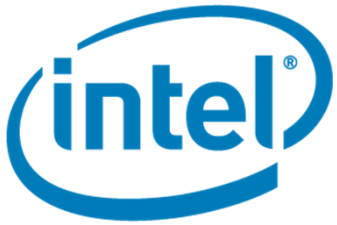


Efficient (Parallel) System Programming for Heterogeneous SoC Platforms

Wei Liu

Intel Corporation

Feb 24th, 2013

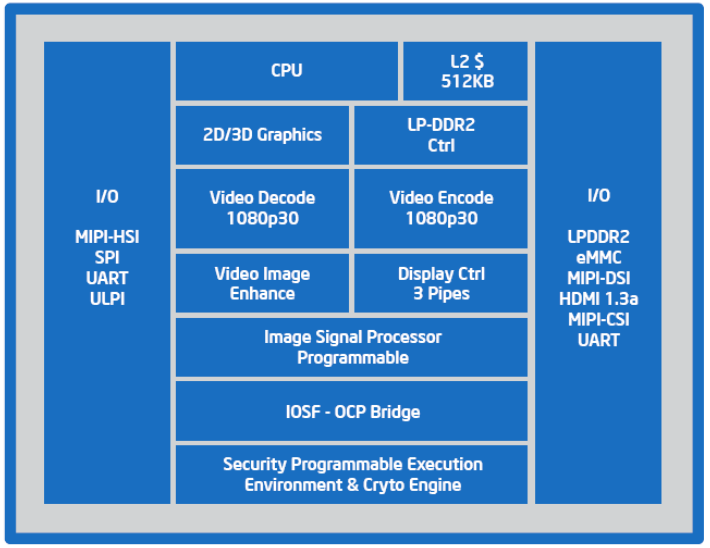
Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL®PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL®PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user
- Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.
- Intel, Intel Inside and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- Other names and brands may be claimed as the property of others.
-

Heterogeneous System-on-Chip

- Heterogeneous platform consists of many IP blocks
  - Application processor/CPU
  - ISP, DSP, GPU, Video processors …

- Specialized accelerators are efficient for *domain-specific* computations
  - \+ Higher performance
  - \+ Less power
  - \+ Smaller size
  - – Limited programmability



The Intel®Atom™ processor Z2460 is a highly integrated system-on-a-chip (SoC), delivering powerful graphics and video capabilities needed for smartphones.

- SoC is becoming mainstream computing

Impose grand SW challenges

Software Key to SoC Success

- Software delivers experiences to end users
  - SoC vendors provide complete SW stack with SoC
  - SW must ship with the SoC silicon

- SoC software is very *complex*
  - SW harnesses the complex SoC
  - Across multiple SW layers (FW, driver, framework, apps)
  - Across multiple segments/platforms/OSes
  - Millions lines of code today and keep increasing…

Parallelism becomes the DEFAULT

Key Challenges for SoC Software

- To-to-market (TTM)
  - Critical to be competitive in mobile market
  - Reduce the traditional HW/SW development cycle
- Multiple products support
  - Different market segments
    - smart phone, tablet, smart TV, in-vehicle infotainment etc
  - Different OSes
    - Windows, Android, Linux, RTOS etc.
- High SW Quality
  - Maintain SW quality under tight TTM
- Innovations: differentiate the product!

Key Challenges for Programmers

- Need efficient way to develop new features
  - Copy-paste? ➔ Use existing library/module
  - Avoid parallel programming or parallelism awareness?
    ➔ No way!

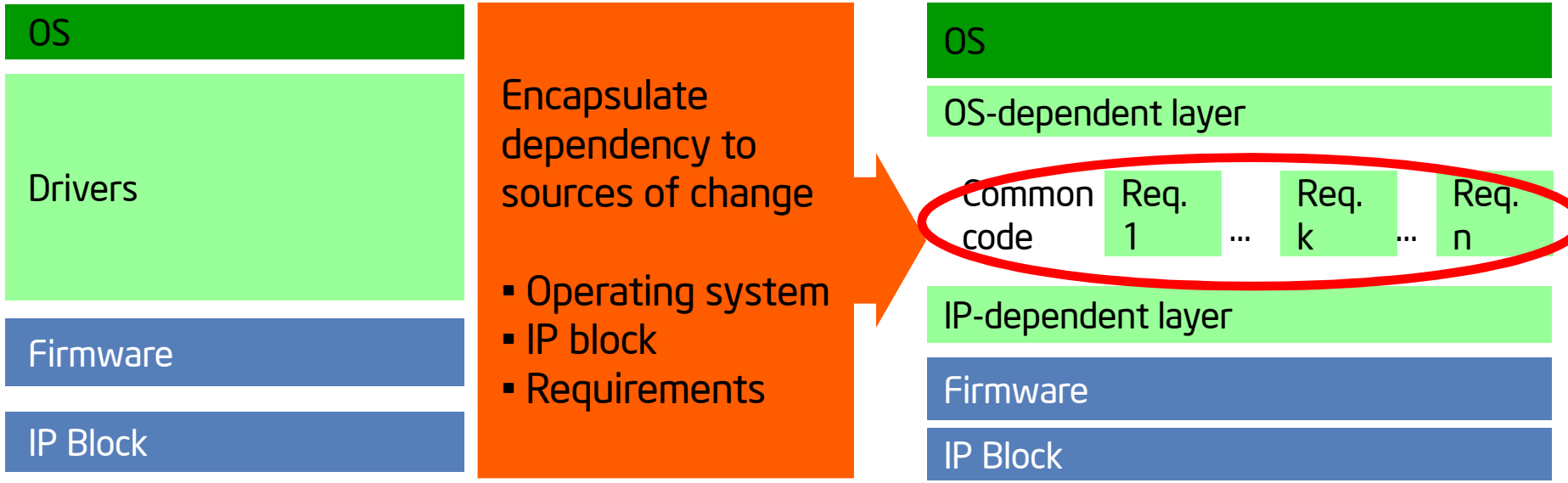
- Need efficient tools for programming
  - Visibilities to black boxes
  - Visibilities to multiple components
  - Bug reproducibility
  - etc.

Efficient SW development: methodology, programming and debugging

Outline

- Motivation
- Modular Software Architecture
- System Software Programming
- Conclusions

Modular Software Architecture



Monolithic drivers

Work well with one or two products but not scalable

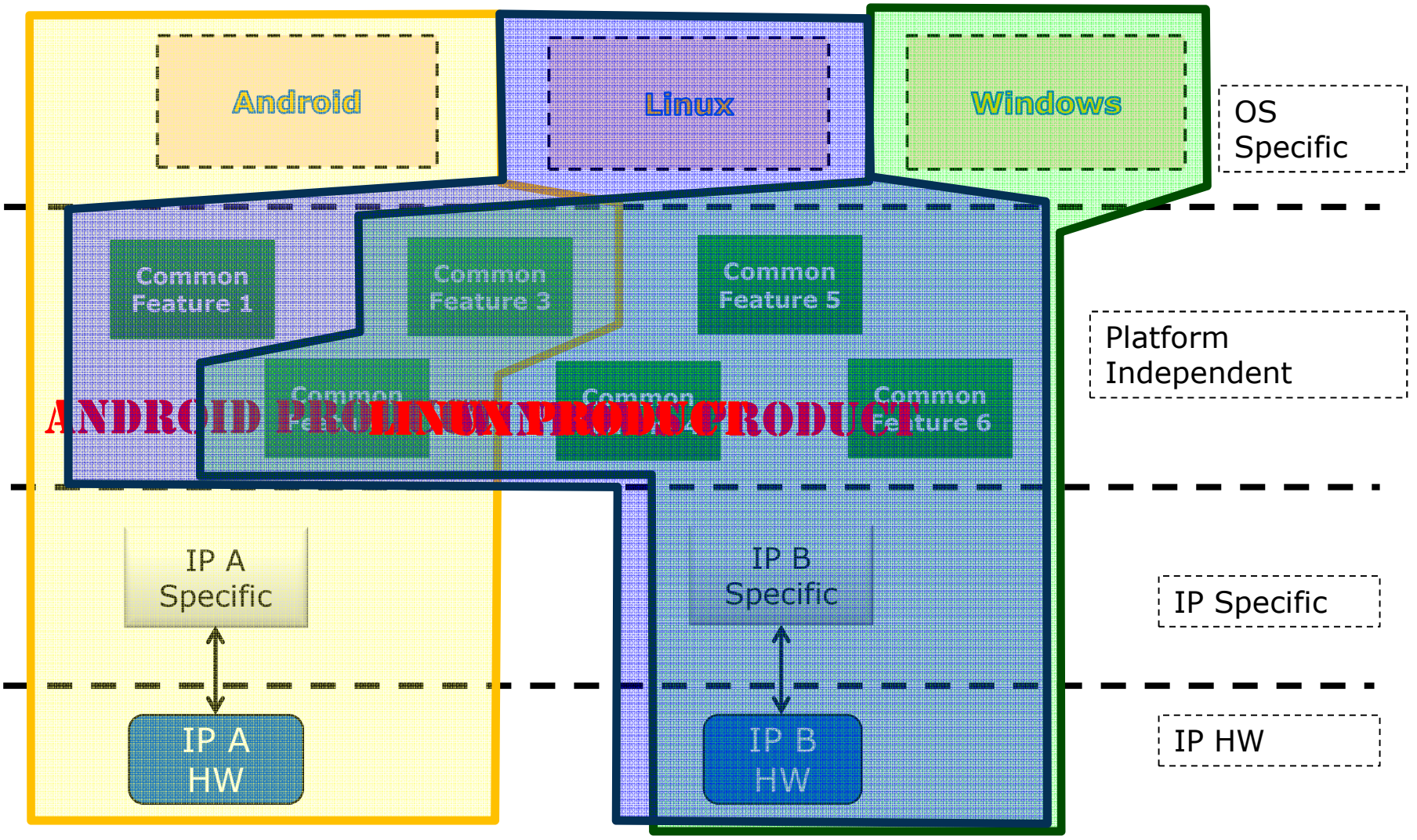
Layered drivers

OS and IP block dependencies encapsulated
- Ease OS and platform customization

Shared code modularized to atomic requirements
- Ease feature customization

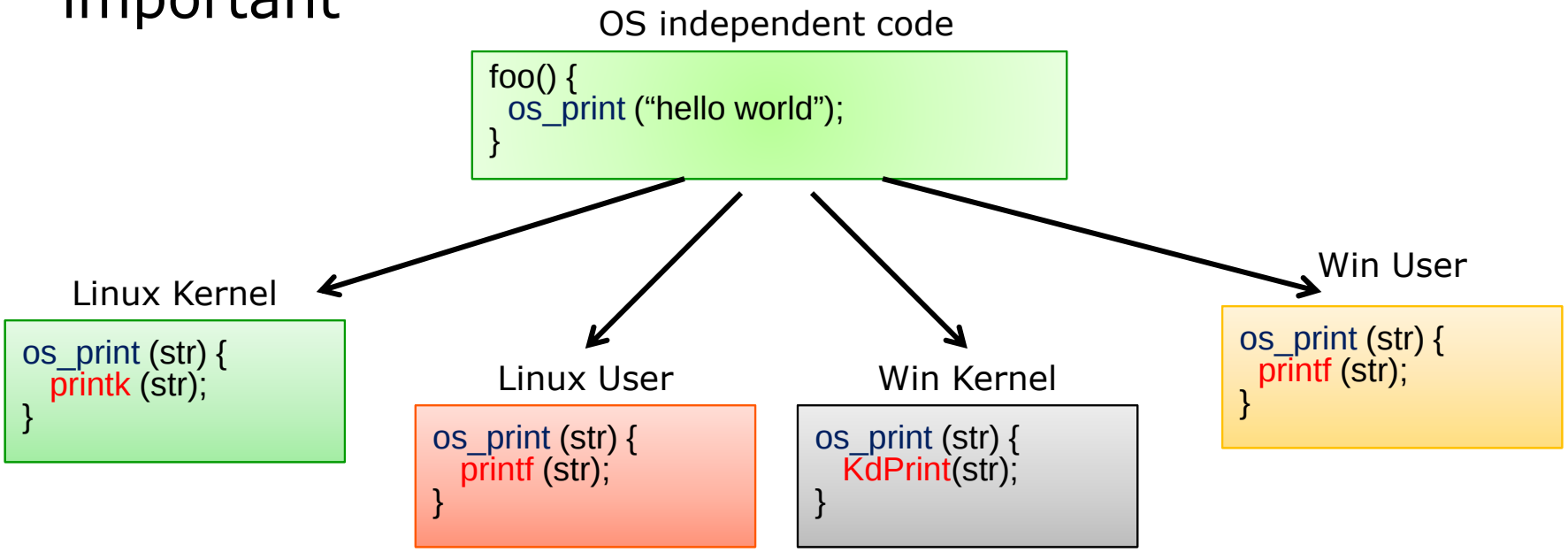
Enable Reuse across OSes and products

Product Customization



Cross OS Reuse

- Pervasive OS services make OS abstraction very important



- OS abstraction must support:
  - Interaction with upstream Linux kernel
  - Cross-OS reuse at the user level and kernel level
  - Kernel-user reuse

Outline

- Motivation
- Modular Software Architecture
- System Software Programming
- Conclusions

Why is SoC System Programming Different?

- User experience are the driving force for mobile
  - Not only the performance
- SW and HW are optimized for the whole system
  - Not only the individual components
- SW is integrated by SoC vendors as reference
  - Not only the isolated drivers

Need System View for SW Development

Where is the Difficulty From?

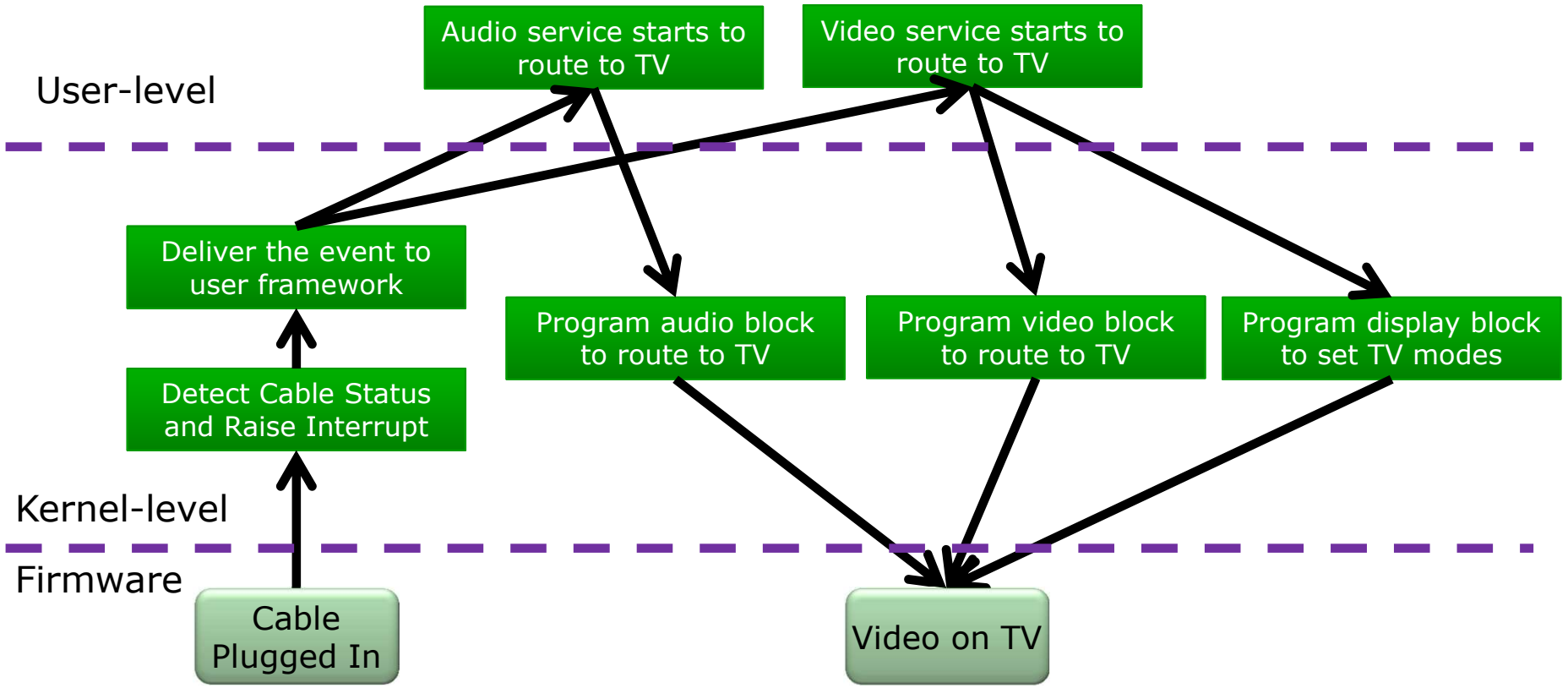
- "*Difficult*" system bugs are the long-pole of products
  - Interactions between layers (framework, driver, FW)
  - Interactions between components (audio, video, display etc)
  - 80/20 rule applies

- Why? (not complete, but some insights)
  - Difficult to track between layers
  - Often different ISA between IP blocks
  - Not well thought/designed interaction between components
  - Most of them are non-deterministic bug and hard to reproduce
    - Thanks to parallelism. ☺

Case Study: Connect Smartphone to TV

- Playback your favorite video to TV
- User Expectation
  - HDMI cable connected:
    - Audio/Video automatically goes to TV
  - HDMI cable disconnected:
    - Audio/Video automatically comes back to smartphone

Behind the Scene



- Multiple flows going on at the same time
- More system interactions can happen unexpectedly
  - Cable disconnected
  - Other audio streams
  - etc

What Do We Need in System Programming Model?

- Explicit Concurrency Awareness
  - Model can be used to check any violations
  - Force programmers to think in term of possible races
- Tolerate unexpected interrupts
  - e.g. cable disconnected, phone call comes in
- Synchronization support across user and kernel
- Enable debugging tools
  - Trace across components and layers
  - Reproducibility

Summary

- SoC, as a heterogeneous system, poses new challenges to computing
- **Software** is the key to success of system-on-chip
- Modular SW architecture and system-centric programming models are desirable
- Open challenges for the community
  - New SW methodology required for SoC software development
  - Programming, debugging, verification
  - System optimizations to leverage different IPs

Questions?