Layers of Functionality

- Many layers of simple-input-API —> simple-output-API —> simple-input-API —> simple-output-API —> …
- ‘+’ is an implementation detail
- implementation details appear *only* in bottom-most layer
    - e.g. all other layers have no syntax for ‘+’
    - e.g. layers can call functions in layer immediately below them (not up, not layers below-below, etc.)
    - e.g. diagrams makes this layering painfully clear
        - use DaS (Diagrams as Syntax)
        - diagrams show input APIs as a set of input ports
        - diagrams show output APIs as set of output ports (N.B. *output* API as opposed to return value(s))
        - diagrams show *composition* as boxes on diagram
            - no need to dig any further down than one layer, before switching to another diagram
            - hierarchy of diagrams (3D <u>across</u> and <u>in</u>, as opposed to 2D <u>across</u>-only)
    -

Compiling Layers

- compiler for *one* layer is uncomplicated
- defer checking details which are not included in a layer
  - loader does final check before running code
- *one* layer is “small”, compiler can be “inefficient”, e.g. backtracking
  - no need to compile whole system at once
  - compile bits of system
    - leave unresolved check breadcrumbs for loader
  - loader does final check before running
    - most checking has already been done incrementally
  - allows mix-and-match of components
    - one app might be X layers deep
    - another app might be Y layers deep
    - incremental checking leaves intermediate “object files” usable in any app, regardless of depth