# Towards a Principle of Least Surprise for bidirectional transformations

James Cheney, Jeremy Gibbons, James McKinna and **Perdita Stevens**

Universities of Edinburgh and Oxford, UK

July 2015

# Plan

- Why do we need such a Principle?
- Optimality: Least Change via metrics
- Why isn't that just the right answer?
- Reasonableness: Least Surprise
- What next?

# Model-driven development

**Definition**

A model is an abstract, usually graphical, representation of some aspect of a system.

# Model-driven development

**Definition**

A model is an abstract, usually graphical, representation of some aspect of a system.

**Definition**

MDD is software development in which models are important.

# Model-driven development

**Definition**

A model is an abstract, usually graphical, representation of some aspect of a system.

**Definition**

MDD is software development in which models are important.

**Motivation**

Manage information overload, by separating concerns and providing representations suitable for each set of decisions.

# Model-driven development

**Definition**

A model is an abstract, usually graphical, representation of some aspect of a system.

**Definition**

MDD is software development in which models are important.

**Motivation**

Manage information overload, by separating concerns and providing representations suitable for each set of decisions.

# Managing information overload means...

...avoiding distracting the expert with information that isn't relevant to them.

# Managing information overload means...

...avoiding distracting the expert with information that isn't relevant to them.

In an ideal world, their model would contain all and only the information that's relevant to them

and would change only because of things they need to know anyway.

# Managing information overload means...

...avoiding distracting the expert with information that isn't relevant to them.

In an ideal world, their model would contain all and only the information that's relevant to them

and would change only because of things they need to know anyway.

The world is not ideal, and that's why software development is hard.

# Managing information overload means...

...avoiding distracting the expert with information that isn't relevant to them.

In an ideal world, their model would contain all and only the information that's relevant to them

and would change only because of things they need to know anyway.

The world is not ideal, and that's why software development is hard.

Least Surprise is the whole point of MDD.

# Managing information overload means...

...avoiding distracting the expert with information that isn't relevant to them.

In an ideal world, their model would contain all and only the information that's relevant to them

and would change only because of things they need to know anyway.

The world is not ideal, and that's why software development is hard.

Least Surprise is the whole point of MDD.

So we'd better get it right.

# Informal idea

**Meertens' formulation**

The action taken by the maintainer of a constraint after a violation should change no more than is needed to restore the constraint.

Extreme case is easy to formalise (and, usually, agree on):

**Hippocraticness**

If nothing needs to be changed, change nothing.

# Naive approach

Let $X$ be how much we need to change to restore the constraint.

Let $Y$ be how much the bx actually changes.

Require $Y \leq X$.

Oh, so $Y = X$.

Now: how do we measure $X$ and $Y$?

# Metrics

Given $m$ and $m'$, need to be able to say how much changed from $m$ to $m'$.

A sensible way to do this is called a metric.

**Definition**

A metric on $M$ is $d : M \times M \to \mathbb{R}$
satisfying
$d(m, m') = 0$ iff $m = m'$
$d(m, m') = d(m', m)$
$d(m, m') + d(m', m'') \geq d(m, m'')$

E.g. minimal edit distance (typically).

# Metric least change

Assume given:

- a bx $(R, \overrightarrow{R}, \overleftarrow{R}) : M \leftrightarrow N$;
- metrics $d_M$, $d_N$ on $M$ and $N$.

Then

**Definition**

$R$ is metric-least if for all $m \in M$ and for all $n, n' \in N$

$$R(m, n') \Rightarrow d_N(n, n') \geq d_N(n, \overrightarrow{R}(m, n))$$

and dually.

# Implementing metric least change

Macedo and Cunha implemented this approach, providing a new semantics to the syntax of QVT-R.

Given consistency relation $R$, and models $m$, $n$, with the task to find a new $n'$ consistent with $m$:

1. set $d = 0$;
2. search exhaustively for consistent $n' \in N$ at distance $d$ from $n$;
3. if there are any, present them all to the user as options;
4. else, increment $d$ and goto 2.

# Problems with metric least change

1. Scalability. Clever solutions? Probably not (cf Buneman Khanna and Tan PODS'02).
2. Usability: no easy way to ensure unique closest model, or to allow bx programmer to specify the choice.
3. Doesn't compose (even where that makes sense).
4. Inflexibility (+/−?).

Can mitigate inflexibility by allowing bx programmer to choose metrics: but hmm.

# Example where inflexibility bites

$M =$ UML models

$N =$ test suite (in JUnit say)

$R(m, n)$ iff every $\langle\langle$persistent$\rangle\rangle$ class in $m$ has a test class of the same name in $n$, containing an "appropriate" set of tests for each public operation...

# Example where inflexibility bites

$M =$ UML models

$N =$ test suite (in JUnit say)

$R(m, n)$ iff every $\langle\langle$persistent$\rangle\rangle$ class in $m$ has a test class of the same name in $n$, containing an "appropriate" set of tests for each public operation...

You modify the test class for a $\langle\langle$persistent$\rangle\rangle$ class: `int` $\rightarrow$ `long` throughout.

# Example where inflexibility bites

$M =$ UML models

$N =$ test suite (in JUnit say)

$R(m, n)$ iff every $\langle\langle$persistent$\rangle\rangle$ class in $m$ has a test class of the same name in $n$, containing an "appropriate" set of tests for each public operation...

You modify the test class for a $\langle\langle$persistent$\rangle\rangle$ class: `int` $\rightarrow$ `long` throughout.

Propagate back to UML model. What happens?

- `int` $\rightarrow$ `long` throughout?

# Example where inflexibility bites

$M =$ UML models

$N =$ test suite (in JUnit say)

$R(m, n)$ iff every $\langle\langle$persistent$\rangle\rangle$ class in $m$ has a test class of the same name in $n$, containing an "appropriate" set of tests for each public operation...

You modify the test class for a $\langle\langle$persistent$\rangle\rangle$ class: `int` $\rightarrow$ `long` throughout.

Propagate back to UML model. What happens?

- `int` $\rightarrow$ `long` throughout?
- remove $\langle\langle$persistent$\rangle\rangle$ ?

# Mitigation attempt 1

Use a different metric: make removing the $\langle\langle$persistent$\rangle\rangle$ be seen as an enormously expensive change.

But now we have a metric defined specifically for this bx.

# Mitigation attempt 2

Why do we feel that removing $\langle\langle$persistent$\rangle\rangle$ is not as cheap as it looks?

Because it disrupts the connection between the two models: breaks links.

It's a small change to the model, but a large change to the witness structure explaining their consistency?

# Witness structures

**Definition**

A witness structure is a structure whose intention is to capture something about the relationship between the models being kept consistent.

- Relational bx: "they're consistent because I say so"
- TGGs: "they're consistent because this is how they are built up together using the TGG rules"
- MDD with trace links: "they're consistent because this part of this links to that part of that"
- QVT-R game: "they're consistent because here's how Verifier wins a consistency game on them"
- Or even: "they're consistent because Coq gave me this proof that they are".

# So metric least change may yet be useful

in some settings, or when its deficiencies can be mitigated as mentioned.

But it does not deserve to be seen as the sole Right Answer.

Back to the drawing board.

# Principle of Least Surprise

or Least Astonishment, for user interface design, adapted to language design, etc. E.g.

**Saltzer and Kaashoek, via Wikipedia**

People are part of the system. The design should match the user's experience, expectations, and mental models.

# Principle of Least Surprise

or Least Astonishment, for user interface design, adapted to language design, etc. E.g.

**Saltzer and Kaashoek, via Wikipedia**

People are part of the system. The design should match the user's experience, expectations, and mental models.

The essential point is not that user should always know what will be on the next screen.

Rather, that they should seldom be surprised, i.e., taken aback.

Despite "least", we really seek reasonable, rather than optimal, behaviour.


BOSS
Keep me informed, OK? No nasty surprises!
MINION
Yessir!


Hmm. I have to change my model.
How much change can Boss take? ε?
Better inform him before I cause his model >ε change.
So like, before my model changes by δ?
OINIW

# Continuity

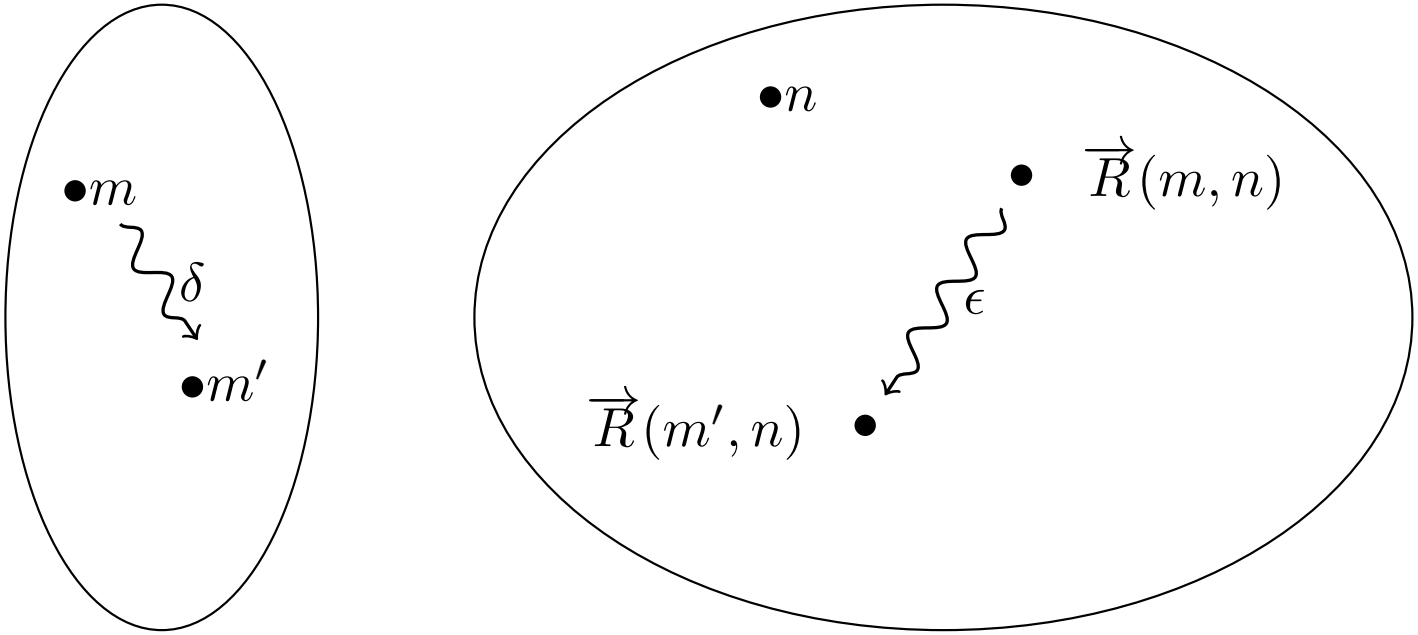
**Definition**

$f : S \to T$ is continuous at $s$ iff

$$\forall \epsilon > 0 \,.\, \exists \delta > 0 \,.\, \forall s' \,.\, d_S(s, s') < \delta \Rightarrow d_T(f(s), f(s')) < \epsilon$$

We say just "$f$ is continuous" if it is continuous at all $s$.

# Continuity of $\overrightarrow{R}$ at $(m, n)$




## Definition

$\overrightarrow{R}$ is continuous at $(m, n)$ iff

$$\forall \epsilon > 0 \,.\, \exists \delta > 0 \,.\, \forall m' \,.\, d_M(m, m') < \delta \Rightarrow d_N(\overrightarrow{R}(m, n), \overrightarrow{R}(m', n)) < \epsilon$$

i.e. iff $\overrightarrow{R}(\_, n) : M \to N$ is continuous at $m$.

# Where do we want continuity?

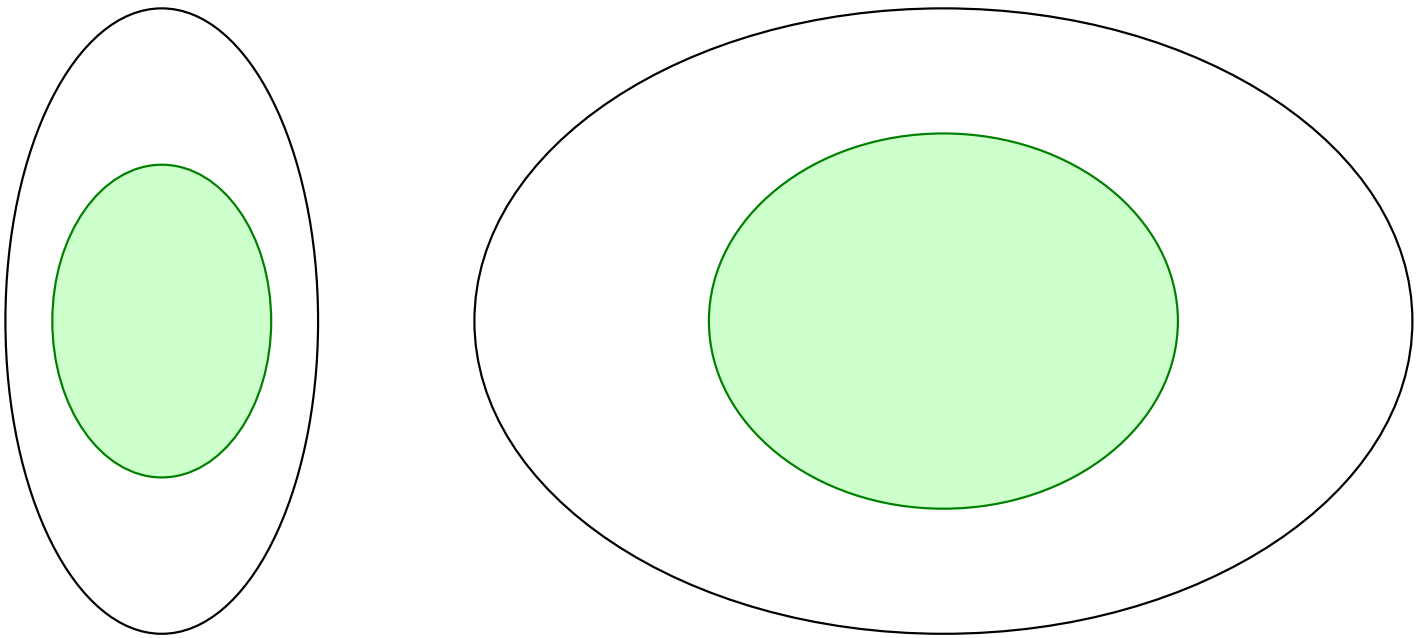
Well, at the points where we want guarantees... e.g.

- Everywhere: strong continuity
- Only at consistent $(m, n)$: weak continuity
- Something else? Where we can get it? On "good" subspace pair?

In the history ignorant (vwb, $\textsc{PutPut}$) case, strong $=$ weak.

$+$ Strongly (rsp. weakly) continuous bx compose, where composition makes sense.

# On good subspace pairs?



# The problem with continuity

– Every function is continuous at an isolated point :-(

Variants with better overall behaviour? See paper. Hölder continuity quite promising.

Prediction: way too strong to expect it everywhere in realistic bx, but identifying subspace pairs on which a bx is Hölder continuous may be worthwhile.

# Other approaches/Future work

category theory

topology

dependent type theory (HoTT??)

subspaces

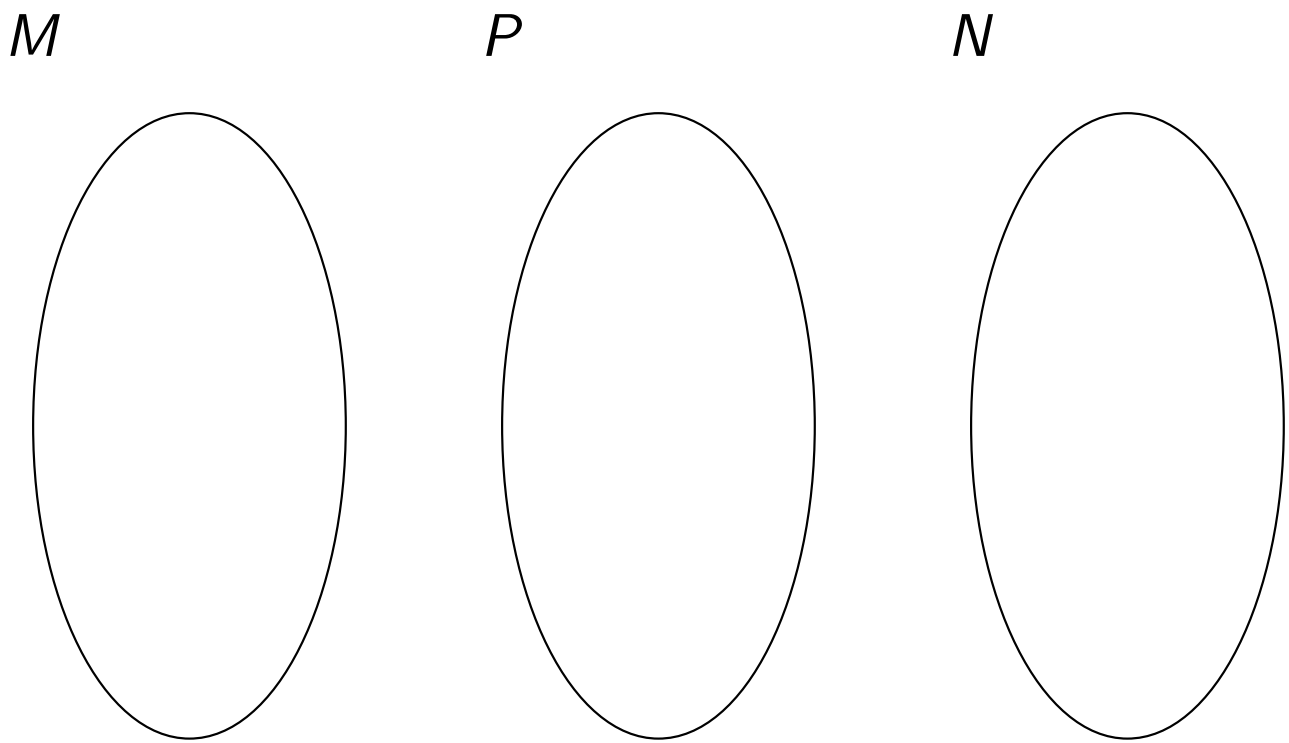
partiality

heuristics?

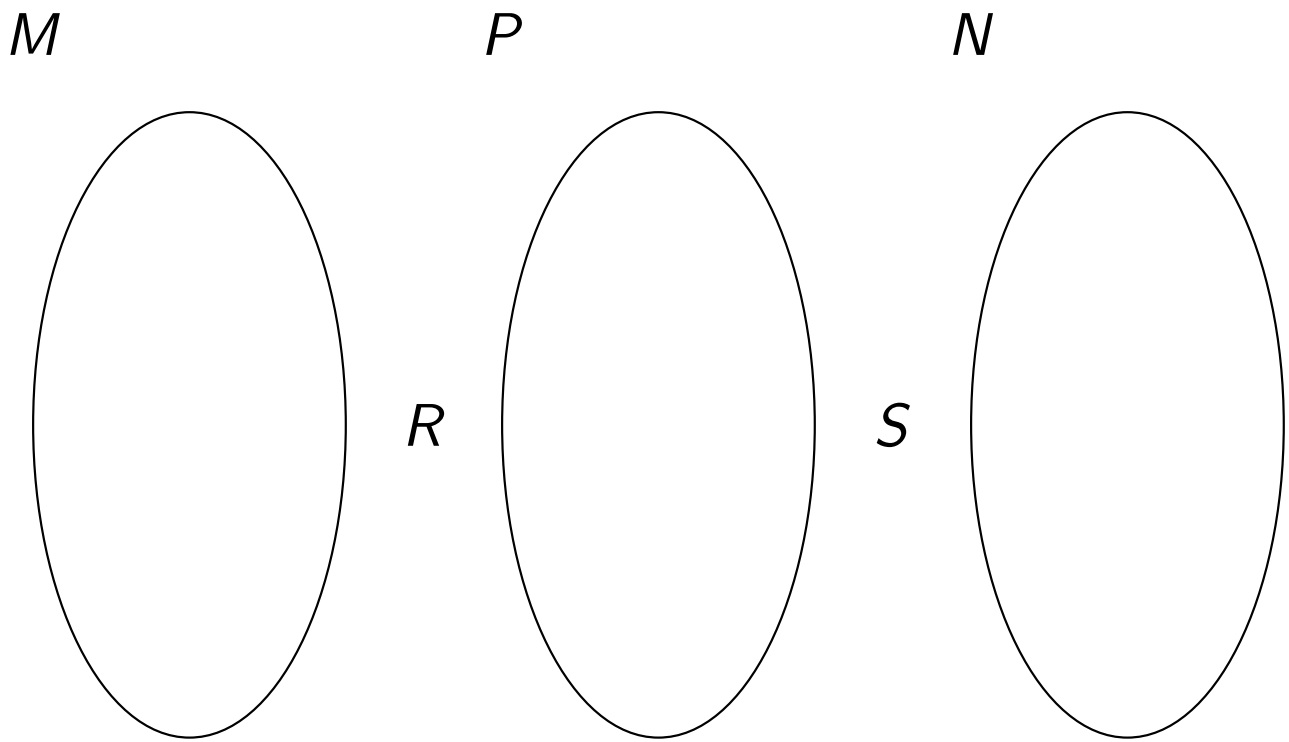
Lagrangian/Hamiltonian mechanics?

simulated annealing?

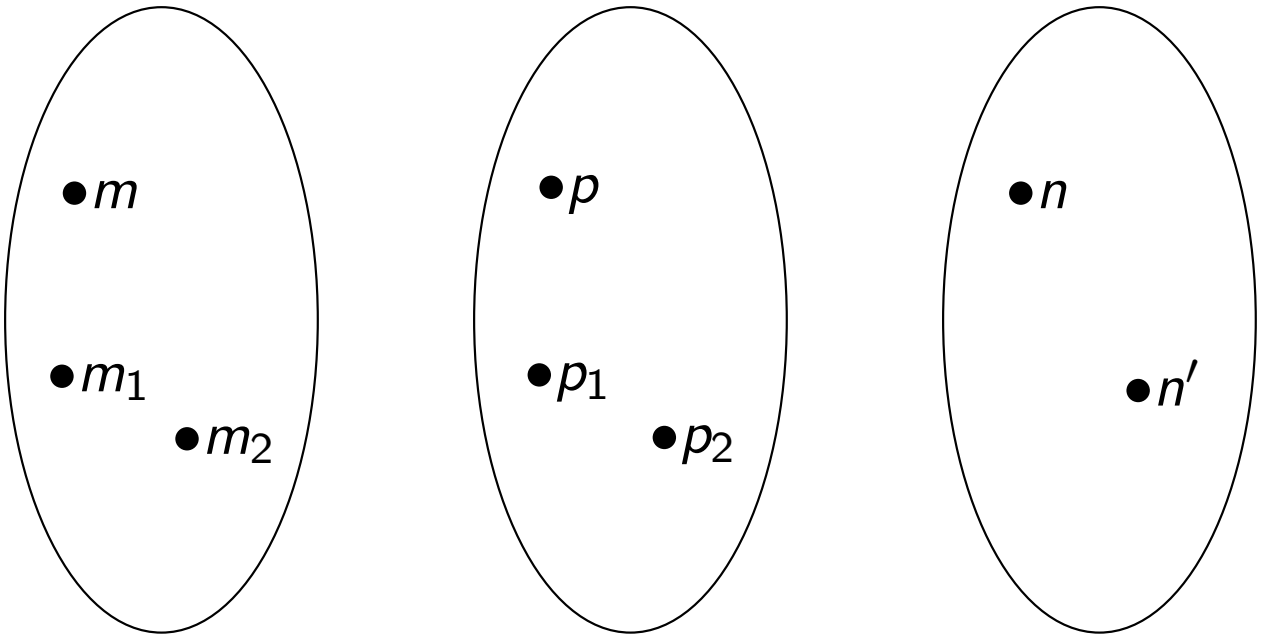
# Why the metric approach doesn't compose



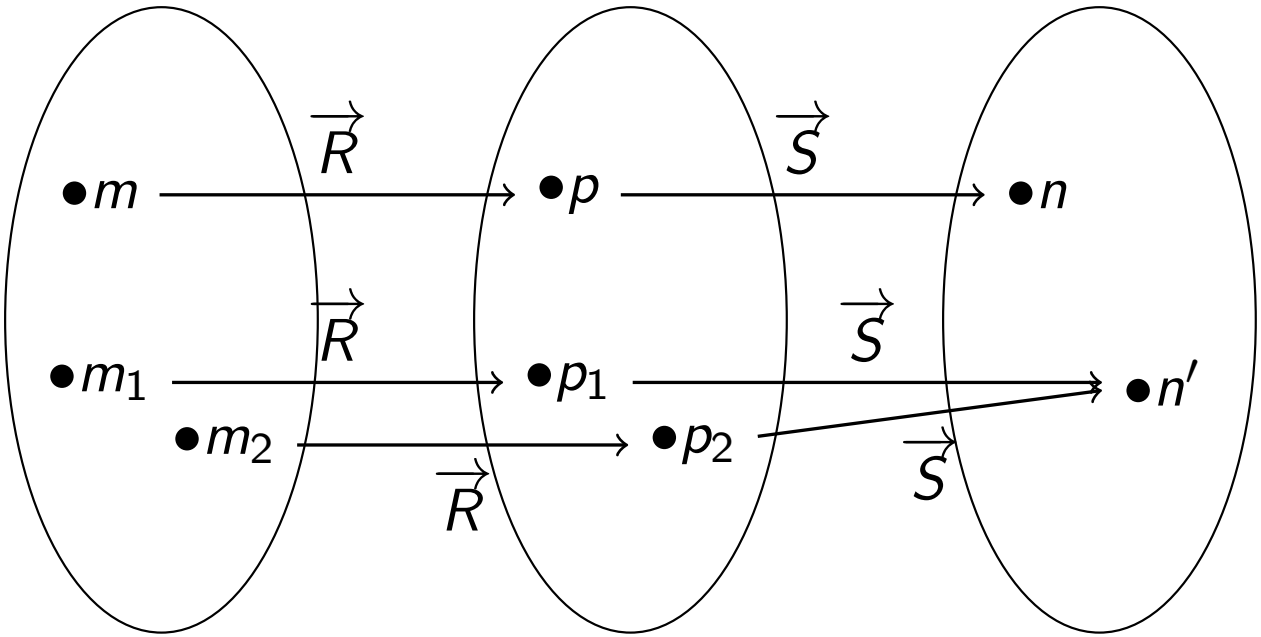
# Why the metric approach doesn't compose



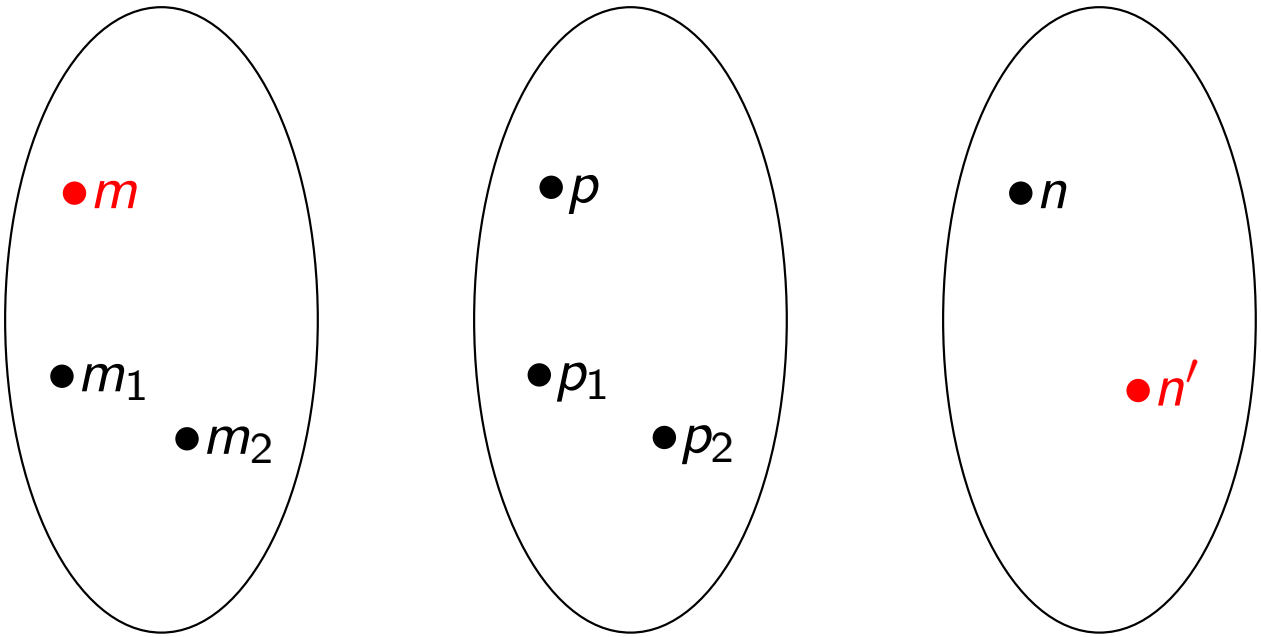
# Why the metric approach doesn't compose



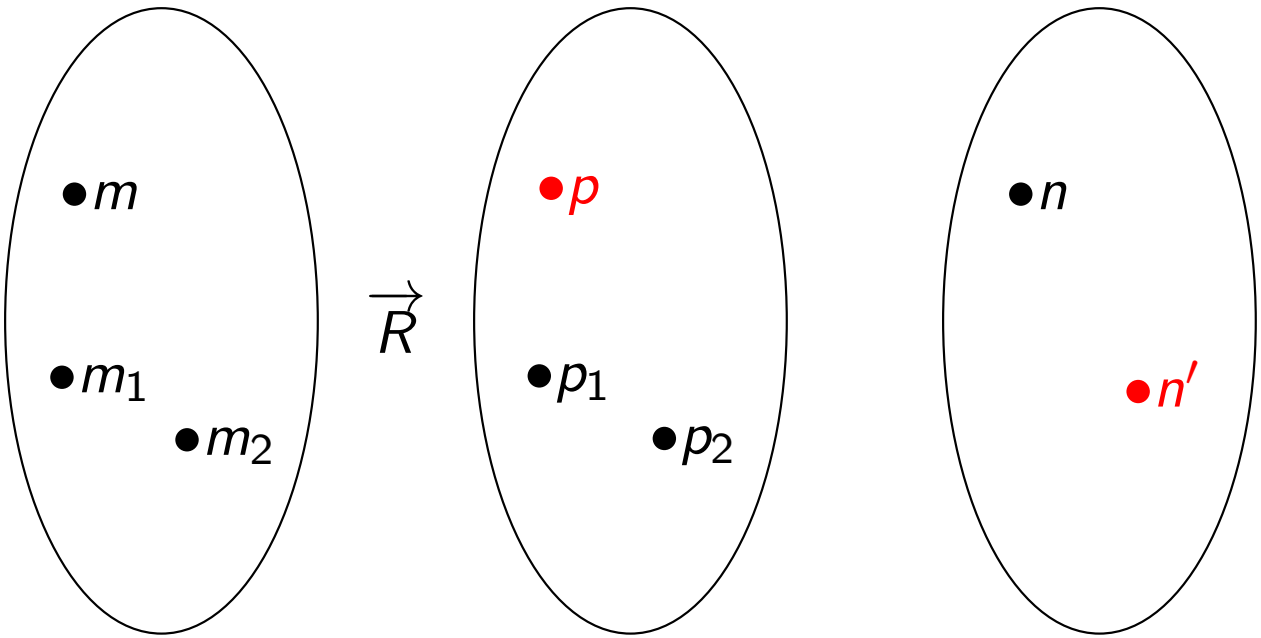
# Why the metric approach doesn't compose



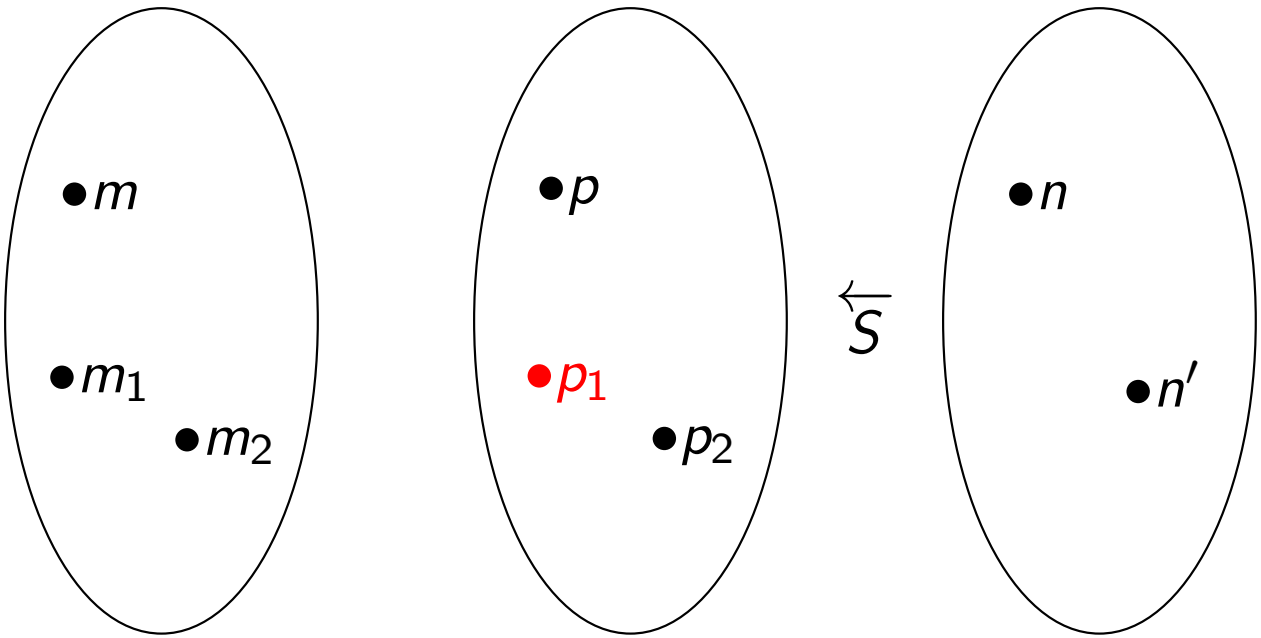
# Why the metric approach doesn't compose



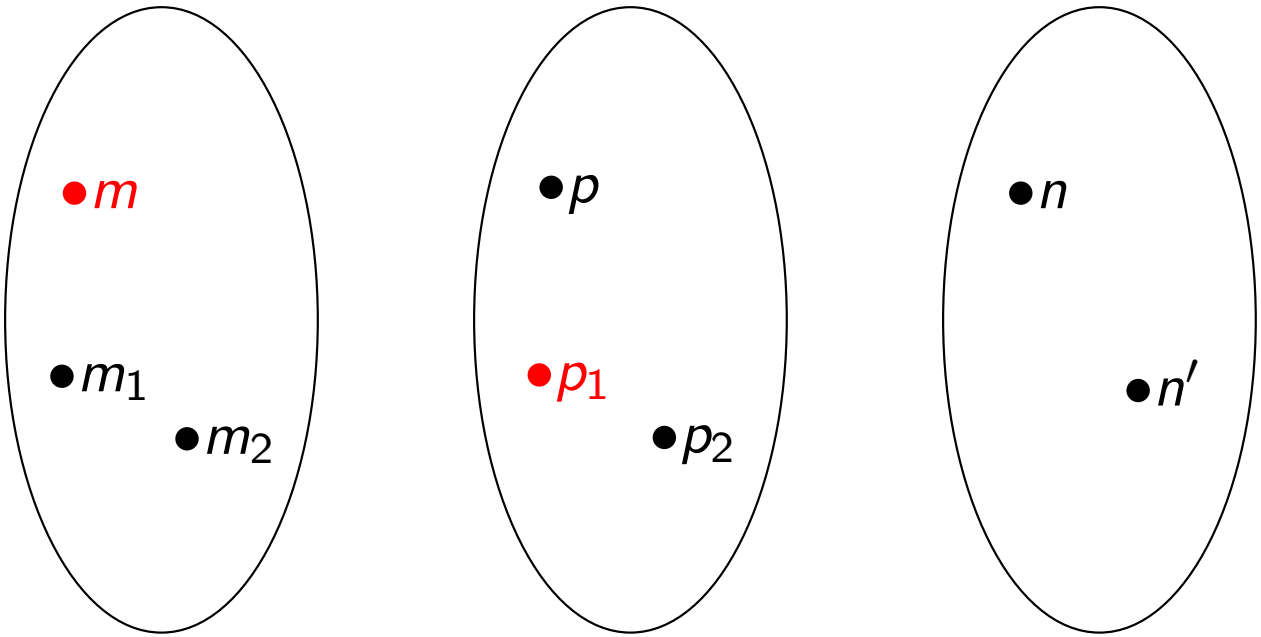
# Why the metric approach doesn't compose



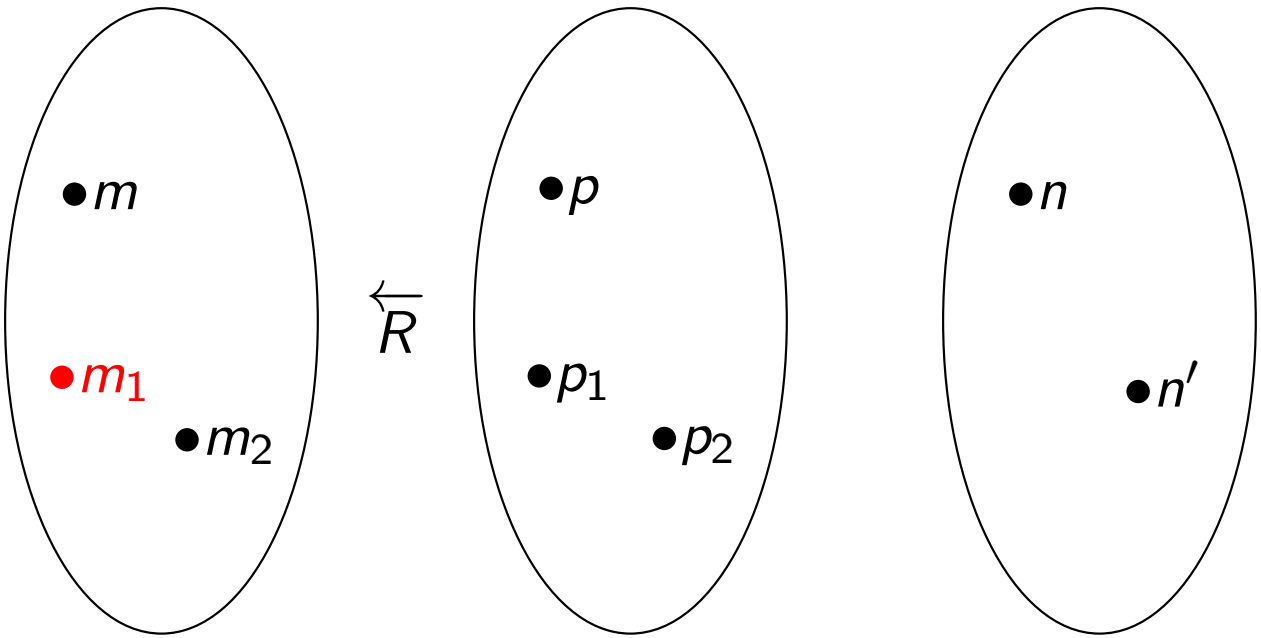
# Why the metric approach doesn't compose



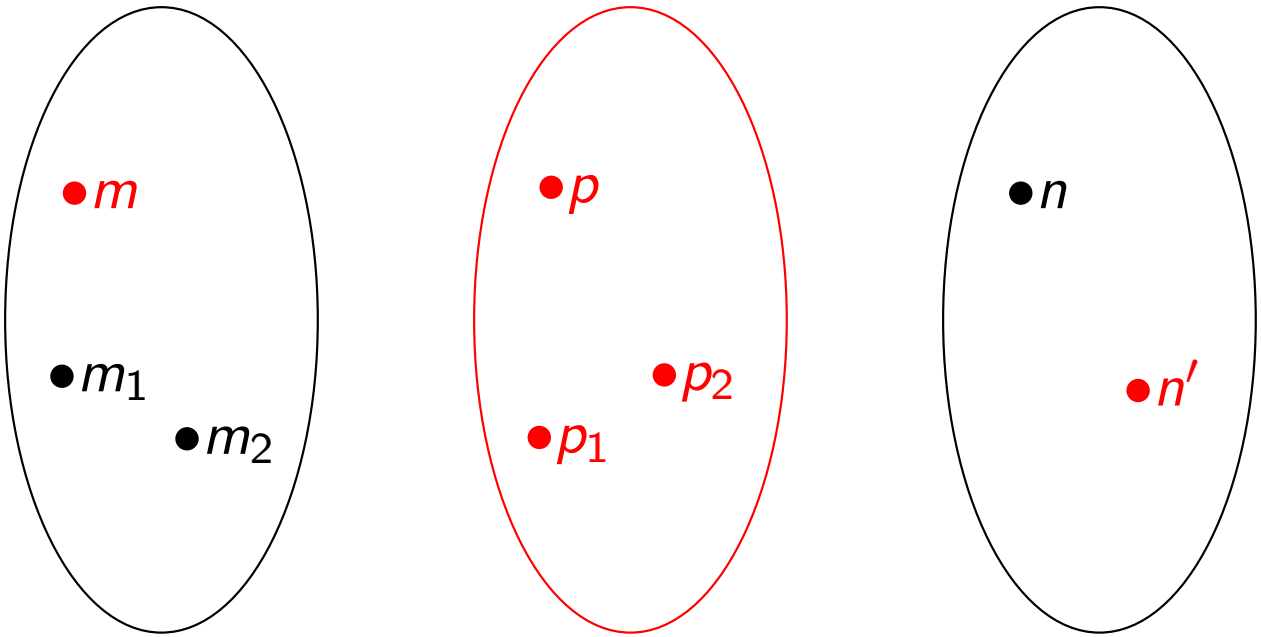
# Why the metric approach doesn't compose



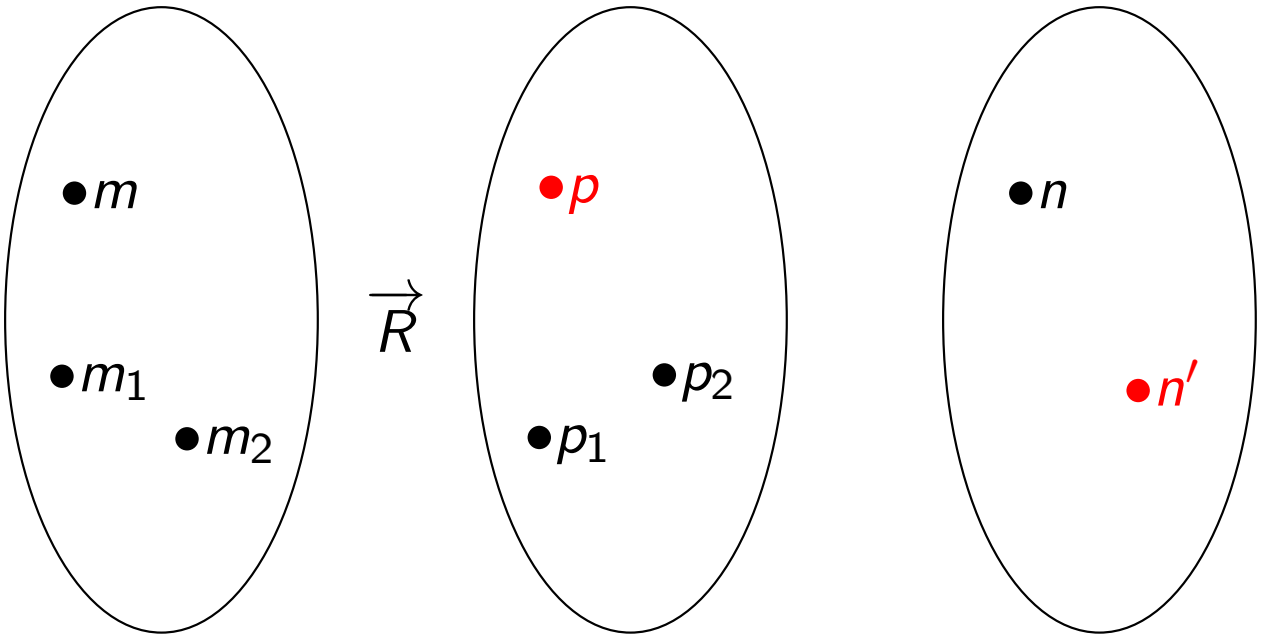
# Why the metric approach doesn't compose



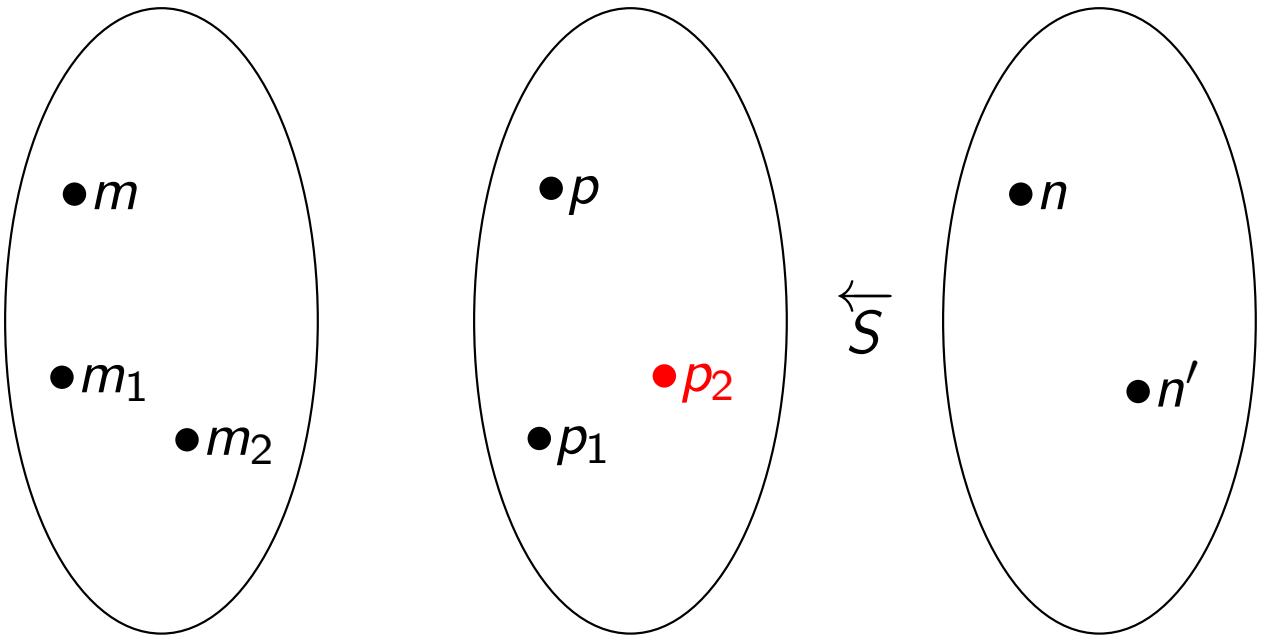
# Why the metric approach doesn't compose



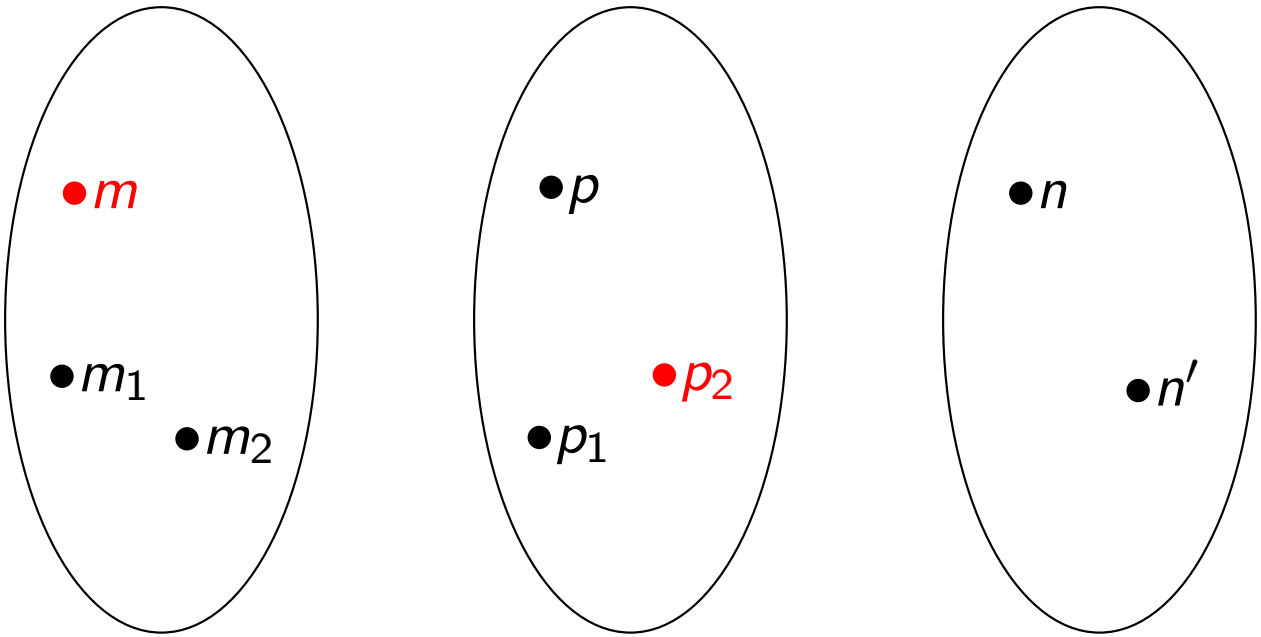
# Why the metric approach doesn't compose



# Why the metric approach doesn't compose



# Why the metric approach doesn't compose



# Why the metric approach doesn't compose