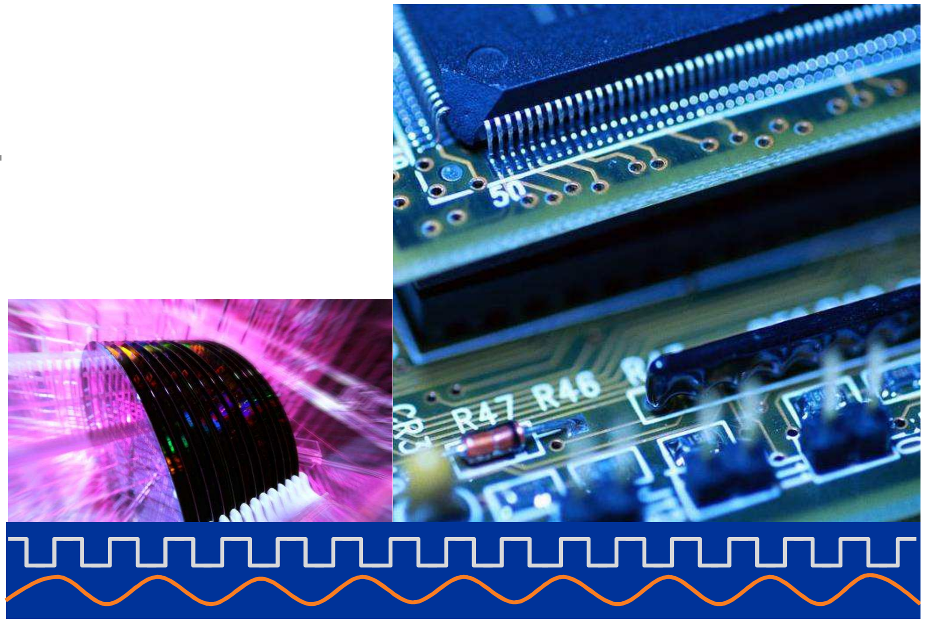
Android Platform Optimizations

ELC-Europe
Prague, October 2011
Ruud Derwig



Helping Design the Chips Inside

Mobile Multimedia ● **Digital Home** ● **Data Center & Networking** ● **Computing & Peripherals**

Medical ● **Automotive** ● **Industrial** ● **Military / Aerospace** ● **Other**

Agenda

- Market & value drivers
- What to optimize?
- How to optimize?
- Results & conclusion

Android Markets

- **Smartphones**
- **Tablets**
- TV
- STB / multimedia

- Others / new

Android Markets

- Smartphones
- Tablets
- **TV**
- **STB / multimedia**

- Others / new

Android Markets

- Smartphones
- Tablets
- TV
- STB / multimedia

- **Others / new**

Key Value Drivers & System Architecture Choices

- Power consumption
  → optimize performance / mW

- Product cost
  → optimize performance / area
  → optimize development efficiency

- Hardware – Software trade-offs
  - Maximum flexibility & developer efficiency : “virtual everything”
    - PC model, multi-GHz SMP processor centric designs
  - Minimal power & optimal performance: “dedicated hardware”
    - dedicated, fixed function device
  - ***Sweetspot : “heterogeneous, HW accelerated multi-core”***
    - ***Mix of CPU, DSP, and dedicated HW***

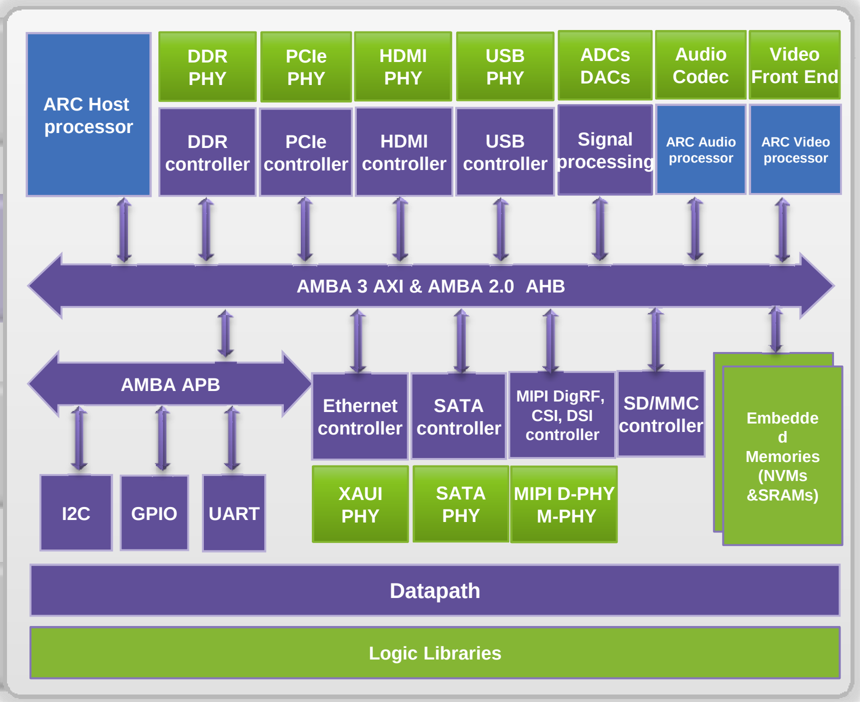
Agenda

Market & value drivers

What to optimize?

How to optimize?

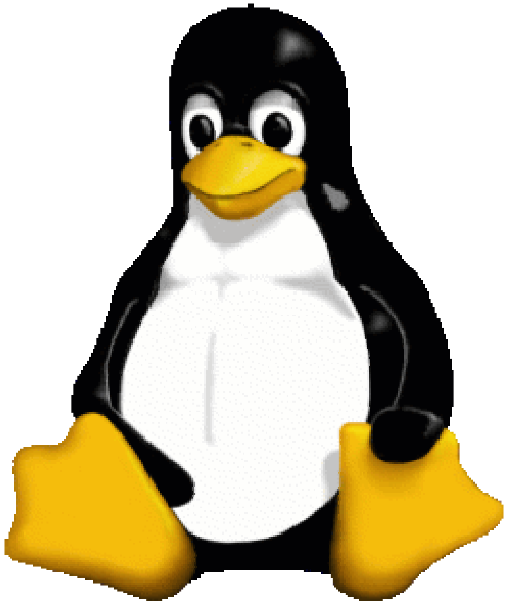
Results & conclusion



Linux Kernel & Library Optimizations

- Important,
- … but not Android specific

- Optimization options
  - Optimize hotspots
    - compiler
    - handwritten assembler
  - CPU hardware optimizations
    - MMU
    - special instructions

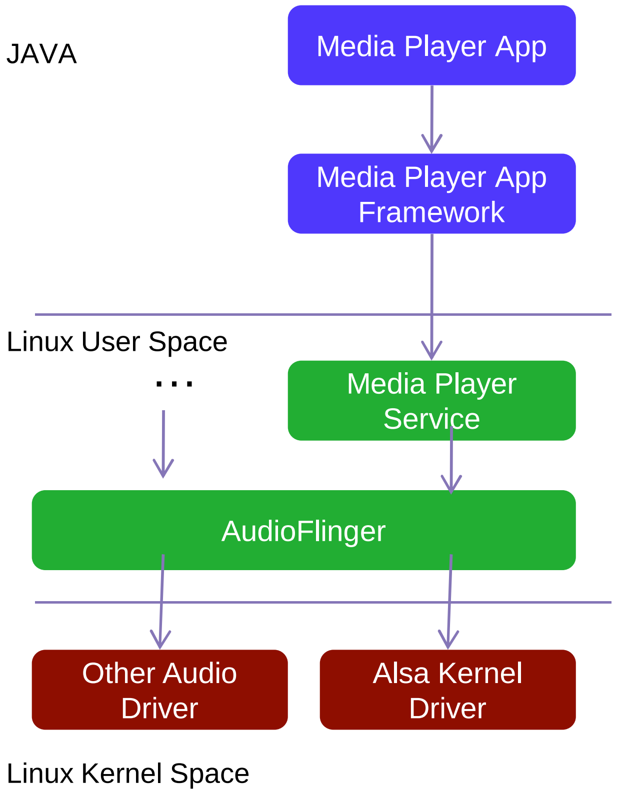


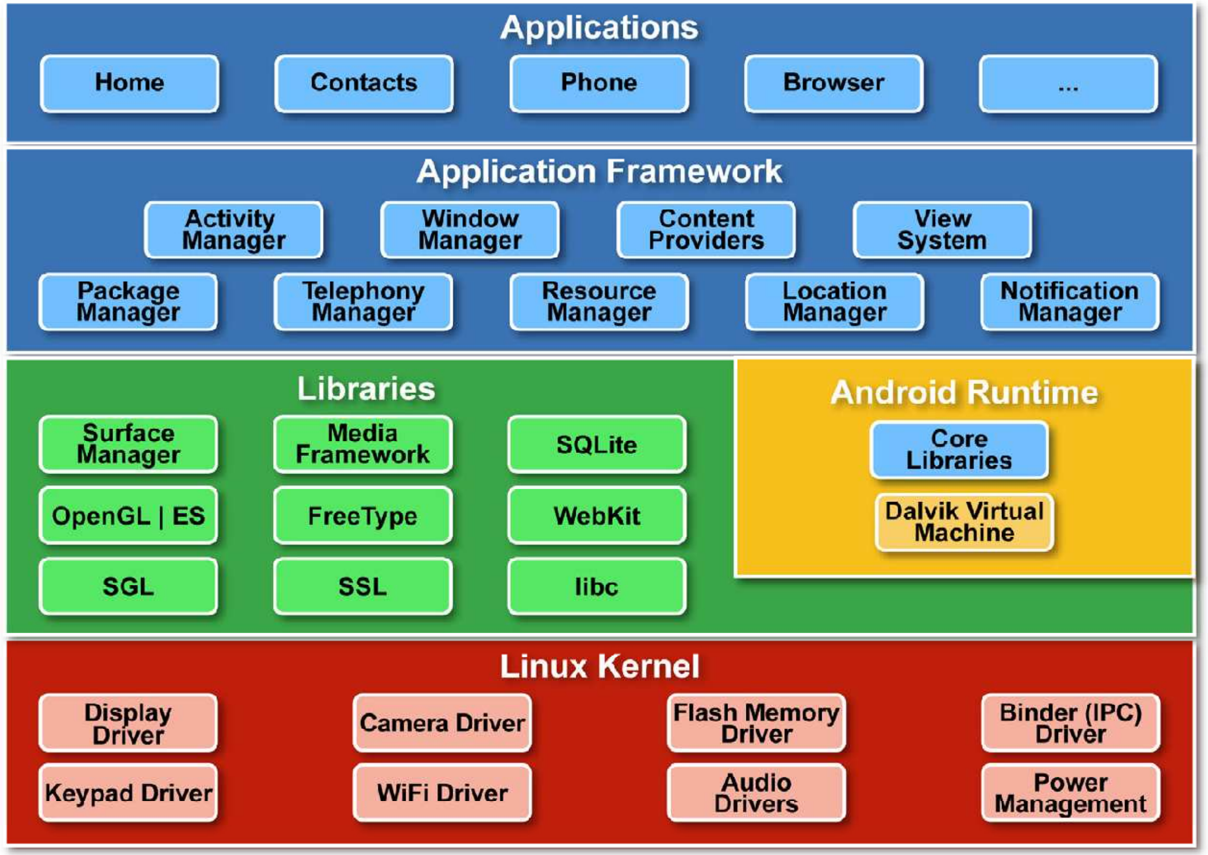
Dalvik Virtual Machine

- "Java" * virtual machine
  - Register-based architecture (Java VMs are stack machines). Dalvik registers are typically stored in memory (on the stack, like local variables in C).
  - Own bytecode
- Three virtual machines
  - Portable: completely C-based, in fact one large **switch{}** statement with a **case** *x***:** for every Dalvik opcode.
  - Fast (a.k.a. MTERP): assembly-coded handlers for every Dalvik opcode, which are aligned on 64 bytes addresses, so that the address of the handler can be easily calculated from the opcode, saving a lookup.
  - JIT: just-in-time compiler, initially starts as fast/mterp interpreter, but will identify 'hot' traces and pass these to the compiler thread.

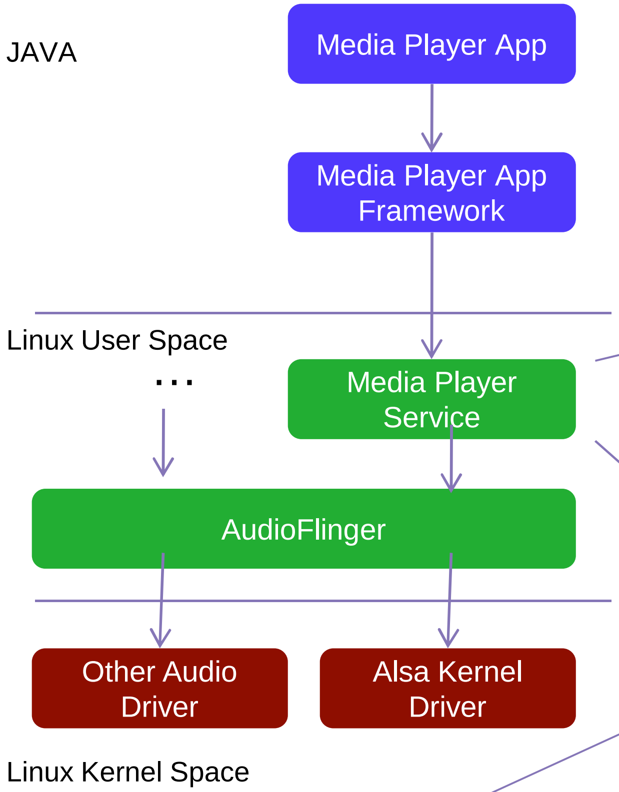
**Dalvik is a clean-room implementation of Java for copyright reasons. The syntax is similar.*

Android Media Player Architecture

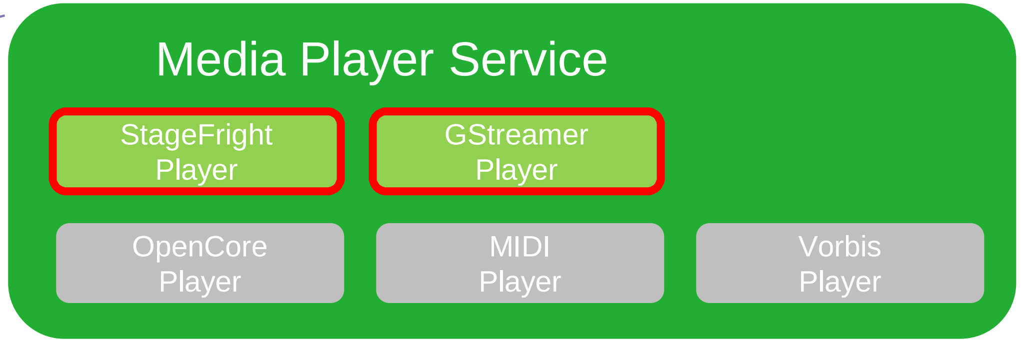


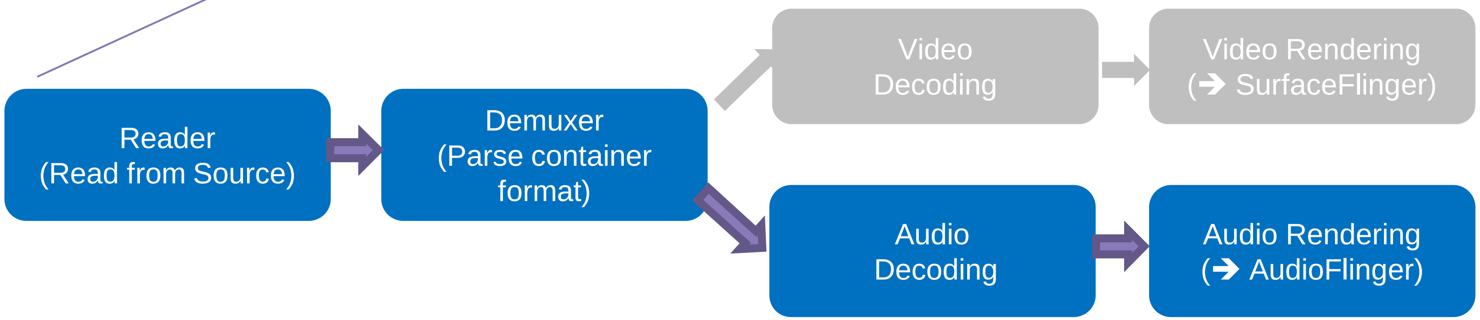


Android Media Player Architecture

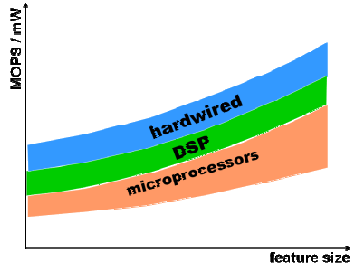


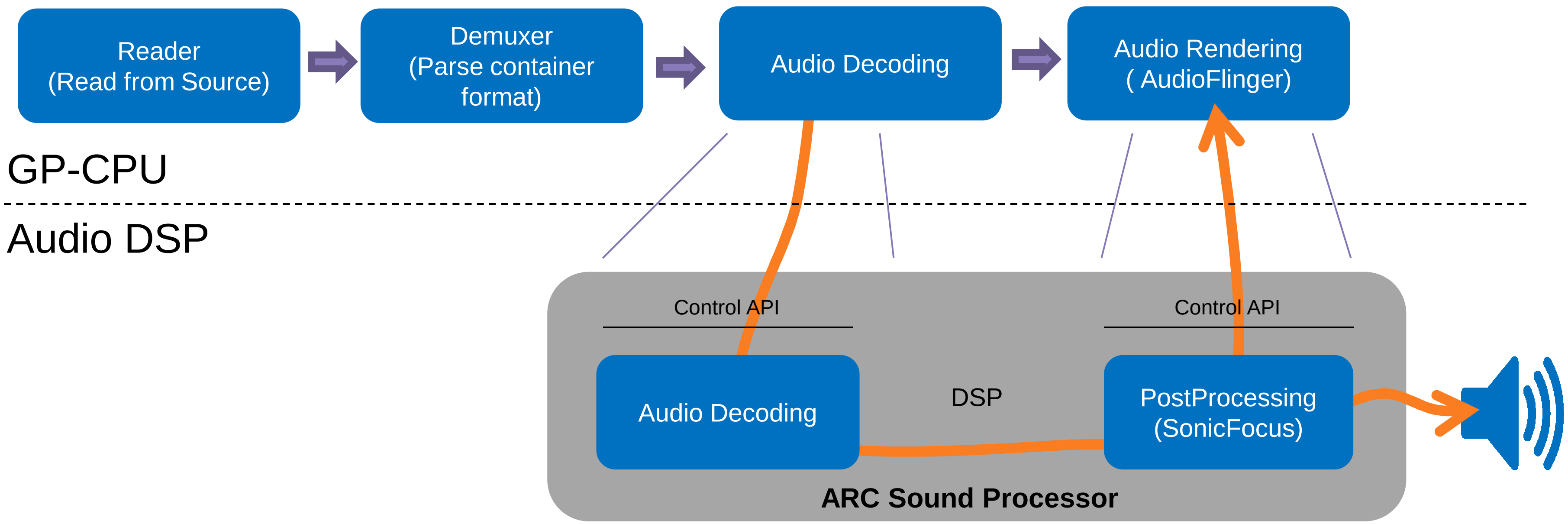
- Google's player of choice is the Stagefright
  - multi-format A/V player, newly developed for Android
  - Simple fixed graph – selects demuxer and decoder based on file extension
- Alternatives exist
  - Gstreamer based
  - proprietary / legacy





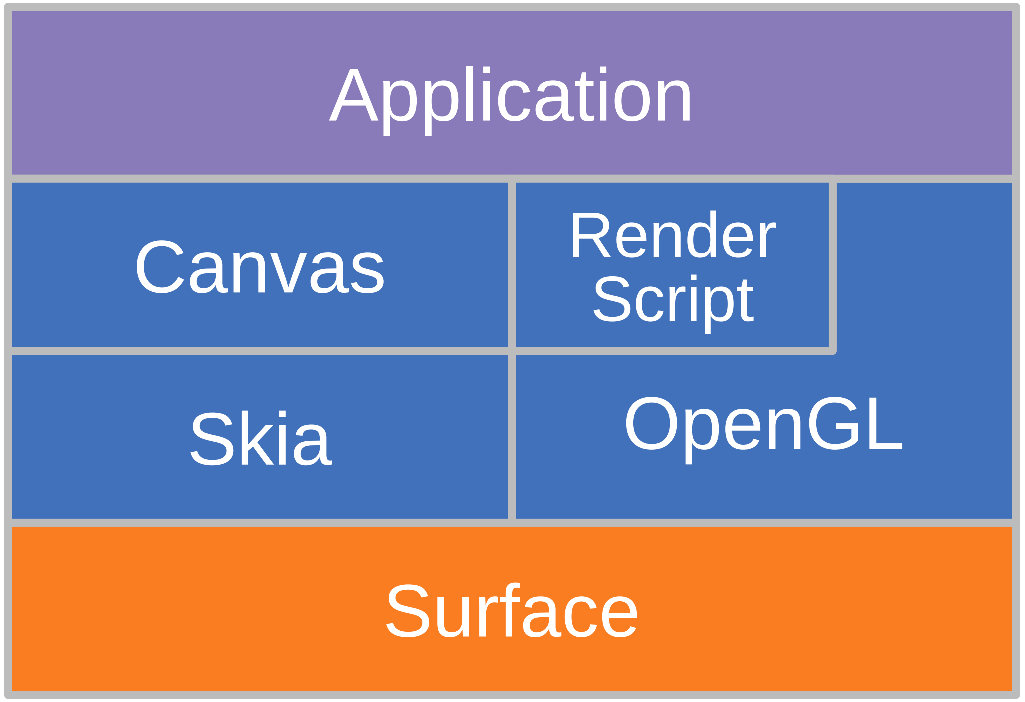
Audio Optimization Option:
off-load audio processing to DSP



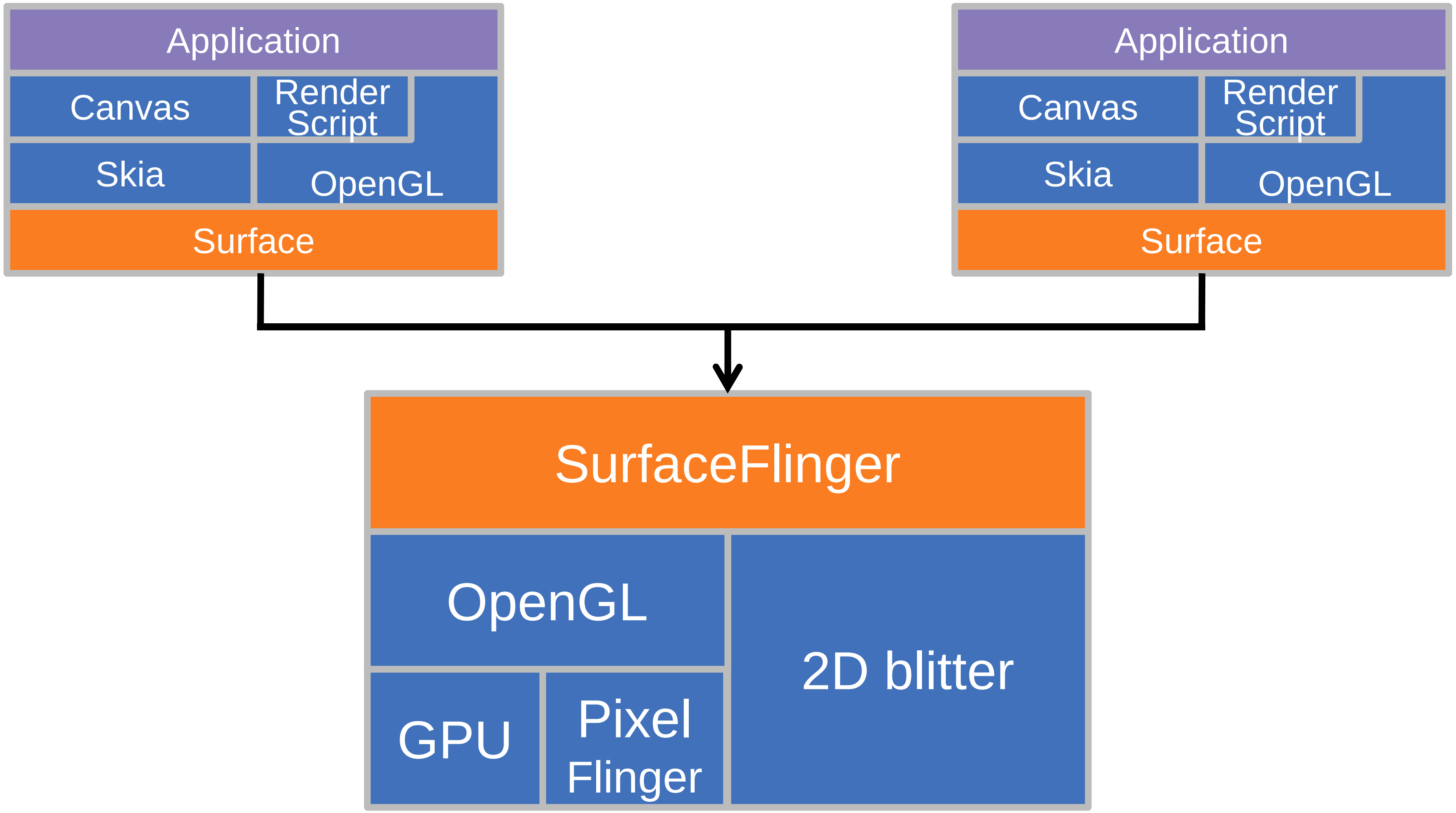


Android Graphics - Architecture

- 2D
  - Canvas/Skia
  - OpenVG
- 3D
  - OpenGL-ES 1.x
  - OpenGL-ES 2
- Renderscript
  - Expose native GPU/SMP to (portable) applications
  - C99 ->LLVM intermediate bitcode -> machine code



Android Graphics - Compositor



Graphics Optimization Options

- Graphics drawing/rendering
  - Software/assembler optimization
    - Skia, PixelFlinger
  - Hardware acceleration
    - GPU (OpenGL-ES 2)
    - 2D accelerator (OpenVG compatible or other)
    - Memory architecture, caching
  - Renderscript

- Surface Composition
  - Scaling, colorspace conversion
    - Custom instructions
    - GPU
    - Dedicated hardware acceleration (bitblit)

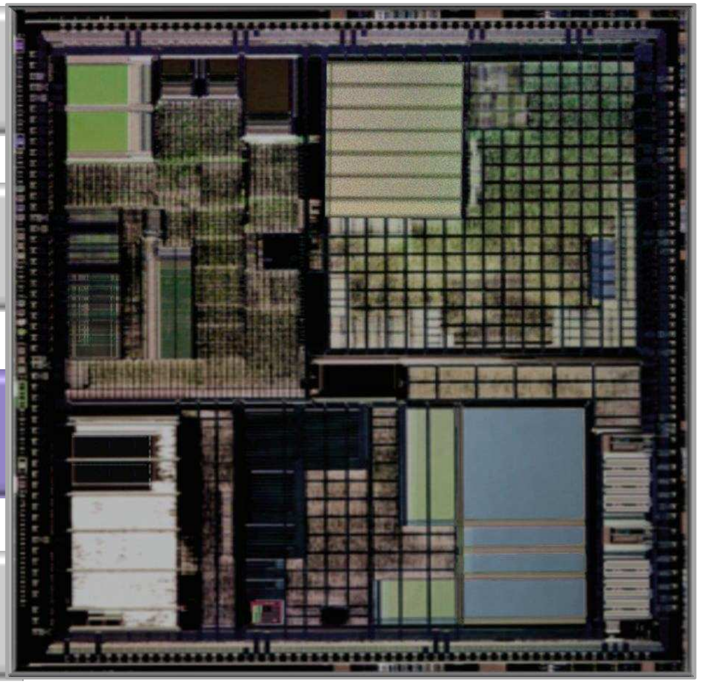
Agenda

Market & value drivers

What to optimize?

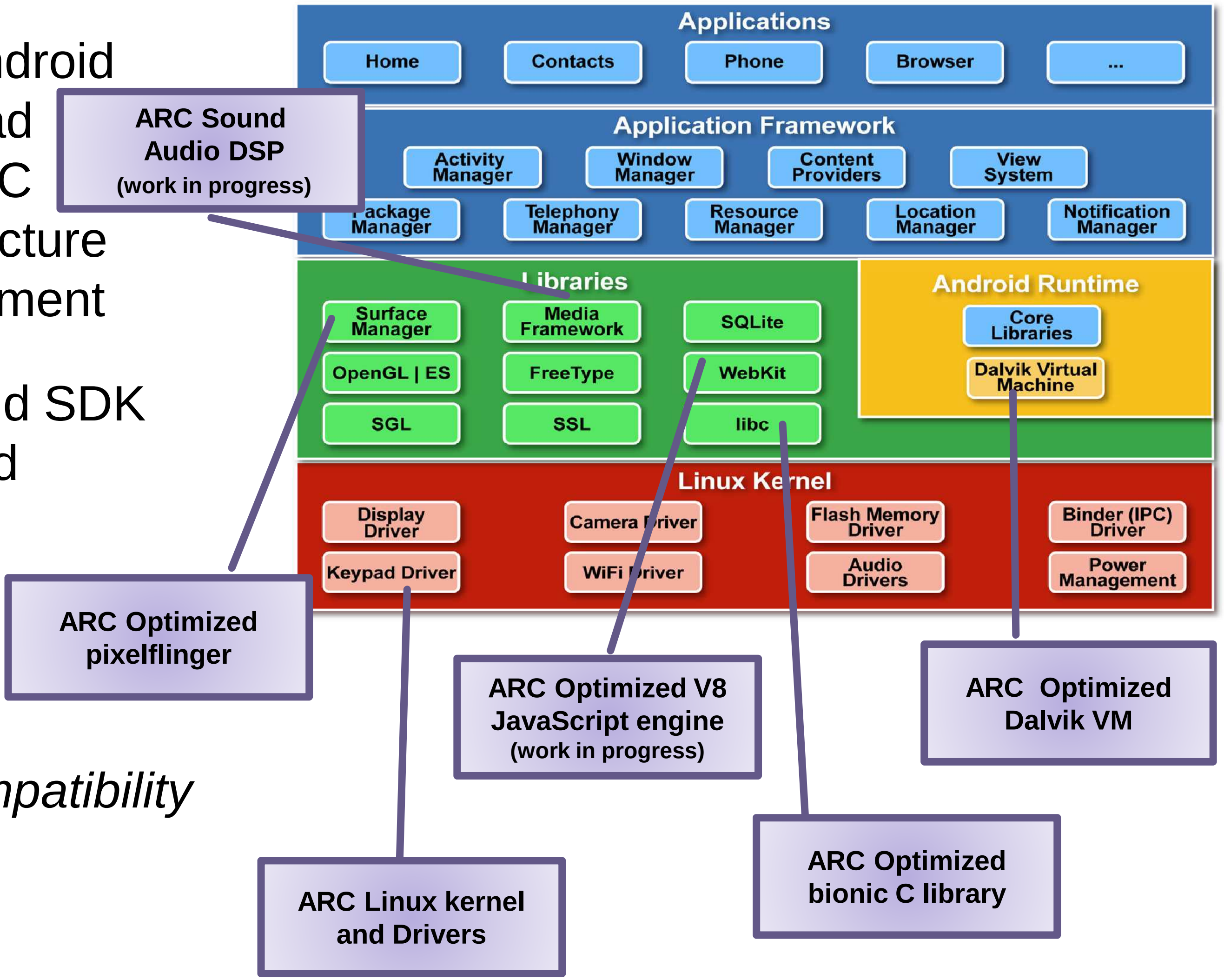
How to optimize?

Results & conclusion



Optimized Designware ARC Android

- Full port of the Android Froyo/Gingerbread release to the ARC processor architecture and build environment
- Including NDK and SDK to support Android application building/porting
- Google/OHA *Compatibility Test Suite* tested

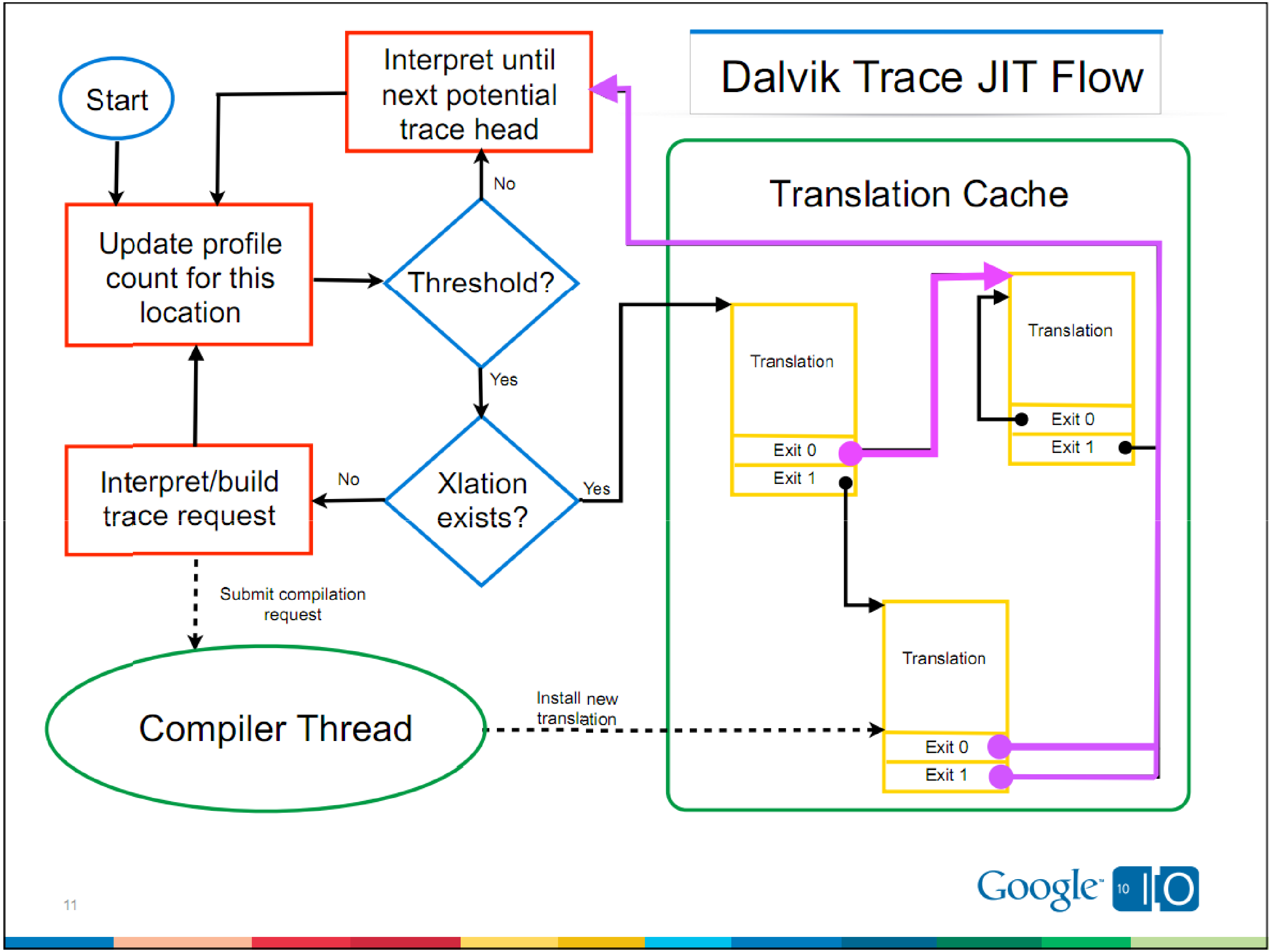


Differences between VM Implementations

Portable	MTerp	JIT

`switch (opcode) {`
`case add: a = b + c;`
`          break;`
`case sub: a = b – c;`
`          break;`
`...`
	`ld r0, [b]`
	`ld r1, [c]`
	`add r0, r0, r1`
	`st r0, [a]`
	`ld r0, [next_opcode]`
	`asl r0, r0, 6`
	`add r0, r13, r0`
	`j [r0]`
		`ld r0, [b]`
		`ld r1, [c]`
		`add r0, r0, r1`
		`st r0, [a]`
		
		OR
		
		`add r20, r20, r21`

```
ld r0, [next_opcode]
<pipeline stall>
ld.as r1, [jump_table, r0]
<pipeline stall>
j [r1]
```



Reused from Google I/O presentation

Register- and Stack-based VMs

Example: **a = b + c**

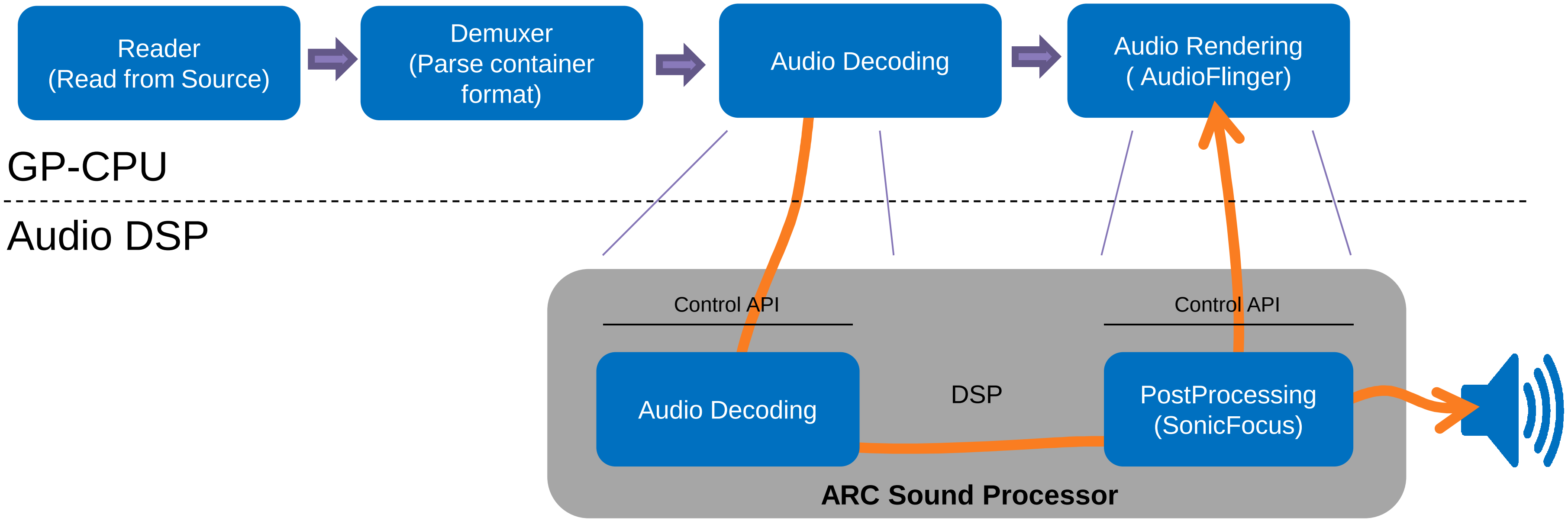
Java	Dalvik	Dalvik for ARC

`iload b`
`iload c`
`iadd`
`istore a`
	`add-int a, b, c`	`add-int a, b, c`

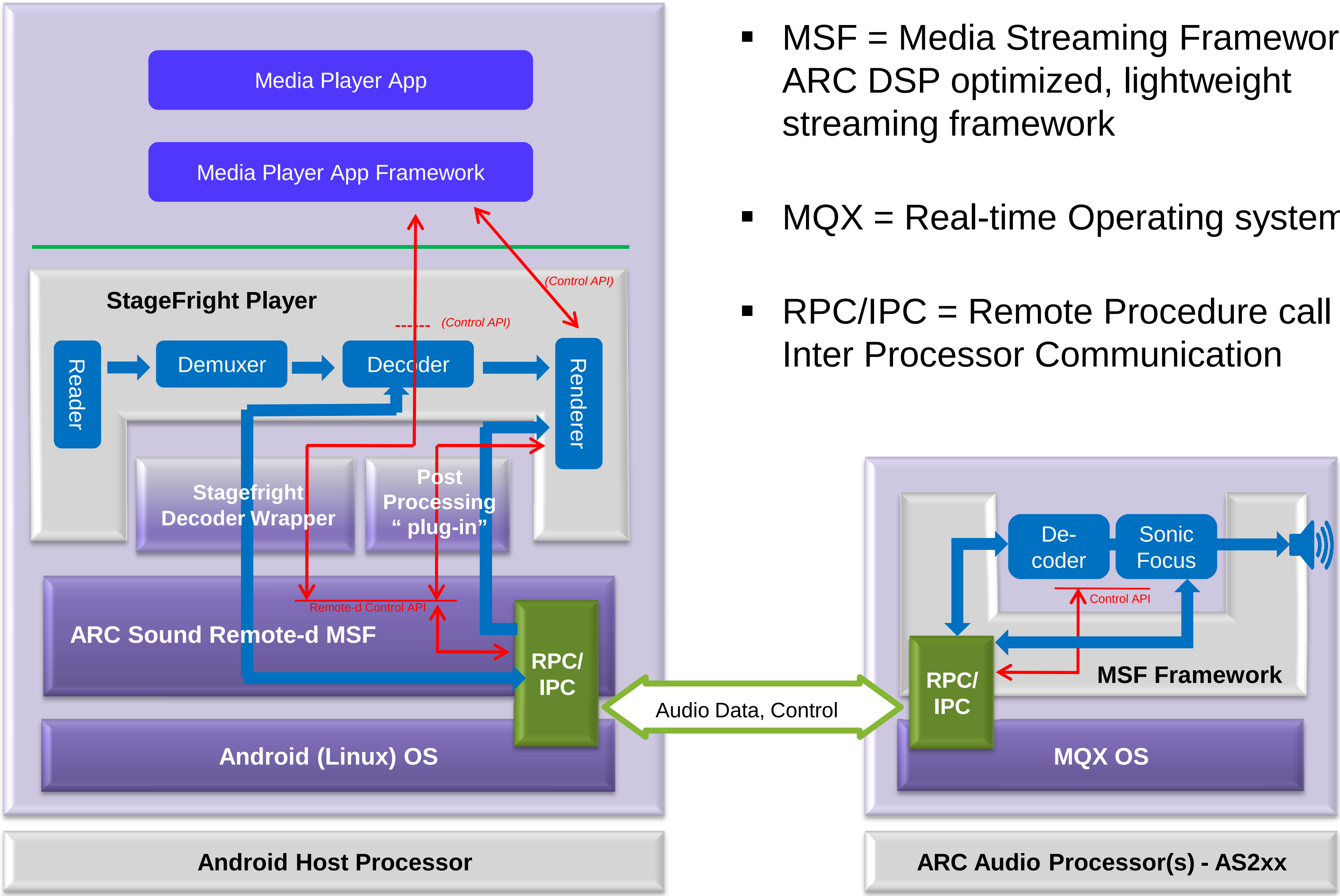
`ld r0, [b]`
`push r0`
`ld r0, [c]`
`push r0`
`pop r0`
`pop r1`
`add r0, r0, r1`
`push r0`
`pop r0`
`st r0, [a]`
	`ld r0, [b]`
	`ld r1, [c]`
	`add r0, r0, r1`
	`st r0, [a]`
		`add r20, r21, r22`
		
		Registers are only saved/restored when changing stack frames or when moving to interpreter

SYNOPSYS®
Predictable Success

Audio Processing on DSP



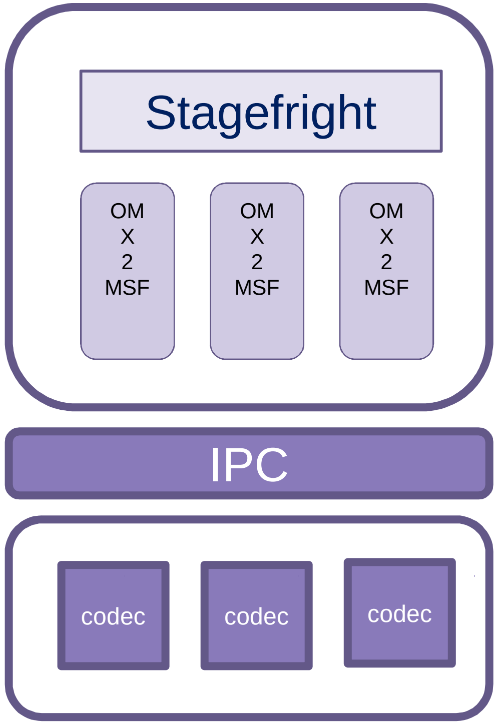
- Audio decoding and Post-processing off-loaded to ARC Sound Processor
- Special host Audio Decoder implementation that takes care of off-loading
  - with standard host decoder interfaces, so seamless integration
- Post-processing control through Renderer on host (special Renderer or Renderer plug-in component)



- MSF = Media Streaming Framework ARC DSP optimized, lightweight streaming framework
- MQX = Real-time Operating system
- RPC/IPC = Remote Procedure call / Inter Processor Communication

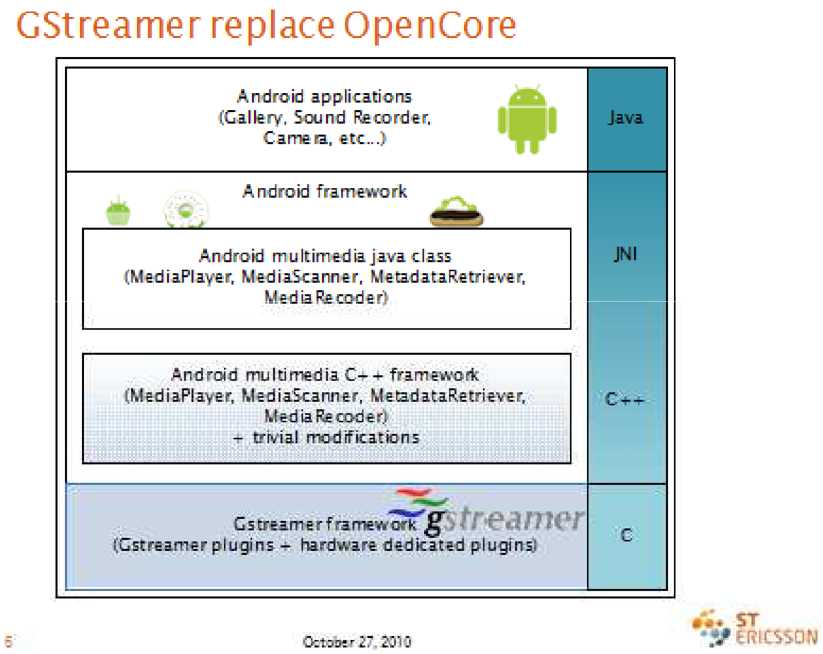
Android & Audio APIs

- Stagefright supports 2 types of interfaces
  - OpenMax-IL : for re-use of OMX components
  - Stagefright codec interface : for native Stagefright codecs
- AudioFlinger uses dedicated interfaces
  - standard implementation using “ALSA” exist
  - developments ongoing (?) to support OpenSL-ES Khronos standard (like OMX)
- SNPS API choice not yet made
  - OMX-IL pro : open standard
  - OMX-IL con: efficiency, complexity: standard by committee…
  - Stagefright pro : efficient integration with Stagefright
  - Stagefright con : not an open standard, no deep tunneling



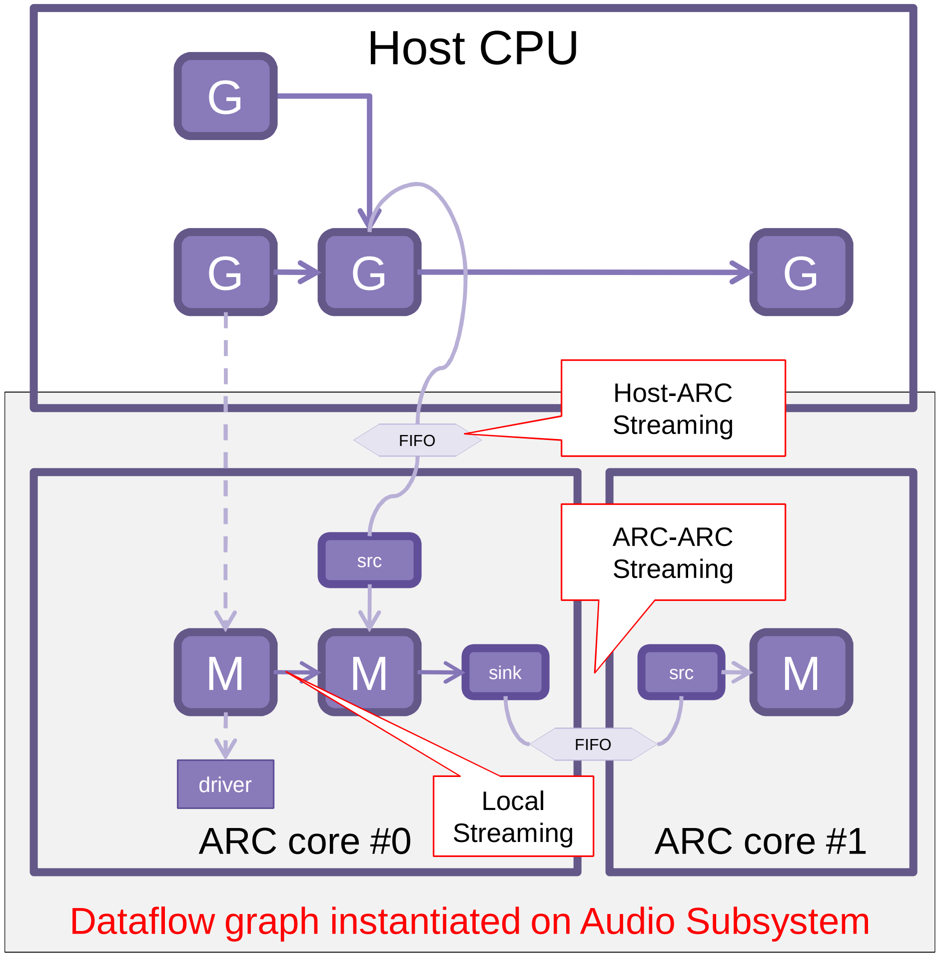
Alternative: Gstreamer

- GStreamer Android Player
  - see e.g. ELC-E 2010 presentation
- “The goal of the project is to both allow hardware makers to standardize on GStreamer accross their software platforms, but also to make the advanced functionality of GStreamer available on the Android platform, like video editing, DLNA Support and Video conferencing.”



GStreamer DSP Off-loading with "Deep Tunneling"

- Gstreamer-MSF integration makes heterogeneous multi-core SW development transparent to user
- Instantiation of Gstreamer element → instantiation of module on one of the ARC cores
- Creation of link → local connection or core-crossing connection between modules

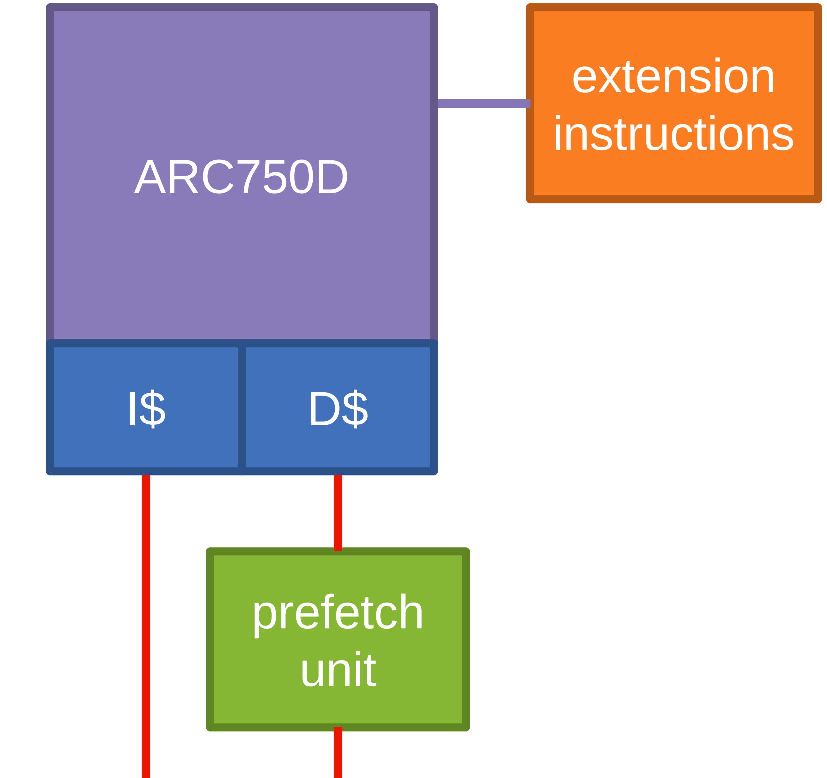


Gstreamer Deep Tunneling

```
static void connect_msf_outpin (GstPad* pad)
{
    GstPad         *peerpad = gst_pad_get_peer(pad);
    GstElement     *element = gst_pad_get_parent_element( pad );
    GstElement     *peerelement = gst_pad_get_parent_element( peerpad );
    GstAudioModule *filter = GST_AUDIOMODULE(element);
    guint32         result;

    if (!pad_is_deeptunnel(pad))
    {
        /* not a deep tunnel */
        /* create sink module */
        msf_api_sink_module_create(filter->msf_coreid, "Sink module", output_fifo_buffer,
                                   sink_pv_data, sizeof(sink_pv_data), &sink_module_id)));
        msf_api_connect_pins(filter->msf_moduleid, sink_module_id, 0, 0)));
    }
    else
    {
        if (pad_is_corecrossing(pad))
        {
            /* deep tunnel AND core-crossing */
            /* create sink module */
            msf_api_sink_module_create(filter->msf_coreid, "Sink module", filter->msf_sharedfifo,
                                       sink_pv_data, sizeof(sink_pv_data), &sink_module_id)));
            msf_api_connect_pins(filter->msf_moduleid, sink_module_id, 0, 0)))
        }
        else
        {
            /* deep-tunnel AND no core-crossing */
            guint32 peer_module_id;

            /* get the module id of the peer MSF module */
            g_object_get (G_OBJECT (peerelement), "msf_moduleid", &peer_module_id, NULL);

            msf_api_connect_pins(filter->msf_moduleid, peer_module_id, 0, 0)))
        }
    }
}
```


SYNOPSYS® Predictable Success

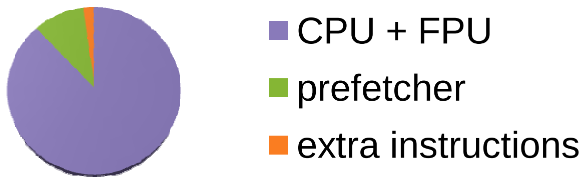
ARC HW Extensions

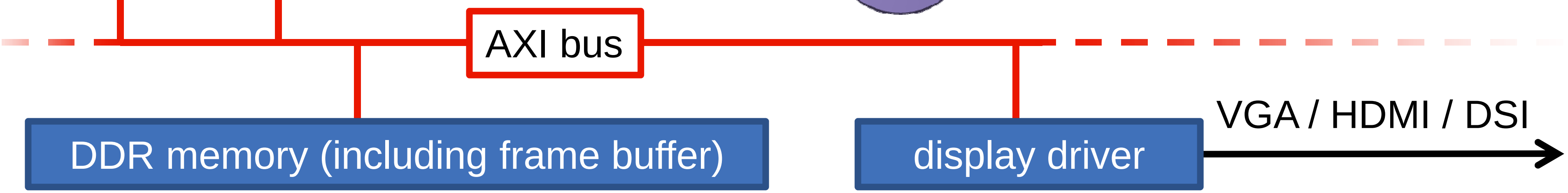


ARC EIA (Extension Interface Automation):
- supports user defined custom instructions
- accelerates typical Dalvik (Java VM) and pixelflinger (2D GUI) instruction sequences

Prefetcher:
- Eliminates pipeline stalls in high latency memory environments
- L2$ not required in this case
- Configurable depending on application

CPU area, excluding memories





Leveraging the ARC EIA Capabilities

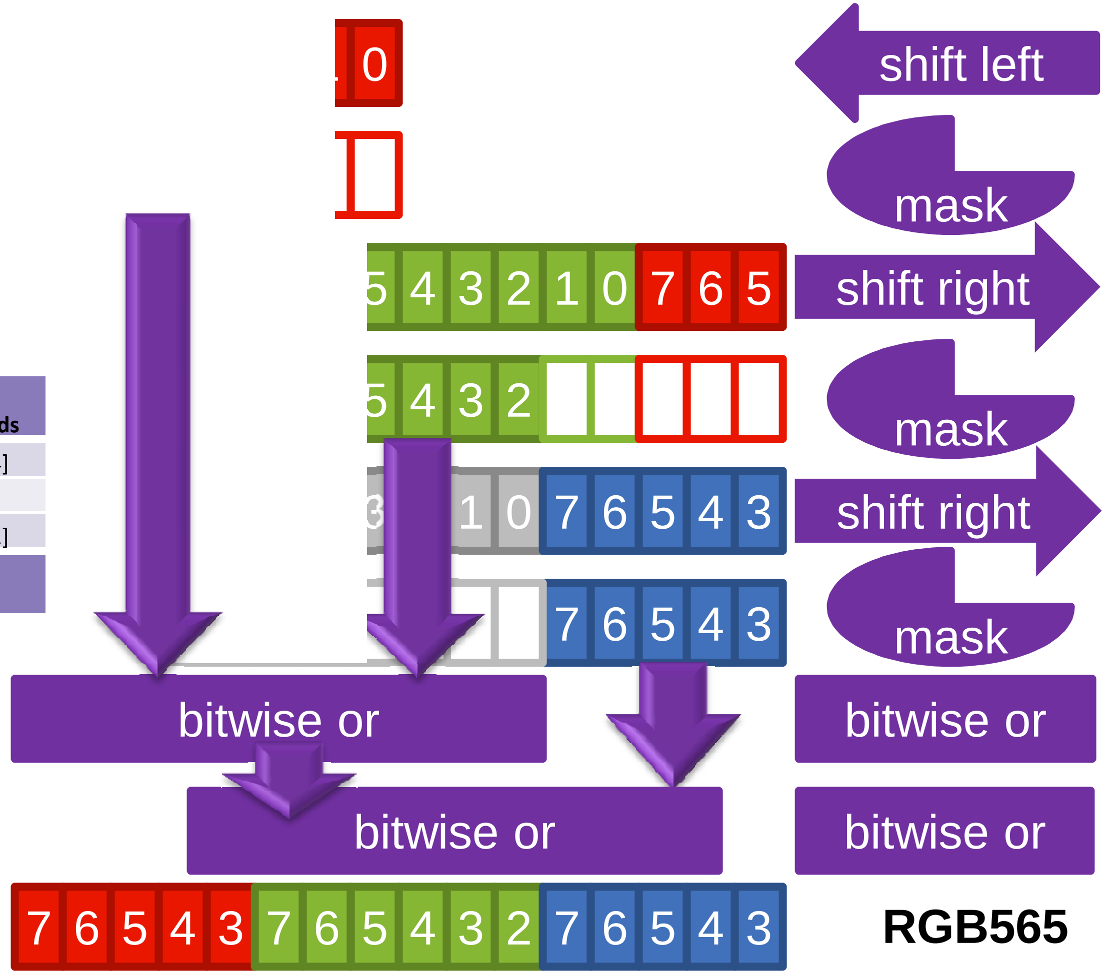
Example: Colour Space Conversion

8 operations are required for a conversion from ABGR8888 to RGB565.

This can be combined into one single EIA instruction.

Instruction	Operands
ld.ab	r1, [r4, 0x4]
and	r2, r1, 0xf8
asl	r2, r2, 8
and	r3, r1, 0xfc00
lsr	r11, r3, 5
or	r2, r2, r11
and	r3, r1, 0xf80000
lsr	r11, r3, 19
or	r2, r2, r11
stw.ab	r2, [r5, 0x2]

Instruction	Operands
ld.ab	r1, [r4, 0x4]
upk8	r2, r1, r6
stw.ab	r2, [r5, 0x2]

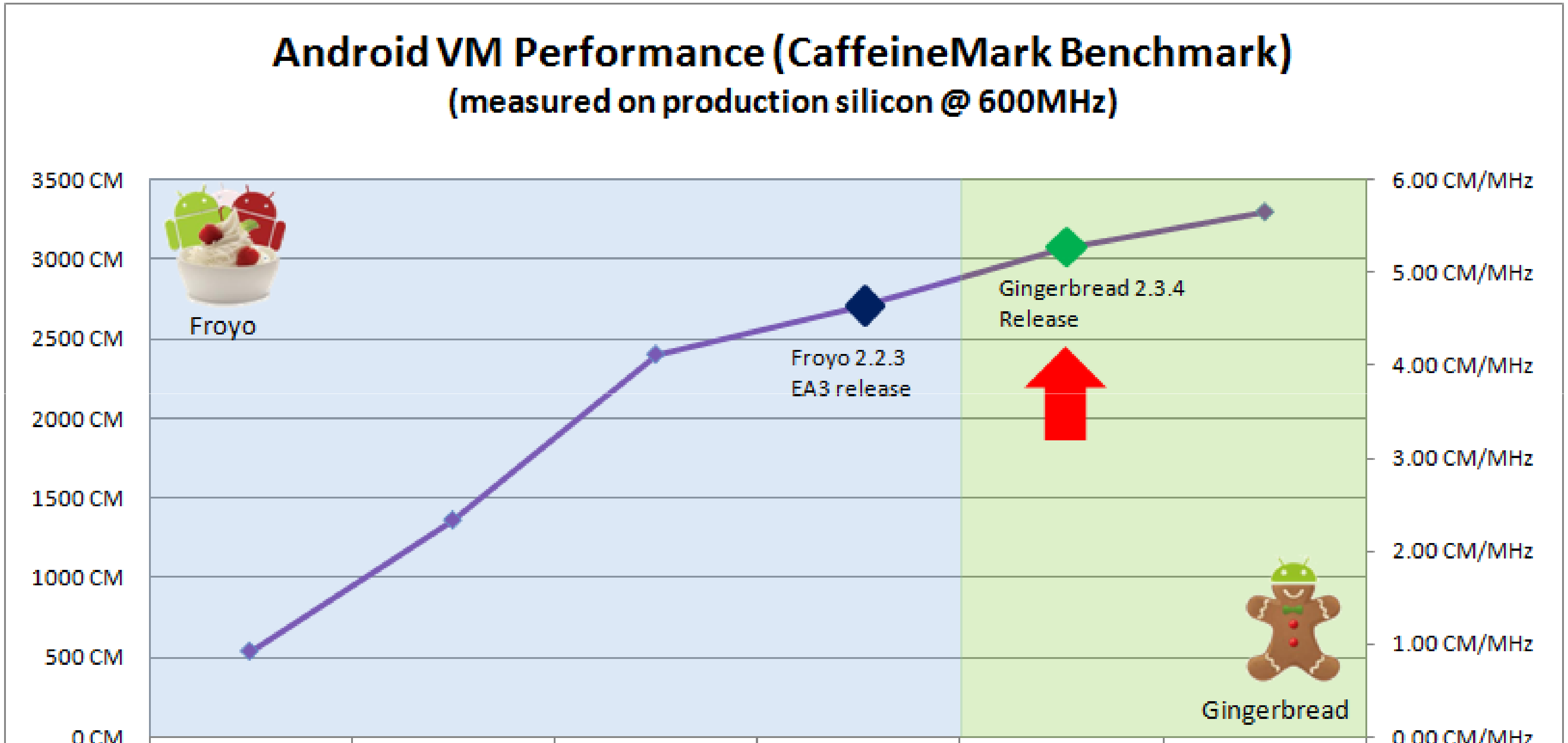


SYNOPSYS® Predictable Success

Agenda

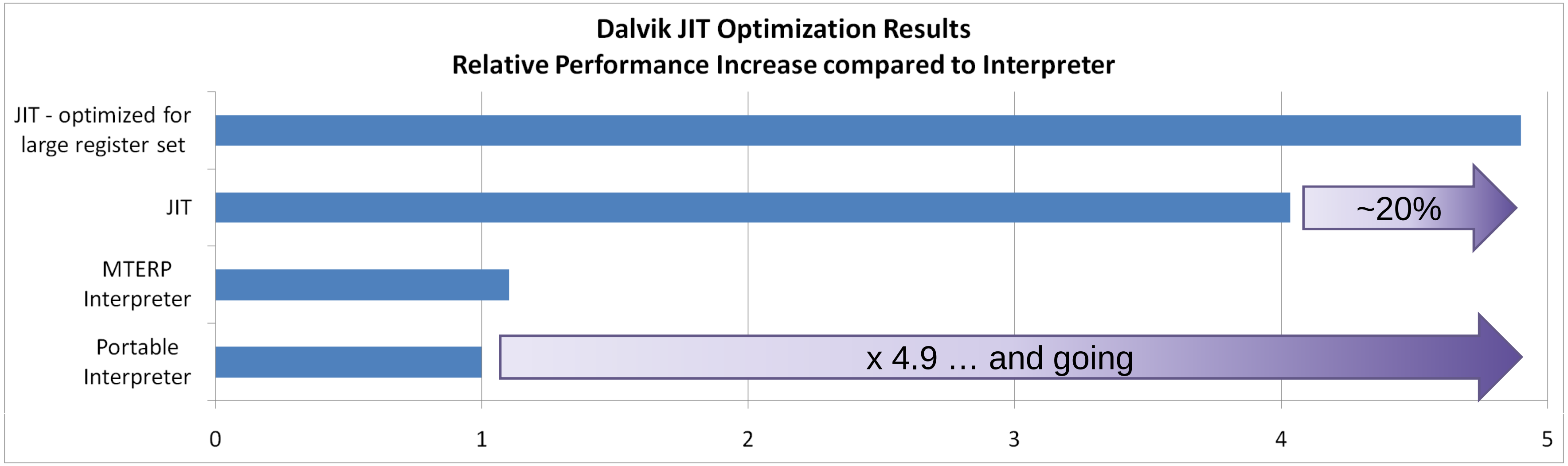
- Market & value drivers
- What to optimize?
- How to optimize?
- **Results & conclusion**

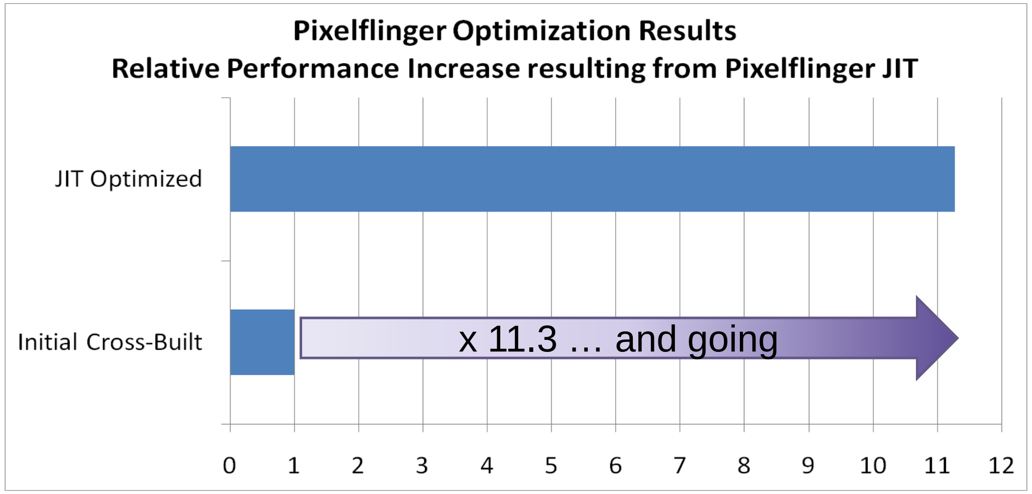
Optimizing Dalvik VM



Oct'2010	Nov'10	Feb'11	July'11	Aug'11	latest
535 CM	1358 CM	2400 CM	2700 CM	3071 CM	3296 CM
0.89 CM/MHz	2.26 CM/MHz	4.00 CM/MHz	4.50 CM/MHz	5.12 CM/MHz	5.49 CM/MHz

Optimizing Dalvik VM

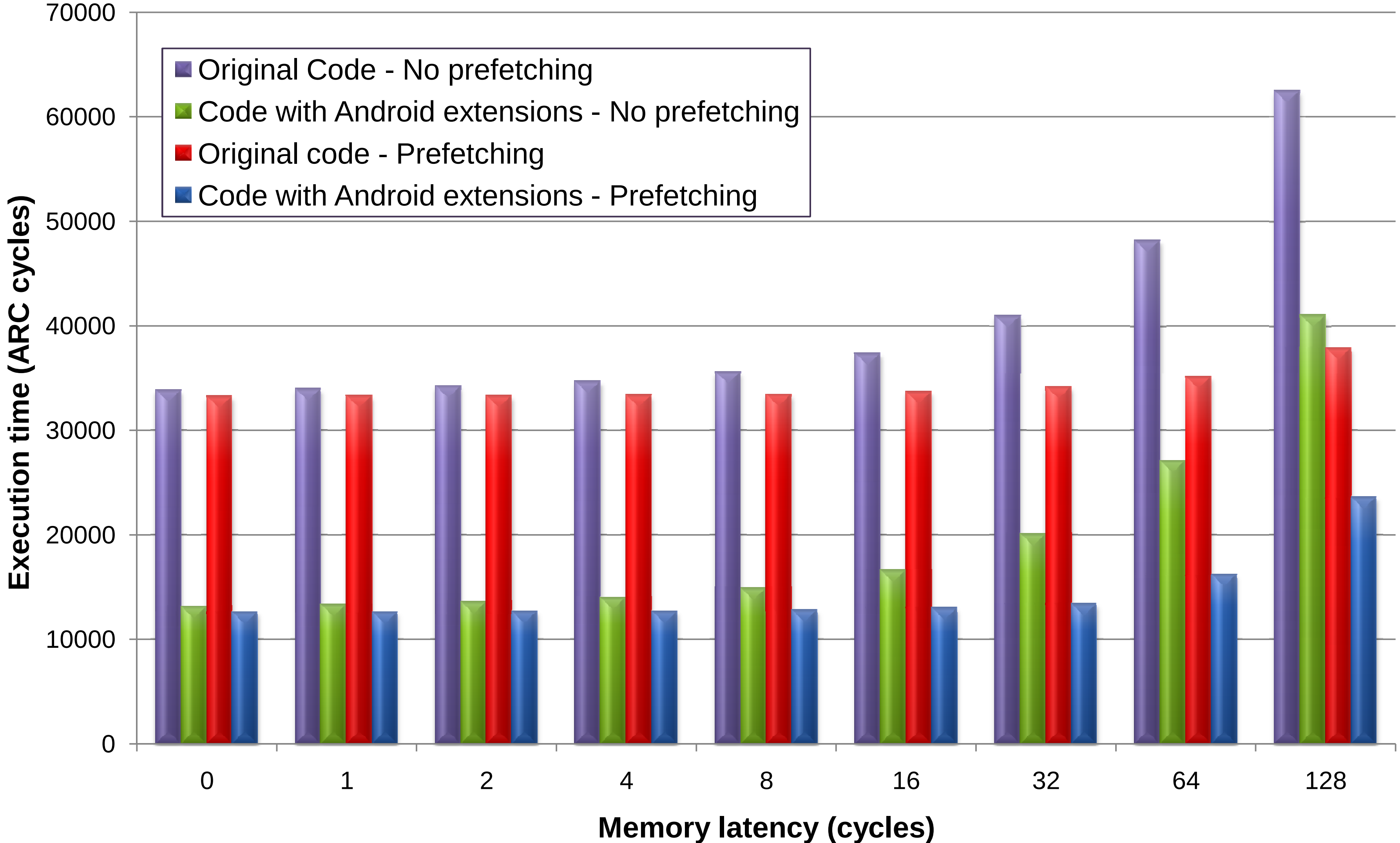




Core Mark	Caffeine Mark	*Without L2 cache*
1,9	4,9	/MHz
37	90	/mW
14	35	/MHz/mm^2

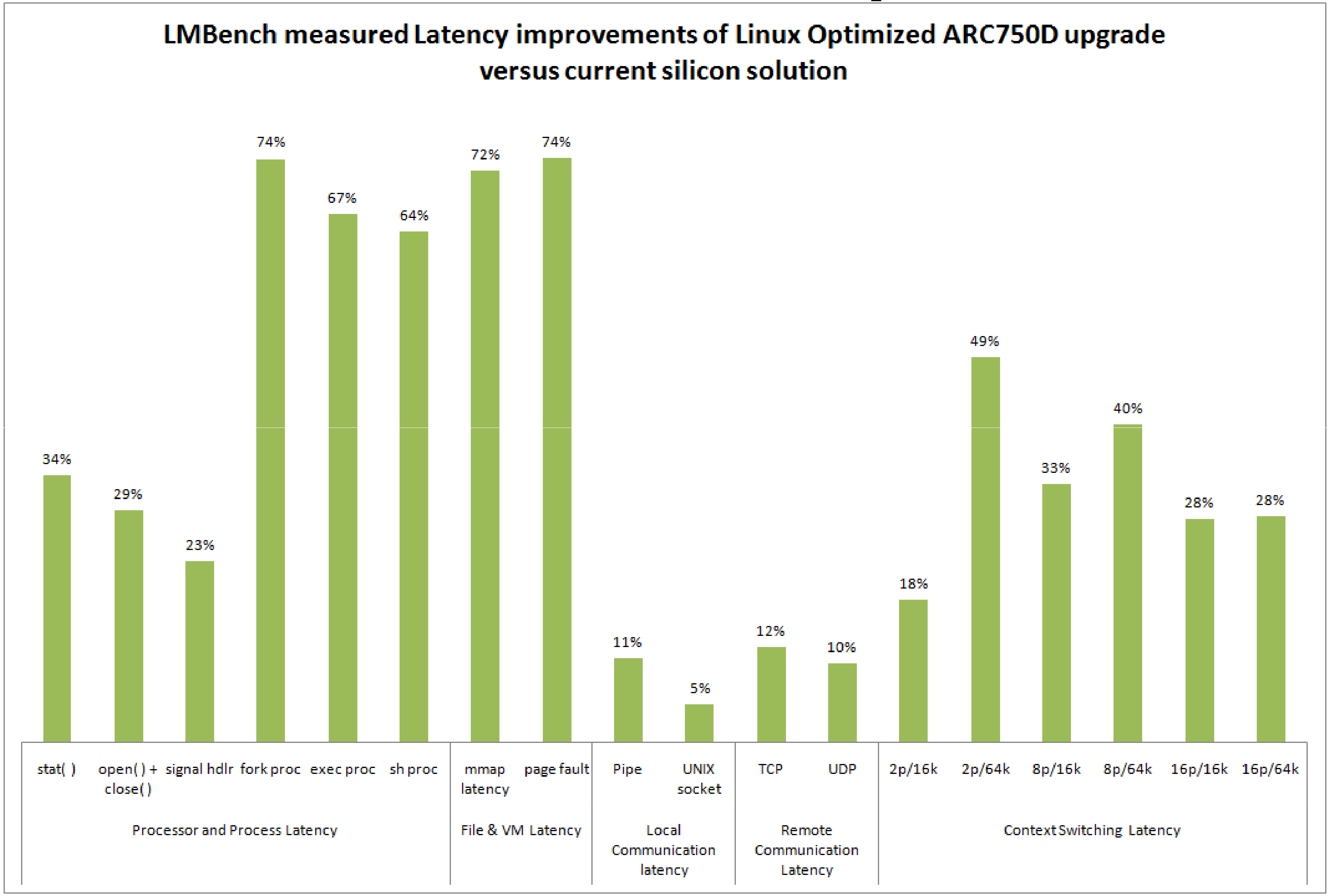
measurements are done on 50MHz FPGA
results are without performance gains from hardware extensions

Optimizing Hardware
Custom Instructions & Prefetching



SYNOPSYS®
Predictable Success

Linux kernel + ARC HW optimizations



Conclusions

- There are more markets for Android than high-end smartphone
- There are more optimizations possible than relying on Moore's law for GHz multi-cores

- Optimize performance / mW & performance / area
- Sweetspot : "heterogeneous, HW accelerated multi-core"
  - Mix of CPU, DSP, and dedicated HW
  - Highly optimized platform infrastructure SW hides heterogeneous complexities

- 'Simple' ARC processor with SW optimized Dalvik VM performs equal or better as others, thanks to careful SW optimizations, and the use of simple HW acceleration
  - Custom instructions tailored for specific tasks
  - Prefetcher iso. general purpose 2nd level cache
  - DSP more efficient in audio processing than CPU

SYNOPSYS®

Fast Forward to Predictable Success