# Databases

Data at Scale

# Early History of Databases

- Even before the computer age, humanity stored and used data
    - Organization of data is often key to effective functioning of organizations, such as the development of Bureaucracy in Napoleonic France
- 1950s: Computers are first used for commercial purposes
    - 1953: American Airlines and IBM start investigating and working on an airline reservation system
        - 1963: SABRE is fully functional after an effort of 400 man-years
        - 2017: Still going strong as an independent service provider

# Early Databases

- First uses of computers for business purposes were specific to the data
  - Storage medium of data
    - Tape, disks, paper
  - Definition of records
  - Logical and physical arrangement of data

# Data Modeling

- In order to be processed, data needs to be put into schemes so that data items can be found
    - Only now are we getting ready to abandon *structured data*
- Data gains value by the way it can be used
    - Usually, making new uses of data implies a reorganization of data

# Databases

- Data needs to be organized
- The entity/relationship model is one way to represent data graphically
    - Entity sets
        - Formed by abstract objects of some sort
    - Attributes
        - Properties of an entity
    - Relationships
        - Connections between entity sets

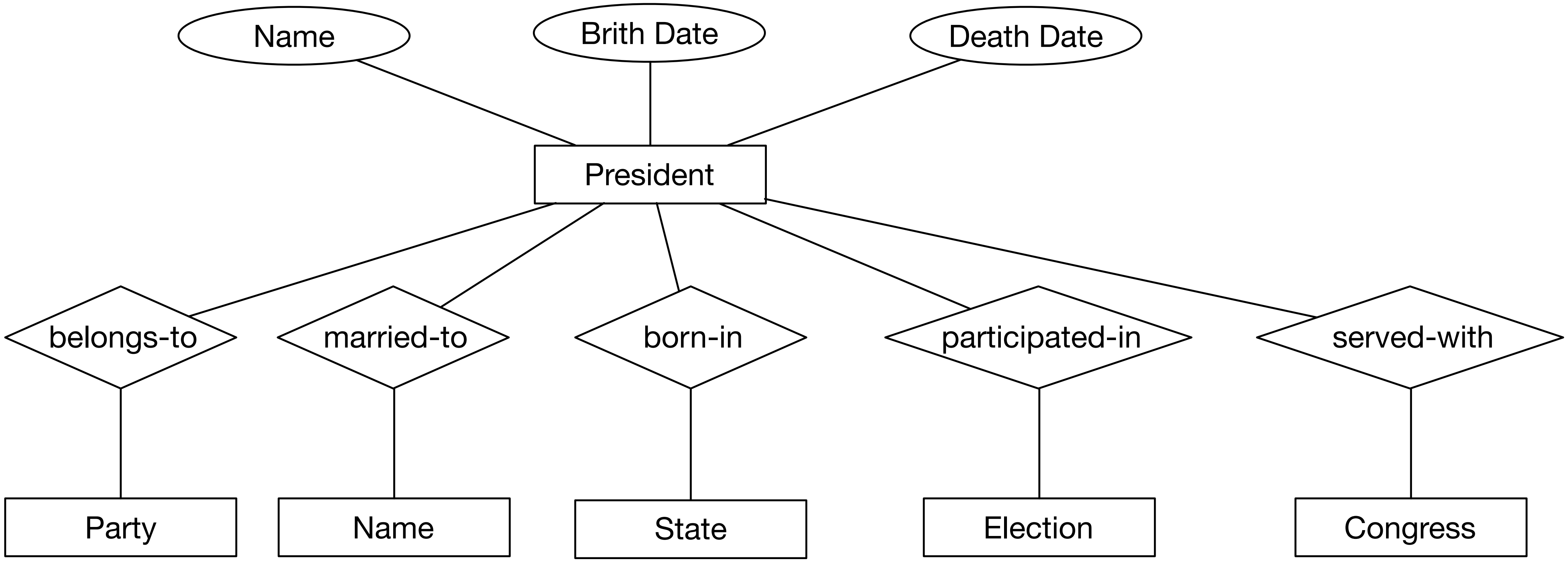
# E/R Model

- E/R diagrams
  - Entity sets are represented by triangles
  - Attributes are represented by ovals
  - Relationships are represented by diamonds
  - Relationships can be
    - one-one
    - one-many
    - many-one

# E/R Model

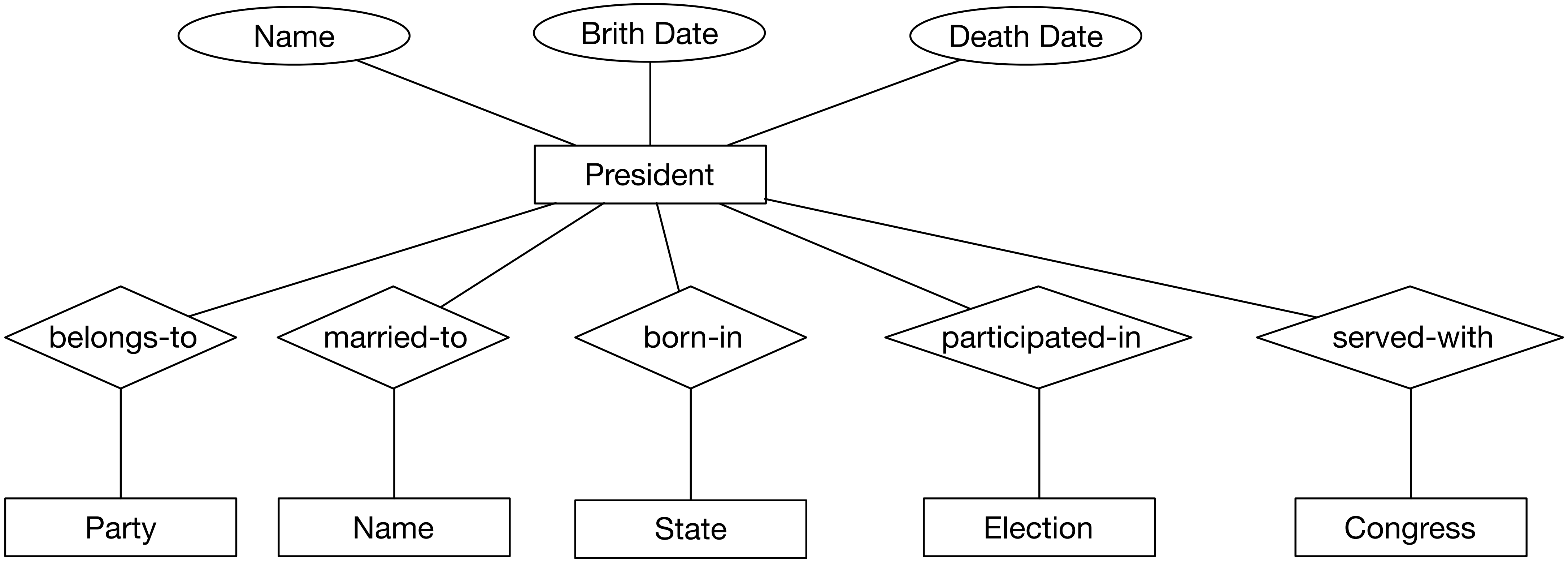
- Example: A presidential database for the last century
  - Inspired by: A. Michaels, B. Mittman, C. Carlson: A Comparison of the Relational and CODASYL Approached to Data-Base Management, ACM Computer Surveys, vol. 8(1), March 1976
- Keywords: Presidents, elections, losers, native-sons, congresses

# Presidents



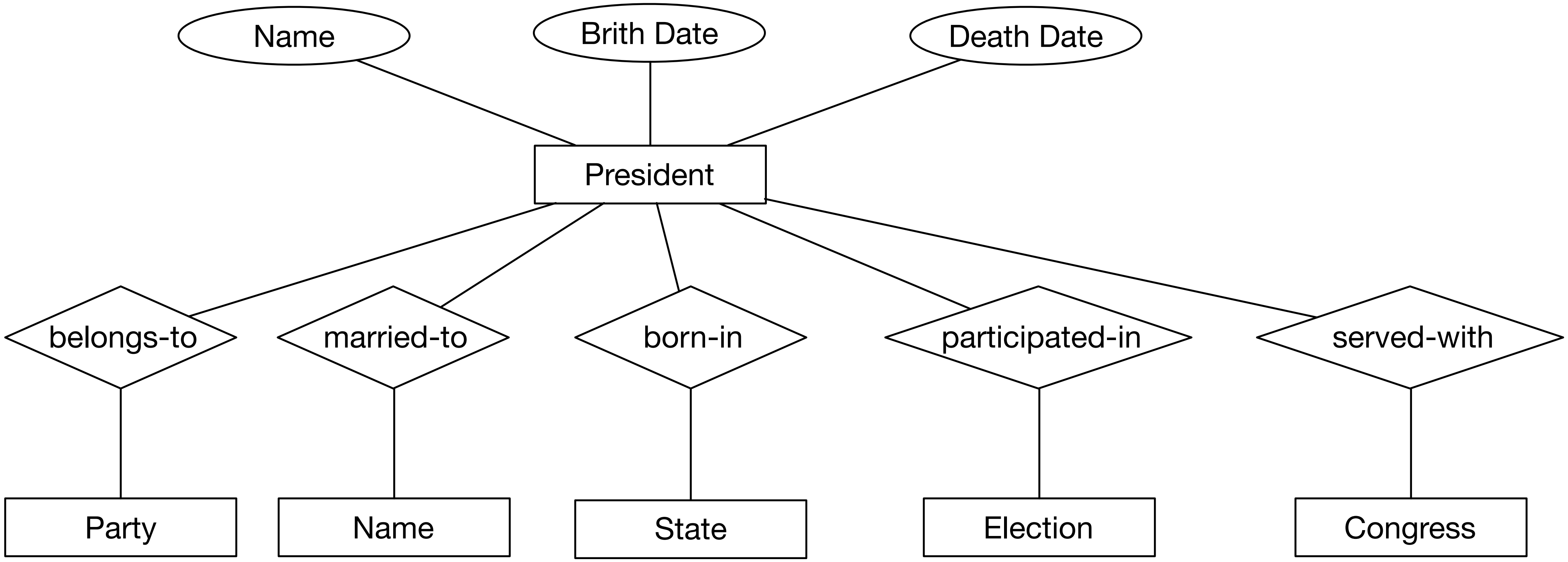
- Start out with the attributes of a president
    - Presidents have a name, a birth date, and a death date, though the latter might still be in the future
    - They belong to parties, but parties cannot be attributes, since some presidents belonged to more than one party.
        - Abraham Lincoln was a Whig who joined the Republican party when it was founded
        - Monroe did not want to belong to a party, but he was not a Federalist

# Presidents



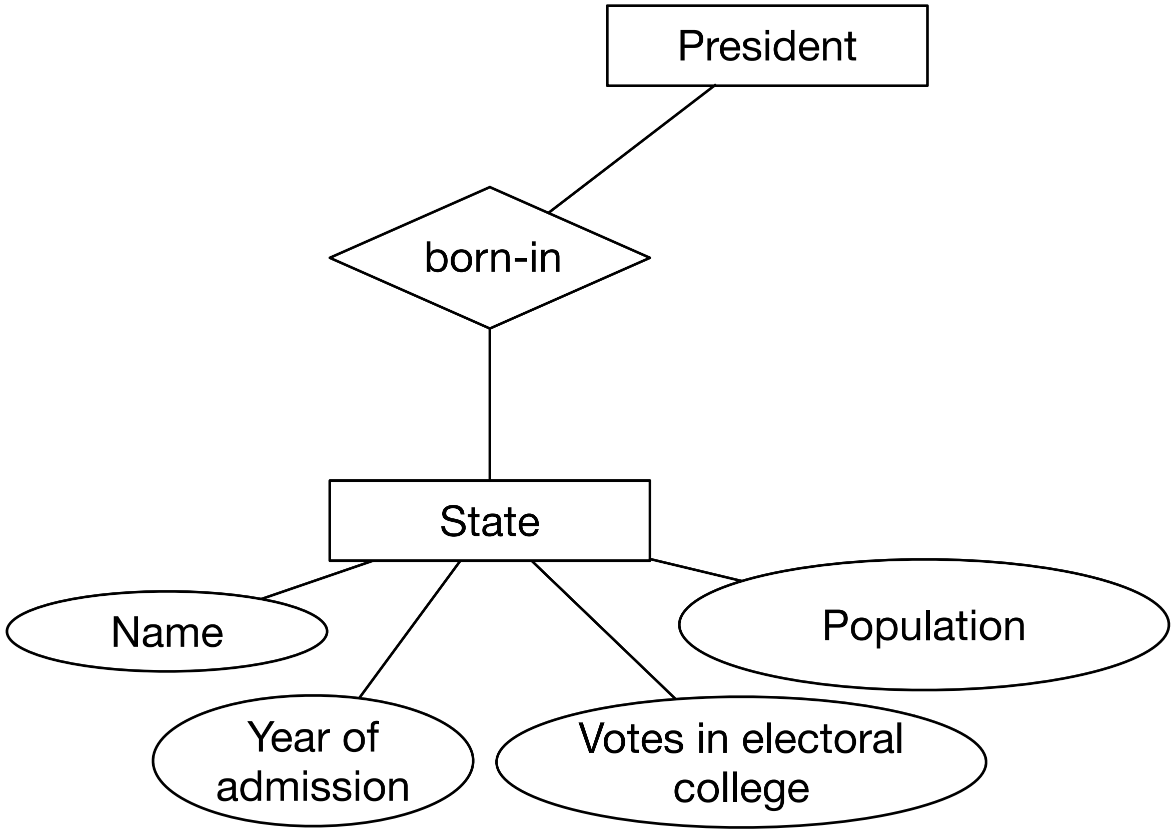
- Start out with the attributes of a president
  - Some presidents where married more than once, so the spouse cannot be an attribute

# Presidents



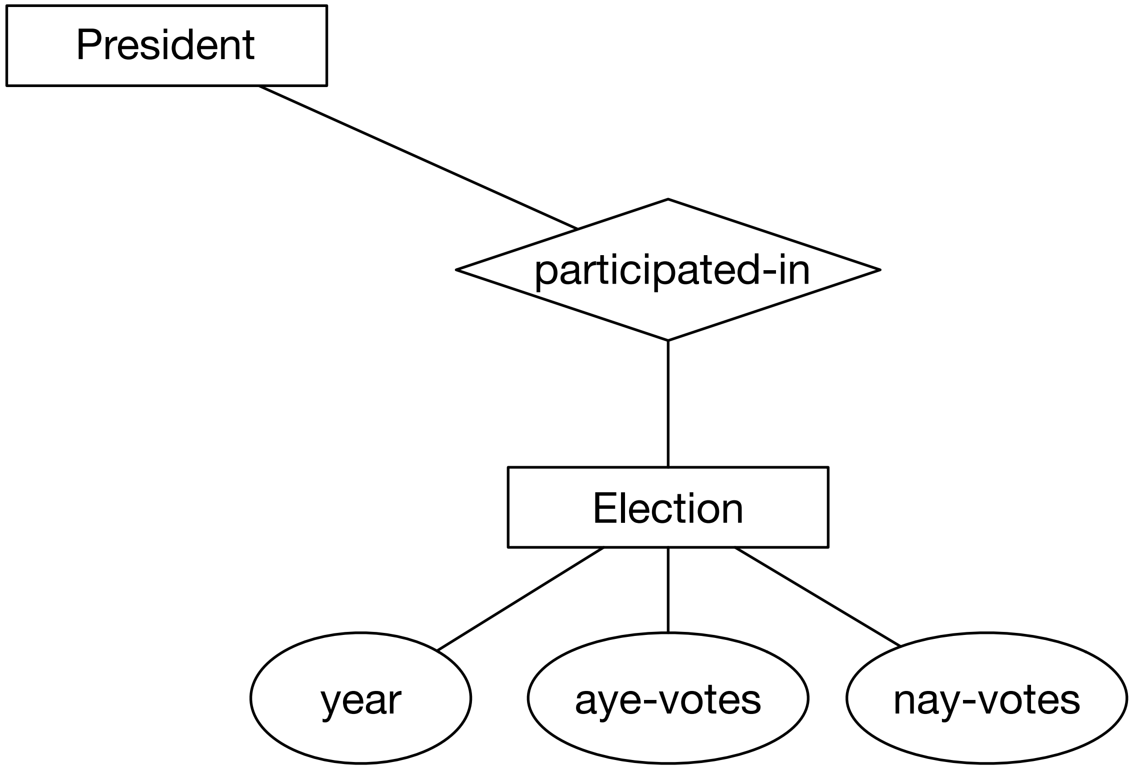
- Relationships are with other entities.
  - We have an anemic entity name that stands for a person
  - By design, we assume that the name of the first lady (first spouse?) is the only thing that we care about.
  - Parties similarly are entities with only one attribute, namely their name
  - States, Elections, and Congresses however are more involved.
  - Even that is complicated: Andrew Jackson’s marriage might have been illegal and therefore void

# Presidents



- The State-of-the-Union part of the E/R model
  - A state has the “born-in” or “native-son” relationship with presidents
  - A state has attributes: her name, the year of admission into the union (whatever that might be for Delaware or Vermont), an official population number, and the votes in the electoral college
  - Strictly speaking, this is not good design since the number of votes in the electoral college and the official population varies with each new census. A reelected president might span two different census and in the case of F. D. Roosevelt, three.

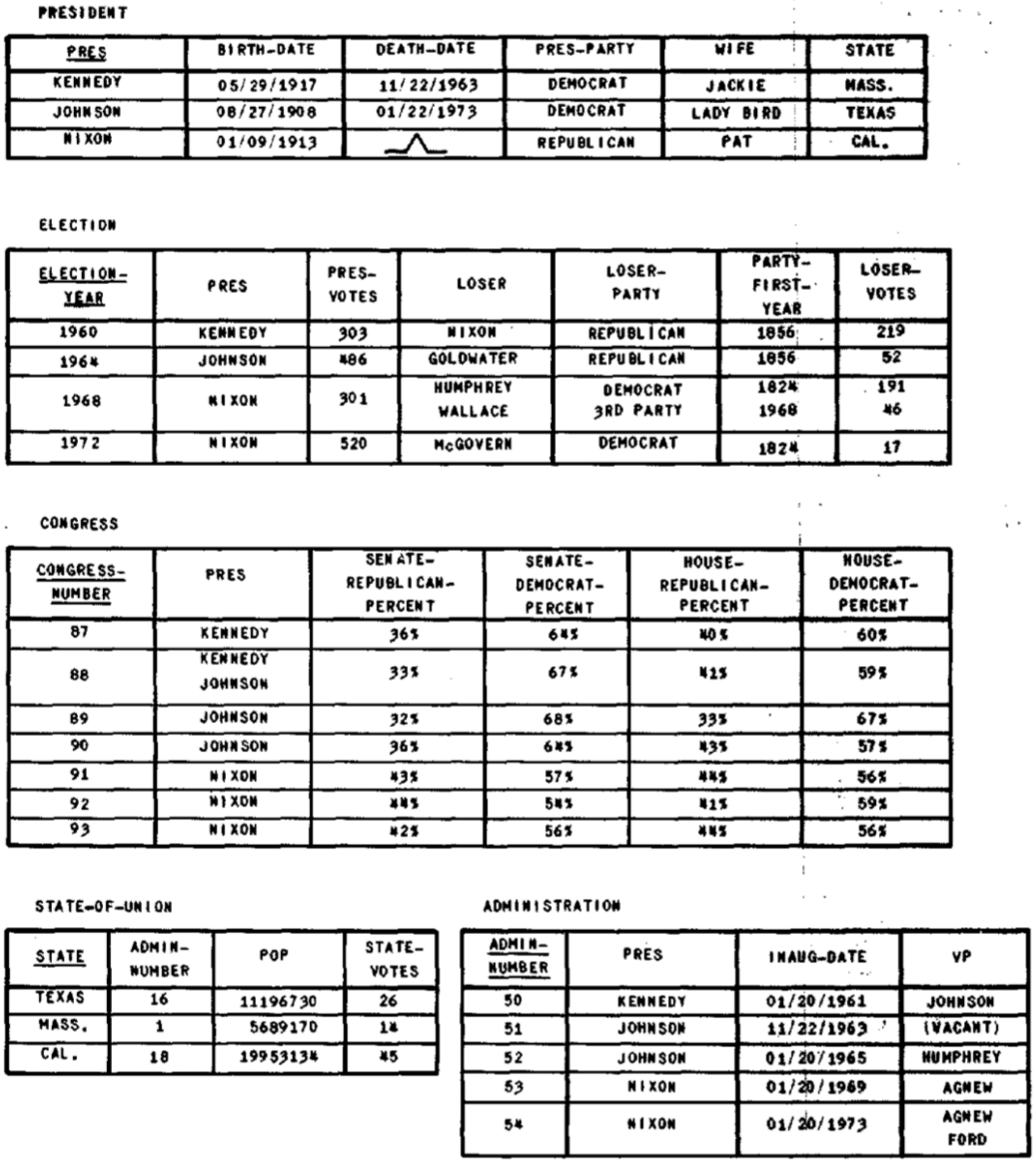
# Presidents



- Elections (in the electoral college) provide even more challenges.
  - Try to extend the data model to give information about the votes of the candidates!
  - In the current model, we only have votes for and votes for other than the winner
  - It would be nice to have the names of people with votes in the college together with their party
  - This is made more complicated because we can have several people. For example, Wallace got 46 electoral votes as a third-party candidate and the 1860 election had four, even though Lincoln garnered a comfortable majority in the college

# Group Task

- Draw an E/R diagram for the data given in Figure 1. However, use your knowledge of US history in order to determine the capacity of your E/R diagram to capture all reasonable past, present, and future presidential elections.



PRESIDENT

| PRES | BIRTH-DATE | DEATH-DATE | PRES-PARTY | WIFE | STATE |
|---|---|---|---|---|---|
| KENNEDY | 05/29/1917 | 11/22/1963 | DEMOCRAT | JACKIE | MASS. |
| JOHNSON | 08/27/1908 | 01/22/1973 | DEMOCRAT | LADY BIRD | TEXAS |
| NIXON | 01/09/1913 | ⋀ | REPUBLICAN | PAT | CAL. |

ELECTION

| ELECTION-YEAR | PRES | PRES-VOTES | LOSER | LOSER-PARTY | PARTY-FIRST-YEAR | LOSER-VOTES |
|---|---|---|---|---|---|---|
| 1960 | KENNEDY | 303 | NIXON | REPUBLICAN | 1856 | 219 |
| 1964 | JOHNSON | 486 | GOLDWATER | REPUBLICAN | 1856 | 52 |
| 1968 | NIXON | 301 | HUMPHREY<br>WALLACE | DEMOCRAT<br>3RD PARTY | 1824<br>1968 | 191<br>46 |
| 1972 | NIXON | 520 | McGOVERN | DEMOCRAT | 1824 | 17 |

CONGRESS

| CONGRESS-NUMBER | PRES | SENATE-REPUBLICAN-PERCENT | SENATE-DEMOCRAT-PERCENT | HOUSE-REPUBLICAN-PERCENT | HOUSE-DEMOCRAT-PERCENT |
|---|---|---|---|---|---|
| 87 | KENNEDY | 36% | 64% | 40% | 60% |
| 88 | KENNEDY<br>JOHNSON | 33% | 67% | 41% | 59% |
| 89 | JOHNSON | 32% | 68% | 33% | 67% |
| 90 | JOHNSON | 36% | 64% | 43% | 57% |
| 91 | NIXON | 43% | 57% | 44% | 56% |
| 92 | NIXON | 44% | 54% | 41% | 59% |
| 93 | NIXON | 42% | 56% | 44% | 56% |

STATE-OF-UNION

| STATE | ADMIN-NUMBER | POP | STATE-VOTES |
|---|---|---|---|
| TEXAS | 16 | 11196730 | 26 |
| MASS. | 1 | 5689170 | 14 |
| CAL. | 18 | 19953134 | 45 |

ADMINISTRATION

| ADMIN-NUMBER | PRES | INAUG-DATE | VP |
|---|---|---|---|
| 50 | KENNEDY | 01/20/1961 | JOHNSON |
| 51 | JOHNSON | 11/22/1963 | (VACANT) |
| 52 | JOHNSON | 01/20/1965 | HUMPHREY |
| 53 | NIXON | 01/20/1969 | AGNEW |
| 54 | NIXON | 01/20/1973 | AGNEW<br>FORD |

Figure 1. Sample presidential data base.

# Databases

- Databases need to
  - allow users to retrieve and modify data
    - Users have different capacities for programming, so a simpler model is needed
    - For performance reasons, this needs to be done in parallel
  - allow database administrators to change the physical and logical layout of the data (for performance tuning)
  - provide safety guarantees
    - Access control for users
    - Checks to find implausible updates
    - Allow data to be hidden from the user
    - Allow surviving system crashes and hardware / software failures without dataloss

# Cautionary Aside

- A cautionary tale about mixing levels
    - IBM invented the hard drive
        - IBM 305 RAMAC computer system announced September 13, 1956
    - Decided on a block size of 512B
        - Very reasonable but now replaced with 4KB blocks
    - Noticed that hard drives were often used for what we now call dictionary look-ups
        - Key is small number of bytes, value is contained in a block
    - Decided to offer disks that had an additional 8B key
    - Feature was never really used, but meant that for compatibility, all IBM drives had to have 520B blocks
    - So, IBM disks only had 512/520 of their physical capacity (some 1.6%)
- Moral: Be careful where in a hierarchy you are optimizing

# Database Organizations

- 1970 — Existing approaches:
    - A hierarchical model of data organization — IBM Information Management System (released 1960)
        - Data is organized in a tree and access goes from top to bottom
            - Administration —> President —> State —> Population
        - Works well with one-one and one-many relationships
        - Based on how data would be stored
        - Access is by programming navigation in a tree
    - Security, transactions, etc. are difficult to program

# Database Organizations

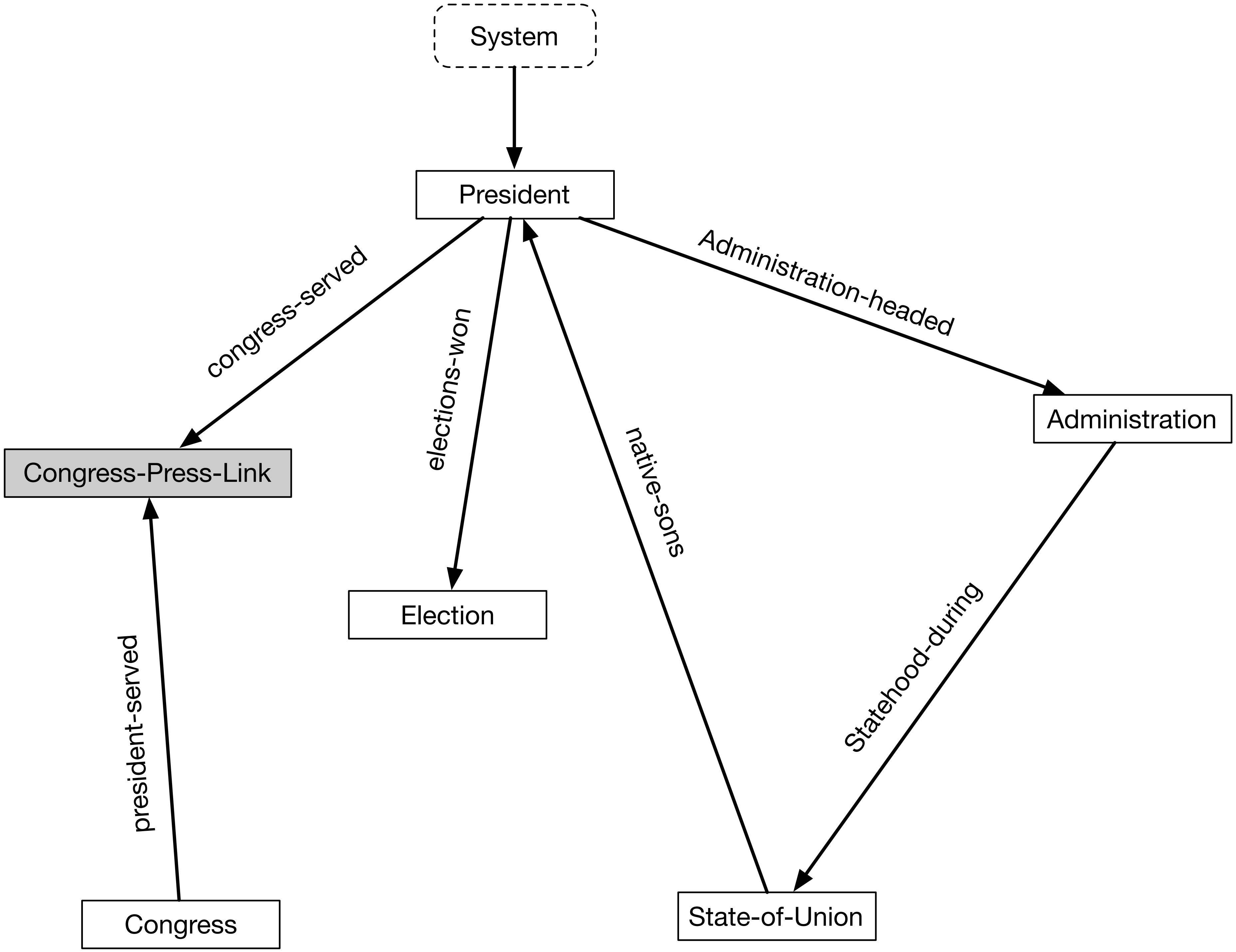
- 1970 — Existing approaches:
  - CoDaSyL: Conference/Committee on Data Systems Languages formed 1959
  - Develops the network model for data as well as query languages and data definition languages

# Database Organization

- Network model
  - Has Records and Sets
  - Plus an entry into the system, called System

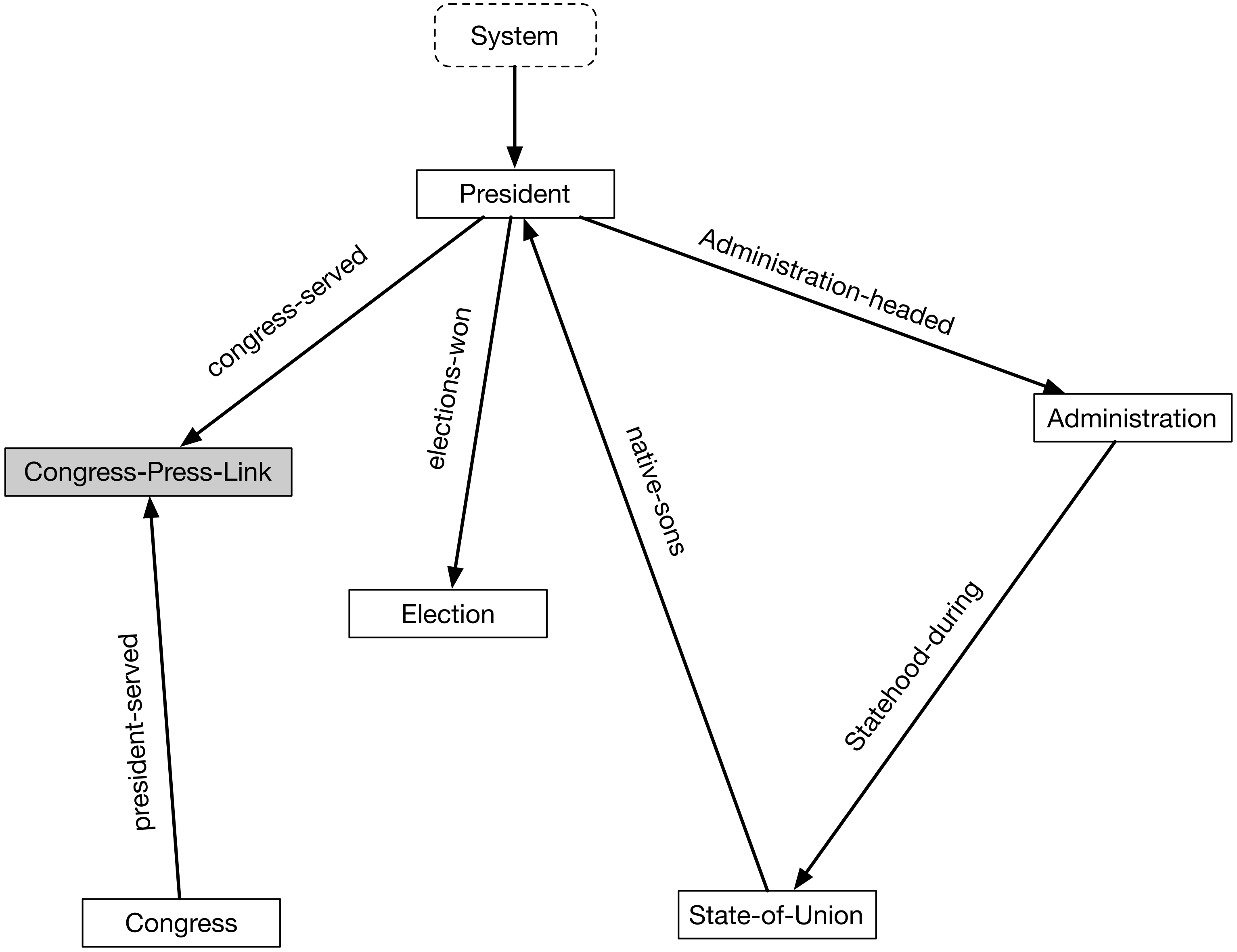
# Database Organization

- Network model example



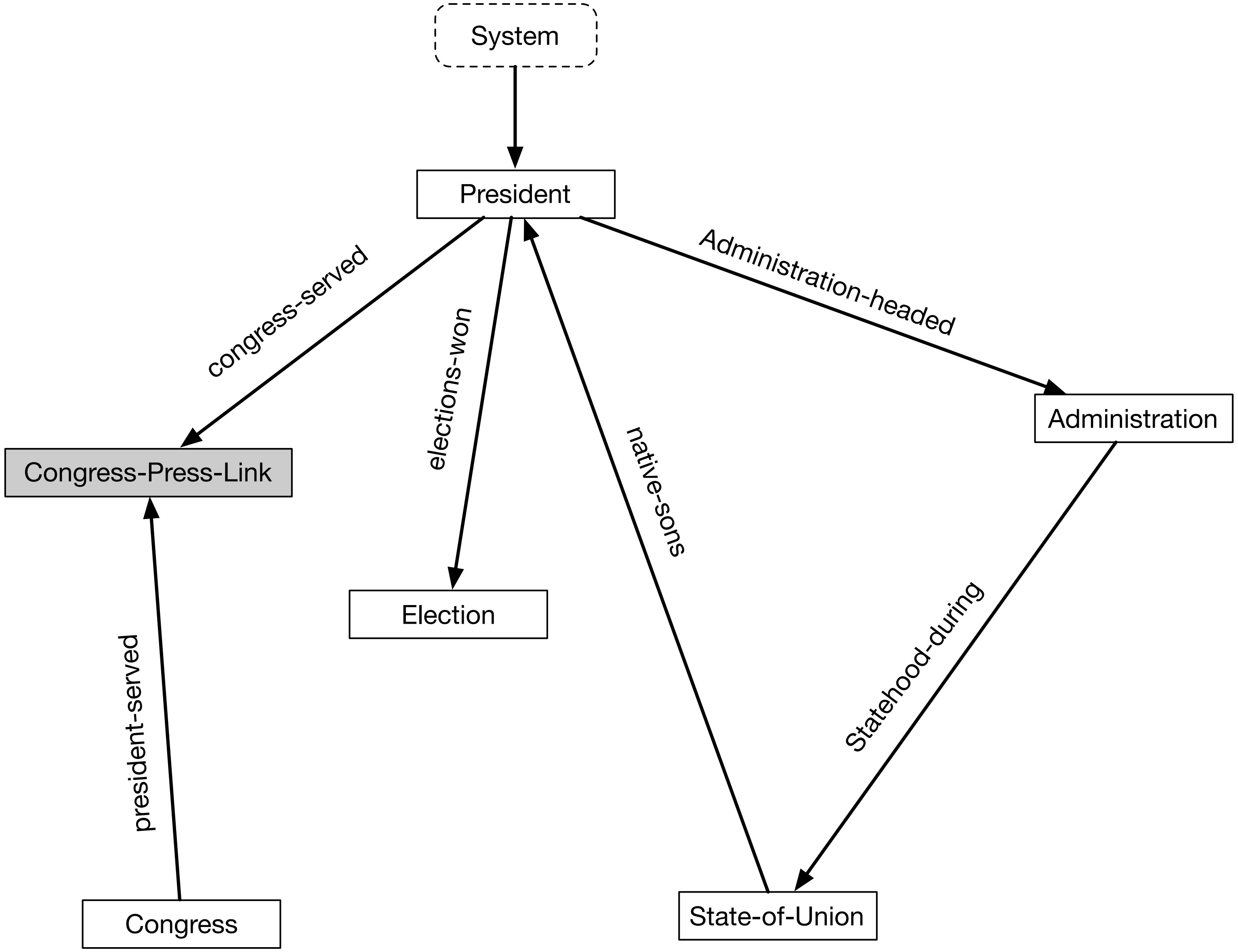
# Database Organization

- Network model example
  - Records are for president, administration, election, congress, state
  - Sets are represented by arrows
  - E.g. a president record is associated with a set of elections that the president has won
  - Sets allow us to represent one-one and one-many relationships
  - Not shown are the attributes of the records



# Database Organization

- Network model example
  - Presidents can change during a congress (murder of Kennedy and resignation of Nixon)
  - A president usually serves with several congresses
  - To model this many-many relationship, a special record needs to be invented, the Congress-Press-Link record



# Database Organization

- We can observe that the hierarchical and less the network model of databases is tied to the logical organization of data access
- Network model based databases were commercially successful
- In order to allow untrained or untrainable users to interact with them, manipulation mechanisms became more sophisticated
- Network model based databases still have problems with parallelism and record protection

# Relational Databases

- E. F. Codd (IBM, San José, CA) proposed relational databases in a 1970 paper
- Pressured IBM into developing System R, with a non-relational access language called Sequel
- Based on preprints of papers, Ellison founded Oracle, with a similar language called SQL

# Relational Databases

- A database need to:
    - Give correct answers to queries
        - Expressability
            - What queries are supported
        - Maintainability
            - Needs to support transactions:
                - Atomicity Consistency Isolation and Durability
    - User friendly

# Relational Databases

- Transactions
  - Atomicity
    - A transaction can be rolled back
  - Consistency
    - A transaction transforms a database from one valid state to another valid state
  - Isolation
    - A transaction is invisible to others until it commits
  - Durability
    - Once committed, the results are permanent and survive system and media failures

# Relational Databases

"Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). ... Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

"Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model."

Codd: ”A Relational Model of Data for Large Shared Data Banks”, CACM 1970

https://dl.acm.org/citation.cfm?id=362685

# Relational Databases

- Data is stored as tuples.
  - A tuple is an array of values.
  - Each coordinate is an *attribute*
- Customary to present tuples as rows in a matrix

| title | year | length | genre |
|---|---|---|---|
| Gone with the wind | 1939 | 231 | drama |
| Star wars | 1977 | 124 | SciFi |
| Wayne’s World | 1992 | 95 | comedy |

# Relational Databases

- Columns are called attributes
- Name and the set of attributes are called the *scheme*
  - `Movies(title, year, length, genre)`
- Entries are called tuples
- Strict relational model requires that all attributes are atomic: an elementary type
  - `Movies(title:str, year:int, length:int, genre:str)`

# Relational Databases

- Relational databases change over time through inserts and deletions
    - The state of a database at one time is called the *current instance*

# Relational Databases

- Keys:
  - Relations are not ordered
    - To allow fast access, need indices
  - Information represented by the data also needs to be coherent
    - A change in the information should result in a single update to a tuple
    - Otherwise, programming errors are likely to render the information incoherent

# Relational Databases

- Notation of keys supports both
  - Artificial keys:  an auto-generated ID that characterizes each tuple uniquely
  - A or a combination of attributes that are unique to the tuple (for all eternity)
    - Movie database example:
      - Title is not sufficient, there were two King Kong movies
      - Underline keys in a scheme
      - $\texttt{Movies(}\underline{\texttt{title}}\texttt{, }\underline{\texttt{year}}\texttt{, length, genre)}$

# Relational Databases

- Group Exercise
  - Create schemes for a Movie Database with relations
    - Movies, MovieStar, MovieExec
  - that allows us to answer such question as :
    - Which studios did John Wayne work for
    - Who was responsible for hiring John Wayne
    - What was the first year in which John Wayne stared?

# SQL Statements

- SQL Data Definition Sublanguage
  - Stored relations —> tables
  - Relations defined by computation —> views
  - Relations defined during computation —> temporary tables
    - These are accessible through nested SQL query statements, but are not explicitly defined

# SQL Statements

- SQL implementations differ in the exact types such as for data and time, but their expressibility is about the same
- Defining a table:

```
 CREATE TABLE Movies(
    title            CHAR(100),
    year             INT,
    genre            CHAR(10),
    studioName       CHAR(30),
    producerC#       INT
)
```

# SQL Statements

- Drop a table

```
DROP TABLE movies;
```

- Add an attribute

```
ALTER TABLE moviestar ADD phone CHAR(16)
```

- Drop an attribute

```
ALTER TABLE moviestar DROP birthdate
```

# SQL Statements

- Can use default values

```
…
gender CHAR(1) DEFAULT ‘?”’,
birthday DATA DEFAULT DATE ‘0000-00-00’
…
```

# SQL Statements

- Can declare an attribute to be unique or a primary key
  - Primary keys are used for indexing tuples
  - Lookup using primary keys is then particularly fast

```
CREATE TABLE moviestar (
   name CHAR(30) PRIMARY KEY,
   address VARCHAR(255),
   gender CHAR(1) DEFAULT ‘?’,
   birthdate DATE
);
```

# SQL Statements

- Alternative declaration

```
CREATE TABLE moviestar (
   name CHAR(30),
   address VARCHAR(255),
   gender CHAR(1) DEFAULT ‘?’,
   birthdate DATE,
   PRIMARY KEY (name)
);
```

# SQL Statements

- Composite keys

```
CREATE TABLE movies(
   title                    CHAR(100),
   year                     INT,
   length                   INT,
   genre                    CHAR(10),
   studioname               CHAR(30),
   producerC#               INT,
   PRIMARY KEY  (title, year)
);
```

# Relational Databases

- Query languages
  - Less powerful than general purpose HL programming languages
    - Is the number of tuples in a table even or odd?
  - Easier to program and the ability to produce highly optimized code for execution
  - Typically an interface to relational algebra

# Relational Databases

- Set operations on set of tuples
  - , $R \cup S$ $R \cap S$ $R - S$
  - To apply, tuples need to have the *same* attributes in the *same* order

# Relational Databases

- Selection

Relation Sells:

| bar | beer | price |
|---|---|---|
| Joe’s | Bud | 2.50 |
| Joe’s | Miller | 2.75 |
| Sue’s | Bud | 2.50 |
| Sue’s | Miller | 3.00 |

JoeMenu := $\sigma_{bar=\text{“Joe’s”}}$(Sells):

| bar | beer | price |
|---|---|---|
| Joe’s | Bud | 2.50 |
| Joe’s | Miller | 2.75 |

# Relational Databases

- Projection

Relation Sells:

| bar | beer | price |
|---|---|---|
| Joe’s | Bud | 2.50 |
| Joe’s | Miller | 2.75 |
| Sue’s | Bud | 2.50 |
| Sue’s | Miller | 3.00 |

Prices := $\pi_{beer,price}$(Sells):

| beer | price |
|---|---|
| Bud | 2.50 |
| Miller | 2.75 |
| Miller | 3.00 |

# Relational Databases

- Extended projection
  - Can define a new attribute:  C := A+B
  - Can duplicate attributes

R = (

| A | B |
|---|---|
| 1 | 2 |
| 3 | 4 |

)

$\pi_{A+B\text{->}C,A,A}$ (R) =

| C | A1 | A2 |
|---|---|---|
| 3 | 1 | 1 |
| 7 | 3 | 3 |

# Relational Databases

- Product
    - Pair each tuple t1 of R1 with each tuple t2 of R2
    - Concatenate to obtain tuple t1t2
    - Schema of result is the attributes of R1 and then R2 in order
    - If an attribute appears in the schemes of R1 and R2, need to disambigue

# Relational Databases

- Example

R1(

| A, | B |
|---|---|
| 1 | 2 |
| 3 | 4 |

)

R2(

| B, | C |
|---|---|
| 5 | 6 |
| 7 | 8 |
| 9 | 10 |

)

R3(

| A, | R1.B, | R2.B, | C |
|---|---|---|---|
| 1 | 2 | 5 | 6 |
| 1 | 2 | 7 | 8 |
| 1 | 2 | 9 | 10 |
| 3 | 4 | 5 | 6 |
| 3 | 4 | 7 | 8 |
| 3 | 4 | 9 | 10 |

)

# Relational Databases

- Theta Join $R_3 = R_1 \bowtie_c R_2$
  - Take the product $R_1 \times R_2$
  - Then apply $\sigma_c$ to the result

- Traditionally, only operators allowed were of the form

$$A \ \theta \ B$$

  - where $\theta \in \{ = , < \leq , > , \geq \}$

# Relational Databases

Sells(

| bar, | beer, | price |
|---|---|---|
| Joe's | Bud | 2.50 |
| Joe's | Miller | 2.75 |
| Sue's | Bud | 2.50 |
| Sue's | Coors | 3.00 |

)

Bars(

| name, | addr |
|---|---|
| Joe's | Maple St. |
| Sue's | River Rd. |

)

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo(

| bar, | beer, | price, | name, | addr |
|---|---|---|---|---|
| Joe's | Bud | 2.50 | Joe's | Maple St. |
| Joe's | Miller | 2.75 | Joe's | Maple St. |
| Sue's | Bud | 2.50 | Sue's | River Rd. |
| Sue's | Coors | 3.00 | Sue's | River Rd. |

)

# Relational Databases

- Natural join
    - Subset where *c* equates all attributes of the same name
    - $R_3 = R_1 \bowtie R_2$

# Relational Databases

Sells( )

| bar, | beer, | price |
|---|---|---|
| Joe's | Bud | 2.50 |
| Joe's | Miller | 2.75 |
| Sue's | Bud | 2.50 |
| Sue's | Coors | 3.00 |

Bars( )

| name, | addr |
|---|---|
| Joe's | Maple St. |
| Sue's | River Rd. |

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo( )

| bar, | beer, | price, | name, | addr |
|---|---|---|---|---|
| Joe's | Bud | 2.50 | Joe's | Maple St. |
| Joe's | Miller | 2.75 | Joe's | Maple St. |
| Sue's | Bud | 2.50 | Sue's | River Rd. |
| Sue's | Coors | 3.00 | Sue's | River Rd. |

# Relational Databases

- Renaming
  - $\rho$ -operator gives a new scheme to a relation
  - $R_1 = \rho_{R_1(A_1,\ldots A_n)}(R_2)$ makes R1 a relation with attributes A1, … An and the same tuples as in R2

# Relational Databases

Bars(

| name, | addr |
|---|---|
| Joe’s | Maple St. |
| Sue’s | River Rd. |

)

R(bar, addr) := Bars

R(

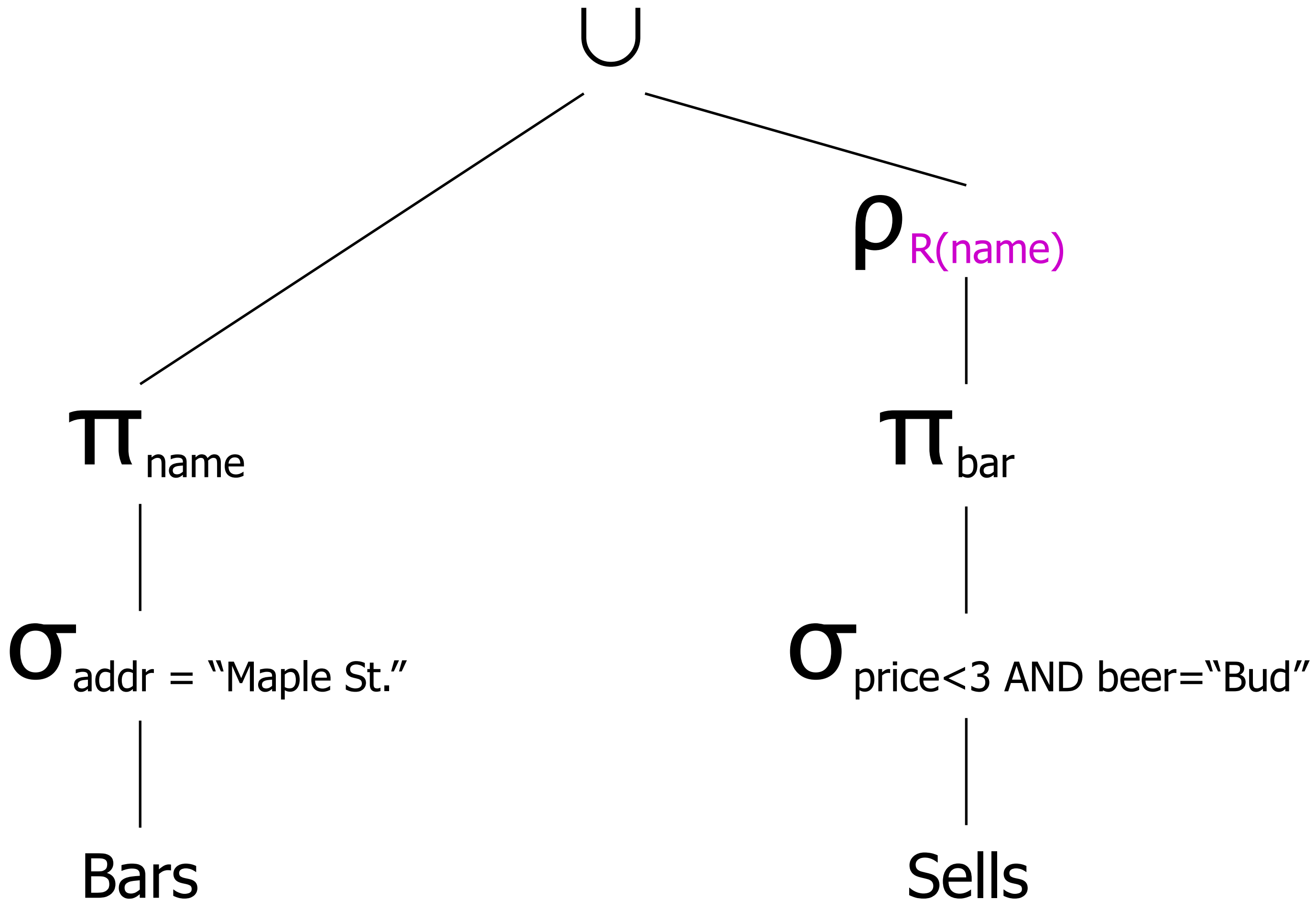
| bar, | addr |
|---|---|
| Joe’s | Maple St. |
| Sue’s | River Rd. |

)

# Relational Databases

- Using the relations Bars(name, addr) and Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3.

# Relational Databases



$\cup$

$\rho_{R(name)}$

$\pi_{name}$

$\pi_{bar}$

$\sigma_{addr\ =\ \text{“Maple St.”}}$

$\sigma_{price<3\ \text{AND}\ beer=\text{“Bud”}}$

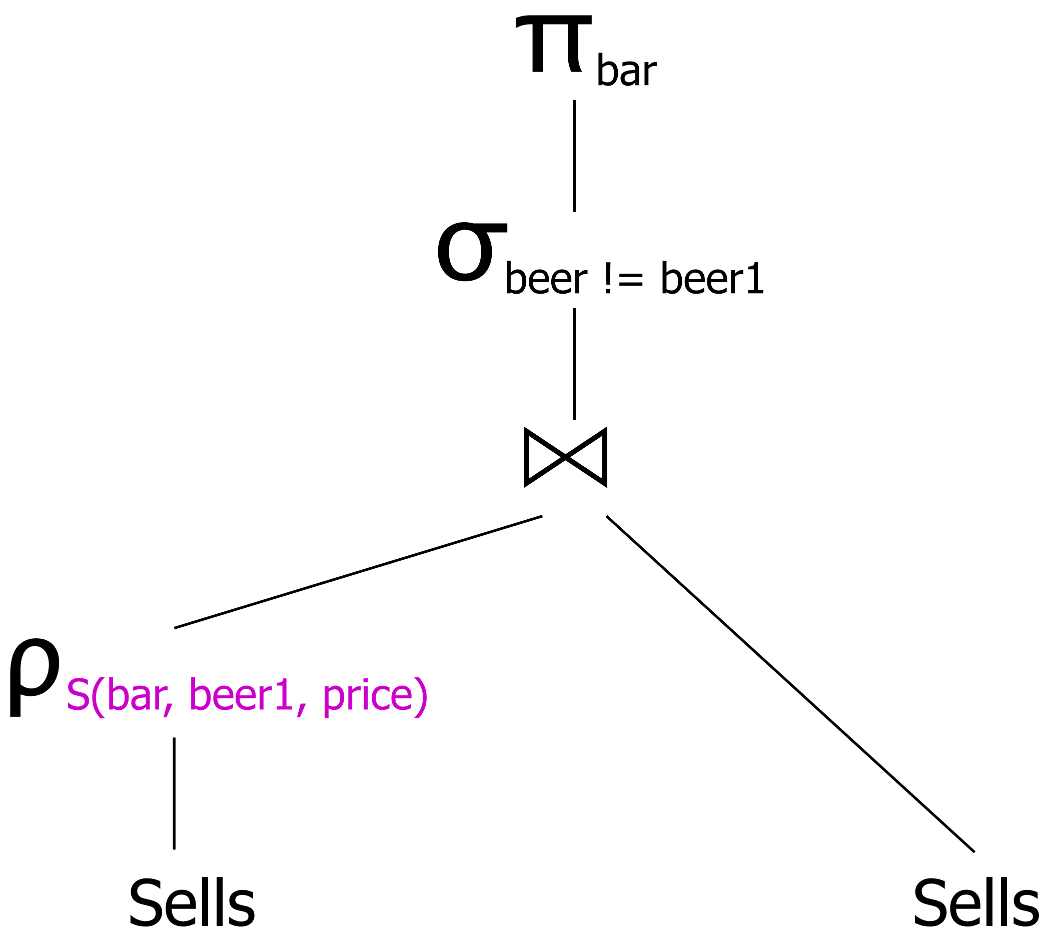
Bars

Sells

# Relational Databases

- Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.
- Strategy: by renaming, define a copy of Sells, called S(bar, beer1, price). The natural join of Sells and S consists of quadruples (bar, beer, beer1, price) such that the bar sells both beers at this price.

# Relational Databases



$\pi_{\text{bar}}$

$\sigma_{\text{beer != beer1}}$

$\bowtie$

$\rho_{\text{S(bar, beer1, price)}}$

Sells

Sells

# Relational Database Design

- Not all set of schemes for information are created equally
    - Good design makes it difficult to create a database with contradictory information

# Relational Database Design

- Functional dependencies
  - A1, A2, … , An —> B1, B2, … , Bm
  - If two tuples have the same values for A1, A2, … An
    - Then they have the same value for B1
    - Then they have the same value for B2
    - …
    - Then they have the same value for Bm

# Relational Database Design

- Movie database
  - `Movies1(title, year, length, genre, studioName, starName)`
  - `title year —> length, genre, studioName, starName` is True
  - `title year —> starName` is False

# Relational Database Design

- Formal definition of a key
    - `R(A1, A2, … , Am)`
    - `B1, B2, …, Bn` is a superkey if
        - `B1 B2 … Bn -> A1 A2 … Am`
    - `B1, B2, …, Bn` is a key if no true subset is a superkey

# Relational Database Design Anomalies

- Redundancy Anomaly: information is repeated in various tuples
  - `Movies1(title, year, length, genre, studioName, starName)`
  - The length of star wars is repeated information

# Relational Database Design

- Update Anomaly
  - If information in one tuple is changed, it might need to be changed in many other tuples

    - Discover that star wars is really 129 minutes long.
    - Change it in one tuple but not in another
    - Information is no longer coherent

# Relational Database Design

- Deletion anomaly
  - If a set of values becomes empty, we might loose other information as well
  - Remove Vivian Leigh as star from Gone with the Wind:
    - No more information on Gone with the Wind survives if she was the only star

# Relational Database Design

- Dealing with these anomalies
  - Decompose relations
  - `Movies1(title, year, length, genre, studioName, starName)`
  - becomes
  - `movies2(title, year, length, genre, studioName)`
  - `movies3(title, year, starName)`

# Relational Database Design

- Notice that we cannot prevent repeating information that a certain movie was made in a certain year
  - Only title, year is a key
- We need to repeat this information in order to disambiguate movies with the same title

# Relational Database Design

- Boyce - Codd Normal Form
    - Simple condition to prevent all anomalies
    - In any functional dependency
        - A1 A2 … An —> B1 … Bm
    - A1 A2 … An is a superkey

# Relational Database Design

- Example:
  - MovieExec(title, year, studioName, president, presAddr)
- has dependencies
  - title year —> studioName
  - studioName —> president
  - president —> presAddr
- But only title year is a key

# Relational Database Design

- Need to decompose FD
  - `(title, year, studioName), (studioName, president, presAddr)`
- Second table still not in Boyce-Codd NF
  - Decompose into
  - `(title, year, studioName), (studioName, president), (president, presAddr)`
    -

# Relational Database Design

- Decomposition yields
    - Elimination of anomalies
    - Recoverability of information: original data can be recovered
    - Preservation of Functional Dependencies
        - This is unfortunately not always given