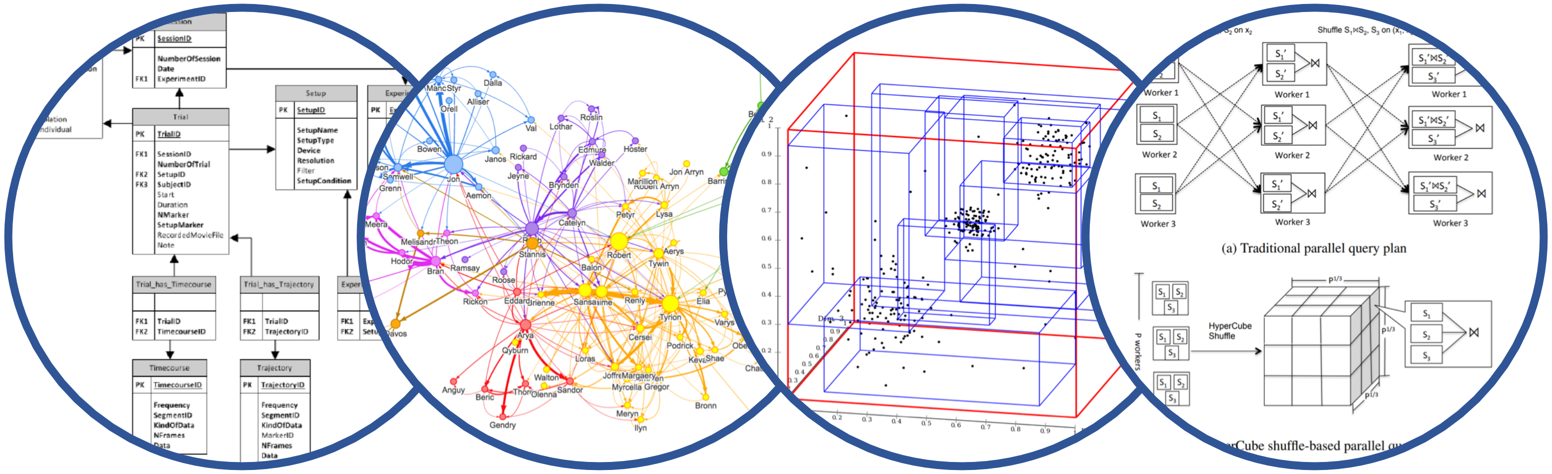


Introduction to Data Management

Relational DB Design Theory

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW3 due Saturday (11:00 PM)

Recap

- ER Diagrams
  - Conceptual modeling
  - Rules of thumb for converting diagram into schema

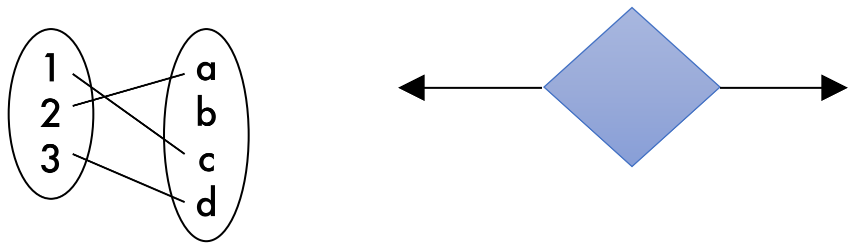
```
CREATE TABLE Product (
    name VARCHAR(100) PRIMARY KEY,
    ...);
CREATE TABLE Company (
    name VARCHAR(100) PRIMARY KEY,
    ...);
CREATE TABLE Makes (
    cname VARCHAR(100) REFERENCES Company,
    pname VARCHAR(100) REFERENCES Product,
    PRIMARY KEY (cname, pname),
    ...);
```

Unique Constraint

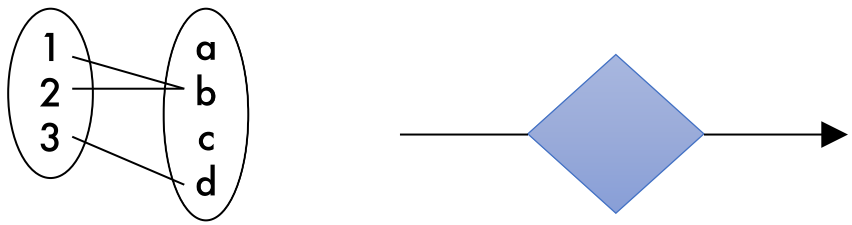
- A `UNIQUE` attribute has a single unique value for each tuple in the relation.
- Same behavior we get with Primary Key, but we can have multiple `UNIQUE`

Multiplicity of E/R Relations

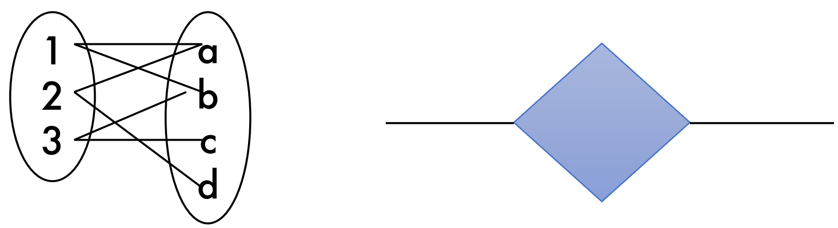
- **one-one:**



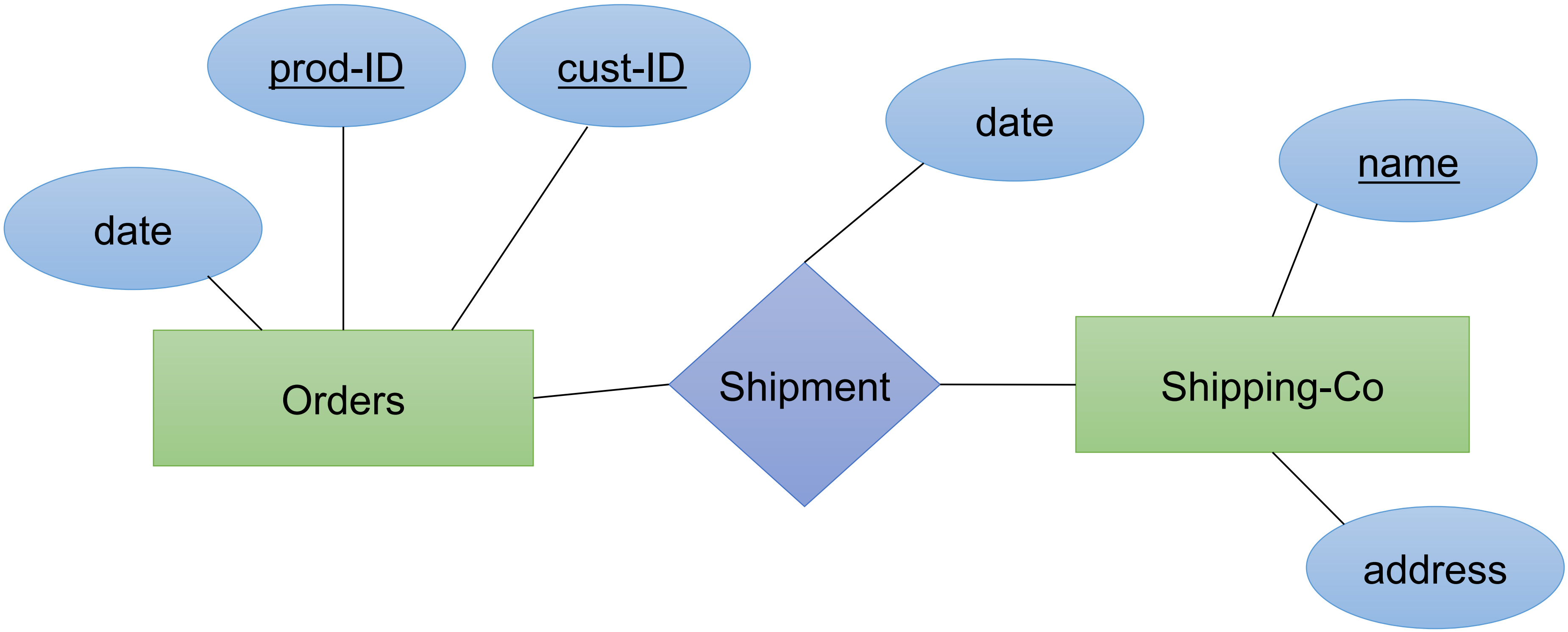
- **many-one**



- **many-many**

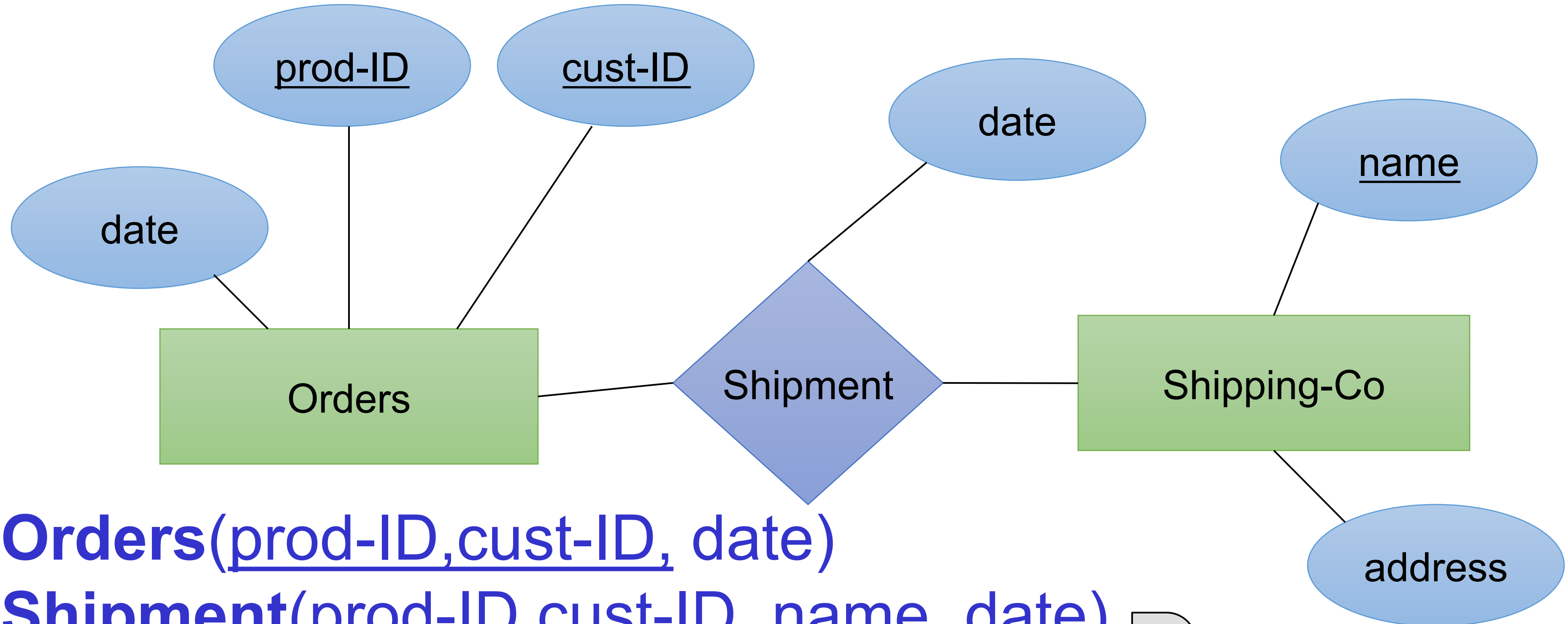


N to N Relationships to Relations



Represent this in relations

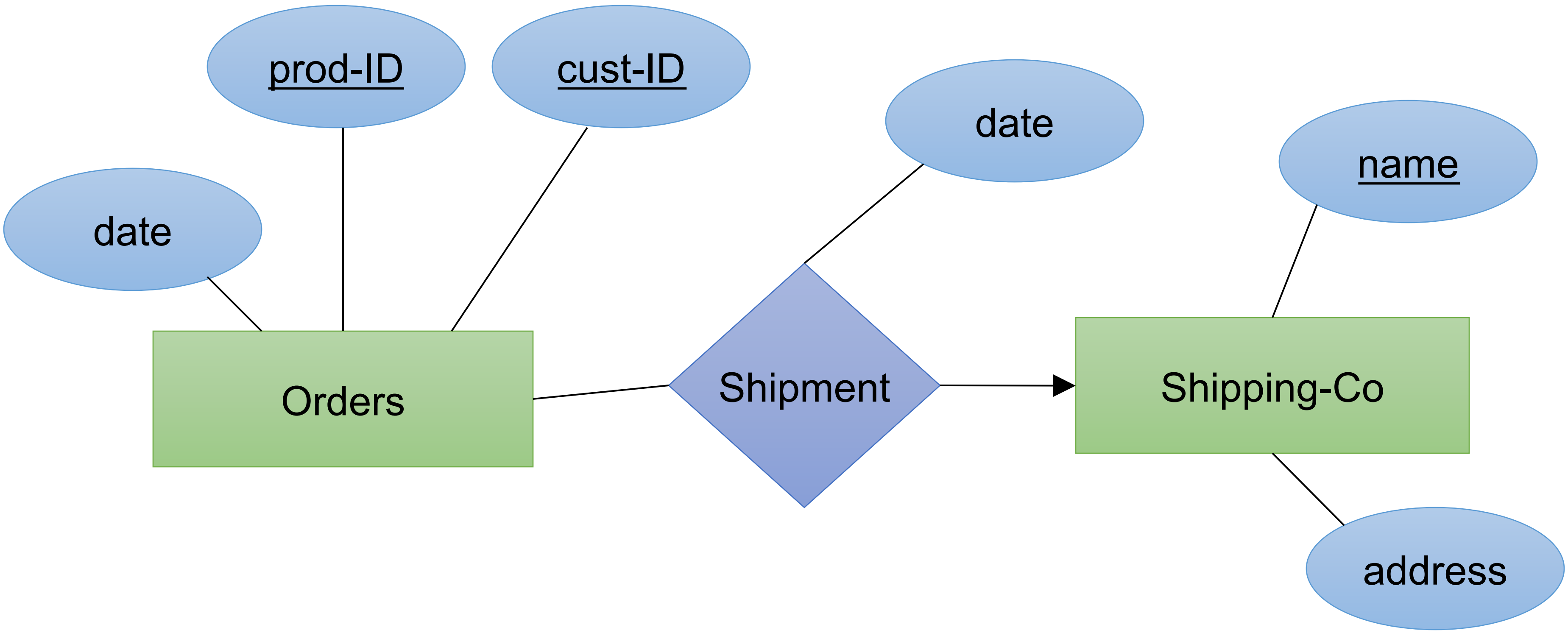
N to N Relationships to Relations



Orders(prod-ID,cust-ID, date)
Shipment(prod-ID,cust-ID, name, date)
Shipping-Co(name, address)

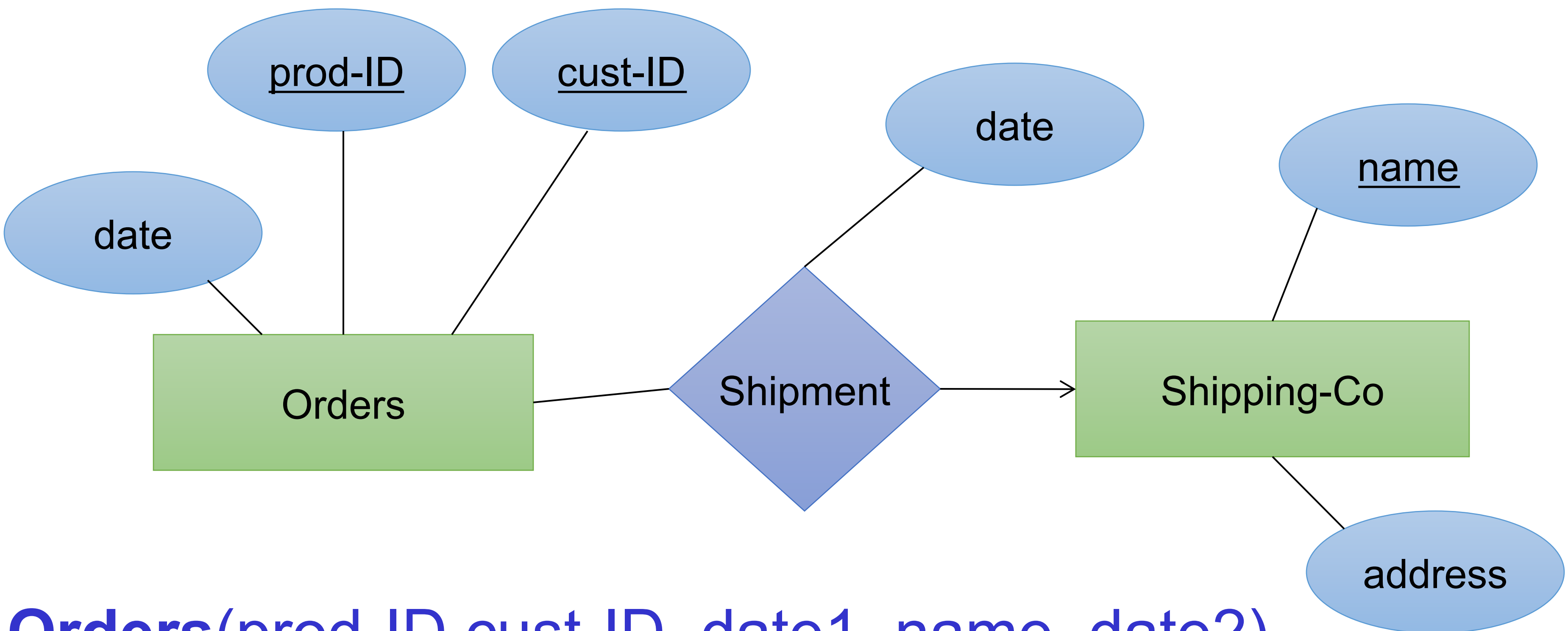
prod-ID	cust-ID	name	date
Gizmo55	Joe12	UPS	4/10/2011
Gizmo55	Joe12	FEDEX	4/9/2011

N to 1 Relationships to Relations



Represent this in relations

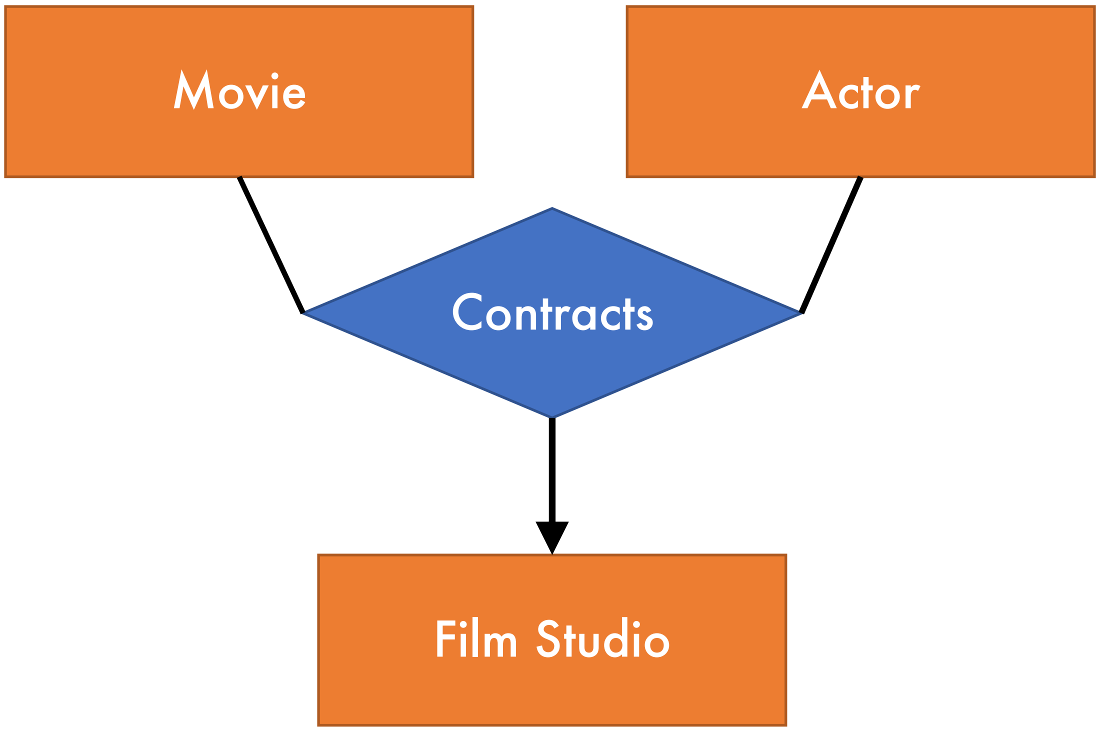
N to 1 Relationships to Relations



Orders(<u>prod-ID,cust-ID,</u> date1, name, date2)
Shipping-Co(<u>name</u>, address)

Remember: no separate relations for many-one relationship

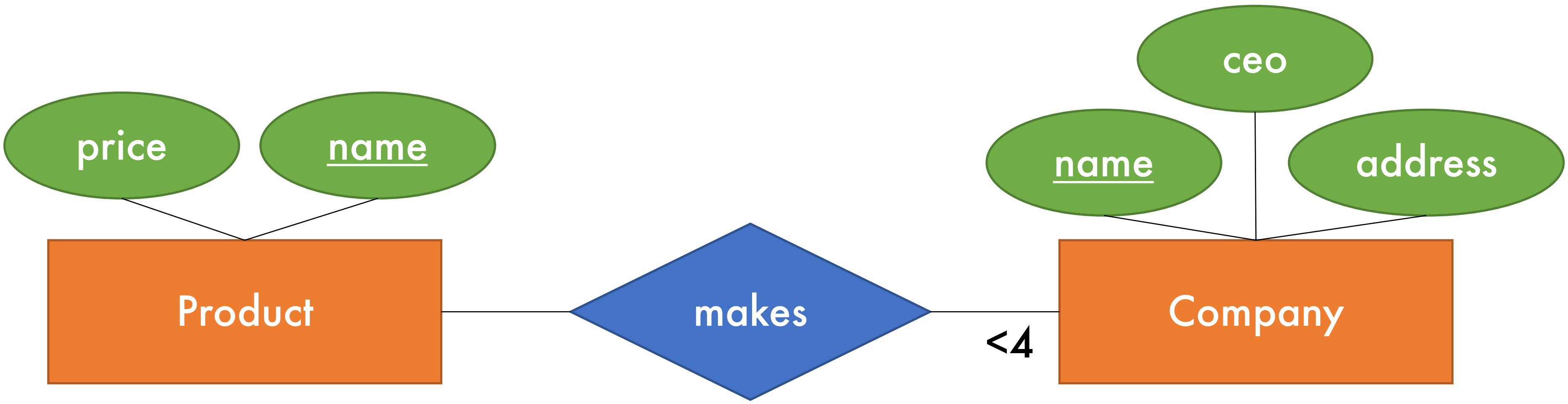
Multi-Way Relations



[precisely: Arrow pointing to some E means that if we select one entity from each of the other entity sets in the relationship, that **combination of entities** is related to at most one entity in E]

Misc Constraints

- Normal arrows are shorthand versions of (<=1)
- Rounded arrows are shorthand versions of (=1)



Each product can be made by, at most, 3 companies

Other Constraints

- `CHECK (condition)`
  - Single attribute
  - Single tuples

```
CREATE TABLE User (
    uid INT PRIMARY KEY,
    firstName TEXT,
    lastName TEXT,
    age INT CHECK (age > 12 AND age < 120),
    email TEXT,
    phone TEXT,
    CHECK (email IS NOT NULL OR phone IS NOT NULL)
);
```

Referential Constraint Maintenance

`ON UPDATE/ON DELETE`

- `NO ACTION` → (default) error out
- `CASCADE` → update/delete referencers
- `SET NULL` → set referencers' field to NULL
- `SET DEFAULT` → set referencers' field to default
  - Assumes default was set, e.g.

```
CREATE TABLE Table (
  id INT DEFAULT 42 REFERENCES OtherTable,
  ...
);
```

Referential Constraint Maintenance

```
CREATE TABLE Company (
    name VARCHAR(100) PRIMARY KEY);
CREATE TABLE Product (
    name VARCHAR(100) PRIMARY KEY,
    cname VARCHAR(100)
      REFERENCES Company
      ON UPDATE CASCADE
      ON DELETE SET NULL);
```

Company

name
Hasbro
Nyform

Product

name	cname
Beyblade	Hasbro
Troll	Hasbro

Referential Constraint Maintenance

```
CREATE TABLE Company (
    name VARCHAR(100) PRIMARY KEY);
CREATE TABLE Product (
    name VARCHAR(100) PRIMARY KEY,
    cname VARCHAR(100)
      REFERENCES Company
      ON UPDATE CASCADE
      ON DELETE SET NULL);
```

Company

name
Hasbro
Nyform

Product

name	cname
Beyblade	Hasbro
Troll	Hasbro

```
UPDATE Company
   SET name = 'lmao’
 WHERE name = ‘Hasbro’;
```

Referential Constraint Maintenance

```
CREATE TABLE Company (
    name VARCHAR(100) PRIMARY KEY);
CREATE TABLE Product (
    name VARCHAR(100) PRIMARY KEY,
    cname VARCHAR(100)
      REFERENCES Company
      ON UPDATE CASCADE
      ON DELETE SET NULL);
```

Company

name
lmao
Nyform

Product

name	cname
Beyblade	lmao
Troll	lmao

```
UPDATE Company
   SET name = 'lmao’
 WHERE name = ‘Hasbro’;
```

Referential Constraint Maintenance

```
CREATE TABLE Company (
    name VARCHAR(100) PRIMARY KEY);
CREATE TABLE Product (
    name VARCHAR(100) PRIMARY KEY,
    cname VARCHAR(100)
      REFERENCES Company
      ON UPDATE CASCADE
      ON DELETE SET NULL);
```

Company

name
lmao
Nyform

Product

name	cname
Beyblade	lmao
Troll	lmao

```
DELETE FROM Company
 WHERE name = 'lmao';
```

Referential Constraint Maintenance

```
CREATE TABLE Company (
    name VARCHAR(100) PRIMARY KEY);
CREATE TABLE Product (
    name VARCHAR(100) PRIMARY KEY,
    cname VARCHAR(100)
      REFERENCES Company
      ON UPDATE CASCADE
      ON DELETE SET NULL);
```

Company

name
Nyform

Product

name	cname
Beyblade	NULL
Troll	NULL

```
DELETE FROM Company
 WHERE name = ‘lmao’;
```

Assertions

- Hard to support
- Usually impractical
- Usually not supported
  - Simulated with triggers

```
CREATE ASSERTION myAssert CHECK
     (NOT EXISTS (
          SELECT Product.name
            FROM Product, Purchase
           WHERE Product.name = Purchase.prodName
           GROUP BY Product.name
          HAVING count(*) > 200));
```

Triggers

- Triggers activate on a specified event

```
CREATE TRIGGER LowCredit ON Purchasing.PurchaseOrderHeader
AFTER INSERT AS
  IF (ROWCOUNT_BIG() = 0) RETURN;
  IF EXISTS (SELECT *
             FROM Purchasing.PurchaseOrderHeader AS p
             JOIN inserted AS i
             ON p.PurchaseOrderID = i.PurchaseOrderID
             JOIN Purchasing.Vendor AS v
             ON v.BusinessEntityID = p.VendorID
             WHERE v.CreditRating = 5
            )
    BEGIN
      RAISERROR ('A vendor''s credit rating is too
                  low to accept new purchase orders.', 16, 1);
      ROLLBACK TRANSACTION;
      RETURN
    END;
GO
```

= you don't need to study this for the class

Goals for Today

- Figure out the fundamentals of what makes a good schema

Outline

- Background
  - Anomalies, i.e. things we want to avoid
  - Functional Dependencies (FDs)
  - Closures and formal definitions of keys
- Normalization: BCNF Decomposition
- Losslessness

Think About This

Make a simple directory that can:

- Hold information about name, SSN, phone, and city
- Associate **people** with the **city** they live in
- Associate **people** with any **phone numbers** they have

Name	SSN	Phone	City
Fred	123-45-6789	206-555-9999	Seattle
Fred	123-45-6789	206-555-8888	Seattle
Joe	987-65-4321	415-555-7777	San Francisco

The above instance does the job, but are there issues?

Think About This

Make a simple directory that can:

- Hold information about name, SSN, phone, and city
- Associate **people** with the **city** they live in
- Associate **people** with any **phone numbers** they have

Name	SSN	Phone	City
Fred	123-45-6789	206-555-9999	Seattle
Fred	123-45-6789	206-555-8888	Seattle
Joe	987-65-4321	415-555-7777	San Francisco

Anomalies:

- **Redundancy → Slow Update**
  - Change Fred's city to Bellevue (two rows!)
- **Deletion Anomalies**
  - How to delete Joe's phone without deleting Joe?

Think About This

Make a simple directory that can:

- Hold information about name, SSN, phone, and city
- Associate **people** with the **city** they live in
- Associate **people** with any **phone numbers** they have

Name	SSN	Phone	City
Fred	123-45-6789	206-555-9999	Seattle
Fred	123-45-6789	206-555-8888	Seattle
Joe	987-65-4321	415-555-7777	San Francisco

Anomalies:

- **Redundancy → Slow Update**
  - Change Fred's city to Bellevue (two rows!)
- **Deletion Anomalies**
  - How to delete Joe's phone without deleting Joe?

Think About This

Make a simple directory that can:

- Hold information about name, SSN, phone, and city
- Associate **people** with the **city** they live in
- Associate **people** with any **phone numbers** they have

Name	SSN	Phone	City
Fred	123-45-6789	206-555-9999	Seattle
Fred	123-45-6789	206-555-8888	Seattle
Joe	987-65-4321	415-555-7777	San Francisco

Anomalies:

- **Redundancy → Slow Update**
  - Change Fred's city to Bellevue (two rows!)
- **Deletion Anomalies**
  - How to delete Joe's phone without deleting Joe?

Think About This

We can solve the anomalies by converting this

Name	SSN	Phone	City
Fred	123-45-6789	206-555-9999	Seattle
Fred	123-45-6789	206-555-8888	Seattle
Joe	987-65-4321	415-555-7777	San Francisco

into this:

Think About This

We can solve the anomalies by converting this

Name	SSN	Phone	City
Fred	123-45-6789	206-555-9999	Seattle
Fred	123-45-6789	206-555-8888	Seattle
Joe	987-65-4321	415-555-7777	San Francisco

into this:

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	San Francisco

SSN	Phone
123-45-6789	206-555-9999
123-45-6789	206-555-8888
987-65-4321	415-555-7777

How can we systematically avoid anomalies?

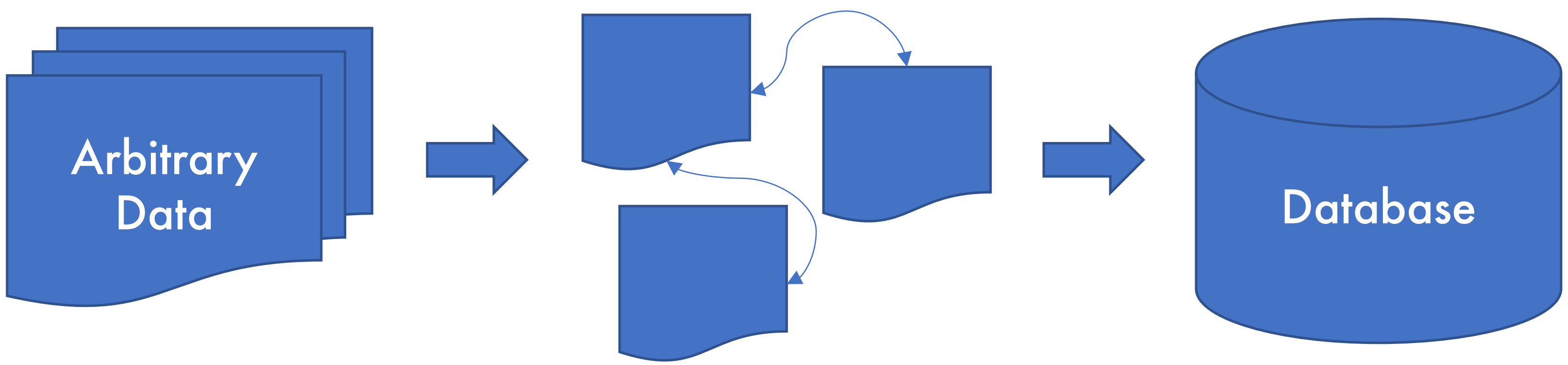
Informal Design Guidelines

- Semantics of attributes should be self-evident
- Avoid redundant information in tuples
- Avoid NULL values in tuples
- Disallow the generation of “spurious” tuples
  - If certain tuples shouldn’t exist, don’t allow them

Database Design

Database Design

Database Design or **Logical Design** or **Relational Schema Design** is the process of organizing data into a database model. This is done by considering what data needs to be stored and the interrelationship of the data.



Database Design

Database Design is about
(1) characterizing data and (2) organizing data

Database Design

Database Design is about

(1) characterizing data and (2) organizing data

Database Design

Database Design is about
(1) characterizing data and (2) organizing data

How to talk about properties
we know or see in the data

Data Interrelationships

How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

Data Interrelationships

How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

The rules that are known to us since we **made them up** or they correlate to **things in the real world**

Data Interrelationships

How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

The rules that are known to us since we **made them up** or they correlate to **things in the real world**

[ex] An engineer knows that a plane model determines the plane's wingspan

Data Interrelationships

How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

Data Interrelationships

How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

Rules that are found
by finding correlations
within the given data

Data Interrelationships

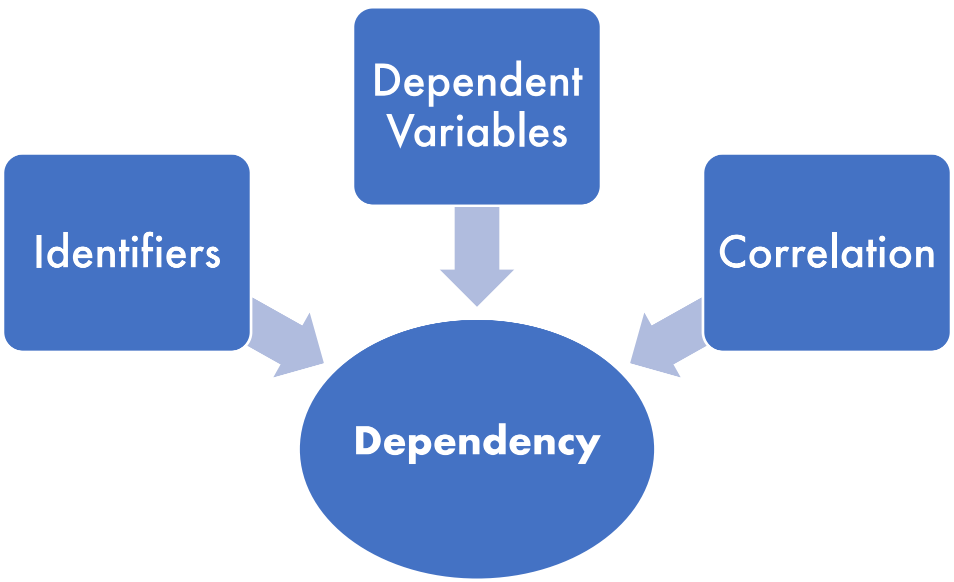
How do we start talking about data interrelationships?

- What **rules** govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

Data Interrelationships

How do we start talking about data interrelationships?

- What **rules** govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis



Data Interrelationships

Functional Dependency

A **Functional Dependency** $A_1, \dots, A_m \rightarrow B_1, \dots, B_n$ holds in the relation R if:

$$\forall t, t' \in R, (t.A_1 = t'.A_1 \land \dots \land t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \land \dots \land t.B_n = t'.B_n)$$

Informally, some attributes determine other attributes.

$$A_1, \dots, A_m \rightarrow B_1, \dots, B_n$$

This is the antecedent

This is the consequent

Warning! Dependency does not imply causation!

Fundamentals of FDs

Armstrong's Axioms

- Axiom of **Reflexivity (Trivial FD)**
- Axiom of **Augmentation**
- Axiom of **Transitivity**

Fundamentals of FDs

Armstrong's Axioms

- Axiom of **Reflexivity (Trivial FD)**

  If $B \subseteq A$ then $A \rightarrow B$

- Axiom of **Augmentation**

- Axiom of **Transitivity**

Fundamentals of FDs

Armstrong's Axioms

- Axiom of **Reflexivity (Trivial FD)**

  If $B \subseteq A$ then $A \rightarrow B$

  [ex] $\{name\} \subseteq \{name, job\}$ so $\{name, job\} \rightarrow \{name\}$

- Axiom of **Augmentation**

- Axiom of **Transitivity**

Fundamentals of FDs

Armstrong's Axioms

- Axiom of **Reflexivity (Trivial FD)**

  If $B \subseteq A$ then $A \rightarrow B$

  [ex] $\{name\} \subseteq \{name, job\}$ so $\{name, job\} \rightarrow \{name\}$

- Axiom of **Augmentation**

  If $A \rightarrow B$ then $\forall C, AC \rightarrow BC$

- Axiom of **Transitivity**

Fundamentals of FDs

Armstrong's Axioms

- Axiom of **Reflexivity (Trivial FD)**

  If $B \subseteq A$ then $A \to B$

  [ex] $\{name\} \subseteq \{name, job\}$ so $\{name, job\} \to \{name\}$

- Axiom of **Augmentation**

  If $A \to B$ then $\forall C, AC \to BC$

  [ex] $\{ID\} \to \{name\}$ so $\{ID, job\} \to \{name, job\}$

- Axiom of **Transitivity**

Fundamentals of FDs

Armstrong's Axioms

- Axiom of **Reflexivity (Trivial FD)**

  If $B \subseteq A$ then $A \to B$

  [ex] $\{name\} \subseteq \{name, job\}$ so $\{name, job\} \to \{name\}$

- Axiom of **Augmentation**

  If $A \to B$ then $\forall C$, $AC \to BC$

  [ex] $\{ID\} \to \{name\}$ so $\{ID, job\} \to \{name, job\}$

- Axiom of **Transitivity**

  If $A \to B$ and $B \to C$ then $A \to C$

Fundamentals of FDs

Armstrong's Axioms

- Axiom of **Reflexivity (Trivial FD)**

  If $B \subseteq A$ then $A \to B$

  [ex] $\{name\} \subseteq \{name, job\}$ so $\{name, job\} \to \{name\}$

- Axiom of **Augmentation**

  If $A \to B$ then $\forall C$, $AC \to BC$

  [ex] $\{ID\} \to \{name\}$ so $\{ID, job\} \to \{name, job\}$

- Axiom of **Transitivity**

  If $A \to B$ and $B \to C$ then $A \to C$

  [ex] $\{ID\} \to \{name\}$ and $\{name\} \to \{initials\}$
  so $\{ID\} \to \{initials\}$

Fundamentals of FDs

Interesting Secondary Rules

- **Pseudo Transitivity**

If $A \rightarrow BC$ and $C \rightarrow D$ then $A \rightarrow BD$

- **Extensivity**

If $A \rightarrow B$ then $A \rightarrow AB$

Fundamentals of FDs

Can I do this to FDs?

I only know $\{ID\} \rightarrow \{name\}$

So $\{ID, hair\ color\} \rightarrow \{name\}$

Fundamentals of FDs

Can I do this to FDs?

I only know $\{ID\} \rightarrow \{name\}$
So $\{ID, hair\ color\} \rightarrow \{name\}$

Yes!

Fundamentals of FDs

Can I do this to FDs?

I only know $\{ID\} \rightarrow \{name\}$

So $\{ID, hair\ color\} \rightarrow \{name\}$

Yes!

Adding more attributes to the antecedent can never remove attributes in the consequent.

Fundamentals of FDs

What about this?

I only know $\{ID\} \rightarrow \{name\}$

So $\{ID\} \rightarrow \{name, hair\ color\}$

Fundamentals of FDs

What about this?

I only know $\{ID\} \rightarrow \{name\}$

So $\{ID\} \rightarrow \{name, hair\ color\}$

No!

Fundamentals of FDs

What about this?

I only know $\{ID\} \to \{name\}$

So $\{ID\} \to \{name, hair\ color\}$

No!

No way to use the axioms to introduce hair color to the consequent without also introducing it to the antecedent.

Finding Keys

All this talk about FDs sounds awfully similar to keys…

Closure

Closure

The **Closure** of the set $\{A_1, \dots, A_m\}$, written as $\{A_1, \dots, A_m\}^+$, is the set of attributes B is such that $A_1, \dots, A_m \rightarrow B$.

A closure finds everything a set of attributes determines.

Closure (example)

Given the functional dependencies:

- $SSN \rightarrow Name$
- $Name \rightarrow Initials$

We can derive some closures:

- $Name^+ =$
- $SSN^+ =$
- $Initials^+ =$
- $\{SSN, Initials\}^+ =$

Closure

Closure

The **Closure** of the set $\{A_1, \dots, A_m\}$, written as $\{A_1, \dots, A_m\}^+$, is the set of attributes B is such that $A_1, \dots, A_m \to B$.

A closure finds everything a set of attributes determines.

Closure (example)

Given the functional dependencies:

- $SSN \to Name$
- $Name \to Initials$

We can derive some closures:

- $Name^+ = \{Name, Initials\}$
- $SSN^+ =$
- $Initials^+ =$
- $\{SSN, Initials\}^+ =$

Closure

Closure

The **Closure** of the set $\{A_1, \dots, A_m\}$, written as $\{A_1, \dots, A_m\}^+$, is the set of attributes B is such that $A_1, \dots, A_m \to B$.

A closure finds everything a set of attributes determines.

Closure (example)

Given the functional dependencies:

- $SSN \to Name$
- $Name \to Initials$

We can derive some closures:

- $Name^+ = \{Name, Initials\}$
- $SSN^+ = \{SSN, Name, Initials\}$
- $Initials^+ =$
- $\{SSN, Initials\}^+ =$

Closure

Closure

The **Closure** of the set $\{A_1, \ldots, A_m\}$, written as $\{A_1, \ldots, A_m\}^+$, is the set of attributes B is such that $A_1, \ldots, A_m \rightarrow B$.

A closure finds everything a set of attributes determines.

Closure (example)

Given the functional dependencies:

- $SSN \rightarrow Name$
- $Name \rightarrow Initials$

We can derive some closures:

- $Name^+ = \{Name, Initials\}$
- $SSN^+ = \{SSN, Name, Initials\}$
- $Initials^+ = \{Initials\}$
- $\{SSN, Initials\}^+ =$

Closure

Closure

The **Closure** of the set $\{A_1, \dots, A_m\}$, written as $\{A_1, \dots, A_m\}^+$, is the set of attributes B is such that $A_1, \dots, A_m \rightarrow B$.

A closure finds everything a set of attributes determines.

Closure (example)

Given the functional dependencies:

- $SSN \rightarrow Name$
- $Name \rightarrow Initials$

We can derive some closures:

- $Name^+ = \{Name, Initials\}$
- $SSN^+ = \{SSN, Name, Initials\}$
- $Initials^+ = \{Initials\}$
- $\{SSN, Initials\}^+ = \{SSN, Name, Initials\}$

Closure

Closure Algorithm

Find the closure of $\{A_1, \dots, A_m\}$

$X = \{A_1, \dots, A_m\}$

Repeat until X **does not change**:
if $B1, \dots, Bn \rightarrow C$ is a FD **and** $B1, \dots, Bn \in X$
then $X \leftarrow X \cup C$

In practice:

Repeated use of transitivity

Closure

Closure Algorithm

Find the closure of $\{A_1, \dots, A_m\}$

$X = \{A_1, \dots, A_m\}$

Repeat until X **does not change**:
  if $B1, \dots, Bn \rightarrow C$ is a FD **and** $B1, \dots, Bn \in X$
  then $X \leftarrow X \cup C$

If a FD applies, add the consequent to the answer

In practice:

Repeated use of transitivity

Closure Example

Let’s say we have the following relations and FDs:

Restaurants(rid, name, rating, popularity)

rid → name

rid → rating

rating → popularity

Compute $\{rid\}^+$

Closure Example

Let's say we have the following relations and FDs:

Restaurants(rid, name, rating, popularity)
rid → name
rid → rating
rating → popularity

Compute $\{rid\}^+$

$\{rid\}^+$ = $rid, name, rating, popularity$ (it's a key!)

Finding Keys

What do FDs and Closures do for us?

- Characterize the interrelationships of data
- Able to find keys

Finding Keys

Superkey

A **Superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. for any single attribute B:

$$A_1, \ldots, A_n \rightarrow B$$

In other words, for the set of all attributes C in the relation R, the set $\{A_1, \ldots, A_n\}$ is a superkey iff $\{A_1, \ldots, A_n\}^+ = C$

Finding Keys

Superkey

A **Superkey** is a set of attributes $A_1, \dots, A_n$ s.t. for any single attribute B:

$$A_1, \dots, A_n \rightarrow B$$

In other words, for the set of all attributes C in the relation R, the set $\{A_1, \dots, A_n\}$ is a superkey iff $\{A_1, \dots, A_n\}^+ = C$

Key

A **Key** is a minimal superkey, i.e. no subset of a key is a superkey.

Finding Keys

Superkey

A **Superkey** is a set of attributes $A_1, \dots, A_n$ s.t. for any single attribute B:

$$A_1, \dots, A_n \rightarrow B$$

In other words, for the set of all attributes C in the relation R, the set $\{A_1, \dots, A_n\}$ is a superkey iff $\{A_1, \dots, A_n\}^+ = C$

Key

A **Key** is a minimal superkey, i.e. no subset of a key is a superkey.

Candidate Key

When a relation has multiple keys, each key is a **Candidate Key**.

Usefulness of Keys in Design

What intuitions do we get from data interrelationships?

- FDs that are not superkeys hint at redundancy we can decompose
  - If a FD antecedent <u>is **not** a superkey</u>, we can remove redundant information, i.e. the FD consequent

Usefulness of Keys in Design

What intuitions do we get from data interrelationships?

- FDs that are not superkeys hint at redundancy we can decompose
  - If a FD antecedent <u>is **not** a superkey</u>, we can remove redundant information, i.e. the FD consequent
- Rephrased
  - If $A \rightarrow B$ is holds on our relation, that's great if A is a superkey
  - Otherwise, we can extract B into another table

Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

rid → name

rid → rating

rating → popularity

rid	name	rating	popularity
1	Mee Sum Pastry	3	Respectable
2	Café on the Ave	4	Poppin
3	Guanaco's Tacos	4	Poppin
4	Aladdin Gyro-Cery	5	Poppin

Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

rid → name
rid → rating

Fine because rid is a superkey

rating → popularity

rid	name	rating	popularity
1	Mee Sum Pastry	3	Respectable
2	Café on the Ave	4	Poppin
3	Guanaco's Tacos	4	Poppin
4	Aladdin Gyro-Cery	5	Poppin

Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

rid → name

rid → rating

Fine because rid is a superkey

rating → popularity

rid	name	rating	popularity
1	Mee Sum Pastry	3	Respectable
2	Café on the Ave	4	Poppin
3	Guanaco's Tacos	4	Poppin
4	Aladdin Gyro-Cery	5	Poppin

Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

rid → name
rid → rating

Fine because rid is a superkey

rating → popularity

Redundancy!

rid	name	rating	popularity
1	Mee Sum Pastry	3	Respectable
2	Café on the Ave	4	Poppin
3	Guanaco's Tacos	4	Poppin
4	Aladdin Gyro-Cery	5	Poppin

Database Design

Database Design is about
(1) characterizing data and (2) organizing data

How to talk about properties
we know or see in the data

Database Design

Database Design is about
(1) characterizing data and (2) organizing data

How to organize data to promote
ease of use and efficiency

Normal Forms

Normal Forms

- 1NF → Flat (no nested tables)
- 2NF → No partial FDs (obsolete)
- 3NF → Preserve all FDs, but allow anomalies
- BCNF → No transitive FDs, but can lose FDs
- 4NF → Considers multi-valued dependencies
- 5NF → Considers join dependencies (hard to do)

Only a couple of these are practical outside of database theory

Normal Forms

Normal Forms

- 1NF → Flat (no nested tables)
- 2NF → No partial FDs (obsolete)
- 3NF → Preserve all FDs, but allow anomalies
- BCNF → No transitive FDs, but can lose FDs
- 4NF → Considers multi-valued dependencies
- 5NF → Considers join dependencies (hard to do)

Normal Forms

1NF

A relation R is in **First Normal Form** if all attribute values are atomic. Attribute values cannot be multivalued. Nested relations are not allowed.

We call data in 1NF "flat."

BCNF

BCNF

A relation R is in **Boyce-Codd Normal Form (BCNF)** if for every non-trivial dependency, $X \rightarrow A$, X is a superkey.

Equivalently, a relation R is in BCNF if $\forall X$ either $X^+ = X$ or $X^+ = C$ where C is the set of all attributes in R

Examples

- $R(A, B, C)$ with FDs $A \rightarrow B$ and $B \rightarrow C$
- $R(A, B, C)$ with FDs $A \rightarrow BC$
- $R(A, B, C)$ and $S(A, D, E)$ with FDs $A \rightarrow BCDE$ and $E \rightarrow AD$

BCNF

BCNF

A relation R is in **Boyce-Codd Normal Form (BCNF)** if for every non-trivial dependency, $X \to A$, X is a superkey.

Equivalently, a relation R is in BCNF if $\forall X$ either $X^+ = X$ or $X^+ = C$ where C is the set of all attributes in R

Examples

- $R(A, B, C)$ with FDs $A \to B$ and $B \to C$ is not in BCNF
- $R(A, B, C)$ with FDs $A \to BC$ is in BCNF
- $R(A, B, C)$ and $S(A, D, E)$ with FDs $A \to BCDE$ and $E \to AD$ is in BCNF

Decomposition

- "Extracting" attributes can be done with **decomposition** (split the schema into smaller parts)
- For this class, decomposition means the following:

$$R(A_1, \dots, A_n, B_1, \dots, B_m, C_1, \dots, C_k) \begin{cases} \nearrow R_1(A_1, \dots, A_n, B_1, \dots, B_m) \\ \searrow R_2(A_1, \dots, A_n, C_1, \dots, C_k) \end{cases}$$

Decomposition

- "Extracting" attributes can be done with **decomposition** (split the schema into smaller parts)
- For this class, decomposition means the following:

$$R(A_1, \dots, A_n, B_1, \dots, B_m, C_1, \dots, C_k) \begin{cases} R_1(A_1, \dots, A_n, B_1, \dots, B_m) \\ R_2(A_1, \dots, A_n, C_1, \dots, C_k) \end{cases}$$

Some common attributes are present so we can rejoin data

BCNF Decomposition Algorithm

$Normalize(R)$
  $C \leftarrow$ the set of all attributes in R
  find X **s.t.** $X^+ \neq X$ **and** $X^+ \neq C$
  if X is not found
  then "R is in BCNF"
  else
    decompose R into $R_1(X^+)$ and $R_2\big((C - X^+) \cup X\big)$
    $Normalize(R_1)$
    $Normalize(R_2)$

BCNF

BCNF Decomposition Algorithm

$Normalize(R)$
  $C \leftarrow$ the set of all attributes in R
  find X **s.t.** $X^+ \neq X$ **and** $X^+ \neq C$
  if X is not found
  then "R is in BCNF"
  else
    decompose R into $R_1(X^+)$ and $R_2\big((C - X^+) \cup X\big)$
    $Normalize(R_1)$
    $Normalize(R_2)$

Determine if R is in BCNF already

BCNF Decomposition Algorithm

$Normalize(R)$
  $C \leftarrow$ the set of all attributes in R
  find X **s.t.** $X^+ \neq X$ **and** $X^+ \neq C$
  if X is not found
  then "R is in BCNF"
  else
    decompose R into $R_1(X^+)$ and $R_2\big((C - X^+) \cup X\big)$
    $Normalize(R_1)$
    $Normalize(R_2)$

Determine if R is in BCNF already

Decompose into a relation where X is a superkey

Decompose into a relation with X and attributes X cannot determine

BCNF Decomposition Example

$Normalize(R)$
  $C \leftarrow$ the set of all attributes in R
  find X **s.t.** $X^+ \neq X$ **and** $X^+ \neq C$
  if X is not found
  then "R is in BCNF"
  else
    decompose R into $R_1(X^+)$ and $R_2\big((C - X^+) \cup X\big)$
    $Normalize(R_1)$
    $Normalize(R_2)$

Restaurants(rid, name, rating, popularity, recommended)

rid → name, rating

rating → popularity

popularity → recommended

Losslessness

Definition

Lossless Decomposition is a reversible decomposition, i.e. rejoining all decomposed relations will always result exactly with the original data.

This is the opposite of a **Lossy Decomposition**, an irreversible decomposition, where rejoining all decomposed relations may result something other than the original data, specifically with extra tuples.

This concept might be familiar if you have ever encountered lossless data compression (e.g. Huffman encoding or PNG) or lossy data compression (e.g. JPEG).

Losslessness

Is BCNF decomposition lossless?

Losslessness

Is BCNF decomposition lossless?

Yes!

Proofs below, for those interested.
We will not test on them

Note on Best Practice

- You may inherit a database that could be lossy. Before you use it, it may be worth your time to check if it is lossy.
- Full normalization is nice but can be inefficient
  - Denormalization → don't normalize all the way

Losslessness

Definition – Heath's Theorem

Suppose we have the relation R and three disjoint subsets of the attributes of R we will write as $A_1, \dots, A_n$, $B_1, \dots, B_m$, and $C_1, \dots, C_k$. Suppose we also have a FD that is $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$.

Heath's Theorem states that the decomposition of R into $R_1(A_1, \dots, A_n, B_1, \dots, B_m)$ and $R_2(A_1, \dots, A_n, C_1, \dots, C_k)$ is lossless where R_1 and R_2 are the projections of R on their respective attributes.

$$R(A_1, \dots, A_n, B_1, \dots, B_m, C_1, \dots, C_k) \begin{cases} \nearrow R_1(A_1, \dots, A_n, B_1, \dots, B_m) \\ \searrow R_2(A_1, \dots, A_n, C_1, \dots, C_k) \end{cases}$$

By reflection, the same decomposition of R under the alternate FD $A_1, \dots, A_n \rightarrow C_1, \dots, C_k$ is also lossless.

Losslessness

We can get to some lossless decomposition, but we should have a way to verify losslessness.

Losslessness

Verifying losslessness mathematically is checking that joining decompositions of the data equals the original data

- Assume we have a decomposition of R into $S_1, \ldots, S_n$
- We want to show that $R = S_1 \bowtie \cdots \bowtie S_n$
- Show $R \subseteq S_1 \bowtie \cdots \bowtie S_n$ and $R \supseteq S_1 \bowtie \cdots \bowtie S_n$

Losslessness

Showing $R \subseteq S_1 \bowtie \cdots \bowtie S_n$ is somewhat simple:

The decomposition of R will have it so that every tuple of R is represented at least once → never omit data in decomposition (projection)

Joining the decompositions (natural join → join on same attribute names with same values) will produce at least the original tuples

Losslessness

$R \supseteq S_1 \bowtie \cdots \bowtie S_n$?

Losslessness

Chase Test (Tableau Method)

1. Generate a tableau of generic tuples representing each schema.
2. Each generic tuple has known values corresponding to the respective projection.
3. Until a row reflects the original generic tuple, continue to chase on FDs (extract more agreements of values)

Losslessness

$R(A, B, C, D)$ **decomposed** $S1(A, D), S2(A, C), S3(B, C, D)$

FDs: $A \rightarrow B$, $B \rightarrow C$, $CD \rightarrow A$

Prove: $R \supseteq S_1 \bowtie \cdots \bowtie S_n$

Known: $S1 = \pi_{A,D}(R)$, $S2 = \pi_{A,C}(R)$, $S3 = \pi_{B,C,D}(R)$

$(a, d) \in S1$, $(a, c) \in S2$, $(b, c, d) \in S3$

Losslessness

$R(A, B, C, D)$ **decomposed** $S1(A, D), S2(A, C), S3(B, C, D)$

FDs: $A \rightarrow B$, $B \rightarrow C$, $CD \rightarrow A$

Prove: $(a, b, c, d) \in S1 \bowtie S2 \bowtie S3 \rightarrow (a, b, c, d) \in R$

Known: $S1 = \pi_{A,D}(R)$, $S2 = \pi_{A,C}(R)$, $S3 = \pi_{B,C,D}(R)$

$(a, d) \in S1$, $(a, c) \in S2$, $(b, c, d) \in S3$

Losslessness

$R(A, B, C, D)$ **decomposed** $S1(A, D), S2(A, C), S3(B, C, D)$

FDs: $A \to B,\ B \to C,\ CD \to A$

Prove: $(a, b, c, d) \in S1 \bowtie S2 \bowtie S3 \to (a, b, c, d) \in R$

Known: $S1 = \pi_{A,D}(R),\ S2 = \pi_{A,C}(R),\ S3 = \pi_{B,C,D}(R)$

$(a, d) \in S1,\ (a, c) \in S2,\ (b, c, d) \in S3$

R must contain:

A	B	C	D
a	b1	c1	**d**
a	b2	**c**	d2
a3	**b**	**c**	**d**

Losslessness

$R(A, B, C, D)$ **decomposed** $S1(A, D), S2(A, C), S3(B, C, D)$

FDs: $A \rightarrow B,\ B \rightarrow C,\ CD \rightarrow A$

Prove: $(a, b, c, d) \in S1 \bowtie S2 \bowtie S3 \rightarrow (a, b, c, d) \in R$

Known: $S1 = \pi_{A,D}(R),\ S2 = \pi_{A,C}(R),\ S3 = \pi_{B,C,D}(R)$

$(a, d) \in S1,\ (a, c) \in S2,\ (b, c, d) \in S3$

R must contain:

A	B	C	D
a	b1	c1	**d**
a	b2	**c**	d2
a3	**b**	**c**	**d**

$\Rightarrow$ ($A \rightarrow B$)

A	B	C	D
a	b1	c1	**d**
a	b1	**c**	d2
a3	**b**	**c**	**d**

Losslessness

$R(A, B, C, D)$ **decomposed** $S1(A, D), S2(A, C), S3(B, C, D)$

FDs: $A \rightarrow B,\ B \rightarrow C,\ CD \rightarrow A$

Prove: $(a, b, c, d) \in S1 \bowtie S2 \bowtie S3 \rightarrow (a, b, c, d) \in R$

Known: $S1 = \pi_{A,D}(R),\ S2 = \pi_{A,C}(R),\ S3 = \pi_{B,C,D}(R)$

$(a, d) \in S1,\ (a, c) \in S2,\ (b, c, d) \in S3$

R must contain:

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

$A \rightarrow B$

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

$B \rightarrow C$

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

Losslessness

$R(A, B, C, D)$ **decomposed** $S1(A, D), S2(A, C), S3(B, C, D)$

FDs: $A \rightarrow B,\ B \rightarrow C,\ CD \rightarrow A$

Prove: $(a, b, c, d) \in S1 \bowtie S2 \bowtie S3 \rightarrow (a, b, c, d) \in R$

Known: $S1 = \pi_{A,D}(R),\ S2 = \pi_{A,C}(R),\ S3 = \pi_{B,C,D}(R)$

$(a, d) \in S1,\ (a, c) \in S2,\ (b, c, d) \in S3$

R must contain:

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

$A \rightarrow B$

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

$B \rightarrow C$

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

$CD \rightarrow A$

A	B	C	D
a	b1	c	d
a	b1	c	d2
a	b	c	d

Losslessness

$R(A,B,C,D)$ **decomposed** $S1(A,D), S2(A,C), S3(B,C,D)$

FDs: $A \to B,\ B \to C,\ CD \to A$

Prove: $(a,b,c,d) \in S1 \bowtie S2 \bowtie S3 \to (a,b,c,d) \in R$

Known: $S1 = \pi_{A,D}(R),\ S2 = \pi_{A,C}(R),\ S3 = \pi_{B,C,D}(R)$

$(a,d) \in S1,\ (a,c) \in S2,\ (b,c,d) \in S3$

R must contain:

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

$A \to B$

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

$B \to C$

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

$CD \to A$

A	B	C	D
a	b	c	d
a	b	c	d
a	b	c	d

Losslessness

$R(A, B, C, D)$ **decomposed** $S1(A, D), S2(A, C), S3(B, C, D)$

FDs: $A \to B$, $B \to C$, $CD \to A$

Prove: $(a, b, c, d) \in S1 \bowtie S2 \bowtie S3 \to (a, b, c, d) \in R$

Known: $S1 = \pi_{A,D}(R)$, $S2 = \pi_{A,C}(R)$, $S3 = \pi_{B,C,D}(R)$

$(a, d) \in S1$, $(a, c) \in S2$, $(b, c, d) \in S3$

R must contain:

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

$A \to B$

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

$B \to C$

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

$CD \to A$

A	B	C	D
a	b	c	d
a	b	c	d
a	b	c	d

$(a, b, c, d) \in R$

Losslessness

$R(A,B,C,D)$ **decomposed** $S1(A,D), S2(A,C), S3(B,C,D)$

FDs: $A \to B$, $B \to C$, $CD \to A$

Prove: $(a,b,c,d) \in S1 \bowtie S2 \bowtie S3 \to (a,b,c,d) \in R$

Known: $S1 = \pi_{A,D}(R)$, $S2 = \pi_{A,C}(R)$, $S3 = \pi_{B,C,D}(R)$

$(a,d) \in S1$, $(a,c) \in S2$, $(b,c,d) \in S3$

R must contain:

A	B	C	D
a	b1	c1	d
a	b2	c	d2
a3	b	c	d

$A \to B$

A	B	C	D
a	b1	c1	d
a	b1	c	d2
a3	b	c	d

$B \to C$

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

$CD \to A$

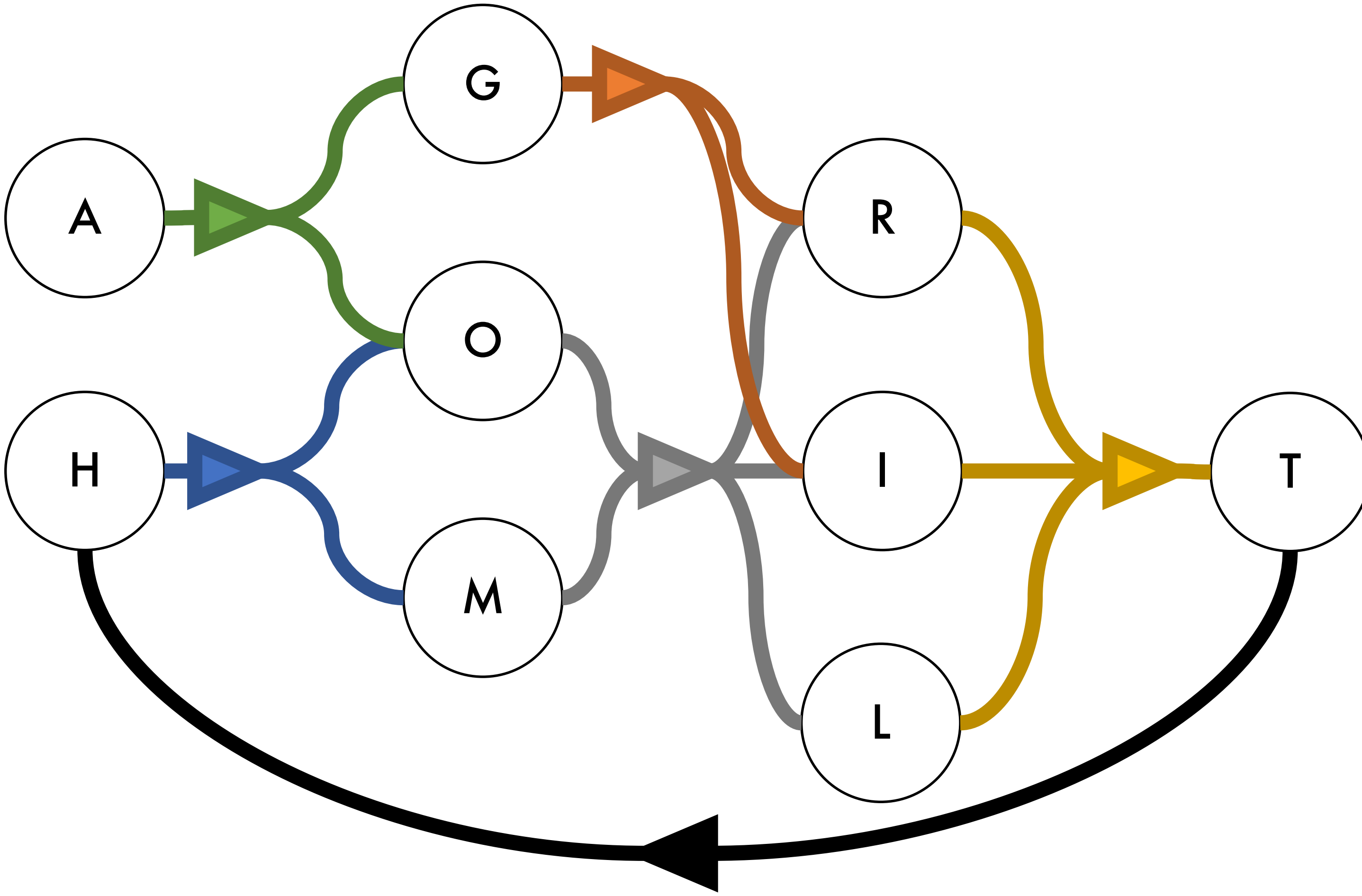
A	B	C	D
a	b	c	d
a	b	c	d
a	b	c	d

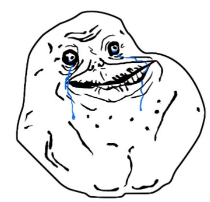
$(a,b,c,d) \in R$

The decomposition of R into S1, S2, and S3 is lossless

Think About This

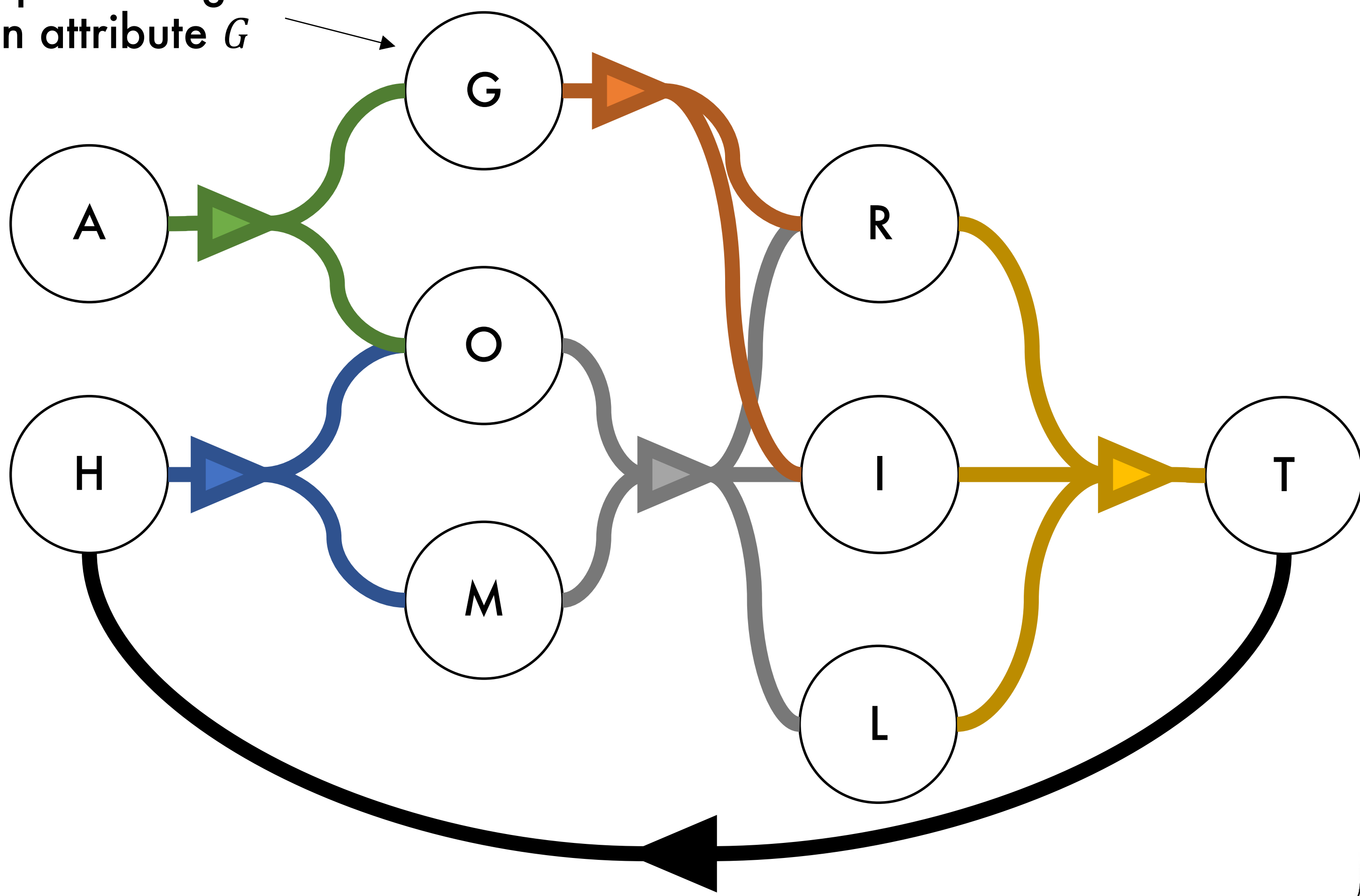
Visually representing FDs: Directed Hypergraphs!

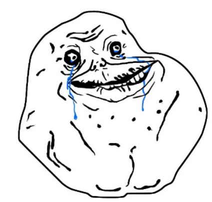




Think About This

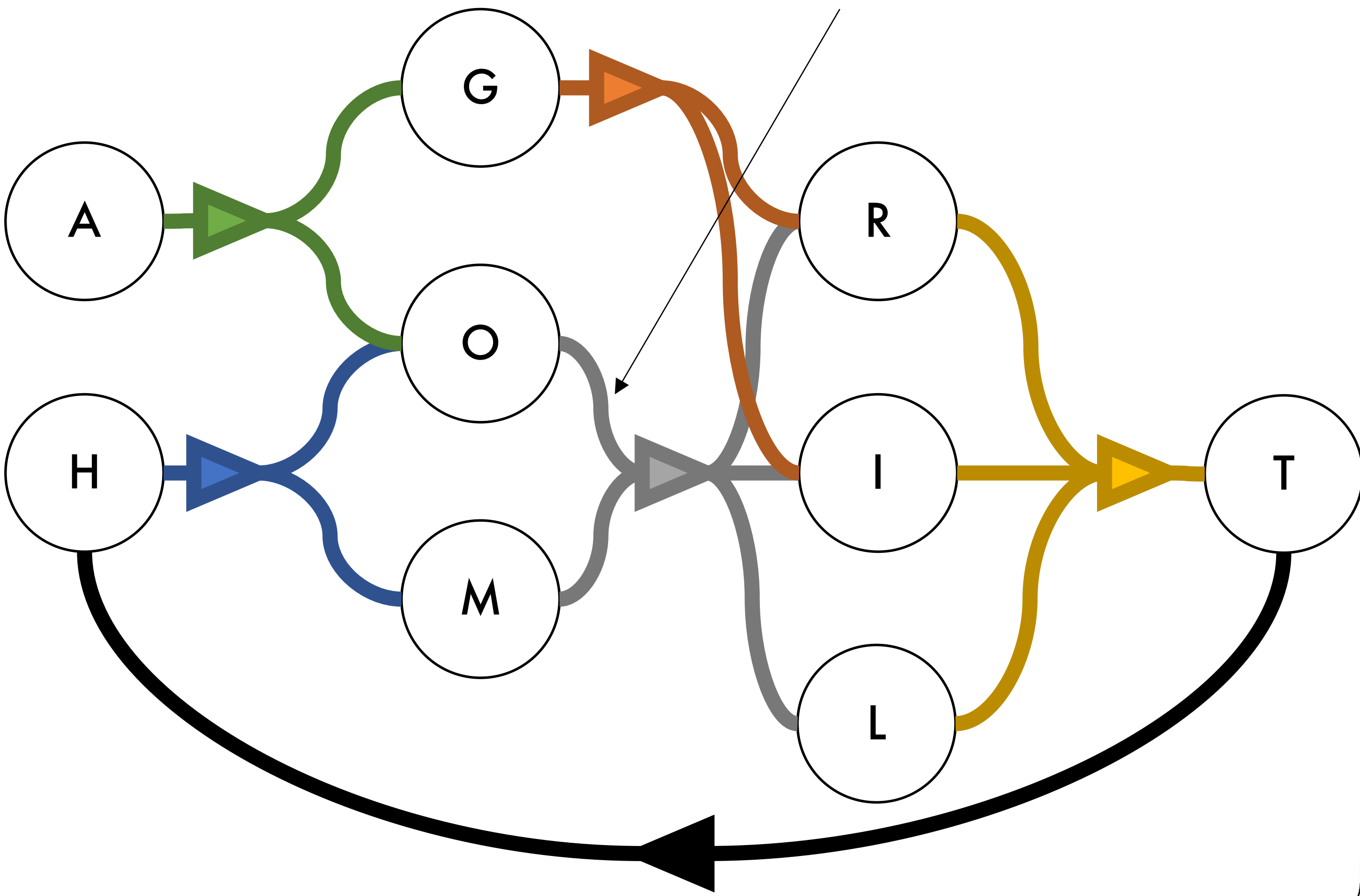
This is a node representing an attribute G

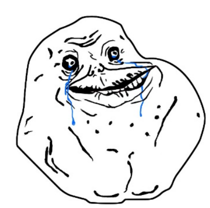




Think About This

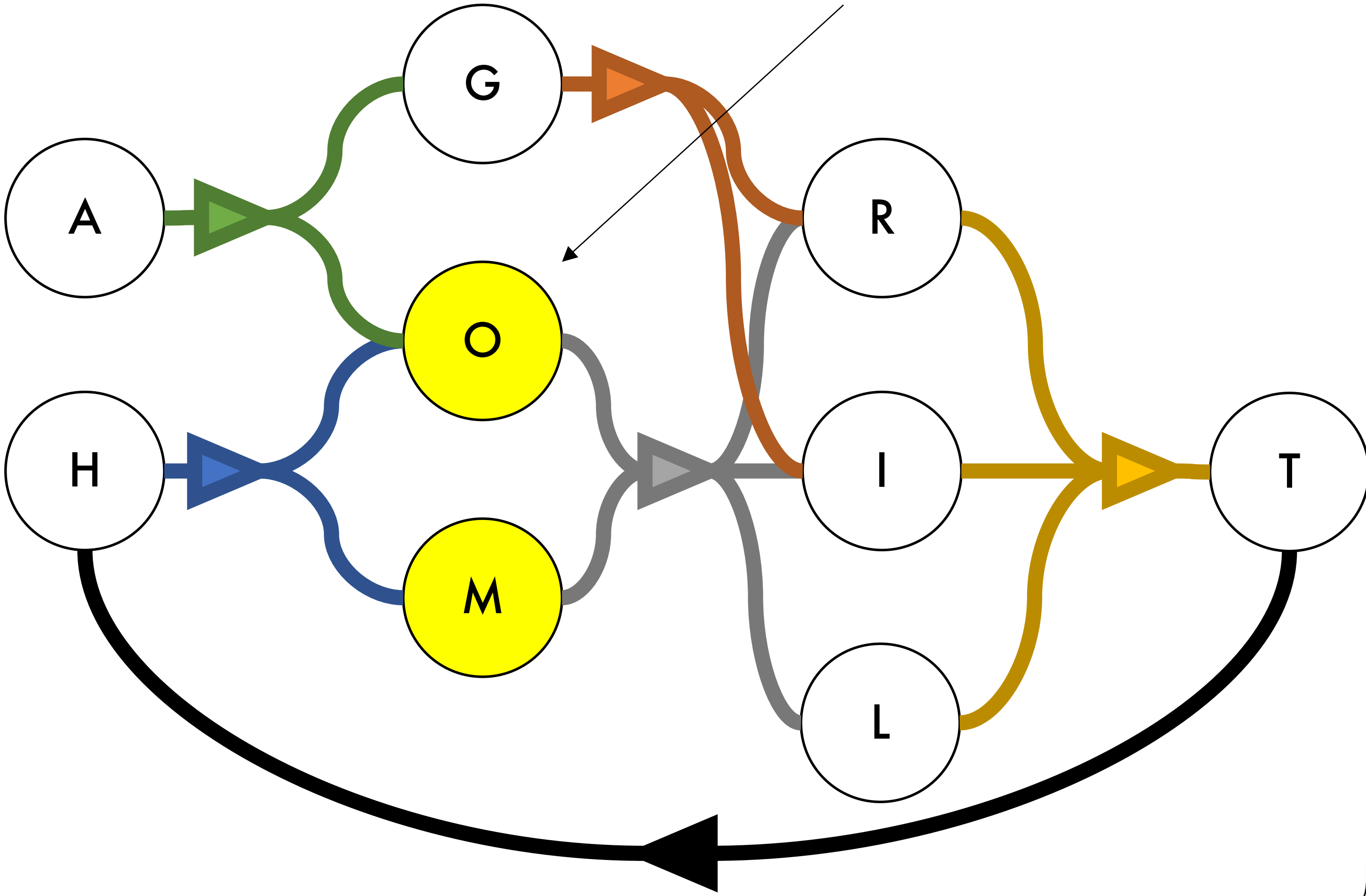
This is a directed hyperedge representing the FD $OM \rightarrow RIL$

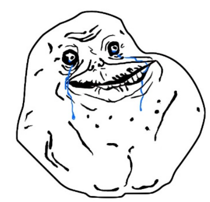




Think About This

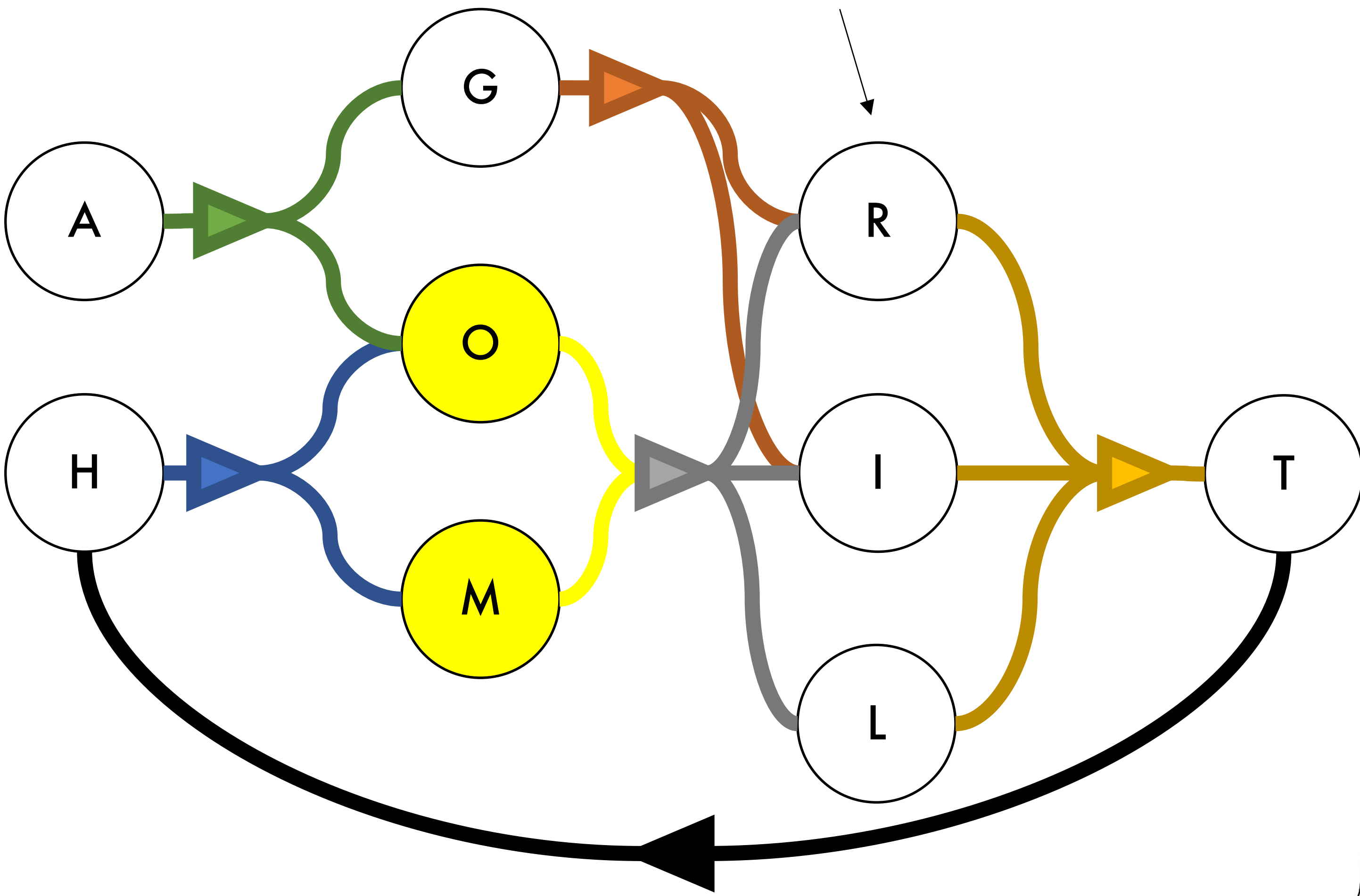
If we know O and M we can use the hyperedge to know R, I, and L

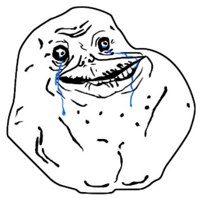




Think About This

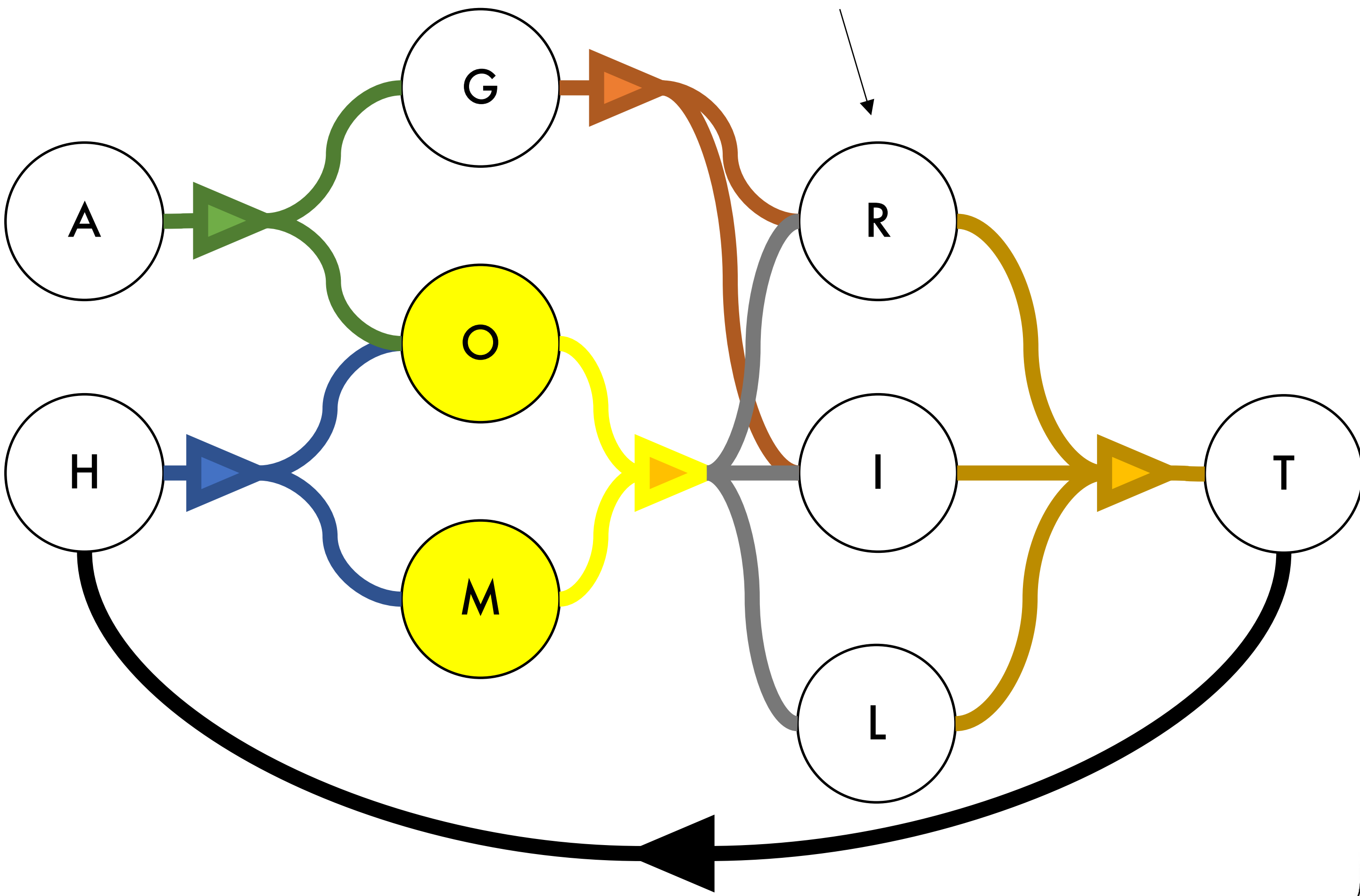
If we know O and M we can use the hyperedge to know R, I, and L

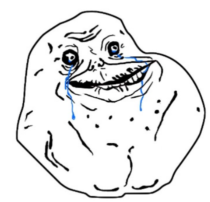




Think About This

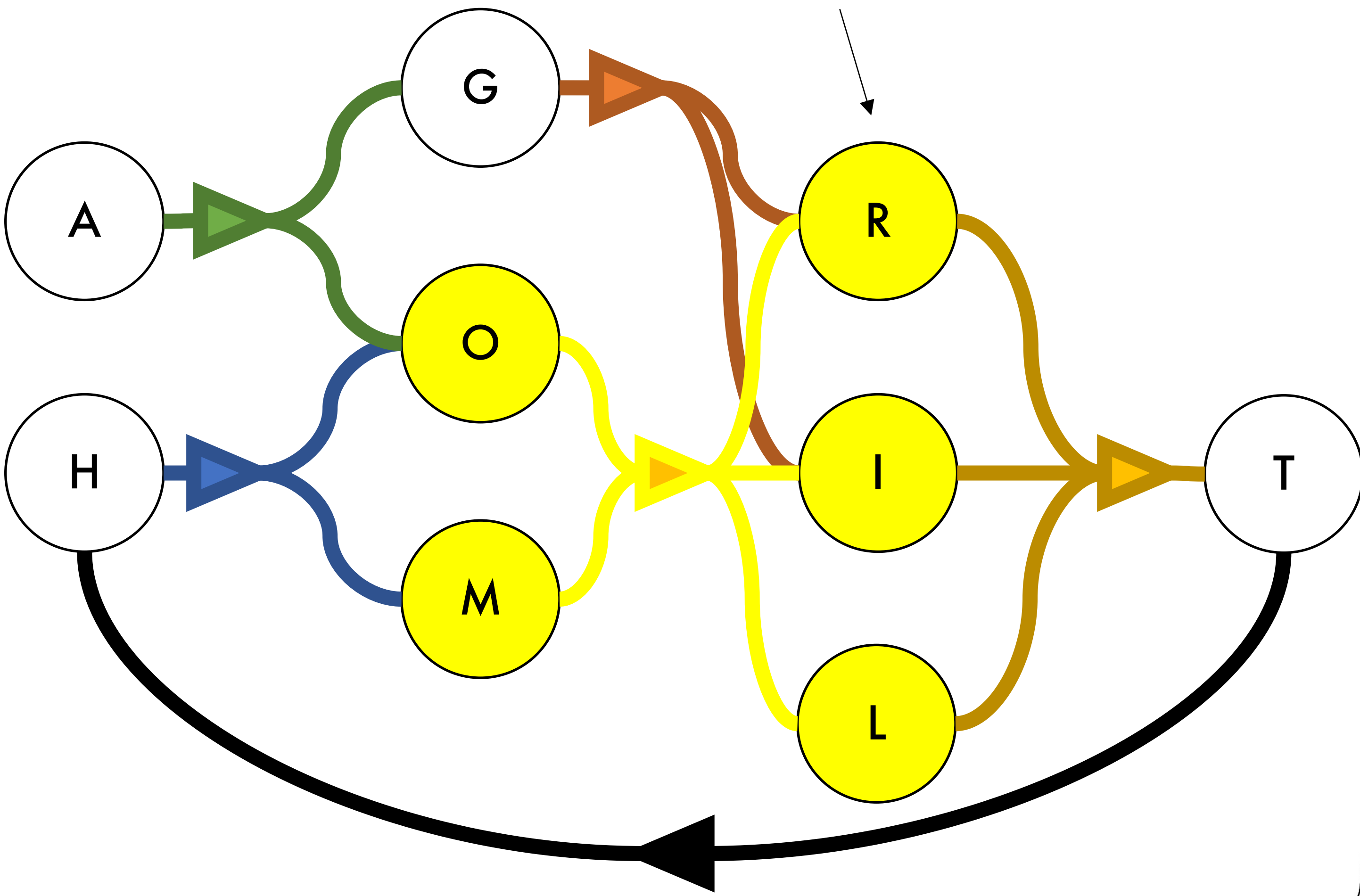
If we know O and M we can use the hyperedge to know R, I, and L

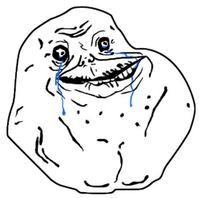




Think About This

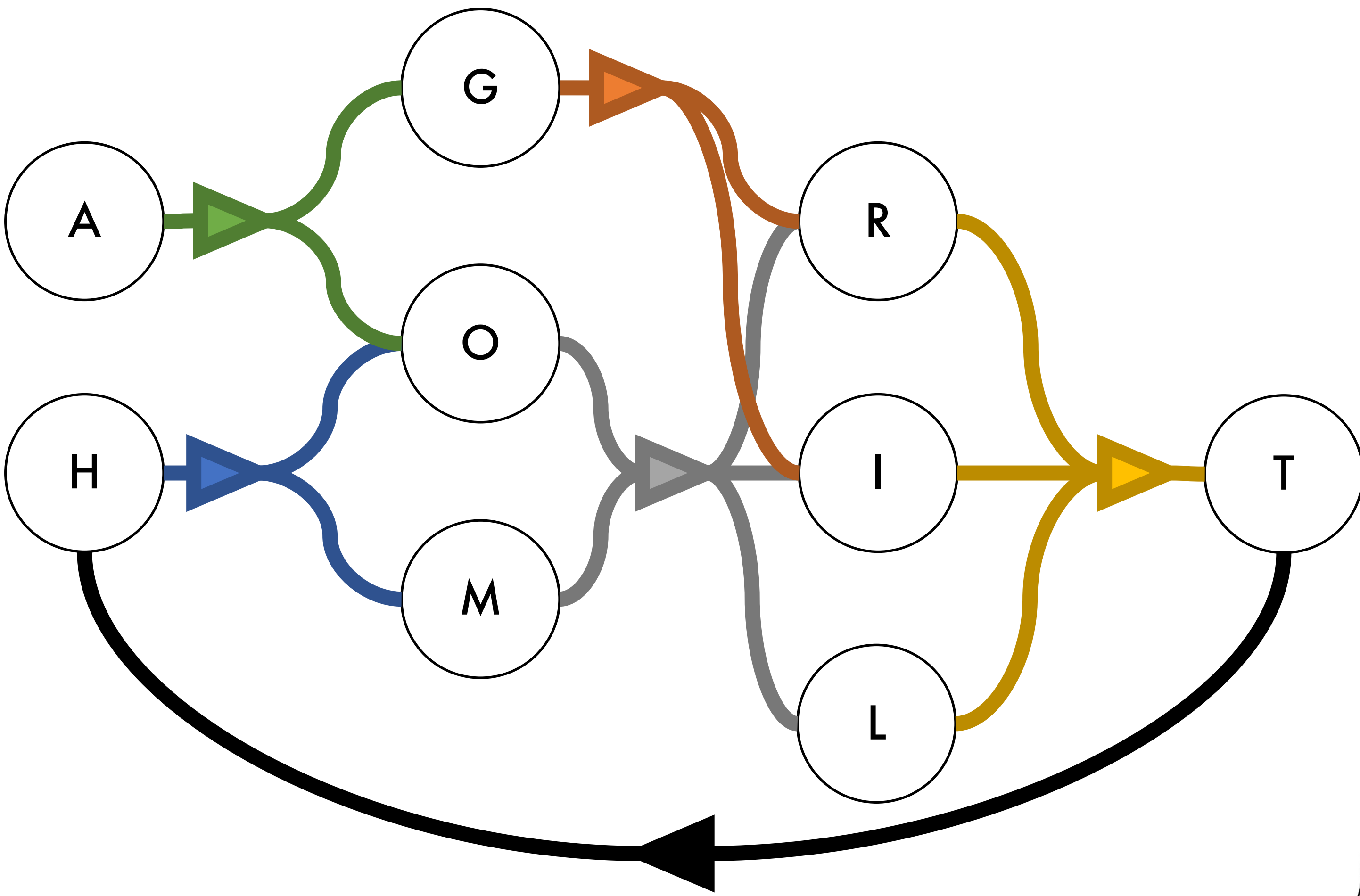
If we know O and M we can use the hyperedge to know R, I, and L

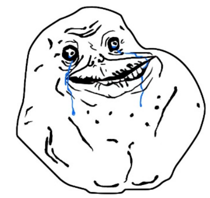




Think About This

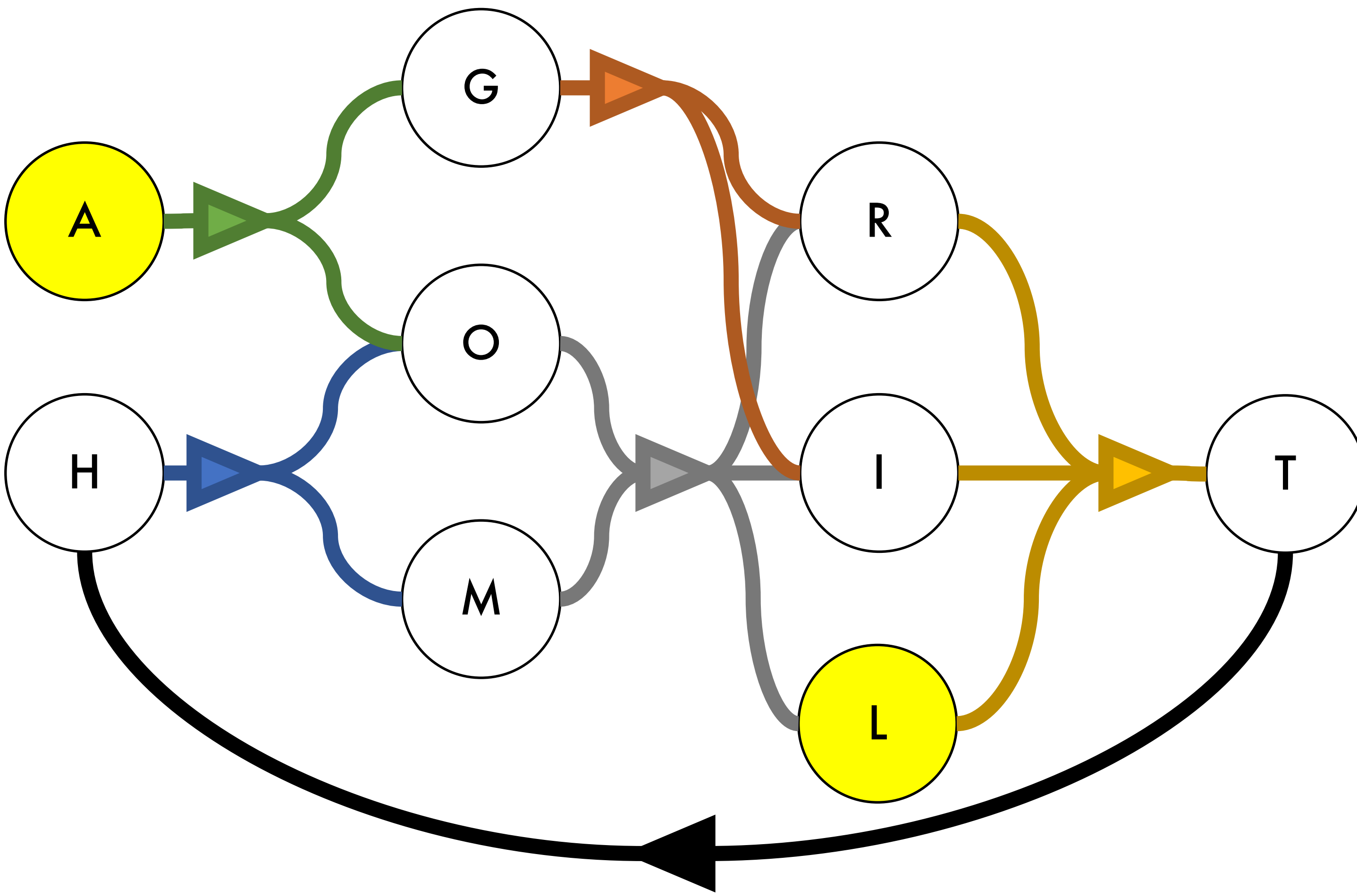
Compute the closure $\{A, L\}^+$

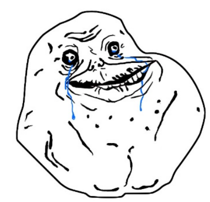




Think About This

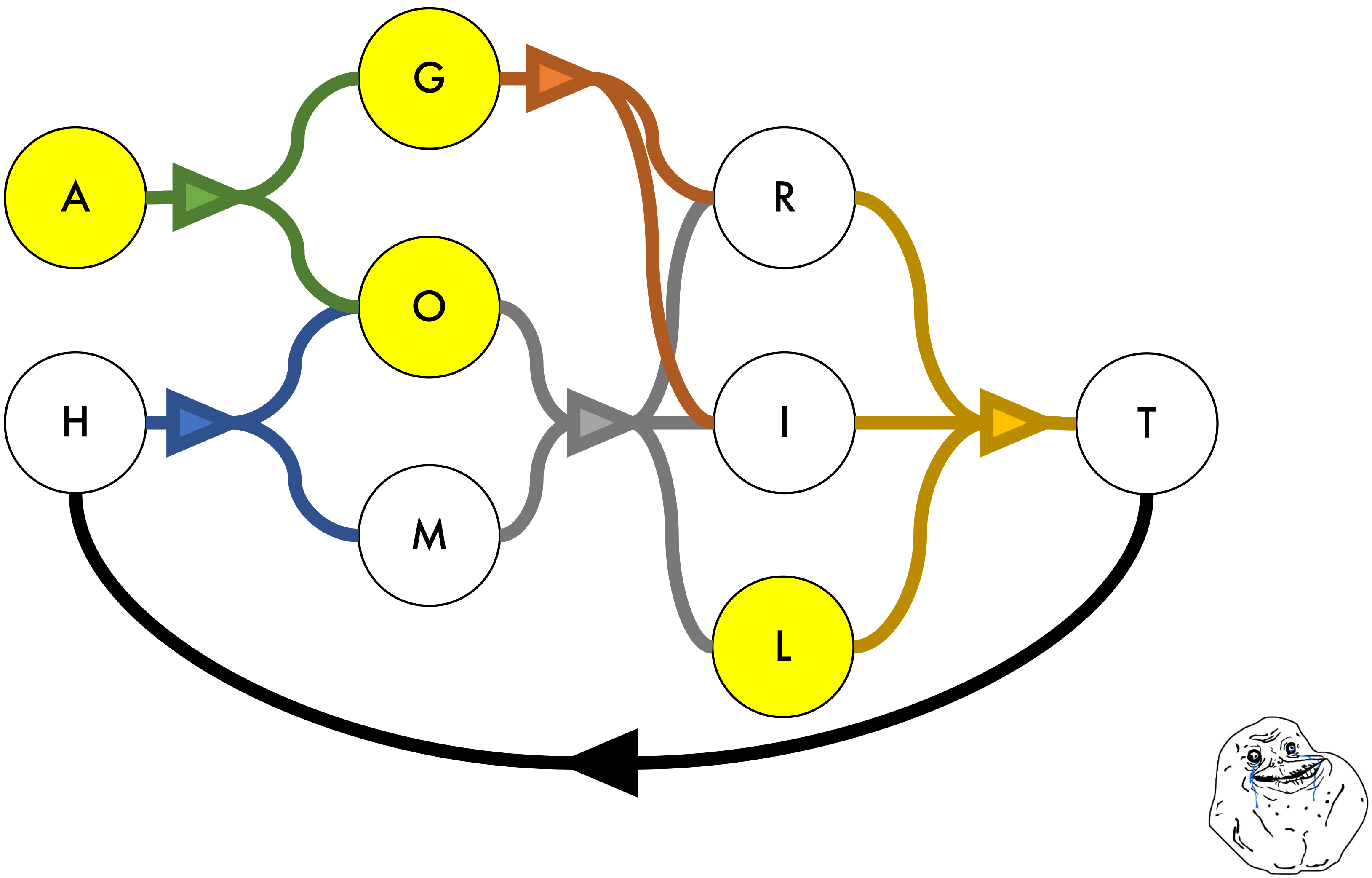
$$\{A, L\}^{+} = \{A, L, \ldots\}$$





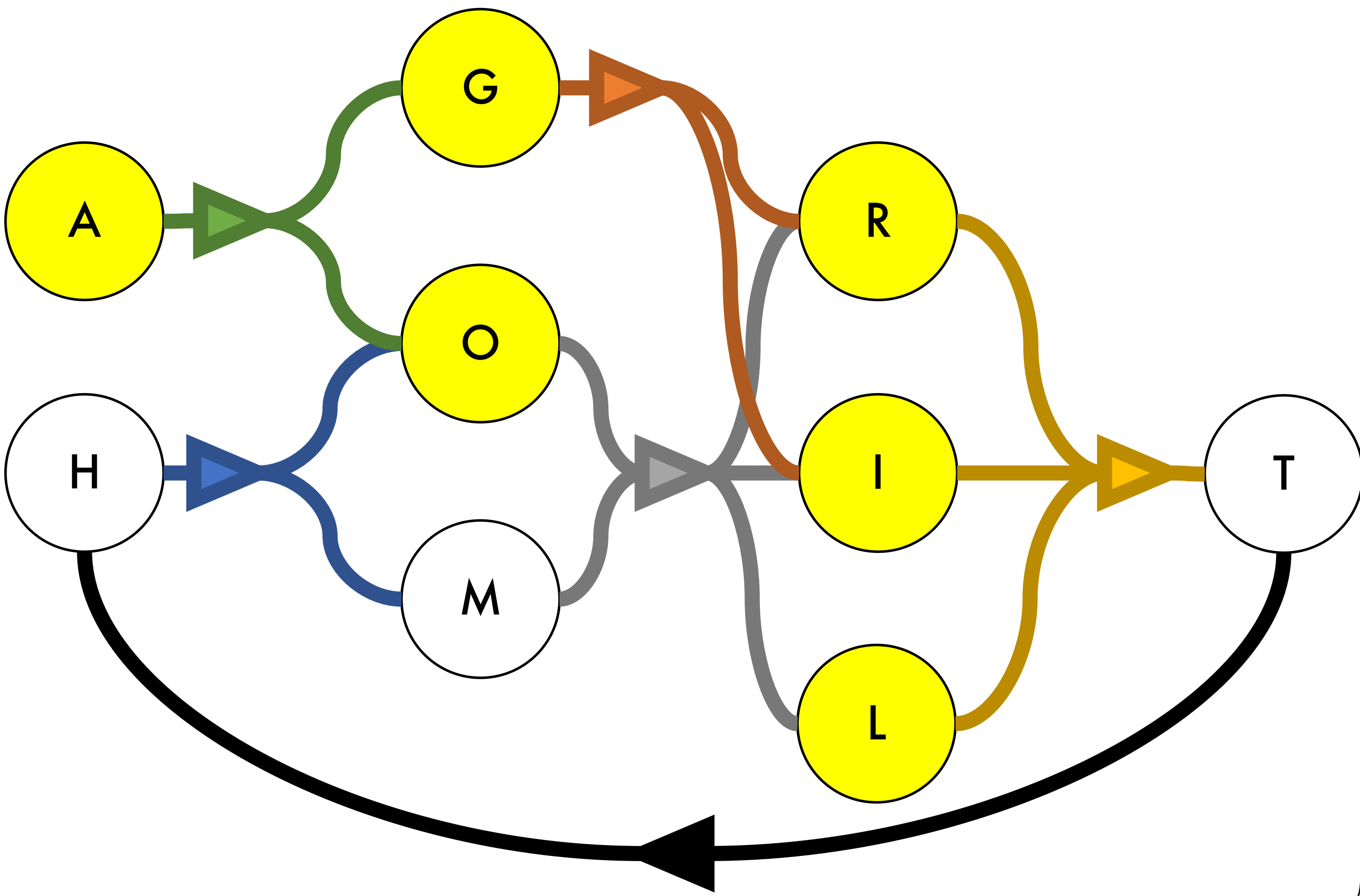
Think About This

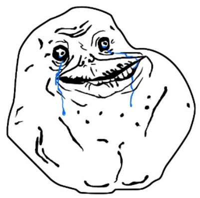
$$\{A, L\}^+ = \{A, L, G, O, \ldots\}$$



Think About This

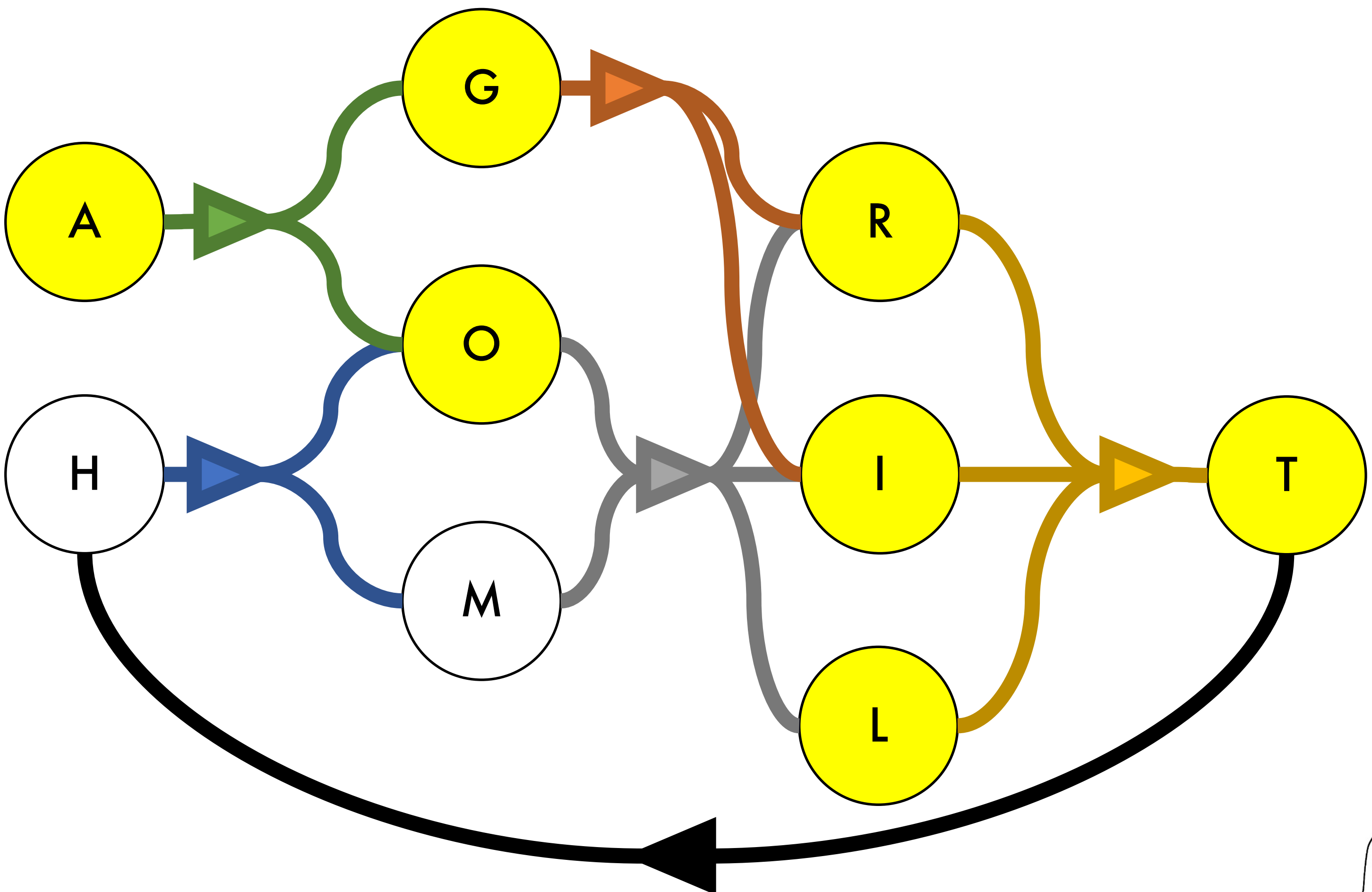
$$\{A, L\}^+ = \{A, L, G, O, R, I \ldots\}$$

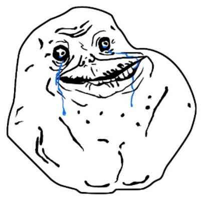




Think About This

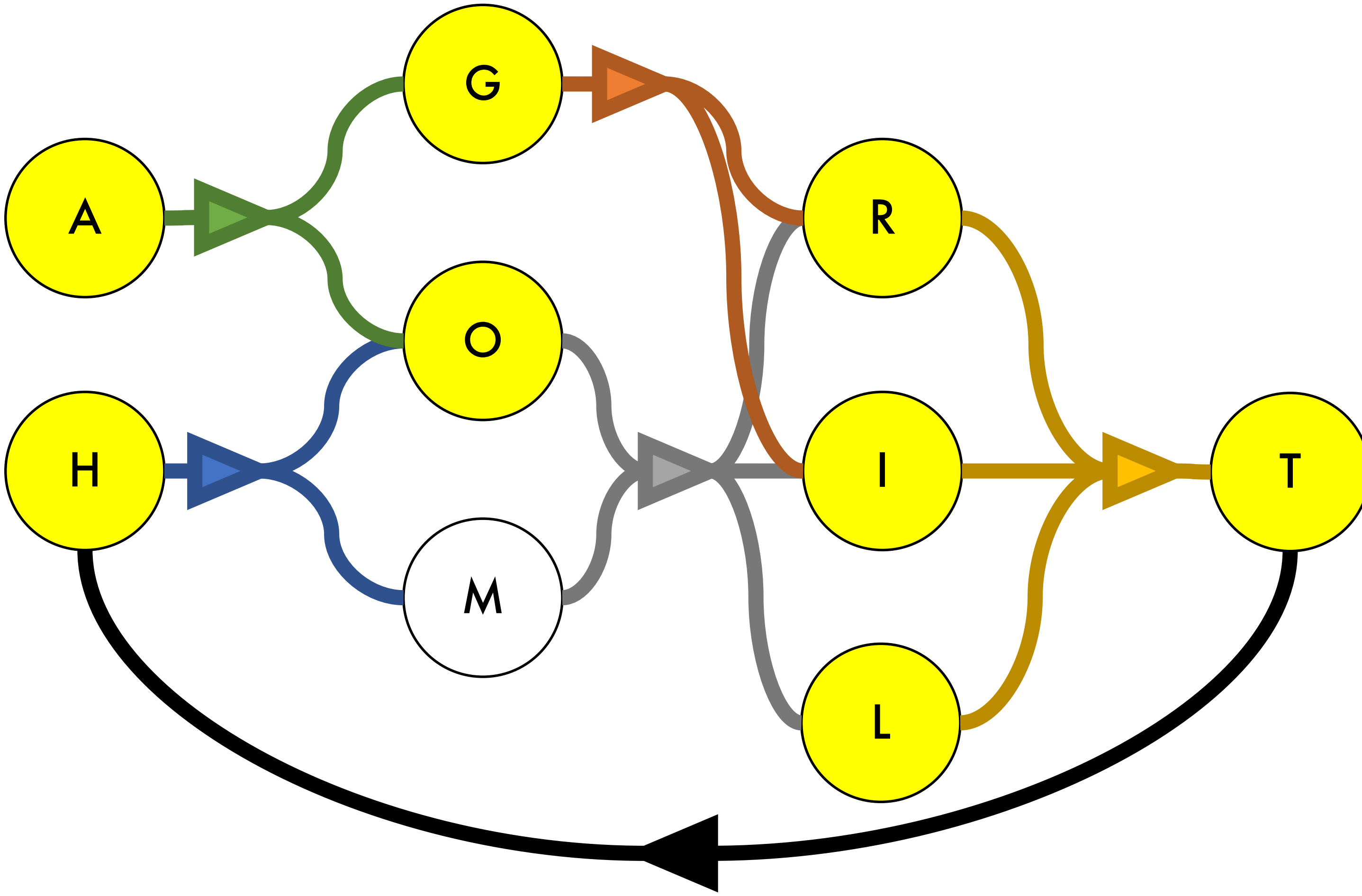
$$\{A, L\}^+ = \{A, L, G, O, R, I, T \ldots\}$$

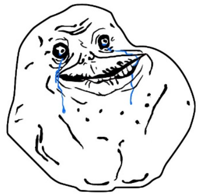




Think About This

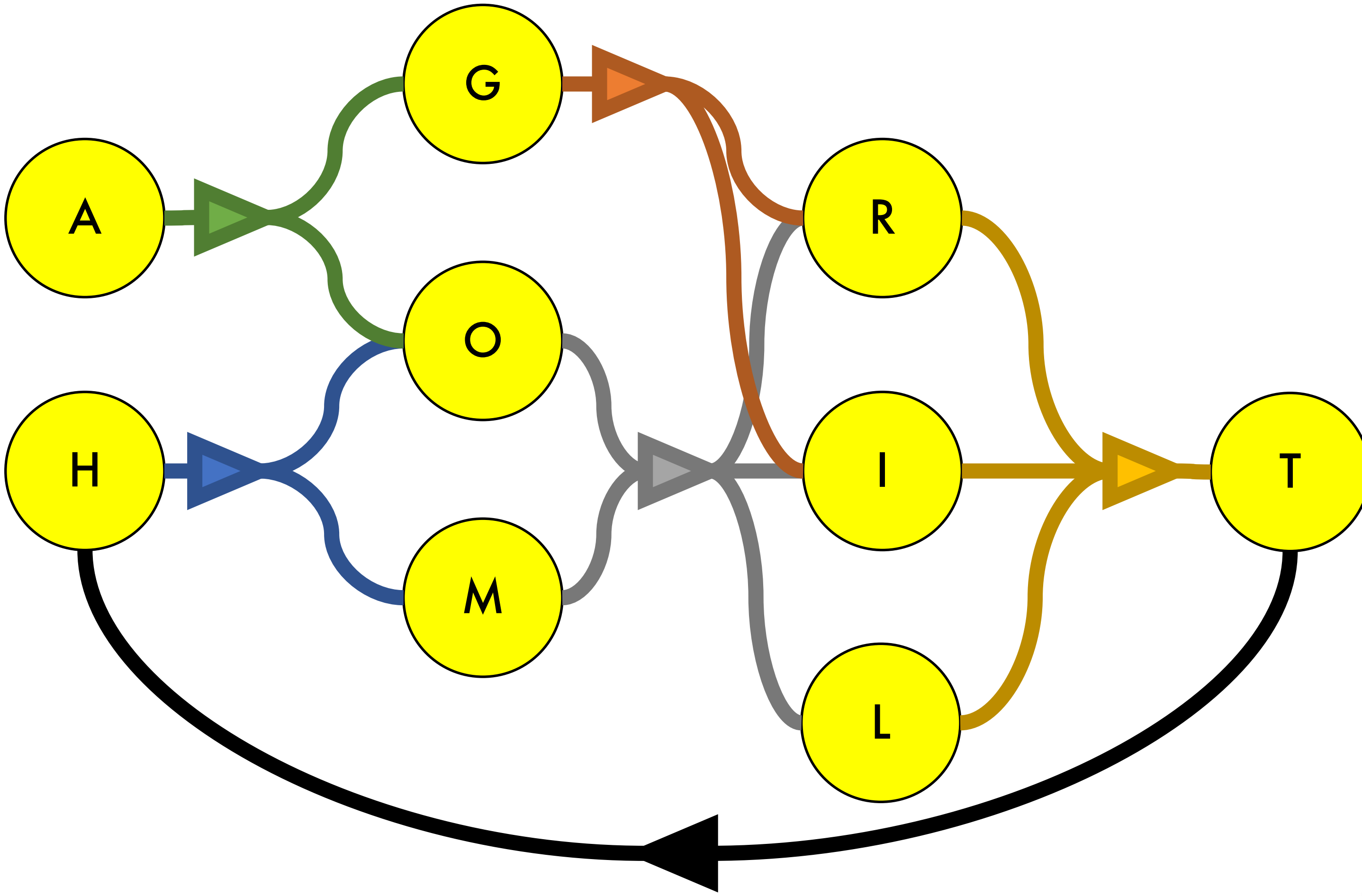
$$\{A, L\}^+ = \{A, L, G, O, R, I, T, H \ldots\}$$

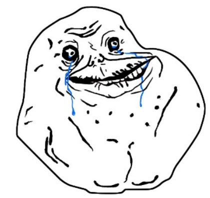




Think About This

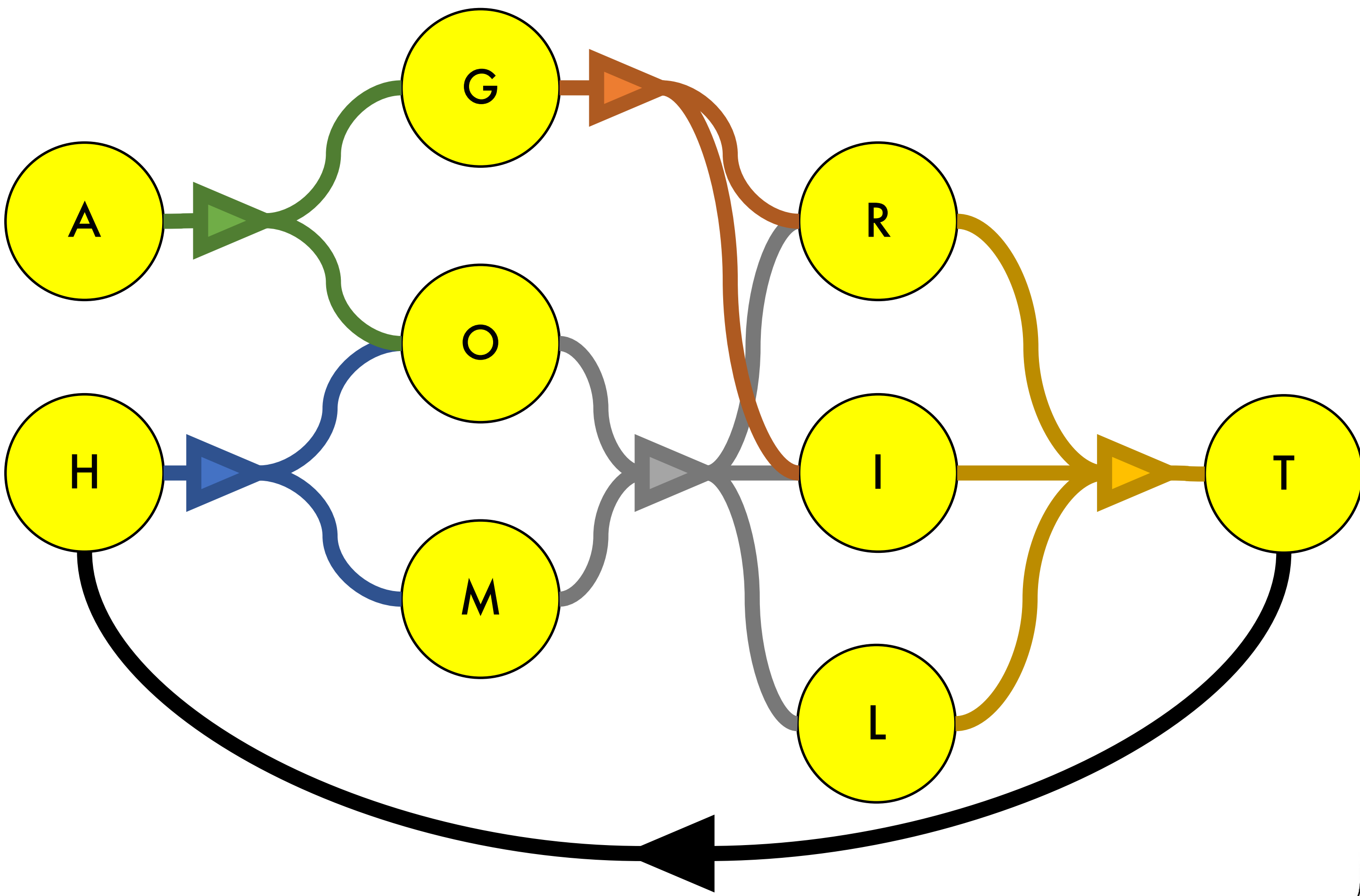
$$\{A, L\}^+ = \{A, L, G, O, R, I, T, H, M\}$$

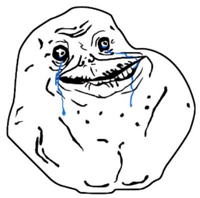




Think About This

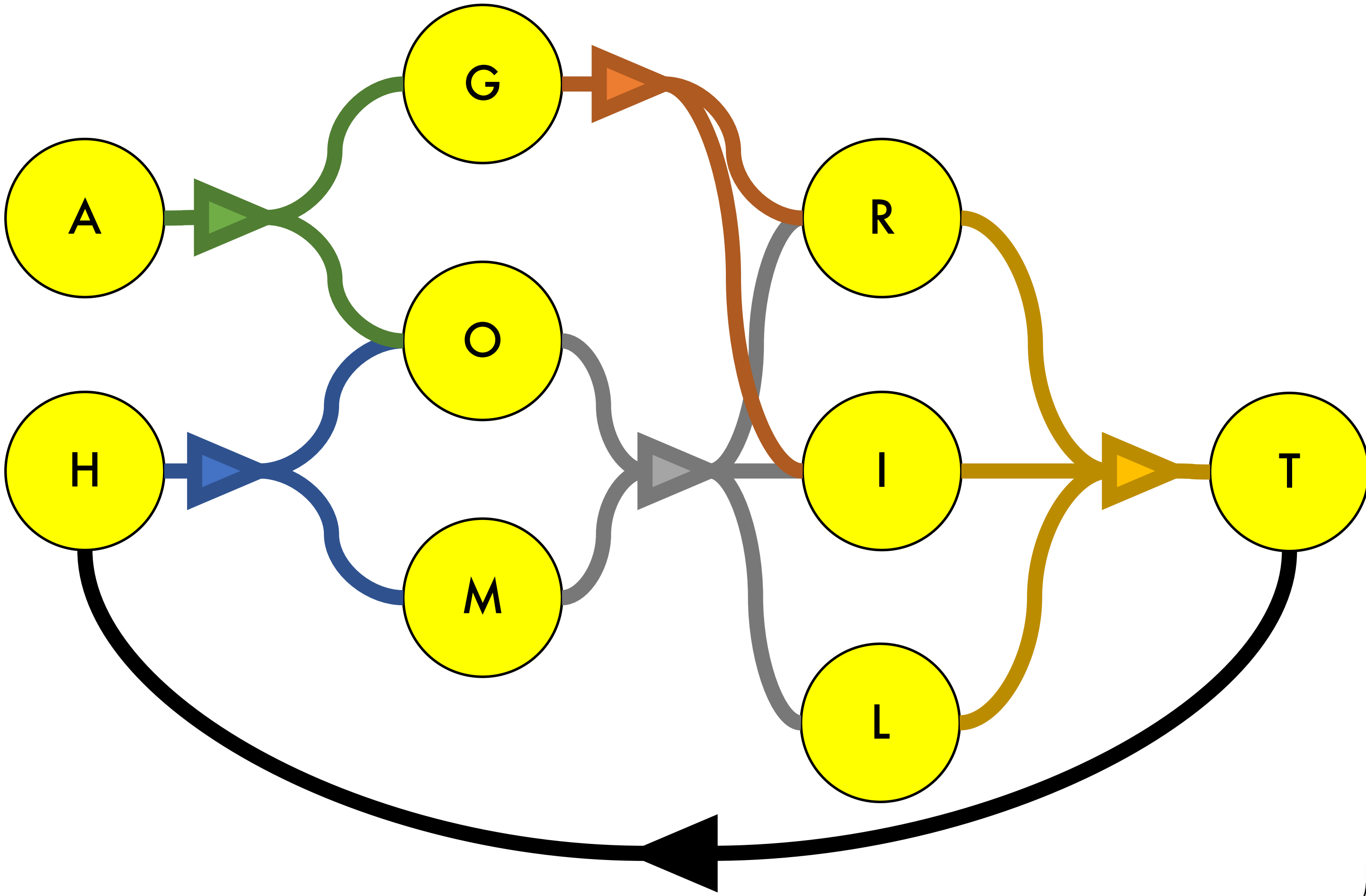
$\{A, L\}$ determines all attributes





Think About This

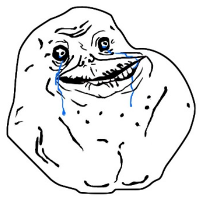
$\{A, L\}$ is a (super)key!



Finding Keys

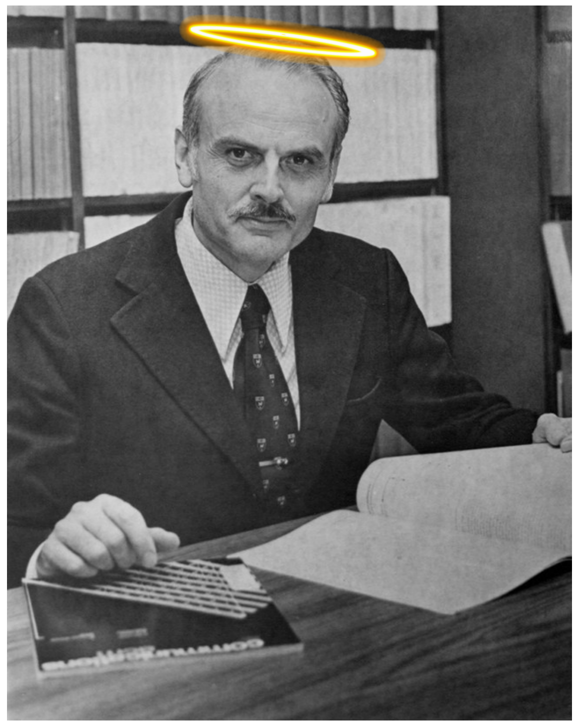
Definition – Prime Attribute

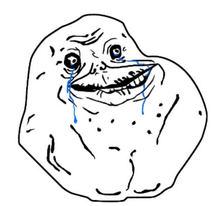
An attribute is a **Prime Attribute** if it is part of a candidate key, otherwise the attribute is considered **Nonprime**



Normal Forms

"The key (1NF), the whole key (2NF), and nothing but the key (3NF), so help me Codd."





Normal Forms

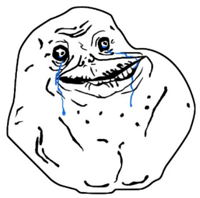
Definition – Full Functional Dependency

In the functional dependency $X \rightarrow A$, an attribute A is **Fully Functionally Dependent** on X if there is no subset Y of X in which $Y \rightarrow A$ holds.

Otherwise, if there is some Y where $Y \rightarrow A$ holds, A is **Partially Dependent** on X.

Definition – Second Normal Form (2NF)

A relation R is in **Second Normal Form** if it is in 1NF and all nonprime attributes are fully functionally dependent on the primary key of R.



Normal Forms

Definition – Third Normal Form (3NF)

A relation R is in **Third Normal Form** if it is in 2NF and if for all non-trivial FDs, $X \rightarrow A$, X is a superkey or A contains only prime attributes