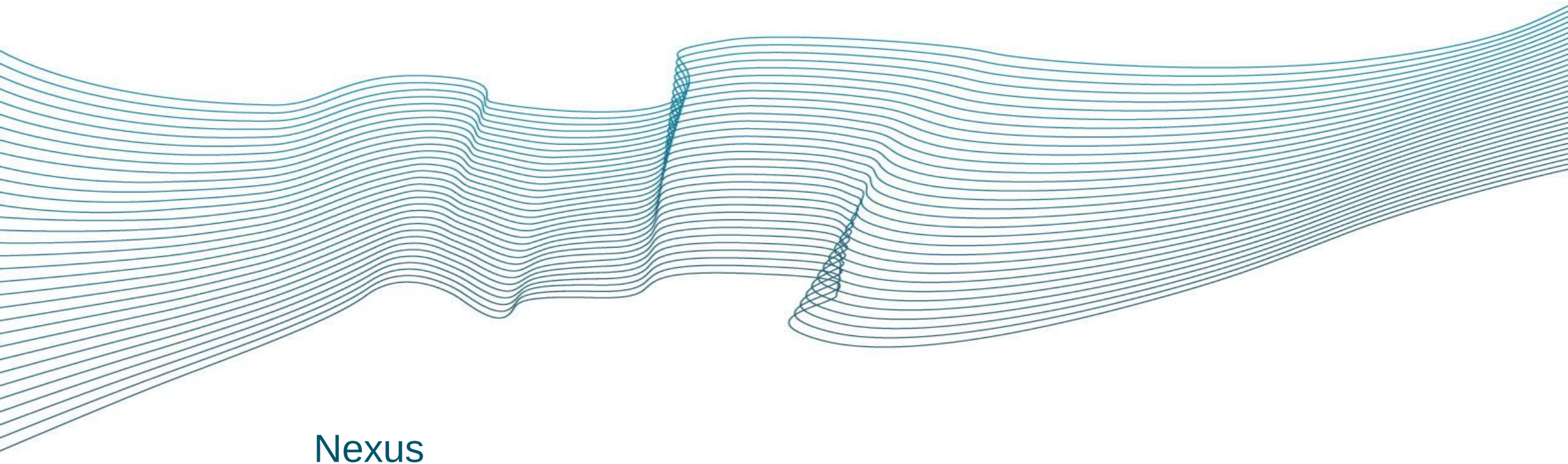
NEXUS OVERVIEW

SEPTEMBER 25, 2012

David Erickson
Associate Technical Director

PRESENTATION OVERVIEW

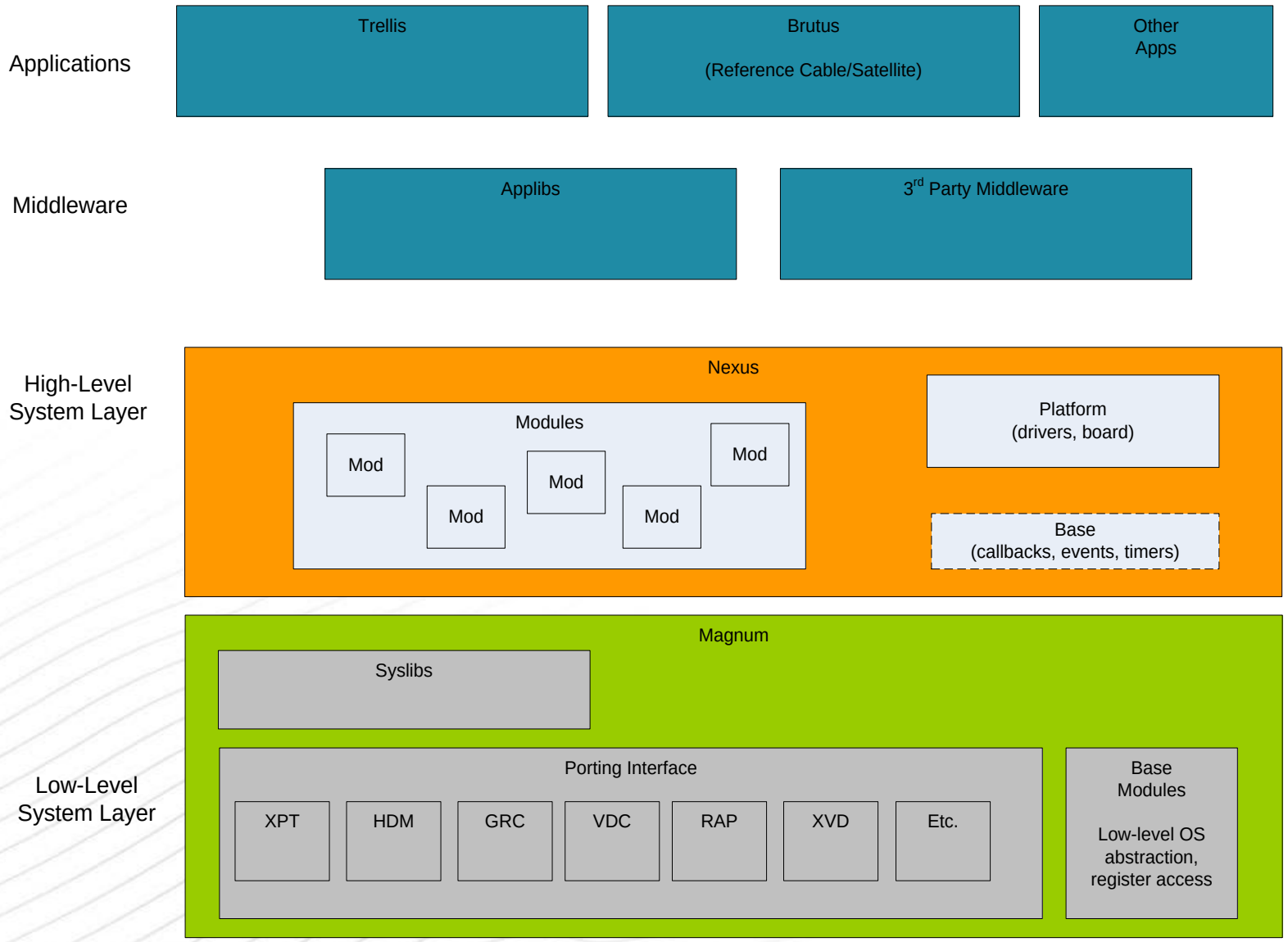
- **Reference Software Stack**
  - How Nexus fits into Broadcom's overall reference software stack
- **Nexus Architecture**
  - Synchronization, interfaces, drivers, Linux user/kernel mode, etc.
- **Basic Usage**
  - Audio/video decode, graphics, DVR, etc.
- **Advanced Usage**
  - Transcode, 3DTV, SVC/MVC, multi-process, power management, etc.
  - AppLibs integration including DirectFB, WebKit, DLNA, Flash



Nexus

REFERENCE SOFTWARE STACK

REFERENCE SOFTWARE STACK

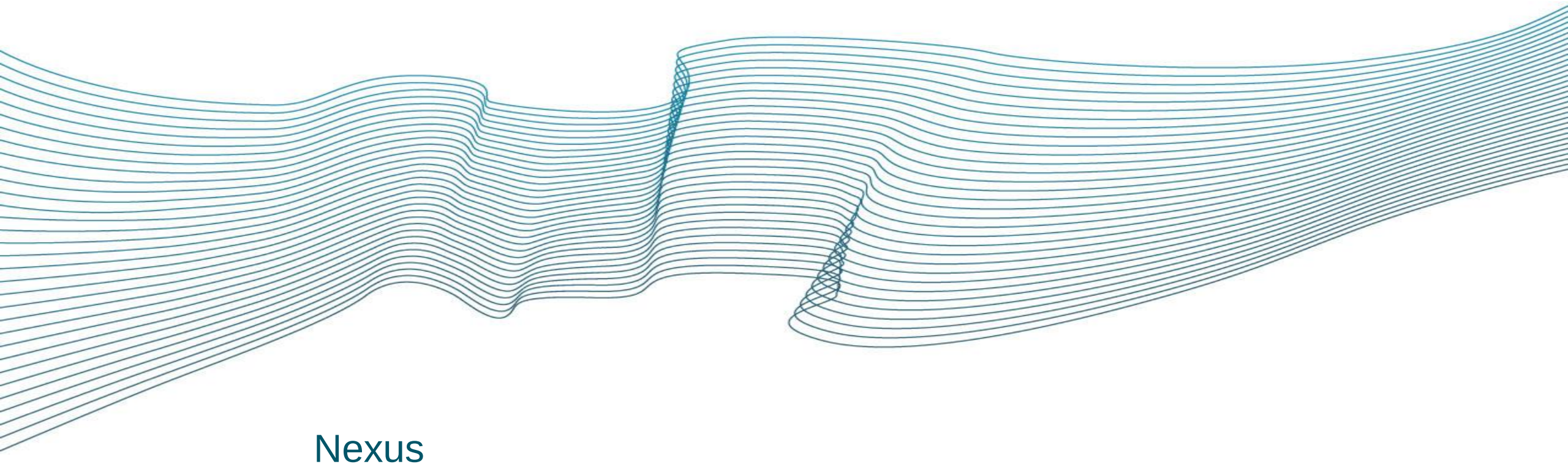


WHAT IS NEXUS?

Nexus is a modular, high-level API for Broadcom set-top boxes

- **High-Level API**
  - Easily map customer software stack to Broadcom software and hardware
  - Quickly learn API because of its usage-oriented design
  - Encapsulated internal complexity
  - Leverage years of development and debug on millions of systems deployed around the world

- **Modular**
  - Majority of code is reused across multiple set-top chips
  - New code can be added in a modular fashion
  - Software modules can be updated/customized without rippling through entire code base



Nexus
ARCHITECTURE

NEXUS ARCHITECTURE

Customers code to Nexus in their app or HAL

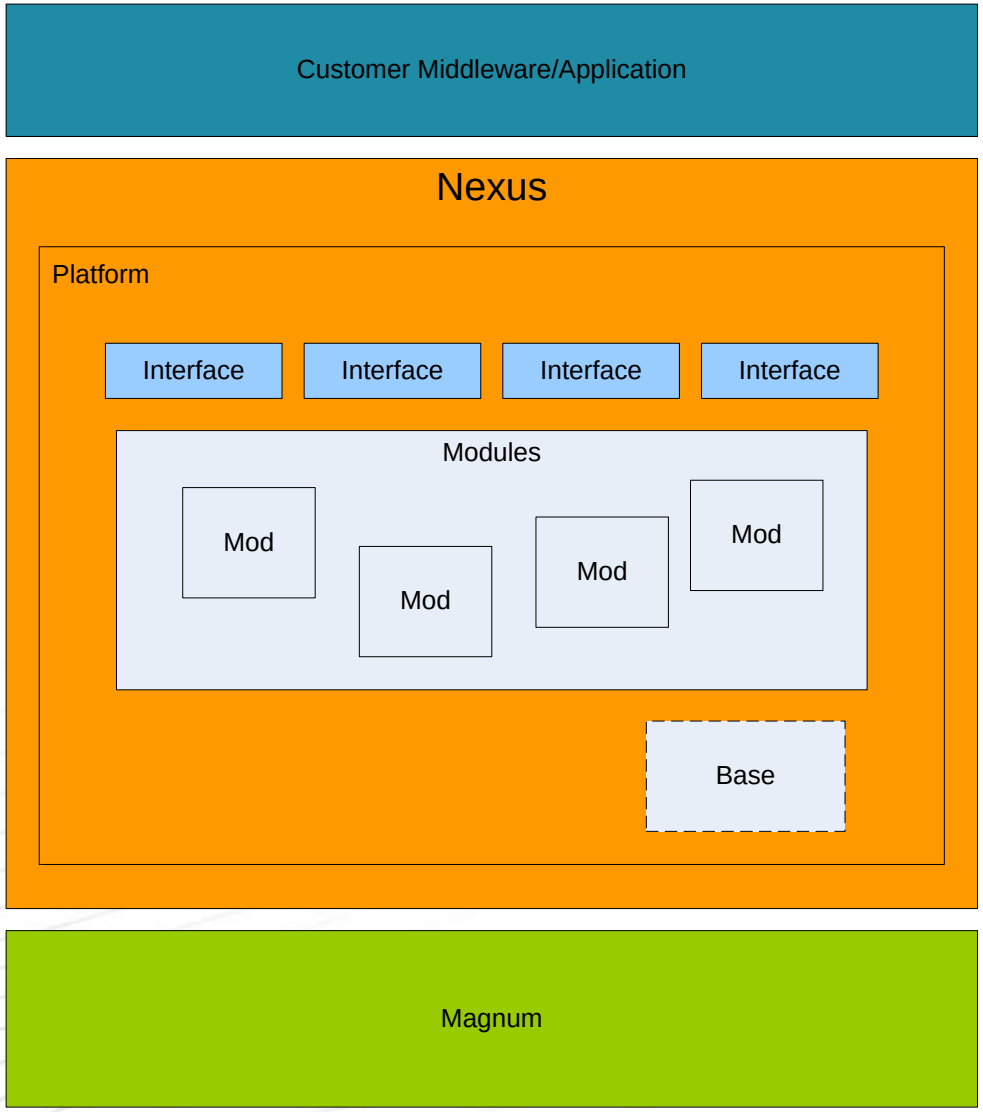
Nexus Platform integrates modules, contains board, OS and customer specific code

Nexus Interfaces are the user's view of Nexus

Nexus Modules contains implementation. They can be replaced or extended.

Nexus Base provides OS services required by modules

Magnum provides interface to HW



NEXUS ARCHITECTURE

- **Key components**
  - Synchronization
  - Thunks
  - Modules and Interfaces
  - Callbacks
  - Coding convention
  - Separation of chip-specific and customer/board-specific code in Nexus Platform
  - OS abstraction
  - Linux user-mode/kernel-mode implementation
- **See nexus/docs/Nexus_Architecture.pdf for details**

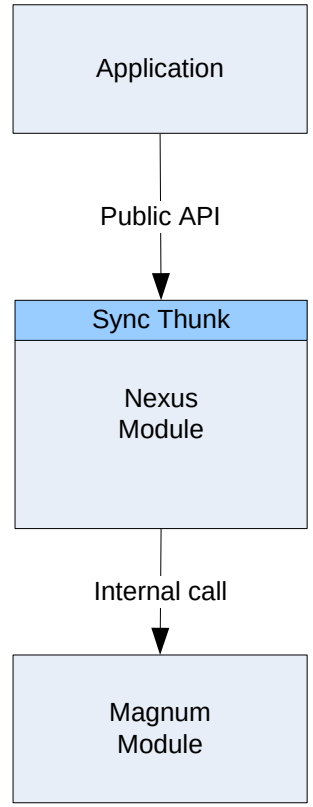
SYNCHRONIZATION

- **Automatic synchronization of the public API achieved through Perl-generated thunk layer.**
  - Avoids re-entrancy and synchronization programming errors
- **Modular implementation with strict hierarchy**
  - Avoids deadlock
  - Prevents fast modules from waiting on slow modules
- **Callbacks delivered through a set of dedicated callback threads**
  - Avoids re-entrancy and deadlock
  - Allows fast modules (e.g. transport) to not wait on slow modules (e.g. frontend)
- **ISR calls and callbacks across modules allow immediate execution of time-sensitive code**
- **Manual synchronization between modules using a private API allows complex and efficient interconnections**

PERL-GENERATED THUNK FOR SYNC & MULTI-PROCESS

Example synchronization thunk code:

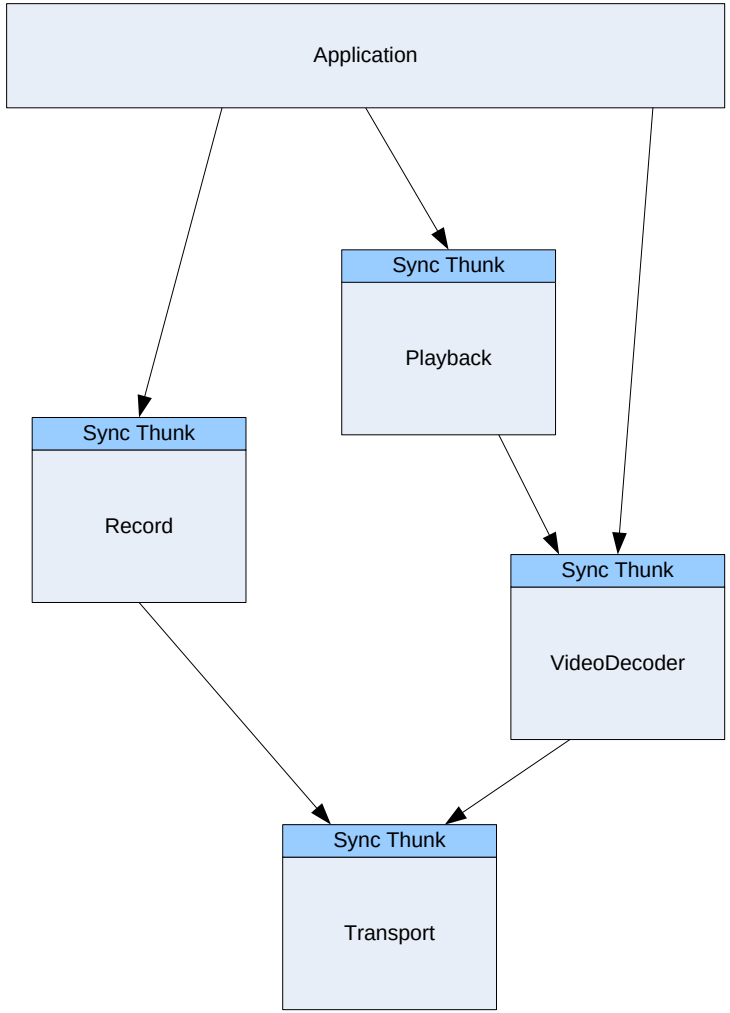
```
NEXUS_Error NEXUS_Message_GetStatus(hndl, pSettings)
{
    NEXUS_Error rc;
    NEXUS_LockModule();
    rc = NEXUS_Message_GetStatus_impl(hndl, pSettings);
    NEXUS_UnlockModule();
    return rc;
}
```



- "Thunking" gives Nexus elements of a 4GL language without the overhead.
- Thunk for ioctls to kernel mode driver
- Thunk for sockets to user mode server
- Thunk for enforcing power management standby mode

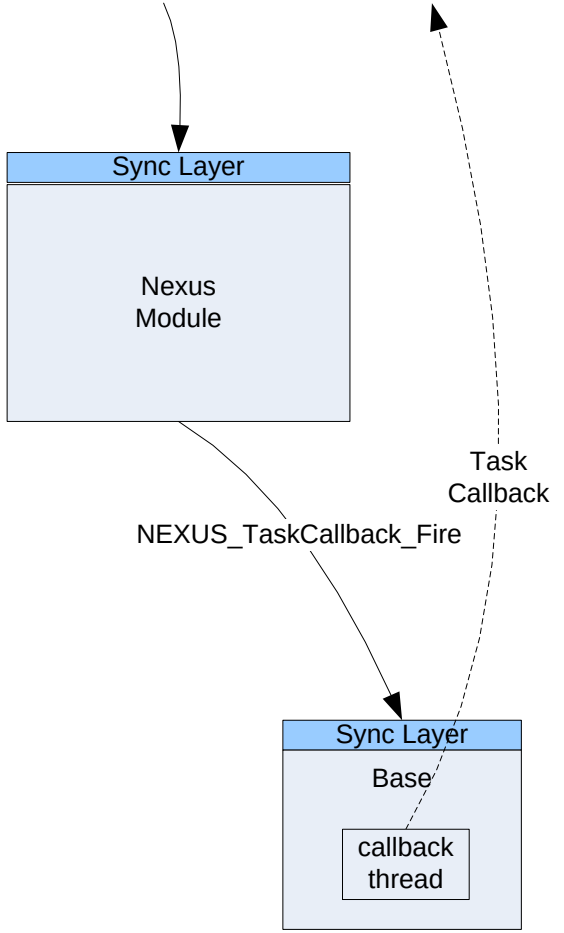
MODULES AND INTERFACES

- **How can the system be safe and efficient?**
  - If each module automatically locks, how do we avoid deadlocks?
  - If each module automatically locks, how do we avoid fast modules waiting on slow modules?
- **Modules arranged in a tree**
  - Fast modules can't call slow modules, making the system efficient.
  - Deadlock is impossible, making the system safe.
- **Modules distributed between client & server**
- **The application's view of Nexus is interconnected Interfaces**
  - The application doesn't have to know the module tree. It sees a flat collection of interfaces.
  - Nexus encapsulates internal connections. This makes your application easier to code.



CALLBACKS

- **How can we safely call back into the application?**
  - If we don't unlock, we risk deadlock.
  - If we unlock, we have to code for re-entrancy in every Nexus module.
- **Nexus Base provides prioritized asynchronous callbacks**
  - Nexus Base runs one thread per module priority. Keeps slow and fast modules segregated.
  - Standard callback mechanism works across kernel and process boundaries.
  - One context switch is required going from kernel → user mode. Nexus' implementation adds no more.
  - Nexus can fire callback directly from ISR context. Again, only one context switch.
  - Application can make almost any call from inside a callback. No re-entrancy in Nexus modules.
  - Nexus close functions automatically synchronize with an interfaces callbacks. This eliminates very tricky race conditions.



CODING CONVENTION

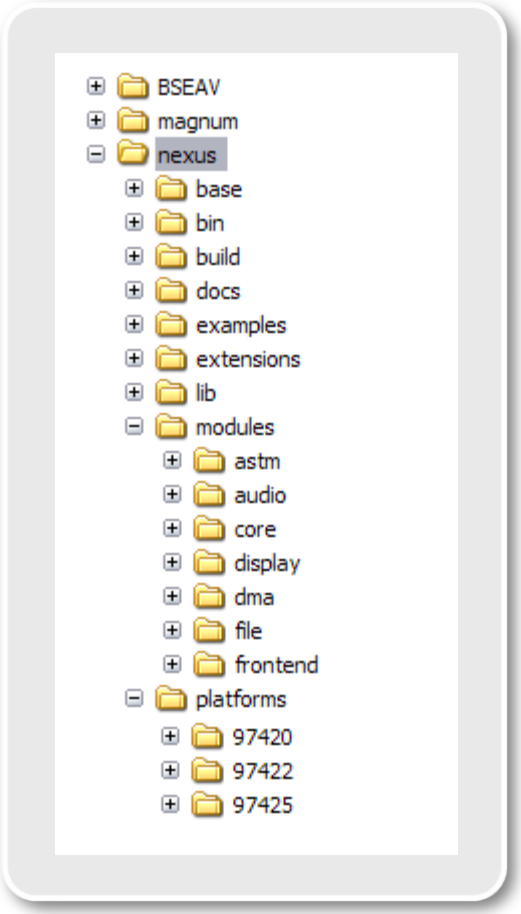
- **Consistent naming convention**
  - “NEXUS_” prefix establishes a namespace
  - CamelCase for consistency
  - Interface name included in every member of the interface (e.g. NEXUS_VideoDecoder_Open)
- **Interface patterns**
  - Open/Close, Create/Destroy
  - GetDefaultSettings/GetSettings/SetSettings
  - GetBuffer/ReadComplete
  - AddInput/RemoveInput
- **Restrictions**
  - No function pointers (synchronization)
  - No pointers to structs inside structs (no deep copy)
- **Advantages**
  - Easy to understand
  - Future proof API techniques (increased chances of backward compatibility on re-compile)
  - Thunkable

```
NEXUS_SurfaceHandle NEXUS_Surface_Crea
    const NEXUS_SurfaceCreateSettings
    );

void NEXUS_Surface_Destroy(
    NEXUS_SurfaceHandle surface
    );

NEXUS_Error NEXUS_Surface_SetSettings(
    NEXUS_SurfaceHandle surface,
    const NEXUS_SurfaceSettings *pSett
    );

void NEXUS_Surface_GetSettings(
    NEXUS_SurfaceHandle surface,
```

See “Coding Conventions” in nexus/docs/Nexus_Development.pdf

NEXUS PLATFORM

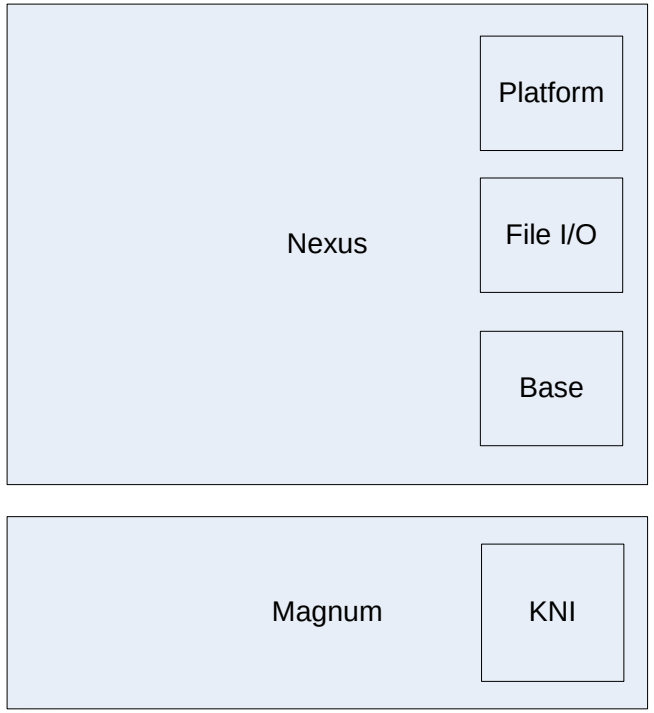
- **Nexus Platform allows customer- and board-specific code to be separated from chip-specific code**
  - Maximizes code reuse
  - Allows full customization
- **Typical features include**
  - OS driver code
  - Pin-muxing
  - L1 interrupt mapping from OS
  - Build system
  - Frontend configuration
  - DAC/ADC mapping



OS ABSTRACTION

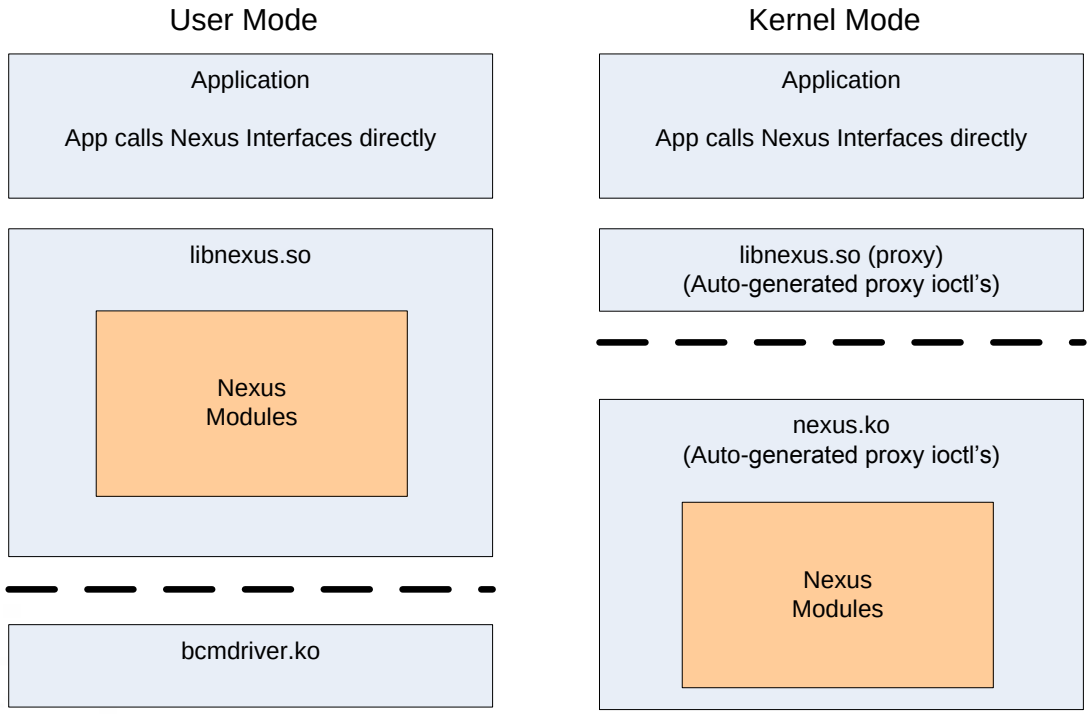
- **OS abstraction provided in layers**
  - Only provide the minimum required at each point for the architecture
  - Nexus currently supported on Linux, VxWorks, Nucleus, uC-OS, and WinCE

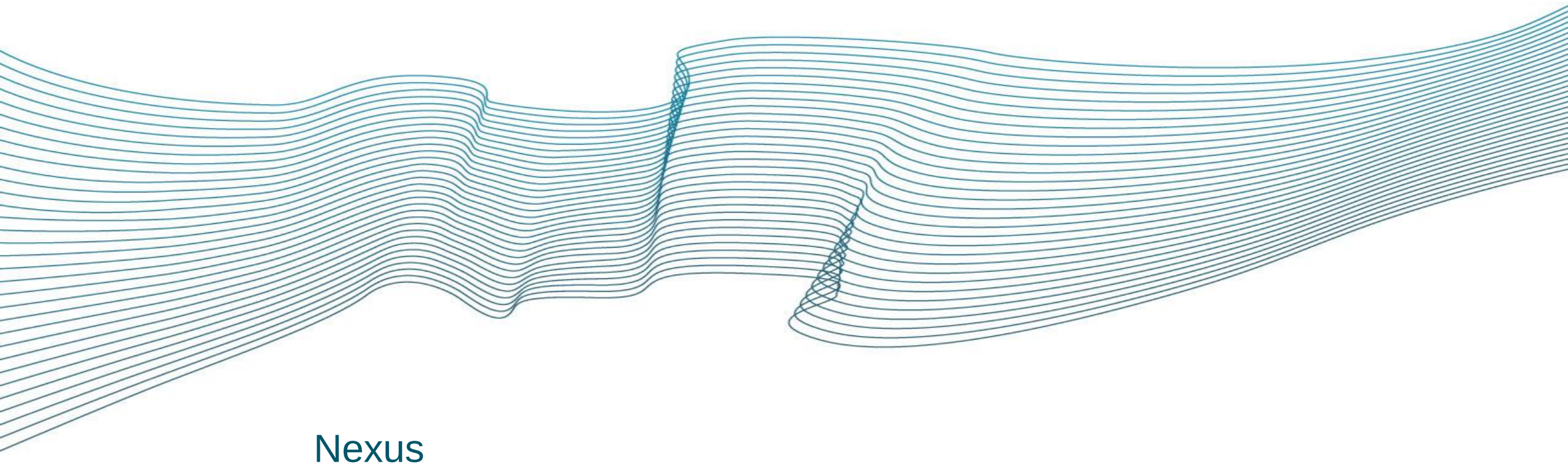
- **Nexus Platform**
  - Memory mapping
  - L1 interrupt mapping
  - OS driver(s)
- **Nexus File**
  - DVR file I/O
- **Nexus Base**
  - threads, timers, event callbacks
- **Magnum KNI**
  - events, mutexes, critical section



LINUX USER MODE/KERNEL MODE SUPPORT

- **Thunk provides seamless support for user and kernel mode**
  - Automatic kernel mode proxy
  - Same libnexus.so binary interface
- **Kernel mode advantages**
  - Lower interrupt latencies for connected products using Ethernet, USB, etc.
- **User mode advantages**
  - Easier development & debug





Nexus

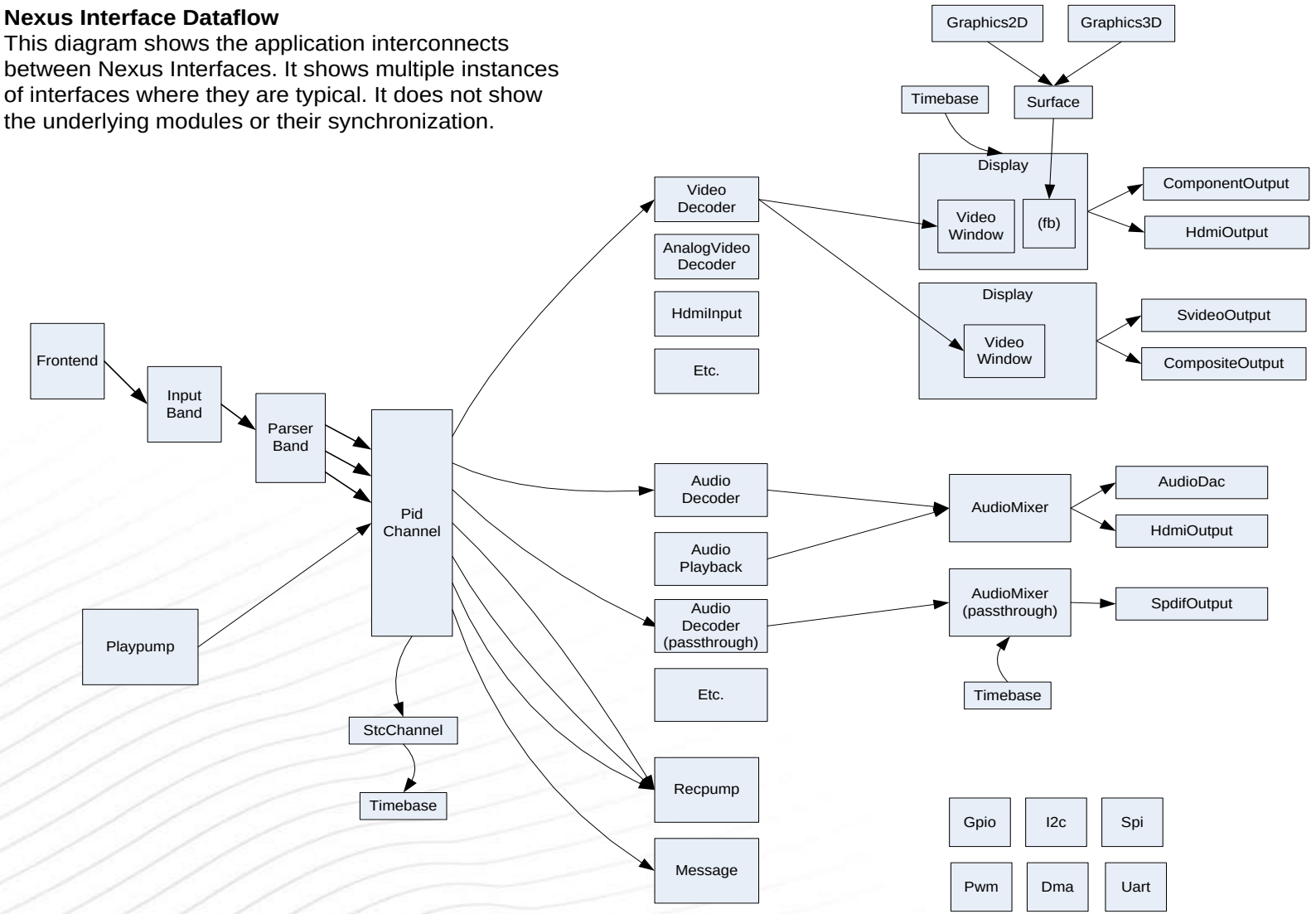
BASIC USAGE

BASIC USAGE

- **The fastest way to learn Nexus is to read the example code**
  - Every reference software release comes with a set of working examples
  - You can run them on a Broadcom reference board
  - After you bring up your platform, you can likely run them on your board
  - From there, you can read the API-level documentation in the Nexus header files
- **An example for every use case**
  - nexus/examples has subdirectories for video, audio, DVR, graphics and more
  - If something is not covered, please ask and we'll add it
- **Using examples for debug**
  - Examples have proven helpful to debug problems.
  - Instead of trying to get your whole app running in another environment, recreate the essential problem with an example.
  - Often, the process of reducing to that minimal app will expose the bug clearly.
- **Building filter graphs**
  - Designing a Nexus system is connecting a filter graph of interfaces, then configuring each interface

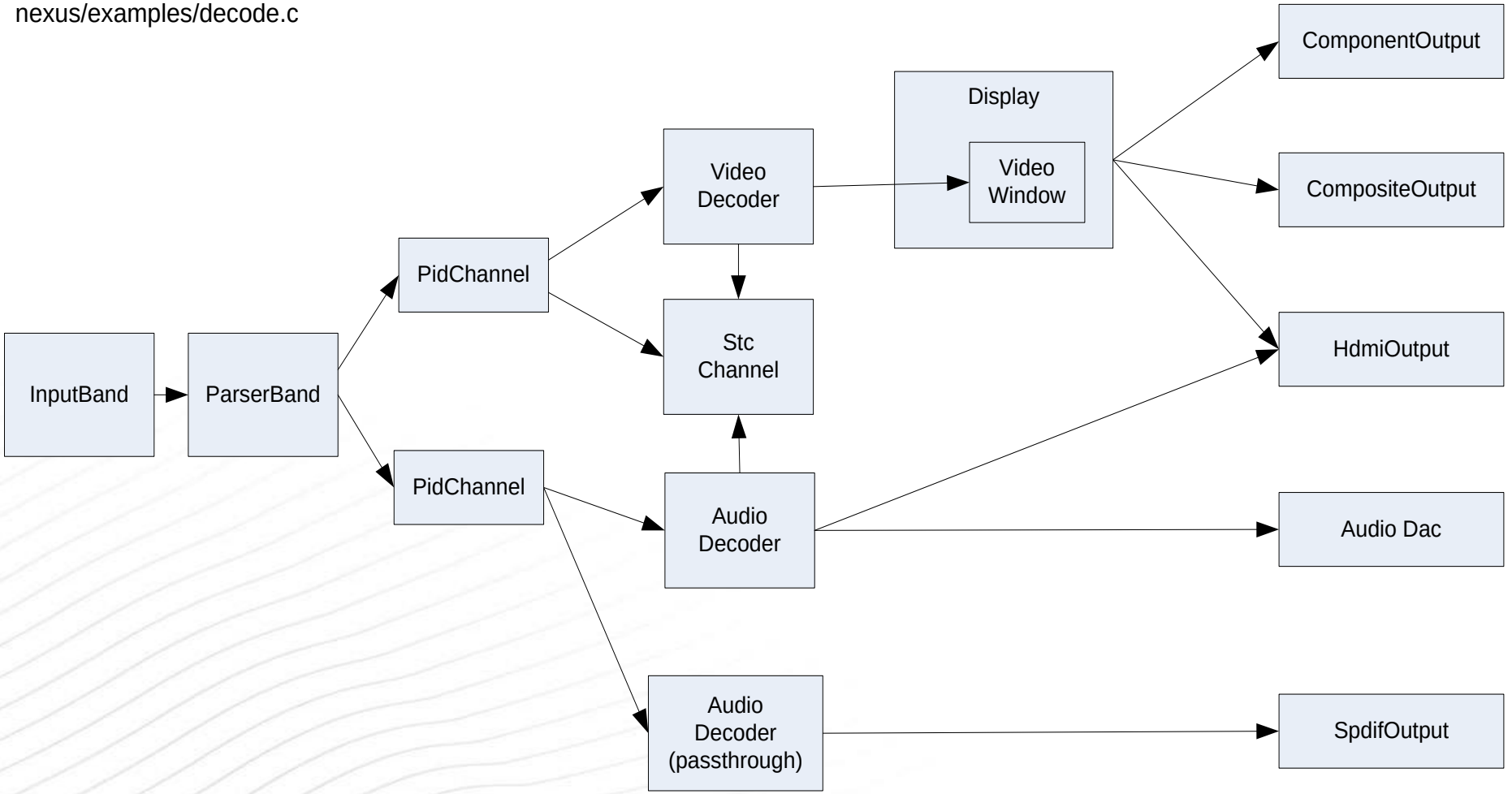
A COMPLEX FILTER GRAPH

Nexus Interface Dataflow
This diagram shows the application interconnects between Nexus Interfaces. It shows multiple instances of interfaces where they are typical. It does not show the underlying modules or their synchronization.

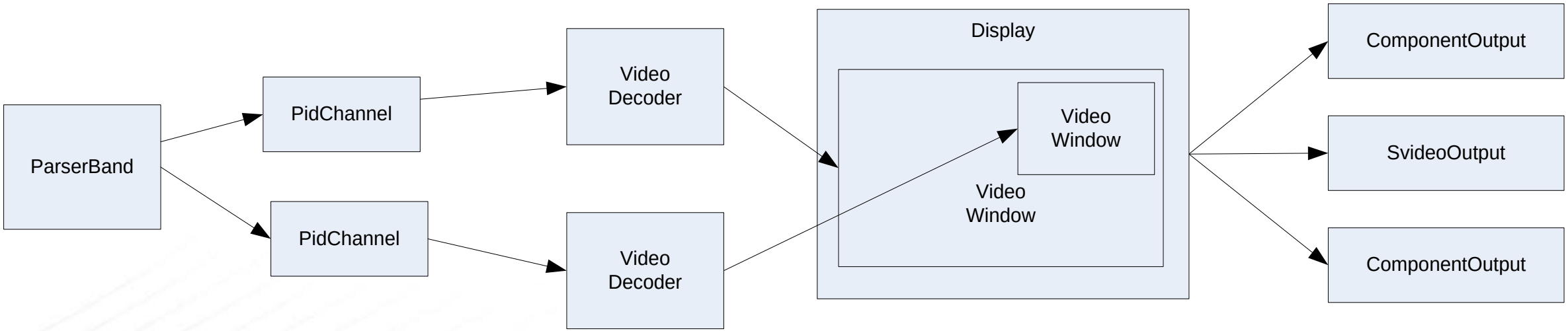


LIVE AUDIO/VIDEO DECODE

nexus/examples/decode.c

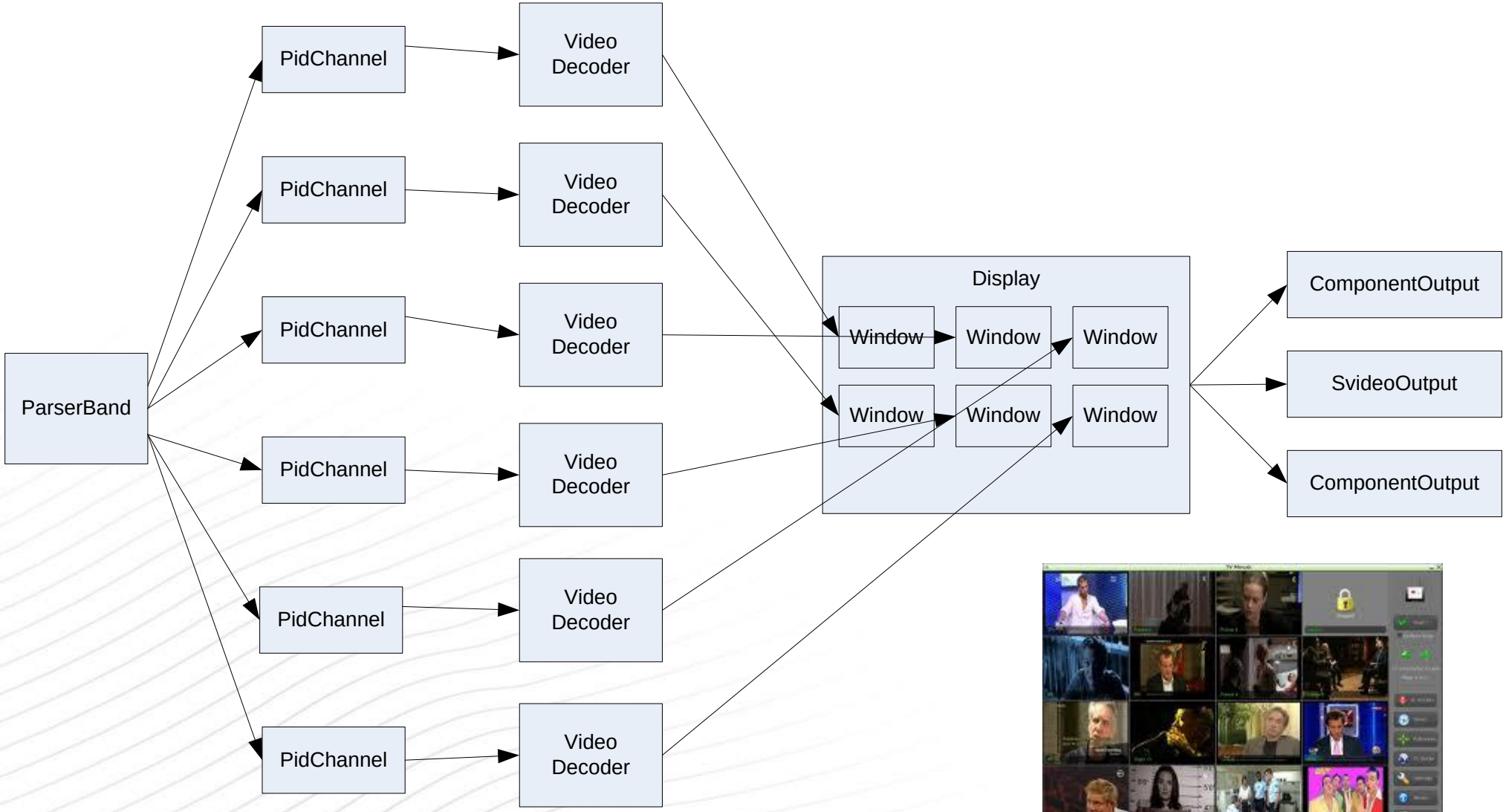


PIP

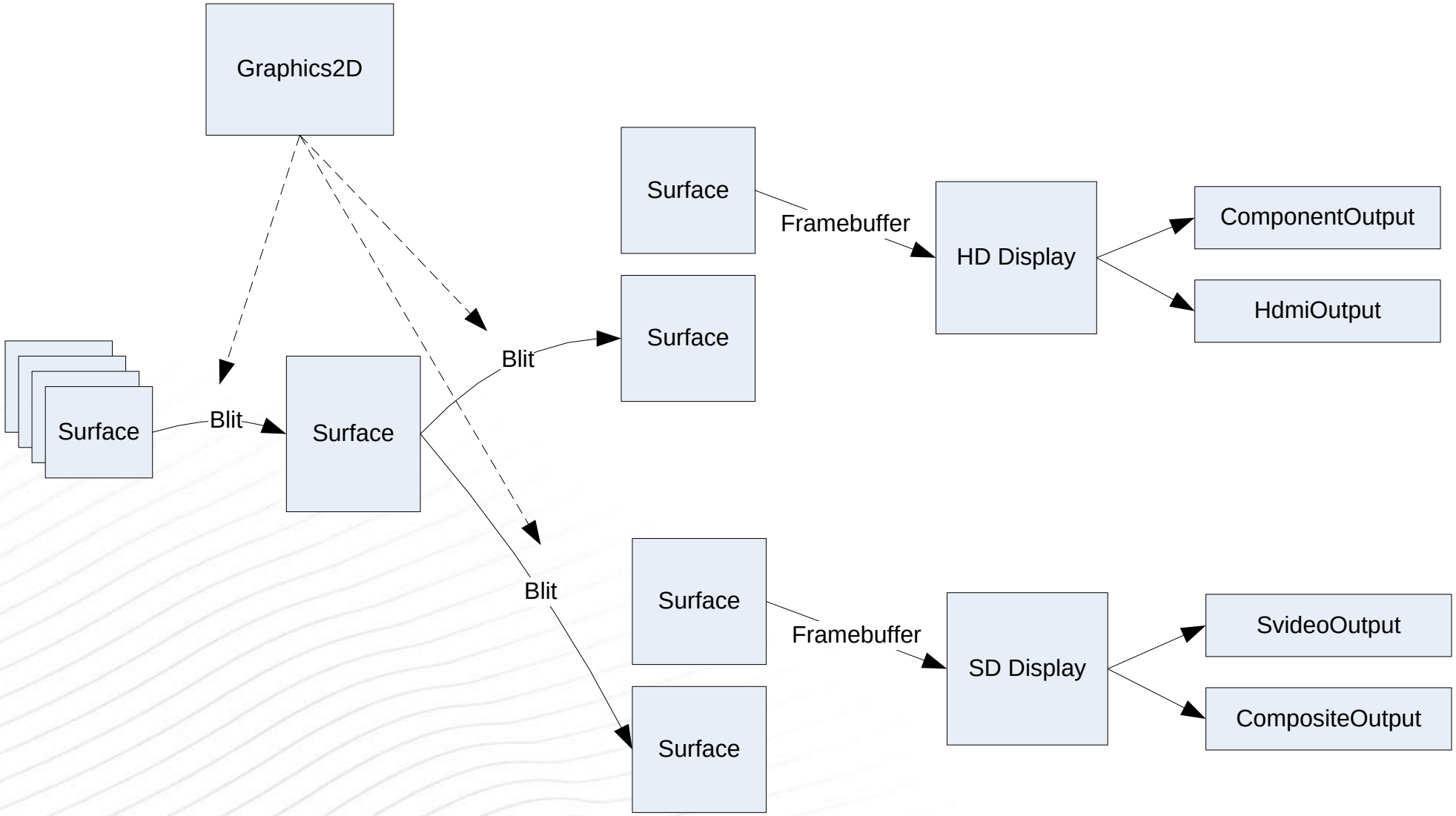


Arrows show data flow, not connection types

MOSAIC MODE

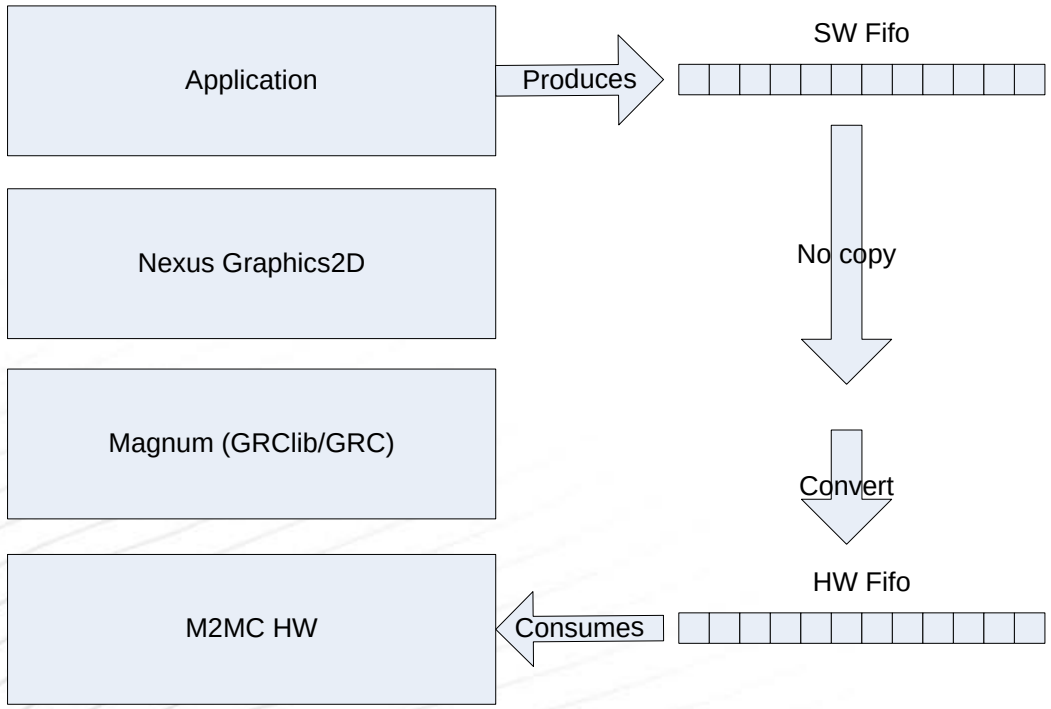


GRAPHICS



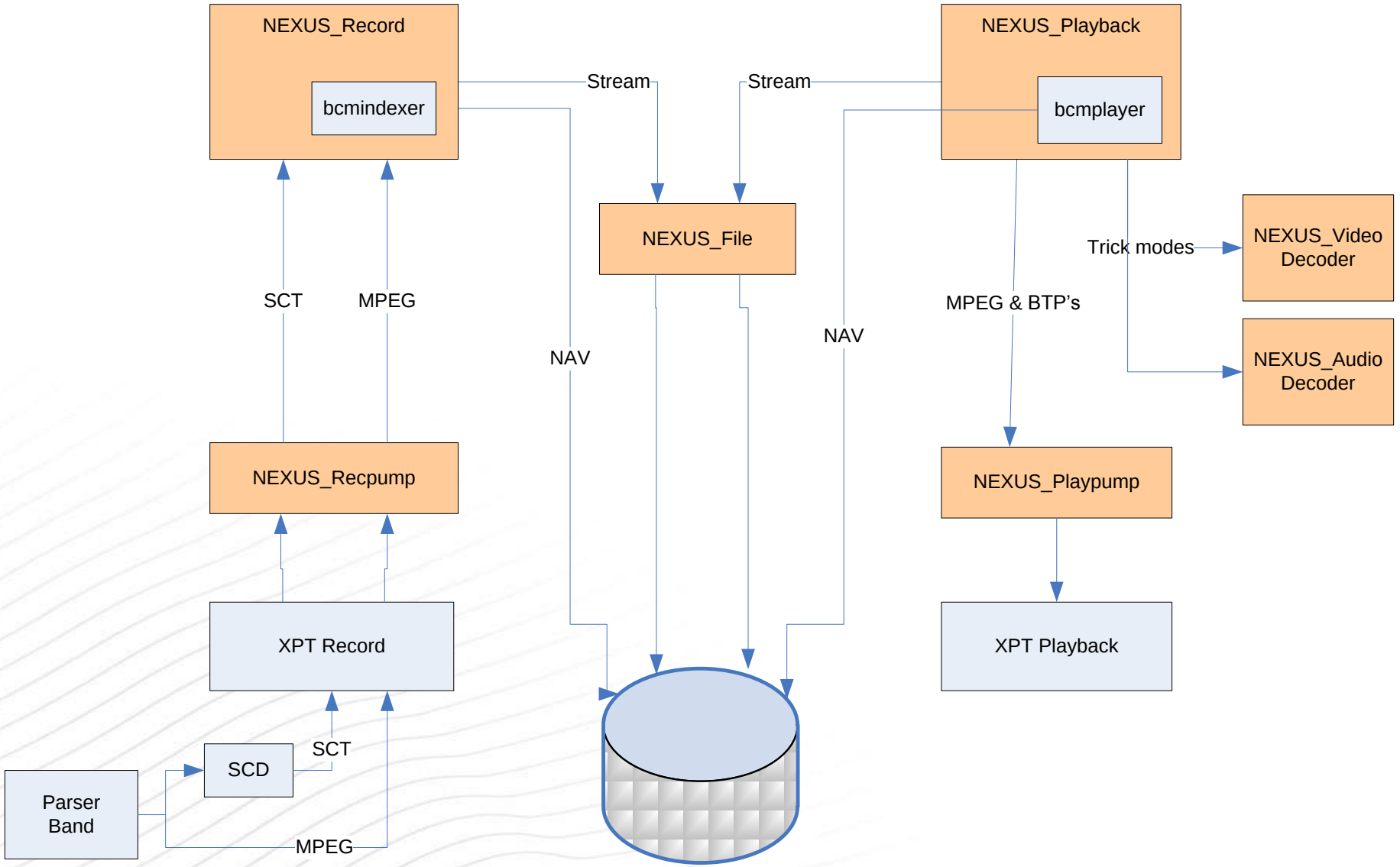
2D PACKET BLIT

- **High-performance 2D blitting with Nexus Packet Blit API**
  - Alternative to function-based blit API
- **Use data-driven technique to pipeline blits, minimize CPU overhead**

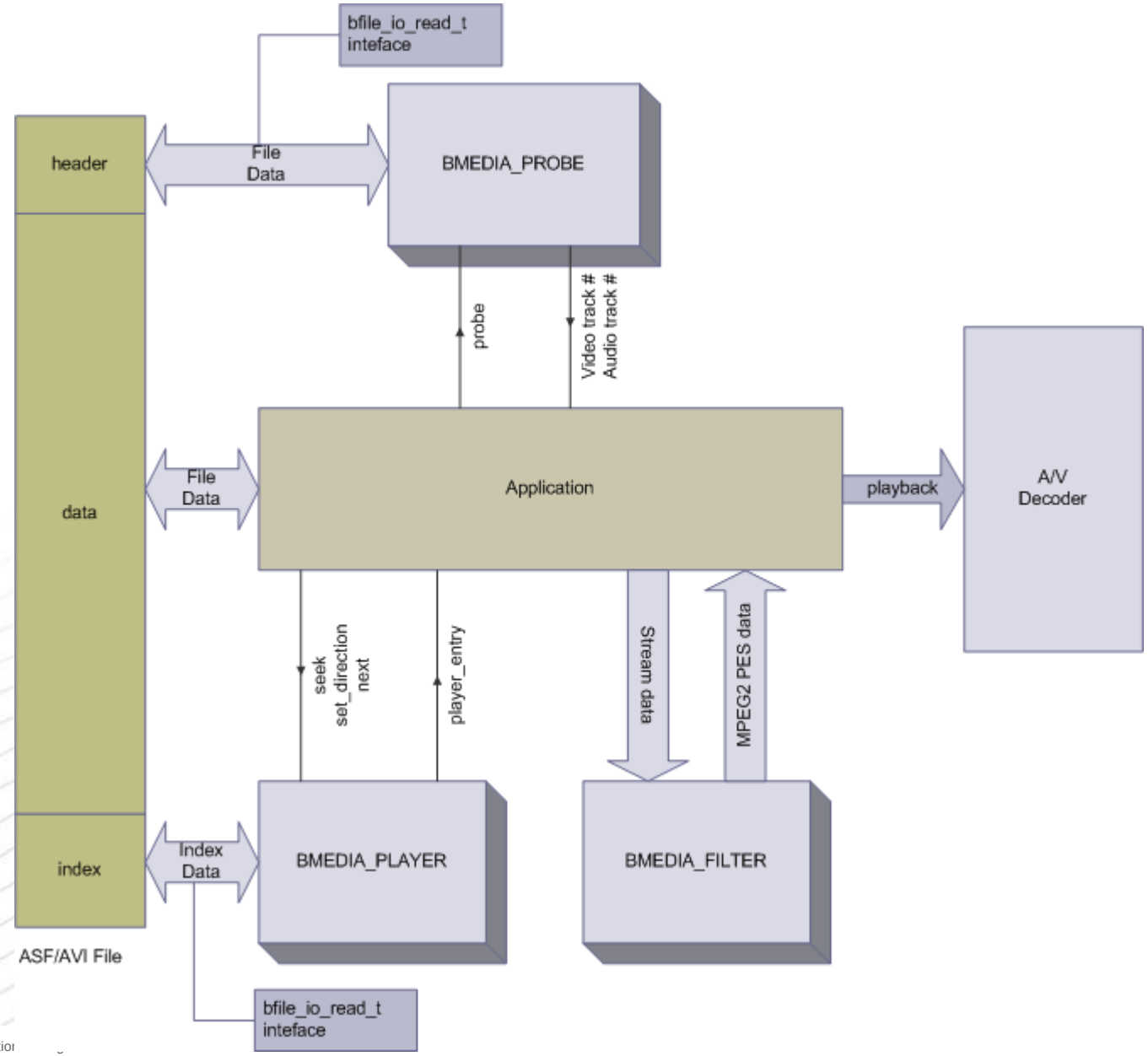


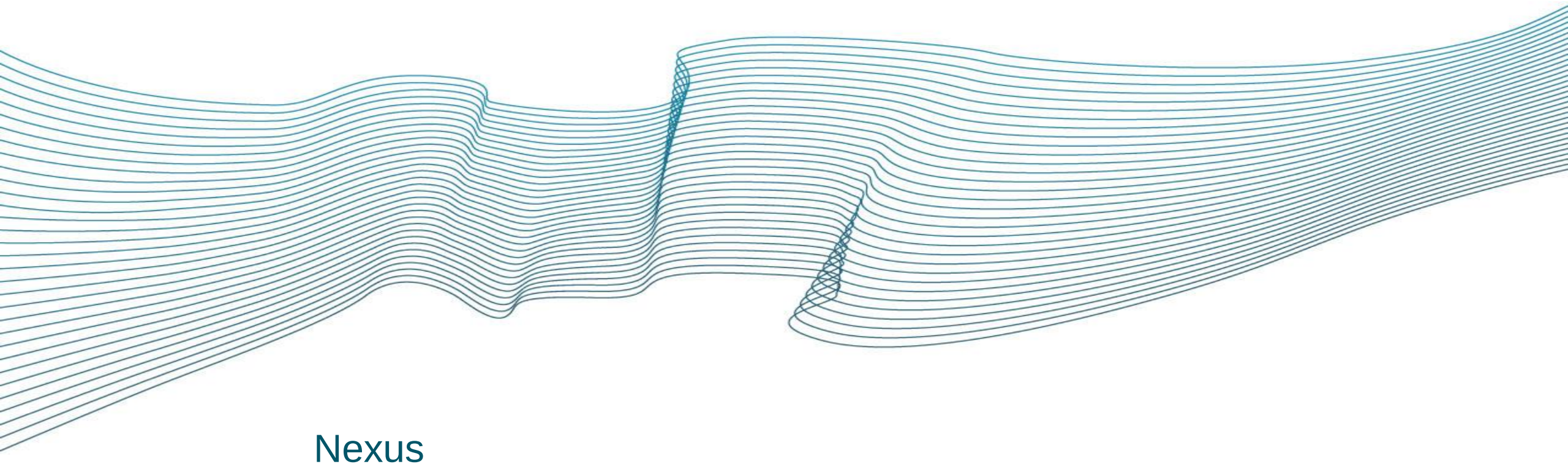
- **Ideal for multi-threaded applications (each thread gets a SW fifo context)**
- **Ideal for multi-process applications (IPC only the SW fifo pointers)**

DVR (DIGITAL VIDEO RECORDING)



MEDIA FRAMEWORK FOR DVR CONTAINER SUPPORT





Nexus

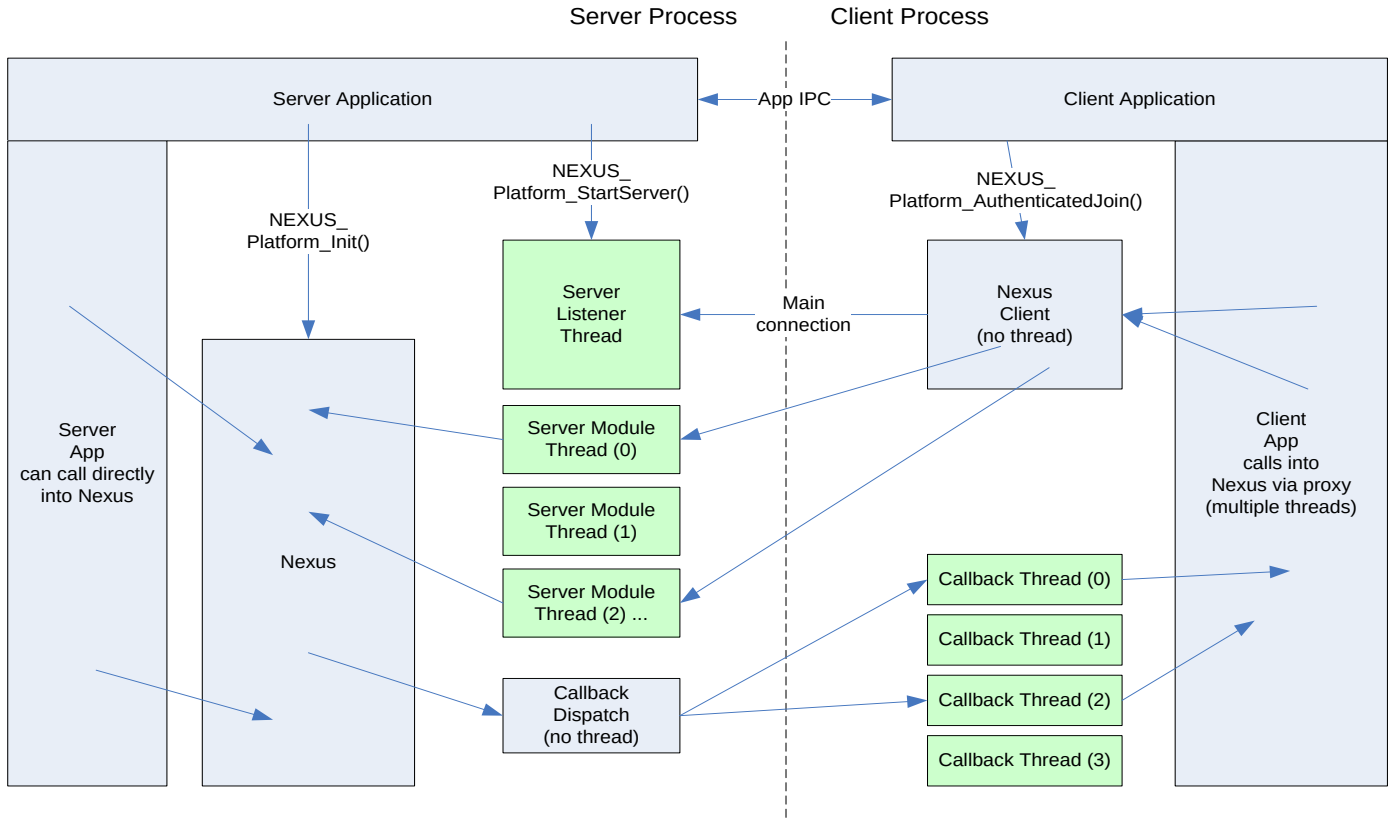
ADVANCED USAGE

ADVANCED USAGE

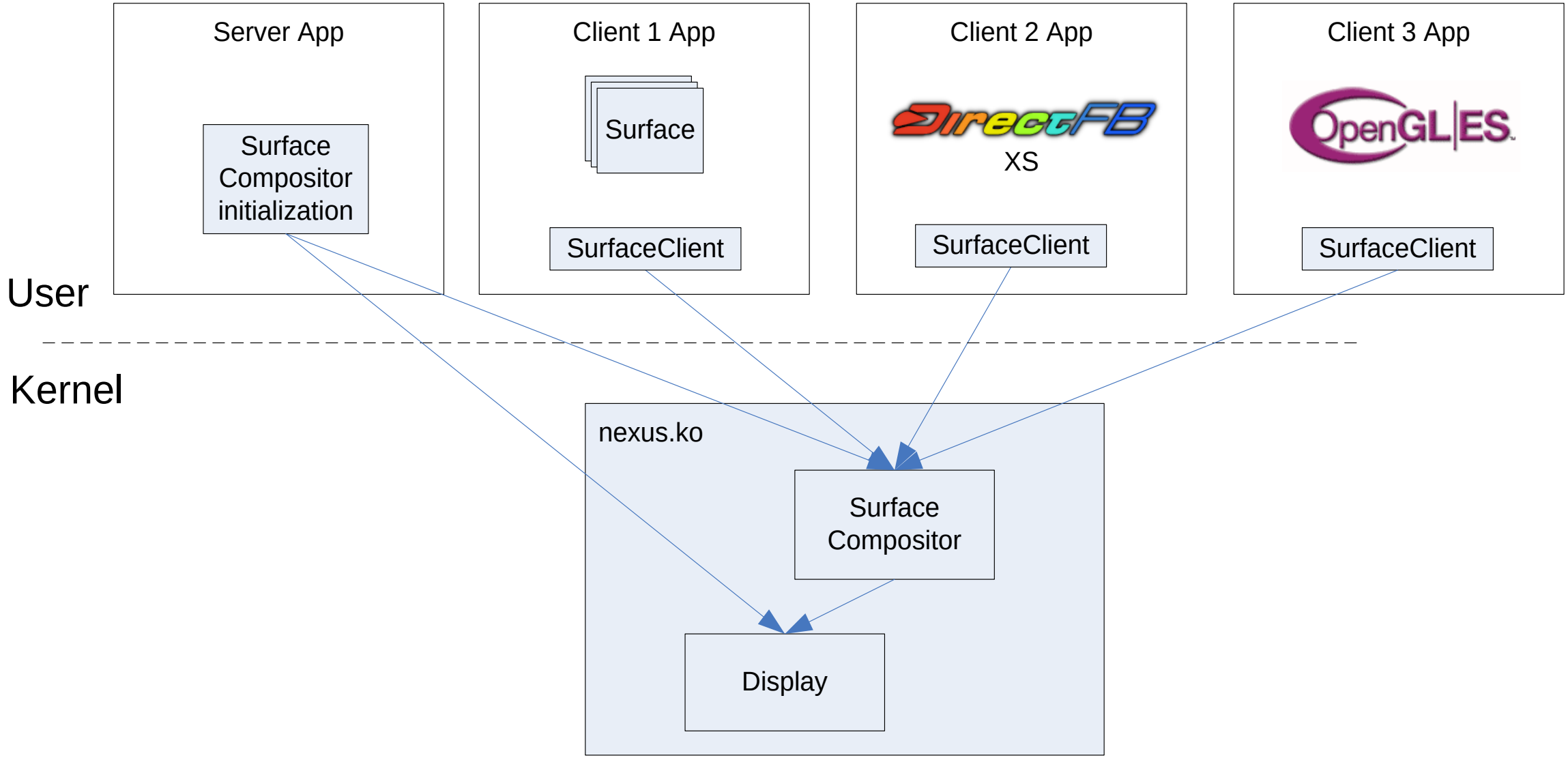
- **Multi-process support**
  - Automatic proxy of Nexus API calls from multiple processes & applications
  - Allowing untrusted clients in a secure system
- **3D graphics using OpenGL-ES 2.0**
- **Power management**
  - Dynamic power management
  - Active and passive standby
- **Audio/video transcode**
- **3DTV support**
- **SVC/MVC video decode**
  - Scalable Video Coding, Multiview Video Coding
- **Large memory systems**
  - Managing 2GB of memory or more on 32-bit MIPS
- **AppLibs**

MULTI-PROCESS SUPPORT

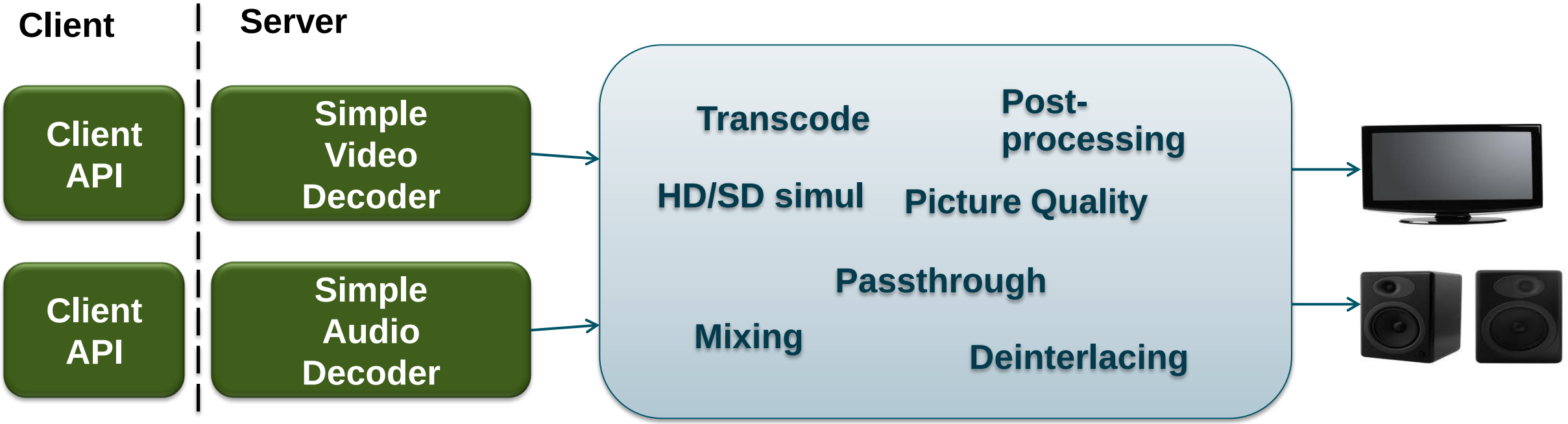
- **Call the Nexus API from multiple applications**
  - Complex software systems
  - Run untrusted applications without compromising your main app
- **Implemented with Nexus thunking**
  - Kernel mode proxy is inherently multi-process (with necessary callback routing added)
  - User mode socket-based IPC thunk has been added
- **Variety of techniques used to secure the server**
  - Object database of nexus handles and clients
  - Heap isolation – non-root clients can only map memory granted by the server
  - All function parameters verified on the server
  - Limited client API



SURFACE COMPOSITOR FOR MULTI-PROCESS GRAPHICS



SIMPLE DECODER FOR UNTRUSTED CLIENTS

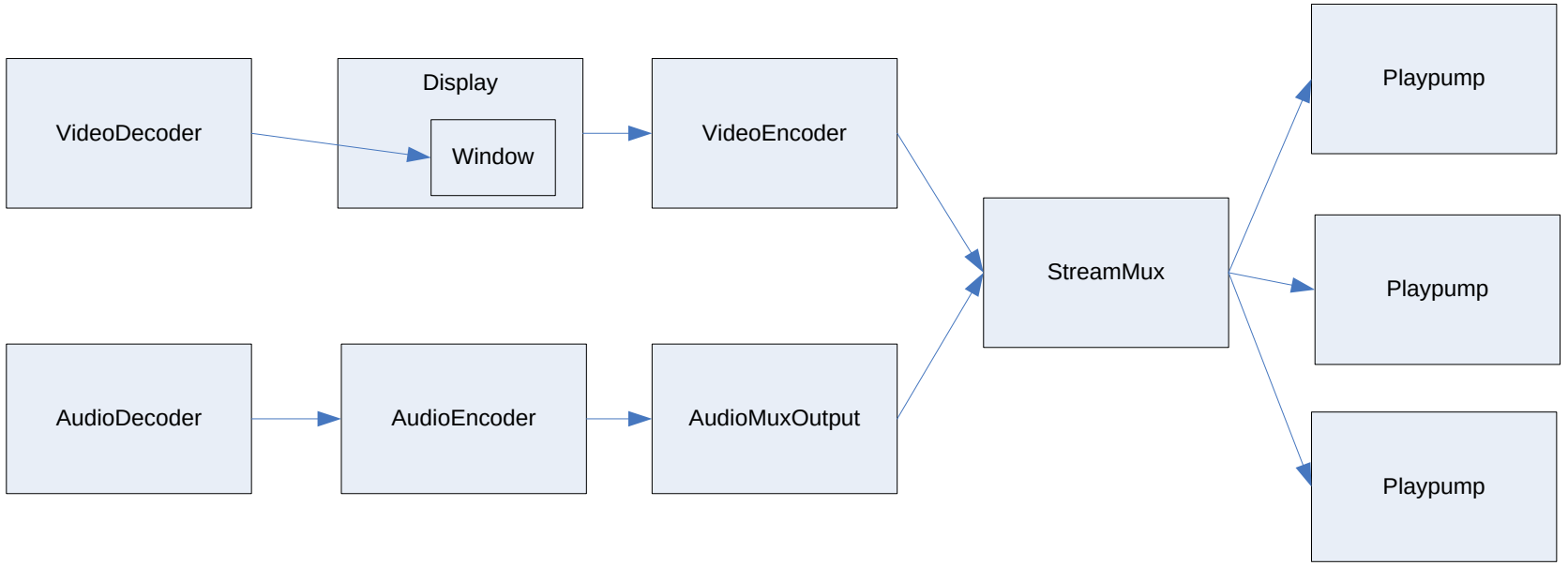


POWER MANAGEMENT

- **ACPI (Advanced Configuration and Power Interface) states**
  - S0 – Running
    - Full operation.
    - Dynamic power management of unused devices
  - S1 – Active standby
    - CPU running. Some devices clock gated (maintain state)
    - Typically frontend and transport running for background DVR
  - S2 – Passive standby
    - CPU powered down. Non-wakeup devices clock gated (maintain state)
  - S3 – Deep sleep
    - CPU powered down. Non-wakeup devices power gated (state must be saved to RAM)
    - Minimum power, but longer wake up time.
- **Variety of wake-up devices**
  - IR, front panel, UHF, HDMI CEC, Wakeup-on-LAN, USB, GPIO, timer

AUDIO/VIDEO TRANSCODE

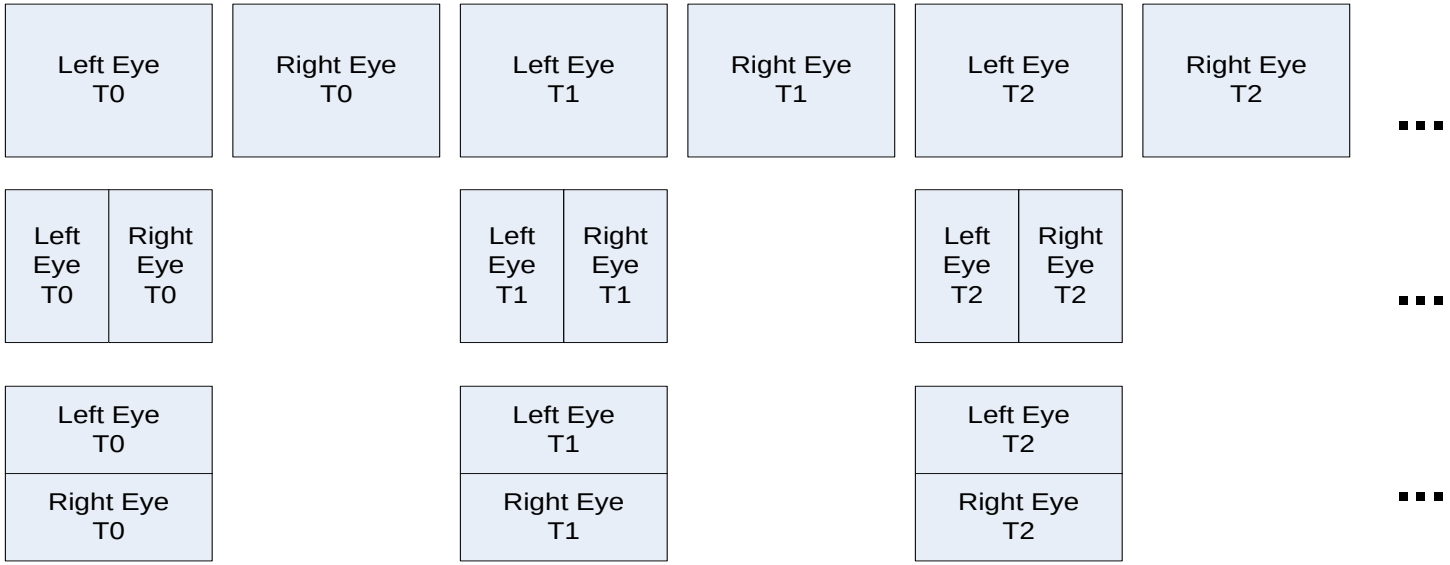
- **Video transcode through video decode (AVD) and video encode (VICE)**
  - MPEG/VC1/AVC/MPEG4, up to 1080p30
- **Audio transcode through audio decode and encode (Raaga)**
  - AAC in Phase 1, others to follow



- **Output**
  - NEXUS_StreamMux output to transport for IP streaming
  - NEXUS FileMux output to disk for containers like MP4, ASF, AVI, MKV
  - Real-time and non-real-time modes

3DTV

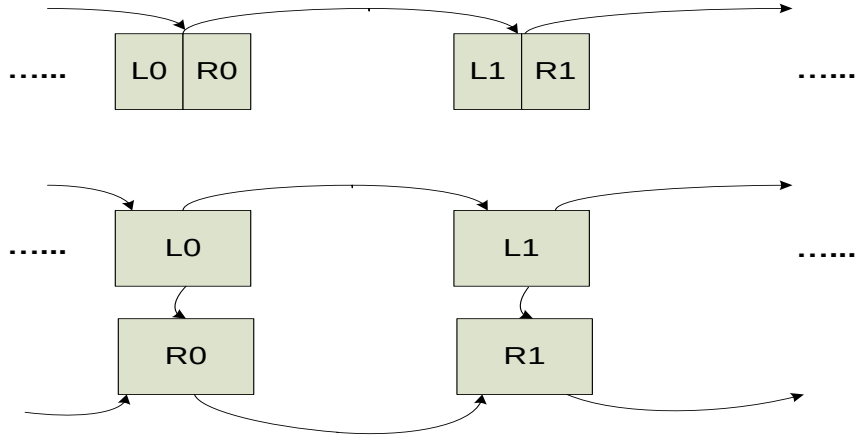
- **40nm chips have integrated 3DTV support (e.g. 7425, 7231, …)**
  - Full-res decode 1080p24 or 720p60 for each eye
  - MVC and SVC decode can be used for 3DTV streams
  - Integrated L/R graphics support
  - HDMI VSI, AVC SEI signaling support
- **65nm chips have more manual 3DTV support (e.g. 7420, 7342, …)**
  - Half-res decode support
  - Application must composite L/R graphics framebuffer
  - HDMI VSI, AVC SEI signaling support



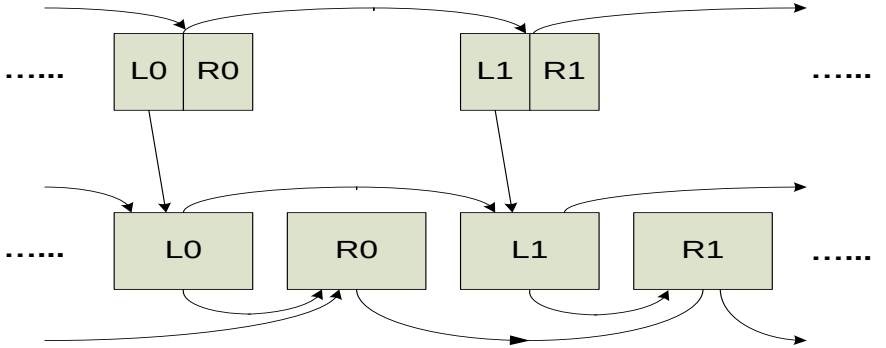
SVC/MVC DECODE

- **SVC = Scalable Video Coding**
  - Base Layer & Enhancement Layer (e.g. 720p → 1080p)
- **MVC = Multiview Video Coding**
  - Add second camera angle (e.g. main camera, close-up camera)
- **Scalable 3D broadcast**
  - Both MVC and SVC can be used to add 3D
  - For example: 1080p24 half res → 1080p24 full res

MVC



SVC



- **Extensible Nexus API**
  - Just add the enhancement pid to NEXUS_VideoDecoderStartSettings

3D GRAPHICS

- **40nm chips support 3D graphics through Video Core IV (VC4) core**
- **Broadcom's 3D graphics API is OpenGL-ES**
  - Nexus has no underlying 3D graphics API
- **Specifications**
  - 12M rendered vertices/sec
  - 180M pixels/sec with single bi-linear texturing, simple shading, 4x multisampling
  - Supports 16x coverage mask anti-aliasing for 2D rendering at full pixel rate
  - 720p standard resolution with 4x multisampling
  - Fully supports OpenGL-ES 1.1/2.0 and OpenVG 1.1

LARGE MEMORY SYSTEMS

- **For 32-bit MIPS, user space and kernel space has a maximum of 2GB of virtual address space**
  - A good portion of that space must be used for code, data, register access and other overhead
  - Also, some memory must be mapped twice (cached and uncached access)
- **Nexus provides full heap control**
  - Minimal mapping of each memory region
  - Default heaps provided; fully customizable

NEXUS_MemoryType	Memory Mapping
NEXUS_MemoryType_eDriver	Kernel mode cached/uncached CPU access
NEXUS_MemoryType_eApplication	User mode cached CPU access
NEXUS_MemoryType_eFull	Full kernel and user mode CPU access
NEXUS_MemoryType_eDeviceOnly	No mapping. No CPU access required.

APPLIBS INTEGRATION

- **Provide suite of integrated libraries and applications on top of Nexus**
  - Provided as reference code
  - Can also be used directly
- **Standard open source libraries**
  - DirectFB
  - WebKit
  - DLNA
  - OpenGL-ES
- **Third party plug-ins**
  - Adobe Flash 10

LEARNING MORE ABOUT NEXUS

Every Broadcom reference software release contains what you need to learn about Nexus

- **Nexus Documentation**
  - nexus/docs
- **Nexus Example Applications**
  - nexus/examples
- **Nexus API header files**
  - nexus/modules/*/include/*.h

Q & A