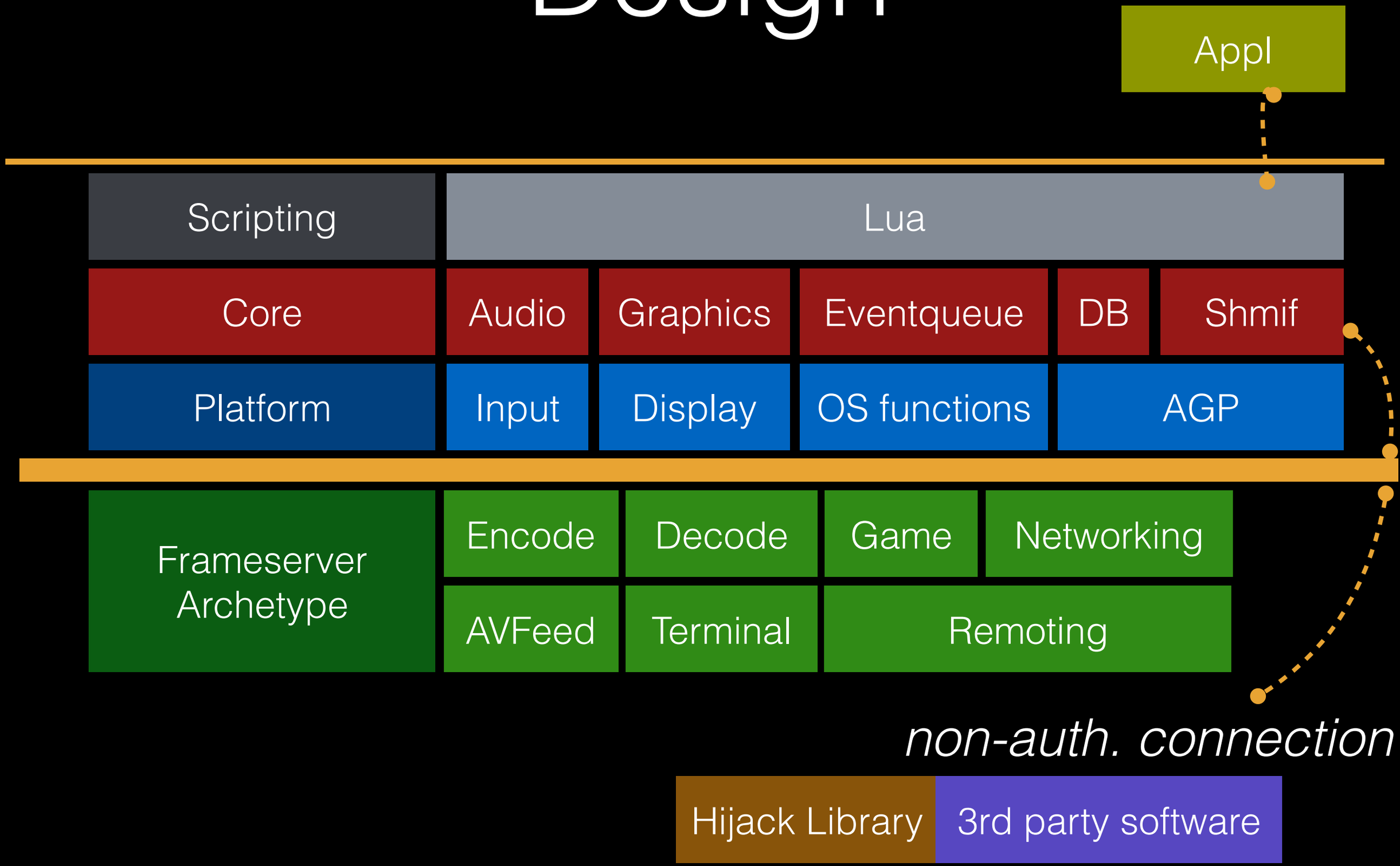
Arcan

Design and Principles

Outline

- Design
- Shmif
- Threat Model
- Governing Principles
- Technical Points and Tradeoffs
- Test Setup
- Current State

Design

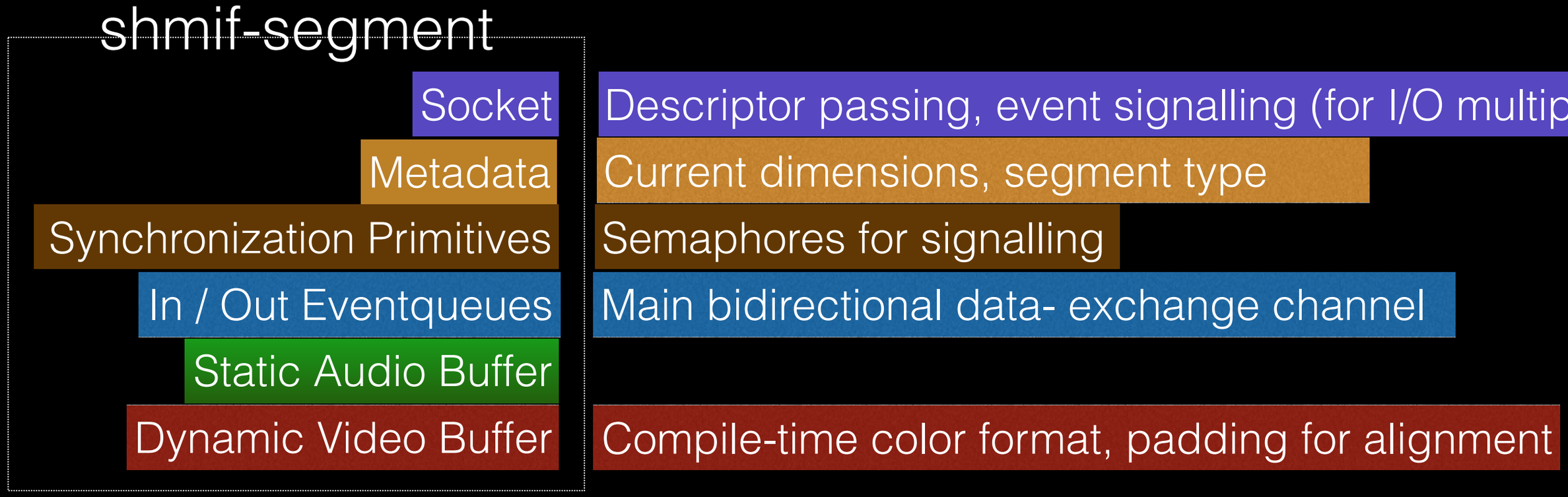

Appl
Scripting
Lua
Core
Audio
Graphics
Eventqueue
DB
Shmif
Platform
Input
Display
OS functions
AGP
Frameserver Archetype
Encode
Decode
Game
Networking
AVFeed
Terminal
Remoting
non-auth. connection
Hijack Library
3rd party software

Frameservers

- Engine can act authoritatively, i.e. kill / control state with minimized risk for cascade or corruption
- Archetype implies specialized behavior / response to possible shmif events
- Should be trivial to swap out the default implementation for one archetype, or have multiple sets to chose from

Shmif

not a ‘public’ interface or protocol

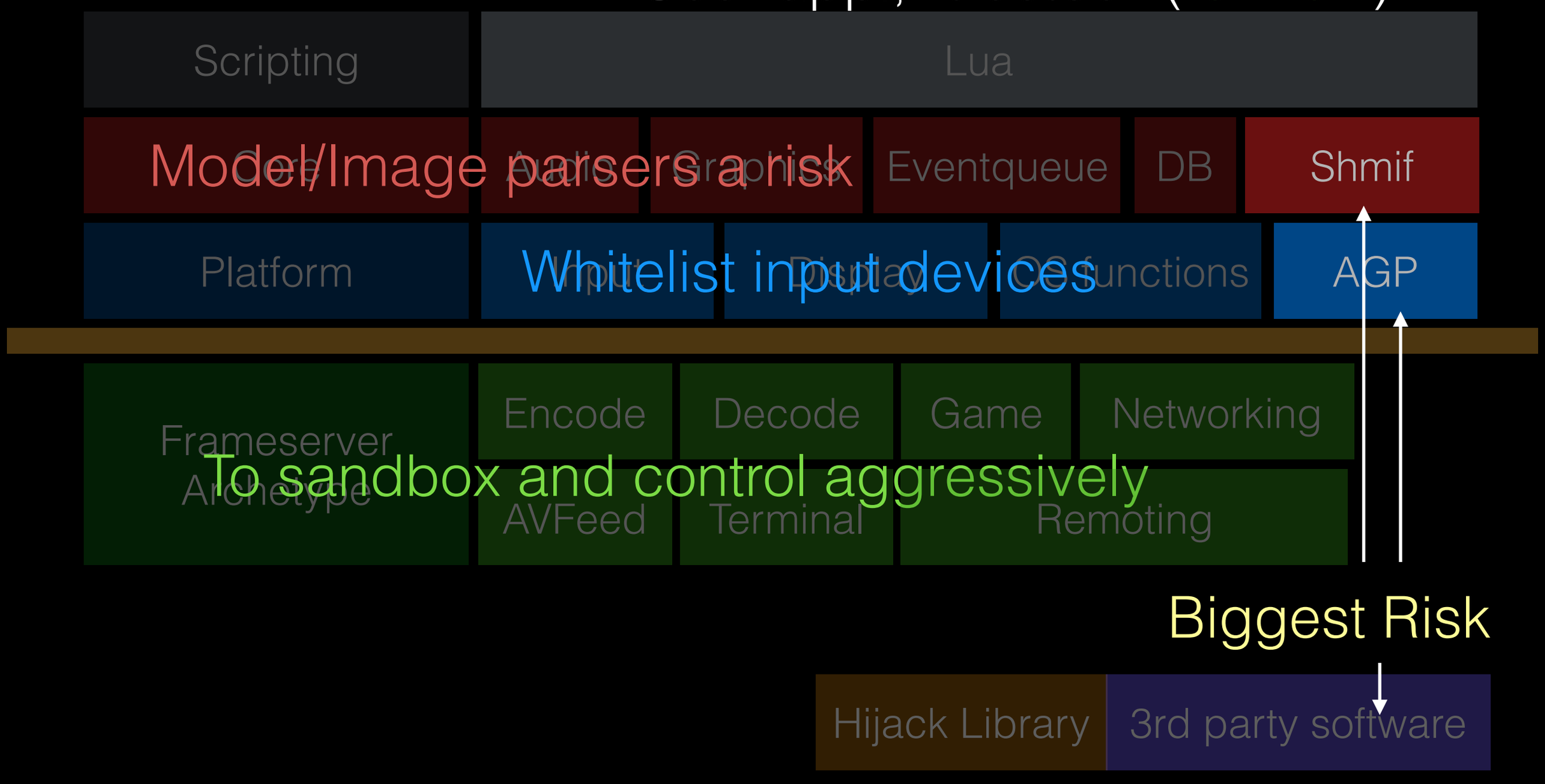


1 (guaranteed), additional ones can be requested or forced

unidirectional (produce or consume)

ordered so that the most error prone targets overflows into something audible or visible

Threat Model



Governing Principles

- No-surprises
  - Safe, Passive, Defaults
  - Running *appl* dictates behaviour
    - And user specifies appl
  - All external connections are explicitly enabled
  - Don’t try to be clever, provide mechanisms for **the user**, make them obvious and accessible - not automated policies

Governing Principles

- Be Untrusting
  - Compartmentation - sensitive actions get their own processes with restricted capabilities - monitor and kill if suspect
  - 3rd Party Applications are not to be trusted
    - Legacy (times change), Ignorance (didn't care about your case) or Personal Agendas (drm, stealing data, building empires...)
    - Any interface that provides a percievable truth should also be able to provide corresponding lies - *this is the virtualization ideal*
    - Should not be able to (or, if possible, only at considerable cost) tell truth from lies
    - communication is a privilege - not a right (cp command does not need network access, firefox does not need .bashrc access)

Governing Principles

- Be Conservative
  - “Modern” is appeal-to-authority nonsense
    - Comes at the cost of exclusion
  - Define the features you want, commit to them
    - Feature/scope creep leads to ‘solving’ general problems that does not fit the problem space of any single individual
    - The Web-browser is the end-game of feature creep and feature creeps (“wouldn’t it be cool and funny to put this in a browser lol?”)*
- Interfaces you export are interfaces you commit to
  - i.e. “we do not break userspace”
- but - pragmatism, **not** ideology

* shoot to kill

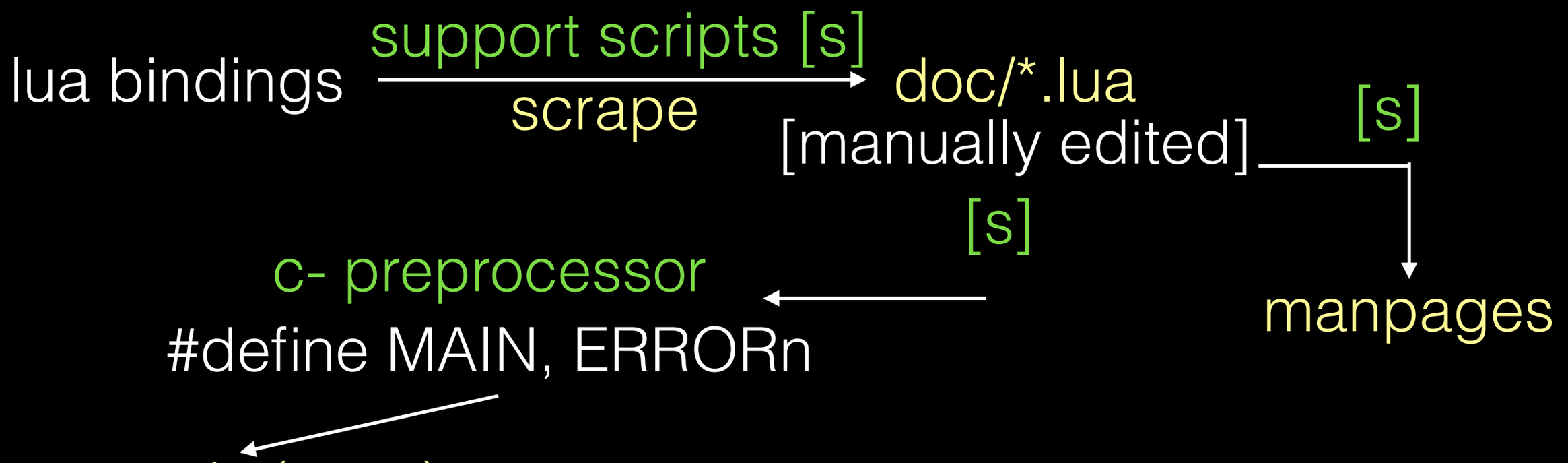
Governing Principles

- Stay Pragmatic
  - Minimize dependencies
    - CM work becomes more complicated, you replace ‘bugs you are guilty of with’ ‘bugs others decide’
    - Never rely so hard on an external solution that you can’t pack your bags and leave
- Stay portable
  - Lets other systems question the validity of your own
- Ignore Appeal to Performance
  - Hard Evidence - Data from specific test cases, not ‘benchmarks’
- Ability to debug drives design choices

Technical Points and Tradeoffs

- Core: 100% C (8998:1999) style, aiming for C11
    - Due to the minimal set of runtime requirements, least variance in implementation quality and complexity. This is a simplicity versus performance tradeoff.
- Primarily single-threaded with domain specific or process separated concurrency. This is a debugability versus performance tradeoff.
- Engine configuration is build-time static with embedded tag (platform, git revision etc.). This is a simplicity versus flexibility tradeoff.

Test / Doc Setup



+ handwritten: tests/
(interactive, benchmark, regression, security, exercises)

atests.rb also generates build permutations etc.

[s] :- docgen.rb, atests.rb

Current State

- Detail on individual components / platforms, “components and status” @ wiki
- See Roadmap on Overview slides
- Default archetype implementations are very *‘barebone’*
- Lots of work left in completing and automating the test setup