Apps with Hardware

Enabling Run-time Architectural Customization in Smart Phones

Michael Coughlin, Ali Ismail, Eric Keller
University of Colorado Boulder

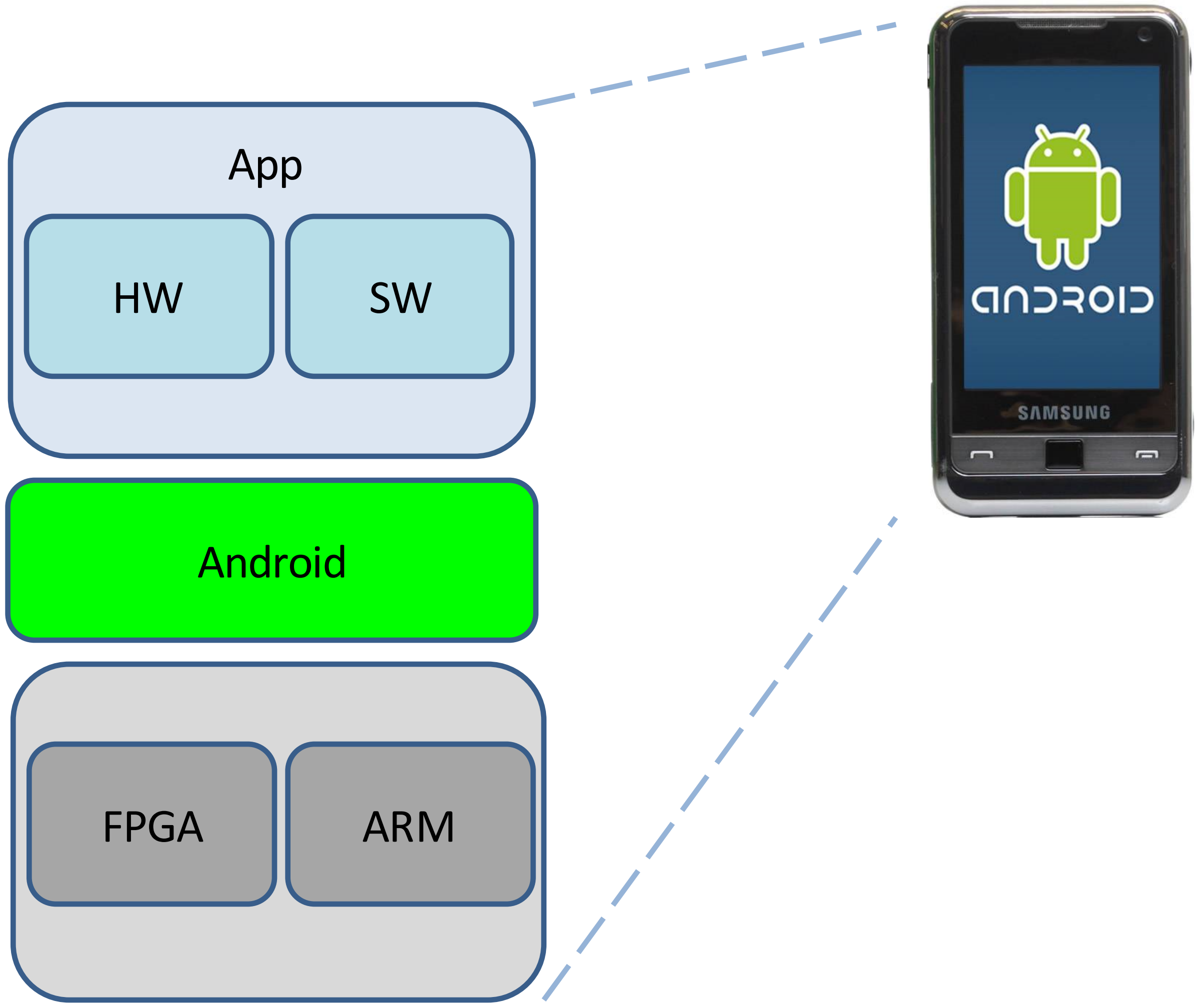
Mobile Devices

Devices are designed around certain restrictions

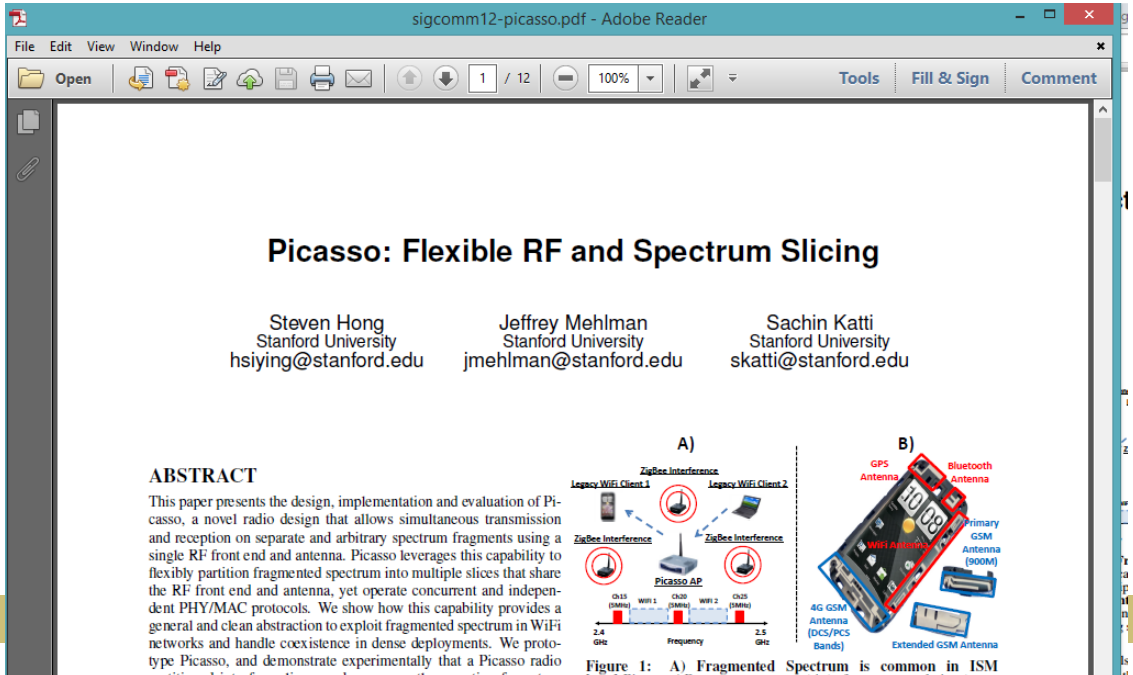
This leads vendors to make tradeoffs

What if users and developers could choose?

Vision: Smart Phone with an FPGA



Software-defined Radio



Picasso: Flexible RF and Spectrum Slicing

Steven Hong
Stanford University
hsiying@stanford.edu

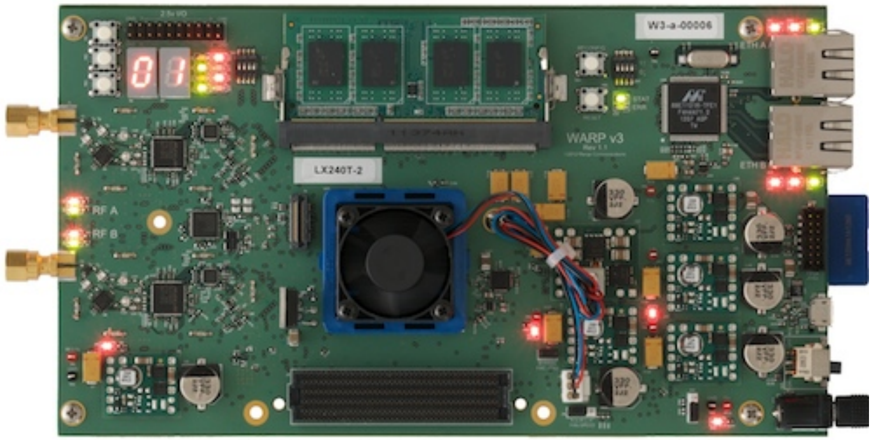
Jeffrey Mehlman
Stanford University
jmehlman@stanford.edu

Sachin Katti
Stanford University
skatti@stanford.edu

ABSTRACT

This paper presents the design, implementation and evaluation of Picasso, a novel radio design that allows simultaneous transmission and reception on separate and arbitrary spectrum fragments using a single RF front end and antenna. Picasso leverages this capability to flexibly partition fragmented spectrum into multiple slices that share the RF front end and antenna, yet operate concurrent and independent PHY/MAC protocols. We show how this capability provides a general and clean abstraction to exploit fragmented spectrum in WiFi networks and handle coexistence in dense deployments. We prototype Picasso, and demonstrate experimentally that a Picasso radio

Figure 1: A) Fragmented Spectrum is common in ISM



High-performance Computing

Cryptography

Technology	Throughput (GBytes/Sec)
E5503 Xeon Processor	0.01 (Single core)
AMD Radeon HD 7970	0.33
PCIe385 FPGA Accelerator	5.20

http://www.nallatech.com/40gbit-aes-encryption-using-opencl-and-fpgas/

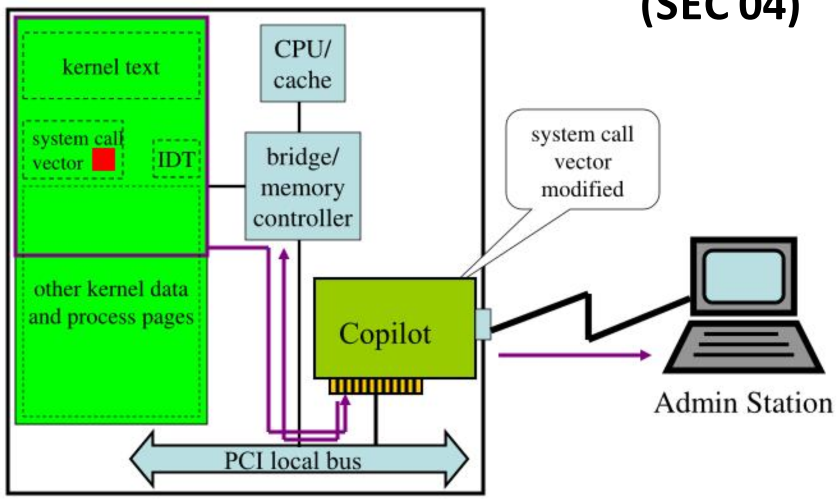
Analytics

Ryft ONE Primitive	Analytics throughput of a single, fully-populated 1U Ryft ONE device	Equivalent Spark cluster size (to match Ryft ONE performance)
Search	~10GB/sec	> 100 nodes[1]
Fuzzy Search	~10GB/sec	100-200 nodes[2]
Term Frequency	~2.5GB/sec	100 nodes[1]

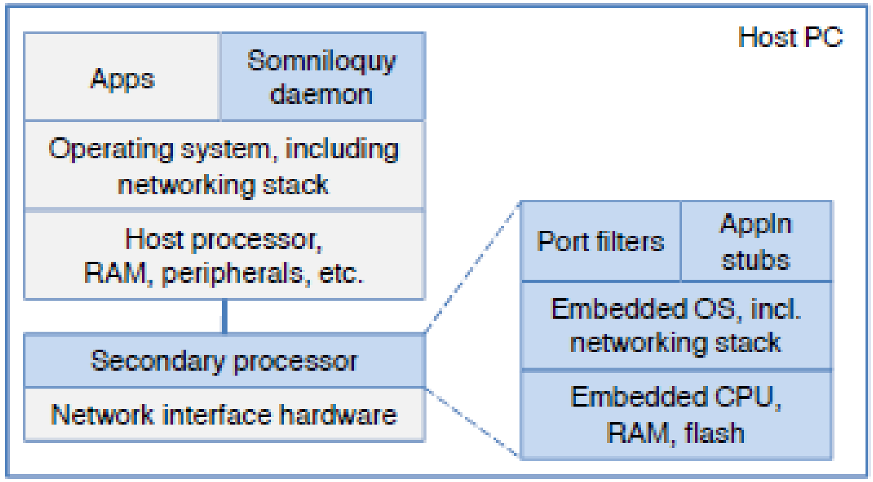
http://www.datanami.com/2015/03/10/fpga-system-smokes-spark-on-streaming-analytics/

Architectural Enhancements

Copilot Integrity Protection (SEC 04)

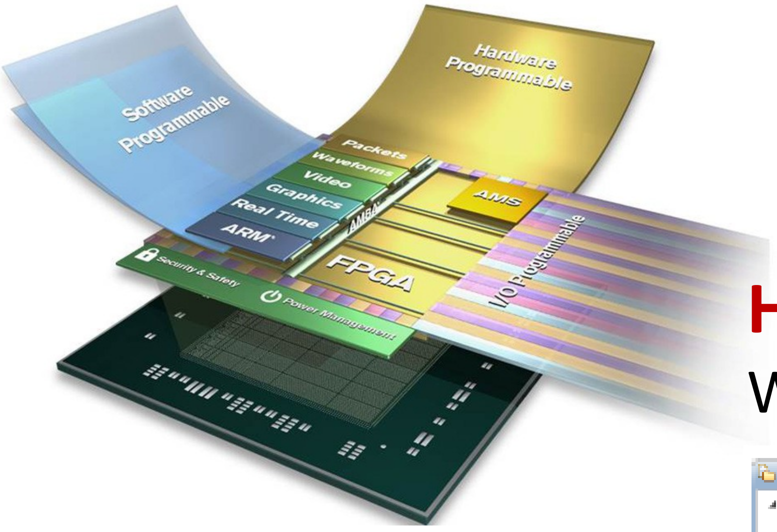


Somniloquy (NSDI 09)

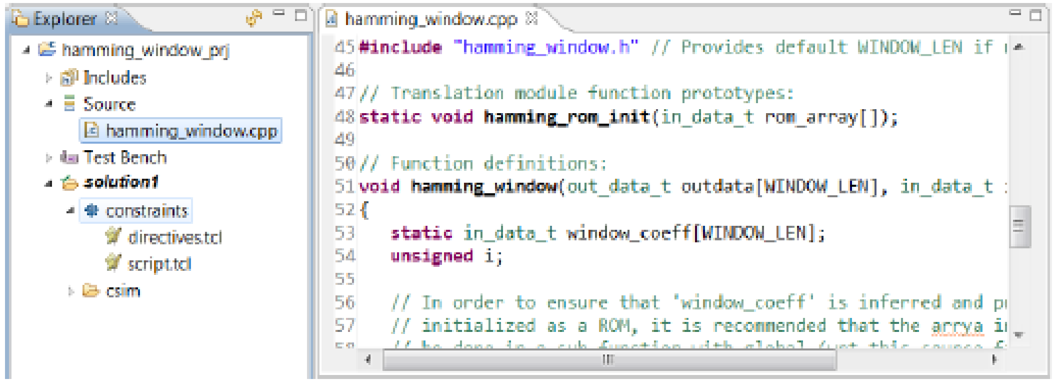


Why is now the right time?

SoCs with Programmable Logic coupled with
ARM Cortex A9 (same as iPhone 4 and many other smartphones)



High-level Synthesis
Write C / C++ / SystemC / OpenCL code

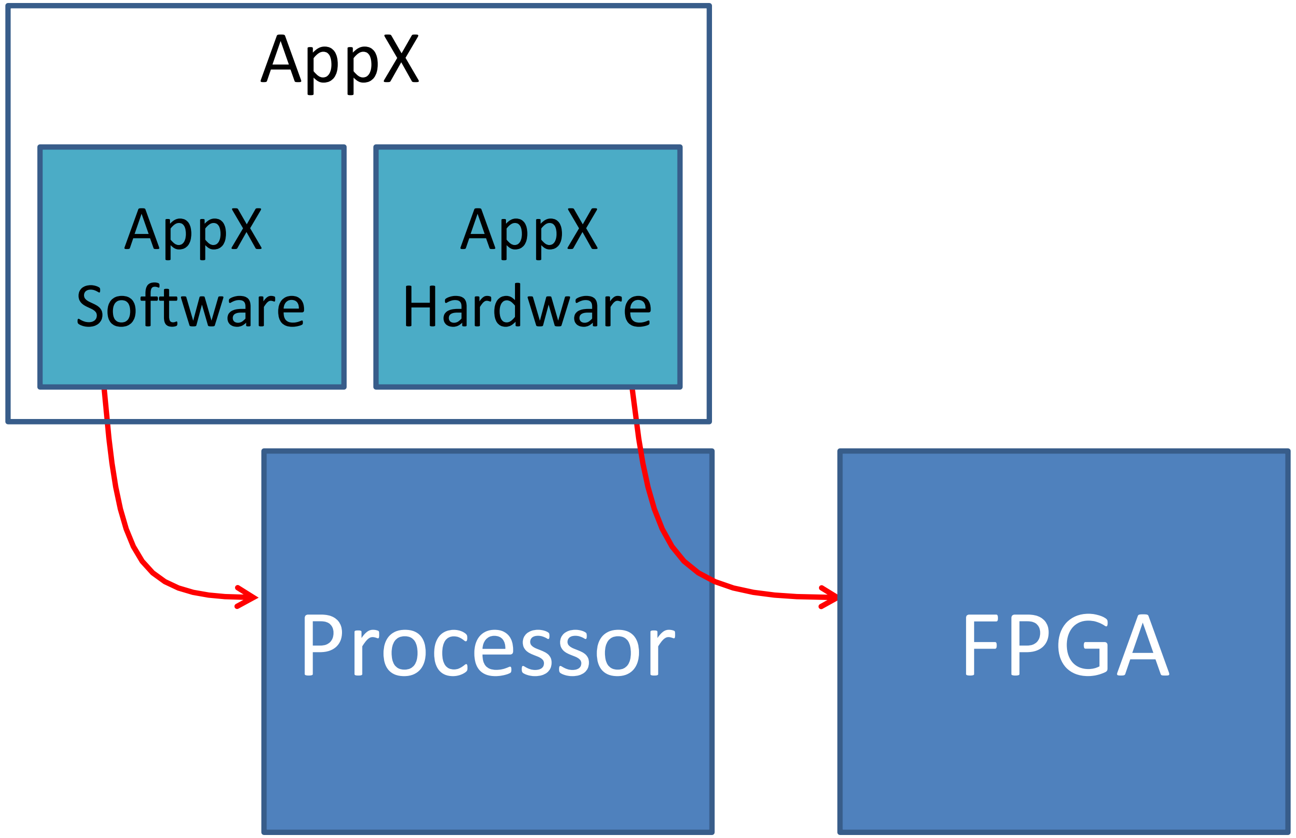


Fundamental Problem:

Sharing the FPGA between applications

What we can already do

App loads: software runs on processor, FPGA configured with hardware

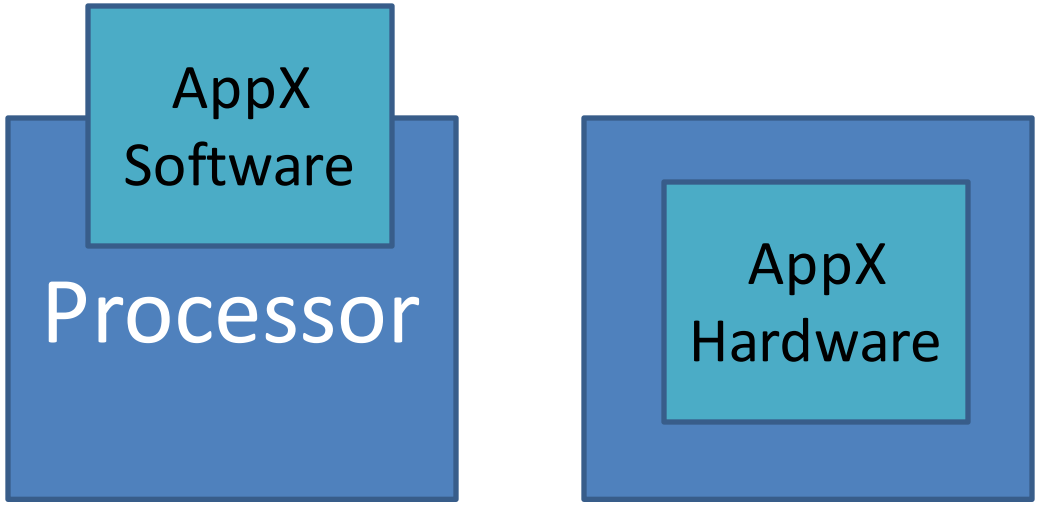


What we can already do

App loads: software runs on processor, FPGA configured with hardware

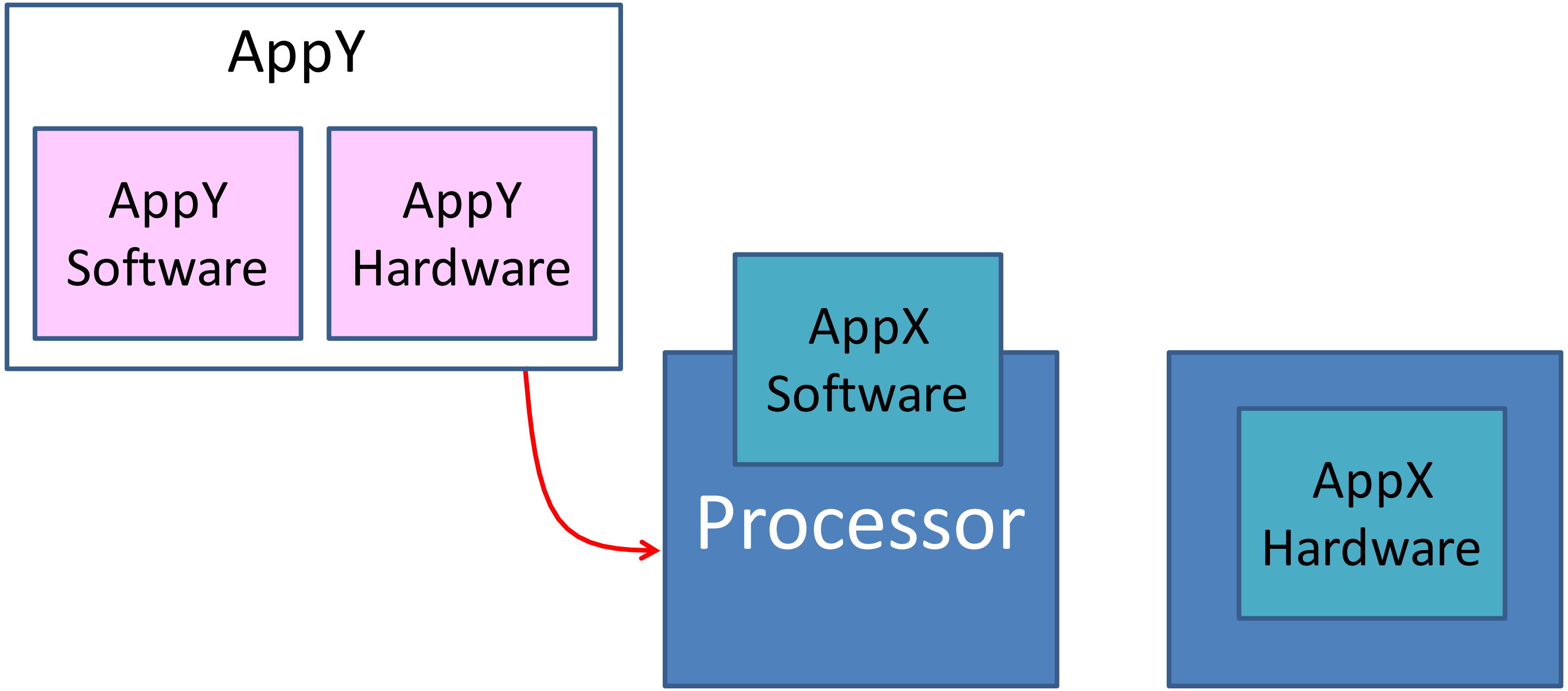
This is currently possible – ***run-time reconfiguration***

Sort of



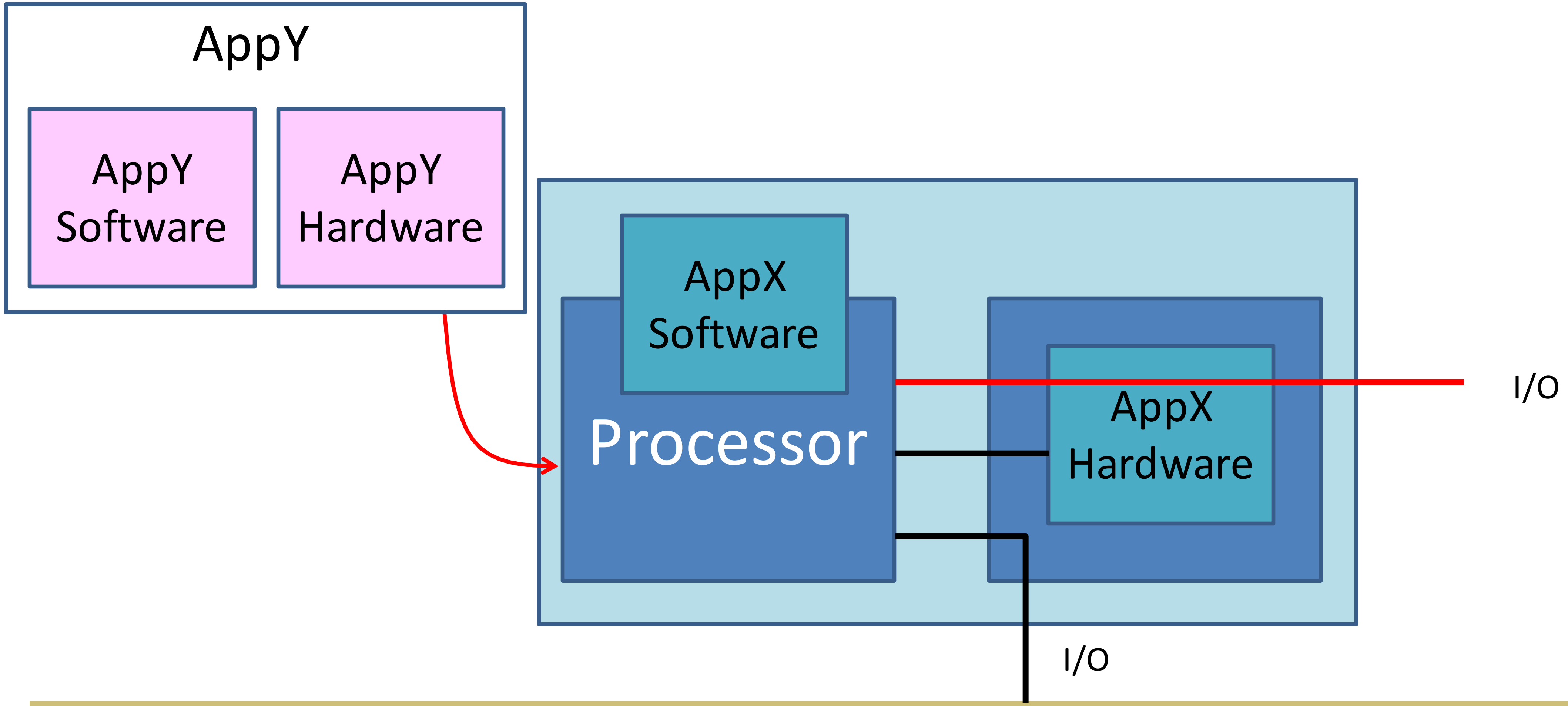
What we can’t do

What if we have two apps?



What we can’t do

What if it’s a single chip (and some I/O goes through the FPGA)

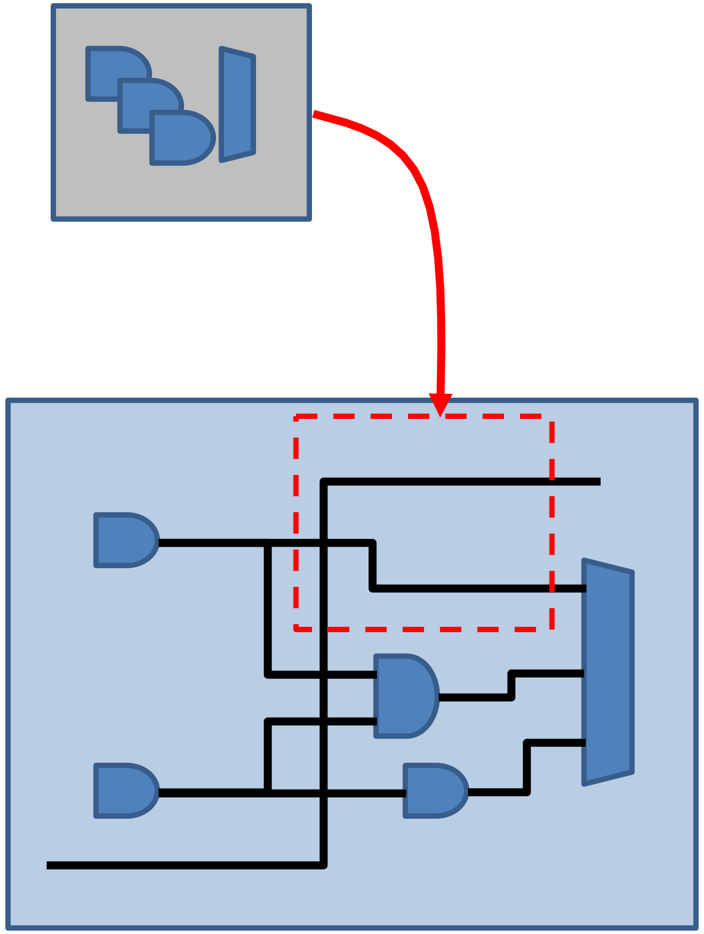


Why hasn't this been solved before?

- Over a decade of research has proposed two main solutions:
  - Run-time place-and-route
  - Slot-based reconfiguration

Approach 1: Run-time Place/Route

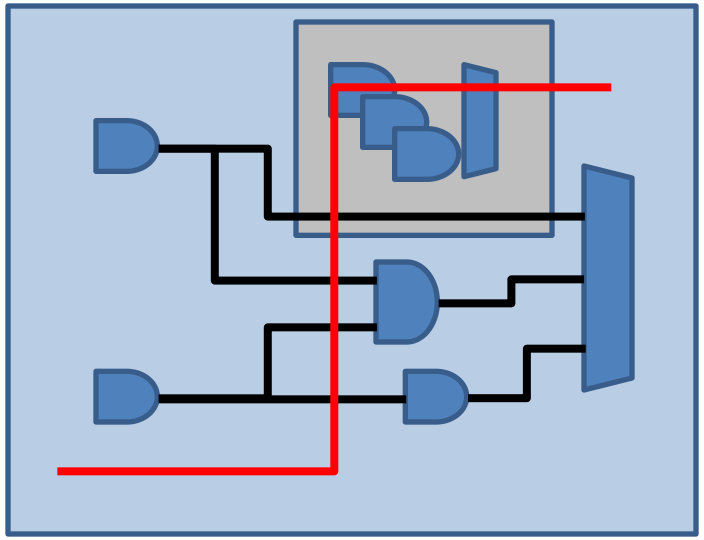
- There is free space in the FPGA
- Place a new module there



Approach 1: Run-time Place/Route

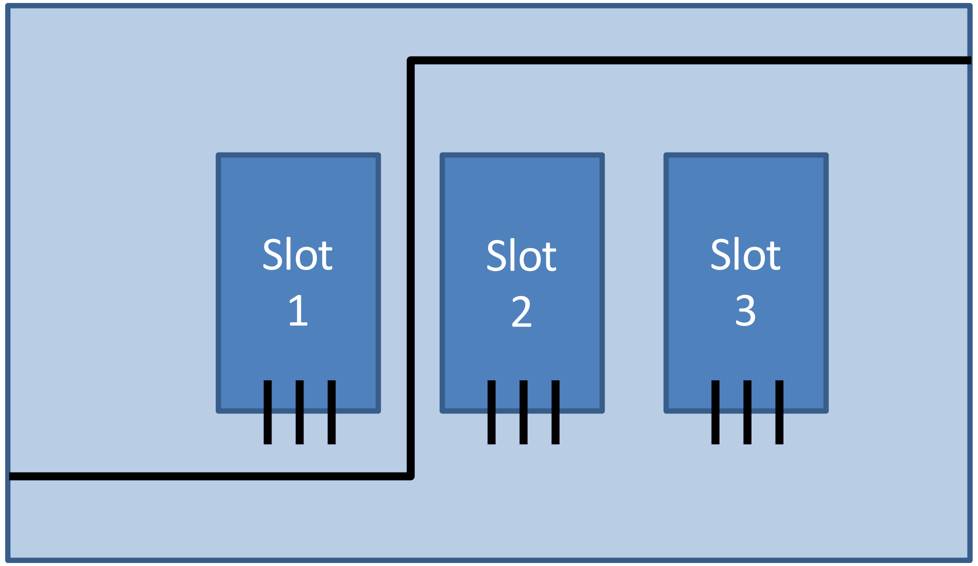
- Routing can fail
- Routing is also very time consuming

- **Therefore, is not practical**



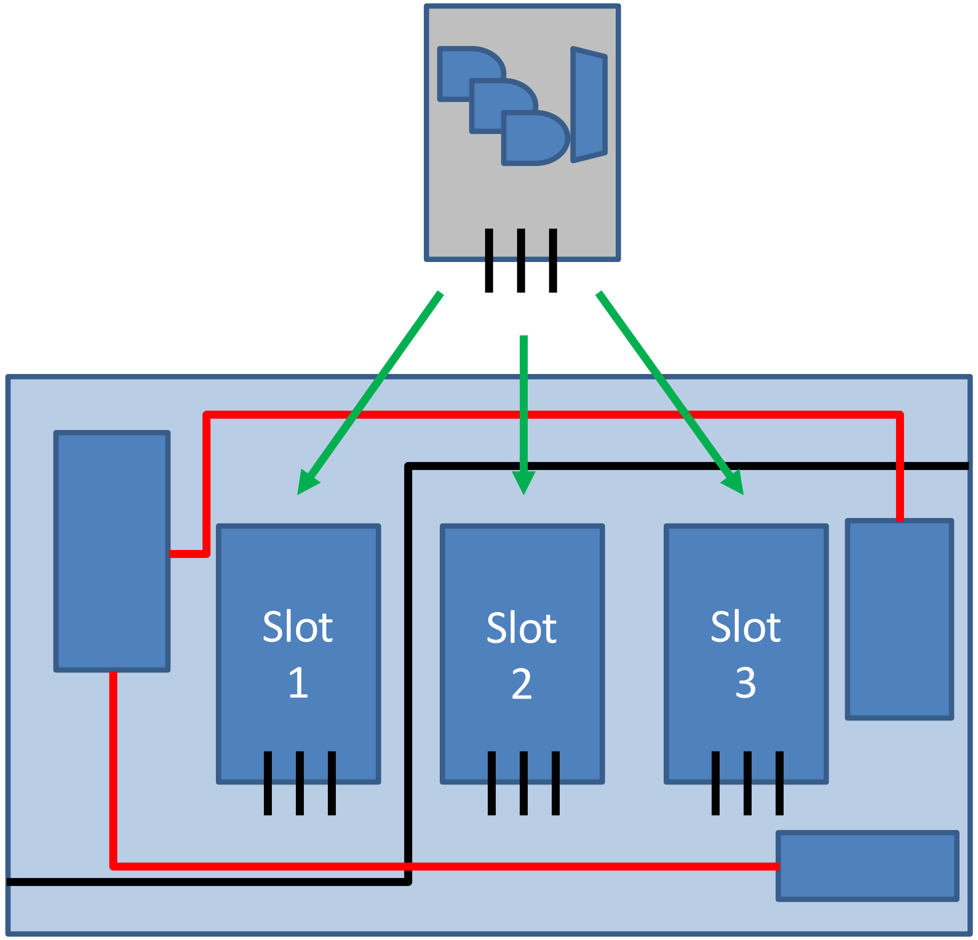
Approach 2: Slot-Based Reconfiguration

- Identical empty regions are reserved in FPGA
- Constrain tools to:
  - Not use wires/logic inside of slots
  - Use exact same wires for interface



Approach 2: Slot-Based Reconfiguration

- Hardware is loaded into slots
- Problem: if other logic exists, wire routing becomes very constrained
- **Therefore, is also not practical**



Previous Research

- Run-time Place and Route
  - Is very computationally expensive
  - Can possibly fail

- Slot-base Reconfiguration
  - Constrained routing is very restrictive and not applicable generally

- **Therefore, previous research is not practical**

Introducing Cloud RTR

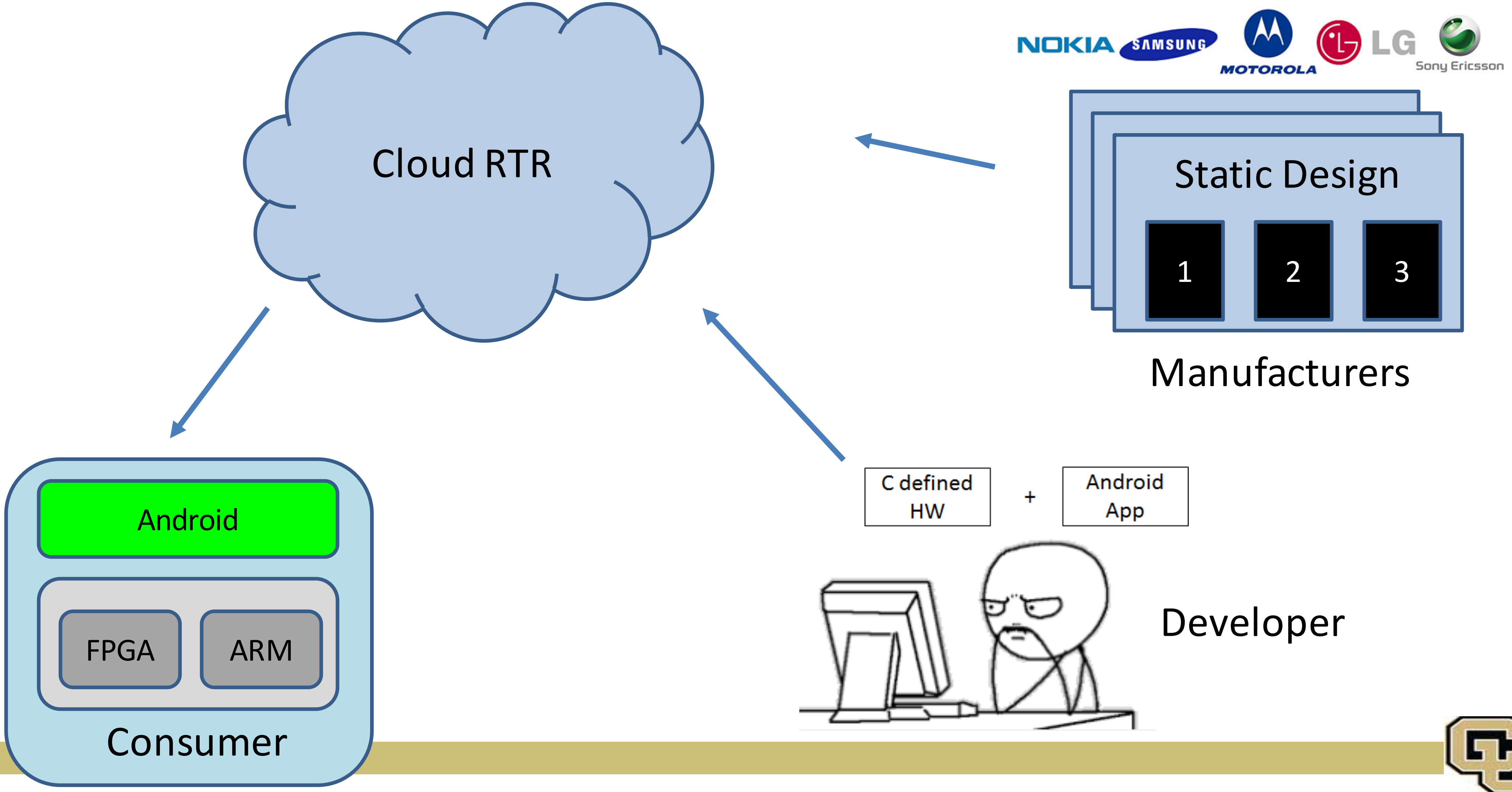
- Allows for sharing of the FPGA between general apps
- Uses existing vendor technologies
- Adopts the idea of slots from previous research
- Cloud RTR makes existing vendor technology work for general apps

The App Deployment Model

ANDROID APP ON
Google play

Available on the
App Store

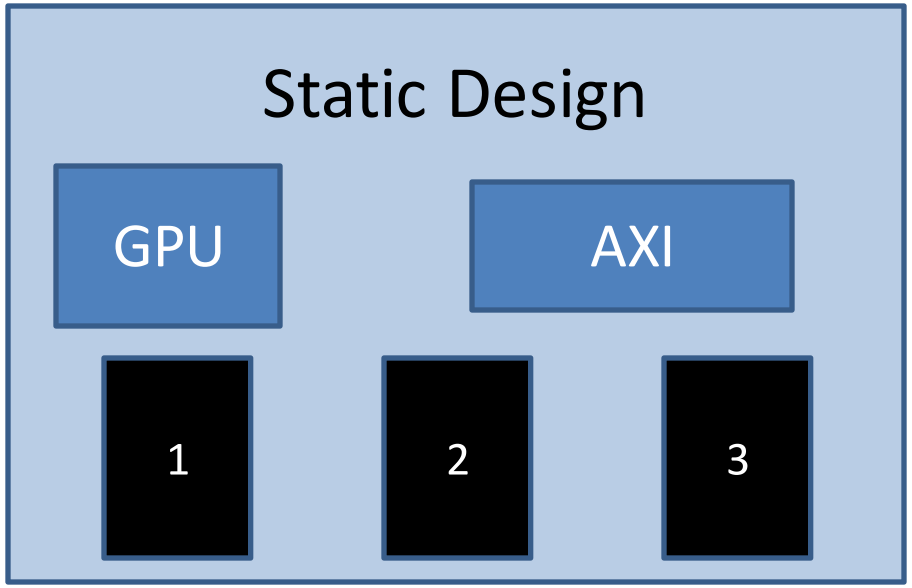
Cloud RTR



Cloud RTR
NOKIA
SAMSUNG
MOTOROLA
LG
Sony Ericsson
Static Design
1
2
3
Manufacturers
Android
FPGA
ARM
Consumer
C defined HW
+
Android App
Developer

Manufacturer

- Creates a static design
  - All logic that does not change
- Design includes areas reserved for slots
- Sends this to the cloud compiler



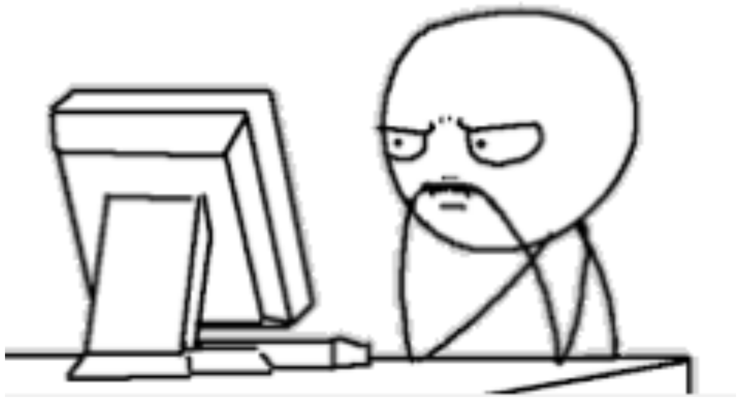
Developer

- Create an app using existing tools

C defined HW + Android App

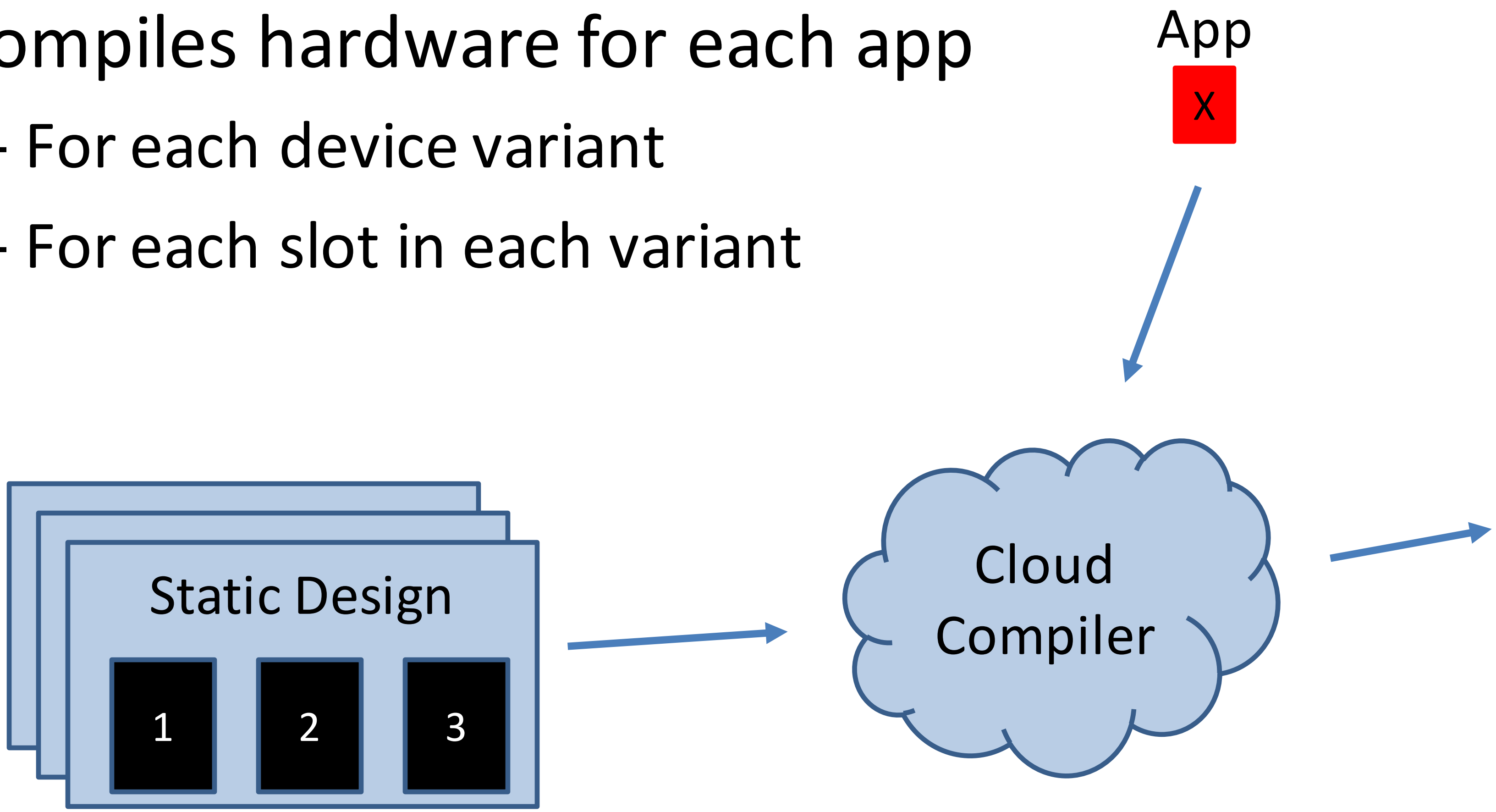
- Create a hardware definition in C

```
bool example(ap_uint<32> *in
             ap_uint<32> *out,
             bool *enabled,
)
```



App Store (Cloud Compiler)

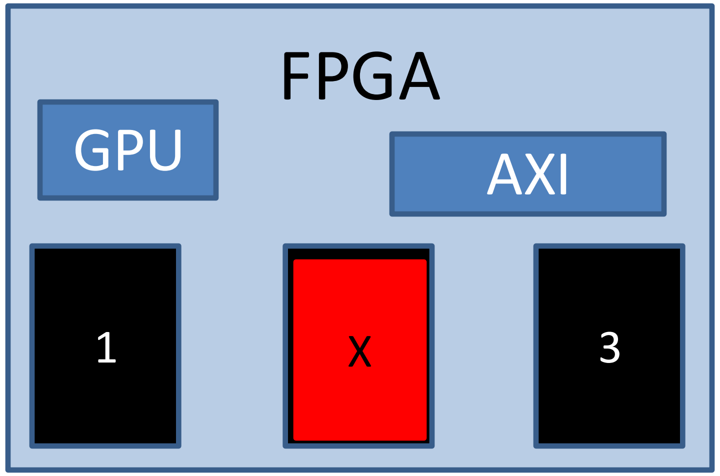
- Compiles hardware for each app
  - For each device variant
  - For each slot in each variant



[device1:
  [slot1: a.bit,
   slot2: b.bit,
   slot3: c.bit]]
[device 2:
  [slot1: d.bit,
   slot2: e.bit]]

User (Operating System)

- A system service manages slots
- Downloaded apps include slot hardware
- The system service loads app hardware for apps



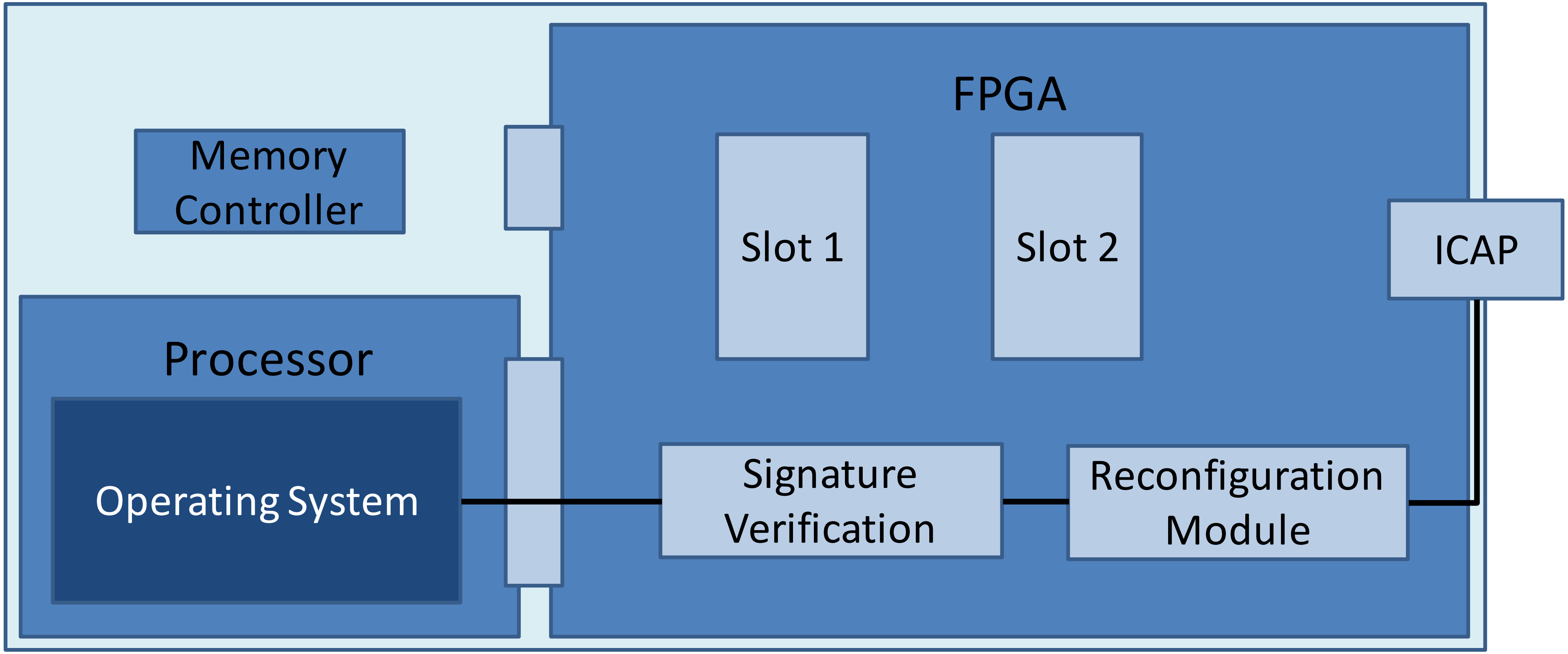

```
.apk:
  [device 1:
    [slot1: a.bit,
     slot2: b.bit,
     slot3: c.bit]]
```

Security Considerations

- The slot manager enforces access to hardware
- However, FPGAs can theoretically directly access sensitive resources (while bypassing the OS)
- A secure loading system ensures that apps cannot access sensitive resources

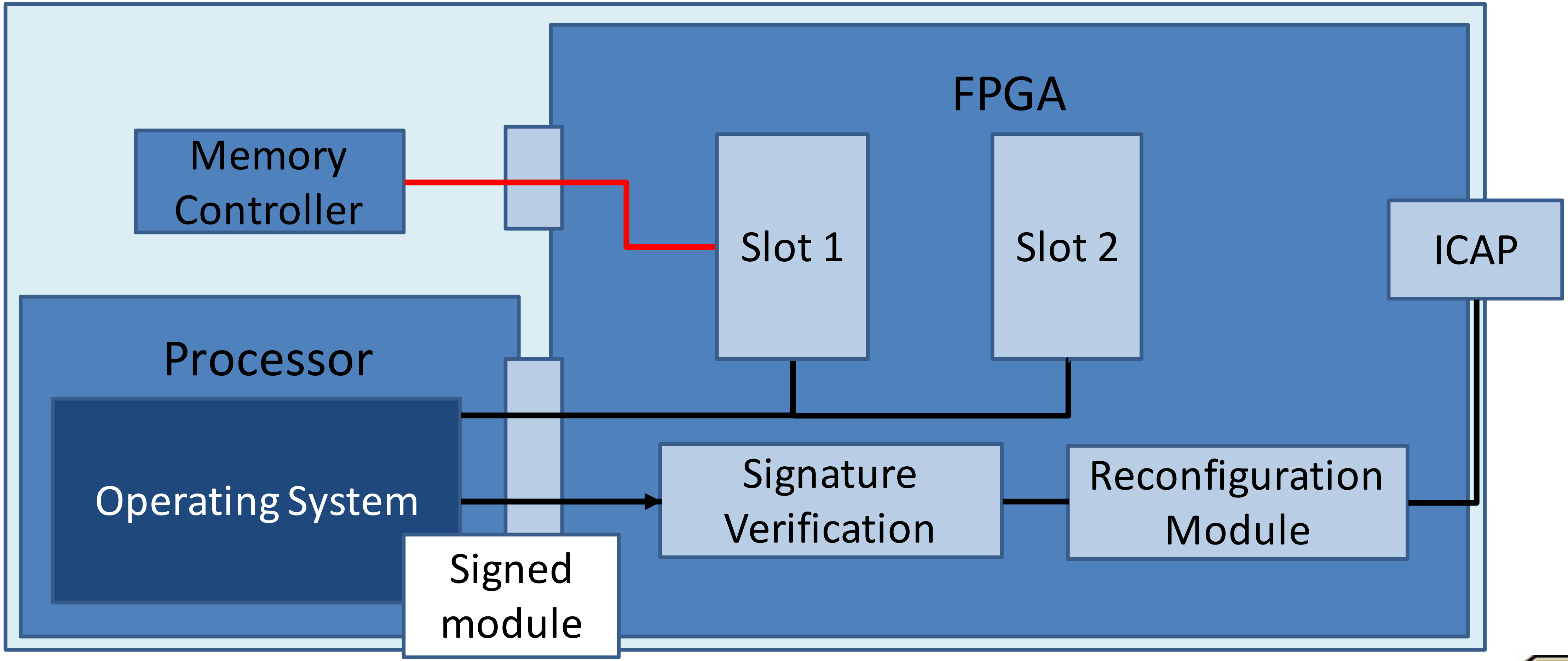
Secure loading system

How does the secure loader work?



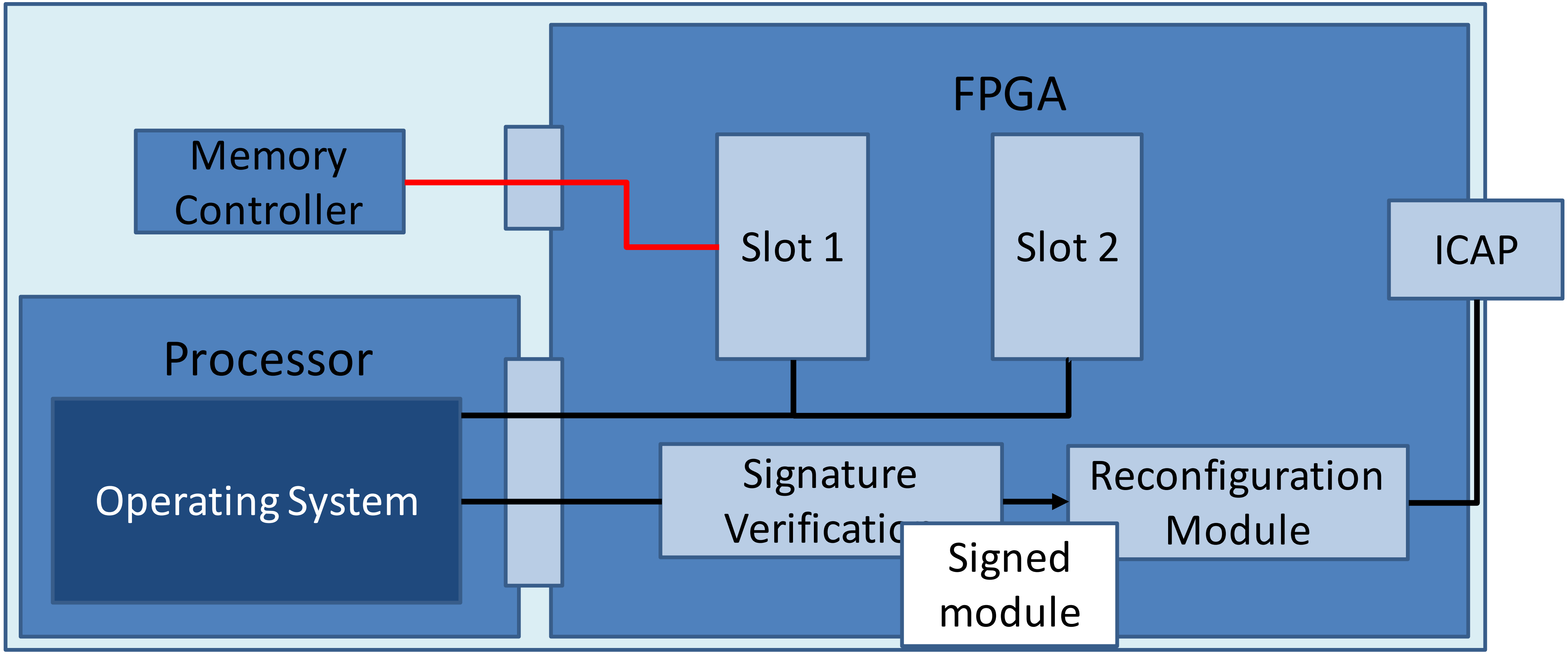
Secure loading system

The OS wants to reconfigure Slot 1



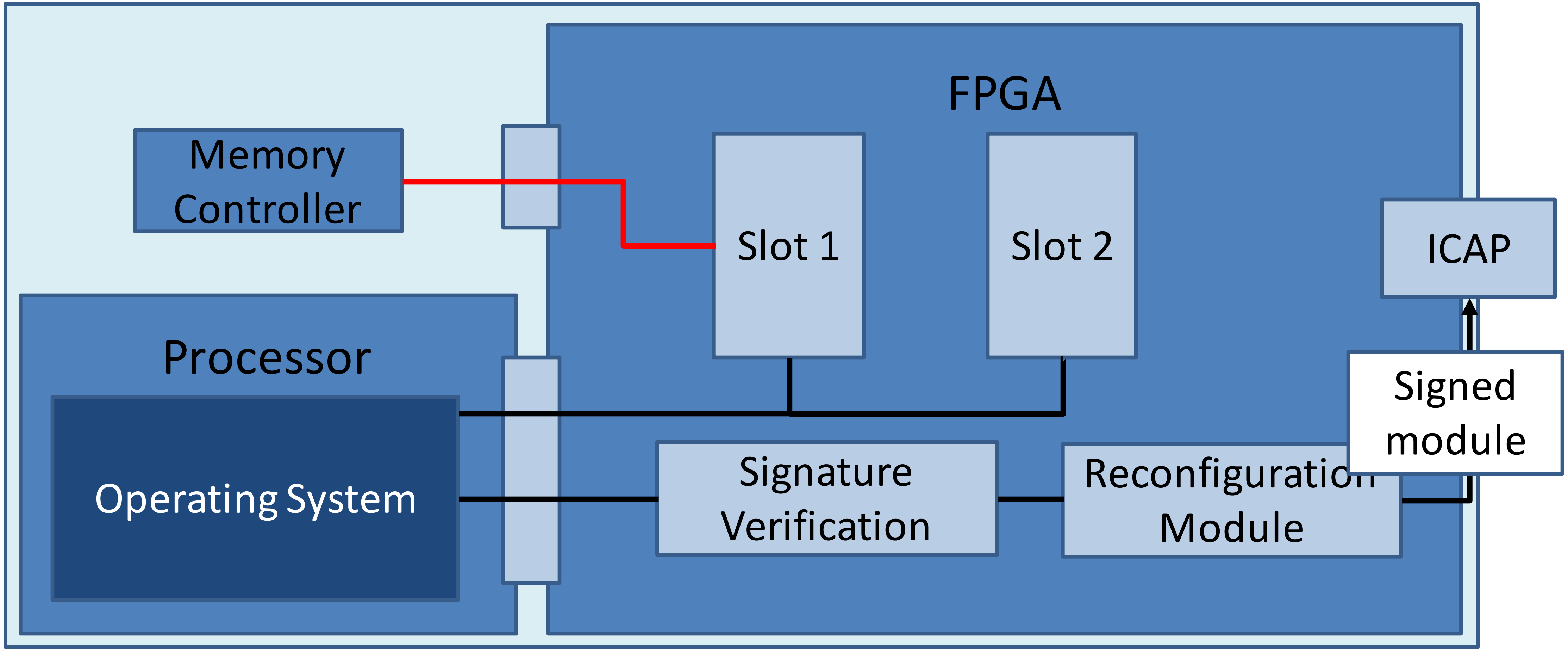
Secure loading system

The signature of the module is verified



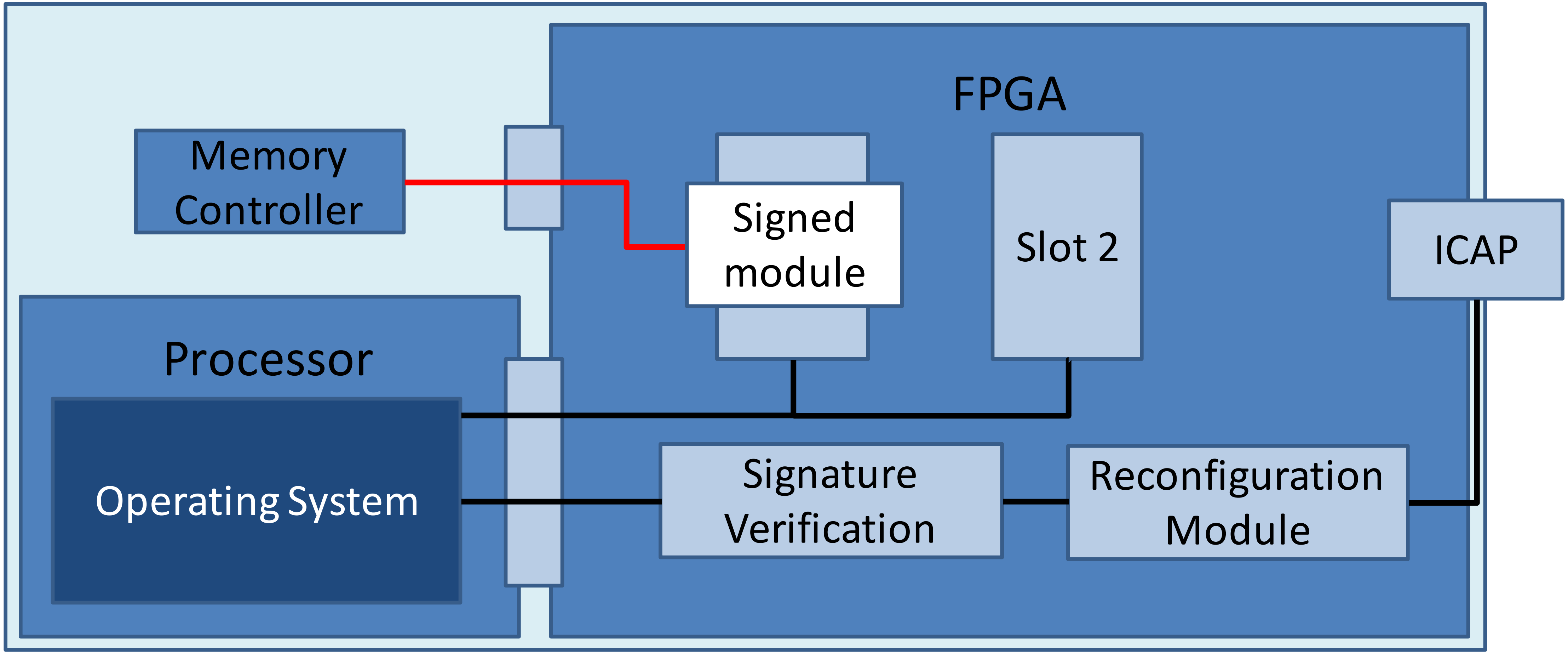
Secure loading system

The module is written to the ICAP



Secure loading system

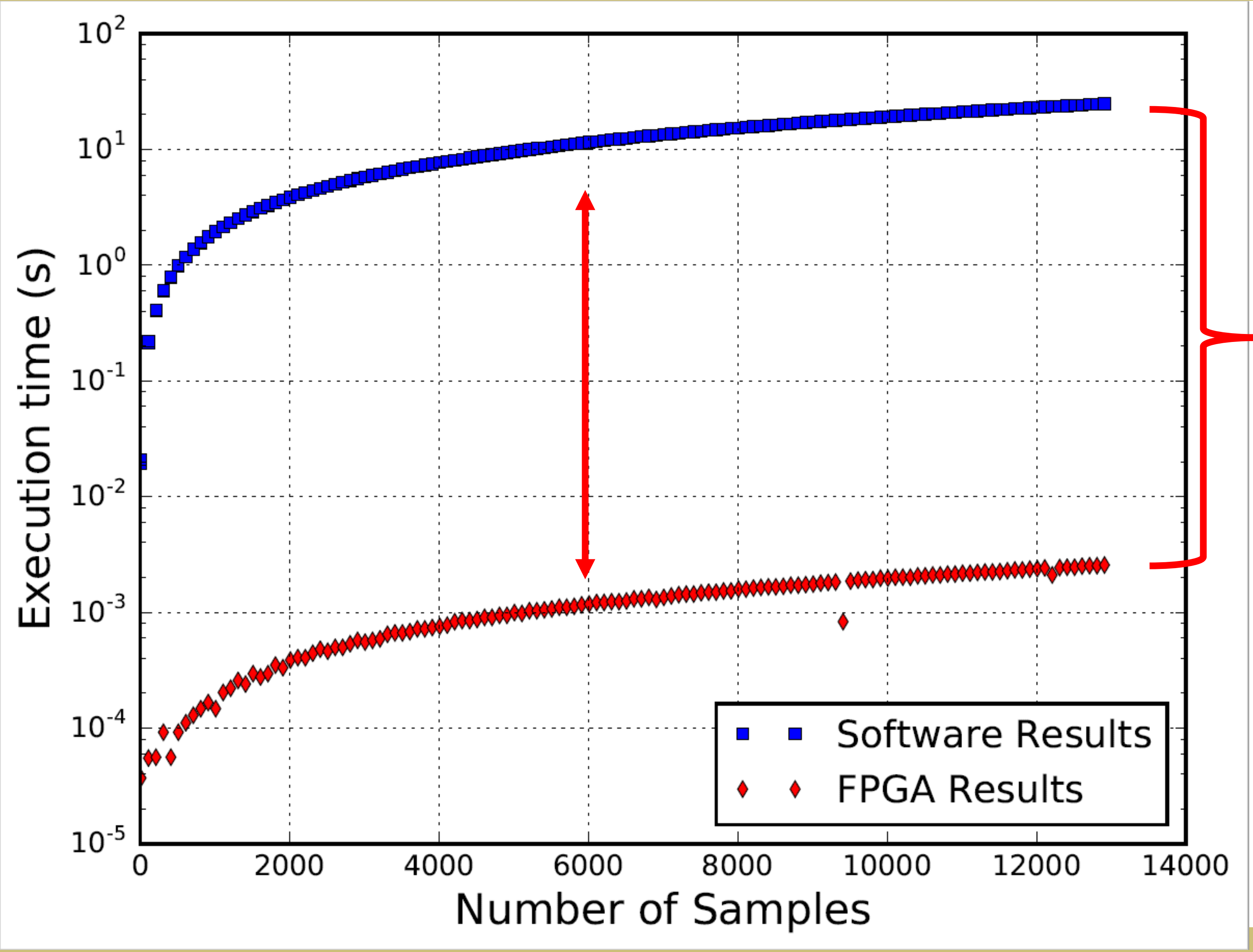
The ICAP performs the reconfiguration



Evaluation

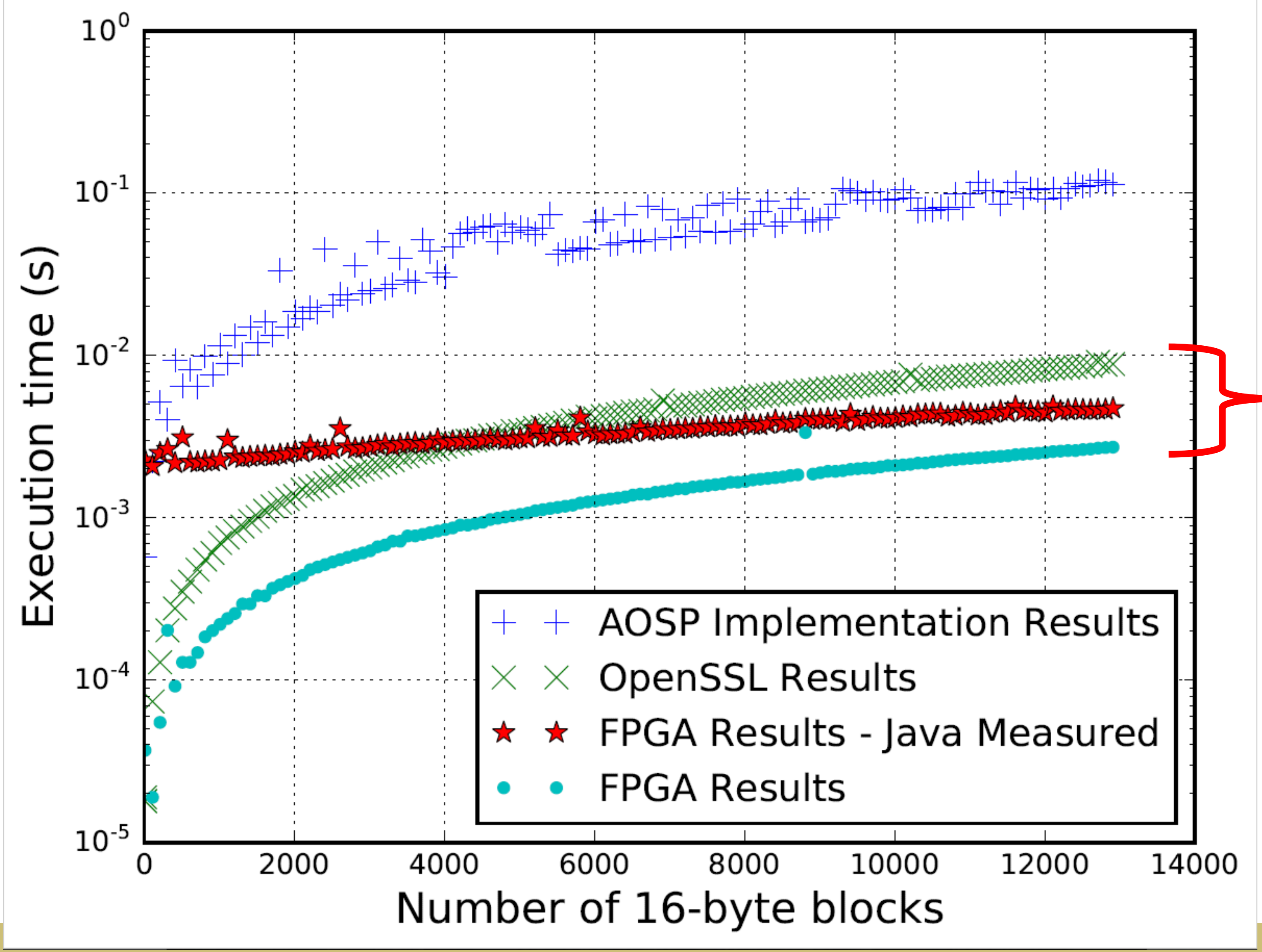
- Is there value in apps with hardware?
- Is the cloud-based compilation of Cloud RTR practical?

Micro benchmark 1: QAM demodulator



4 orders of magnitude

Micro benchmark 2: AES



FPGA is 3x vs. OpenSSL

Micro benchmark 3: Memory Scanner

- We also implemented a hardware memory scanner
- It can scan the entire address space transparently to the OS
  - 2.7% memory read performance hit
  - 5.5% memory write performance hit
- We tested this using the LMbench testbench

Brute-force compilation

Google Play Store Figures	
# of Apps as of Dec 14	1.43 Million
Average Monthly App Growth	6.10%
# of Apps for January 16	117,521

provided by AppFigures.

Brute-force compilation

Max # of Apps Compiled per day	
# of Slots	Apps
2	121
3	96
4	76
5	59
6	51

2 Slots Requirements	% of April Apps that use Hardware (# of Apps Uploaded per Day)		
	0.1 (3)	1 (34)	10 (347)
# of Device Variants	# of Machines Required to Compile Apps		
1	1	1	3
10	1	3	29
100	3	29	288
1000	29	288	2875

Reasonable for most scenarios

Brute-force compilation

Max # of Apps Compiled per day	
# of Slots	Apps
2	121
3	96
4	76
5	59
6	51

6 Slots Requirements	% of April Apps that use Hardware (# of Apps Uploaded per Day)		
	0.1 (3)	1 (34)	10 (347)
# of Device Variants	# of Machines Required to Compile Apps		
1	1	1	7
10	1	7	69
100	7	69	681
1000	69	681	6809

Still reasonable for most scenarios

Reducing the numbers even more

- Compilation can be offloaded to manufacturers
- Manufacturers will likely reuse designs (Qualcomm, ARM chips are often reused)
- Developers will likely use libraries

Implementation Case Study: Orbot

- Tor on Android
- AES is on the critical path
- Examine AES as an integration study

Implementation Case Study: Orbot

What we found:

- Memory operations are the bottleneck
  - Data must be placed correctly in memory
  - Userspace I/O has high overhead
  - Many system calls are incompatible with UIO
- It is easier to build an application from ground-up

Conclusion

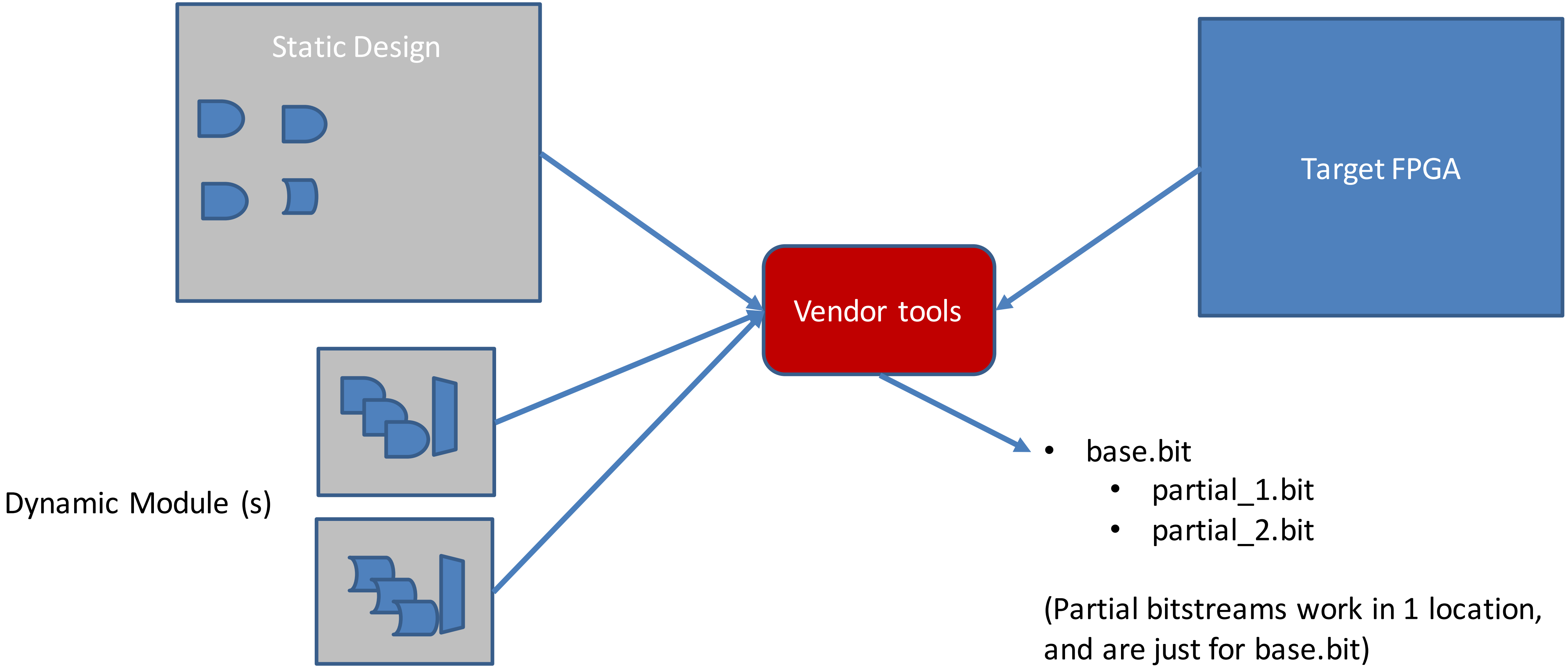
- We have presented our vision of apps with hardware
- Cloud RTR implements our vision by leveraging the mobile app deployment model
- We have demonstrated the value and practicality of our vision

Questions?

- Email: michael.coughlin@colorado.edu
- Source code: https://github.com/nsr-colorado/cloud-rtr

Vendor Supported Partial Reconfiguration

Goal: Space saving for customer



Examples of Libraries

- Crypto
  - Asymmetric (RSA, ECDSA, etc…)
  - Symmetric (3DES, Twofish, Blowfish)
- Soft processors
- Encoding
  - Network encoding (Reed-Solmon, etc…)
  - Media encoding (JPEG, MPEG, etc…)
- DSP
  - FFTs, Filters, etc…

Example hardware definition

```
bool example(ap_uint<32> *in
             ap_uint<32> *out,
             bool *enabled,
)
```

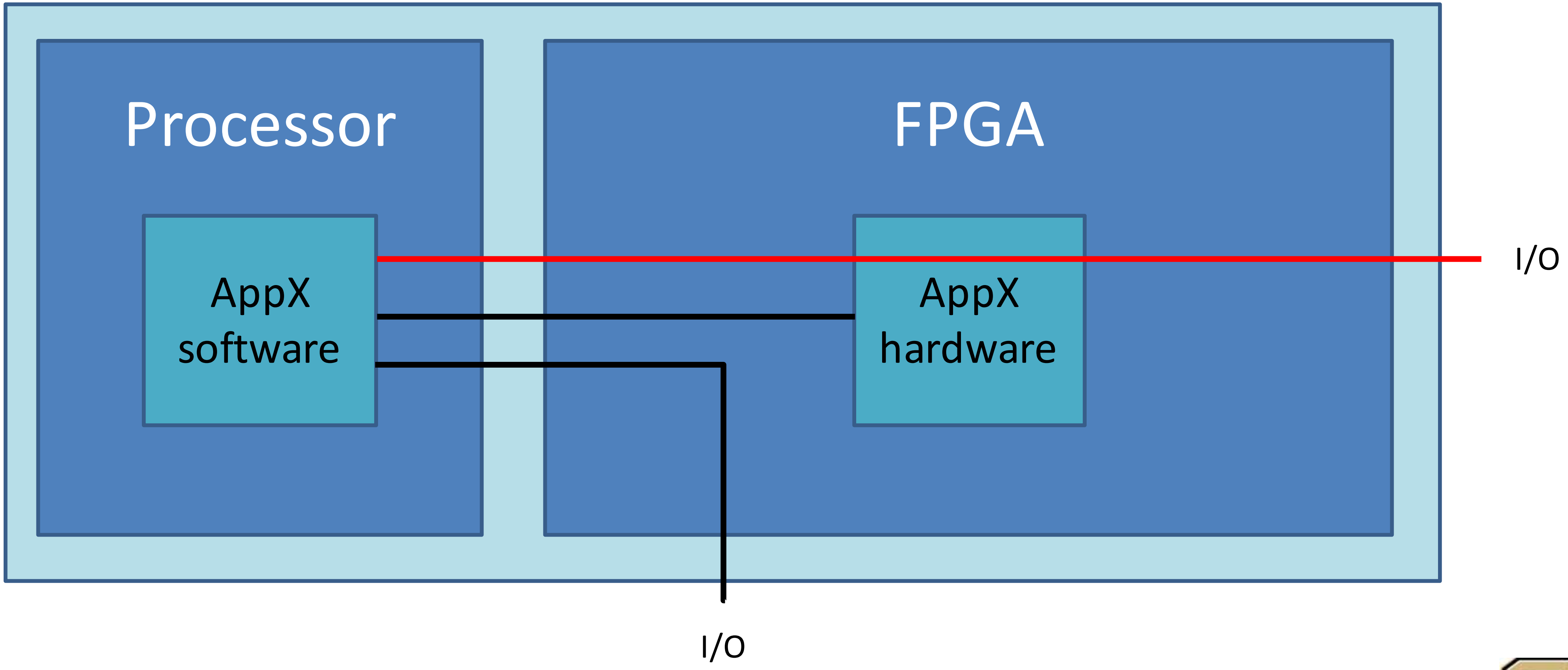
More complicated hardware definition

```
typedef ap_uint<32> uint32_t_hw;
typedef hls::stream<uint32_t_hw> mem_stream32;

bool aes(volatile unsigned int m_mm2s_ctl [500],
            volatile unsigned int m_s2mm_ctl[500],
            volatile unsigned sourceAddress,
            ap_uint<128> *key_in,
            ap_uint<128> *iv,
            volatile unsigned destinationAddress,
            unsigned int numBytes,
            int mode,
            mem_stream32& s_in,
            mem_stream32& s_out
)
```

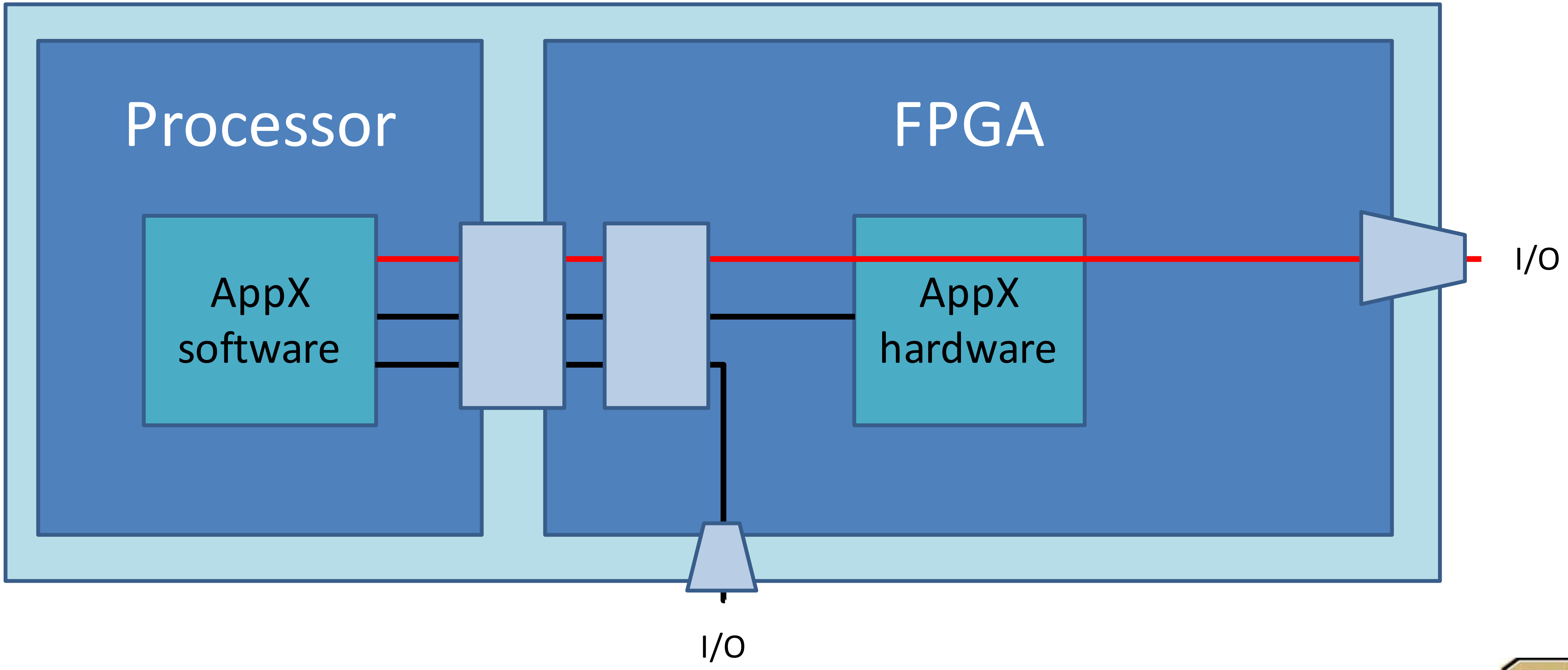
The problem

Let's examine the problem



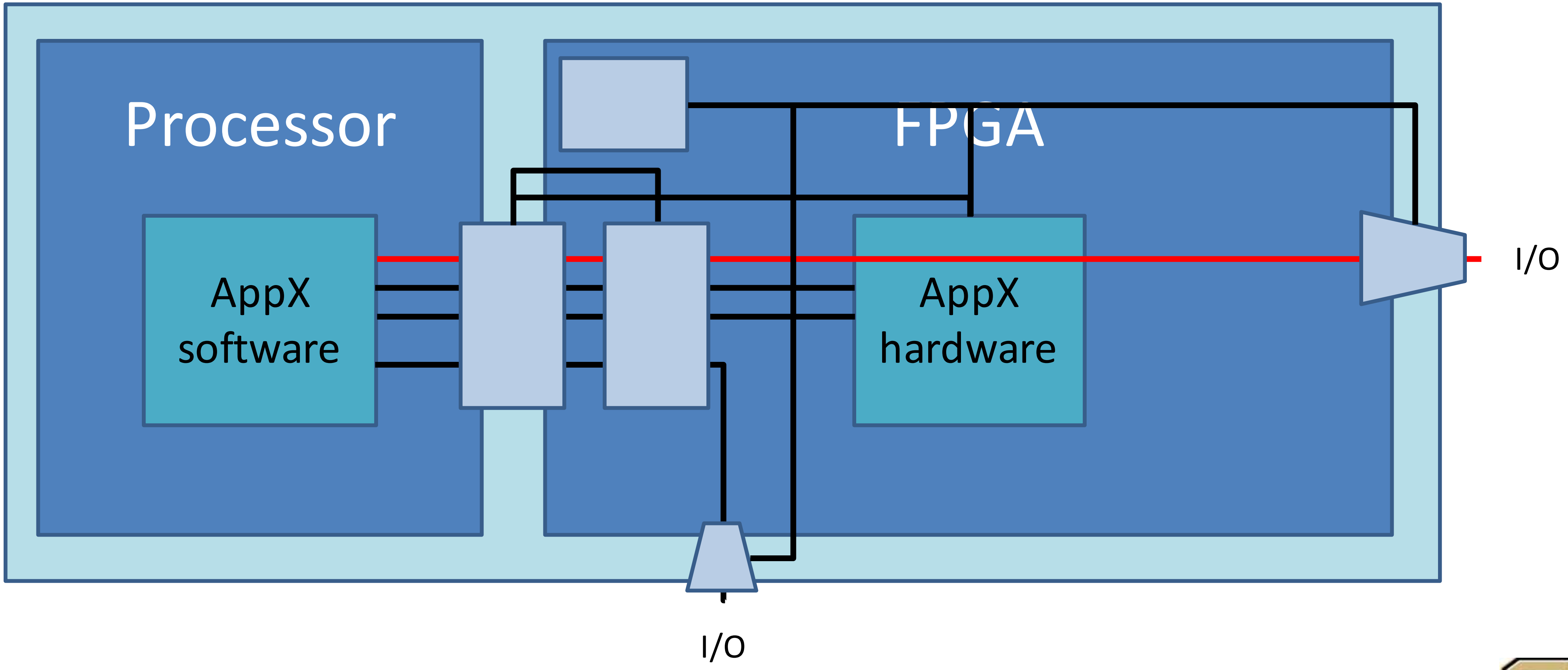
The problem

First, there are various interconnects needed



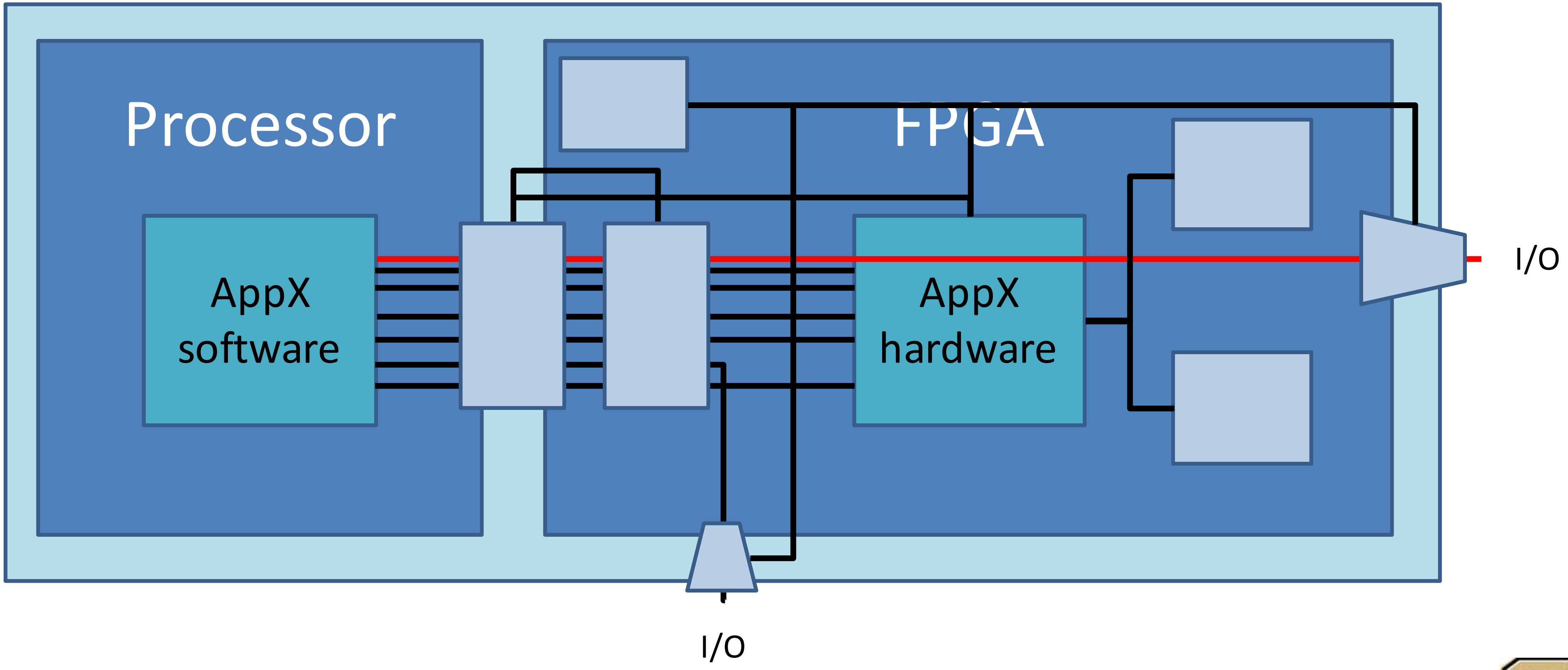
The problem

Control signals and logic must also be placed



The problem

The app may have complex inputs, or need to interact with other logic



Secure loading system

- A trusted system is booted with Secure Boot
- Included is a static module that reconfigures slots
- This module only allows signed modules into slots that access sensitive resources

Our solution

- Builds off of prior research…
- …but in a way that is compatible with vendor tools
- To do this, we leverage the deployment model for mobile apps