We discussed noun formats at some length.

Cons cells have wonderful and terrible properties; chief among the terrible ones is that one has to allocate them. High among the wonderful ones is that they're natively supported by the VM's host VM, and indeed we just use them! What we really want, however, is something with inherently native subtree duplication; at the least we can 80% this by manually deduplicating known duplicates.

Atoms serve a variety of purposes; usually they're binaries, but sometimes they're integers, and in the specific case of noun-on-elixir []

(the null terminator for lists) is another sort of zero. For execution we treat these in a very listlike way, i.e. we pop an opcode, pop some arguments to it, and attempt evaluation, but they're hardly ever actual (terminated) lists.

One nice thing we can do is e.g. replace every copy of stdlib with literally the atom :stdlib

, which execution knows and understands (and understands references inside). This even allows us to fast-lane past some hashing to use jets. Stdlib is known!

We also want to print things nicely during development; this probably means a wrapper where some additional metadata can be cached too. During execution it's just naked cons cells, though, for speed of recursion.

Wishlist notes

- use cons cells because they are cons cells
- don't use cons cells because they suck
- use elixir lists directly as noun lists
- support binaries since important atoms are binary
- keep atoms their original type as often as possible
- don't have giant inspect output
- play easily with recursive functions (like nock)
- don't wastefully allocate, allocation is pricey (in time, don't care about space)
- be deduplicable and don't copy the same thing (e.g. stdlib) constantly
- carry type information, which we usually know by means of knowing provenance

What atoms?

- very small atoms: opcodes
- medium sized atoms: tree addresses
- large atoms: binaries

How to print?

- **Noun<1.abc @ stdlib>**

- needs type info
- in the middle of execution might just show #Noun
- at the top level we know the type
- needs type info
- in the middle of execution might just show #Noun
- at the top level we know the type

How to allocate?

- nock_vm

should fix this! * at least on a per-tx level

- stdlib only once
- literally the atom :stdlib
- literally the tuple {:blob [hash]}
- at least on a per-tx level
- stdlib only once
- literally the atom :stdlib
- literally the tuple {:blob [hash]}

How to write?

- defnock

macro