TALA

User Manual

Version 0.3.4 (compatible with D2 0.2.6 and up)

Terrastruct

March 18, 2023

Contents

1 Introduction

Thank you for purchasing a license for TALA (or if you're using it in the web app, for being a Terrastruct customer!).

TALA (short for Terrastruct's AutoLayout Approach) is a proprietary layout engine designed for software architecture diagrams. The software is written entirely in Go, and was developed by Terrastruct over the span of 2 years. TALA is new, but already its current form can often produce diagrams that are preferred over the layouts generated by any other engine.

Unlike other layout engines, TALA is not one algorithm. It is comprised of many algorithms in multiple stages. It identifies parts of the diagram that are structurally like trees, or hierarchies, clusters, etc, and runs different algorithms for each. We used a combination of ideas from the decades of research done in the field of graph visualization, with modifications and new additions to tailor it to software architecture diagrams. More on this in a following section.

If you find an issue in TALA (or this manual), please file it on TALA's GitHub[1]. If it's a D2 issue, please file it on D2's GitHub[2]. Usually anything that pertains to the layout (sizing, positioning) falls under TALA. You can see how active we are in public and be sure that your issue will quickly be attended to.

This manual is meant to accompany the documentation at https://d2lang.com[3]. You should familiarize yourself with D2's language first, as this manual does not cover usage of D2.

We'd love to hear from you. Suggestions, ideas for integrating TALA into something, bugs, good things, bad things – all welcome. The above GitHub links are best, so that everyone can see/search and chime in on an open discussion, but we're also around for live chat ∼18hrs a day on Discord[4]. For private enquiries, email info@terrastruct.com.

2 Command-line usage

If you're using TALA through Terrastruct's web app, you can skip this section (as it's installed on our servers).

[1] https://github.com/terrastruct/TALA
[2] https://github.com/terrastruct/d2
[3] https://d2lang.com
[4] https://discord.gg/aKXhwpCvrY

2.1 Installation

The most convenient way to install is through the install script, together with D2.

```
# With --dry-run the install script will print the commands it will use
# to install without actually installing so you know what it's going to do.
curl -fsSL https://d2lang.com/install.sh | sh -s -- --tala --dry-run
# If things look good, install for real.
curl -fsSL https://d2lang.com/install.sh | sh -s -- --tala
```

Listing 1: Installing TALA

If you want to upgrade versions, just re-run the script.

Alternatively, you can also find binaries releases[5] for Linux and MacOS, for both AMD and ARM, and an installer for Windows.

We also have a Homebrew tap, so you can install via Homebrew if you'd like (the install script uses Homebrew if detected on your machine).

```
brew install terrastruct/tap/tala
```

TALA's binary is named `d2plugin-tala`. To check that it was properly installed and in your PATH, run the following.

```
d2plugin-tala --version
```

To check that D2 can find TALA, run the following.

```
d2 layout tala
```

2.2 Using TALA

To use TALA for D2's layout engine, pass the flag `layout` flag:

[5] https://github.com/terrastruct/TALA/releases

```
d2 --layout tala input.d2
```

Shorthand:

```
d2 -l tala input.d2
```

You can also set it to an environment variable to not have to specify it each time:

```
export D2_LAYOUT = tala
d2 input.d2
```

2.3 License key

The easiest way to reference your license key is to put it in your PATH.

```
export TSTRUCT_TOKEN = "tstruct_..."
```

You can also write the key to a file. By default, TALA will look for this file in your home directory's `.config/tstruct/auth.json`. You can override that.

```
export TSTRUCT_AUTHFILE = "/home/my/path/config.json"
```

3 Considerations for software architecture

Instead of relying solely on theoretical cross-minimizating heuristics to dictate layout like other engines might, TALA emulates how software engineers might draw diagrams on a whiteboard. What follows is a non-exhaustive list of considerations that TALA makes towards that goal.

3.1 Symmetry

Symmetry is an important part of aesthetics and its weighted in TALA's algorithm. There are instances where a shape could've been closer to its neighbor, but doing so would've broken symmetry, and so it gets placed in the position that gives it more symmetry.

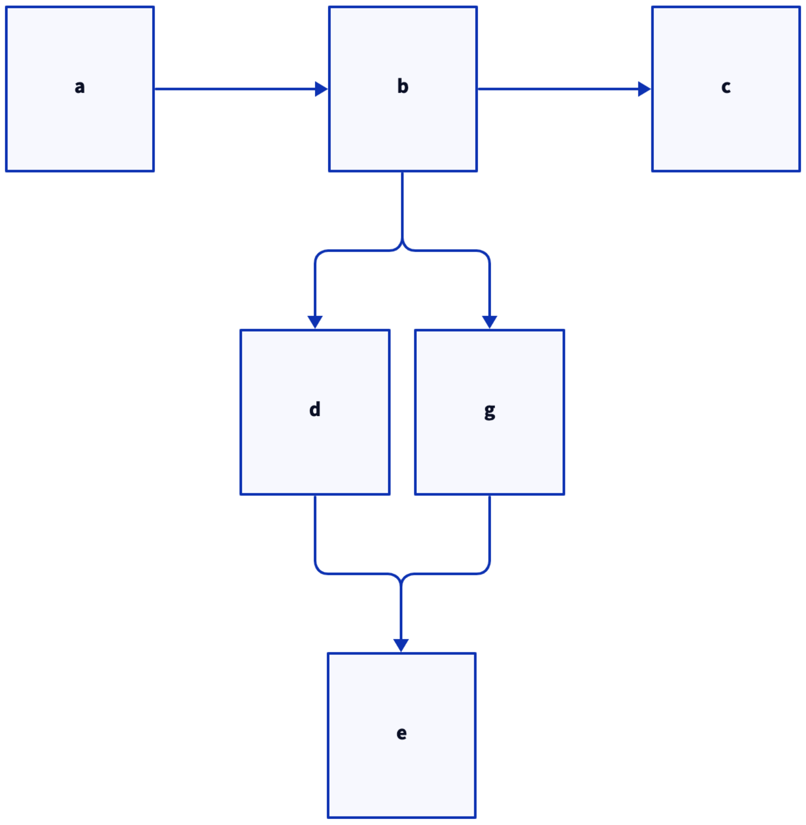


Figure 1: Example of symmetry

3.2 Clusters

Clusters are multiple shapes connected to the same shape as its sibling, at one or both ends. They are laid out all on the same side as their cluster siblings.

For example, notice that "a" and "c" could have minimized distance by going to "x"'s left and right sides, but chose to remain clustered together instead.

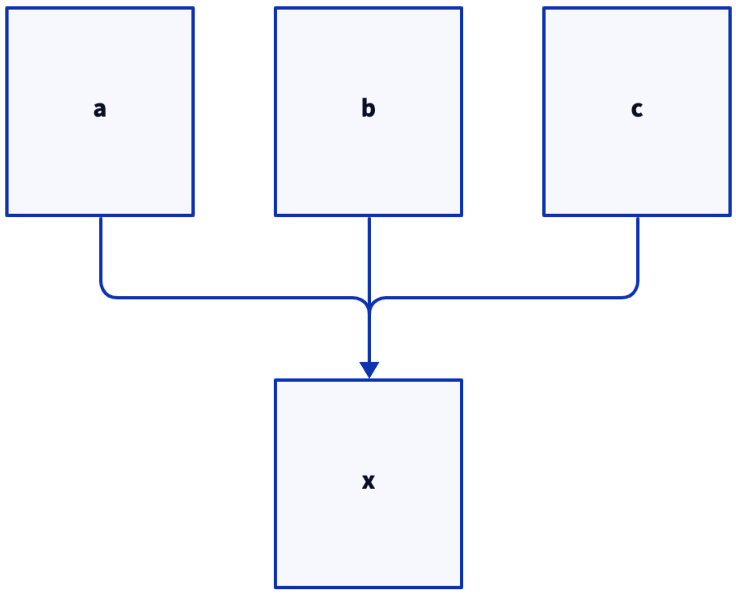


Figure 2: Example of a cluster

3.3 Hierarchy

Hierarchical structures are identified by multiple levels of shapes connected in the same direction and undergoes layout with a special set of rules (e.g. even spacing between levels). Uniquely, TALA handles containers within hierarchies.

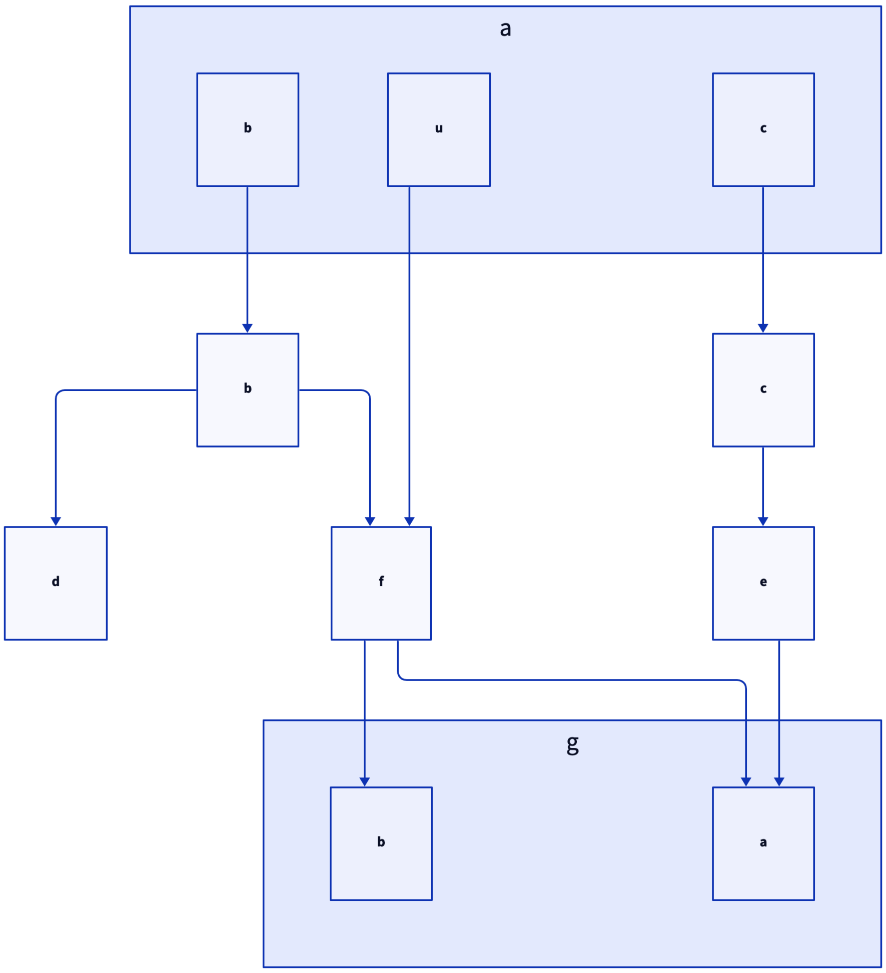
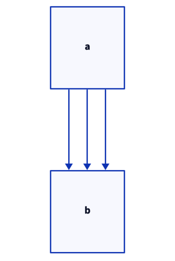


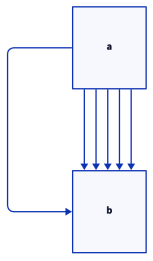
Figure 3: Example of a hierarchy with containers

3.4 Balanced connections

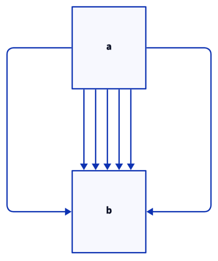
When routing connections, TALA tries to target ports that make them look balanced, given other connections. For example, if it's the only connection on that side, it will try to route to the center – if 3, then at 1/3 intervals, etc.



(a) Balanced connections



(b) Only if enough room



(c) Balancing on different sides

Figure 4: Example of connection balancing

3.5 Dynamic label positioning

Label positions are fixed everywhere. They are positioned to where they avoid obstructions like other shapes and connections.

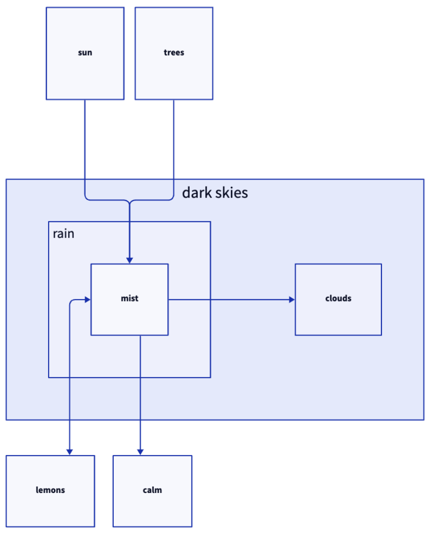


Figure 5: Shape label going to top-left to not collide with routes

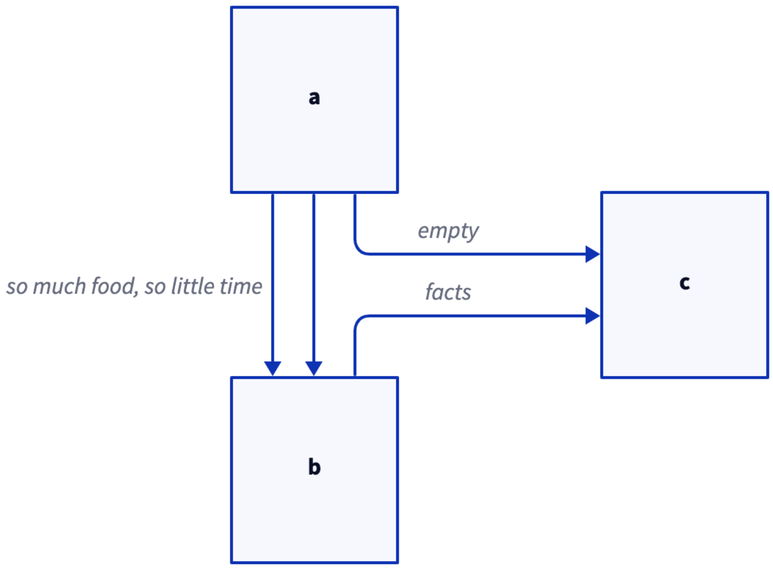


Figure 6: Connection labels avoiding each other

3.6 Square aspect ratio

Non-connected subgraphs are bin-packed to prefer a square aspect ratio.

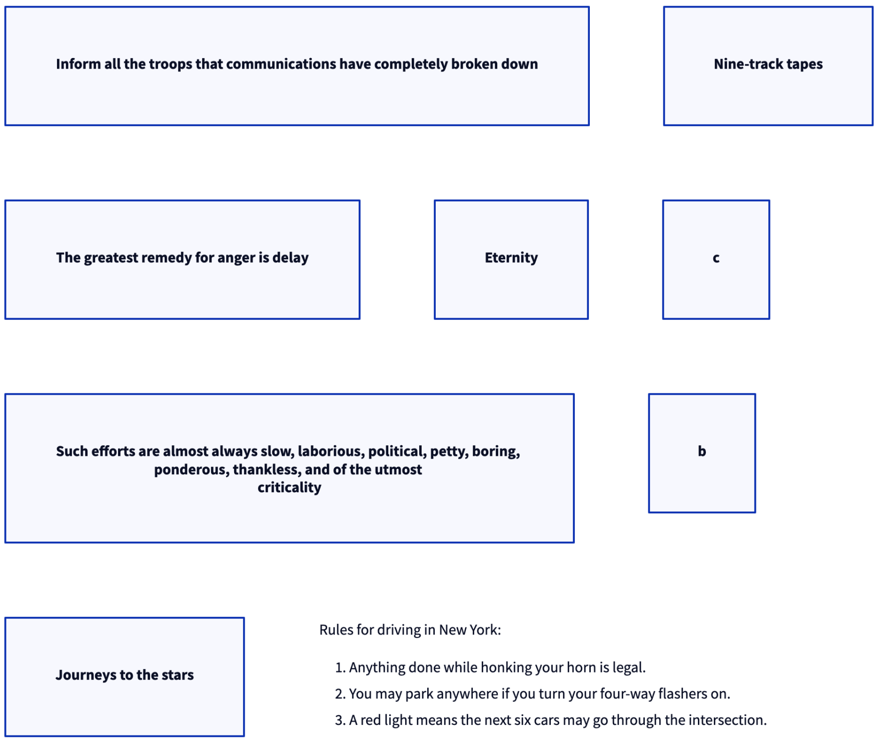

Inform all the troops that communications have completely broken down
Nine-track tapes
The greatest remedy for anger is delay
Eternity
c
Such efforts are almost always slow, laborious, political, petty, boring, ponderous, thankless, and of the utmost criticality
b
Journeys to the stars
Rules for driving in New York:
1. Anything done while honking your horn is legal.
2. You may park anywhere if you turn your four-way flashers on.
3. A red light means the next six cars may go through the intersection.

Figure 7: Example of bin-packing

3.7 Direction

Unlike other layout directions, TALA is able to specify `direction` keyword per container level. The default direction puts a slight preference on down-right (connections point down or right), though if you specify, the preference is stronger.

This keyword is a suggestion, not a requirement. Some connections may still flow the other direction when it improves other heuristics. But in general, the children of container will prefer the direction specified.

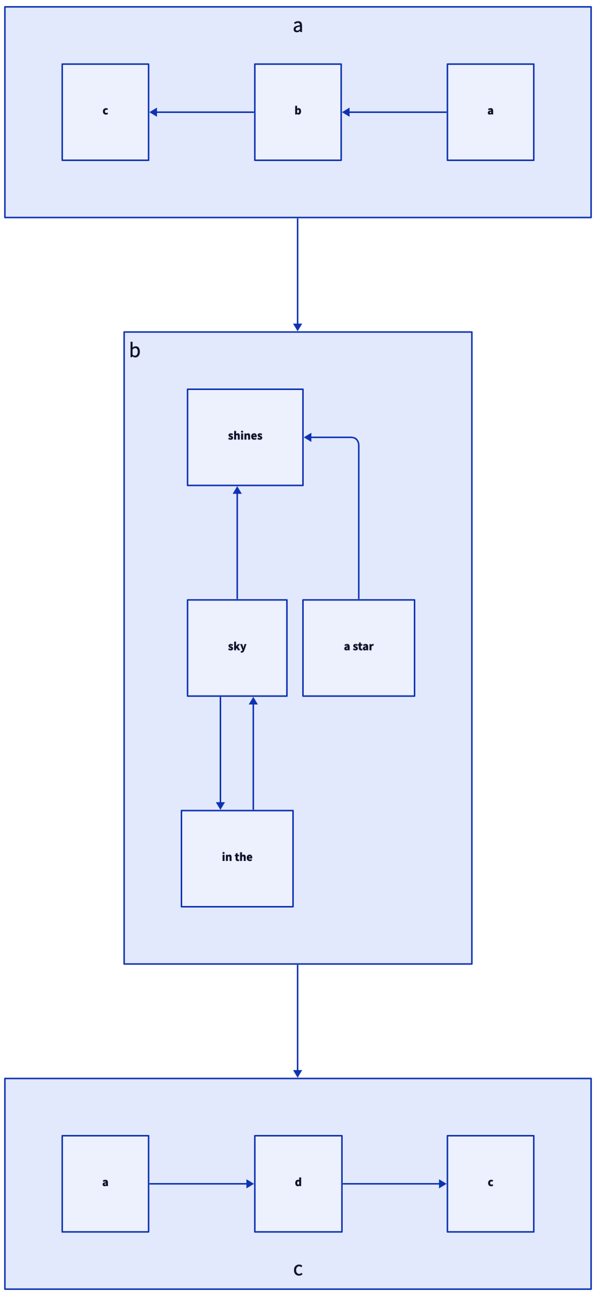
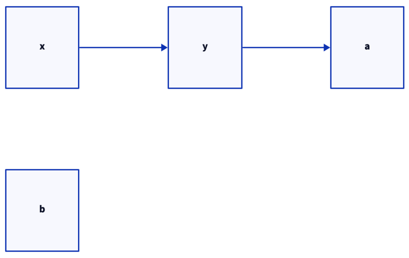


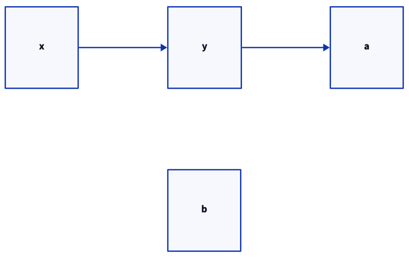
Figure 8: Top level direction is down, container a’s direction is left, container b’s direction is up, container c’s direction is right

3.8 Near

On top of constants (e.g. `top-center`), `near` can target other shapes in TALA. Provide the absolute ID (e.g. a shape `world` inside of container `hello` is specified as `hello.world`) to the `near` property to tell TALA to place it in proximity of that shape.



(a) b set to near x

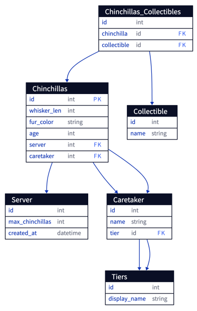


(b) b set to near y

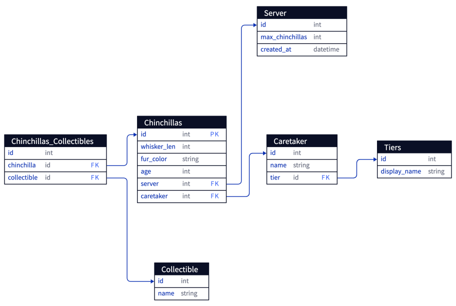
Figure 9: Example of the near attribute

3.9 SQL table column matching

In TALA, connections between two `sql_table` shapes that specify the column will route to that specific column. These diagrams are also known as Entity-Relation Diagrams (ERDs).



(a) Dagre



(b) TALA

Figure 10: Comparison of ERD routing in Dagre vs TALA

3.10 Sequences

If you connect a chain of shapes with their type set to `step`, TALA arranges them tail to head instead of using connections as other shape types would.

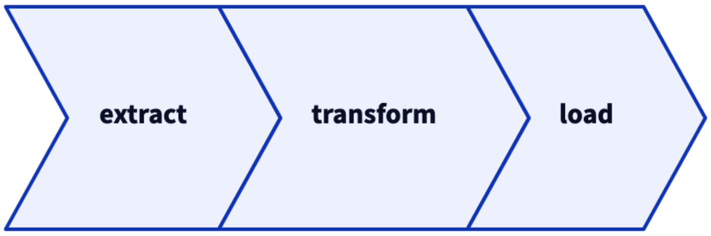


Figure 11: Example of multiple step shapes to form sequence

3.11 Self connections

A self-connection is `x -> x`. In TALA, one corner is reserved for one type of connection (there are 3 types: directed, undirected, bidirected). If there are multiple connections of the same type, the corner is reused.

Notice how the outer loop makes room for the inner loop's label.

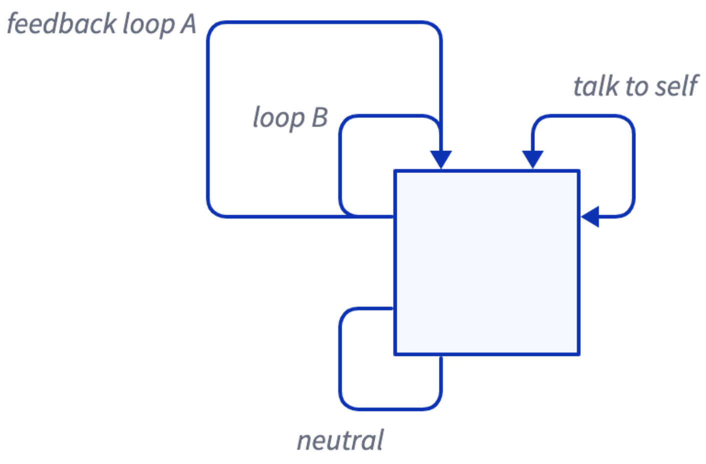


Figure 12: Example of multiple self-connections of varying relationships

4 Seed

Optimal placements of nodes that minimizes distance and crossings is NP-hard. More so when constraints like hierarchical are removed. TALA searches with heuristics to approximate. This search space has some randomness at each step. Choosing a different seed for this randomness can have significant impact on the overall layout, as it may converge on an entirely different look at the end.

The Terrastruct web app takes 3 seeds and runs them all in parallel, and uses a different set of heuristics to choose the best one. On the CLI, you can specify the seed with a flag, `tala-seeds`.

```
d2 --layout tala --tala-seeds 44 input.d2
```

Listing 2: Specifying the seed on the CLI

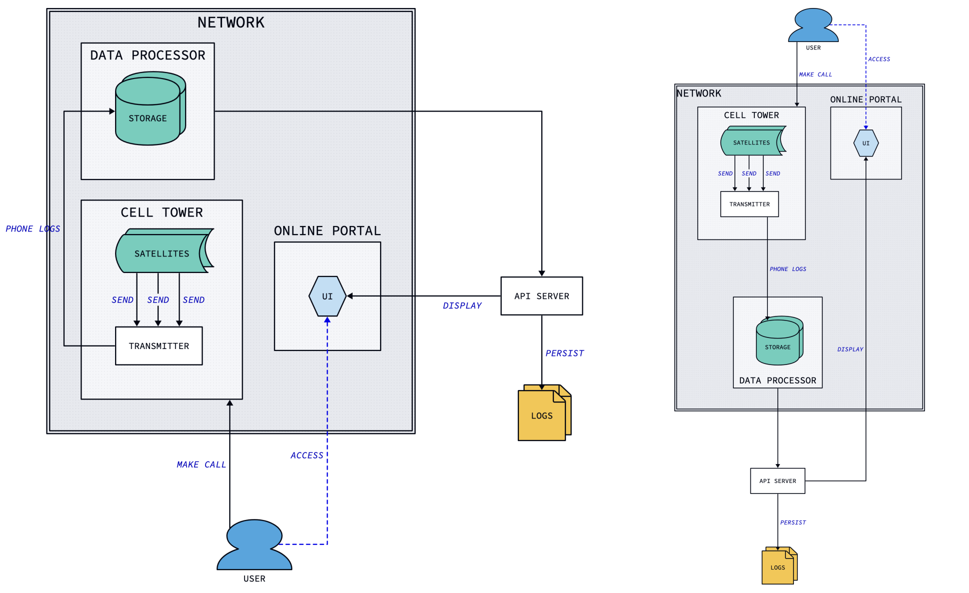
You can also provide more than one seed, which runs multiple executions using the different seeds, in parallel, and uses a heuristic to choose the best outcome.

```
d2 --layout tala --tala-seeds 8,3,24 input.d2
```

Listing 3: Specifying multiple seeds on the CLI

The intended usage is to "re-roll" for a better layout if TALA did not give an ideal layout with the default. The default is to race the seeds 1, 2, 3.

For example, below, the seed 999 happens to give a better outcome (more symmetrical, straighter lines) than the default. So when it's passed, even alongside others, the resulting diagram is different.



(a) Default seeds (b) –tala-seeds=1,2,3,999

Figure 13: Same diagram, different seeds

If your computer is more powerful, you can run more seeds in parallel all the time. If TALA is taking too much resources by default, you can set it to use one seed all the time.

Not all diagrams will look different with different seeds – multiple random seeds may converge to the same diagram.

5 Positioning

Being able to specify fixed positions is an extremely powerful way to customize into your ideal diagram.

There are 2 scenarios where you might want to specify positions:

- You have a specific configuration in mind that can't be described in D2. (e.g. maybe you want to overlap two shapes in a specific way).
- You want to override a local layout decision.

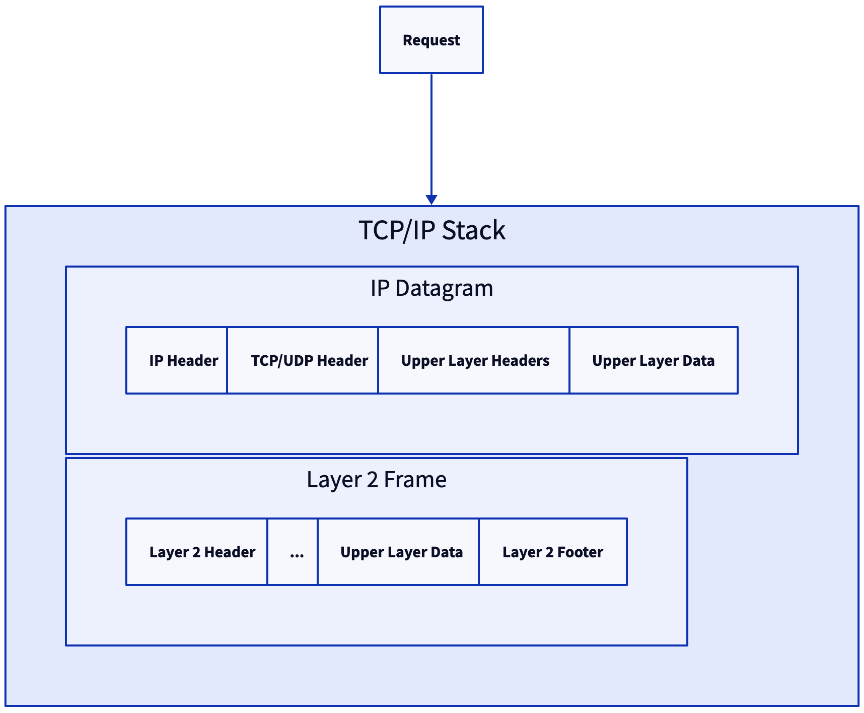


Figure 14: Using custom positions to construct a very specific configuration

```
direction: down
TCP/IP Stack: {
  IP Datagram: {
    IP Header: {
      top: 0
      left: 0
    }
    TCP/UDP Header: {
      top: 0
      left: 100
    }
    Upper Layer Headers: {
      top: 0
      left: 250
    }
    Upper Layer Data: {
      top: 0
      left: 440
    }
    top: 0
    left: 0
  }

  Layer 2 Frame: {
    Layer 2 Header: {
      top: 0
      left: 0
    }
    Rest: ... {
      top: 0
      left: 140
    }

    Upper Layer Data: {
      top: 0
      left: 190
    }
    Layer 2 Footer: {
      top: 0
      left: 350
    }
    top: 190
    left: 0
  }
}
Request -> TCP/IP Stack
```

5.1 Relative to the container

Positions are relative to the shape's container. If the shape does not have a container, it's relative to the bounding box of the diagram.

When a position is specified within a container, the top-left is after padding has been applied. Here's an example:

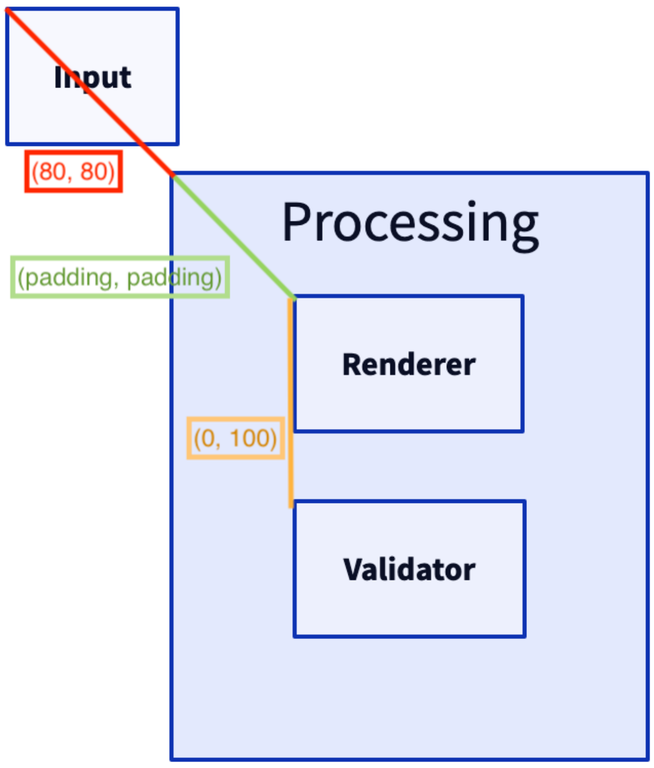


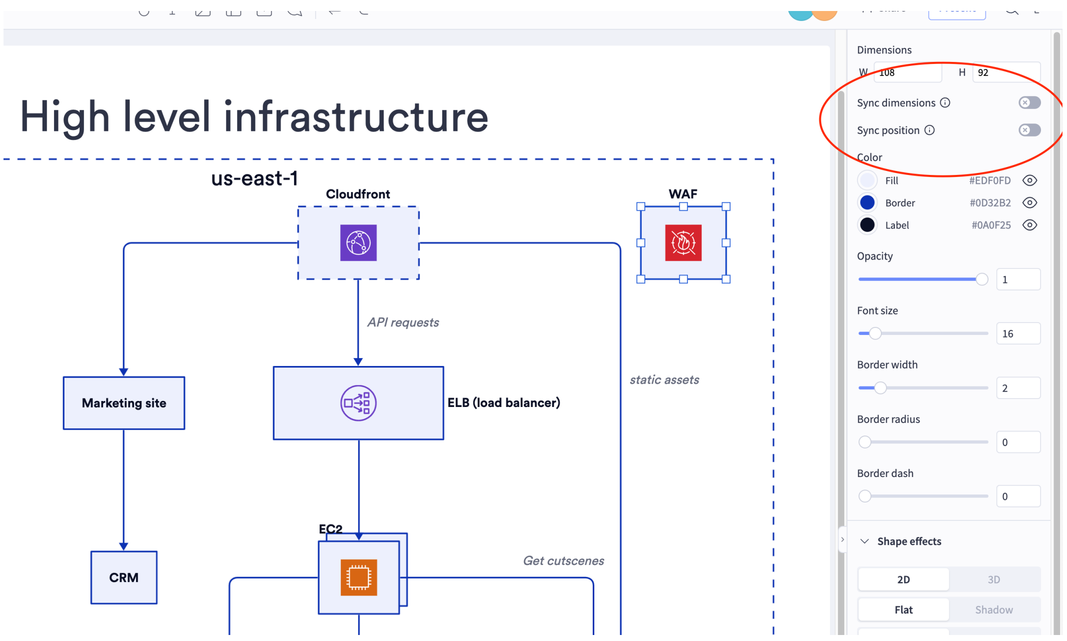
Figure 15: Relative positioning

```
Input: {
  top: 0
  left: 0
}

Processing: {
  top: 80
  left: 80

  Renderer: {
    top: 0
    left: 0
  }
  Validator: {
    top: 100
    left: 0
  }
}
```

`Processing` is specified at `80, 80`, and its bounding box is itself + `Input`. There is no padding. `Renderer` however is specified inside a container, and though it's specified as `0, 0`, it doesn't occupy the same position as `Processing`, because of the container padding.

5.2 Web app

Positioning and sizing are precise operations. D2's watch mode lets you easily choose a value, save, see the result, and repeat until you get the desired effect. And while that's a quick feedback loop, this is one area where using a precise tool is best. In the web app, turning on syncing for positions or dimensions lets you drag and drop to your desired position, updating the values automatically.



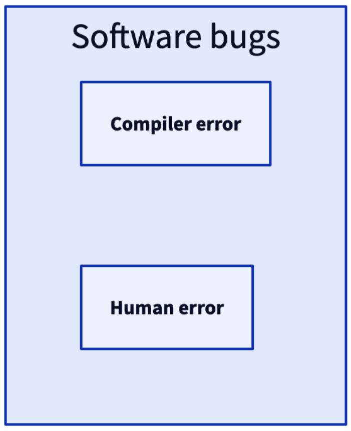
Dimensions
W 108 H 92
Sync dimensions
Sync position
Color
Fill #EDF0FD
Border #0D32B2
Label #0A0F25
Opacity 1
Font size 16
Border width 2
Border radius 0
Border dash 0
Shape effects
2D 3D
Flat Shadow

Figure 16: Turning on syncing

6 Tips

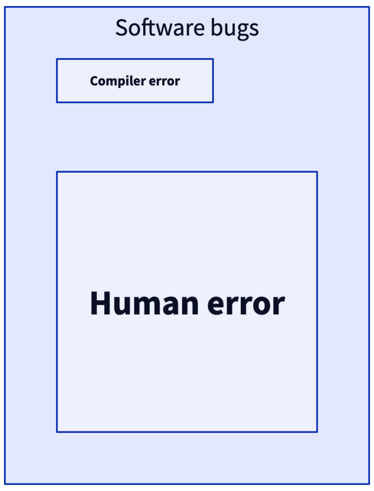
6.1 Dimensions

By default, the dimension of a shape is the dimension of its label, with some padding. TALA does not currently change dimensions other than to expand containers enough to fit children and make cluster siblings uniform. It won't know if you intended a shape to have a larger presence than others. For example, a central "hub" that many connect to. To achieve layouts that more closely match the ideal, you should supply `width` and `height` to shapes where you imagine it to be larger or smaller in scale than others.



```
Software bugs: {
  Compiler error
  Human error
}
```

Figure 17: Default dimensions



```
Software bugs: {
  Compiler error: {
    width: 180
    height: 50
  }
  Human error: {
    width: 300
    height: 300
    style.font-size: 40
  }
}
```

Figure 18: Custom dimensions

Also, connections may look worse bending and twisting to try to reach a well-connected shape situated in the middle of obstructions. Giving custom dimensions can give more space for routes to connect to.

6.2 Label collisions

Though labels dynamically avoid collisions, it will not influence the layout to make room for a disproportionately sized label. For a long label, you may find better results splitting them up into multiple lines with `\n`.

6.3 Legends

Making a legend is a common task that you can do by combining positions and dimensions.

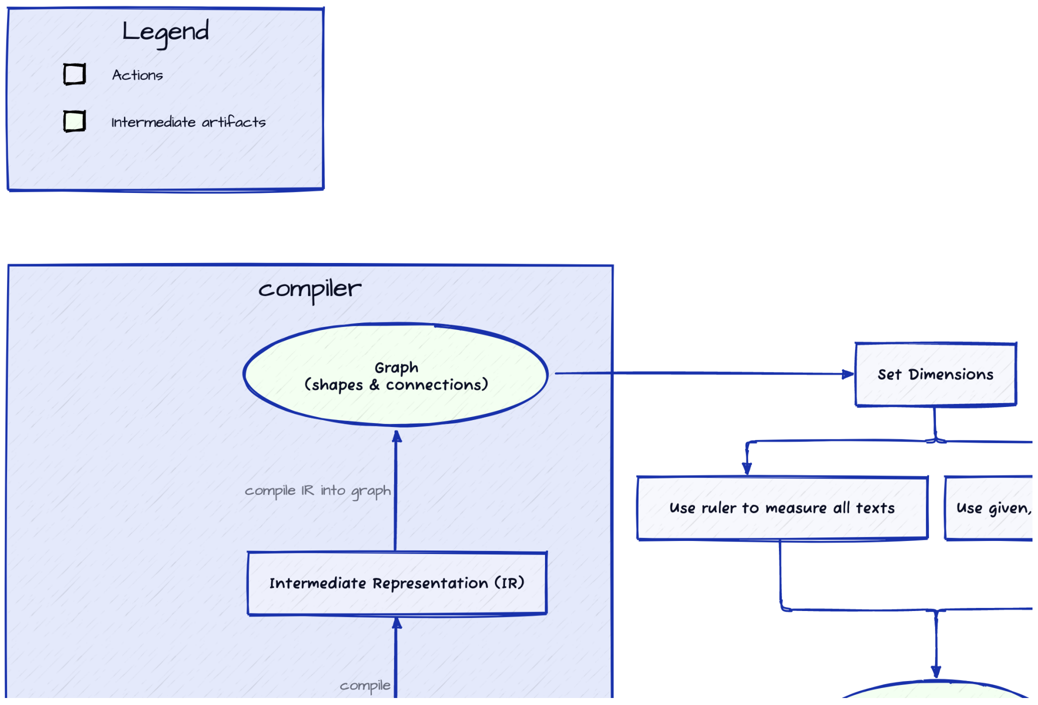


Figure 19: Creating a custom legend for your diagram

```
...
Legend: {
  top: 0
  left: 0
  blue: "" {
    width: 20
    height: 20
    left: 0
    top: 0
    style.stroke: black
  }
  blue-explanation: Actions {
    shape: text
    left: 50
    top: 0
  }

  green: "" {
    width: 20
    height: 20
    left: 0
    top: 50
    style.fill: honeydew
    style.stroke: black
  }
  green-explanation: Intermediate artifacts {
    shape: text
    left: 50
    top: 50
  }
}
...
```

7 FAQ

7.1 Does TALA on the CLI require an internet connection?

No, TALA does not collect telemetry or use the internet in any way except to ping to check the status of a license. This is only done when necessary, e.g. if you purchased a month subscription, TALA will ping at the start of the next month and renew automatically if the subscription is ongoing. If you purchased a year, it won't ping for a year. The only data that's sent in these pings is the API token itself. No diagrams or anything else leaves your computer.

7.2 Can I use TALA for diagrams which are not software architecture?

Yes, and it's known to give good results in many cases. But there's two things to note.

1. There are many categories of diagram types which are "solved". For example, for mind maps, the algorithm that mind-map-specific software has no improvements left to be made. If your diagram type falls under one of these categories, you'll definitely get better results with specific software. Perhaps D2 will support these, but it will be outside of TALA (e.g. D2's sequence diagram layout engine is bundled in D2).
2. TALA is tested exclusively on software architecture diagrams. It is the focus, and we will not change the algorithm to better handle any other type.

7.3 Can I use TALA to layout separately from D2?

Yes, it's possible – they communicate with a serialized, JSON representation of a diagram. There's no reason you couldn't build that JSON from somewhere other than D2. However, it's not an intended use case, so isn't well-documented. If you have a use case in mind, please let us know and we'll help you.