The buzzword-laden phrase is “global referentially transparent namespace”, but what does this mean?

- global: the context you’re reading it from is irrelevant. reading key K on a programmable loom in the 19th century and reading key K on a computer constructed entirely from mental meditation on the capital planet of the galactic empire in the year 999999 work the same.
- referentially transparent: if a read has a result, it always has that result, galactic empires nonwithstanding.
- namespace: names, which we sometimes call “keys”, indicate “values”, whatever those are. the obvious thing everyone goes for is a hierarchical key system, with keys like /anoma/538209580/325029850/32985032850/[more opaque numbers elided]/3258093285032/storage/2024-06-08T10:05:27,802856745+00:00/ray/references-for-examples/counters/counter-1

. But there could be non-obvious things which are better; many things don’t form natural hierarchies! Often what we wish we had is “tags” on values.

The “scry” semantic works as follows:

scry(type, key)

reads the value at fully qualified global &c. key key

with expected type type

. There are three possible results for this operation:

- negative result. this key has no value and never will.
- positive result. here’s the value, which will be the same for all time and all space.

the above two could be collapsed into just “a result is read”. they’re separate because type

and type + 1

are different types.

- not returning at all. nontermination could happen in a few ways; one useful way is to block until getting one of the two real result types, another is to produce a crash, but the semantic remains nontermination. it could

terminate later, as in the blocking case, but sometimes in practice you affirmatively return nontermination, just like sometimes a program loops forever and sometimes it merely crashes to a higher virtualization level.

keys may go unavailable after having had a result read from them at a prior time; this is fine. you can always throw the computer into the ocean, as long as you never change the value read when a value is actually read. so, for example, if your lawyer tells you to expunge this key (say its value is a copy of disney’s latest movie), all you can do is cause the system to nonterminate on attempts to read it. similarly if your read is somehow unauthorized under system semantics, the result is nonterminate (unless it’s some kind of system where the credential is in the key itself, but in that case the credential must either never or always have access to the value - at least at the level of scry; the higher level system servicing scry attempts can always make that credential’s reads nonterminate.)

sometimes this nontermination is in fact just blocking until the value is available; maybe the computer wasn’t thrown into the ocean but just unplugged from the network for a bit. this sort of thing depends on implementation decisions in both the reader and the servicer of read attempts.