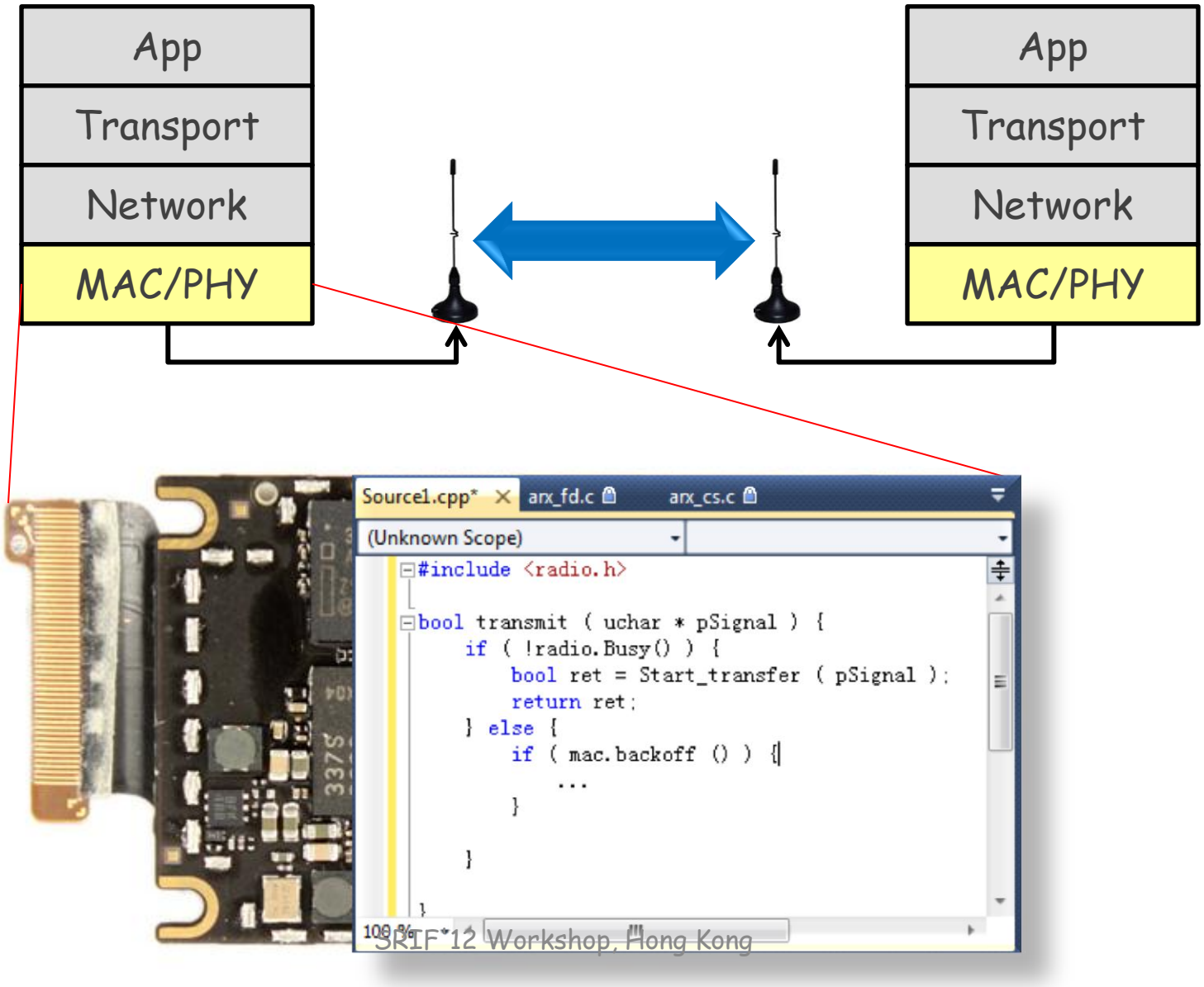
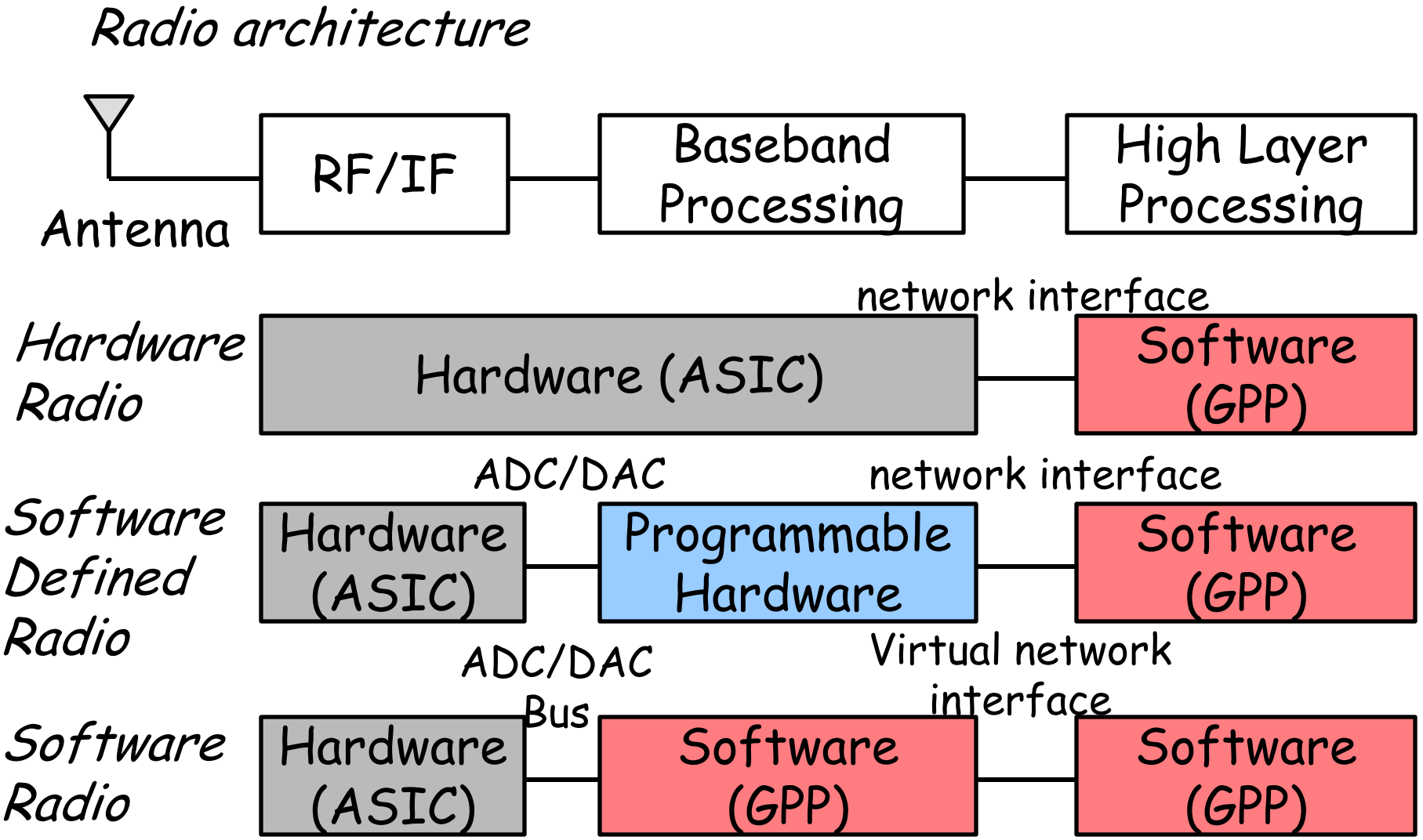
Empower Wireless Revolution with the Magic of Software

Kun Tan
Lead Researcher
Wireless and Networking Group, MSR Asia

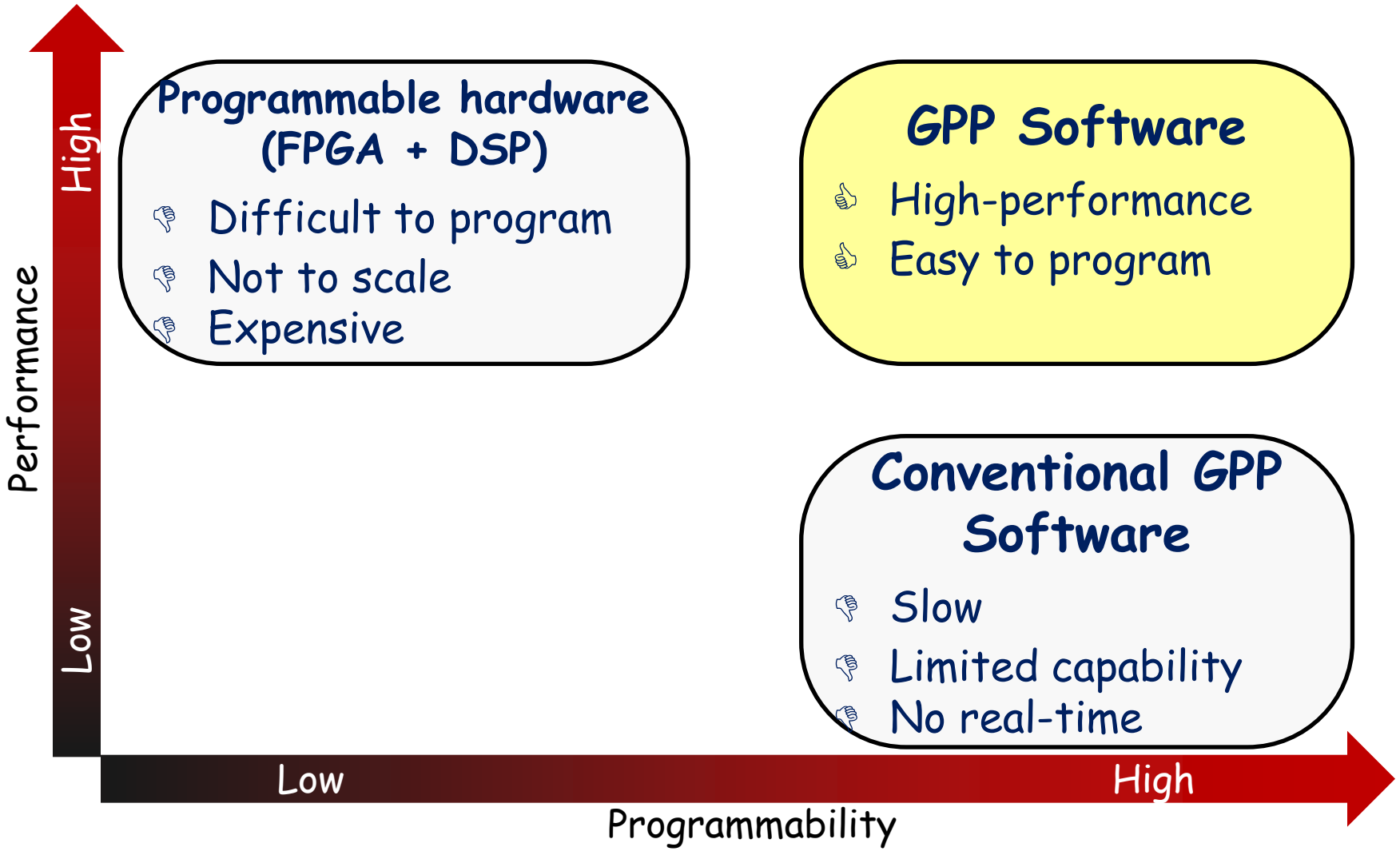
The Dream of Software Radio



From Hardware to Software



Approaches


Performance
High
Low
Programmable hardware (FPGA + DSP)
Difficult to program
Not to scale
Expensive
GPP Software
High-performance
Easy to program
Conventional GPP Software
Slow
Limited capability
No real-time
Low
High
Programmability

Fundamental Challenges

- System interface throughput
  - Large volume of high-fidelity digital samples
  - From 1Gbps to 10Gbps
- Computation
  - Large amount of arithmetic calculations for digital signal processing
  - Tens of GOPS estimated
- Real-time support
  - Hard deadline and accurate timing control for wireless protocols
  - From 10ms (multi-media) to 1 μs

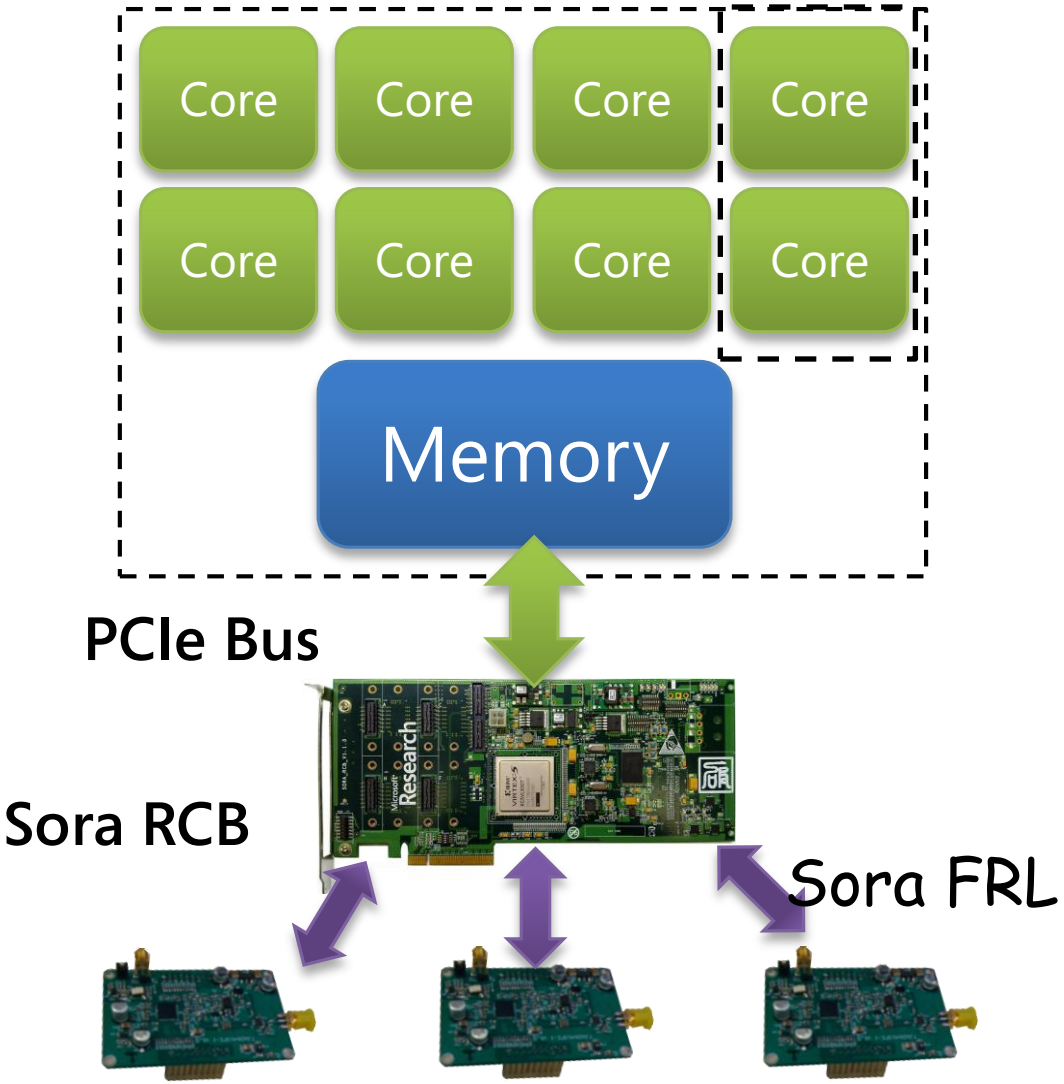
The Sora Project

- A multi-year research project (started 2007)
  - Bet on the commodity multi-core PC
- Successfully turning a commodity PC into a powerful software radio
  - Orders of magnitude better than prior art
  - First standard compliant WiFi (802.11 a/b/g)
  - First demo LTE (uplink)

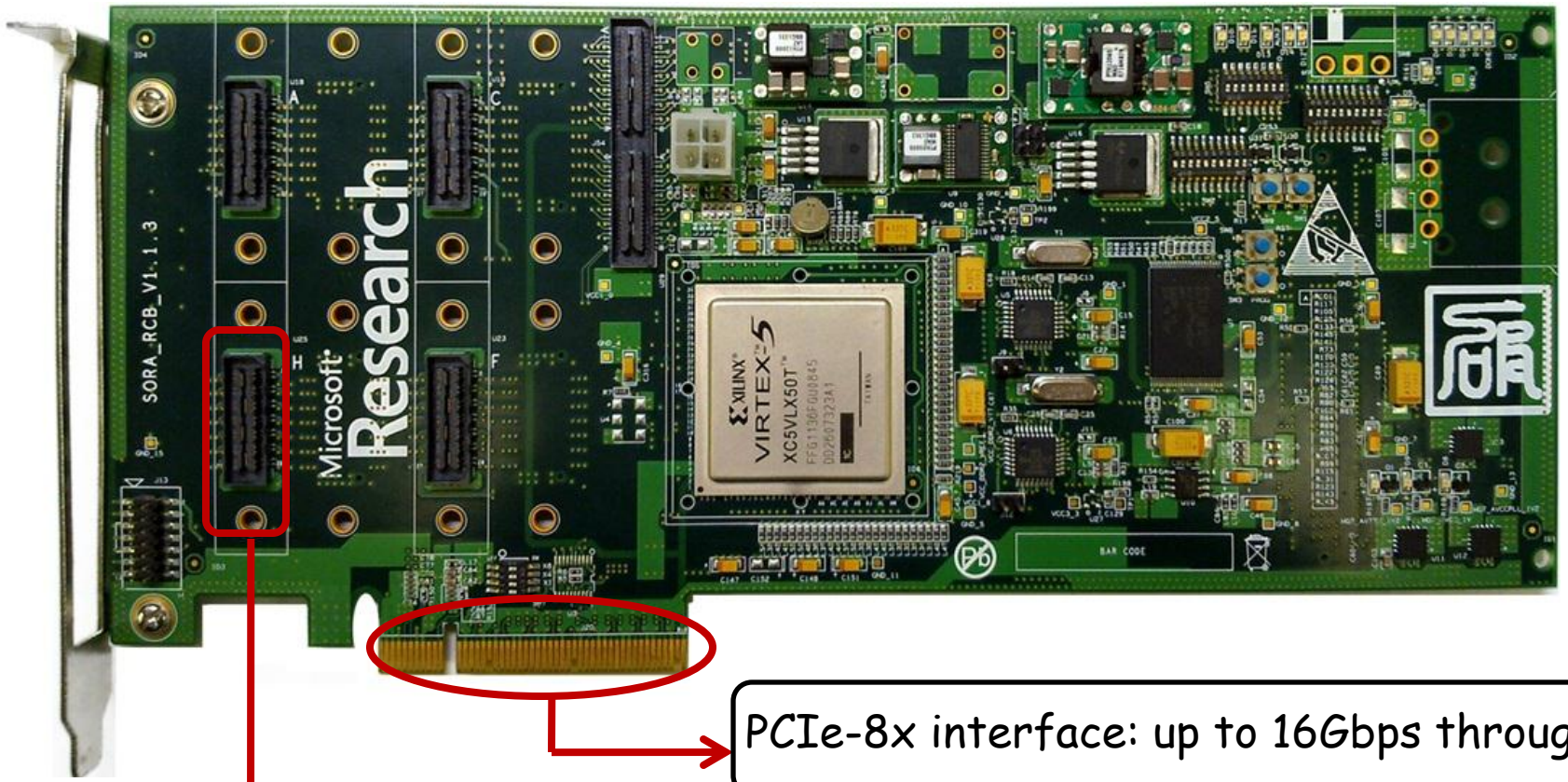
Meeting the Challenges

- Take advantage of modern PC bus technologies
  - Driven by high-speed interconnection among PC subsystems
  - PCIe-based interface (> 10Gbps)
- Ride the wave of multi-core technology
  - Sustain Moore's law when CPU hits the heat wall
  - Software innovation to unlatch the full power of modern CPU

Sora Architecture



Sora Radio Control Board



PCIe-8x interface: up to 16Gbps throughput

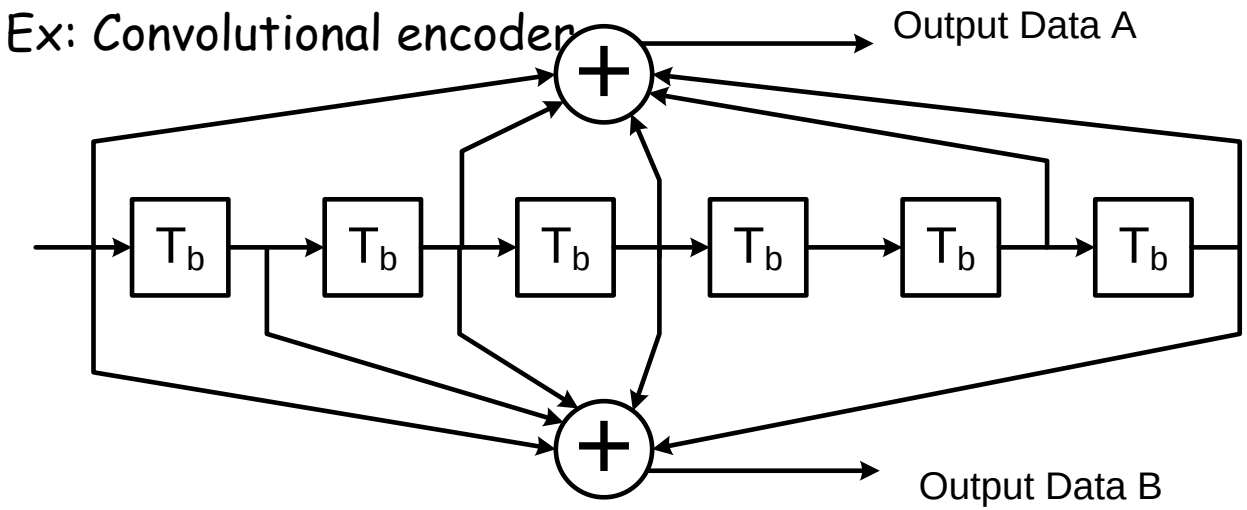
Sora Fast Radio Link Slot
- 2Gbps per channel
- up to 8 channels (8x8 MIMO)

Software Optimizations

- High performance PHY implementation on multi-core
  - Trade memory for computation
  - Exploit data parallel with SIMD
  - Streamline across multiple cores
  - Core dedication for real-time support

Trade Memory for Computation

- Exploit large high-speed cache memory of CPU
  - Multiple mega byte L2/L3 cache
- Extensive use of lookup tables (LUT) to store the computation



Direct impl. 8 ops per bit

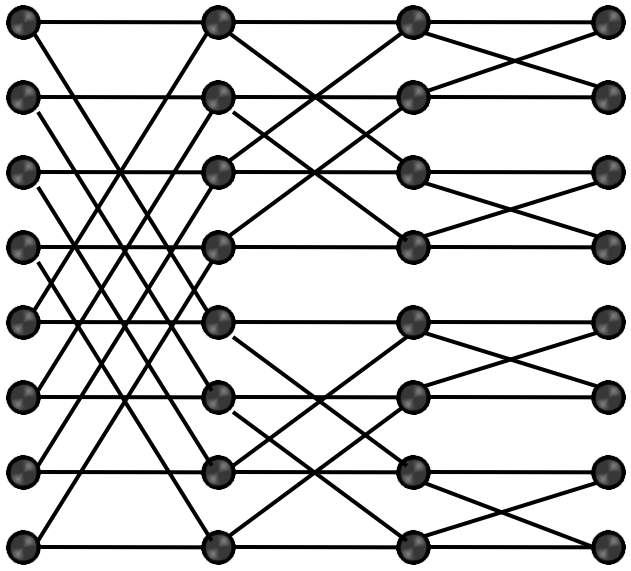
LUT impl. 2 Look-up op for 8 bits! (size 32KB)

  - Applicable for more than half of the common algorithms; speedup ranges from 1.5x to 22x

Exploit Data Parallelism with SIMD

- Utilize short-vector SIMD extension in CPU
  - Simultaneously perform calculations on multiple elements of vectors

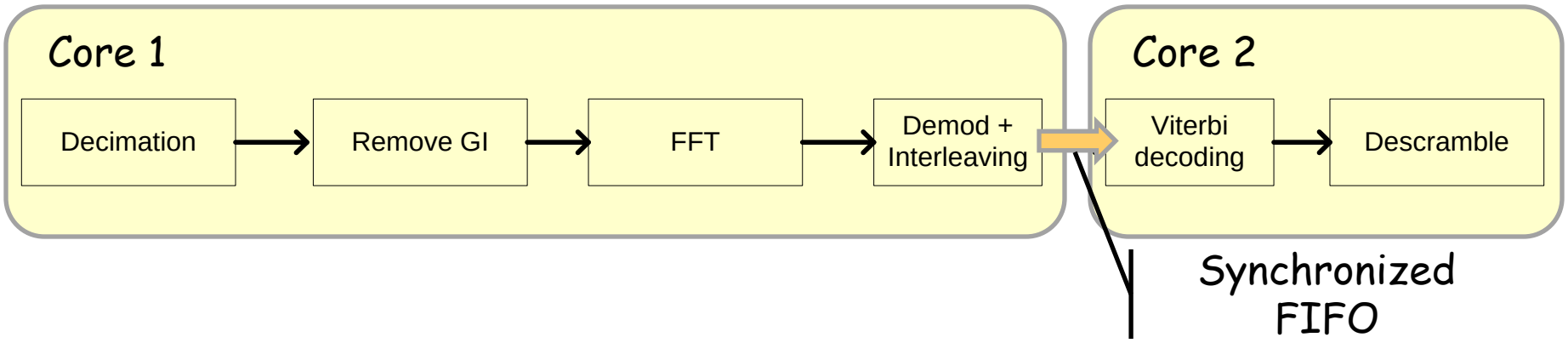
Ex. (I)FFT



  - Applicable to many PHY algorithms with significant speedups (1.6x ~ 50x)

Pipeline across Cores

- Partition PHY processing work across cores
- Interconnecting sub-pipeline with light-weight, synchronized FIFOs



Core Dedication for Realtime

- Software is considered "uncertain" in traditional OS
  - Multiplexed with multiple tasks/processes/threads
  - Interrupts and contention in memory/bus
  - RTOS – complex/overhead/limited functionality
- Core dedication – a dumb idea whose time comes
  - Exclusively allocate enough cores for RT tasks
  - I/O Polling, instead of interrupt
  - Precise timing control at CPU clock level
  - Simple abstraction, and easier to implement in standard OSes
    - Even implemented in Windows ☺

Worldwide Recognition

- NSDI'09 Best Paper, Best Demo
- Called out at SIGCOMM'09 keynote speech
- "Honorable Mention" Demo, Mobicom'09
- Highlighted at CACM January 2011
  "One the most significant wireless papers"
- SIGCOMM'10 - Best Demo Award


research highlights

DOI:10.1145/1866739.1866759

Technical Perspective
Sora Promises Lasting Impact
By Dina Katabi

THE TERM SOFTWARE defined radio (SDR) first appeared in 1993 and referred to a radio transceiver where the basic signal processing components (for example, filtering, frame detection, synchronization, and demodulation) are all done in a general purpose processor. The goal of an SDR was to enable a single radio to support multiple wireless technologies (for example, AM, VHF, FM) and be easily upgradable with a software patch.

While the concept of SDR has been around for decades, only recently have SDRs become common in academic wireless research. However, research projects typically employ SDR as a development platform, that is, they use software radios to develop new physical layer designs with the understanding that if these designs make it to a product they will be built in ASICs. The reason why SDR has become a development platform rather than a fully functional software radio is that building high performance SDRs has turned out to be very challenging.

Sora has revived the original SDR vision. The objective of Sora is to build an SDR that combines the performance and flexibility of hardware platforms with the programmability and flexibility of general purpose processors. To do so, Sora must overcome the following challenge: How can a radio deliver high throughput and support real time protocols when all signal processing is done in software on a PC?

Sora's approach uses various features common in today's multicore architectures. For example, transferring the digital waveform samples from the radio board to the PC requires very high bus throughput. While alternative SDR technologies employ USB 2.0 or Gigabit Ethernet, Sora opts for PCI Express. This design decision enables Sora to achieve significantly higher transfer rates, which are important for high bandwidth multi antenna designs. The choice of PCI express also enables Sora to reduce the transfer latency to sub microseconds, which is necessary for wireless protocols with timing constraints (for example, MAC protocols). Further, to accelerate wireless processing, Sora replaces computation with memory lookups, exploits single instruction multiple data (SIMD), and dedicates certain cores exclusively to real time signal processing.

There are many reasons why the following paper about Sora stands out as one of the most significant wireless papers in the past few years. First, it presents the first SDR platform that fully implements IEEE 802.11b/g on standard PCs. Second, the design choices it makes (for example, the use of PCIe, SIMD, trading computation for memory lookups, and core dedication) are highly important if software radios are ever to meet their original goal of one radio for all wireless technologies. Third, the paper is a beautiful and impressive piece of engineering that spans signal processing, hardware design, multicore programming, kernel optimization, and so on. For all these reasons, this paper will have a lasting impact on wireless research.

The Sora platform has been used in multiple research projects and real time demos. It has enabled demanding designs, such as LTE and AP virtualization, to be built fully in software. However, currently most SDR based research uses the GNU Radio/USRP platform. Despite the limitations of this platform, previous attempts at replacing it with more capable platforms did not experience significant success. In fact, history shows that wide adoption is not necessarily correlated with the more capable design. One of the classic papers we teach our undergraduate students is "The Rise of Worse is Better" by Richard Gabriel that explains why the Lisp language lost to C and Unix. Gabriel argues that for wide adoption, a system must be good enough and as simple as possible. Such a design (termed worse is better) tends to appear first because the implementer did not spend an excessive amount of time over optimizing. Therefore, if good enough, it will be adopted by developers because of its simplicity. Once adopted, the system will gradually improve until it is *almost* the right design. One may argue that the history of the GNU Radio/USRP SDR is fairly similar; the platform originally provided just enough for people to start experimenting with the wireless physical layer. As a result, it was simple and cheap, which caused it to spread. Once it was accepted, it kept improving just enough to enable the next step in research.

The Sora team has recently started a program that awards Sora kits to academic institutions to enable them to experiment with this new platform. It will be interesting to see whether Sora with its higher performance can eventually replace the GNU Radio/USRP platform. If this happens, it will be a major success for Sora.

Dina Katabi (dina@mit.mit.edu) is an associate professor in the Electrical Engineering and Computer Science Department at Massachusetts Institute of Technology, Cambridge, MA.

There are many reasons why the following paper about Sora stands out as one of the most significant wireless papers in the past few years.

- Sora Academic Kit release!
  - HW/SW available for academic research
  - Over 230 units shipped worldwide

Sora Kit Evolution

- Sora ver 1.02
  - Many assembly code for SIMD
  - Kernel mode only
  - Integrated programs
  - 802.11b samples
- Sora ver 1.1
  - Vector1 Library
  - Kernel mode only
  - Integrated programs
  - 802.11a/b/g samples

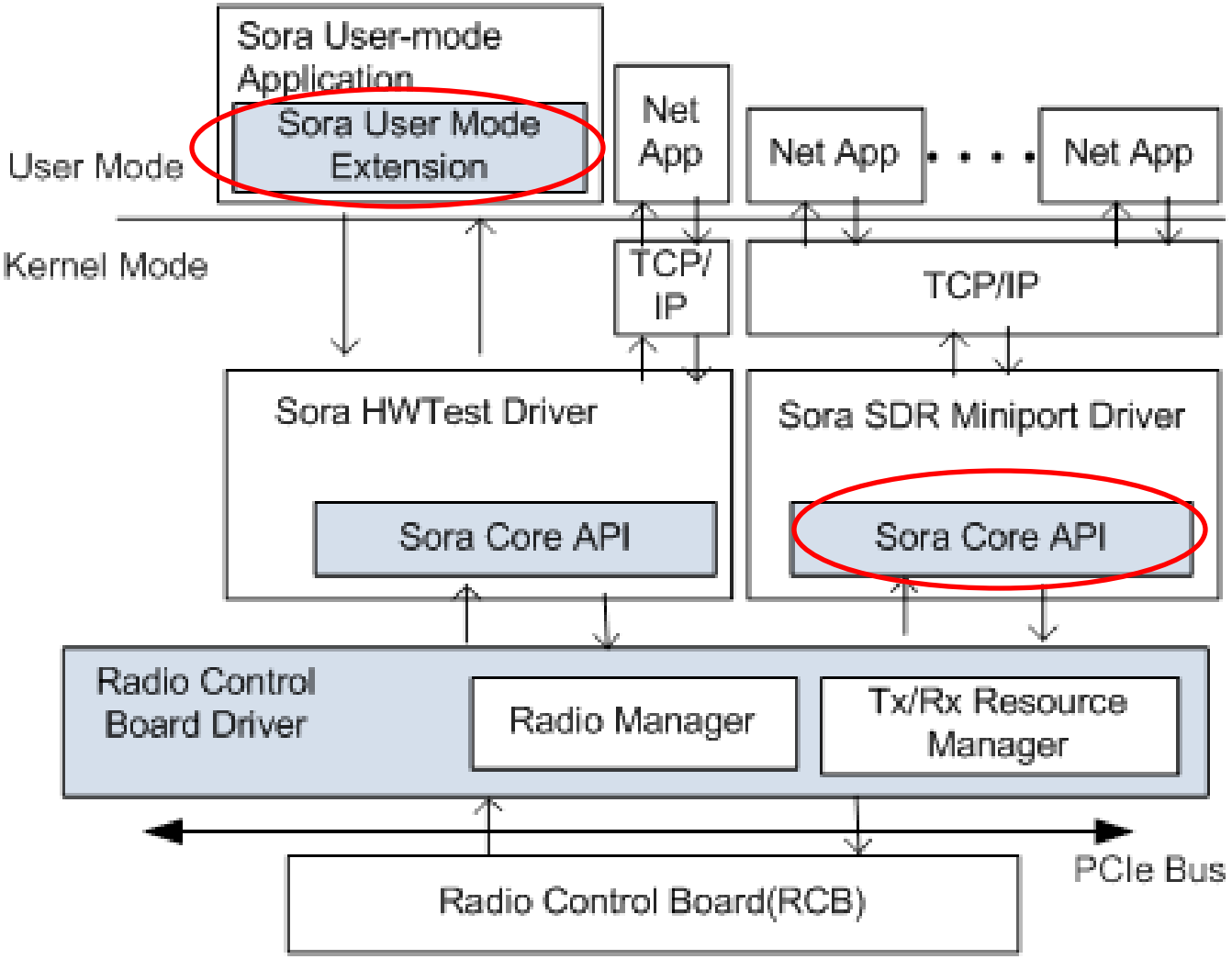
Sora Kit Evolution (cont.)

- Sora ver 1.5 (lastest Sept 2011)
  - Kernel mode and User Mode Extension
  - Full capability with Windows XP
  - HW verification tool
  - Integrated program
- Sora ver 1.6 (coming soon)
  - UMX reflection
  - Brick modular architecture
  - 802.11b Brick Sample
  - DebugPlot tool

Sora Kit Evolution (cont.)

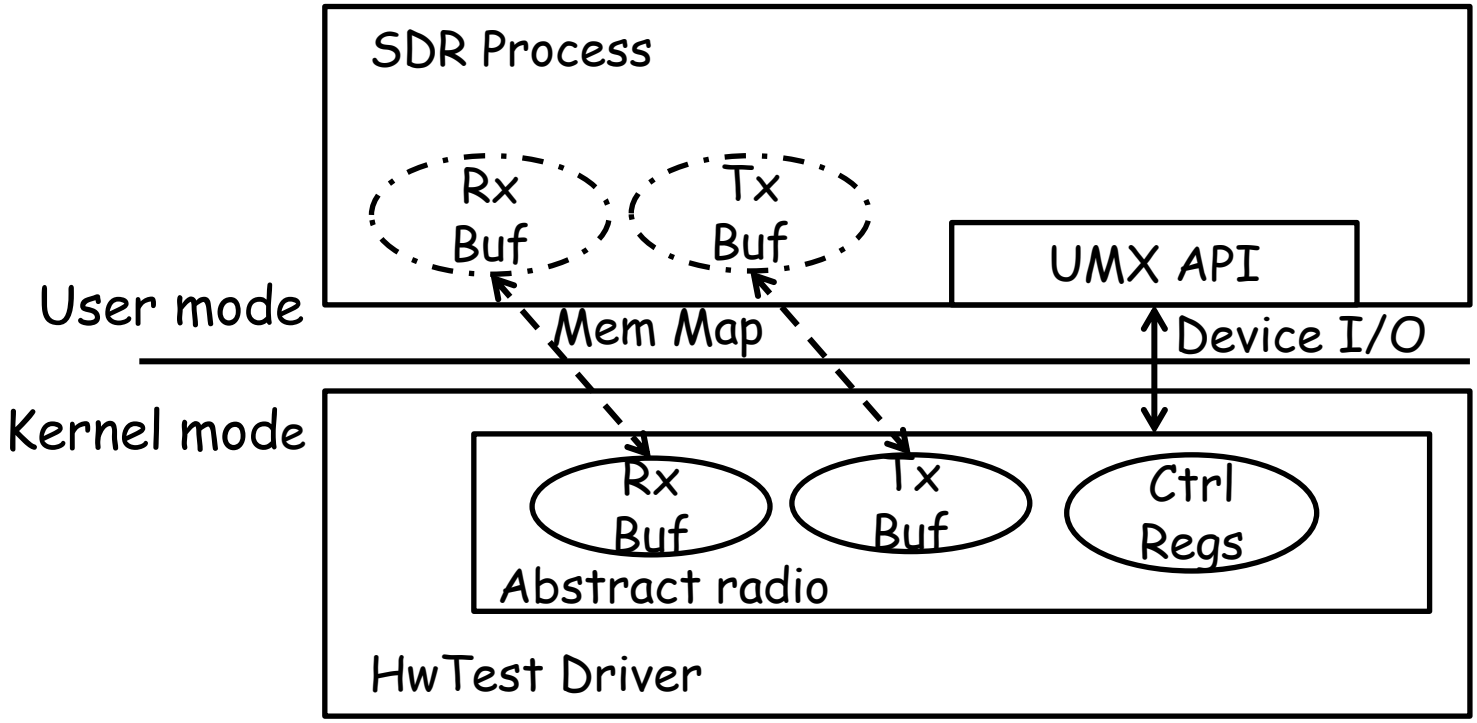
- Sora ver 1.7 (Mar 2012)
  - 802.11abg Brick sample
- Sora ver 2.0
  - MIMO support
  - 802.11n sample

Software Architecture in Sora 1.5



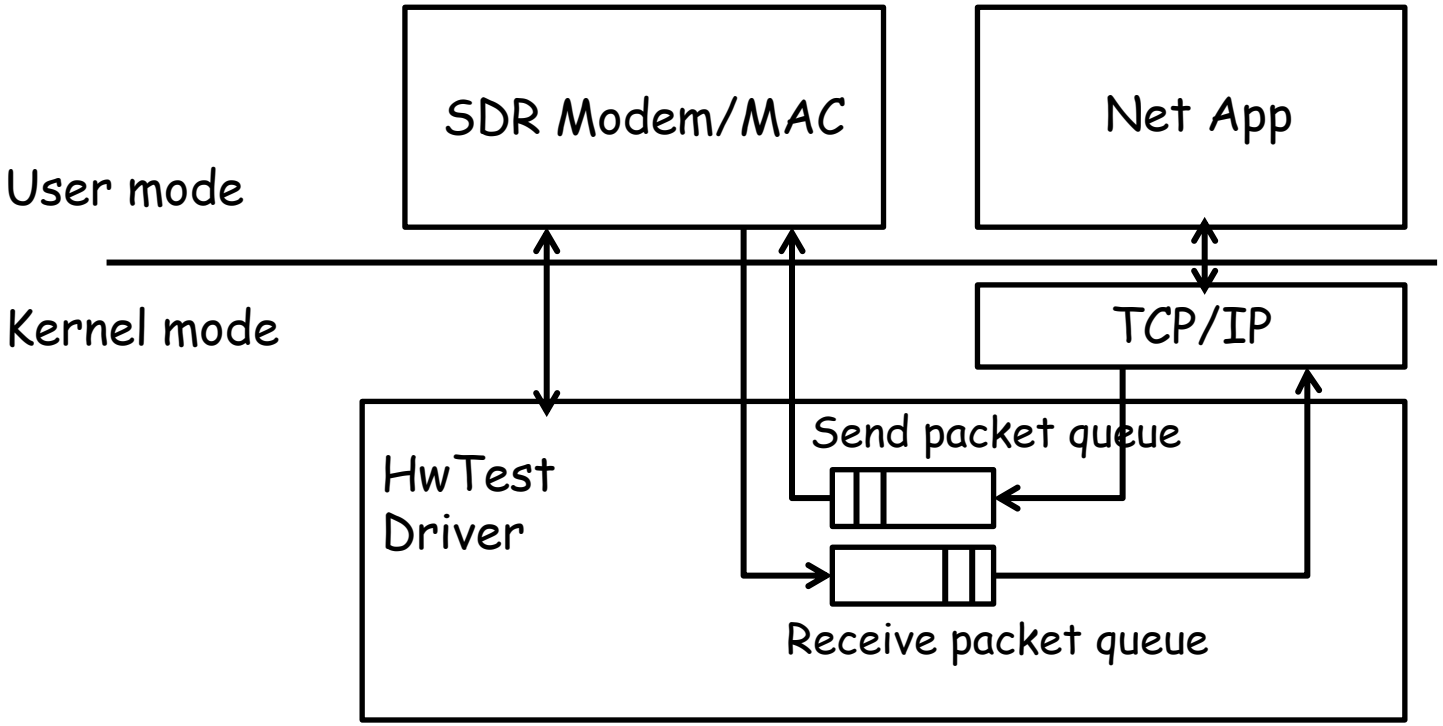
UMX Architecture

- Direct memory mapping for data path
  - High performance/zero copy
- Device I/O for control path
  - Protection
- User-mode Sora thread library for realtime



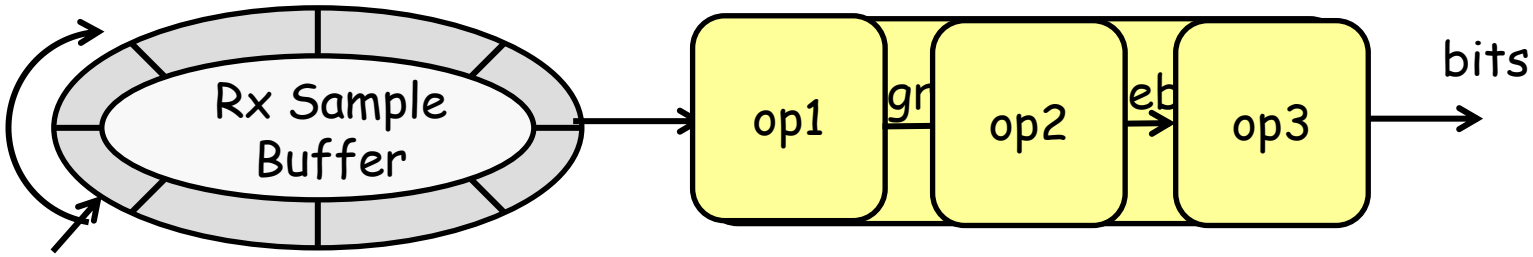
UMX Reflection

- Integrate UMX app into network stack
- Will be shipped in Sora Ver 1.6



Brick Library

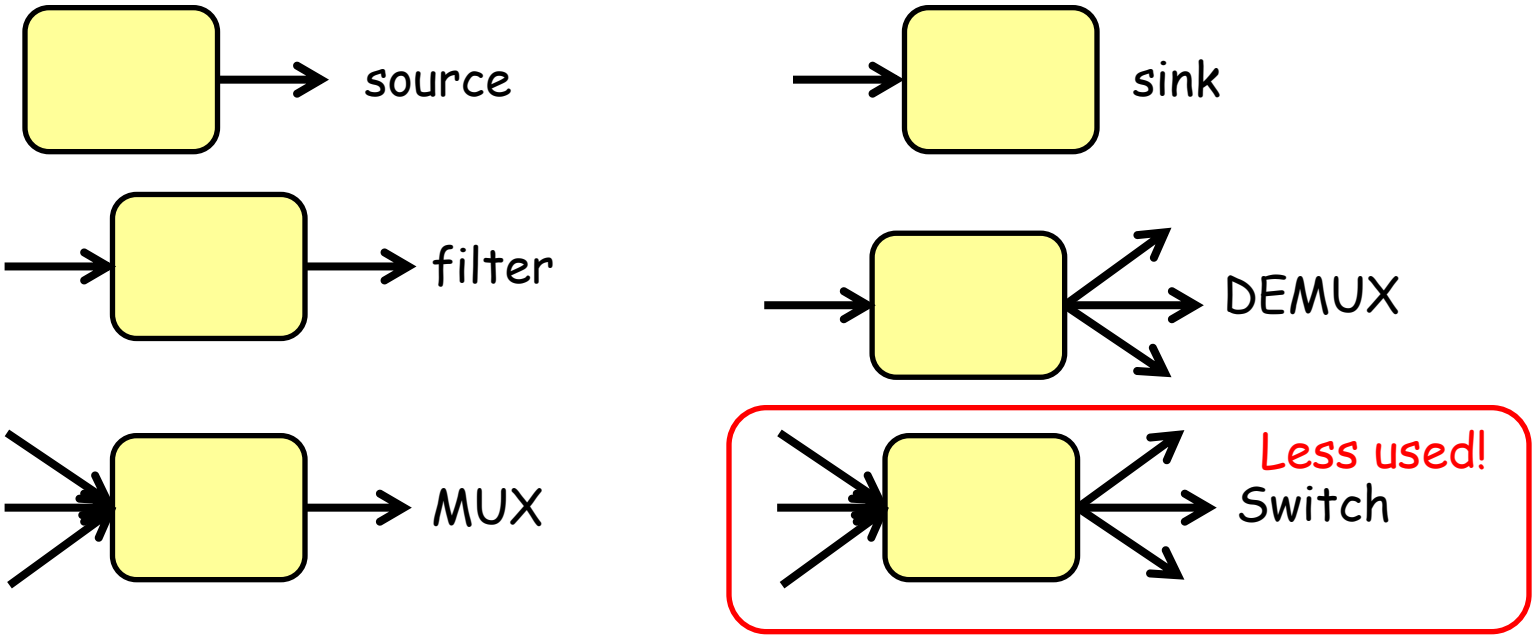
- A modular software architecture for DSP programs



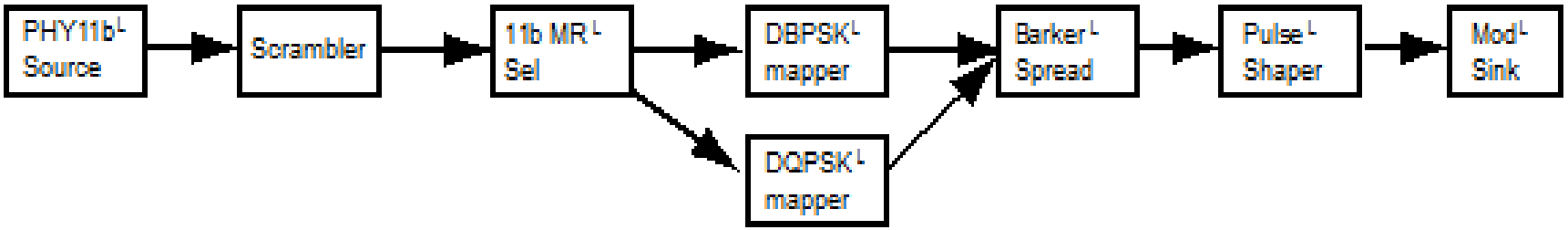
- Design goals
  - Modularize and flexible
  - Better code reuse
  - High performance
    - Introduce minimal overhead

Brick Model

- A brick is a basic functional module



- Mode DSP program as a directed graph

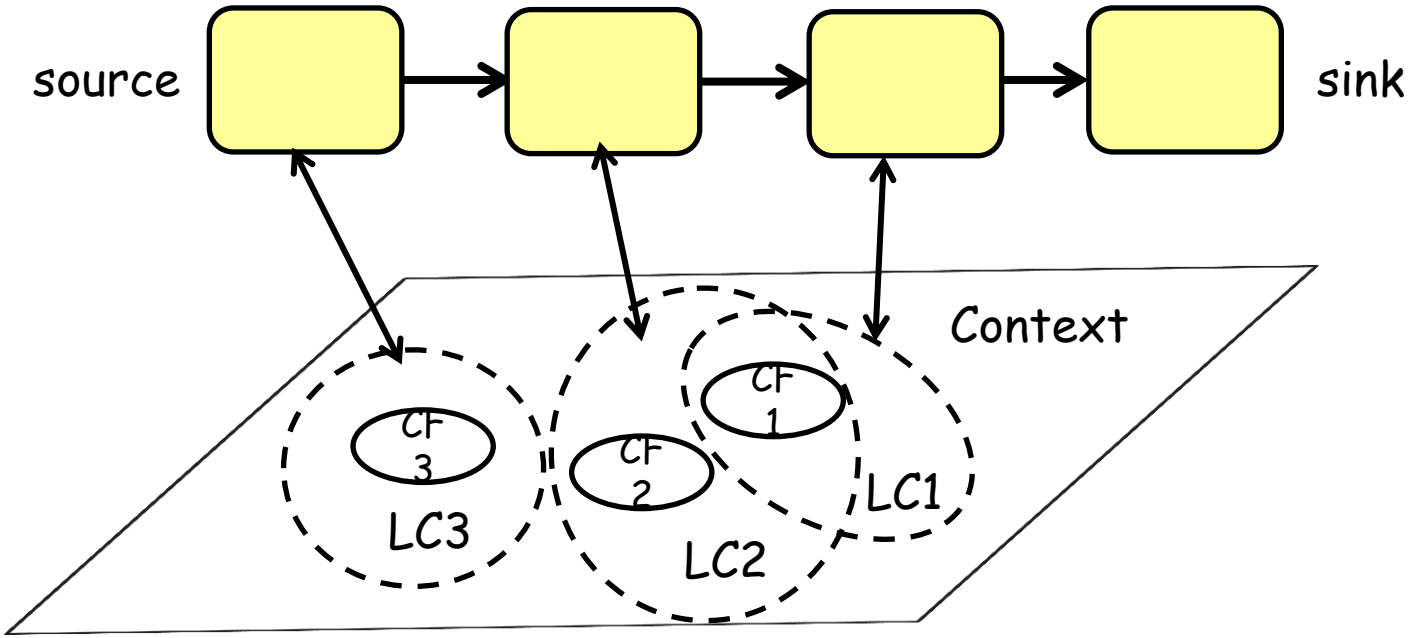


Define a Brick

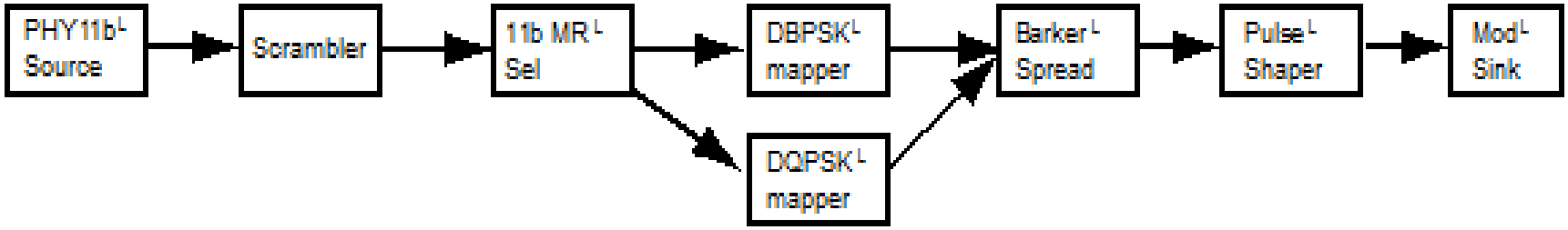
- **Ports**
  - Data type to read and write
- **Interface**
  - **Process**: the main method called to operate on input data and write results to outport
  - **Reset**: reinitialize the module
  - **Flush**: end a data stream
- **Context binding**
  - Access shared state variables

Brick Context Hierarchy

- A mechanism to enable sharing states among bricks in a processing graph
  - Ex. the decoding stage; channel coefficients



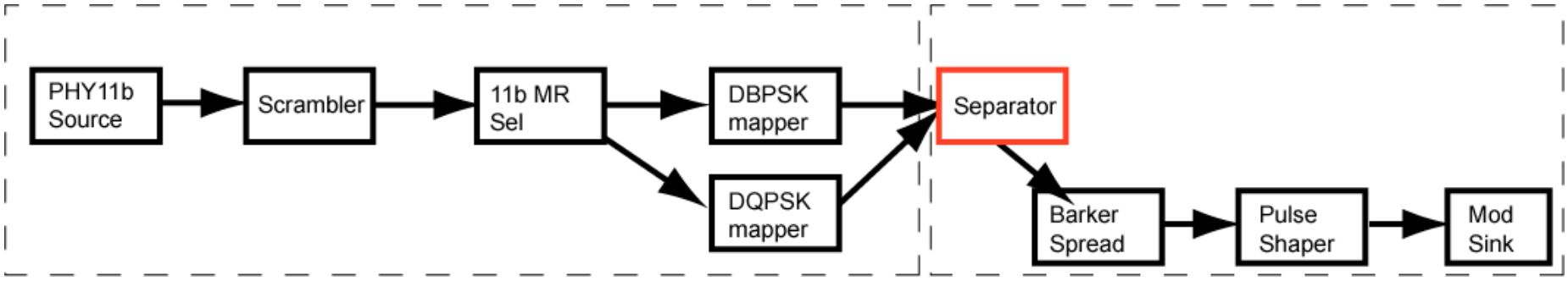
Write a SDR program




```
ISource* CreateModGraph () {
   CREATE_BRICK_SINK    ( modsink, TModSink, fb11bModCtx );
   CREATE_BRICK_FILTER ( pack, TPackSample16to8, fb11bModCtx, modsink );
   CREATE_BRICK_FILTER ( shaper, TQuickPulseShaper,  fb11bModCtx, pack );
   CREATE_BRICK_FILTER ( spread, TBarker11Spread,  fb11bModCtx, shaper);
   CREATE_BRICK_FILTER ( bpsk, TDBPSKMap,  fb11bModCtx, spread);
   CREATE_BRICK_FILTER ( qpsk, TQBPSKMap,  fb11bModCtx, spread);
   CREATE_BRICK_DEMUX2 ( mrsel, T11bMRSel, fb11bModCtx,  bpsk, qpsk );
   CREATE_BRICK_FILTER   ( sc741, TSc741, fb11bModCtx, mrsel  );
   CREATE_BRICK_SOURCE ( ssrc, TB11bPHYSrc, fb11bModCtx, sc741 );
   return ssrc;
}
```

Previously requires 6000+ lines of code!

Multi-threading in Brick graph



CREATE_BRICK_FILTER (spread, TBarker11Spread, fb11bModCtx, shaper);

CREATE_BRICK_SEPARATOR (sep, Tseparator<COMPLEX16,1>, fb11bModCtx, spread);

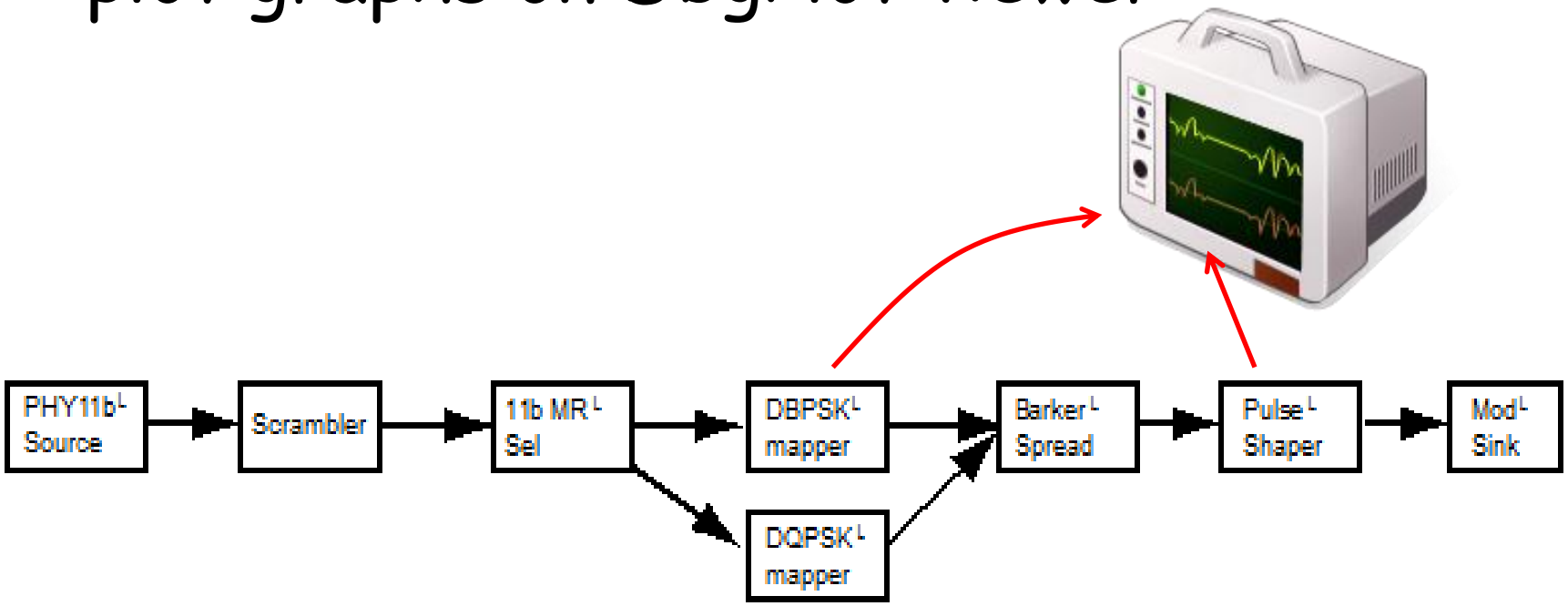
CREATE_BRICK_FILTER (bpsk, TDBPSKMap, fb11bModCtx, sep);
CREATE_BRICK_FILTER (qpsk, TQBPSKMap, fb11bModCtx, sep);

Brick implementation

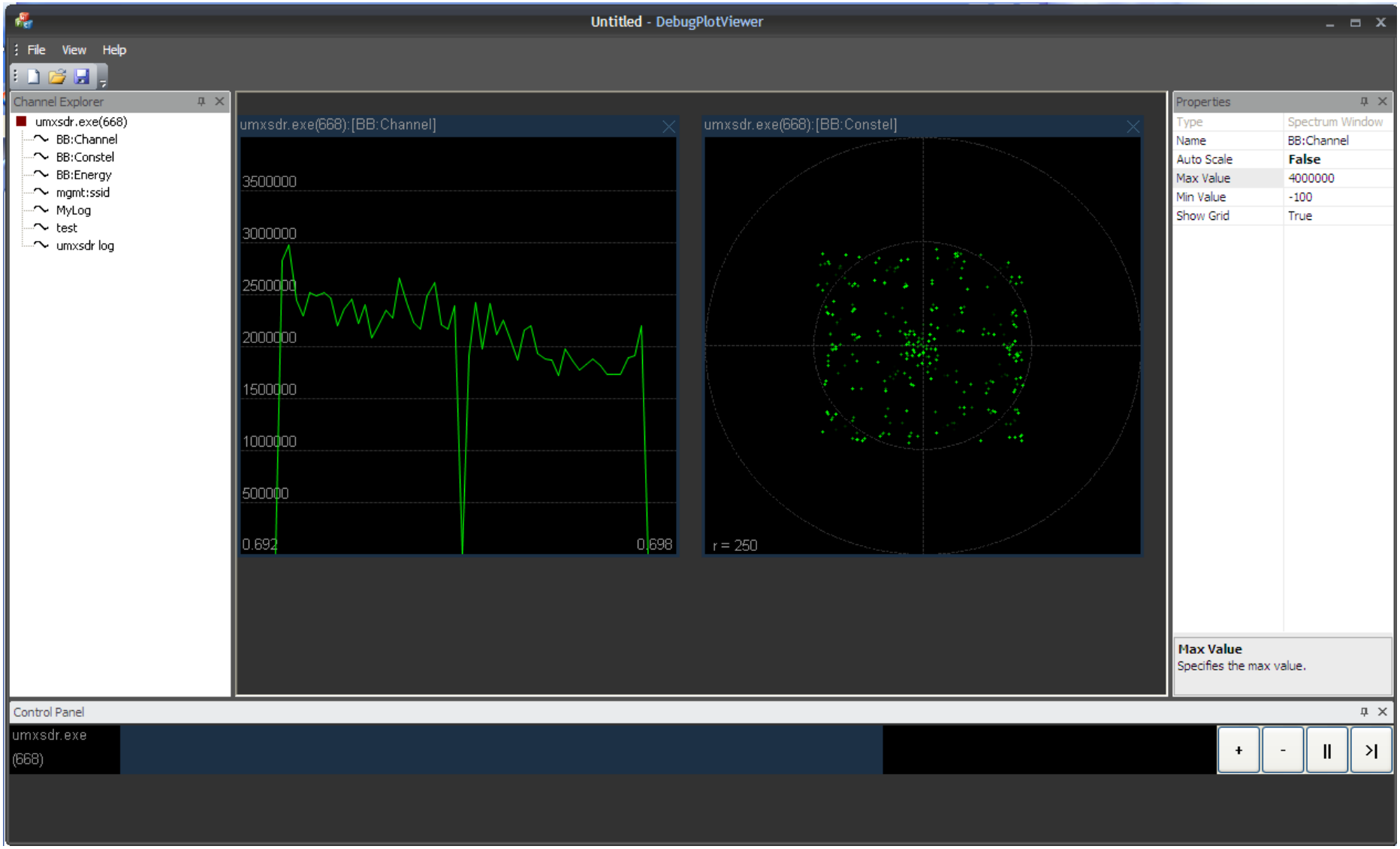
- C++ template library, instead of C++ virtual function
  - Enable inline function optimization
- Avoid function call overhead
- Enable compiler optimization at compile time

DebugPlot

- A graphic tool to help debug/monitor the real-time SDR programs
- Easy-to-use and light-weight APIs to plot graphs on DbgPlot viewer



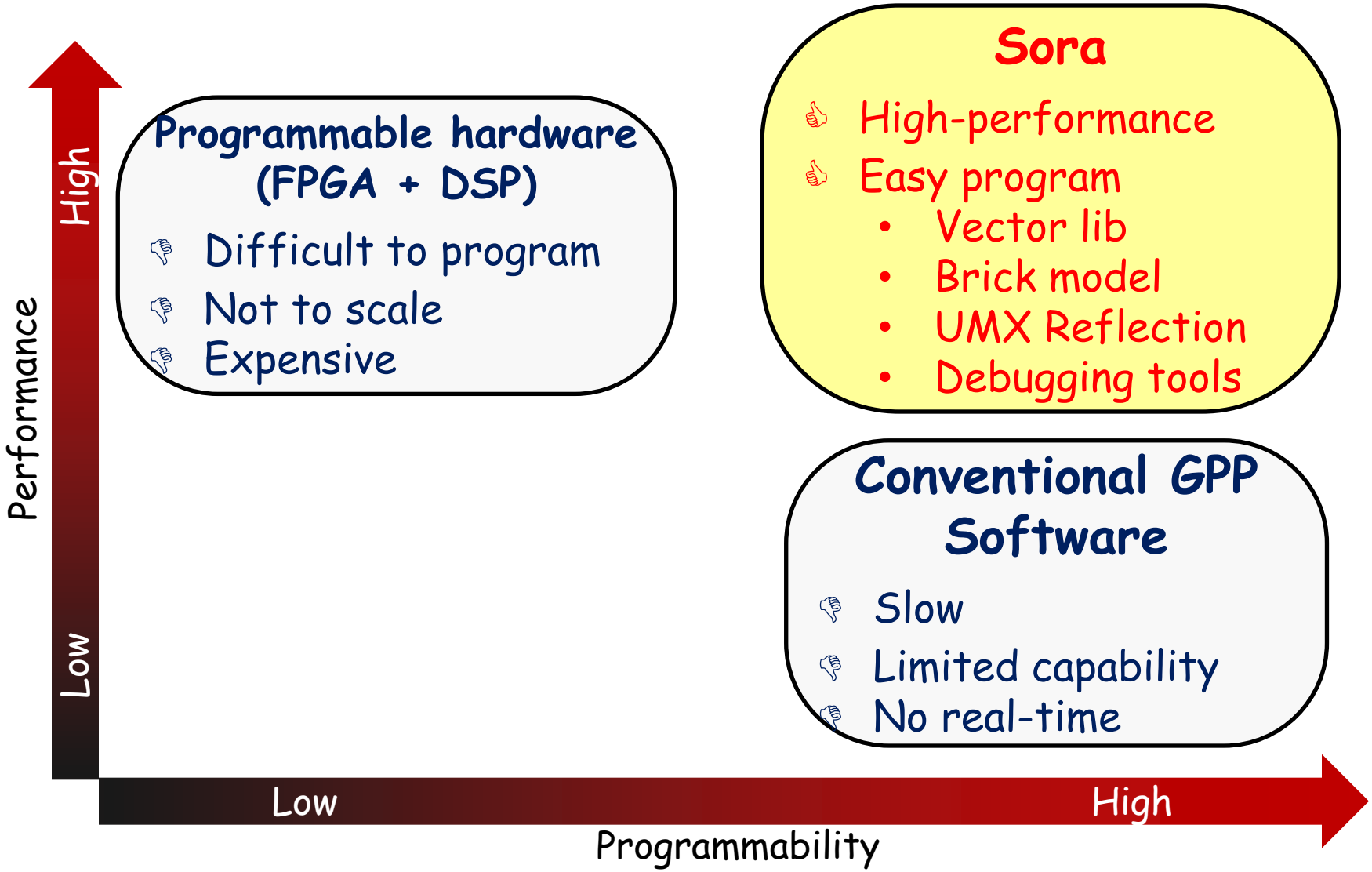
DebugPlot (cont.)



Sora MIMO Kit and Whitespace

- MIMO Kit
  - External radio unit with PCIe cable extension
  - 4x4 MIMO, 20/40MHz channel
  - Extensible to 8x8
- Whitespace front-end

Summary


Performance
High
Low
Programmable hardware (FPGA + DSP)
Difficult to program
Not to scale
Expensive
Sora
High-performance
Easy program
Vector lib
Brick model
UMX Reflection
Debugging tools
Conventional GPP Software
Slow
Limited capability
No real-time
Low
High
Programmability

Thanks! Questions?

Coming to Sora demo this afternoon…