FORMAL METHODS FOR DATABASES

ZACHARY MORING

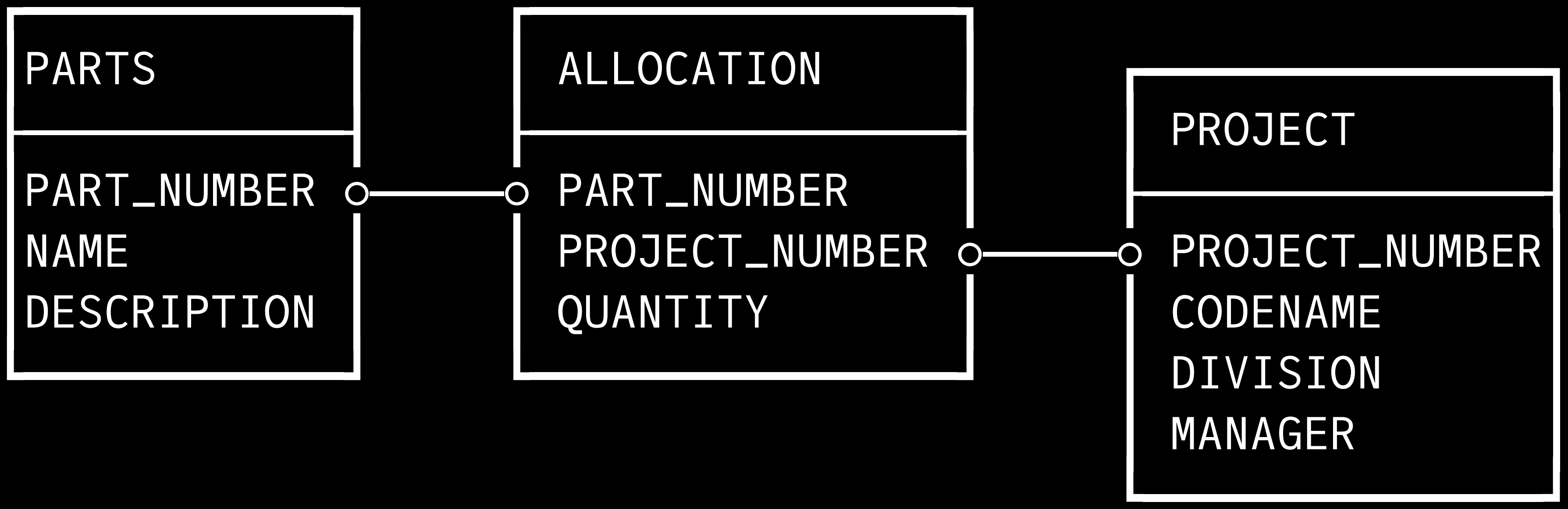
FORMAL METHODS FOR DATABASES

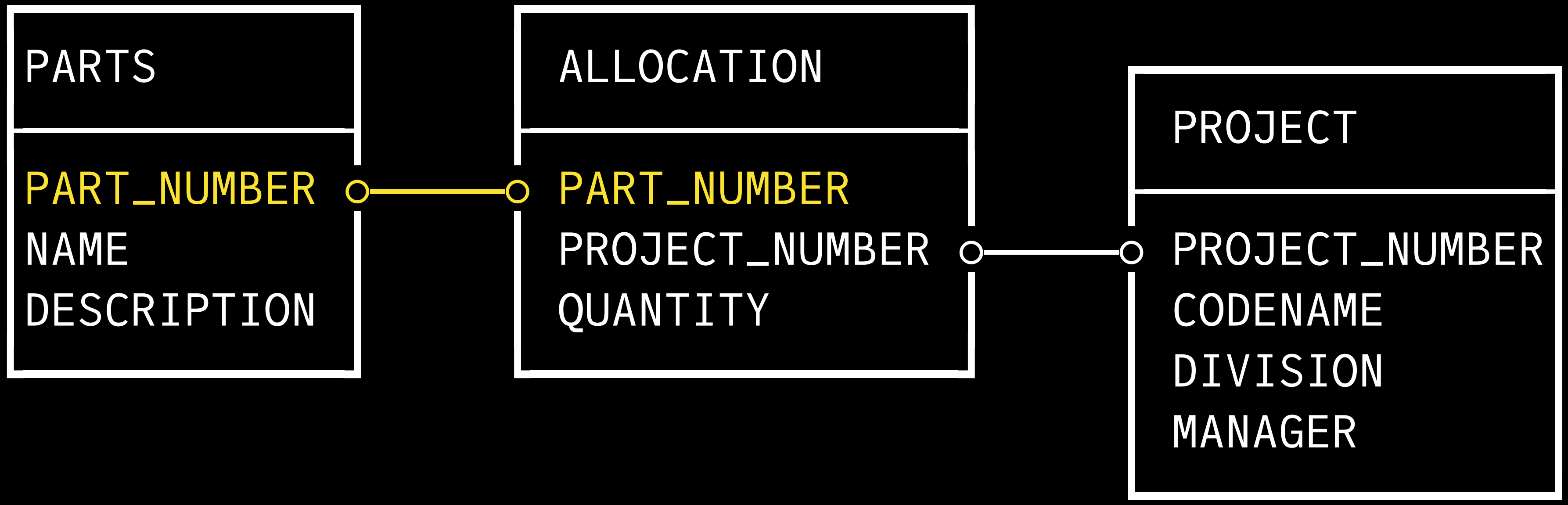
CAS–CS–511
FALL 2024

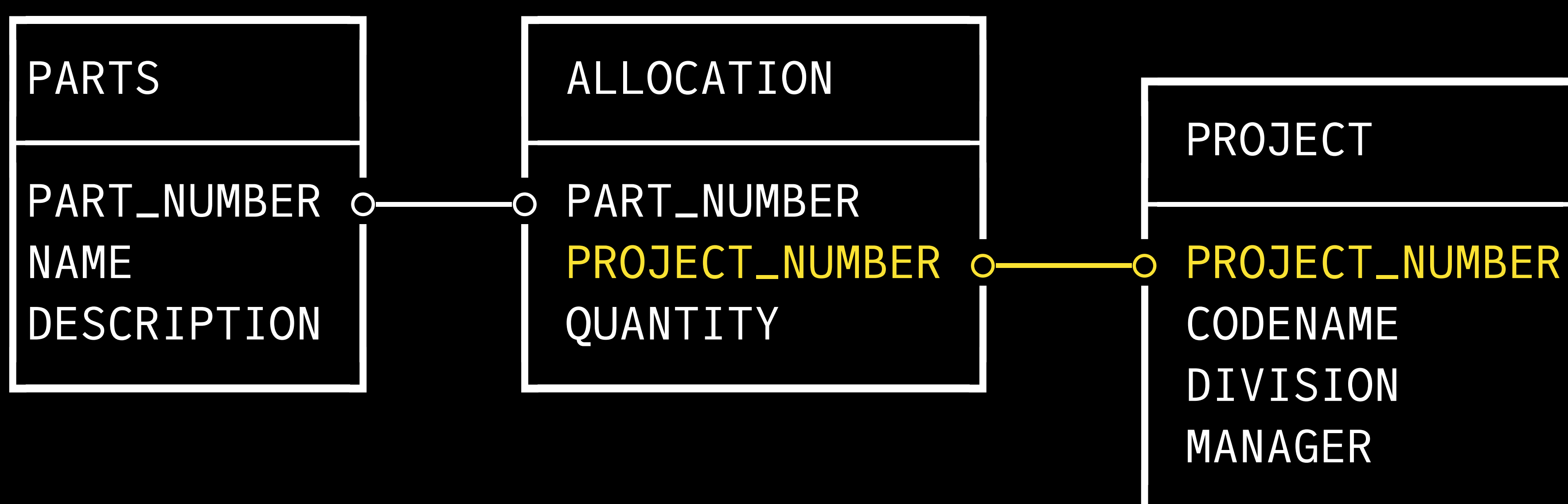
- Introduction to formal database theory
  - Relational algebra
  - Relational calculus
- Introduction to category theory
  - Categories
  - Functors
  - Database migrations

DATABASE THEORY

CODD 1970


PARTS
PART_NUMBER
NAME
DESCRIPTION
ALLOCATION
PART_NUMBER
PROJECT_NUMBER
QUANTITY
PROJECT
PROJECT_NUMBER
CODENAME
DIVISION
MANAGER


PARTS
PART_NUMBER
NAME
DESCRIPTION
ALLOCATION
PART_NUMBER
PROJECT_NUMBER
QUANTITY
PROJECT
PROJECT_NUMBER
CODENAME
DIVISION
MANAGER


PARTS
PART_NUMBER
NAME
DESCRIPTION
ALLOCATION
PART_NUMBER
PROJECT_NUMBER
QUANTITY
PROJECT
PROJECT_NUMBER
CODENAME
DIVISION
MANAGER

Introduction to Database Theory

CODD 1970

	Number	String	Number	UUID
	PROJECT_NUMBER	CODENAME	DIVISION	MANAGER
i_1				
i_2				
i_2				

DOMAIN	(Possibly infinite) Set of possible values	⟷	TYPE
TUPLE	$(d_1 \in D_1, \ldots, d_n \in D_n)$	⟷	ROW
RELATION	Set of tuples; all *i*th domains are the same	⟷	TABLE

	Number	String	Number	UUID
	PROJECT_NUMBER	CODENAME	DIVISION	MANAGER
i_1				
i_2				
i_2				

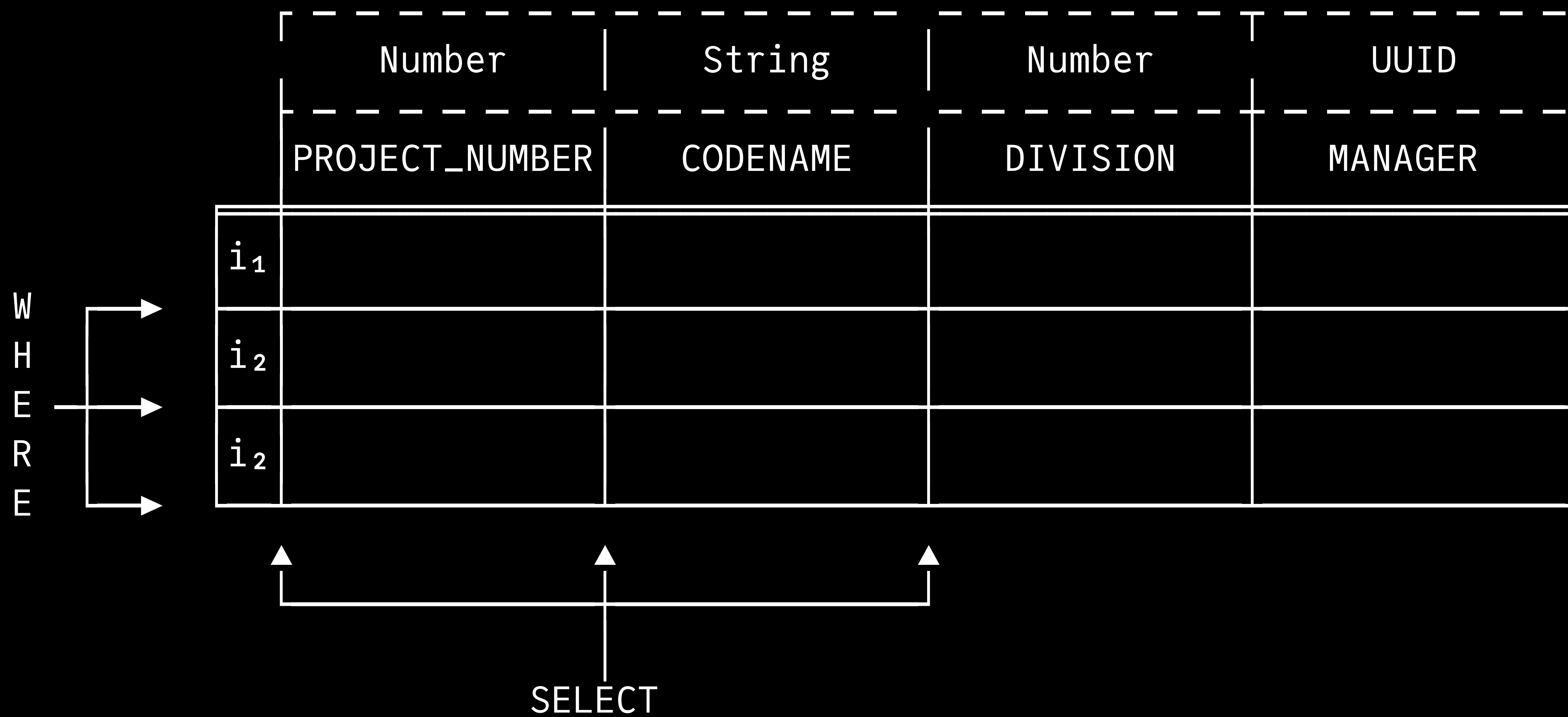
DOMAIN	(Possibly infinite) Set of possible values	↔	TYPE
TUPLE	$(d_1 \in D_1, \ldots, d_n \in D_n)$	↔	ROW
RELATION	Set of tuples; all *i*th domains are the same	↔	TABLE

	Number	String	Number	UUID
	PROJECT_NUMBER	CODENAME	DIVISION	MANAGER
i_1				
i_2				
i_2				

DOMAIN	(Possibly infinite) Set of possible values	↔	TYPE
TUPLE	$(d_1 \in D_1, \ldots, d_n \in D_n)$	↔	ROW
RELATION	Set of tuples; all *i*th domains are the same	↔	TABLE

	Number	String	Number	UUID
	PROJECT_NUMBER	CODENAME	DIVISION	MANAGER
i_1				
i_2				
i_2				

DOMAIN	(Possibly infinite) Set of possible values	⟷	TYPE
TUPLE	$(d_1 \in D_1, \ldots, d_n \in D_n)$	⟷	ROW
RELATION	Set of tuples; all *ith* domains are the same	⟷	TABLE



$$\text{Result} = \{\ (d_{sel1},\ d_{sel2},\ \ldots) : \text{"WHERE" condition is true}\ \}$$

```sql
SELECT
  P.MANAGER
FROM
  PROJECTS AS P
WHERE
  P.DIVISION = 4
  AND NOT EXISTS (
      SELECT 'x'
      FROM ALLOCATIONS AS A
      WHERE (A.PROJECT_NUMBER
        = P.PROJECT_NUMBER)
      AND A.QUANTITY > 0
  )
;
```

```
{
  P[MANAGER] :
  PROJECTS(P)
  AND P[DIVISION] = 4
  AND ¬∃x ∈ ALLOCATIONS(A) (
    P[PROJECT_NUMBER]
    = x[PROJECT_NUMBER]
    AND x.QUANTITY > 0
  )
}
```

```sql
SELECT
  P.MANAGER
FROM
  PROJECTS AS P
WHERE
  P.DIVISION = 4
  AND NOT EXISTS (
      SELECT 'x'
      FROM ALLOCATIONS AS A
      WHERE (A.PROJECT_NUMBER
        = P.PROJECT_NUMBER)
      AND A.QUANTITY > 0
  )
;
```

```
{
  P[MANAGER] :
  PROJECTS(P)
  AND P[DIVISION] = 4
  AND ¬∃x ∈ ALLOCATIONS(A) (
    P[PROJECT_NUMBER]
    = x[PROJECT_NUMBER]
    AND x.QUANTITY > 0
  )
}
```

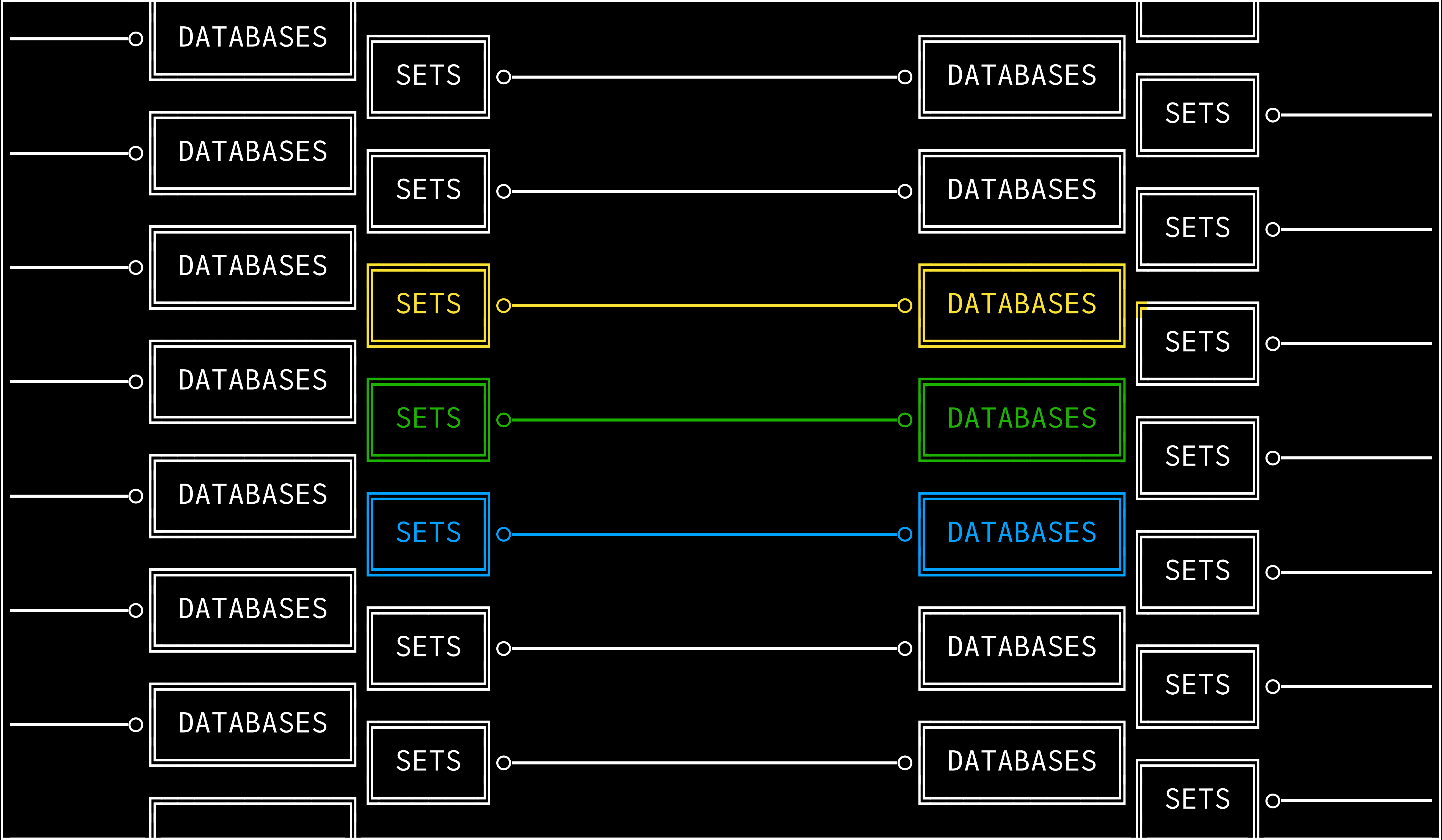
```sql
SELECT
  P.MANAGER
FROM
  PROJECTS AS P
WHERE
  P.DIVISION = 4
  AND NOT EXISTS (
      SELECT 'x'
      FROM ALLOCATIONS AS A
      WHERE (A.PROJECT_NUMBER
        = P.PROJECT_NUMBER)
      AND A.QUANTITY > 0
  )
;
```

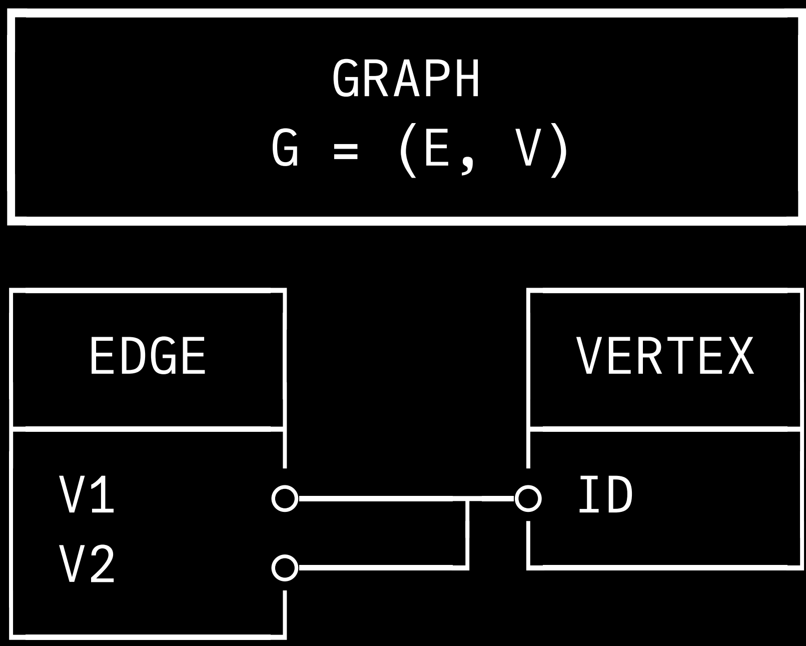
```
{
  P[MANAGER] :
  PROJECTS(P)
  AND P[DIVISION] = 4
  AND ¬∃x ∈ ALLOCATIONS(A) (
    P[PROJECT_NUMBER]
    = x[PROJECT_NUMBER]
    AND x.QUANTITY > 0
  )
}
```

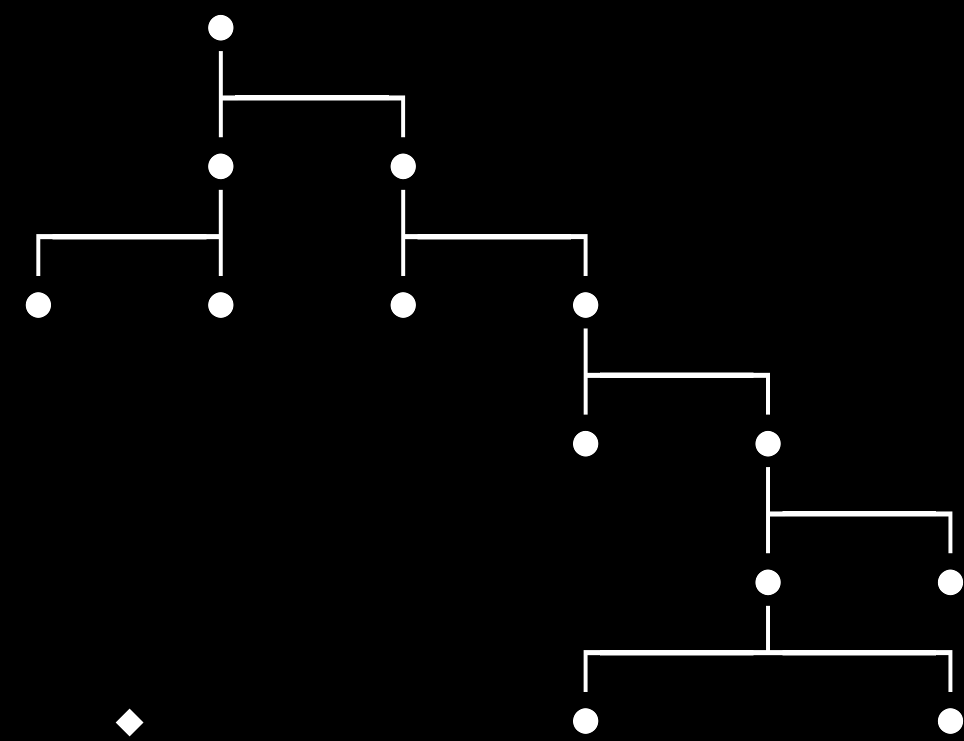
```
SELECT
  P.MANAGER
FROM
  PROJECTS AS P
WHERE
  P.DIVISION = 4
  AND NOT EXISTS (
      SELECT 'x'
      FROM ALLOCATIONS AS A
      WHERE (A.PROJECT_NUMBER
        = P.PROJECT_NUMBER)
      AND A.QUANTITY > 0
  )
;
```

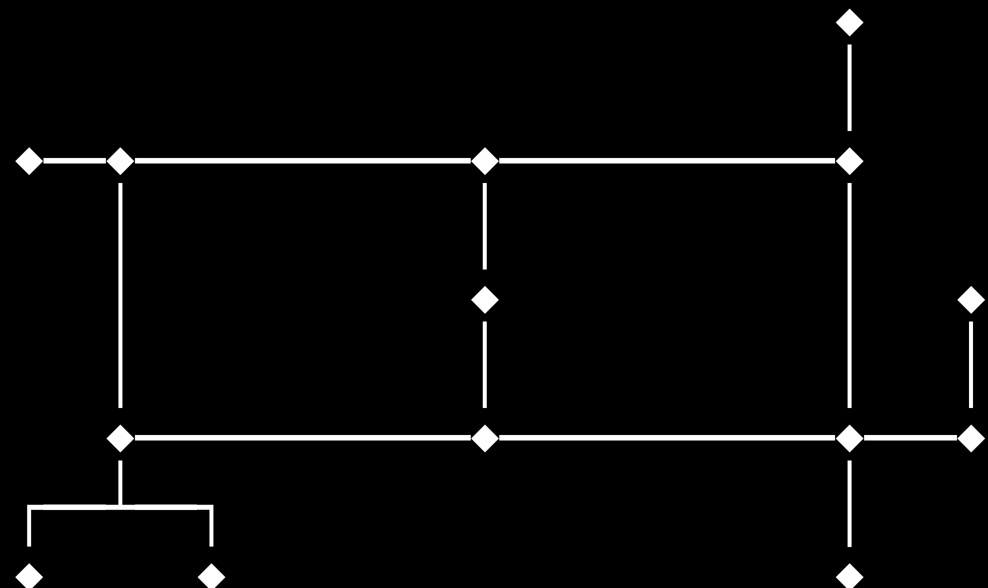
```
{
  P[MANAGER] :
  PROJECTS(P)
  AND P[DIVISION] = 4
  AND ¬∃x ∊ ALLOCATIONS(A) (
    P[PROJECT_NUMBER]
    = x[PROJECT_NUMBER]
    AND x.QUANTITY > 0
  )
}
```

```
φ(k) := k ∊ PROJECTS
  ∧ k[DIVISION] = 4
  ∧ ¬∃(x ∊ ALLOCATIONS) (...)
```


DATABASES
DATABASES
DATABASES
DATABASES
DATABASES
DATABASES
DATABASES
SETS
SETS
SETS
SETS
SETS
SETS
SETS
DATABASES
DATABASES
DATABASES
DATABASES
DATABASES
DATABASES
DATABASES
SETS
SETS
SETS
SETS
SETS
SETS
SETS

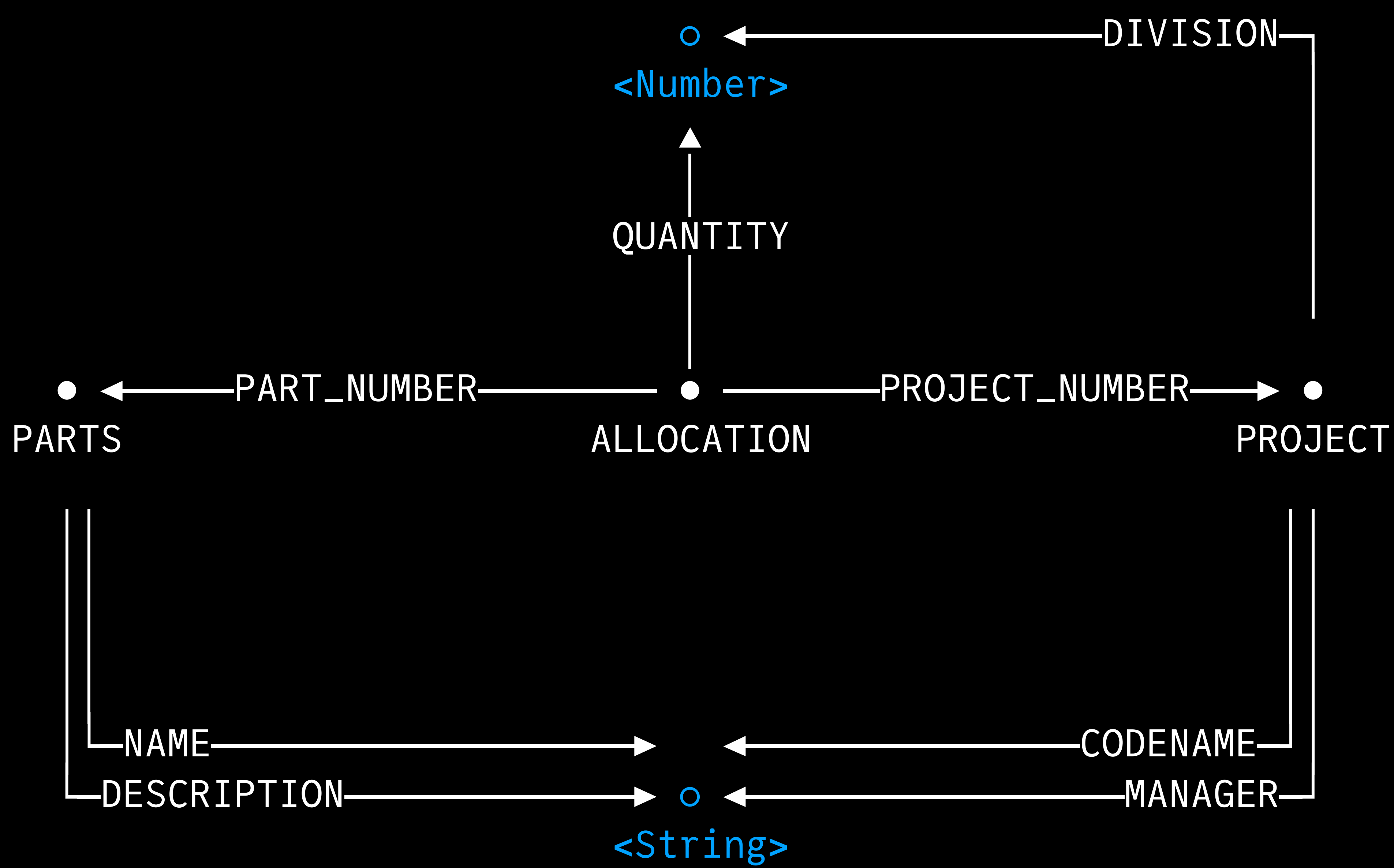

GRAPH
G = (E, V)
EDGE
V1
V2
VERTEX
ID



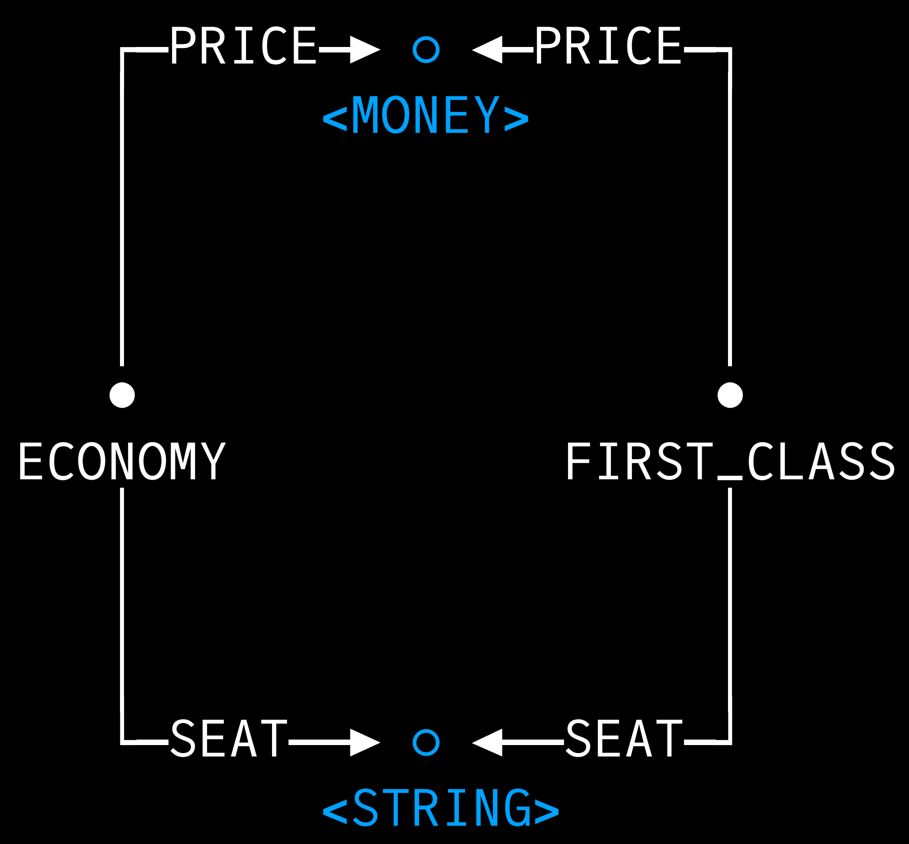


CATEGORY THEORY

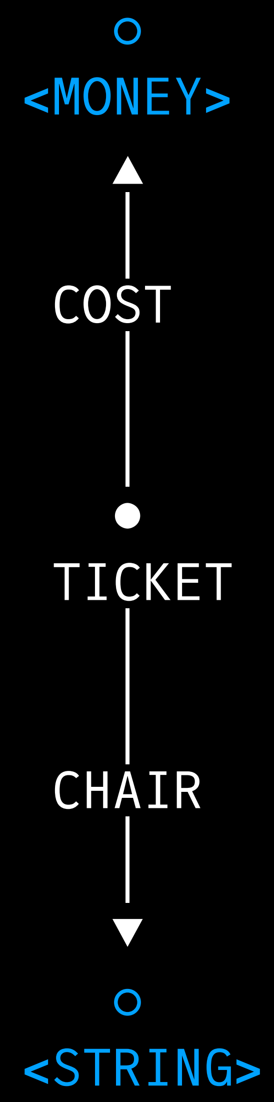
FONG, SPIVAK 2018



THE VOEWING COMPANY



USONIAN AIRLINES



SPECTER AIRLINES

SET ◆—◆ TABLE

FUNCTIONS ◆—◆ FOREIGN KEY

{Mercury,
Venus,
Earth,
Mars,
Jupiter,
Saturn,
Uranus,
Neptune}

PLANET
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune

$f : E \to P$

ELEMENT	PLANET
WATER	MERCURY
WOOD	JUPITER
FIRE	MARS
METAL	VENUS
EARTH	SATURN

● SET

● DOMAIN ——FUNCTION——→ ● CODOMAIN

03/nn	Category Theory and Databases	FONG, SPIVAK 2018

SET ◆——◆ **TABLE**

{Mercury,
Venus,
Earth,
Mars,
Jupiter,
Saturn,
Uranus,
Neptune}

PLANET
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune

FUNCTIONS ◆——◆ **FOREIGN KEY**

$f : E \to P$

ELEMENT	PLANET
WATER	MERCURY
WOOD	JUPITER
FIRE	MARS
METAL	VENUS
EARTH	SATURN

● SET

$\text{DOMAIN} \xrightarrow{\text{FUNCTION}} \text{CODOMAIN}$

SET ◆—◆ TABLE

{Mercury,
Venus,
Earth,
Mars,
Jupiter,
Saturn,
Uranus,
Neptune}

PLANET
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune

FUNCTIONS ◆—◆ FOREIGN KEY

$f : E \to P$

ELEMENT	PLANET
WATER	MERCURY
WOOD	JUPITER
FIRE	MARS
METAL	VENUS
EARTH	SATURN

● SET

● DOMAIN —FUNCTION→ ● CODOMAIN

SET ⟷ TABLE

{Mercury,
Venus,
Earth,
Mars,
Jupiter,
Saturn,
Uranus,
Neptune}

PLANET
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune

FUNCTIONS ⟷ FOREIGN KEY

$f : E \to P$

ELEMENT	PLANET
WATER	MERCURY
WOOD	JUPITER
FIRE	MARS
METAL	VENUS
EARTH	SATURN

• SET

DOMAIN $\xrightarrow{\text{FUNCTION}}$ CODOMAIN

SET ◆——◆ TABLE

FUNCTIONS ◆——◆ FOREIGN KEY

{Mercury,
Venus,
Earth,
Mars,
Jupiter,
Saturn,
Uranus,
Neptune}

PLANET
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune

$f : E \to P$

ELEMENT	PLANET
WATER	MERCURY
WOOD	JUPITER
FIRE	MARS
METAL	VENUS
EARTH	SATURN

• SET

DOMAIN • ——FUNCTION——→ • CODOMAIN

Foreign Keys

WELL-DEFINED
Referential integrity
"Enumerated" options

INJECTIVE
"One-to-one relation"

SURJECTIVE
A "complete" reference

BIJECTIVE
Perfect correspondence

FUNCTIONS ◆———◆ FOREIGN KEY

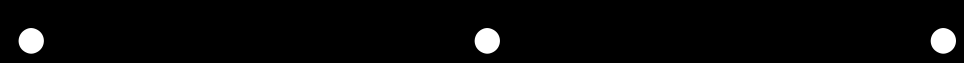
$f : E \to P$

ELEMENT	PLANET
WATER	MERCURY
WOOD	JUPITER
FIRE	MARS
METAL	VENUS
EARTH	SATURN

$$\underset{\text{DOMAIN}}{\bullet} \xrightarrow{\text{FUNCTION}} \underset{\text{CODOMAIN}}{\bullet}$$

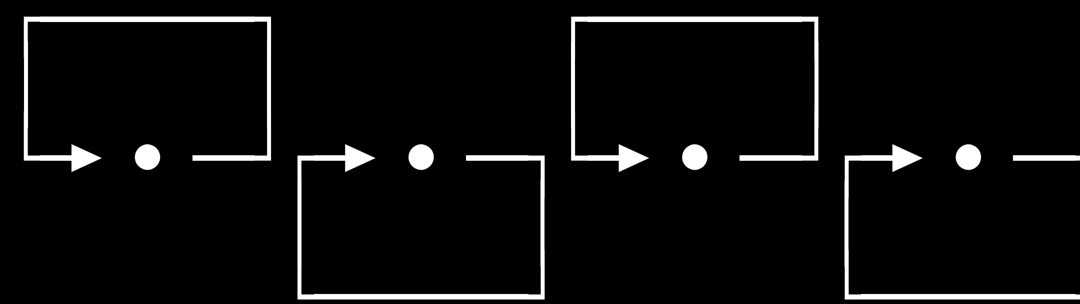
OBJECTS

Ob($\mathcal{C}$), elements of $\mathcal{C}$
[These guys are dots in $\mathcal{C}$]



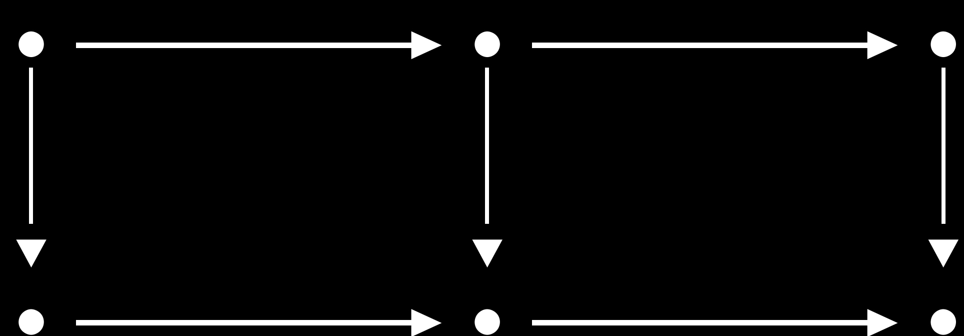
IDENTITY

$\forall c \in \mathcal{C}$ ($\exists$ id, id $\in$ Hom(c, c))



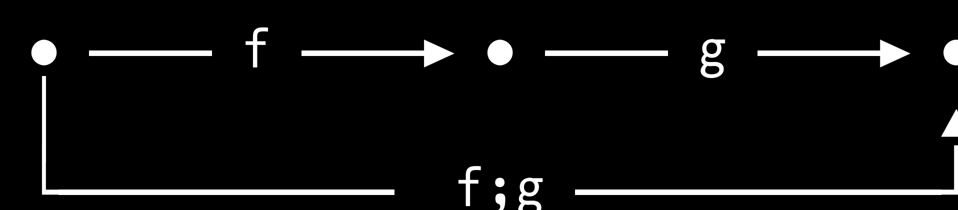
(HOMO)MORPHISMS

$\mathcal{C}$(c, d) = Hom(c, d) = "Homset" of $\mathcal{C}$
[These guys are "arrows" in $\mathcal{C}$]



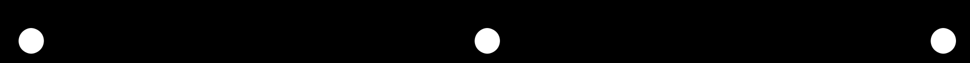
COMPOSITION

If c,d,e in $\mathcal{C}$ and f $\in$ $\mathcal{C}$(c,d), g $\in$ $\mathcal{C}$(d,e)
exists some f;g $\in$ $\mathcal{C}$(c,e)
such that f;g is the same as f then g



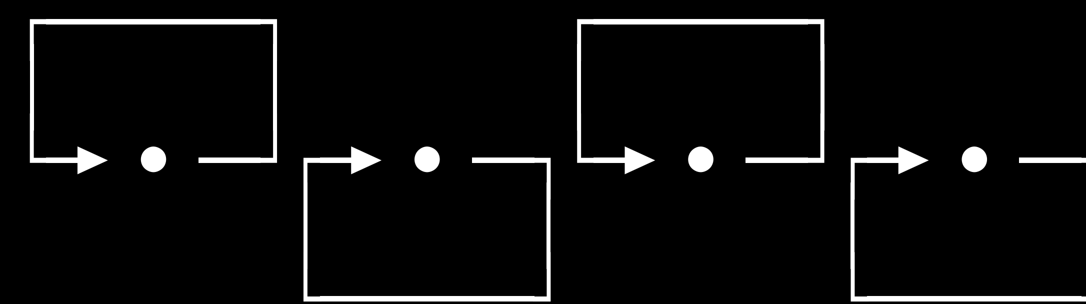
OBJECTS

Ob($\mathcal{C}$), elements of $\mathcal{C}$
[These guys are dots in $\mathcal{C}$]



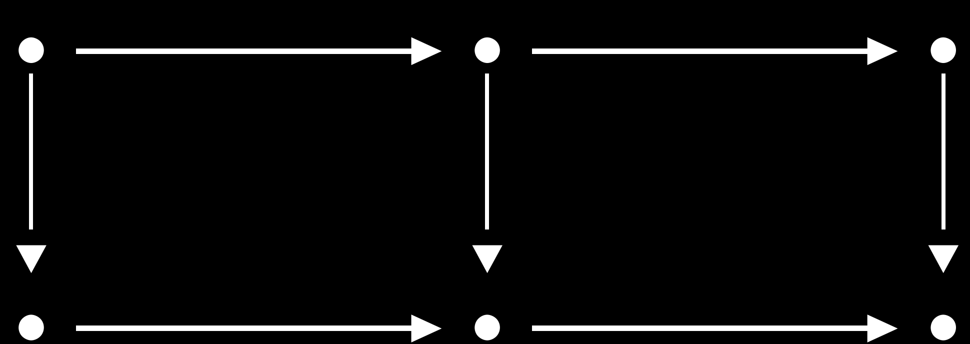
IDENTITY

$\forall c \in \mathcal{C}$ ($\exists$ id, id $\in$ Hom(c, c))



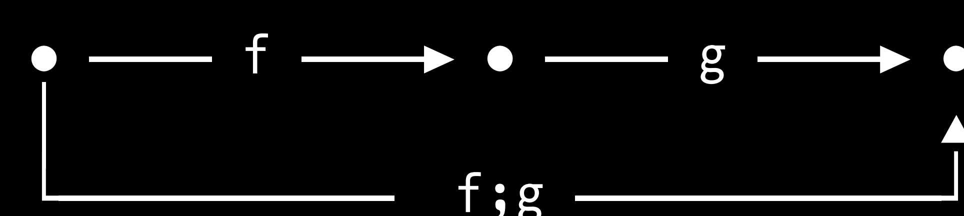
(HOMO)MORPHISMS

$\mathcal{C}$(c, d) = Hom(c, d) = "Homset" of $\mathcal{C}$
[These guys are "arrows" in $\mathcal{C}$]



COMPOSITION

If c,d,e in $\mathcal{C}$ and f $\in \mathcal{C}$(c,d), g $\in \mathcal{C}$(d,e)
exists some f;g $\in \mathcal{C}$(c,e)
such that f;g is the same as f then g

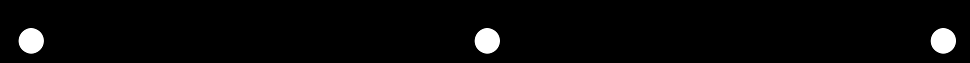


Category Theory and Databases

FONG, SPIVAK 2018

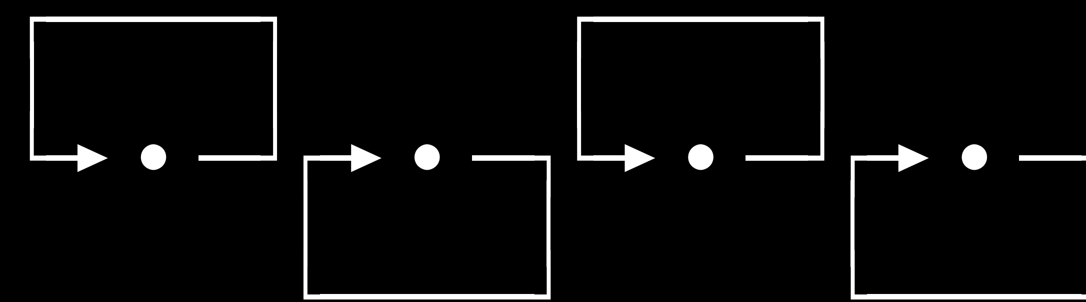
OBJECTS

Ob($\mathcal{C}$), elements of $\mathcal{C}$
[These guys are dots in $\mathcal{C}$]



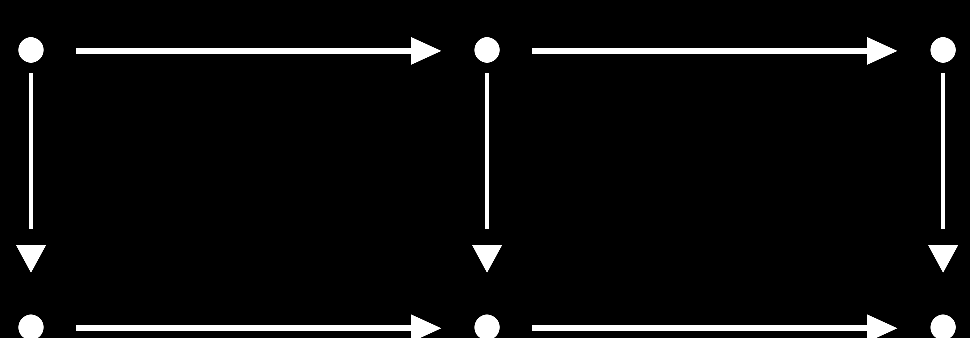
IDENTITY

$\forall c \in \mathcal{C}\ (\exists\ id,\ id \in \mathrm{Hom}(c,\ c))$



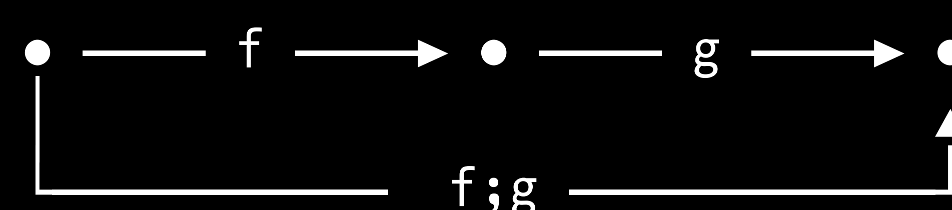
(HOMO)MORPHISMS

$\mathcal{C}$(c, d) = Hom(c, d) = "Homset" of $\mathcal{C}$
[These guys are "arrows" in $\mathcal{C}$]



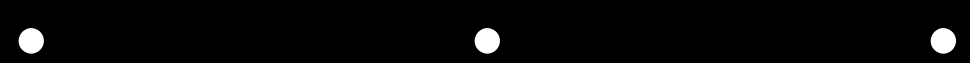
COMPOSITION

If c,d,e in $\mathcal{C}$ and f $\in$ $\mathcal{C}$(c,d), g $\in$ $\mathcal{C}$(d,e)
exists some f;g $\in$ $\mathcal{C}$(c,e)
such that f;g is the same as f then g



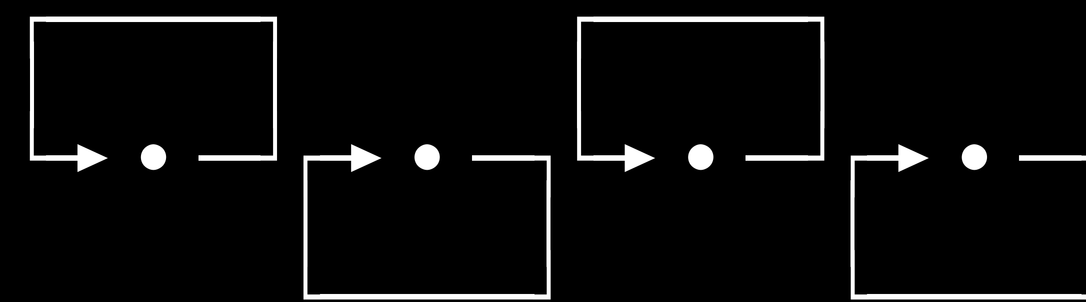
OBJECTS

Ob($\mathcal{C}$), elements of $\mathcal{C}$
[These guys are dots in $\mathcal{C}$]



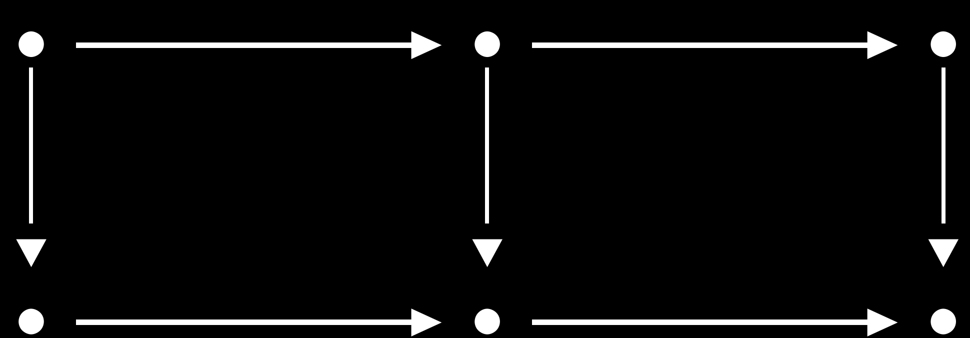
IDENTITY

$\forall c \in \mathcal{C}$ ($\exists$ id, id $\in$ Hom(c, c))



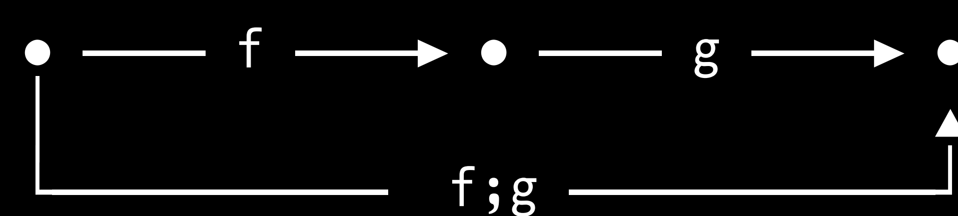
(HOMO)MORPHISMS

$\mathcal{C}$(c, d) = Hom(c, d) = "Homset" of $\mathcal{C}$
[These guys are "arrows" in $\mathcal{C}$]



COMPOSITION

If c,d,e in $\mathcal{C}$ and f $\in$ $\mathcal{C}$(c,d), g $\in$ $\mathcal{C}$(d,e)
exists some f;g $\in$ $\mathcal{C}$(c,e)
such that f;g is the same as f then g



OBJECTS

Ob($\mathcal{C}$), elements of $\mathcal{C}$
[These guys are dots in $\mathcal{C}$]

$$\begin{tikzcd} \bullet & \bullet & \bullet \end{tikzcd}$$

IDENTITY

$\forall c \in \mathcal{C}$ ($\exists$ id, id $\in$ Hom(c, c))

$$\begin{tikzcd} \bullet \arrow[loop above] & \bullet \arrow[loop below] & \bullet \arrow[loop above] & \bullet \arrow[loop below] \end{tikzcd}$$

(HOMO)MORPHISMS

$\mathcal{C}$(c, d) = Hom(c, d) = "Homset" of $\mathcal{C}$
[These guys are "arrows" in $\mathcal{C}$]

$$\begin{tikzcd} \bullet \arrow[r] \arrow[d] & \bullet \arrow[r] \arrow[d] & \bullet \arrow[d] \\ \bullet \arrow[r] & \bullet \arrow[r] & \bullet \end{tikzcd}$$

COMPOSITION

If c,d,e in $\mathcal{C}$ and f $\in \mathcal{C}$(c,d), g $\in \mathcal{C}$(d,e)
exists some f;g $\in \mathcal{C}$(c,e)
such that f;g is the same as f then g

$$\begin{tikzcd} \bullet \arrow[r, "f"] \arrow[rr, bend right, "f;g"'] & \bullet \arrow[r, "g"] & \bullet \end{tikzcd}$$

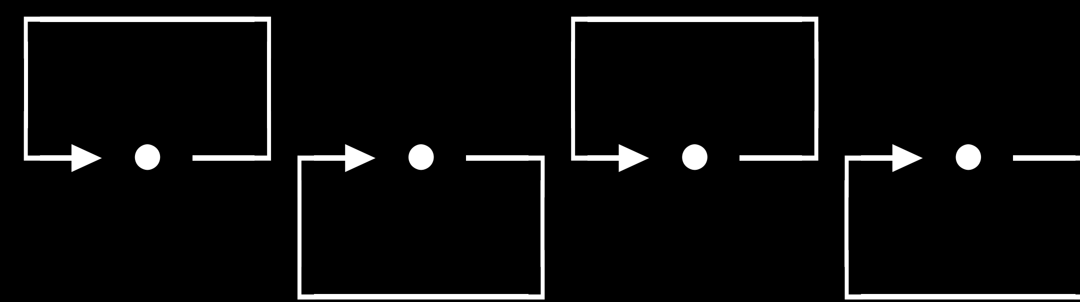
Since identity and composition are universal, we often don't write them.

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "h"] \\ C \arrow[r, "i"'] & D \end{tikzcd}$$

POP QUIZ: What are the 10 morphisms?

IDENTITY

$\forall c \in \mathcal{C}\ (\exists\ id,\ id \in Hom(c,\ c))$



COMPOSITION

If c,d,e in $\mathcal{C}$ and $f \in \mathcal{C}(c,d)$, $g \in \mathcal{C}(d,e)$
exists some $f;g \in \mathcal{C}(c,e)$
such that $f;g$ is the same as f then g

$$\begin{tikzcd} \bullet \arrow[r, "f"] \arrow[rr, bend right, "f;g"'] & \bullet \arrow[r, "g"] & \bullet \end{tikzcd}$$

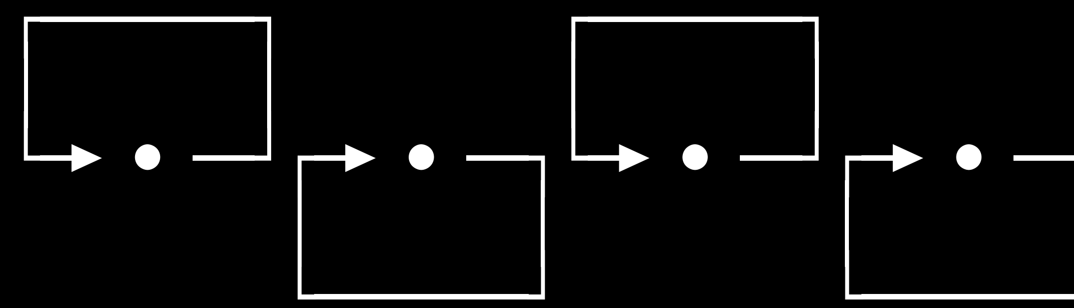
> Since identity and composition are universal, we often don't write them.

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "h"] \\ C \arrow[r, "i"'] & D \end{tikzcd}$$

POP QUIZ: What are the 10 morphisms?

IDENTITY

$\forall c \in \mathcal{C}\ (\exists\ \mathrm{id},\ \mathrm{id} \in \mathrm{Hom}(c,\ c))$



COMPOSITION

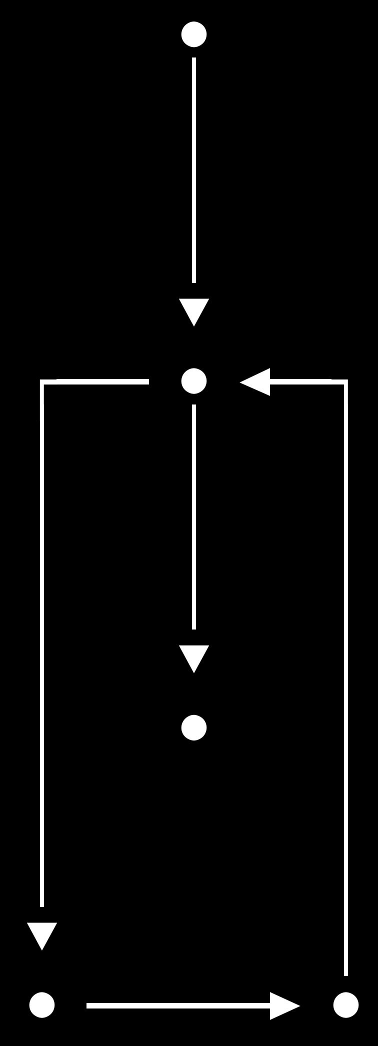
If c,d,e in $\mathcal{C}$ and $f \in \mathcal{C}(c,d)$, $g \in \mathcal{C}(d,e)$
exists some $f;g \in \mathcal{C}(c,e)$
such that $f;g$ is the same as f then g

$$\begin{tikzcd} \bullet \arrow[r, "f"] \arrow[rr, bend right, "f;g"'] & \bullet \arrow[r, "g"] & \bullet \end{tikzcd}$$

Any graph G can be made into a category

1. Make it reflexive
2. All vertices become objects in Ob($\mathcal{C}$)
3. All paths become morphisms

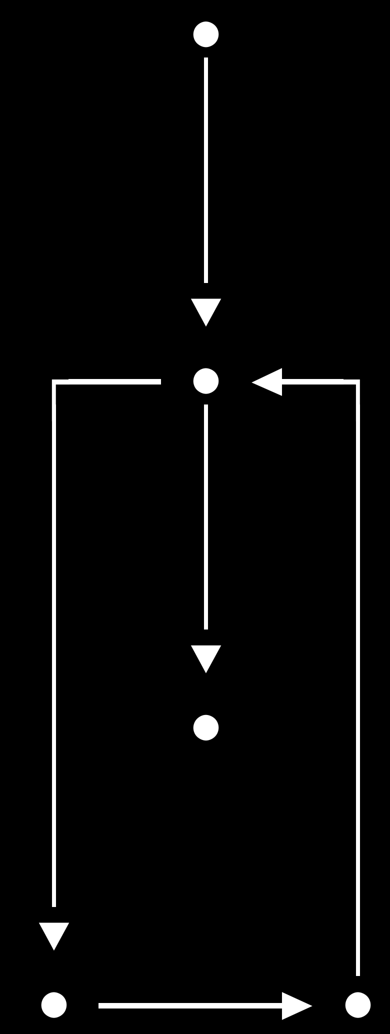
This is called the "free category" on G



Any graph G can be made into a category

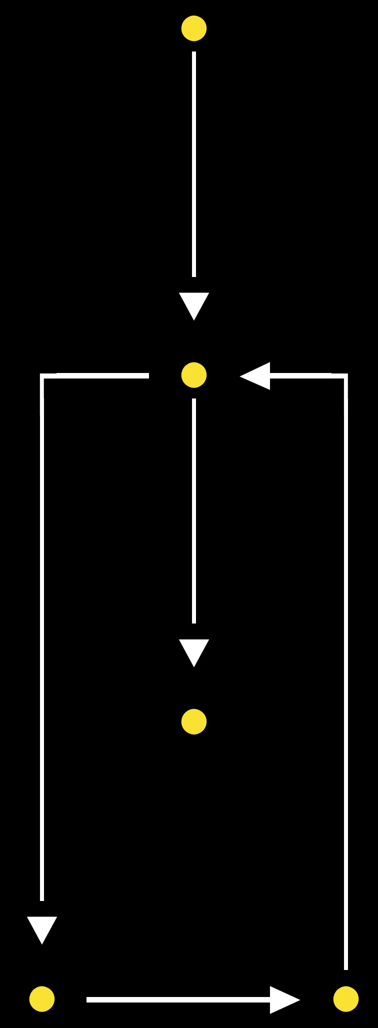
1. Make it reflexive
2. All vertices become objects in Ob($\mathcal{C}$)
3. All paths become morphisms

This is called the "free category" on G



Any graph G can be made into a category

1. Make it reflexive
2. All vertices become objects in Ob($\mathcal{C}$)
3. All paths become morphisms

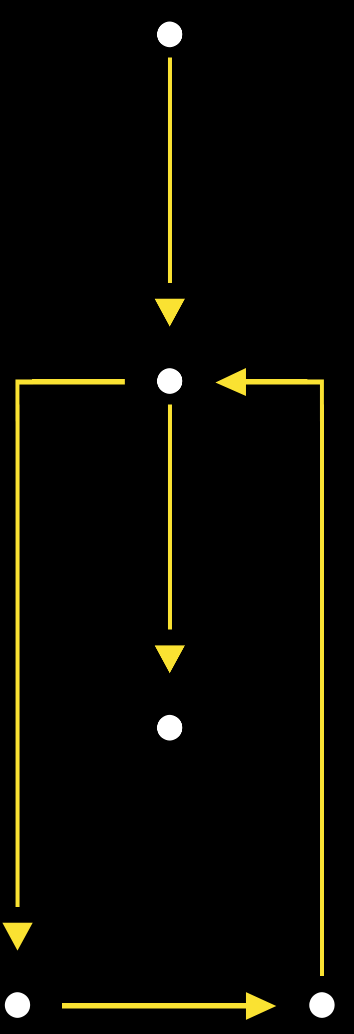


This is called the "free category" on G

Any graph G can be made into a category

1. Make it reflexive
2. All vertices become objects in Ob($\mathcal{C}$)
3. All paths become morphisms

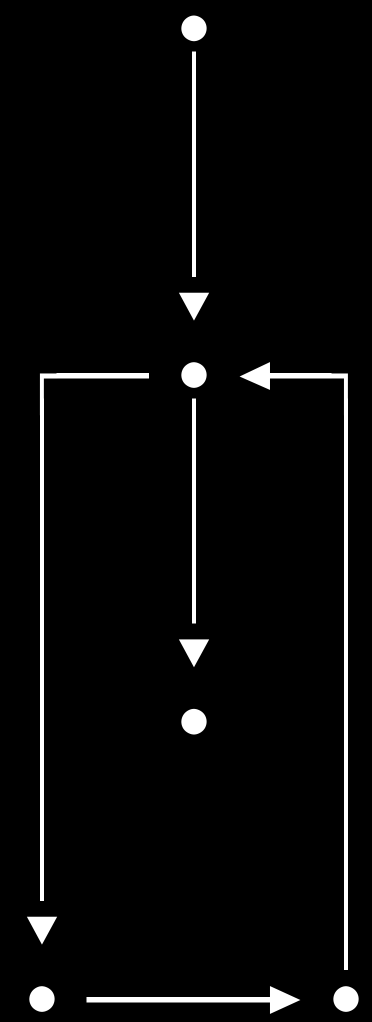
This is called the "free category" on G



Any graph G can be made into a category

1. Make it reflexive
2. All vertices become objects in Ob($\mathcal{C}$)
3. All paths become morphisms

This is called the "free category" on G



The category **Set**

1. Ob(**Set**) is all sets
2. The morphisms are all (proper) functions between sets
3. Each set has an identity that sends every element to itself
4. Composition of morphisms is composition of functions

$$\begin{tikzcd}
\mathbb{B} = \{\mathrm{True}, \mathrm{False}\} \arrow[r, "b \mapsto \mathrm{int}(b)+1"] & \mathbb{N} \arrow[l, bend right, "n \mapsto !\mathrm{isodd}(n)"'] \arrow[r, "n \mapsto 3n"] \arrow[r, bend right, "n \mapsto \ln(n)"'] & \mathbb{R}
\end{tikzcd}$$

The category **Set**

1. Ob(**Set**) is all sets
2. The morphisms are all (proper) functions between sets
3. Each set has an identity that sends every element to itself
4. Composition of morphisms is composition of functions

$$\begin{tikzcd} \mathbb{B} \arrow[r, "b \mapsto \mathrm{int}(b)+1"] & \mathbb{N} \arrow[l, bend right, "n \mapsto !\mathrm{isodd}(n)"'] \arrow[r, "n \mapsto 3n"] \arrow[r, bend right, "n \mapsto \ln(n)"'] & \mathbb{R} \end{tikzcd}$$

$\mathbb{B} = \{\mathrm{True}, \mathrm{False}\}$

The category **Set**

1. Ob(**Set**) is all sets
2. The morphisms are all (proper) functions between sets
3. Each set has an identity that sends every element to itself
4. Composition of morphisms is composition of functions

$$\begin{tikzcd} \bullet \arrow[r, "b \mapsto \mathrm{int}(b)+1"] & \overset{\mathbb{N}}{\bullet} \arrow[l, bend right, "n \mapsto !\mathrm{isodd}(n)"'] \arrow[r, "n \mapsto 3n"] \arrow[r, bend right, "n \mapsto \ln(n)"'] & \overset{\mathbb{R}}{\bullet} \end{tikzcd}$$

$\mathbb{B}$ = {True, False}

The category **Set**

1. Ob(**Set**) is all sets
2. The morphisms are all (proper) functions between sets
3. Each set has an identity that sends every element to itself
4. Composition of morphisms is composition of functions

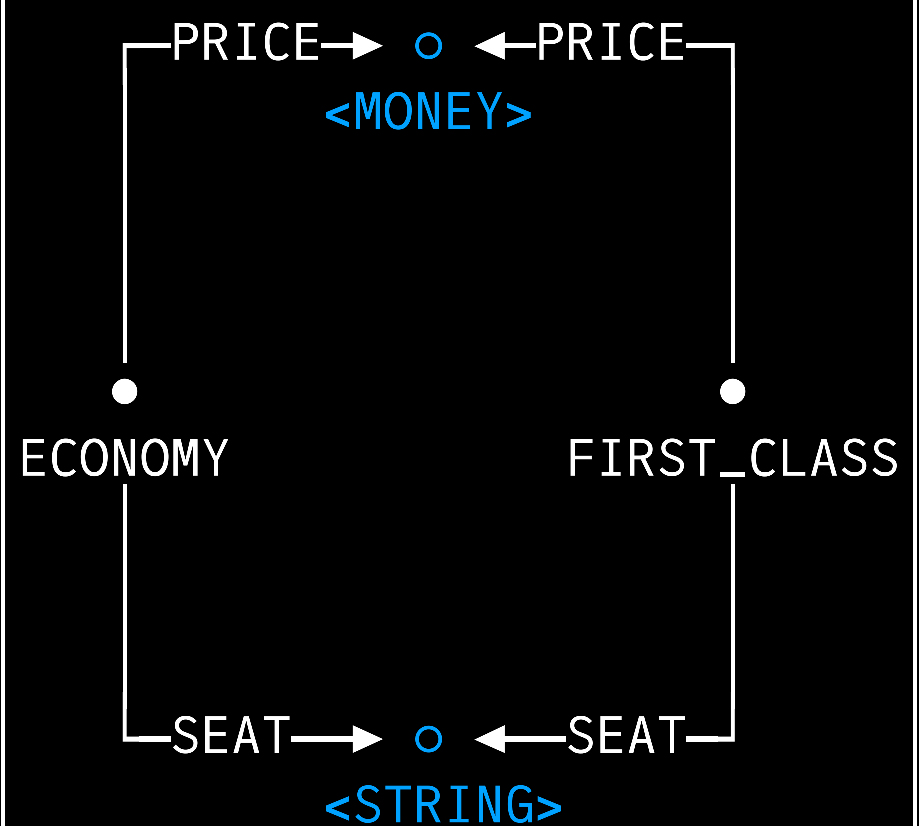
$$\begin{tikzcd}
\mathbb{B} = \{\mathrm{True}, \mathrm{False}\} \arrow[r, "b \mapsto \mathrm{int}(b)+1"] & \mathbb{N} \arrow[l, bend right, "n \mapsto !\mathrm{isodd}(n)"'] \arrow[r, "n \mapsto 3n"] \arrow[r, bend right, "n \mapsto \ln(n)"'] & \mathbb{R}
\end{tikzcd}$$

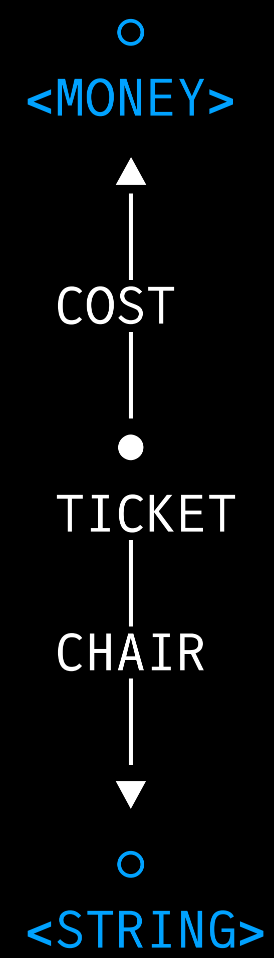
The category **Set**

1. Ob(**Set**) is all sets
2. The morphisms are all (proper) functions between sets
3. Each set has an identity that sends every element to itself
4. Composition of morphisms is composition of functions

$$\begin{tikzcd} \mathbb{B} \arrow[r, "b \mapsto \mathrm{int}(b)+1"] & \mathbb{N} \arrow[l, bend right, "n \mapsto !\mathrm{isodd}(n)"'] \arrow[r, "n \mapsto 3n"] \arrow[r, bend right, "n \mapsto \ln(n)"'] & \mathbb{R} \end{tikzcd}$$

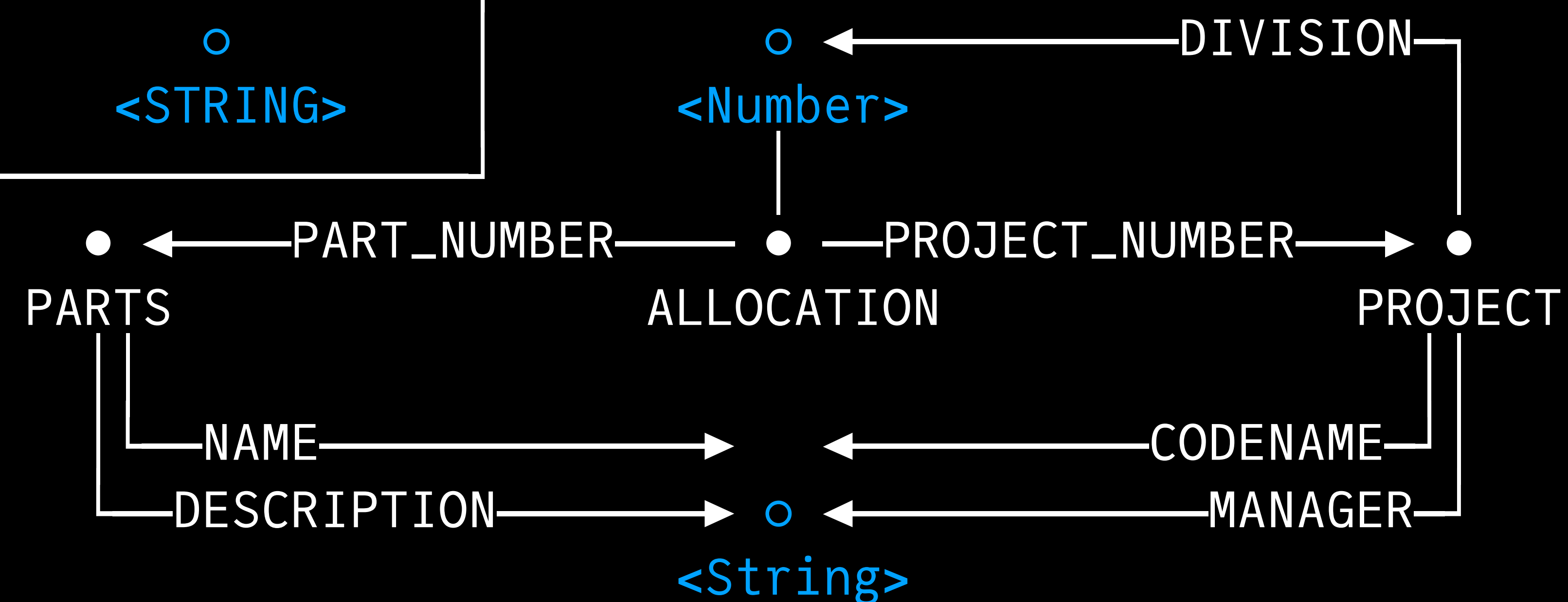
$\mathbb{B} = \{\mathrm{True}, \mathrm{False}\}$





DATABASES AND CATEGORIES ARE (ALMOST) THE SAME THING

(* Up to implementation details)



FUNCTOR

A mapping between categories: objects ↦ objects, morphisms ↦ morphisms.

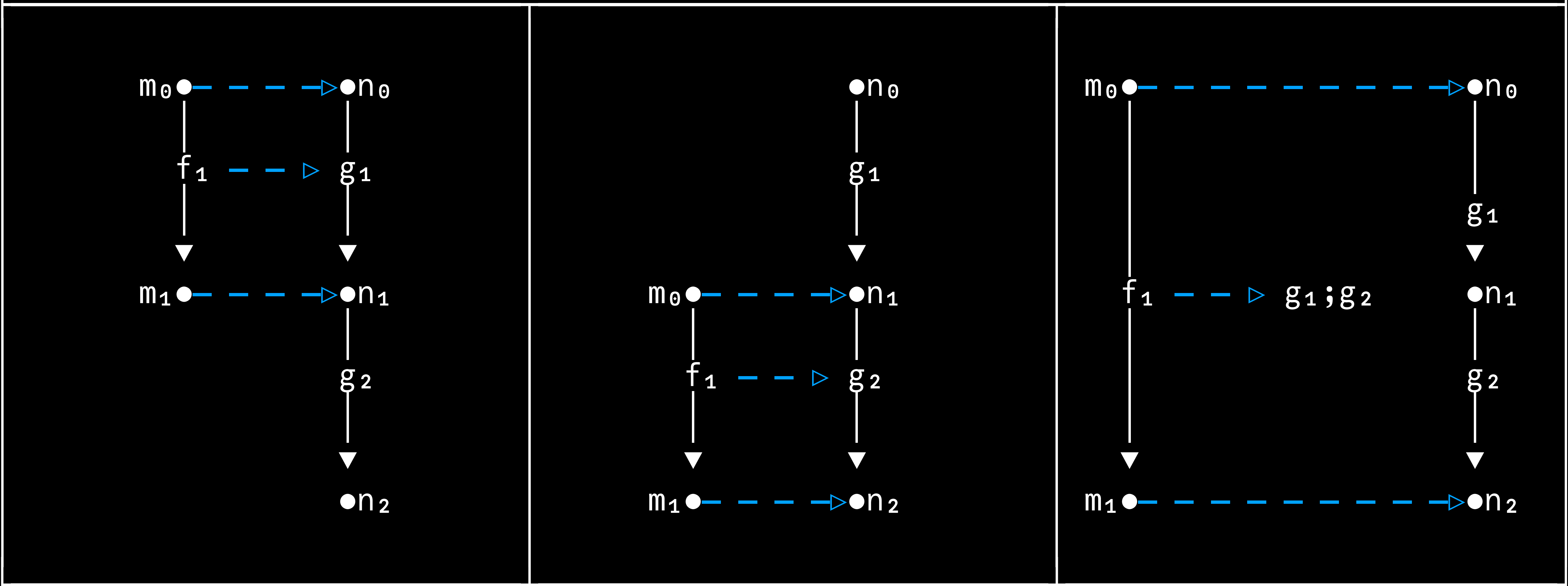
$F : \mathcal{C} \to \mathcal{D}$

$\quad \forall\, c \in \mathcal{C},\ F(c) \in \mathcal{D}$

$\quad \forall\, f : c_1 \to c_2 \in \mathrm{Hom}(\mathcal{C}),\ F(f) \in \mathrm{Hom}(\mathcal{D})$

Compositions go to compositions.

Identities go to identities.



FUNCTOR

A mapping between categories: objects ↦ objects, morphisms ↦ morphisms.

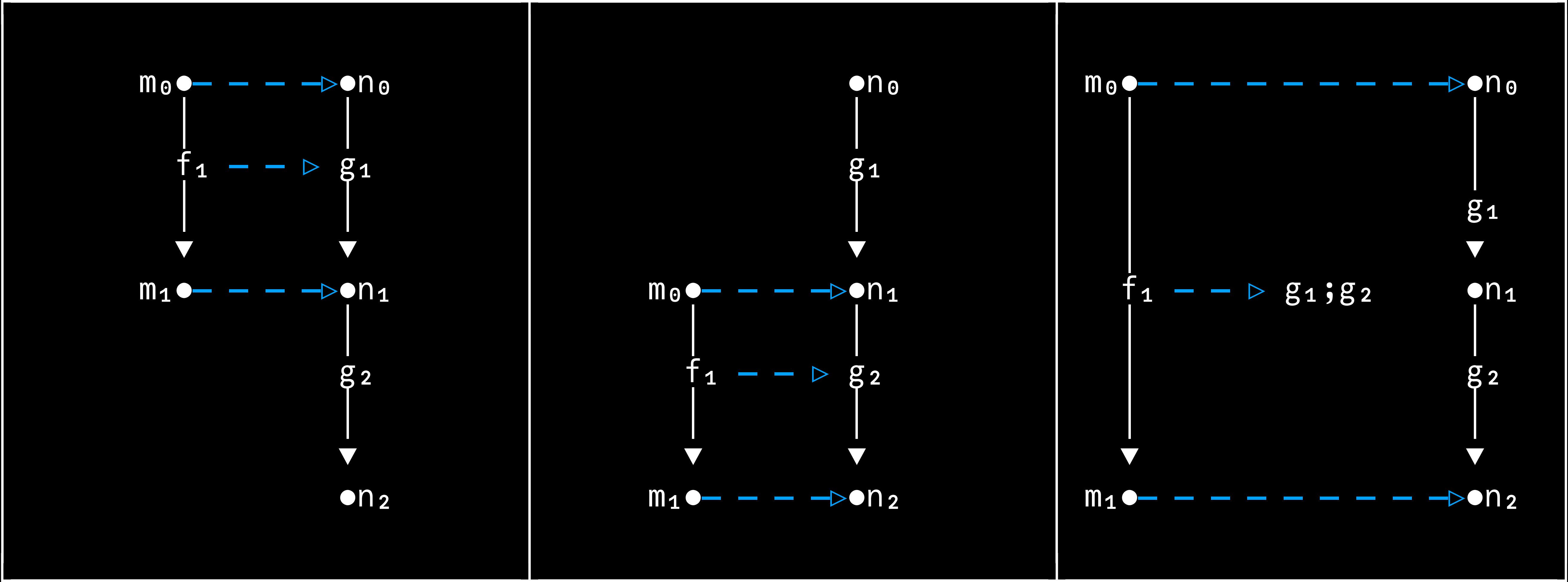
$F : \mathcal{C} \to \mathcal{D}$

$\quad \forall\ c \in \mathcal{C},\ F(c) \in \mathcal{D}$

$\quad \forall\ f : c_1 \to c_2 \in \mathrm{Hom}(\mathcal{C}),\ F(f) \in \mathrm{Hom}(\mathcal{D})$

Compositions go to compositions.

Identities go to identities.



FUNCTOR

A mapping between categories: objects ↦ objects, morphisms ↦ morphisms.

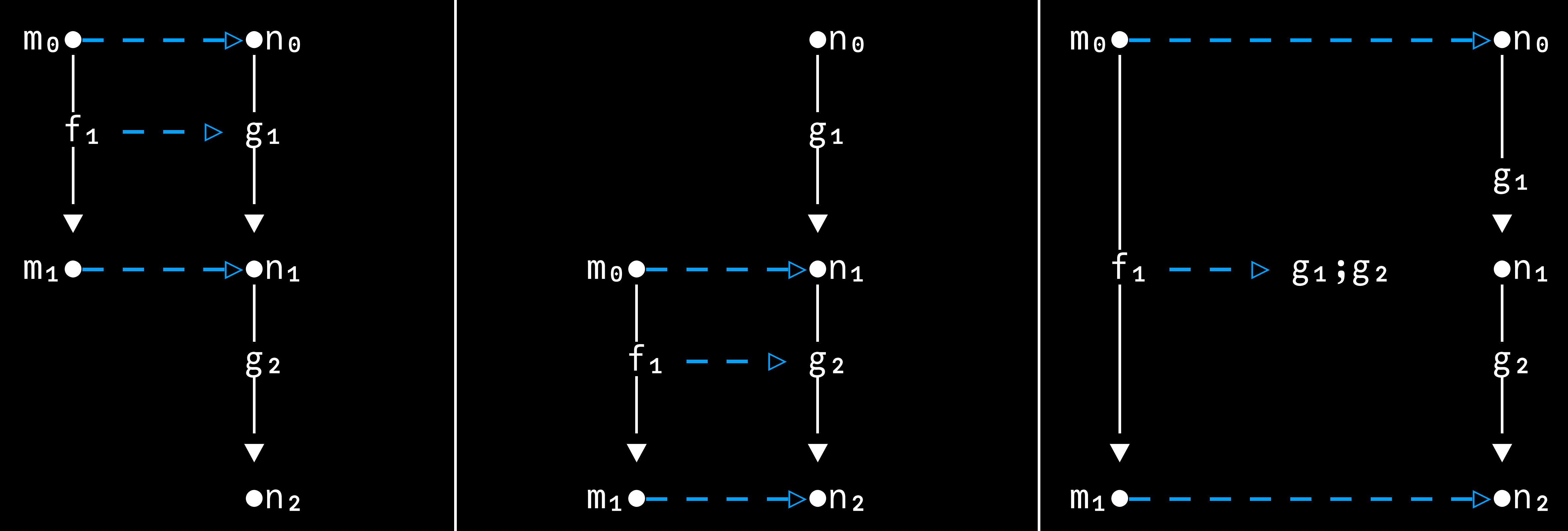
$F : \mathcal{C} \to \mathcal{D}$

$\quad \forall\ c \in \mathcal{C},\ F(c) \in \mathcal{D}$

$\quad \forall\ f : c_1 \to c_2 \in \mathrm{Hom}(\mathcal{C}),\ F(f) \in \mathrm{Hom}(\mathcal{D})$

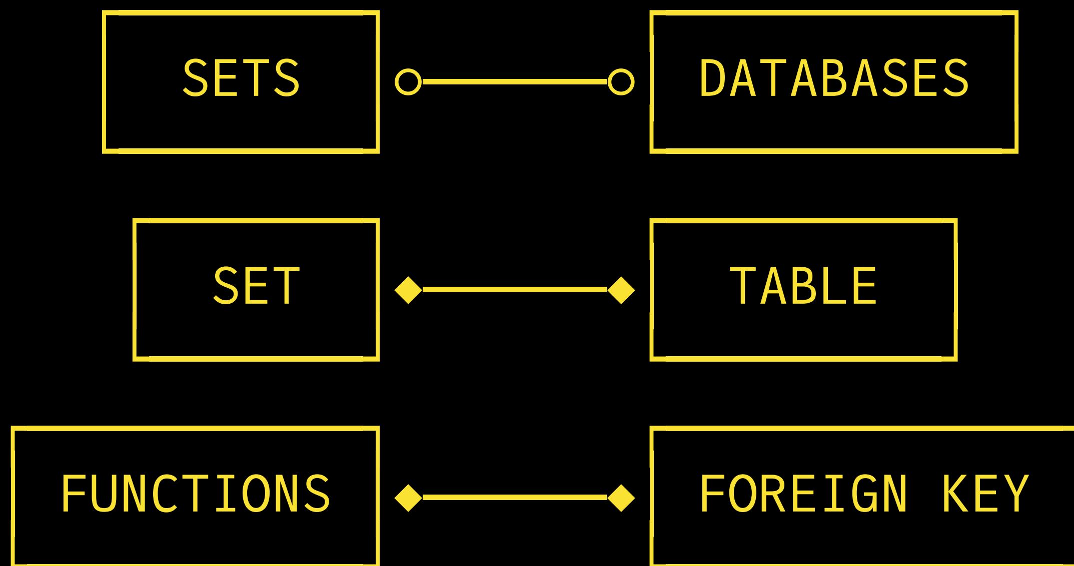
Compositions go to compositions.

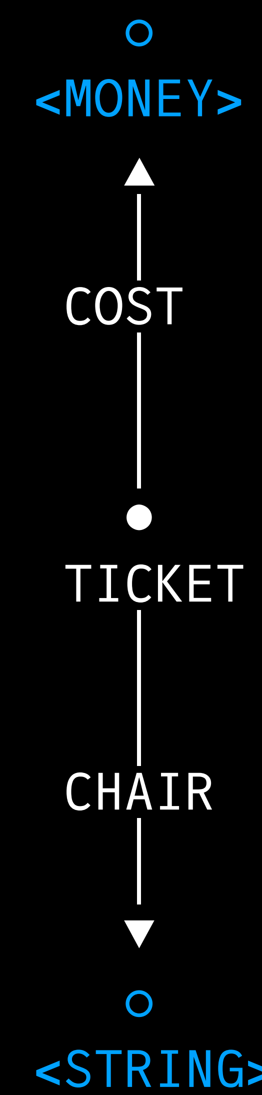
Identities go to identities.



Category Theory and Databases

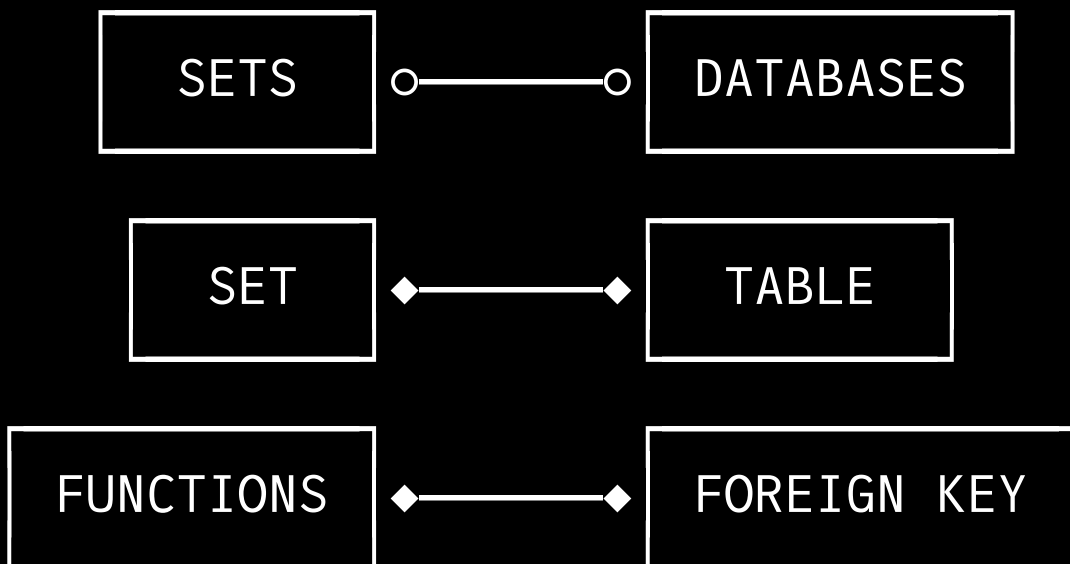
FONG, SPIVAK 2018

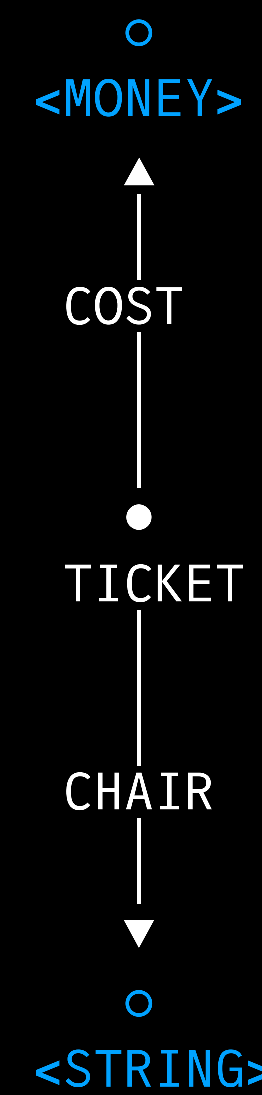




A schema presents a category $\mathcal{C}$.

The specific data in a database is a functor DATA : $\mathcal{C} \to$ Set.





A schema presents a category $\mathcal{C}$.

The specific data in a database is a functor DATA : $\mathcal{C} \to \mathbf{Set}$.

Natural transformation

[This is the most complicated thing we're going to discuss!]

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left, "F"{name=F}] \arrow[r, bend right, "G"'{name=G}] & \mathcal{D} \arrow[from=F, to=G, Rightarrow, "\alpha"] \end{tikzcd}$$

You've got two categories $\mathcal{C}$ and $\mathcal{D}$.

You've got two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$.
(That means F and G send dots of $\mathcal{C}$ to dots of $\mathcal{D}$, arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$, and do so in a way that preserves how the arrows connect.)

A natural transformation $\alpha : F \Rightarrow G$ is a collection of morphisms α_c, which map $F(c)$ to $G(C)$.

$$\begin{tikzcd} F(c) \arrow[r, "\alpha_c"] \arrow[d, "F(f)"] & G(c) \arrow[d, "G(f)"] \\ F(d) \arrow[r, "\alpha_d"] & G(d) \end{tikzcd}$$

Each specific α_c has the property of "naturality."
"Naturality" means that when we consider the functors F and G, it doesn't matter if you do the functor first then the transformation, or the transformation first then the functor; you get the same result either way.

Natural transformation

[This is the most complicated thing we're going to discuss!]

You've got two categories $\mathcal{C}$ and $\mathcal{D}$.

You've got two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$.
(That means F and G send dots of $\mathcal{C}$ to dots of $\mathcal{D}$, arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$, and do so in a way that preserves how the arrows connect.)

$$\begin{tikzcd} \mathcal{C} \arrow[rr, bend left, "F"] \arrow[rr, bend right, "G"'] \arrow[rr, phantom, "\Downarrow\alpha"] & & \mathcal{D} \end{tikzcd}$$

A natural transformation $\alpha : F \Rightarrow G$ is a collection of morphisms α_c, which map $F(c)$ to $G(C)$.

Each specific α_c has the property of "naturality."
"Naturality" means that when we consider the functors F and G, it doesn't matter if you do the functor first then the transformation, or the transformation first then the functor; you get the same result either way.

$$\begin{tikzcd} F(c) \arrow[r, "\alpha_c"] \arrow[d, "F(f)"'] & G(c) \arrow[d, "G(f)"] \\ F(d) \arrow[r, "\alpha_d"'] & G(d) \end{tikzcd}$$

Natural transformation

[This is the most complicated thing we're going to discuss!]

You've got two categories $\mathcal{C}$ and $\mathcal{D}$.

You've got two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$.
(That means F and G send dots of $\mathcal{C}$ to dots of $\mathcal{D}$, arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$, and do so in a way that preserves how the arrows connect.)

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left, "F"] \arrow[r, bend right, "G"'] \arrow[r, phantom, "\Downarrow\alpha"] & \mathcal{D} \end{tikzcd}$$

A natural transformation $\alpha : F \Rightarrow G$ is a collection of morphisms α_c, which map $F(c)$ to $G(C)$.

Each specific α_c has the property of "naturality."
"Naturality" means that when we consider the functors F and G, it doesn't matter if you do the functor first then the transformation, or the transformation first then the functor; you get the same result either way.

$$\begin{tikzcd} F(c) \arrow[r, "\alpha_c"] \arrow[d, "F(f)"] & G(c) \arrow[d, "G(f)"] \\ F(d) \arrow[r, "\alpha_d"] & G(d) \end{tikzcd}$$

Natural transformation

[This is the most complicated thing we're going to discuss!]

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left, "F"] \arrow[r, bend right, "G"'] \arrow[r, phantom, "\Downarrow\alpha"] & \mathcal{D} \end{tikzcd}$$

You've got two categories $\mathcal{C}$ and $\mathcal{D}$.

You've got two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$.
(That means F and G send dots of $\mathcal{C}$ to dots of $\mathcal{D}$, arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$, and do so in a way that preserves how the arrows connect.)

A natural transformation $\alpha : F \Rightarrow G$ is a collection of morphisms α_c, which map $F(c)$ to $G(C)$.

$$\begin{tikzcd} F(c) \arrow[r, "\alpha_c"] \arrow[d, "F(f)"'] & G(c) \arrow[d, "G(f)"] \\ F(d) \arrow[r, "\alpha_d"] & G(d) \end{tikzcd}$$

Each specific α_c has the property of "naturality."
"Naturality" means that when we consider the functors F and G, it doesn't matter if you do the functor first then the transformation, or the transformation first then the functor; you get the same result either way.

Natural transformation

[This is the most complicated thing we're going to discuss!]

You've got two categories $\mathcal{C}$ and $\mathcal{D}$.

You've got two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$.
(That means F and G send dots of $\mathcal{C}$ to dots of $\mathcal{D}$, arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$, and do so in a way that preserves how the arrows connect.)

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left=50, "F"{name=U}] \arrow[r, bend right=50, "G"'{name=D}] & \mathcal{D} \arrow[from=U, to=D, Rightarrow, "\alpha"] \end{tikzcd}$$

A natural transformation $\alpha : F \Rightarrow G$ is a collection of morphisms α_c, which map $F(c)$ to $G(C)$.

Each specific α_c has the property of "naturality."
"Naturality" means that when we consider the functors F and G, it doesn't matter if you do the functor first then the transformation, or the transformation first then the functor; you get the same result either way.

$$\begin{tikzcd} F(c) \arrow[r, "\alpha_c"] \arrow[d, "F(f)"'] & G(c) \arrow[d, "G(f)"] \\ F(d) \arrow[r, "\alpha_d"'] & G(d) \end{tikzcd}$$

Natural transformation

[This is the most complicated thing we're going to discuss!]

You've got two categories $\mathcal{C}$ and $\mathcal{D}$.

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left=50, "F"{name=F}] \arrow[r, bend right=50, "G"'{name=G}] & \mathcal{D} \arrow[Rightarrow, from=F, to=G, "\alpha"] \end{tikzcd}$$

You've got two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$.
(That means F and G send dots of $\mathcal{C}$ to dots of $\mathcal{D}$, arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$, and do so in a way that preserves how the arrows connect.)

A natural transformation $\alpha : F \Rightarrow G$ is a collection of morphisms α_c, which map $F(c)$ to $G(C)$.

$$\begin{tikzcd} F(c) \arrow[r, "\alpha_c"] \arrow[d, "F(f)"'] & G(c) \arrow[d, "G(f)"] \\ F(d) \arrow[r, "\alpha_d"'] & G(d) \end{tikzcd}$$

Each specific α_c has the property of "naturality."
"Naturality" means that when we consider the functors F and G, it doesn't matter if you do the functor first then the transformation, or the transformation first then the functor; you get the same result either way.

Natural transformation

[This is the most complicated thing we're going to discuss!]

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left=50, "F"{name=U}] \arrow[r, bend right=50, "G"'{name=L}] & \mathcal{D} \arrow[from=U, to=L, Rightarrow, "\alpha"] \end{tikzcd}$$

You've got two categories $\mathcal{C}$ and $\mathcal{D}$.

You've got two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$.
(That means F and G send dots of $\mathcal{C}$ to dots of $\mathcal{D}$, arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$, and do so in a way that preserves how the arrows connect.)

A natural transformation $\alpha : F \Rightarrow G$ is a collection of morphisms α_c, which map $F(c)$ to $G(C)$.

$$\begin{tikzcd} F(c) \arrow[r, "\alpha_c"] \arrow[d, "F(f)"] & G(c) \arrow[d, "G(f)"] \\ F(d) \arrow[r, "\alpha_d"] & G(d) \end{tikzcd}$$

Each specific α_c has the property of "naturality."
"Naturality" means that when we consider the functors F and G, it doesn't matter if you do the functor first then the transformation, or the transformation first then the functor; you get the same result either way.

Natural transformation

[This is the most complicated thing we're going to discuss!]

$$\begin{tikzcd} \mathcal{C} \arrow[r, bend left, "F"] \arrow[r, bend right, "G"'] \arrow[r, phantom, "\Downarrow \alpha"] & \mathcal{D} \end{tikzcd}$$

You've got two categories $\mathcal{C}$ and $\mathcal{D}$.

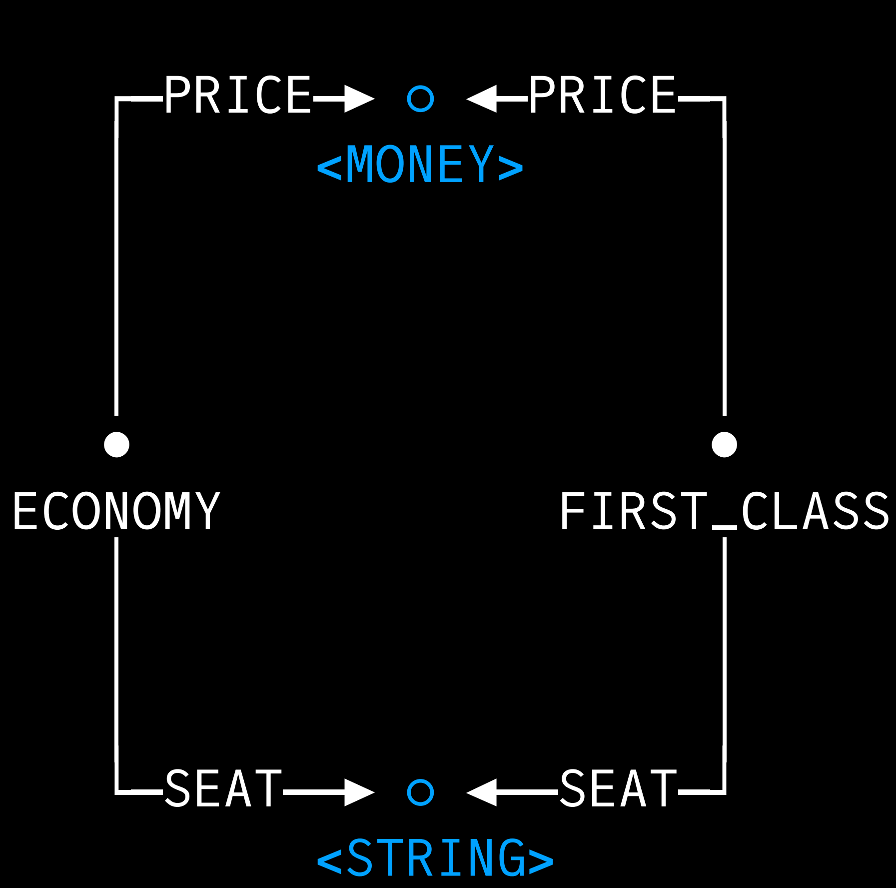
You've got two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$.
(That means F and G send dots of $\mathcal{C}$ to dots of $\mathcal{D}$, arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$, and do so in a way that preserves how the arrows connect.)

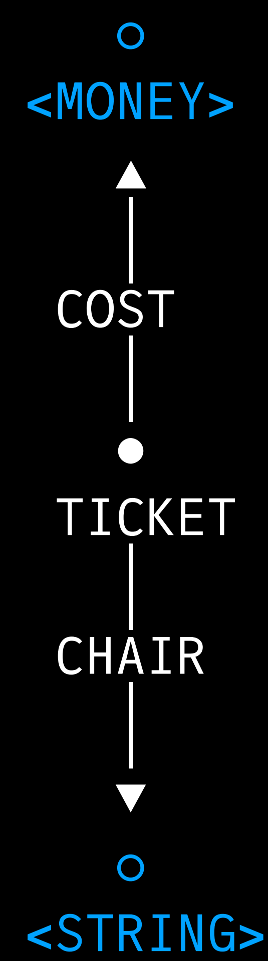
A natural transformation $\alpha : F \Rightarrow G$ is a collection of morphisms α_c, which map $F(c)$ to $G(C)$.

Each specific α_c has the property of "naturality."
"Naturality" means that when we consider the functors F and G, it doesn't matter if you do the functor first then the transformation, or the transformation first then the functor; you get the same result either way.

$$\begin{tikzcd} F(c) \arrow[r, "\alpha_c"] \arrow[d, "F(f)"] & G(c) \arrow[d, "G(f)"] \\ F(d) \arrow[r, "\alpha_d"] & G(d) \end{tikzcd}$$

If there is a functor from one schema to another, you can migrate data between them.





The tables and keys are transformed by the functor.

The data itself is already a functor DATA : $\mathcal{C} \to \mathbf{Set}$. As you transition from one schema to the other, you apply a natural transformation to the data.