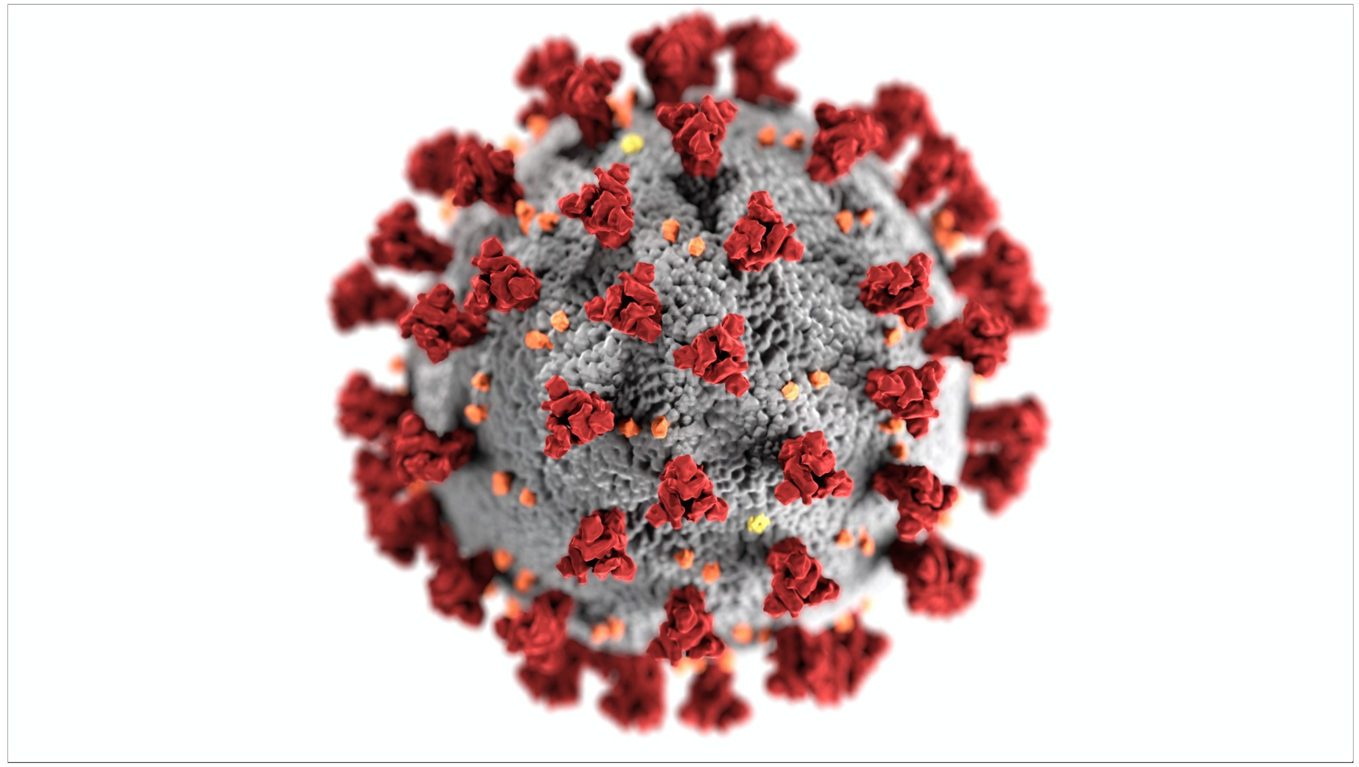
accu 2022

THROWING GRAPHS AT COVID

DOM DAVIS



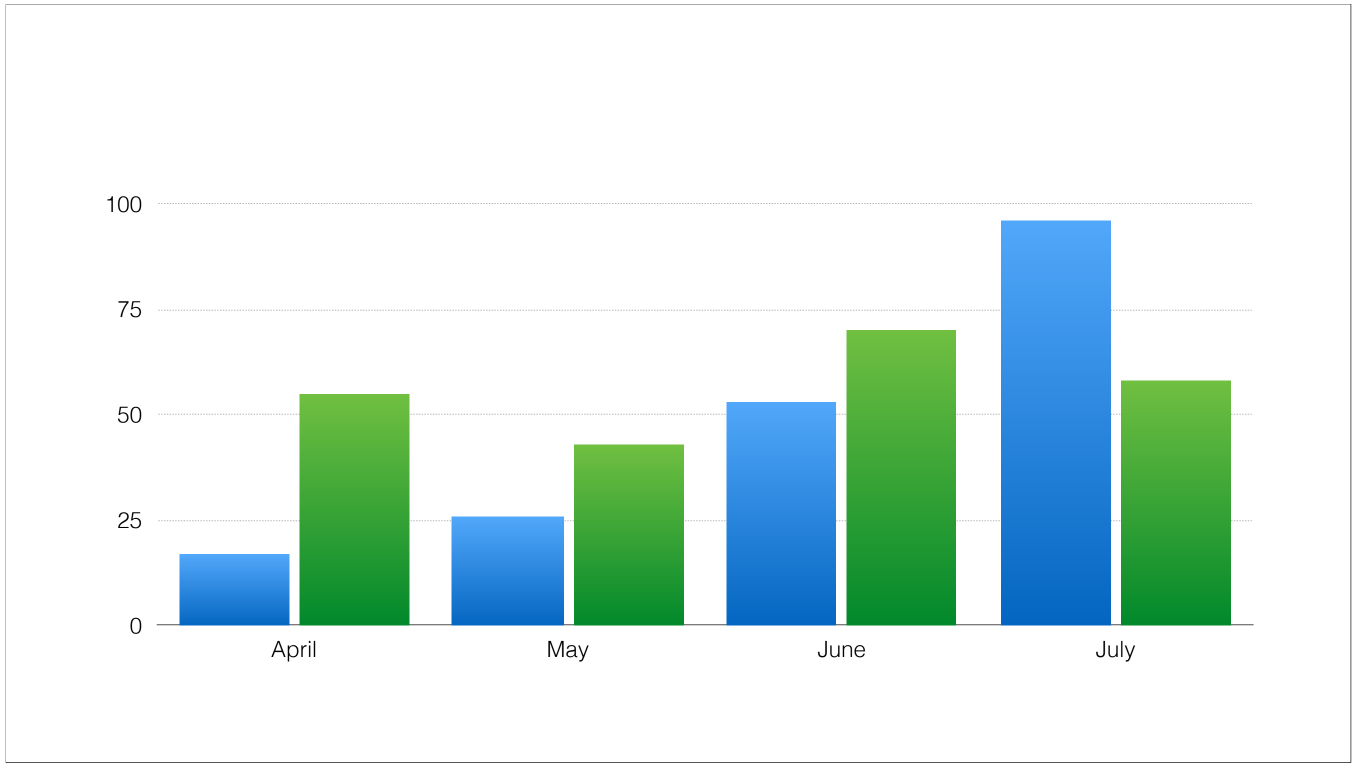
So I'm pretty sure that COVID will need no introduction. Although to be fair this talk will work with any communicable disease, plus a lot of other things that behave in a similar fashion, such as memes.

Dom Davis
@idomdavis
about.me/idomdavis

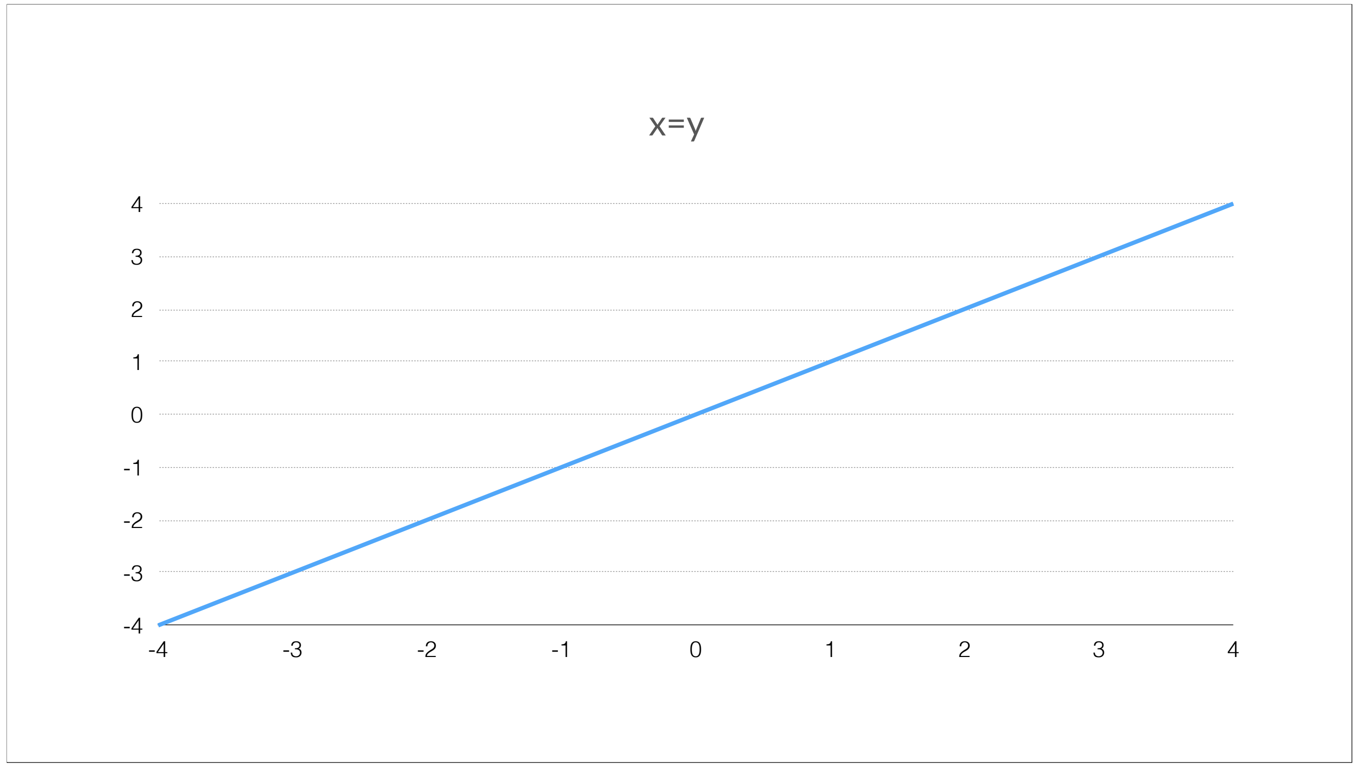
I probably need some introduction. My name is Dom Davis and I am a go developer who has been doing bad things to innocent graphs for a long time.

Dom Davis
@idomdavis
about.me/idomdavis

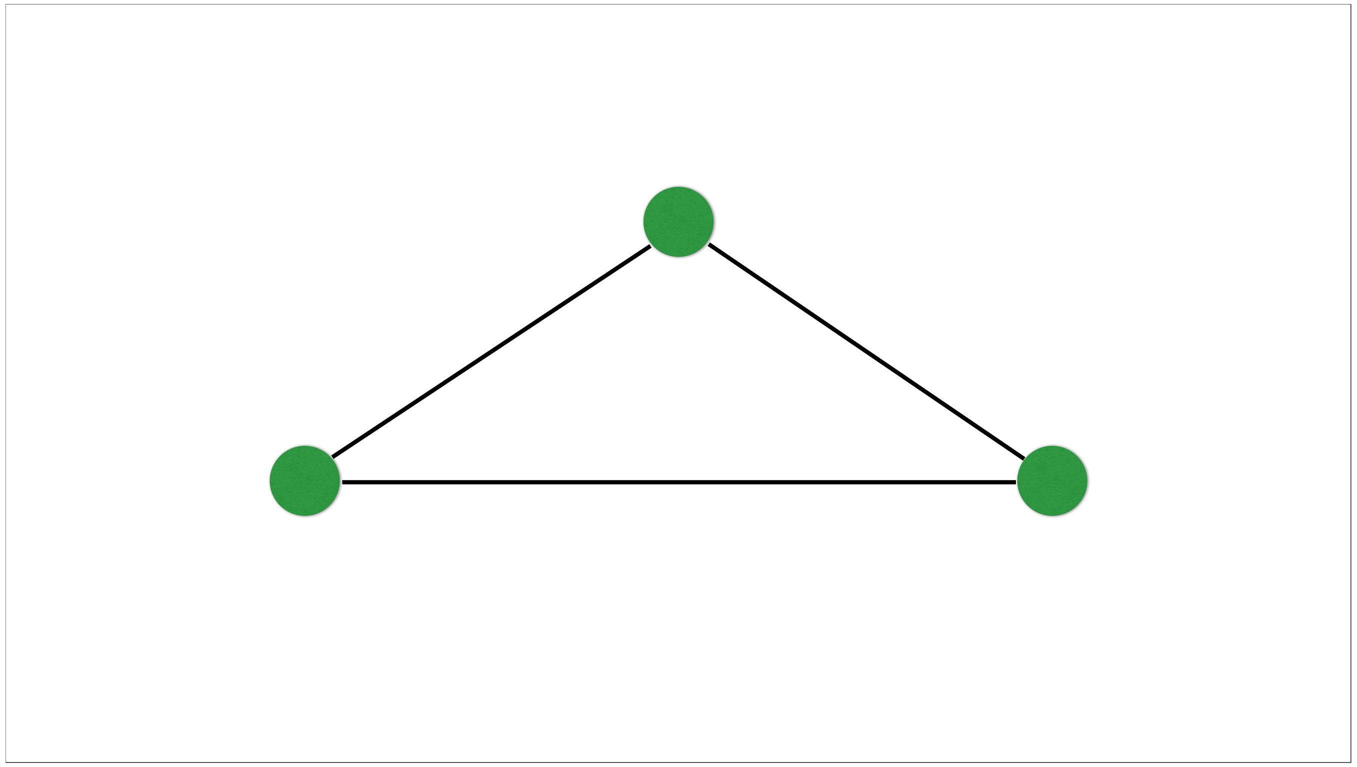
Despite having decades of experience, I always feel like an impostor at this conference because there are loads of super intelligent people who ask my technical questions I can't answer. Which is why I use Go. Go is a simple language, and I am a simple person.



Graphs almost certainly need an introduction. This is not a graph. This is a chart.



This is a graph of a function, but that's not the type of graph we're talking about.



What we're going to be talking about is the discrete maths definition of graphs. The confusion comes about because people routinely refer to charts as graphs...

USAGE

In its standard use **literally** means ‘in a literal sense, as opposed to a non-literal or exaggerated sense’, as for example in *I told him I never wanted to see him again, but I didn't expect him to take it **literally***. In recent years an extended use of **literally** (and also **literal**) has become very common, where **literally** (or **literal**) is used deliberately in non-literal contexts, for added effect, as in *they bought the car and **literally** ran it into the ground*. This use can lead to unintentional humorous effects (*we were **literally** killing ourselves laughing*) and is not acceptable in formal contexts, though it is widespread.

but given we're in a world where literally literally now means figuratively, this is not a hill I'm willing to die on.

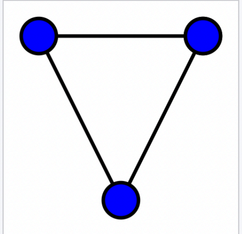
Graph [edit]

In one restricted but very common sense of the term,[1][2] a **graph** is an ordered pair $G = (V, E)$ comprising:

- V, a set of **vertices** (also called **nodes** or **points**);
- $E \subseteq \{\{x, y\} \mid x, y \in V \text{ and } x \neq y\}$, a set of **edges** (also called **links** or **lines**), which are unordered pairs of vertices (that is, an edge is associated with two distinct vertices).

To avoid ambiguity, this type of object may be called precisely an **undirected simple graph**.

In the edge $\{x, y\}$, the vertices x and y are called the **endpoints** of the edge. The edge is said to **join** x and y and to be **incident** on x and on y. A vertex may exist in a graph and not belong to an edge. **Multiple edges**, not allowed under the definition above, are two or more edges that join the same two vertices.



A graph with three vertices and three edges.

In one more general sense of the term allowing multiple edges,[3][4] a **graph** is an ordered triple $G = (V, E, \phi)$ comprising:

- V, a set of **vertices** (also called **nodes** or **points**);
- E, a set of **edges** (also called **links** or **lines**);
- $\phi : E \to \{\{x, y\} \mid x, y \in V \text{ and } x \neq y\}$, an **incidence function** mapping every edge to an unordered pair of vertices (that is, an edge is associated with two distinct vertices).

To avoid ambiguity, this type of object may be called precisely an **undirected multigraph**.

A **loop** is an edge that joins a vertex to itself. Graphs as defined in the two definitions above cannot have loops, because a loop joining a vertex x to itself is the edge (for an undirected simple graph) or is incident on (for an undirected multigraph) $\{x, x\} = \{x\}$ which is not in $\{\{x, y\} \mid x, y \in V \text{ and } x \neq y\}$. So to allow loops the definitions must be expanded. For undirected simple graphs, the definition of E should be modified to $E \subseteq \{\{x, y\} \mid x, y \in V\}$. For undirected multigraphs, the definition of ϕ should be modified to $\phi : E \to \{\{x, y\} \mid x, y \in V\}$. To avoid ambiguity, these types of objects may be called **undirected simple graph permitting loops** and **undirected multigraph permitting loops** (sometimes also **undirected pseudograph**), respectively.

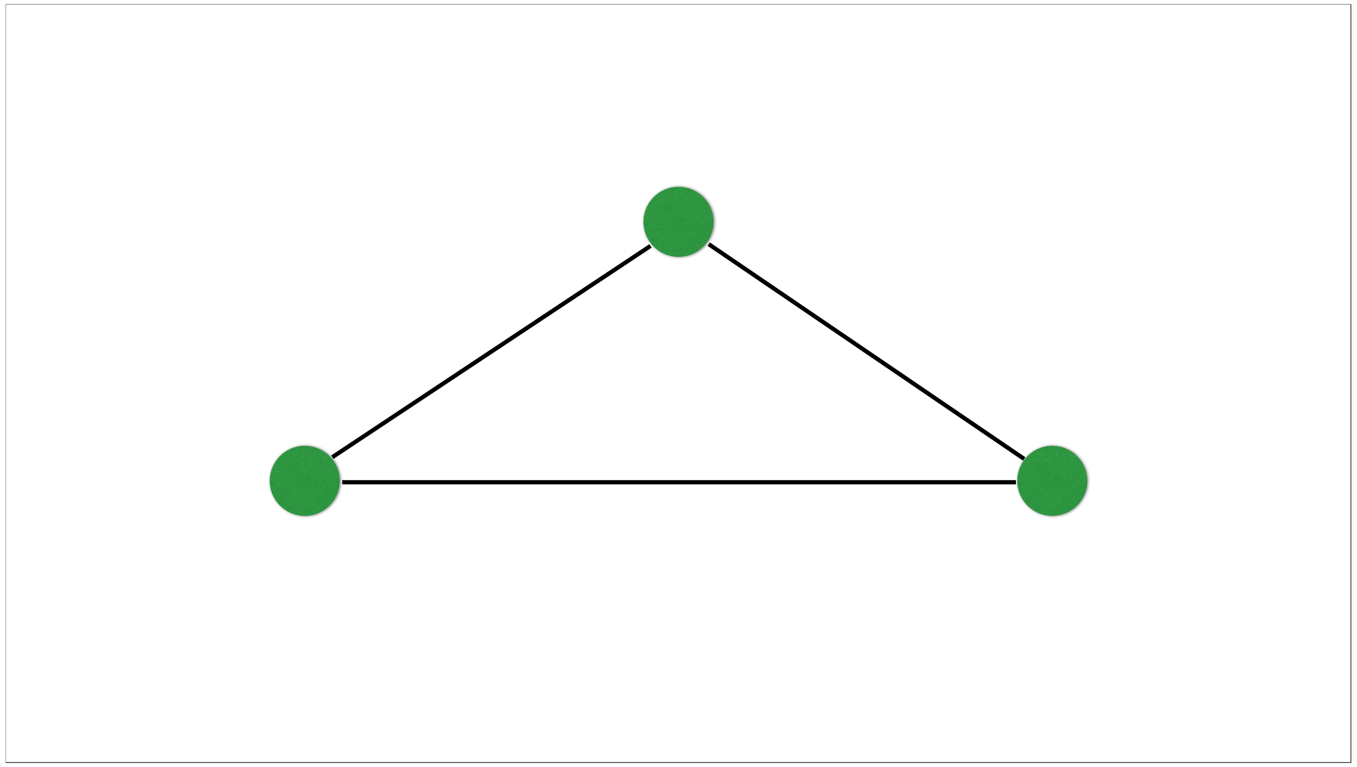
V and E are usually taken to be finite, and many of the well-known results are not true (or are rather different) for infinite graphs because many of the arguments fail in the infinite case. Moreover, V is often assumed to be non-empty, but E is allowed to be the empty set. The **order** of a graph is $|V|$, its number of vertices. The **size** of a graph is $|E|$, its number of edges. The **degree** or **valency** of a vertex is the number of edges that are incident to it, where a loop is counted twice. The **degree** of a graph is the maximum of the degrees of its vertices.

In an undirected simple graph of order *n*, the maximum degree of each vertex is *n* − 1 and the maximum size of the graph is *n*(*n* − 1)/2.

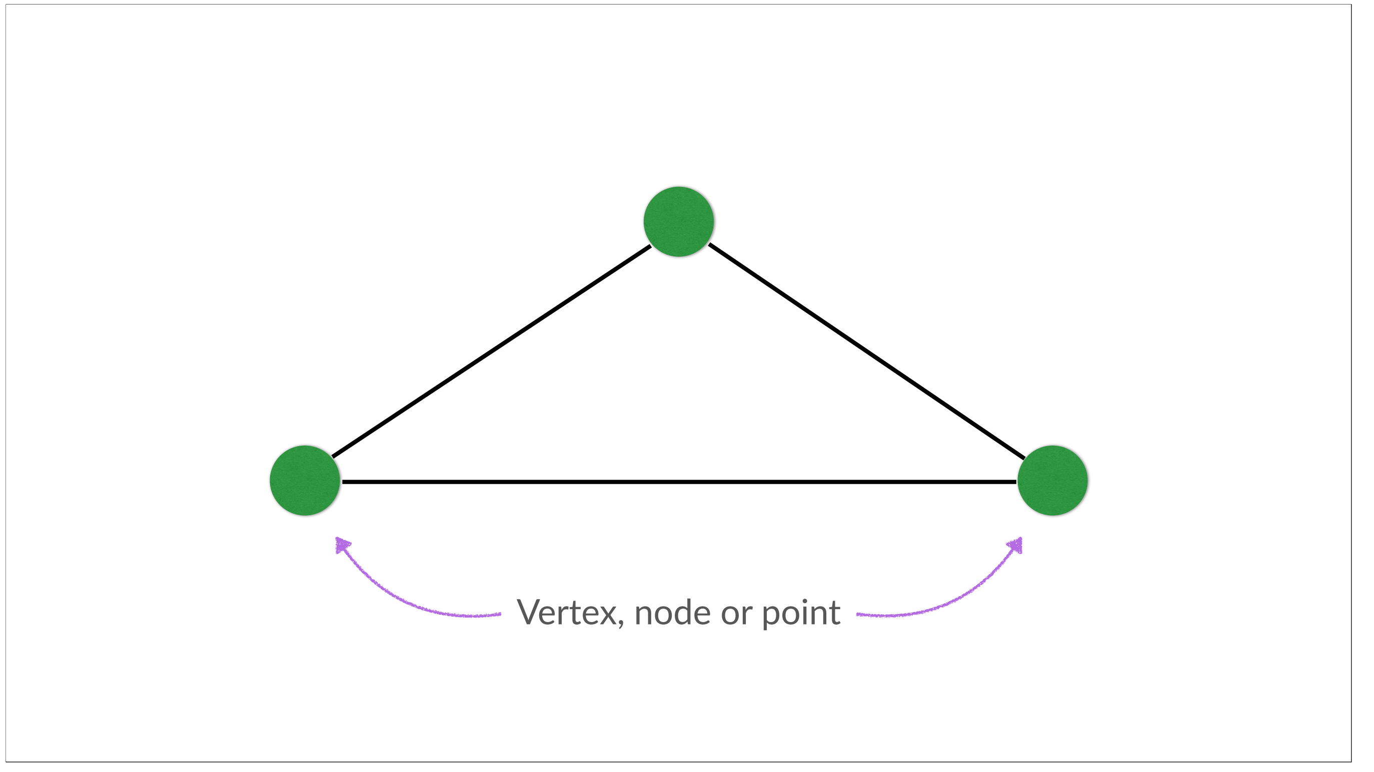
The edges of an undirected simple graph permitting loops G induce a symmetric homogeneous relation ~ on the vertices of G that is called the **adjacency relation** of G. Specifically, for each edge (x, y), its endpoints x and y are said to be **adjacent** to one another, which is denoted $x \sim y$.

https://en.wikipedia.org/wiki/Graph_theory

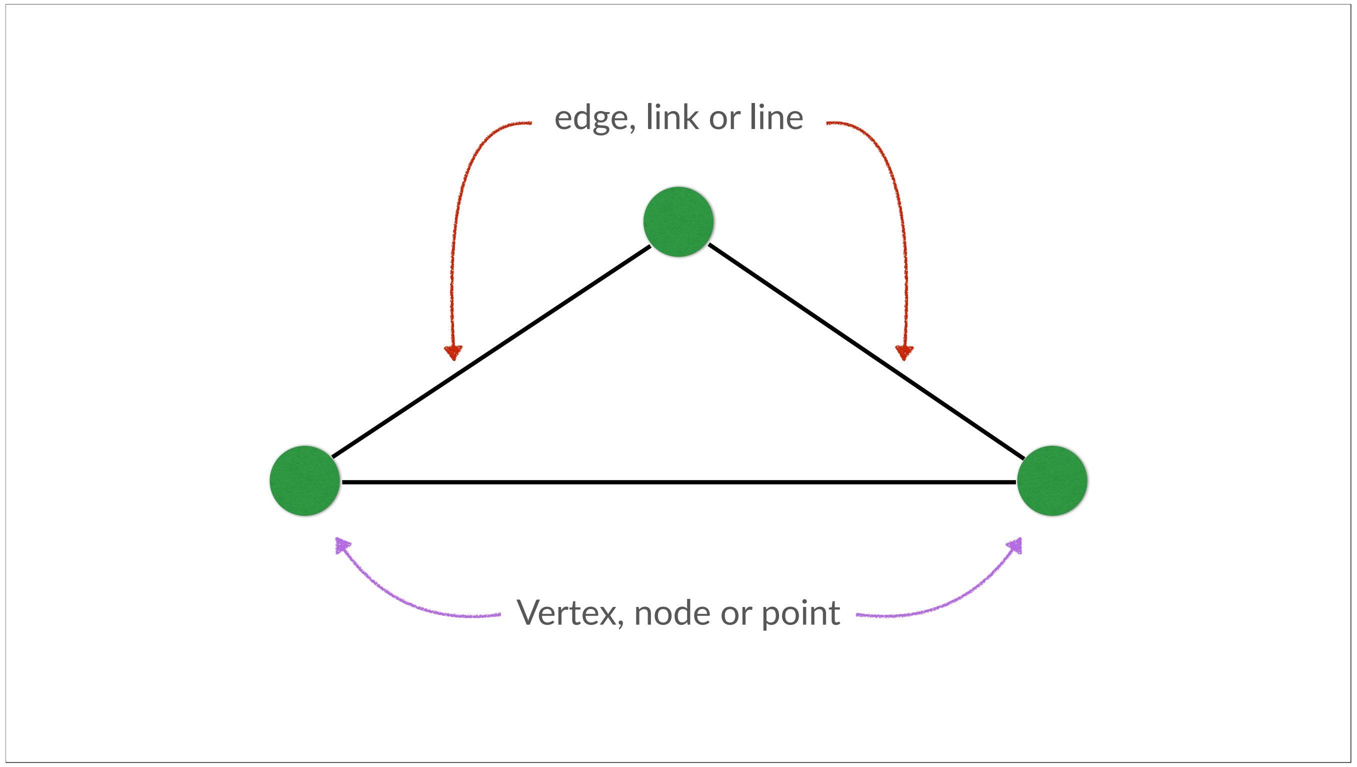
So wikipedia provides what it calls a "restrictive but common sense" definition. I... I mean yes, I guess, although I'm having to infer what half of that means.



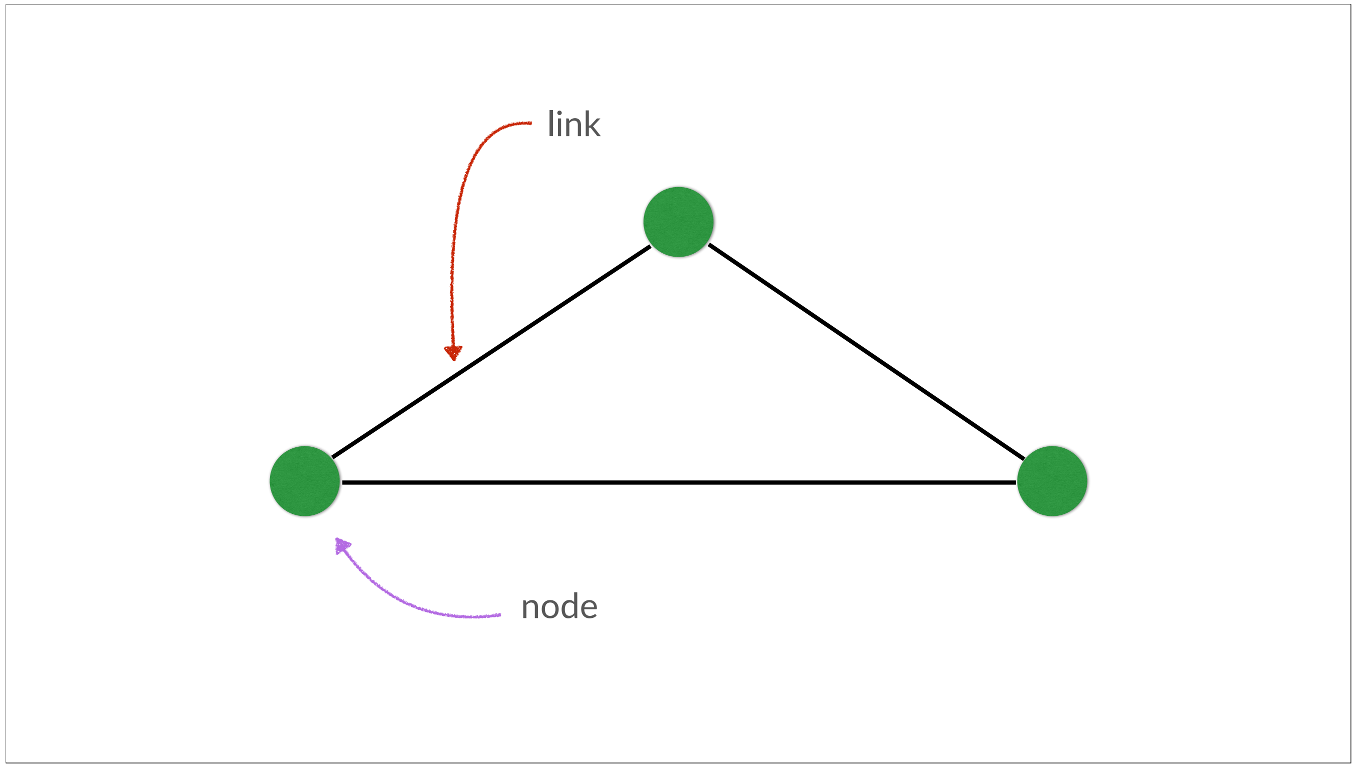
In this case a picture paints a thousand words, or hides a bunch of mathematical symbols.



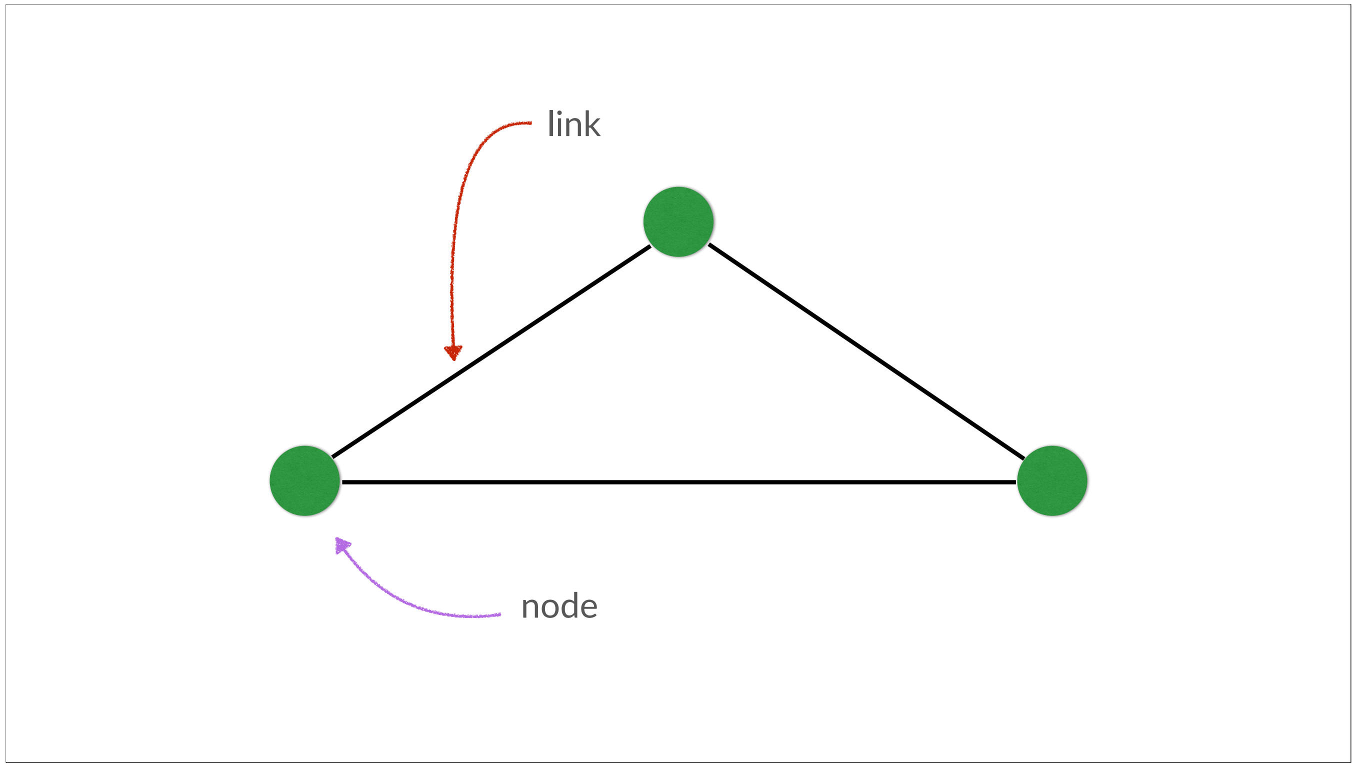
A graph is simply a set of things we call vertices, or nodes, or points



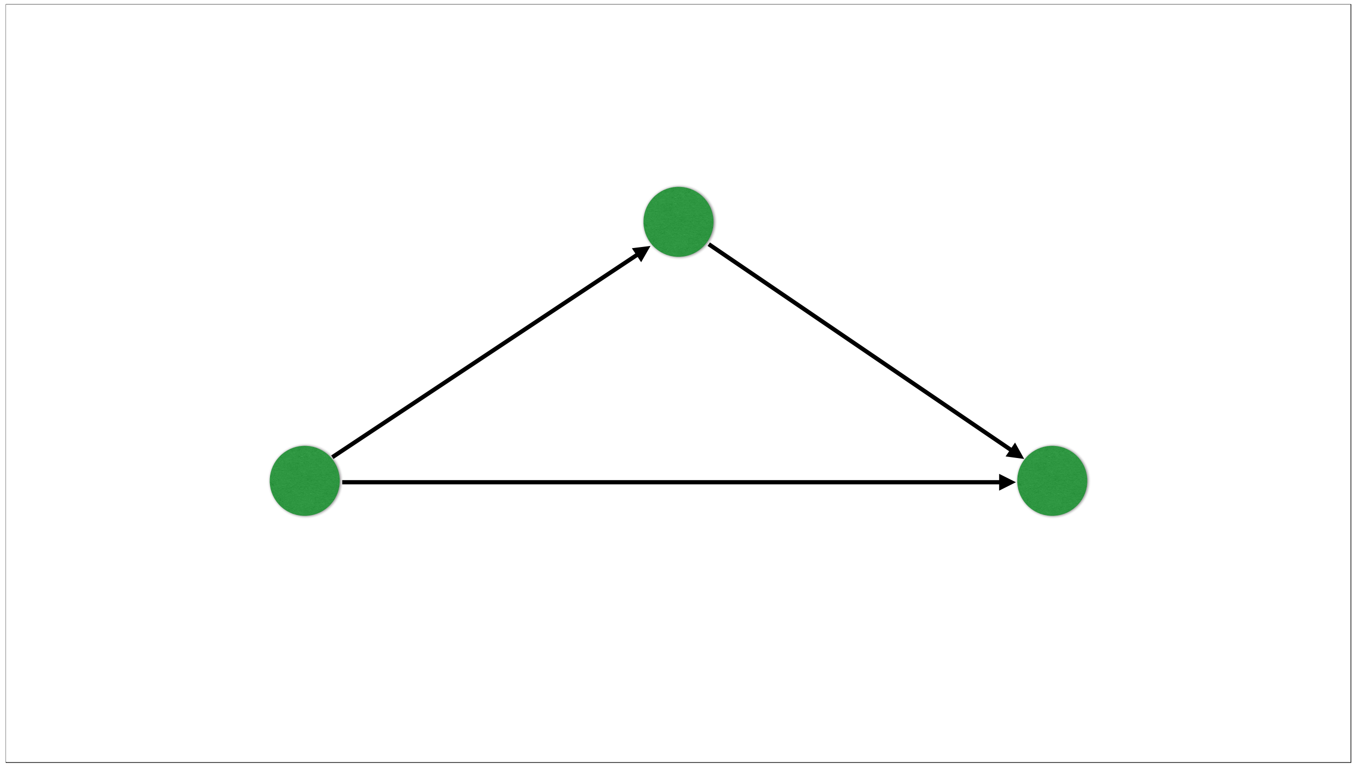
which can be joined with a set of things we call edges, links or lines.



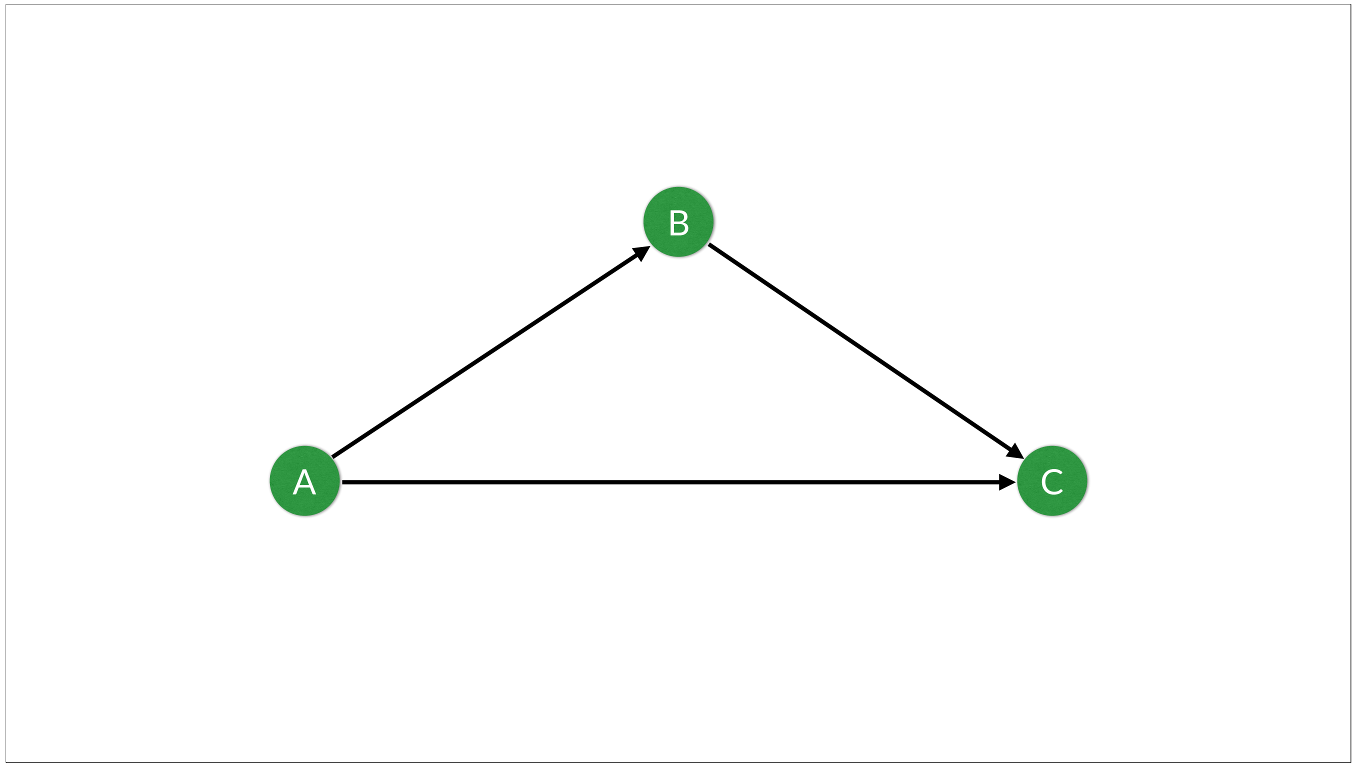
Initially I shall use the words nodes and links, although we'll change links later for reasons that will hopefully be obvious.



Typically nodes are circles, links are lines.



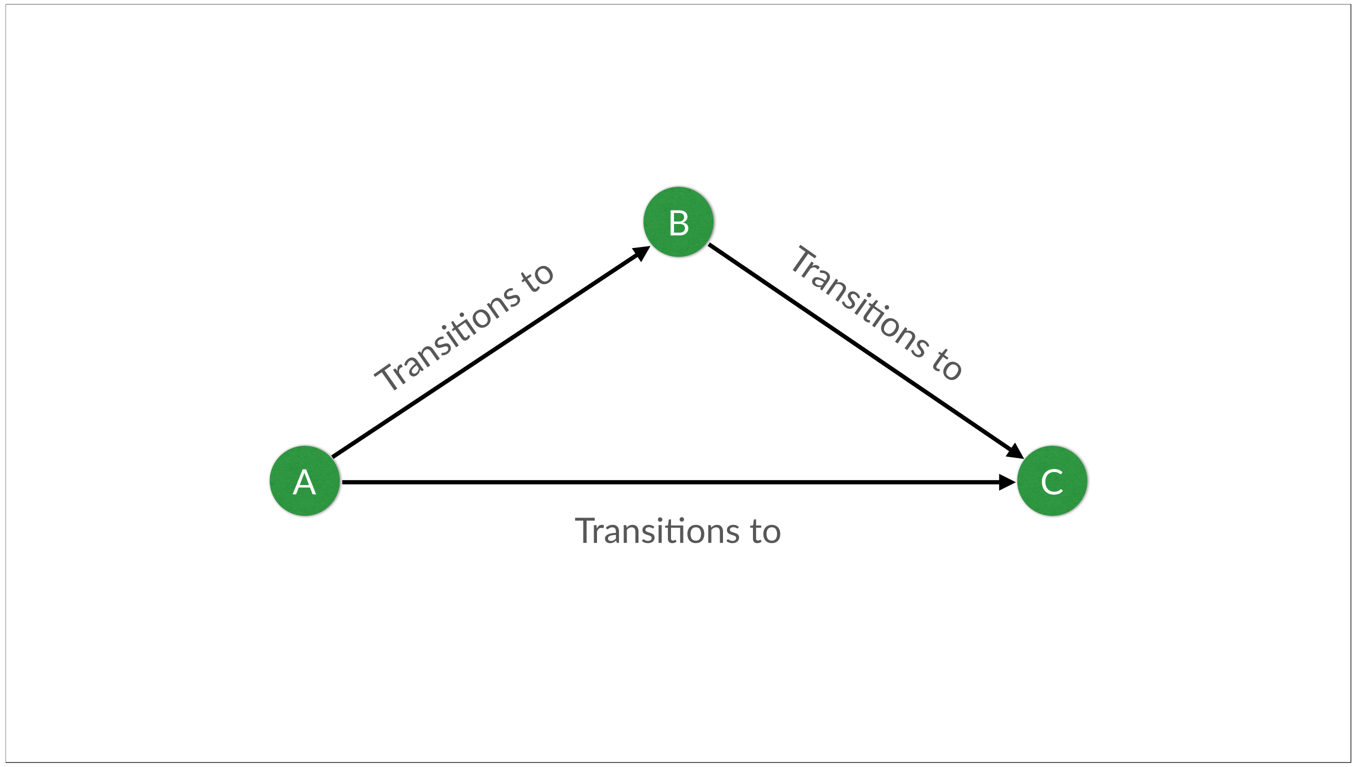
A slightly more complex graph definition allows for links to have direction. This is a directed graph. Unsurprisingly the previous version was an undirected graph.



We can go further and store information on the nodes which now makes our graph useful.

```go
package main

import "fmt"

func main() {
    fmt.Println("We should probably write some code!")
}
```

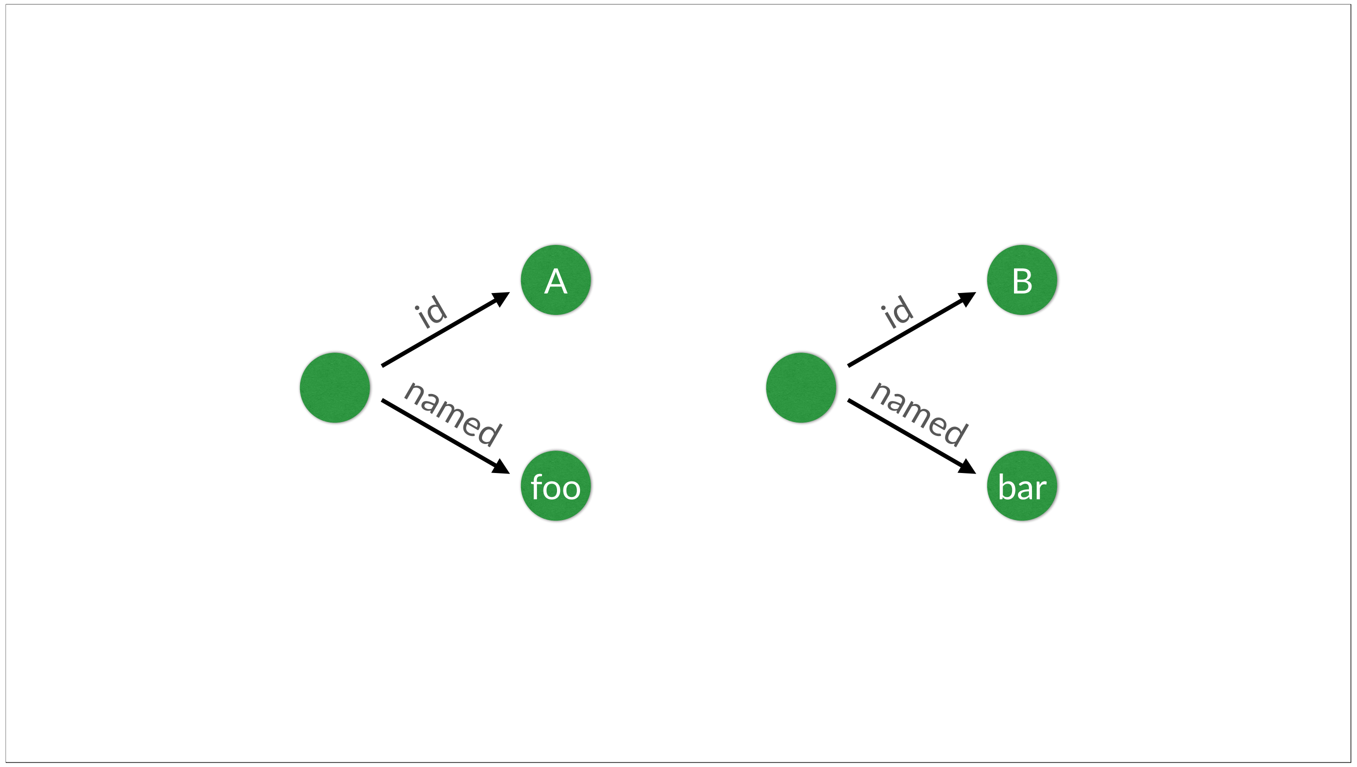
And we can represent this in code:



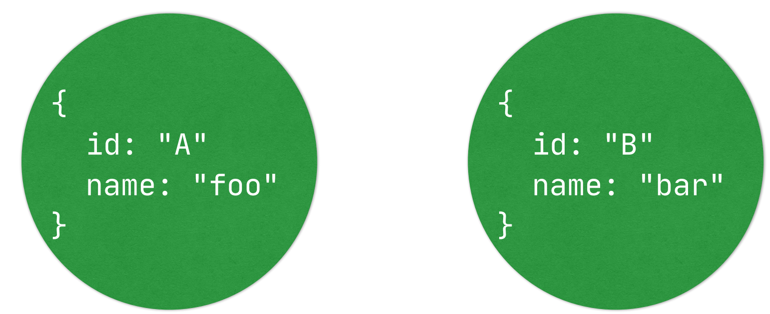
But there's more we can do. I prefer to call links "relationships" because I also store information on the link. This information helps us understand how two nodes are linked, that is, the relationship between them. So lets add that to our data structure.

```go
package main

import "fmt"

func main() {
    fmt.Println("Let's add relationships")
}
```

So lets add that to our data structure.



So let's consider two subgraphs, that is portions of a graph that look like this. If we were to allow more complex values on our nodes ...



we could do this... This is semantically equivalent. This sort of structure is what backs a property graph.

```go
package main

import "fmt"

func main() {
    fmt.Println("Let's write more code")
}
```

So lets add that to our data structure.

```go
package main

import "fmt"

func main() {
    fmt.Println("Introducing Neo4j")
}
```

```go
package main

import "fmt"

func main() {
	fmt.Println("A basic population")
}
```

```go
package main

import "fmt"

func main() {
	fmt.Println("\"Contact\"")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("Infection")
}
```

```go
package main

import "fmt"

func main() {
	fmt.Println("Location")
}
```

```go
package main

import "fmt"

func main() {
	fmt.Println("Trace")
}
```

Dom Davis
@idomdavis
about.me/idomdavis