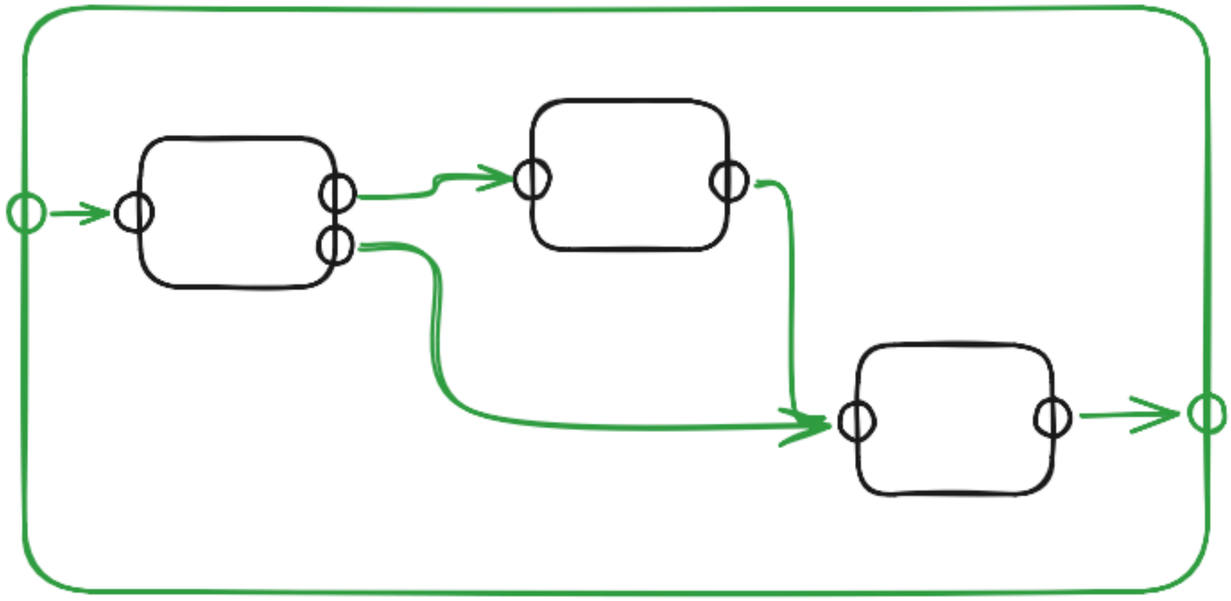
Connections Are Triples



The connections (green) belong to the outer layer (green) not the inner components (black). Example: the left-most component creates 2 outputs. The top output is fed into the input of the 2nd component, but, the left-most component doesn't know that this is the case. Only the green component knows that the left-most component is wired to the 2nd component. Some other Container (green') might wire the components together in a different way. This lets you shuffle the architecture by simply replacing the green component (say, with green''). The left-most component doesn't have knowledge of the 2nd component hard-wired into its code, hence, remains flexible. UNIX pipes are like that, but use of textual syntax ("|") makes it hard to express combinations like above. The diagram *can* be expressed in UNIX but I find the diagram to be instantly obvious. You can't write the diagram with only functions, you need 0D to give you the freedom to move wires around. If you use only functions, without 0D, then you end up having to specify an ordering of execution.

Components have named inputs and outputs, but, Leaves cannot refer to inputs and outputs of other components. Container components can *only* refer to inputs and outputs of their direct children. Container components can contain other components, Leaf or Container, and, provides all routing between children (the children cannot do routing of their own messages). A routing connection is a triple {direction, {src-instance:pin-name}, {receiver-instance:pin-name}}. Most drawing tools assume that a routing connector is a double, not a triple. The Mermaid sample drawing encourages the use of doubles, not triples. The current version of Odin0D does this correctly, but, optimizes a the triple to be { direction, {src-instance:pin-name}, {receiver-queue:pin-name}}. A system of components *can* be optimized to hard-code routing, but, in general this is not flexible enough. Optimization destroys scalability for the sake of "efficiency".

In function-based programming language, you write ‘f(x)’ which hard-wires a call to ‘f’ into the code. This reduces flexibility.

See Also

References *https://guitarvydas.github.io/2024/01/06/References.html*
Blog *https://guitarvydas.github.io/*
Blog *https://publish.obsidian.md/programmingsimplicity*
Videos *https://www.youtube.com/@programmingsimplicity2980*
[see playlist “programming simplicity”]
Discord *https://discord.gg/Jjx62ypR*
X *(Twitter) @paul_tarvydas*
More writing *(WIP): https://leanpub.com/u/paul-tarvydas*