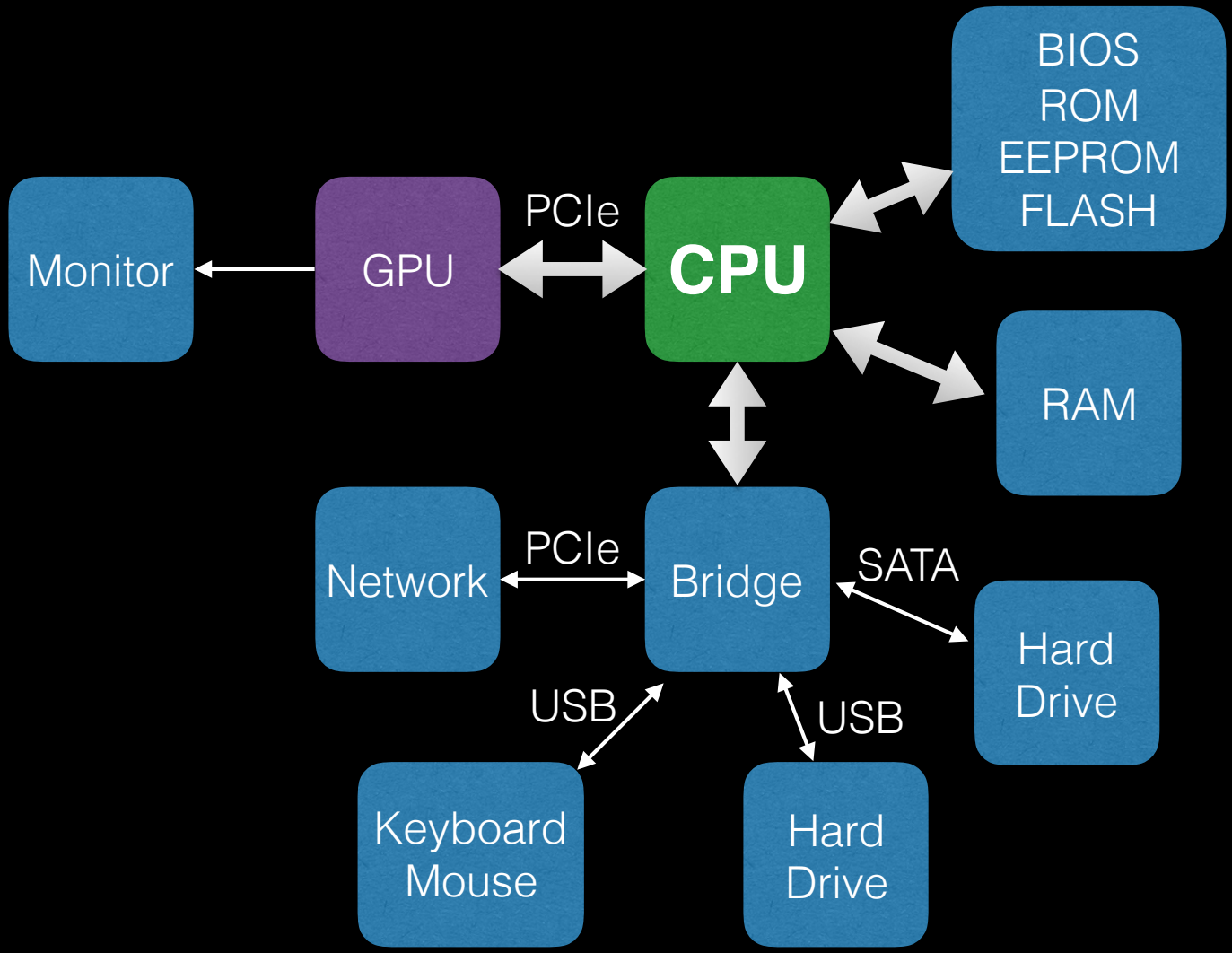
Computer Hardware Data Compression & Databases

Lecture 10

Inside a Computer



Acronyms

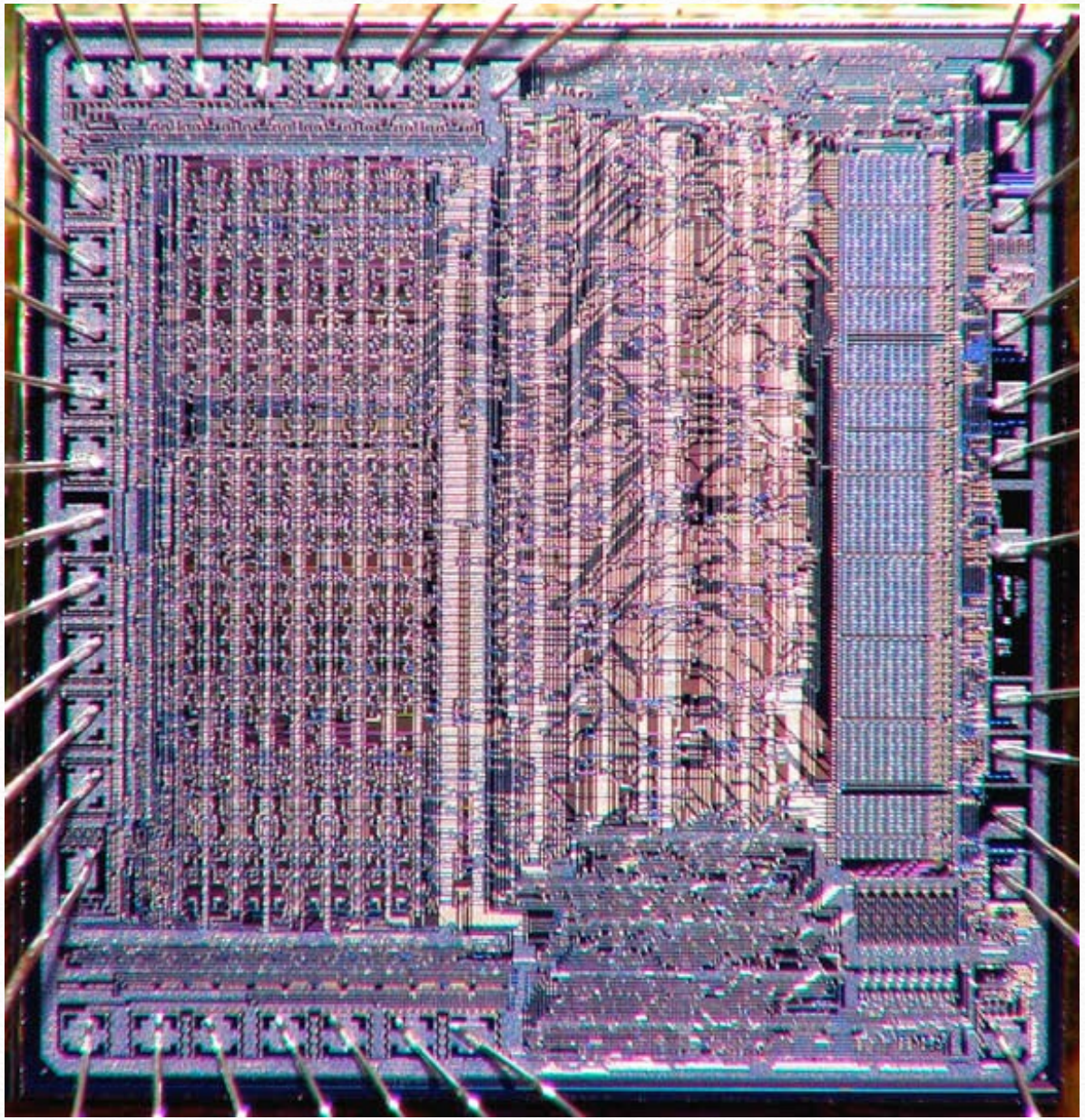
- CPU - Central Processing Unit
- GPU - Graphics Processing Unit
- RAM - Random Access Memory
- BIOS - Basic Input/Output System
- ROM - Read Only Memory
- EEPROM - Electrically Erasable Programmable Read Only Memory
- PCI - Peripheral Component Interconnect
- USB - Universal Serial Bus
- SATA - Serial AT Attachment
- SSD - Flash memory based hard drive

What Can CPUs Do ?

- Store numbers (1 & 0)
- Rearrange stored numbers
- Math
- Simple decisions based on numbers
- Communicate

6800 CPU

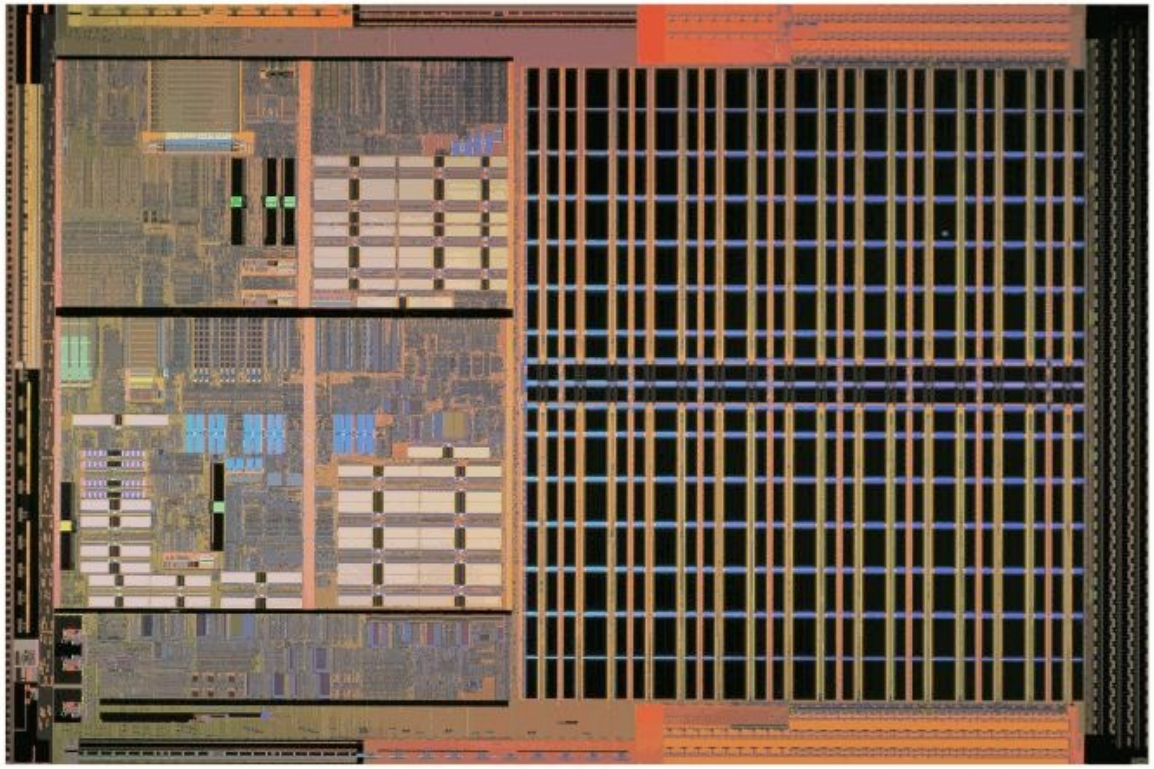
- Introduced 1974
- 4000 transistors
- 1.0-2.5 MHz
- 3, 8 bit registers
- 3, 16 bit registers



www.cpu-world.com

Athlon-64

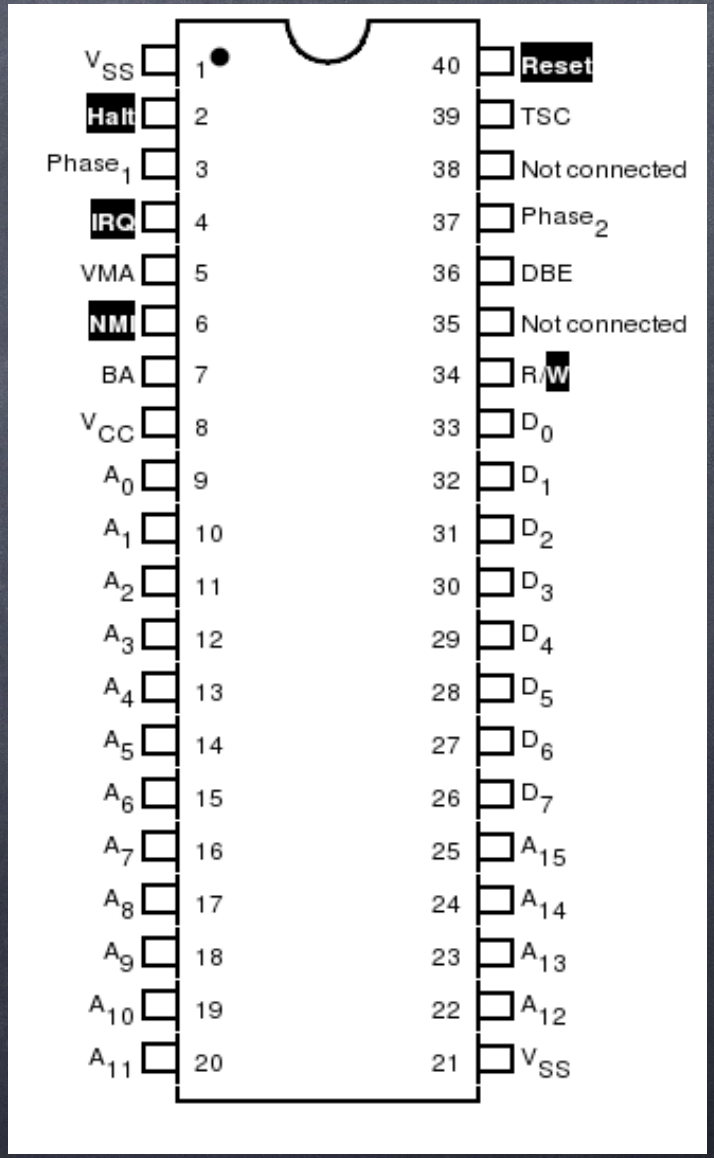
- ~106 million transistors (~10 m^3 if individually packaged)
- Socket-939 (939 pins)
- 40 bit addressing (1 TB)
- 64 bit data bus
- ~2 GHz
- registers:
- 16, 64 bit integer
- 16, 128 bit 'media'
- 8, 64 bit float



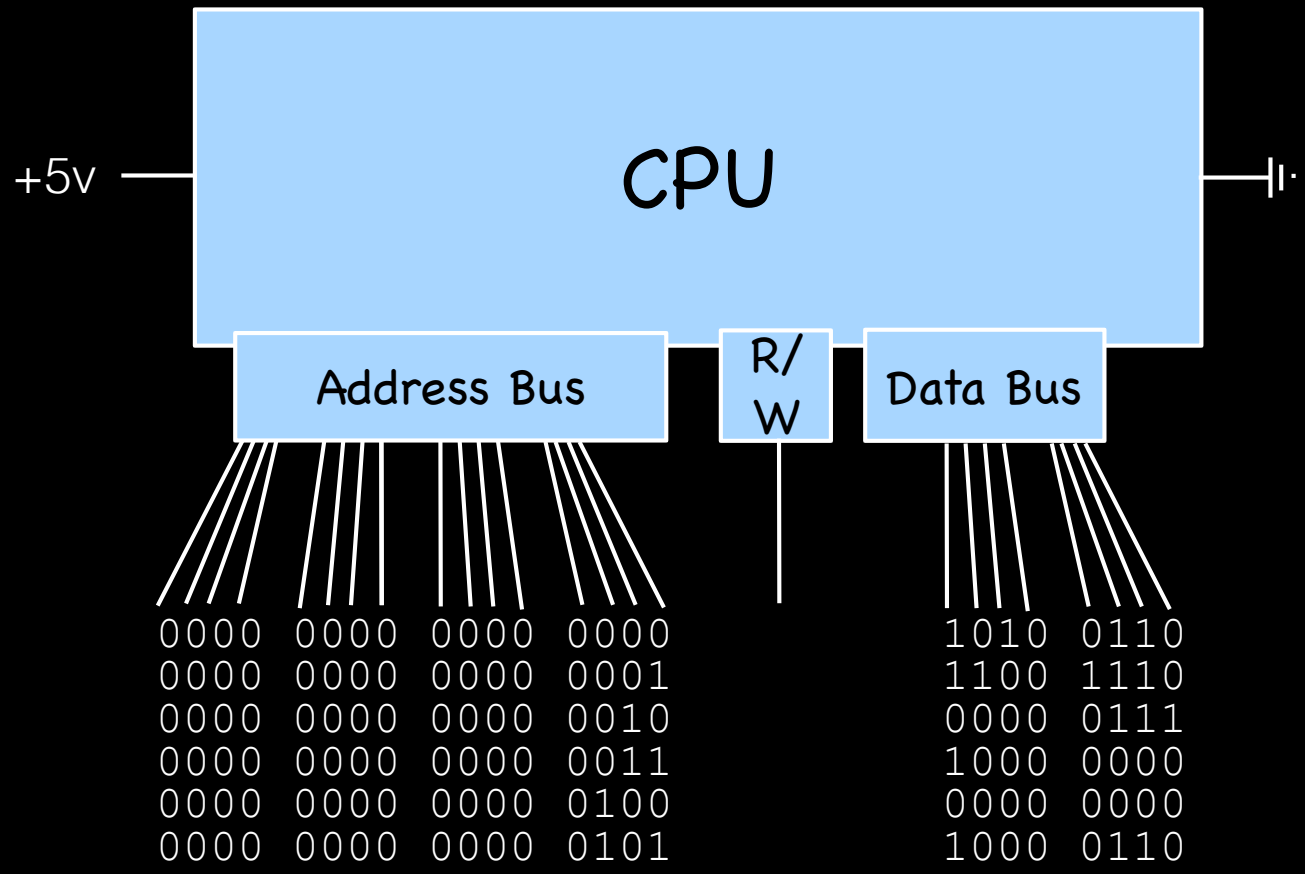
AMD 64 Athlon

Motorola 6800 CPU

- 72 instructions (197 opcodes)
- 8 bit data bus (0-255)
- 16 bit address bus (64k max RAM)
- 6 registers:
  - 8 bit ACCA
  - 8 bit ACCB
  - 16 bit IX
  - 16 bit PC
  - 16 bit SP
  - 6 bit CC



Digital Circuits



6800 Assembly Language

```
33+10/2


Memory:
0000: 1000 0110        LDA
0001: 0010 0001        33                  33 -> ACCA
0002: 1000 1011        ADDA
0003: 0000 1010        10                  43 -> ACCA
0004: 0100 0110        RORA                21 -> ACCA
0005: 1001 0111        STAA
0006: 0000 1010        10                  ACCA -> mem(10)
...
000A: 0001 0101        21
```

Microprocessors

- 6800 ~4000 transistors, 8 bit (1974)
- 68000 ~70,000 transistors, 16/32 bit (1979)
- 68040 ~1,200,000 transistors 32 bit+FPU (1990)
- Core2 duo, 2 core ~291,000,000 transistors 64 bit (2006)
- Xeon E7v2, 15 core ~4,300,000,000 (2014)
- IPhone6 (A8) ~2,000,000,000 transistors (2014)

Microprocessors

• 1977	$666.66	Apple I	0.0004 GIPS
• 1984	$15,000,000	Cray X-MP/48	0.8 GFLOPS
• 2014	$650	iPhone6	~7.5 GFLOPS(CPU)+
			115 GFLOPS(GPU)

+ 2 cameras, light sensor, accelerometer, magnetic field sensor, GPS microphone, two speakers, 4 radios (cell, wifi, bluetooth,GPS)

Shared Clusters at BCM

- Genome Center
- Cancer Center
- CIBR Co-op:
  - 5 clusters (CIBR + 6 PIs)
    - 960+704+640+256+180 = 2740 cores
    - ~24,000,000 CPU-hr/year
    - 350 TB reliable storage
    - 60,000 CPU-hr/qtr free for any CIBR PI

SMP or Distributed

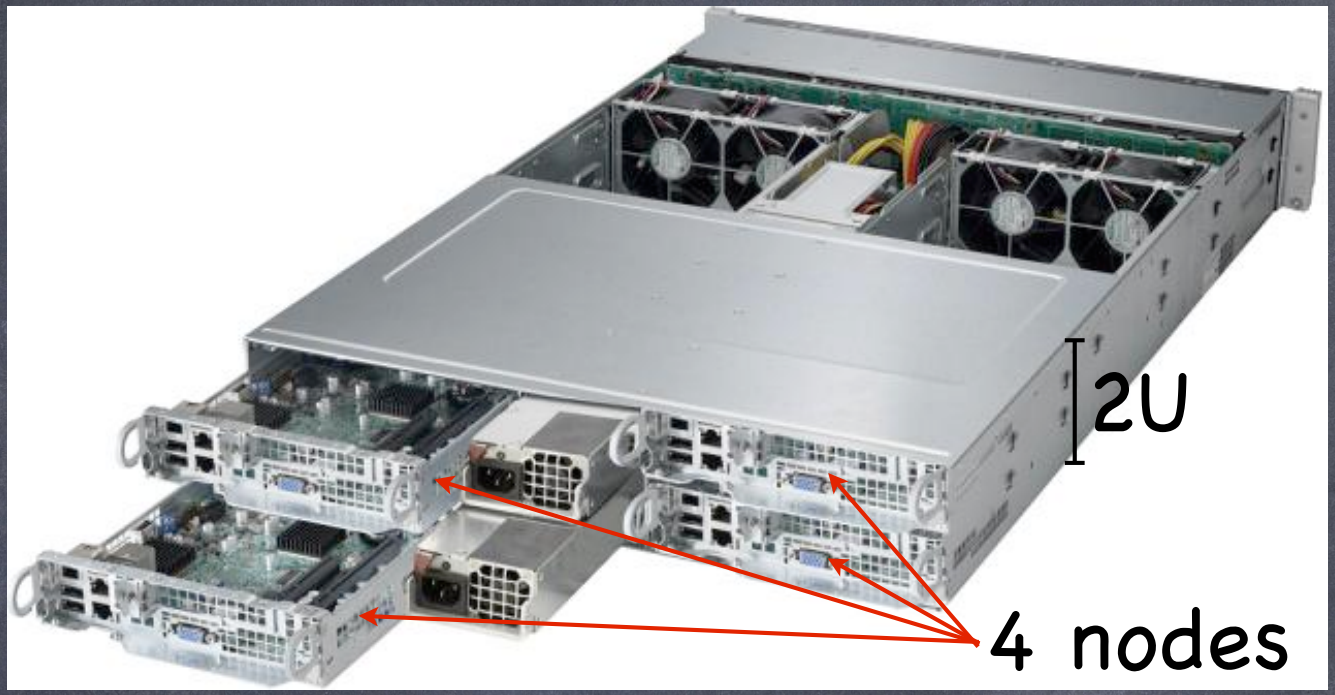
- S 1964, CDC 6600, $60m (2012 $), 500 kFlops, 1 CPU
- S 1977, CRAY 1, $33m, 80 MFlops, 1 CPU
- S 1984, CRAY XMP, $25m, 800 MFlops, 4 CPUs - vector
- D 1987, CM-2, $22m, 6 GFlops, 65,536 CPUs, 2048 MPU
- S 1996, Origin 2000, ~$3m, 10 GFlops, 32 CPUs SMP
- D 2005, Cluster, $0.4m, 900 GFlops, 106 nodes, 212 cores
- ? 2011, Cluster, $0.22m, 5.5 TFlops, 48 nodes, 576 cores
- ! 2015, Tesla K80 GPU, $0.005m, 5.6 TFlops, 1 PCIe board

iPhone4S

iPhone6

Typical Rack 42U

Cluster Hardware



- 1 Chassis:
  - 4 nodes
    - 2 processors/node:
      - 12 cores/processor
    - 128 GB RAM/node
    - 2 TB Hard Drive/node
  - 10 Gb ethernet

- 96 cores
  - 2-4 TFLOPS
  - 68 GB/sec RAM
- 512 GB RAM (5GB/core)

- $26,000 ($270/core)

Cluster Hardware

- 1 Rack:
  - 20 * 2U ->
    - $26,000 * 20 -> $520k + ~$30k (rack, etc.)
    - 20*96 cores -> 1920 cores
      - 40-80 TFLOPS Peak
      - ~30 KW
        - 30 KW * 8700 hr/yr = 260 MWH/yr
        - ~$30,000/yr electric bill
        - A/C bill !?

Comparison of Languages

Loop/Array/Math Benchmark

Language	Time
C++ (-O2)	1
C++ (no opt)	2
Javascript (JIT)	2
Java	5.1
Python	16.5
Perl	24.6
PHP	55.6

Speed Estimates

- 30,000 MIPS - (Million instructions per second) current peak capabilities of a single CPU (with multiple cores)
- 100,000 MB/sec - Level 1 cache memory bandwidth (32 kbytes/core)
- 50,000 MB/sec - Level 2 cache memory bandwidth (256 kbytes/core)
- 35,000 MB/sec - Level 3 cache memory bandwidth (8000 kbytes/CPU)
- 18,000 MB/sec - RAM (typical DDR3 dual channel)
- 8,000 MB/sec - PCIe x16 (2.0)
- 1,500 MB/sec - 12 drive RAID6 with PCIe controller
- 800 MB/sec - QDR Infiniband
- 150 MB/sec - Typical sequential disk read bandwidth for one drive
- 100 MB/sec - Gigabit network

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

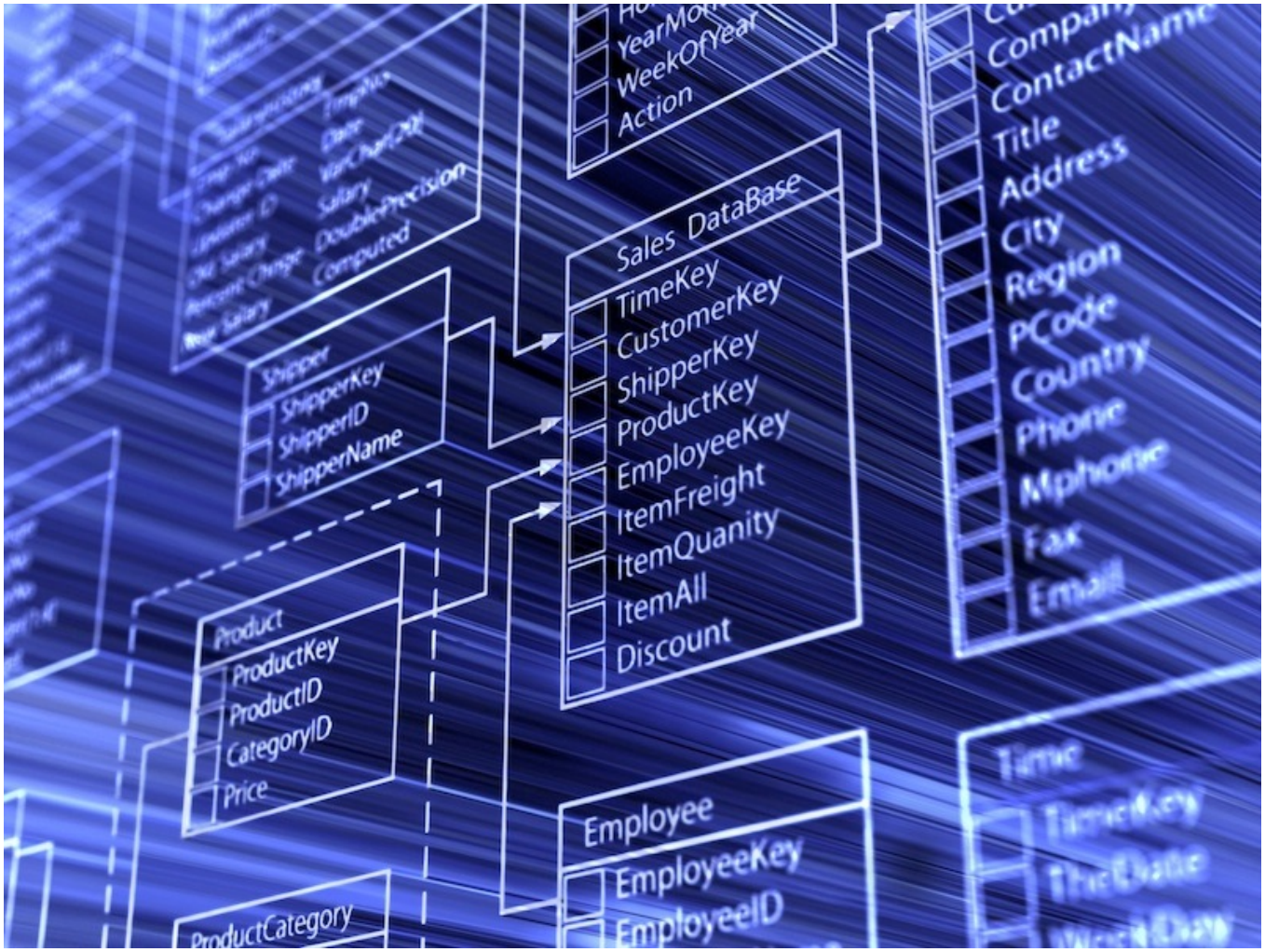
- With gzip, etc.
  - Slooow, 2.8 GB gator genome takes 10 min !
  - No random access of compressed data
- Can we come up with something better ?

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=open("/Users/stevel/Downloads/gator_small.seq","r")
outseq=open("/Users/stevel/Downloads/gator_small.cmp","w")

while True:
    trip=seq.read(3).lower()
    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("/Users/stevel/Downloads/gator_small.seq","r");
    out=fopen("/Users/stevel/Downloads/gator_small.cmp","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
ProductCategory
Time
Action
Title
Address
City
Region
PCode
Country
Phone
Fax

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

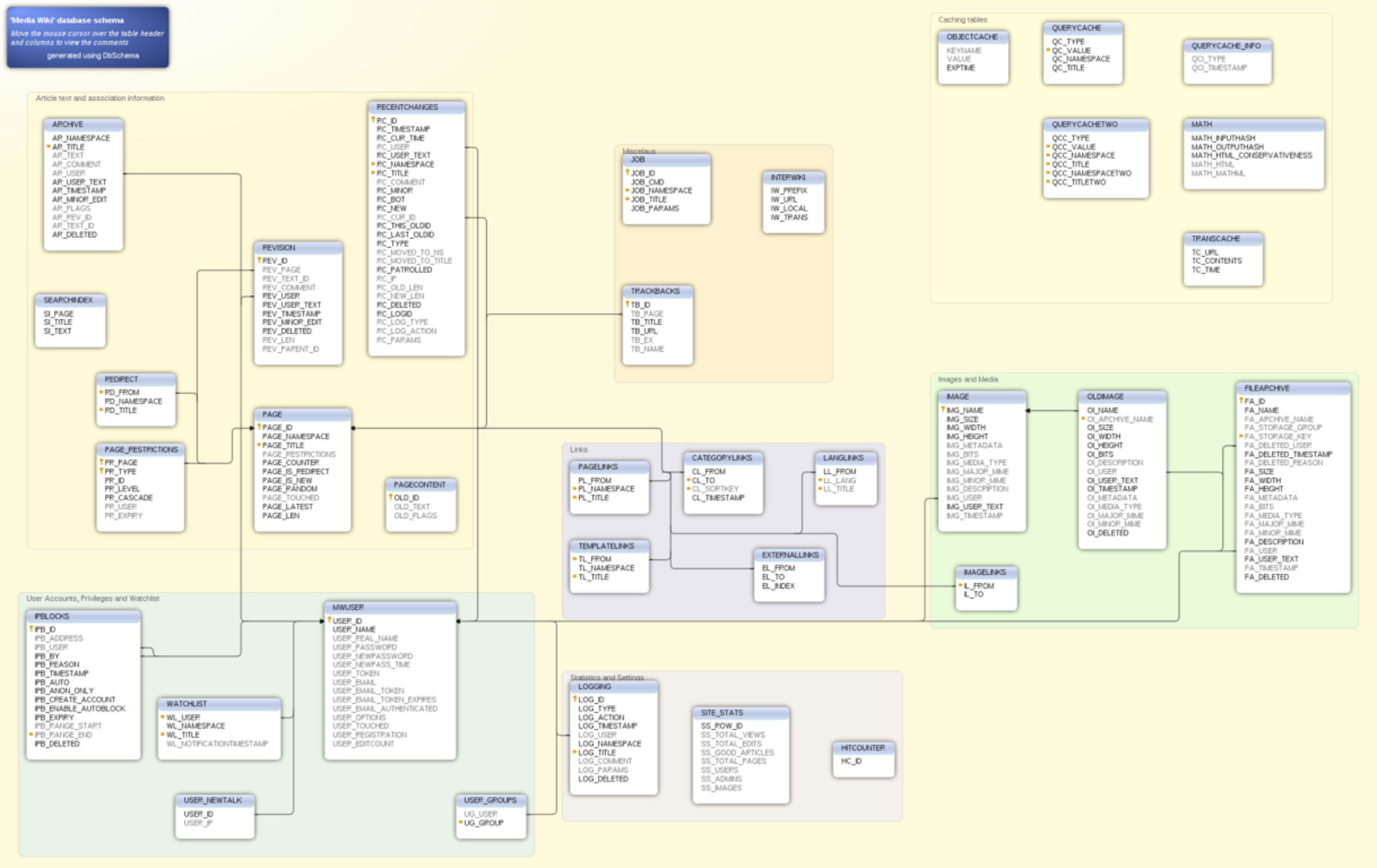
Tables (Relations)

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+

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

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

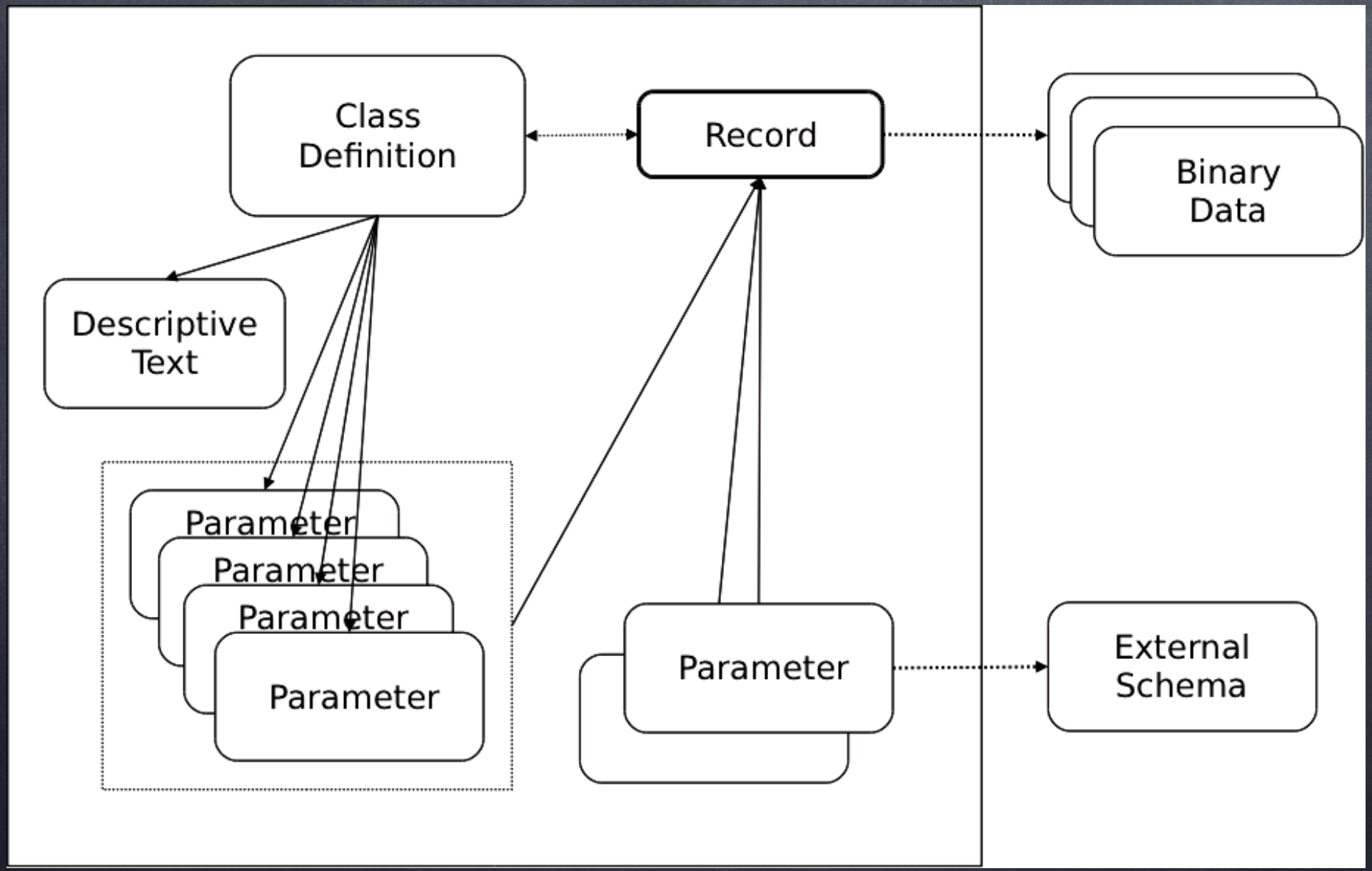
Database Schema



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")

Homework

- Make sure you have an SSH client available on your computer.
  - Mac/Linux, comes with the OS, nothing to do
  - Windows, a number of choices, this one is good:
  - http://www.chiark.greenend.org.uk/~sgtatham/putty/