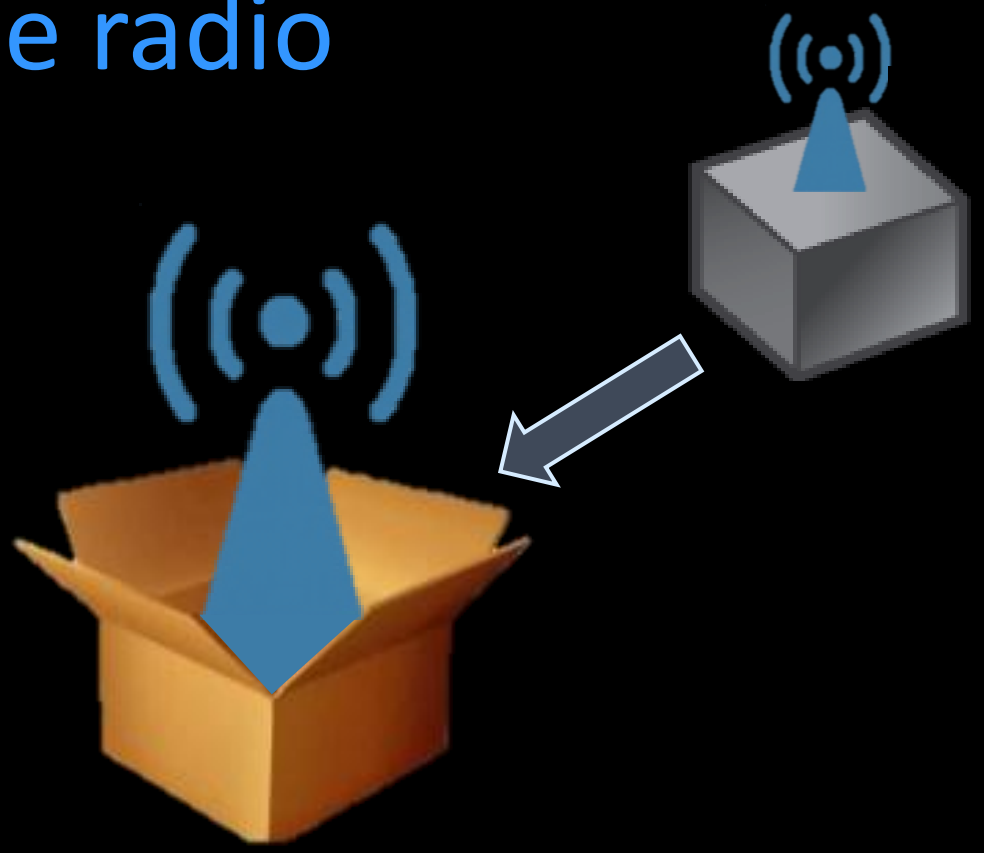
OpenRadio
A programmable wireless dataplane

Manu Bansal
Stanford University

Joint work with Jeff Mehlman, Sachin Katti, Phil Levis

HotSDN ‘12, August 13, 2012, Helsinki, Finland

Opening up the radio



Why?

- Evolving protocols
- Diverse applications
- Network growth and Diverse scenarios

What?

- Flexible radio stack
- Deployable performance
- Convenient programming

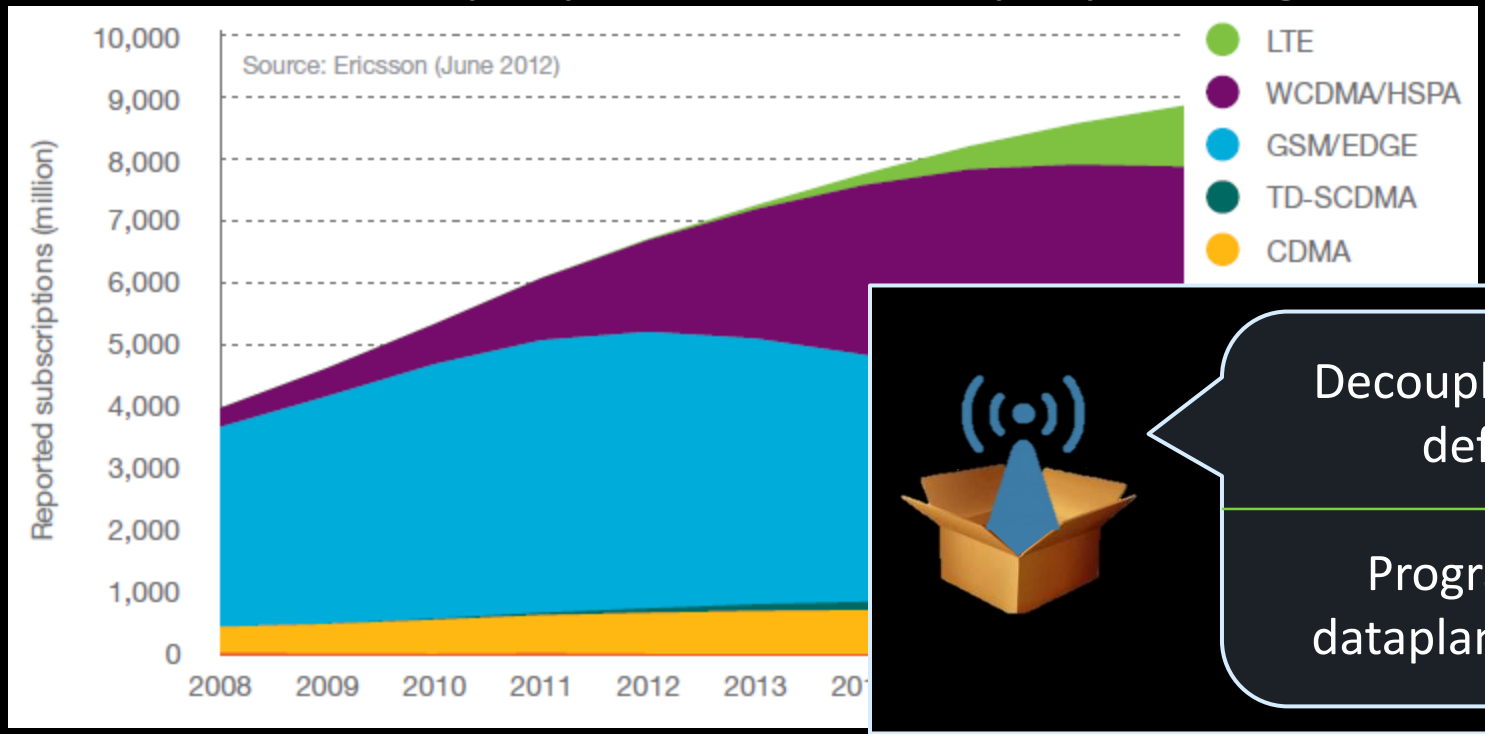
How?

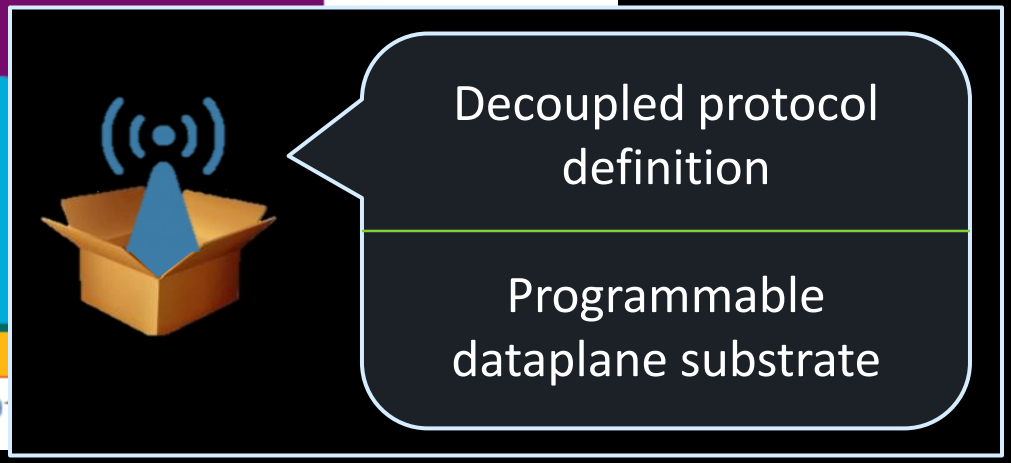
- Decouple functionality and HW
- Judicious split of protocols
- High-level abstractions

MOTIVATION

Evolving standards

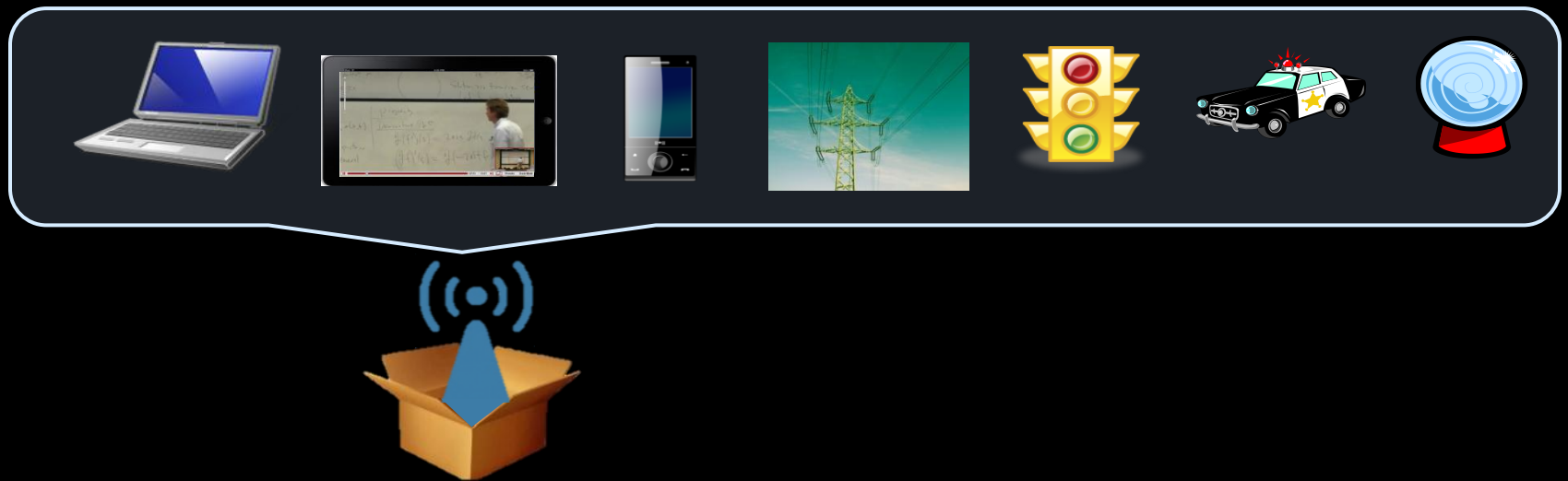
- Major 3GPP LTE releases every 18 months
- Continuous minor updates
- Old standards don't die
  - Multi-mode basestation radios
- Can we deploy once and keep updating?





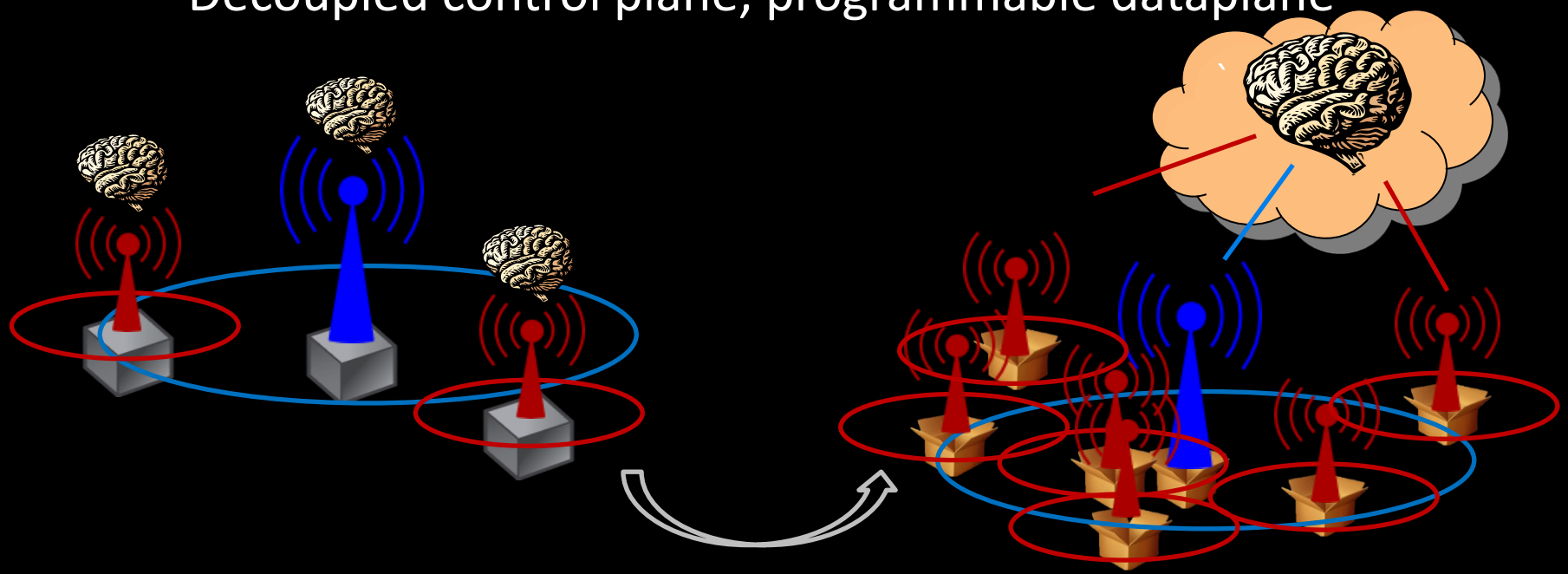
Application diversity

- Can do better than one-size-fits-all radio stack
  - Eg. Unequal error protection (UEP) for video
- LTE specifies several traffic classes
  - How do I implement them?
  - Future traffic classes?
- How about a programmable infrastructure?



Network growth & scenario diversity

- Reducing cell-sizes to meet capacity demands
  - Smaller macro-cells → less users per cell
  - Picocells (open), femtocells (closed) just thrown in
  - Interference dominates, mobility is harder
- How can we make basestations coexist?
  - Dynamic scenario-specific adaptation
  - Decoupled control plane, programmable dataplane

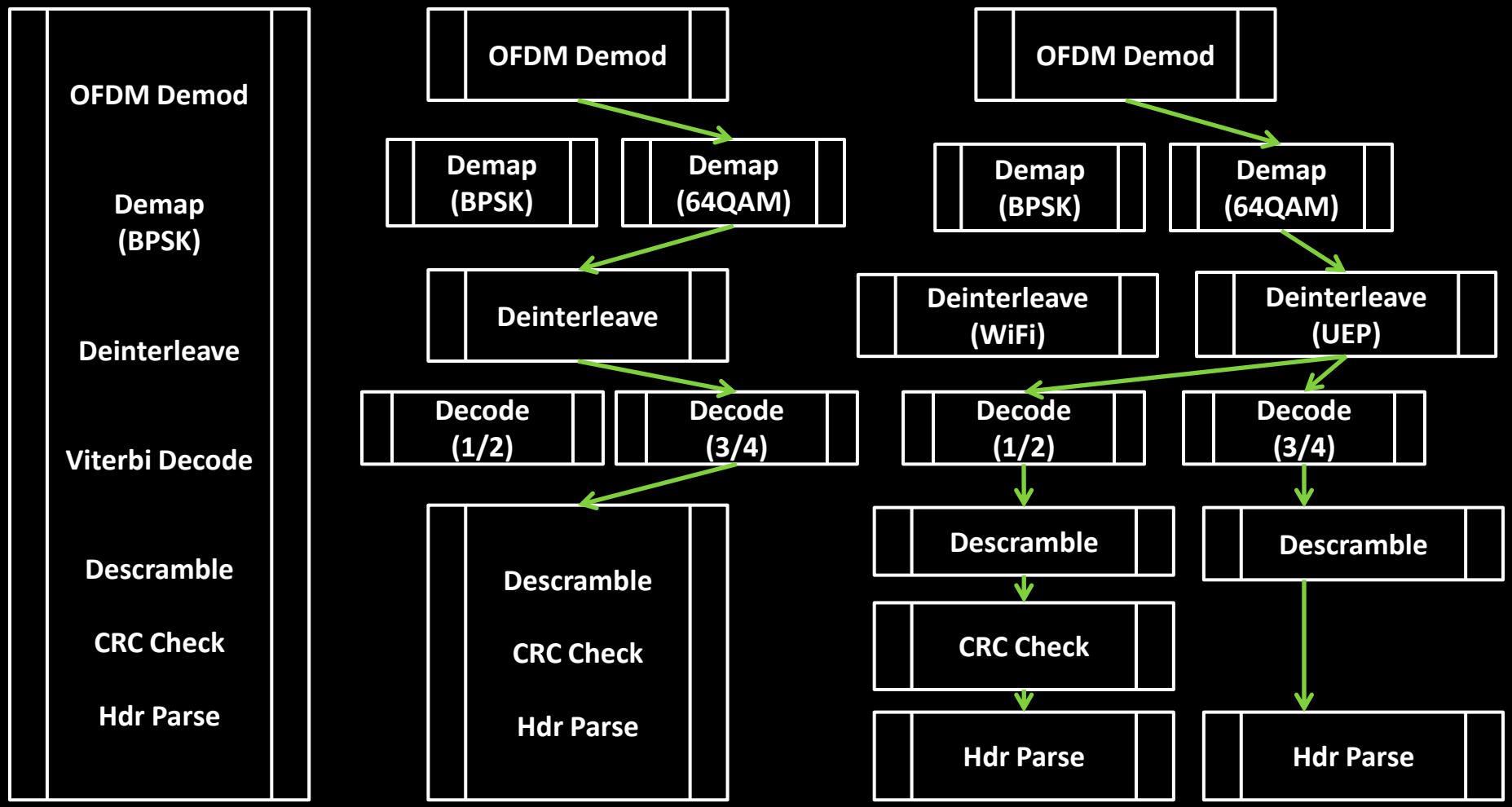


Design goals and challenges

- Programmable wireless dataplane
  - Customize remotely after deployment
  - At least 20MHz OFDM-complexity performance
    - More than 100 GLOPS computation
    - Strict processing deadlines, eg. 25us ACK in WiFi
  - Modularity to provide ease of programmability
    - Only modify affected components, reuse the rest
    - Hide hardware details and stitching of modules
  - Built using off-the-shelf components

PROGRAMMING ABSTRACTIONS

Wireless programming

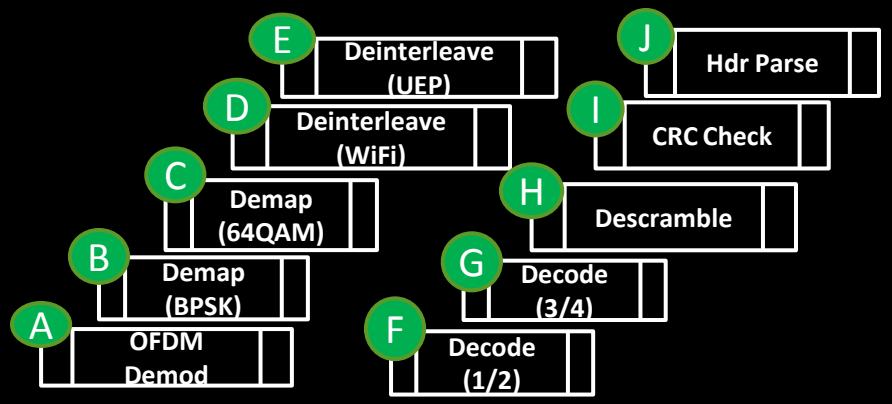


WiFi 6mbps

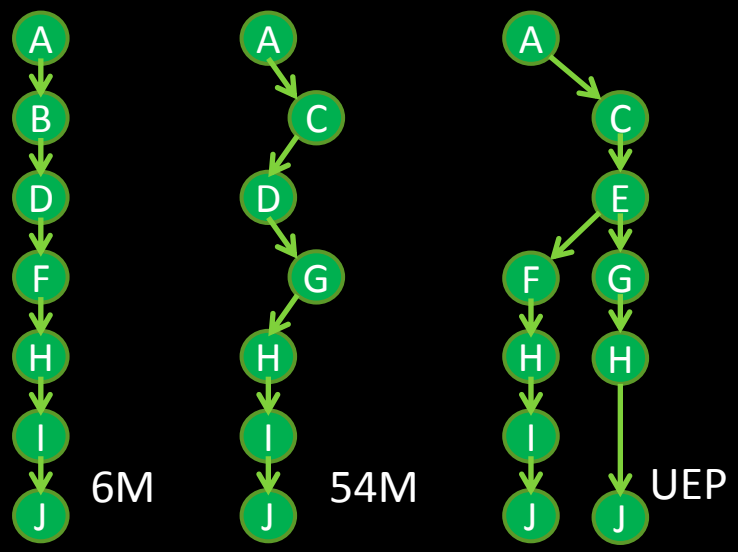
WiFi 6, 54mbps

WiFi 6, 54mbps and UEP

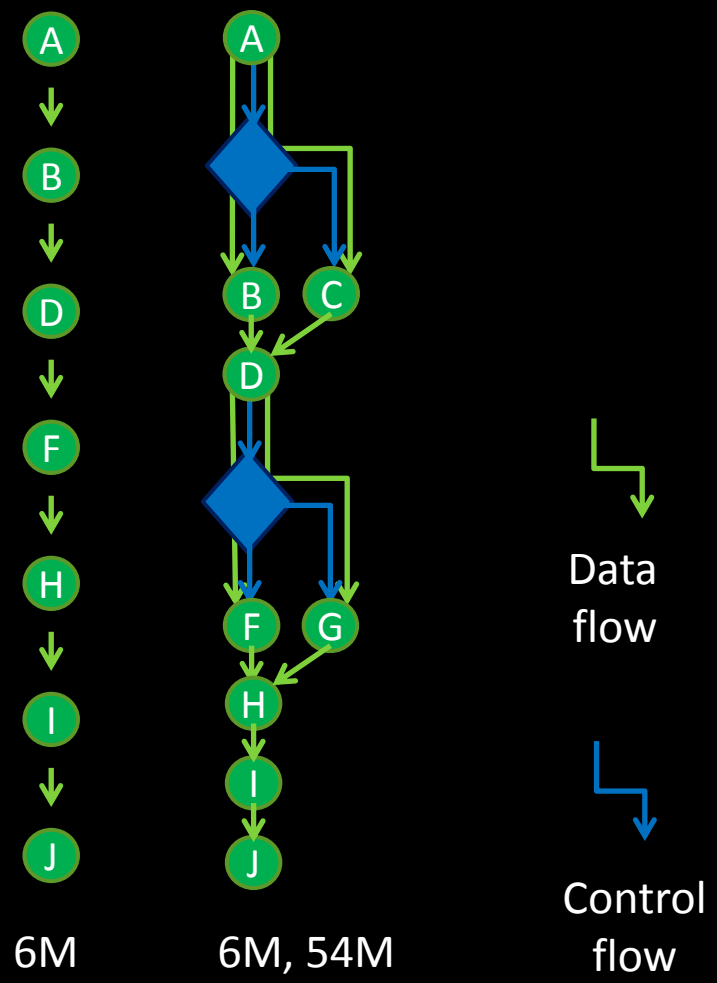
Modular declarative interface



Modular library of blocks



Composing Actions: DAGs of blocks



Declaring Rules: Branching logic

DESIGN PRINCIPLES

Design principle I
Judicious scoping of flexibility

- Provide coarse-grained blocks
  - FFT block, Viterbi decoder block
- Configurable parameters
  - FFT length, Trellis structure
- Just enough flexibility

- Higher level of abstraction
- High performance through hardware acceleration
  - Viterbi co-processor
  - FFT co-processor

- Off-the-shelf hardware
  - Heterogeneous multicore DSPs
  - TI, CEVA, Freescale etc.

Algorithm	WiFi	LTE	3G	DVB-T
FIR / IIR	√	√	√	√
Correlation	√	√	√	√
Spreading			√	
FFT	√	√		√
Channel Estimation	√	√	√	√
QAM Mapping	√	√	√	√
Interleaving	√	√	√	√
Convolution Coding	√	√	√	√
Turbo Coding		√	√	
Randomi-zation	√	√	√	√
CRC	√	√	√	

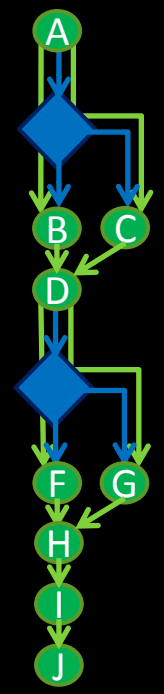
Design principle II

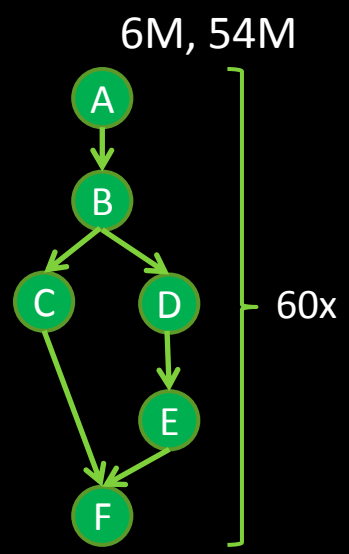
Decision-processing separation

- Logic pulled out to *decision plane SW*
- Branch free actions in the *processing plane SW*

- Deterministic execution times for blocks/actions
- Algorithmic schedule with pipelining
  - Analogous to instruction scheduling
  - Blocks = Instructions, Actions = Loops

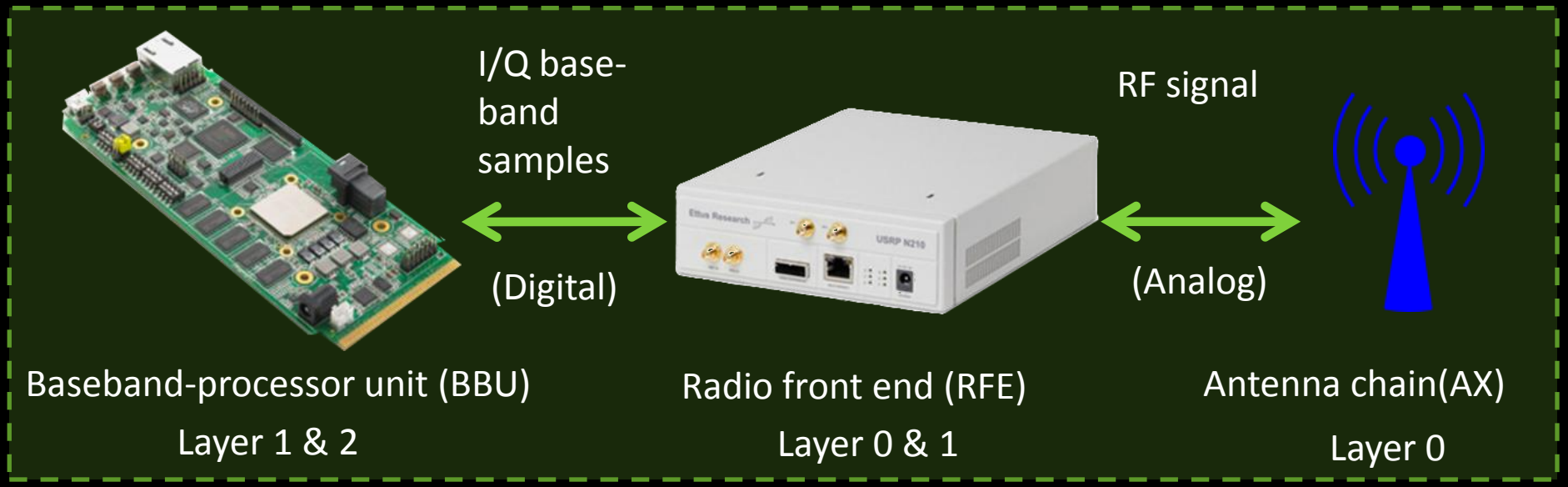
- Meet deadlines reliably (or deduce infeasibility)
- Abstract away the hardware





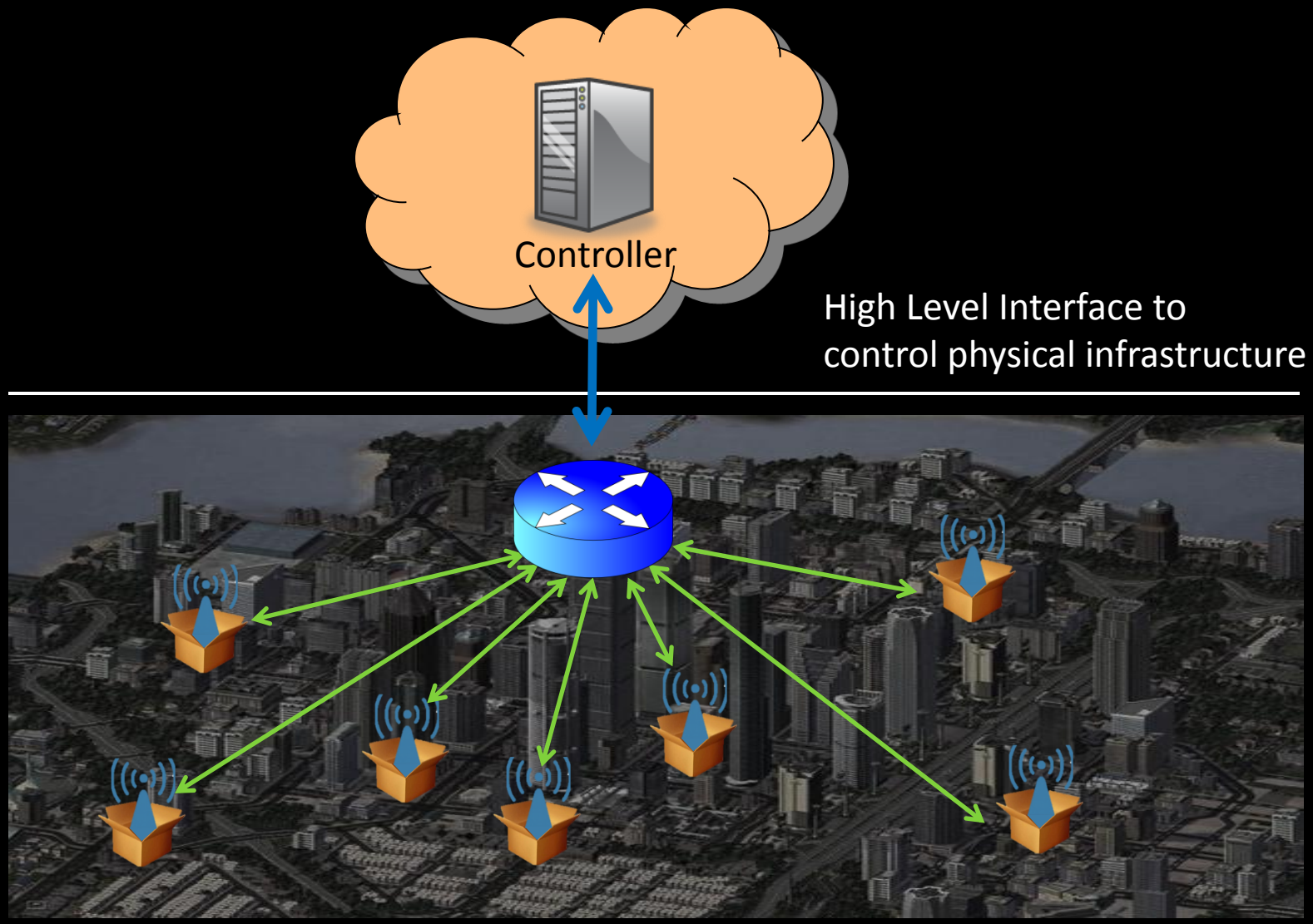
PRELIMINARY IMPLEMENTATION

Prototype



- Off-the-shelf TI KeyStone multicore DSP platform (EVM6618, two chips with 4 cores each at 1.2GHz)
- Configurable hardware accelerators for common, heavy processing blocks (eg. FFT, Viterbi, Turbo)
- USRP2 for RF conversion, I/Q sample stream
- Prototype can process 2 x 20MHz, 54Mbps
  - Room left for implementing variations and optimizations

OpenRadio architecture



Related work

- OpenRadio is not a software radio
  - Judicious tradeoff between flexibility of pure software and performance of ASICs
- OpenRadio is not a protocol stack, it is an enabler
  - Eg. LTE can be implemented conveniently with OpenRadio

Conclusion

- A programmable wireless dataplane
  - Rich programming interface for wireless radios
  - Principled design for efficient implementation
  - Built using off-the-shelf components
- Unique balance of flexibility, performance and modularity

Thanks! Questions?

snsg.stanford.edu/openradio

BACKUP SLIDES

Challenges

- Can these programming abstractions be implemented efficiently?
  - more than 100Gflops
- Can we meet processing deadlines reliably?
  - as tight as 25us for 2ms computation run

Design limitations

- Design works well for bulk of computation coming from processing plane
- Heavy decision-planes will cause performance bottlenecks and inefficient hardware use
- Model assumes processing/decision separation is meaningful, blocks are small
- Logic-heavy blocks or heavily sequential, indecomposable blocks will not execute well on multi-core platforms

More Related work

- An SDN approach to wireless radios
- Same goals but different challenges
  - Heavy computational load
  - Strict deadlines
- OpenRadio is not a software radio
  - Judicious tradeoff between flexibility of pure software and performance of ASICs
- Design is not tied to a specific hardware
  - Can implement on an FPGA or a desktop machine
  - Net performance is a function of hardware capabilities
  - Heterogeneous multicore platform is one good fit
- OpenRadio is not a protocol stack, it is an enabler
  - Eg. LTE can be implemented conveniently with OpenRadio

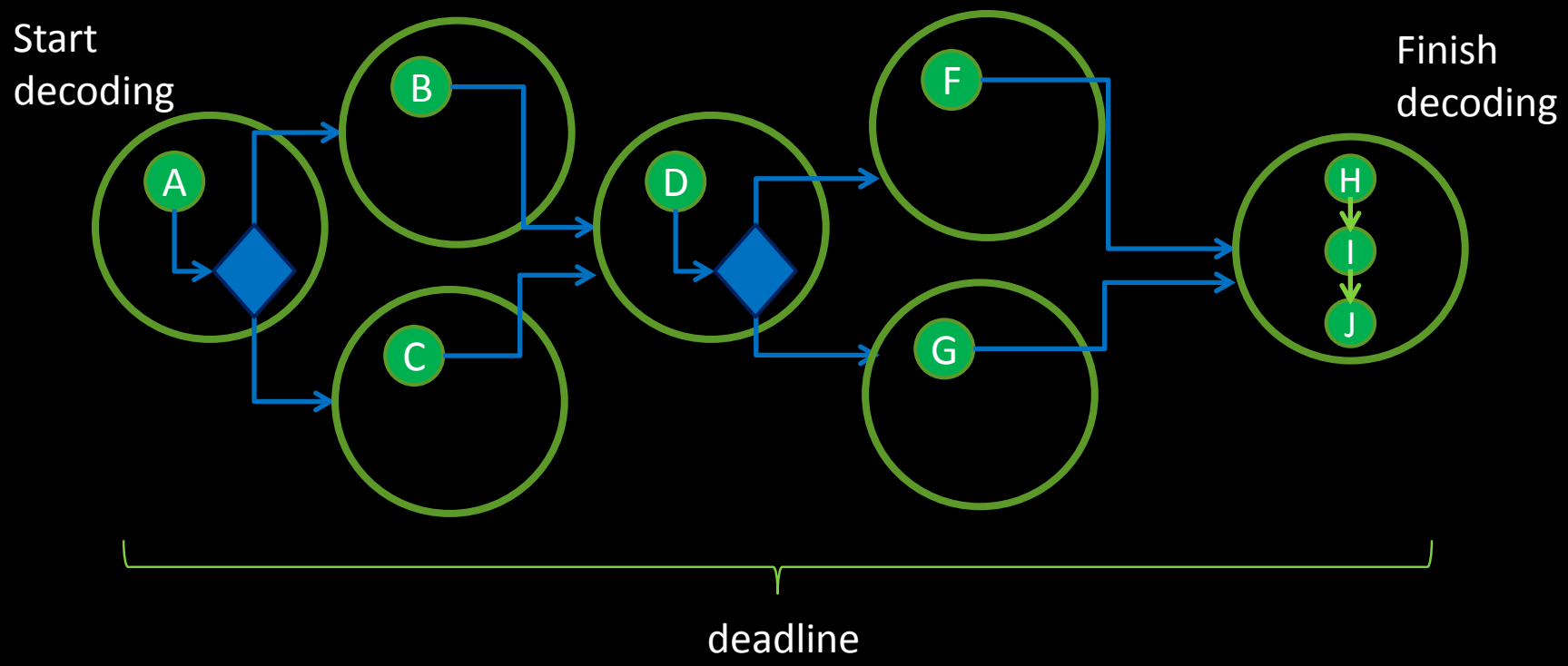
Rule-action programming model

- Protocols can be tied together using “rules” and “actions”
- Actions are DAGs of processing plane blocks
- Rules define the logic to conditionally pick DAGs

```
Rule: if (data packet  and wifi_6mbps)
    Action: BPSK and 1/2 rate
Rule: if (data packet and CRC match)
    Action: Send ACK
Rule: if (video packet)
    Action: UEP decoding
```

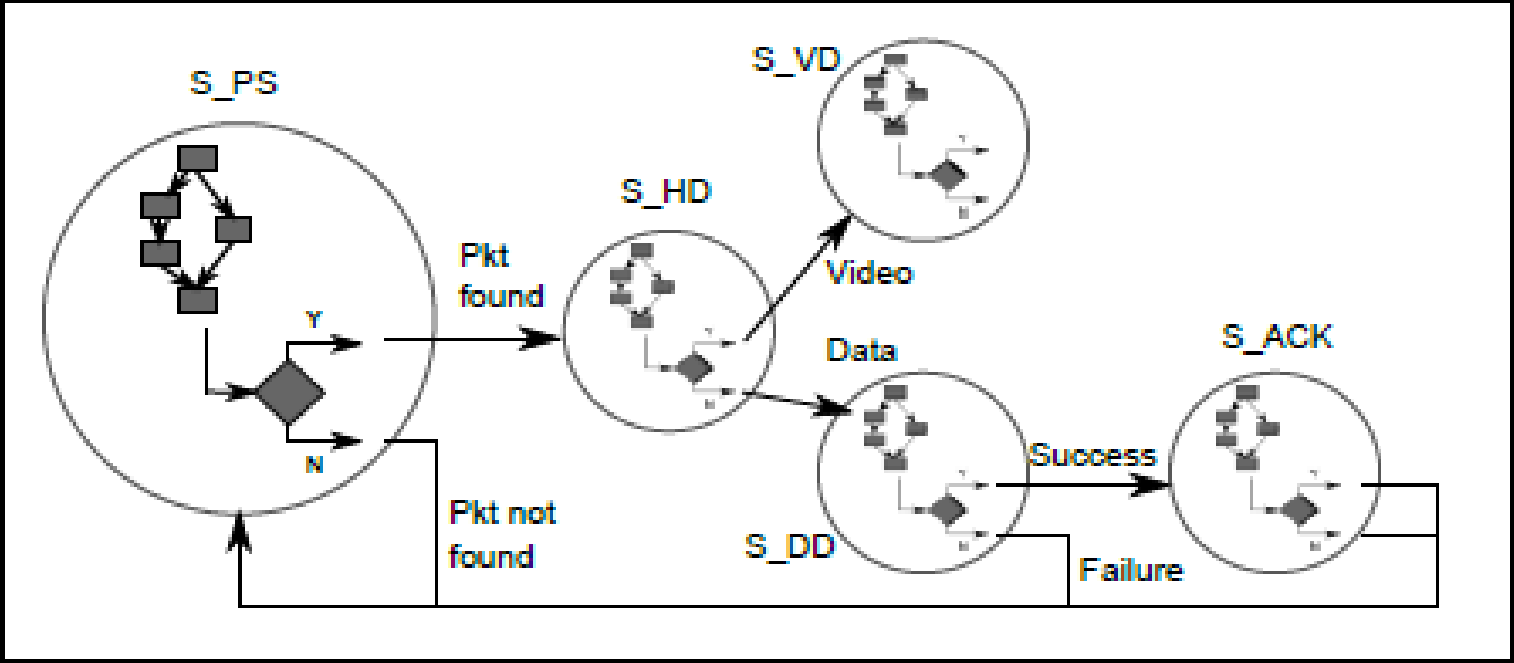
State machines and deadlines

- Rules and actions encode the protocol state machine
  - Rules define state transitions
  - Each state has an associated action
- Deadlines are expressed on state sequences



State machines and deadlines

State_HeaderDecode (S_HD):
Action HeaderDecode
Rule: if (data packet) transition to State_DataDecode (S_DD)
[Deadline: finishing S_DD by Deadline_DD from now]
Rule: if (video_packet) transition to State_VideoDecode (S_VD)
[Deadline: finishing S_VD ASAP]



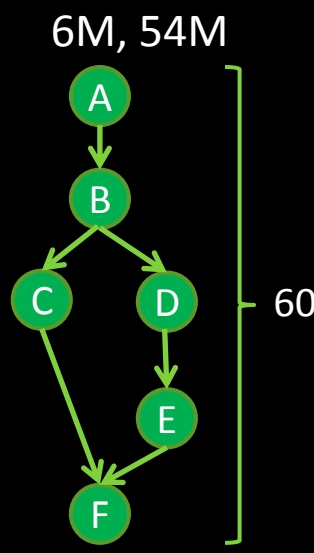
Design principle II
Decision-processing separation

- Logic pulled out to *decision plane SW*
- Branch free actions in the *processing plane SW*

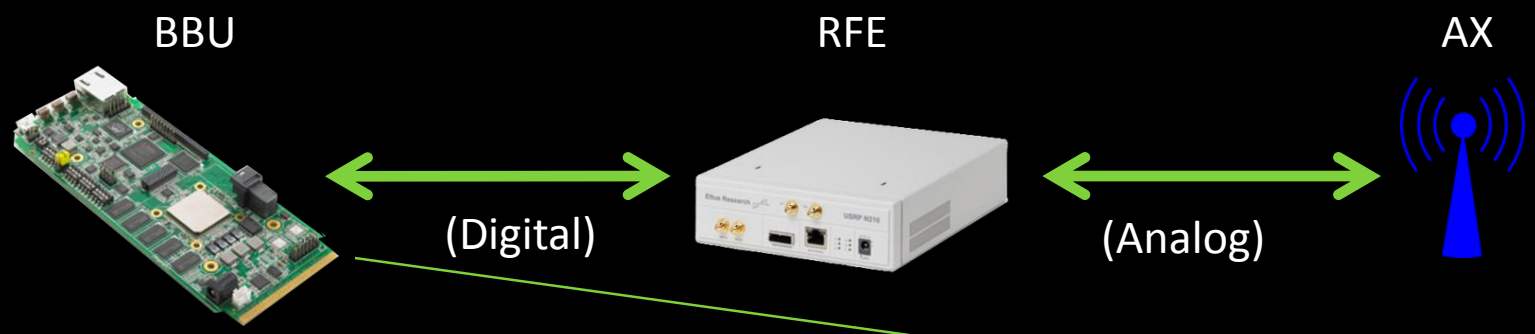
- Deterministic execution times for blocks/actions
- Efficient pipelining, algorithmic scheduling

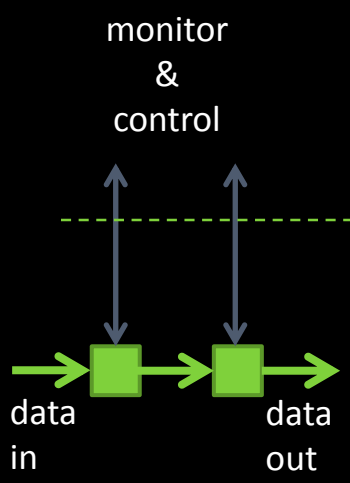
Regular compilation	OpenRadio scheduling
Instructions	Atomic processing blocks
Heterogeneous functional units	Heterogeneous cores
Known cycle counts	Predictable cycle counts
Argument data dependency	FIFO queue data dependency

- Meet deadlines reliably (or deduce infeasibility)
- Hardware is abstracted out



Software architecture





OR Wireless Decision Plane
protocol state machine, flowgraph composition, block configurations, knowledge plane, RFE control logic

OR Wireless Processing Plane
deterministic signal processing blocks, header parsing, channel resource scheduling, multicore fifo queues, sample I/O blocks

Bare-metal with drivers

OR Runtime System
compute resource scheduling, deterministic execution ensuring protocol deadlines are met

Anticipated questions

- What about the UE side?
  - UE side evolves much faster and incrementally
- Mostly talked about PHY. Is it just about PHY?
  - The dataplane refers to both PHY and MAC. In fact, the boundary between PHY and MAC does not exist for the dataplane. They are both made up of processing blocks and decision logic. An example for MAC is the decomposition of channel scheduler – the decision plane involves finding the mapping of data to channel resources, the processing plane operation is to actually map data into its correct resource block. Our ongoing work includes studying concrete cases, design of interfaces best suited to MAC and the balance between processing and decision plane loads.
- Goal is cellular basestations but you study WiFi?
  - Yes. WiFi has similar computation requirements being 20MHz OFDM/54Mbps and much more stringent deadlines (25us) than LTE or WiMAX. Though solving WiFi does not imply solving LTE, it is a strong proof of concept.
- What is the unit of data on which the blocks operate?
  - Blocks generally have a natural granularity of operation, for example, an OFDM symbol worth of data (FFT works on full symbol as the smallest unit). Smaller data units mean smaller pipeline latencies. You can always increase the data unit size in multiples of the smallest unit, if your latency budget permits.