Penrose: from mathematical notation to beautiful diagrams

Katherine Ye*, Nimo Ni*, Max Krieger*, Dor Ma'ayan†, Jenna Wise*, Jonathan Aldrich*, Joshua Sunshine*, and Keenan Crane*

*Carnegie Mellon University
†Technion

SIGGRAPH 2020

Our goal: create high-quality diagrams directly from abstract mathematical statements.

set theory

venn-3d

run | sign in | step | inspector

venn 3d

sub | sty | dsl

```
1
2
3 AutoLabel All
```

editing | fill

Click run to render your diagram.

resample | autostep (off) | download

set theory

venn-3d

run | sign in | step | inspector

venn 3d

sub | sty | dsl

```
1
2
3 AutoLabel All
```

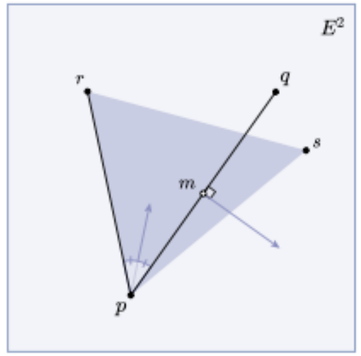
editing | fill

Click run to render your diagram.

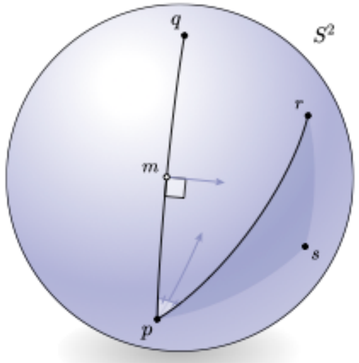
resample | autostep (off) | download

Can draw diagrams from any user-defined domain:

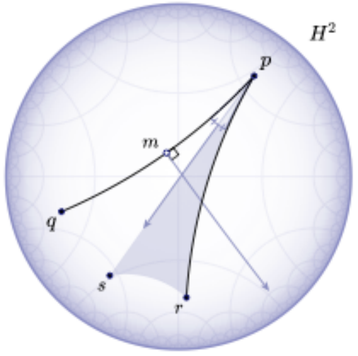
Can draw diagrams from any user-defined domain:



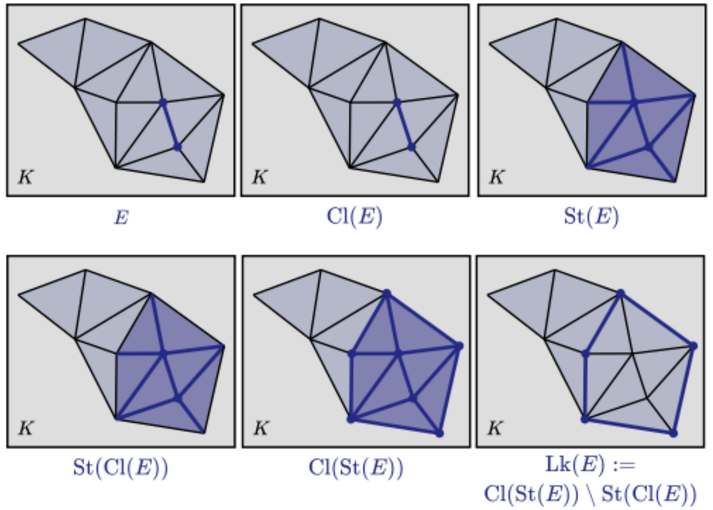
STYLE — Euclidean

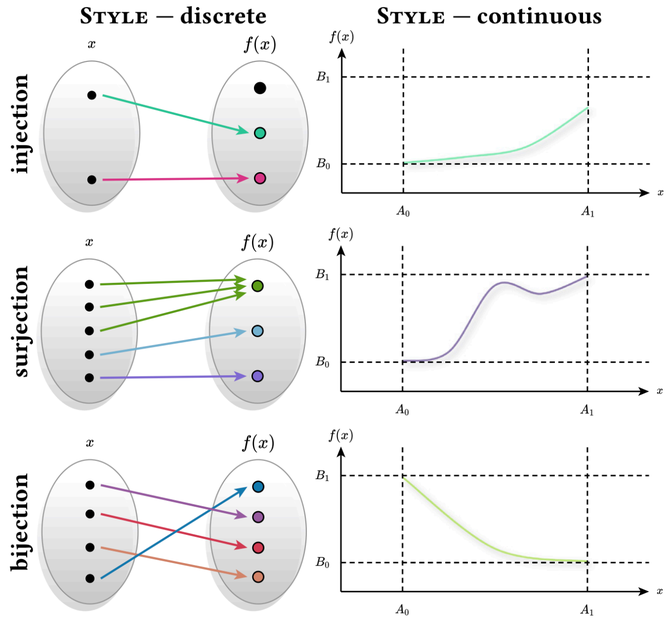


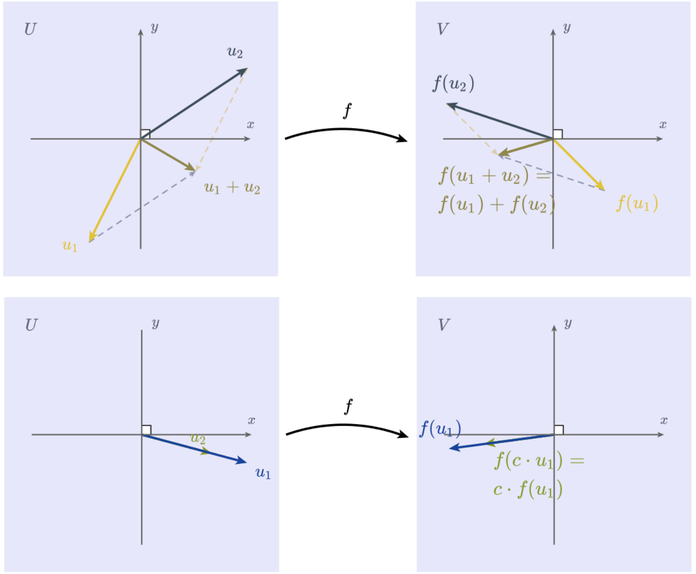
STYLE — spherical



STYLE — hyperbolic







Abstract ideas can be (very) hard to visualize.

3.1 THE NOTION OF A TOPOLOGY

The fundamental properties of open subsets of a metric space are outlined in Proposition 2 of Chapter 2. Mathematicians have found from experience that families of subsets having these same properties arise in contexts other than those of metric spaces; hence it is reasonable to study these properties in their own right, abstracted from the limitations that metric spaces impose. In particular, the properties of open sets in metric spaces inspire the following definition.

Definition 1. Let X be any set. A collection τ of subsets of X is said to be a *topology* on X if the following axioms are satisfied:

i) X and ϕ are members of τ.

ii) The intersection of any two members of τ is a member of τ.

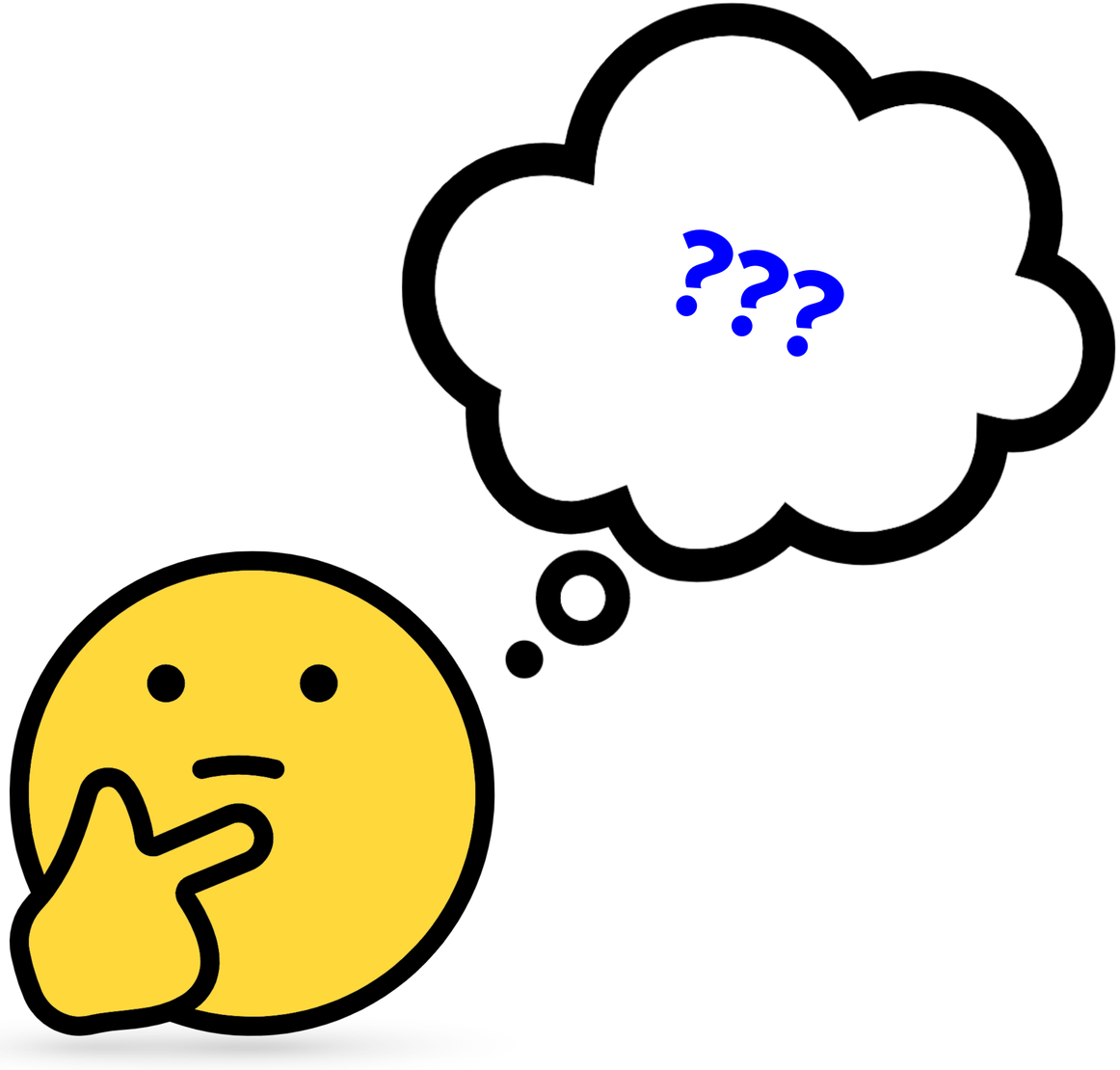
iii) The union of any family of members of τ is again in τ.

The members of τ are then said to be *τ-open* subsets of X, or merely *open* subsets of X if no confusion may result.

Example 1. If X, D is a metric space, then the D-open subsets of X form a topology on X. This topology is called the *metric topology* induced on X by D. It was, of course, this topology that we studied in Chapter 2.

Example 2. Let X be any set. Then the family of all subsets of X forms a topology on X. This topology consisting of all of the subsets of X is called the *discrete topology* on X. The discrete topology contains the maximum possible number of open sets since, relative to the discrete topology, every subset of X is open.

Example 3. If X is any set, then the collection $\{X, \phi\}$ of subsets of X also forms a topology on X. This topology is called the *trivial* (by some, the



Abstract ideas can be (very) hard to visualize.

3.1 THE NOTION OF A TOPOLOGY

The fundamental properties of open subsets of a metric space are outlined in Proposition 2 of Chapter 2. Mathematicians have found from experience that families of subsets having these same properties arise in contexts other than those of metric spaces; hence it is reasonable to study these properties in their own right, abstracted from the limitations that metric spaces impose. In particular, the properties of open sets in metric spaces inspire the following definition.

Definition 1. Let X be any set. A collection τ of subsets of X is said to be a *topology* on X if the following axioms are satisfied:

i) X and ϕ are members of τ.

ii) The intersection of any two members of τ is a member of τ.

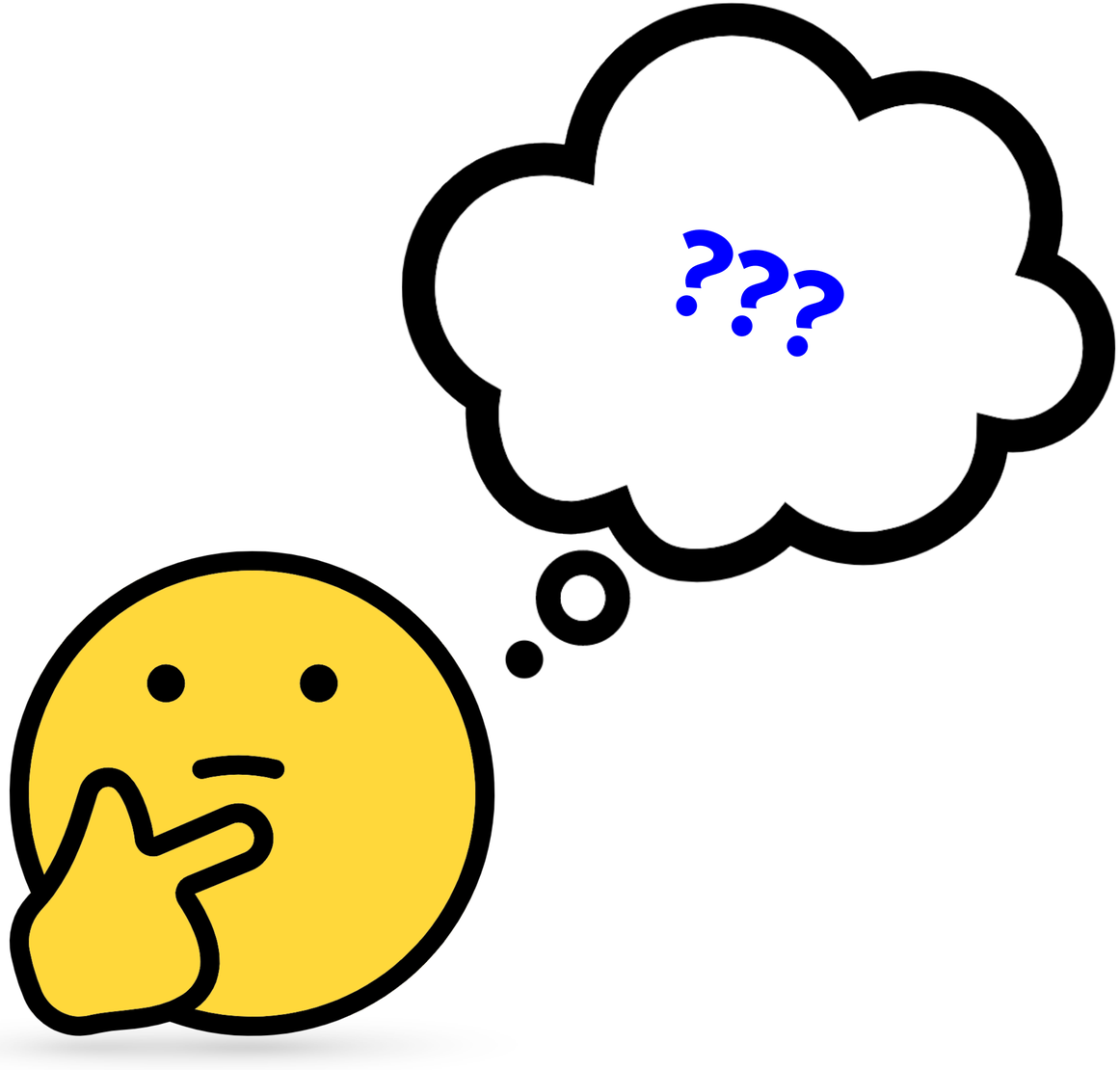
iii) The union of any family of members of τ is again in τ.

The members of τ are then said to be *τ-open* subsets of X, or merely *open* subsets of X if no confusion may result.

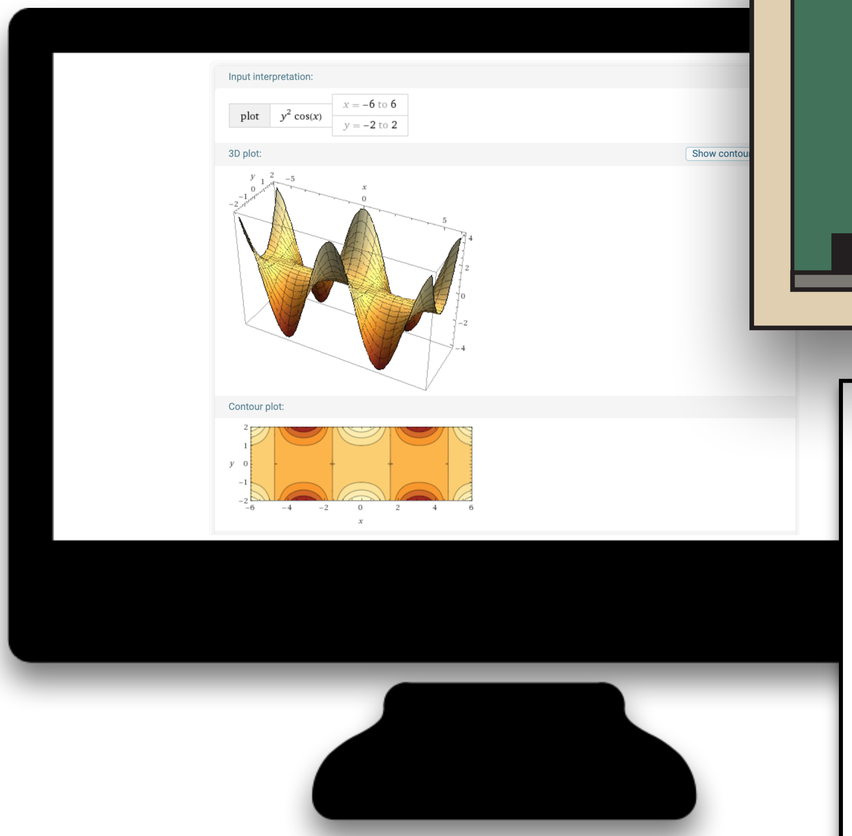
Example 1. If X, D is a metric space, then the D-open subsets of X form a topology on X. This topology is called the *metric topology* induced on X by D. It was, of course, this topology that we studied in Chapter 2.

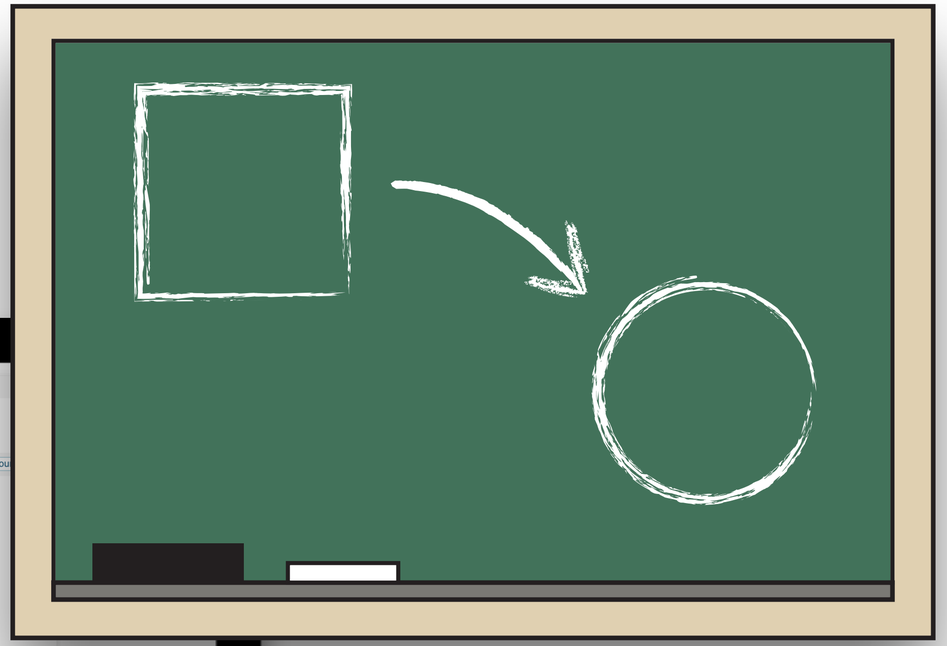
Example 2. Let X be any set. Then the family of all subsets of X forms a topology on X. This topology consisting of all of the subsets of X is called the *discrete topology* on X. The discrete topology contains the maximum possible number of open sets since, relative to the discrete topology, every subset of X is open.

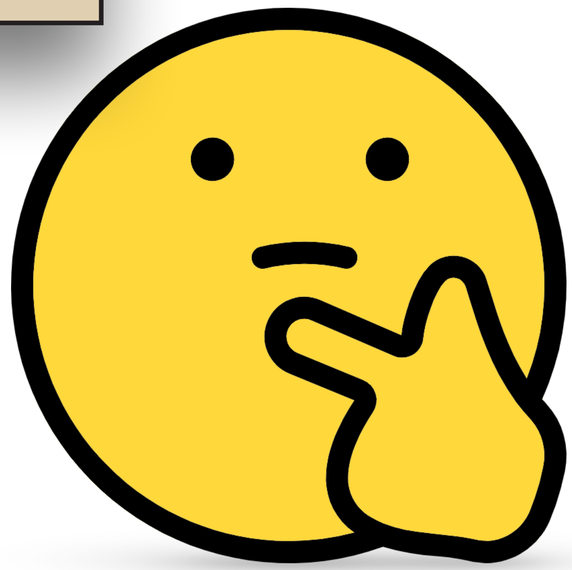
Example 3. If X is any set, then the collection $\{X, \phi\}$ of subsets of X also forms a topology on X. This topology is called the *trivial* (by some, the



Plotting/drawing tools don't solve the problem.







3.1 THE NOTION OF A TOPOLOGY

The fundamental properties of open subsets of a metric space are outlined in Proposition 2 of Chapter 2. Mathematicians have found from experience that families of subsets having these same properties arise in contexts other than those of metric spaces; hence it is reasonable to study these properties in their own right, abstracted from the limitations that metric spaces impose. In particular, the properties of open sets in metric spaces inspire the following definition.

Definition 1. Let X be any set. A collection τ of subsets of X is said to be a *topology* on X if the following axioms are satisfied:

i) X and ϕ are members of τ.

ii) The intersection of any two members of τ is a member of τ.

iii) The union of any family of members of τ is again in τ.

The members of τ are then said to be *τ-open* subsets of X, or merely *open* subsets of X if no confusion may result.

Example 1. If X, D is a metric space, then the D-open subsets of X form a topology on X. This topology is called the *metric topology* induced on X by D. It was, of course, this topology that we studied in Chapter 2.

Example 2. Let X be any set. Then the family of all subsets of X forms a topology on X. This topology consisting of all of the subsets of X is called the *discrete topology* on X. The discrete topology contains the maxi-

Need tools that consider the larger process:

Need tools that consider the larger process:

specify the mathematical idea

Need tools that consider the larger process:

specify the mathematical idea

translate it into a visual interpretation

Need tools that consider the larger process:

specify the mathematical idea

translate it into a visual interpretation

make a high-quality drawing

Need tools that consider the larger process:

specify the mathematical idea

↓

translate it into a visual interpretation

↓

make a high-quality drawing

Penrose

Organizing principles

Organizing principles

Organizing principles

- (Specification) A diagram is a **mapping** from abstract mathematical statements to a concrete visual representation.

Organizing principles

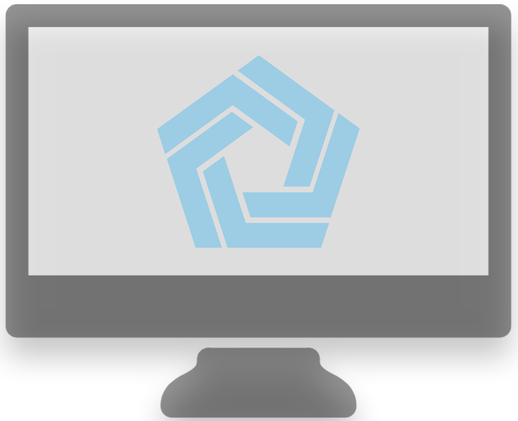
- (Specification) A diagram is a **mapping** from abstract mathematical statements to a concrete visual representation.
- (Synthesis) Given a specification, specific diagrams are realized via numerical **optimization**.

Organizing principles

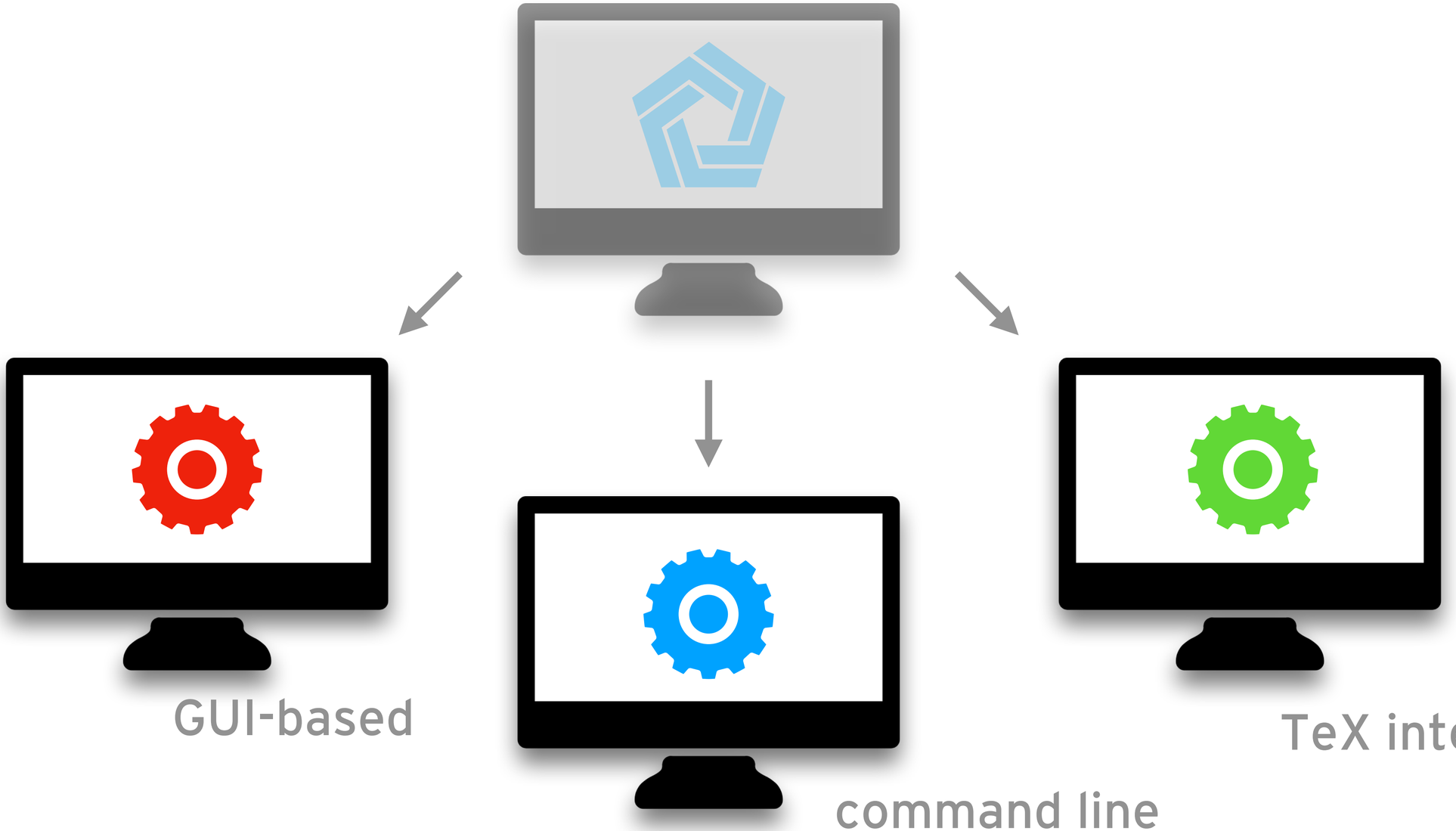
- (Specification) A diagram is a **mapping** from abstract mathematical statements to a concrete visual representation.
- (Synthesis) Given a specification, specific diagrams are realized via numerical **optimization**.

Specification is encoded via domain-specific languages

Provide a platform for building diagramming tools.



Provide a platform for building diagramming tools.



Benefits

Benefits

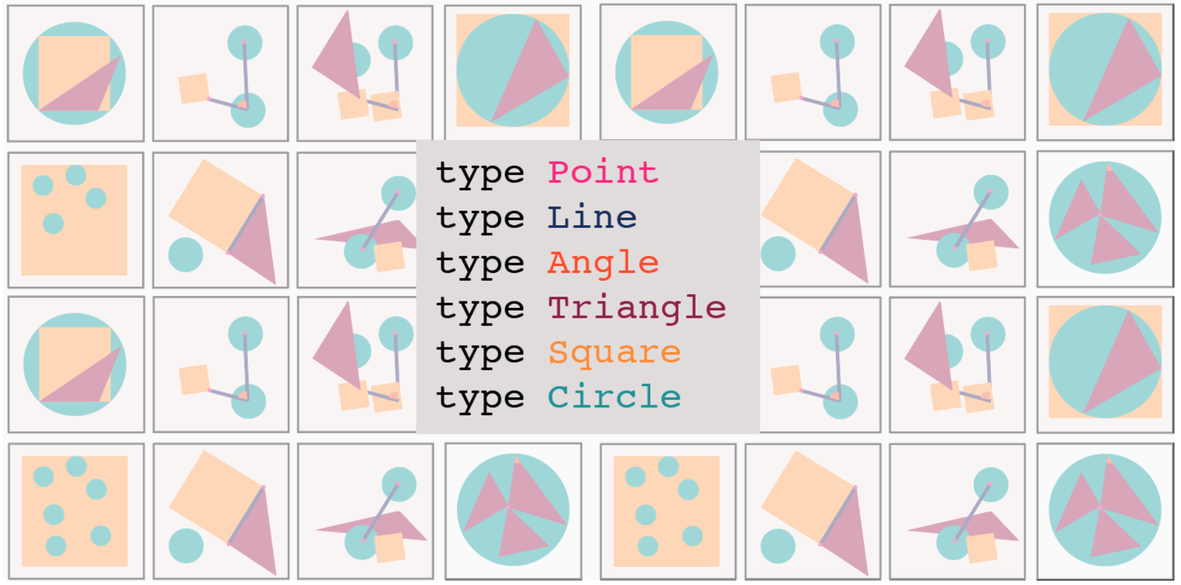
- Greater accessibility to novices

Benefits

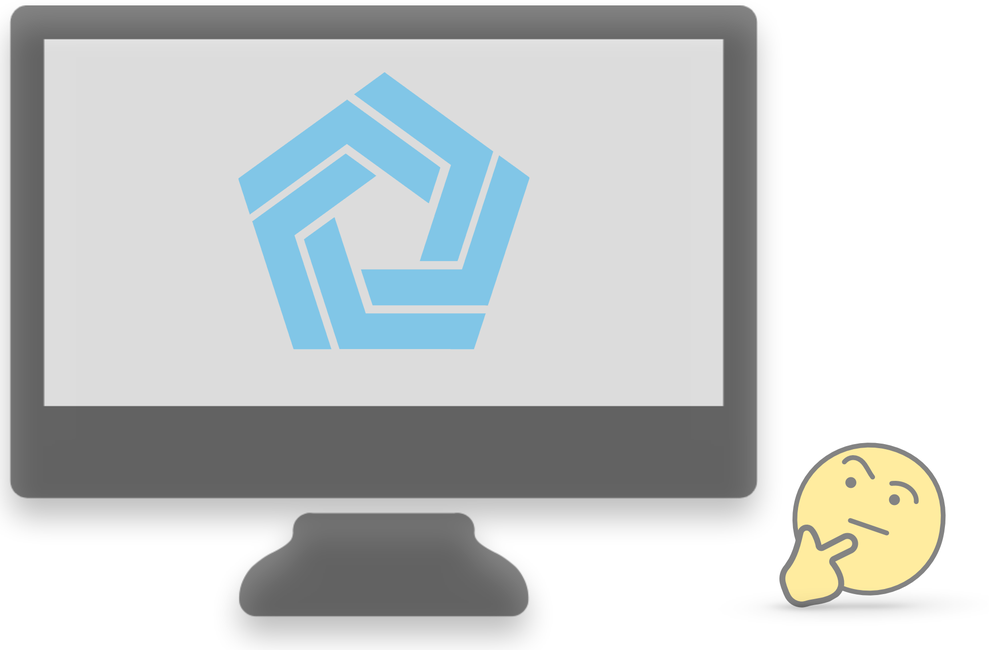
- Greater accessibility to novices
- Separation of content and presentation

Benefits

- Greater accessibility to novices
- Separation of content and presentation
- Large-scale generation and evolution of collections

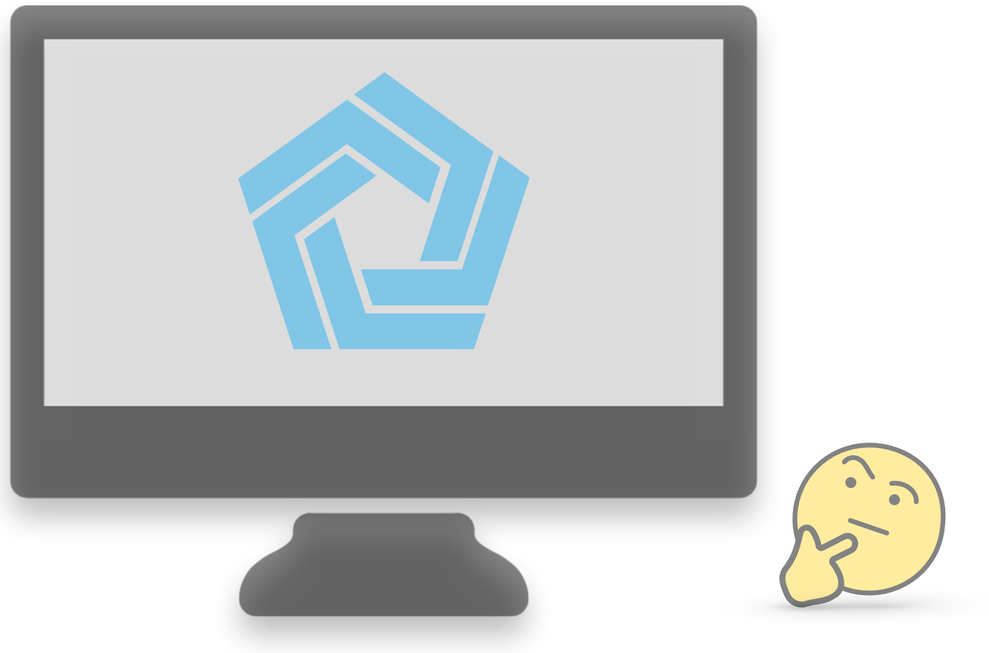


Talk outline



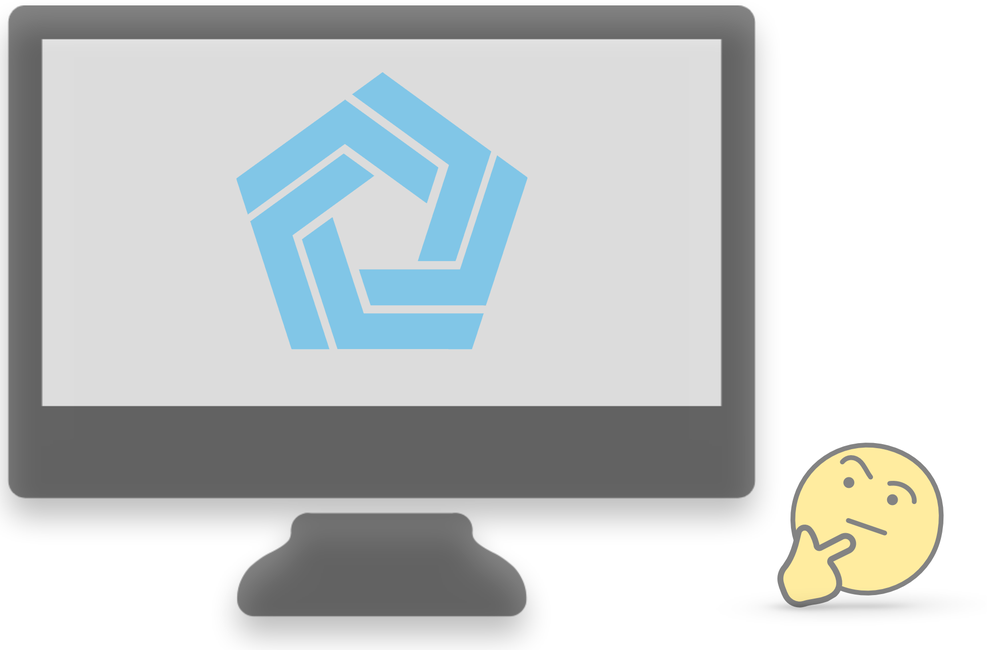
Talk outline

- Related tools



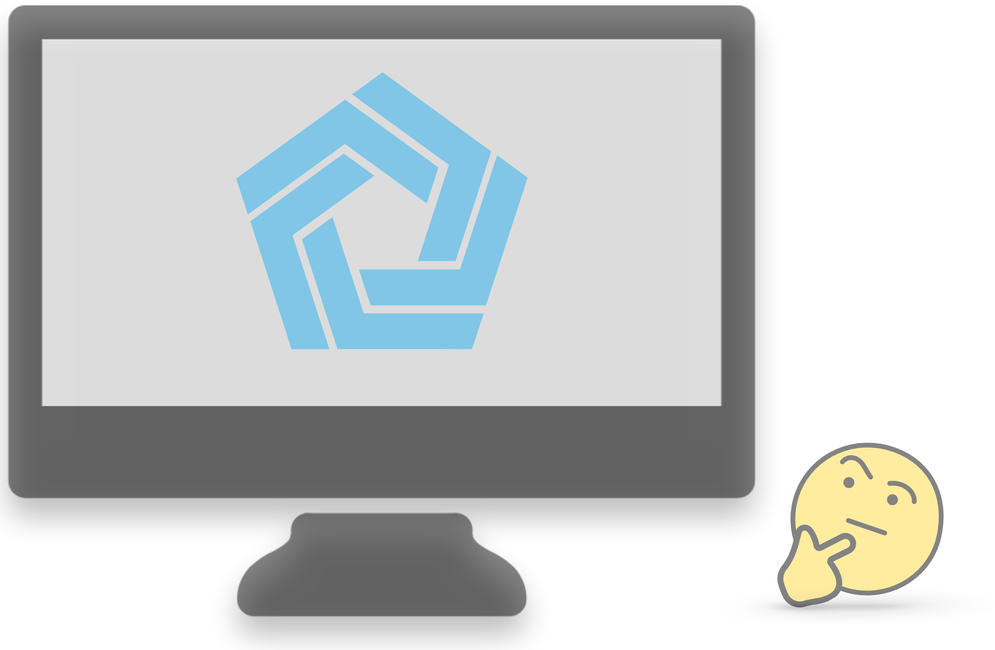
Talk outline

- Related tools
- System design principles



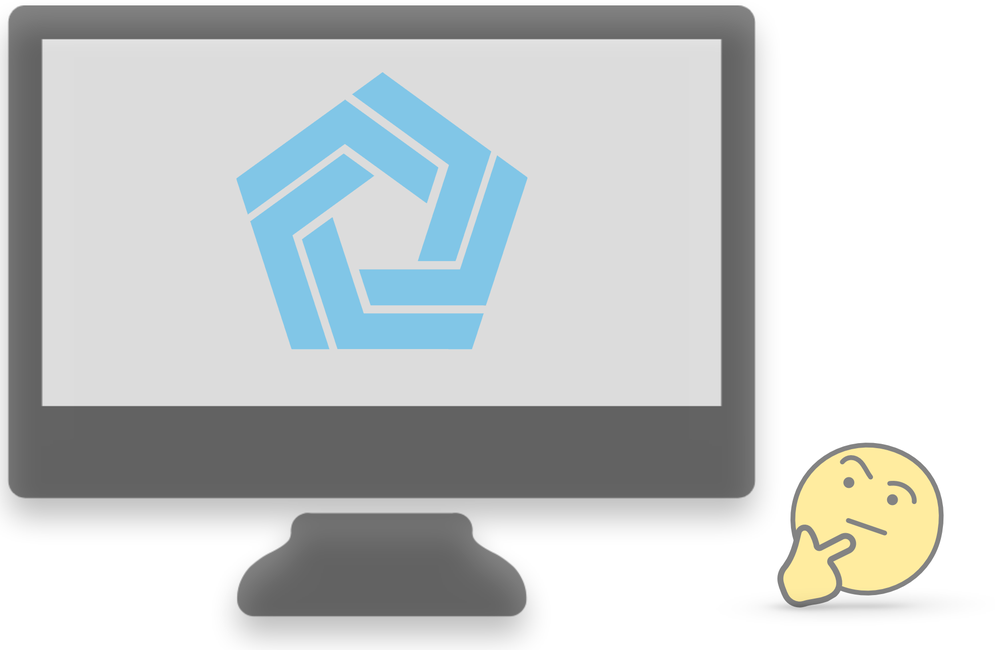
Talk outline

- Related tools
- System design principles
- System walkthrough



Talk outline

- Related tools
- System design principles
- System walkthrough
- The system in action



Goals & related work

What would an ideal diagramming tool look like?

What would an ideal diagramming tool look like?

Design goals for a mathematical diagramming tool

Design goals for a mathematical diagramming tool

- **Familiarity** – input should look like ordinary mathematics

Design goals for a mathematical diagramming tool

Familiarity – input should look like ordinary mathematics

Universality – should be possible to talk about any domain

Design goals for a mathematical diagramming tool

Familiarity – input should look like ordinary mathematics

Universality – should be possible to talk about any domain

Extensibility – should be possible to use any visualization

Design goals for a mathematical diagramming tool

Familiarity – input should look like ordinary mathematics

Universality – should be possible to talk about any domain

Extensibility – should be possible to use any visualization

Beauty – no inherent limitation of visual sophistication

Design goals for a mathematical diagramming tool

Familiarity – input should look like ordinary mathematics

Universality – should be possible to talk about any domain

Extensibility – should be possible to use any visualization

Beauty – no inherent limitation of visual sophistication

Productivity – should be easy to iterate on diagrams

Design goals for a mathematical diagramming tool

Familiarity – input should look like ordinary mathematics

Universality – should be possible to talk about any domain

Extensibility – should be possible to use any visualization

Beauty – no inherent limitation of visual sophistication

Productivity – should be easy to iterate on diagrams

Reusability – effort should be amortized across diagrams

Existing tools don't consider the larger process

specify the mathematical idea

↓

translate it into a visual interpretation

↓

make a high-quality drawing

Existing tools don't consider the larger process

specify the mathematical idea

translate it into a visual interpretation

make a high-quality drawing

Existing tools don't consider the larger process

specify the mathematical idea

↓

translate it into a visual interpretation

GUI tools (e.g. Illustrator)

↓

make a high-quality drawing

Existing tools don't consider the larger process

specify the mathematical idea

↓

translate it into a visual interpretation


GUI tools (e.g. Illustrator)


plotting tools (e.g. Mathematica)

↓

make a high-quality drawing

Existing tools don't consider the larger process

specify the mathematical idea

↓

translate it into a visual interpretation


GUI tools (e.g. Illustrator)


plotting tools (e.g. Mathematica)

↓


graphical specification languages
(e.g. TikZ)

make a high-quality drawing

Existing tools don't consider the larger process

specify the mathematical idea

translate it into a visual interpretation

GUI tools (e.g. Illustrator)

plotting tools (e.g. Mathematica)

graphical specification languages
(e.g. TikZ)

make a high-quality drawing

Existing tools don't consider the larger process

specify the mathematical idea

domain-specific packages
(e.g. GraphViz)

translate it into a visual interpretation

GUI tools (e.g. Illustrator)

plotting tools (e.g. Mathematica)

graphical specification languages
(e.g. TikZ)

make a high-quality drawing

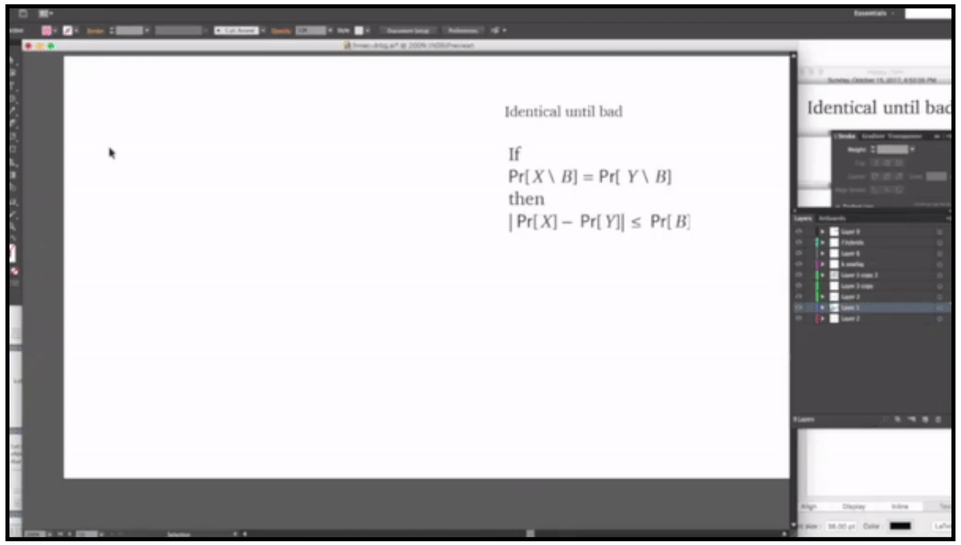
Goal: provide translation; avoid tedium

Goal: provide translation; avoid tedium

Less of this:

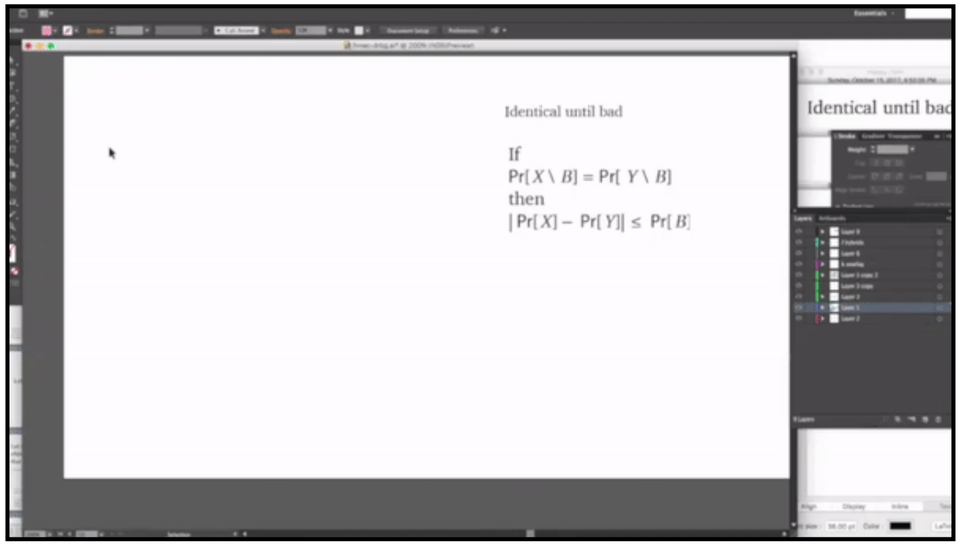
Goal: provide translation; avoid tedium

Less of this:



Goal: provide translation; avoid tedium

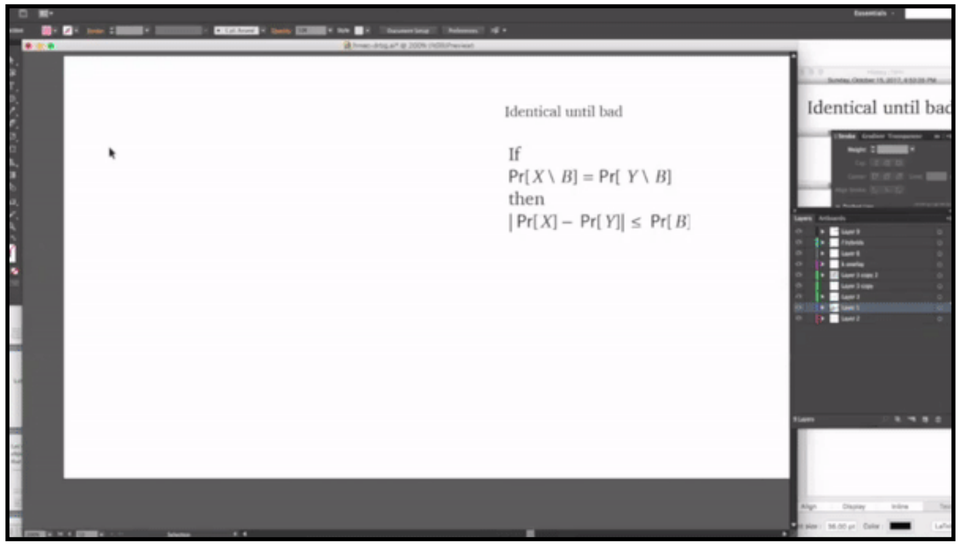
Less of this:




```
\usepackage{tikz}
\begin{document}
\pagestyle{empty}
\begin{tikzpicture}
   \begin{scope}[shift={(3cm,-5cm)},fill opacity=0.5]
      \draw[fill=red,draw=black](0,0) circle(3);
      \draw[fill=green,draw=black](-1.5,1) circle(3);
      \draw[fill=blue,draw=black](1.5,1) circle(3);
      \node at (0,4) (A) {\large\textbf{A}};
      \node at (-2,1) (B) {\large\textbf{B}};
      \node at (2,1) (C) {\large\textbf{C}};
      \node at (0,0) (D) {\large\textbf{D}};
   \end{scope}
\end{tikzpicture}
\end{document}
```

Goal: provide translation; avoid tedium

Less of this:

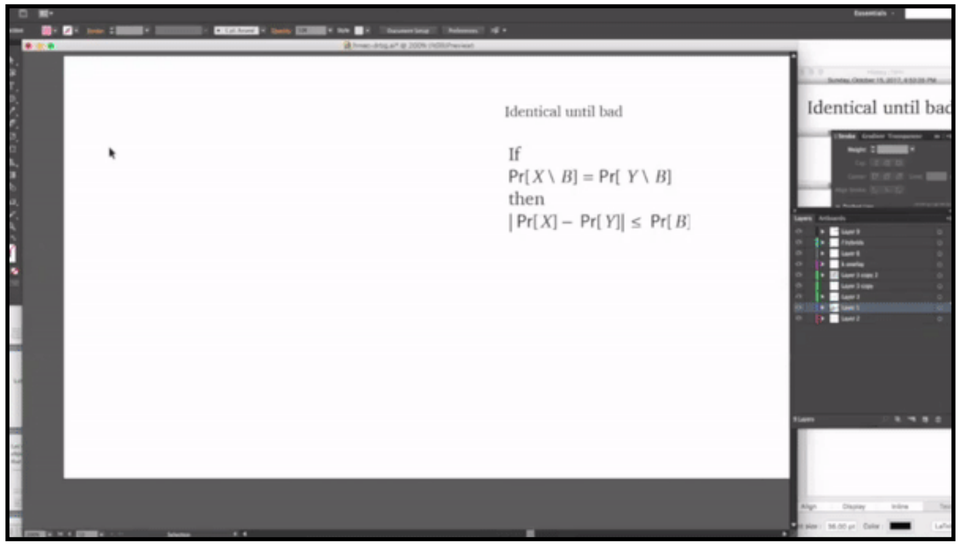



```
\usepackage{tikz}
\begin{document}
\pagestyle{empty}
\begin{tikzpicture}
   \begin{scope}[shift={(3cm,-5cm)},fill opacity=0.5]
      \draw[fill=red,draw=black](0,0) circle(3);
      \draw[fill=green,draw=black](-1.5,1) circle(3);
      \draw[fill=blue,draw=black](1.5,1) circle(3);
      \node at (0,4) (A) {\large\textbf{A}};
      \node at (-2,1) (B) {\large\textbf{B}};
      \node at (2,1) (C) {\large\textbf{C}};
      \node at (0,0) (D) {\large\textbf{D}};
   \end{scope}
\end{tikzpicture}
\end{document}
```

More of this:

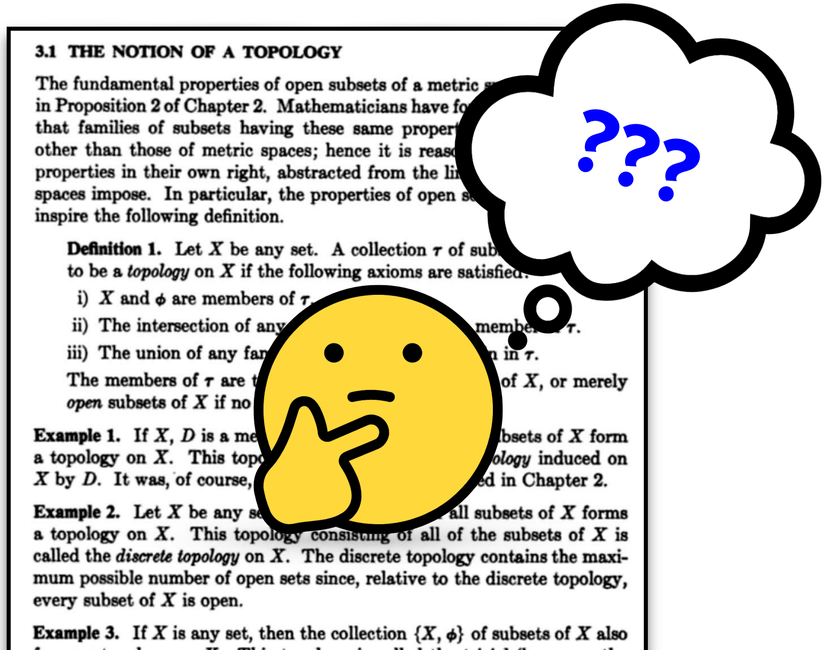
Goal: provide translation; avoid tedium

Less of this:




```
\usepackage{tikz}
\begin{document}
\pagestyle{empty}
\begin{tikzpicture}
   \begin{scope}[shift={(3cm,-5cm)},fill opacity=0.5]
      \draw[fill=red,draw=black](0,0) circle(3);
      \draw[fill=green,draw=black](-1.5,1) circle(3);
      \draw[fill=blue,draw=black](1.5,1) circle(3);
      \node at (0,4) (A) {\large\textbf{A}};
      \node at (-2,1) (B) {\large\textbf{B}};
      \node at (2,1) (C) {\large\textbf{C}};
      \node at (0,0) (D) {\large\textbf{D}};
   \end{scope}
\end{tikzpicture}
\end{document}
```

More of this:



3.1 THE NOTION OF A TOPOLOGY

The fundamental properties of open subsets of a metric s
in Proposition 2 of Chapter 2. Mathematicians have fo
that families of subsets having these same proper
other than those of metric spaces; hence it is reas
properties in their own right, abstracted from the li
spaces impose. In particular, the properties of open s
inspire the following definition.

Definition 1. Let X be any set. A collection τ of sub
to be a *topology* on X if the following axioms are satisfied:

i) X and ϕ are members of τ.

ii) The intersection of any member of τ.

iii) The union of any fa in τ.

The members of τ are of X, or merely *open* subsets of X if no

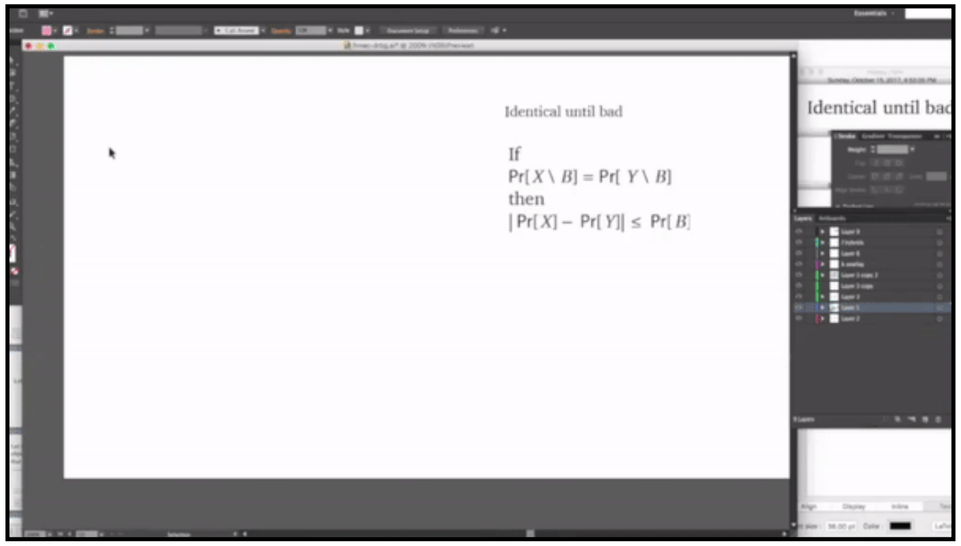
Example 1. If X, D is a me bsets of X form a topology on X. This topo ology induced on X by D. It was, of course, ed in Chapter 2.

Example 2. Let X be any se all subsets of X forms a topology on X. This topology consisting of all of the subsets of X is called the *discrete topology* on X. The discrete topology contains the maximum possible number of open sets since, relative to the discrete topology, every subset of X is open.

Example 3. If X is any set, then the collection $\{X, \phi\}$ of subsets of X also

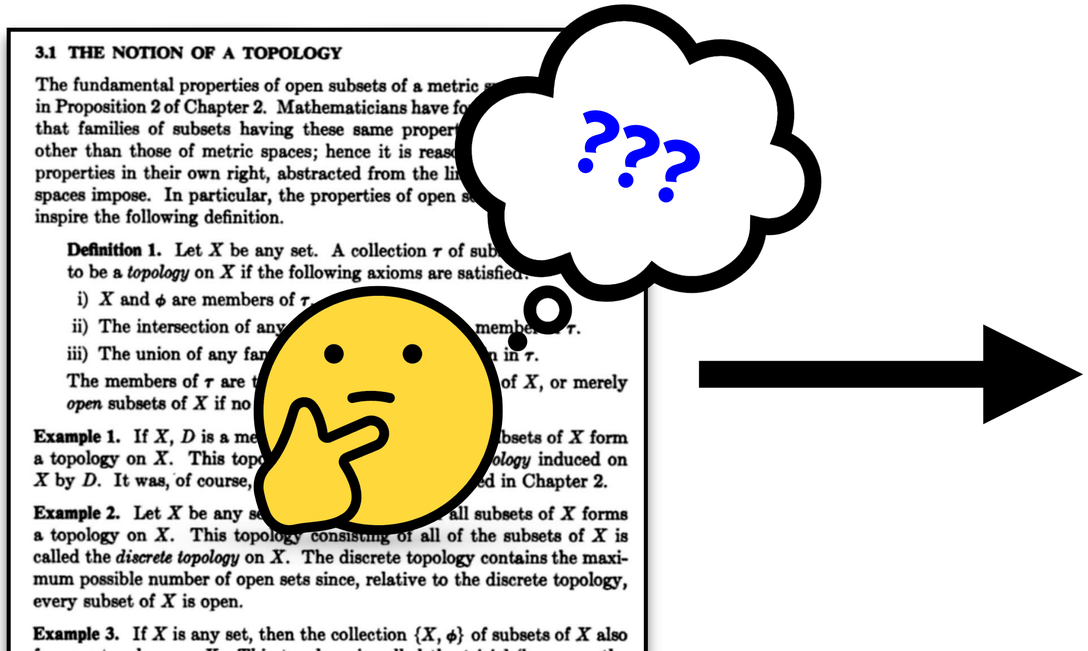
Goal: provide translation; avoid tedium

Less of this:



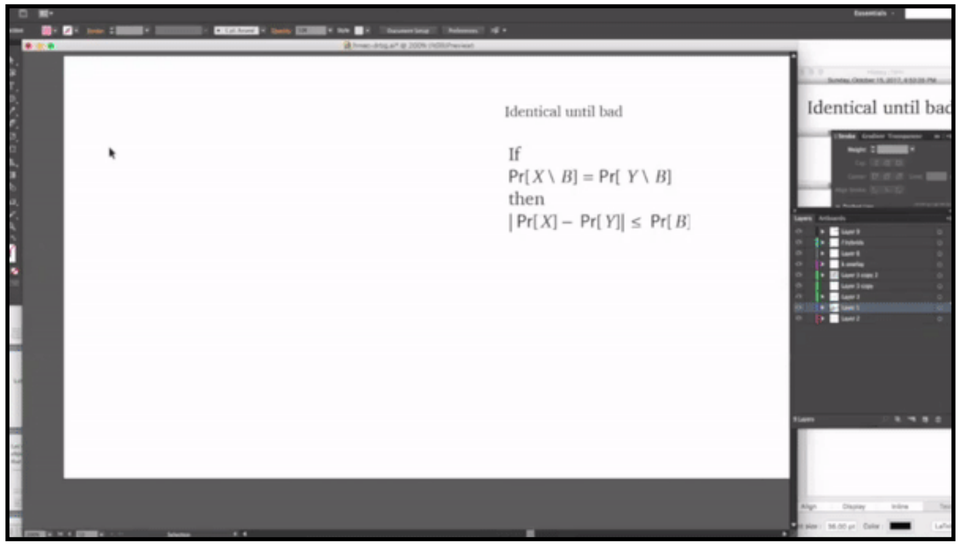

```latex
\usepackage{tikz}
\begin{document}
\pagestyle{empty}
\begin{tikzpicture}
   \begin{scope}[shift={(3cm,-5cm)},fill opacity=0.5]
      \draw[fill=red,draw=black](0,0) circle(3);
      \draw[fill=green,draw=black](-1.5,1) circle(3);
      \draw[fill=blue,draw=black](1.5,1) circle(3);
      \node at (0,4) (A) {\large\textbf{A}};
      \node at (-2,1) (B) {\large\textbf{B}};
      \node at (2,1) (C) {\large\textbf{C}};
      \node at (0,0) (D) {\large\textbf{D}};
   \end{scope}
\end{tikzpicture}
\end{document}
```

More of this:



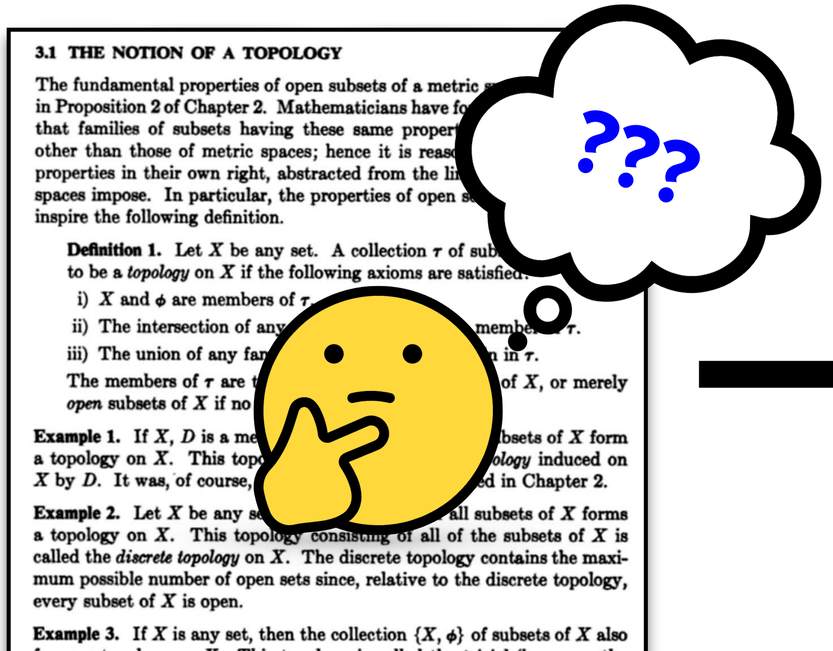
Goal: provide translation; avoid tedium

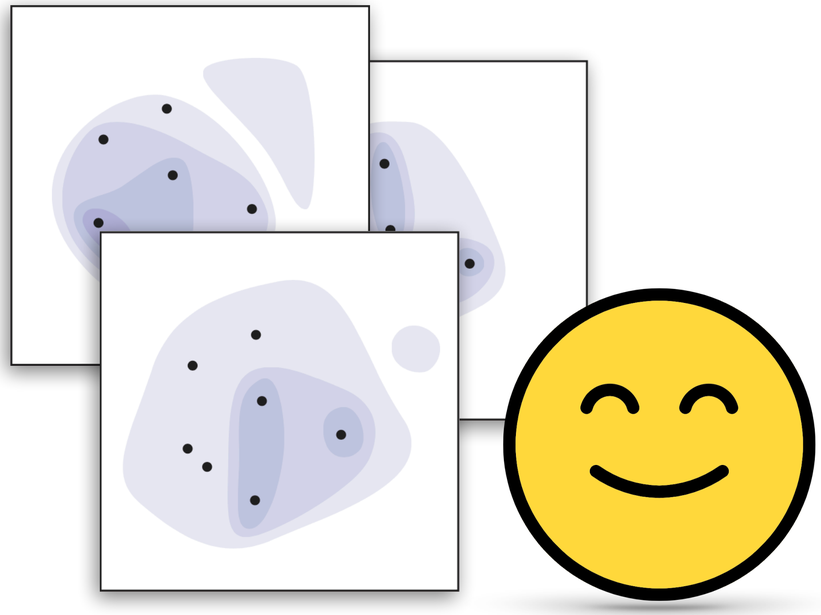
Less of this:




```
\usepackage{tikz}
\begin{document}
\pagestyle{empty}
\begin{tikzpicture}
   \begin{scope}[shift={(3cm,-5cm)},fill opacity=0.5]
      \draw[fill=red,draw=black](0,0) circle(3);
      \draw[fill=green,draw=black](-1.5,1) circle(3);
      \draw[fill=blue,draw=black](1.5,1) circle(3);
      \node at (0,4) (A) {\large\textbf{A}};
      \node at (-2,1) (B) {\large\textbf{B}};
      \node at (2,1) (C) {\large\textbf{C}};
      \node at (0,0) (D) {\large\textbf{D}};
   \end{scope}
\end{tikzpicture}
\end{document}
```

More of this:

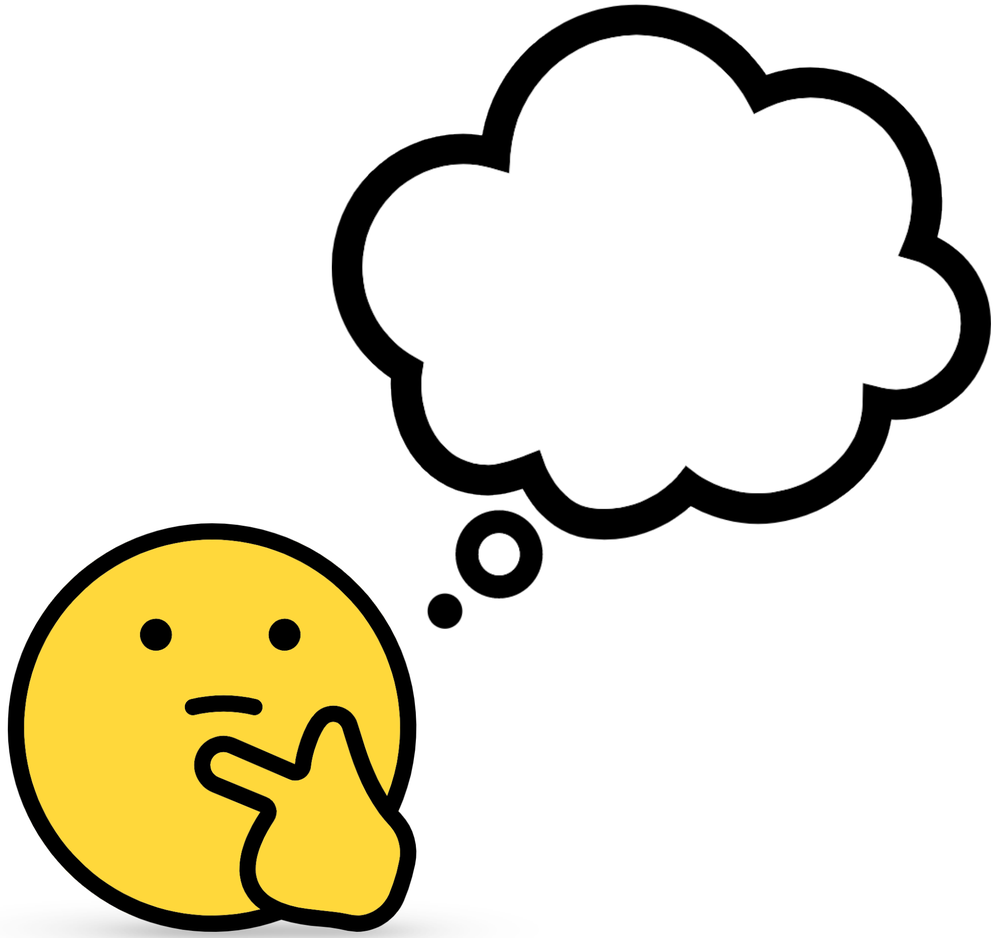


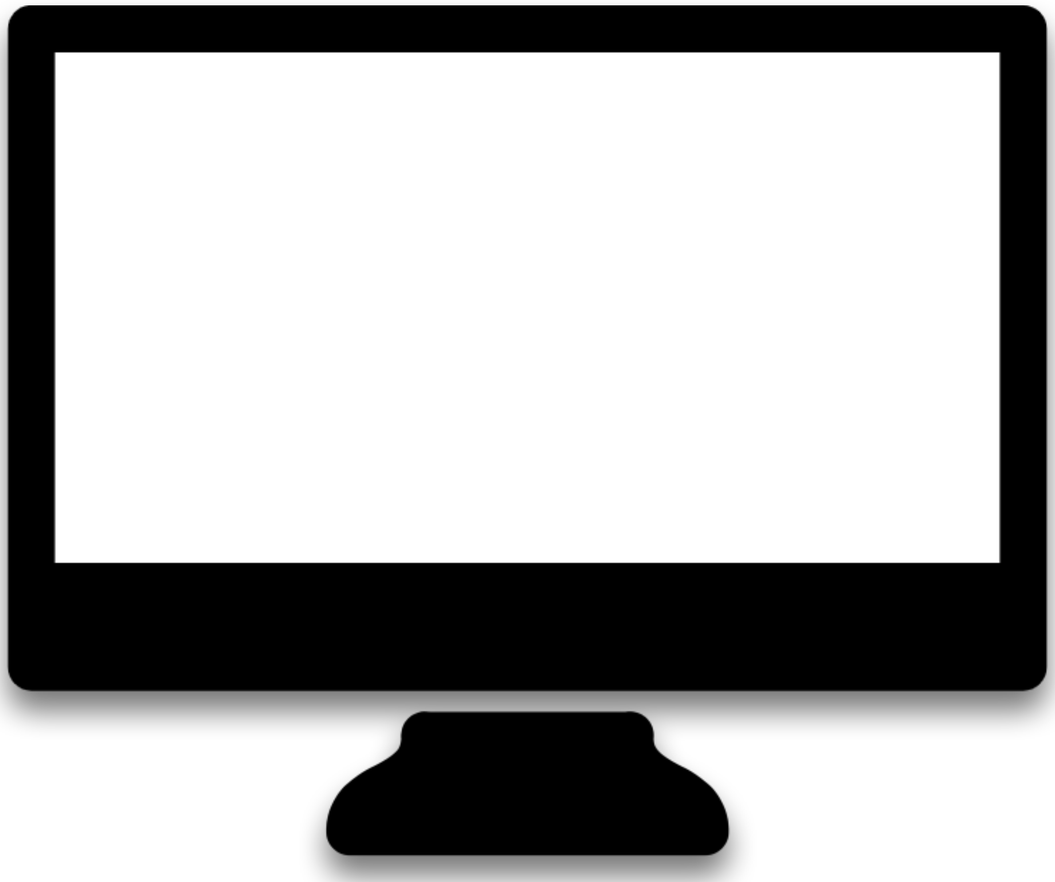


System design principles

Modeling how people visualize abstract ideas by hand

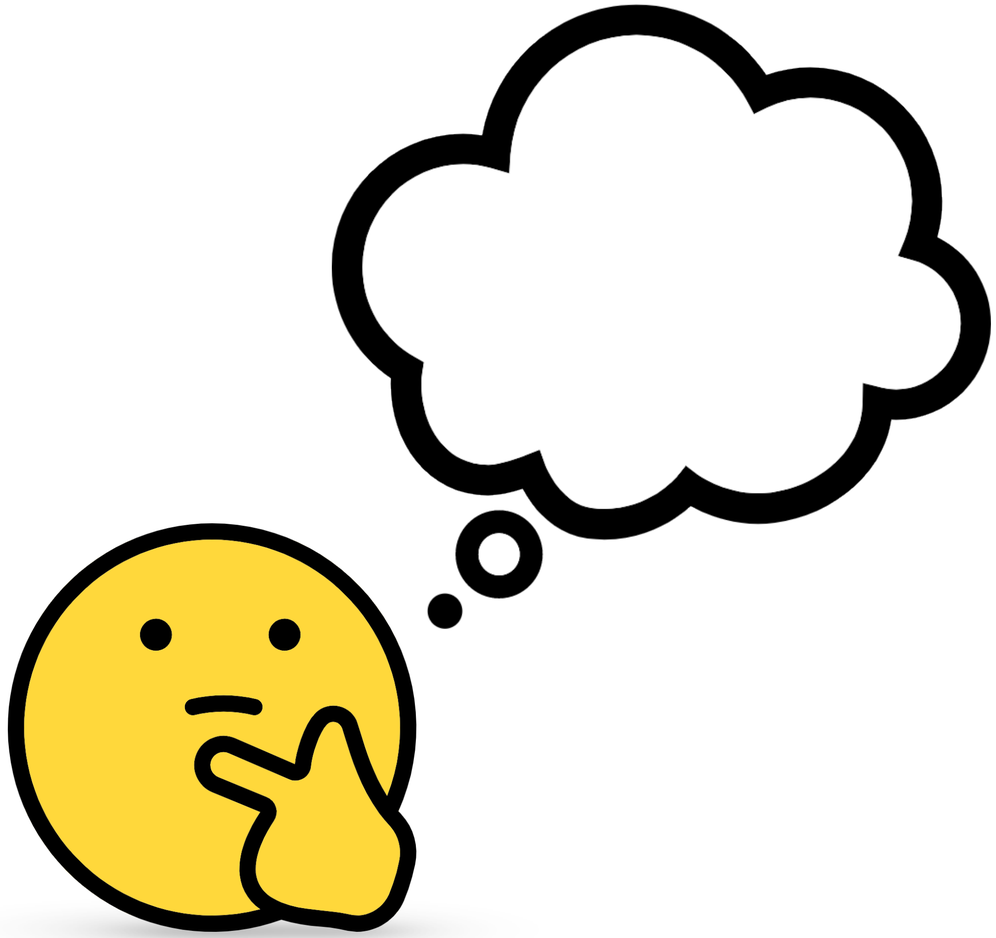
There exist sets A, B such that $B \subseteq A$.

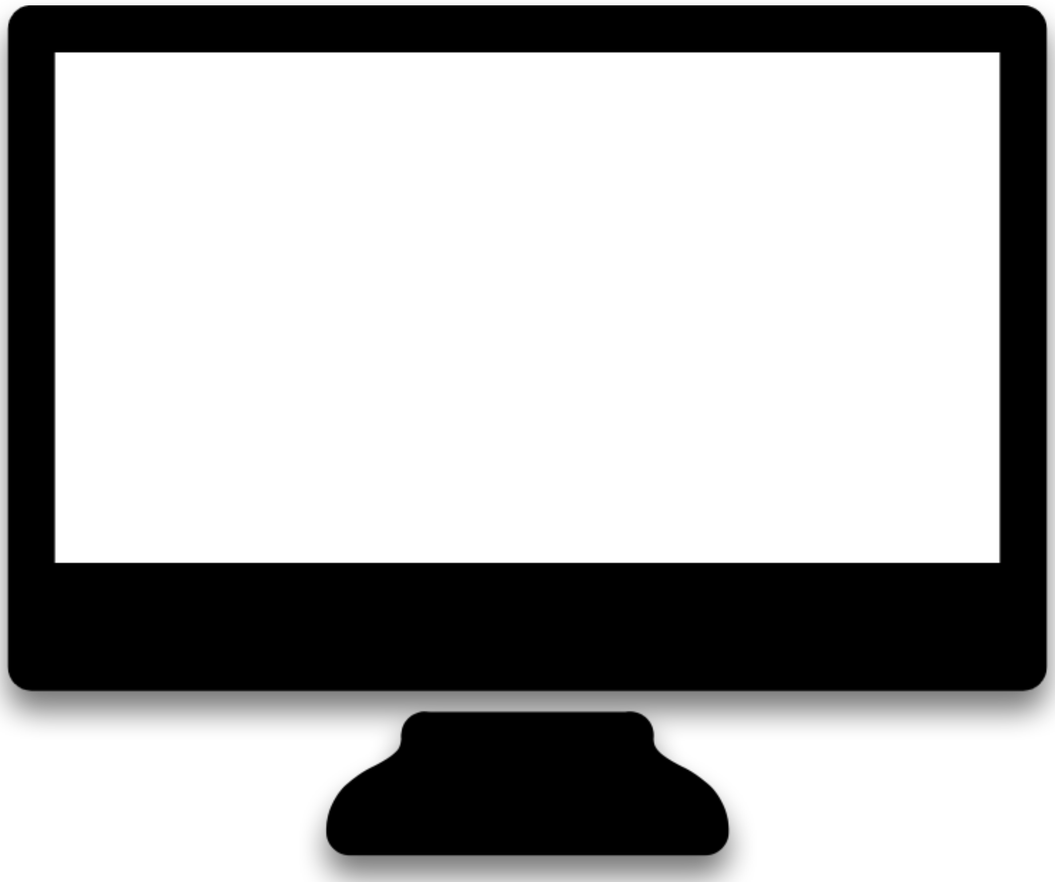




Modeling how people visualize abstract ideas by hand

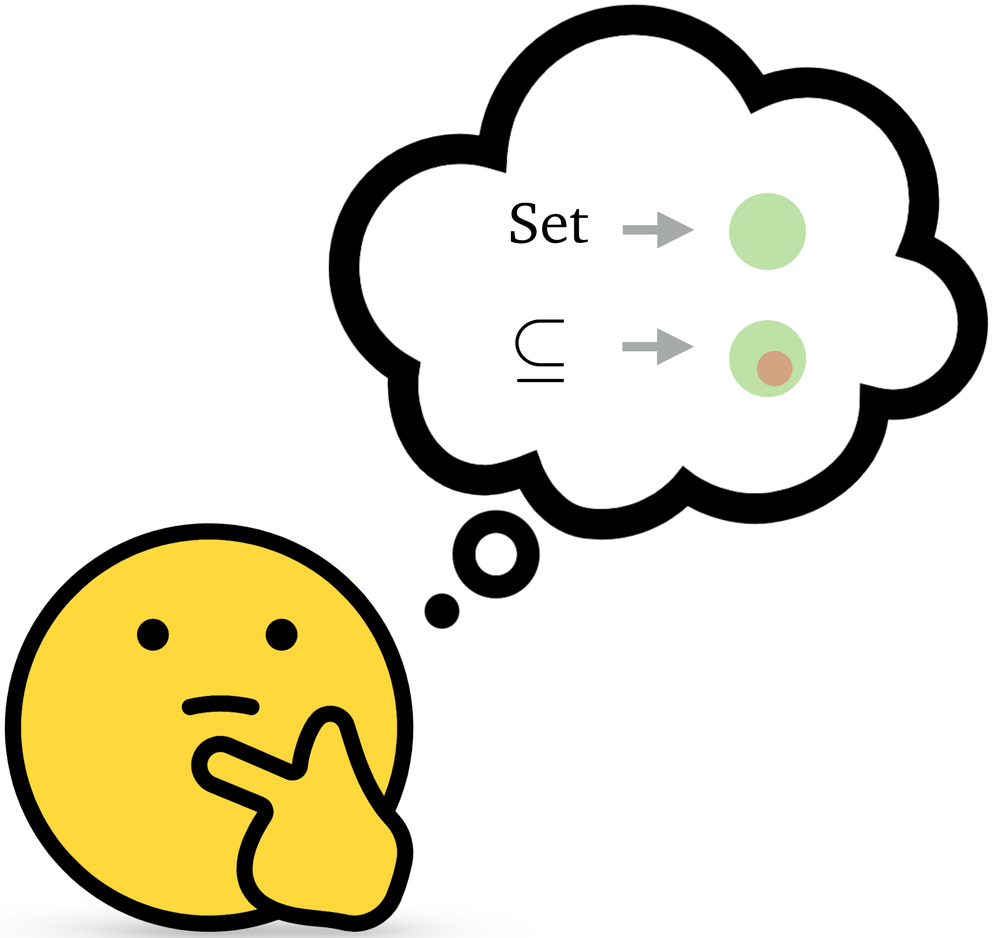
There exist sets A, B such that $B \subseteq A$.

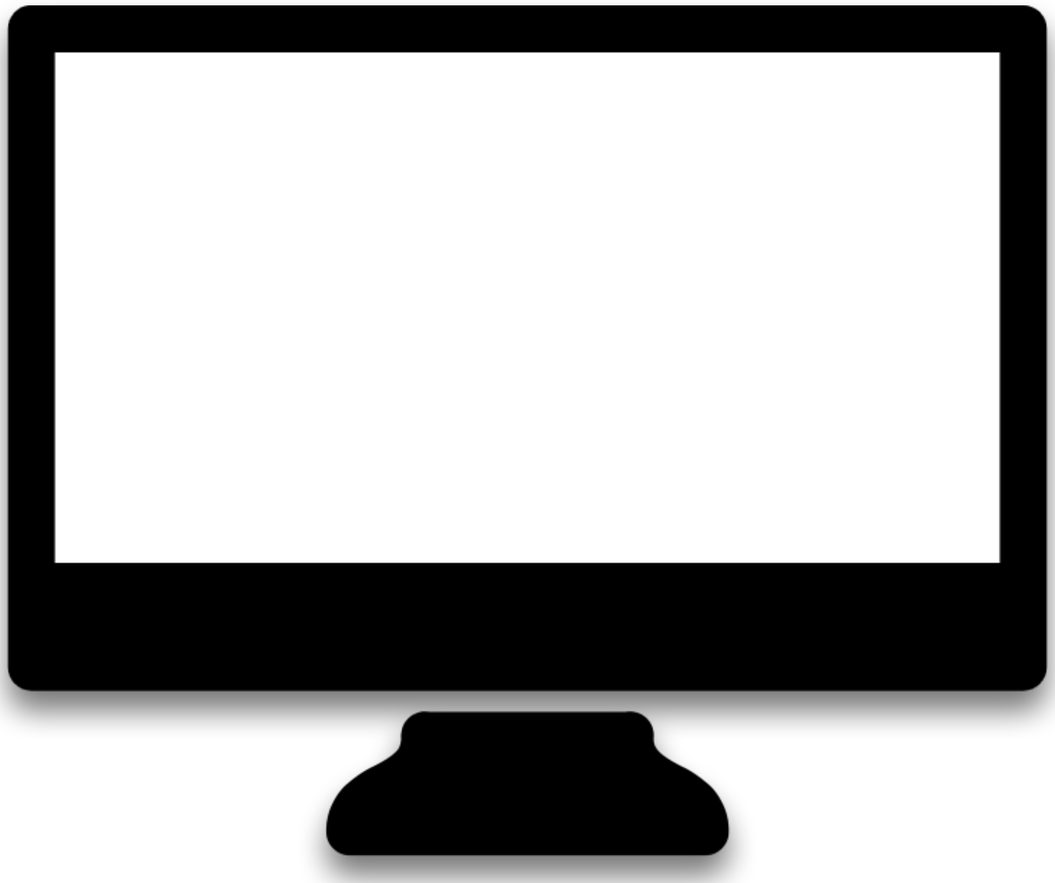




Modeling how people visualize abstract ideas by hand

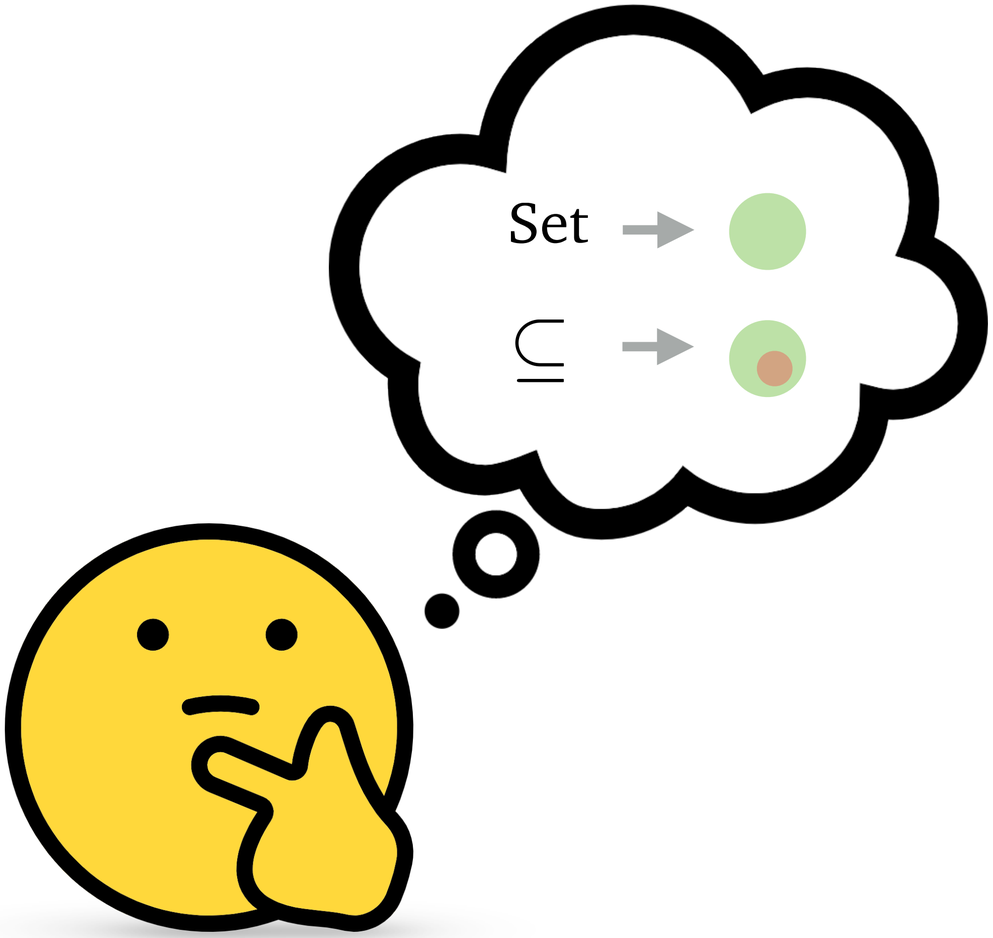
There exist sets A, B such that $B \subseteq A$.

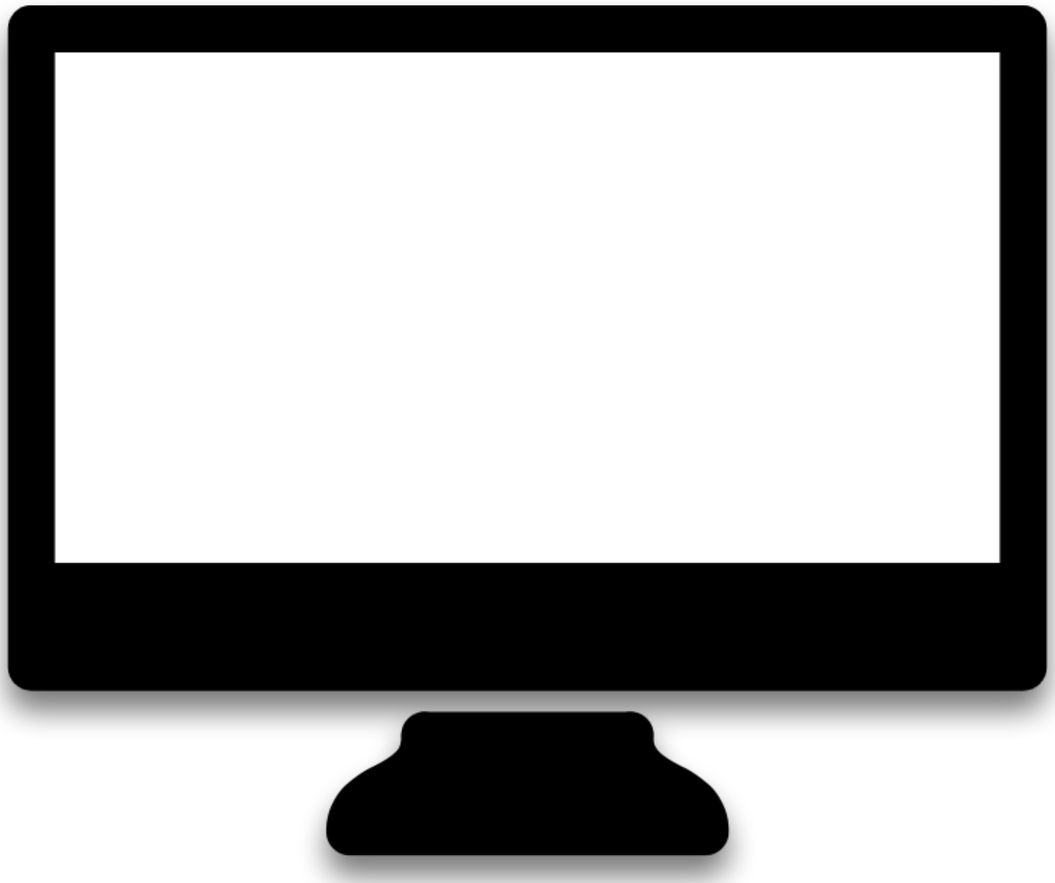




Modeling how people visualize abstract ideas by hand

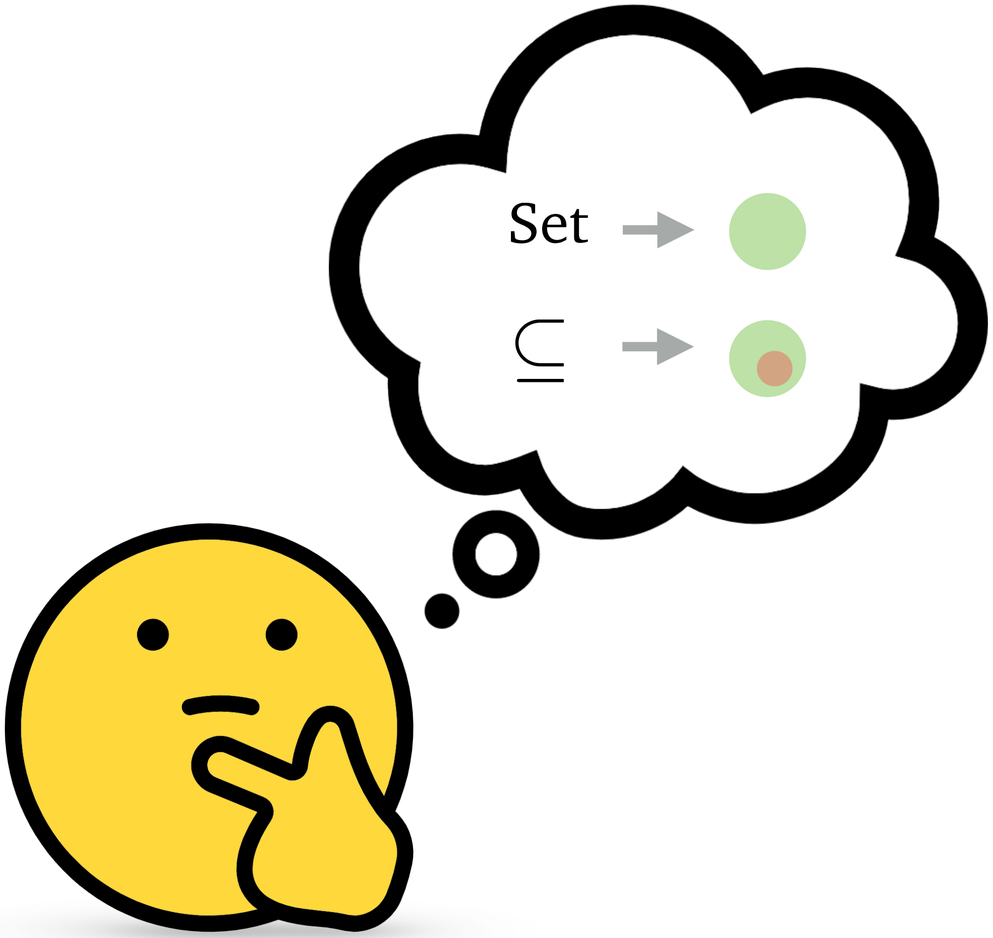
There exist sets A, B such that $B \subseteq A$.

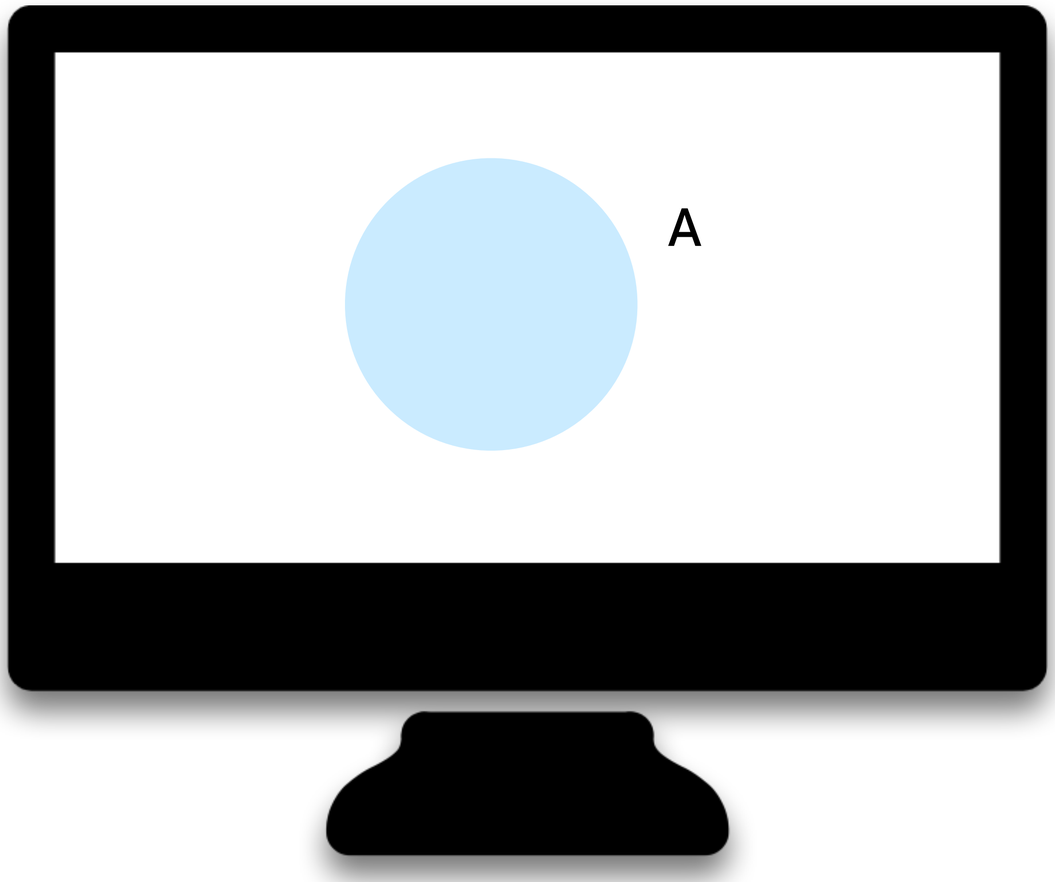




Modeling how people visualize abstract ideas by hand

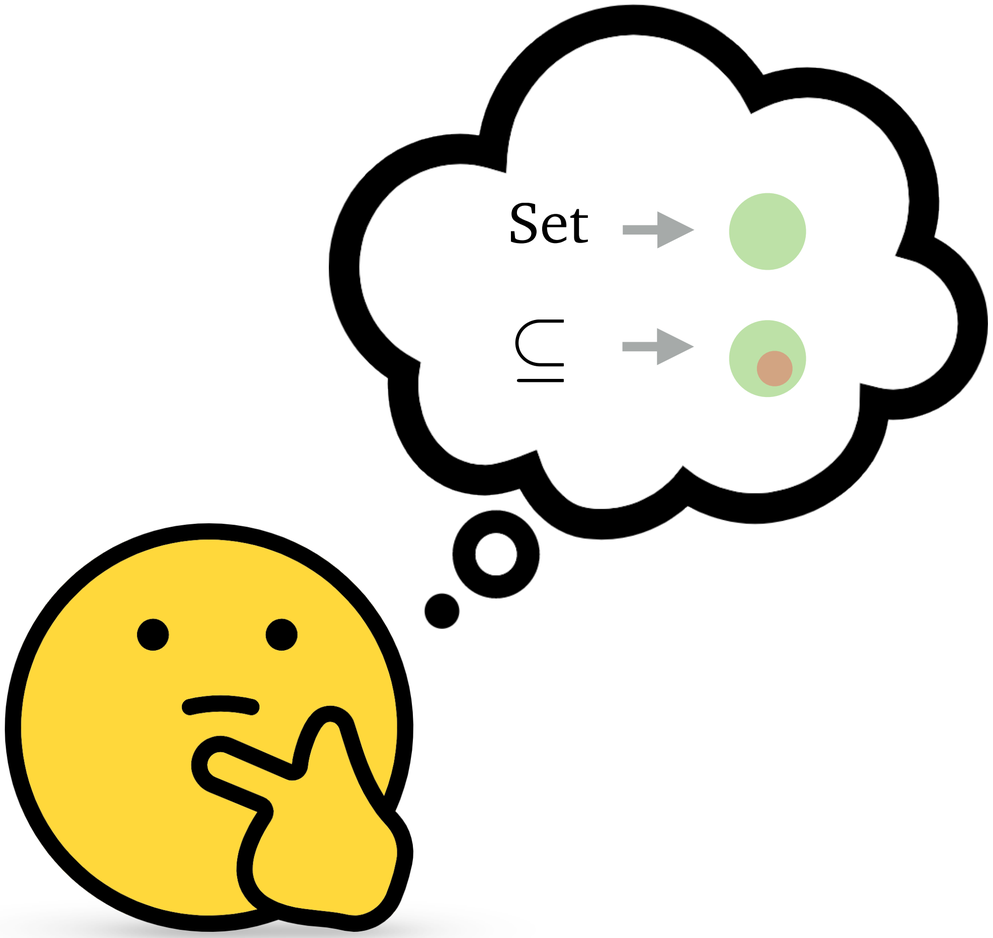
There exist sets A, B such that $B \subseteq A$.

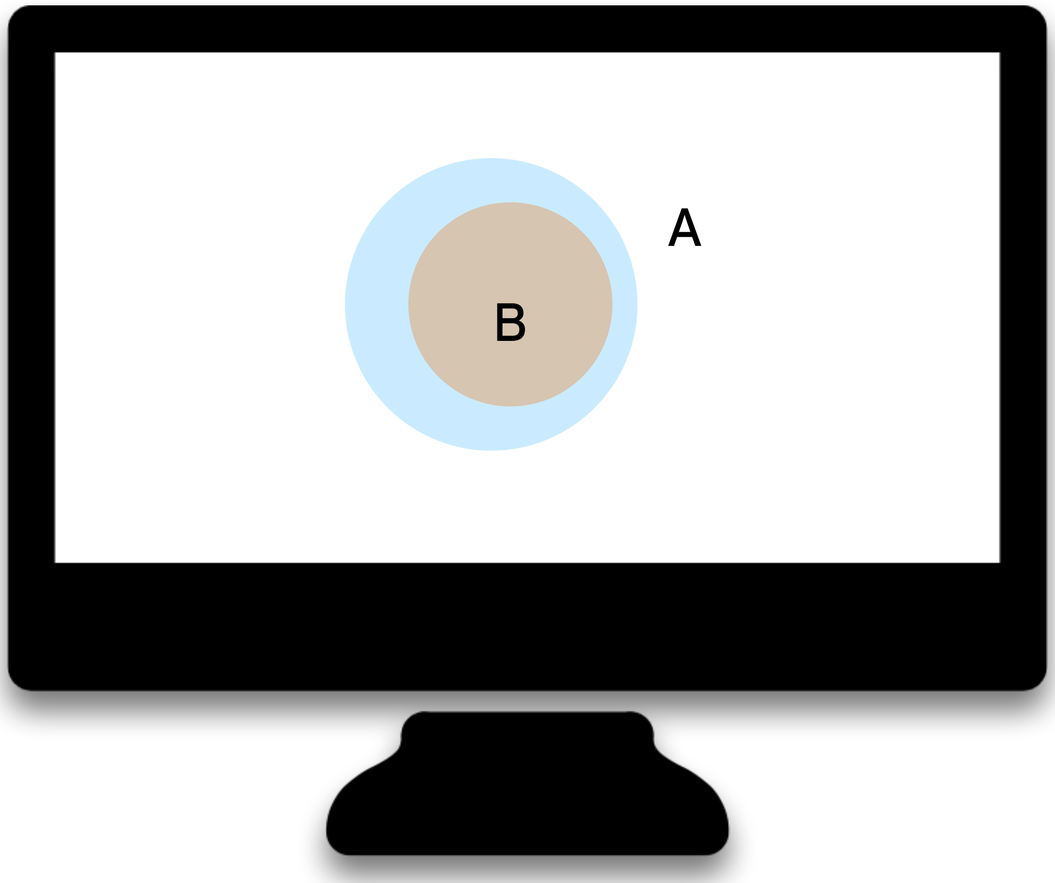




Modeling how people visualize abstract ideas by hand

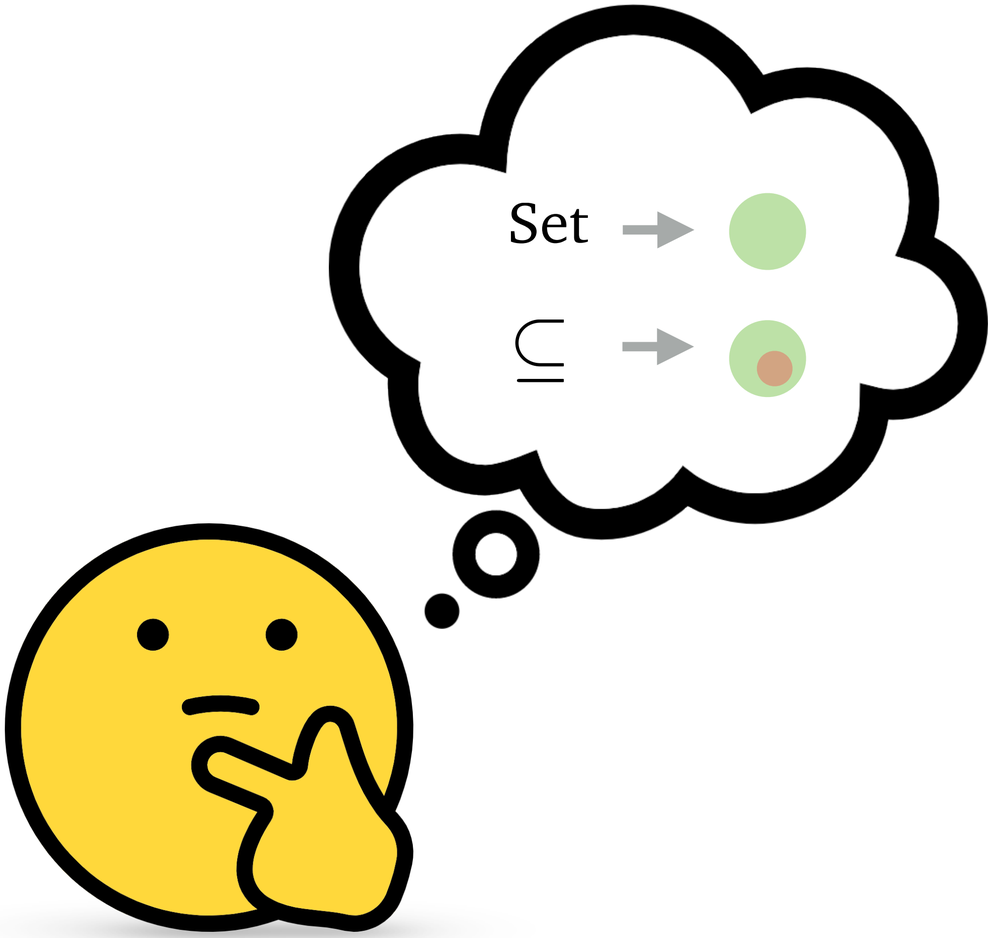
There exist sets A, B such that $B \subseteq A$.

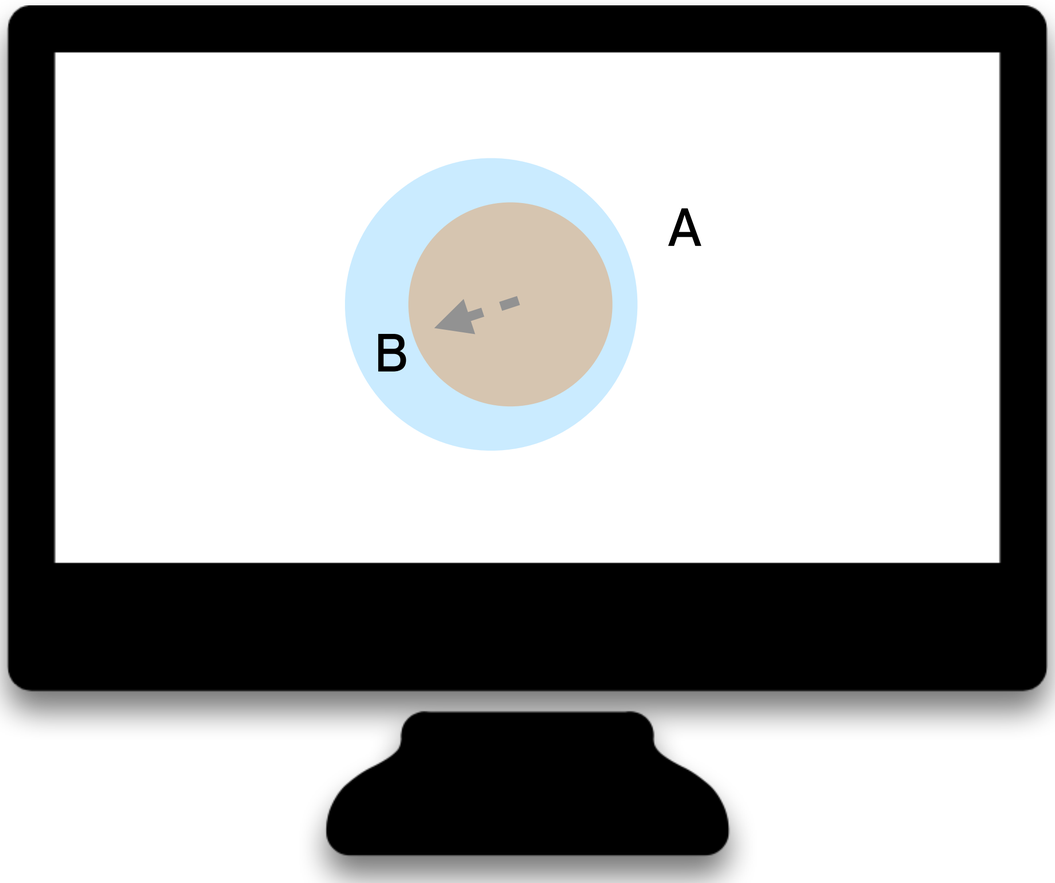




Modeling how people visualize abstract ideas by hand

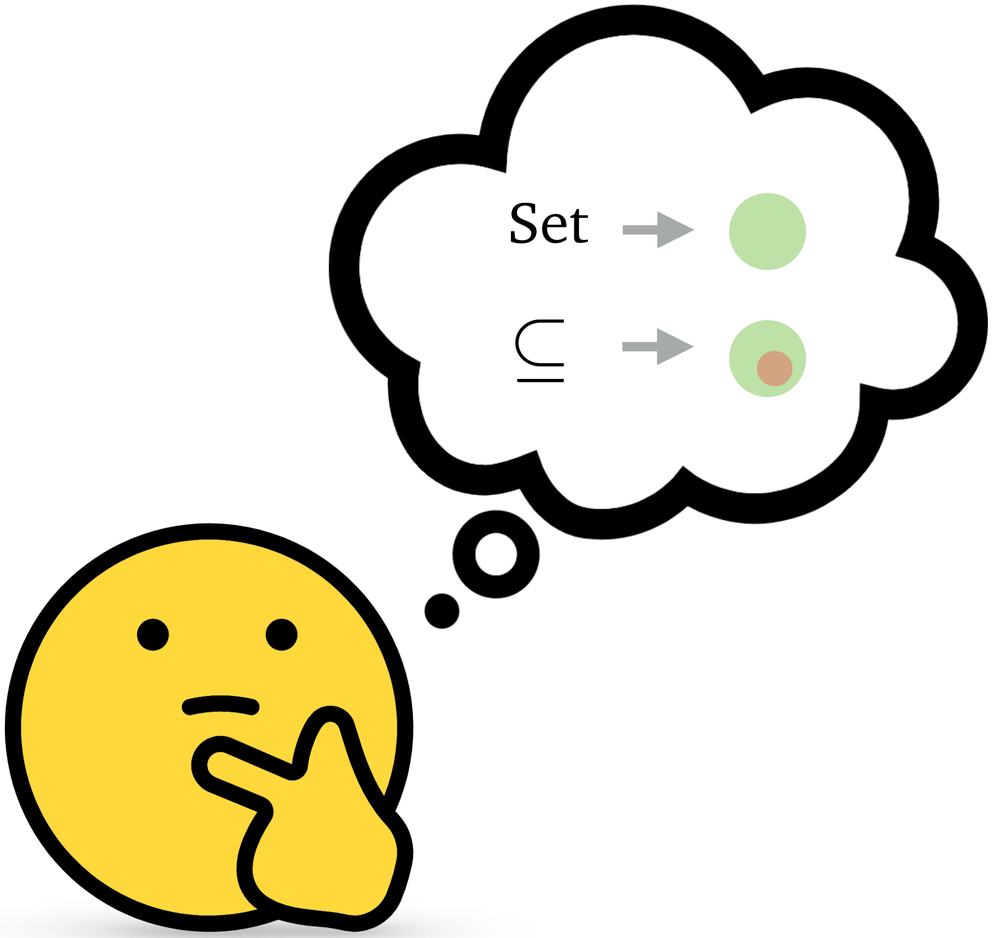
There exist sets A, B such that $B \subseteq A$.

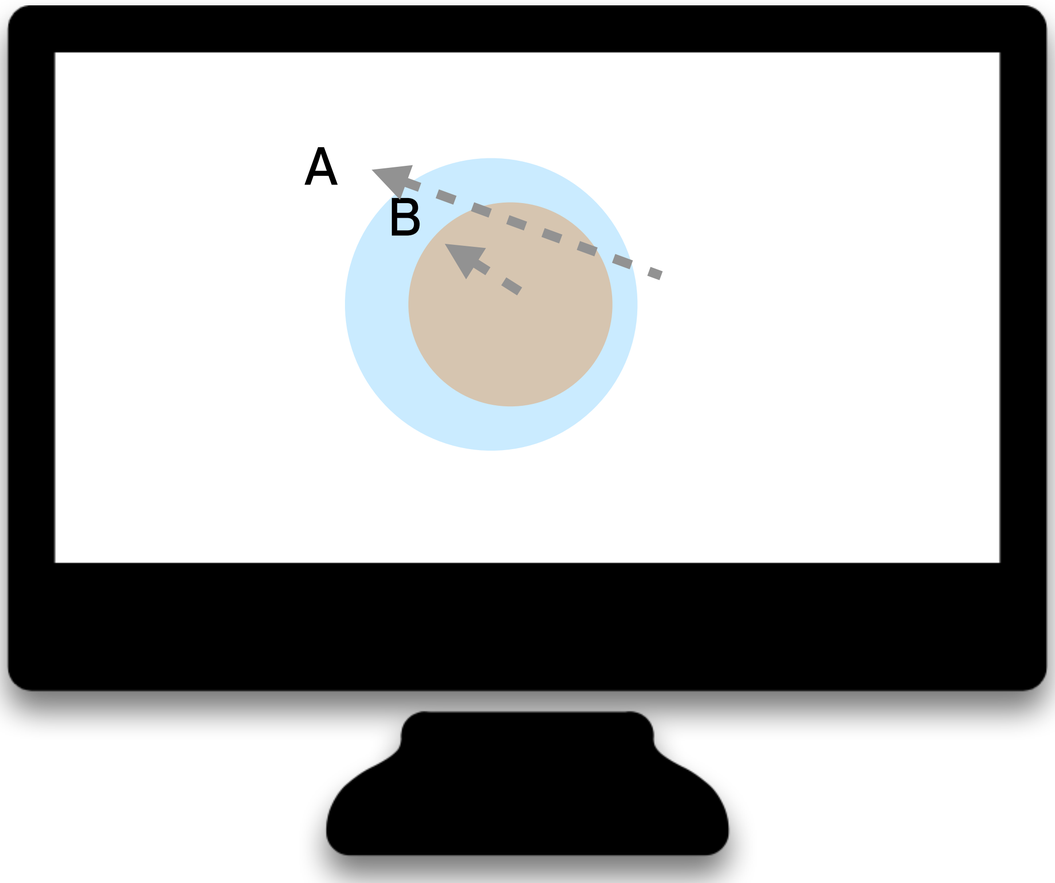




Modeling how people visualize abstract ideas by hand

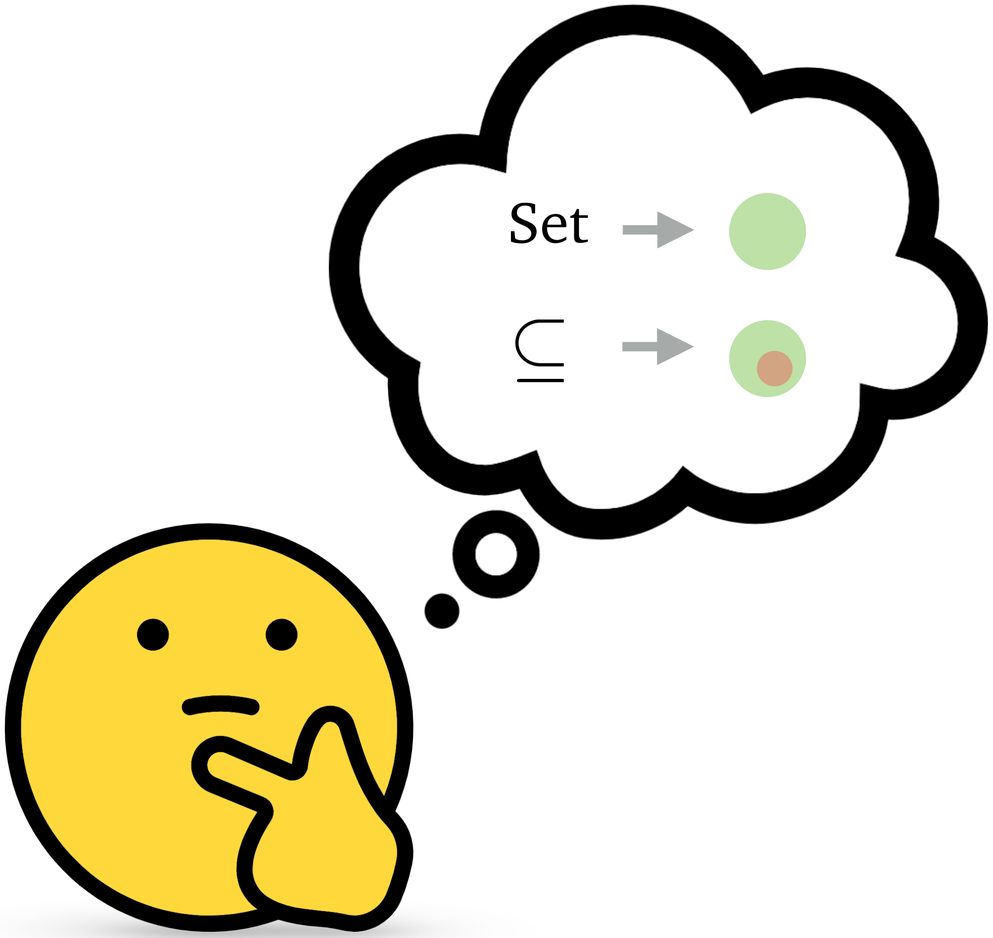
There exist sets A, B such that $B \subseteq A$.

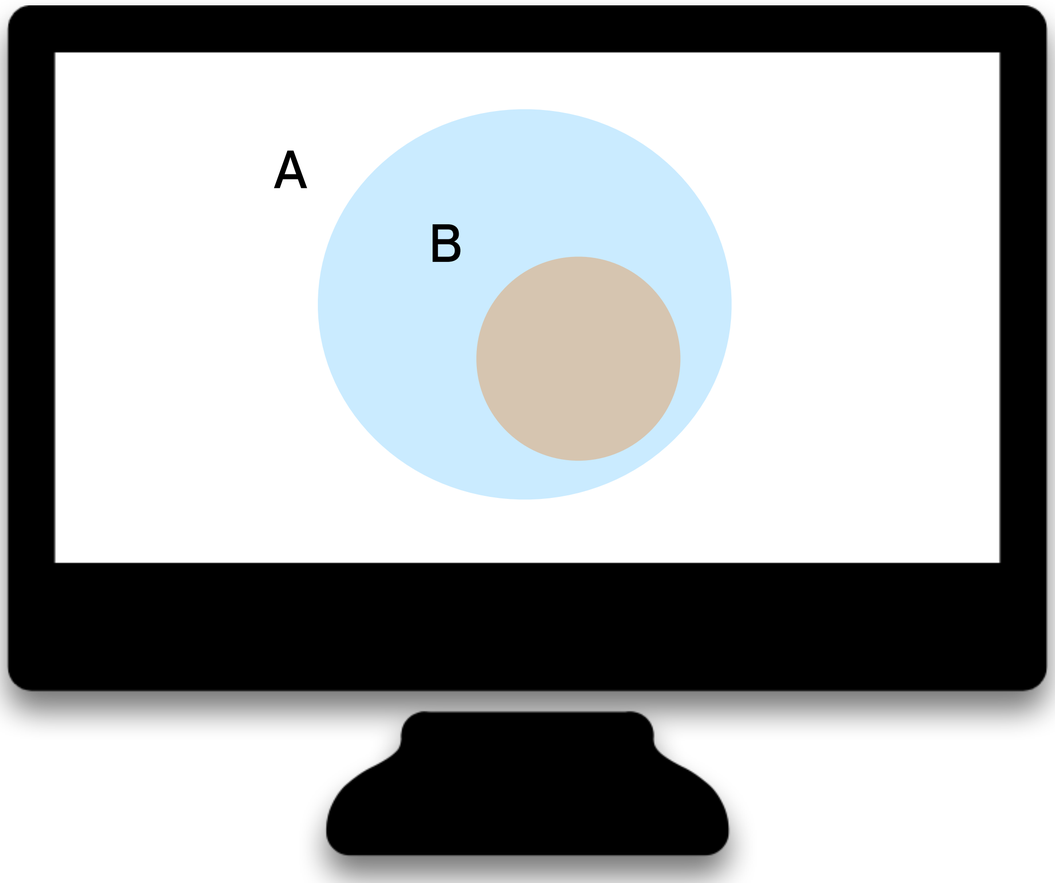




Modeling how people visualize abstract ideas by hand

There exist sets A, B such that $B \subseteq A$.





General design principles for math diagramming systems

There exist sets A, B such that $B \subseteq A$.

Specification

Synthesis

General design principles for math diagramming systems

There exist sets A, B such that $B \subseteq A$.

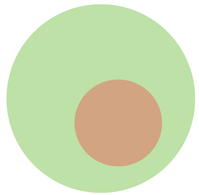
Specification

applying a formal mapping

logical object → visual object

Set → 

logical relationship → visual relationship

$\subseteq$ → 

Synthesis

General design principles for math diagramming systems

There exist sets A, B such that $B \subseteq A$.

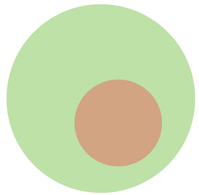
Specification

applying a formal mapping

logical object → visual object

Set → 

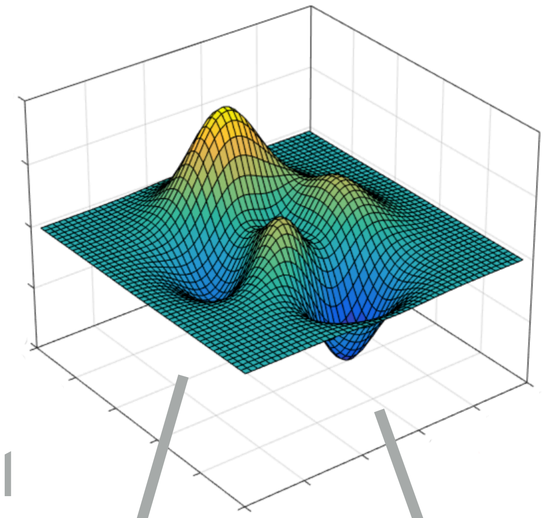
logical relationship → visual relationship

$\subseteq$ → 

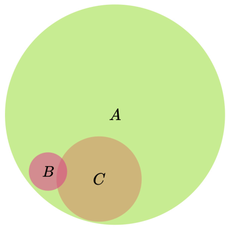
Synthesis

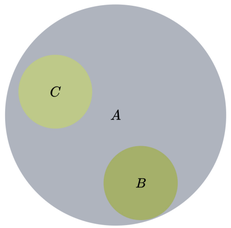
finding a satisfying diagram

objective: "how good is this diagram?"



potential solutions





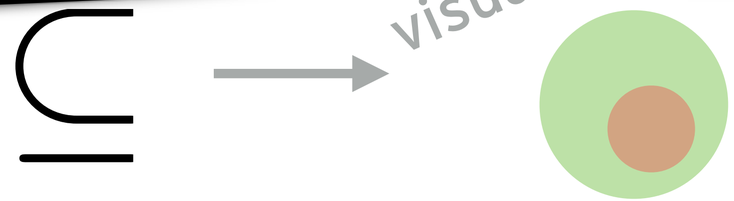
General design principles for math diagramming systems

There exist sets A, B such that $B \subseteq A$.

Specification

applying a formal mapping

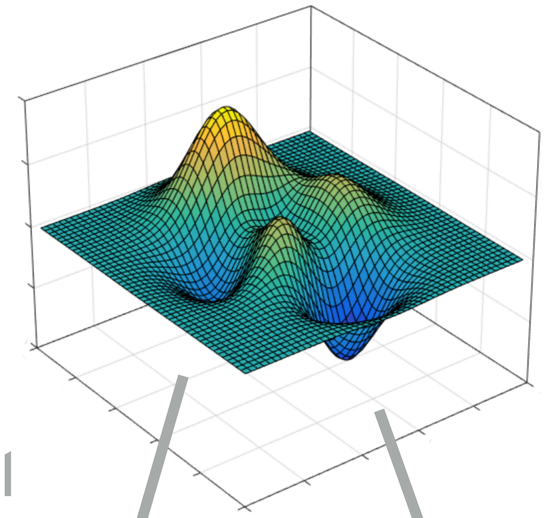
good tool for the job:
domain-specific languages
+ compiler



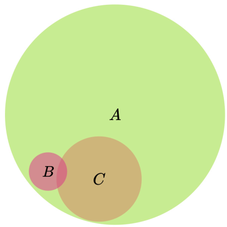
Synthesis

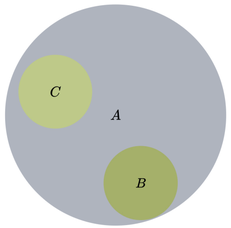
finding a satisfying diagram

objective: "how good is this diagram?"



potential solutions





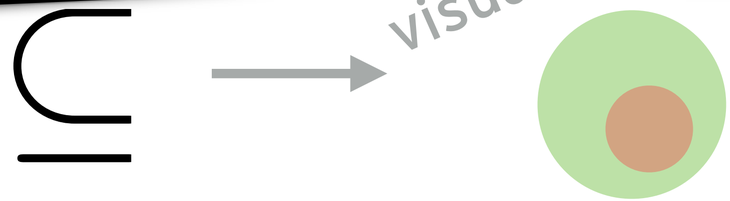
General design principles for math diagramming systems

There exist sets A, B such that $B \subseteq A$.

Specification

applying a formal mapping

good tool for the job:
domain-specific languages
+ compiler



Synthesis

finding a satisfying diagram

good tool for the job:
constrained optimization
+ numerical solver

potential solutions

Building these design principles into a system

Specification

Synthesis

Building these design principles into a system

Specification

Synthesis

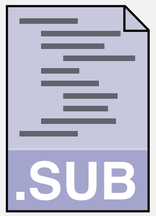
source code

Building these design principles into a system

Specification

Synthesis

source code



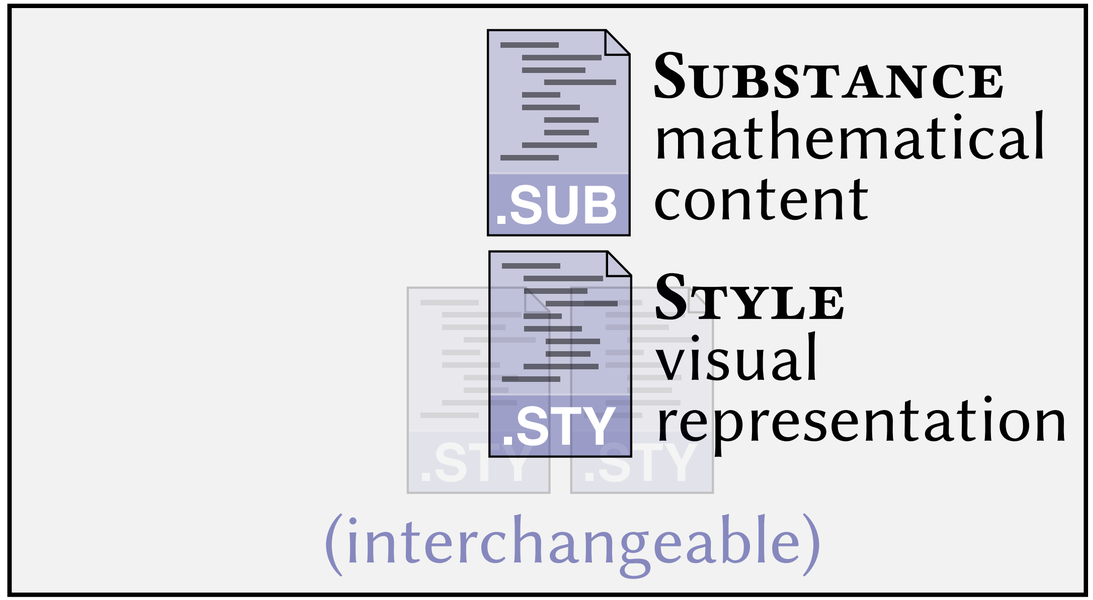
SUBSTANCE
mathematical
content

Building these design principles into a system

Specification

Synthesis

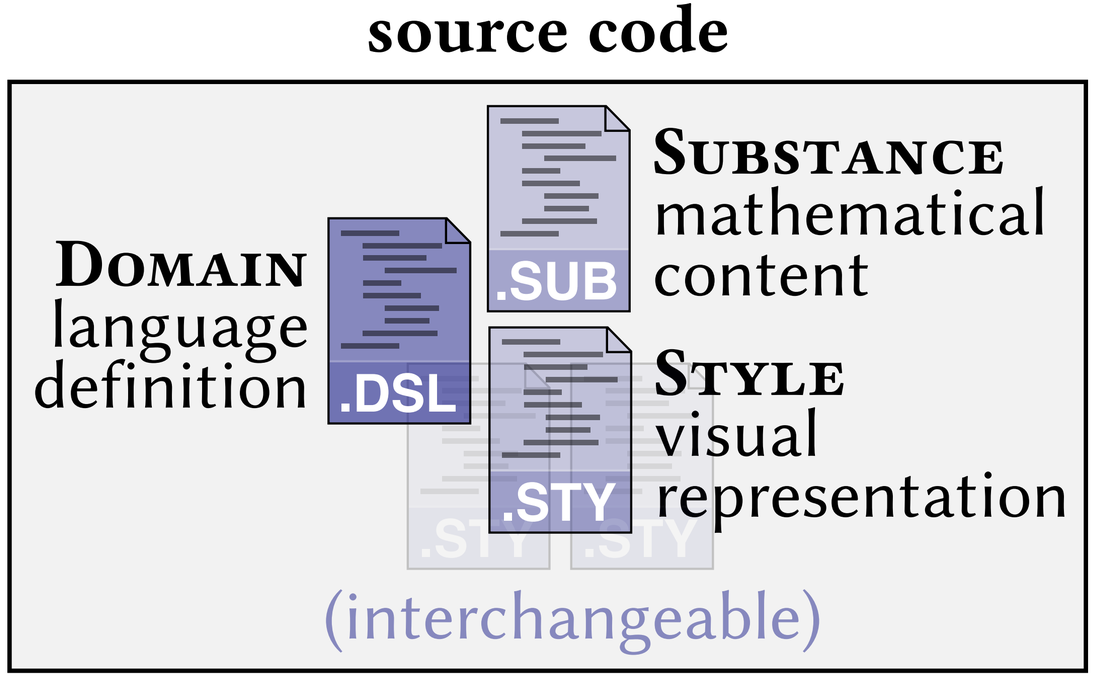
source code



Building these design principles into a system

Specification

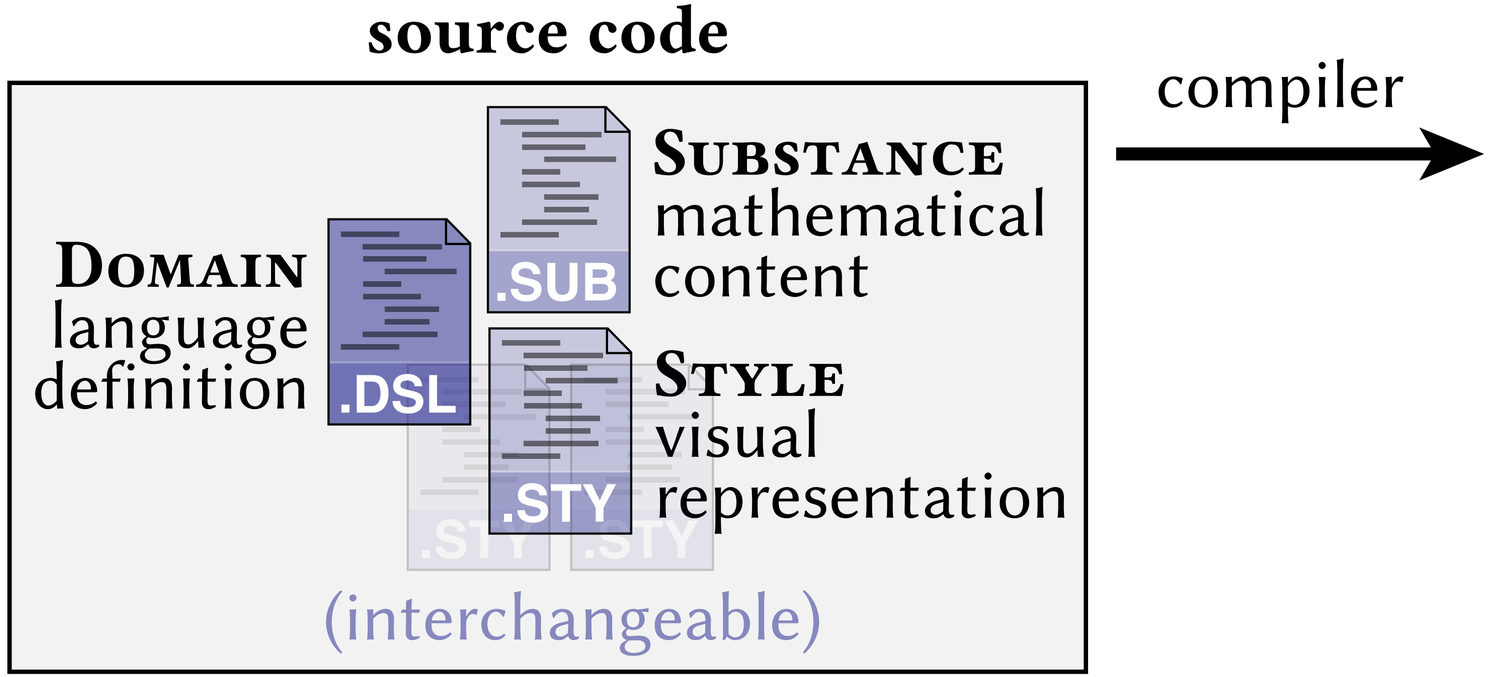
Synthesis



Building these design principles into a system

Specification

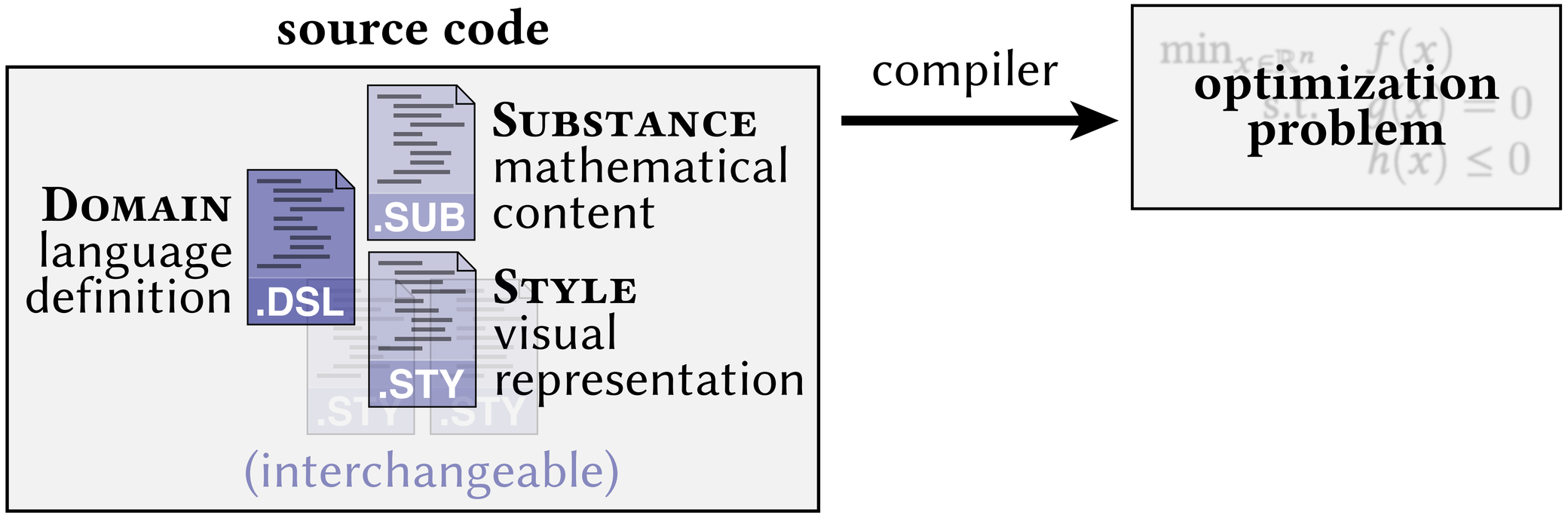
Synthesis



Building these design principles into a system

Specification

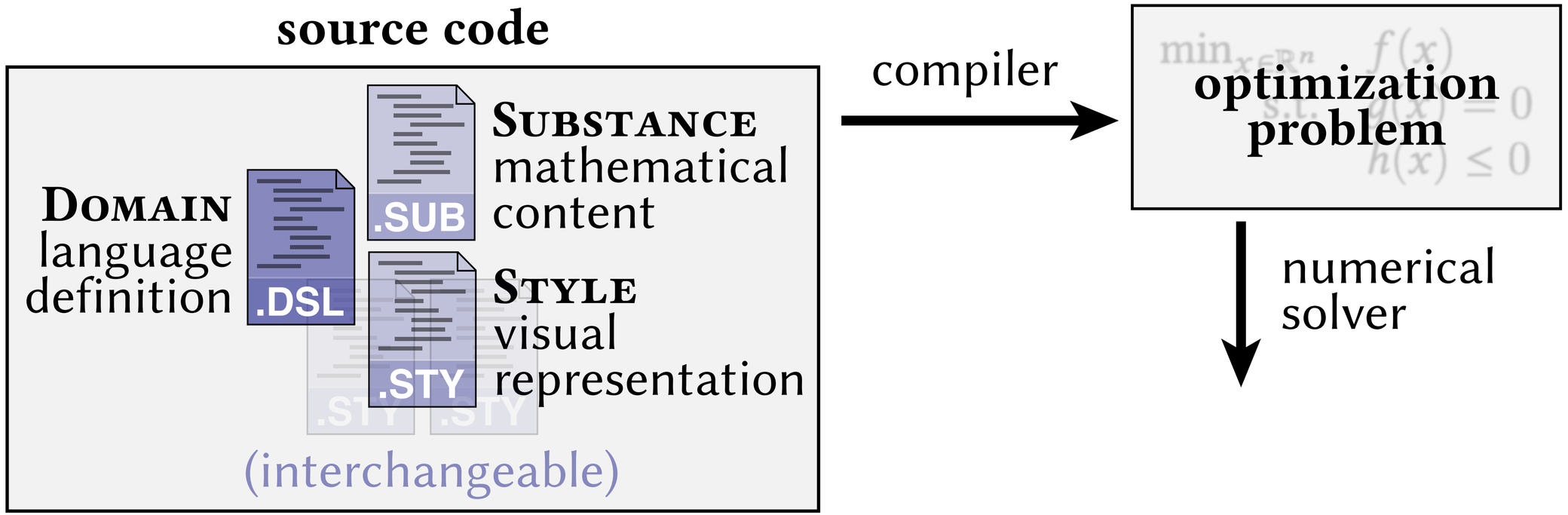
Synthesis



Building these design principles into a system

Specification

Synthesis



Building these design principles into a system

Specification

source code

Domain language definition

.DSL

.SUB

Substance mathematical content

.STY

Style visual representation

(interchangeable)

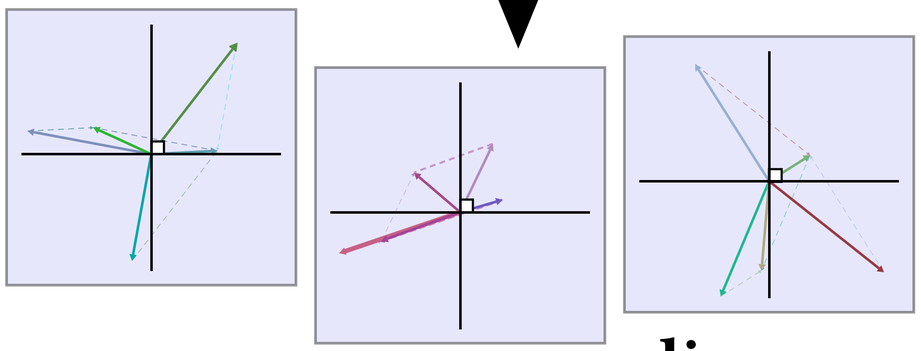
Synthesis

compiler

$\min_{x \in \mathbb{R}^n} f(x)$
$s.t.\ g(x) = 0$
$h(x) \leq 0$

optimization problem

numerical solver

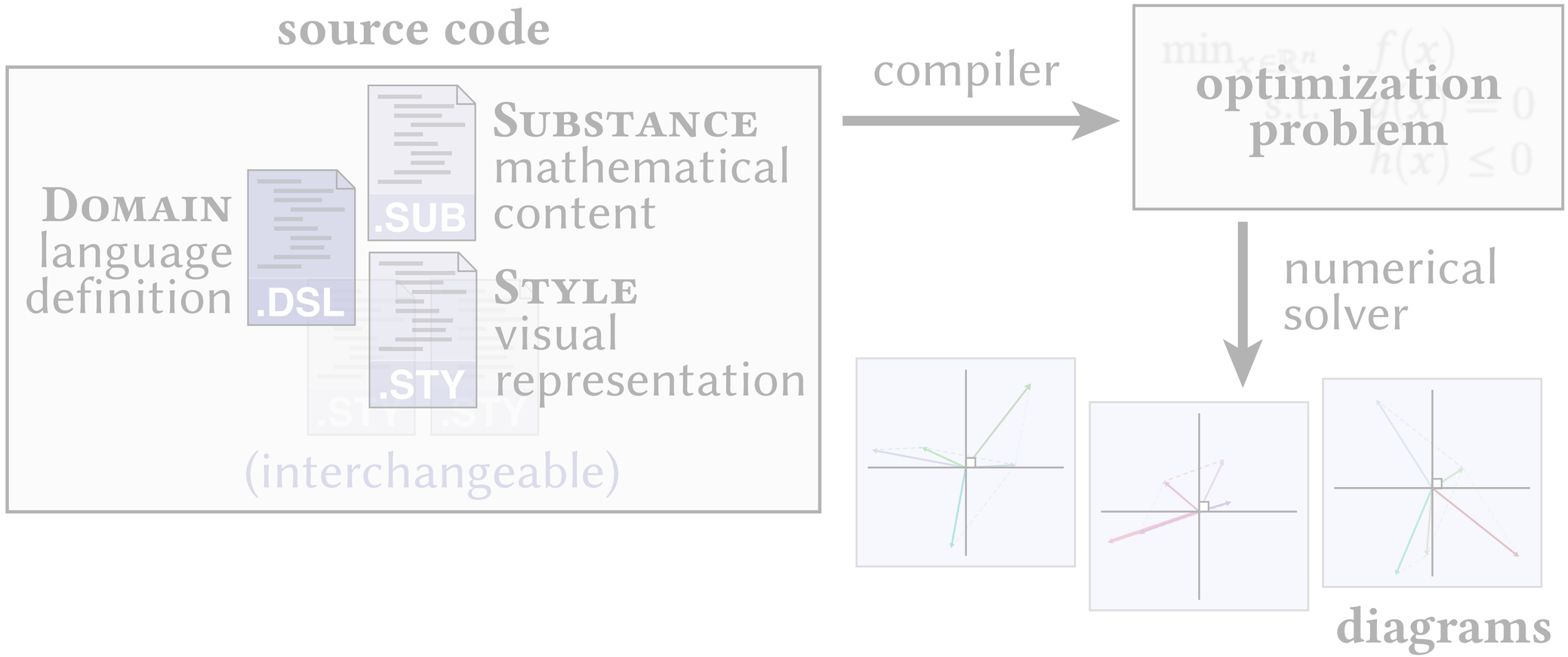


diagrams

Webpages follow similar design principles

Specification

Synthesis



Webpages follow similar design principles

Specification

Synthesis

source code HTML

Domain
language
definition

.DSL

.SUB

Substance
mathematical
content

.STY

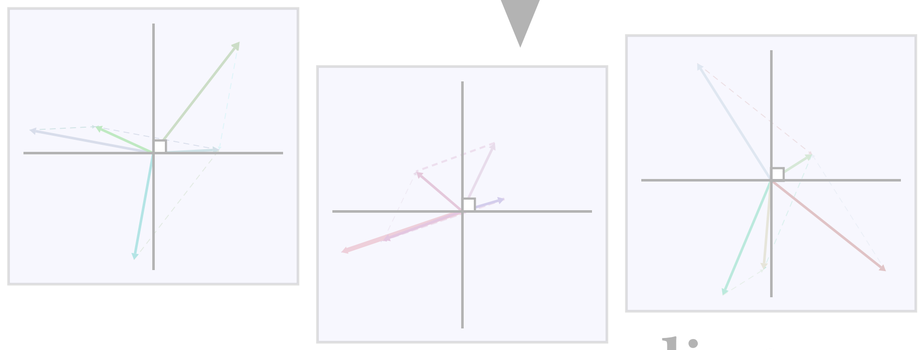
Style
visual
representation

(interchangeable)

compiler

optimization problem

numerical solver



diagrams

Webpages follow similar design principles

Specification

Synthesis

source code HTML

DOMAIN
language
definition

.DSL

.SUB

SUBSTANCE
mathematical
content

CSS

.STY

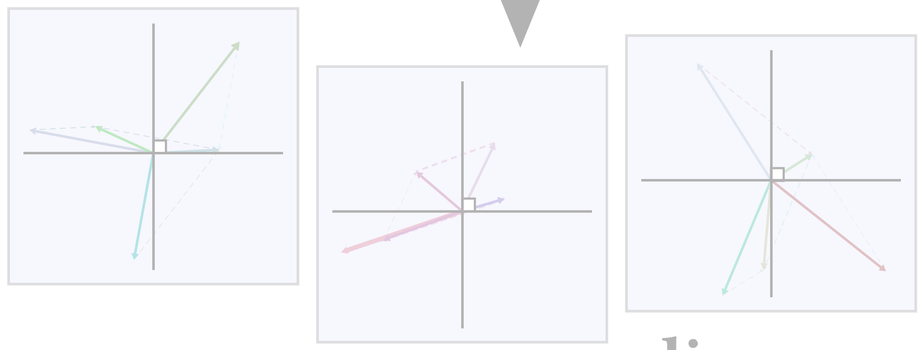
STYLE
visual
representation

(interchangeable)

compiler

optimization
problem

numerical
solver

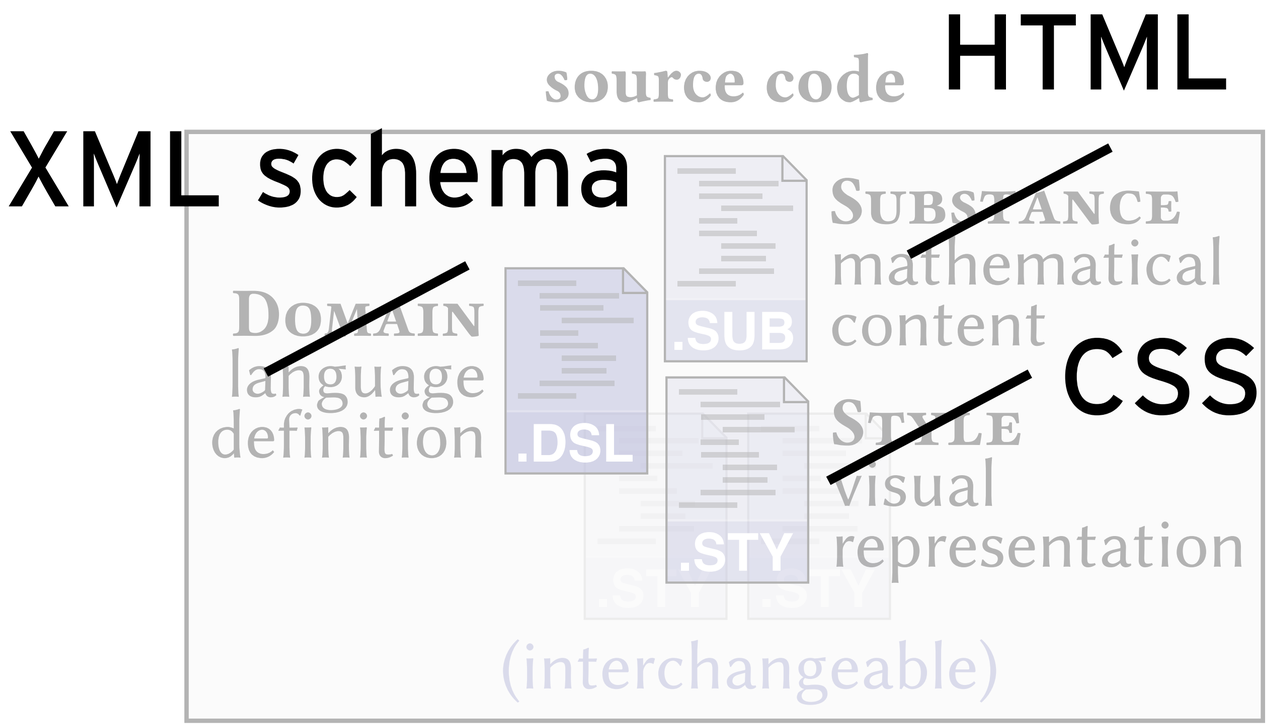


diagrams

Webpages follow similar design principles

Specification

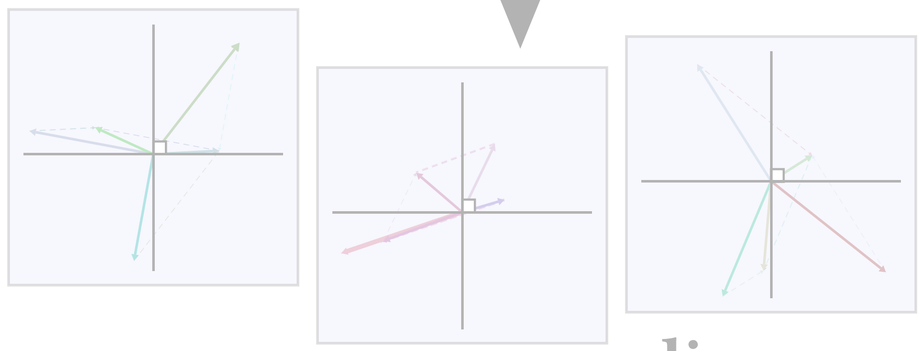
Synthesis



compiler

optimization problem

numerical solver



diagrams

Webpages follow similar design principles

Specification

Synthesis

source code

HTML

XML schema

CSS

SUBSTANCE
mathematical content

.SUB

DOMAIN
language definition

.DSL

STYLE
visual representation

.STY

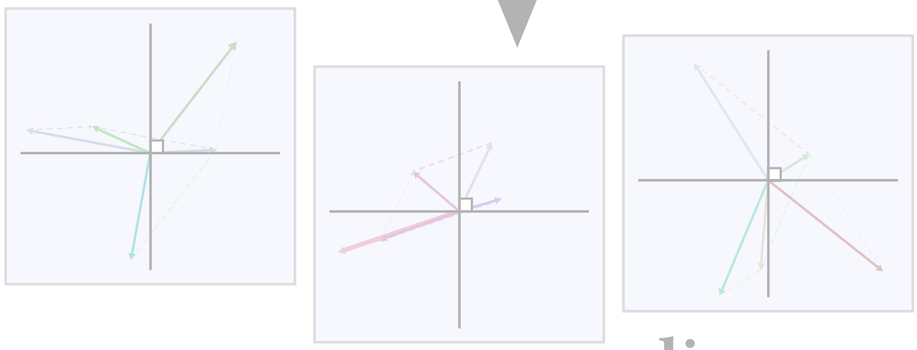
(interchangeable)

compiler

optimization problem

numerical solver

web browser engine



diagrams

Webpages follow similar design principles

Specification

Synthesis

source code

HTML

XML schema

CSS

SUBSTANCE
mathematical content

.SUB

DOMAIN
language definition

.DSL

STYLE
visual representation

.STY

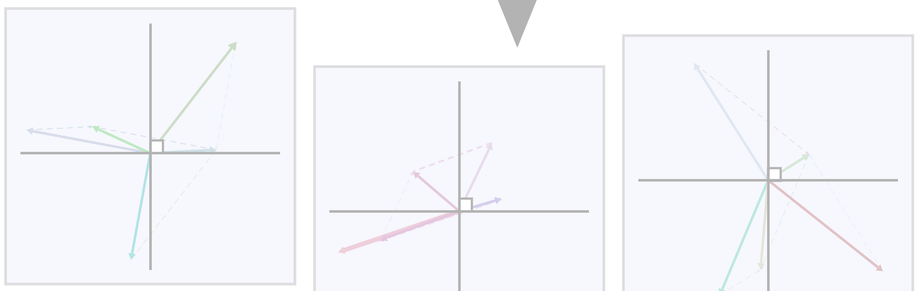
(interchangeable)

compiler

optimization problem

web browser engine

numerical solver



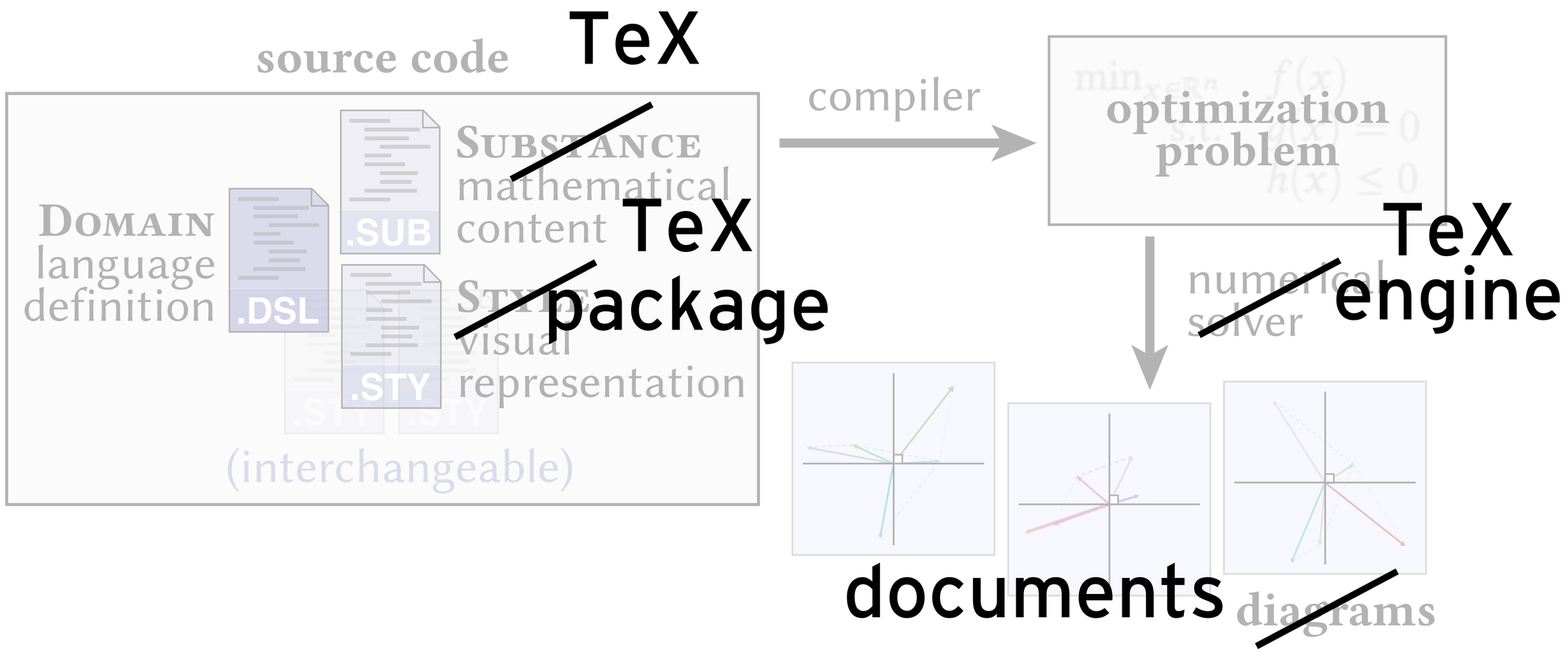
webpages

diagrams

TeX/LaTeX follows similar design principles

Specification

Synthesis



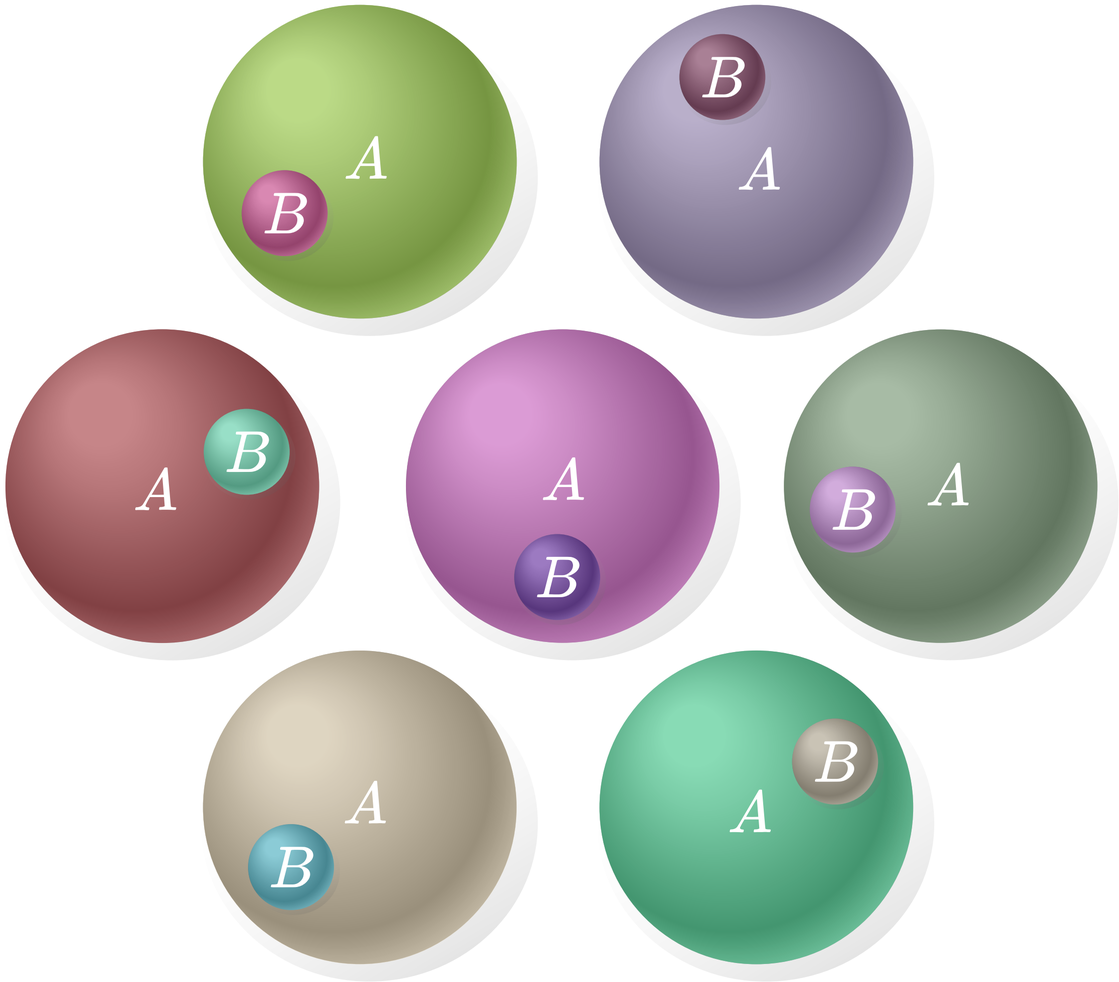
System walkthrough

A simple example

A simple example

Set A, B

B ⊂ A



Substance defines the content of a diagram

There exist sets A *and* B
such that B *is a subset of* A.

Substance defines the content of a diagram

specify objects

There exist sets A *and* B
such that B *is a subset of* A.

Substance defines the content of a diagram

specify objects

There exist sets A *and* B
such that B *is a subset of* A.

specify a relationship

Substance defines the content of a diagram

There exist sets A *and* B
such that B *is a subset of* A.

specify a relationship

Substance defines the content of a diagram

There exist sets A *and* B
such that B *is a subset of* A.

Substance defines the content of a diagram

There exist sets A *and* B
such that B *is a subset of* A.

Substance defines the content of a diagram

There exist sets A *and* B
such that B *is a subset of* A.

Substance defines the content of a diagram

There exist sets A *and* B
such that B *is a subset of* A.

```
Set A, B
B ⊂ A
```

Substance defines the content of a diagram

There exist sets A *and* B
such that B *is a subset of* A.

specify objects

```
Set A, B
B ⊂ A
```

Substance defines the content of a diagram

There exist sets A *and* B
such that B *is a subset of* A.

```
Set A, B
B ⊂ A
```

specify objects

specify a relationship

Substance is compositional

"Consider sets *A*, *B*, and *C* such that *B* is a subset of *A*, *C* is a subset of *A*, and *B* and *C* don't intersect, and …"

Substance is compositional

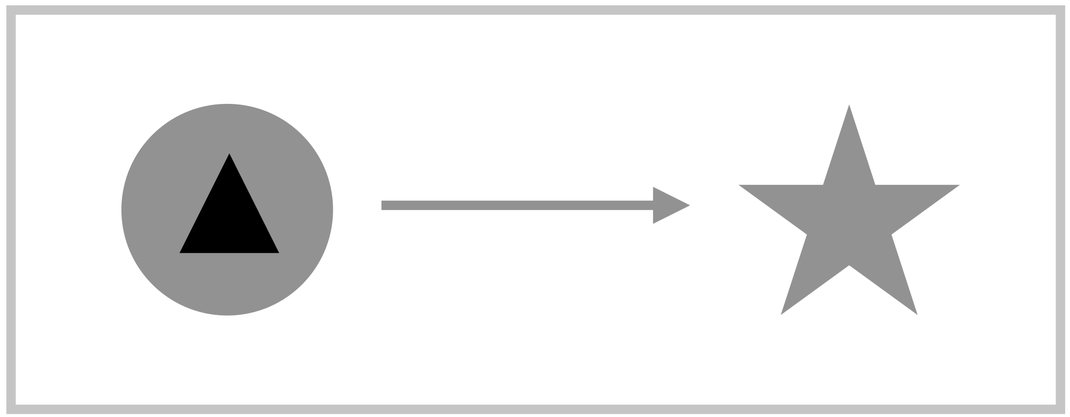
"Consider sets *A*, *B*, and *C* such that *B* is a subset of *A*, *C* is a subset of *A*, and *B* and *C* don't intersect, and …"

```
Set A, B, C
B ⊂ A
C ⊂ A
B ∩ C = ∅
…
```

Style defines a domain's visual representation

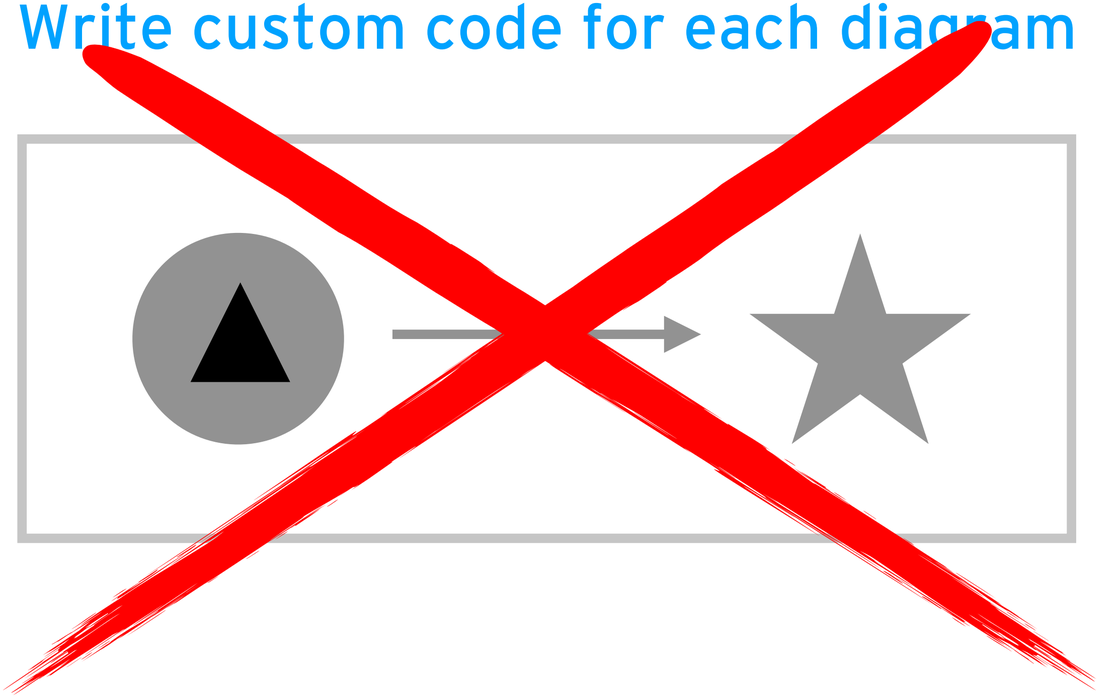
Style defines a domain's visual representation

Write custom code for each diagram

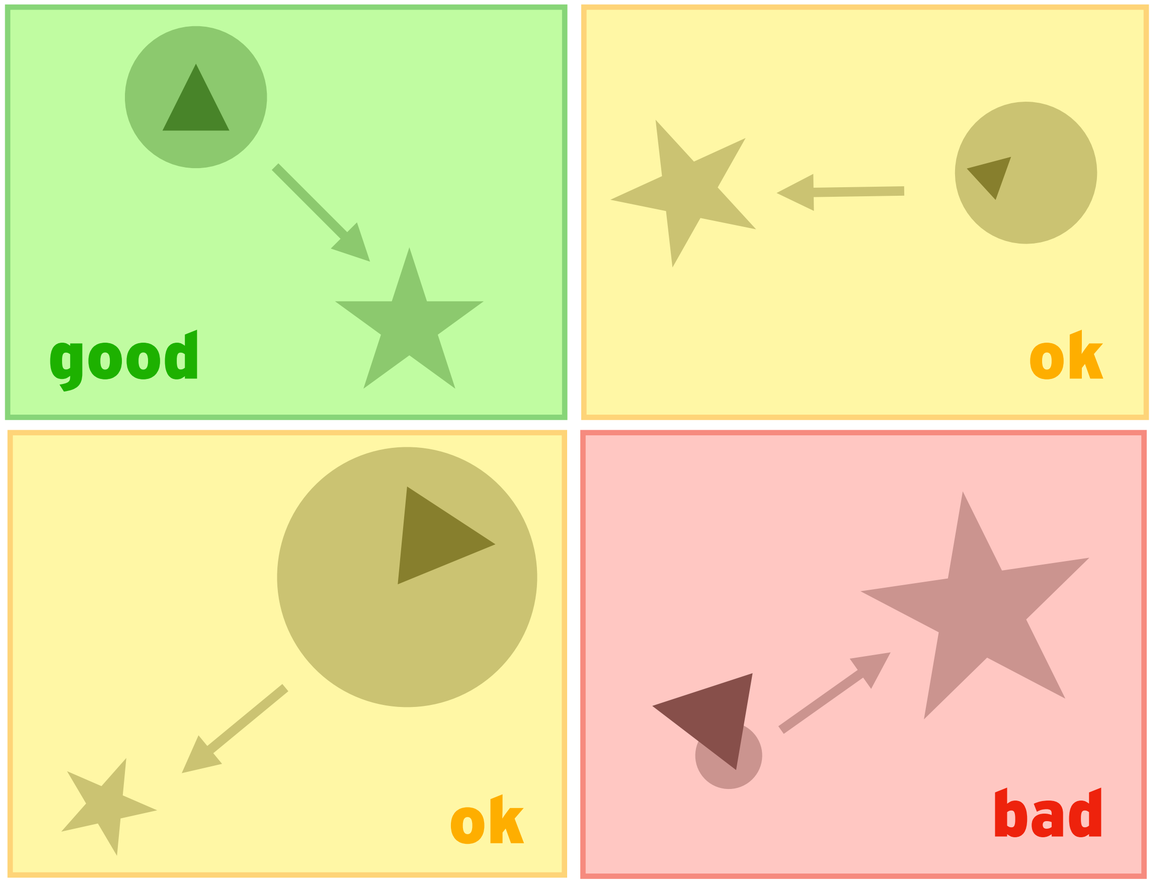


Style defines a domain's visual representation

Write custom code for each diagram



Write code **once** for many diagrams



A *Style* is a list of selectors & rules

A *Style* is a list of selectors & rules

Selector

Rule (e.g. make a shape)

Rule (e.g. make a constraint)

A *Style* is a list of selectors & rules

Selector

Rule (e.g. make a shape)

Rule (e.g. make a constraint)

More specific selector

Rule refinement

Rule refinement

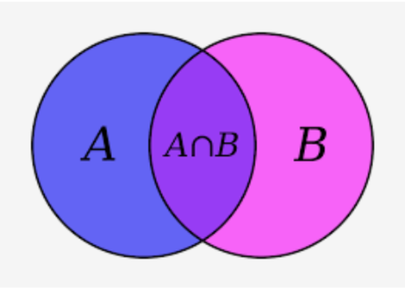
Visualizing sets & subsets

Visualizing sets & subsets

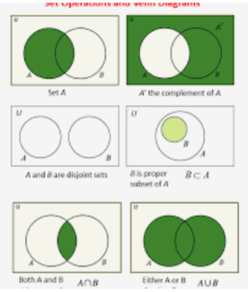
Google set theory

All Videos Images Books News More Settings Tools Collections SafeSearch

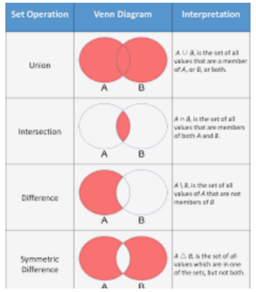
venn diagram | math | symbol | complement | union | intersection | probability | notation | subset



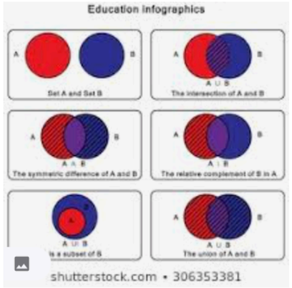
Set theory - Wikipedia
en.wikipedia.org



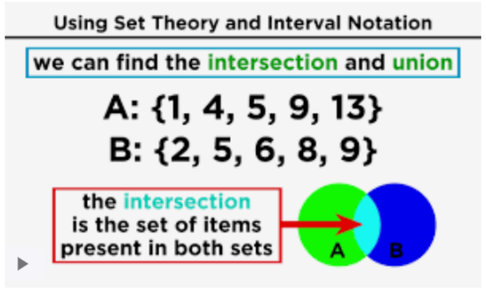
Understanding Set Theor...
medium.com



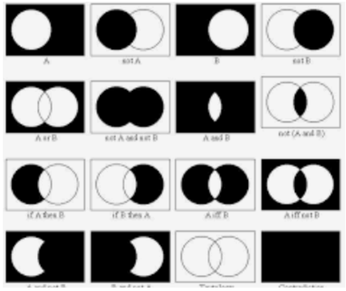
Tutorial) Python Sets vs ...
in.pinterest.com



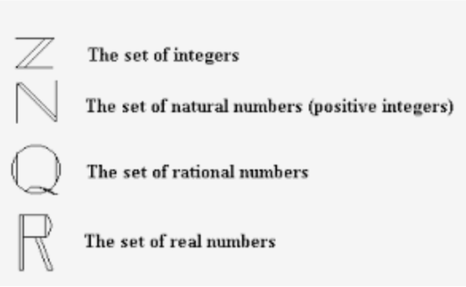
Set Theory Images, Stock Ph...
shutterstock.com



Set Theory: Types of Sets, Unions and ...
youtube.com

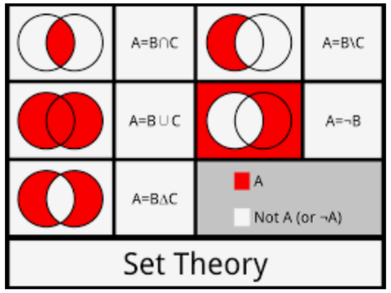


Set Theory
paperpandao1o.blogspot.com

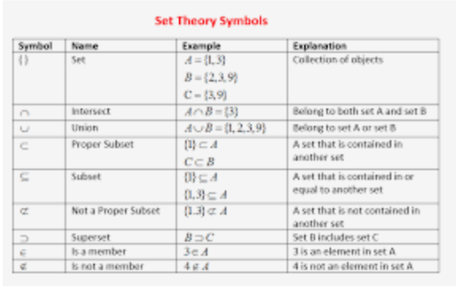


Set Theory - Mathematics A-Level Revision
revisionmaths.com

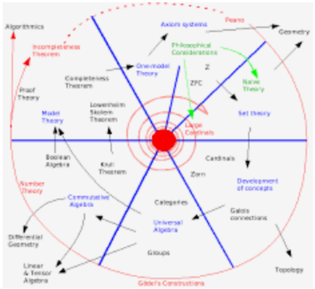
IPM using basic set theory ...
researchgate.net



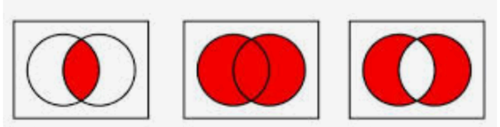
File:Set Theory Operations.svg ...
commons.wikimedia.org



Describing and Defining Sets (examples ...
onlinemathlearning.com



Set Theory and Foundations of ...
settheory.net



A simple *Style* for sets & subsets

A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```

A simple *Style* for sets & subsets

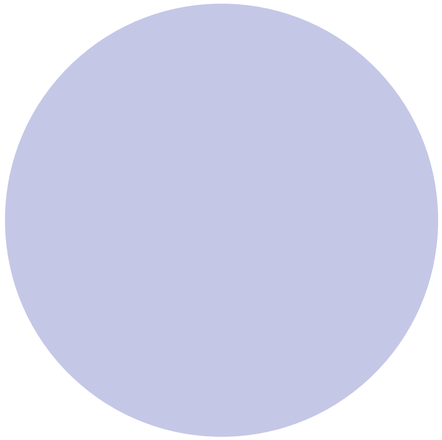
```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```

A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```

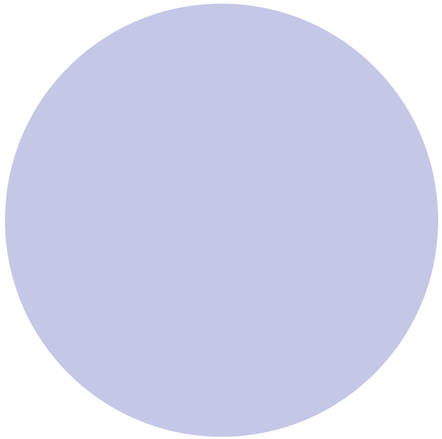
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```



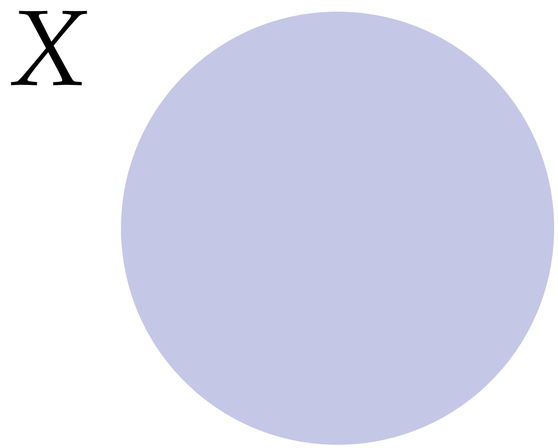
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```



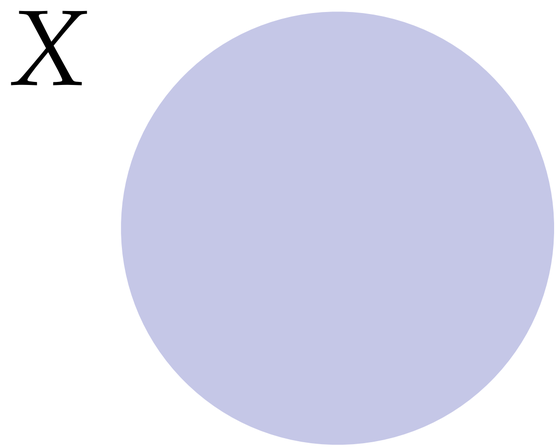
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```



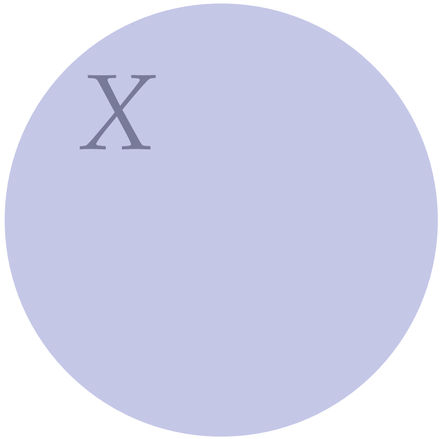
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```



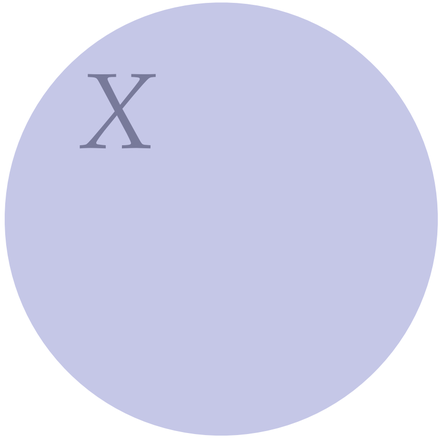
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```



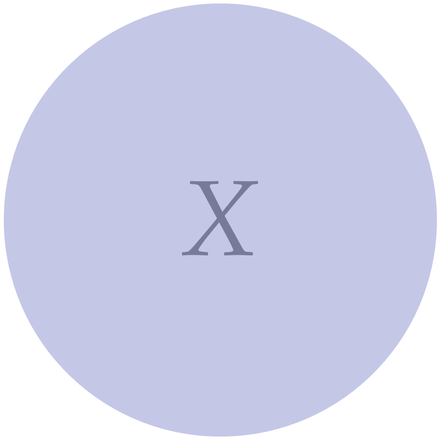
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```



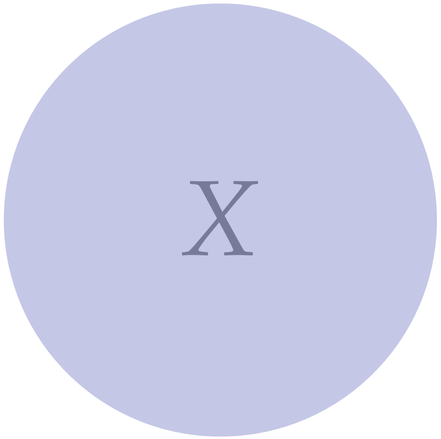
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```



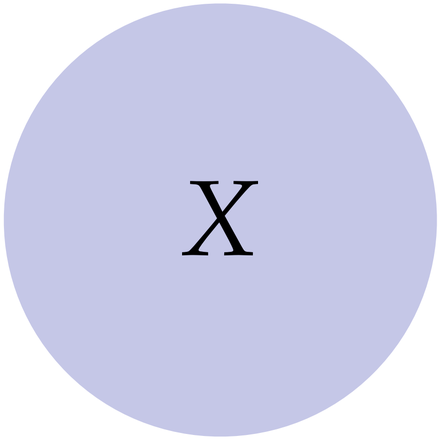
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```



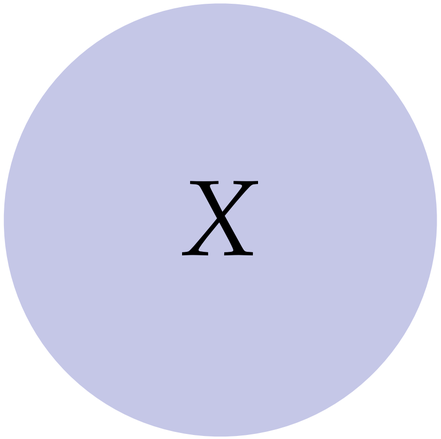
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```



A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}
```

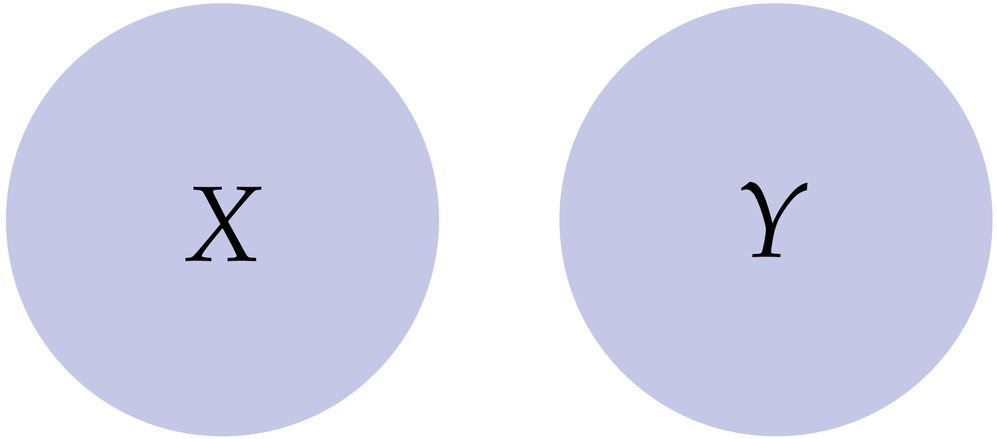


A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }
```

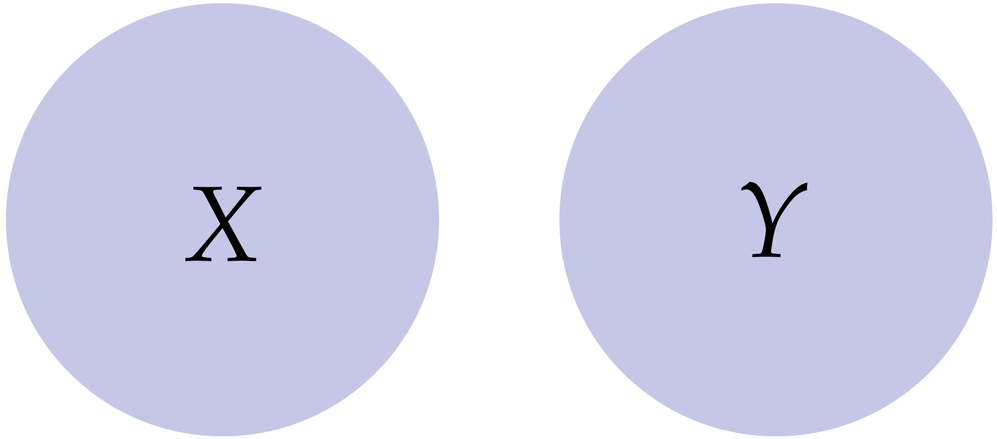
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }
```



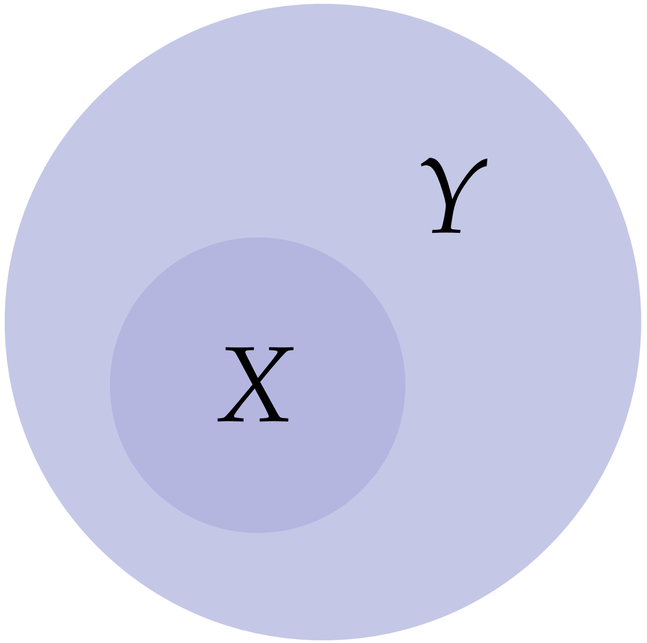
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }
```



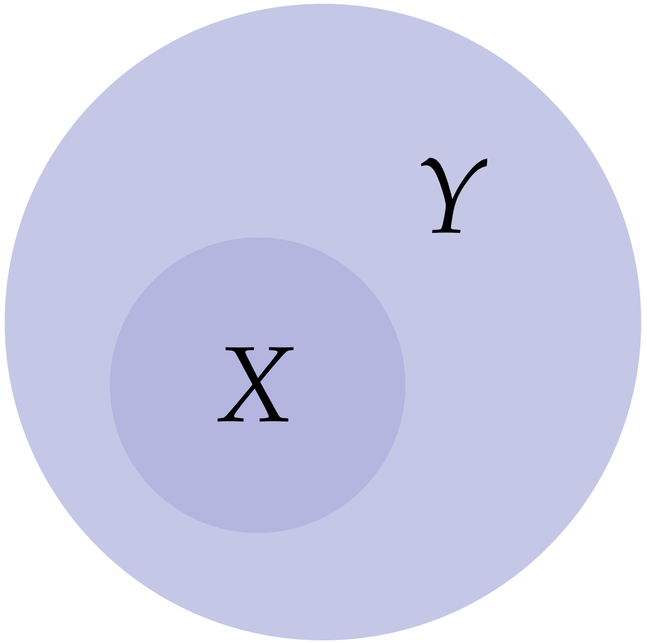
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }
```



A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }
```

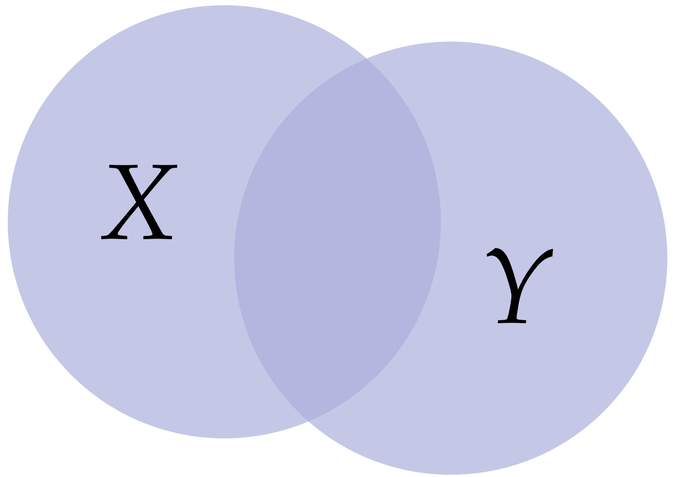


A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }

 forall Set x; Set y
 where Not(Intersecting(x, y)) {
     ensure disjoint(x.shape, y.shape)
 }
```

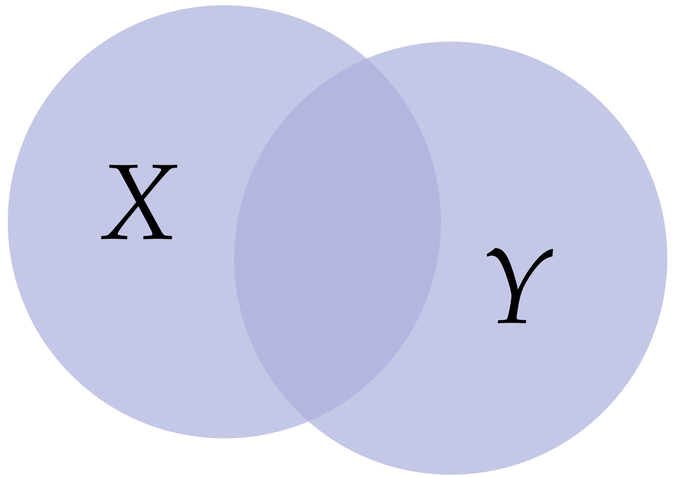
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }

 forall Set x; Set y
 where Not(Intersecting(x, y)) {
     ensure disjoint(x.shape, y.shape)
 }
```



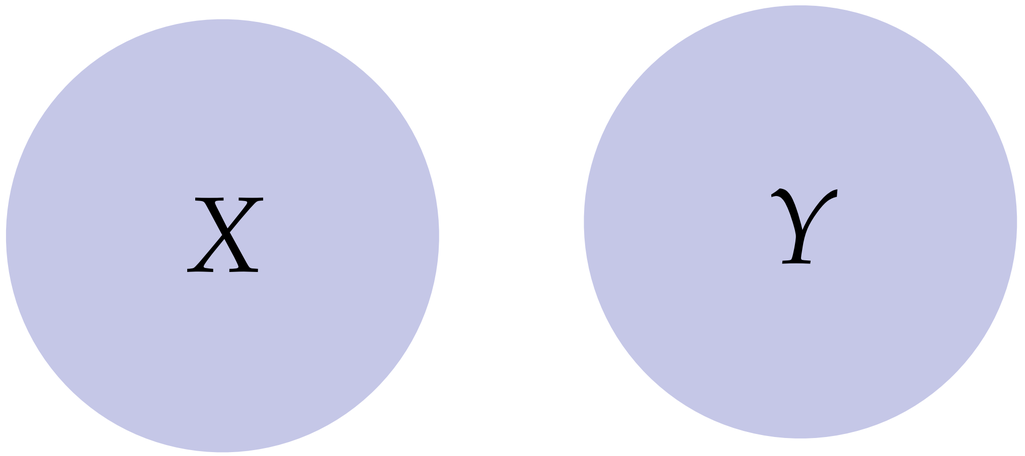
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }

 forall Set x; Set y
 where Not(Intersecting(x, y)) {
     ensure disjoint(x.shape, y.shape)
 }
```



A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }

 forall Set x; Set y
 where Not(Intersecting(x, y)) {
     ensure disjoint(x.shape, y.shape)
 }
```



A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }

 forall Set x; Set y
 where Not(Intersecting(x, y)) {
     ensure disjoint(x.shape, y.shape)
 }
```

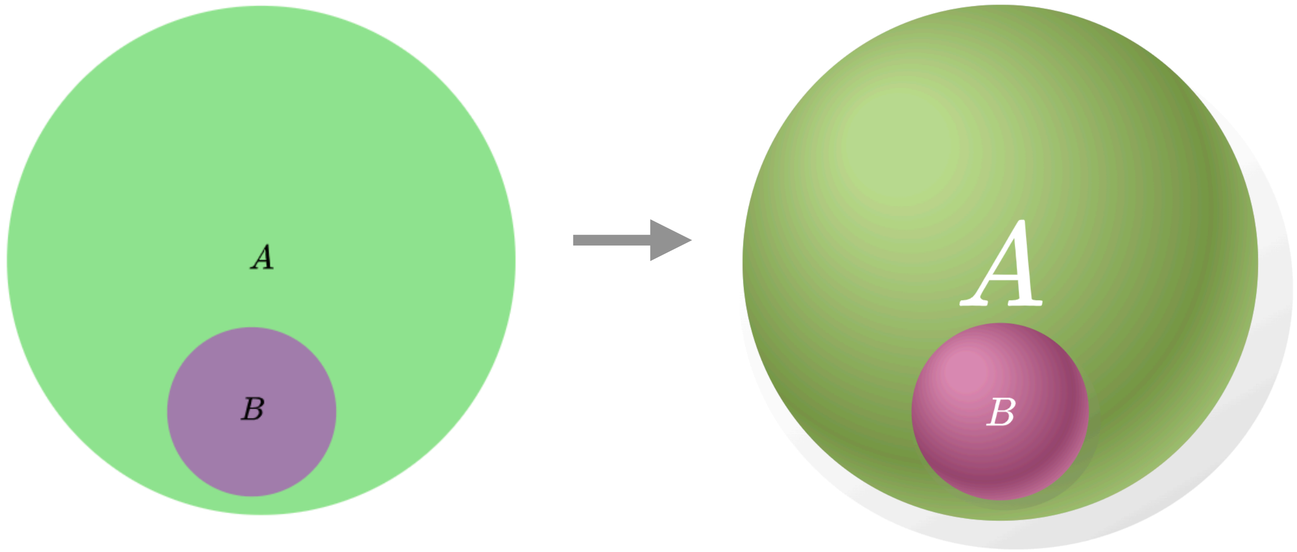
A simple *Style* for sets & subsets

```
forall Set x {
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

 forall Set x; Set y
 where IsSubset(x, y) {
     ensure contains(y.shape, x.shape)
     ensure smallerThan(x.shape, y.shape)
     ensure outsideOf(y.text, x.shape)
     layer x.shape above y.shape
     layer y.text below x.shape
 }

 forall Set x; Set y
 where Not(Intersecting(x, y)) {
     ensure disjoint(x.shape, y.shape)
 }
```

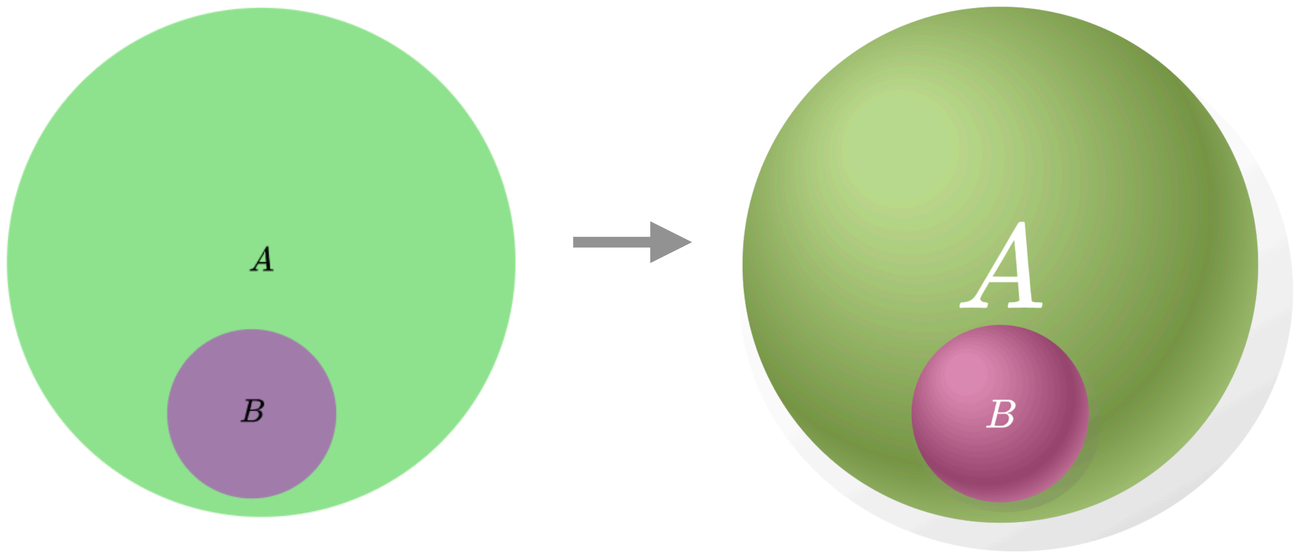
defines a family of diagrams

Style makes it easy to refine the visual style

Style makes it easy to refine the visual style



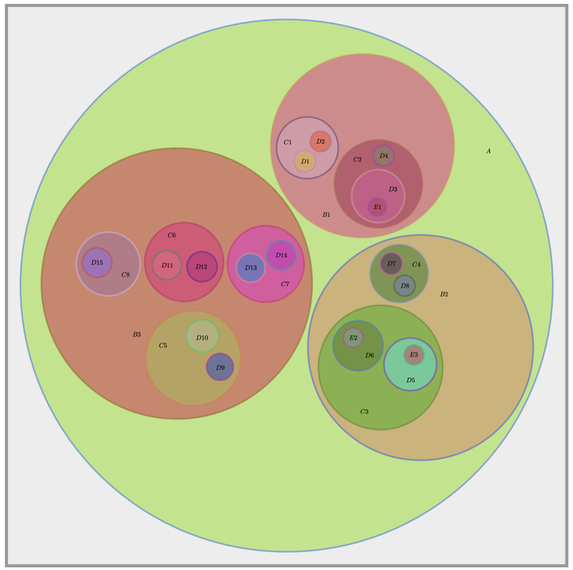
Style makes it easy to refine the visual style



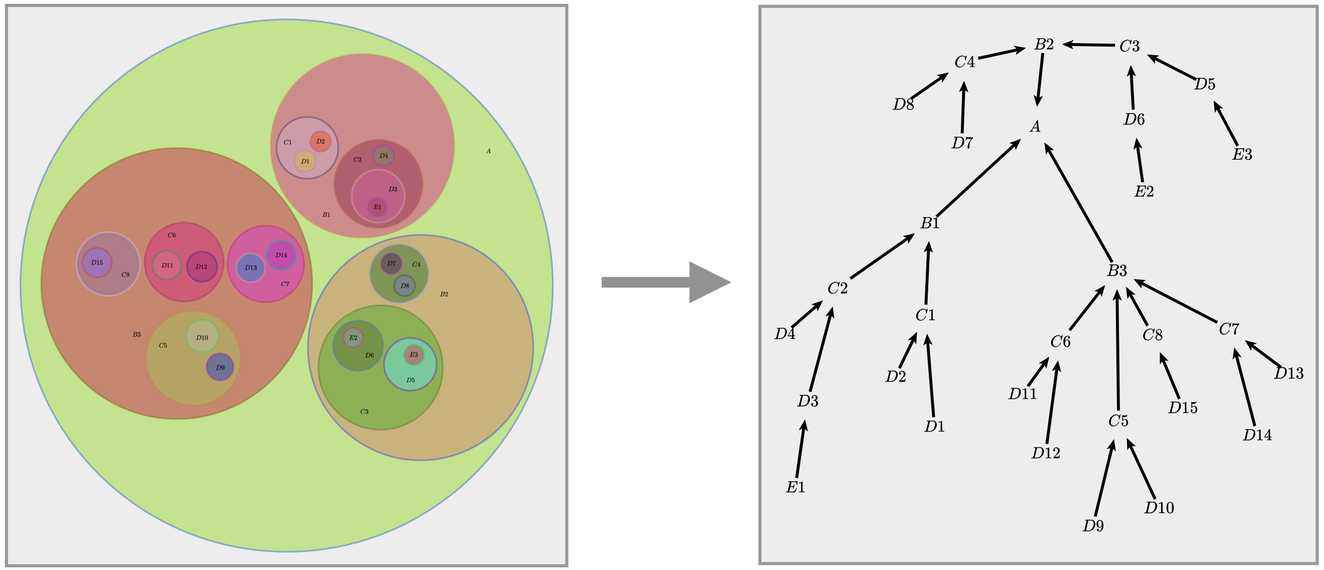

```
x.shading = Image {
    x : x.shape.x
    y : x.shape.y
    w : x.shape.r * 2.0
    h : x.shape.r * 2.0
    path : "shading.svg"
}

x.shadow = Image {
    path : "shadow.svg"
    w : x.shape.r * 2.15
    h : x.shape.r * 2.22
    x : x.shape.x + 0.03 * x.shading.w
    y : x.shape.y - 0.051 * x.shading.h
}

x.text = Text {
    string : x.label
    color: rgba(1.0, 1.0, 1.0, 1.0)
    w: 0.5 * x.shape.r
    h: 0.5 * x.shape.r
}
```

Style makes it easy to change the visual representation

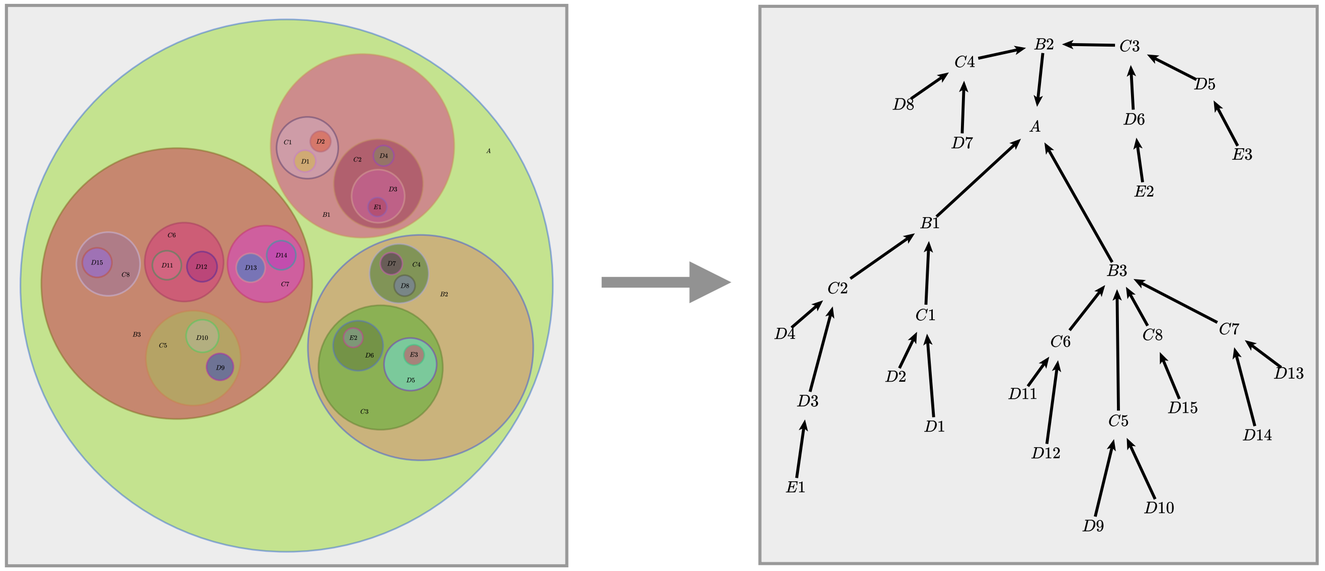
Style makes it easy to change the visual representation



Style makes it easy to change the visual representation



Style makes it easy to change the visual representation



```
Set x {
    x.shape = Text { string : x.label }
}

Set x; Set y {
   encourage repel(x.shape, y.shape, 5.0)
}

Set x; Set y
where IsSubset(x, y) {
    arrow = Arrow {
      thickness : 2.0
      color : rgba(0.0, 0.0, 0.0, 1.0)
    }

    encourage centerArrow(LOCAL.arrow,
                  x.shape, y.shape)
    encourage above(y.shape, x.shape)
    encourage equal(x.shape.x, y.shape.x)
}
```

Domain schemas define the mathematical environment

Domain schemas define the mathematical environment

- There's no way to anticipate every object/ idea a user might want to diagram...

Domain schemas define the mathematical environment

- There's no way to anticipate every object/ idea a user might want to diagram…

image credit: Martin Kuppe

Domain schemas define the mathematical environment

- There's no way to anticipate every object/ idea a user might want to diagram…

- But users can **extend** the Substance language by writing Domain schemas that define their own domain-specific dialects

image credit: Martin Kuppe

Domain schemas define abstract types & notation

Domain schemas define abstract types & notation


```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```

Domain schemas define abstract types & notation


```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```

Domain schemas define abstract types & notation


```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```


```
type Set
```

Domain schemas define abstract types & notation


```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```


```
type Set
predicate IsSubset : Set s1 * Set s2
```

Domain schemas define abstract types & notation


```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```


```
type Set
predicate IsSubset : Set s1 * Set s2
predicate Intersecting : Set s1 * Set s2
```

Domain schemas define abstract types & notation

.SUB

```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```

.DSL

```
type Set
predicate IsSubset : Set s1 * Set s2
predicate Intersecting : Set s1 * Set s2
predicate Not : Prop p
```

Domain schemas define abstract types & notation

.SUB

```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```

.DSL

```
type Set
predicate IsSubset : Set s1 * Set s2
predicate Intersecting : Set s1 * Set s2
predicate Not : Prop p
notation "A ⊂ B" ~ "IsSubset(A, B)"
```

Domain schemas define abstract types & notation

.SUB

```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```

.DSL

```
type Set
predicate IsSubset : Set s1 * Set s2
predicate Intersecting : Set s1 * Set s2
predicate Not : Prop p
notation "A ⊂ B" ~ "IsSubset(A, B)"
notation "A ∩ B = ∅" ~ "Not(Intersecting(A, B))"
```

Domain schemas define abstract types & notation

.SUB

```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```

purely abstract

.DSL

```
type Set
predicate IsSubset : Set s1 * Set s2
predicate Intersecting : Set s1 * Set s2
predicate Not : Prop p
notation "A ⊂ B" ~ "IsSubset(A, B)"
notation "A ∩ B = ∅" ~ "Not(Intersecting(A, B))"
```

Domain schemas define abstract types & notation

.SUB

```
Set A, B, C, D,
    E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```

.DSL

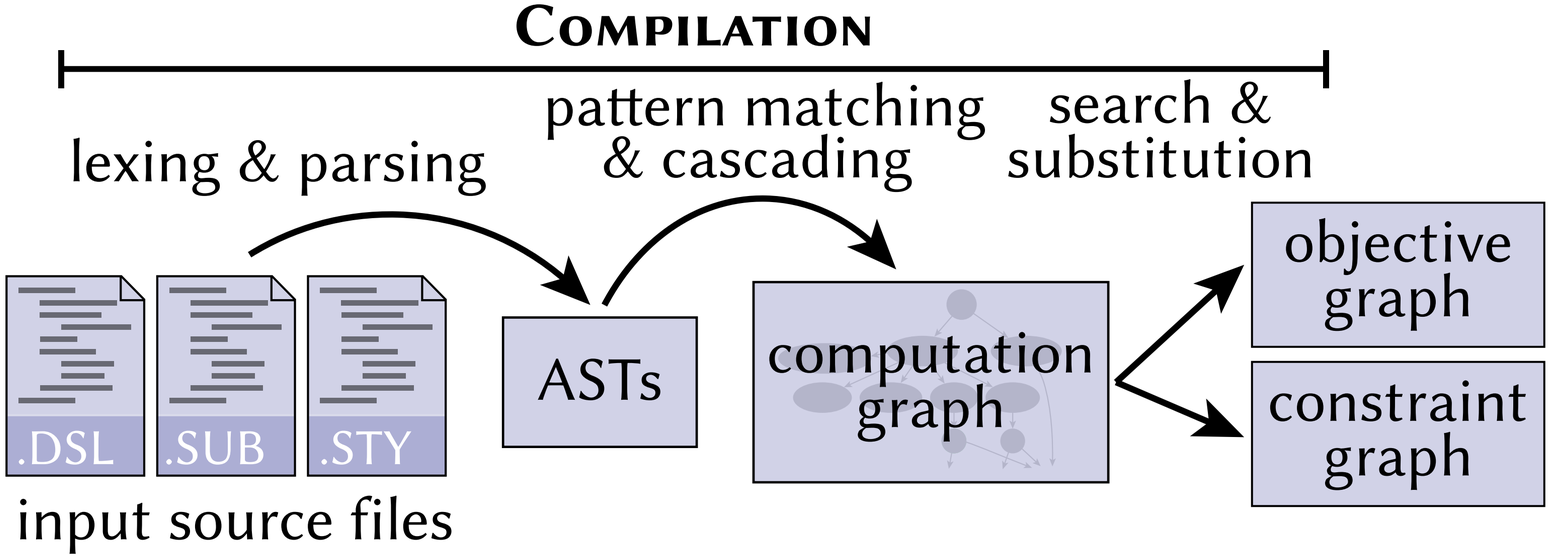
```
type Set
predicate IsSubset : Set s1 * Set s2
predicate Intersecting : Set s1 * Set s2
predicate Not : Prop p
notation "A ⊂ B" ~ "IsSubset(A, B)"
notation "A ∩ B = ∅" ~ "Not(Intersecting(A, B))"
```

purely abstract

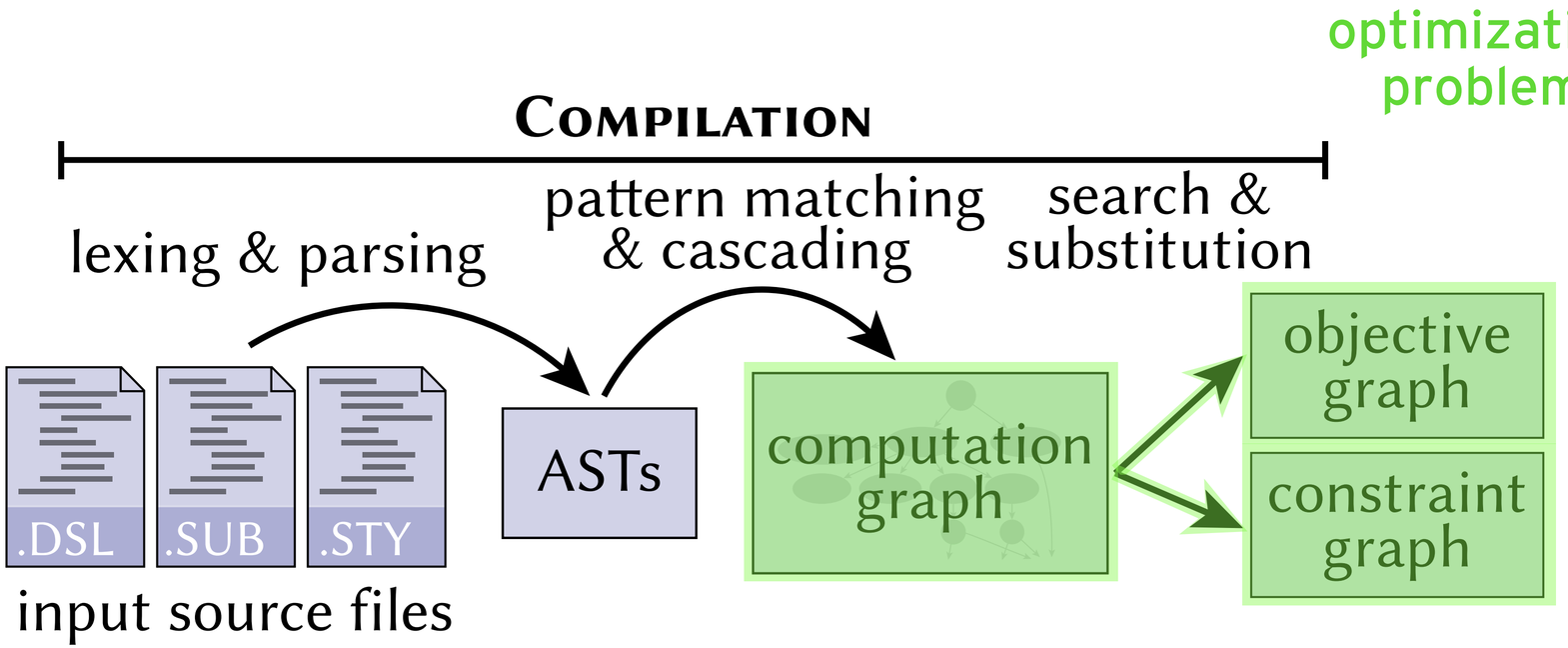
custom notation

The Penrose pipeline links specification and synthesis

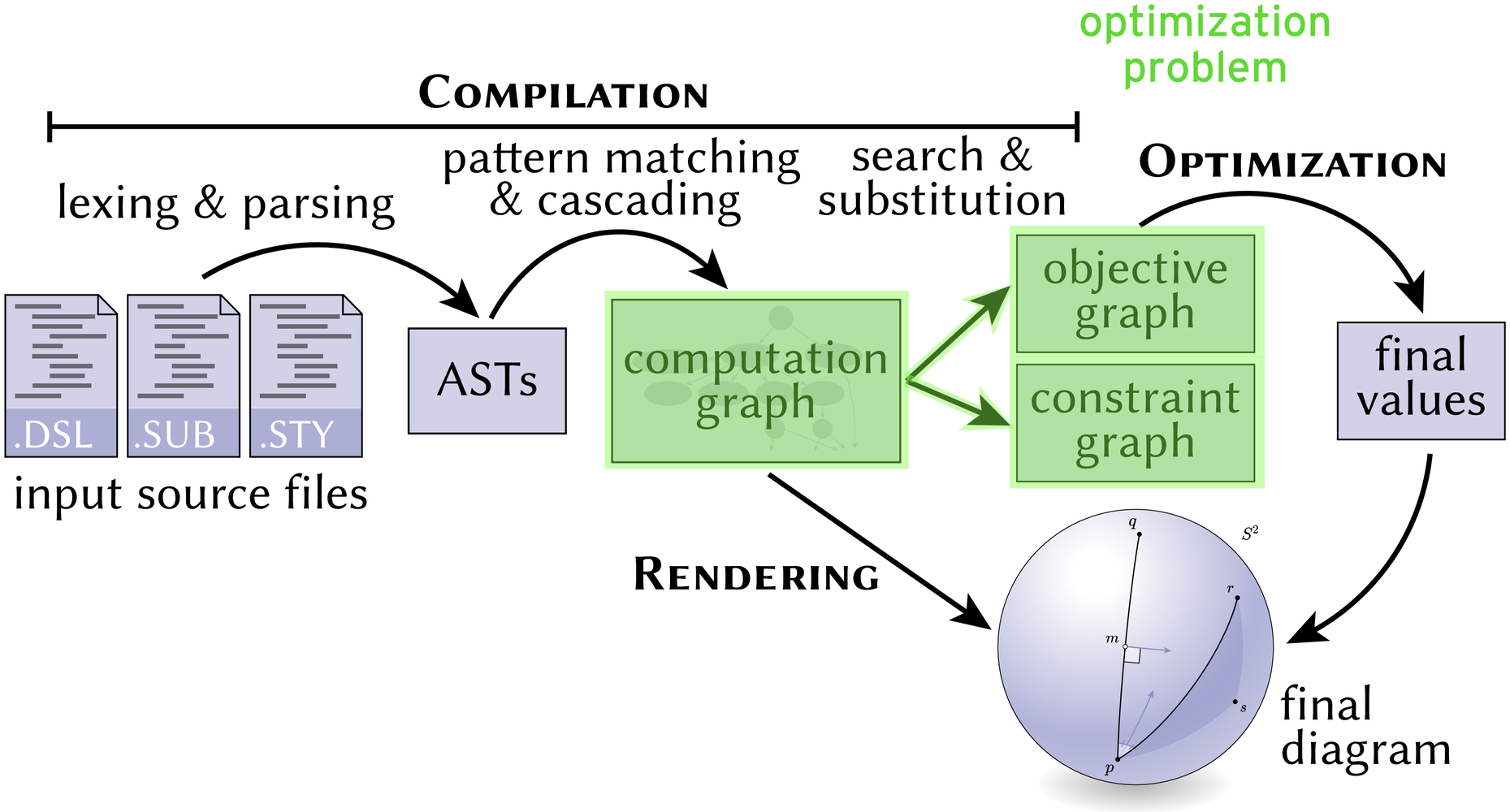
The Penrose pipeline links specification and synthesis



The Penrose pipeline links specification and synthesis



The Penrose pipeline links specification and synthesis



Building the computation graph

Building the computation graph

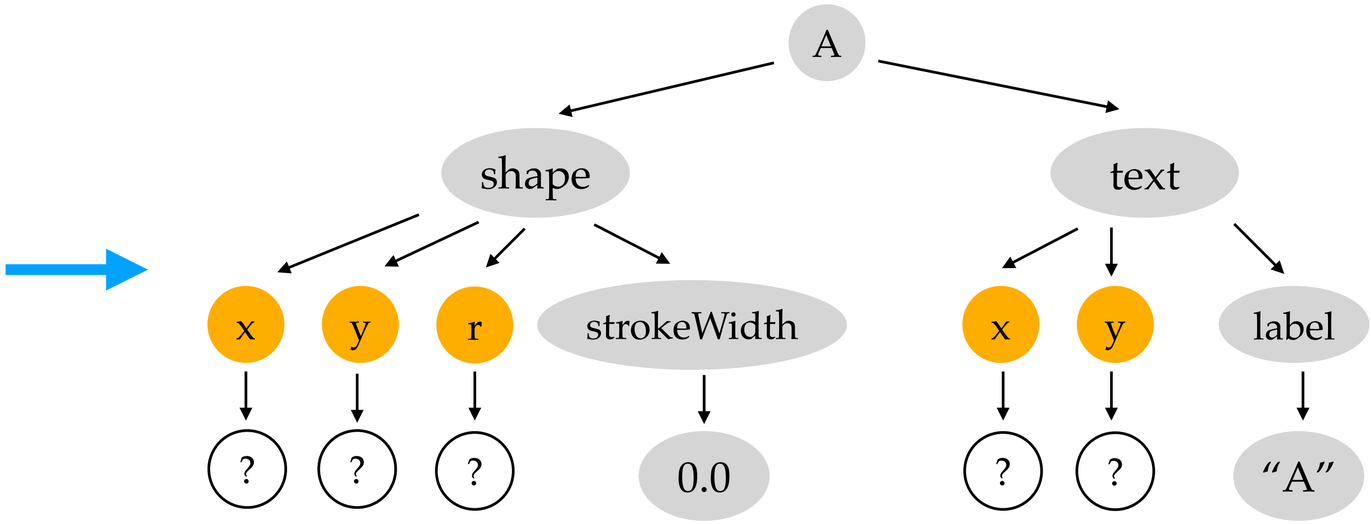
```
Set A, B
B ⊂ A
```

```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```

Building the computation graph

```
Set A, B
B ⊂ A
```

```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```

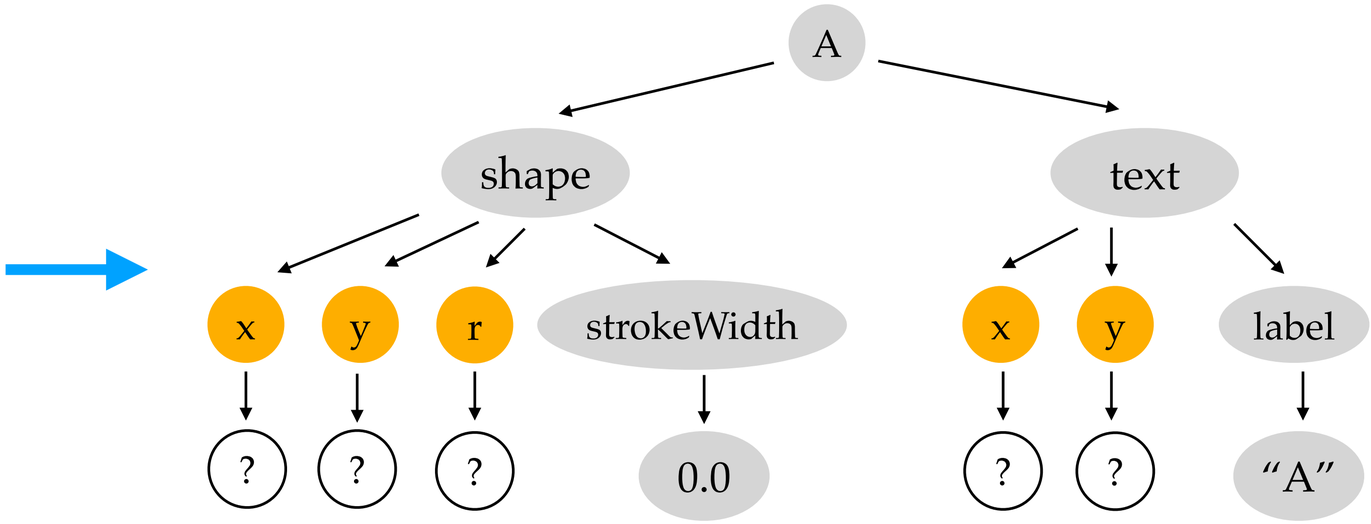


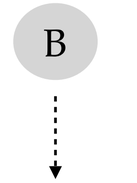
Building the computation graph

Set A, B

B ⊂ A

```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```



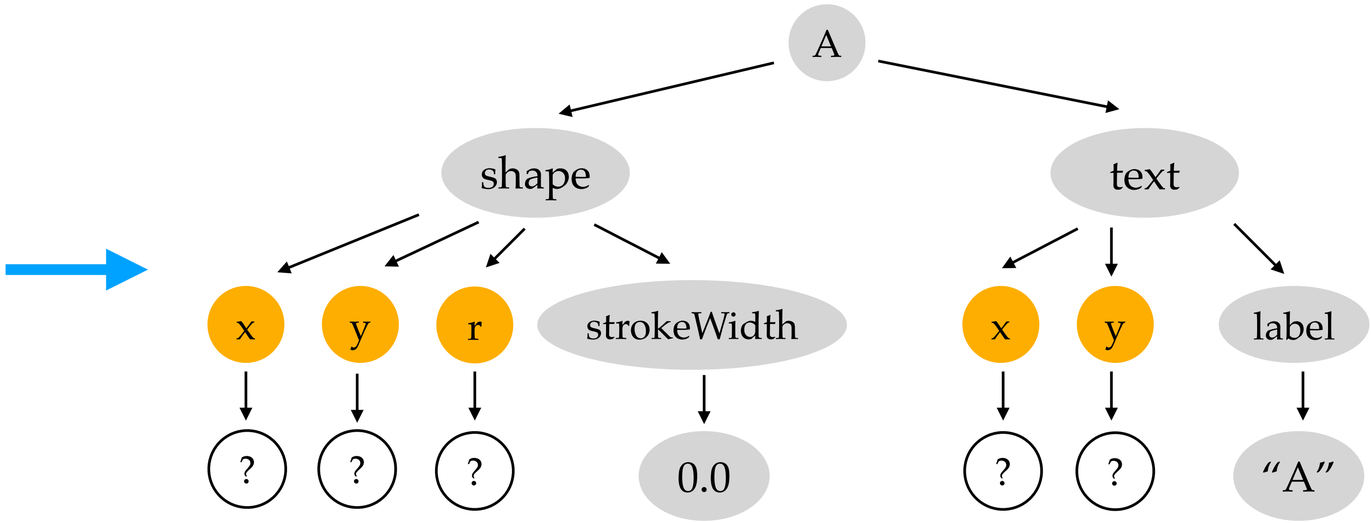


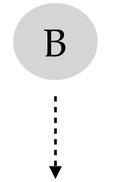
Building the computation graph

Set A, B

B ⊂ A

```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```





Degrees of freedom:

A.shape.x	B.shape.x
A.shape.y	B.shape.y
A.shape.r	B.shape.r
A.text.x	B.text.x
A.text.y	B.text.y

Building the constraint graph

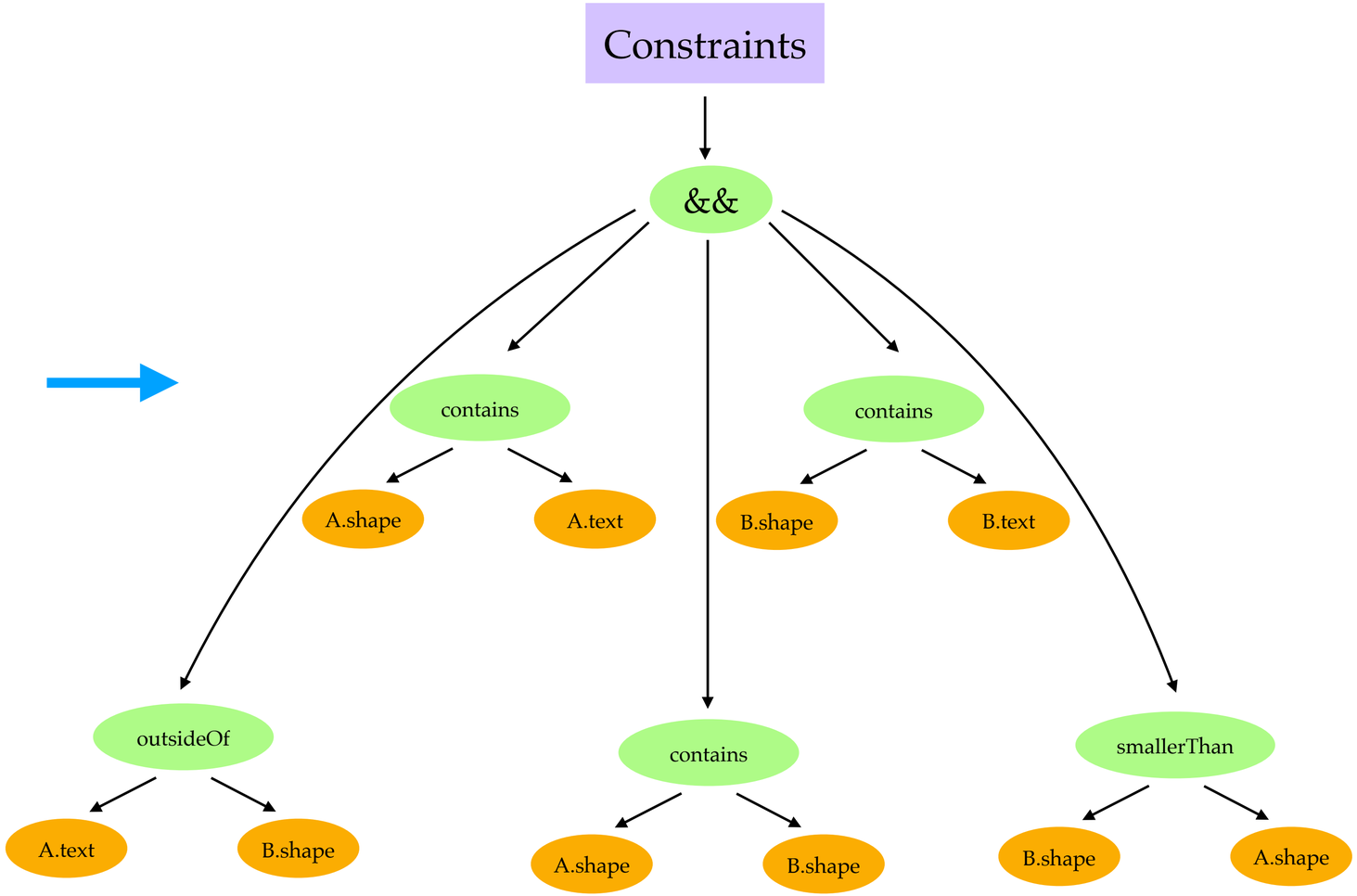
```
Set A, B
B ⊂ A
```

```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```

Building the constraint graph

```
Set A, B
B ⊂ A
```

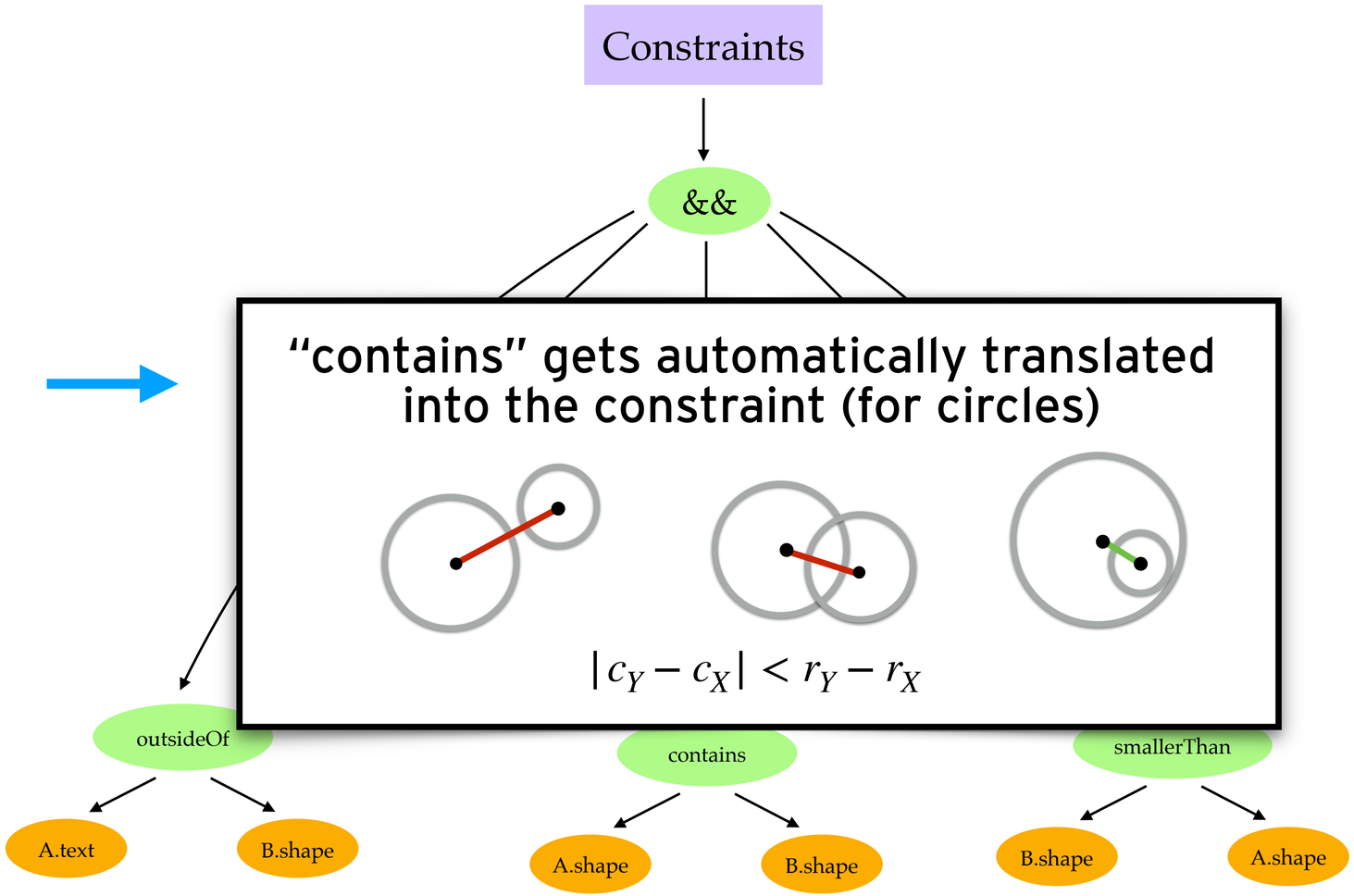
```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```



Building the constraint graph

```
Set A, B
B ⊂ A
```

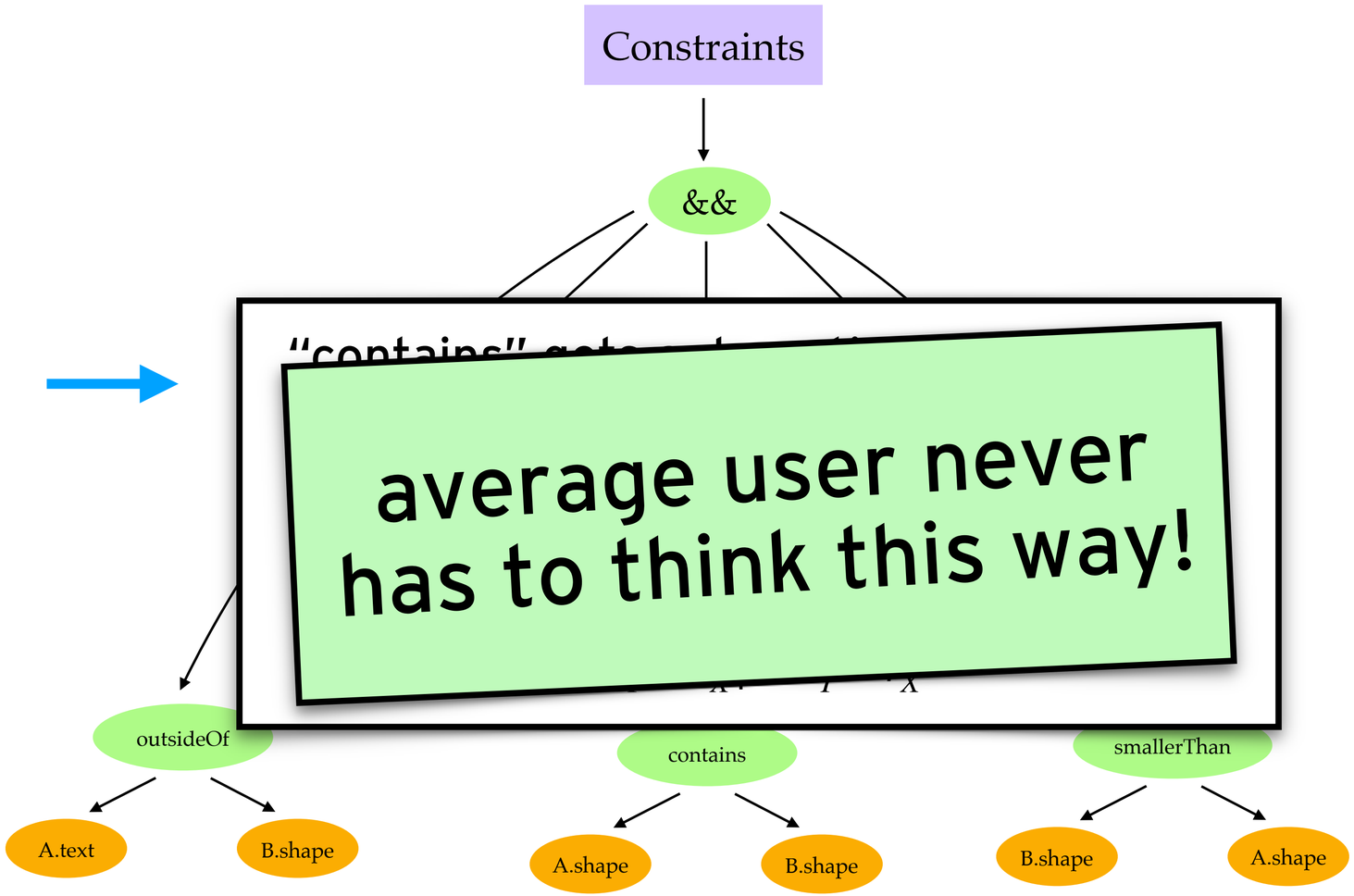
```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```



Building the constraint graph

```
Set A, B
B ⊂ A
```

```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```



Building the objective graph

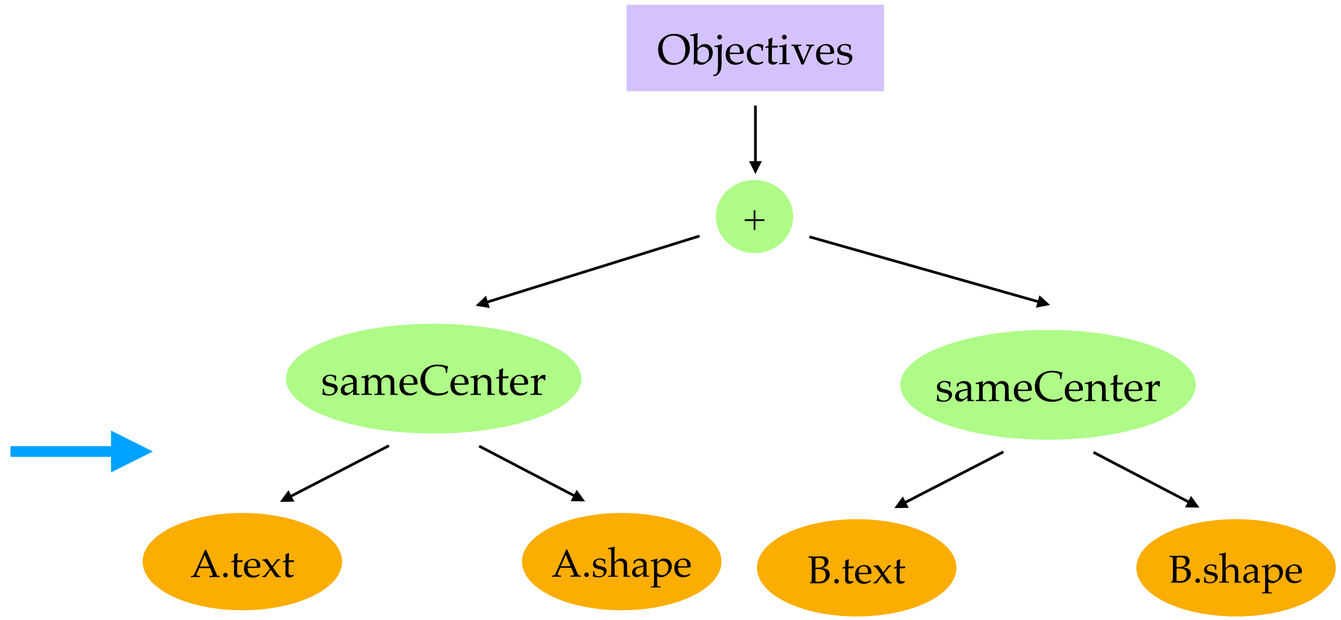
```
Set A, B
B ⊂ A
```

```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```

Building the objective graph

```
Set A, B
B ⊂ A
```

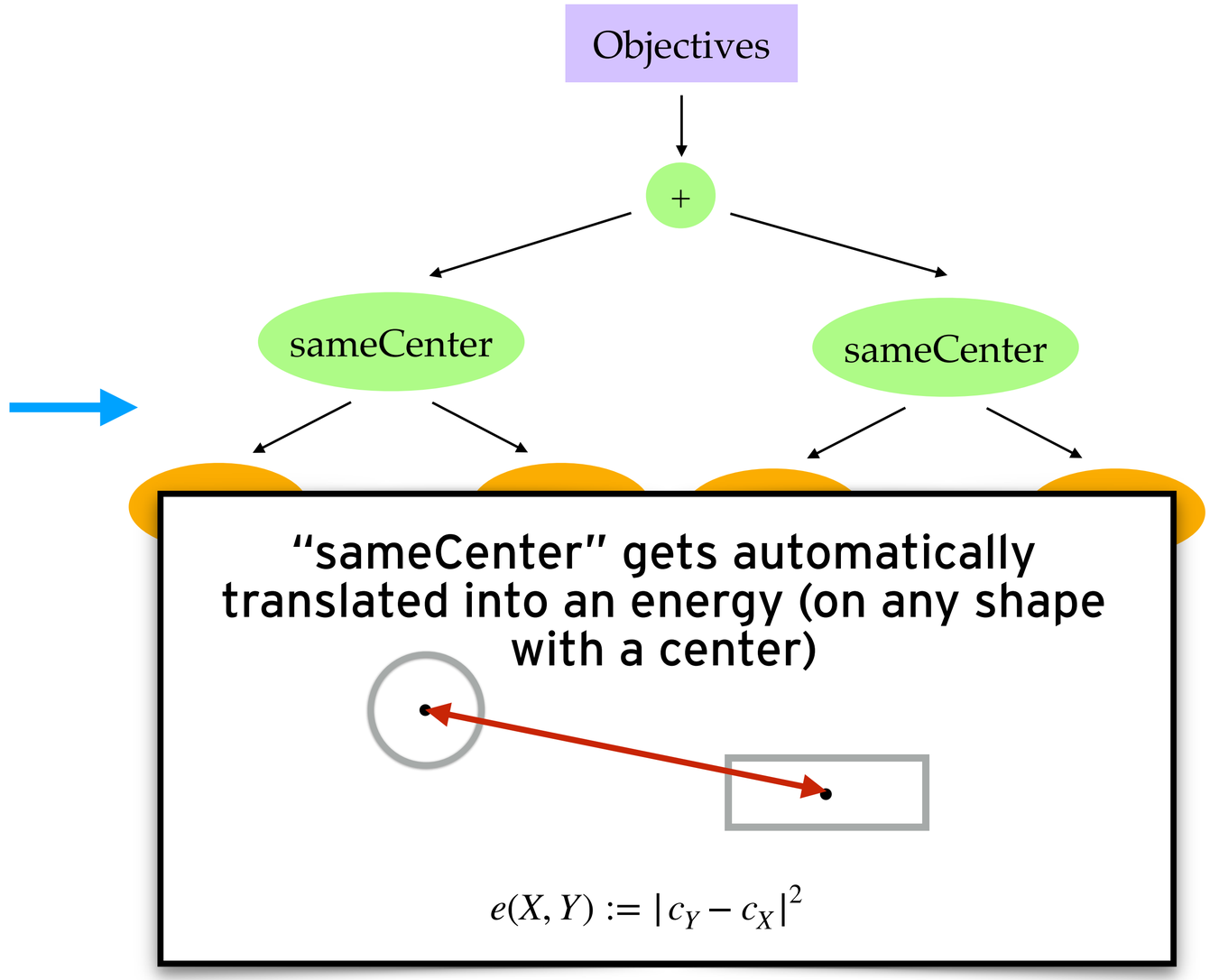
```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```



Building the objective graph

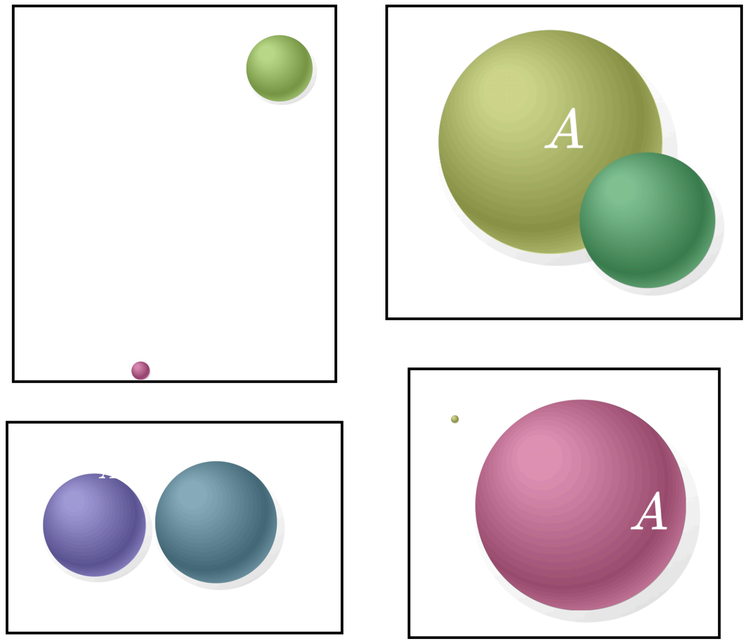
```
Set A, B
B ⊂ A
```

```
forall Set x { -- Sets-Disks.sty
    x.shape = Circle { strokeWidth : 0.0 }
    x.text = Text { string : x.label }
    ensure contains(x.shape, x.text)
    encourage sameCenter(x.text, x.shape)
    layer x.shape below x.text
}

forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```

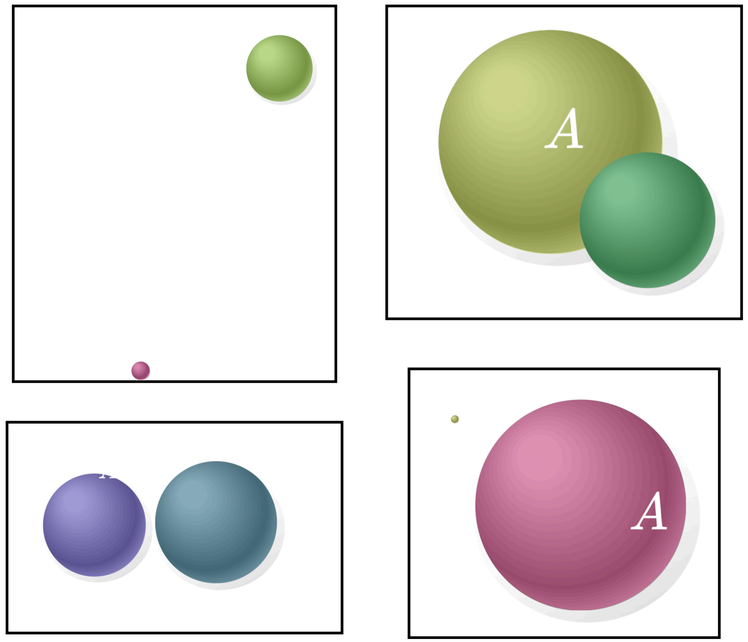


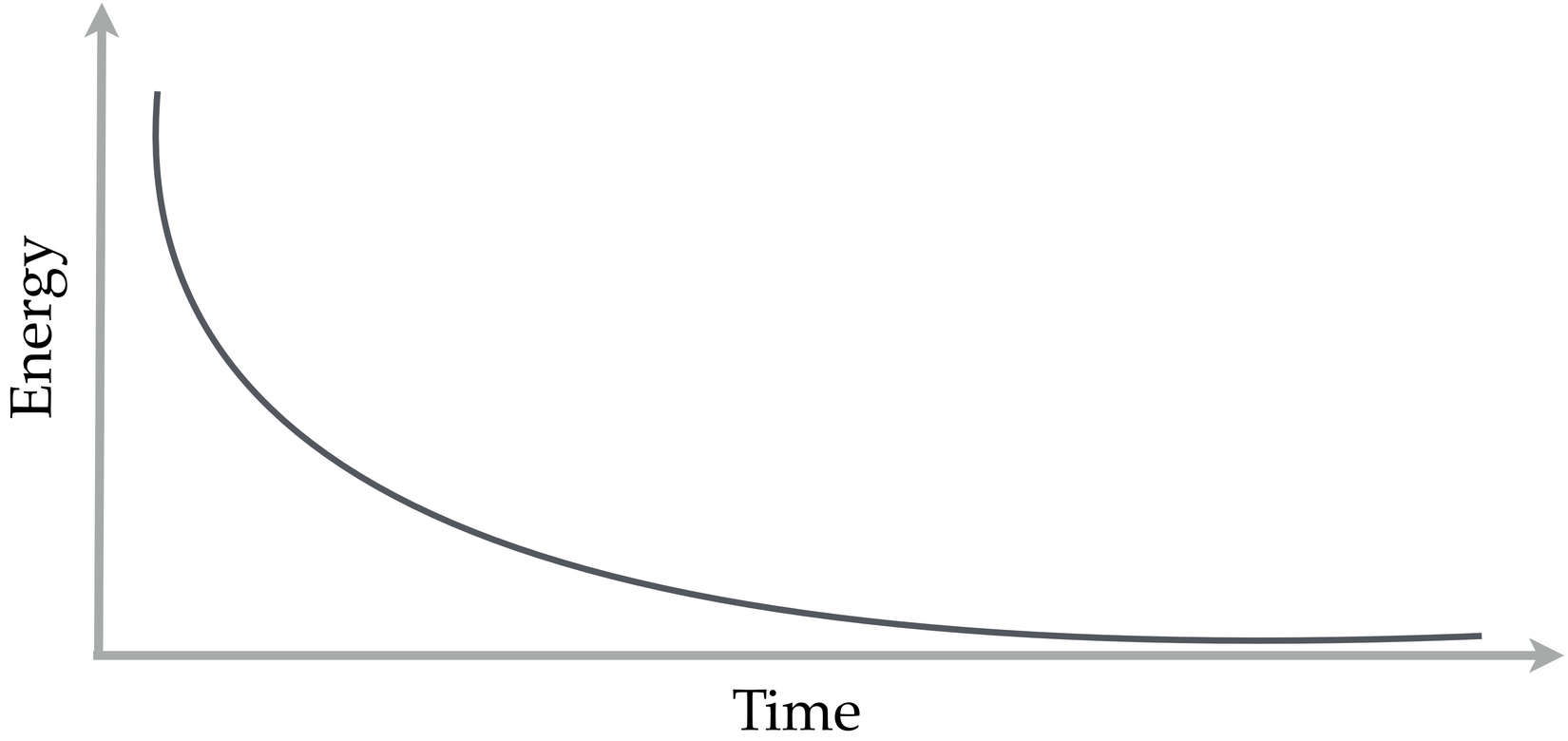
Constrained optimization yields diagrams

Constrained optimization yields diagrams

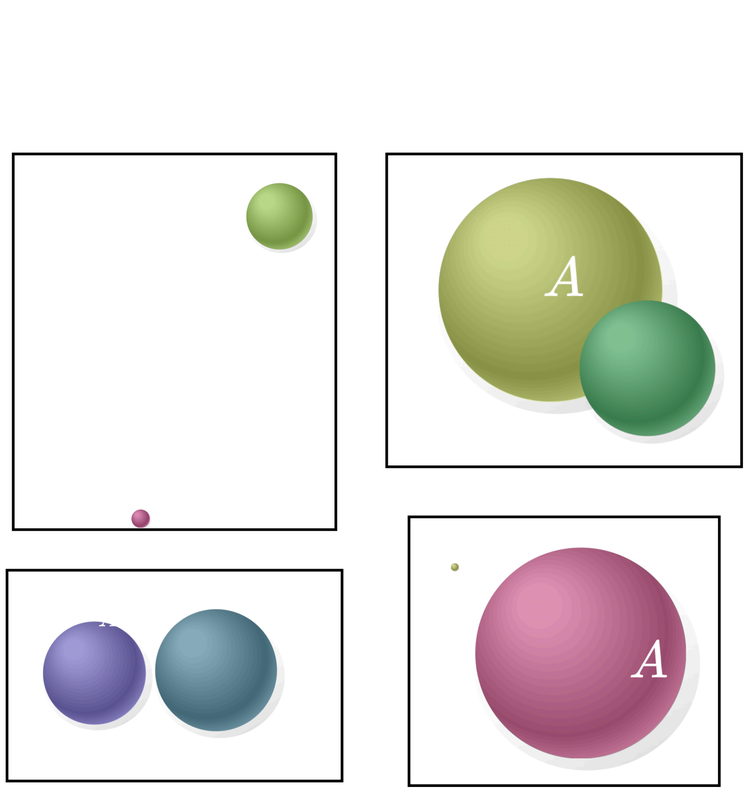


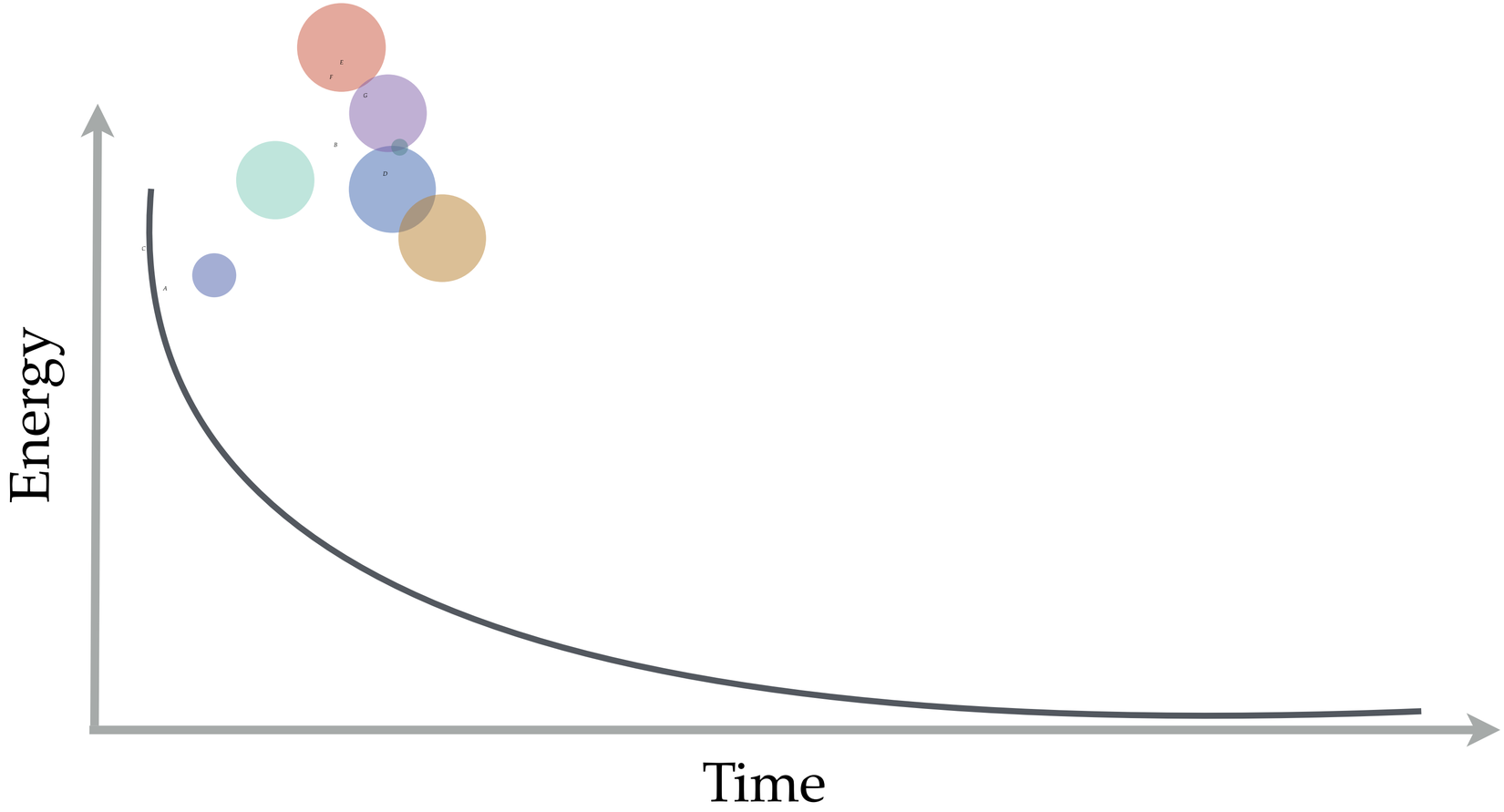
Constrained optimization yields diagrams



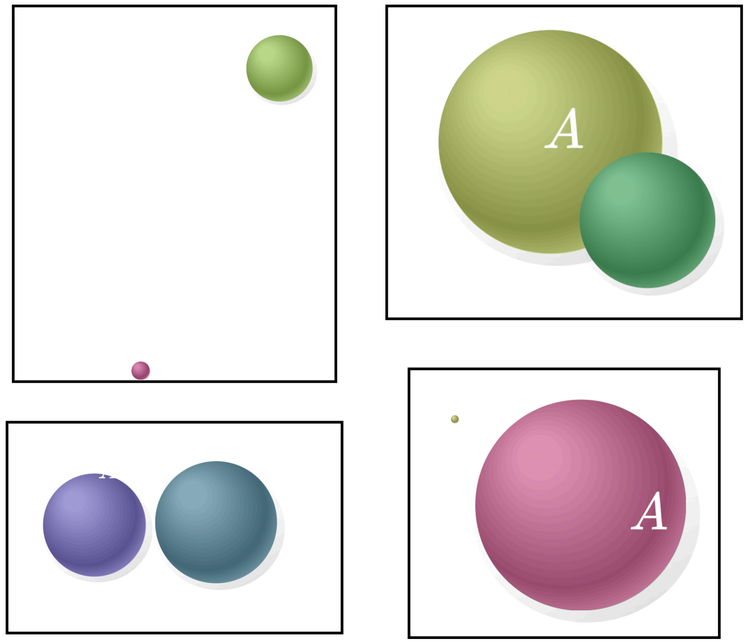


Constrained optimization yields diagrams





Constrained optimization yields diagrams



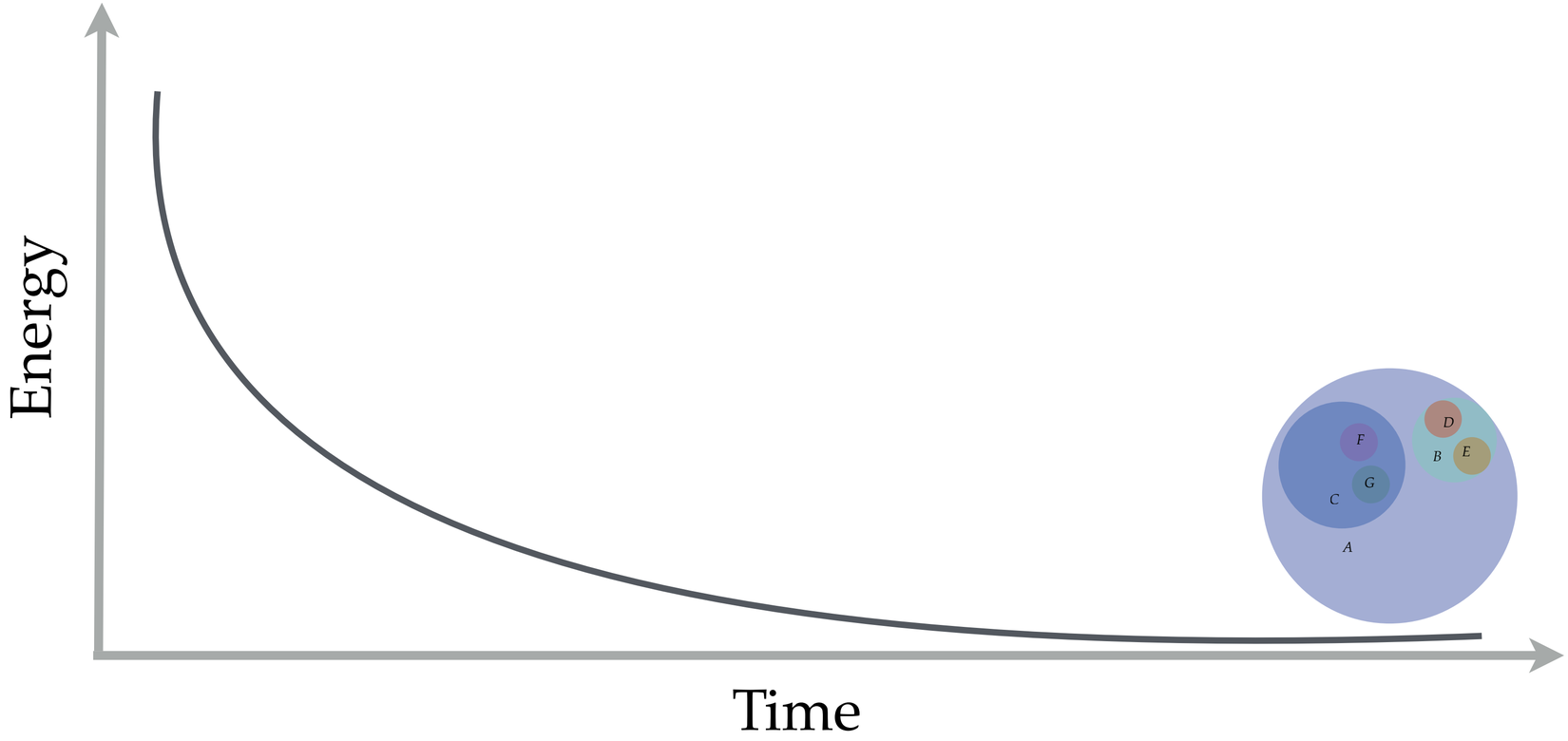


Diagram optimization is hard but tractable

Why it's hard

Why it's tractable

Diagram optimization is hard but tractable

Why it's hard

- User-defined / unpredictable

Why it's tractable

Diagram optimization is hard but tractable

Why it's hard

- User-defined / unpredictable
- Objective has many local minima (nonconvex)

Why it's tractable

Diagram optimization is hard but tractable

Why it's hard

- User-defined / unpredictable
- Objective has many local minima (nonconvex)
- Problem can be ill-conditioned, highly constrained & heterogeneous

Why it's tractable

Diagram optimization is hard but tractable

Why it's hard

- User-defined / unpredictable
- Objective has many local minima (nonconvex)
- Problem can be ill-conditioned, highly constrained & heterogeneous

Why it's tractable

- Only need local minima

Diagram optimization is hard but tractable

Why it's hard

- User-defined / unpredictable
- Objective has many local minima (nonconvex)
- Problem can be ill-conditioned, highly constrained & heterogeneous

Why it's tractable

- Only need local minima
- Few degrees of freedom (10-100)

Diagram optimization is hard but tractable

Why it's hard

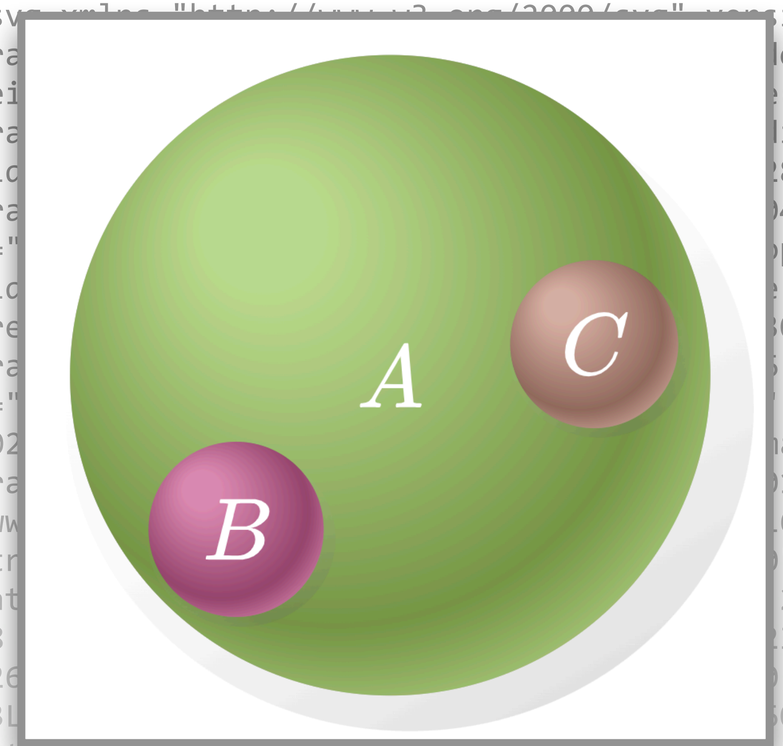
- User-defined / unpredictable
- Objective has many local minima (nonconvex)
- Problem can be ill-conditioned, highly constrained & heterogeneous

Why it's tractable

- Only need local minima
- Few degrees of freedom (10-100)
- Special problem structure can be exploited by solvers

Rendering a solution yields a 2D vector diagram

Rendering a solution yields a 2D vector diagram



Rendering a solution yields a 2D vector diagram

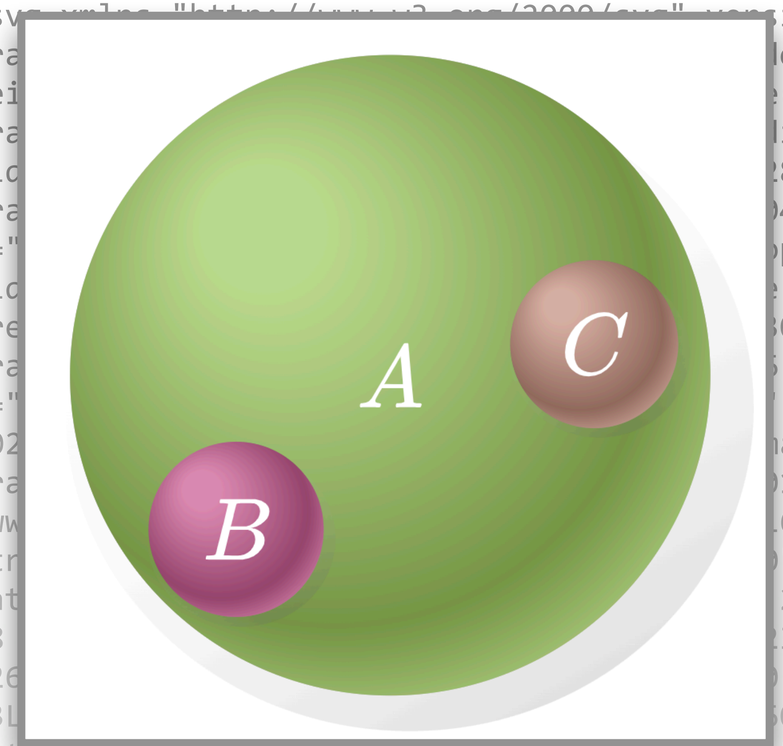
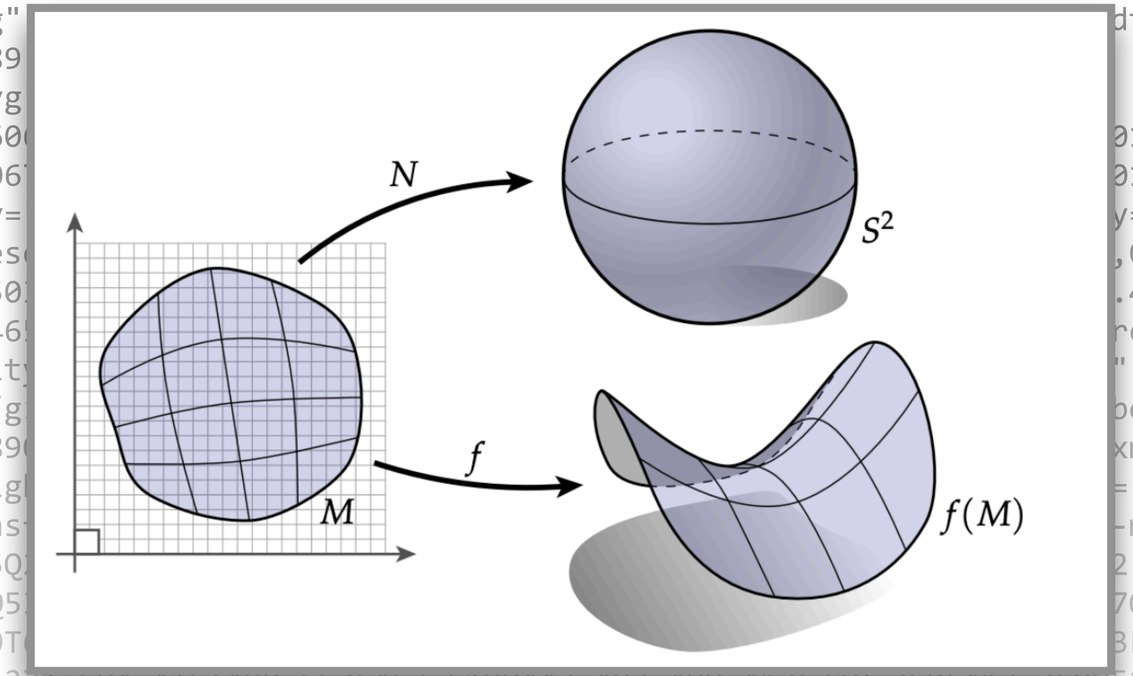


image credit: Crane et al 2013



Rendering a solution yields a 2D vector diagram

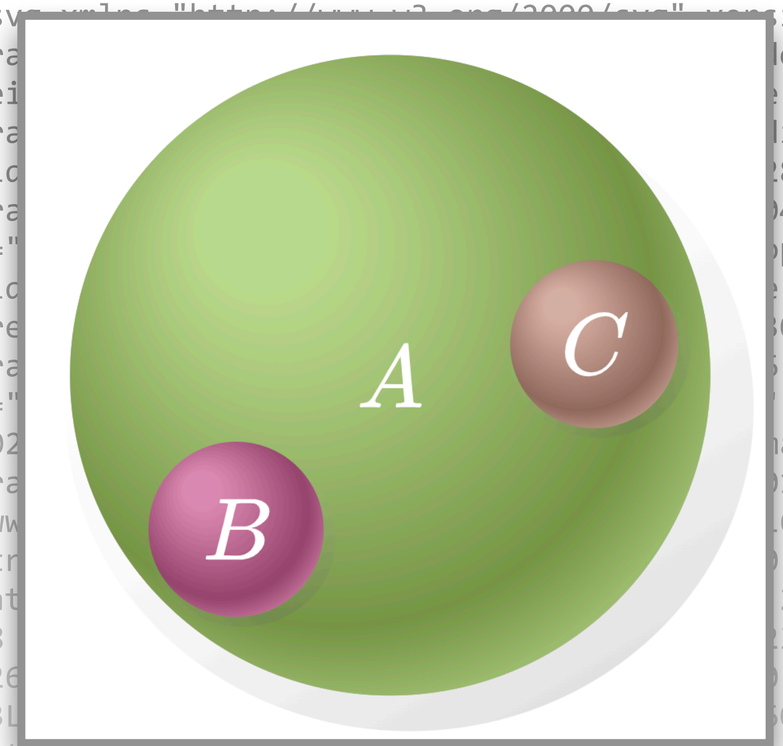
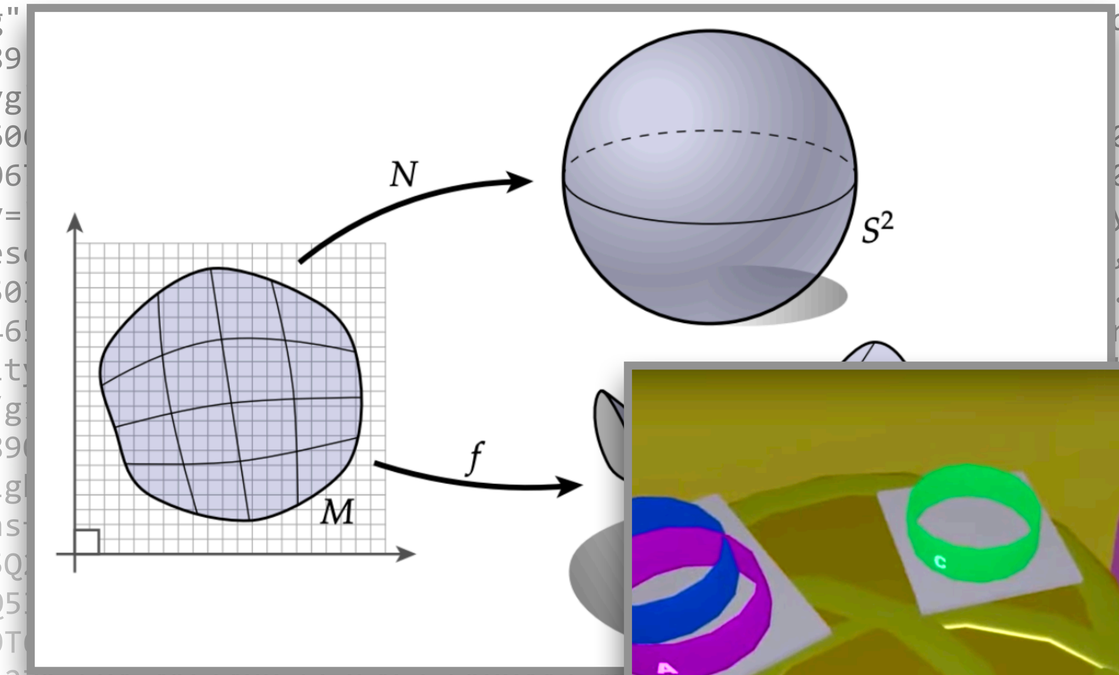


image credit: Crane et al 2013



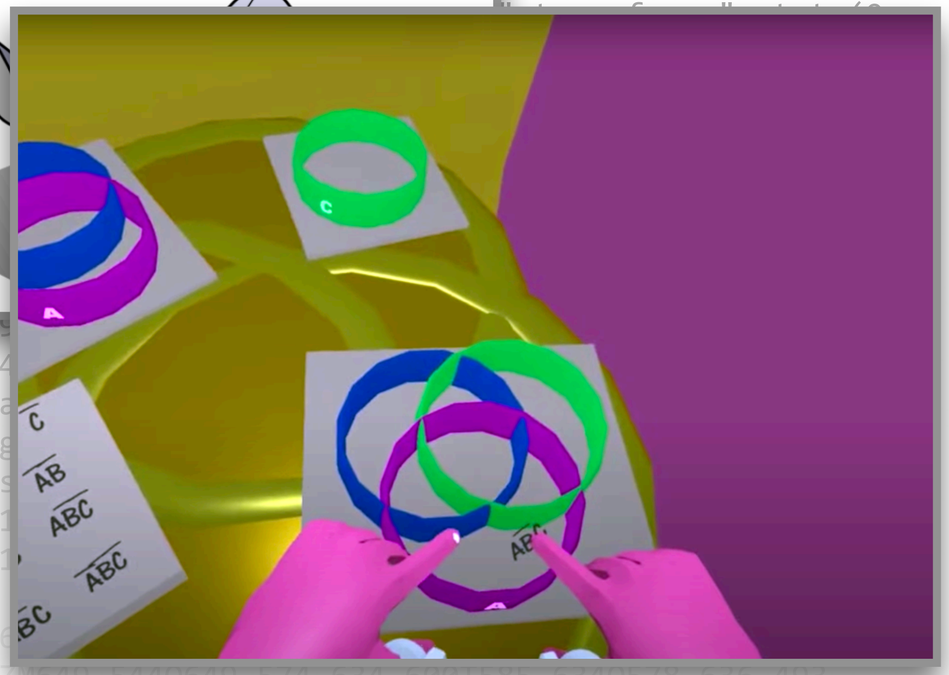
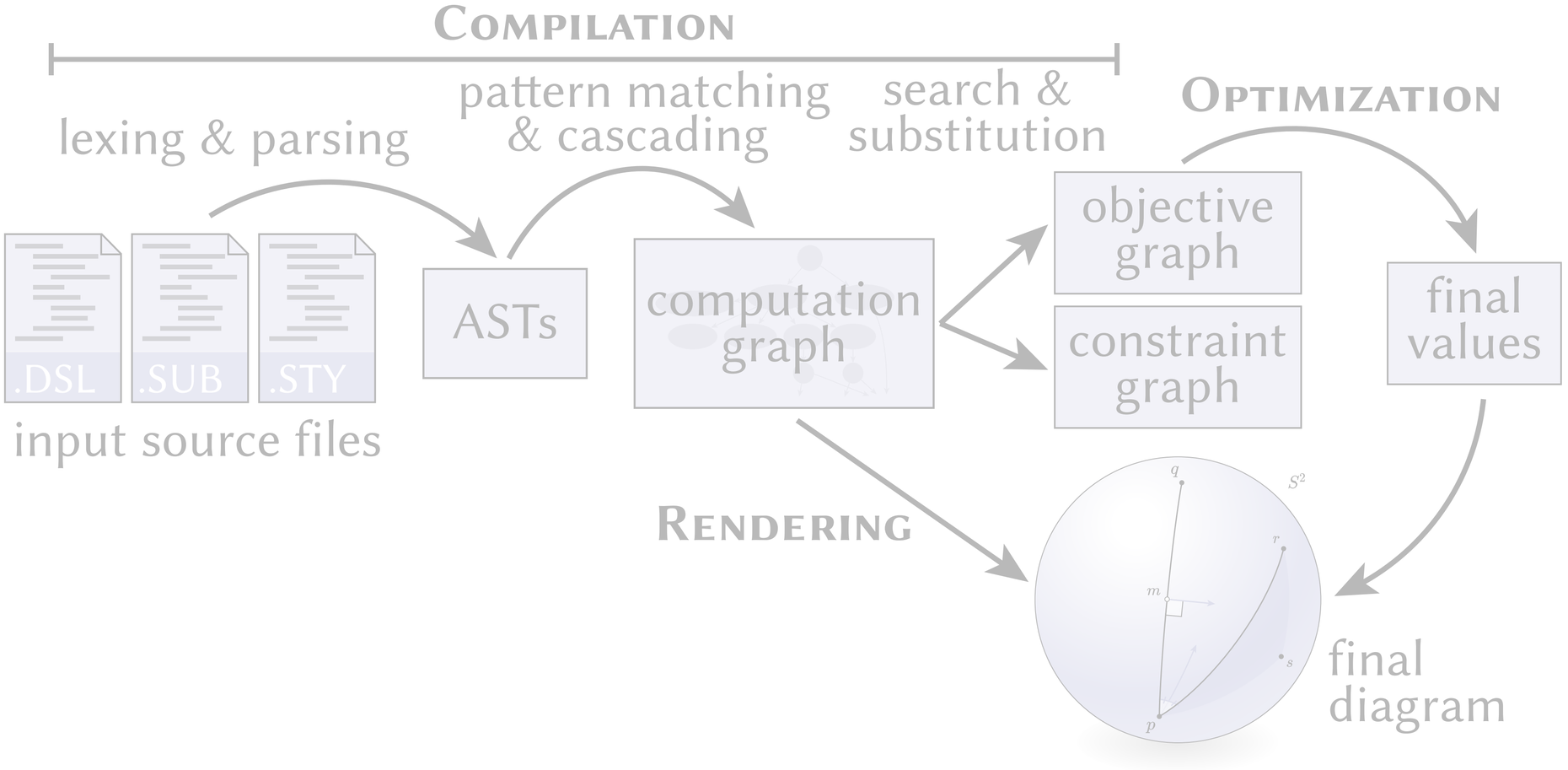
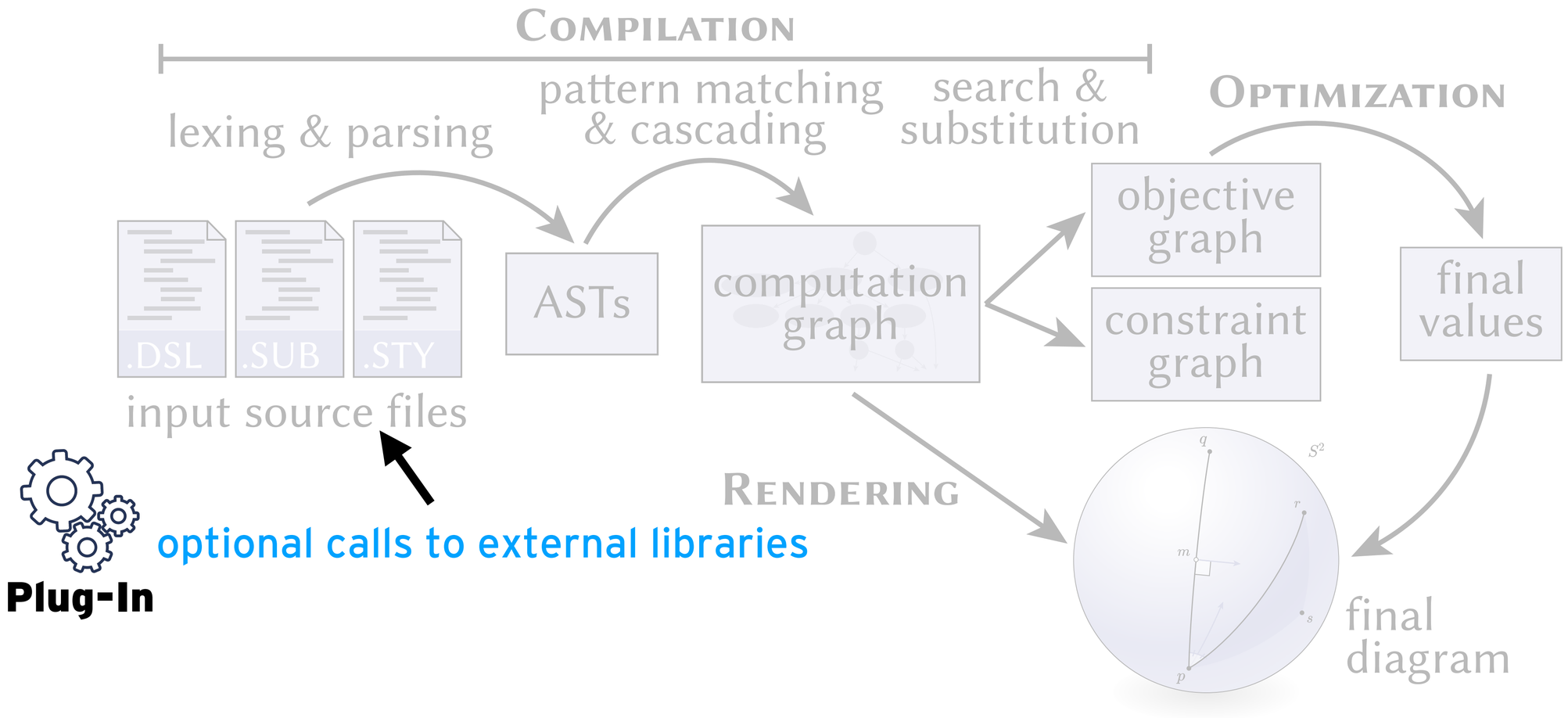


image credit: eleVR / Venn Diagram Museum

Plug-ins enable integration with external code



Plug-ins enable integration with external code



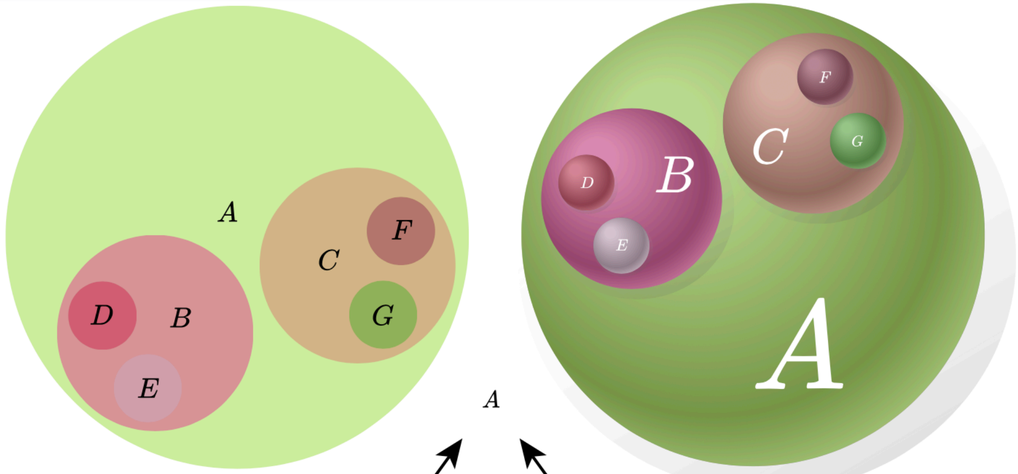
The system in action

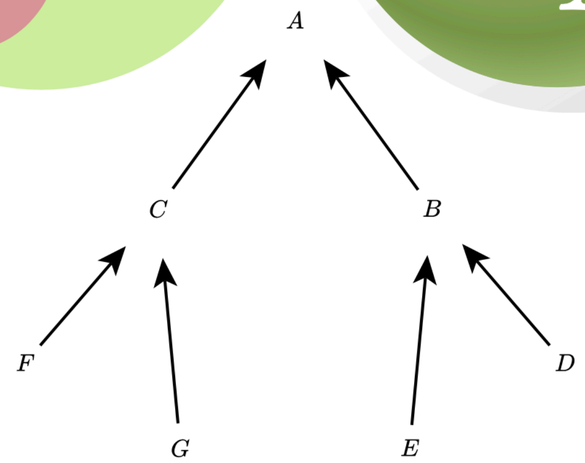
Visualizing sets

Visualizing sets

changing style or visual representation

```
Set A, B, C, D, E, F, G        F ⊂ C        -- Sets.sub
B ⊂ A                          G ⊂ C
C ⊂ A                          E ∩ D = ∅
D ⊂ B                          F ∩ G = ∅
E ⊂ B                          B ∩ C = ∅
```

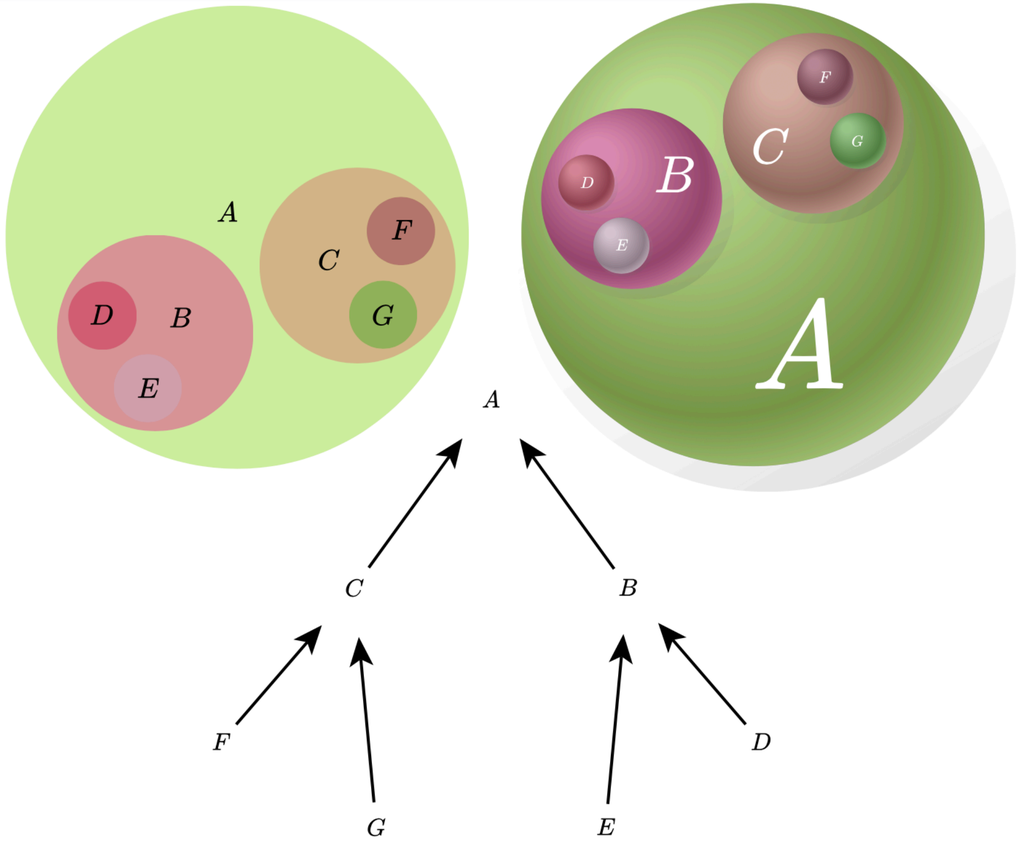




Visualizing sets

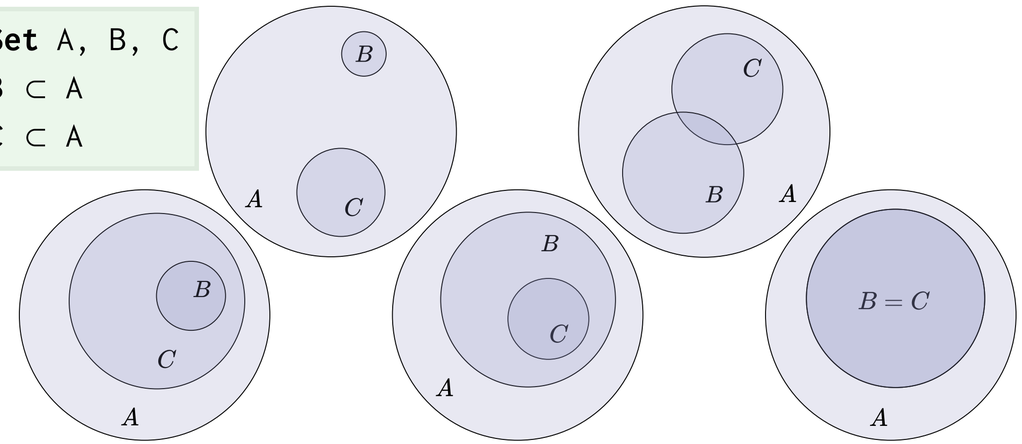
changing style or visual representation

```
Set A, B, C, D, E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C        -- Sets.sub
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```



enumerating possibilities

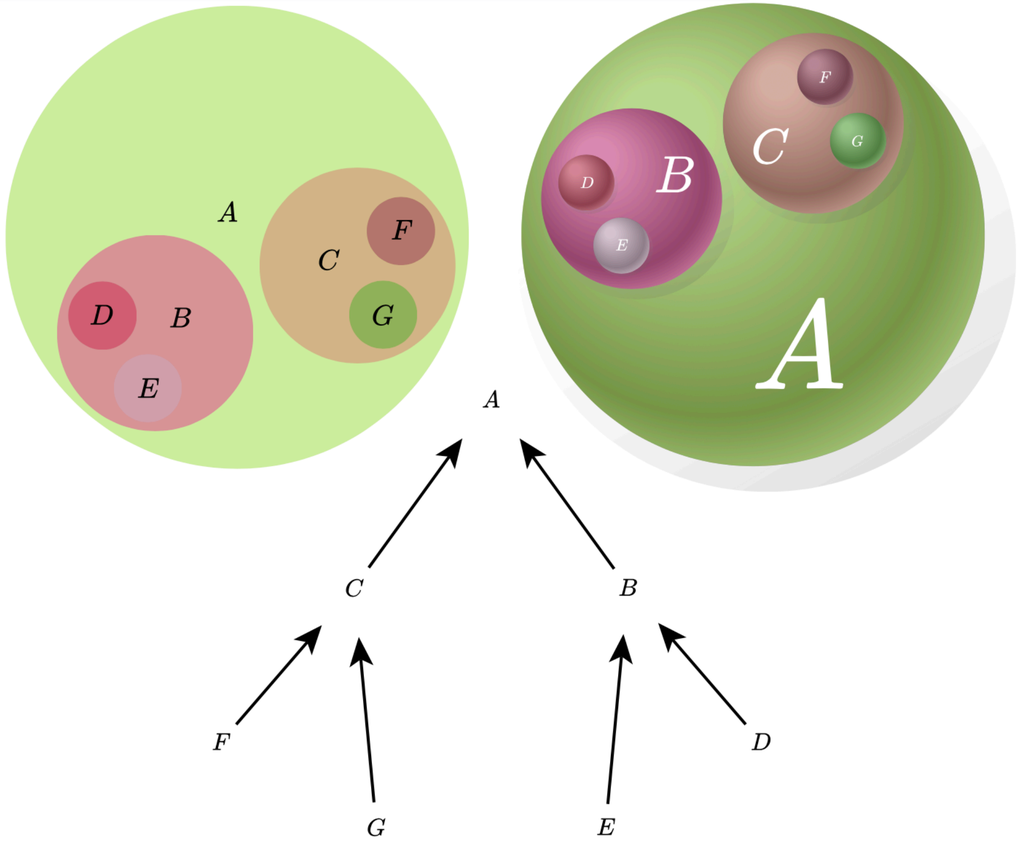
```
Set A, B, C
B ⊂ A
C ⊂ A
```



Visualizing sets

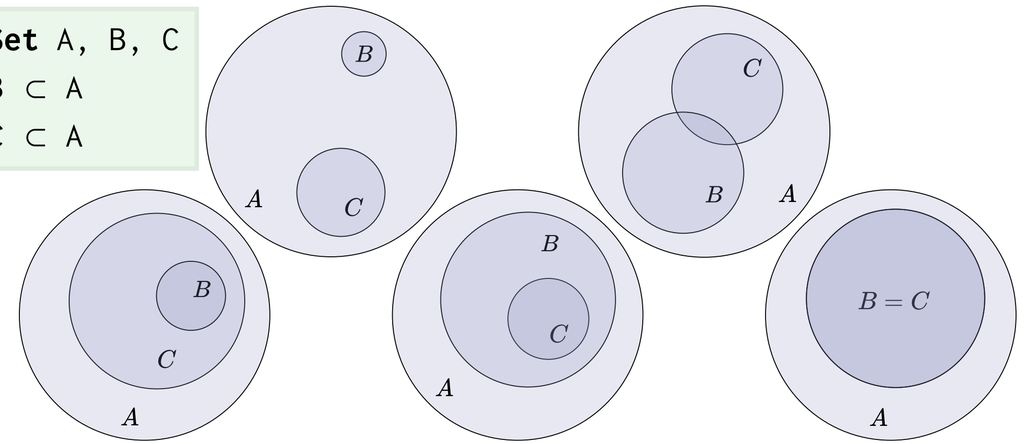
changing style or visual representation

```
Set A, B, C, D, E, F, G
B ⊂ A
C ⊂ A
D ⊂ B
E ⊂ B
F ⊂ C        -- Sets.sub
G ⊂ C
E ∩ D = ∅
F ∩ G = ∅
B ∩ C = ∅
```

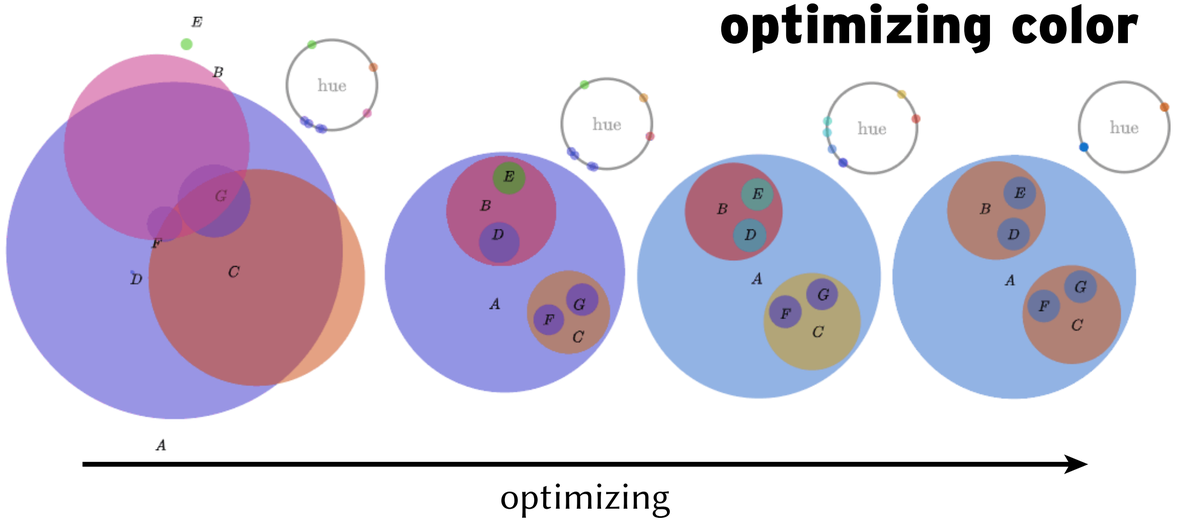


enumerating possibilities

```
Set A, B, C
B ⊂ A
C ⊂ A
```



optimizing color



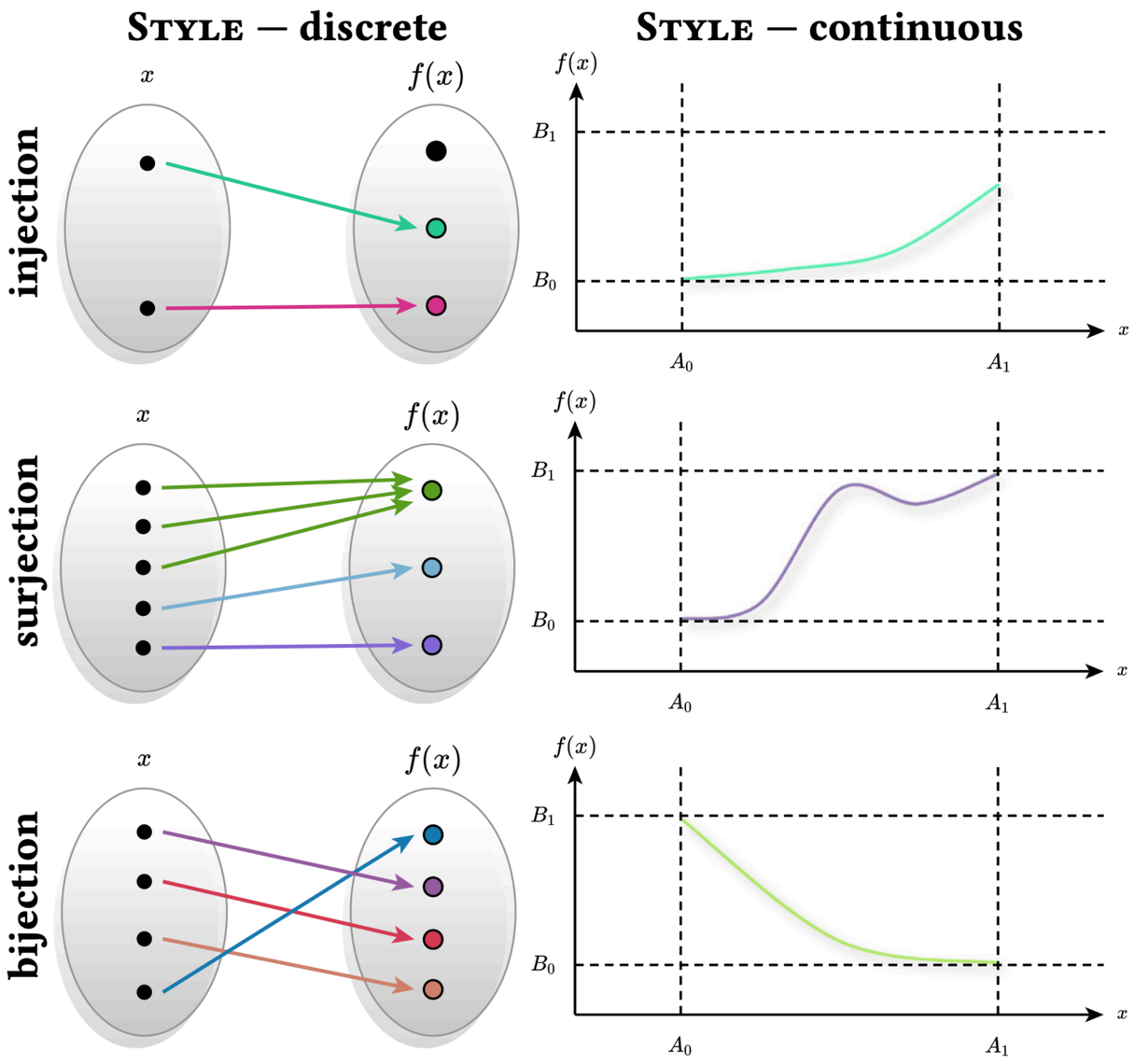
Visualizing functions

Visualizing functions

```
-- Injection.sub
Set A, B
f: A -> B
Injection(f)
Not(Surjection(f))
```

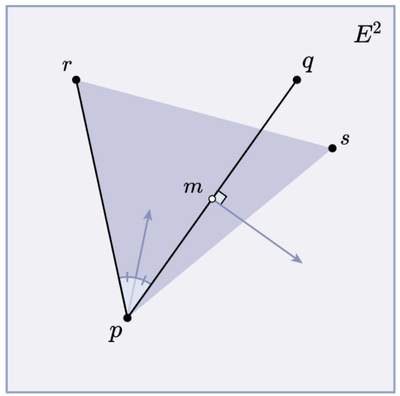
```
-- Surjection.sub
Set A, B
f: A -> B
Surjection(f)
Not(Injection(f))
```

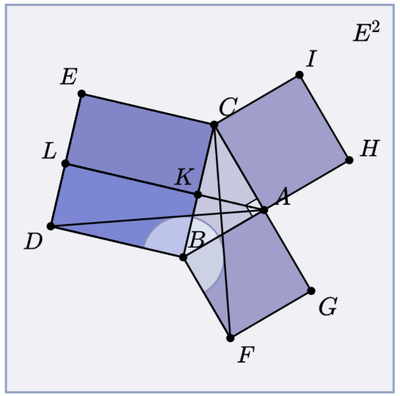
```
-- Bijection.sub
Set A, B
f: A -> B
Surjection(f)
Injection(f)
```

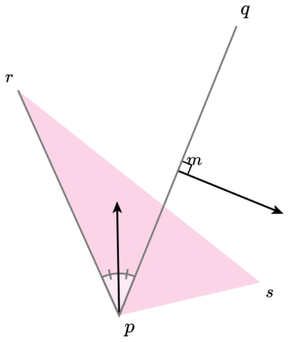


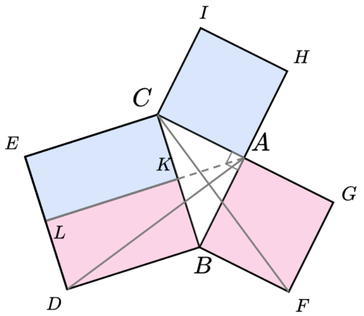
Visualizing Euclidean geometry

Visualizing Euclidean geometry

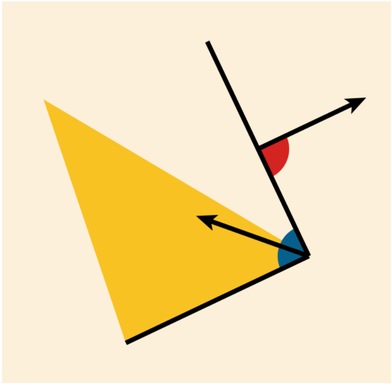


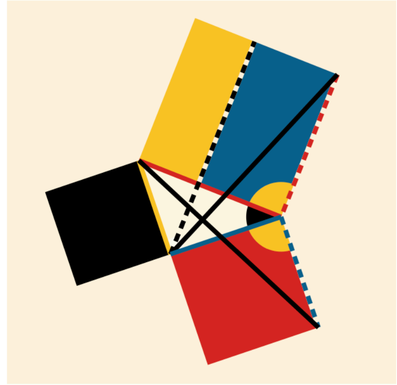






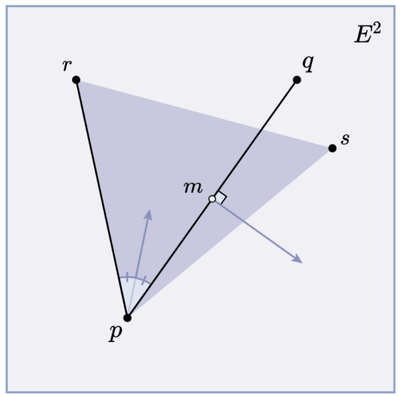
changing the styling

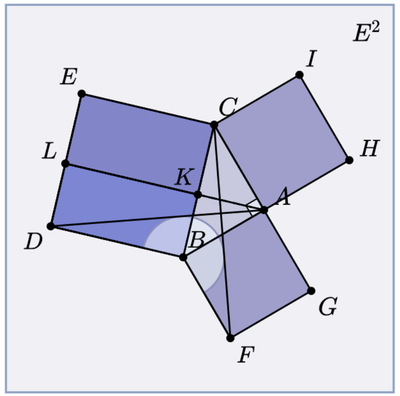


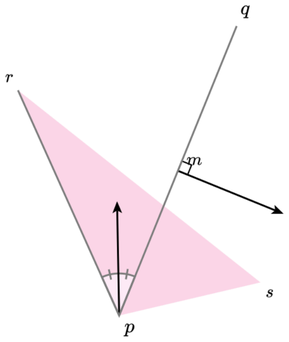


```
Point A, B, C                        -- PythagoreanTheorem.sub
-- define a right triangle           -- split hypotenuse area
Triangle ABC := {A,B,C}              Segment AK := Altitude(ABC,θ)
Angle θ := ∠(C,A,B)                  Point K := Endpoint(AK)
Right(θ)                             Segment DE := {D,E}
-- square each side                  Point L
Point D, E, F, G, H, I               On(L, DE)
Square CBDE := [C,B,D,E]             Segment KL := {K,L}
Disjoint(CBDE, ABC)                  Perpendicular(KL, DE)
Square BAGF := [B,A,G,F]             Rectangle BDLK := {B,D,L,K}
Disjoint(BAGF, ABC)                  Rectangle CKLE := {C,K,L,E}
Square ACIH := [A,C,I,H]             -- (plus additional objects
Disjoint(ACIH, ABC)                  -- from Byrne's diagram)
```

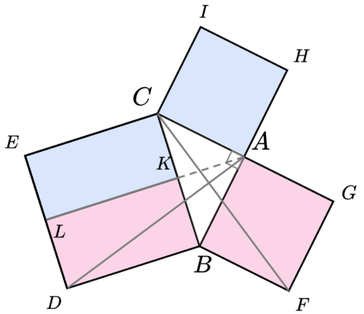
Visualizing Euclidean geometry

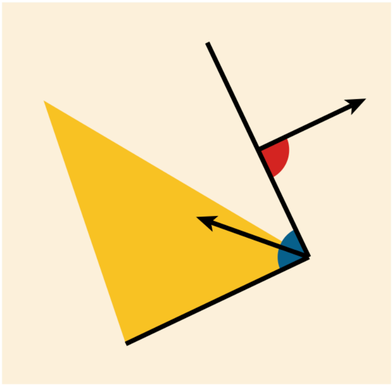


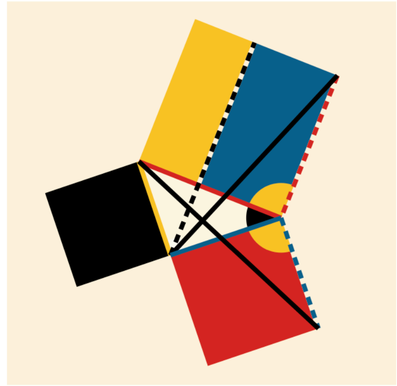




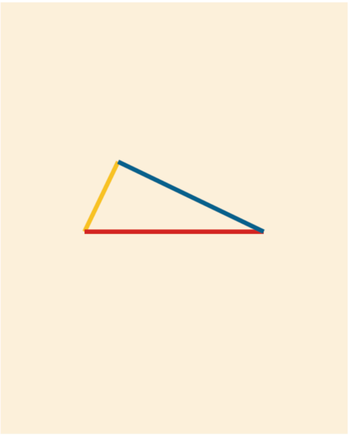
changing the styling

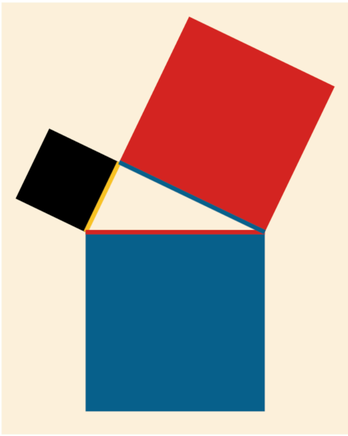


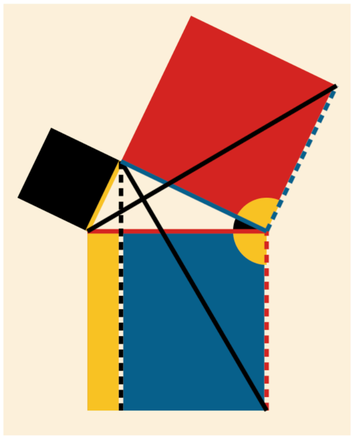




sequences of diagrams





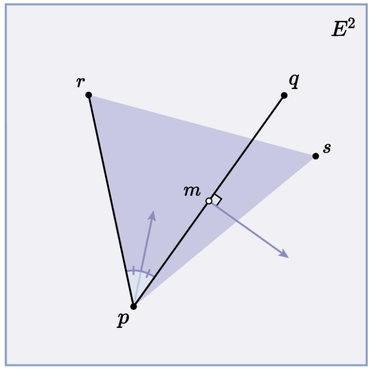


```
Point A, B, C                          -- PythagoreanTheorem.sub
-- define a right triangle             -- split hypotenuse area
Triangle ABC := {A,B,C}                Segment AK := Altitude(ABC,θ)
Angle θ := ∠(C,A,B)                    Point K := Endpoint(AK)
Right(θ)                               Segment DE := {D,E}
-- square each side                    Point L
Point D, E, F, G, H, I                 On(L, DE)
Square CBDE := [C,B,D,E]               Segment KL := {K,L}
Disjoint(CBDE, ABC)                    Perpendicular(KL, DE)
Square BAGF := [B,A,G,F]               Rectangle BDLK := {B,D,L,K}
Disjoint(BAGF, ABC)                    Rectangle CKLE := {C,K,L,E}
Square ACIH := [A,C,I,H]               -- (plus additional objects
Disjoint(ACIH, ABC)                    -- from Byrne's diagram)
```

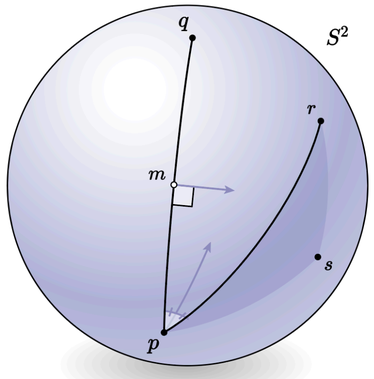
Visualizing neutral geometry

Visualizing neutral geometry

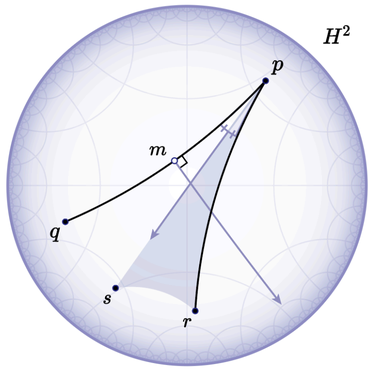
changing the geometric interpretation



Style–Euclidean



Style–spherical

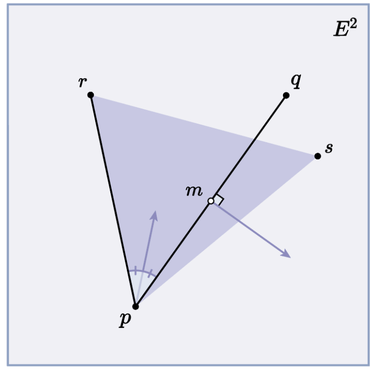


Style–hyperbolic

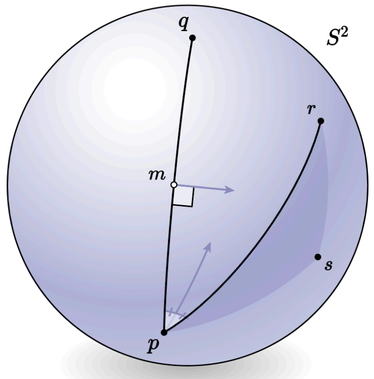
```
Point p, q, r, s
Segment a := p, q
Segment b := p, r
Point m := Midpoint(a)
Angle theta := ∠(q, p, r)
Triangle t := p, r, s
Ray w := Bisector(theta)
Ray h := PerpendicularBisector(a)
```

Visualizing neutral geometry

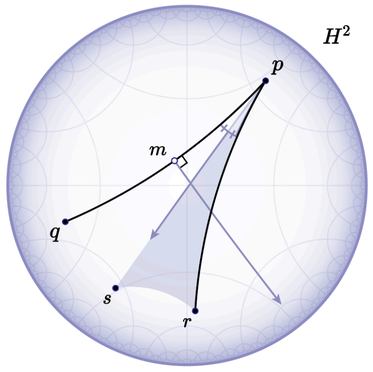
changing the geometric interpretation



Style–Euclidean



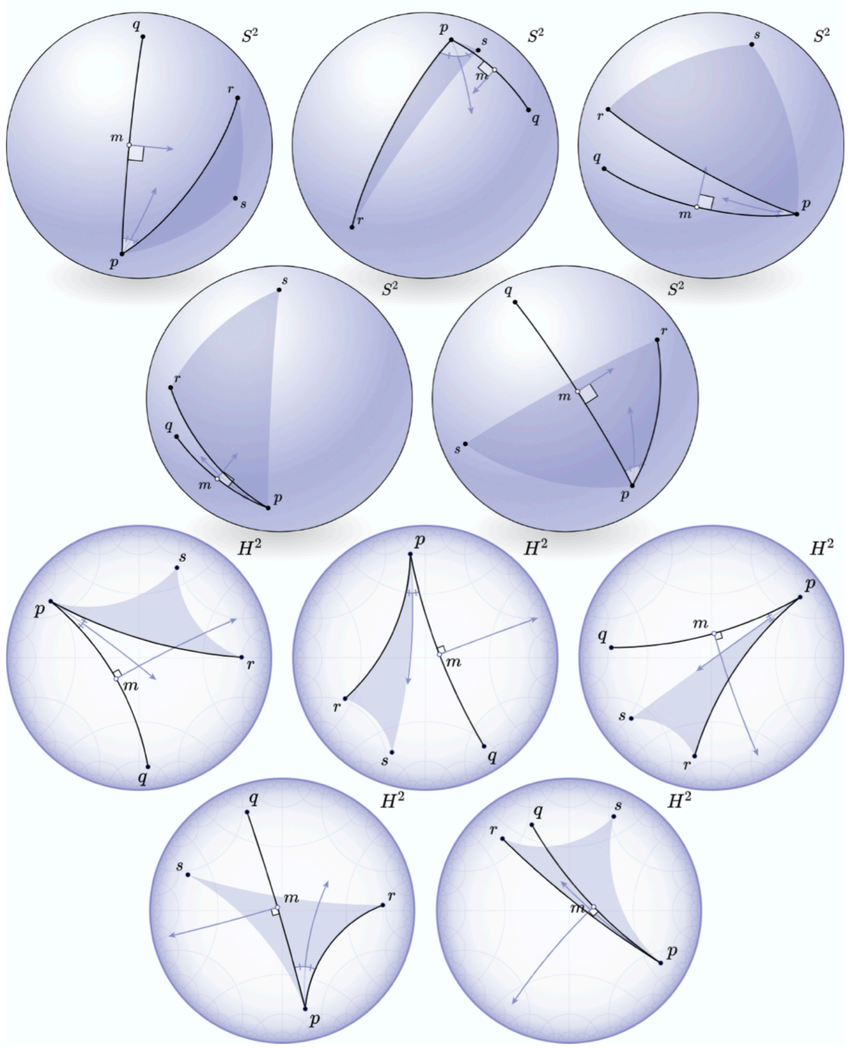
Style–spherical



Style–hyperbolic

```
Point p, q, r, s
Segment a := p, q
Segment b := p, r
Point m := Midpoint(a)
Angle theta := ∠(q, p, r)
Triangle t := p, r, s
Ray w := Bisector(theta)
Ray h := PerpendicularBisector(a)
```

generating a gallery of alternatives

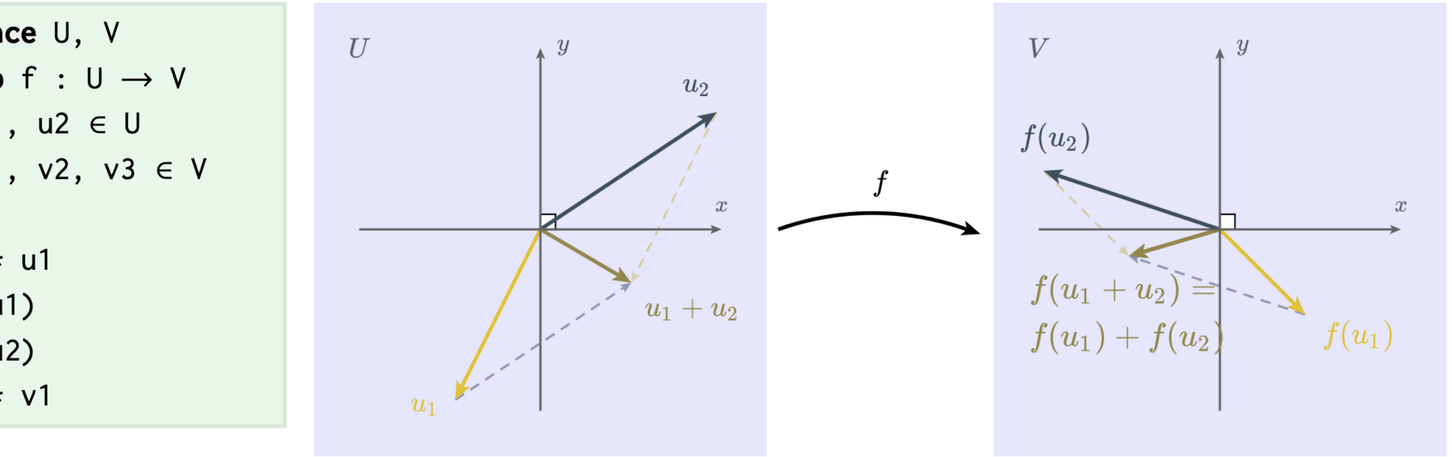


Visualizing vectors

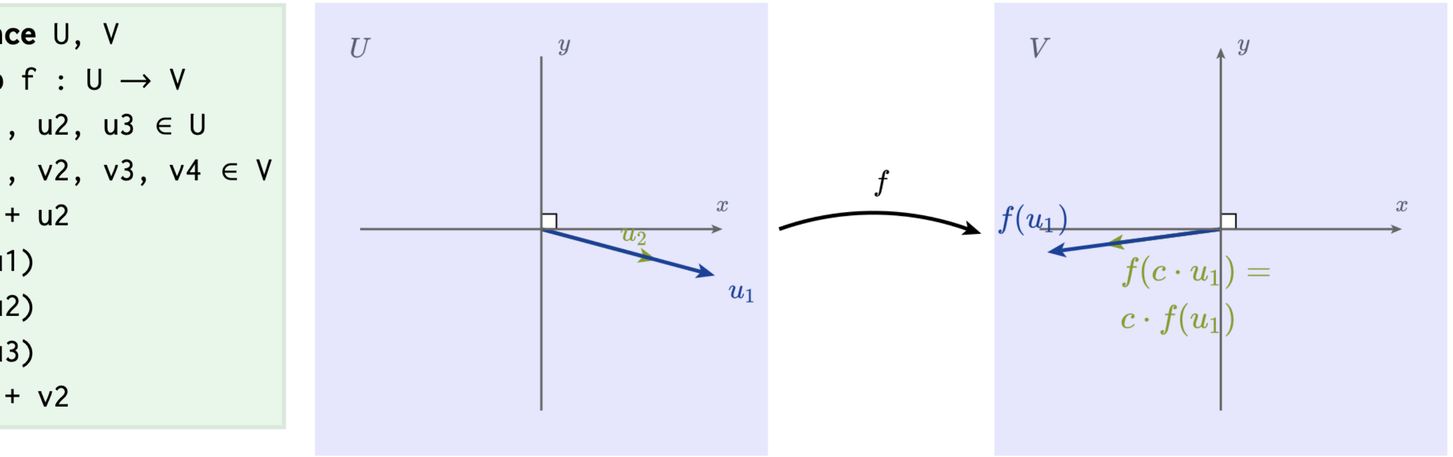
Visualizing vectors

composition of math translates to composition of visualization

```
VectorSpace U, V
LinearMap f : U ⟶ V
Vector u1, u2 ∈ U
Vector v1, v2, v3 ∈ V
Scalar c
u2 := c * u1
v1 := f(u1)
v2 := f(u2)
v3 := c * v1
```



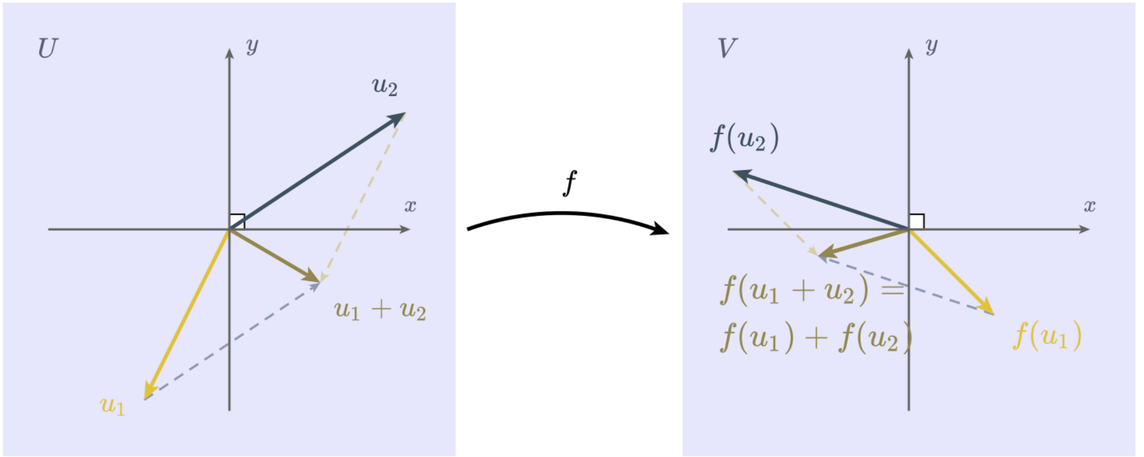

```
VectorSpace U, V
LinearMap f : U ⟶ V
Vector u1, u2, u3 ∈ U
Vector v1, v2, v3, v4 ∈ V
u3 := u1 + u2
v1 := f(u1)
v2 := f(u2)
v3 := f(u3)
v4 := v1 + v2
```



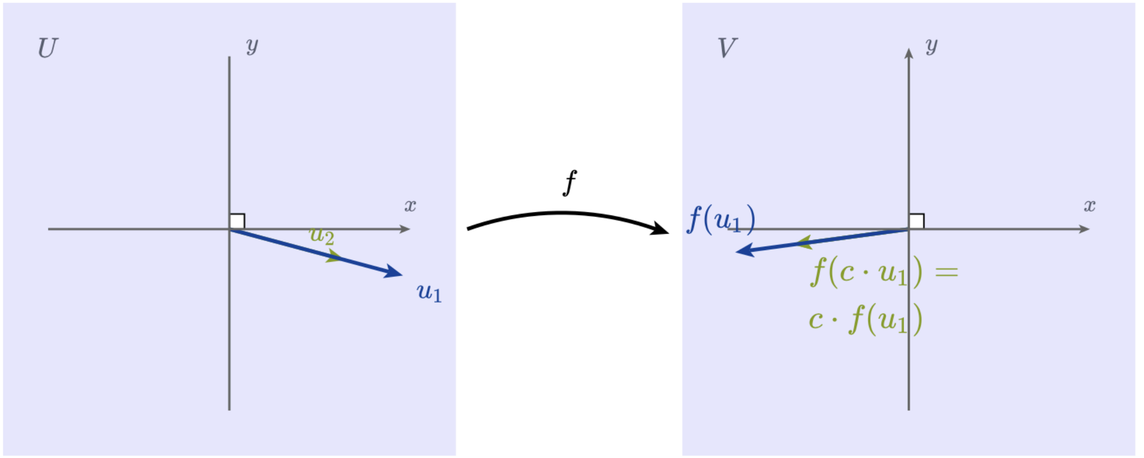
Visualizing vectors

composition of math translates to composition of visualization

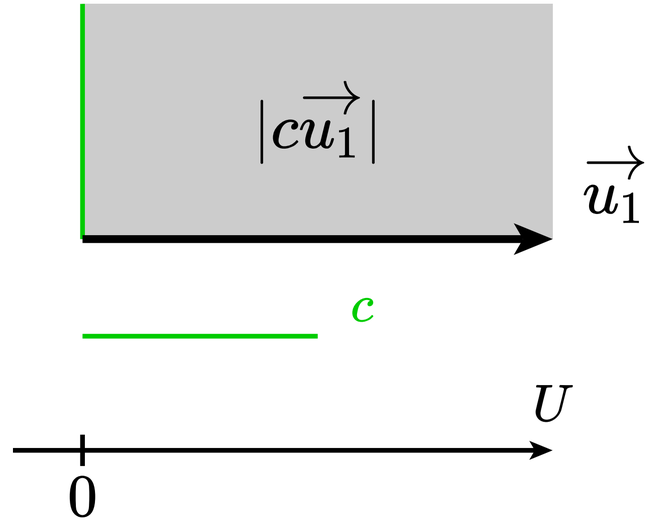
```
VectorSpace U, V
LinearMap f : U → V
Vector u1, u2 ∈ U
Vector v1, v2, v3 ∈ V
Scalar c
u2 := c * u1
v1 := f(u1)
v2 := f(u2)
v3 := c * v1
```




```
VectorSpace U, V
LinearMap f : U → V
Vector u1, u2, u3 ∈ U
Vector v1, v2, v3, v4 ∈ V
u3 := u1 + u2
v1 := f(u1)
v2 := f(u2)
v3 := f(u3)
v4 := v1 + v2
```



adapting the representation to the dimension

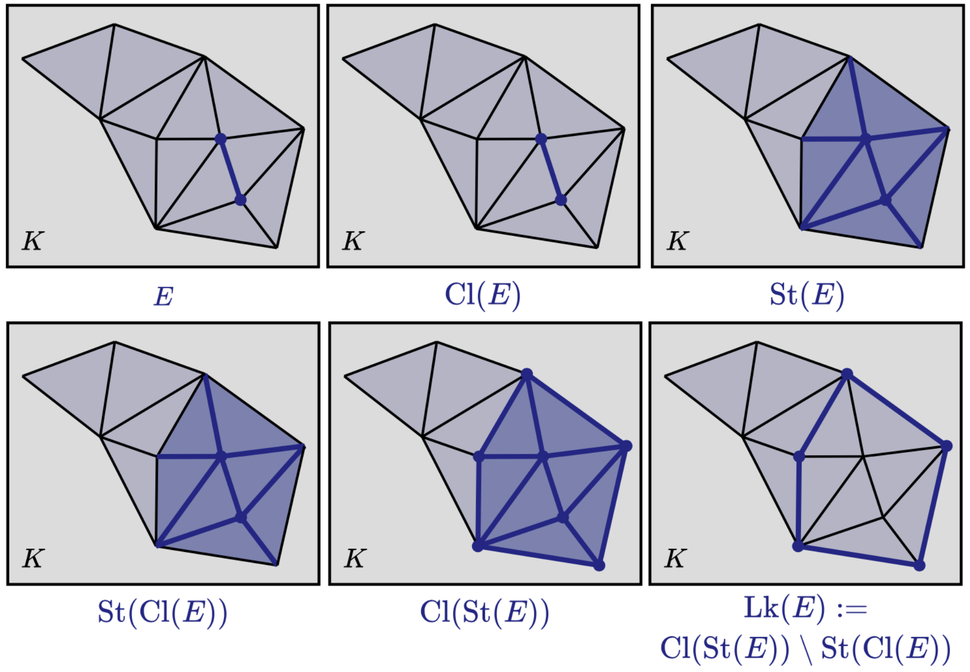


Visualizing meshes

Visualizing meshes

illustrating steps of a complex process

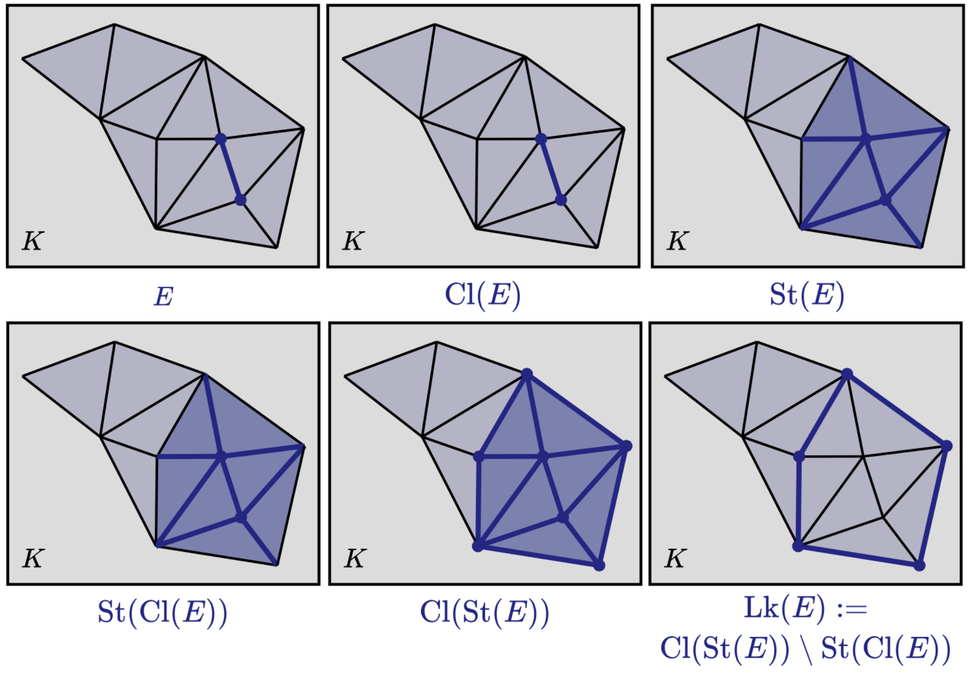
```
SimplicialComplex K
Edge e ∈ K
Subcomplex E ⊆ K
E := Closure(e)
SimplicialSet StE ⊆ K
StE := Star(E)
Subcomplex ClStE ⊆ K
ClStE := Closure(StE)
Subcomplex ClE ⊆ K
ClE := Closure(E)
SimplicialSet StClE ⊆ K
StClE := Star(ClE)
SimplicialSet LkE ⊆ K
LkE := SetMinus(ClStE, StClE)
```



Visualizing meshes

illustrating steps of a complex process

```
SimplicialComplex K

Edge e ∈ K
Subcomplex E ⊆ K
E := Closure(e)

SimplicialSet StE ⊆ K
StE := Star(E)

Subcomplex ClStE ⊆ K
ClStE := Closure(StE)
Subcomplex ClE ⊆ K
ClE := Closure(E)
SimplicialSet StClE ⊆ K
StClE := Star(ClE)
SimplicialSet LkE ⊆ K
LkE := SetMinus(ClStE, StClE)
```

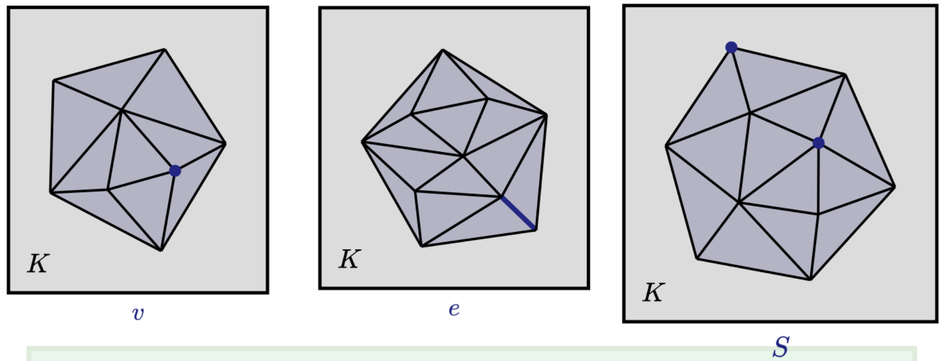


using many examples to build intuition

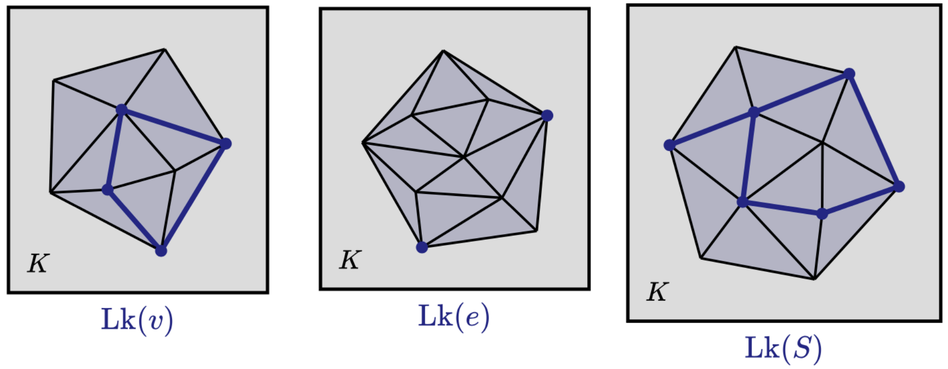
```
Vertex S ∈ K
```

```
Edge S ∈ K
```

```
Vertex v1 ∈ K
Vertex v2 ∈ K
SimplicialSet S ⊆ K
S := {{v1},{v2}}
```



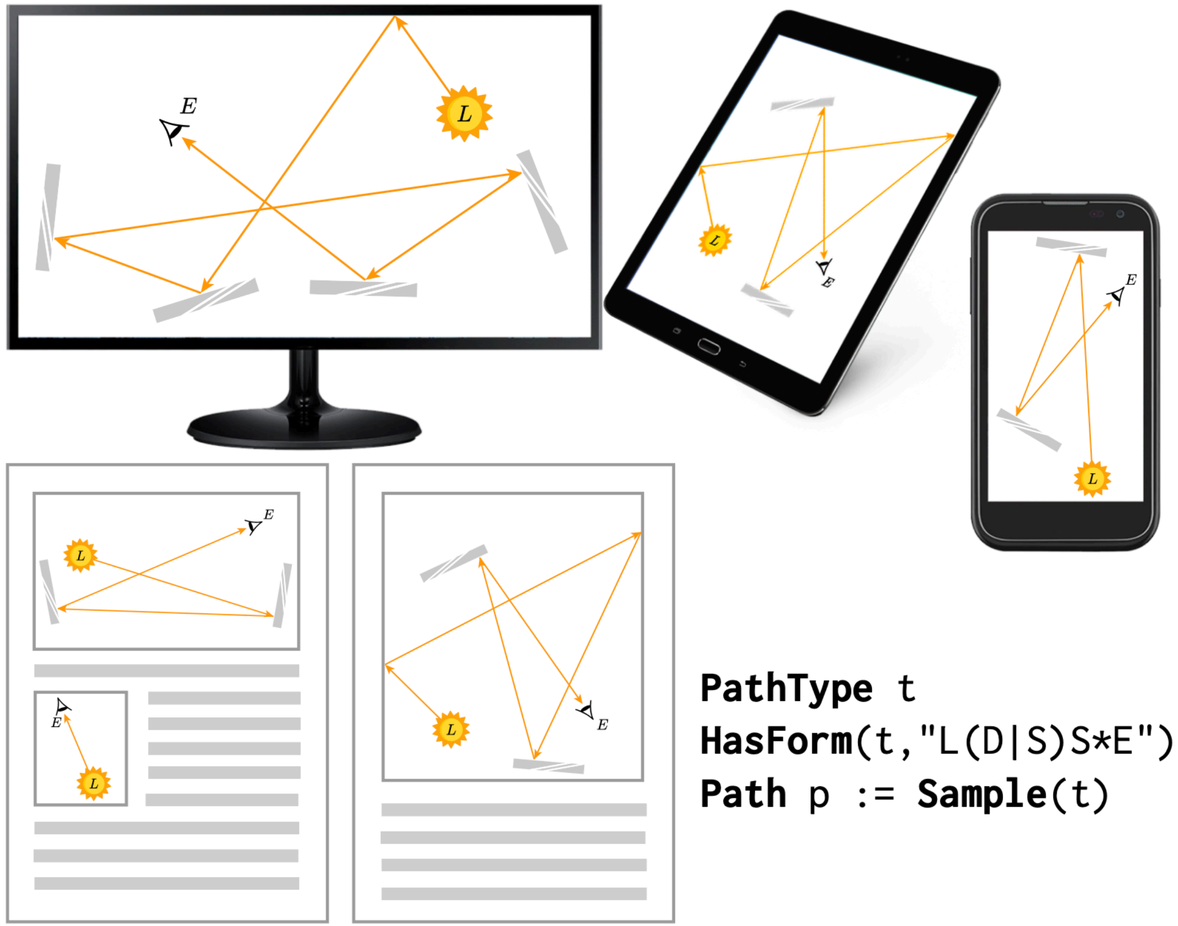

```
SimplicialComplex K
SimplicialSet LkS := Link(S)
```



Visualizing rays

Visualizing rays

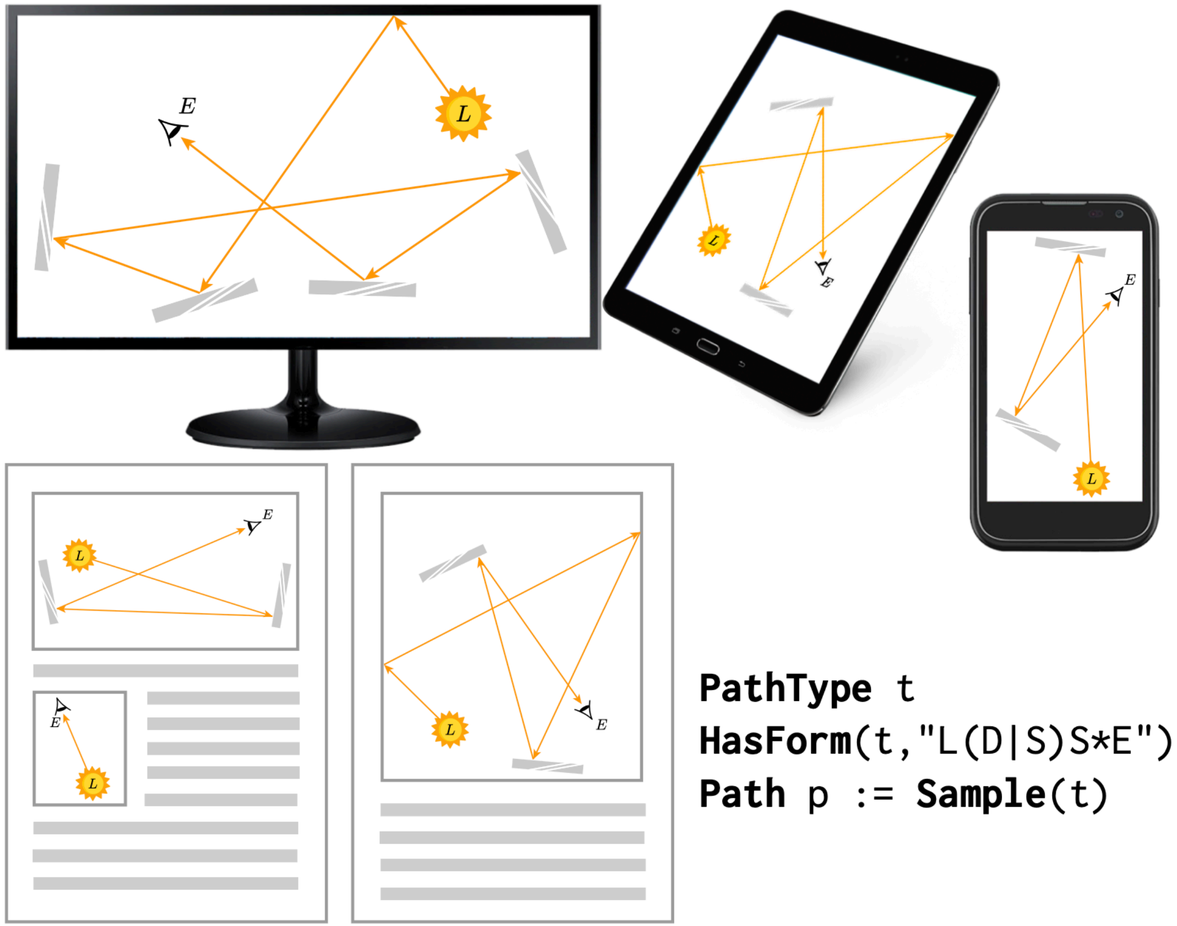
re-generating diagrams for a target platform




```
PathType t
HasForm(t,"L(D|S)S*E")
Path p := Sample(t)
```

Visualizing rays

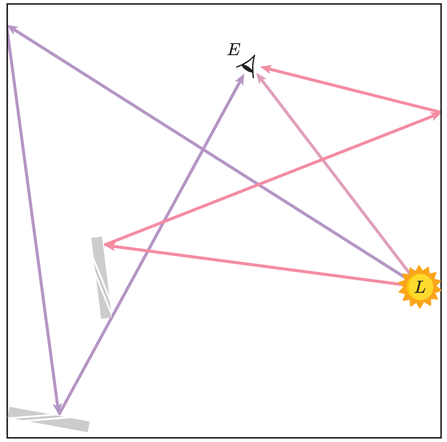
re-generating diagrams for a target platform

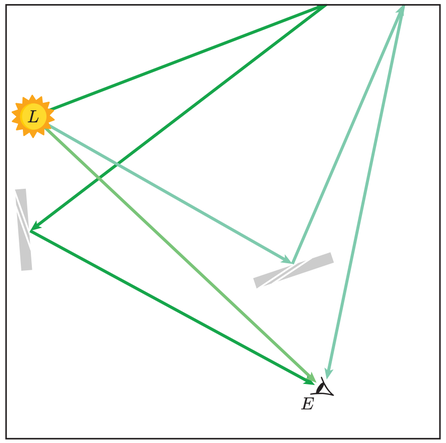


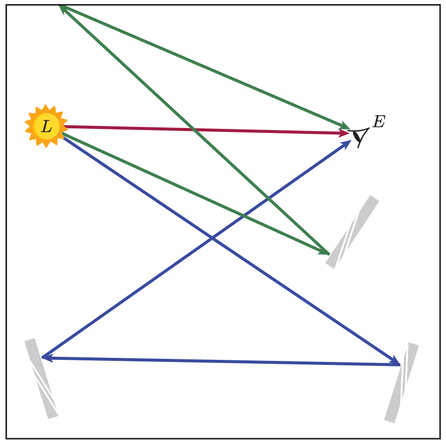

```
PathType t
HasForm(t,"L(D|S)S*E")
Path p := Sample(t)
```

handling a tricky specification

```
PathType t1, t2, t3
HasForm(t1, "L(D|S)SE")
HasForm(t2, "LSDE")
HasForm(t3, "LE")

Path p1, p2, p3
p1 := Sample(t1)
p2 := Sample(t2)
p3 := Sample(t3)
```

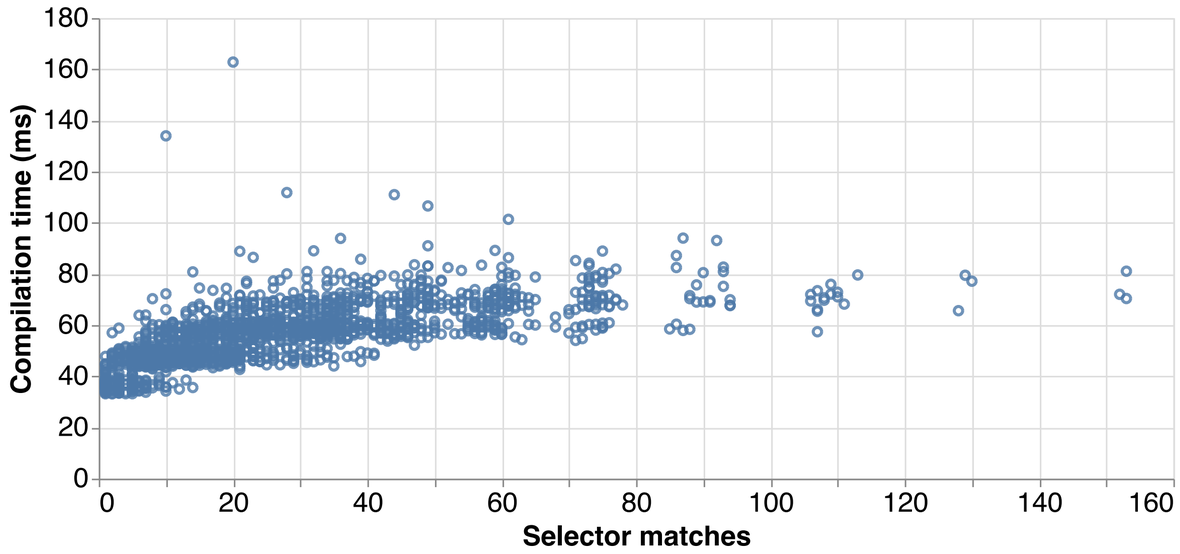




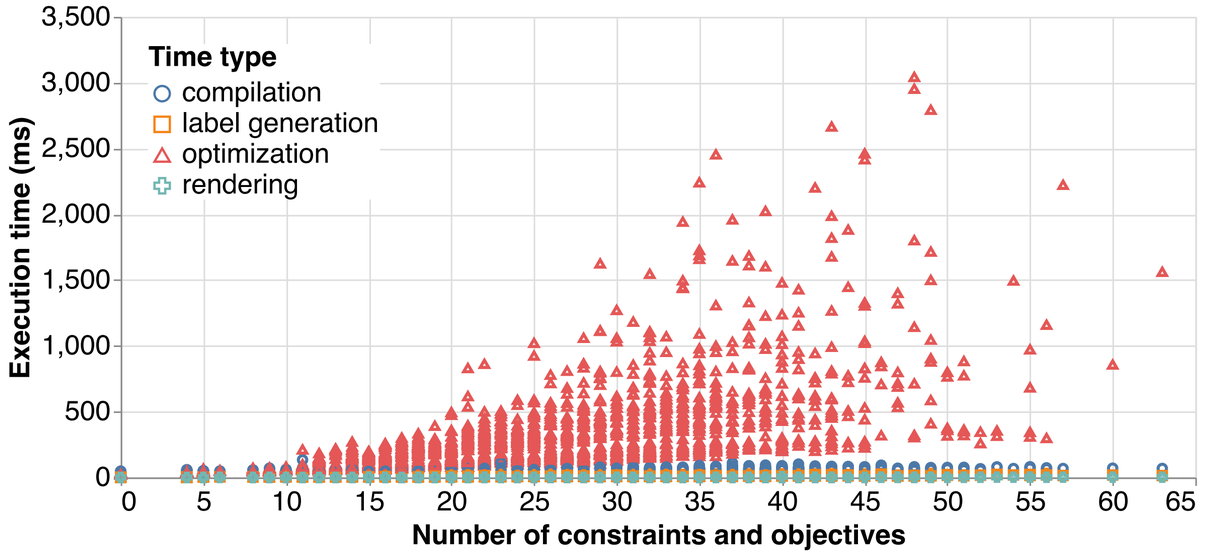


Performance scaling

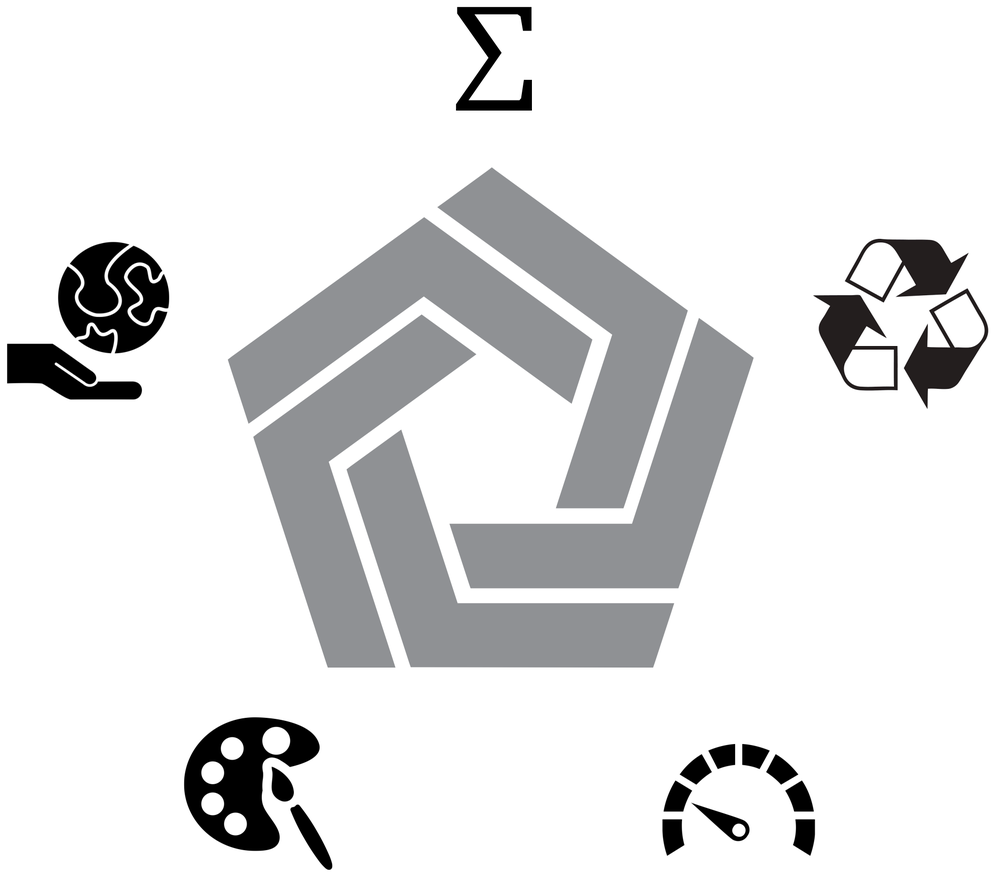
compilation time vs. program size



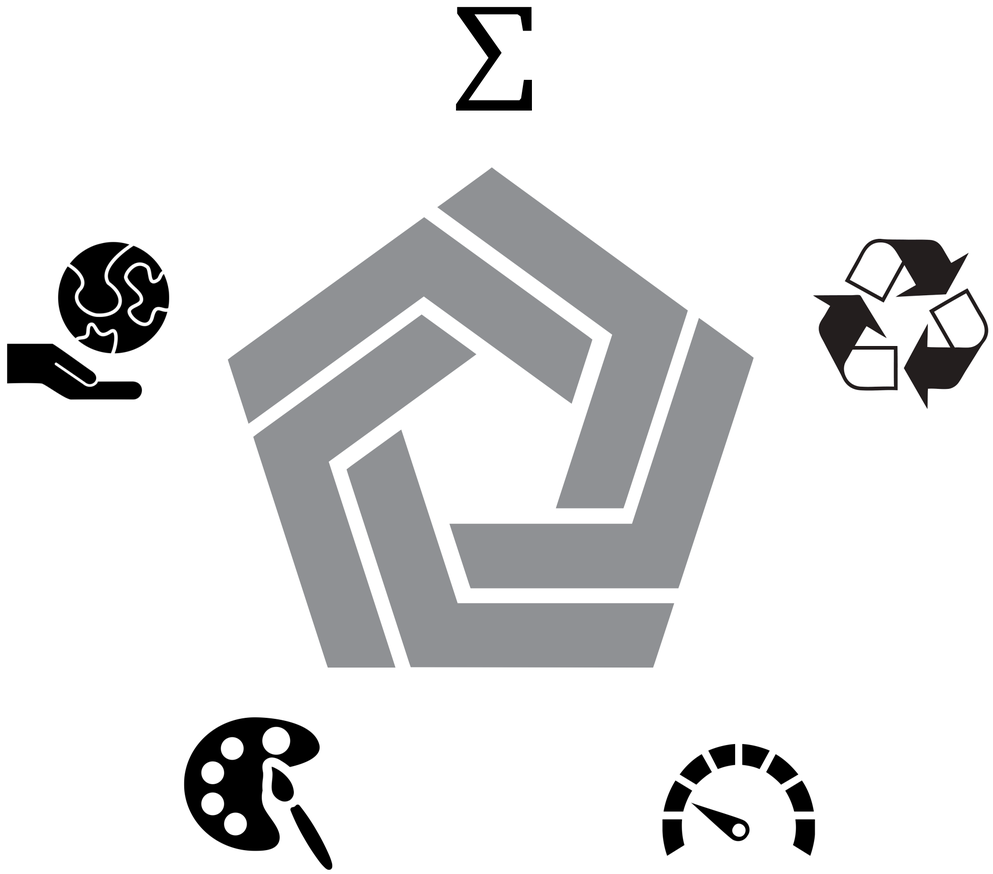
optimization time vs. problem size



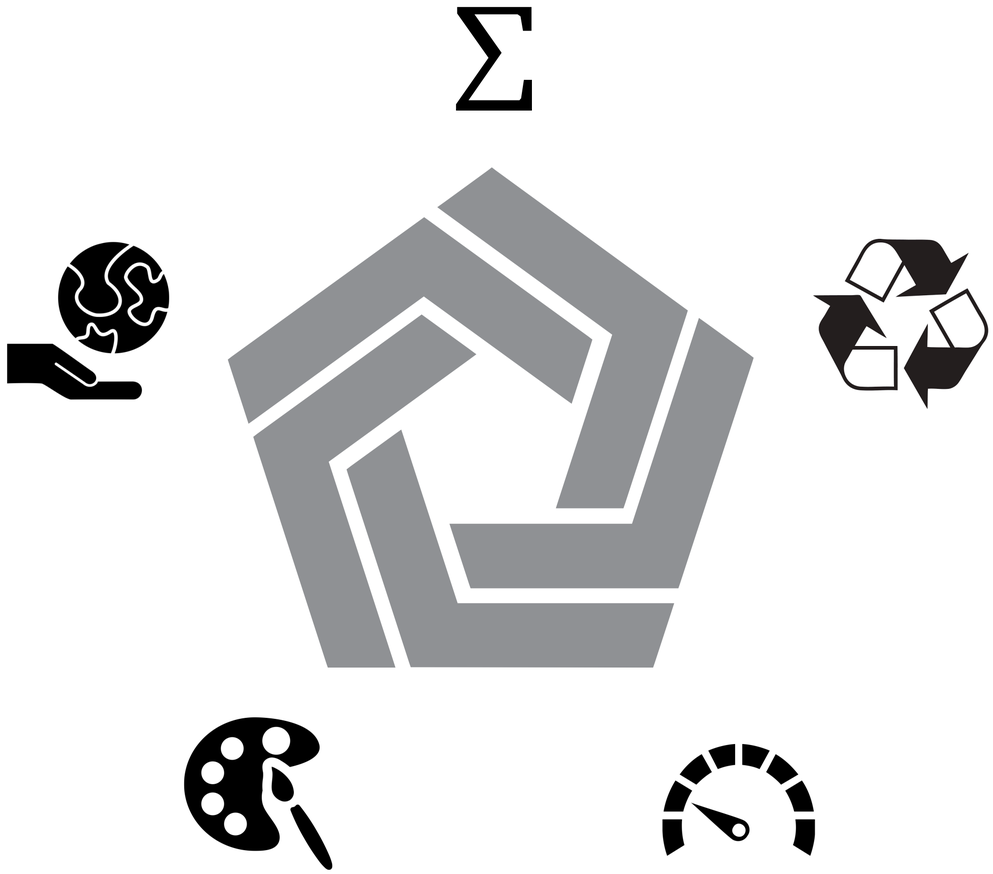
Insights



Insights

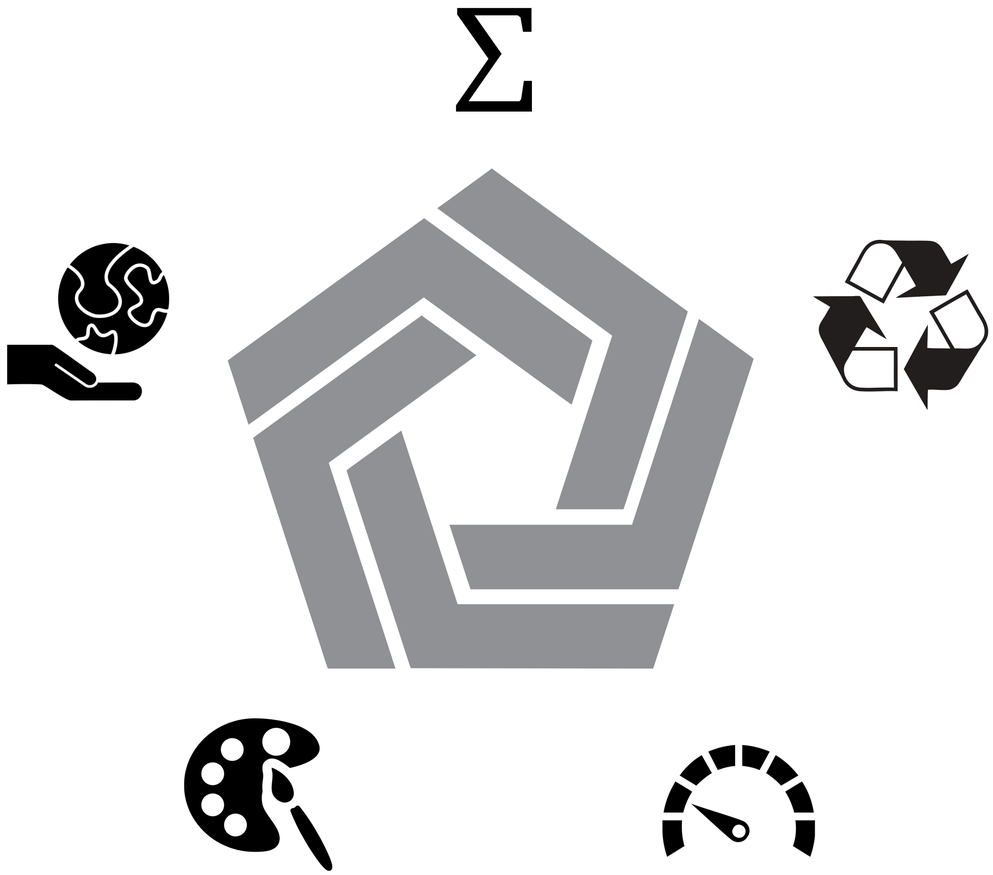


Insights



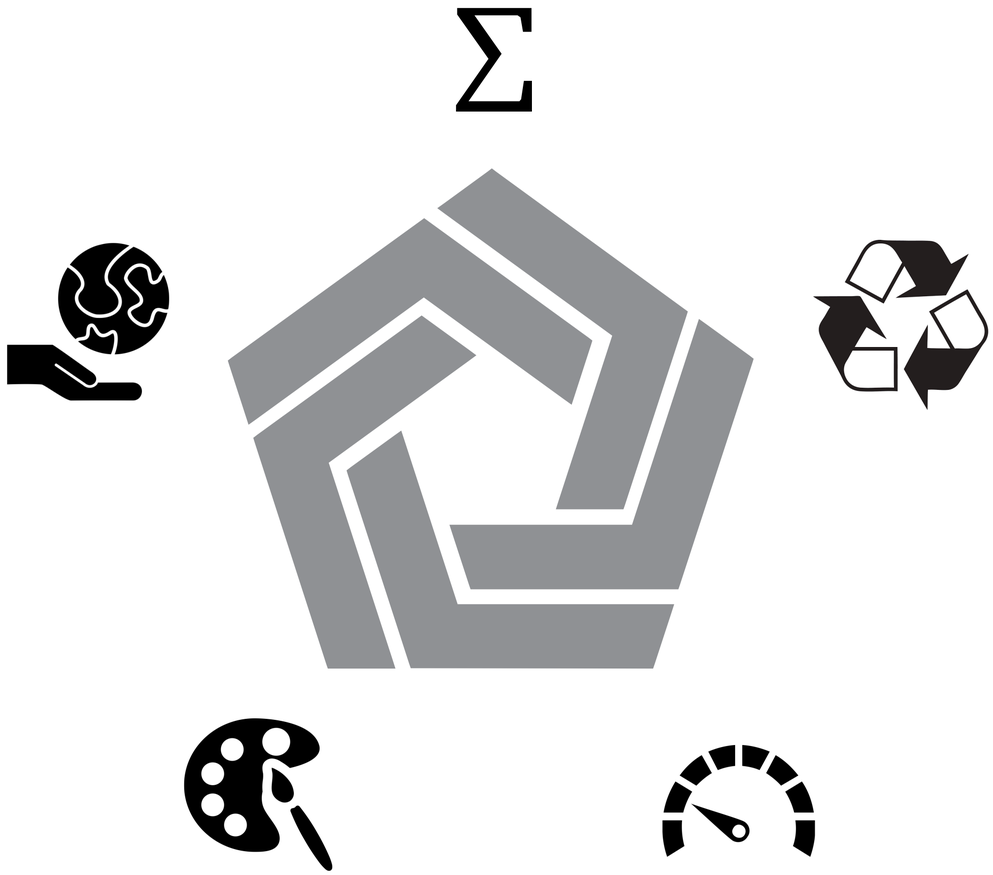
Insights

- Separating content and visual representation provides new capabilities.



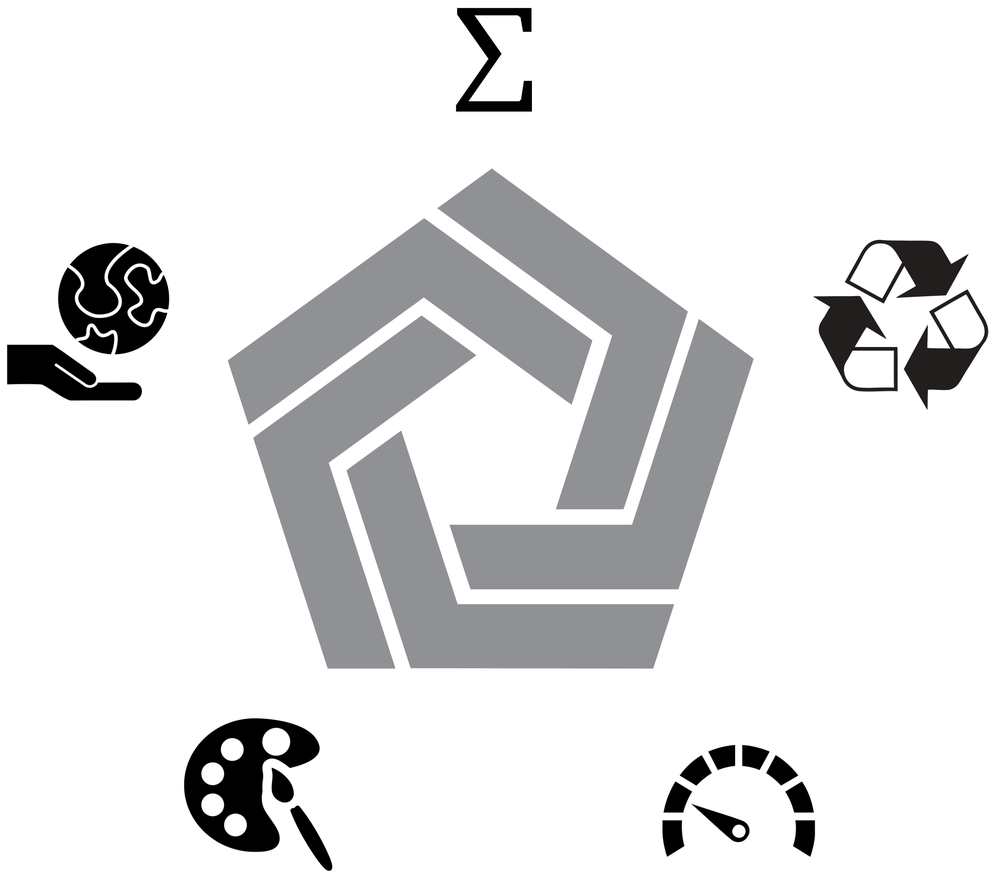
Insights

- Separating content and visual representation provides new capabilities.
- The mapping from abstract objects to a visual representation provides rich semantics.



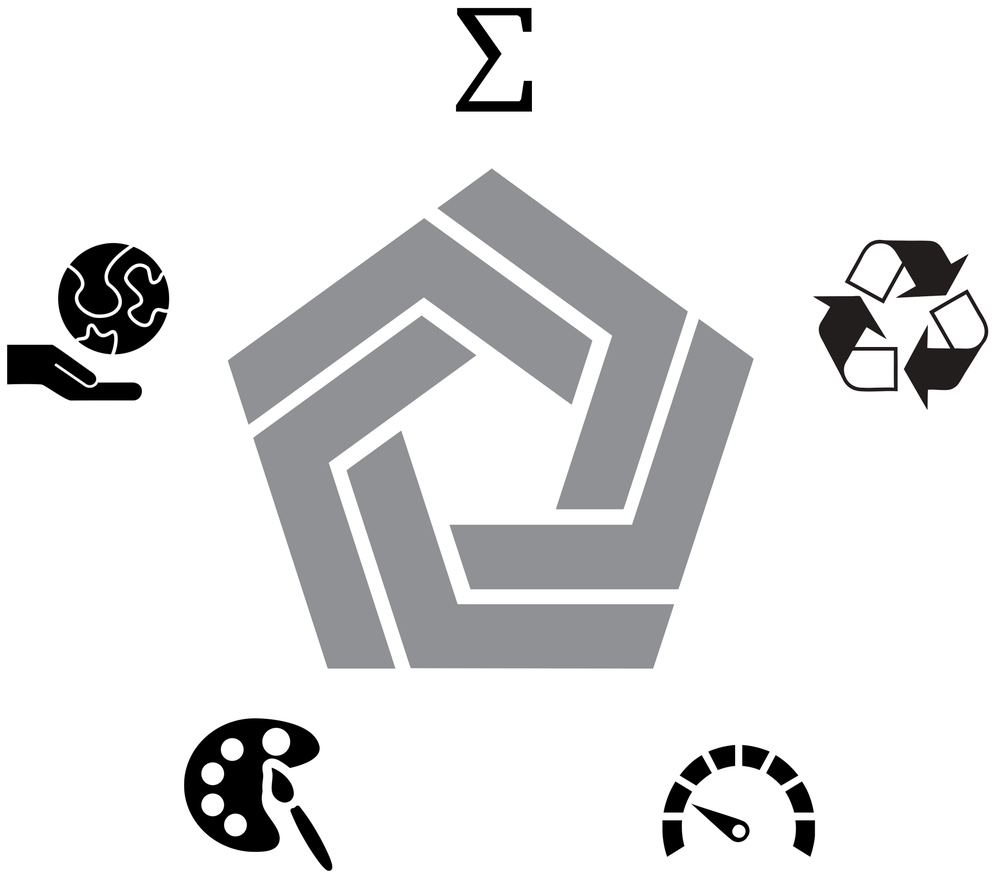
Insights

- Separating content and visual representation provides new capabilities.
- The mapping from abstract objects to a visual representation provides rich semantics.
- Constrained optimization can be an effective general-purpose solution for diagram layout.



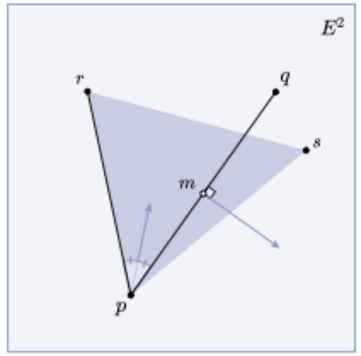
Insights

- Separating content and visual representation provides new capabilities.
- The mapping from abstract objects to a visual representation provides rich semantics.
- Constrained optimization can be an effective general-purpose solution for diagram layout.
- Penrose acts as a nexus for diagram generation, connecting **specification** and **synthesis**.

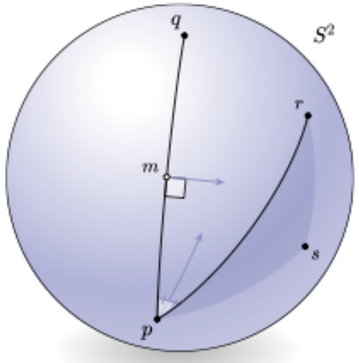


We want to make diagrams like these the norm, not the exception!

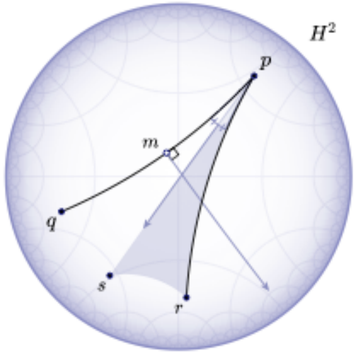
We want to make diagrams like these the norm, not the exception!



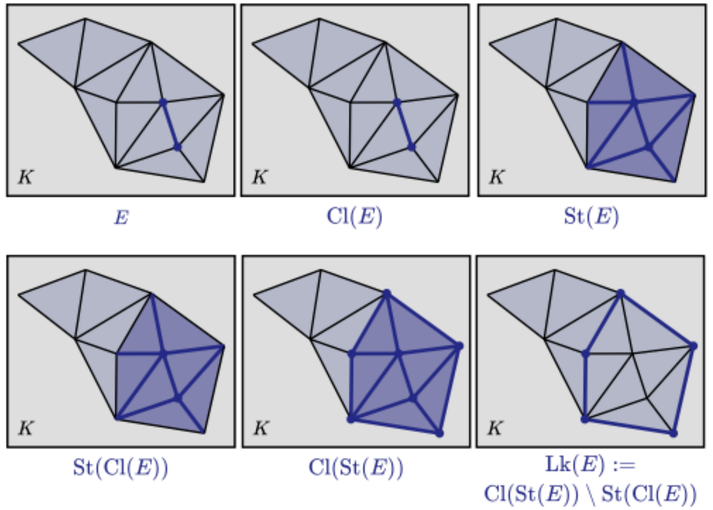
STYLE — Euclidean

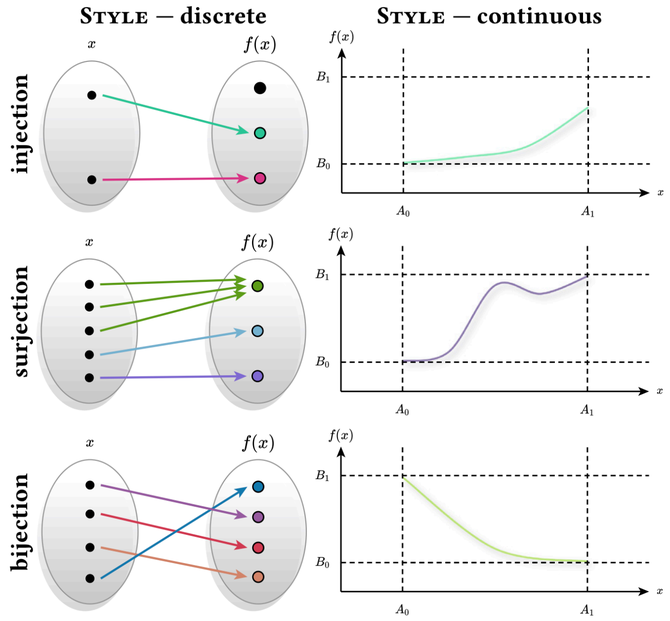


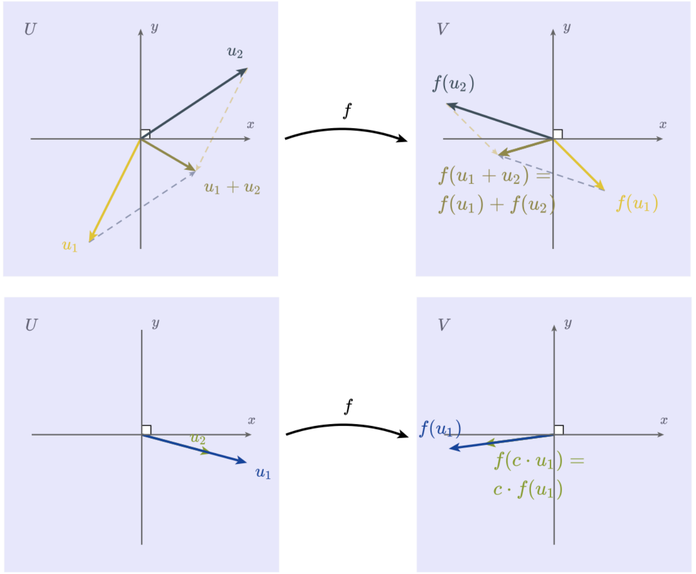
STYLE — spherical



STYLE — hyperbolic







Thanks!

https://penrose.ink