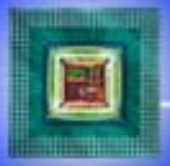
Network Processors: Building Block for programmable networks

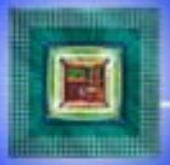
Raj Yavatkar
Chief Software Architect
Intel® Internet Exchange Architecture

raj.yavatkar@intel.com

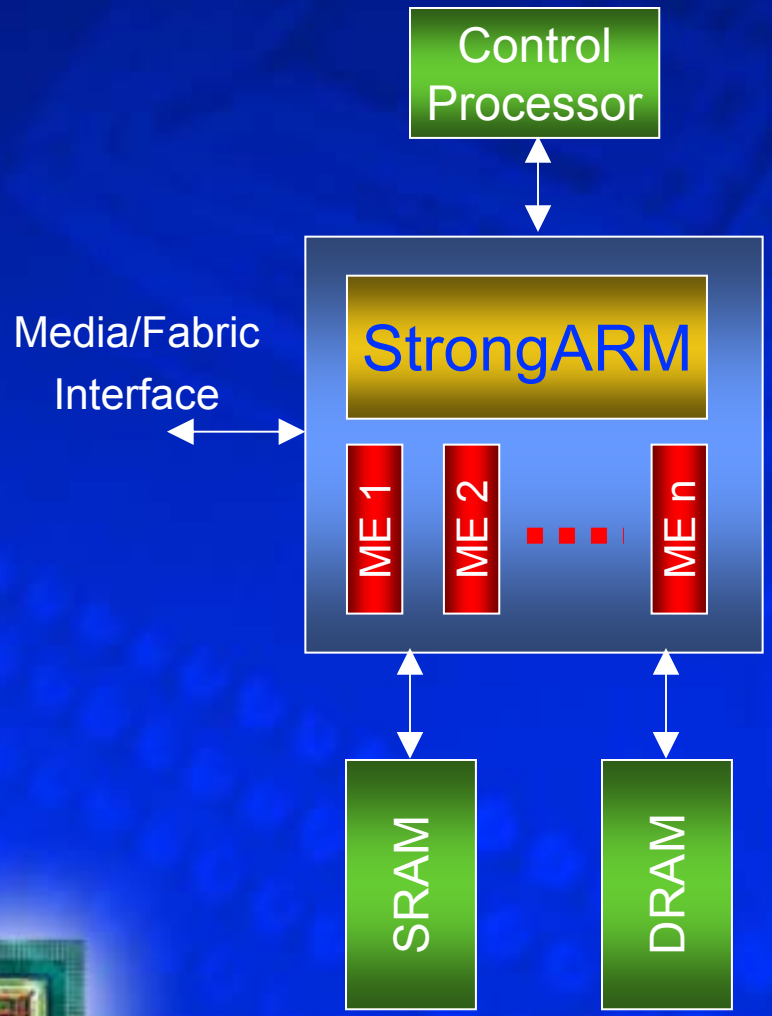


Outline

- IXP 2xxx hardware architecture
- IXA software architecture
- Usage questions
- Research questions



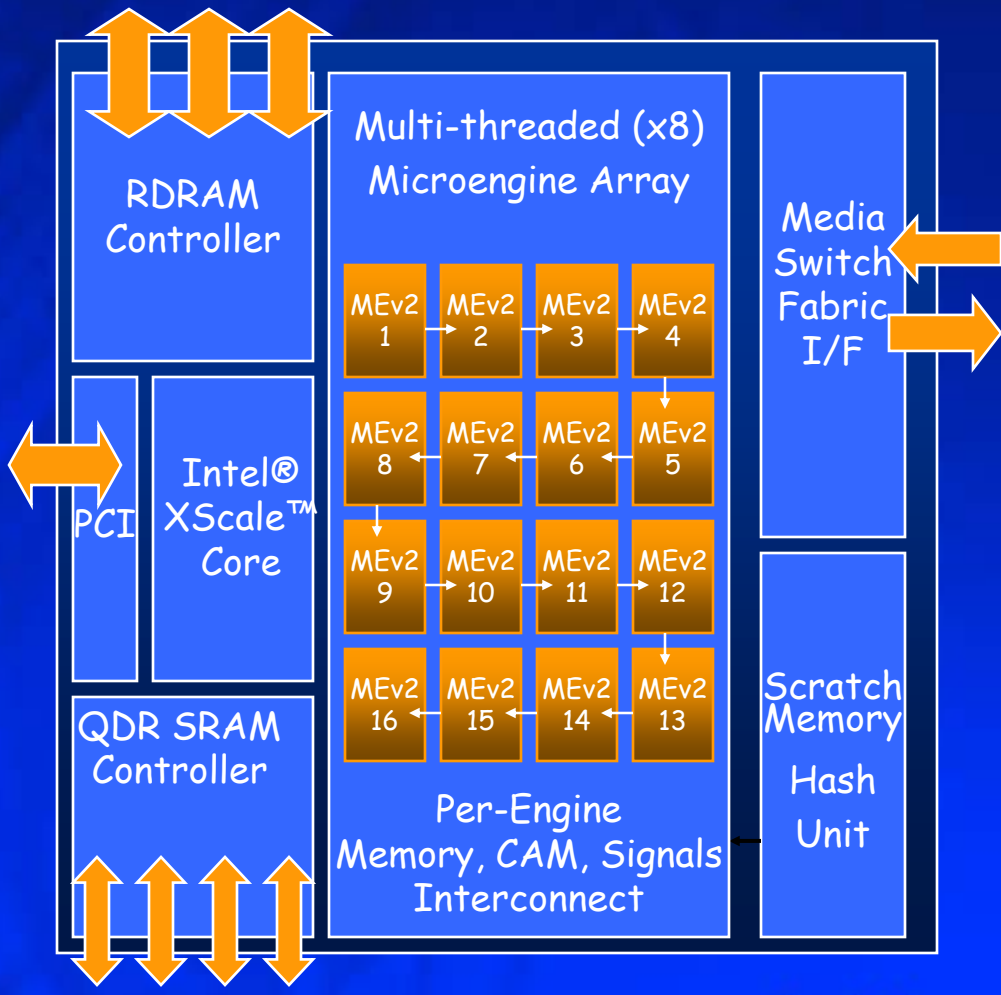
IXP Network Processors



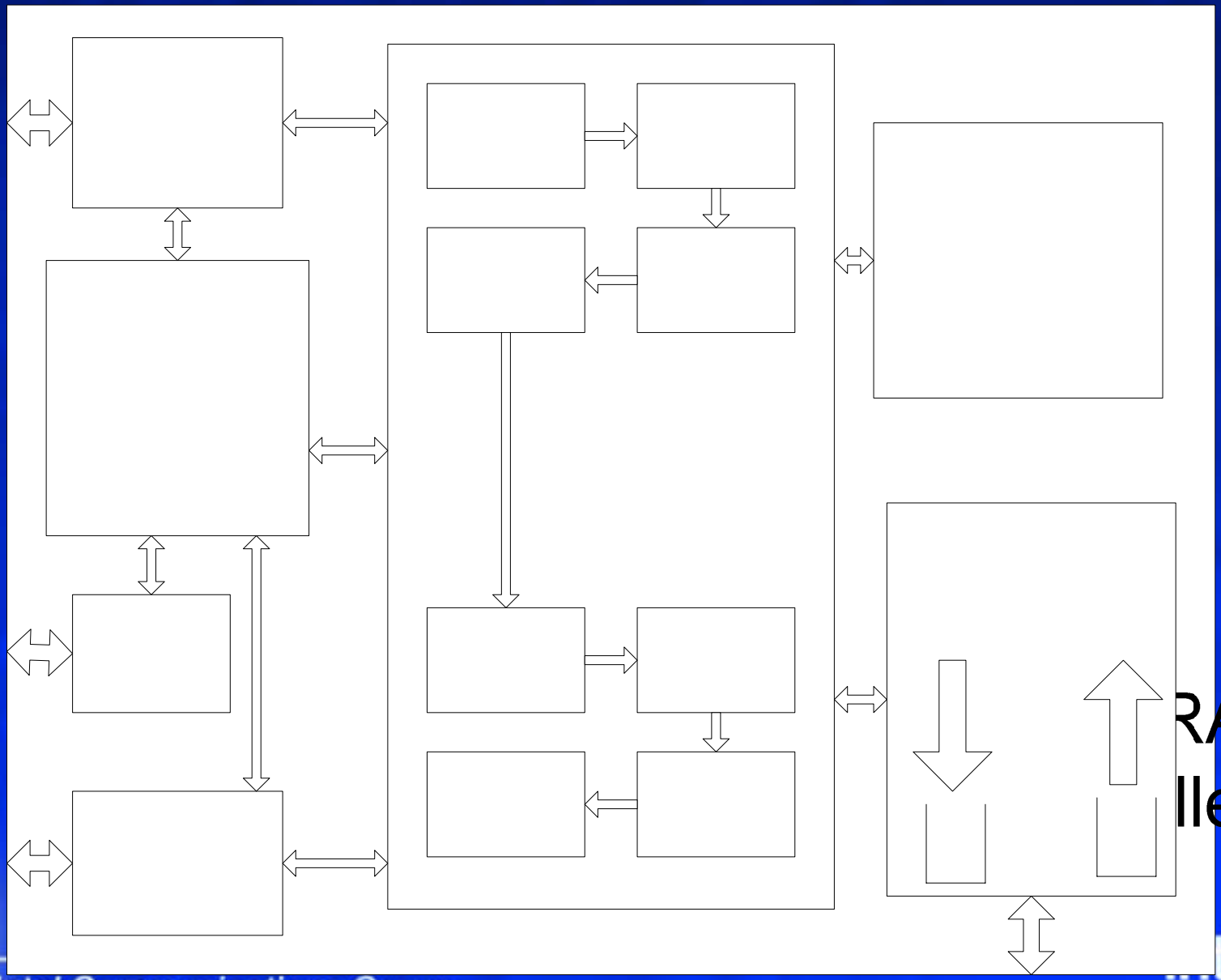
- **Microengines**
  - RISC processors optimized for packet processing
  - Hardware support for multi-threading
- **Embedded StrongARM/Xscale**
  - Runs embedded OS and handles exception tasks

IXP: A Building Block for Network Systems

- Example: IXP2800
  - 16 micro-engines + XScale core
  - Up to 1.4 Ghz ME speed
  - 8 HW threads/ME
  - 4K control store per ME
  - Multi-level memory hierarchy
  - Multiple inter-processor communication channels
- NPU vs. GPU tradeoffs
  - Reduce core complexity
    - No hardware caching
    - Simpler instructions ➔ shallow pipelines
  - Multiple cores with HW multi-threading per chip

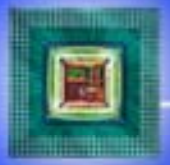


IXP 2400 Block Diagram



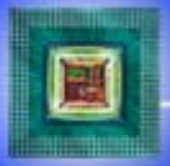
XScale Core processor

- Compliant with the ARM V5TE architecture
  - support for ARM's thumb instructions
  - support for Digital Signal Processing (DSP) enhancements to the instruction set
  - Intel's improvements to the internal pipeline to improve the memory-latency hiding abilities of the core
  - does not implement the floating-point instructions of the ARM V5 instruction set

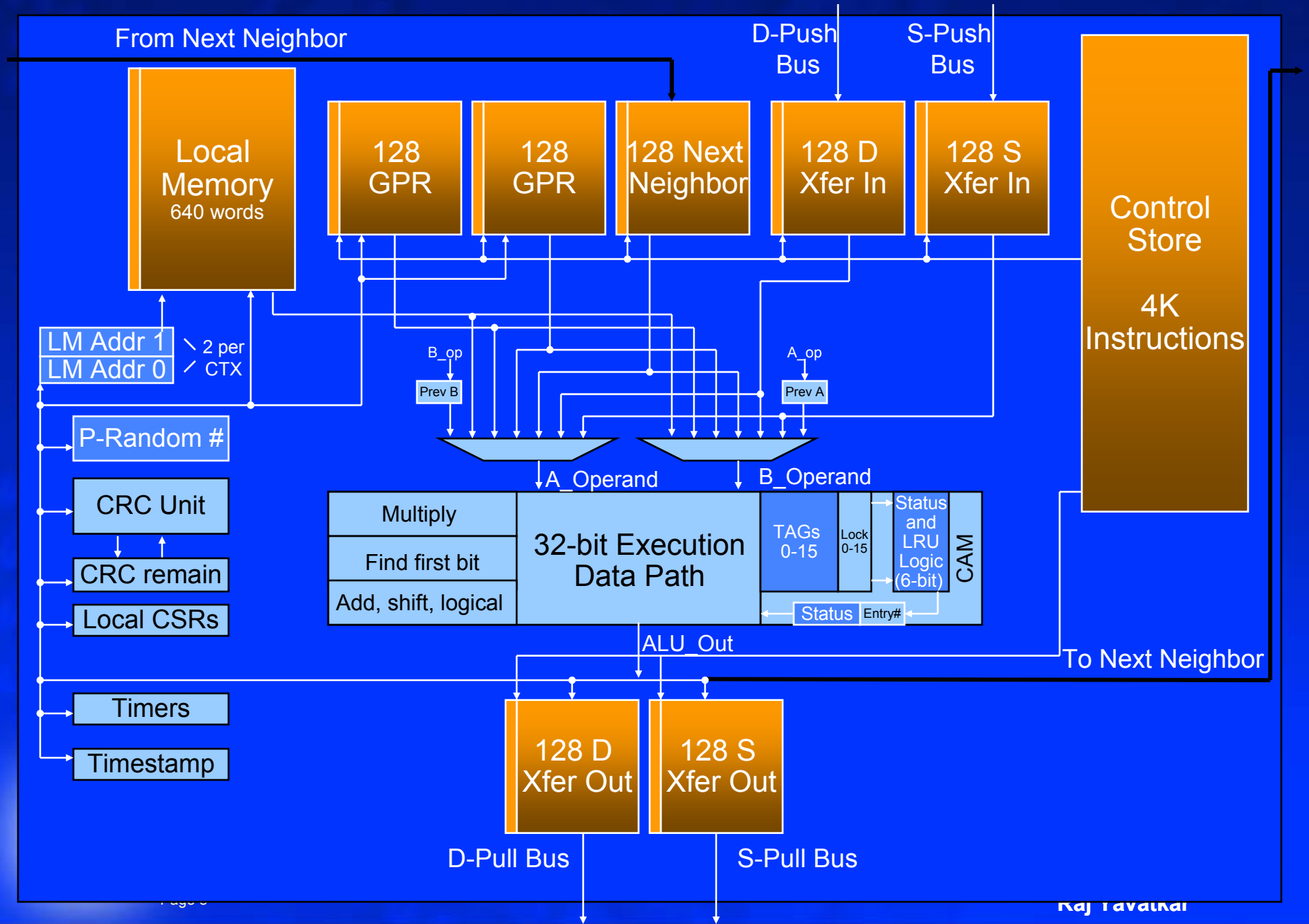


Microengines – RISC processors

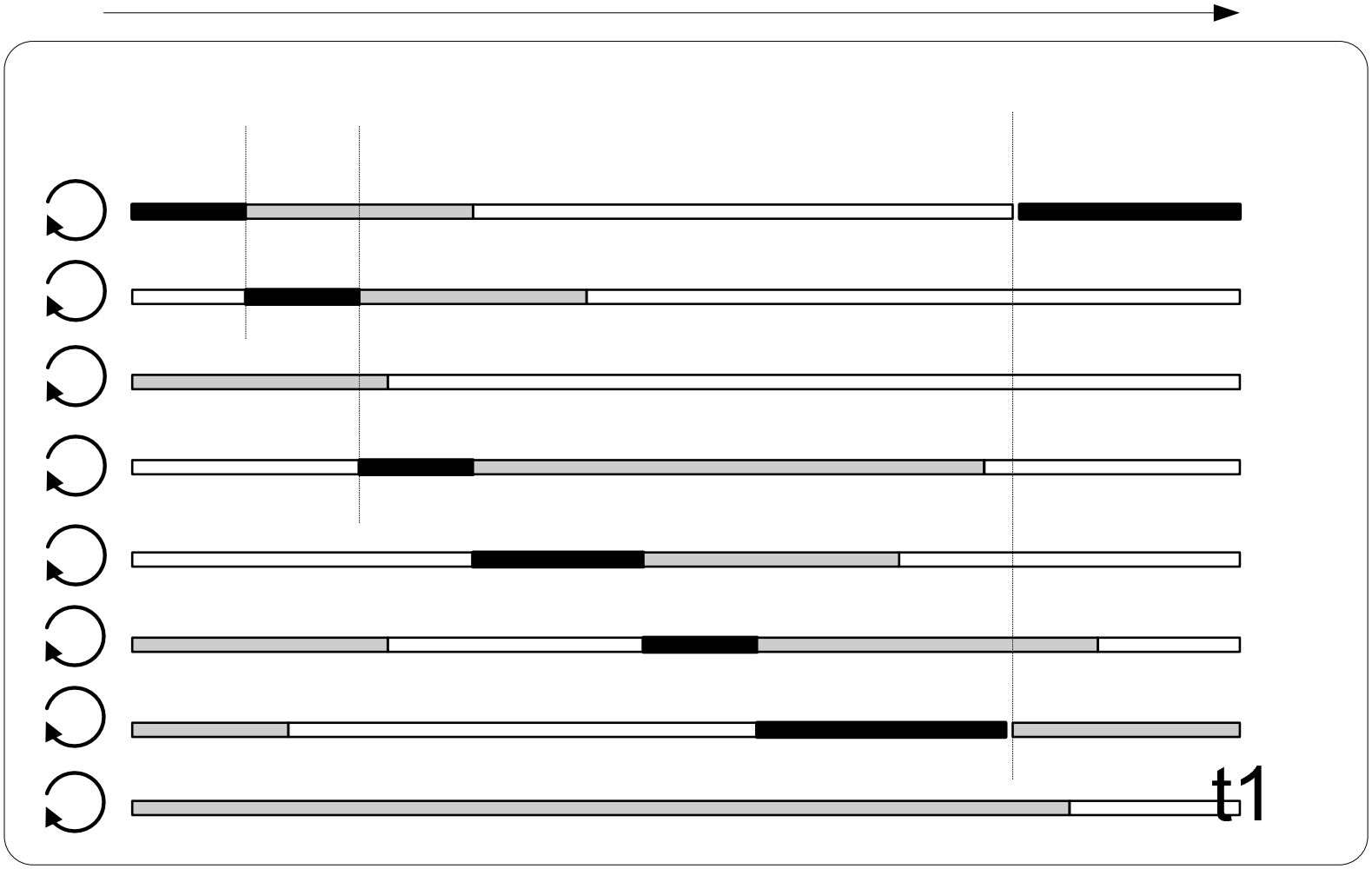
- IXP 2800 has 16 microengines, organized into 4 clusters (4 MEs per cluster)
- ME instruction set specifically tuned for processing network data
  - Arithmetic and Logical operations that operate at bit, byte, and long-word levels
  - can be combined with shift and rotate operations in single instructions.
  - integer multiplication provided; no division or FP operations
- 40-bit x 4K control store
- six-stage pipeline in an instruction
  - On an average takes one cycle to execute
- Each ME has eight hardware-assisted threads of execution
  - can be configured to use either all eight threads or only four threads
- The non-preemptive hardware thread arbiter swaps between threads in round-robin order



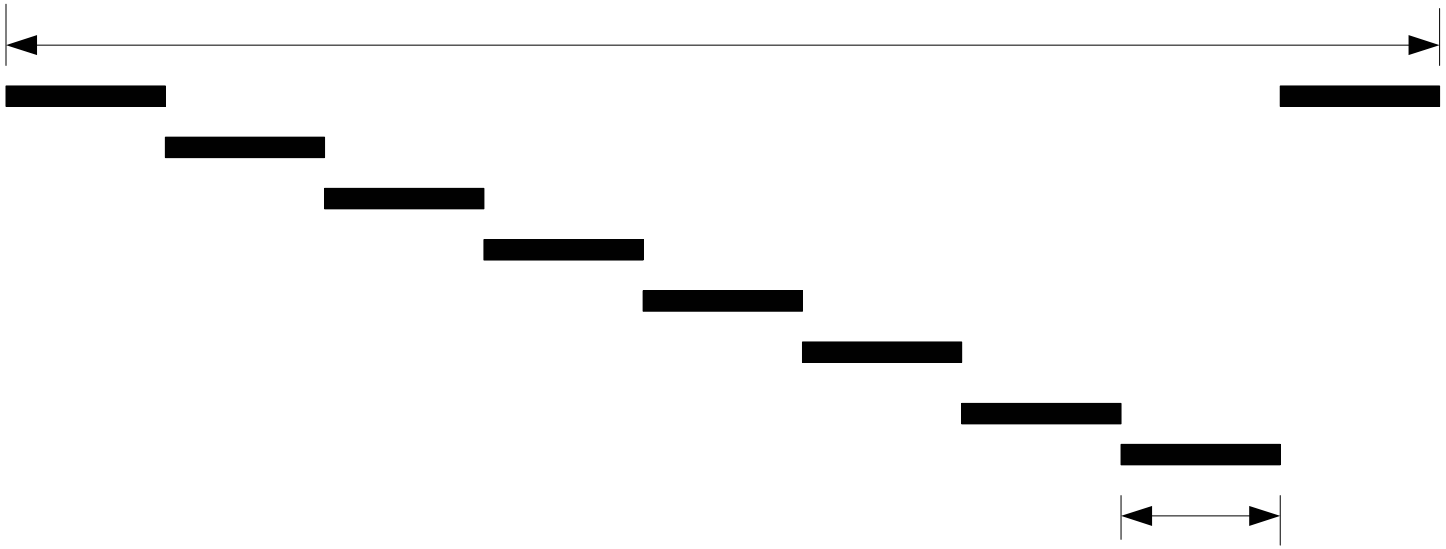
MicroEngine v2



Why mult-ithreading?

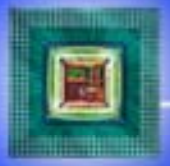


Packet processing using multi-threading within a MicroEngine



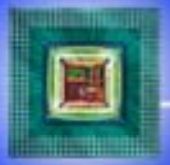
Registers available to each ME

- four different types of registers
  - general purpose, SRAM transfer, DRAM transfer, next-neighbor (NN)
  - Also, access to many CSRs
- 256, 32-bit GPRs
  - can be accessed in thread-local or absolute mode
- 256, 32-bit SRAM transfer registers.
  - used to read/write to all functional units on the IXP2xxx except the DRAM
- 256, 32-bit DRAM transfer registers
  - divided equally into read-only and write-only
  - used exclusively for communication between the MEs and the DRAM
- Benefit of having separate transfer and GPRs
  - ME can continue processing with GPRs while other functional units read and write the transfer registers



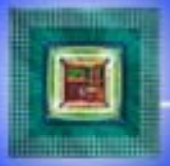
Next-Neighbor Registers

- Each ME has 128, 32-bit next-neighbor registers
  - makes data written in these registers available in the next microengine (numerically)
  - E.g., if ME 0 writes data into a next-neighbor register, ME 1 can read the data from its next-neighbor register, and so on
- In another mode, these registers are used as extra GPRs
  - Data written into a next-neighbor register is read back by the same microengine



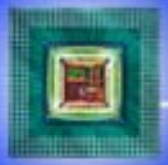
Generalized thread signaling

- Each ME thread has 15 numbered signals.
- Most accesses to functional units outside of the ME can cause a signal to any one signal number
- The signal number generated for any functional unit access is under the programmer's control
- A ME thread can test for the presence or absence of any of these signals
  - used to control branching on the signal presence
  - Or, to specify to the thread arbiter that a ME thread is ready to run only after the signal is received
- Benefit of the approach
  - software can have multiple outstanding references to the same unit and wait for all of them to complete using different signals



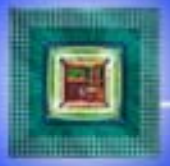
Different Types of Memory

Type of Memory	Logical width (bytes)	Size in bytes	Approx unloaded latency (cycles)	Special Notes
Local to ME	4	2560	3	Indexed addressing post incr/decr
On-chip scratch	4	16K	60	Atomic ops 16 rings w/at. get/put
SRAM	4	256M	150	Atomic ops 64-elem q-array
DRAM	8	2G	300	Direct path to/fro MSF



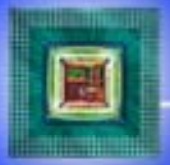
IXP2800 Features

- Half Duplex OC-192 / 10 Gb/sec Ethernet Network Processor
- XScale Core
  - 700 MHz (half the ME)
  - 32 Kbytes instruction cache / 32 Kbytes data cache
- Media / Switch Fabric Interface
  - 2 x 16 bit LVDS Transmit & Receive
  - Configured as CSIX-L2 or SPI-4
- PCI Interface
  - 64 bit / 66 MHz Interface for Control
  - 3 DMA Channels
- QDR Interface (w/Parity)
  - (4) 36 bit SRAM Channels (QDR or Co-Processor)
  - Network Processor Forum LookAside-1 Standard Interface
  - Using a “clamshell” topology both Memory and Co-processor can be instantiated on same channel
- RDR Interface
  - (3) Independent Direct Rambus DRAM Interfaces
  - Supports 4i Banks or 16 interleaved Banks
  - Supports 16/32 Byte bursts



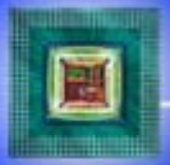
Hardware Features to ease packet processing

- Ring Buffers
  - For inter-block communication/synchronization
  - Producer-consumer paradigm
- Next Neighbor Registers and Signaling
  - Allows for single cycle transfer of context to the next logical micro-engine to dramatically improve performance
  - Simple, easy transfer of state
- Distributed data caching within each micro-engine
  - Allows for all threads to keep processing even when multiple threads are accessing the same data



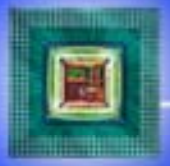
Outline

- IXP 2xxx hardware architecture
- IXA software architecture
- Usage questions
- Research questions

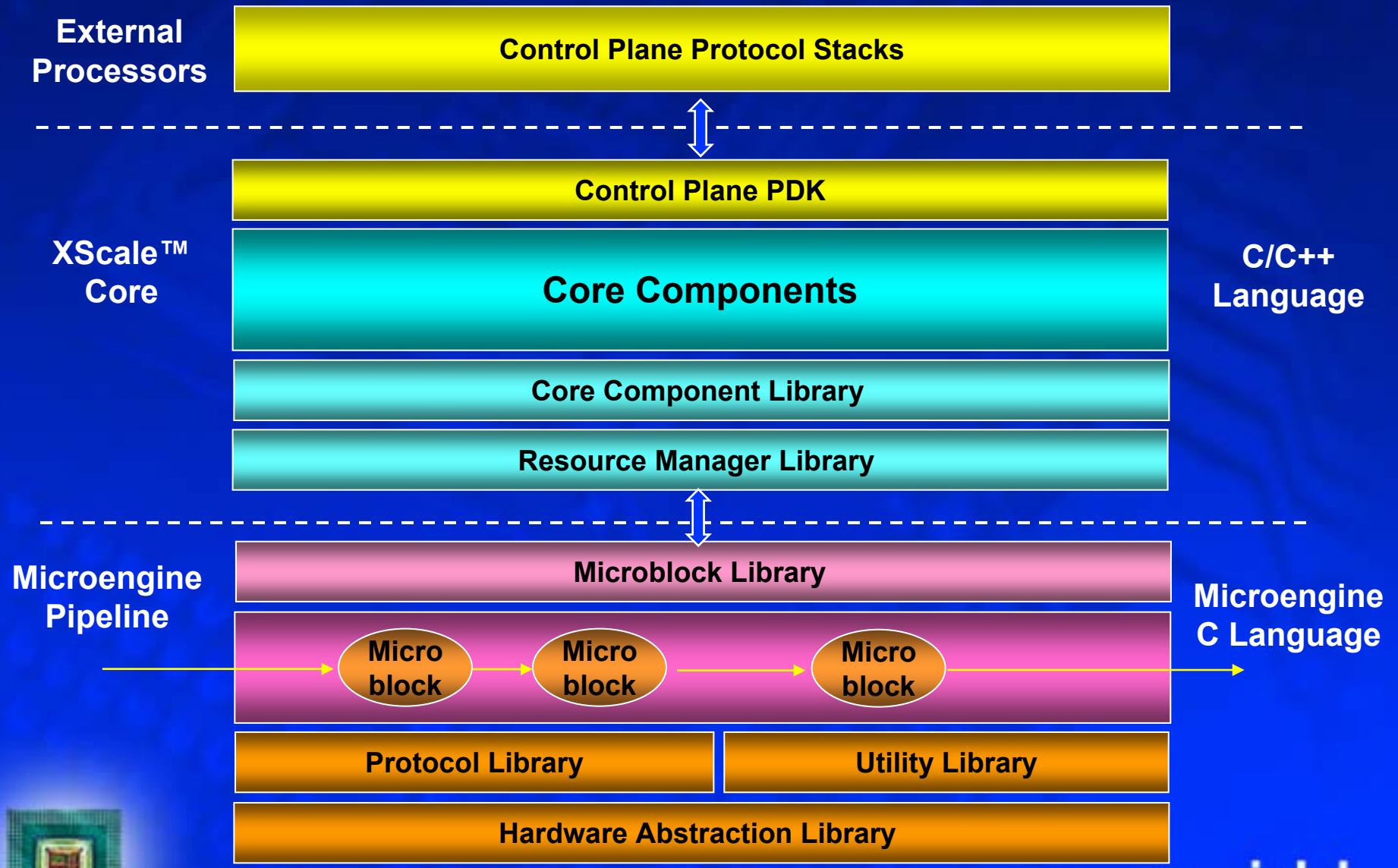


IXA Portability Framework - Goals

- Accelerate software development for the IXP family of network processors
- Provide a simple and consistent infrastructure to write networking applications
- Enable reuse of code across applications written to the framework
- Improve portability of code across the IXP family
- Provide an infrastructure for third parties to supply code
  - for example, to support TCAMs

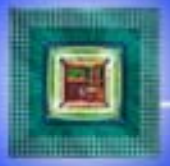


IXA Software Framework



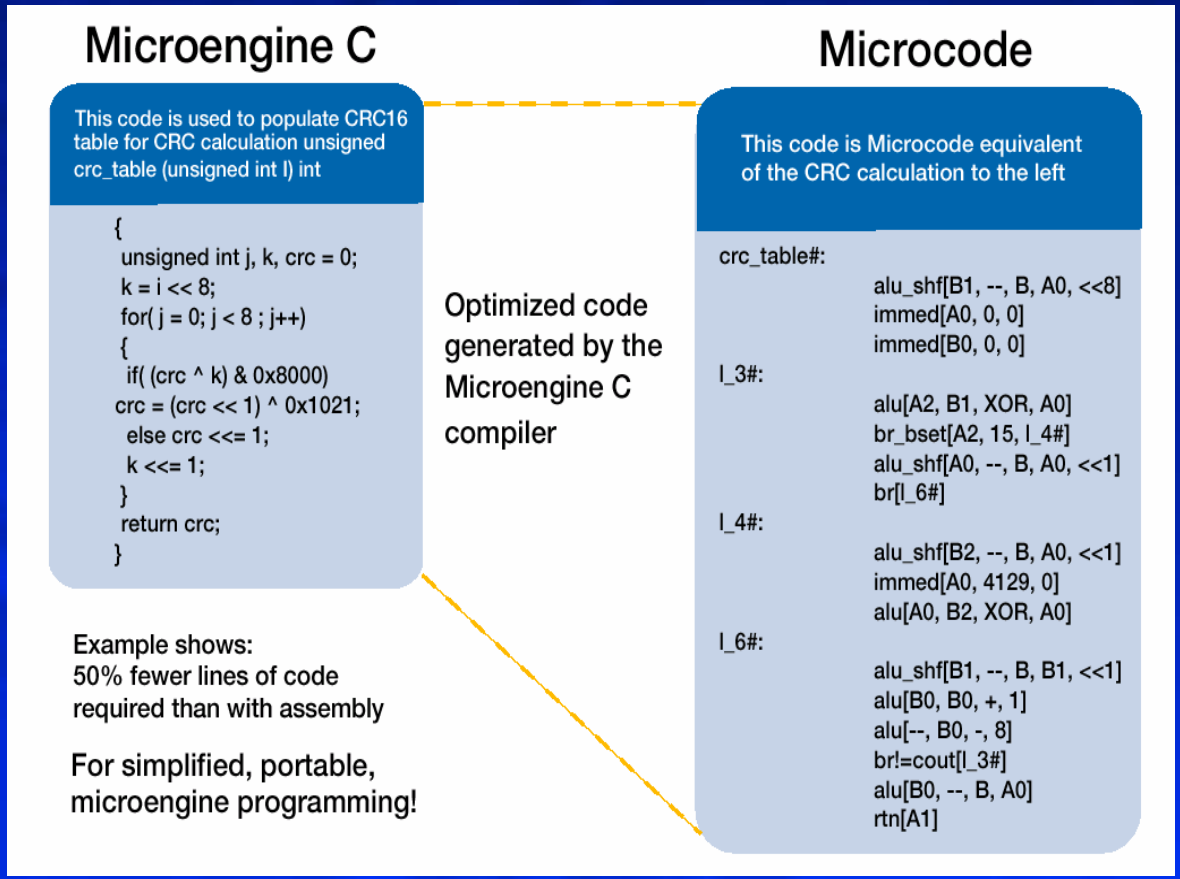
Software Framework on the MEv2

- Microengine C compiler (language)
- Optimized Data Plane Libraries
  - Microcode and MicroC library for commonly used functions
- Microblock Programming Model
  - Enables development of modular code building blocks
  - Defines the data flow model, common data structures, state sharing between code blocks etc
  - Ensures consistency and improves reuse across different apps
- Core component library
  - Provides a common way of writing slow-path components that interact with their counterpart fast-path code
- Microblocks and example applications written to the microblock programming model
  - IPv4/IPv6 Forwarding, MPLS, DiffServ etc.



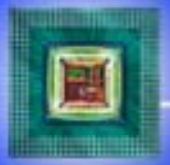
Micro-engine C Compiler

- C language constructs
  - Basic types, pointers, bit fields
- In-line assembly code support
- Aggregates
  - Structs, unions, arrays
  - Intrinsics for specialized ME functions
  - Different memory models and special constructs for data placement (e.g., *__declspec(sdram) struct msg_hdr hd)*

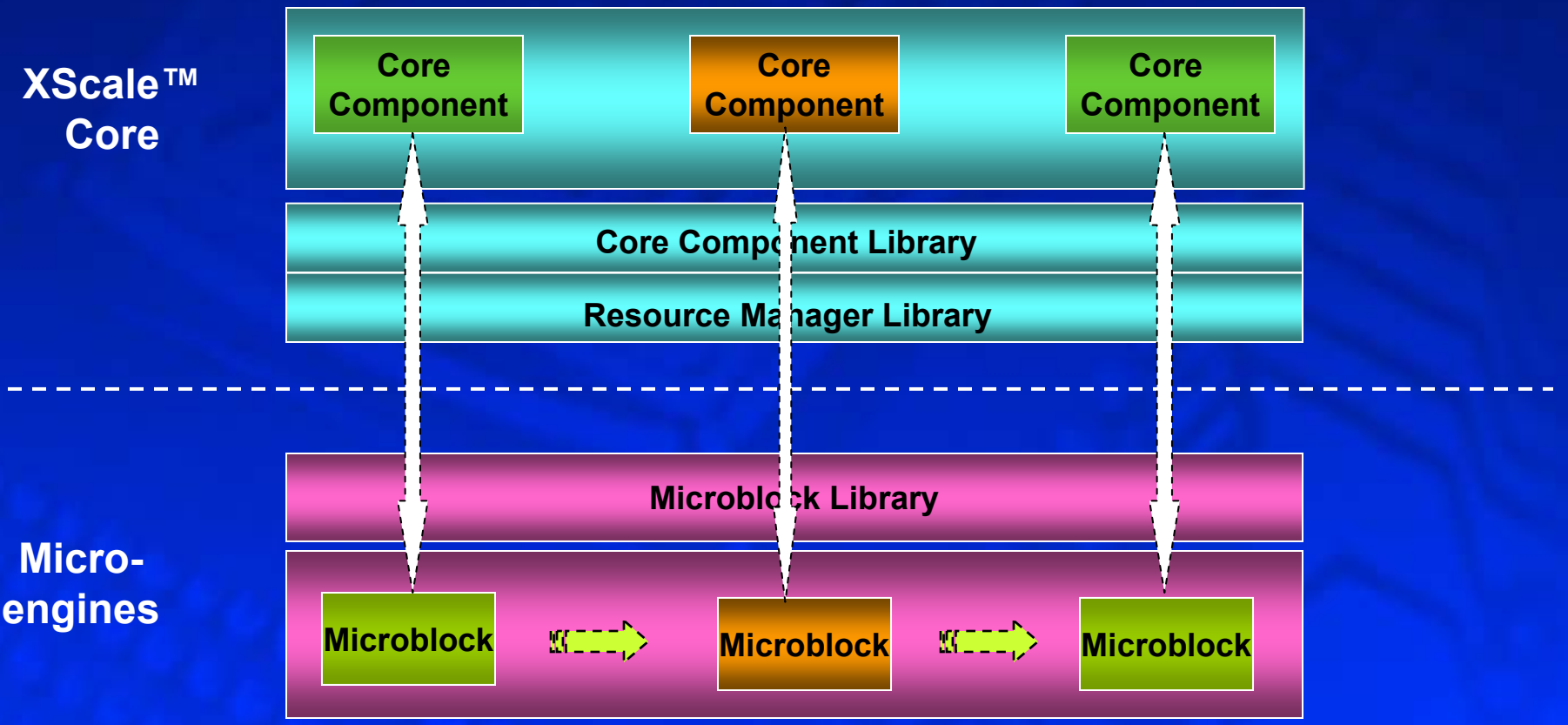


What is a Microblock?

- Data plane packet processing on the microengines is divided into logical functions called microblocks
- Coarse Grain and stateful
- Example
  - 5-Tuple Classification
  - IPv4 Forwarding
  - NAT
- Several microblocks running on a microengine thread can be combined into a microblock group.
  - A microblock group has a *dispatch loop* that defines the dataflow for packets between microblocks
  - A microblock group runs on each thread of one or more microengines
- Microblocks can send and receive packets to/from an associated Xscale Core Component



Core Components and Microblocks



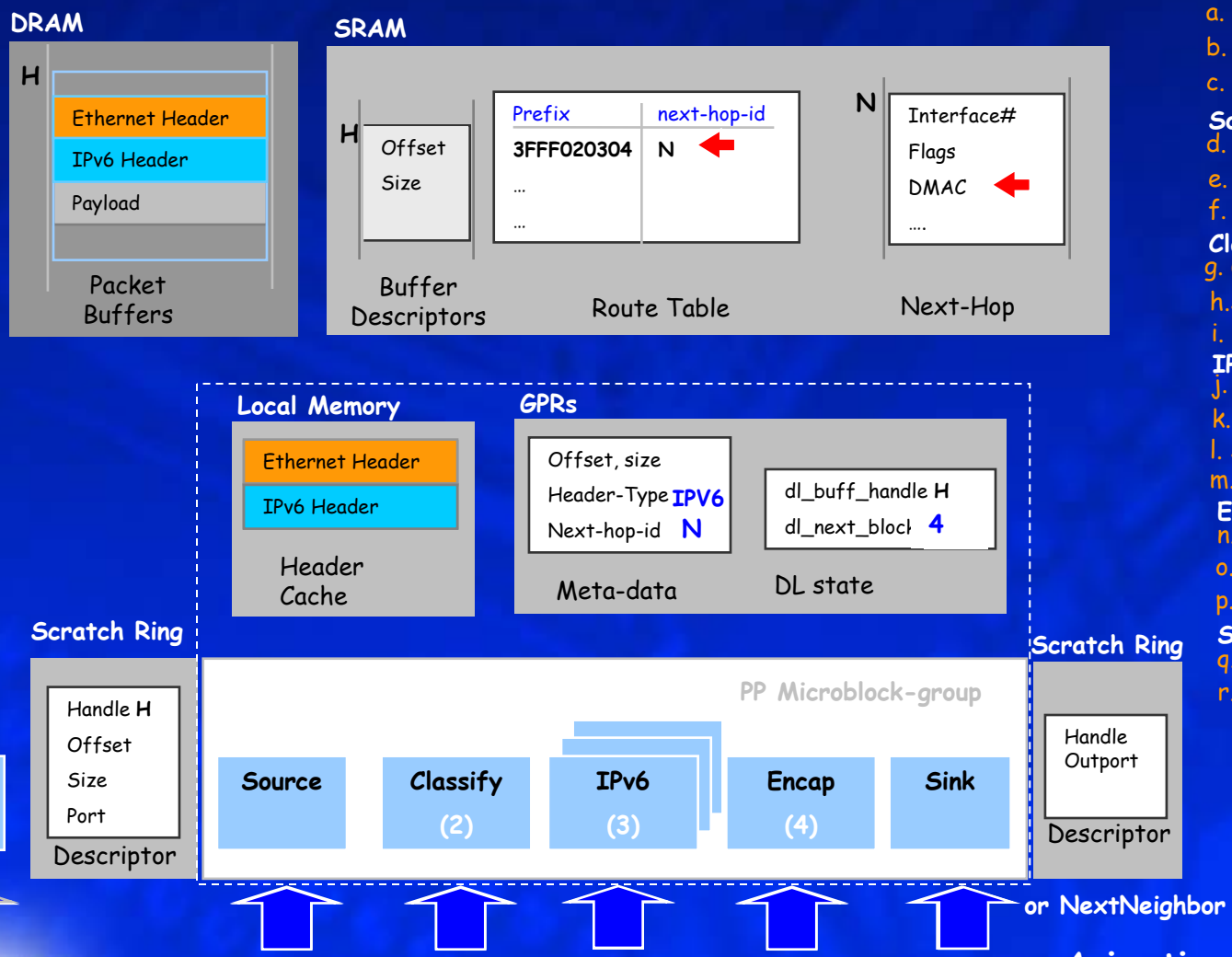
Microblock Library

Intel/3rd party blocks

User-written code

Core Libraries

Simplified Packet Flow (IPv6 example)



Rx
a. Put Packet in DRAM
b. Put Descriptor in SRAM
c. Queue Handle on ring

Source
d. Pull meta-data in GPRs
e. Set DL state in GPRs
f. Set next_blk = Classify

Classify
g. Get Headers in HCache
h.Set HeaderType to IPv6
i. Set next_blk = IPv6

IPv6
j. Get DAddr from HCache
k. Search RouteTable
l. Set next-hop-id = N
m. Set next_blk = Encap

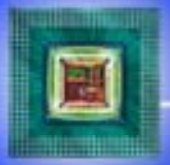
Encap
n. Get DMAC from next-hop N
o. Set Eth Hdr in HCache
p. Flush HCache to DRAM

Sink
q. Flush Meta-data to SRAM
r. Queue Handle to Ring

Animation: press PgDn 19 times (PgUp to backup)

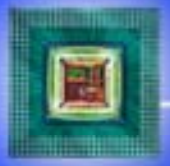
Outline

- IXP 2xxx hardware architecture
- IXA software architecture
- Usage questions
- Research questions



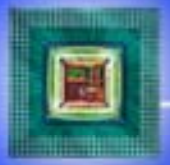
What can I do with an IXP?

- **Fully programmable architecture**
  - **Implement any packet processing applications**
    - **Examples from customers**
      - Routing/switching, VPN, DSLAM, Multi-servioce switch, storage, content processing
    - **Intrusion Detection (IDS) and RMON**
      - needs processing of many state elements in parallel
  - **Use as a research platform**
    - **Experiment with new algorithms, protocols**
  - **Use as a teaching tool**
    - **Understand architectural issues**
    - **Gain hands-on experience withy networking systems**



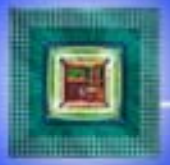
Technical and Business Challenges

- Technical Challengers
  - Shift from ASIC-based paradigm to software-based apps
  - Challenges in programming an NPU (next)
  - Trade-off between power, board cost, and no. of NPUs
  - How to add co-processors for additional functions?
- Business challenges
  - Reliance on an outside supplier for the key component
  - Preserving intellectual property advantages
  - Add value and differentiation through software algorithms in data plane, control plane, services plane functionality
  - Must decrease TTM to be competitive *(To NPU or not to NPU?)*



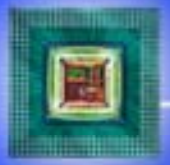
Outline

- IXP 2xxx hardware architecture
- IXA software architecture
- Usage questions
- Research questions



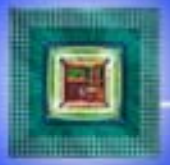
Architectural Issues

- How to scale up to OC-768 and beyond?
- What is the "right" architecture?
  - A set of reconfigurable processing engines
    vs
    carefully architected pipelined stages
    vs
    a set of fixed-function blocks
- Questionable hypotheses
  - No locality in packet processing?
    - Temporal vs spatial
    - Working set size vs available cache capacity
  - Little or no dependency in packets from different flows?



Challenges in Programming an NP

- Distributed, parallel programming model
  - Multiple microengines, multiple threads
- Wide variety of resources
  - Multiple memory types (latencies, sizes)
  - Special-purpose engines
  - Global and local synchronization
- Significantly different from the problem seen in scientific computing



NPU Programming Challenges

- Programming environments for NPUs and network systems are different from those for conventional multi-processors
- Automatic allocation of network system resources: Memory

Conventional MP systems

- Rely on locality of memory accesses to utilize memory hierarchy effectively
- Programmers program to a single-level memory hierarchy
- Compilers are unaware of the memory levels or their performance characteristics

Network systems

- Packet processing applications demonstrate little temporal locality
- Minimizing memory access latencies is crucial
- ***Compilers should manage memory hierarchy explicitly***
  - Allocate data structures to appropriate memory levels
  - Allocation depends on data structure sizes, access pattern, sharing requirements, memory system characteristics, …

Memory management is more complex in network systems

NPU Challenges - 2

- **Automatic allocation of network system resources: Processors**

Conventional MP systems

- Parallel compilers exploit loop- or function-level parallelism to utilize multiple processors to speed-up execution of programs
- Operating systems utilize idle processors to execute multiple programs in parallel

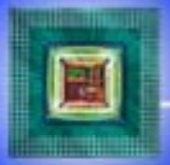
Network systems

- Individual packet processing is inherently sequential; little loop- or function-level parallelism
- Process packets belonging to different flows in parallel
- High-throughput and robustness requirements
  - *Compilers should create efficient packet processing pipelines*
    - Granularity of pipeline stage depends on instruction cache size, amount of communication between stages, computational complexity of stages, sharing and synchronization requirements, …

Network applications are explicitly parallel ➔ concurrency extraction is simpler;
but throughput and robustness requirements introduce a new problem of pipeline construction !

Challenges (contd.)

- How to enable a wide range of network applications
  - TCP offload/termination
  - How to distribute functionality between SA/Xscale, Pentium, and microengines?
    - Hierarchy of compute vs I/O capabilities
  - How to allow use of multiple IXPs to solve more compute intensive problems
- Networking research
  - How to take advantage of programmable, open architecture?
  - Designing “right” algorithms for LPM, range matching, string search, etc
  - QoS-related algorithms – TM4.1, WRED, etc



Questions?