Programming for Multicore & big.LITTLE

Ed Plowman
Director of Solutions Architecture
Media Processing Group, ARM

Multicore & big.LITTLE

The case for multiprocessing

- Platform trends
  - Clear rise in quad+ cores from mid to high-end
  - Everything's getting bigger – LTE, GPU, camera, display
  - Single thread performance improvements diminishing – focus on multi-core
  - It is not just about performance - thermally constrained use cases are now commonplace

- Software trends
  - OS vendors taking more advantage of multicore
  - Wider awareness of multiprocessing support libraries
  - Increased combined use of devices – e.g. augmented reality

Multiprocessing

Taking advantage of parallelism

In the Core

NEON/SIMD

Use of common parallelizing tools

OpenMP, Renderscript, OpenCL, etc.

Multi-threading where possible

Never easy, but increasingly necessary

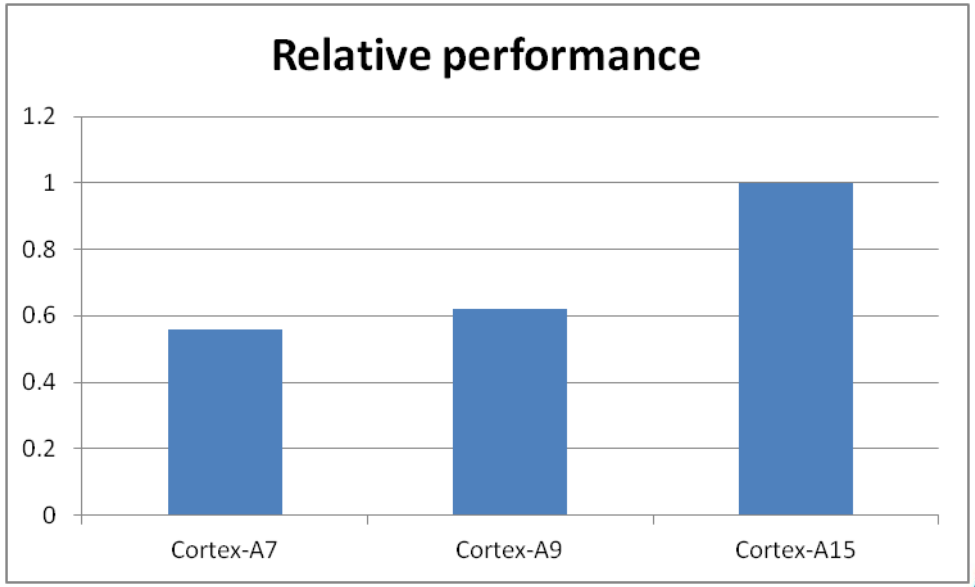
ARM

Looking Ahead – Multi-core Trends for 2014-5

- Cortex-A15/Cortex-A7 big.LITTLE is the premium product in 2014
  - Range of core count: 4 (2+2), 6 (2+4) and 8 (4+4) cores
  - Cortex-A17/Cortex-A7 (32b) coming in 2015
- ARMv8-A (64b) chipsets emerging across all segments in 2014
  - Quad and Octa-core Cortex-A53 coming into entry level and mid-range
- High-end mobile expected to move to A57 and A53 big.LITTLE for 2015
  - Multiple big.LITTLE topologies expected

MEDIATEK

- New LITTLE processors offer similar performance to Cortex-A9
- Significant performance boost with big processor e.g. Cortex-A15

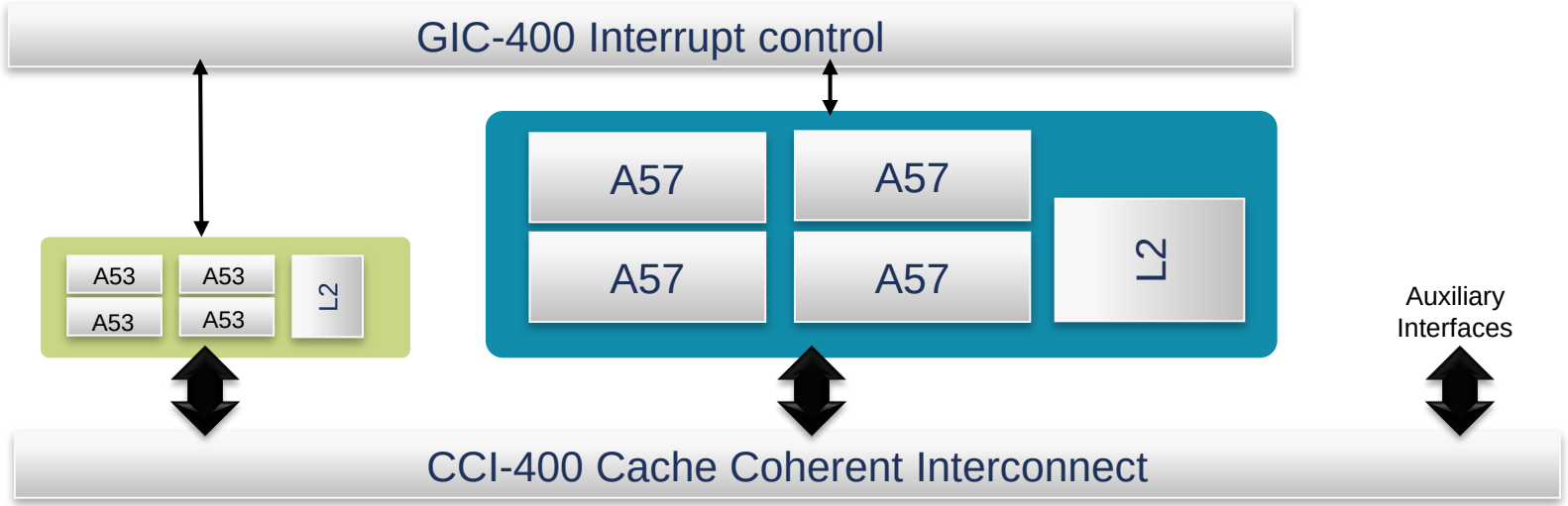


ARM

A big.LITTLE System

Programmer's view of hardware

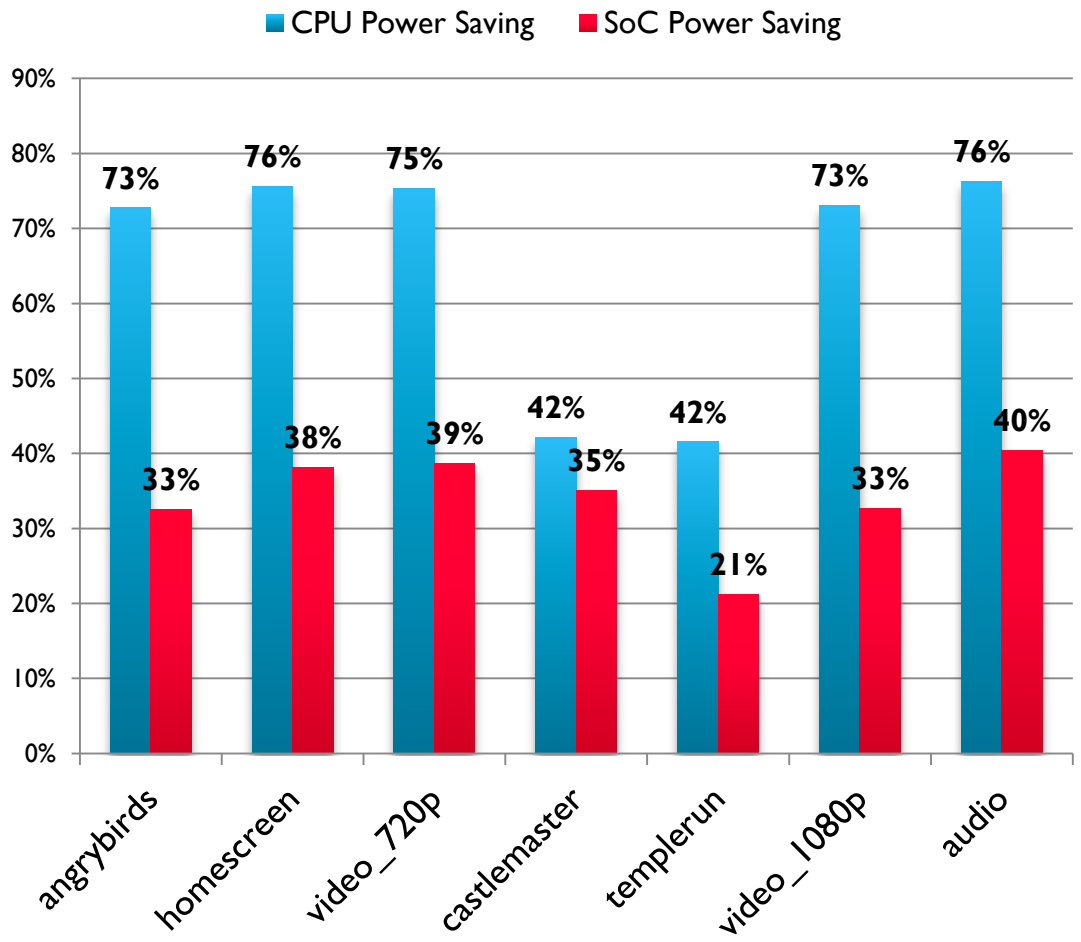
- High performance Cortex-A57 CPU cluster
- Energy efficient Cortex-A53 CPU cluster
- CCI-400 maintains cache-coherency between clusters
- GIC-400 provides transparent virtualized Interrupt control

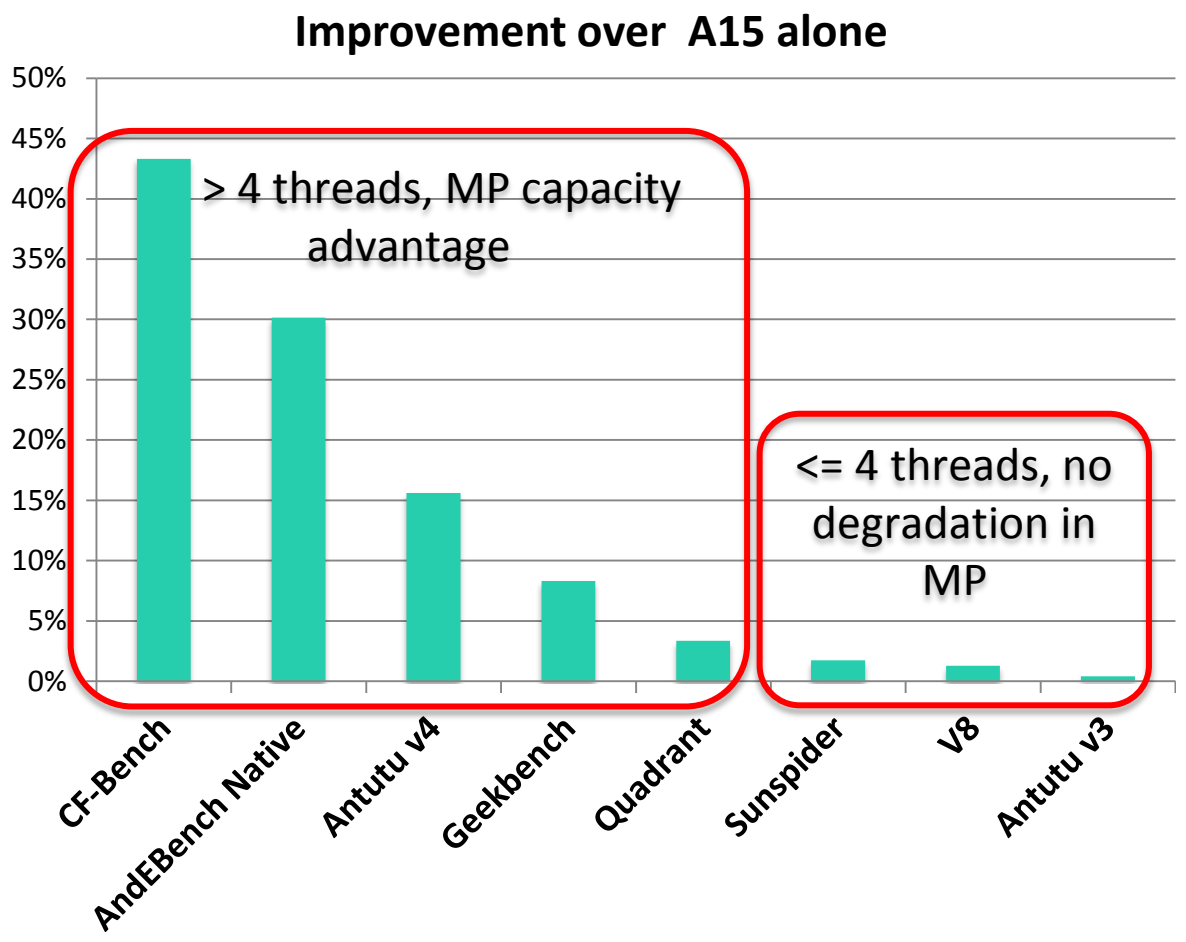


ARM

big.LITTLE

The evidence from a 4+4 MP system vs Quad Cortex-A15





ARM

big.LITTLE Development

General advice for Global Task Scheduling (GTS)

- Trust the scheduler…
  - Linux will schedule for performance and efficiency
  - All new tasks started on big to avoid latency
  - Quickly adapts to a task's needs
- …Unless
  - You know a thread is intensive but not urgent
  - Affine to LITTLE, never to big
  - e.g. Maybe use this for asset loading on a separate thread
- LITTLE cores are great
  - You'll be using them a lot
  - Cortex-A53 ~20% greater perf than Cortex-A9
  - Most workloads will run on LITTLE
  - More thermal headroom for other SoC components
- big cores are serious powerhouses
  - Think of them as short-burst accelerators – e.g. Physics based special effects
  - Think about the trade offs during design

big.LITTLE Development

Things to avoid

- Imbalanced threads sharing common data
  - Cluster coherency is excellent but not free
- If you have real-time threads note that…
  - RT threads are not auto migrated
  - RT threads are a design decision, think carefully about affinity
  - http://linux.die.net/man/2/sched_setaffinity
  - And TBB too https://www.threadingbuildingblocks.org/
- Avoid long running tasks on big cores
  - You'll rarely need that processing power for long periods
  - Can the task be parallelized?

Takeaways

- big.LITTLE & Global Task Scheduling (or HMP) in 2014 devices
  - Fantastic peak performance
  - Energy-efficient, sustainable compute for long running workloads
- Multi-processing
  - Get ahead of the limits on single thread performance
  - Avoid thermal constraints on performance

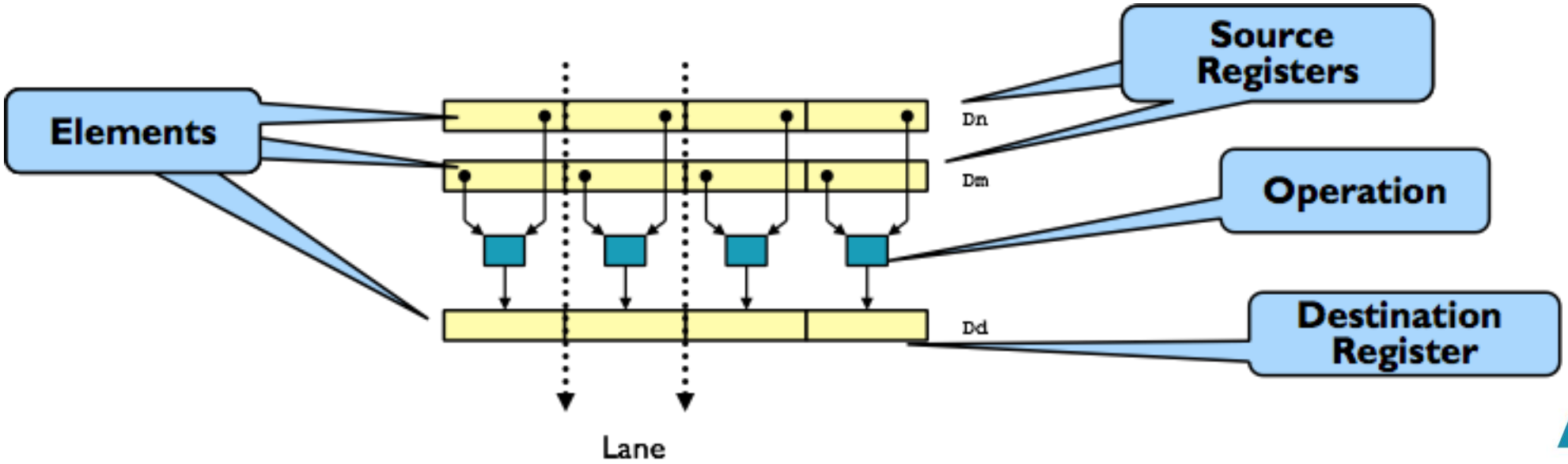
NEON and SIMD

Matt DuPuy
Staff Software Engineer, ARM

Single Instruction Multiple Data

NEON is a wide SIMD data processing architecture

- Extension of the ARM® instruction set
  - 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide in ARMv7)
- NEON Instructions perform "Packed SIMD" processing
  - Registers are considered as vectors of elements of the same data type
  - Data types: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single/double prec., floating or integer
- Instructions perform the same operation in all lanes



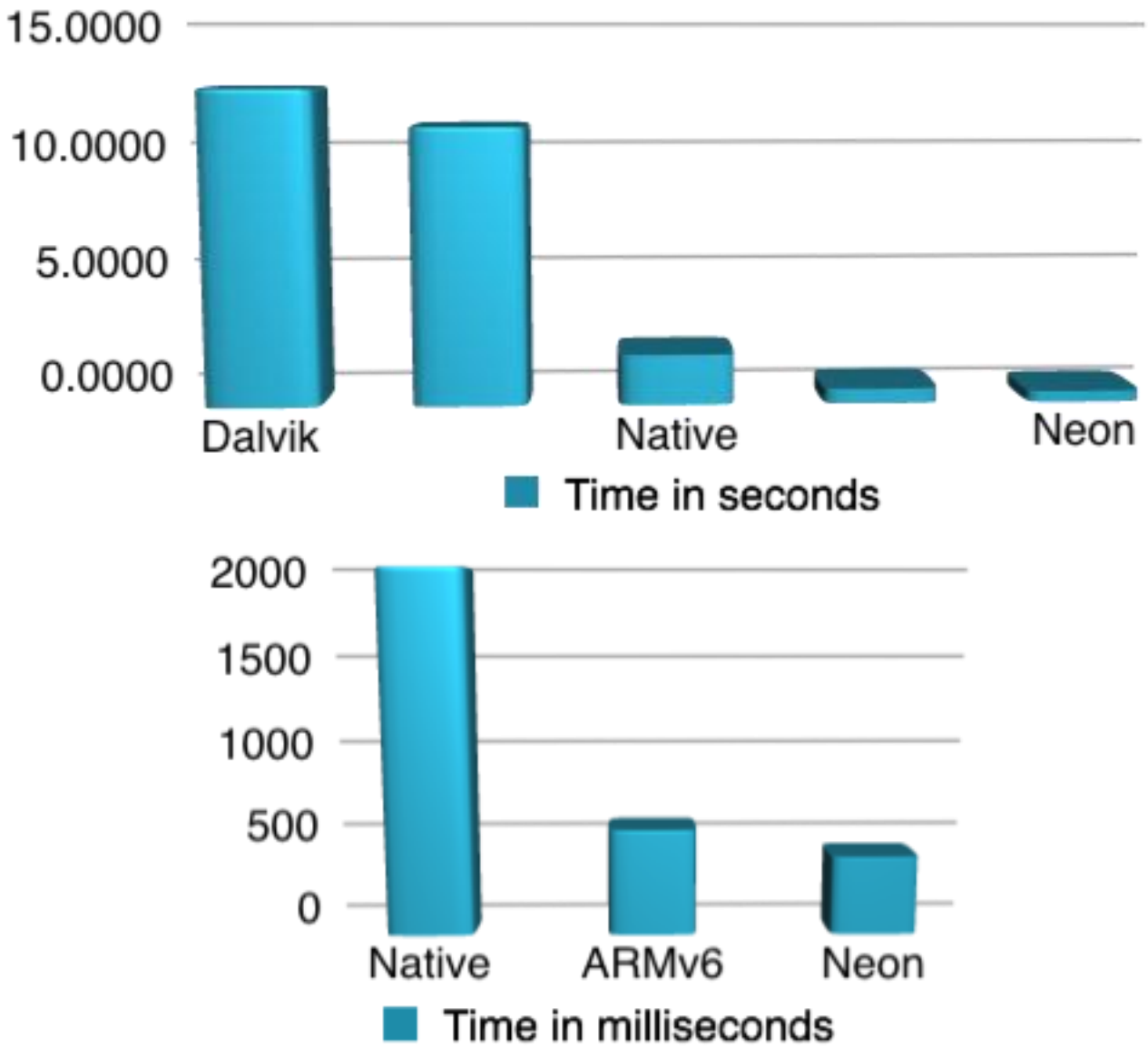
ARM

General purpose SIMD processing useful for many applications

- **Supports widest range multimedia codecs used for internet applications**
  - Many soft codec standards: MPEG-4, H.264, On2 VP6/7/8/9, Real, AVS, …
  - Supports all internet and digital home standards in software
- **Fewer cycles needed**
  - NEON will give 1.6x-2.5x performance on complex video codecs
  - Individual simple DSP algorithms can show larger performance boost (4x-8x)
  - Processor can sleep sooner => overall dynamic **power saving**
- **Straightforward to program**
  - Clean orthogonal vector architecture
  - Applicable to a wide range of data intensive computation.
  - Not just for codecs – applicable to 2D/3D graphics and other processing
  - Off-the-shelf Tools, OS, commercial & open source ecosystem support

ARM

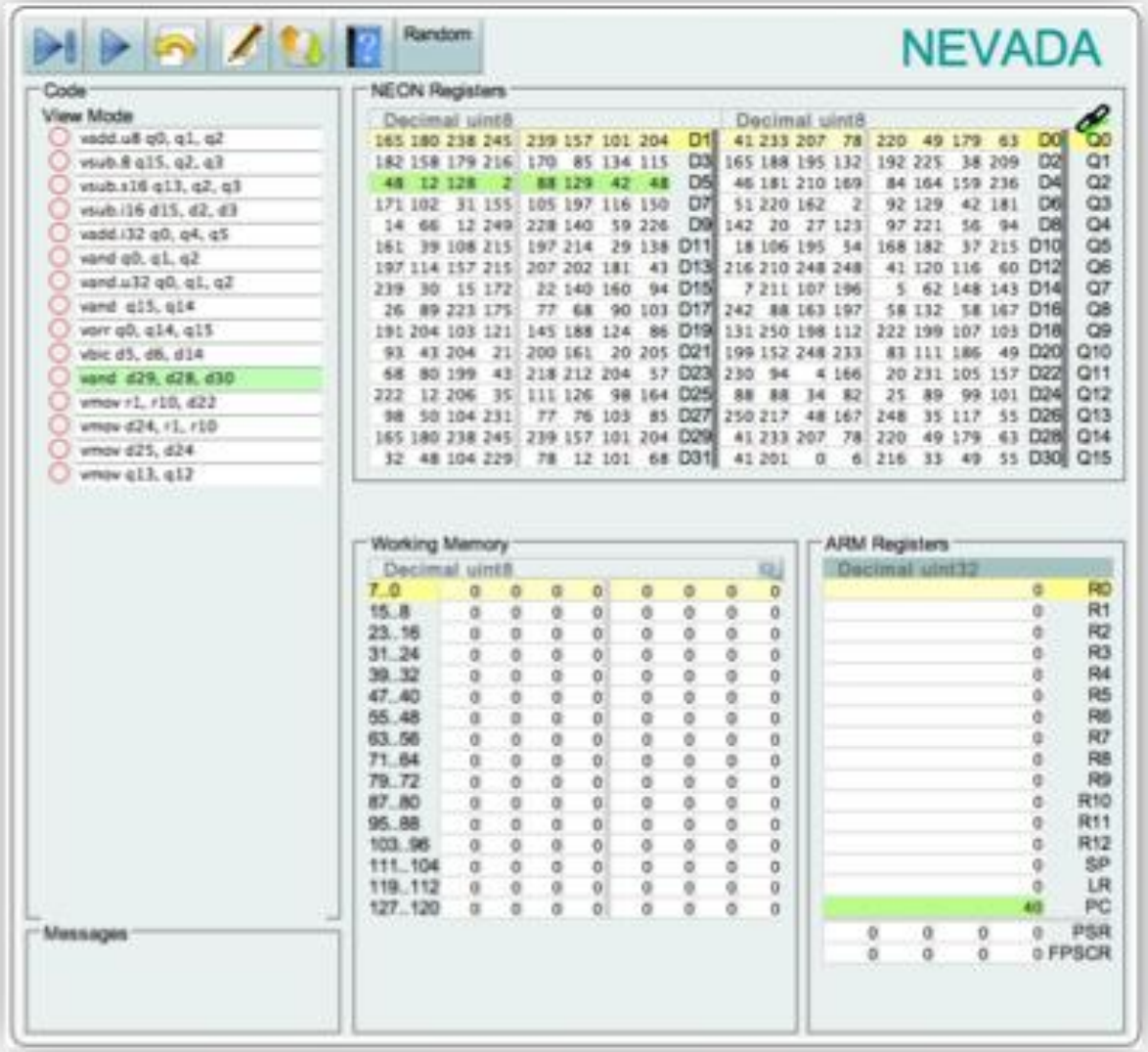
Specific media intensive test case using Android NDK



ARM

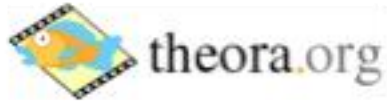
NEON Visualizer

http://szeged.github.com/nevada/



Don’t Reinvent the wheel! NEON in Open Source Today

- **Google WebM** – 11,000 lines NEON assembler!
- **Bluez** – official Linux Bluetooth protocol stack
- **Pixman** (part of cairo 2D graphics library)
- **ffmpeg (libav) – libavcodec**
- LGPL media player used in many Linux distros and products
- Extensive NEON optimizations
- **x264** – Google Summer Of Code 2009
- GPL H.264 encoder – e.g. for video conferencing
- **Android** – NEON optimizations
- **Skia** library, S32A_D565_Opaque **5x** faster using NEON
- Available in Google Skia tree from 03-Aug-2009
- **LLVM** – code generation backend used by Android RenderScript
- **Eigen2** – C++ vector math / linear algebra template library
- **TheorARM** – libtheora NEON version (optimized by Google)
- **libjpeg / libjpeg-turbo** – optimized JPEG decode
- **libpng** – optimized PNG decode
- **FFTW** – NEON enabled FFT library
- **Liboil / liborc** – runtime compiler for SIMD processing
- **webkit/blink** – used by Chrome Browser



ARM

Optimization Paths for Neon

- **Opensource libraries, e.g. OpenMAX, libav, libjpeg, Android Skia, etc.**
  - **Freely available Open Source optimizations**
- **Vectorizing Compilers**
  - Exploits NEON SIMD automatically with existing source code
  - **Status:** Released (in DS-5 armcc, CodeSourcery, Linaro gcc and now LLVM)
- **NEON C Instrinsics**
  - C function call interface to NEON operations
  - Supports all data types and operations supported by NEON
  - **Status:** Released (in DS-5 and gcc), LLVM/Clang under development
- **Assembler**
  - For those who really want to optimize at the lowest level
  - **Status:** Released (in DS-5 and gcc/gas)
- **Commercial vendors**
  - Optimized and supported off-the-shelf packages

Introducing NE10

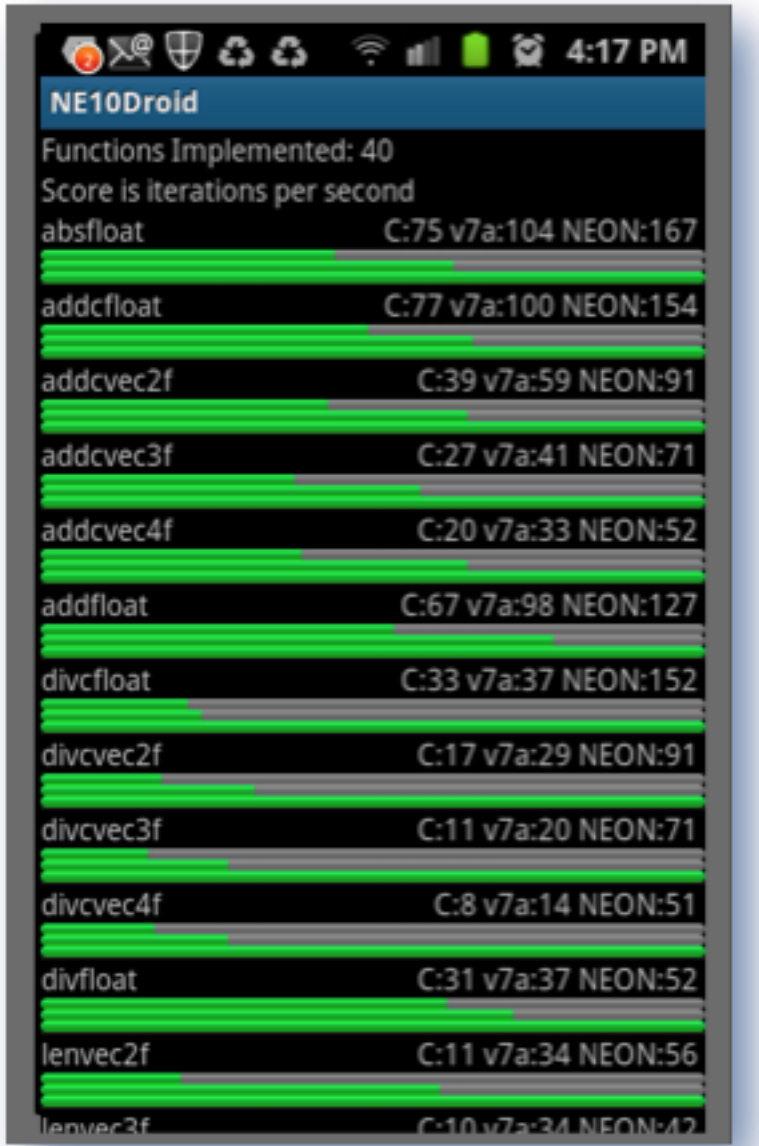
- **Ne10 is designed to provide a set of common, useful functions which**
  - have been optimised for ARMv7 and NEON, many v8 functions available in intrinsic C
  - provide consistent well tested behaviour
  - and that can be easily incorporated into applications
  - Is targeted at Android and Linux to maximize app performance and tested under iOS
- **Features**
  - Usable from C/C++ and Java/JNI
  - The library is modular; functionality that is *not* required within an App can be discarded
  - Functions similar to the Accelerate Framework provided by iOS
- **It is Free**
  - No commercial complications- 'build and ship' BSD License
  - well-tested behavior with example code
- **Use of the Ne10 library should be a joy, not a chore**
  - Out-of-box and user experience is critical to success
  - Build and go, accessible documentation, clear code
  - Supported by ARM, community contributions welcome

ARM

Ne10Droid – The App in action

- NE10Droid is a benchmarking Android App that uses NE10.
- Routines are written using VFP in C, VFP in Assembly and NEON.

```
Example routines:
arm_result_t normalize_vec2f(arm_vec2f_t *
dst, arm_vec2f_t * src, unsigned int
count);
arm_result_t normalize_vec3f(arm_vec3f_t *
dst, arm_vec3f_t * src, unsigned int
count);
arm_result_t normalize_vec4f(arm_vec4f_t *
dst, arm_vec4f_t * src, unsigned int
count);
```



ARM

64-bit Is the New Black

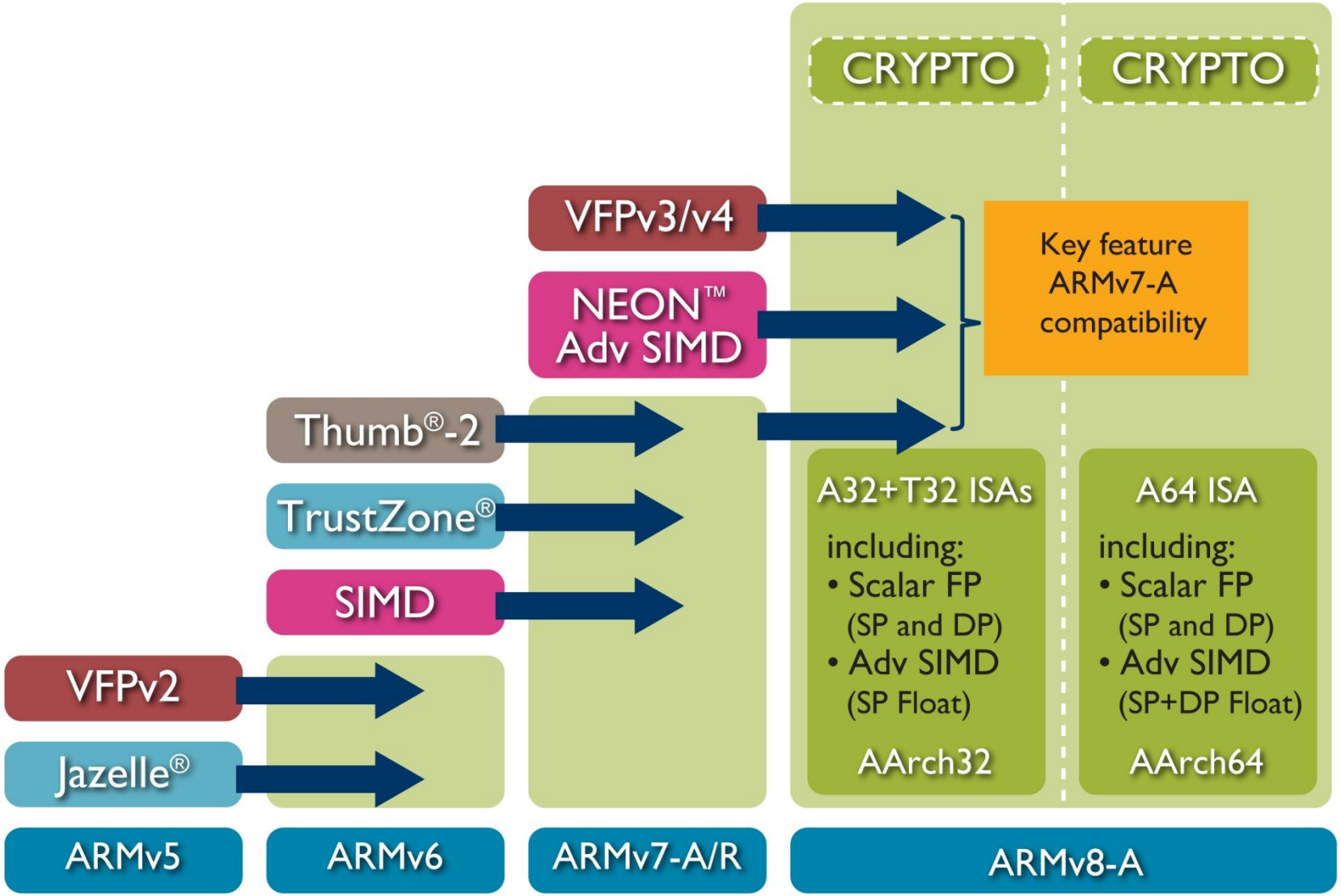
Jesse Barker
Principal Software Engineer, ARM

A Little Taxonomy

- ARMv{Version/Extension/Class} – Generic Architecture Name
  - ARMv8-A – ARM architecture version 8, application class
- AArch64 – 64-bit execution state
  - A64 – ARM instruction set
  - LP64 – 64-bit data model
  - ILP32 – 32-bit data model
- AArch32 – 32-bit execution state
  - A32 – ARM instruction set
  - T32 – Thumb instruction set
  - ILP32 – 32-bit data model
- Interprocessing – Interaction of execution environments

A Little Perspective

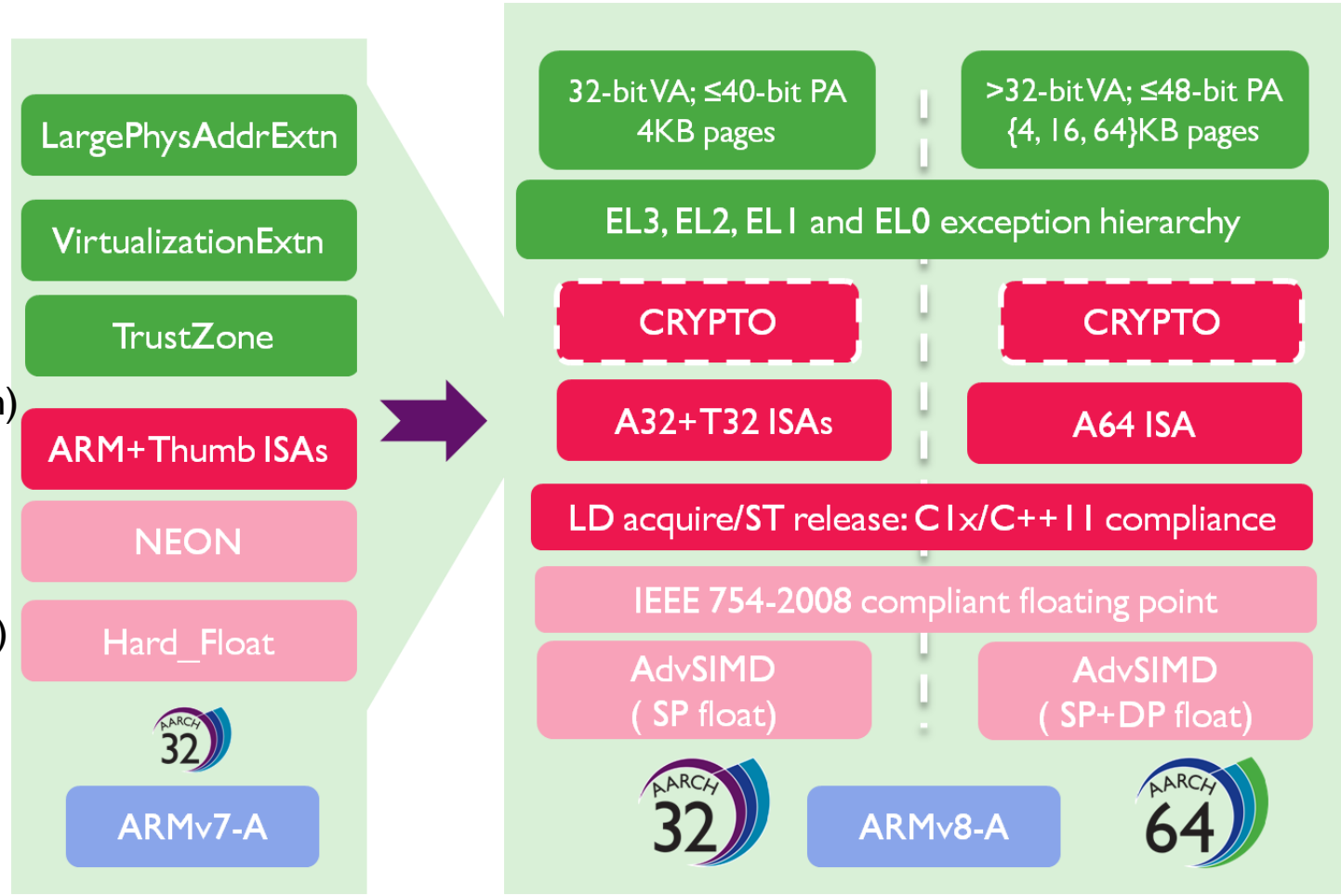
Use this one or previous one?



ARM

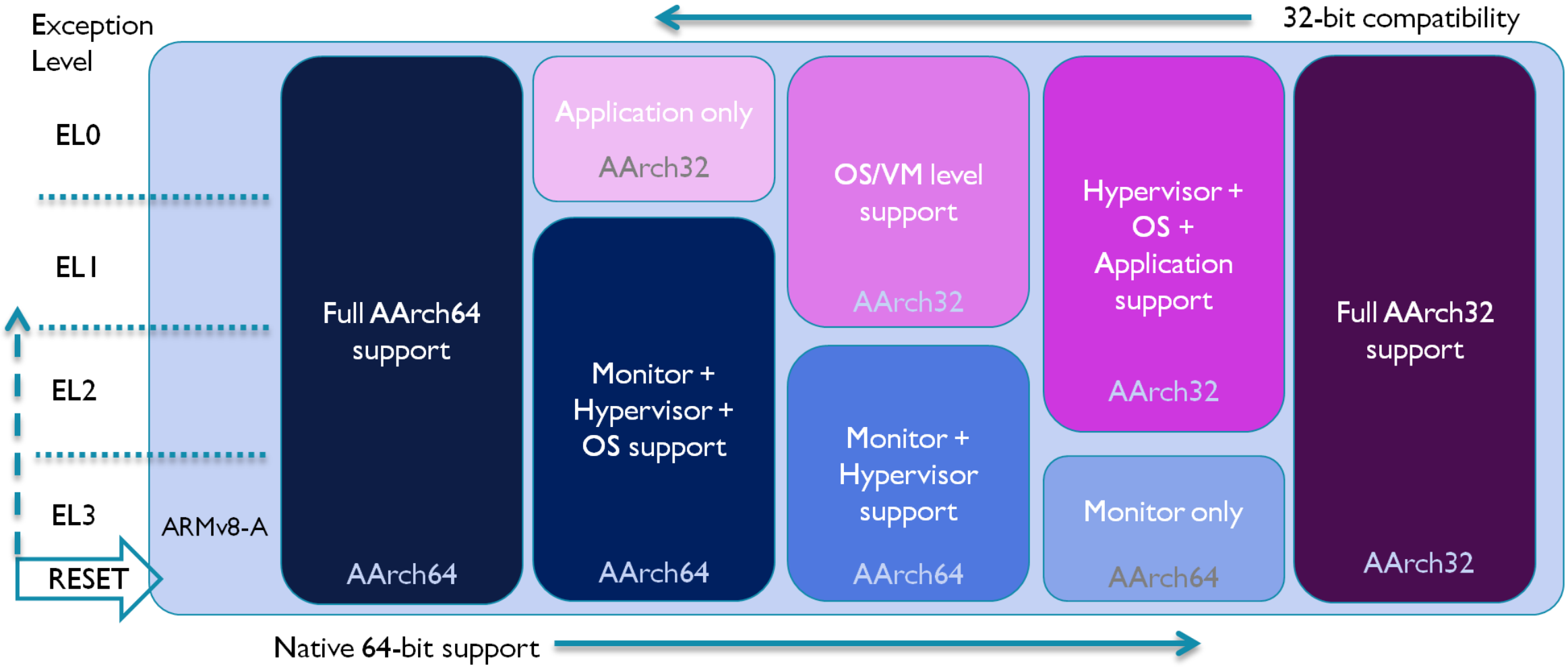
A Closer Look

- ARMv8-A is one of the most significant architecture changes in ARM's history
- AArch64 can access ALL ARMv8-A features
  - Larger address space (>4GB memory for Application)
  - Wider data register (64-bit)
  - Better SIMD (NEON)
  - New Crypto Instructions
  - More data registers (31 general, 32 FP/SIMD/Crypto)
  - More…
- In the long term, delivers an unified architecture across Mobile, Client (Table/Desktop) and Enterprise markets



ARM

Exception Levels & Interprocessing



ARM

So, You Have a 32-bit Application…

- Moving to ARMv8-A
  - No swap instruction, CP15 barriers, load/store multiple
  - Load/store pair
- Moving to A64
  - Mnemonically similar to A32
  - More and larger registers
  - No instruction predication
  - Conditional select
  - Dedicated return instructions
- Why move?
  - Significant performance gains come with ARMv8-A

But You Do Not Use Assembly Language

- Beware of
  - Object sizes in LP64
  - Casting between pointer and non-pointer types
  - Implicit type/size conversions
  - Bit-wise manipulations
  - Magic numbers
- Multiarch can be your friend
- Trust your compiler!

Toolchains

- ARM is actively involved in two major Open Source Compilers
  - LLVM
    - AArch64 supported upstream as of LLVM 3.3
    - Ongoing work to address outstanding defects
    - OpenCL™ support
    - Buildbots available http://lab.llvm.org:8011/builders/
  - GCC
    - AArch64 supported upstream as of GCC 4.8
    - Support for dynamic linking, TLS, cross-compiler and glibc
    - Support for C/C++ ABI and PCS
    - ARM® NEON™ auto-vectorization and intrinsics

ARM

Questions?

ARM

Thank You!

ARM