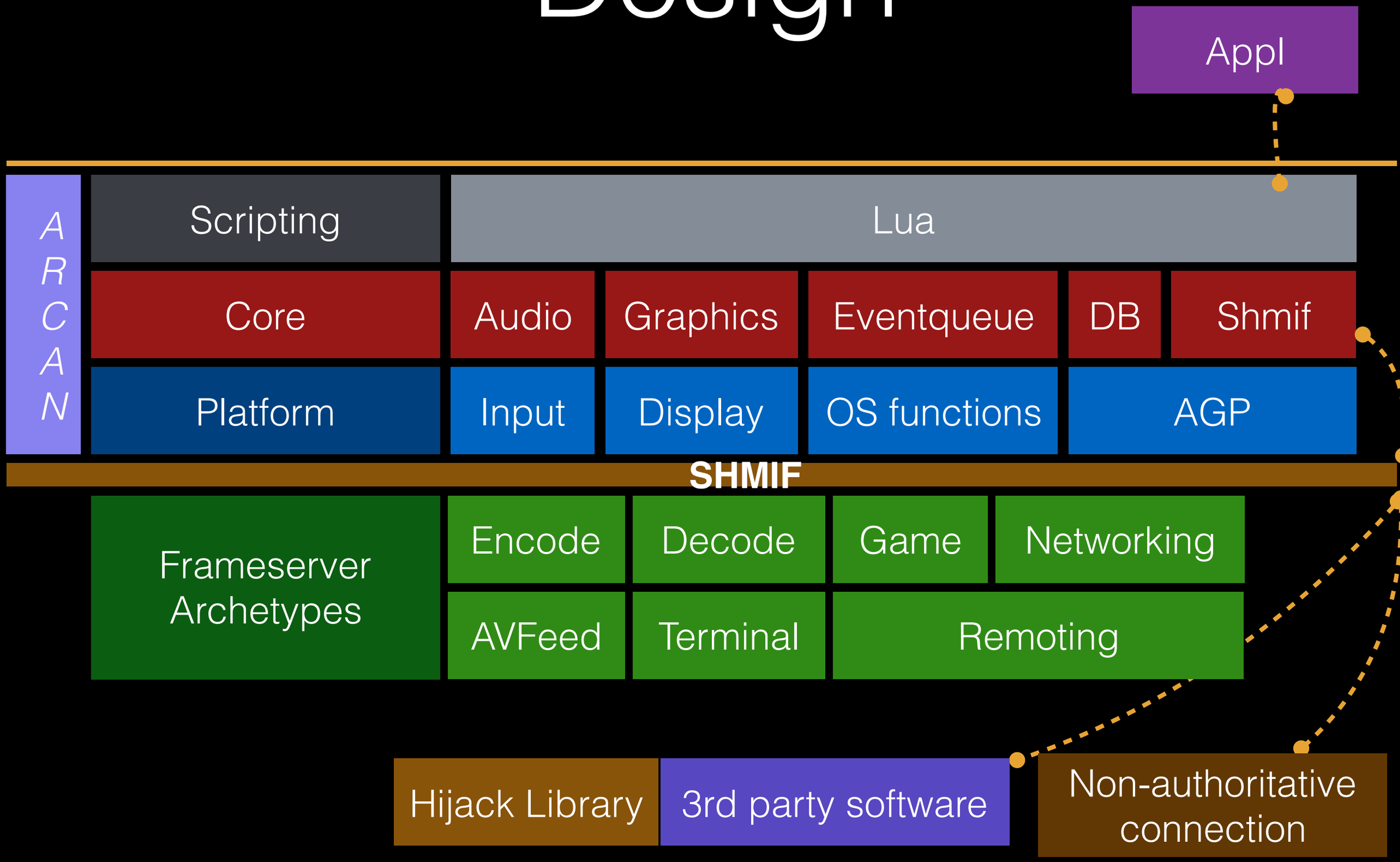
Arcan

Design and Development

Outline

- Design
  - Event Loop
  - Frameservers
  - Namespaces
  - Shmif
  - LWA
  - Threat Model
- Development
  - Event Loop
  - Frameservers
  - Shmif
  - Threat Model
  - Principles, Current State

Design


Appl
ARCAN
Scripting
Lua
Core
Audio
Graphics
Eventqueue
DB
Shmif
Platform
Input
Display
OS functions
AGP
SHMIF
Frameserver Archetypes
Encode
Decode
Game
Networking
AVFeed
Terminal
Remoting
Hijack Library
3rd party software
Non-authoritative connection

Main Loop
<Simplified>

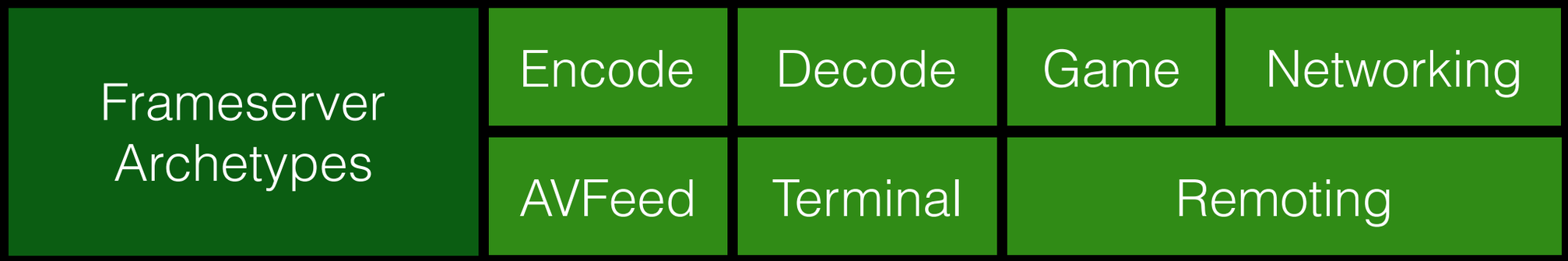
Init

- Sanity check environment
- Setup Platform Layer
- Map Namespaces, Database
- Configure Lua, fallback recovery points, Appl

Loop

- Check timing:
  - [~monotonic] update logical data model
  - [~heuristic] sample data model, update visual state
- Flush and filter event queues into VM handlers
- Sleep until external triggers or heuristic timeout
  - platform input devices, output display synch
  - frameserver data

Frameservers



- Separated producer/consumer processes
- Engine can act authoritatively, i.e. kill / control state with minimised risk for cascade or corruption
- Archetype implies specialised behaviour / response to possible shmif events
- Sandboxing / Policy behaviour per archetype
- *Trivial* to swap out the default implementation for one/several archetypes, with custom set

Frameserver Archetype

Decode

- Arbitrary data stream Input as descriptor, uri or path
- Outputs decoded A/V representation (best effort)
- Metadata for playback status, alternate streams, overlays
- Controls for seeking, stream selection

Default dependency: libvlc

Frameserver Archetype

Encode

- Primary segment type: ‘output’ (arcan. → frameserver)
- For streaming/ non-interactive/ lossy output encoding
  - Soon: secondary (Fsrv → Arc) segment for lossy abstract interpretation
    - Examples: Voice-Synthesis, OCR
- *Slightly* abused for remote desktop server behaviour
  (due to the client-interaction / authentication needs)

Default dependency: ffmpeg [optional: tesseract, libvncserver]

Frameserver Archetype

Terminal

- Or ‘rather’ interface to a class of applications signified by:
  - textual input from keyboard devices or streams
  - monospaced text output in a strict grid layout
- *Dynamic* privilege domain (think ‘login/su/sudo’)

  Possibly the most *useful* frameserver right now :-)

Default dependency: libtsm

Frameserver Archetype

Game

- Implements front-end side of libretro API (www.libretro.com)
  - Plugs ‘cores’ (primarily retro- style games and emulators)
- Good basis for testing/stressing:
  - A/V/input latency tradeoffs

(emulators typically output synthesised audio with weird sampling rates rather than mixing sample playback and streamed prerecorded output and therefore harder to “hide” buffering artifacts without latency or skipping)

- State snapshot / Management
- “Quirky” Input devices and dynamic input configuration
- Accelerated buffer passing, High CPU utilisation, …

Frameserver Archetype

Remoting

- Intended as [client role] access to different graphical desktop / computing environments
  - Requires interactive and event-driven A/V/I/ Clipboard/File translation/packing
  - Inherently ‘networked’
  - Default implementation lacking (poor choice of protocol)
    - Likely to be switched to SPICE or RDP

Default dependency: libvnclient

Frameserver Archetype

Networking

- Highly experimental (i.e. useless until ~0.6)
- Primary target: [local] service discovery and authenticated/encrypted communication across networked boundaries
- Application area: control-message / state passing between arcan instances across networked boundaries
  - e.g. live appl- migration, state redundancy

Frameserver Archetype

AVFeed

- “Dumb” / simple A/V provider
- Skeleton, Used for testing
- For quick’n’dirty interface wrapping 3rd party libraries / devices
- Can (mostly) be ignored

Lua

- Integrated VM (stuck at 5.1 / Luajit 2.0)
- Some added restrictions (no string eval or bytecode, no FFI, default I/O, system etc. functions dropped)
- Imperative API model, event driven from hooks (derived from applname_eventname()
- See Developer- intro slides for more information

Appl

- Namespaced collection of related scripts and resources
- layout like: ./ myappl1 / myappl1.lua
  (function myappl1 as entrypoint and event handler prefix)
- Three types: Main (running), Fallback (adopts external connections on fail)
  Monitor (optional, for debugging)

Minimal Terminal example:

```
function myappl1()
  term = launch_avfeed(“terminal”,
  function(src, statustbl) – eventhandler for fsrv -> arcan (see shmif/evmodel slide)
    if (statustbl.kind == “resized”) then
      resize_image(src, statustbl.width, statustbl.height);
  end);
  show_image(term); – starts as invisible
  target_displayhint(term, VRESW, VRESH); – tell process about display dimensions
end

function myappl1_input(iotbl)
– iotbl can cover analog / digital / device-plug / device-unplug events
 target_input(term, iotbl);
end
```

See also: dev. intro slides

Namespaces

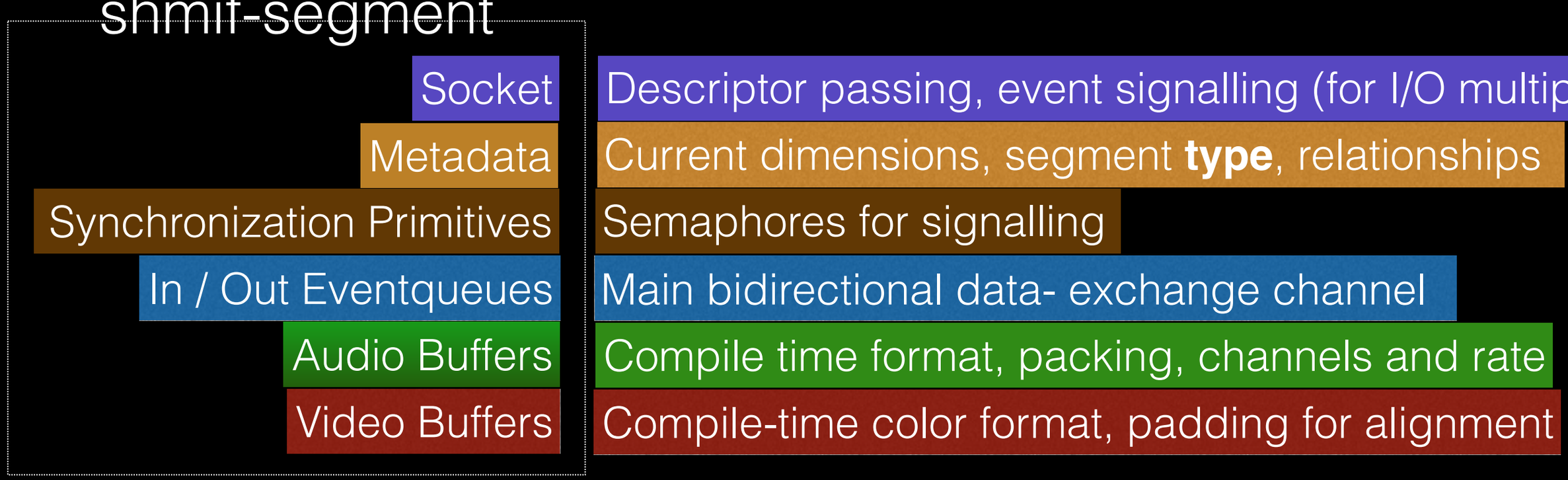
- Per Arcan- instance defined search paths
  - Restricts / filters search and access for script- resources and storage locations
- Examples:
  - APPLBASE - search space for appl loading and switching
  - STATEBASE - target state snapshots
  - APPLTEMP - writable, appl- generated content
  - (many others for FONTS, LOGGING, …)

Database

- Used for Application whitelist (execution model)
  - Target [binary + search path, format, base args, env]
    - Config [tied to target, additional base args]
- Includes library overrides (think LD_PRELOAD for shmif inject.)
- Constrains *launch_target* API calls
- key/value store both for target, target/config and for current appl.
- Will (0.6+) also cover sandboxing policies / state
- External tool (arcan_db) to manage

Shmif

(not a ‘public’ interface or protocol)



1 (guaranteed) segment per connection.
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

Shmif

- Segment Type dictates assumed use (e.g. HMD, interleave-odd/even, titlebar, icon, debug, accessibility, clipboard, drag'n'drop, popup, ...)
- Downsides:
  - Complex rules for switching between shared-memory and handle-passing video buffers (shm always available, buffer passing is privileged, intermittent and volatile)
  - Event-queue saturation ("**A**pplication **N**ot **R**esponding") management is terrible, but fixable
  - Tightly coupled with engine internals, no 'protocol' - built / updated in lock-step, shared struct ABI without serialization format.
  - Not all events are processed in order, some (e.g. analog axis motion, multiple displayhints / fonthints may merge)

Shmif Synchronization

- Data transfers are ‘signal’ operations on semaphores
  - (SHMIF_SIGVID | SHMIF_SIGAUD).
- “May” use accelerated buffers (zero-copy, …) when available
  - But controlled arcan- side and forced fallback to shm- only
- Multiple strategies (to handle latency, blocking and tearing tradeoffs)
- Semaphores + atomics + socket ‘ping-packet’ for block and I/O multiplexing
- Resize operation on segment (or subsegment) blocks until negotiated.

Shmif Event Model

(arcan → frameserver)

+ TARGET_IO struct namespace

TARGET_COMMAND_RESET - reset to initial state
TARGET_COMMAND_EXIT - connection terminated
TARGET_COMMAND_NEWSEGMENT - connection data for subsegments

TARGET_COMMAND_COREOPT - set initial Key / Value config entry
TARGET_COMMAND_SETIODEV - plug / unplug device mapping
[d]TARGET_COMMAND_DEVICE_NODE - switch device input / hw render

TARGET_COMMAND_PAUSE - connection / synch suspended
TARGET_COMMAND_UNPAUSE - follows PAUSE
TARGET_COMMAND_STEPFRAME - manual frame control (or CLOCKREQ callback)

TARGET_COMMAND_REQFAIL - previous subsegment request failed / was rejected
TARGET_COMMAND_BUFFER_FAIL - accelerated buffer passing rejected, fallback to shm- render

TARGET_COMMAND_DISPLAYHINT - segment display properties (dimensions, density)
[d]TARGET_COMMAND_FONTHINT - transfer fonts and metadata
TARGET_COMMAND_SEEKTIME - seek in datastream

[d]TARGET_COMMAND_STORE - serialize internal state
[d]TARGET_COMMAND_RESTORE - deseralize internal state
[d]TARGET_COMMAND_BCHUNK_IN - binary data blob in
[d]TARGET_COMMAND_BCHUNK_OUT - binary data blob out
TARGET_COMMAND_MESSAGE - archetype specific short message (multipart)

TARGET_COMMAND_GEOHINT - location, orientation, language ...
TARGET_COMMAND_ATTENUATE - volume hint (for connections not using audio- part of shmif)

TARGET_COMMAND_FRAMESKIP - switch heuristic for adv. synch
TARGET_COMMAND_AUDDELAY - increment or decrement audio playback timing

EVENT_EXTERNAL_STREAMSET - switch sub-datastream (decode archetype)
TARGET_COMMAND_SEEKCONTENT - content panning (scrolling)
TARGET_COMMAND_GRAPHMODE - alternate rendering modes (archetype specific)

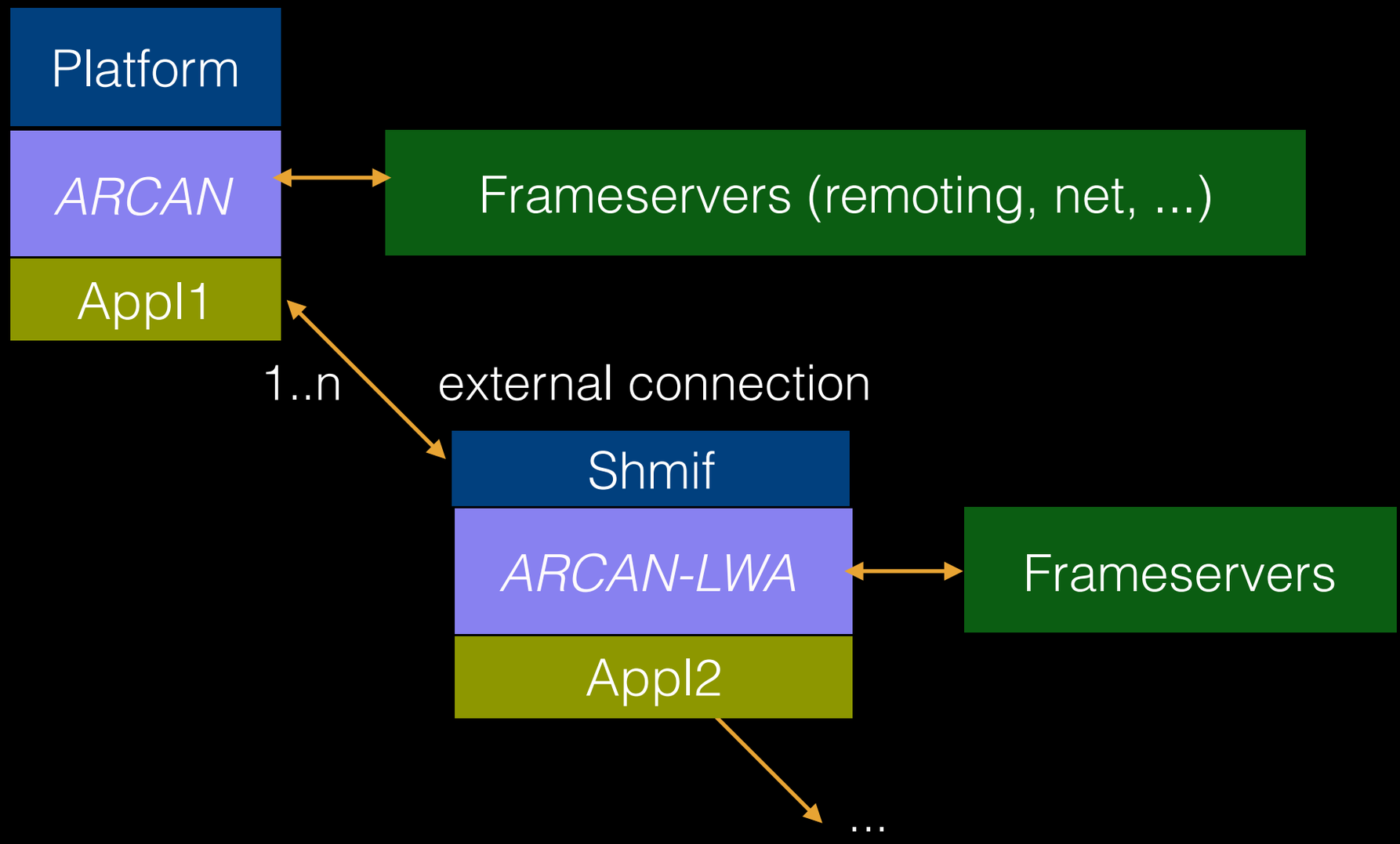
Shmif Event Model

(frameserver → arcan)

EVENT_EXTERNAL_MESSAGE - Archetype or segment type specific [multipart] short UTF-8 message
EVENT_EXTERNAL_IDENT - Content Identification
EVENT_EXTERNAL_STREAMSTATUS - Streaming playback, position
EVENT_EXTERNAL_STREAMINFO - Alternate data stream notification
EVENT_EXTERNAL_FAILURE - State serialization failure
EVENT_EXTERNAL_STATESIZE - Estimate current state block size (0- disabled)
EVENT_EXTERNAL_CURSORHINT - Hint at cursor visual state when on surface
EVENT_EXTERNAL_LABELHINT - Hint digital or analog input data tag
EVENT_EXTERNAL_COREOPT - Key/Val configuration option
EVENT_EXTERNAL_SEGREQ - Request additional subsegment
EVENT_EXTERNAL_KEYINPUT - Request limited keyboard input (remoting)
EVENT_EXTERNAL_CURSORINPUT - Request limited mouse cursor input (remoting)
EVENT_EXTERNAL_REGISTER - [once] specify + sign UUID or hint at archetype
EVENT_EXTERNAL_FLUSHAUD - Request that pending audio buffers be discarded
EVENT_EXTERNAL_VIEWPORT - Reduce active surface use or map multiple views on same surface
EVENT_EXTERNAL_CONTENT - State indicator for content (scrollbars)
EVENT_EXTERNAL_CLOCKREQ - Request a periodic or one-fire timer
EVENT_EXTERNAL_ALERT - UI Notification hint
(*)EVENT_EXTERNAL_BUFFERSTREAM - Handled internally, used for accelerated buffer status and delivery timing
(*)EVENT_EXTERNAL_FRAMESTATUS

LWA

- Specialized Build that uses shmif as A/V/I/O

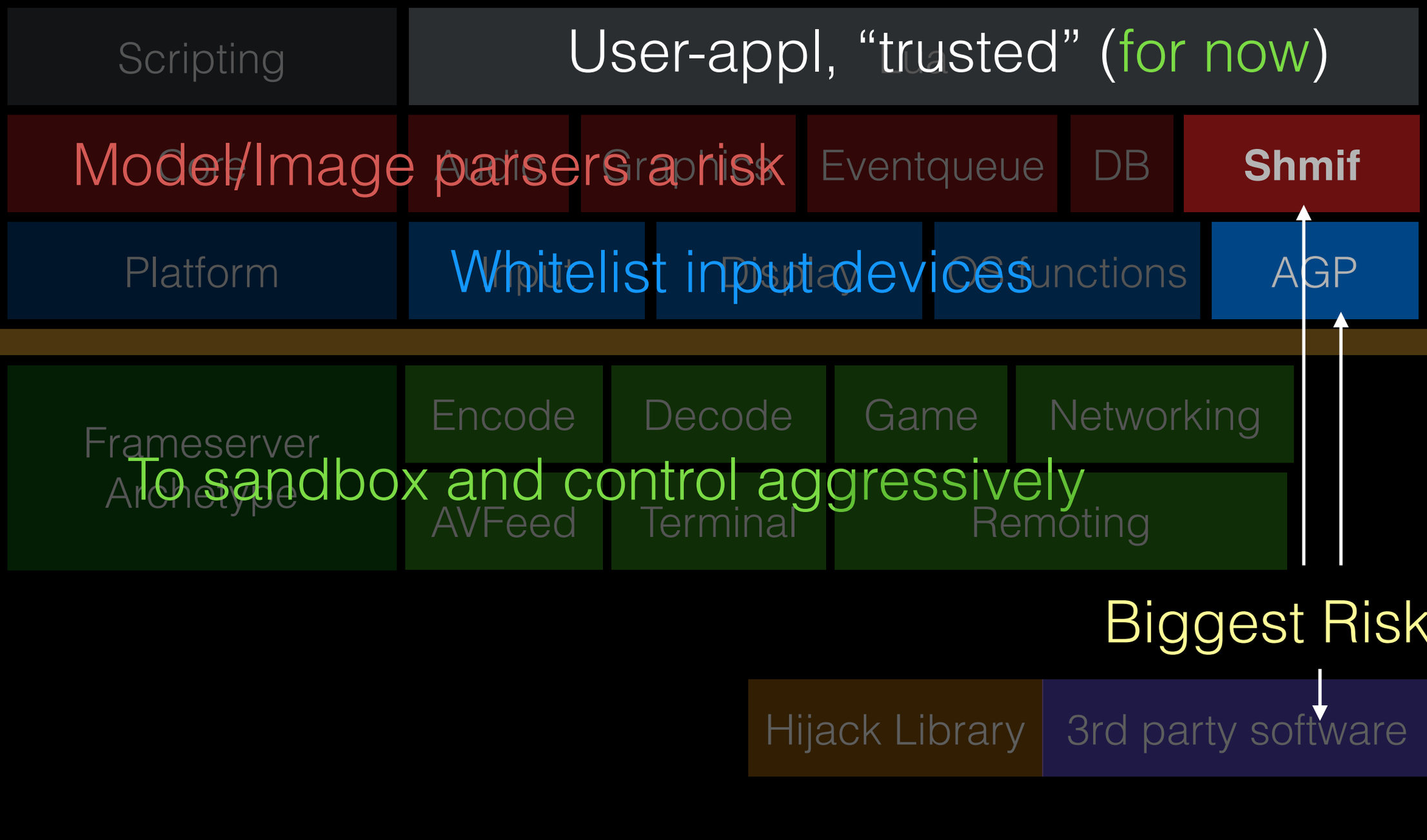


- But possible latency/... increase with level

Security Model

- Appl dynamically define permitted interaction (e.g. target_input(dstid, itbl)). Control should flow *from* user to appl to arcan to external. Every step is a *possible* reduction of privilege.
  - Includes output segments (clipboard-paste, video-recording/screen readers) using define_recordtarget(dstid, {set of audio sources}, {set of video sources}), allows fine-grained controlled sharing.
- Frameserver archetype *dictates* sandboxing model (*still* in its infancy), “basic” requirements: CloudABI syscalls + seccmp-bpf/capsicum/obsd-pledge + fuse profile
  - Based on the assumption that any external connection can / should be contained in Sandbox and/or VM.
  - *Without* sacrificing user-expected features.
  - e.g. “Skype” should have transparent/user-regulated(overridable) access to A/V feeds, but **not** be able to discern, grasp or request /dev/video0 vs. goatse.mkv

Threat Model



Governing Principles

- No-surprises
  - Safe, Passive, Defaults
  - Running *appl* dictates behaviour
    - And user specifies appl
  - All external connections are explicit
  - Don’t try to be clever, provide mechanisms for the user, make them obvious and accessible - not automated ‘default’ 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, protecting business interests, building empires...)
    - Any interface that provides a perceivable truth should also be able to provide corresponding lies and half-truths - *this is the virtualization ideal*
    - Application 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)
    - User- placed trust in an application is a dynamic (context-sensitive) property, sandboxing controls *should* reflect this.

Governing Principles

- Be Conservative
  - “Modern” is appeal-to-authority nonsense
    - Comes at the cost of exclusion of those that reject the “authority”
  - Define the features needed, articulate well in advance, *then* commit to them
    - Feature/scope creep leads to ‘solving’ general problems that does not fit the problem space of any single stakeholder
    - The Web-browser is the final stage 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”
  - Steer away from Funky IPC and Turing Complete or Context Sensitive Parsers
- but - pragmatism, **not** ideology

* shoot to kill

Governing Principles

- Stay Pragmatic
  - Minimize dependencies, ‘Done’ when no more lines of code can be removed
  - CM work grow with dependency-set, you replace ‘bugs you are guilty of’ with ‘bugs other people decide’
    - Never rely so hard on an external solution that you can’t pack your bags and leave
- Stay portable -- commit to the chosen standards, avoid fancy in-house features
  - 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 tradeoff selections in both design and implementation

Technical Points and Tradeoffs

- Core: 100% C (ISO 8998:2011) style, due to the requirement of minimising runtime and dependencies. This is a simplicity versus performance tradeoff.

- Primarily single-threaded with domain specific or process separated concurrency. This is a debugability versus performance tradeoff.

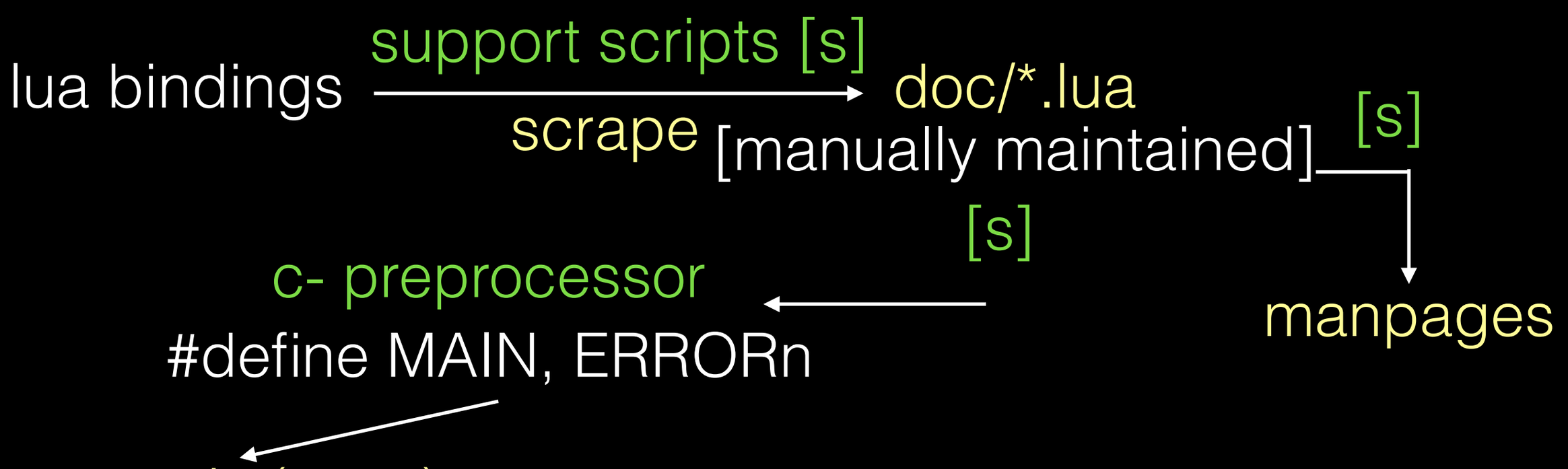
- Engine configuration is build-time static with embedded tag (platform, git revision etc.). This is a simplicity versus flexibility tradeoff.

- Lua VM configuration is *rather* restrictive to *avoid* dependency creep. Extend for individual use-cases with system_load and .so:s

Debugging / Stability

- Scripting layer: fail early, often and hard.
  - data model snapshot as .lua script stored in debug data namespace (see system_snapshot call)
    - possible cause in stdout output (and in dump)
- Monitor : periodic snapshot serialization to another arcan instance over a pipe that the _appl can access and draw.
- Fall-back appl: “on crash, rebuild env, keep external connections and expose to new appl in _adopt callback”

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 test setup, contributions page @ wiki is up to date.