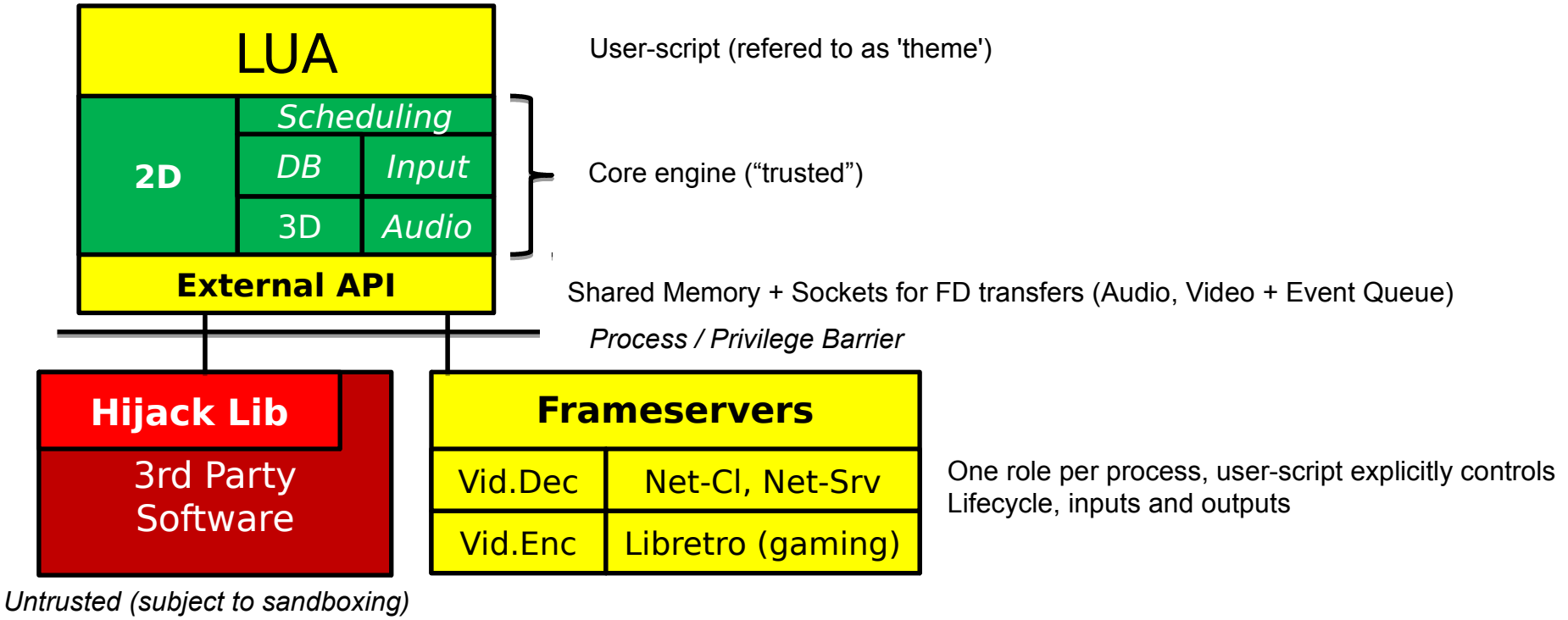
Arcan

Toolsuite for real-time visualization

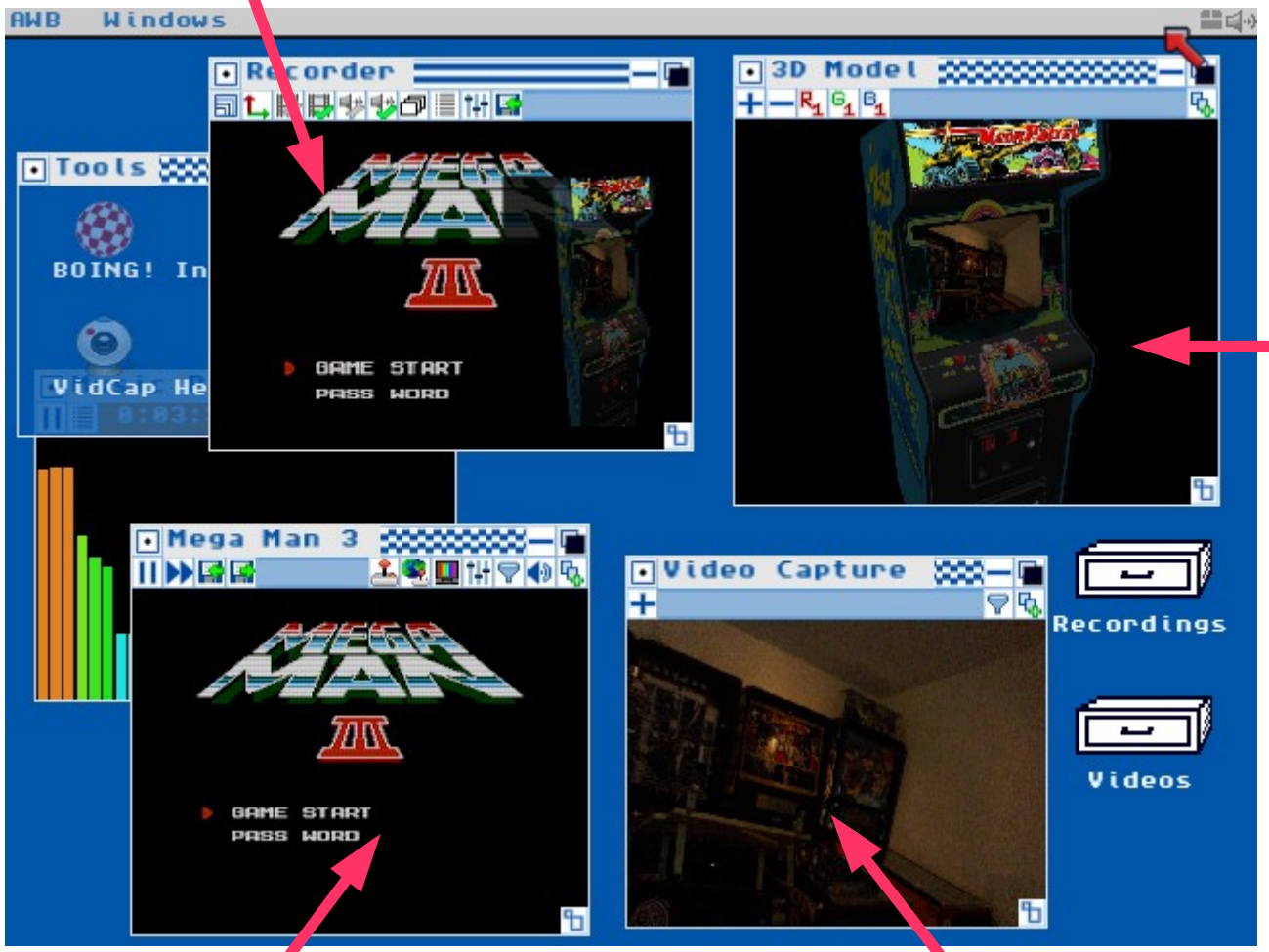
- Personal Sandbox for ideas and experiments since ~2003
- Open Source since dec-jan 2012 (integration, refactoring, documentation)
- High-level APIs for 2D/3D video, audio, networking, database, device input
- Key goal: balancing security, performance and debugging
- For FreeBSD/Linux/Windows



Example Theme

AWB (Reasonably competent desktop environment)

Video Encoding / Streaming



Webcam feed mapped
To 3D model

Libretro Emulator Core

Webcam Capture Feed

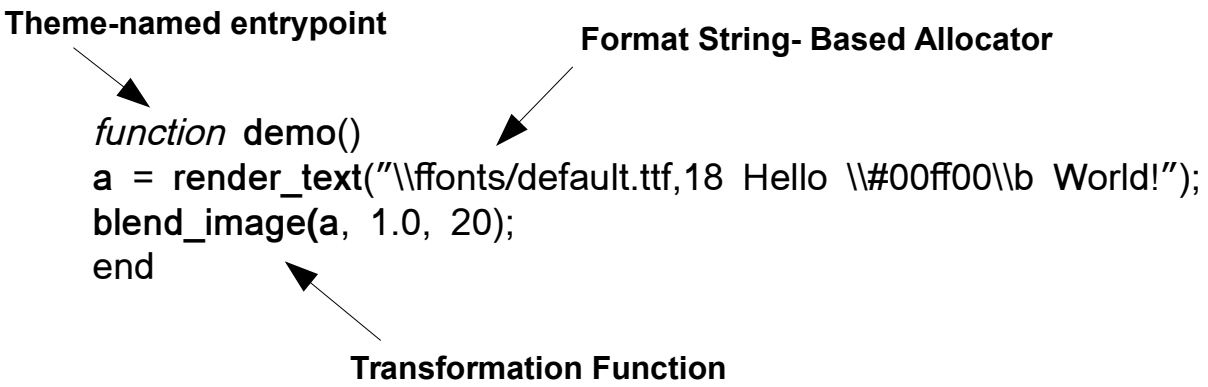
Scripting Model

Why LUA?

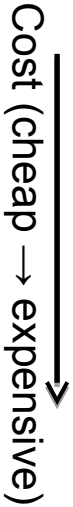
- Easy and cheap to embed
- Strong separation between framework and language
- Good dynamic sandboxing opportunities
- No Forced OO, *Execution-flow more important than Data-model!*
- Single Threaded (which is a good thing!)
- Above points combined, ideal for debugging

Approach

- Event driven (key events: *clock_pulse*, *device_input*, *video frame-delivered* + asynchronous event-handlers connected to video objects)
- Two-tiered resource name-space (theme-specific (**R/W**) shared (**RO**)



Life-cycle



Allocator functions:
null_surface
fill_surface
color_surface
load_image
load_image_asynch
render_text
launch_target
load_movie

BADID

VID

Key Attributes:
Opacity (0..1), def: 0
Position
Size
Orientation

Transformation functions:
move_image(x, y, *dt)
show_image(*dt)
hide_image(*dt)
nudge_image(dx, dy, *dt)
scale_image(dfw, dfh, *dt)
resize_image(w, h, *dt)
blend_image(opa, *dt)
rotate_image(ang, *dt)

*dt : deadline – transformation will be scheduled and stretched across (dt) engine-logical time steps, **engine interpolates!**

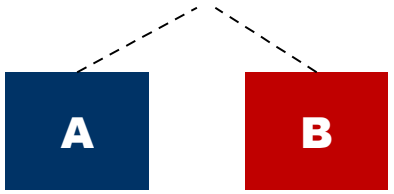
Every transformation group can be chained (multiple calls stack):

```
function demo()
    a = load_image("images/icons/ok.png");
    show_image(a);
    resize_image(a, 1, 1);
    move_image(100, 100, 80);
    move_image(0, 0, 80);
    rotate_image(120, 160);
    resize_image(128, 128, 160);
end
```

These chains can be copied, transformed, translated, reset etc.
Also used for collision/Intersection detection e.g. “in n steps, will a intersect b”?

Slightly more advanced:

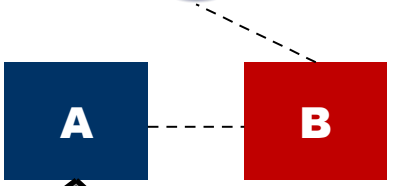
1. All visual entities implicitly inherit properties from **WORLDID**
2. Transformation chains are defined in local space and then resolved into world space.
3. Entities and their respective chains can be linked into more advanced hierarchies.




```
function demo()
   b = fill_surface(64, 64, 255, 0, 0);
   a = fill_surface(64, 64, 0, 0, 255);
   show_image({a, b});
   move_image(b, 128, 0);
   move_image(WORLDID, 80, 0);
end
```

2.

link_image(a, b);



1. Linked objects gets hierarchically defined properties (with associated cost in rendering etc.)

2. Inherited properties can be controlled fine-grained through masking (e.g. inherit position but not orientation)

3.

```
c = instance_image(a);
d = instance_image(a);
```

1. Clones are lightweight objects explicitly tied to a parent (particle systems etc.) where only a subset of attributes can be overridden (e.g. active shader)

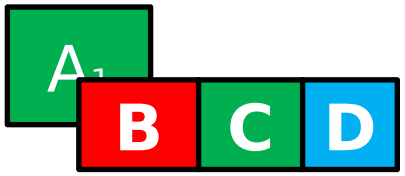
Even more advanced:

Context Stack

1. Contexts
The current rendering context is part of a stack.
This means that all visible objects can be *pushed* (stored for later reconstriction) or *poped* (for mass-deallocation)
This can be used for memory management, optimizing subsets, rendering target), sandboxing, ...

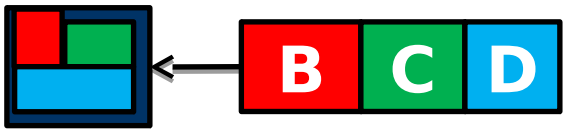
Special case: launch_target(id, LAUNCH_EXTERNAL)
1. current context gets pushed
2. engine subsystems deallocated, external program launched
3. external program termination, everything reallocated

Framesets



2. Entities can be set to share storage, for the primary frame or compose a set of frames using several other entities (framesets).
This is useful for:
animation (auto-cycling every n frames or n ticks)
History-frames (where you want access to several frames from a frameserver)
Multitexturing (for 3d objects and for advanced shaders when combined with history frames).

Rendertargets

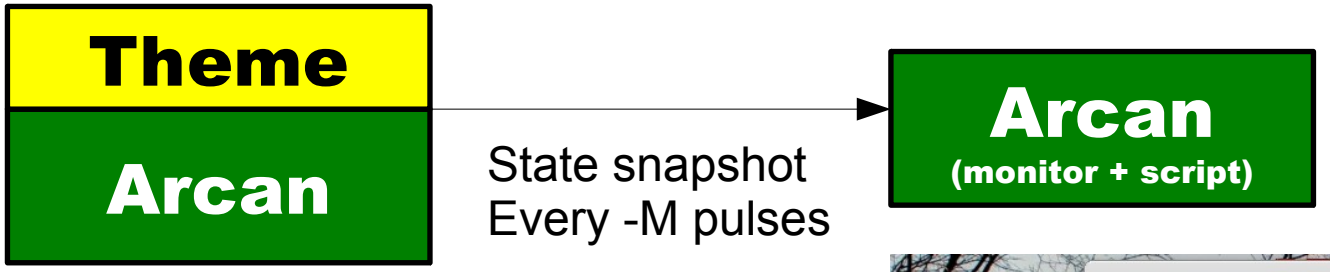


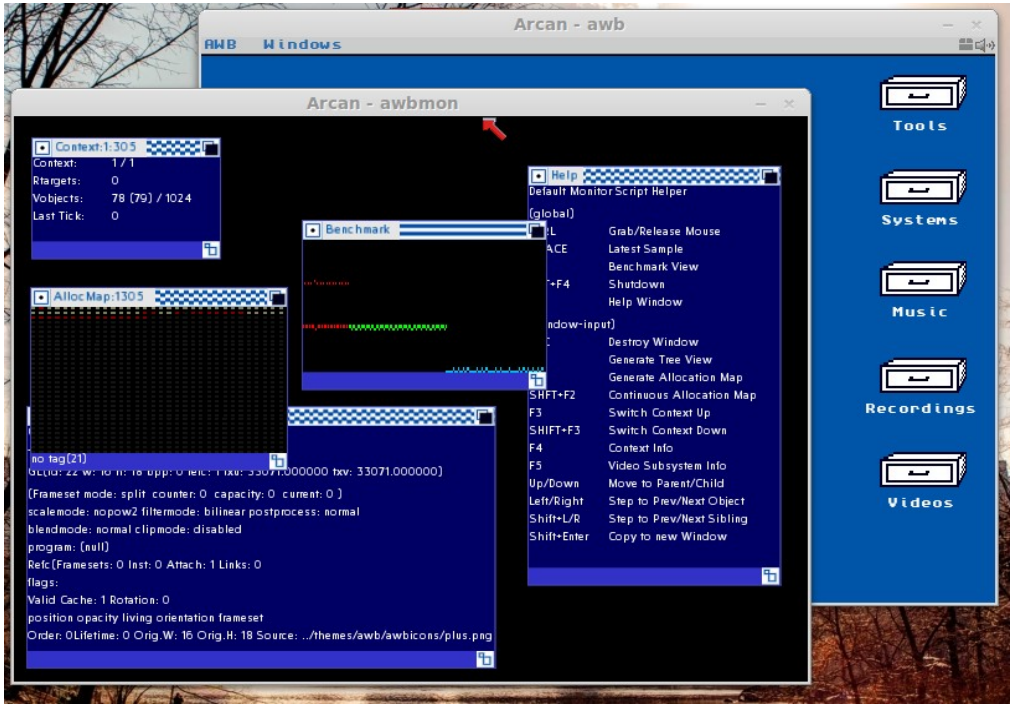
3. Rendertargets is a separate renderpath in the active context (there's a standard out and a fixed number of others) where the output is saved in the storage of another entity (commonly called RTT, render to texture).
This is useful for:
1. Multipass rendering effects (blurs, stereoscopic rendering, shadows, reflection)
2. When combined with readbacks; video streaming, recording etc.

Details

- 170+ functions (documentation, examples and testcase coverage project underway, soon to be completed).
- Monitoring Mode:

Example> ./arcan -M **100** -O **awbmon/awbmon.lua** awb





Also works for crash dump analysis!

Exercise

"Take a video capture device, scale to half display and draw a copy with a black/white shader. Combine these into a render target that reads back every two videoframes (60fps input => 30fps output), into a encoding frameserver and record as h264 @ 30fps in a MKV container"

Solution:

```
function demo()
	local source = "demo.avi";
	local camsource = "capture:device=0:width=320:height=240";
	local vid, aid = load_movie(source, FRAMESERVER_NOLOOP,
		function(source, status)
			play_movie(source);
			resize_image(source, 320, 480);
		end);
	show_image(vid);
	local bw = instance_image(vid);
	local shid = build_shader(nil, bwshader, "bwshader");
	image_shader(bw, shid);
	resize_image(bw, 320, 480);
	move_image(bw, 320, 0);
	local dst = fill_surface(640, 480, 0, 0, 0, 640, 480);
	show_image({bw, dst});
	define_recordtarget(dst, "testout.mkv",
	"container=mkv:vcodec=H264:fps=30:vpreset=8:noaudio",
	{bw, vid}, {}, RENDERTARGET_DETACH,
	RENDERTARGET_NOSCALE, -2);
end
```

```
local bwshader = [[
	varying vec2 texco;
	uniform sampler2D map_diffuse;

	void main()
	{
		vec4 col = texture2D(map_diffuse, texco);
		float f = (col.r + col.g + col.b) / 3.0;
		gl_FragColor = vec4(f, f, f, 1.0);
	}
]];
```

On the Horizon...

- Better Hijack libraries (xlib emulation)
- Sandboxing FUSE driver (target → chroot, dynamic map/unmap/simulate* resources)
- 3D pipeline improvements (rather basic atm.) Focus on stereoscopic rendering (HMDs etc.)
- Stand-alone (theme :- window manager, console + rdesktop/vnc implementation as frameservers, hijack libs for rest).