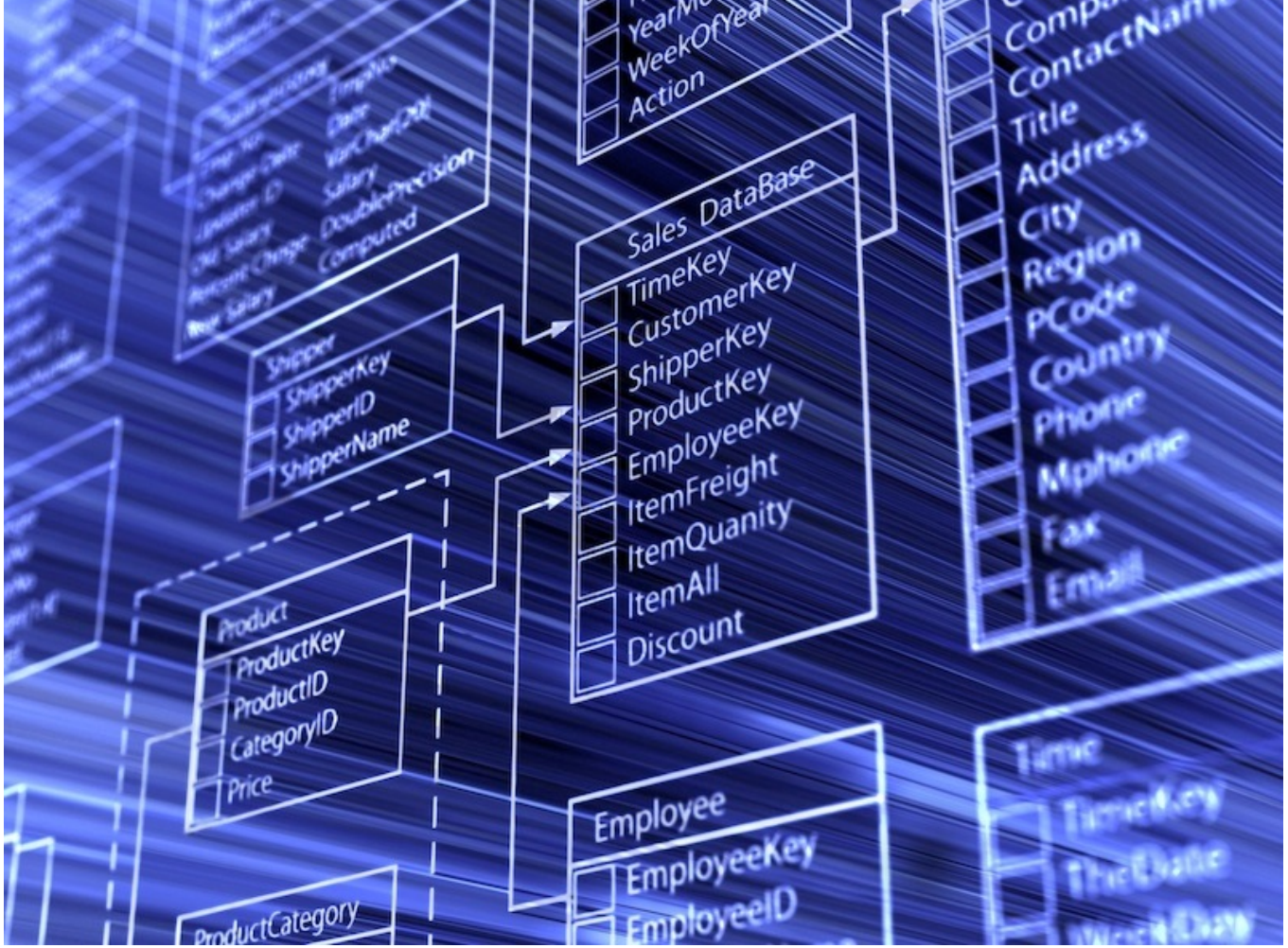


Data Compression & Databases

Practical Introduction to Programming for Scientists - Lecture 11

DNA Sequence

- seq="atggcagctaaagacgtaaaattcggtaacgacgctcgtgtgaaaatgctgcgcggcgtaaacgtactggcagatgca
  gtgaaagttaccctcggtccgaaaggccgtaacgtagttctggataaatctttcggtgcaccgaccatcaccaaagatggtgtttcc
  gttgctcgtgaaatcgaactggaagacaagttcgaaaacatgggtgcgcagatggtgaaagaagttgcctctaaagcgaacga
  cgctgcaggcgacggtaccaccactgcaaccgtactggctcaggctatcatcactgaaggtctgaaagctgttgctgcgggcat
  gaacccgatggacctgaaacgtggtatcgacaaagctgttaccgctgcagttgaagaactgaaagcgctgtccgtaccgtgctct
  gactctaaagcgattgctcaggttggtactatctccgctaactccgacgaaaccgtaggtaaactgatcgctgaagcgatggaca
  aagtcggtaaagaaggcgttatcaccgttgaagacggtaccggtctgcaggacgaactggacgtggttgaaggtatgcagttcg
  accgtggctacctgtctccttacttcatcaacaagccggaaactggcgcagtagaactggaaagcccgttcatcctgctggctga
  caagaaaatctccaacatccgcgaaatgctgccggttctggaagccgttgccaaagcaggcaaaccgctgctgatcatcgctg
  aagatgtagaaggcgaagcgctggcaactctggttgttaacaccatgcgtggcatcgtgaaagttgctgcagttaaagctccggg
  cttcggcgatcgtcgtaaagctatgctgcaggatatcgcaaccctgactggcggtaccgtaatctctgaagagatcggtatggag
  ctggaaaaagcaaccctggaagacctgggtcaggctaaacgcgttgtgatcaacaaagacaccaccaccatcatcgatggcg
  tgggcgaagaagctgcaatccagggccgtgttgctcagatccgtcagcagattgaagaagcaacttctgactacgaccgtgaa
  aaactgcaggagcgcgtagcgaaactggcaggcggcgttgcagttatcaaagtaggtgctgctaccgaagttgaaatgaaaga
  gaaaaaagcacgcgttgaagacgccctgcacgcgacccgtgctgcggtagaagaaggcgtggttgctggtggtggtgttgcg
  ctgatccgcgtagcgtctaaactggctgacctgcgtggtcagaacgaagaccagaacgtgggtatcaaagttgcactgcgtgca
  atggaagctccgctgcgtcagatcgtcctgaactgcggcgaagaaccgtctgttgttgctaacaccgttaaaggcggcgacggc
  aactacggttacaacgcagcaaccgaagaatacggcaacatgatcgacatgggtatcctggacccaaccaaagtaacccgttc
  tgctctgcagtacgcggcttctgtggctggcctgatgatcaccaccgaatgcatggttaccgacctgccgaaaaacgatgcagct
  gacttaggcgctgctggcggtatgggcggcatgggtggcatgggcggcatgatgtaa"

Data Compression

- gzip, zip, bzip2, …
- from zlib import compress,decompress
- compress(<str>,<level>)
- decompress(<cmpstr>)
- great, but…

Compress DNA Seq

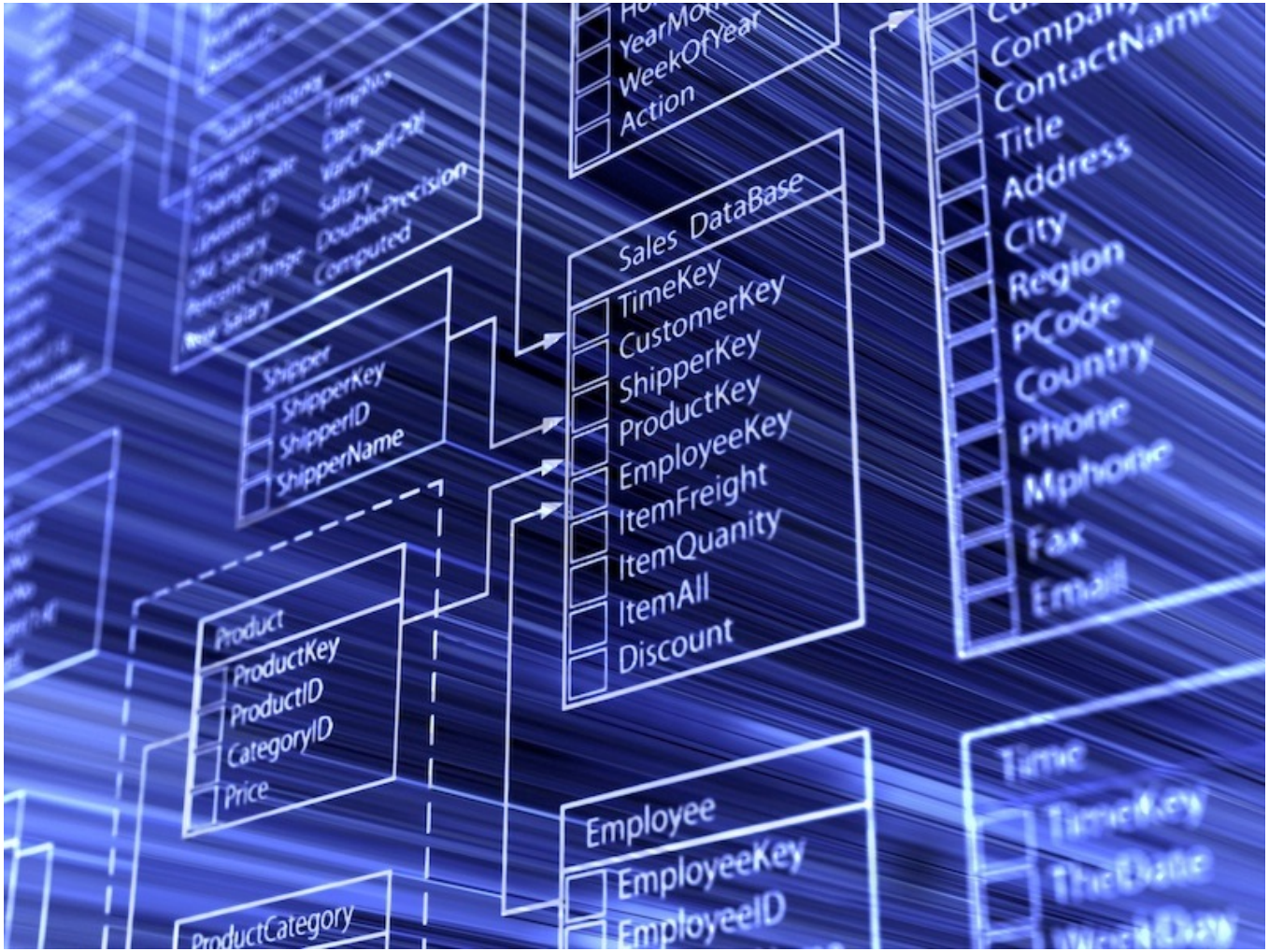
- With gzip, etc.
  - Slooow, 2.8 GB gator genome takes 10 min !
  - No random access of compressed data
- Can we come up with something better ?

Compress DNA Seq

- DNA, only 4 possible values -> 2 bits
- 1 Byte = 8 bits -> 4x compression !!!
- But…
- What about unknowns ?
- 5*5*5 = 125
  - 3x compression
  - naturally ordered into triplets

```
import os

os.chdir("/tmp")

letmap={"a":1,"c":2,"g":3,"t":4,"n":0}

# build compression/decompression dict
tripmap={}
tripmapinv={}
for a in letmap.keys():
    for b in letmap.keys():
        for c in letmap.keys():
            val=chr(letmap[a]*25+letmap[b]*5+letmap[c])
            tripmap[a+b+c]=val
            tripmapinv[val]=a+b+c

print tripmap

seq=file("gator_small.seq","r")
outseq=file("gator_small.cmp","w")

while True:
    trip=seq.read(3)
    if len(trip)==0 : break
    try: outseq.write(tripmap[trip])
    except: break
```

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    FILE *in,*out;

    in=fopen("/tmp/gator_small.seq","r");
    out=fopen("/tmp/gator_small.cmpr","w");

    char seq[3];
    char map[256];
    map['a']=1;
    map['c']=2;
    map['g']=3;
    map['t']=4;
    map['n']=0;
    while (fread(seq,3,1,in)==1) {
        unsigned char xlate=map[seq[0]]*25+map[seq[1]]*5+map[seq[2]];
        fwrite(&xlate,1,1,out);
    }
    fclose(in);
    fclose(out);
}
```


Sales DataBase
TimeKey
CustomerKey
ShipperKey
ProductKey
EmployeeKey
ItemFreight
ItemQuanity
ItemAll
Discount
Shipper
ShipperKey
ShipperID
ShipperName
Product
ProductKey
ProductID
CategoryID
Price
Employee
EmployeeKey
EmployeeID
Title
Address
City
Region
PCode
Country
Phone
Fax
WeekOfYear
Action

How to Store Complex Data ? (again)

- Students
  - Name, address, ...
- Classes
  - Description, Instructor, when offered, ...
- Class Year
  - Which class, year offered, students
- Grades
  - Class, student, grade

Spreadsheet !

Database Terms

- Table - equivalent to a single spreadsheet page
- Database - a set of tables
- Column - one specific type of data for all records in a table
- Row - one record in a table
- Tuple - another name for one row in the database
- Schema - The titles and datatypes of all the rows in all of the tables
- RDMS - Relational Database Management System

Relational Database

Tables (Relations)

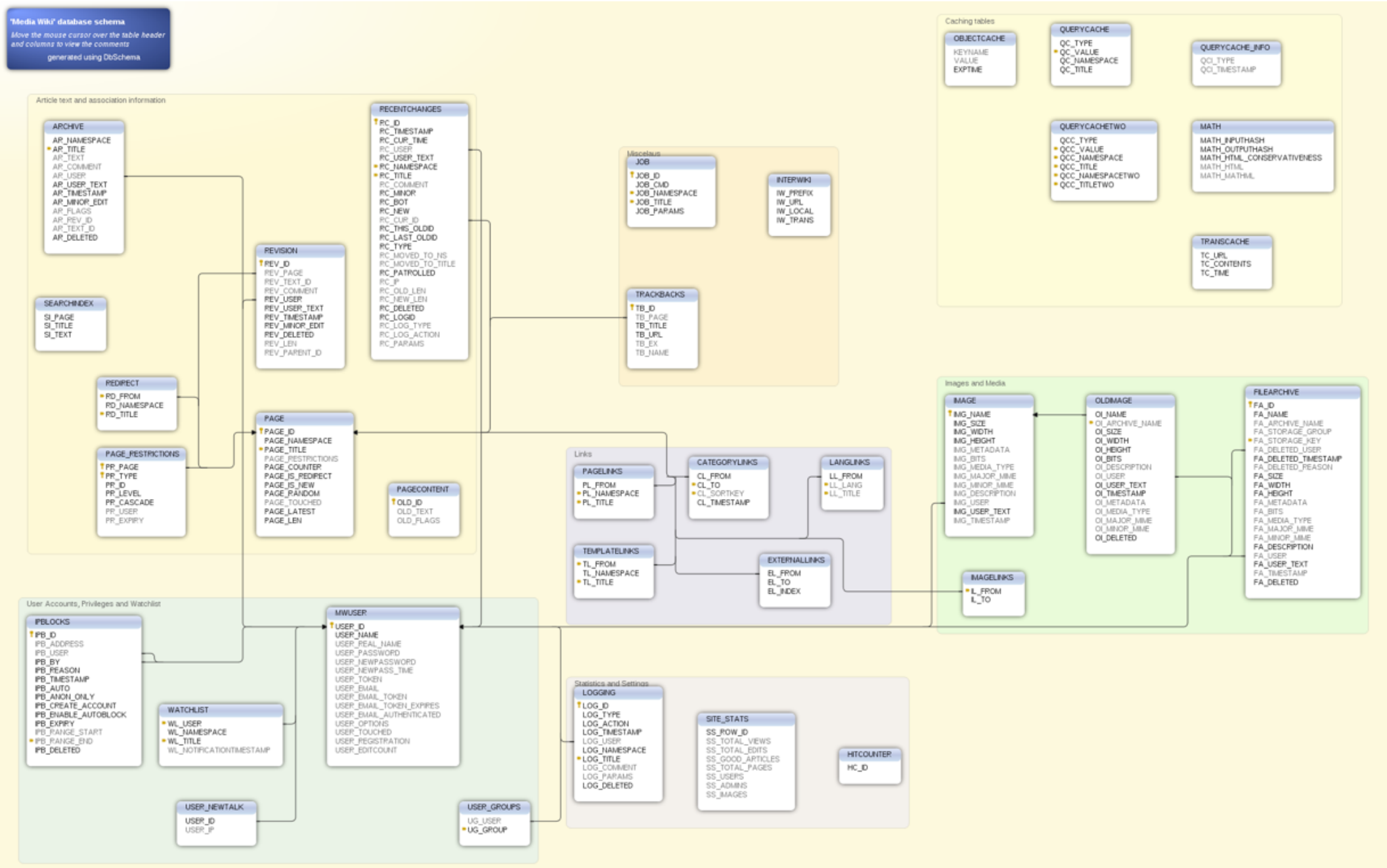
First Name	Last Name	Program	Grade
John	Barnes	1	A
Frank	Smith	1	B+
Carol	Franklin	2	A-
Steve	Black	3	C+
Charles	Baker	1	B+

1 to 1

id	Name	Institution
1	SCBMB	BCM
2	Biochemistry	BCM
3	Biochemistry	Rice

For many->many we use a 'junction table'.

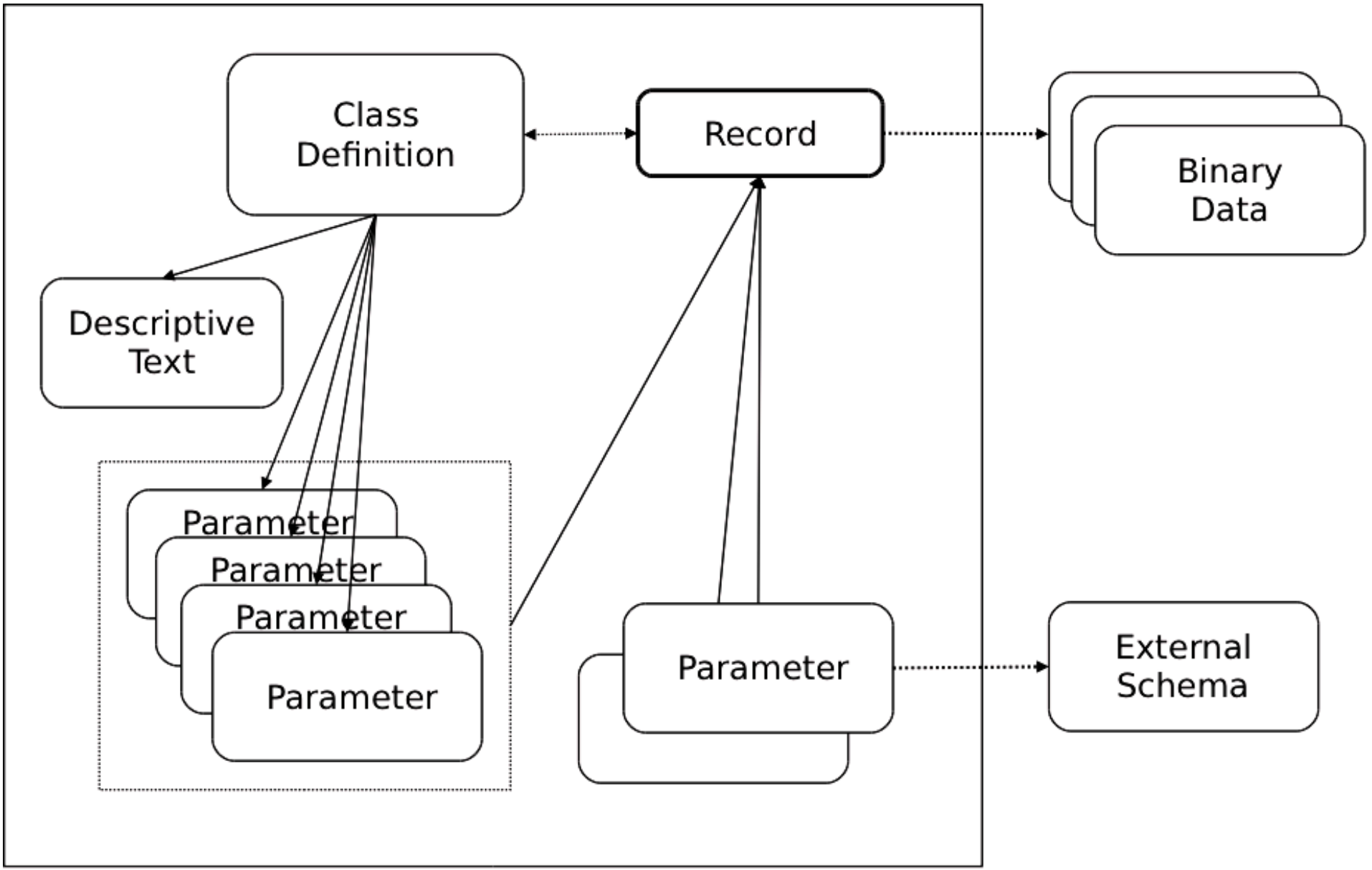
Database Schema



Pythonic Approach

- Dictionary
- Class
- List

OODB


Class Definition
Record
Binary Data
Descriptive Text
Parameter
Parameter
Parameter
Parameter
Parameter
External Schema

Databases

- Simple (embedded) Database
  - dbm, BerkeleyDB (bsddb), SQLite*
- Relational Database
  - MySQL, Oracle, DB2, SQLite*
- Object Oriented Database (OODB)
  - Zope, Databeans
- XML Database
  - Sedna, Oracle, BerkeleyDB-XML

Simple/Embedded Database

- Key/Value pairs
- Python
  - Shelve — Persistent dictionary
  - anydbm — Generic access to DBM-style databases
  - whichdb — Guess which DBM module created a database
  - dbm — Simple "database" interface
  - gdbm — GNU's reinterpretation of dbm
  - dbhash — DBM-style interface to the BSD database library
  - dumbdbm — Portable DBM implementation

    bsddb (bsddb3) — Interface to Berkeley DB library

pickle

- 'Serialization' - converting a complex object to a stream of data

```
from cPickle import dump,load,dumps,loads

dump(obj,file[,protocol])   # stores ‘obj’ in file

obj=load(file)              # restores ‘obj’ from file

str=dumps(obj[,protocol])   # pickled representation of obj

obj=loads(str)              # restore representation of obj
```

shelve

```
import shelve      # dictionary-like object on disk
dct=shelve.open(filename[,protocol])
dct=shelve.open(filename,writeback=True)
dct.close()
```

ACID

- Atomicity
  - Transactions are 'all or none', no partial modifications
- Consistency
  - Contents of the database are always valid and self-consistent
- Isolation
  - Readers will not see partially completed transactions of writers
- Durability
  - Completed transactions can survive any possible system crash

SQL

- Structured Query Language
- Developed at IBM in the 70s
- ANSI Standard in 1986
- Now fairly ubiquitous for Relational Databases
- (see Wikipedia for more trivia)

SQLite3

```
import sqlite3

conn=sqlite3.connect(path)        # Create a database 'connection'

curs=conn.cursor()                # Create a cursor object

curs.execute(SQLstring)           # Execute a query

conn.commit()                     # After making any changes

curs.close()                      # close the cursor when you're done
```

SQLite3

Python type	SQLite type
`None`	`NULL`
`int`	`INTEGER`
`long`	`INTEGER`
`float`	`REAL`
`str` (UTF8-encoded)	`TEXT`
`unicode`	`TEXT`
`buffer`	`BLOB`

SQLite3

- conn=sqlite3.connect("mydb.db")
- cur=conn.cursor()
- create table
  - cur.execute("CREATE TABLE Person(Id INT, Name TEXT, Zip INT);")
- insert
  - cur.execute("INSERT INTO Person VALUES (1,'Steve',77884)")
- select
  - cur.execute("SELECT * FROM Person")
  - print cur.fetchall()
- update
  - cur.execute("UPDATE Person SET zip=77584 WHERE Name='Steve'")
- drop
  - cur.execute("DROP TABLE Person")

Presentations Monday!

- Remember to send me your class projects by midnight SATURDAY
- Follow all directions on the class website!
- Remember you will have only 5 minutes to present. Practice your talk in advance, and if possible, test the projector with your laptop.