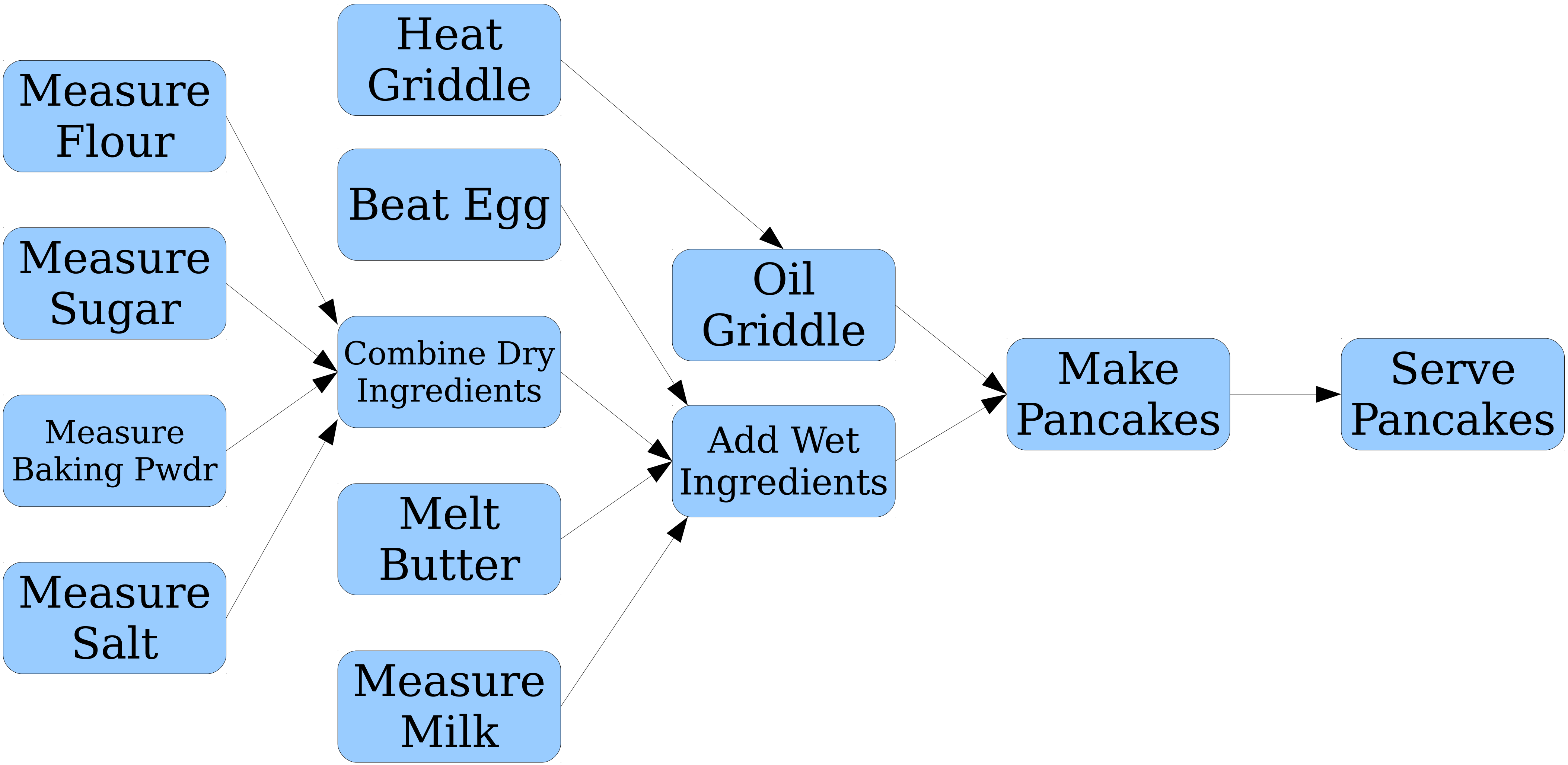
Relations and Graphs

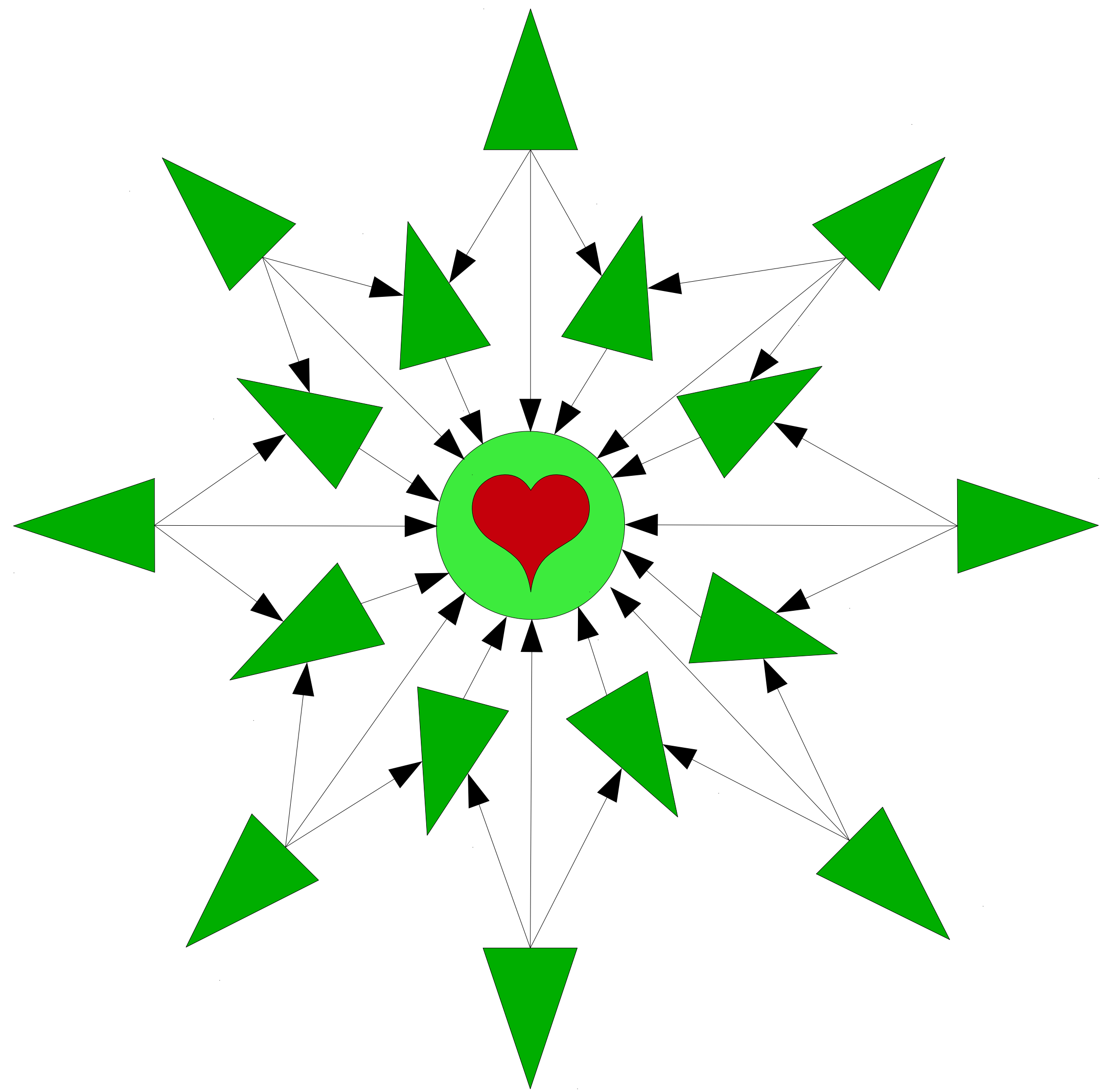
Recap from Last Time

Binary Relations

- A ***binary relation over a set A*** is a predicate R that can be applied to pairs of elements drawn from A.
- If R is a binary relation over A and it holds for the pair (a, b), we write $\boldsymbol{aRb}$.
  - For example: $3 = 3$, $5 < 7$, and $\emptyset \subseteq \mathbb{N}$.
- If R is a binary relation over A and it does not hold for the pair (a, b), we write $\boldsymbol{a \not R b}$.
  - For example: $4 \neq 3$, $4 \not< 3$, and $\mathbb{N} \not\subseteq \emptyset$.

Prerequisite Structures

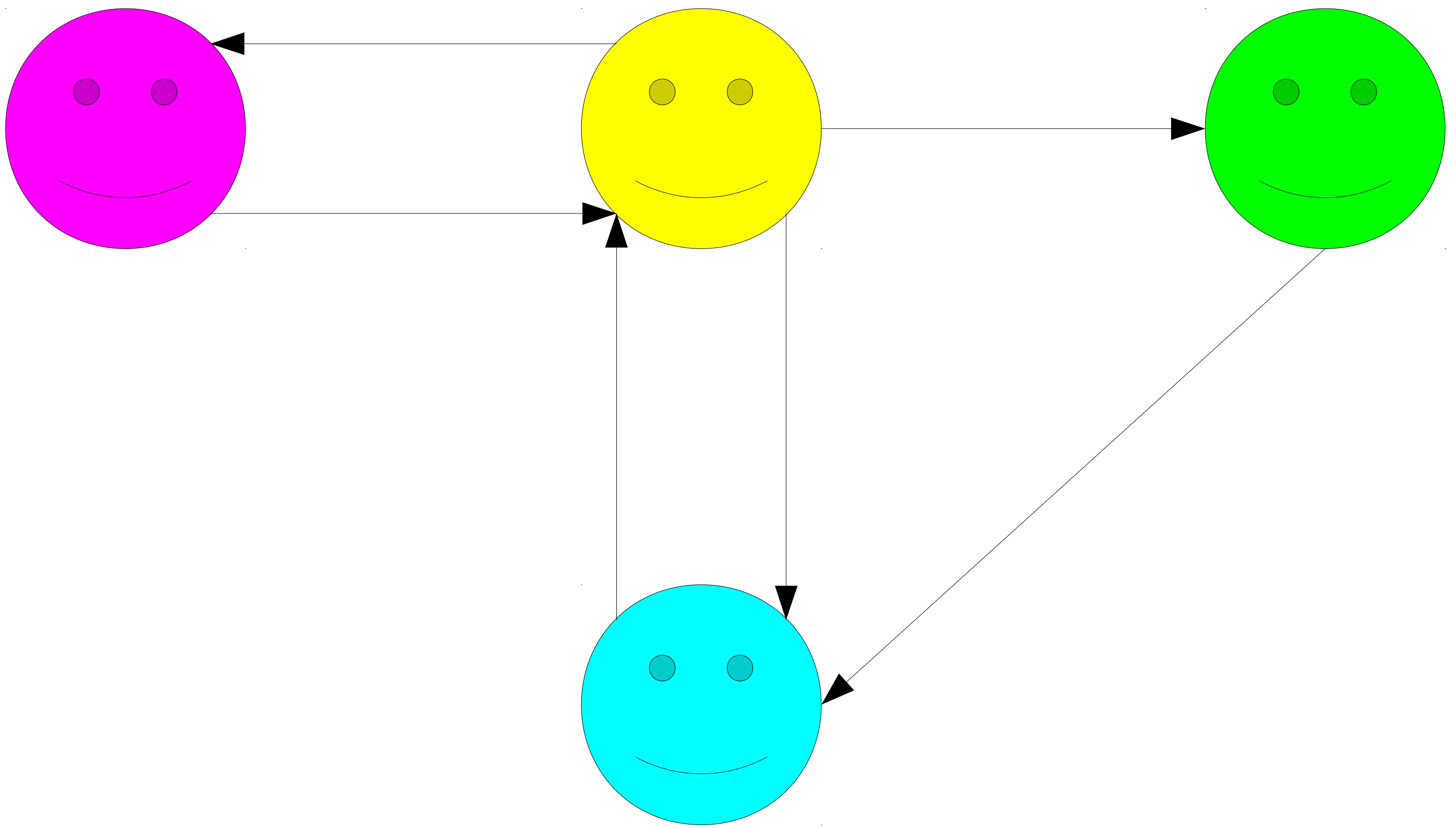

Measure Flour
Measure Sugar
Measure Baking Pwdr
Measure Salt
Heat Griddle
Beat Egg
Combine Dry Ingredients
Melt Butter
Measure Milk
Oil Griddle
Add Wet Ingredients
Make Pancakes
Serve Pancakes

Strict Orders

- A ***strict order*** is a relation that is irreflexive, asymmetric and transitive.
  - We'll refresh those definitions in a second.
- Some examples:
  - $x < y$.
  - a can run faster than b.
  - $A \subset B$ (that is, $A \subseteq B$ and $A \neq B$).

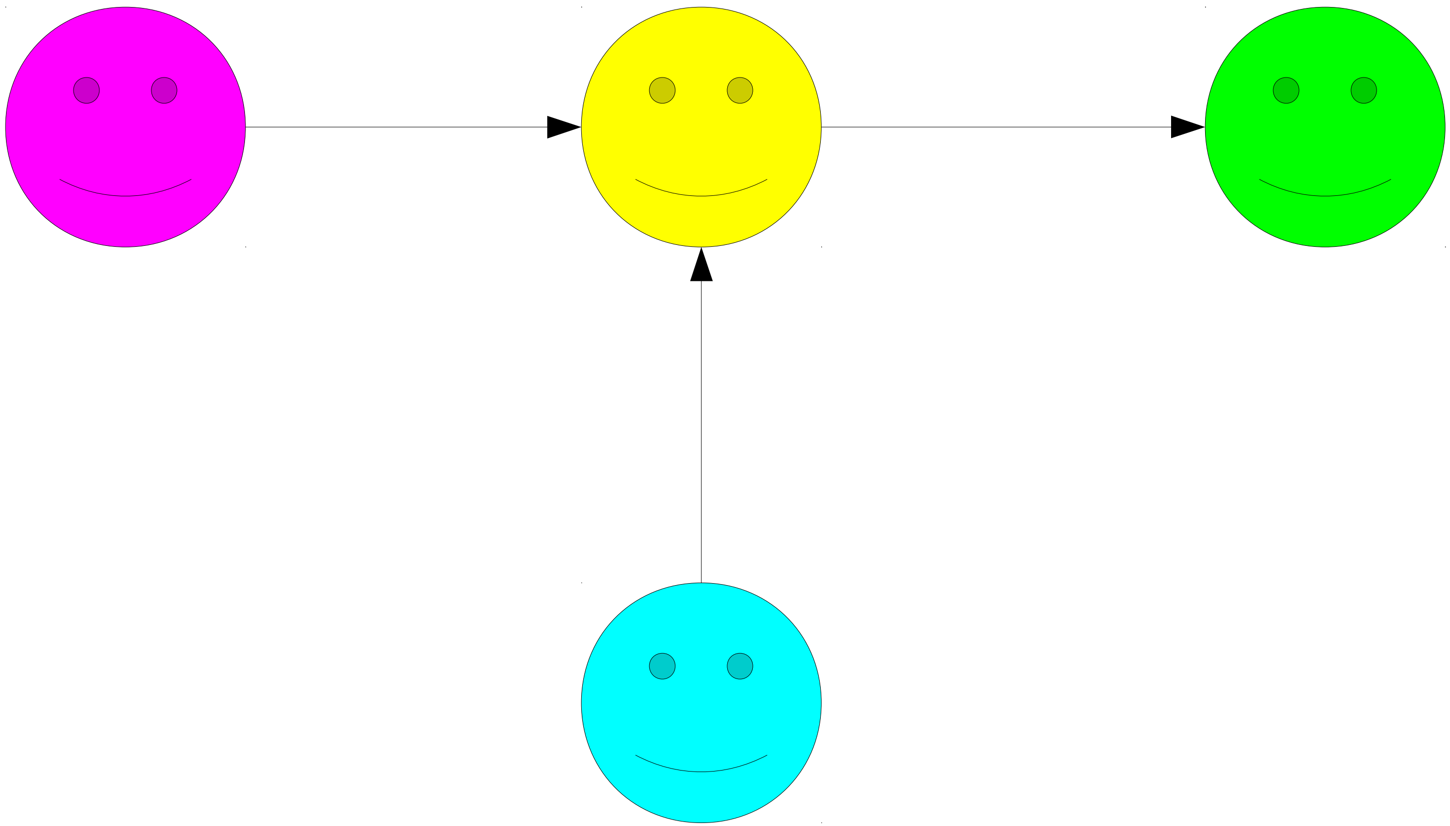
Irreflexivity Visualized



$$\forall a \in A.\ a \not{R} a$$

(*“No element is related to itself.”*)

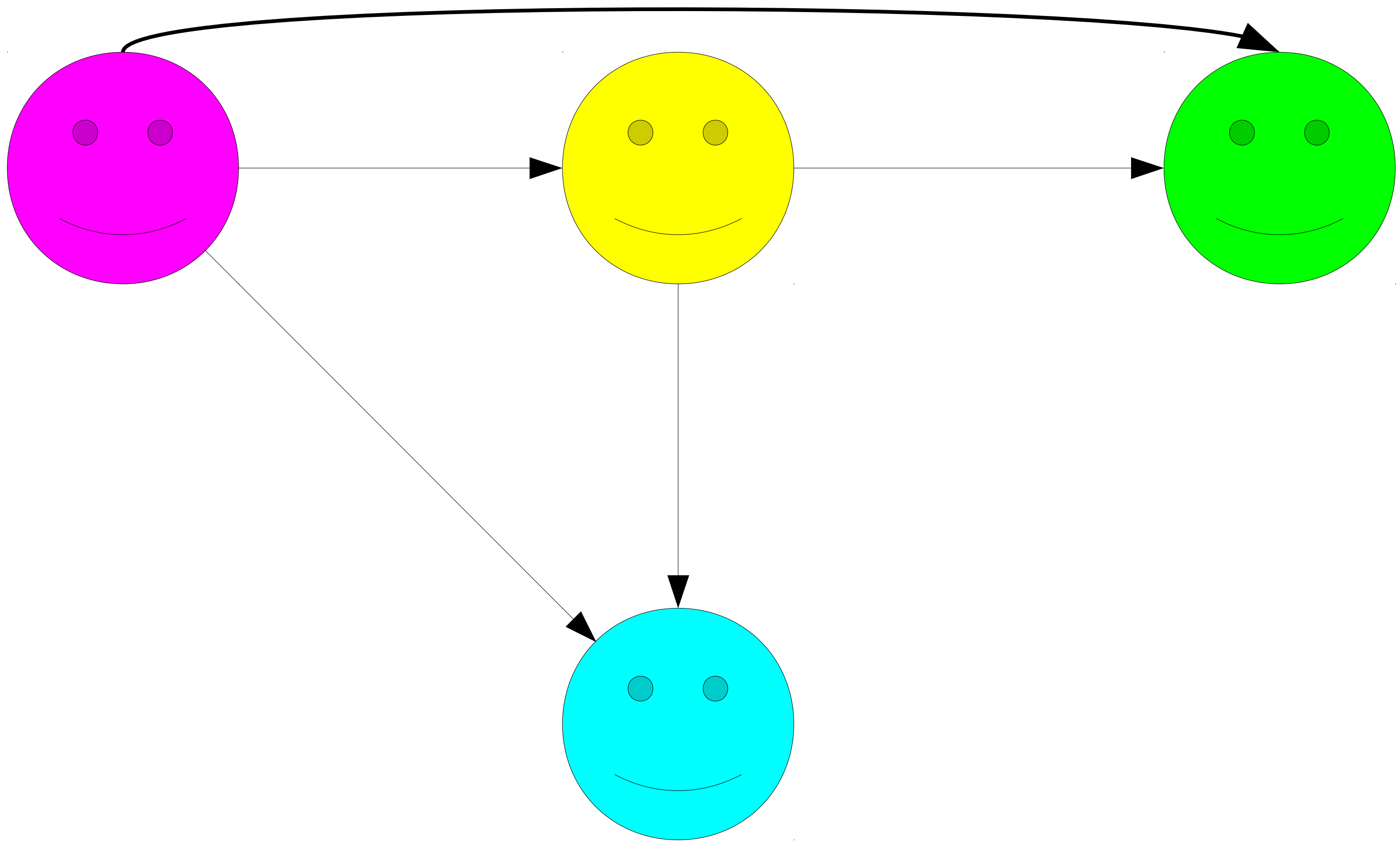
Asymmetry Visualized



$$\forall a \in A.\ \forall b \in A.\ (aRb \rightarrow b\not R a)$$

(*"If a relates to b, then b does not relate to a."*)

Transitivity Visualized



$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \land bRc \to aRc)$

(*"Whenever a is related to b and b is related to c, we know a is related to c."*)

New Stuff!

Strict Order Proofs

- Let's suppose that you're asked to prove that a binary relation is a strict order.
- Calling back to the definition, you could prove that the relation is asymmetric, irreflexive, and transitive.
- However, there's a slightly easier approach we can use instead.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

What's the high-level structure of this proof?

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

What's the high-level structure of this proof?

$$\forall R.\ (\mathit{Asymmetric}(R) \to \mathit{Irreflexive}(R))$$

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

What's the high-level structure of this proof?

$$\forall R.\ (\mathit{Asymmetric}(R) \to \mathit{Irreflexive}(R))$$

Therefore, we'll choose an arbitrary asymmetric relation R, then go and prove that R is irreflexive.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

What's the high-level structure of this proof?

$$\forall R.\ (\mathit{Asymmetric}(R) \to \mathit{Irreflexive}(R))$$

Therefore, we'll choose an arbitrary asymmetric relation R, then go and prove that R is irreflexive.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction.

What is the definition of irreflexivity?

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction.

What is the definition of irreflexivity?

$$\forall x \in A.\ x \not R x$$

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction.

What is the definition of irreflexivity?

$$\forall x \in A.\ x \not R x$$

What is the negation of this statement?

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction.

What is the definition of irreflexivity?

$$\forall x \in A.\ x \not{R} x$$

What is the negation of this statement?

$$\exists x \in A.\ xRx$$

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction.

What is the definition of irreflexivity?

$$\forall x \in A.\ x \not R x$$

What is the negation of this statement?

$$\exists x \in A.\ xRx$$

So let's suppose that there is some element $x \in A$ such that xRx and proceed from there.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction. Suppose that R is not irreflexive.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction. Suppose that R is not irreflexive. That means that there must be some $x \in A$ such that xRx.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction. Suppose that R is not irreflexive. That means that there must be some $x \in A$ such that xRx.

Since R is asymmetric, we know for any $a, b \in A$ that if aRb holds, then $b\not{R}a$ holds.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction. Suppose that R is not irreflexive. That means that there must be some $x \in A$ such that xRx.

Since R is asymmetric, we know for any $a, b \in A$ that if aRb holds, then $b\not{R}a$ holds. Plugging in $a=x$ and $b=x$, we see that if xRx holds, then $x\not{R}x$ holds.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction. Suppose that R is not irreflexive. That means that there must be some $x \in A$ such that xRx.

Since R is asymmetric, we know for any $a, b \in A$ that if aRb holds, then $b\not{R}a$ holds. Plugging in $a=x$ and $b=x$, we see that if xRx holds, then $x\not{R}x$ holds. We know by assumption that xRx is true, so we conclude that $x\not{R}x$ holds.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction. Suppose that R is not irreflexive. That means that there must be some $x \in A$ such that xRx.

Since R is asymmetric, we know for any $a, b \in A$ that if aRb holds, then $b \not{R} a$ holds. Plugging in $a = x$ and $b = x$, we see that if xRx holds, then $x \not{R} x$ holds. We know by assumption that xRx is true, so we conclude that $x \not{R} x$ holds. However, this is impossible, since we can't have both xRx and $x \not{R} x$.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction. Suppose that R is not irreflexive. That means that there must be some $x \in A$ such that xRx.

Since R is asymmetric, we know for any $a, b \in A$ that if aRb holds, then $b\not{R}a$ holds. Plugging in $a = x$ and $b = x$, we see that if xRx holds, then $x\not{R}x$ holds. We know by assumption that xRx is true, so we conclude that $x\not{R}x$ holds. However, this is impossible, since we can't have both xRx and $x\not{R}x$.

We have reached a contradiction, so our assumption must have been wrong.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction. Suppose that R is not irreflexive. That means that there must be some $x \in A$ such that xRx.

Since R is asymmetric, we know for any $a, b \in A$ that if aRb holds, then $b\not{R}a$ holds. Plugging in $a = x$ and $b = x$, we see that if xRx holds, then $x\not{R}x$ holds. We know by assumption that xRx is true, so we conclude that $x\not{R}x$ holds. However, this is impossible, since we can't have both xRx and $x\not{R}x$.

We have reached a contradiction, so our assumption must have been wrong. Thus R must be irreflexive.

Theorem: Let R be a binary relation over a set A. If R is asymmetric, then R is irreflexive.

Proof: Let R be an arbitrary asymmetric binary relation over a set A. We will prove that R is irreflexive.

To do so, we will proceed by contradiction. Suppose that R is not irreflexive. That means that there must be some $x \in A$ such that xRx.

Since R is asymmetric, we know for any $a, b \in A$ that if aRb holds, then $b\not{R}a$ holds. Plugging in $a=x$ and $b=x$, we see that if xRx holds, then $x\not{R}x$ holds. We know by assumption that xRx is true, so we conclude that $x\not{R}x$ holds. However, this is impossible, since we can't have both xRx and $x\not{R}x$.

We have reached a contradiction, so our assumption must have been wrong. Thus R must be irreflexive. ■

Theorem: If a binary relation R is asymmetric and transitive, then R is a strict order.

Proof: Let R be a binary relation that is asymmetric and transitive. Since R is asymmetric, by our previous theorem we know that R is also irreflexive. Therefore, R is asymmetric, irreflexive, and transitive, so by definition R is a strict order. ■

To prove that some binary relation R is a strict order, you can just prove that R is asymmetric and transitive. In the next problem set, you'll see an even simpler technique!

Drawing Strict Orders

2012 Summer Olympics

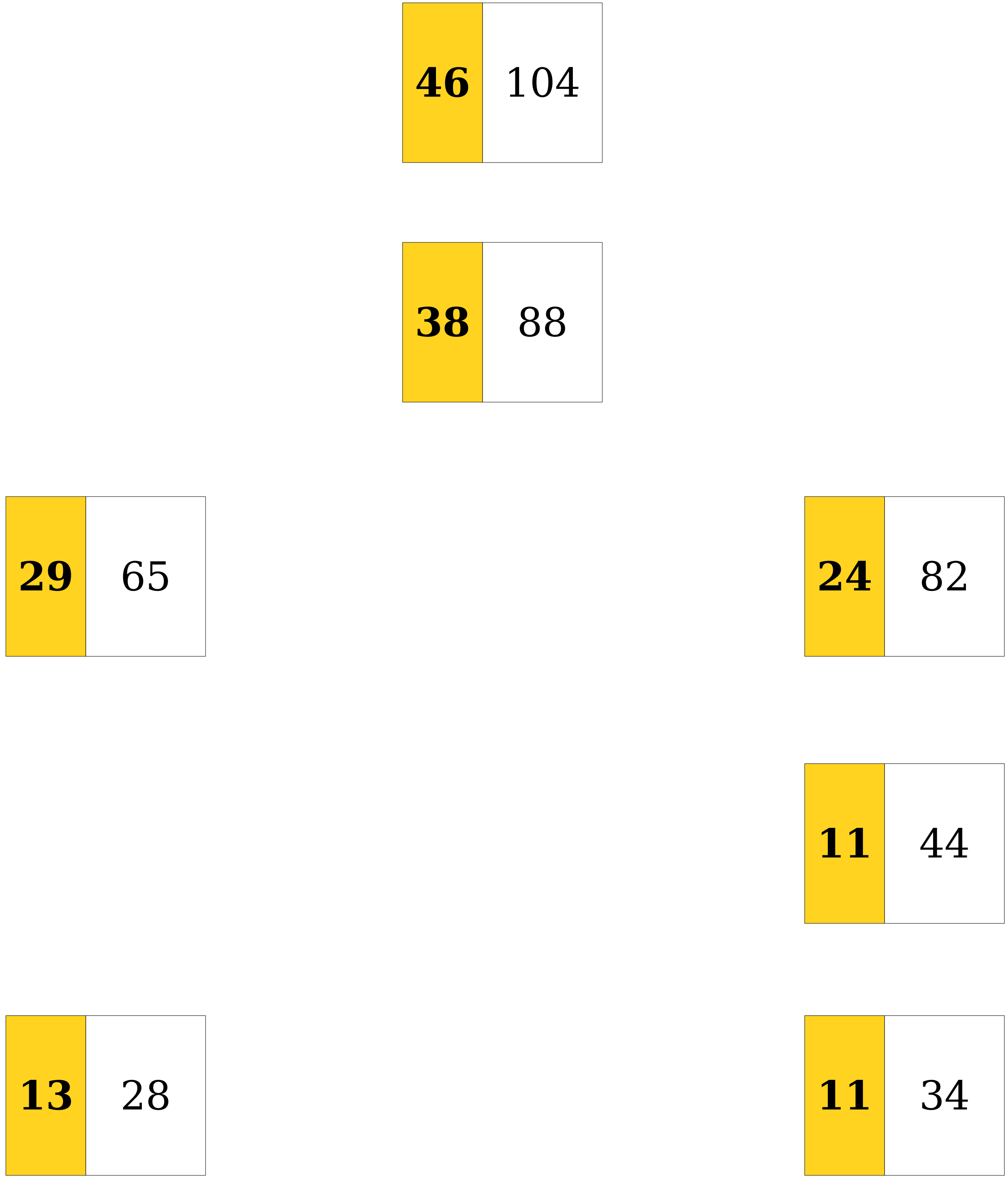
Gold	Silver	Bronze	Total
46	29	29	104
38	27	23	88
29	17	19	65
24	26	32	82
13	8	7	28
11	19	14	44
11	11	12	34

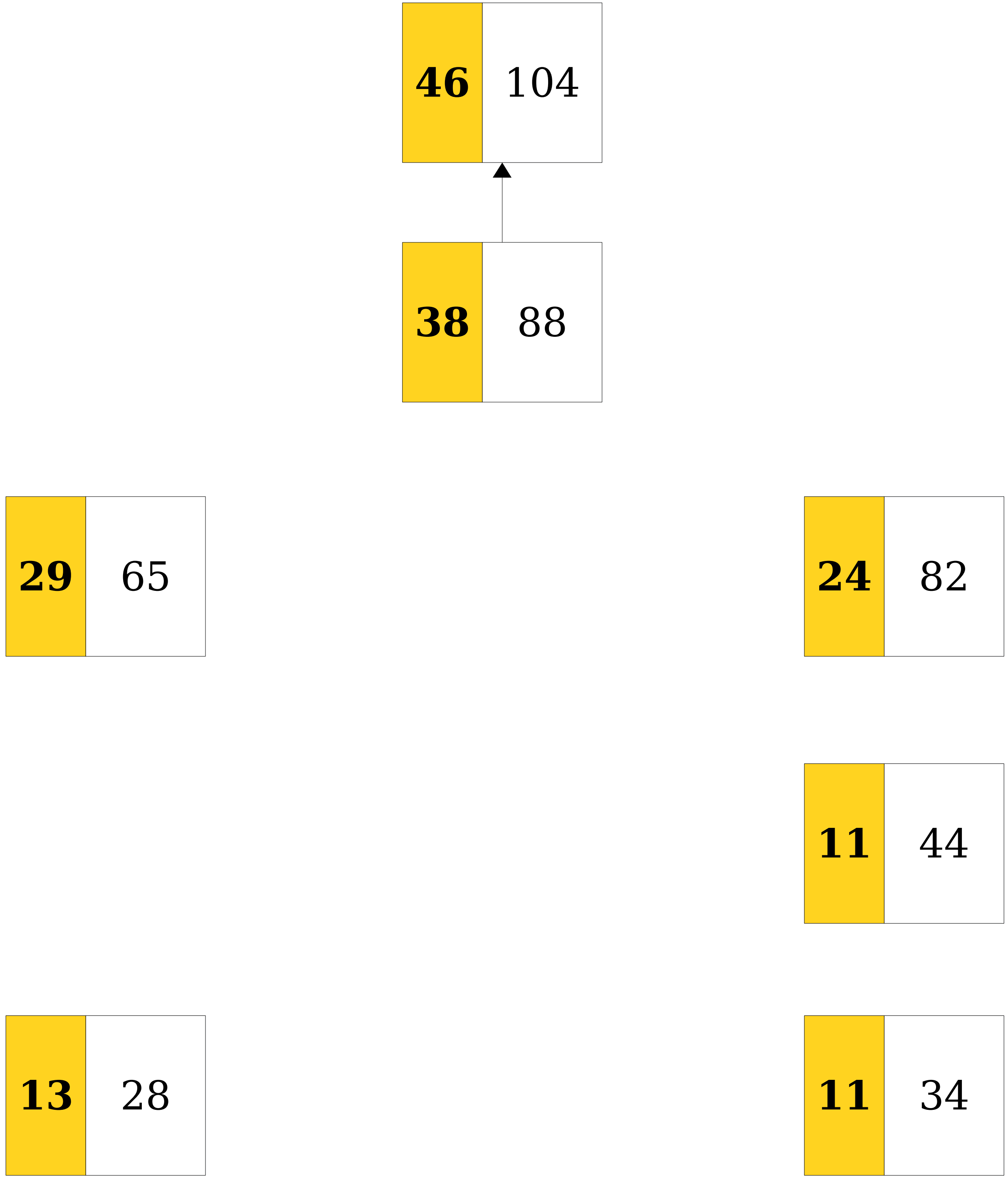
Inspired by http://tartarus.org/simon/2008-olympics-hasse/
Data from http://www.london2012.com/medals/medal-count/

2012 Summer Olympics

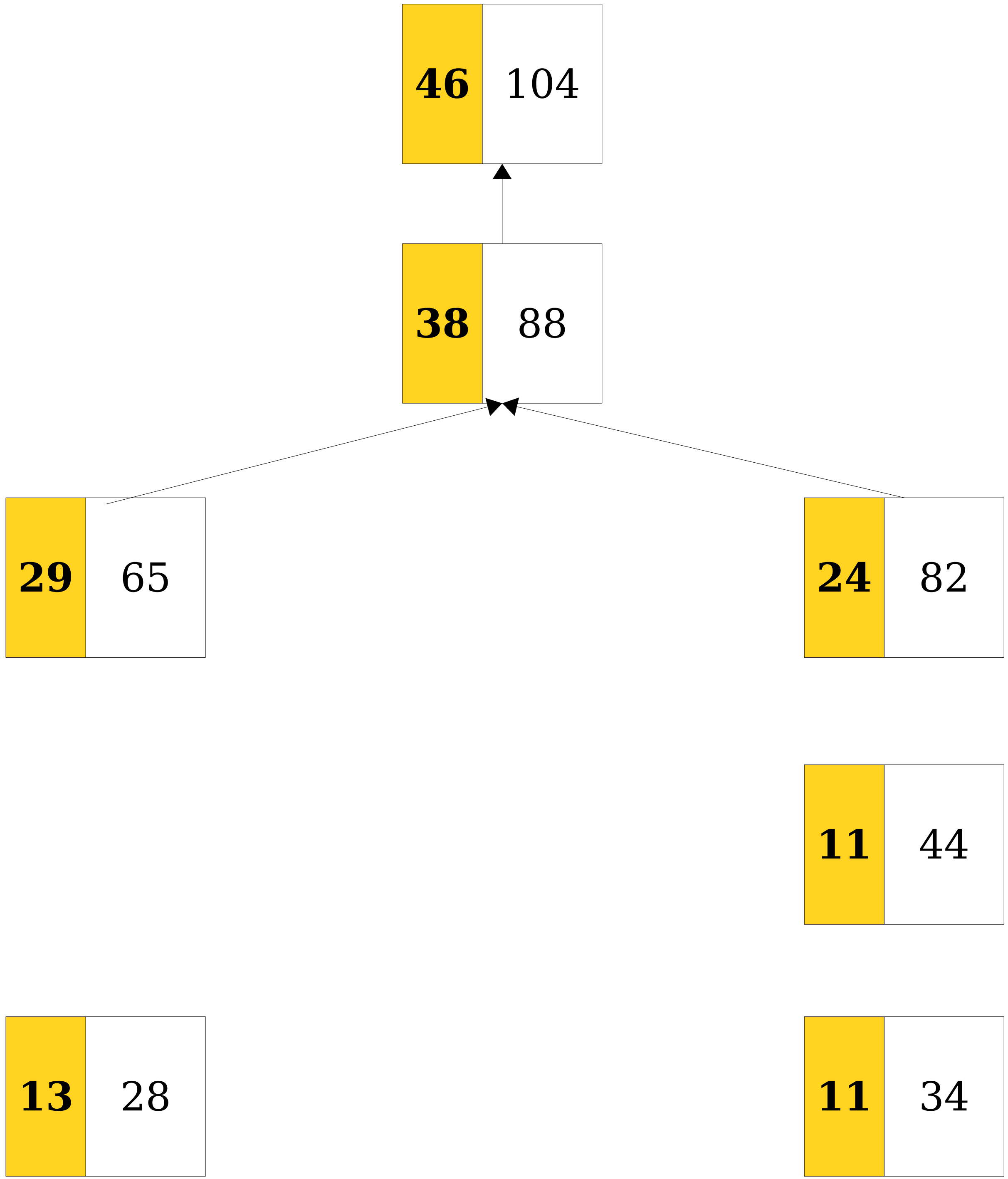
Gold	Silver	Bronze	Total
46	29	29	104
38	27	23	88
29	17	19	65
24	26	32	82
13	8	7	28
11	19	14	44
11	11	12	34

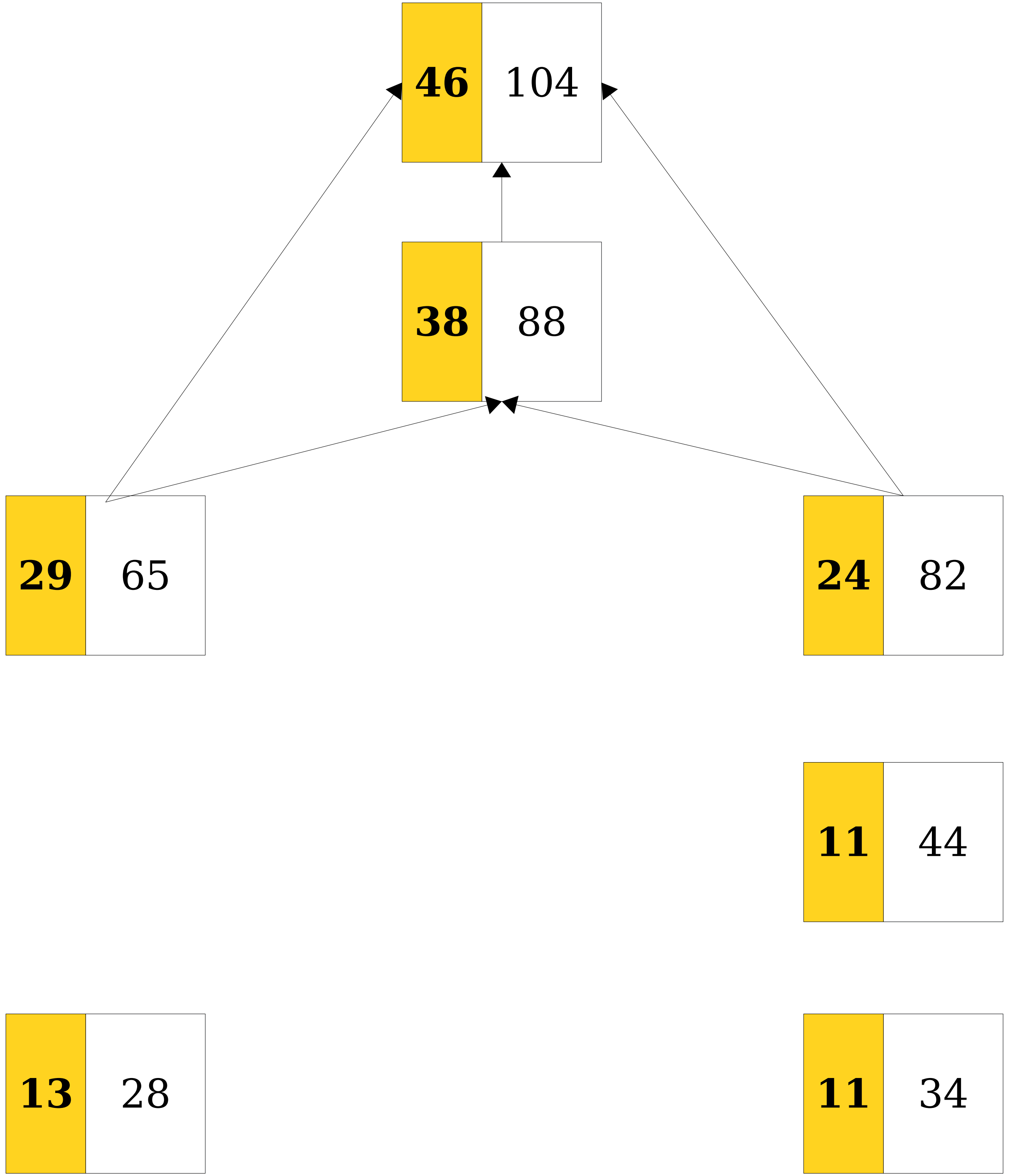
Inspired by http://tartarus.org/simon/2008-olympics-hasse/
Data from http://www.london2012.com/medals/medal-count/


46 104
38 88
29 65
24 82
11 44
13 28
11 34

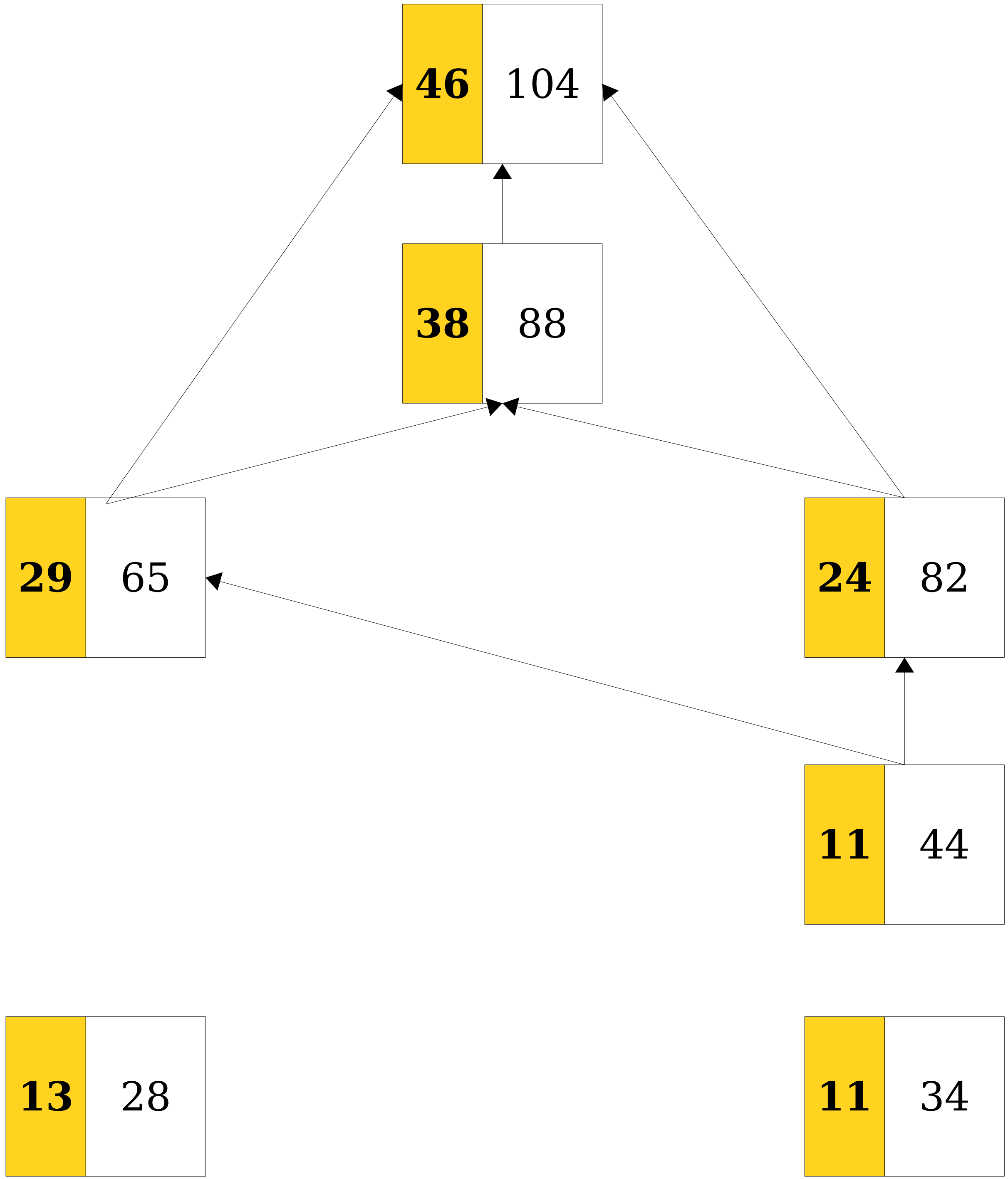


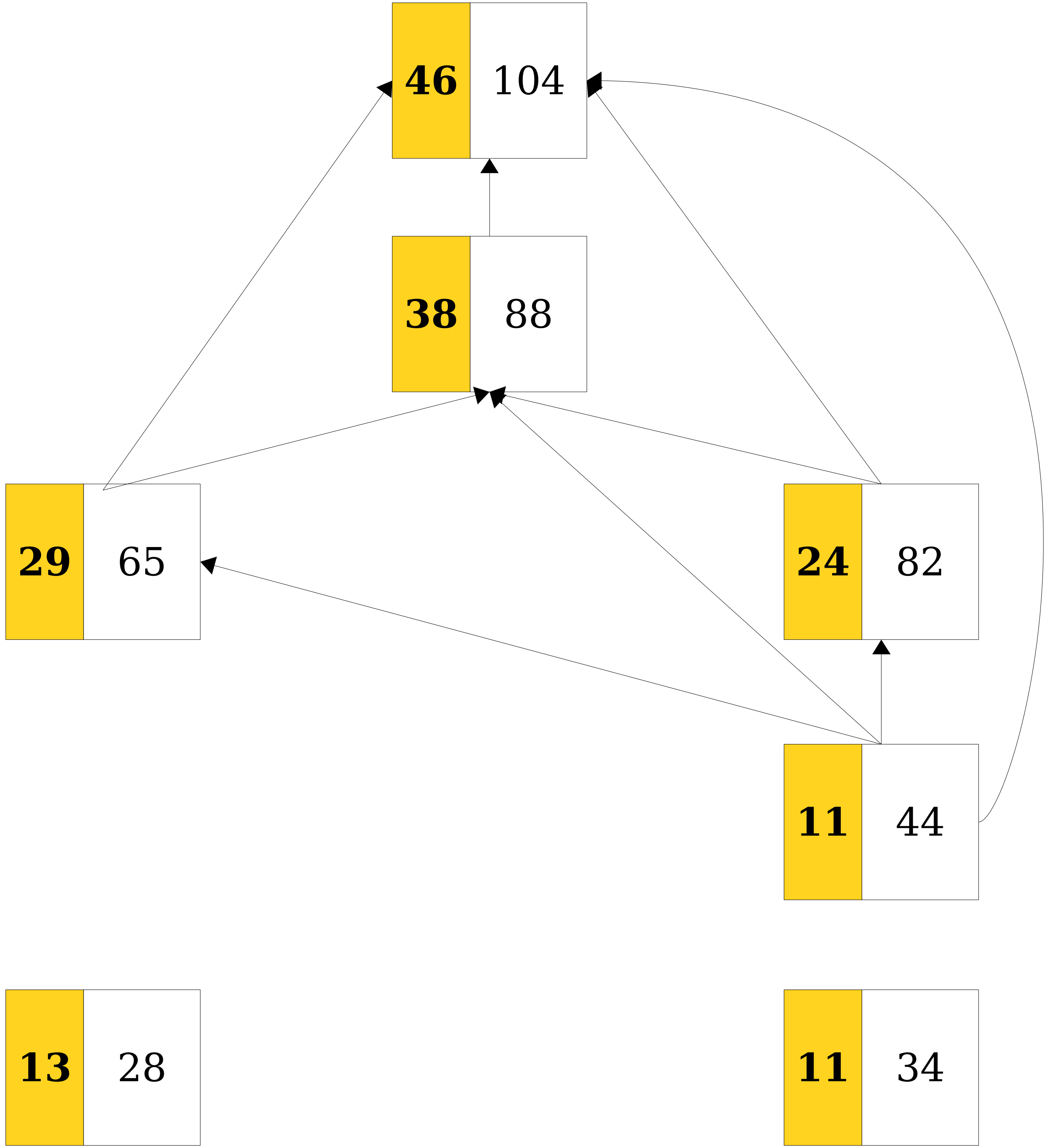
46
104
38
88
29
65
24
82
11
44
13
28
11
34

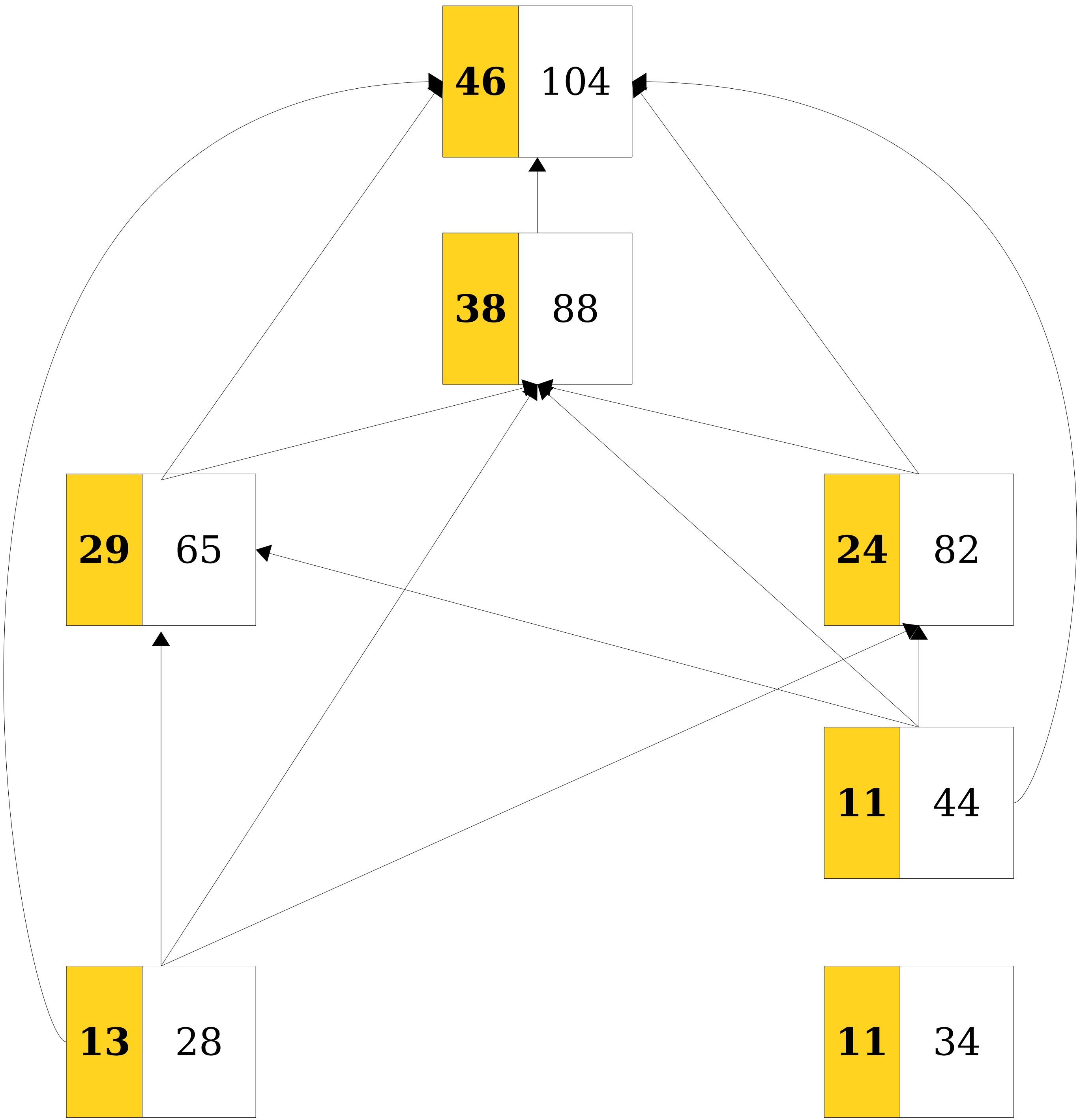

46 104
38 88
29 65
24 82
11 44
13 28
11 34



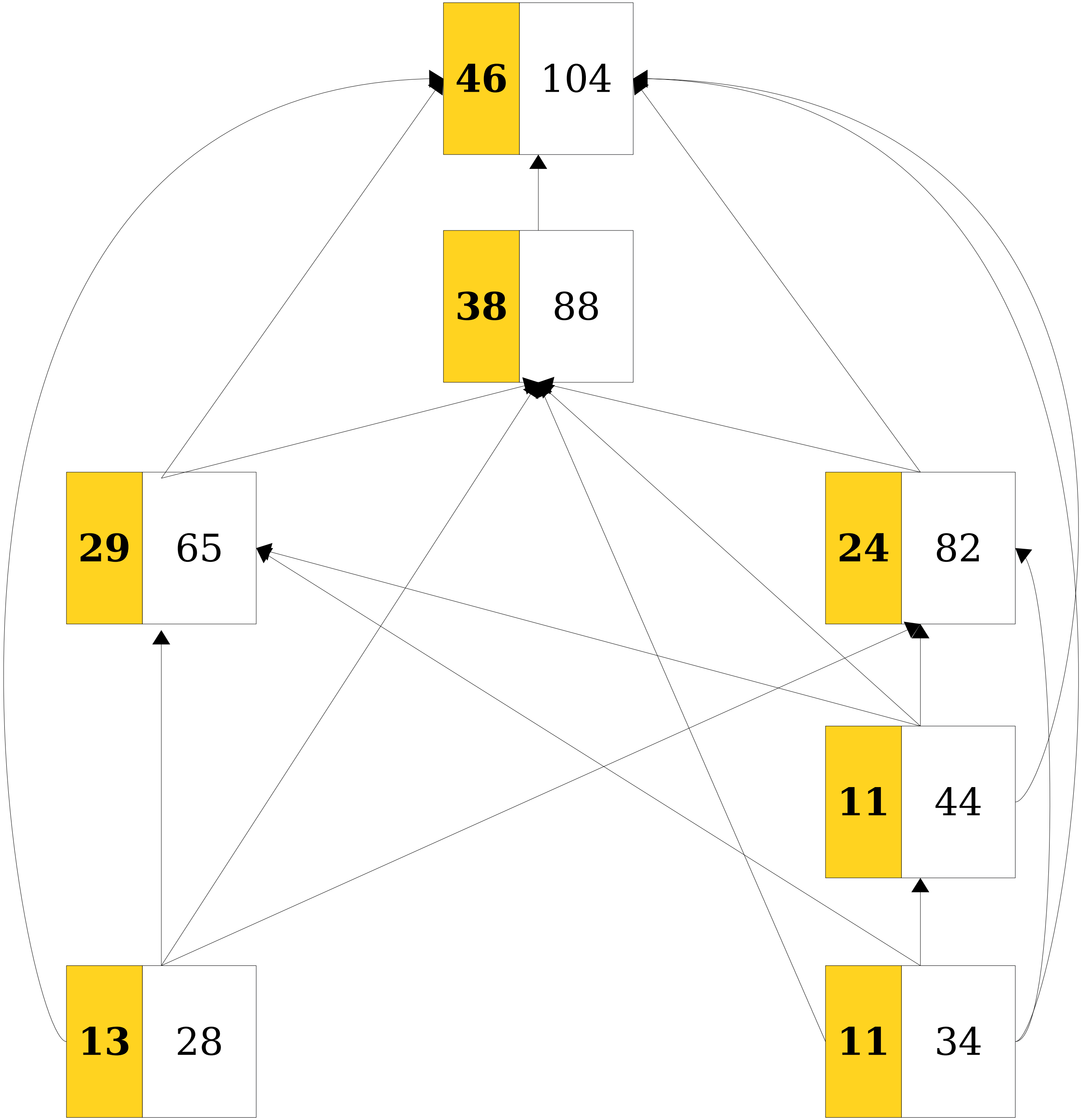
46 104
38 88
29 65
24 82
11 44
13 28
11 34

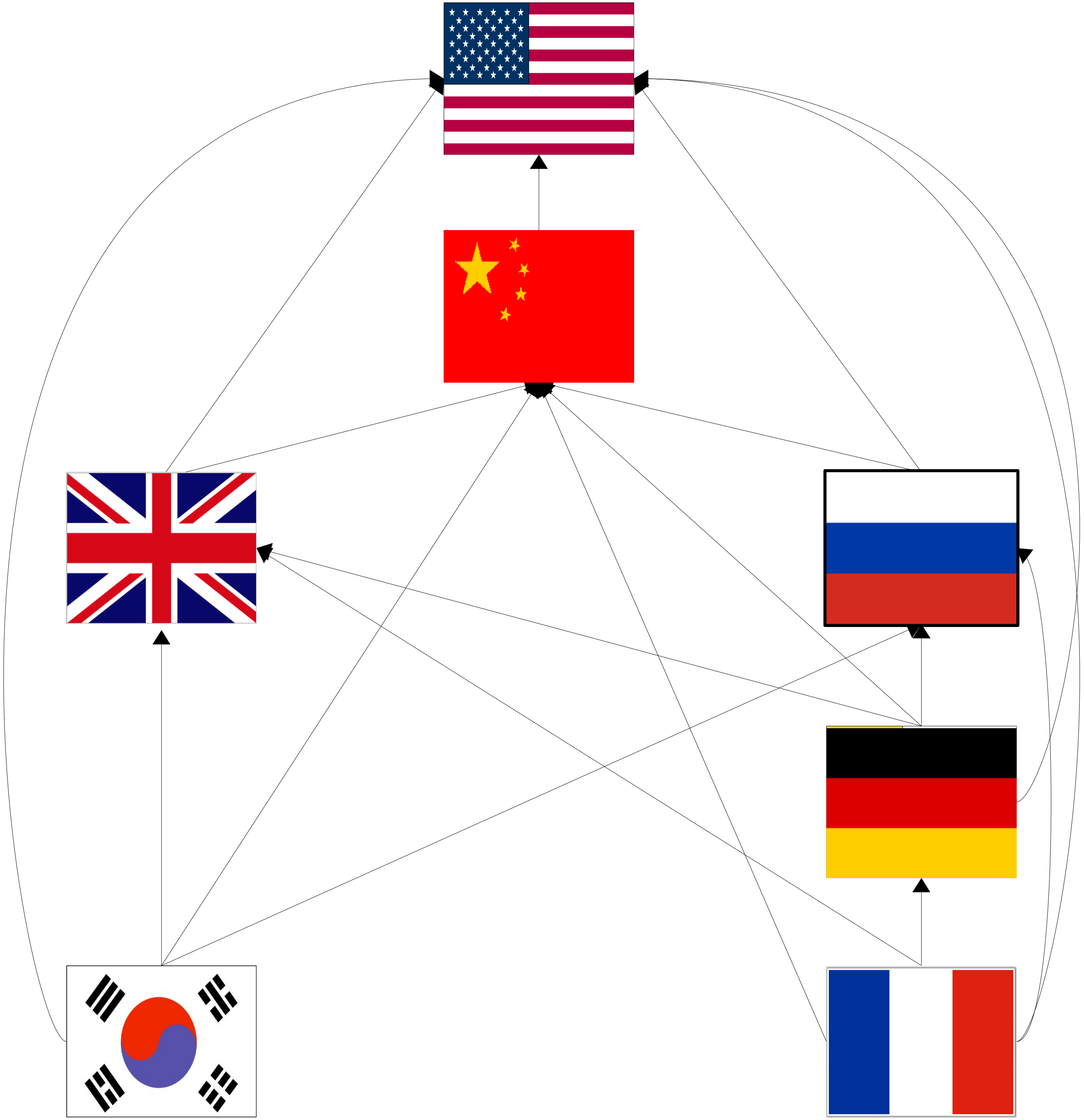

46
104
38
88
29
65
24
82
11
44
13
28
11
34

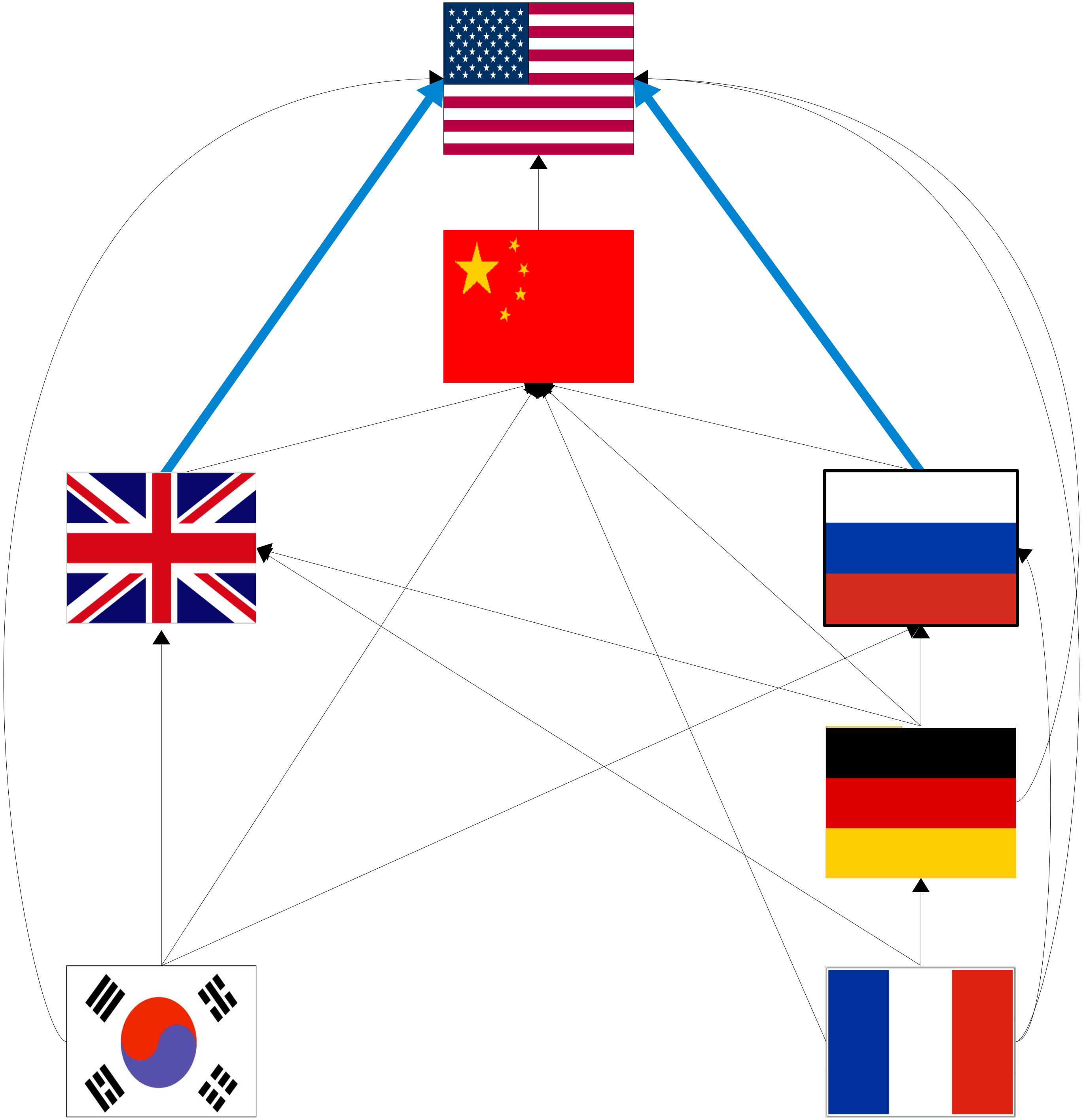

46
104
38
88
29
65
24
82
11
44
13
28
11
34

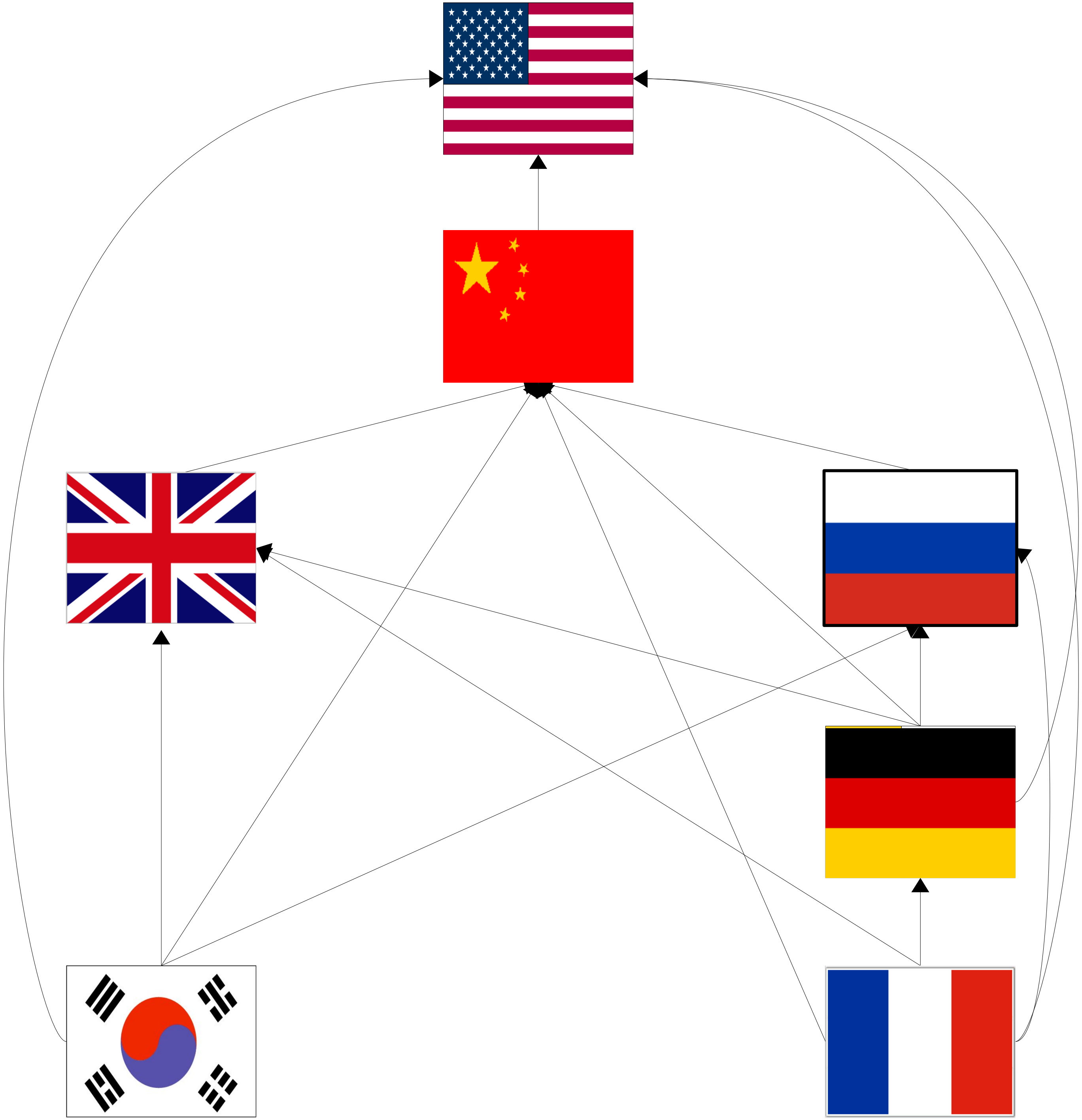


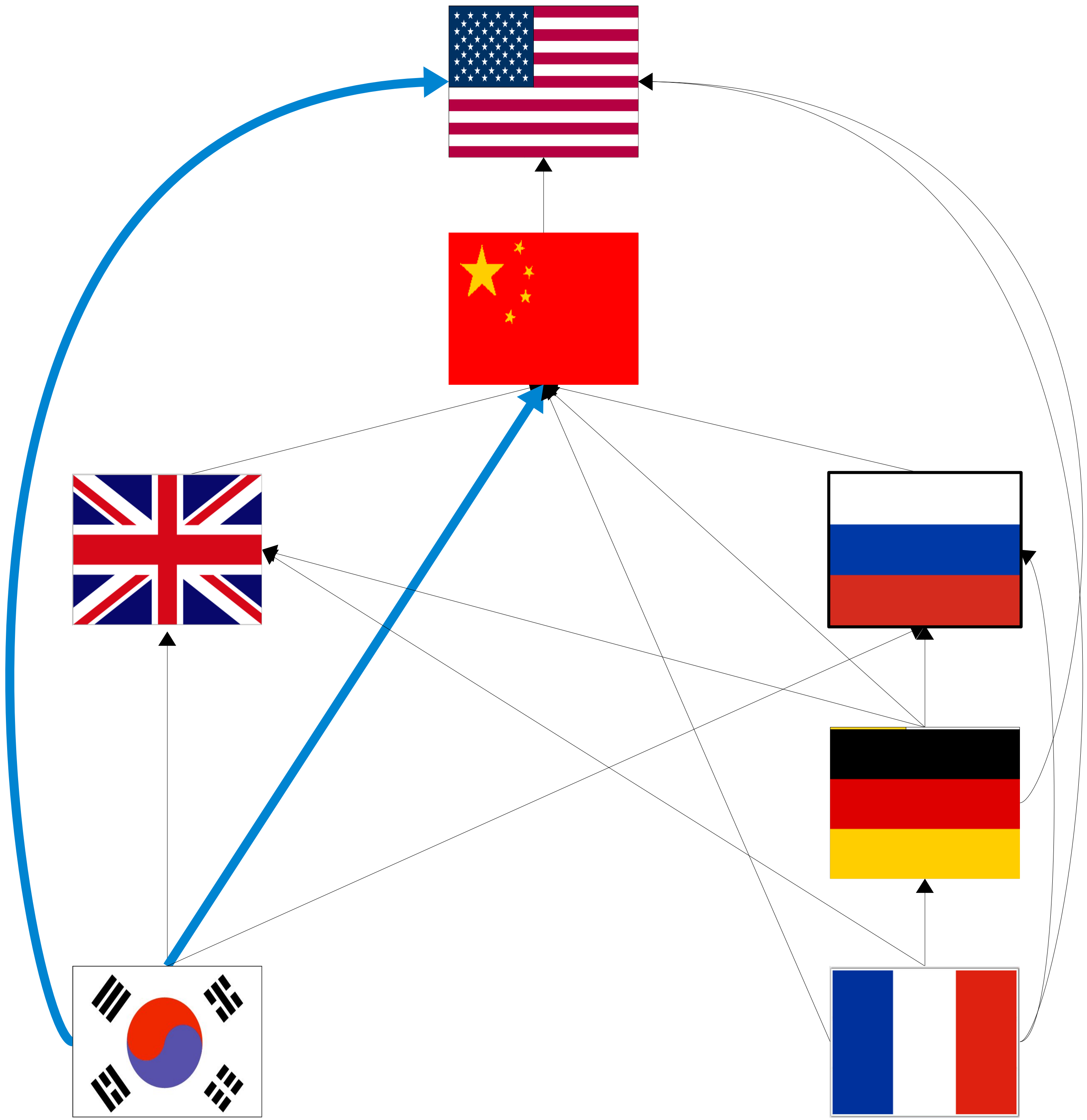
46
104
38
88
29
65
24
82
11
44
13
28
11
34

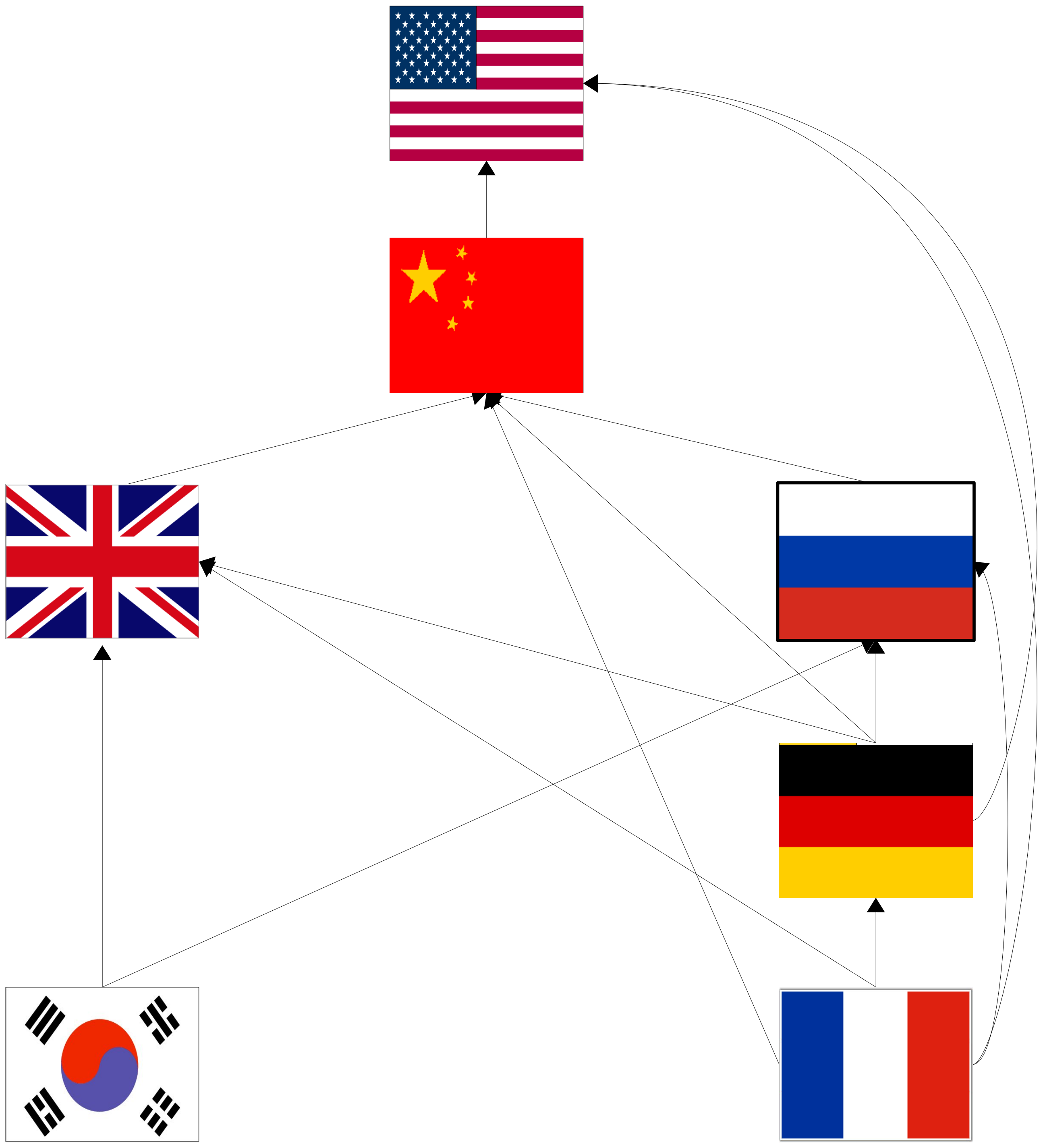

46
104
38
88
29
65
24
82
11
44
13
28
11
34

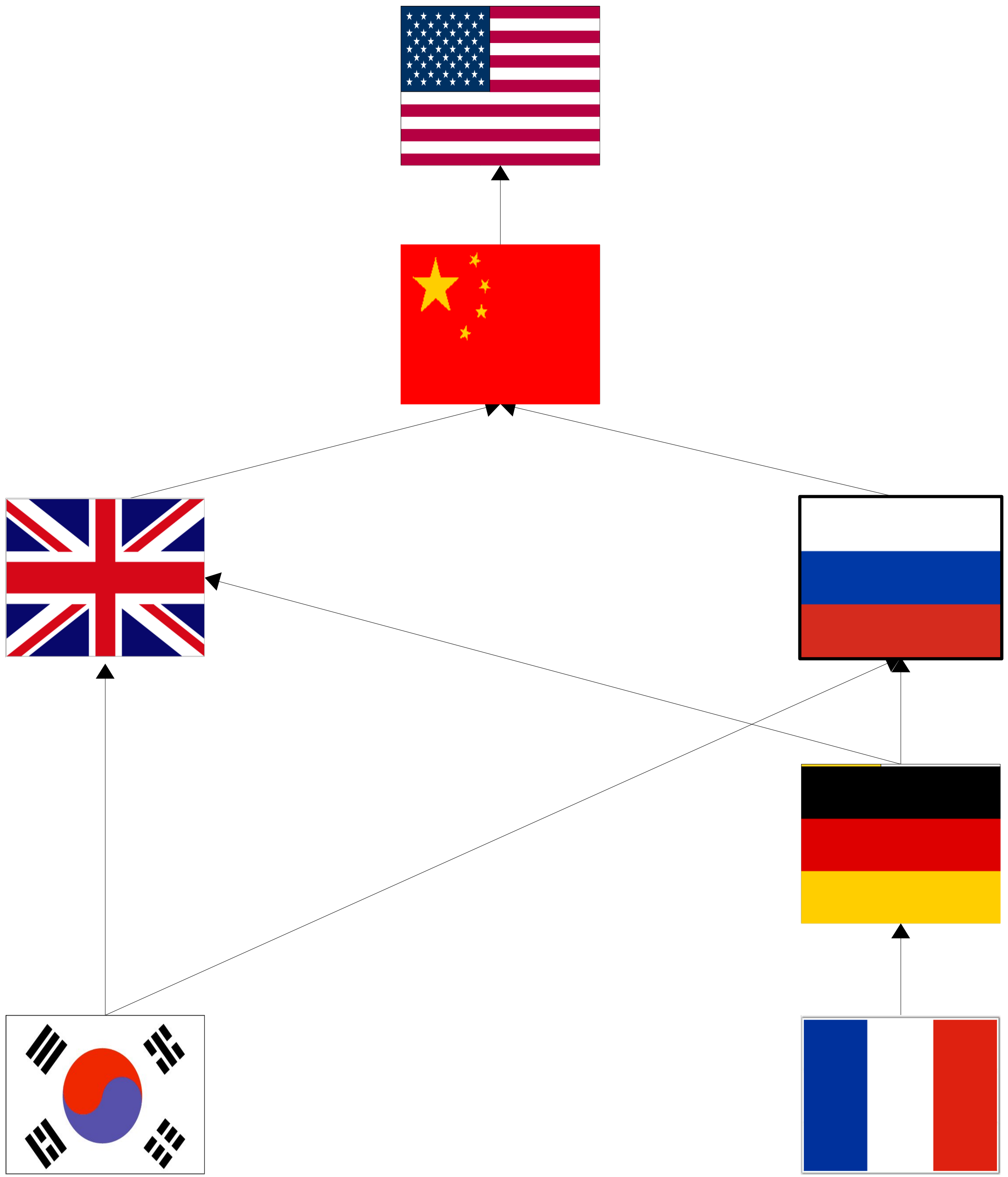


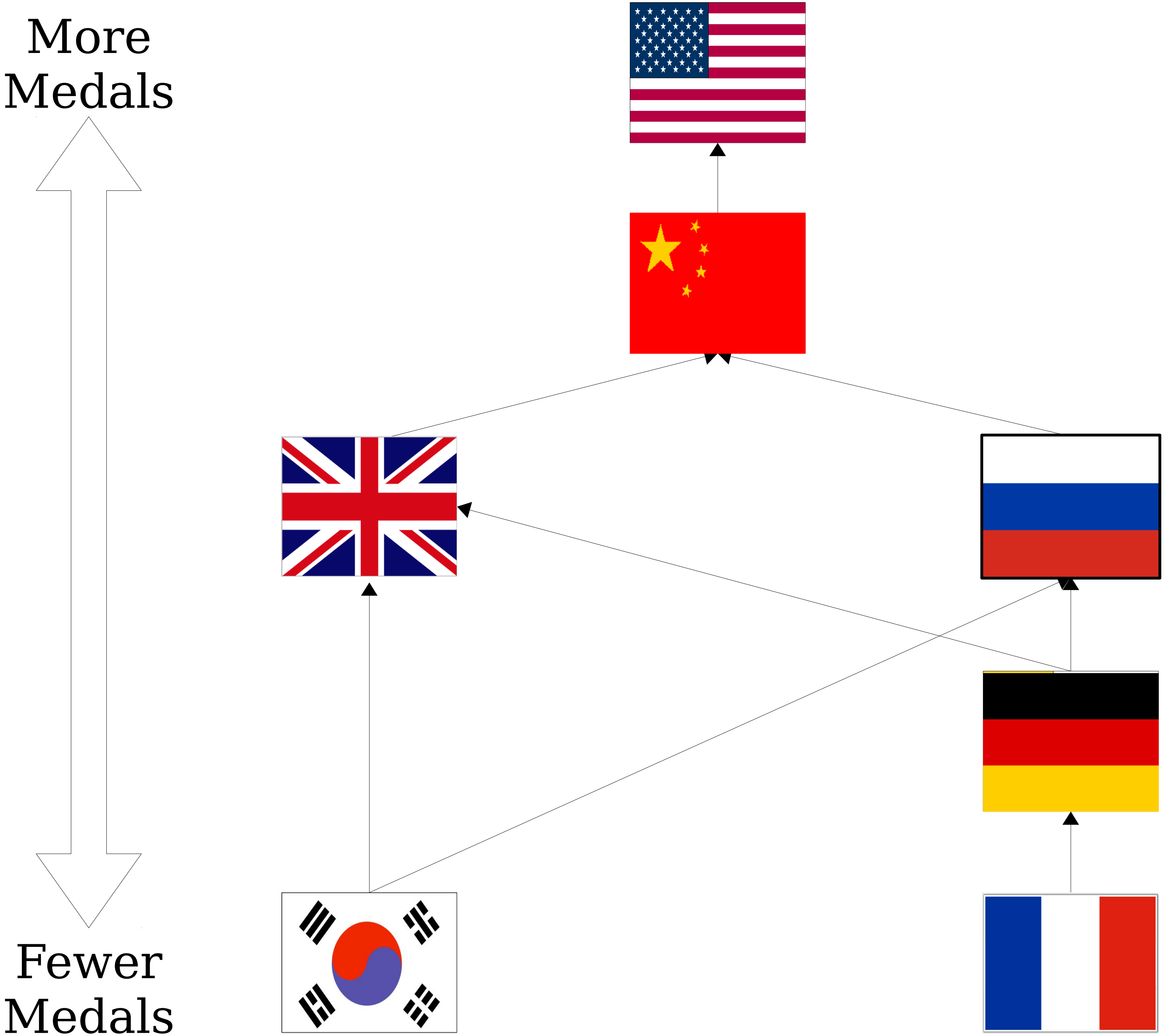


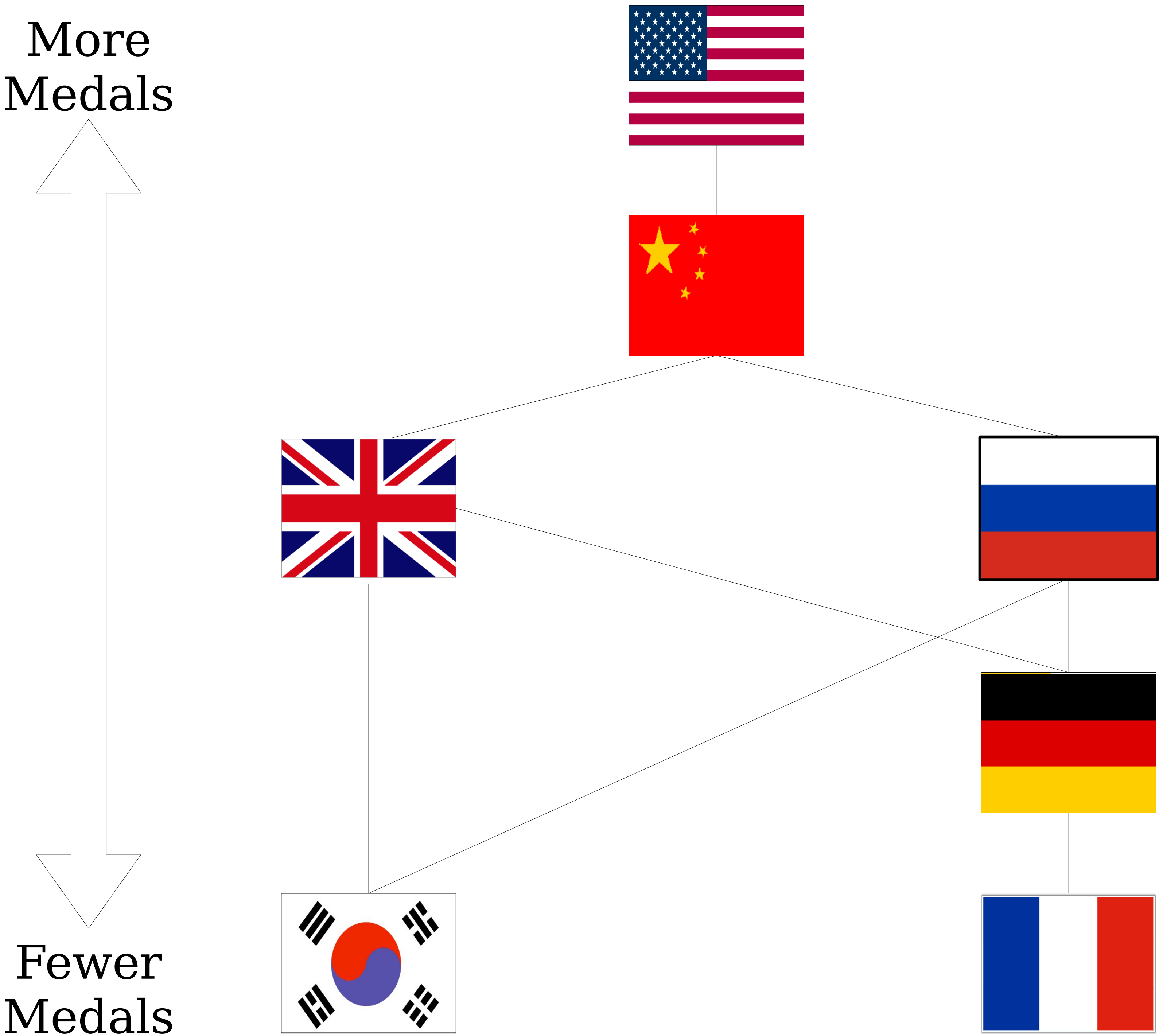








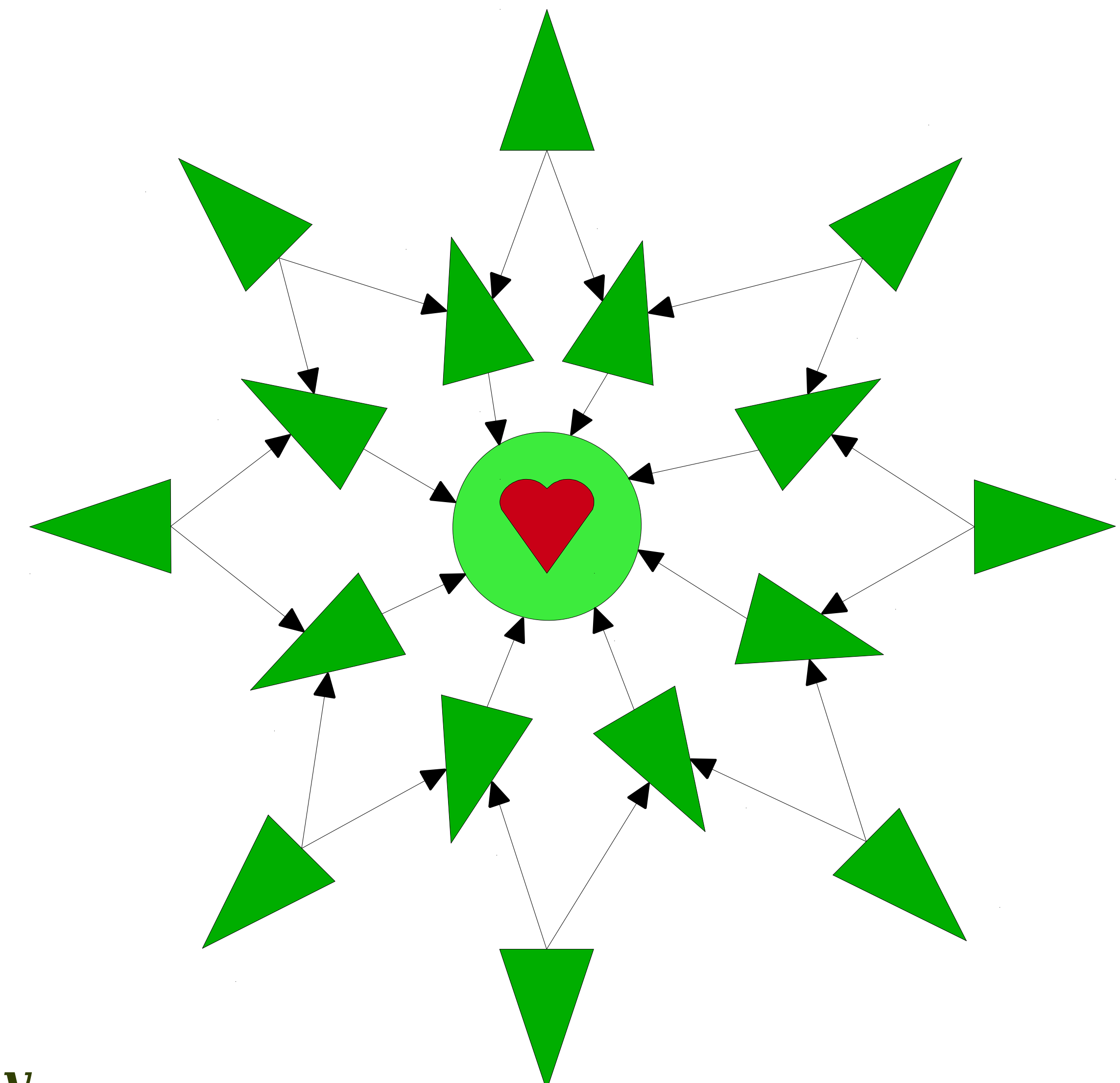

More Medals
Fewer Medals


More Medals
Fewer Medals

Hasse Diagrams

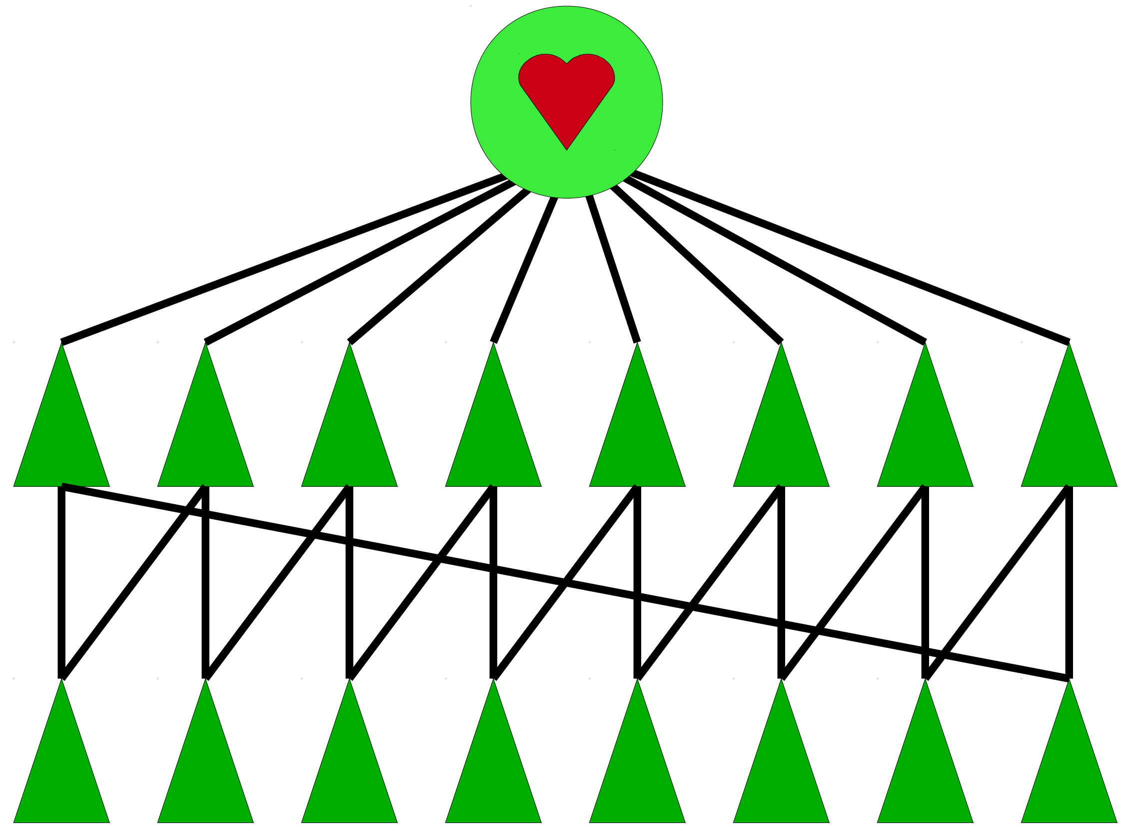
- A ***Hasse diagram*** is a graphical representation of a strict order.
- Elements are drawn from bottom-to-top.
- Higher elements are bigger than lower elements: by ***asymmetry***, the edges can only go in one direction.
- No redundant edges: by ***transitivity***, we can infer the missing edges.

Hasse Artichokes



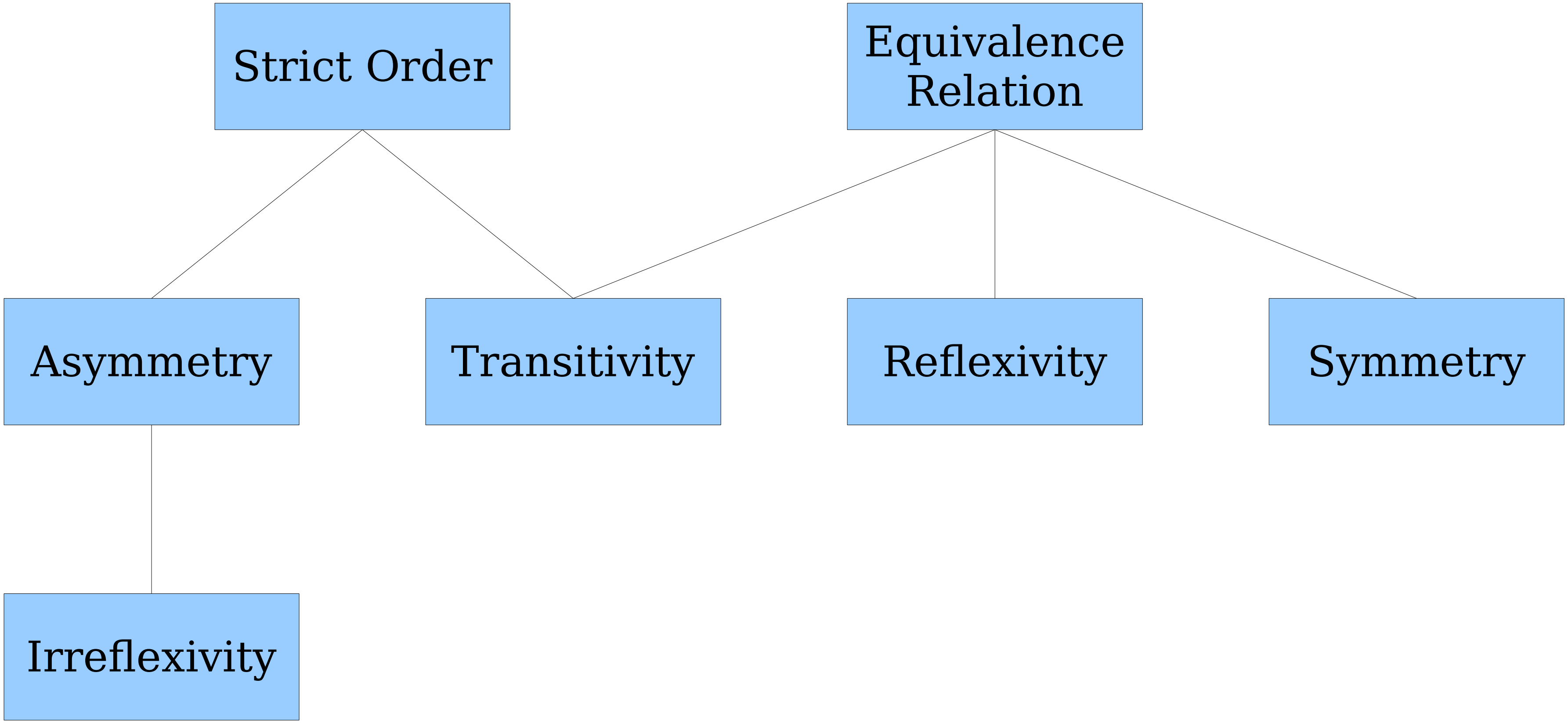
xRy if x must be eaten before y

Hasse Artichokes



xRy* if *x must be eaten before y

The Meta Strict Order



aRb **if** ***a* is less specific than *b***

The Binary Relation Editor

Time-Out for Announcements!

Recommended Courses

- I asked you for input on interesting and exciting humanities and social science courses. Here's the list of classes you recommended:
  - ***"The American West"*** (heavily cross-listed).
  - ***"Rock, Sex, and Rebellion"*** (Music 8A).
  - ***"History of South Africa"*** (History 147).
  - ***"Gandhi in His Time and Ours"*** (History 196).
  - ***"World History of Science"*** (History 140).
  - ***"Introduction to the Visual Arts"*** (ArtHist 1B).
  - "***Creative Nonfiction"*** (English 91).
- Have anything else to recommend? Let me know!

PS3 Checkpoints

- The PS3 checkpoint has also been graded and feedback should be up on GradeScope.
- So... 10% of you didn't submit the checkpoint. Was that intentional? If so, let us know why!

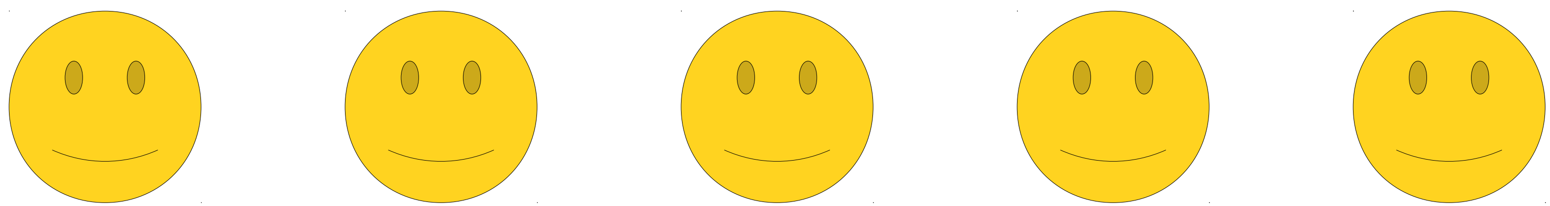
Problem Set Two

- Problem Set Two has been graded. Your feedback should be available on GradeScope right now.
  - Late submissions will be graded by tomorrow afternoon.
- You are strongly encouraged to read over the solutions set for PS2 and to make sure you understand all the feedback you received – this stuff is tricky!

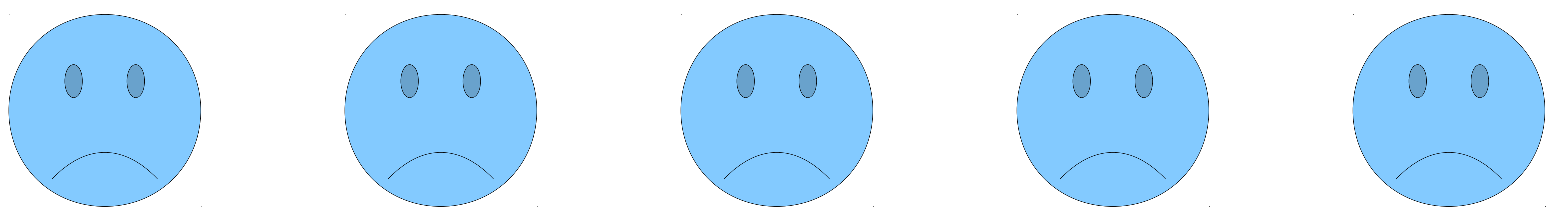
Common Mistakes on PS2

$$\exists p \in P.\ (Happy(p) \to \forall q \in P.\ Happy(q))$$

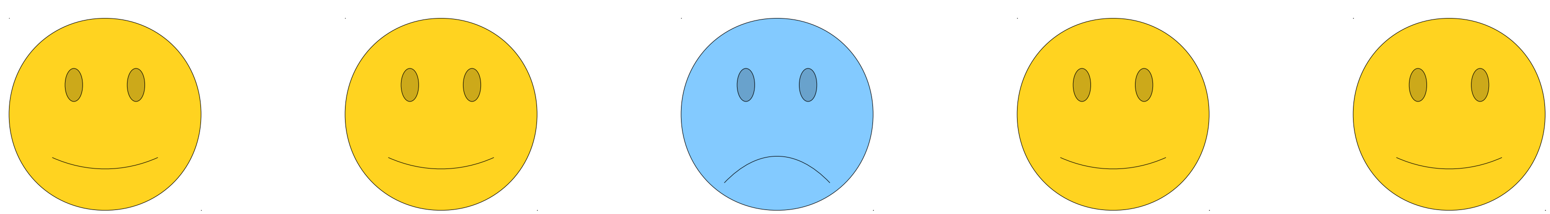
Case 1



Case 2

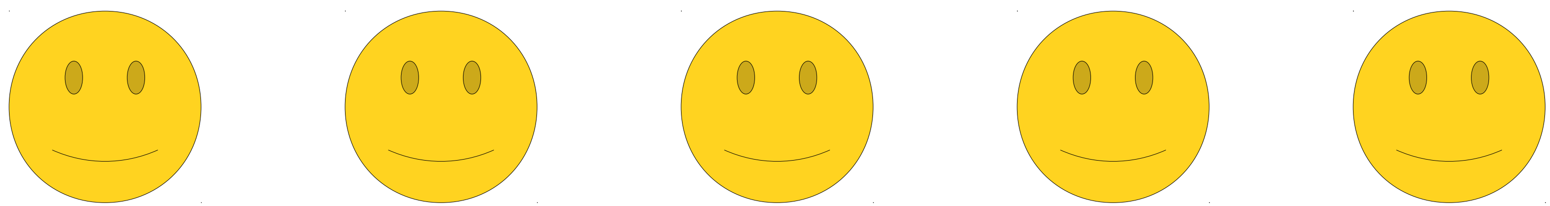


Case 3

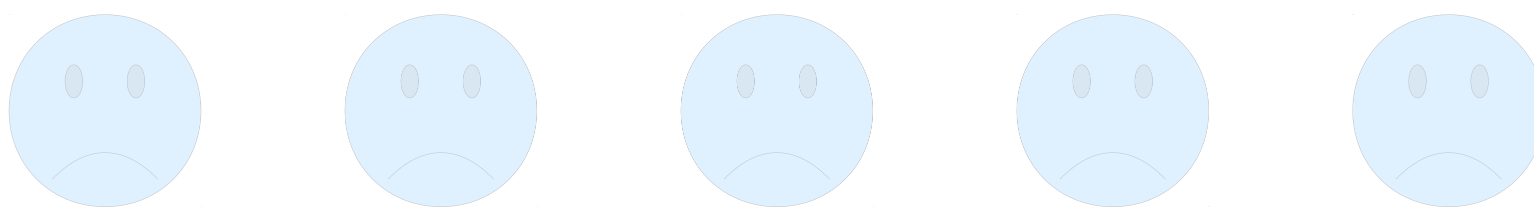


$$\exists p \in P.\ (Happy(p) \rightarrow \forall q \in P.\ Happy(q))$$

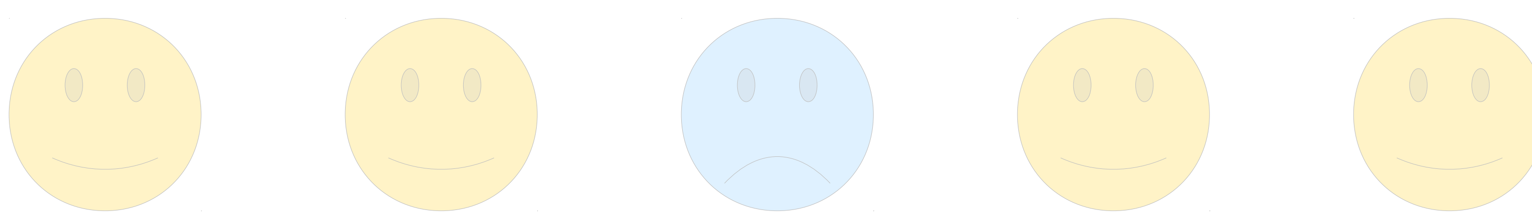
Case 1



Case 2



Case 3

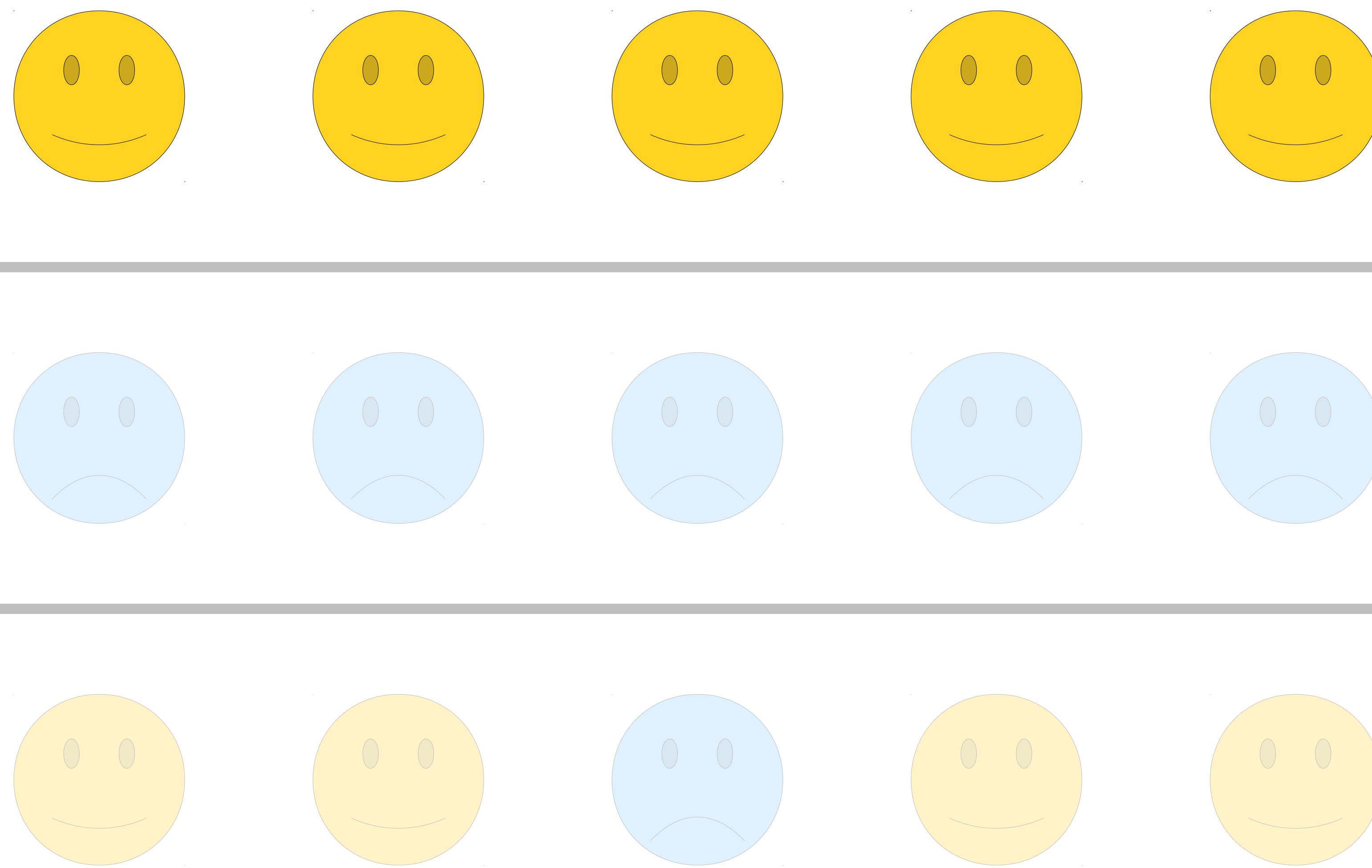


$$\exists p \in P.\ (Happy(p) \to \forall q \in P.\ Happy(q))$$

Case 1

Case 2

Case 3

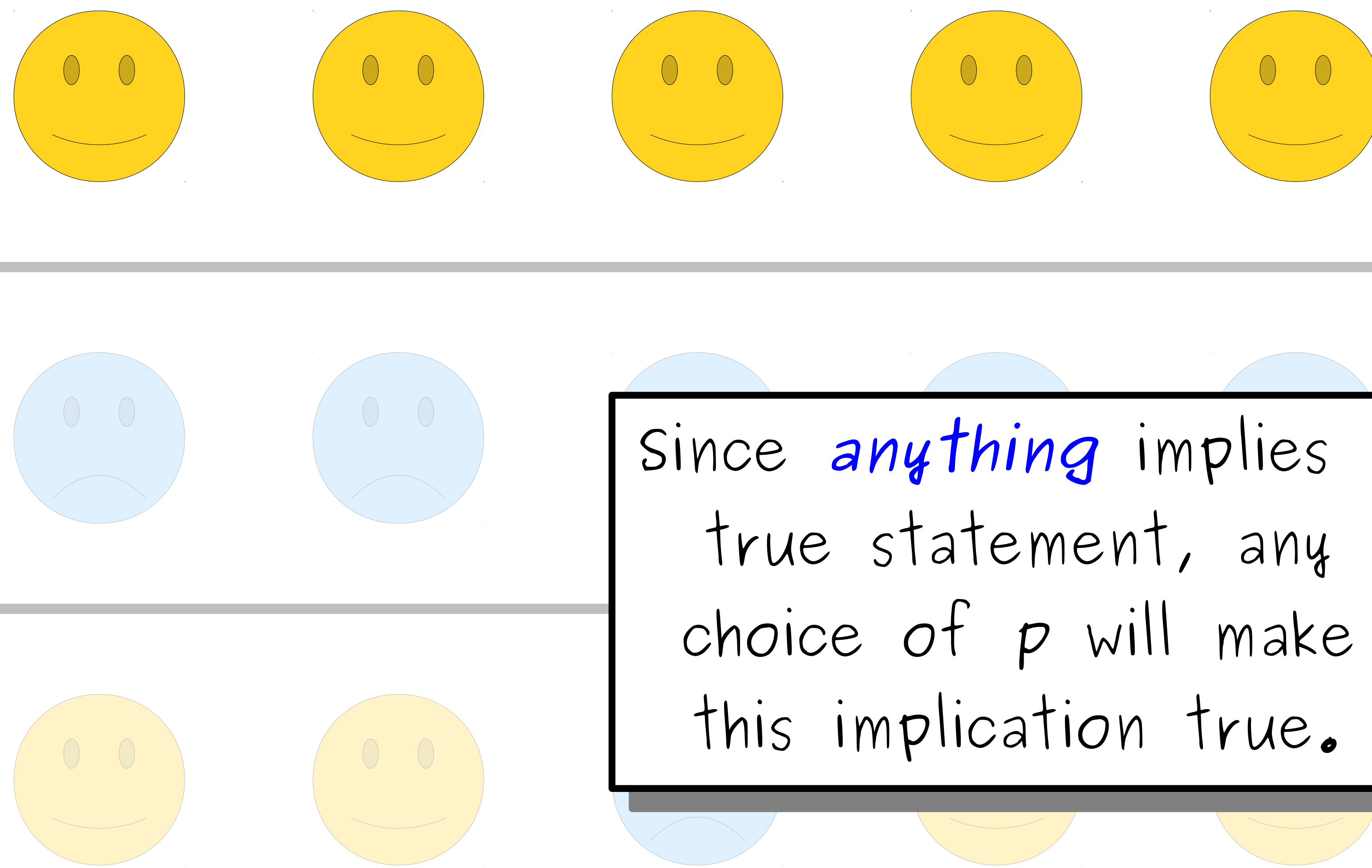


$$\exists p \in P.\ (Happy(p) \to \forall q \in P.\ Happy(q))$$

Case 1

Case 2

Case 3



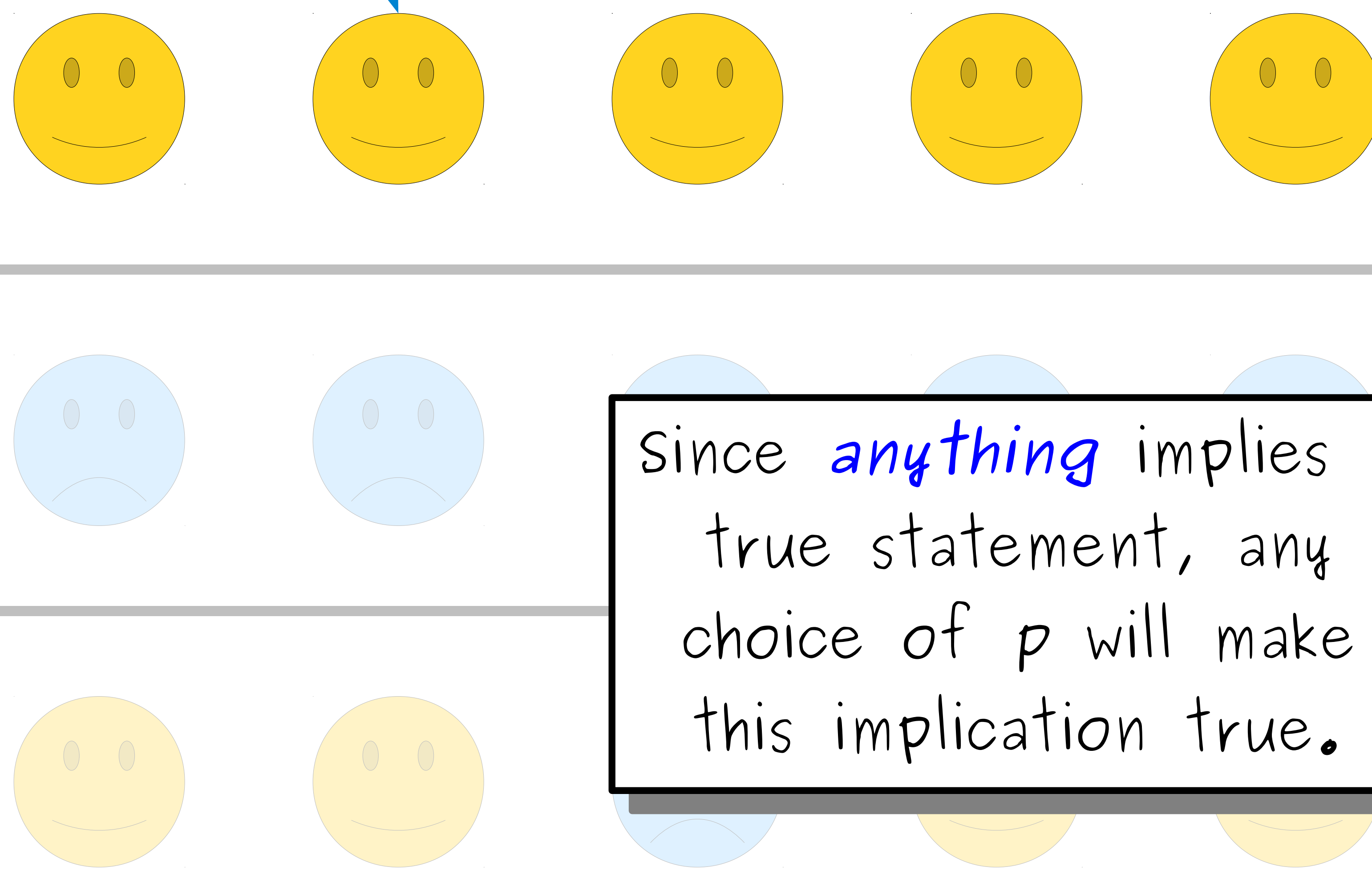
Since *anything* implies a true statement, any choice of p will make this implication true.

$$\exists p \in P.\ (Happy(p) \to \forall q \in P.\ Happy(q))$$

Case 1

Case 2

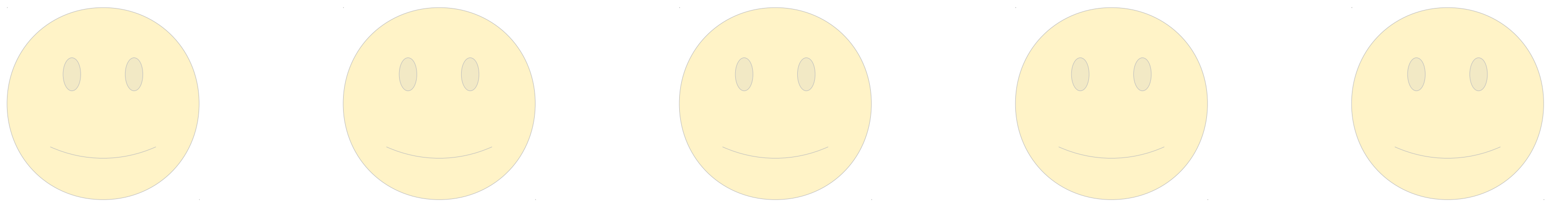
Case 3



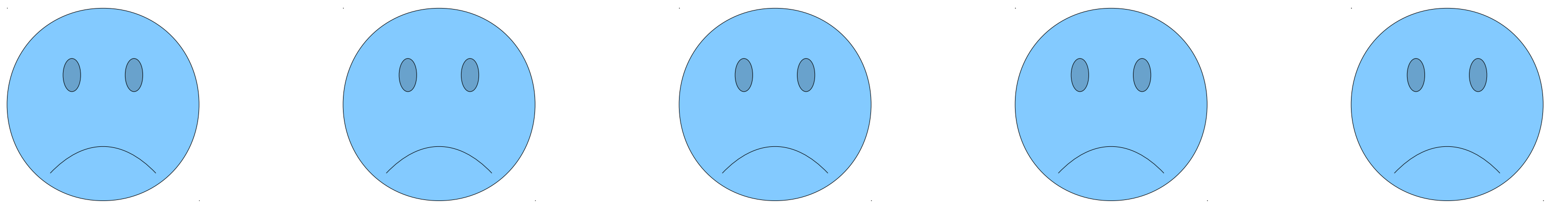
Since *anything* implies a true statement, any choice of p will make this implication true.

$$\exists p \in P.\ (Happy(p) \to \forall q \in P.\ Happy(q))$$

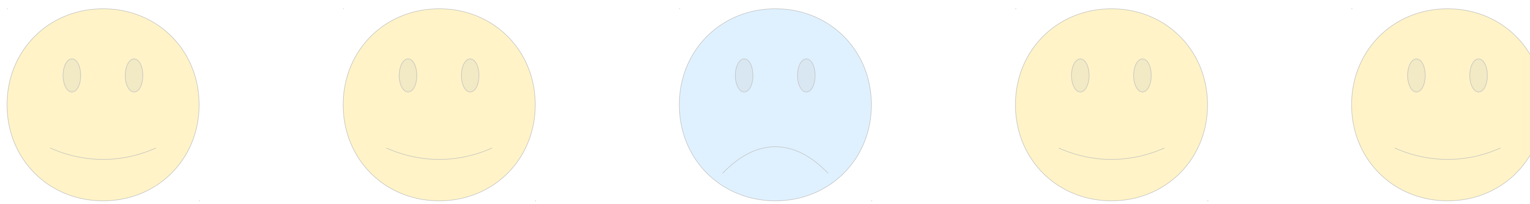
Case 1



Case 2



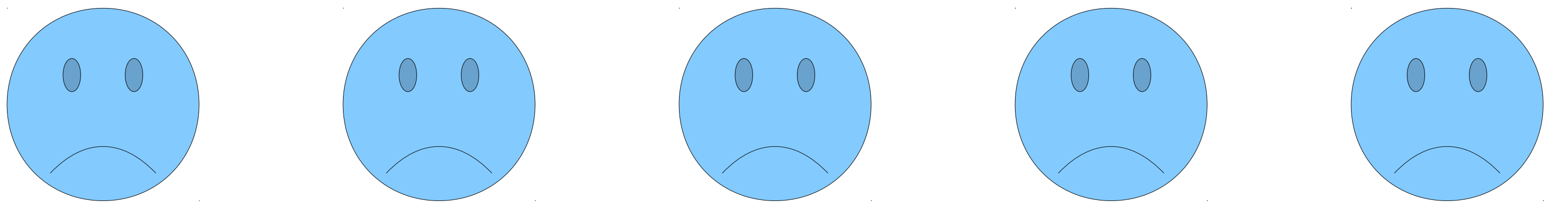
Case 3



$$\exists p \in P.\ (Happy(p) \rightarrow \forall q \in P.\ Happy(q))$$

Case 1

Case 2



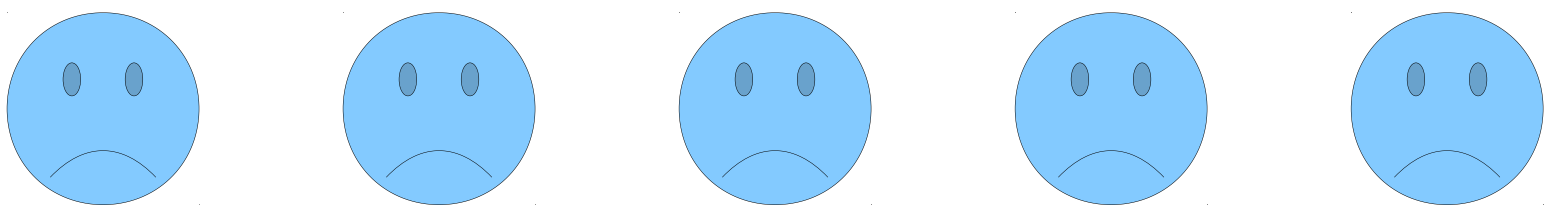
Case 3

$$\exists p \in P.\ (Happy(p) \to {\color{red}\forall q \in P.\ Happy(q)})$$

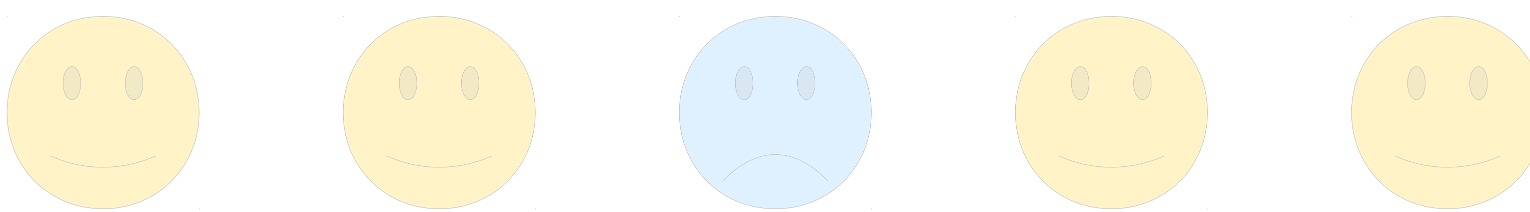
This implication will only be true if we can choose someone for p who isn't happy. Can we do that?

Case 1

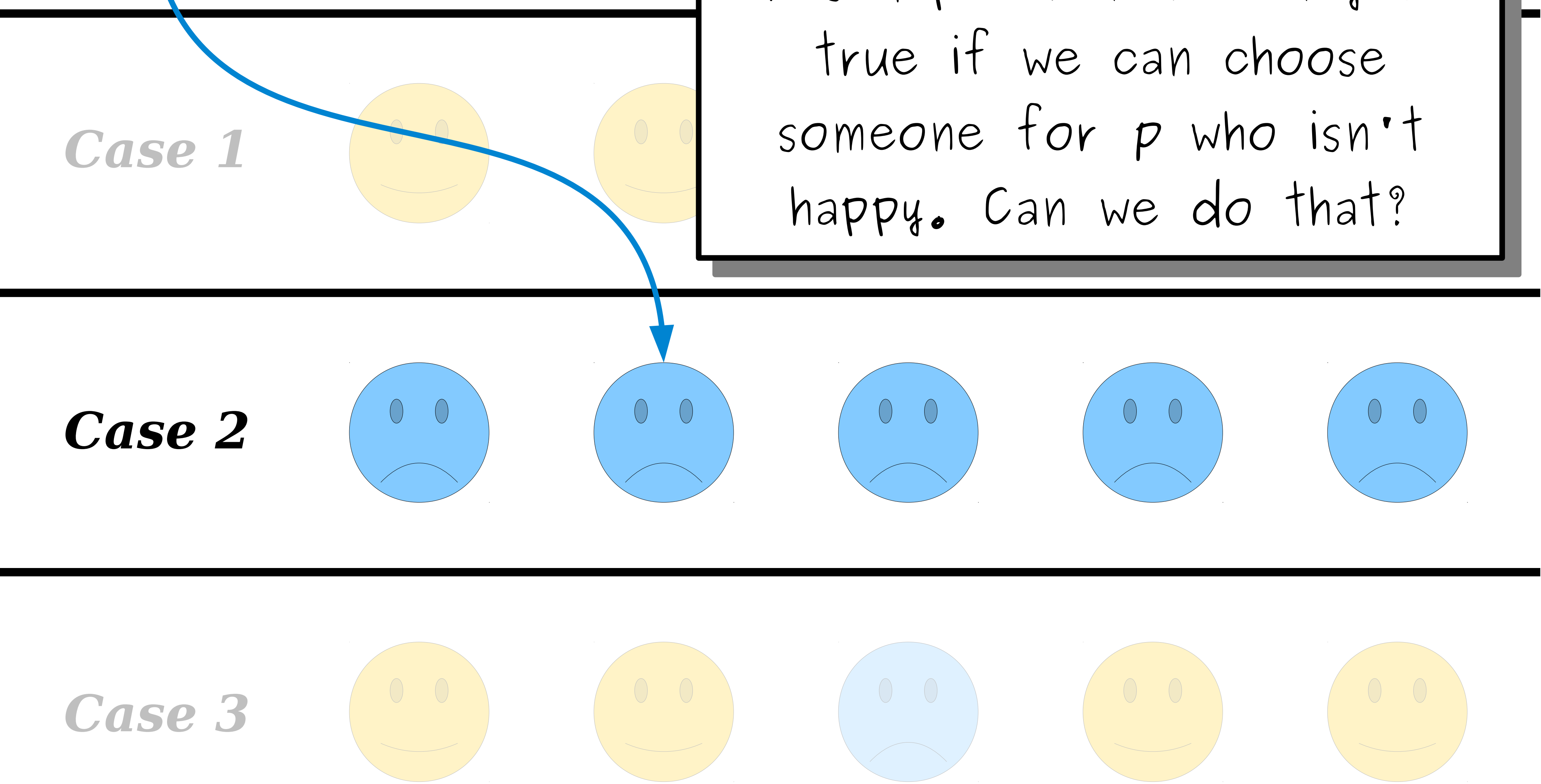
Case 2



Case 3



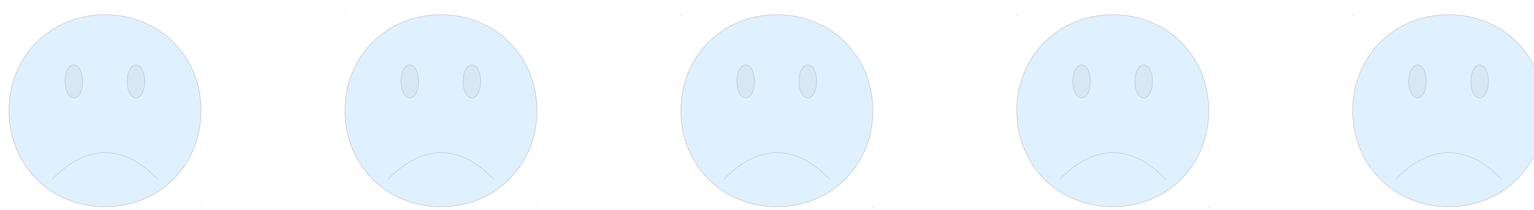
$$\exists p \in P.\ (Happy(p) \to \forall q \in P.\ Happy(q))$$



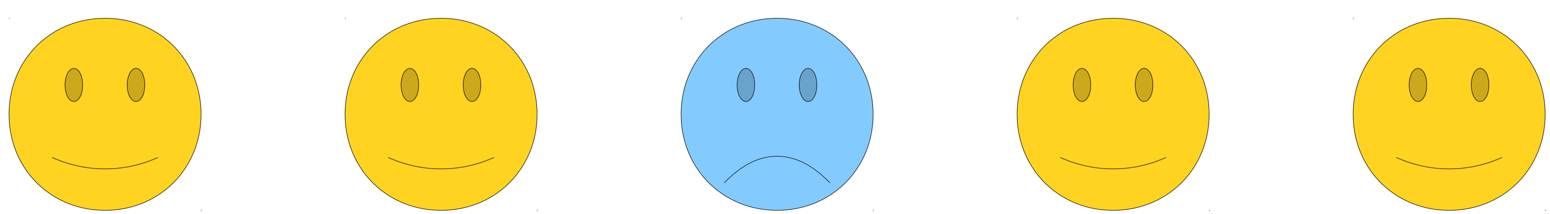
$$\exists p \in P.\ (Happy(p) \rightarrow \forall q \in P.\ Happy(q))$$

Case 1

Case 2



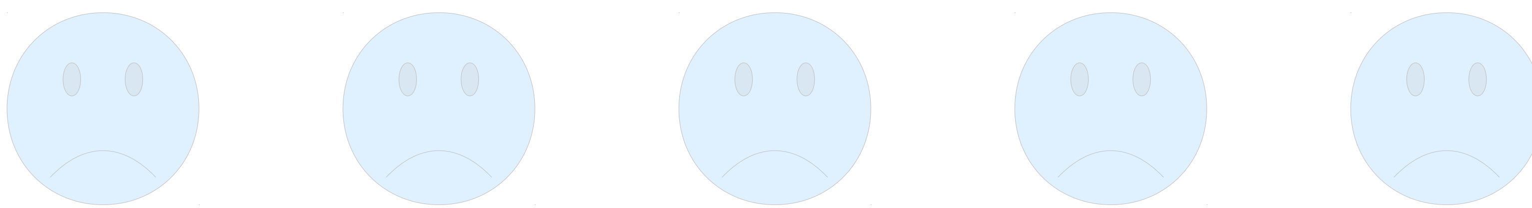
Case 3



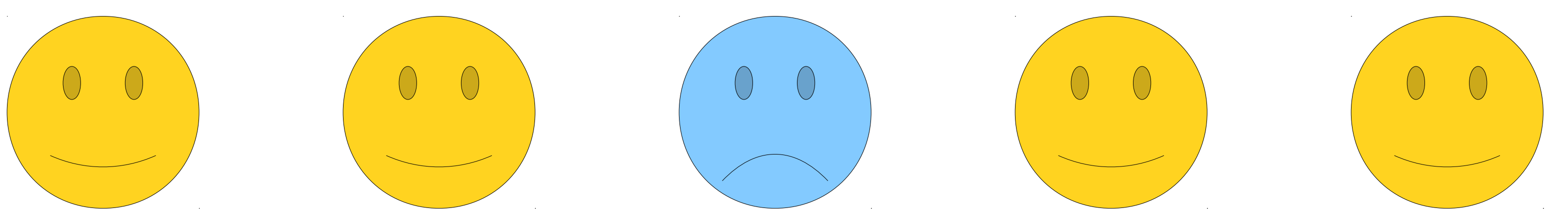
$$\exists p \in P.\ (Happy(p) \to \forall q \in P.\ Happy(q))$$

Case 1

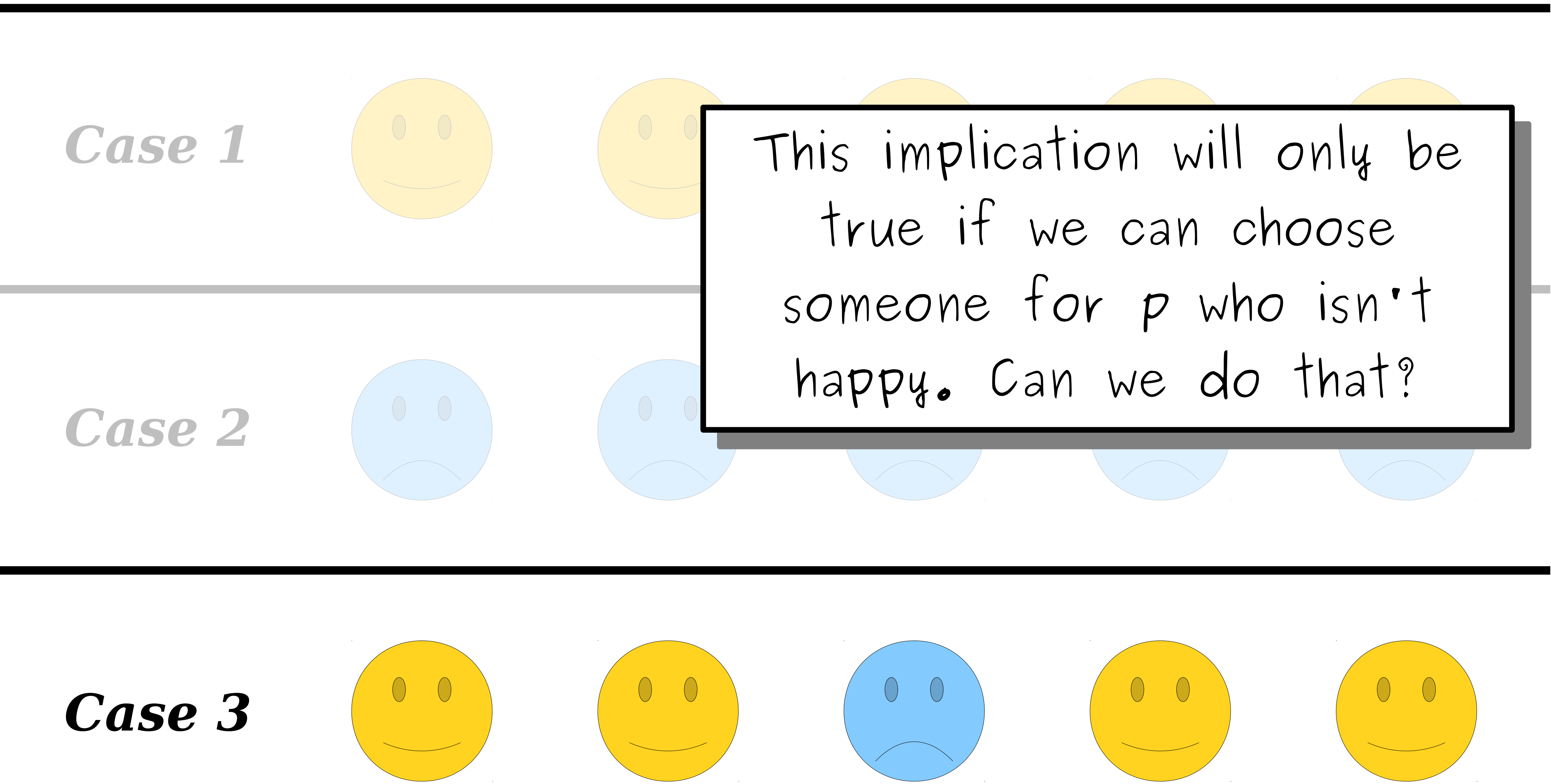
Case 2



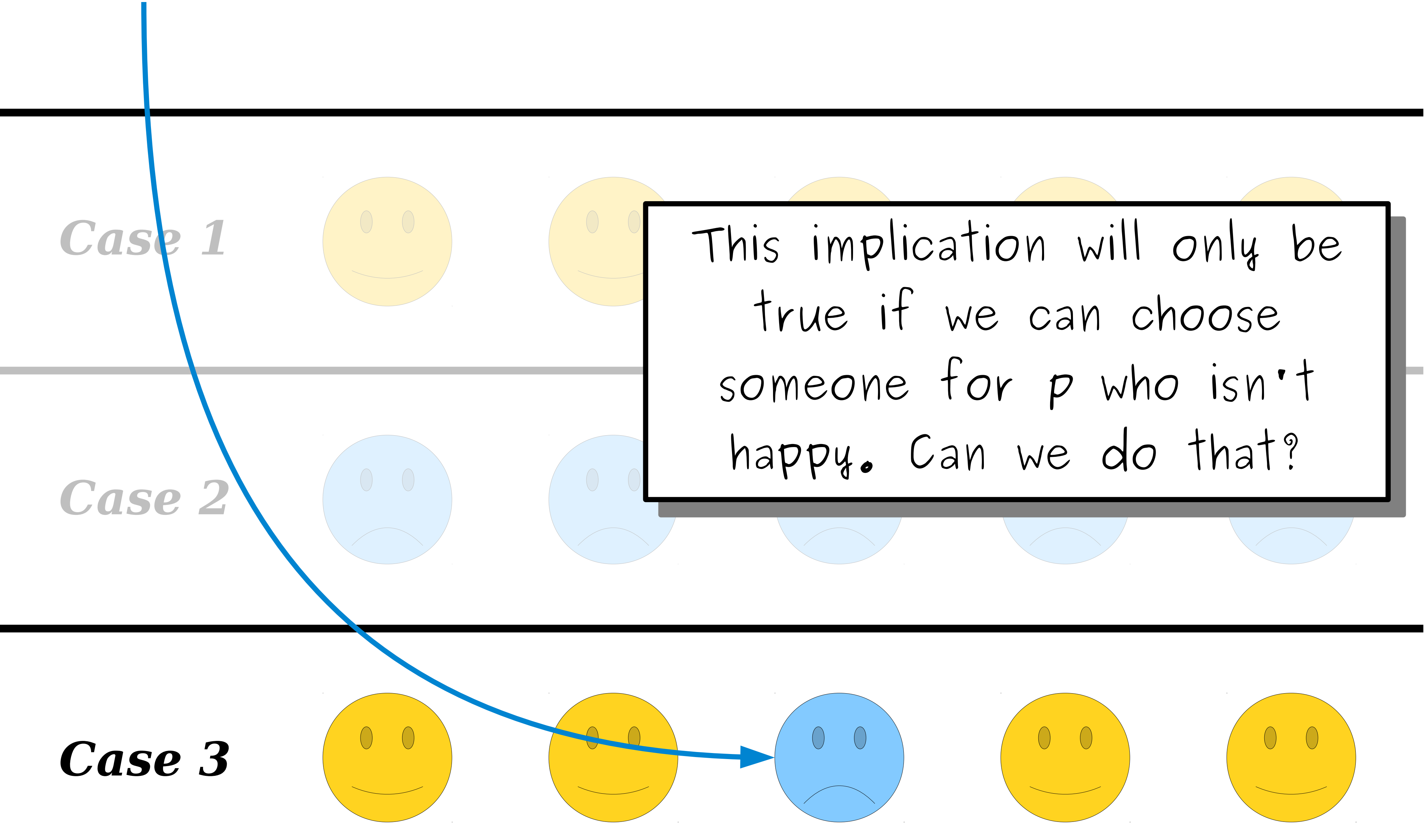
Case 3



$$\exists p \in P.\ (Happy(p) \to \forall q \in P.\ Happy(q))$$



$$\exists p \in P.\ (Happy(p) \to \forall q \in P.\ Happy(q))$$



Remember: implications ***rarely*** go with existential statements. You can make the whole statement true by choosing something where the antecedent is false.

“Someone has two pet kittens
and no other pets.”

“Someone has two pet kittens
and no other pets.”

$$
\begin{array}{l}
\exists p.\ (Person(p)\ \land \\
\quad \exists k_1.\ (Kitten(k_1)\ \land\ HasPet(p, k_1)\ \land \\
\quad\quad \exists k_2.\ (Kitten(k_2)\ \land\ HasPet(p, k_2)) \\
\quad) \\
)
\end{array}
$$

"Someone has two pet kittens
and no other pets."

$$
\begin{aligned}
&\exists p.\ (Person(p)\ \wedge \\
&\quad \exists k_1.\ (Kitten(k_1)\ \wedge\ HasPet(p, k_1)\ \wedge \\
&\qquad \exists k_2.\ (Kitten(k_2)\ \wedge\ HasPet(p, k_2)) \\
&\quad) \\
&)
\end{aligned}
$$

“Someone has two pet kittens
and no other pets.”

$$
\begin{aligned}
&\exists p.\ (Person(p)\ \land \\
&\quad \exists k_1.\ (Kitten(k_1)\ \land\ HasPet(p, k_1)\ \land \\
&\quad\quad \exists k_2.\ (Kitten(k_2)\ \land\ k_1 \neq k_2\ \land\ HasPet(p, k_2)) \\
&\quad) \\
&)
\end{aligned}
$$

"Someone has two pet kittens
and no other pets."

$$\exists p.\ (Person(p)\ \land$$
$$\quad \exists k_1.\ (Kitten(k_1)\ \land\ HasPet(p, k_1)\ \land$$
$$\qquad \exists k_2.\ (Kitten(k_2)\ \land\ k_1 \neq k_2\ \land\ HasPet(p, k_2))$$
$$\quad)$$
$$)$$

“Someone has two pet kittens
and no other pets.”

$$
\begin{array}{l}
\exists p.\ (\mathit{Person}(p) \land \\
\quad \exists k_1.\ (\mathit{Kitten}(k_1) \land \mathit{HasPet}(p, k_1) \land \\
\quad\quad \exists k_2.\ (\mathit{Kitten}(k_2) \land k_1 \neq k_2 \land \mathit{HasPet}(p, k_2) \land \\
\quad\quad\quad \forall q.\ (\mathit{HasPet}(p, q) \to q = k_1 \lor q = k_2) \\
\quad\quad) \\
\quad) \\
)
\end{array}
$$

“Someone has two pet kittens and no other pets.”

$$
\begin{aligned}
&\exists p.\ (Person(p)\ \land \\
&\quad \exists k_1.\ (Kitten(k_1)\ \land\ HasPet(p, k_1)\ \land \\
&\quad\quad \exists k_2.\ (Kitten(k_2)\ \land\ k_1 \neq k_2\ \land\ HasPet(p, k_2)\ \land \\
&\quad\quad\quad \forall q.\ (HasPet(p, q) \to q = k_1 \lor q = k_2) \\
&\quad\quad) \\
&\quad) \\
&)
\end{aligned}
$$

1. Remember that multiple quantifiers can range over the same objects!

2. To express “and nothing else does,” show that anything matching the property must be equal to something you already know.

Your Questions

“Do teachers/you ever notice people sleeping in your class? What goes through your mind?”

Oh yeah. And when people are on their laptops not paying attention. ☺

Usually I feel bad for people who are falling asleep. I assume they're probably working too hard and could use some rest.

I can't speak for other professors, though.

“What is the most reassuring thing you can say to a struggling CS103 student? Asking for a friend ;)”

The first three problem sets focus on three major skills:

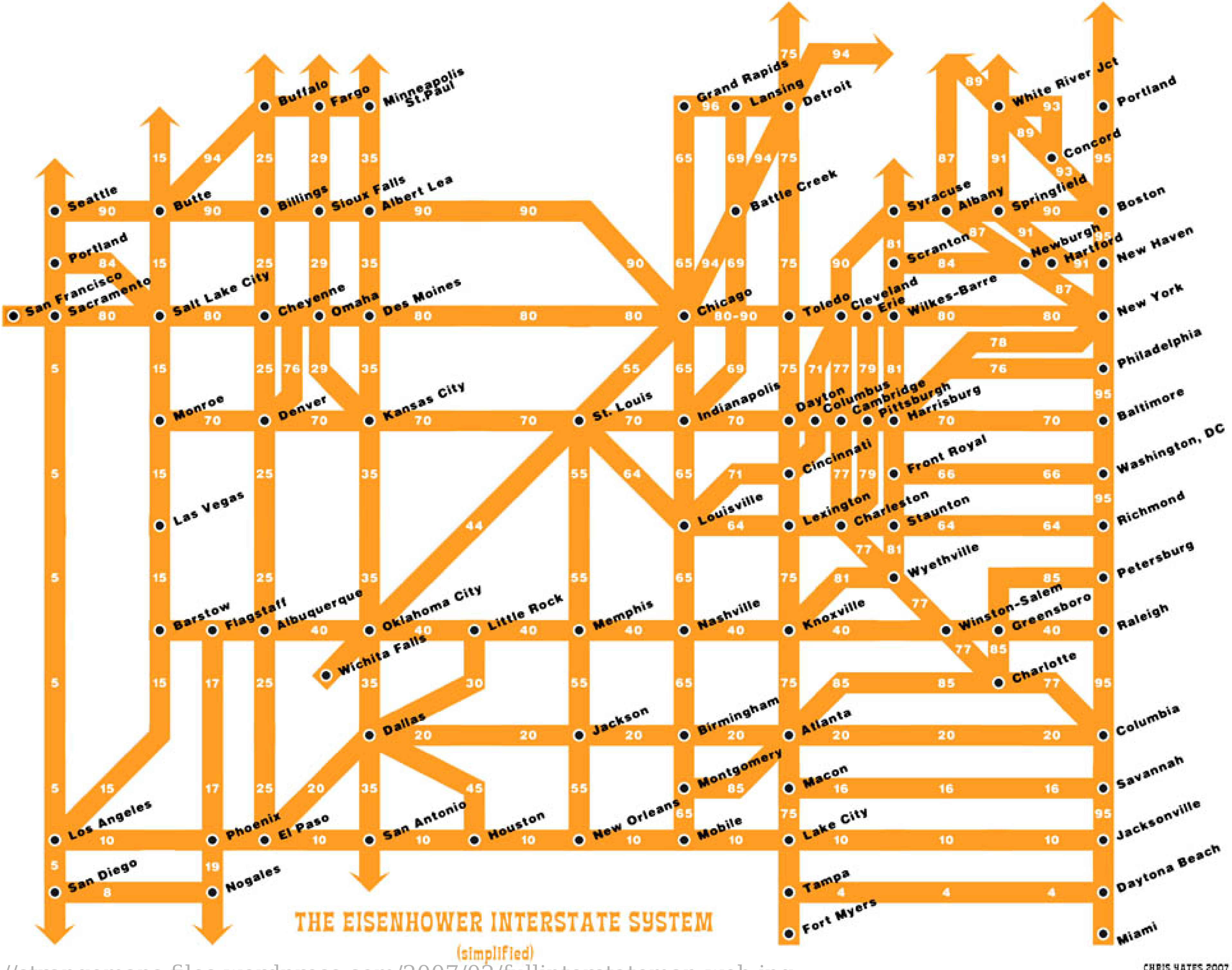
PS1: General proof mechanics.
PS2: Understanding and manipulating first-order logic.
PS3: Writing proofs that call back to first-order definitions.

These skills are tough! Most schools spend an entire quarter just on these three areas. The big step in the first ¾ of this class is getting comfortable with these areas.

It's okay to struggle while you're picking up these skills. First, don't panic. Your goal is to pick up these skills and you still have plenty of time to do so. Second, look over the above list and figure out where you're having trouble. Third, seek out our advice on how to specifically improve in those areas. Finally, remember that <u>you will get this</u> and that <u>you can do this</u>. Hang in there, and let us know how we can help!

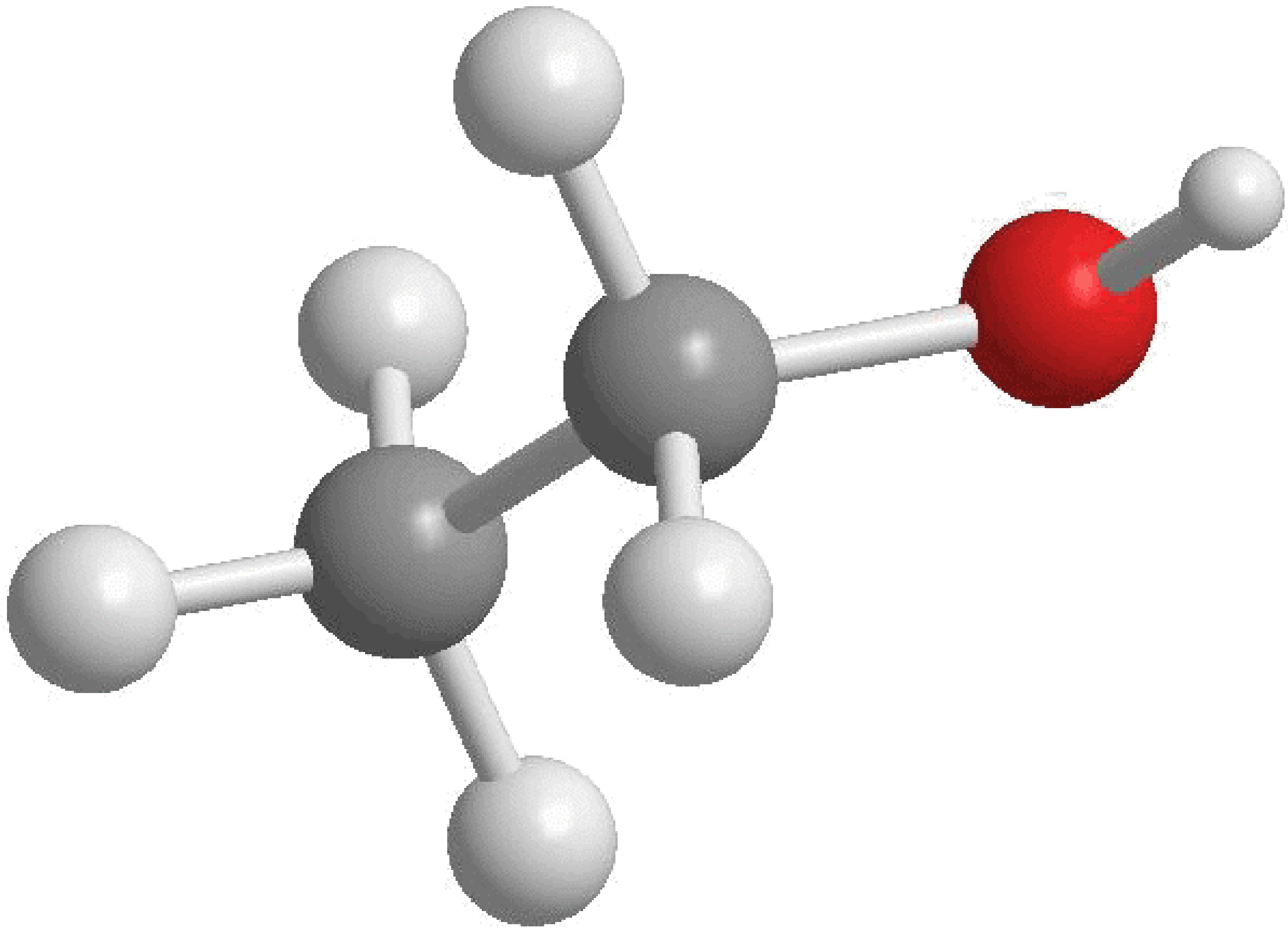
Back to CS103!

Graphs

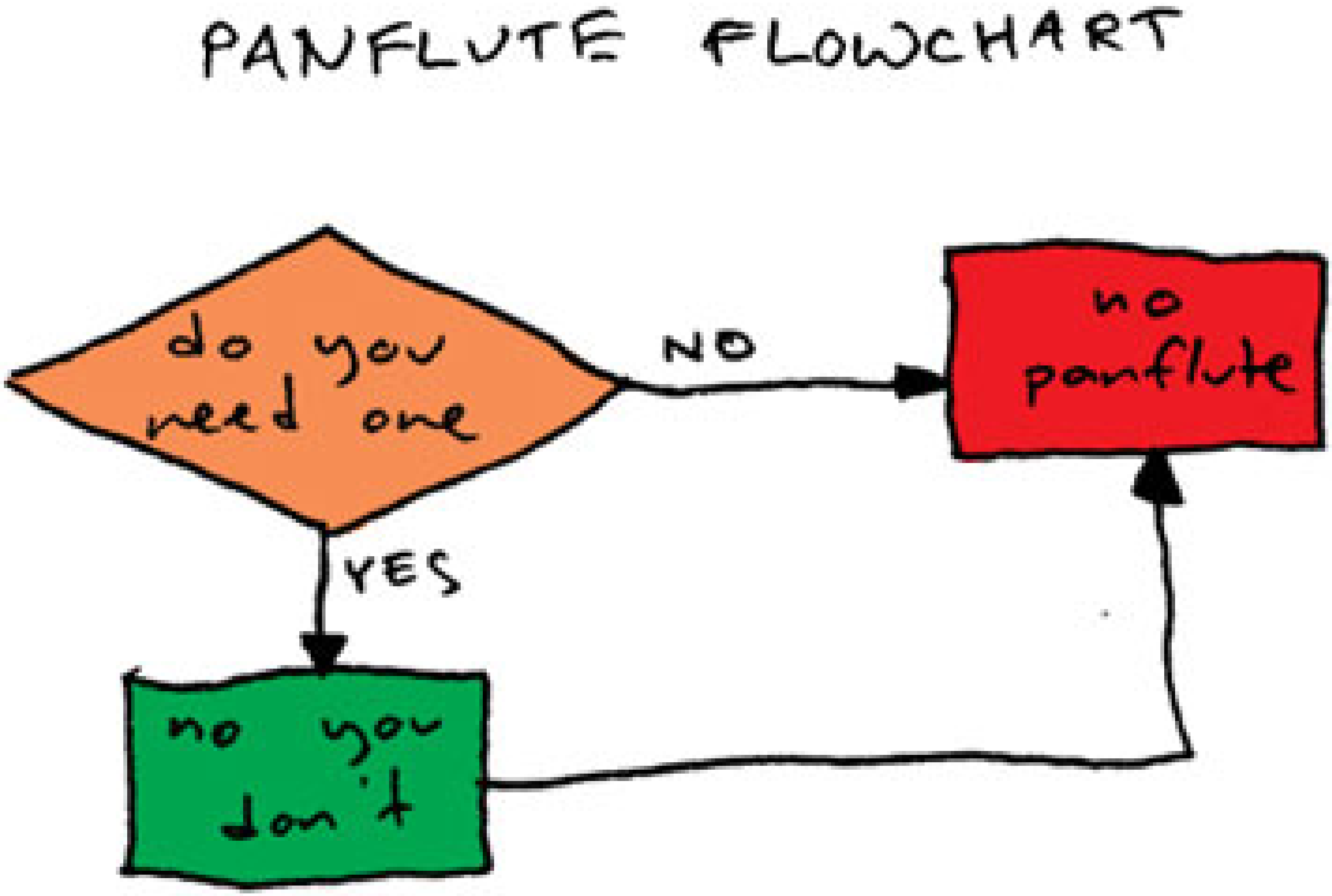


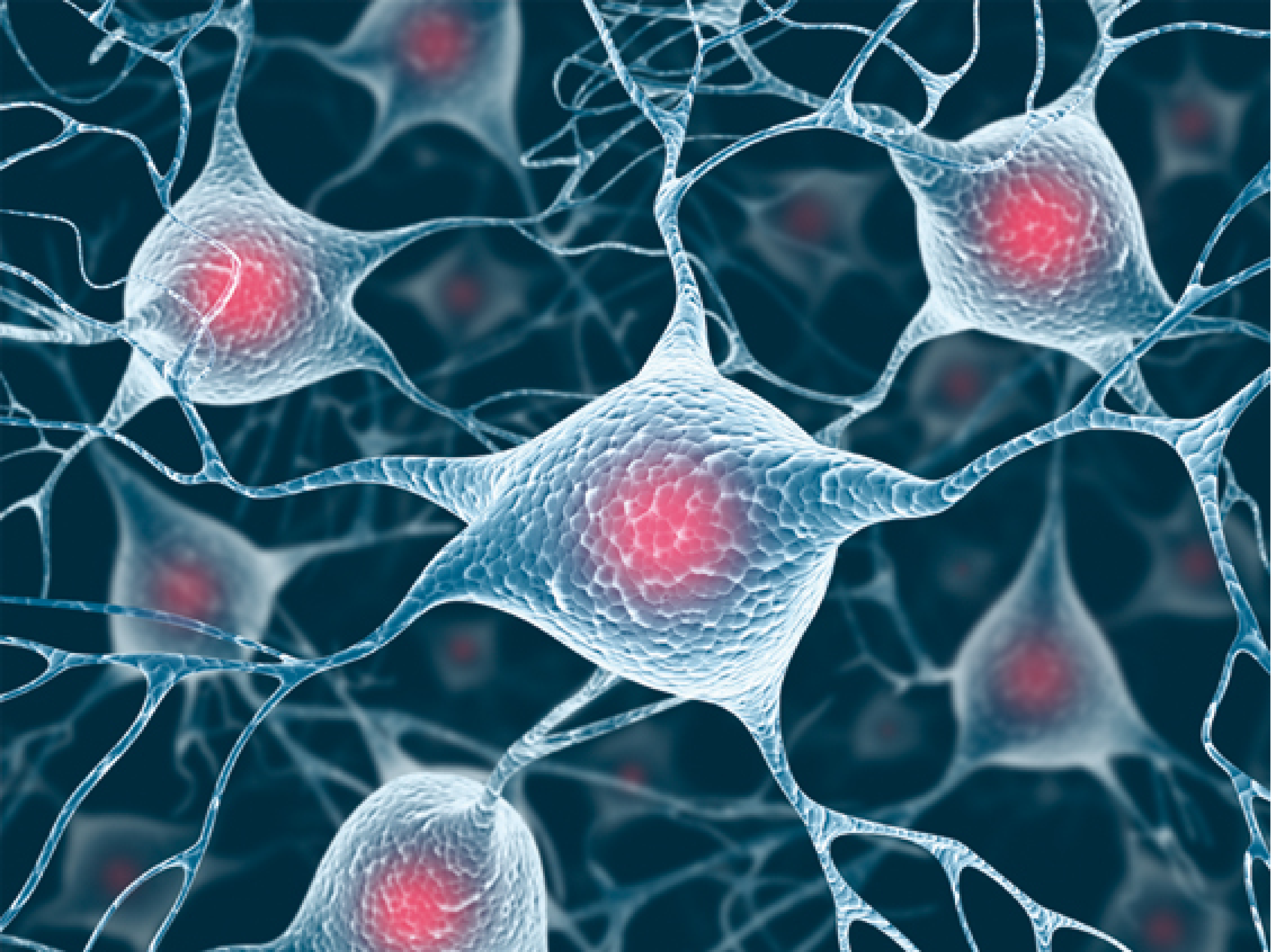
http://strangemaps.files.wordpress.com/2007/02/fullinterstatemap-web.jpg

Chemical Bonds



http://4.bp.blogspot.com/-xCtBJ8lKHqA/Tjm0BONWBRI/AAAAAAAAAK4/-


PANFLUTE FLOWCHART
do you need one
NO
no panflute
YES
no you don't



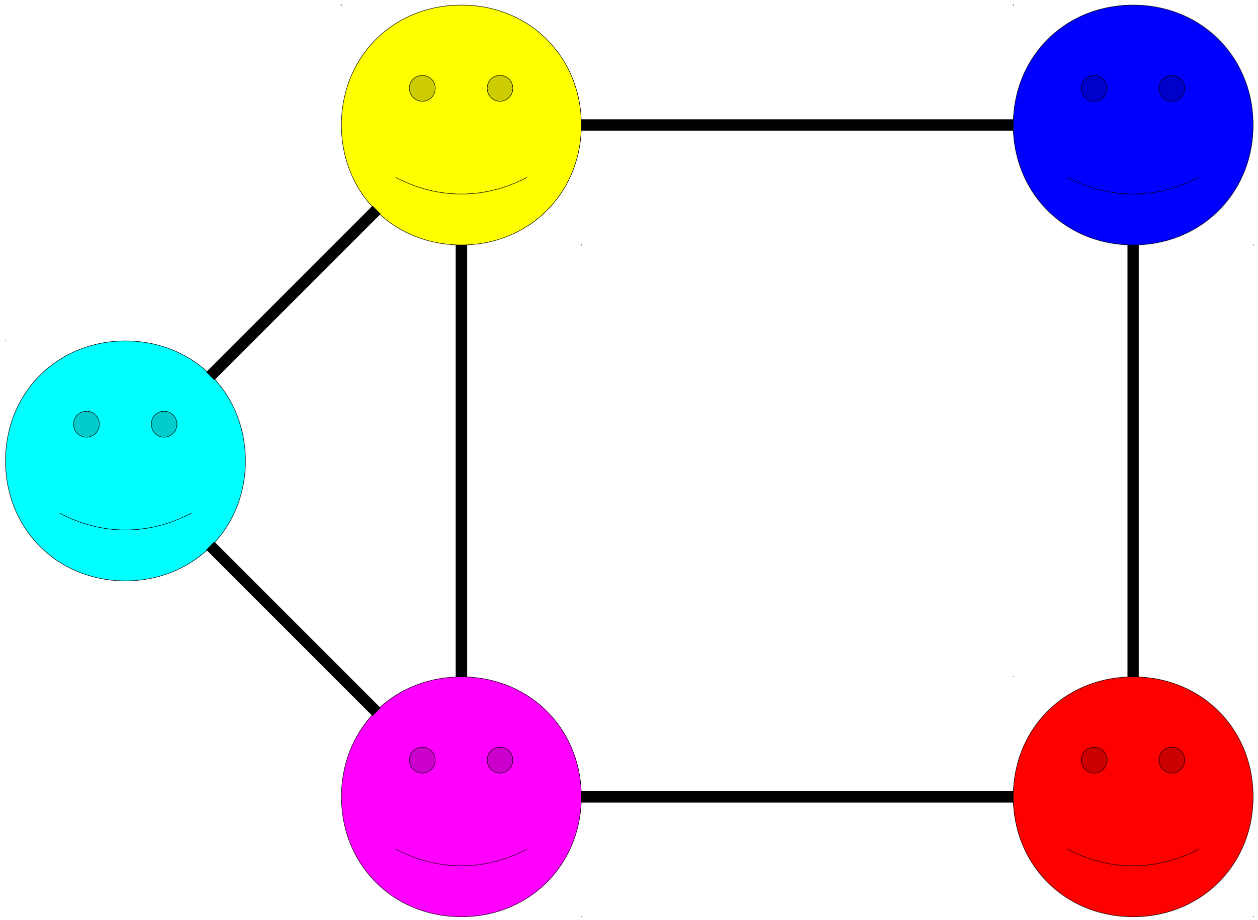
facebook®

Me too!

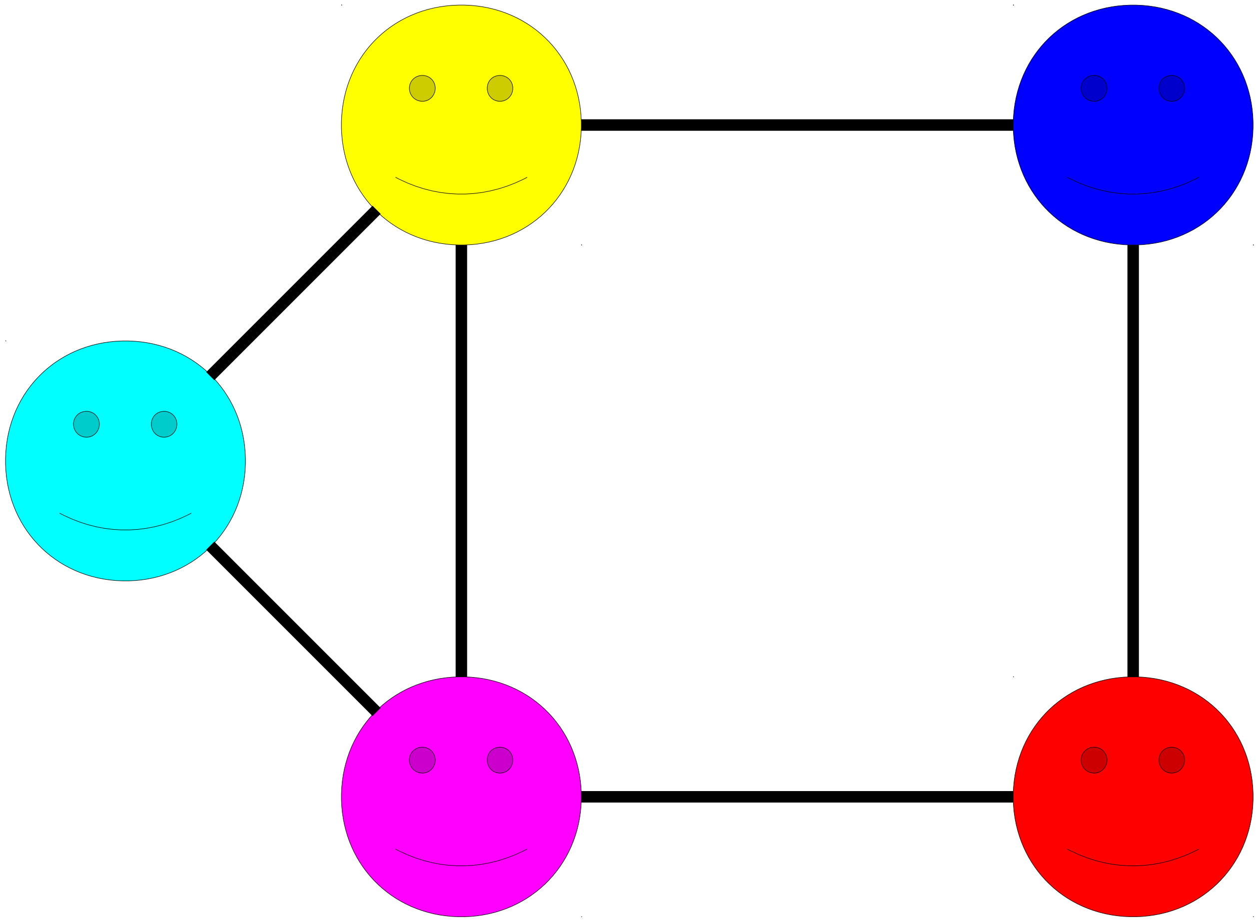
What's in Common

- Each of these structures consists of
  - a collection of objects and
  - links between those objects.
- ***Goal:*** find a general framework for describing these objects and their properties.

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

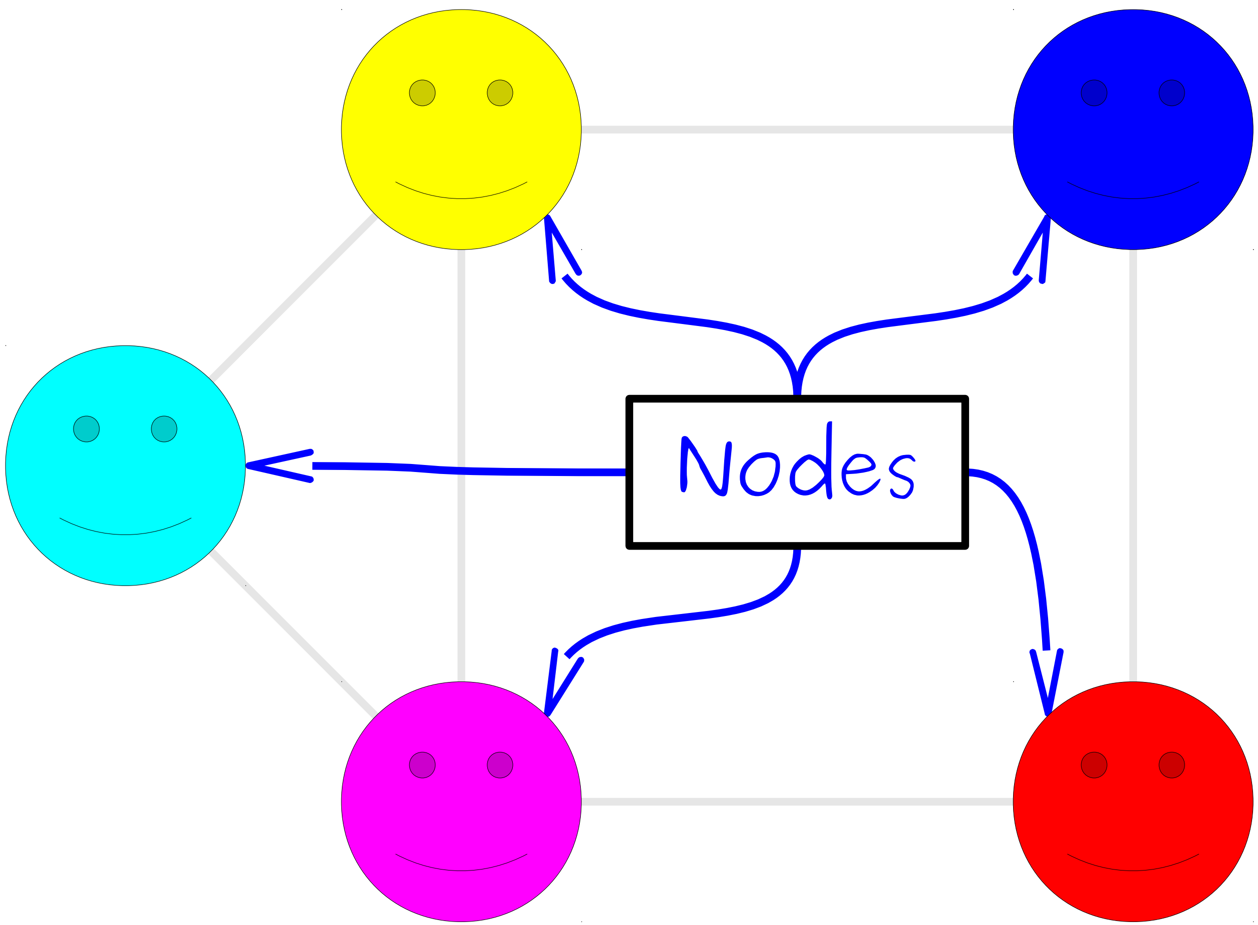


A ***graph*** is a mathematical structure for representing relationships.



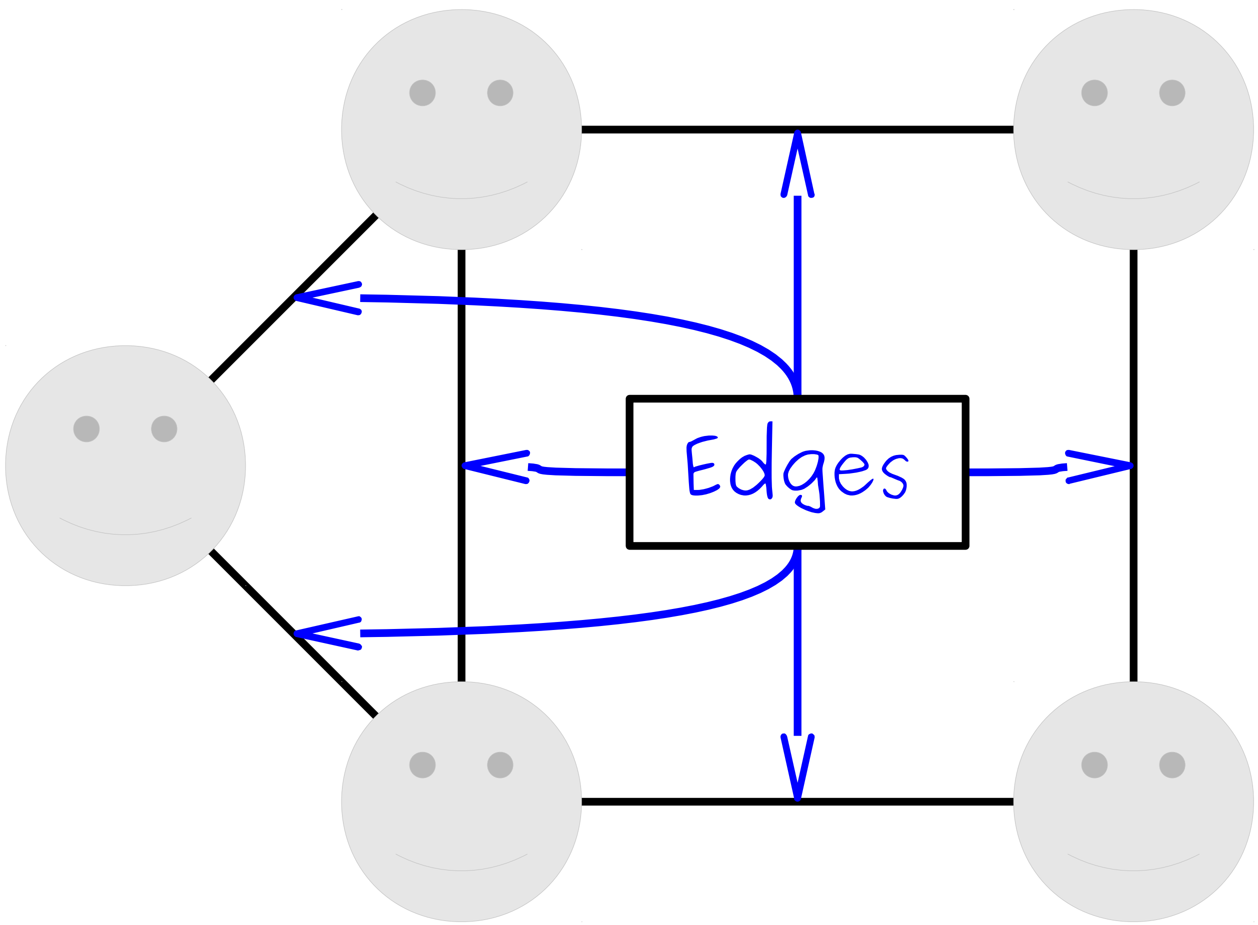
A graph consists of a set of ***nodes*** (or ***vertices***) connected by ***edges*** (or ***arcs***)

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



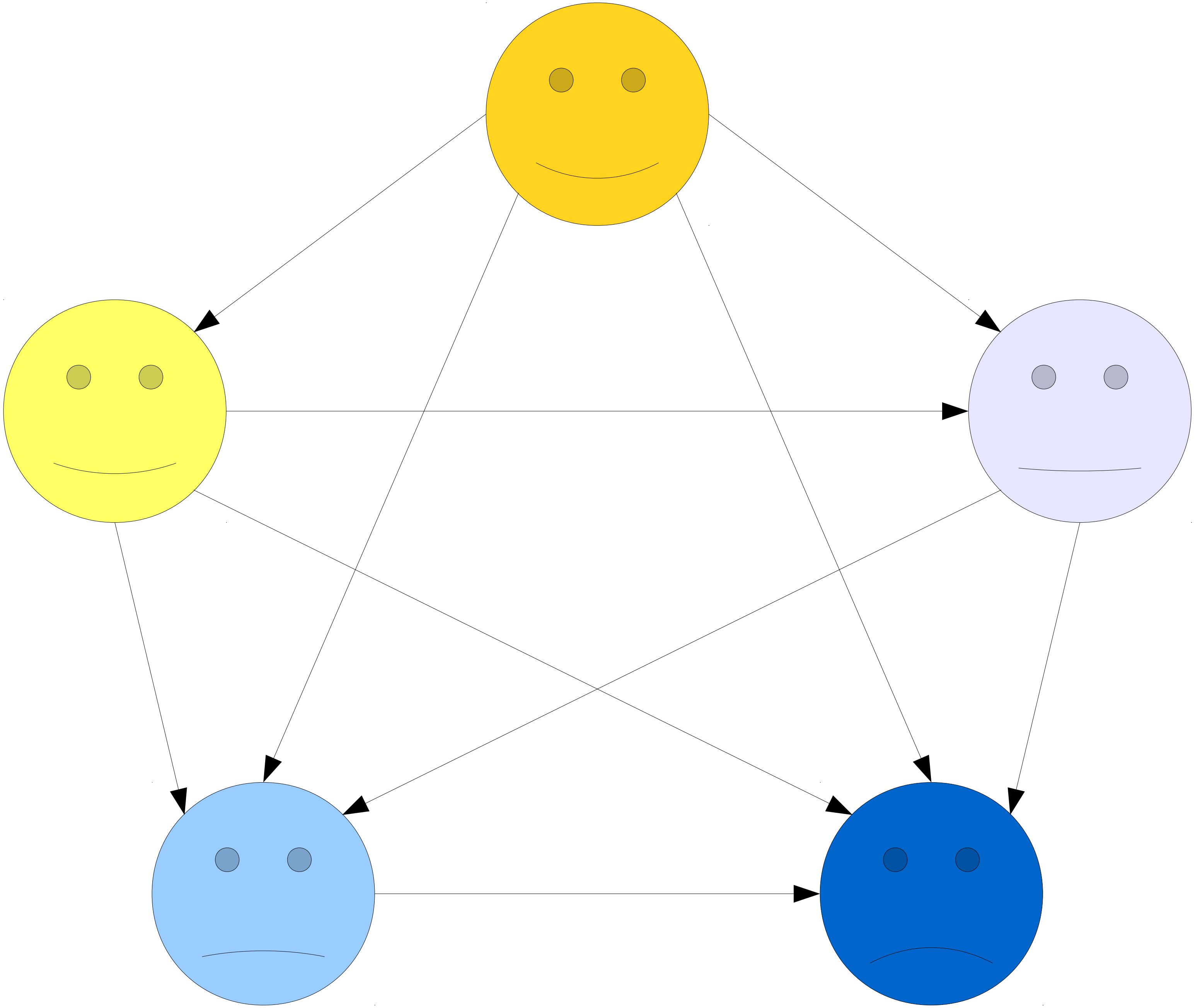
A graph consists of a set of ***nodes*** (or ***vertices***) connected by ***edges*** (or ***arcs***)

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

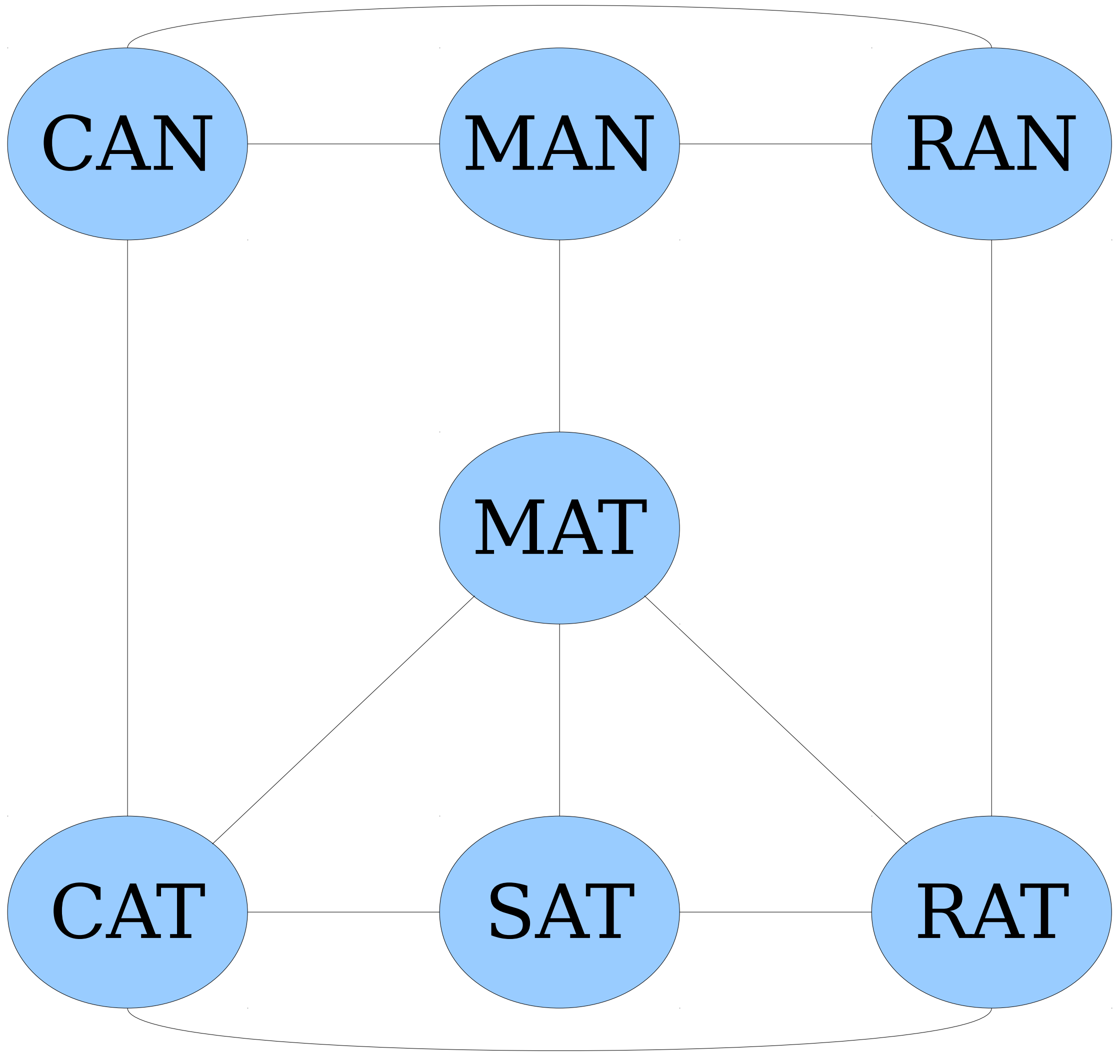


A graph consists of a set of ***nodes*** (or ***vertices***) connected by ***edges*** (or ***arcs***)

Some graphs are ***directed***.



Some graphs are ***undirected***.



Going forward, we're primarily going to focus on undirected graphs.

The term “graph” generally refers to undirected graphs unless specified otherwise.

Formalizing Graphs

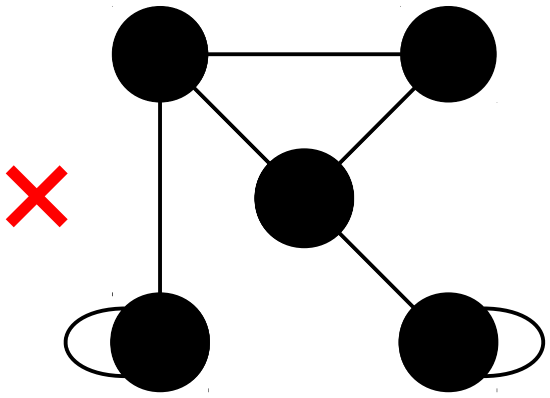
- How might we define a graph mathematically?
- We need to specify
  - what the nodes in the graph are, and
  - which edges are in the graph.
- The nodes can be pretty much anything.
- What about the edges?

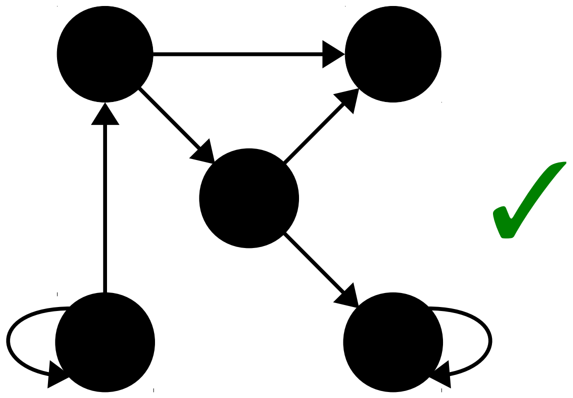
Formalizing Graphs

- An ***unordered pair*** is a set $\{a, b\}$ of two elements (remember that sets are unordered).
  - $\{0, 1\} = \{1, 0\}$
- An ***undirected graph*** is an ordered pair $G = (V, E)$, where
  - V is a set of nodes, which can be anything, and
  - E is a set of edges, which are unordered pairs of nodes drawn from V.
- A ***directed graph*** is an ordered pair $G = (V, E)$, where
  - V is a set of nodes, which can be anything, and
  - E is a set of edges, which are *ordered* pairs of nodes drawn from V.

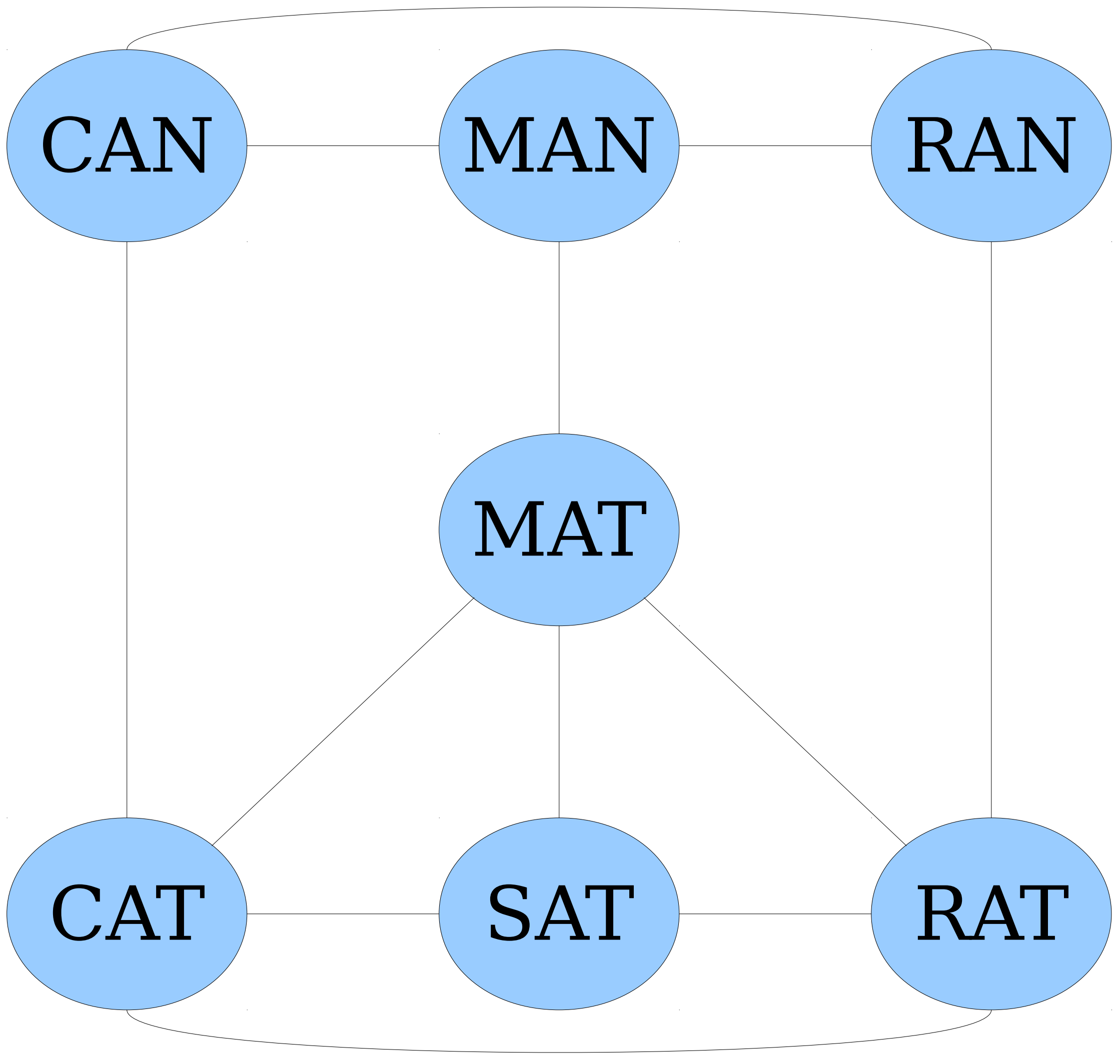
Self-Loops

- An edge from a node to itself is called a ***self-loop***.
- In undirected graphs, self-loops are generally not allowed unless specified otherwise.
    - This is mostly to keep the math easier. If you allow self-loops, a lot of results get messier and harder to state.
- In directed graphs, self-loops are generally allowed unless specified otherwise.

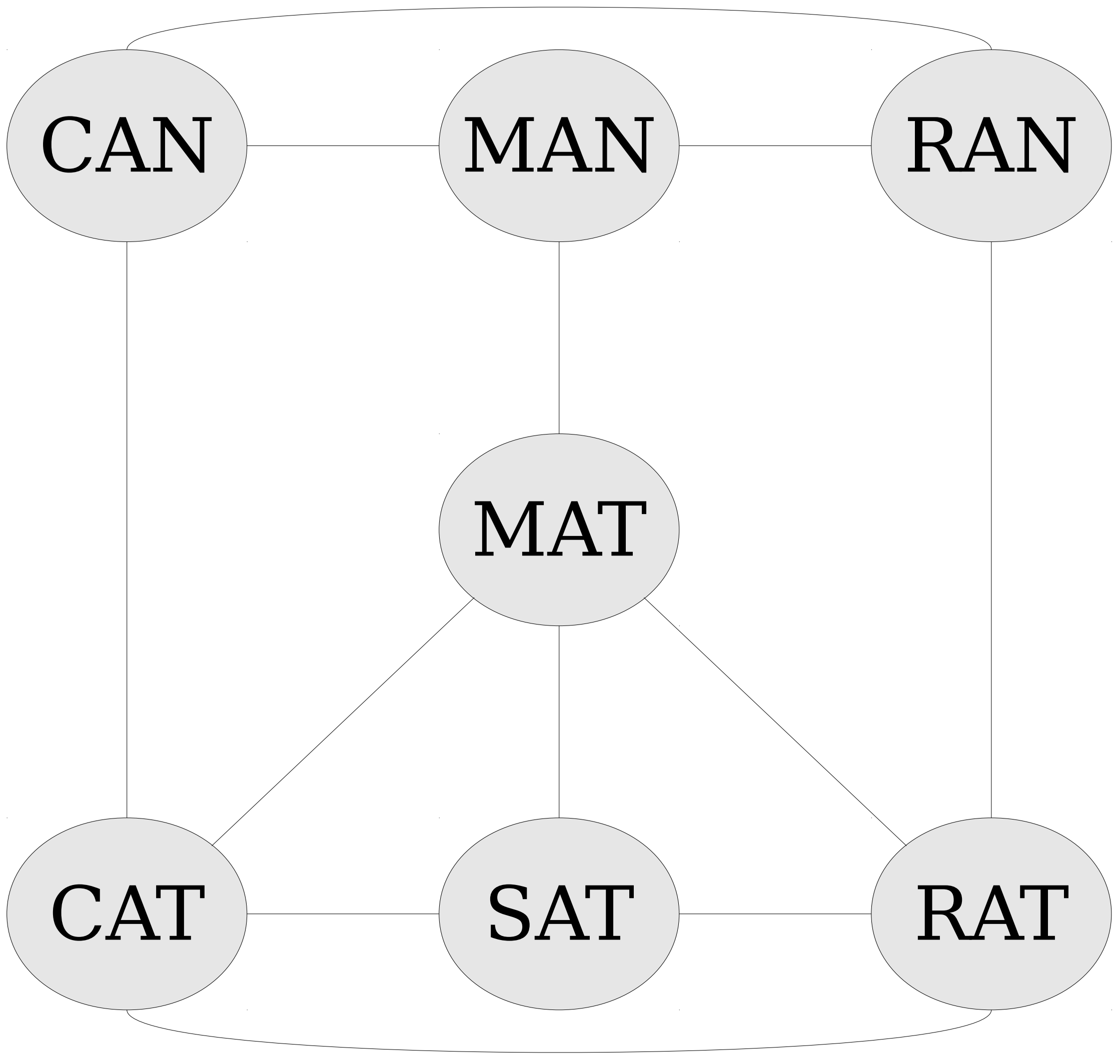




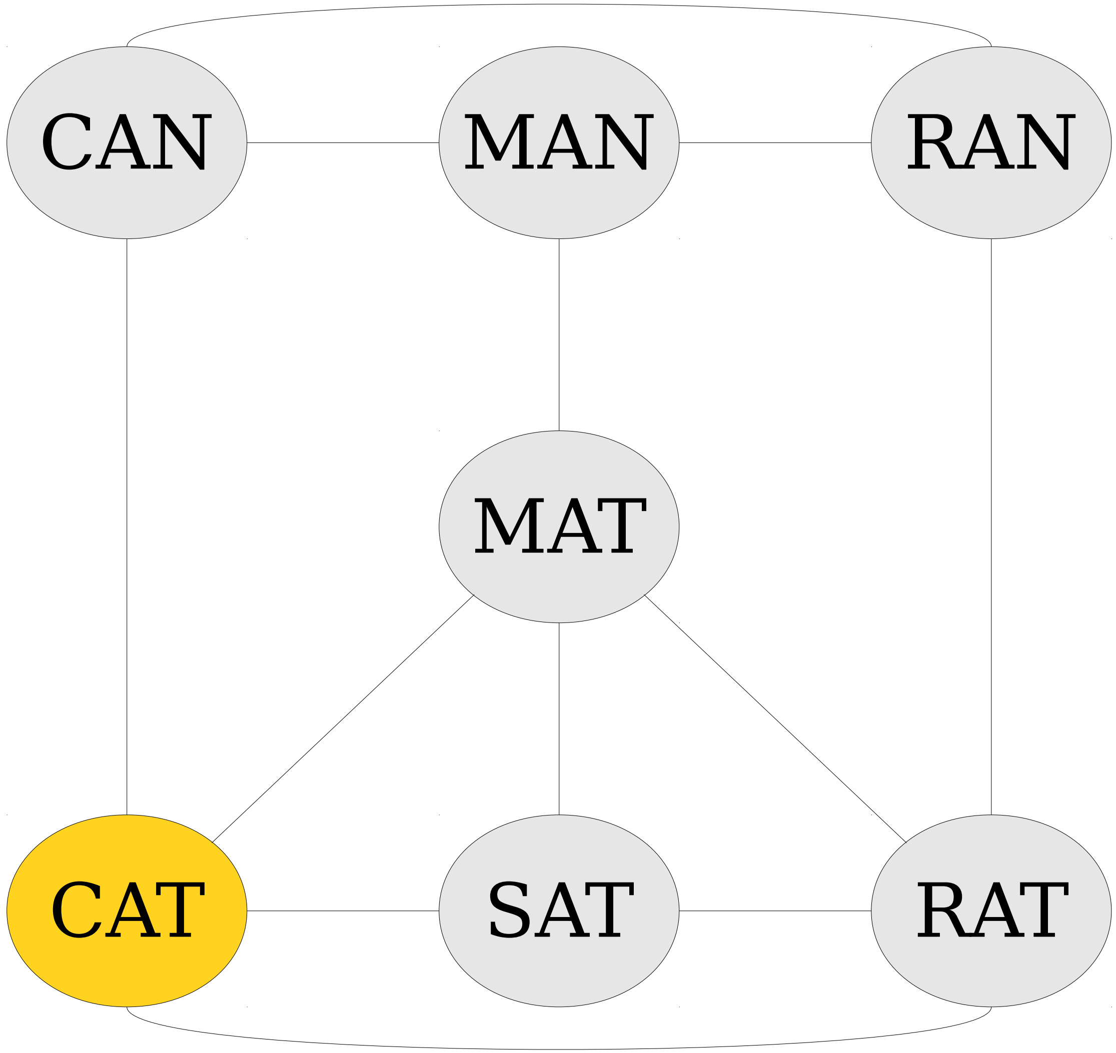
Standard Graph Terminology



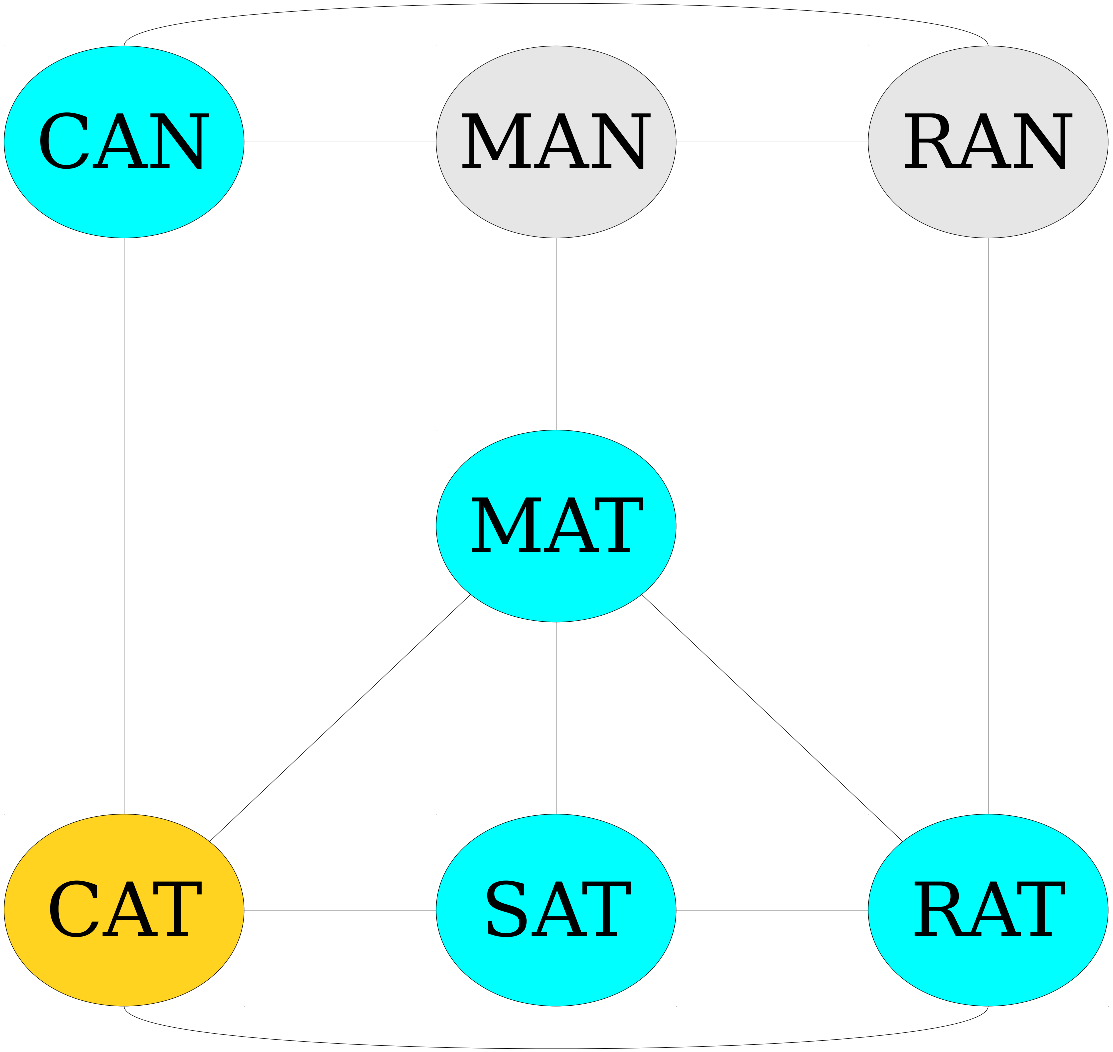
Two nodes are called ***adjacent*** if there is an edge between them.



Two nodes are called ***adjacent*** if there is an edge between them.



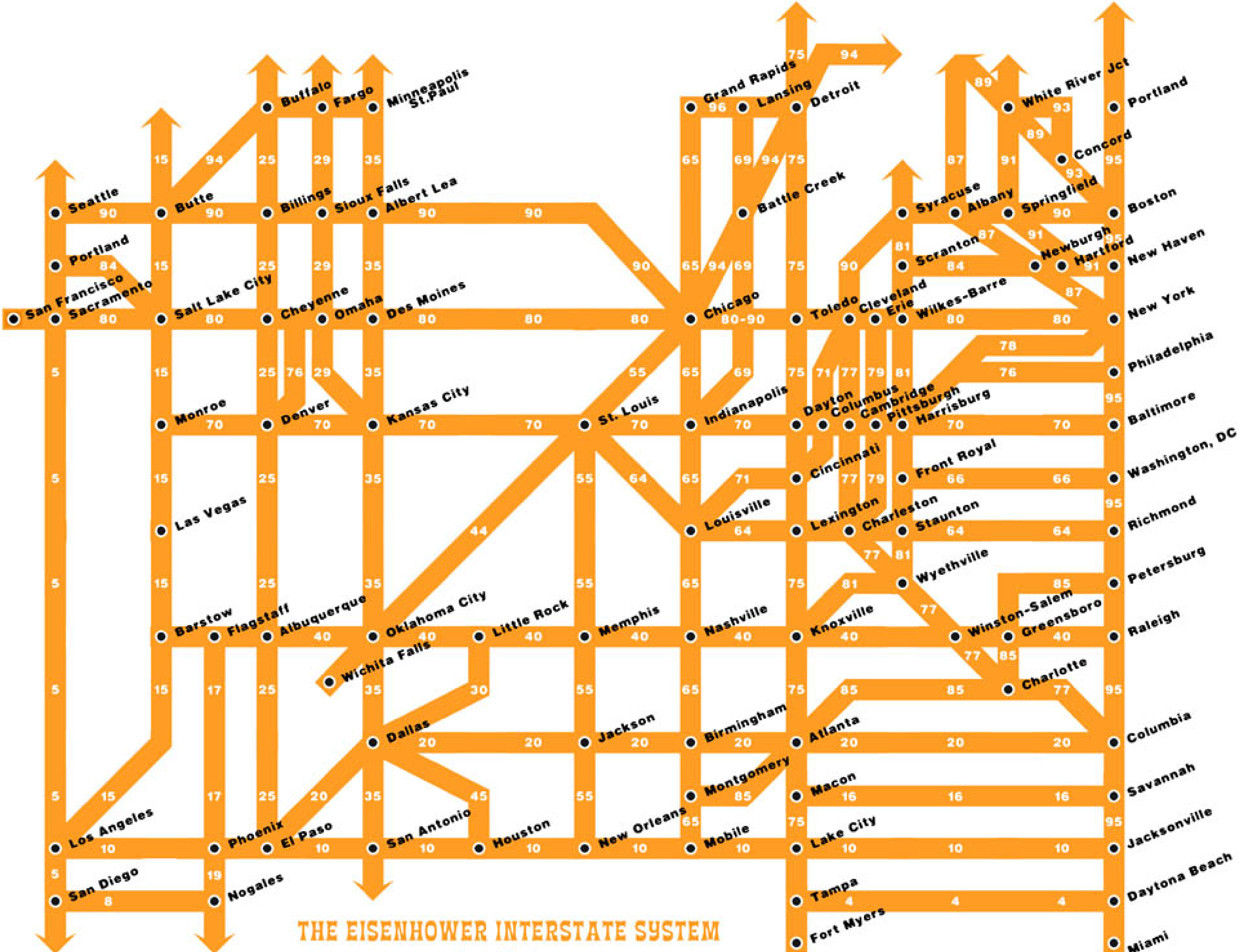
Two nodes are called ***adjacent*** if there is an edge between them.



Two nodes are called ***adjacent*** if there is an edge between them.

Using our Formalisms

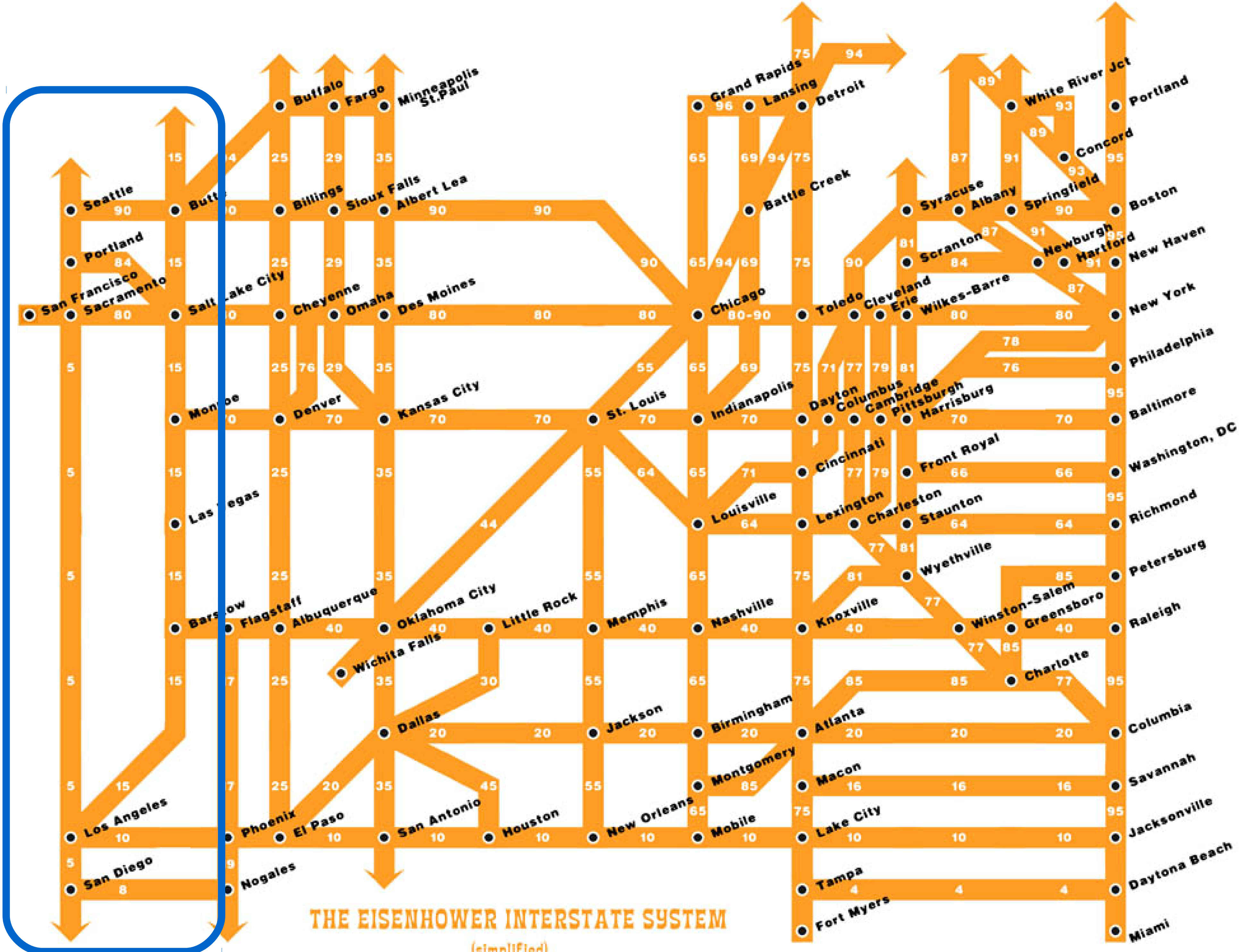
- Let $G = (V, E)$ be a graph.
- Intuitively, two nodes are adjacent if they're linked by an edge.
- Formally speaking, we say that two nodes $u, v \in V$ are ***adjacent*** if $\{u, v\} \in E$.



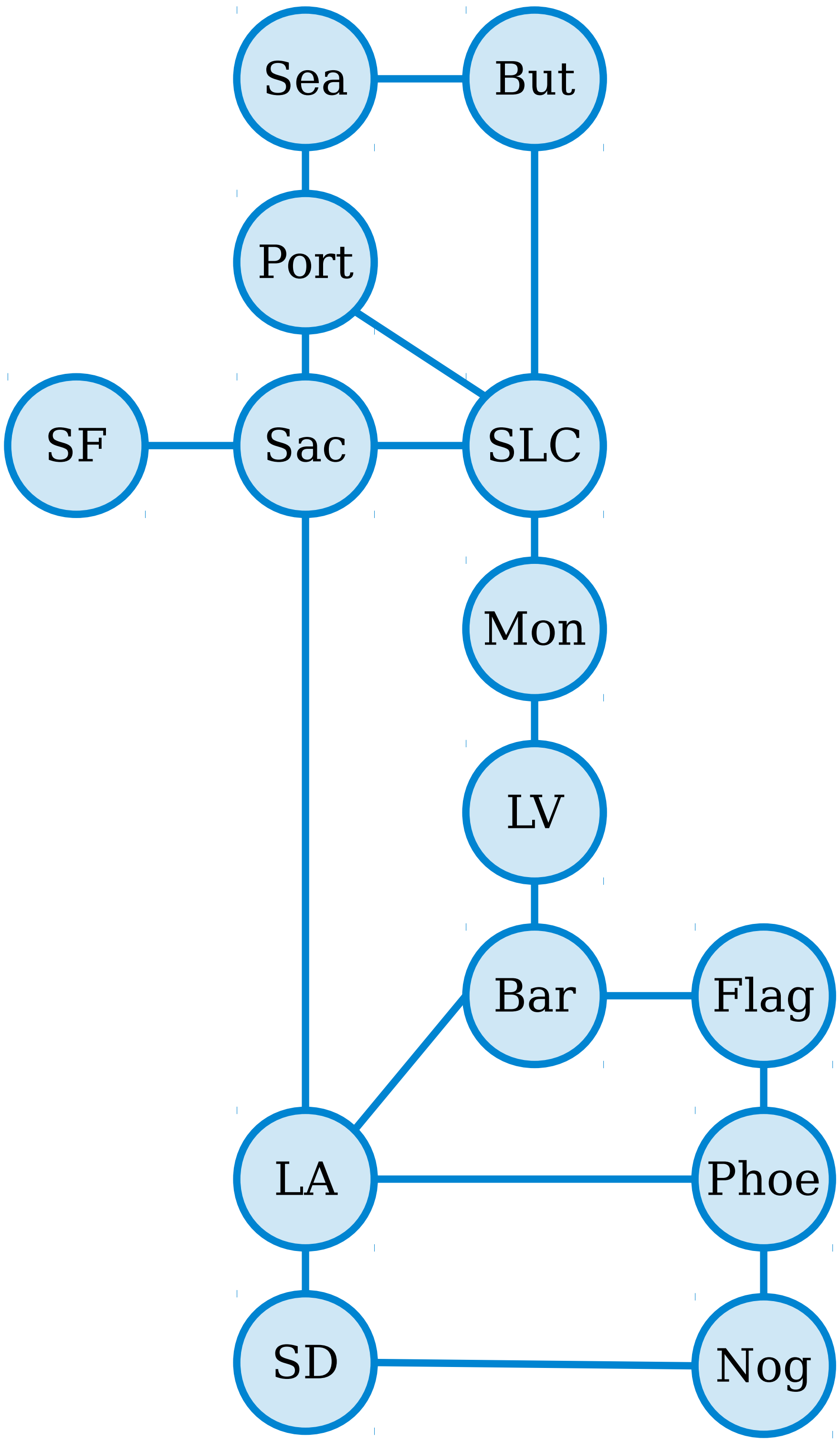
THE EISENHOWER INTERSTATE SYSTEM

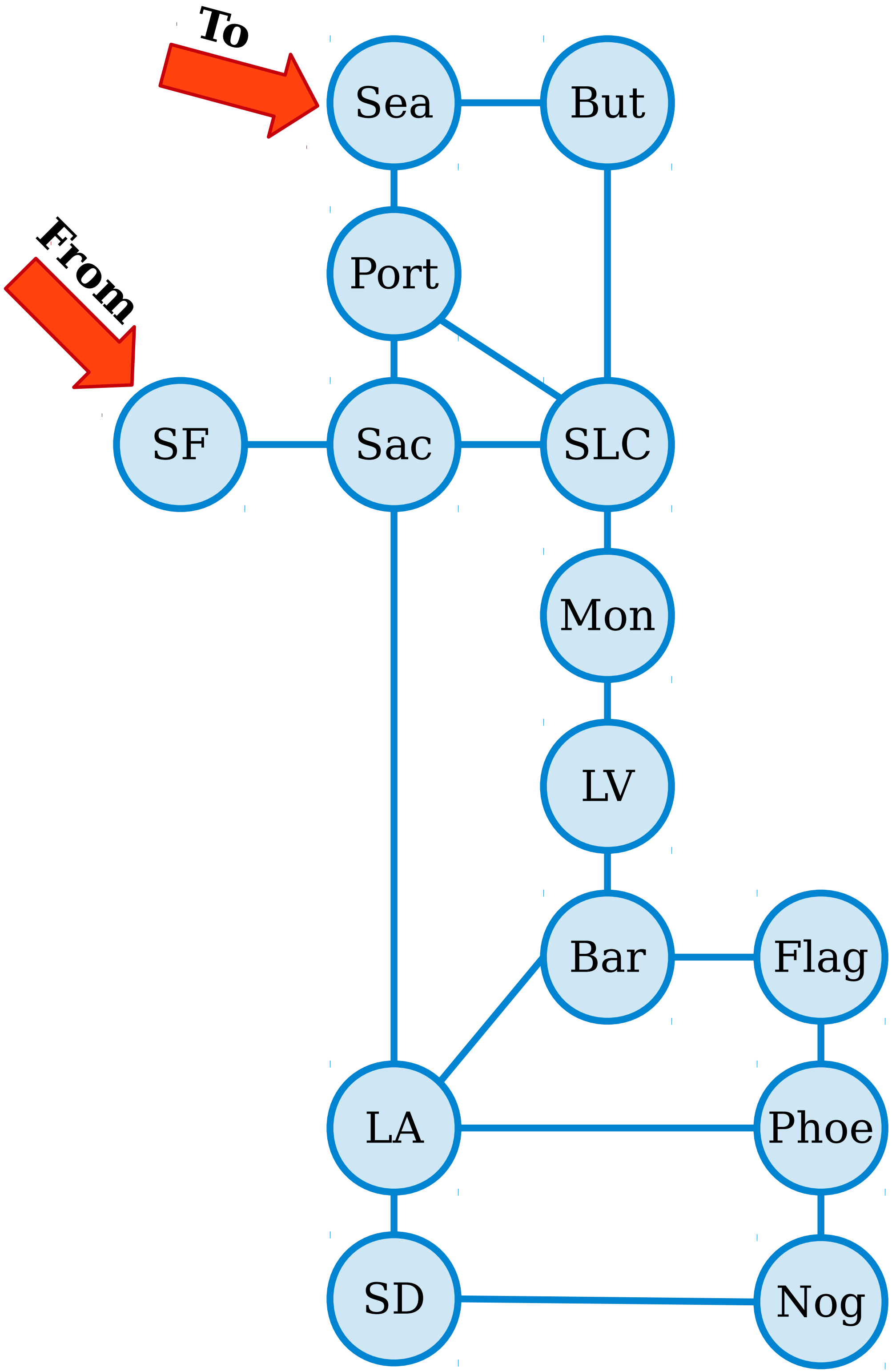
(simplified)

CHRIS YATES 2007

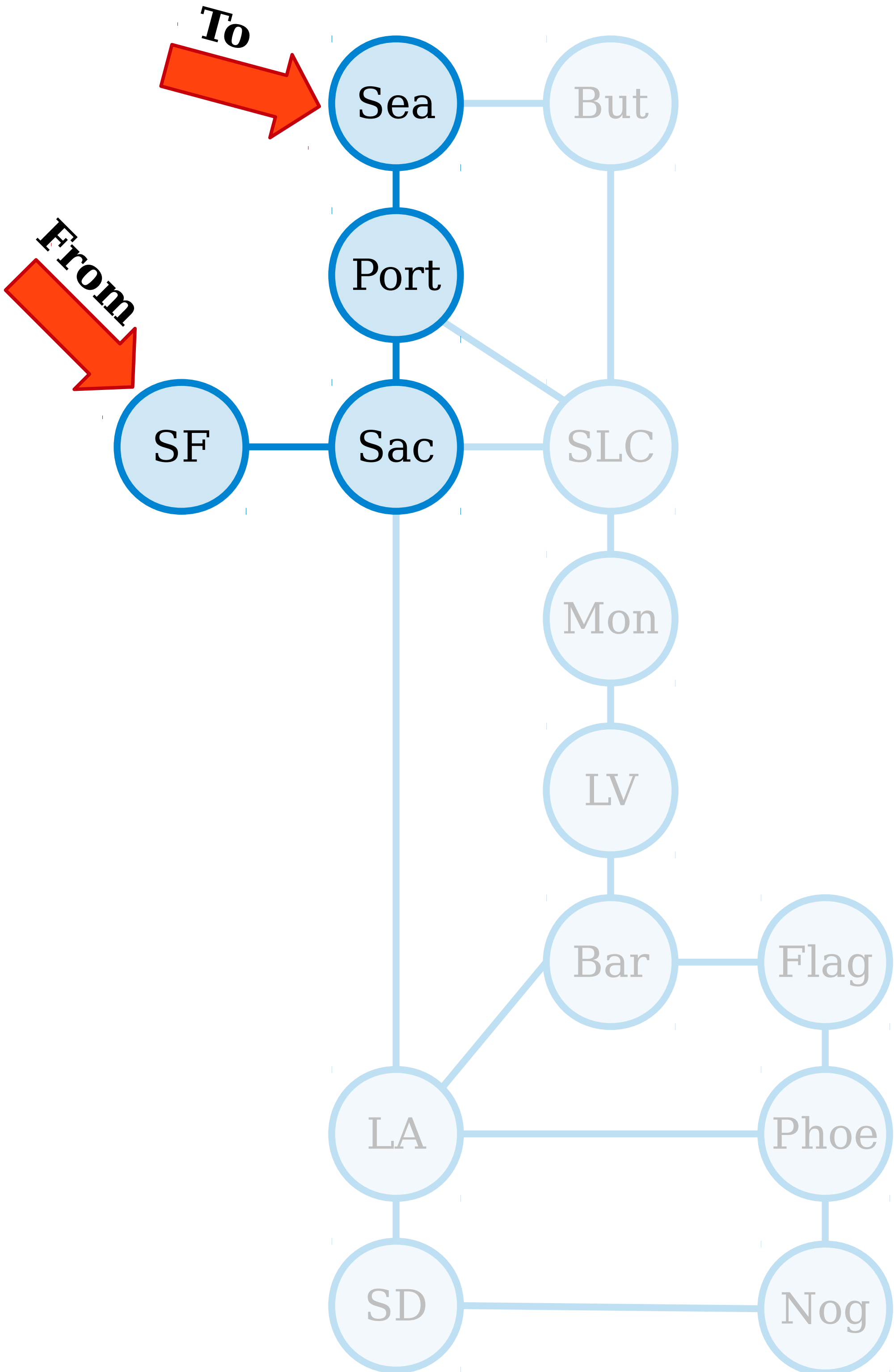


http://strangemaps.files.wordpress.com/2007/02/fullinterstatemap-web.jpg

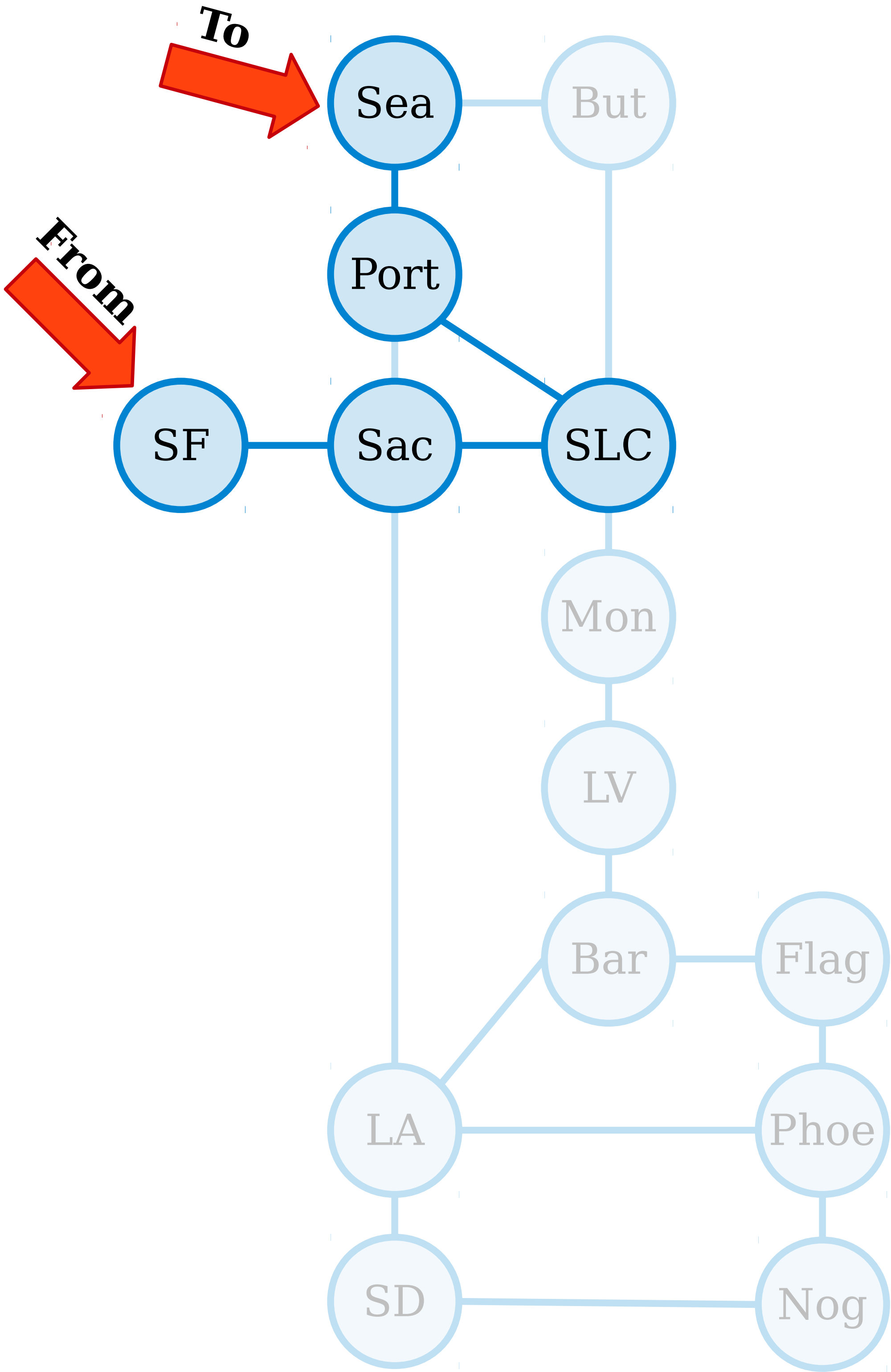

Sea
But
Port
SF
Sac
SLC
Mon
LV
Bar
Flag
LA
Phoe
SD
Nog



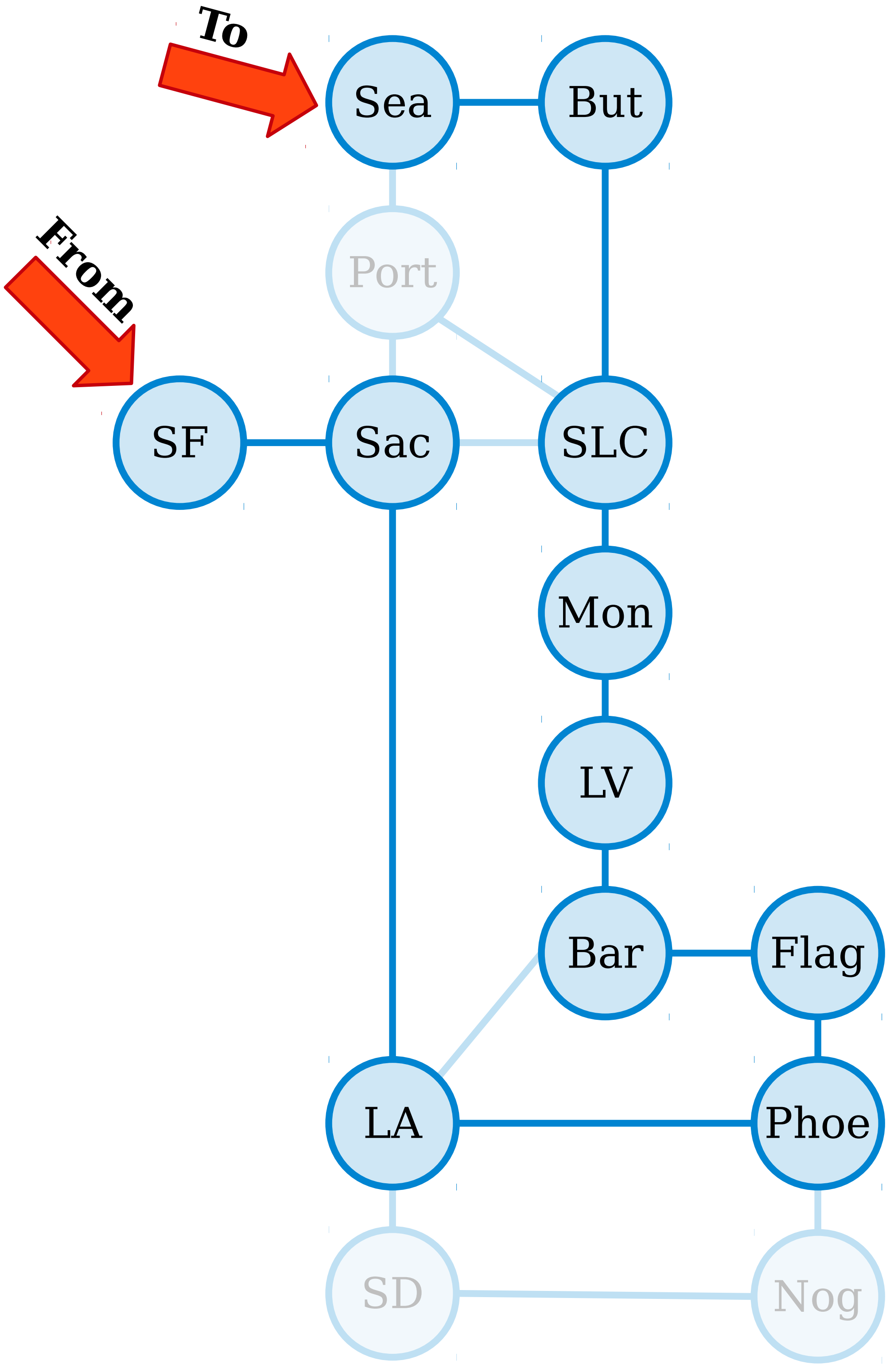
To
From
Sea
But
Port
SF
Sac
SLC
Mon
LV
Bar
Flag
LA
Phoe
SD
Nog



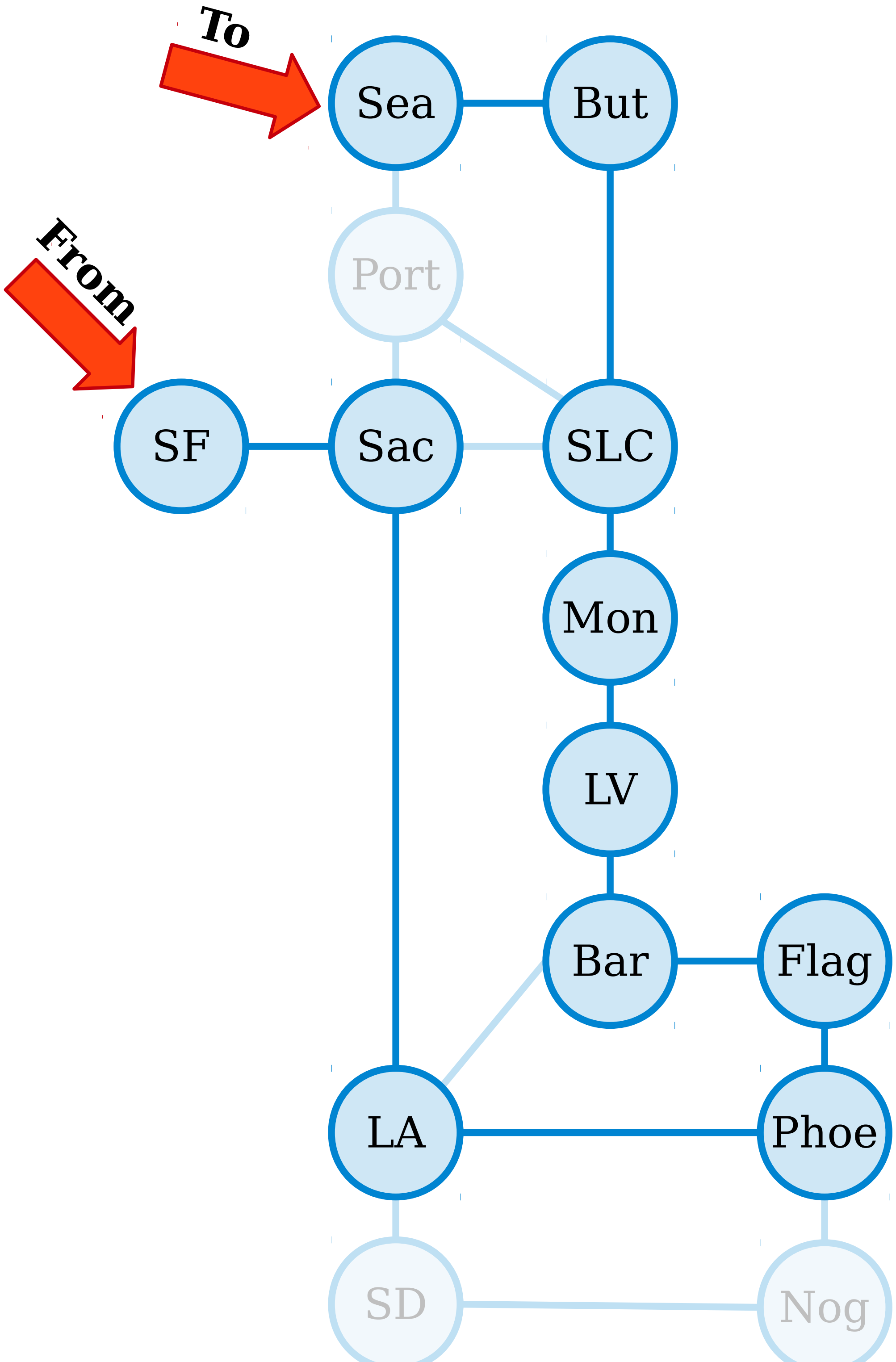
To
From
Sea
But
Port
SF
Sac
SLC
Mon
LV
Bar
Flag
LA
Phoe
SD
Nog



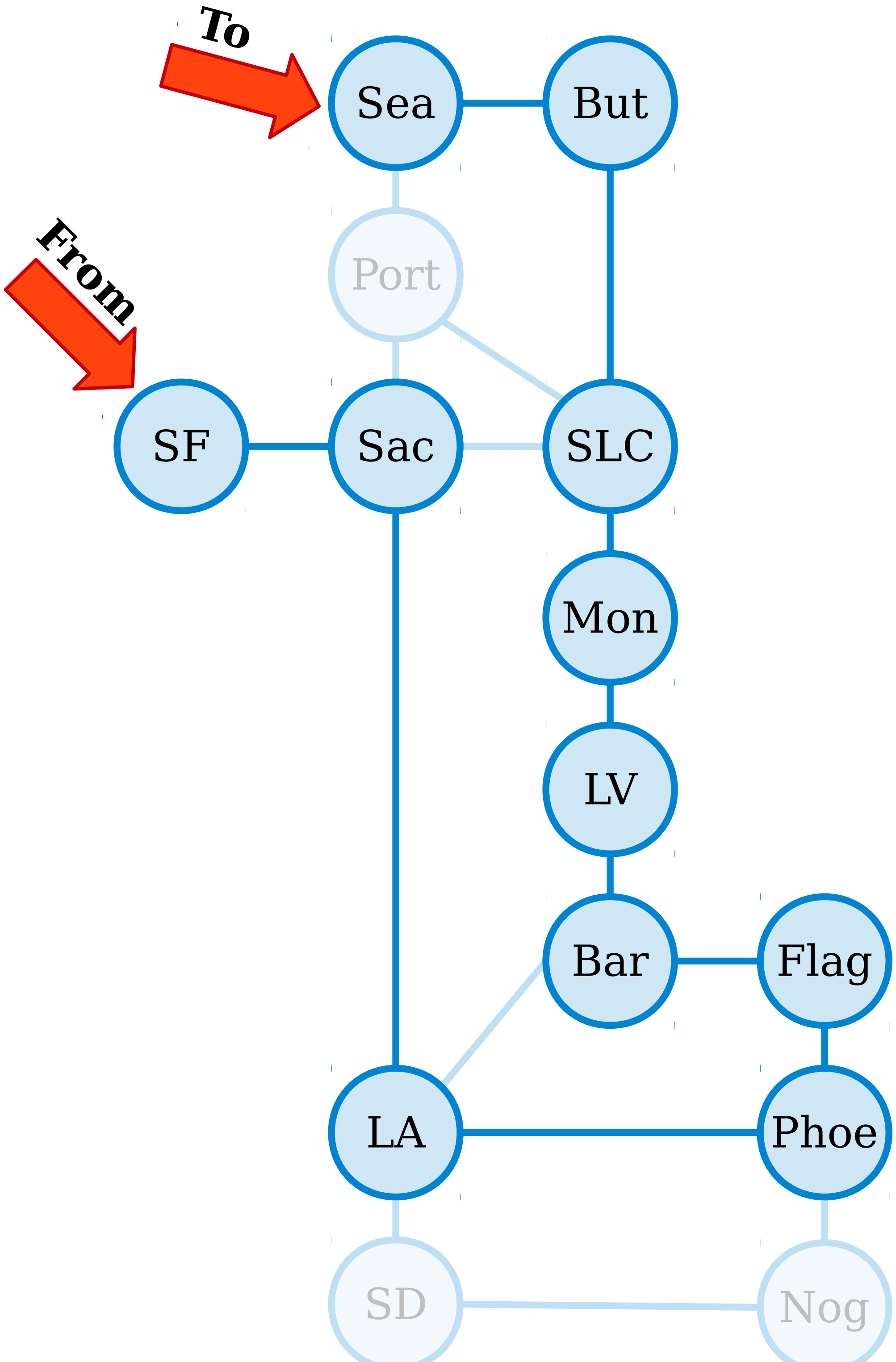
To
From
Sea
But
Port
SF
Sac
SLC
Mon
LV
Bar
Flag
LA
Phoe
SD
Nog



To
From
Sea
But
Port
SF
Sac
SLC
Mon
LV
Bar
Flag
LA
Phoe
SD
Nog

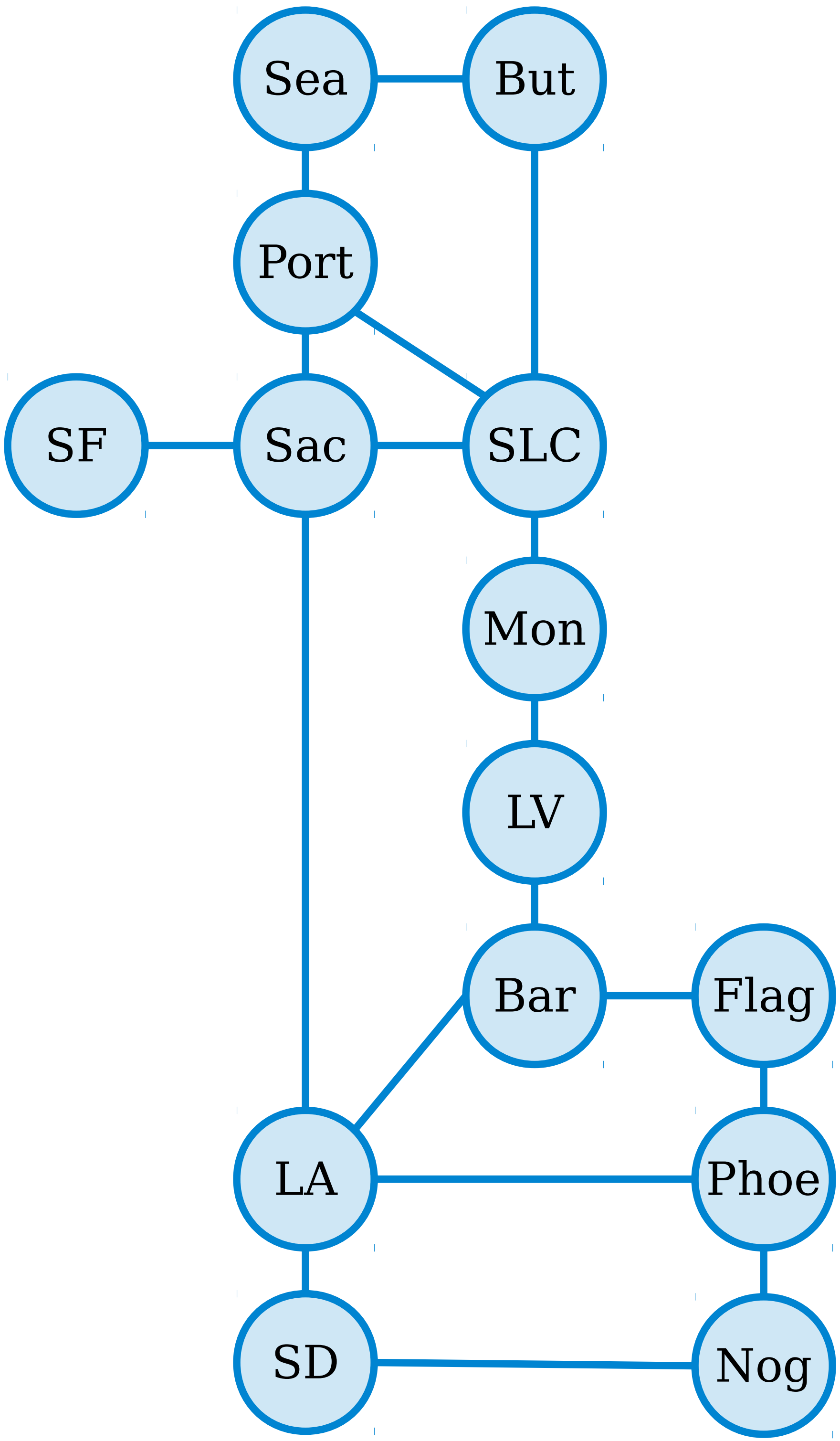


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.



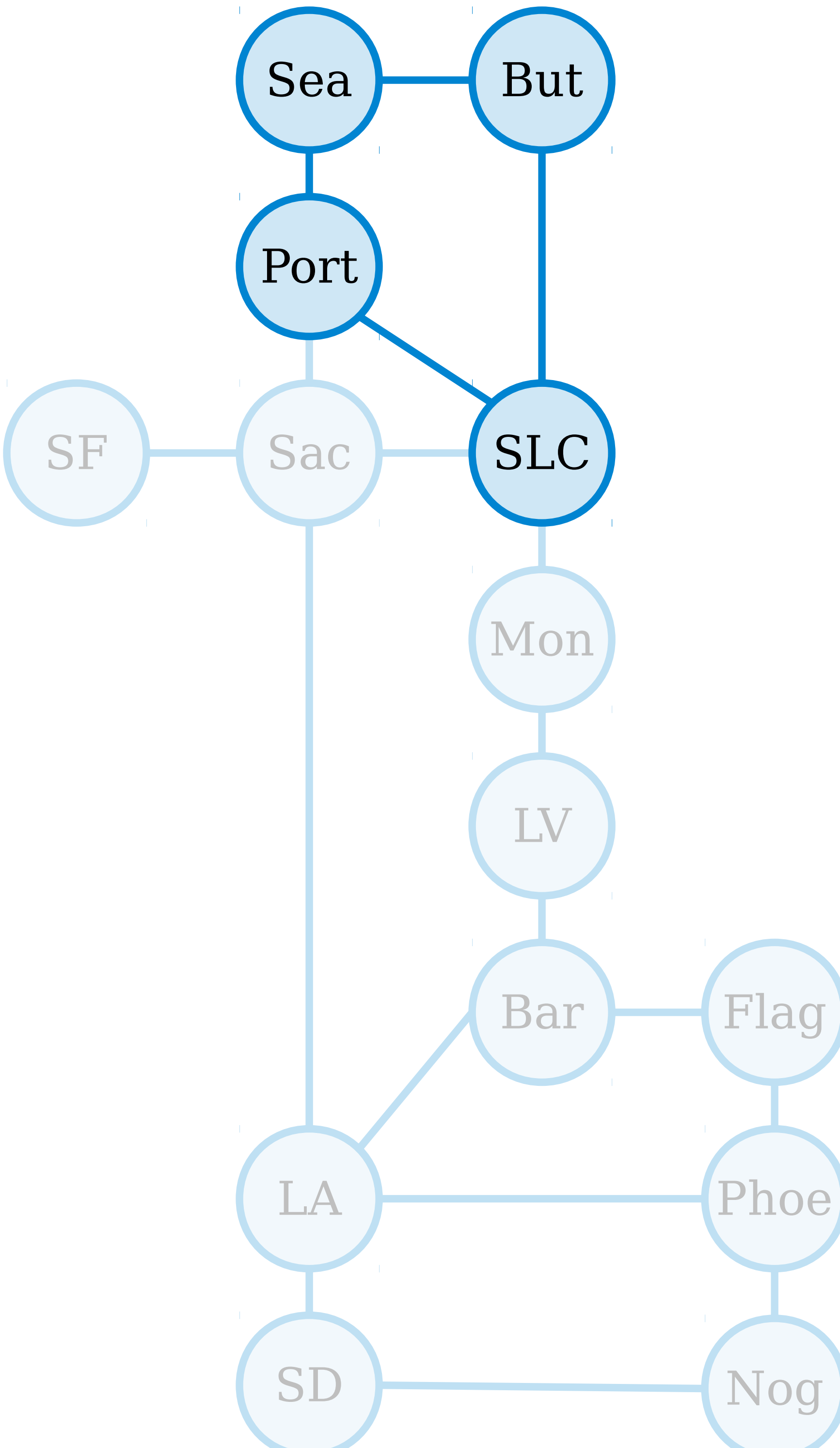
A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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



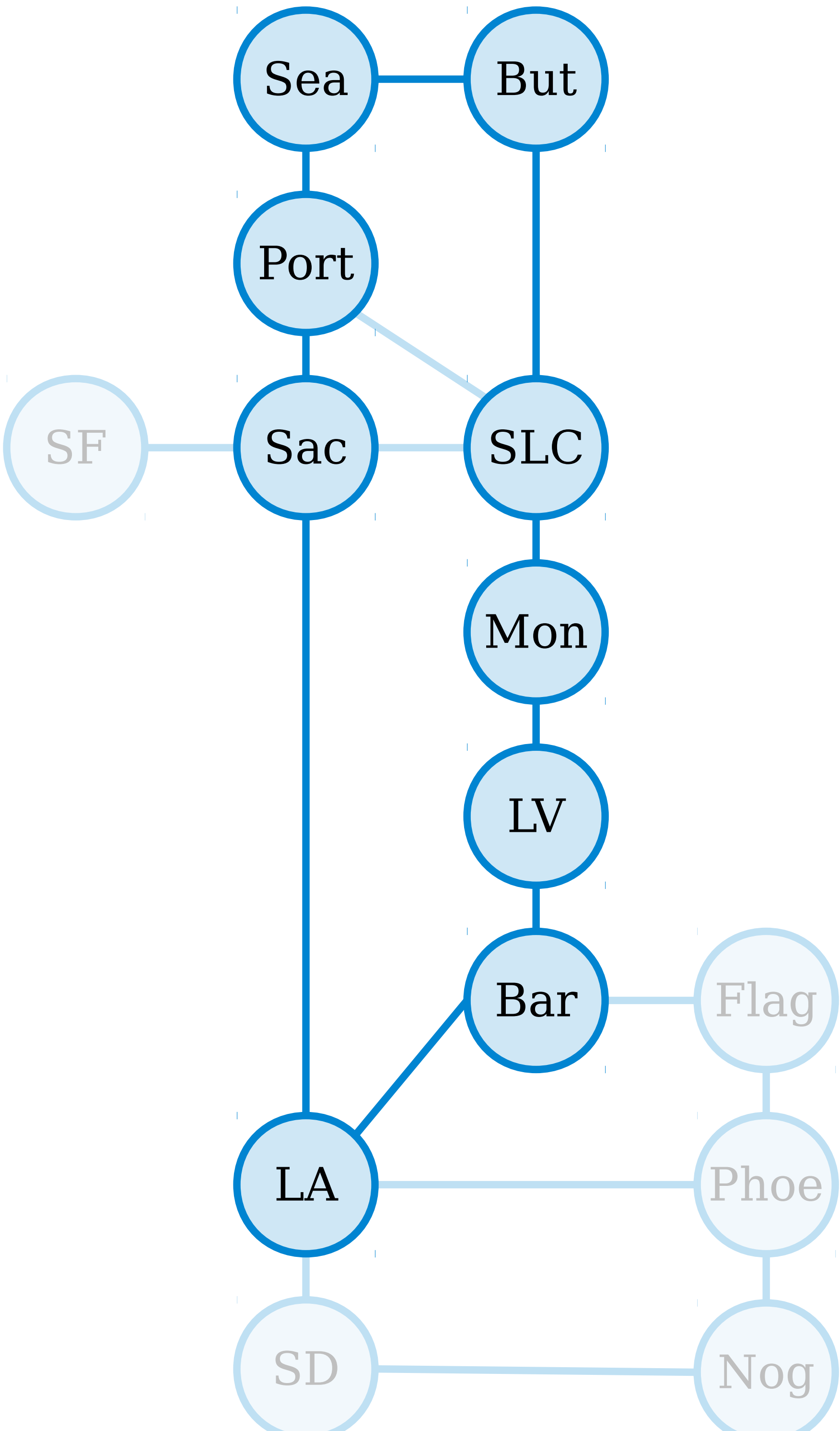
A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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



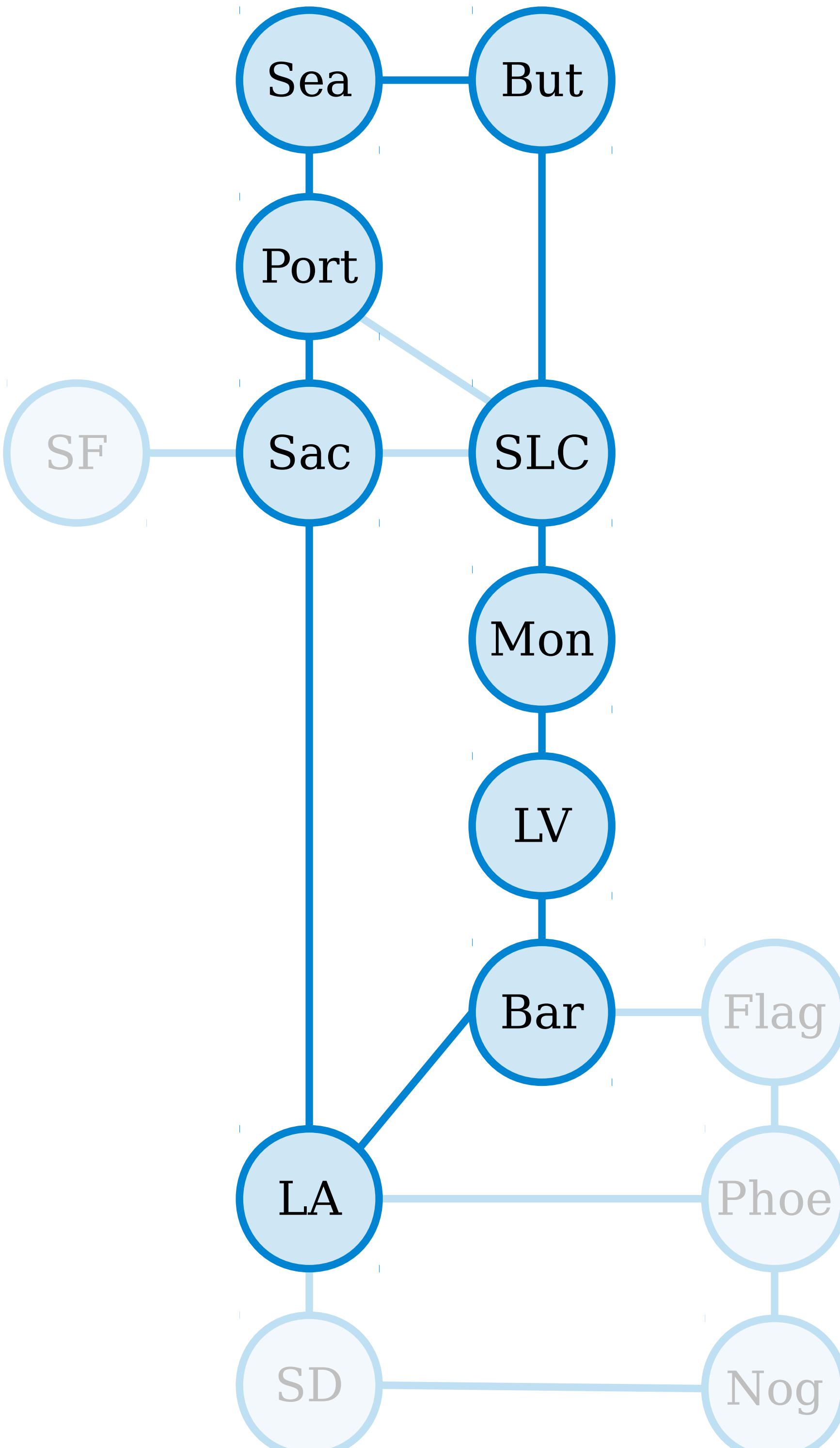
A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

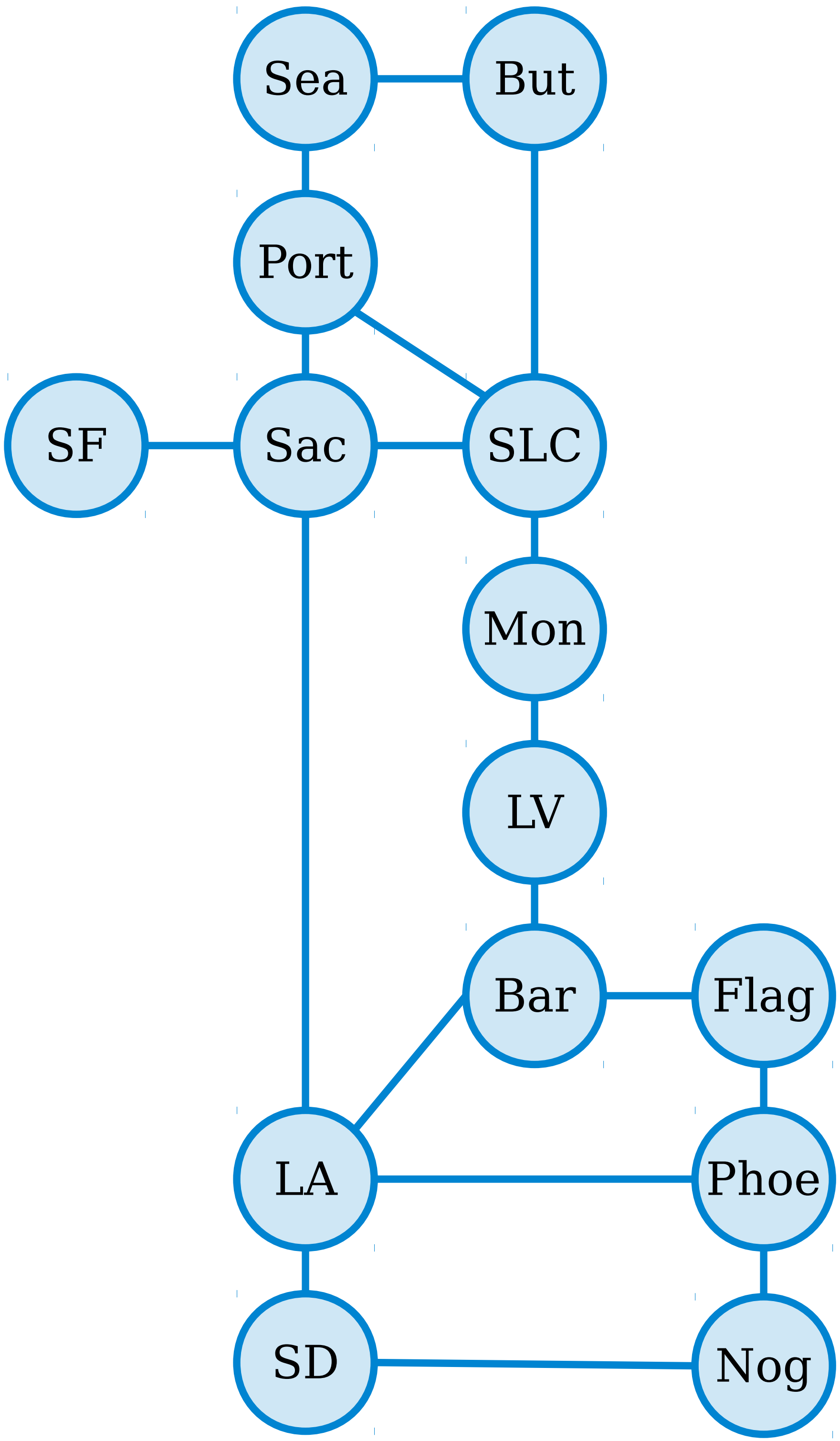
The ***length*** of a path is the number of edges in it.



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

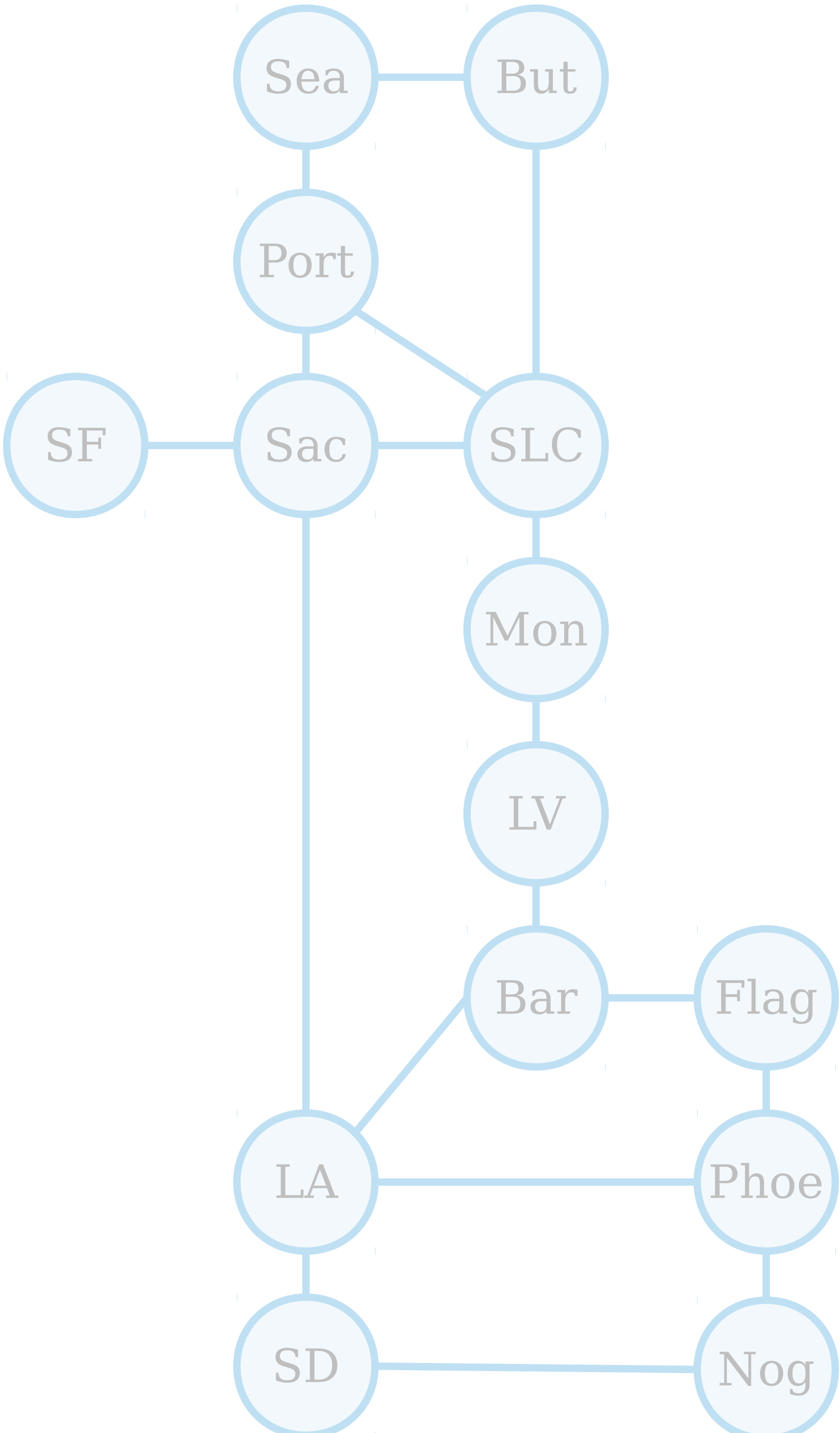
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

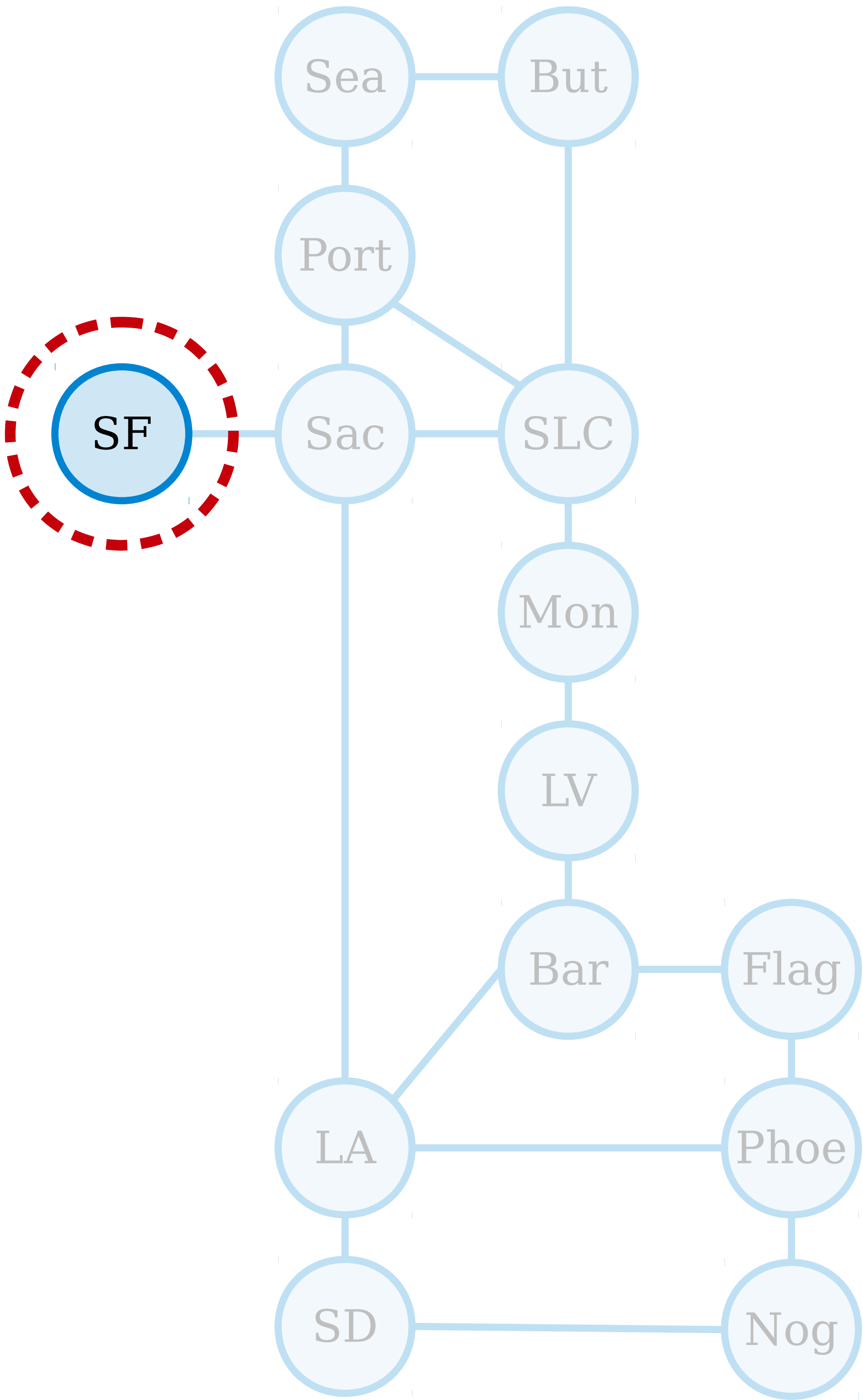
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

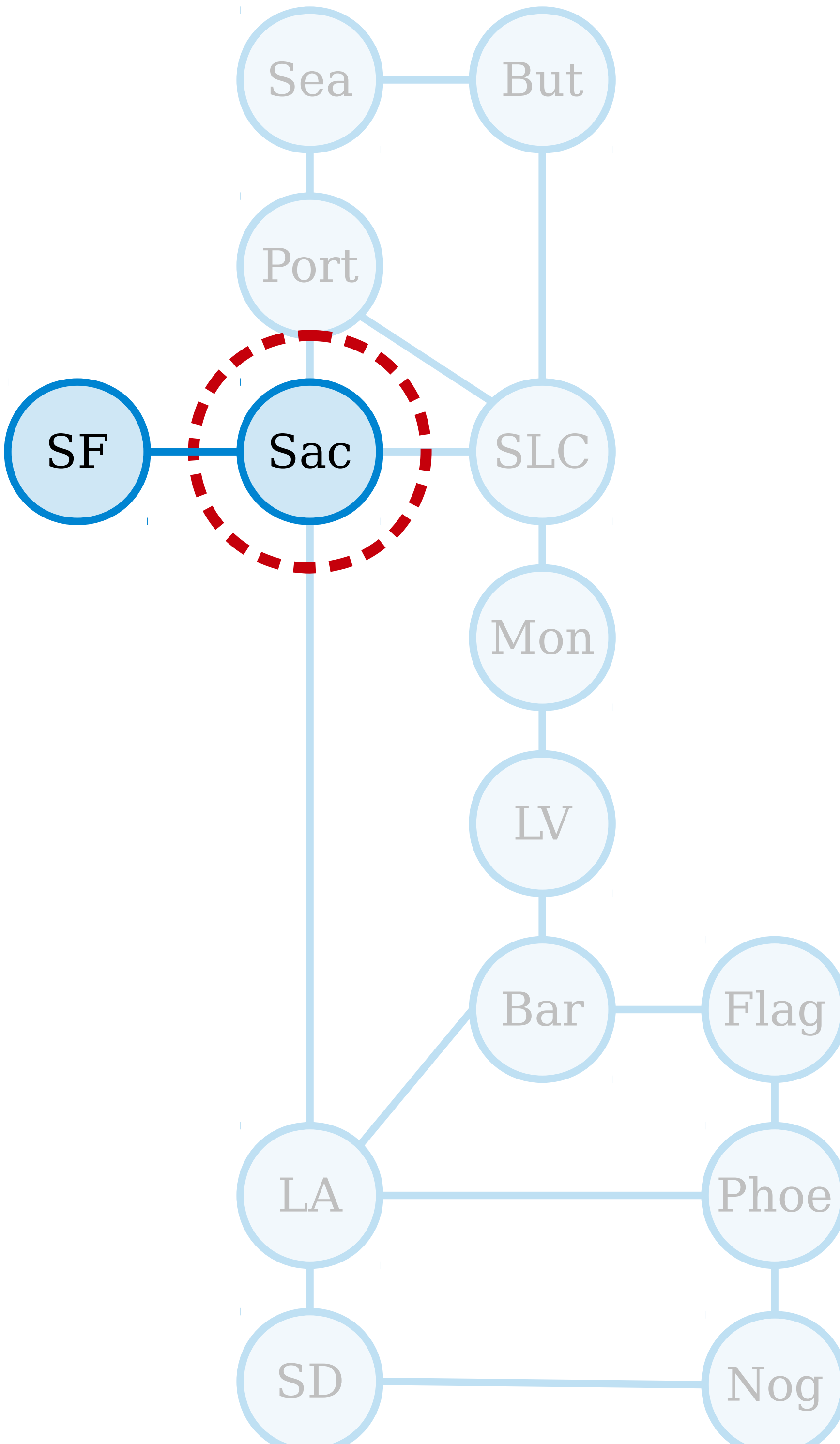
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

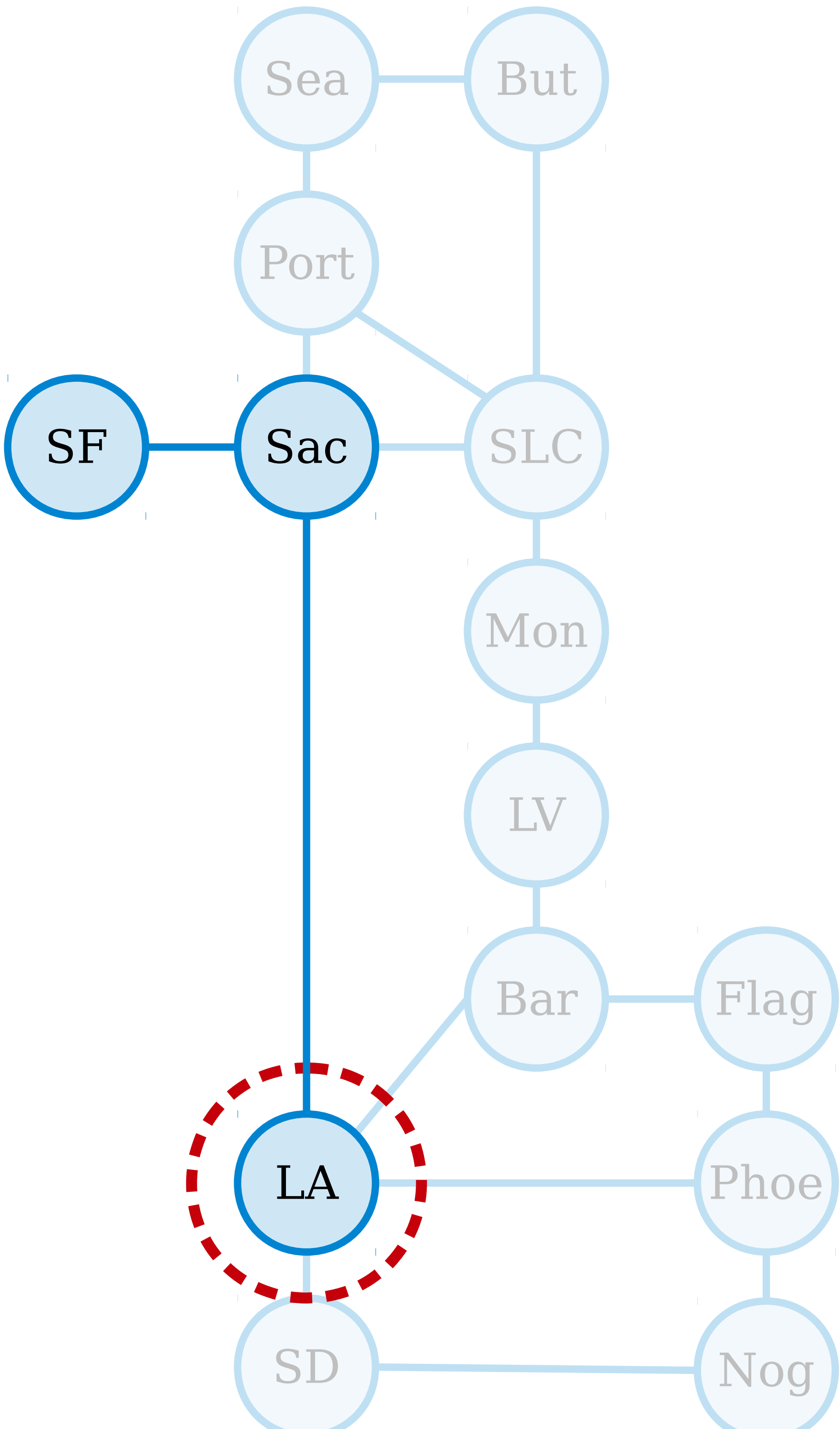
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

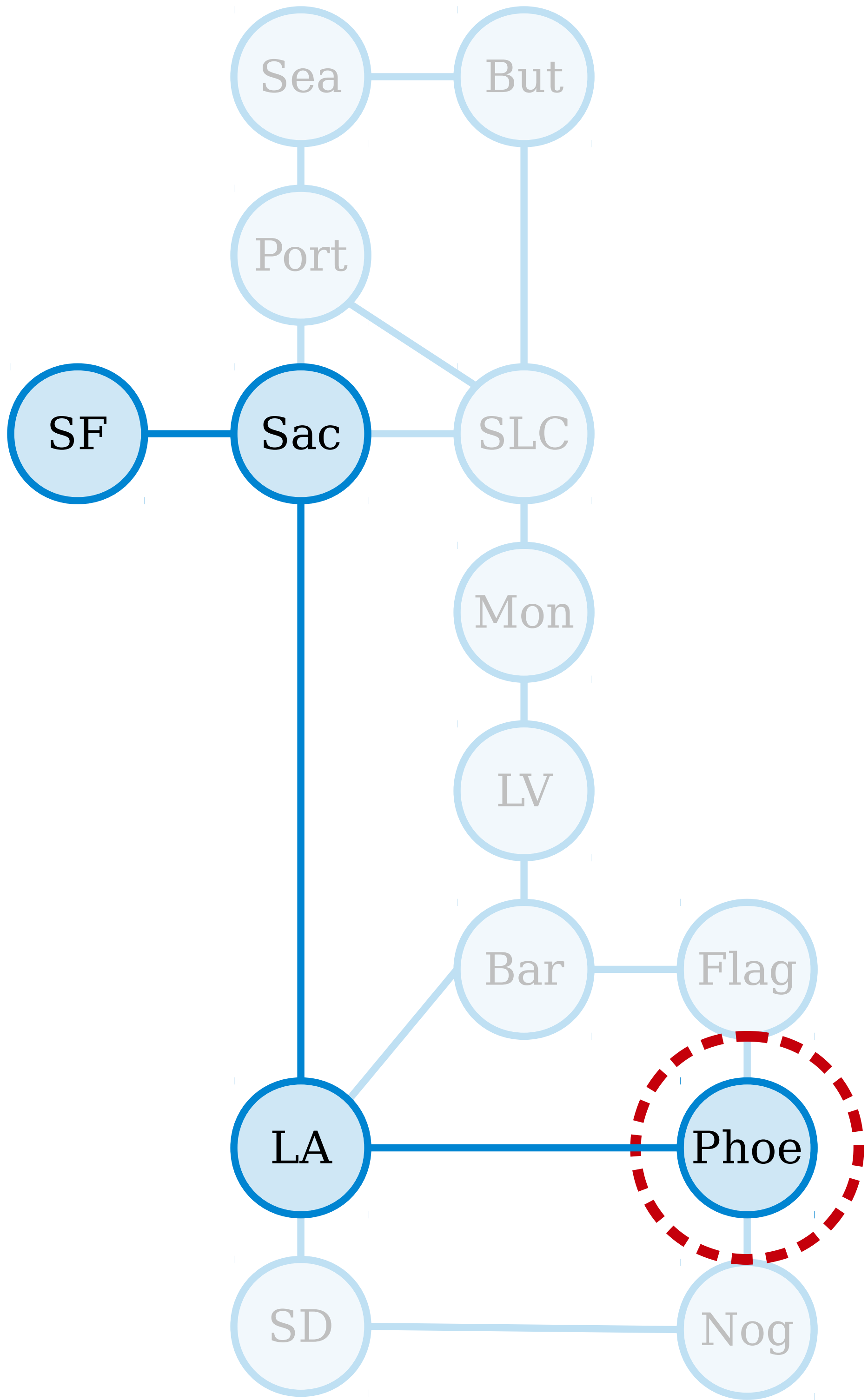
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

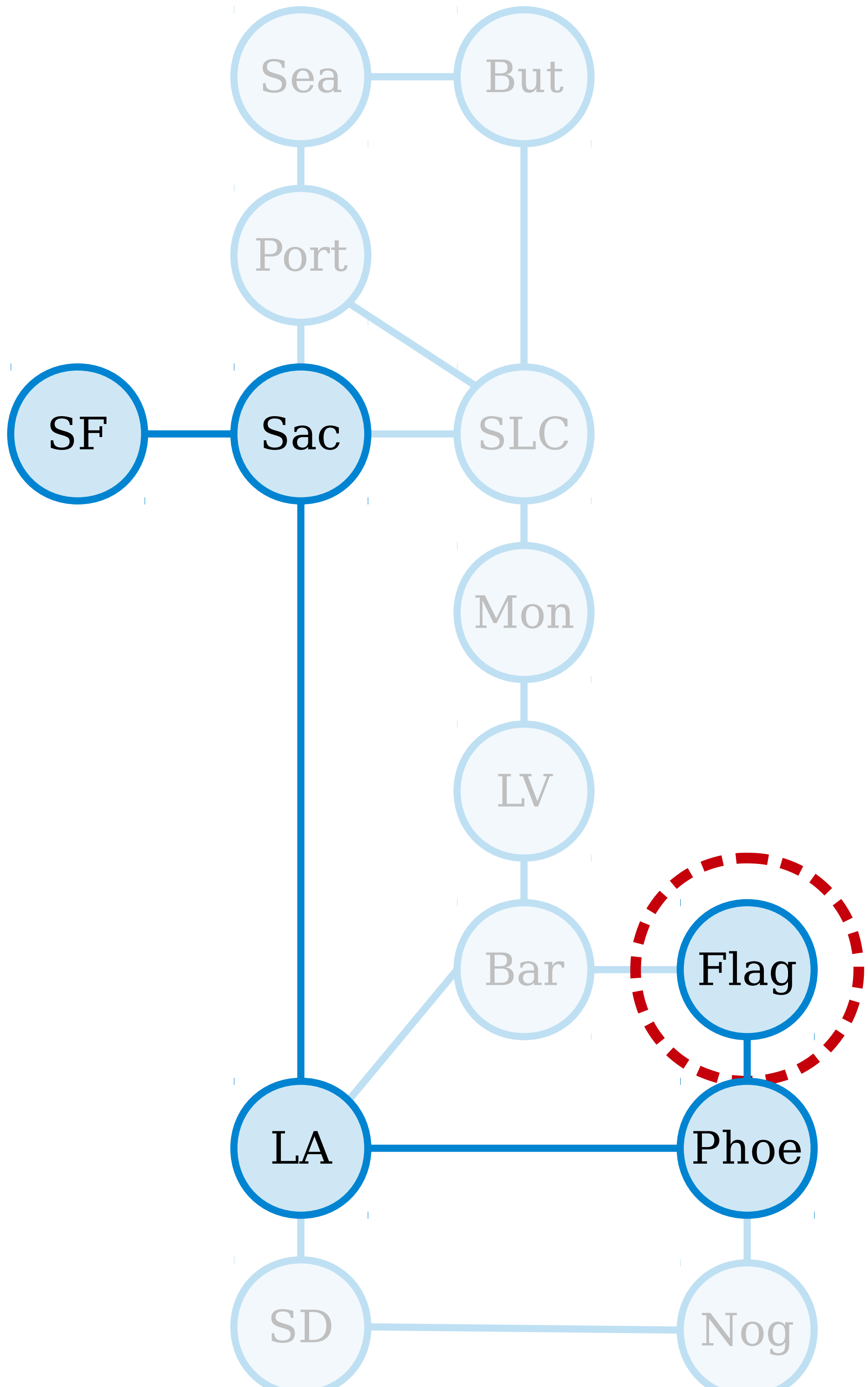
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

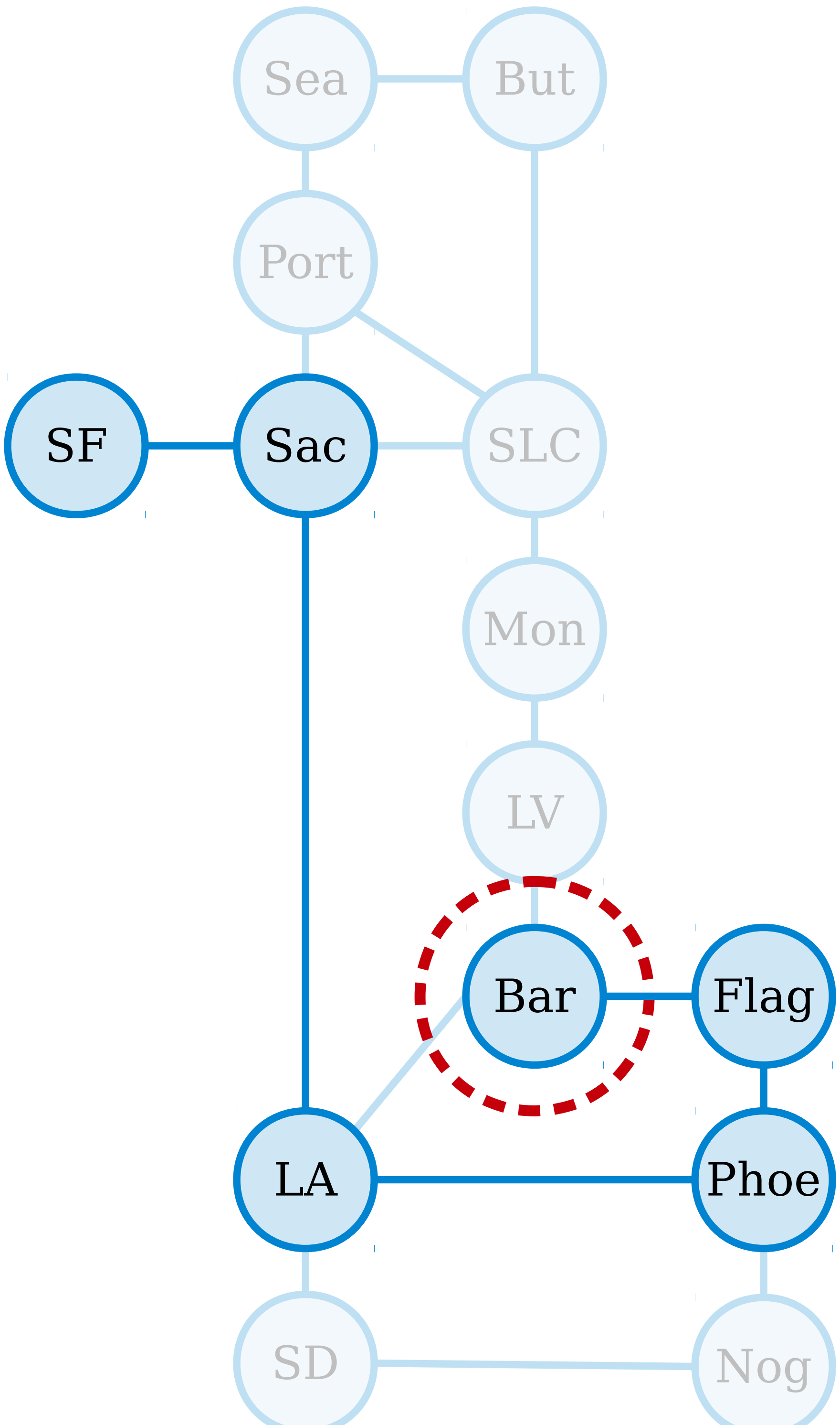
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

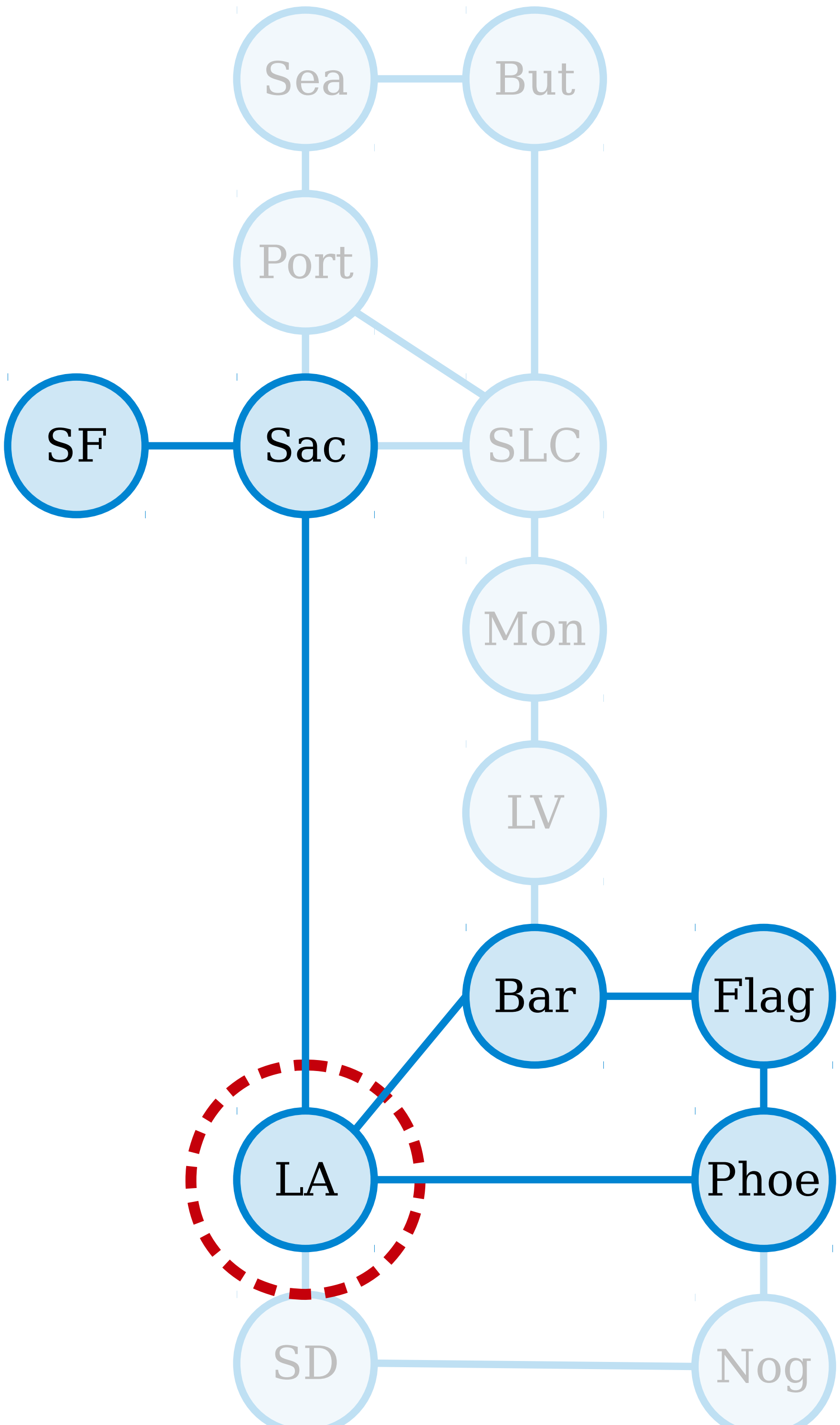
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

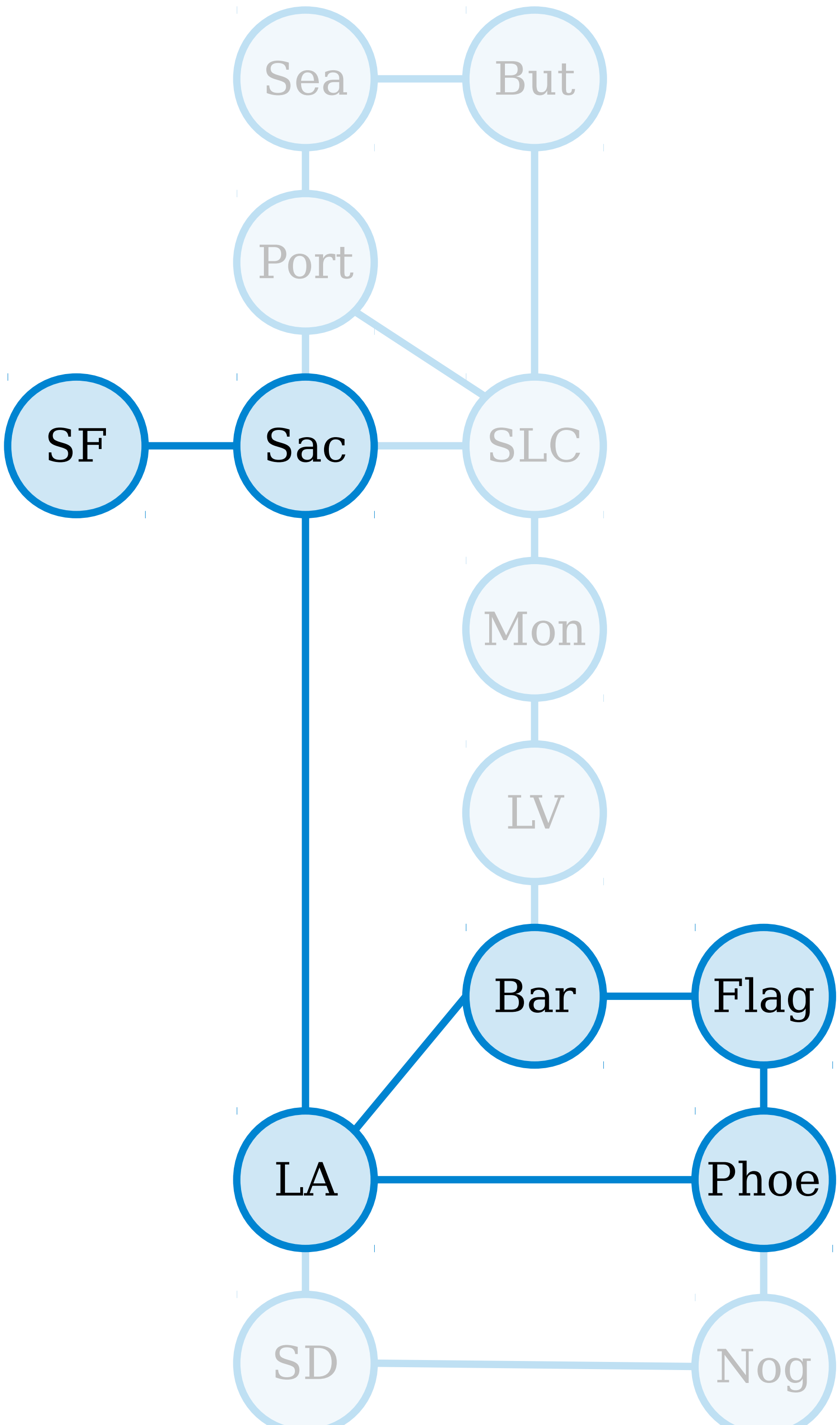
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

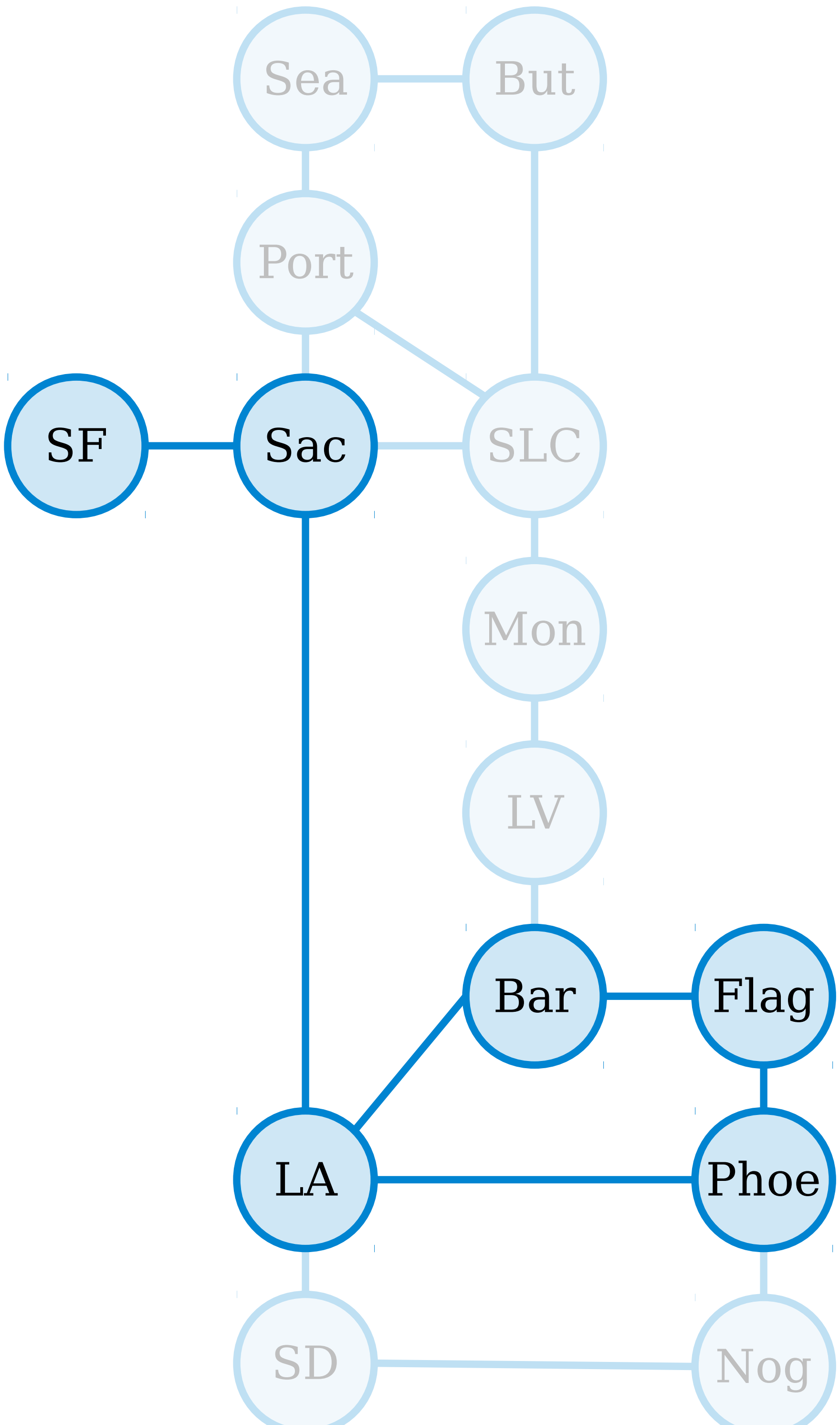
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

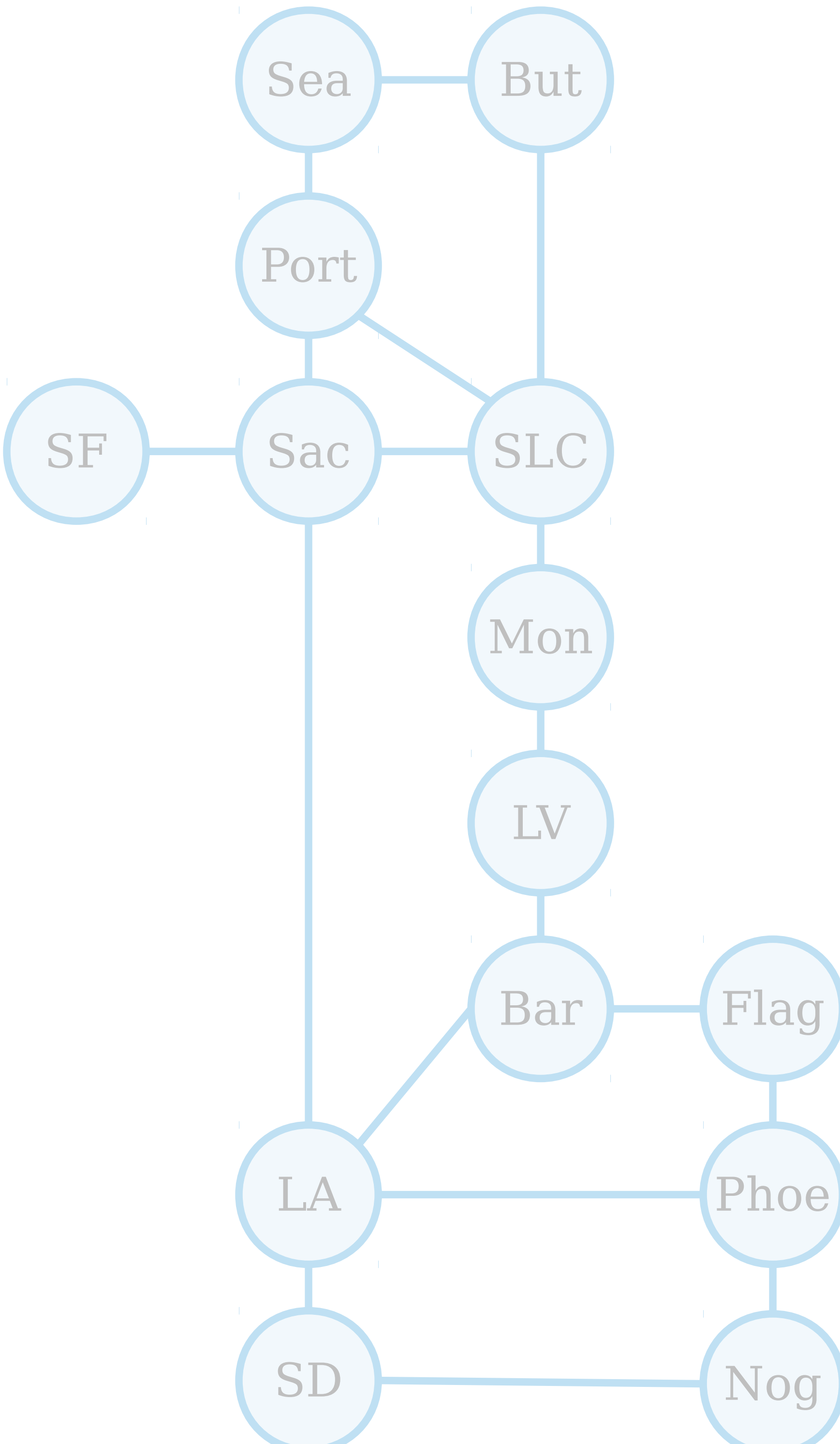


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.

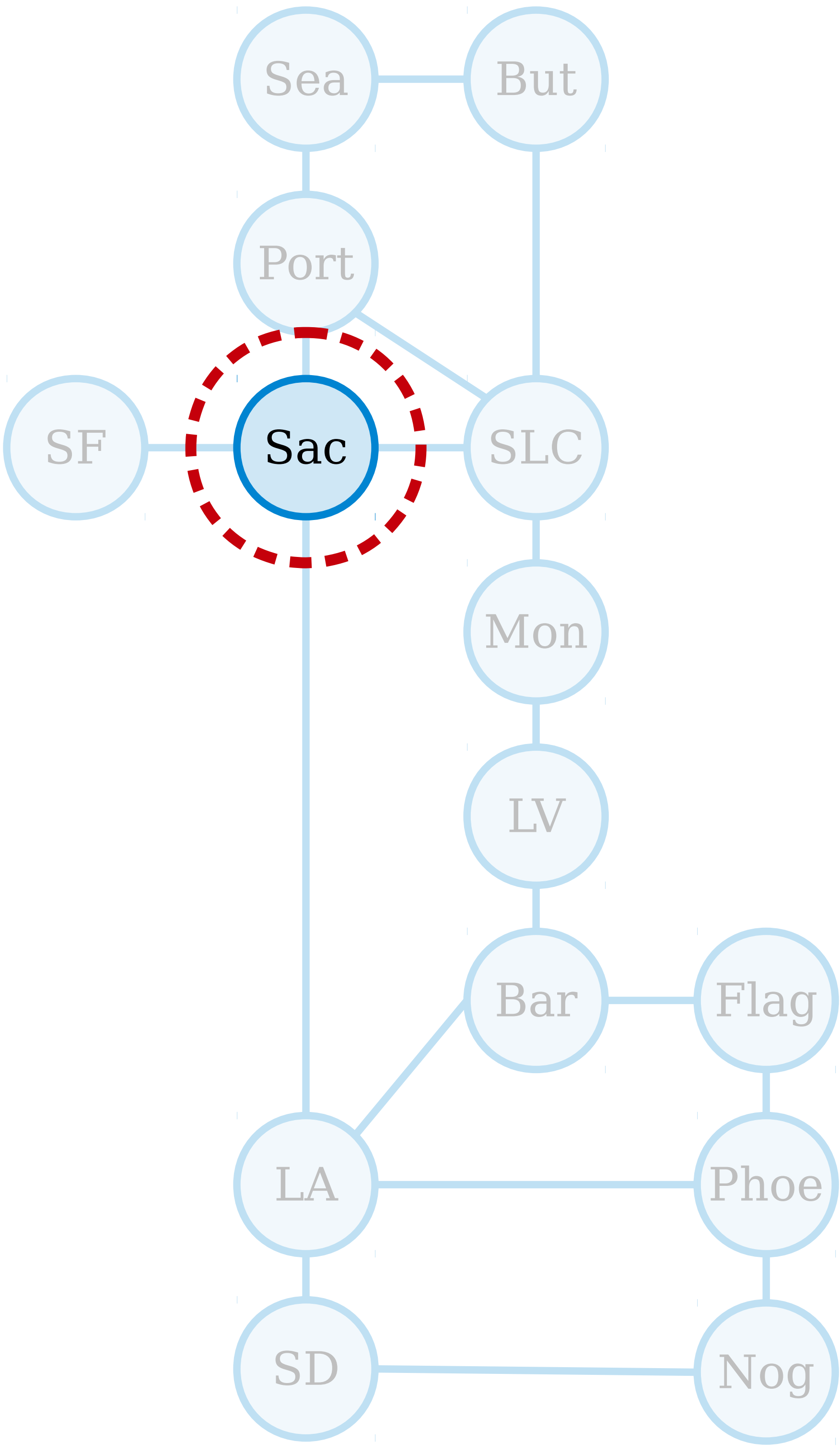


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.

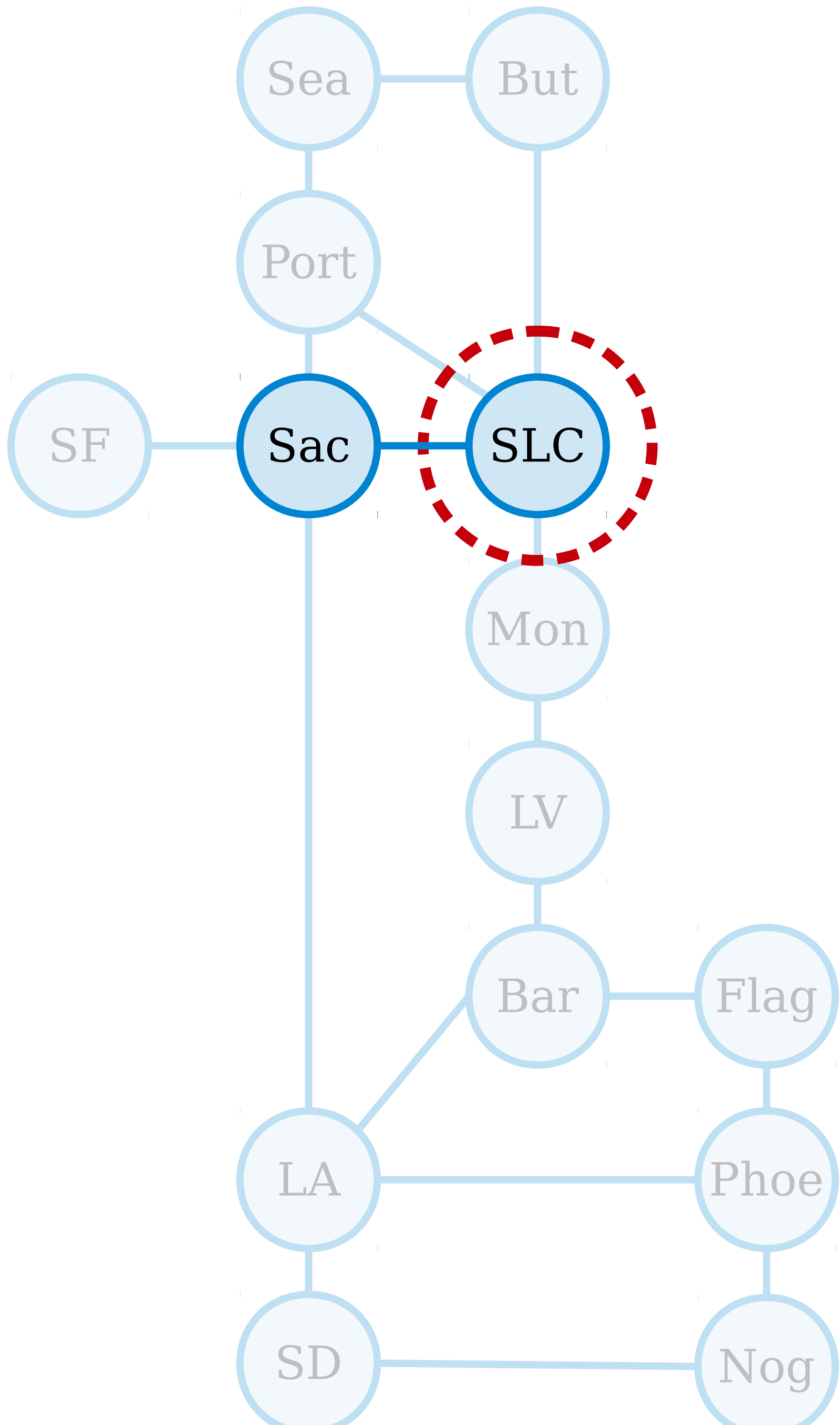


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.

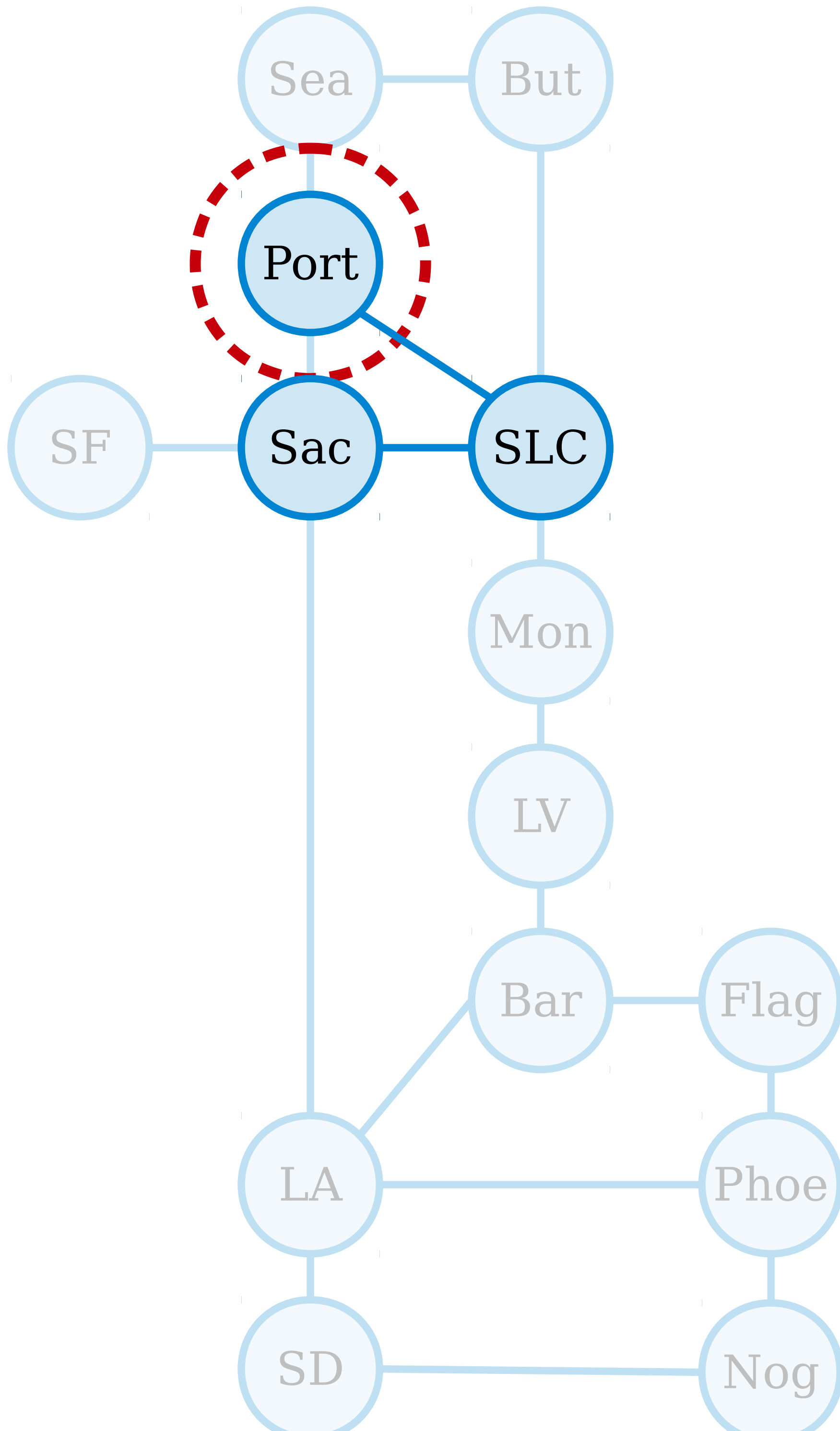


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.

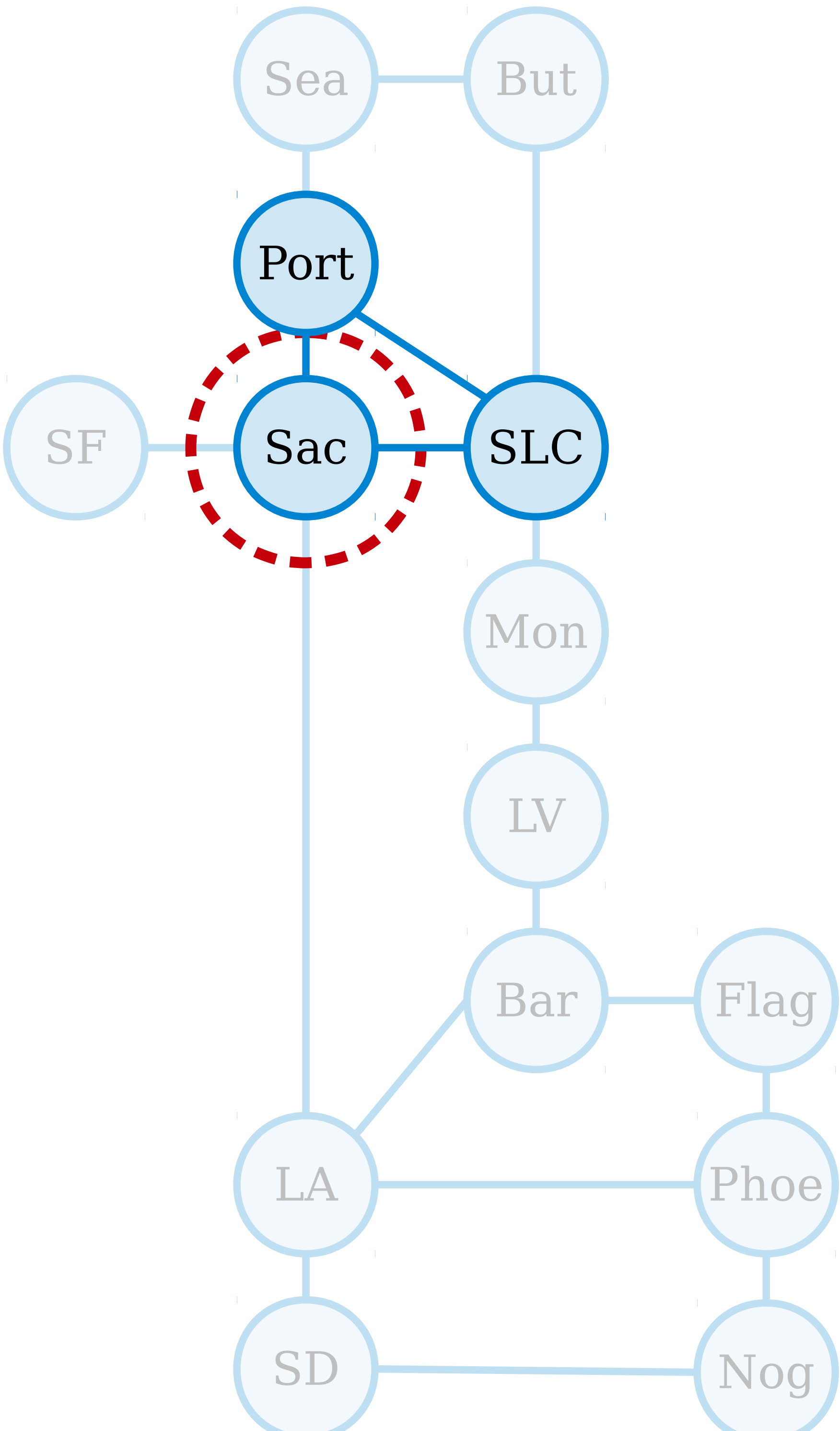


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.

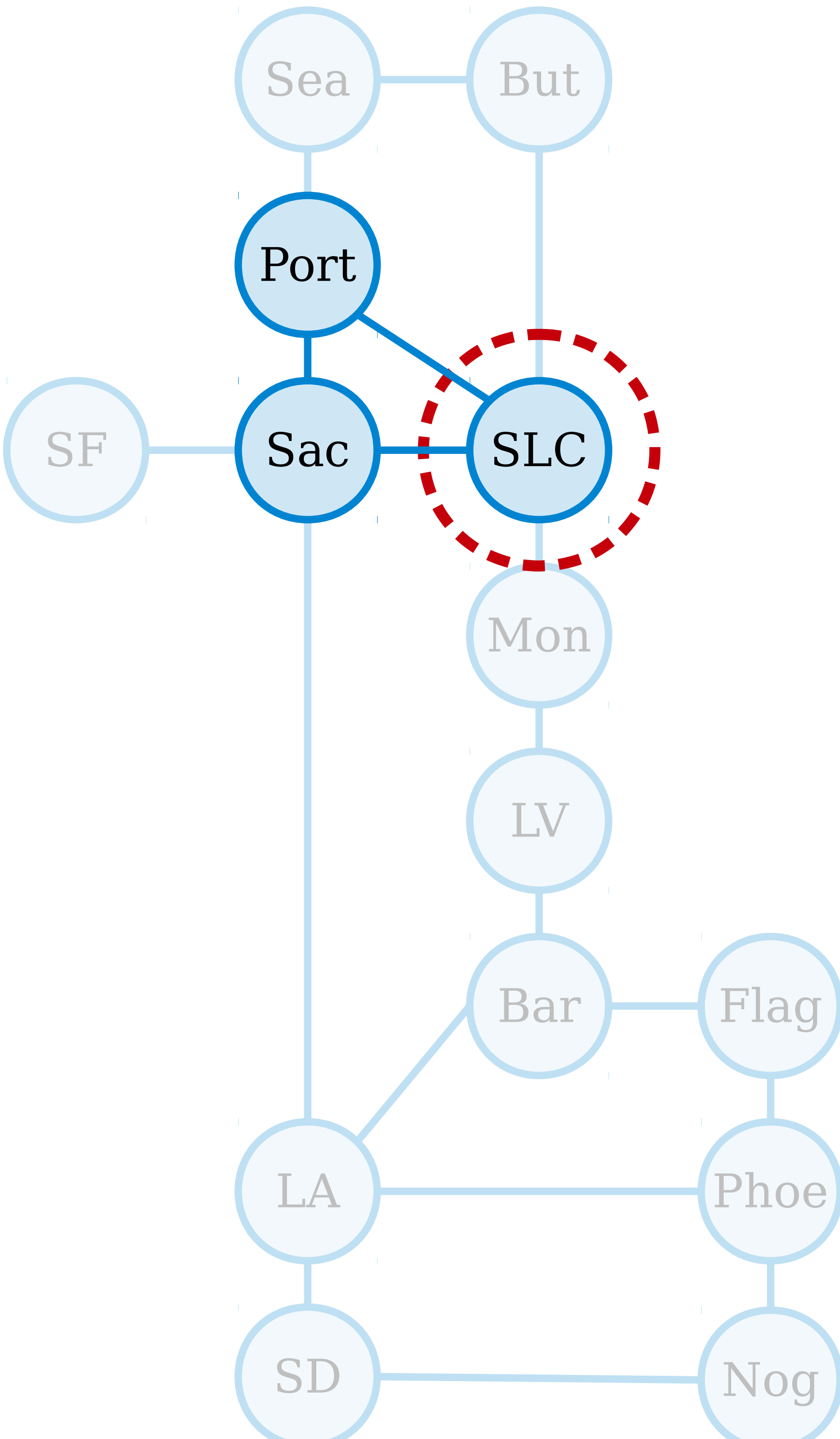


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.

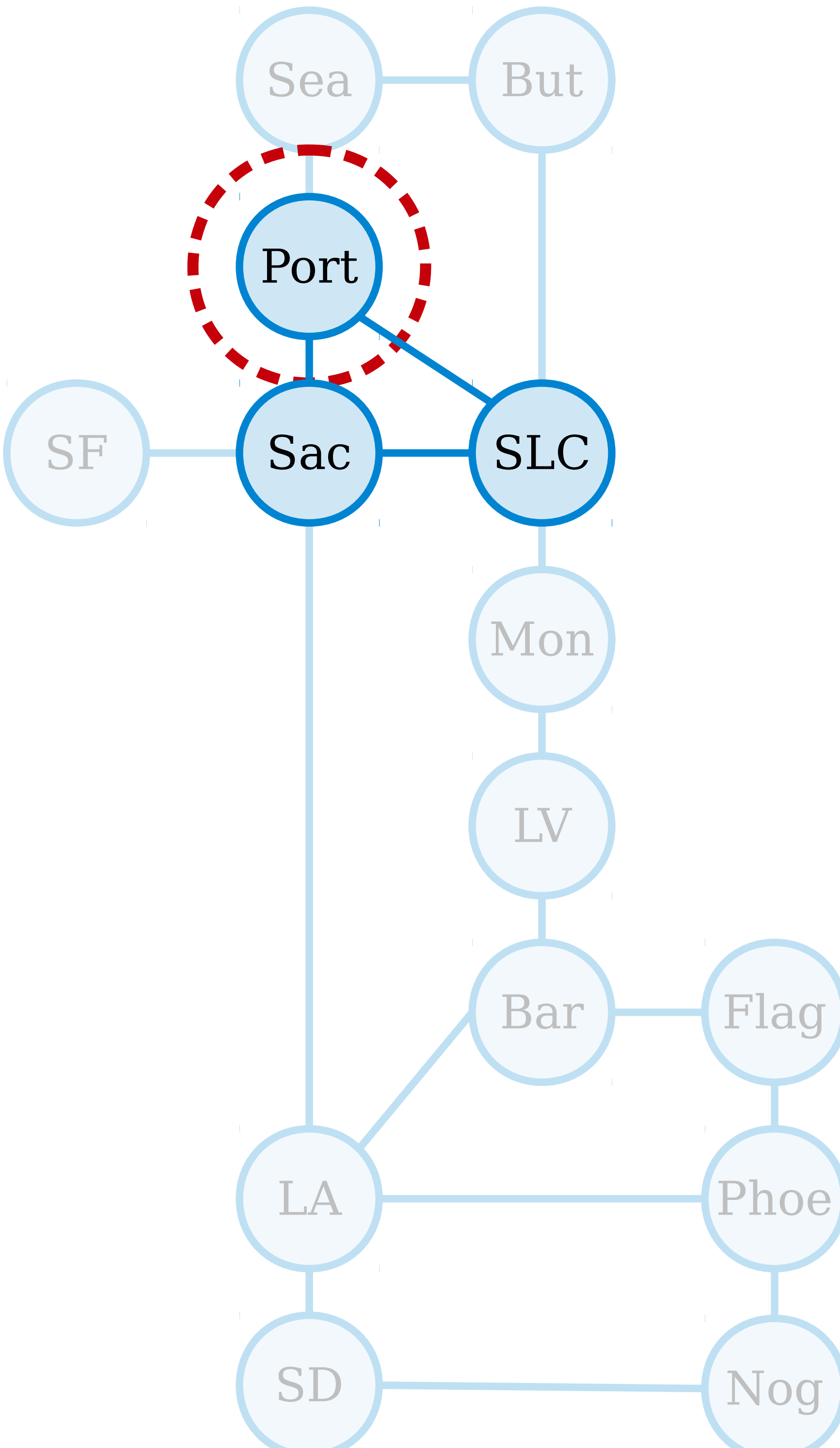


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.

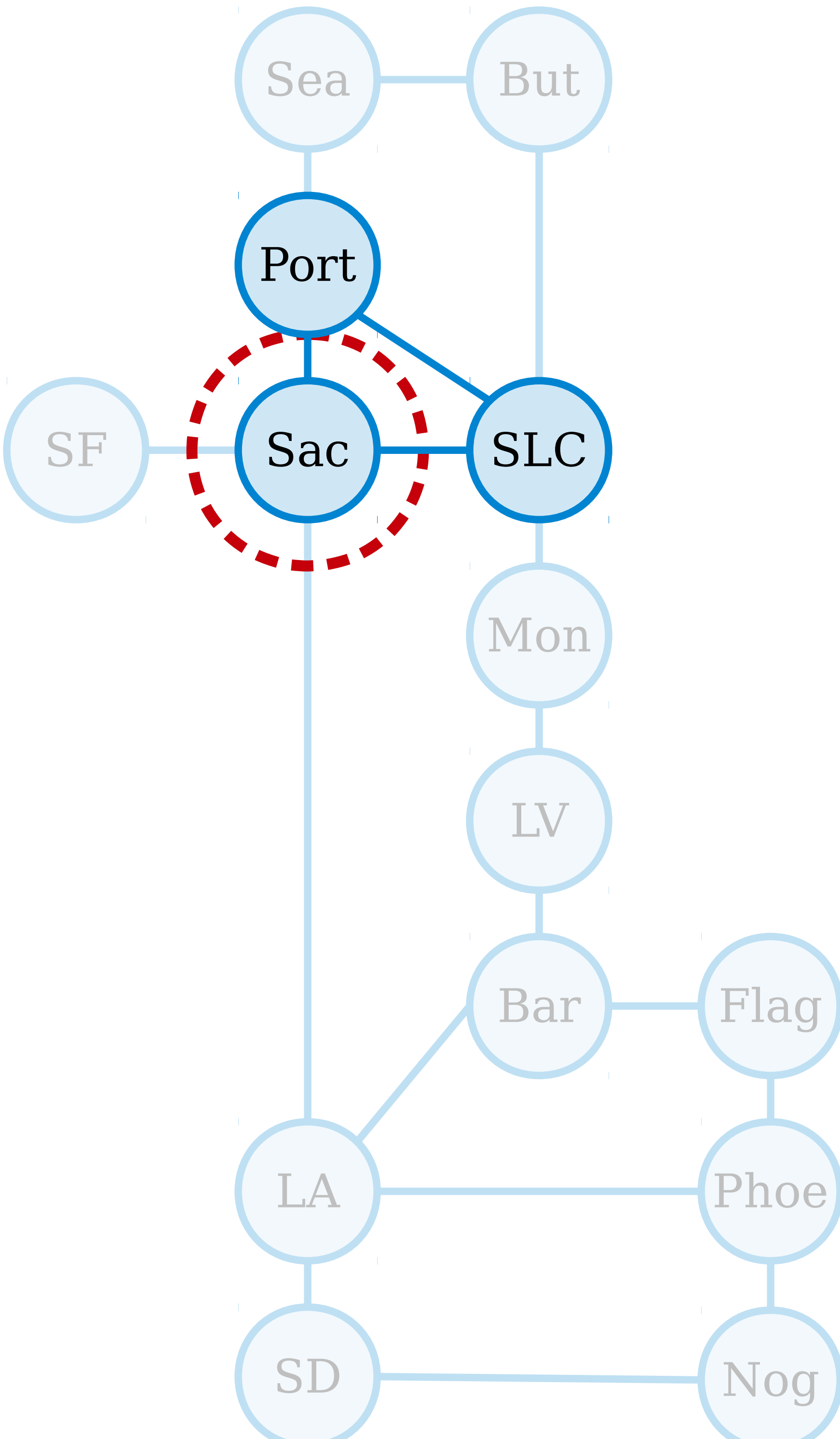


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.

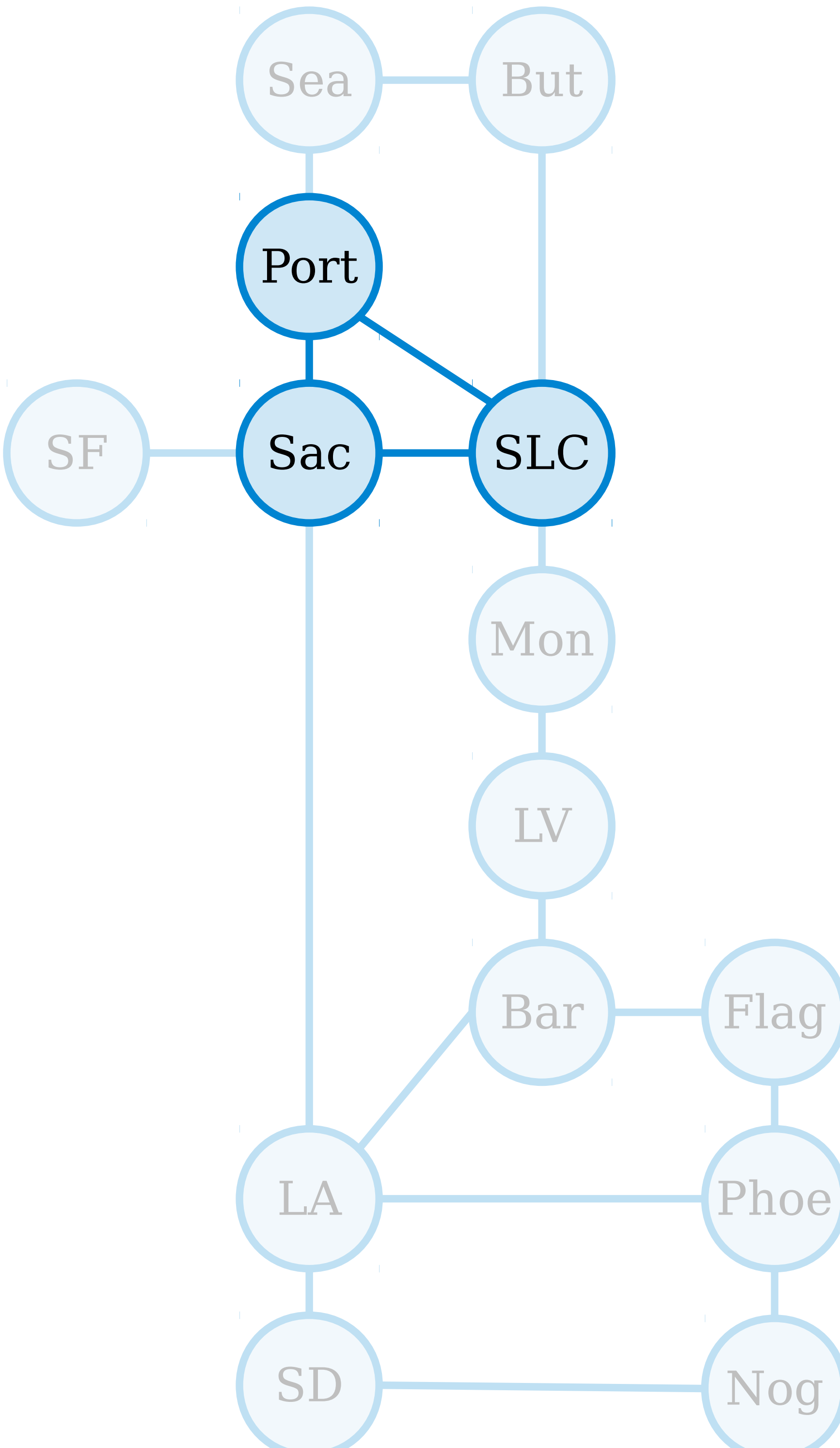


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.



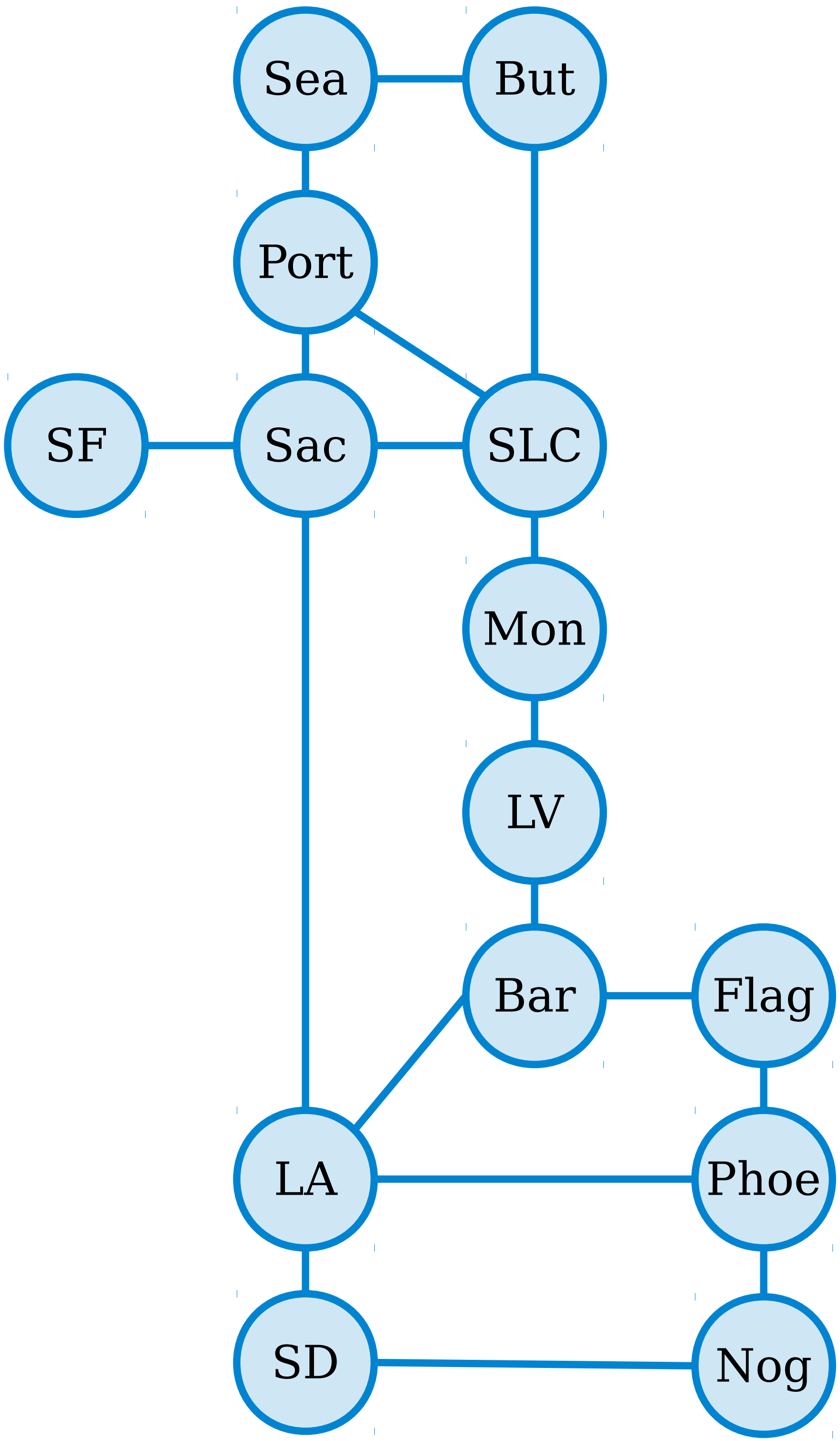
A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

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

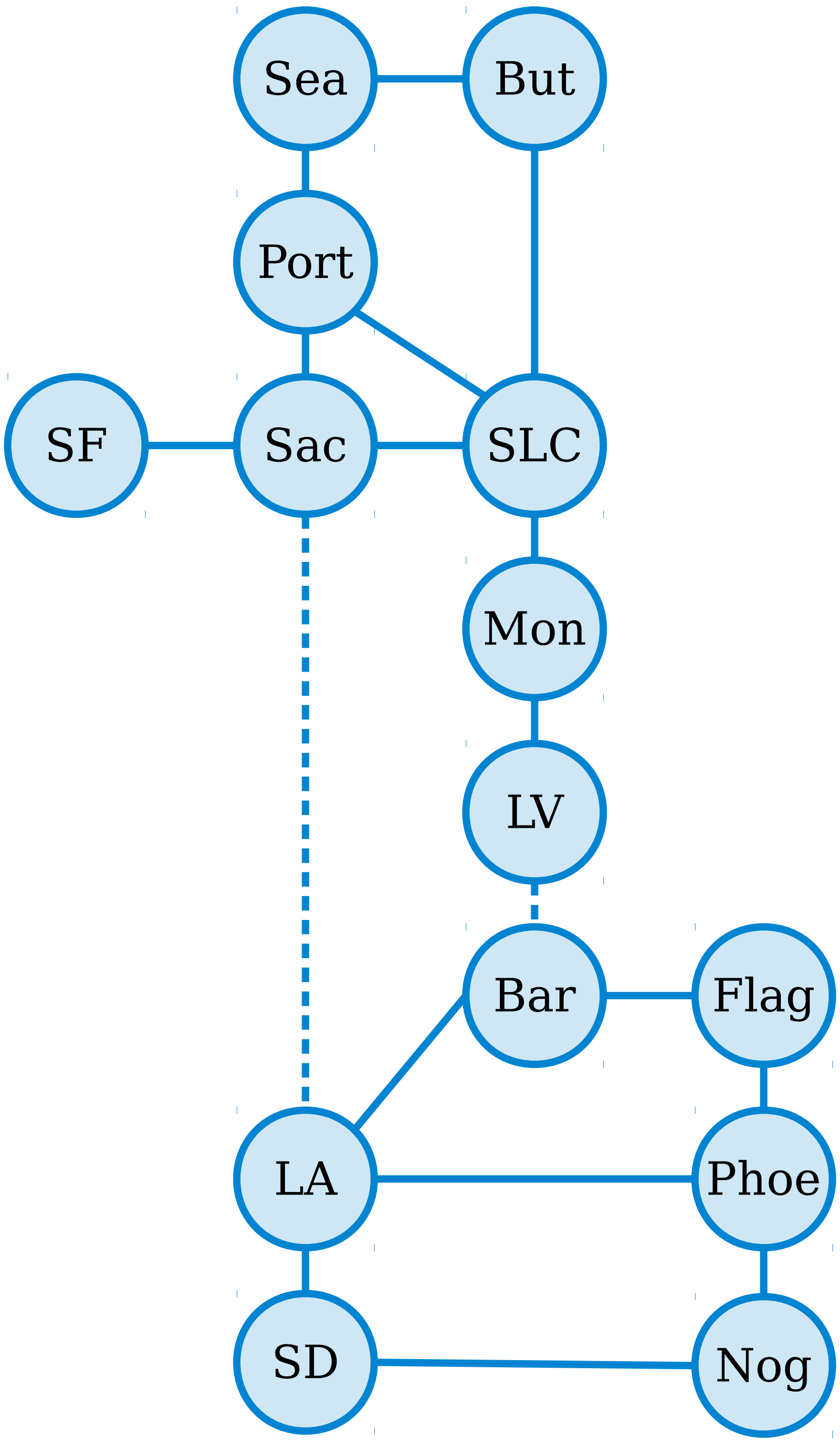
A ***cycle*** in a graph is a path from a node back to itself. (By convention, a cycle cannot consist of a single node.)

A ***simple path*** in a graph is path that does not repeat any nodes or edges.

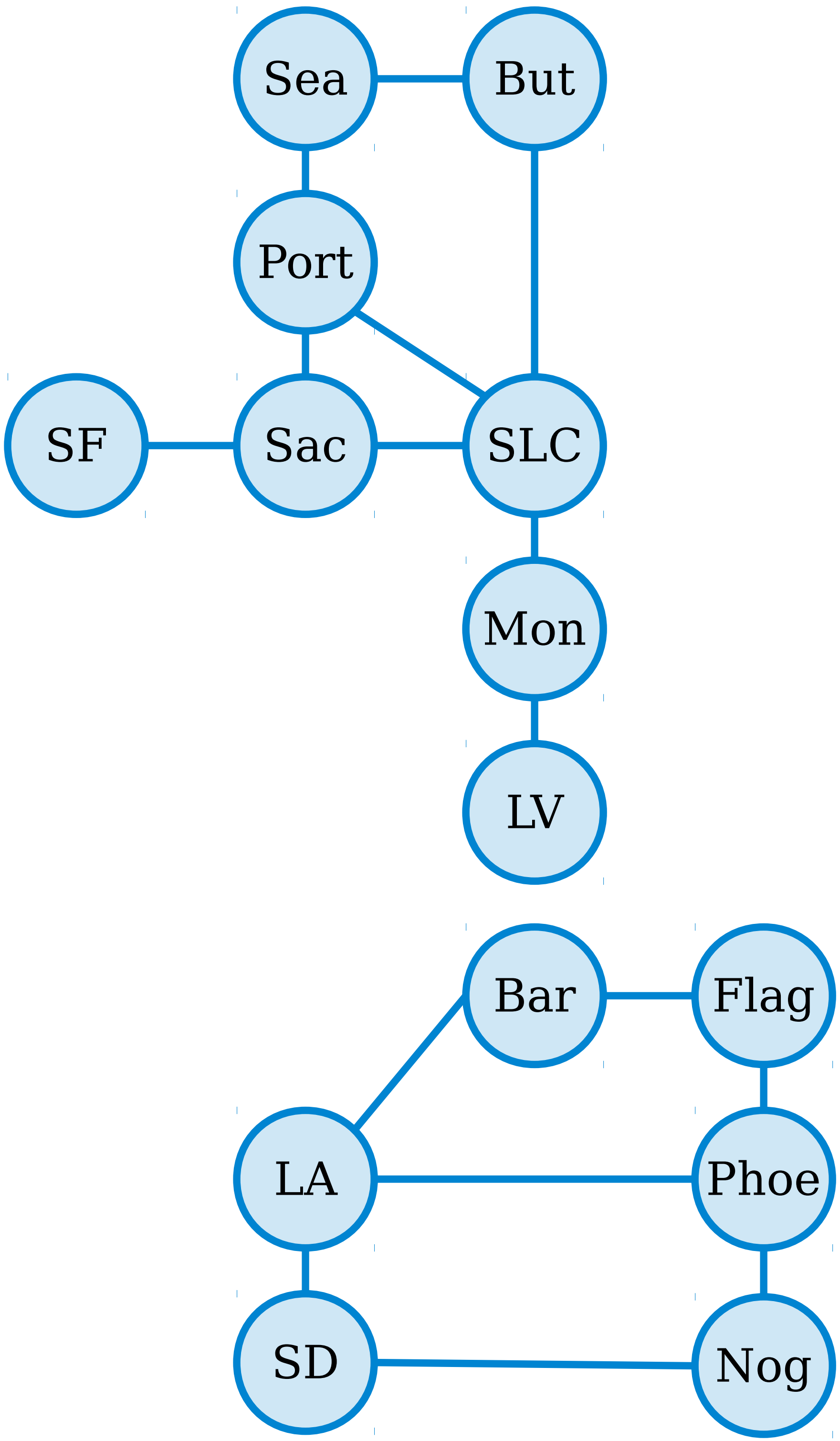
A ***simple cycle*** in a graph is cycle that does not repeat any nodes or edges except the first/last node.



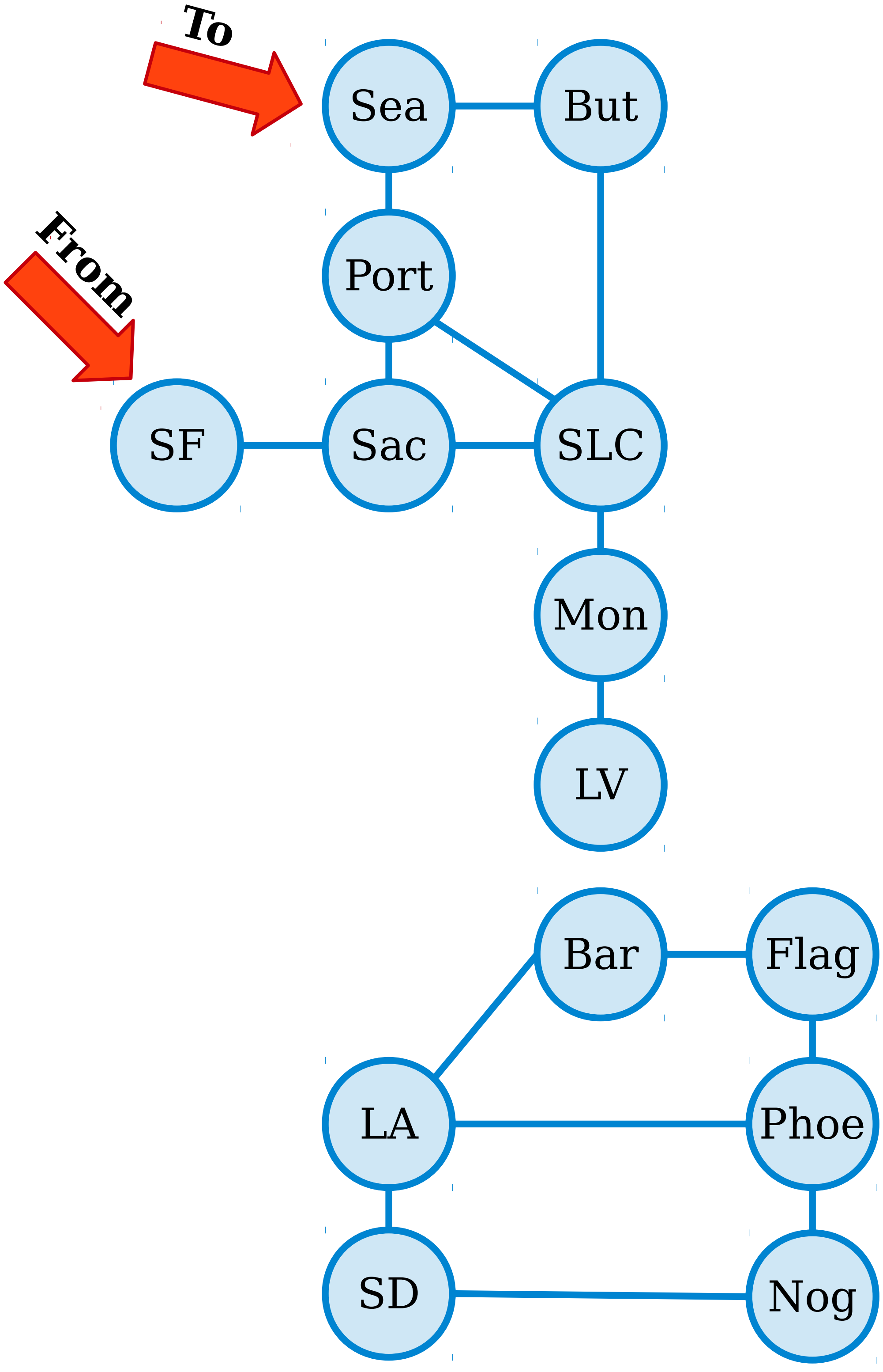
A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.



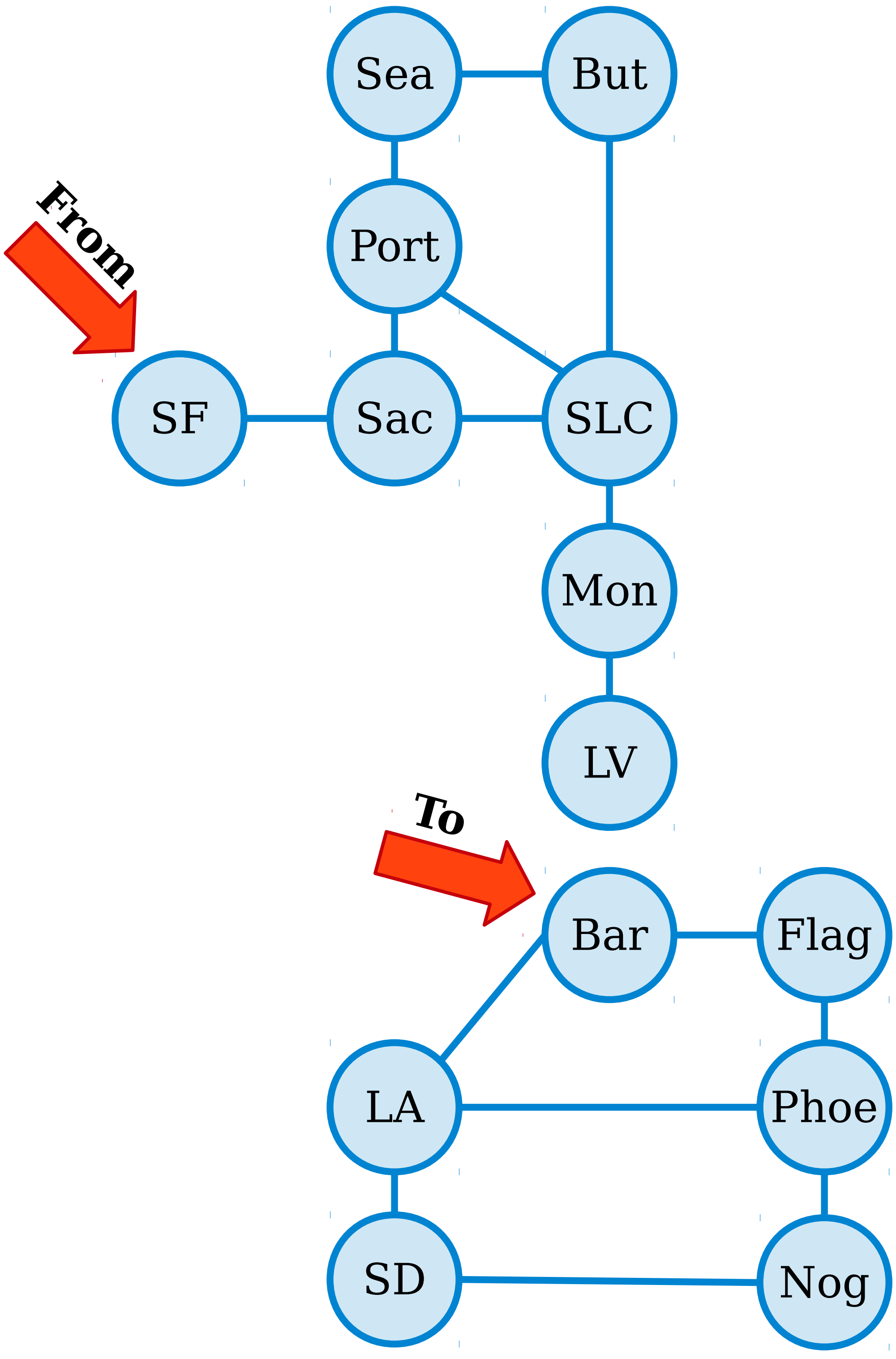
A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.



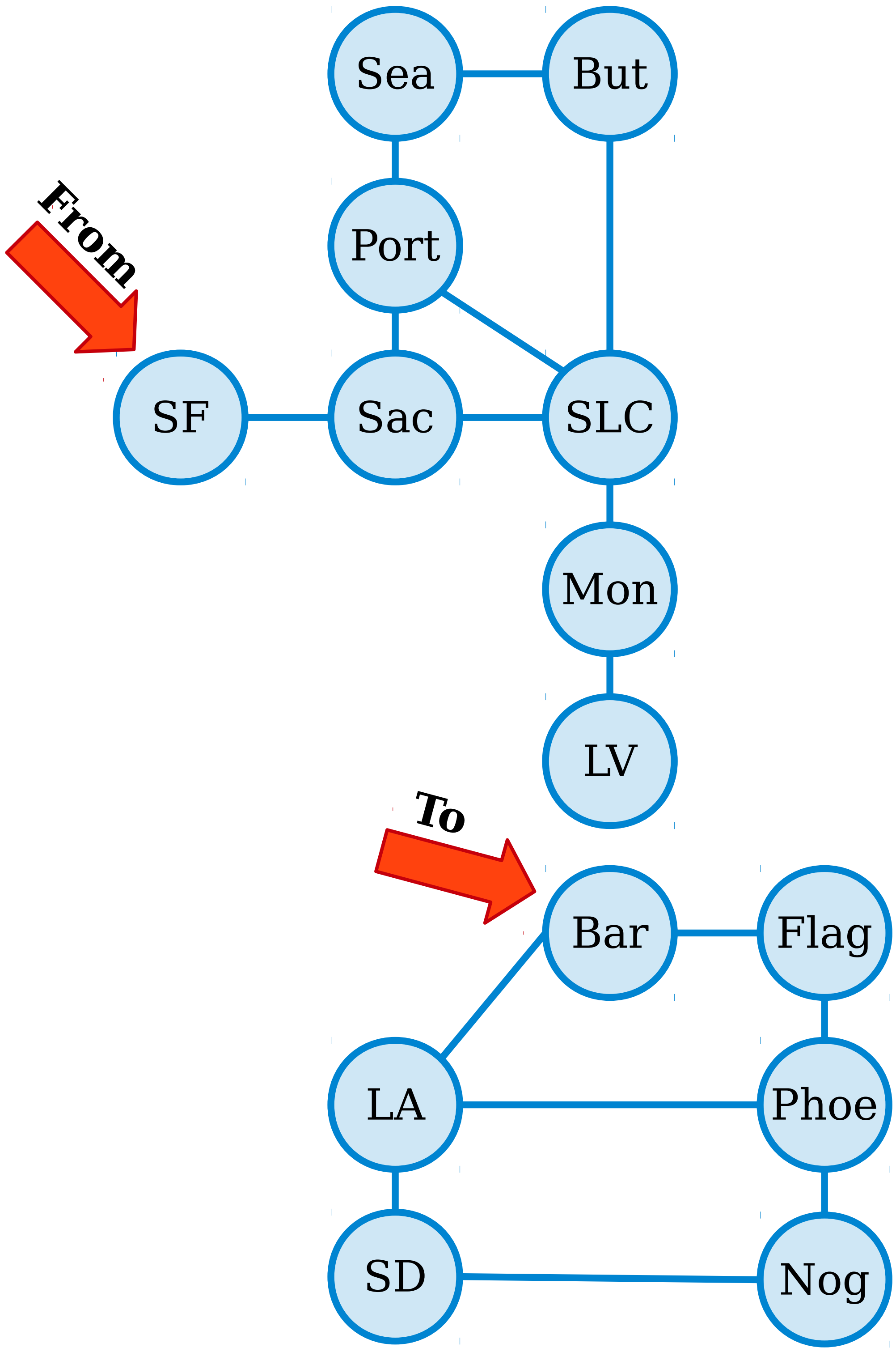
A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

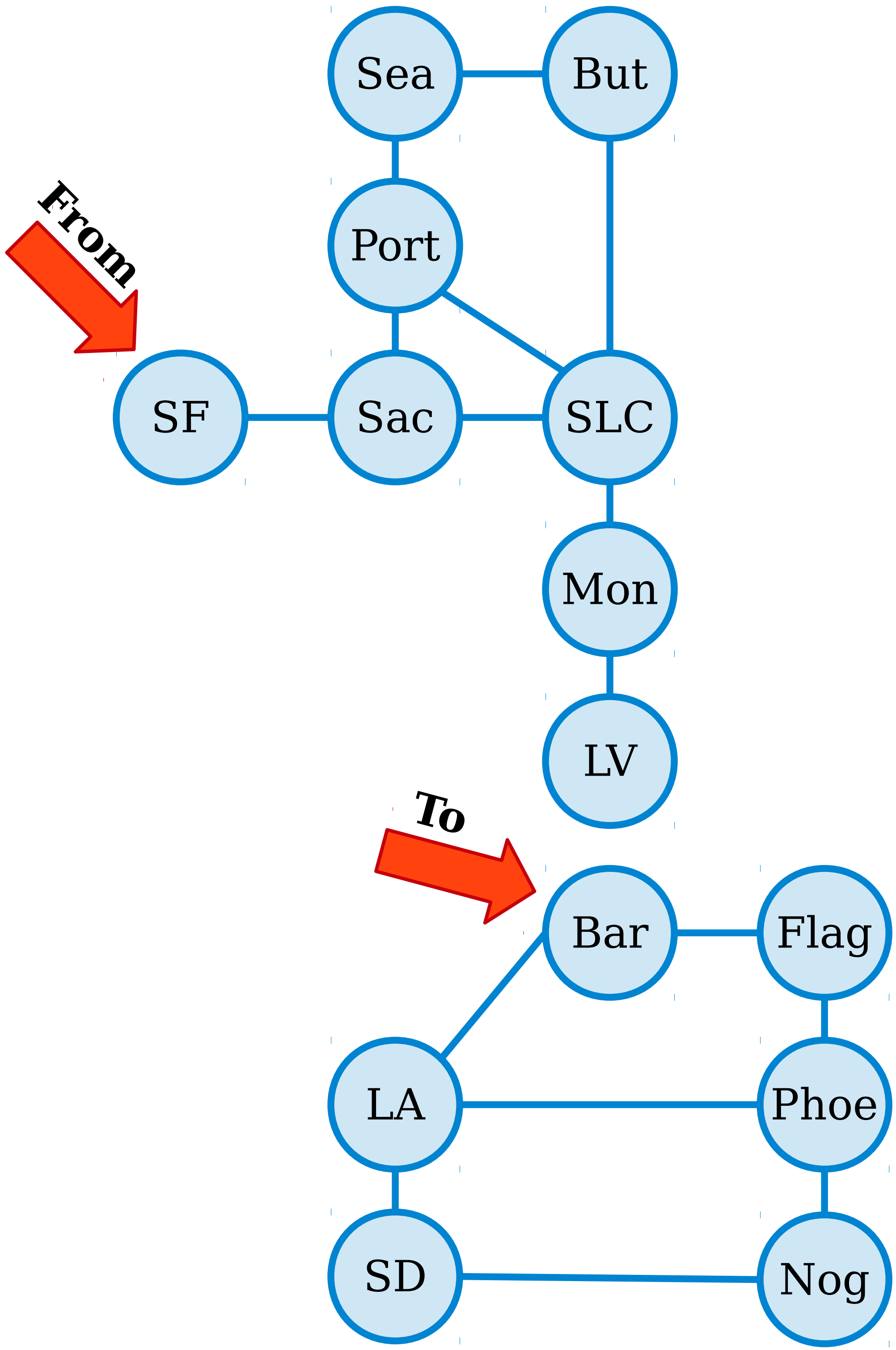


A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.



A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

Two nodes in a graph are called ***connected*** if there is a path between them

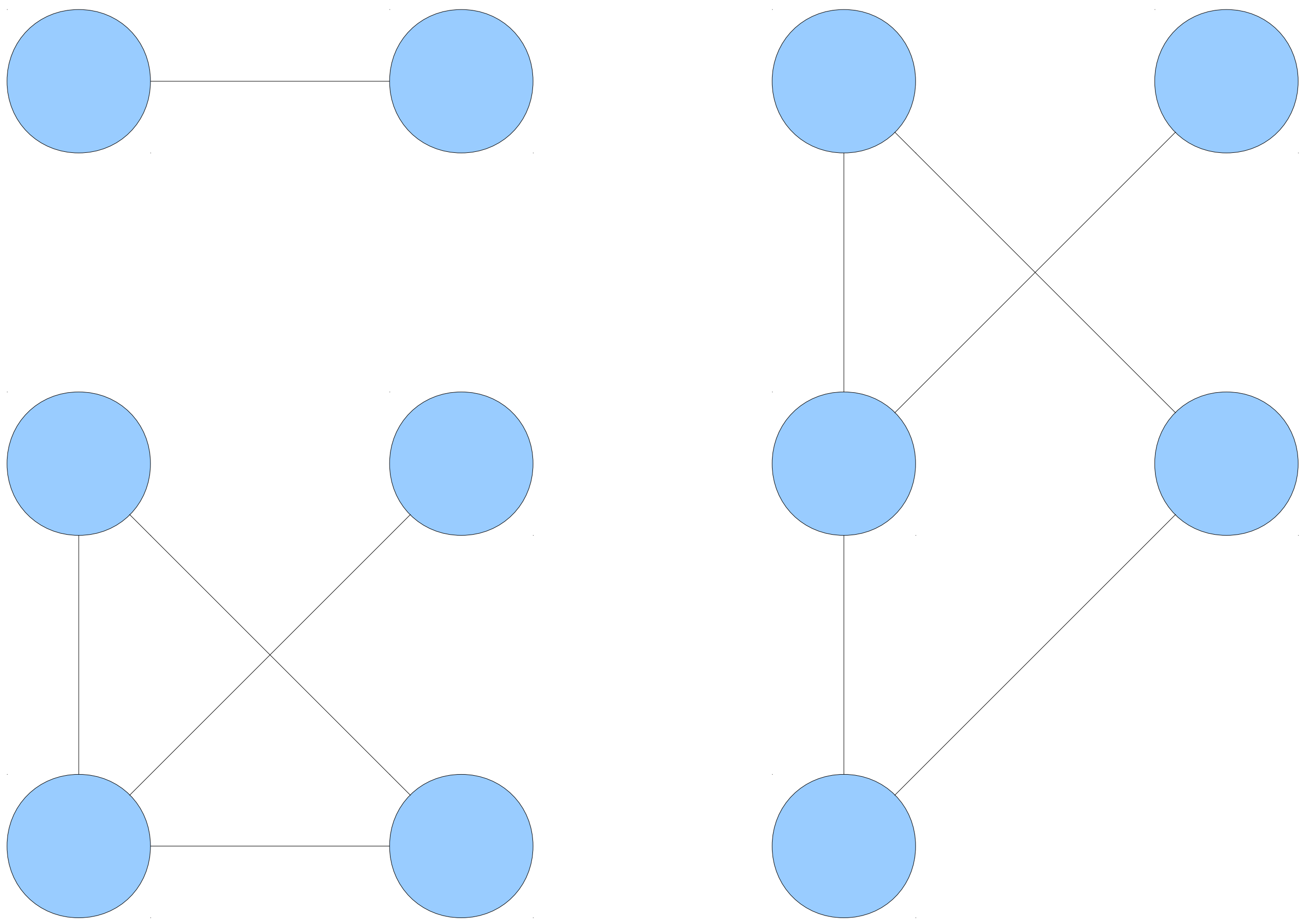


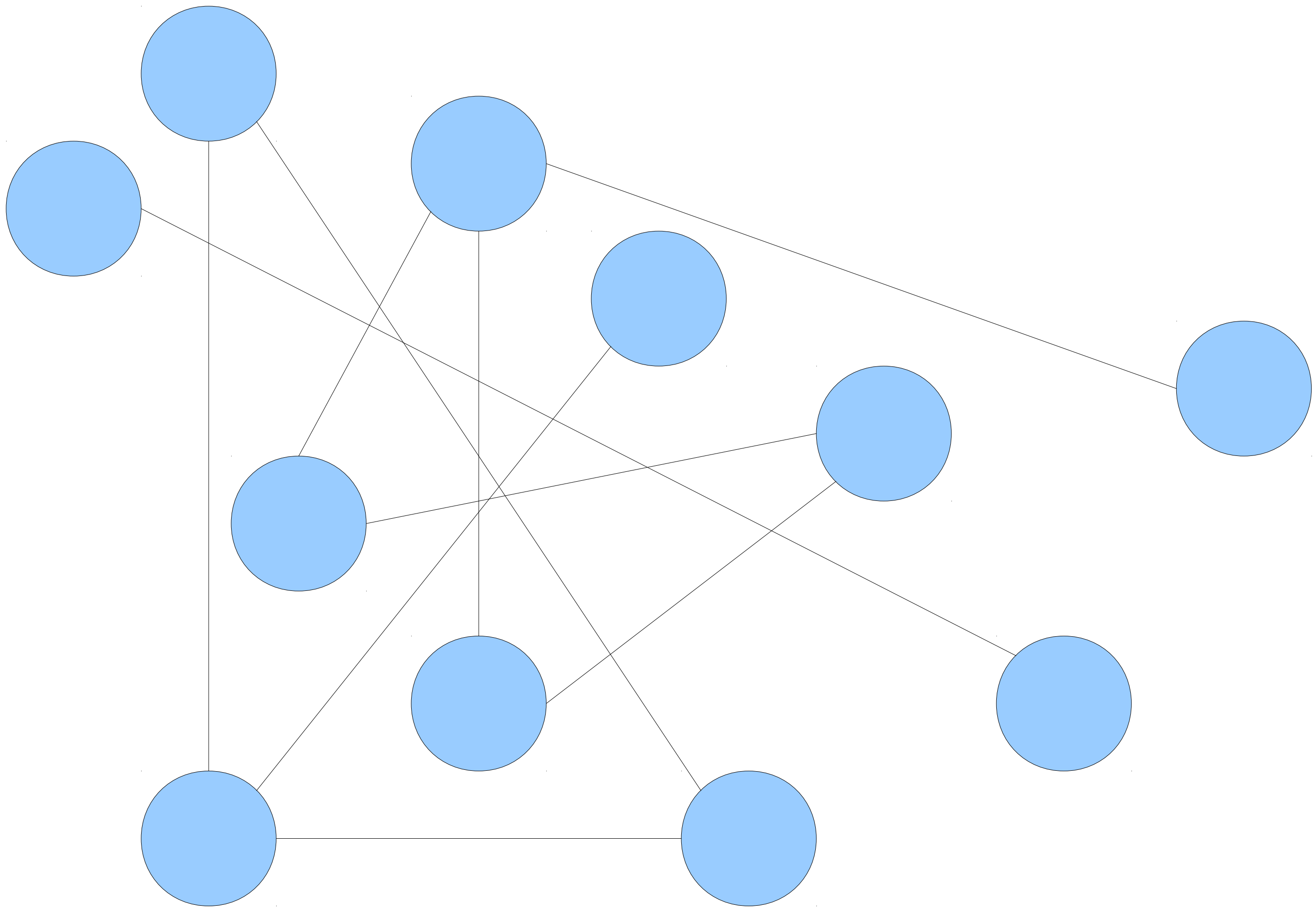
A ***path*** in a graph $G = (V, E)$ is a sequence of one or more nodes $v_1, v_2, v_3, \ldots, v_n$ such that any two consecutive nodes in the sequence are adjacent.

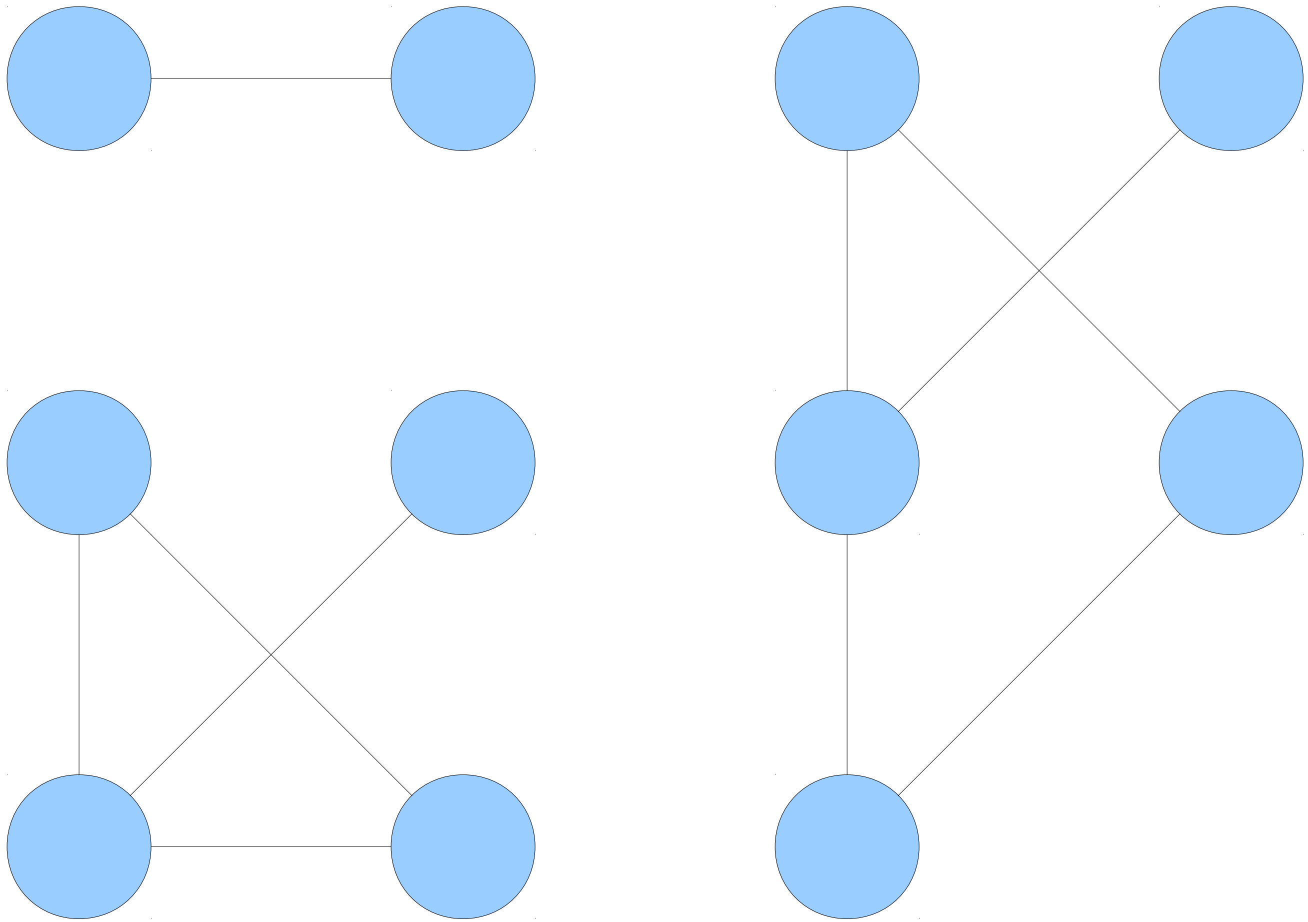
Two nodes in a graph are called ***connected*** if there is a path between them

A graph G as a whole is called ***connected*** if all pairs of nodes in G are connected.

Connected Components







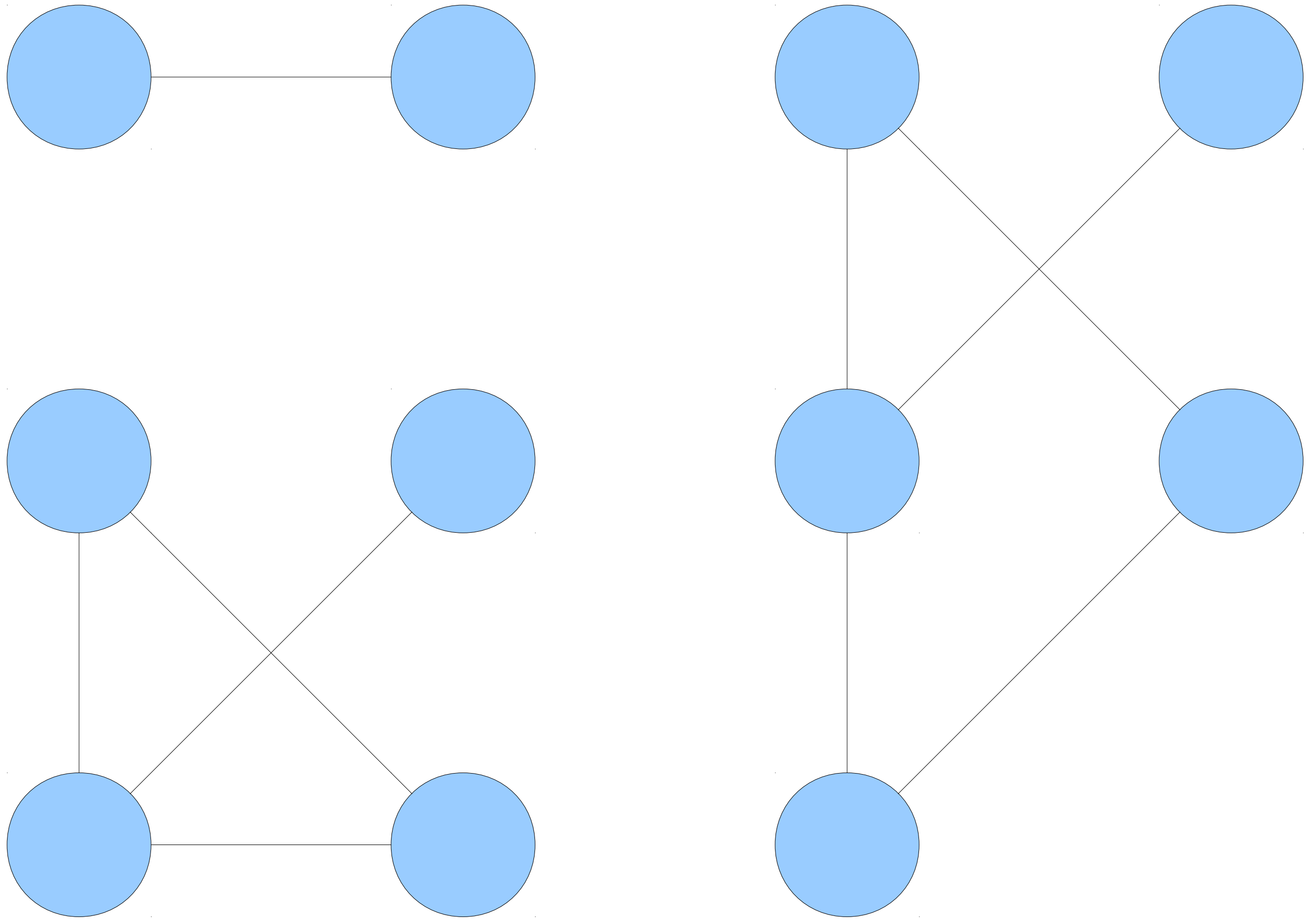
An Initial Definition

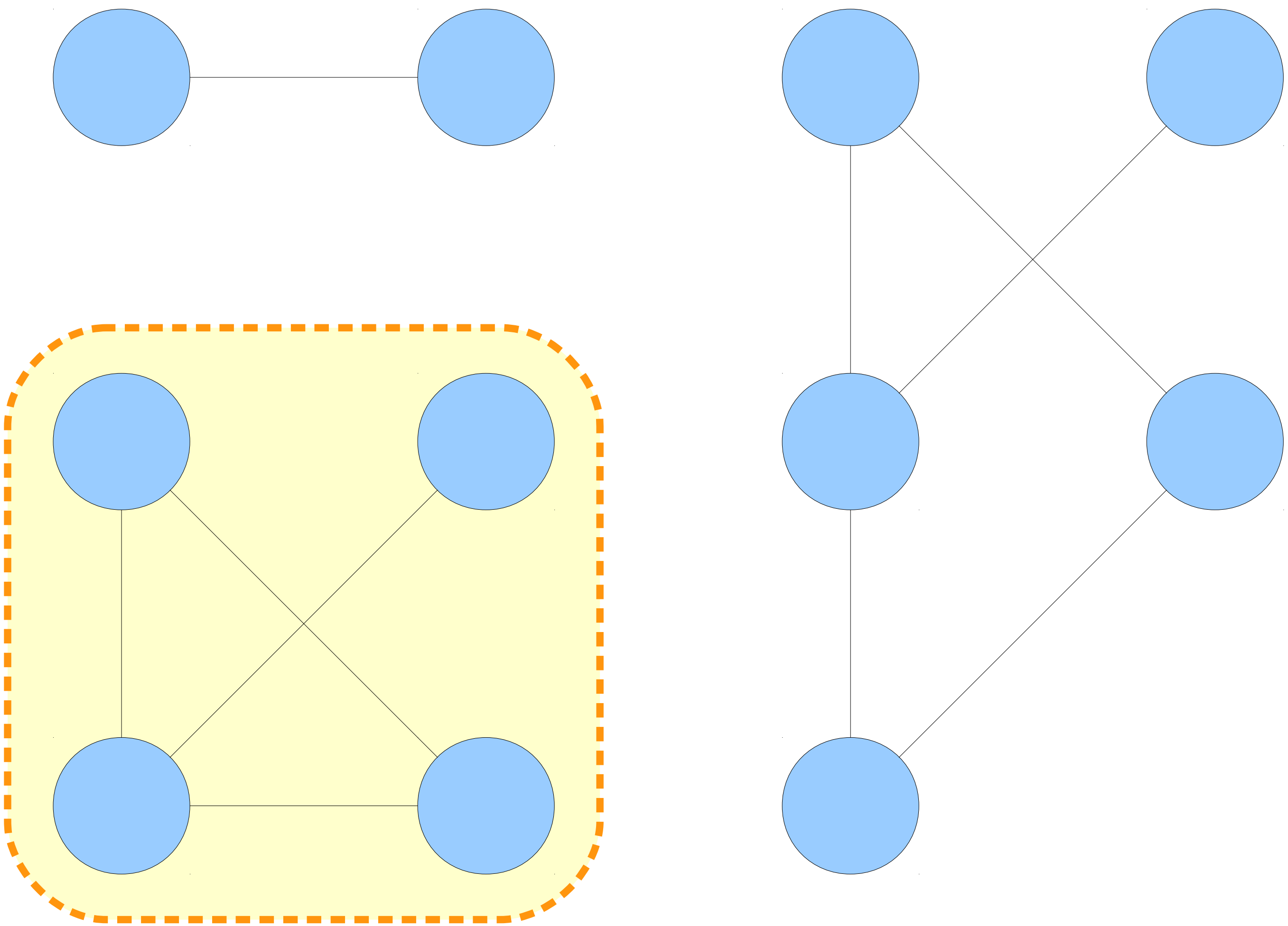
- **Attempted Definition #1:** A *piece* of an undirected graph $G = (V, E)$ is a set $C \subseteq V$ where

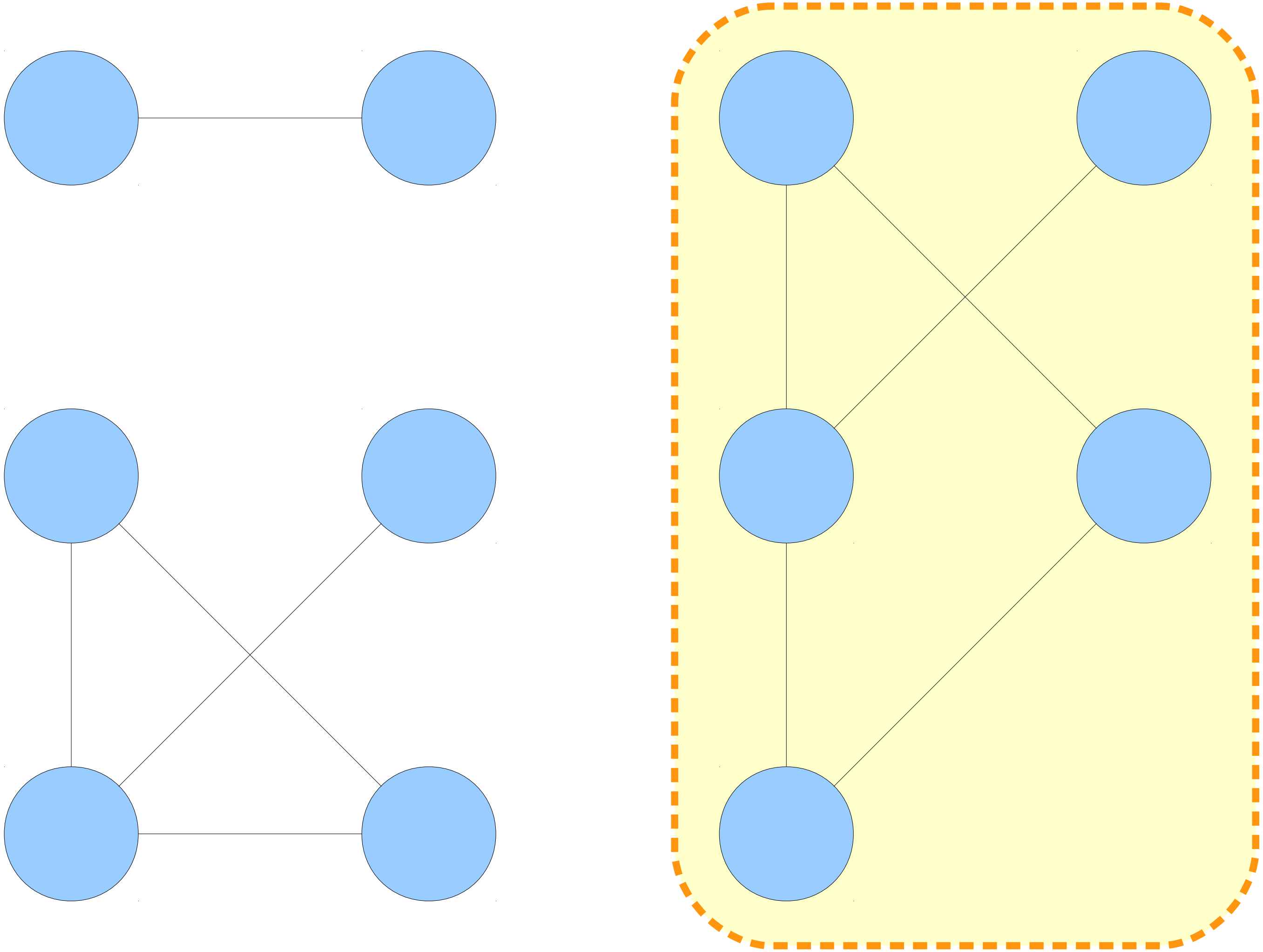
$$\forall u \in C.\ \forall v \in C.\ Connected(u, v)$$

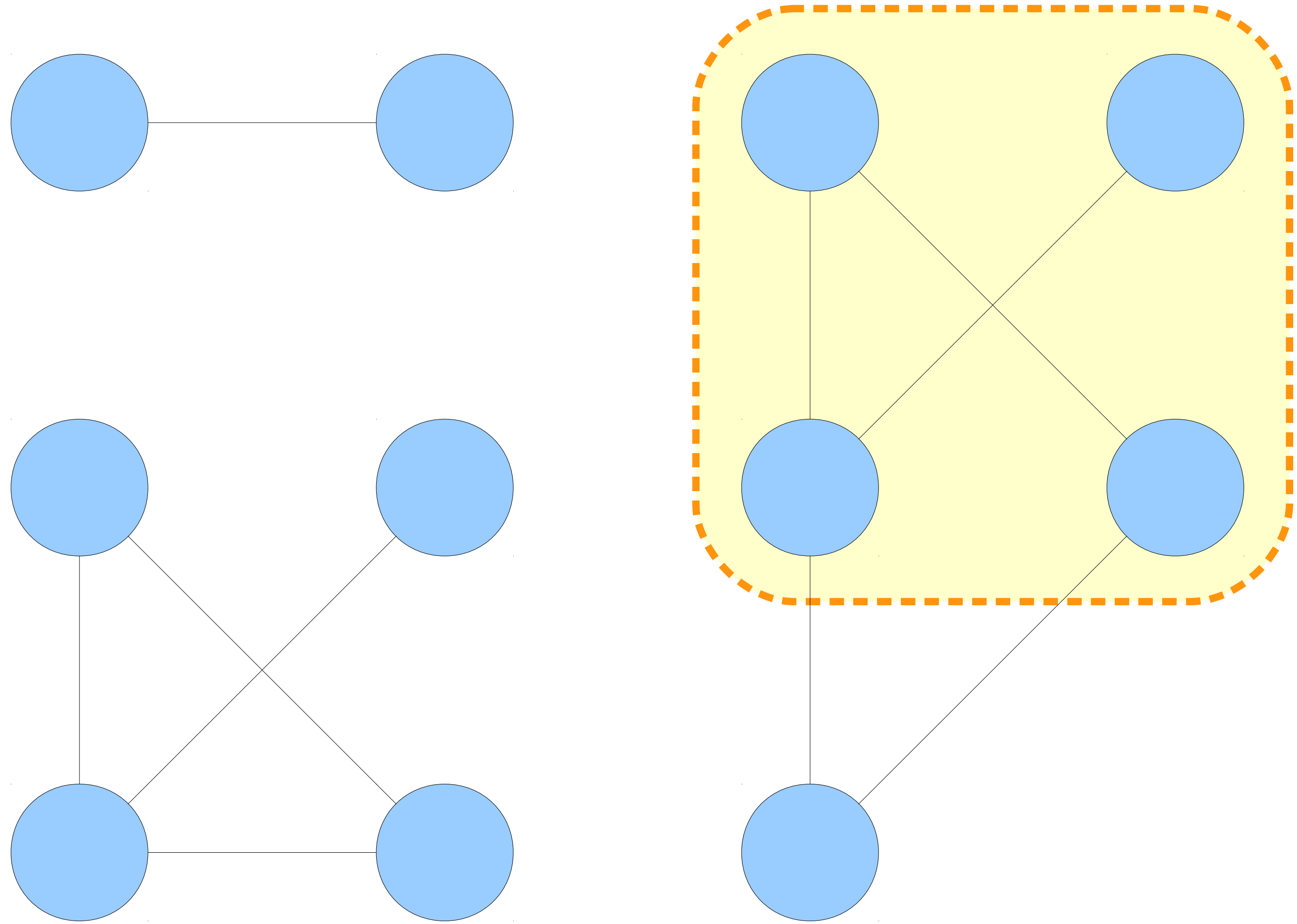
- Intuition: a piece of a graph is a set of nodes that are all connected to one another.

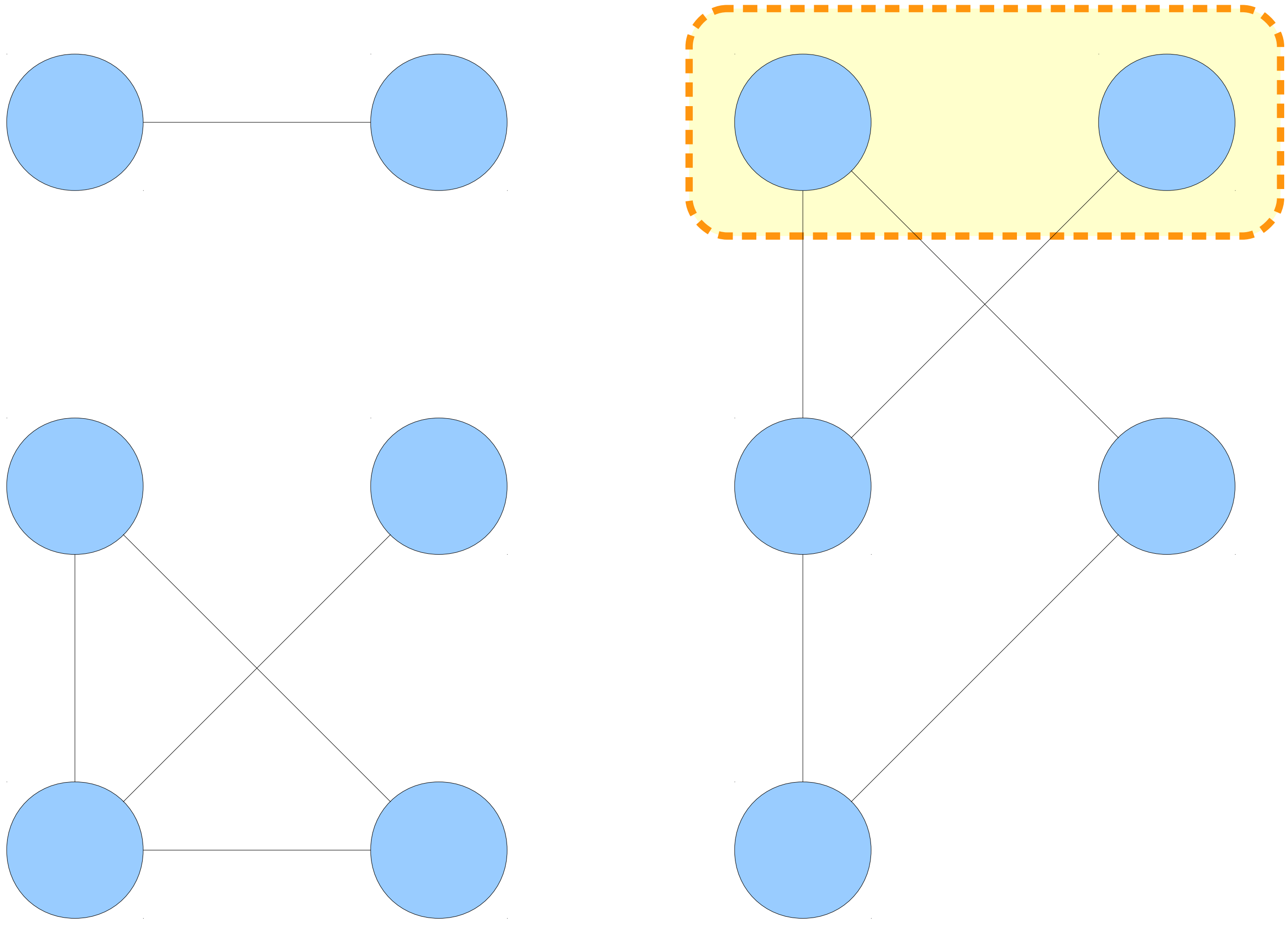
⚠ *This definition has some problems;* ⚠
please don't use it as a reference.

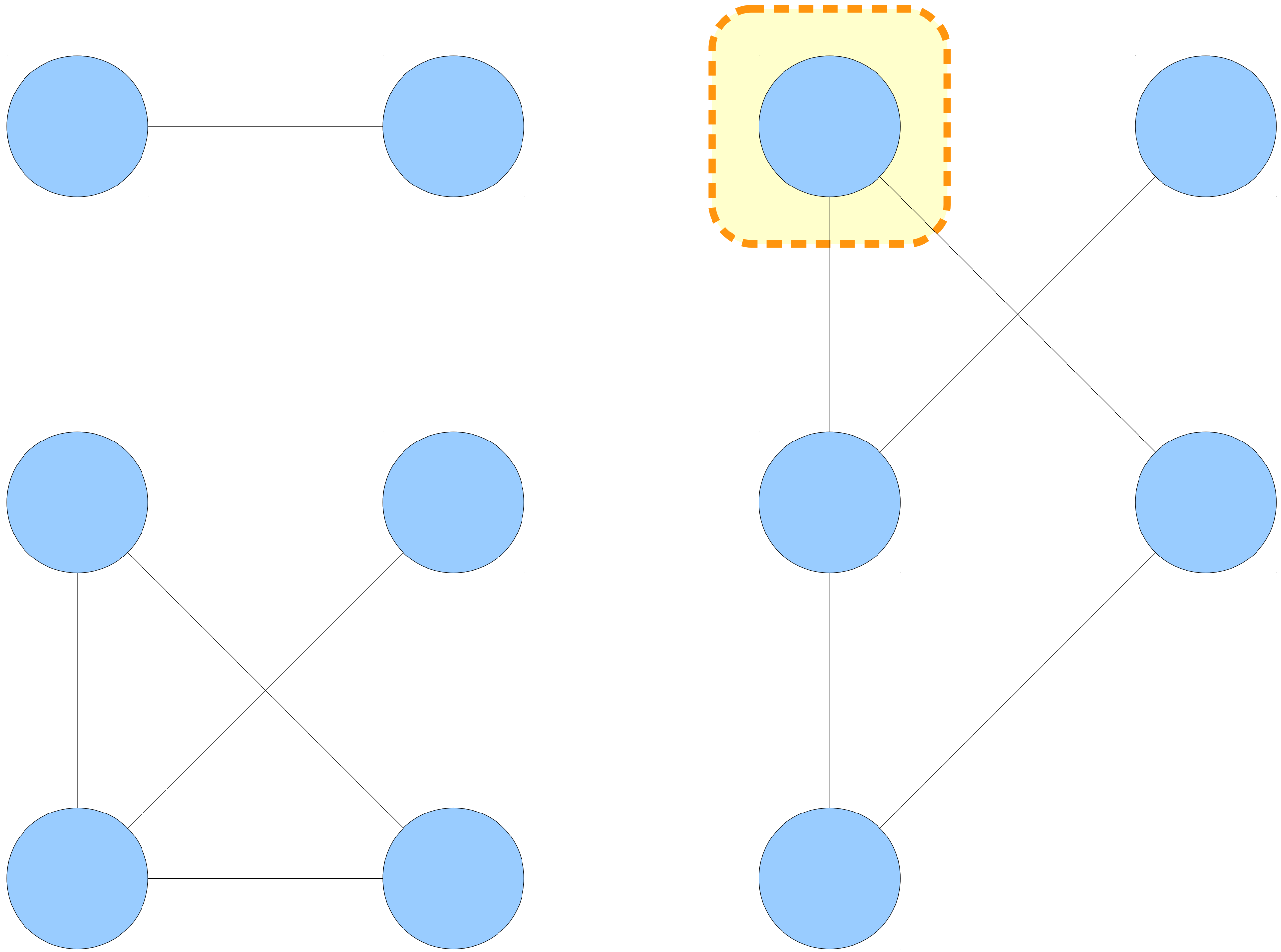








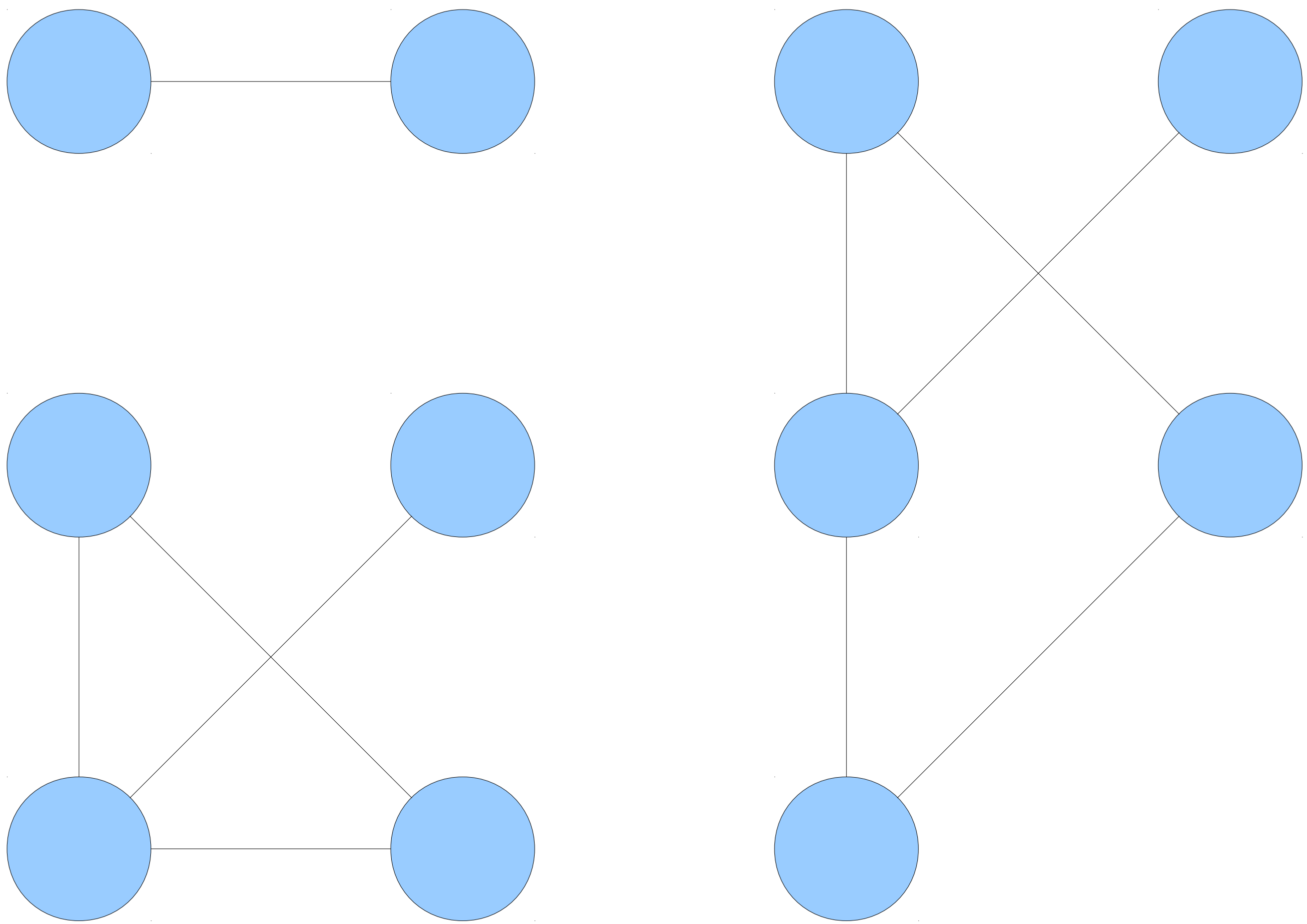


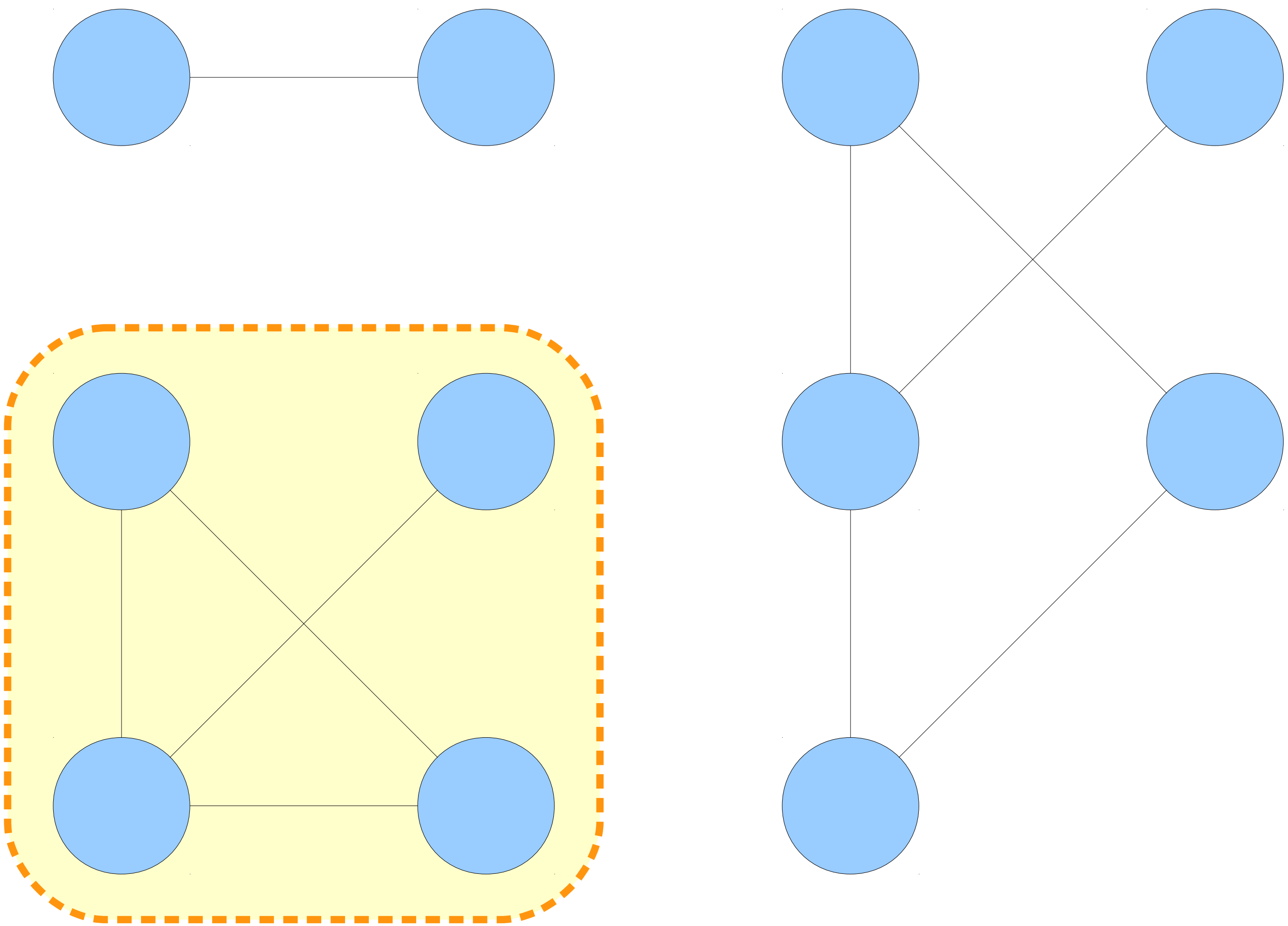


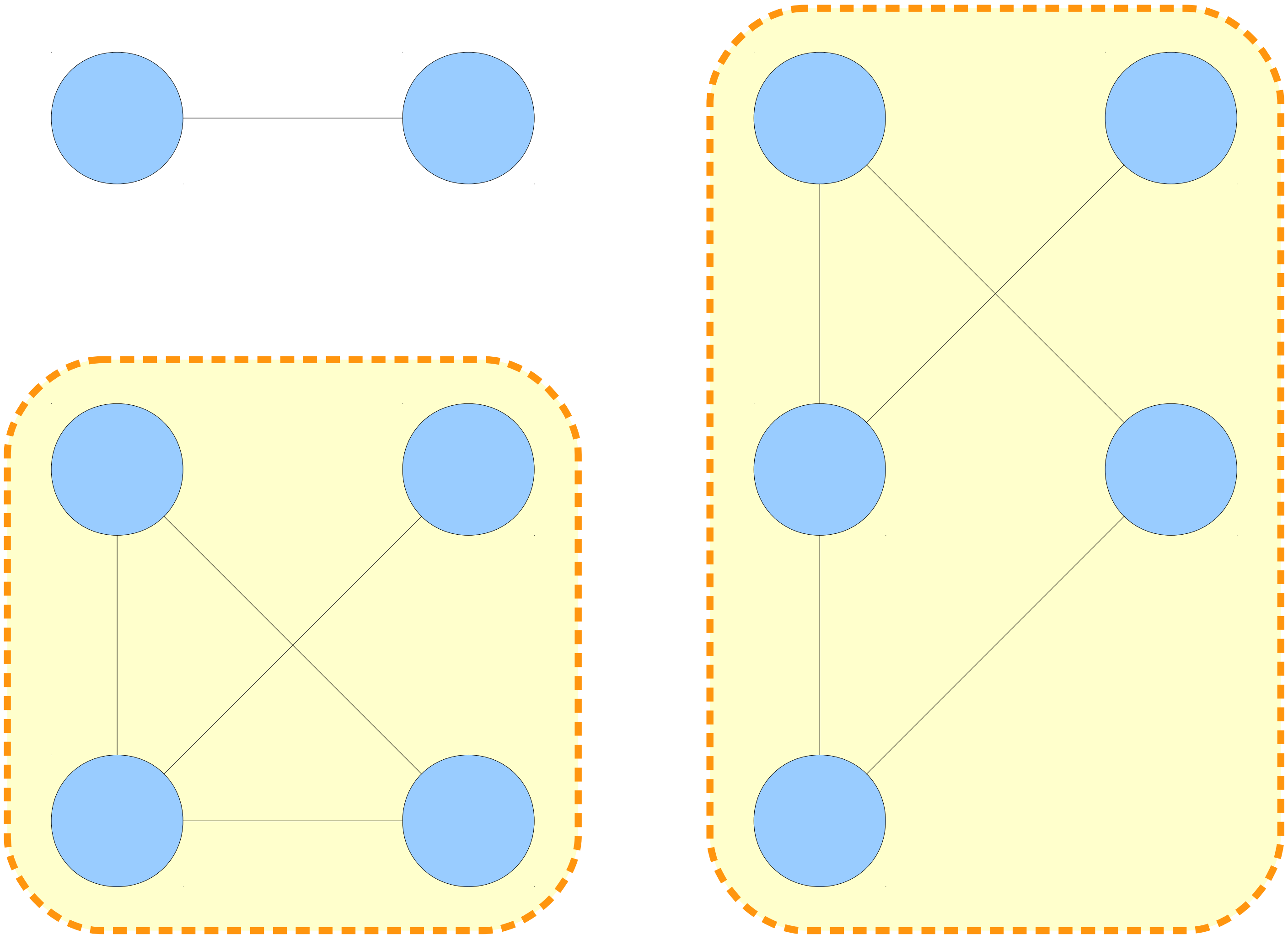
An Updated Definition

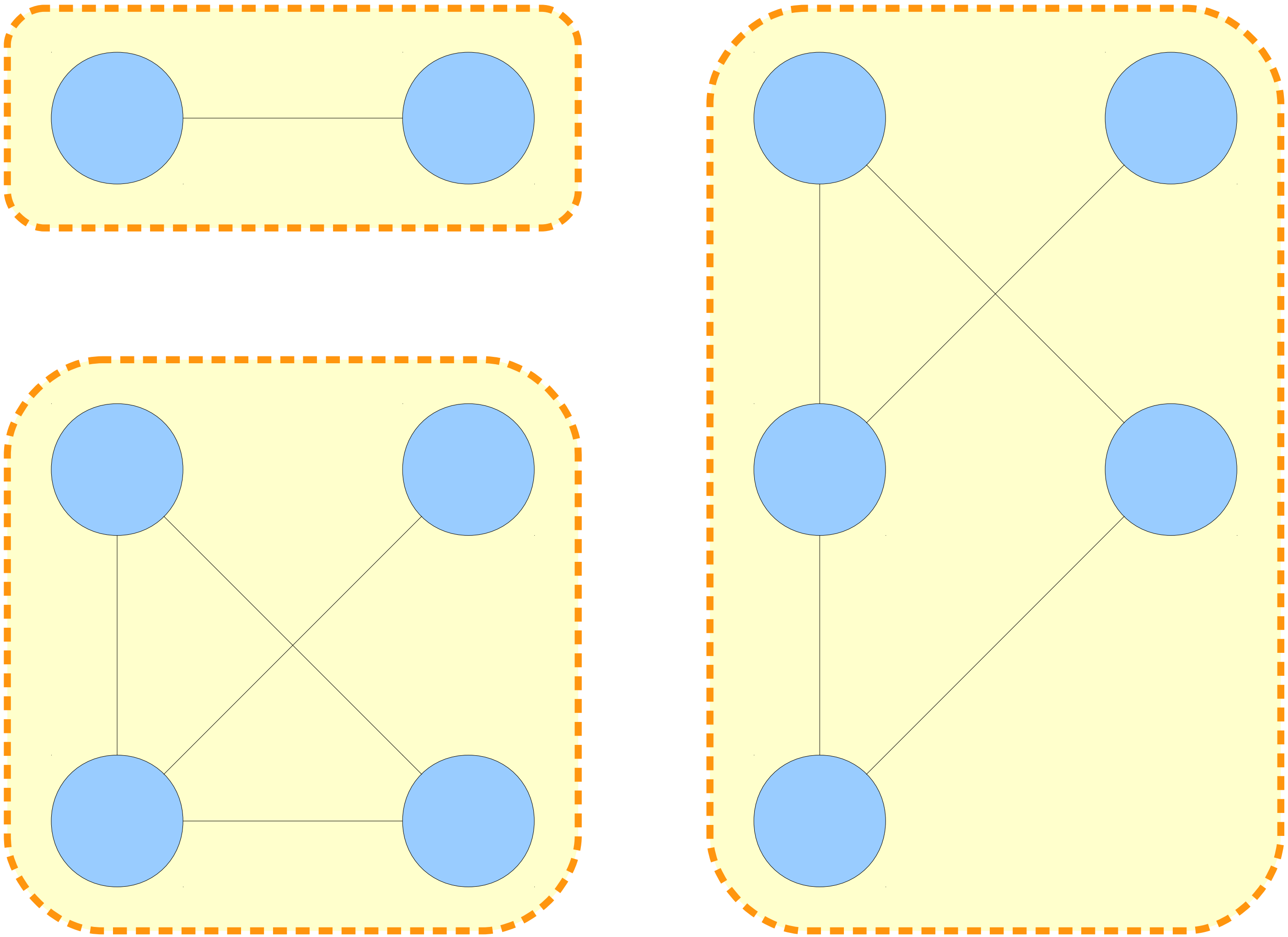
- **Attempted Definition #2:** A *piece* of an undirected graph $G = (V, E)$ is a set $C \subseteq V$ where
  - $\forall u \in C. \forall v \in C. Connected(u, v)$
  - $\forall u \in C. \forall v \in V - C. \neg Connected(u, v)$
- Intuition: a piece of a graph is a set of nodes that are all connected to one another that doesn't “miss” any nodes.

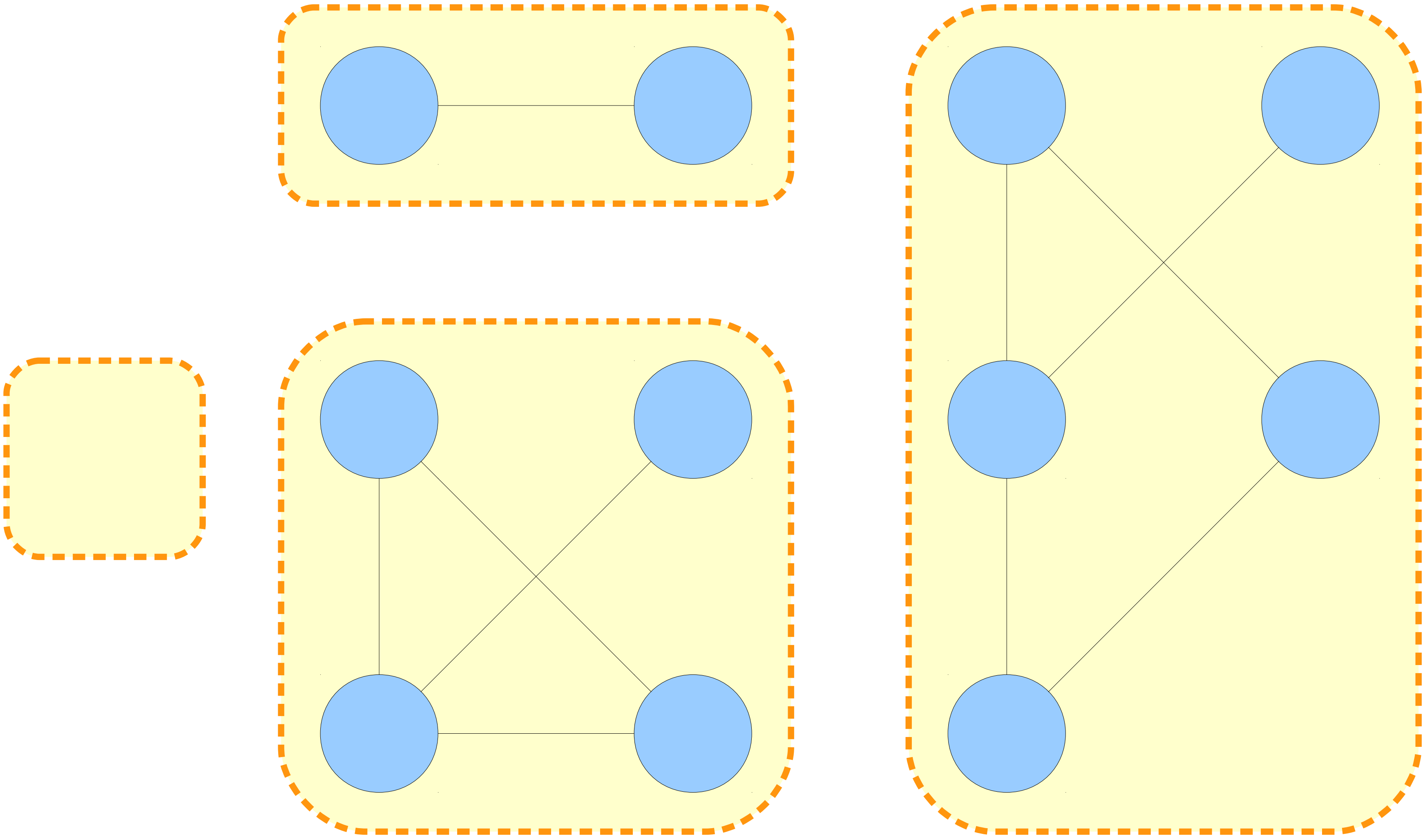
⚠ *This definition has some problems;* ⚠ *please don't use it as a reference.*

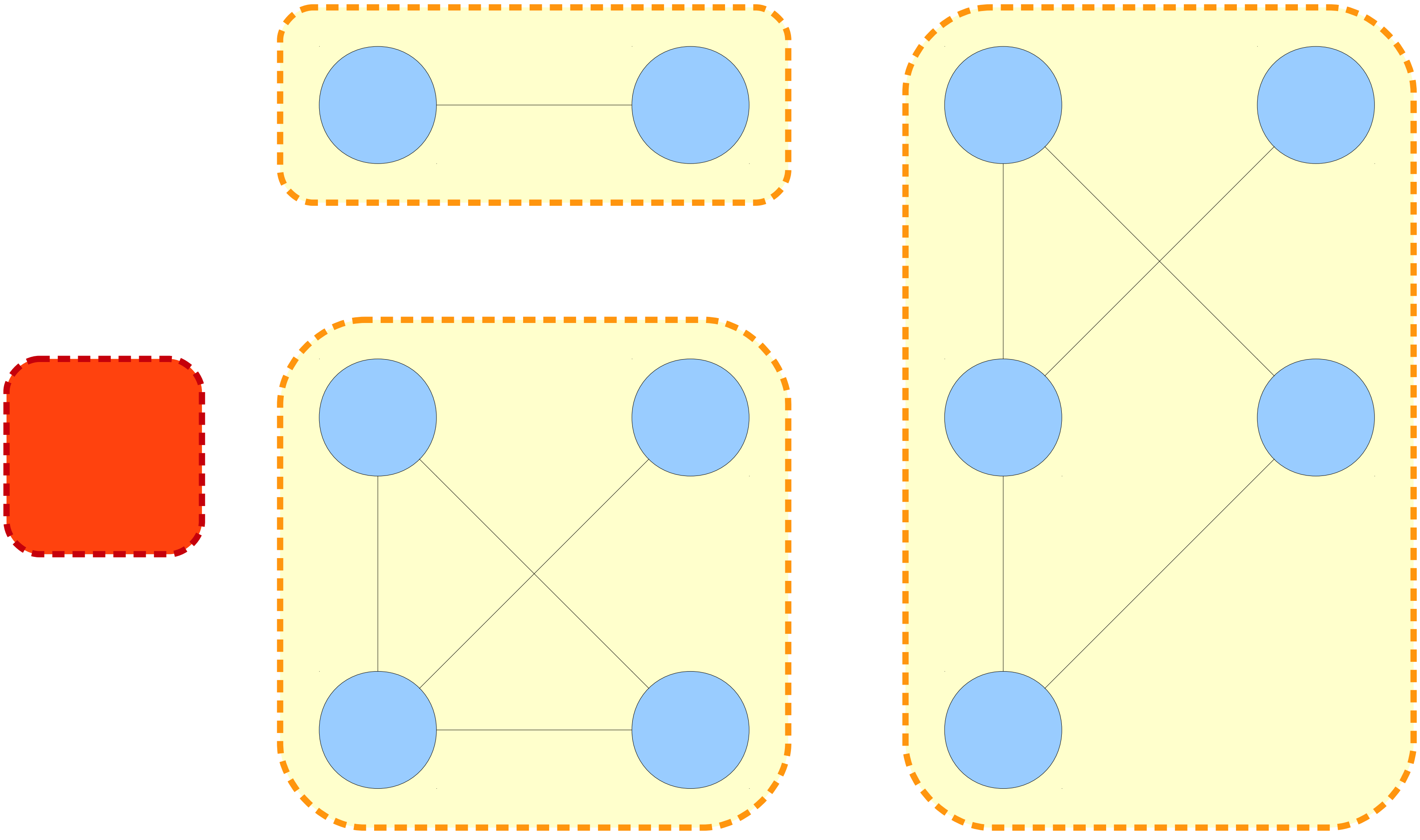












A Final Definition

- ***Definition:*** A ***connected component*** of an undirected graph $G = (V, E)$ is a *nonempty* set $C \subseteq V$ where
  - $\forall u \in C.\ \forall v \in C.\ Connected(u, v)$
  - $\forall u \in C.\ \forall v \in V - C.\ \neg Connected(u, v)$
- ***Theorem:*** Every node in a graph belongs to exactly one connected component.
- There is an analogous concept called a ***strongly connected component*** for directed graphs. To learn about them, take CS161!

Why This Matters

- The field of ***social and information network analysis*** studies human relations and interactions by modeling them as graphs.
- Pinning down definitions of terms like connected components lets us write computer programs that manipulate those definitions to discover properties of real social networks.
- Curious to learn more? Take CS224W!