

Bioinformatics as a Queryable Knowledge Map: the Pygr Project

Chris Lee

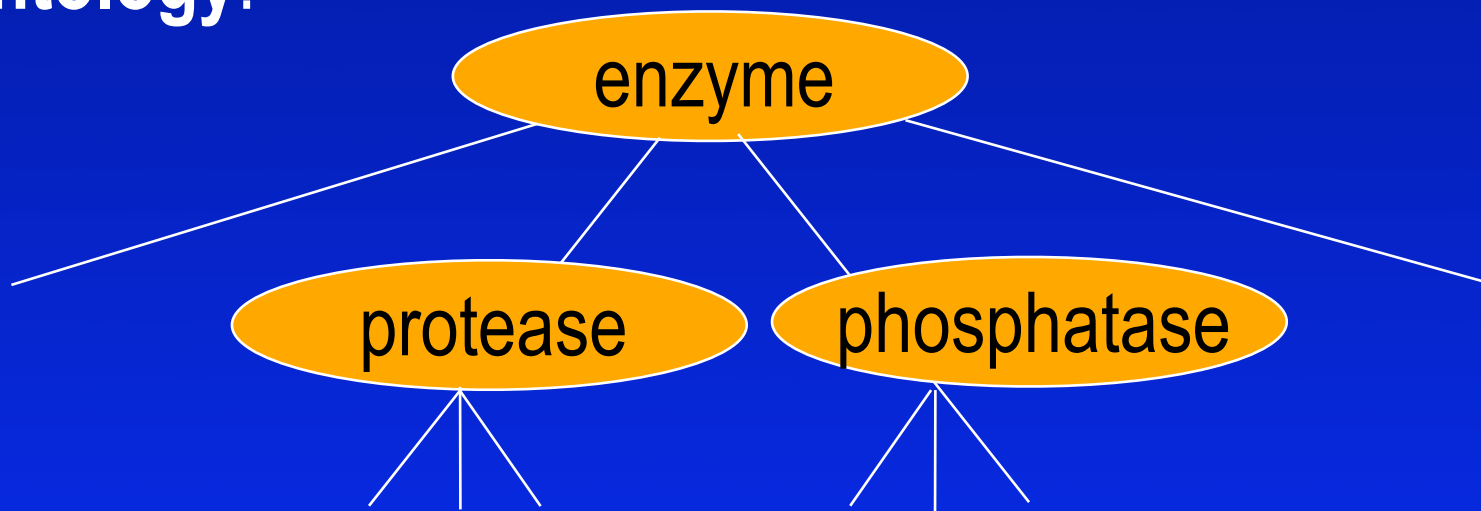
Center for Computational Biology, UCLA

Graph Databases as Knowledge Maps

- If an *in-the-small* tool is like **building a road** that goes one from one kind of data to another, *in-the-large* development is like **making a map** of how all the roads connect, so we can travel everywhere.
- Store these maps as a **graph database**, in which each data *type* is a **node**, and each *transformation* (analysis tool) is a **directed edge** that connects one node to another.

Hypergraphs are a General Model for Bioinformatics

Ontology:



Nodes: terms

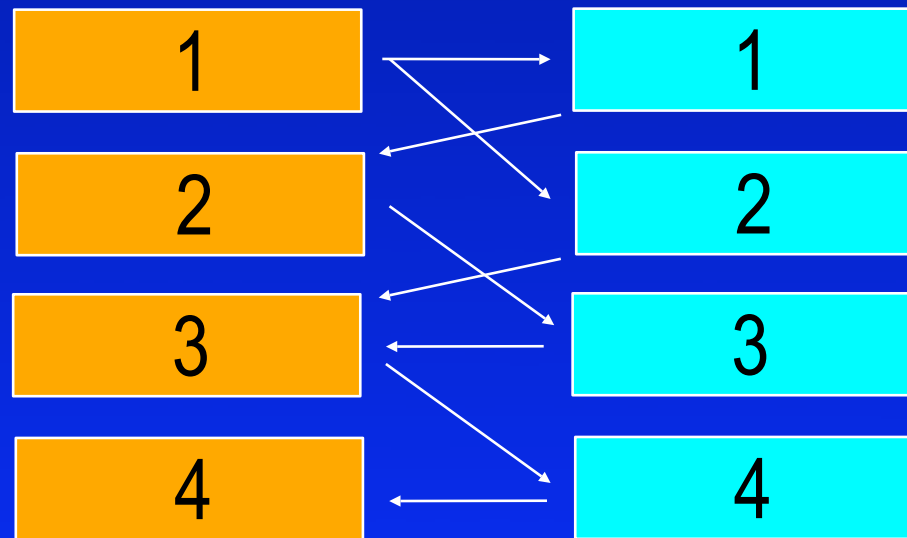
Edges: IS-A, HAS-A relations

Hypergraphs are a General Model for Bioinformatics

Databases:

Exons

Splices

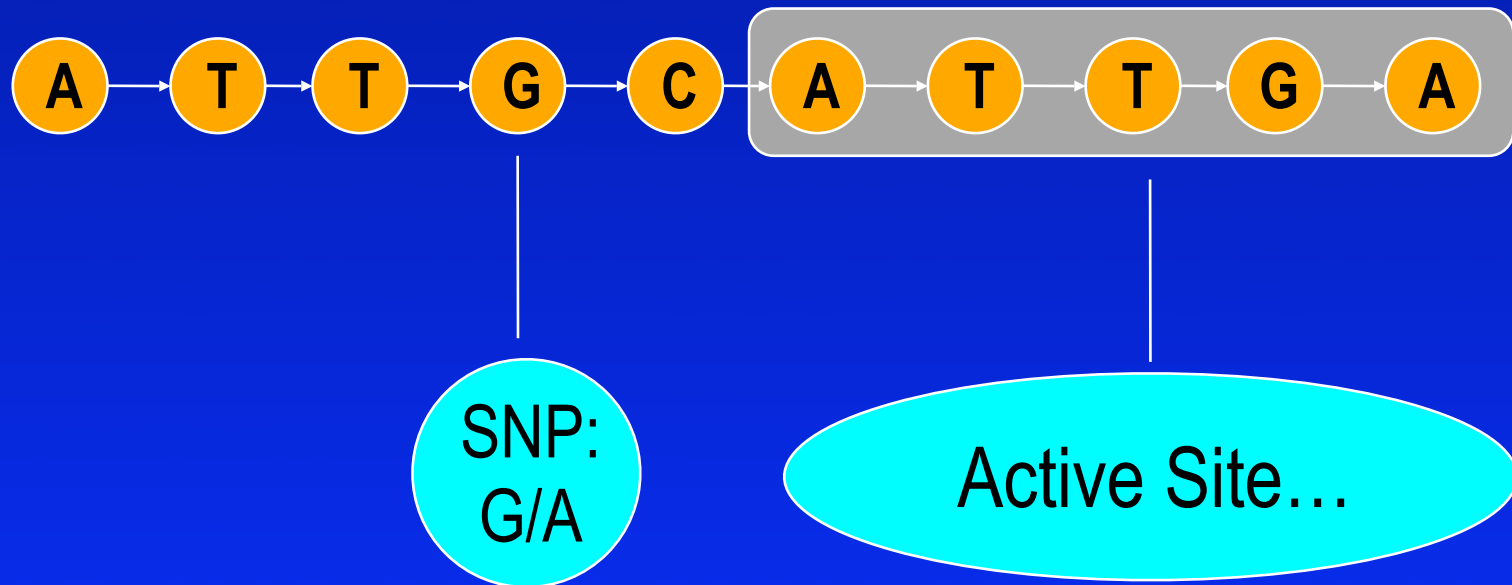


Nodes: rows

Edges: foreign-key relations

Hypergraphs are a General Model for Bioinformatics

Sequence Annotation:

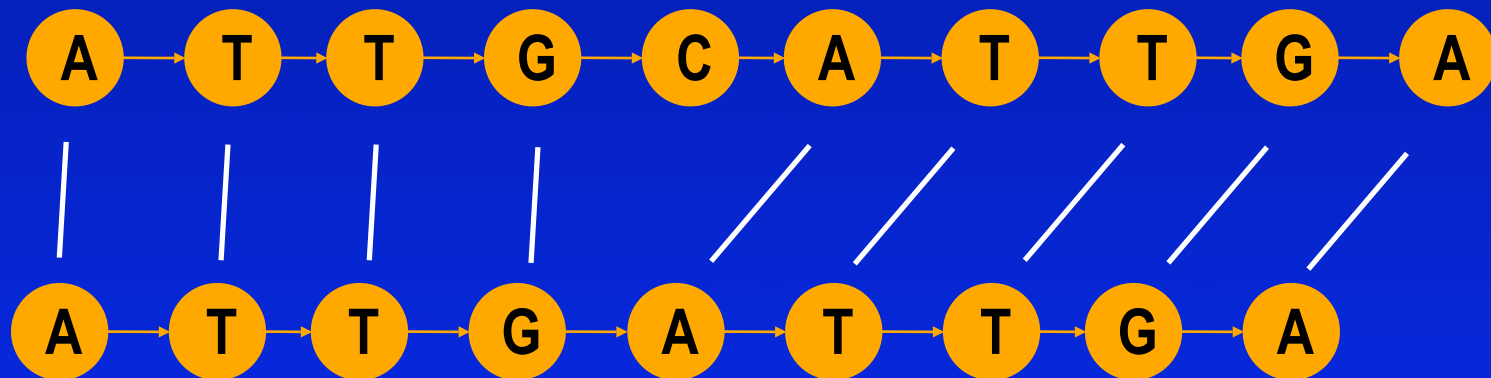


Nodes: **sequence letters**, **annotations**

Edges: links between sequence and annotations

Hypergraphs are a General Model for Bioinformatics

Sequence Alignment:



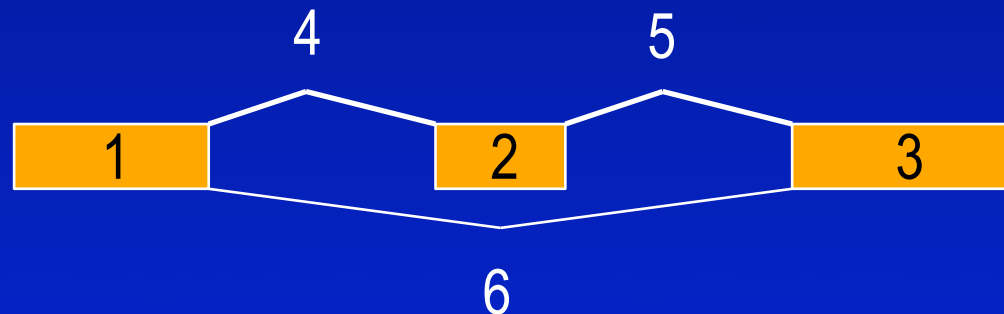
Nodes: sequence letters or intervals

Edges: alignment between letters

Pygr: Python Graph Database Framework

- Map any set of inputs onto any set of outputs
 $M[\text{node1}] \rightarrow \text{node2}$
- The core of the language; but fully extensible.
- Python mappings use indexing for speed.
- Graph: a node can have multiple mappings
 $M[\text{node1}][\text{node2}] \rightarrow \text{edgeinfo}$
(That's all, folks!)

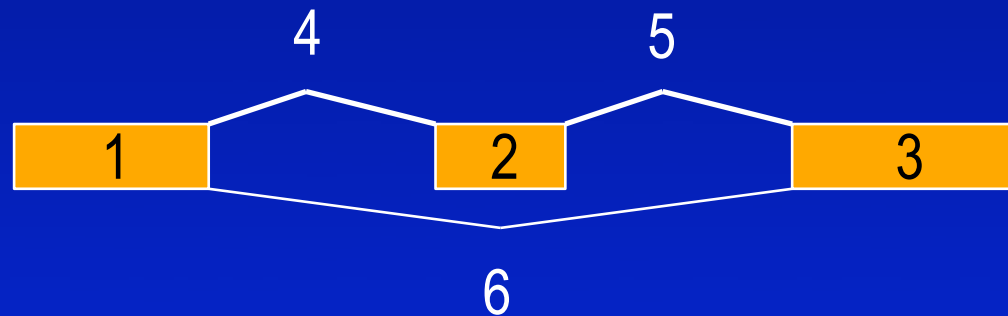
Alternative Splicing Example: SQL



In SQL, a simple exon-skip query requires a 6-way JOIN (groan!)

```
SELECT * FROM exons t1, exons t2, exons t3, splices t4, splices t5,
splices t6 WHERE t1.cluster_id=t4.cluster_id AND
t1.gen_end=t4.gen_start AND t4.cluster_id=t2.cluster_id AND
t4.gen_end=t2.gen_start AND t2.cluster_id=t5.cluster_id AND
t2.gen_end=t5.gen_start AND t5.cluster_id=t3.cluster_id AND
t5.gen_end=t3.gen_start AND t1.cluster_id=t6.cluster_id AND
t1.gen_end=t6.gen_start AND t6.cluster_id=t3.cluster_id AND
t6.gen_end=t3.gen_start;
```

Pygr Graph Query Engine



In Python, the query graph is just:

```
q = {1: {2:None, 3:None}, 2: {3:None}} # no edge info
```

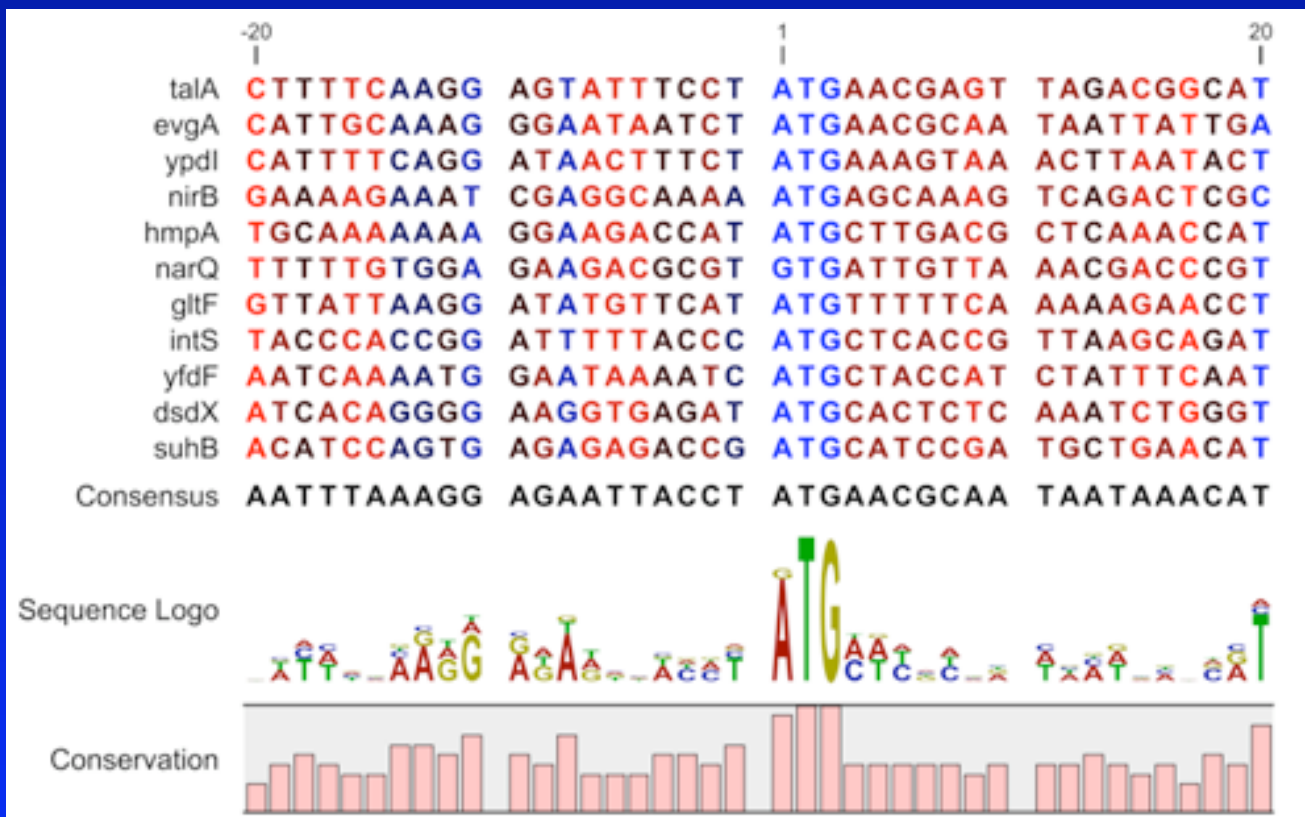
Simpler because the data *really* is a graph; SQL schema is just a (not very good) representation of that.

```
for d in GraphQuery(g,q): print 'exons',d[1],d[2],d[3]
```

Benefits of Graph Query

- The data is actually a graph. Query is also a graph
- The SQL is just a mess, unusable except by experts
- Graph query is simple, works on all these data types
- SQL would require very different queries for each.
- Graph query can easily traverse multiple data types
- Graphs are indexed; e.g. whole genome exonskip query takes 20 - 60 sec.

Multiple Sequence Alignment as a Graph Database



A trivial
alignment
(no gaps or
insertions)

Courtesy of
CLC Bio

Rows are sequences; aligned to show evolutionary homology

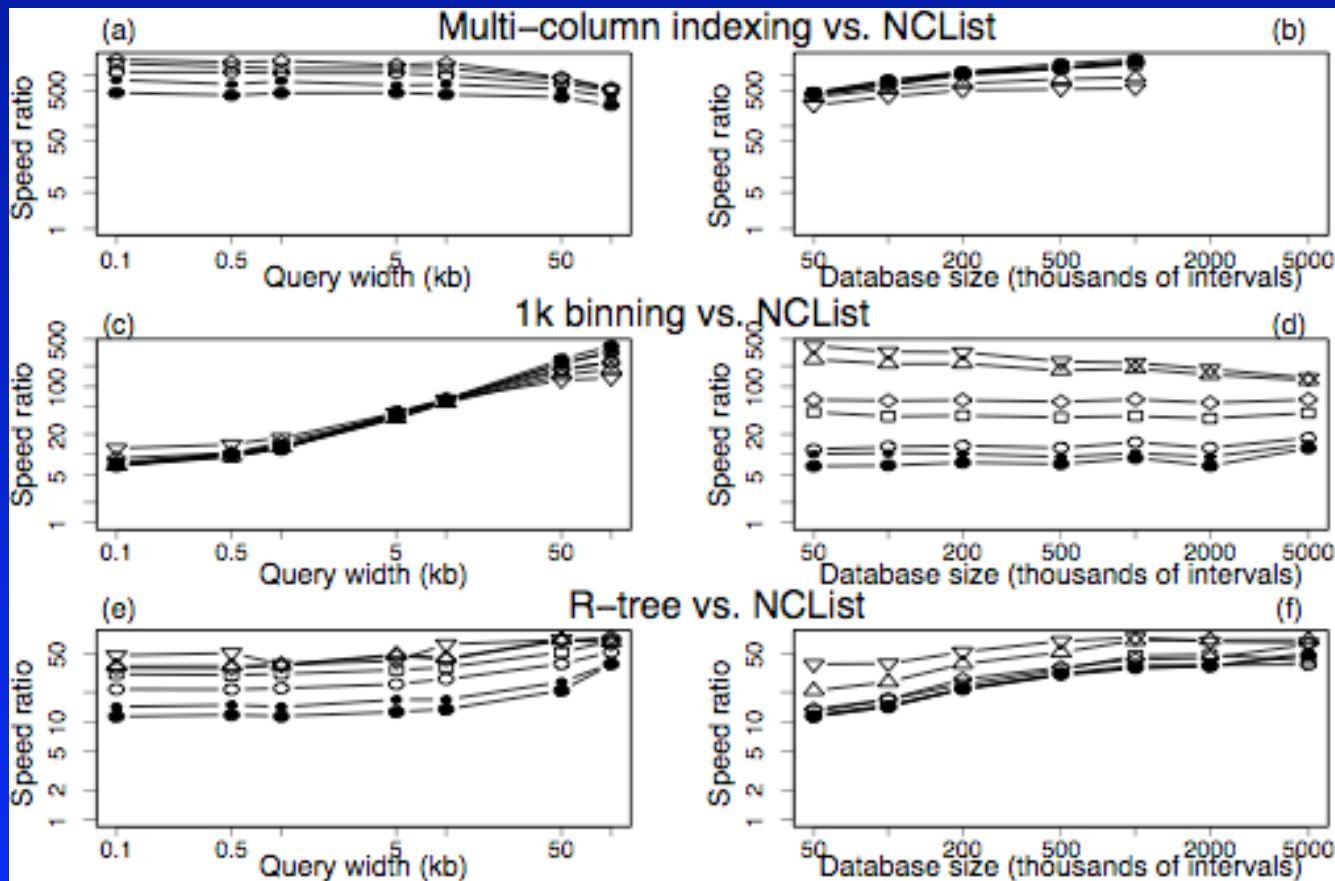
Alignment Query is Graph Query

- *Nodes* are two aligned regions of sequence; *edge* gives information about their alignment.
`msa[region1][region2] → edgeInfo`
- Query: find regions aligned with `region1`:
`for region2 in msa[region1]:`
- This representation scales from single gene, to multiple whole genomes: different storages (disk, memory) will all be interchangeable in usage!

Nice Idea, but is it Scalable?

- Elegant paradigms often fail to be practical for huge problems -- and bioinformatics has big datasets. A normal alignment is typically several hundred letters in length, but just one genome can be 3 billion letters. One comparative genomics project might easily use a terabyte of alignment / sequence data.
- Can Python graph query handle this scale?

NLMSA is 10-500x Faster than R-tree, MySQL for interval overlap query



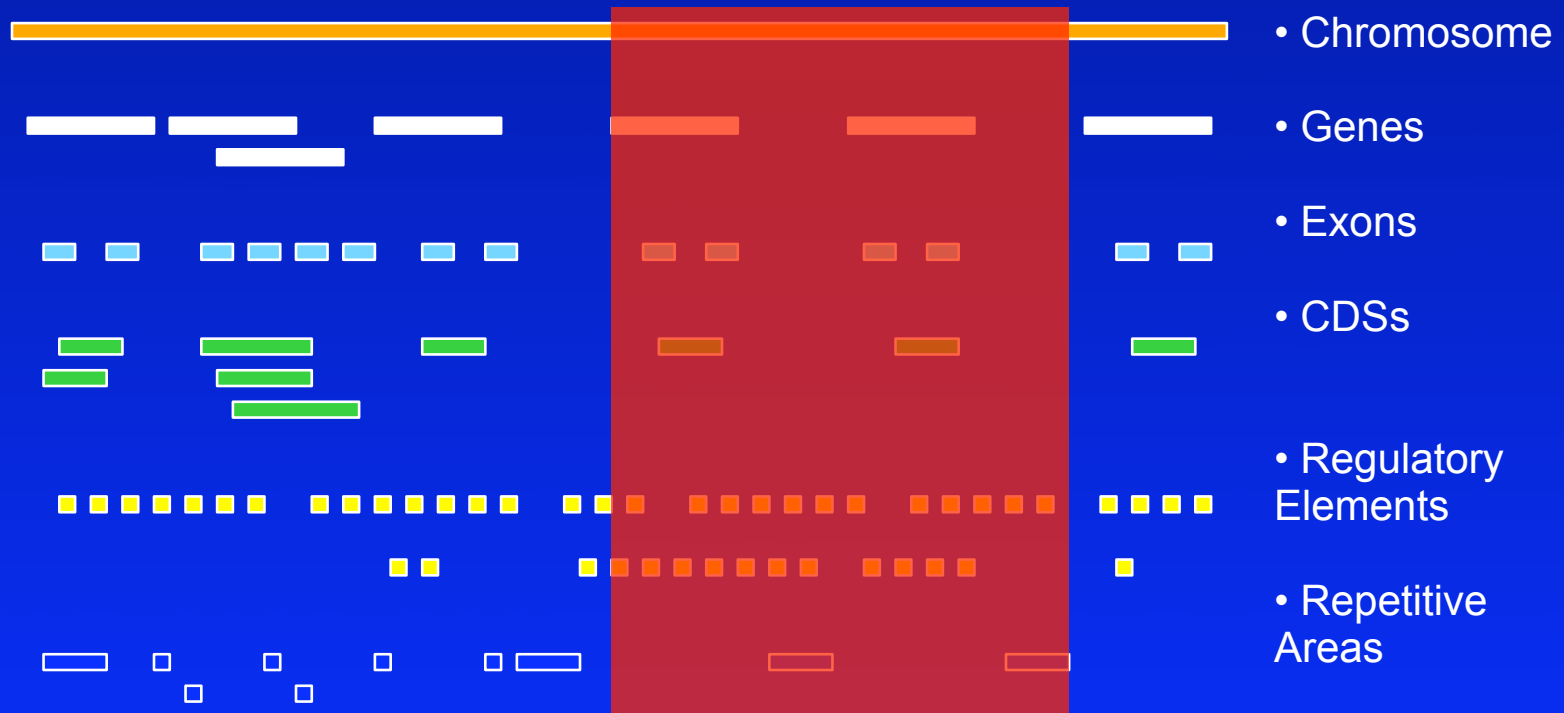
Alekseyenko & Lee, *Bioinformatics* 23: 1386 (2007)

Working with 100+ GB of MSA and Genome Data is Easy

- **NLMSA** stores alignment on disk, provides fast C extension to perform all queries (Pyrex + C).
- **BlastDB** stores sequence databases on disk, provides fast `fseek()` access to sequence.
- With Pygr, analyzing the UCSC 28 vertebrate genome alignment on a laptop is trivial.

Can Also Store and Query Annotations Using NLMSA

- Alignment Interval Overlap Query is the Key Operation



- What does the “interval” I’m interested in align to?

Annotation / Alignment Graph Query

“Find exons in the human genome that have single nucleotide polymorphisms within regions that are highly conserved among 17 vertebrate genomes.”



As a graph query, this biologically interesting question is trivial to formulate.

pygr.Data: a Namespace for Scientific Data & Schema

- We would like to make obtaining a specific dataset (and whatever it depends on) as easy as Python `import foo.bar.you`. All you need is *name*
- The ultimate graph database: all bioinformatics data and their relations could be available in this namespace, and queries could traverse them transparently, because individual data only accessed if a query specifically tries to access it.

Retrieving Resources from pygr.Data

```
import pygr.Data # our data namespace  
hg17 = pygr.Data.Bio.Seq.Genome.HUMAN.hg17() # get it!
```

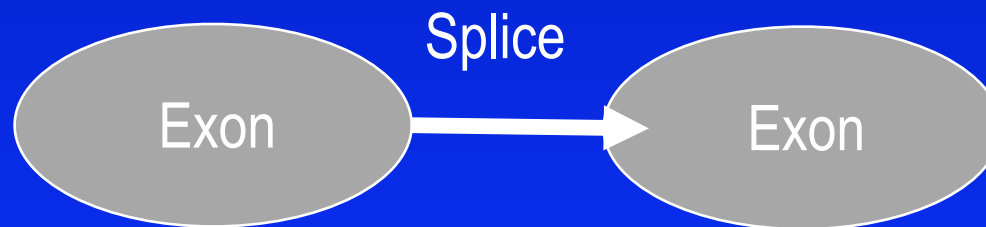
- User doesn't need to know how to find or construct this data, or even what code to use to load it. All you need to know is its name, and you can start working!
- Stores HOW, not WHAT. Works with databases, files
- Searches list of pygr.Data resource databases for this ID. If it has dependencies, searches for their IDs too...

Real Usage Example

- Postdoc gets faculty job (yay!) before finishing a particular project (boo!). I want to finish it, but don't even know where the data are.
- `pygr.Data.dir('Bio.Annotation.DSCAM')` finds the relevant data (with descriptions). To work with a dataset, all I need to know is its name.
- Within seconds I'm working with thousands of exons, 20 different insect genomes, dozens of annotation mappings... totally transparently.

The pygr.Data Schema Graph

Nodes are databases, edges are relations (mappings) as in an ER diagram. e.g. `ManyToManyRelation(exons, exons, splices, bindAttrs=('next', 'previous', 'exons'))`



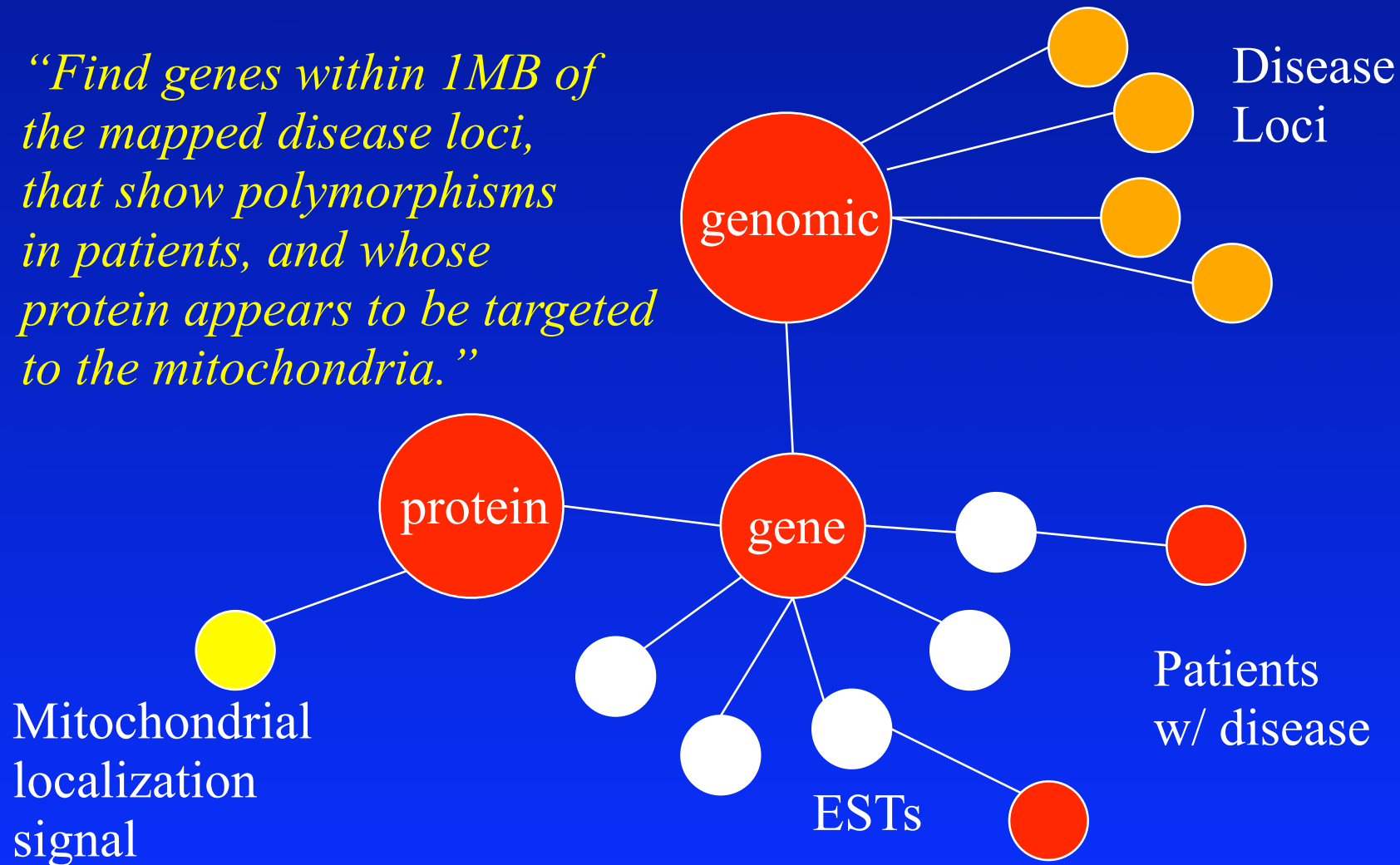
Adds an edge to the *schema graph*, and automatic *attribute bindings*

Pygr.Data Applies Schema Automatically

- If `exon` is obtained from `pygr.Data`, can say:
for `exon2,splice` in `exon.next.items()`:
- `exon.next` means same as `splicegraph[exon]`
- Schema adds `next` as class property (descriptor)
- Automatically seeks `splicegraph`, `splices` only if user actually accesses `exon.next`.
- Transparent: user only needs *attribute name*.
- <http://bioinformatics.ucla.edu/pygr>

Pygr.Data is a Queryable Graph DB spanning Complex Data Types

*“Find genes within 1MB of
the mapped disease loci,
that show polymorphisms
in patients, and whose
protein appears to be targeted
to the mitochondria.”*



Automatically pulls in the right datasets for a query, transparently

Real Usage Example

- I want to check some features of our old exon annotation dataset from my laptop, so I pygr Data
- Shouldn't work; I don't even have the right draft genome on my laptop, I realized later.
- But it did work. Why? Because pygr.Data automatically used fallback XMLRPC service for the genomes that I was missing. Just not as fast.
- Like the web, *always* have access to *all* data.

PYGRDATAPATH: your list of resource databases to use

- E.g. `~`, `.`, `/usr/local/pygr`, `mysql:PYGR.rindex`, `http://biodb2.bioinformatics.ucla.edu:5000`
- I.e. `$HOME`, current directory, `/usr/local/pygr`, a MySQL database, an XMLRPC server, in order.
- Read from first db that finds the ID.
- Local resource rules can take precedence.
- Can specify a desired db via its *layer* name.

Pygr.Data XMLRPC Services

- NLMSA, BlastDB, schema etc. via XMLRPC!
Good performance via caching. Transparent.
- Create server in 3 lines of code; access in 1 line.
- Let your users “plug in” to your data instantly.
- Can try UCLA resource database immediately!

Pygr.Data Auto Download

- The advantage of remote services is that you avoid all of the work required to setup a resource locally. But what if you need maximum speed?
- **download=True** option makes pygr.Data download and build local copies of your desired resource and all necessary dependencies.
- Totally automatic, zero config. Just ask and ye shall receive.

Easy to Add Your Own Resources to pygr.Data

```
from pygr import seqdb
import pygr.Data # our persistent data namespace
hg17 = seqdb.BlastDB('/download/genome/hg17')
hg17.__doc__ = 'human genome sequence draft 17' # required!
pygr.Data.Bio.Seq.Genome.HUMAN.hg17 = hg17 # save as this name
pygr.Data.save() # commit all pending data to resource database
```

- pygr.Data acts like a persistent name space you can assign to. That's all there is to it!
- Use this to manage your own data: it remembers everything you're going to forget!

URS: a Web for Sharing Scientific Data

- URL: just an address; meaningless; unstable.
- Uniform Resource Symbol: a hierarchical catalog of data types; a way for people to agree on standard names for data types and datasets.
- Returns fully “Living Data” with all its methods, relations, schema, and (coming) security.
- Naming conventions; enforceable rules:
Domain.Class.Subclass.Provider.Dataset.Version

The Pygr Project

- 8 releases over the last five years.
- Growing rapidly. Developers from multiple universities (UCLA; Michigan State; UBC; U. Penn.; Korean Bioinformatics Inst.). Awarded two Google Summer of Code projects by PSF...
- Hosted on Google Code; git; Google Groups; weekly developer teleconference. Try it or get involved... Runs on Unix, Mac, Windows.

<http://bioinformatics.ucla.edu/pygr>