Computation Wireless Communications

Kun Tan
Wireless and Networking Group
MSR Asia

2010 Summer

Agenda

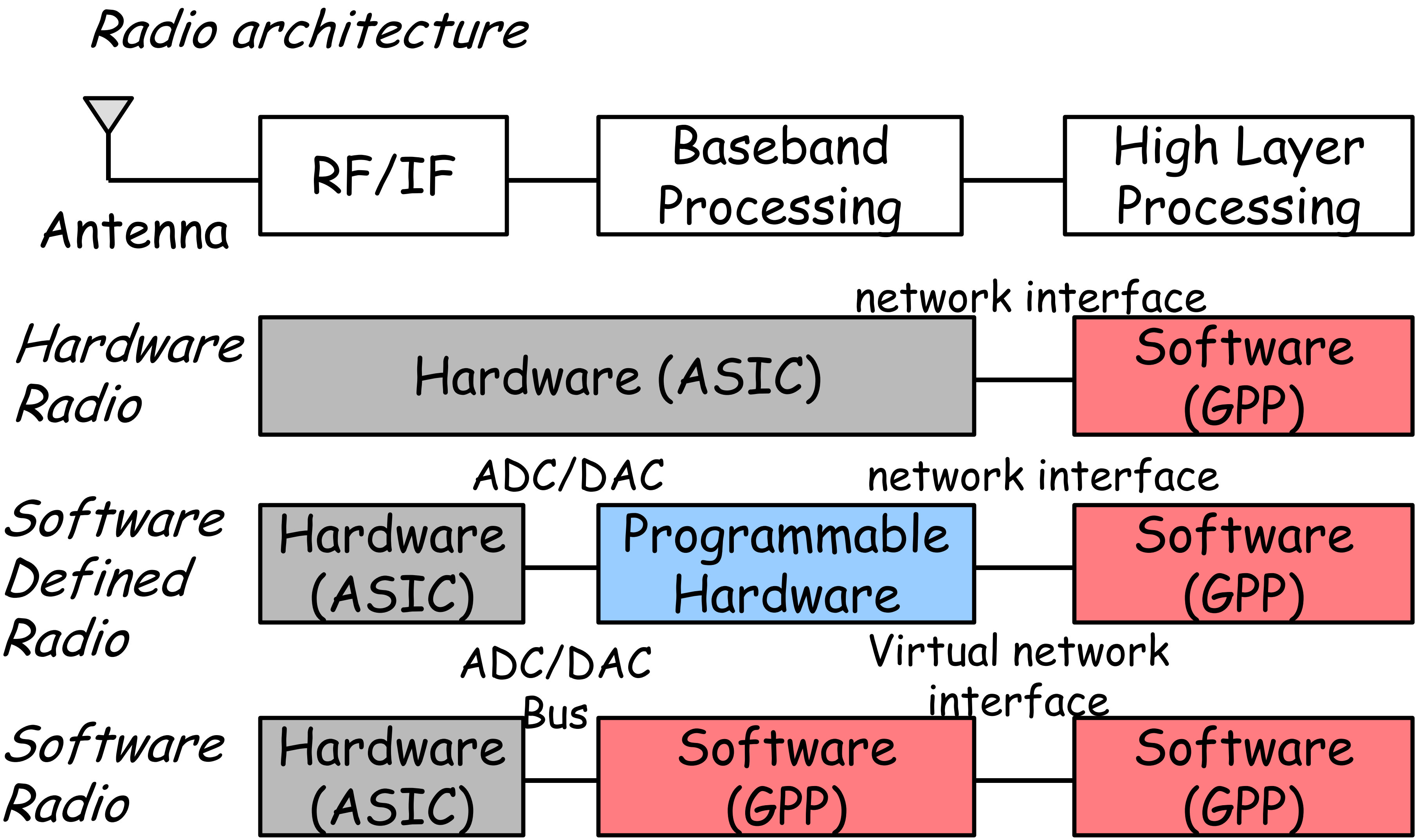
- Software (Defined) Radio
  - Basic concept, Architecture and System
- Computational Thinking in Wireless Research
  - Examples from three SIGCOMM papers
- Sora Tutorial
  - Platform, tools and programming
- Digital Wireless Communication Illustrated (optional)
  - Case study of IEEE 802.11

Software Radio
Basic concept, Architecture and System

What is Software (Defined) Radio

- Historically defined by Mitola (1992)
  - Radio's physical layer behavior is primarily defined in software
  - Accepts fully programmable data and control traffic
  - Supports broad range of frequencies, air interfaces (protocol), and applications
  - Be able to dynamic changes its configuration according to user requirements

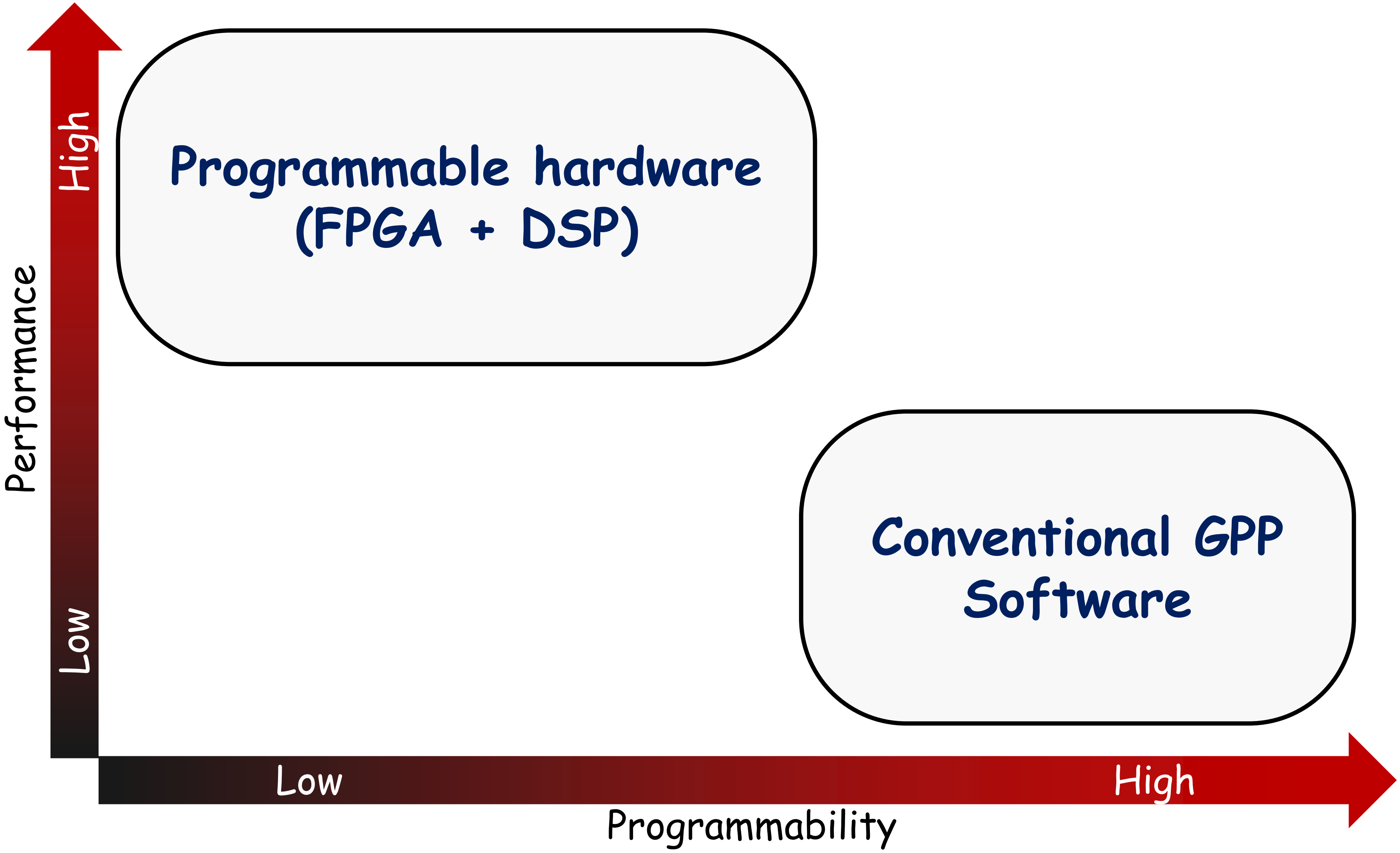
From Hardware to Software



Software (Defined) Radio Benefit

- Flexibility and programmability
- Reduced research-product-market cycle
- Scalability to potential large systems
- Open architecture
- Reduce the cost for customized silicon
- Better performance/market trajectory
- Enhanced maintainability

Approaches

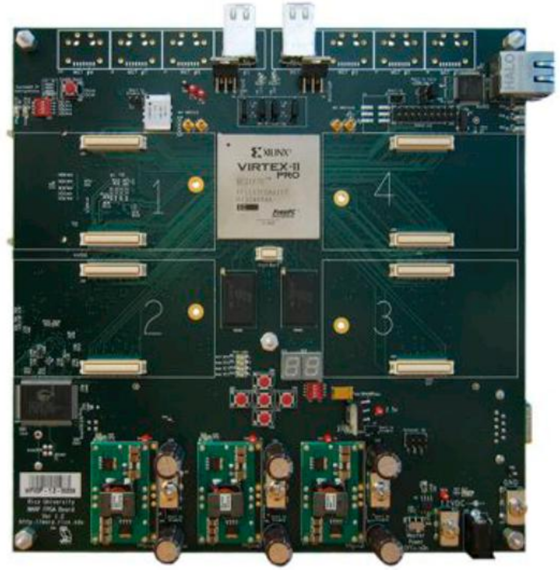


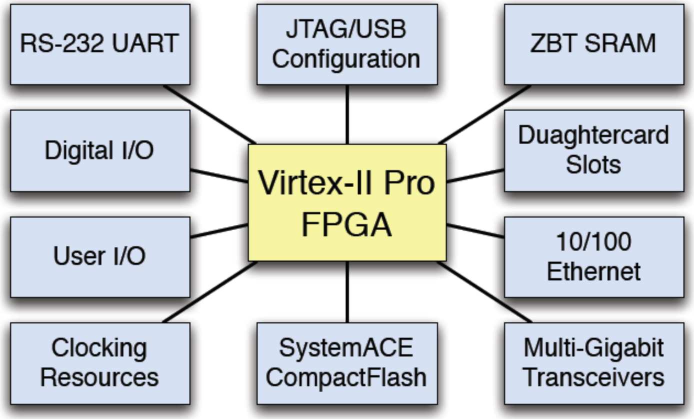
Programmable hardware (FPGA + DSP)

Conventional GPP Software

WARP Platform (Rice University)

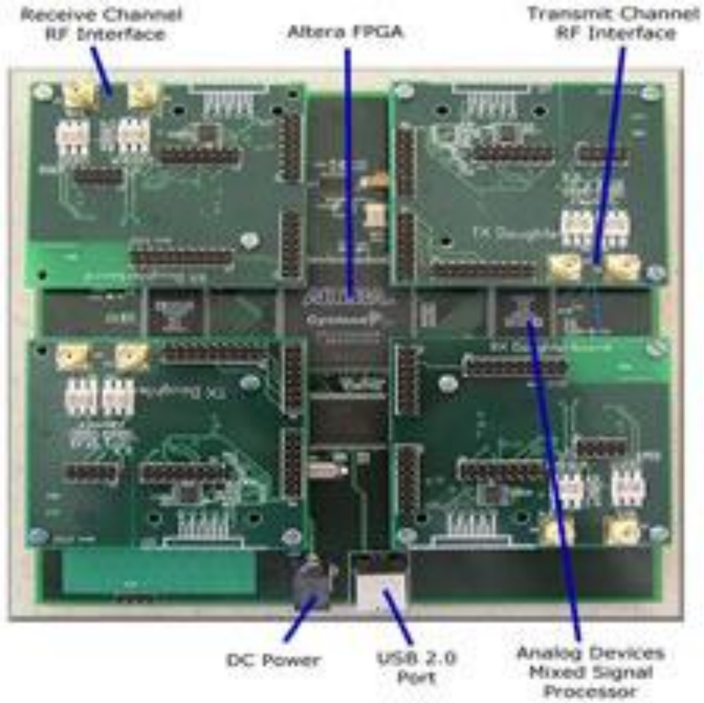
- Xilinx XC2VP70 FPGA for DSP
  - With 2 embedded PowerPC processor
- Low-level programming on hardware logic (Verilog)
- Matlab + sysgen
- Limited by single chip FPGA, difficult to scale
- High price (~$10,000USD)
- Promised good performance, but…
  - No sophisticated PHY (e.g. 802.11 or 3G/4G)
  - Limited achievable network throughput demoed (~Mbps)



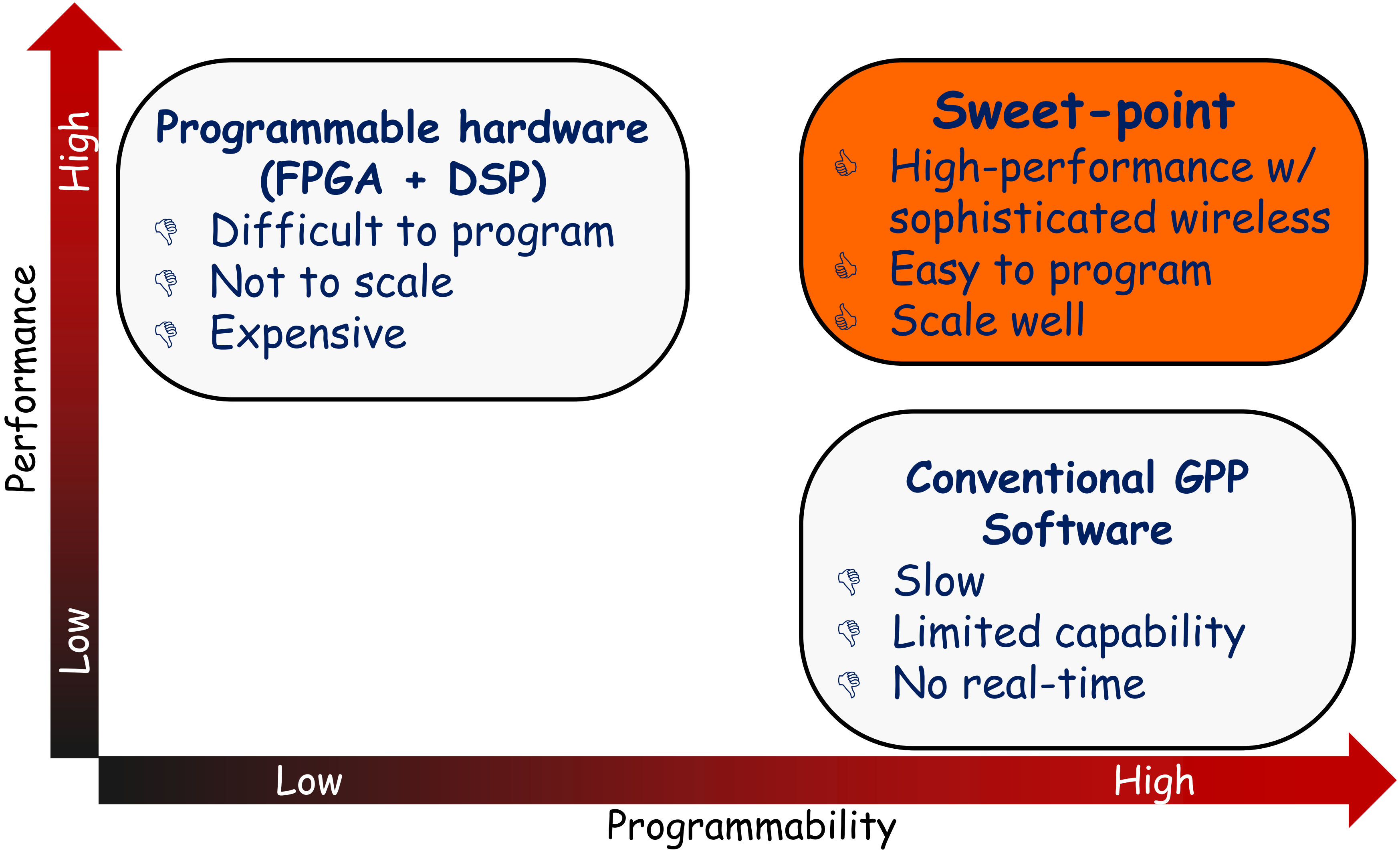


GNURadio/USRP

- All DSP is performed on CPU
- Modular design
  - C/C++ and Python script
- Slow interface to PC
  - v1: USB2.0
  - v2: Gigabit Ethernet
- Large latency (ms)
- Software is not optimized
  - Large overhead
  - Not ready for multiple core
- Limited performance
  - Achievable wireless throughput is low


Receive Channel RF Interface
Altera FPGA
Transmit Channel RF Interface
DC Power
USB 2.0 Port
Analog Devices Mixed Signal Processor

Approaches



Programmable hardware (FPGA + DSP)

- 👎 Difficult to program
- 👎 Not to scale
- 👎 Expensive

Sweet-point

- 👍 High-performance w/ sophisticated wireless
- 👍 Easy to program
- 👍 Scale well

Conventional GPP Software

- 👎 Slow
- 👎 Limited capability
- 👎 No real-time

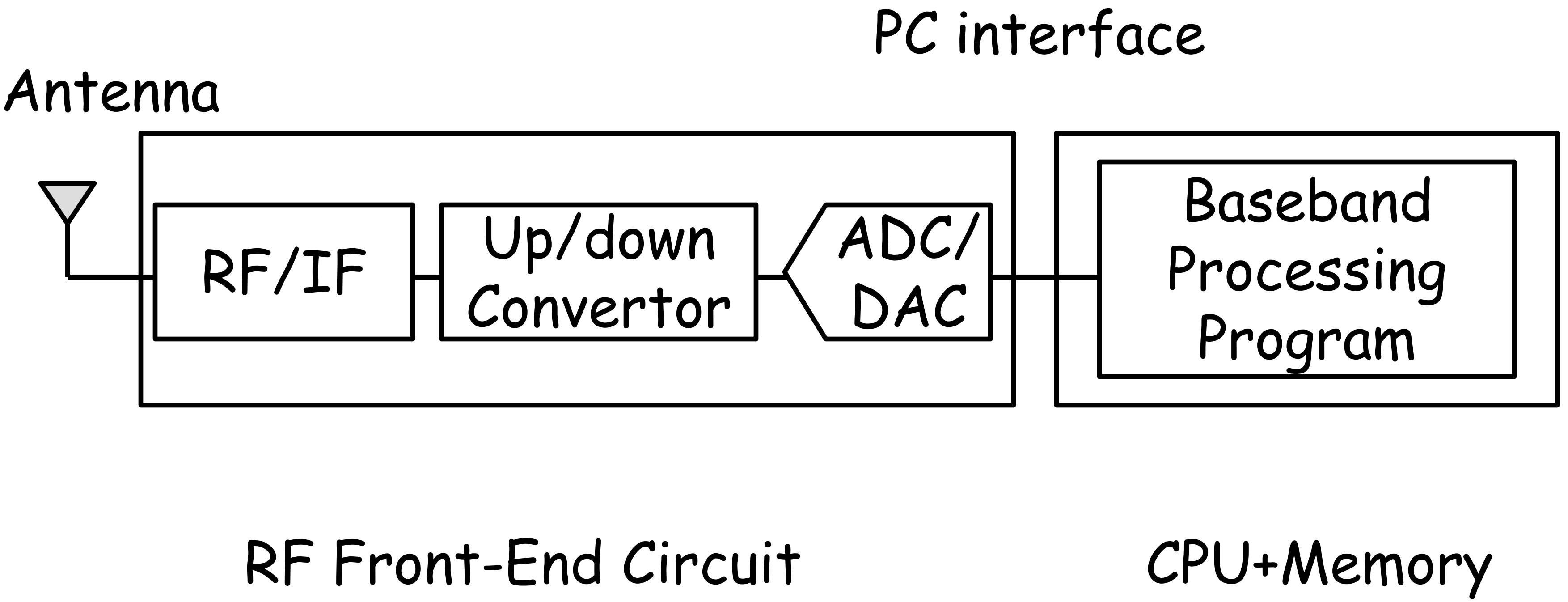
Programmable Hardware vs. GPP

	General Purpose Processor	Programmable Hardware
Programmability & Flexibility	Programming in high-level language and abstraction Mature tools for programming and debugging	Low level programming Limited programming tools
Open architecture	Standard instruction set and architecture Portability	Specialized or proprietary architecture Limited interoperability
Scalability	Loosely coupled Scale with the state-of-the-art computer system	Tightly coupled embedded design Difficult to scale

Programmable Hardware vs. GPP (cont.)

	General Purpose Processor	**Programmable Hardware**
Trajectory	Quicker evolution/ better optimization driven by large market Moore's law continues in multi-core trend	Isolated specialized market
Whole system price	Performance/$ decays exponentially	Hold still mostly, though computing components' price reduced
Performance for DSP	Mainly designed for general computing tasks, not for specific DSP How well can we do?	Satisfy the requirement with great engineering efforts

Architecture of a GPP-based SDR System



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

How Can We Meet the Requirements?

- Take advantage of modern PC bus technologies
  - Driven by high-speed interconnection among PC subsystems
  - PCIe – 2Gbps per lane and up to 64Gbps w/ x32; standard in current PCs
  - PCIeV2 – 5Gbps per lane, up to 128Gbps w/ x32;
  - Up coming PCIeV3 (256Gbps) and Silicon Photonics (billion bps promised)
- Ride the wave of multi-core technology
  - Sustain Moore's law when CPU hits the heat wall
  - Four core processing is standard; six/eight cores for high-end configuration
  - 64 and more cores are coming
  - Software innovation to unlatch the power of parallel programming

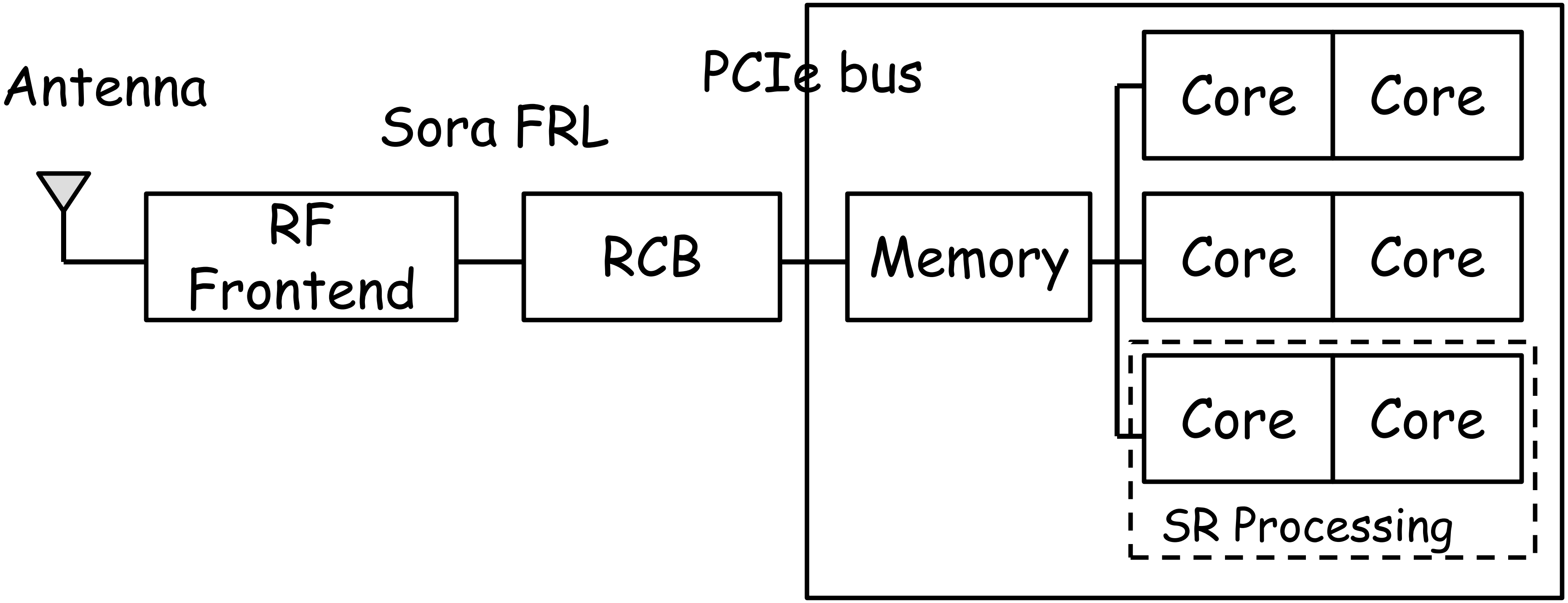
The Sora Approach

- New hardware interconnection board based on PCIe

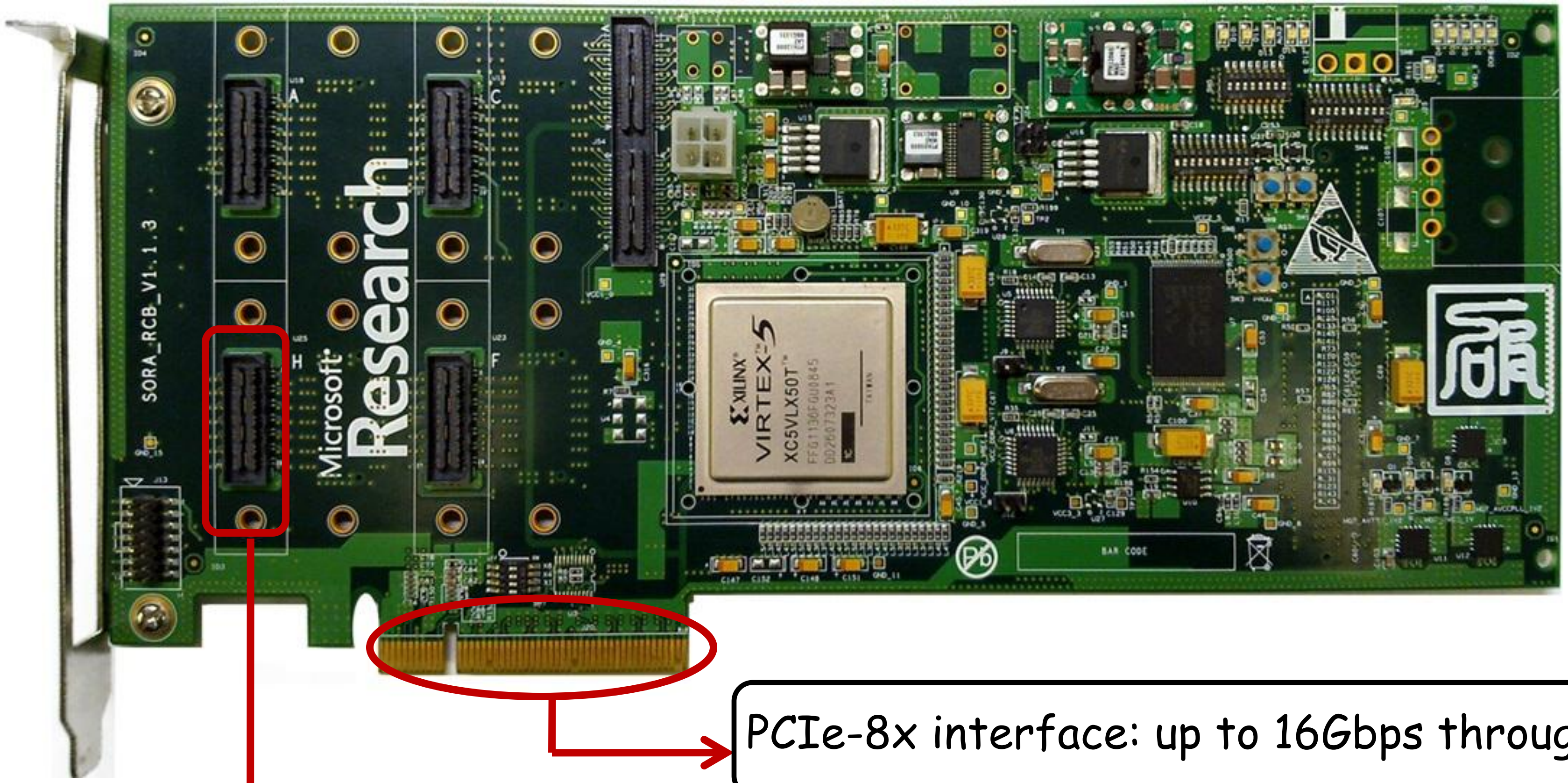
- High performance PHY implementation on multi-core
  - Trade memory for computation
  - Exploit data parallel with SIMD
  - Streamline across multiple cores
  - Scale with the ever increased cores

- Core dedication for real-time support
  - A dumb idea who age comes

Sora Architecture



Sora Radio Control Board

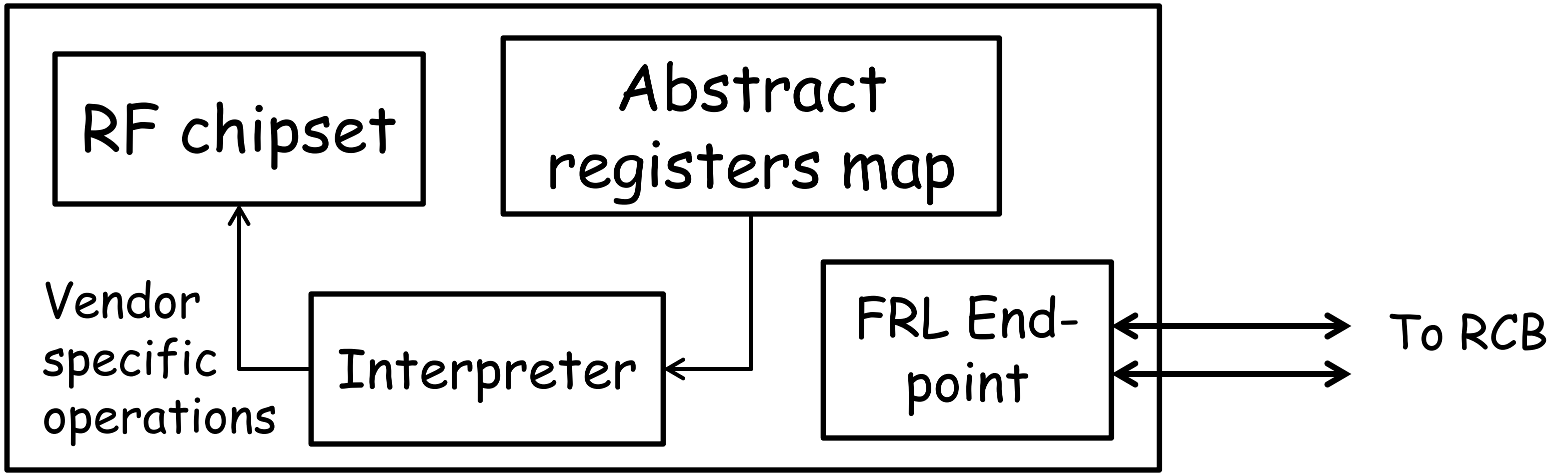


PCIe-8x interface: up to 16Gbps throughput

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

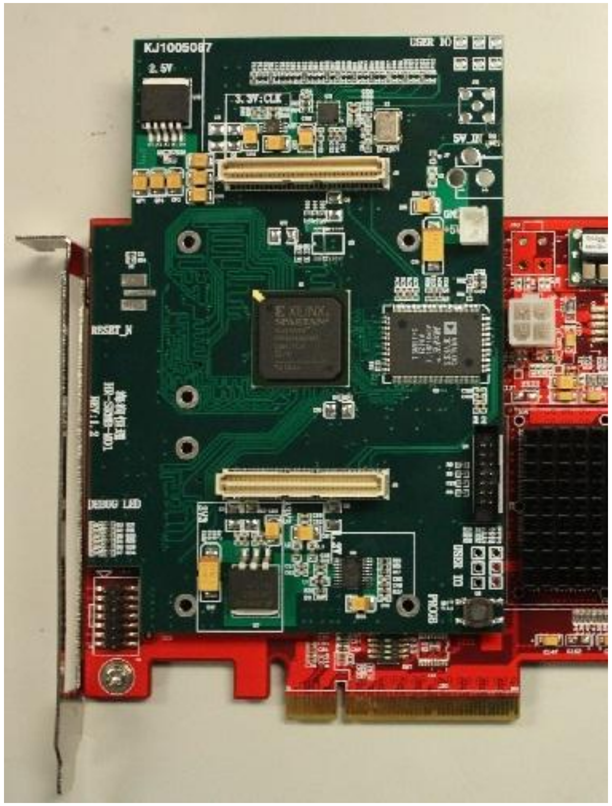
Sora Fast Radio Link

- An abstract protocol to transfer radio control information and digital samples between RF front-end and baseband
  - Abstract registers for RF control and status information
  - Common I/Q sample format
- Enable interoperation between any baseband and any RF hardware

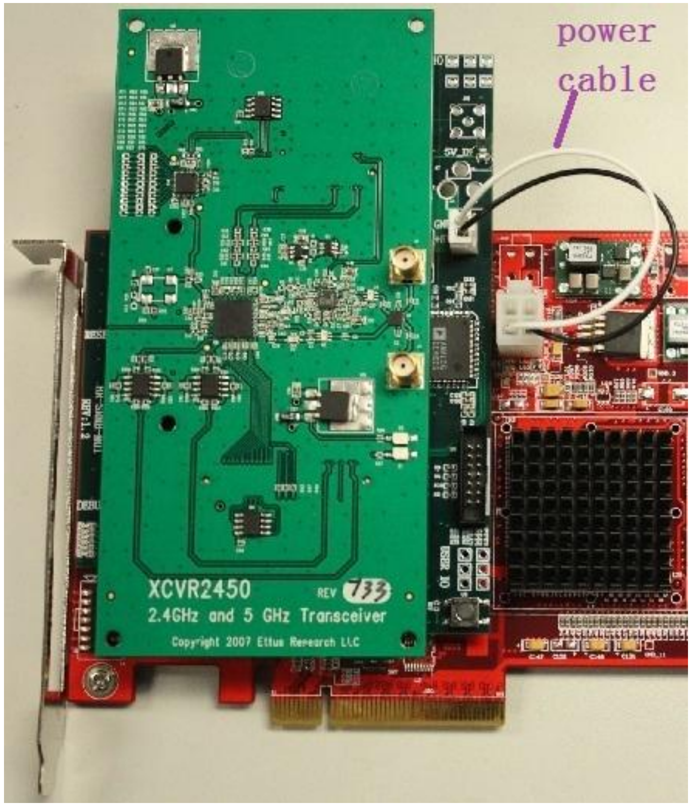


Connect to a RF front-end

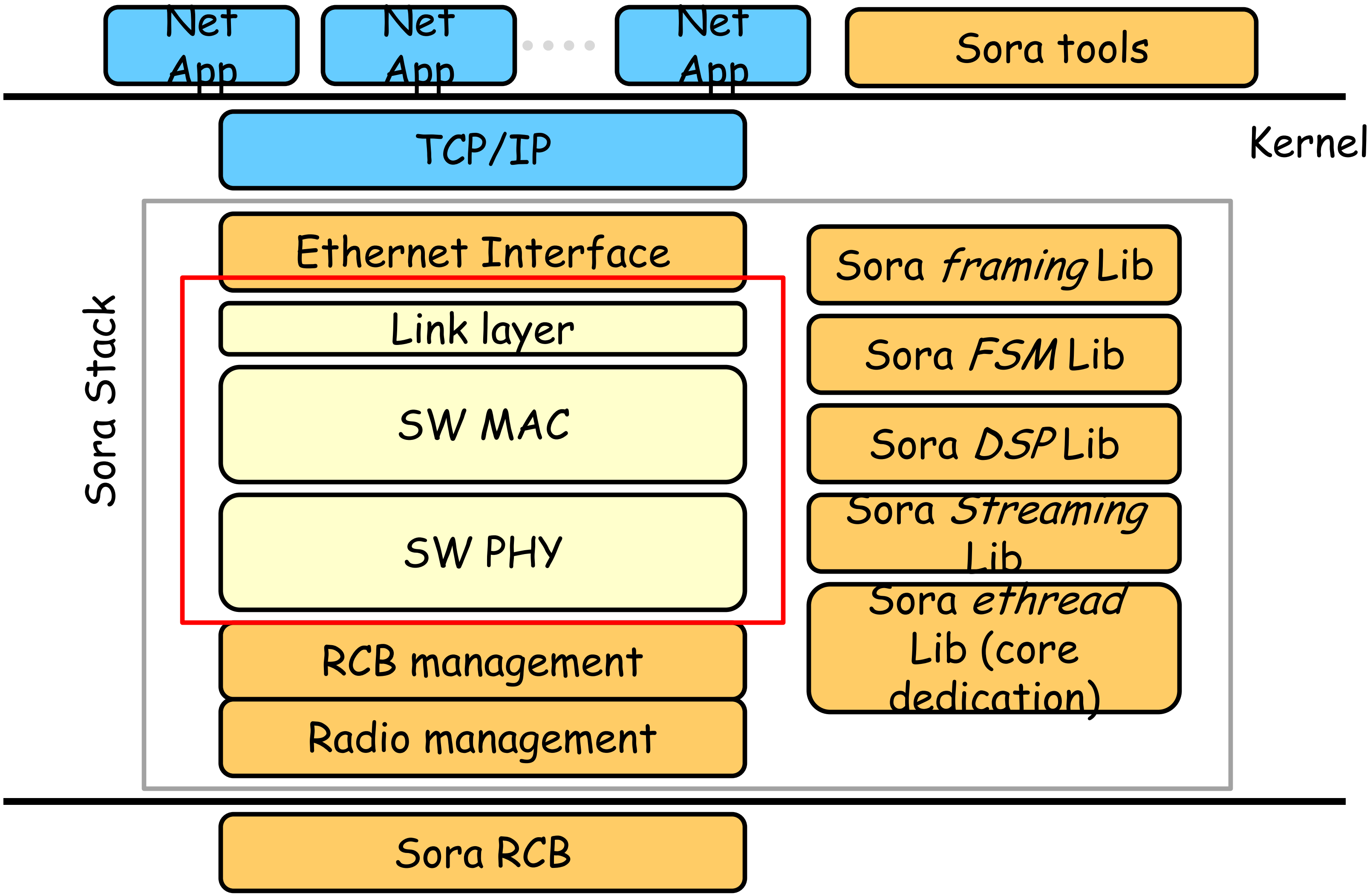
Convertor board implements Sora FRL



Connect to a 2.4GHz RF front-end board

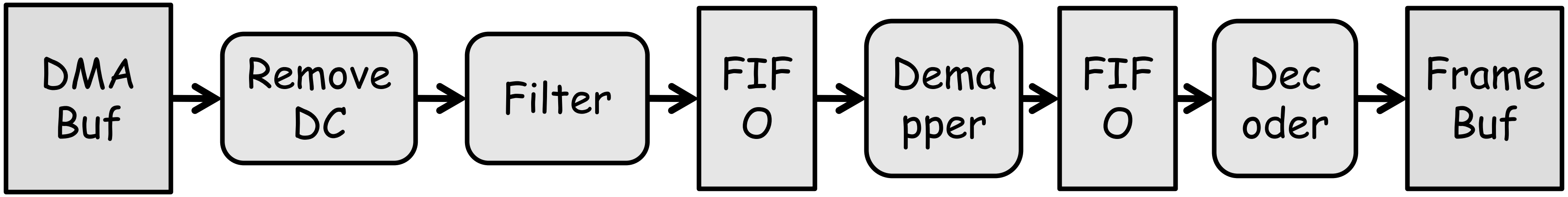

power
cable
XCVR2450
REV 733
2.4GHz and 5 GHz Transceiver
Copyright 2007 Ettus Research LLC

Sora Software Architecture



Programming PHY

- Basic model – a pipeline of processing block

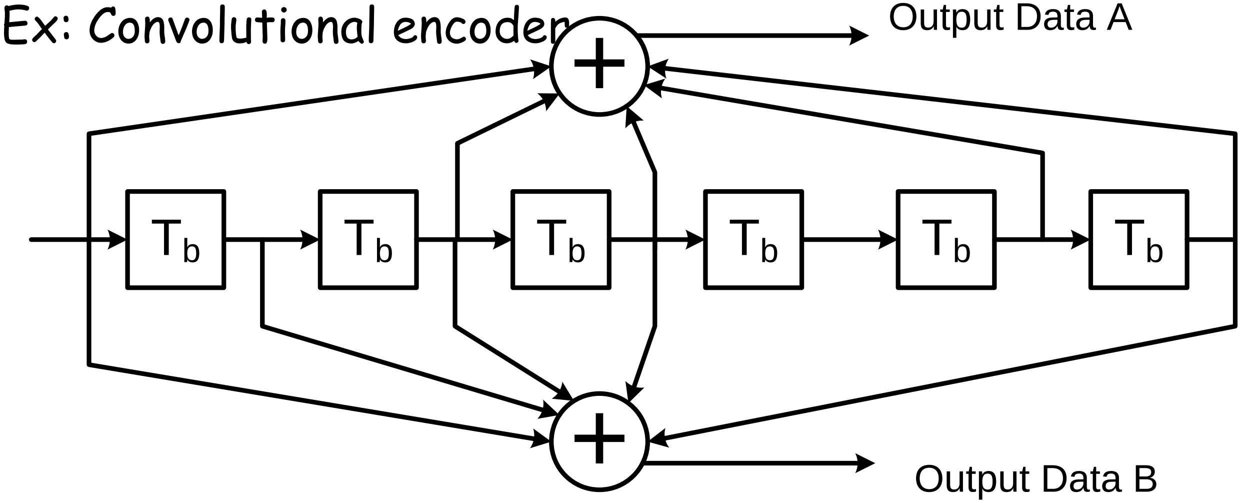


- Key design questions
  - What is the right implementation tradeoff?
  - How to exploit the parallelism in PHY and scale with multiple cores?
  - How to schedule the execution of processing blocks?
  - How to realize real-time services?

Key Design Choice One: 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

Ex: Convolutional encoder



Direct impl. 8 ops per bit

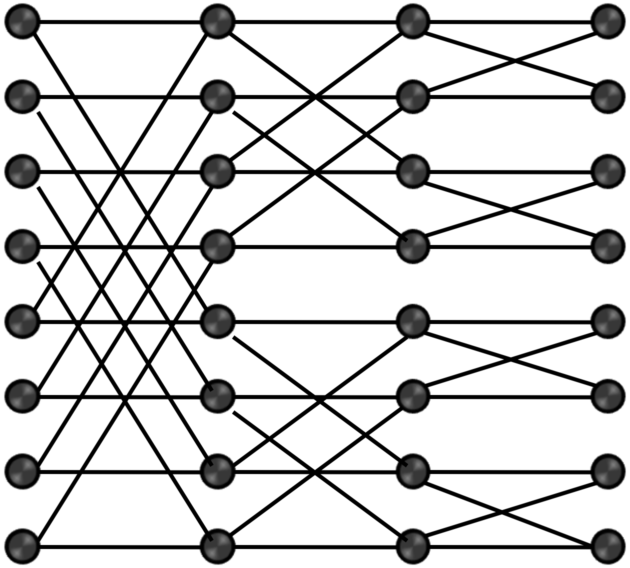
LUT impl. 2 Look-up op for 8 bits!
(size 32KB)

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

Key Design Choice Two: Exploit Data Parallelism with SIMD

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

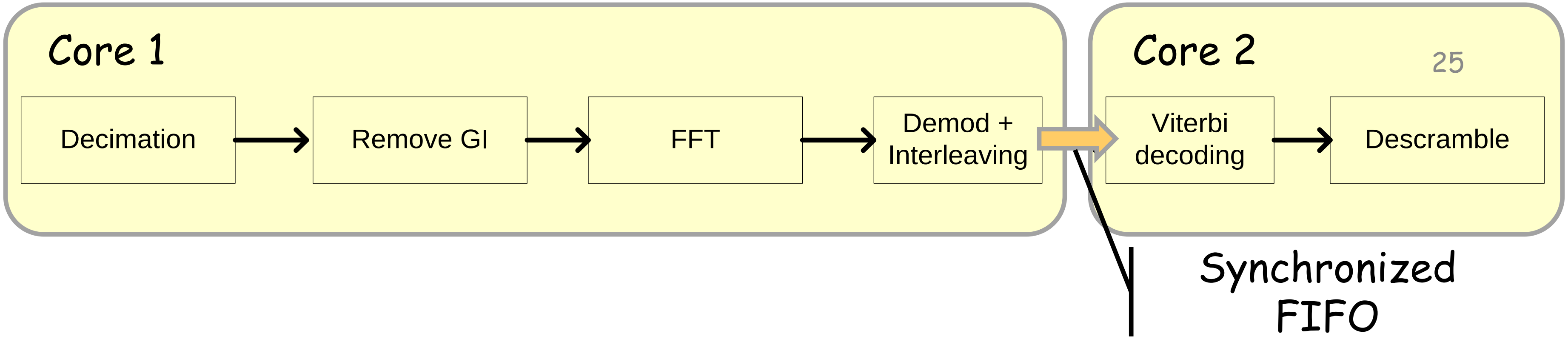
Ex. (I)FFT



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

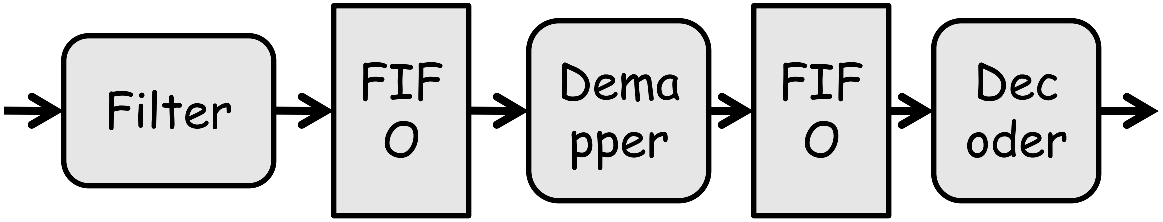
Key Design Choice Three: Pipeline across Cores

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



Key Design Choice Four: Static Scheduling

- PHY pipeline is synchronized
  - Behavior of each processing block is pre-deterministic
  - Compute a lock-free schedule at the compile time
- PHY processing block is very fine granulate
- Benefit over multi-thread model
  - No context switching overhead
  - No need to check execution condition
  - Avoid cache flushing/missing
  - Mitigate synchronization overhead



Key Design Choice Five: Core Dedication for Realtime

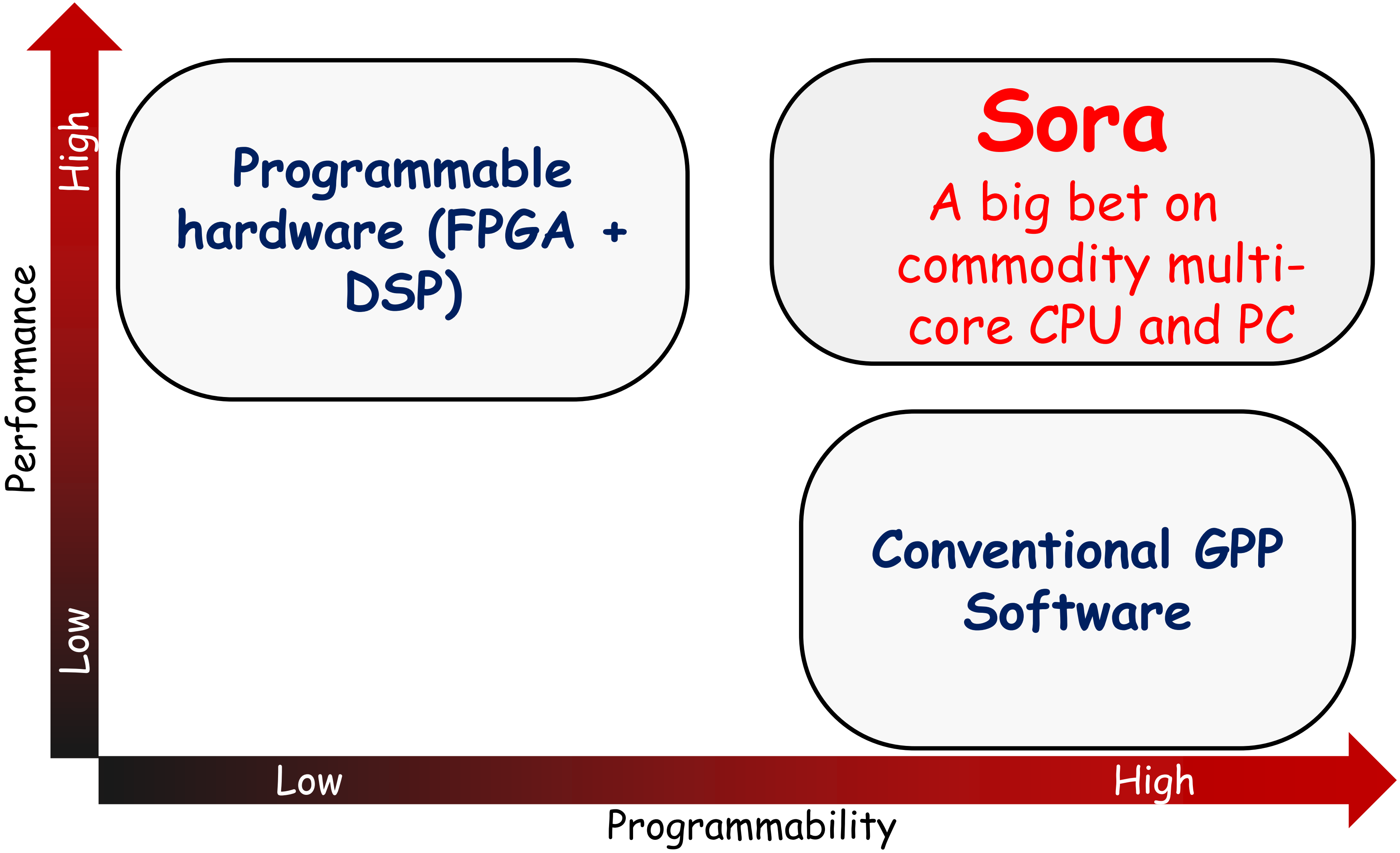
- Software is considered “uncertain” in traditional OS
  - Why?
  - Multiplexed with multiple tasks/processes/threads
  - Contention in memory/bus
  - Interrupts
  - RTOS – complex/overhead/limited functionality
- Core dedication – a dumb idea for multi-core
  - Exclusively allocate enough cores for RT tasks
  - Guaranteed resource
    - CPU/Cache/Memory
  - 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 ☺

Put Them Together

- Turn a commodity multi-core Windows PC into a powerful software radio
- Orders of magnitude better than prior art

- Demonstrated capabilities
  - First WiFi (802.11 a/b/g) in pure software
    - Standard compliant up to 54Mbps data rate
    - Interoperable with commercial NIC
  - First LTE (uplink) in pure software running online
    - 43Mbps peak data rate

Sora in the Sweet Point



Programmable hardware (FPGA + DSP)

Sora

A big bet on commodity multi-core CPU and PC

Conventional GPP Software

MSR Software Radio Kits

- For academic non-commercial research
  - Pre-order for RCB hardware
  - Software download online

- More information on Sora
  - http://research.microsoft.com/sora
  - http://research.microsoft.com/en-us/projects/sora/academickit.aspx

Summary - Time for Software Radio Has Come

- History will always repeat itself
  - General computing platforms always win over specialized platforms, when the processing ability of the former exceeds the application requirements
  - Consider multimedia twenty years ago
- Practical applications
  - Test, measurement, monitoring instruments
  - Cellular base-station/Access points
  - Prototype and research
- Challenges remain
  - Power consumption
  - Security
  - Fear of “unknown” of people

Computational Thinking in Wireless Research

Examples from three SIGCOMM papers

What is Computational Thinking?

- It is the fundamental skill for a computer scientist to solve problem
  - Esp, large scale, difficult problems
- It is all originated from the fundamental question of What is computable?
- Computation thinking rests on solid theory, but differs from the mathematical thinking
  - It is about not only to characterize the solution, but also how efficient you can obtain the solution
- Computation thinking relies on excellent engineering, but differs from the engineering thinking
  - It is about not only a good solution, but also the fundamental understanding of the difficulty and solution space

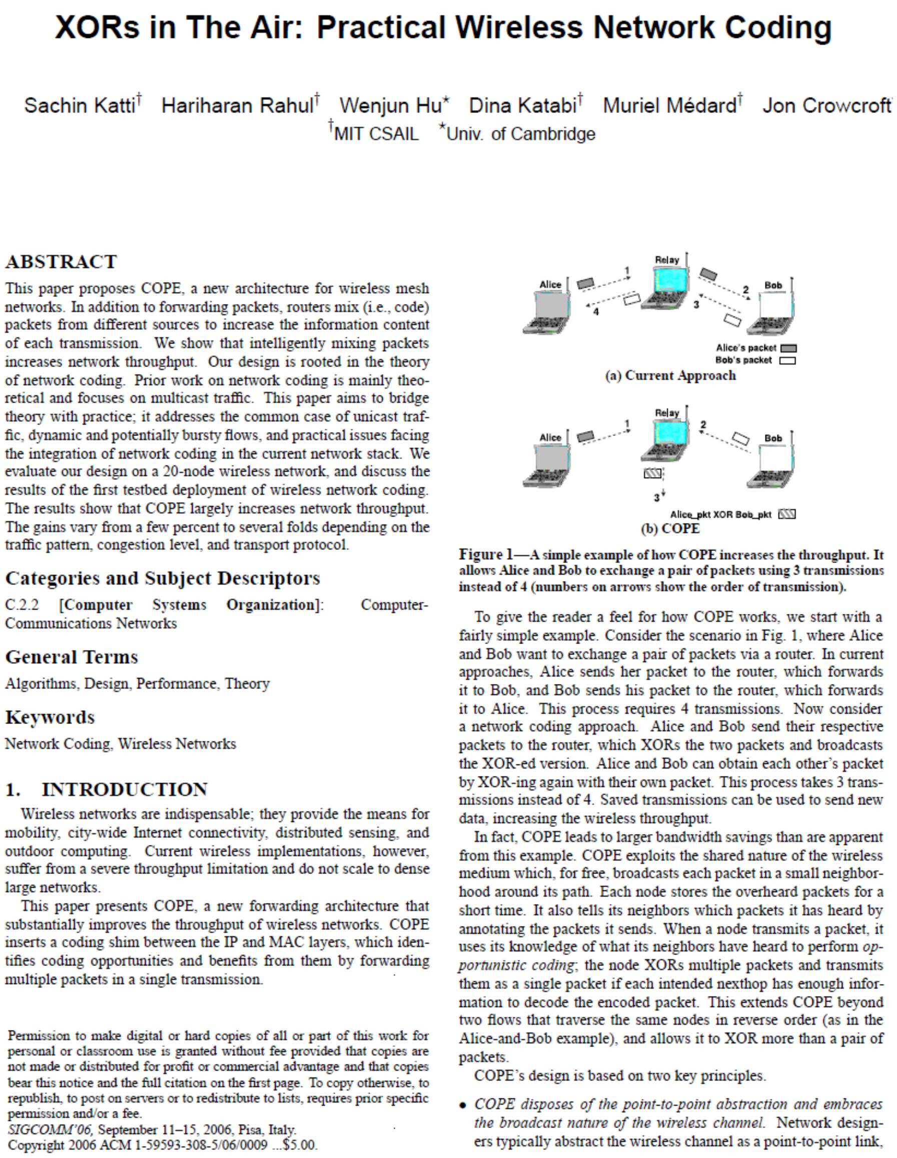
Key Mental Tools for Computational Thinking

- Critical thinking on fundamental tradeoff
  - Why the problem is difficult?
  - What are the fundamental constraints?
  - Whether an approximate solution is good enough?
- Intuitive thinking with heuristic reasoning
  - Getting quick insight on the problem
  - Constructive way to find a solution or a counterexample
- Thinking using abstraction and decomposition
  - Choosing the right representations
  - Separation of concerns
  - Divide-and-conquer

Key Mental Tools for Computational Thinking (cont.)

- Thinking recursively
  - Thinking about layering, indirections, and architecture things with simple rules
  - Recognizing the virtues and the dangers of a concept, an abstraction
  - Understanding the power and the cost of a method
- Thinking in terms of prevention, protection and recovery
  - Prepare for the worst-case; not optimize for it
- Thinking in aesthetic
  - Computational thinking is also about the beauty of a solution

Paper I: XORs in The Air: Practical Wireless Network Coding



XORs in The Air: Practical Wireless Network Coding

Sachin Katti[†] Hariharan Rahul[†] Wenjun Hu[*] Dina Katabi[†] Muriel Médard[†] Jon Crowcroft
[†]MIT CSAIL [*]Univ. of Cambridge

ABSTRACT

This paper proposes COPE, a new architecture for wireless mesh networks. In addition to forwarding packets, routers mix (i.e., code) packets from different sources to increase the information content of each transmission. We show that intelligently mixing packets increases network throughput. Our design is rooted in the theory of network coding. Prior work on network coding is mainly theoretical and focuses on multicast traffic. This paper aims to bridge theory with practice; it addresses the common case of unicast traffic, dynamic and potentially bursty flows, and practical issues facing the integration of network coding in the current network stack. We evaluate our design on a 20-node wireless network, and discuss the results of the first testbed deployment of wireless network coding. The results show that COPE largely increases network throughput. The gains vary from a few percent to several folds depending on the traffic pattern, congestion level, and transport protocol.

Categories and Subject Descriptors

C.2.2 **[Computer Systems Organization]**: Computer-Communications Networks

General Terms

Algorithms, Design, Performance, Theory

Keywords

Network Coding, Wireless Networks

1. INTRODUCTION

Wireless networks are indispensable; they provide the means for mobility, city-wide Internet connectivity, distributed sensing, and outdoor computing. Current wireless implementations, however, suffer from a severe throughput limitation and do not scale to dense large networks.

This paper presents COPE, a new forwarding architecture that substantially improves the throughput of wireless networks. COPE inserts a coding shim between the IP and MAC layers, which identifies coding opportunities and benefits from them by forwarding multiple packets in a single transmission.

(a) Current Approach

(b) COPE

Figure 1—A simple example of how COPE increases the throughput. It allows Alice and Bob to exchange a pair of packets using 3 transmissions instead of 4 (numbers on arrows show the order of transmission).

To give the reader a feel for how COPE works, we start with a fairly simple example. Consider the scenario in Fig. 1, where Alice and Bob want to exchange a pair of packets via a router. In current approaches, Alice sends her packet to the router, which forwards it to Bob, and Bob sends his packet to the router, which forwards it to Alice. This process requires 4 transmissions. Now consider a network coding approach. Alice and Bob send their respective packets to the router, which XORs the two packets and broadcasts the XOR-ed version. Alice and Bob can obtain each other's packet by XOR-ing again with their own packet. This process takes 3 transmissions instead of 4. Saved transmissions can be used to send new data, increasing the wireless throughput.

In fact, COPE leads to larger bandwidth savings than are apparent from this example. COPE exploits the shared nature of the wireless medium which, for free, broadcasts each packet in a small neighborhood around its path. Each node stores the overheard packets for a short time. It also tells its neighbors which packets it has heard by annotating the packets it sends. When a node transmits a packet, it uses its knowledge of what its neighbors have heard to perform *opportunistic coding*; the node XORs multiple packets and transmits them as a single packet if each intended nexthop has enough information to decode the encoded packet. This extends COPE beyond two flows that traverse the same nodes in reverse order (as in the Alice-and-Bob example), and allows it to XOR more than a pair of packets.

COPE's design is based on two key principles.

- *COPE disposes of the point-to-point abstraction and embraces the broadcast nature of the wireless channel.* Network designers typically abstract the wireless channel as a point-to-point link,

SIGCOMM'06, September 11–15, 2006, Pisa, Italy.

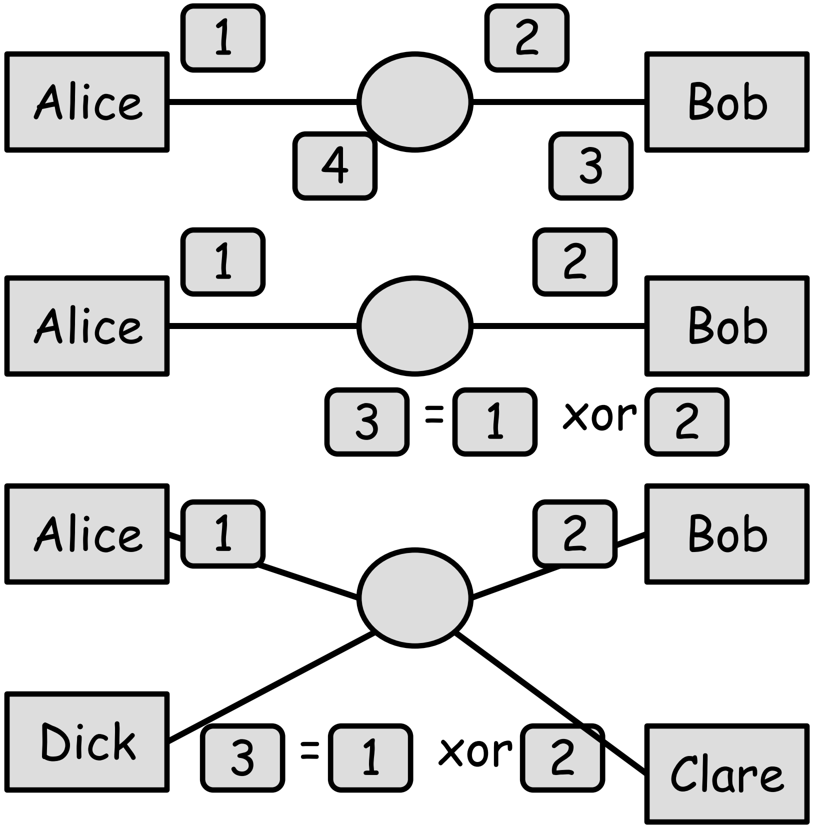
Published in
ACM SIGCOMM'06

Problem

Increasing the throughput of dense wireless mesh networks

Traditional approach

Network coding approach



Two Departures

- Accept wireless as a broadcast medium
  - Dispose of the point to point abstraction

- Routers mix bits in packets, then forward them → Network coding
  - Dispose of the store-and-forward primitive of routing

COPE Design

- Difficulty
  - When and how to code?
  - How to decode?

- Key principle – Coding opportunistically
  - Gain when chances happen; large throughput increase in general case
  - Bounded worst case performance

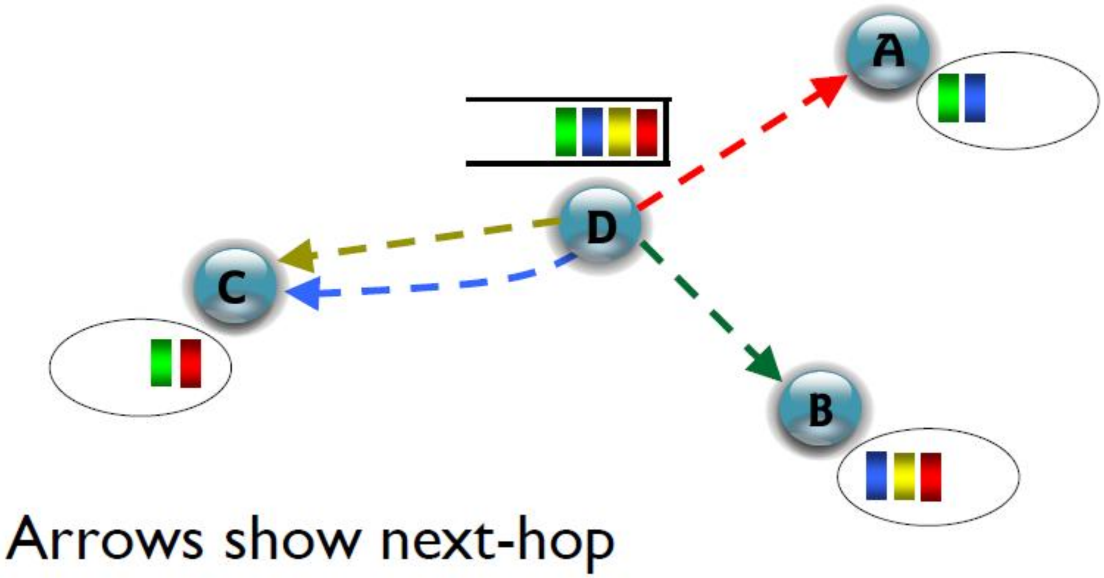
Snooping

- Exploit wireless broadcast
- Snoops on all frames in a time window
- Tradeoff between storage and coding gain

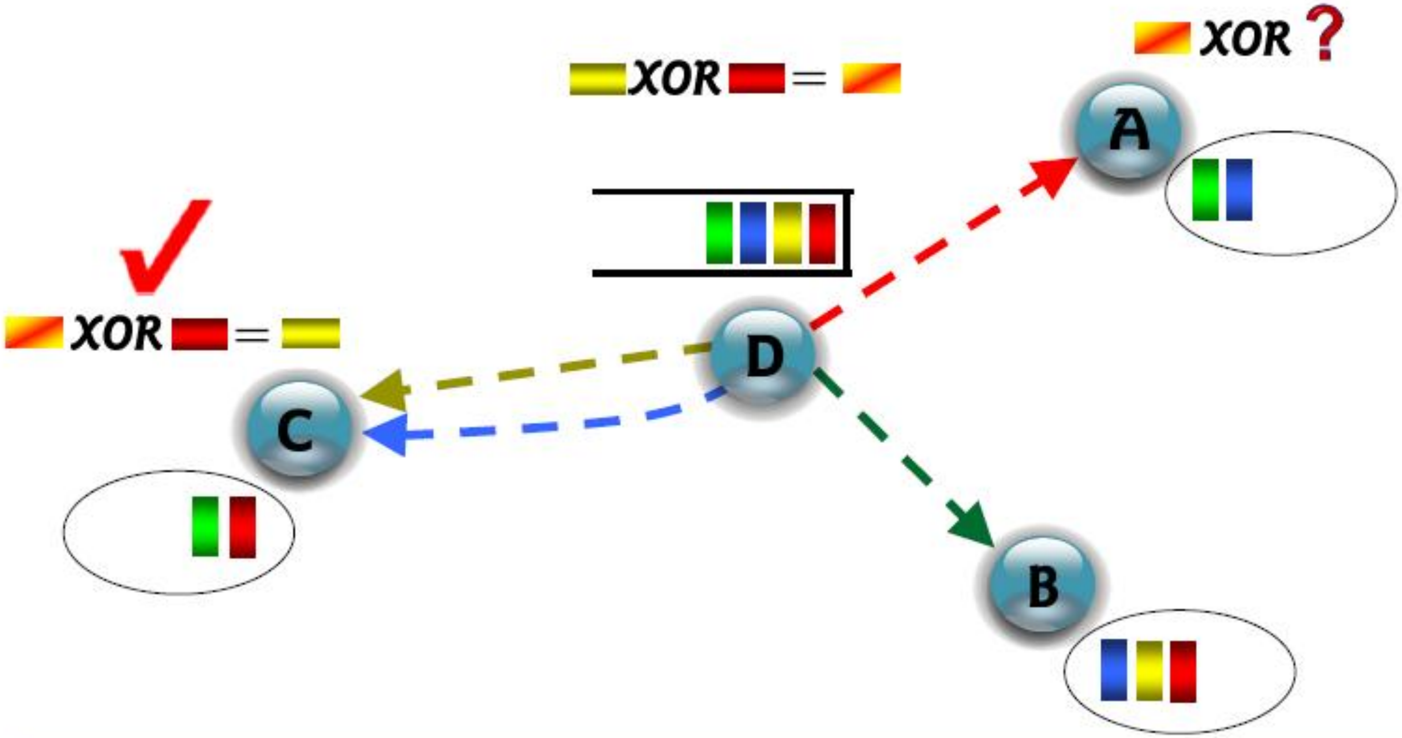
- Node sends Reception Reports to tell its neighbors what packets it heard
  - Piggyback to mitigate overhead
  - Periodically send standalone reports when no frames transmit
    - Preventing deadlocks

Coding

- Bound the worst case
  - To send frame p to neighbor A, XOR p with frames already known to A
    - Thus, A can always decode; no worse than the traditional design
- Benefit multiple neighbors opportunistically



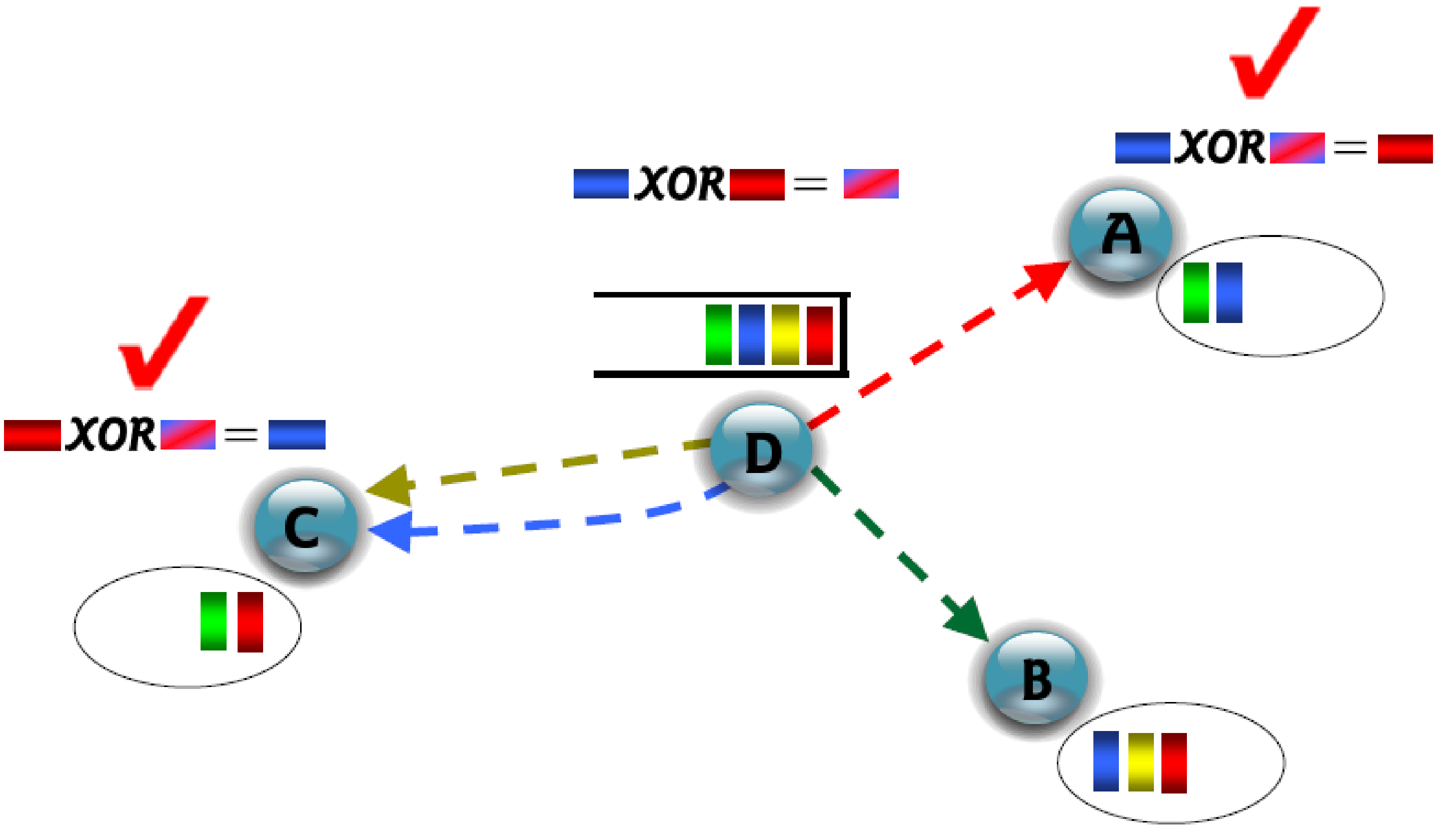
Coding Illustration



Bad Coding

Only one neighbor benefits from one transmission

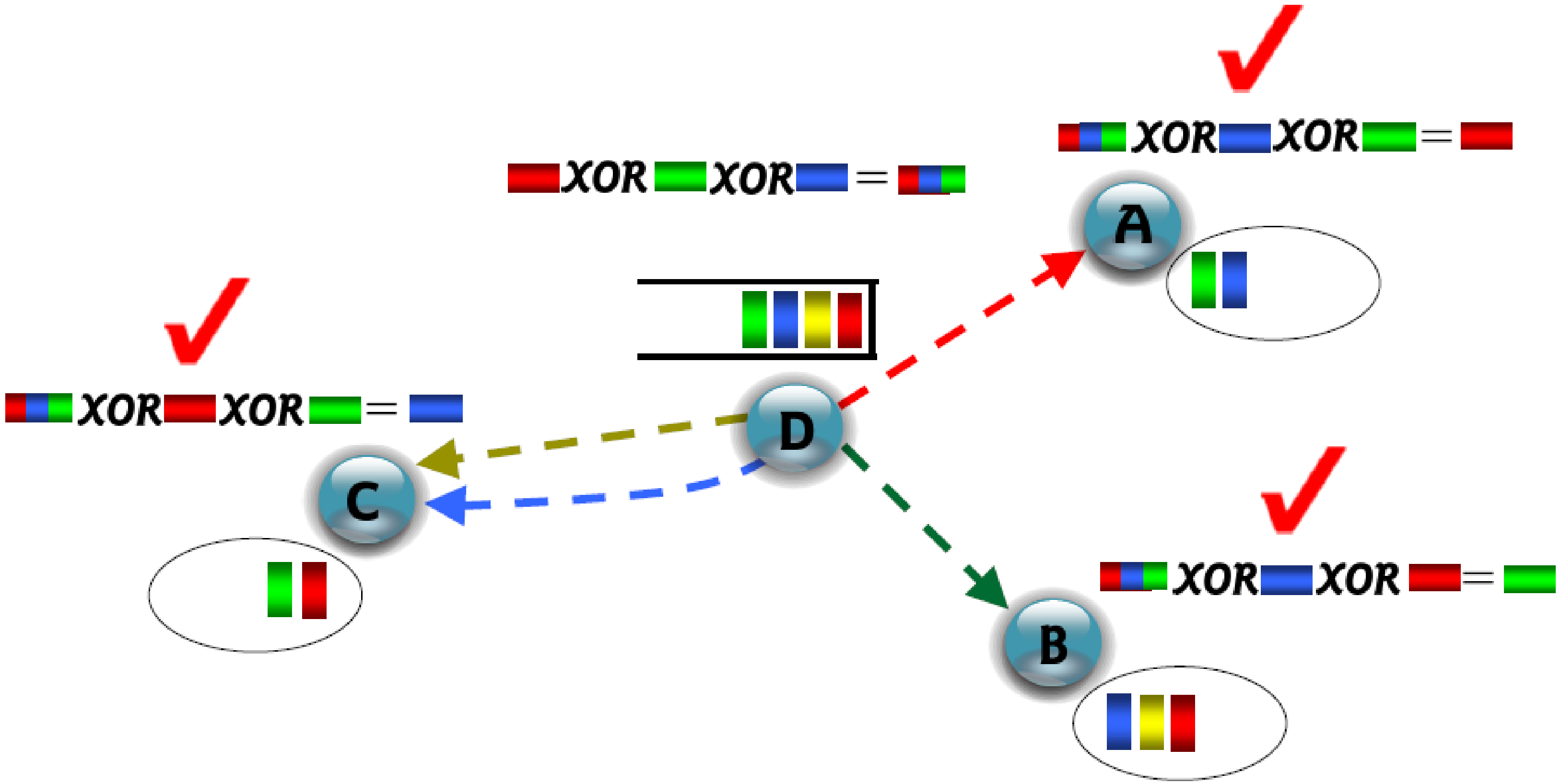
Coding Illustration (cont.)



Good Coding

Two neighbors benefit from one transmission!

Coding Illustration (cont.)



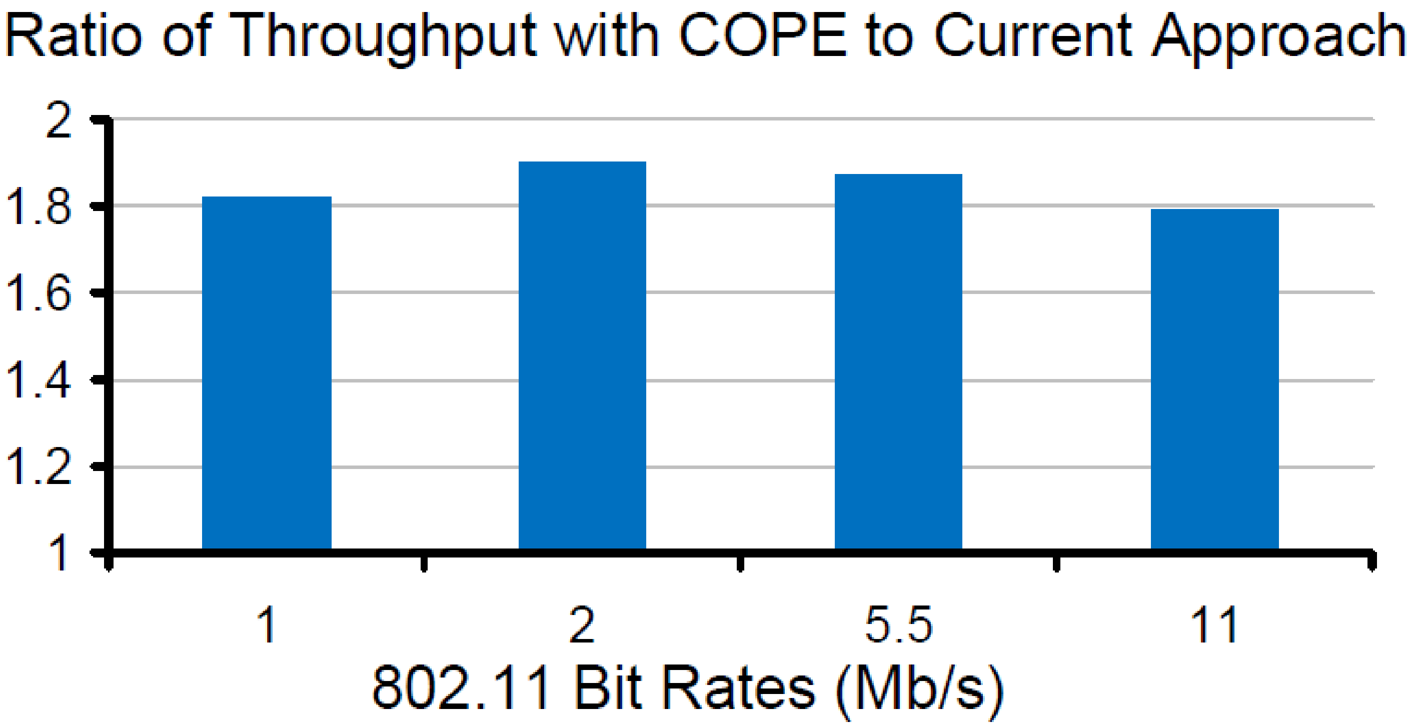
Best Coding

Three neighbors benefit from one transmission!

Coding Algorithm

- Simple rule – decodable condition
  - XOR n frames together iff the next hop of each frame already has the other n-1 frames apart from the one he wants
- How does a node know its neighbors' frames?
  - Reception reports
  - Guesses based on delivery rate between the two nodes
    - Dealing with reports lose or delay
  - If error occurs, recover by retransmission
- If there is no coding chance, just forward directly; never delay a frame

Performance: Alice-Bob Case

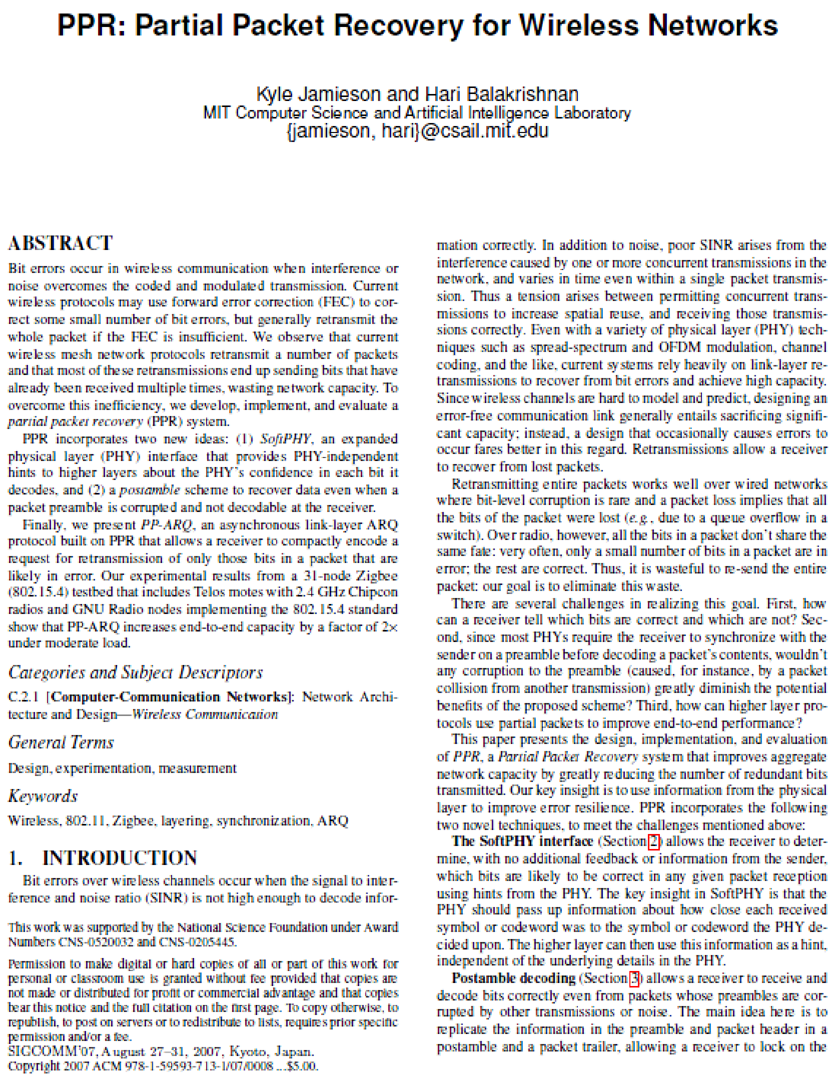


Almost double the throughput

Summary

- First integration of network coding into the network stack
- Simple ideas that work → Elegant
  - Opportunistic design
  - Optimize for the normal case and bound the worst case
  - Carefully making tradeoffs

Paper II: PPR: Partial Packet Recovery for Wireless Networks



PPR: Partial Packet Recovery for Wireless Networks

Kyle Jamieson and Hari Balakrishnan
MIT Computer Science and Artificial Intelligence Laboratory
{jamieson, hari}@csail.mit.edu

ABSTRACT

Bit errors occur in wireless communication when interference or noise overcomes the coded and modulated transmission. Current wireless protocols may use forward error correction (FEC) to correct some small number of bit errors, but generally retransmit the whole packet if the FEC is insufficient. We observe that current wireless mesh network protocols retransmit a number of packets and that most of these retransmissions end up sending bits that have already been received multiple times, wasting network capacity. To overcome this inefficiency, we develop, implement, and evaluate a *partial packet recovery* (PPR) system.

PPR incorporates two new ideas: (1) *SoftPHY*, an expanded physical layer (PHY) interface that provides PHY-independent hints to higher layers about the PHY's confidence in each bit it decodes, and (2) a *postamble* scheme to recover data even when a packet preamble is corrupted and not decodable at the receiver.

Finally, we present *PP-ARQ*, an asynchronous link-layer ARQ protocol built on PPR that allows a receiver to compactly encode a request for retransmission of only those bits in a packet that are likely in error. Our experimental results from a 31-node Zigbee (802.15.4) testbed that includes Telos motes with 2.4 GHz Chipcon radios and GNU Radio nodes implementing the 802.15.4 standard show that PP-ARQ increases end-to-end capacity by a factor of 2× under moderate load.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless Communication*

General Terms

Design, experimentation, measurement

Keywords

Wireless, 802.11, Zigbee, layering, synchronization, ARQ

1. INTRODUCTION

Bit errors over wireless channels occur when the signal to interference and noise ratio (SINR) is not high enough to decode information correctly. In addition to noise, poor SINR arises from the interference caused by one or more concurrent transmissions in the network, and varies in time even within a single packet transmission. Thus a tension arises between permitting concurrent transmissions to increase spatial reuse, and receiving those transmissions correctly. Even with a variety of physical layer (PHY) techniques such as spread-spectrum and OFDM modulation, channel coding, and the like, current systems rely heavily on link-layer retransmissions to recover from bit errors and achieve high capacity. Since wireless channels are hard to model and predict, designing an error-free communication link generally entails sacrificing significant capacity; instead, a design that occasionally causes errors to occur fares better in this regard. Retransmissions allow a receiver to recover from lost packets.

Retransmitting entire packets works well over wired networks where bit-level corruption is rare and a packet loss implies that all the bits of the packet were lost (*e.g.*, due to a queue overflow in a switch). Over radio, however, all the bits in a packet don't share the same fate: very often, only a small number of bits in a packet are in error; the rest are correct. Thus, it is wasteful to re-send the entire packet: our goal is to eliminate this waste.

There are several challenges in realizing this goal. First, how can a receiver tell which bits are correct and which are not? Second, since most PHYs require the receiver to synchronize with the sender on a preamble before decoding a packet's contents, wouldn't any corruption to the preamble (caused, for instance, by a packet collision from another transmission) greatly diminish the potential benefits of the proposed scheme? Third, how can higher layer protocols use partial packets to improve end-to-end performance?

This paper presents the design, implementation, and evaluation of *PPR*, a *Partial Packet Recovery* system that improves aggregate network capacity by greatly reducing the number of redundant bits transmitted. Our key insight is to use information from the physical layer to improve error resilience. PPR incorporates the following two novel techniques, to meet the challenges mentioned above:

The SoftPHY interface (Section 2) allows the receiver to determine, with no additional feedback or information from the sender, which bits are likely to be correct in any given packet reception using hints from the PHY. The key insight in SoftPHY is that the PHY should pass up information about how close each received symbol or codeword was to the symbol or codeword the PHY decided upon. The higher layer can then use this information as a hint, independent of the underlying details in the PHY.

Postamble decoding (Section 3) allows a receiver to receive and decode bits correctly even from packets whose preambles are corrupted by other transmissions or noise. The main idea here is to replicate the information in the preamble and packet header in a postamble and a packet trailer, allowing a receiver to lock on the

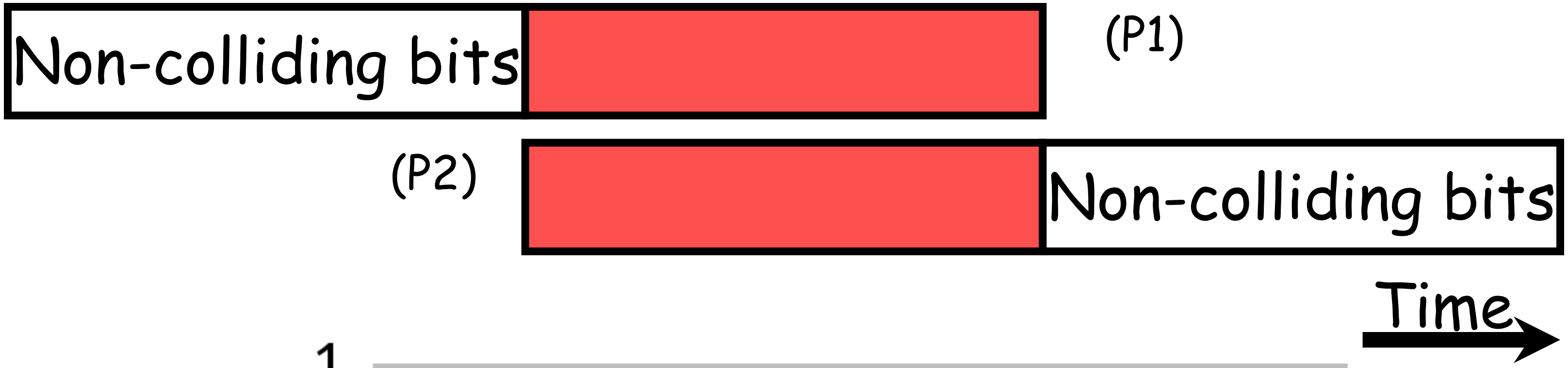
This work was supported by the National Science Foundation under Award Numbers CNS-0520032 and CNS-0205445.

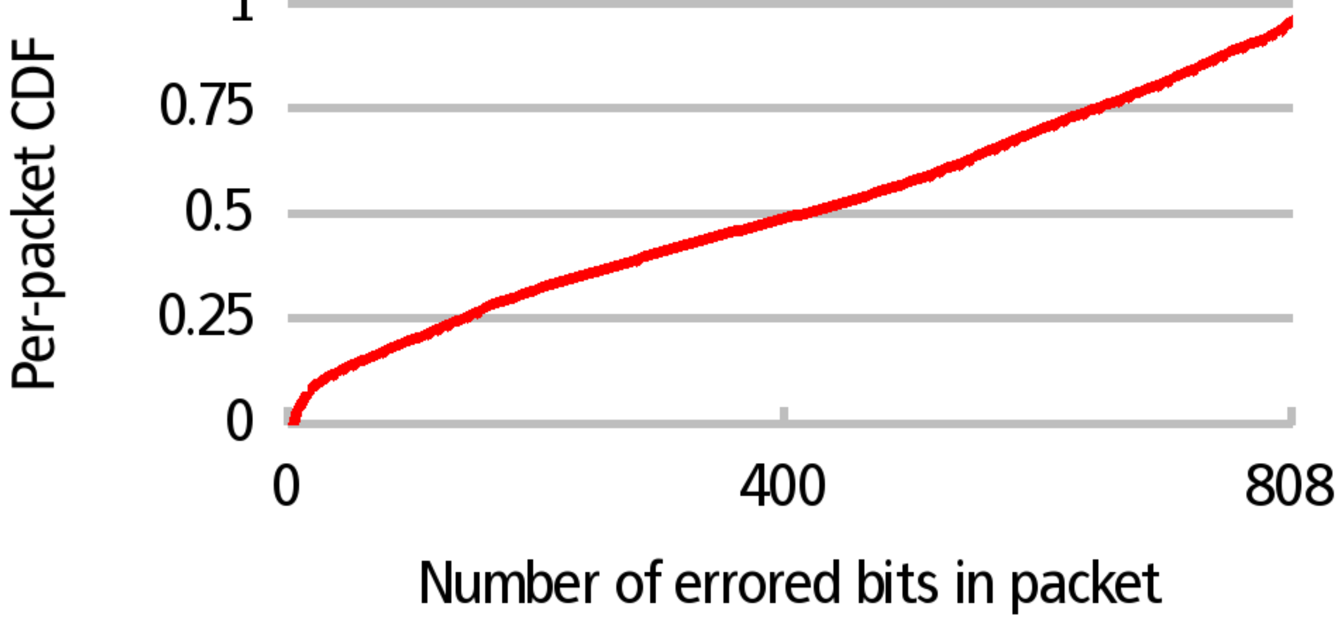
SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.
Copyright 2007 ACM 978-1-59593-713-1/07/0008 ...$5.00.

Published in
ACM SIGCOMM'07

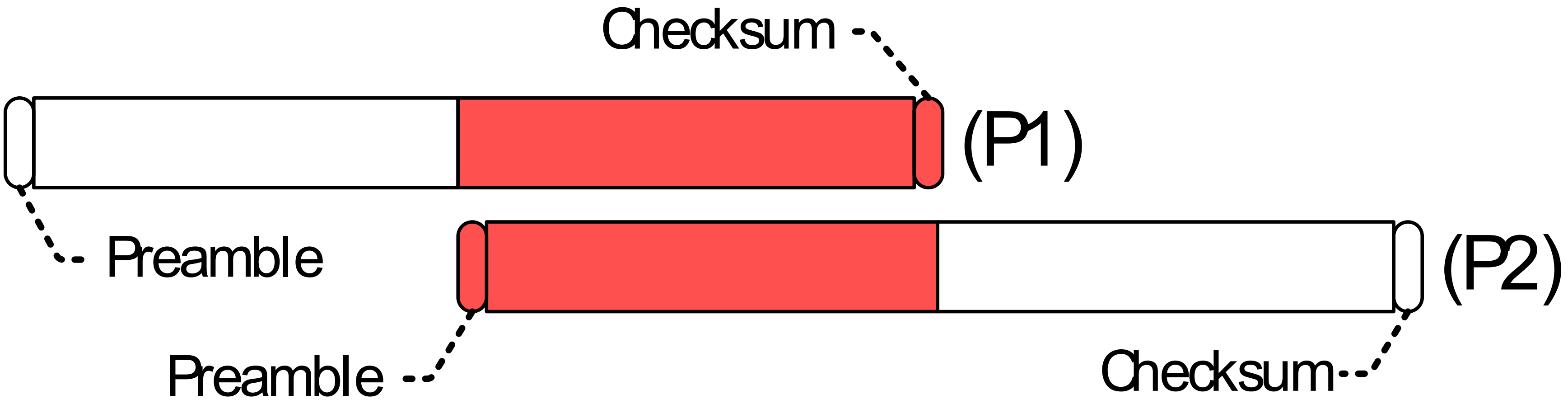
Problem

- How many bits are erroneous when a frame lost?





Key Question – How to Make Use of Correct Bits?



Difficulties

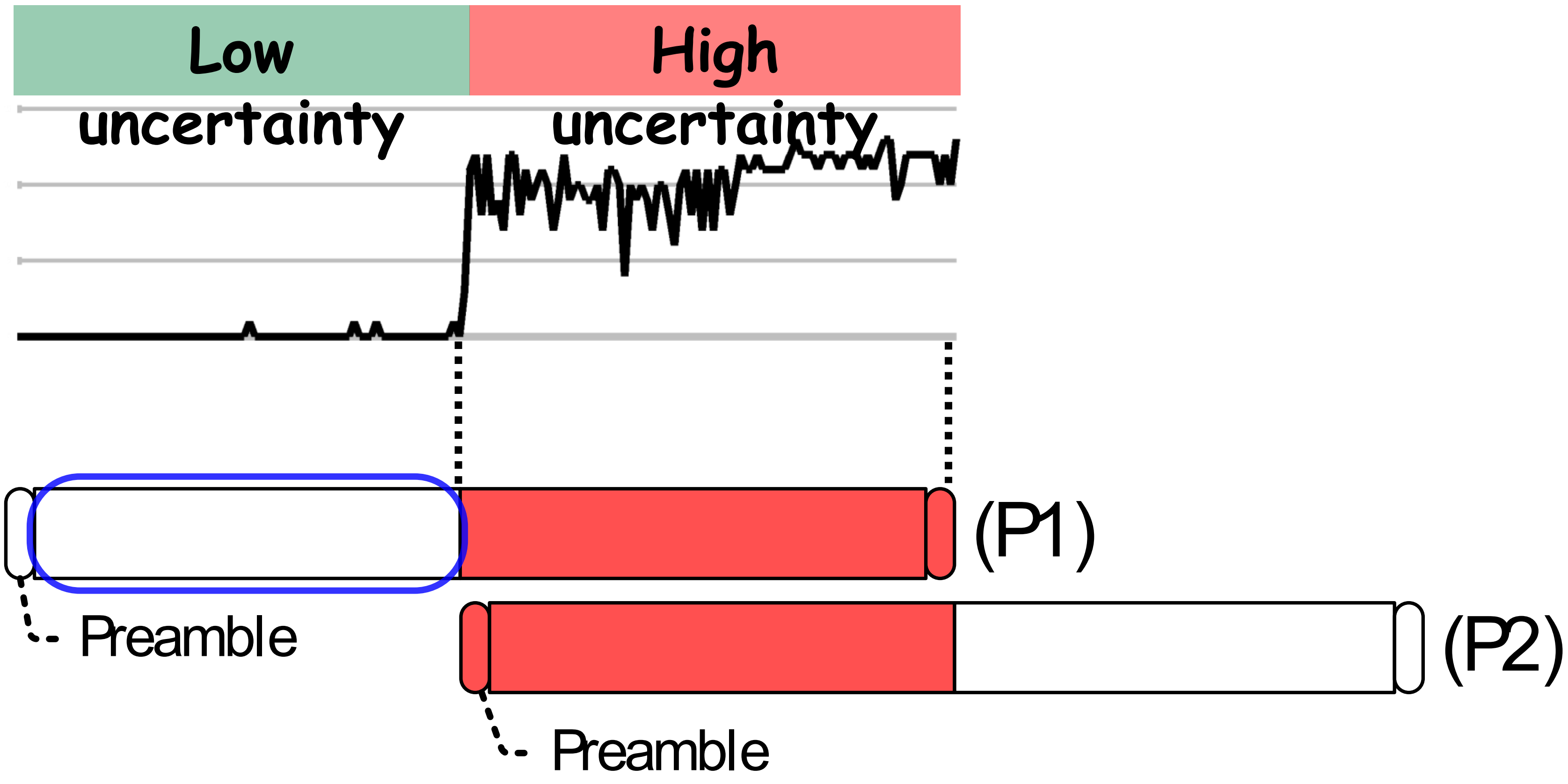
1. How does receiver know which bits are correct?
2. How does receiver know P2 is there at all?
3. How to design an efficient ARQ protocol?

How can receiver identify correct bits?

- Use physical layer (PHY) hints: **SoftPHY**
  - Receiver PHY has the information!
  - Pass this **confidence information** to higher layer as a hint

- New interface abstraction: SoftPHY
  - PHY-independent

Opportunistically Guess Erroneous Bits



Example: SoftPHY hint for spread spectrum

Hamming distance between **received** chips and **decided-upon** codeword

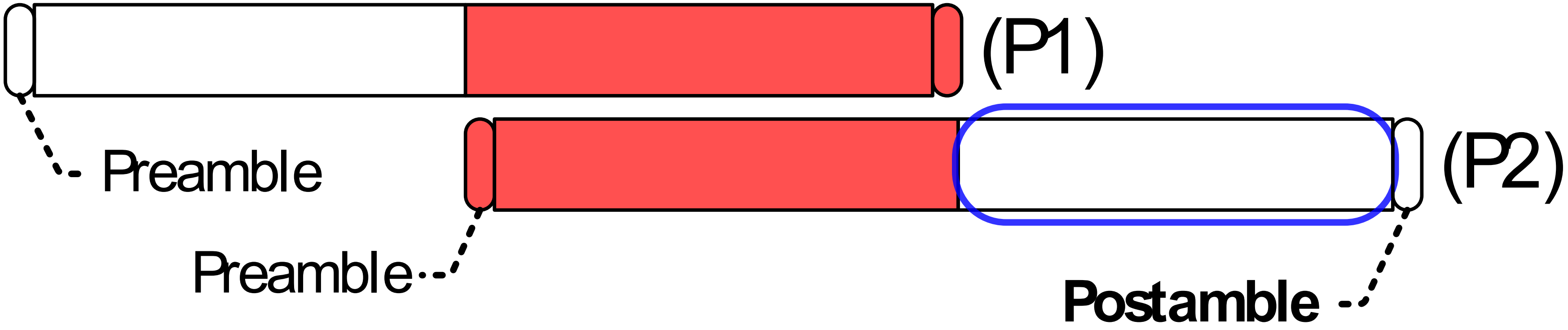
Receive: 11101101000111000011010110100010
C_1: 11101101100111000011010100100010

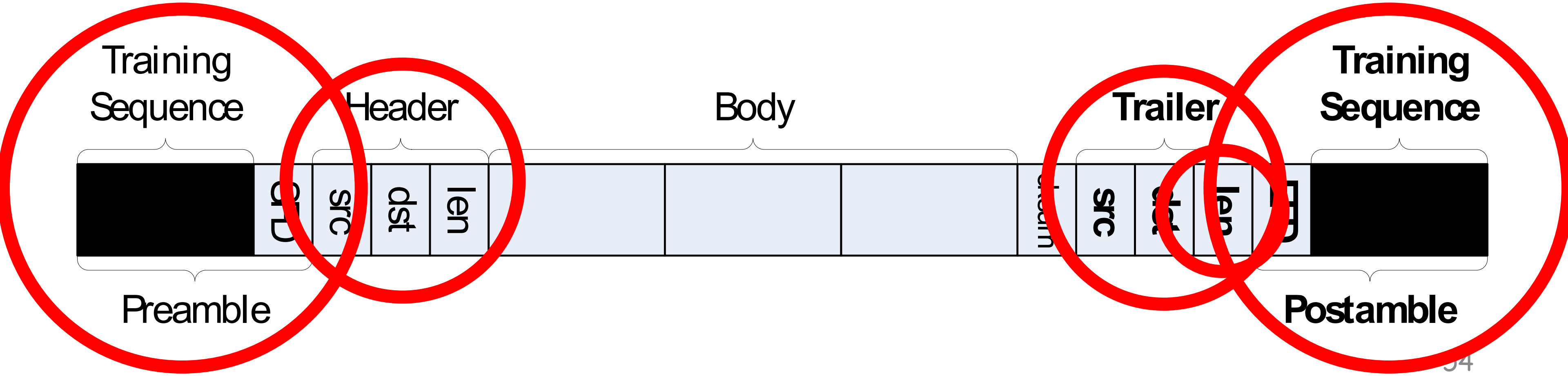
⇒ SoftPHY hint is **2**

Receive: 11001101000111010111011110110111
C_1: 11101101100111000011010100100010

⇒ SoftPHY hint is **9**

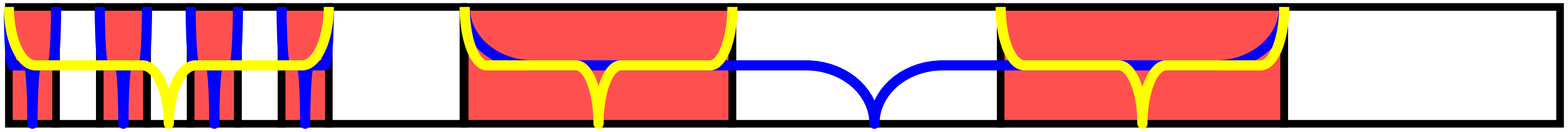
Postamble Decoding - Store and Decode





ARQ with partial packets

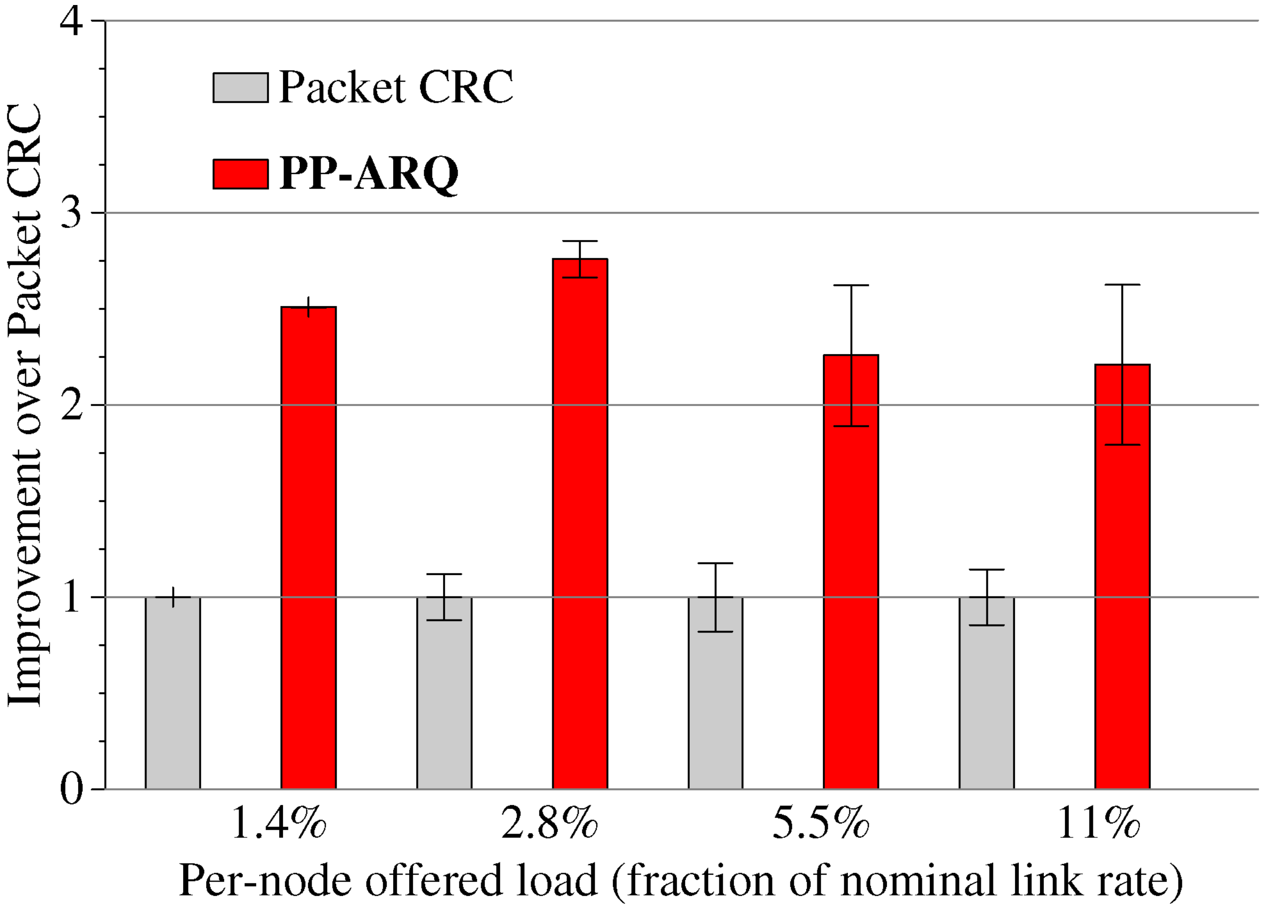
- Resend **only** incorrect bits
- Difficulty
  - How to **efficiently** tell sender about what happened?
- Efficient feedback
  - Labeling bits "good" or "bad"
  - Merging the good/bad range with minimal bit cost
    - Dynamic programming problem
    - Accept some false positive and false negatives



□ "Good" bits

🟥 "Bad" bits

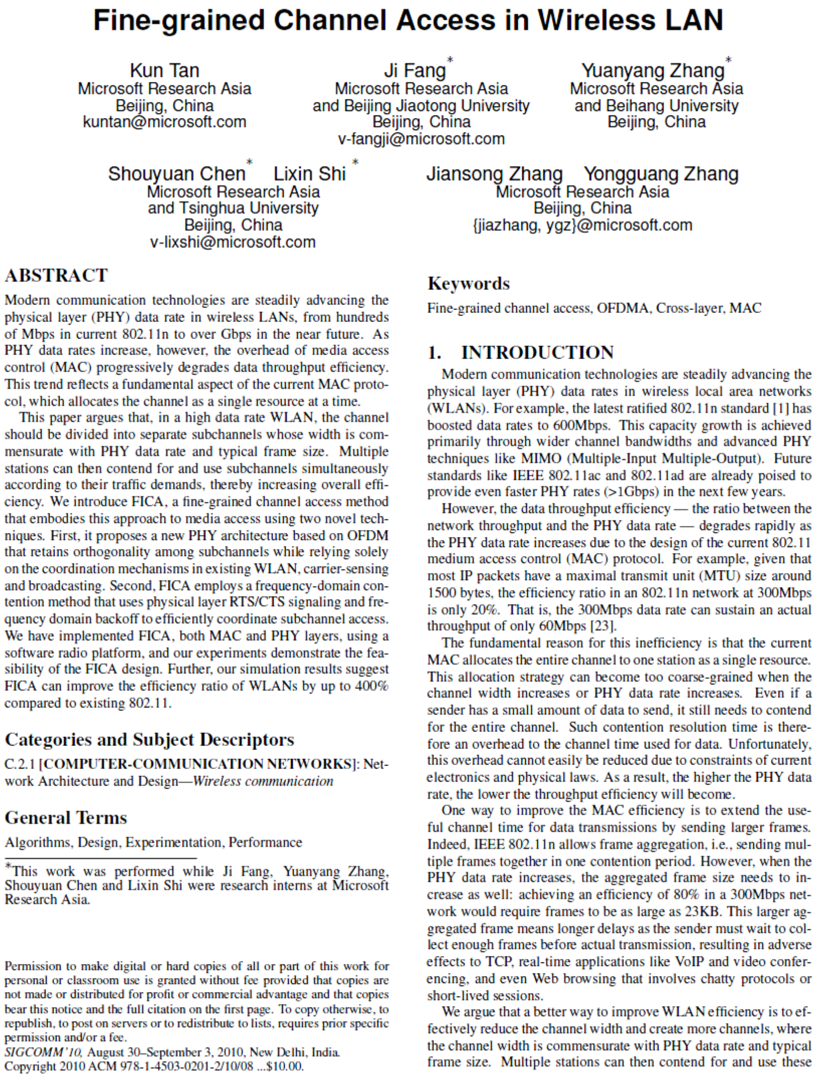
Throughput improvement 2.3-2.8x



Summary

- New mechanisms for recovering correct bits from parts of frames
  - SoftPHY interface (PHY-independent)
  - Postamble decoding
- Inspirations on new applications
  - Collision detection
  - Opportunistic forwarding

Paper III: Fine-grained Channel Access in WLAN



Fine-grained Channel Access in Wireless LAN

Kun Tan
Microsoft Research Asia
Beijing, China
kuntan@microsoft.com

Ji Fang*
Microsoft Research Asia
and Beijing Jiaotong University
Beijing, China
v-fangji@microsoft.com

Yuanyang Zhang*
Microsoft Research Asia
and Beihang University
Beijing, China

Shouyuan Chen* Lixin Shi *
Microsoft Research Asia
and Tsinghua University
Beijing, China
v-lixshi@microsoft.com

Jiansong Zhang Yongguang Zhang
Microsoft Research Asia
Beijing, China
{jiazhang, ygz}@microsoft.com

ABSTRACT

Modern communication technologies are steadily advancing the physical layer (PHY) data rate in wireless LANs, from hundreds of Mbps in current 802.11n to over Gbps in the near future. As PHY data rates increase, however, the overhead of media access control (MAC) progressively degrades data throughput efficiency. This trend reflects a fundamental aspect of the current MAC protocol, which allocates the channel as a single resource at a time.

This paper argues that, in a high data rate WLAN, the channel should be divided into separate subchannels whose width is commensurate with PHY data rate and typical frame size. Multiple stations can then contend for and use subchannels simultaneously according to their traffic demands, thereby increasing overall efficiency. We introduce FICA, a fine-grained channel access method that embodies this approach to media access using two novel techniques. First, it proposes a new PHY architecture based on OFDM that retains orthogonality among subchannels while relying solely on the coordination mechanisms in existing WLAN, carrier-sensing and broadcasting. Second, FICA employs a frequency-domain contention method that uses physical layer RTS/CTS signaling and frequency domain backoff to efficiently coordinate subchannel access. We have implemented FICA, both MAC and PHY layers, using a software radio platform, and our experiments demonstrate the feasibility of the FICA design. Further, our simulation results suggest FICA can improve the efficiency ratio of WLANs by up to 400% compared to existing 802.11.

Categories and Subject Descriptors

C.2.1 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Architecture and Design—*Wireless communication*

General Terms

Algorithms, Design, Experimentation, Performance

*This work was performed while Ji Fang, Yuanyang Zhang, Shouyuan Chen and Lixin Shi were research interns at Microsoft Research Asia.

SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.
Copyright 2010 ACM 978-1-4503-0201-2/10/08 ...$10.00.

Keywords

Fine-grained channel access, OFDMA, Cross-layer, MAC

1. INTRODUCTION

Modern communication technologies are steadily advancing the physical layer (PHY) data rates in wireless local area networks (WLANs). For example, the latest ratified 802.11n standard [1] has boosted data rates to 600Mbps. This capacity growth is achieved primarily through wider channel bandwidths and advanced PHY techniques like MIMO (Multiple-Input Multiple-Output). Future standards like IEEE 802.11ac and 802.11ad are already poised to provide even faster PHY rates (>1Gbps) in the next few years.

However, the data throughput efficiency — the ratio between the network throughput and the PHY data rate — degrades rapidly as the PHY data rate increases due to the design of the current 802.11 medium access control (MAC) protocol. For example, given that most IP packets have a maximal transmit unit (MTU) size around 1500 bytes, the efficiency ratio in an 802.11n network at 300Mbps is only 20%. That is, the 300Mbps data rate can sustain an actual throughput of only 60Mbps [23].

The fundamental reason for this inefficiency is that the current MAC allocates the entire channel to one station as a single resource. This allocation strategy can become too coarse-grained when the channel width increases or PHY data rate increases. Even if a sender has a small amount of data to send, it still needs to contend for the entire channel. Such contention resolution time is therefore an overhead to the channel time used for data. Unfortunately, this overhead cannot easily be reduced due to constraints of current electronics and physical laws. As a result, the higher the PHY data rate, the lower the throughput efficiency will become.

One way to improve the MAC efficiency is to extend the useful channel time for data transmissions by sending larger frames. Indeed, IEEE 802.11n allows frame aggregation, i.e., sending multiple frames together in one contention period. However, when the PHY data rate increases, the aggregated frame size needs to increase as well: achieving an efficiency of 80% in a 300Mbps network would require frames to be as large as 23KB. This larger aggregated frame means longer delays as the sender must wait to collect enough frames before actual transmission, resulting in adverse effects to TCP, real-time applications like VoIP and video conferencing, and even Web browsing that involves chatty protocols or short-lived sessions.

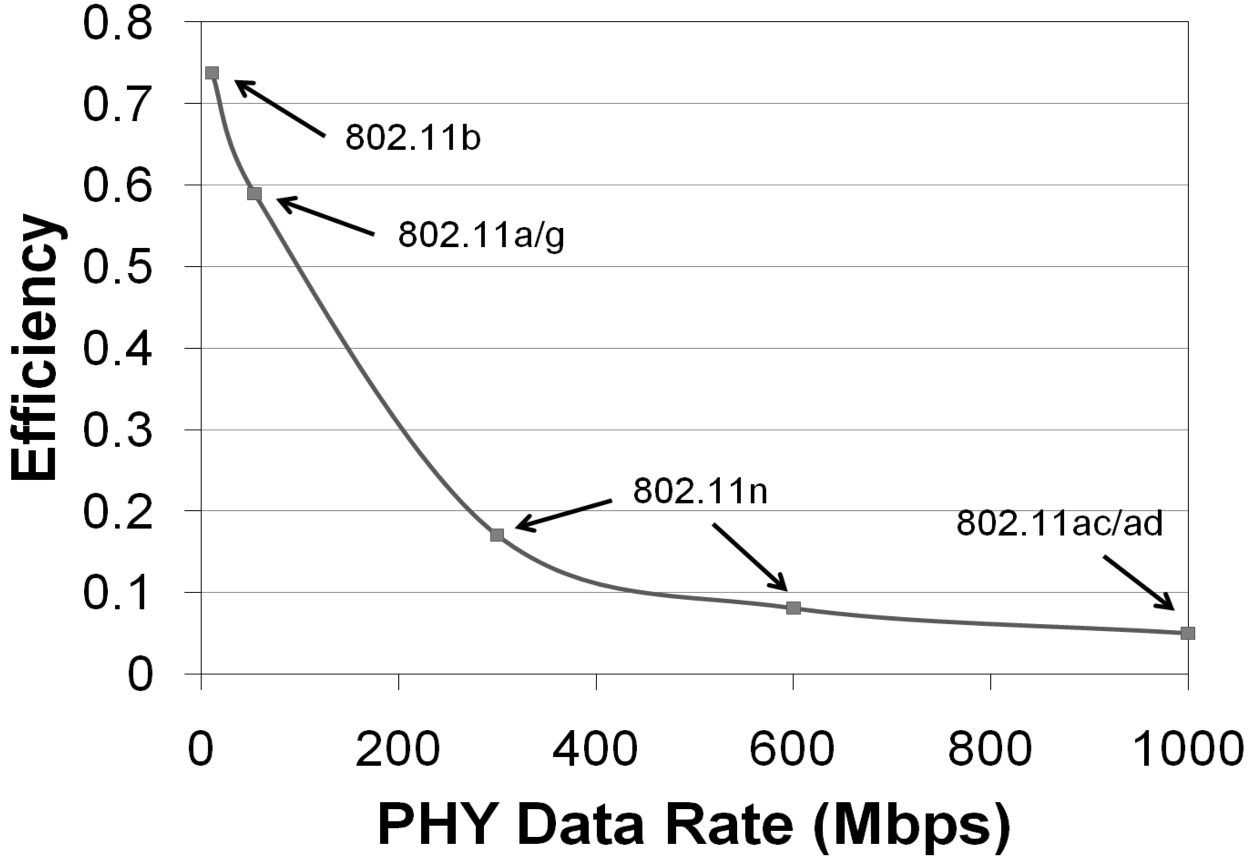
We argue that a better way to improve WLAN efficiency is to effectively reduce the channel width and create more channels, where the channel width is commensurate with PHY data rate and typical frame size. Multiple stations can then contend for and use these

To appear in
ACM SIGCOMM'10

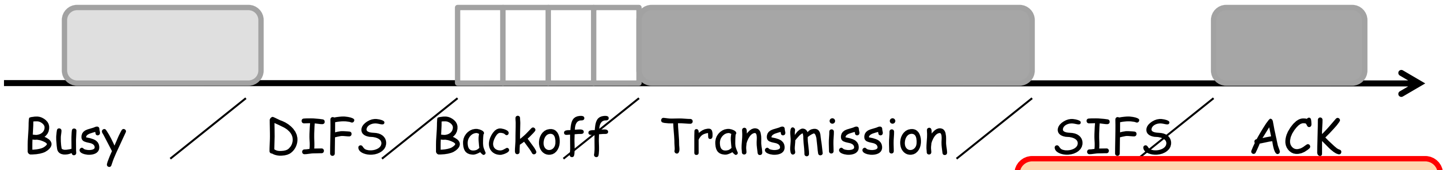
Problem

- How much we can benefit from the increase of PHY data rate?

- Fixed frame size 1500B



Understanding the Overhead



Useful air time

$$\eta = \frac{t_{data}}{t_{DIFS} + t_{slot} \cdot \bar{W} + t_{data} + t_{SIFS} + t_{ACK}}$$

$t_{DIFS} = t_{SIFS} + 2t_{slot}$

⇨ # of nodes

$t_{SIFS} = t_{RFDelay} + t_{proc} + t_{TxRx}$ ⇨ 10~16μs

$t_{slot} = t_{CCA} + t_{TxRx} + t_{prop}$ ⇨ 9~20μs

- When PHY data rate increases high, the useful air time reduces, but the overhead remains the same
  - Constrained by fundamental physics laws and electronics

Limitation of Existing MAC

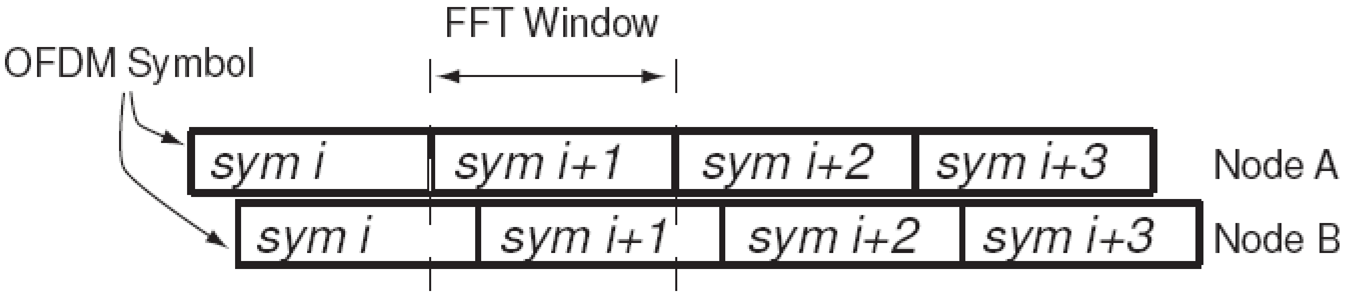
- Allocate whole channel to one user at a time
  - Single Carrier : too coarse when
    - The bandwidth is wide
    - Data rate is high

  - Aggregate a large amount of data to be efficient
    - 23KB per frame for 80% efficiency at 300Mbps data rate
    - Adversary interaction w/ RT, interactive, Web traffic, etc.

Call for a new access method – Fine Grained Channel Access

Why Difficult?

- Direct reduce the channel width
  - Guard-band overhead increase significantly with fine subchannels
    - 75% overhead if channel width is 5MHz w/ 802.11a
- Use orthogonal subcarriers
  - Overlapped but cross-interference is zero
  - OFDM-like transmission among different nodes
  - Require coordination among nodes

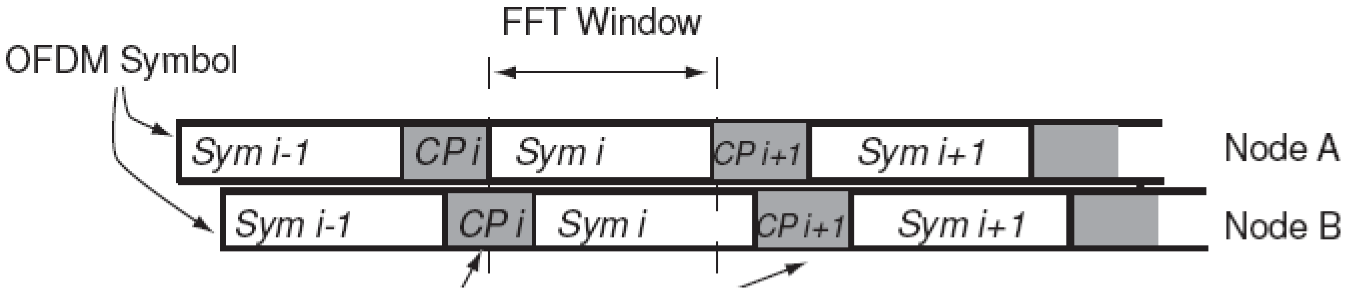


Design Space

- Tight centralized control
  - Tight time synchronization and transmission control
  - Mitigate the symbol misalignment
  - Proposed in 4G standards; but not fit for unlicensed wireless
- Distributed coordination ➔ FICA
  - Rely on carrier sensing and broadcasting
  - Accommodate the possible misalignment
  - Suitable for WLAN in unlicensed band

FICA Approach

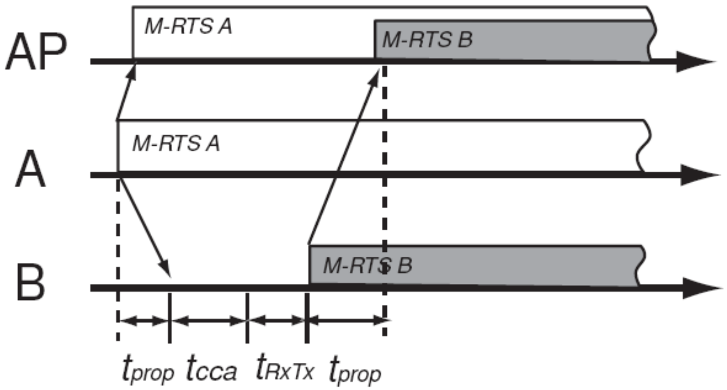
- A new PHY architecture
  - Adopt a large guard-interval (or CP) and large OFDM symbol time to accommodate the time misalignment



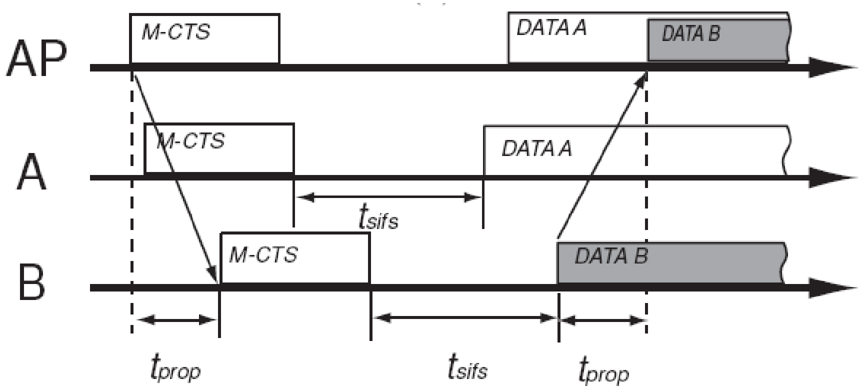
- A new MAC contention and backoff scheme
  - Time-domain random backoff is inefficient
  - Frequency-domain contention and backoff

New PHY Architecture

- Timing misalignment in WLAN
  - Bounded by carrier sensing (< 11 μs)and broadcasting (< 2 μs)



Carrier-sensing $t_{mis} < 11\mu s$

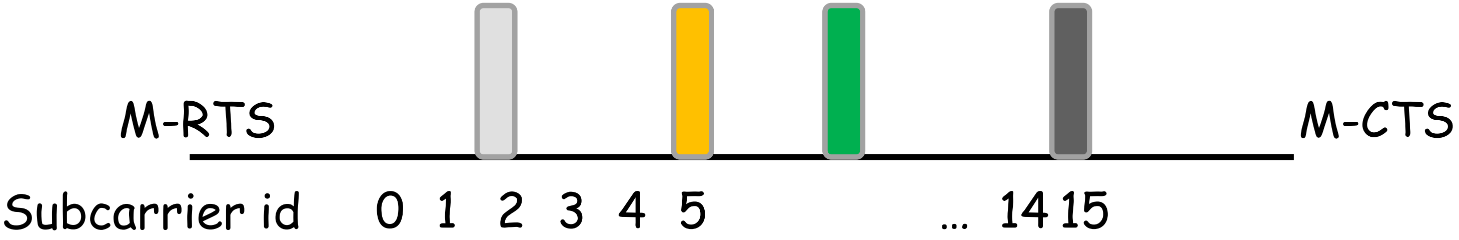


Broadcasting $t_{mis} < 2\mu s$

- Symbol structure
  - Long CP 11.8μs, short CP 2.8μs
  - Data symbol 15.6 μs (20% CP overhead)
- Subchannelization: 1.33MHz (17 subcarriers of 256-FFT)

Frequency Domain Contention

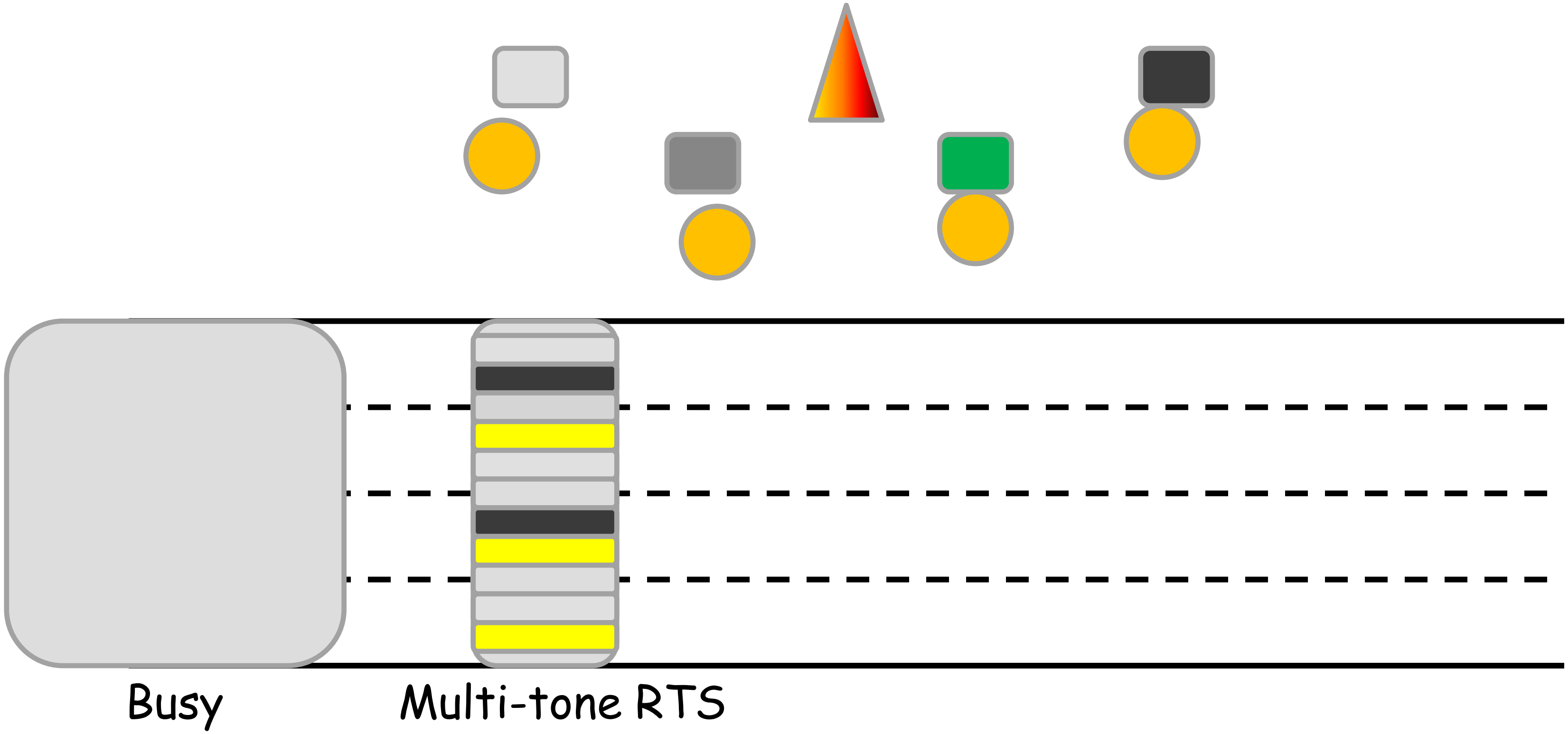
- Use PHY signaling
  - Special design OFDM symbols for M-RTS/CTS
- Module contention-information on different contention subcarriers
- Contention resolved by choosing a winning subcarrier



Frequency Domain Backoff

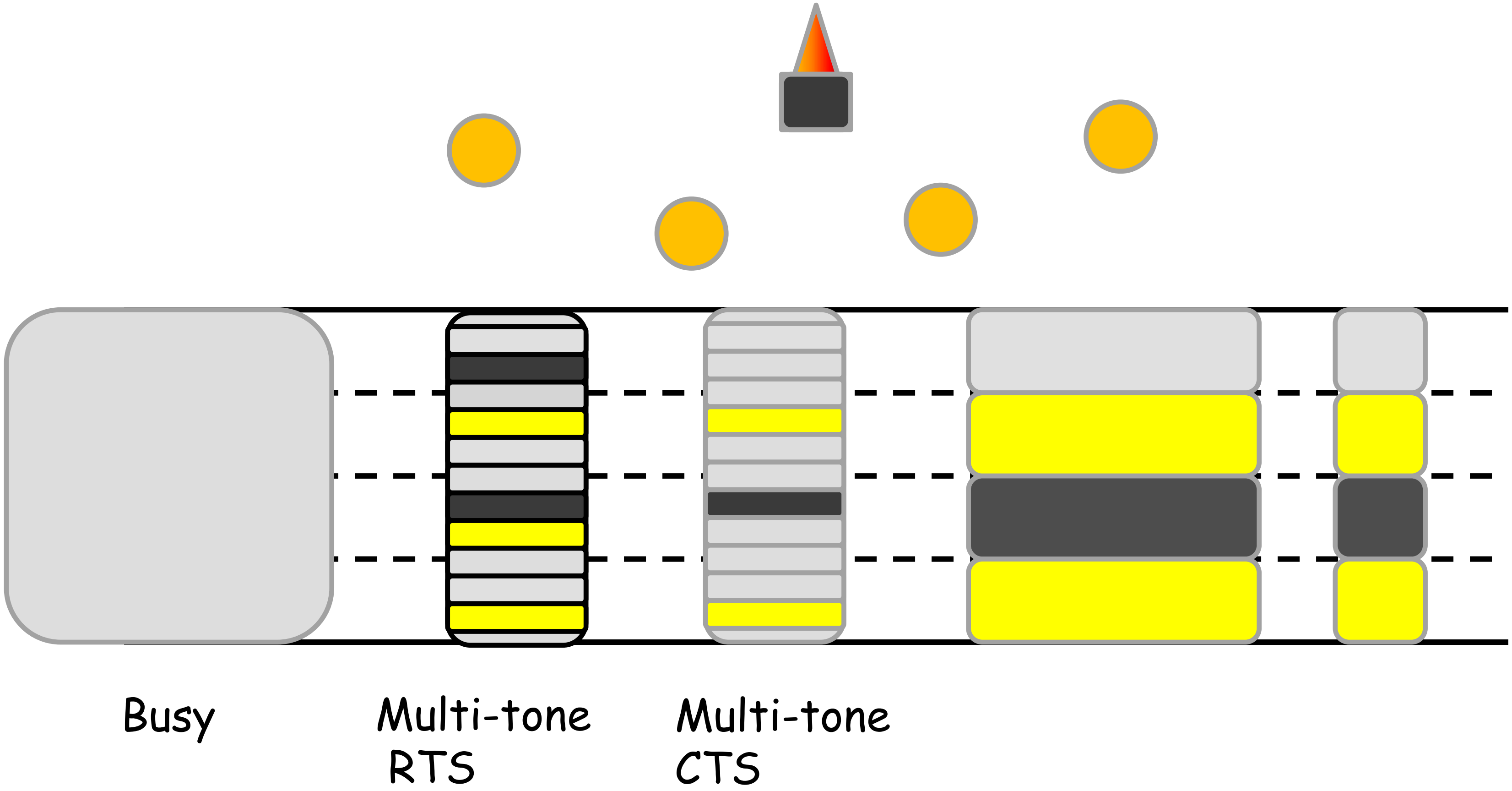
- Make frequency-domain contention scale
- Apply congestion control principles
  - Control the maximal number (*cmax*) of subchannels to access
    - Reduce *cmax* if collision
    - Increase *cmax* if success
  - Various rules: Reset-to-Max, AIMD

Access with FICA



- M-RTS: coordinated with carrier-sensing, using long CP
- Other frames: coordinated with previous frames, using short CP

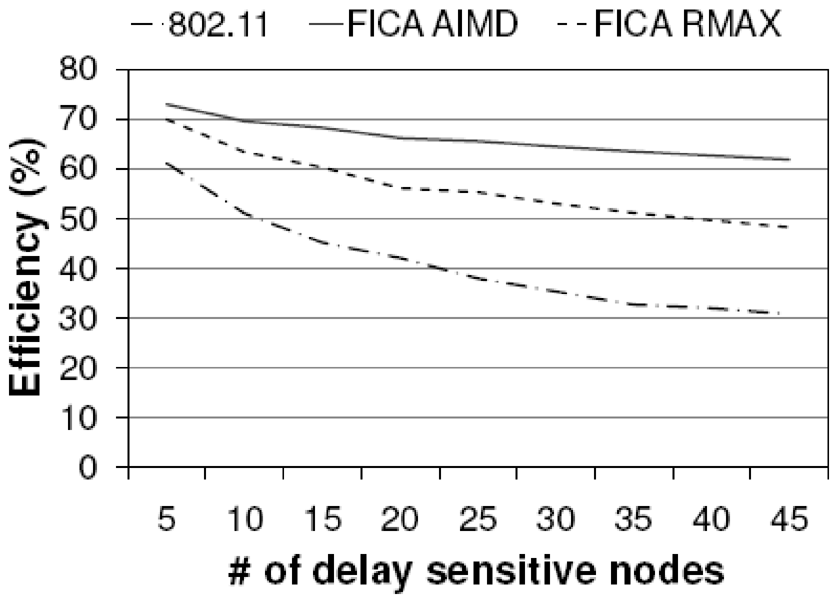
Access with FICA

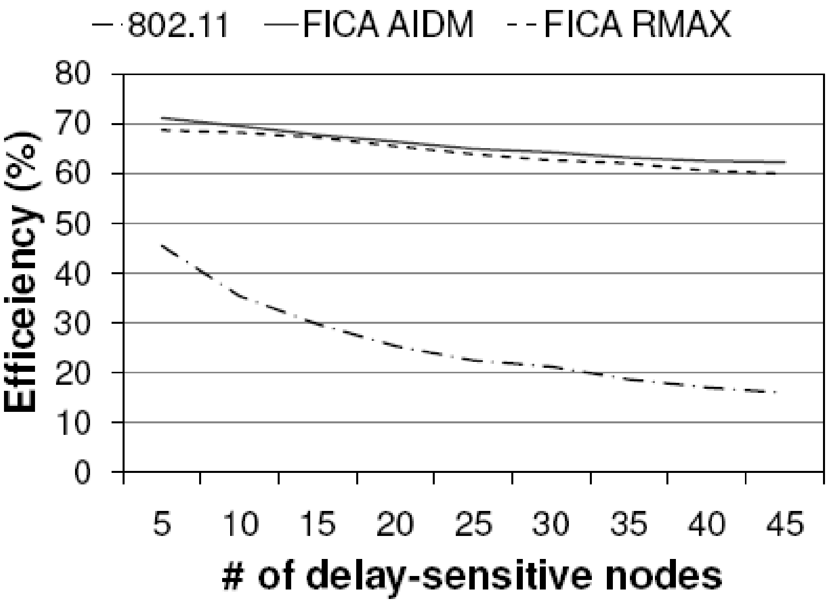


- M-RTS: coordinated with carrier-sensing, using long CP
- Other frames: coordinated with previous frames, using short CP

Results

- Mixed traffic: 5 saturated; others are with random traffic rate from 800K~5Mbps





11n PHY: 150Mbps; FICA PHY: 145Mbps

11n PHY: 600Mbps; FICA PHY: 580Mbps

Summary

- A radical design for random access in wireless network
  - Balanced design among synchronization requirements and PHY efficiency
  - PHY signaling and frequency domain contention to reduce the signaling overhead
  - Frequency domain backoff
    - Explorer another dimension for backoff
- Inspirit new thinking on wireless protocol design

Conclusions

- Computational thinking is a fundamental skill that every one should have
- It is thinking on problems and solutions, but fundamentally on problems
- It is conceptualizing, but not programming or artifact
- It complements and combines both mathematical and engineering thinking, but differs from either of them

- It is an ART

Sora Tutorial

platform, tools and programming

Using Sora for Wireless Experiments

- Option 1: Hardware Unit Test Tool
  - A versatile driver for Sora platform, originally developed for hardware testing purpose
  - Main functions:
    - Transmit a pre-computed wave-form – signal generator
    - Dump a snap-shot of wireless channel – signal capture
  - Simple, but very useful for offline signal processing
- Option 2: User Mode Extension
  - User-mode APIs to develop a SDR program
  - Simple programming model; easy debug
  - Could be interrupted by critical kernel events

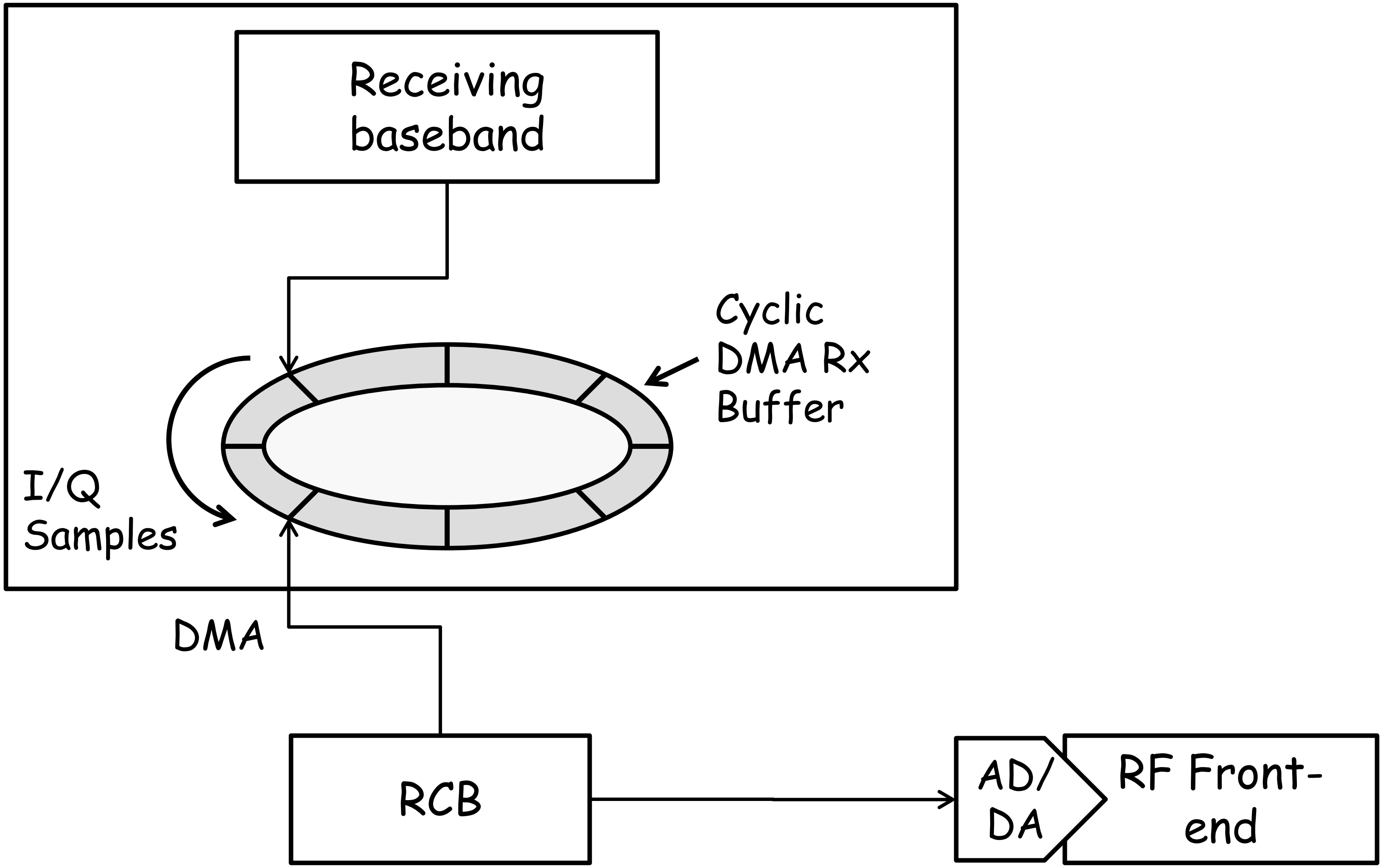
Using Sora for Wireless Experiments (cont.)

- Option 3: Kernel SDR driver
  - Integration to the network stack; virtual NICs to support existing applications
  - High privilege and real-time guarantee (with Core dedication)
  - Difficult to program, but loved by system hackers ☺

What you will learn next?

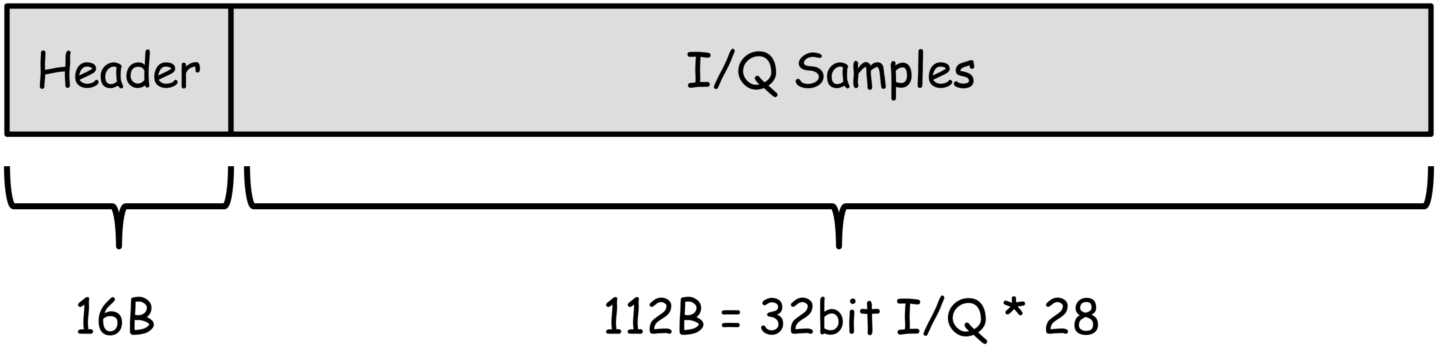
- How Sora sending and receiving are implemented?
- How to use handful tools provided by Sora SDK?
- How to develop a SDR application using Sora User Mode Extension?
- How to program parallel processing using Vector1 library?

Sora Receiving Architecture



Rx Buffer

- Rx Descriptor format



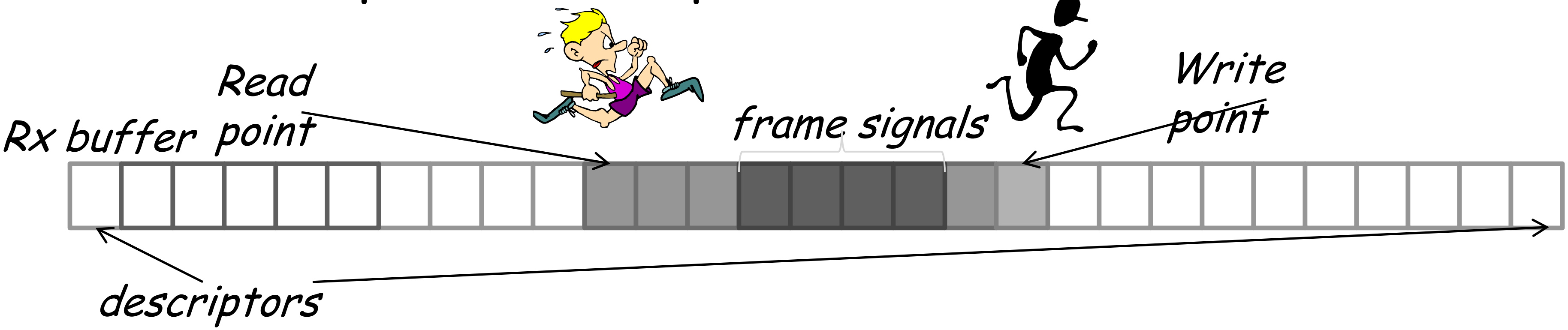
Header

I/Q Samples

16B

112B = 32bit I/Q * 28

- CPU chases behind DMA write point and wait when no more samples left for process



Read point

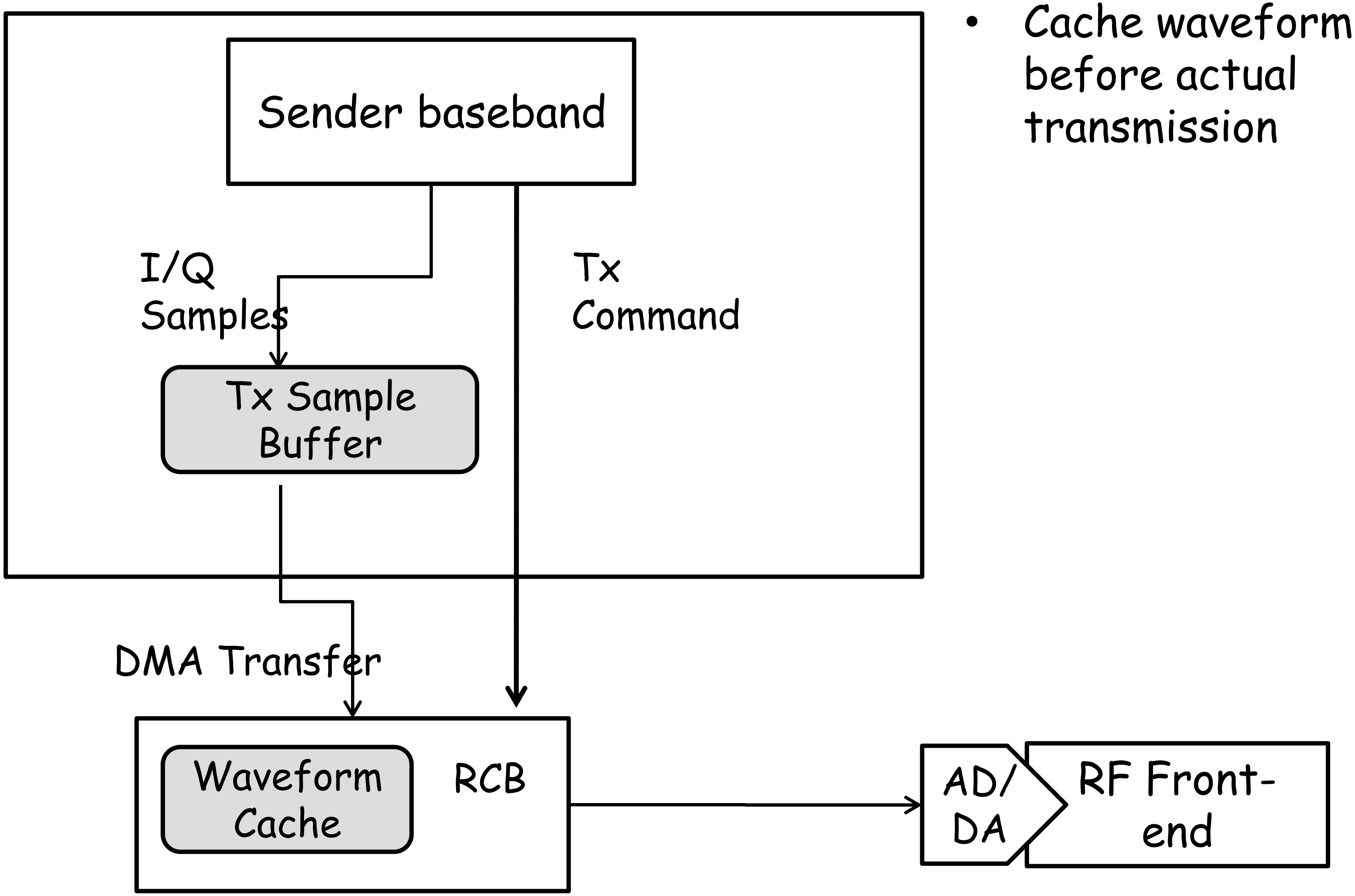
Rx buffer

frame signals

Write point

descriptors

Sora Tx Architecture



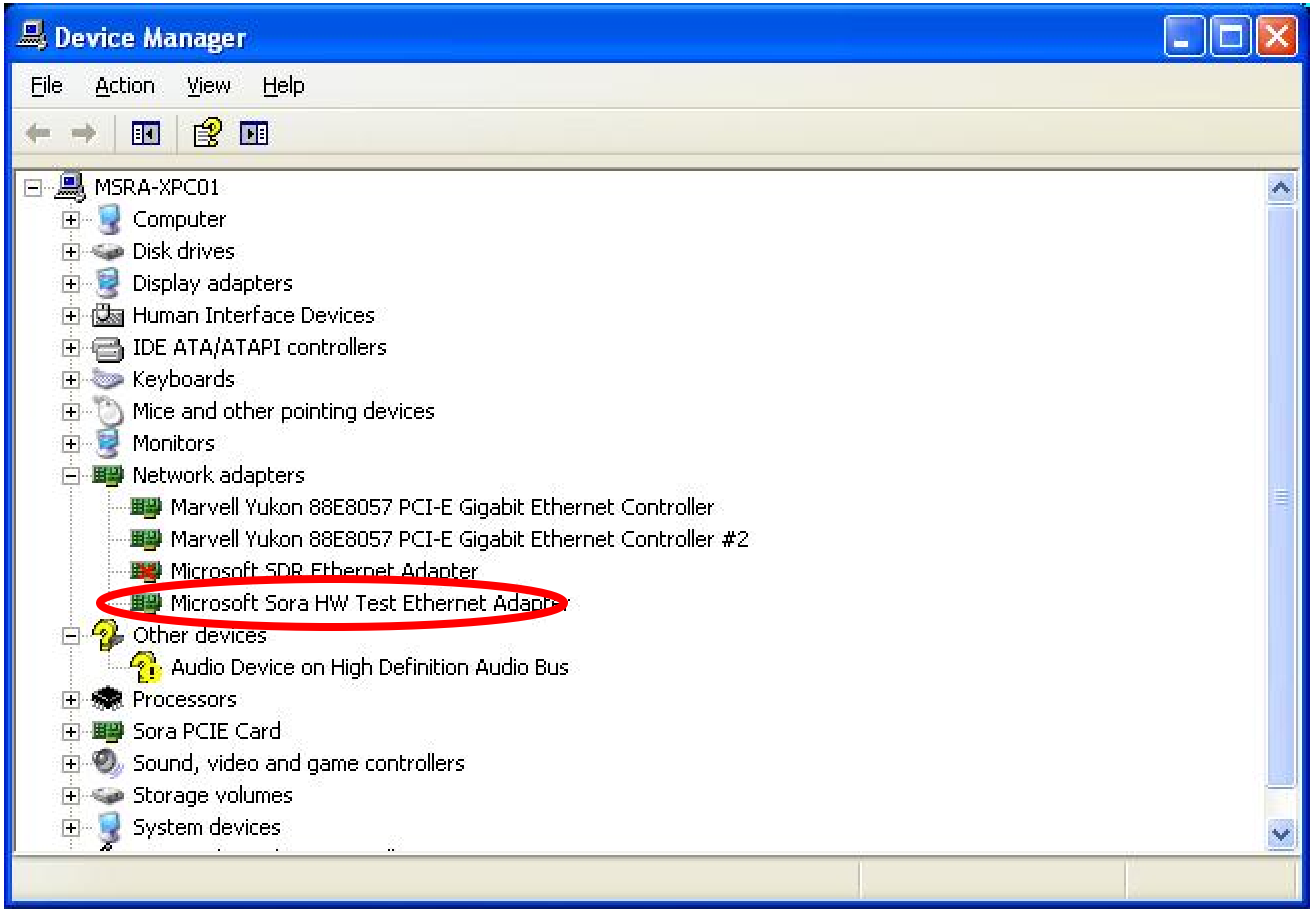
- Cache waveform before actual transmission

What you will learn next?

- How Sora sending and receiving are implemented?
- How to use handful tools provided by Sora SDK?
- How to develop a SDR application using Sora User Mode Extension?
- How to program parallel processing using Vector1 library?

Hardware Unit Test Tool

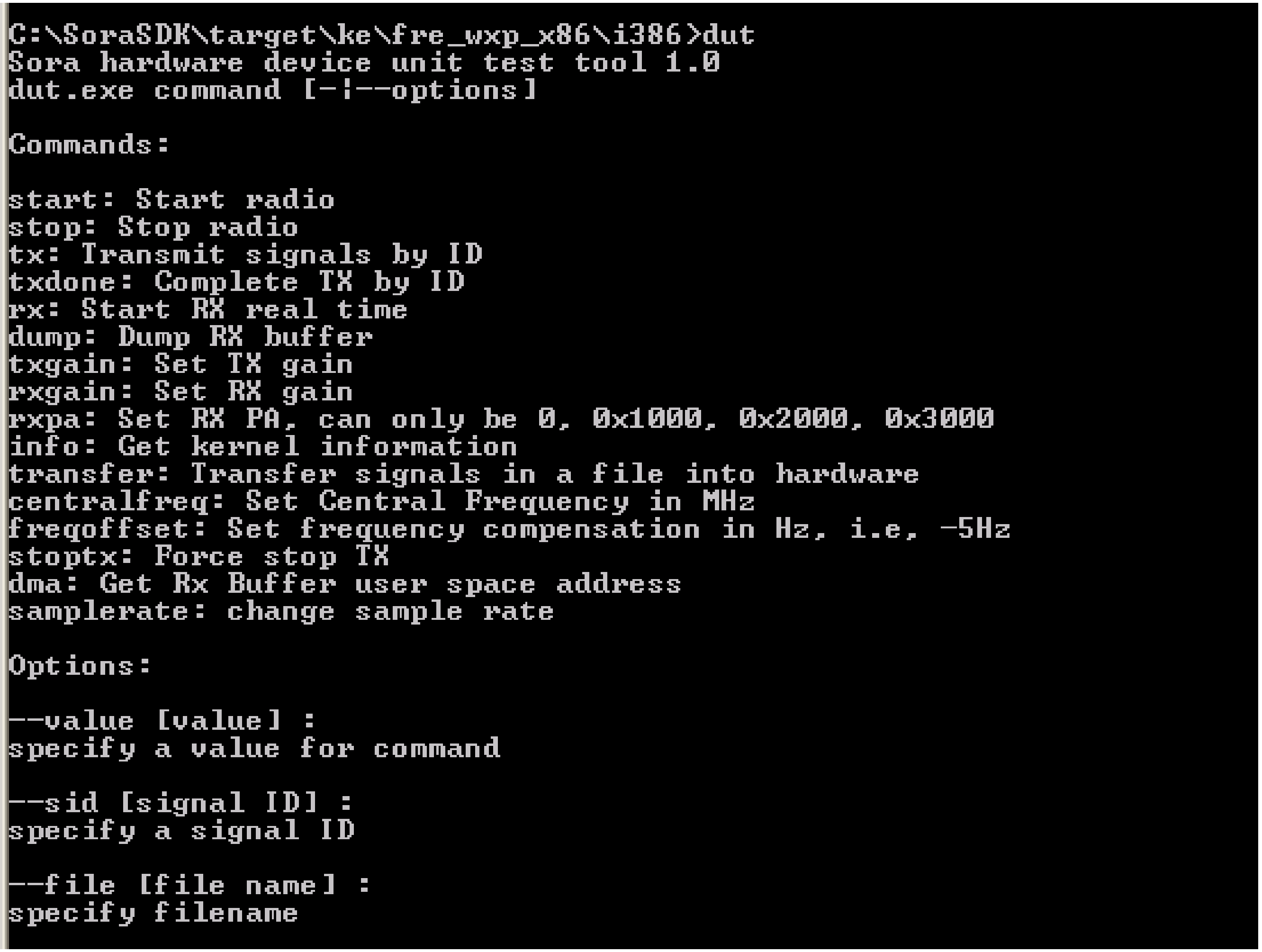
- Install driver



```
MSRA-XPC01
  Computer
  Disk drives
  Display adapters
  Human Interface Devices
  IDE ATA/ATAPI controllers
  Keyboards
  Mice and other pointing devices
  Monitors
  Network adapters
    Marvell Yukon 88E8057 PCI-E Gigabit Ethernet Controller
    Marvell Yukon 88E8057 PCI-E Gigabit Ethernet Controller #2
    Microsoft SDR Ethernet Adapter
    Microsoft Sora HW Test Ethernet Adapter
  Other devices
    Audio Device on High Definition Audio Bus
  Processors
  Sora PCIE Card
  Sound, video and game controllers
  Storage volumes
  System devices
```

Hardware Unit Test Tool (cont.)

- dut.exe



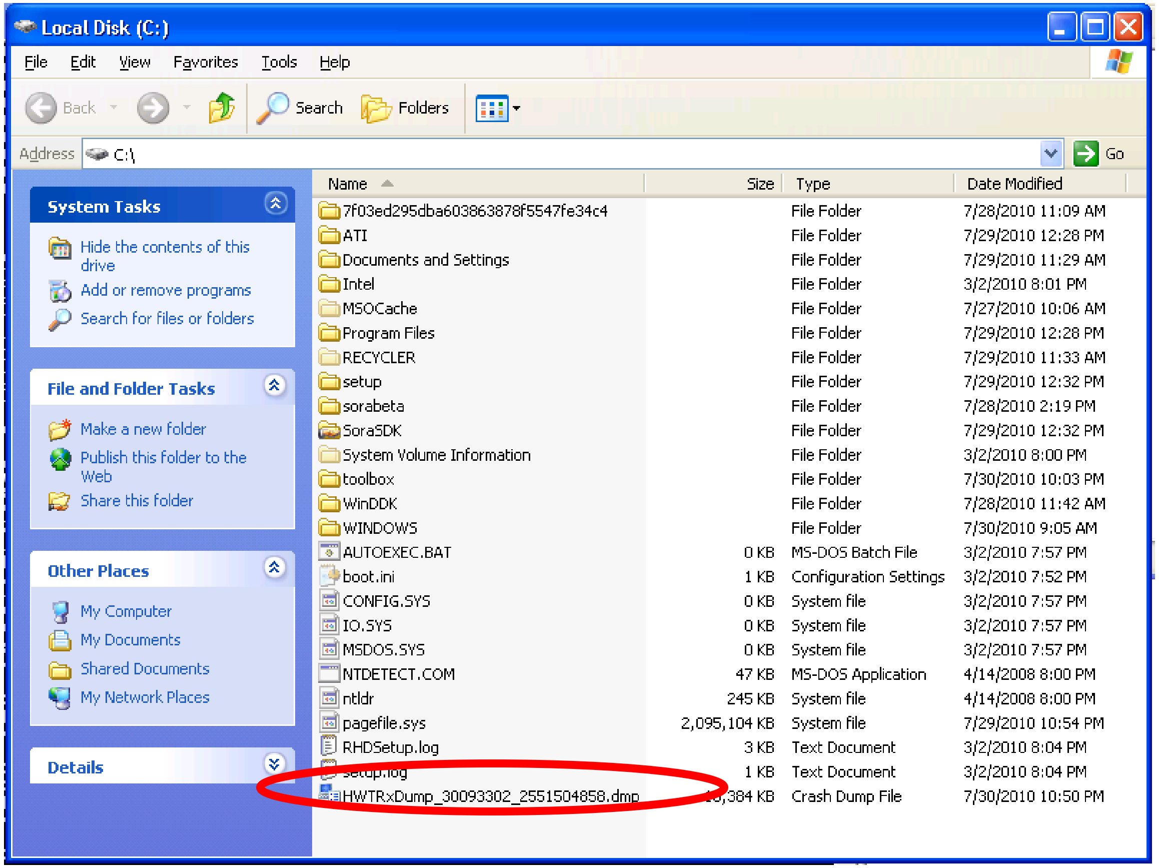
```
C:\SoraSDK\target\ke\fre_wxp_x86\i386>dut
Sora hardware device unit test tool 1.0
dut.exe command [-|--options]

Commands:

start: Start radio
stop: Stop radio
tx: Transmit signals by ID
txdone: Complete TX by ID
rx: Start RX real time
dump: Dump RX buffer
txgain: Set TX gain
rxgain: Set RX gain
rxpa: Set RX PA, can only be 0, 0x1000, 0x2000, 0x3000
info: Get kernel information
transfer: Transfer signals in a file into hardware
centralfreq: Set Central Frequency in MHz
freqoffset: Set frequency compensation in Hz, i.e, -5Hz
stoptx: Force stop TX
dma: Get Rx Buffer user space address
samplerate: change sample rate

Options:

--value [value] :
specify a value for command

--sid [signal ID] :
specify a signal ID

--file [file name] :
specify filename
```

Start Hardware Test

- Start Sora radio
  - dut.exe start
- Start receiving module
  - dut.exe rx
- Set proper gain values
  - Remain Tx gain as default
  - Set Rx gain
  - dut.exe rxpa –-value 0x2000
  - dut.exe rxgain –-value 0x1800

Capture Channel

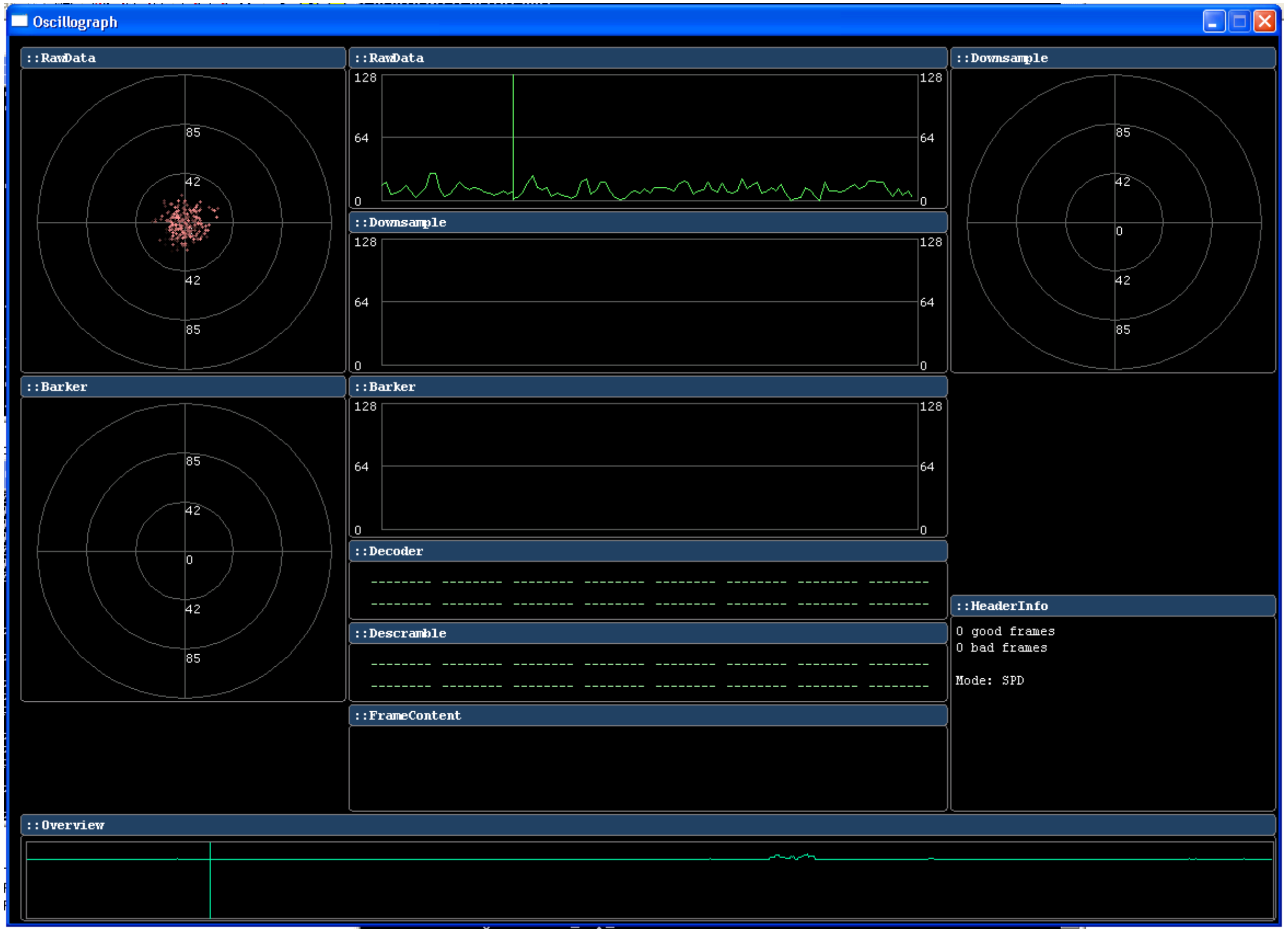
- Run: dut.exe dump
- And you can get a dump file at c:\



Name	Size	Type	Date Modified
7f03ed295dba603863878f5547fe34c4		File Folder	7/28/2010 11:09 AM
ATI		File Folder	7/29/2010 12:28 PM
Documents and Settings		File Folder	7/29/2010 11:29 AM
Intel		File Folder	3/2/2010 8:01 PM
MSOCache		File Folder	7/27/2010 10:06 AM
Program Files		File Folder	7/29/2010 12:28 PM
RECYCLER		File Folder	7/29/2010 11:33 AM
setup		File Folder	7/29/2010 12:32 PM
sorabeta		File Folder	7/28/2010 2:19 PM
SoraSDK		File Folder	7/29/2010 12:32 PM
System Volume Information		File Folder	3/2/2010 8:00 PM
toolbox		File Folder	7/30/2010 10:03 PM
WinDDK		File Folder	7/28/2010 11:42 AM
WINDOWS		File Folder	7/30/2010 9:05 AM
AUTOEXEC.BAT	0 KB	MS-DOS Batch File	3/2/2010 7:57 PM
boot.ini	1 KB	Configuration Settings	3/2/2010 7:52 PM
CONFIG.SYS	0 KB	System file	3/2/2010 7:57 PM
IO.SYS	0 KB	System file	3/2/2010 7:57 PM
MSDOS.SYS	0 KB	System file	3/2/2010 7:57 PM
NTDETECT.COM	47 KB	MS-DOS Application	4/14/2008 8:00 PM
ntldr	245 KB	System file	4/14/2008 8:00 PM
pagefile.sys	2,095,104 KB	System file	7/29/2010 10:54 PM
RHDSetup.log	3 KB	Text Document	3/2/2010 8:04 PM
setup.log	1 KB	Text Document	3/2/2010 8:04 PM
HWTRxDump_30093302_2551504858.dmp	16,384 KB	Crash Dump File	7/30/2010 10:50 PM

Analyze Dump File Offline

- File Format: a direct image of the Rx DMA buffer



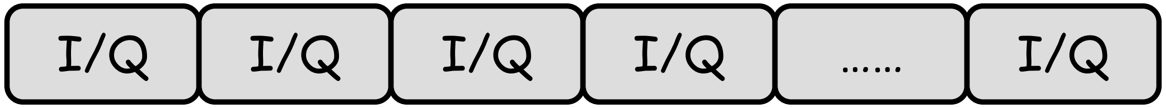
```
-------- -------- -------- -------- -------- -------- -------- --------
-------- -------- -------- -------- -------- -------- -------- --------
```

```
-------- -------- -------- -------- -------- -------- -------- --------
-------- -------- -------- -------- -------- -------- -------- --------
```

```
0 good frames
0 bad frames

Mode: SPD
```

Signal Generator

- Prepare a signal file using any favorite tool
- File format

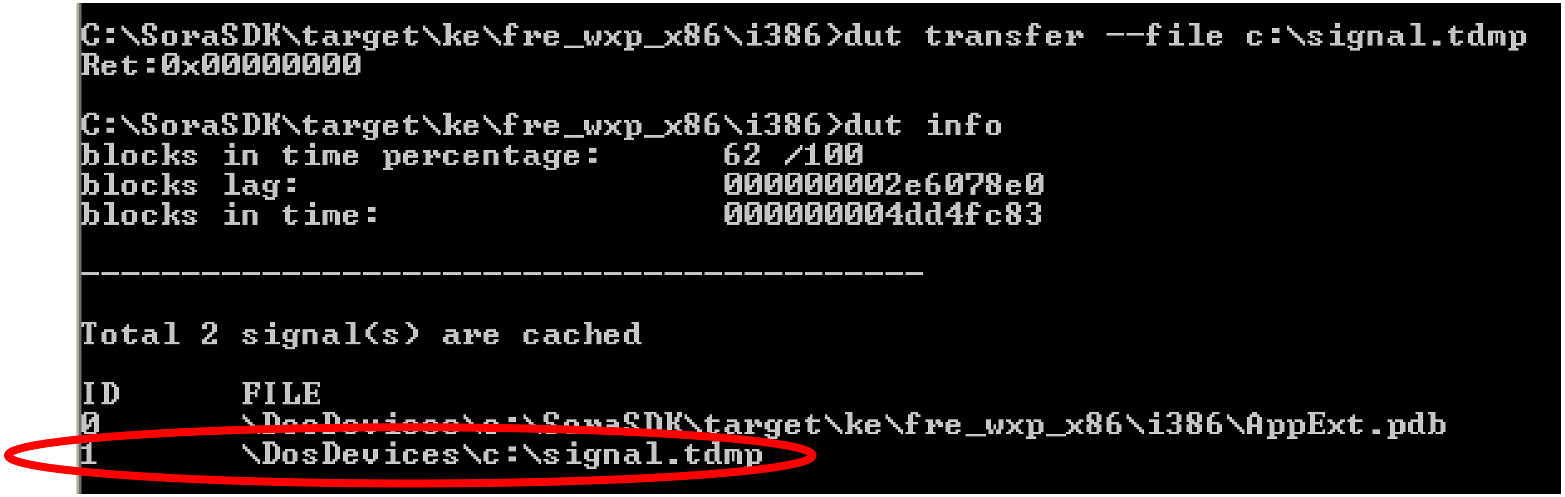


I/Q I/Q I/Q I/Q …… I/Q

  - Each I/Q sample is 16 bit complex number (I-8b, Q-8b)
  - For some history reasons, subject to change in later version

Load Signal File

- dut.exe transfer --file d:\signal.tdmp
- Note: the filename must be full path!
- Use dut.exe info tduto check the status



```
C:\SoraSDK\target\ke\fre_wxp_x86\i386>dut transfer --file c:\signal.tdmp
Ret:0x00000000

C:\SoraSDK\target\ke\fre_wxp_x86\i386>dut info
blocks in time percentage:      62 /100
blocks lag:                     000000002e6078e0
blocks in time:                 000000004dd4fc83

------------------------------------------

Total 2 signal(s) are cached

ID      FILE
0       \DosDevices\c:\SoraSDK\target\ke\fre_wxp_x86\i386\AppExt.pdb
1       \DosDevices\c:\signal.tdmp
```

Transmit Your Signal

- Use dut.exe tx --sid 0x01
- Write a batch commend to repeat transmission
  - E.g. transmit for 100 times
  - for /L %i in (1,1,100) do dut.exe tx –sid 0x01

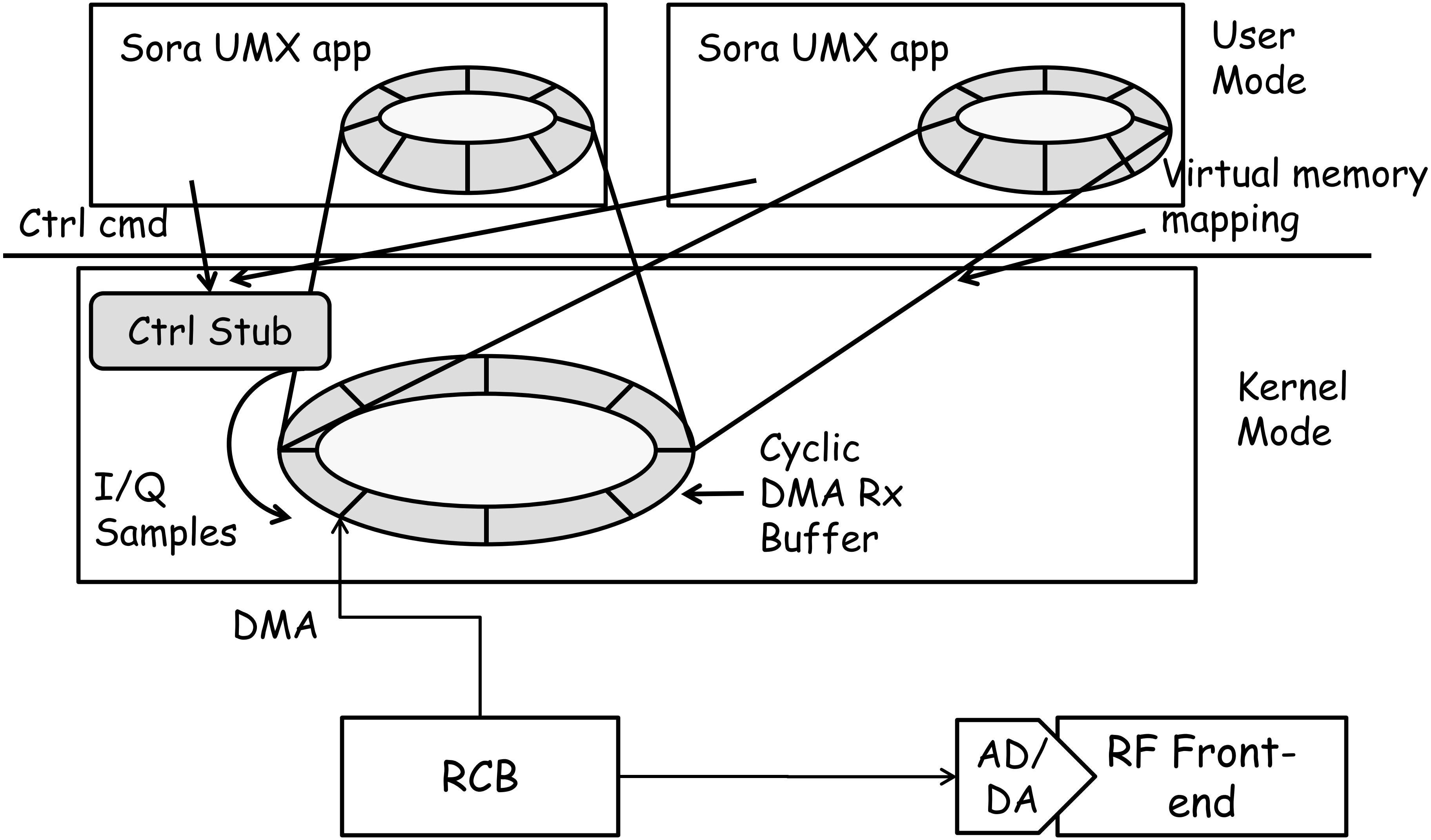
What you will learn next?

- How Sora sending and receiving are implemented?
- How to use handful tools provided by Sora SDK?
- How to develop a SDR application using Sora User Mode Extension?
- How to program parallel processing using Vector1 library?

Sora User Mode Extension

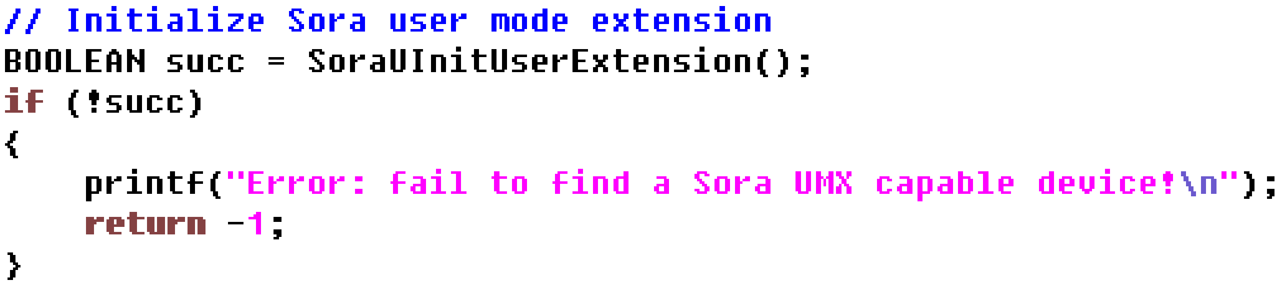
- Quick way to compose a SDR application w/o losing too much performance
- Pros:
  - Easy programming
  - Easy debugging
  - Low overhead in receiving samples
  - Multiple streams virtualization
  - Core dedication support
- Cons:
  - Long latency in control path
  - No integration in network stack

Sora UMX Architecture



Initialize and Deinitialize UMX

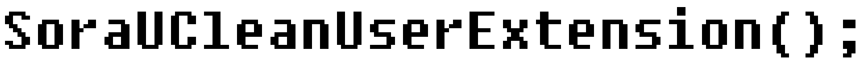
- Initialize UMX



```
// Initialize Sora user mode extension
BOOLEAN succ = SoraUInitUserExtension();
if (!succ)
{
    printf("Error: fail to find a Sora UMX capable device!\n");
    return -1;
}
```

- Receive or transmit data

- Clean up UMX



```
SoraUCleanUserExtension();
```

Receiving using UMX

- Map Rx buffer to user mode

```
//
// Map Rx Buffer
//
hr = SoraURadioMapRxBuffer( TARGET_RADIO, // radio id
                            & pRxBuf,     // mapped buffer pointer
                            & nRxBufSize,     // size of mapped buffer
                            & nMask       // System Reserved Parameter. Do not modify
                          );

if ( FAILED (hr) ) {
    printf ( "Error: Fail to map Rx buffer!\n" );
    return;
}
```

- Bind a RxStream object to Rx buffer

```
// Generate a sample stream from mapped Rx buffer
SoraGenRadioRxStream( &SampleStream,
                      (PUCHAR)pRxBuf,
                      nRxBufSize);
```

Receiving using UMX (cont.)

- Get a sample block through RxStream object

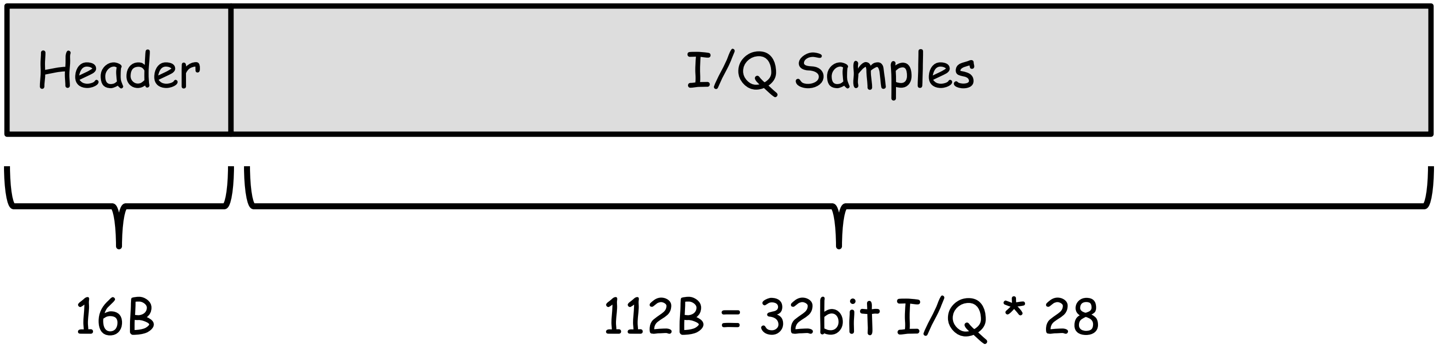
```
// start reading the sample stream and compute the energy
bRxStop = FALSE;
FLAG fReachEnd;

int index = 0;
PRX_BLOCK pRxBlock = (PRX_BLOCK) SoraGetRadioCurrentScanPoint ( & SampleStream );
```

- Wait for the block to be valide and process on it

```
while ( !bRxStop ) {

    hr = SoraUWaitMarkSignalBlock ( pRxBlock,      // current scan point
                                    100,           // how many retries before return
                                    & fReachEnd,   // indicate if end of stream reached (you must wait for hardware)
                                    nMask          // parameter return from SoraURadioMapRxBuffer
                                  );

    {
        // do something here

        // get next block
        pRxBlock = (PRX_BLOCK) SoraGetRadioNextScanPoint ( & SampleStream, (PUCHAR) pRxBlock );
    }


    FAILED_BREAK(hr);
}
```

Receiving using UMX (cont.)

- Sample bock
  - An image of a Rx Descriptor



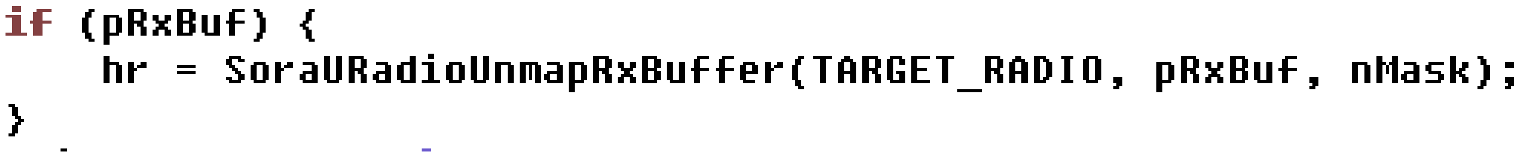
Header

I/Q Samples

16B

112B = 32bit I/Q * 28

  - 28 samples per block
- Clean up after you done



```
if (pRxBuf) {
    hr = SoraURadioUnmapRxBuffer(TARGET_RADIO, pRxBuf, nMask);
}
```

Sending using UMX

- Map modulation buffer to user mode

```
hr = SoraURadioMapModBuffer(
        TARGET_RADIO, //#0 radio, always 0 currently
        &SampleBuffer,
        &SampleBufferSize);
printf("map mod buffer ret: %08x\n", hr);
```

- Fill up the buffer with your own sample values

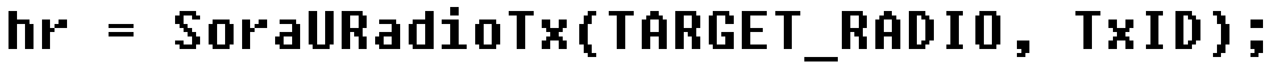
```
ULONG SigLength = PrepareSamples(fname, (char*)SampleBuffer, SampleBufferSize);
```

- Allocate RCB resource and transfer samples to RCB

```
//
// First Allocate Tx Resource
// The size should be a multiple of 128
//
ALIGN_WITH_RCB_BUFFER(SigLength);
hr = SoraURadioTxResAllocate(TARGET_RADIO, SigLength, &TxID);
```

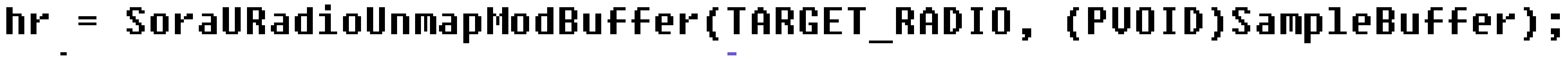
Sending using UMX (cont.)

- Instruct RCB to emit your waveform



```
hr = SoraURadioTx(TARGET_RADIO, TxID);
```

- Clean up



```
hr = SoraURadioUnmapModBuffer(TARGET_RADIO, (PVOID)SampleBuffer);
```

More information: UMX_sample at $(sorasdk)\src\AppExt

What you will learn next?

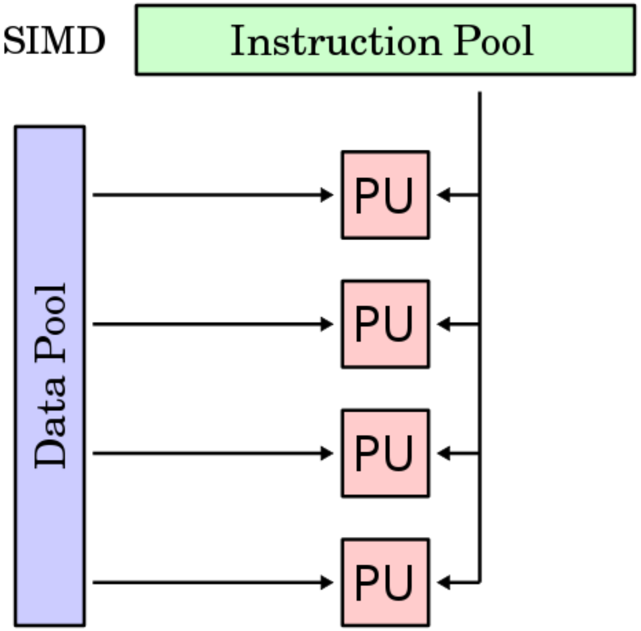
- How Sora sending and receiving are implemented?
- How to use handful tools provided by Sora SDK?
- How to develop a SDR application using Sora User Mode Extension?
- How to program parallel processing using Vector1 library?

Programming with Vector1 Library

- M. Flynn's taxonomy on computer architectures

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

- SIMD to handle data parallel
  - Operate on vectors of parallel data
  - Inexpensive compared MIMD
  - Good enough for many apps.
- Common SIMD H/W
  - Intel SSE/AVX; IBM AltiVec
  - GPU



Vector1 Library

- An abstract library to support SIMD programming
- High-level syntax and integration to C++
  - Hiding significant details, e.g. register allocation
  - Advanced operation abstractions
  - Type-checking
- Platform independent
  - Current implementation on Intel SSE
  - Portable to other platform
- Highly efficient code generation
  - Integrated with intrinsic of modern C++ compiler
  - Same/better performance compared to hand-made optimization

Vector1 Data Type

- Vector size 128b.
- Must be 16 aligned!

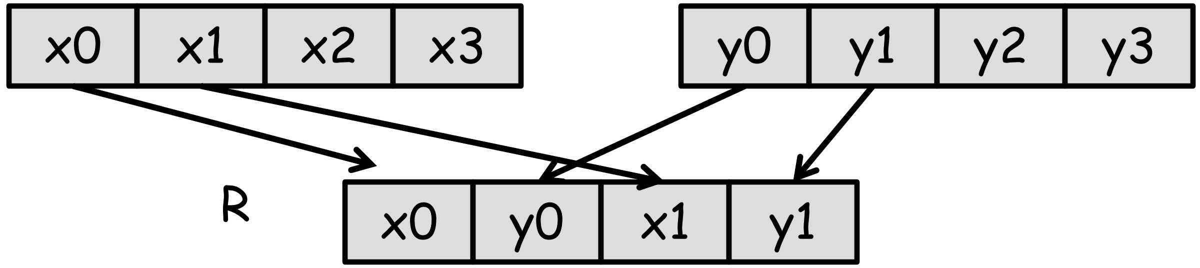
	Element Type	Element Size	Vector Length
vb/vub	Byte (unsigned byte)	8b	16
vs/vus	short(unsigned short)	16b	8
vi/vui	Int (unsigned int)	32b	4
vf	Float	32b	4
vcb/vcub	Complex (unsigned) byte	16b	8
vcs/vcus	Complex (unsigned) short	32b	4
vci/vcui	Complex (unsigned) int	64b	2
vcf	Complex float	64b	2

Vector1 Operations

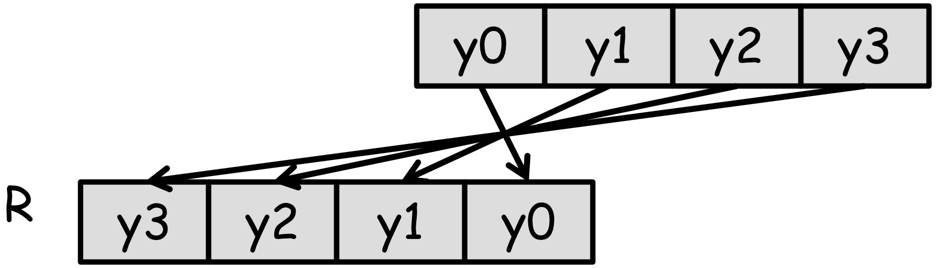
- Arithmetic operations
  - Add/saturate_add
  - Sub/saturate_sub
  - mul_low/mul_high
    - Element-wise multiple and with low (high) part of the results
  - conj_mul
    - Complex multiply with a complex conjuncture
- Max/Min
  - smax/smin
    - Obtain element-wise max/min value
- Absolute value
  - Abs
- Initialization
  - Set_zero/set_all

Vector1 Operations (cont.)

- Interleave
  - Interleave_low/interleave_high
    - R = interleave_low (x, y);



- Permutation
  - permutation<idx>
    - R = permutation<3,2,1,0> (x)

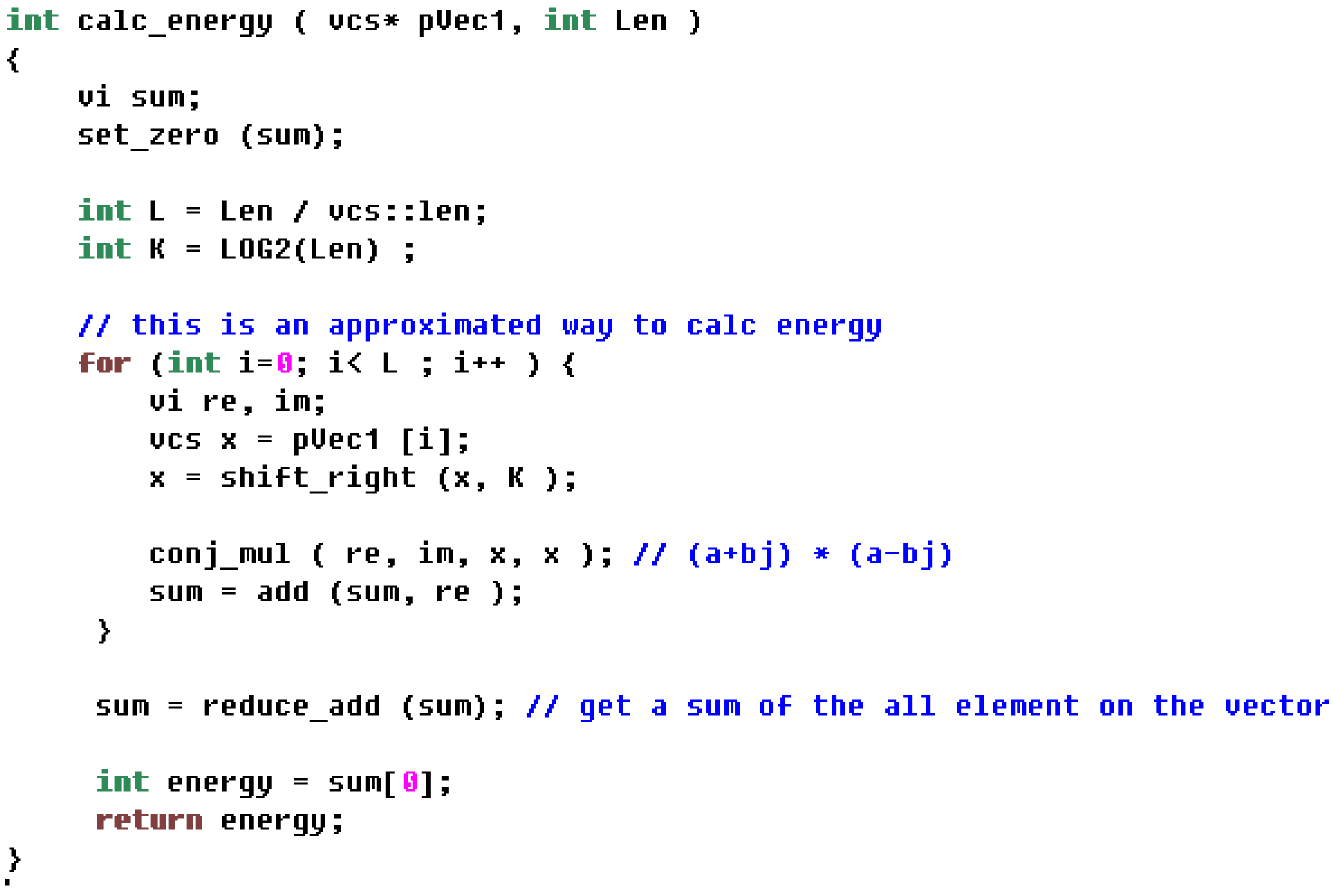


Vector1 Sample 1 - Calculate Energy

- Algorithm

$$E = Re(\vec{x} \cdot \vec{x}^*) = Re(\textstyle\sum_0^L x_i \cdot {x^*}_i)$$

- Code



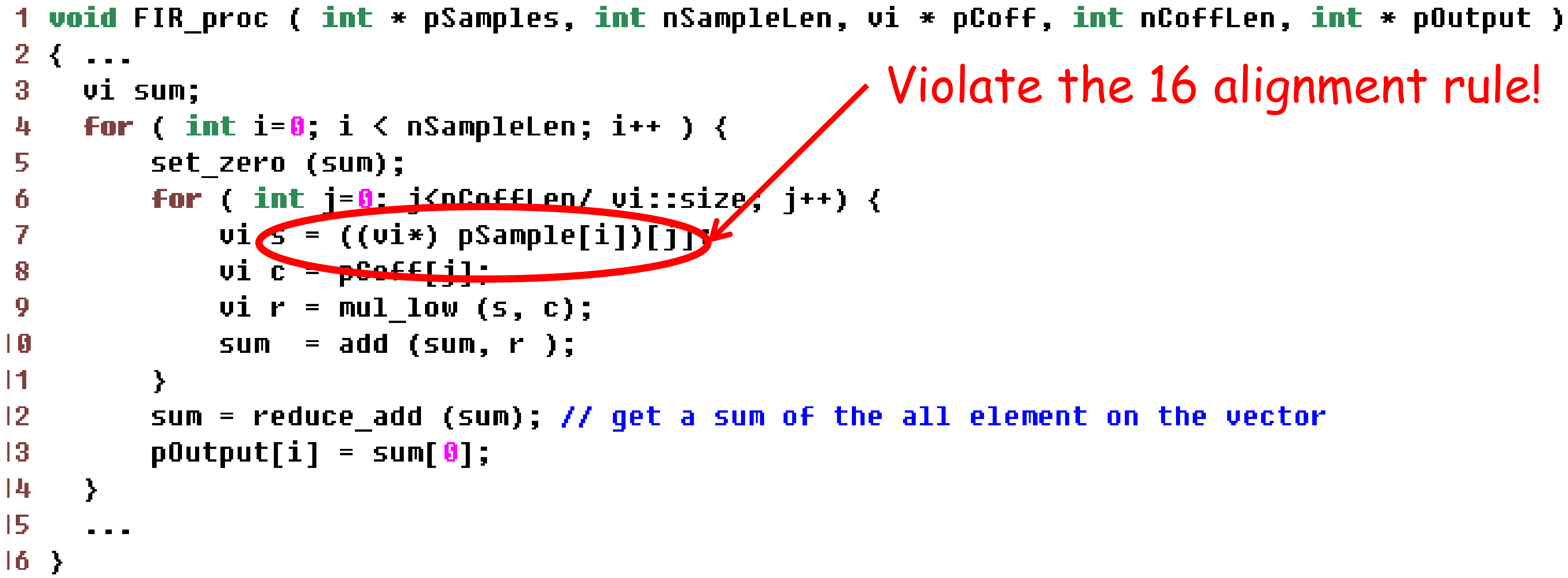
```
int calc_energy ( vcs* pVec1, int Len )
{
    vi sum;
    set_zero (sum);

    int L = Len / vcs::len;
    int K = LOG2(Len) ;

    // this is an approximated way to calc energy
    for (int i=0; i< L ; i++ ) {
        vi re, im;
        vcs x = pVec1 [i];
        x = shift_right (x, K );

        conj_mul ( re, im, x, x ); // (a+bj) * (a-bj)
        sum = add (sum, re );
     }

     sum = reduce_add (sum); // get a sum of the all element on the vector

     int energy = sum[0];
     return energy;
}
```

Vector1 Sample 2 - FIR Filter

- Algorithm

$$y[k] = \sum_{j=0}^{L} c_j \cdot x[k-j]$$

- One quick idea

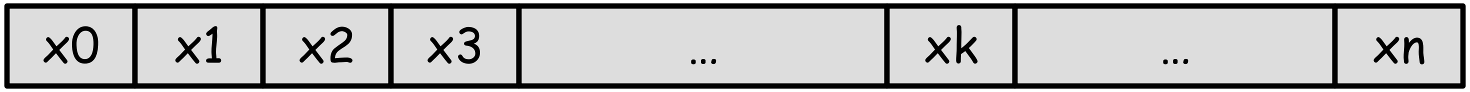


```
void FIR_proc ( int * pSamples, int nSampleLen, vi * pCoff, int nCoffLen, int * pOutput )
{ ...
  vi sum;
  for ( int i=0; i < nSampleLen; i++ ) {
      set_zero (sum);
      for ( int j=0; j<nCoffLen/ vi::size; j++) {
          vi s = ((vi*) pSample[i])[j];
          vi c = pCoff[j];
          vi r = mul_low (s, c);
          sum  = add (sum, r );
      }
      sum = reduce_add (sum); // get a sum of the all element on the vector
      pOutput[i] = sum[0];
  }
  ...
}
```

Violate the 16 alignment rule!

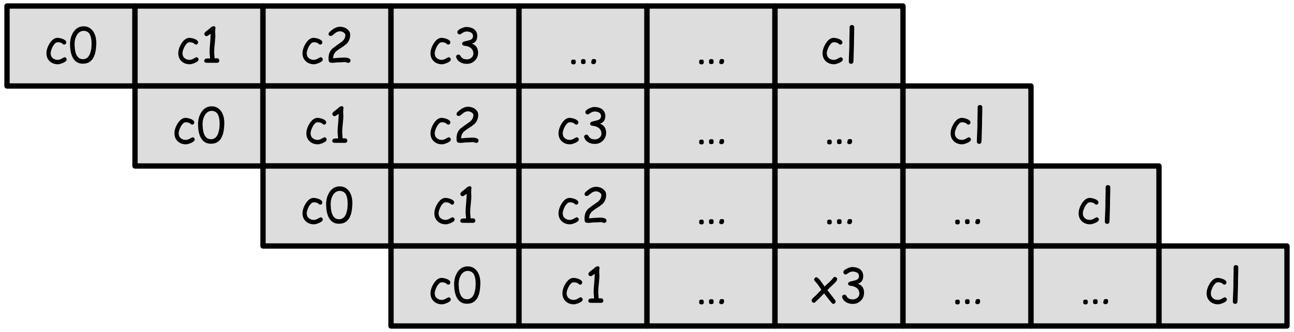
Vector1 Sample 2 – FIR Filter (cont.)

- A better way



x0	x1	x2	x3	…	xk	…	xn

Use multiple coefficient bands

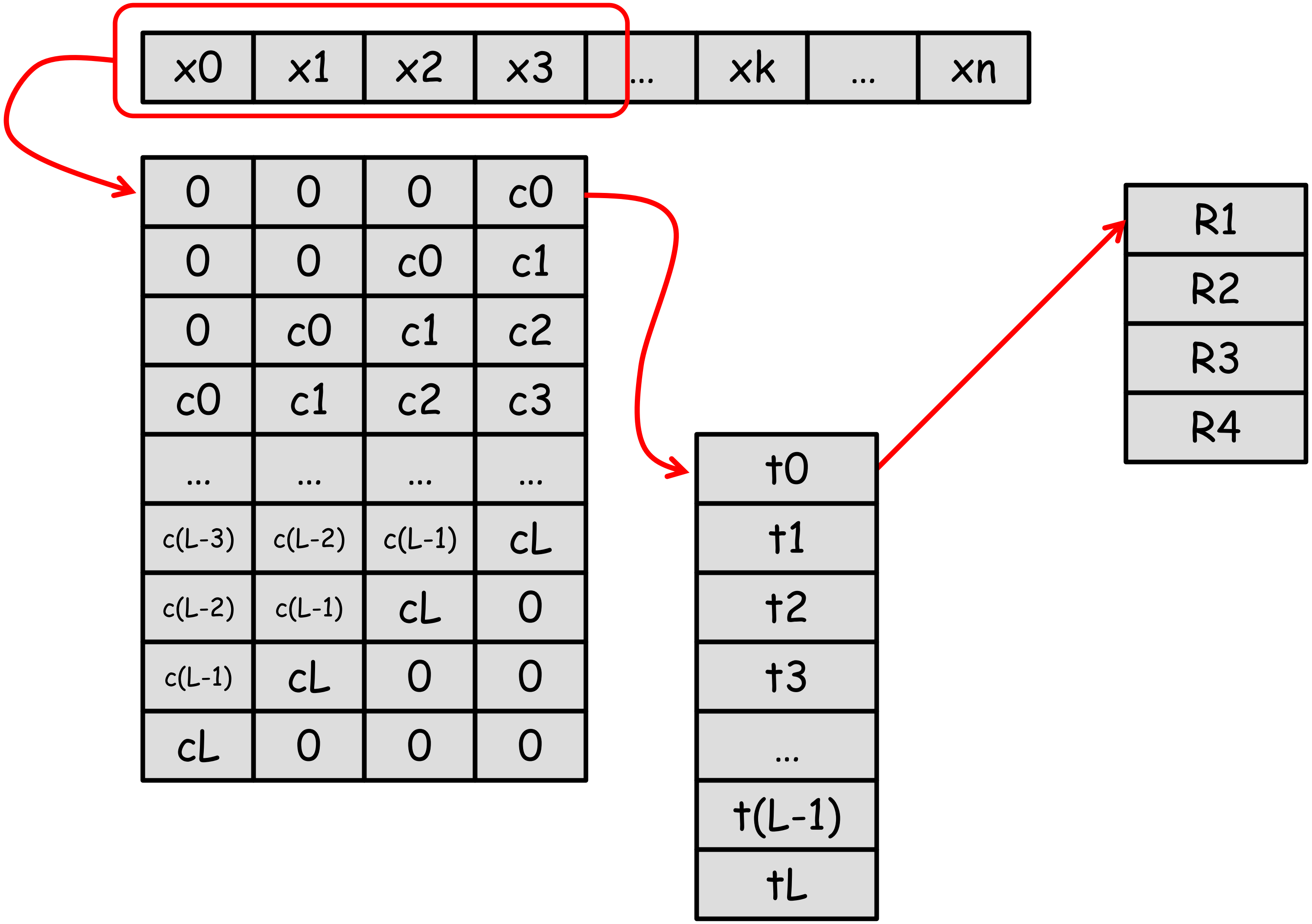


c0	c1	c2	c3	…	…	cl			
	c0	c1	c2	c3	…	…	cl		
		c0	c1	c2	…	…	…	cl	
			c0	c1	…	x3	…	…	cl

Avoid 16 alignment issue, but any more issue?

Vector1 Sample 2 – FIR Filter (cont.)

- Multiple coefficient bands w/ cache tilting



x0	x1	x2	x3	…	xk	…	xn

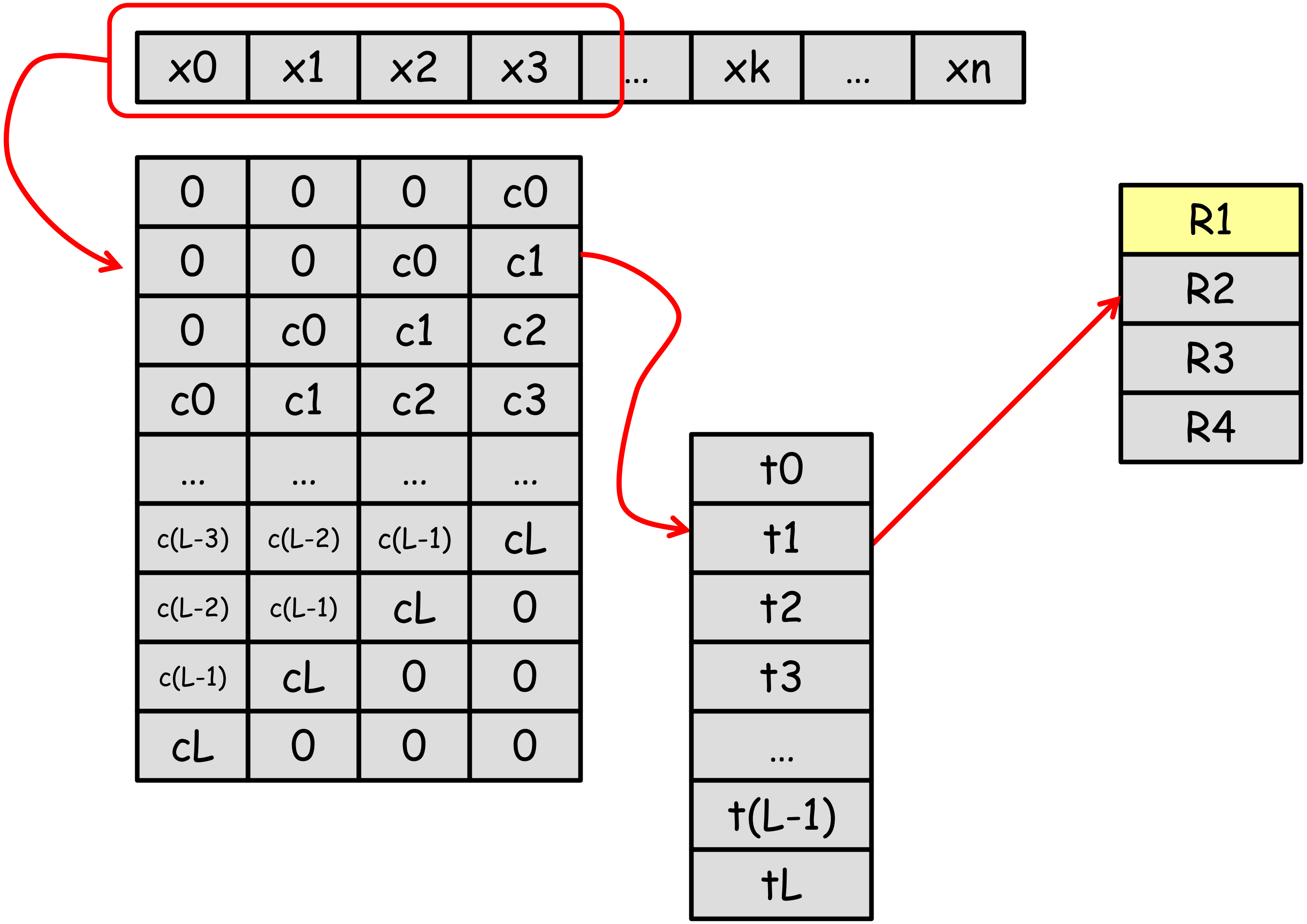
0	0	0	c0
0	0	c0	c1
0	c0	c1	c2
c0	c1	c2	c3
…	…	…	…
c(L-3)	c(L-2)	c(L-1)	cL
c(L-2)	c(L-1)	cL	0
c(L-1)	cL	0	0
cL	0	0	0

t0
t1
t2
t3
…
t(L-1)
tL

R1
R2
R3
R4

Vector1 Sample 2 – FIR Filter (cont.)

- Multiple coefficient bands w/ cache tilting



x0	x1	x2	x3	…	xk	…	xn

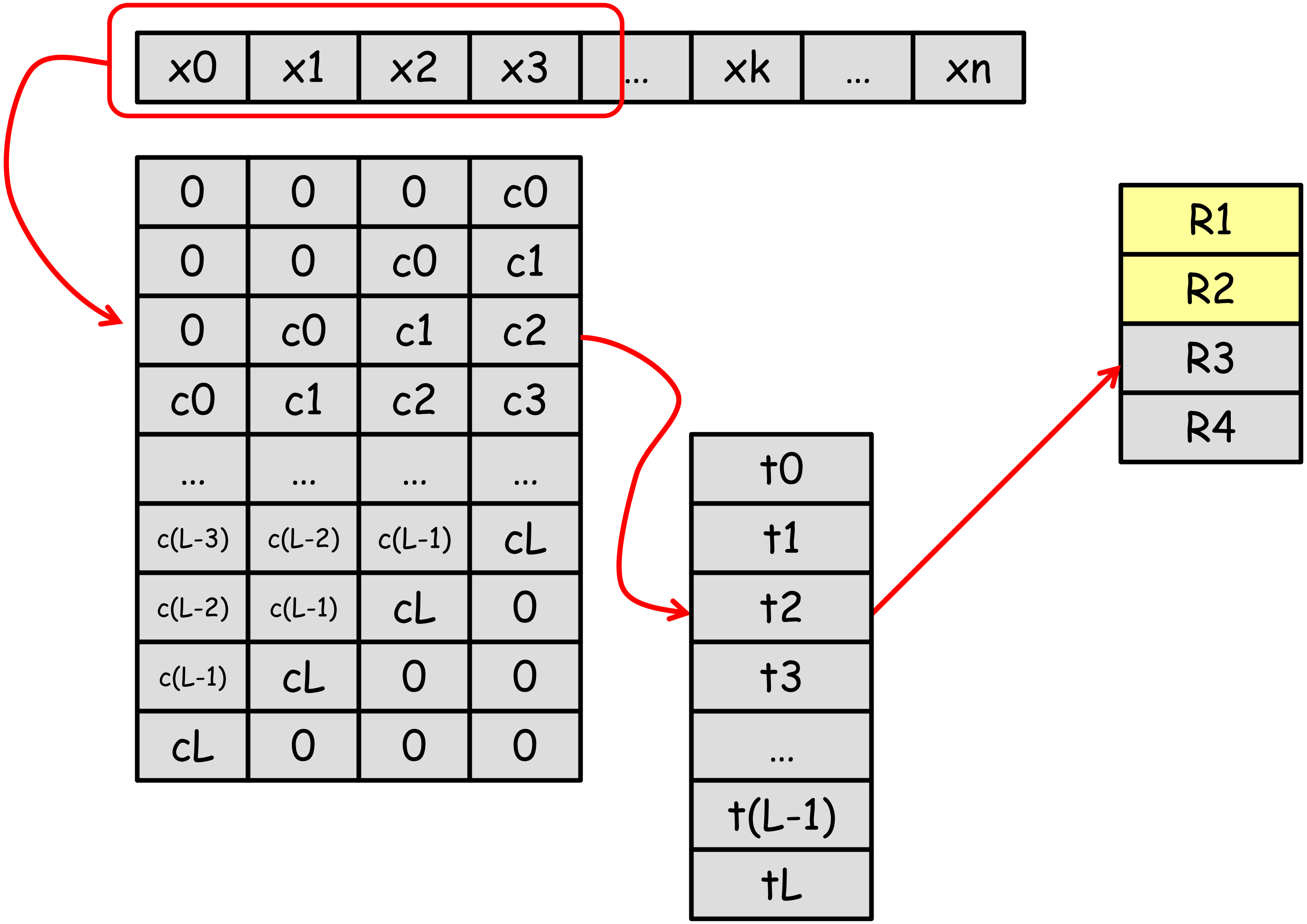
0	0	0	c0
0	0	c0	c1
0	c0	c1	c2
c0	c1	c2	c3
…	…	…	…
c(L-3)	c(L-2)	c(L-1)	cL
c(L-2)	c(L-1)	cL	0
c(L-1)	cL	0	0
cL	0	0	0

t0
t1
t2
t3
…
t(L-1)
tL

R1
R2
R3
R4

Vector1 Sample 2 – FIR Filter (cont.)

- Multiple coefficient bands w/ cache tilting



x0	x1	x2	x3	…	xk	…	xn

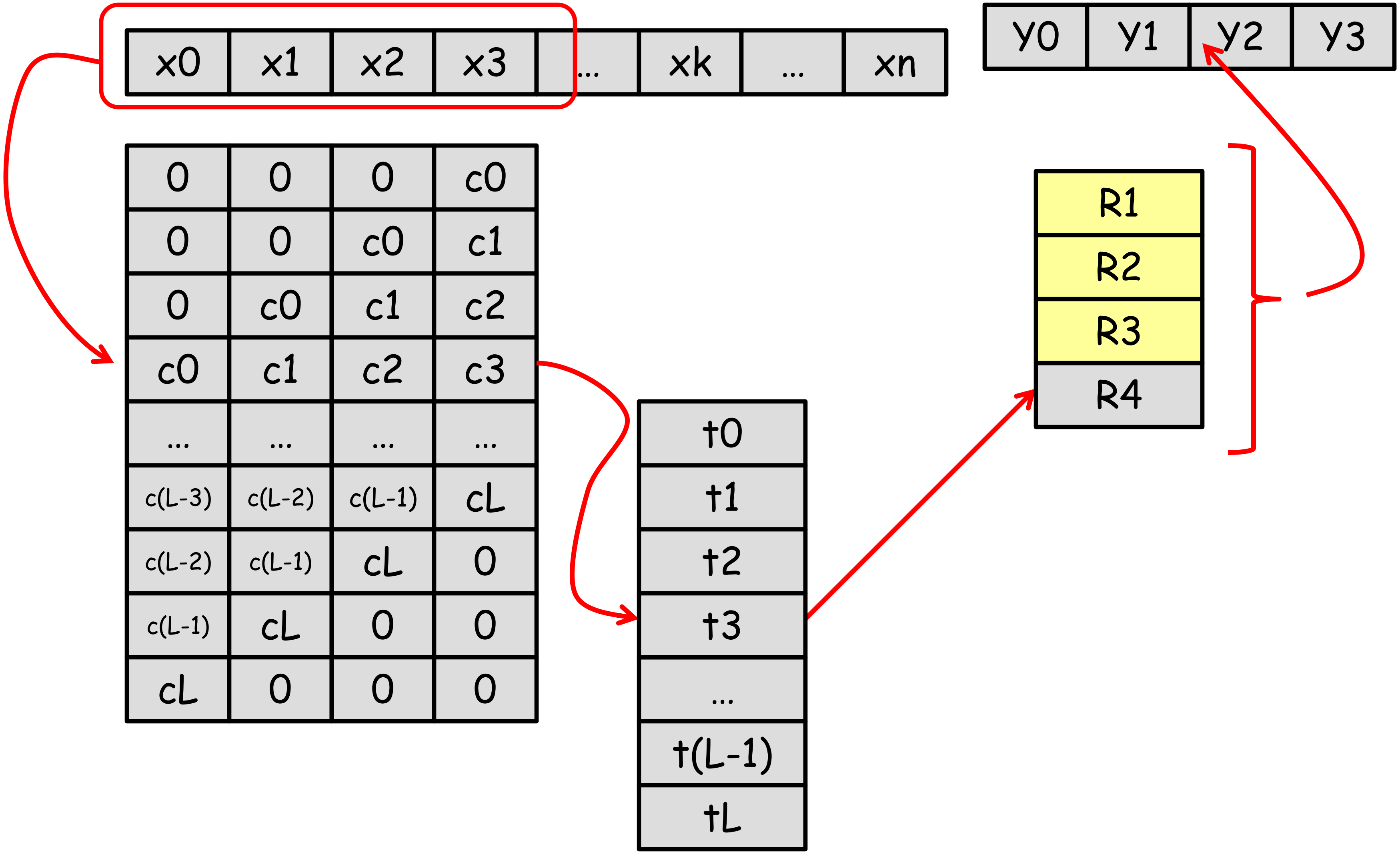
0	0	0	c0
0	0	c0	c1
0	c0	c1	c2
c0	c1	c2	c3
…	…	…	…
c(L-3)	c(L-2)	c(L-1)	cL
c(L-2)	c(L-1)	cL	0
c(L-1)	cL	0	0
cL	0	0	0

t0
t1
t2
t3
…
t(L-1)
tL

R1
R2
R3
R4

Vector1 Sample 2 – FIR Filter (cont.)

- Multiple coefficient bands w/ cache tilting



x0	x1	x2	x3	…	xk	…	xn

y0	y1	y2	y3

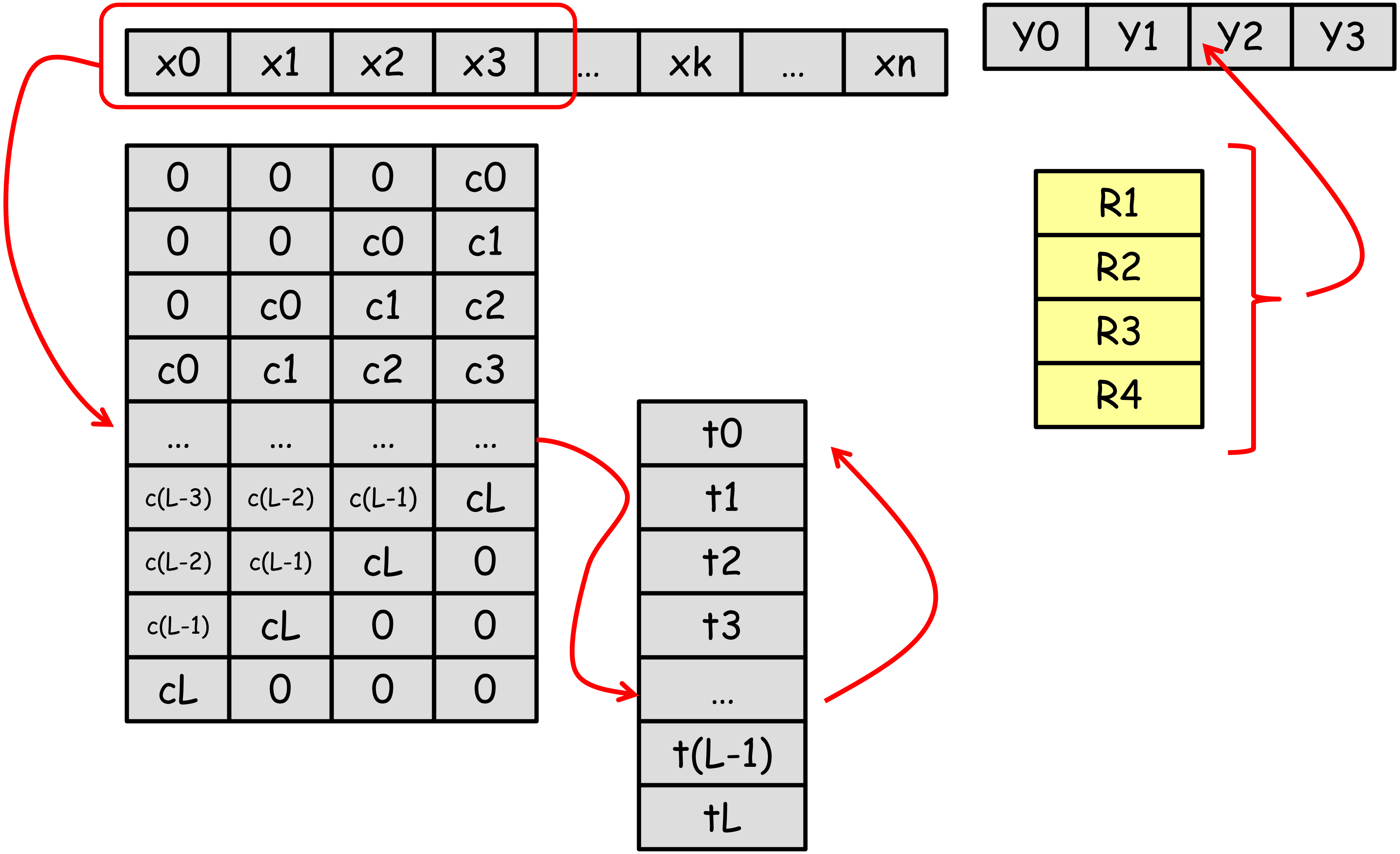
0	0	0	c0
0	0	c0	c1
0	c0	c1	c2
c0	c1	c2	c3
…	…	…	…
c(L-3)	c(L-2)	c(L-1)	cL
c(L-2)	c(L-1)	cL	0
c(L-1)	cL	0	0
cL	0	0	0

t0
t1
t2
t3
…
t(L-1)
tL

R1
R2
R3
R4

Vector1 Sample 2 – FIR Filter (cont.)

- Multiple coefficient bands w/ cache tilting



x0	x1	x2	x3	…	xk	…	xn

y0	y1	y2	y3

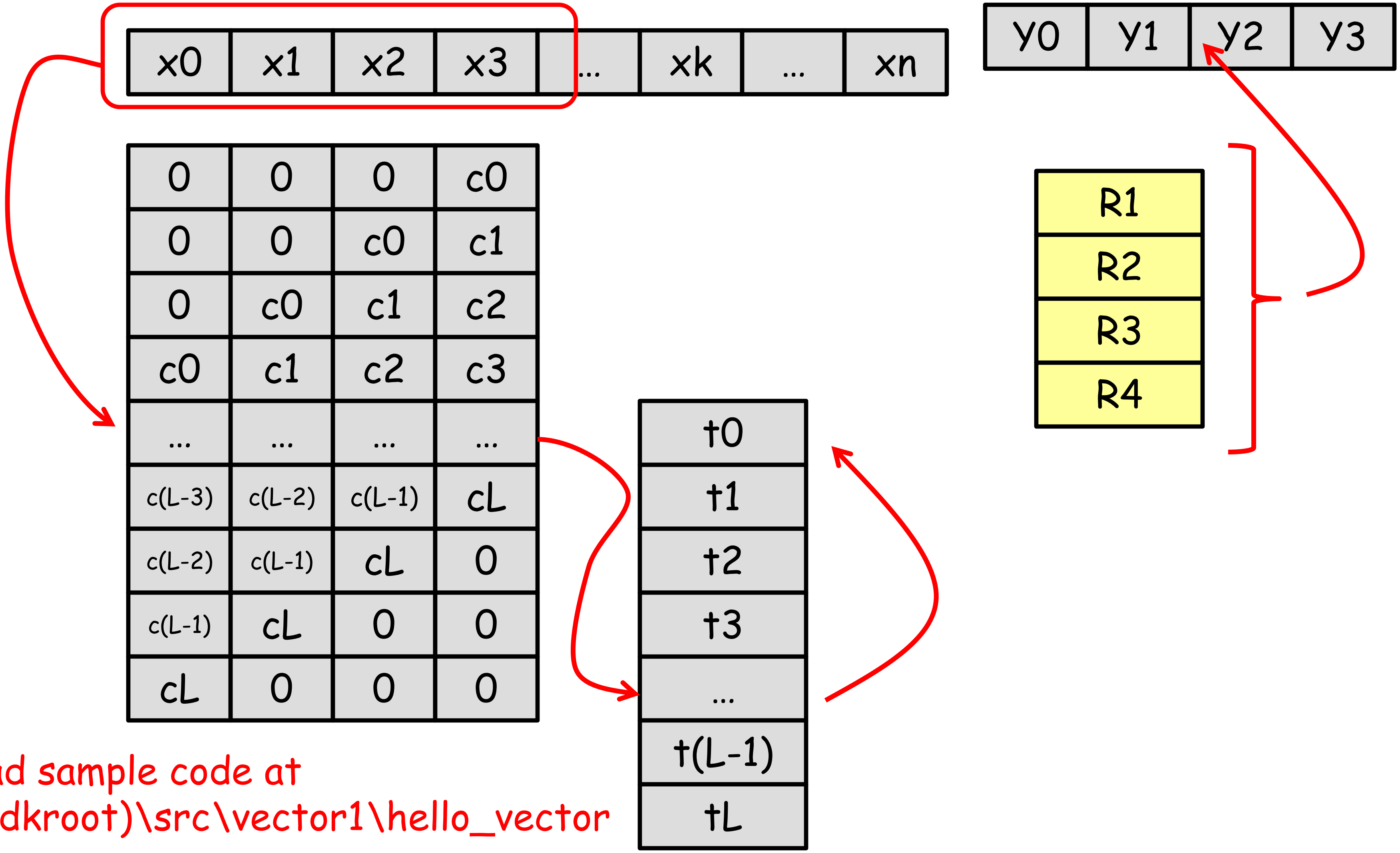
0	0	0	c0
0	0	c0	c1
0	c0	c1	c2
c0	c1	c2	c3
…	…	…	…
c(L-3)	c(L-2)	c(L-1)	cL
c(L-2)	c(L-1)	cL	0
c(L-1)	cL	0	0
cL	0	0	0

t0
t1
t2
t3
…
t(L-1)
tL

R1
R2
R3
R4

Vector1 Sample 2 – FIR Filter (cont.)

- Multiple coefficient bands w/ cache tilting



x0	x1	x2	x3	…	xk	…	xn

y0	y1	y2	y3

0	0	0	c0
0	0	c0	c1
0	c0	c1	c2
c0	c1	c2	c3
…	…	…	…
c(L-3)	c(L-2)	c(L-1)	cL
c(L-2)	c(L-1)	cL	0
c(L-1)	cL	0	0
cL	0	0	0

t0
t1
t2
t3
…
t(L-1)
tL

R1
R2
R3
R4

Read sample code at
$(sdkroot)\src\vector1\hello_vector

Summary

- Vector1 reduces the efforts to program SIMD
- But it is still in the development stage
  - Try it now and understand it
  - Write your code with it and test performance
  - Give us feedbacks and your suggestion may be reflected in later release! ☺

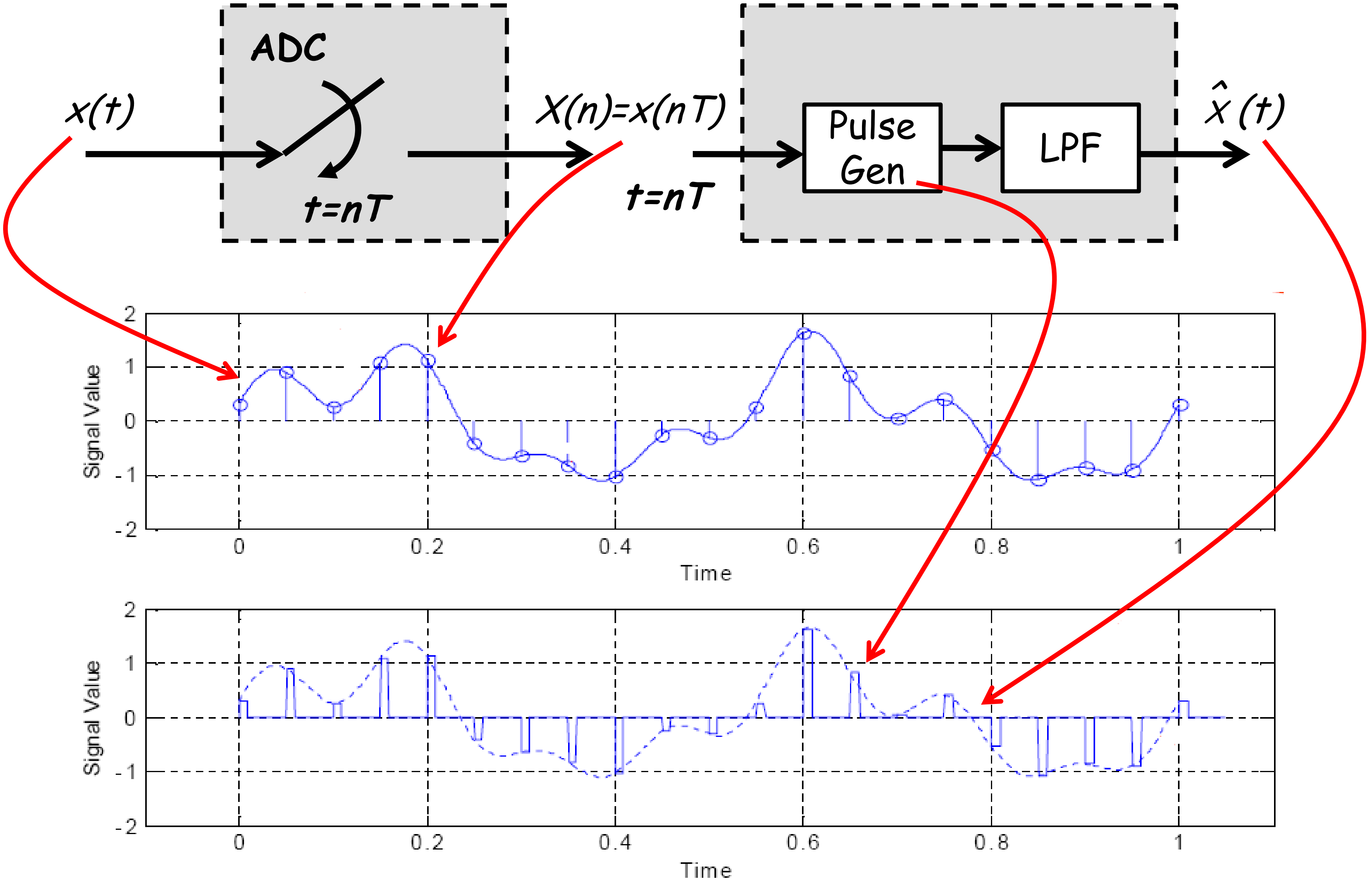
Digital Wireless Communication Illustrated

Kun Tan
Wireless and Networking Group
MSR Asia

Outline

- Communication basis
  - Sampling Theory and System
  - Time Domain and Frequency Domain
- Overview of Digital Communication System
- Case study
  - Spectrum spreading – 802.11b
  - OFDM - 802.11a

Sampling System: ADC and DAC



ADC

$x(t)$

$t=nT$

$X(n)=x(nT)$

$t=nT$

Pulse Gen

LPF

$\hat{x}(t)$

Nyquist–Shannon Sampling Theory

- Study the condition where we can have

$$\hat{x}(t) = x(t)$$

- Theory:

  if a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $T=1/(2B)$ seconds apart.

  f_s=1/T is called sampling rate.

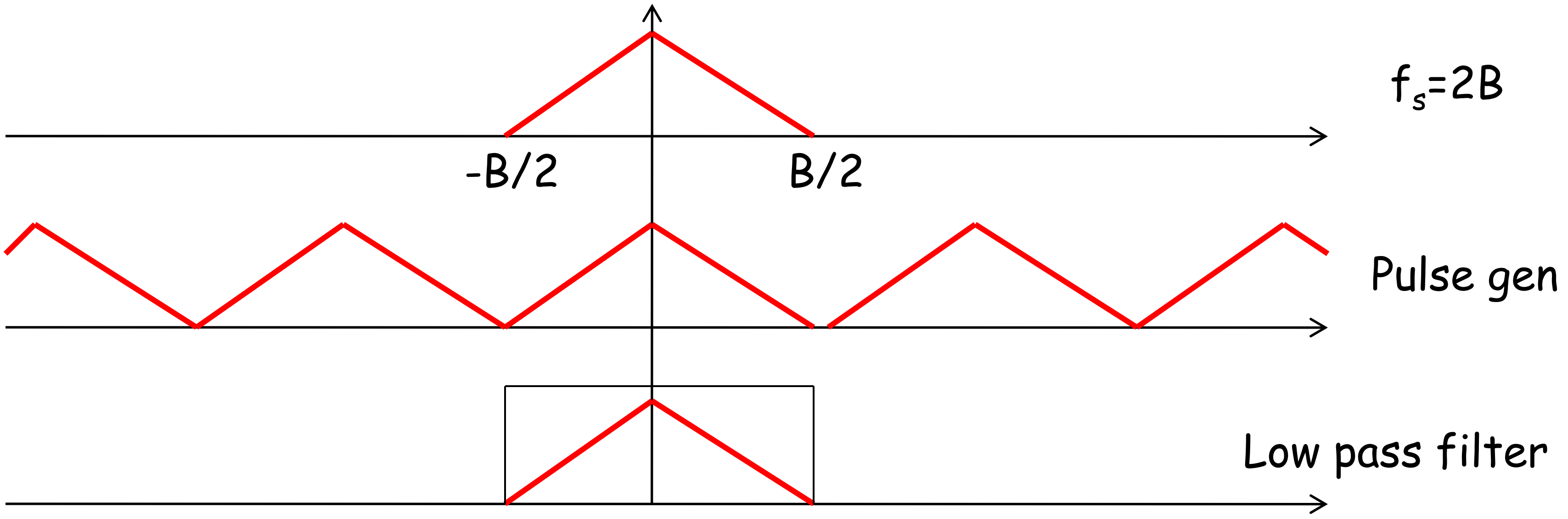
  Given sampling rate is f_s, the maximal frequency component can be reconstructed is limited by f_s/2.

  f_s= 1/(2B) is also called Nyquist sampling rate

Impact of The Sampling Theory

- With properly taken samples, operating on those discrete samples has exactly the same effects as operating on analog signal itself
- The fundamental of **Digital Signal Processing**

- Signal reconstruction from samples



Time Domain and Frequency Domain

- Fourier transform
  - Analytical tools for better present and study the properties of a function, say *f(t)*
  - Idea: represent an arbitrate function f(t) using a series of simple periodic functions

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j2\pi\omega t}dt$$

  - The frequency of $\mathrm{e}^{-2\pi j\omega t}$ is ω

  - F(ω) is called frequency domain presentation of f(t)

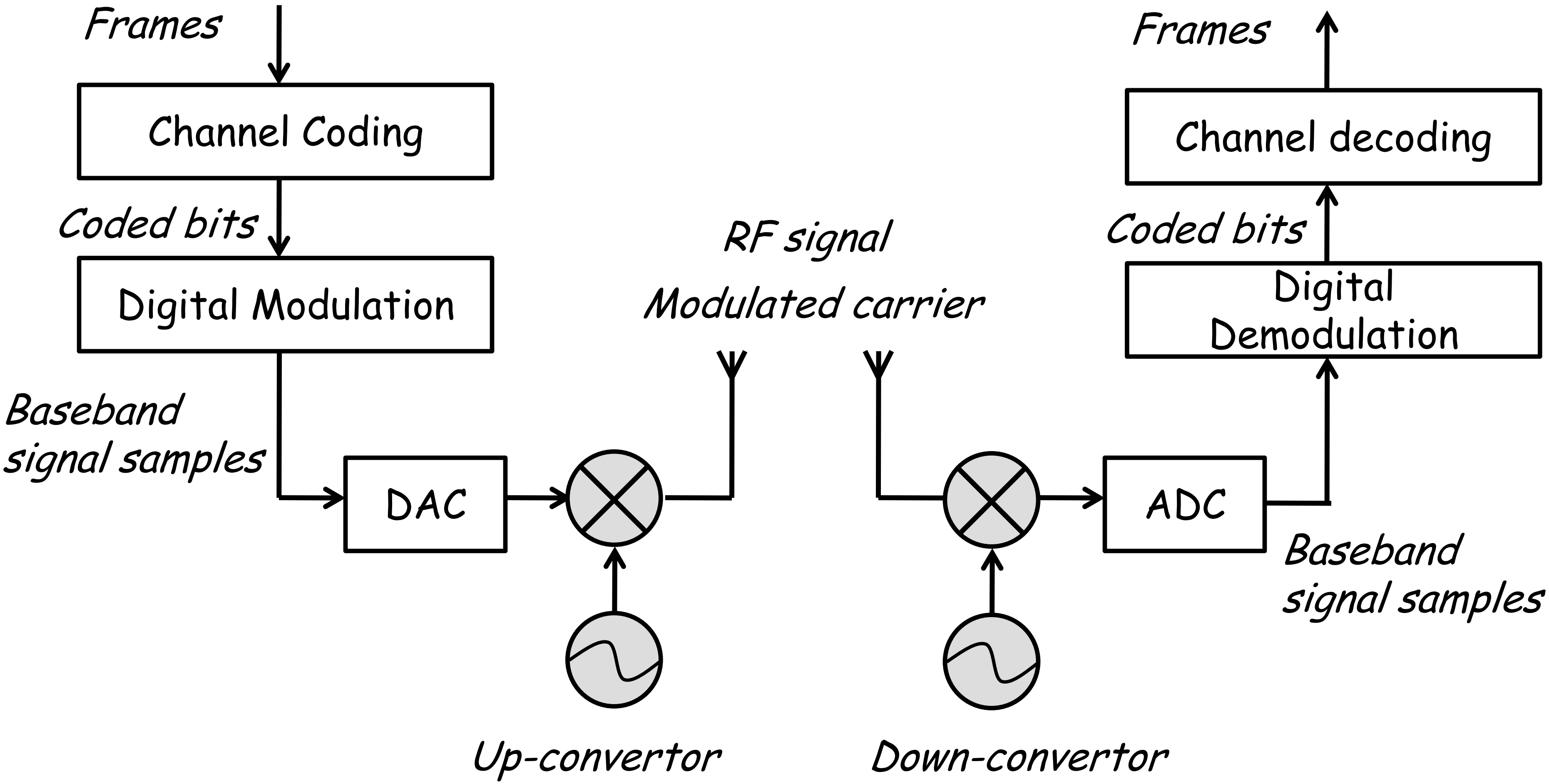
Discrete Fourier Transform (DFT)

- Operate on N discrete samples

$$X(k) = \sum_{n=0}^{N-1} x(n)\mathrm{e}^{-\frac{2\pi jkn}{N}}, k = 0..N$$

- Provide samples at discrete frequencies at $\frac{\mathrm{kn}}{\mathrm{N}}\mathrm{B}$
  where B=f_s/2
- When N increases, we have a finer samples at the frequency domain; but with a longer delay to gather enough samples

Overview of Communication System



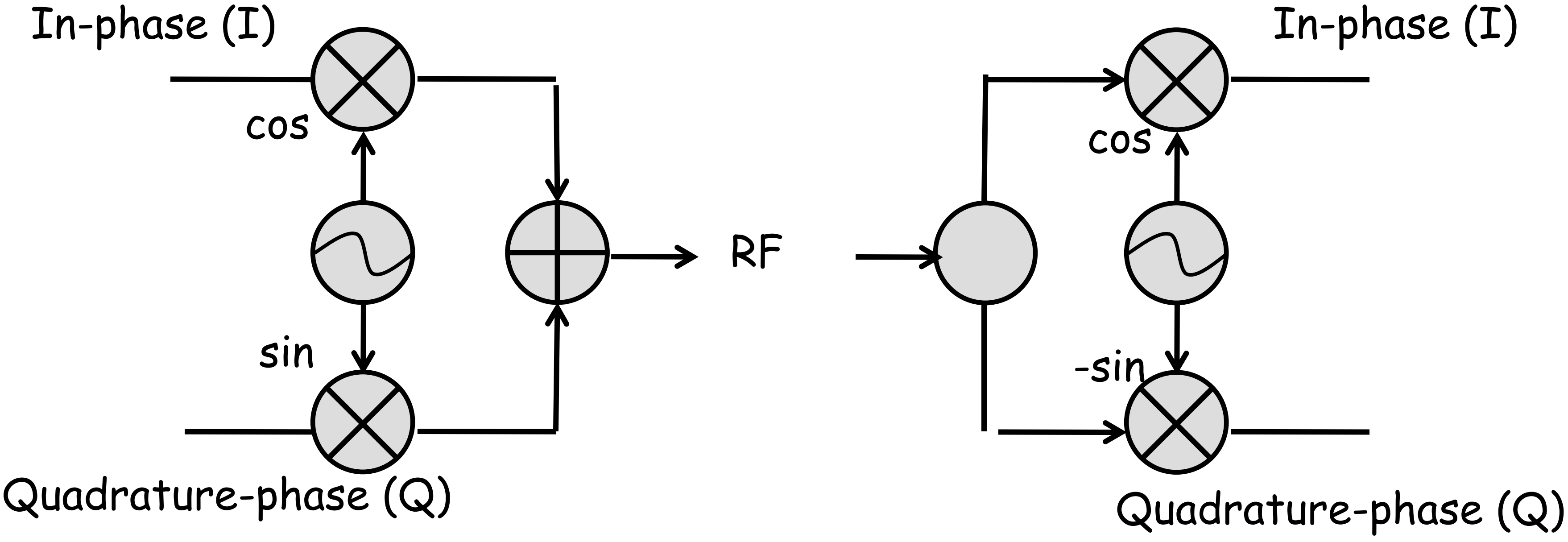
Digital Modulation and Demodulation

- Digital Modulation
  - Map a series of bits onto a finite number of M alternative symbols
  - Fundamental methods (a.k.a. Keying)
    - Phase-Shift Keying (PSK)
    - Amplitude-Shift Keying (ASK)
    - Quadrature Amplitude Modulation (QAM)

QAM is a more general form of digital modulation that combines both PSK and ASK

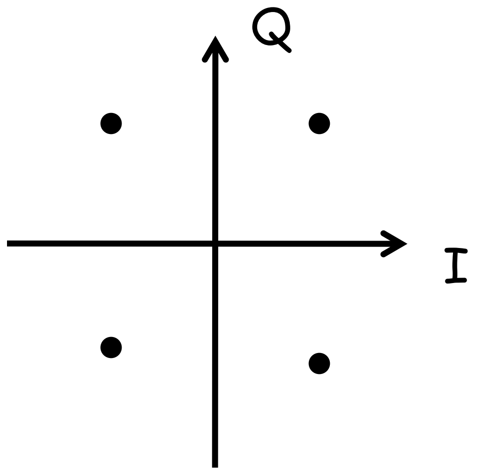
QAM and I/Q Signals

- Modulate symbols on two quadrature carriers
  - Quadrature: phase shift of the two carriers is 90 degree
    - $\sin(2\pi\omega t)$, $\cos(2\pi\omega t)$

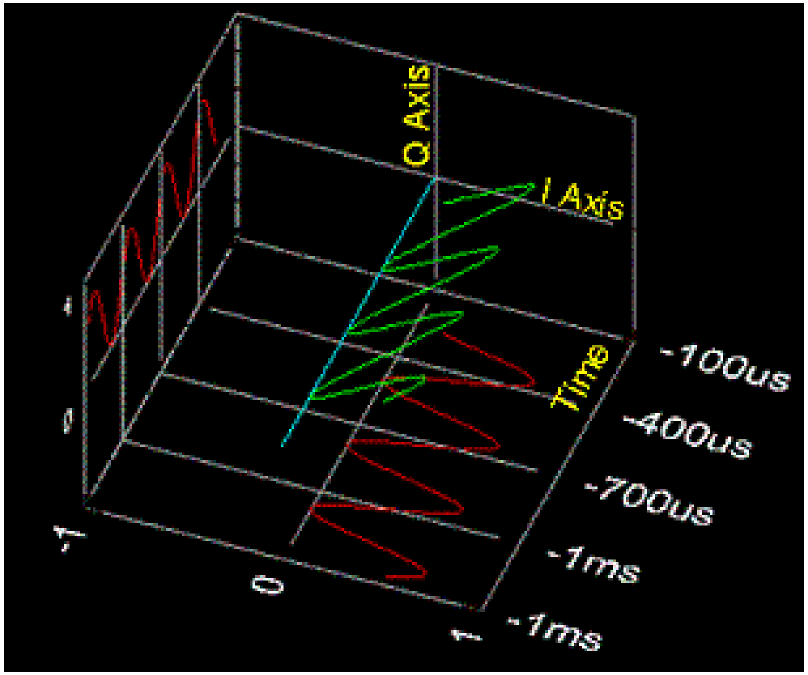


Constellation Graph

- Useful to describe QAM
- Present symbols as points on a complex plane
  - In-phase amplitude → Real part
  - Quadrature-phase amplitude → Image part



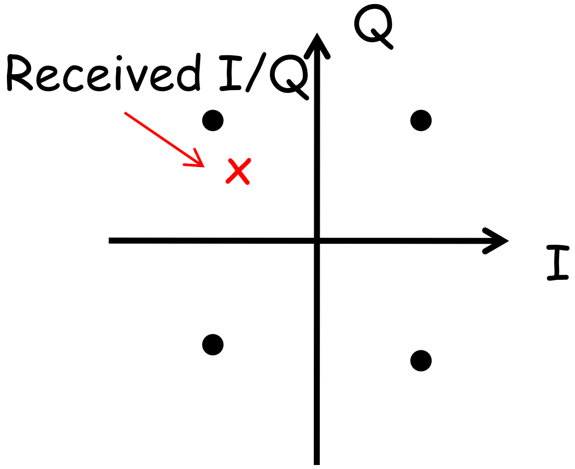
Constellation graph of QPSK



Digital Demodulation

- Determine the transmitted symbol based on received I/Q values
- Maximum a Posteriori (MAP) rule
  - Find the symbol that is mostly like the received I/Q

$$\hat{s} = \mathrm{argmax}_i\, p(s_i|y)$$



- Maximum like-hood

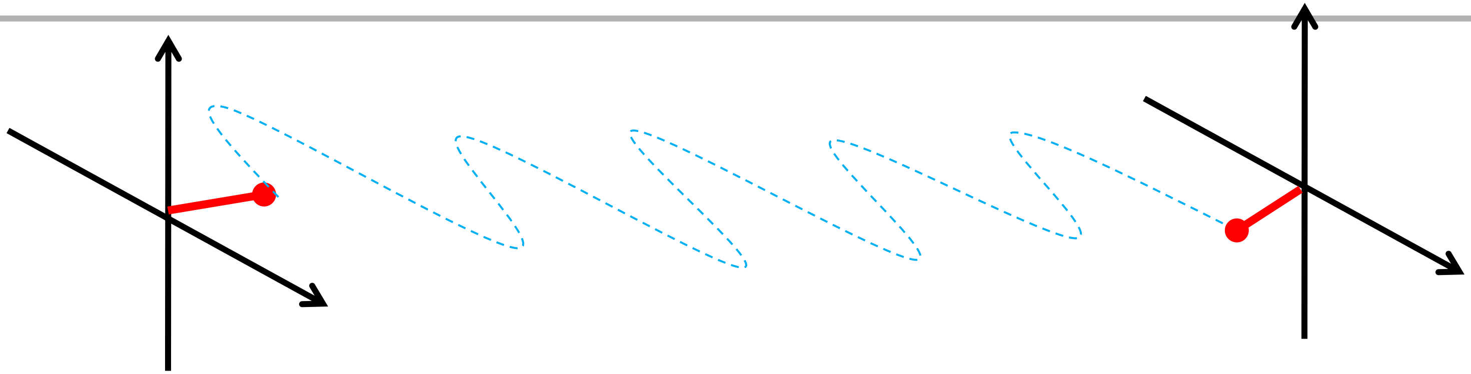
$$p(s_i|y) = \frac{p(y|s_i)\boxed{p(s_i)}}{\boxed{p(y)}}$$

  - MAP → ML (hard decision)
- Maximum like-hood ratio

$$\Lambda(y) = \frac{\sup\{p(y|b_i = 0)\}}{\sup\{p(y|b_i = 1)\}} \qquad \mathrm{LLR}(y) = \mathrm{Log}\,\Lambda(y)$$

  - LLR → Soft information

Wireless Channel



- A wireless channel is an abstract concept
- It represents a composite effects that changes properties of a received symbol compared to the transmitted symbol
  - Attenuation on the amplitude
  - Rotation in phase
  - Shift in frequency
  - Multi-path superposition

Wireless Channel Model

- Simple flat-fading model
  - No multipath
  - Model channel as a single complex value

    $h(i)=a(i)e^{-2\pi j\theta(i)}$, $y(i) = h(i)\, x(i)$
  - Only amplitude and phase of a symbol is distorted
- Multi-path fading model
  - Model channel as a series complex values

    $H(i) =[h_1(i), h_2(i), h_3(i),\dots,h_k(i)]$
  - Each component presents a path
  - $Y(i) = H(i) * X(i)$

    "*" is the convolution operation

$$z(i) = y(i) * x(i) = \sum_{m=-\infty}^{\infty} y(i-m)x(m)$$

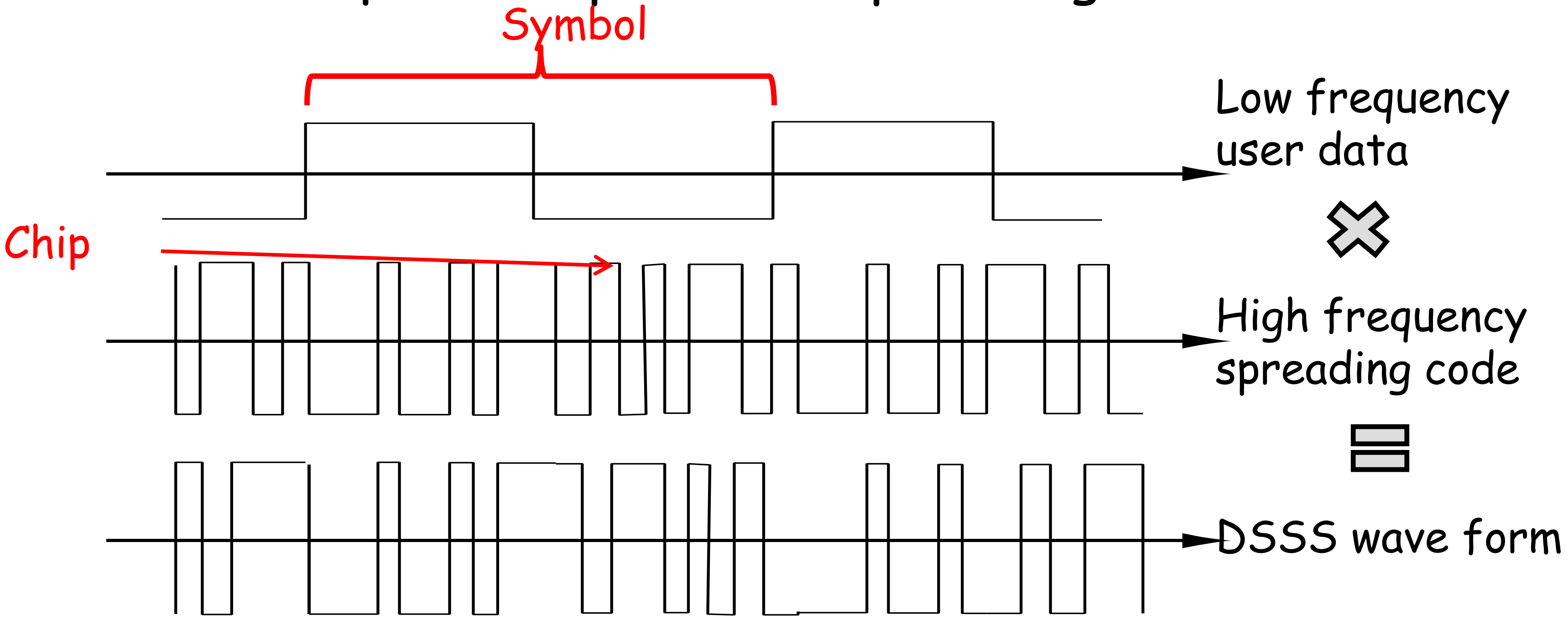
Receiver Synchronization and Equalization

- Timing synchronization
  - Find out the start point of a symbol
- Frequency synchronization
  - Compensate the frequency offset between the sender and the receiver
- Equalization
  - Compensate the distortion of wireless channel
  - Find $H^*(t)$, so that $H^*(t)\ H(t) = I$

- The synchronization and equalization method varies with different standards

Case Study I: Spectrum Spreading

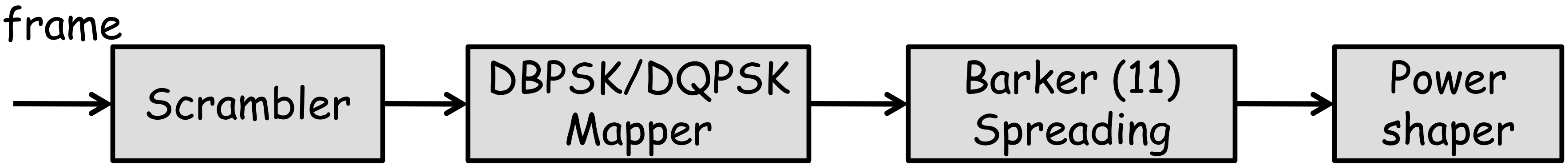
- Why spreading?
  - Gain more reliability at the cost of more spectrum use
  - Equivalent to channel coding
- Direct-sequence Spectrum Spreading



IEEE 802.11b

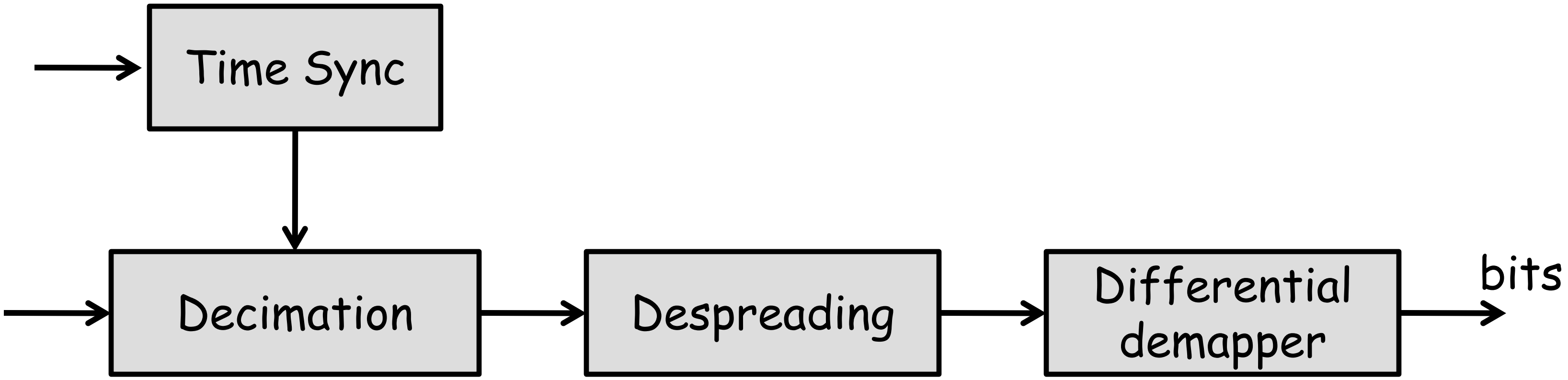
- IEEE Standard for WLAN
- DSSS PHY on 22MHz channel 2.4GHz
  - Symbol rate: 1Mbps
  - Spreading sequence: Barker sequence; CCK
- Data rate up to 11Mbps
- Modulation: DBPSK, DQPSK, CCK
- Channel coding: no (why?)

802.11b Transmission Structure



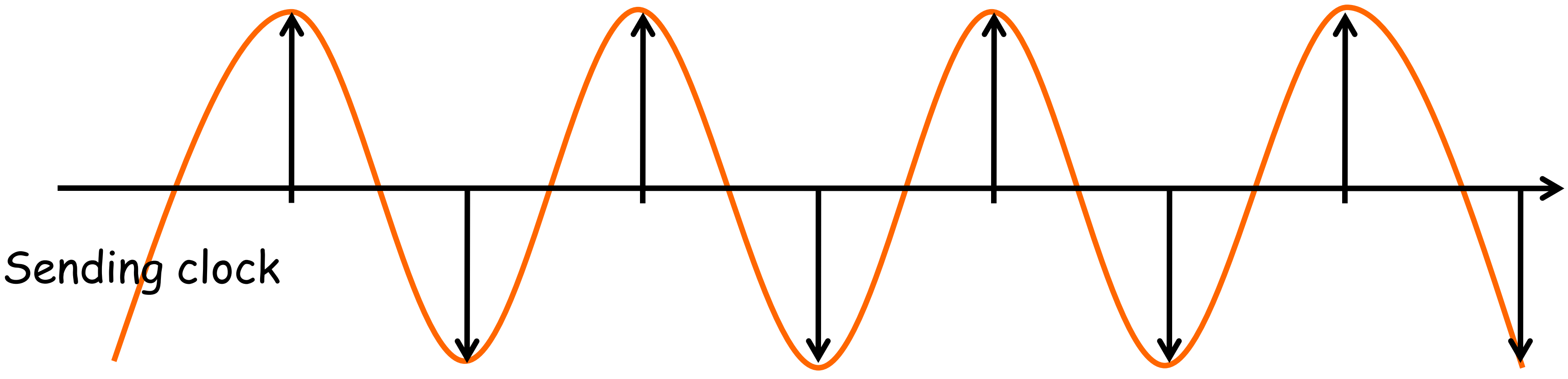
- Scrambler
  - Randomize the input bit stream to favorite RF circuits
- Mapper
  - Map bits to I/Q symbols: DBPSK/DQPSK
  - x[k] = s[k] x[k-1]
- Spreading
  - Multiple symbols by the spreading sequence
- Power shaper
  - Oversample
  - Raised root cosine filter

802.11b Receiver Structure



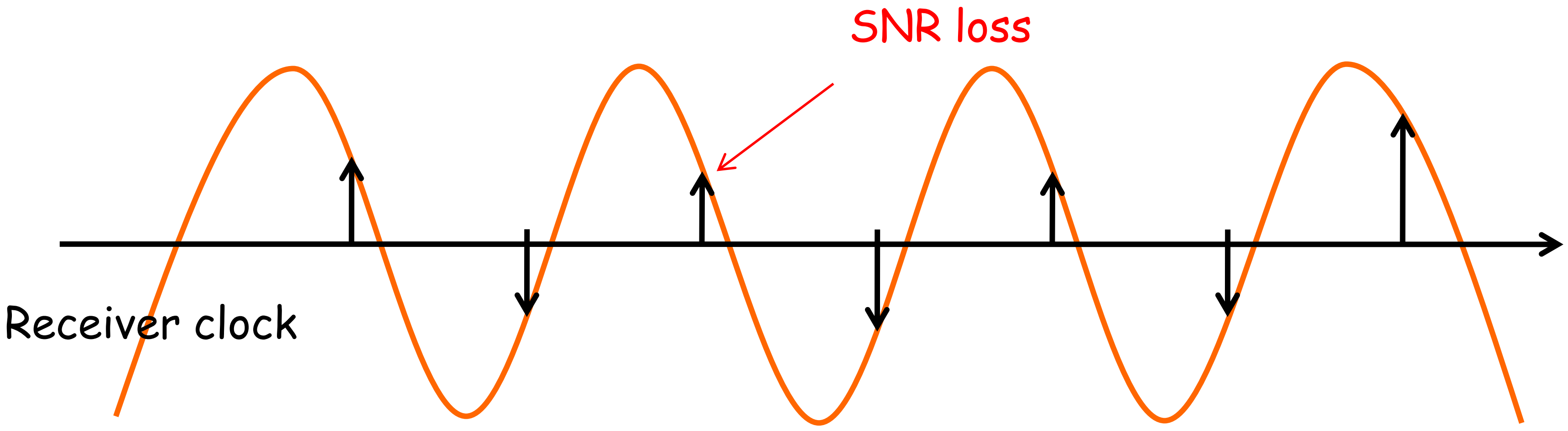
Time Synchronization

- Sample time recovery



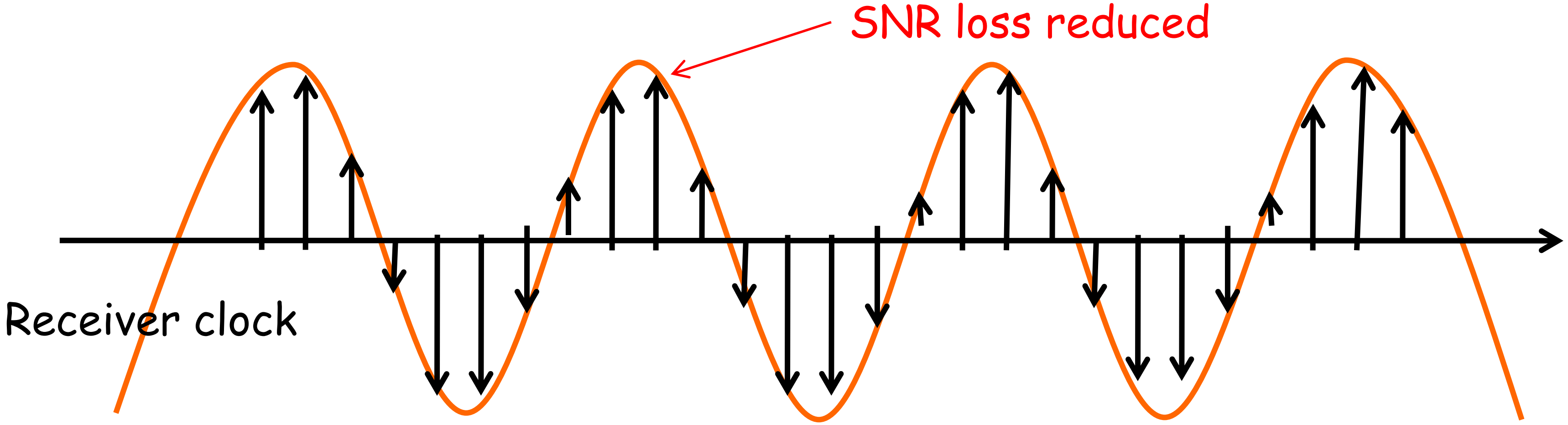
Time Synchronization

- Sample time recovery
  - How to recovery to the ideal sampling time?



Time Synchronization

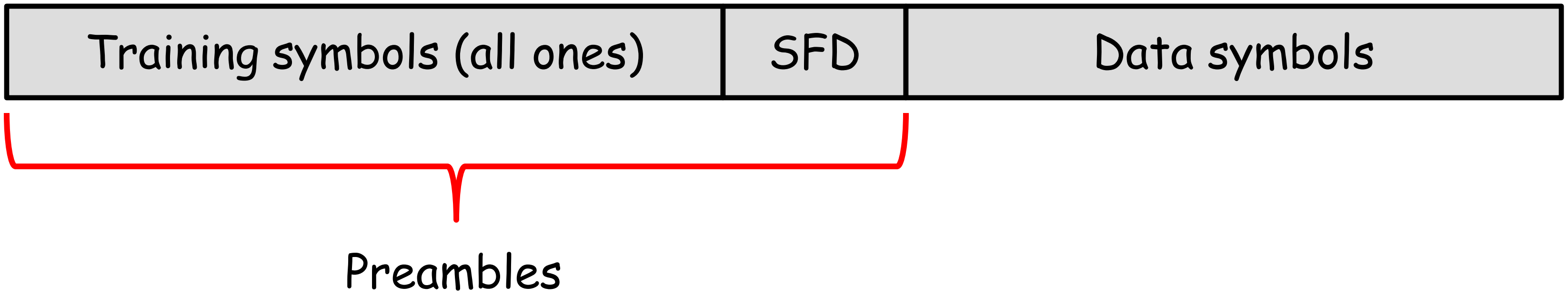
- Sample time recovery
  - How to recovery to the ideal sampling time?
  - Simple solution: over-sampling!
  - Search for the sample index with the maximal energy level
  - What if the sampling frequency of sender and receiver offsets slight?



Time Synchronization (cont.)

- Symbol time recovery
  - Correlate with Barker
  - Looking for a peak

- Locate the start of data symbols
  - Decode to find all ones (training symbols)
  - Search SFD after lock onto training symbols



Training symbols (all ones)

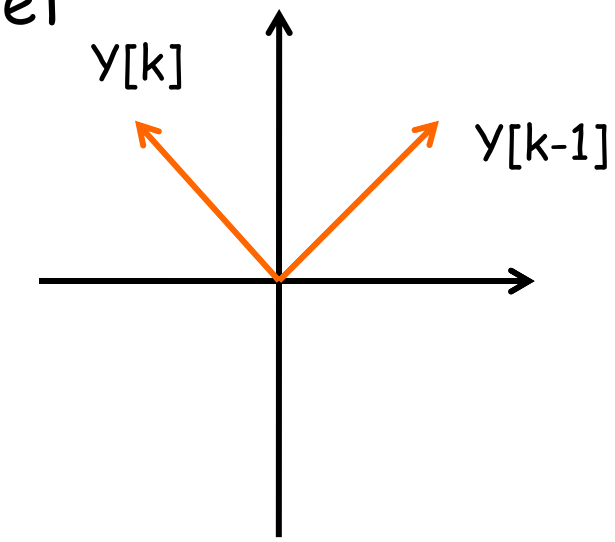
SFD

Data symbols

Preambles

Differential Demapper

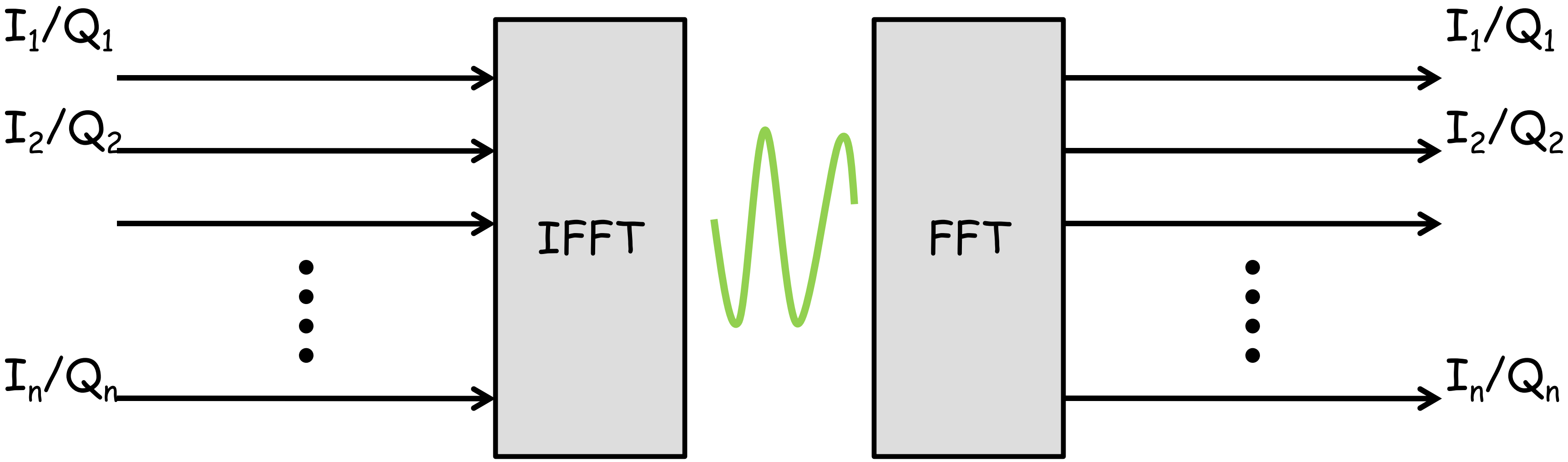
- Calculate the angle rotated and demap it to bits
- Why using differential modulation?
  - Why the carrier frequency offset is not considered?
  - Why initial symbol phase is not considered?
  - What is cost?



Case study II : OFDM

- Orthogonal Frequency Division Multiplexing
- A multi-carrier modulation scheme for high-speed wireless systems
- Sub-carrier symbols are modulated on a series of orthogonal sub-carriers
- Pros:
  - Simple equalization
  - Robust against inter-symbol interference (ISI) and multi-path fading
  - High spectral efficiency
  - Low sensitivity to time synchronization errors
- Cons:
  - Sensitive to Doppler shift
  - Sensitive to frequency synchronization errors
  - High peak-to-average-power ratio (PAPR), requiring high dynamic range ADC/DAC

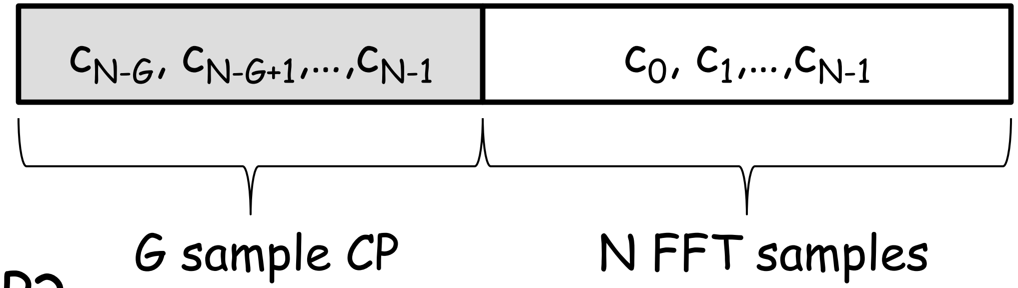
OFDM Example



- OFDM is easily implemented using FFT
- Sub-carrier locates at Bk/N, k=1,2,…,N-1.

Cyclic Prefix

- N-point IFFT translates N frequency-domain sub-carrier symbols to N time-domain OFDM samples $c_0, c_1, c_2,\dots,c_{N-1}$
- Cyclic prefix – copy G tailing samples to the front and form a (G+N) sample OFDM symbol



$c_{N-G}, c_{N-G+1},\dots,c_{N-1}$ | $c_0, c_1,\dots,c_{N-1}$

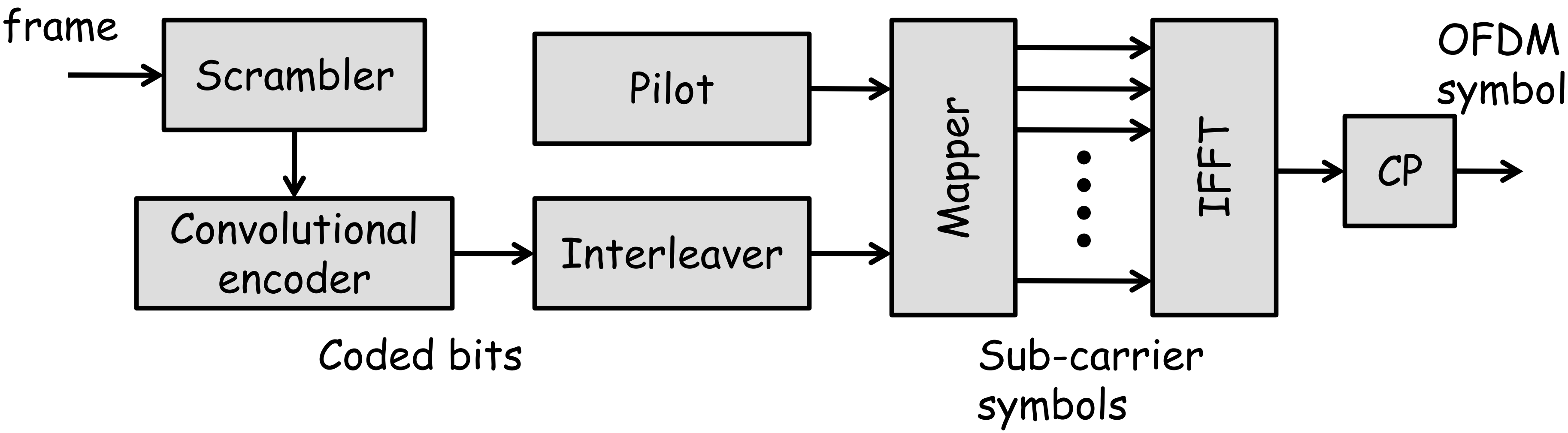
G sample CP | N FFT samples

- Why CP?
  - Handling ISI – CP works as a guide time
  - Handling time synchronization error
    - Rotation of time-domain samples results only phase-shift in frequency-domain symbols
    - Latter can be corrected by equalization

Case Study: IEEE 802.11a

- IEEE Standard for WLAN
- OFDM PHY on 20MHz channel 5GHz
  - Same PHY is adopted in 2.4GHz spectrum as 802.11g
  - 64 sub-carriers (FFT size is 64)
- Data rate up to 54Mbps
- Modulation: BPSK, QPSK, 16QAM, 64QAM
- Channel coding: convolutional code with 1/2, 2/3 and 3/4 coding rate

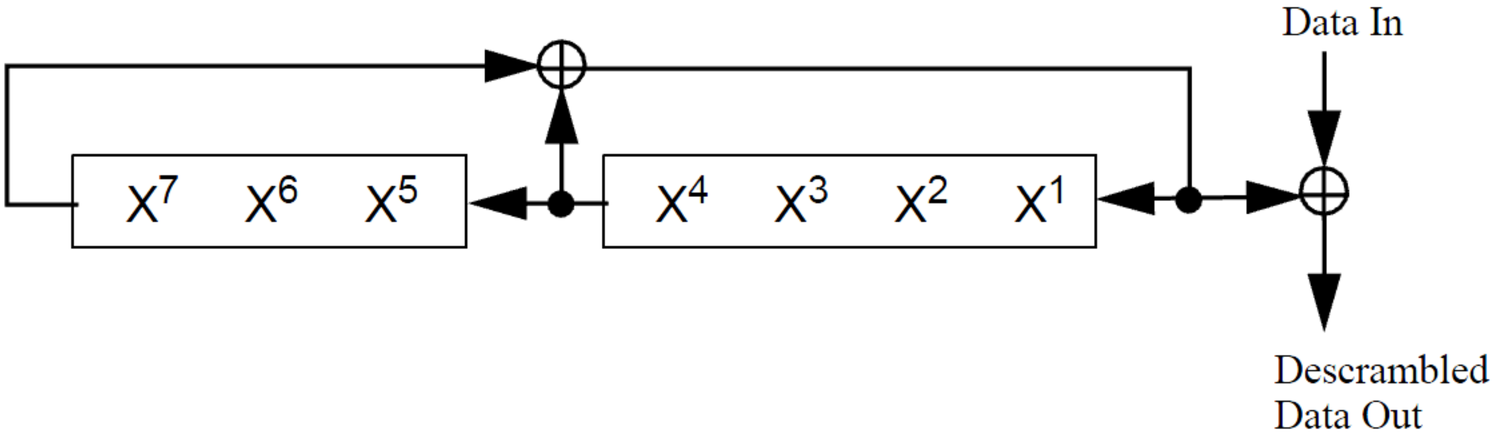
Transmitter structure of 802.11a



- Scrambler
  - Randomize the input bit stream to favorite RF circuits
- Interleaver
  - Spread coded bits among different subcarriers and bit positions in subcarrier symbol
- Pilot
  - Known symbol sequence on known subcarriers; providing training sequence to track the channel changes

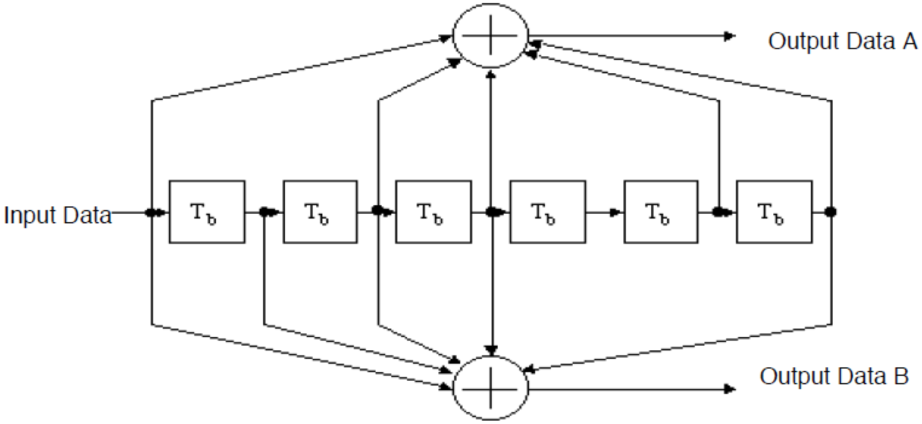
Scrambler

- Simple XOR between the bit stream and a 127-bit template
- The initial states of the scrambler are pseudo random non-zero state from a known pattern
  - Why ?
- This initial states are encoded in a field in PHY header and transmitted to the receiver

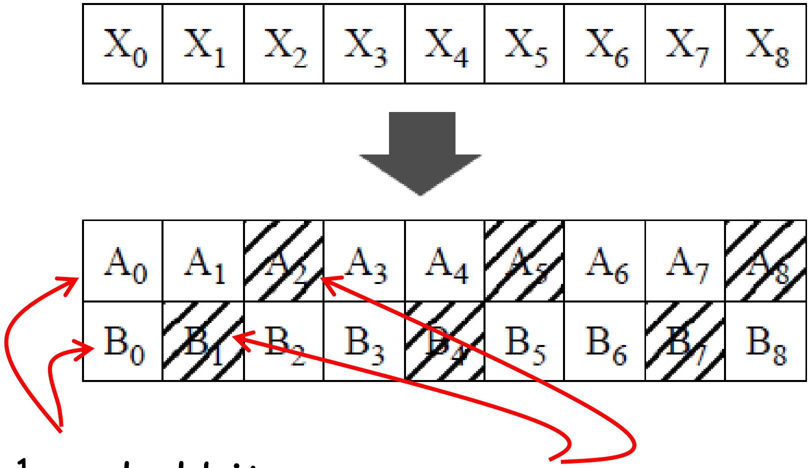


Convolutional Encoder

- A sort of Forward Error Correction code
  - Encode m bits into n bits (n>=m); m/n code
- Encoder used in 802.11a
  - Basic rate ½
  - Achieving higher rate by punching



Original bits



X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8

½ coded bits

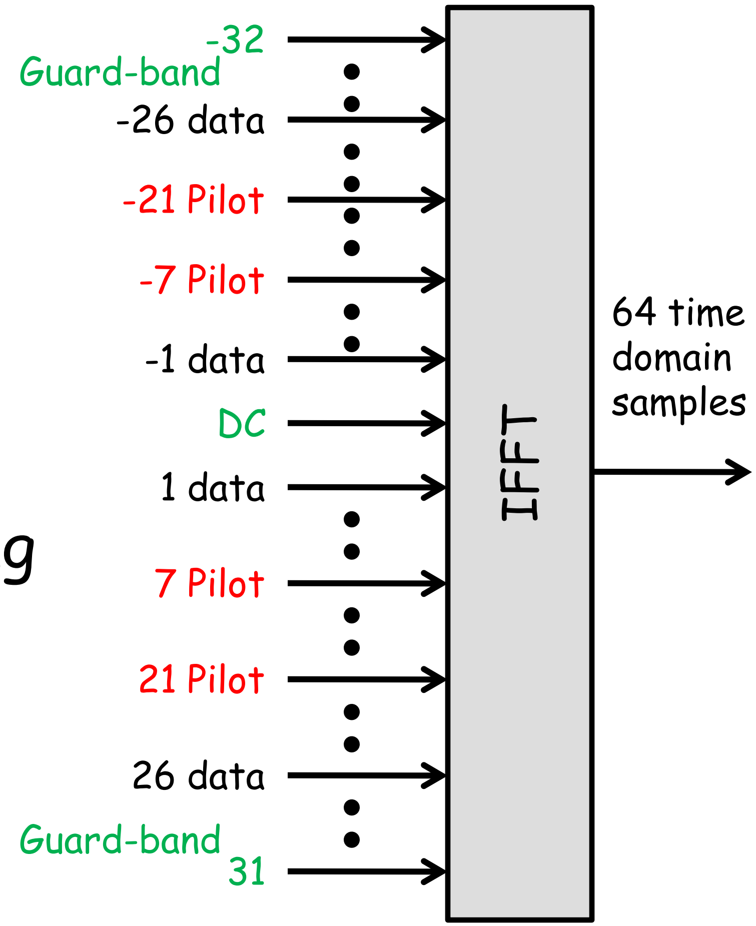
Punched bits

Interleaver

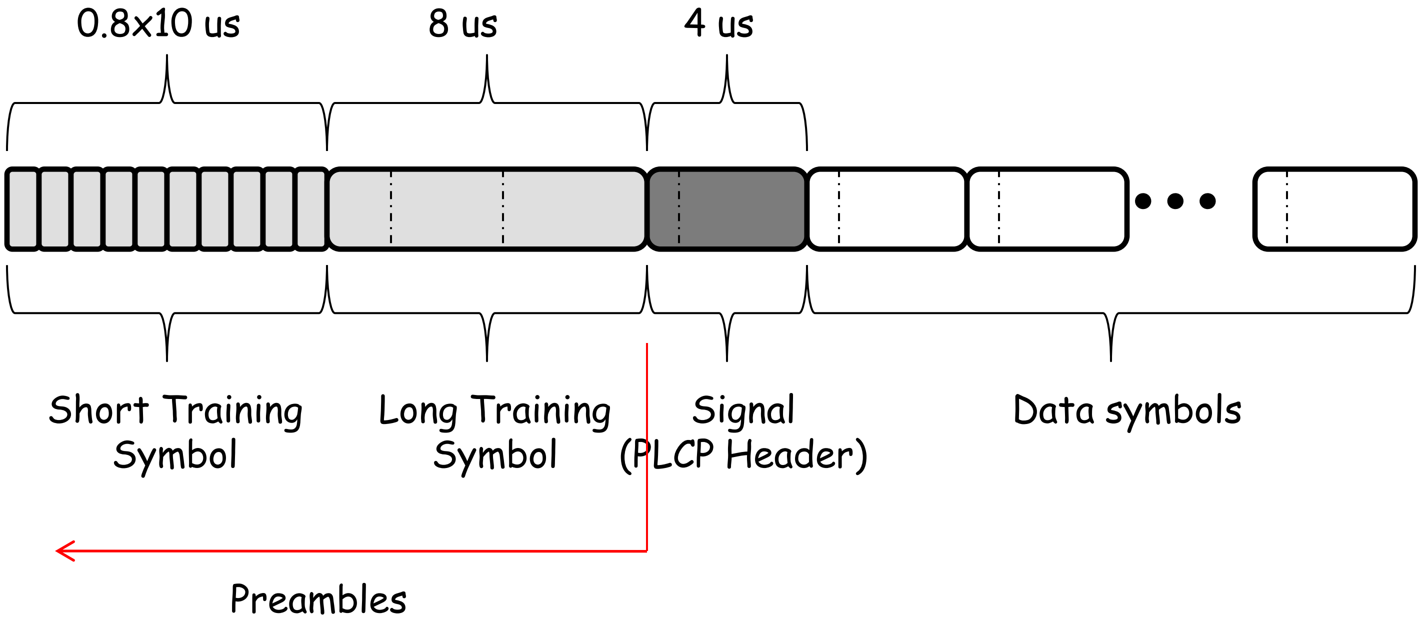
- Why interleaving?
  - Channel code works best with random independent errors
  - Wireless errors are in practical correlated
- Error correlation in subcarriers
  - If one subcarrier faces deep fading, it is very likely that nearby subcarriers suffer the similar fading – coherent bandwidth
- Spread adjacent coded bits to
  - Non-adjacent subcarriers
  - Alternate on less and or significant bits on constellation

Pilot and Mapping

- Subcarrier usage
  - 48 data subcarriers
  - 4 pilot subcarriers
  - 12 null subcarriers
- Pilot subcarriers
  - Initial {1,1,1,-1}
  - Polarity changes according a pre-defined sequence



PHY Frame Structure



- Preamble
  - 10 short training symbols (STS)
  - Long training symbol (LTS)
    - Two repeated 64 FFT samples (3.2 us * 2) + CP (1.6 us)
- Signal and data symbols
  - 64 FFT samples (3.2 us) + CP (0.8 us)

Receiver Structure of 802.11a

- Two states in receiver
  - Synchronization state
    - Tries to find a preamble (frame detection)
    - Synchronizes to the preamble (time sync, freq. sync, equalization, etc.)
    - Operates on preamble
  - Decoding state
    - Starts to demodulate and decode the transmitted bits
    - Tracks the channel changes via pilots
    - Operate on data symbols

Receiver Synchronization (I)

- Frame detection
  - Idea: utilize the repeat pattern of STS
  - Algorithm: auto-correlation

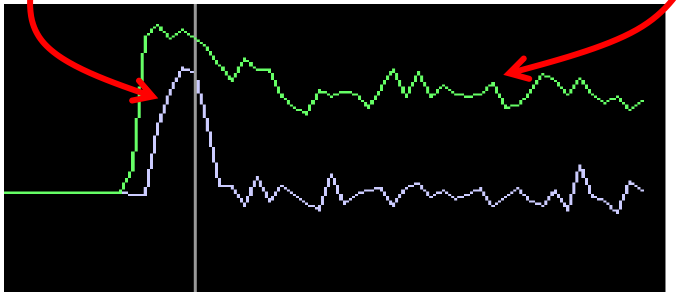
    Detect if a patter has repeated periodically
- Time synchronization
  - Idea: utilize the known waveform of STS
  - Algorithm: cross-correlation

    Detect if a patter has appeared

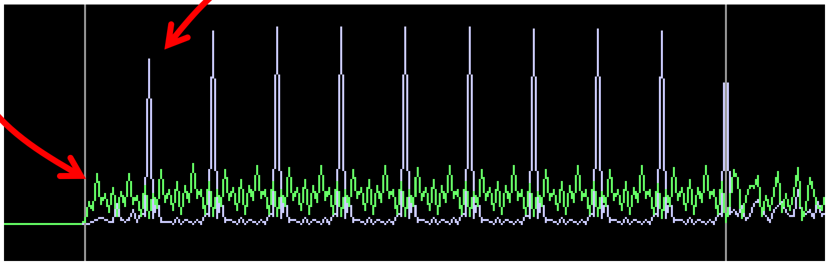
A periodical patter is determined

Auto-correlation

Average energy

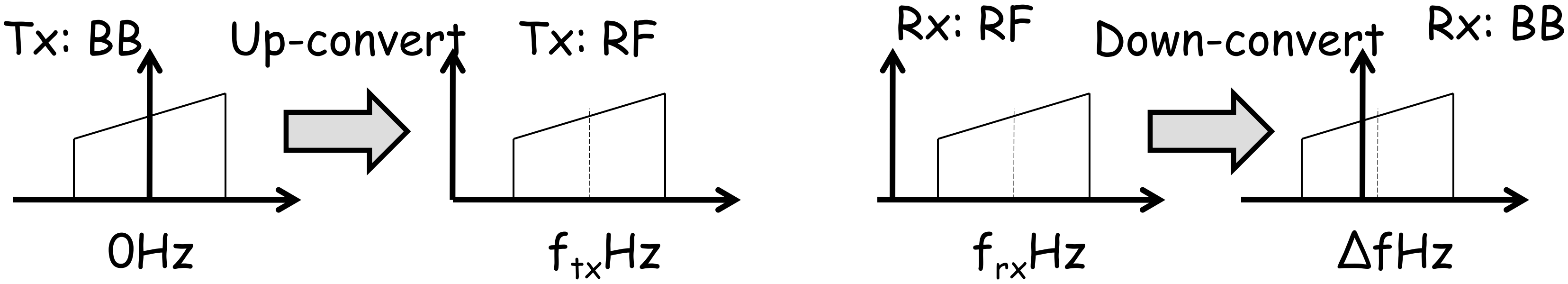


Cross-correlation peaks



Receiver Synchronization (II)

- Frequency offset
  - Transmitter's frequency is not exactly equal to the receiver's frequency
  - Range from 500Hz~100KHz without calibration
  - Destroy the orthonality and cause inter-subcarrier interference



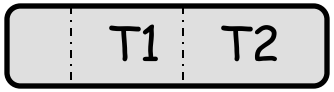
$$y_i = x_i e^{j2\pi i \Delta f}$$

inter-subcarrier interference

$$Y_n = \sum_{i=0}^{N-1} y_i e^{-j2\pi\frac{in}{N}} = \sum_{i=0}^{N-1} x_i e^{-j2\pi\frac{in}{N}} \boxed{e^{j2\pi i \Delta f}}$$

Frequency Offset Estimation and Compensation

- Utilize the LTS for FOE
  - LTS has two repeating blocks



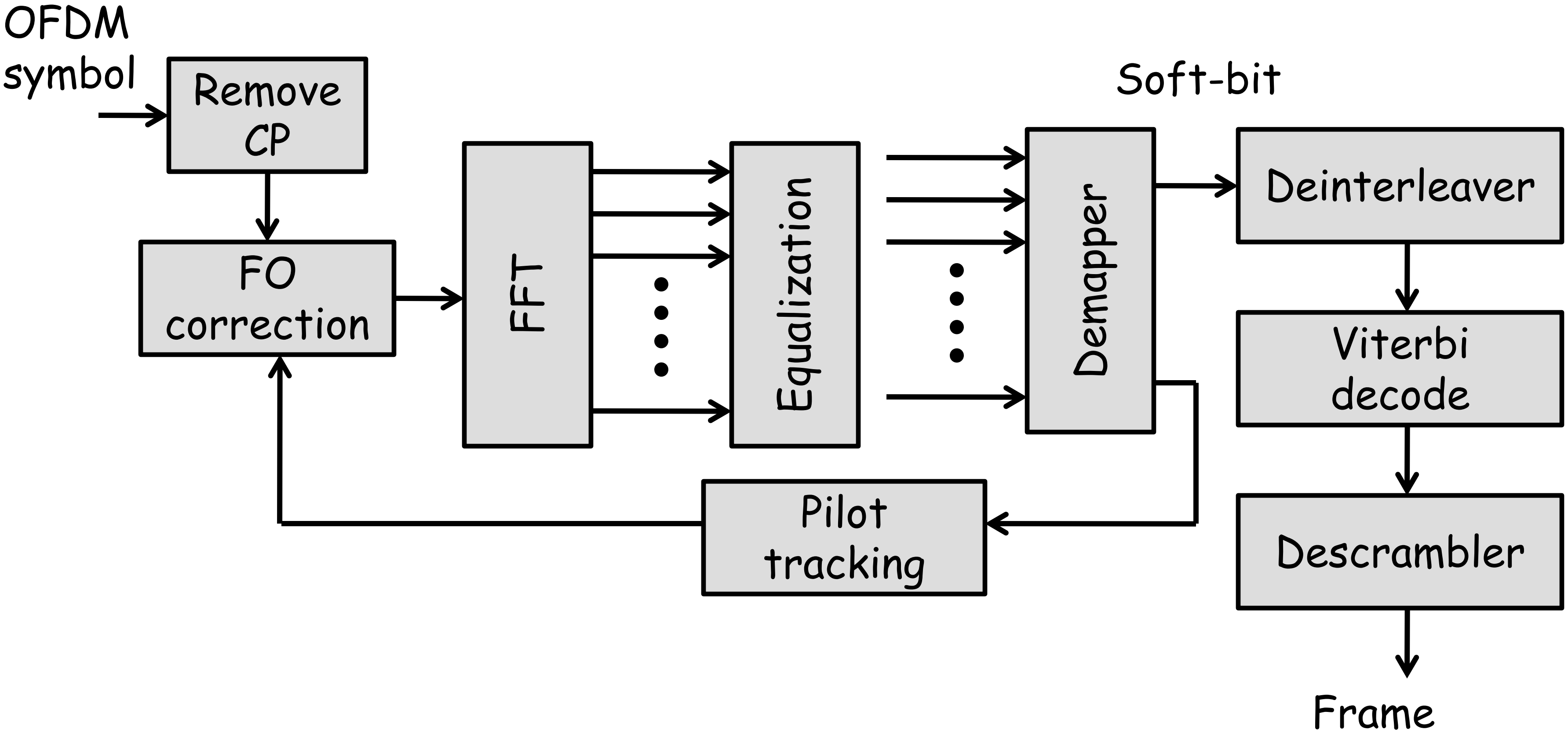
$$y_{i+N} = y_i e^{j2\pi N \Delta f}$$

  - For each sample, we can compute a Δf; the estimation take the average
  - A coarse estimation based on STS is needed if the frequency offset is very large
    - i.e. $N\,\Delta f > f_s$
- Freq. offset compensation
  - The ith sample is rotated by 2πiΔf before feeding to FFT

Equalization

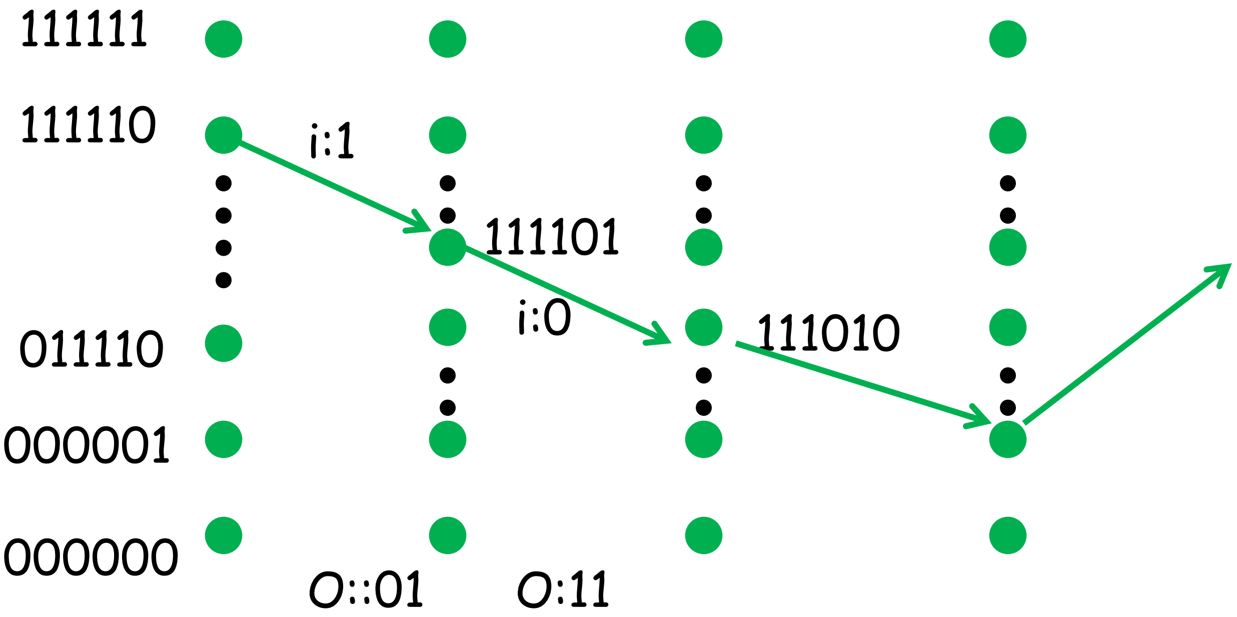
- Separate equalizer for each subcarrier
- Simple flat fading channel for each subcarrier
  - Narrow band assumption
- Utilize LTS
  - Compute h_i for each subcarrier
    $h_i=s_i/y_i$,
    where s_i is the known transmitted symbol on subcarrier i, and y_i is the received symbol
- Equivalent to FIR filter at time domain
  - But the filter parameters are easier to determine

Demodulation and Decoding Data Symbols



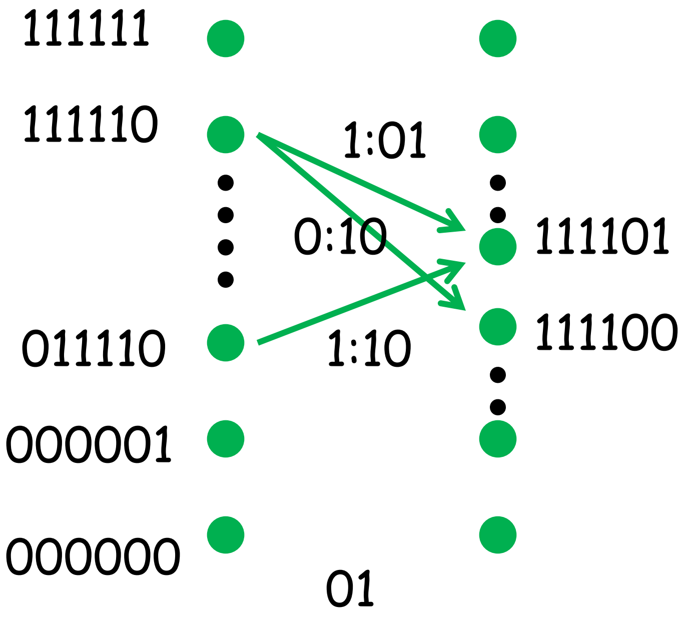
Viterbi Decode (I)

- Dynamic programming to compute the most likely sequence of hidden states
- Trellis presentation of convolutional codes
  - K-bit convolutional code needs 2^{K-1} states
  - 64 states in 802.11a
  - A path represents an encoded bit sequence



Viterbi Decode (II)

- Decoding – given the output sequence, find the most likely encoding sequence
  - Find the path whose output is more like the given sequence
- Path metric
  - Hamming distance
  - Soft-info
- Algorithm
  - Path metric compute and select
  - Trace-back along the minimal path


111111
111110
011110
000001
000000
1:01
0:10
1:10
111101
111100
01