

Distributed Query Processing and Catalogs for P2P Systems

Vassilis Papadimos David Maier Kristin Tufte
{vpapad,maier,tufte}@cse.ogi.edu

OGI School of Science & Engineering
Oregon Health & Science University

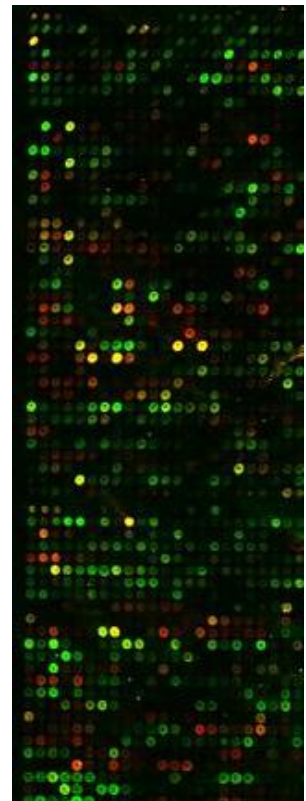
Part of the *Niagara* project: <http://www.cs.wisc.edu/niagara>

Query Processing for P2P?

- P2P systems (Gnutella, Freenet, ...) are great in terms of ease of deployment and use, scalability and fault tolerance
- But the queries are limited (simple selections)
- ... and the shared data are structure-free bags of bytes
- A world of potential P2P applications outside file sharing!
- Example: Sharing and querying gene expression data

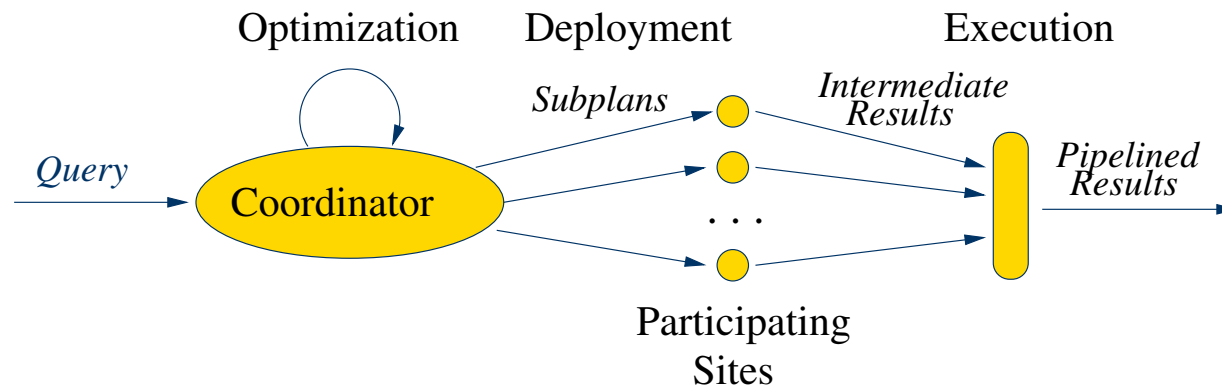
Gene Expression Data

- Genes \times Samples \rightarrow Expression Levels
- MIAME: Minimum Information About a Microarray Experiment
 - Rich metadata, come with their own controlled vocabularies, taxonomies, ontologies: Array design, organism, sample preparation, cell type, ...
 - XML interchange format (MAGE-ML)
- Autonomous research groups producing, hosting, and querying *distributed data*



M. Diehn *et al.*
Stanford University

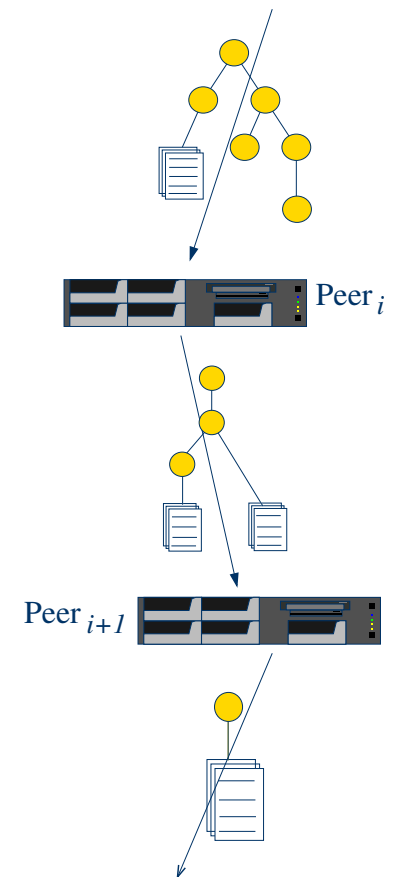
Classical Distributed Query Processing



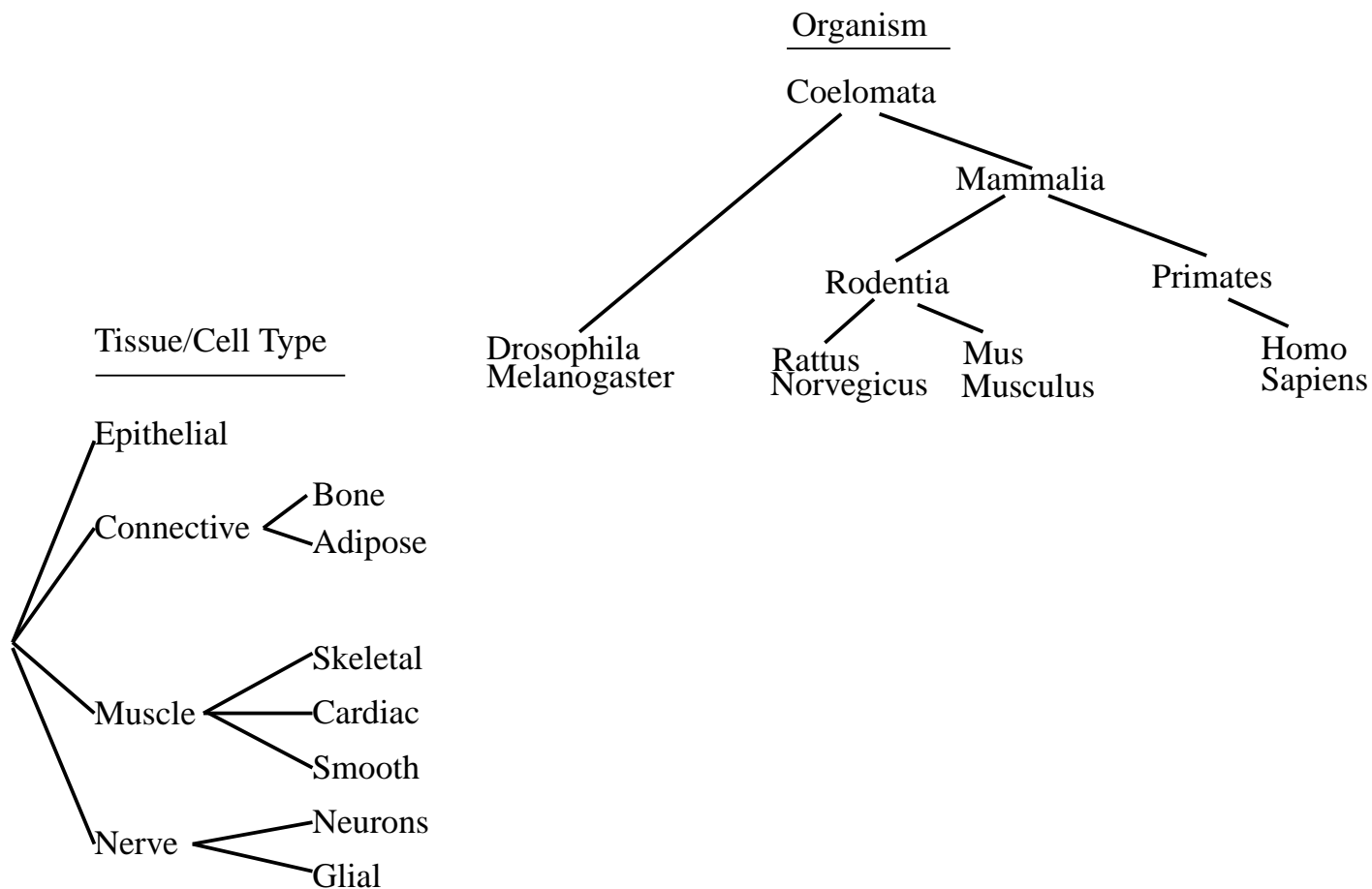
- Optimization depends on accurate catalog info at the coordinator
- During deployment and execution we erect and maintain a complex distributed data structure
- High risk and cost of mistakes with large/long-running queries

Mutant Query Plans

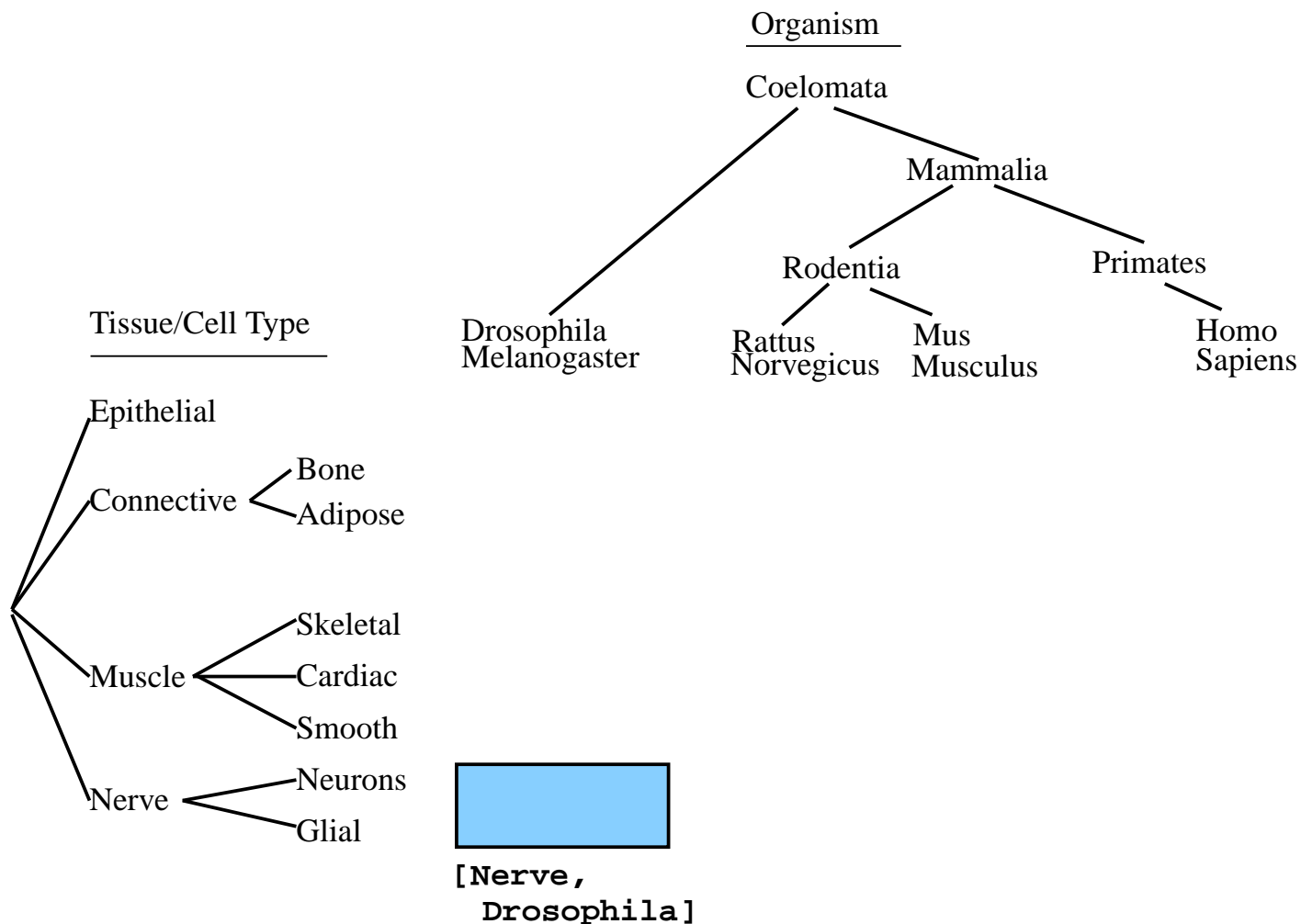
- Package of query operators, resource references, verbatim XML data
- Peers optimize incoming query plans independently, execute subplans, insert results, route mutated queries along
- Use local knowledge, adapt to changing conditions, route around outages
- Distributed query state vs. local state
- But how can we specify what a query is looking for? How do we route it there?



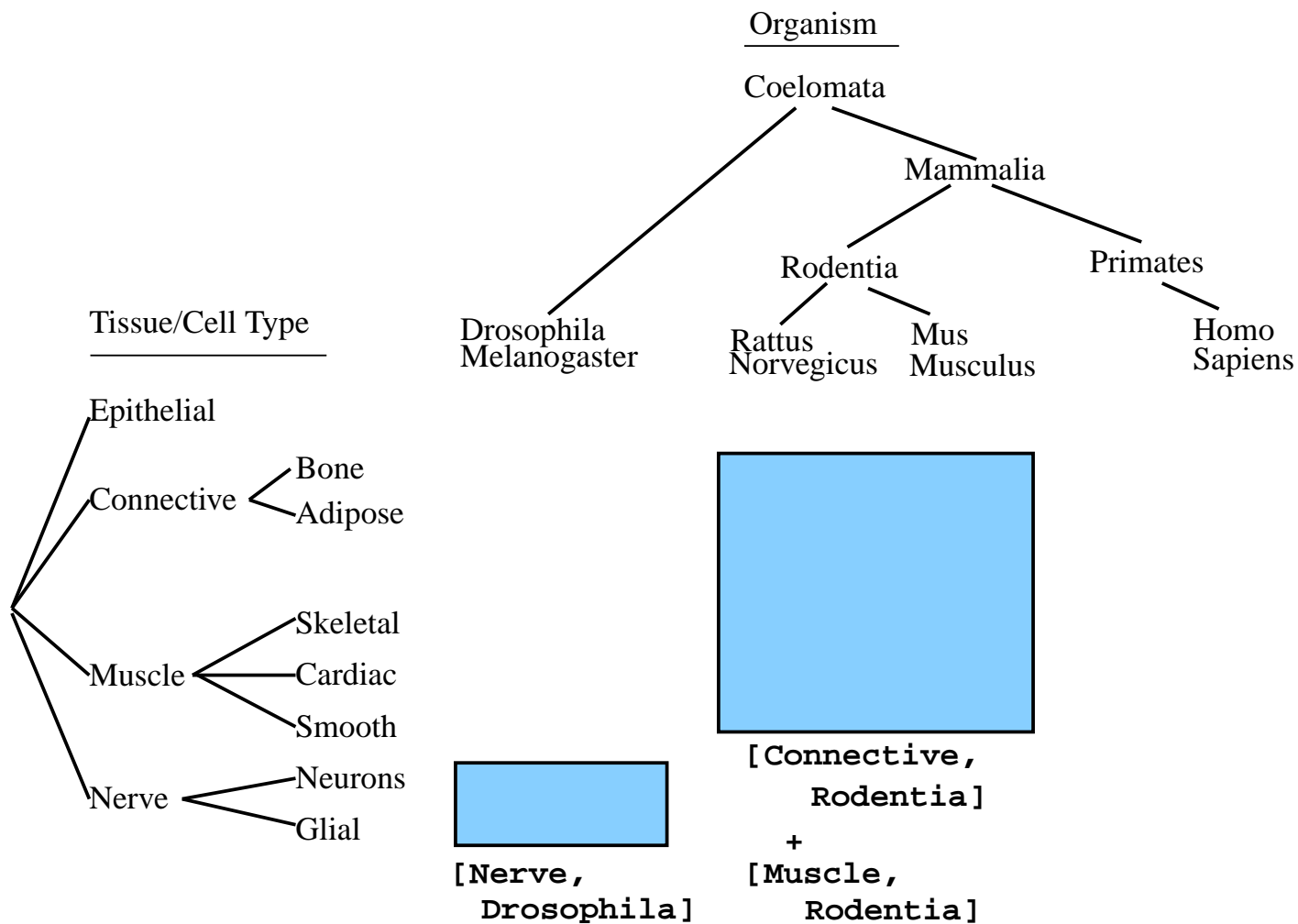
Metadata Hierarchies and Interest Areas



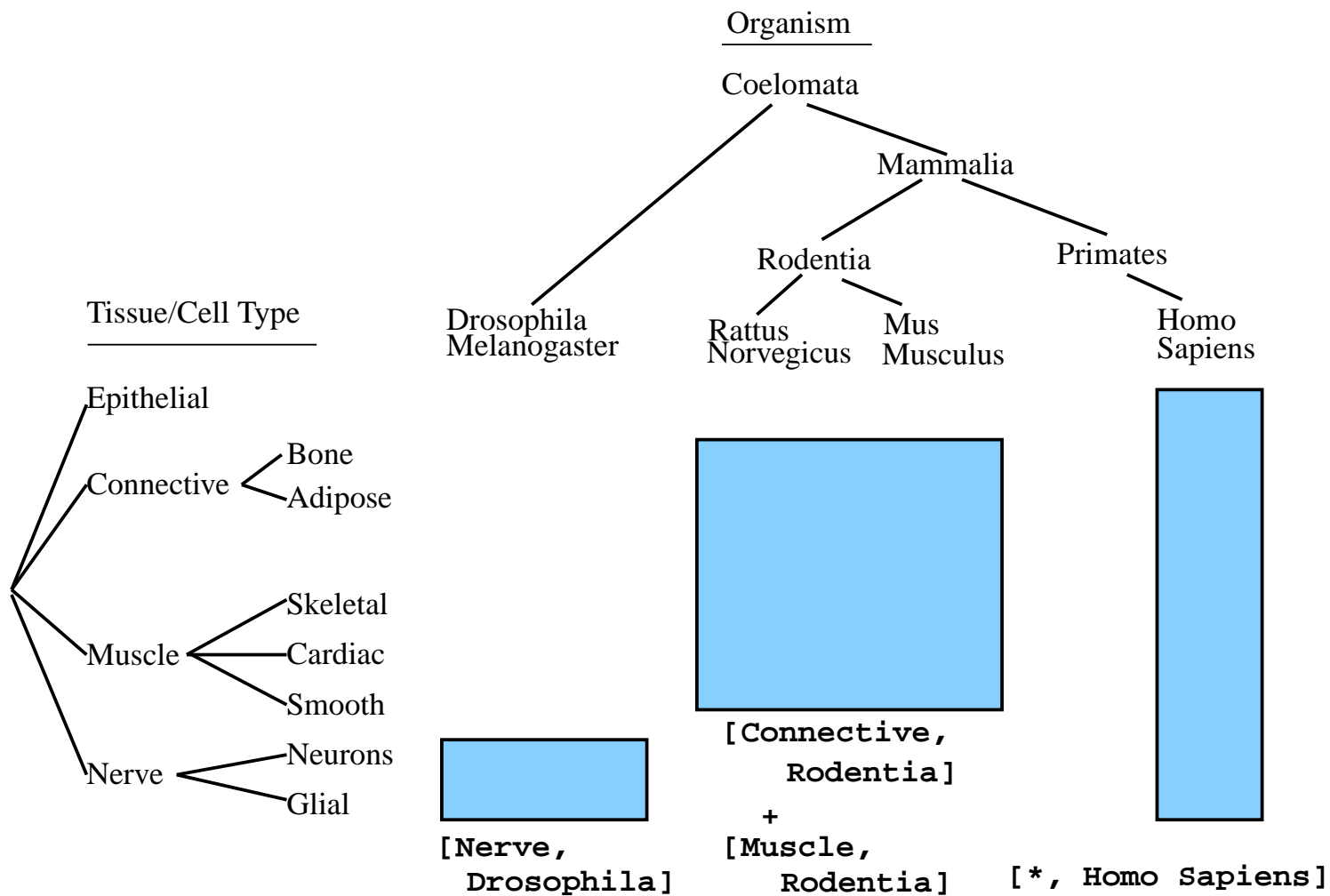
Metadata Hierarchies and Interest Areas



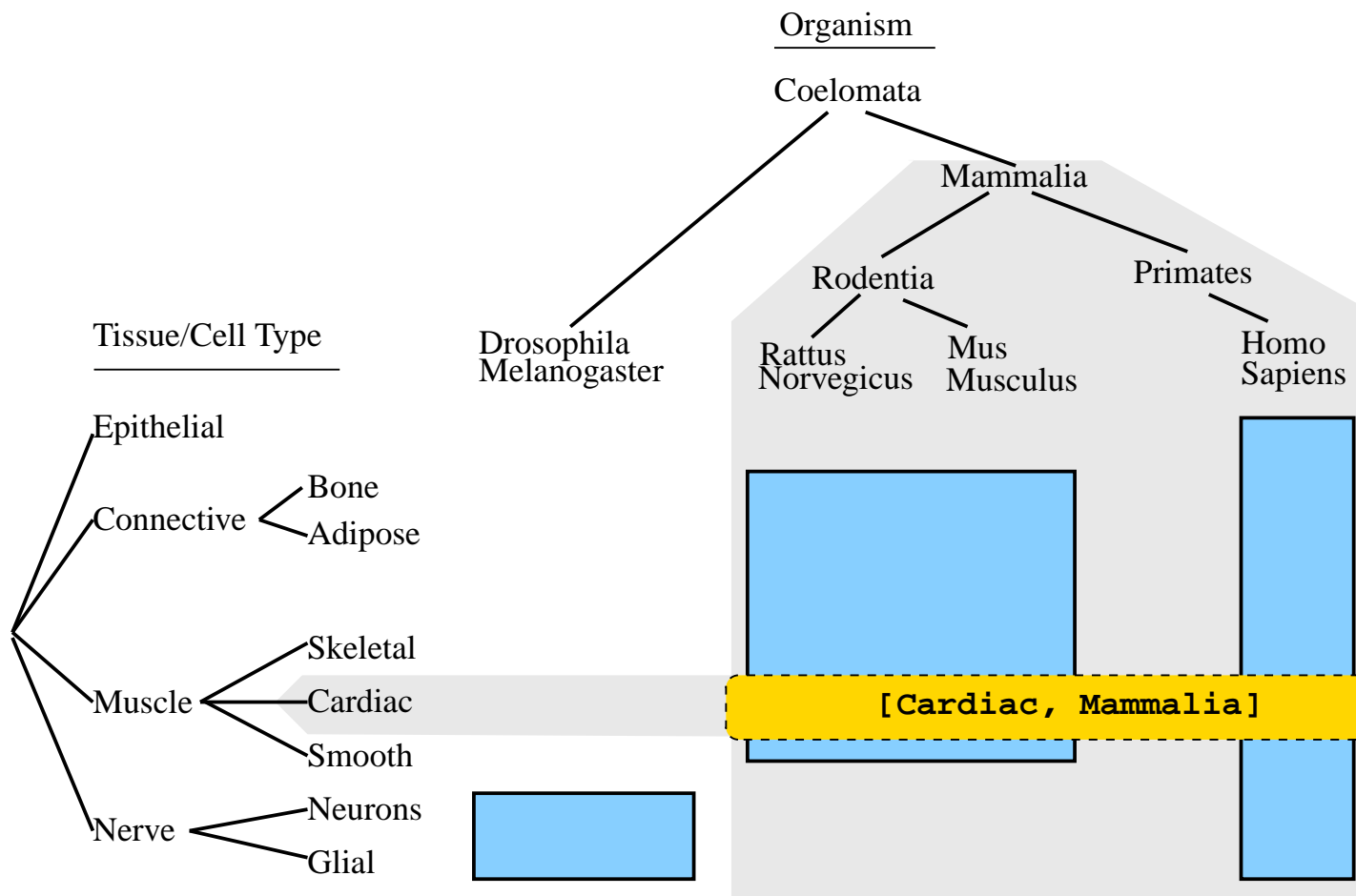
Metadata Hierarchies and Interest Areas



Metadata Hierarchies and Interest Areas



Metadata Hierarchies and Interest Areas



Multi-hierarchic Namespaces

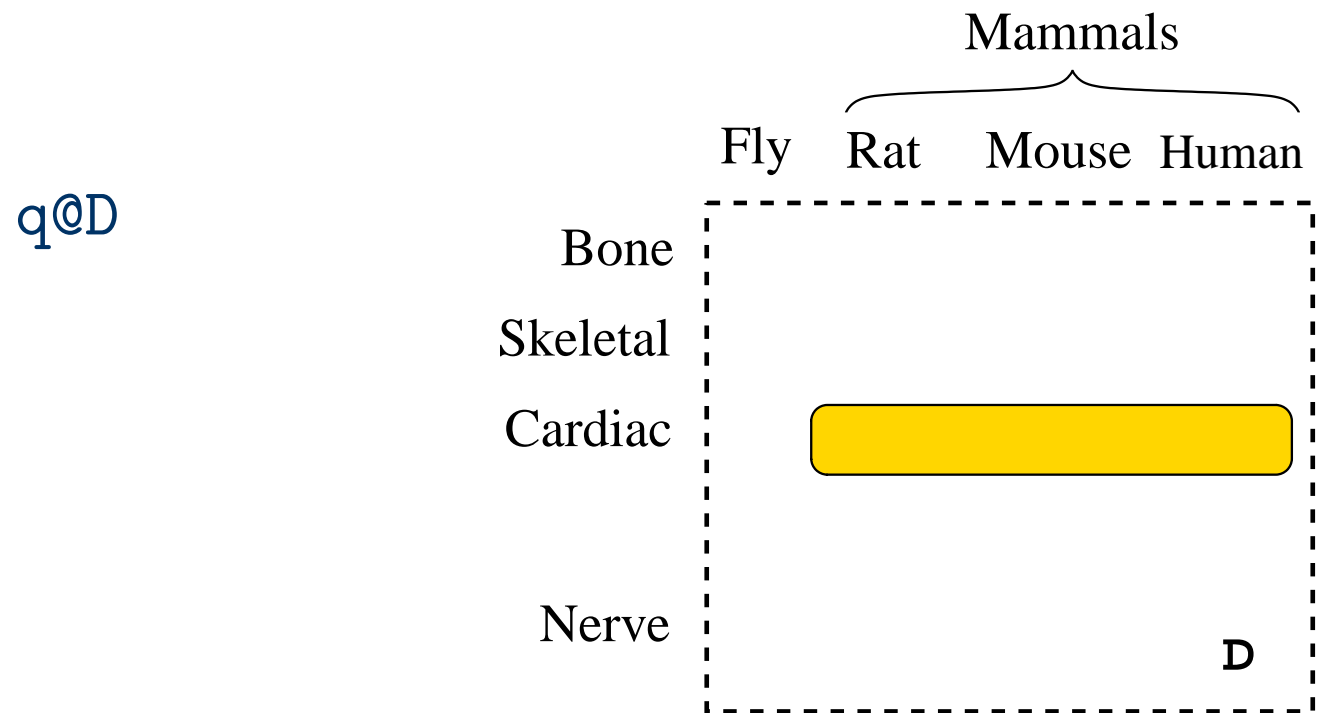
- A (sub)set of the available metadata attributes for an application domain forms its *multi-hierarchic namespace*
- Pick a place in the hierarchy for each dimension and you have an *interest area* (e.g. [Muscle/Cardiac, Primates])
- Peers use interest areas to advertise what they store or index
- Queries include interest areas to specify what they're looking for

Peer Roles

During query execution, a peer may play one or more of the following roles:

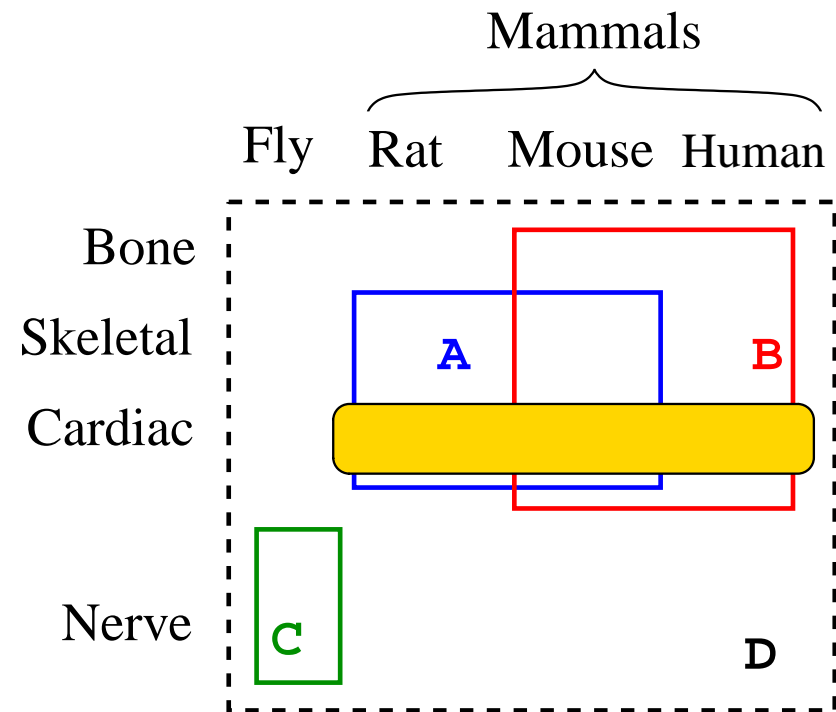
- *Base servers* serve data collections in their interest area
- *Index servers* keep track of base and index servers in their interest area, and can also index non-hierarchic attributes of the base data (e.g. experiment date)
- *Meta-index servers* maintain mappings from interest areas to servers with relevant data (do not index non-hierarchic attributes, typically cover larger areas than index servers)
- *Category servers* handle meta-queries about the hierarchies
- *Clients* are peers that are interested in the query's results

Simple Query Routing



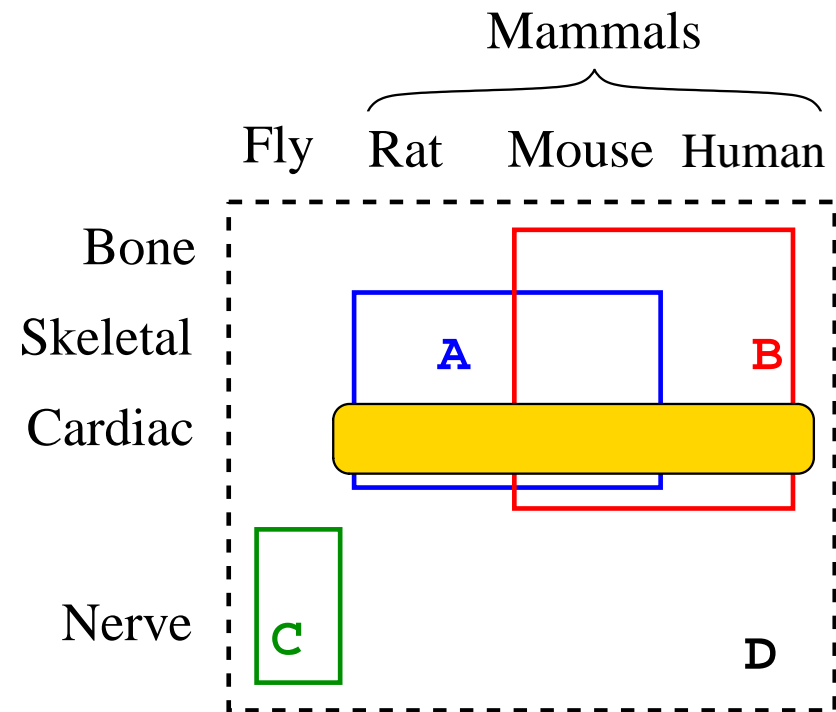
Simple Query Routing

$$\begin{aligned} & q@D \\ = & q@A \cup q@B \\ \rightarrow & A \end{aligned}$$



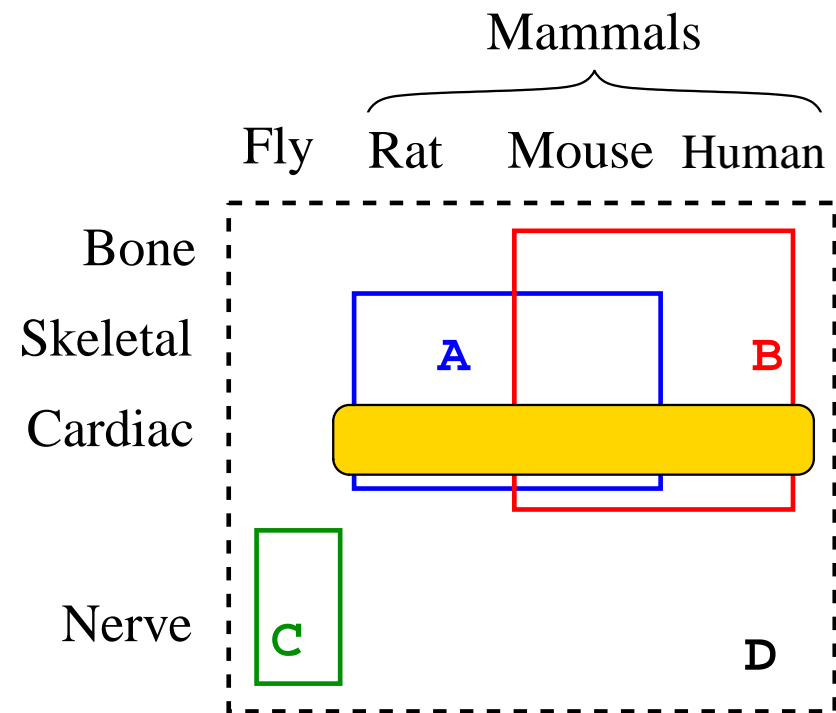
Simple Query Routing

$$\begin{aligned} & q@D \\ = & q@A \cup q@B \\ & \rightarrow A \\ = & \text{ans}(A) \cup q@B \\ & \rightarrow B \end{aligned}$$



Simple Query Routing

$$\begin{aligned} & q@D \\ &= q@A \cup q@B \\ &\quad \rightarrow A \\ &= \text{ans}(A) \cup q@B \\ &\quad \rightarrow B \\ &= \text{ans}(A) \cup \text{ans}(B) \\ &\quad \rightarrow \text{client} \end{aligned}$$



Comprehensive Servers

- Individual index and meta-index servers may not (and usually do not) know all the data sources in their area ...
- but *groups of servers* can collectively strive to be *comprehensive* (at the data or index level)
- Routing a query through a server that belongs to a comprehensive group for its interest area gives the query a good chance to discover all the pertinent base data
- Comprehensive groups maintain containment relationships
- When a new server joins the network, it must find its place among the comprehensive groups that overlap its interest area

Query Routing

- If we're lucky we've cached the address of a server in a comprehensive group that covers the query
- Otherwise, pick a dimension (hopefully the most restricted dimension in the query), and discover servers that cover *, this dimension's root (e.g. through category servers)
- Descend the hierarchy until you find a comprehensive group with the smallest area that still covers the query
- Recursively follow index references to route the query through servers with relevant data

Category servers

- Data about a categorization hierarchy (e.g. what are the families in the `Primates` order?)
- Hierarchies themselves change over time
 - Hopefully, the further up you go, the more stable it gets
 - Inclusion constraints mean that we can rewrite a query to use nodes further up in the hierarchy, with a loss of precision, but no loss of recall
- References to meta-index servers covering the hierarchy root (`[..., *, ...]`)
- Like a DNS server, category servers can delegate portions of their namespace

Signing up

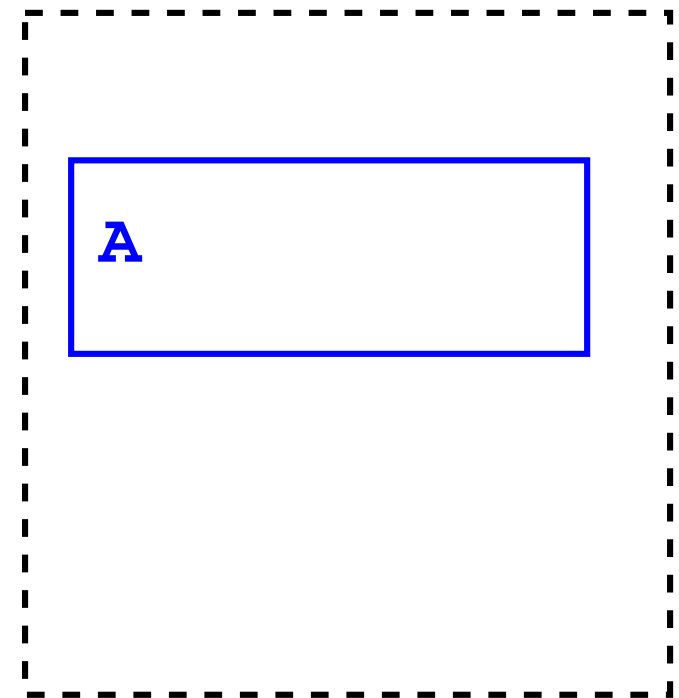
Server A joins the P2P network:



Signing up

Server A joins the P2P network:

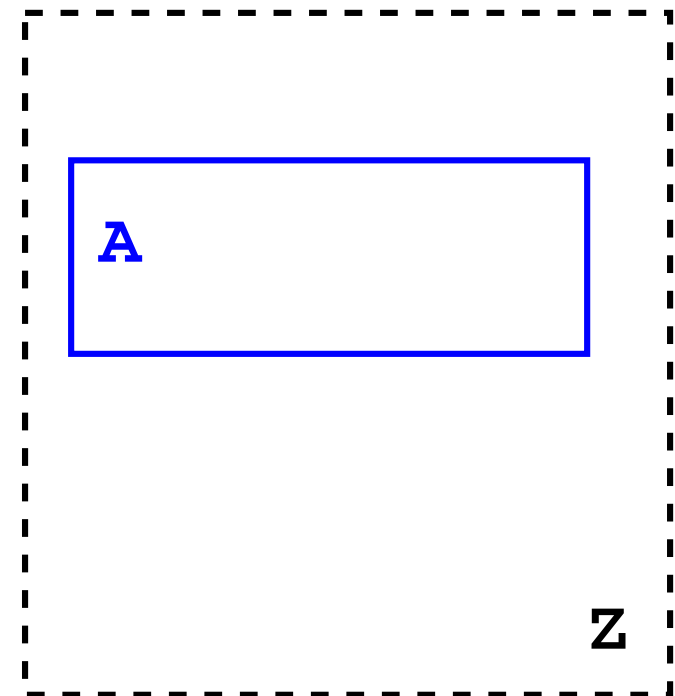
- Discover the smallest comprehensive group that covers its interest area



Signing up

Server A joins the P2P network:

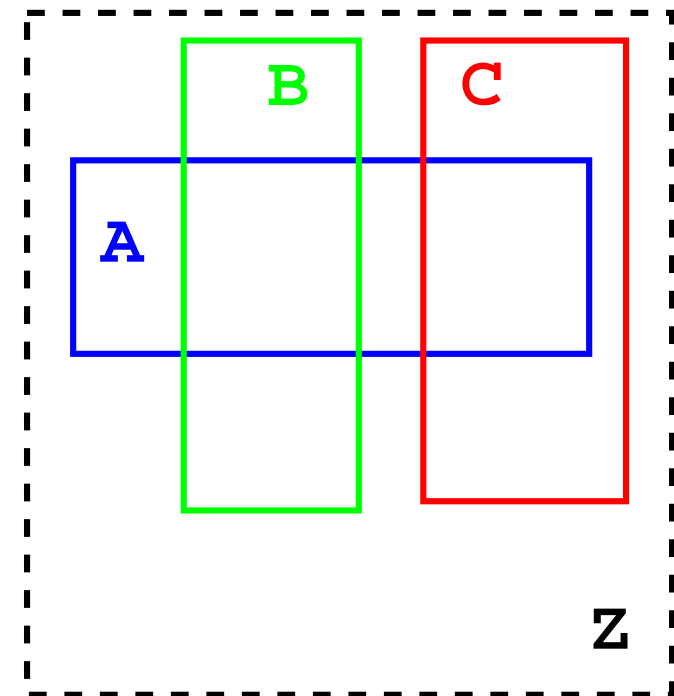
- Discover the smallest comprehensive group that covers its interest area
- Pick a server Z in that group to register with



Signing up

Server A joins the P2P network:

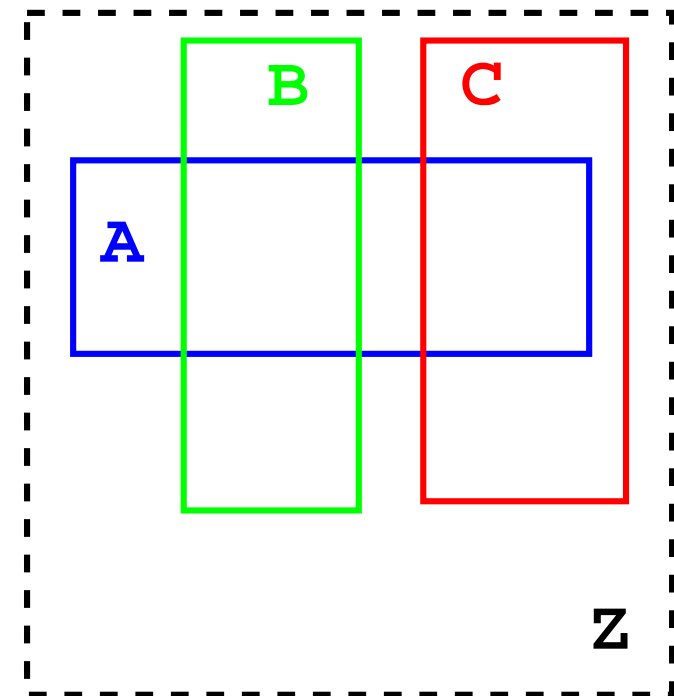
- Discover the smallest comprehensive group that covers its interest area
- Pick a server Z in that group to register with
- Optionally find servers that *overlap* its area (B and C) and form or join comprehensive groups



Signing up

Server A joins the P2P network:

- Discover the smallest comprehensive group that covers its interest area
- Pick a server Z in that group to register with
- Optionally find servers that *overlap* its area (B and C) and form or join comprehensive groups
- Start receiving queries and server registrations



Service Guarantees

- How do we know that a query routed through the network will find all the relevant data?
- In general, we don't!
- We can't compel peers to join comprehensive groups, store or index data, or even stay connected to the network
- We also can't provide any sort of transactional semantics, or guarantee the "freshness" of the results
- But servers can use intentional statements so that queries can make choices between completeness, currency, and latency

Coverage and Redundancy

One way we can improve fault tolerance and query performance is by replicating data and index entries across servers. We can express these relationships using intensional statements:

- Server R replicates all of S's base data on primates:

$$\text{base}[* , \text{Primates}]@R = \text{base}[* , \text{Primates}]@S$$

- R indexes all the data of S and T on humans (and more):

$$\text{index}[* , \text{Homo Sapiens}]@R \supseteq$$

$$\text{base}[* , \text{Homo Sapiens}]@S \cup \text{base}[* , \text{Homo Sapiens}]@T$$

Currency and Latency

Replication and indexing will usually be poll-based processes. Servers can express the polling frequency with intentional statements:

- R replicates all of S's data on bone tissue at least every 30 minutes:

$$\text{base}[\text{Connective/Bone}, *]@R \supseteq \text{base}[\text{Connective/Bone}, *]@S\{30\}$$

Given this statement, how should we route queries on bone tissue?

Tradeoffs

We have two options:

- Visit R and then S (up-to-date, but slow)
- Just visit R (fast, but possibly out of date)

Given a query's "latency budget", and preferences for completeness vs. currency, we can come up with many alternative routing policies to resolve such tradeoffs.

Security- and privacy-sensitive applications come with additional constraints on query routing to avoid sending unencrypted or sensitive data through untrusted servers.

More fun with Mutant Queries

Mutant query plans, besides query operator trees and intermediate data can carry additional information with them, such as:

- Statistics (resource sizes, cardinalities, etc.)
- Updated catalog information
- Result provenance

Future Work

- We have an implementation for the MQP framework
- Working on:
 - Catalog infrastructure implementation
 - Indexing and replication policies
 - * effects on data availability and query performance
 - Query routing policies
 - * completeness/currency/latency tradeoffs