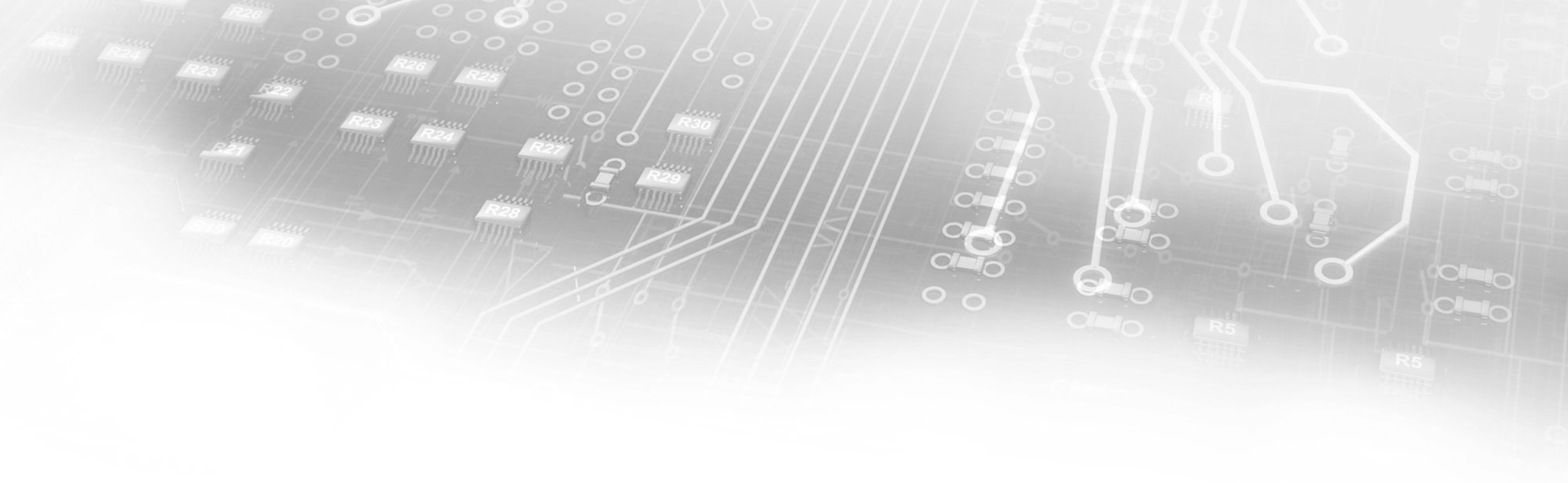
Nexus Overview

David Erickson

March 28, 2016

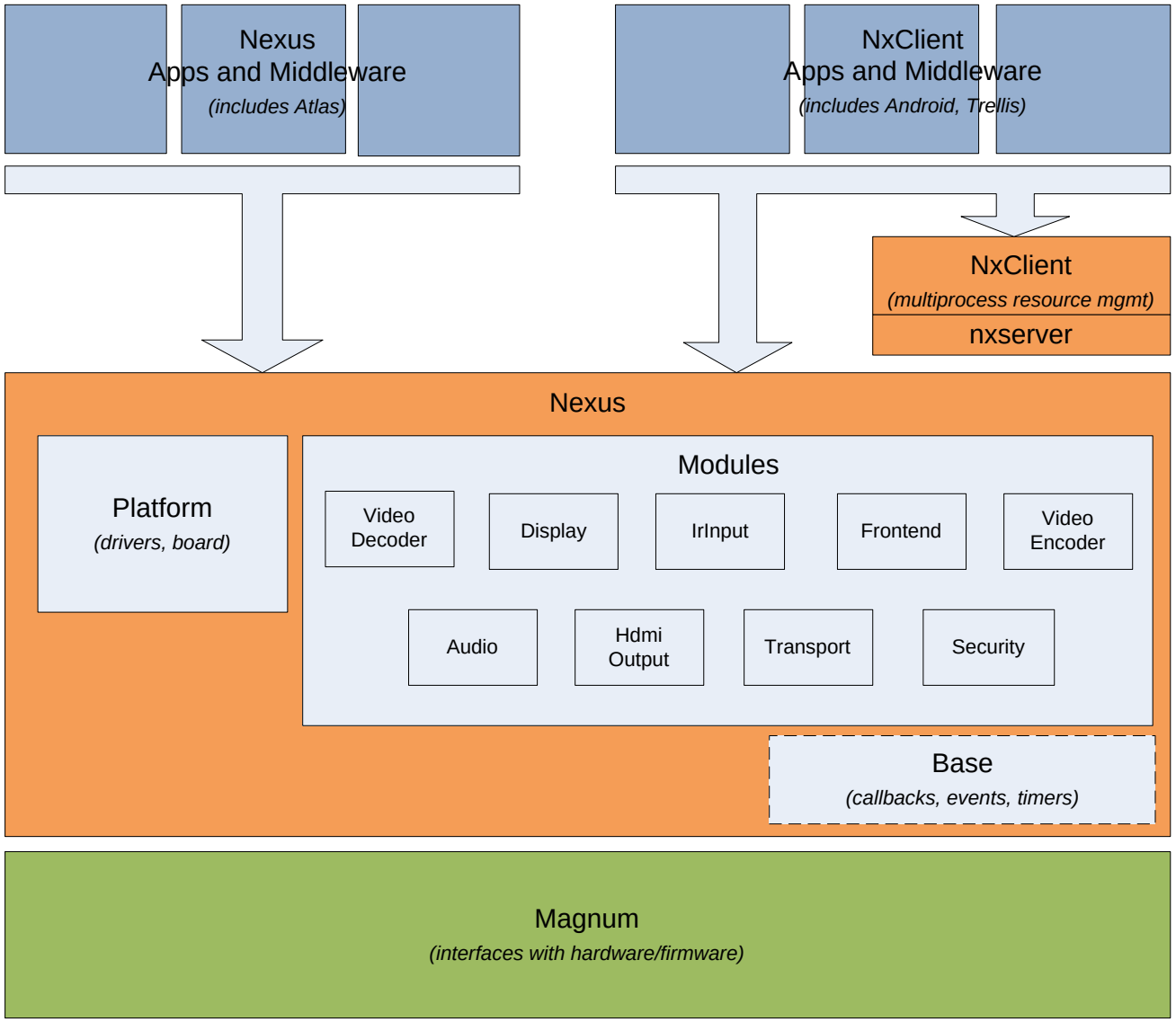
Presentation Overview

- Basics
  - Reference software stack and unified release
  - Simple app
  - Audio/video decode, graphics, DVR
- Architecture
  - Synchronization, interfaces, drivers, Linux user/kernel mode
- Multi-Process
  - Proxy mechanism
  - Resource tracking and garbage collection
  - Virtualization, dynamic allocation and new client-server interfaces
  - NxClient
- Advanced Usage
  - Transcode, 3DTV, packet blit, power management
  - AppLibs integration including DirectFB, WebKit, DLNA, Flash



Nexus Basics

Reference Software Stack



BROADCOM

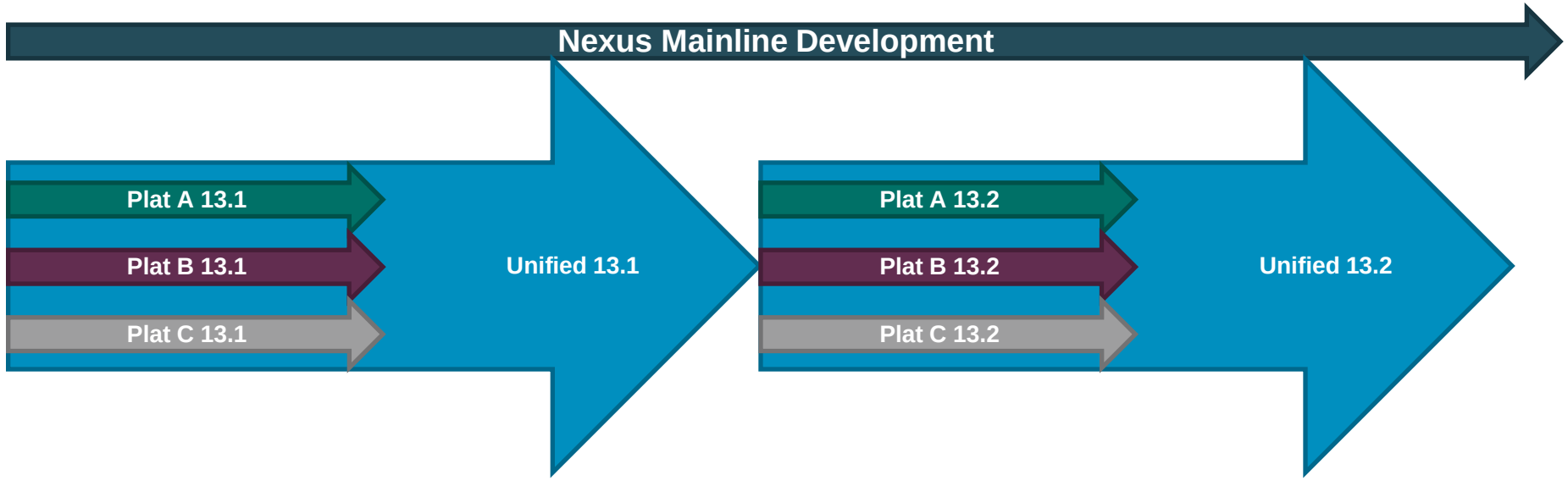
What is Nexus?

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

- Easy to use API
  - Based on customer usage, not internal implementation
  - Mapped to a variety of middleware
  - Designed for backward-compatible API changes
- Robust internal architecture
  - Automatic synchronization that is safe and efficient
  - Multi-process proxy
- Flexible modular design
  - Abstractions for multiple OS's
  - Internal modularity makes updates easier
  - Platform-specific code is isolated, making updates easier

Unified Release Model

- All platforms released concurrently using common codebase
- Easy to understand quarterly release schedule
  - URSR 14.1 = Q1 of 2014
- Phase 0.5 engineering release for new silicon still independent



BROADCOM

Simple Nexus App

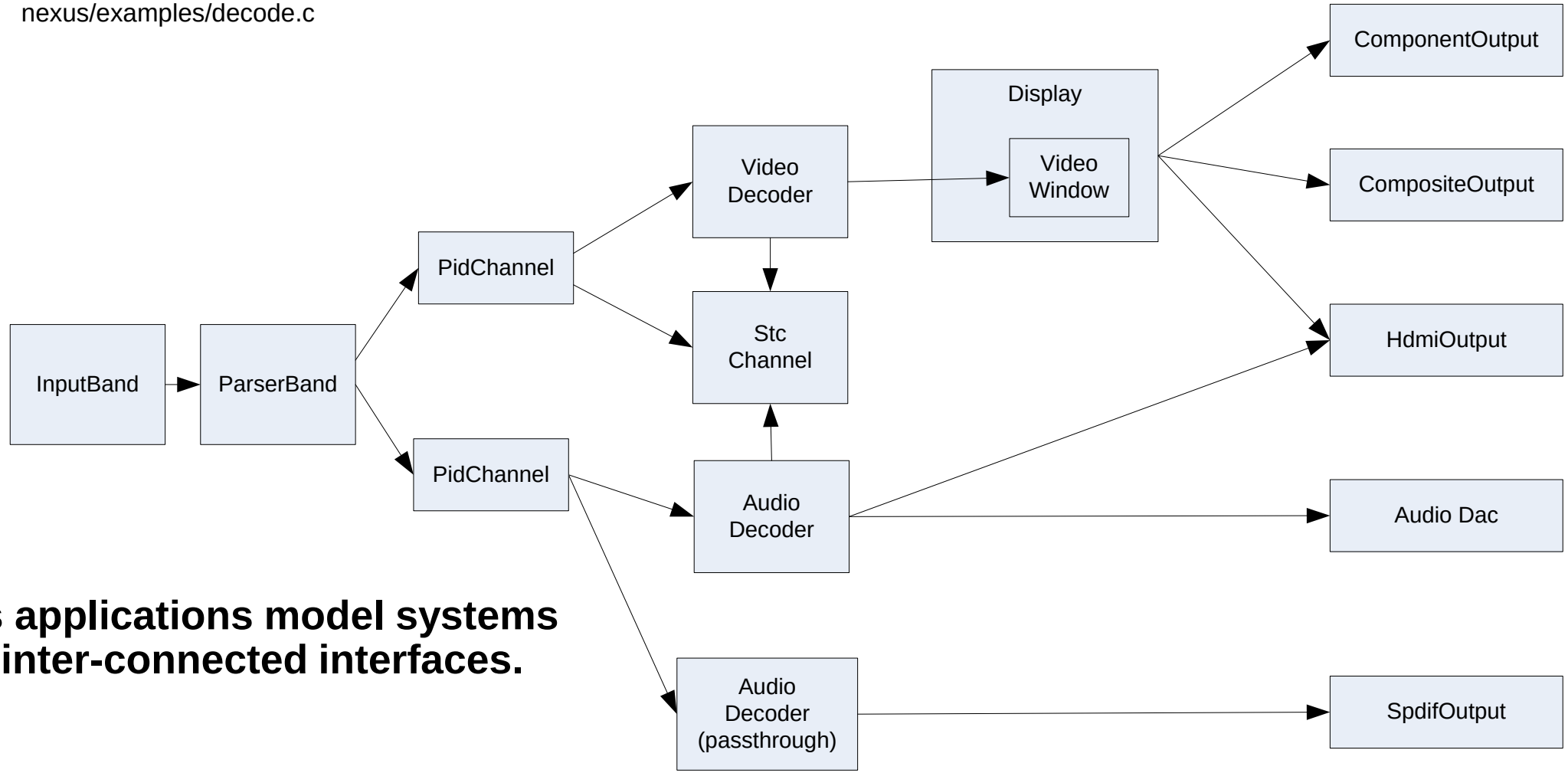
Learn Nexus by reading and running example code.


```
int main() {
  NEXUS_Platform_Init();

  display = NEXUS_Display_Open(0, &openDisplaySettings);
  window = NEXUS_VideoWindow_Open(display, 0);

  decoder = NEXUS_VideoDecoder_Open(0, &openSettings);
  NEXUS_VideoWindow_AddInput(window, GetConnector(decoder));

  startSettings.pid = NEXUS_PidChannel_Open(parserBand,0x11);
  startSettings.codec = NEXUS_VideoCodec_eH264;
  NEXUS_VideoDecoder_Start(decoder, &startSettings);

  printf("video decoding…\n");
}
```


BROADCOM

Live Audio/Video Decode

nexus/examples/decode.c

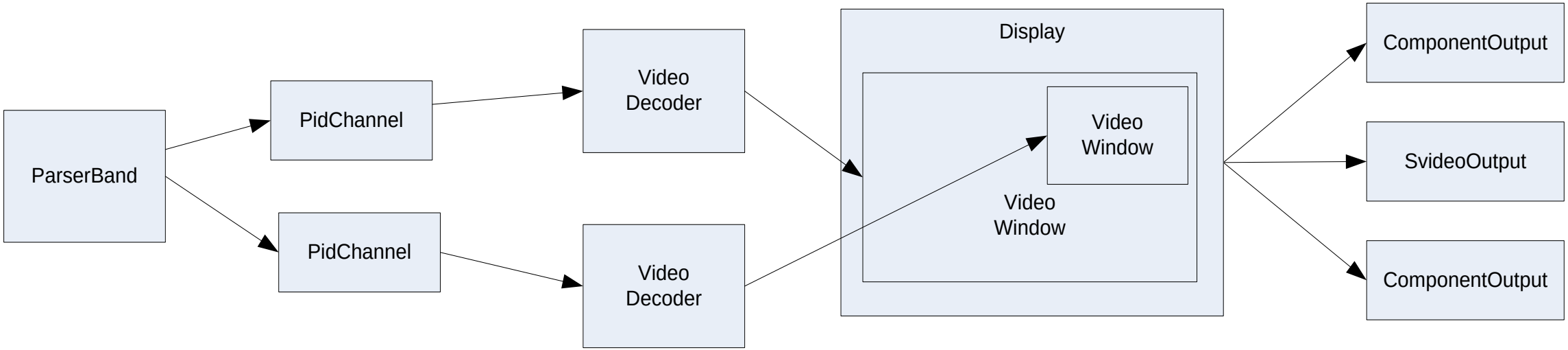


Nexus applications model systems using inter-connected interfaces.

BROADCOM

PIP

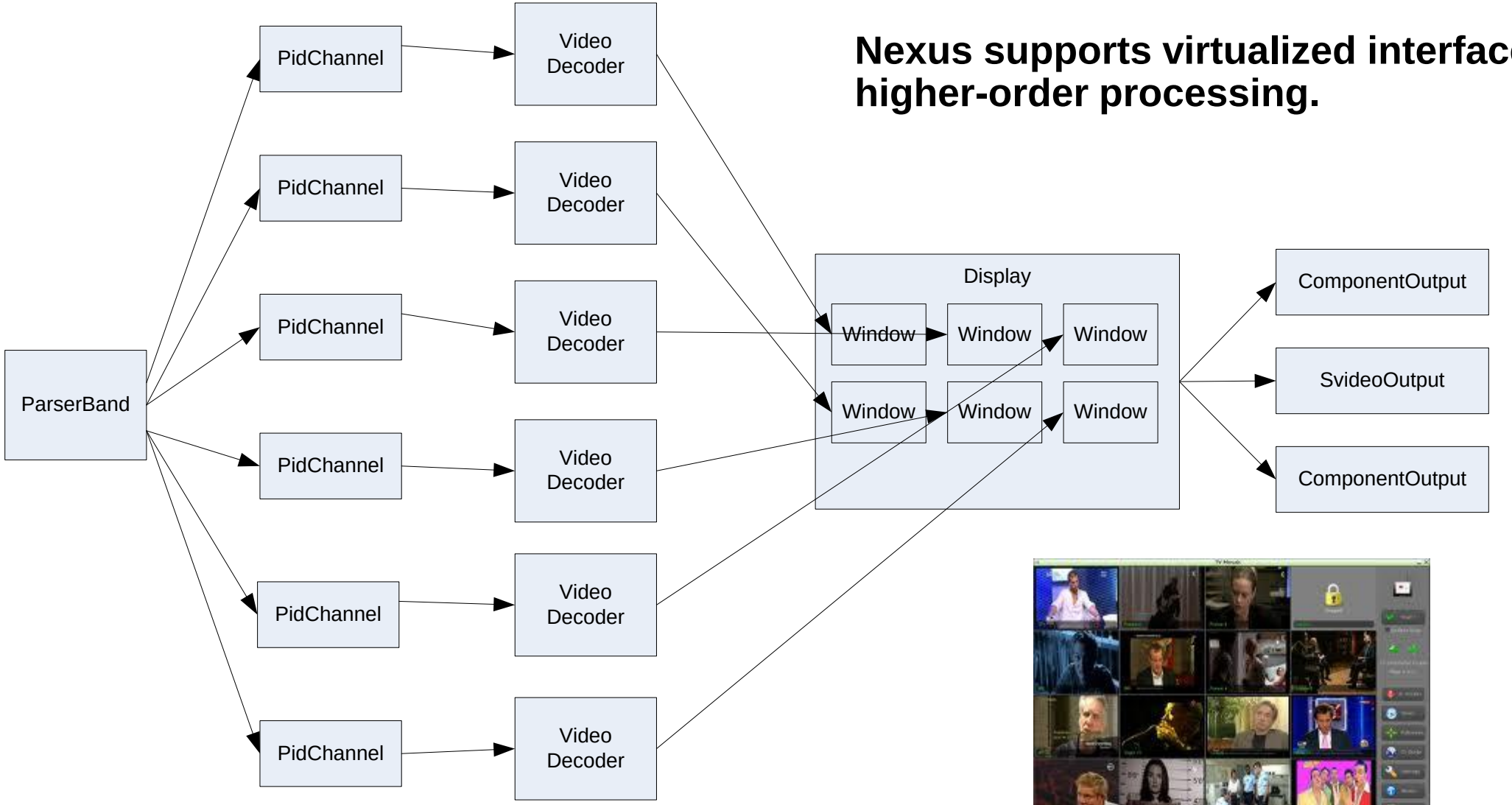
nexus/examples/video/pip.c



Nexus apps can model complex usage modes by creating and connecting more interfaces.

BROADCOM

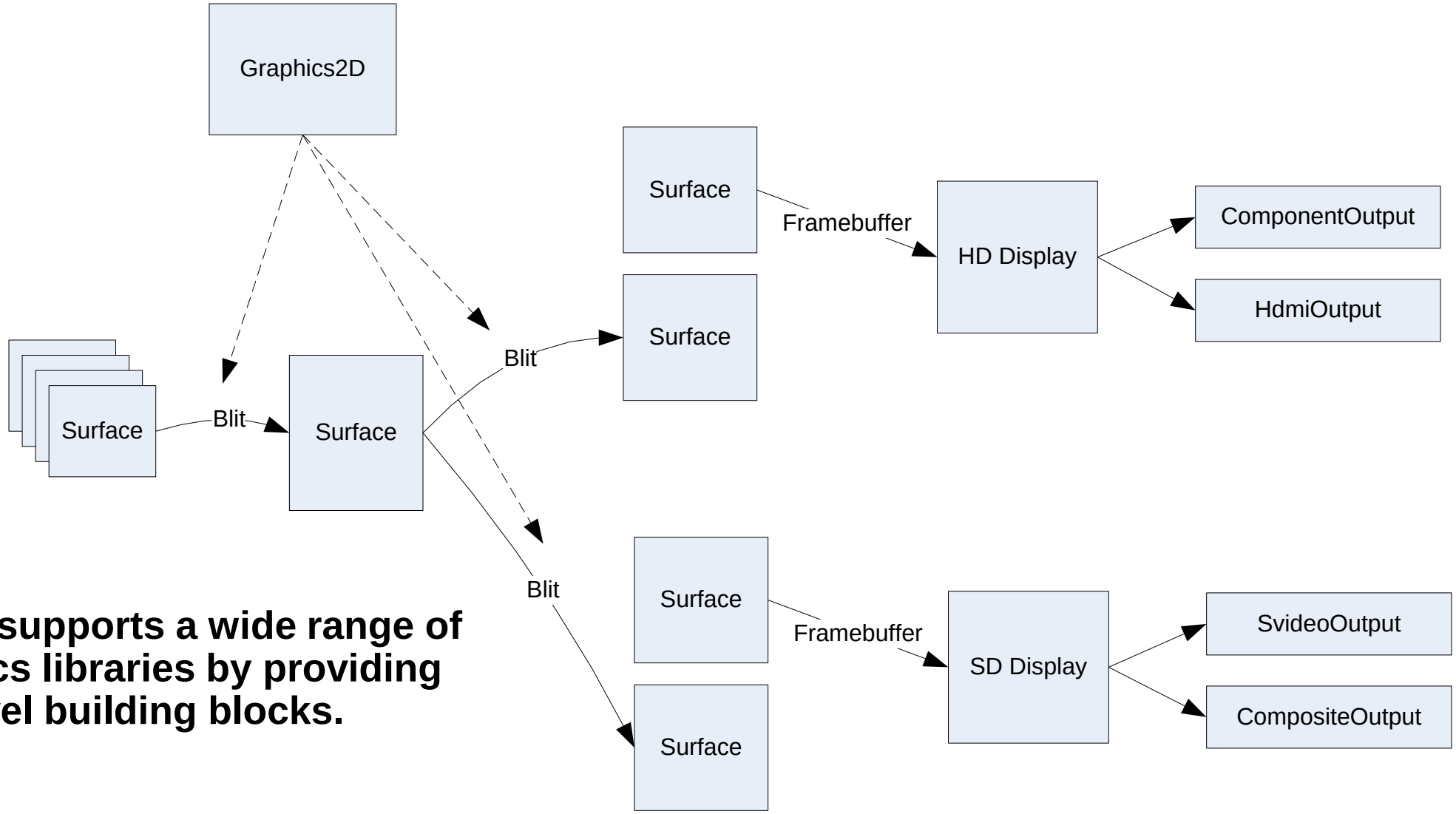
Mosaic Mode



Nexus supports virtualized interfaces for higher-order processing.

BROADCOM

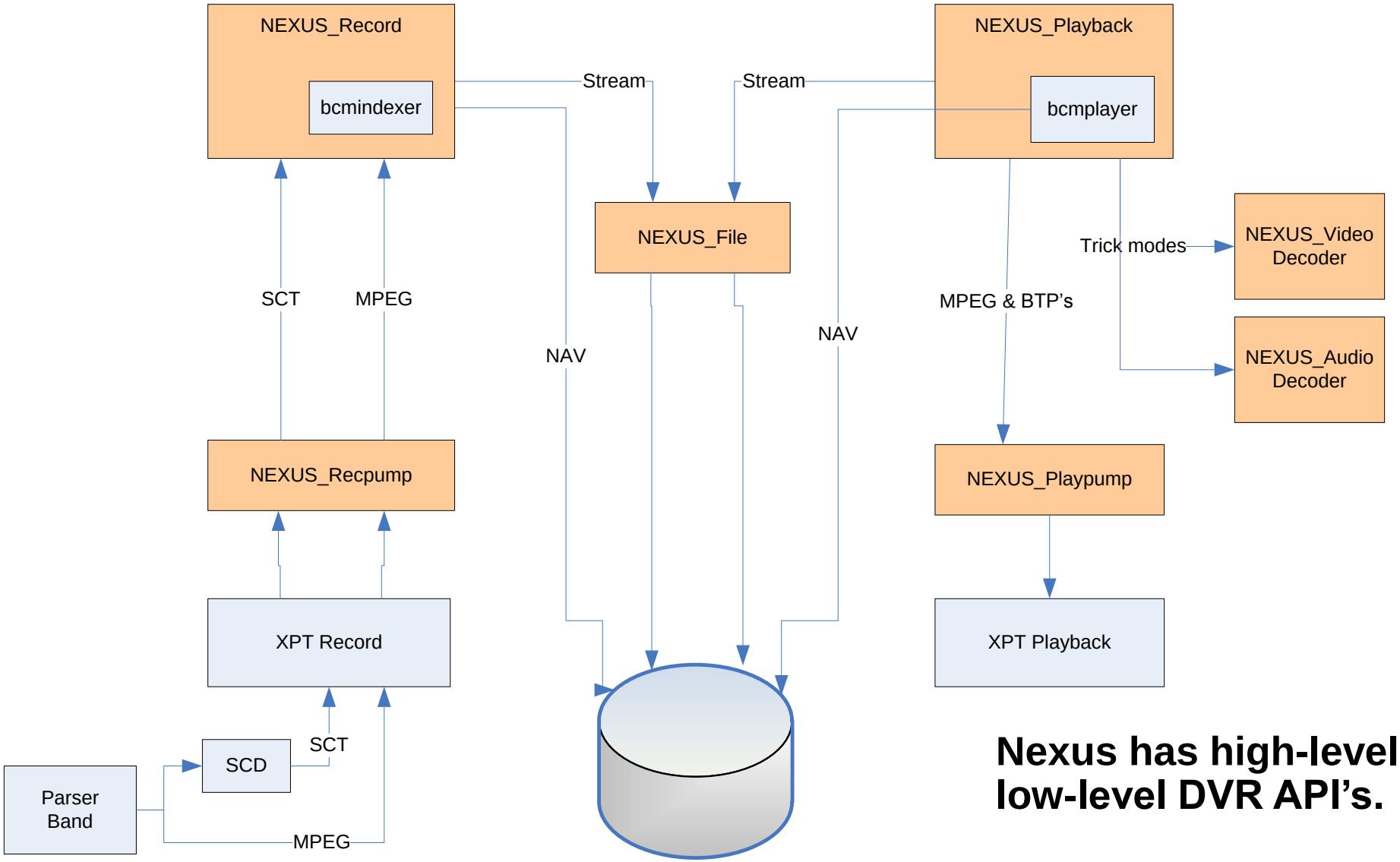
Graphics



Nexus supports a wide range of graphics libraries by providing low-level building blocks.

BROADCOM

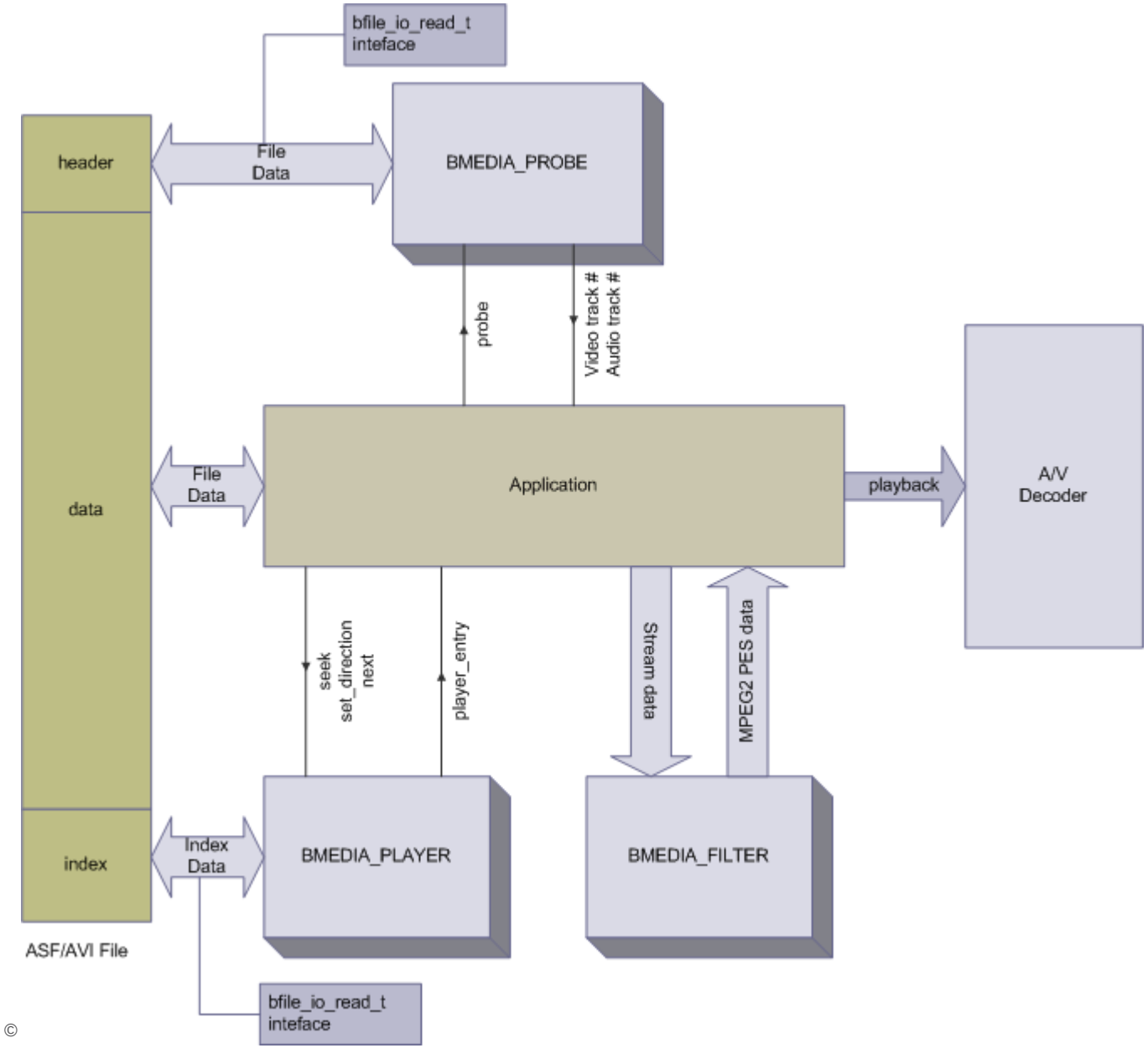
DVR (Digital Video Recording)

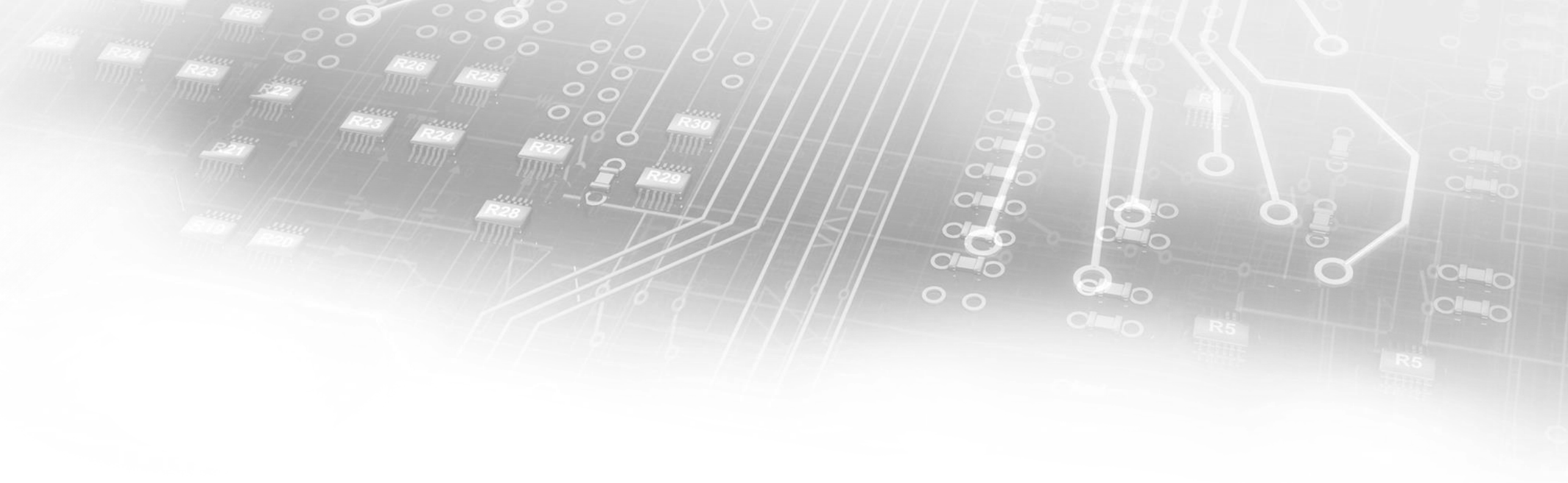


Nexus has high-level and low-level DVR API's.

BROADCOM

Media Framework for DVR Container Support



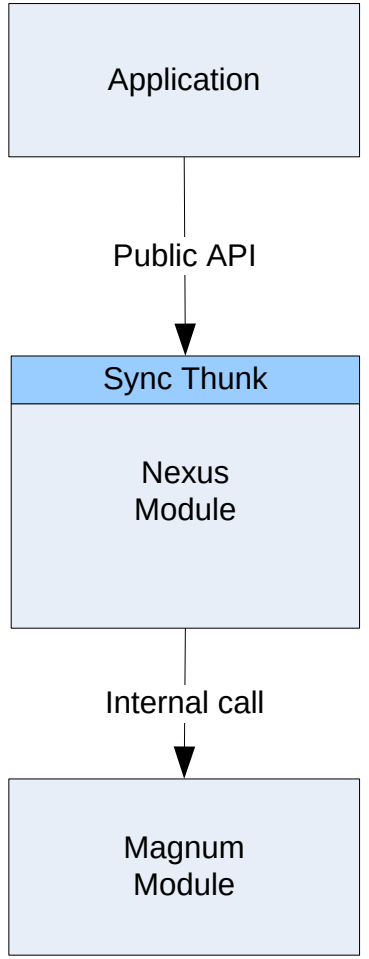


Nexus Architecture

Perl-generated Thunk for Sync & Multi-Process

Example synchronization thunk code:

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

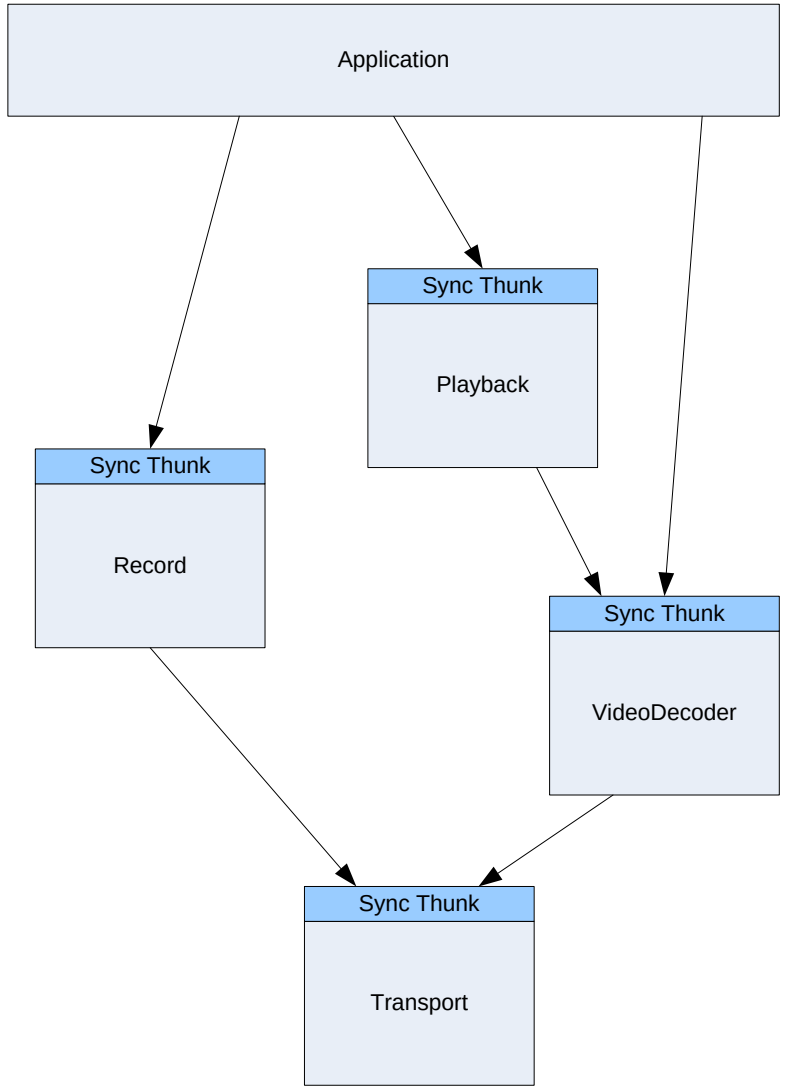


- "Thunking" gives Nexus elements of a 4GL language without the overhead.
- Thunk for ioctls to kernel mode driver
- Thunk for sockets to user mode server
- Thunk for enforcing power management standby mode

BROADCOM

Nexus Modules and Interfaces

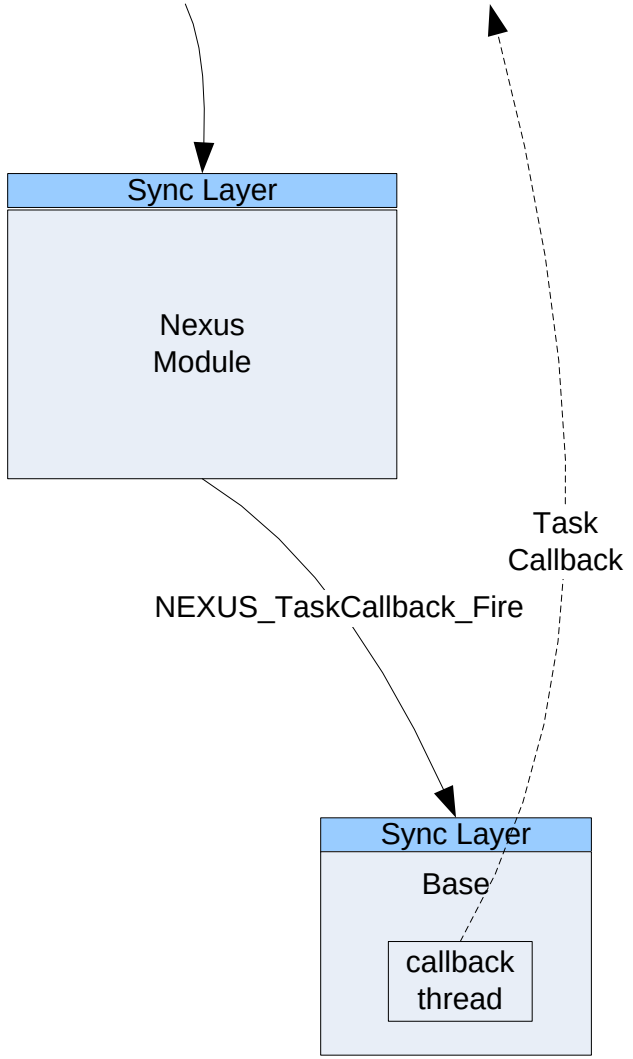
- How can the system be safe and efficient?
  - If each module automatically locks, how do we avoid deadlocks?
  - If each module automatically locks, how do we avoid fast modules waiting on slow modules?
- Modules arranged in a tree
  - Fast modules can't call slow modules, making the system efficient.
  - 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.



BROADCOM

Nexus 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 can fire callback directly from ISR context.
  - Application can make almost any call from inside a callback with no re-entrancy in Nexus modules.
  - Nexus close functions automatically synchronize with an interfaces callbacks. This eliminates very tricky race conditions.



BROADCOM

Nexus Coding Convention

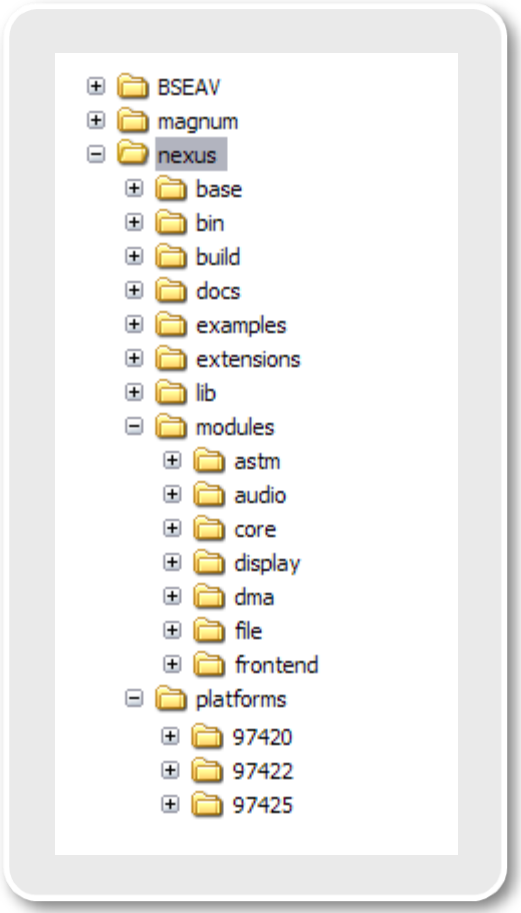
- Consistent naming convention
  - “NEXUS_” prefix establishes a namespace
  - CamelCase for consistency
  - Interface name included in every member of the interface (e.g. NEXUS_VideoDecoder_Open)
- Interface patterns
  - Open/Close, Create/Destroy
  - GetDefaultSettings/GetSettings/SetSettings
  - GetBuffer/ReadComplete
  - 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

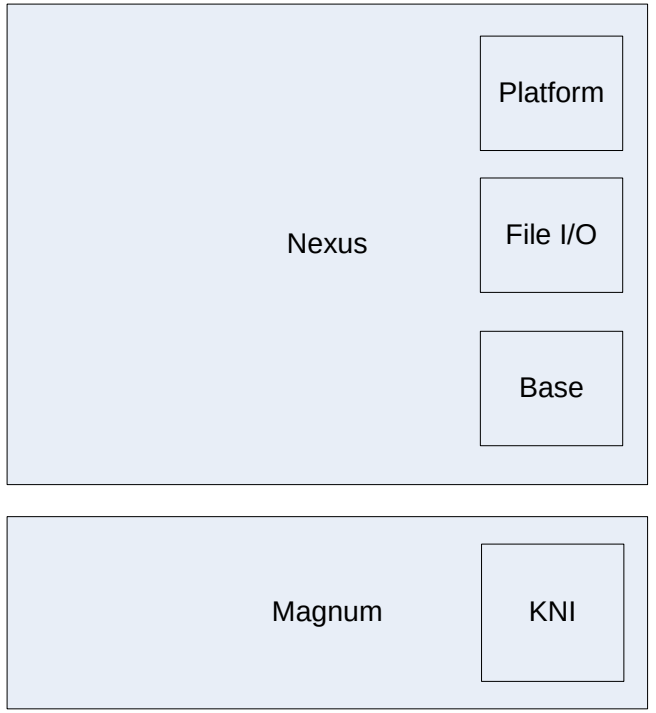
- Nexus Platform allows customer- and board-specific code to be separated from chip-specific code
  - Maximizes code reuse
  - Allows full customization
- Typical features include
  - OS driver code
  - Pin-muxing
  - L1 interrupt mapping from OS
  - Build system
  - Frontend configuration
  - DAC/ADC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



BROADCOM

Nexus OS Abstraction

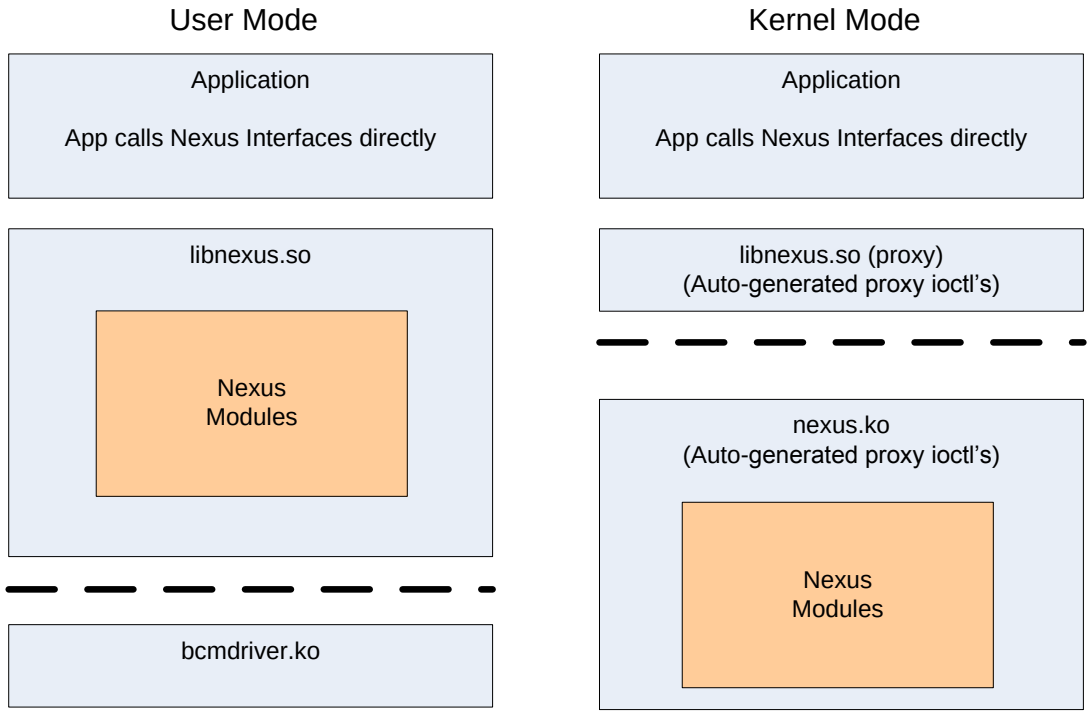
- OS abstraction provided in layers
  - Only provide the minimum required at each point for the architecture
  - Nexus currently supported on Linux, VxWorks, Nucleus, uC-OS, and WinCE
- Nexus Platform
  - Memory mapping
  - L1 interrupt mapping
  - OS driver(s)
- Nexus File
  - DVR file I/O
- Nexus Base
  - threads, timers, event callbacks
- Magnum KNI
  - events, mutexes, critical section



BROADCOM

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 worst-case interrupt latencies for connected products using Ethernet, USB, etc.
- User mode advantages
  - Easier development & debug
  - Full stack traces



BROADCOM

MIPS and ARM Support

- Nexus supports both MIPS and ARM seamlessly
  - Same Nexus API
  - All Broadcom app and library development runs on both.
- Changes the user will notice
  - Different kernel and toolchain. Require Linux 3.3 and beyond.
  - Moved from CFE to BOLT bootloader. Makes use of device tree for better configuration.
  - Set NEXUS_PLATFORM=97xxx, B_REFSW_ARCH=arm-linux, then compile.
- Internal differences that may not be noticed
  - MIPS uses proprietary BMEM system to carve out device memory; ARM uses standard CMA. Nexus heaps map to both.
  - MIPS uses 2/2 user/kernel space split; ARM uses 3/1. Less virtual memory in the kernel, more in user space.
  - MIPS Linux had standard cache flush syscall; ARM Linux does not. Nexus provides one.

Nexus Power Management

- Dynamic Power Management
  - Implicitly power down unused resources. No special API required.
    - NEXUS_VideoDecoder_Stop powers down the video decoder
    - NEXUS_Display_RemoveOutput powers down the DAC's
  - Linux support for dynamic power down of peripherals includes USB, SATA, Ethernet, etc.
- Active and Passive Standby
  - Active Standby leaves CPU running along with a subset of modules, typically frontend and transport
  - Passive Standby powers down all devices (except wake-up devices) and puts the CPU to sleep
  - Variety of wake-up devices: IR, front panel, UHF, HDMI CEC, Wakeup-on-LAN, USB, GPIO, timer
- Government Regulations and Standards
  - Energy Star (US)
  - European regulation for Simple STBs
  - European VIA (Voluntary Industry Agreement) for Complex STBs

Power states and Sample measurements

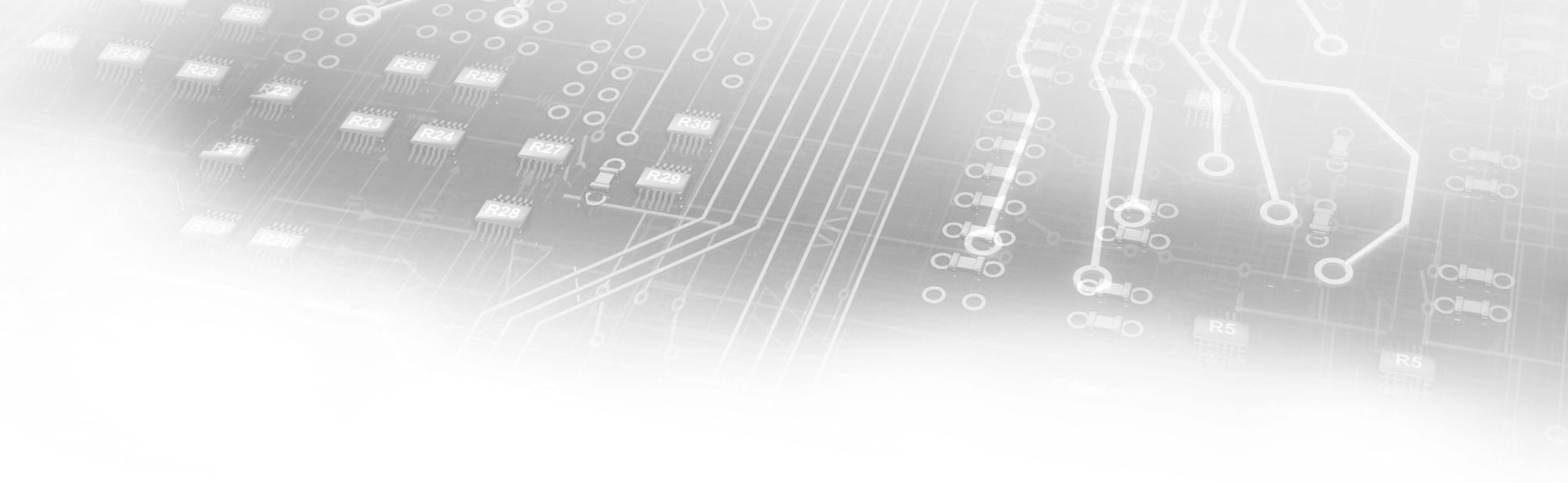
ACPI Power State	Nexus Standby Mode	Functionality	Sample power level (7425)	Sample wake up time (7425, local DVR, no SATA spin down)
S0	eOn	Full operation	4.4 W	N/A
S1	eActive	Frontend, transport and CPU active for DVR record, network messages and EPG data.	1.1 W	2.5 sec
S2	ePassive	All wake-up devices are available. All other hardware is clock gated for faster wake-up. DDR in self-refresh.	0.53 W	2.95 sec
S3	eDeepSleep	Only wake-up devices are active using Always On (AON) block: IR, timer, CEC, GPIO and keypad. All other hardware is power gated for minimal power. DDR in self-refresh.	0.013 W	3.63 sec

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

BROADCOM

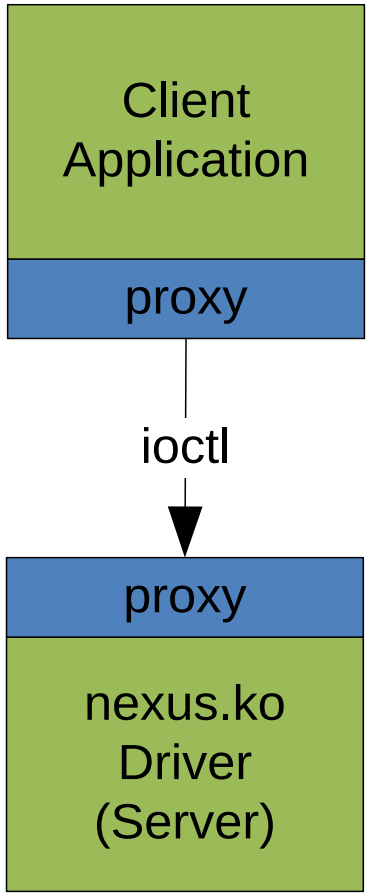


Nexus Multi-Process

Linux Kernel mode Multi-process

Auto-generated ioctl calls

```
NEXUS_MessageHandle NEXUS_Message_Open(unsigned index,
    pSettings)
{
    MessageOpenIoctl data;
    data.index = index;
    data.pSettings = pSettings;
    ioctl(fd, IOCTL_MESSAGE_OPEN, &data);
    return data.returnCode;
}
```

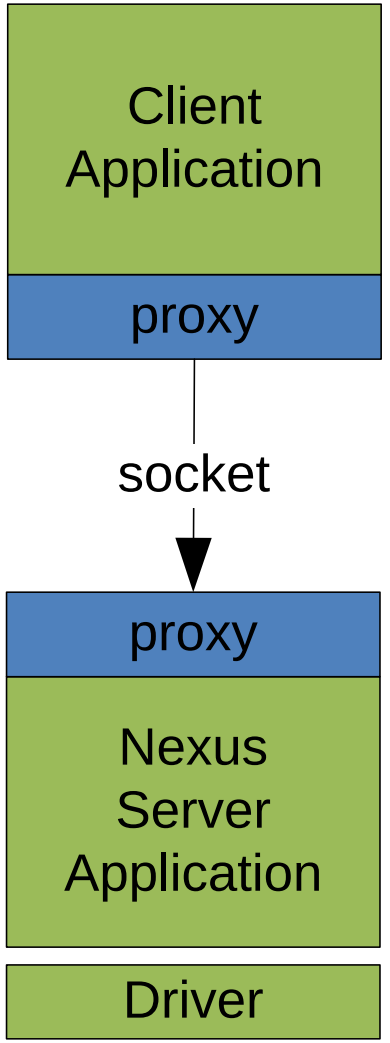


BROADCOM

Linux User mode Multi-process

Auto-generated socket calls

```
NEXUS_MessageHandle NEXUS_Message_Open(unsigned index,
    pSettings)
{
    MessageOpenData *data = LockBuffer(connection);
    MessageOpenOutParams outparams;
    data->index = index;
    data->settings = *pSettings;
    SendMessage(connection, data, &outparams);
    return outparams.returnCode;
}
```

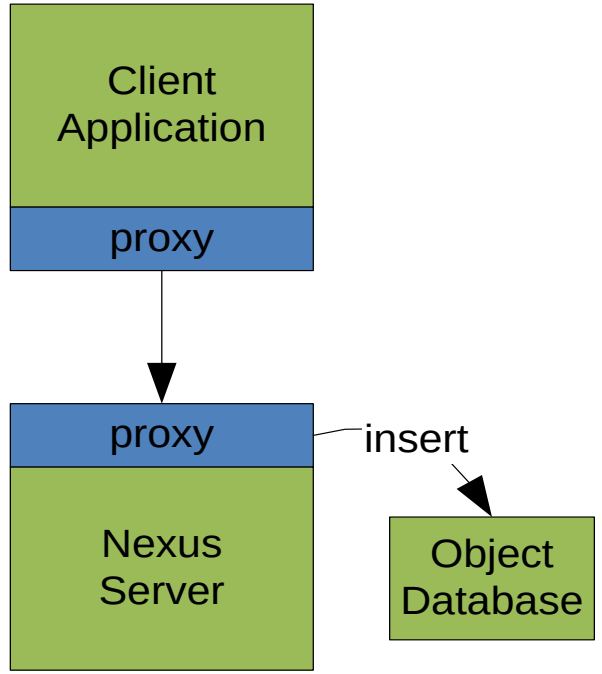


BROADCOM

Resource Tracking in the Proxy

Store handle/type/client on every Open/Create call

```
NEXUS_MessageHandle NEXUS_Message_Open(unsigned index, pSettings)
{
    NEXUS_MessageHandle msg;
    msg = NEXUS_Message_Open_impl(index, pSettings);
    if (msg) {
      b_objdb_insert(msg, MESSAGE, b_client());
    }
    return msg;
}
```

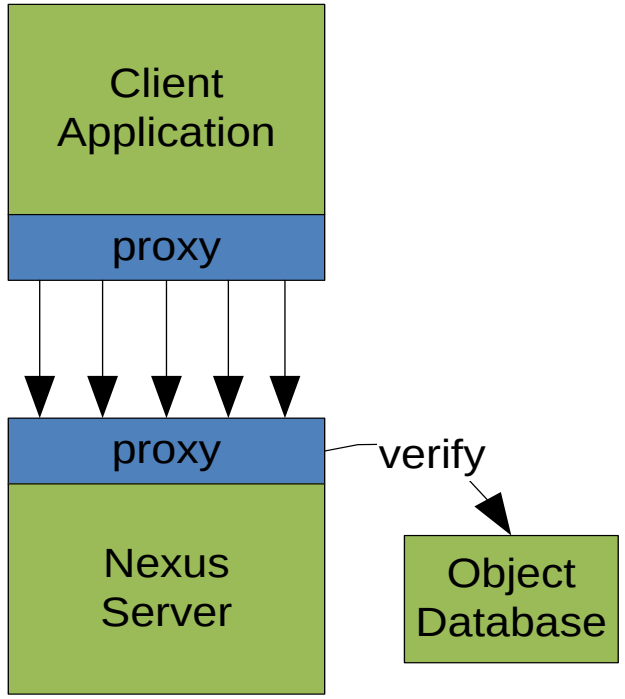


BROADCOM

Protecting the Server

Validate handle/type/client on every other call

```
int NEXUS_Message_Start(NEXUS_MessageHandle msg,
    pSettings)
{
    if (!b_objdb_verify(msg, MESSAGE, b_client()) {
       return NEXUS_INVALID_PARAM;
    }
    return NEXUS_Message_Start_impl(msg, pSettings);
}
```



BROADCOM

Crash Recovery

Client starts up…

```
--- 08.132 b_objdb: insert NEXUS_SurfaceClient:0x74d668 client=0x66800c
--- 08.132 b_objdb: acquire NEXUS_SurfaceClient:0x74d668 client=0x74c550
--- 08.134 b_objdb: insert NEXUS_Graphics2D:0x7524b8 client=0x74c550
--- 08.135 b_objdb: insert NEXUS_Surface:0x753c40 client=0x74c550
--- 08.161 b_objdb: insert NEXUS_Surface:0x753e00 client=0x74c550
```

…then crashes

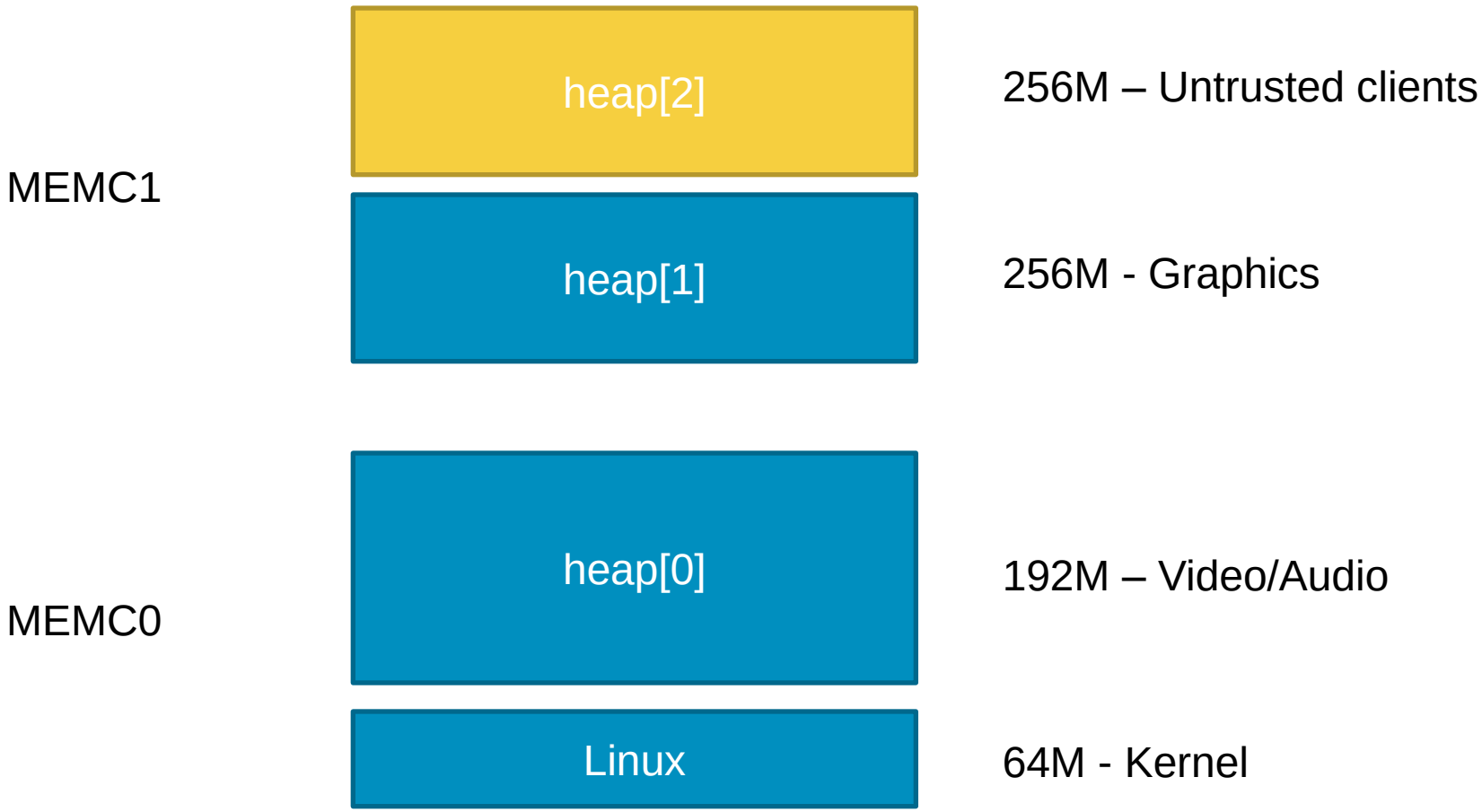
```
--- 09.964 b_objdb: auto-release: [order 1] NEXUS_SurfaceClient:0x74d668 client=0x74c550
--- 09.964 b_objdb: remove NEXUS_Surface:0x753e00 client=0x74c550
--- 09.965 b_objdb: auto-remove: [order 1] NEXUS_Graphics2D:0x7524b8 client=0x74c550
--- 09.965 b_objdb: auto-remove: [order 5] NEXUS_Surface:0x753c40 client=0x74c550
--- 09.965 b_objdb: remove NEXUS_SurfaceClient:0x74d668 client=0x66800c
```


Protecting dependencies

Reference counting dependencies

```
int NEXUS_Message_Start_impl(NEXUS_MessageHandle msg,
    pSettings)
{
    if (!pSettings->pidChannel) {
       return NEXUS_INVALID_PARAMETER;
    }
    NEXUS_OBJECT_ACQUIRE(pSettings->pidChannel);
    ConfigPidChannel(pSettings->pidChannel->index);
    ConfigMessageFilter(msg->index);
    return NEXUS_SUCCESS;
}
```


Secure Memory Management



BROADCOM

Nexus Client Modes

Feature	Verified Mode	Protected Mode	Untrusted Mode
API	Full	Full	Limited
Handle verification	Partial (skip owner test)	Yes	Yes
Garbage collection	Yes	Yes	Yes
Heaps	Only what is granted	Only what is granted	Only what is granted
Use Case	Server, Status clients	In-house clients	Third-party clients

Safely Sharing Resources with Clients

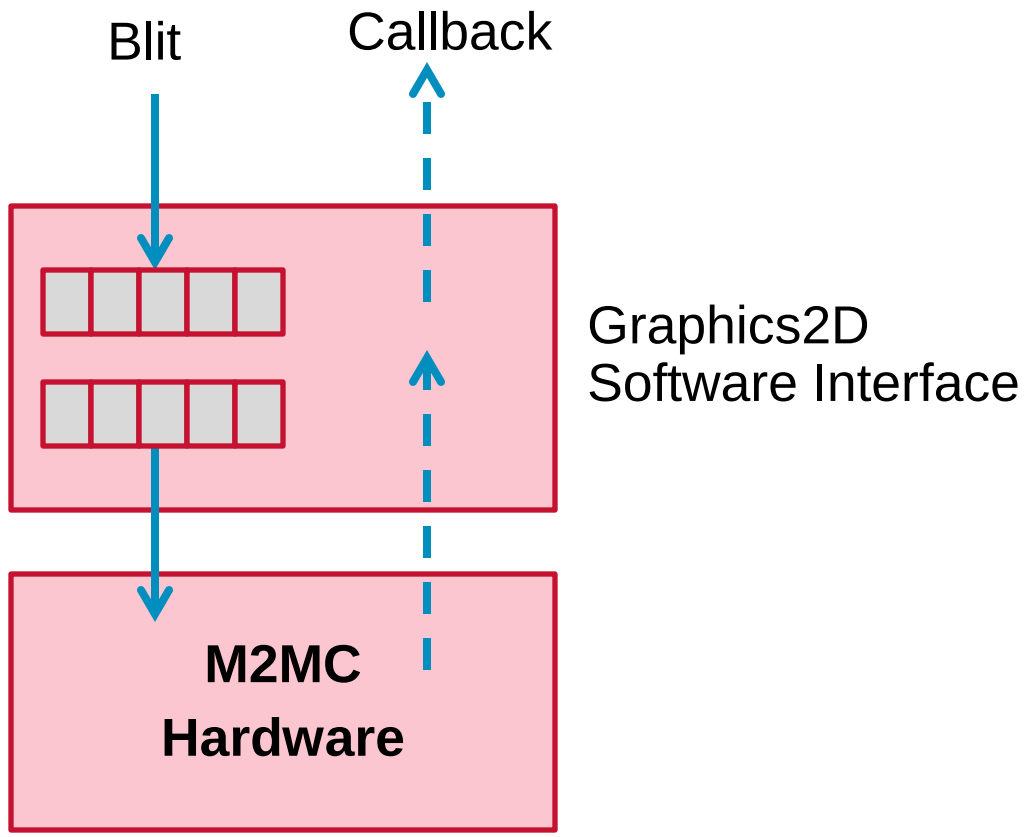
- Abstraction for resource sharing
  - An abstraction is a higher-level representation which hides implementation details
  - Solve a problem in one spot instead of spreading it to all spots

- Multiple abstractions are useful
  - Virtualization
  - Dynamic allocation
  - Higher-level client/server API’s

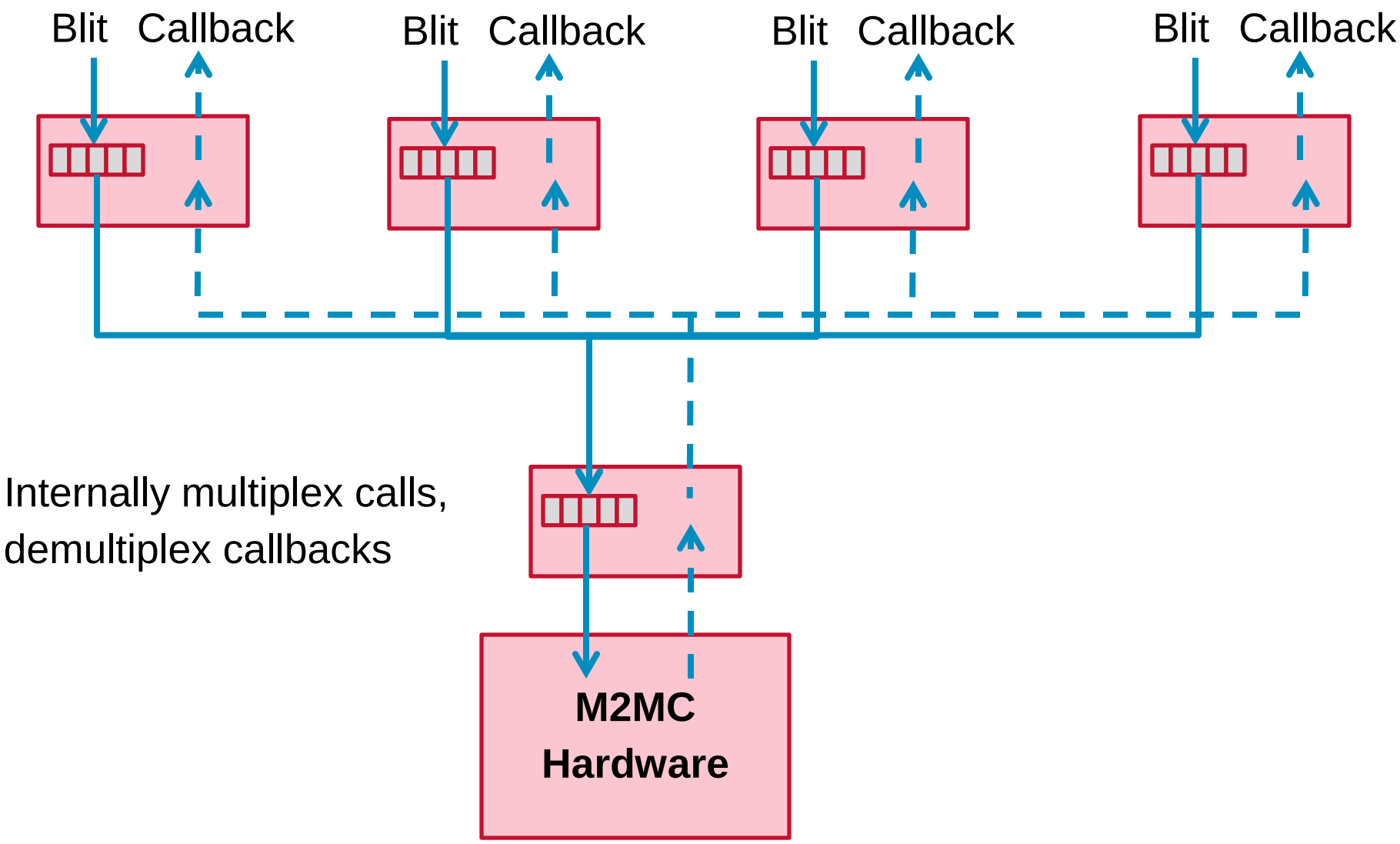
Solution #1 - Virtualization

- Rework implementation so each process thinks it owns the resource, then multiplex in software.

- Example
  - 2D graphics blitter



Solution #1 – Virtualization (cont.)



Solution #2 - Dynamic Allocation

- Provide dynamic allocators for plentiful and orthogonal resources
- Examples
  - Pid channels, message filters, playbacks, records, parser bands

```
p = NEXUS_Playpump_Open(2, &openSettings);
NEXUS_Playpump_Close(p);
```

```
p = NEXUS_Playpump_Open(NEXUS_ANY_ID, &openSettings);
NEXUS_Playpump_Close(p);
```


Solution #2 - Dynamic Allocation (cont.)

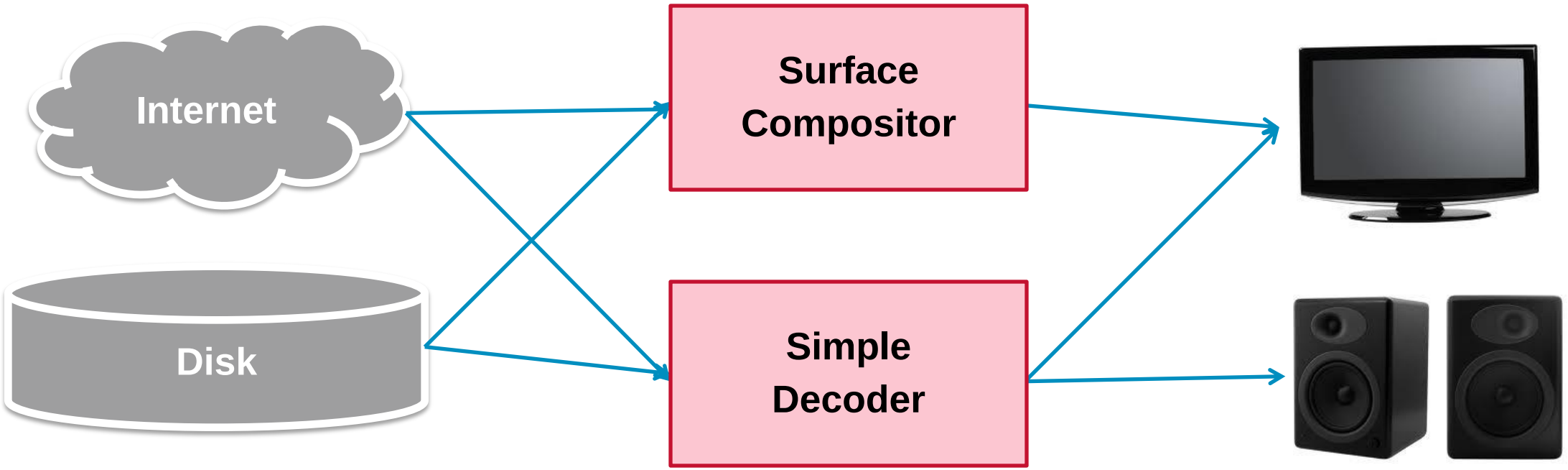
- Converting enumerated types to handles
  - Constructor/destructor needed for dynamic allocation
  - Still backward compatible

```
NEXUS_ParserBand_SetSettings(NEXUS_ParserBand_e0, &settings);
```

```
NEXUS_ParserBand pb;
pb = NEXUS_ParserBand_Open(NEXUS_ANY_ID);
NEXUS_ParserBand_SetSettings(pb, &settings);
NEXUS_ParserBand_Close(pb);
```

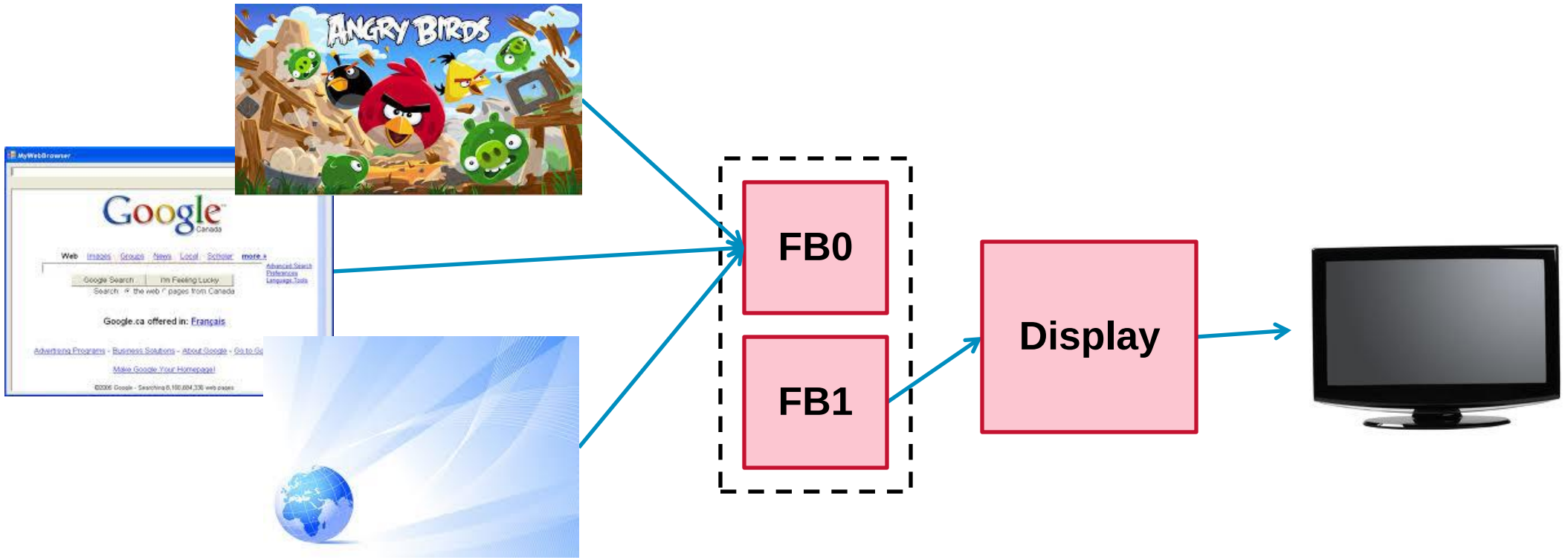

Solution #3 – Higher-level Client/Server APIs

- Multiplexing video and graphics to the display
- Multiplexing audio and sound effects to outputs



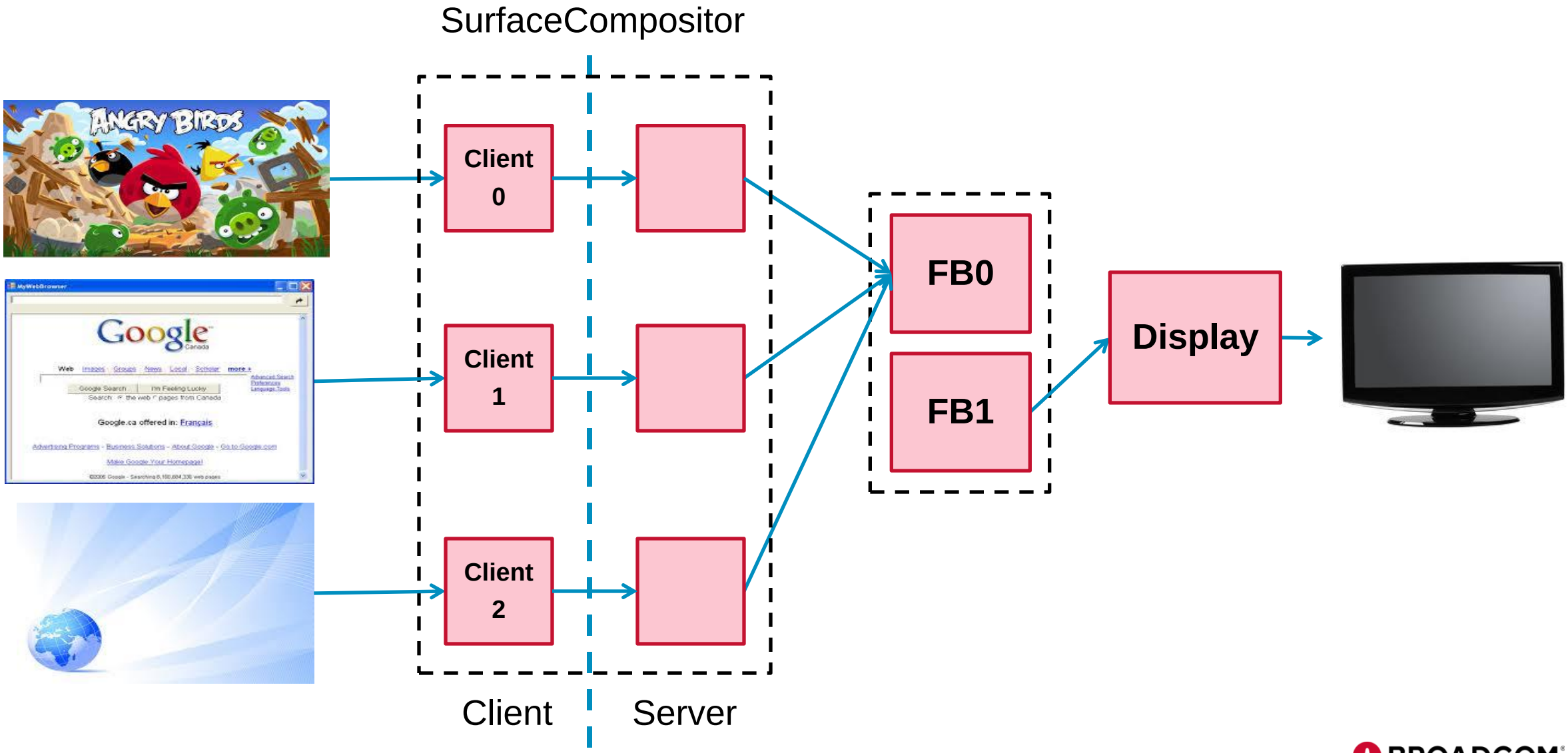
BROADCOM

Single process Graphics



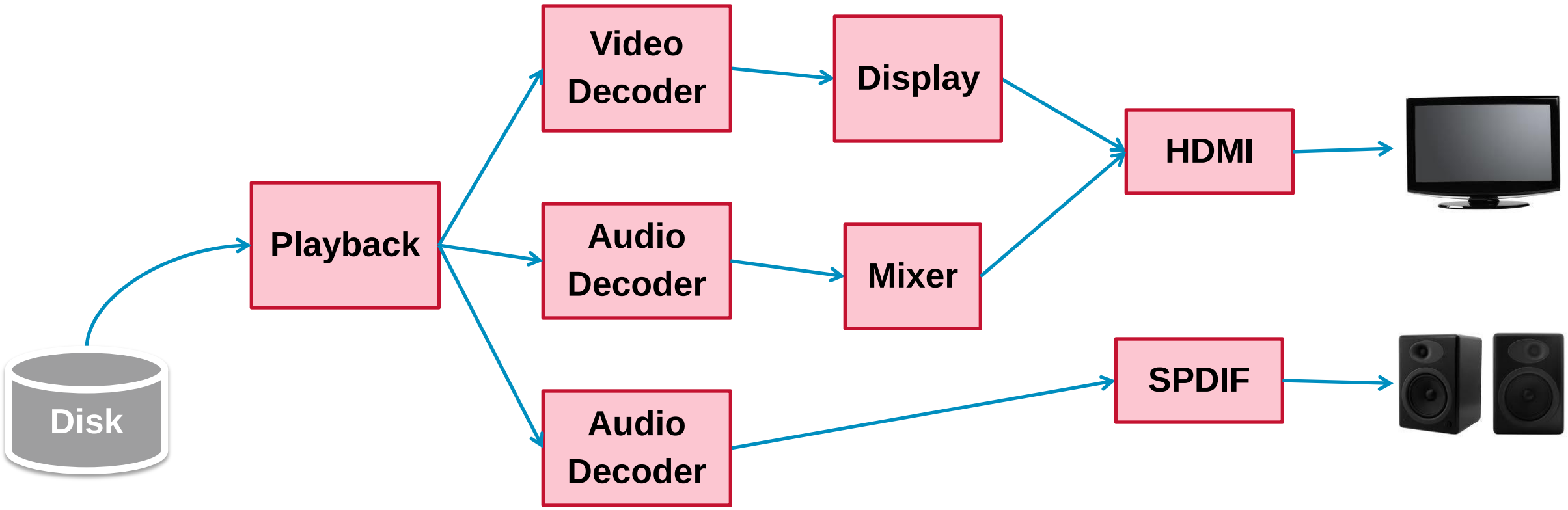
BROADCOM

Multi-process Graphics with Nexus Surface Compositor



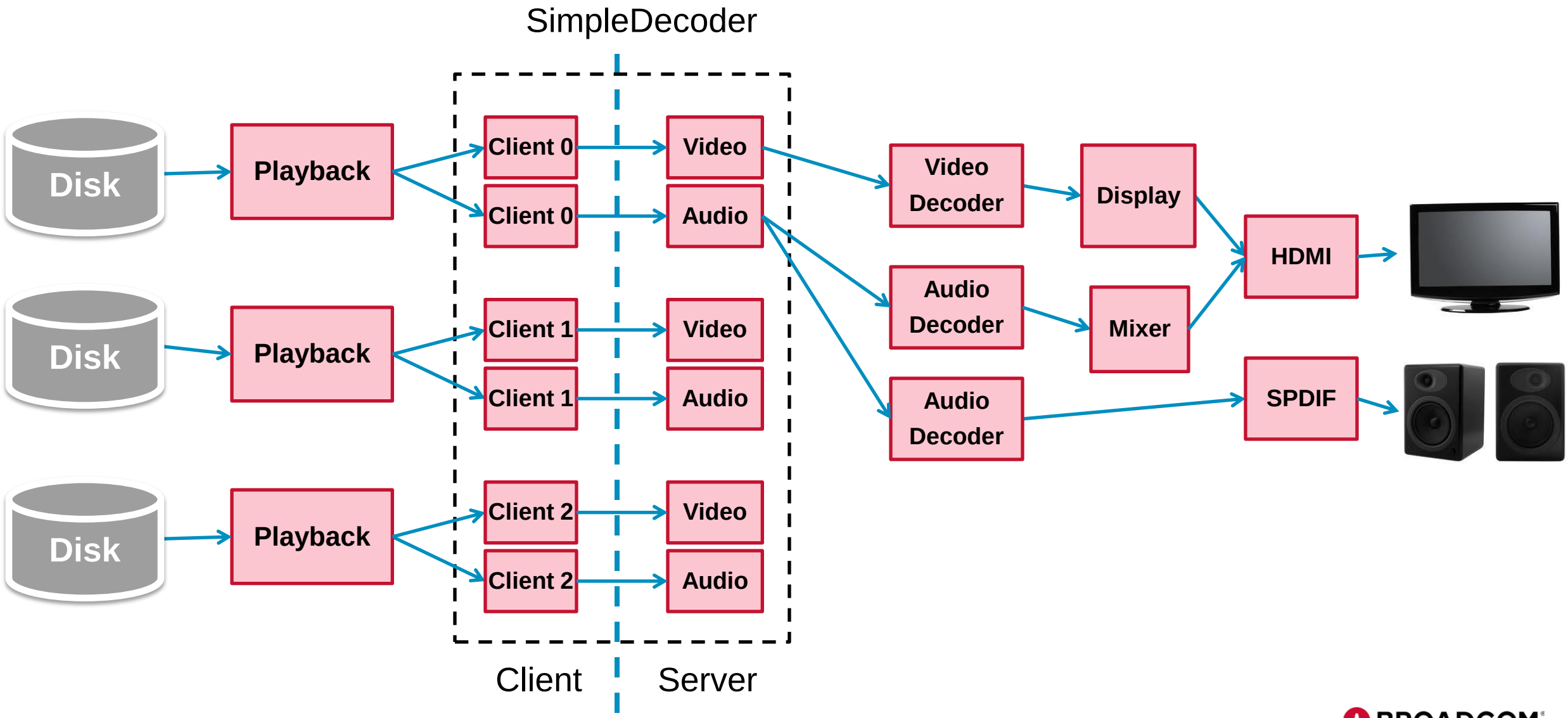
BROADCOM

Single process Decode



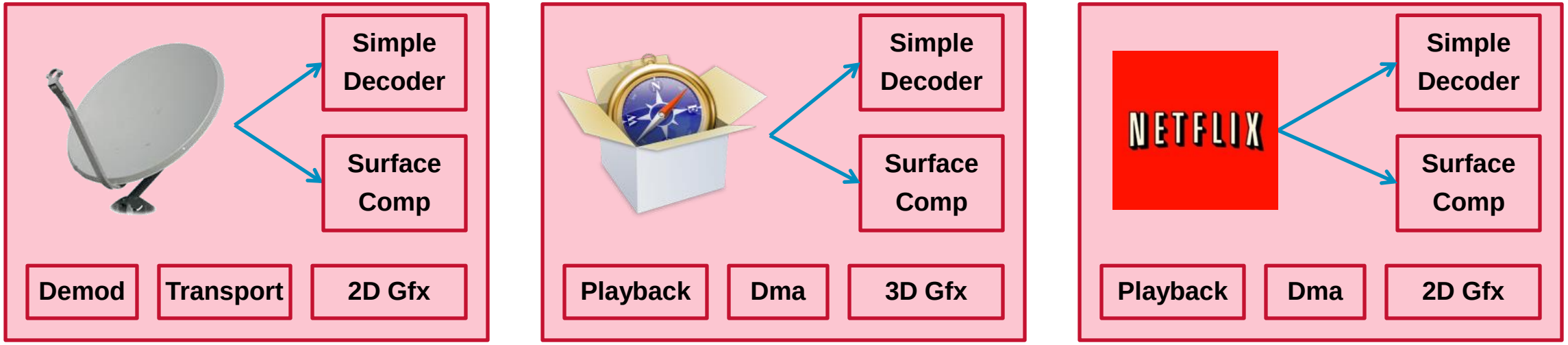
BROADCOM

Multi-process Decode with Nexus Simple Decoder



BROADCOM

Nexus Multi-process System



Nexus Library/Driver

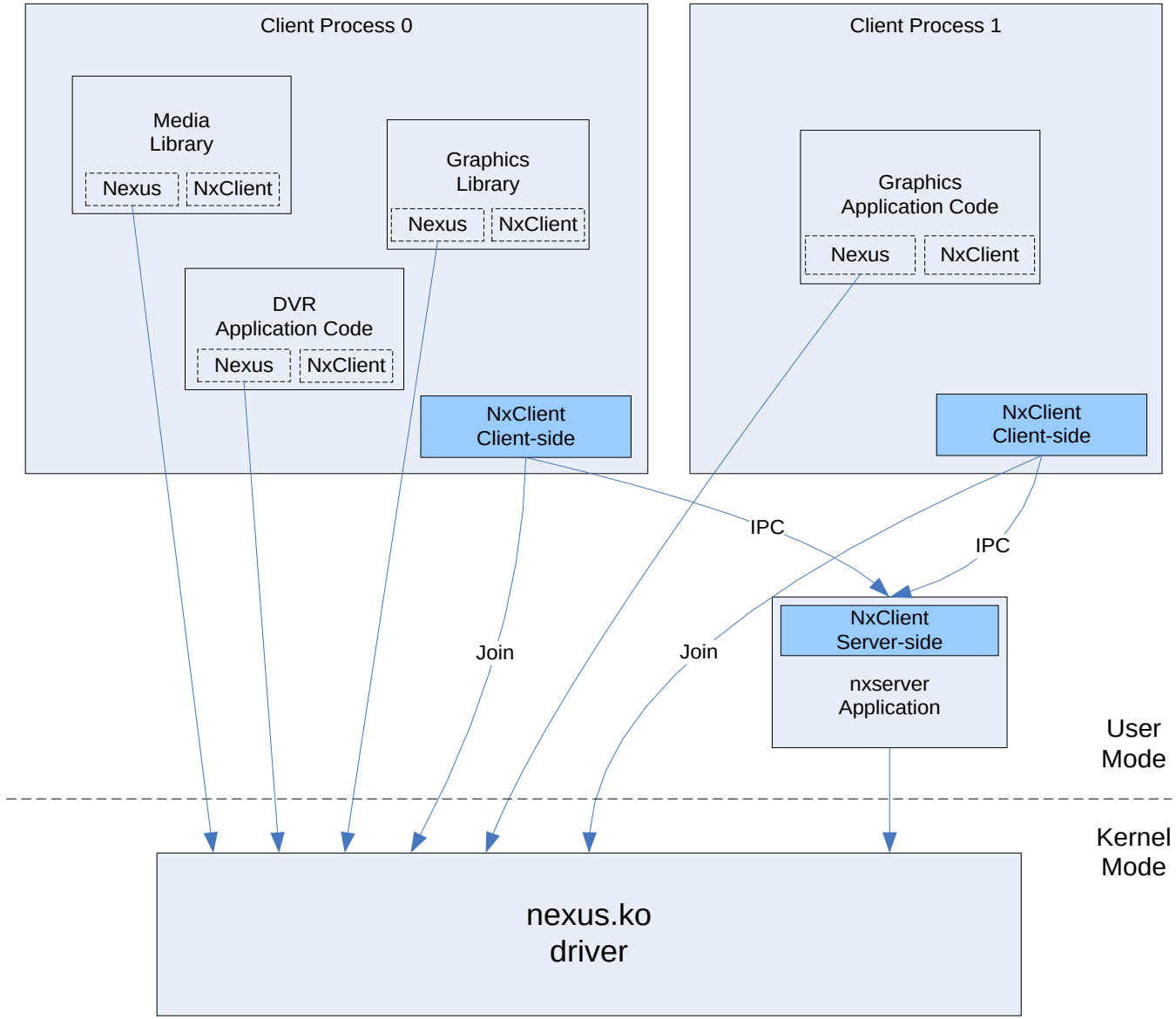
Silicon

BROADCOM

NxClient

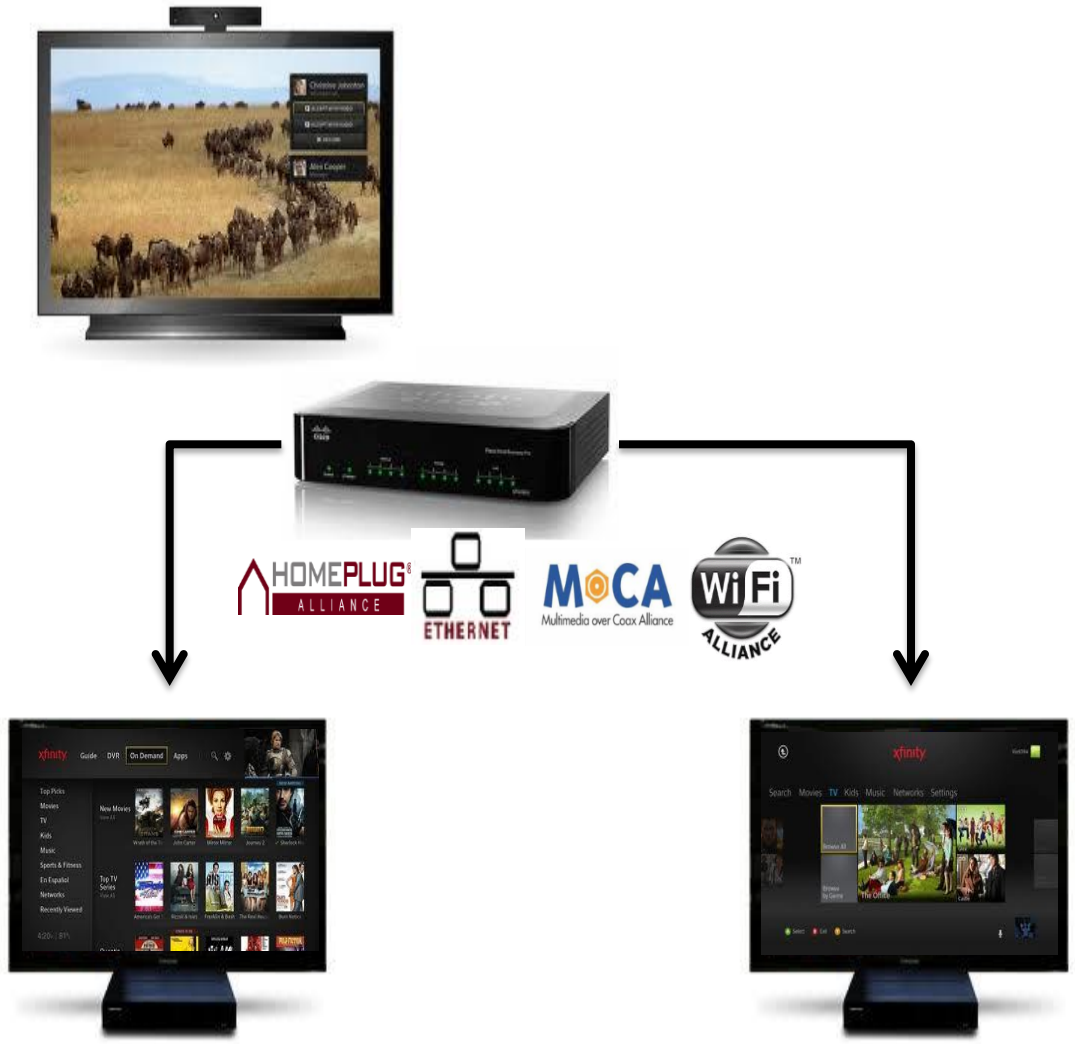
- Thin layer above Nexus Multi-Process for resource management and global settings
  - Not a wrapper of Nexus. 95% of application calls are still to Nexus.
  - Allows for client apps and libraries to be developed independently.
- Reusable nxserver application
  - HDMI EDID, HDCP algo, audio post processing including Dolby MS11/12
  - Video decoder allocation, timebase/STC allocation
  - HD/SD simul & 3DTV display
  - Transcoding backend
- Minimal API
  - NxClient_Join – connect to the server and to Nexus
  - NxClient_Alloc – ask the server to create resources which can be acquired
  - NxClient_Connect – ask the server to connect video and audio decoders
- See nexus/nxclient/README.txt
  - Learn by running sample apps in combination

NxClient



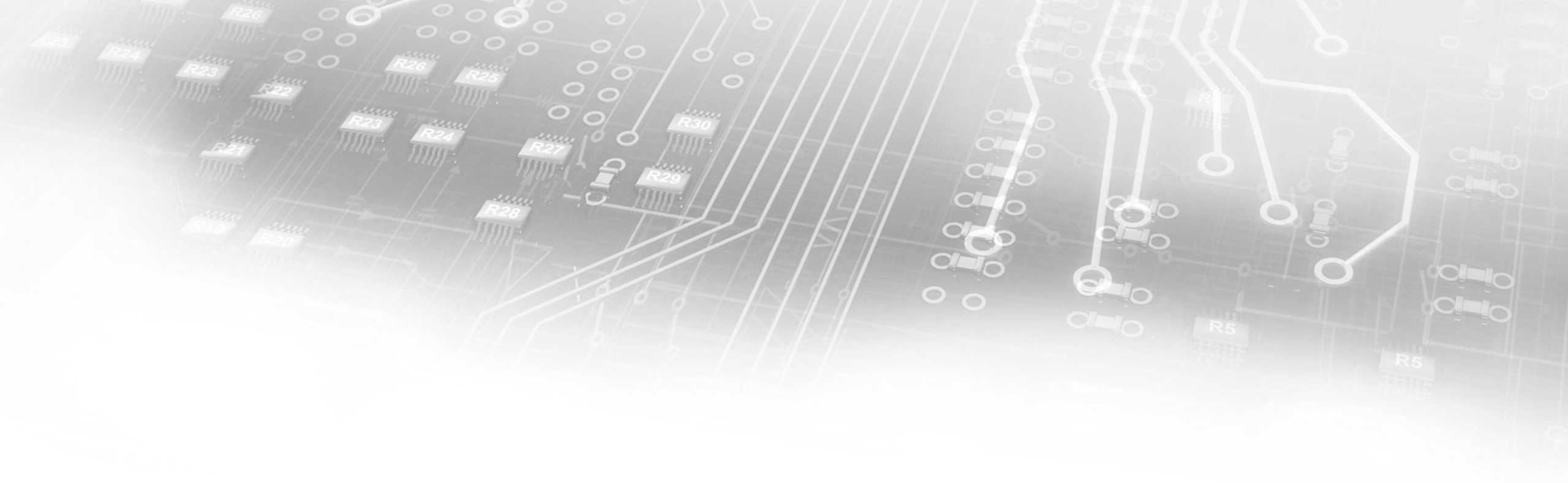
BROADCOM

NxClient Multi-Session



- Server runs multiple sessions with separate graphics/video/audio presentation
- Decode and encode resources are shared
- Remote sessions can be streamed via Miracast or other standard
- Server can be headed or headless

BROADCOM



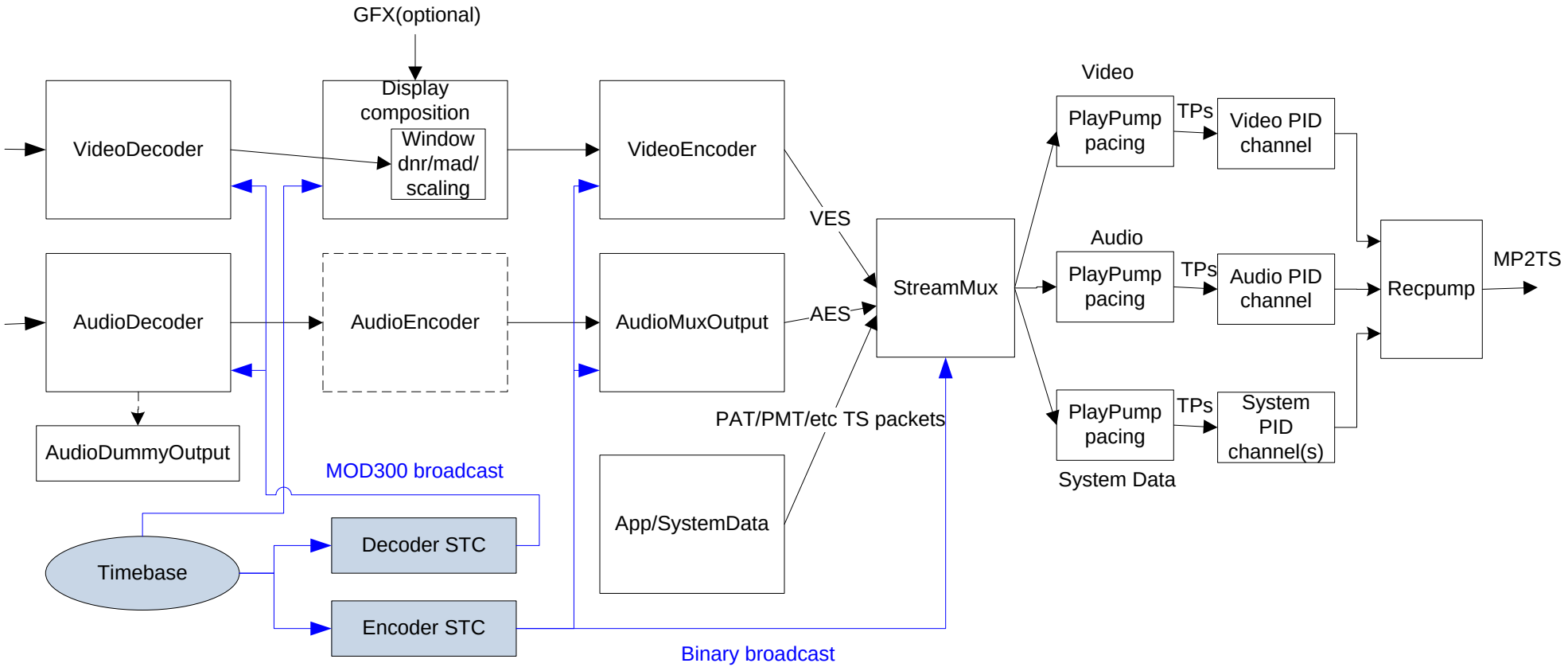
Nexus Advanced Usage

Advanced Usage

- Audio/video transcode
- Packet blit
- Fast channel change
- 3DTV support
- 3D graphics using OpenGL-ES 2.0
- Large memory systems
- AppLibs

Audio/Video Transcode

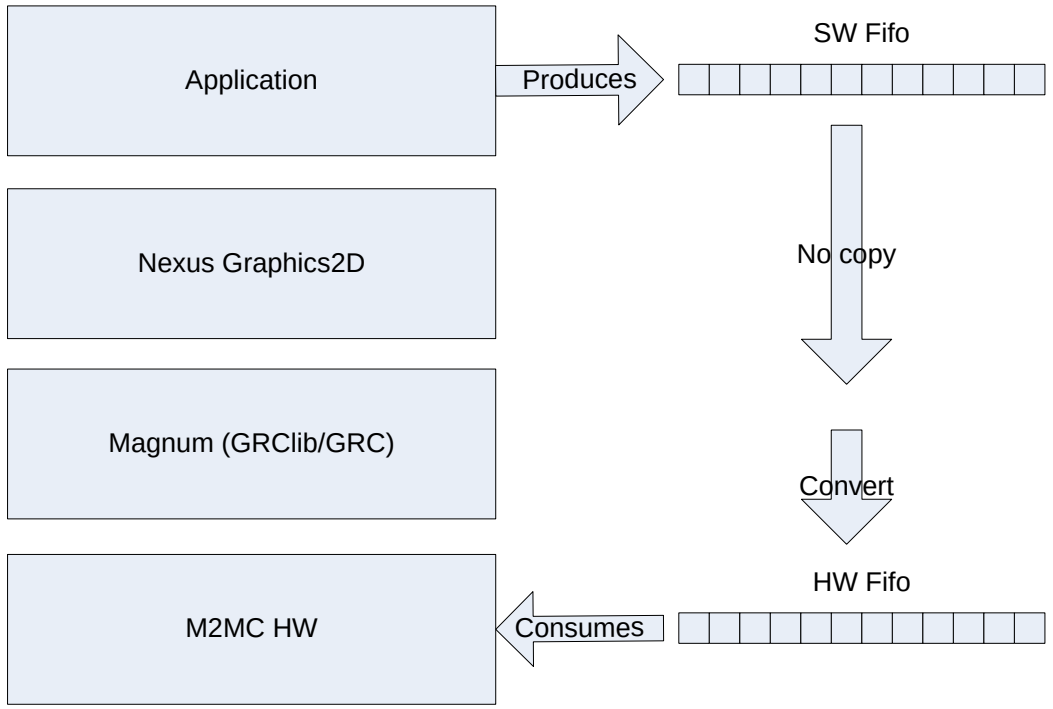
- Video transcode through video decode (AVD) and video encode (VICE)
  - MPEG/VC1/AVC/MPEG4, up to 1080p30
- Audio transcode through audio decode and encode (Raaga)



BROADCOM

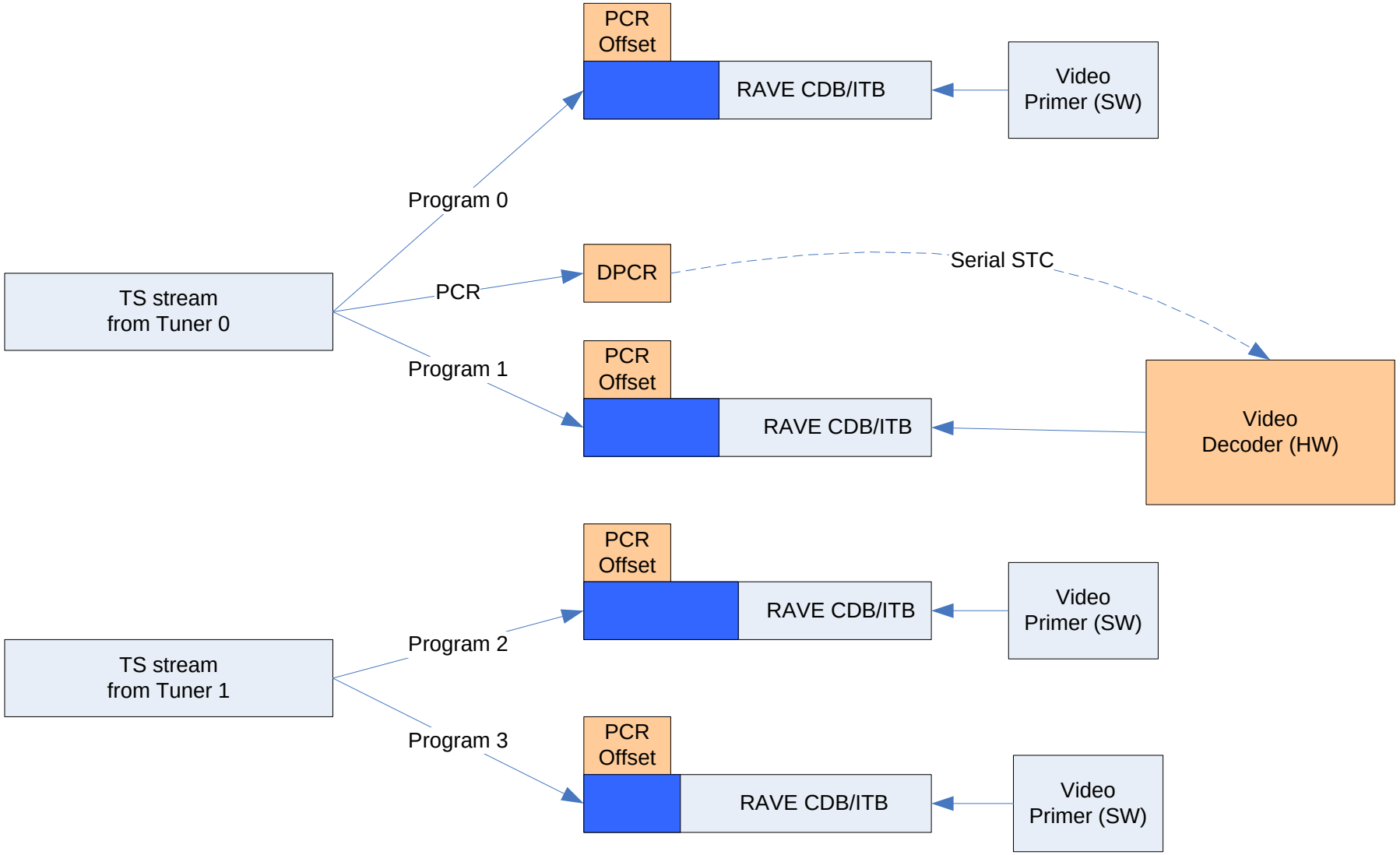
Nexus 2D Packet Blit

- 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)

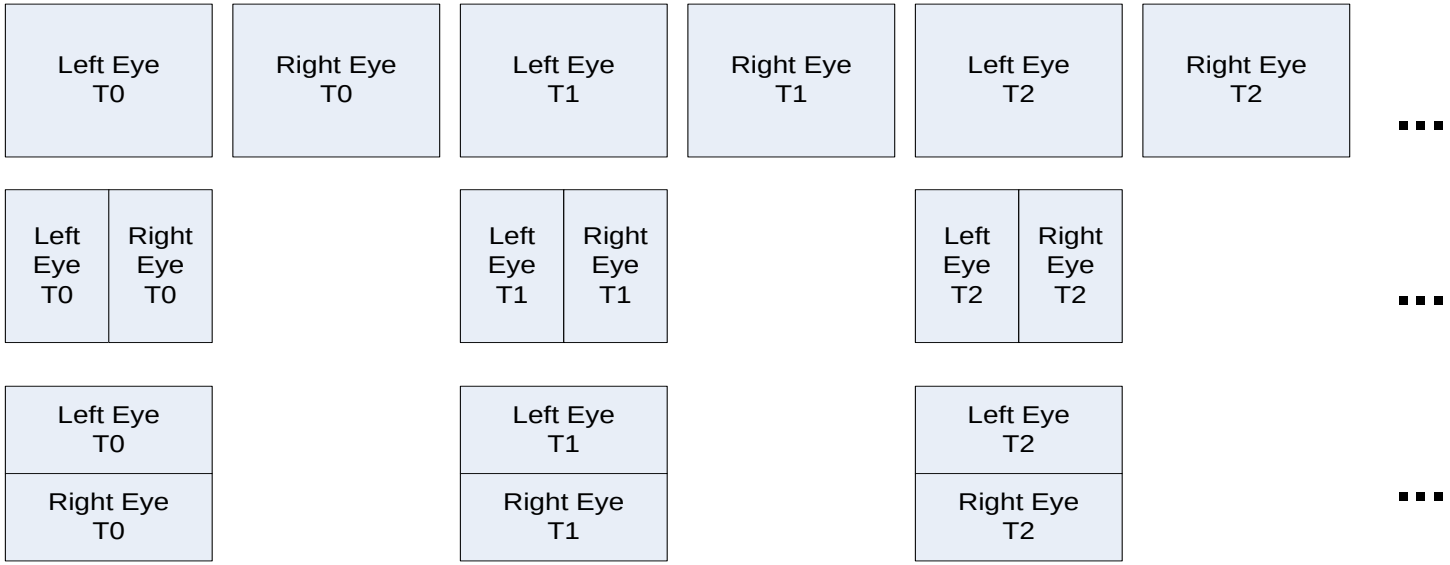
Fast Channel Change



BROADCOM

3DTV (Stereoscopic TV)

- Integrated 3DTV support starting with 40nm silicon
  - 65nm silicon has subset of functionality, more application code required
- Full-res decode 1080p24 or 720p60 for each eye
- Multi-view Video coding (MVC) decode can be used for 3DTV streams
- Integrated L/R graphics support
- HDMI VSI, AVC SEI signaling support



BROADCOM

OpenGL 3D Graphics

- 3D graphics through Video Core IV and V (VC4 and VC5) cores
- 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

BROADCOM

Large Memory Systems

- Limited virtual address space in 32 bits
  - Kernel/user split is 2G/2G for MIPS; 1G/3G for ARM. Yet we deploy 2-3 GB systems.
  - Must map memory according to use case.
- Nexus provides full heap control
  - Minimal mapping of each memory region
  - Default heaps provided; fully customizable

NEXUS_MemoryType	Memory Mapping
NEXUS_MemoryType_eApplication	User mode CPU access
NEXUS_MemoryType_eFull	Full kernel and user mode CPU access
NEXUS_MemoryType_eDeviceOnly	No mapping. No CPU access required.
NEXUS_MemoryType_eSecure	No mapping. Limited device access.

Trellis and 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

Thank You


BROADCOM®
connecting everything®