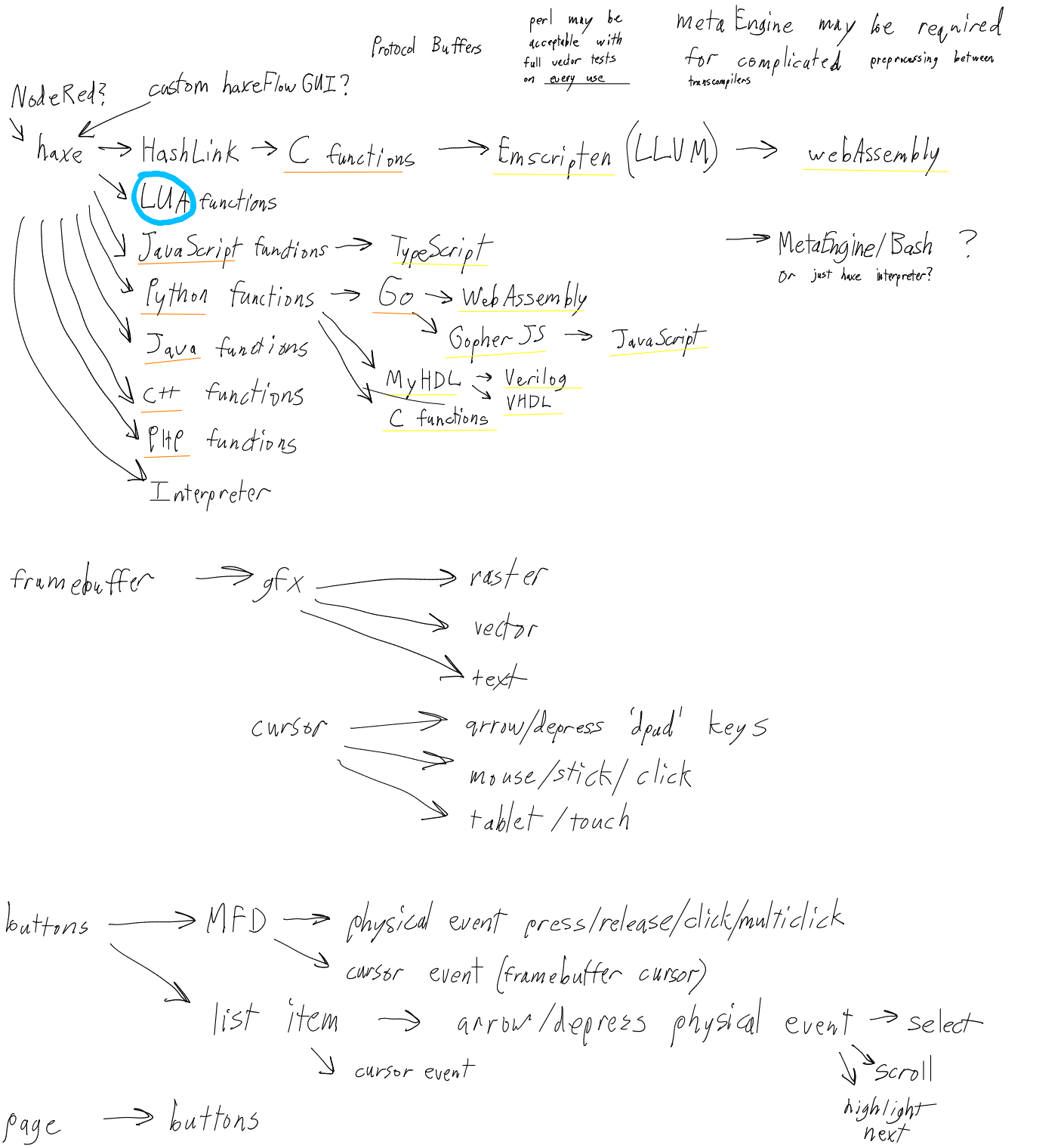


u8glib?

CARDinal --> object benchmarks?

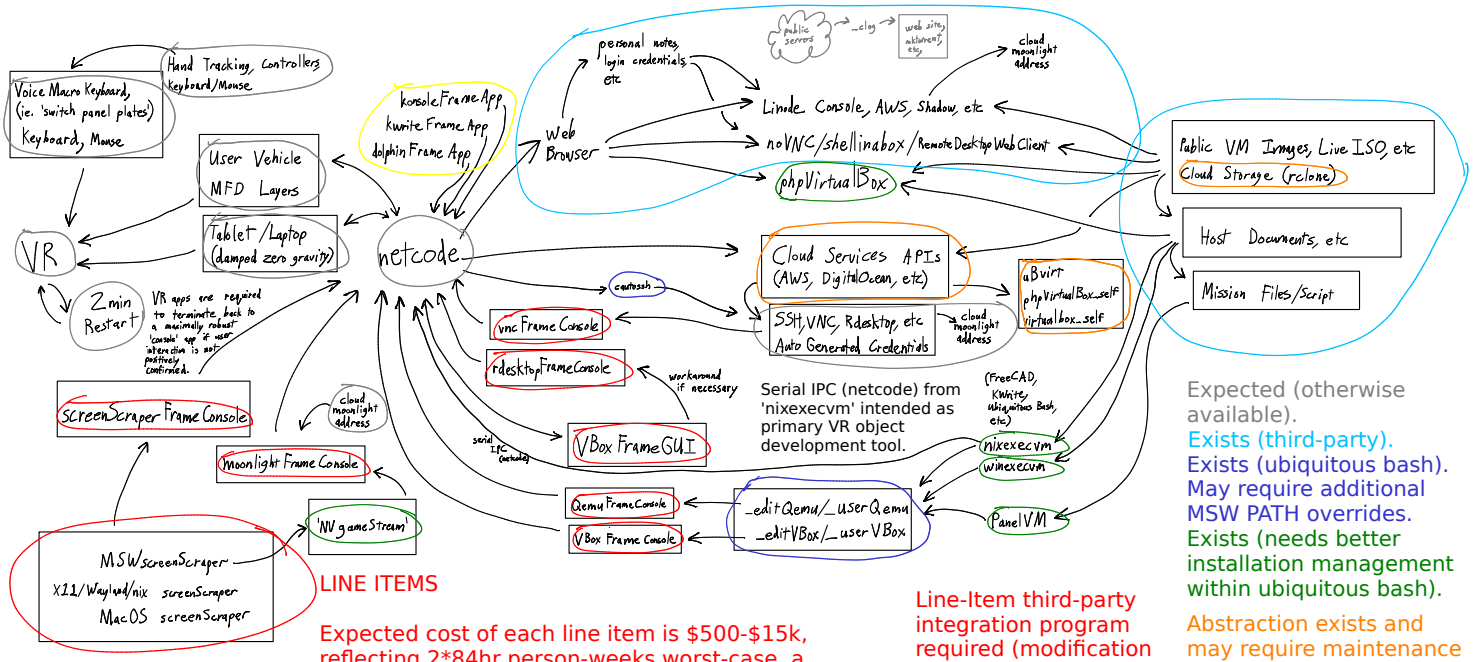
abstraction --> software/hardware UI, netcode, etc

WORLD dynamic objects, voxels, anti-cheat, physics, cross-engine terrain

TODO

- LUA interpreter function access?
- Protocol Buffers 'bin' fundion
- RPC? REST?
- Run packet through all fundions for a catch?

Protocol Buffers / Haxe Protocol Basic Example



Serial IPC (netcode) from 'nixexecvm' intended as primary VR object development tool.

Expected (otherwise available).
Exists (third-party).
Exists (ubiquitous bash).
May require additional MSW PATH overrides.
Exists (needs better installation management within ubiquitous bash).
Abstraction exists and may require maintenance (ubiquitous bash).

LINE ITEMS

Expected cost of each line item is $500-$15k, reflecting 2*84hr person-weeks worst-case, a $120k worst-case total (assuming relevant codebase familiarity per item). Not a substantial expense compared to the value of ensuring adequate 'traditional' 2D application functionality.

Line-Item third-party integration program required (modification of existing open-source).

FOSS native 'Linux' and 'MSW' implementations strictly required for all line items.

A few lightweight apps ported directly to netcode compatible framebuffer interface on both clients and servers may assist development of VR objects from clients incapable of useful 'screenScraper' or even 'InterProcess-Communication'.

Wayland protocol is NOT to be used by 'FrameConsole' and similar. Only a shared-memory triple-buffer of display frames in the exact resolution of the underlying display, and keyboard/mouse events, are to interact with 'Game Engine'. Display resolution change requests are explicit and NOT 'window resize' events.

Secondary to 'nixexecvm' functionality, some VR objects developed with that method (eg. chisel, kwriteFrameApp) are expected to modify other VR objects (eg. marble, netcode script) allowing netcode assisted voxel-to-polygon conversion VR object development.