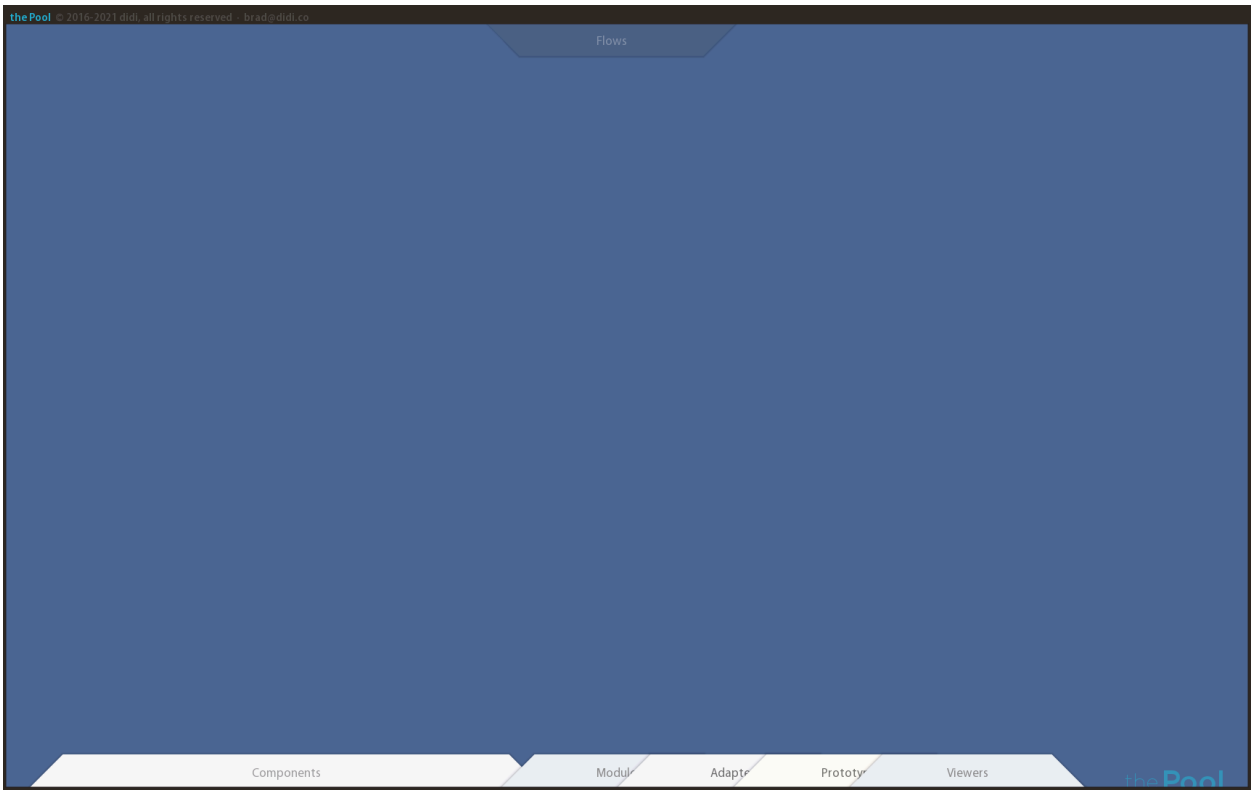
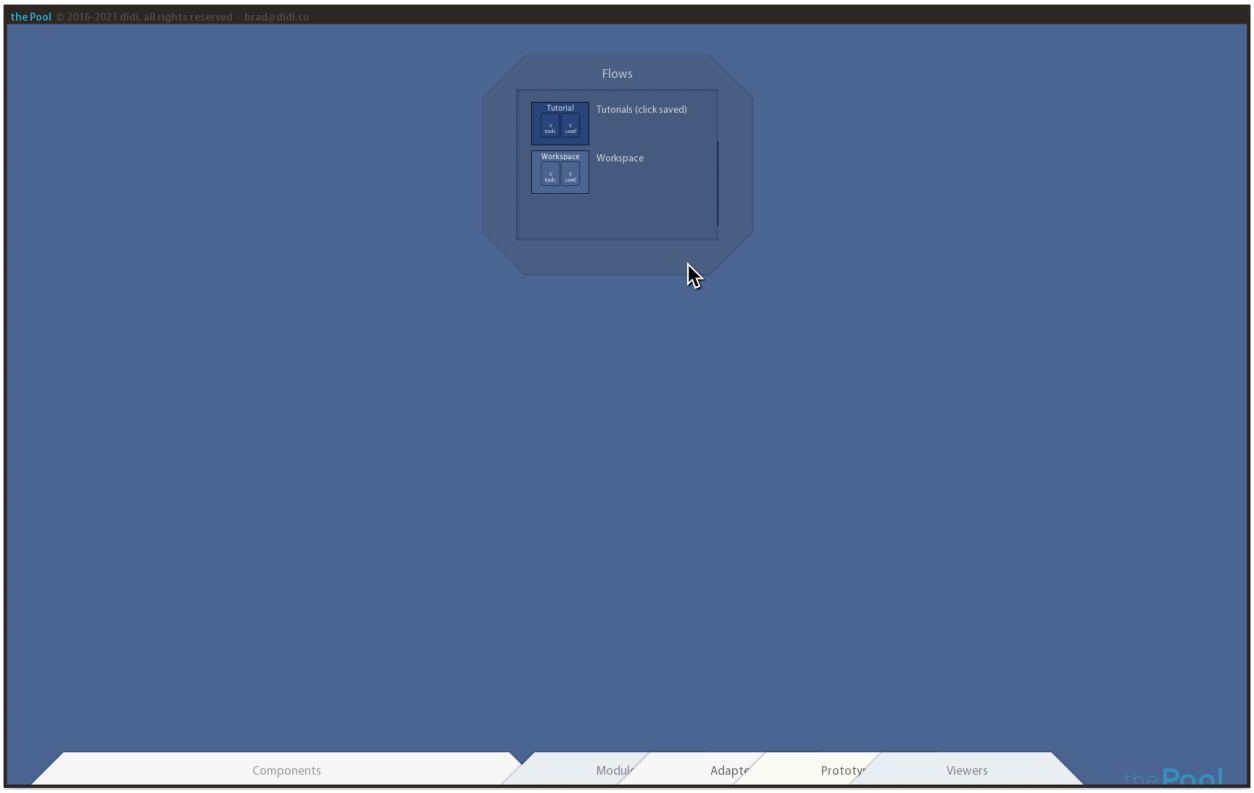
The Pool

Tutorial 1: Modifying a Simple Flow

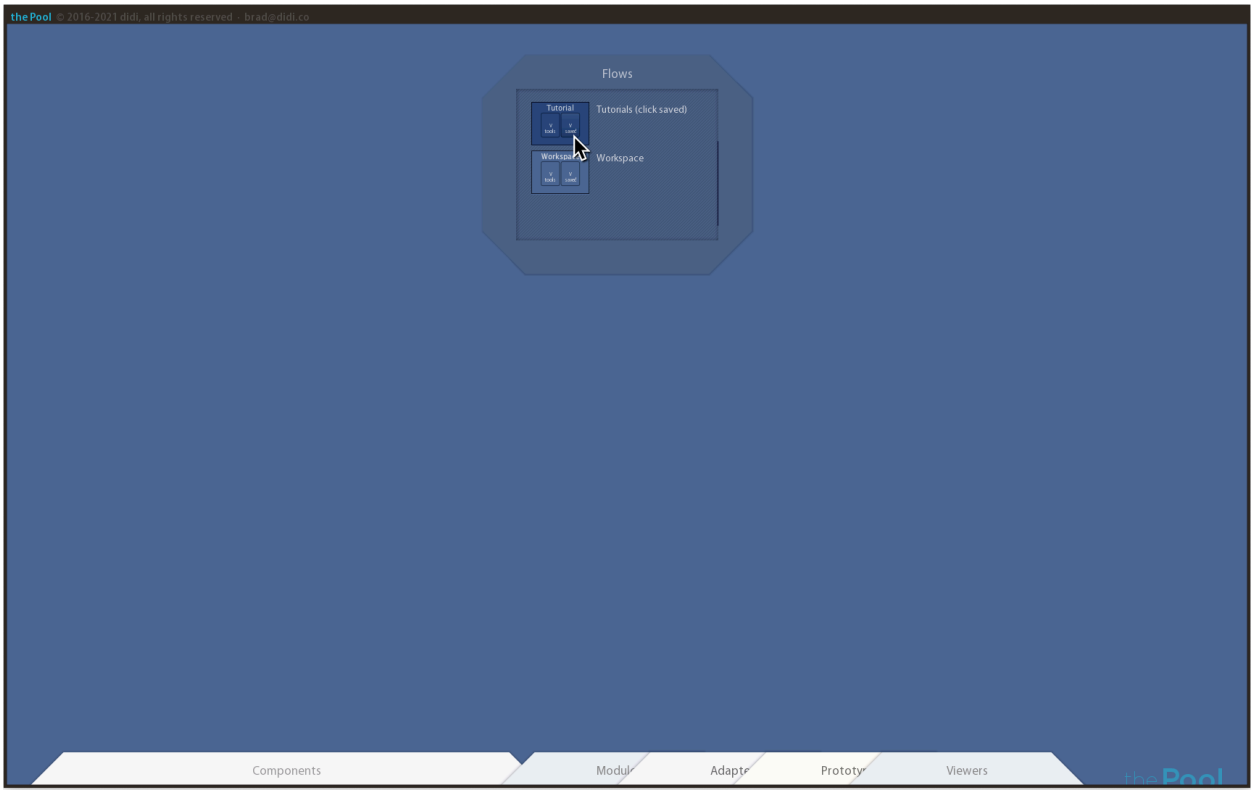
When you start *the Pool* it will look something like this, a nice clean place to think.



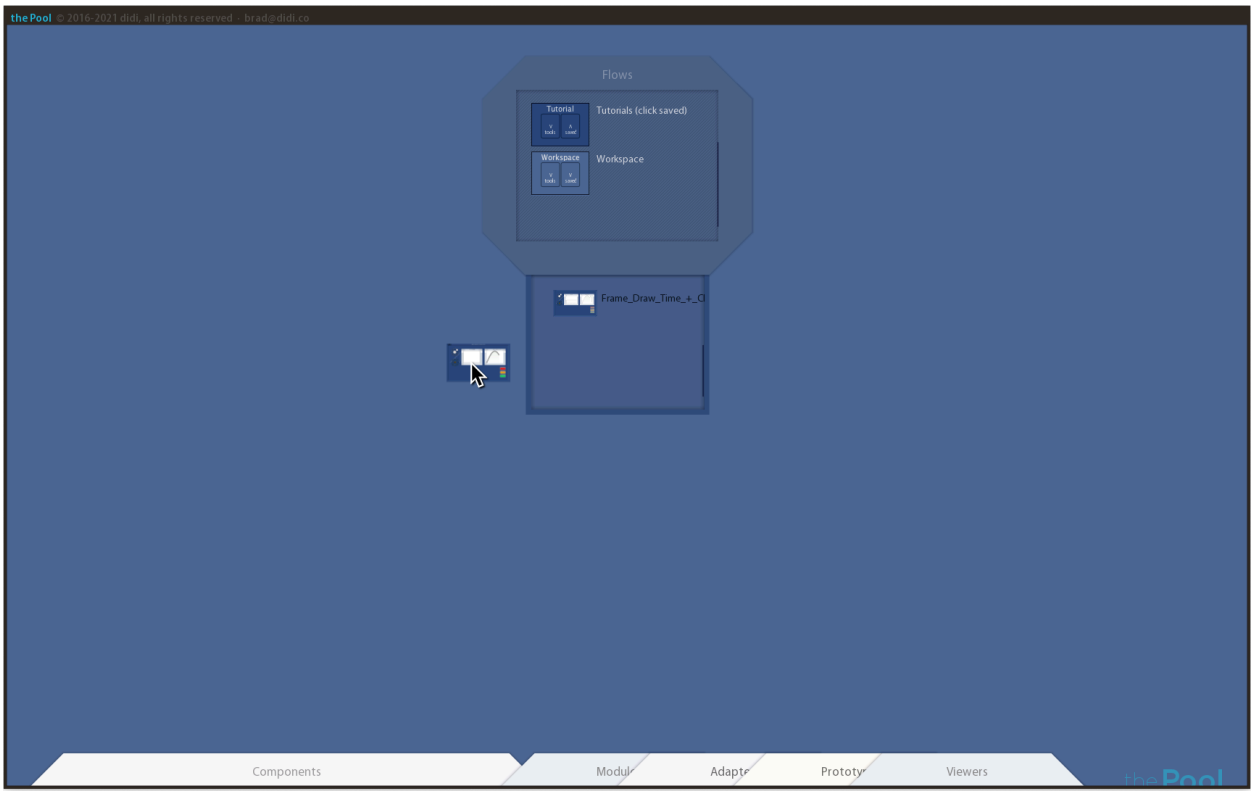
Let's start with an already-existing *Flow* (the Pool's native way of "coding").
Pull down the *Flows* palette, top center.



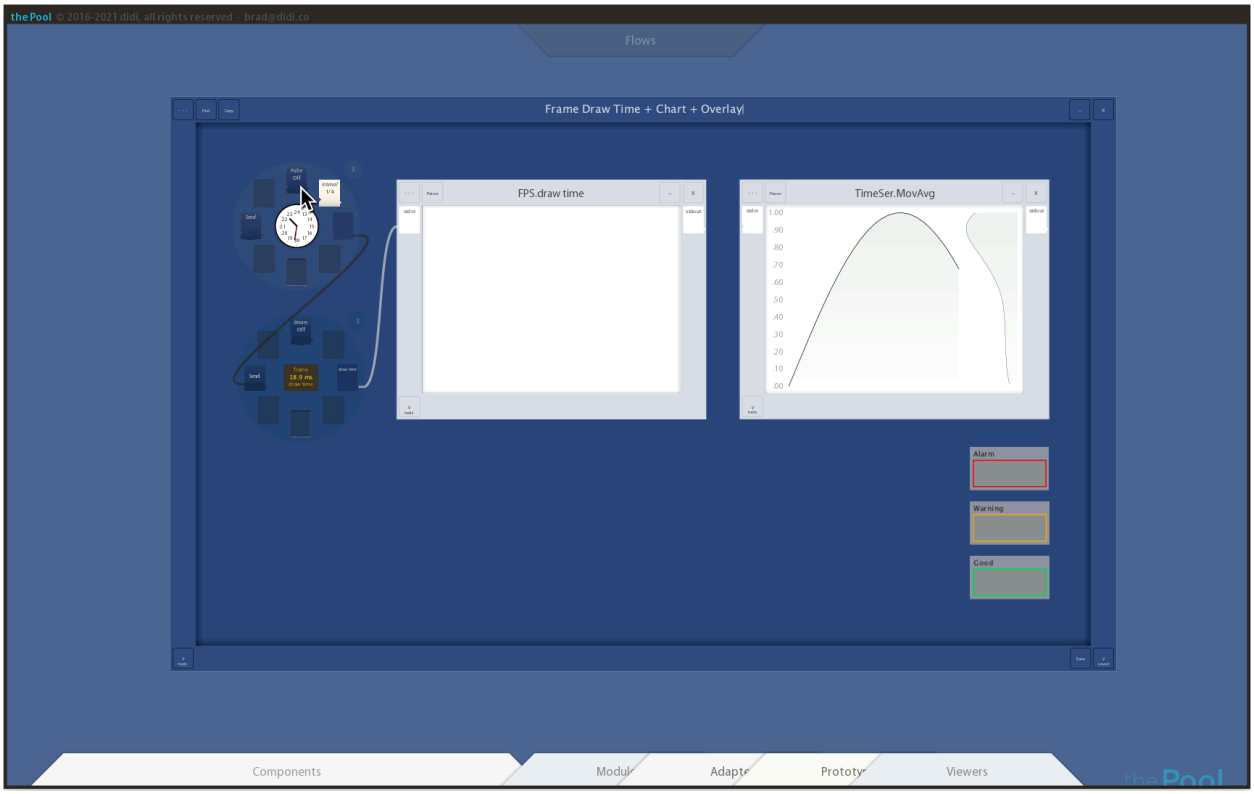
Click the *saved* button in the *Tutorial* box to open its drawer.



Pull *Frame_Draw_Time_+_Chart_+_Overlay* into the Pool.



Click it once to open it. You can fling the palette back to the top; you won't lose it and the drawer will close. Click the *Pulse* button at the top of the *Clock* component to tell it to send pulse messages out its primary outport, at its right.



You'll notice that ports have slightly rounded tops and bottoms and can be clickable buttons, indicated by a slight 3d roundedness. This Pulse button also can handle data flows along pipes as both an inport and an outport; that's indicated by a notch on the left (for inport) and a bump on the right (for outport).

The Pool follows the convention that data flows through pipes, and conceptually from left to right through components and modules, though sometimes the pipes have to be routed to let this happen. You can now see messages flowing along the pipe leaving the Clock outport and entering the Send inport at the left of the frame draw time component.

Components are generally relatively simple sources or operators we can use in flows; there are several in the Components palette at the bottom of the Pool. The frame draw component samples how long it takes the Pool program to draw the image you're looking at: one frame in its real-time animated display.

You could have just clicked its *Send* button, but doing that enough to generate data for our charts would be tedious. The clock takes care of that for us by sending a time message out its outport at certain servals (which can be changed by clicking the *interval* button).

Most components send specific types of messages out their outports—but sometimes the type of message or contents of the messages don't matter. Send (and other clickable) buttons generally respond to any incoming message by just doing what they would have done had you clicked them.

Click the tools button, lower left on the FPS module to open its tool drawer.

Modules do significantly more work than components, or need more space for displays or controls. But their inports and outports work the same way.

Tools related to a module's contents are often useful as ports, so they can be dragged from the module's drawer (or often the drawer of another module of the same type, to save time in applying many tools). They can be dragged directly to the frame where they'll stay, or sometimes into the primary display space of the module to "try it on for size."

Drag the Moving Average tool over the accumulating data—but before letting it go, drag it away again. You'll see the tool apply itself and show its work in the module's display. You'll also see the port associated with this tool's work appear on the left border of the module as an outport, since it's primarily creating data we may want to use downstream.

It is automatically removed if you decide you don't want it and drag it out of the module's primary display area. But this time you do want it, so just let let go while you see the Moving Average's kernel and outport.

The incoming data is somewhat variable and ragged, but its moving average is easier to interpret. That's all we really want to deal with later in our flow, so let's get it into the next chart.

We pipe the data from our new Moving Average outport into the next chart by drawing a line *with the right mouse button* from the outport to the inport where we want it to go.

We do this with a different button because the Pool has two primary purposes: creating and experimenting with data, then later just using the results of our experiments or those of others. When you use the usual left button you're just using the flow—you don't have to worry about breaking it or unplugging something and wondering why it doesn't work any more.

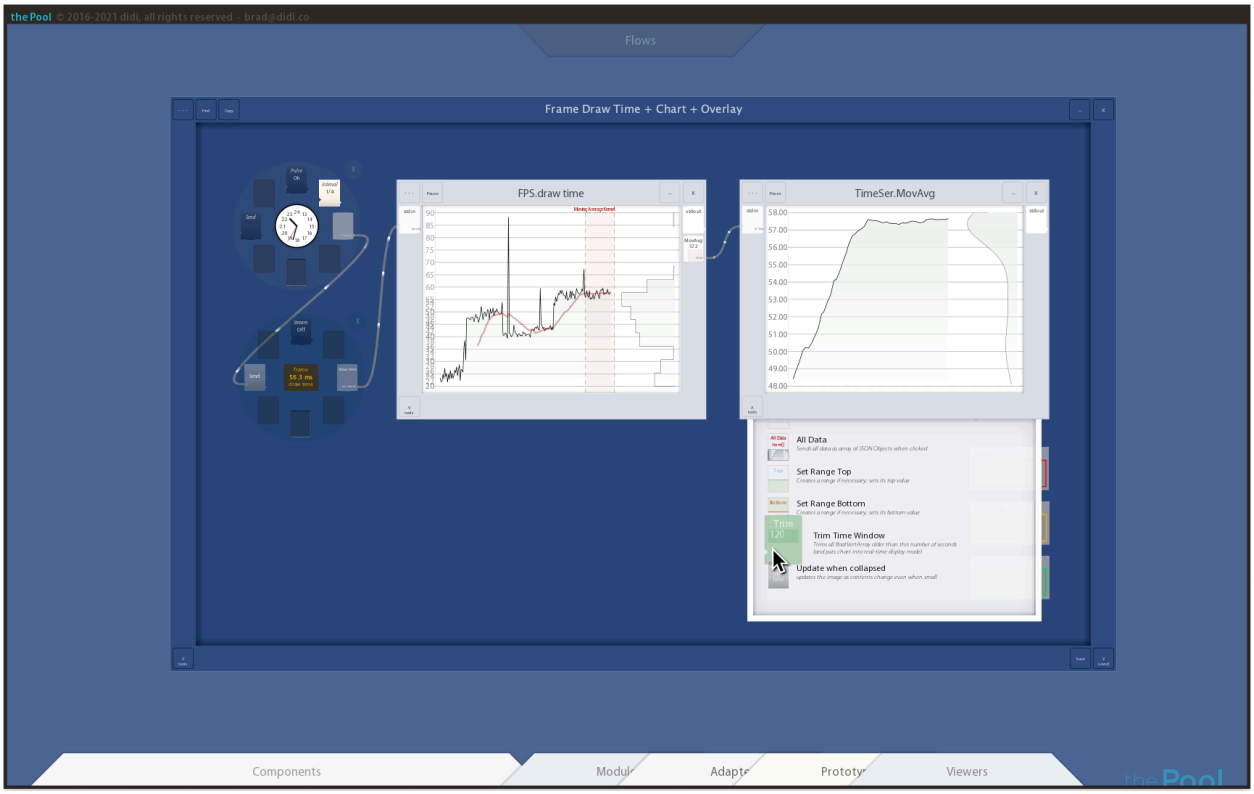
When you use the right button you're creating or modifying a flow, so you *can* pipe things together or unplug a pipe. You can break things—but you can also create things; the maker process. This distinction may be a bit awkward at first, but it corrects for a significant flaw in how we interact with computers that we don't generally see in the physical world.

Imagine if every time you picked up a check or other financial instrument you had to be careful not to touch the amount or other transaction-defining fields. In fact, people in finance have the term "fat-fingered" (as in "I fat-fingered the amount, trying to move the form—what was it again?") It evolved for exactly this problem: it *is* possible to change the nature of many things on a computer by mistake. Imagine if when driving a car you could accidentally detach the steering wheel.

Making and using almost always employ different behaviors and tools in the physical world. By bringing this practice into the Pool we bring the safety and subtly calmer sense that we can move faster without breaking things. We also reinforce the feeling of making and focus needed when making by using different behaviors: different fingers and actions.

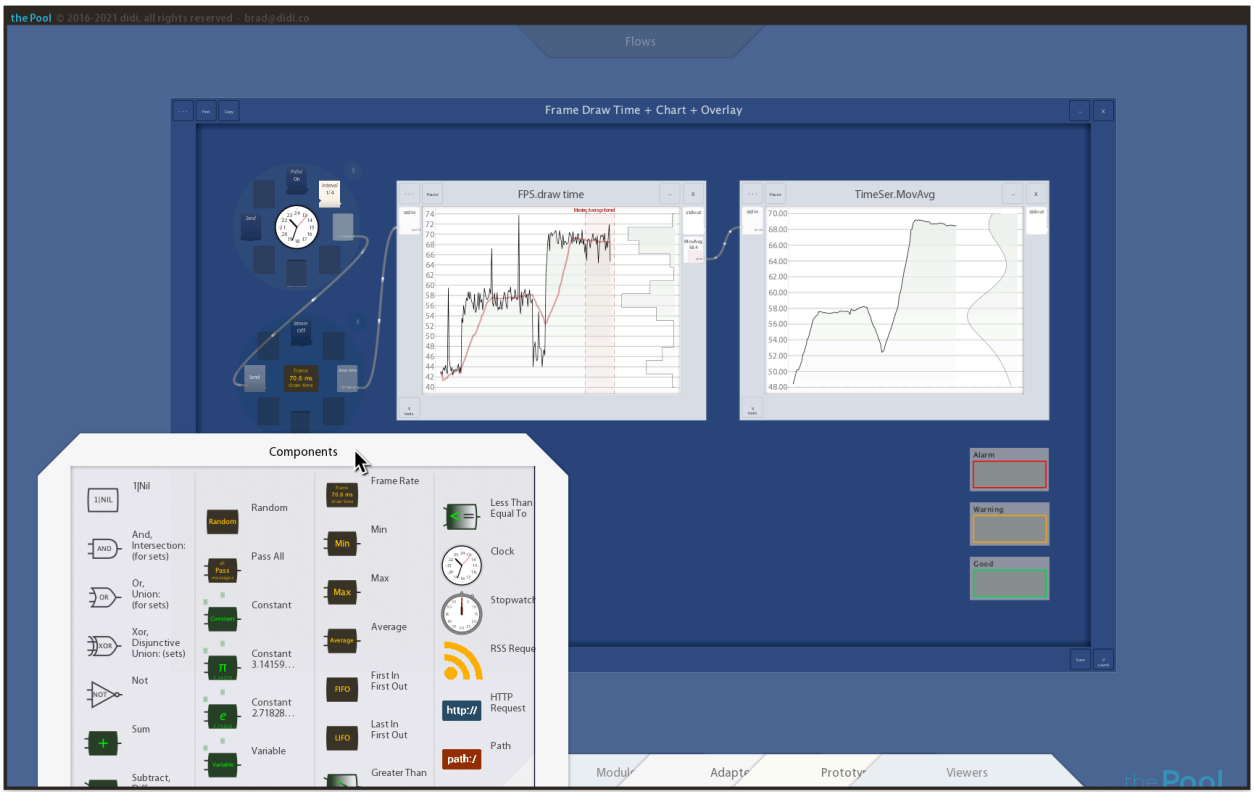
The spikey incoming data probably settled down a bit while you read this, and the less-volatile averaged chart on the right should show something like a plateau. (The Pool prototype is very limited by Java's drawing speeds, so the initial ramp you see on the left is showing how the longer line in the first chart, then the additional line in the second chart made drawing each frame take longer.)

We may need more time to do the next operations, so open the second chart's tool drawer. Find the Time Trim Window tool, and type 120 into its input field to increase the size of the time window displayed in this module to two minutes.



Now that we have some time to experiment, pull up the Components palette on the lower right to see a double-handful of other components we can build into flows.

All that drawing slows the prototype down—giving us a sample of longer frame-draw times. Let it settle for a while longer at this higher-number plateau.



Now press function key **F12** to toggle the palettes off entirely and you'll see faster times.

You've noticed the three alert level modules at the lower right. Let's pipe them into the flow and get them doing something we can pretend is useful for the purposes of this example.

Let's say you can tolerate a moderate amount of complicated-drawing slowness because you know the visual clarity and distinctions will make the system easier to learn, easier to use, and impose lower misinterpretation risk on the experts using the system. And you respect your users so much you'd rather spend a little more time yourself up front—just once during development—than impose the extra effort and risk for every expert, every operation, for the rest of their lives (or at least the life of your system).

But at some point the frame rate's slow enough that the animation choppiness itself imposes cognitive load, so you need a reminder to simplify the drawing or do some optimization.

A red alert might be a good way to catch your eye. But can we stay in this data-flow paradigm and trip that alert without resorting to typing? More important, many smart people can't code, and there's no reason to keep them from doing operations on at least some of their data.

And even coders conceive of many things spatially before they translate that into numbers and code. Here the essence is high numbers means a long time and that choppy animation is bad, low numbers are good, and there's a transitional area that's worth watching because things might get bad if you're keep doing what got you three from the low numbers.

In the right chart use your *right* mouse button (we're acting as *makers* now, not users) to sweep out a vertical range. Click where you want one boundary and drag vertically to where you want the other. Let's put the middle range around beginning of our sample, leaving the longest times above and the shortest times below. Don't worry if you miss at first: you can always click to remove the ranges (and their outports), or right-drag either boundary.

Notice that the ranges and ports are very distinct, quiet colors but without cultural meaning yet: we don't want to assume high numbers are good or bad yet: that's a *meaning* assignment that changes when we use the chart for different purposes. But we do want to subtly reinforce the high/mid/low parts of the range—especially since splitting things into three ranges is a very common act (think Goldilocks' bears' too hot, too cold, and just right).

There's no color key—generally a mistake in common information visualization. But we're building a tool that will be used over and over, so it doesn't want to be too visually cluttered and can stand a tiny bit of training, especially if it's quick, sure, memorable, and almost fun.

Here high numbers are in the sky blue range, low numbers earth brown, and the middle ones are in the tree green range.

That conceptual-to-real-world mapping one of the hundreds of Cognitive Engineering principles that guide the design of tools in the Pool. "One sentence training" is likely to be remembered years later, because it ties meaningful distinctions people already know (sky is up) to what would otherwise be temporary data distinctions that might need to be re-learned every time ("what color are high number again?") This process is known as *embellishment* or *deep processing* in the memory literature: tying new information to things already in memory; called metaphors in the Psycholinguistics literature when the structure of what's already in memory informs and helps organize the new information.

This becomes especially important when data elements are "painted" with colors in one view, and reorganized in another, but that's the topic of another example. The process is called bushing and linking in the information visualization literature.

Let's do that re-mapping of data-space high/mid/low colors into culturally-meaningful red=bad, amber=warning, green=good colors with a couple more pipes, and get our red alert display working. Right-draw a pipe from the big range outport to the small, closed top/red Alarm module. (Piping to a closed module automatically connects to its stdin port: the usual inport that almost all modules have).

Connect the middle, green outport to our Warning module, and figuring out how to get the green one to light up where all systems are go is left as an exercise for the reader…

Play with the Pool's draw complexity input signal to trip the alerts: remember F12 turns on and off the display of the palettes, and revealing the complicated contents of the Components palette is a very high drawing-load task that trips our red alert module. Hide it to see amber.

These ports are a bit more sophisticated than the simple push-button ports that we saw at the beginning of this example. There's a special data type you see as a black message blip in the pipes: it signals "no data." The chart ranges send it out when the charted data line leaves a range; it's used by the alert modules to turn off their fill color.

The Pool allows both strong typing (inports can refuse data coming from outports which send types they can't use) and no typing at all (like the stdin and stdout conventions of Unix tools).

Zoom out with the mouse wheel and try to re-create the flow outside the tutorial box. The clock and frame draw time components are in the Component palette, and the charts and alert modules are in the Viewers palette.