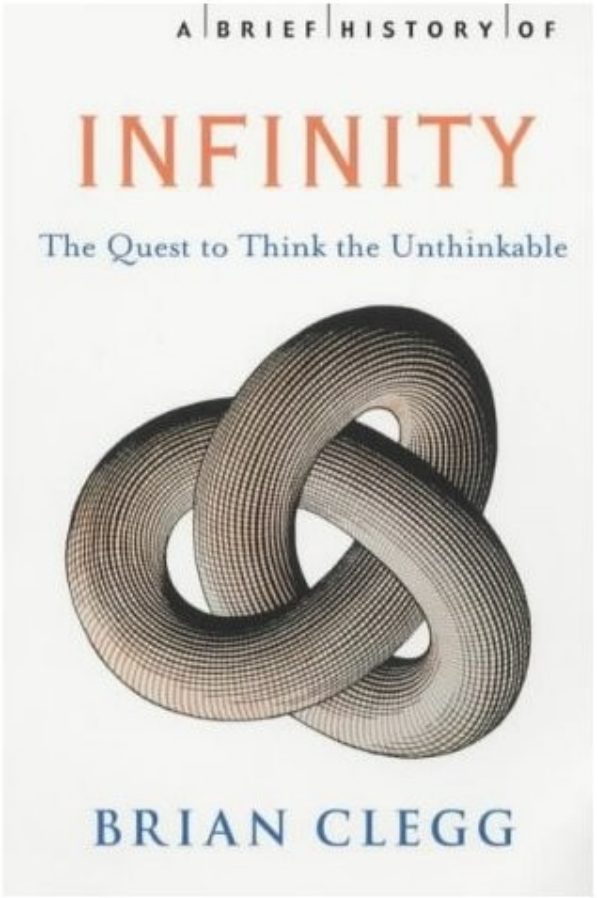
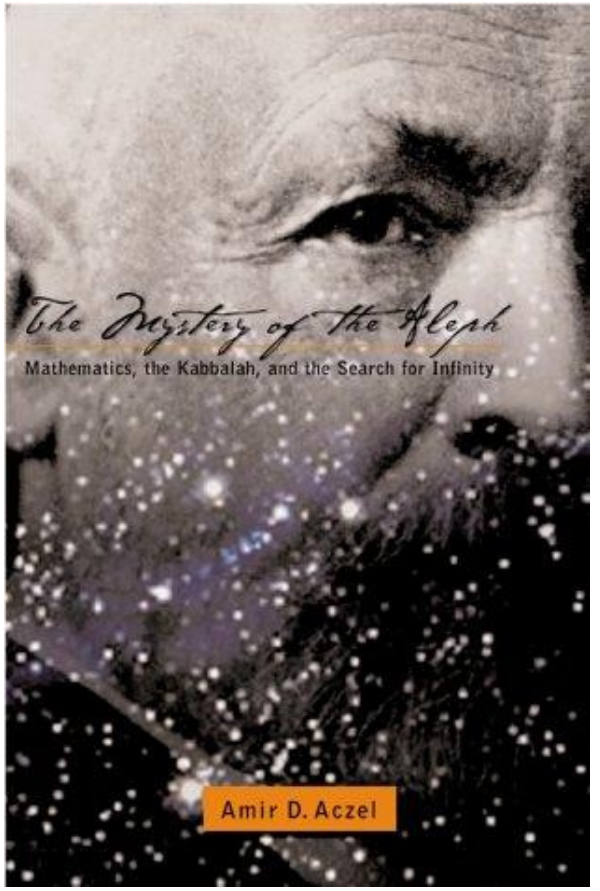
Graphs and Relations

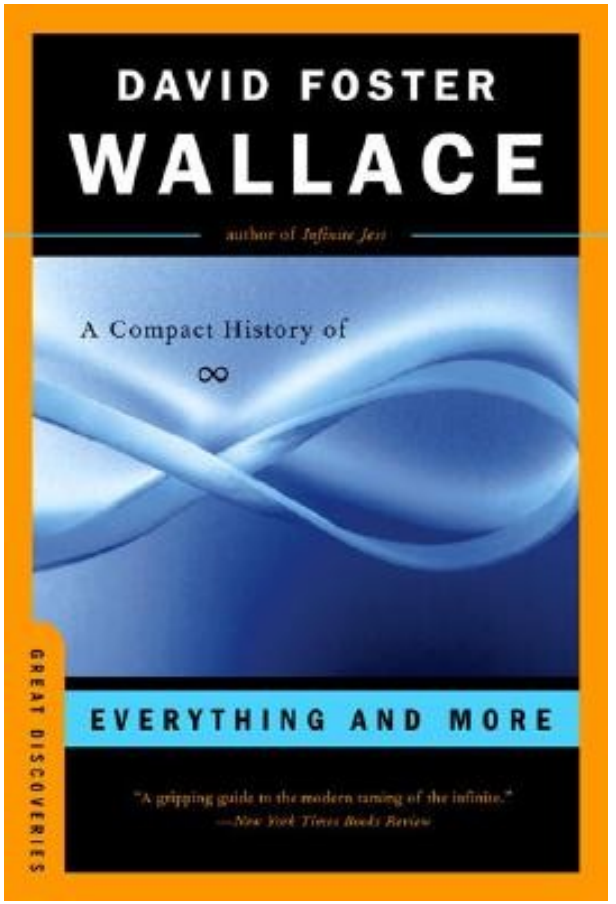
Recommended Reading



A Brief History of Infinity



The Mystery of the Aleph



Everything and More

Some Formalisms

Ordered and Unordered Pairs

- An **unordered pair** is a set {a, b} of two elements (remember that sets are unordered).
  - {0, 1} = {1, 0}
- An **ordered pair** (a, b) is a pair of elements in a specific order.
  - (0, 1) ≠ (1, 0)
- An **unordered tuple** is a set $\{a_0, a_1, \ldots, a_{n-1}\}$ of n elements.
- An **ordered tuple** $(a_0, a_1, \ldots, a_{n-1})$ is an collection of n elements in a specific order.
  - This is sometimes called a **sequence**.

The Cartesian Product

- Recall: The **power set** *P*(S) of a set is the set of all its subsets.
- The **Cartesian Product** of A × B of two sets is defined as

$$A \times B \equiv \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

The Cartesian Product

- Recall: The **power set** *P*(S) of a set is the set of all its subsets.
- The **Cartesian Product** of A × B of two sets is defined as

$$A \times B \equiv \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

$\{ 0, 1, 2 \}$ $\{ a, b, c \}$

A B

The Cartesian Product

- Recall: The **power set** *P*(S) of a set is the set of all its subsets.
- The **Cartesian Product** of A × B of two sets is defined as

$$A \times B \equiv \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

$$\{0, 1, 2\} \times \{a, b, c\} =$$

A B

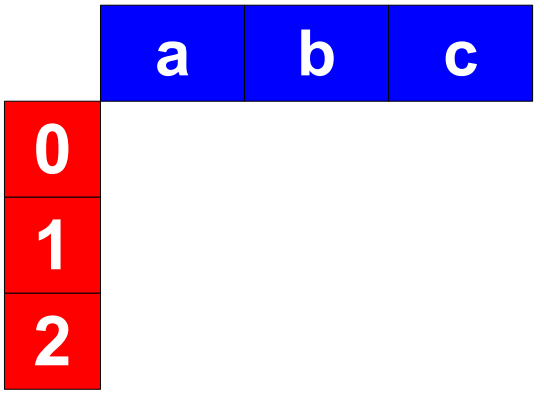
The Cartesian Product

- Recall: The **power set** *P*(S) of a set is the set of all its subsets.
- The **Cartesian Product** of A × B of two sets is defined as

$$A \times B \equiv \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

$$\{0, 1, 2\} \times \{a, b, c\} =$$

A B



The Cartesian Product

- Recall: The **power set** *P*(S) of a set is the set of all its subsets.
- The **Cartesian Product** of A × B of two sets is defined as

$$A \times B \equiv \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

$$\underbrace{\{0, 1, 2\}}_{A} \times \underbrace{\{a, b, c\}}_{B} =$$

	a	b	c
0	(0, a)	(0, b)	(0, c)
1	(1, a)	(1, b)	(1, c)
2	(2, a)	(2, b)	(2, c)

The Cartesian Product

- Recall: The **power set** *P*(S) of a set is the set of all its subsets.
- The **Cartesian Product** of A × B of two sets is defined as

$$A \times B \equiv \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

$$\underbrace{\{0, 1, 2\}}_{A} \times \underbrace{\{a, b, c\}}_{B} = \left\{ \begin{array}{l} (0, a), (0, b), (0, c), \\ (1, a), (1, b), (1, c), \\ (2, a), (2, b), (2, c) \end{array} \right\}$$

The Cartesian Product

- Recall: The **power set** *P*(S) of a set is the set of all its subsets.
- The **Cartesian Product** of A × B of two sets is defined as

$$A \times B \equiv \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

- We denote $A^k \equiv \underbrace{A \times A \times \ldots \times A}_{k \text{ times}}$

$$\underbrace{\{0, 1, 2\}}_{A} \times \underbrace{\{a, b, c\}}_{B} = \left\{ \begin{array}{l} (0, a), (0, b), (0, c), \\ (1, a), (1, b), (1, c), \\ (2, a), (2, b), (2, c) \end{array} \right\}$$

The Cartesian Product

- Recall: The **power set** $P(S)$ of a set is the set of all its subsets.
- The **Cartesian Product** of $A \times B$ of two sets is defined as

$$A \times B \equiv \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

- We denote $A^k \equiv \underbrace{A \times A \times \ldots \times A}_{k \text{ times}}$

$$\underbrace{\{0, 1, 2\}}_{A} \times \underbrace{\{0, 1, 2\}}_{A} = \{ (0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2) \}$$

The Cartesian Product

- Recall: The **power set** $P(S)$ of a set is the set of all its subsets.
- The **Cartesian Product** of $A \times B$ of two sets is defined as

$$A \times B \equiv \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

- We denote $A^k \equiv \underbrace{A \times A \times \ldots \times A}_{k \text{ times}}$

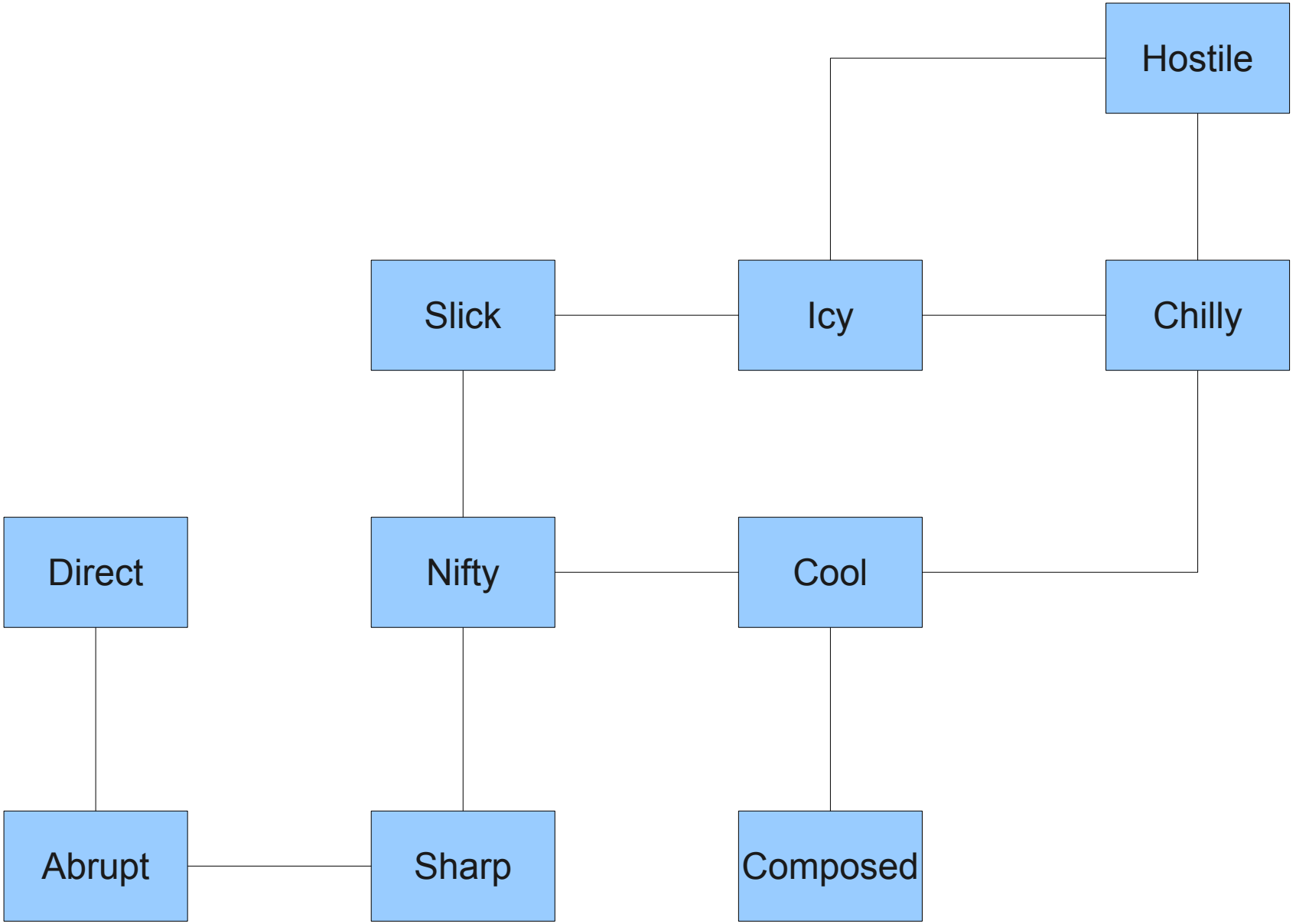
$$\underbrace{\{0, 1, 2\}}_{A}^{2} = \left\{ \begin{array}{l} (0, 0), (0, 1), (0, 2), \\ (1, 0), (1, 1), (1, 2), \\ (2, 0), (2, 1), (2, 2) \end{array} \right\}$$

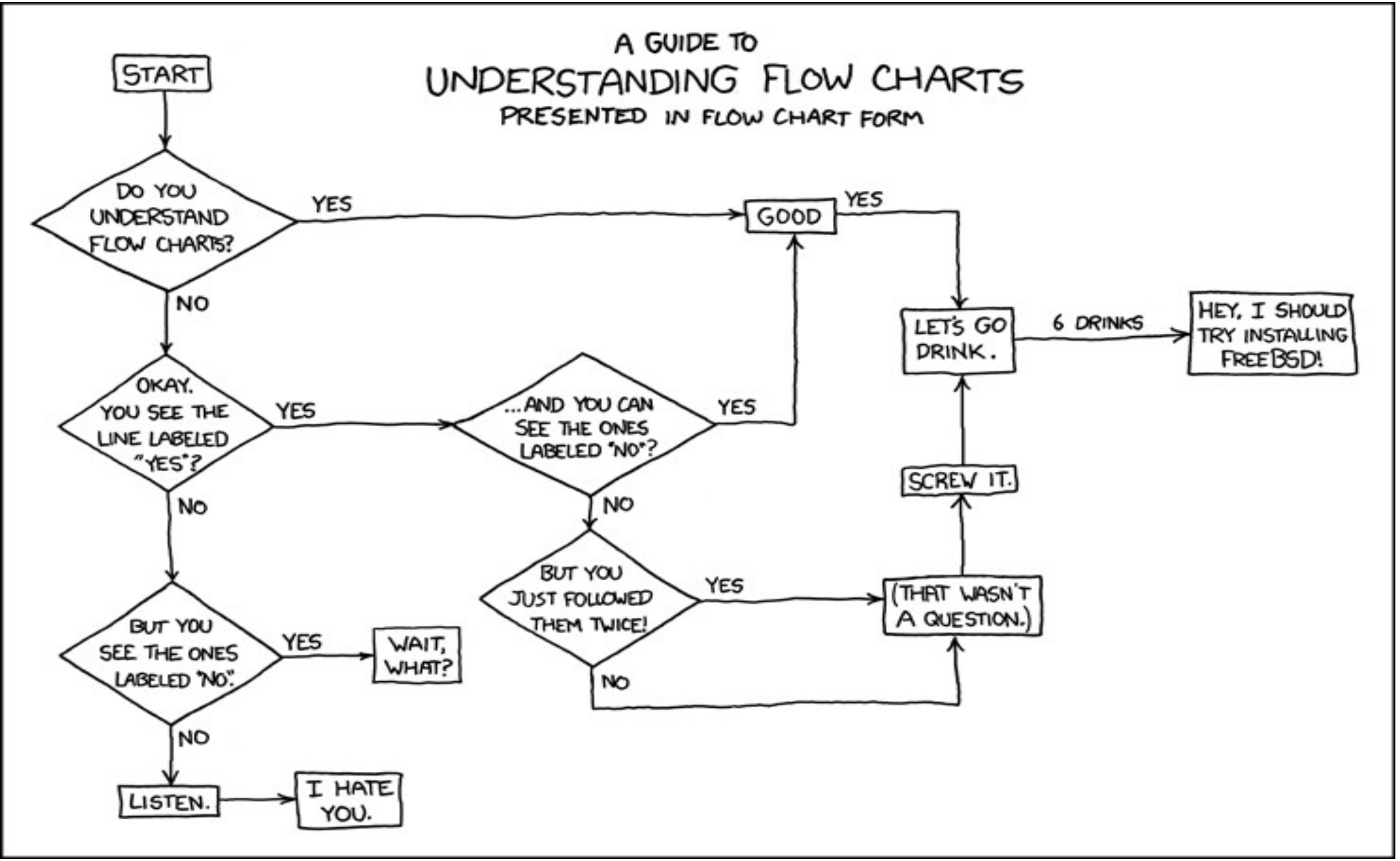
Graphs

A **graph** is a mathematical structure
for representing **relationships**.

Each graph is a set of **vertices** connected by **edges**.

Synonym Graph




A GUIDE TO
UNDERSTANDING FLOW CHARTS
PRESENTED IN FLOW CHART FORM
START
DO YOU UNDERSTAND FLOW CHARTS?
YES
GOOD
YES
NO
LET'S GO DRINK.
6 DRINKS
HEY, I SHOULD TRY INSTALLING FREEBSD!
OKAY. YOU SEE THE LINE LABELED "YES"?
YES
...AND YOU CAN SEE THE ONES LABELED "NO"?
YES
NO
SCREW IT.
NO
BUT YOU JUST FOLLOWED THEM TWICE!
YES
(THAT WASN'T A QUESTION.)
BUT YOU SEE THE ONES LABELED "NO".
YES
WAIT, WHAT?
NO
NO
LISTEN.
I HATE YOU.

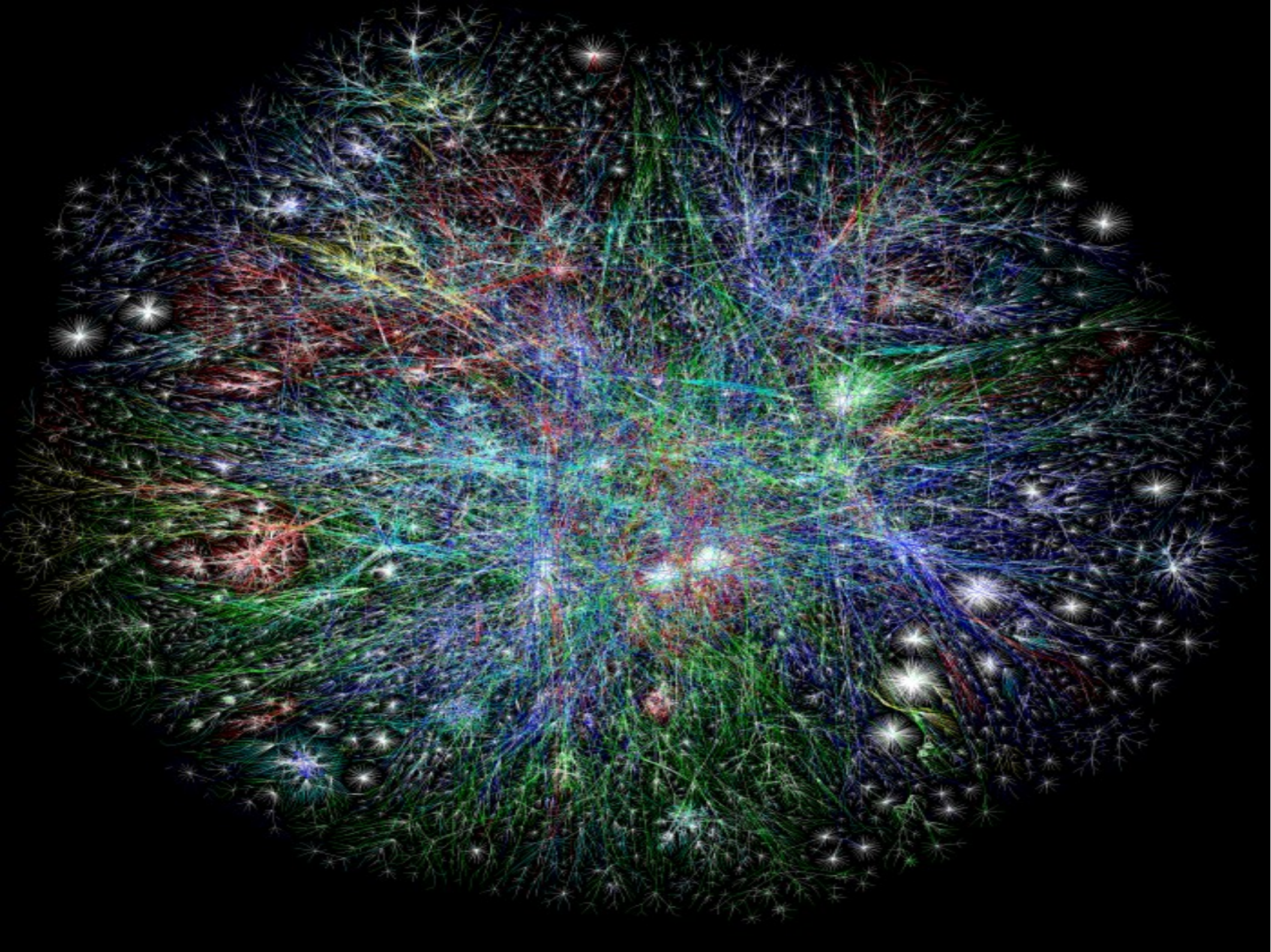
Source: xkcd

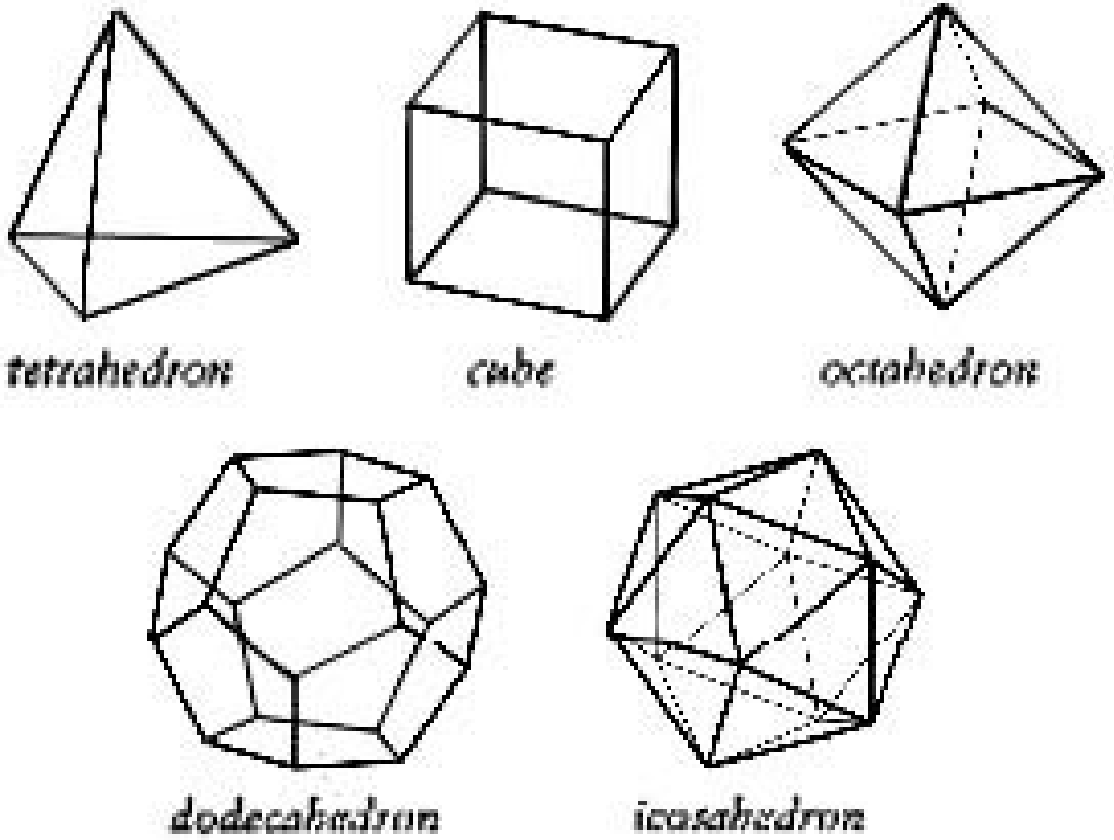

facebook®


facebook


Me too!

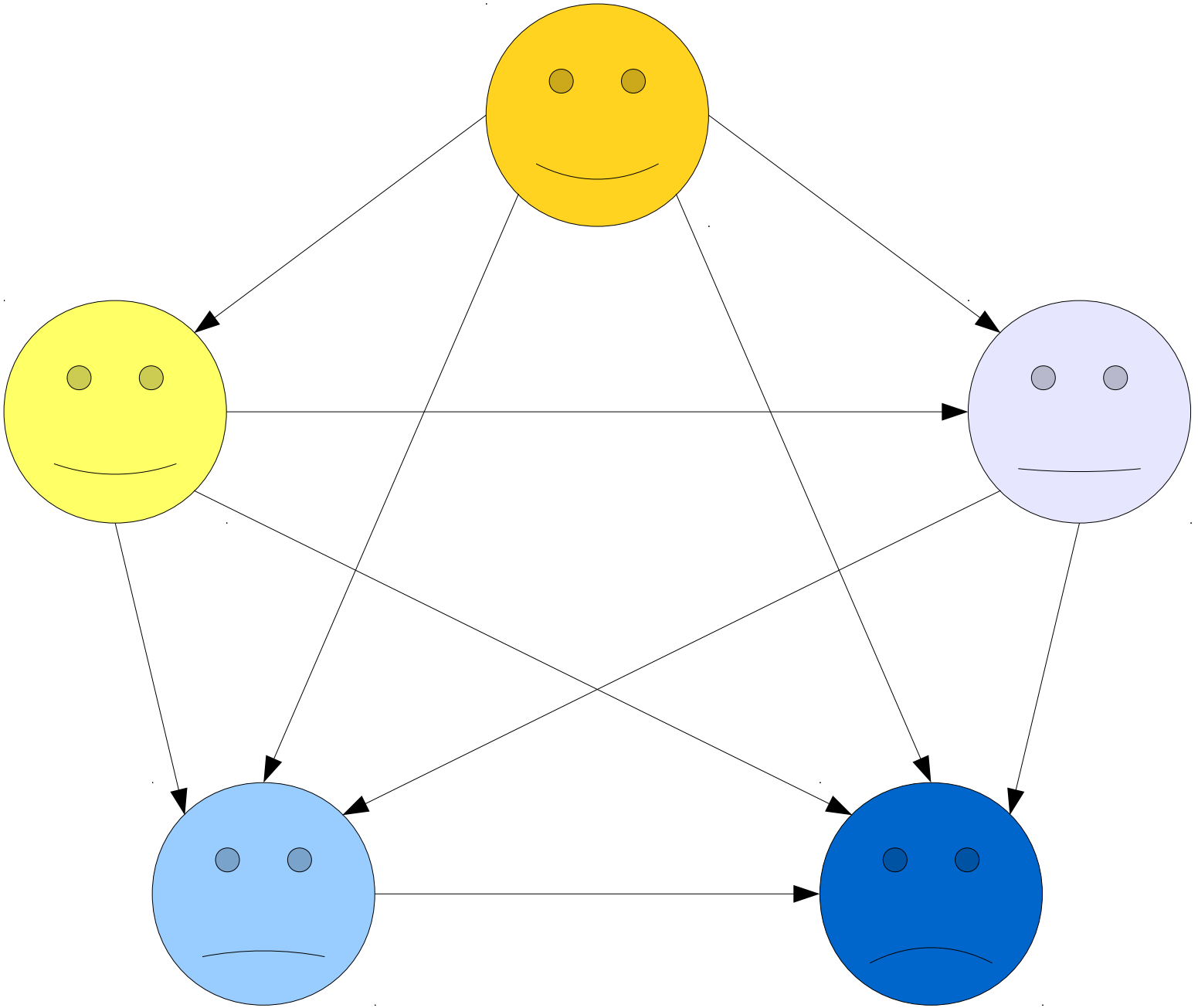

g+

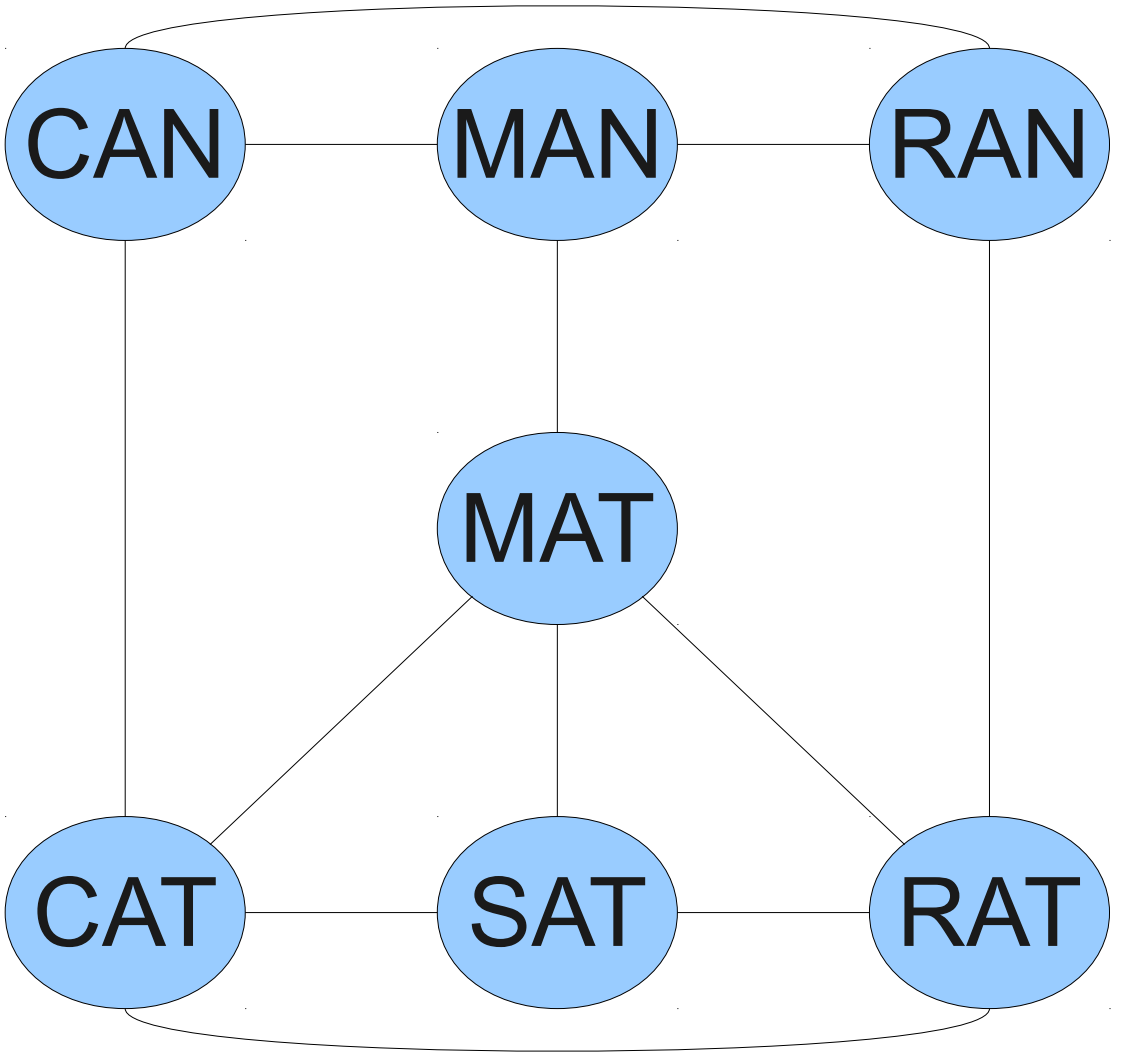



tetrahedron
cube
octahedron
dodecahedron
icosahedron

Formalisms

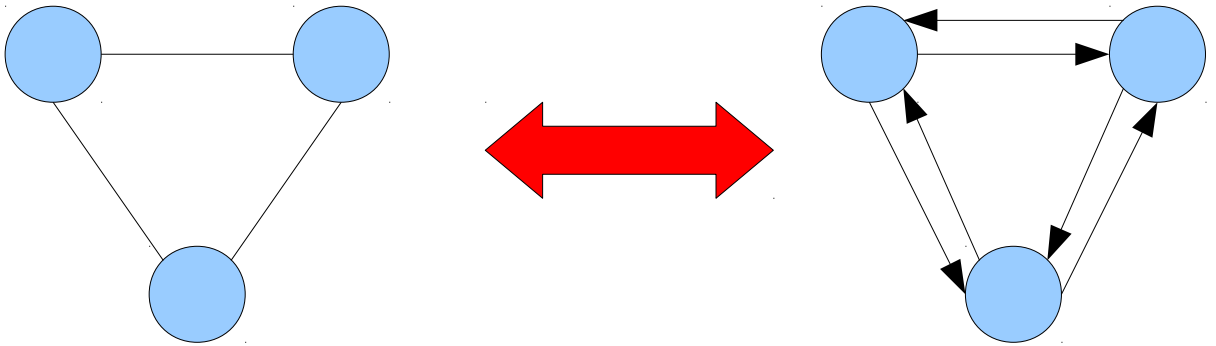
- A **graph** is an ordered pair G = (V, E) where
  - V is a set of the **vertices** (nodes) of the graph.
  - E is a set of the **edges** (arcs) of the graph.
- Each edge is an pair of the **start** and **end** (or **source** and **sink**) of the edge.



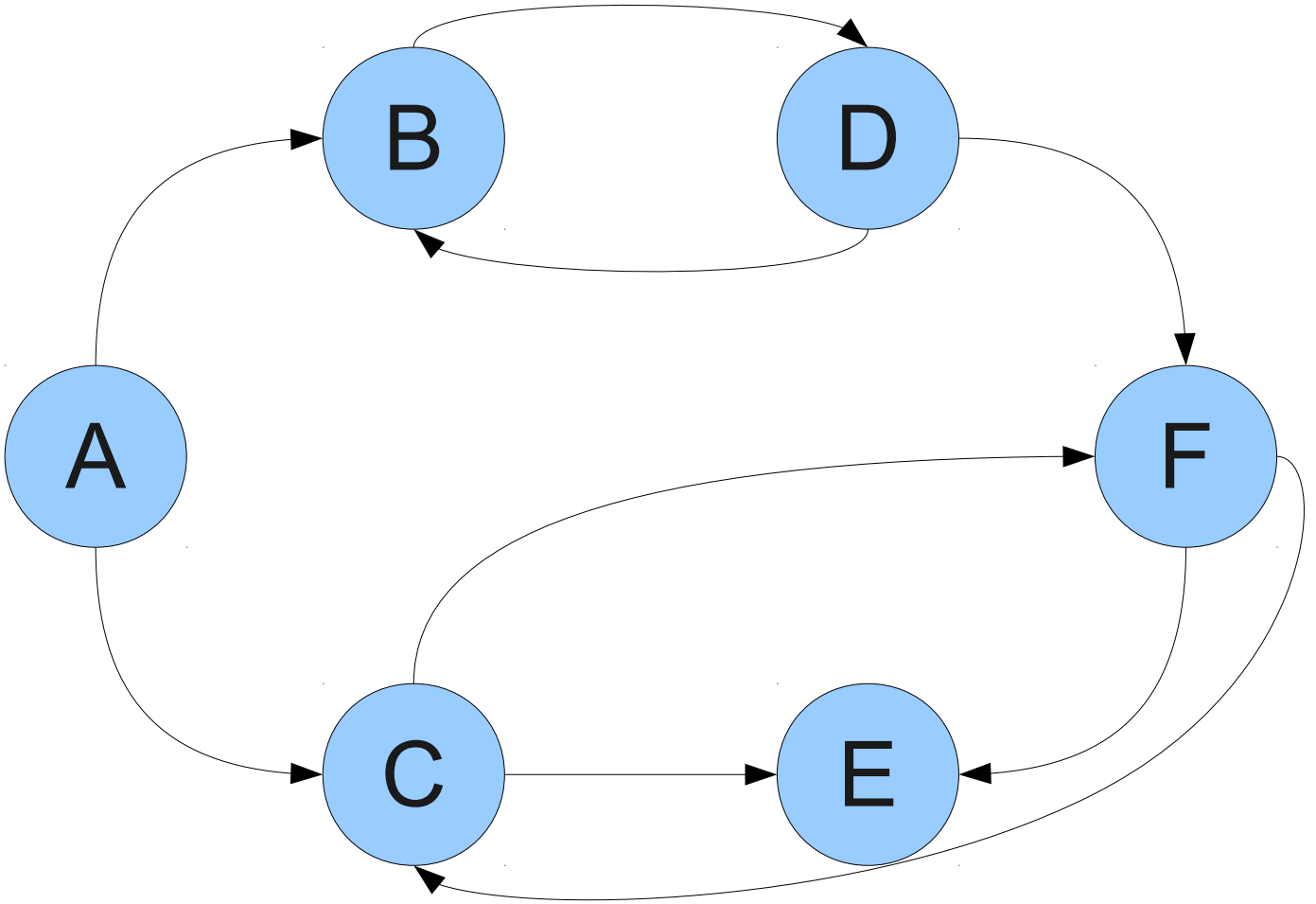

CAN
MAN
RAN
MAT
CAT
SAT
RAT

Directed and Undirected Graphs

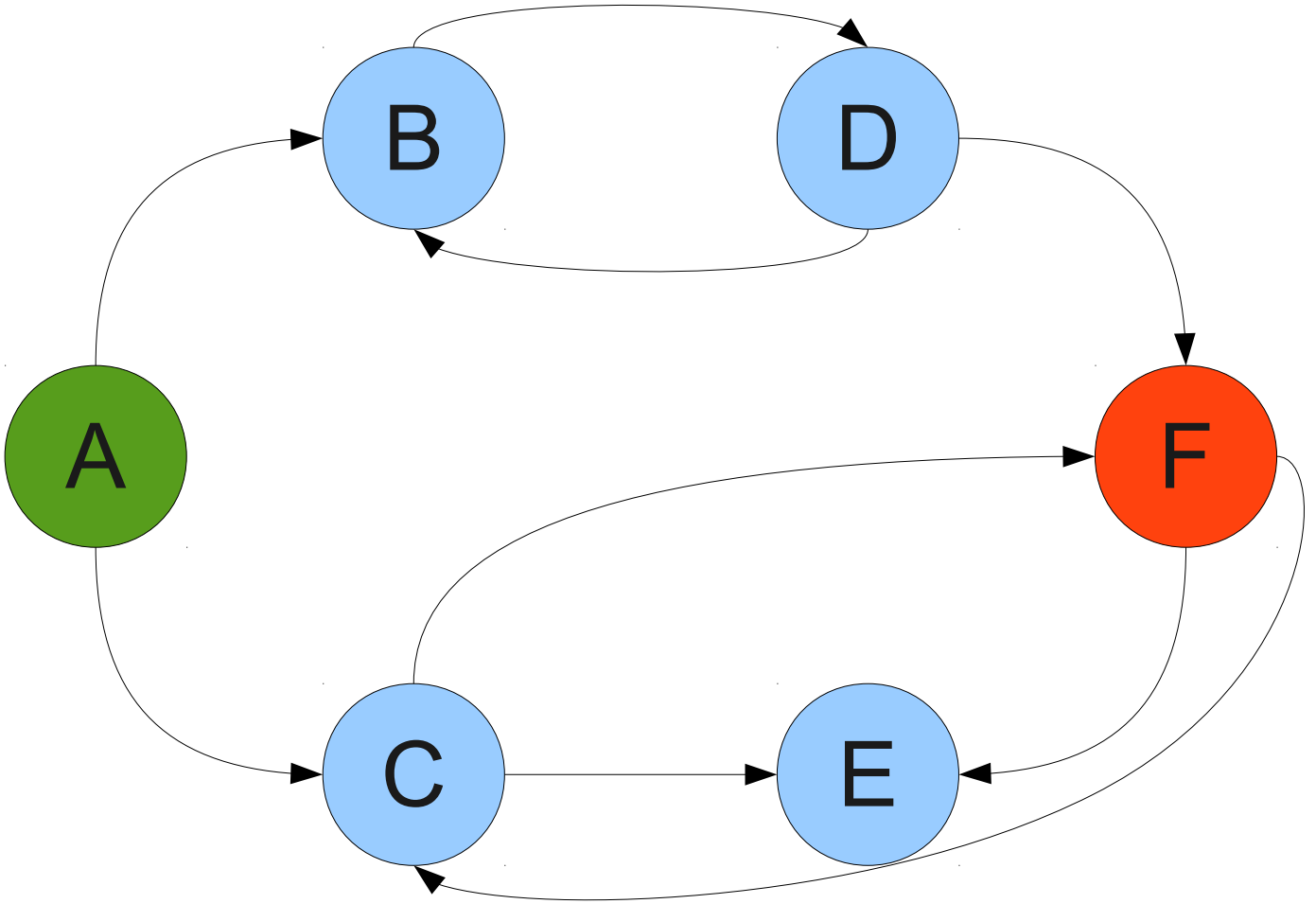
- A graph is **directed** if its edges are ordered pairs.
- A graph is **undirected** if the edges are unordered pairs.
- An undirected graph is a **special case** of a directed graph (just add edges both ways).



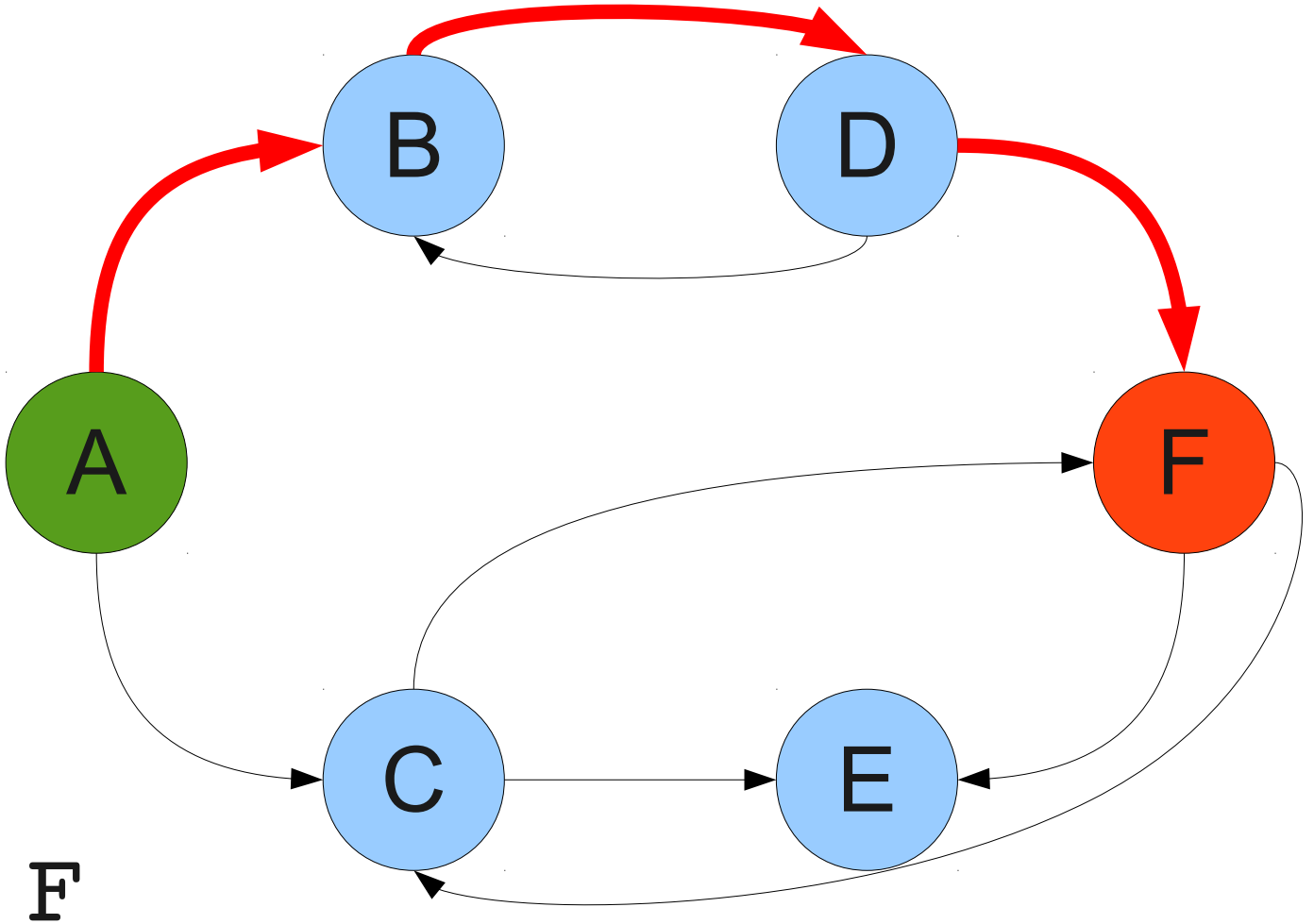
Navigating a Graph



Navigating a Graph

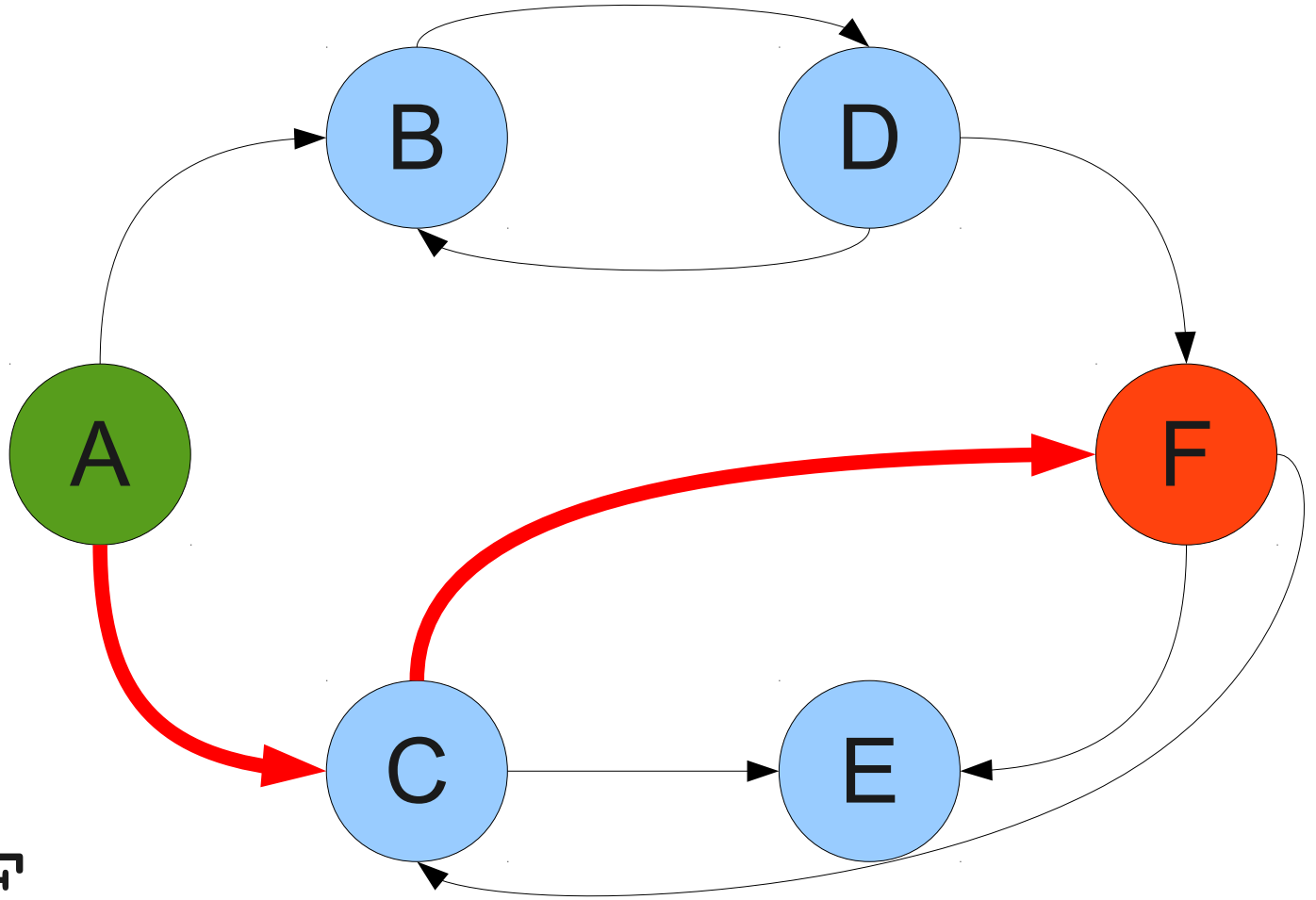


Navigating a Graph



A B D F

Navigating a Graph

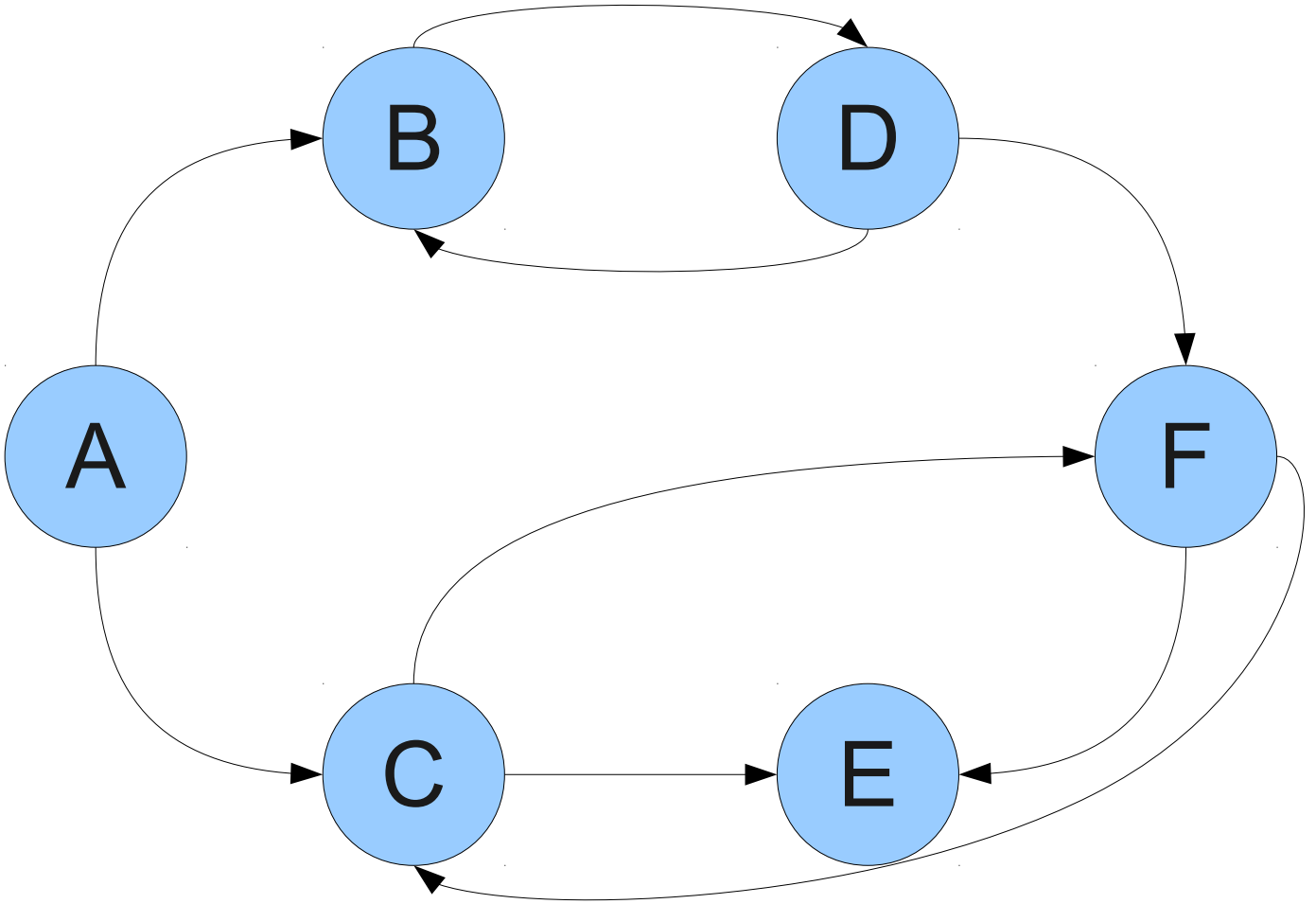


A C F

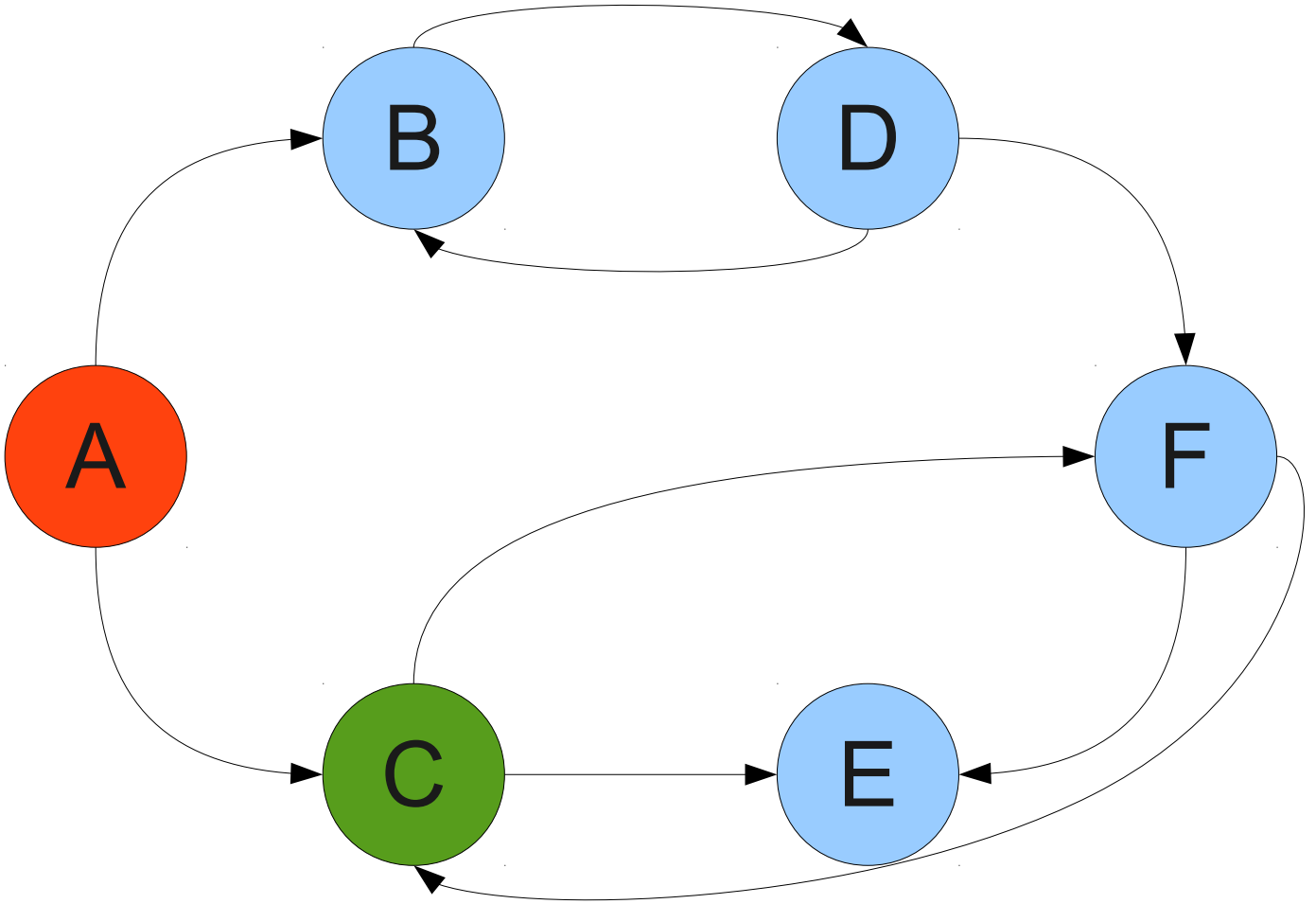
A **path** from v_0 to v_n is a sequence of edges $((v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n))$.

The **length** of a path is the number of edges it contains.

Navigating a Graph

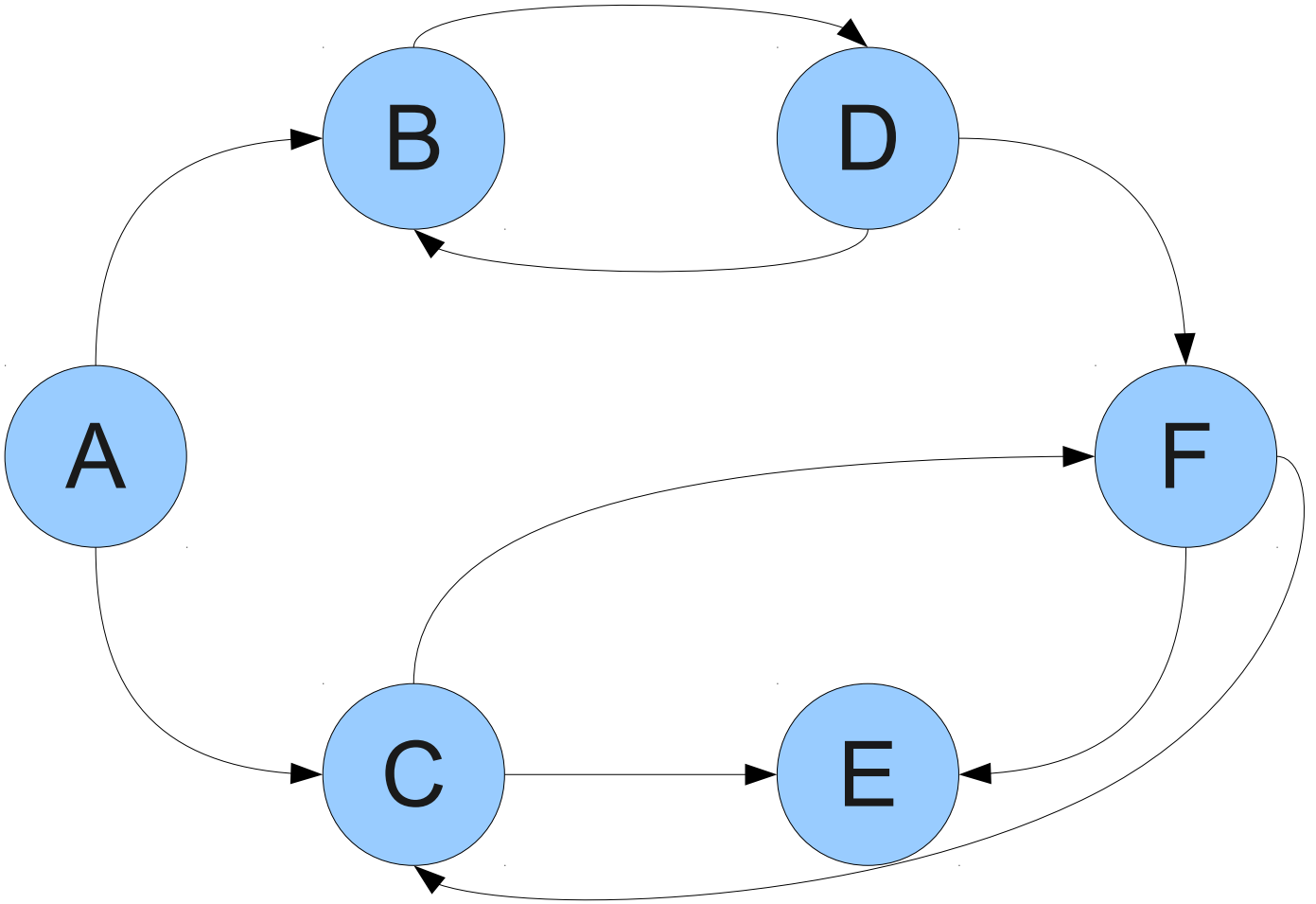


Navigating a Graph

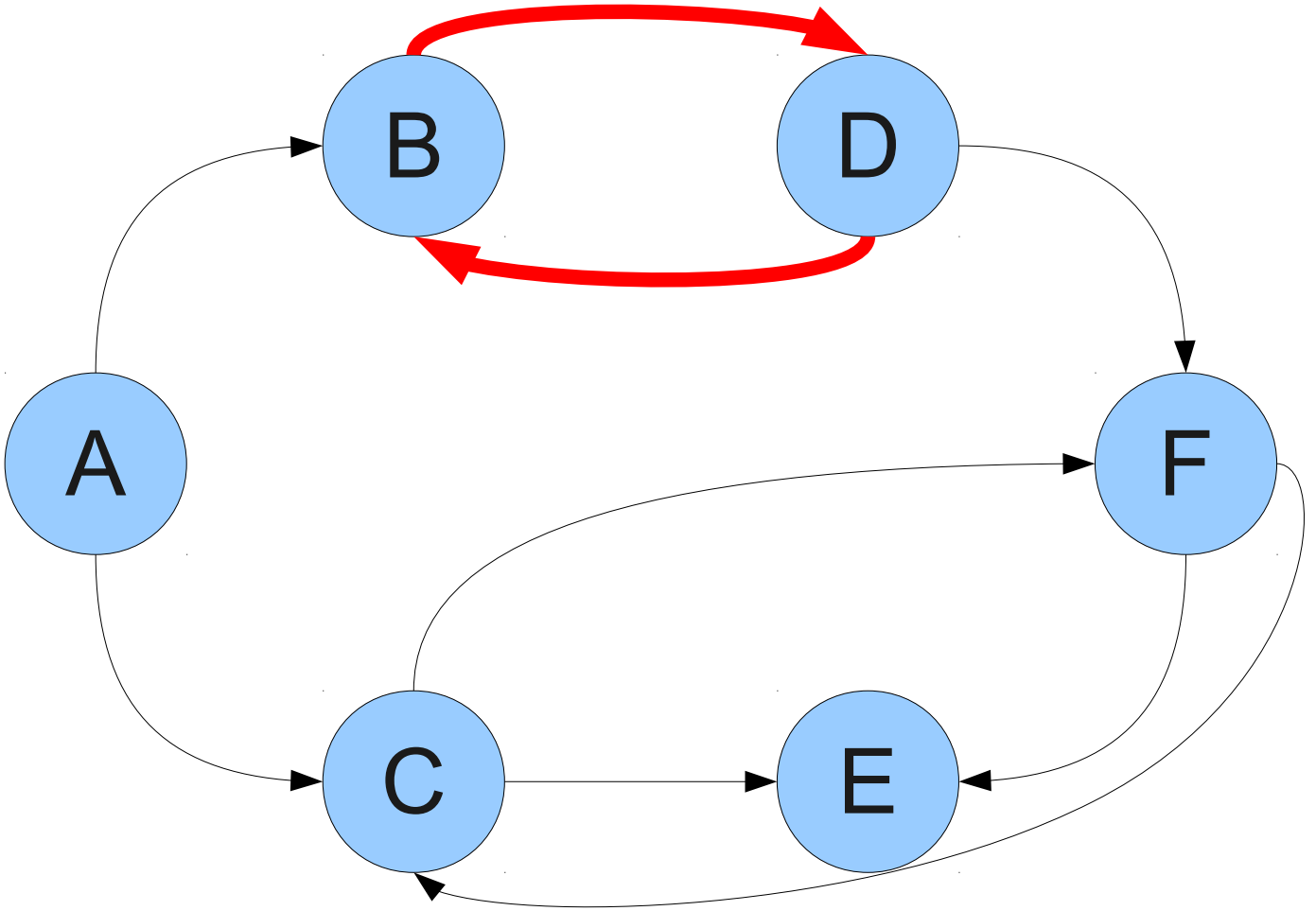


A node v is **reachable** from node u
if there is a path from u to v.

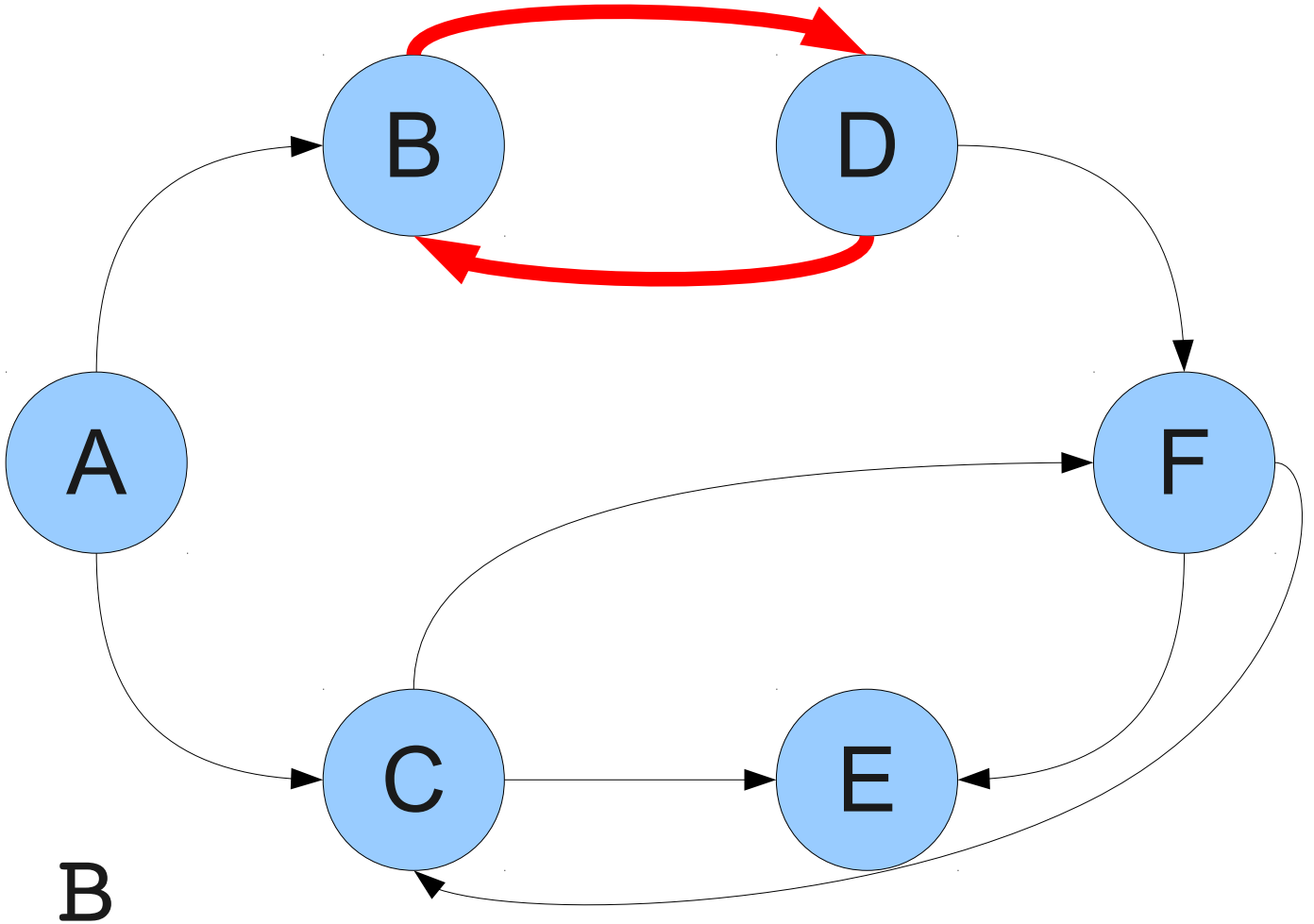
Navigating a Graph



Navigating a Graph

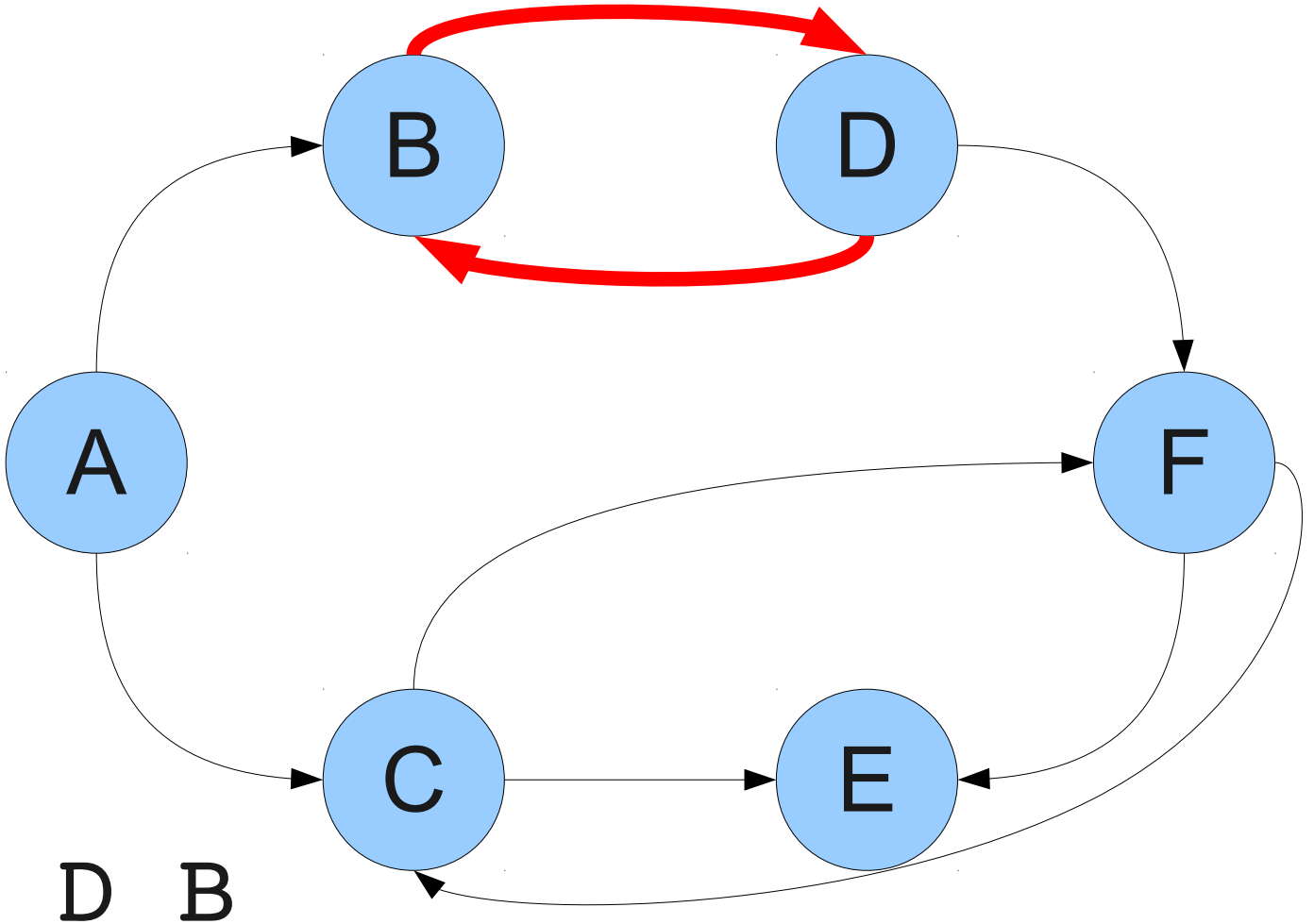


Navigating a Graph



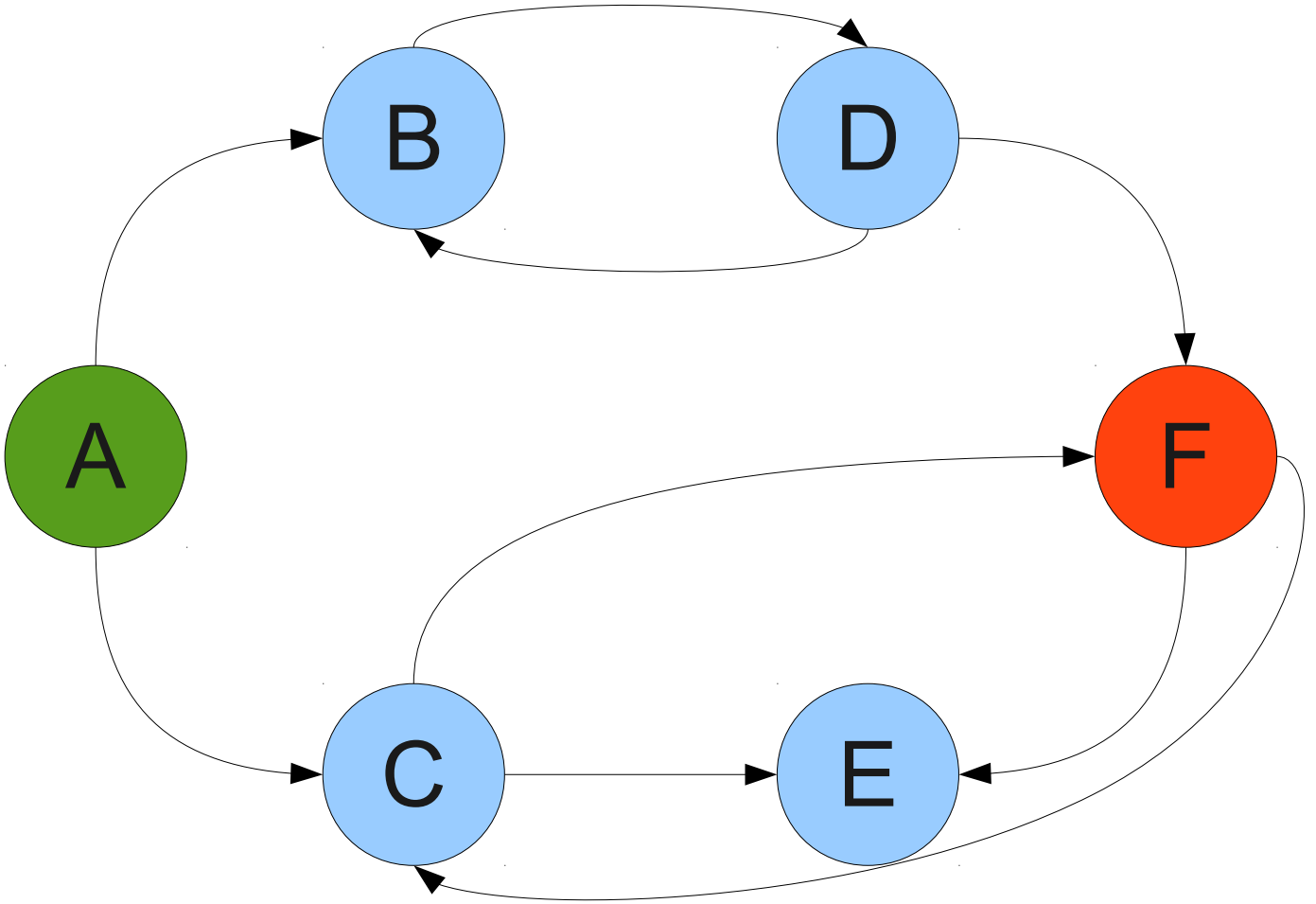
B D B

Navigating a Graph

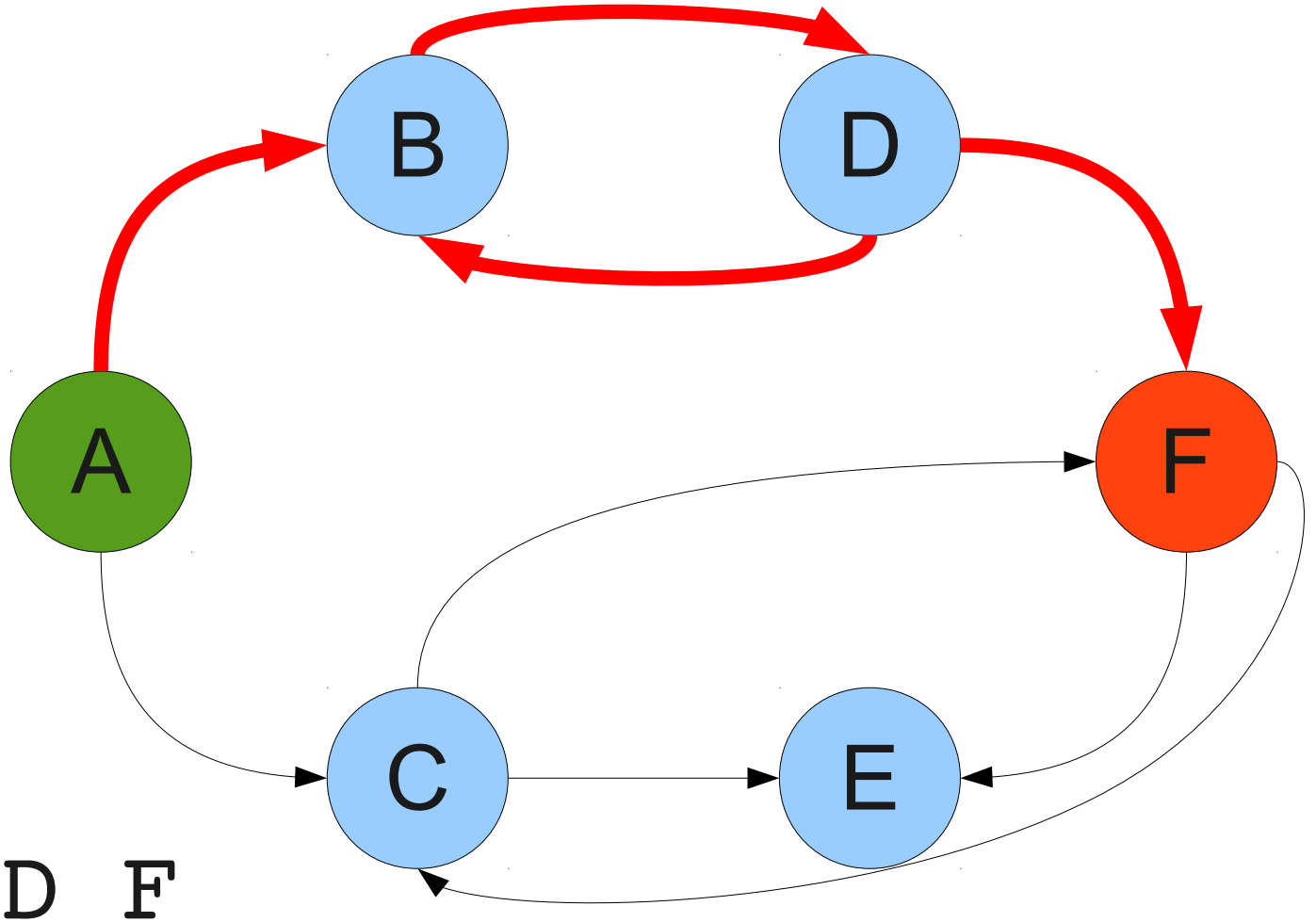


B D B D B

Navigating a Graph

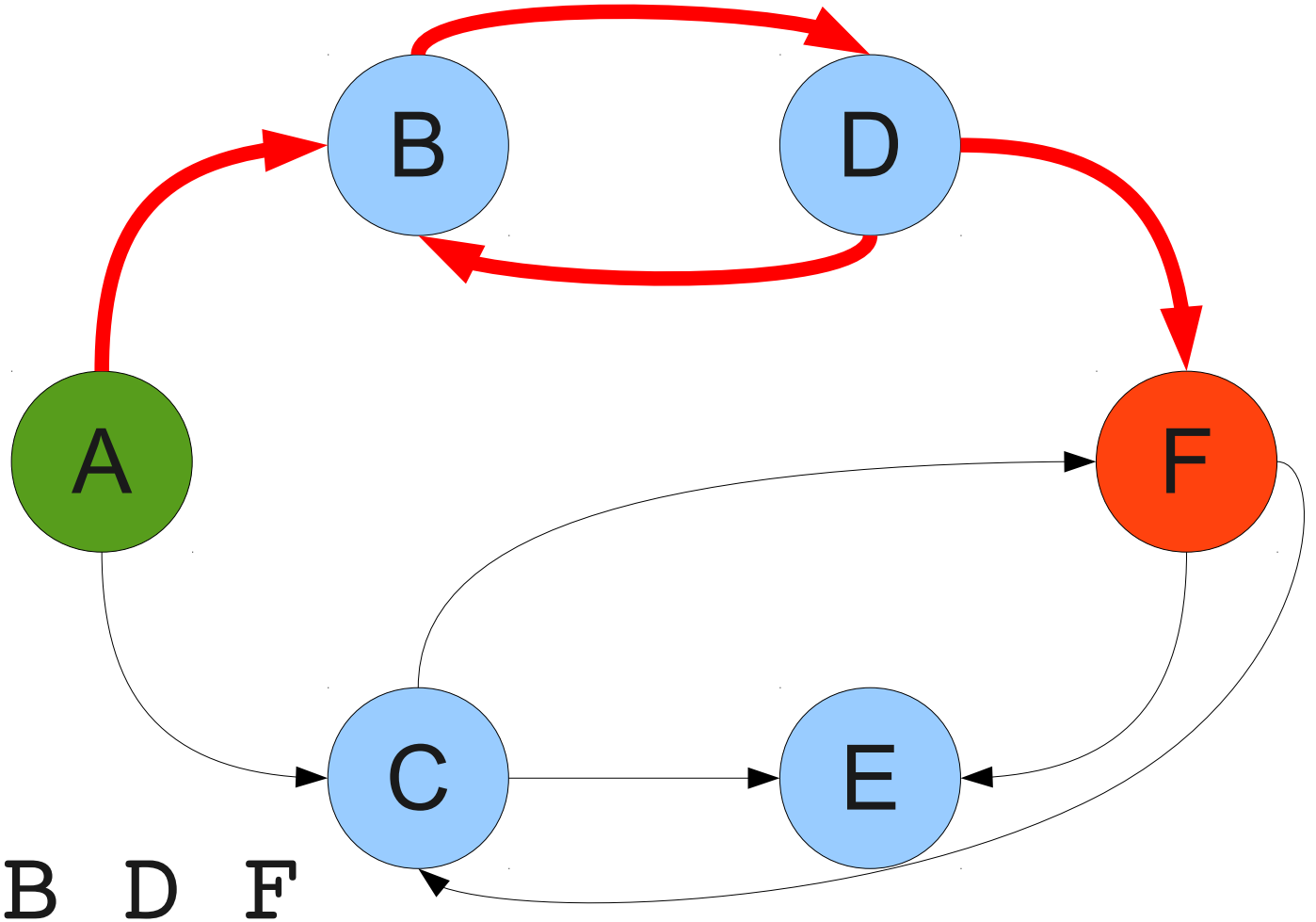


Navigating a Graph



A B D F

Navigating a Graph



A B D B D F

A **cycle** in a graph is a path

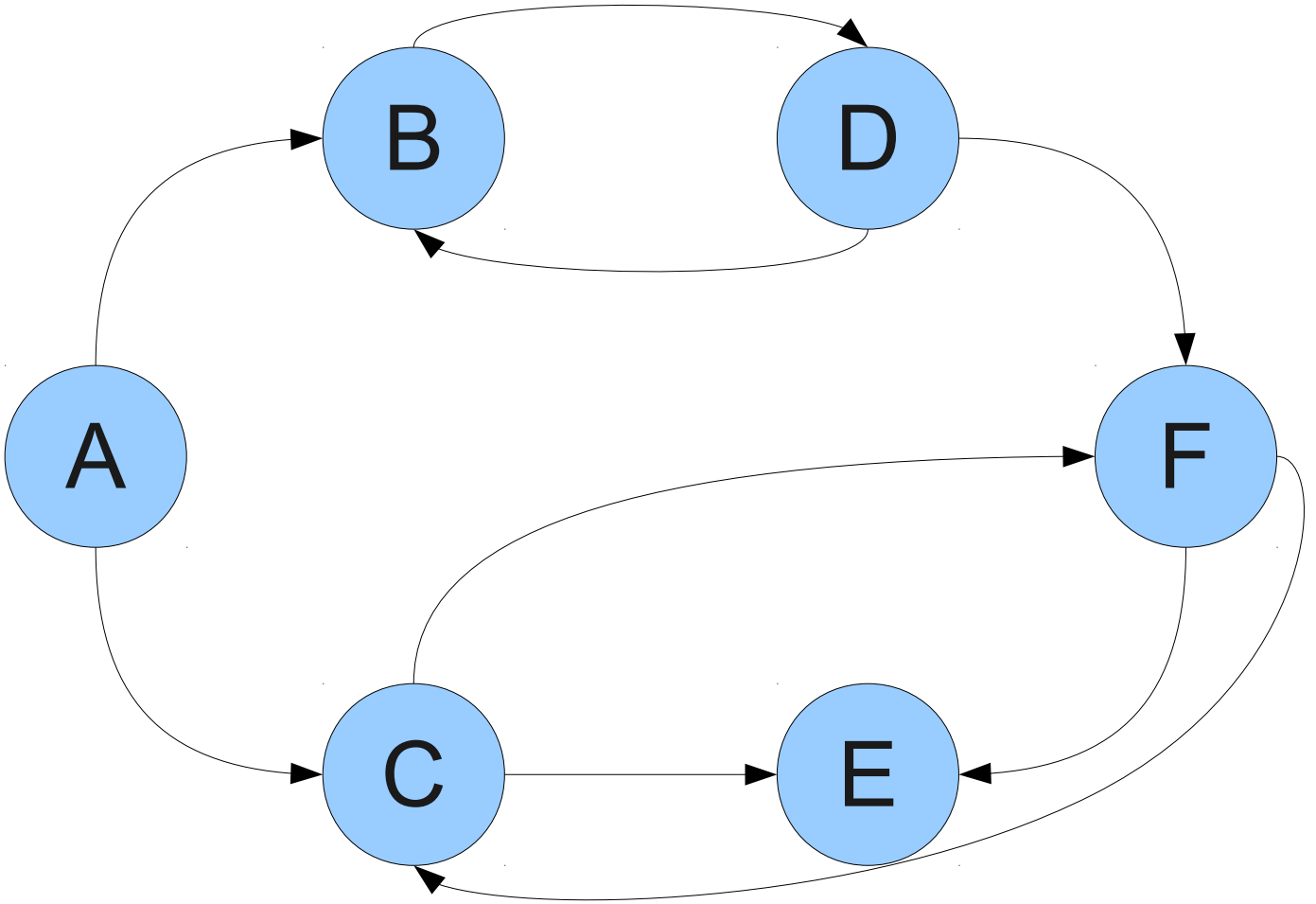
$((v_0, v_1), (v_1, v_2), \ldots, (v_n, v_0))$

that starts and ends at the same node.

A **simple path** is a path that does not contain a cycle.

A **simple cycle** is a cycle that does not contain a smaller cycle

Properties of Nodes



The **indegree** of a node is the number of edges entering that node.

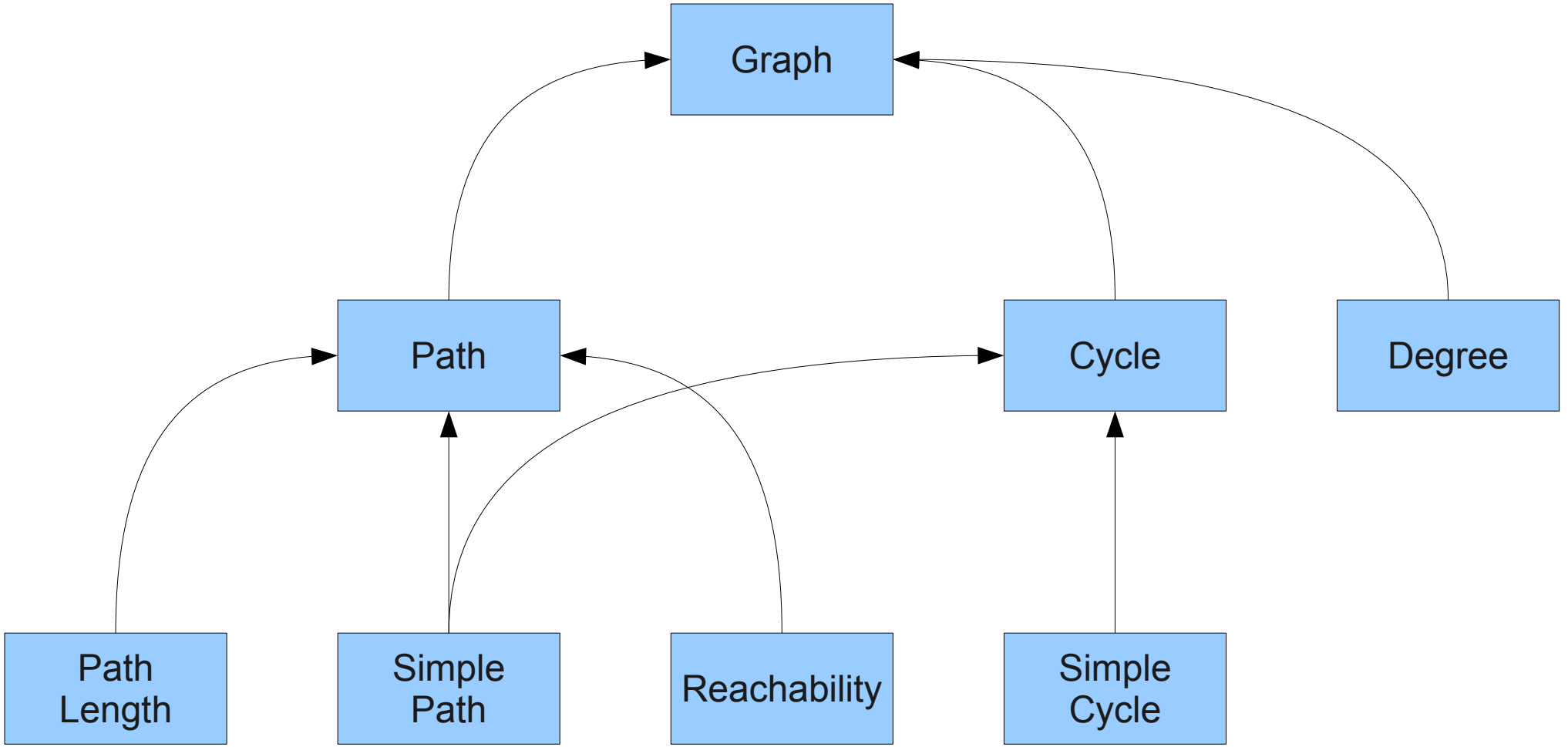
The **outdegree** of a node is the number of edges leaving that node.

In an undirected graph, these are the same and are called the **degree** of the node.

Summary of Terminology

- A **path** is a series of edges connecting two nodes.
  - The **length** of a path is the number of edges in the path.
  - A node v is **reachable** from u if there is a path from u to v.
- A **cycle** is a path from a node to itself.
- A **simple path** is a path without a cycle.
- A **simple cycle** is a cycle that does not contain a nested cycle.
- The **indegree** and **outdegree** of a node are the number of edges entering/leaving it.

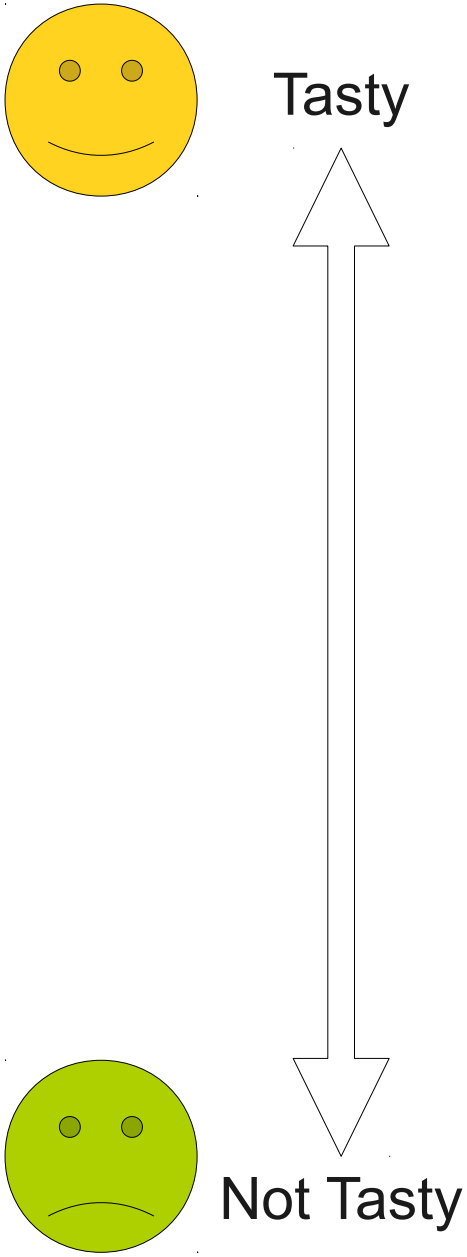
Representing Prerequisites

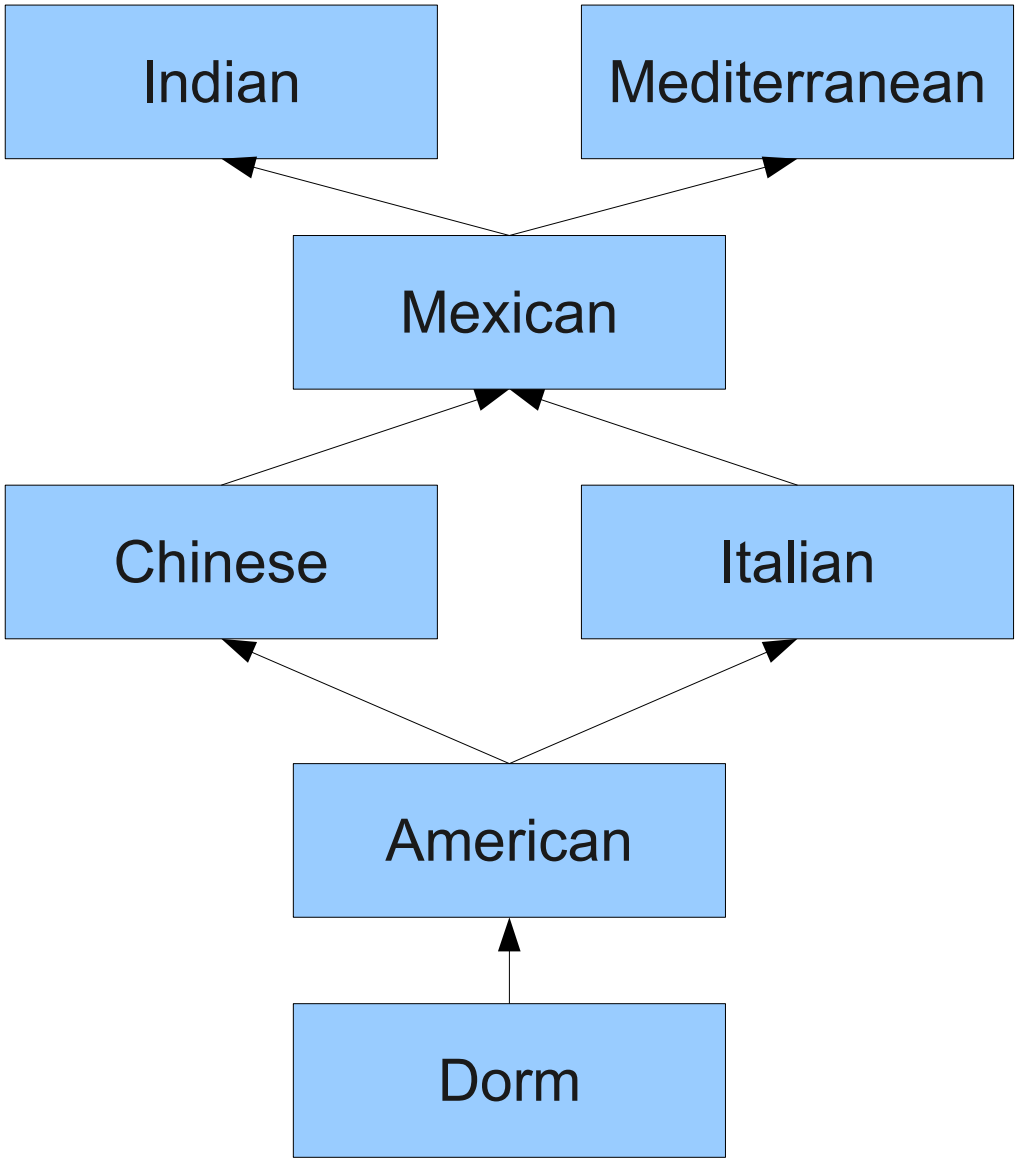


A **directed acyclic graph** (DAG) is a directed graph with no cycles.

Examples of DAGs

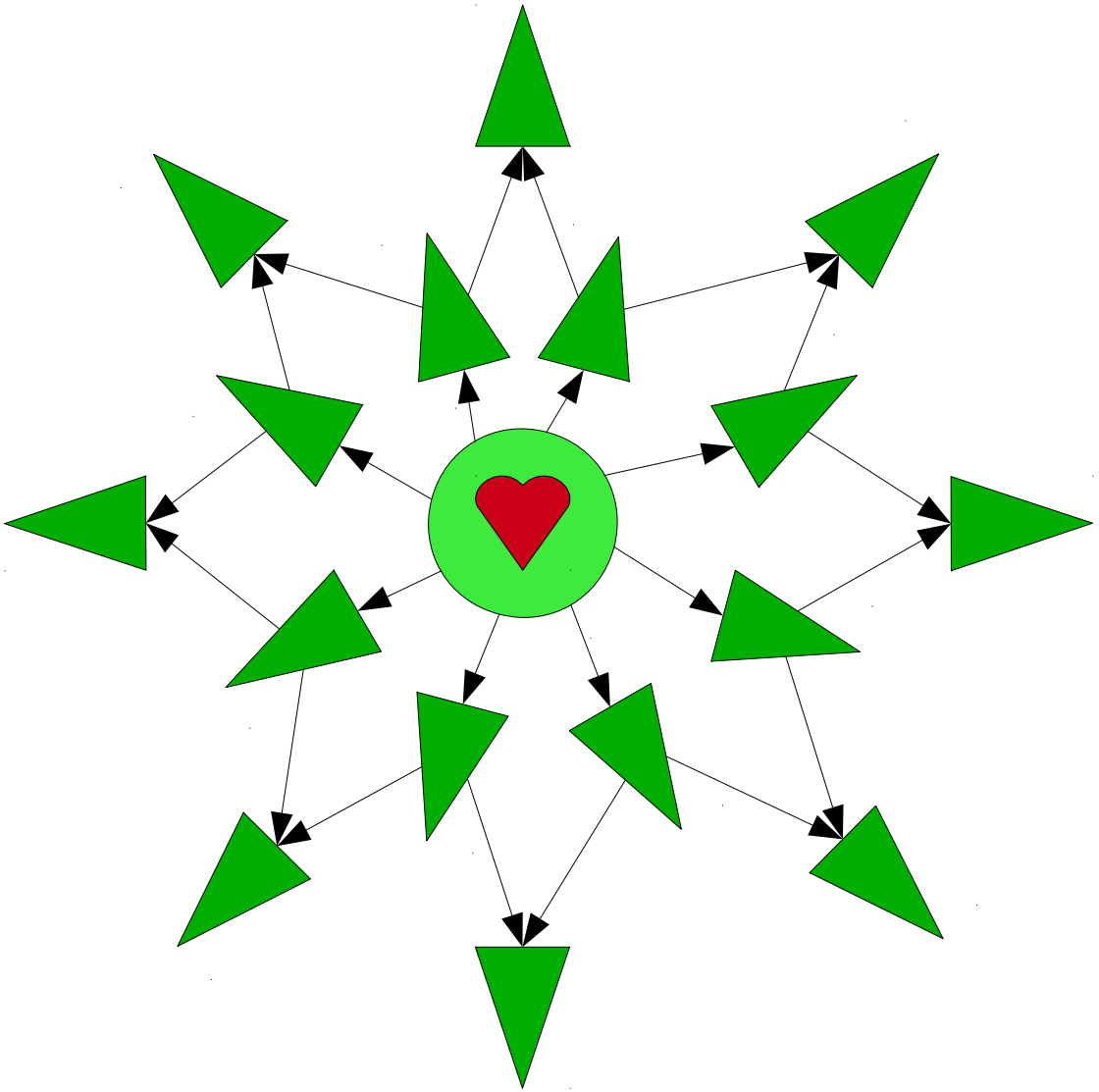
Examples of DAGs



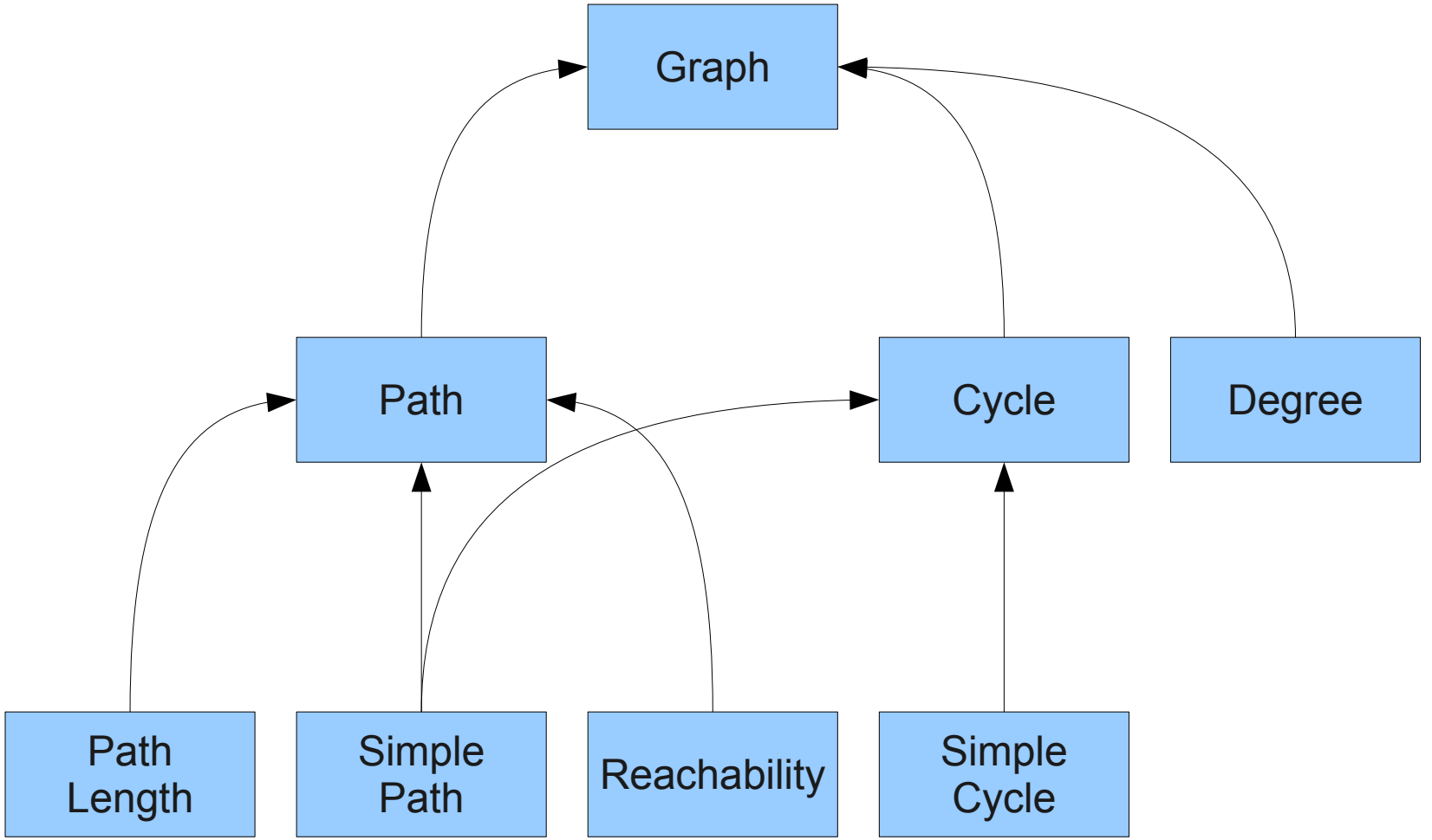


Examples of DAGs

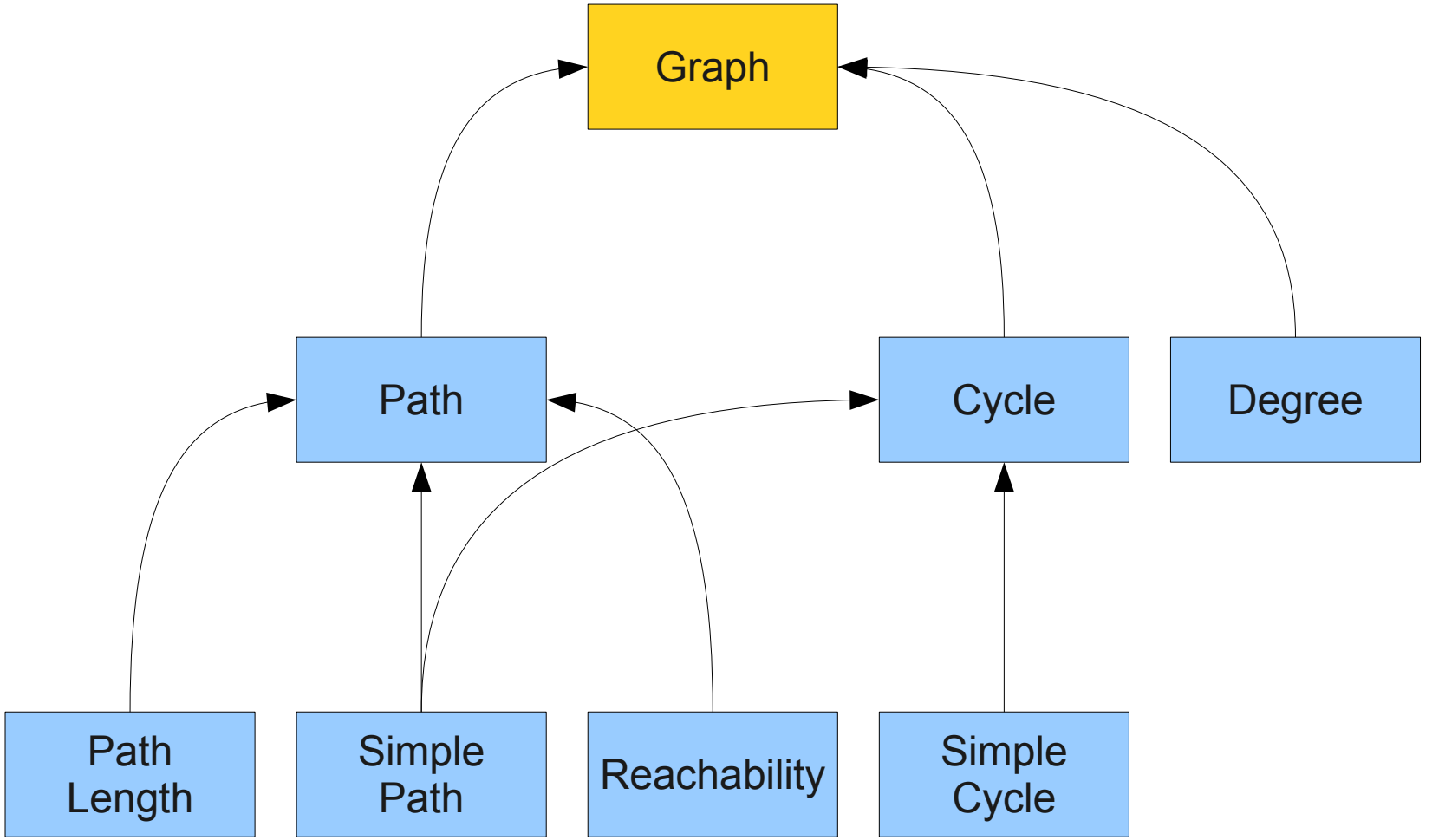
Examples of DAGs



Traversing a DAG

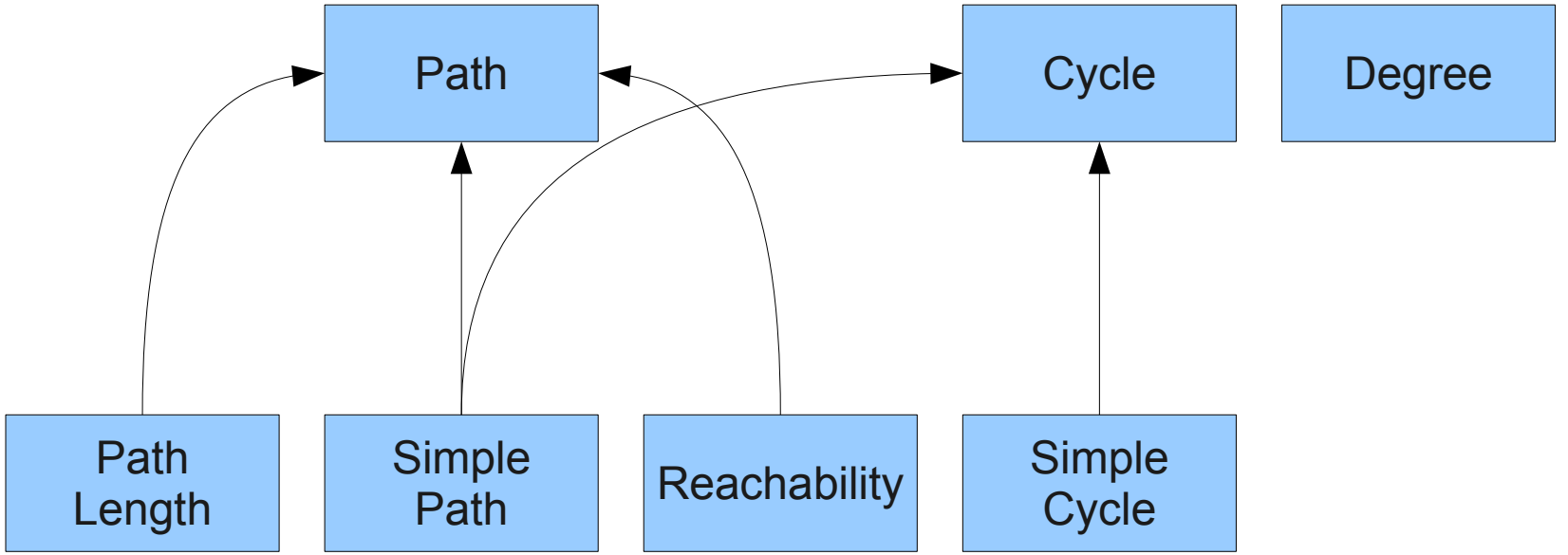


Traversing a DAG



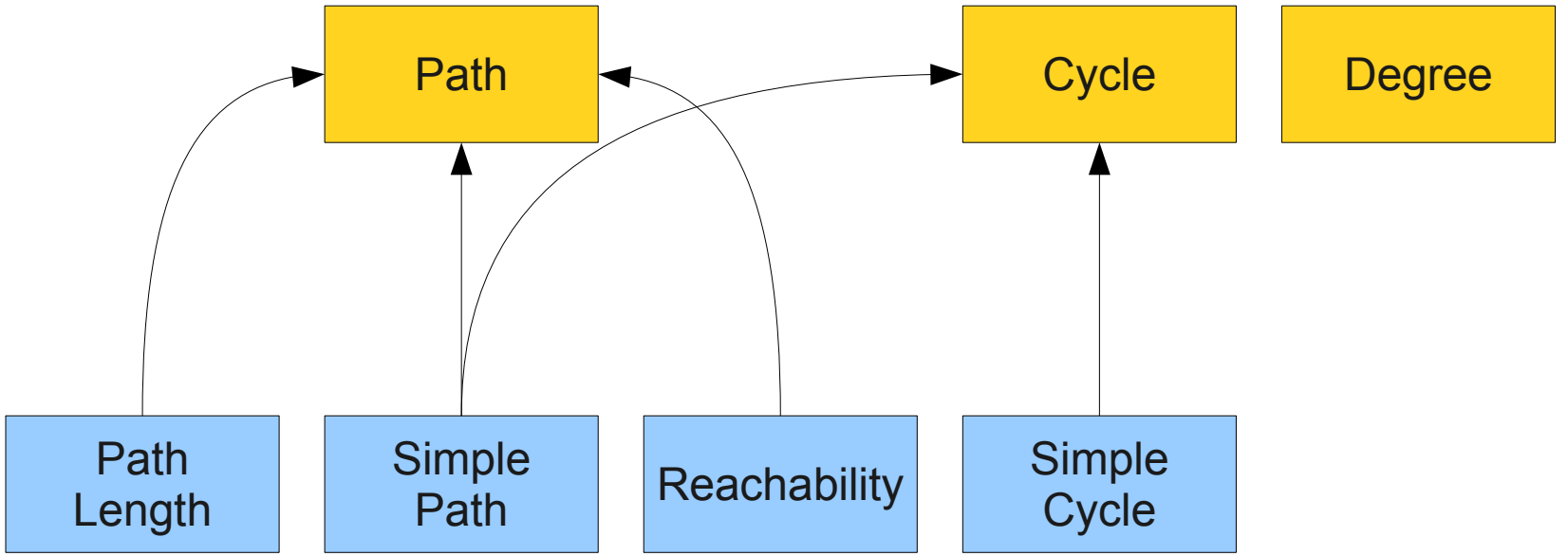
Graph

Traversing a DAG



Graph

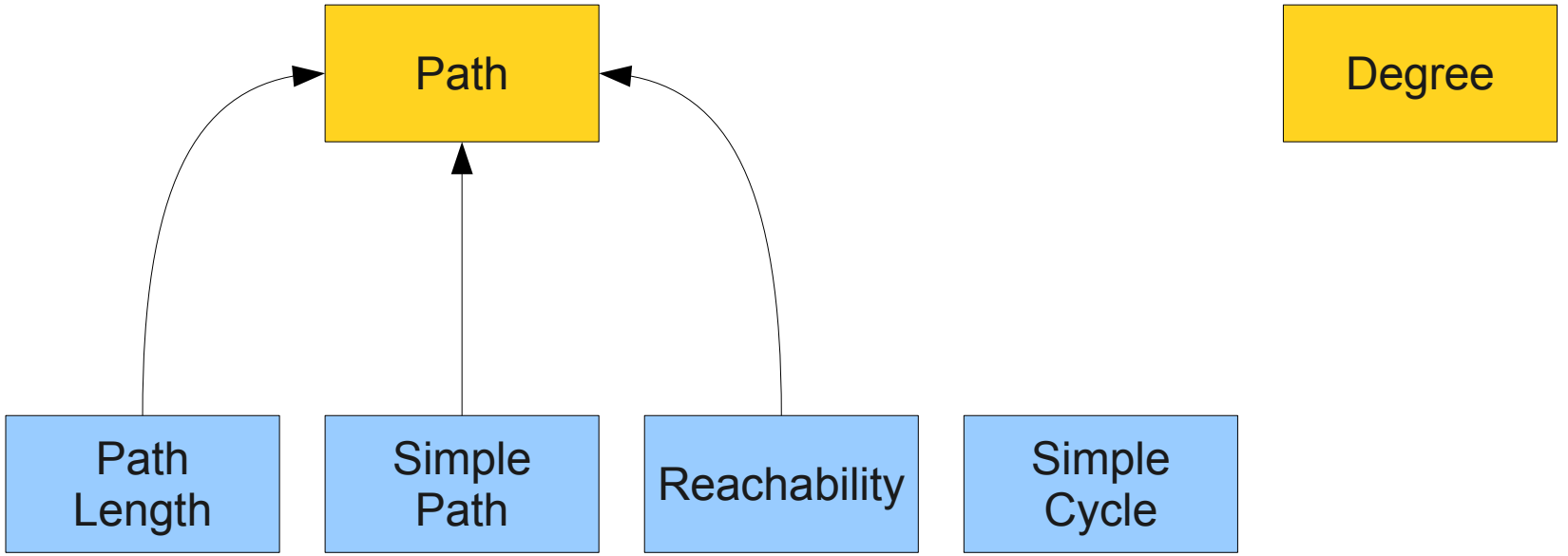
Traversing a DAG



Traversing a DAG

Graph

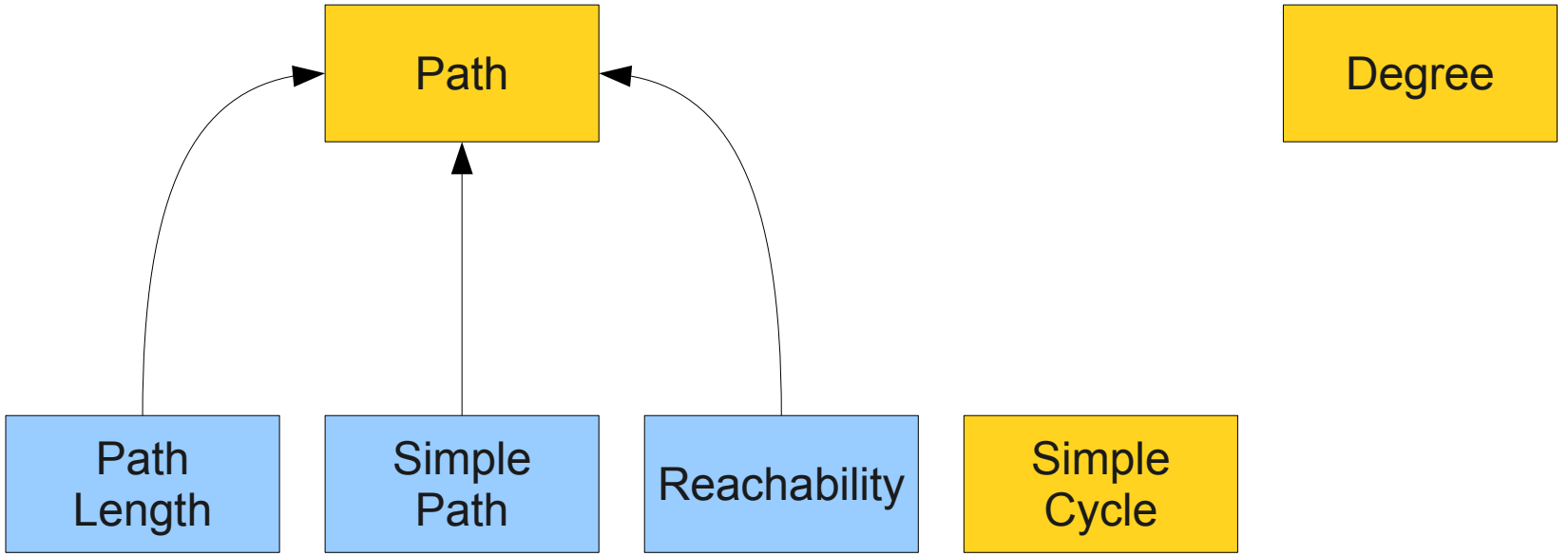
Cycle



Traversing a DAG

Graph

Cycle

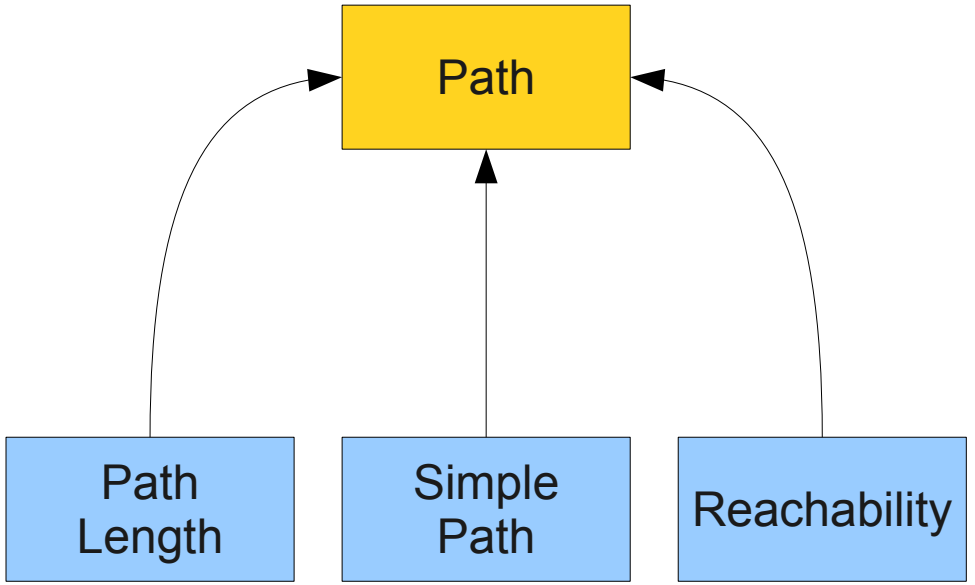


Traversing a DAG

Graph

Cycle

Simple Cycle



Degree

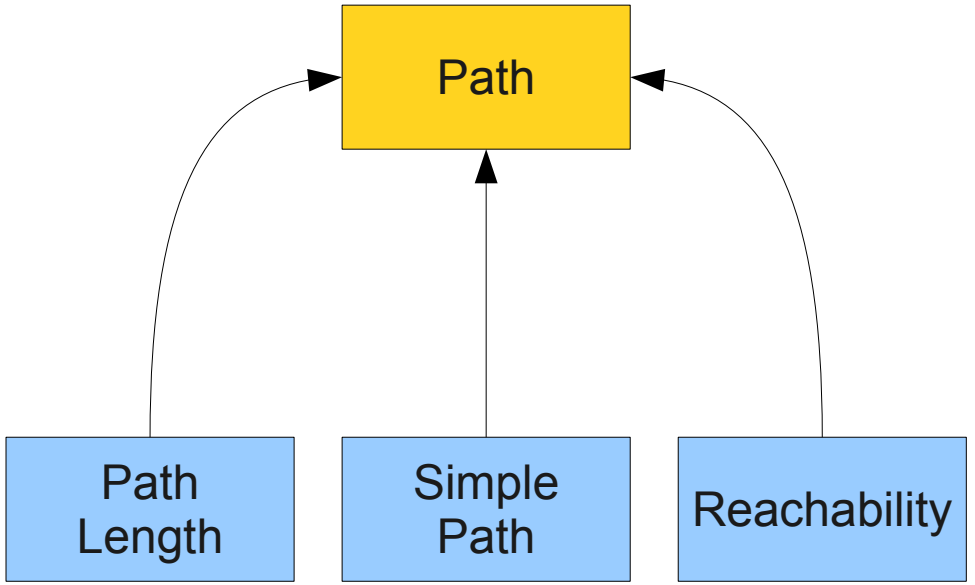
Traversing a DAG

Graph

Cycle

Simple Cycle

Degree



Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

Path

Path Length

Simple Path

Reachability

Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

Path

Path Length

Simple Path

Reachability

Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

Path

Path Length

Simple Path

Reachability

Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

Path

Path Length

Simple Path

Reachability

Traversing a DAG

Graph

Cycle

Simple Cycle

Degree

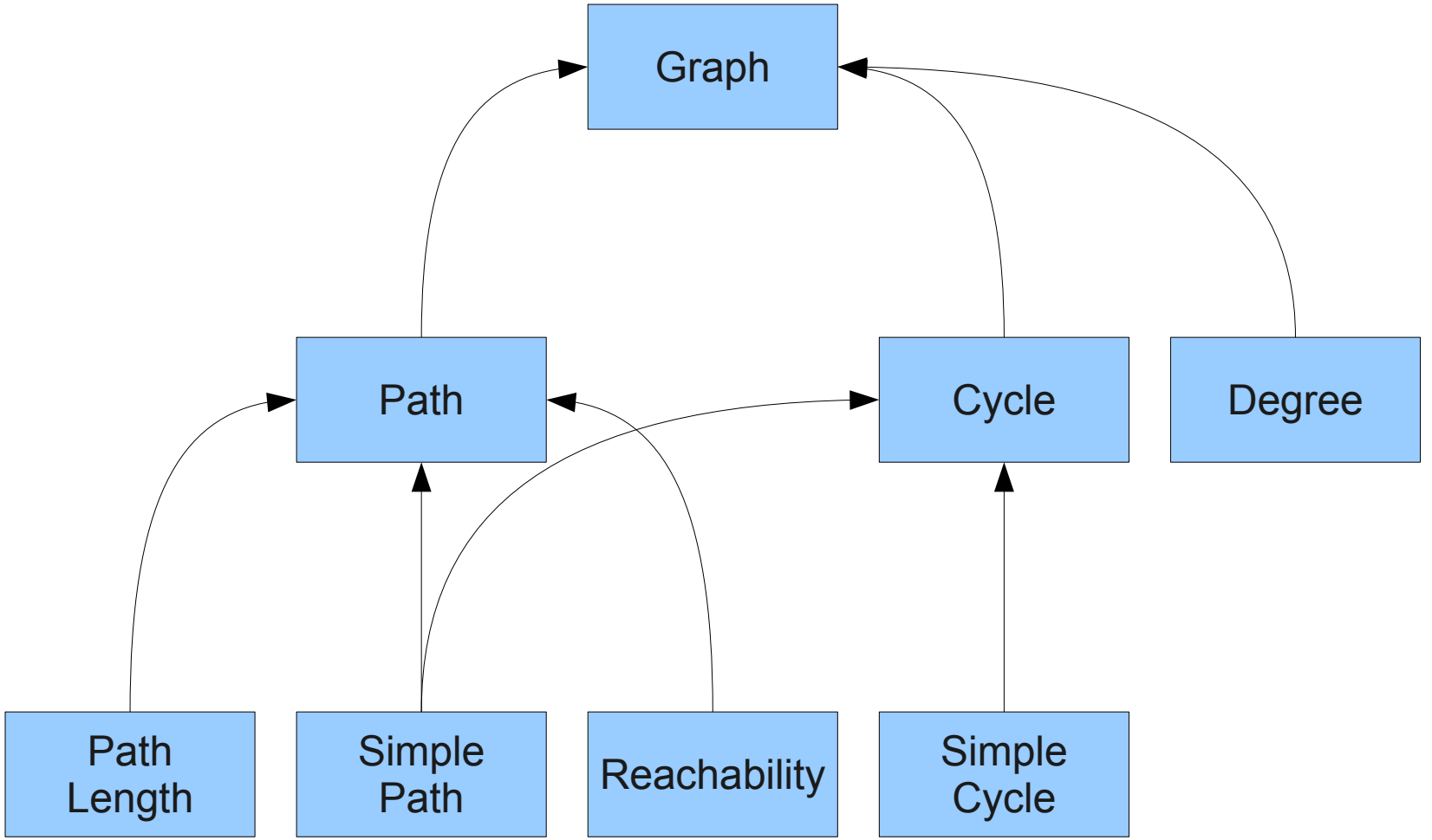
Path

Path Length

Simple Path

Reachability

Traversing a DAG



Graph

Cycle

Simple Cycle

Degree

Path

Path Length

Simple Path

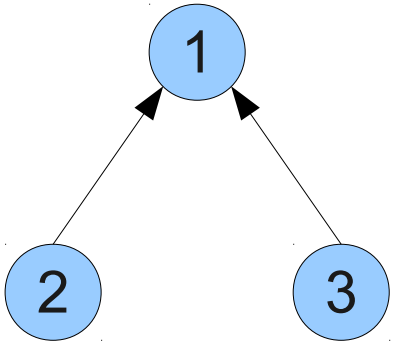
Reachability

Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings

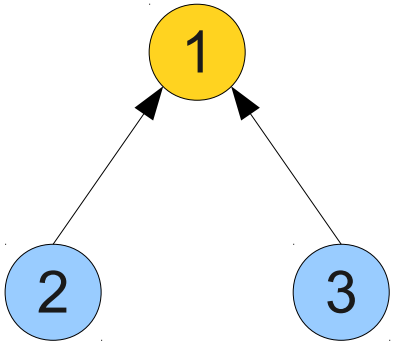
Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings



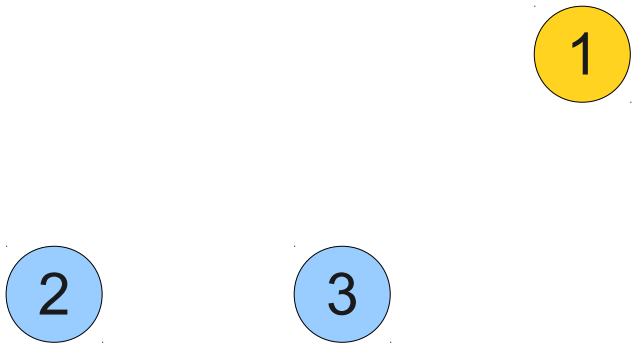
Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings



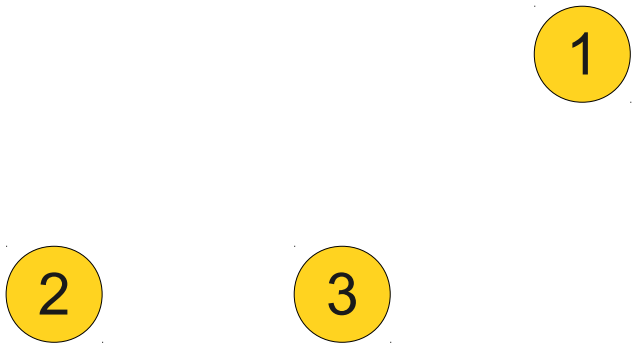
Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings



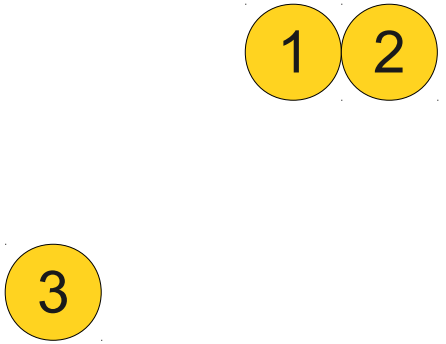
Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings



Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings

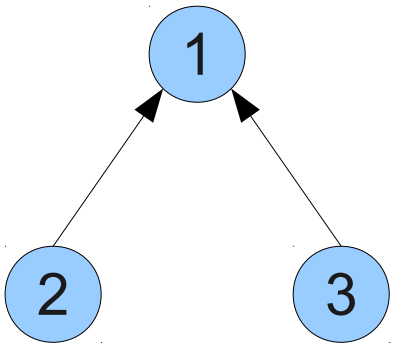


Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings

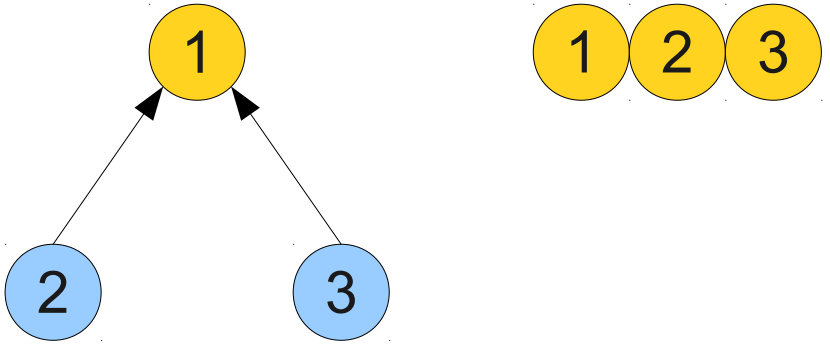
Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings



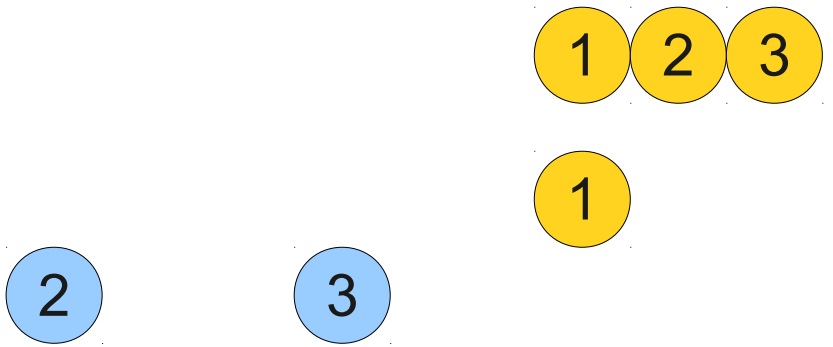
Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings



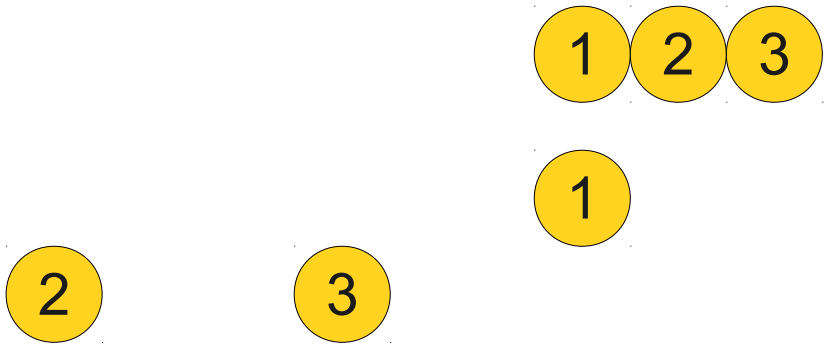
Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings



Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings

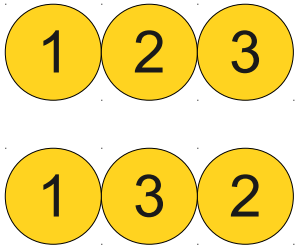


Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings

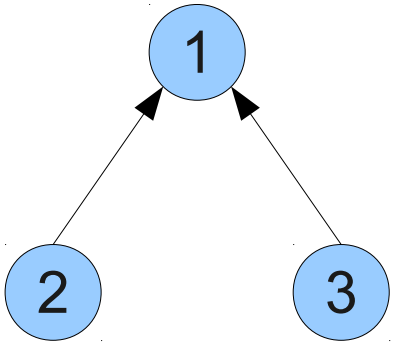
Topological Sort

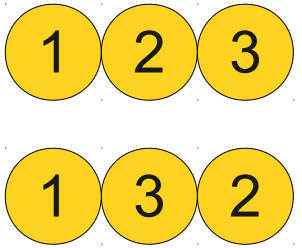
- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings



Topological Sort

- Order the nodes of a DAG so no node is picked before its predecessors.
- Algorithm:
  - Find a node with no outgoing edges (outgoing 0)
  - Remove it from the graph.
  - Add it to the resulting ordering.
- There may be many valid orderings





Relations

Relations

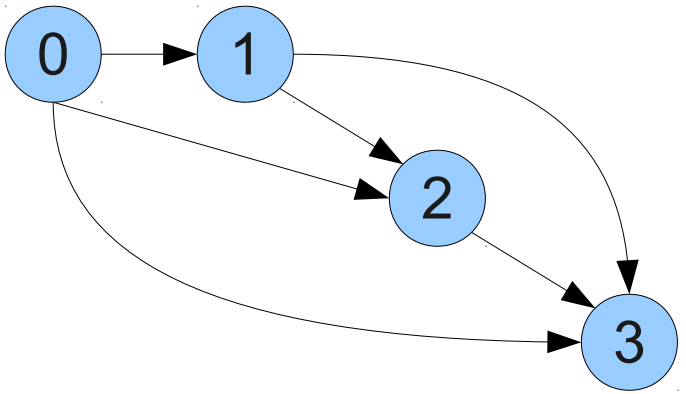
- A **binary relation** is a property that describes whether two objects are related in some way.
- Examples:
  - Less-than: $x < y$
  - Divisibility: x divides y evenly
  - Friendship: x is a friend of y
  - Tastiness: x is tastier than y
- If we have a binary relation R, we write aRb if a is **related** to b.
  - $a = b$
  - $a < b$
  - a “is tastier than” b

Relations as Sets

- Formally, a relation is a set of ordered pairs representing the pairs for which the relation is true.
  - Equality: {(0, 0), (1, 1), (2, 2), … }
  - Less-than: {(0, 1), (0, 2), …, (1, 2), (1, 3), … }
- Formally, we have that
  - $aRb \equiv (a, b) \in R$
- The binary relations we'll discuss today will be **binary relations over a set A**.
  - Each relation is a subset of A^2.

Binary Relations and Graphs

- Each (directed) graph defines a binary relation:
  - aRb if (a, b) is an edge.
- Each binary relation defines a graph:
  - (a, b) is an edge if aRb.
- Example: Less-than



An Important Question

- Why study binary relations and graphs separately?
- **Simplicity**:
  - Certain operations feel more “natural” on binary relations than on graphs and vice-versa.
  - Converting a relation to a graph might result in an overly complex graph.
- **Terminology**:
  - Vocabulary for graphs often different from that for relations.

Equivalence Relations

“x and y have the same color”

“x and y have the same shape”

“x = y”

“x and y have the same area”

“x and y are programs that produce the same output”

Informally

An **equivalence relation** is a relation that indicates when objects have some trait in common.

Do not use this definition in proofs!
It's just an intuition!

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

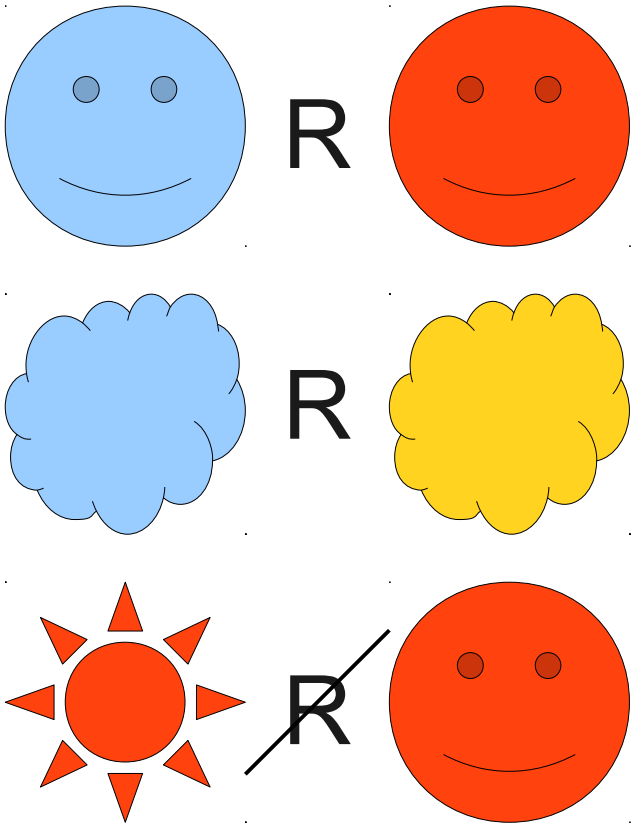
Recall: This symbol means "is defined as"

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

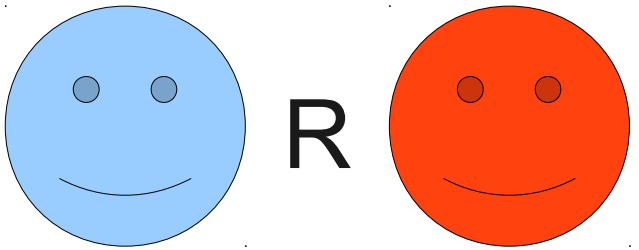


Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

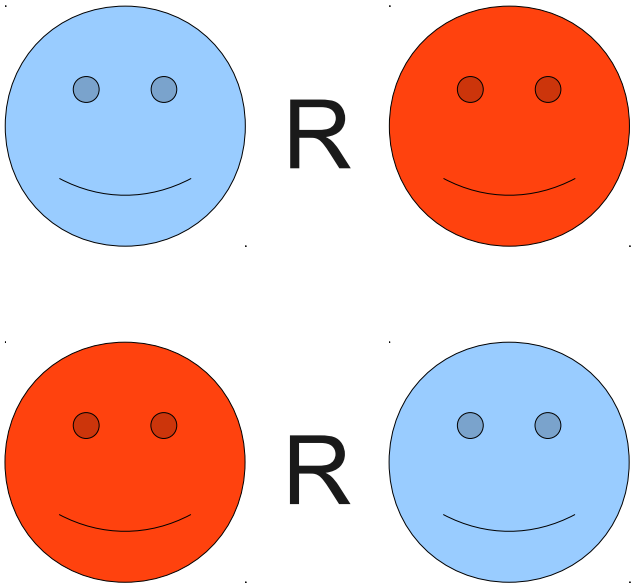
Properties of Equivalence Relations

xRy ≡ x and y have the same shape.



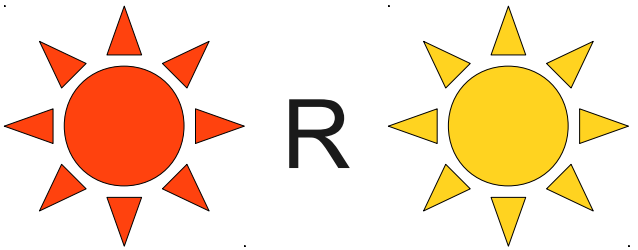
Properties of Equivalence Relations

xRy ≡ x and y have the same shape.



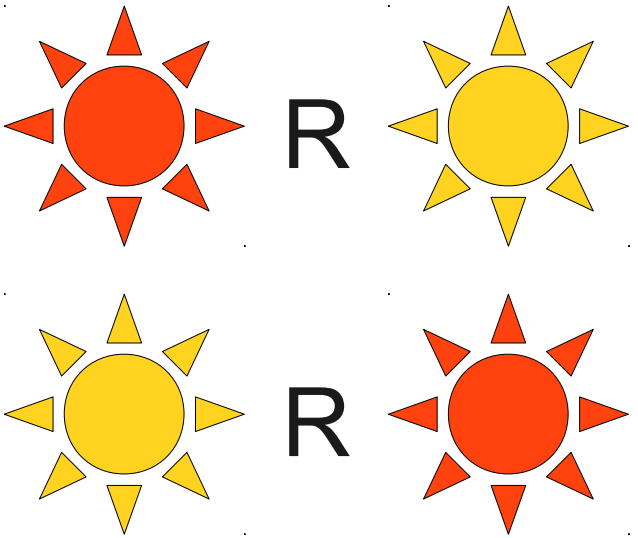
Properties of Equivalence Relations

xRy ≡ x and y have the same shape.



Properties of Equivalence Relations

xRy ≡ x and y have the same shape.



Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

xRy

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

xRy

yRx

Symmetry

A binary relation R is called **symmetric** if whenever xRy, yRx.

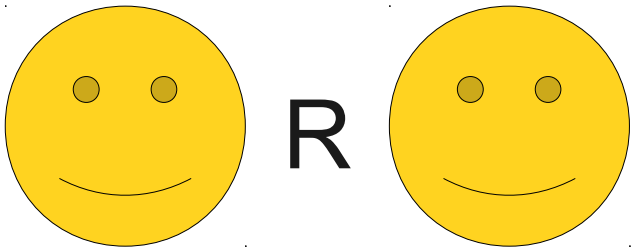
This definition (and others like it) can be used in formal proofs.

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

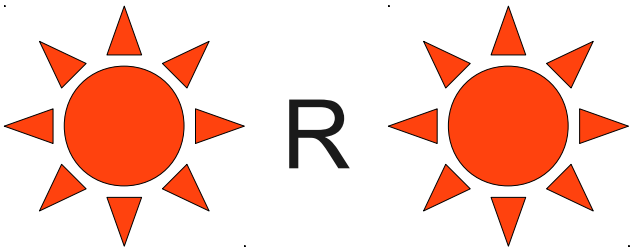
Properties of Equivalence Relations

xRy ≡ x and y have the same shape.



Properties of Equivalence Relations

xRy ≡ x and y have the same shape.



Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

xRx

Reflexivity

A binary relation R over a set A is called **reflexive** if for any $x \in A$, xRx

Some Reflexive Relations

- Equality:
  - For any x, $x = x$.
- Not greater than:
  - For any integer x, $x \leq x$.
- Subset:
  - For any set S, $S \subseteq S$.

Properties of Equivalence Relations

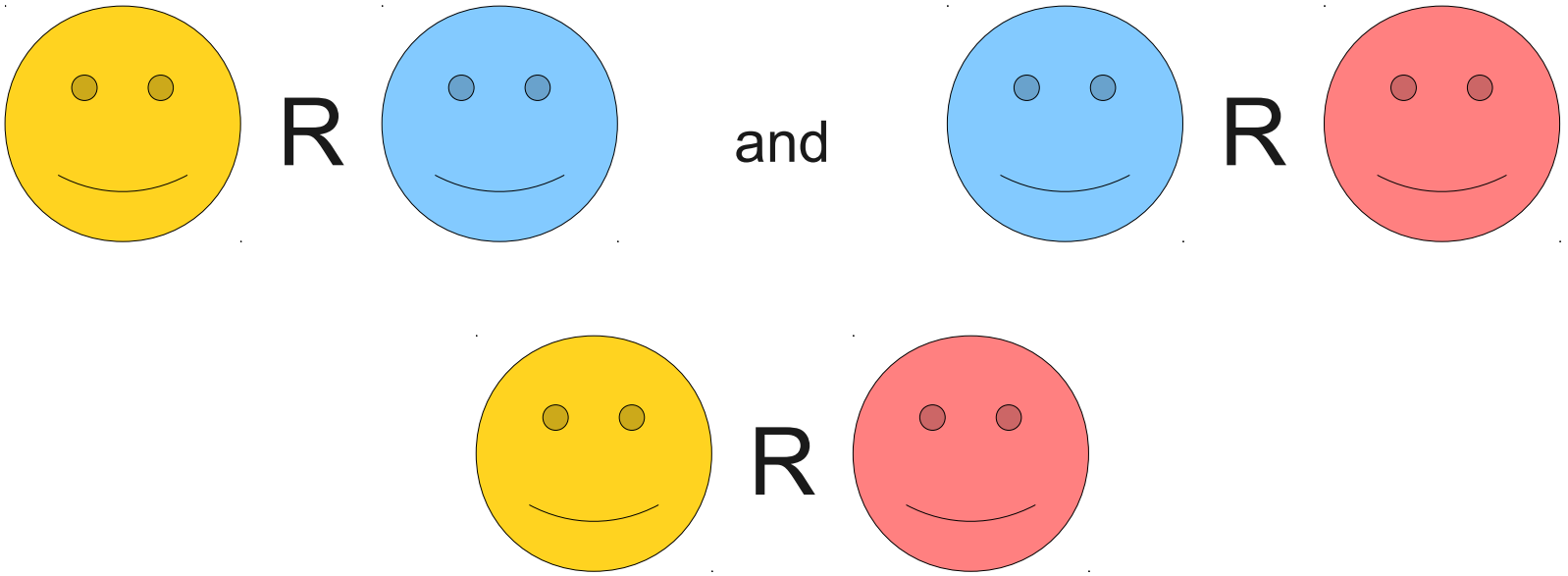
xRy ≡ x and y have the same shape.

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

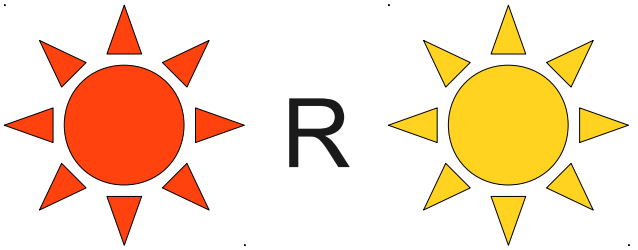


Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.



Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

xRy and yRz

Properties of Equivalence Relations

xRy ≡ x and y have the same shape.

xRy and yRz

xRz

Transitivity

A binary relation R is called **transitive** if whenever xRy and yRz, xRz.

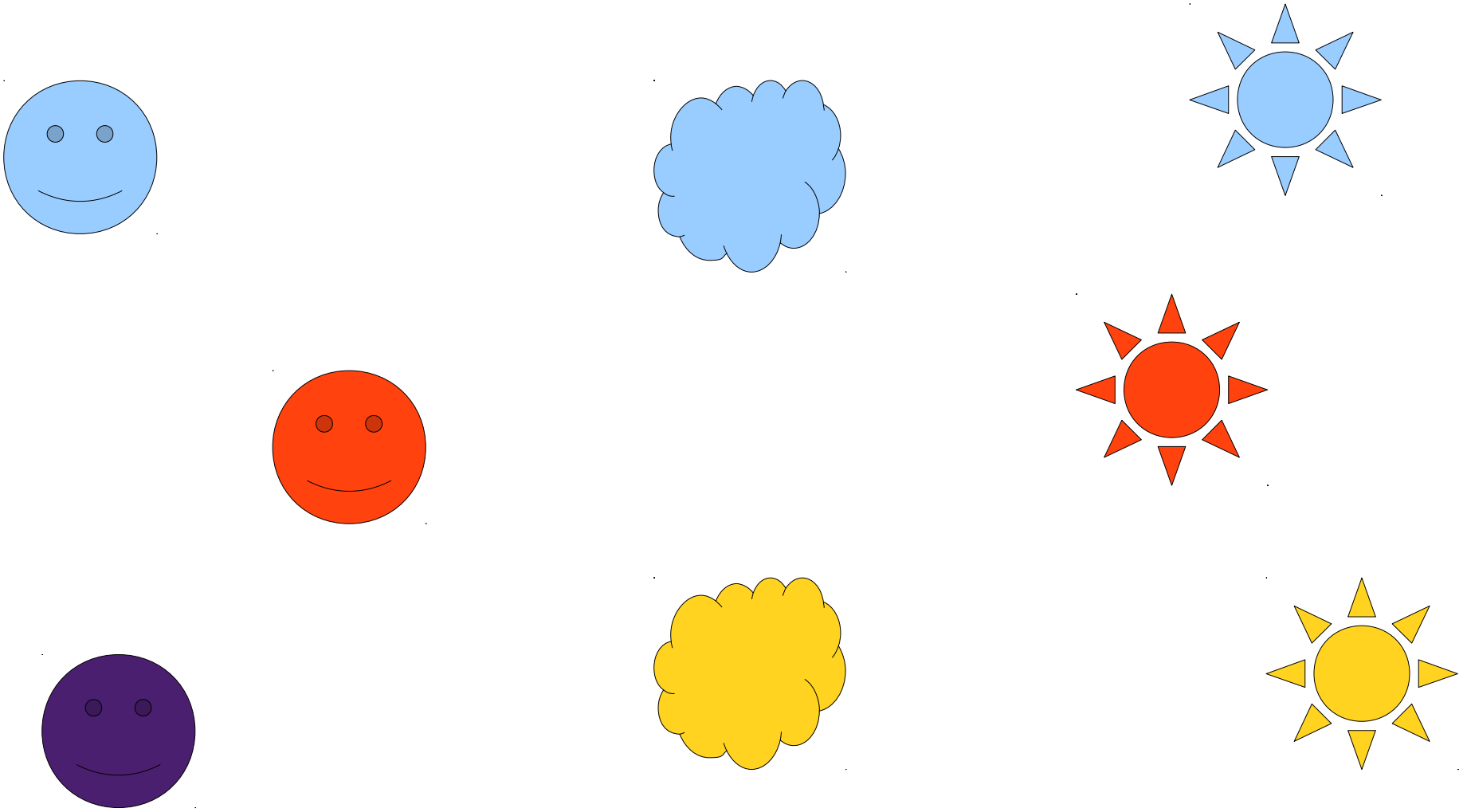
Some Transitive Relations

- Equality:
  - $x = y$ and $y = z$ implies $x = z$.
- Less-than:
  - $x < y$ and $y < z$ implies $x < z$.
- Subset:
  - $S \subseteq T$ and $T \subseteq U$ implies $S \subseteq U$.

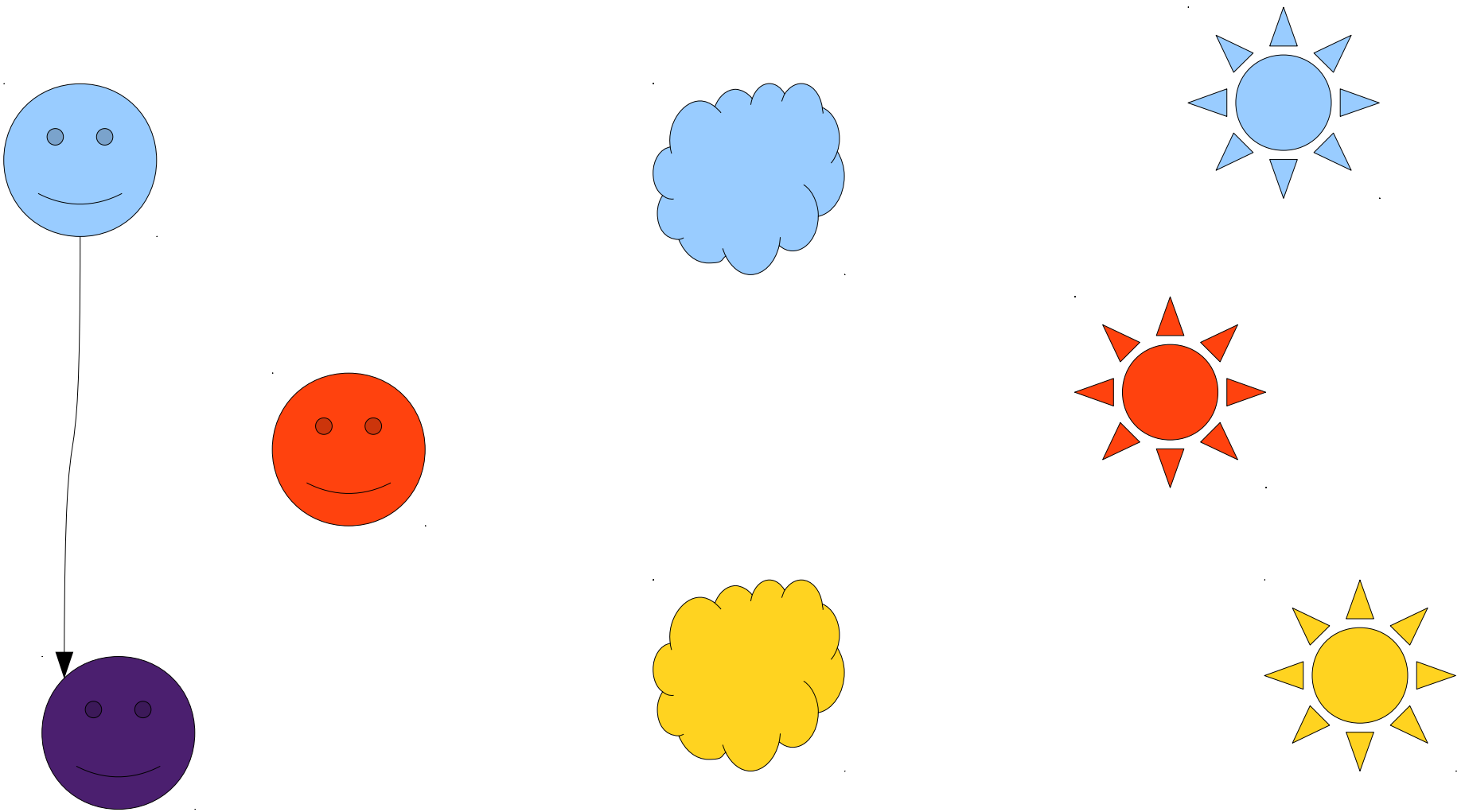
Equivalence Relations

A binary relation R is called an **equivalence relation** if it is

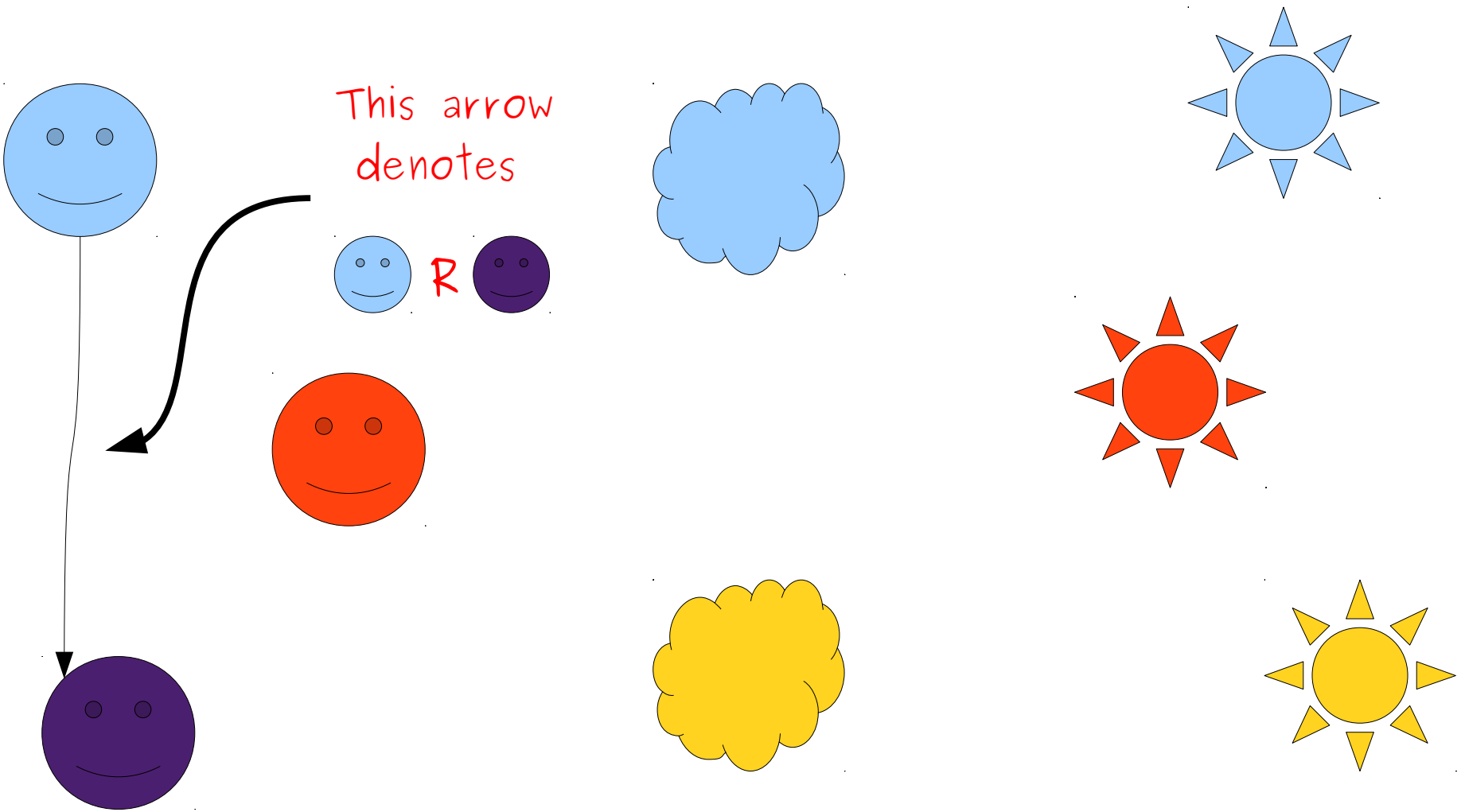
- **reflexive**,
- **symmetric**, and
- **transitive**.



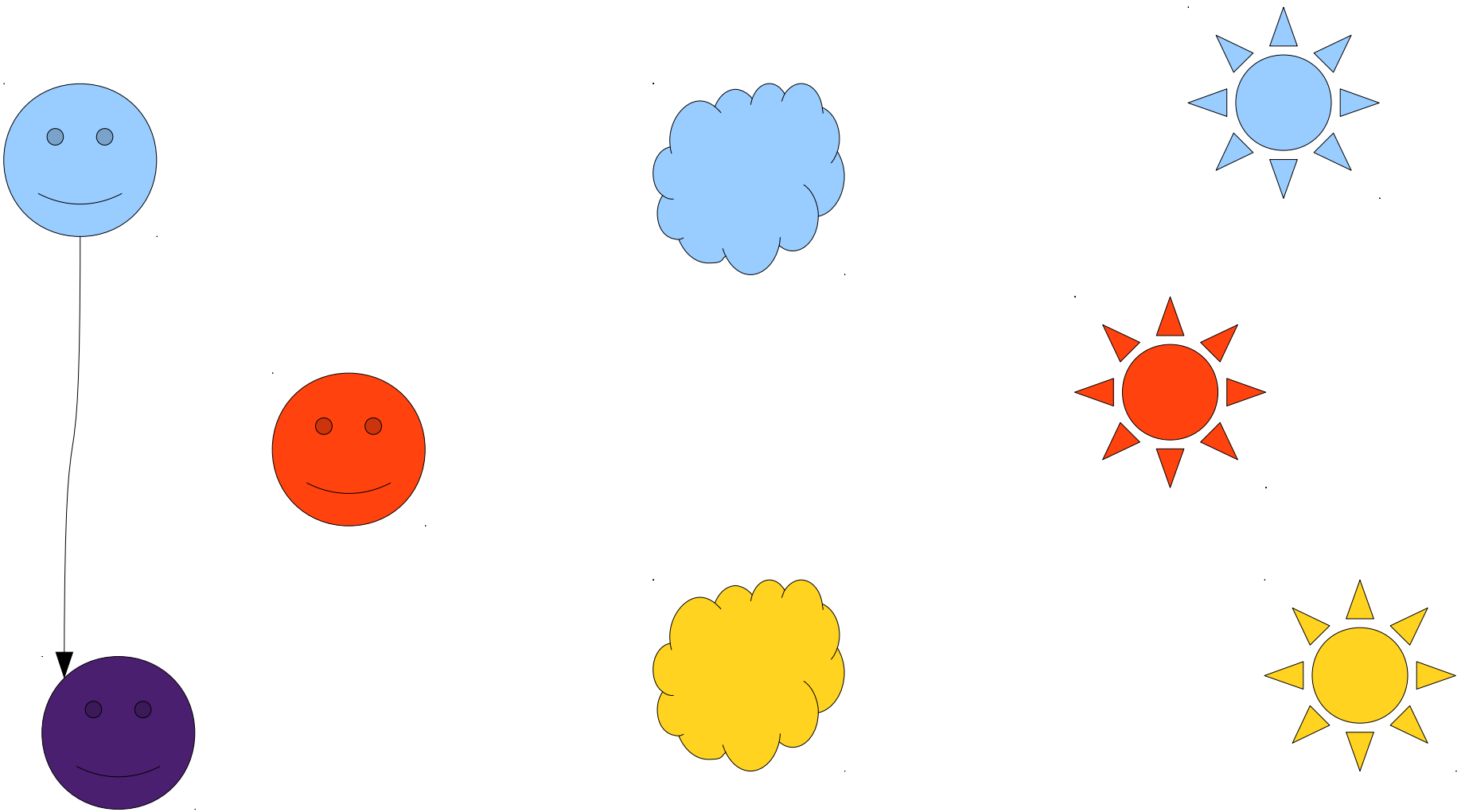
xRy ≡ x and y have the same shape.



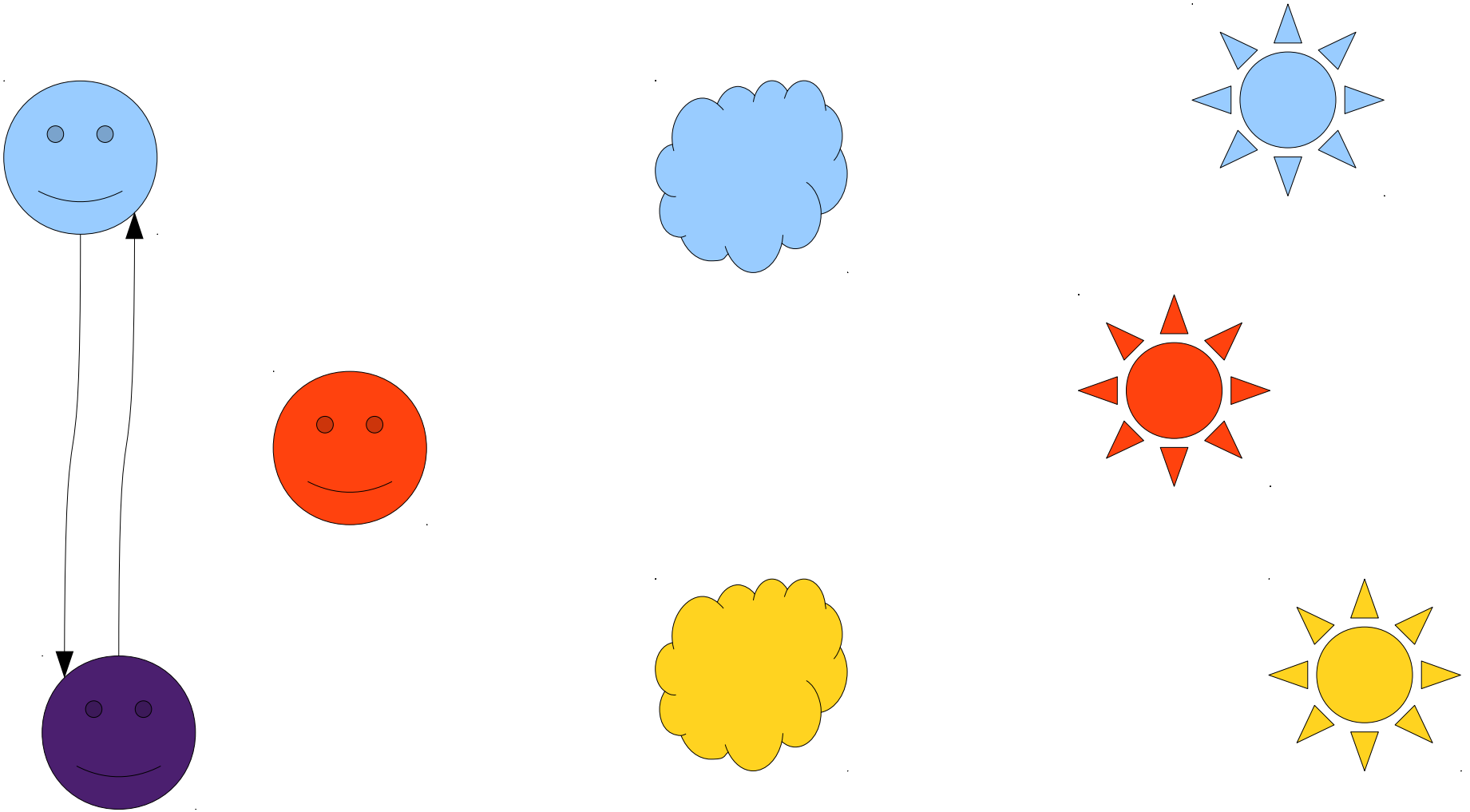
xRy ≡ x and y have the same shape.



xRy ≡ x and y have the same shape.

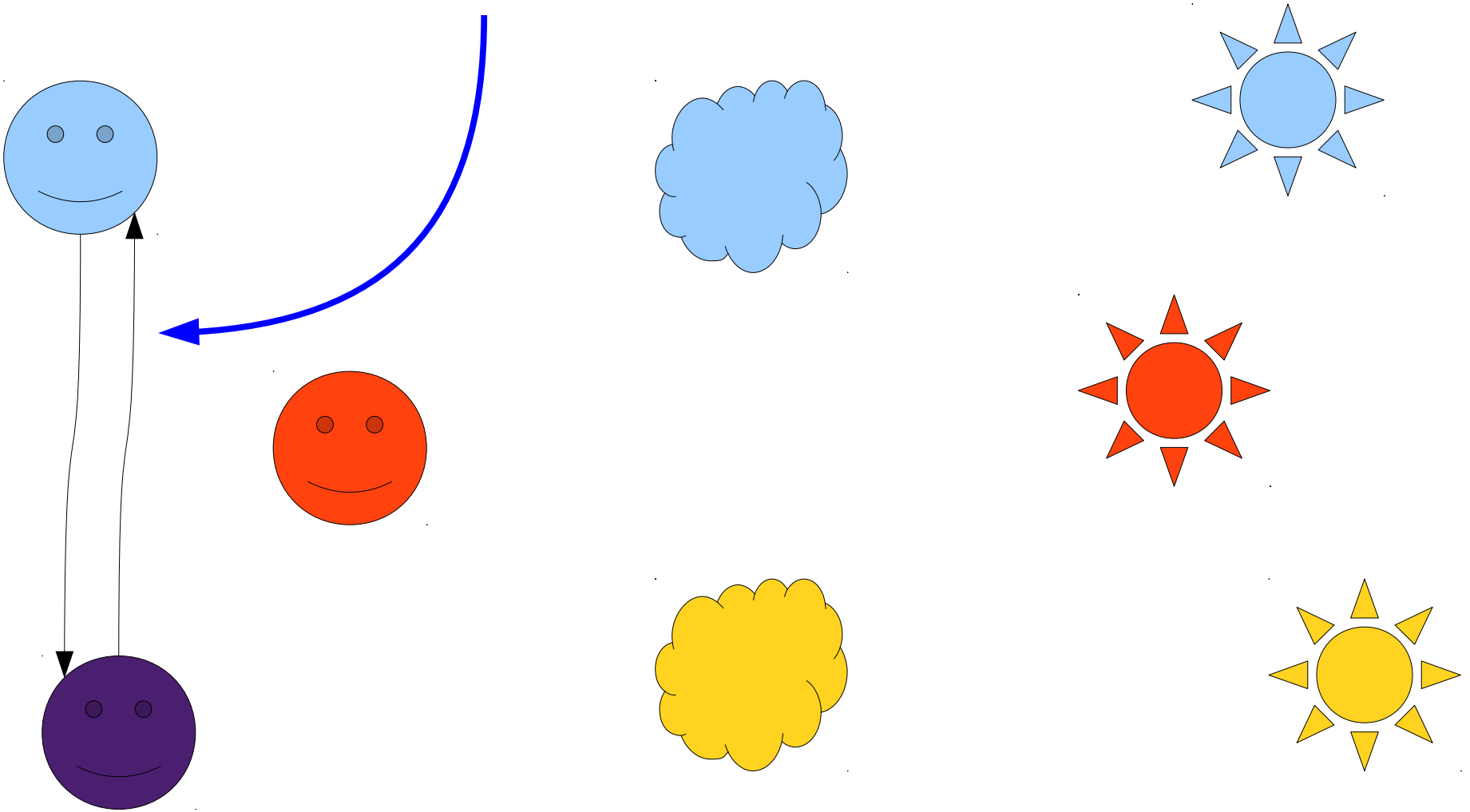


xRy ≡ x and y have the same shape.

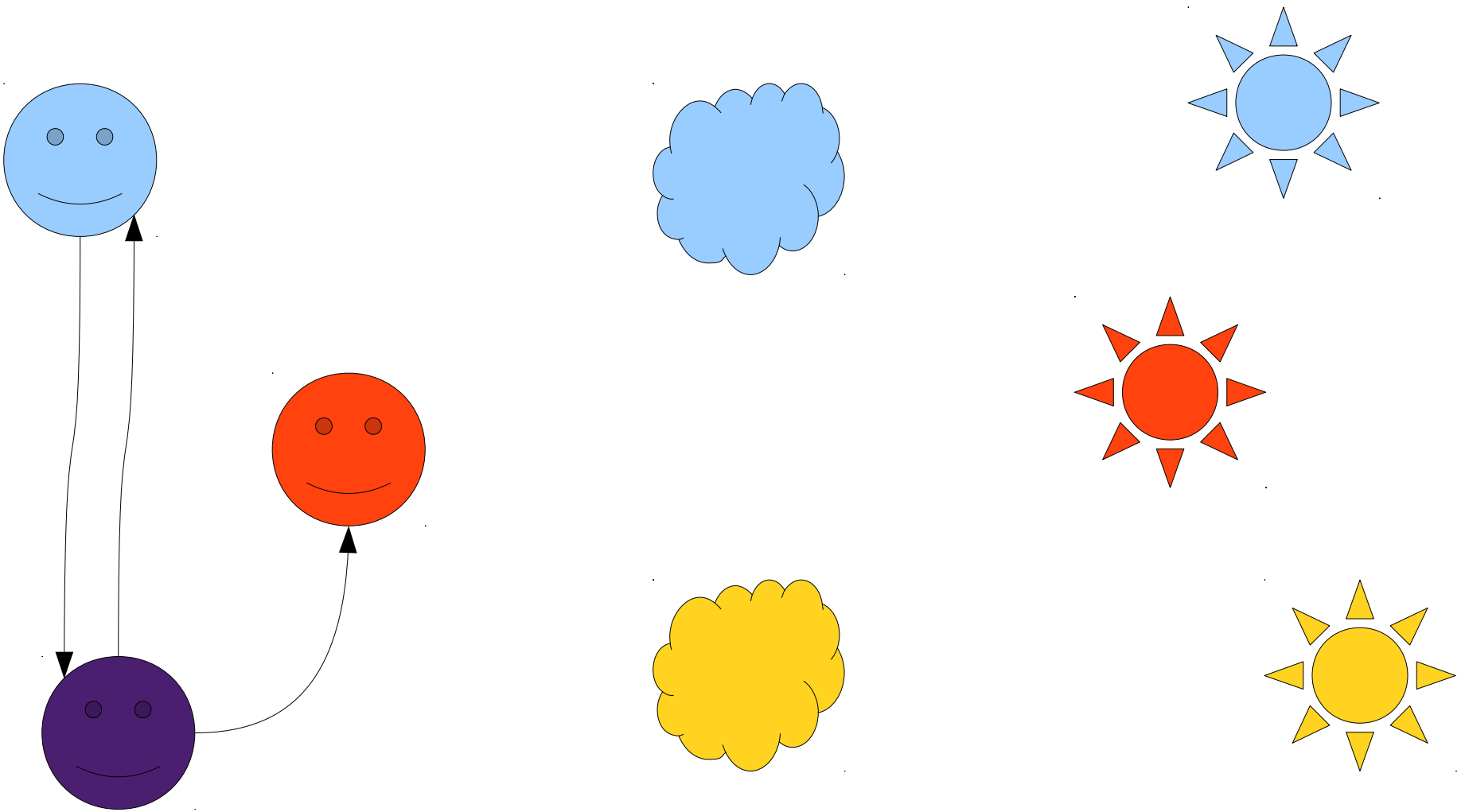


xRy ≡ x and y have the same shape.

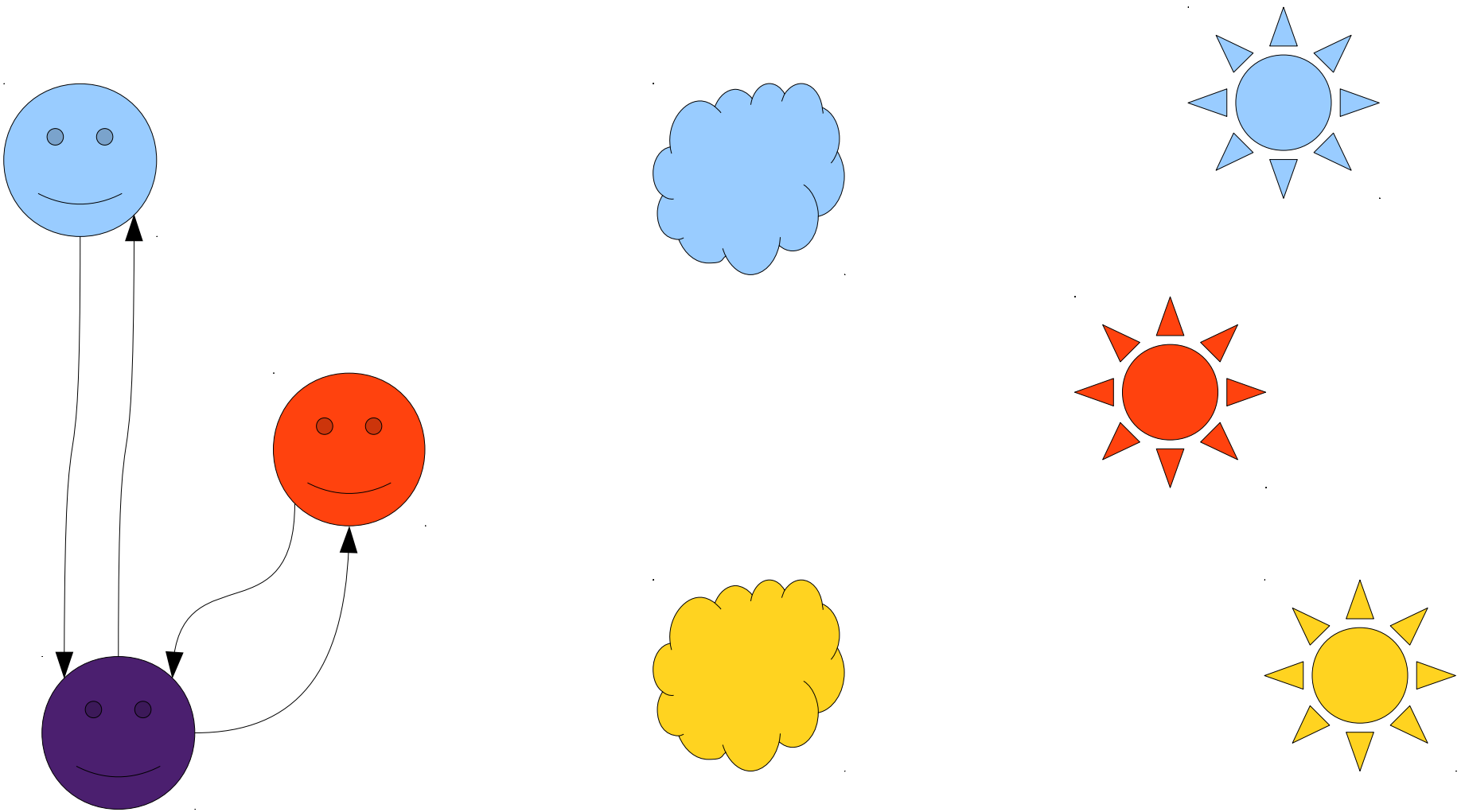
What property says this edge must be here?



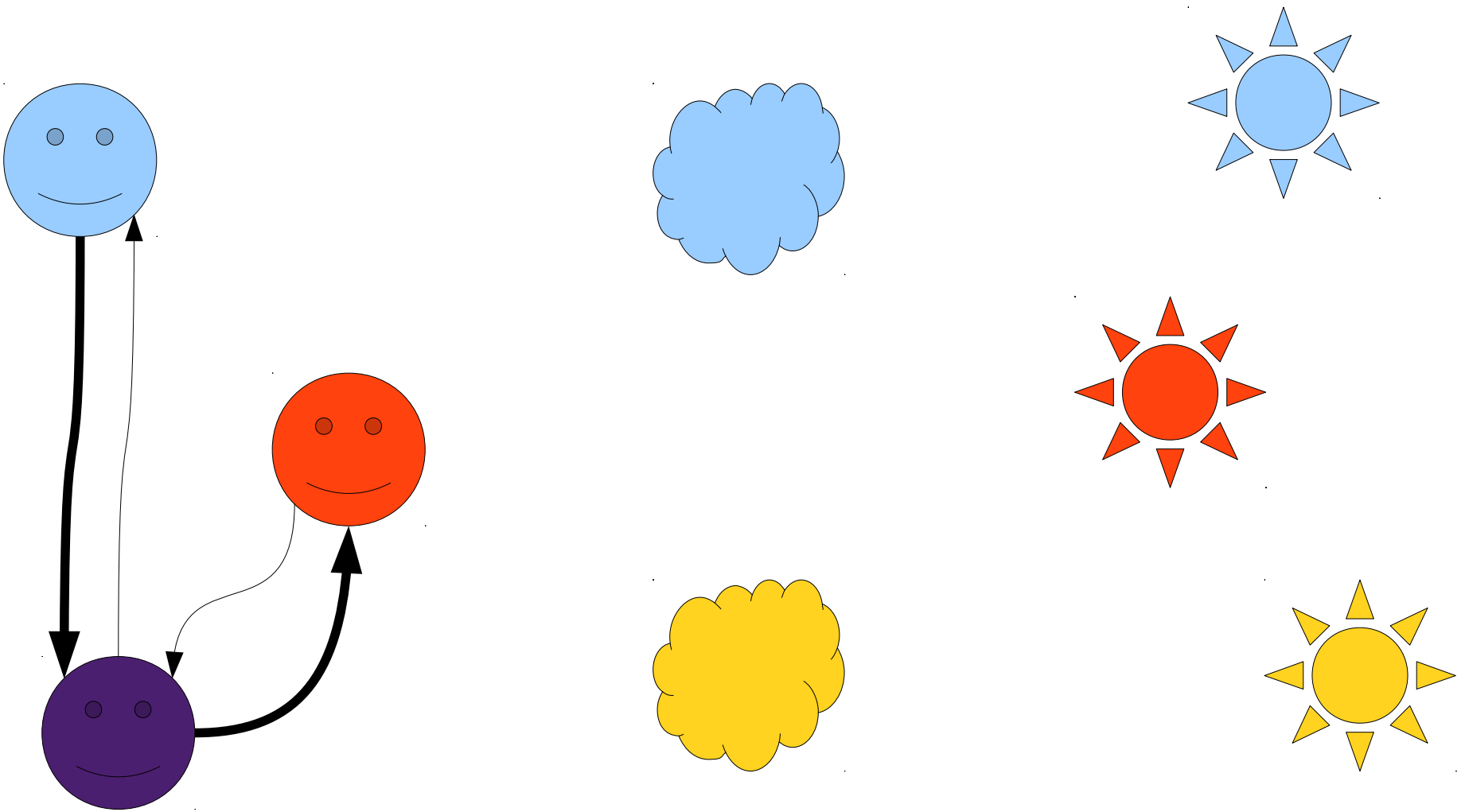
xRy ≡ x and y have the same shape.



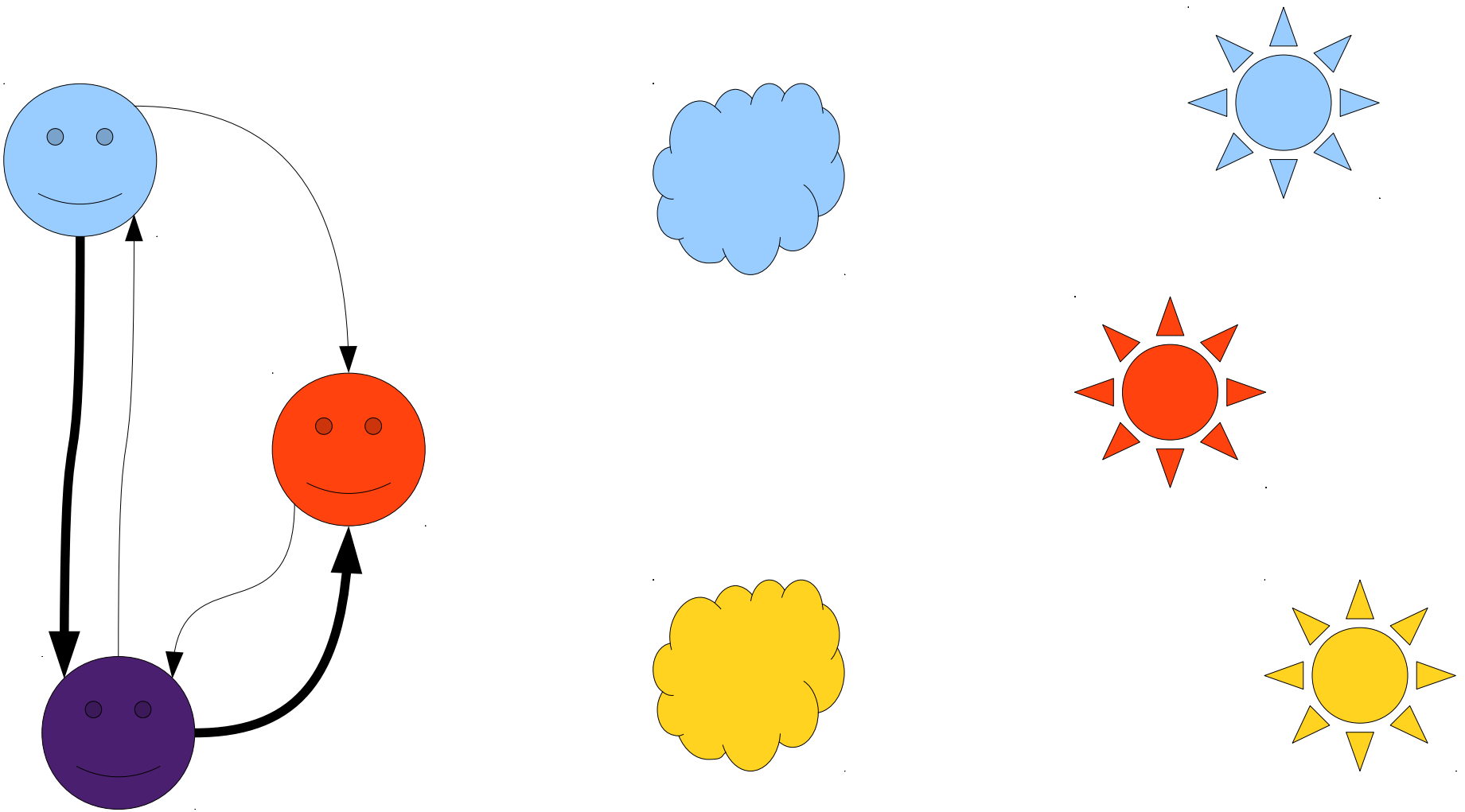
xRy ≡ x and y have the same shape.



xRy ≡ x and y have the same shape.

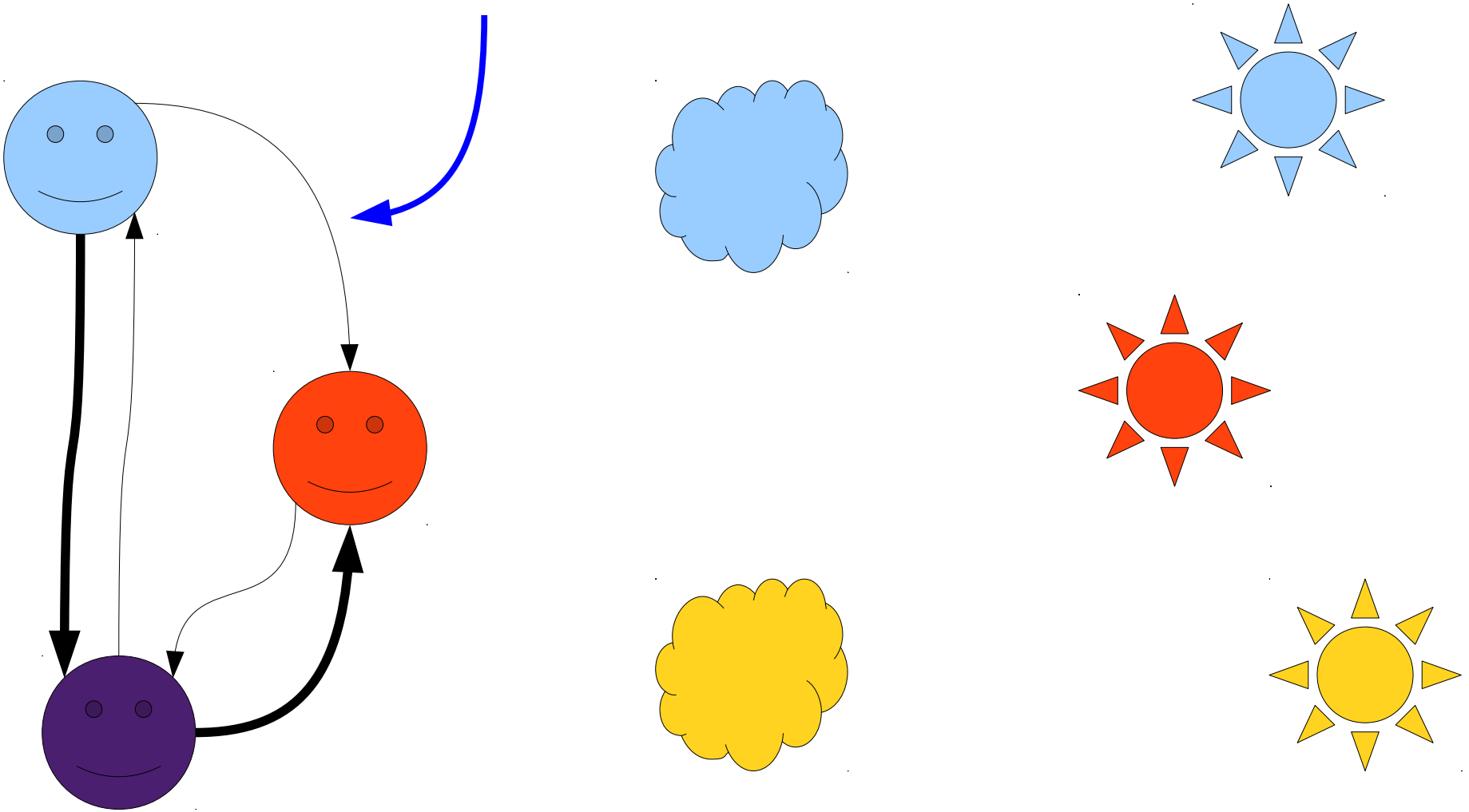


xRy ≡ x and y have the same shape.

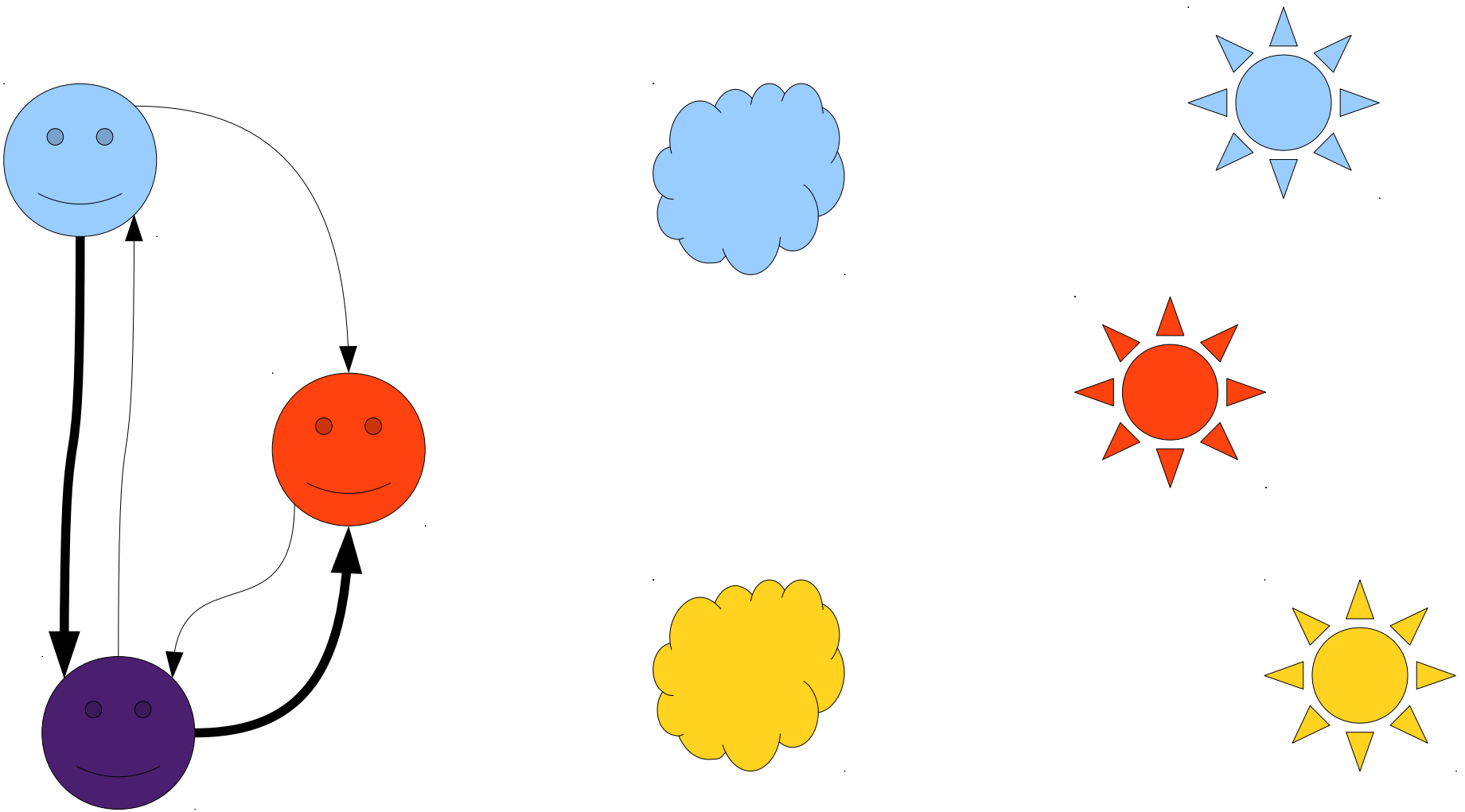


xRy ≡ x and y have the same shape.

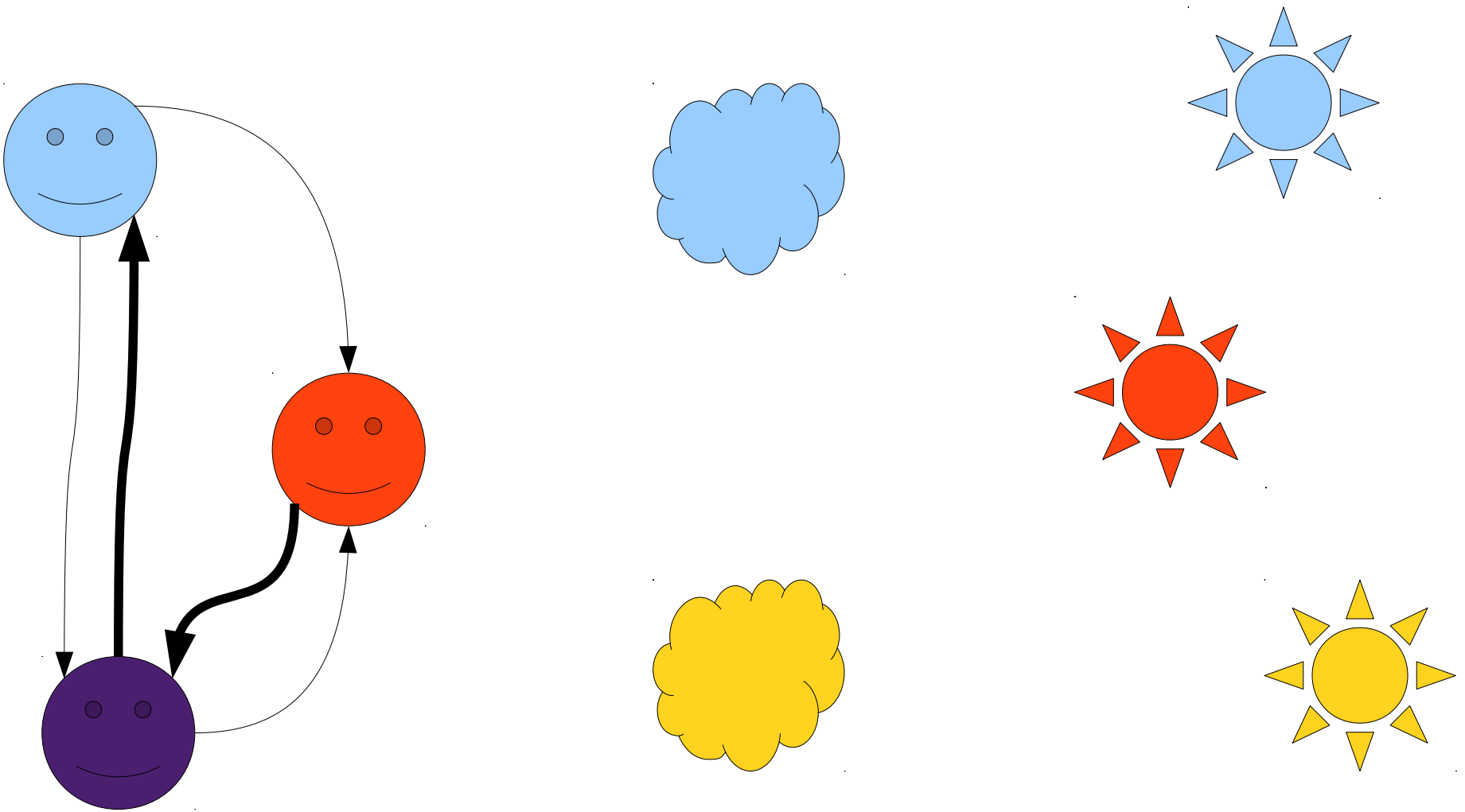
What property says this edge must be here?



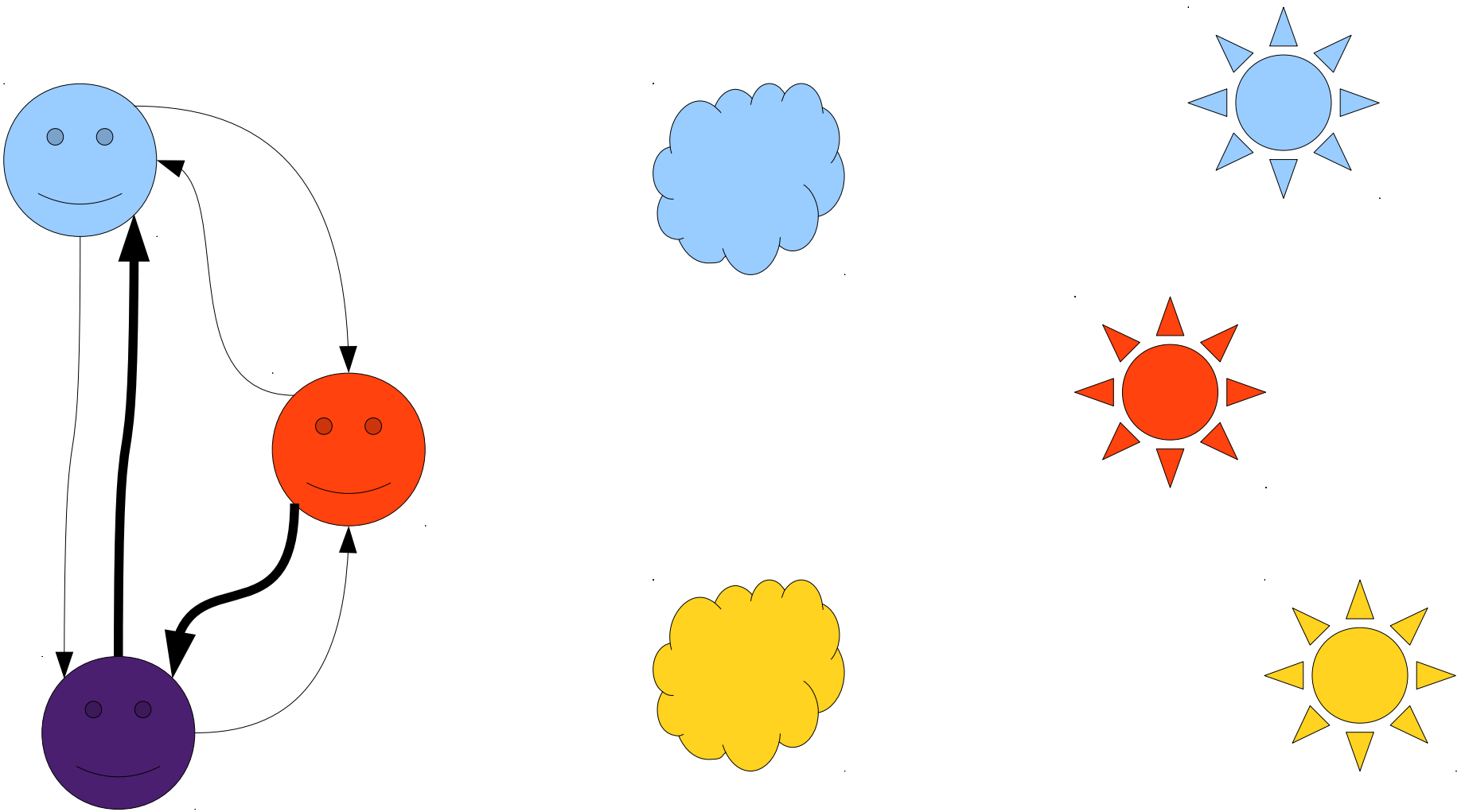
xRy ≡ x and y have the same shape.



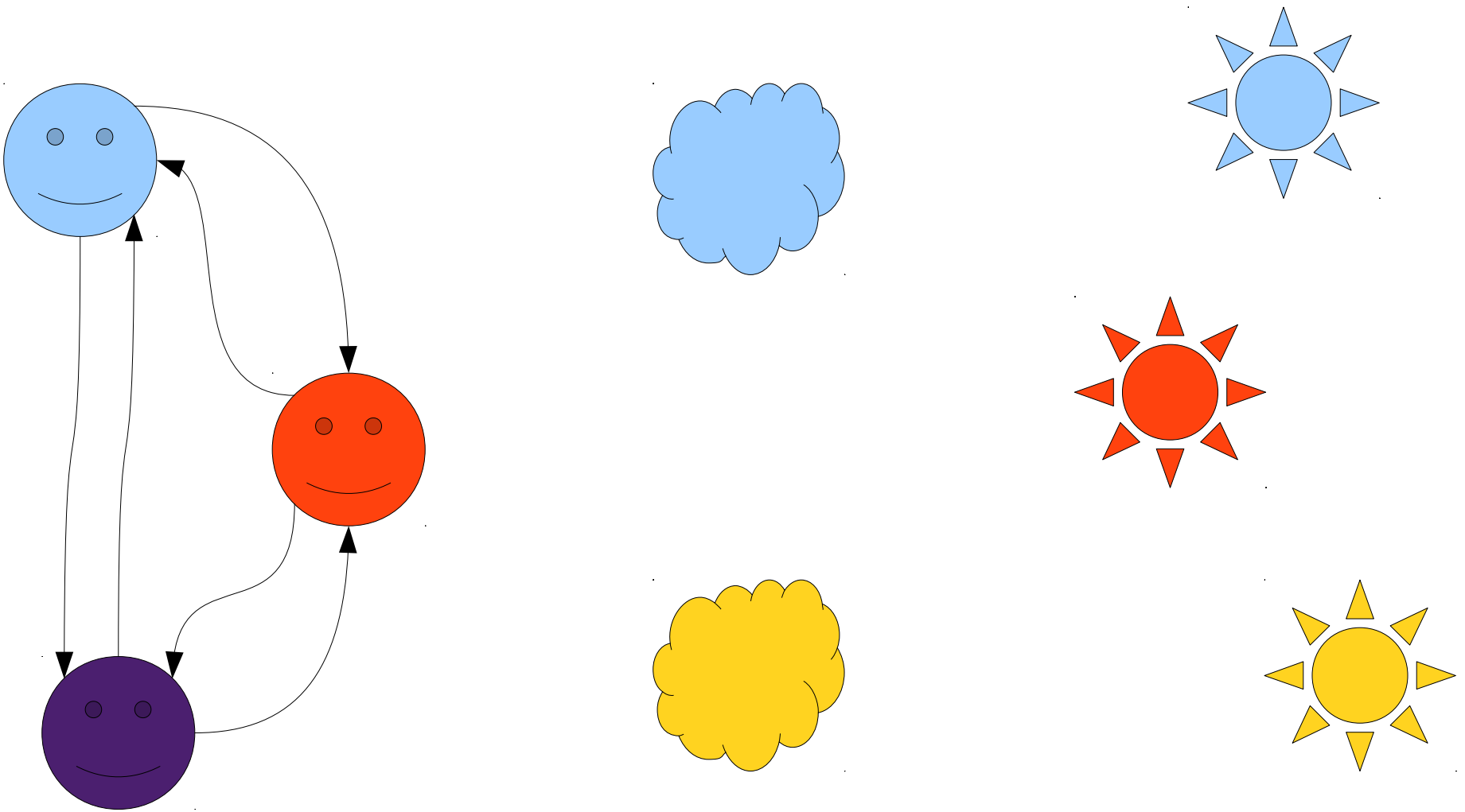
xRy ≡ x and y have the same shape.



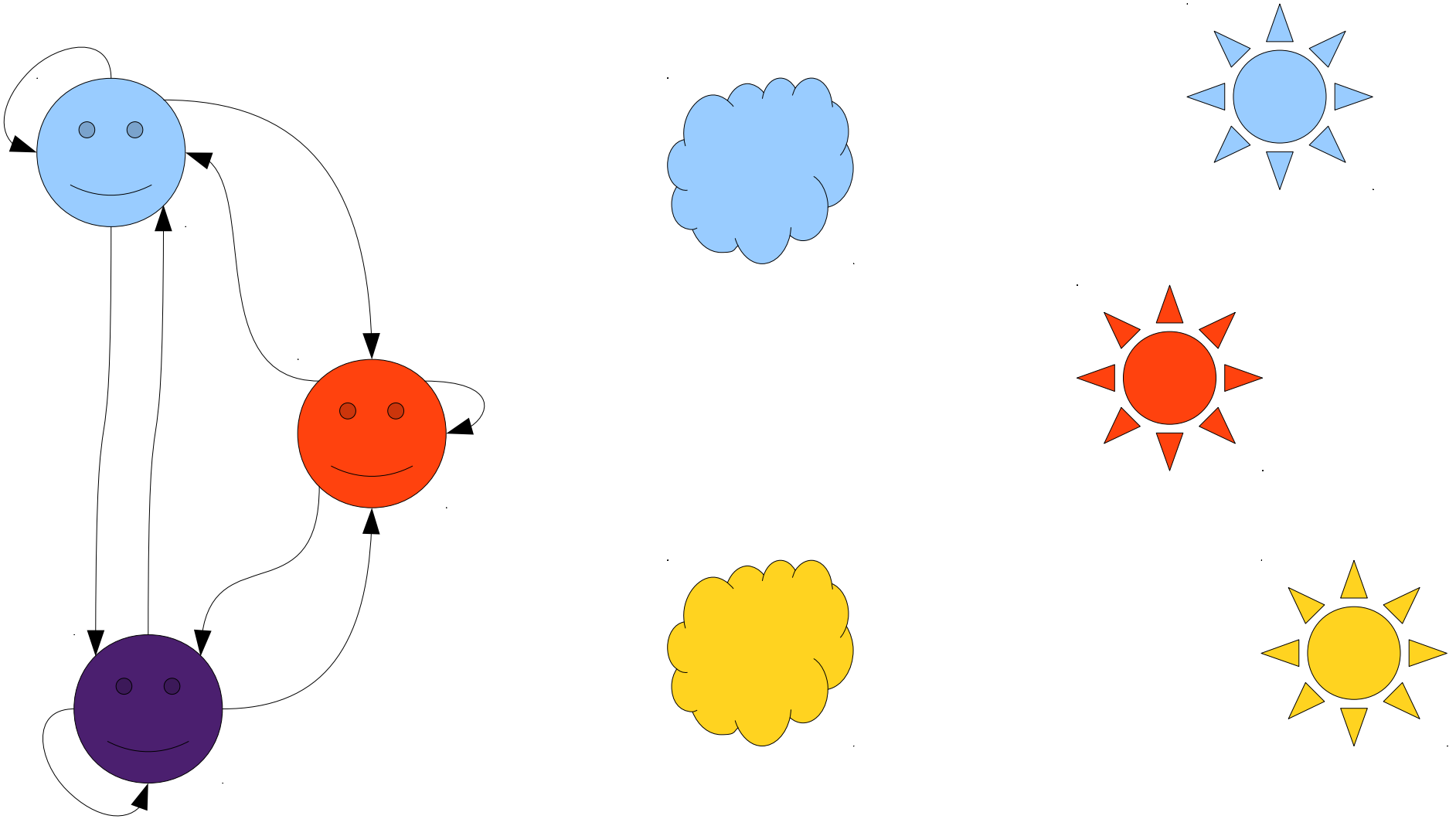
xRy ≡ x and y have the same shape.



xRy ≡ x and y have the same shape.

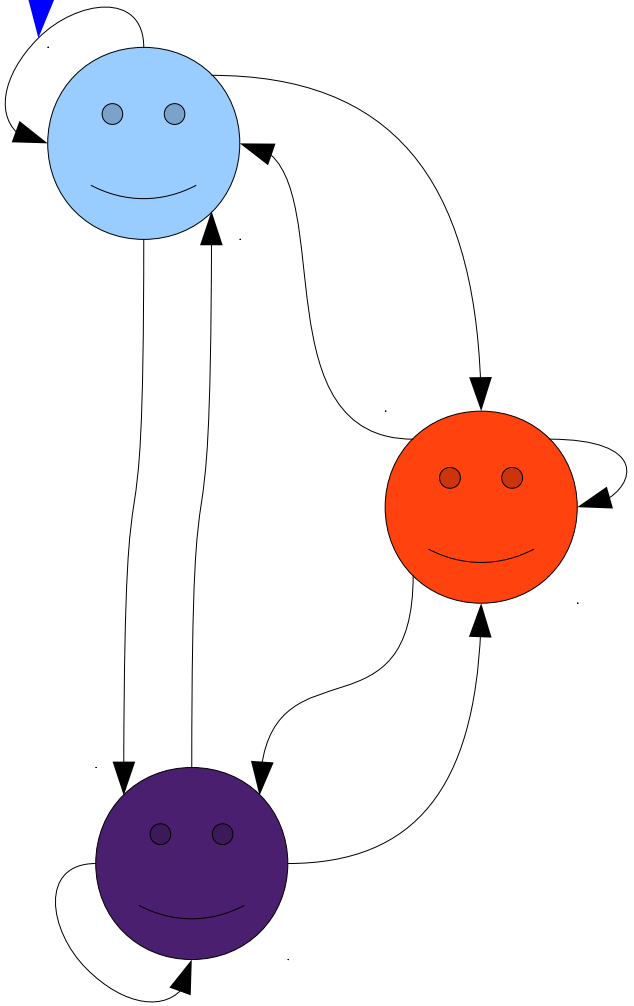


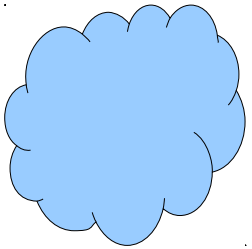
xRy ≡ x and y have the same shape.

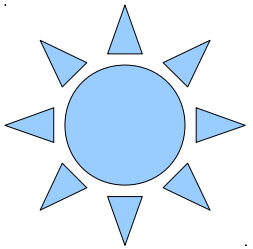


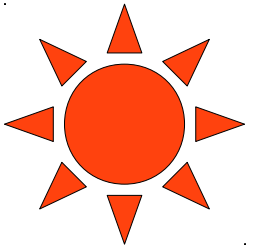
xRy ≡ x and y have the same shape.

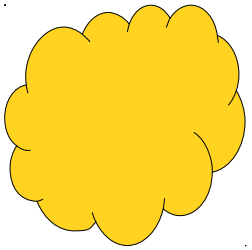
What property says these edges must be here?

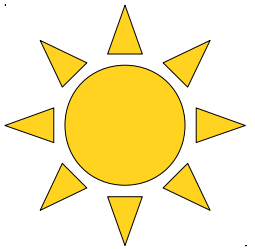




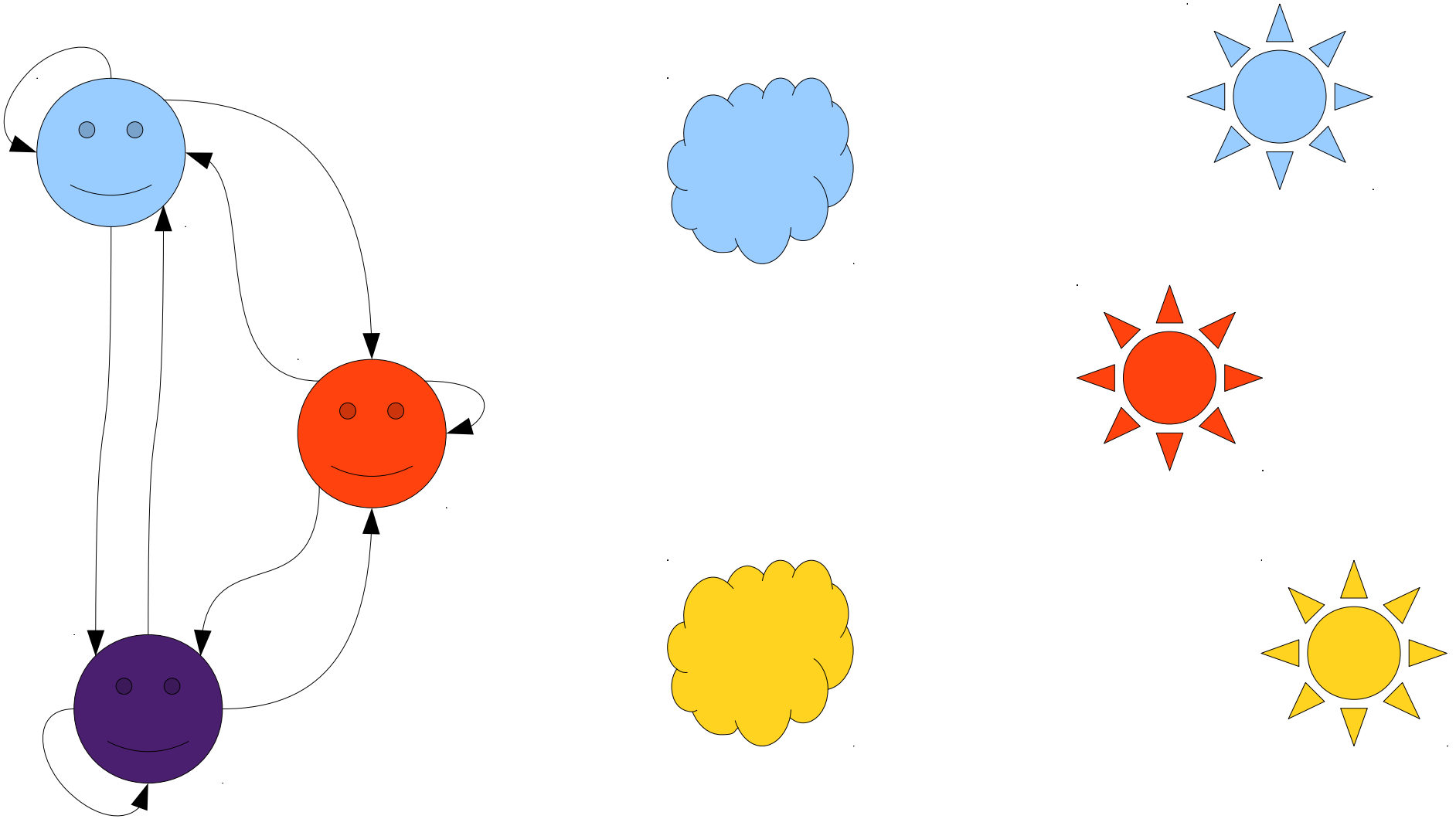




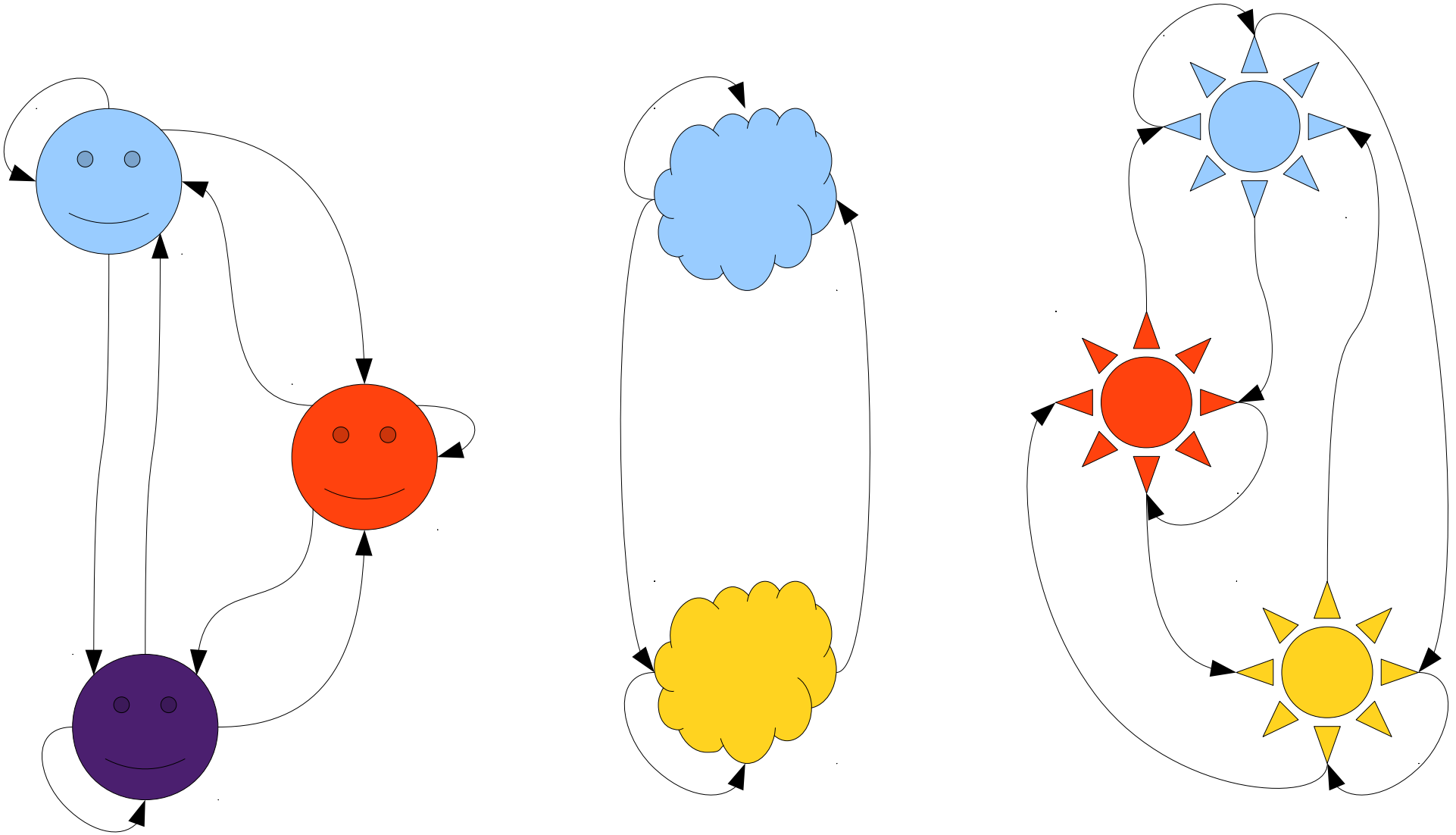




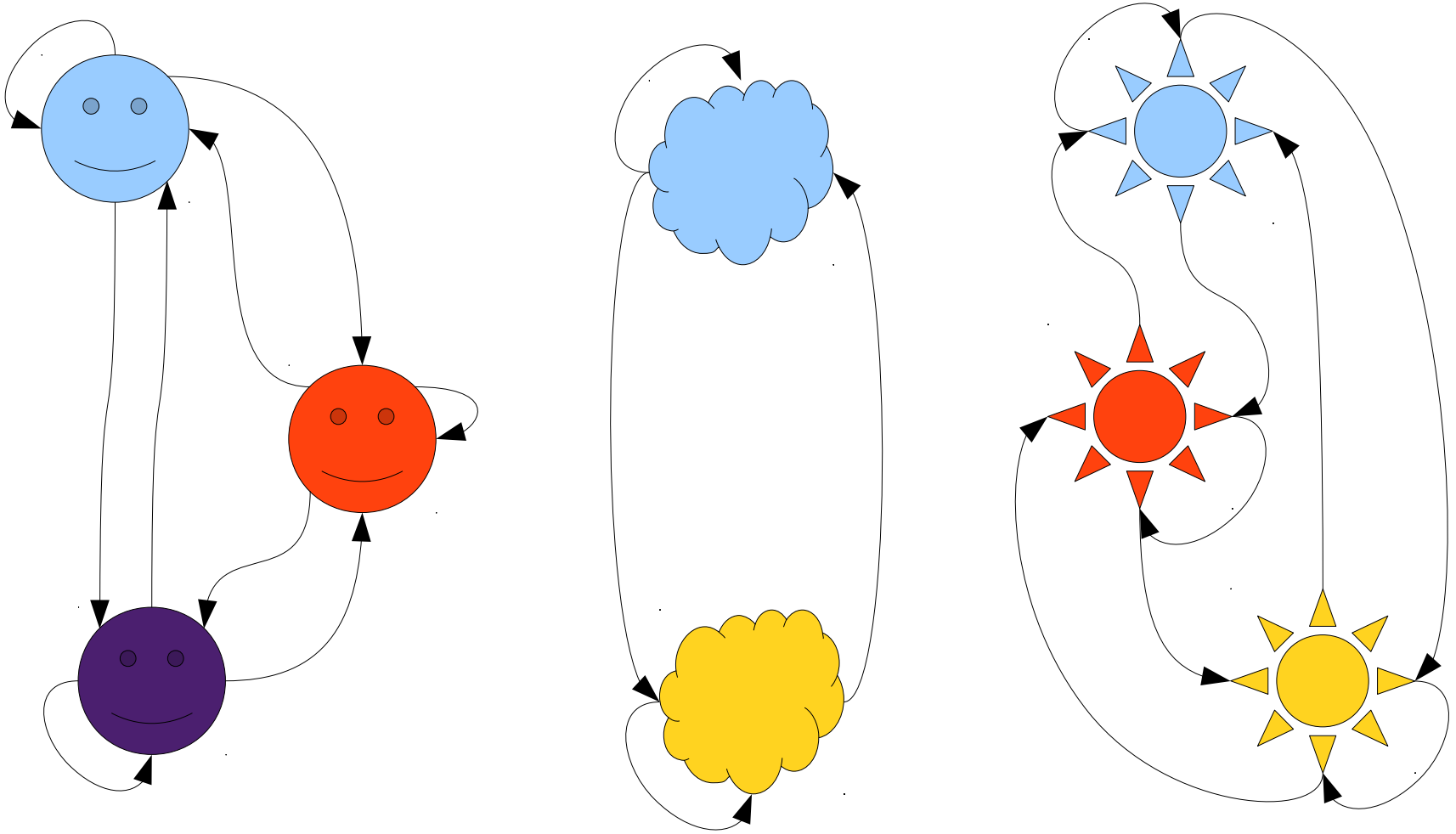
xRy ≡ x and y have the same shape.



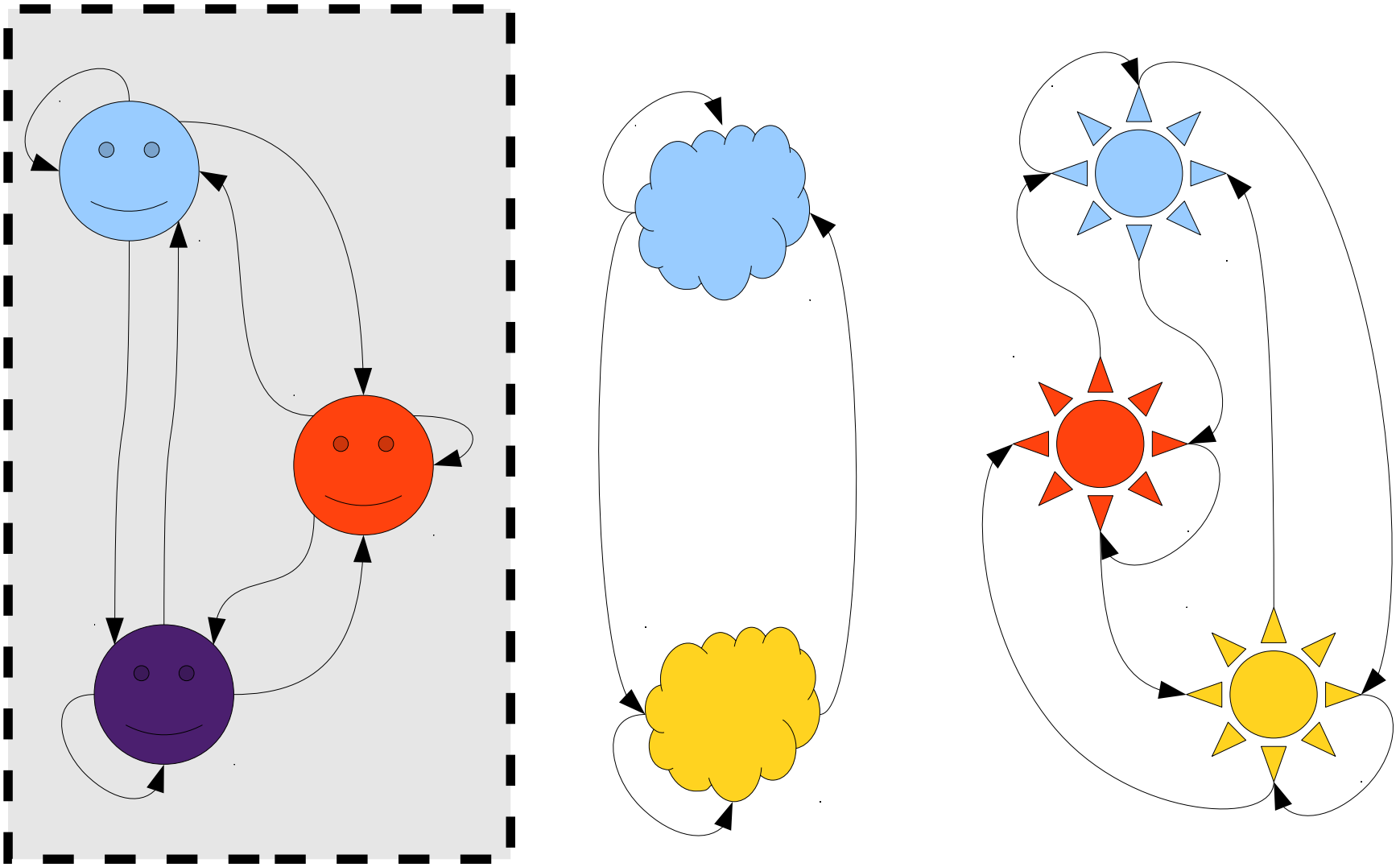
xRy ≡ x and y have the same shape.



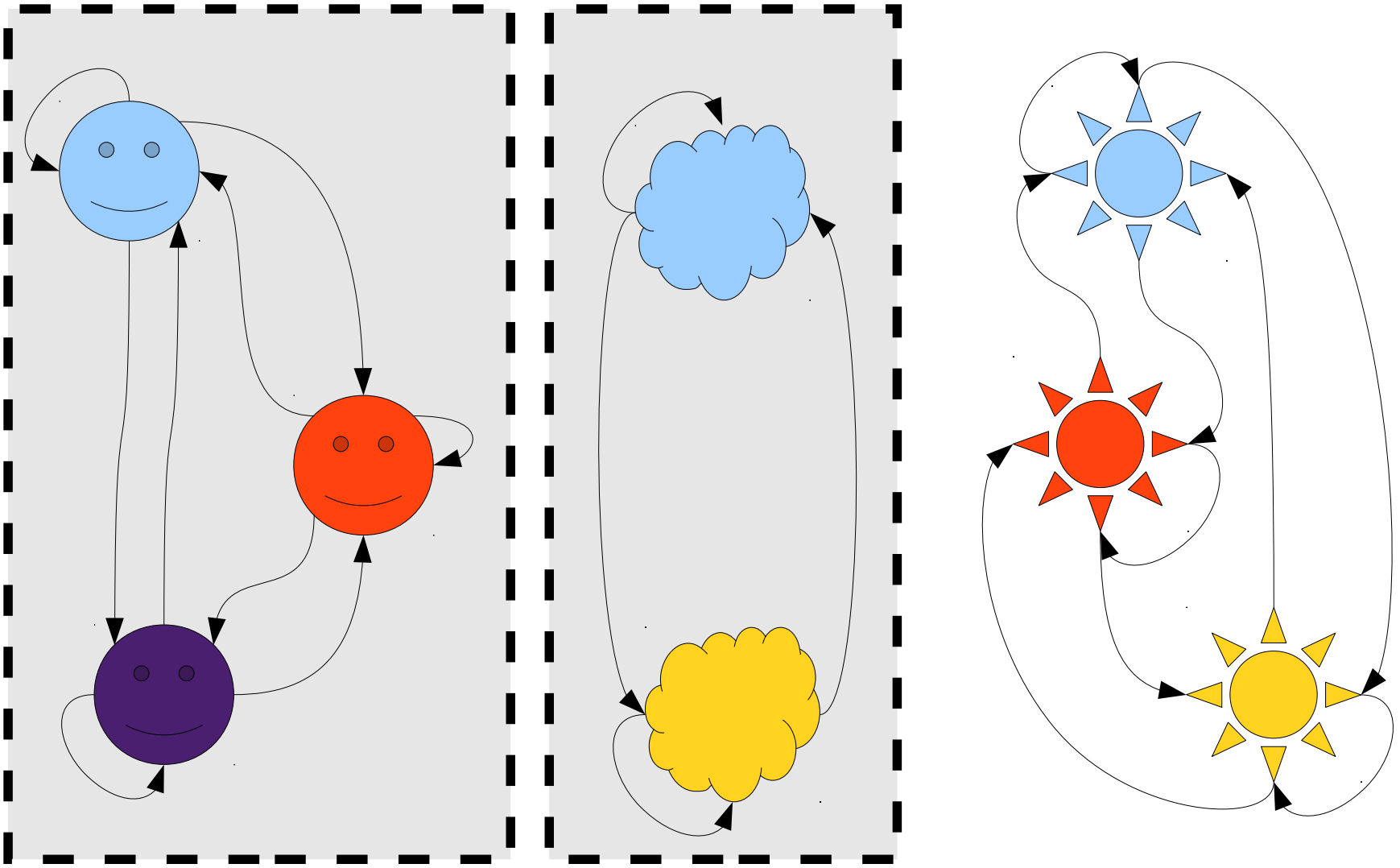
xRy ≡ x and y have the same shape.



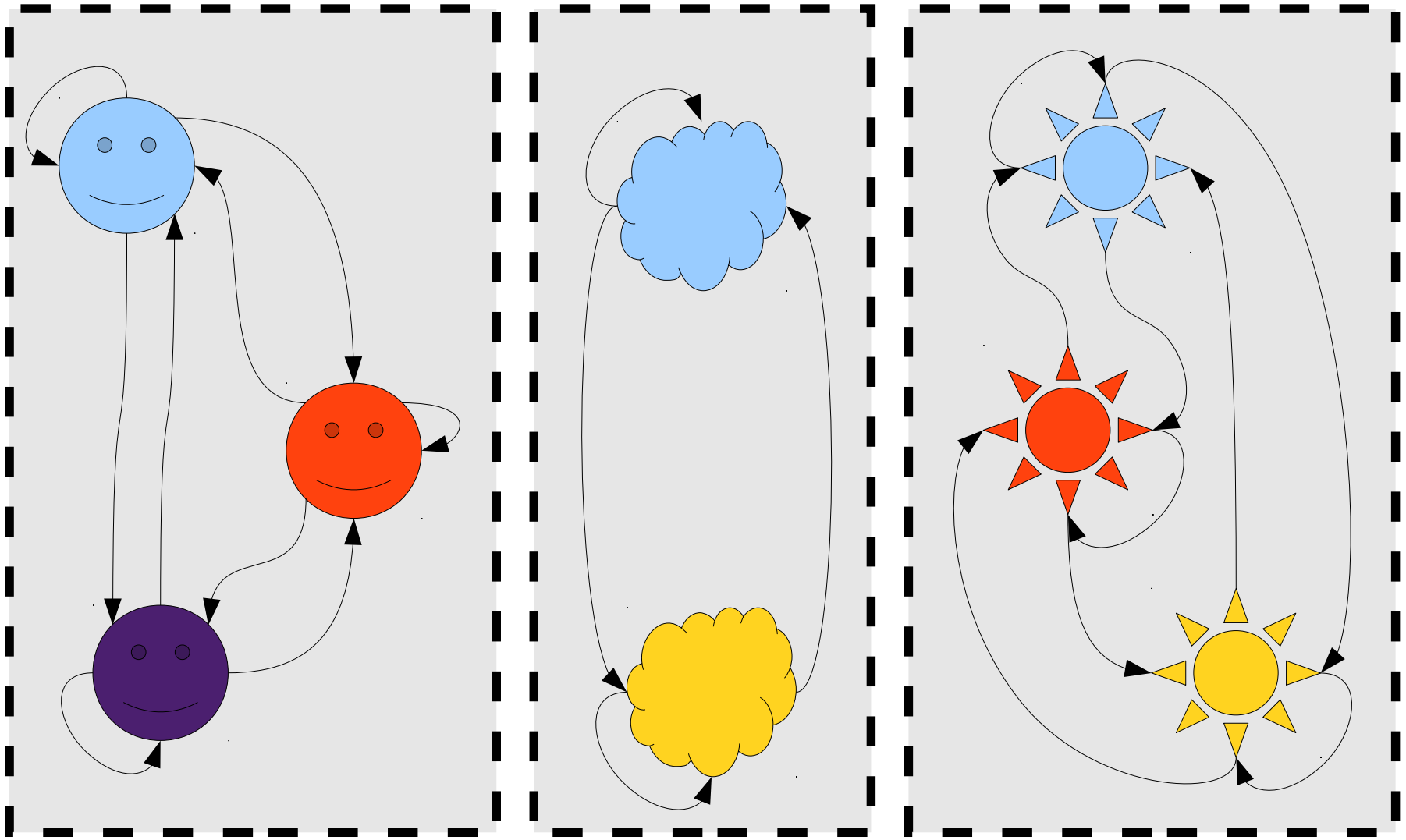
xRy ≡ x and y are the same shape.



xRy ≡ x and y are the same shape.



xRy ≡ x and y are the same shape.



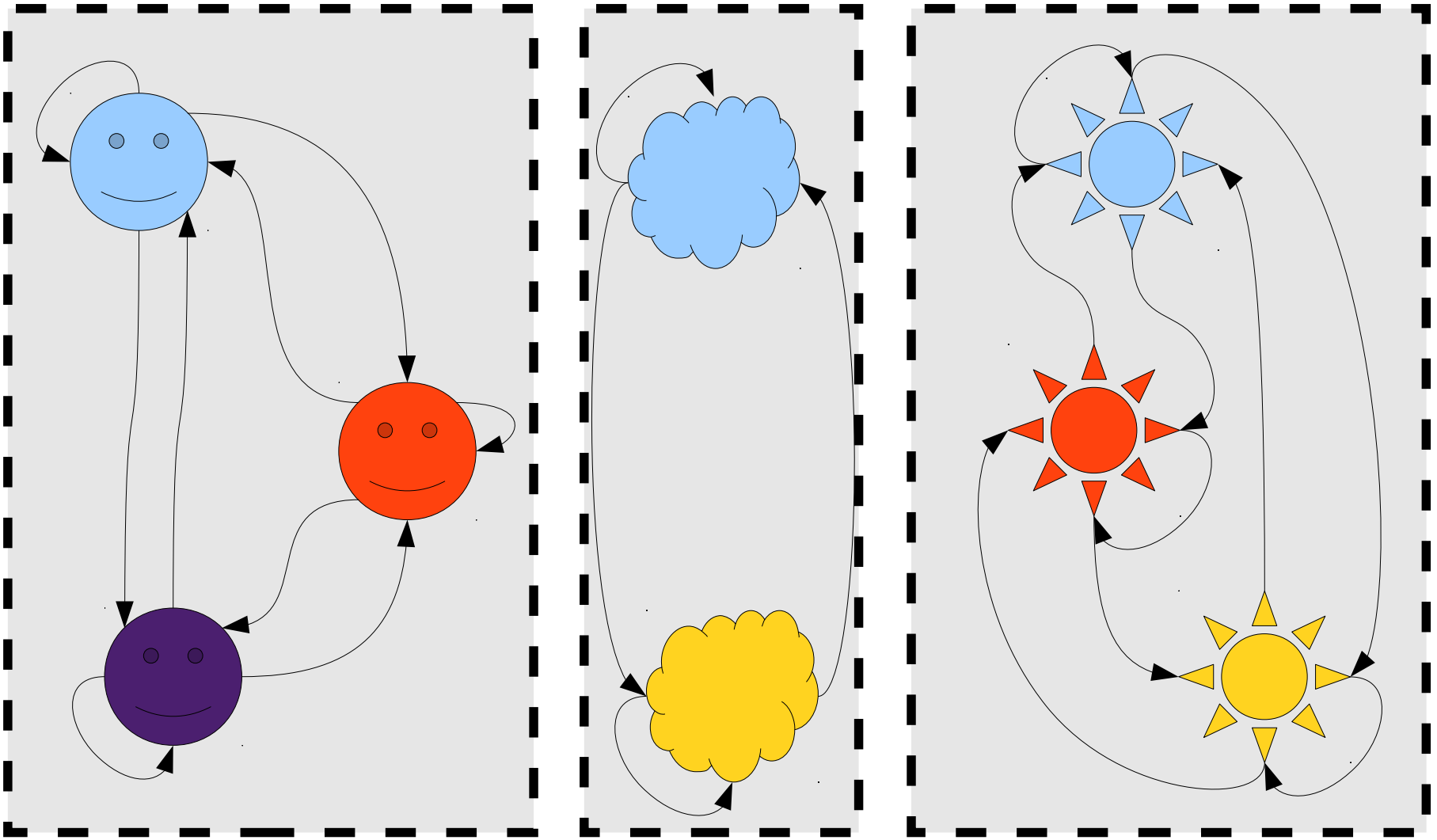
xRy ≡ x and y are the same shape.

Equivalence Classes

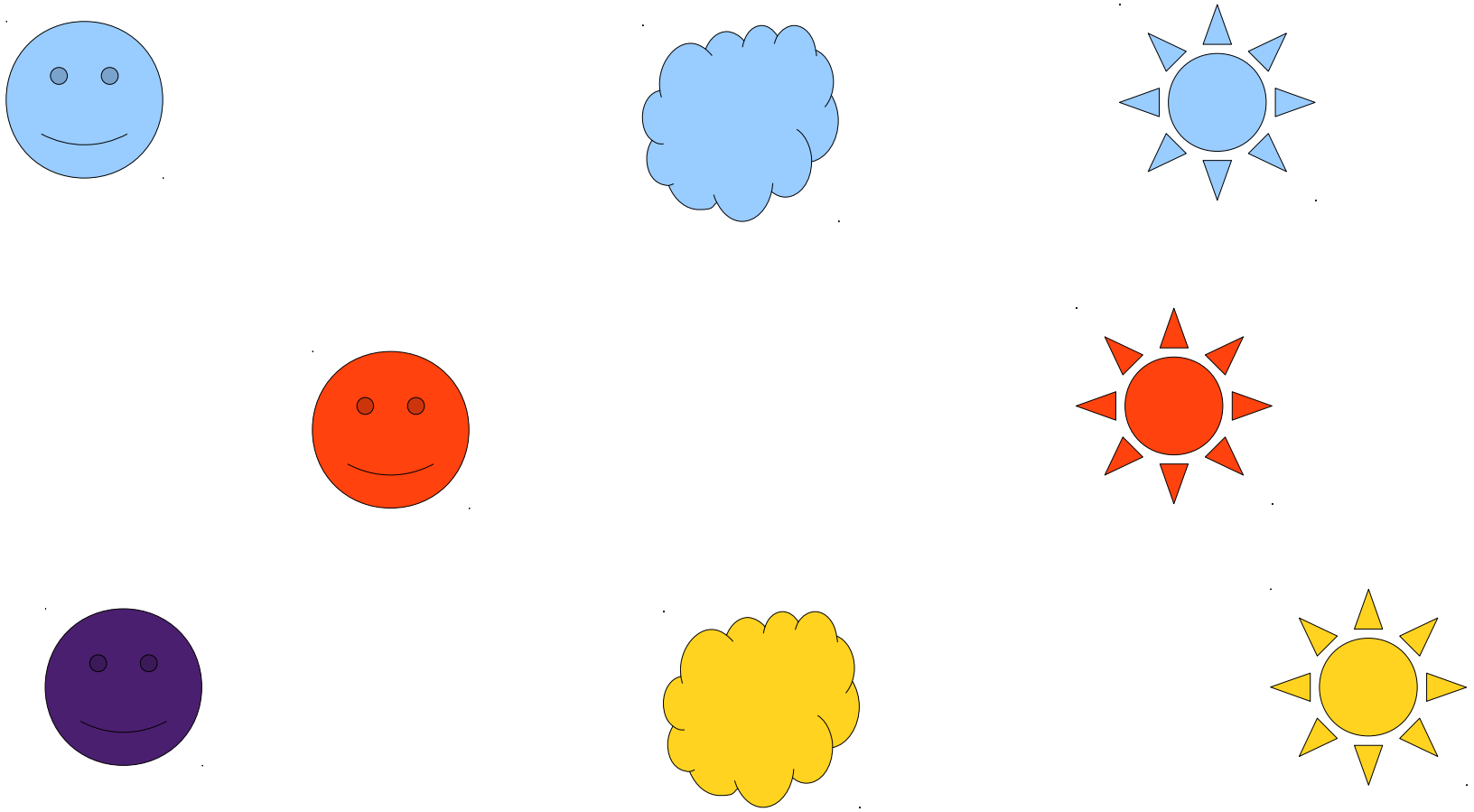
- Given an equivalence relation R over a set A, for any a ∈ A, the **equivalence class of a** is the set

$$[a]_R \equiv \{ b \mid b \in A \text{ and } aRb \}$$

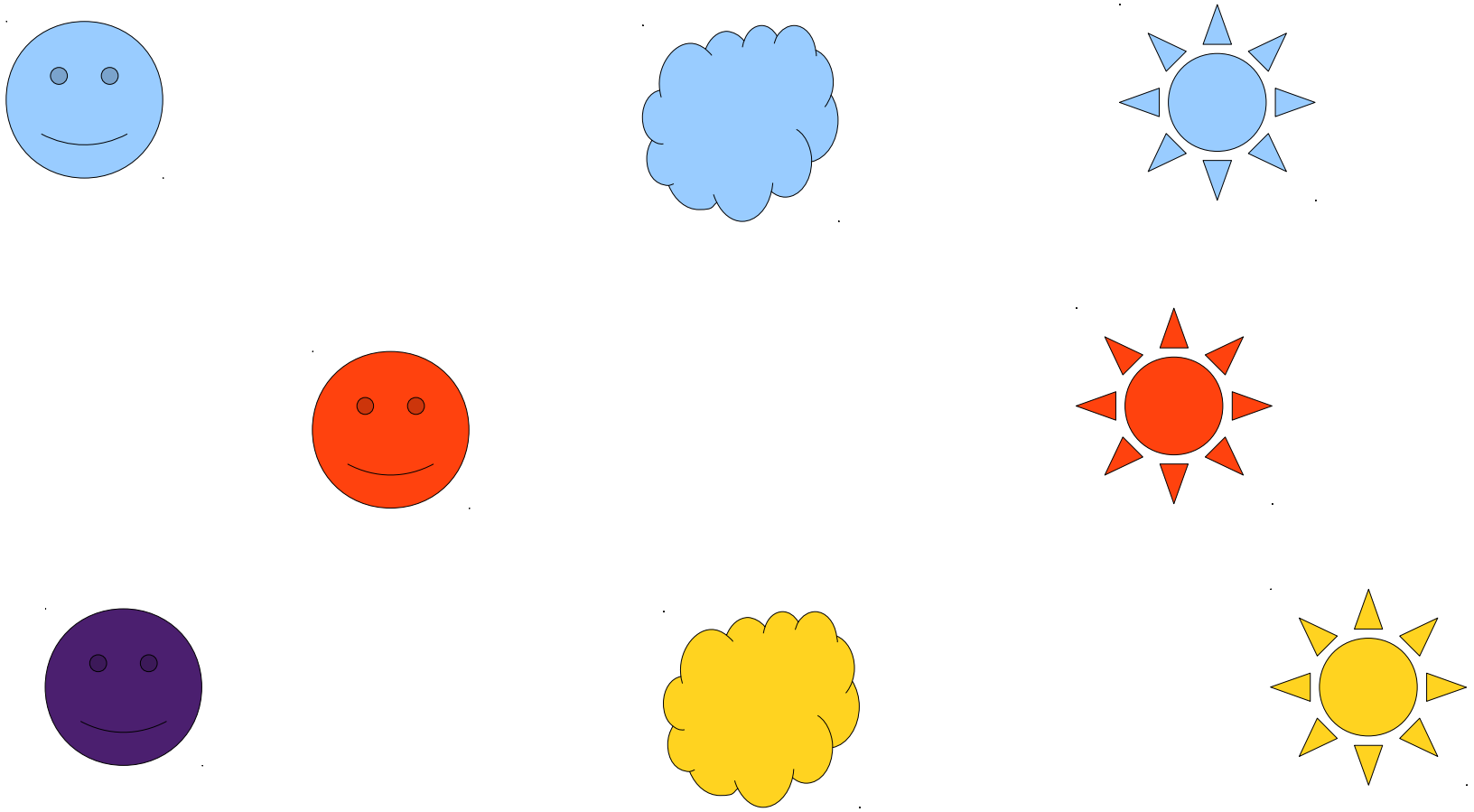
- Informally, the set of all elements equal to a.
- R **partitions** the set A into a set of equivalence classes.



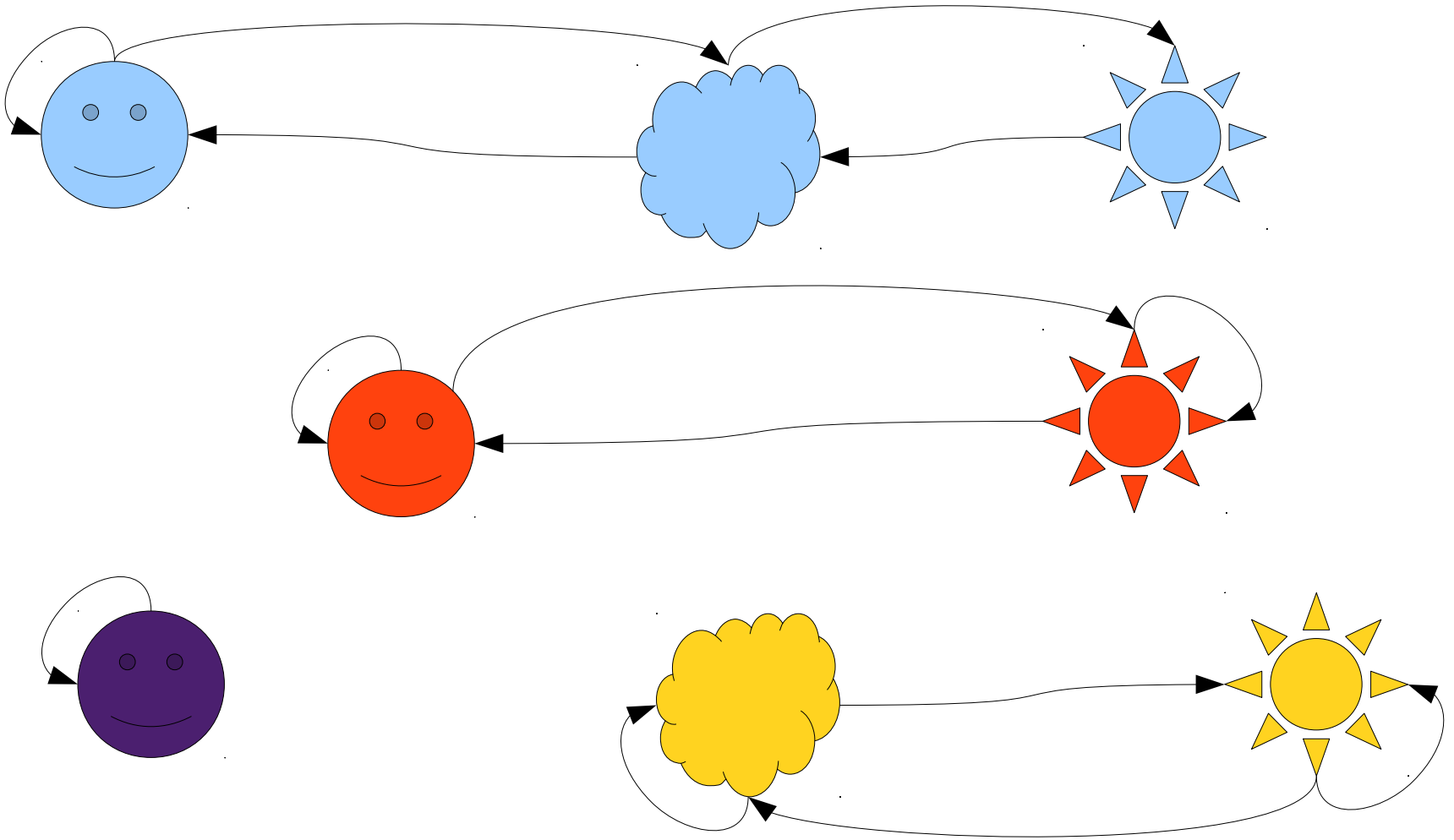
xRy ≡ x and y are the same shape.



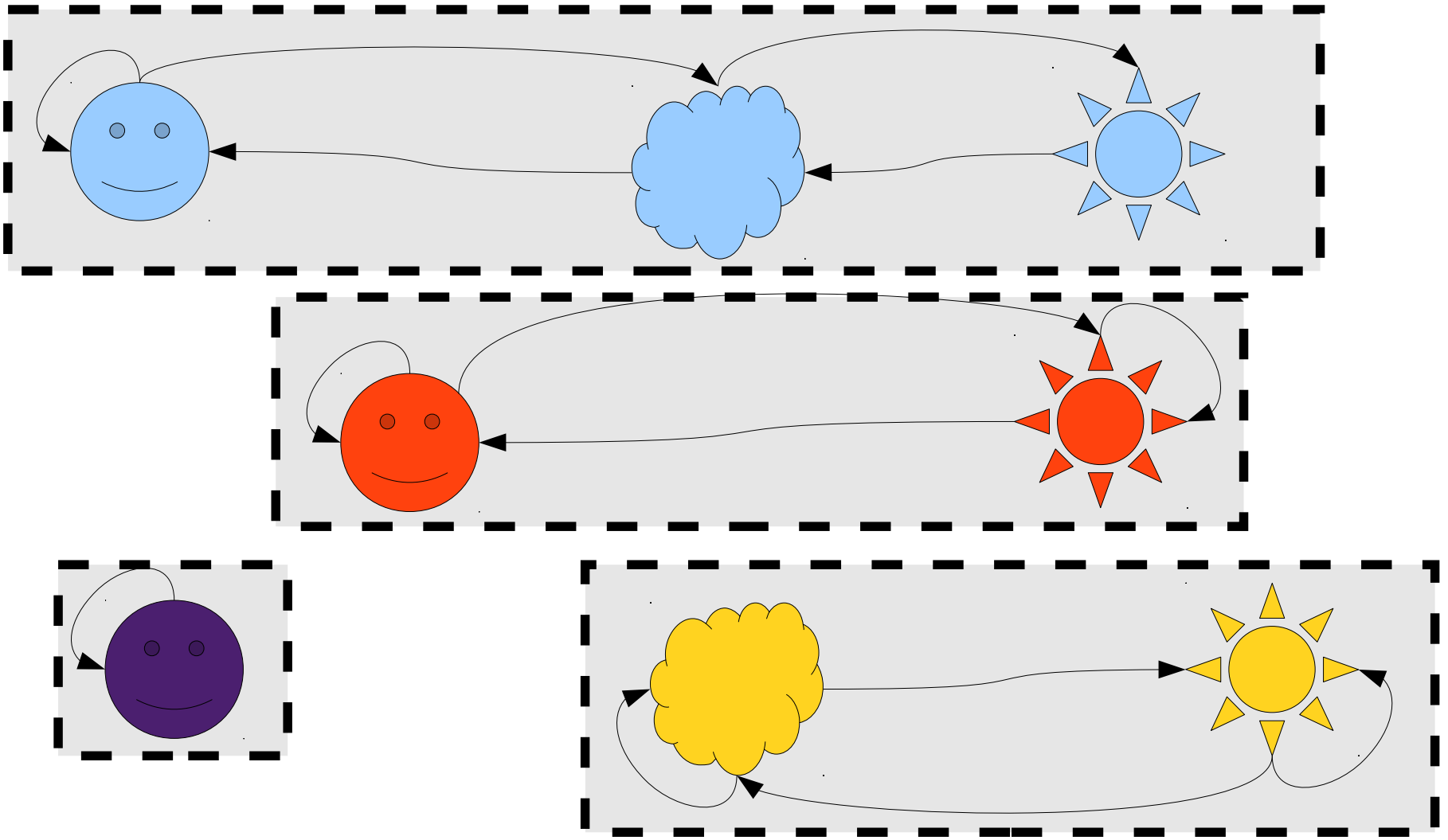
xRy ≡ x and y are the same shape.



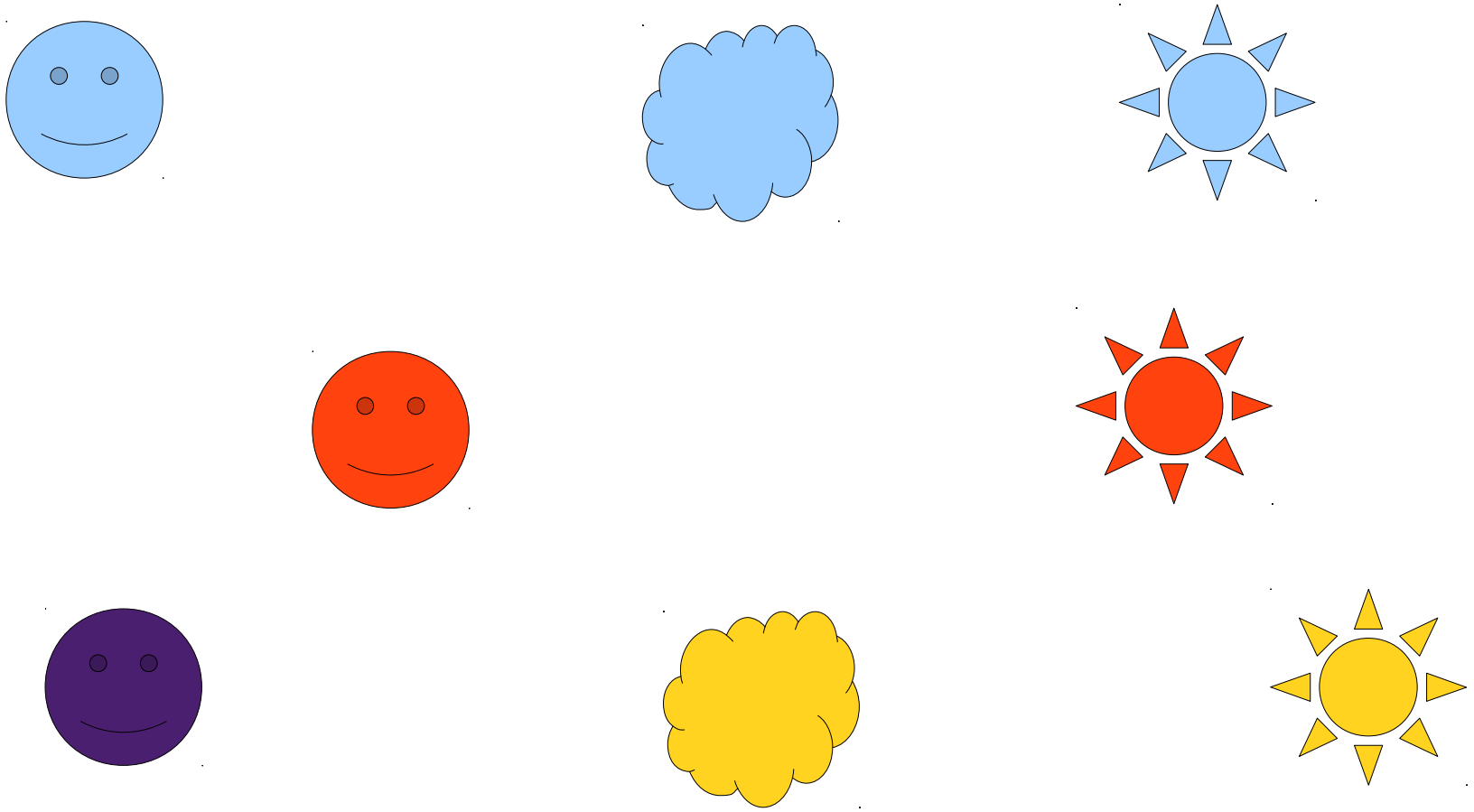
xRy ≡ x and y are the same **color**.



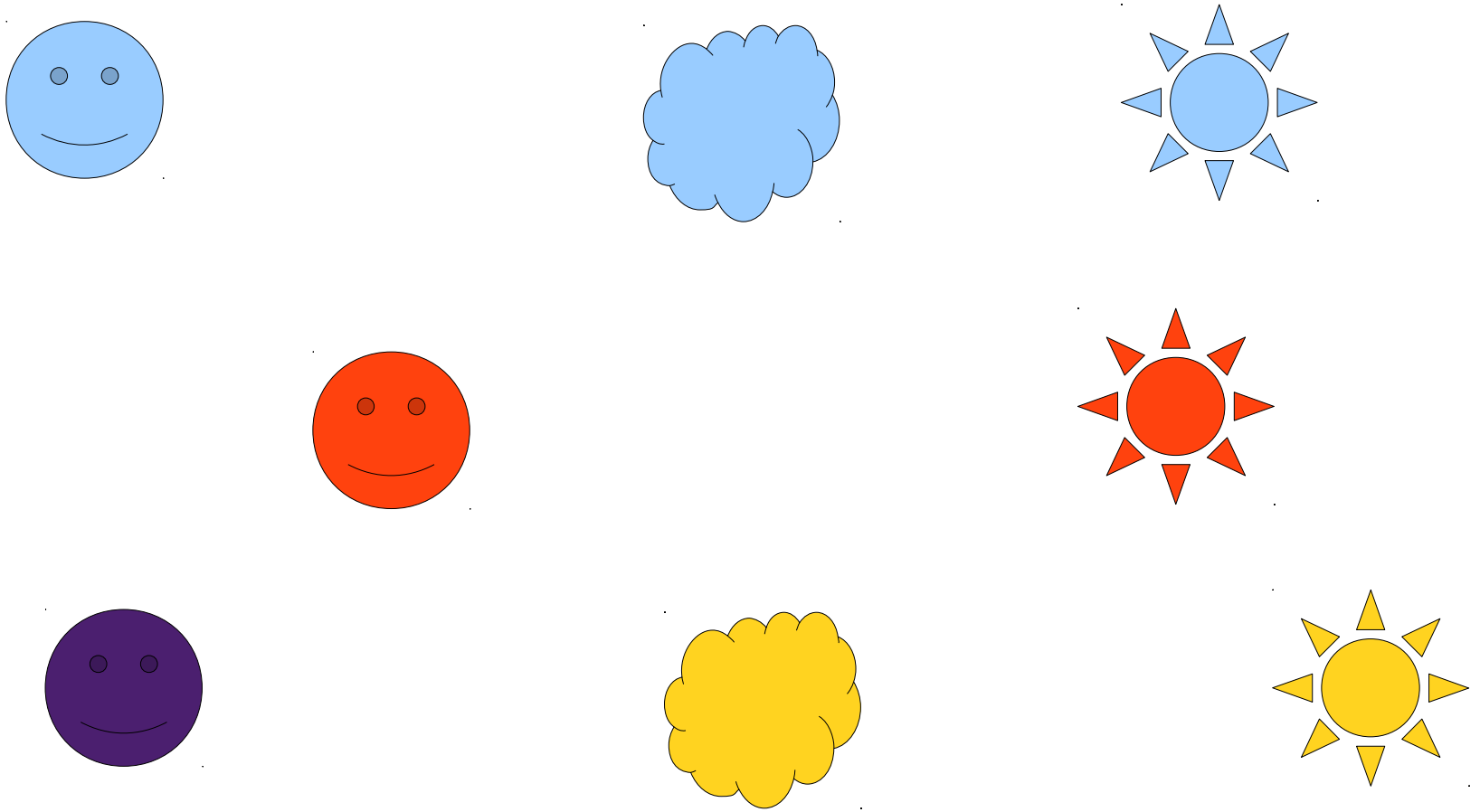
xRy ≡ x and y are the same **color**.



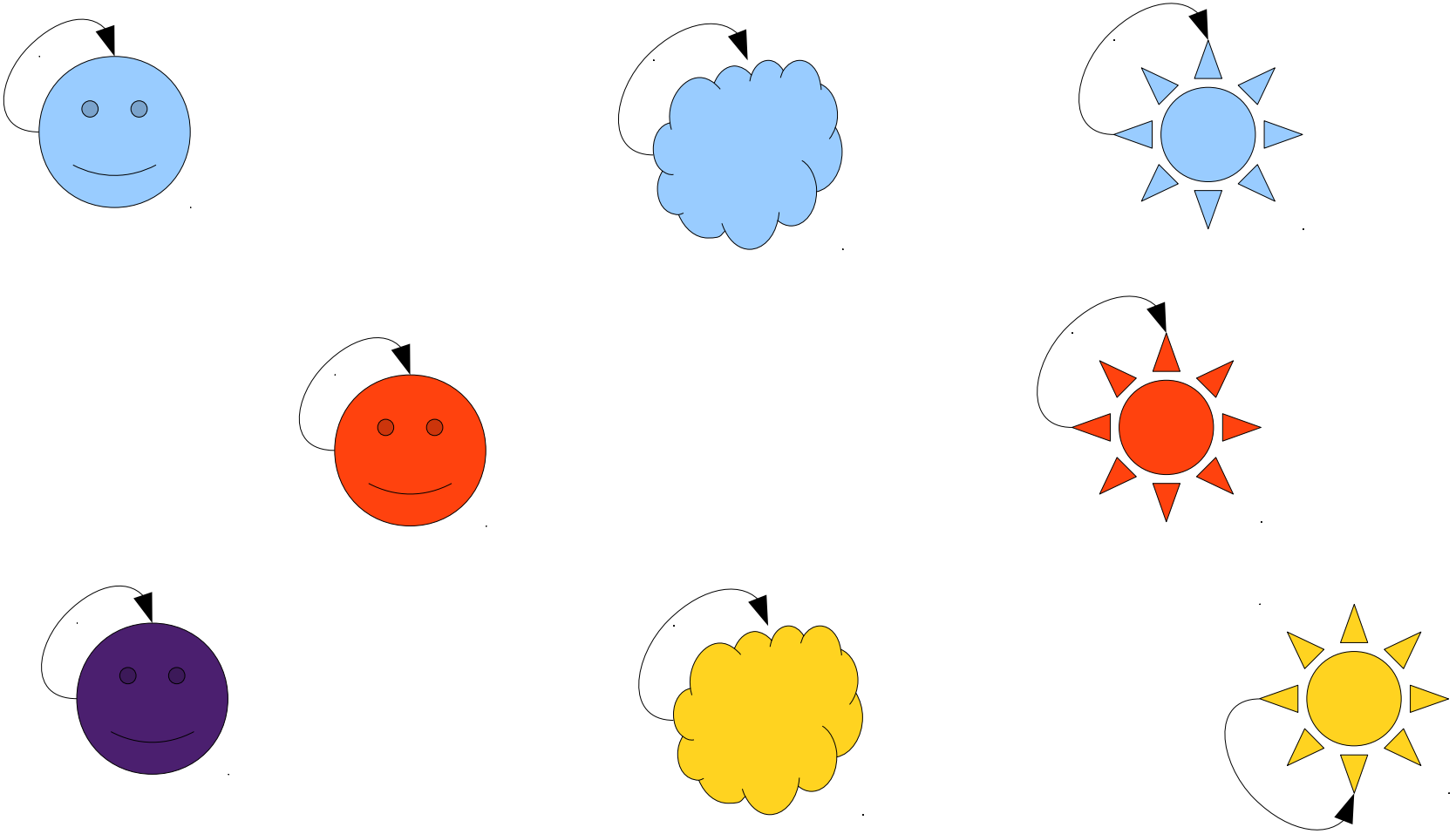
xRy ≡ x and y are the same **color**.



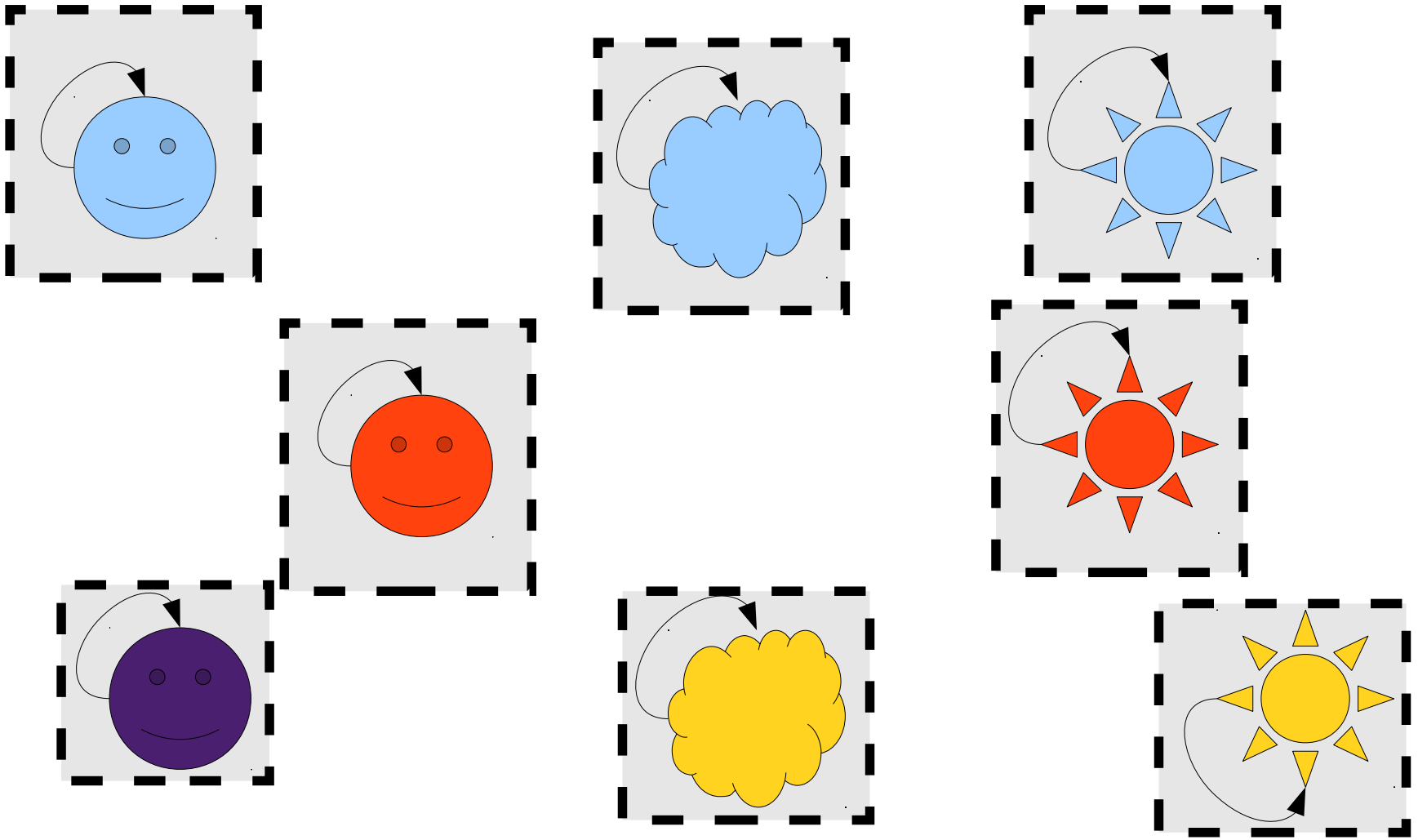
xRy ≡ x and y are the same **color**.



$$xRy \equiv x = y$$



$$xRy \equiv x = y$$



$xRy \equiv x = y$

Order Relations

“x is larger than y”

“x is tastier than y”

“x runs faster than y”

“x is a subset of y”

“x divides y”

“x is a part of y”

Informally

An **order relation** is a relation that ranks elements against one another.

Again, do not use this definition in proofs!
It's just an intuition!

Properties of Order Relations

$x \leq y$

Properties of Order Relations

$x \leq y$

$1 \leq 5$ **and** $5 \leq 8$

Properties of Order Relations

$x \leq y$

$1 \leq 5$ **and** $5 \leq 8$

$1 \leq 8$

Properties of Order Relations

$x \leq y$

$42 \leq 99$ **and** $99 \leq 137$

Properties of Order Relations

$x \leq y$

$42 \leq 99$ **and** $99 \leq 137$

$42 \leq 137$

Properties of Order Relations

$x \leq y$

$x \leq y$ **and** $y \leq z$

Properties of Order Relations

$x \leq y$

$x \leq y$ **and** $y \leq z$

$x \leq z$

Properties of Order Relations

$x \leq y$

$x \leq y$ **and** $y \leq z$

$x \leq z$

Transitivity

Properties of Order Relations

$x \leq y$

Properties of Order Relations

$x \leq y$

$1 \leq 1$

Properties of Order Relations

$x \leq y$

$42 \leq 42$

Properties of Order Relations

$x \leq y$

$137 \leq 137$

Properties of Order Relations

$x \leq y$

$x \leq x$

Properties of Order Relations

$x \leq y$

$x \leq x$

Reflexivity

Properties of Order Relations

$x \leq y$

Properties of Order Relations

$x \leq y$

$19 \leq 21$

Properties of Order Relations

$x \leq y$

$19 \leq 21$

$21 \leq 19?$

Properties of Order Relations

$x \leq y$

$19 \leq 21$

~~$21 \leq 19$~~?

Properties of Order Relations

$x \leq y$

$42 \leq 137$

Properties of Order Relations

$x \leq y$

$42 \leq 137$

$137 \leq 42?$

Properties of Order Relations

$x \leq y$

$42 \leq 137$

~~$137 \leq 42$~~?

Properties of Order Relations

$x \leq y$

$137 \leq 137$

Properties of Order Relations

$x \leq y$

$137 \leq 137$

$137 \leq 137?$

Properties of Order Relations

$x \leq y$

$137 \leq 137$

$137 \leq 137$

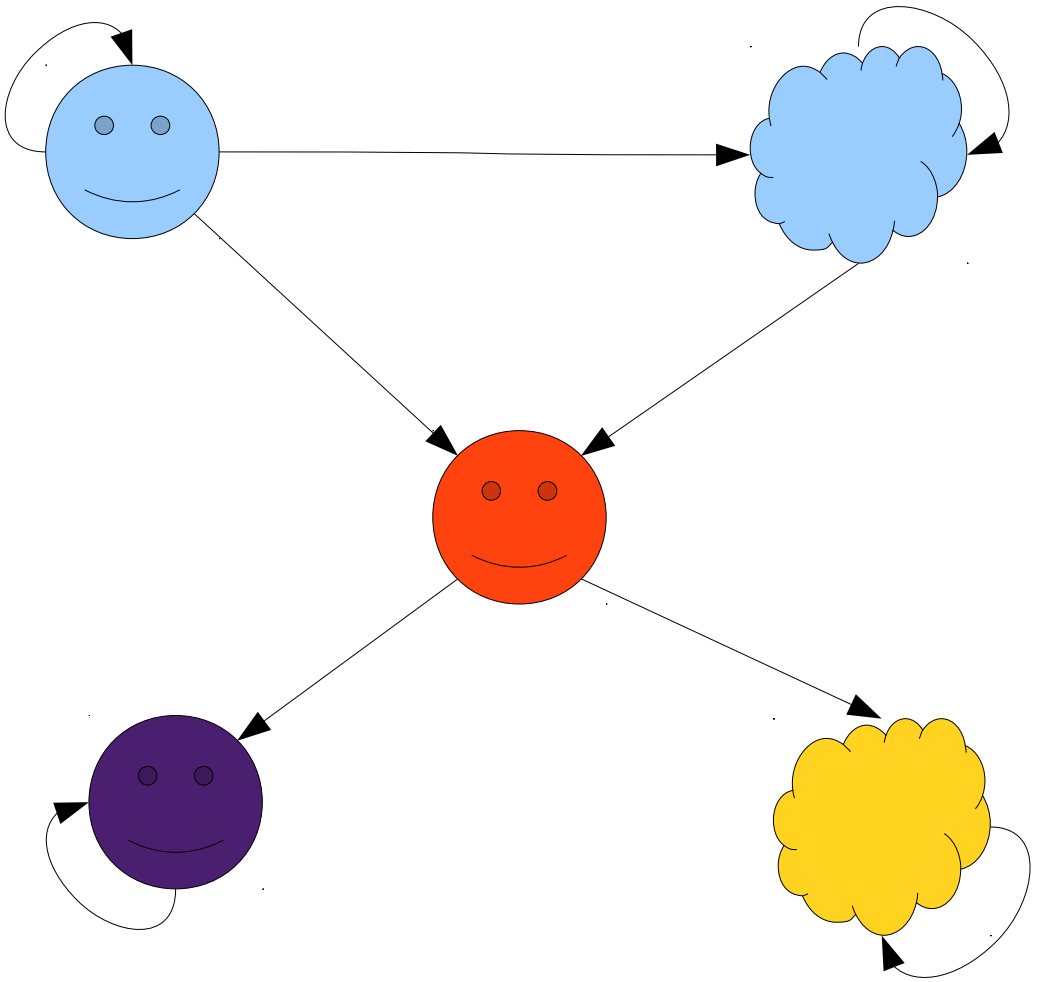
Antisymmetry

A binary relation R is called **antisymmetric** if whenever xRy and yRx, x = y.

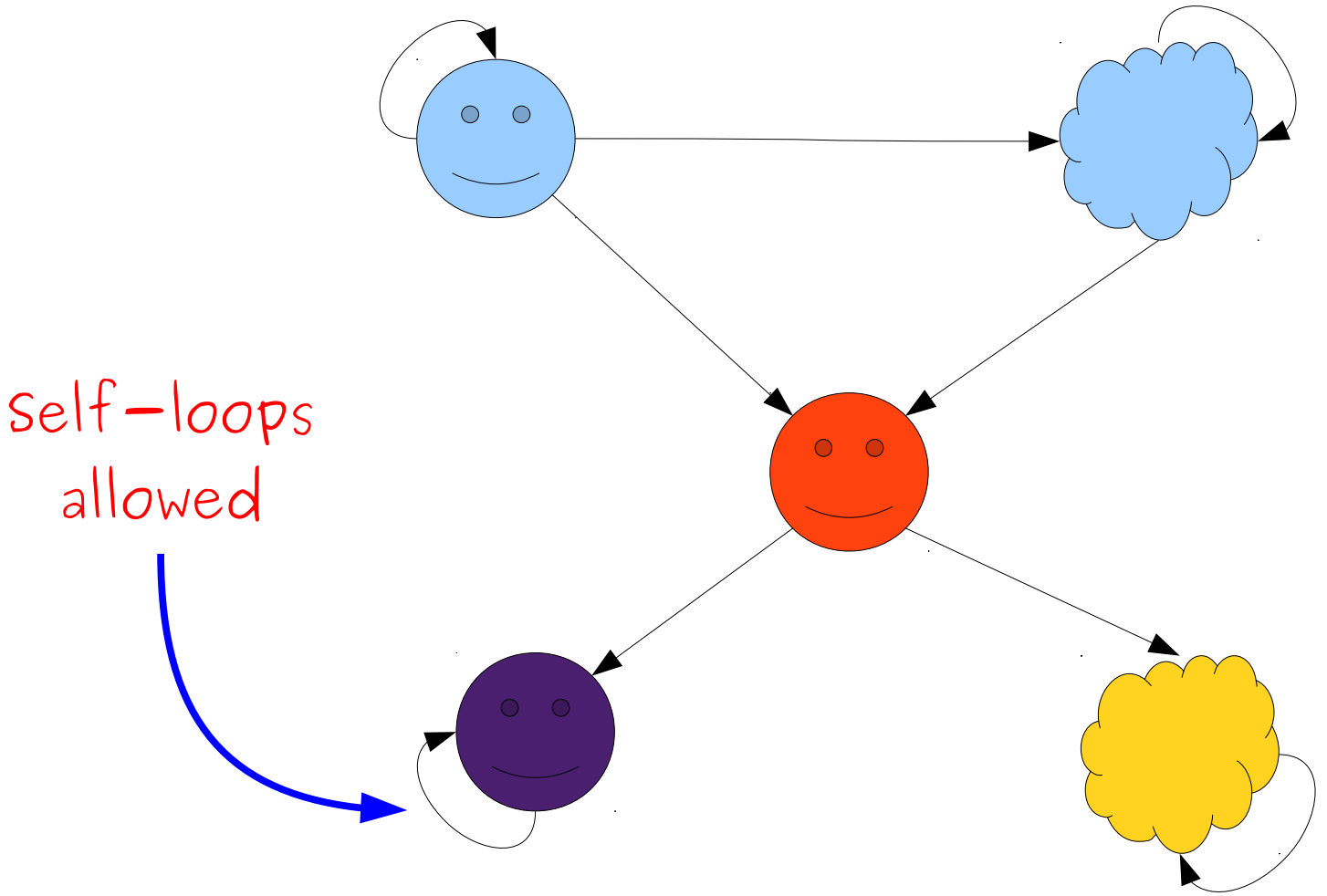
Equivalently: Whenever xRy and x ≠ y, y$\not R$x

An Intuition for Antisymmetry

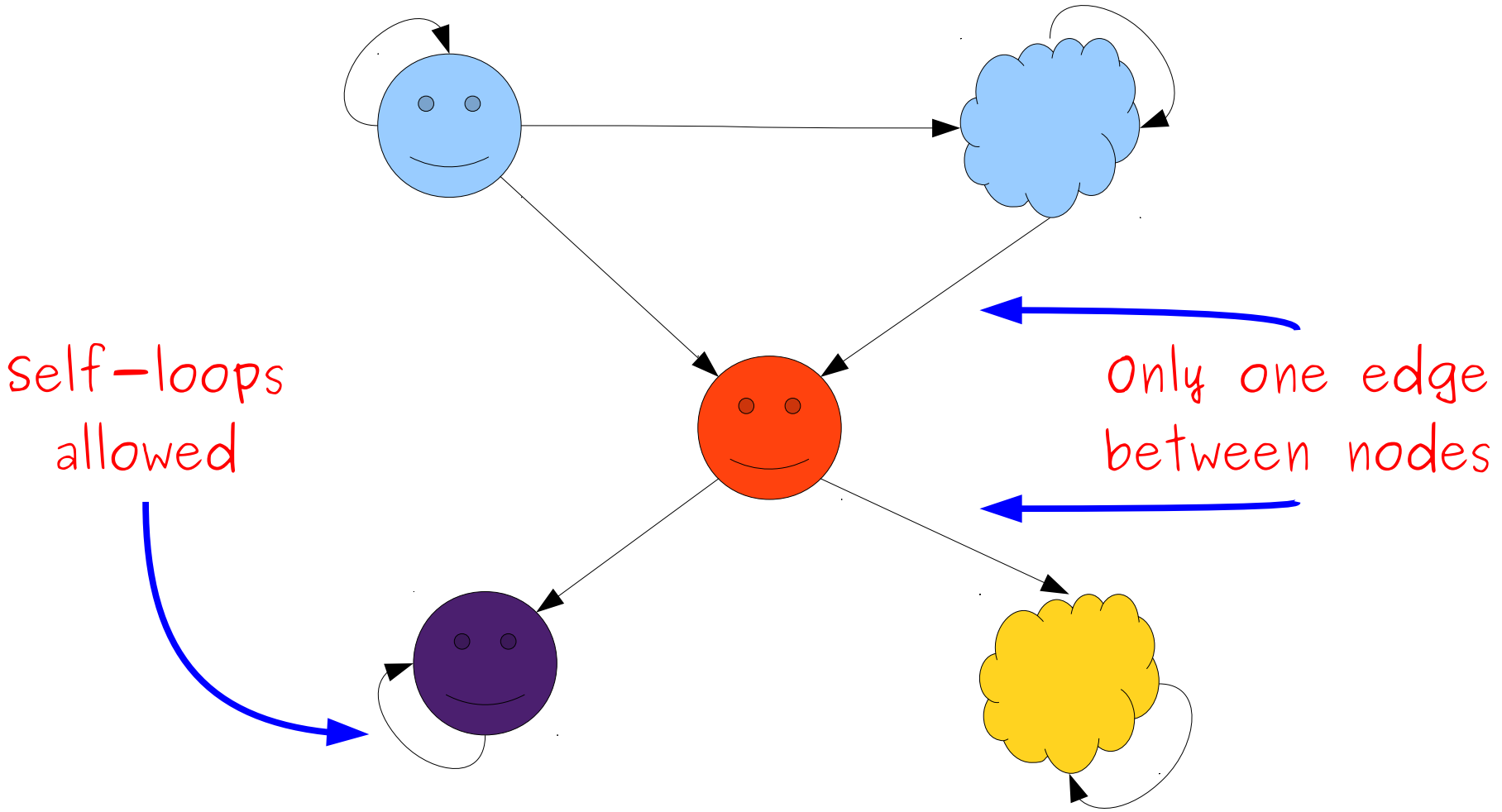
An Intuition for Antisymmetry



An Intuition for Antisymmetry



An Intuition for Antisymmetry



An Important Detail

- A binary relation R is **antisymmetric** if whenever xRy and yRx, x = y.
- Is the relation < antisymmetric?

An Important Detail

- A binary relation R is **antisymmetric** if whenever xRy and yRx, x = y.
- Is the relation < antisymmetric?
- **Yes**: By vacuous truth.
  - It is never possible for both x < y and y < x to hold.
  - Thus it's vacuously true that if x < y and y < x, x = y.

Partial Orders

- A binary relation R is a **partial order** if it is
  - **reflexive**,
  - **antisymmetric**, and
  - **transitive**
- A pair (S, R), where R is a partial order over S, is called a **partially ordered set** or **poset**.

Partial Orders

- A binary relation R is a **partial order** if it is
  - **reflexive**,
  - **antisymmetric**, and
  - **transitive**
- A pair (S, R), where R is a partial order over S, is called a **partially ordered set** or **poset**.

Why "partial"?

2008 Summer Olympics

Gold	Silver	Bronze	Total
51	21	28	100
36	38	36	110
23	21	28	72
19	13	15	47
14	15	17	46

Inspired by http://tartarus.org/simon/2008-olympics-hasse/.
Data from http://news.bbc.co.uk/sport2/hi/olympics/medals_table/default.stm

2008 Summer Olympics

Gold	Silver	Bronze	Total
51	21	28	100
36	38	36	110
23	21	28	72
19	13	15	47
14	15	17	46

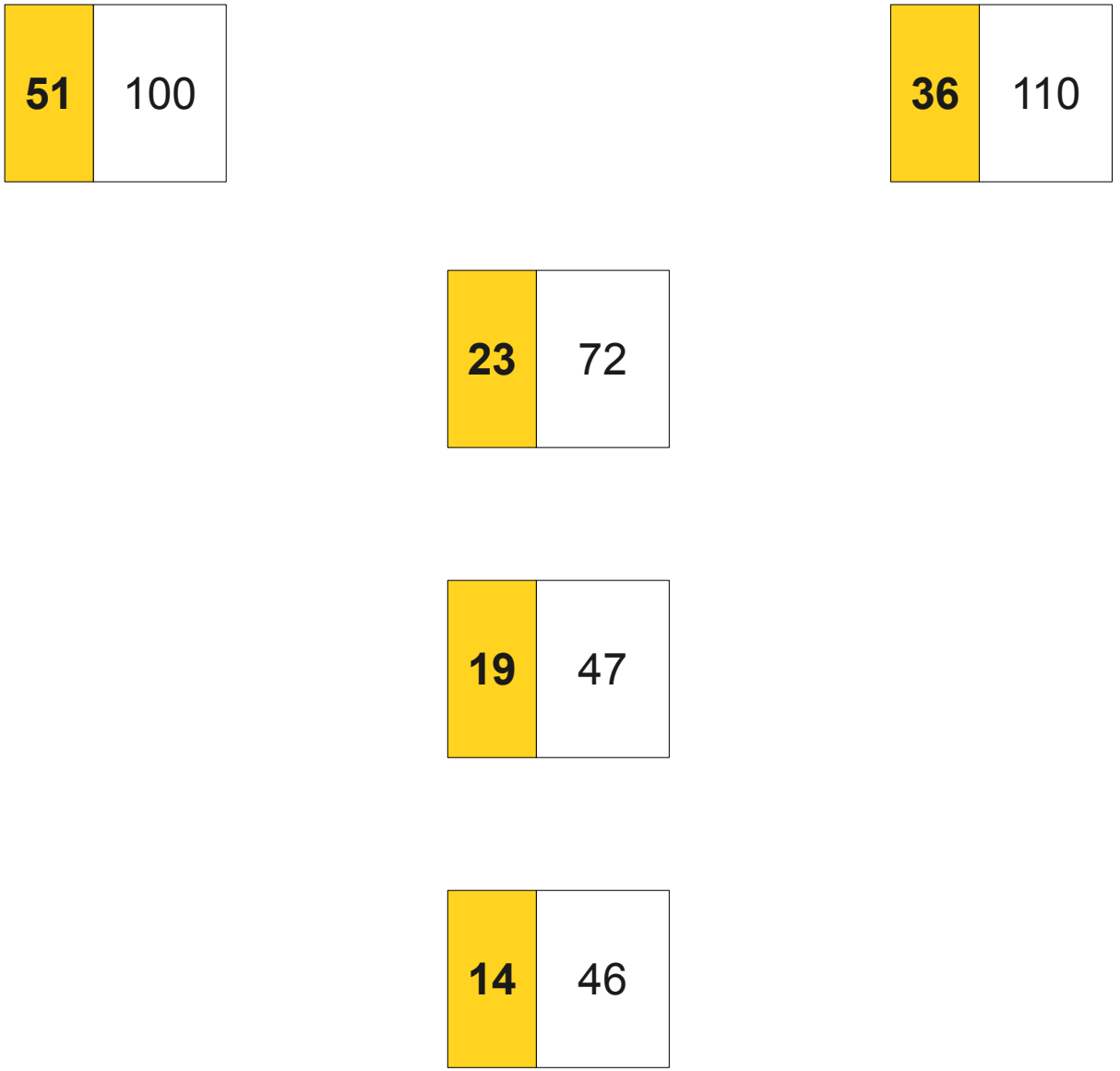
Inspired by http://tartarus.org/simon/2008-olympics-hasse/.
Data from http://news.bbc.co.uk/sport2/hi/olympics/medals_table/default.stm

Define the relationship

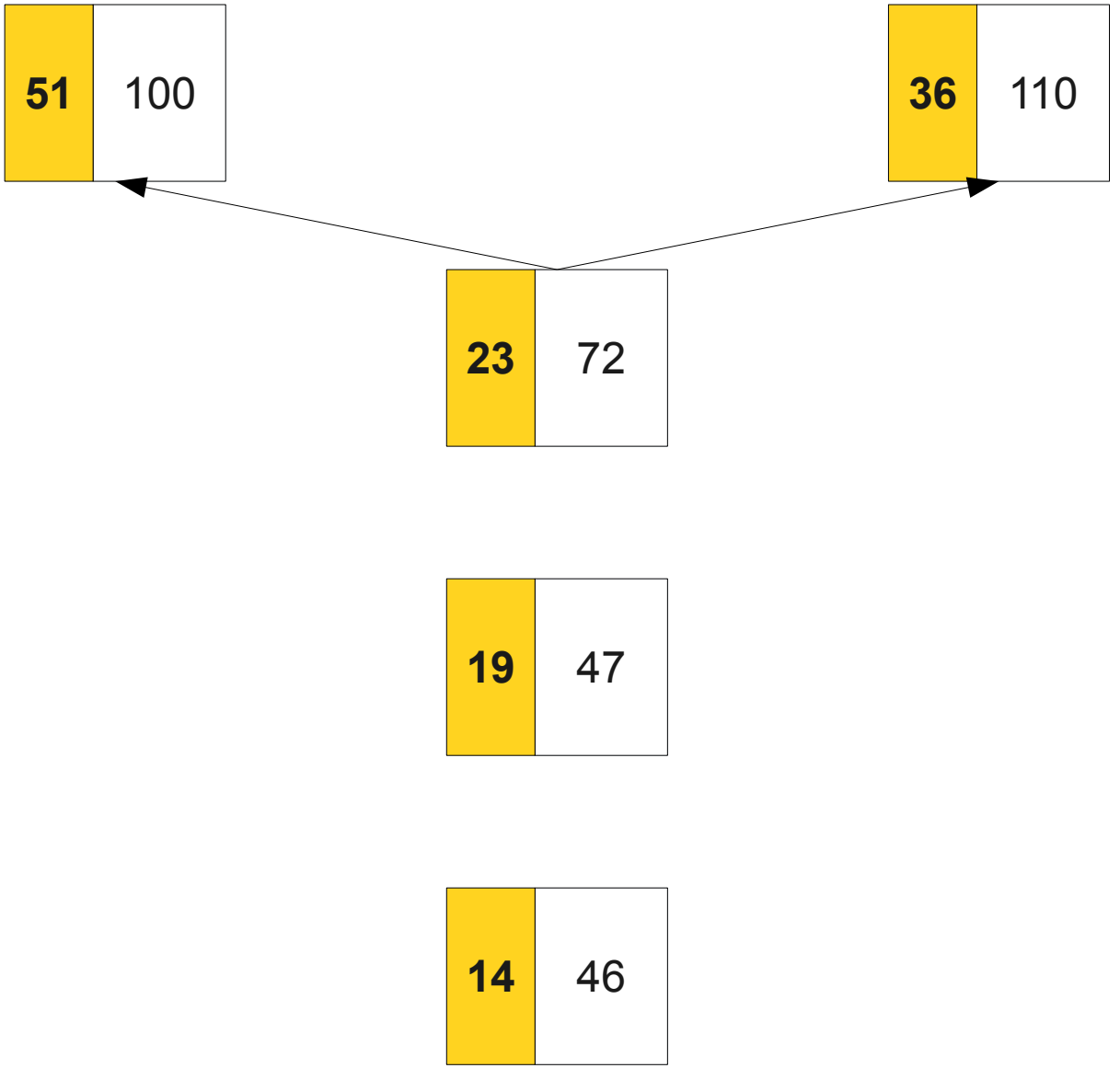
$$(\mathbf{gold}_0, \mathbf{total}_0)\mathbf{R}(\mathbf{gold}_1, \mathbf{total}_1)$$

to be true when

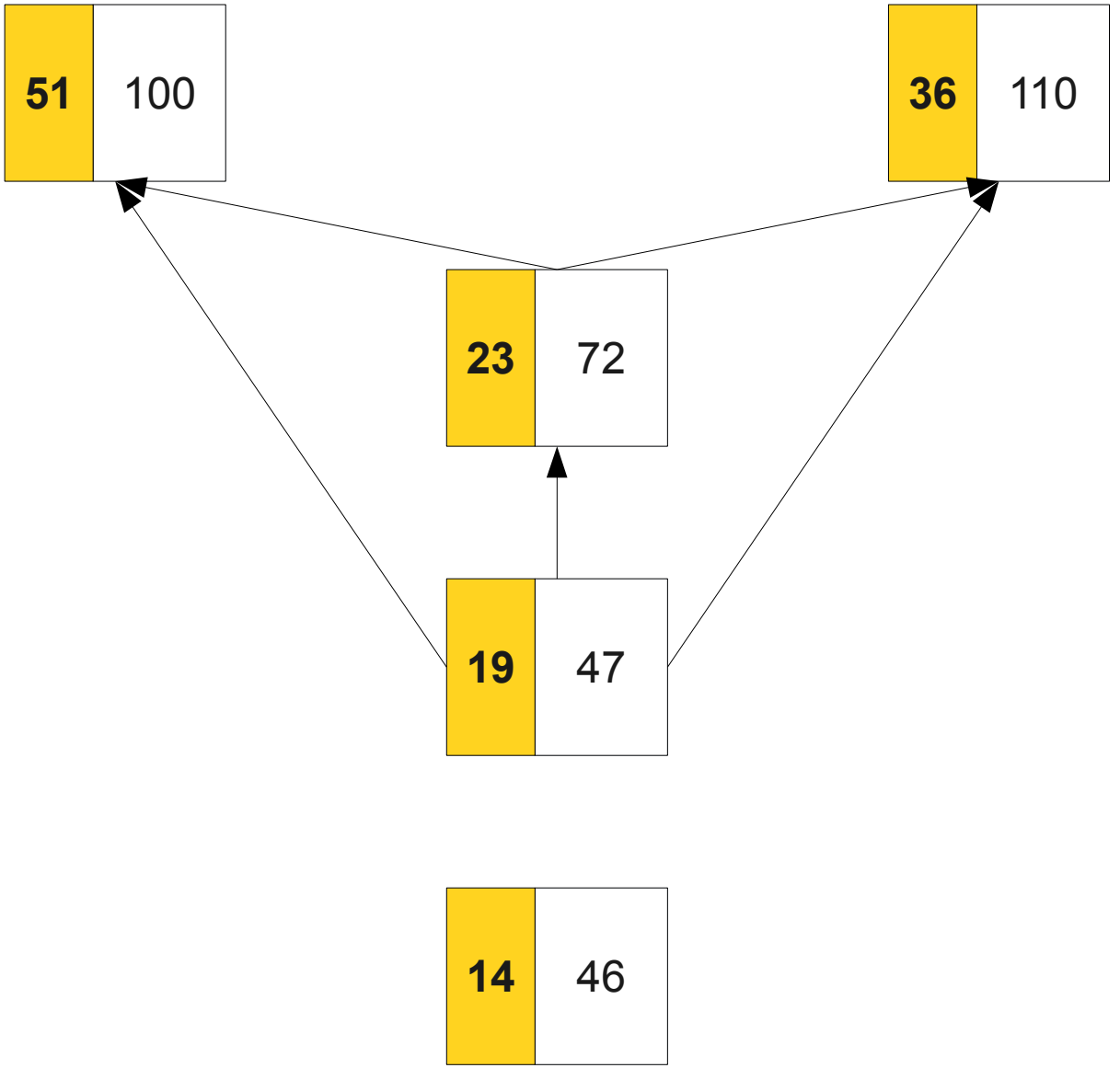
$$\mathbf{gold}_0 \leq \mathbf{gold}_1 \text{ and } \mathbf{total}_0 \leq \mathbf{total}_1$$



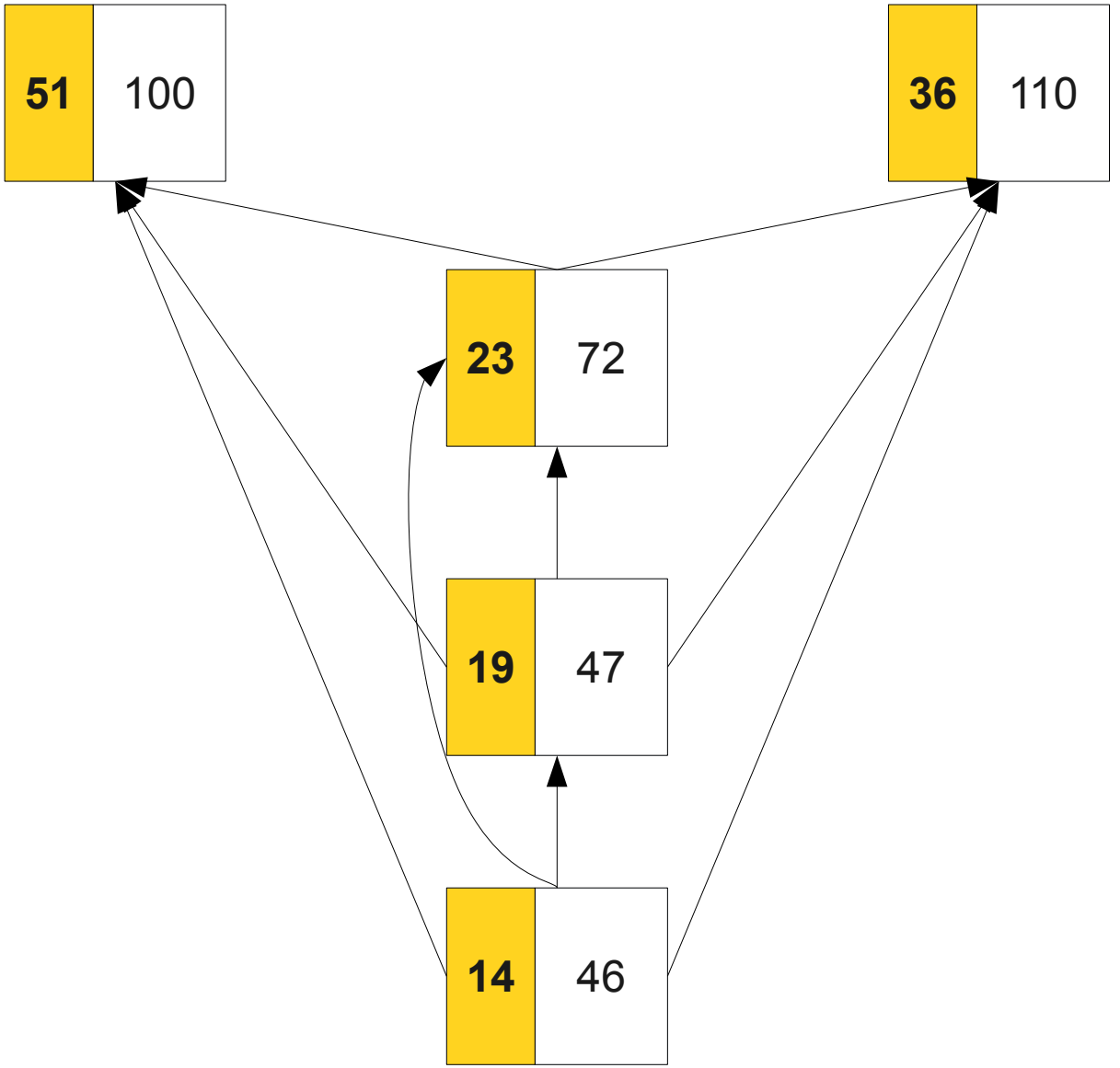
$$(g, t)R(g', t') \equiv g \leq g' \text{ and } t \leq t'$$



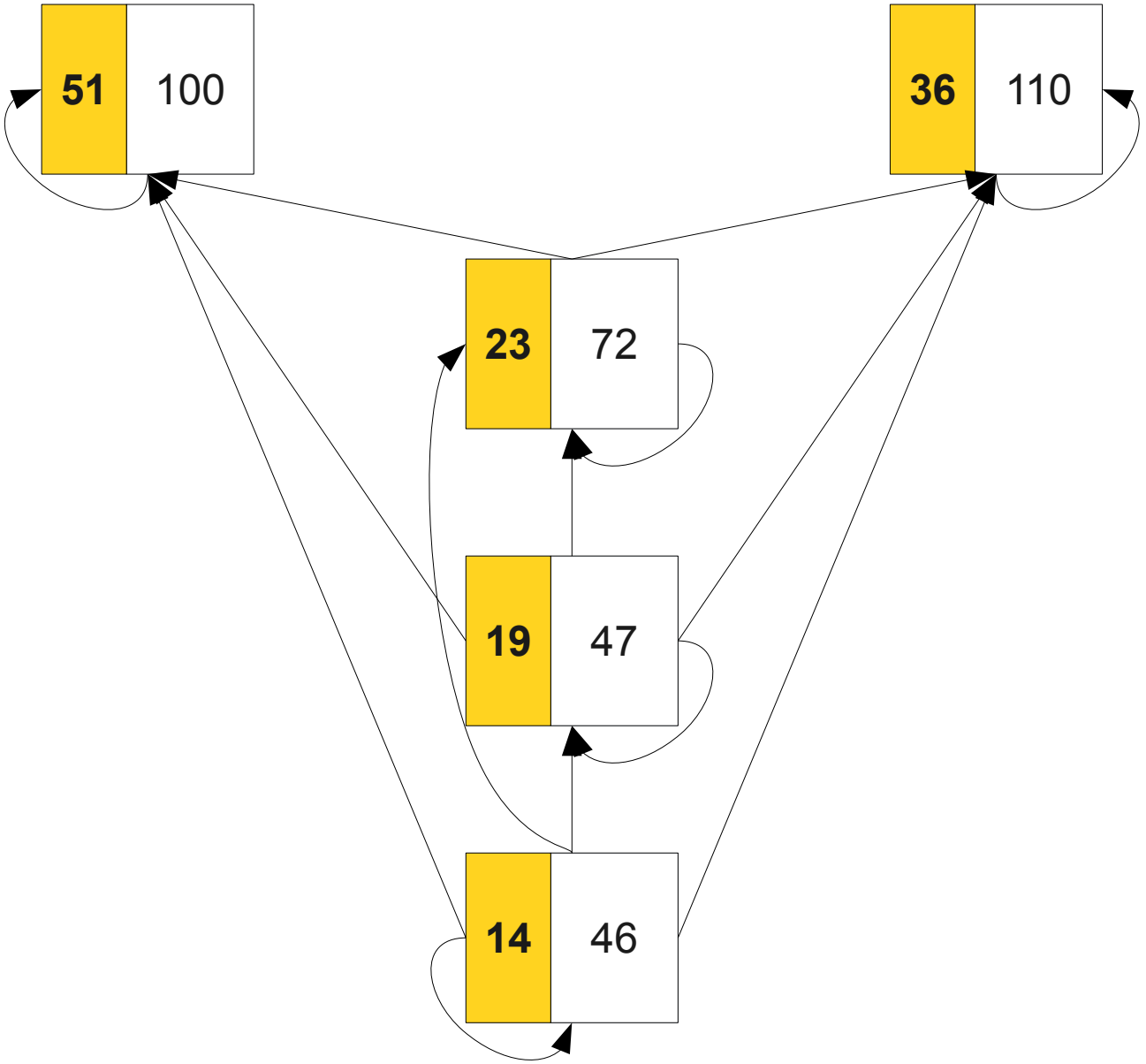
$$(g, t)R(g', t') \equiv g \leq g' \text{ and } t \leq t'$$



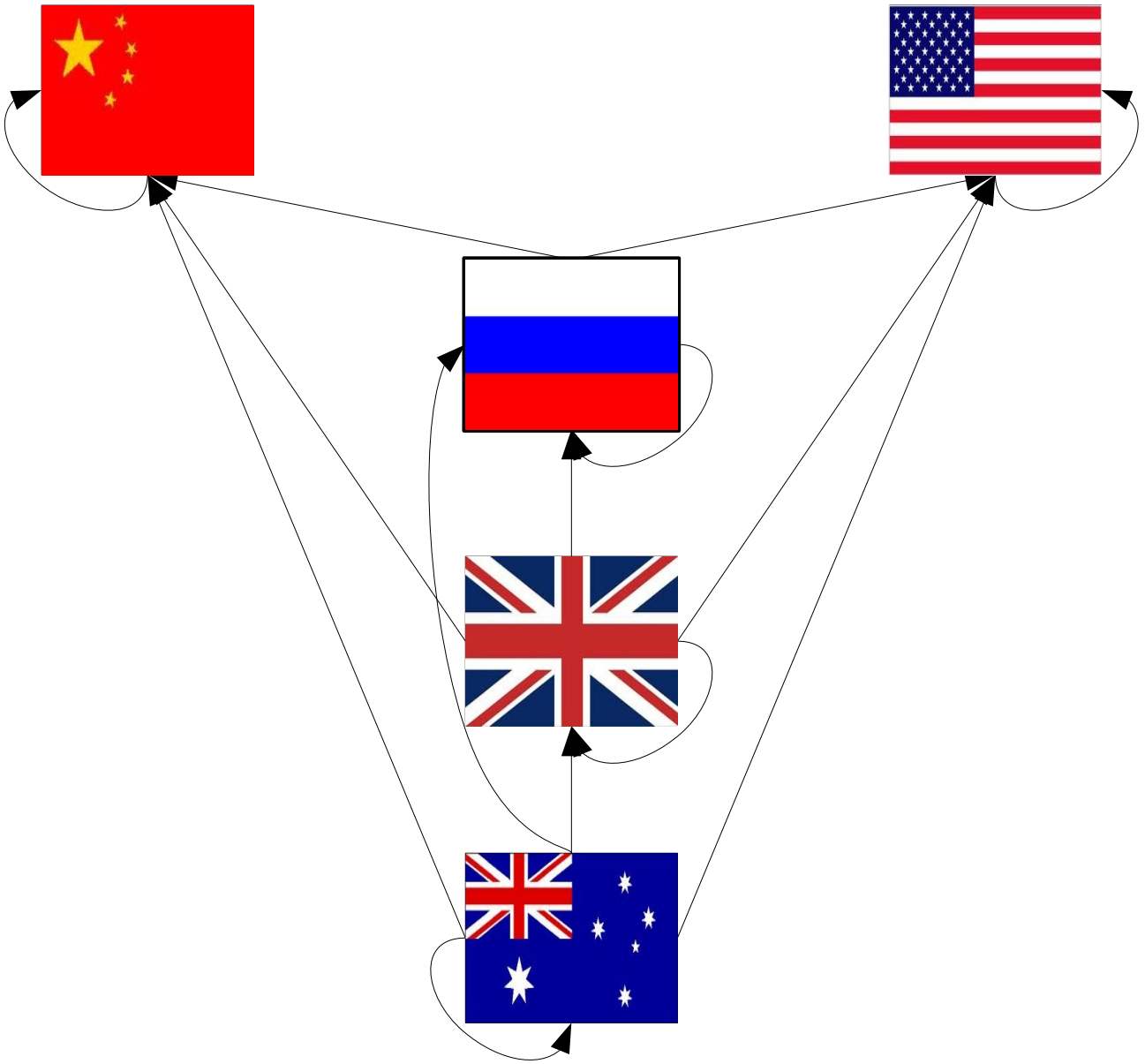
$$(g, t)R(g', t') \equiv g \leq g' \text{ and } t \leq t'$$



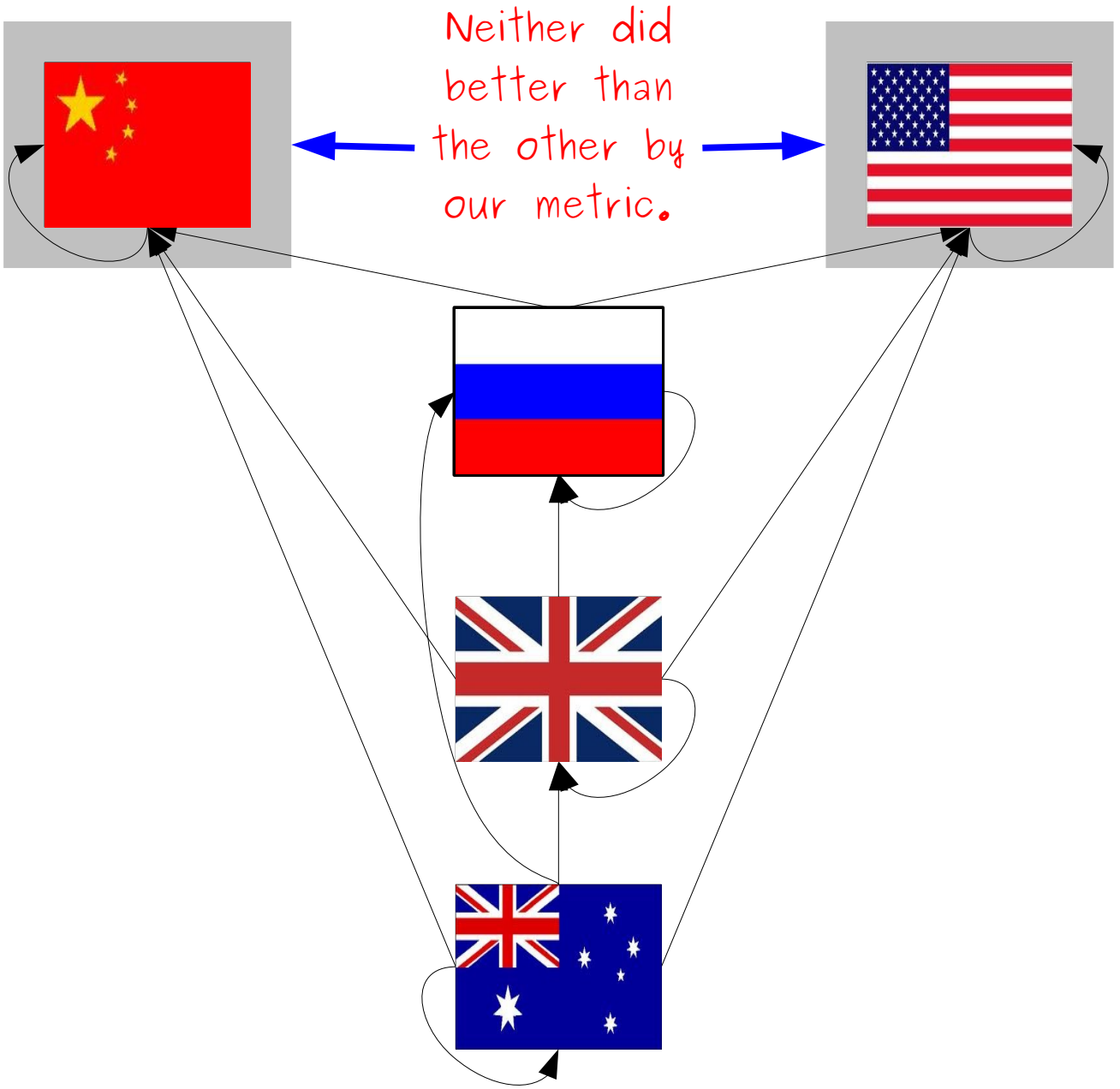
$$(g, t)R(g', t') \equiv g \leq g' \text{ and } t \leq t'$$



$$(g, t)R(g', t') \equiv g \leq g' \text{ and } t \leq t'$$



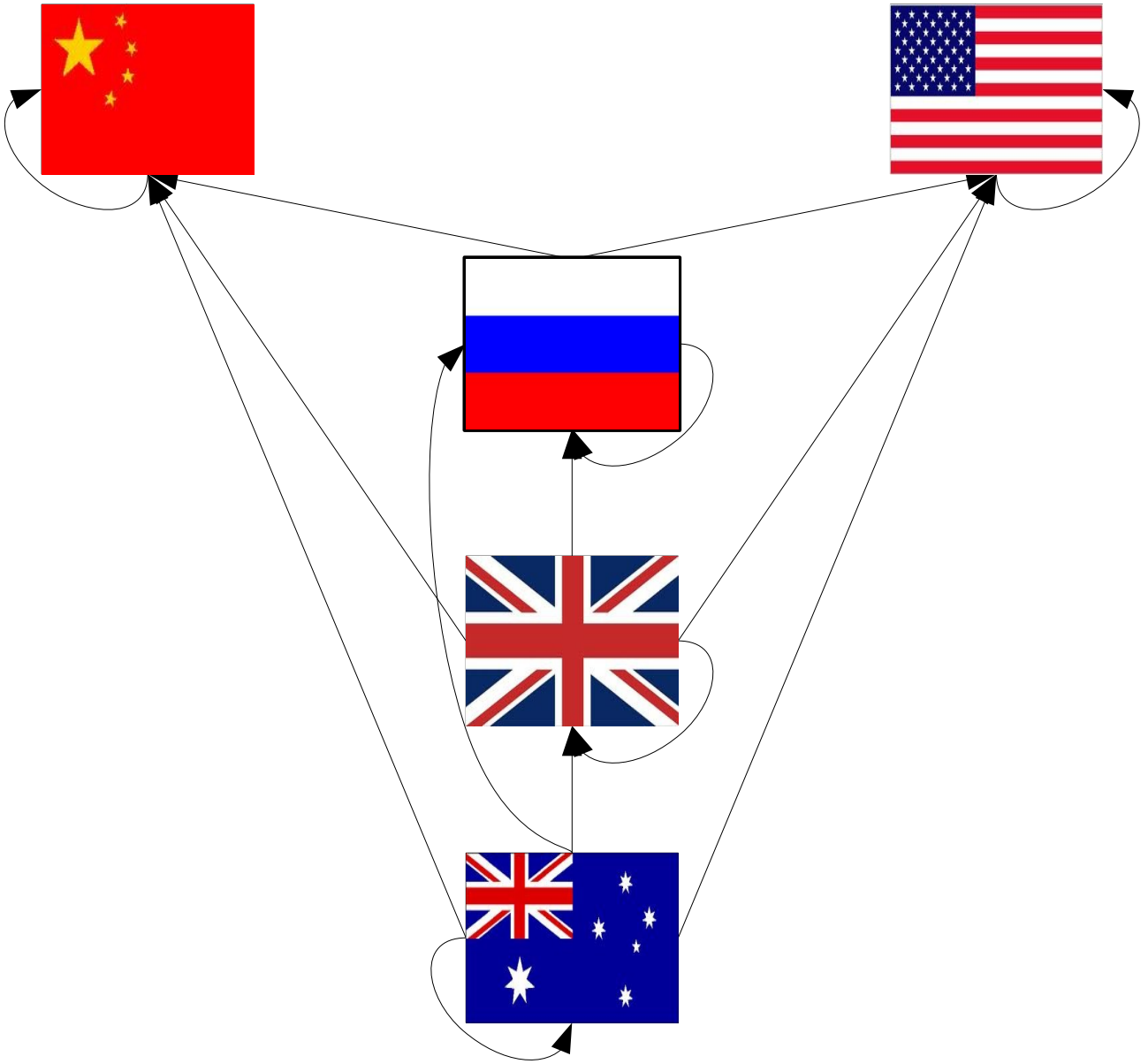
$$(g, t)R(g', t') \equiv g \leq g' \text{ and } t \leq t'$$

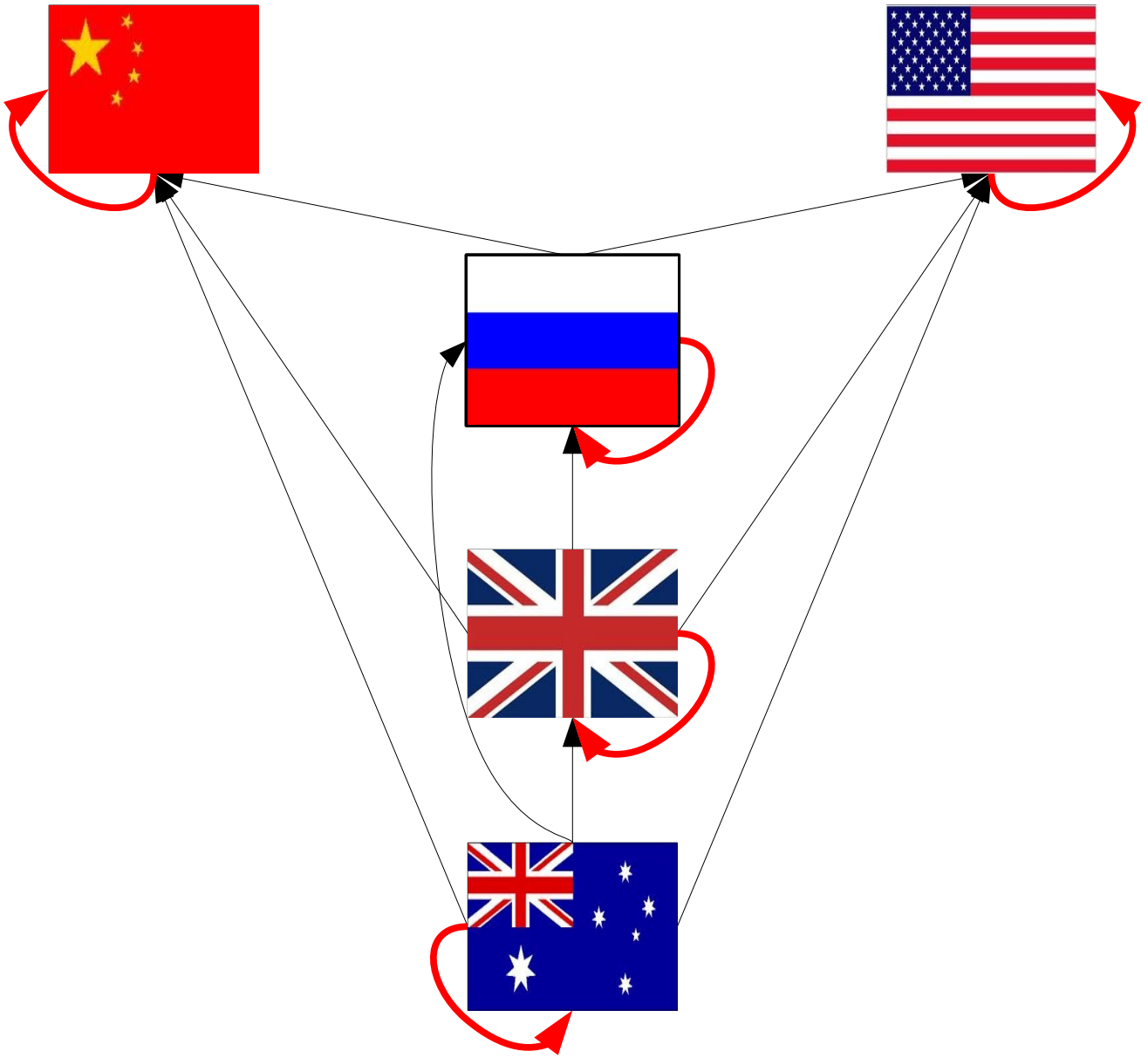


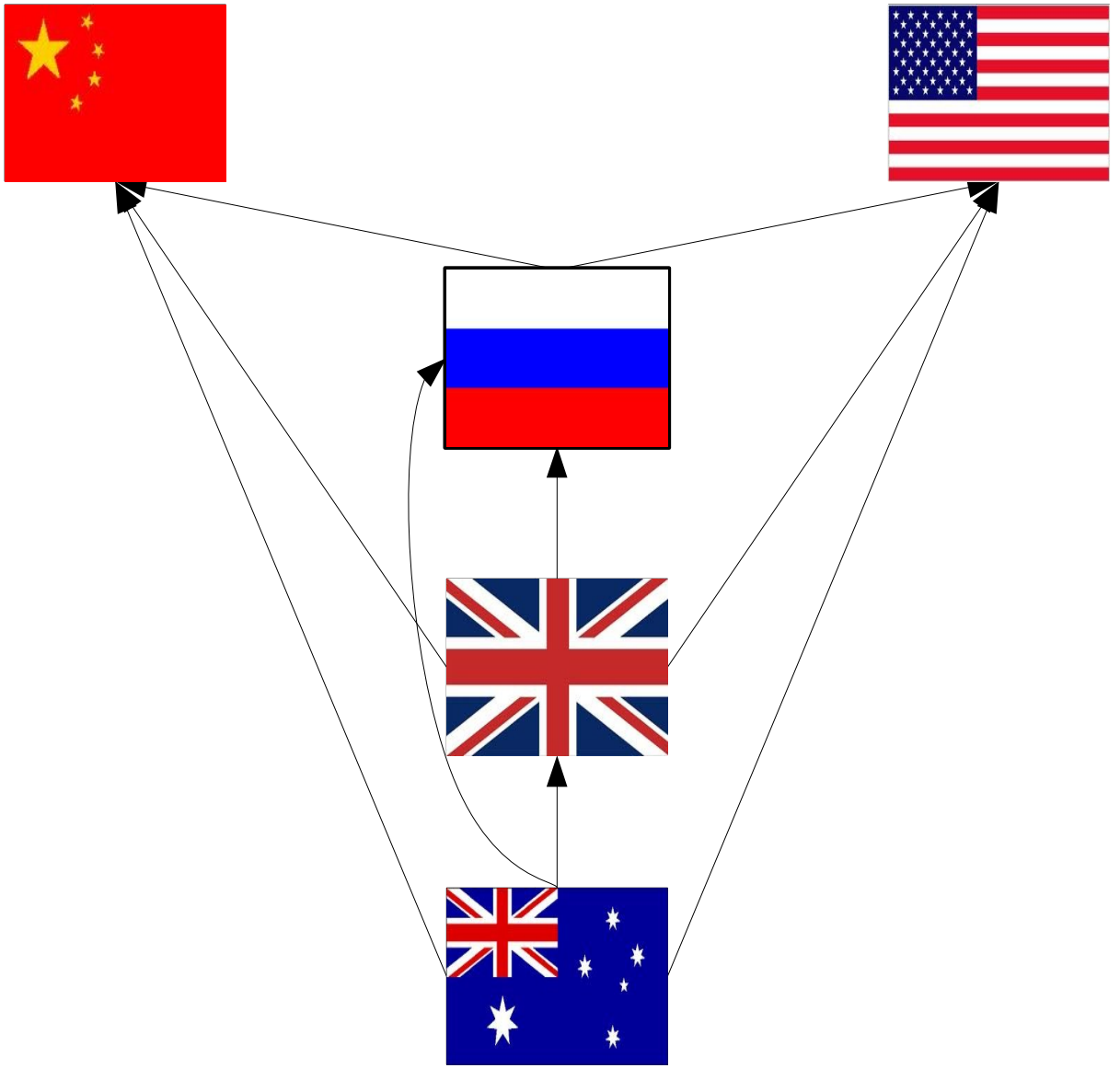
$$(g, t)R(g', t') \equiv g \leq g' \text{ and } t \leq t'$$

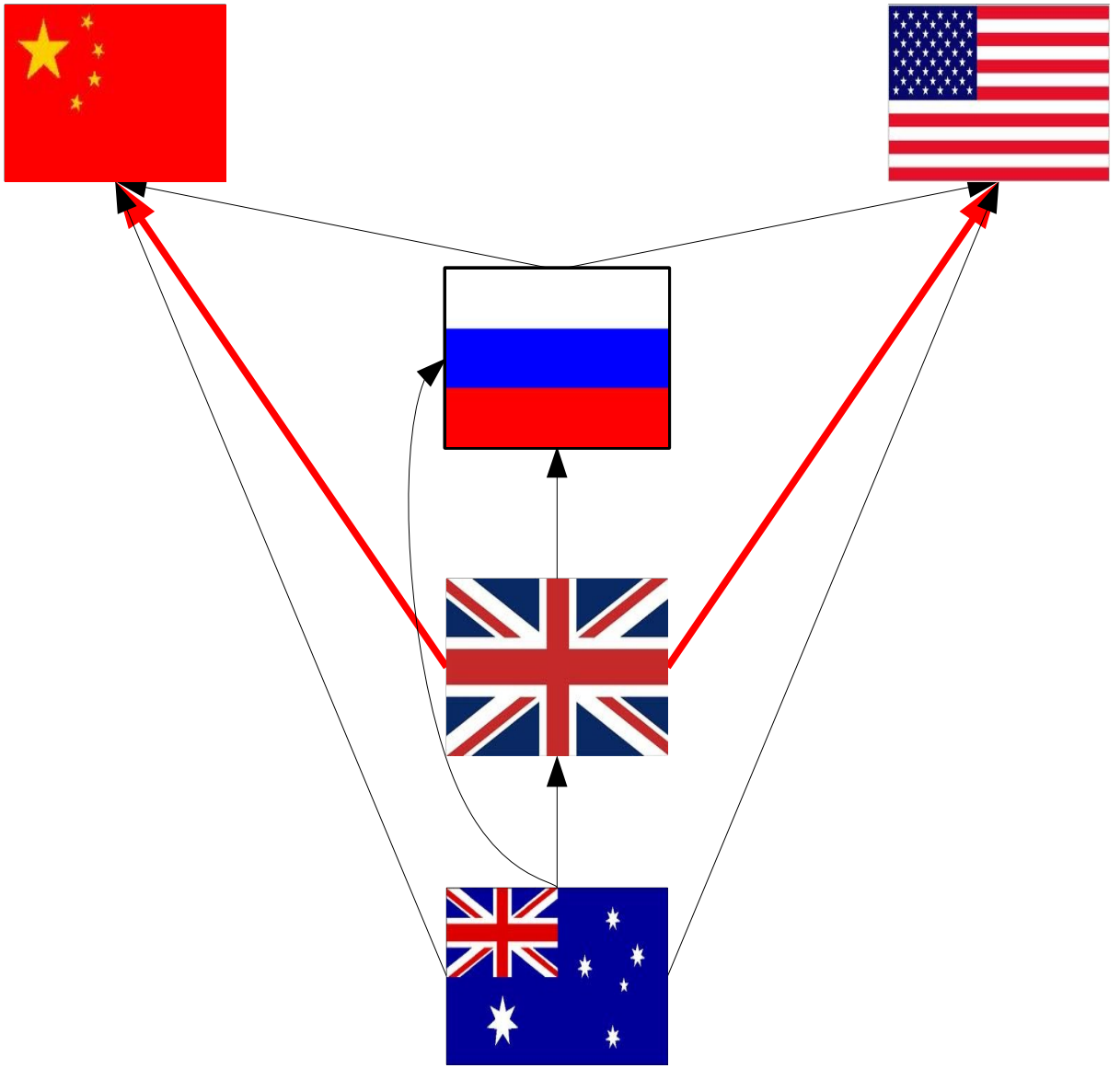
Partial and Total Orders

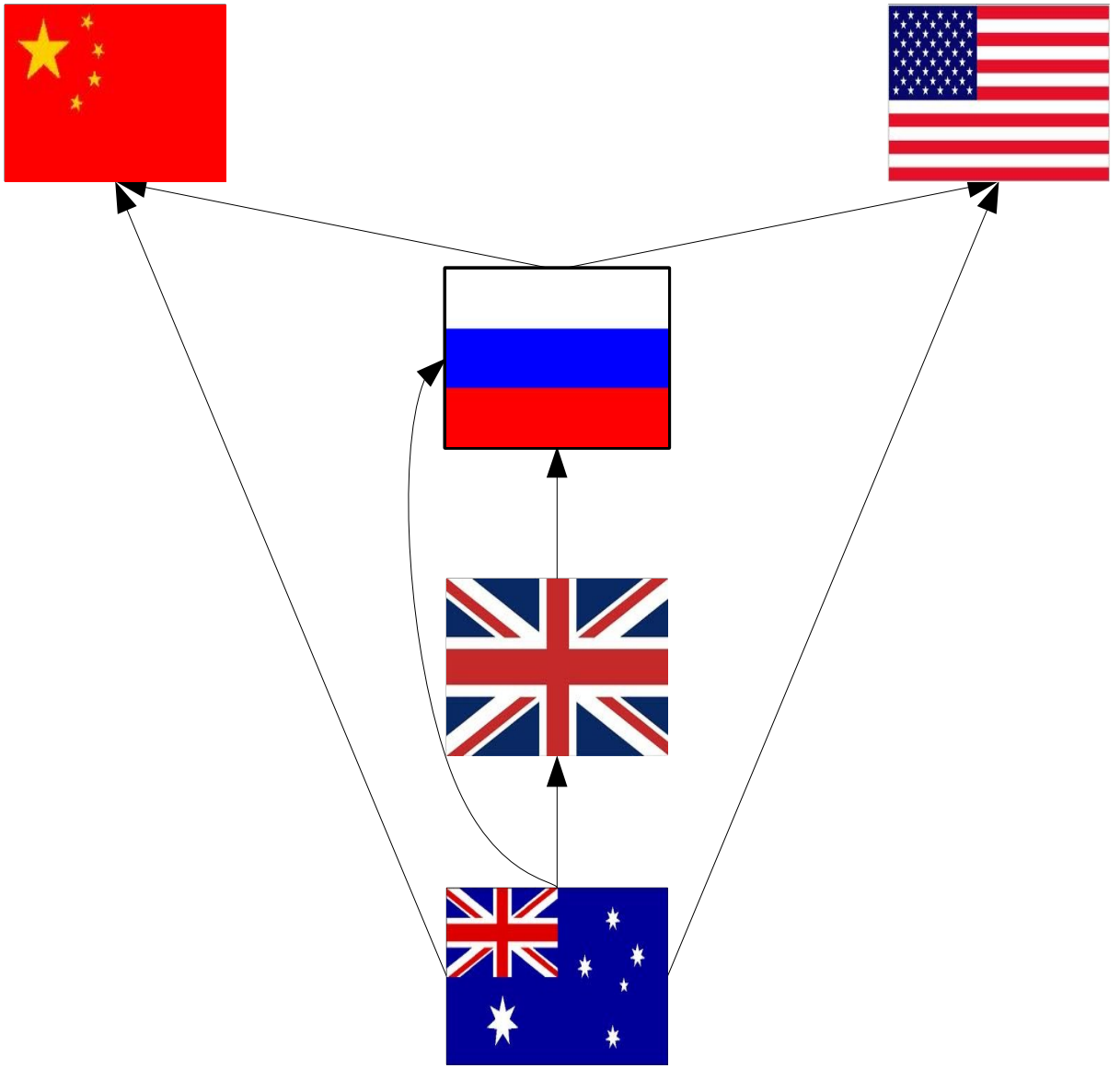
- A relation R is called **total** if, for any a and b, either aRb or bRa.
  - Could both be true?
- A **partial order** is called a **total order** if it is total.
- Examples:
  - Integers ordered by ≤.
  - Strings ordered alphabetically.

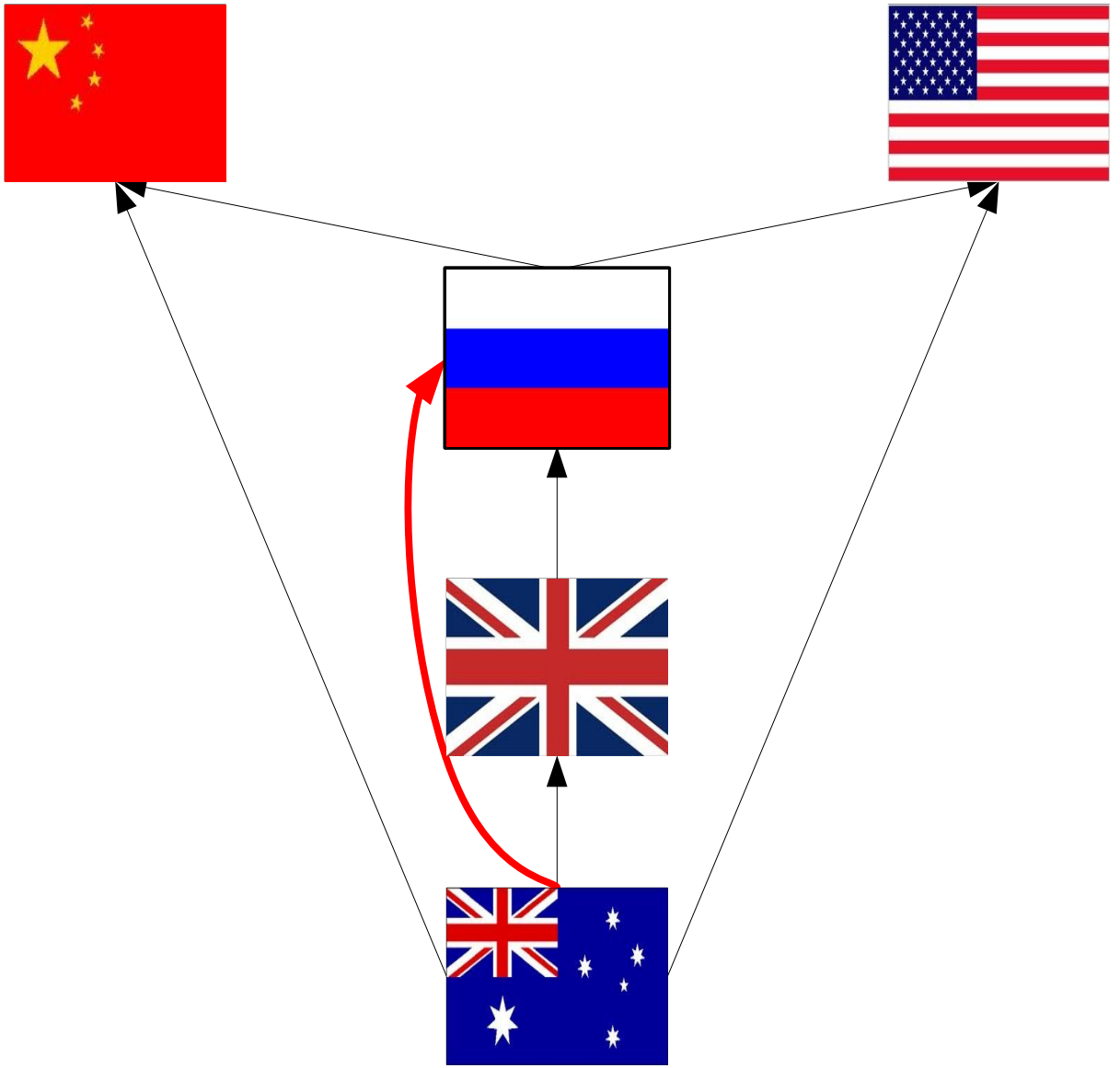


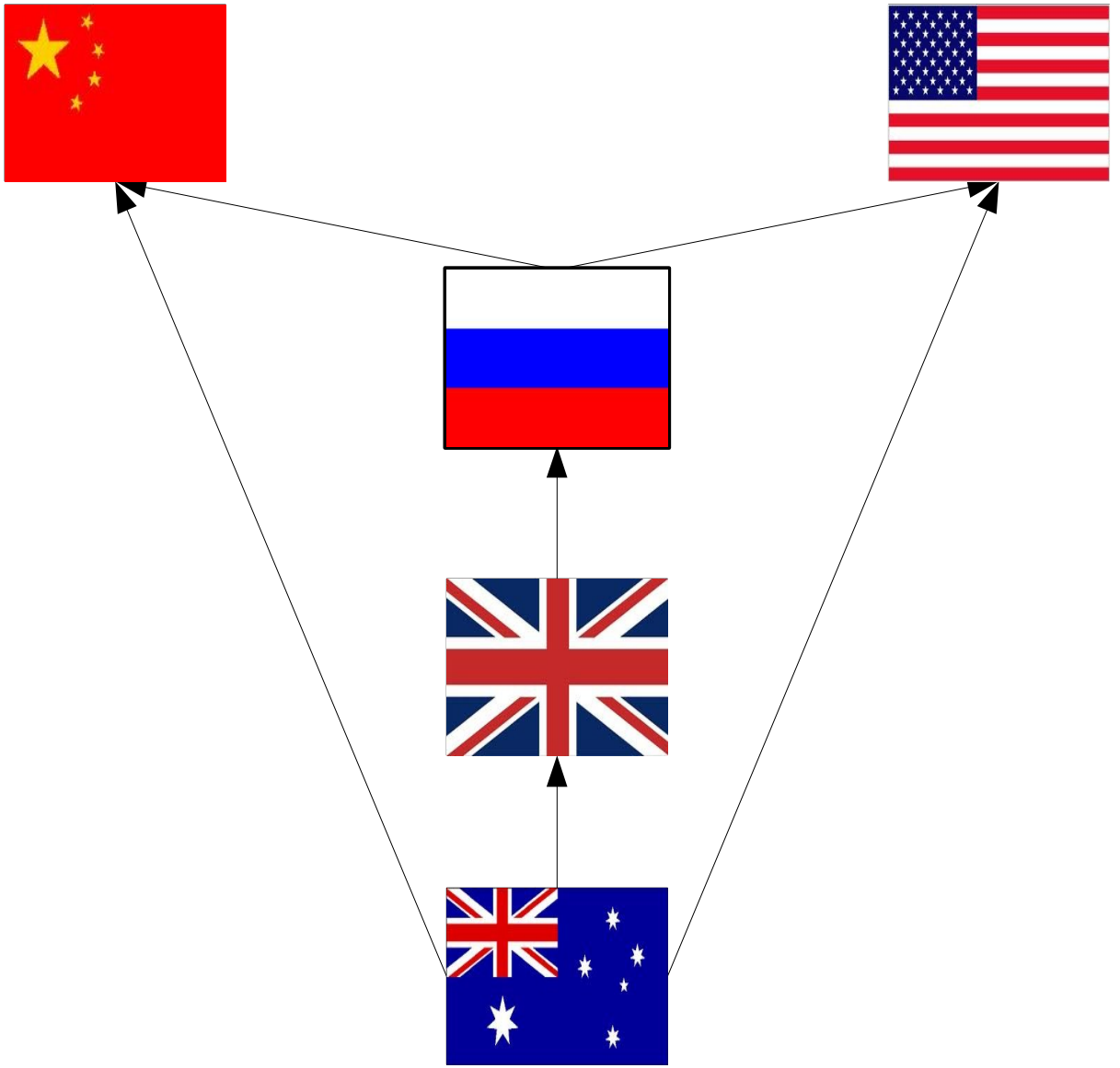


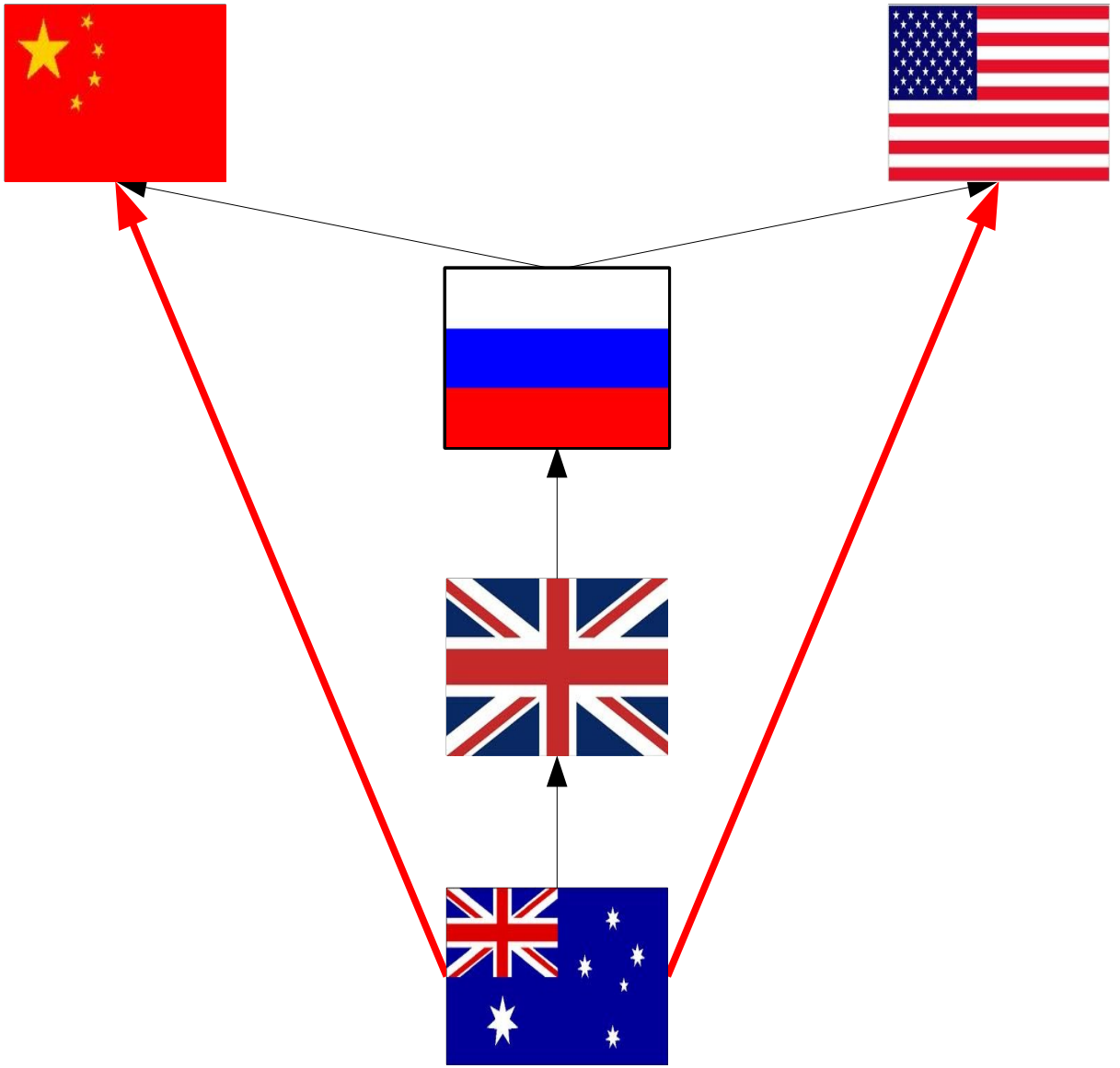


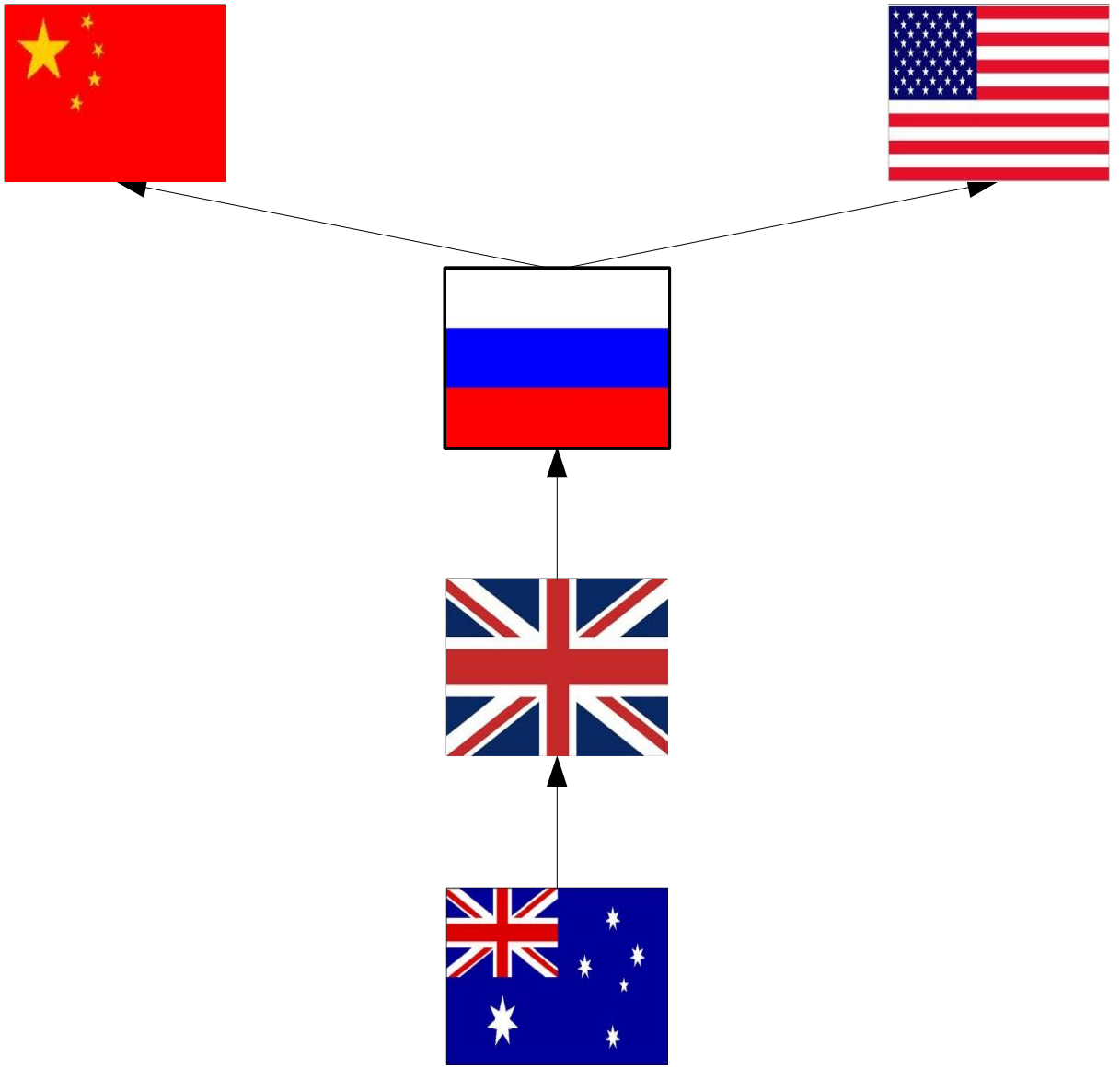




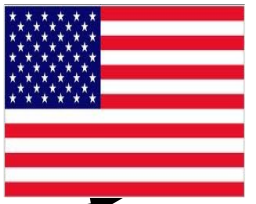


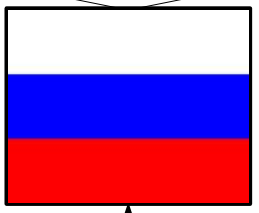






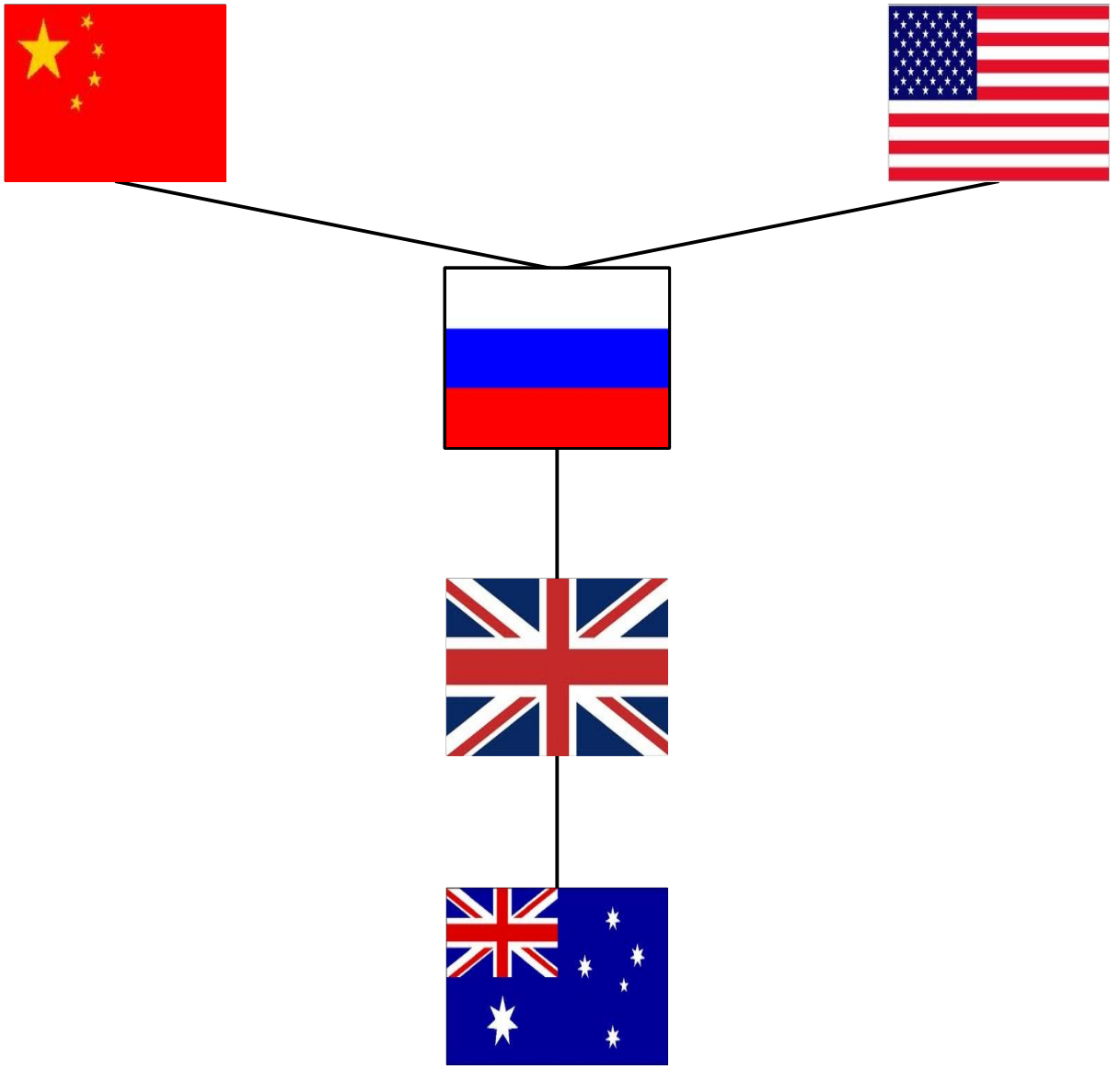
More Medals





Fewer Medals

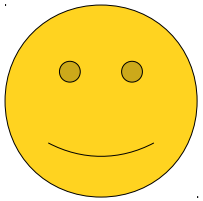
More Medals



Fewer Medals

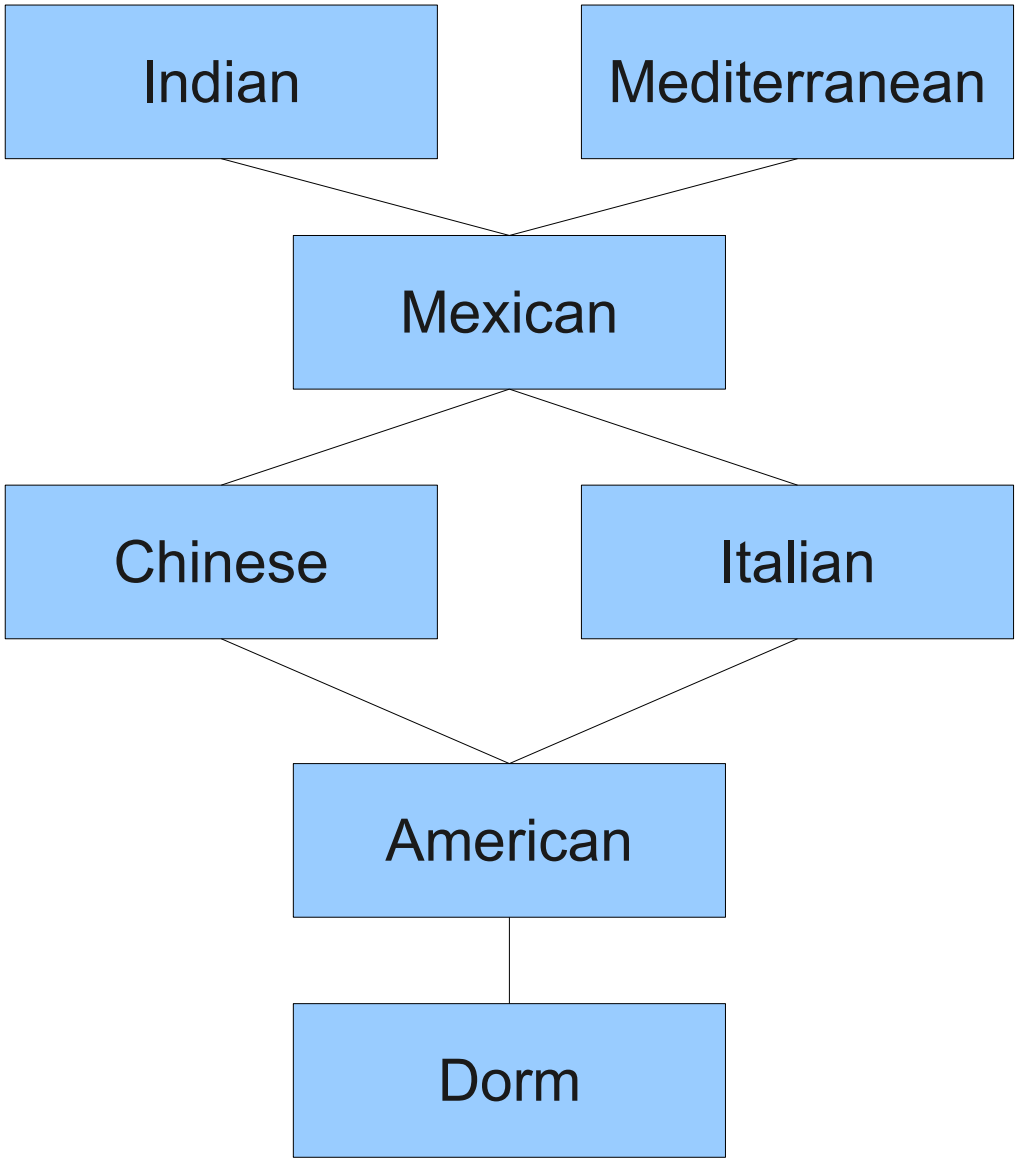
Hasse Diagrams

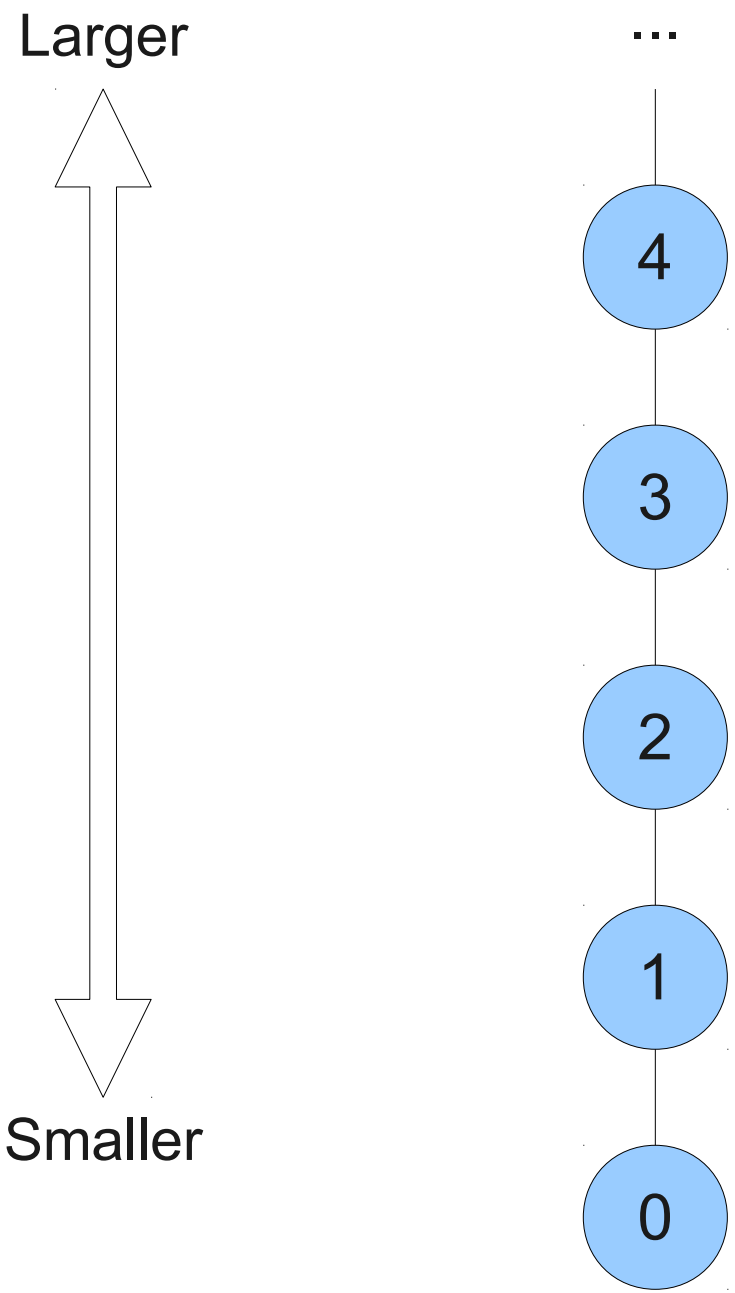
- A **Hasse diagram** is a graphical representation of a partial order.
- No self-loops (**reflexivity**)
- Higher elements are bigger than lower elements (**antisymmetry**)
- No redundant edges (**transitivity**)



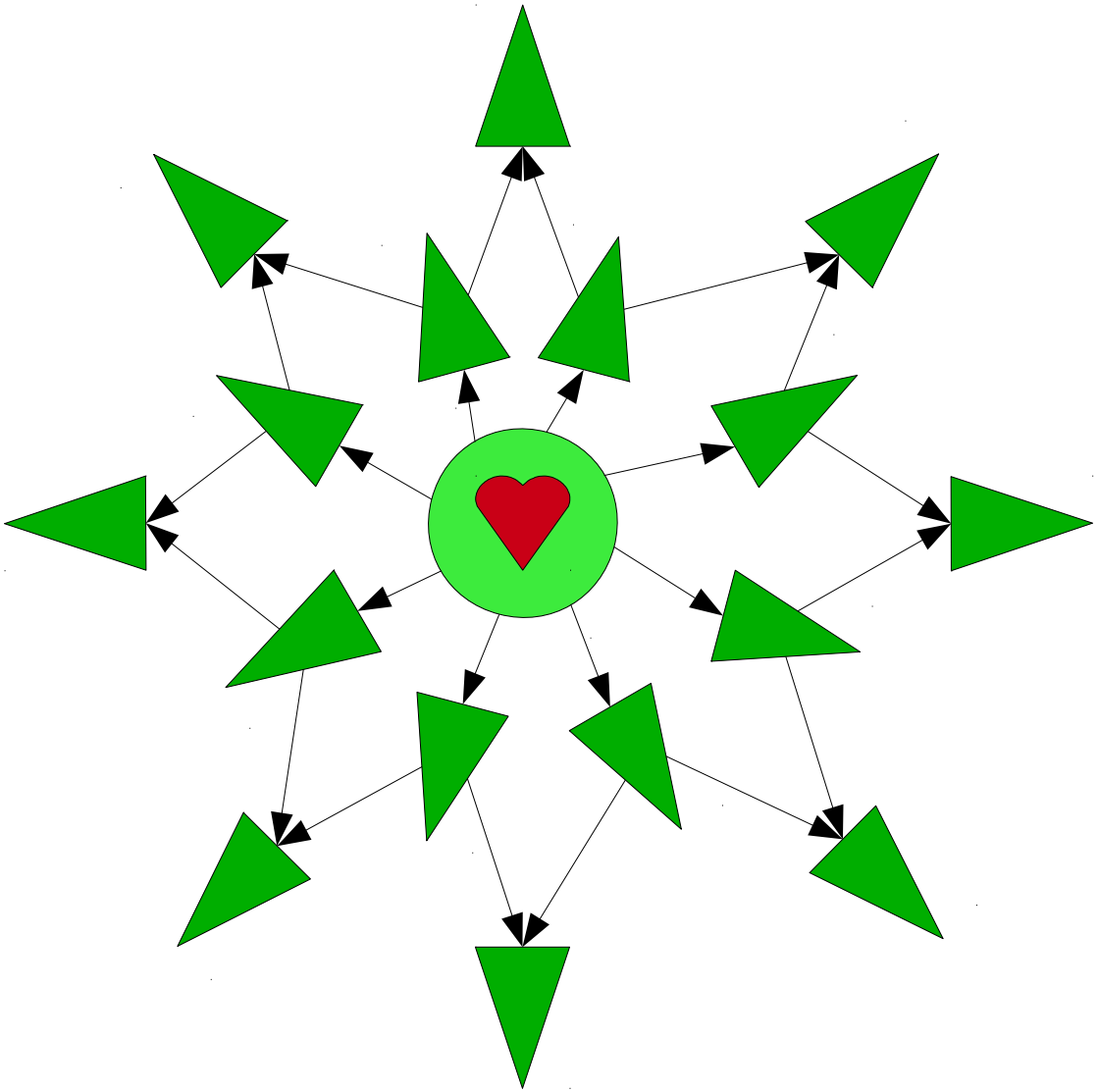
Tasty

Not Tasty

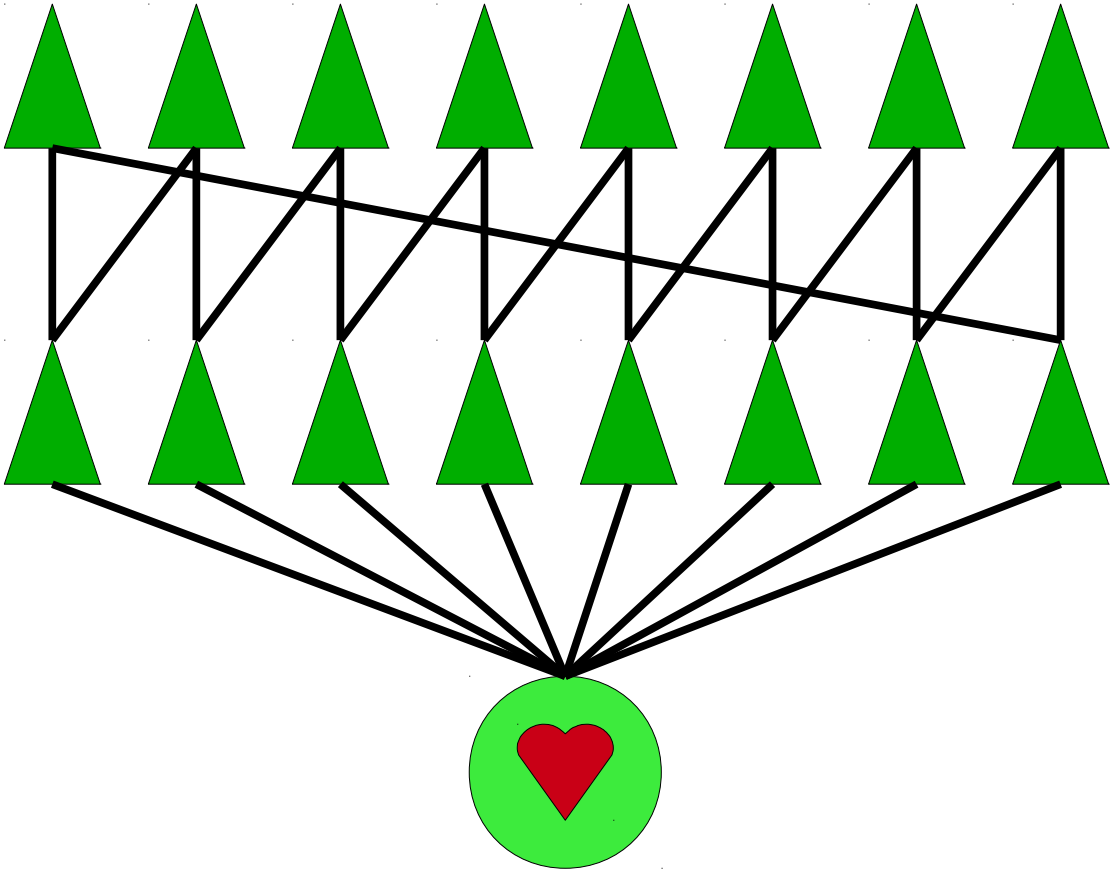



Larger
Smaller
...
4
3
2
1
0

Hasse Artichokes

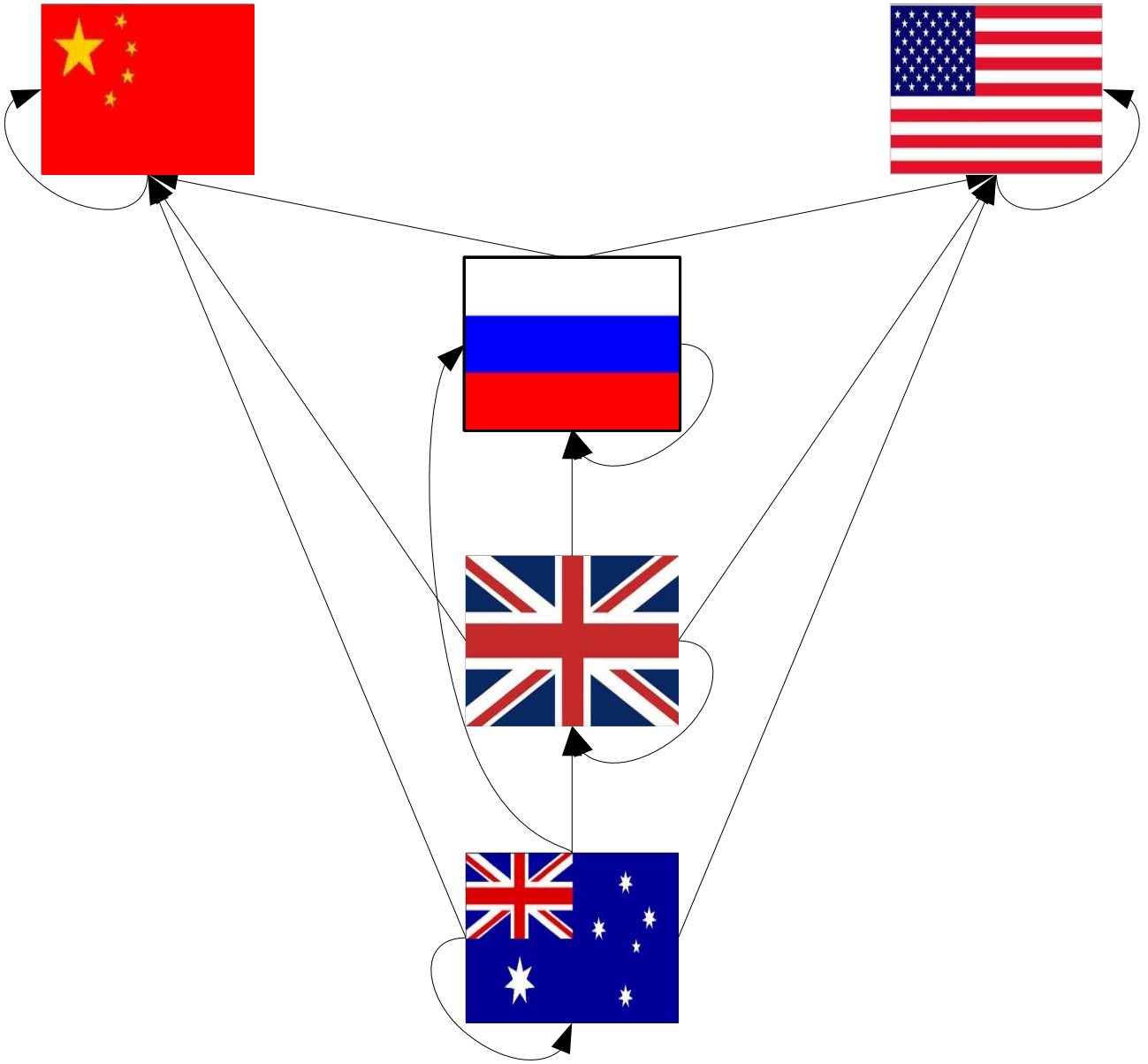


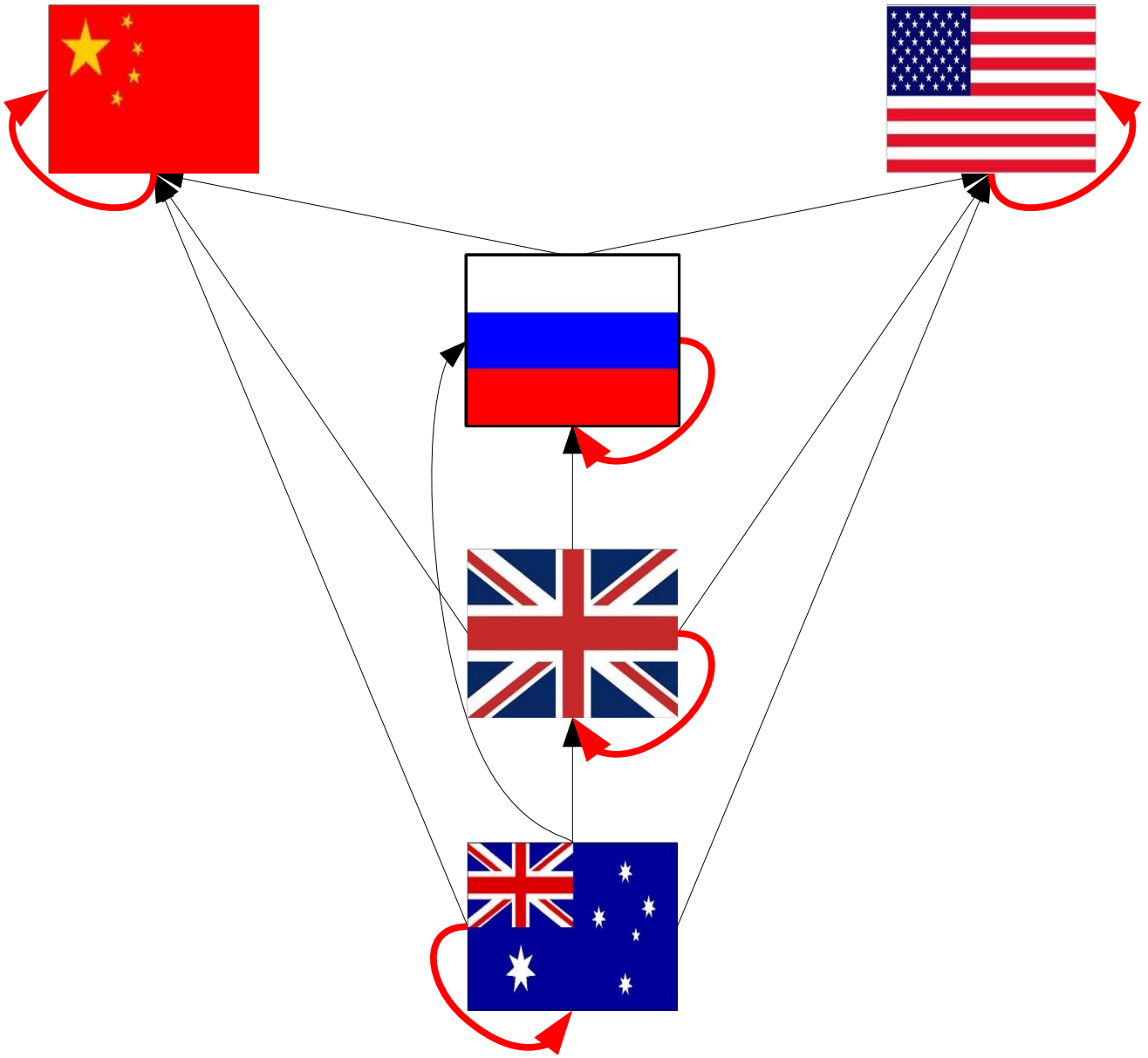
Hasse Artichokes

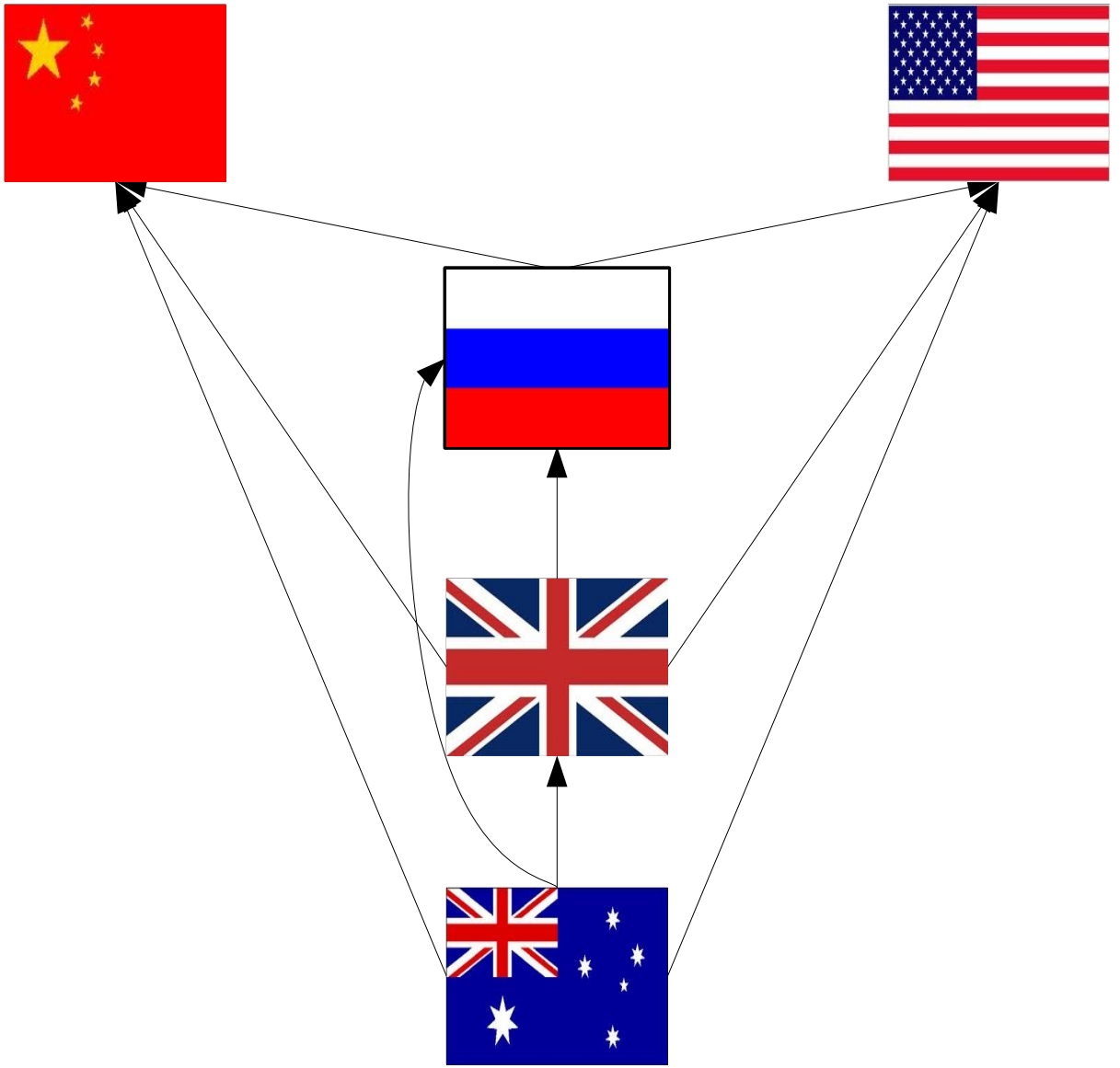


Hass Avocado

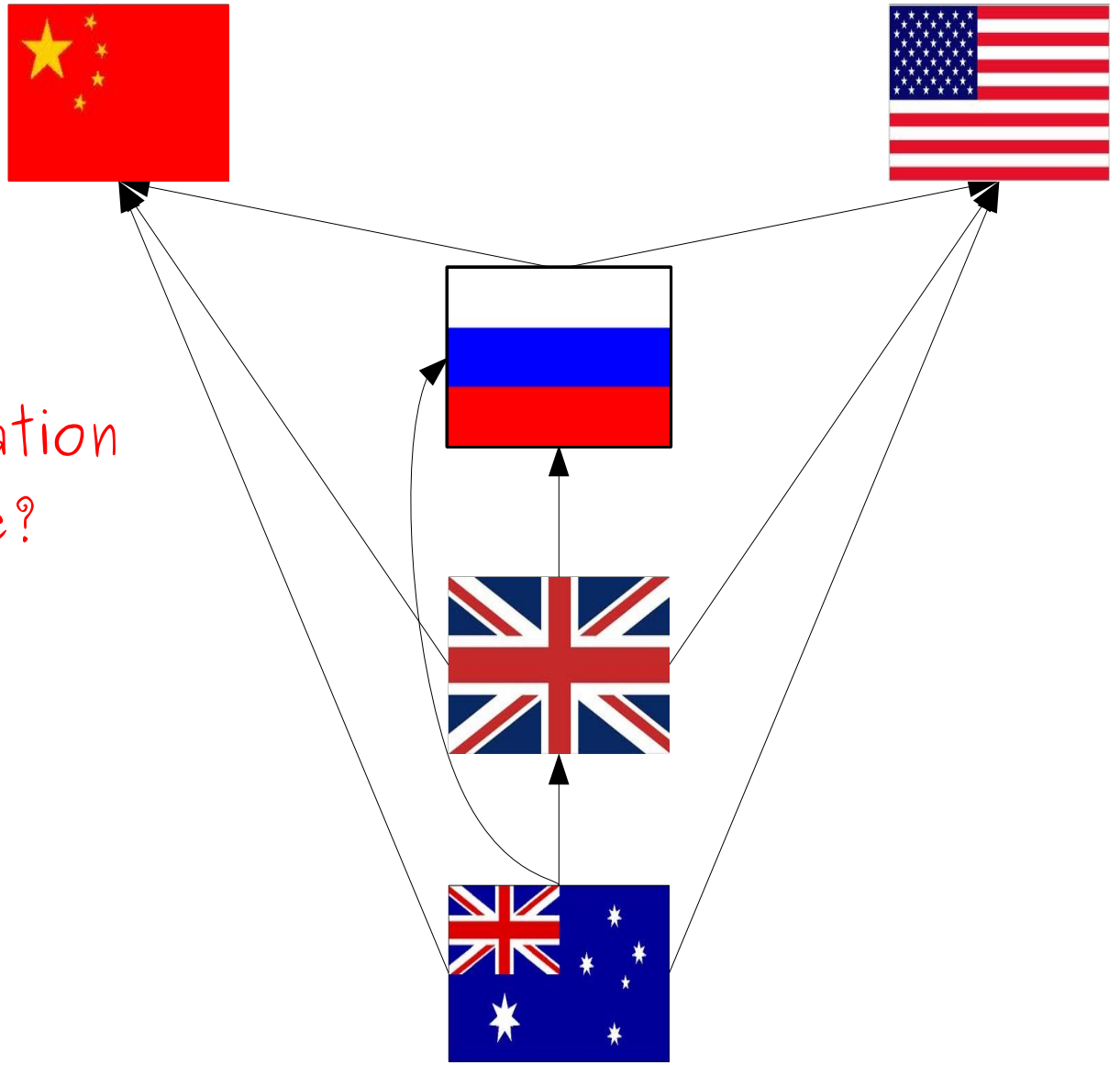
One Final Type of Order







Is this relation reflexive?



Irreflexivity

- Let R be a binary relation over A.
- R is **irreflexive** if for any a ∈ A, aRa is false.
  - x is heavier than y
  - x < y
  - x ≠ y
- Note that irreflexive does **not** mean “not reflexive.”
- Reflexive: Every element is **always** related to itself.
- Irreflexive: Every element is **never** related to itself.

Strict Orders

- A binary relation R over a set A is called a **strict order** if it is
  - **Irreflexive**,
  - **Antisymmetric**, and
  - **Transitive**
- A **strict total order** is a strict order in which for any x and y, exactly one of the following is true:
  - xRy
  - yRx
  - x = y
- This property is called **trichotomy**.

Important Terms for Today

- Cartesian Product
- Ordered Pair
- Graph
- Path
- Connectivity
- Cycle
- Degree
- DAG
- Topological Sort
- Relation
- Reflexivity
- Symmetry
- Transitivity
- Antisymmetry
- Irreflexivity
- Totality
- Equivalence relation
- Equivalence class
- Partial order
- Hasse Diagram
- Total order
- Strict order

Next Time

- Proof techniques:
  - Direct proofs
  - Proofs by contradiction
  - Proofs by contrapositive