TI Developer Conference
March 7-9, 2007 • Dallas, TX

Minds in Motion

Introduction

Embedded DSP Systems Considerations with Audio Signal Processing

Agenda

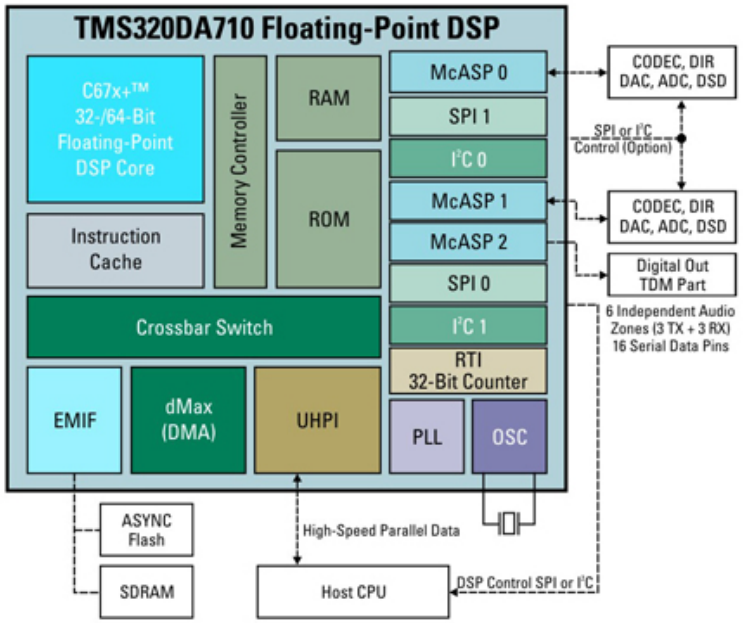
- Introduction
- DSP Architecture Background
- Real-time Operating Systems (RTOS)
- Memory Management
- Frameworks
- Control Architecture
- Device Interfaces
- Multi-zone Architecture
- Mixed Audio/Video Architectures

Introduction

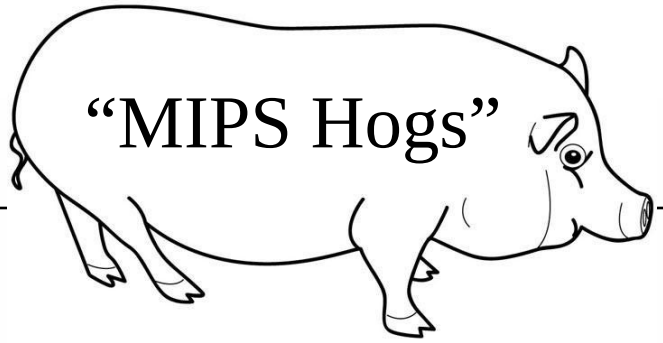
- Why is this topic relevant?
  - More complex SOC than ever before, integration of other functions in the system
  - “MIPS Hogs” (those algorithms you know and love)
  - More complex peripherals and signal routing
  - Time-to-market and Productivity
  - Software quality
- What products are we talking about?
- What are the challenges for Embedded Audio Systems? How have those challenges evolved over the last 10 years?

SOC

- More complex SOC than ever before, integration ↑
  - Device architectures capable of not only “traditional” signal processing (MAC), but also contain complex peripheral mix (DIR, DIT, serial I/O for data/control, DMA, cache, external memory I/F, USB, SRC, etc.)



"MIPS Hogs"



DOLBY DIGITAL
AAC
dts Digital Surround Neo:6® | 96/24 | ES®
DSD to PCM
WMA Lossless
MLP LOSSLESS
DVD AUDIO
HDCD
MP3 Decoder
ATRAC3 PLUS DECODER
mp3 PRO
mp4 AAC Decoder
Plays Windows Media™
WMA9 Decoder
DOLBY DIGITAL PLUS
DOLBY TRUEHD
dts-HD Master Audio
dts-HD High Resolution Audio
DTS-HD Express™ Product

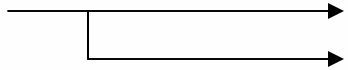
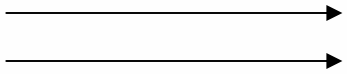
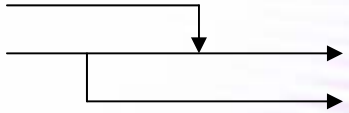
DOLBY DIGITAL EX PRO LOGIC IIx
dts Digital Surround Neo:6®
Virtualizer
2.0
DOLBY PRO LOGIC II VIRTUAL SPEAKER
2.0
Effect Library
Filter Library
Bandwidth Expander
Bass Boost
LUCASFILM THX SELECT
THX ULTRA2
MP3 ENCODER
MPEG2 AAC ENCODER
ATRAC3 PLUS ENCODER
mp4 AAC Encoder
Windows Media™ Compatible
WMA8 Encoder
aacPlus Encoder
aacPlus Decoder

Audyssey MULTEQ XT
Audyssey PREVEQ
SRS Circle Surround II
SRS TruBass
SRS WOW
neural SURROUND
DOLBY DIGITAL Pro ENCODER
dts Interactive
DOLBY DIGITAL 5.1 CREATOR

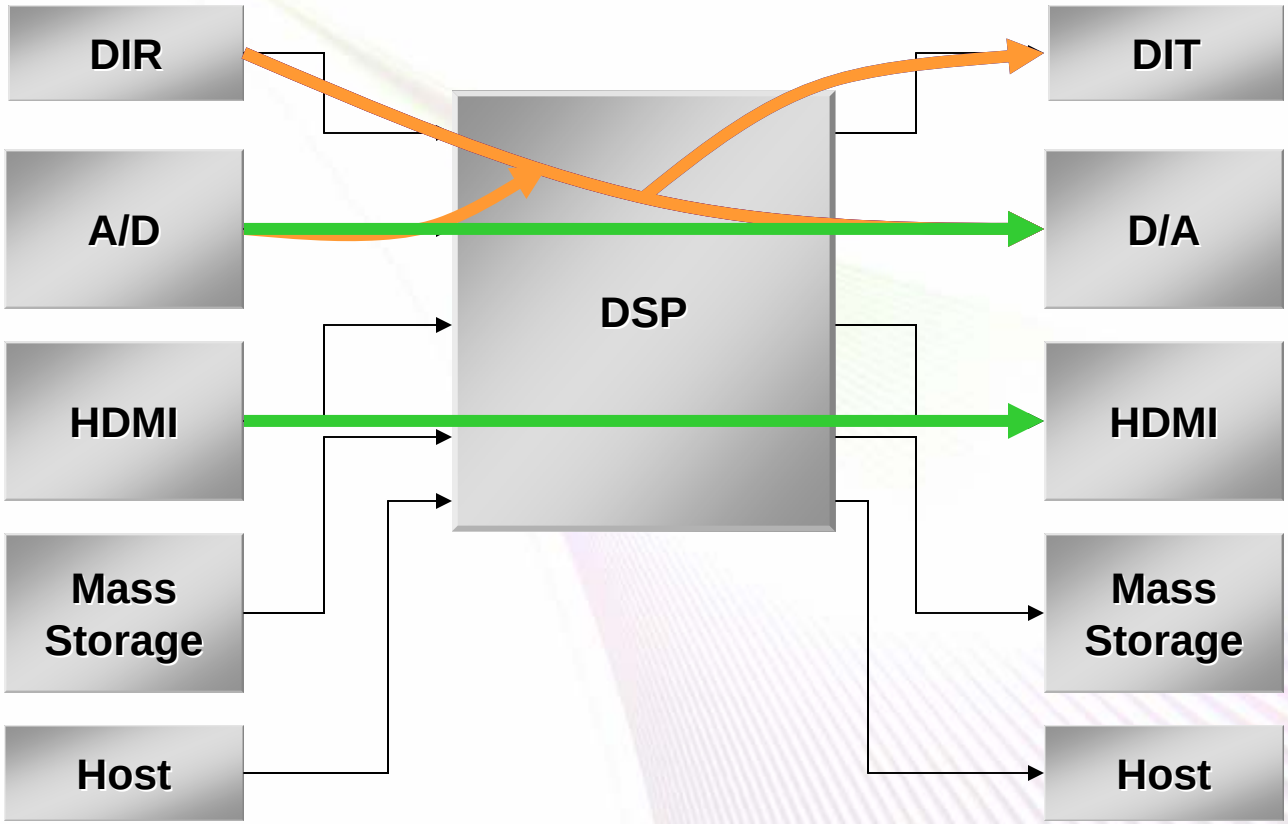
Peripherals and Signal Routing

- Peripherals
  - I/O via:
    - Serial port dedicated for audio (I2S)
    - Memory mapping (USB/MMC/SD/HDD/CD etc.)
    - Control port (SPI/I2C)
- Signal Routing
  - Topologies: number of input/outputs simultaneously available, possibility of independent streams
  - Dependence on peripheral (clock zones)
  - Deadlines
    - Real-time deadline (e.g. “pull” from D/A Converter)
    - Non-real-time (e.g. “pull” from disk)

Topologies

Topology	Description	
I	single stream	
Y	split stream (with outputs to two peripherals)	
H	2 independent I processing chains (dual zone)	
Z	primary (master) with secondary (slave) input/output	

Data Signal Routing



Time-to-Market and Productivity

- Different tools provided alongside embedded systems help speed prototyping and productization:
  - Integrated Development Environment (IDE)
  - Advanced Debugger
  - Advanced C-compilation Tools
  - Operating Systems
  - Frameworks
  - Standard APIs for algorithm, driver development
  - Evaluation Kits/Reference Designs

Consumer Products

Set-top box

AV Receiver

DVD player

HDTV

Digital Sound Projectors

HD-DVD and
Blu-ray players

Networked
AV Receiver

PC multimedia
speakers

Minds in Motion

More Consumer Products

HDD/Navigation systems

Home Media Servers

PVR/DVD Recorders

HDD Mini/Micro
Bookshelf System

CD Ripper to MP3 player/Cell Phone/
USB flash/SDMMC card

Virtual CD Changer/
Automotive Jukebox

Minds in Motion

Embedded Solutions

- General Purpose Processors (GPP/Micro)
- Digital Signal Processors (DSP)
- Application Specific Processors (ASP)
- Application Specific Integrated Circuits (ASIC)
- Field Programmable Gate Arrays (FPGA)

Challenges for Embedded Audio Systems - Then

- Balance performance vs. cost
- Word length
- Mostly fixed-point
- Keep up with features in emerging applications (Pro Audio, A/V Receiver, DVD, HDTV)
- Digital Audio / Analog Video
- Device per function (multi-DSP products common)
- MIPS challenged
- Programming in assembly language
- “Bare-deck” DSP (no software provided)

Challenges for Embedded Audio Systems - Now

- Balance performance vs. cost
- Rapidly add features and integrate (not always signal processing tasks!)
- Floating point more common
- Programming in higher-level languages
- Complex peripheral mixture
- Power
- Ease-of-use, mix of tools
- Digital Audio / Digital Video
- Software and systems provided by silicon providers
- Figure out how to use the available MIPS!

TI Developer Conference
March 7-9, 2007 • Dallas, TX

Minds in Motion

DSP Architecture

Embedded DSP Systems Considerations with Audio Signal Processing

Topics

- Audio Serial Port
- DMA Engine
- CPU
- Memories
- Control/Status Interface

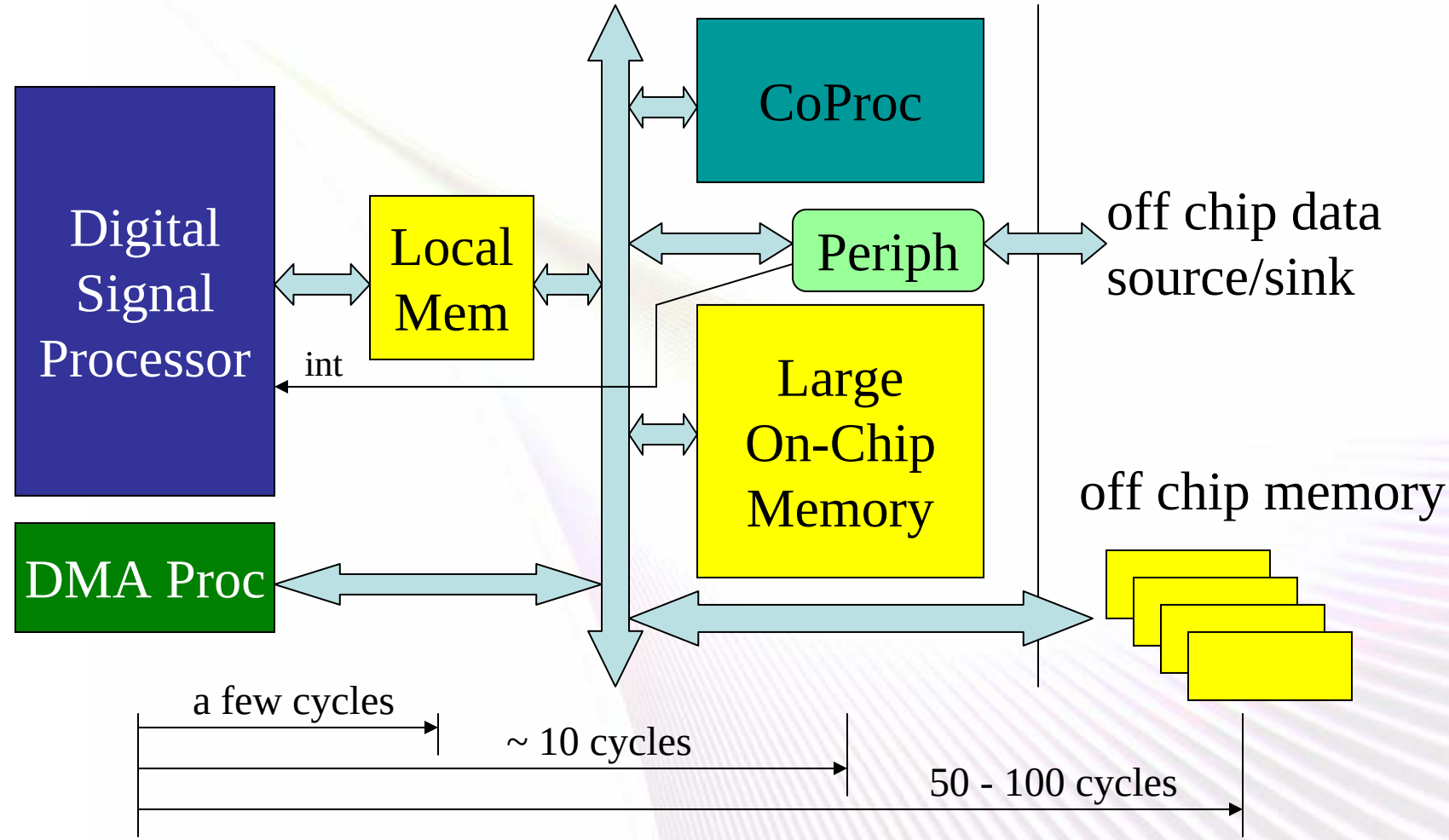
Audio Serial Port

- Purpose:
  - Gluelessly attach to audio devices such as A/D, D/A converters, S/PDIF receiver/transmitters to source and sink audio data to the DSP
- Standard Features
  - Variety of sample rates
  - Good number of input serializers
  - Different serial formats (IIS, left-justified, TDM, etc.)
- Optional Features
  - Error recovery
  - Clock generation
  - Integrated S/PDIF receiver or transmitter
  - Multiple clock “zones”
  - Parallel input mode

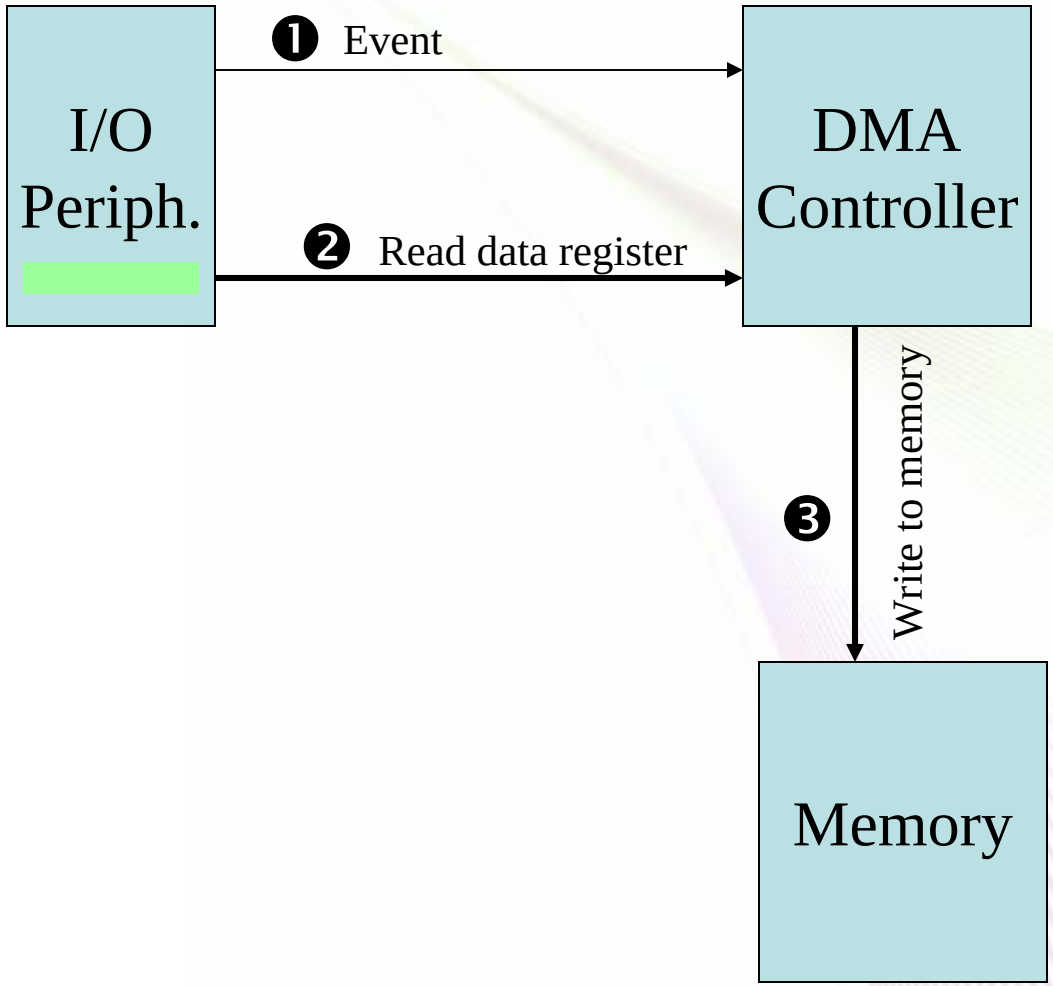
DMA Engine

- Purpose:
  - Efficiently move data/code from peripheral-to-memory or memory-to-memory without interrupting the CPU
    - Most applicable for block-based data movement
  - For external memory, latency must be considered
- Standard Features
  - A number of “channels” supporting ping/pong transfers
  - Highly configurable – supports chaining
  - DMA may be coupled with peripheral or universally available
- Optional Features
  - Circular buffer support
- Good overview of DMA: http://signal.ece.utexas.edu/seminars/dsp_seminars/02spring/DMA_TI.pdf

Model System



Peripheral DMA Flow



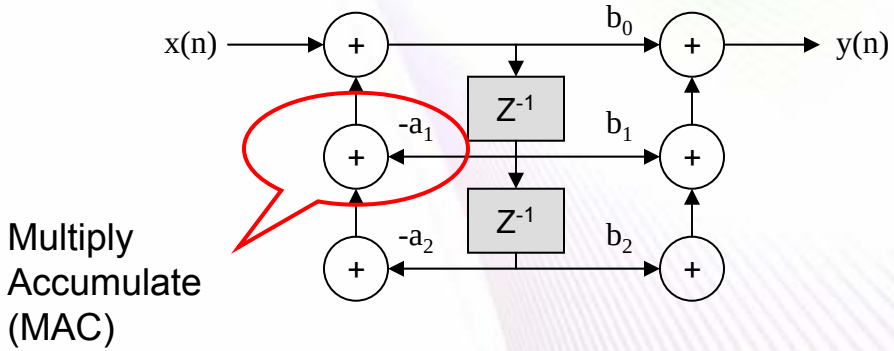
❶ Peripheral signals an event to indicate it has data ready.

❷ DMA Controller reads peripheral data

❸ DMA Controller writes data to memory

CPU

- Purpose:
  - Signal processing functions
    - Filter, scale, transform, encode/decode, correlate, etc.
    - These operations are processor and data intensive
- How is a DSP structured?



  - Around MAC (core of filter/FFT)

CPU

- Then:
  - Harvard architecture
  - Improved variants to pipeline
- Now:
  - SIMD (Single Instruction, Multiple Data)
    - Exploits data level parallelism
      - ↗ Good for vector processing
      - ↗ Often found as a companion to a SISD (Single instruction, single data)
  - VLIW (Very Long Instruction Word)
    - Exploits instruction level parallelism
    - Execute multiple instructions/cycle and use simple, regular instruction sets
      - ↗ More parallelism, higher performance
  - Superscalar
    - Can issue varying numbers of instructions per cycle
    - Can be scheduled either statically by the compiler or dynamically by the processor
  - Hybrids of the above

Memories

- Purpose:
  - Design is predicated on speed
    - Data/instructions must flow into numeric and sequencing sections on every instruction cycle with minimal (or no) delay
    - Memory system focused on throughput
- Internal Memory
  - RAM
    - Data only
    - Code + data shared
    - Cache (Level 1, 2)
  - ROM
- External Memory
  - SRAM
  - SDRAM
  - DDR
- Memory bus

Control/Status Interface

- Purpose:
  - Provide an interface to external host (uC) or peripheral devices (ADC, DAC, etc.)
  - Typically defined by a physical and logical protocol
- Standard Features
  - Support industry-standard protocols (IIC, SPI)
  - Flexibility to adapt to some customizations of these protocols
  - Translation of commands in logical protocol to application registers
- Optional Features
  - High-speed support for supporting audio data I/O in addition to control/status
  - Outbound signaling on status change

TI Developer Conference
March 7-9, 2007 • Dallas, TX

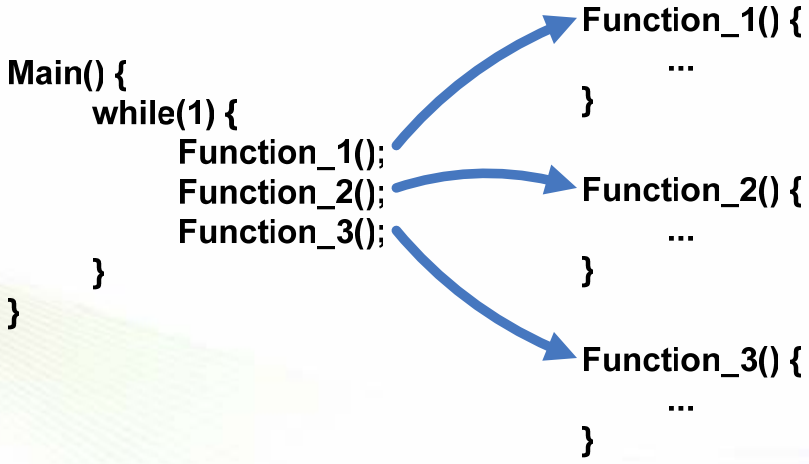
Minds in Motion

Real Time Operating Systems

Embedded DSP Systems Considerations with Audio Signal Processing

Main() Loop

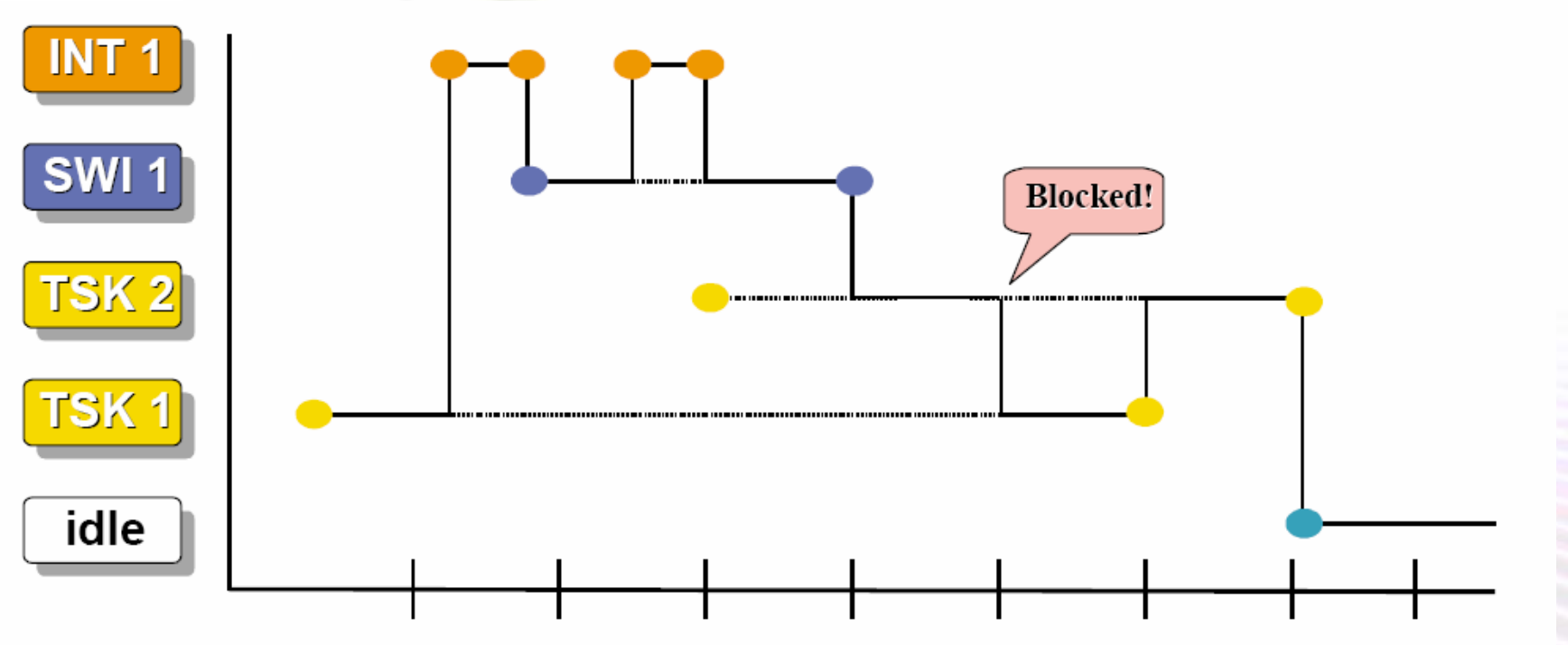
- A simple loop can very efficiently schedule a small number of functions or applications
  - Each function is called – if it is not ready to run, it exits
- Interrupts provide the ability to handle some small asynchronous events
- A main() loop falls short because it becomes difficult to hand schedule more complex or multiple independent applications



Simple RTOS

- A simple RTOS contains two fundamental services:
  - **Real-time scheduling**
    - OS can schedule an application to execute based on priority (pre-emptive multi-tasking)
  - **Synchronization**
    - Enables two independent events to synchronize (semaphore)
- A simple RTOS can enable one or more applications to execute independently where each application contains one or more tasks
  - Each task has an associated priority
  - Each application waits for available input and output data in order to be ready to run
  - The OS scheduler will execute the highest priority task that is ready to run in the system
- A simple RTOS starts to fall short because it becomes difficult to handle very complex tasks that operate for unknown duration, many asynchronous tasks and difficult scheduling

Priority Based OS Scheduling



More Advanced RTOS

- Includes features like memory protections and more complex scheduling
  - Memory protection
    - Works with hardware (MMU) memory management units
      - Enables each application to execute in its own virtual address space – if the application attempts to go outside its address space (bad pointer), OS will prevent and close the application without a crash
  - Complex scheduling
    - Offers other scheduling options
      - For example, time-slicing of tasks running at the same priority – required if these tasks execute for unknown amounts of time
- These features come at the price of performance (both MHz and memory)

TI Developer Conference
March 7-9, 2007 • Dallas, TX

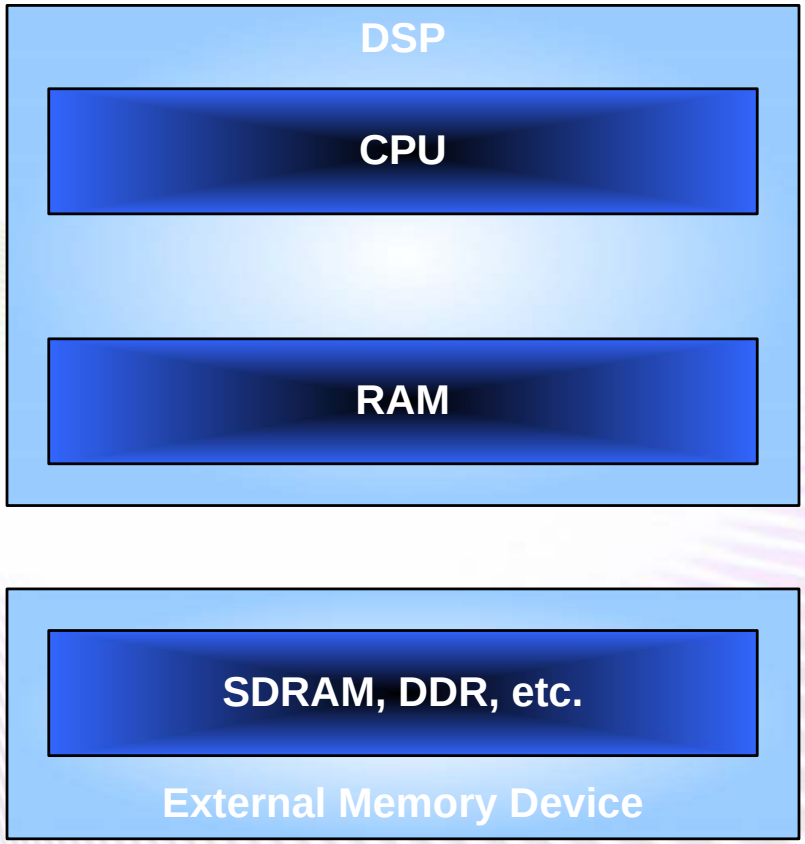
Minds in Motion

Memory Management

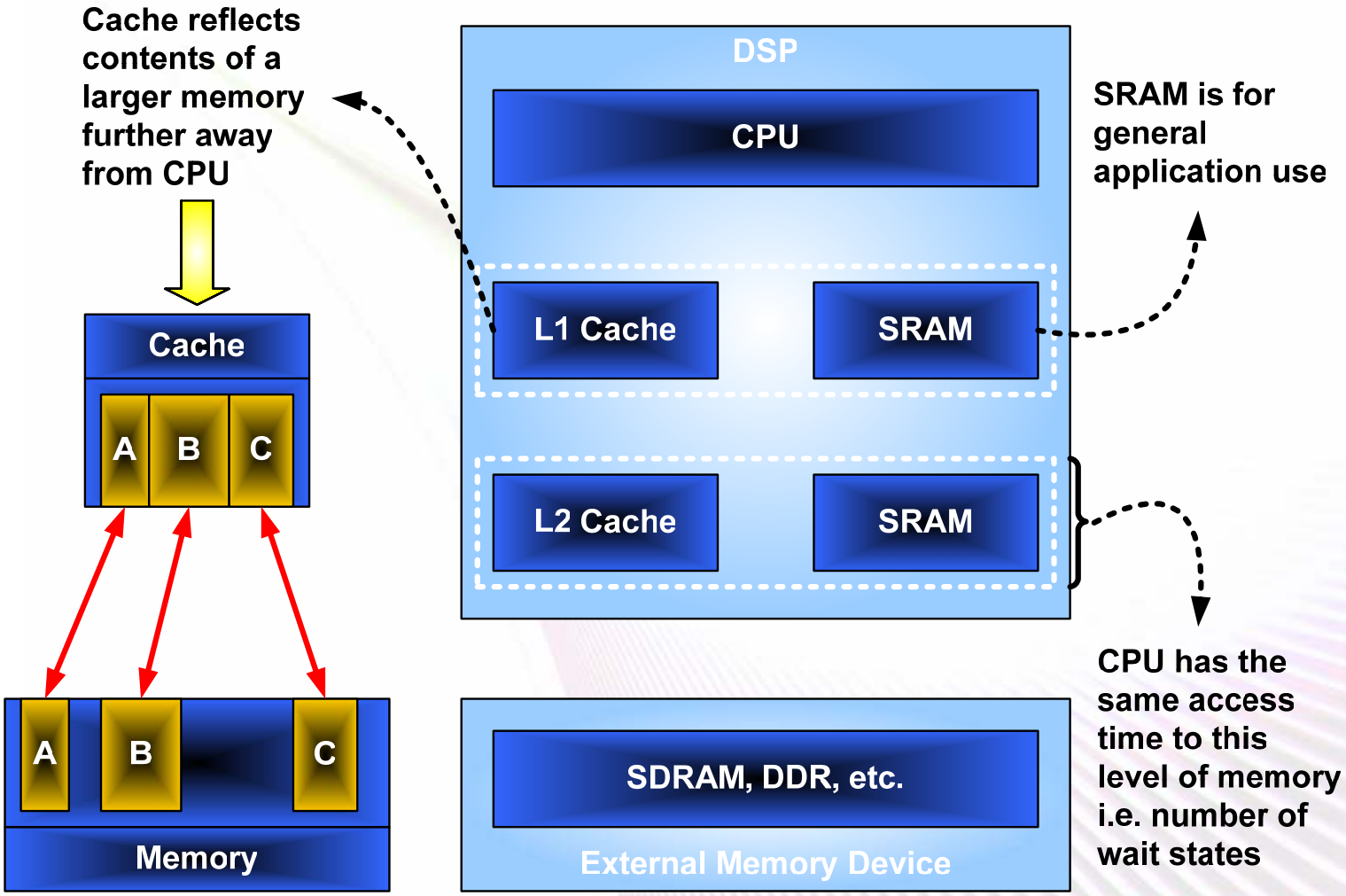
Embedded DSP Systems Considerations with Audio Signal Processing

Memory

- Many DSP applications can be thought of as requiring only a CPU and memory
- The memory configuration is likely to have a large impact on an application
- Processors will always have some amount of on-chip RAM and usually depend on external memory devices for most of the storage

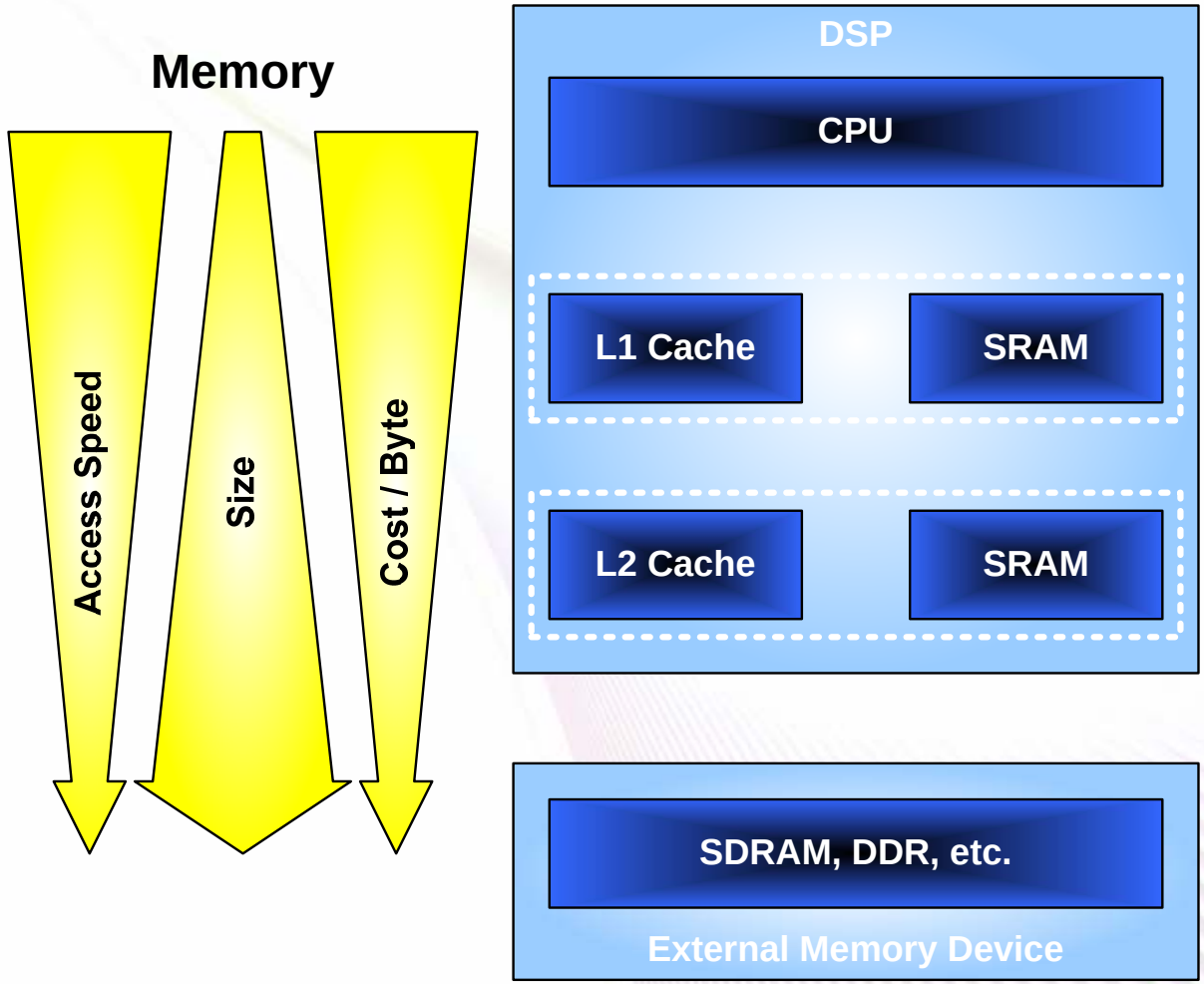


Example DSP Memories

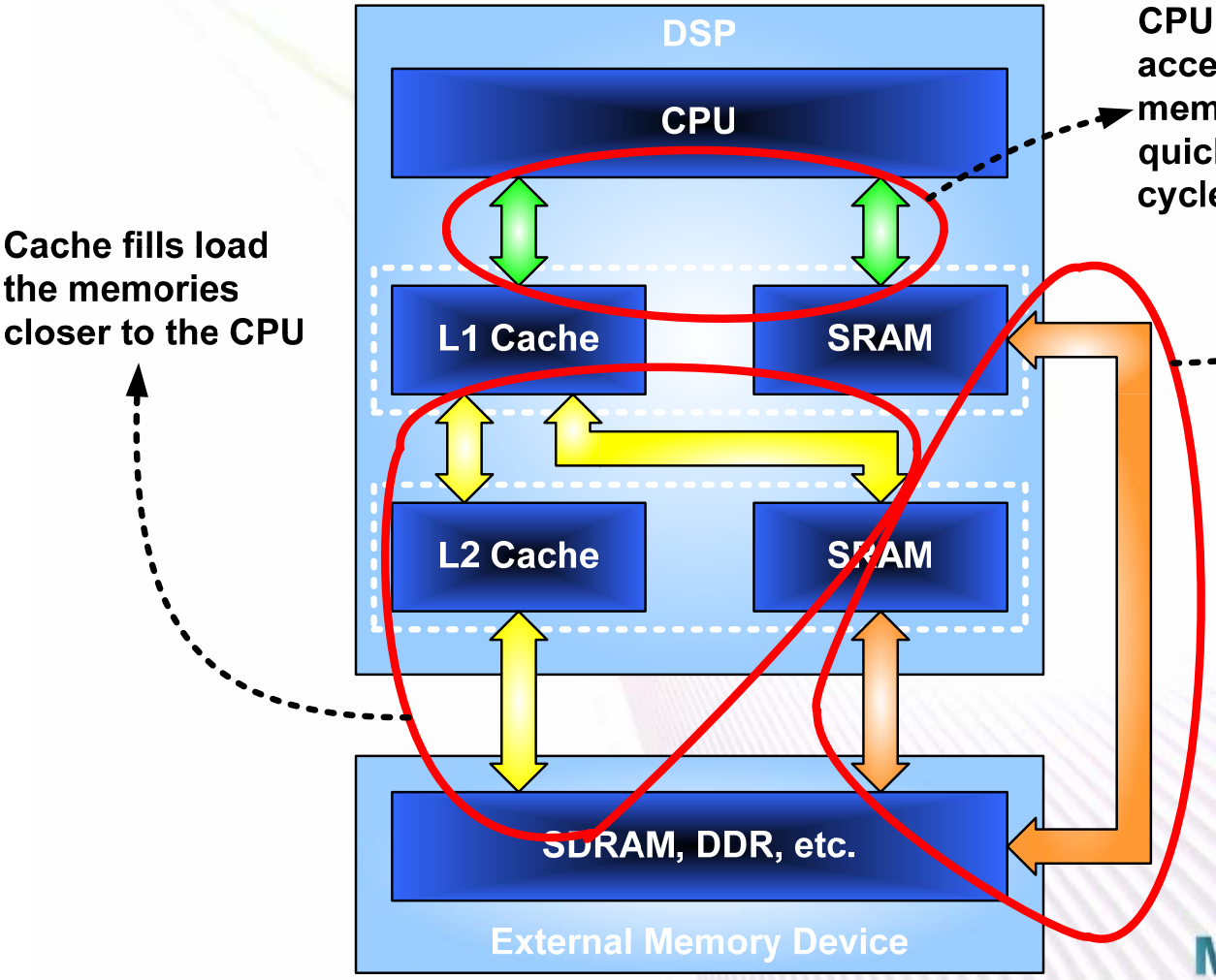


Minds in Motion

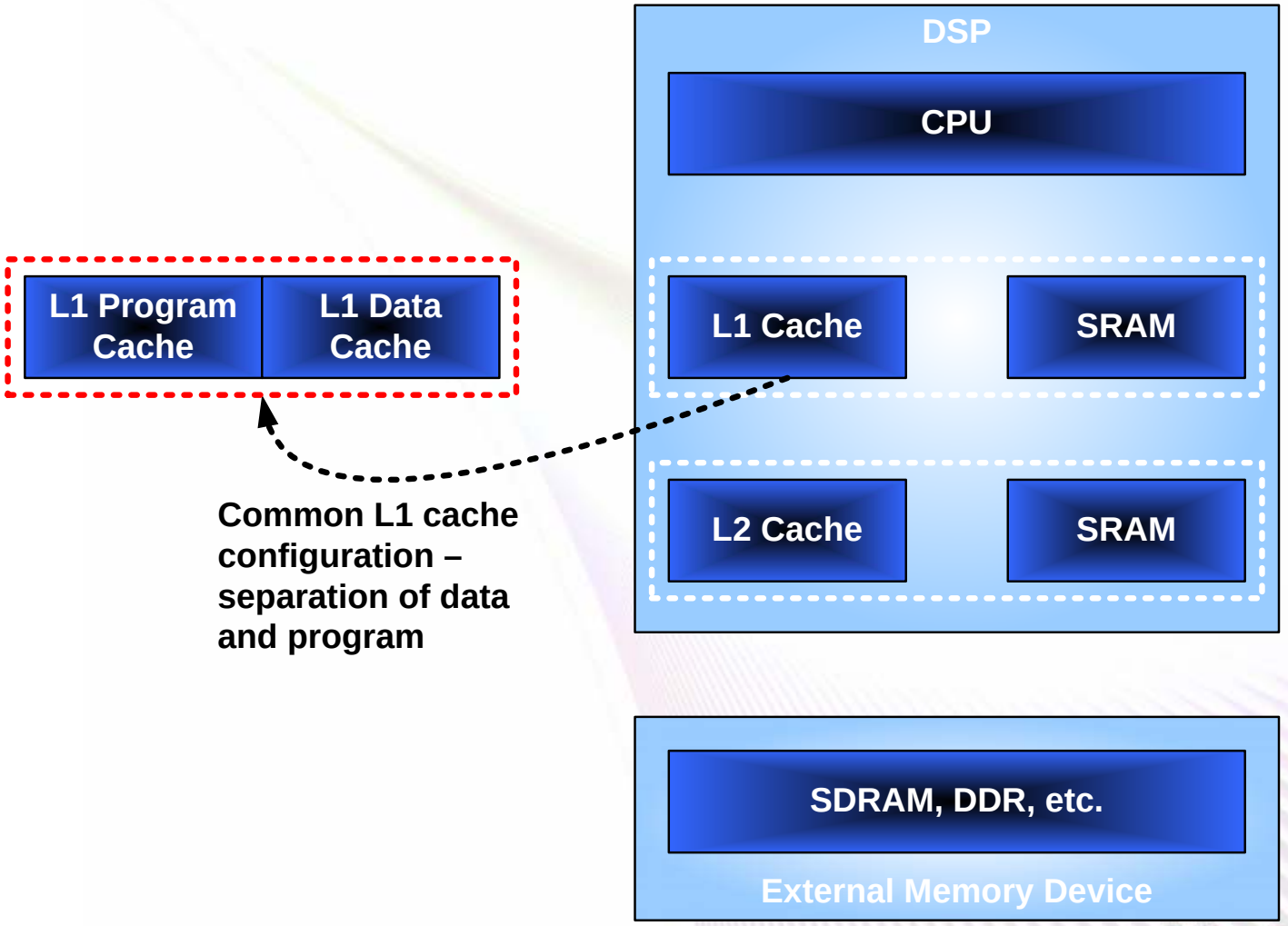
Memory Performance/Cost Tradeoff



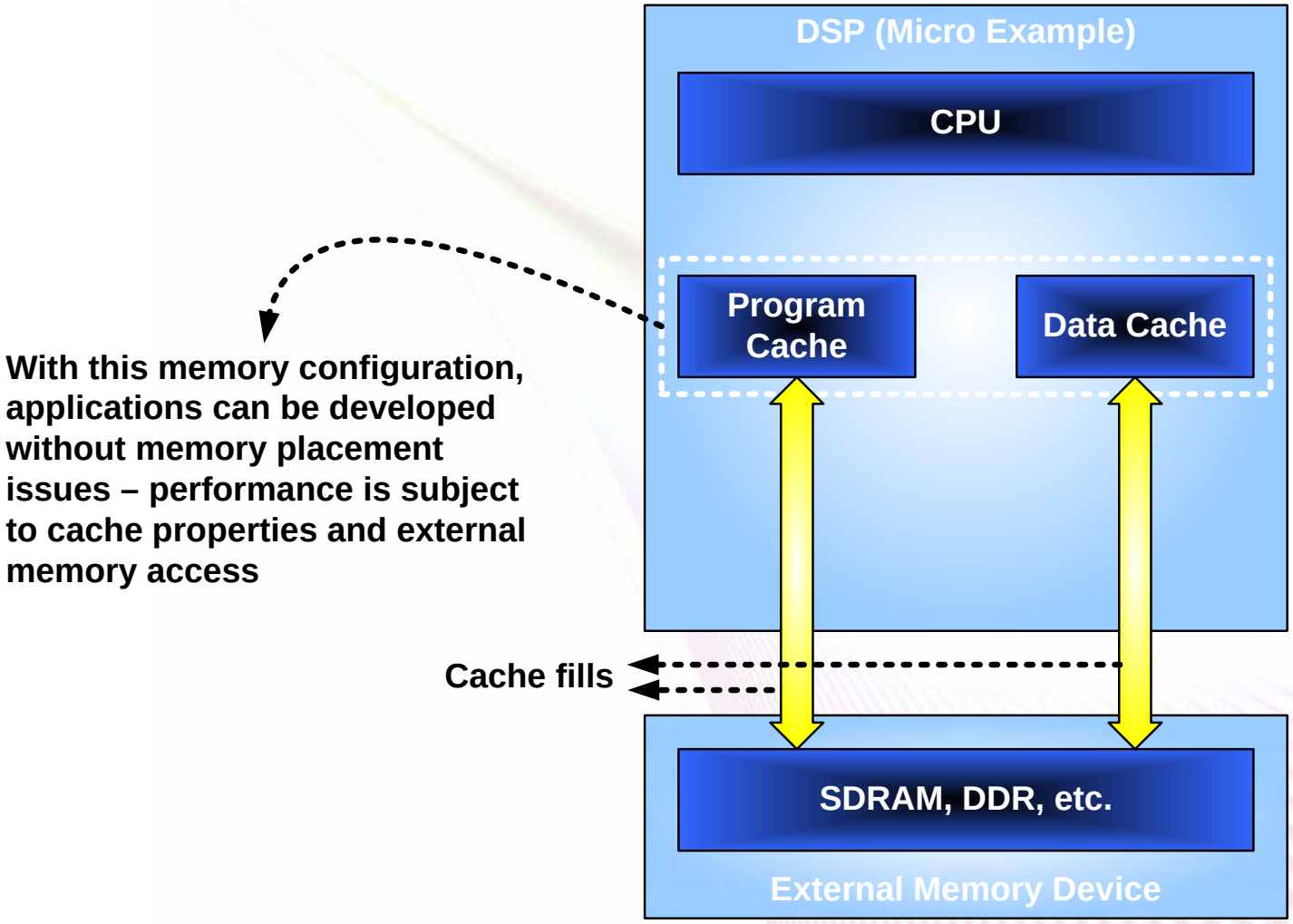
Accessing Memory



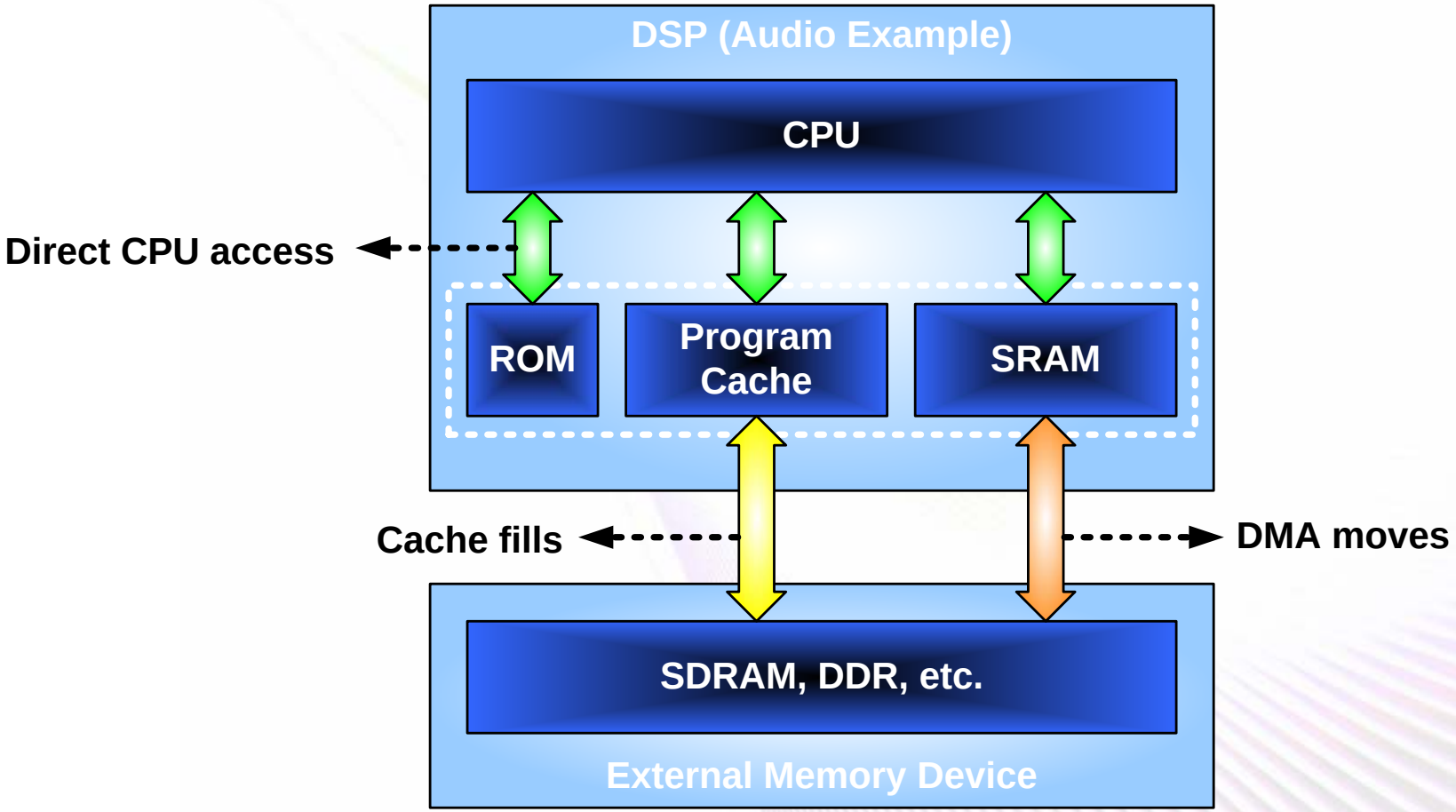
Separate Program and Data Cache



Example Micro Configuration



Possible Audio DSP Configuration

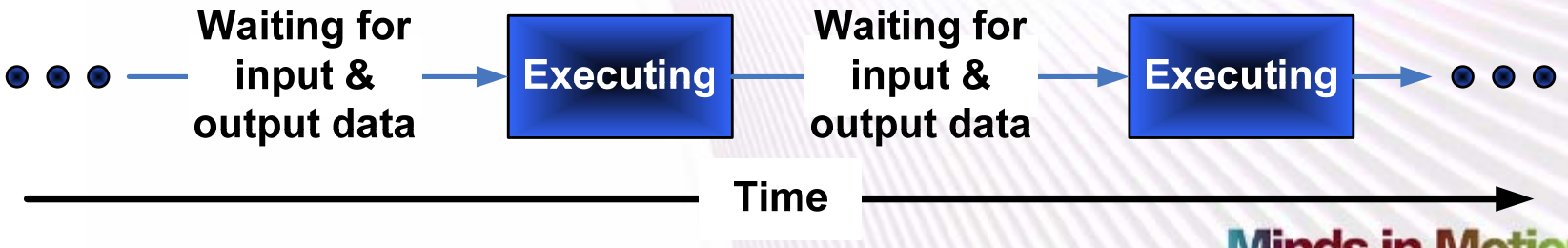


DSP Application Memory Usage

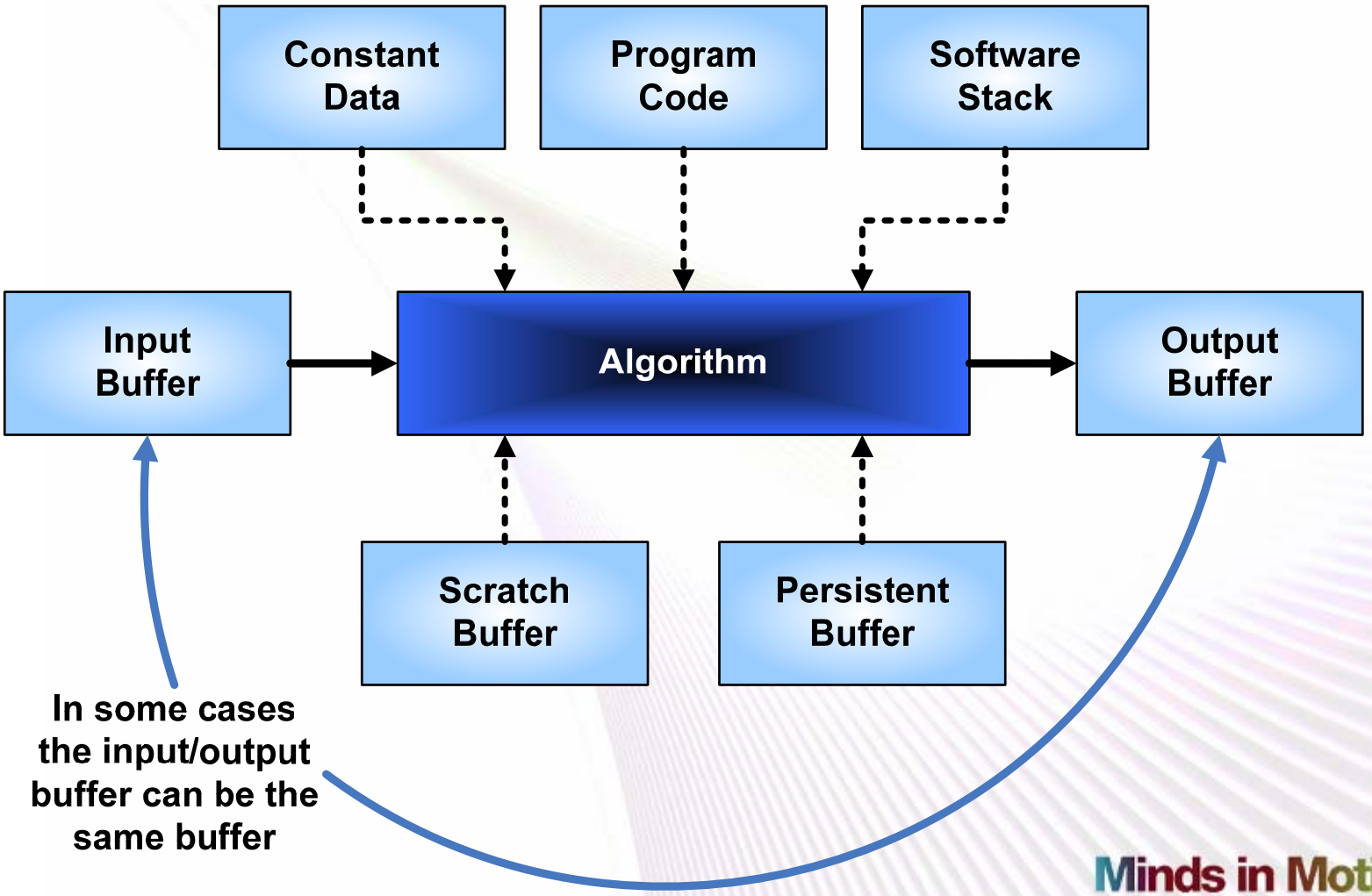
- A DSP application will achieve peak performance when all software components (program code, constants, data, etc.) are stored in the fastest available memory.
- How is this achieved?

DSP Applications

- Typical DSP applications for audio are generally algorithms that are block based
  - Examples include audio compression algorithms and PCM audio processing
- Program flow
  - Wait for input
  - Wait for output
  - Execute
  - Repeat
- Executing
  - Can be thought of as a complex function call
  - Finite amount of processing time is required (fairly predictable)



General Algorithm Memory Requirements

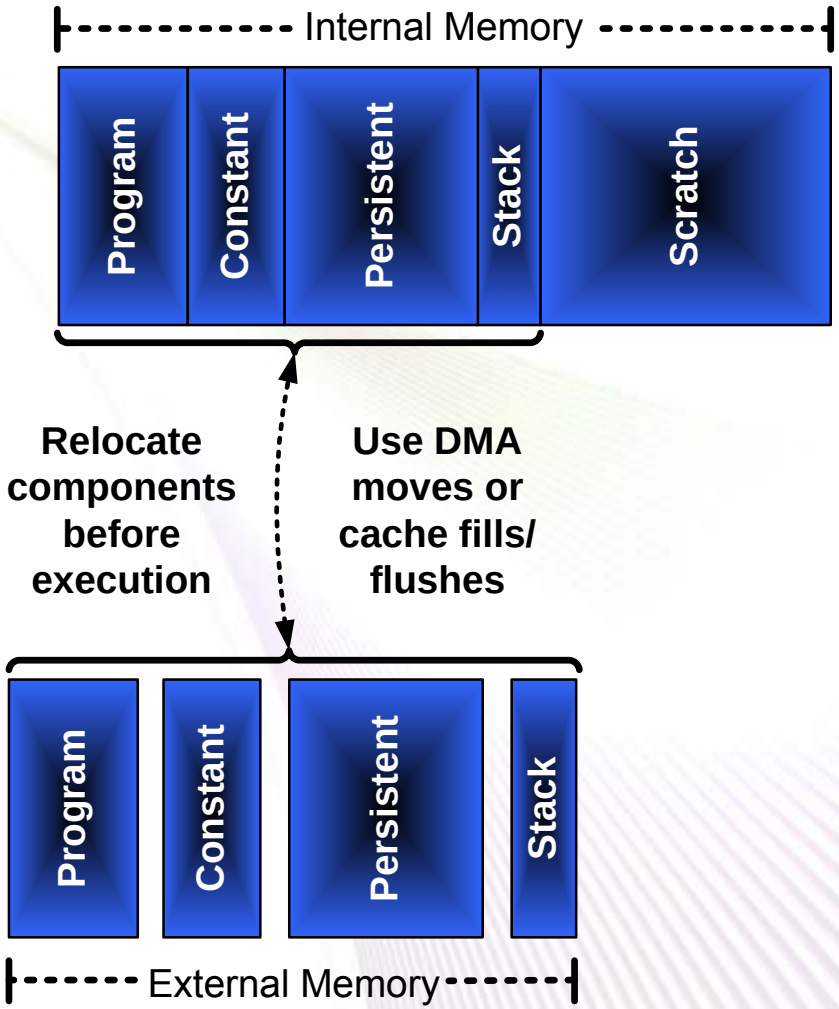


General Algorithm Memory Usage

- DSP applications consist of the following sub-components:
  - *Program code*
    - Program instruction code
  - *Software stack*
    - Memory used for function calls, local variables, etc.
  - *Constant data*
    - Constant tables, etc.
  - *Scratch data*
    - Work buffers used during program execution
  - *Persistent data*
    - State data that must be preserved between application executions

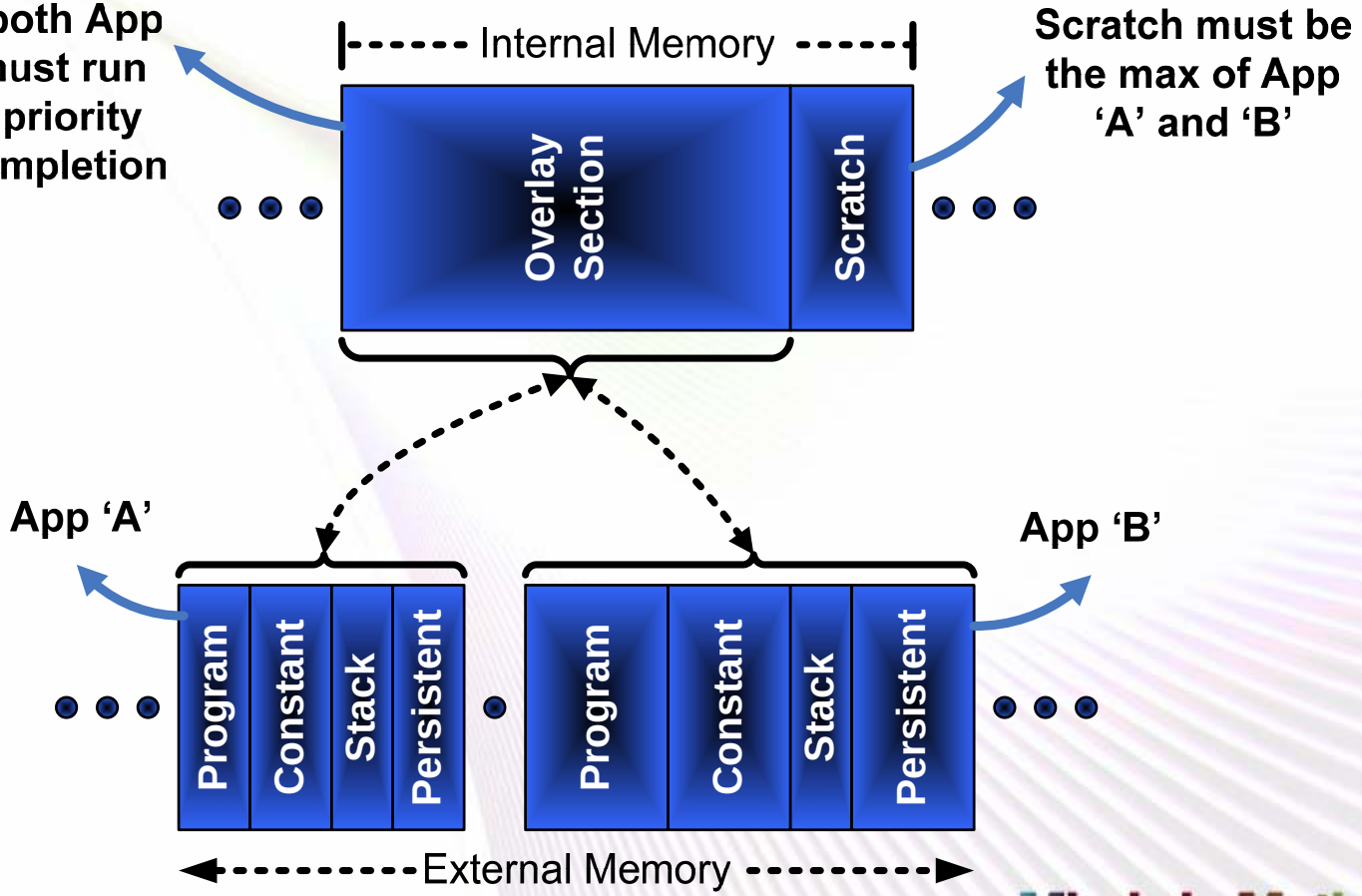
	Execution	Waiting
Program Code	Max	None
Software Stack	Max	Min
Constant Data	Max	None
Scratch Data	Max	None
Persistent Data	Max	Max

Sub-Component Relocation



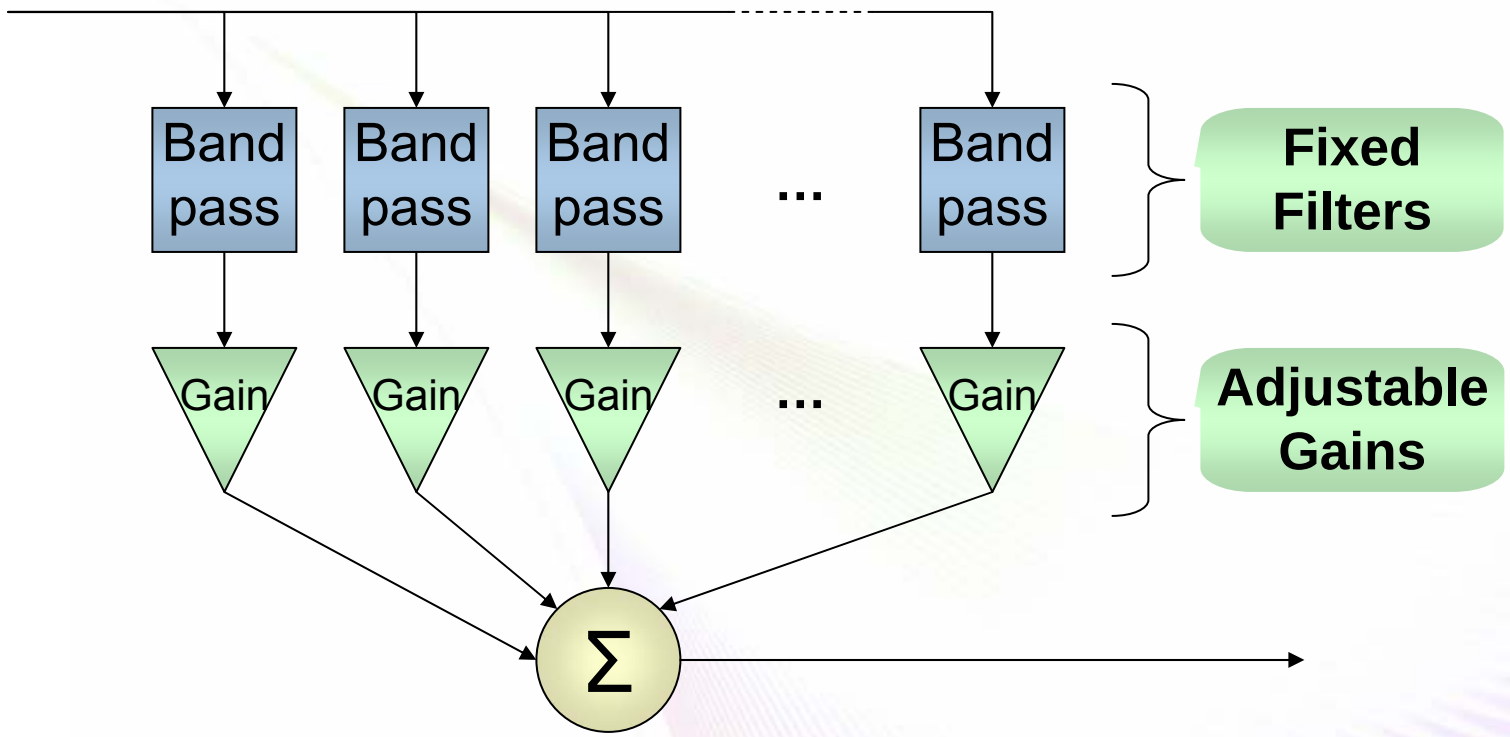
Sharing Internal Memory

To enable sharing of the internal memory resources – both App 'A' and 'B' must run at the same priority and run to completion



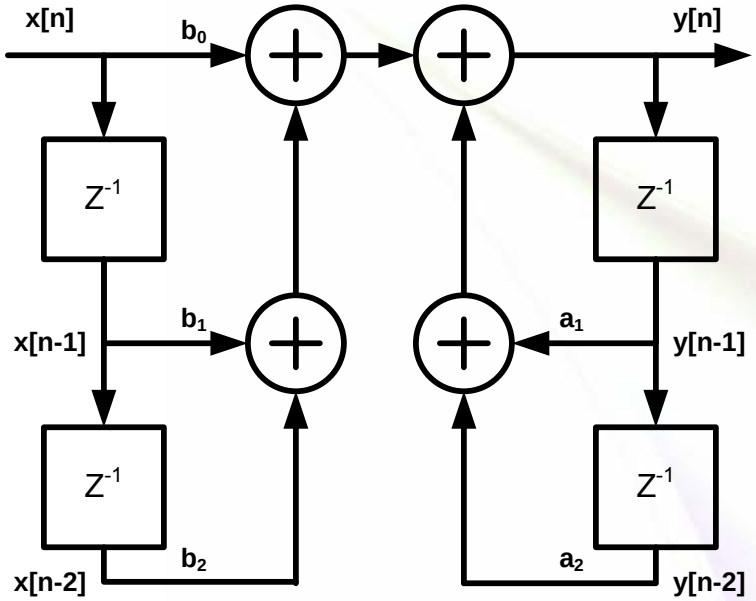
Minds in Motion

PCM Processing Example - Graphic EQ



- Graphic EQ
  - Parallel bank of Bandpass filters (outputs summed)
  - Fixed Q & Center frequency
  - Typically based on Direct Form I filter (biquad)

Graphic EQ Memory Requirements



Direct Form I

States	4 x 4 bytes = 16 bytes
Coefficients	5 x 4 bytes = 20 bytes

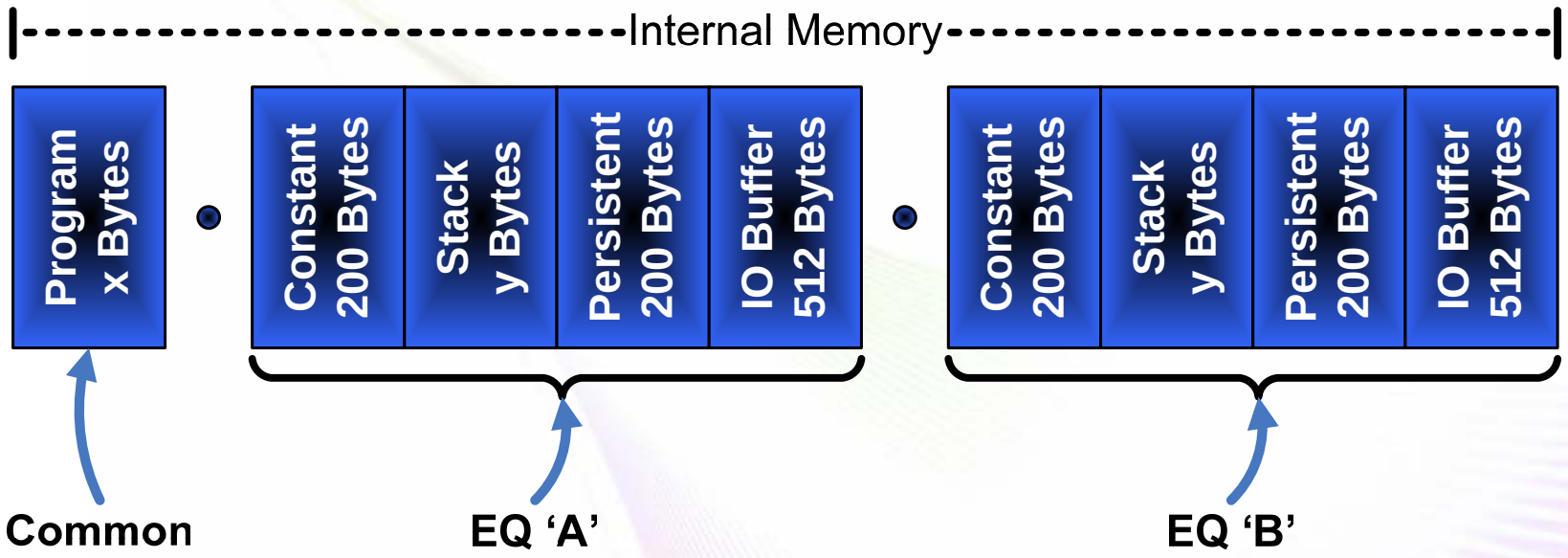
*Assume all data requires 32-bits

10-band EQ	Bytes	Comment
Program Code	x	ISA specific
Software Stack	x	ISA specific (per instance)
Constant Data	200	Coefficients
Scratch Data	0	None
Persistent Data	200	States + gains (per channel)

(Small memory requirements)

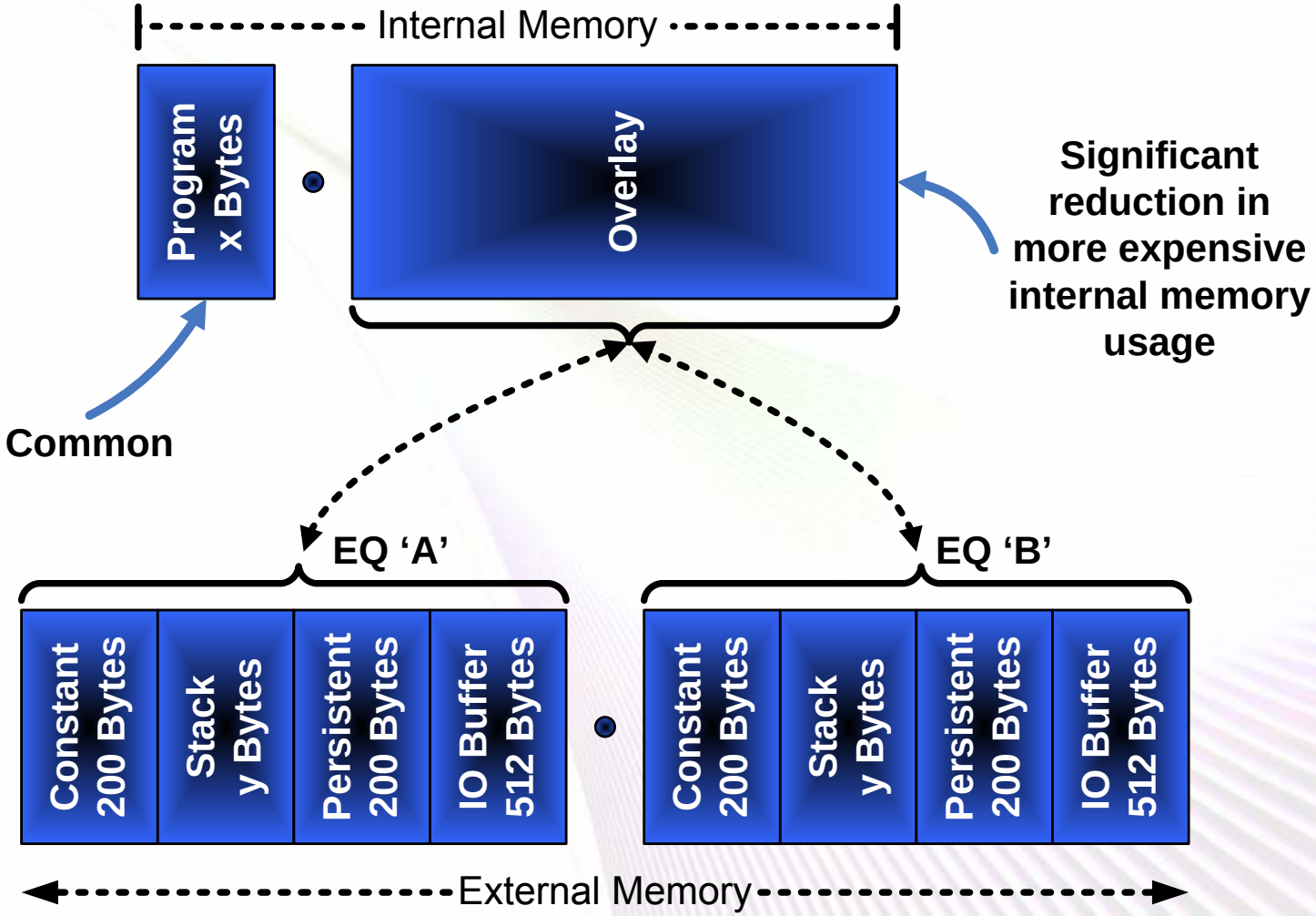
Minds in Motion

Two Concurrent EQs – No Resource Sharing



*Assume 128 PCM samples are processed in-place each 32-bits

Two Concurrent EQs – Shared Resources



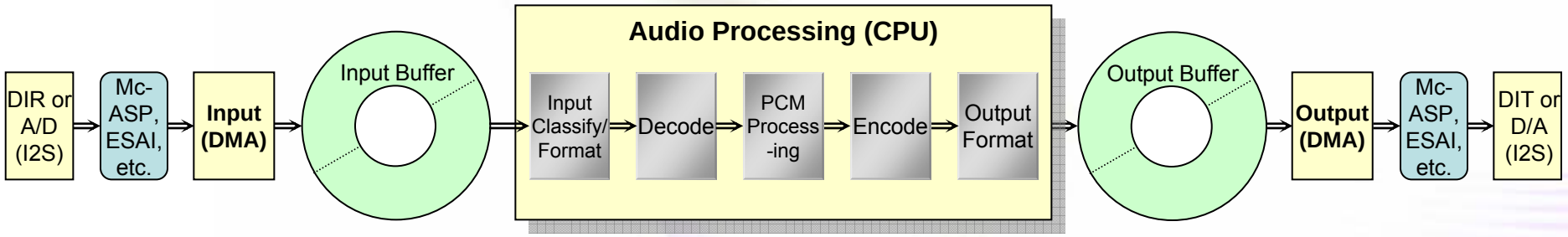
Maximizing Internal Memory Usage

- Use of internal memory can be maximized by applying the following concepts:
  1. If a DSP application is run to completion, all the internal memory consumed during execution can be reused by other applications while the current application waits to execute again.
  2. If a DSP application is interrupted during execution, the interrupting application cannot reuse the same internal memory.
  3. All software sub-components can be stored in external memory between application executions.

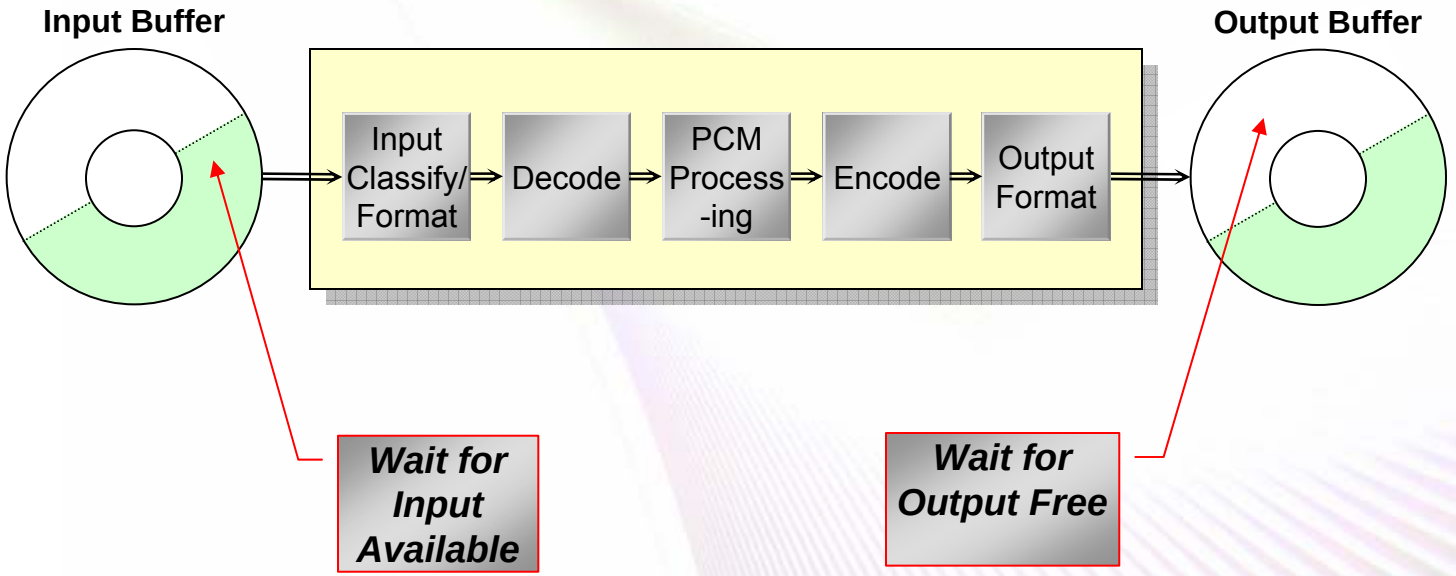
Memory Allocation

- General thoughts on memory allocation in real-time systems
  - Static memory placement is fine
  - Malloc is even okay
  - Freeing is what gets you into trouble
- Freeing memory can cause fragmentation
  - Garbage collection in a real-time system is difficult to do

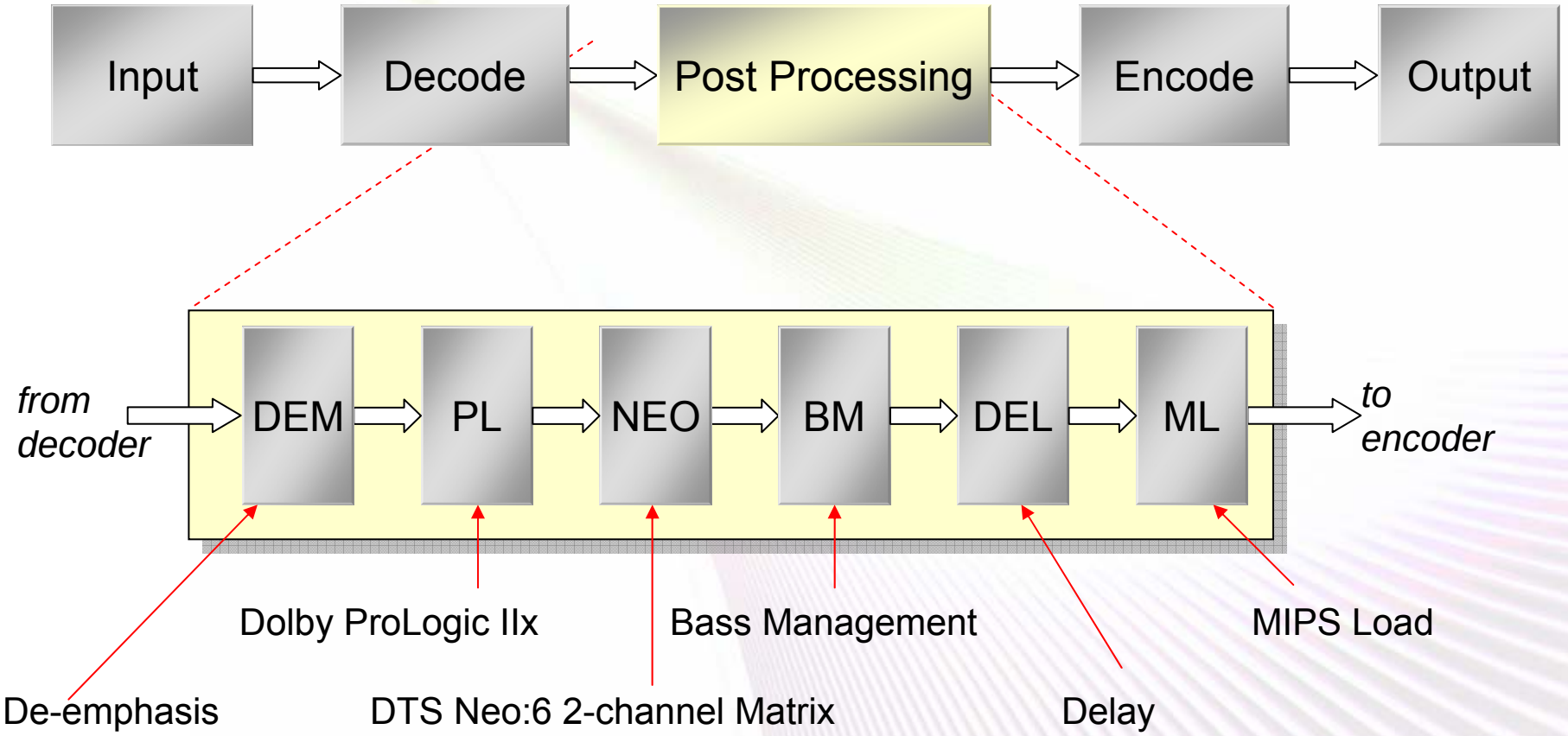
Typical Audio DSP Data Flow



Audio Processing "Main Loop" (CPU)



PCM Processing



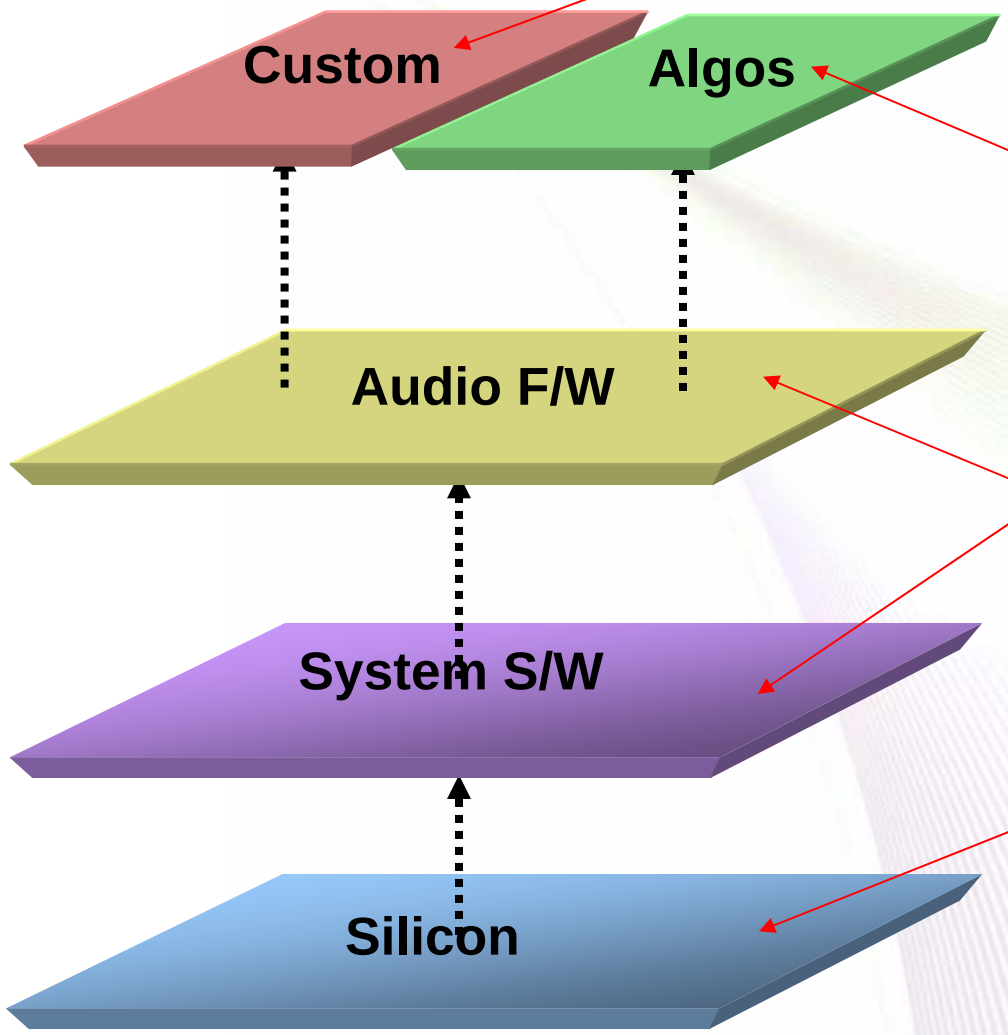
Audio Frameworks

- Provide Infrastructure for Audio Processing and I/O
- Customize for *Product Differentiation*
  - Add/Modify/Delete Post-Processing Algorithms
  - Change Component Parameters
- Standard Interfaces Allow *Component* Changes with little or no *Framework* Changes
  - More Efficient Development
  - Better Market Responsiveness
  - Improved Software Quality

Parameter-Driven Audio Frameworks

- Parameters
  - Feature Sets
    - Decoders
    - Post-Processing Modules
  - Processing Topologies
  - I/O Protocols, Formats, Channels
    - Audio
    - Control
  - Memory usage
    - I/O Buffers, Heap, Overlaps
- Run directly from ROM
  - Patch to add/fix

Performance Audio Framework (PA/F)



CUSTOMIZATION

F/W, System S/W, and standard algorithms allow customers to focus on value added features

AUDIO ALGORITHMS

Comprehensive list of optimized audio decode, encode, post processing and I/O algorithm implementations

FRAMEWORK + SYS S/W

Enables high-quality, complete audio system while delivering flexible design platform

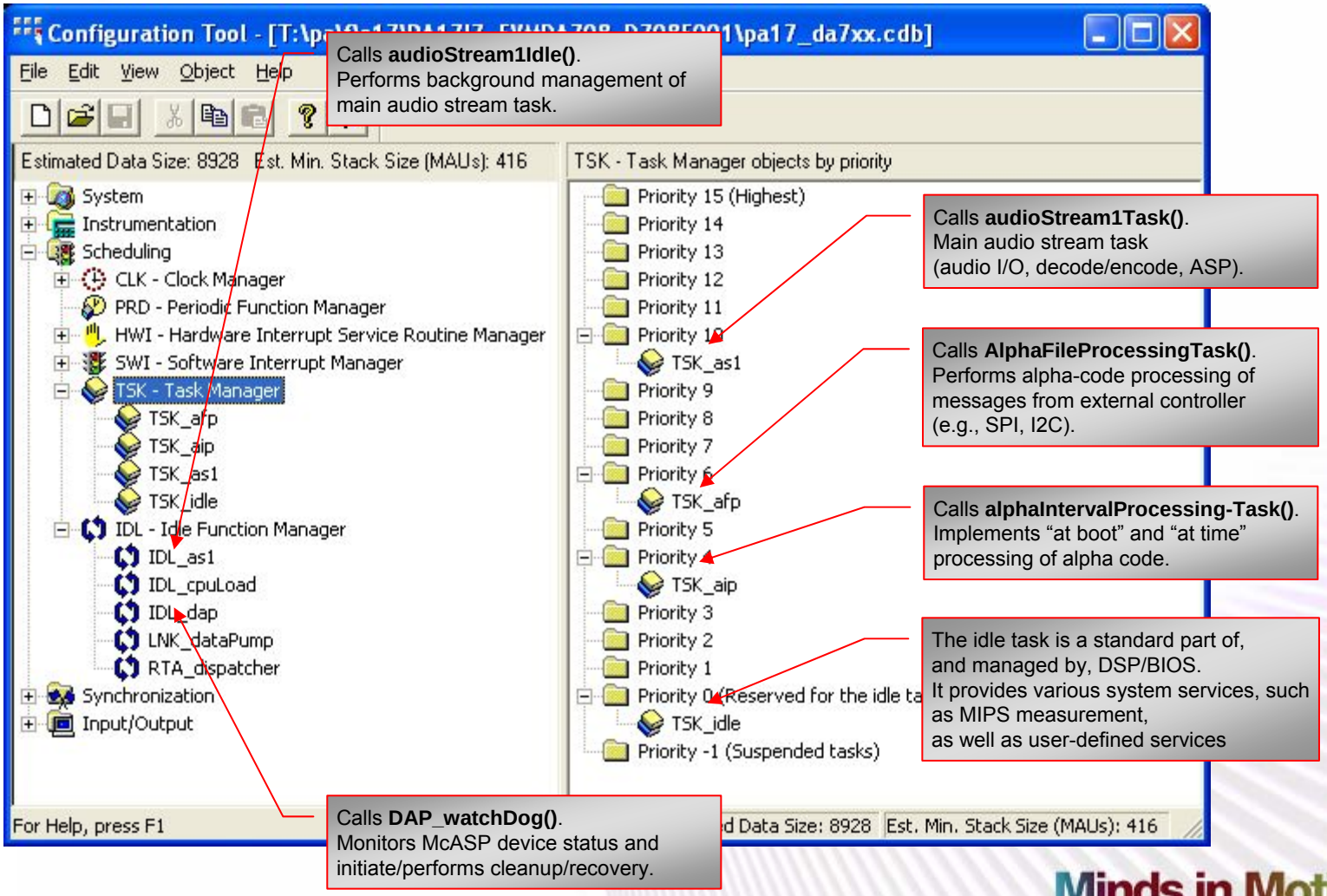
AUREUS™ AUDIO DSP

Scalable Family of Audio DSPs maximizes design reuse and reduces development costs

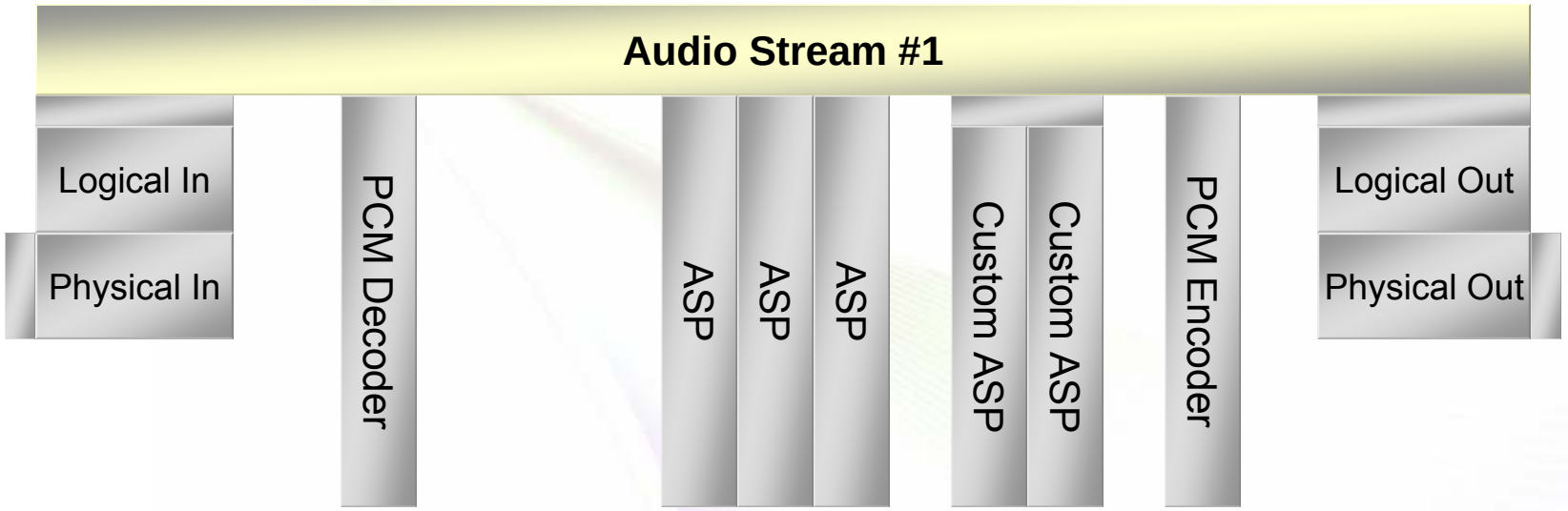
PA/F Foundation — DSP/BIOS

- Processing is organized into Tasks (TSK)
  - Task priorities facilitate real-time artifact-free audio
- I/O is performed using SIO drivers
  - Stacking SIO drivers used for additional “logical” processing
- Memory management
  - IRAM & SDRAM heaps (creation only — no fragmentation)
  - One stack per task and one system stack (quasi-dynamic memory)
  - Global data
    - DSP/BIOS configuration
    - Static function tables

PA/F Task & Idle Function Organization



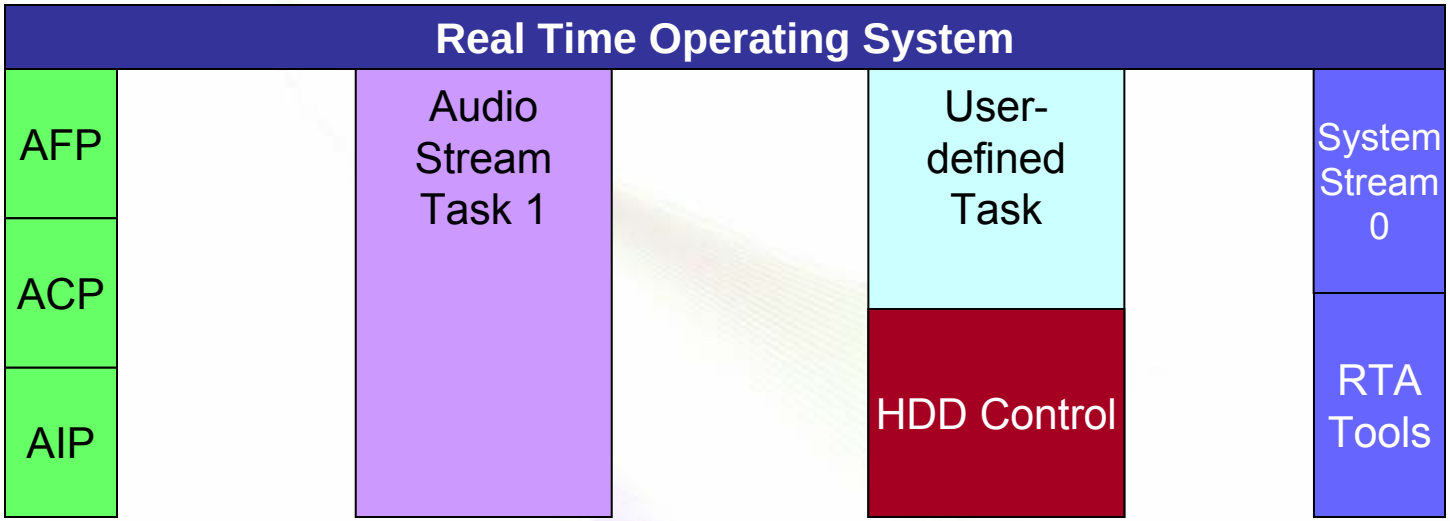
Audio Stream Task



What is the Audio Stream Task?

- Initialization of components
- Control and definition of audio streams

Multiple Tasks



TMS320 DSP Algorithm Standard (xDAIS)

- TI's standard guideline for DSP S/W development
- Standardized way of coding algorithms
- Allows plug-and-play portability/flexibility
- Object-oriented style framework
- Re-entrant, multiple-instance
- CCS supports easy xDAIS coding
- All PA/F components are built to xDAIS standard

PA/F Algorithms — Interface

- Based on xDAIS' ALG & (newer) ALGRF
- Application-specific public functions added
  - E.g., for ASPs: reset(), apply()
- PAF_ALG enhancements
  - Enable safe "common" memory sharing between mutually exclusive algos.

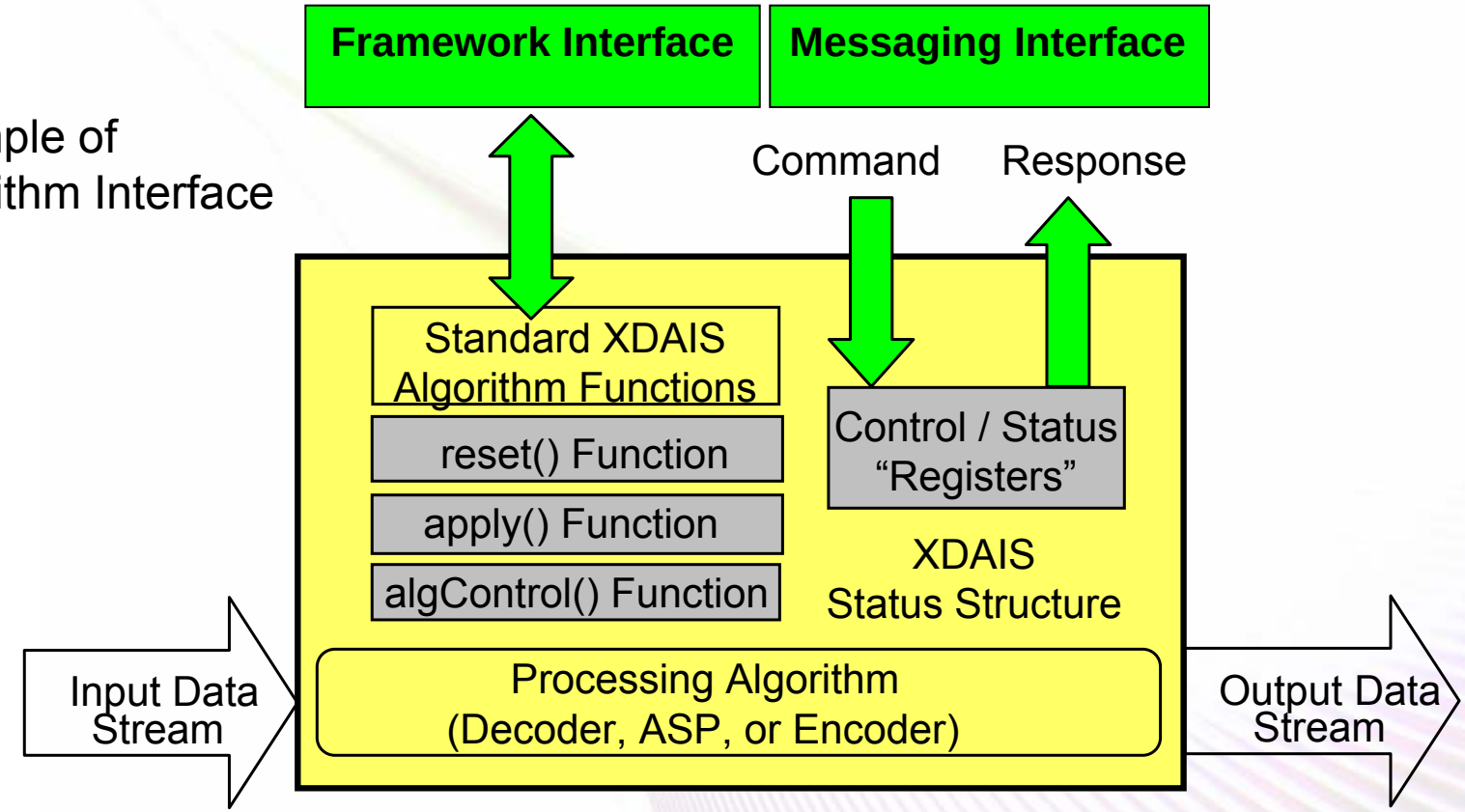
PAF_ALG Common Memory Types

- From `paf_ialg.h`

```
PAF_IALG_COMMON_STEREO_DECODE_PROCESSING
PAF_IALG_COMMON_MULTI_DECODE_PROCESSING
PAF_IALG_COMMON_FRONT_SURROUND_PROCESSING
PAF_IALG_COMMON_RESERVED0
PAF_IALG_COMMON_BACK_SURROUND_PROCESSING
PAF_IALG_COMMON_ENCODE_PROCESSING
PAF_IALG_COMMON_VIRTUAL_PROCESSING
PAF_IALG_COMMON_BASS_MANAGEMENT
PAF_IALG_COMMON_DELAY_MANAGEMENT
PAF_IALG_COMMON_RESERVED1
PAF_IALG_COMMON_RESERVED2
PAF_IALG_COMMON_RESERVED3
PAF_IALG_COMMON_CUSTOM1
PAF_IALG_COMMON_CUSTOM2
PAF_IALG_COMMON_CUSTOM3
PAF_IALG_COMMON_CUSTOM4
PAF_IALG_COMMON_MEM0 IALG_SCRATCH
PAF_IALG_COMMON_MEM1
```

PAF_ALG Interface

Example of Algorithm Interface



- XDAIS compliant C code using above template
- Serial commands communicate with rest of system and algorithm
- Serial commands sent either at initialization time, regular intervals, from external host, or from user-defined task

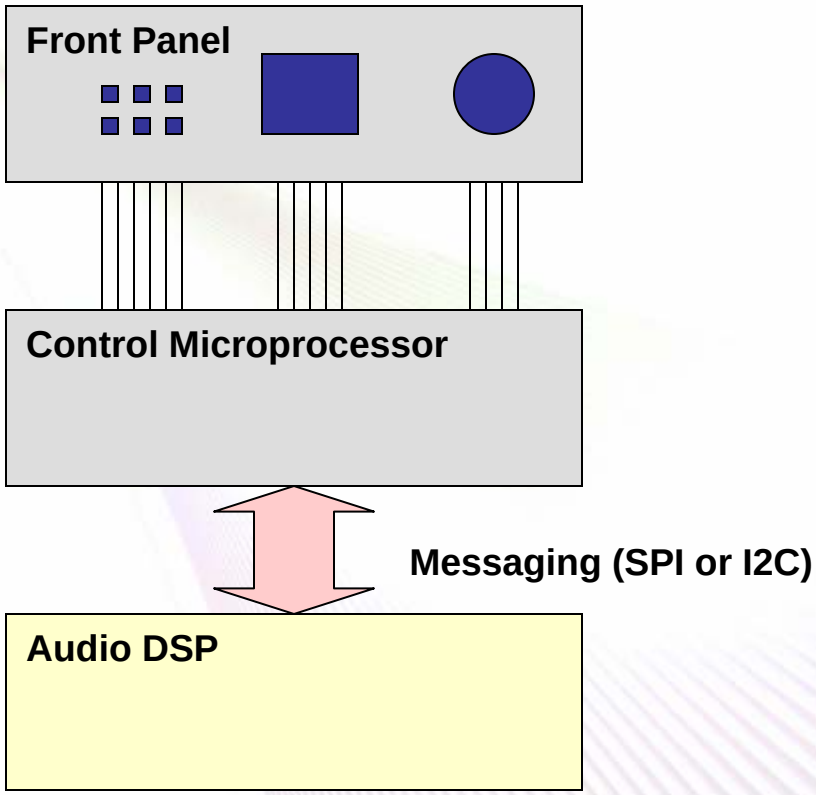
Achieving High (MIPS) Efficiency

- Re-use efficient code "building blocks"
  - Especially Filter algo. and Filter functions
  - Potentially minimizes ROM, RAM, MIPS
- Keep the (8) ALU functional units busy
  - Combine operations in cascade or parallel (e.g. multi-channel)
- Minimize number of read/write "passes" through data to reduce read/write instructions
  - Don't divide processing into too many post processing functions
- Locate frequently used code in IRAM
- Typically use in-place processing for memory saving
- Block Processing ➔Natural fit with …
  - Compressed-bitstream audio (input frames, output blocks)
  - DSP Architectures
    - Register setup cycles can be amortized over block of audio samples in efficient loop kernels
    - Enable use of DMA for buffered I/O transfers

Control Architecture

- Allows run-time monitoring and control (“user interface”)
- Direct communication with individual system components
- Use DSP to organize and assimilate information to be reported
- Regularize multi-task, DSP/micro, DSP/DSP communications
- Consider synchronization requirements

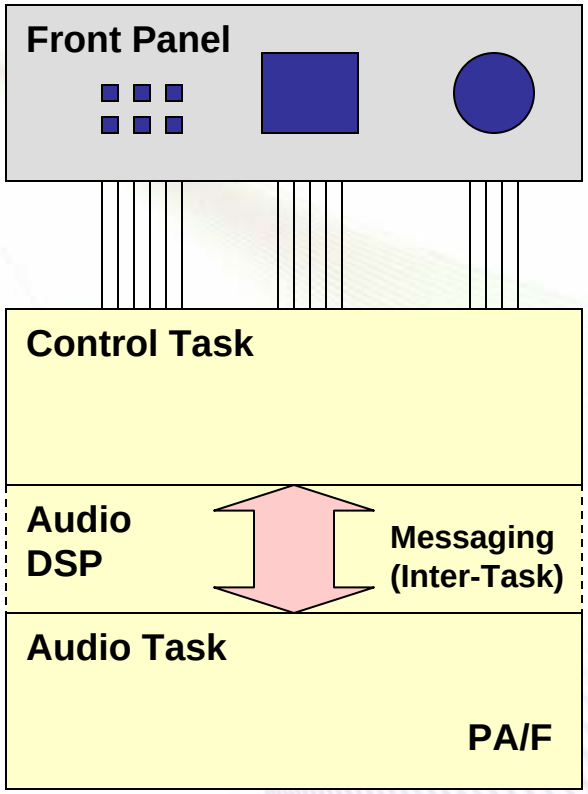
Application — Conventional A/V Receiver



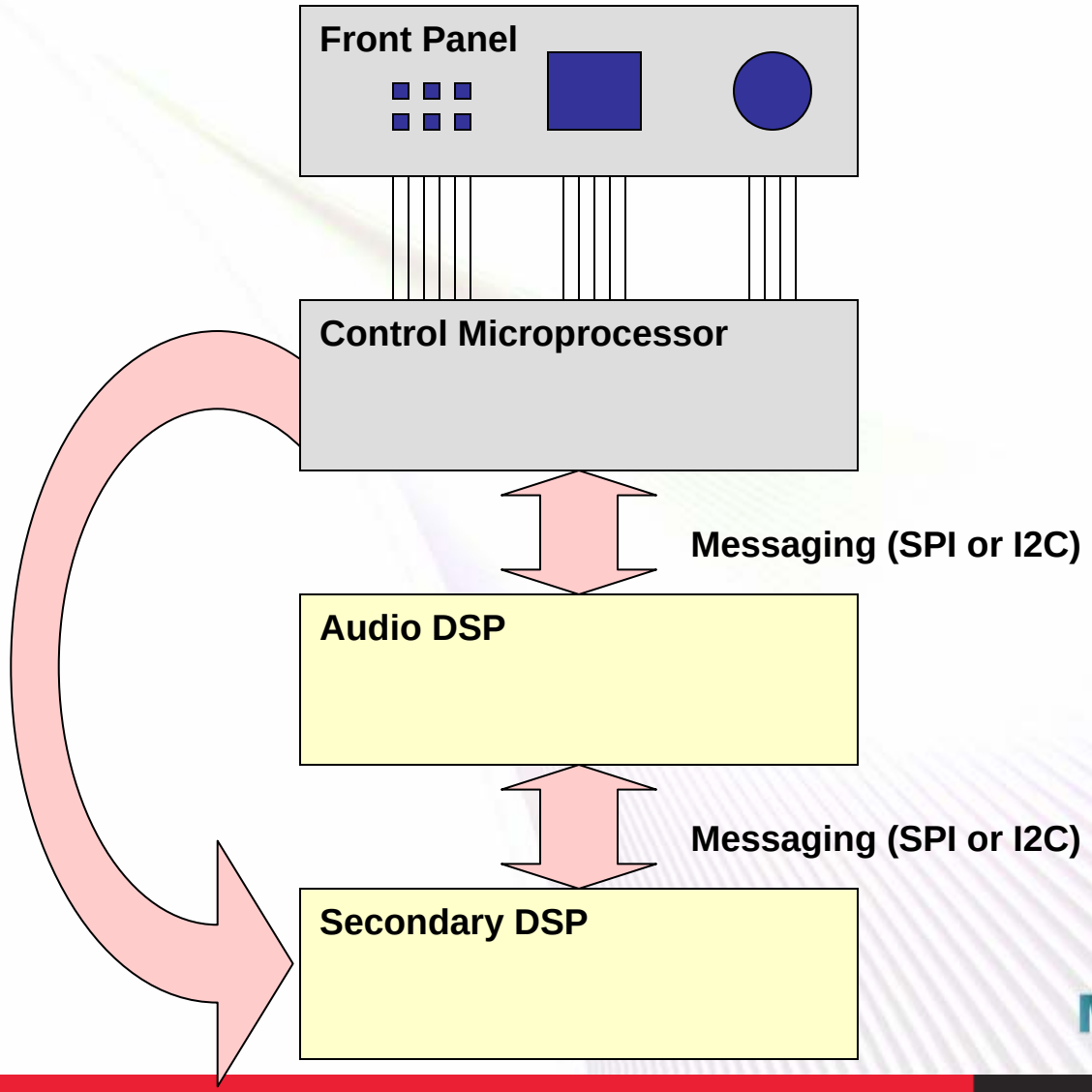
System Information

- Sample Rates
- Listening Modes
- Channel Configurations
- Input, Decode, Output Processing
- Volume Control
- Input/Output Switching (IOS)

Application — Cost-Reduced A/V Receiver



Application — High-End A/V Receiver



PA Messaging (PA/M)

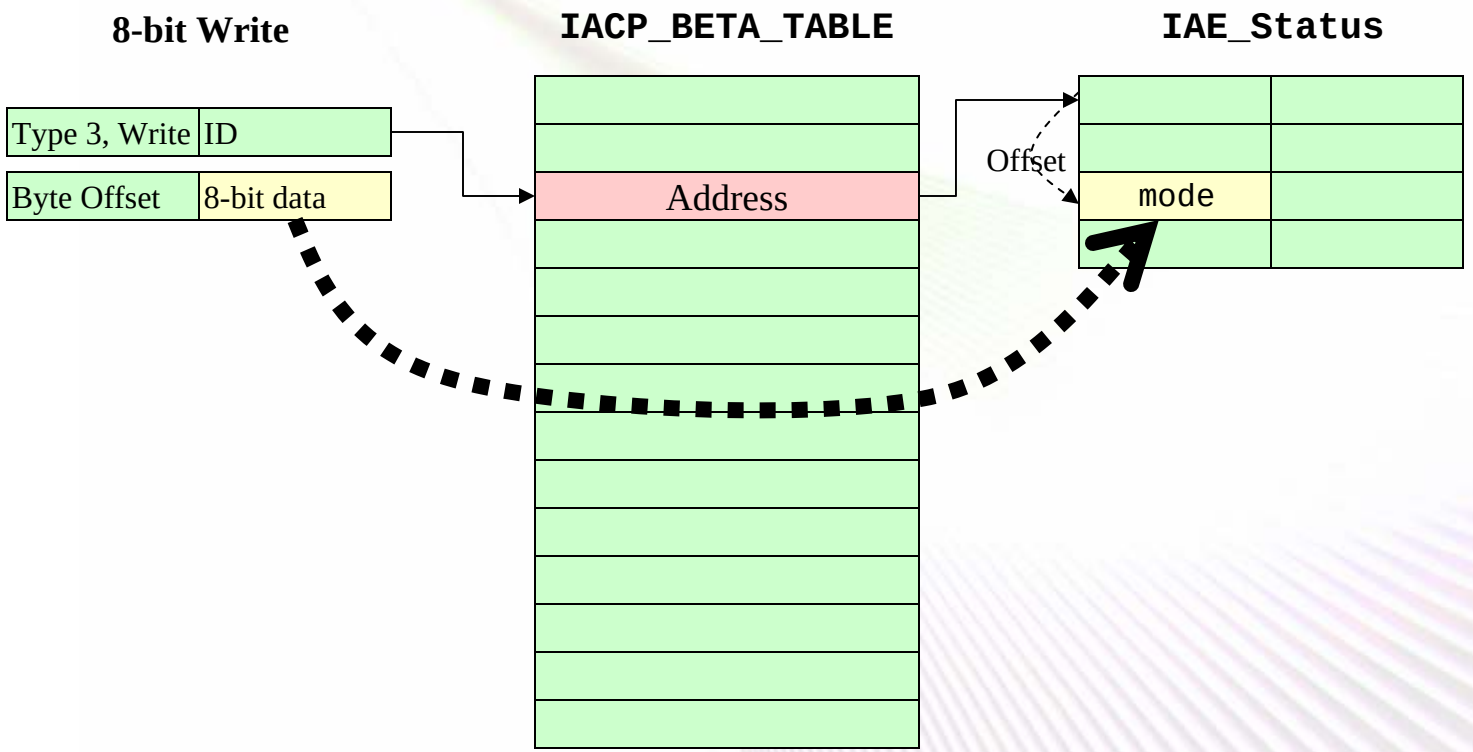
- Multiple message types provide
  - Application flexibility
  - Communications bandwidth efficiency (8/16/32/vector data)
- Most messages perform memory Read or Write
- Remote function calls
- Presets facilitated by customer-defined “shortcuts”
- OEM Extensions implicitly supported
- Future messaging extensions possible
- Host/DSP, Intra-, Inter-DSP communications
- Symbolic control of algorithm parameters/status

PA/M — Write Messages

Length (16-bit wds)	Message Type								
	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB	
2	**Function Invocation**								
	Type 0, Write	Function num.	Calling type	8-bit data					
1+Length	**Encapsulation**								
	Type 1, Write	Len. (Words)	Word 1		[Word 2]		[Word 3]		
2	**8-bit Write**								
	Type 2, Write	Beta ID	Byte Offset	8-bit data					
3	**16-bit Write**								
	Type 3, Write	Beta ID	Byte Offset		16-bit data				
4	**32-bit Write**								
	Type 4, Write	Beta ID	Byte Offset		32-bit data				
>= 1	**Extended Write**								
	Type 5, Write	Ext. Sub-type	[Word 1]		[Word 2]		[Word 3]		
2+Length/2	**Var.-Length Write**								
	Type 6, Write	Beta ID	Byte Offset	Len. (Bytes)	Byte 1	[Byte 2]	[Byte 3]	[Byte 4]	
4+Length/2	**Physical Write**								
	Type 7, Write	Access Type	Length (Bytes)		Physical Address				Data …

PA/M — Component Addressing

- Each addressable component has a unique "ID" (0-255)



Device Interfaces

- Strive for orthogonality between I/O and Processing
  - Allow for changes to one without affecting the other
  - Layered (Stacking) I/O Drivers
    - Physical
    - Logical
- Physical I/O Driver Parameters
  - Audio I/O
    - I2S, TDM
    - Clock direction
    - Channel interleaving
    - Map physical pins to logical channels
  - Control/Status I/O
    - SPI, I2C
    - Clock settings
- Logical Parameters
  - What content types are recognized? (Auto-detection)
  - IEC header analysis
  - Restart conditions based on I/O

Multiple Audio Zones

Embedded DSP Systems Considerations with Audio Signal Processing

Topics

- Why is a multi-zone architecture relevant?
- How to realize multi-zone?

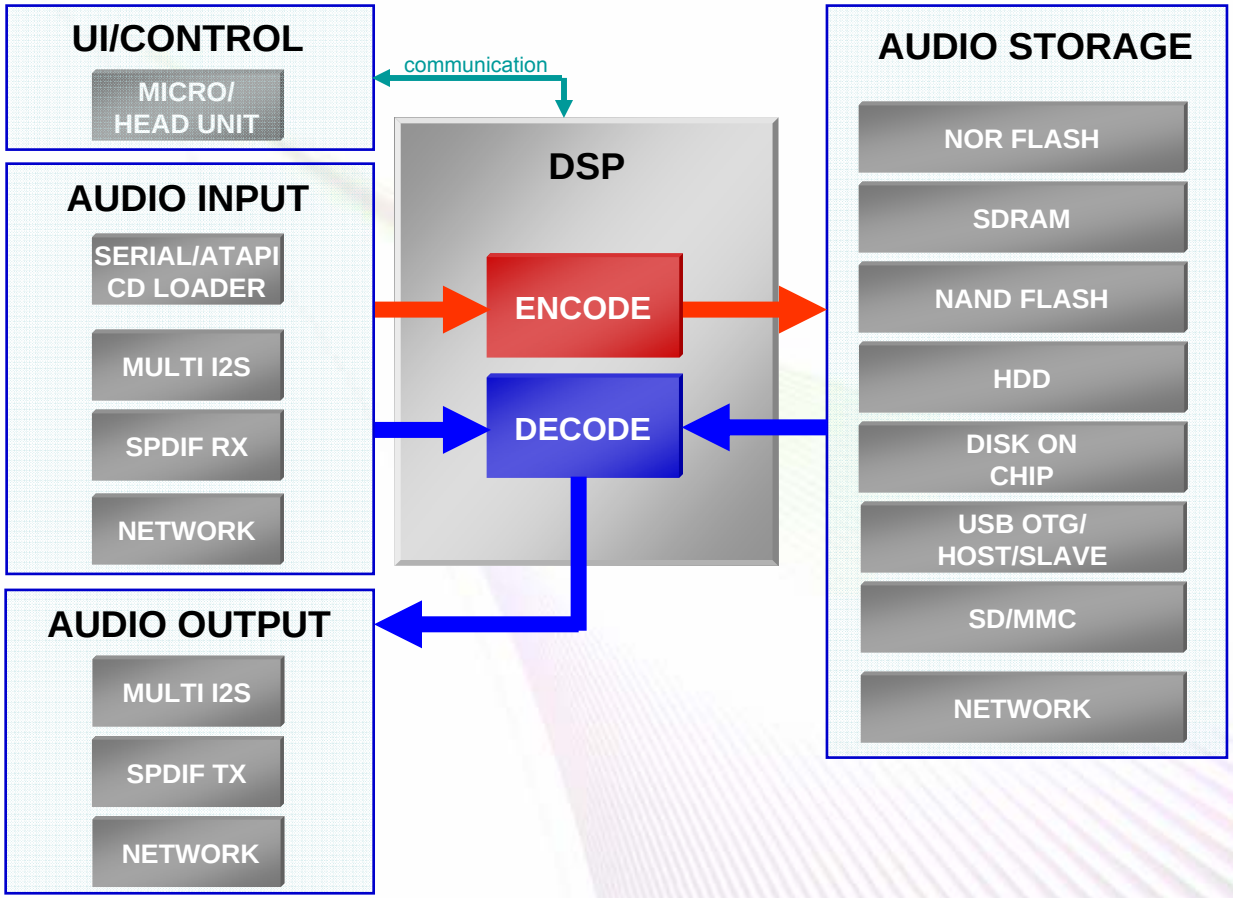
Why is Multi-Zone Relevant?

- Multi-Zone Home Entertainment System
  - Multiple sources, multiple sinks using a centralized processor

Why is Multi-Zone Relevant?

- Two-Way Viewing Angle (TWVA) TV
- "Directed" Audio is necessary for dual sources to arrive at distinct listening positions coordinated with the video.

Why is Multi-Zone Relevant?



- Jukebox/Network/Disk-based Media Systems

How to Realize Multi-Zone on a DSP?

- Hardware
  - Requires support of peripherals
    - Multiple, unrelated clock zones
    - DMA events
- Software
  - Requires use of operating system to manage resources
    - Would you want to hand schedule this?
  - Memory reuse
  - Thread protection
    - Drivers
    - DMA resources
  - Scheduling algorithm

Multi-Zone Scheduling

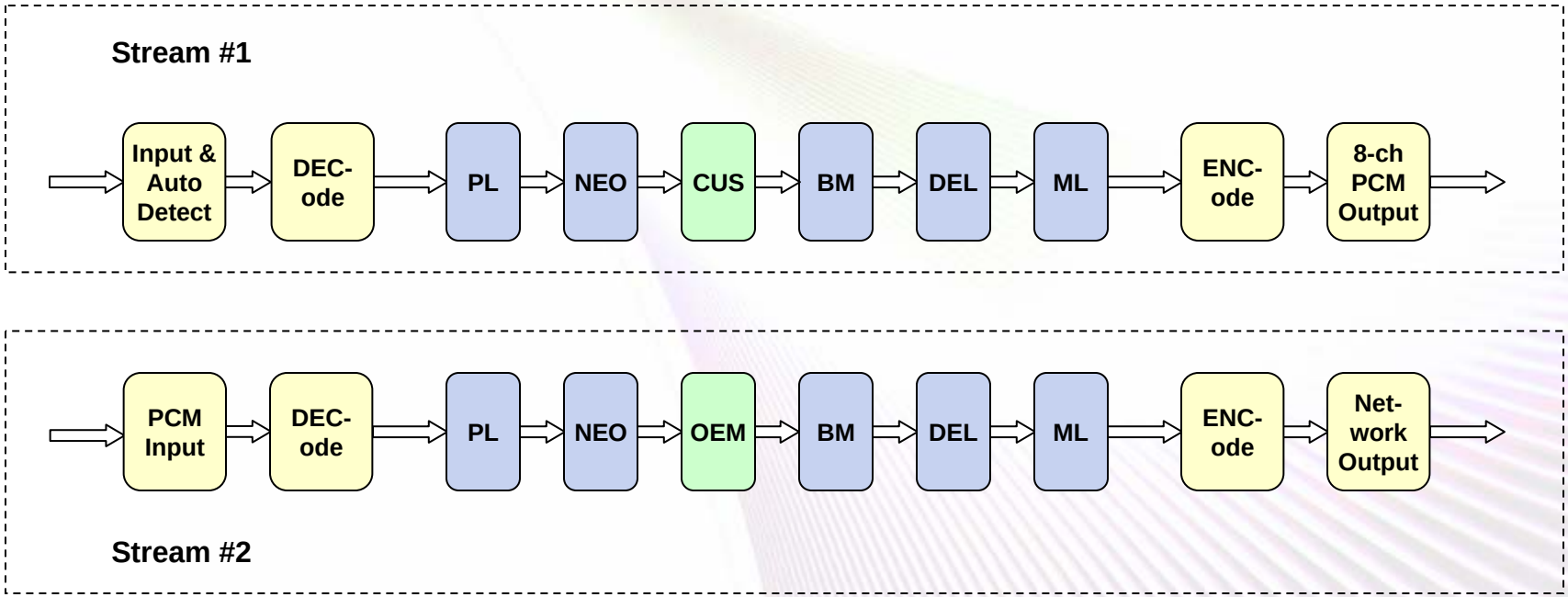
- System:
  - Two asynchronous audio threads at same priority
- Option 1: Run each thread until blocked (co-operative multi-tasking)
  - + Simple
  - – May miss real-time (unless additional latency introduced)
  - – Separate memory space might be required for each thread
- Option 2: Use time-slicing to allow equal time (pre-emptive multitasking)
  - + Good for threads with similar processing load
  - + Able to use Simple RTOS
  - – Separate memory space for each thread
  - – Overhead for context switch
  - – May leave MIPS on the table
- Option 3: More advanced options (deadline scheduling)
  - Use more Advanced RTOS

Multi-Zone Realization

- I/O concerns
  - Buffering required to keep up with peripherals when task is inactive
  - Latency
  - Simplify scheduling by synchronizing threads (asynchronous rate conversion)
- Memory
  - Cache and scratch (inefficient with pre-emptive scheduling)
- RTOS
  - Good method to realize pre-emptive scheduling

Multi-Zone Messaging

- Multiple Streams
  - ➔ Multiple Components
    - ➔ ***Unique IDs for each component***



TI Developer Conference
March 7-9, 2007 • Dallas, TX

Minds in Motion

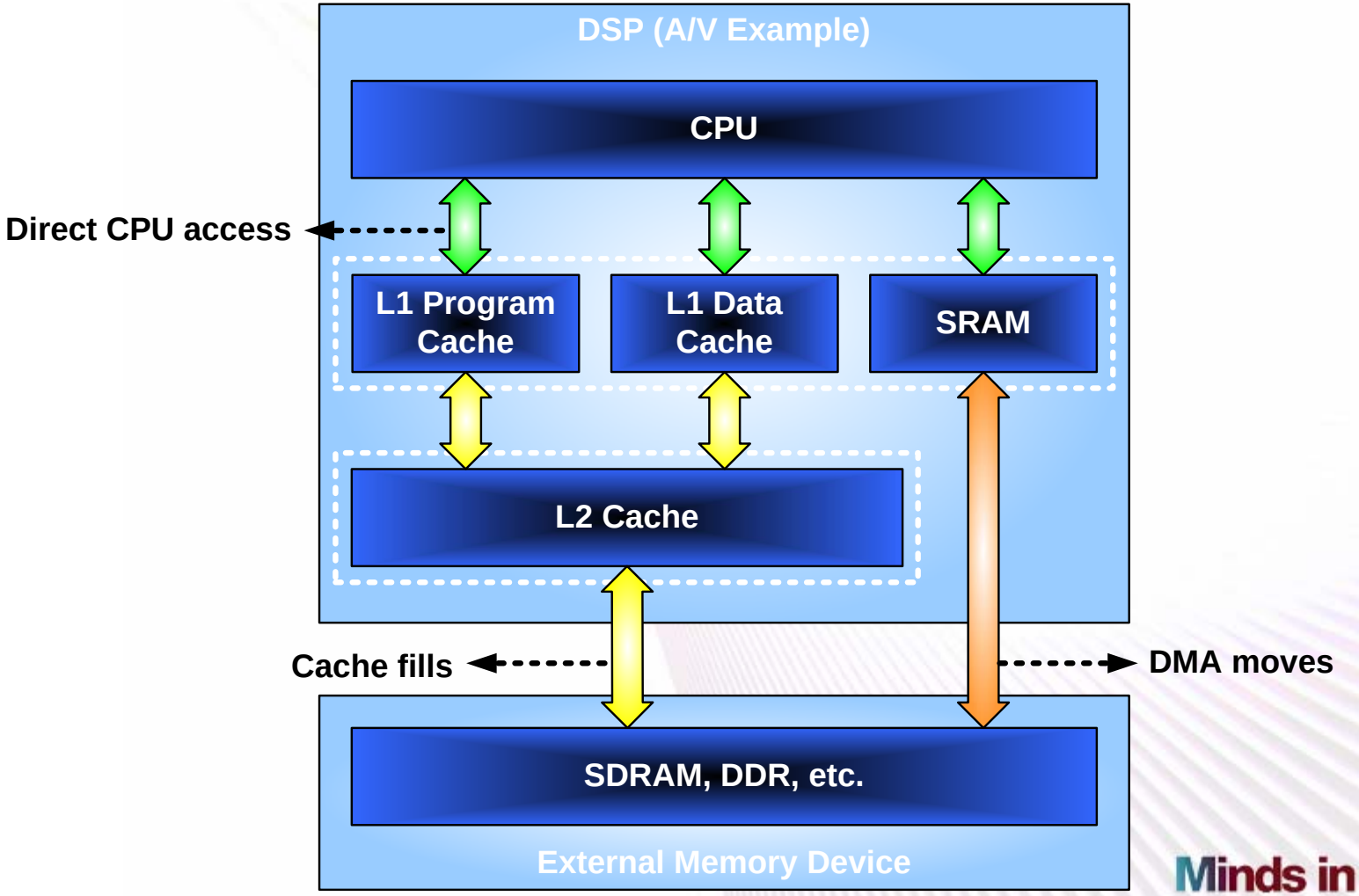
A/V Systems

Embedded DSP Systems Considerations with Audio Signal Processing

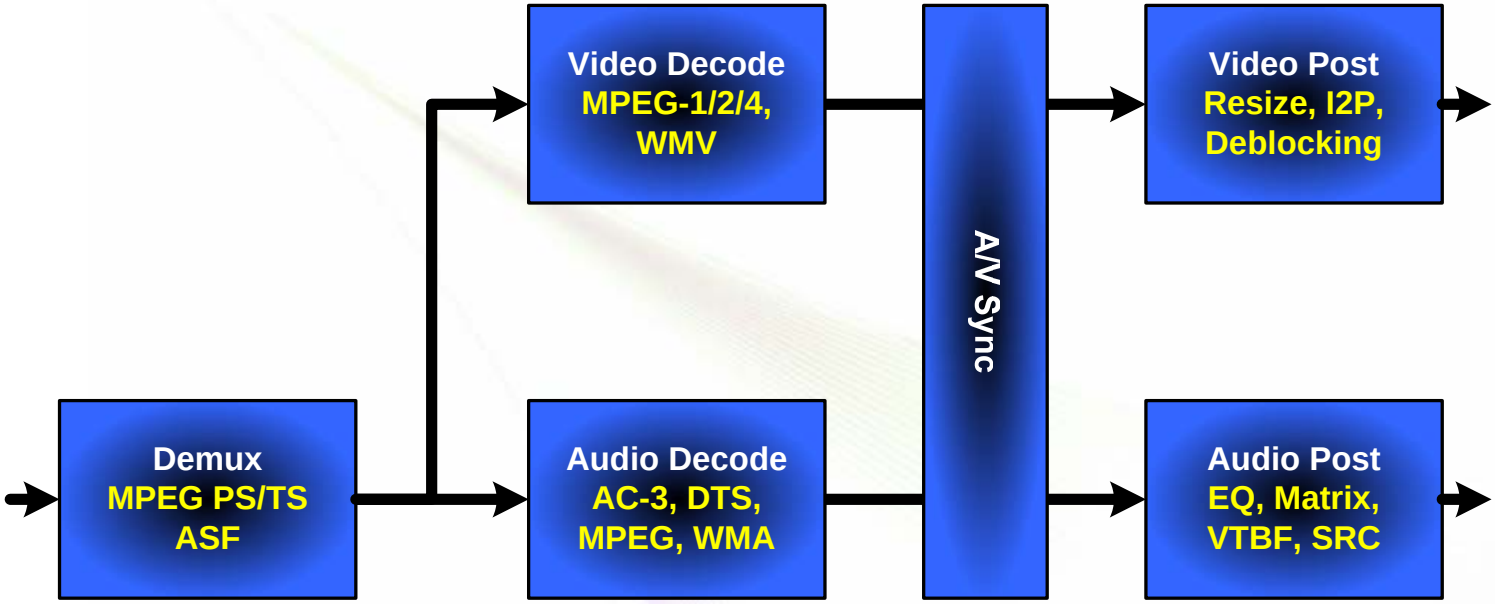
A/V Systems

- Architectures that can support concurrent audio/video are possible
  - Hardware differences include:
    - External memory is mandatory
      - Image buffers are large
    - Video related peripherals
      - Display controllers, Analog video DACs, Digital video outputs, OSD
  - Software differences include:
    - Video processing has higher memory and MHz requirements compared to audio processing
    - Video does not require as much precision as audio

Example A/V Memory Configuration

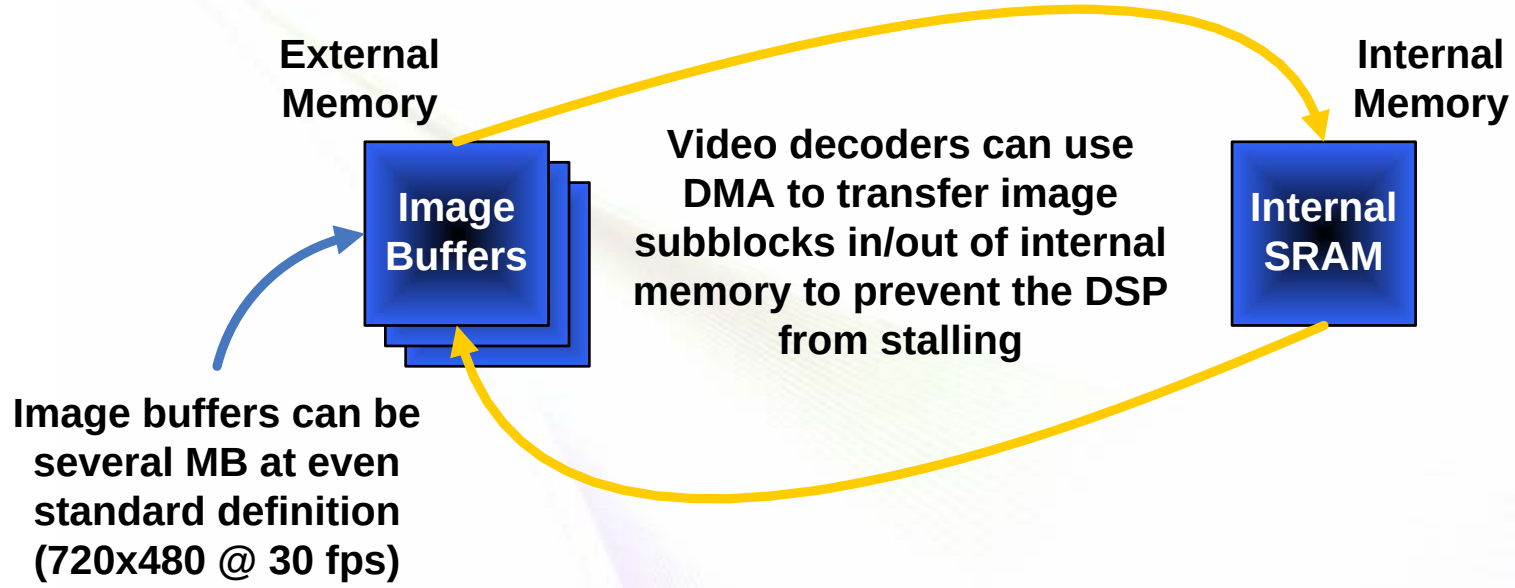


Example Digital Video System



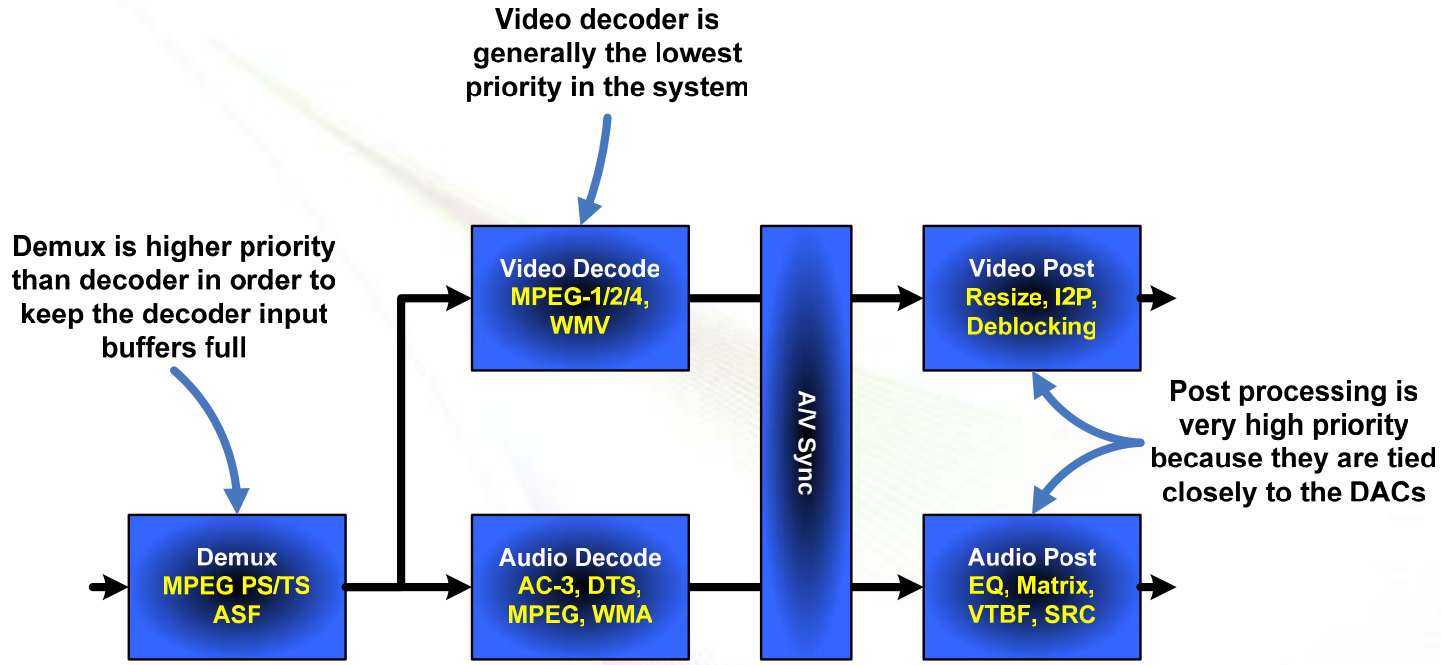
- Example configuration:
  - DVD-V: MPEG Program Stream with MPEG-2 Video + AC-3/DTS Audio
  - ATSC HDTV: MPEG Transport Stream with MPEG-2 Video + AC-3 Audio
  - Internet Video: ASF with WMV Video + WMA Audio

Compressed Video Decoders



- External memory accesses are more random compared to audio so the on-chip cache performance will degrade – keep the video data from flowing through the cache
- Large amount of memory access can be required
  - MPEG-2 video decoder will peak at ~75 MB of external memory access *per second* for standard definition

AV System Considerations



- A relatively complex subsystem like this will require several tasks each with different priorities – RTOS can handle task scheduling
- Most of the components in this system have fairly linear access which will provide good cache performance. This limits the amount of memory moves required by the system