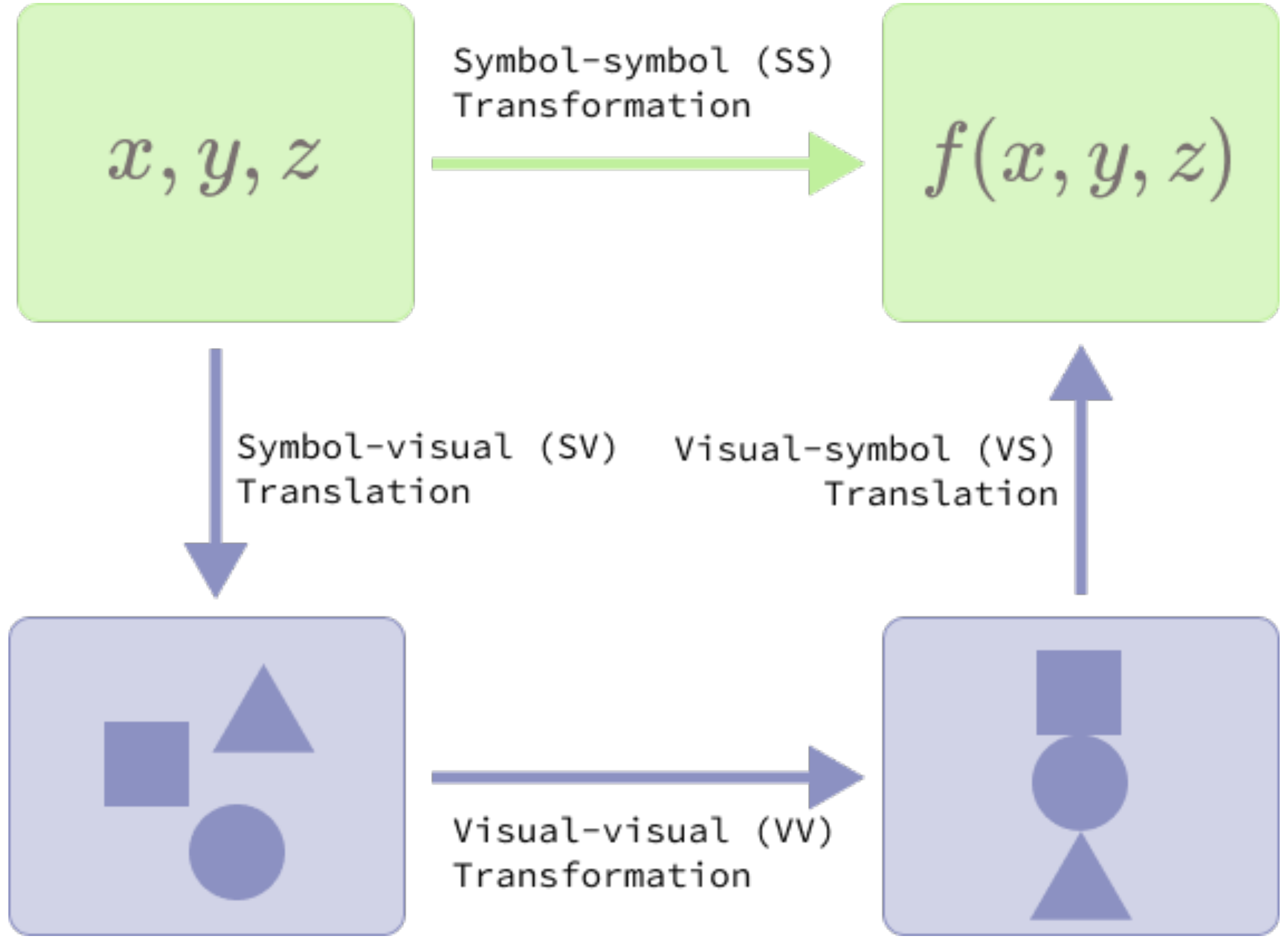
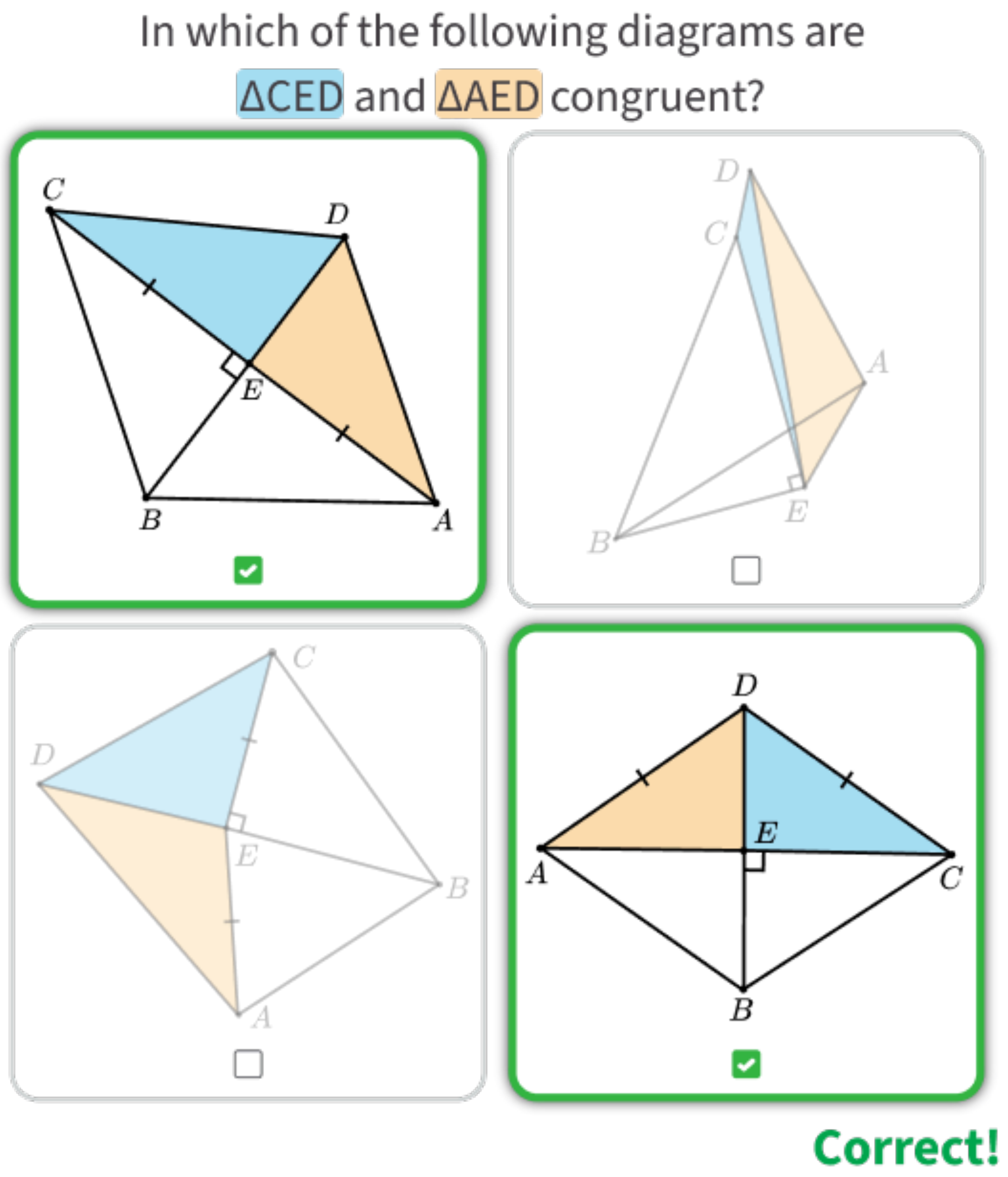
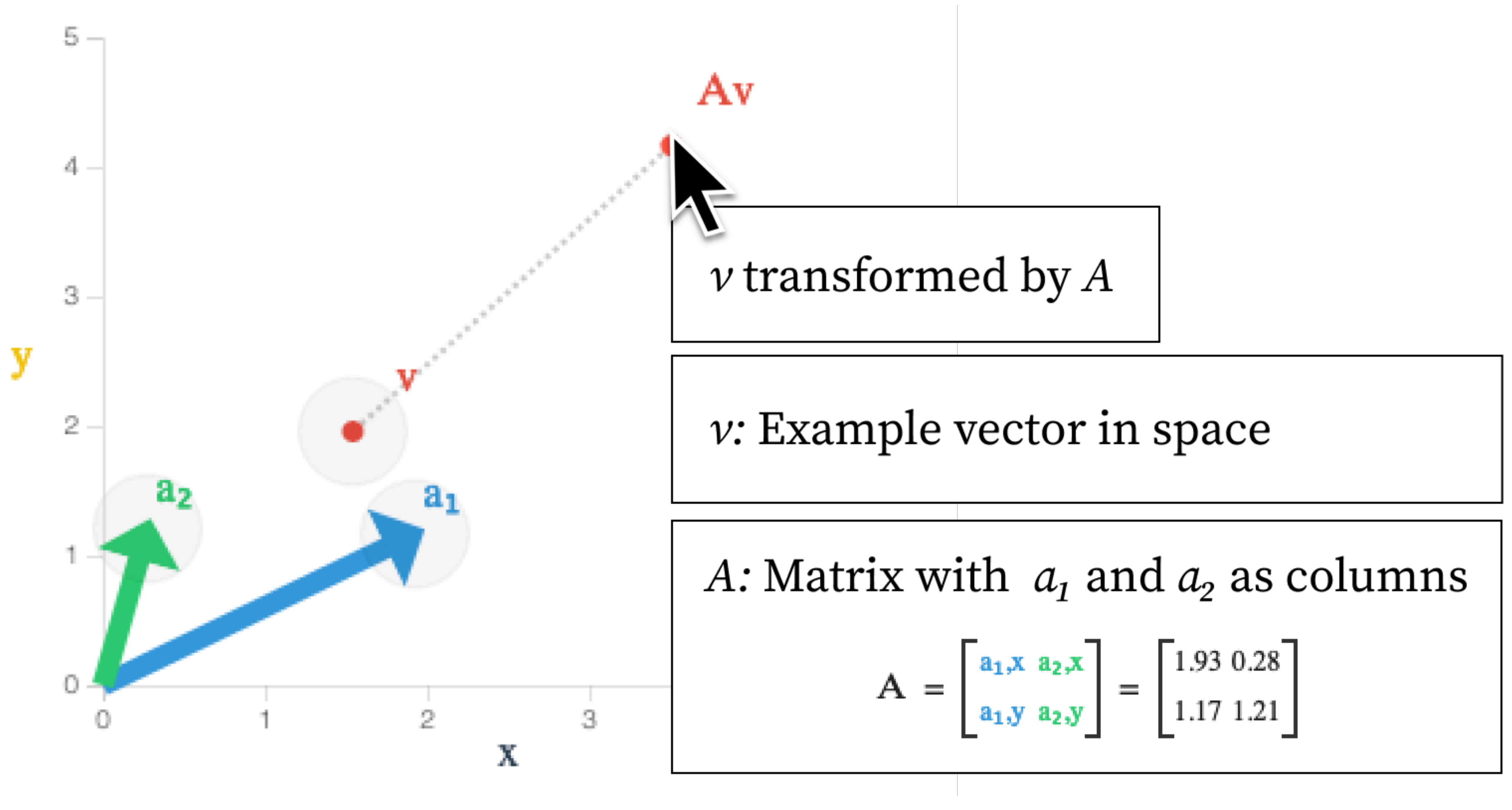






Developing conceptual understanding through interactive diagramming

Wode “Nimo” Ni

What's sin(-π/6)?

$$sin(\frac{\pi}{6}) = \frac{1}{2}$$

$$sin(-\frac{\pi}{6}) =?$$

Pushing symbols: trig identities

$$sin(\frac{\pi}{6}) = \frac{1}{2}$$

$$sin(-\frac{\pi}{6}) = ?$$

$$sin(0-\theta) = -sin(\theta)$$

$$sin(-\frac{\pi}{6}) = -sin(\frac{\pi}{6})$$
$$= -\frac{1}{2}$$

$\sin(0-\theta) = -\sin\theta$	$\sin(\frac{\pi}{2}-\theta) = +\cos\theta$	$\sin(\pi-\theta) = +\sin\theta$	$\sin(\frac{3\pi}{2}-\theta) = -\cos\theta$
$\cos(0-\theta) = +\cos\theta$	$\cos(\frac{\pi}{2}-\theta) = +\sin\theta$	$\cos(\pi-\theta) = -\cos\theta$	$\cos(\frac{3\pi}{2}-\theta) = -\sin\theta$
$\tan(0-\theta) = -\tan\theta$	$\tan(\frac{\pi}{2}-\theta) = +\cot\theta$	$\tan(\pi-\theta) = -\tan\theta$	$\tan(\frac{3\pi}{2}-\theta) = +\cot\theta$

A hint

$$sin(\frac{\pi}{6}) = \frac{1}{2}$$

$$sin(-\frac{\pi}{6}) = ?$$

$$sin(0 - \theta) = -sin(\theta)$$

$$sin(-\frac{\pi}{6}) = -sin(\frac{\pi}{6})$$
$$= -\frac{1}{2}$$

“奇变偶不变 符号看象限” — *Chinese proverb*

Look at the quadrants?

$$sin(\frac{\pi}{6}) = \frac{1}{2}$$

$$sin(-\frac{\pi}{6}) = ?$$

$$sin(0 - \theta) = -sin(\theta)$$

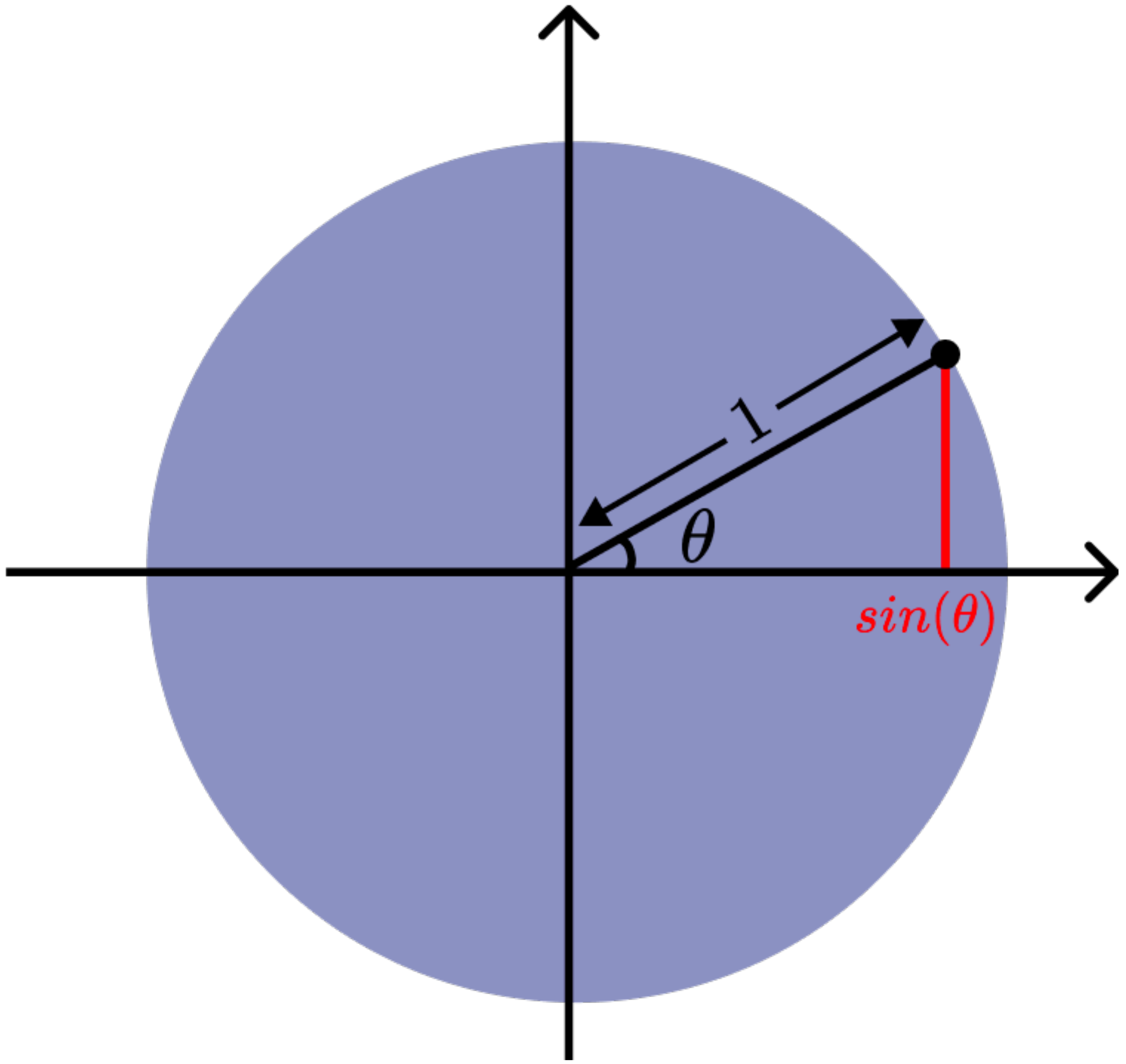
$$sin(-\frac{\pi}{6}) = -sin(\frac{\pi}{6}) = -\frac{1}{2}$$

Nimo's middle-school teacher

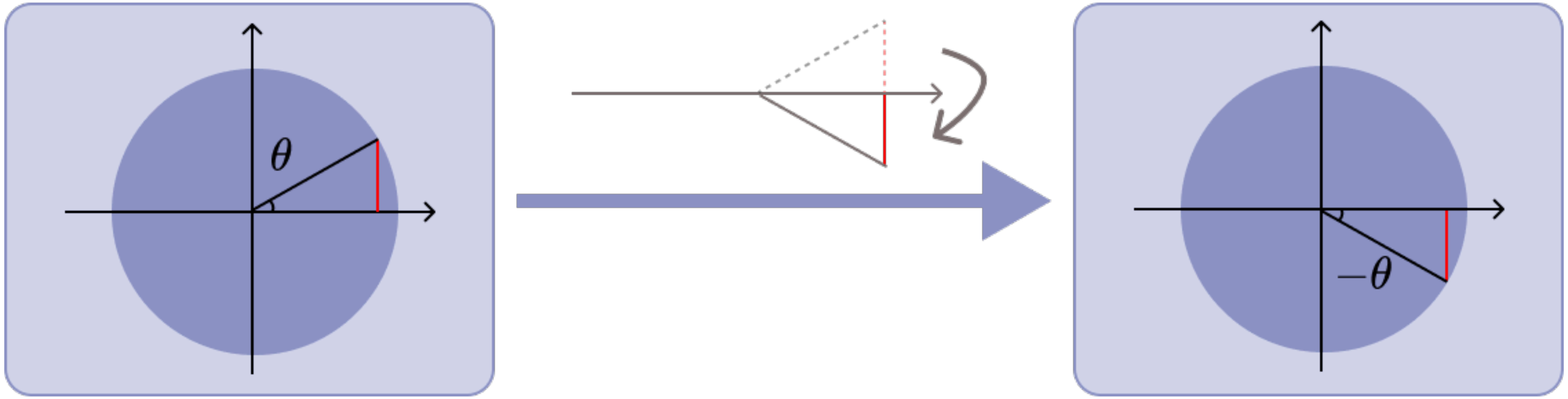
“奇变偶不变 符号看象限” — ~~Chinese proverb~~

“Look at the quadrants for the sign”

A visual representation: the unit circle

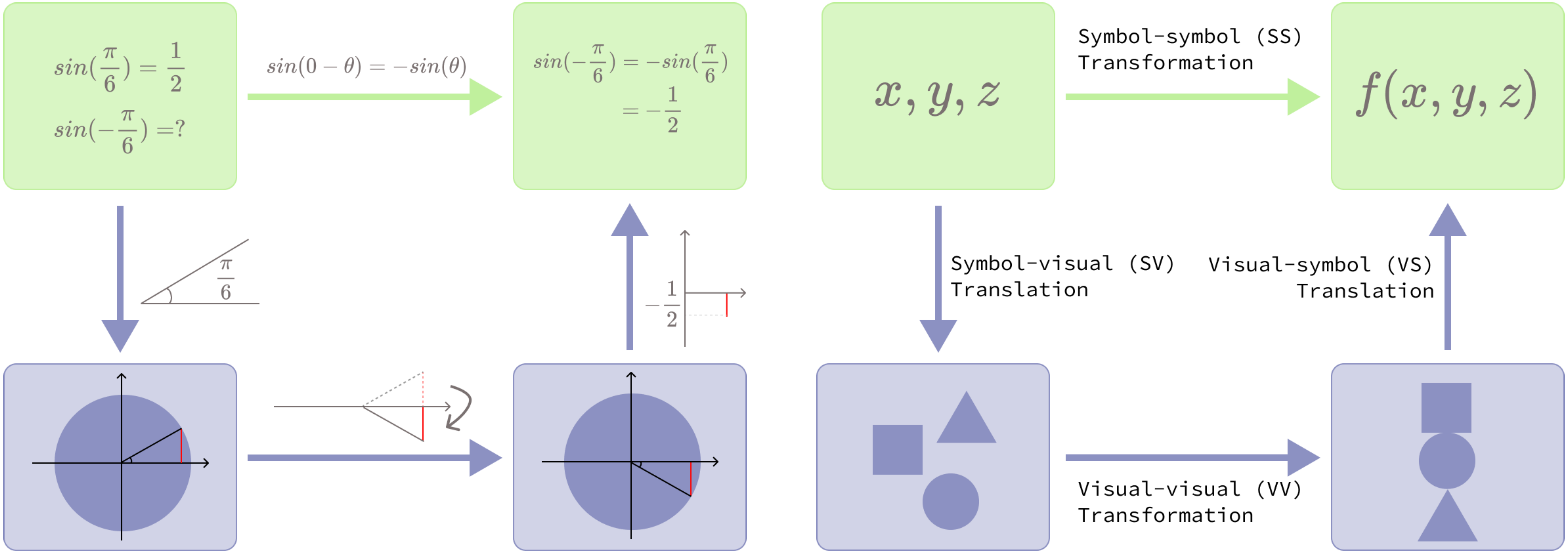


Solving for sin(-π/6) visually

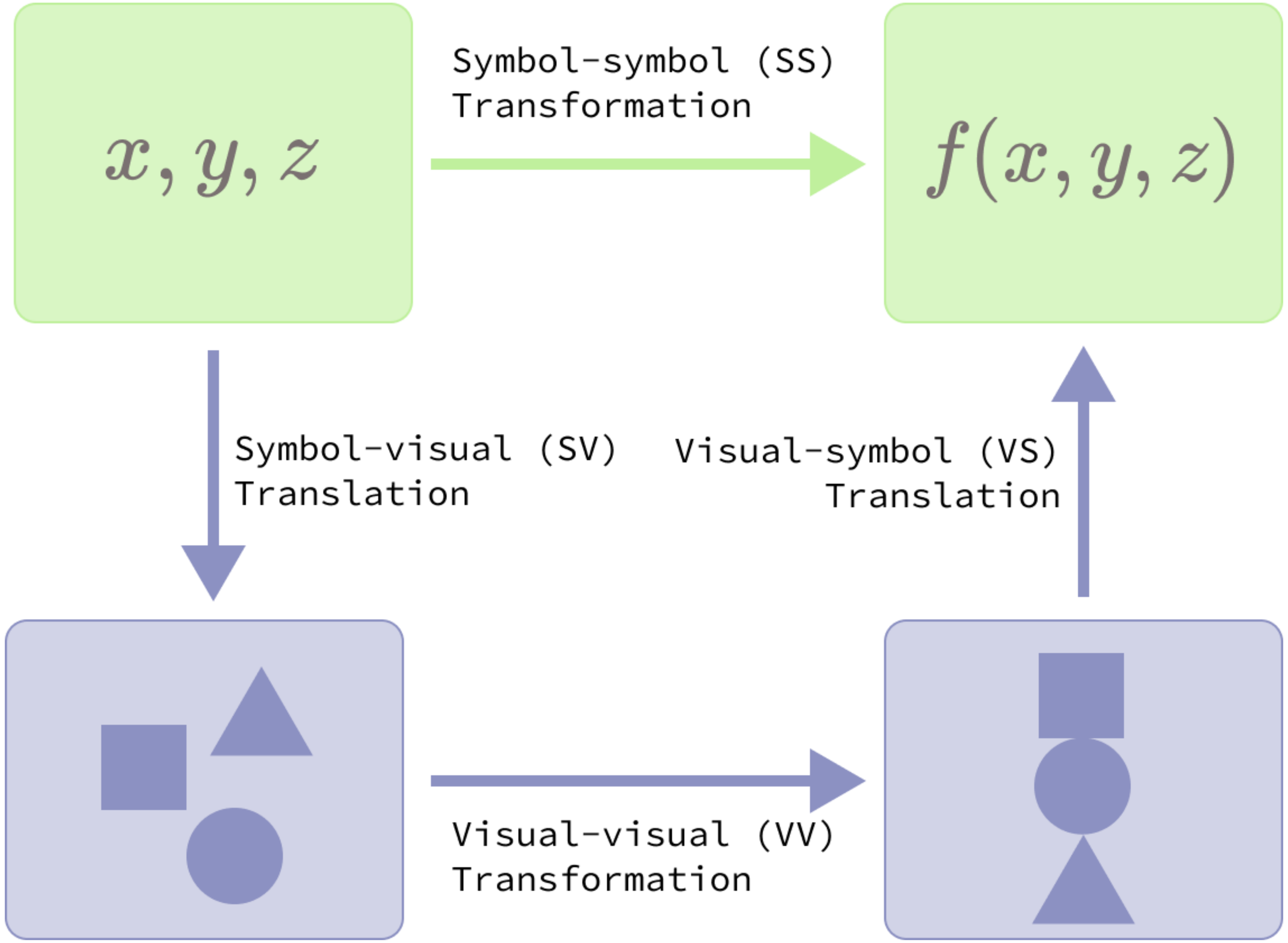


“Look at the quadrants for the sign”

Taking the alternative path



The Grounding Rectangle



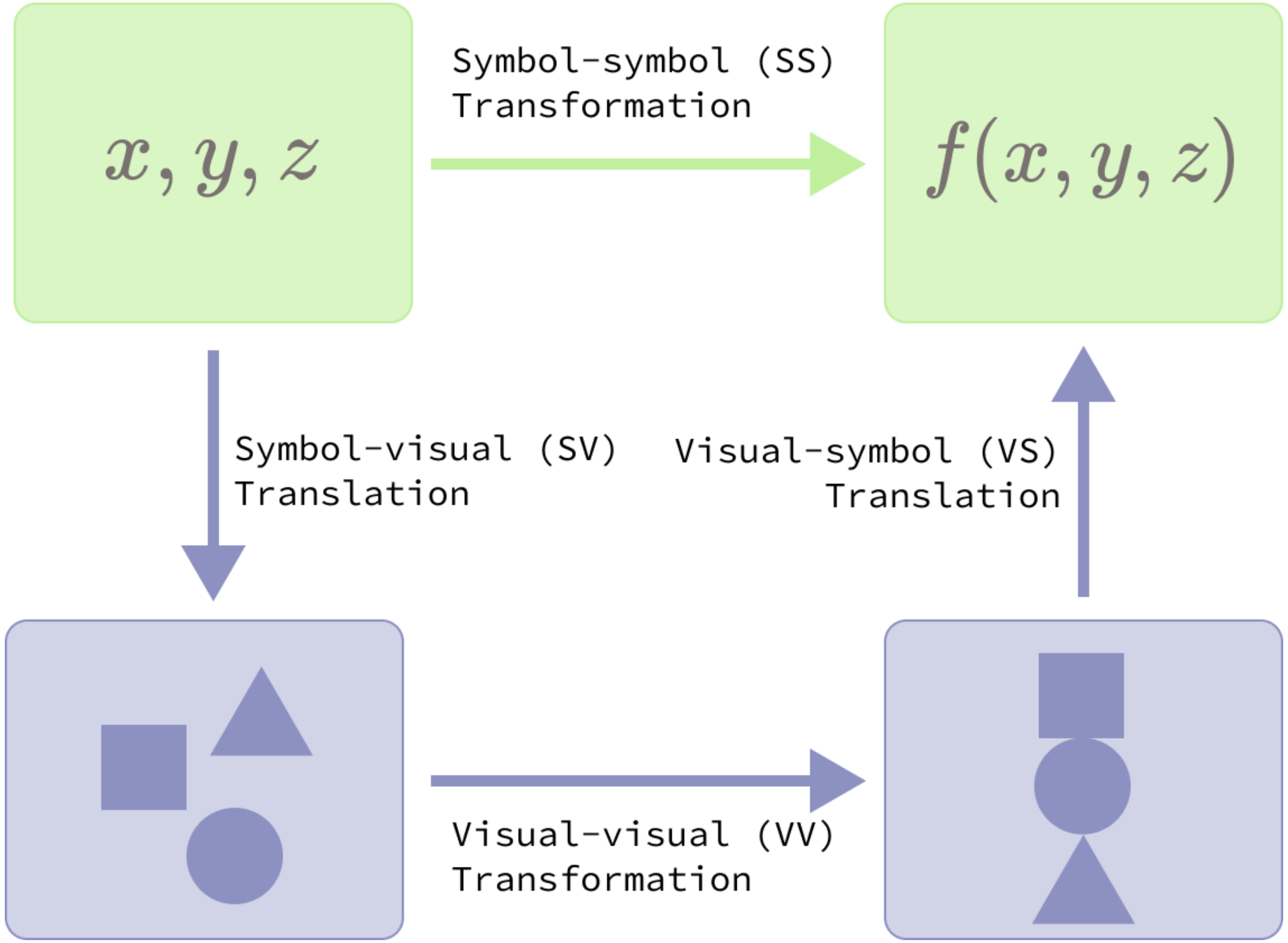
"The Multimedia Principle"

Mayer & Moreno, 2010

Visual+symbolic representations improve knowledge retention and transfer in problem solving.

Wiese. "Grounded Feedback." Rau. "Conceptual learning with multiple representations."

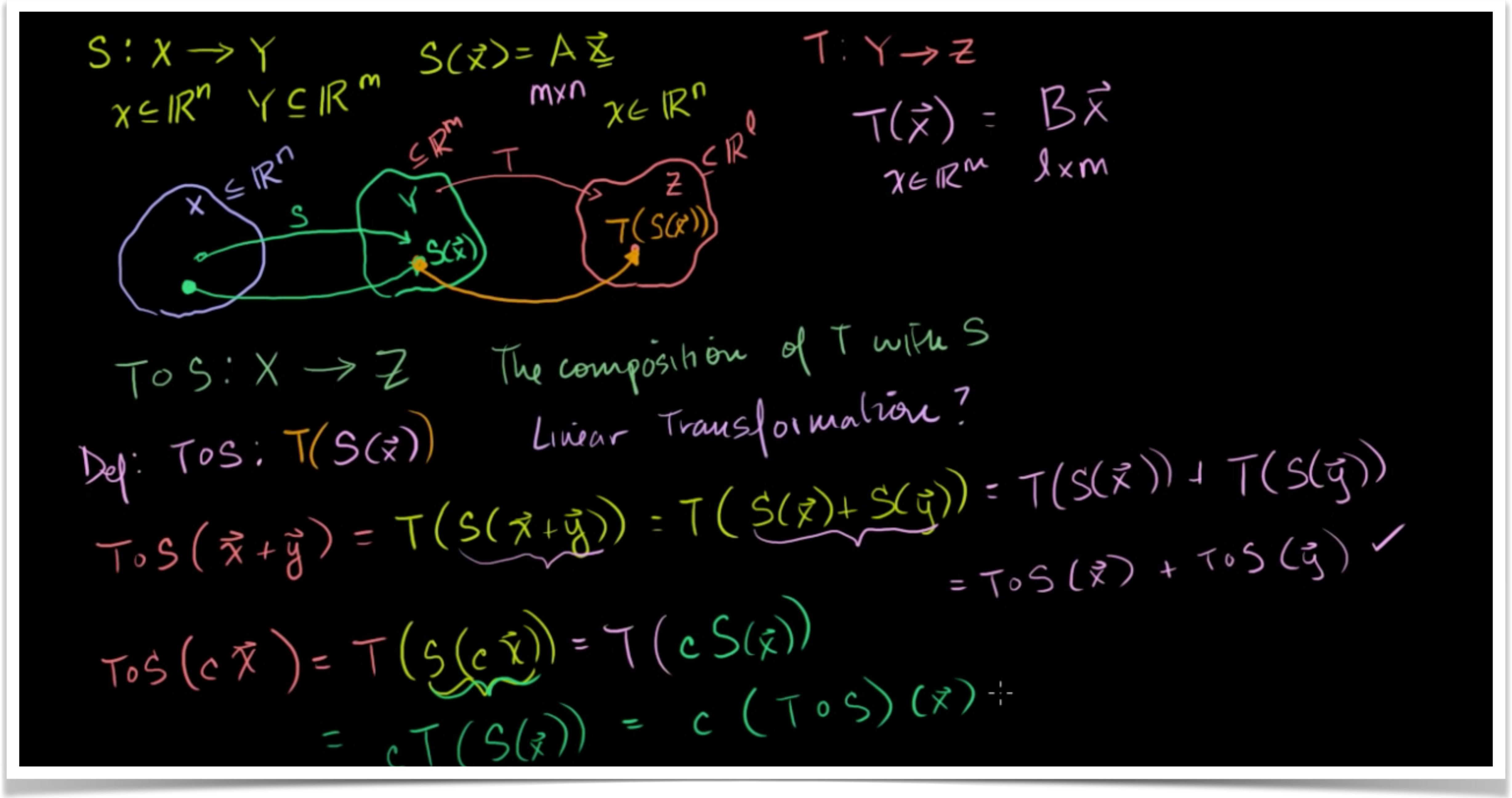
The Grounding Rectangle



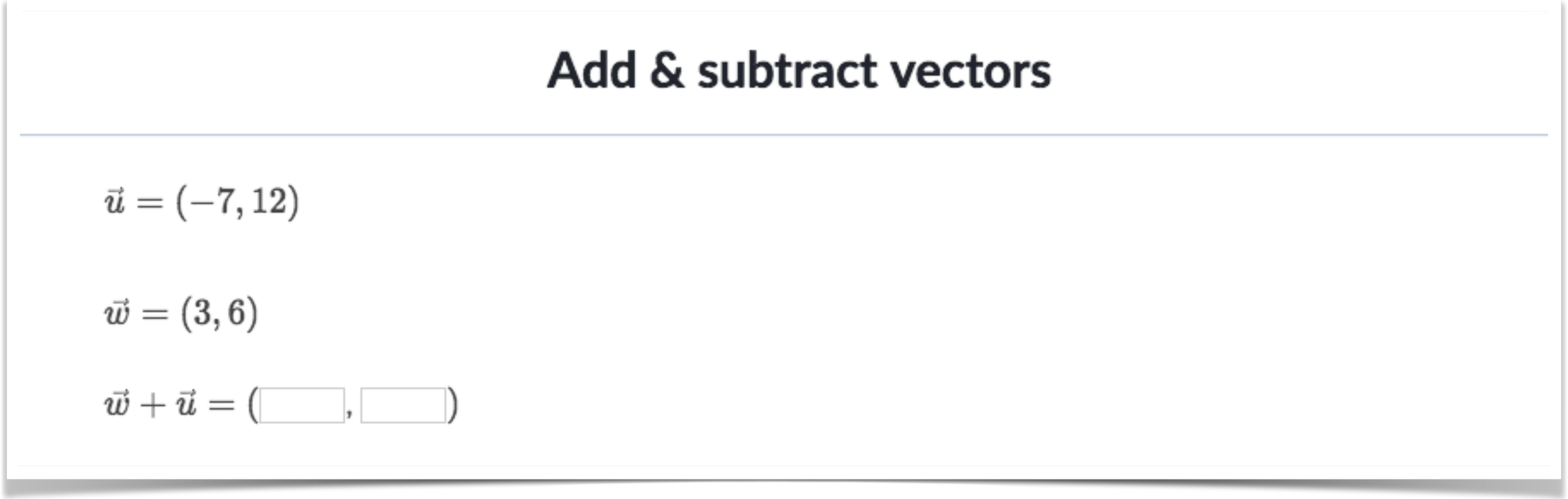
Needs explicit learning and practice: worked examples, feedback, spaced repetition…

Nice. How about some practice?

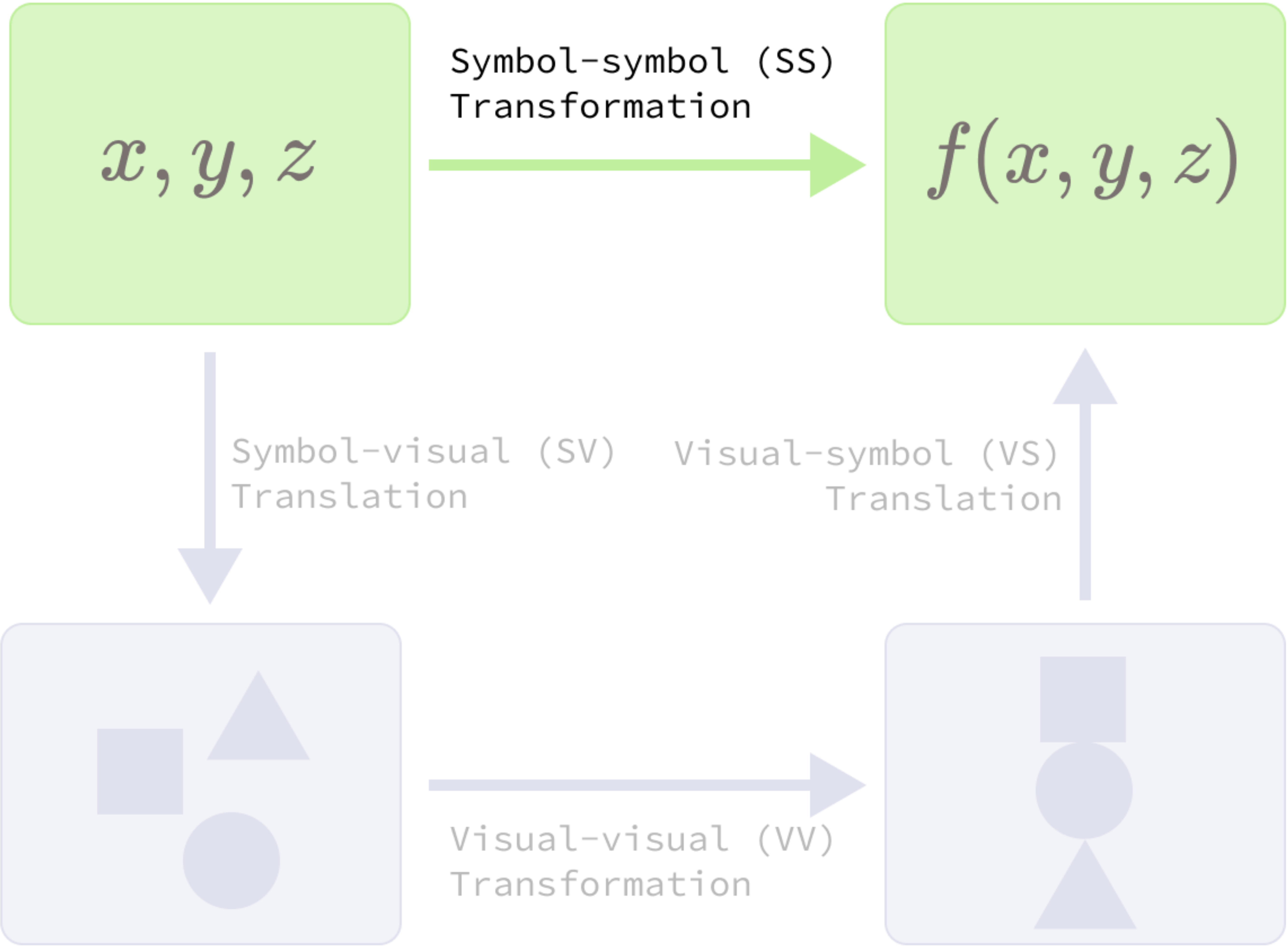
How it started



How it's going

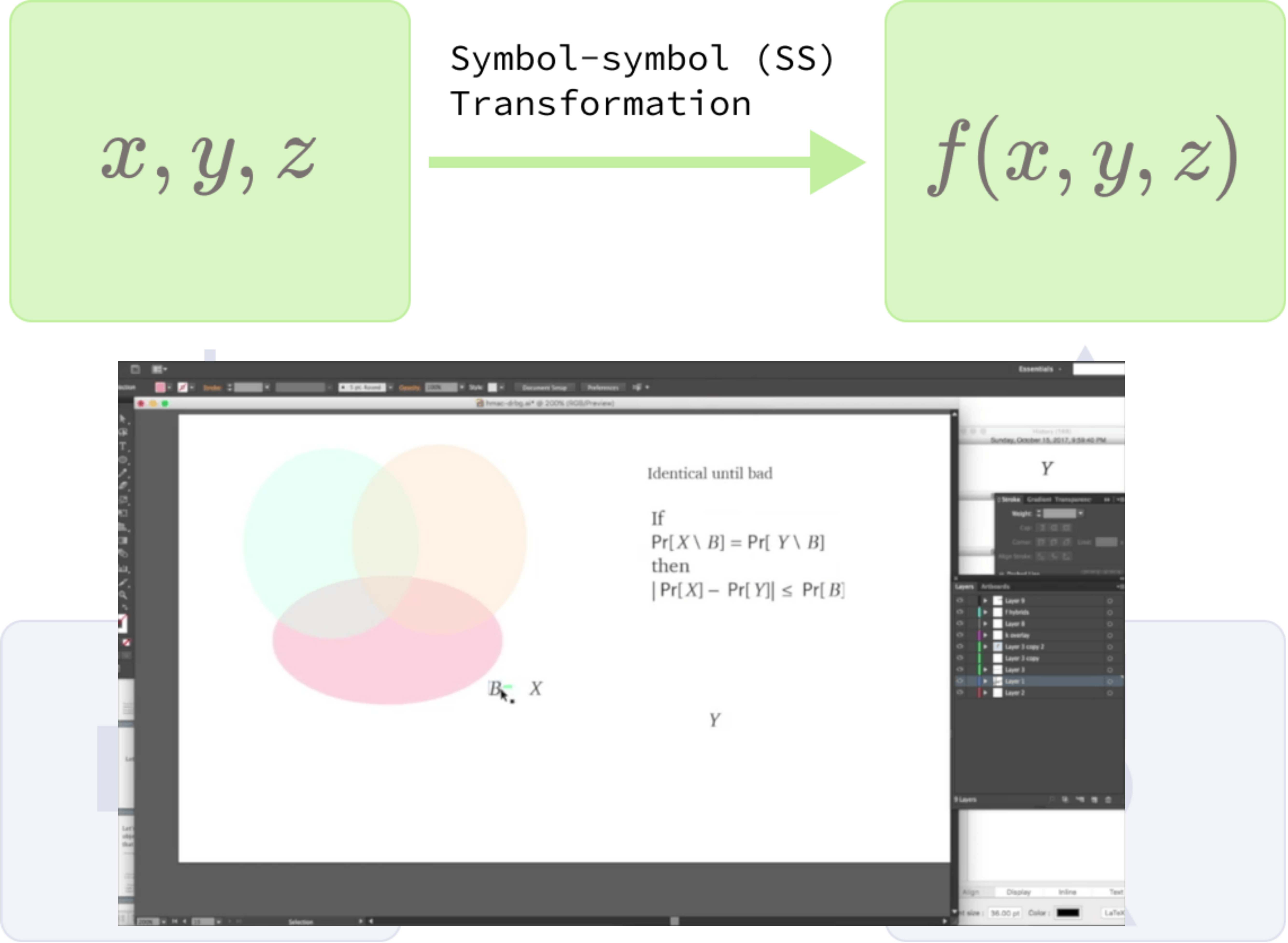


The missing path



Reality: “**Alternative path**” rarely practiced, mostly “symbol pushing”

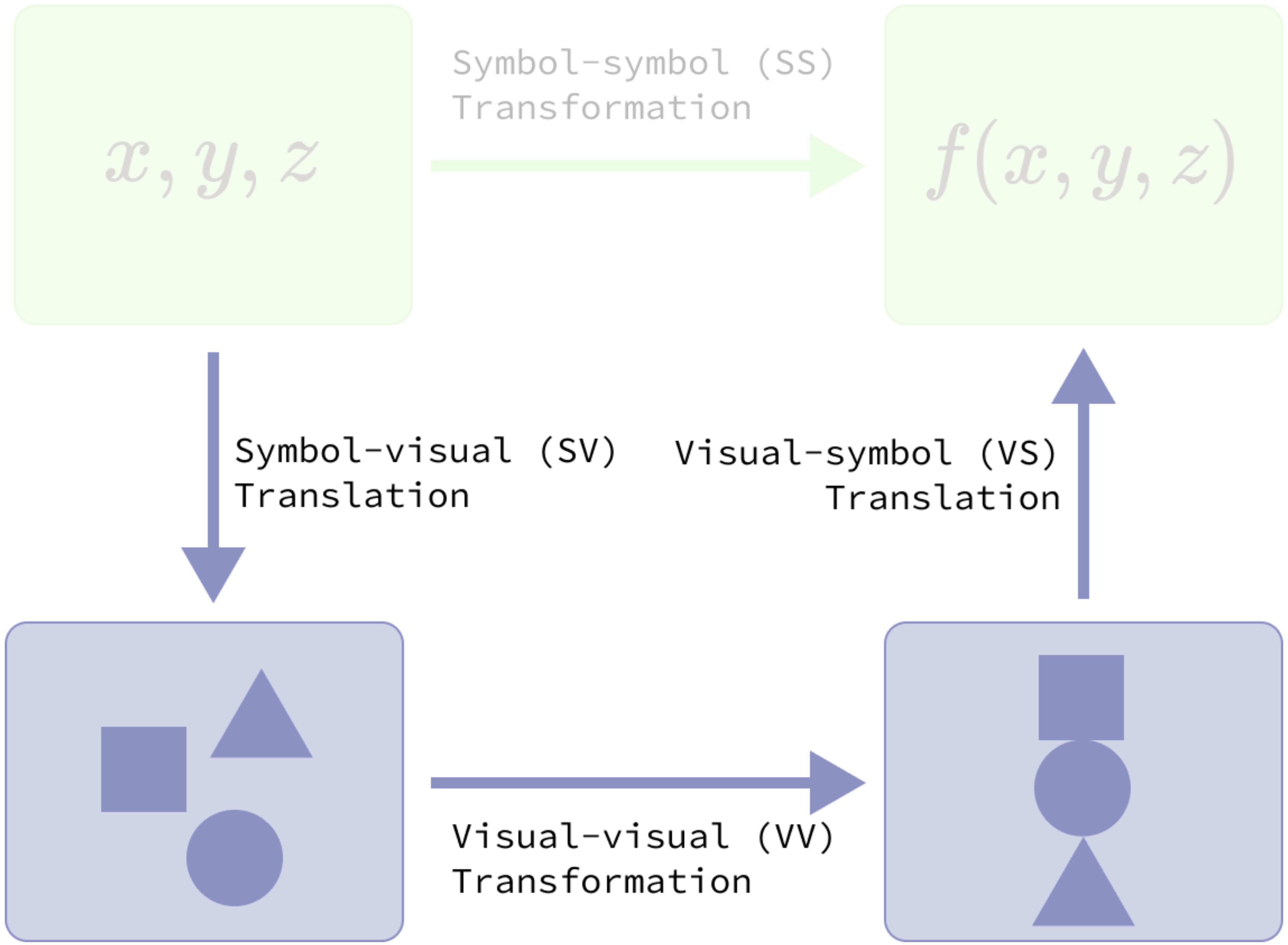
The supply problem of diagrams



Reality: "**Alternative path**" rarely practiced, mostly "symbol pushing"

Reason: diagrammatic contents are hard to make, maintain, and distribute

Need for better tools

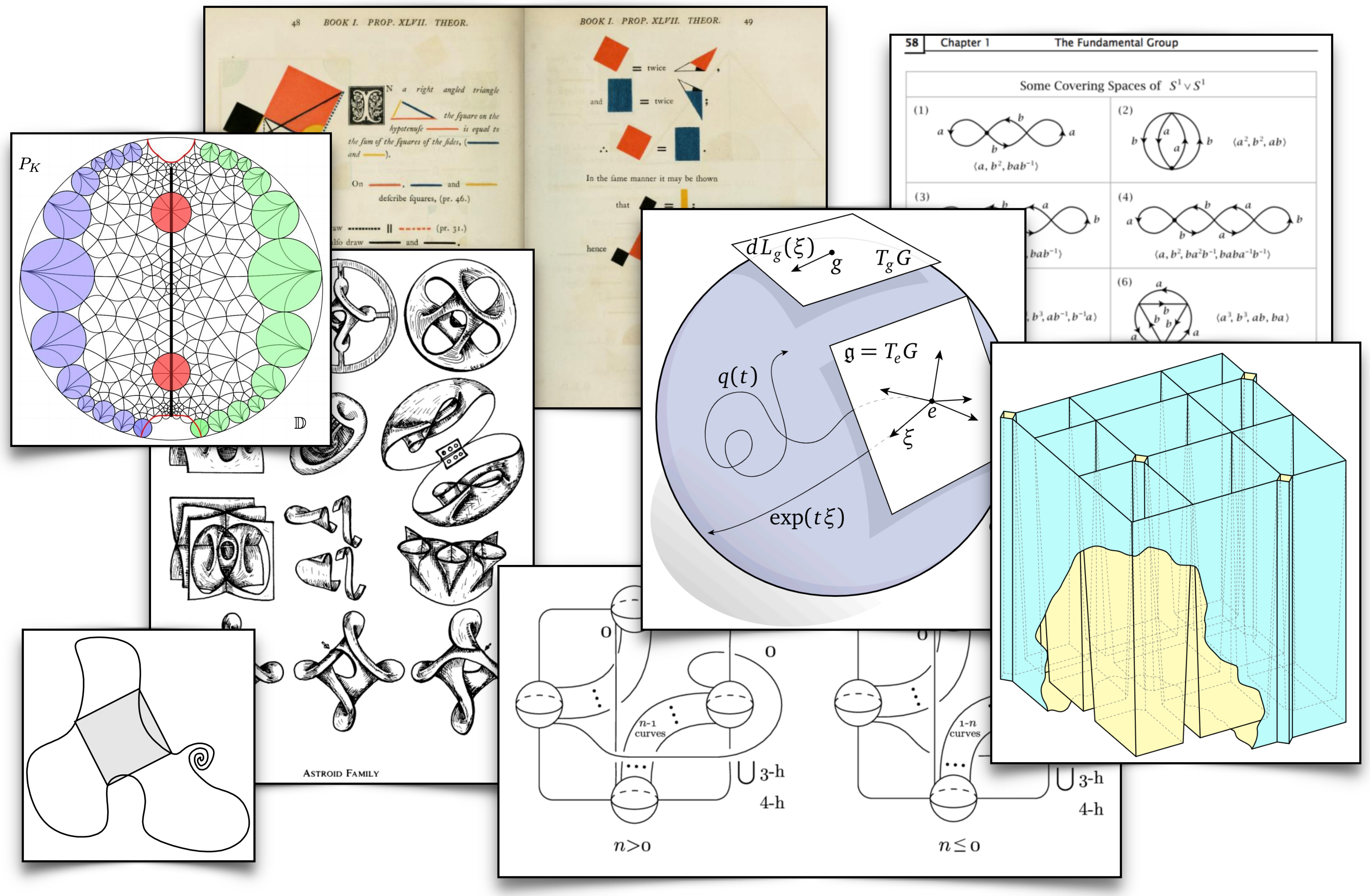


Reality: "Alternative path" rarely practiced, mostly "symbol pushing"

Reason: diagrammatic contents are hard to make, maintain, and distribute

Need: tools that (1) alleviate educators' manual effort in authoring diagrams and (2) facilitate visual thinking for students

Encoding visual representations in diagramming tools simplifies programming of interactive visual activities that provide students with automated feedback at scale.




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

.dsl

```
Set X { shape = Circle { } }
IsSubset(X, Y) {
  ensure contains(X.shape, Y.shape)
}
```

.sty

```
Set A, B
IsSubset(B, A)
```

.sub

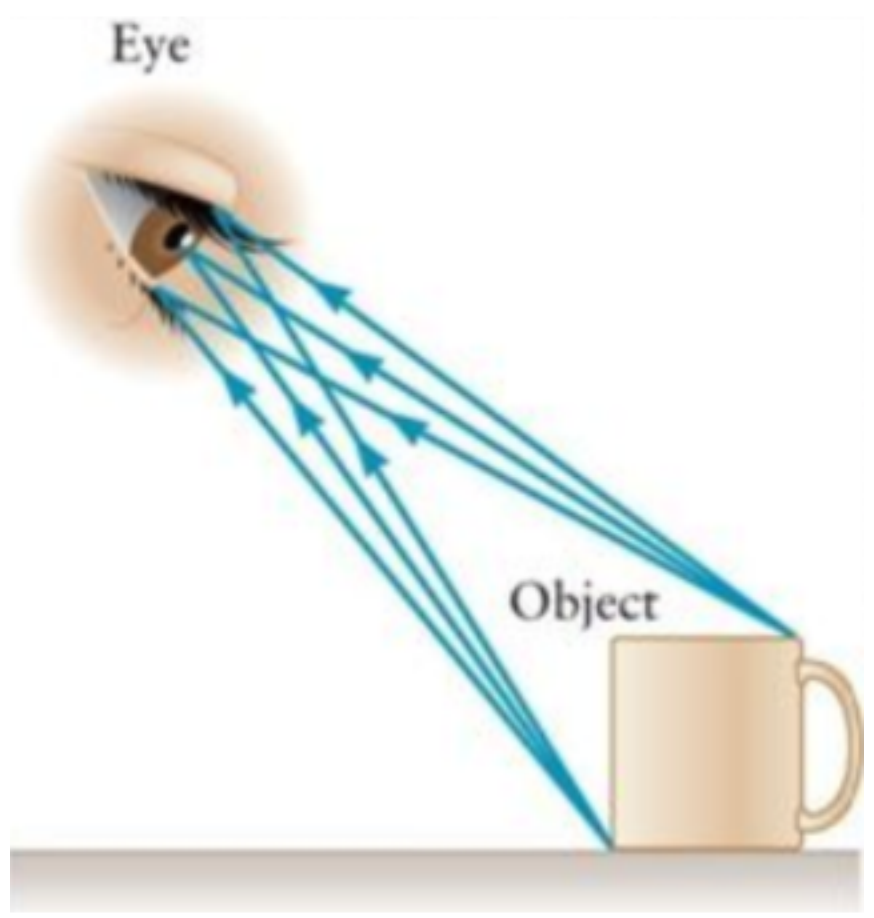
Understanding the diagramming process and encoding visual representations

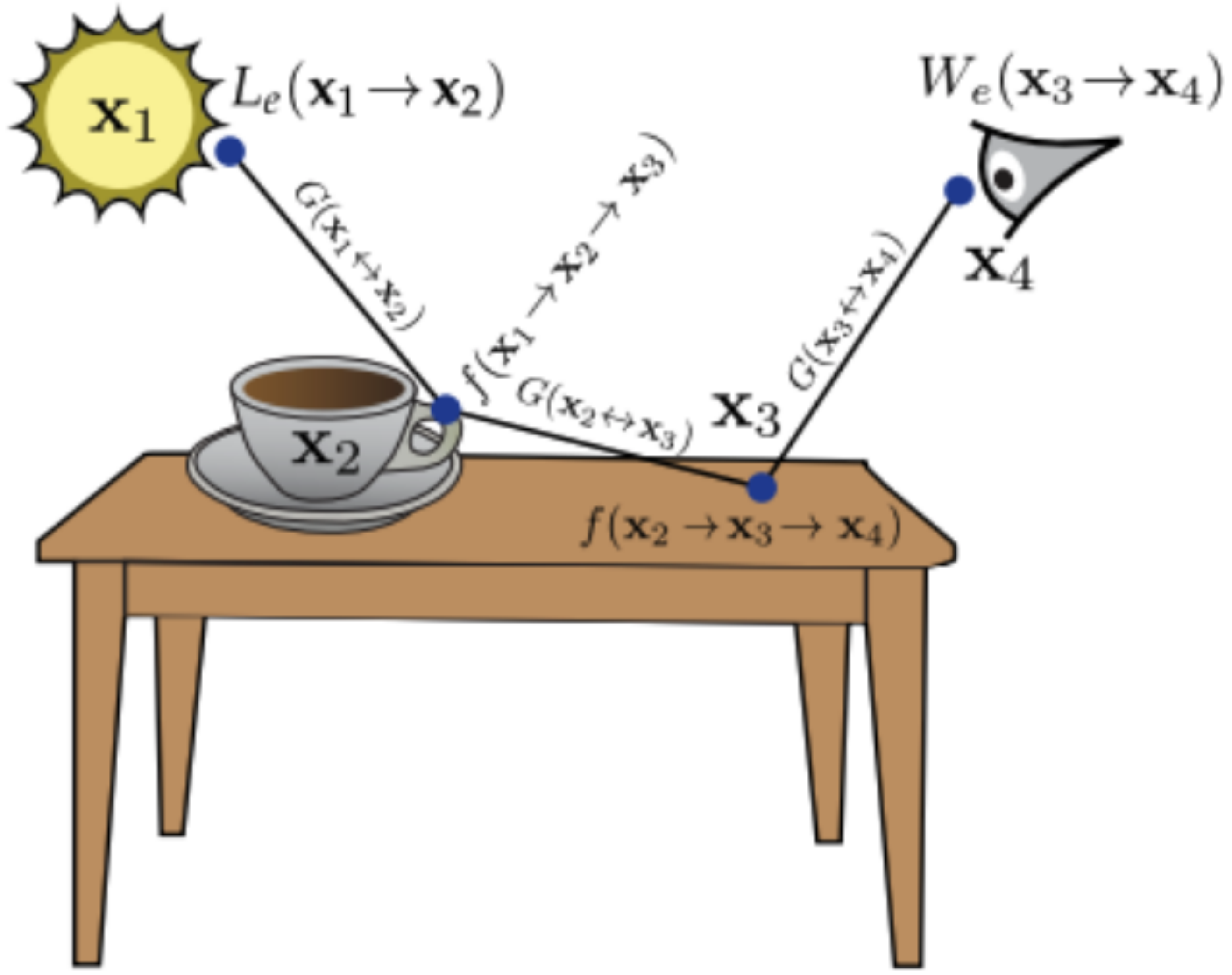
How do people create diagrams?

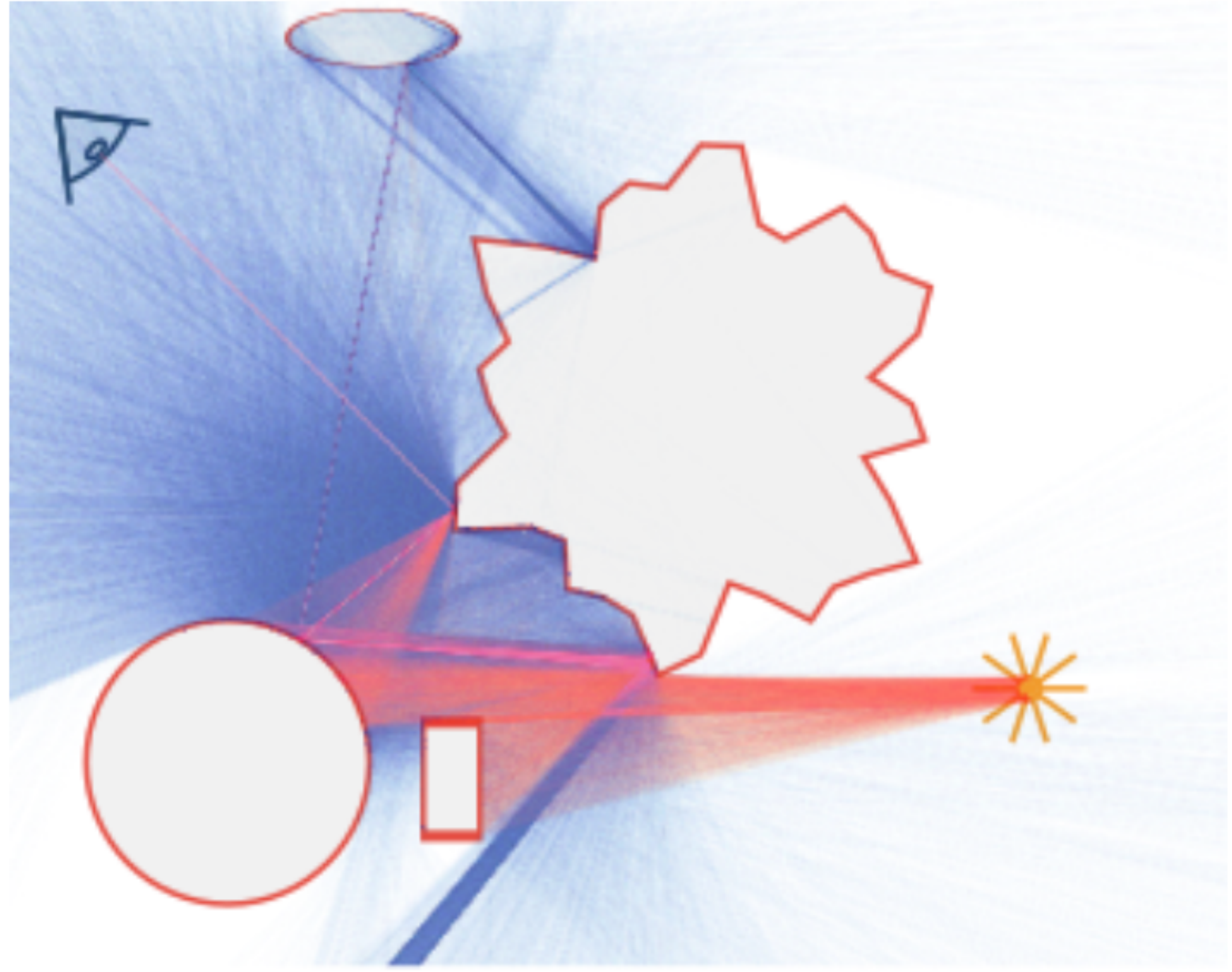
Semi-structured interviews

18 diagrammers, 12 domains

Diagrammers seek existing representations

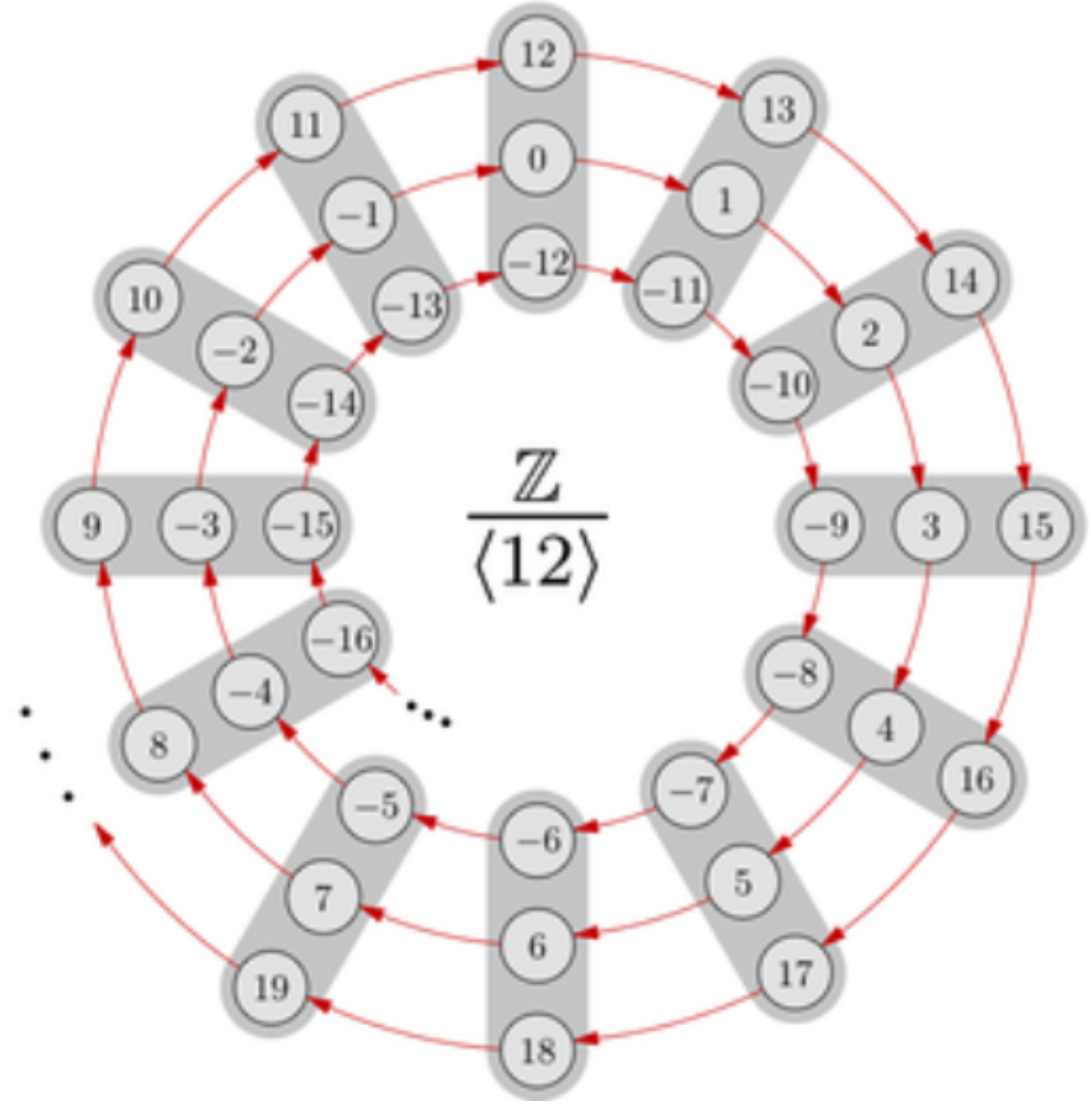


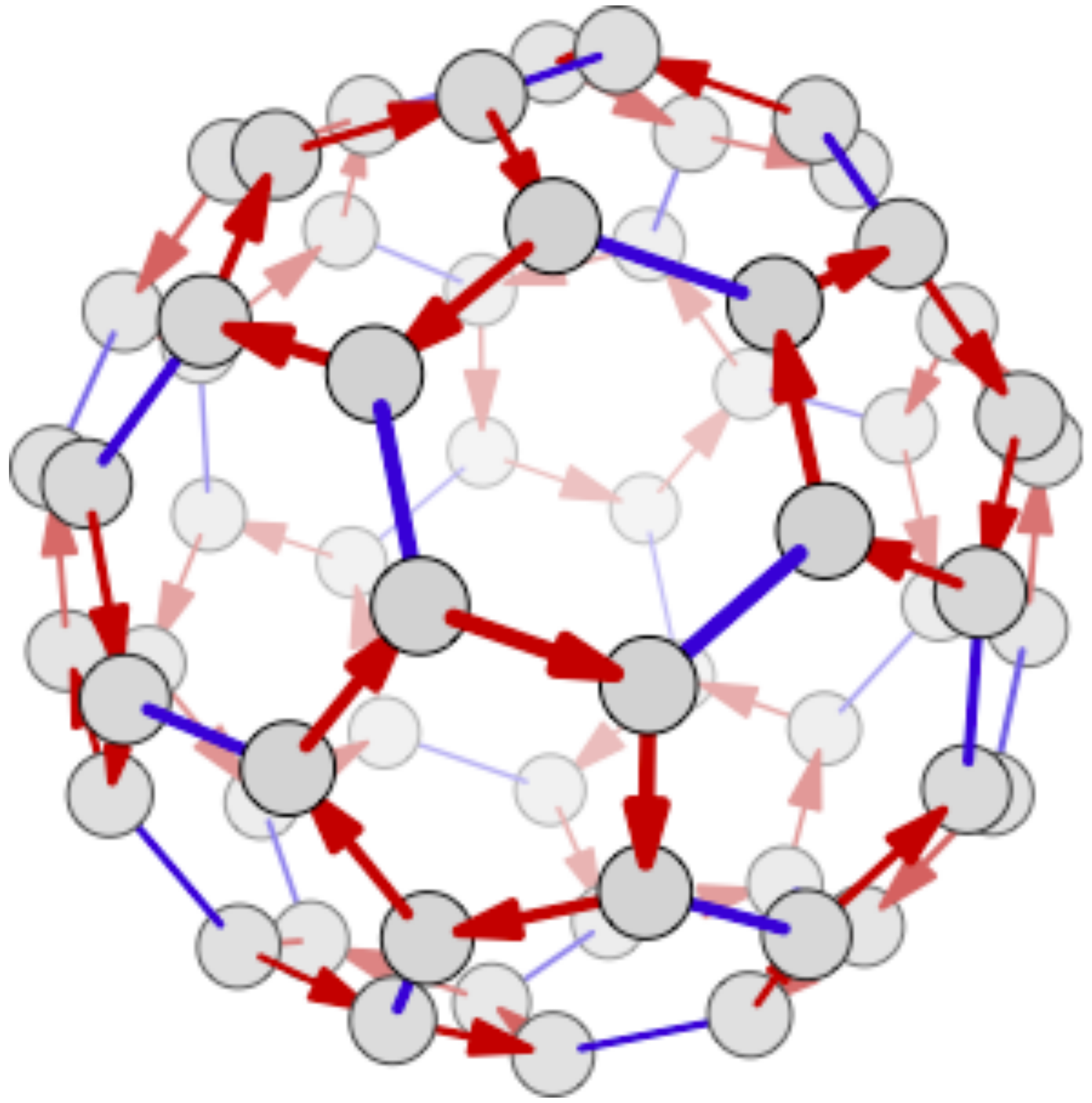




Diagrammers seek existing representations

…or create their own ones

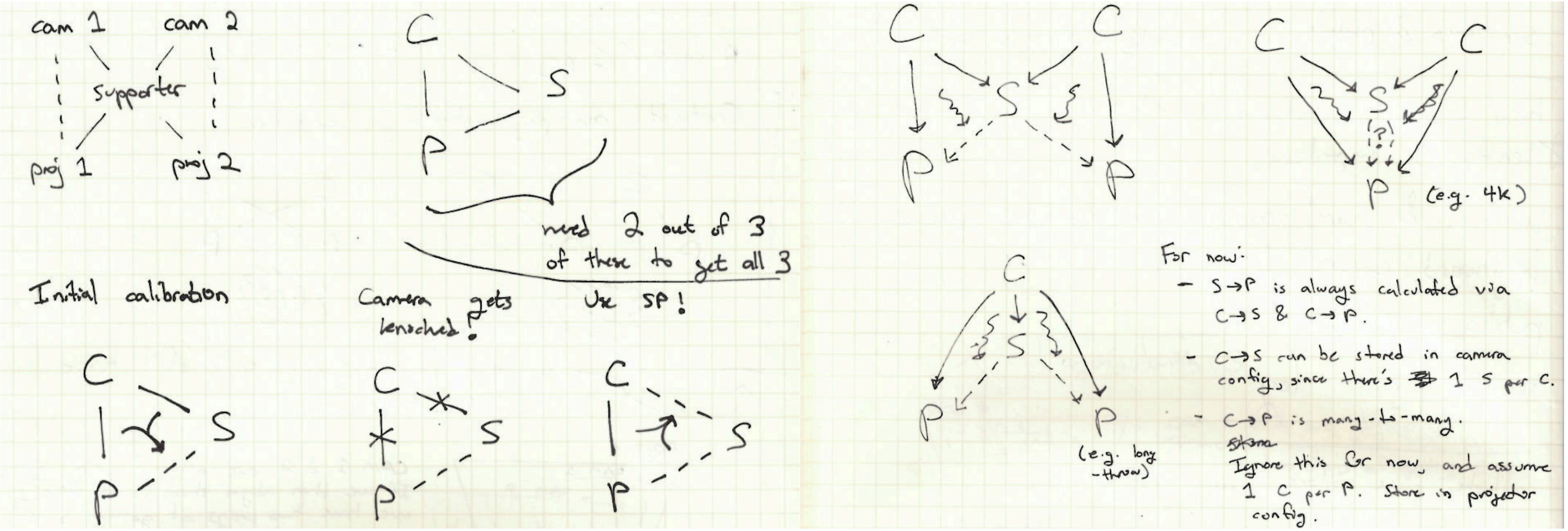




Diagrammers seek existing representations

…or create their own ones

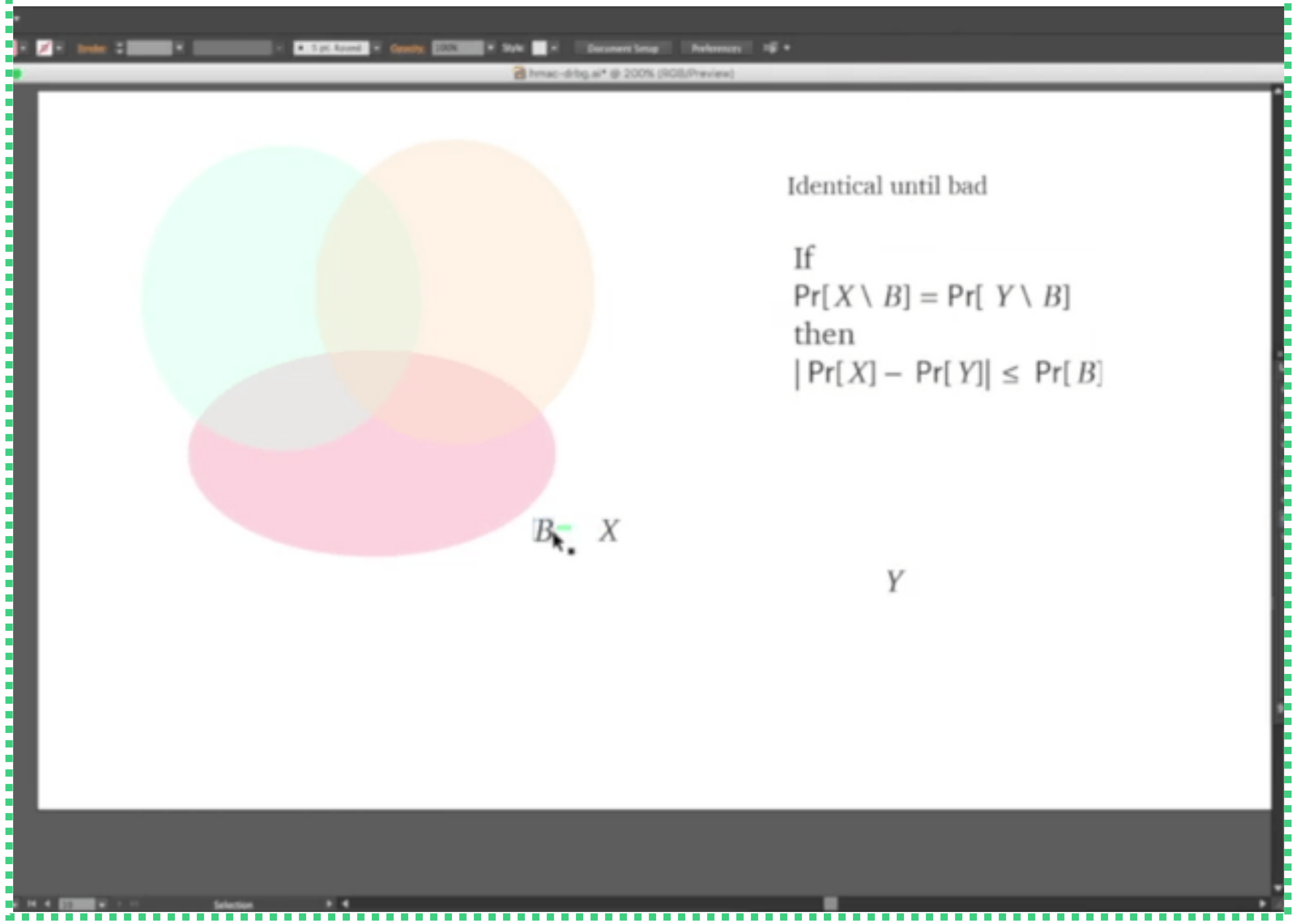
…by sketching



Diagramming tools don’t support representations

Diagramming tools don't support representations

Direct Manipulation



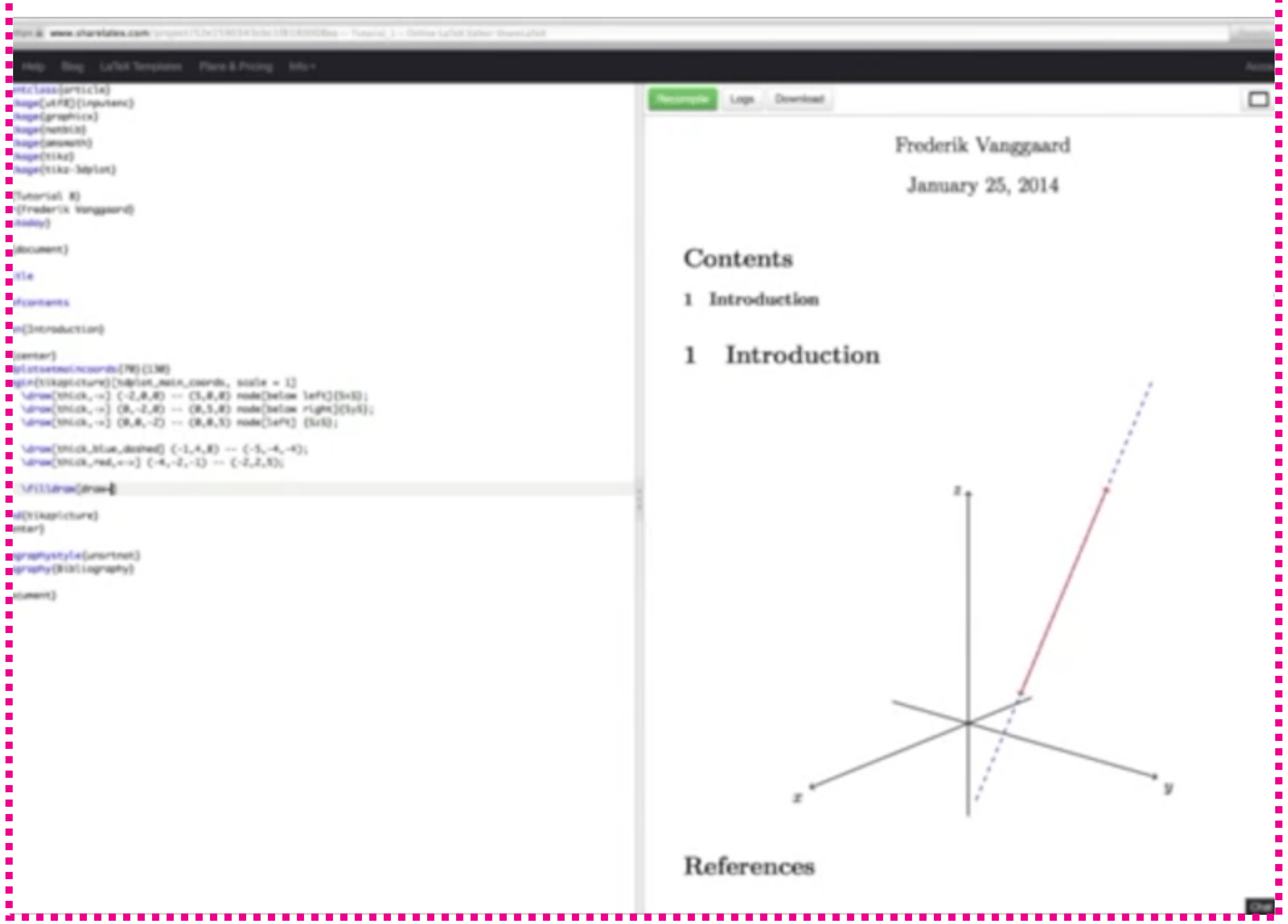
Pixels, shapes, layers

Highly manual

Viscosity

Diagramming tools don’t support representations

Programming languages



#s, macros, functions

Steep learning curve

High upfront cost

> *Macros are terrible, I make macros that are 20 or 30 braces deep [...they’re] just incredibly hard to write and edit. (P11)*

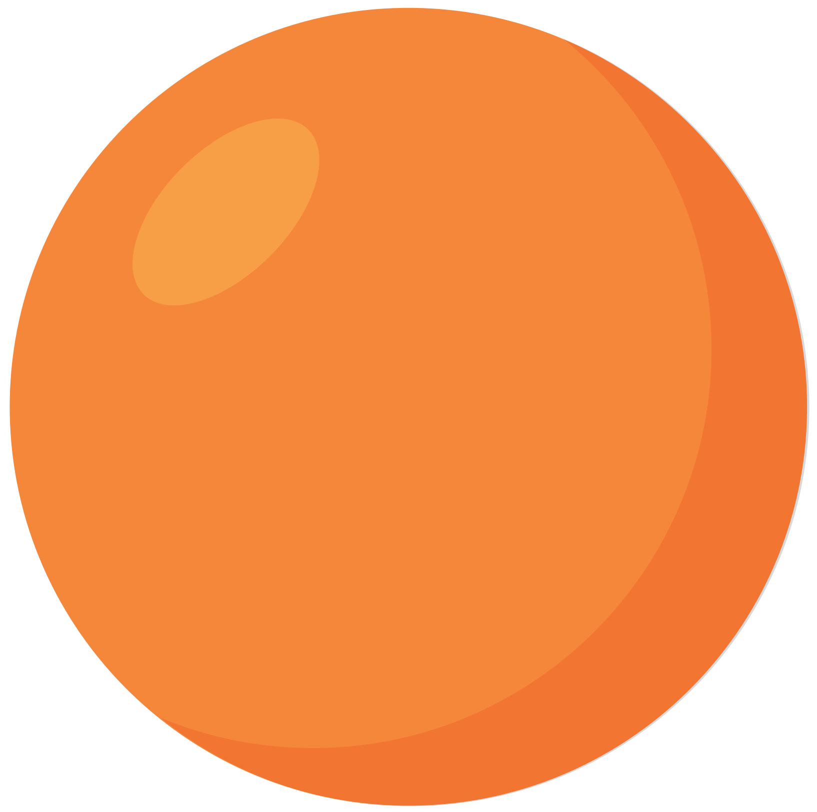
Diagrammers keep representations manually by…

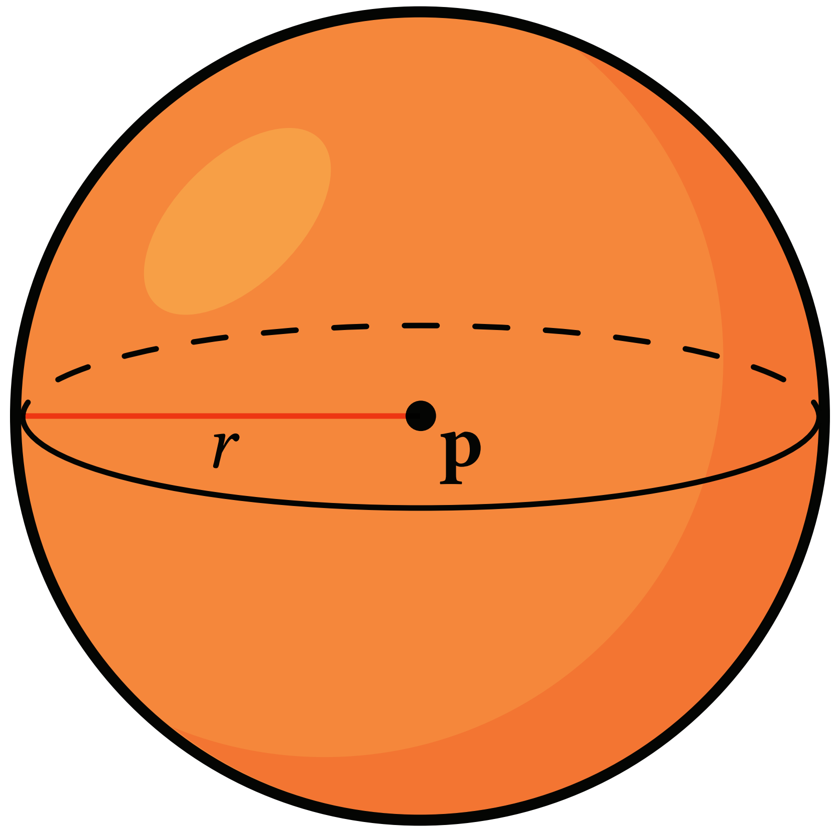
prior versions

Diagrammers keep representations manually by…

prior versions, low-level parameters

```
=== COLORS ===
* Orange
Base: #f7883c
Dark: #000 @ 15% (overlay)
Darker: #000 @ 30% (overlay)
Light: #fff @ 20% (overlay)
Lighter: #fff @ 40% (overlay)
=== STROKES ===
Main (silhouette only): #000 2pt thickness (solid)
Secondary: #000 1pt thickness (solid)
Tertiary: #000 @ 50% (overlay) 1pt thickness
Behind: #000 @ 1pt thickness (dashed: 6pt dash, 7pt gap,
      round cap, align to corners)
=== FONT ===
Base: Linux Libertine (add with TikZ directly in TeX)
Size: 11pt
```





Diagrammers keep…
prior versions, low-level parameters, and personal libraries

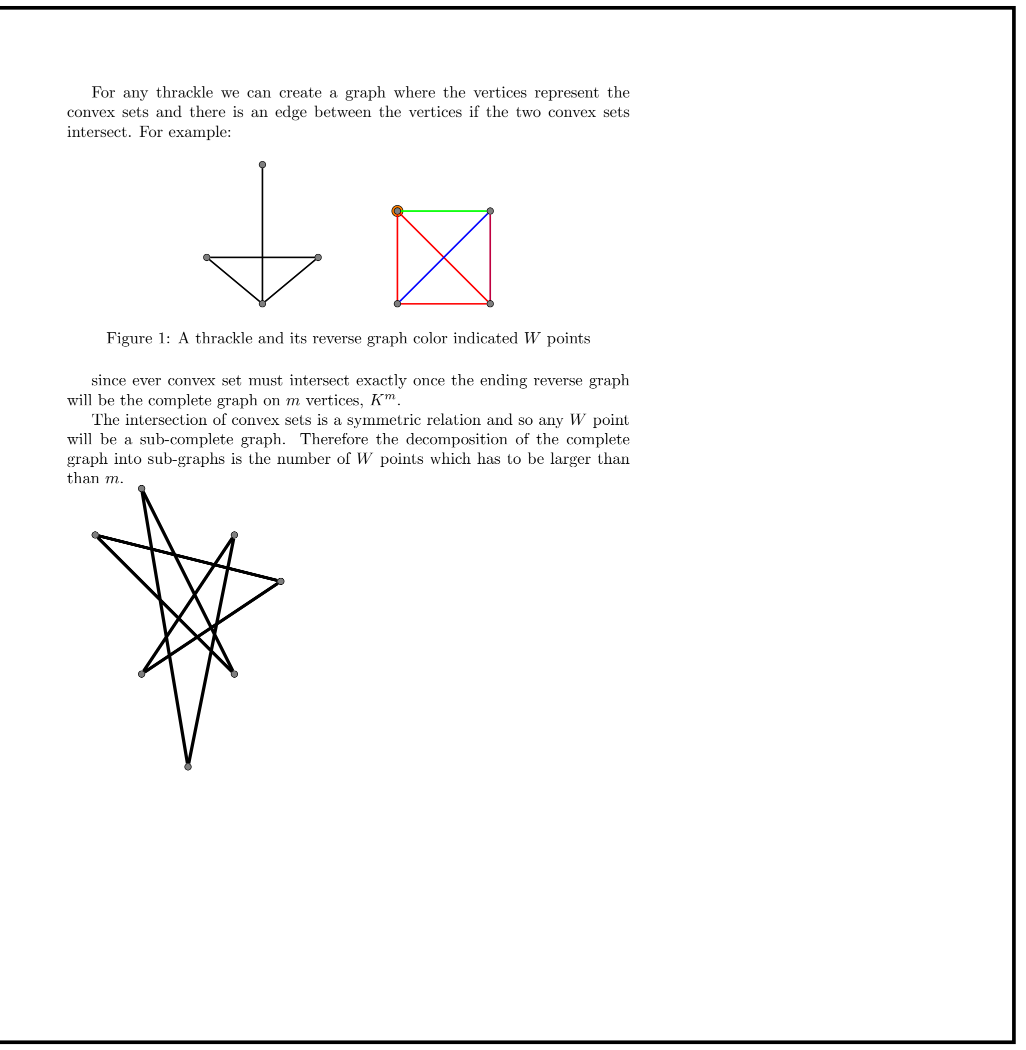
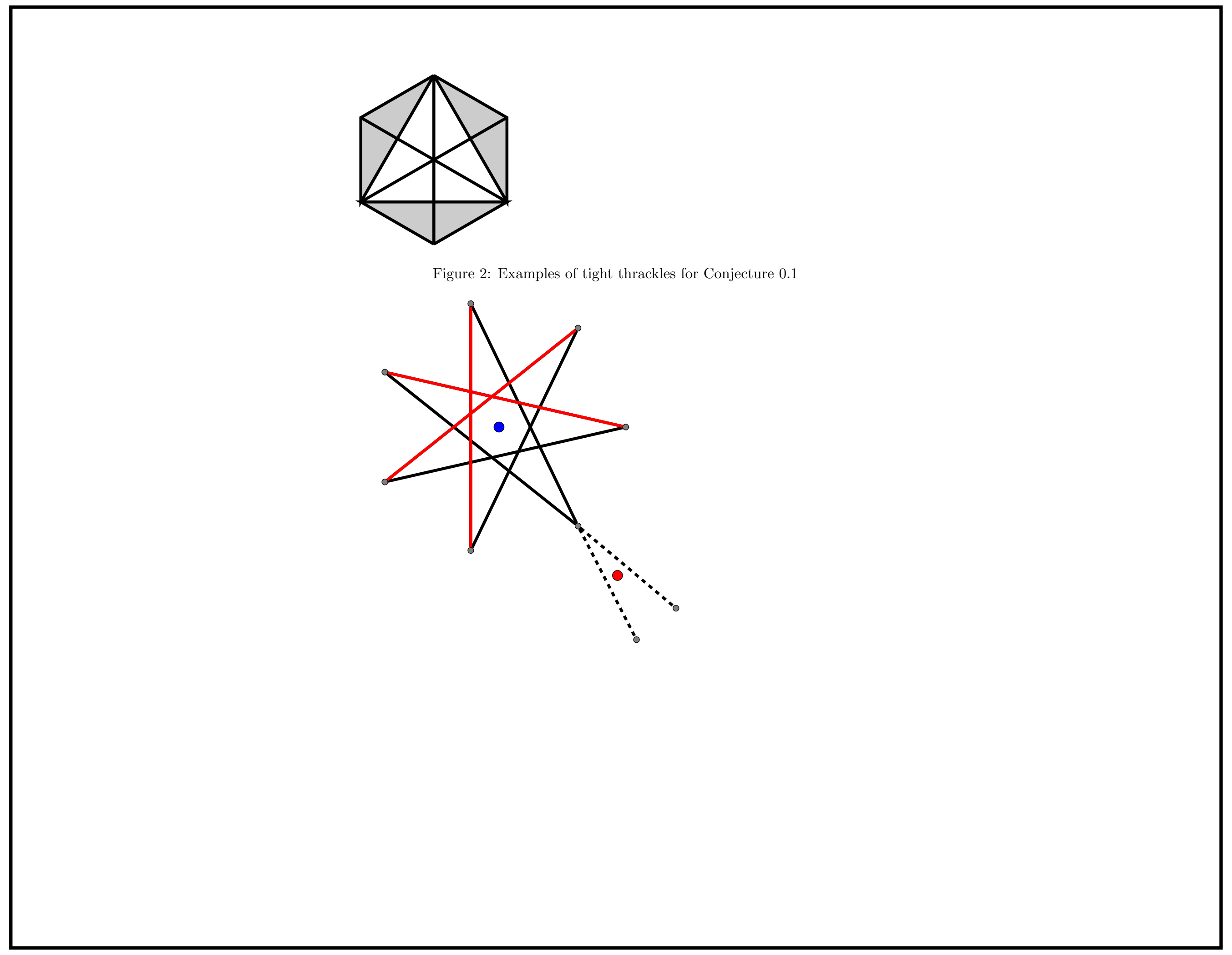
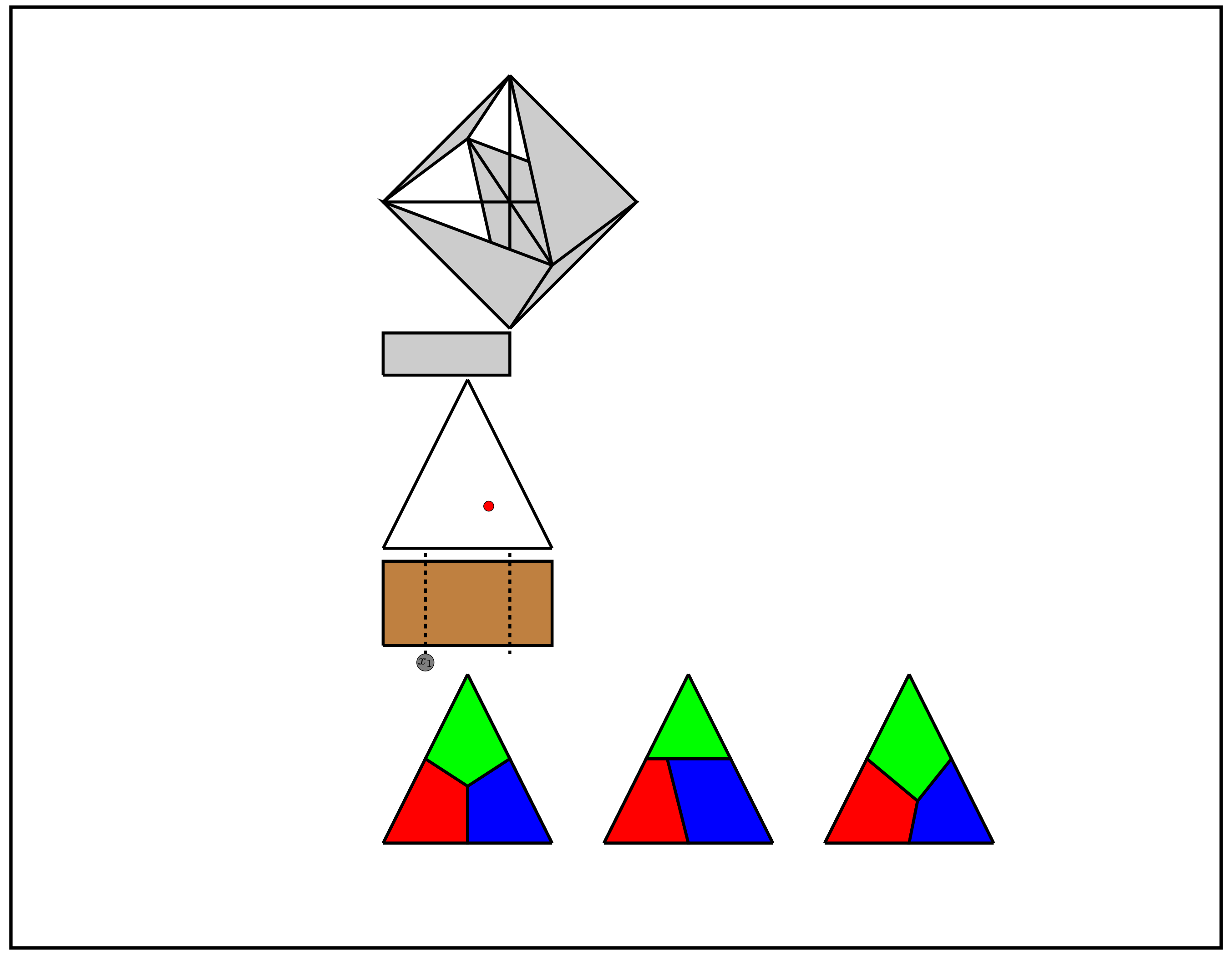

For any thrackle we can create a graph where the vertices represent the convex sets and there is an edge between the vertices if the two convex sets intersect. For example:

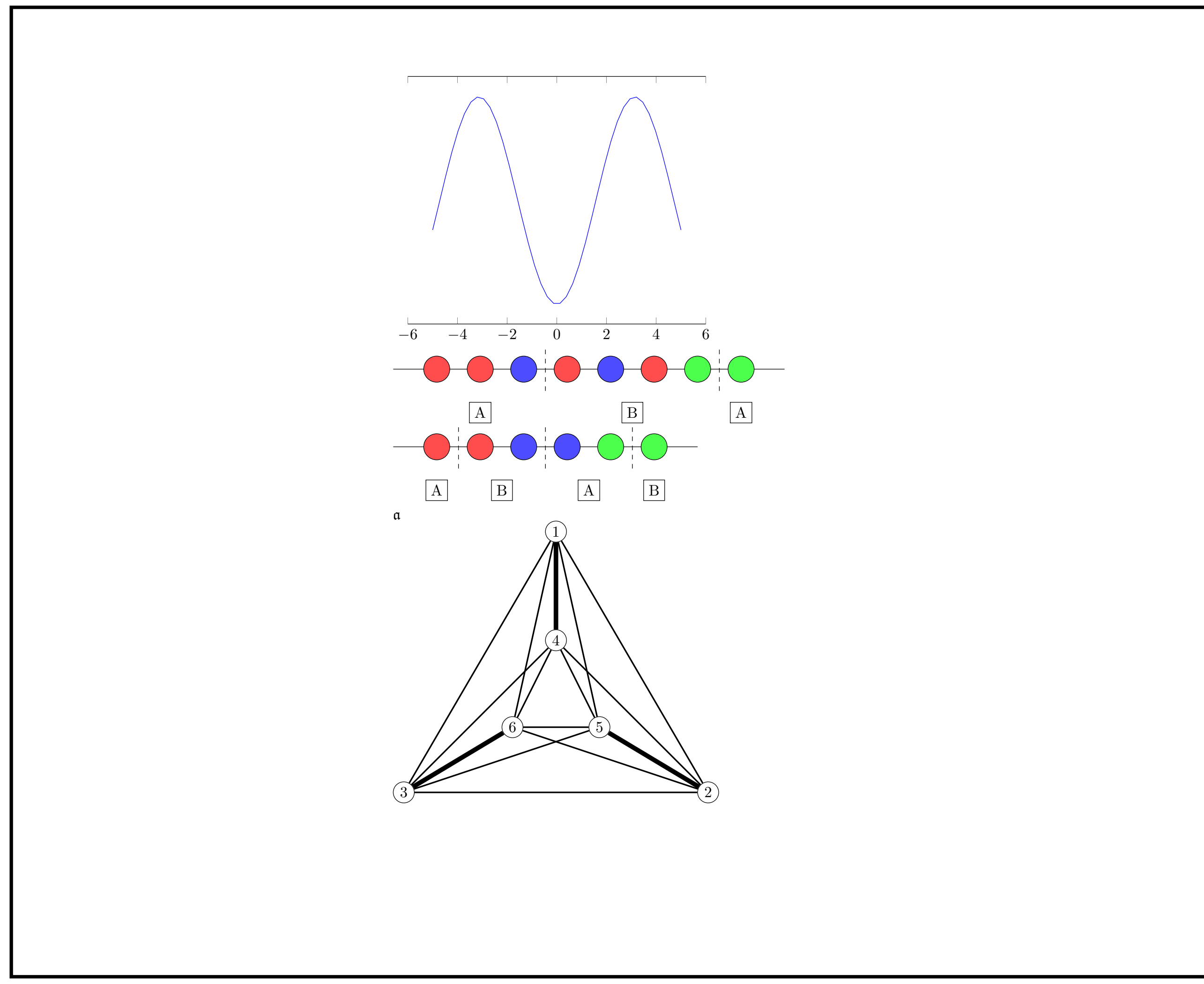
Figure 1: A thrackle and its reverse graph color indicated W points

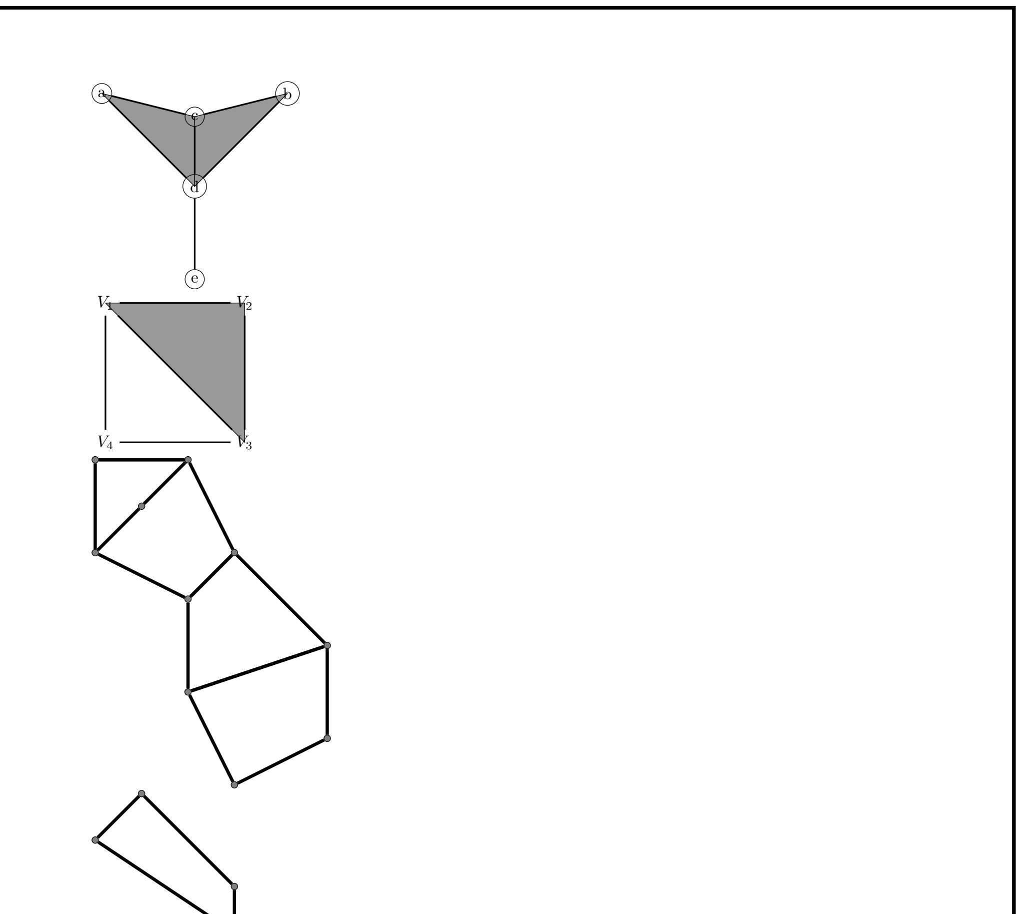
since ever convex set must intersect exactly once the ending reverse graph will be the complete graph on m vertices, K^m.

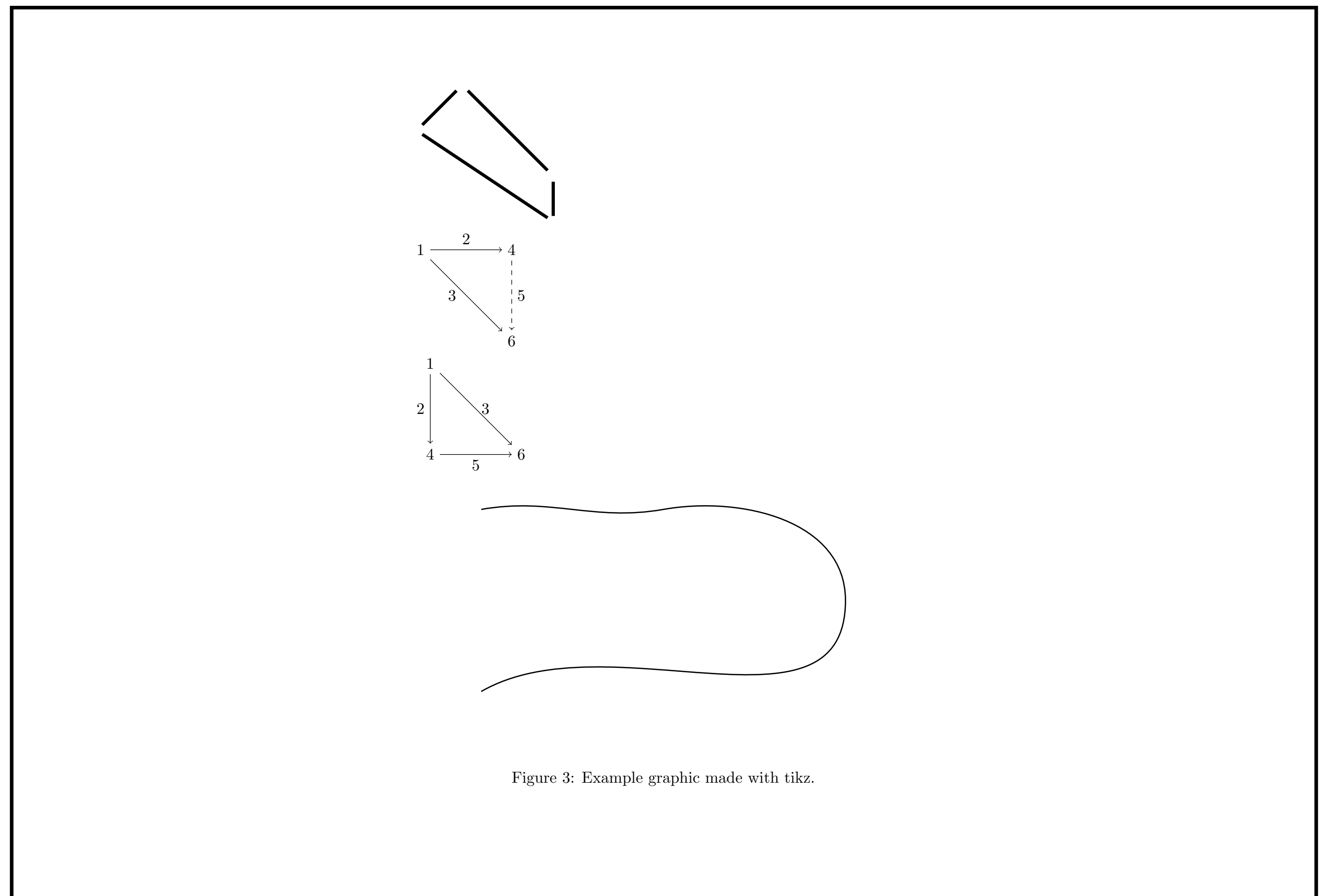
The intersection of convex sets is a symmetric relation and so any W point will be a sub-complete graph. Therefore the decomposition of the complete graph into sub-graphs is the number of W points which has to be larger than than m.

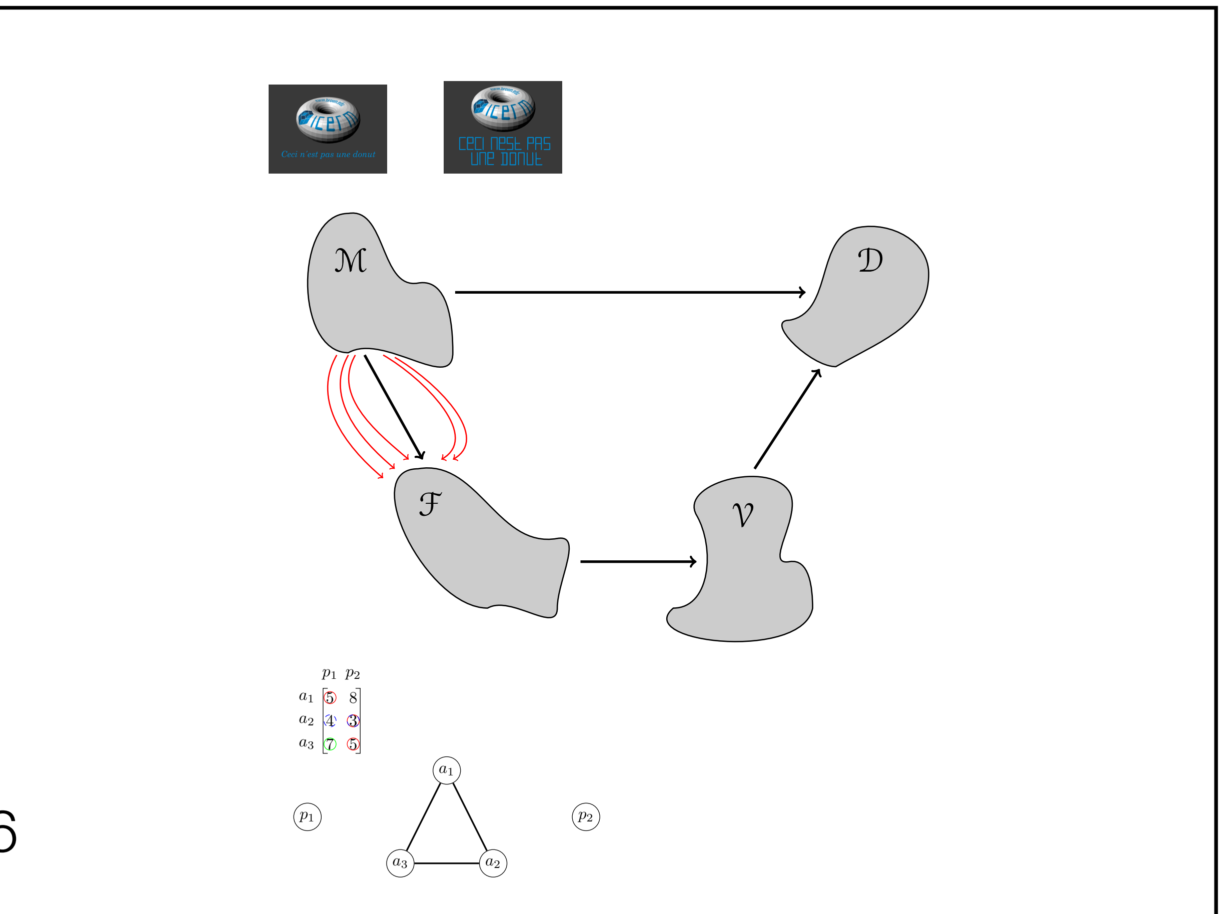


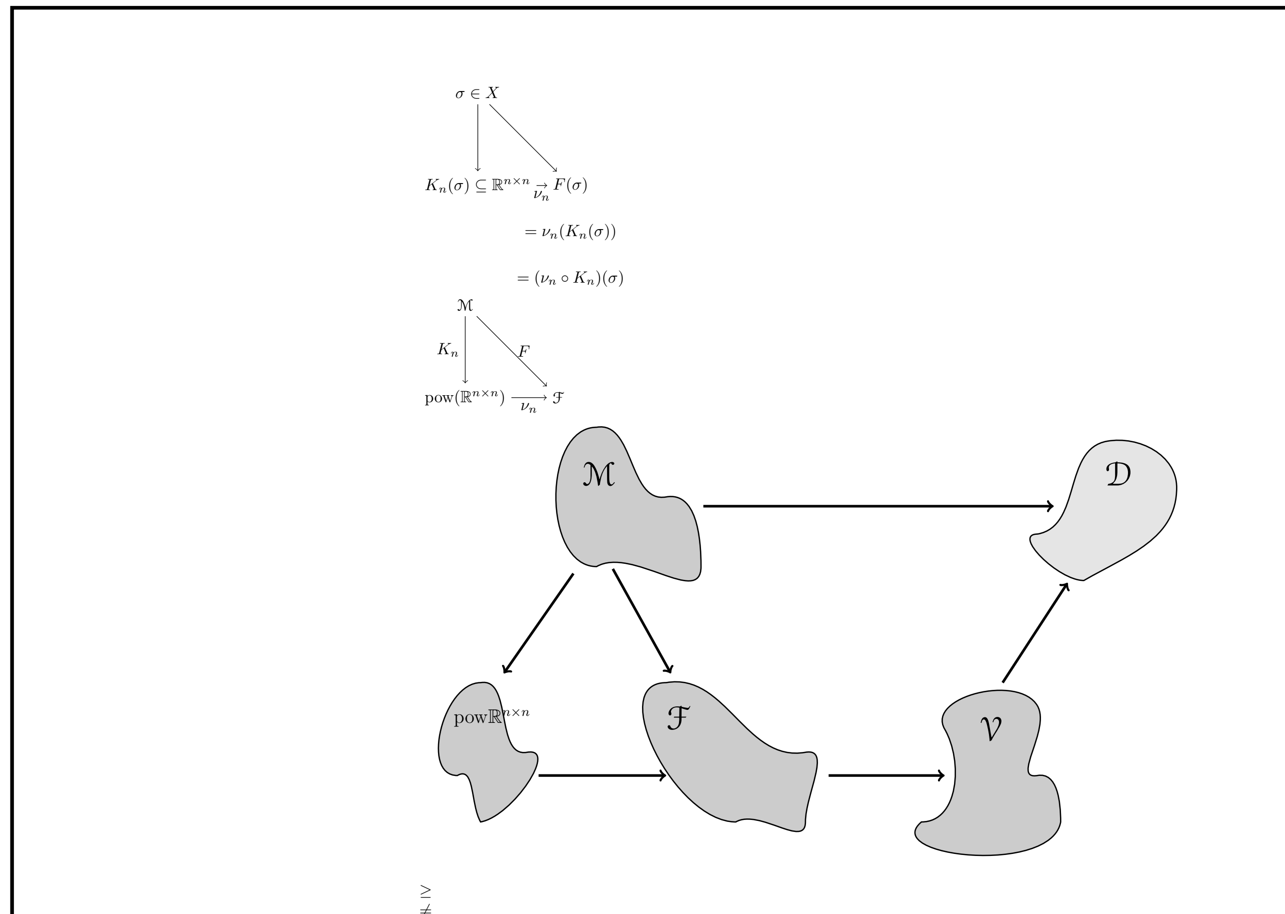












Natural Diagramming

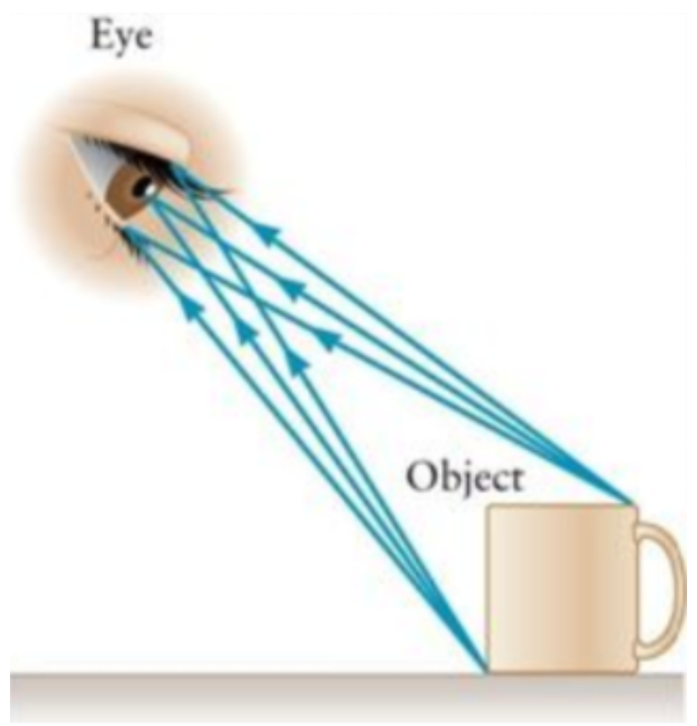
"to express their ideas in the same way they think about them"

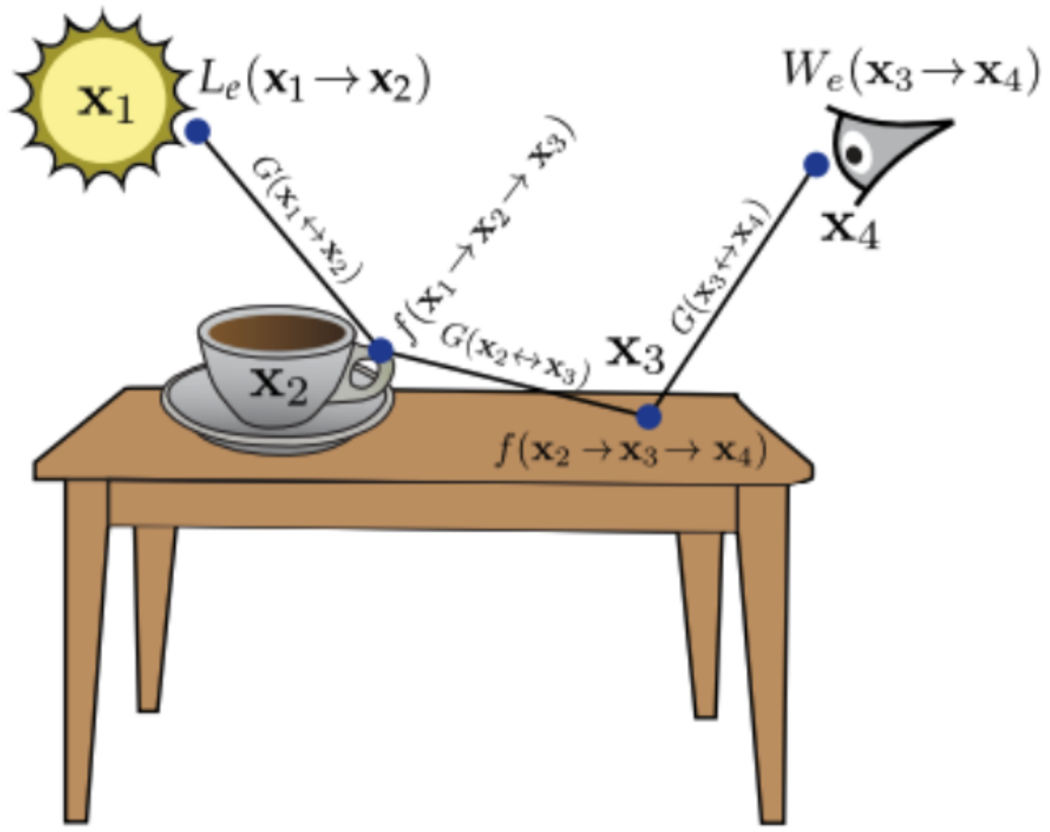
Vocabulary Correspondance

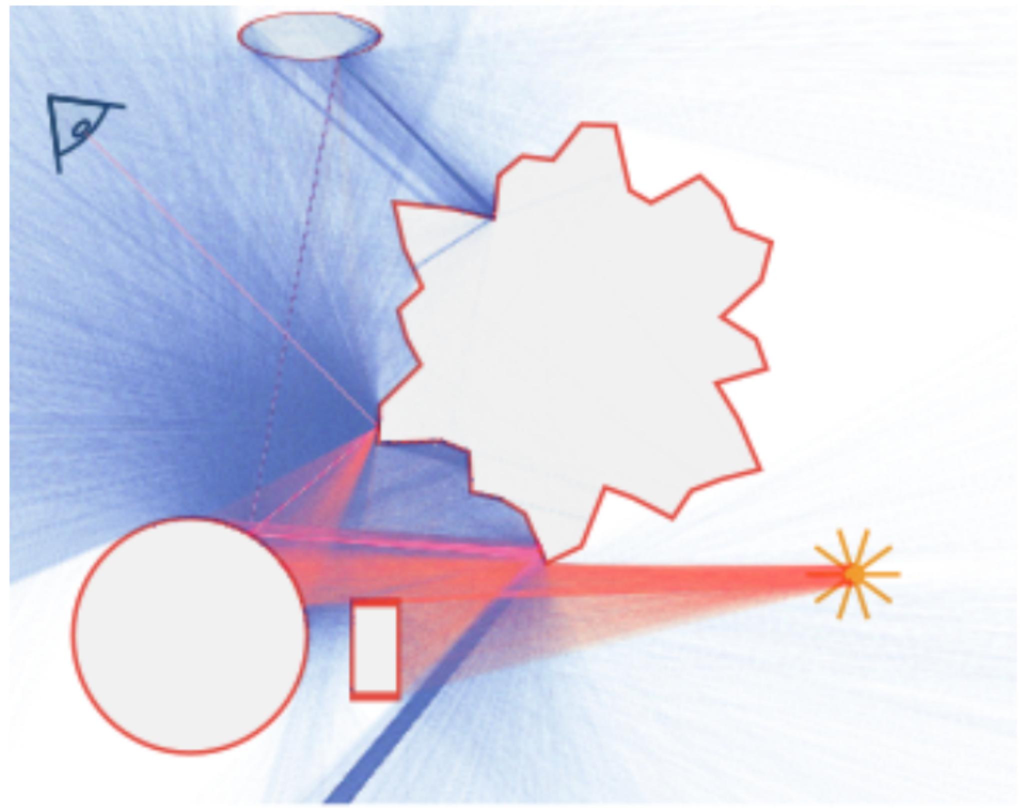
Reduce the *semantic* distance between interaction metaphors and diagrammers' vocabulary

Representation salience

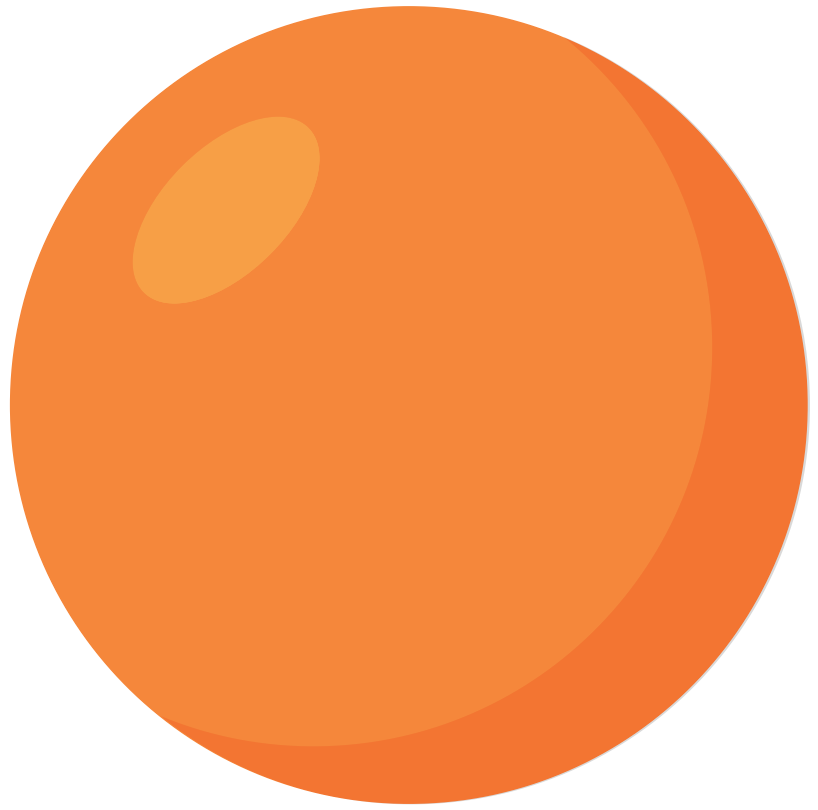
Treating representations as first-class entities

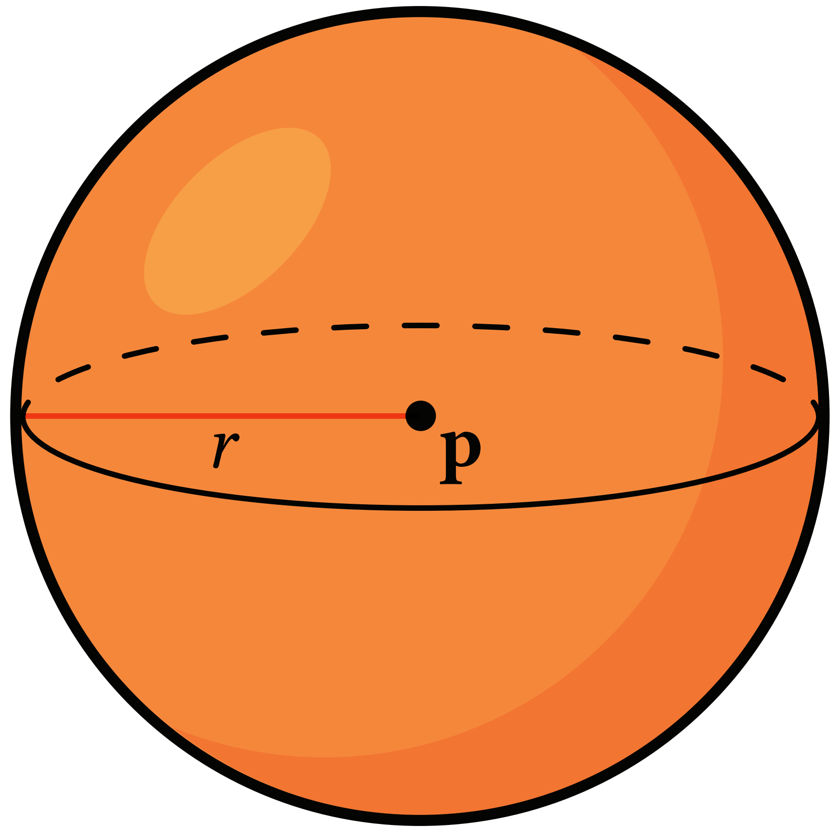




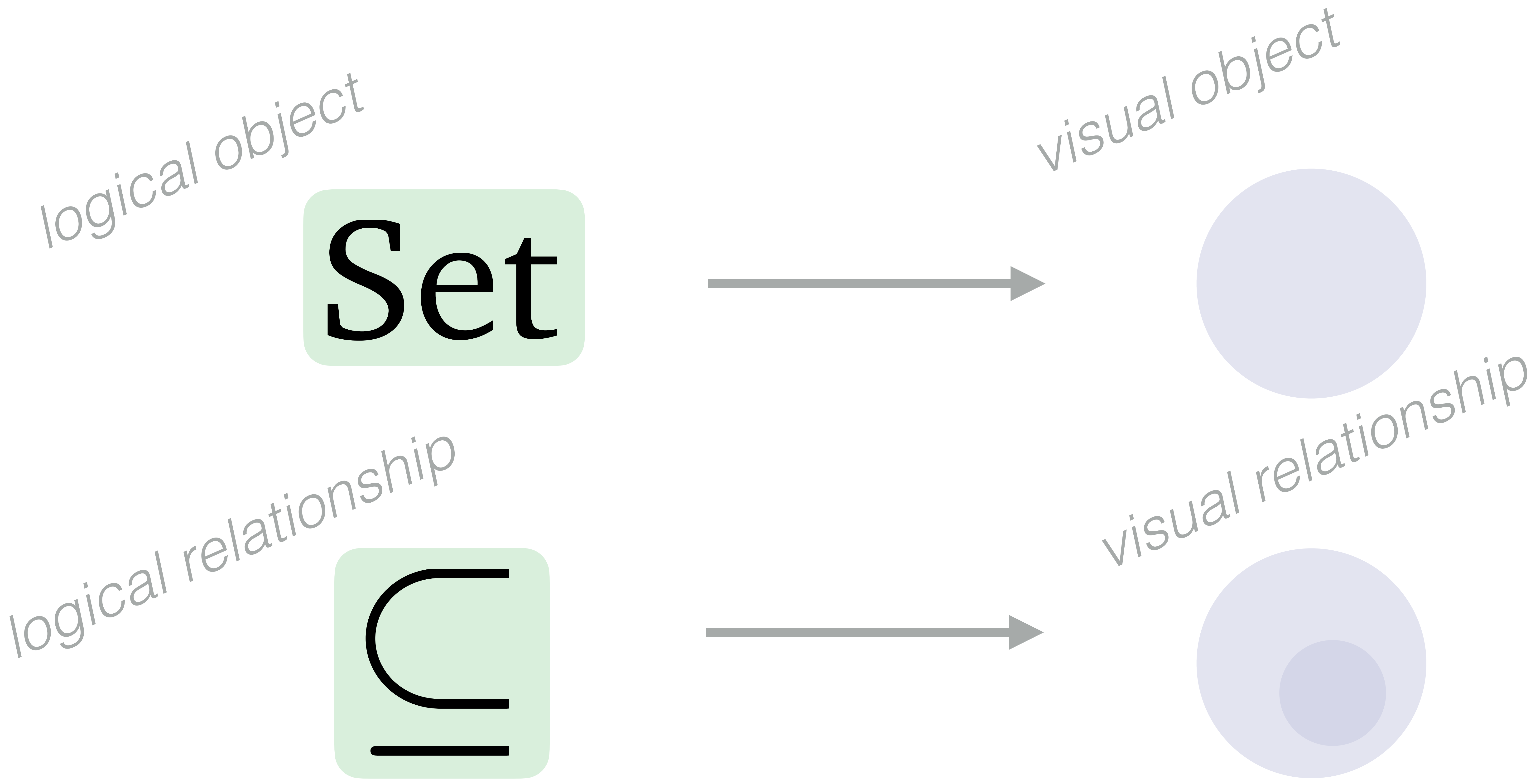


```
=== COLORS ===
* Orange
Base: #f7883c
Dark: #000 @ 15% (overlay)
Darker: #000 @ 30% (overlay)
Light: #fff @ 20% (overlay)
Lighter: #fff @ 40% (overlay)
=== STROKES ===
Main (silhouette only): #000 2pt thickness (solid)
Secondary: #000 1pt thickness (solid)
Tertiary: #000 @ 50% (overlay) 1pt thickness
Behind: #000 @ 1pt thickness (dashed: 6pt dash, 7pt gap,
      round cap, align to corners)
=== FONT ===
Base: Linux Libertine (add with TikZ directly in TeX)
Size: 11pt
```





Penrose: model how math diagrammers think

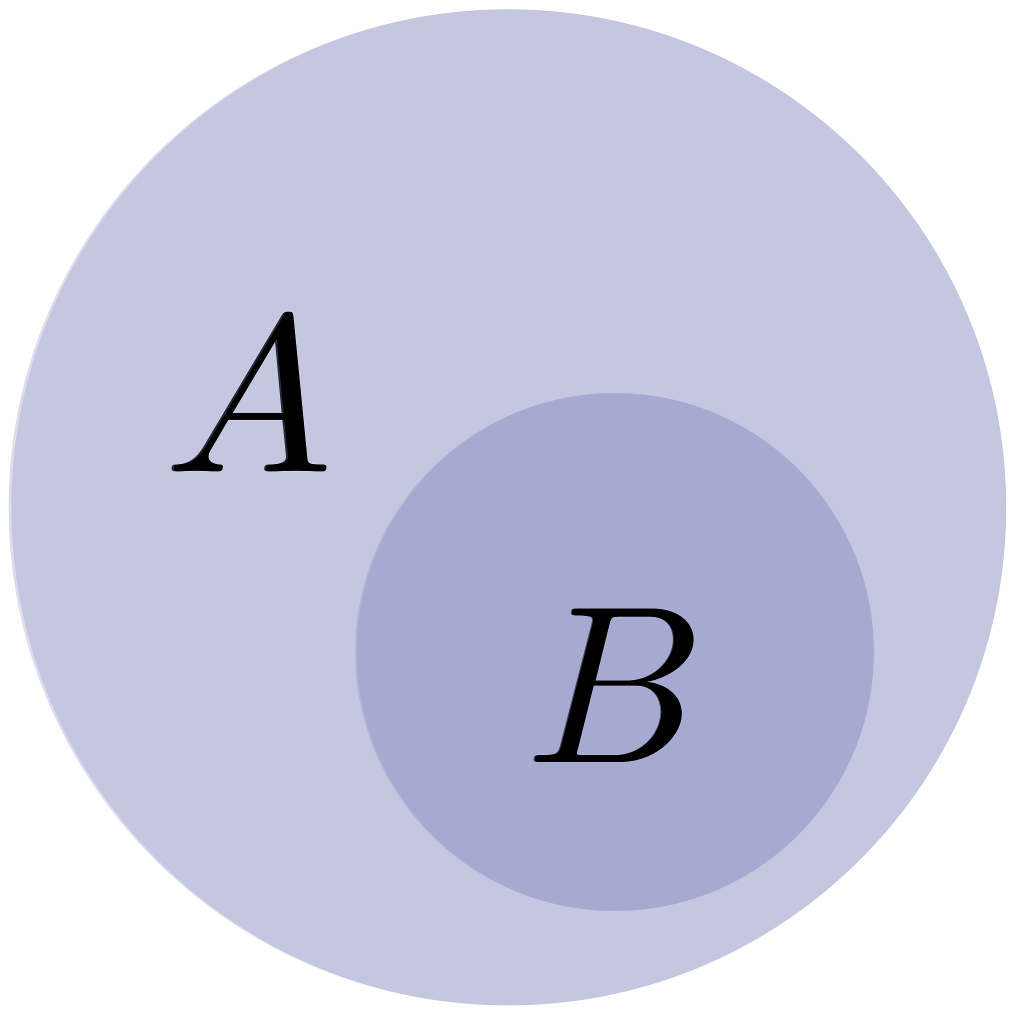


Penrose: explicitly declare mappings from math to graphics

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

Domain

```
type Set
predicate IsSubset(Set s1, Set s2)
notation "B ⊆ A" ~ "IsSubset(B, A)"
```



Penrose: explicitly declare mappings from math to graphics

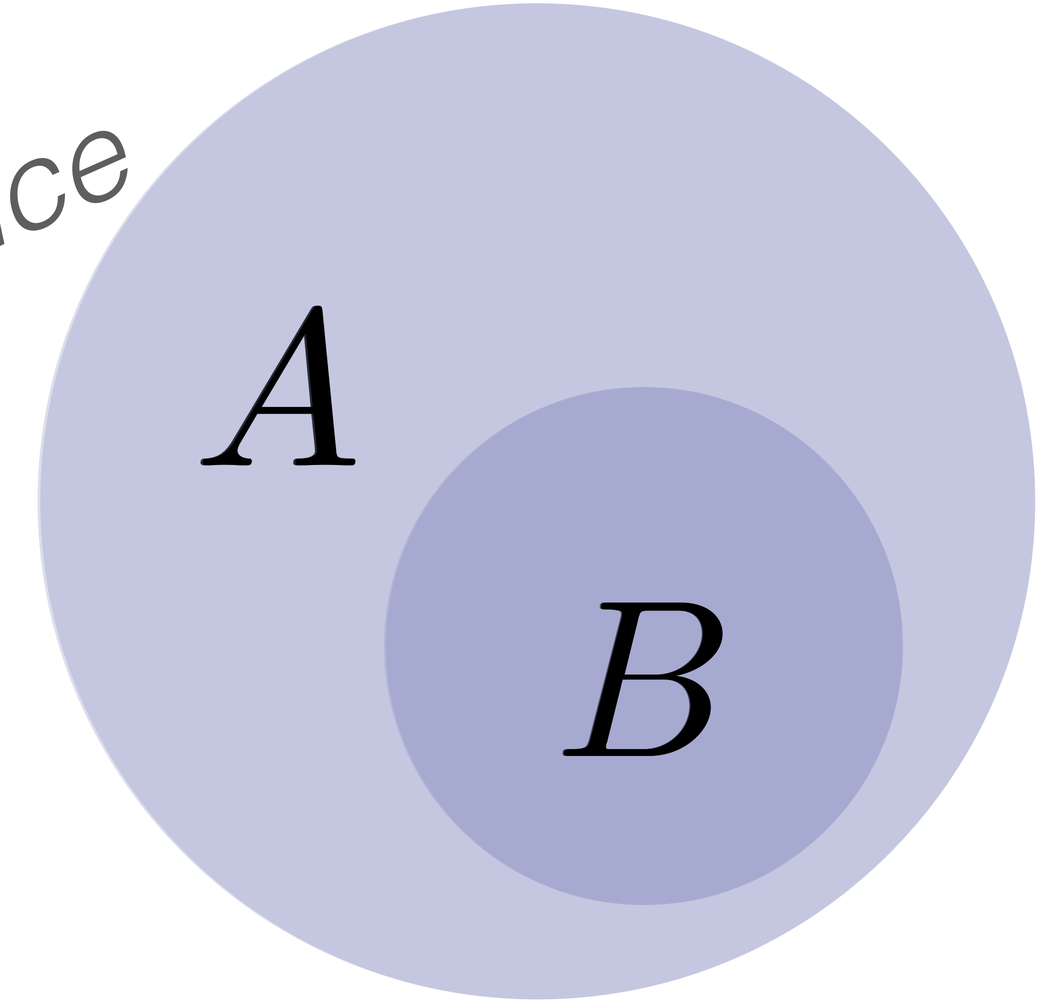
There exist sets A, B such that $B \subseteq A$.

Domain

```
type Set
predicate IsSubset(Set s1, Set s2)
notation "B ⊆ A" ~ "IsSubset(B, A)"
```

Substance

```
Set A, B
B ⊆ A
```



Penrose: explicitly declare mappings from math to graphics

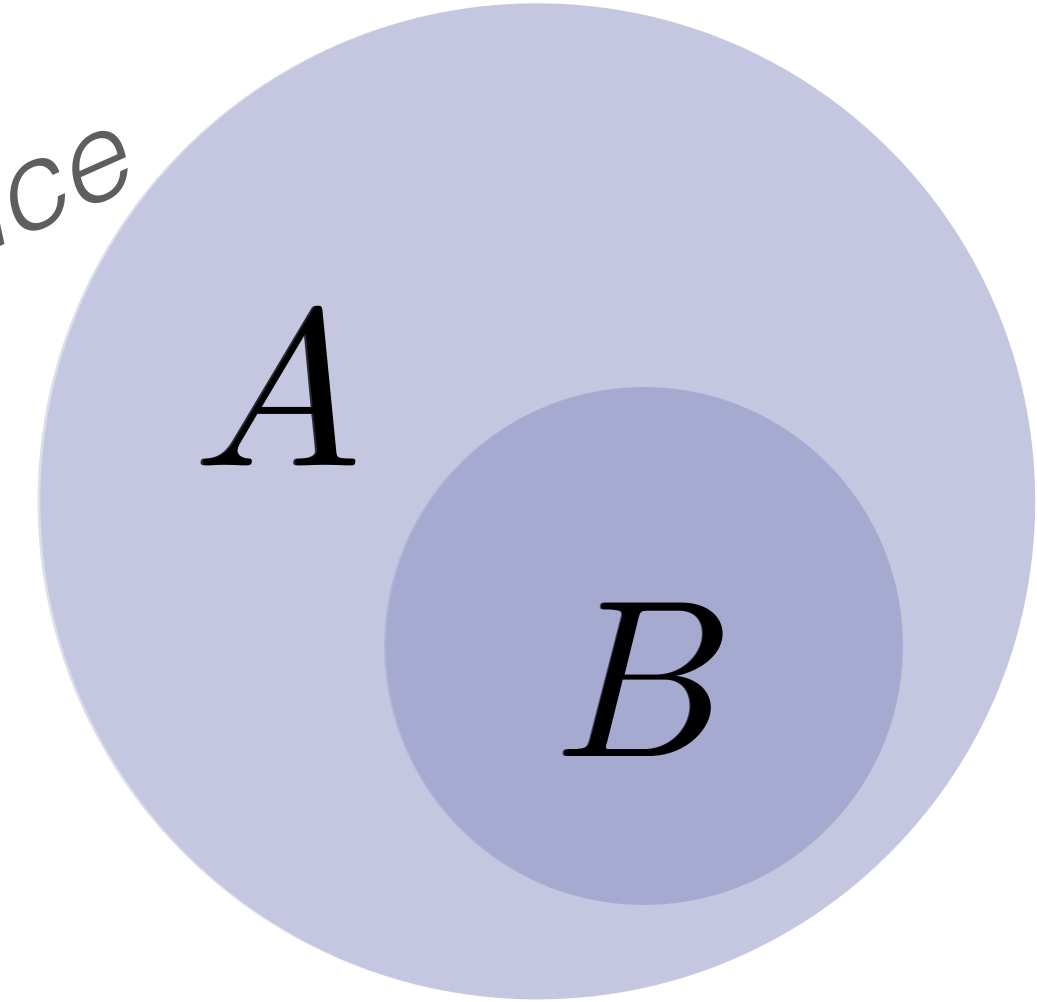
There exist sets A, B such that $B \subseteq A$.

Domain

```
type Set
predicate IsSubset(Set s1, Set s2)
notation "B ⊆ A" ~ "IsSubset(B, A)"
```

Substance

```
Set A, B
B ⊆ A
```



Style

```
Set X { X.shape = Circle { } }
Set X, Y where X ⊆ Y {
  ensure contains(Y.shape, X.shape)
}
```

Penrose: explicitly declare mappings from math to graphics

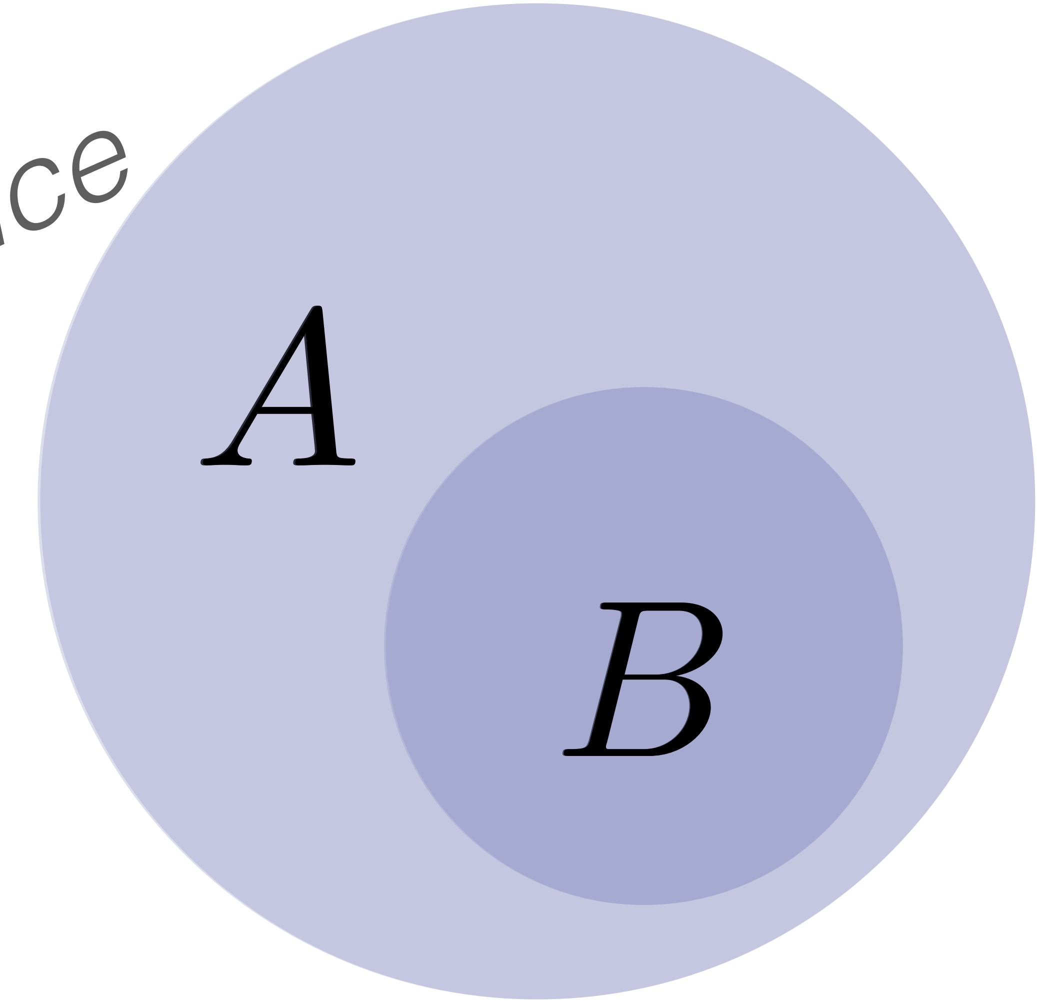
There exist sets A, B such that $B \subseteq A$.

Domain

```
type Set
predicate IsSubset(Set s1, Set s2)
notation "B ⊆ A" ~ "IsSubset(B, A)"
```

Substance

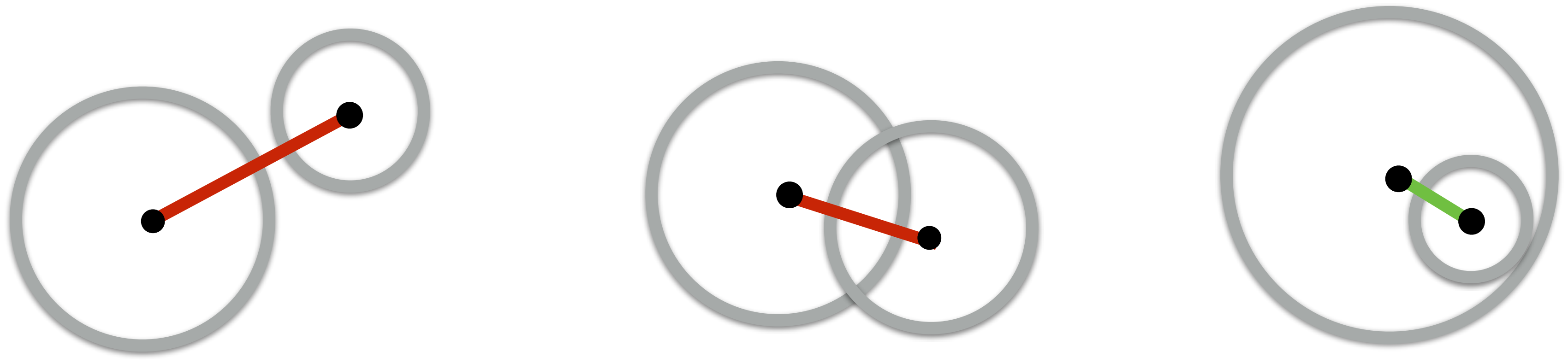
```
Set A, B
B ⊆ A
```



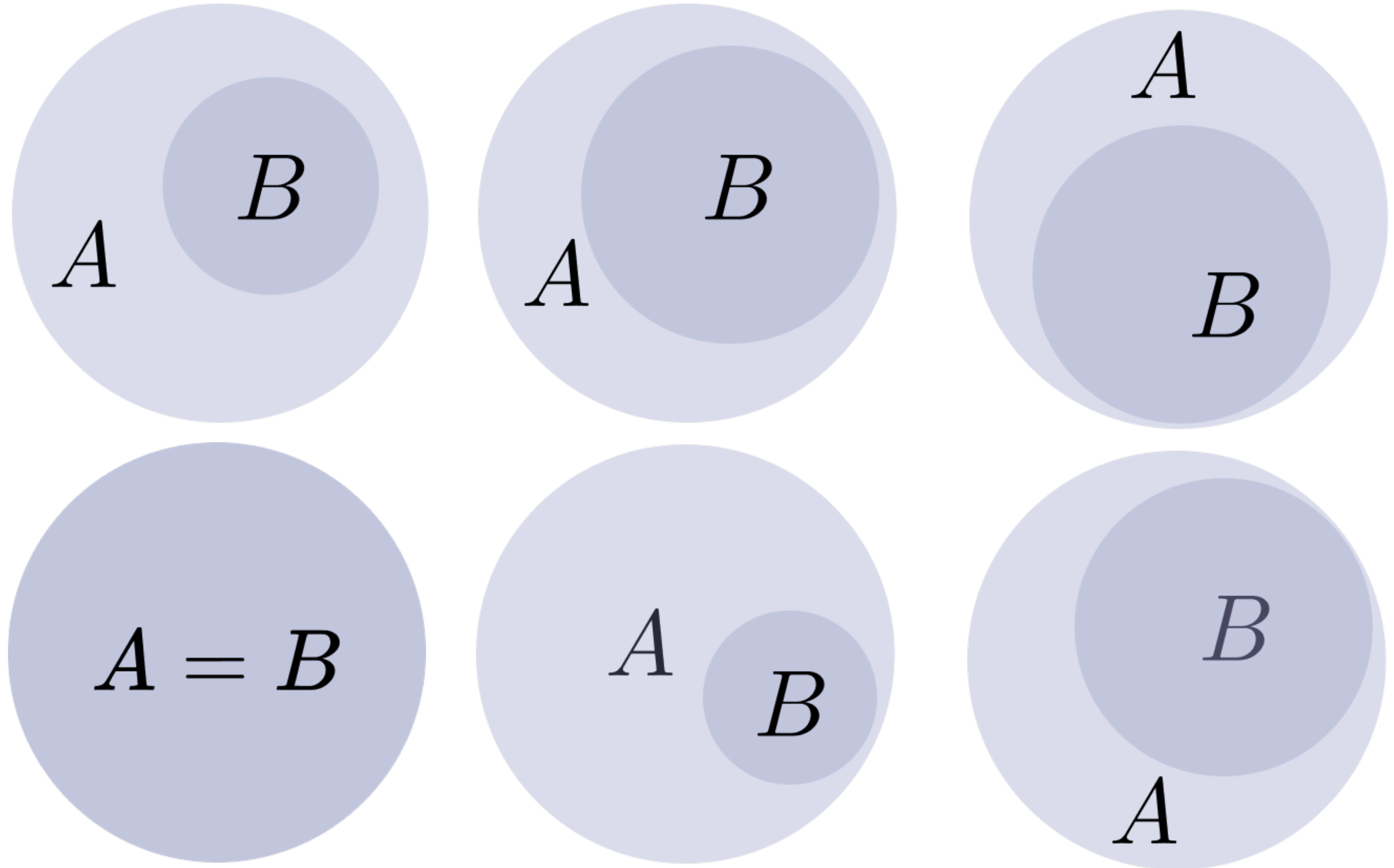
Style

```
Set X { X.shape = Circle { } }
Set X, Y where X ⊆ Y {
  ensure contains(Y.shape, X.shape)
}
```

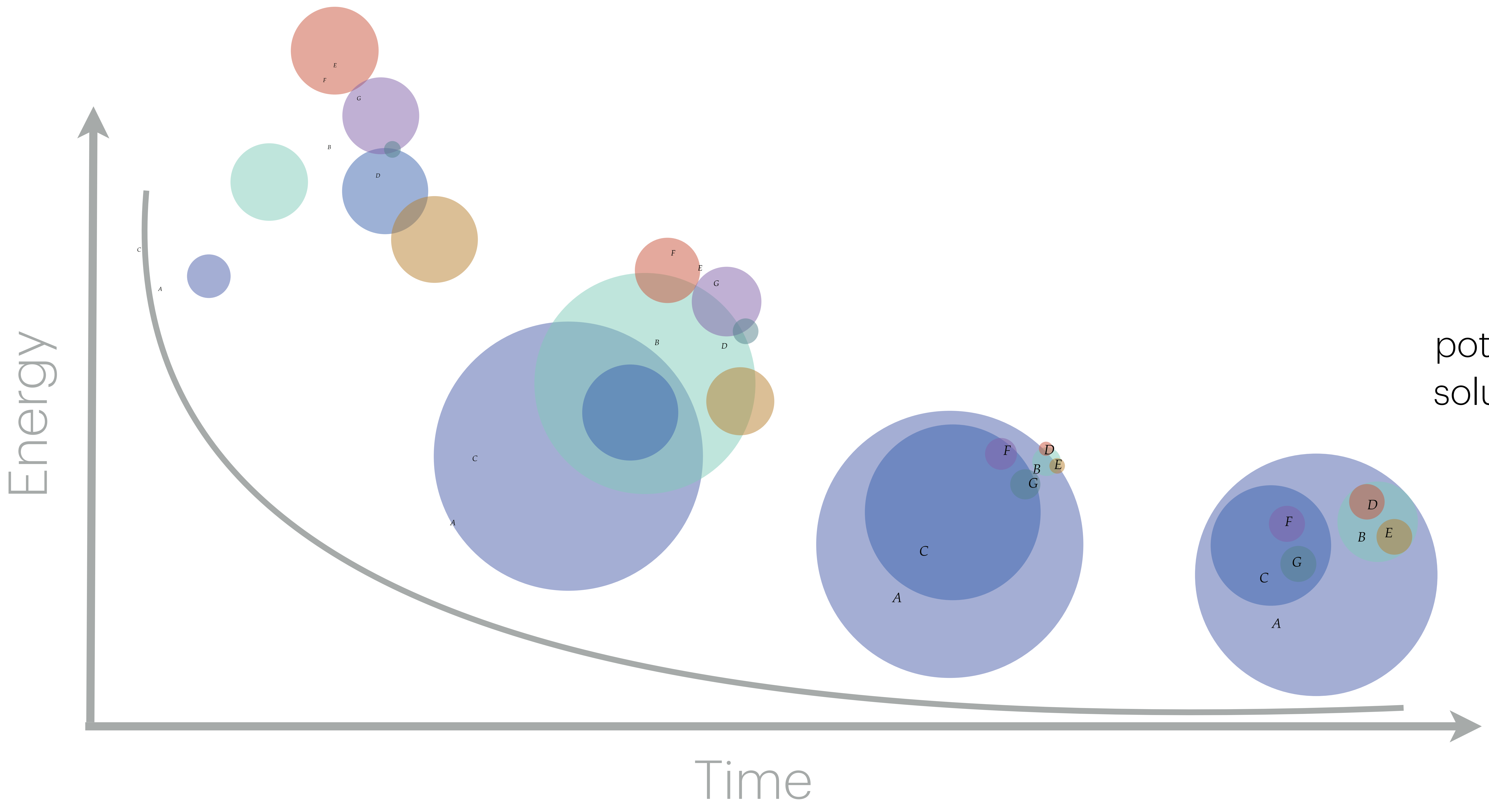
"contains" gets automatically translated into a numerical constraint



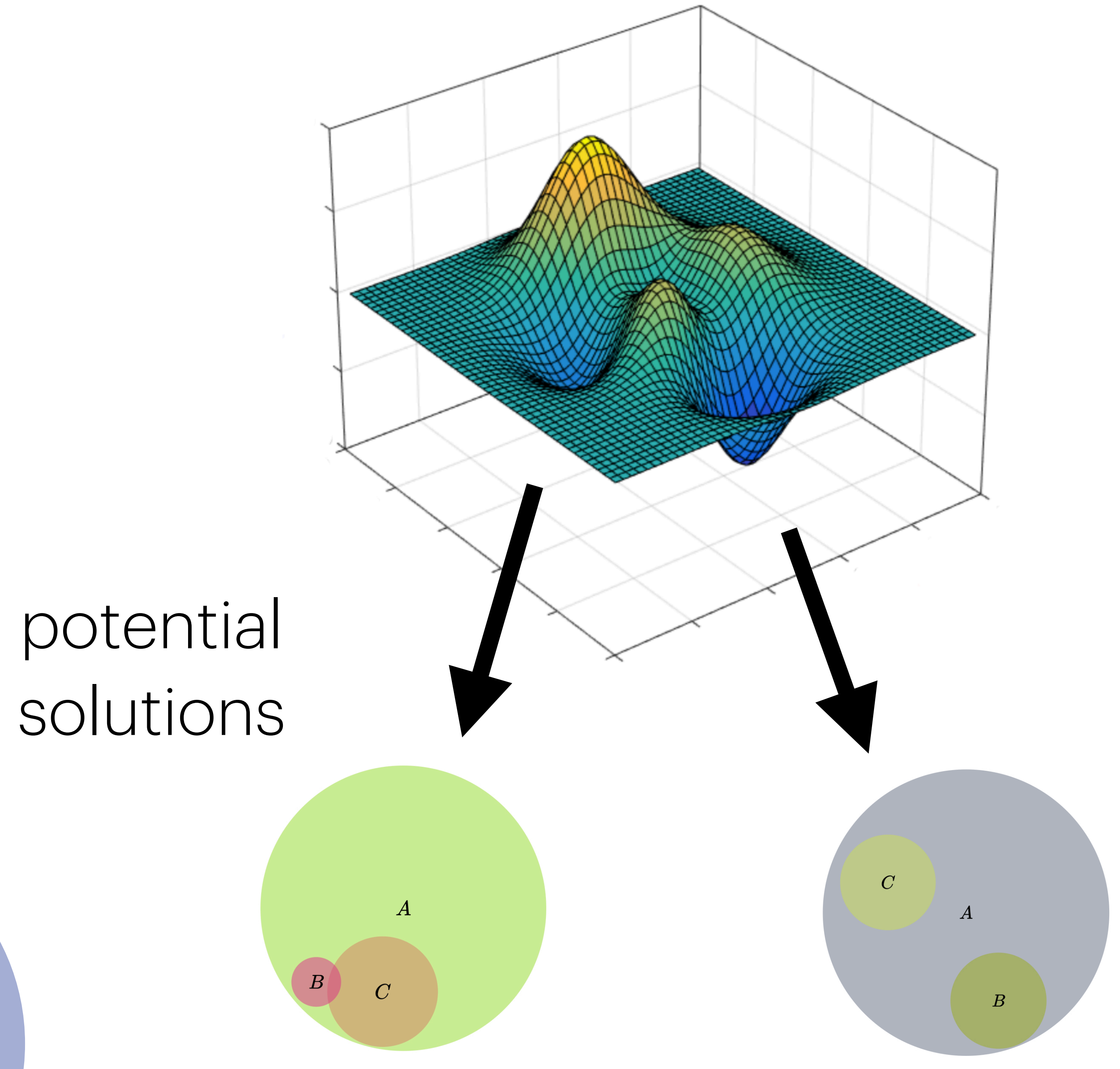
$$|c_Y - c_X| < r_Y - r_X$$



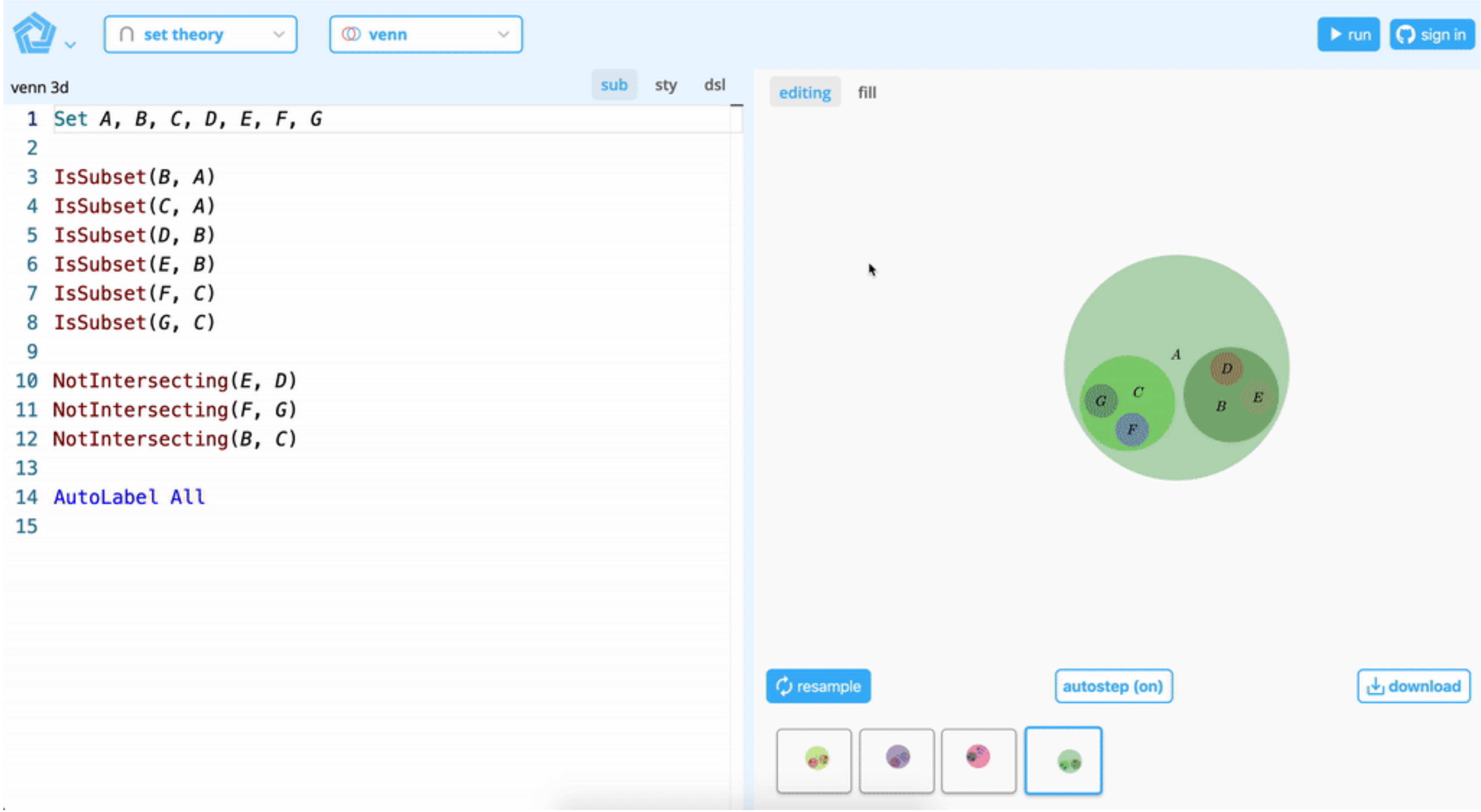
The layout engine: finding instances of visual representation

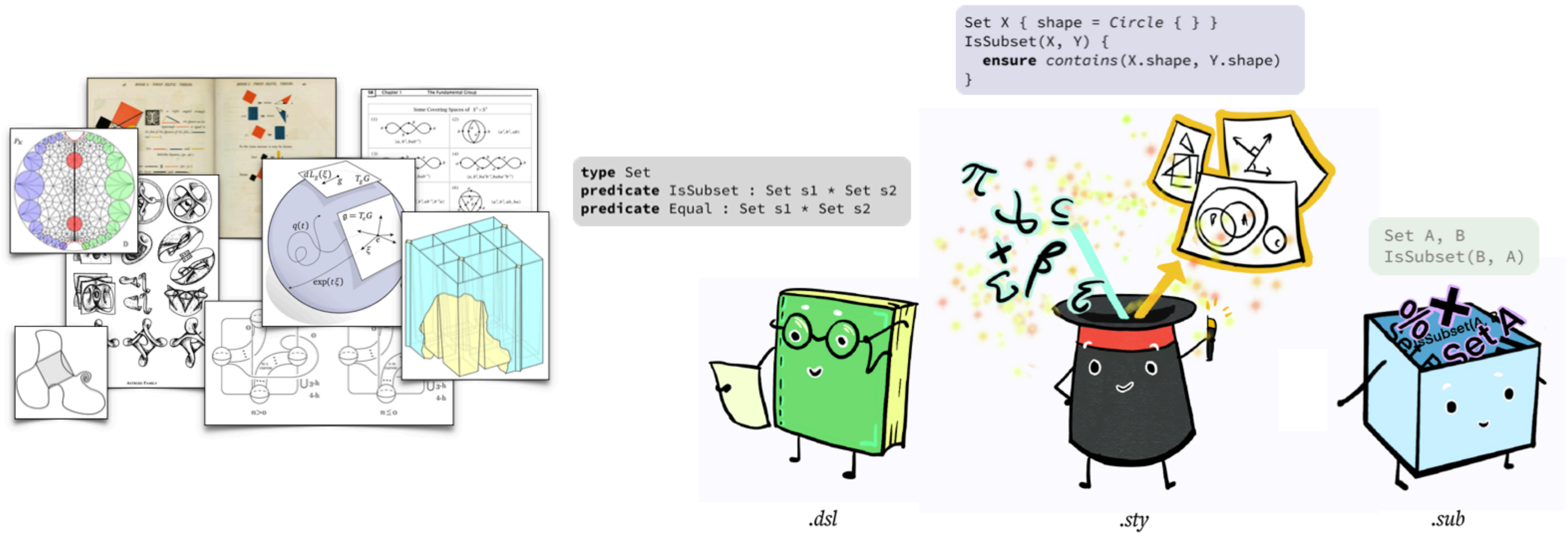


Energy landscape: "how good is this diagram?"



Diagramming with Style(s)





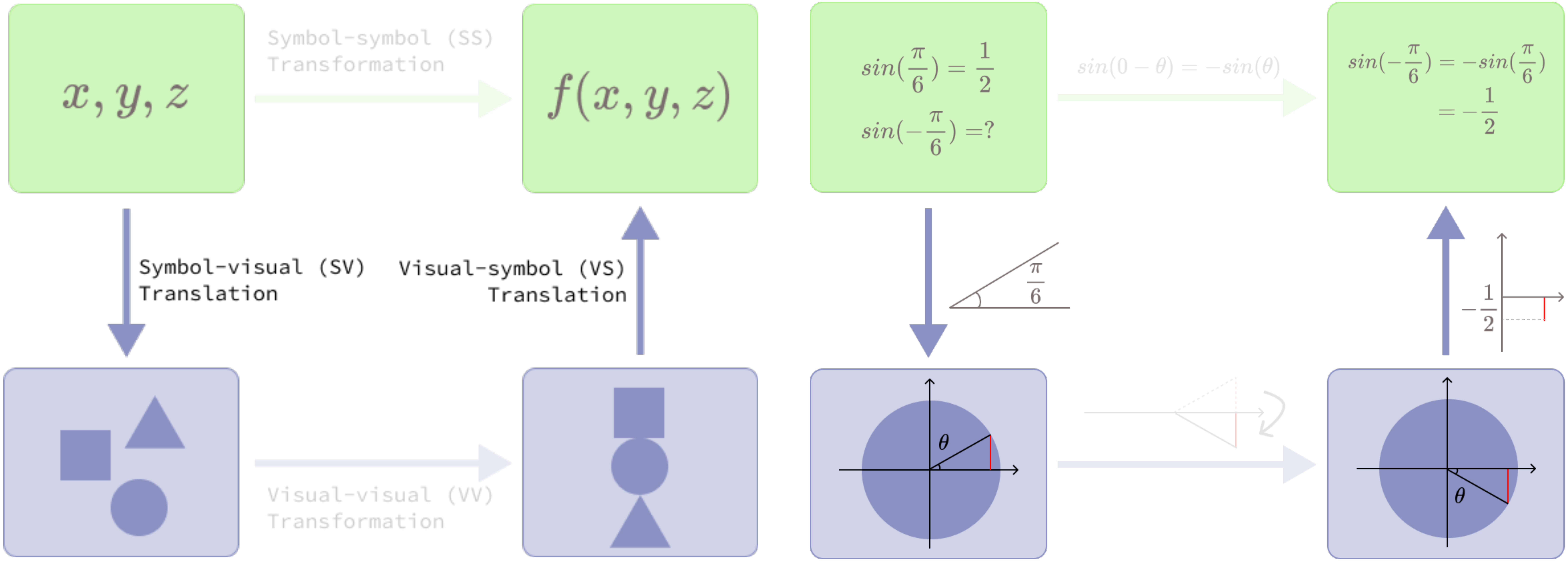
Understanding the diagramming process and encoding visual representations

Encoding visual representations in diagramming tools simplifies programming of interactive visual activities that provide students with automated feedback at scale.

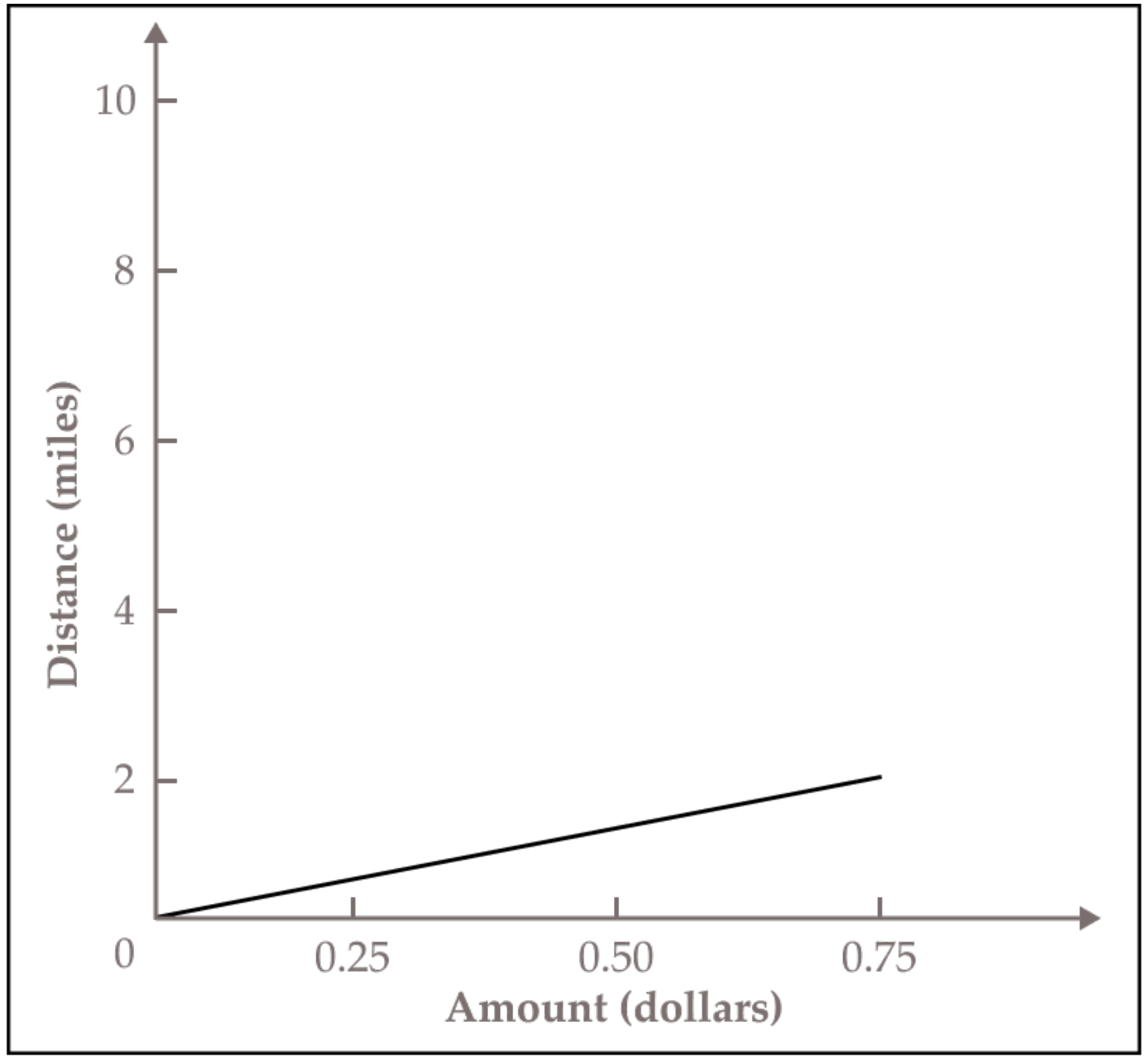
Edgeworth: Diagrammatic Content Authoring at Scale

(In progress)

Connecting symbolic and visual reps.



Translation problems ➡ Representational fluency

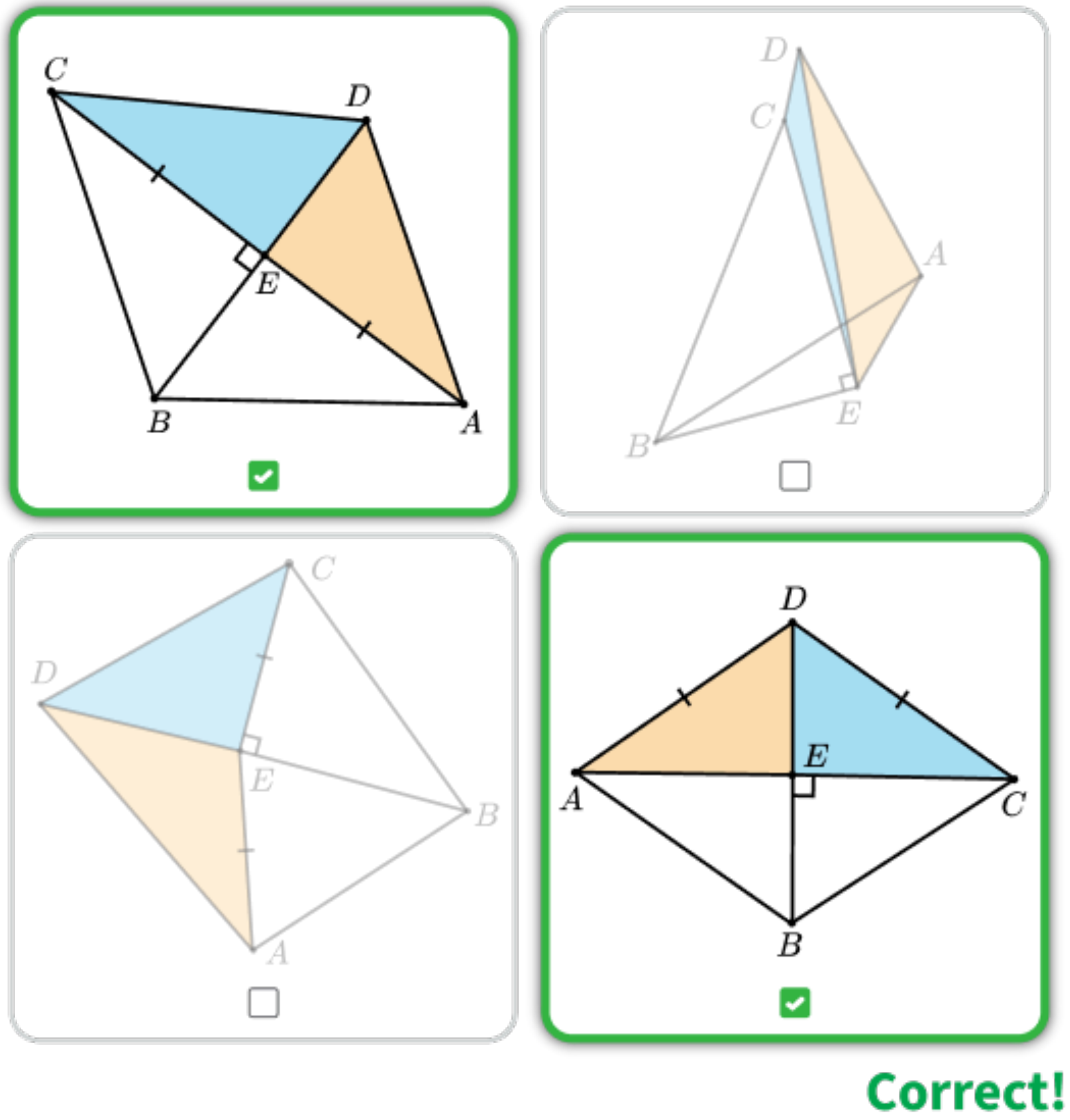


$y = \dfrac{-0.75}{2}$

$y = \dfrac{2}{0.75}x$

$y = \dfrac{0.75}{2}x$

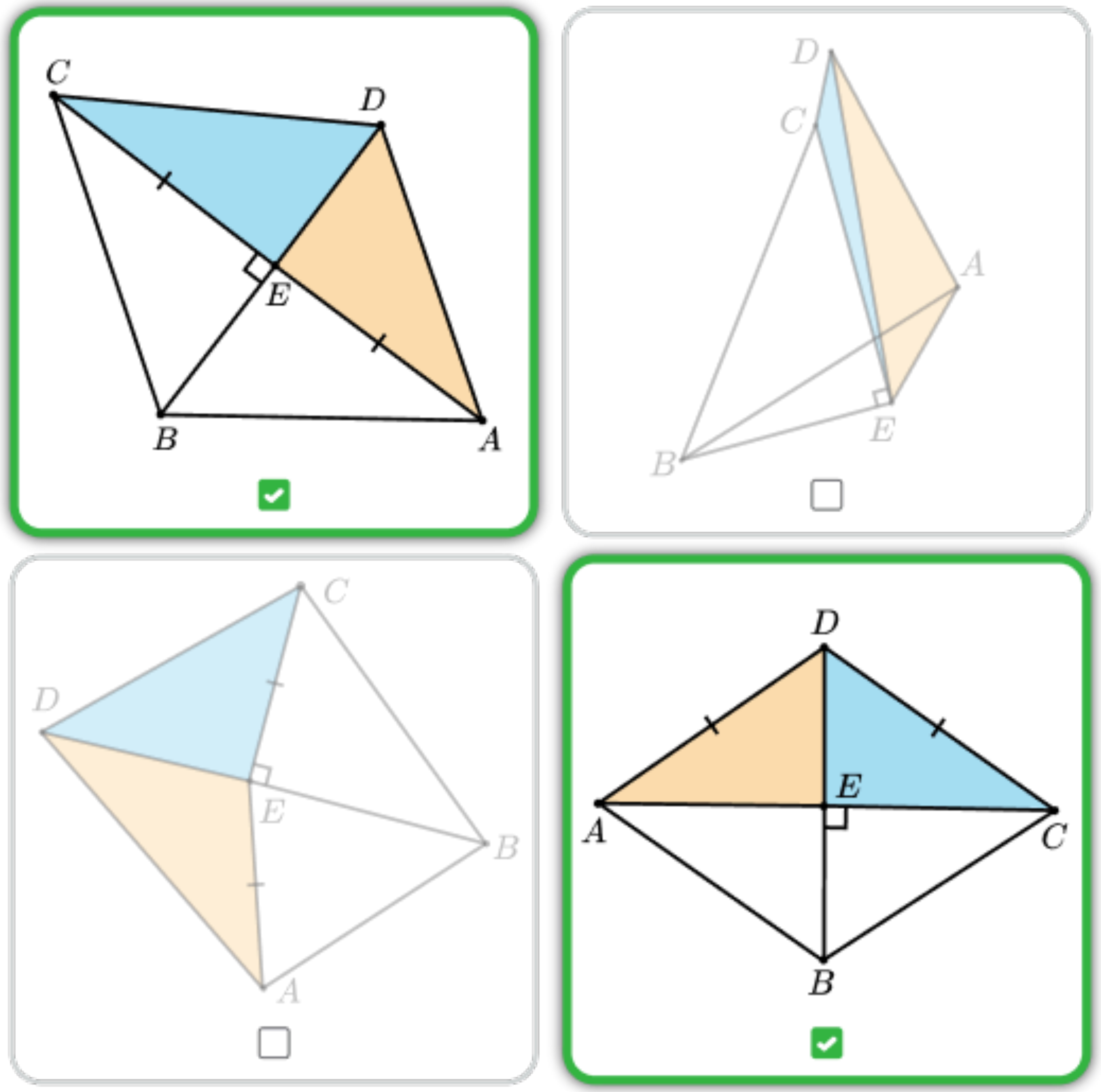
In which of the following diagrams are ΔCED and ΔAED congruent?



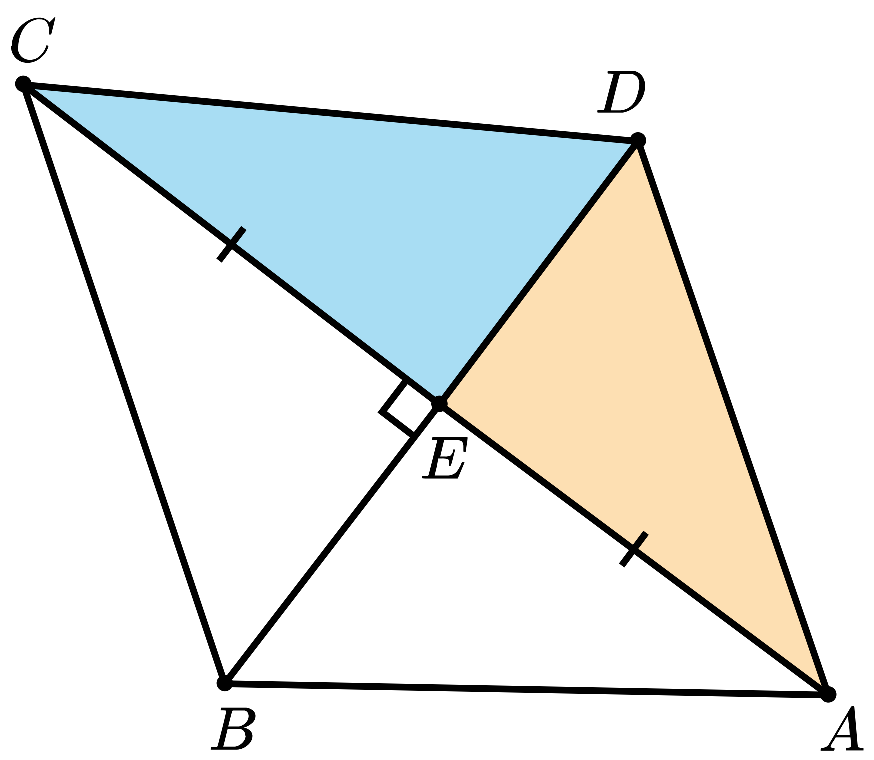
Kellman et al. 2020

Contrasting cases

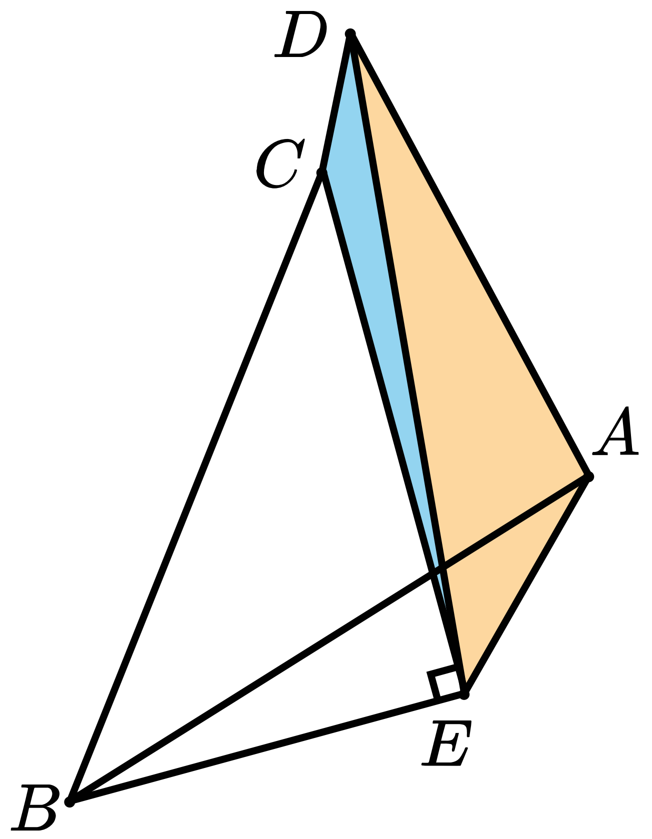
In which of the following diagrams are ΔCED and ΔAED congruent?



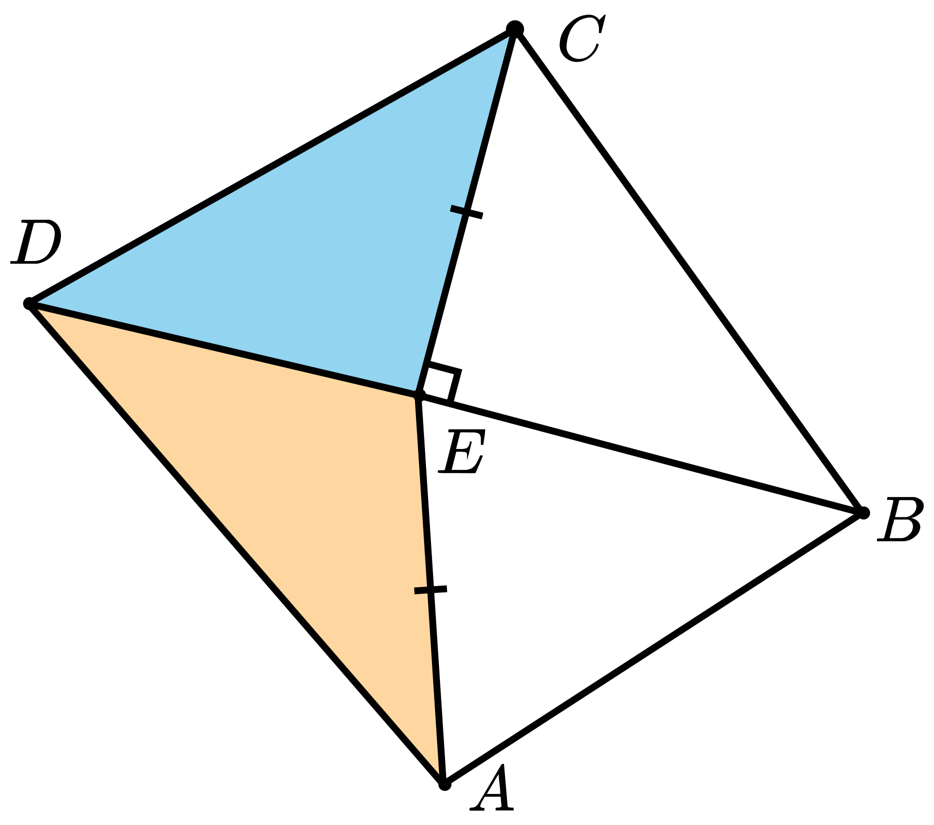
Correct!



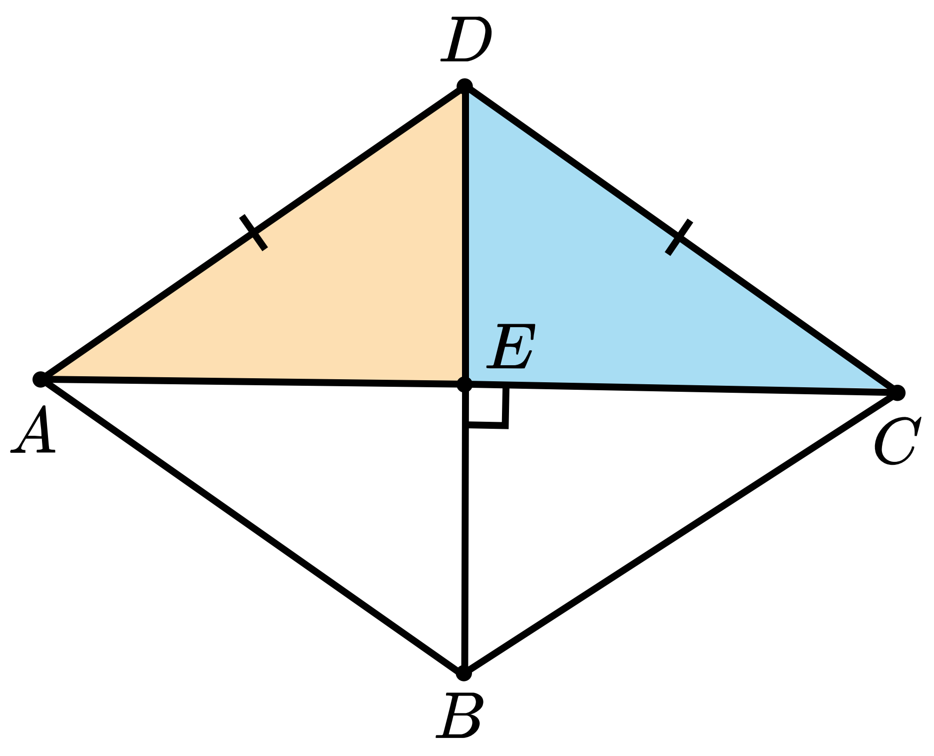
Correct - general position



Incorrect - random

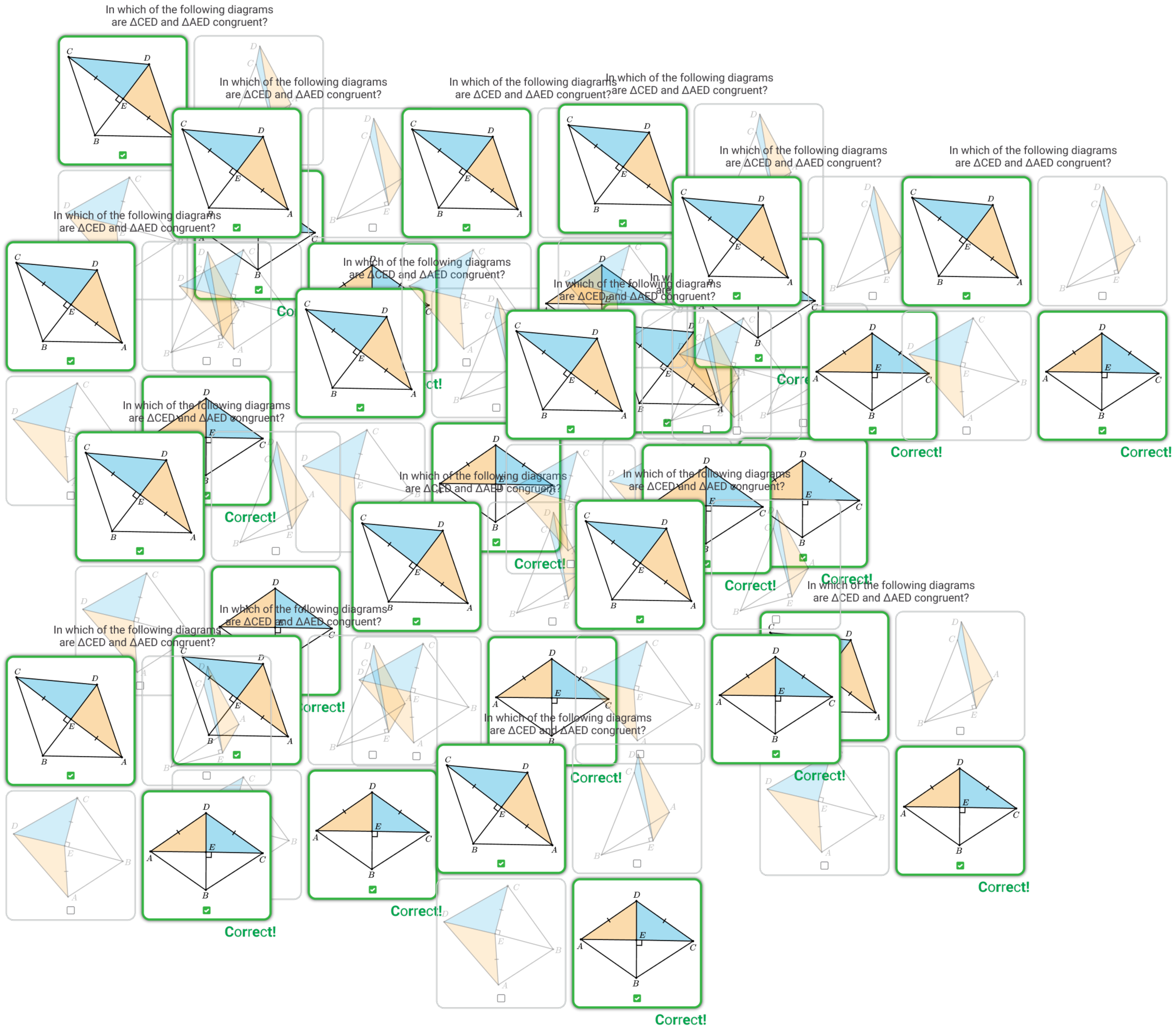


Incorrect - distractor



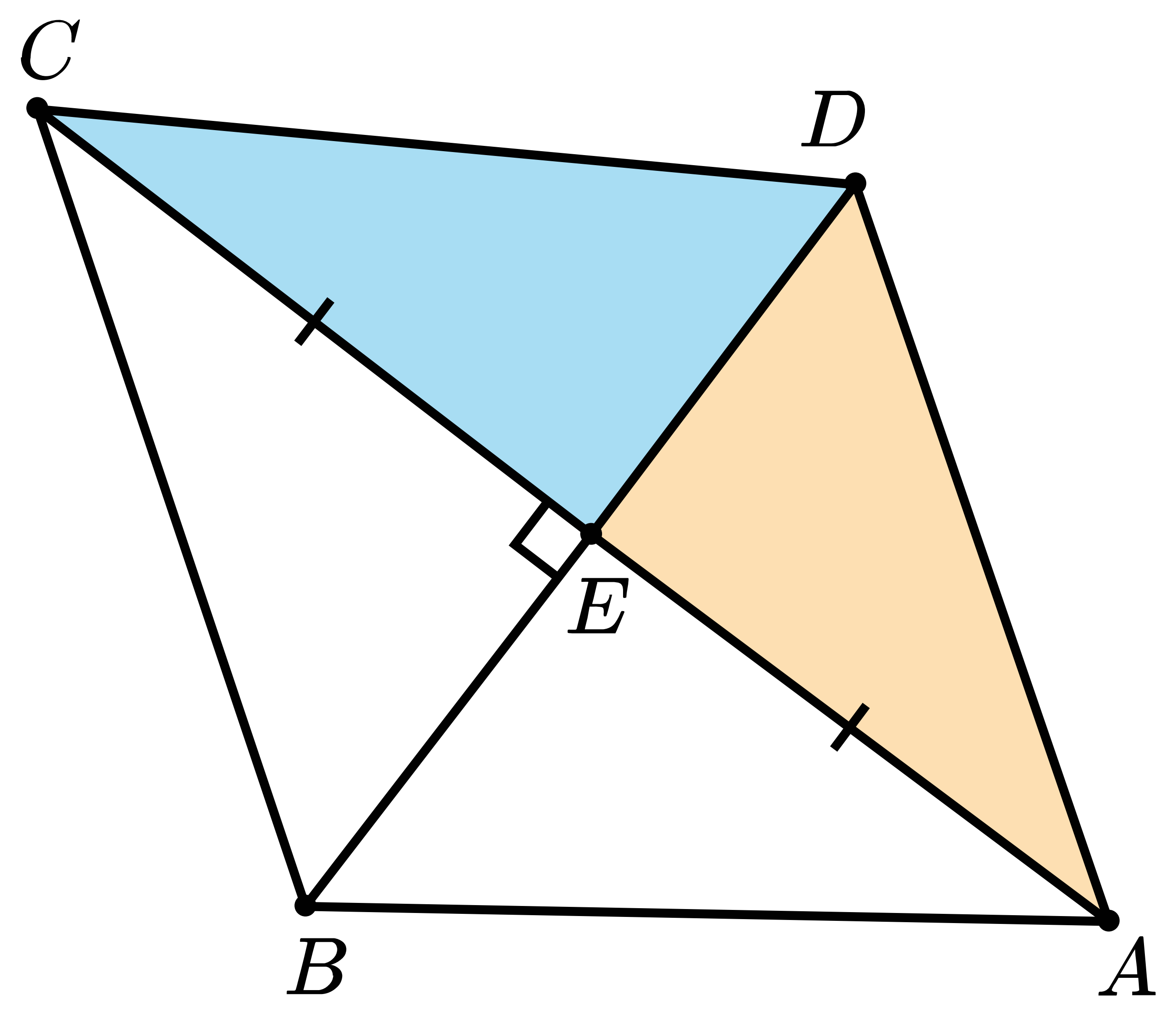
Correct - special cases

Contrasting + corrective feedback ➡ Sensemaking



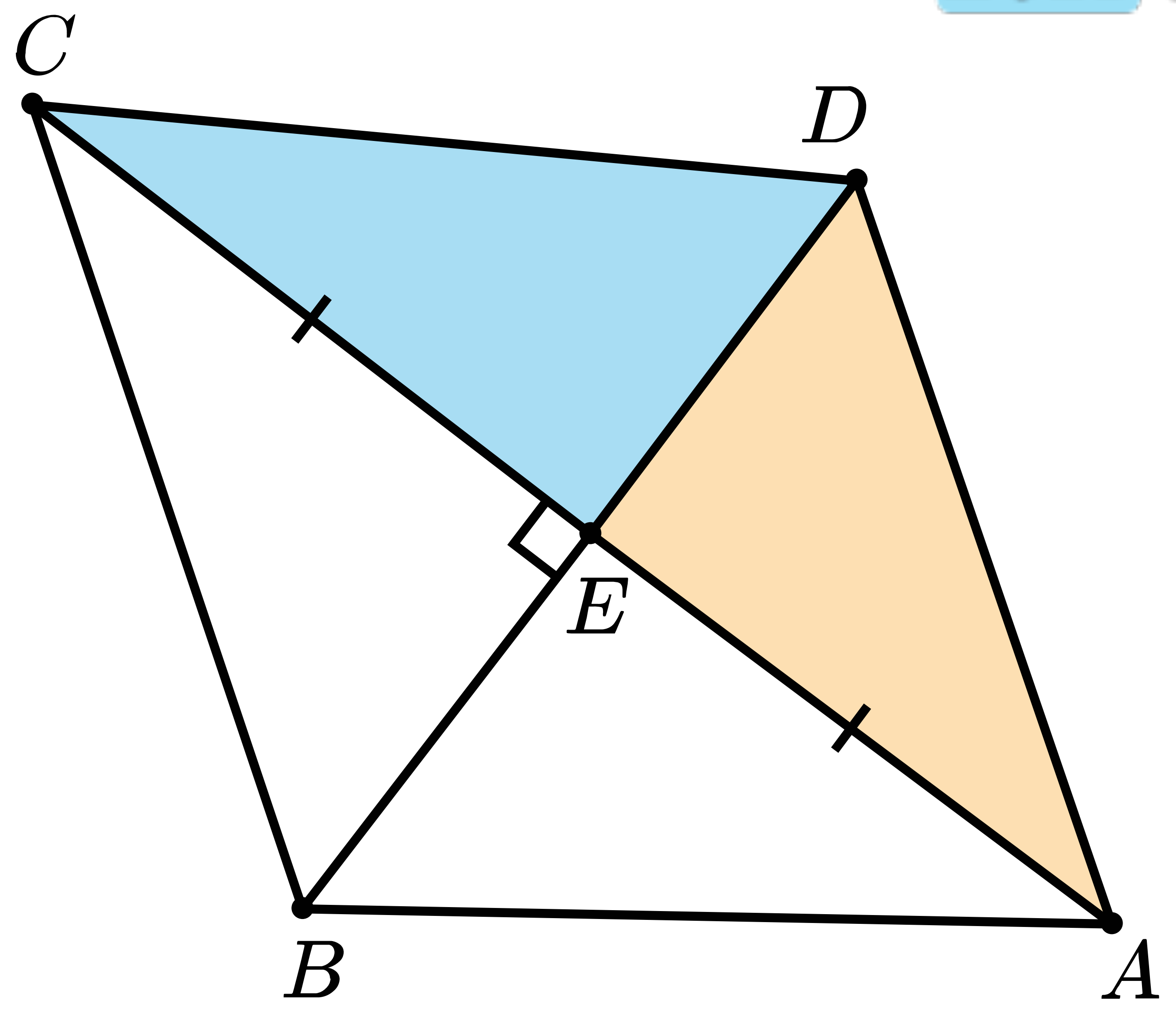
Kellman et al. 2020

How do we get more diagrams?



Let's use the Substance program

In which of the following diagrams are
ΔCED and ΔAED congruent?

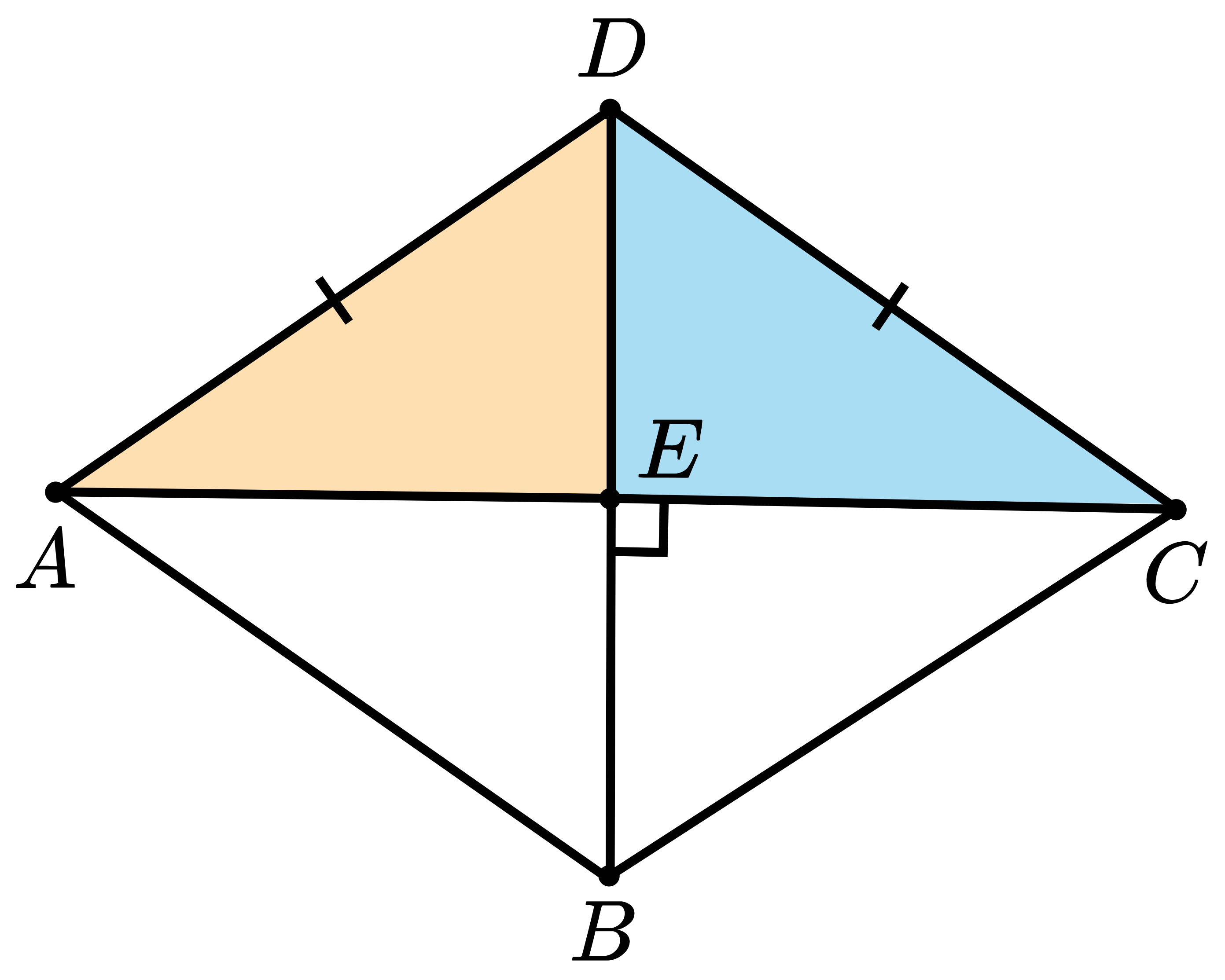



```
Point A, B, C, D, E
Triangle CED, DEA, CEA, BEA
Collinear(D, E, B)
Collinear(C, E, A)
RightMarked(a_CEB)
EqualLengthMarked(CE, EA)
```

Side-Angle-Side Rule

Make a more special correct answer

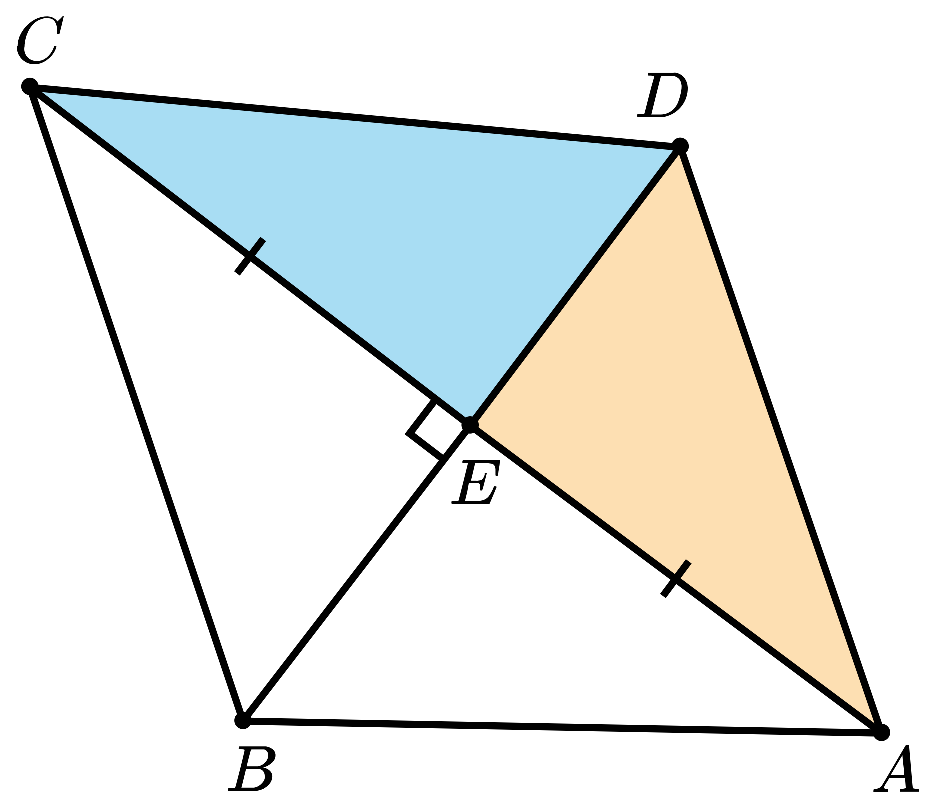
In which of the following diagrams are
ΔCED and ΔAED congruent?




```
Point A, B, C, D, E
Triangle CED, DEA, CEA, BED
Collinear(D, E, B)
Collinear(C, E, A)
RightMarked(a_CEB)
EqualLengthMarked(AD, DC)
```

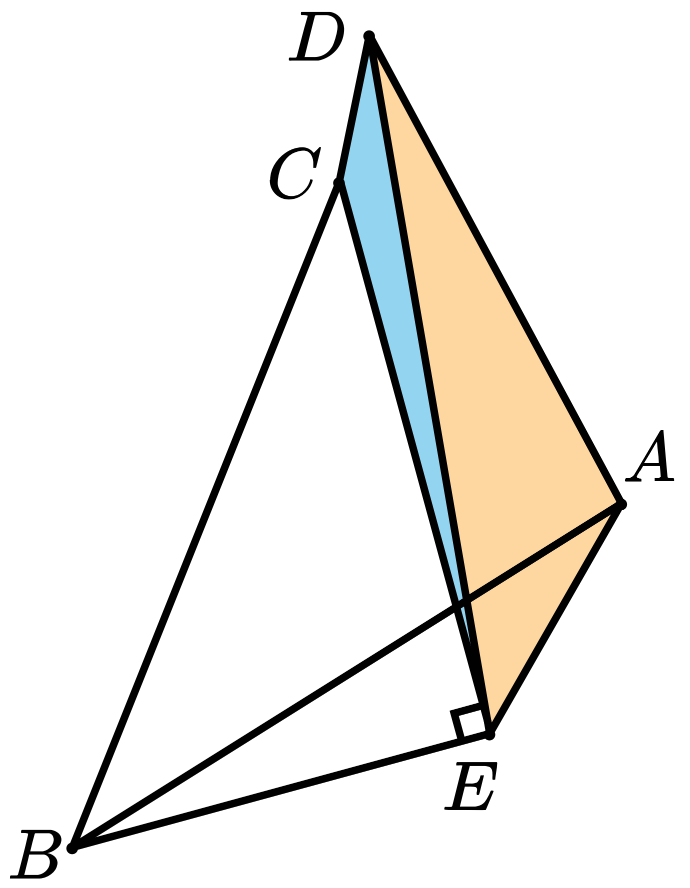
Hypotenuse-Leg Rule

Edgeworth: Authoring Diagrammatic Problems using Program Mutation



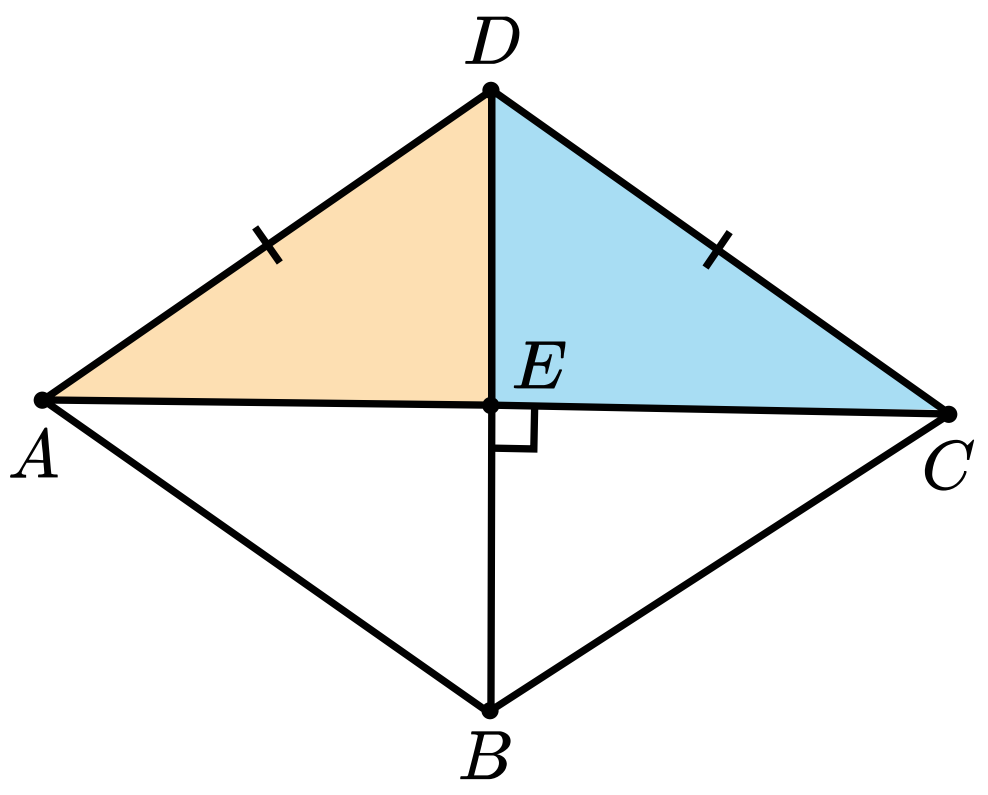
Correct - general position

Point A, B, C, D, E
Triangle CED, DEA, CEA, BEA
Collinear(D, E, B)
Collinear(C, E, A)
RightMarked(a_CEB)
EqualLengthMarked(CE, EA)



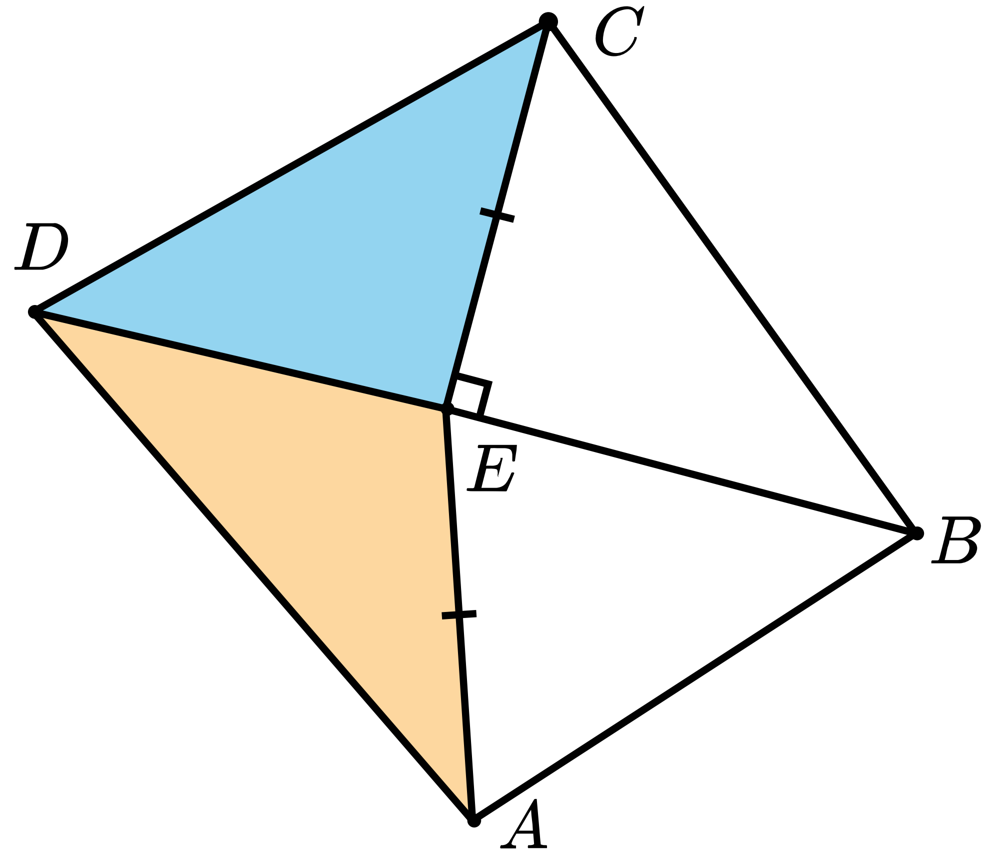
Incorrect - random

Point A, B, C, D, E
Triangle CED, DEA, CEA, BED
~~Collinear(D, E, B)~~
~~Collinear(C, E, A)~~
RightMarked(a_CEB)
~~EqualLengthMarked(CE, EA)~~



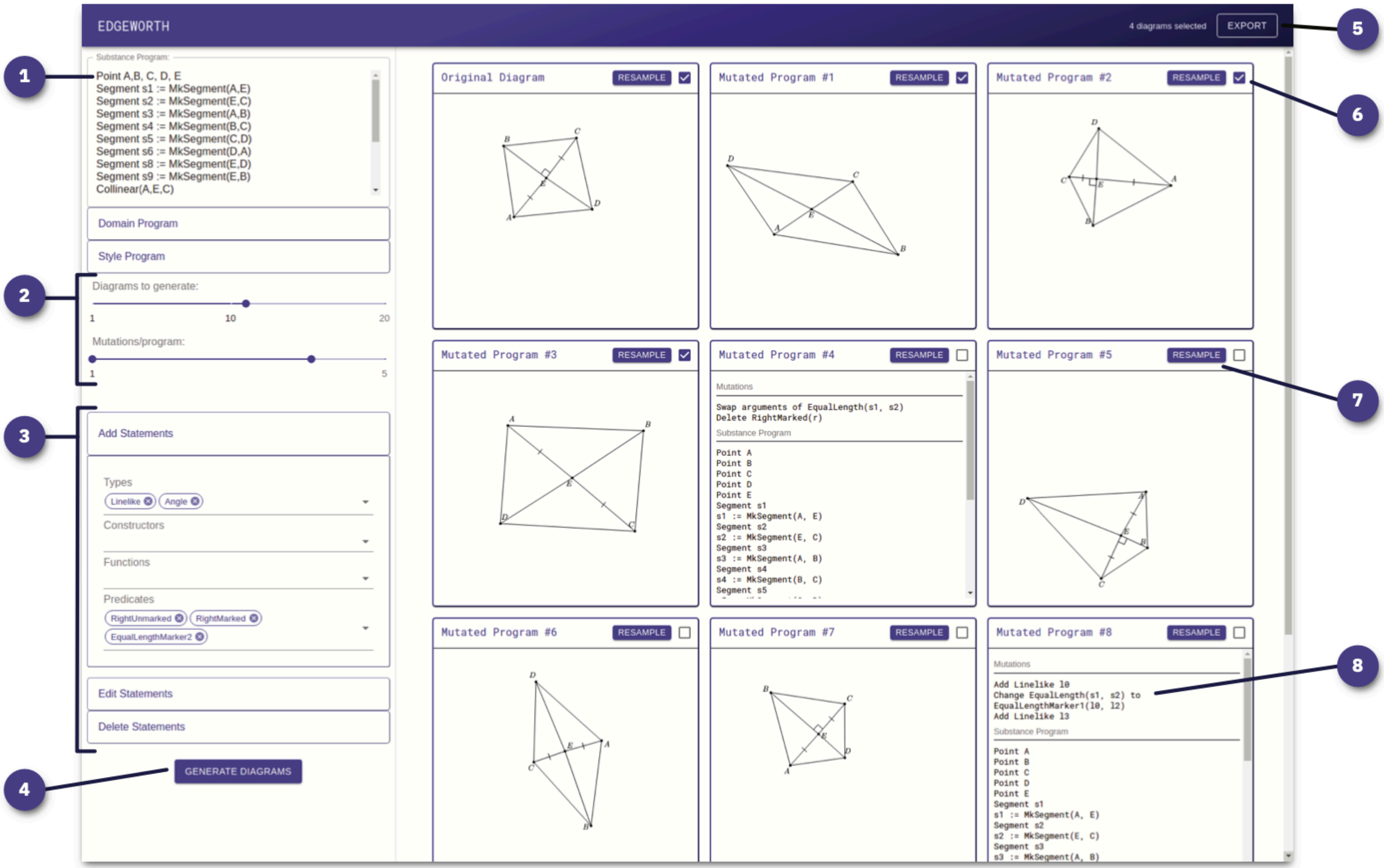
Correct - special cases

Point A, B, C, D, E
Triangle CED, DEA, CEA, BED
Collinear(D, E, B)
Collinear(C, E, A)
RightMarked(a_CEB)
EqualLengthMarked(AD, DC)



Incorrect - distractor

Point A, B, C, D, E
Triangle CED, DEA, CEA, BED
Collinear(D, E, B)
~~Collinear(C, E, A)~~
RightMarked(a_CEB)
EqualLengthMarked(CE, EA)
Acute(a_AEB)



1 Input source program

2 Adjust mutation parameters

3 Select statements of interest

4 Generate batch of diagrams

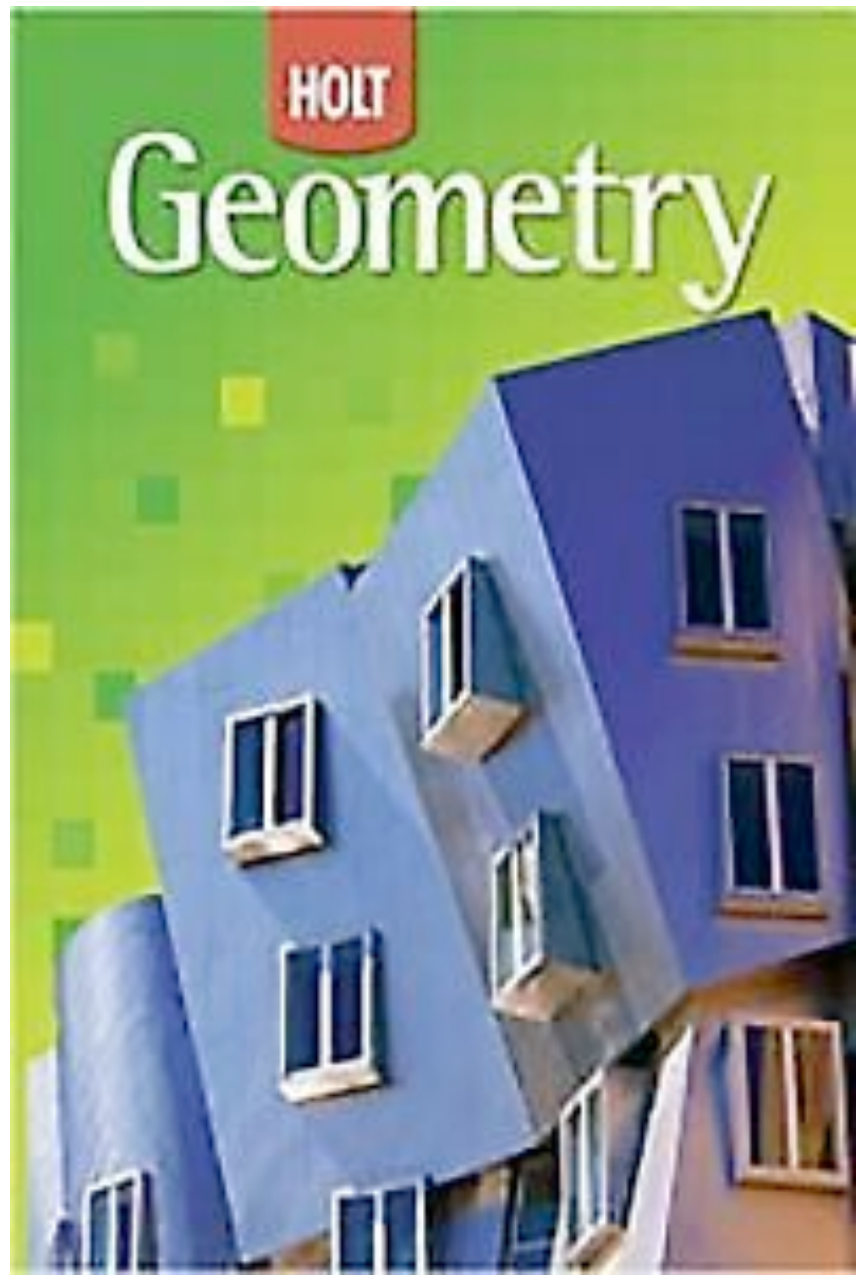
5 Export selected diagrams as SVGs

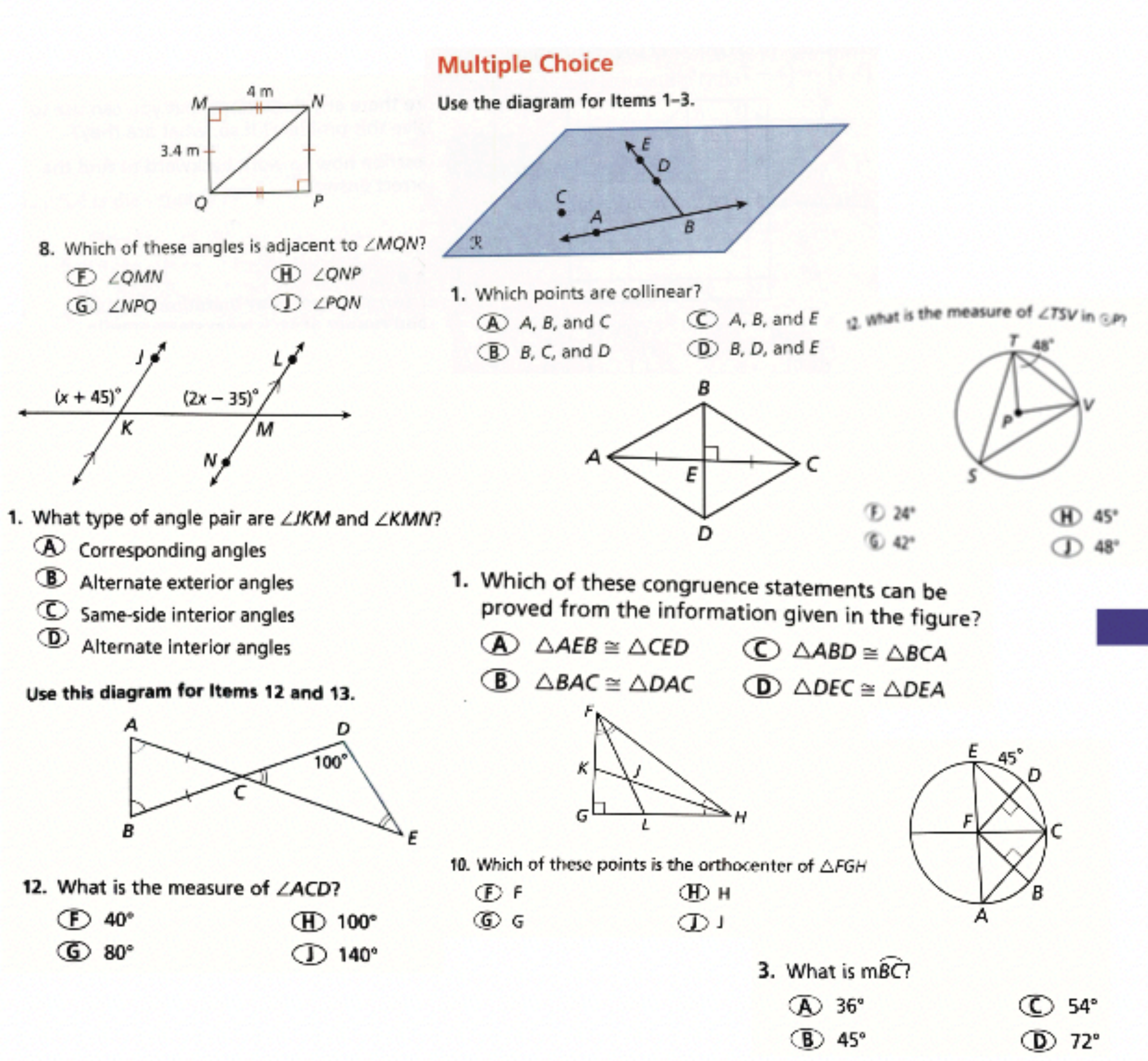
6 Select diagram to export

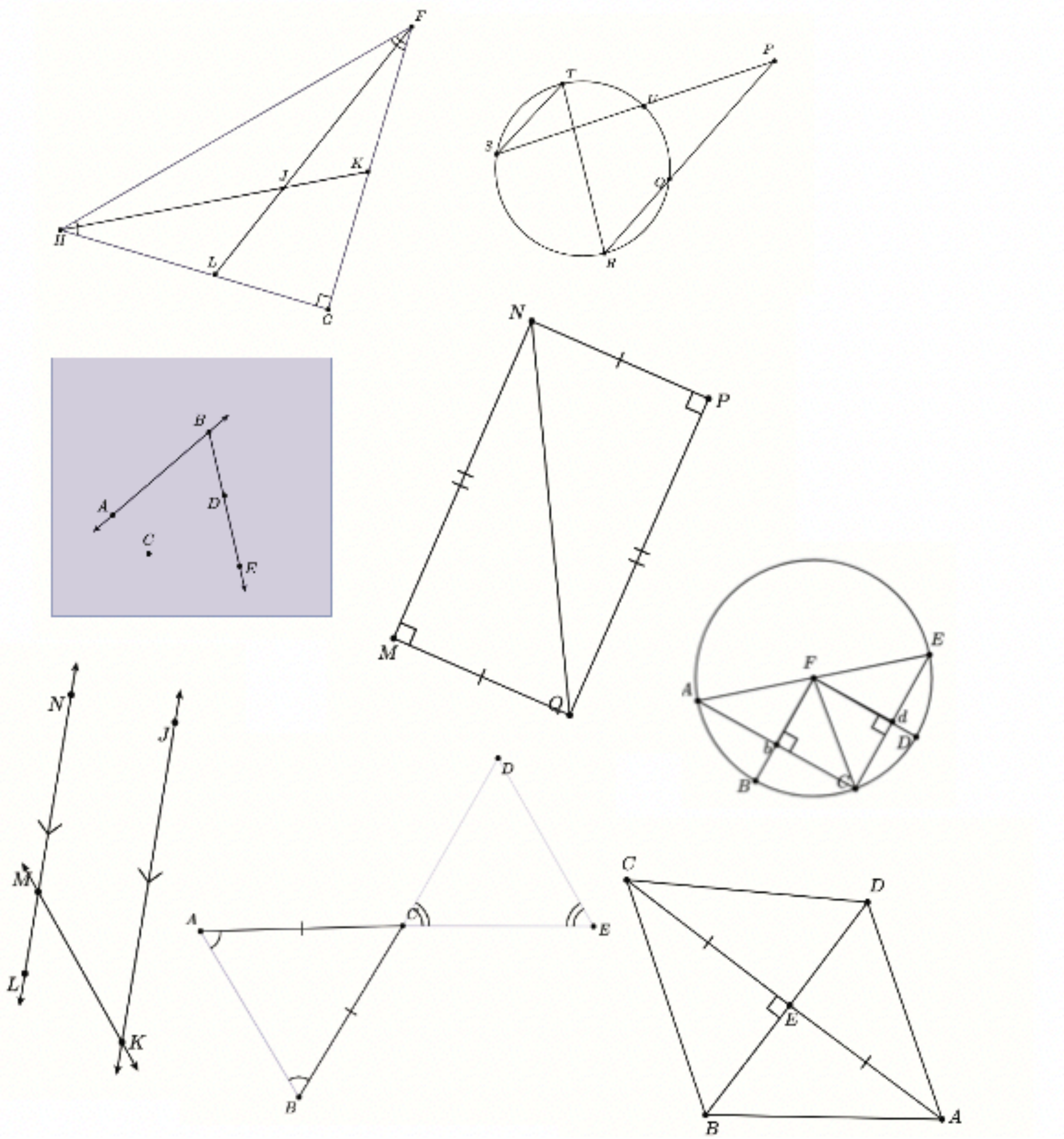
7 Resample diagram layout

8 Specific mutations applied

Prelim. Evaluation: re-creating textbook problems



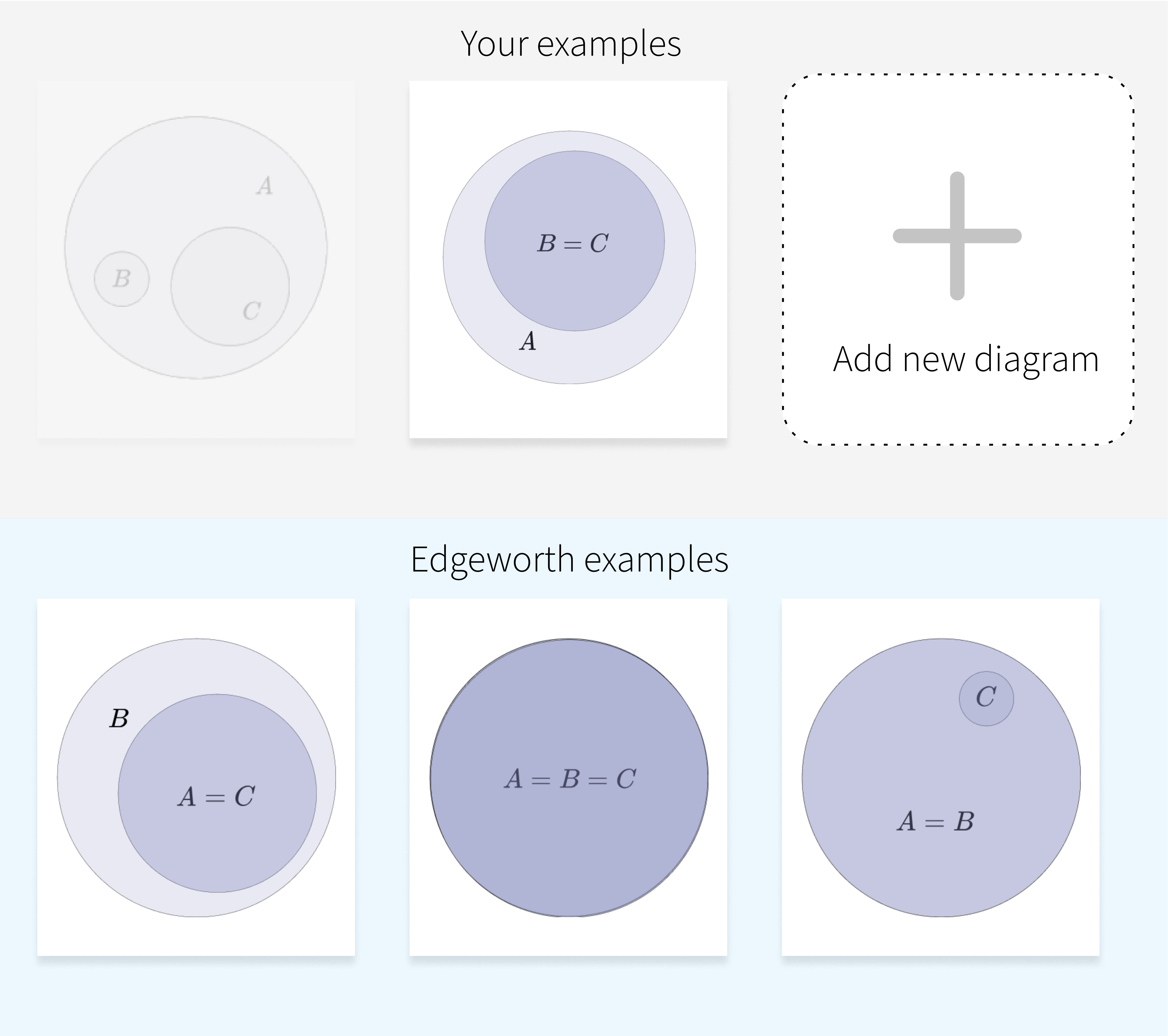




Synthesis-driven authoring

Proposed

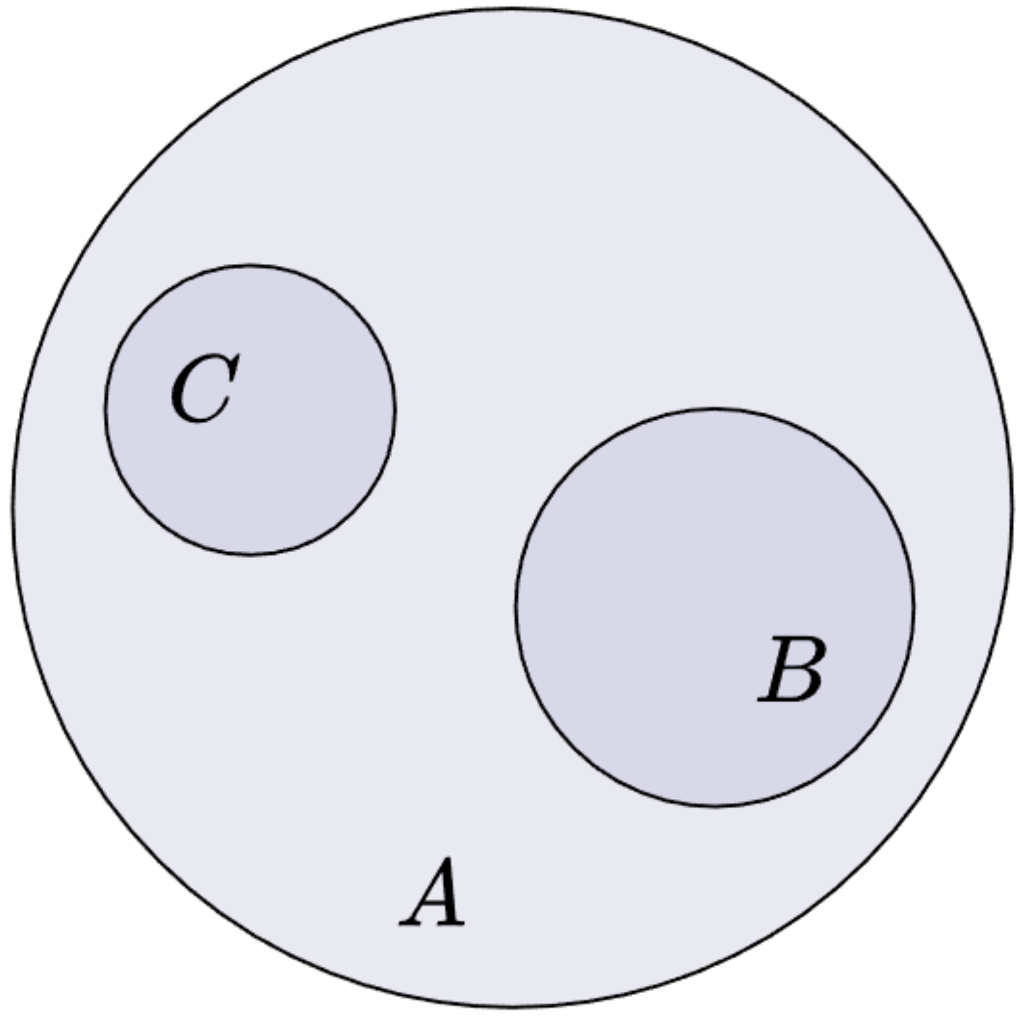
```
Set A, B, C
IsSubset(B, A)
IsSubset(C, A)
Equal(B, C)
```



Measuring semantic consistency

Proposed

Original

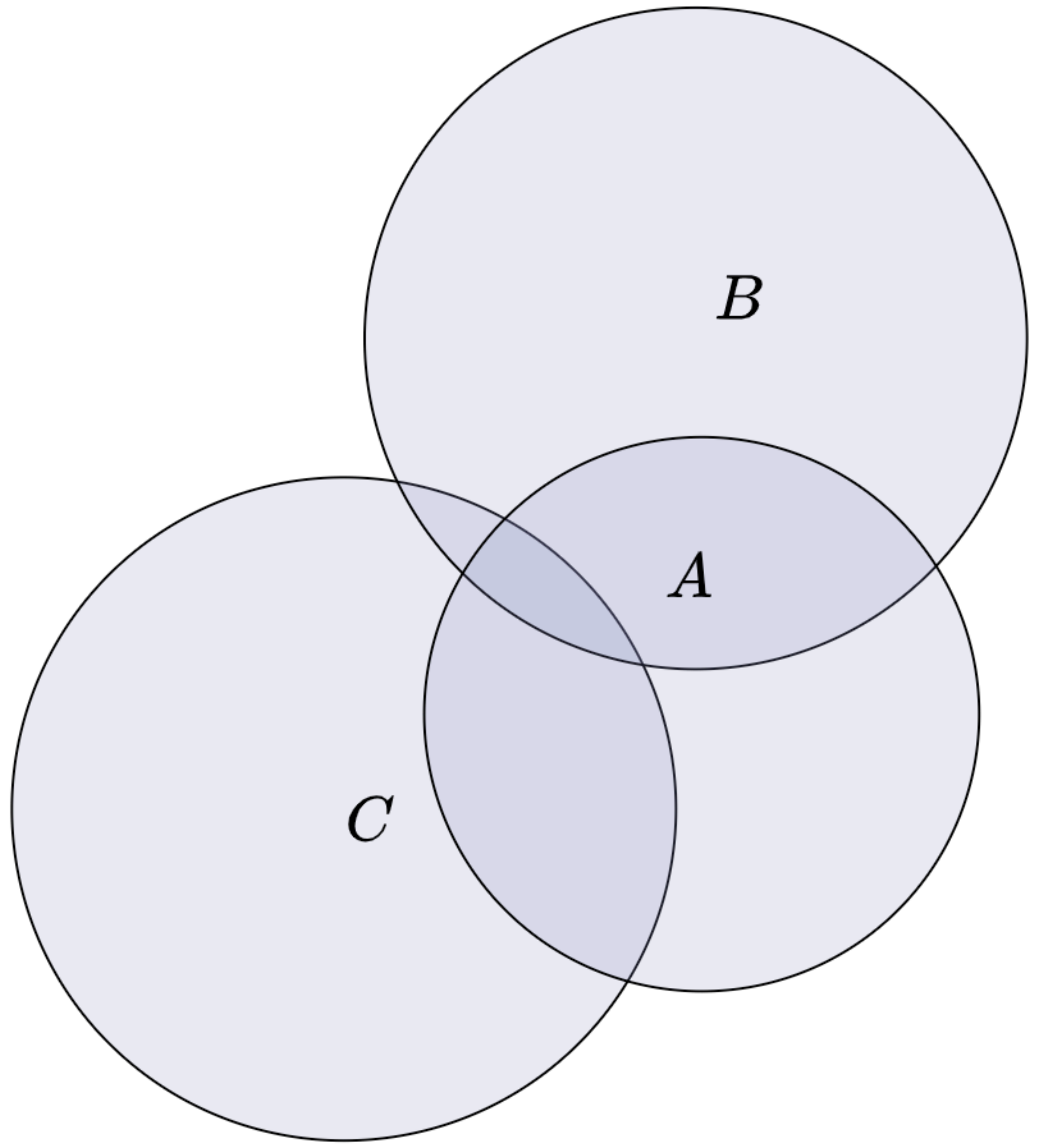


Energy: `0`

A contains B

A contains C

Mutant #7

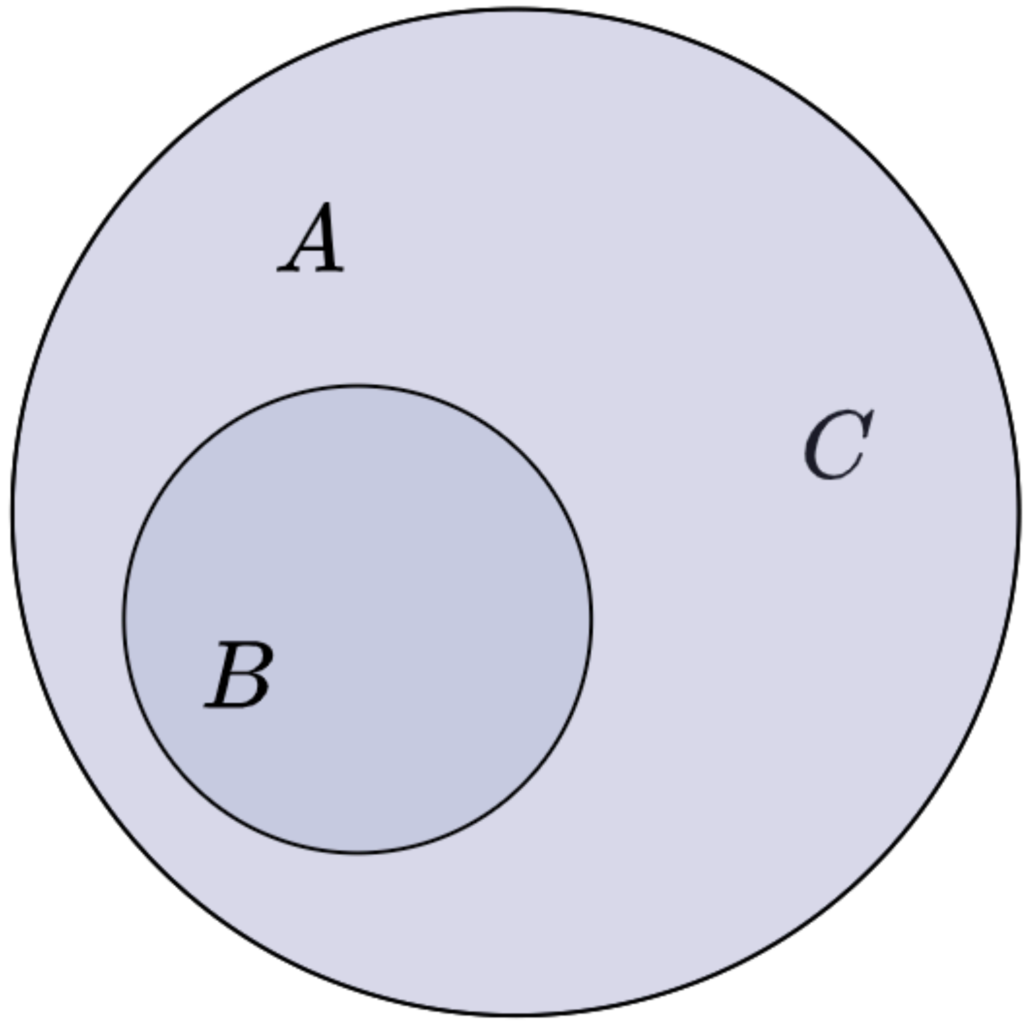


Energy: `78480519`

A contains B

A contains C

Mutant #12

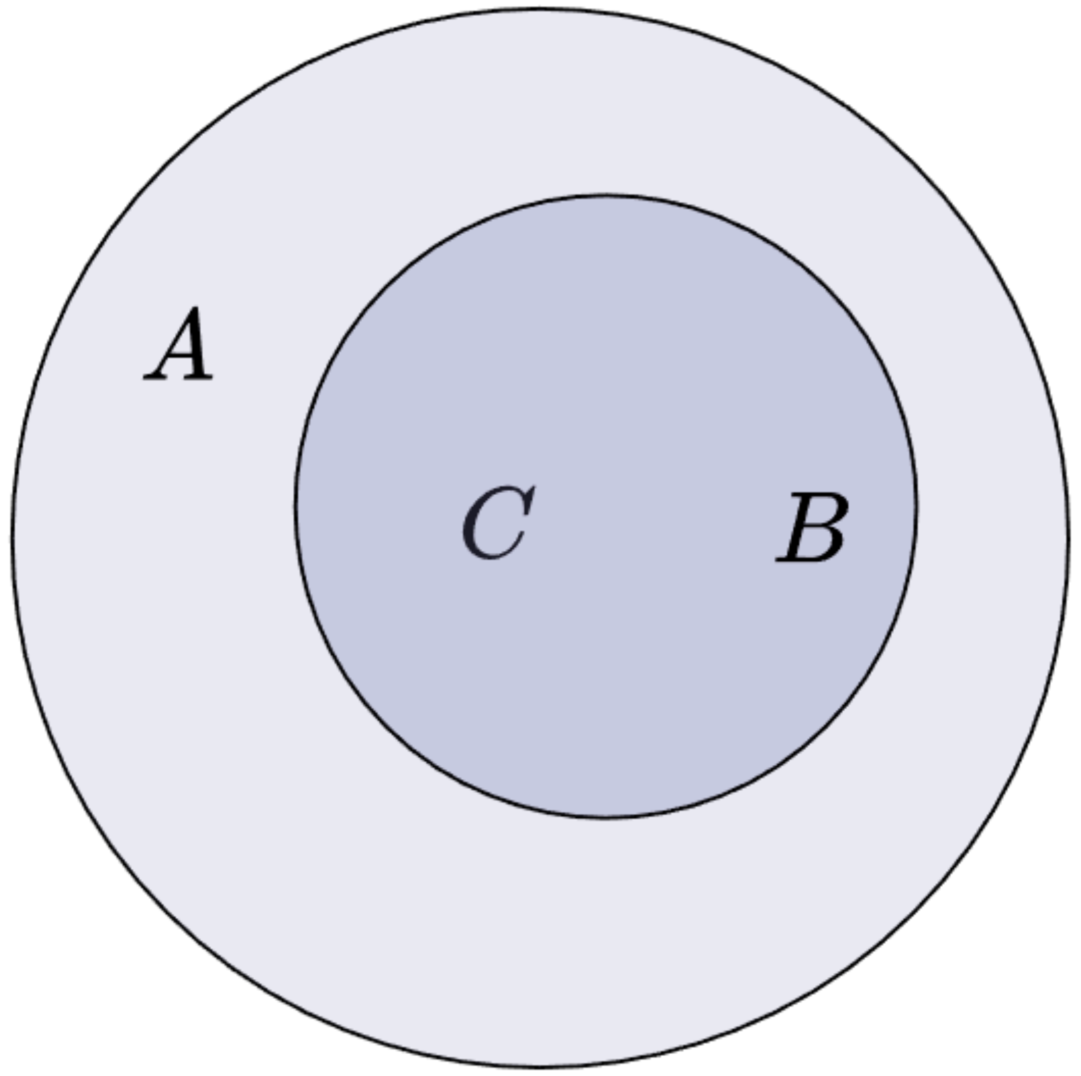


Energy: `6800602`

A contains B

A contains C

Mutant #30

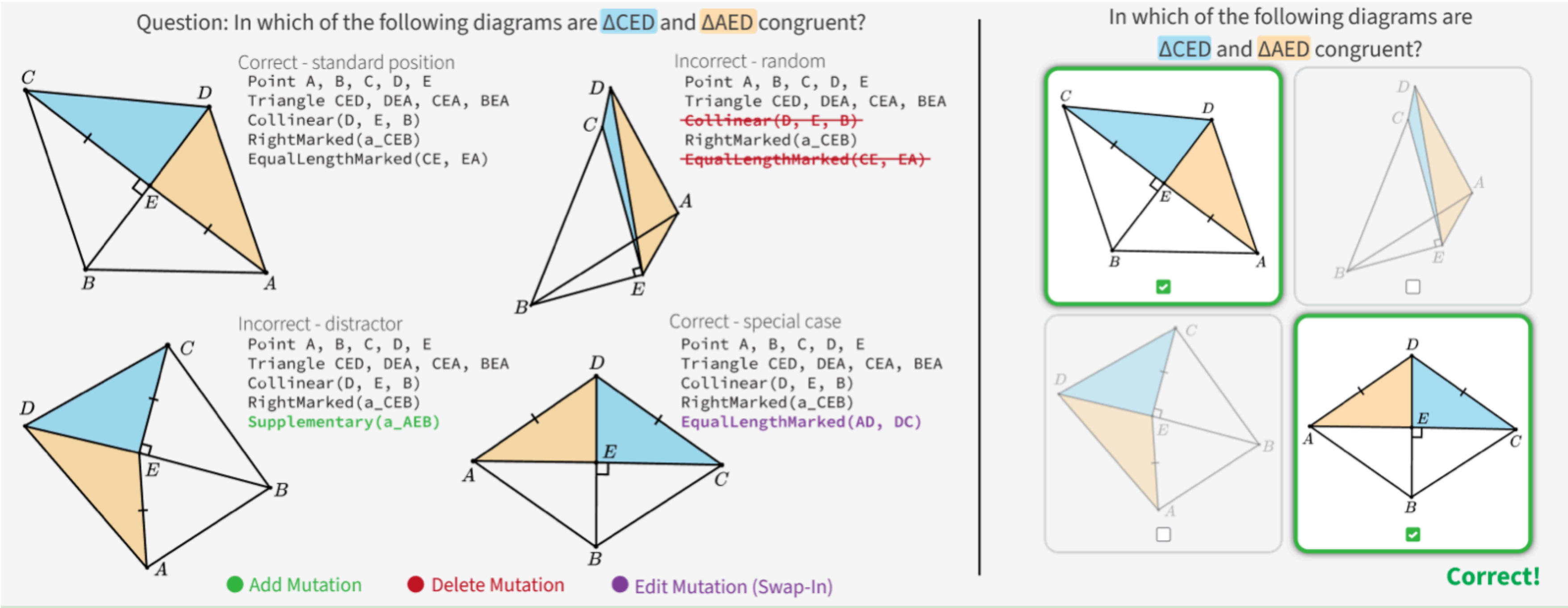


Energy: `0`

A contains B

A contains C

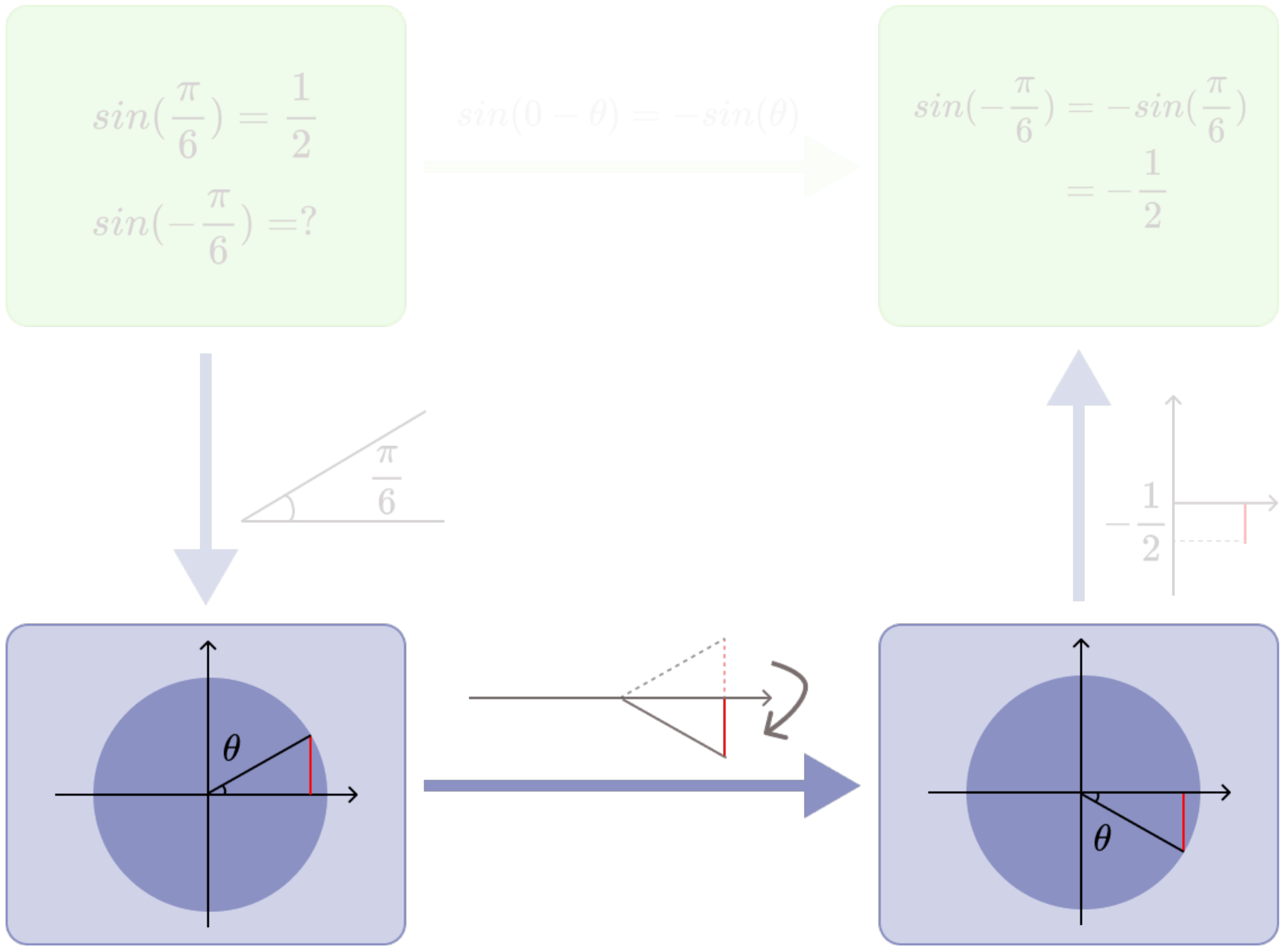
Encoding visual representations in diagramming tools simplifies programming of interactive visual activities that provide students with automated feedback **at scale**.

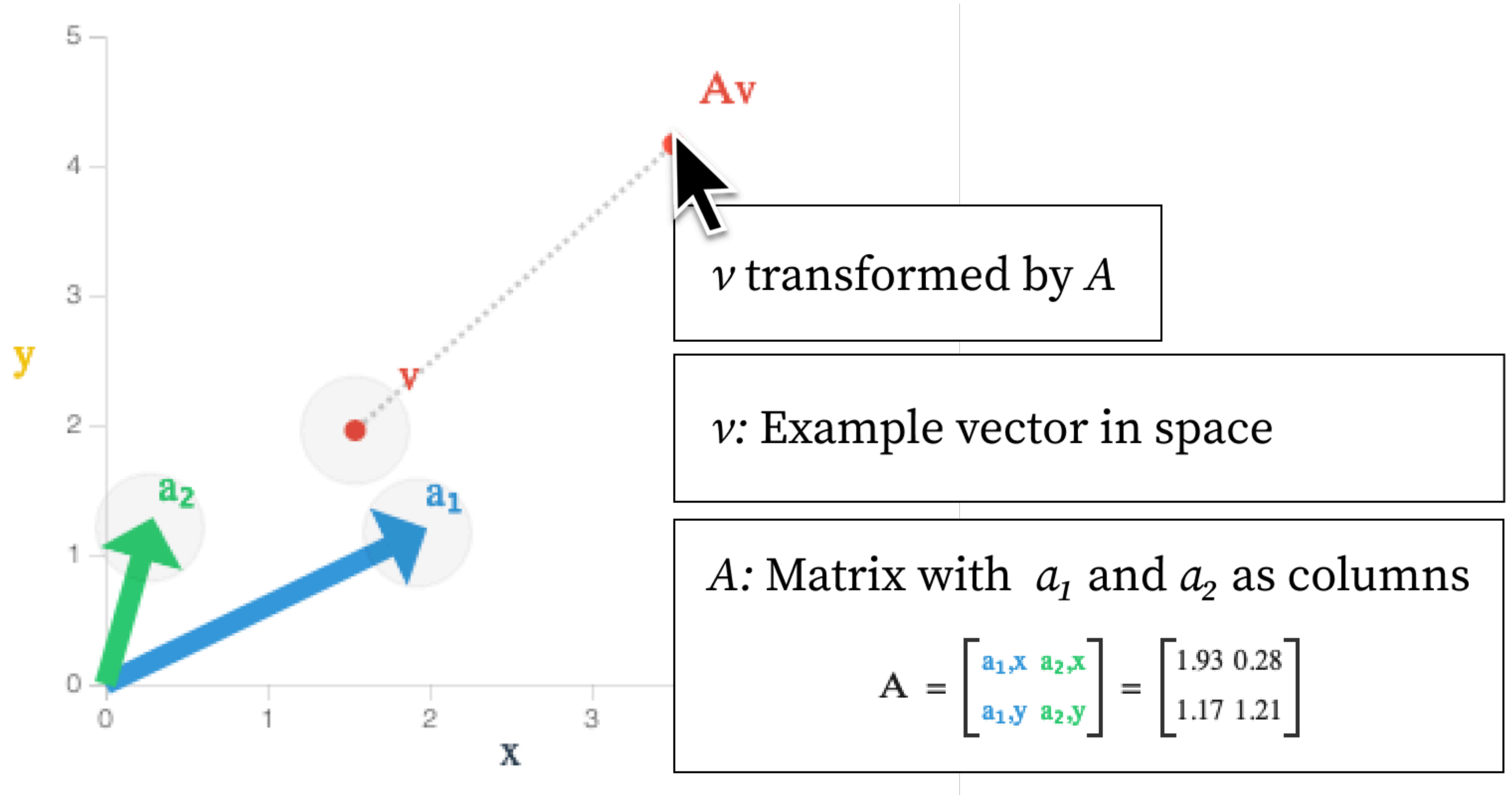


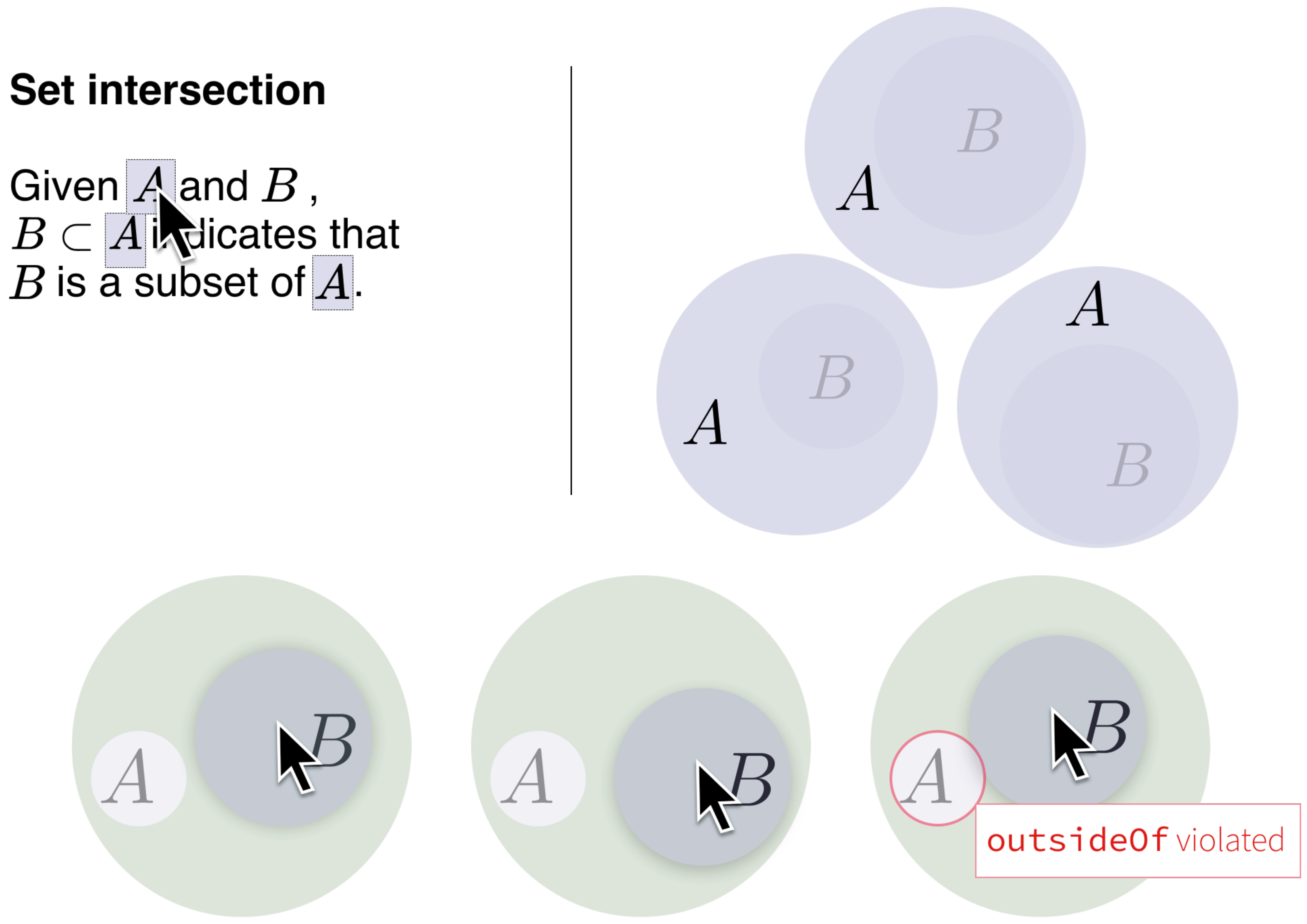
Edgeworth: Diagrammatic Content Authoring at Scale

(In progress)

Interacting with visual representations





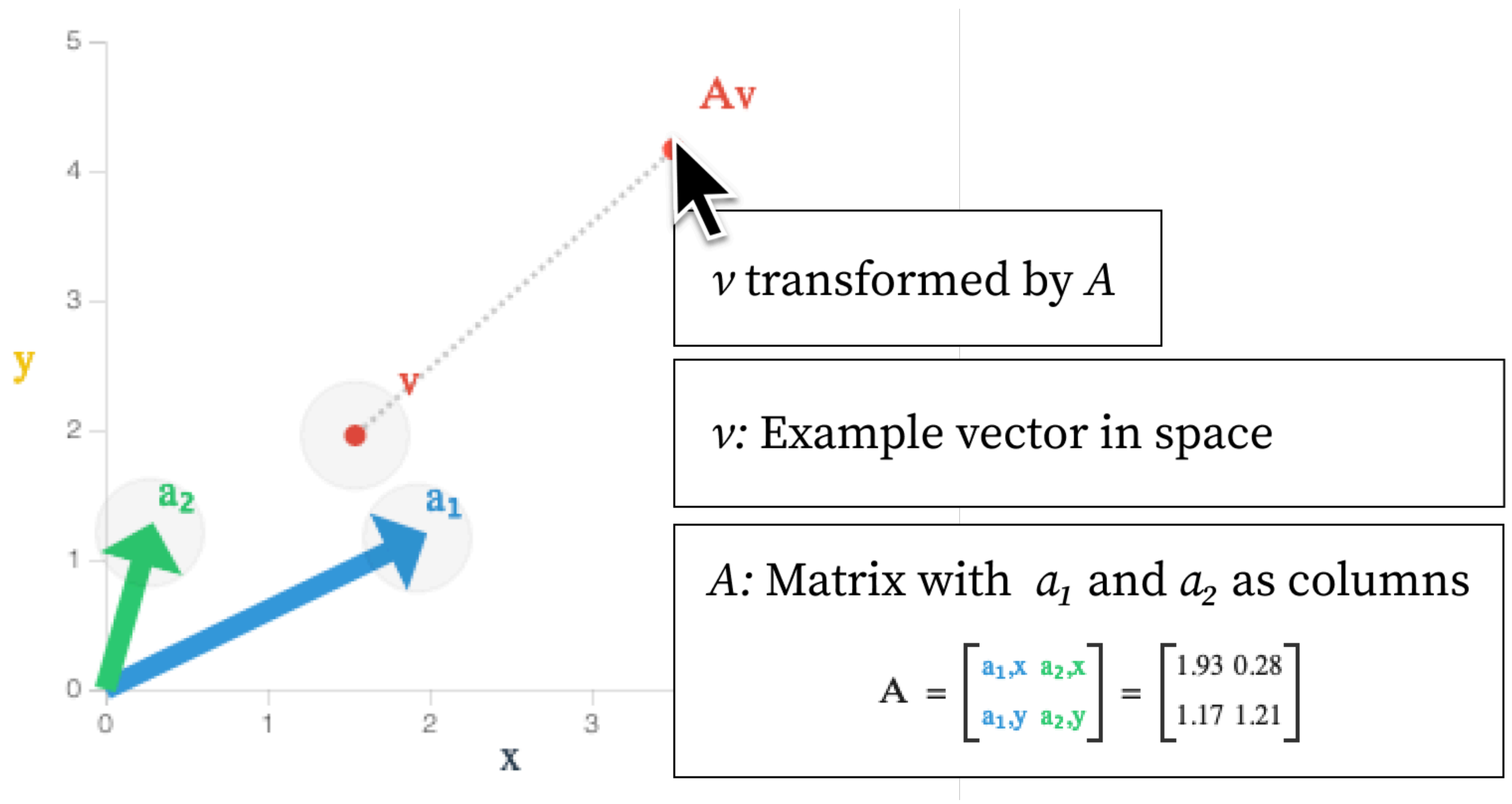


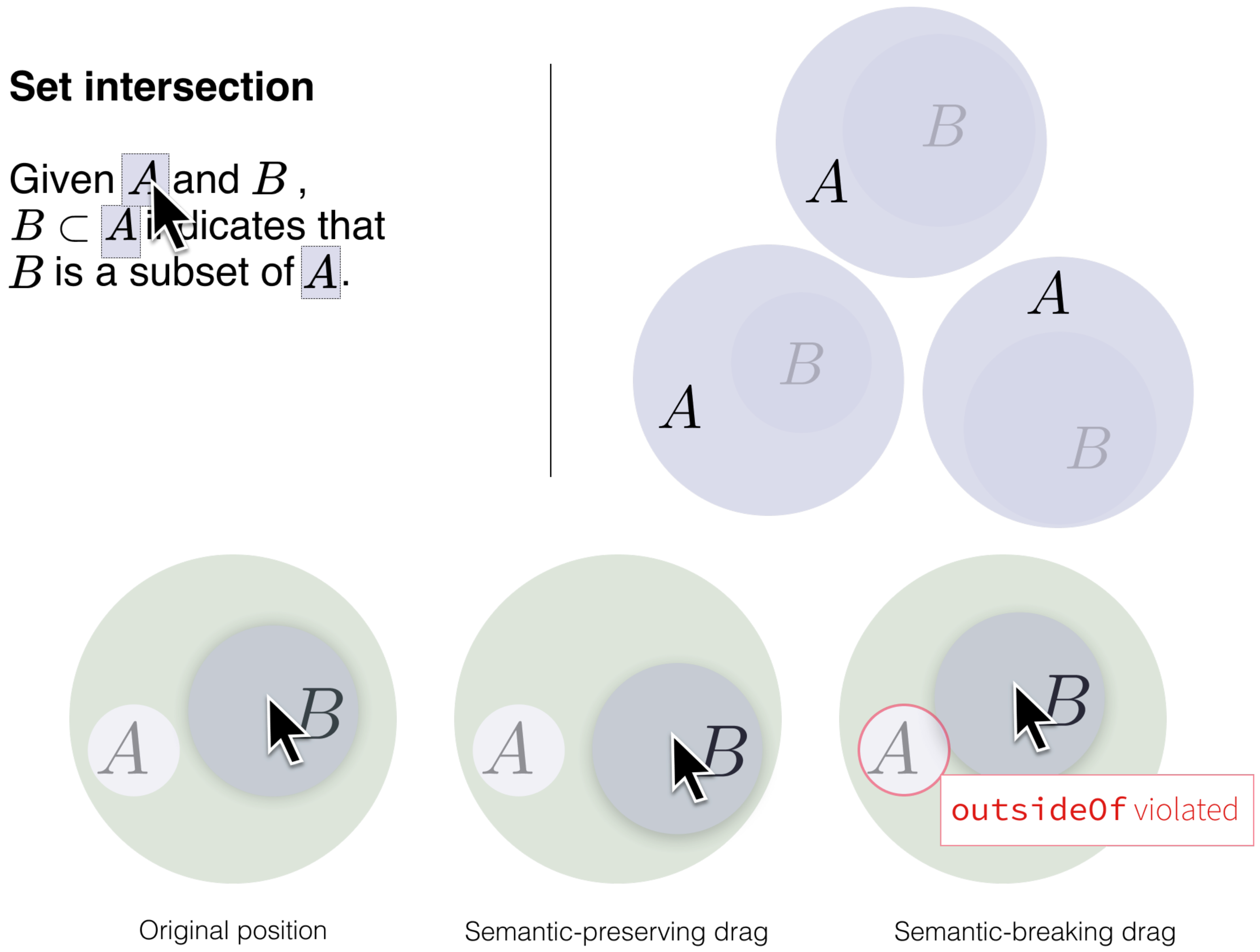
Original position

Semantic-preserving drag

Semantic-breaking drag

From encoding to semantic-preserving interactivity

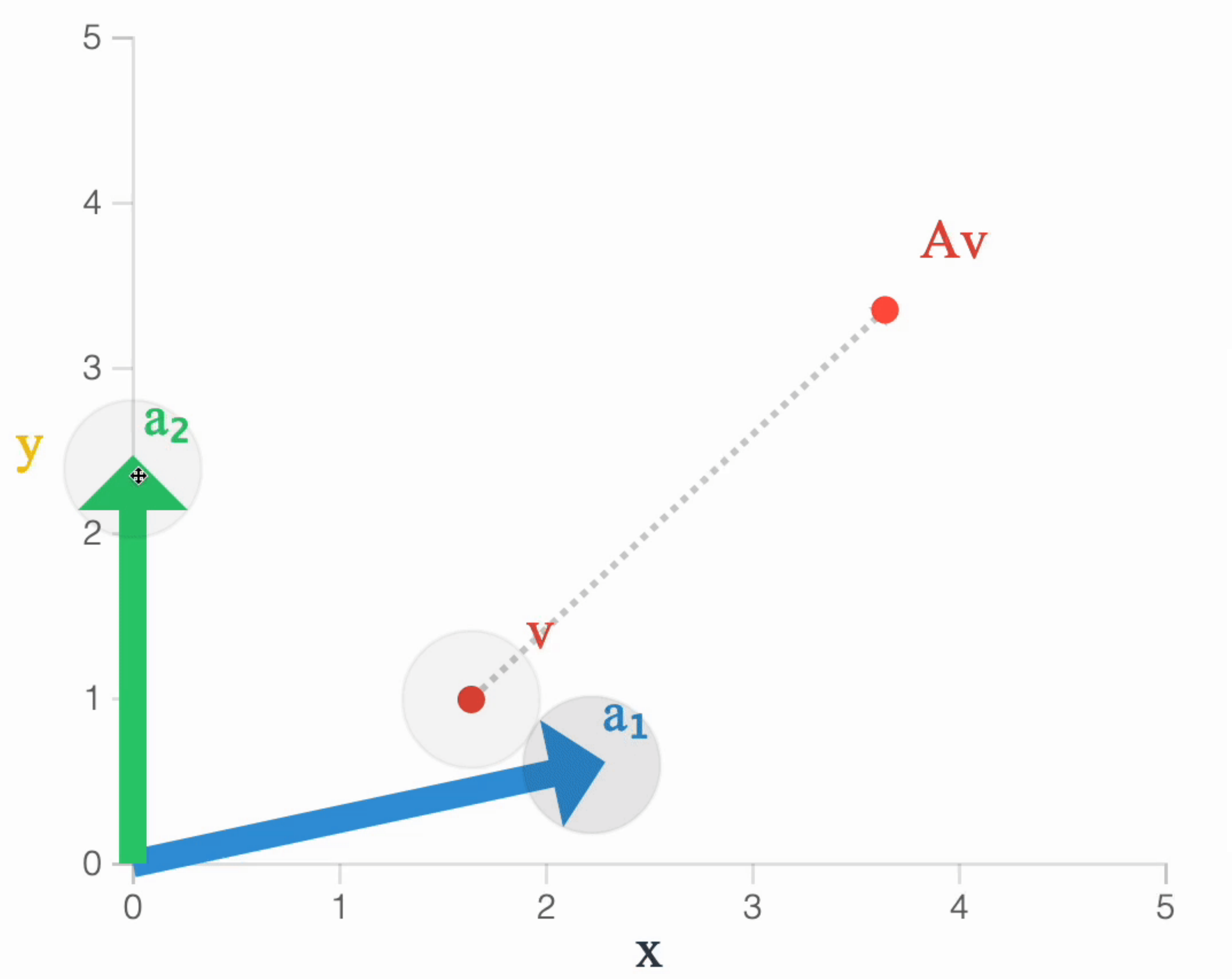


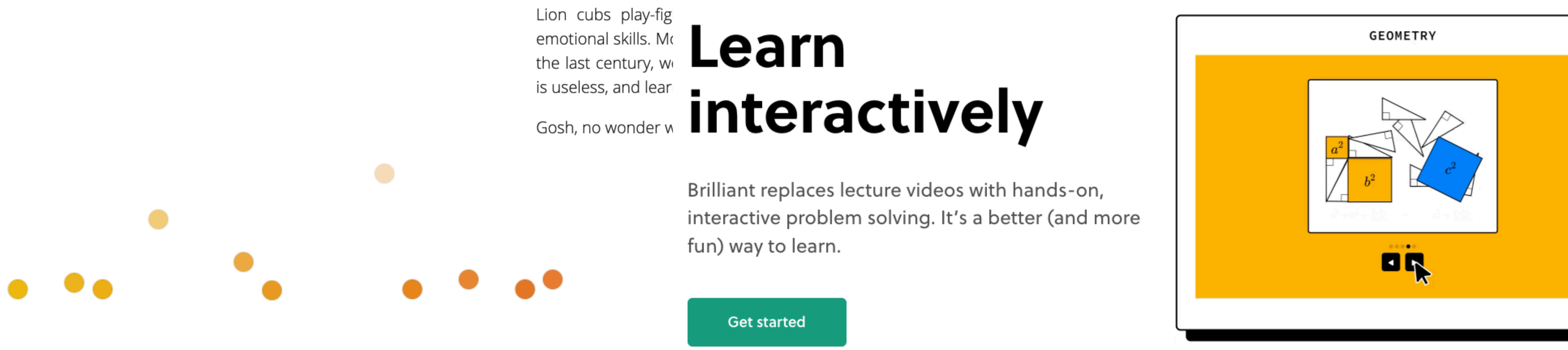


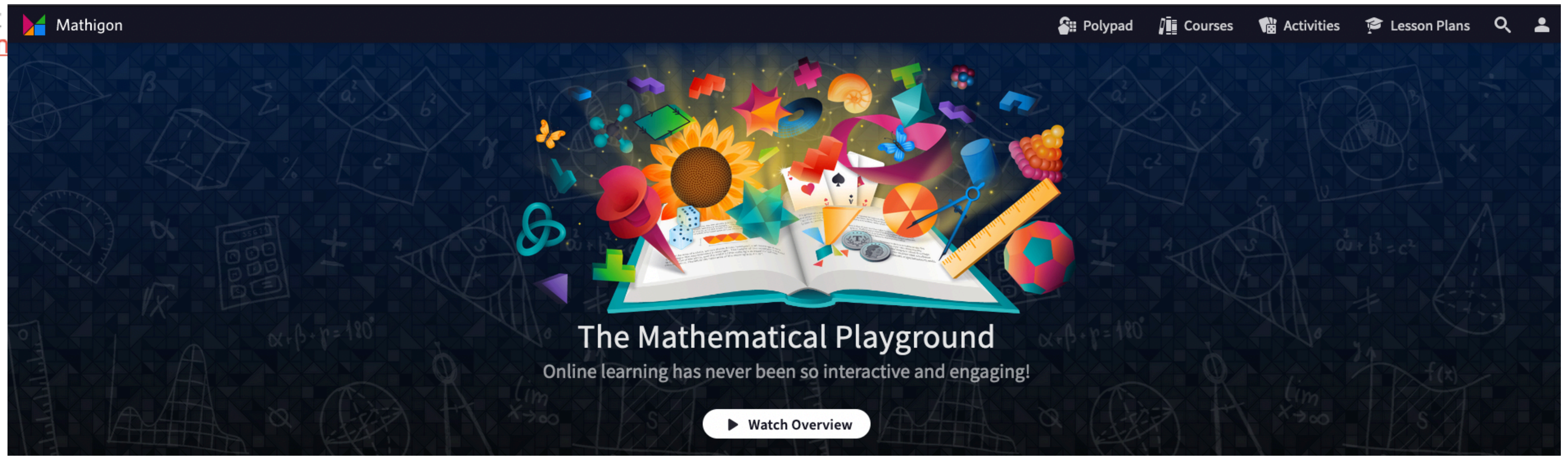
From encoding to semantic-preserving interactivity

Proposed

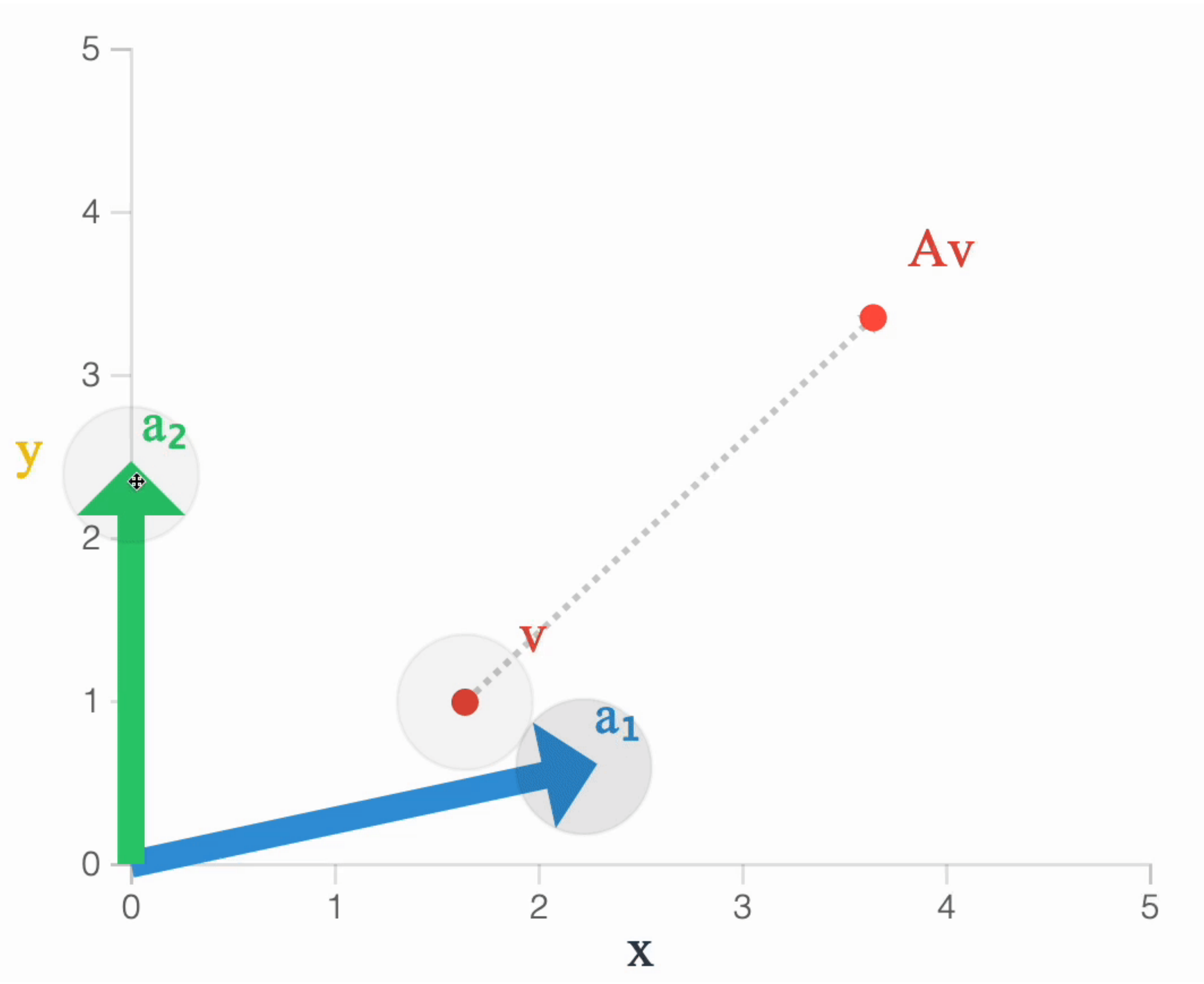
Explorable explanations & interactive problems







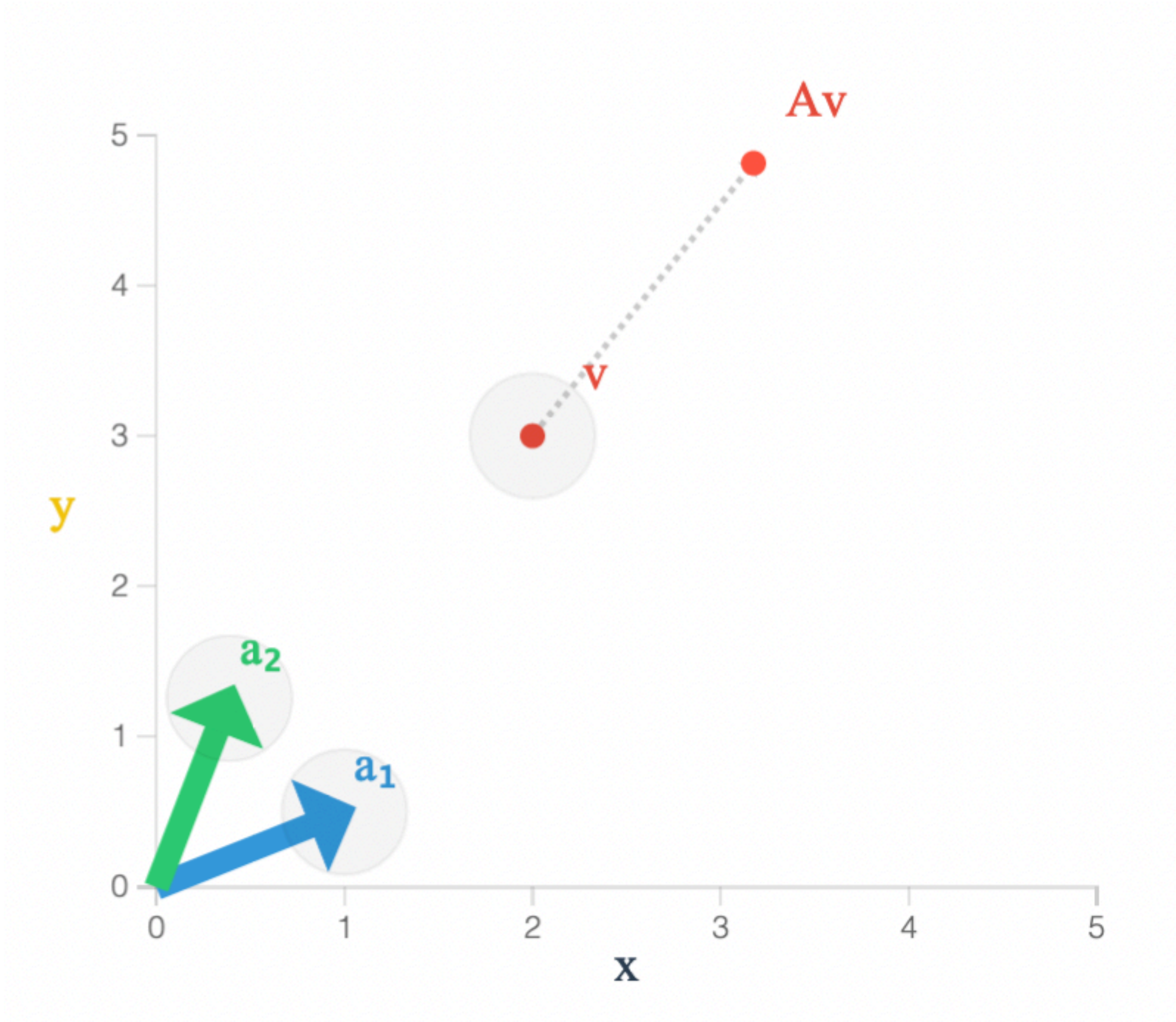
Well, it’s a lot of work




```javascript
// source: https://setosa.io/ev/eigenvectors-and-eigenvalues/script.js
myApp.directive('simplePlot', function() {
  function link(scope, el, attr) {
    el = d3.select(el[0])
    var opt = scope.opt
    var svg = el.append('svg')
    ;[el, svg].map(function(e) { e.attr({width: opt.w, height: opt.h}) })
    var defs = svg.append('defs').call(addMarkers)

    if (opt.ticks === undefined) opt.ticks = 5

    // Axis
    svg.append('g').attr('class', 'axis')
      .selectAll('g.axis').data([
        { axis: d3.svg.axis().scale(opt.xScale).orient('bottom').ticks(opt
          pos: [0, opt.h - opt.m.b] },
        { axis: d3.svg.axis().scale(opt.yScale).orient('left').ticks(opt.t
          pos: [opt.w / 2 - opt.pW / 2, 0] }
      ]).enter()
      .append('g').attr('class', 'axis')
      .each(function(d) { d3.select(this).call(d.axis) })
      .attr('transform', function(d) { return 'translate(' + d.pos + ')' }
      .call(styleAxis)

    // Vectors
    var vectors = svg.append('g').attr('class', 'vectors')
      .selectAll('line')
      .data(opt.vectorData || []).enter()
        .append('line').each(function(d) { d.style(d3.select(this), scope)
        .attr('marker-end', function(d) {
          return d.head && 'url(#vector-head-' + d.head + ')'
        })

    // Points
    var points = svg.append('g').attr('class', 'points')
      .selectAll('g').data(opt.pointData || []).enter().append('g')
    points.append('circle').attr('r', 4).style('fill', function(d, i) {
      return d3.rgb(color.tertiary).brighter(i * 0.3)
      // return d3.rgb(color.tertiary).darker(i * 0.3)
    })

    // Labels
    var labels = svg.append('g').attr('class', 'labels')
      .data(opt.labelData || []).enter().append('text')
        .attr('transform', function(d) {
          return 'translate(' +
            ((typeof d.pos === 'function') ? d.pos(scope) : d.pos )
          + ')'
        })
        .text(function(d) {
          return (typeof d.label !== 'function') && d.label || ''
        })
        .call(styleAxisLabels)

    // Nobs
    var nobs = buildNobs(opt.nobData, scope, svg)

    nobs.call(d3.behavior.drag()
      .on('drag', function(d) {
        scope.$apply(function() {
          d.set(scope, d3.mouse(svg.node()))
        }.bind(this))
      }))

    scope.$watch('opt', redraw, true)
    function redraw() {
      nobs.each(function(d) {
        d3.select(this).attr('transform', 'translate(' + d.get(scope) + ')')
      })

      points
        .filter(function(d) { return typeof d.pos === 'function' })
        .attr('transform', function(d) {
          return 'translate(' + d.pos(scope) + ')'
        })

      vectors.call(updateVector, scope)

      labels.filter(function(d) { return typeof d.pos === 'function' })
        .attr('transform', function(d) {
          return 'translate(' + d.pos(scope) + ')'
        })
      labels.filter(function(d) { return (typeof d.label) === 'function' })
        .text(function(d) { return d.label(scope) })
    }
  }

  function styleAxisLabels(g) {
    g.style('text-anchor', 'middle')
     .each(function(d) { d.style && d.style(d3.select(this)) })
  }
```

https://github.com/vicapow/explained-visually

...not anymore with Penrose

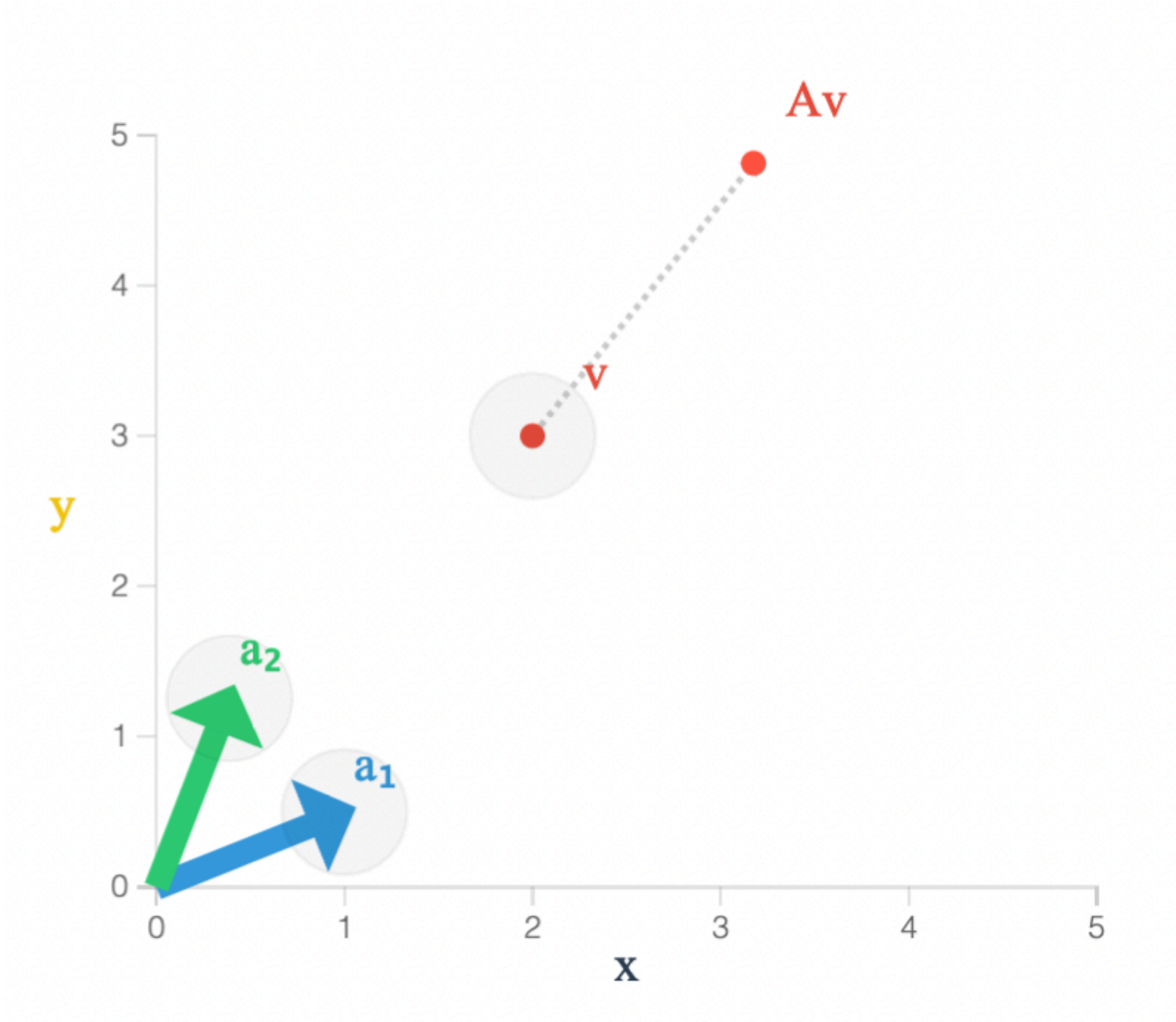


Domain

```
type Vector
type Matrix
Vector <: Matrix
function columns: Vector[] -> Matrix
function multiply: Matrix -> Matrix
```

https://github.com/vicapow/explained-visually

...not anymore with Penrose



Domain

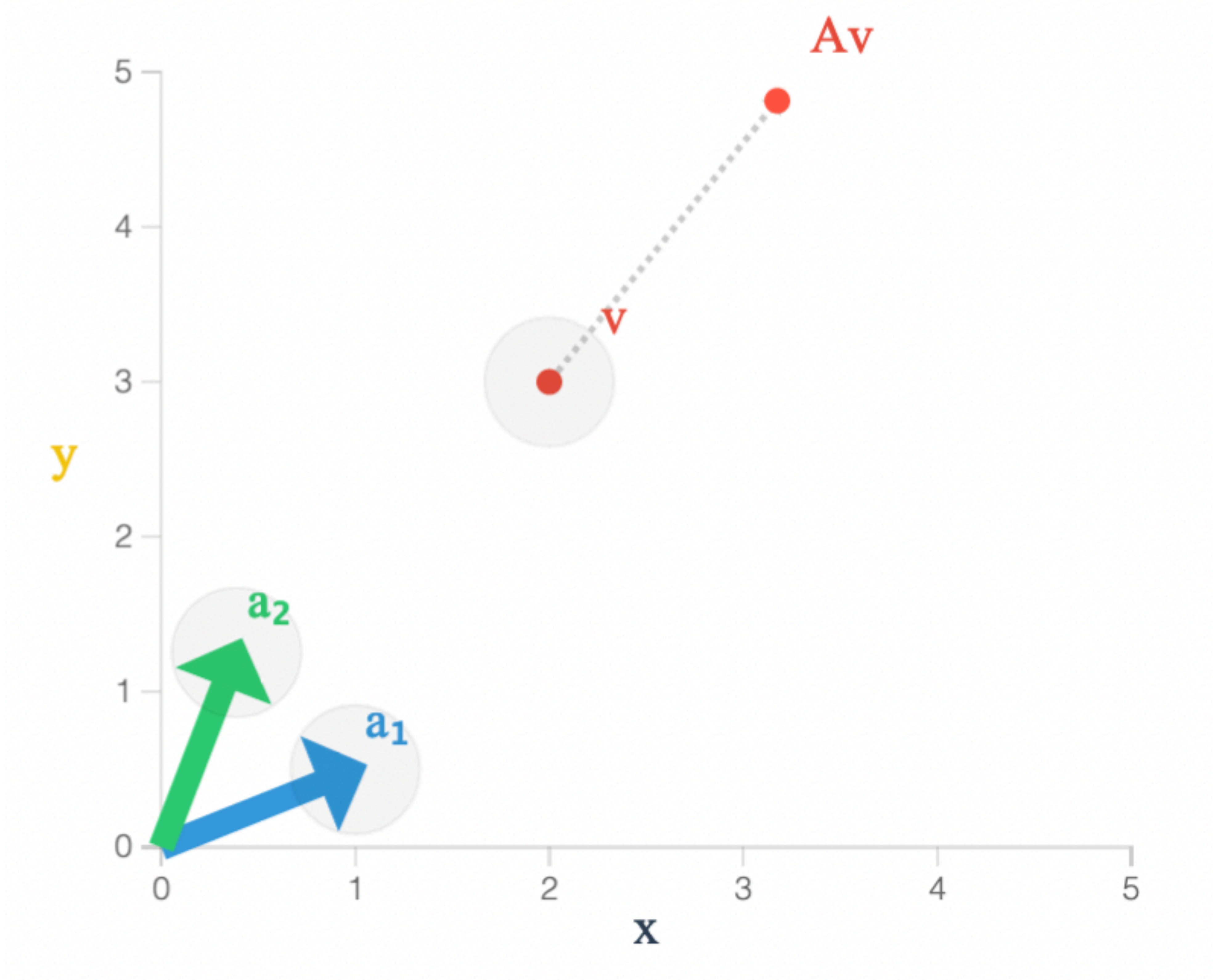
```
type Vector
type Matrix
Vector <: Matrix
function columns: Vector[] -> Matrix
function multiply: Matrix -> Matrix
```

Substance

```
Vector a_1, a_2, v
Matrix A := columns(a_1, a_2)
Vector Av = multiply(A, v)
AutoLabel All
```

https://github.com/vicapow/explained-visually

...not anymore with Penrose



Domain

```
type Vector
type Matrix
Vector <: Matrix
function columns: Vector[] -> Matrix
function multiply: Matrix -> Matrix
```

Substance

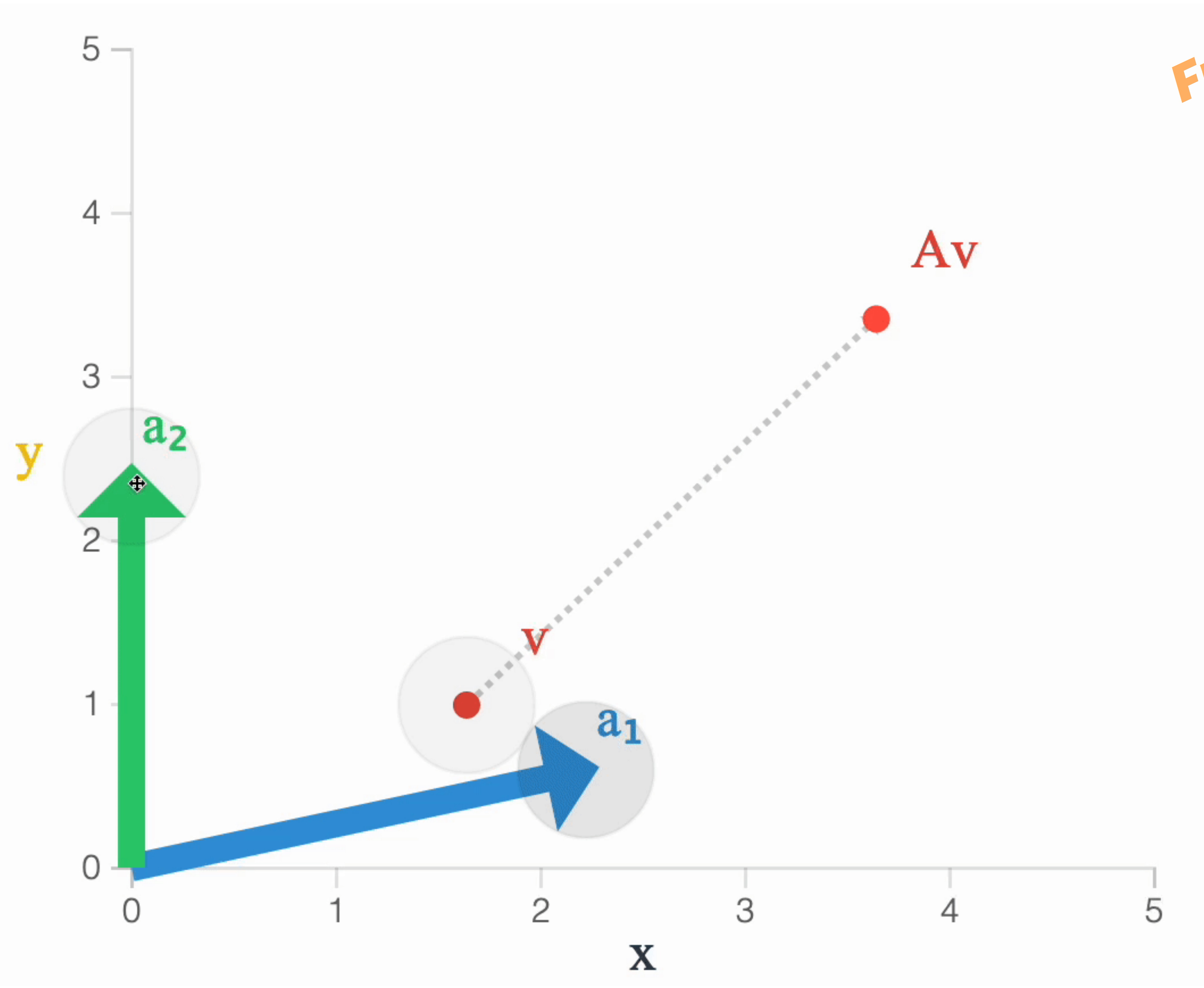
```
Vector a_1, a_2, v
Matrix A := columns(a_1, a_2)
Vector Av = multiply(A, v)
AutoLabel All
```

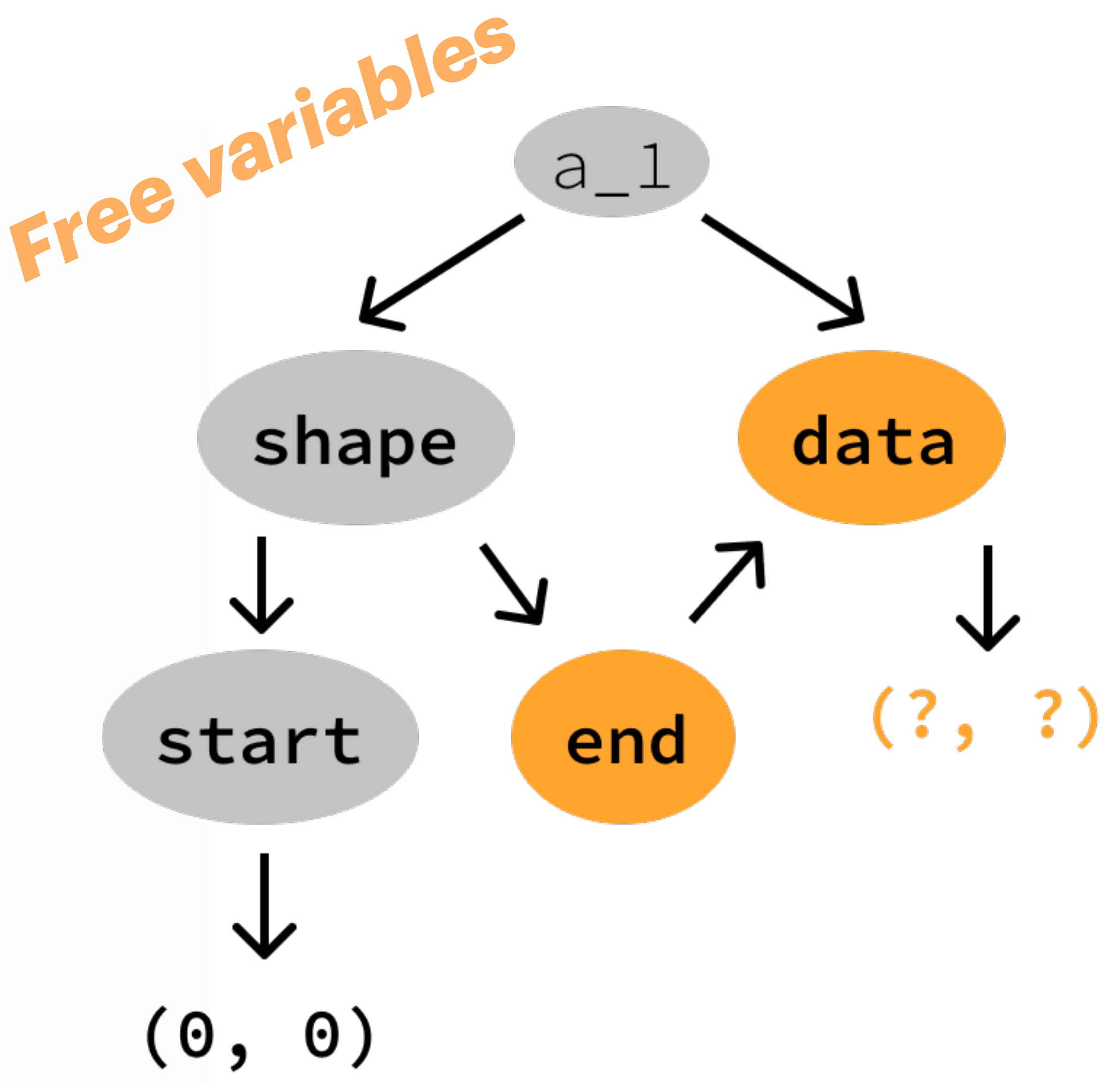
Style

```
Vector v {
  v.data  = (?, ?)
  v.icon = Arrow {
    start : (0, 0)
    end   : v.data
  }
  v.text = Text { string : v.label }
  ensure near(v.text, v.icon)
}
Matrix A
where A := columns(a_1, a_2) {
  A.data = [a_1.data; a_2.data]
}
Vector Av = multiply(A, v) {
  override Av.data = matmul(A.data, v.data)
}
```

https://github.com/vicapow/explained-visually

Interactivity comes for free with free variables





Substance

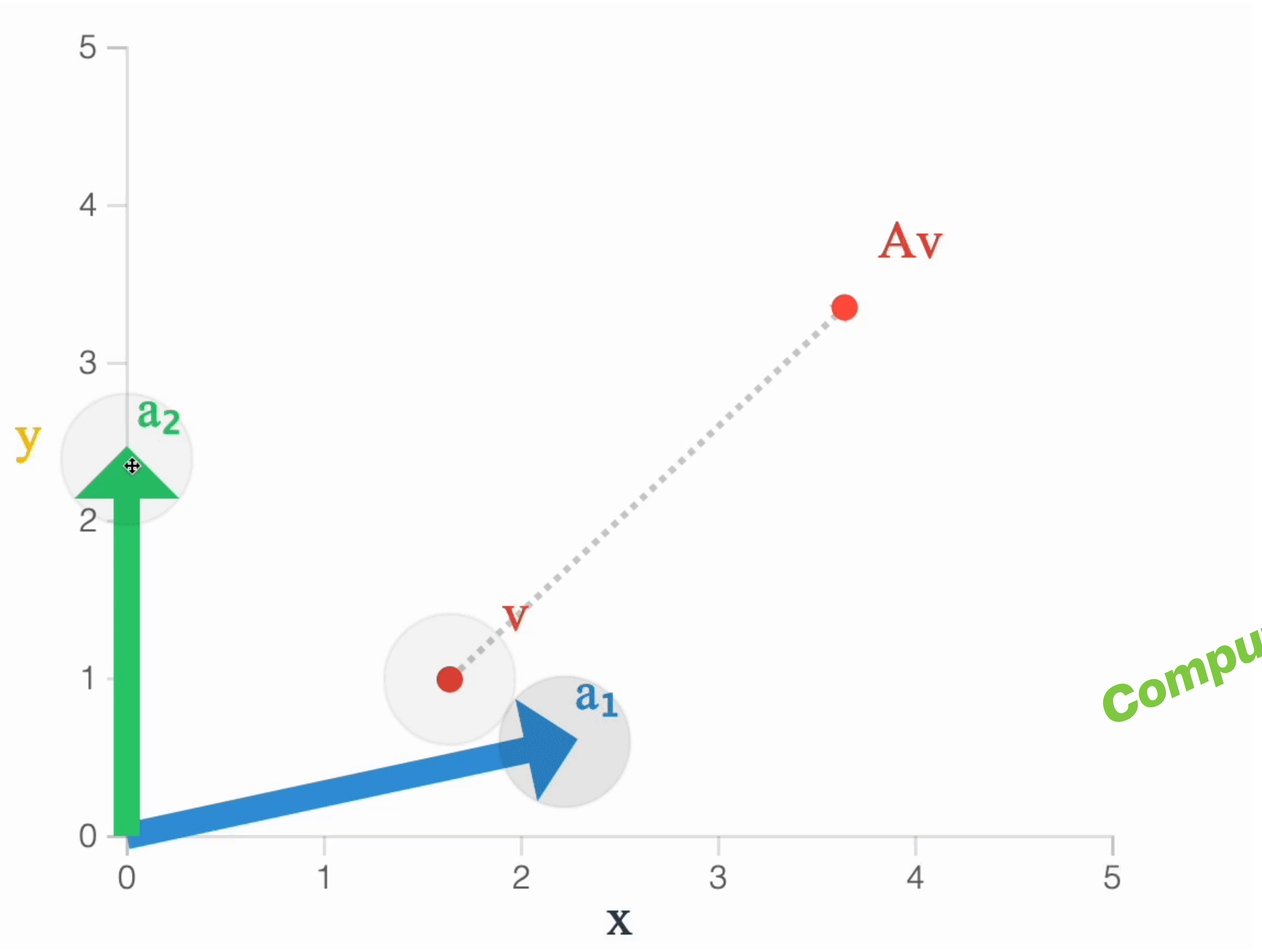
```
Vector a_1, a_2, v
Matrix A := columns(a_1, a_2)
Vector Av = multiply(A, v)
AutoLabel All
```

Style

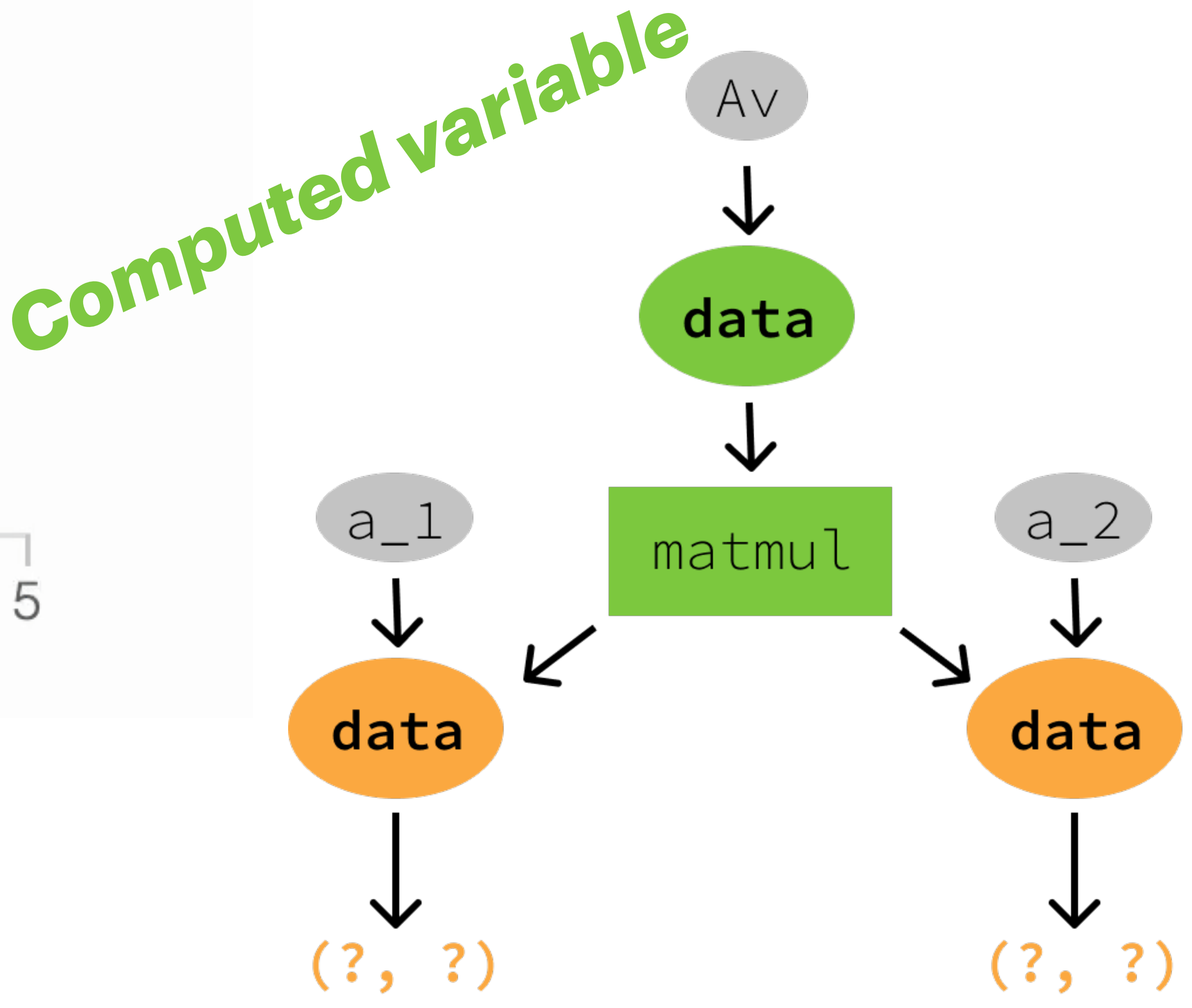
```
Vector v {
  v.data  = (?, ?)
  v.icon = Arrow {
    start : (0, 0)
    end   : v.data
  }
  v.text = Text { string : v.label }
  ensure near(v.text, v.icon)
}
Matrix A
where A := columns(a_1, a_2) {
  A.data = [a_1.data; a_2.data]
}
Vector Av = multiply(A, v) {
  override Av.data = matmul(A.data, v.data)
}
```

https://github.com/vicapow/explained-visually

Computed values aren't free



https://github.com/vicapow/explained-visually



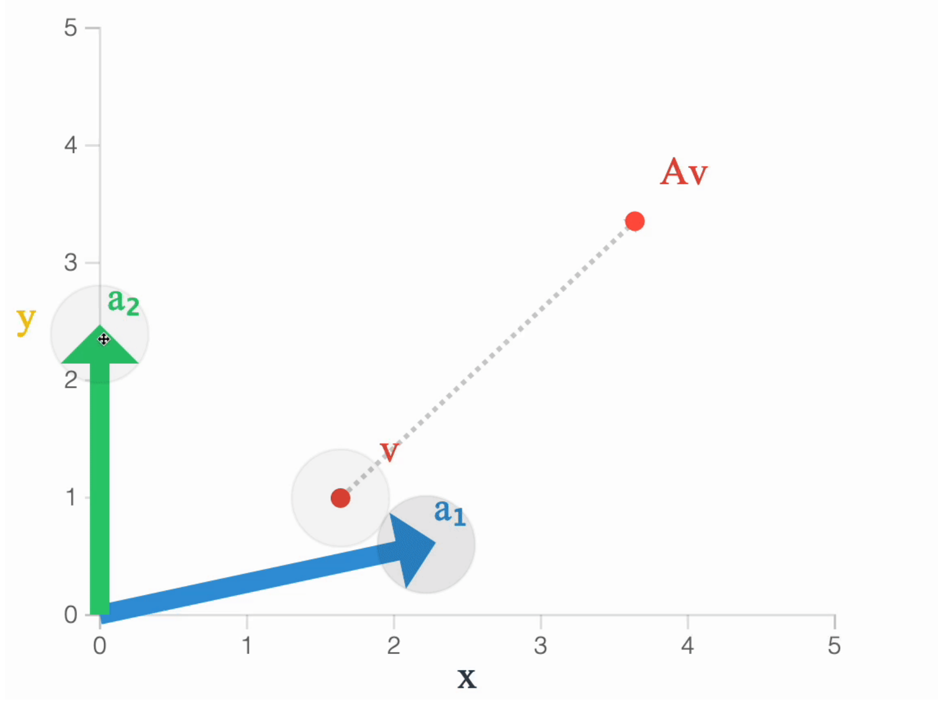
Style

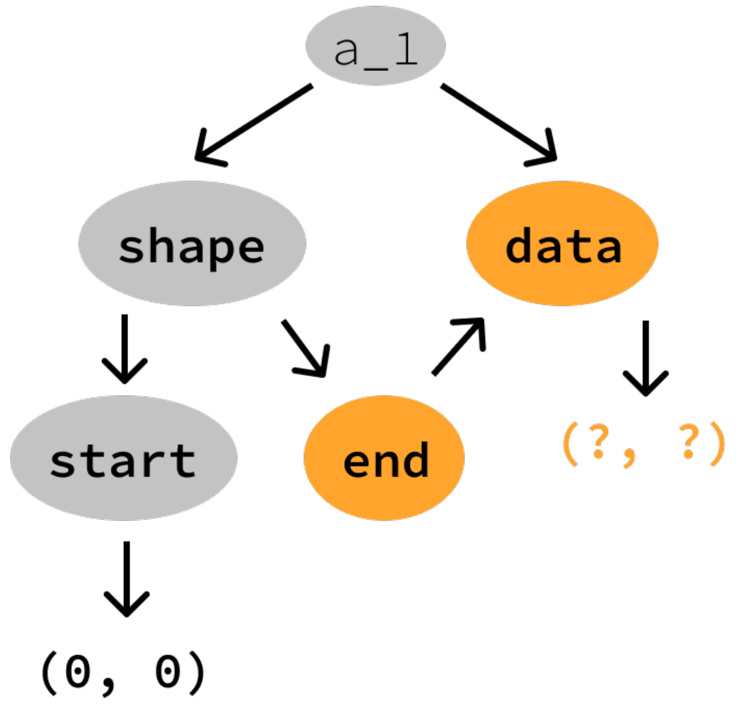
```
Vector v {
  v.data  = (?, ?)
  v.icon = Arrow {
    start : (0, 0)
    end   : v.data
  }
  v.text = Text { string : v.label }
  ensure near(v.text, v.icon)
}
Matrix A
where A := columns(a_1, a_2) {
  A.data = [a_1.data; a_2.data]
}
Vector Av = multiply(A, v) {
  override Av.data = matmul(A.data, v.data)
}
```

Key idea: from semantics to interactive feedback

Translational semantics: how notations are interpreted to diagrams

Visual semantics: how graphical primitives relate to each others in a diagram

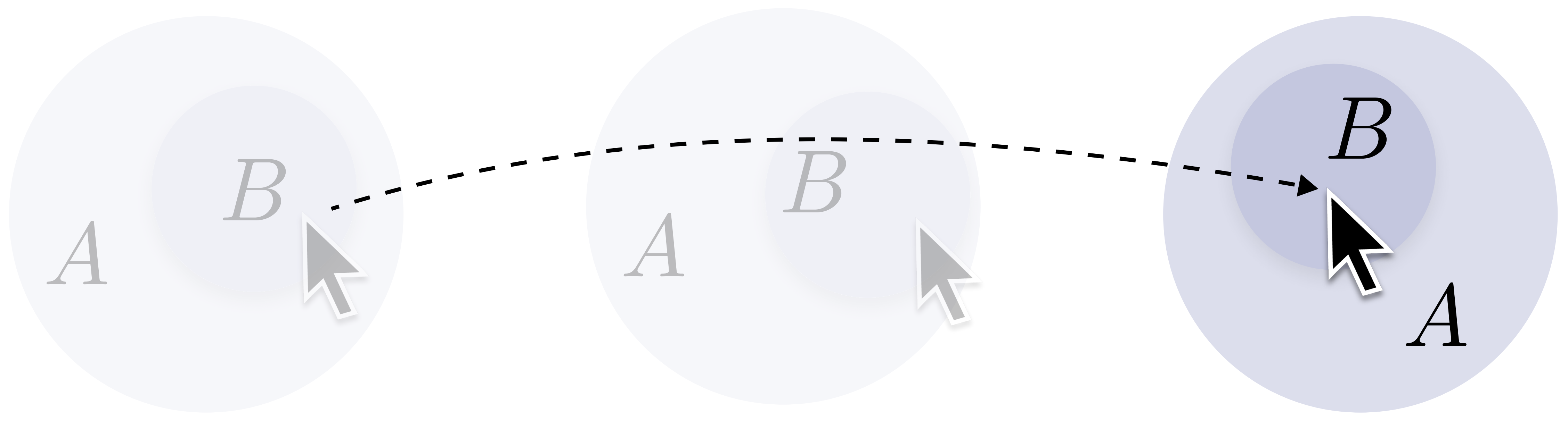




Proposal: simplify the programming of feedback-rich, interactive activities by leveraging translational and visual semantics

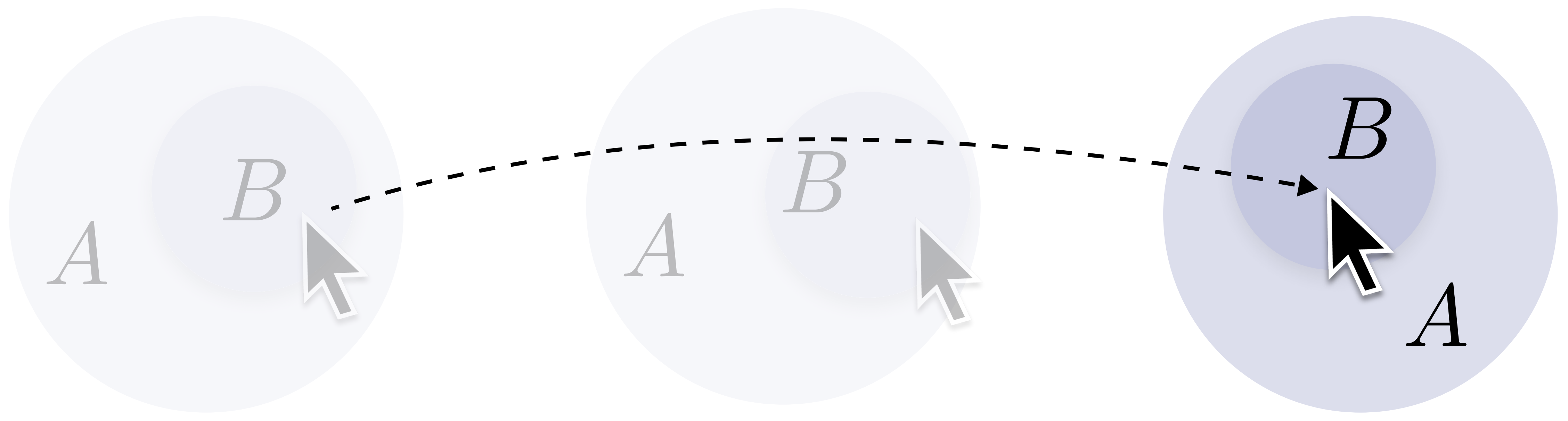
Two modes of drag feedback

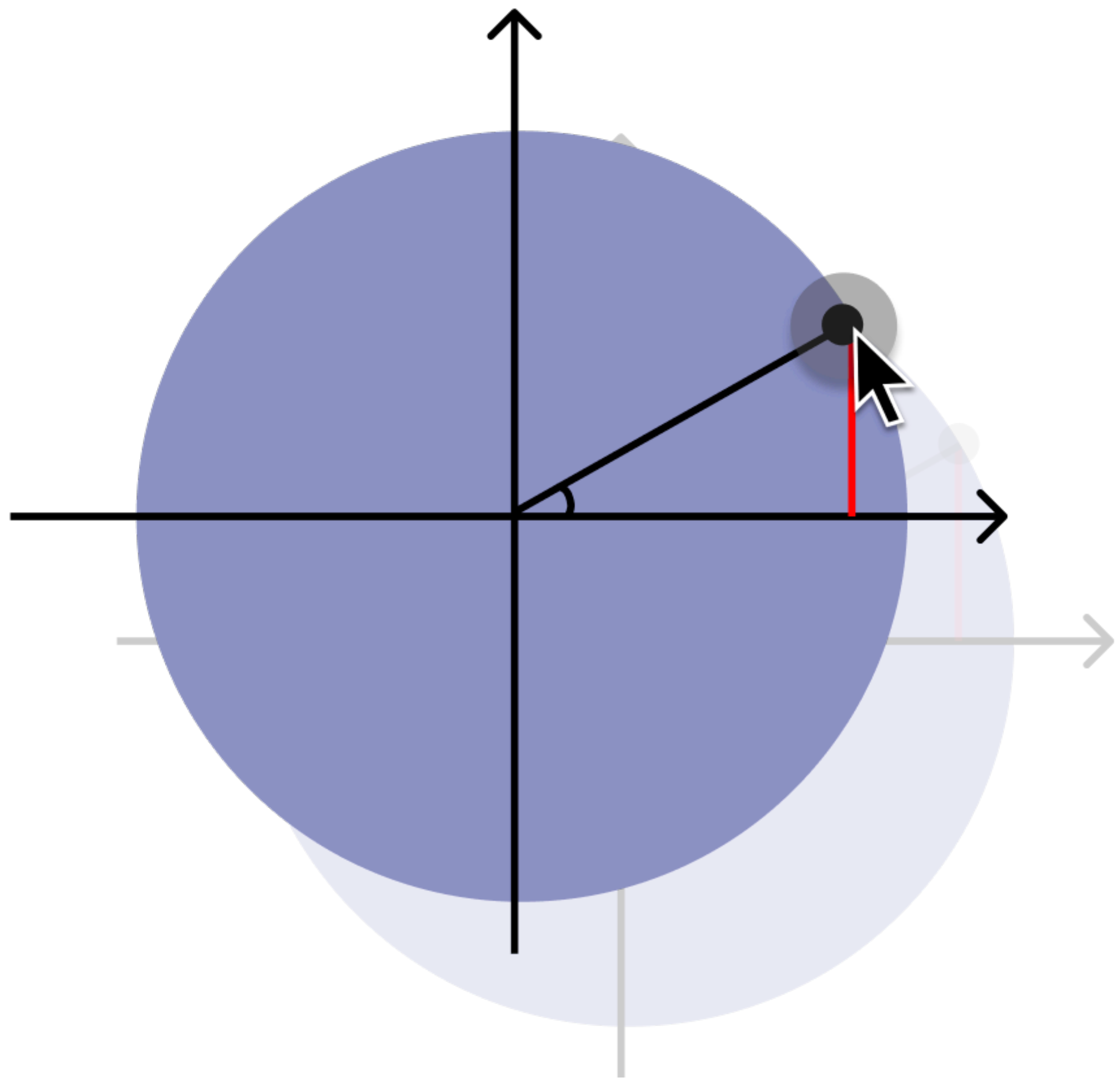
Follow the cursor



Two modes of drag feedback

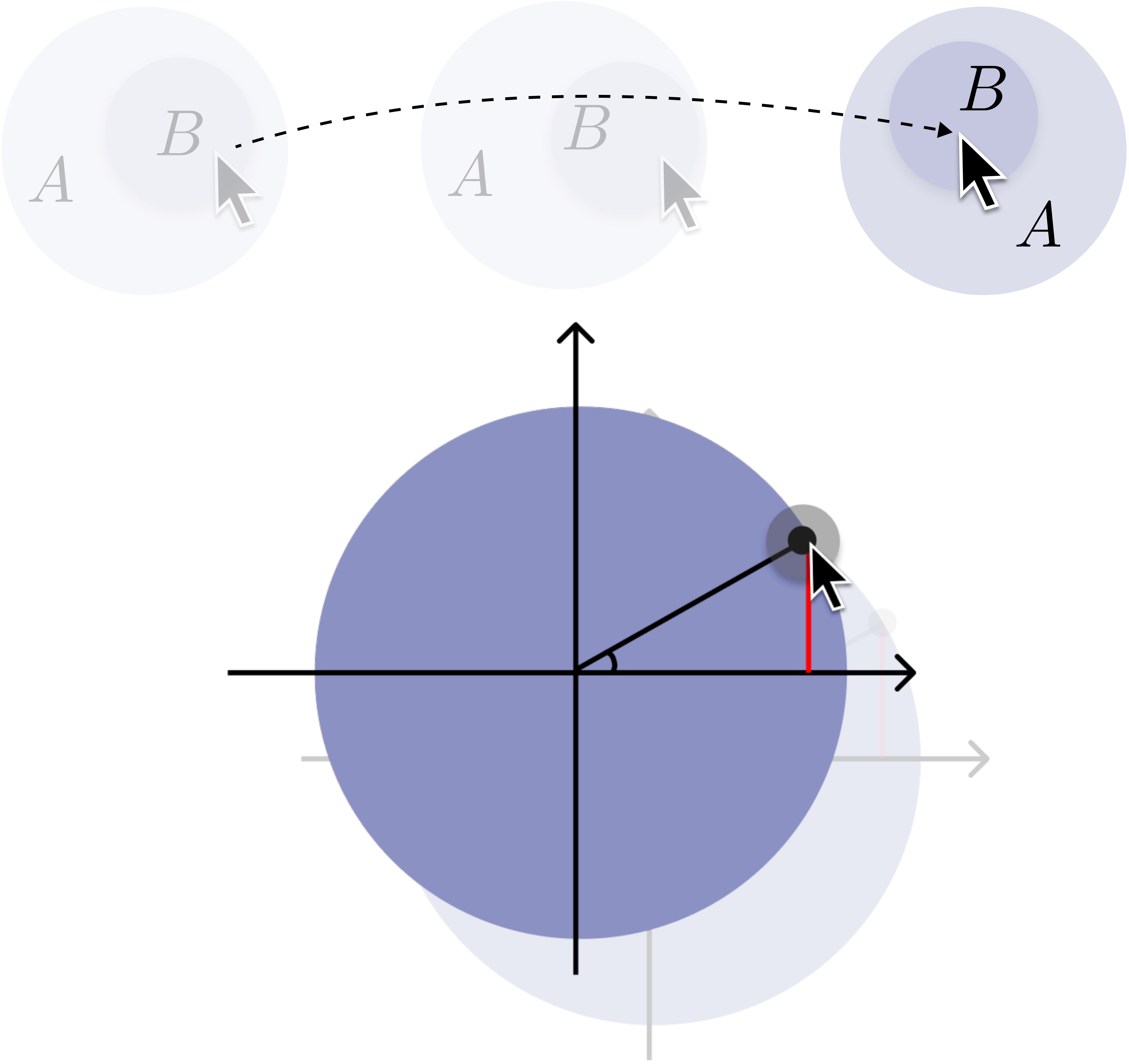
Follow the cursor



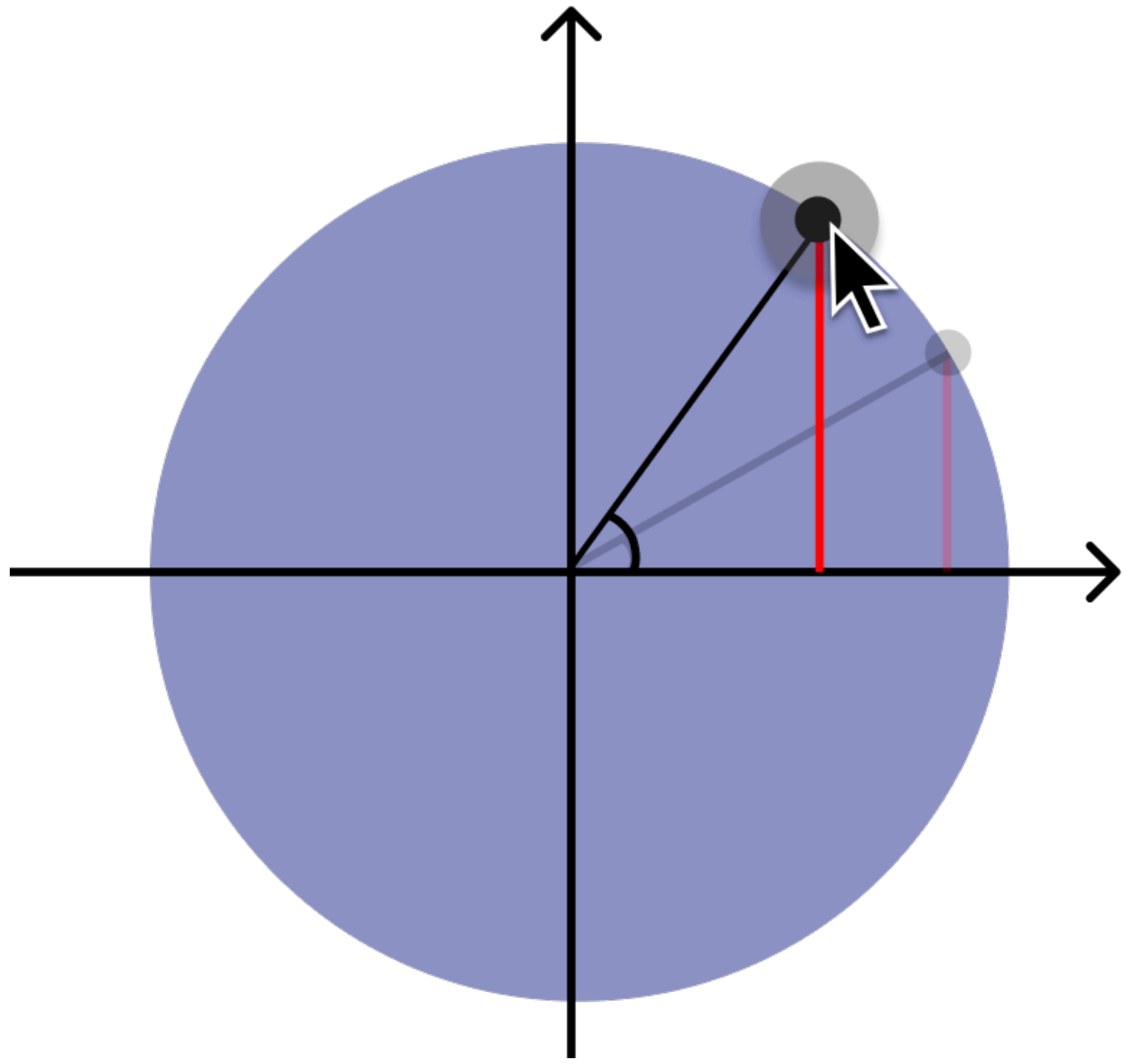


Two modes of drag feedback

Follow the cursor

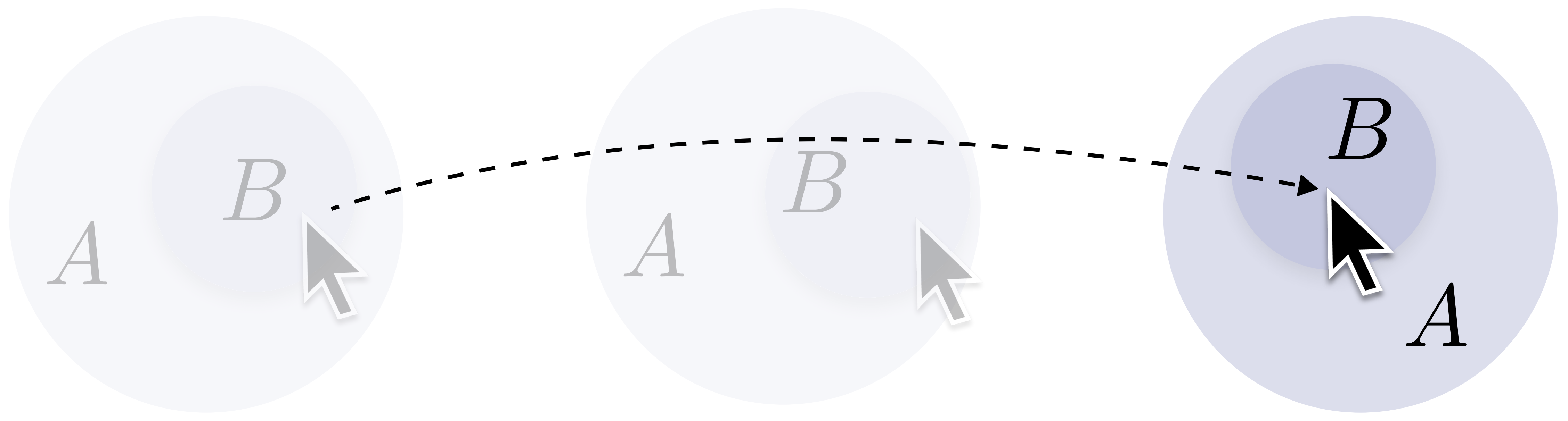


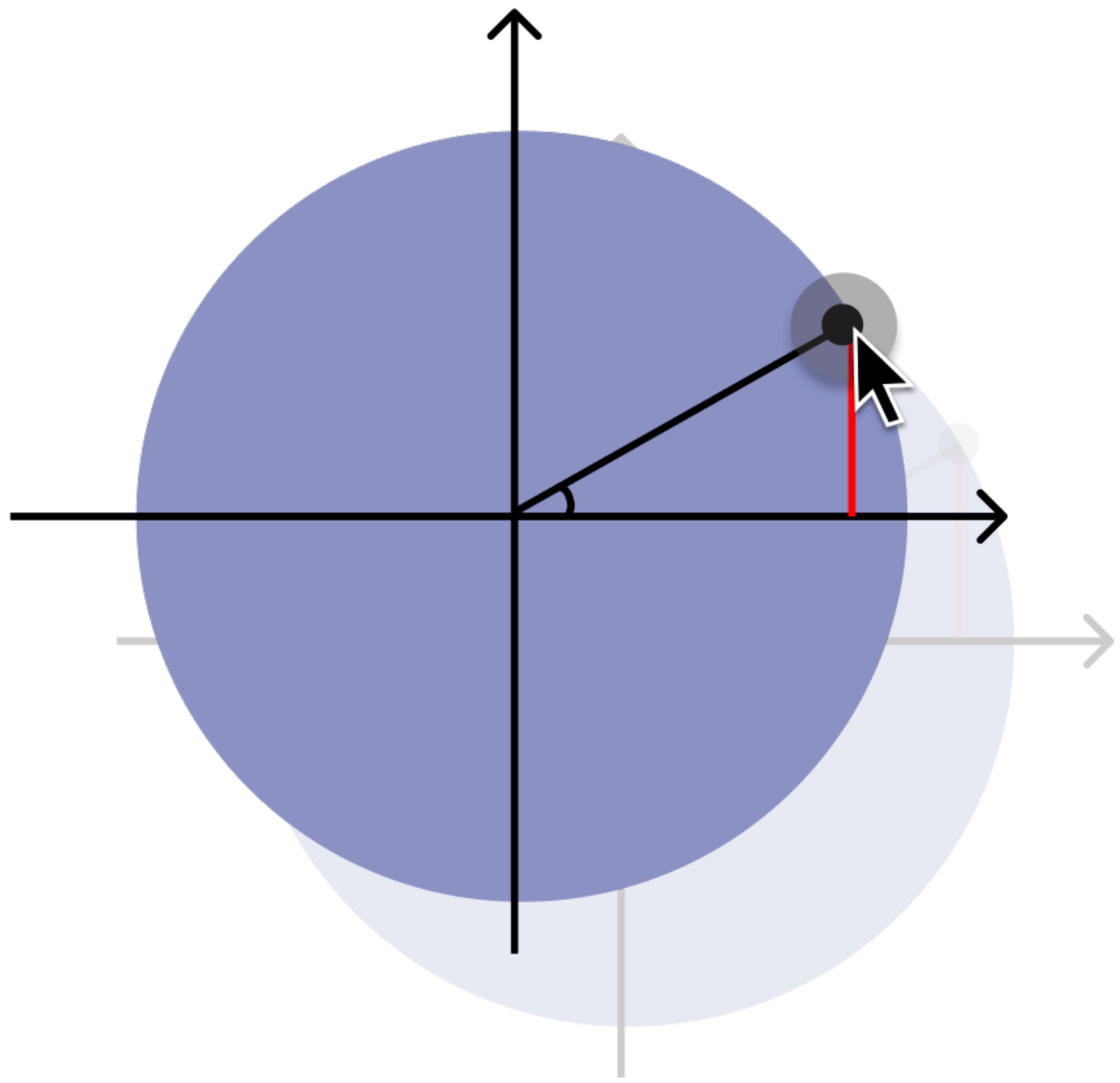
Freeze the world



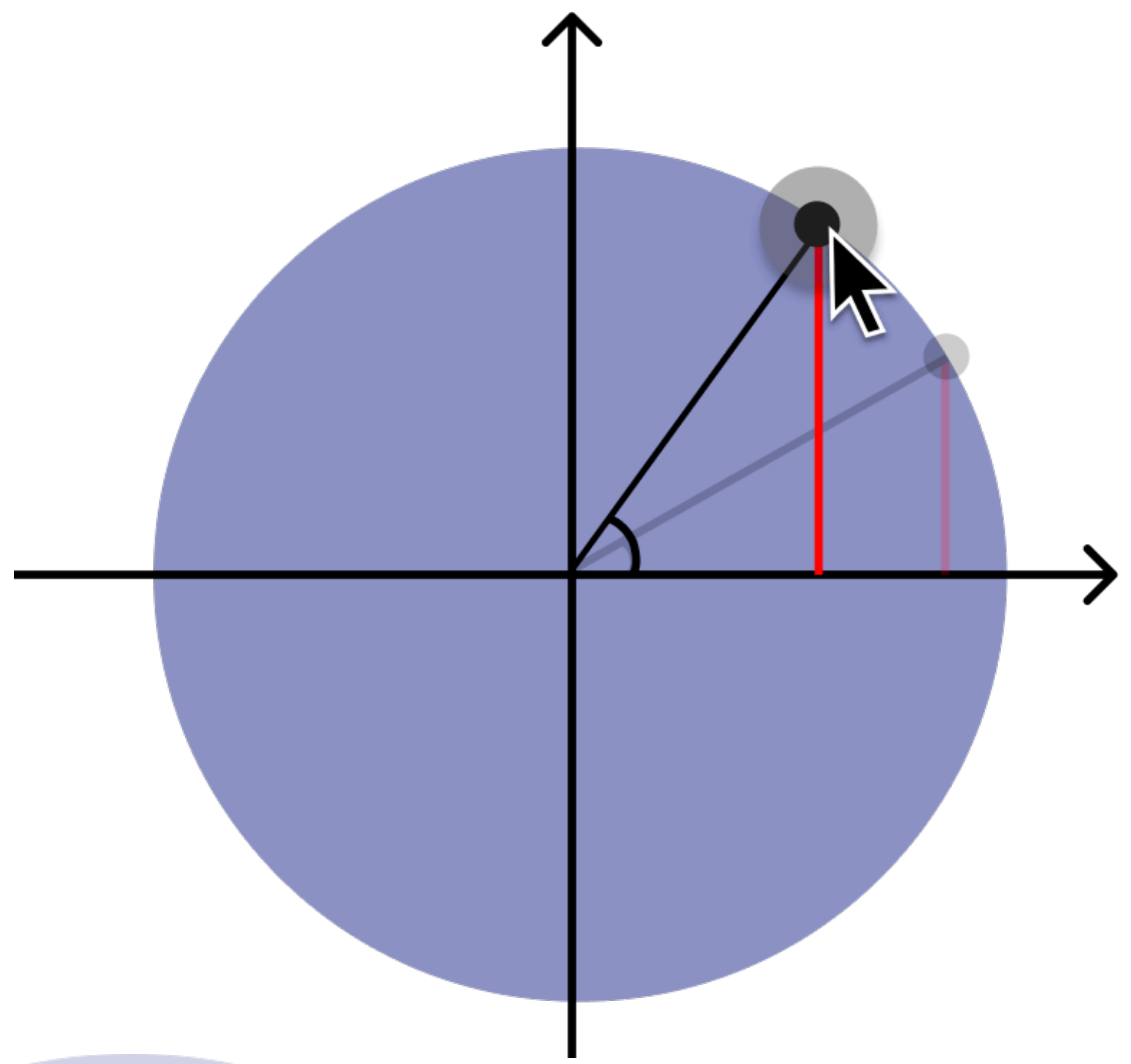
Two modes of drag feedback

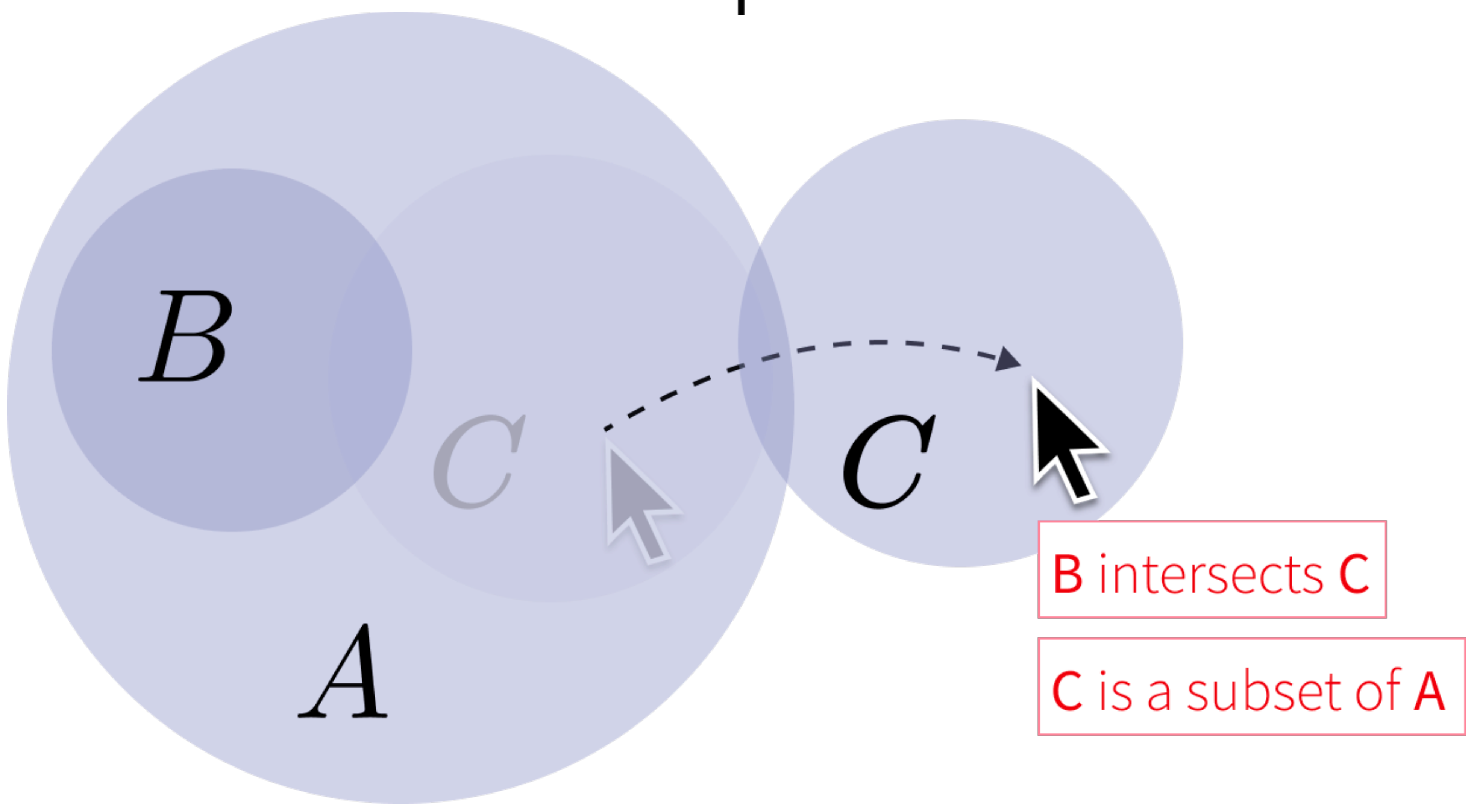
Follow the cursor



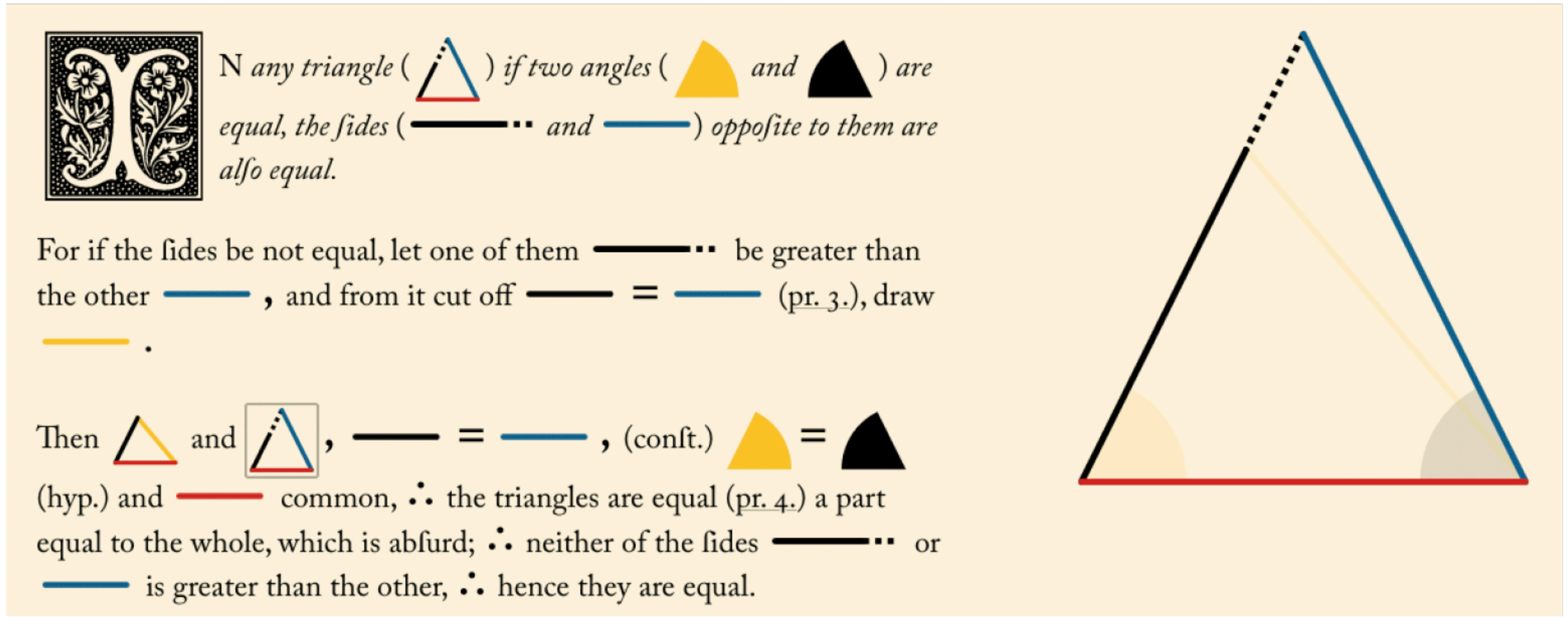


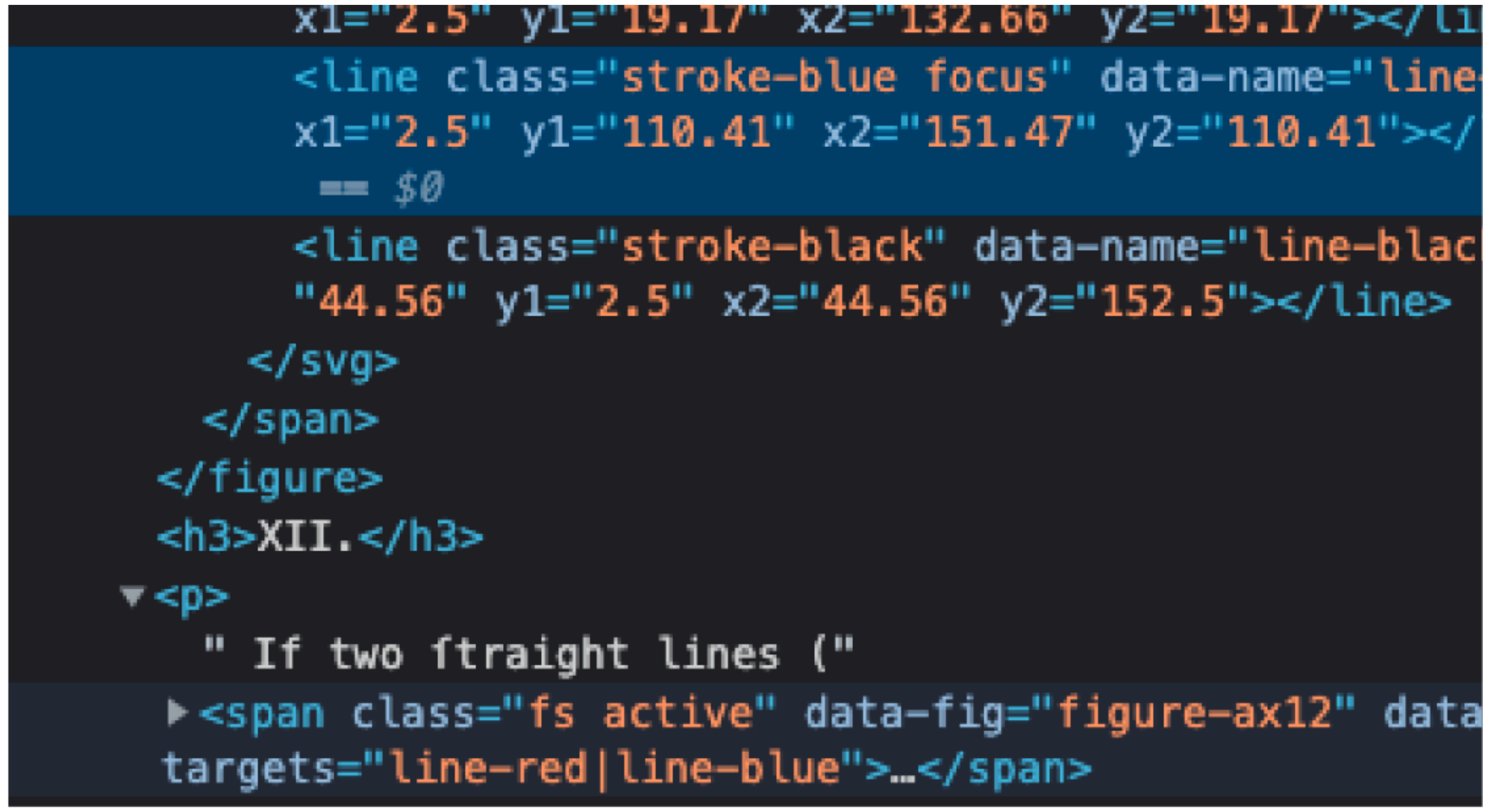
Freeze the world





Existing interactive documents use data labels to link between visuals and text. We can get this for free.

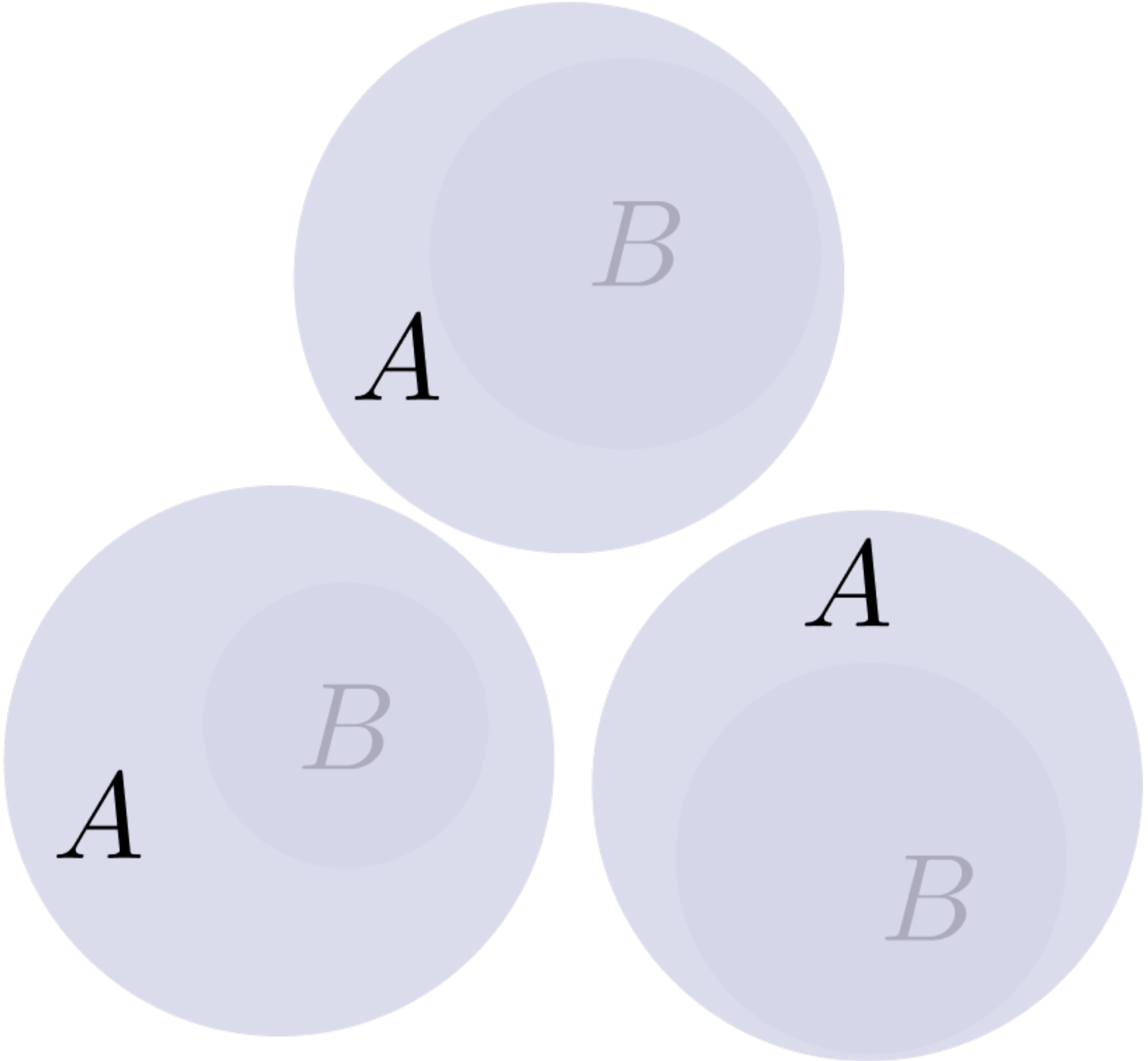





```
# Set intersection

Given `Set A` and `Set B`
, `IsSubset(B, A)`
indicates that `B` is a
subset of `A`
```

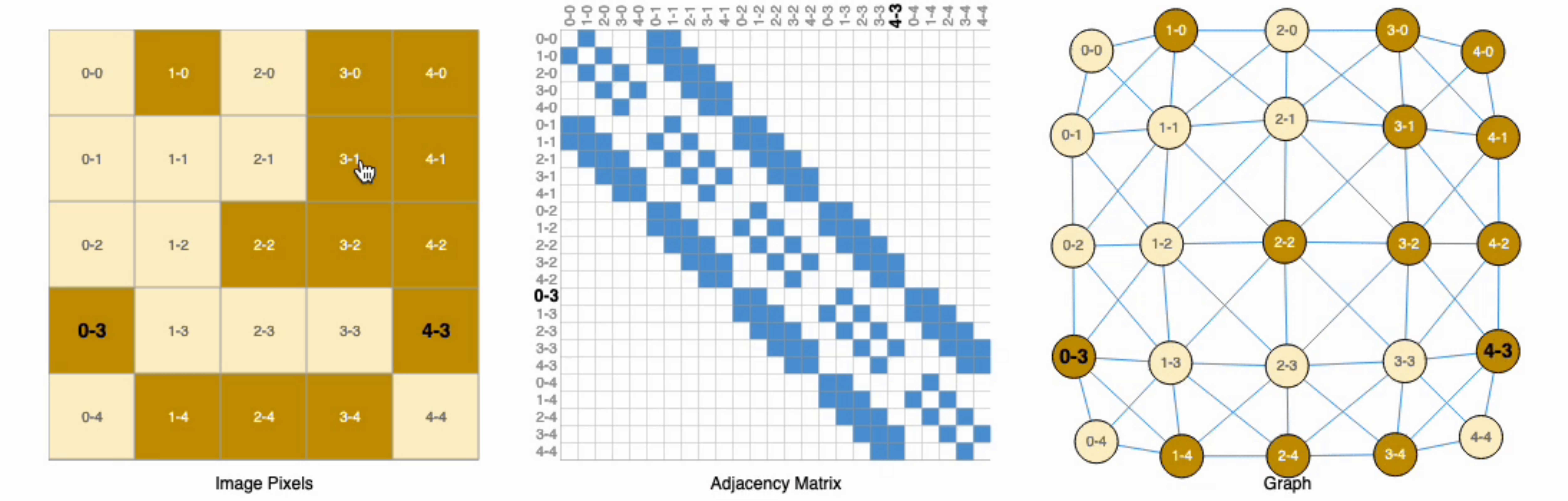
Set intersection

Given A and B, $B \subset A$ indicates that B is a subset of A.

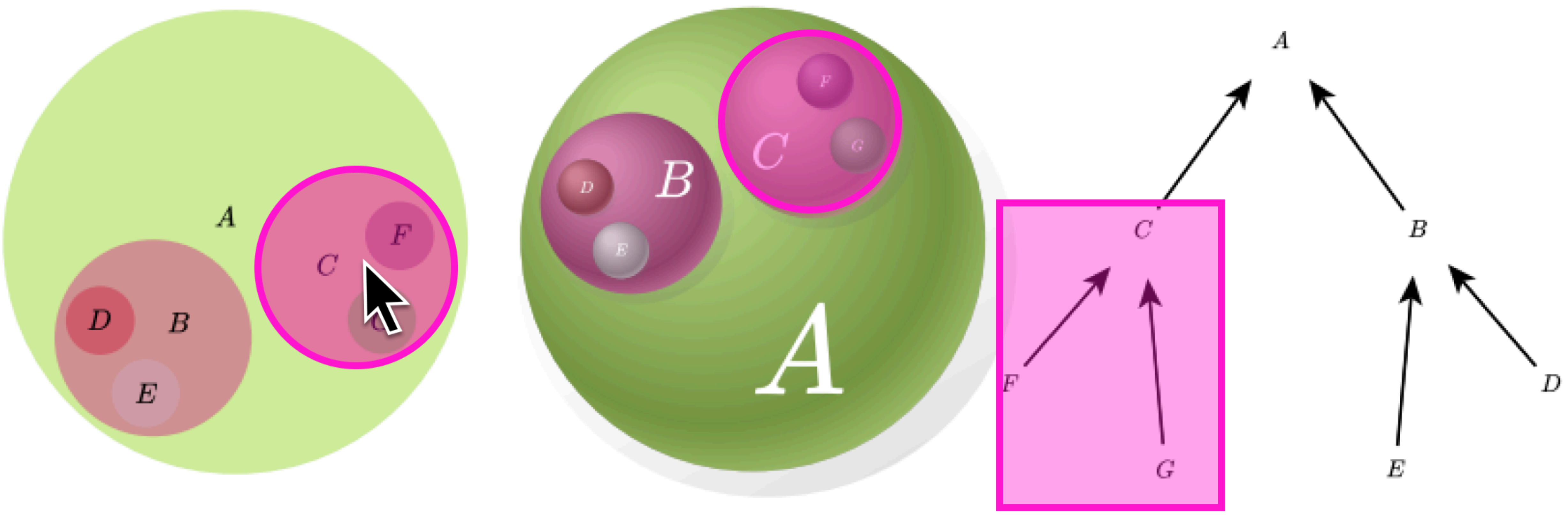


The Elements of Euclid With Coloured Diagrams and Symbols. https://www.c82.net/euclid/.

Multiple representations are great. Can we help people build connections amongst them through interaction?



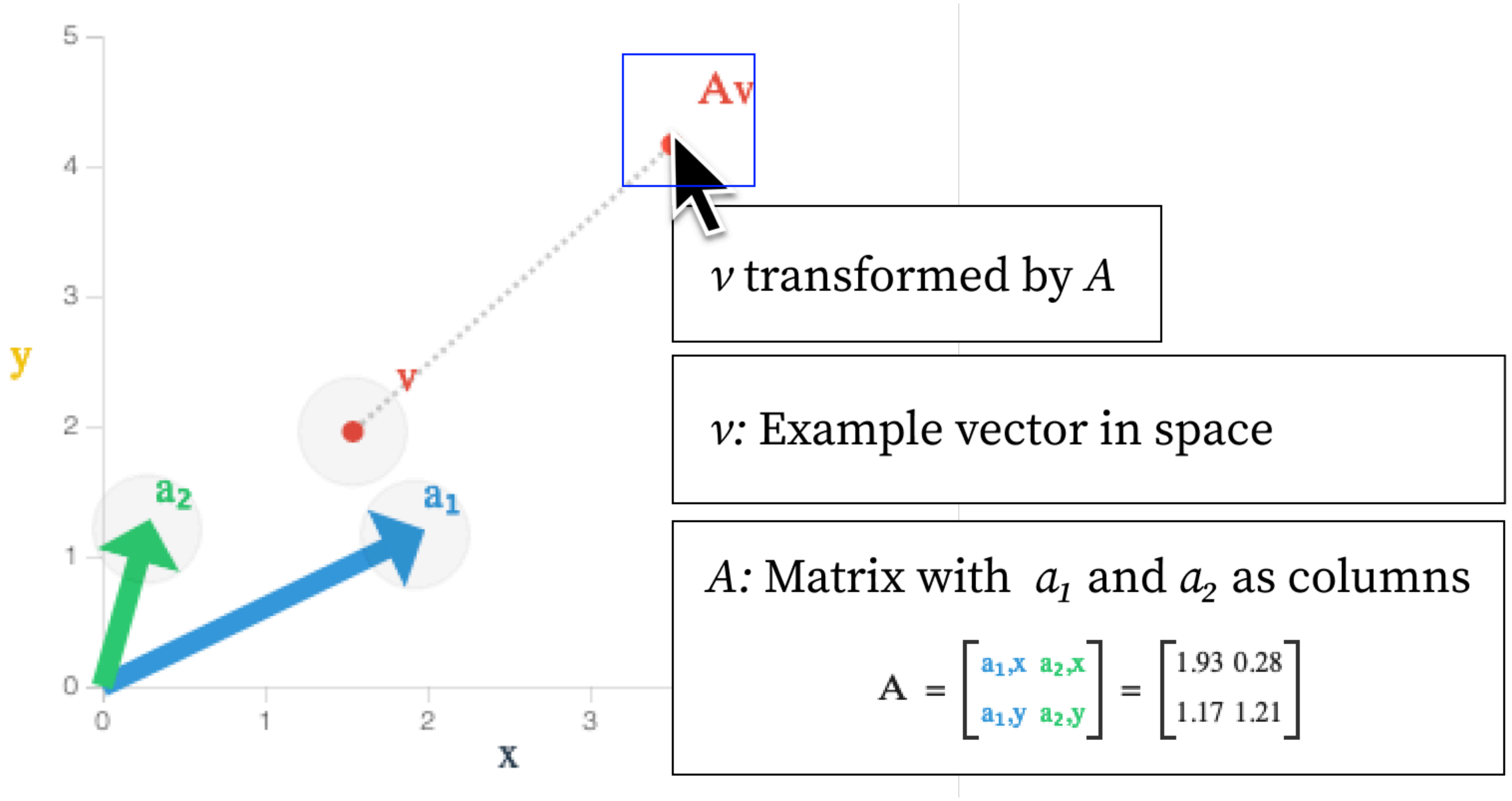
Click on an image pixel to toggle its value, and see how the graph representation changes.



A Gentle Introduction to Graph Neural Networks. https://distill.pub/2021/gnn-intro/

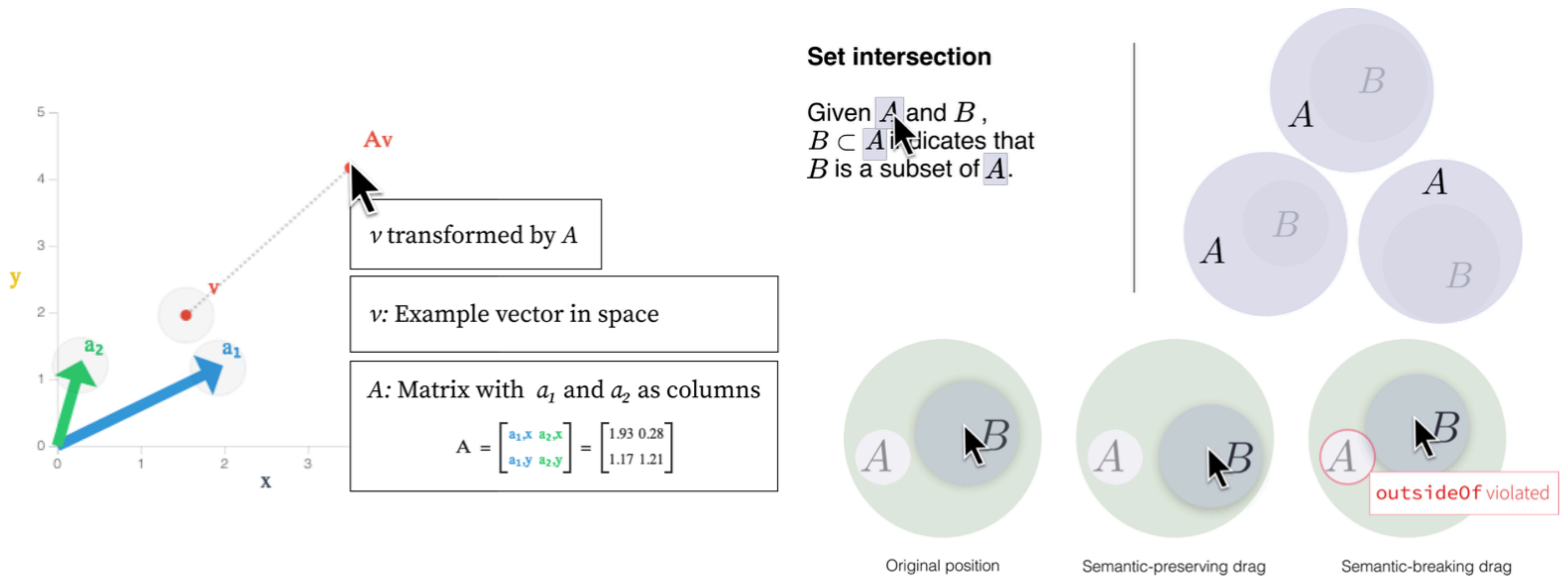
Docs are useful for reading notations. Tooltips help explain them on-demand. How about both?

```
-- First column of `A` [a_1.data]
Vector a_1
-- Second column of `A` [a_2.data]
Vector a_2
-- Example vector in space [v.data]
Vector v
-- Matrix with `a_1` and `a_2` as columns
   [A.data]
Matrix A := columns(a_1, a_2)
-- `v` transformed by `A`
Vector Av = multiply(A, v)
AutoLabel All
```

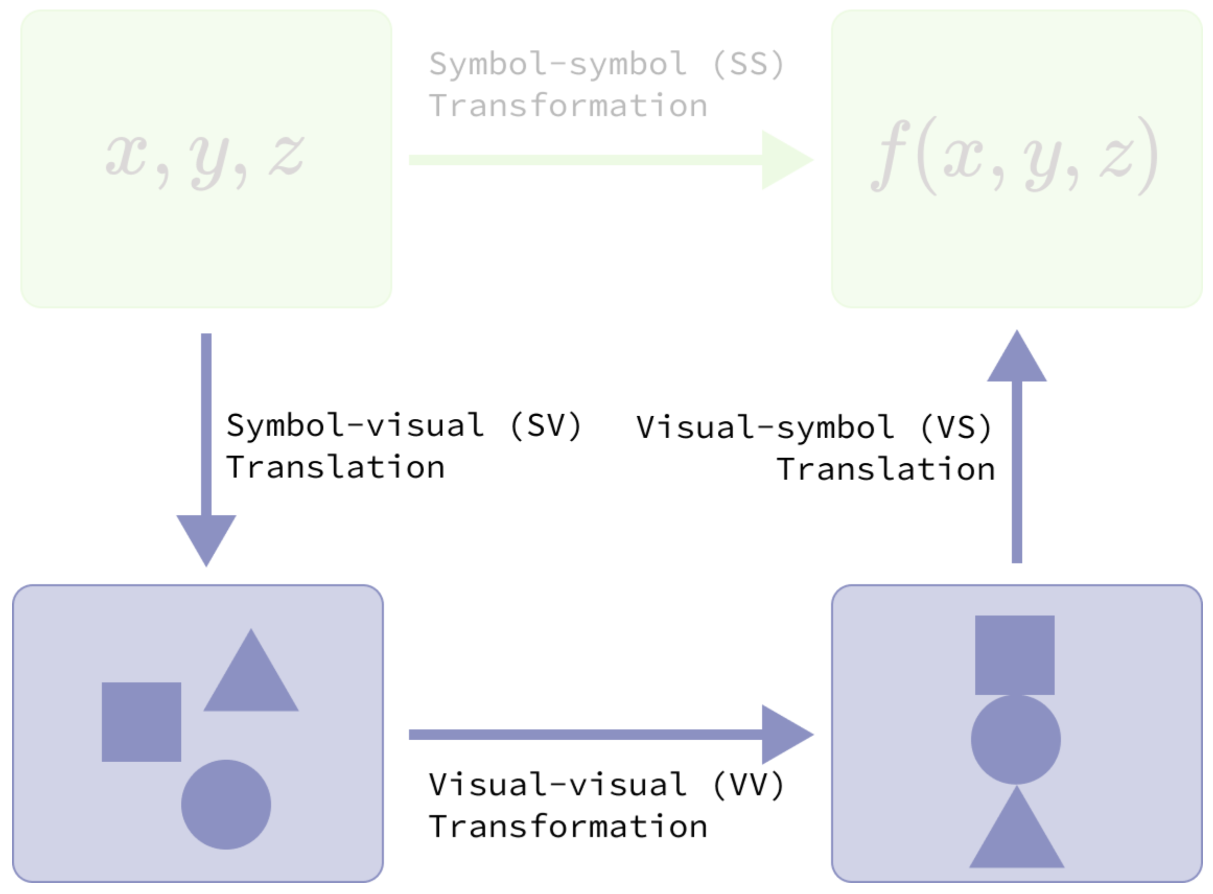


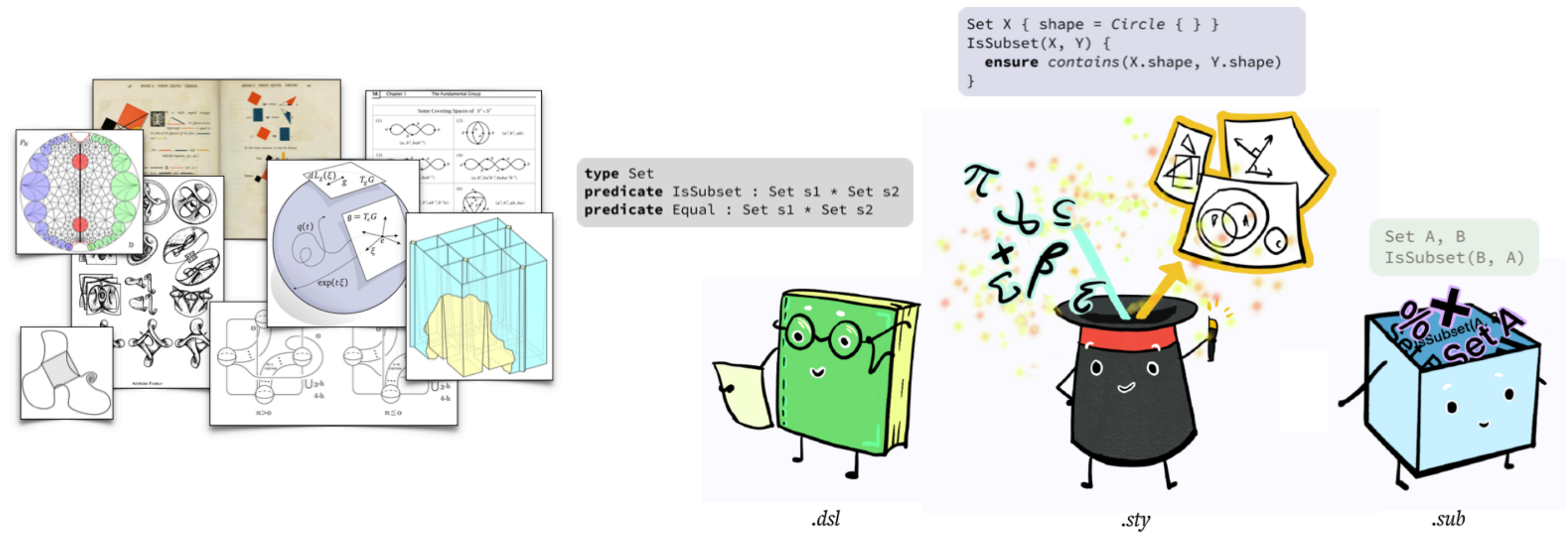
Conlen et al. Idyll: A Markup Language for Authoring and Publishing Interactive Articles on the Web.
Head et al. Augmenting Scientific Papers with Just-in-Time, Position-Sensitive Definitions of Terms and Symbols.
Crichton. A New Medium for Communicating Research on Programming Languages

Encoding visual representations in diagramming tools simplifies programming of **interactive** visual activities that provide students with **automated feedback** at scale.



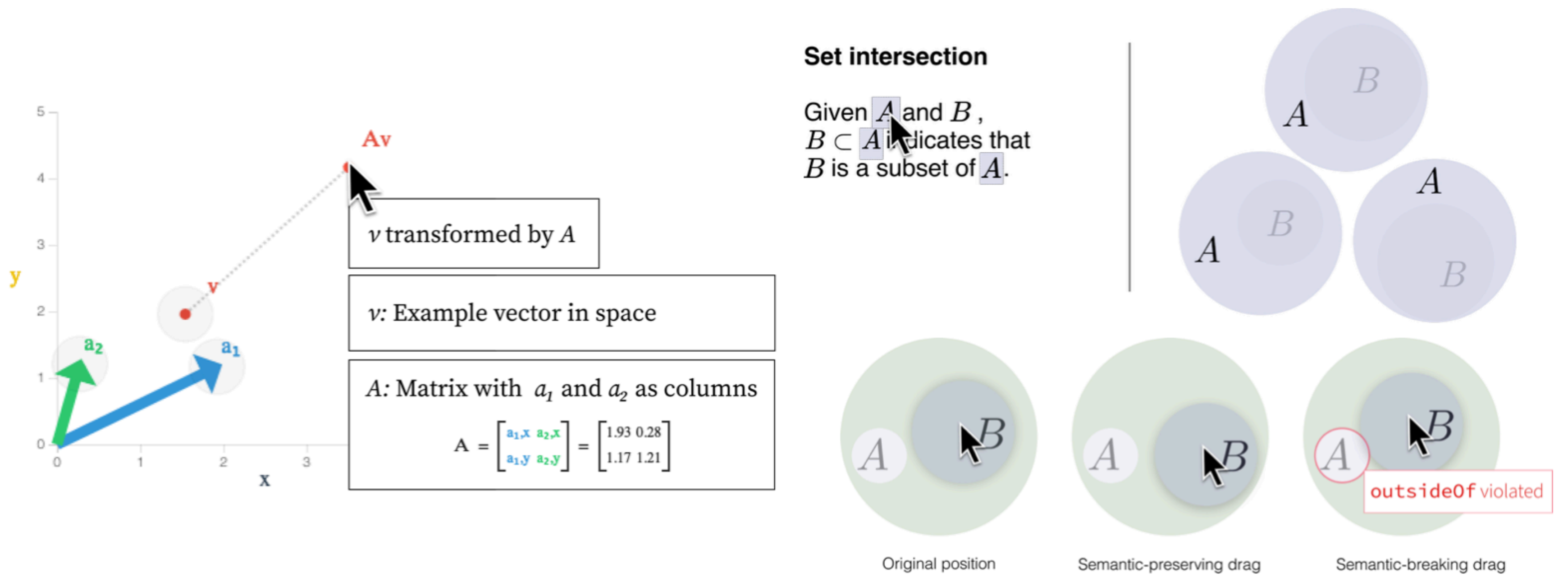
From encoding to semantic-preserving interactivity
(Proposed)





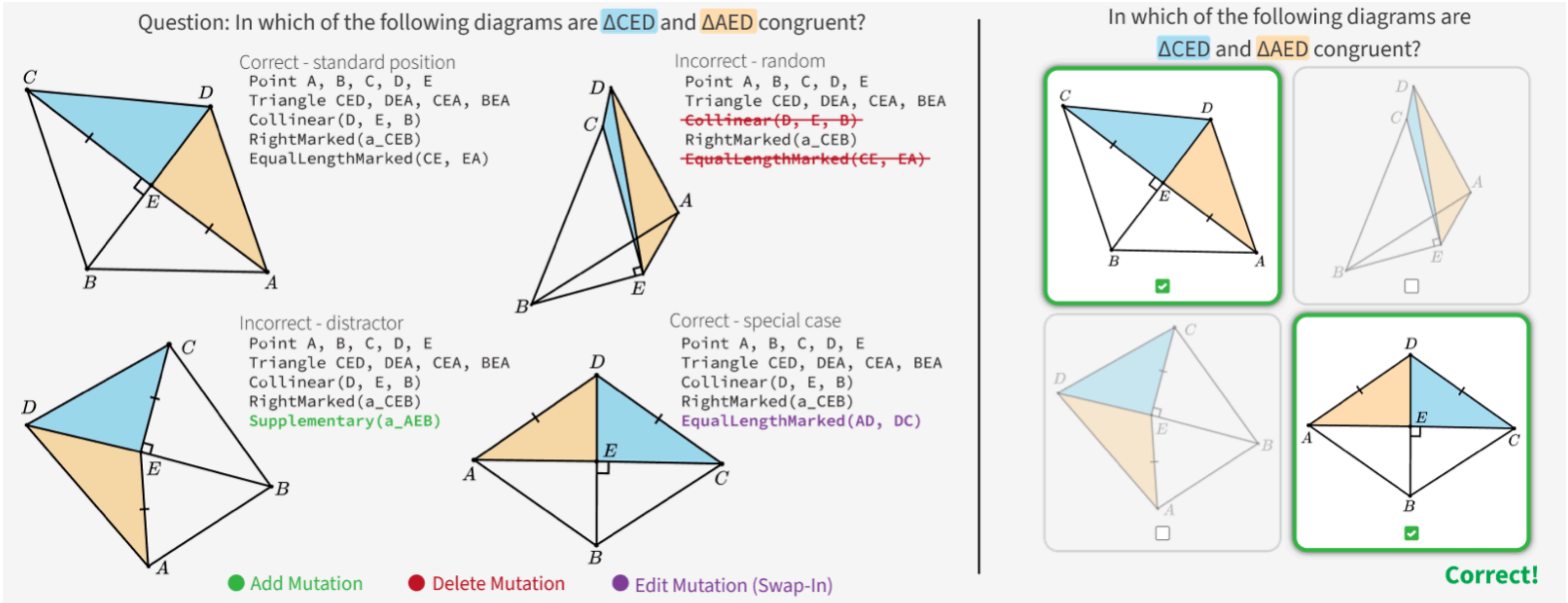
Understanding the diagramming process and encoding visual representations

Encoding visual representations in diagramming tools simplifies programming of **interactive visual activities** that provide students with automated feedback at scale.



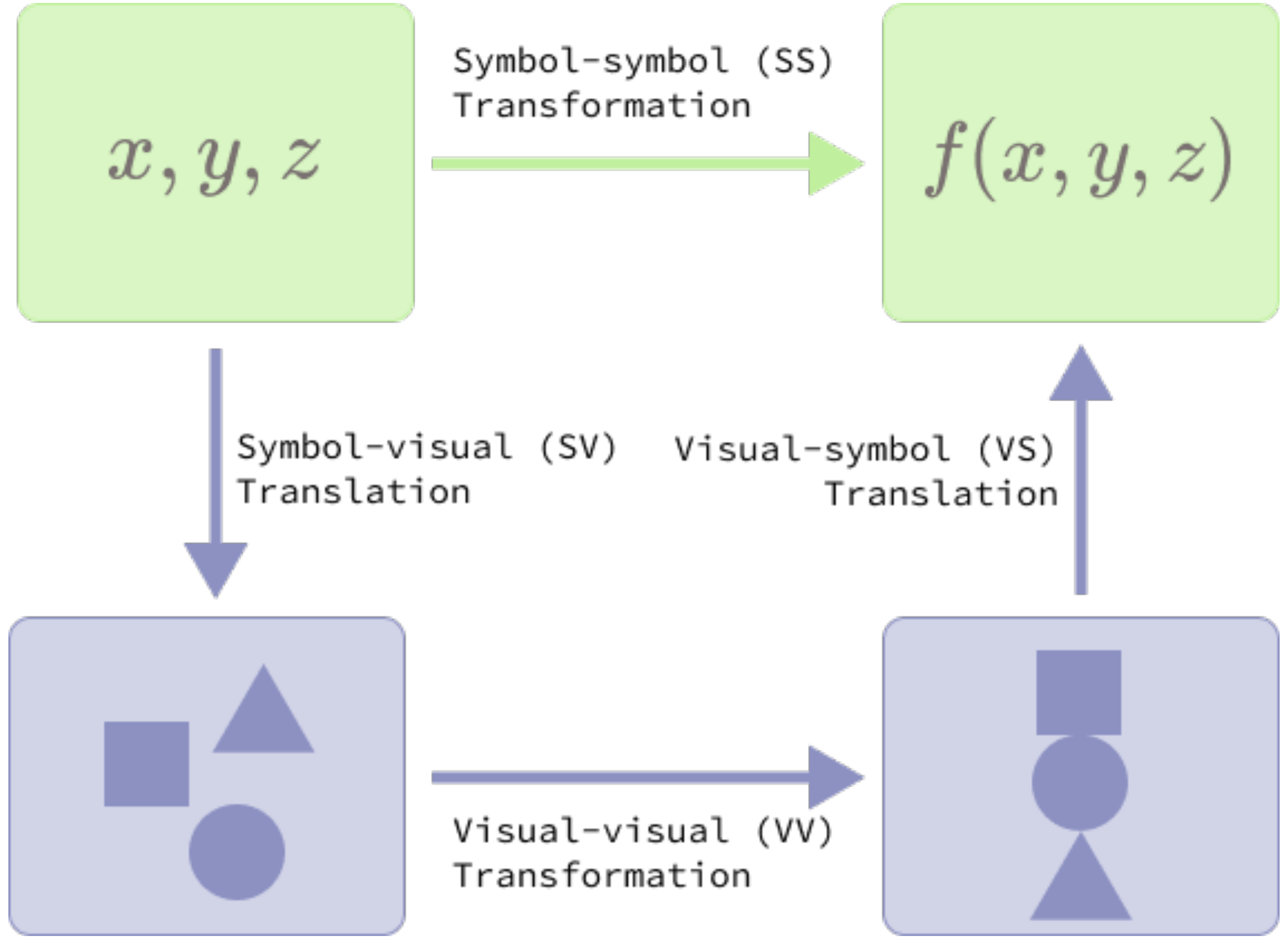
From encoding to semantic-preserving interactivity

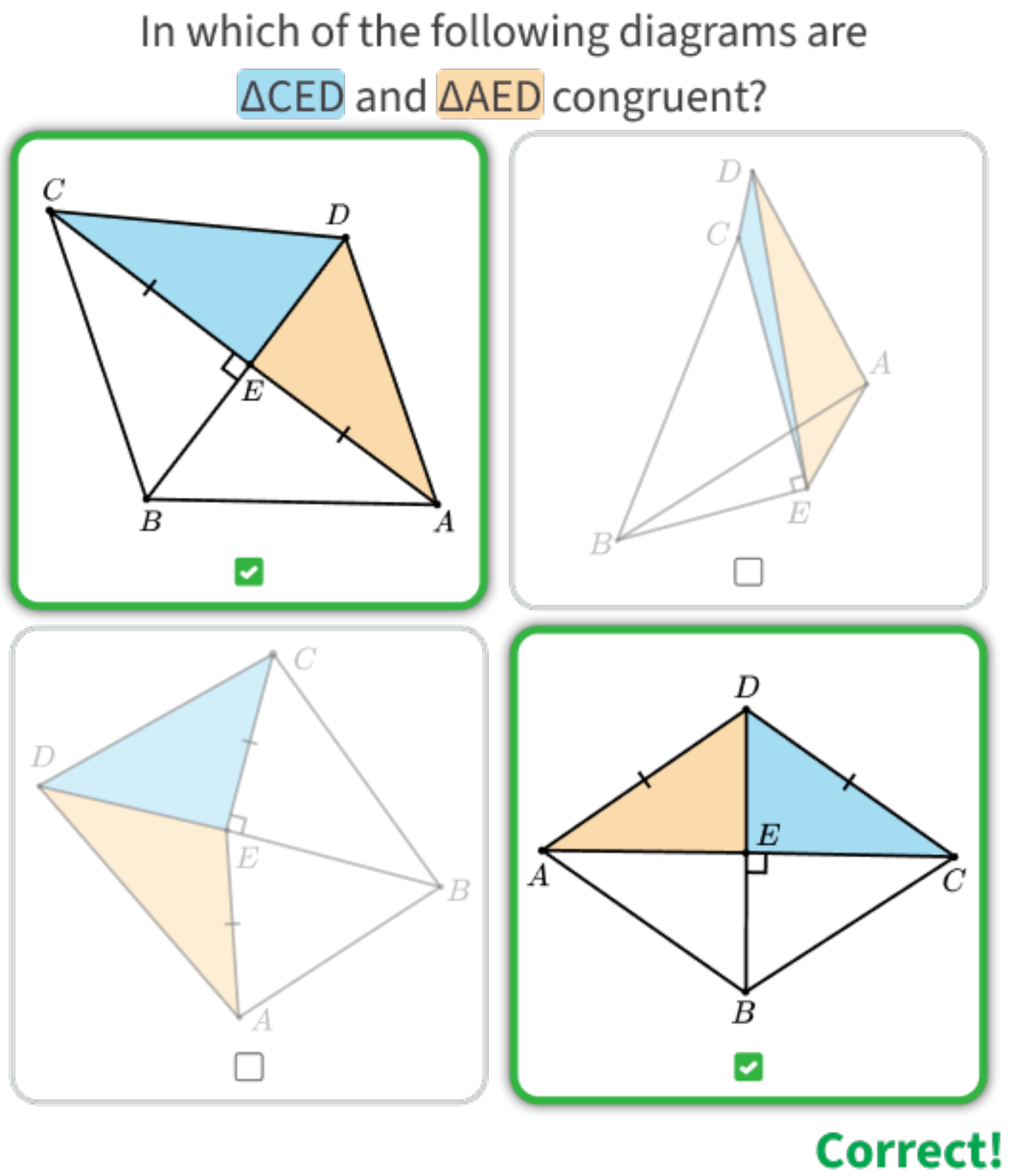
(Proposed)

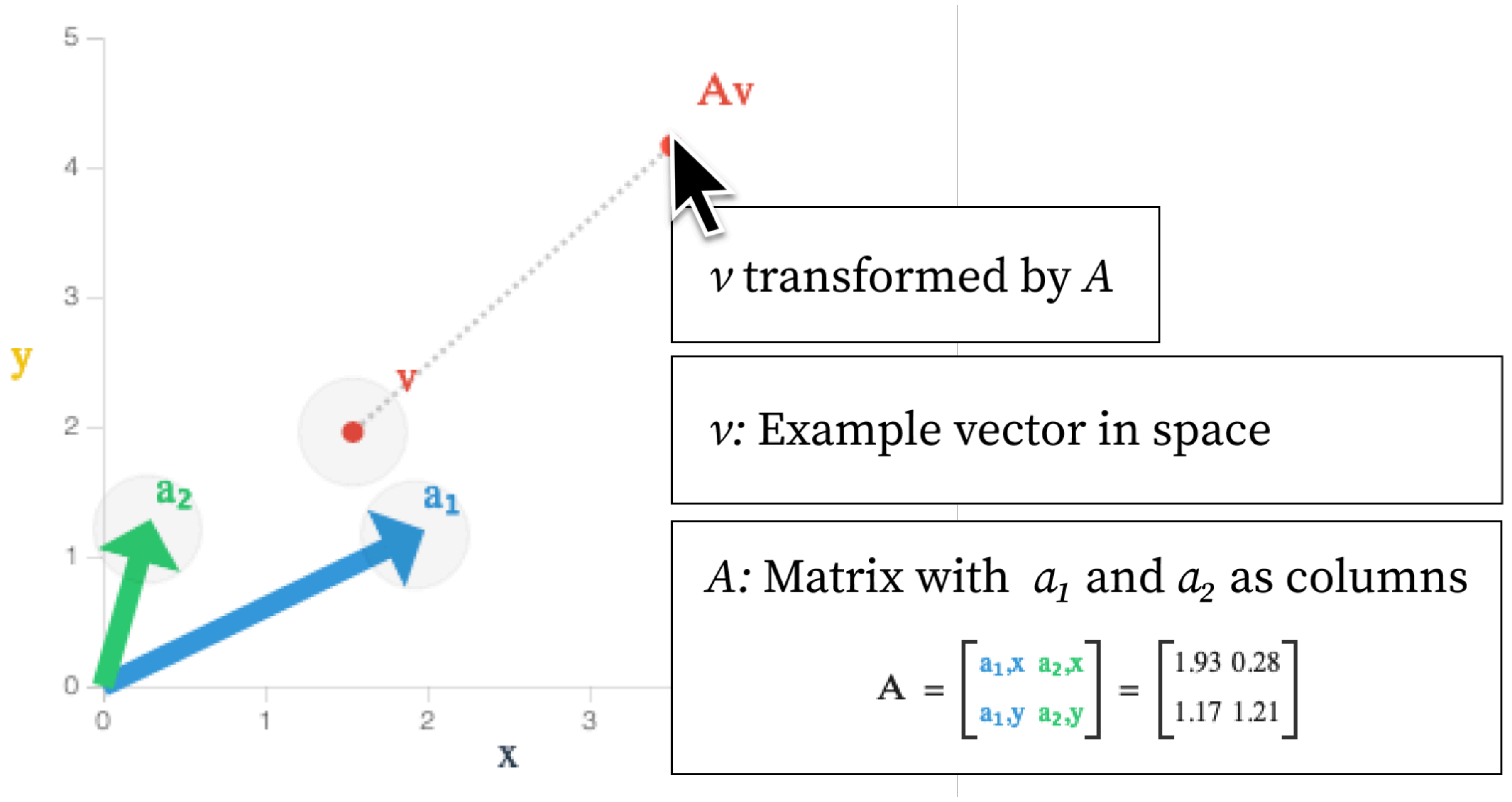


Edgeworth: Diagrammatic Content Authoring at Scale

(In progress)





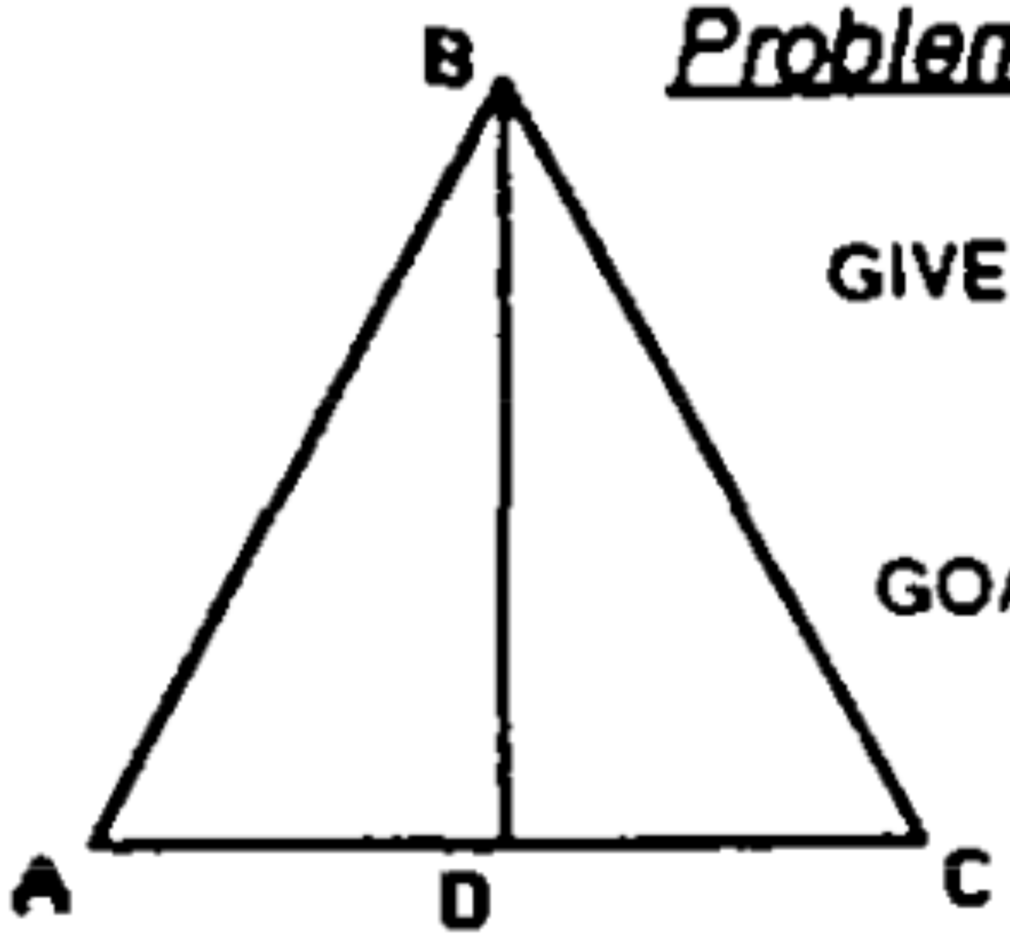


Developing conceptual understanding through interactive diagramming

Wode "Nimo" Ni

Backup slides

Step-skipping in geometry proofs



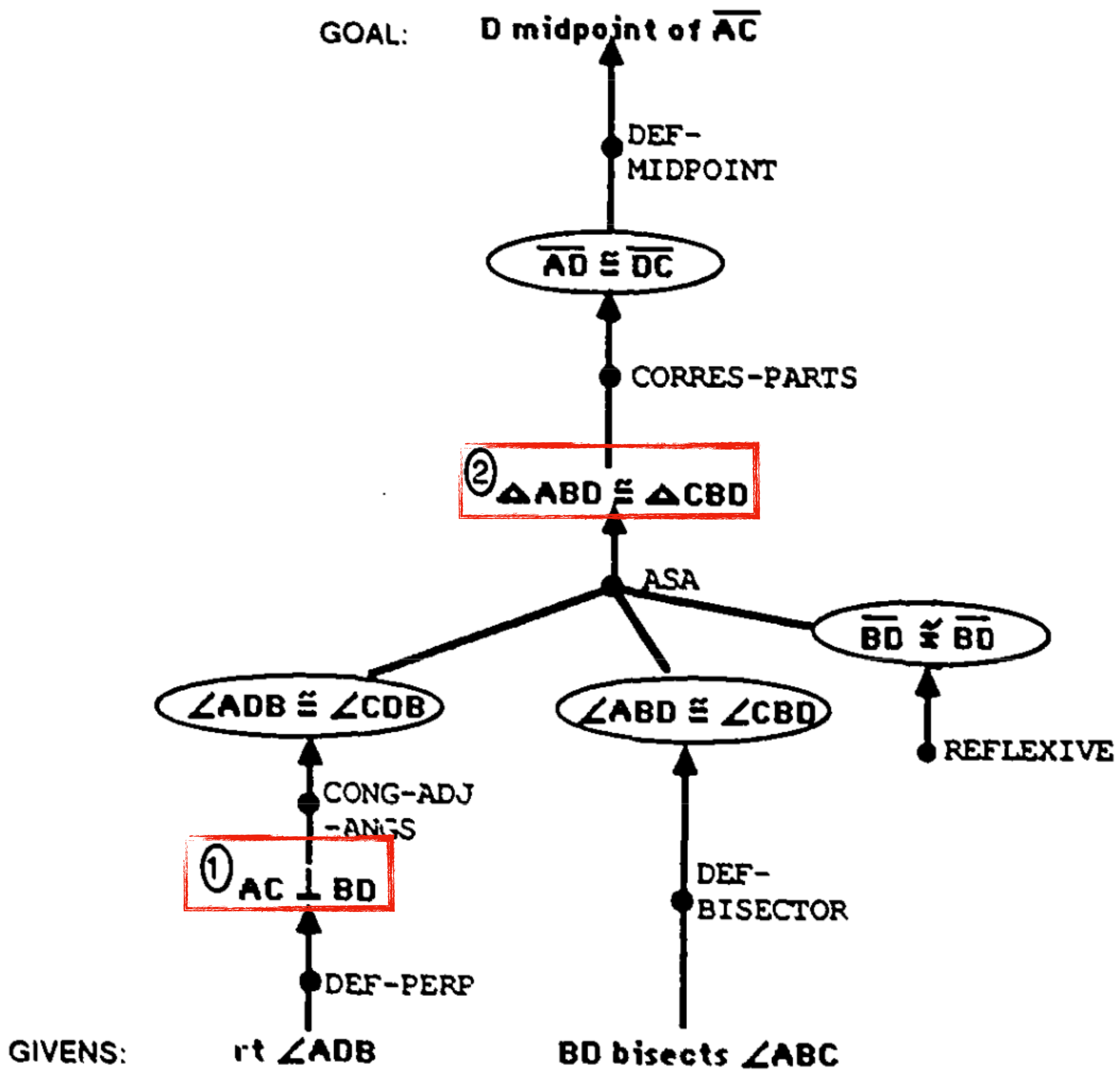
Problem 3

GIVENS: rt ∠ADB
$\overline{BD}$ bisects ∠ABC

GOAL: D midpoint of $\overline{AC}$

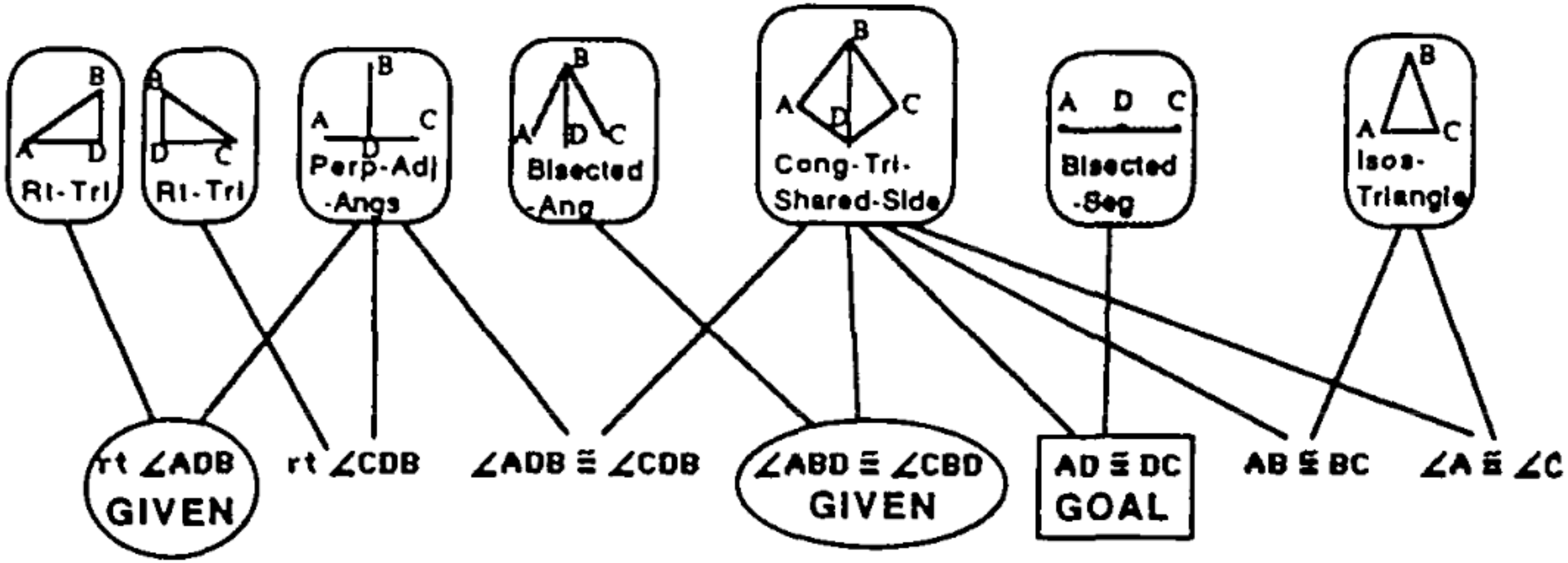
TABLE 1
A Verbal Protocol for a Subject Solving Problem 3

	Planning Phase
B1: We're given a right angle—this is a right angle,	Reading given: rt ∠ADB
B2: perpendicular on both sides [makes perpendicular markings on diagram];	Inference step 1: AC⊥BD
B3: BD bisects angle ABC [marks angles ABD and CBD]	Reading given: BD bisects ∠ABC
B4: and we're done.	Inference step 2: ΔABD≅ΔCBD



Koedinger & Anderson, 1990. Abstract Planning and Perceptual Chunks: Elements of Expertise in Geometry.

How experts solved it: diagram configuration



Conceptual knowledge: understanding of the principles that govern a domain and of the interrelations between units of knowledge in a domain. This knowledge is ***flexible*** and not tied to specific problem types and is therefore ***generalizable***.

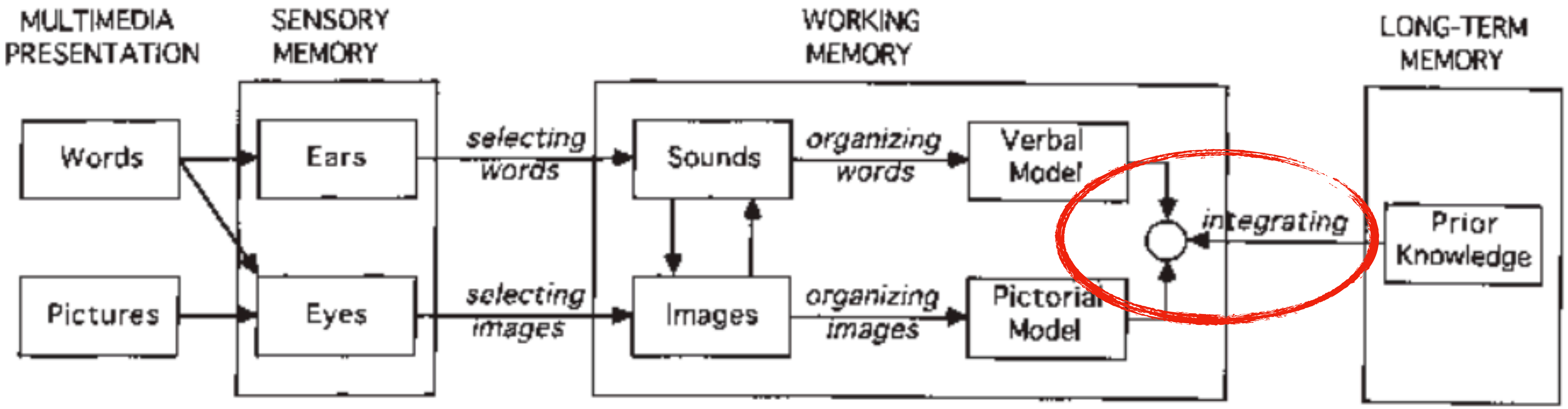
Rittle-Johnson, Siegler & Alibali, 2001

Principle	*Median Effect Size*
Multimedia	1.39
Contiguity	1.10
Coherence	0.86
Modality	0.76
Redundancy	0.86
Personalization:	0.79

“The Multimedia Principle”

Multiple representations improve knowledge retention and transfer in problem solving.

Mayer & Moreno, 2010



> *Jacques Hadamard […] decided to poll […] 100 great mathematicians and physicists on the earth, and he asked them, "**How do you do your thing**?" […] Quite a surprise. All of them said they did it **mostly in imagery or figurative terms**.*

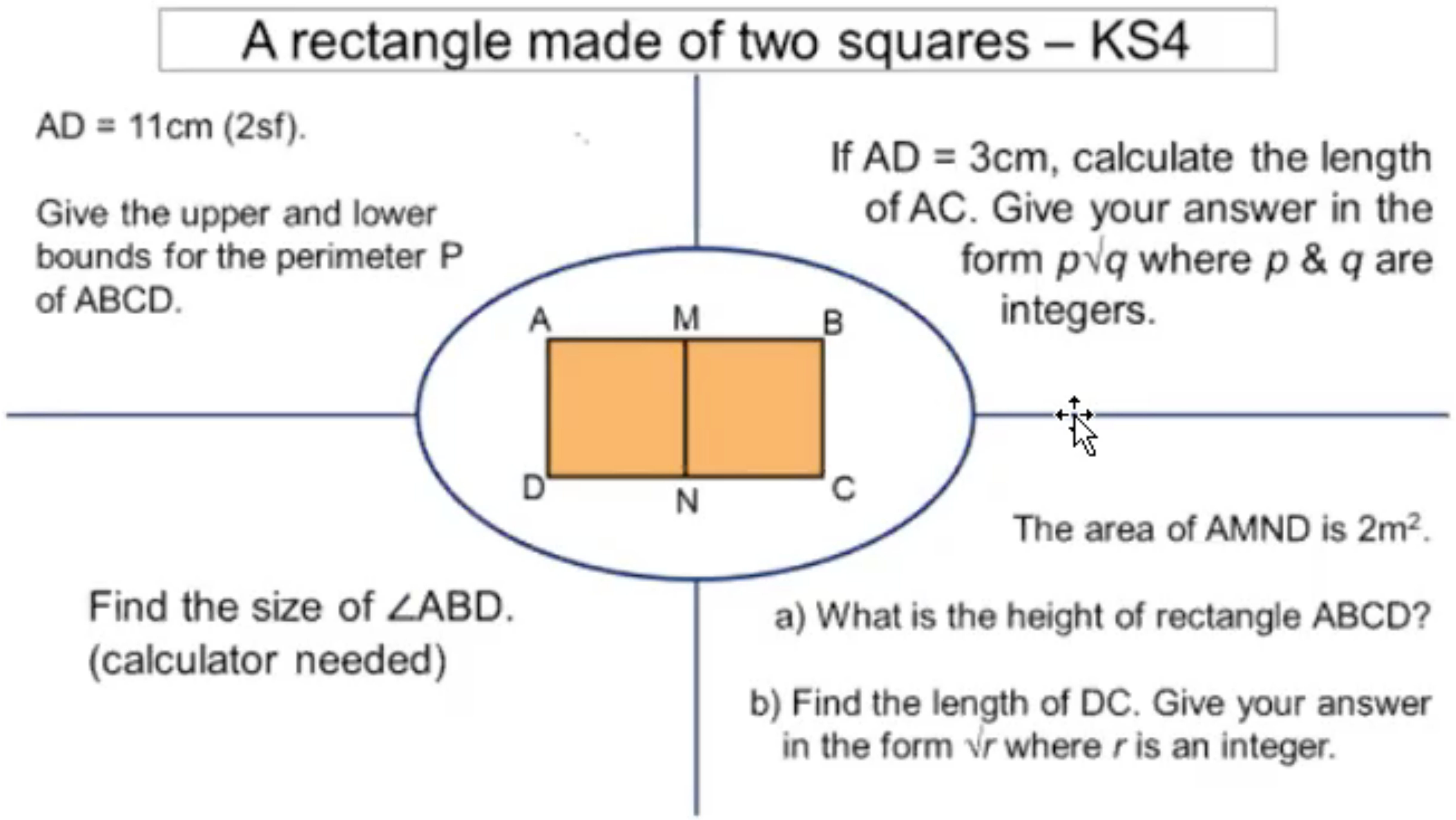
> *The sad part of the diagram is that **every child in the United States is taught math and physics through this [symbolic] channel**. The channel that almost no adult creative mathematician or physicist uses to do it… They use this channel to communicate, but not to do their thing.*

Alan Kay: Doing with Images Makes Symbols (1987)

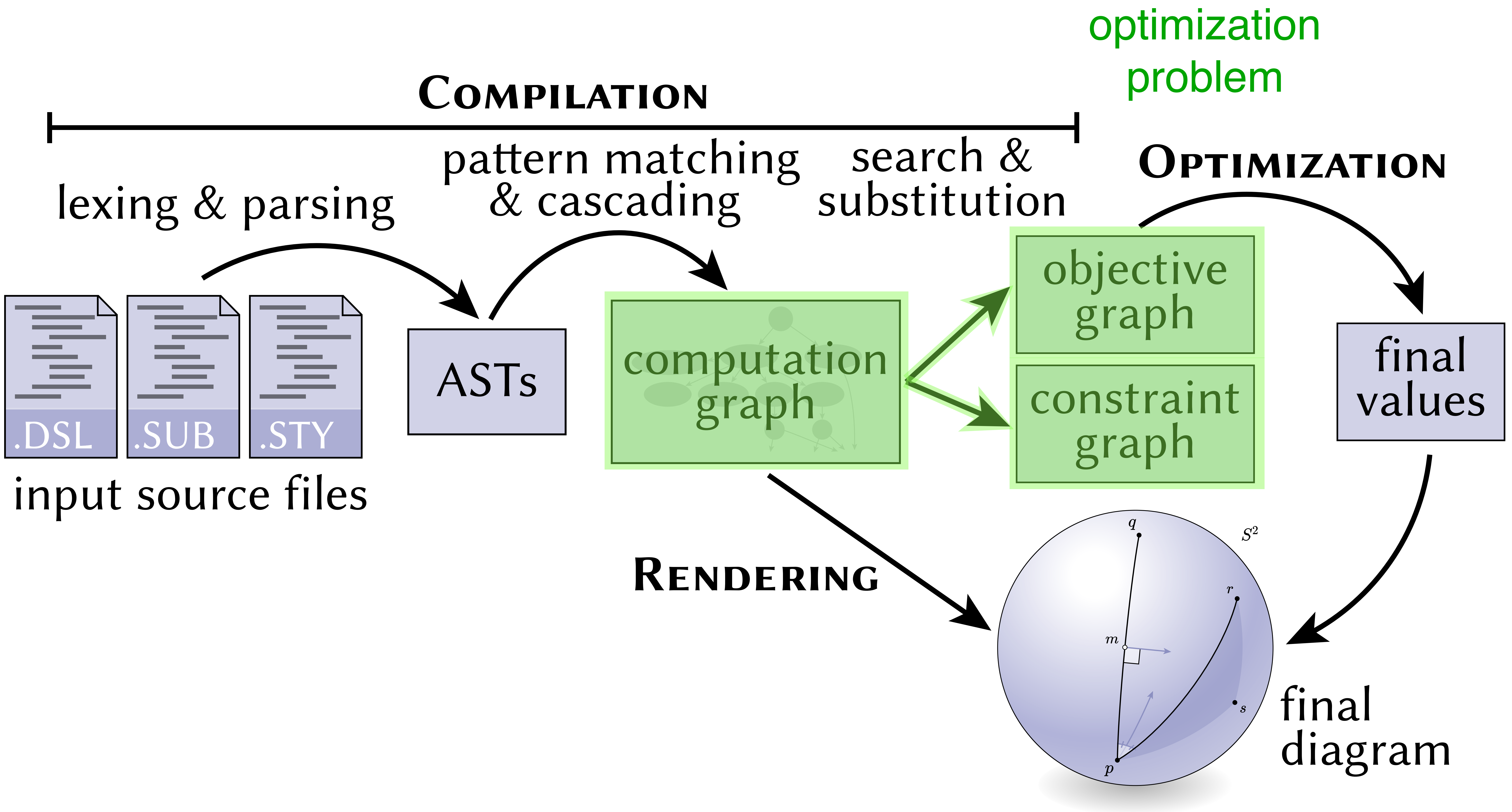
Why don't you just make more diagrams?

*"[Diagram-making] is **mostly copy-pasting**. It's really hard to find, say, a component for a diagram." (P1)*

*"Diagrammatic problems are **complex and take more time to grade**, but I'm willing to spend the extra effort to design and grade them." (P6)*

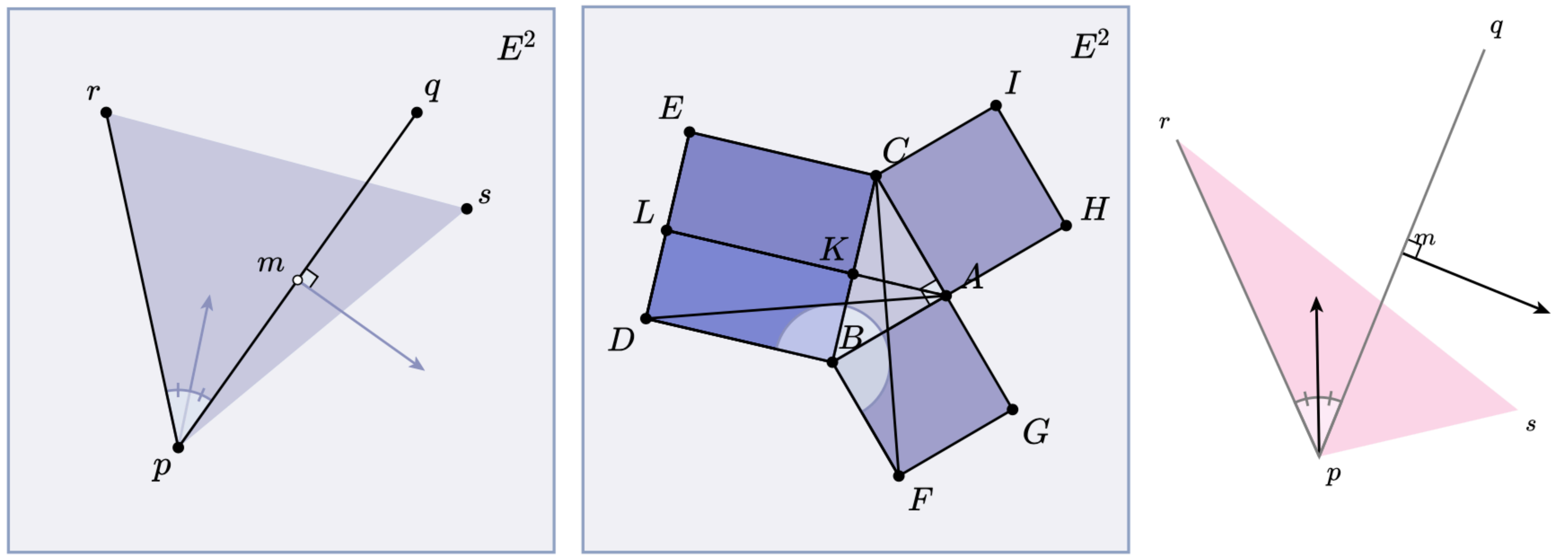


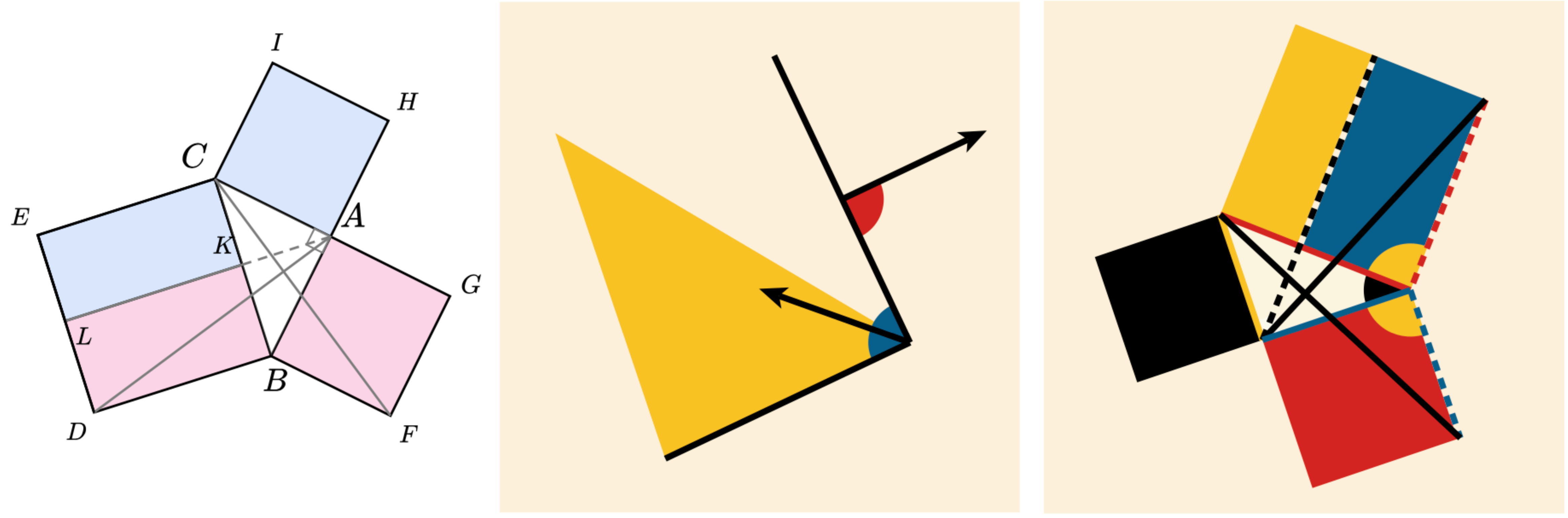
Penrose pipeline links specification to synthesis



Visualizing geometry

changing the styling

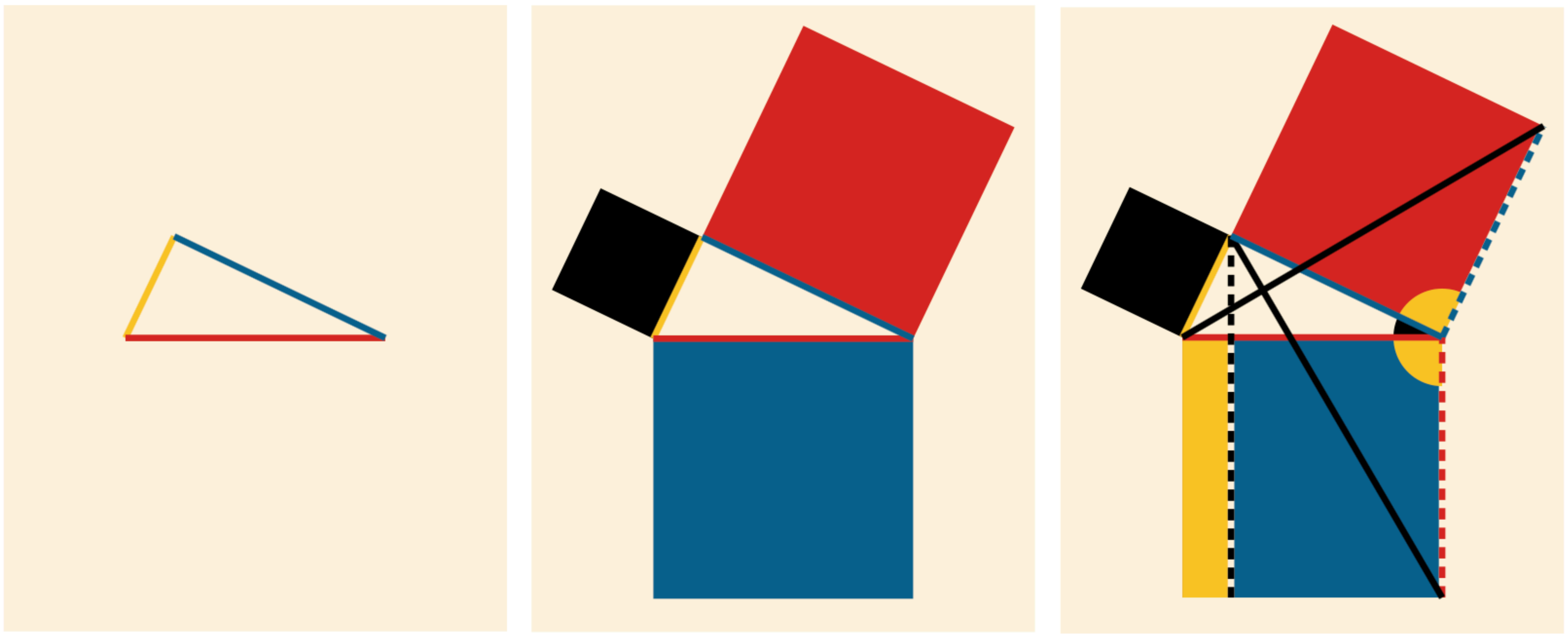





```
Point A, B, C
-- define a right triangle
Triangle ABC := {A,B,C}
Angle θ := ∠(C,A,B)
Right(θ)
-- square each side
Point D, E, F, G, H, I
Square CBDE := [C,B,D,E]
Disjoint(CBDE, ABC)
Square BAGF := [B,A,G,F]
Disjoint(BAGF, ABC)
Square ACIH := [A,C,I,H]
Disjoint(ACIH, ABC)
```

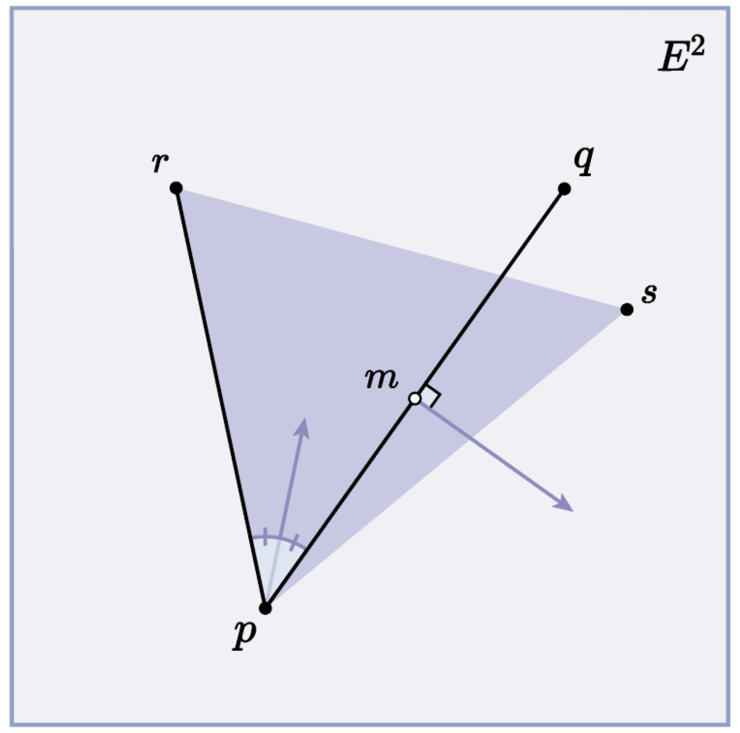
```
-- PythagoreanTheorem.sub
-- split hypotenuse area
Segment AK := Altitude(ABC,θ)
Point K := Endpoint(AK)
Segment DE := {D,E}
Point L
On(L, DE)
Segment KL := {K,L}
Perpendicular(KL, DE)
Rectangle BDLK := {B,D,L,K}
Rectangle CKLE := {C,K,L,E}
-- (plus additional objects
-- from Byrne's diagram)
```

sequences of diagrams

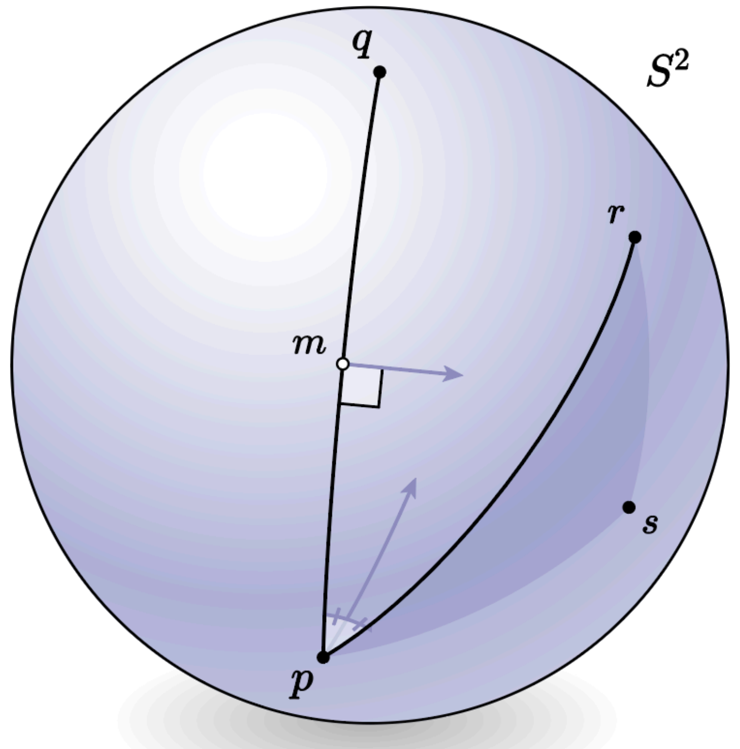


Visualizing geometry

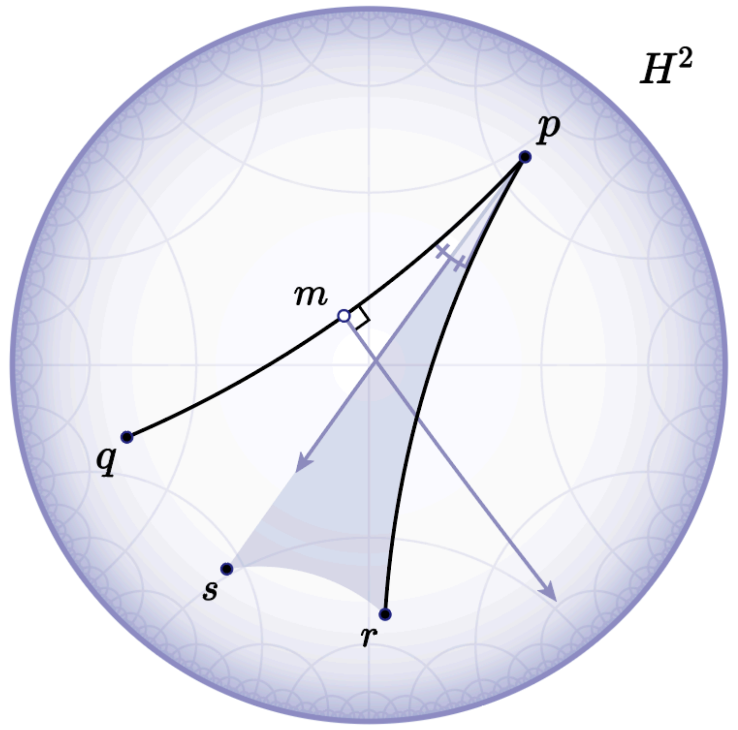
changing the geometric interpretation



Style—Euclidean



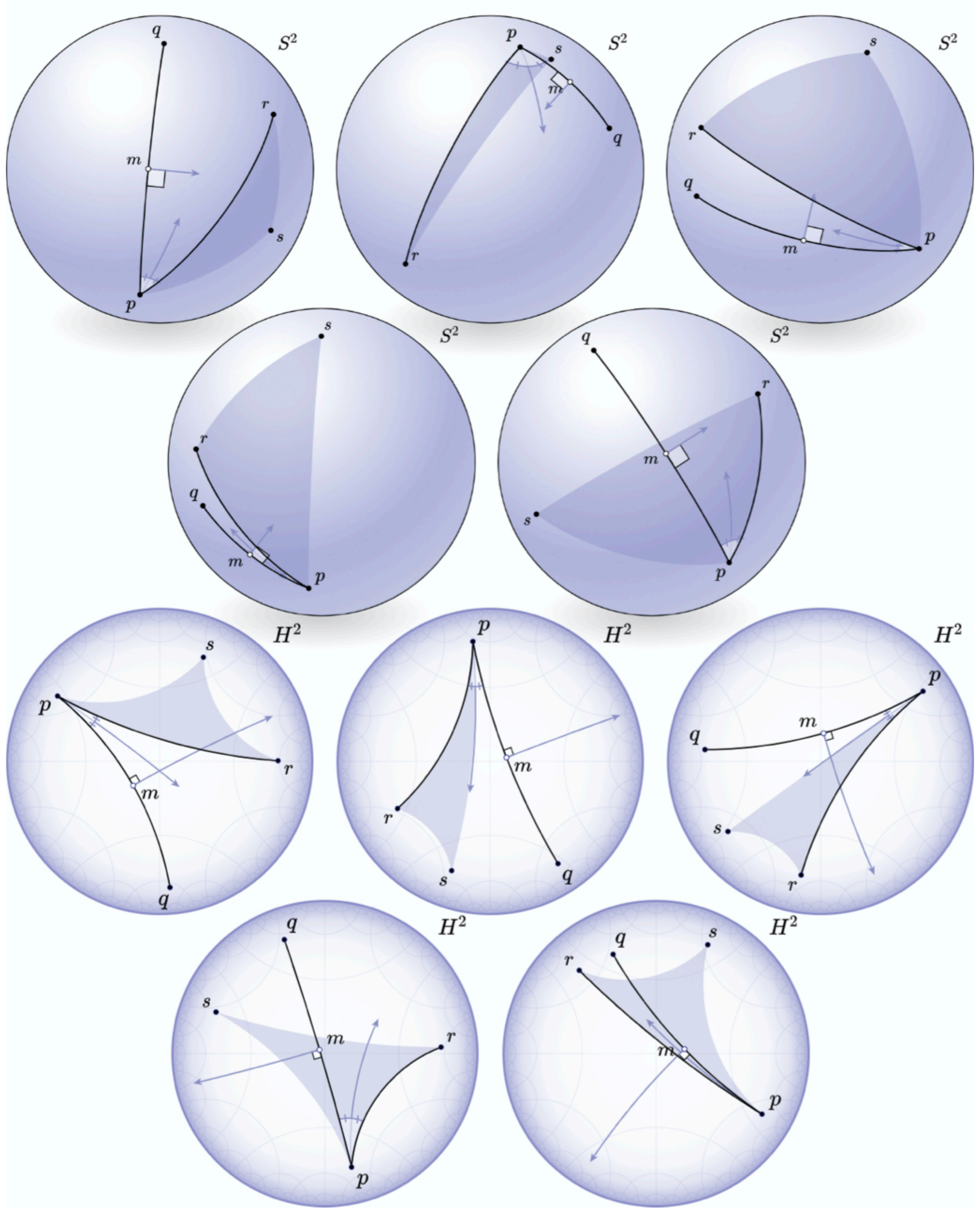
Style—spherical

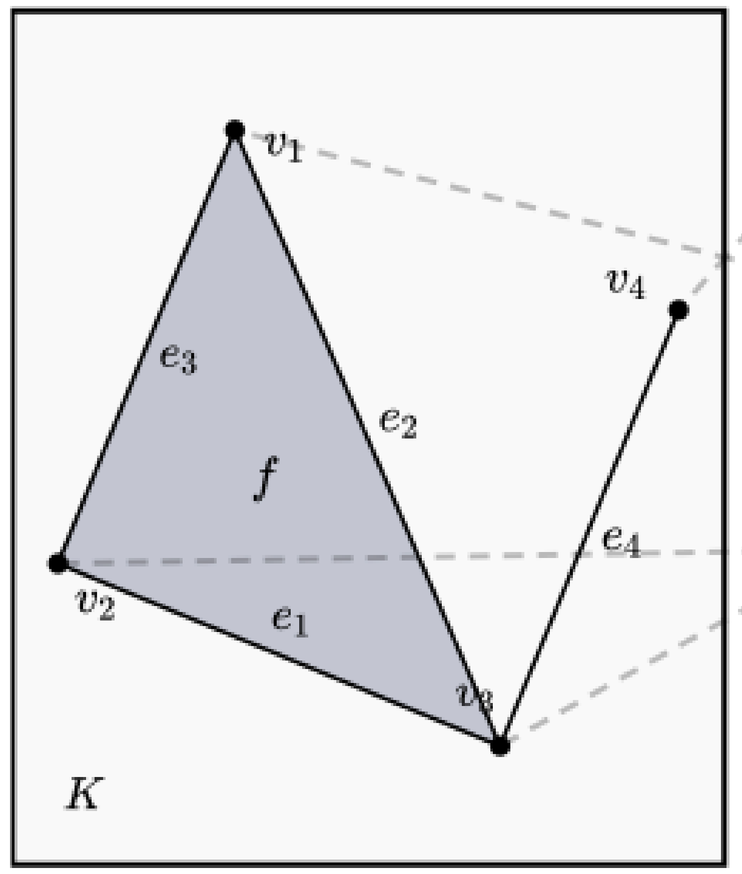


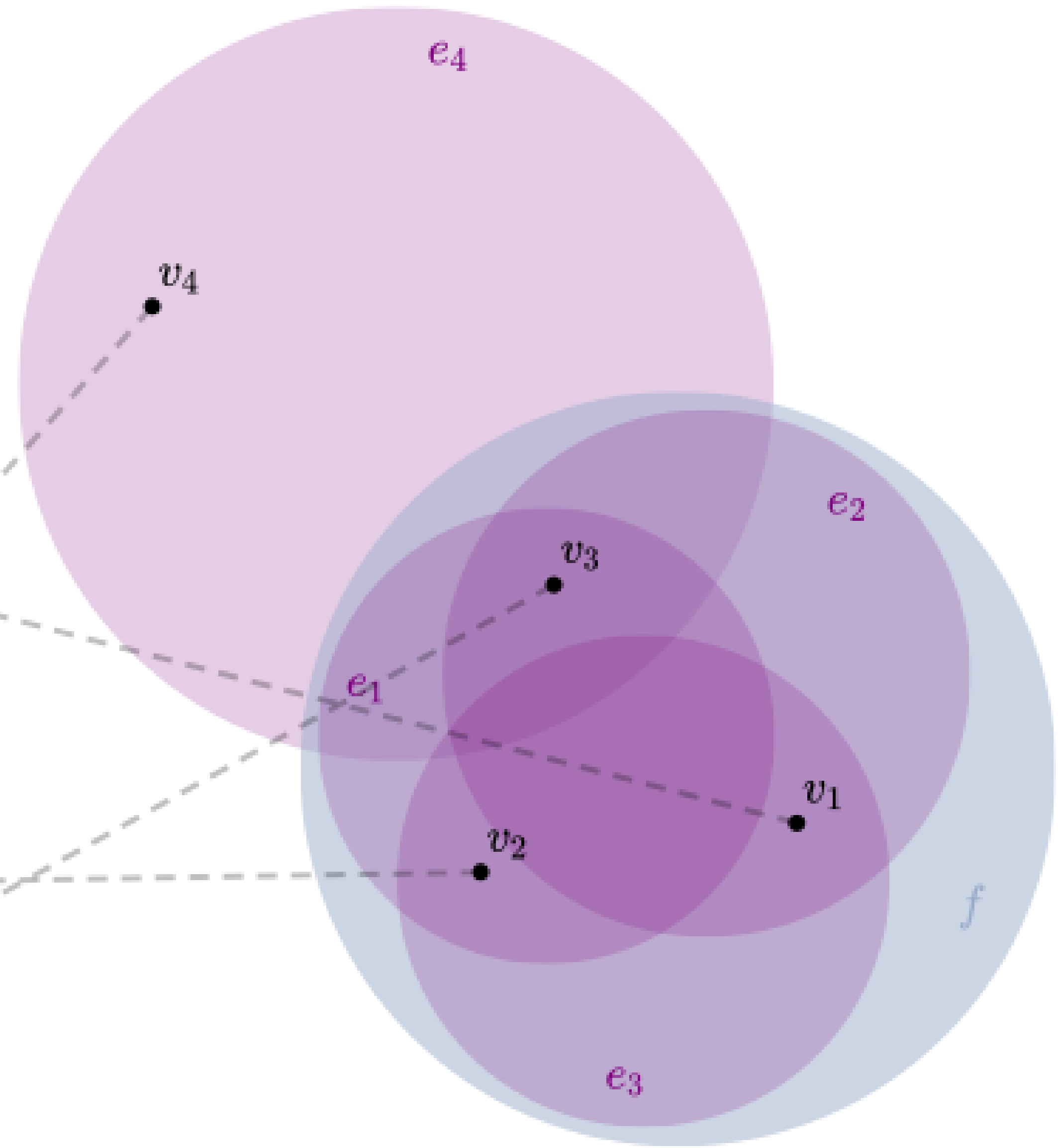
Style—hyperbolic

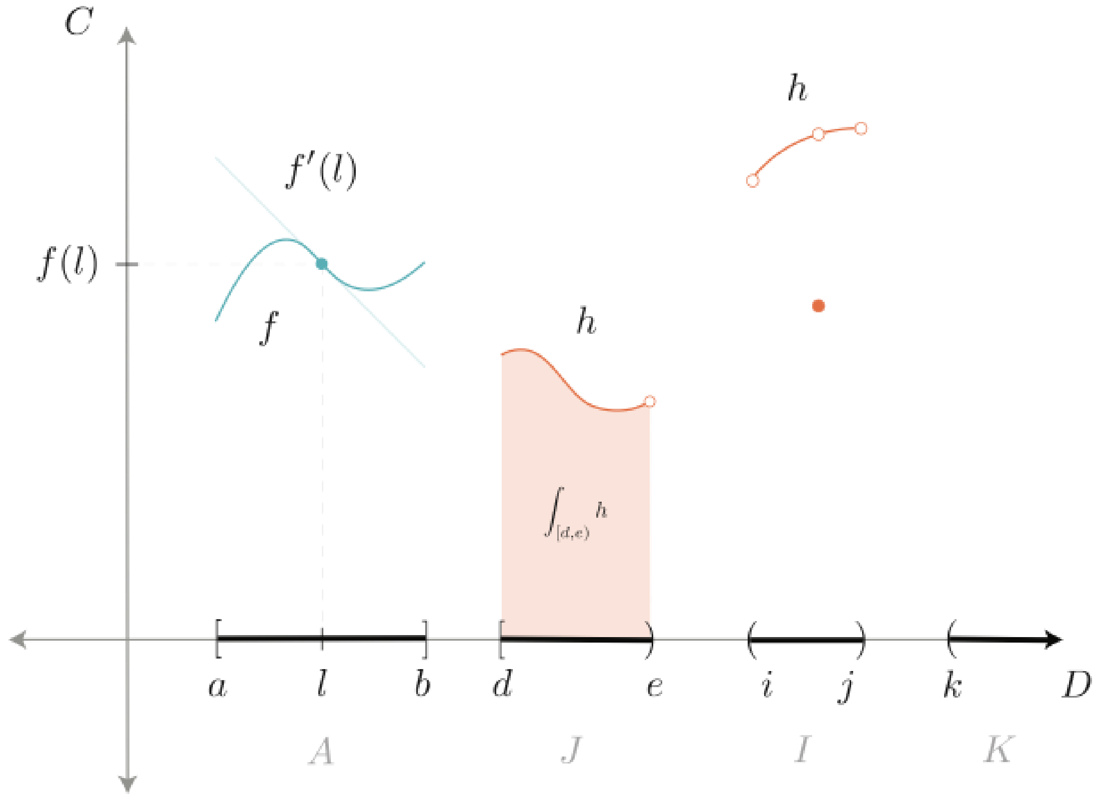
```
Point p, q, r, s
Segment a := p, q
Segment b := p, r
Point m := Midpoint(a)
Angle theta := ∠(q, p, r)
Triangle t := p, r, s
Ray w := Bisector(theta)
Ray h := PerpendicularBisector(a)
```

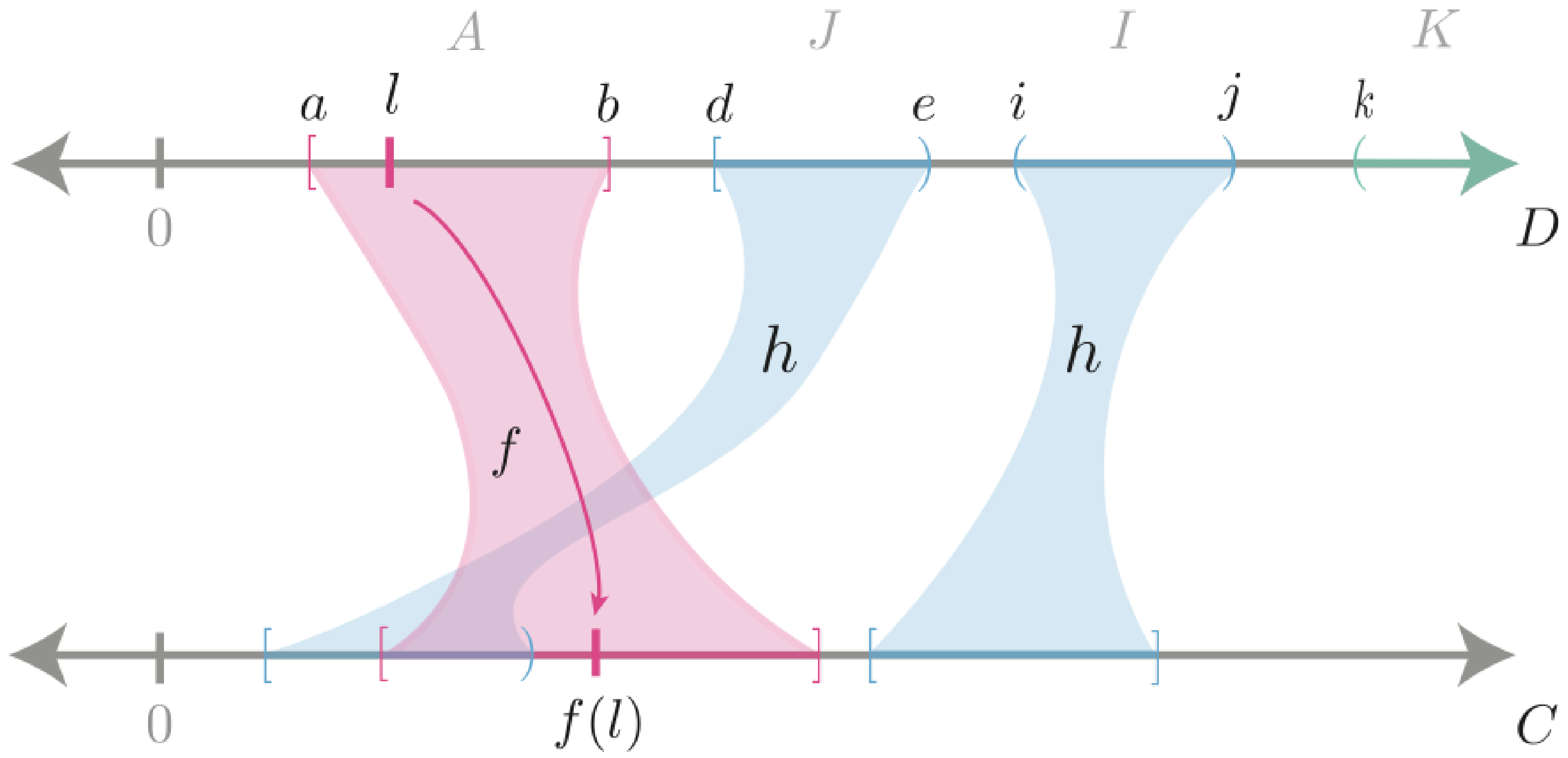
generating a gallery of alternatives

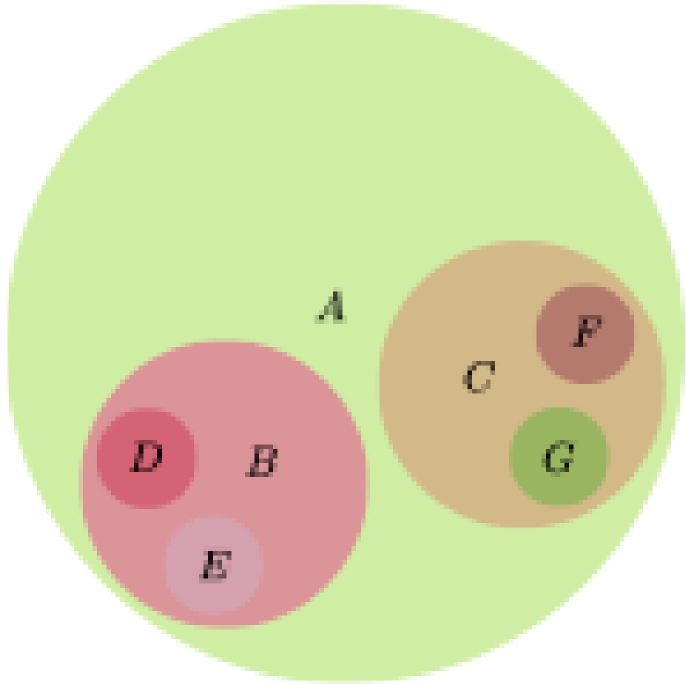


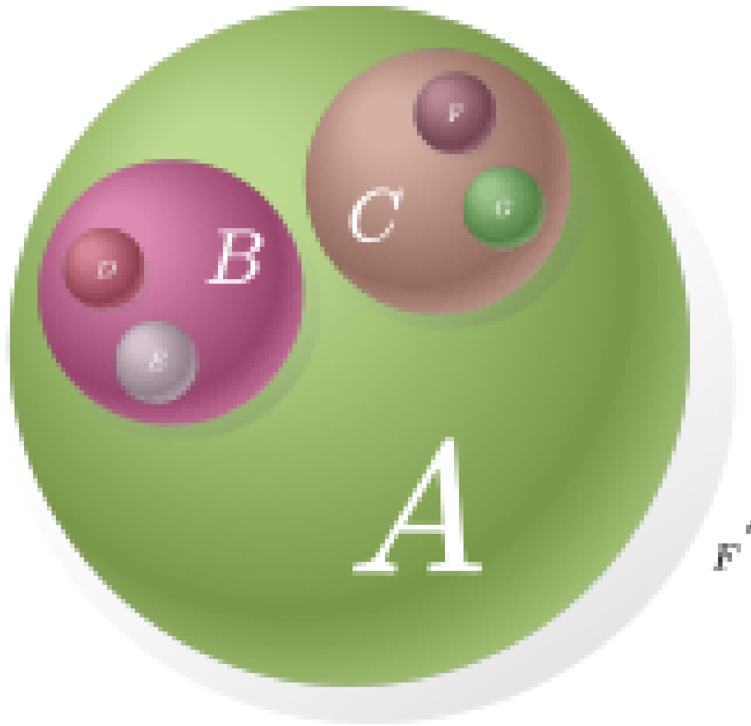


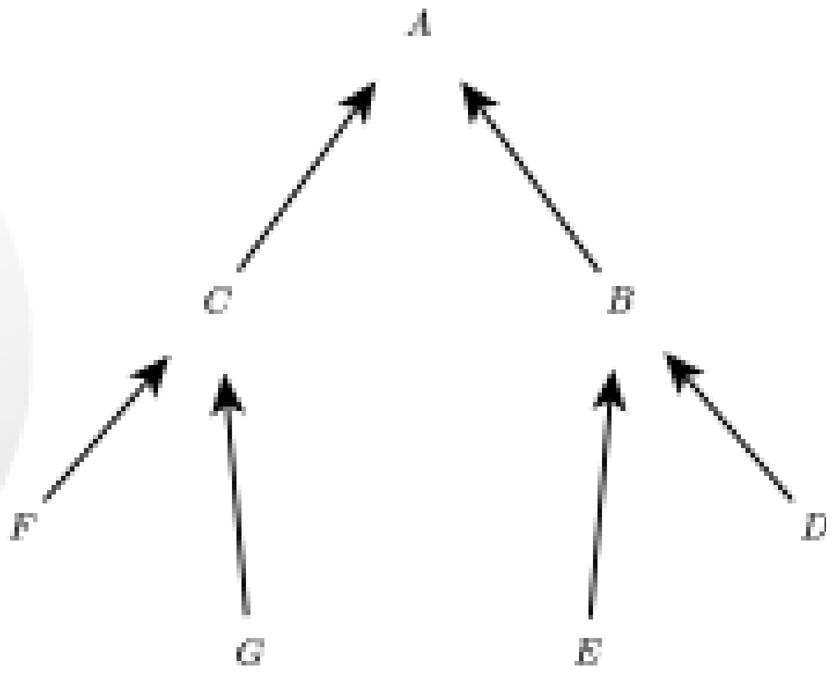


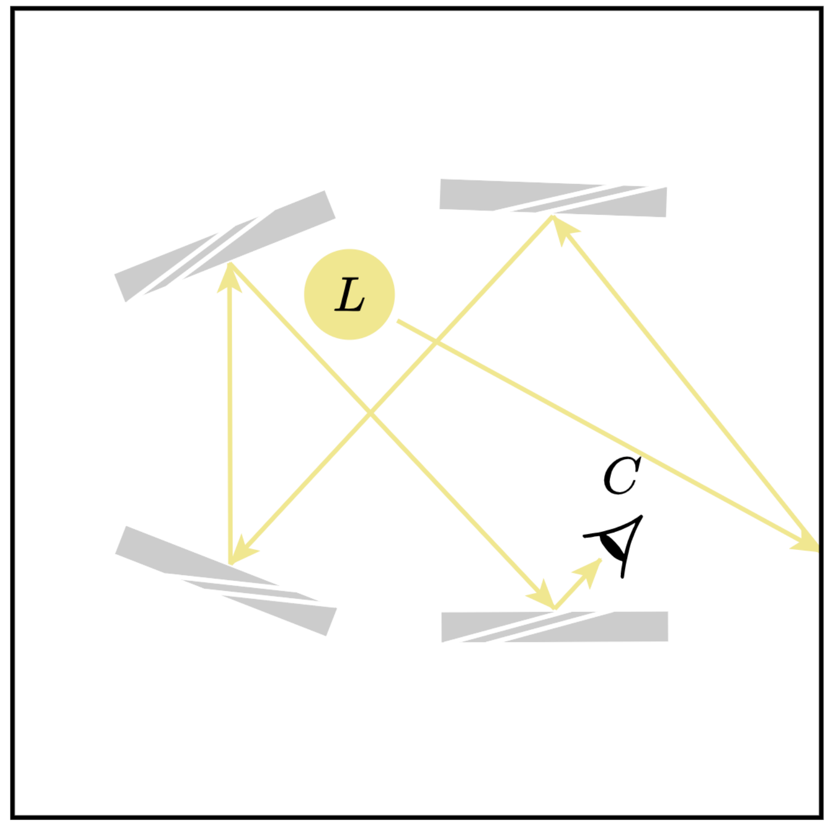


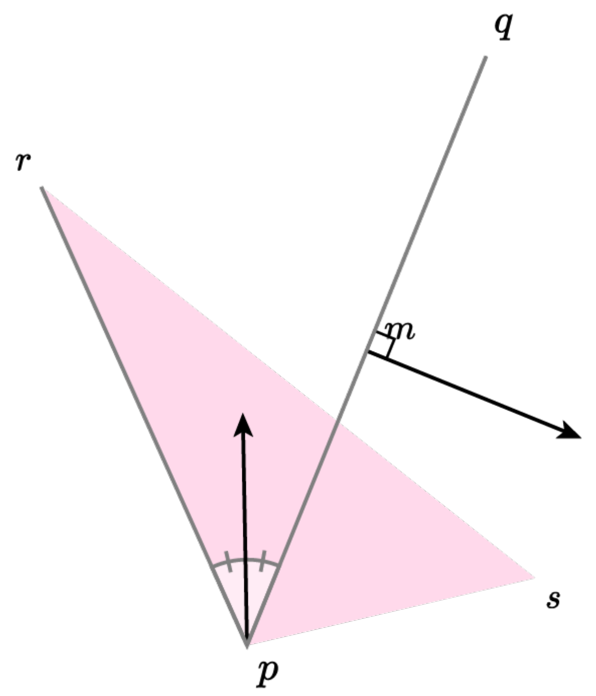


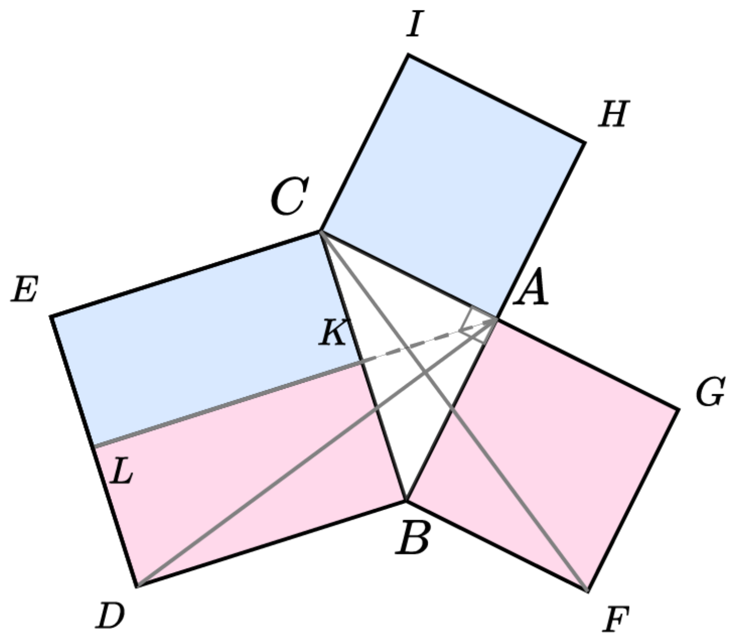










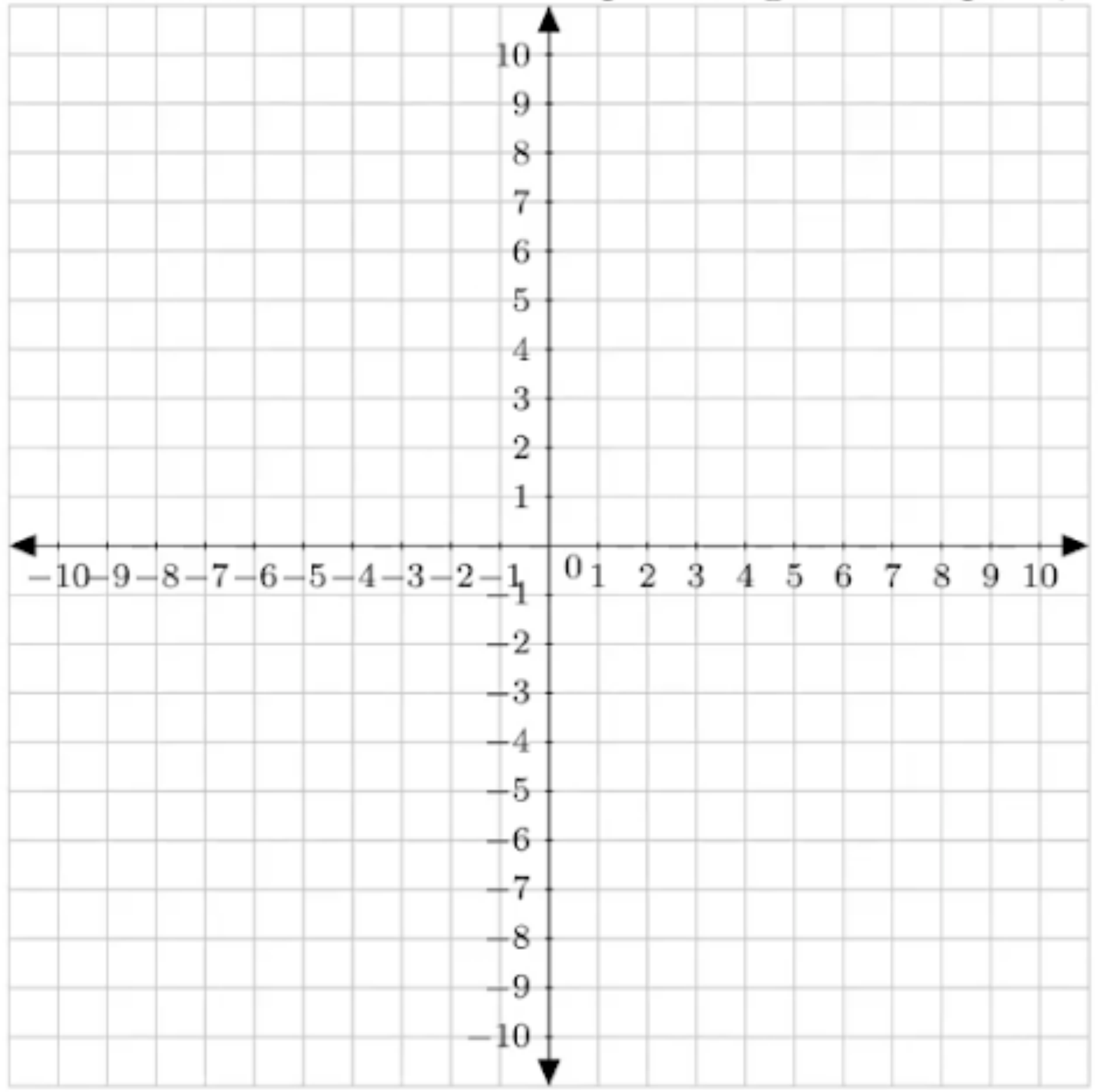


github.com/penrose/penrose

Existing diagramming tools...

```latex
\def\blankgraph{
    \definecolor{cqcqcq}{rgb}{0.7529411764705882,0.7529411764705882,0.7529411764705882}

    \begin{tikzpicture}[line cap=round,line join=round,>=triangle 45,x=1.0cm,y=1.0cm,scale=.4]
        \draw [color=cqcqcq,, xstep=1.0cm,ystep=1.0cm] (-11.,-11.) grid (11.,11.);
        \draw[<->,color=black] (-11.,0.) -- (11.,0.);
        \foreach \x in {-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10}
        \draw[shift={(\x,0)},color=black] (0pt,2pt) -- (0pt,-2pt) node[below] {\footnotesize $\x$};
        \draw[<->,color=black] (0.,-11.) -- (0.,11.);
        \foreach \y in {-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10}
        \draw[shift={(0,\y)},color=black] (2pt,0pt) -- (-2pt,0pt) node[left] {\footnotesize $\y$};
        \draw[color=black] (0pt,-12pt) node[right] {\footnotesize $0$};
        \clip(-11.,-11.) rectangle (11.,11.);
    \end{tikzpicture}
```

1. (6 points) Three points in the plane are $A(1,6)$, $B(-2,-2)$, and $C(4,-3)$. There are 3 possible points for D that will make $ABCD$ a parallelogram. Graph A, B, and C and list all 3 possible points D.



(a) D_1

(b) D_2

(c) D_3

(a) ____________

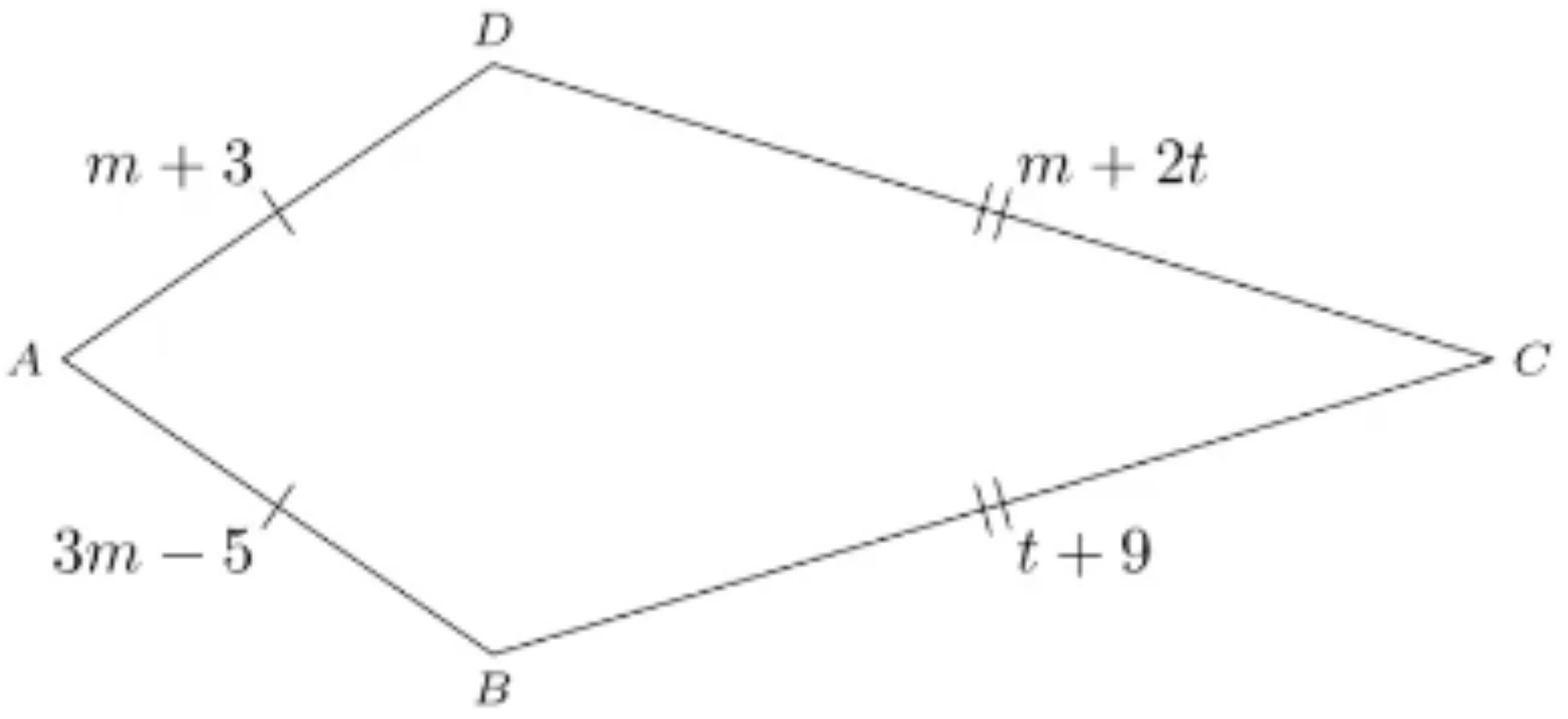
(b) ____________

(c) ____________

How Domain Experts Create Conceptual Diagrams and Implications for Tool Design. CHI'20

Existing diagramming tools…

*“Once I make [the diagram], it’s **hard to change anything**. I want a **tweakable authoring tool** for changing my diagrams along with the problems.” (P2)*



(a) $m =$

(b) $t =$

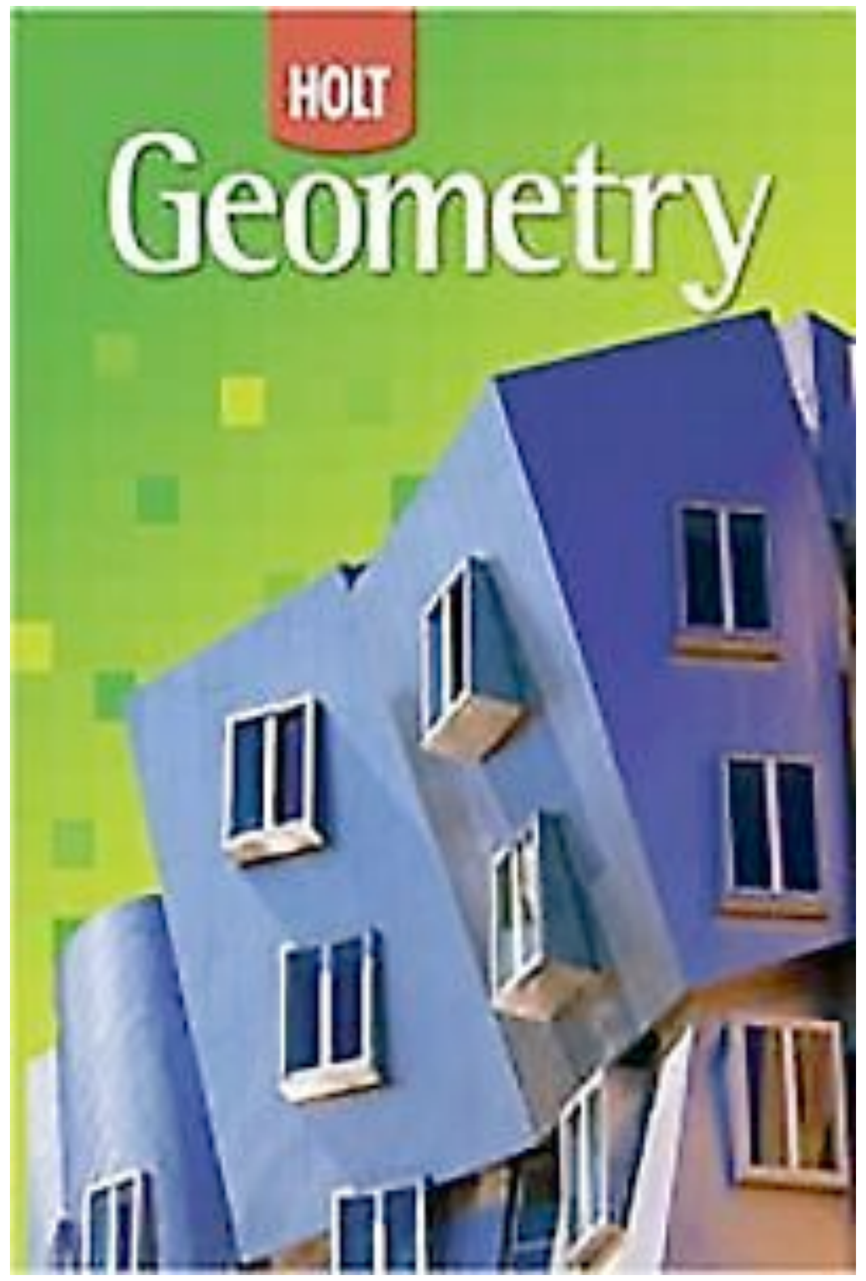
(c) $AB =$

(d) $BC =$

(e) $CD =$

How Domain Experts Create Conceptual Diagrams and Implications for Tool Design. CHI’20

Prelim. Evaluation: re-creating textbook problems



20 Diagrams generated/trial

1-3 Mutations/program

<5 Programs to see first correct and incorrect diagrams

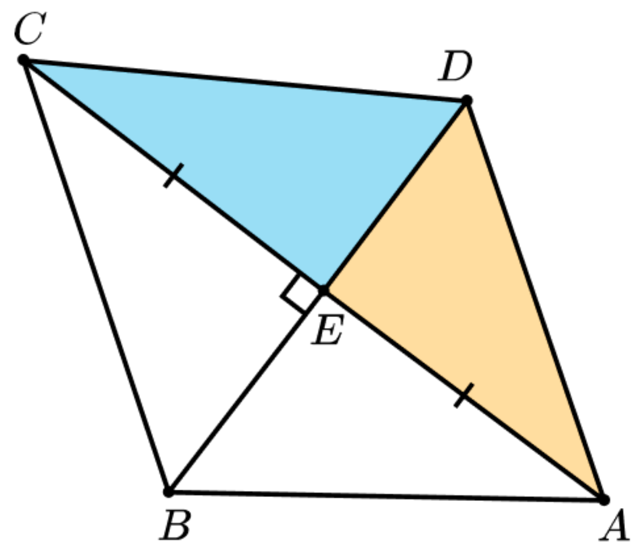
1-2 Special cases/trial*

1-5 Distractors/trial*

* results vary based on tuning of configuration parameters in Edgeworth

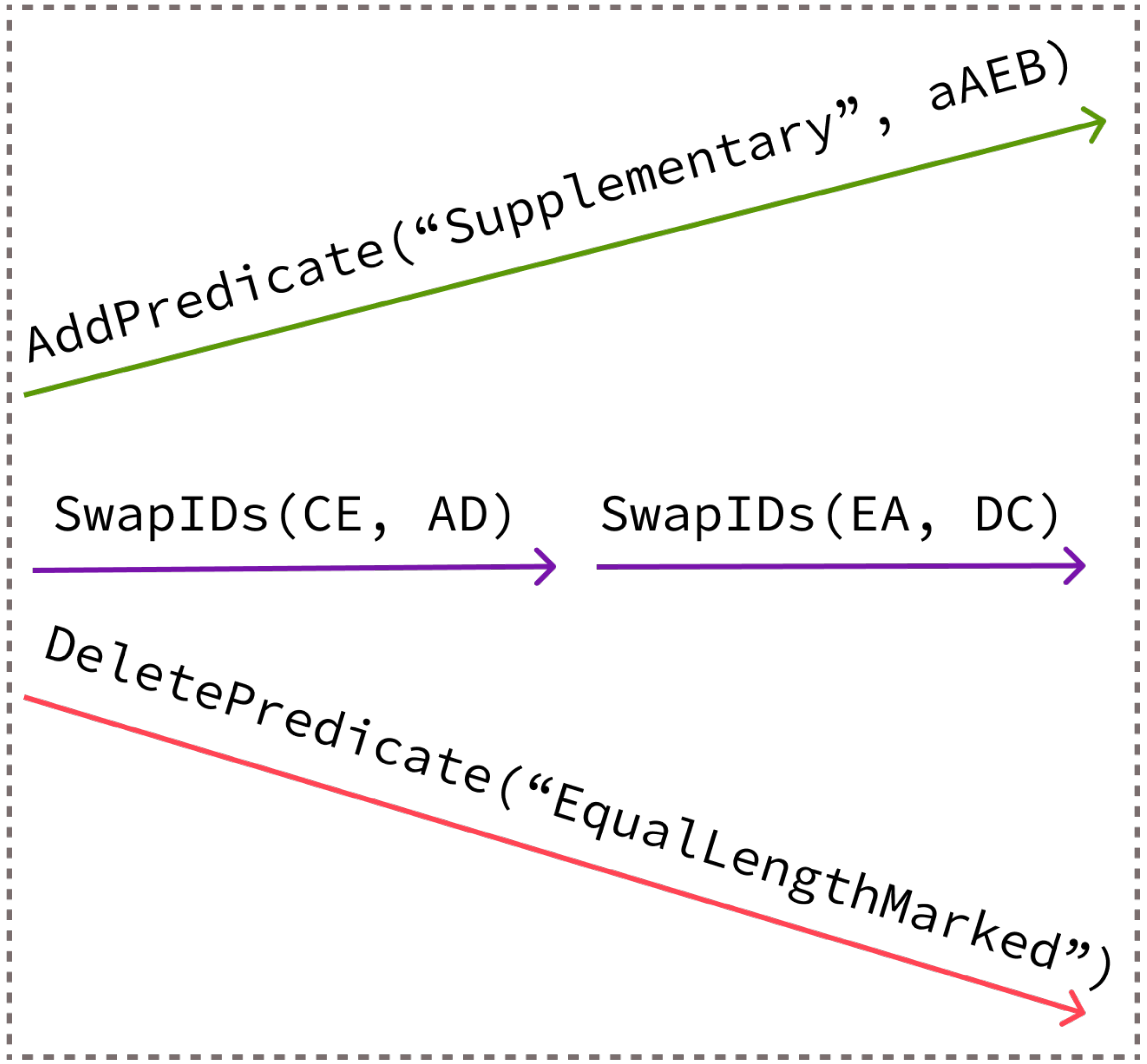
Mutation paths ➡️ templates

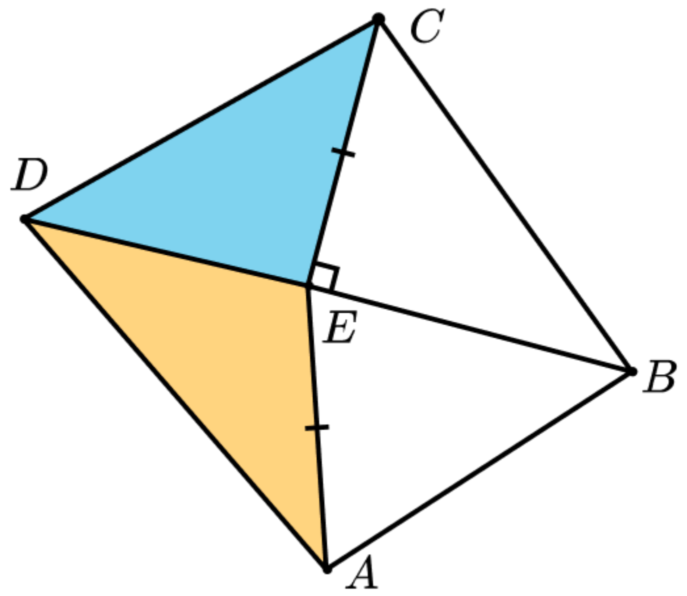
Proposed



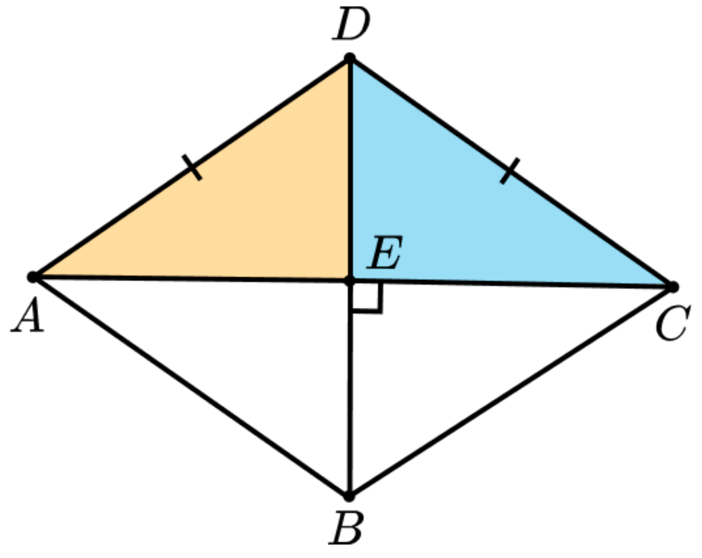
```
Point A, B, C, D, E
Triangle CED, DEA, CEA, BED
Collinear(D, E, B)
RightMarked(a_CEB)
EqualLengthMarked(CE, EA)
```

Template

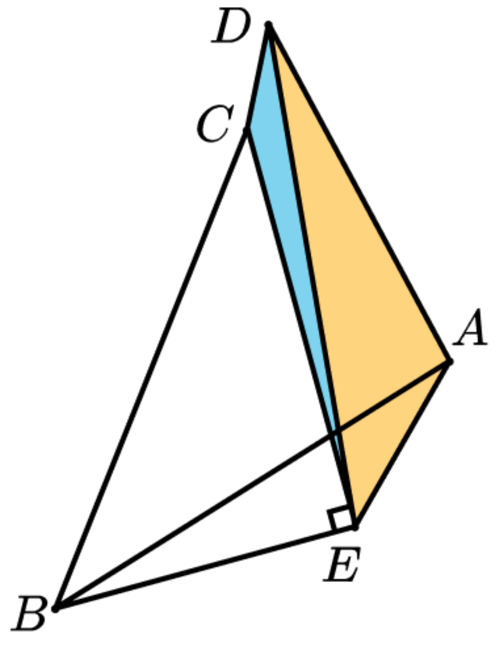




```
Point A, B, C, D, E
Triangle CED, DEA, CEA, BED
Collinear(D, E, B)
RightMarked(a_CEB)
EqualLengthMarked(CE, EA)
Supplementary(a_AEB)
```

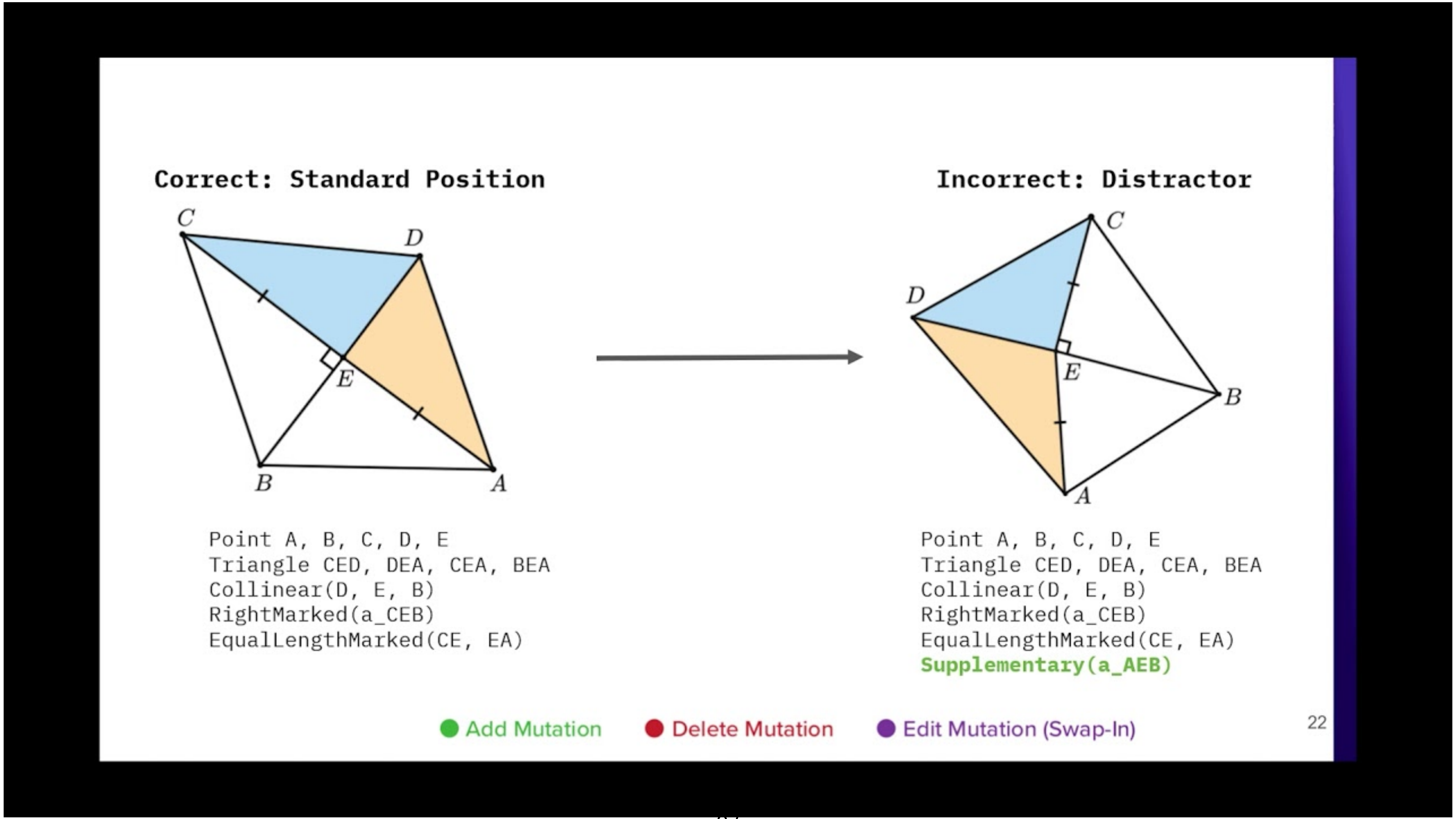


```
Point A, B, C, D, E
Triangle CED, DEA, CEA, BED
Collinear(D, E, B)
RightMarked(a_CEB)
EqualLengthMarked(AD, DC)
```



Point A, B, C, D, E
Triangle CED, DEA, CEA, BED
~~Collinear(D, E, B)~~
RightMarked(a_CEB)
~~EqualLengthMarked(CE, EA)~~

Edgeworth v0.0.1 in action



Diagrammers use…

Programming languages

Abstraction & Automation

Local control

Steep learning curve

High upfront cost

Direct Manipulation

Fast feedback

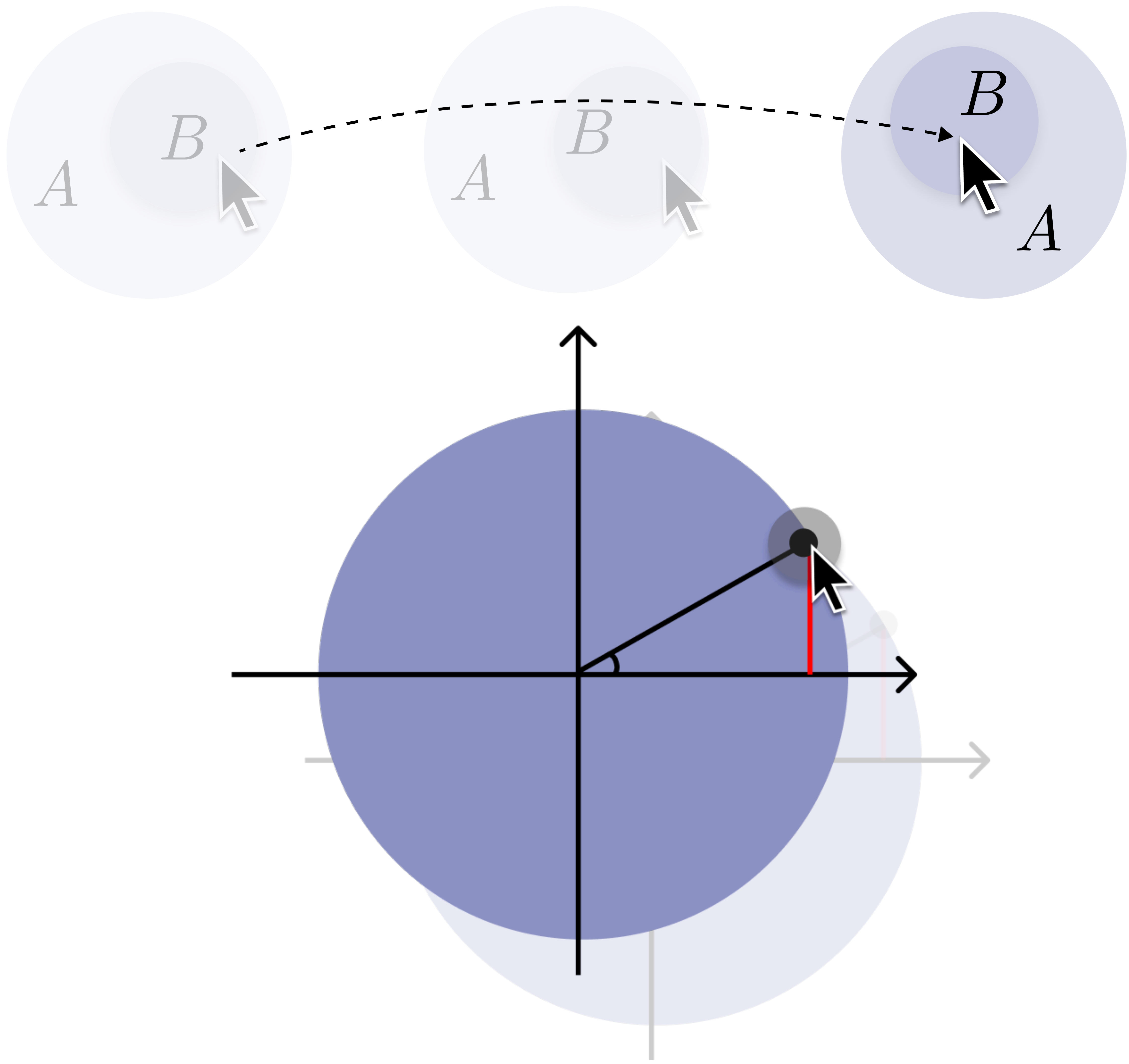
Global control

Very manual

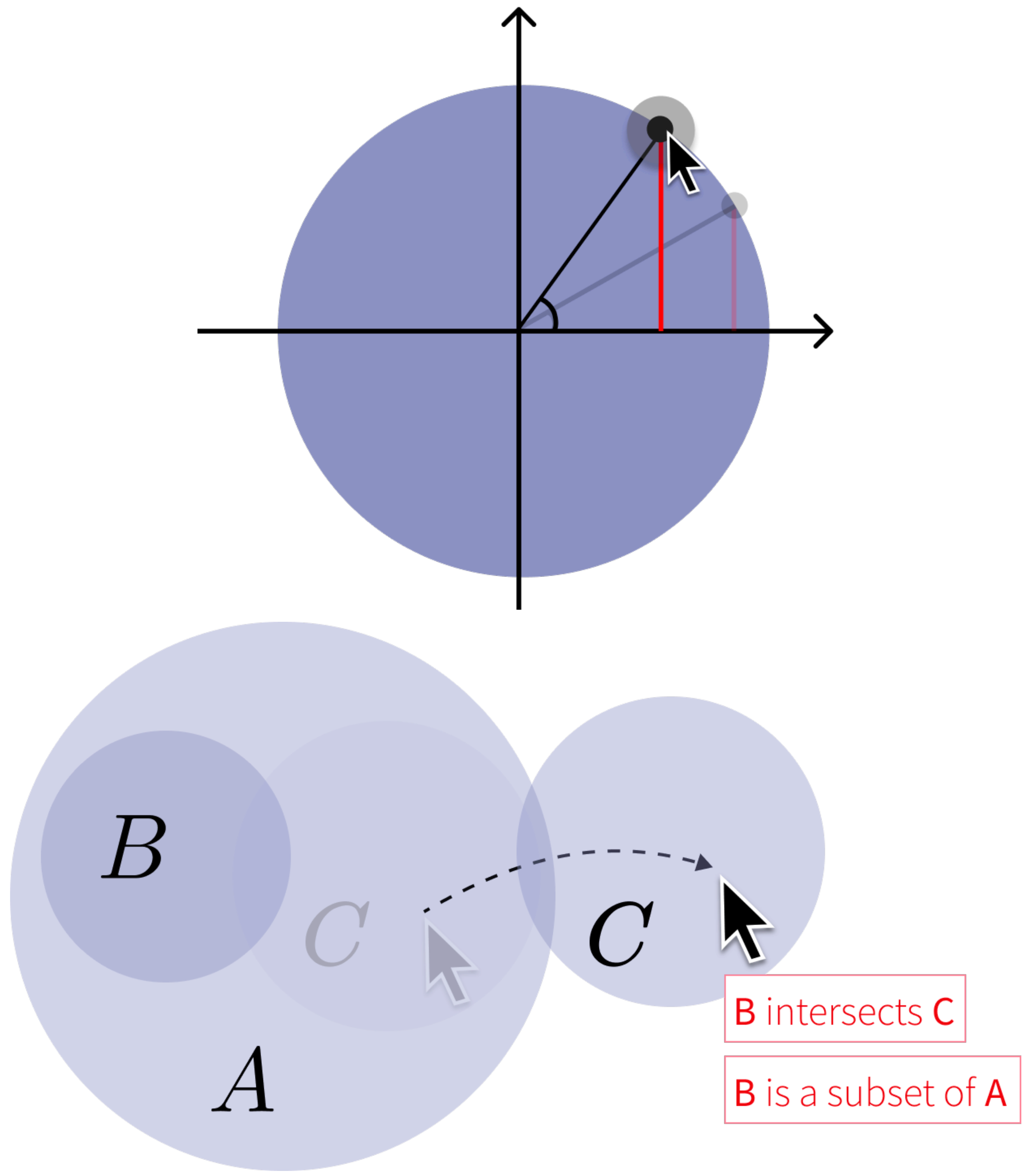
Viscosity

Two modes of drag feedback

Follow the cursor



Freeze the world

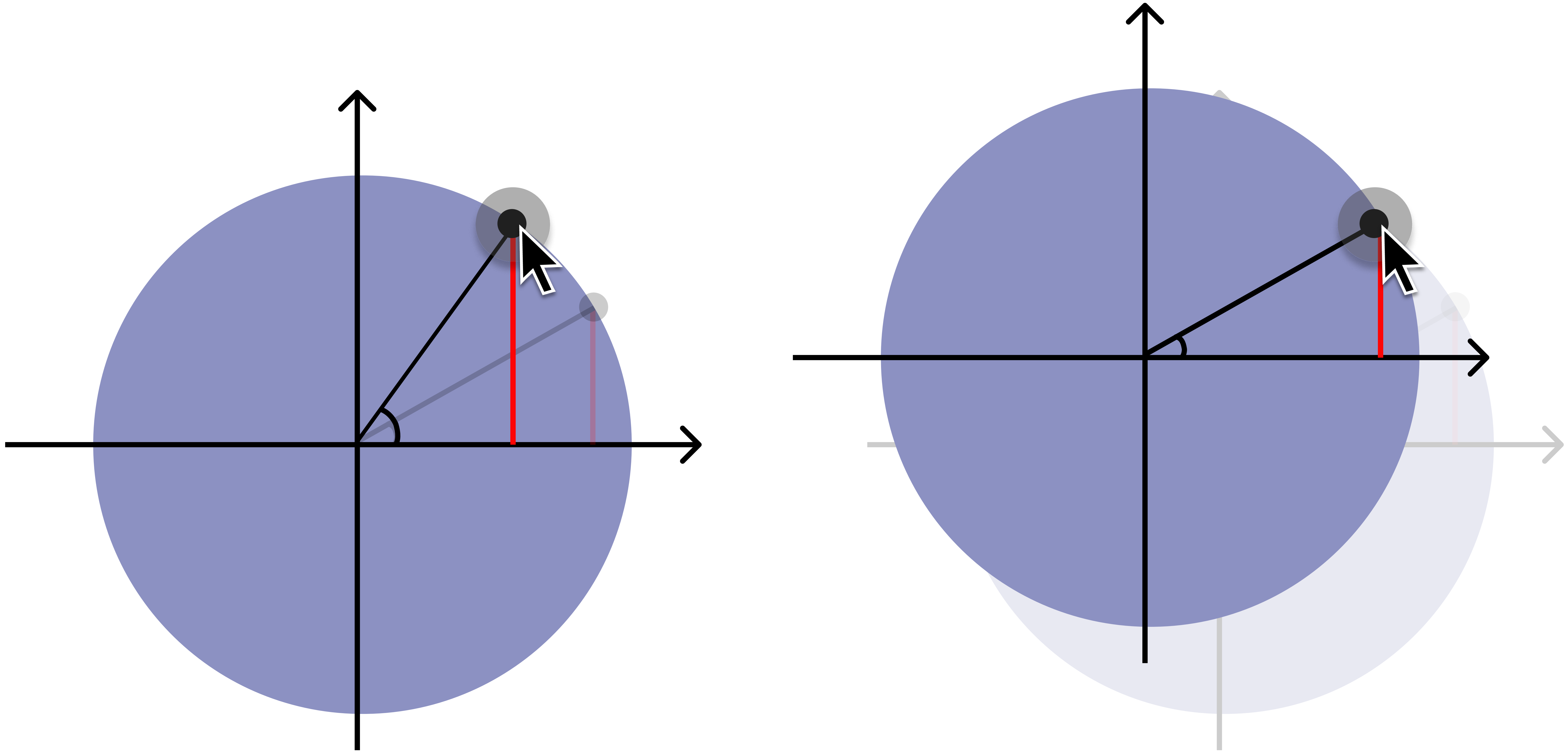


"Freeze the world"

"Follow the cursor" doesn't work universally.

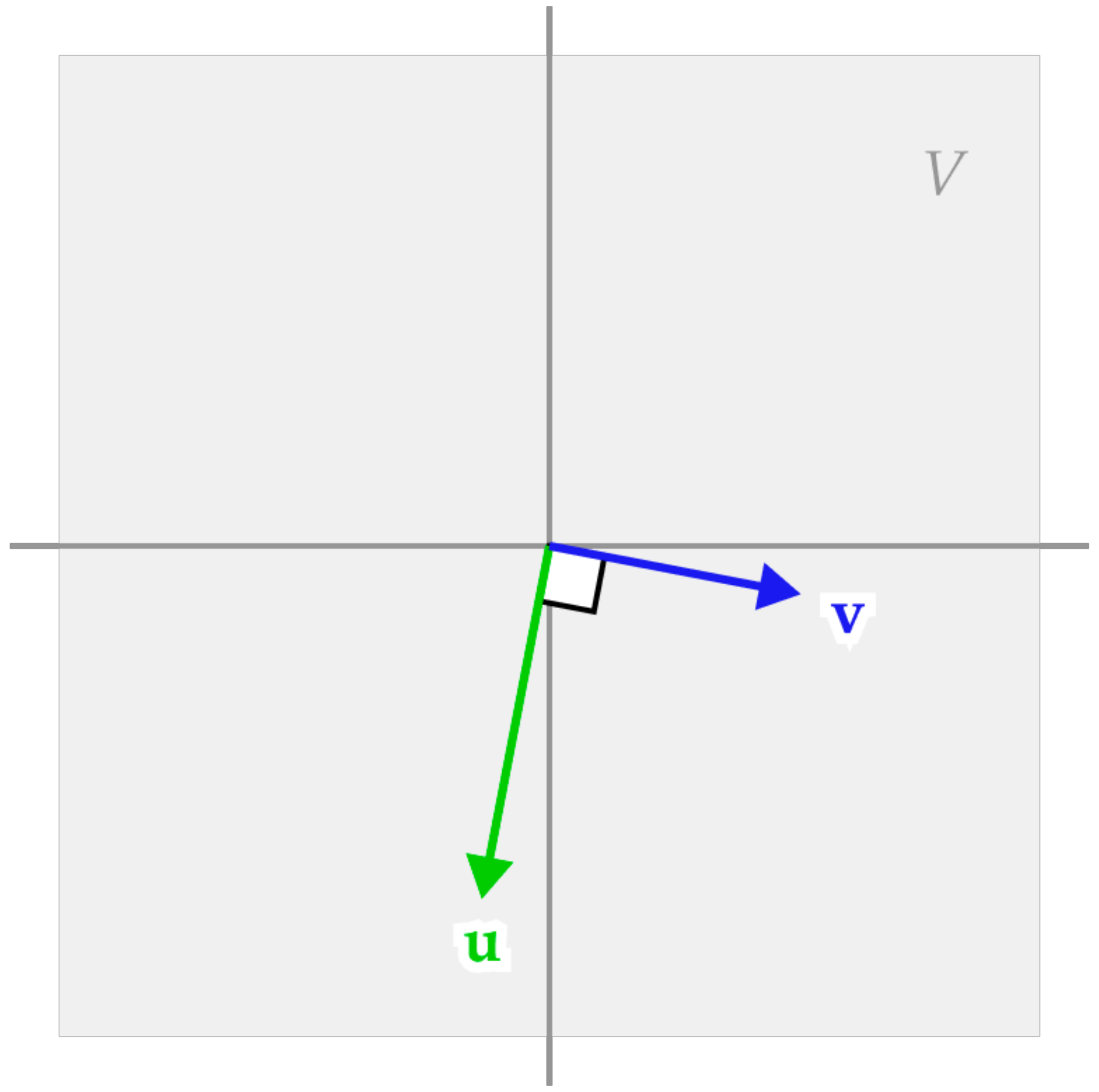
Look at optimization problem to deduce strategy

Approach: Lock all DOFs other than the interactive ones.

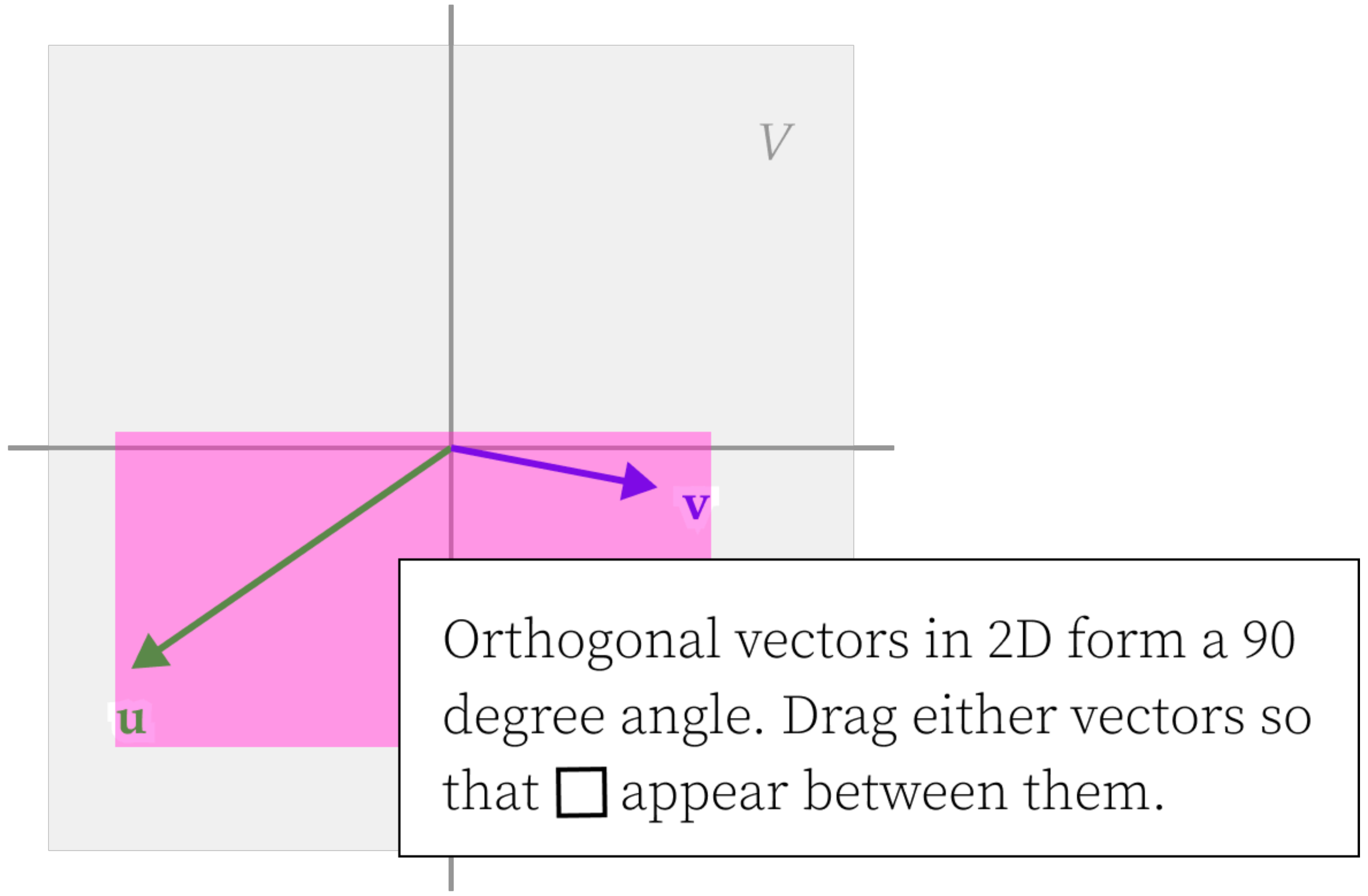


Dynamically update the diagram with decorations and feedback when constraints are violated.

```
forall Vector u; Vector v
with VectorSpace U
where Orthogonal(u, v); In(u, U); In(v, U) {
  perpMark = Path { ... }
  ortho = ensure equal(dot(u.vector, v.vector), 0.0)
  when ortho violated {
    highlight = bbox(u.shape, v.shape) {
       fillColor: Colors.transparantPink
    }
    tooltip = Tooltip {
      message: `Orthogonal vectors in 2D form a 90
degree angle. Drag either vectors so that ${perpMark}
appear between them.`
    }
  }
}
```



orthogonal



orthogonal