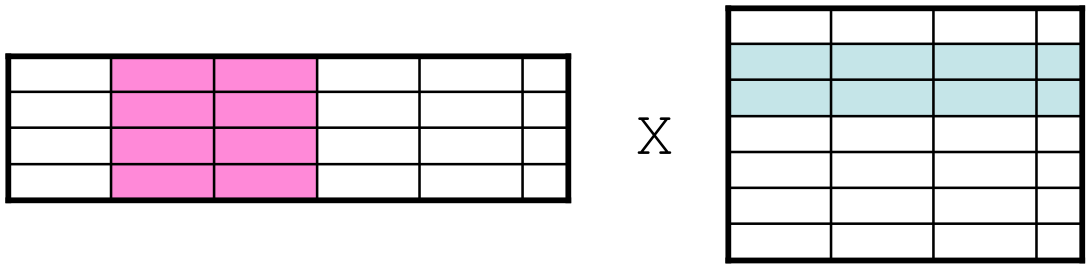
- Issue: Language mismatch
- Issue: Data structure mismatch
- Issue: Repetitive code
- Issue: No centralized domain logic

Language mismatch

```
String sql = "SELECT srcfeature_id, fmax, fmin "+
    "FROM featureloc "+
    "WHERE feature_id =" + featId;
try {
    Statement s = conn.createStatement();
    ResultSet rs = s.executeQuery(sql);
    rs.next();
    sourceFeatureId = rs.getInt("srcfeature_id");
    fmin = rs.getInt("fmin");
    fmax = rs.getInt("fmax");
 } catch (SQLException sqle) {
    System.err.println(this.getClass() +
                       ": SQLException retrieving feature loc" +
                       " for feature_id = " + featId);
    sqle.printStackTrace(System.err);
 }
```

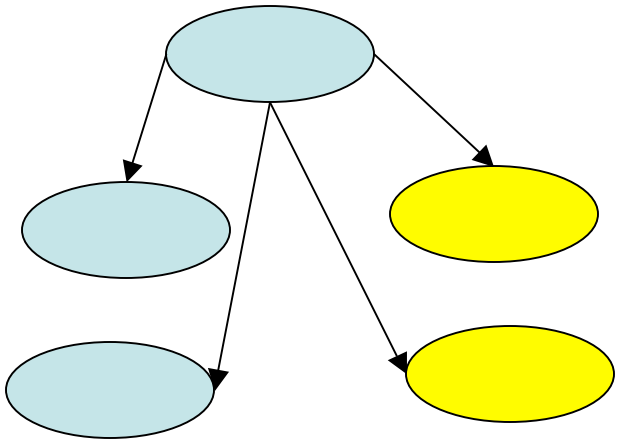
Data structure mismatch

- Different formalisms



relations

- set theoretic
- relational algebra



classes and structs

- pointers
- programs

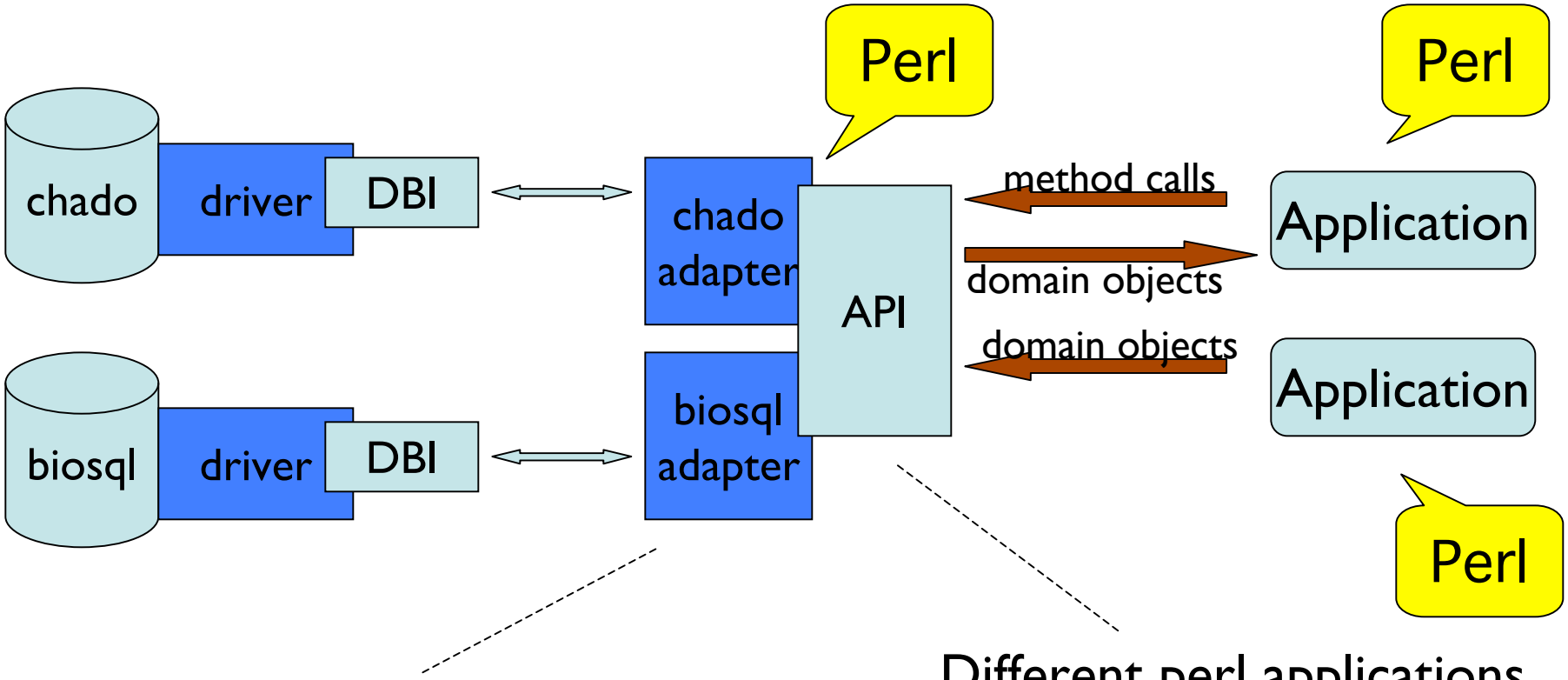
Repetitive code

- Database fetch pattern
  - construct, ask, transform, repeat, stitch
- Example: fetching gene models
  - fetch genes
  - fetch transcripts
  - fetch exons, polypeptides
  - fetch ancillary data (props, cvs, pubs, syns, etc)
- Optimisation is difficult

No centralized domain logic

- Examples of domain logic:
  - project a feature onto a virtual contig
  - revcomp or translate a sequence
  - search by ontology term
  - delete a gene model
- Domain logic should be reusable by different applications

A solution:Object Oriented APIs



Different schemas can be added by writing adapters

Different perl applications share the same API

How do OO APIs help?

- Issue: Language mismatch
  - Seperation of interface from implementation
- Issue: Data structure mismatch
  - API talks objects
  - adapters hide and deal with conversion
- Issue: Repetitive code
  - code centralized in both API and adapter
- Issue: No centralized domain logic
  - object model encapsulates domain logic
  - object model can be used independently of database

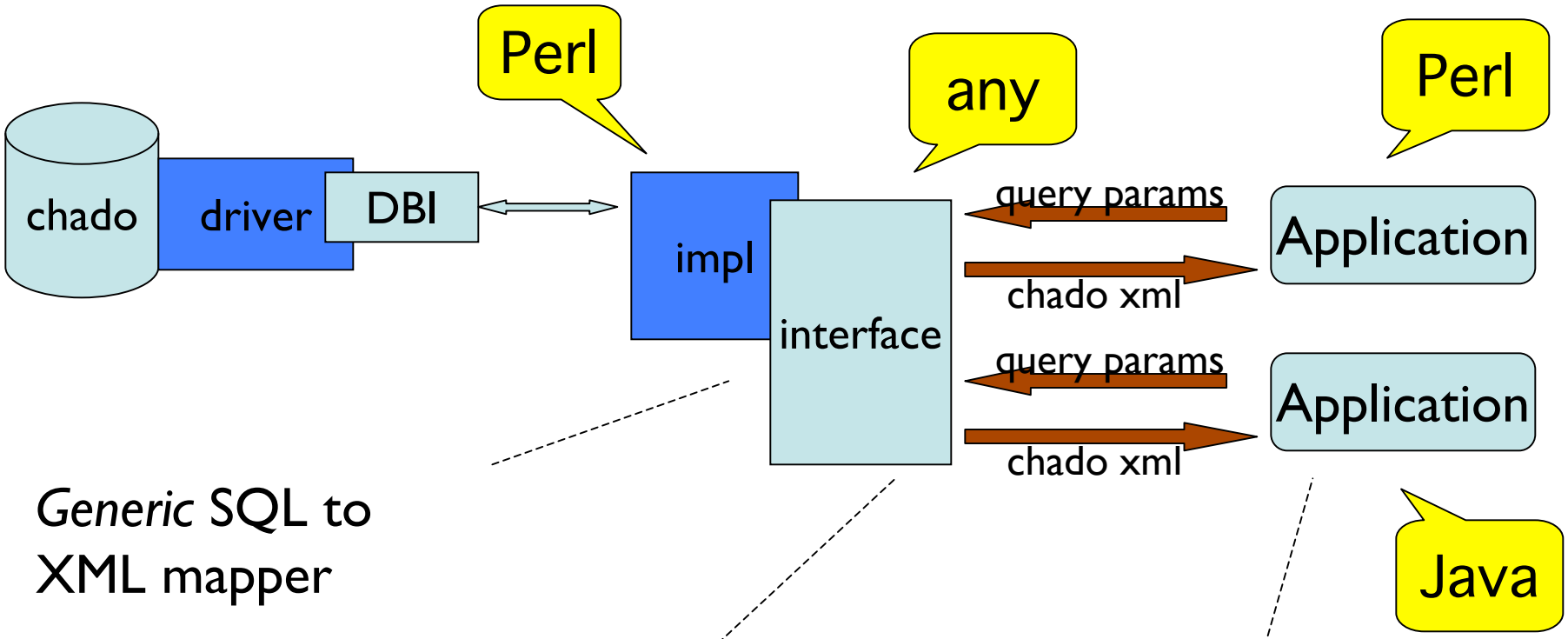
How do OO APIs hinder?

- Writing or generating adapters
  - brittle, difficult to maintain
- Restrictive
  - canned parameterized queries *vs* query language
- Application language bound
  - very difficult to use a perl API from java
- Application bound
  - sometimes generic, but often limited to one application
- Opaque domain logic

XML can help

- XML is application-language neutral
- XML can be used to specify:
  - data
  - transactions
  - queries and query constraints
- XML can be used within both application languages and specialized XML languages
  - XPath
  - XSLT
  - XQuery

XML middleware

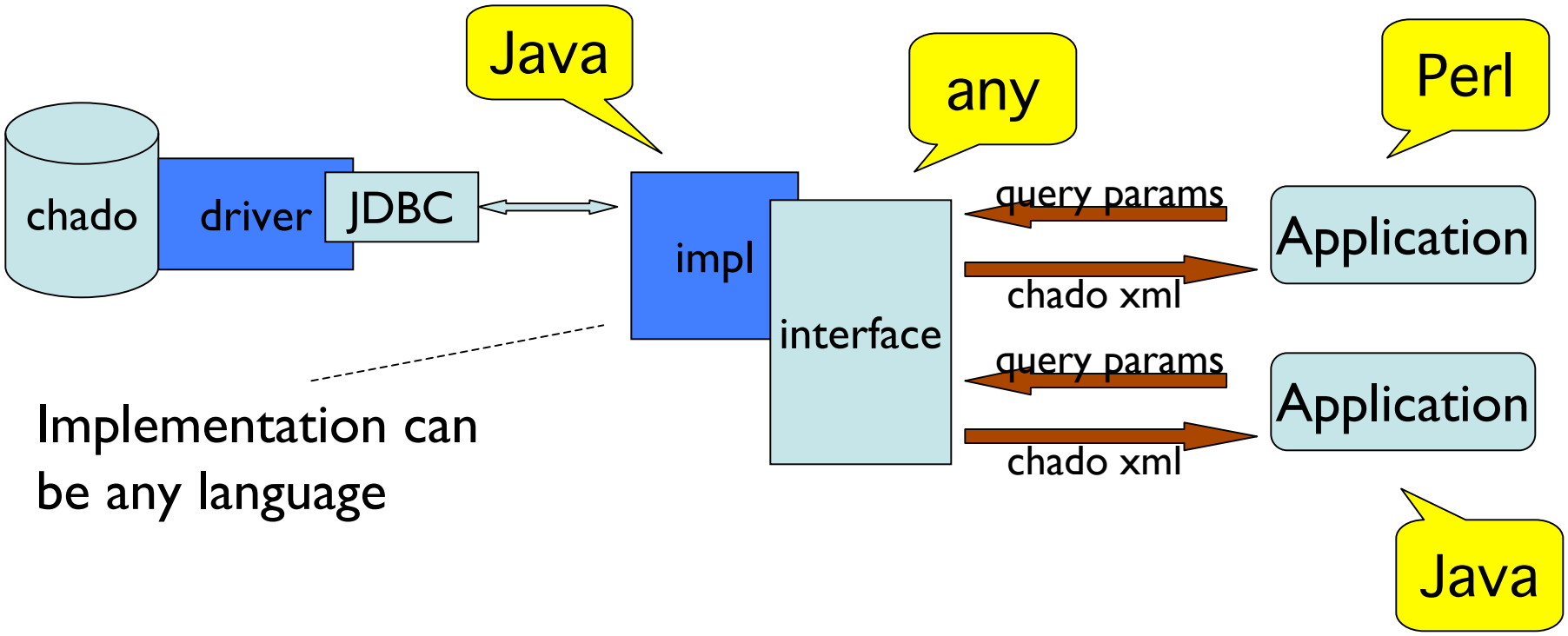


Generic SQL to XML mapper

Database XML as *lingua franca*

OO APIs can be implemented on top of XML layer

XML middleware



Mapping with XORT

- XORT is a specification of how to map XML to the relational model
  - generic: independent of chado and biology
- XML::XORT is a perl implementation of the XORT specification
  - Other implementations possible
    - DBIx::DBStag implements XORT xml->db
- Application language agnostic
  - Easily wrapped for other languages

Highlights

- Proposal: XML mapping specification for Chado
- Tools
- Real Case

XORT Mapping

- **Elements**
  - **Table**
  - **Column (except DB-specific value, e.g primary key in Chado schema -- not visible in XML)**
- **Attributes**
  - **few and generic: transaction and reference control**
- **Element nesting**
  - **column within table**
  - **joined table within table -- joining column is implicit**
  - **foreign key table within foreign key column**
- **Modules**
  - **No module distinctions in chadoXML**
- **Limitations of DTD**
  - **Cardinality, NULLness, data type**

Transactions and Operations

- **Lookup**
  - Select only
- **Insert**
  - Insert explicitly
- **Delete**
  - Unique identifier with unique key(s)
  - One record per operation
- **Update**
  - Two elements
  - Unique identifier with unique key(s)
  - One record per operation
- **Force**

  Combination of lookup, insert and update (if not lookup, then insert, else update)

Referencing Objects

- **By global accession**
  - **Format: dbname:accession[.version]**
  - **Only for dbxref, feature ?, cvterm ?**
- **By a pre-defined local id**
  - **Allows reference to objects in same file**
  - **Need not be in DB**
  - **Can be any symbol**
- **By lookup using unique key value(s)**
  - **Object can be in file or DB**
- **Implicitly, using foreign key to identify information in the related link table**

Object Reference
By Global Accession

```
<feature>
    <uniquename>CG3123</uniquename>
    <type_id>gene</type_id>
    <feature_relationship>
       <subject_id>Gadfly:CG3123-RA:1</subject_id>
       <type_id>producedby</type_id>
    <feature_relationship>
   ……

</feature>
```

Object Reference
By Local ID

```
<cv id="SO">
   <name>Sequence Ontology</name>
</cv>

<cvterm id="exon">
        <cv_id>SO</cv_id>
        <name>exon</name>
</cvterm>

<feature>
          <type_id>exon</type_id>
```

Object Reference

By key Value (s)

```
<feature>
    <type_id>
        <cvterm>
            <cv_id>
                <cv>
                  <name>Sequence Ontology</name>
                </cv>
            </cv_id>
          <name>exon</name>
       </cvterm>
    </type_id>
       ....
```

ChadoXML Example

```
<cv id="SO">
   <name>Sequence Ontology</name>
</cv>
<feature op="lookup" id="CG3312">
     <uniquename>CG3312</uniquename>
     <type_id>
          <cvterm>
               <name>gene</name>
               <cv_id>SO</cv_id>
          </cvterm>
     <type_id>
    <organism_id>
     <feature_relationship>
          <subject_id>Gadfly:CG3312-RA</subject_id>
           <type_id>producedby</type_id>
      </feature_relationship>
 </feature>
```

Schema-Driven Tools

- DTD Generator: DDL-DTD
- Validator
  - **DB Not connected**

    Syntax verification: legal XML, correct element nesting

    Some Semantic verfication: NULLness, cardinality, local ID reference
  - **DB Connected: reference validation**
- Loader: XML->DB
- Dumper:DB->XML
  - Driven by XML "dumpspec"
- XORTDiff: diff two XORT XML files

DumpSpec Driven Dumper

- **Default behavior: given an object class and ID, dump all direct values and link tables, with refs to foreign keys.**

- **Non-default behavior specified by XML dumpspecs using same DTD with a few additions:**
  - attribute dump= all | cols | select | none
  - attribute test = yes | no
  - element _sql
  - element _appdata

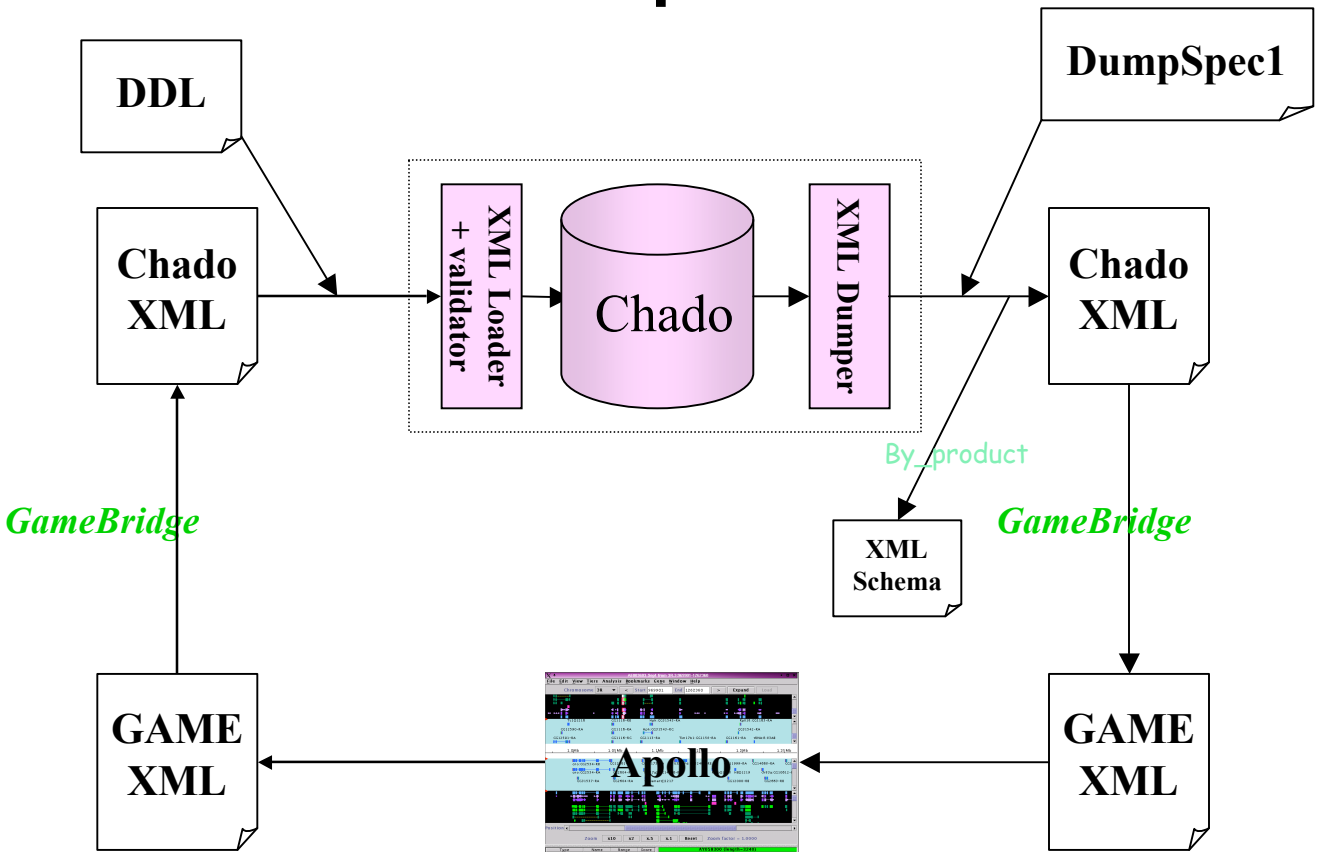
- **Workaround with views, _sql**
- **Current use cases:**
  - Dump a gene for a gene detail page
  - Dump a scaffold for Apollo

DumpSpec Sample

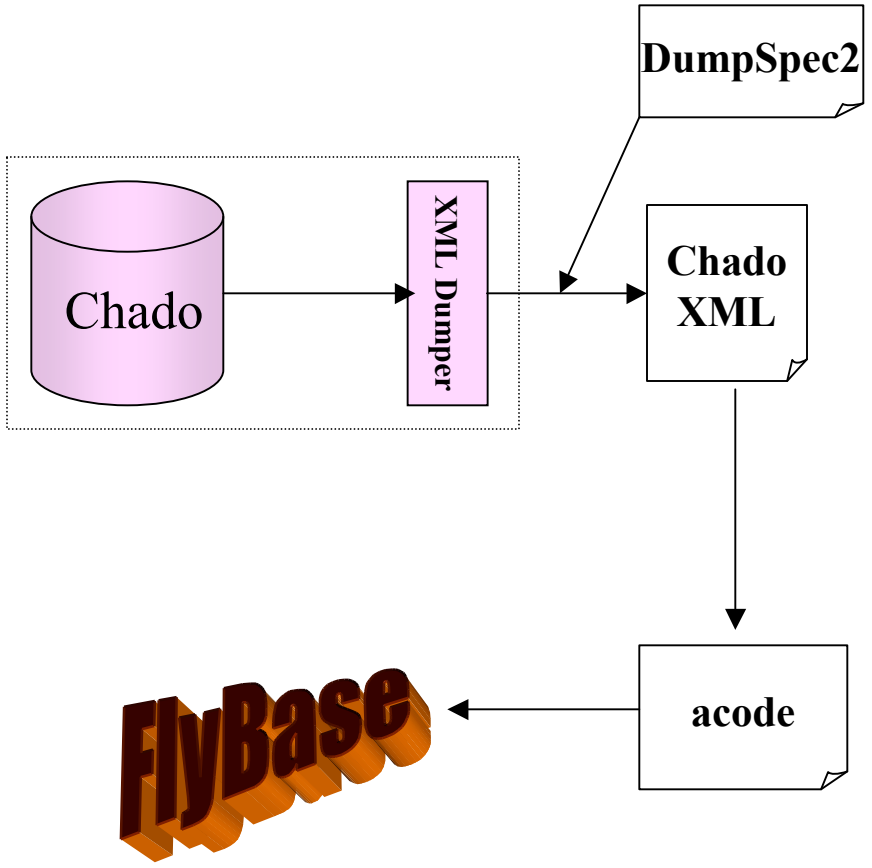
```
<feature dump="all">
    <uniquename test="yes">CG3312</uniquename>
    <!-- get all mRNAs of this gene -->
    <feature_relationship dump="all">
     <subject_id test="yes">
       <feature>
          <type_id><cvterm> <name>mRNA</name> </cvterm> </type_id>
       </feature>
     </subject_id>
     <subject_id>
       <feature dump="all">
          <!-- get all exons of those mRNAs -->
          <feature_relationship dump="all">
            <subject_id test="yes">
               <feature>
                  <type_id> ><cvterm><name>exon</name> </cvterm> </type_id>
                </feature>
            </subject_id>
            <subject_id>
               <feature dump="all“/>
             </subject_id>
         </feature_relationship>
 </feature>
```

Use Case 1

Chado <-> Apollo Interaction



Use Case 2
Gene Page Dataflow



To Do Lists

- External Object reference
- Dump with auto-generated XML Schema
- Output human-friendly

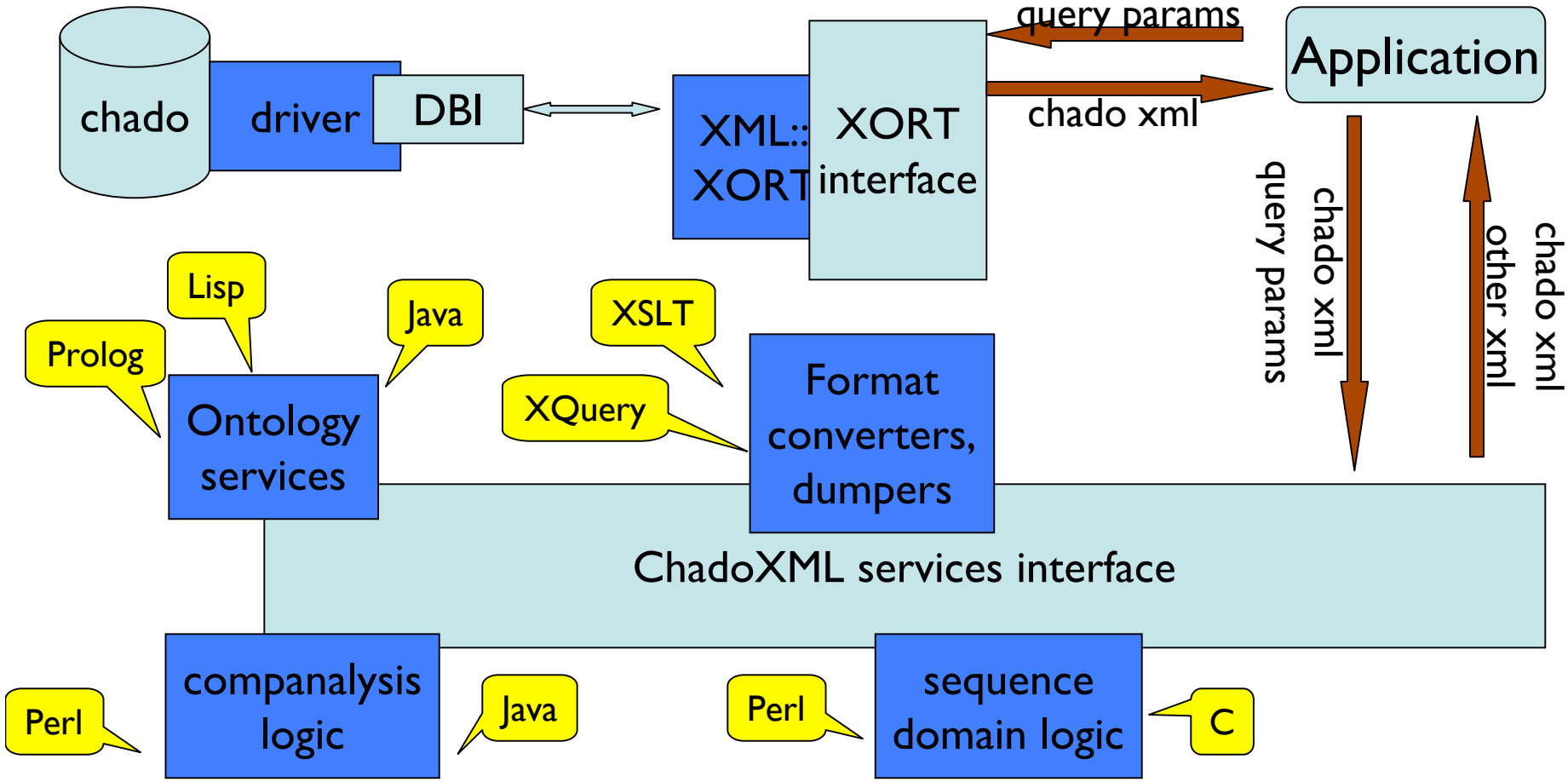
Resources

- Today's slides
- XORT package http://www.gmod.org
- Protocol draft submit to Current Protocol In Bioinformatics
- **Using *chado* to Store Genome Annotation Data**

XORT Key points

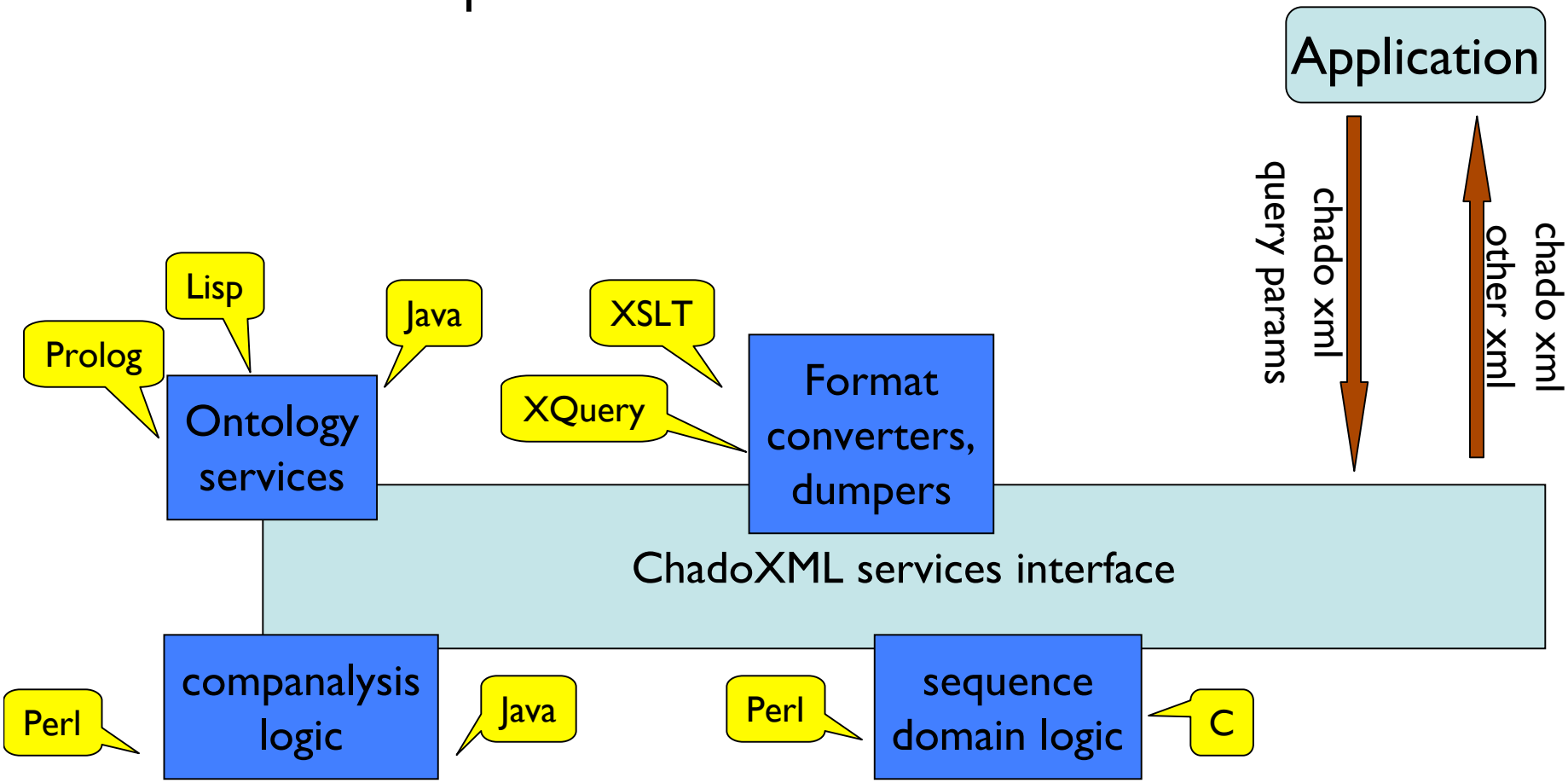
- Application language-neutral
  - reusable from within multiple languages and applications
- Where does the domain logic live?
    - Unlike objects, XML does not have 'behaviour'
  - One solution: ChadoXML Services
  - Another solution: Inside the DBMS

ChadoXML Services

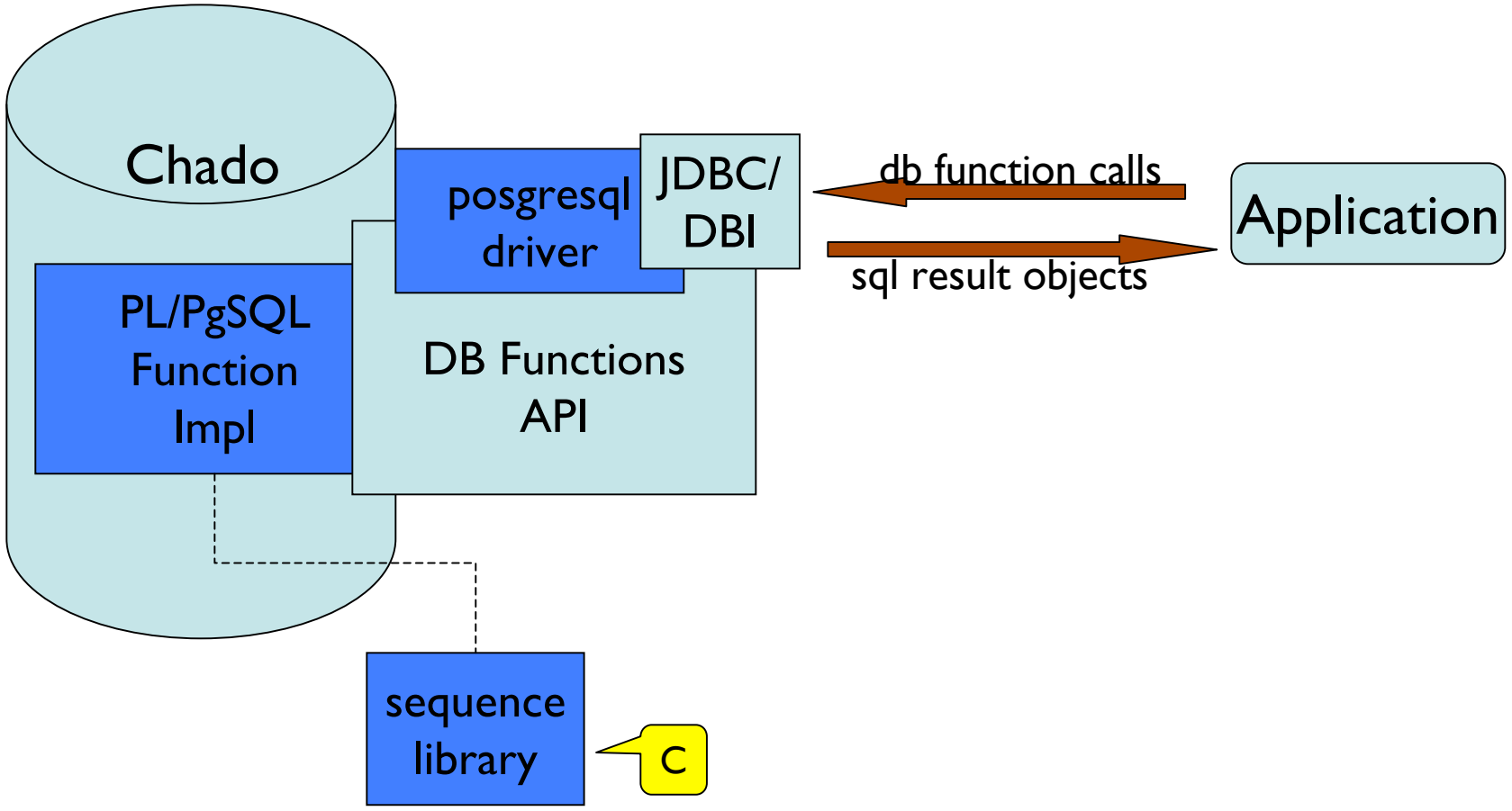


ChadoXML Services

Can be independent of DB

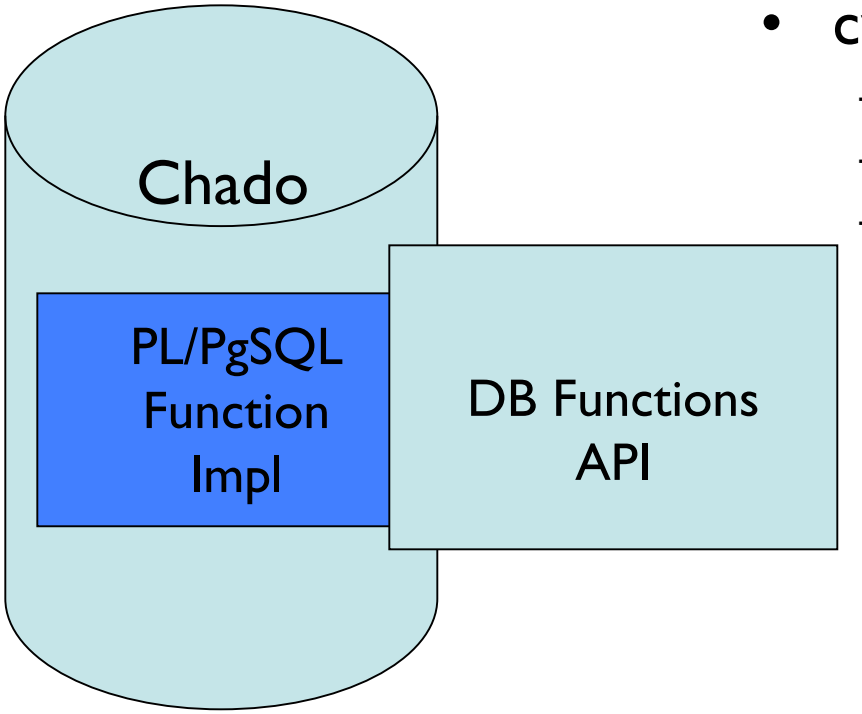


DB Functions API



Implementation *inside* database

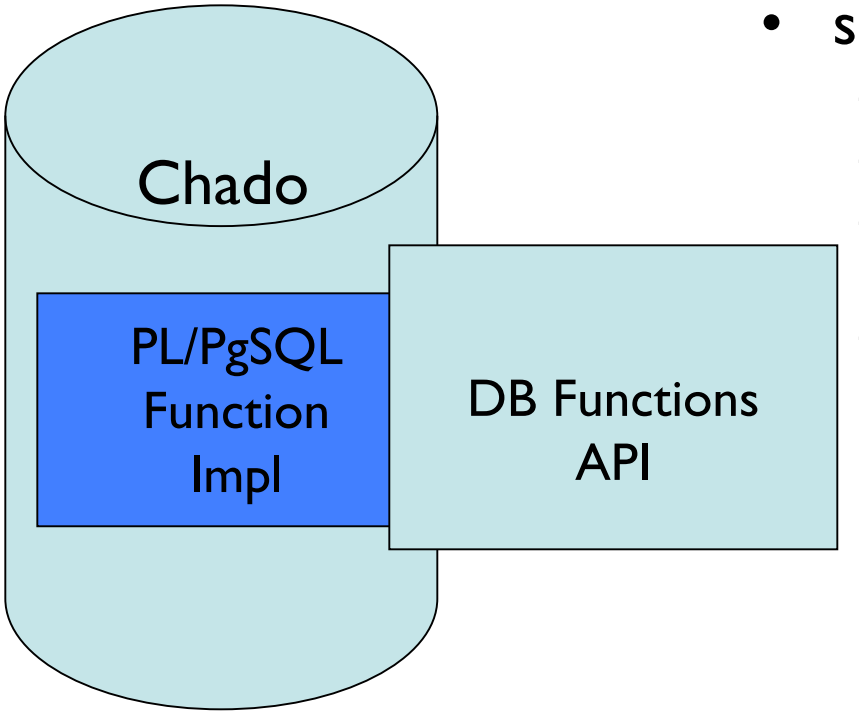
DB Functions API



- cv module
  - get_all_subject_ids(cvterm_id int);
  - get_all_object_ids(cvterm_id int);
  - fill_cvtermpath(cv_id int);

Existing functions

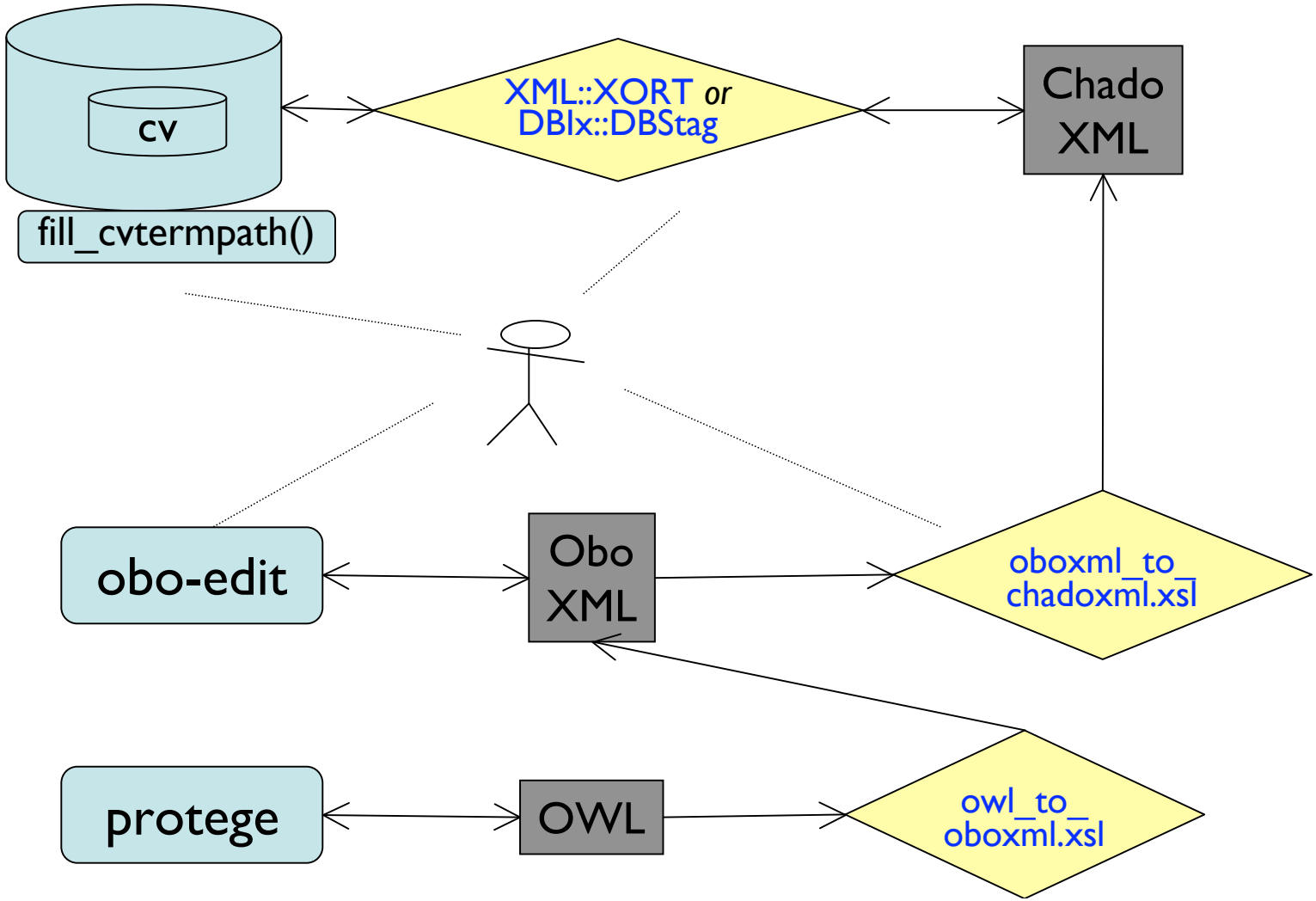
DB Functions API



- sequence module
  - get_sub_feature_ids(feature_id int)
  - featuresplice(fmin int, fmax int)
  - get_subsequence(srcfeature_id int, fmin int, fmax int, strand int)
  - next_uniquename()

Existing functions

Putting it together: storing ontologies in chado



Benefits

- Issue: Language mismatch
  - XORT dumpspecs and sql functions a more natural fit for application languages
- Issue: Data structure mismatch
  - XML maps naturally to objects and structs
- Issue: Repetitive code
  - XORT dumpspecs centralize db-fetch code
  - XORT loader centralizes db-store code
- Issue: No centralized domain logic
  - domain logic can be encoded in:
    - PostgreSQL functions and triggers
    - ChadoXML services

Other issues

- Speed?
  - chained transformations may be slower (-)
  - generic code is often slower (-)
  - single point for optimization(+)
- Verbosity
  - inevitable with a normalized database
  - reduced with XORT macros
- Portability
  - XORT highly portable (+)
  - PostgreSQL functions must be manually ported to different DBMSs (-)

Current plans

- XORT wrappers
- Improving efficiency
- Documentation
- Extend PostgreSQL function repertoire
- More ChadoXML XSLTs
- ChadoXML adapters
  - CGL
  - Apollo
  - BioPerl - Bio::{Seq,Search,Tree,..}IO::chadoxml

Conclusions

- ChadoXML
  - a common GMOD format
  - converted to other formats with XSLTs
- XORT
  - centralises database interoperation logic
- PostgreSQL functions
  - useful for certain kinds of domain logic
- Object APIs
  - still required by many applications
  - can be layered on top of XORT if so desired

Thanks to…

- Richard Bruskiewich
- Scott Cain
- Allen Day
- Karen Eilbeck
- Dave Emmert
- William Gelbart
- Mark Gibson
- Don Gilbert
- Aubrey de Grey
- Nomi Harris
- Stan Letovsky
- Suzanna Lewis
- Aaron Mackey
- Sima Misra
- Emmannel Mongin
- Simon Prochnik
- Gerald Rubin
- Susan Russo
- ShengQiang Shu
- Chris Smith
- Frank Smutniak
- Lincoln Stein
- Colin Wiel
- Mark Yandell
- Peili Zhang
- Mark Zythovicz