

# Models of Database Interactivity

Graham J.L. Kemp

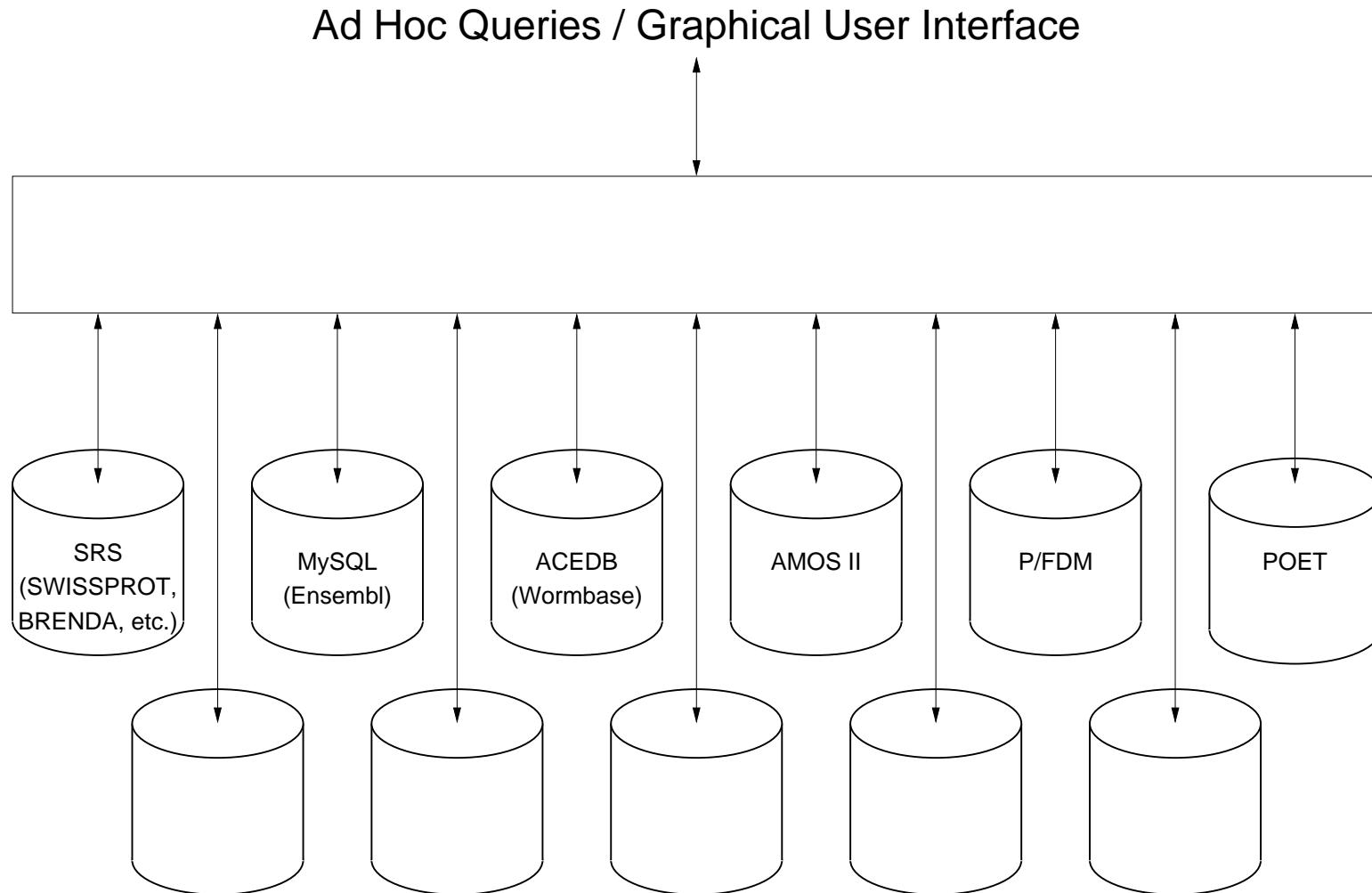
Department of Computing Science, University of Aberdeen,  
King's College, Aberdeen, Scotland, AB24 3UE

<http://www.csd.abdn.ac.uk/~gjk/>

---

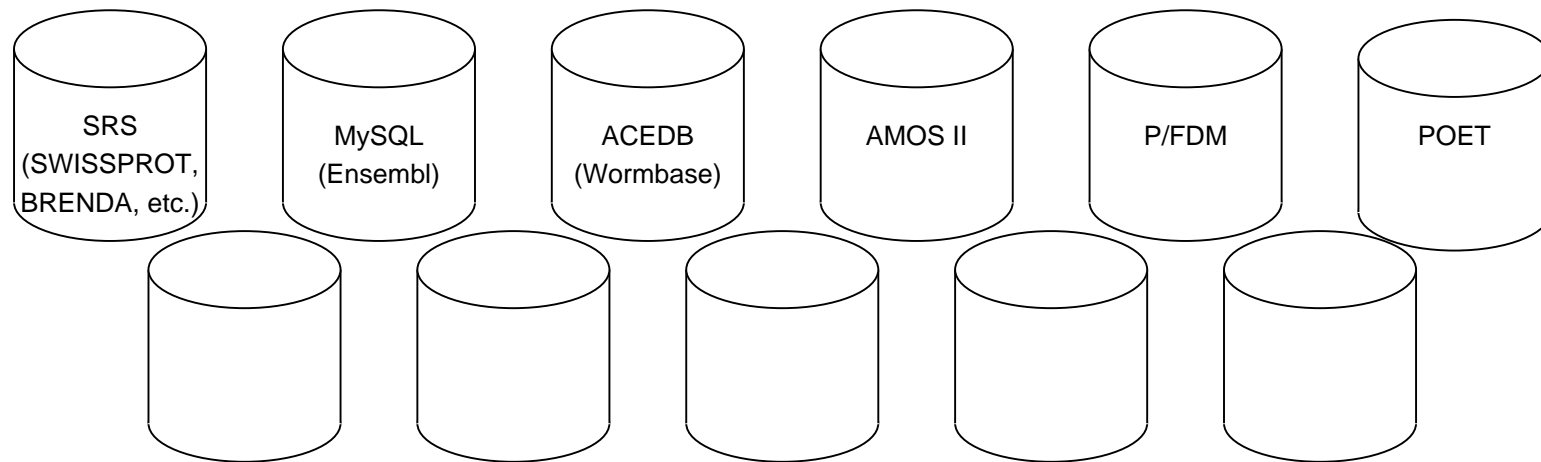
- Motivation
- Spectrum of choices from loose coupling to tight coupling
- The functional data model and P/FDM
- Implementing a database federation
- Issues

# Motivation



# Heterogeneous bioinformatics resources

	Management	Examples
Flat files	UNIX (“grep”, Perl, etc.) SRS Word processor	PDB, SWISS-PROT
Relational	Sybase, Oracle, MySQL	Ensembl, MSD
Object-based	P/FDM, AceDB, OPM, VODAK, POET	C.elegans genomic data, antibody database (P/FDM)



# Tight and Loose Coupling

**Tightly Coupled:**

single organizational entity overseeing information resources relevant to genome research

- 
- 
- 

adoption of common DBMSs at participating sites

shared data model across participating sites

common semantics for data publishing

**Loosely Coupled:**

common syntax for data publishing



R. J. Robbins, 1995

"Bioinformatics: Essential Infrastructure for Global Biology"

# Limitations of Web access to data

- Approximate search
- Lack of type checking
- Page-at-a-time, not set-at-a-time
- **Programs can't click!**

portals for people vs. portals for programs

## Interested in XML?

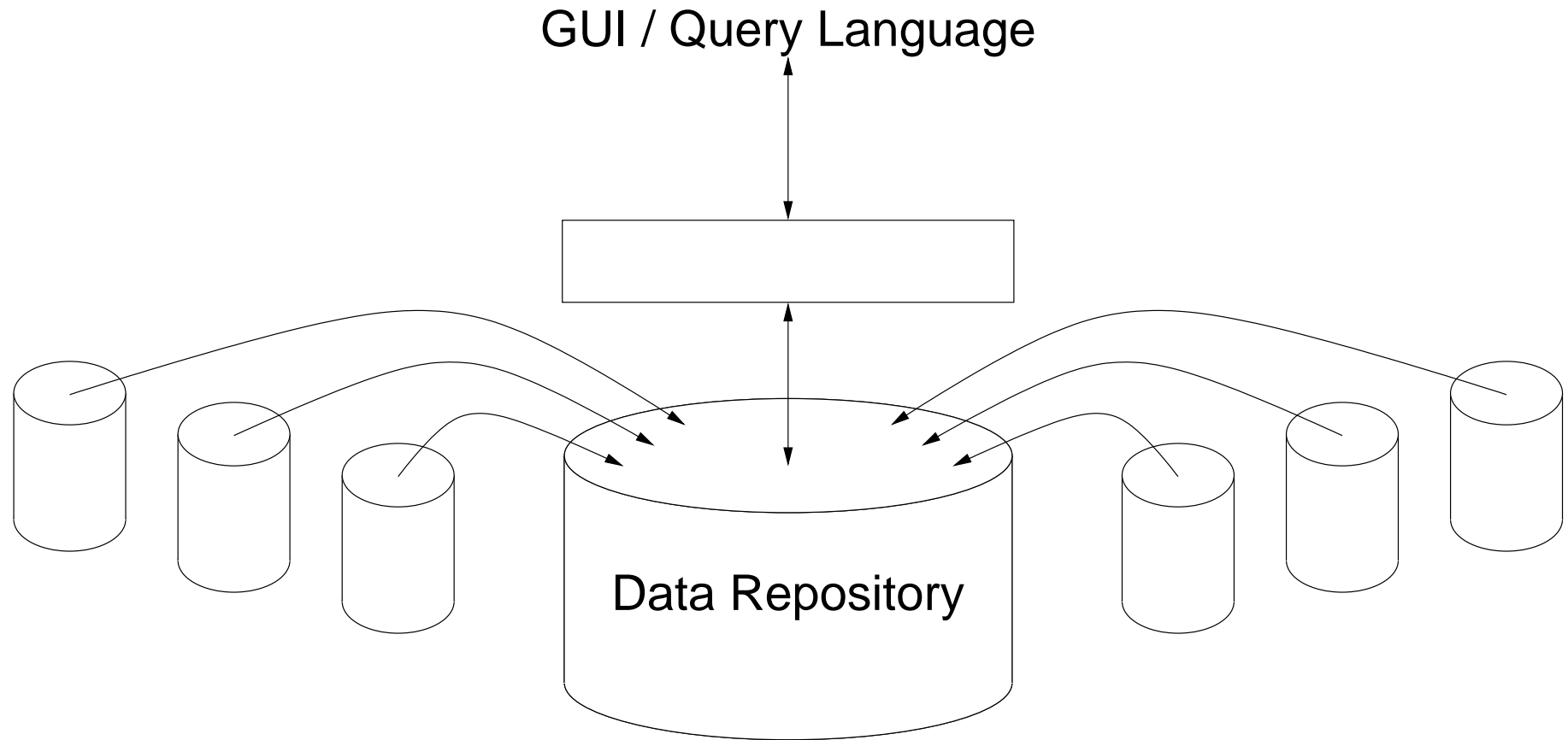
“Many companies report a strong interest in XML. XML however, is so flexible that this is similar to expressing a strong interest in ASCII characters.”

From BizTalk Framework Overview

<http://www.oasis-open.org/cover/BiztalkFrameworkOverviewFinal.html>

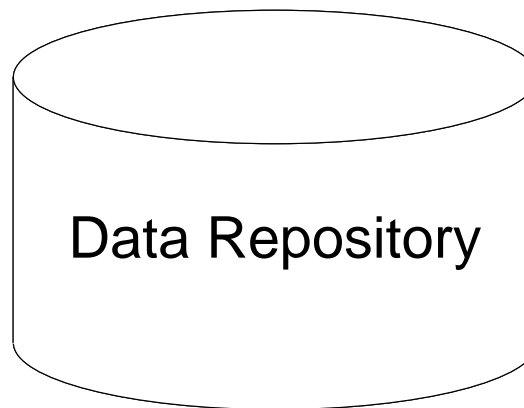
# Data repository

Physically load all data into one database.

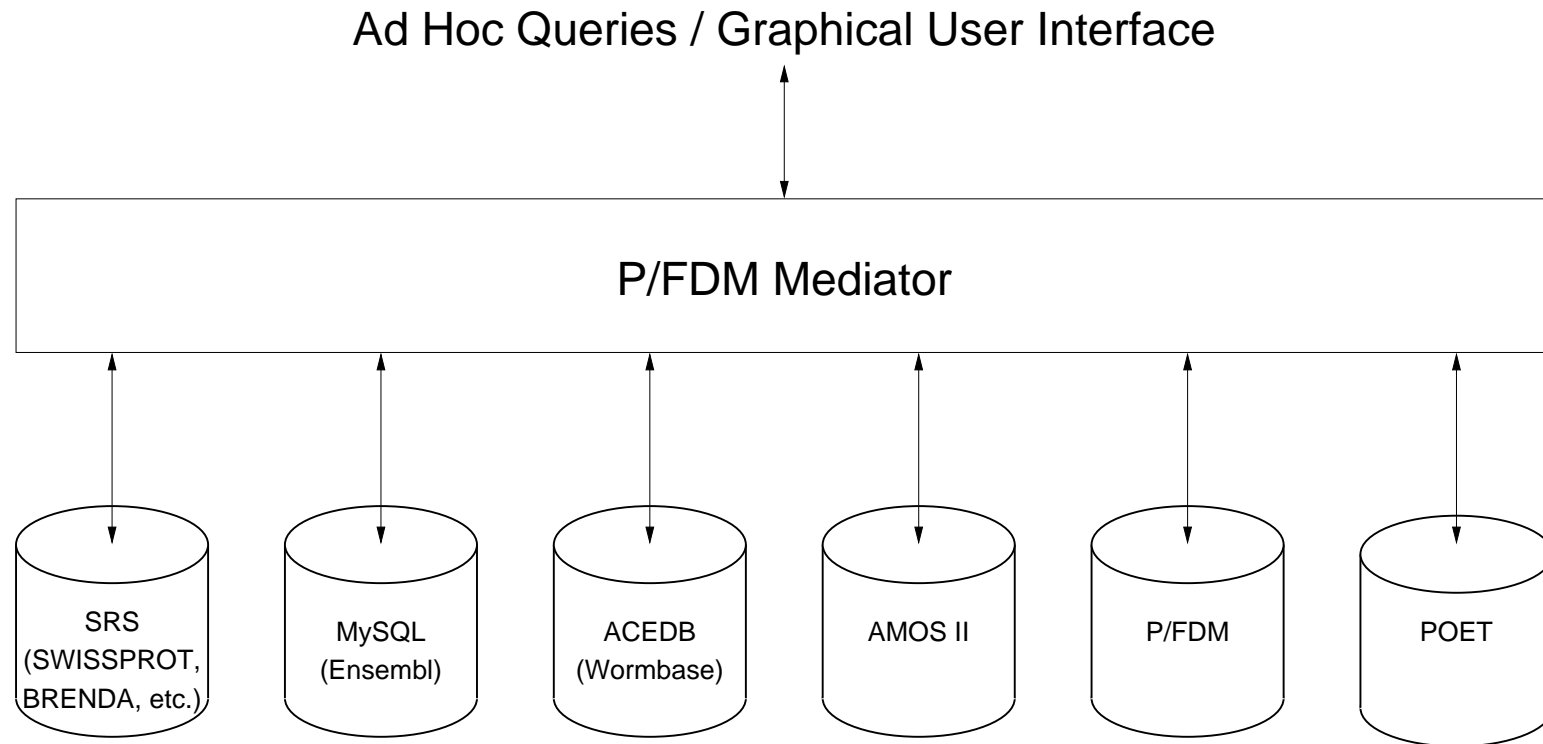


# Problems with a data repository

- space
  - too much data;
- effort
  - reformatting and “cleaning”;
- updates
  - want access to most recent data;
- advantages of heterogeneous systems are lost
  - customised interfaces and search engines.



# Using P/FDM as a “mediator”



- determine what DBs are relevant to answering the query
- translate query into language(s) of the underlying DBs
- combine results and present these to the user

# P/FDM

Object-oriented database implemented mainly in Prolog

Based on the Functional Data Model (FDM)

- entities (classes; hierarchies)
- functions
  - attributes and relationships
  - stored and derived values

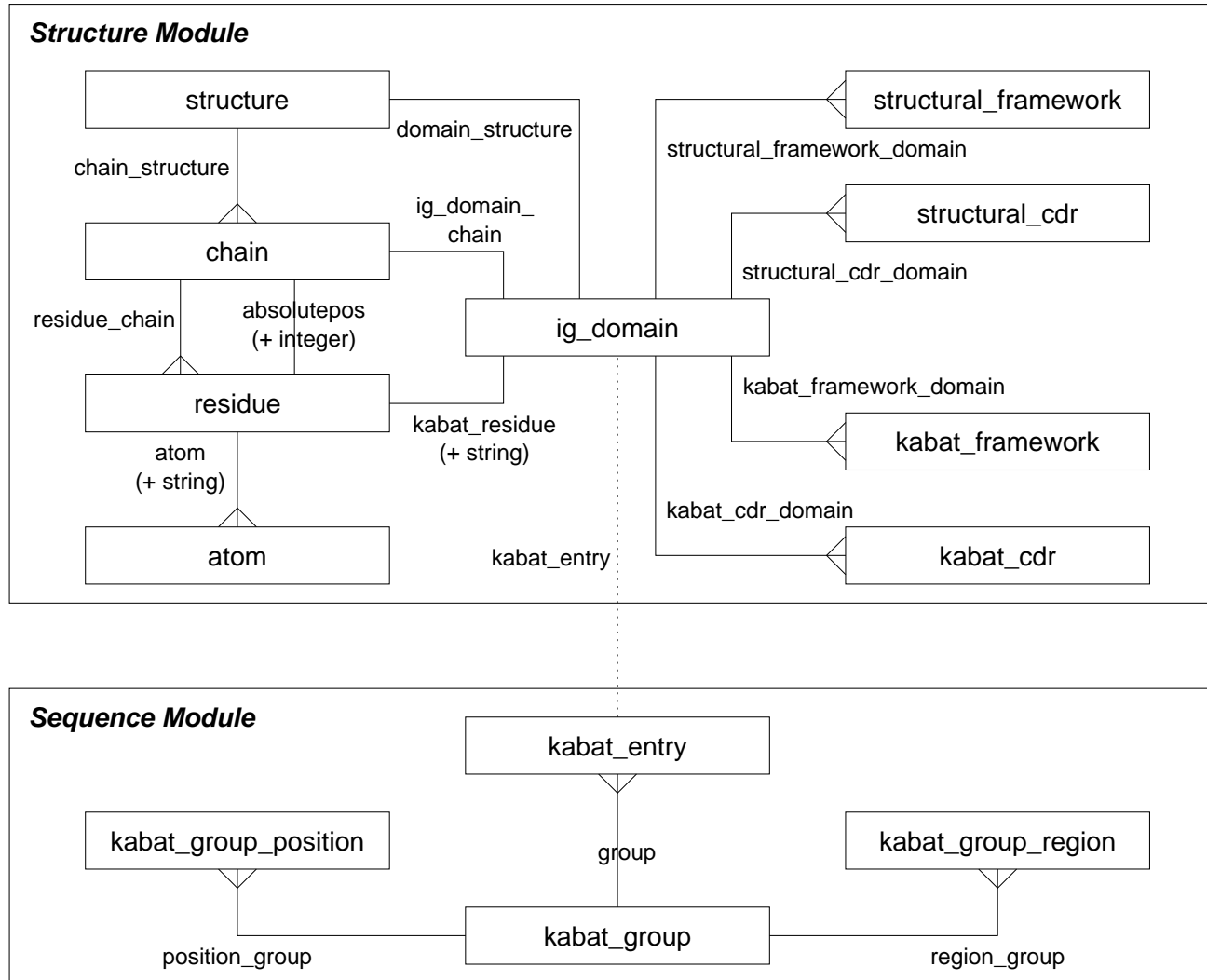
Daplex

- data definition language (DDL)
- data manipulation language (DML)

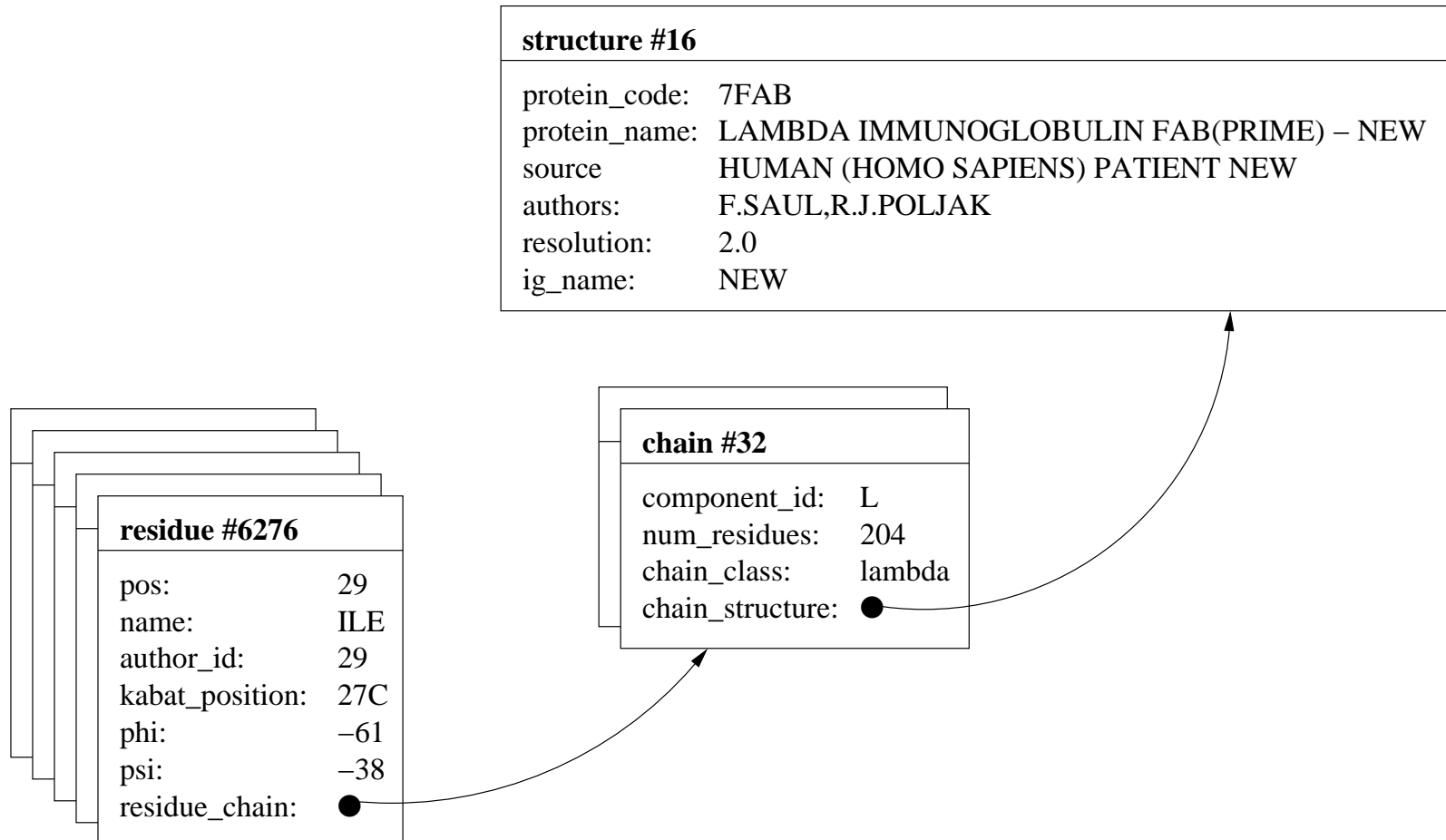
List comprehensions — Intermediate code (ICode)

Keys

# Antibody database schema diagram



# Object Instances



# Declarative query languages

— say **WHAT** you want, not **HOW** to get it

SQL:           SELECT protein\_name  
                  FROM protein  
                  WHERE resolution < 3.0 AND pdb\_code = "7FAB"

DAPLEX:       for each p in protein  
                  such that resolution(p) < 3.0 and pdb\_code = "7FAB"  
                  print(protein\_name(p))

# Keys

For each object class proposed, a set of attributes and relationships are selected whose values will be unique for each instance of the class. Together, these selected properties form a **key** for a class.

- enable users to access specific objects;
- make loading and exchanging data possible;
- enable other data values to be associated with the right object instance.

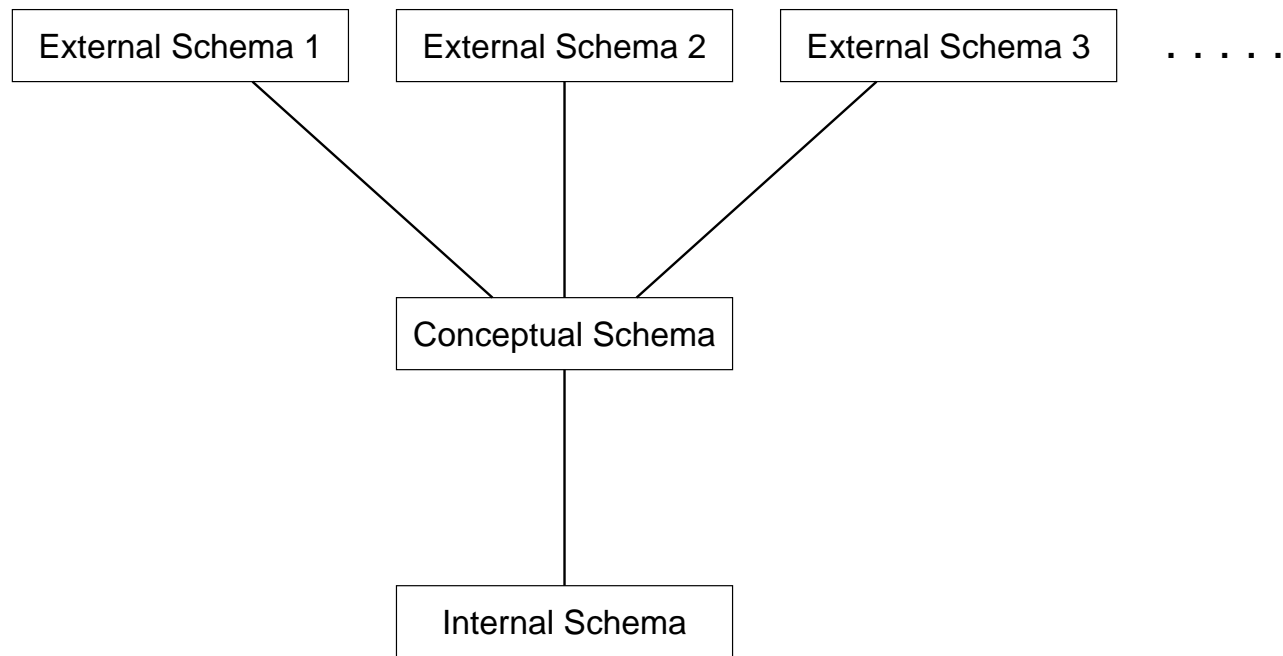
# Schemas

Named framework for persistent data.

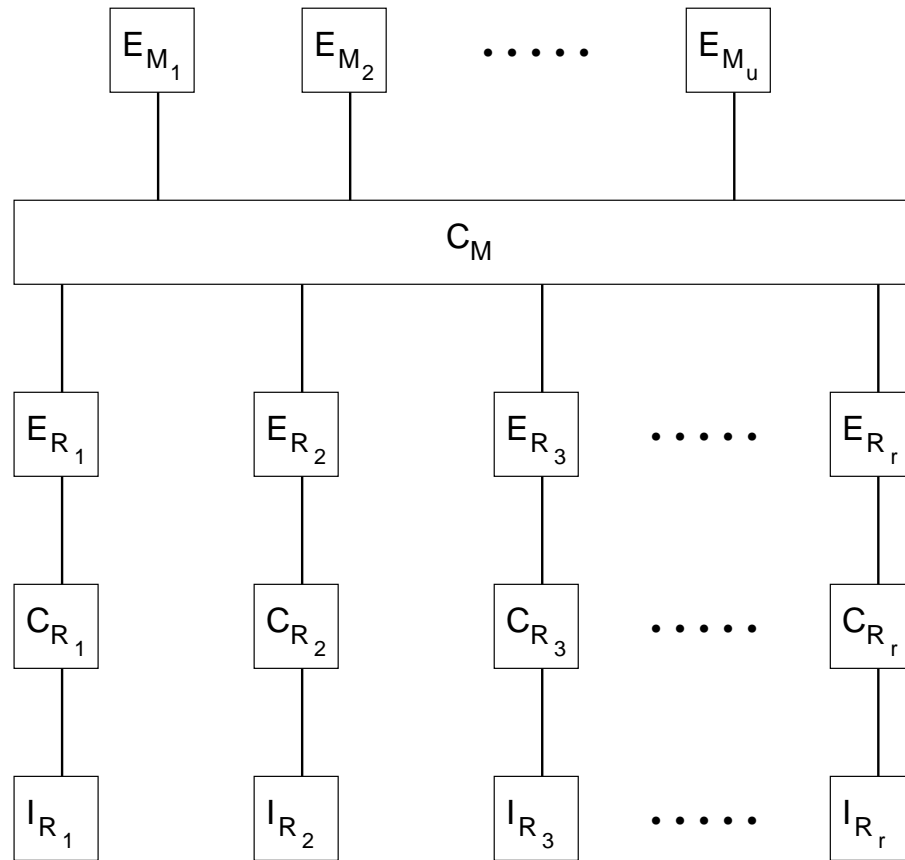
- identify and name objects, attributes and relationships that are to be stored.
- the names selected provide a common vocabulary for referring to data items.
- define constraints on attributes and relationships; these will be used to perform semantic checks on data.

# Three-level schema

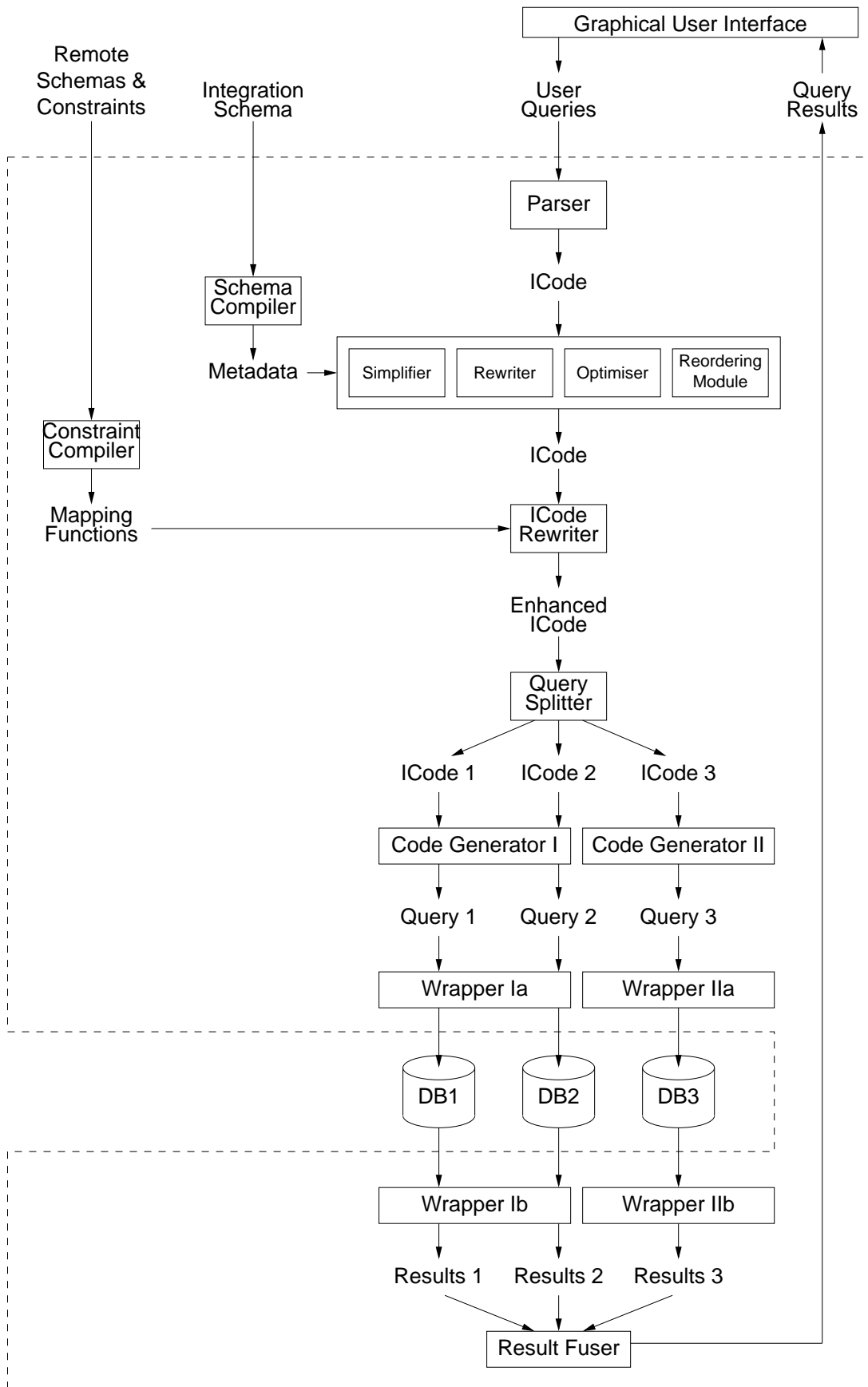
Proposed by ANSI-SPARC standards working party (1975).



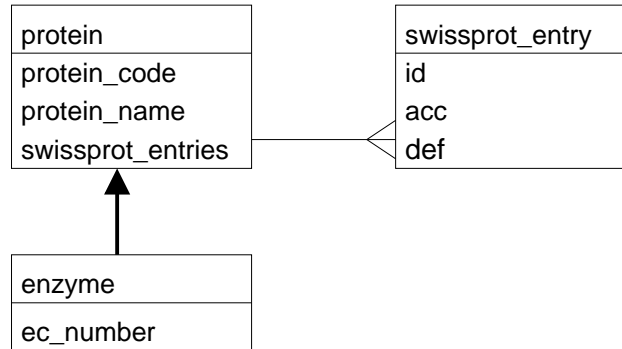
# Schemas in a database federation



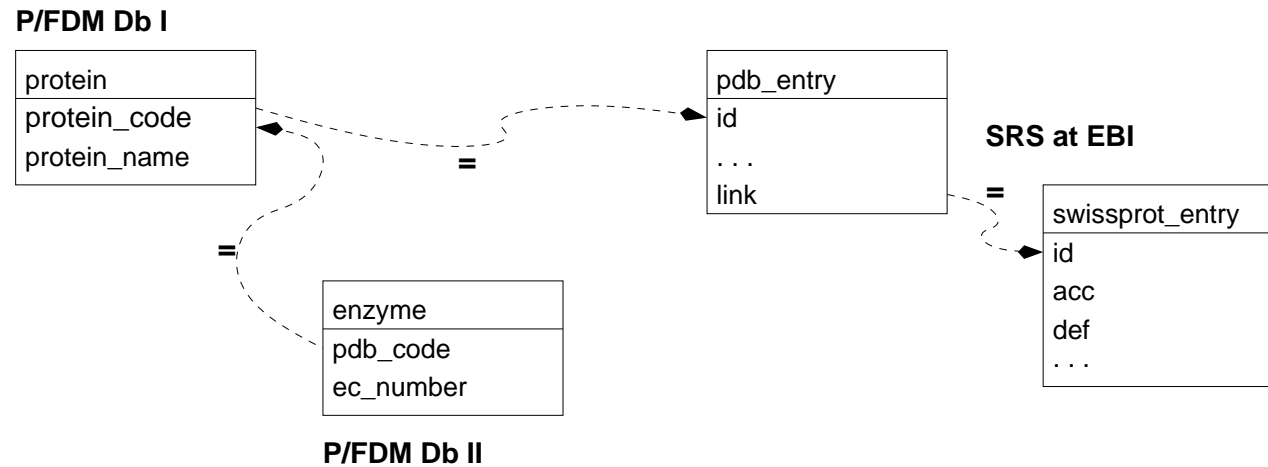
## Mediator architecture



# Integration schema and distributed databases



(a) Integration Schema



(b) Distributed Databases

# Integration schema — Daplex

```
declare swissprot_entry ->> entity
declare id(swissprot_entry) -> string
declare acc(swissprot_entry) -> string
declare dat(swissprot_entry) ->> string
declare def(swissprot_entry) -> string
declare cc(swissprot_entry) ->> string
key_of swissprot_entry is id
```

```
declare protein ->> entity
declare protein_code(protein) -> string
declare protein_name(protein) -> string
declare swissprot_entries(protein) ->> swissprot_entry
key_of protein is protein_code
```

```
declare enzyme ->> protein
declare ec_number(enzyme) -> string;
```

## Query expressed against integration schema

Daplex query:

```
for each e in enzyme such that ec_number(e) = "1.1.1.1"  
for each s in swissprot_entries(e)  
print(protein_name(e), def(s), acc(s));
```

List comprehension (represented internally as "lCode"):

```
[ V6, V4, V3 | V1 <- enzyme;  
              V2 <- swissprot_entries(V1);  
              V3 = acc(V2); V4 = def(V2);  
              V5 = ec_number(V1); V5 = "1.1.1.1";  
              V6 = protein_name(V1) ]
```

# Initial ICode representation of program

```
[  
  generate(enzyme, var(uevar1)),  
  generate(swissprot_entries, [enzyme], [var(uevar1)],  
          swissprot_entry, var(uevar2)),  
  restrict(acc, [swissprot_entry], [var(uevar2)], var(evar3)),  
  restrict(def, [swissprot_entry], [var(uevar2)], var(evar4)),  
  restrict(ec_number, [enzyme], [var(uevar1)], var(evar5)),  
  expression(var(evar5), [], expr(=, var(evar5), 1.1.1.1)),  
  restrict(protein_name, [enzyme], [var(uevar1)], var(evar6))  
]
```

or, pretty-printed:

```
[ V6, V4, V3 | V1 <- enzyme;  
              V2 <- swissprot_entries(V1);  
              V3 = acc(V2); V4 = def(V2);  
              V5 = ec_number(V1); V5 = "1.1.1.1";  
              V6 = protein_name(V1) ]
```

# Mapping functions

Mapping functions, like the one below, are created by compiling declarative constraints, and do **NOT** have to be written by hand!

This is the internal "ICode" representation of the mapping function that relates proteins in the integration schema to SWISS-PROT entries held at the EBI.

```
foreign( swissprot_entries, [protein], srs_sprot, entity, KeyICode, ebi_db ) :-
    KeyICode = (V1,V2,[V3,V4,V5,V6],
    [
        restrict(ebi_db:id,[ebi_db:srs_sprot],[V2],V6),
        restrict_subquery(some(V5),
            [ generate(ebi_db:pdb_entry,V3),
              restrict(ebi_db:id,[ebi_db:pdb_entry],[V3],V4),
              restrict(protein_code,[protein],V1,V4),
              restrict(ebi_db:link,[ebi_db:pdb_entry],[V3],V5) ],
            [ expression([], [V6,V5], expr(=,V6,V5)) ]
        )
    ] )
```

# ICode chunks

From **P/FDM Db II**, fetch:

```
[ V1 | V2 <- enzyme ;  
      V3 = ec_number(V2); V1 = pdb_code(V2);  
      V3 = "1.1.1.1" ]
```

From **P/FDM Db I**, fetch:

```
[ V6 | V4 <- protein;  
      V5 = protein_code(p); V6 = protein_name(V4); V1 = V5 ]
```

From **SRS at EBI**, fetch:

```
[ V7, V8 | V9 <- pdb_entry;  
          V10 = id(V9); V1 = V10;  
          V11 <- link(V9);  
          V12 <- swissprot_entry;  
          V13 = id(V12); V7 = def(V12); V8 = acc(V12);  
          V13 in V11 ]
```

# Execution plan

```
(  findall([A], A='1.1.1.1', [B|C]),
  restructure_univq(1, [B|C], D),
  member(E, D),
  fetch(sicstus_linda(kea,3032),
        (getentity(enzyme,F),
         getfnval(pdb_code,[F],G),
         getfnval(ec_number,[F],E)),
        [G], H),
  restructure_table([1], [], 1, H, I),
  ky_member(J, [J], [J], H, I, G),
  fetch(srs_ebi_sprot,
        sesq(['-lo+([pdb-id:',G,']>SWISSPROT)'],
             ['-e+[swissprot-id:',K,']'],
             K,
             ['AC','DE'],[L,M],G),
        [G,M,L], N),
  restructure_table([1], [2,3], 3, N, O),
  fetch(sicstus_linda(kea,3031),
        (getentity(rem_protein,([P],[],Q)),
         getfnval(rem_protein_code,([P],[],Q)),R),
         getfnval(rem_protein_name,([P],[],Q)),_),
        [R], S),
  restructure_table([1], [], 1, S, Q),
  member([P]-_, Q),
  fn_member(1, [J], [J,T,U], N, O, M),
  fn_member(2, [J], [J,T,U], N, O, L),
  write_list_pp([([P],[],Q),M,L]),
  fail
;  true
).
```

## Accessing local data and SRS

### Query:

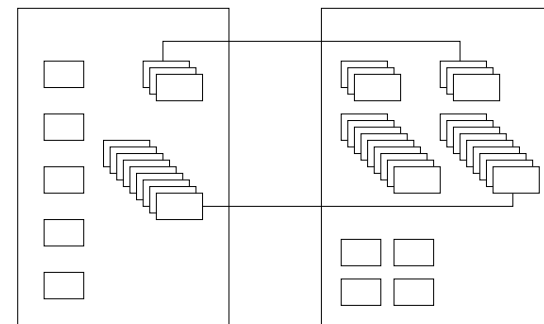
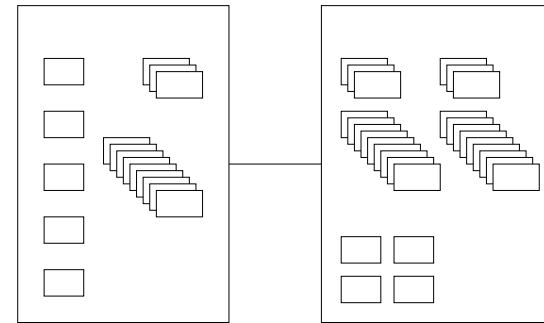
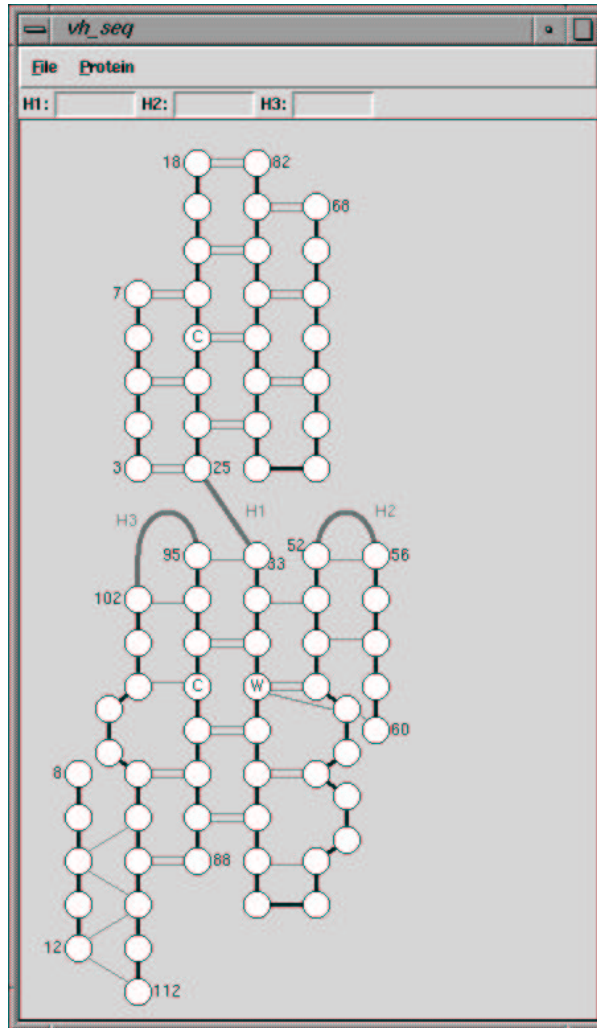
Find all antibody VH domains where the C $\beta$  atoms of the residues at Kabat positions 34 and 78 are within 5 Angstroms.

Print the protein code, the names of these residues, and the titles and references of related publications.

### Daplex:

```
for each d in ig_domain such that name(d) = "VH"
  for each r1 in kabat_residue(d, "34")
    for each r2 in kabat_residue(d, "78") such that
      distance(atom(r1,"CB"), atom(r2,"CB")) < 5.0
      for each p in pdb_entry such that id(p) = protein_code(d)
        for each m in medline_entry such that
          some l in link(p) has id(m)=l
print(protein_code(d), name(r1), name(r2), title(m), ref(m));
```

# Relationships between objects in different databases



# Important issues

- declarative query languages and query optimisation
- data independence
- keys for identifying and linking data
- tight or loose coupling
- heterogeneity and autonomy are inevitable and desirable
- integration requires a shared schema — an agreed understanding of what the data mean and how they are related — expressed in a computer-usable form
- modular design of mediator — federation can evolve incrementally