Programming Challenges in Network Processor Deployment

Chidamber Kulkarni*, Christian Sauer, Matthias Gries, Kurt Keutzer

University of California, Berkeley

Infineon Technologies, Munich

* Now at Xilinx Inc

Outline of Talk

¥ Introduction to Network Processors

¥ IPv4 benchmark Implementation

 —Intel IXP1200

 —Motorola C-Port C-5

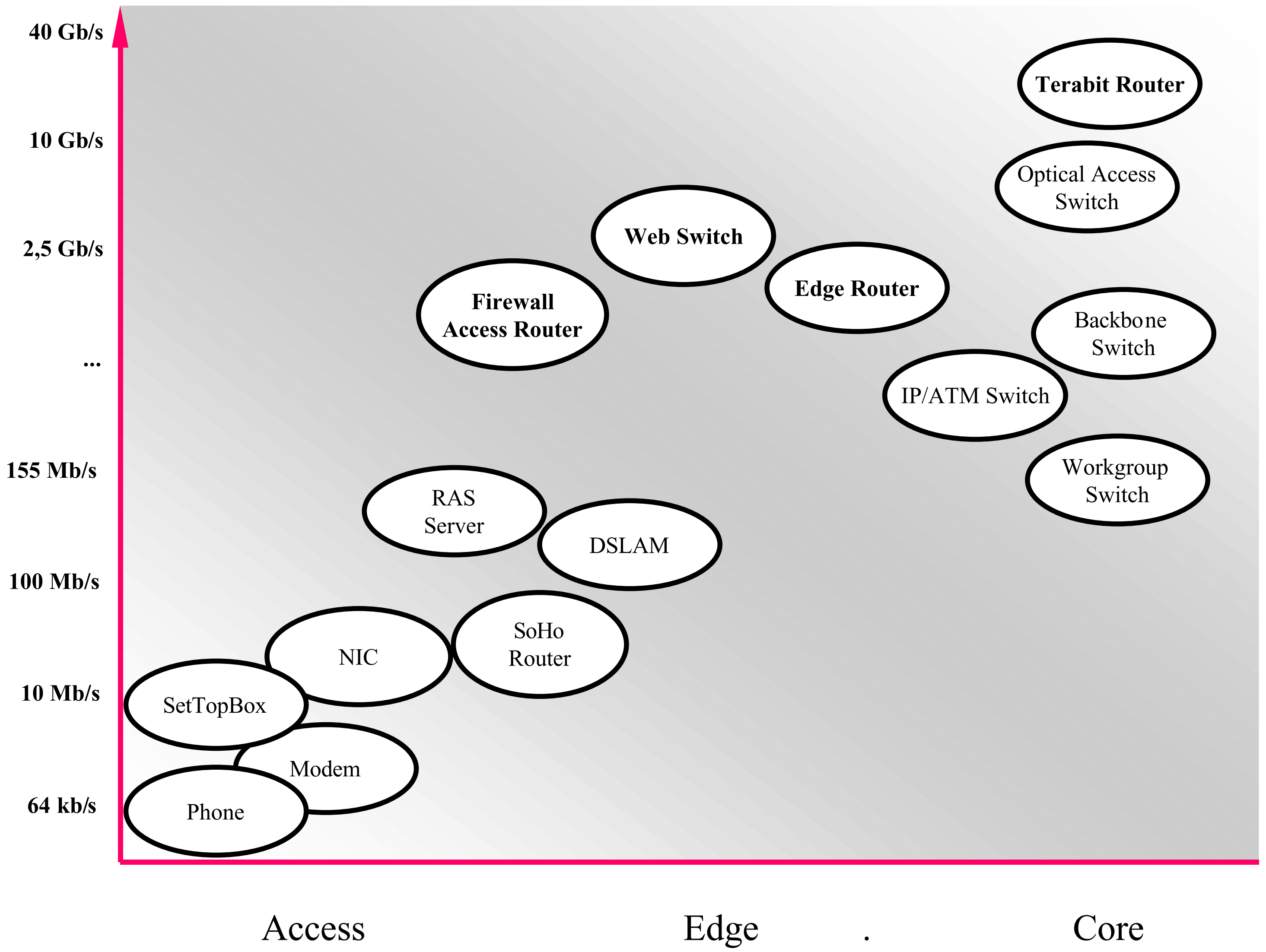
¥ Results & Analysis

 —Throughput, Communication needs, Programming effort

¥ Observations

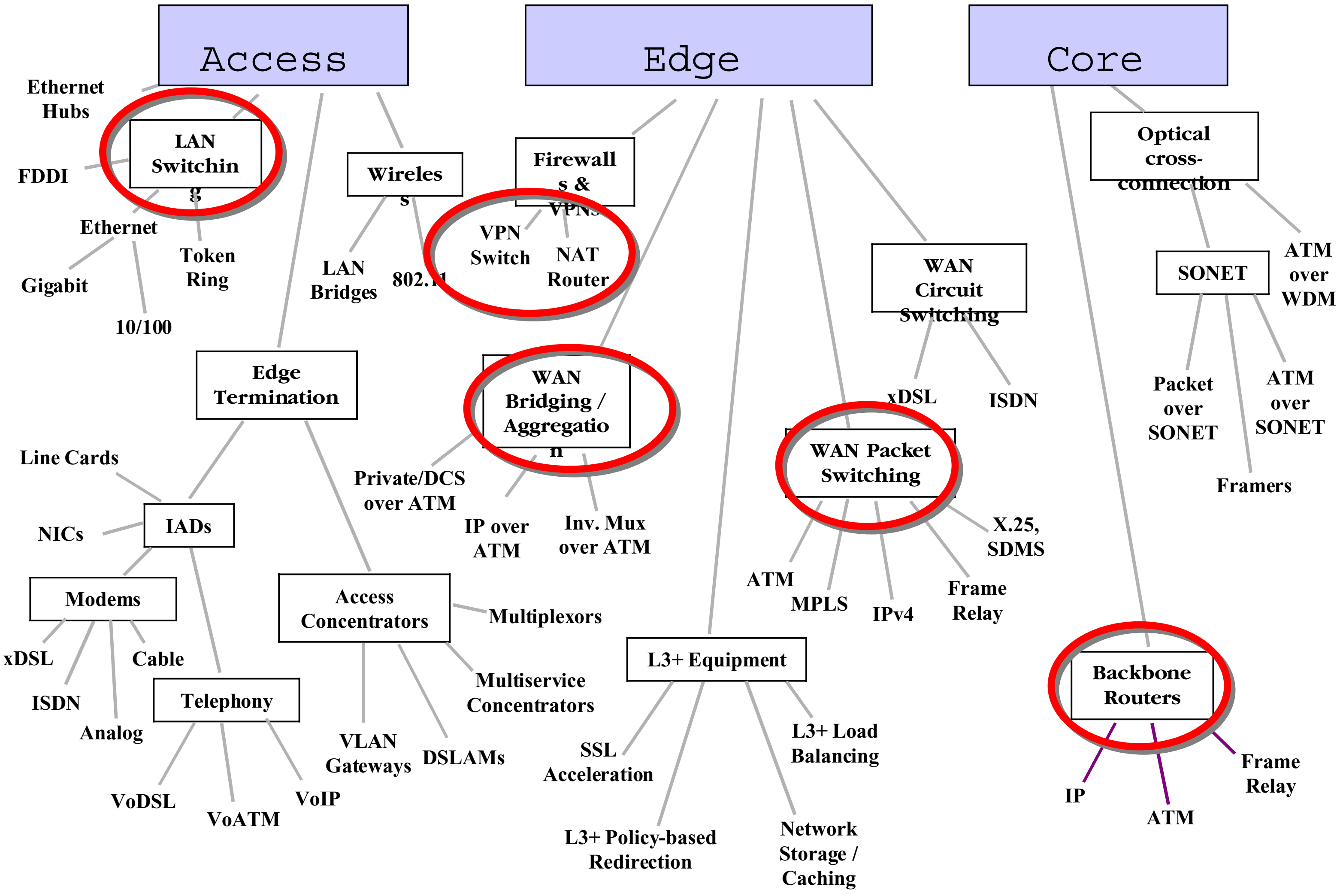
¥ Summary and conclusions

Applications

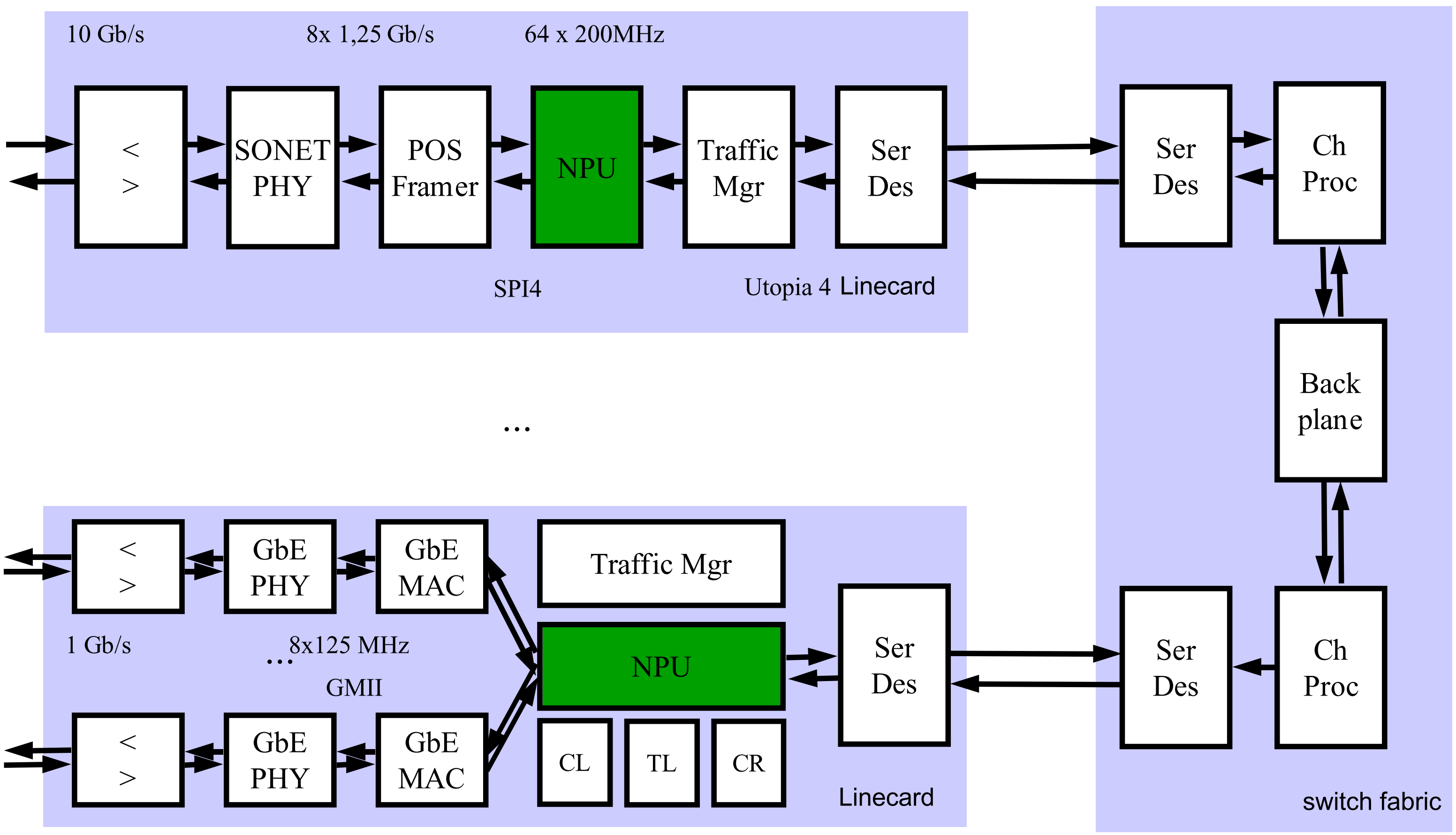


Equipment Centric View —

Router/Switches are most common systems



Router Building Blocks



Kernels for Packet Processing

¥ Pattern Matching and Feature Extraction

  — Find Expression and extract a related value from packet

¥ Lookup

  — Find path based on destination address and extracted features

¥ Computation

  — Checksum (CRC), encryption, fragmentation and reassembly

¥ Header Manipulation

  — TTL, Flags, add/replace tags and header fields

¥ Queue Management

  — Buffering, Storage and Scheduling of Packets

¥ Control Processing

  — Exceptions, table updates, statistics, NP state

Parallelism: Peak Performance and # of Cores

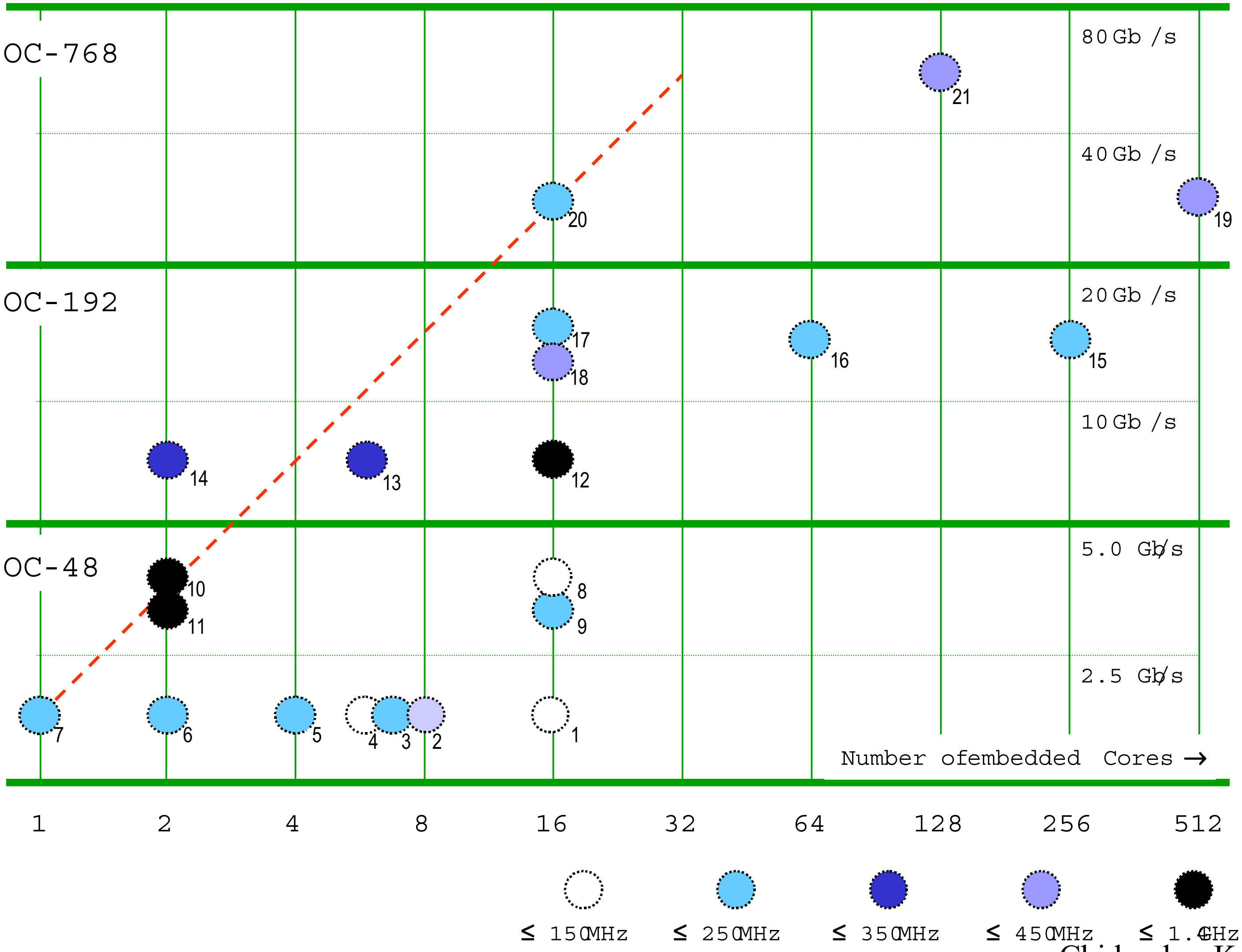
[21] EZchip NP-2
[20] Xel. PacketDev.
[19] Clearspeed

[18] Cognigine RCU/RSF
[17] Lexra Netvortex
[16] EZchip NP-1
[15] Clearspeed

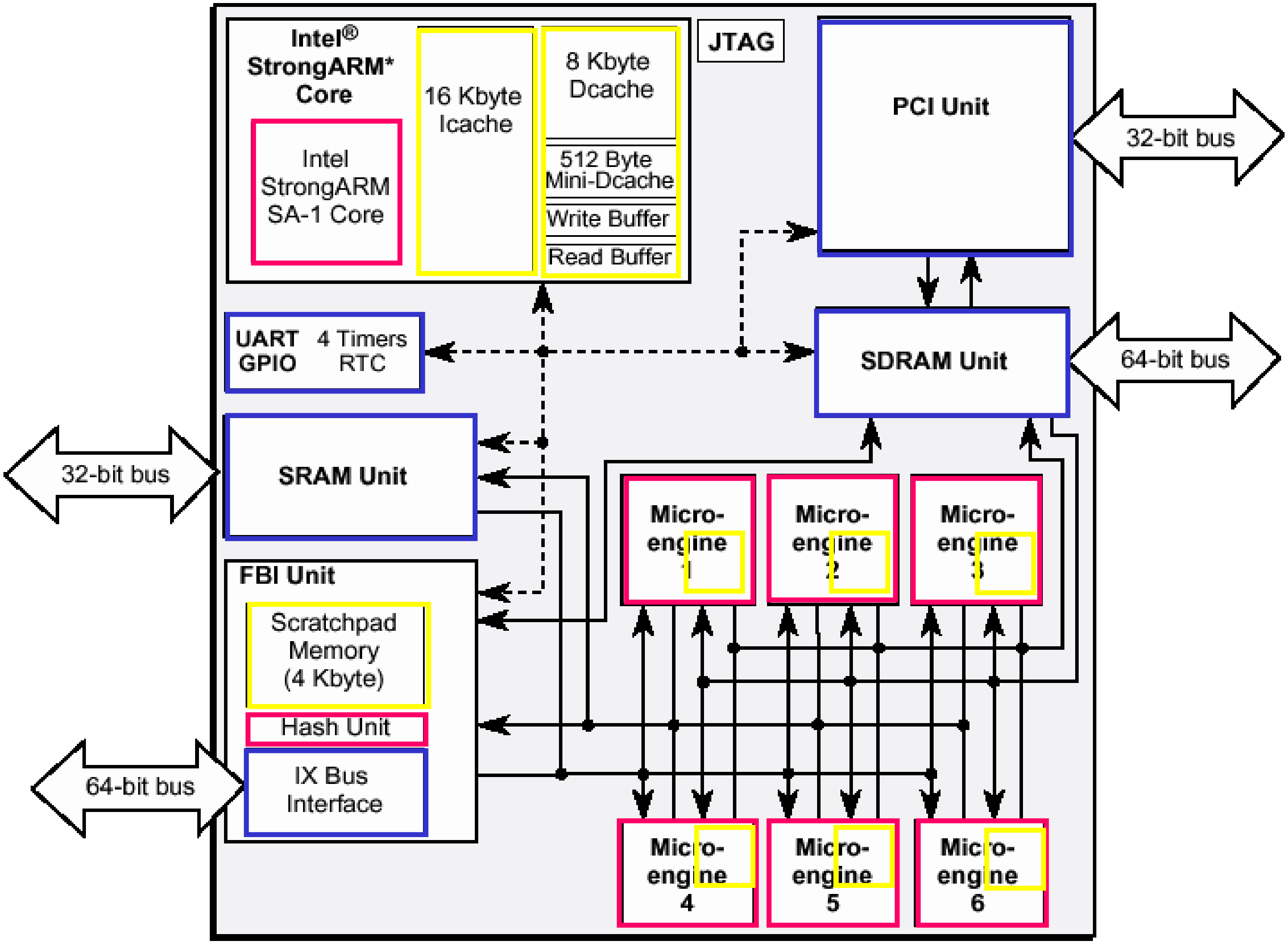
[14] Clearwater CNP810
[13] AMCC nP/7510
[12] Intel IXP2800

[11] PMC Rm9000x2
[10] Broadcom 12500
[9] IBM PowerNP
[8] Motorola C-5

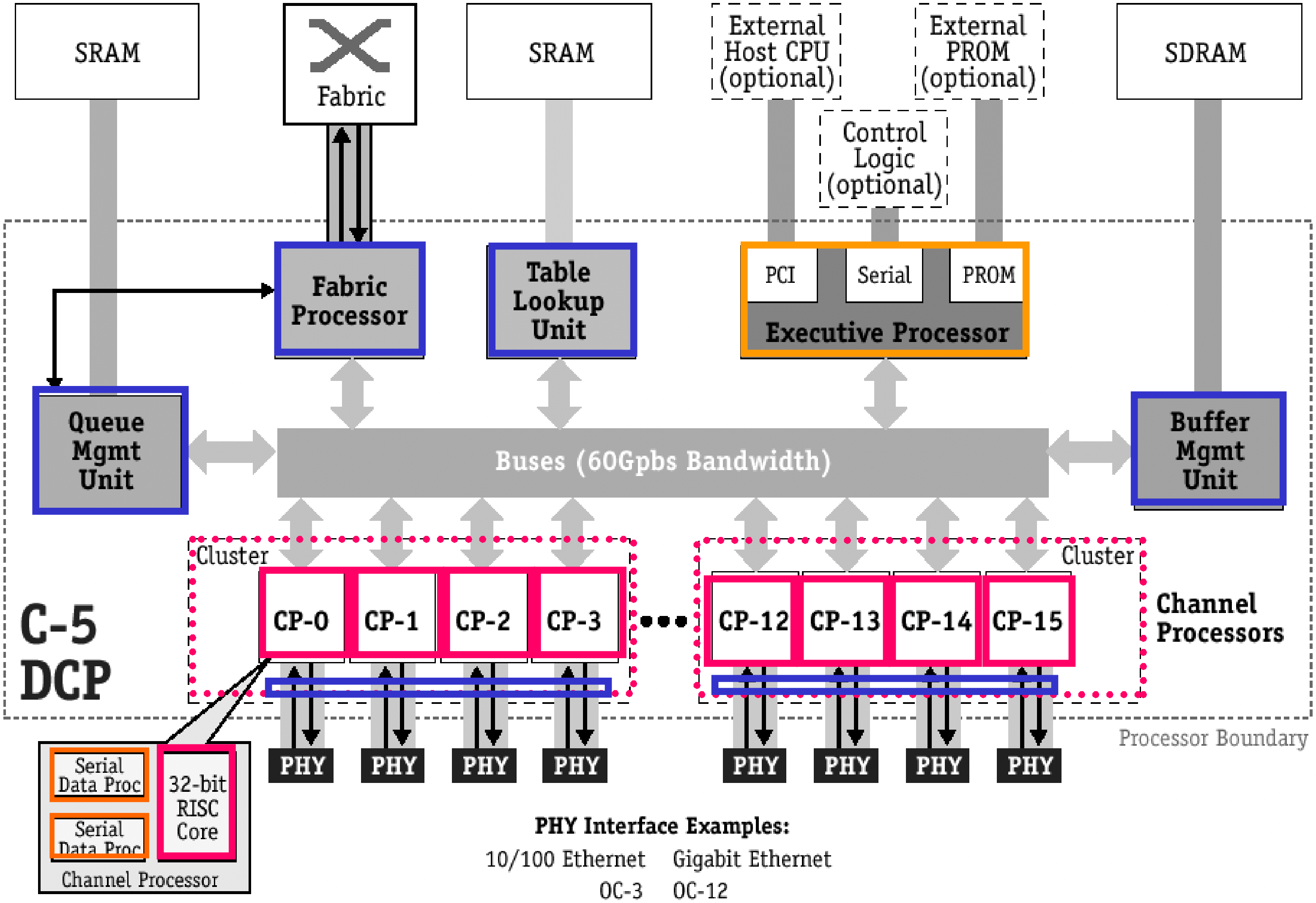
[7] Mindspeed CX27470
[6] MMC nP/7120
[5] Vitesse IQ2000
[4] Agere PayloadPlus
[3] Intel IXP1200
[2] Intel IXP 2400
[1] Cisco PXF



Intel IXP1200



Motorola C-Port C-5



PHY Interface Examples:

10/100 Ethernet Gigabit Ethernet

OC-3 OC-12

Challenge: Architecture Development/Evaluation

¥ Multi-dimensional concerns

— Number of parallel cores per chip, Type of core

— Number of instructions per core, Functionality per instruction

— Number and type of coprocessors, task distribution, etc.

— Memory hierarchy and on-chip communication

— Number and type of interfaces and peripherals

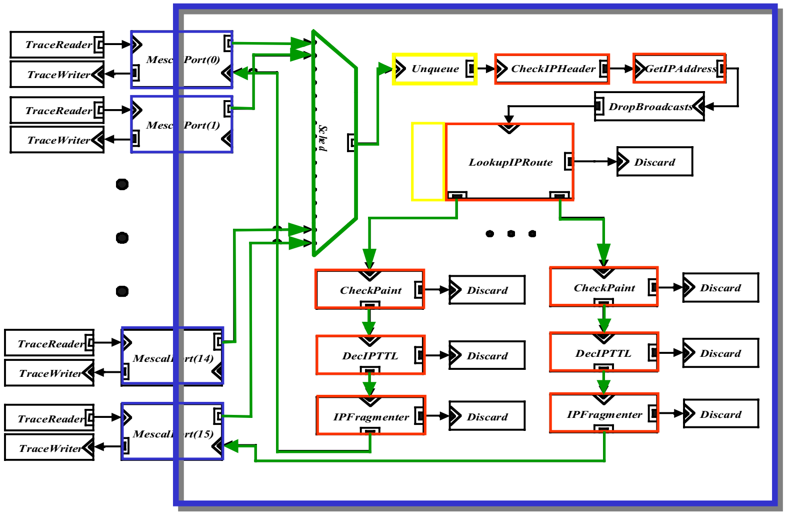
¥ Architectural Development Framework for specification, implementation, integration, and verification of heterogeneous concurrent processors

¥ *Which of existing architectures is optimal(?) and fits best ?*

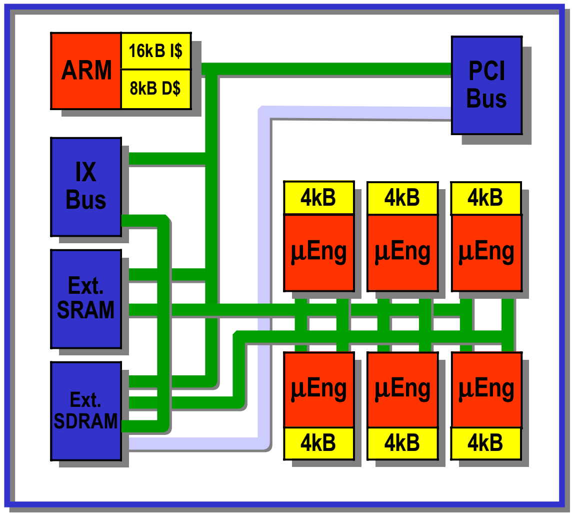
— Need to measure and compare architectures

— That s benchmarking !

Challenge: Architecture Deployment



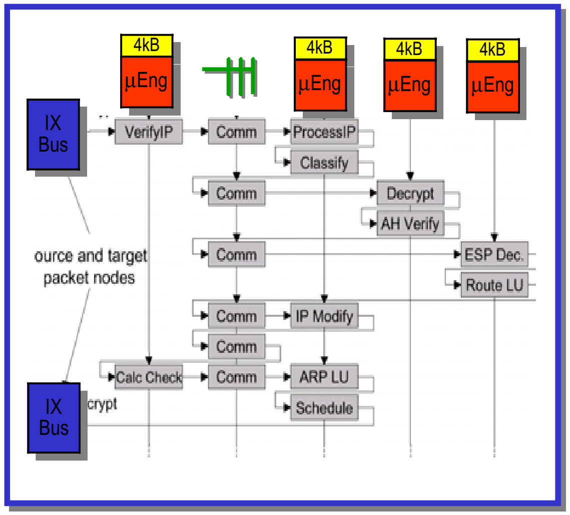
Click Router



IXP 1200

- ¥ ***Identify critical path (dataflow)***
  - — Header: Fifo$_{Rx}$- R$_{SRAM}$ — ALU — R$_{SRAM}$ — FIFO$_{Tx}$
  - — Payload: Fifo$_{Rx}$ — SDRAM — FIFO$_{Tx}$ (uE controlled)
- ¥ ***Interleave tasks w/ respect to communication***
  - — Assign uE/packet or piplined packet flow?
  - — Hide comm. latencies by hardware (!) multithreading
- ¥ ***Support NW data types***
  - — Exploit provided bit level Ops (ASM macros)
  - — Solve packet level data types (not supported)

Mapping & Scheduling



Outline of Talk

¥ Introduction to Network Processors

¥ IPv4 benchmark Implementation

—Intel IXP1200

—Motorola C-Port C-5

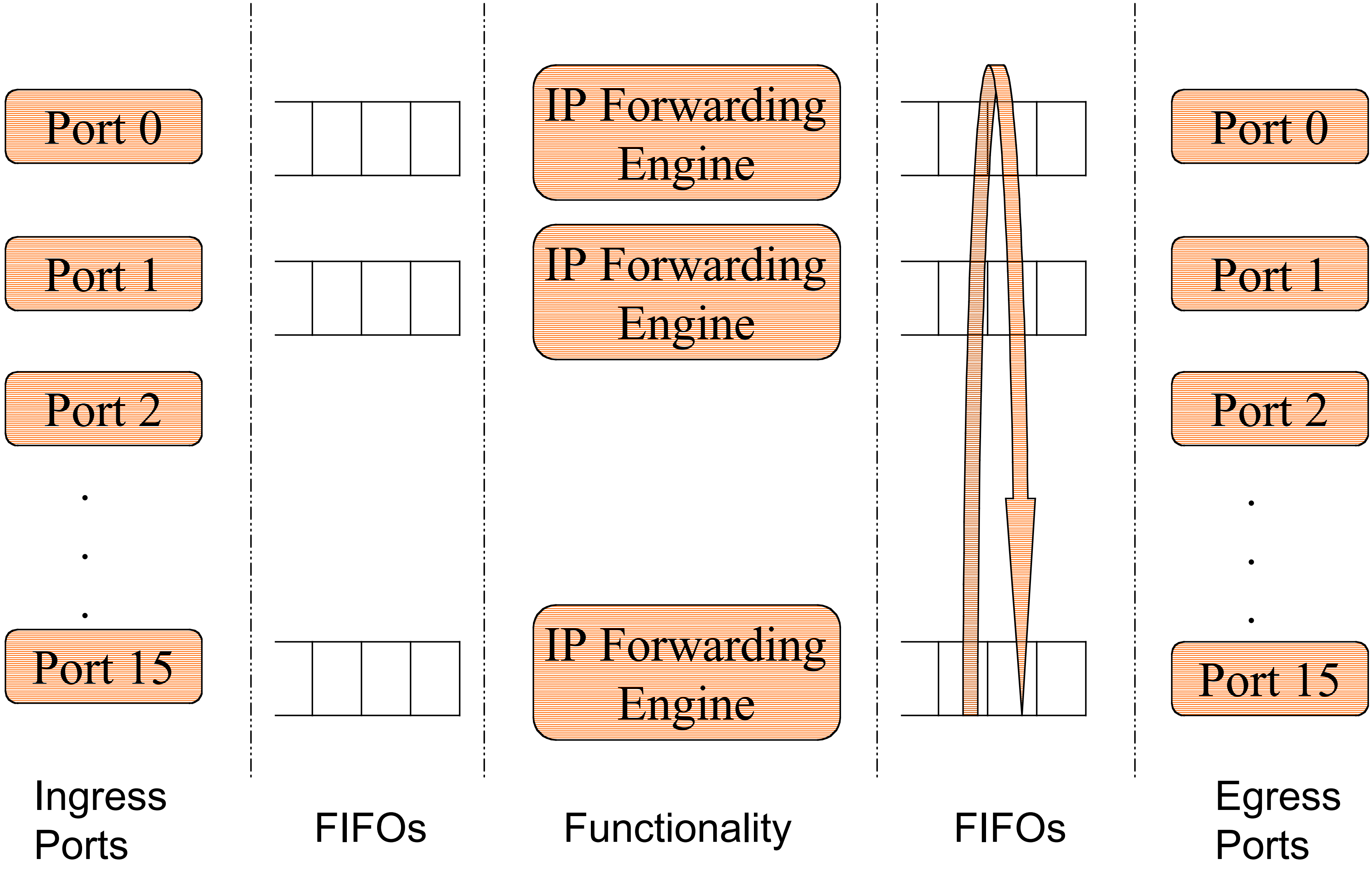
¥ Results & Analysis

—Throughput, Communication needs, Programming effort

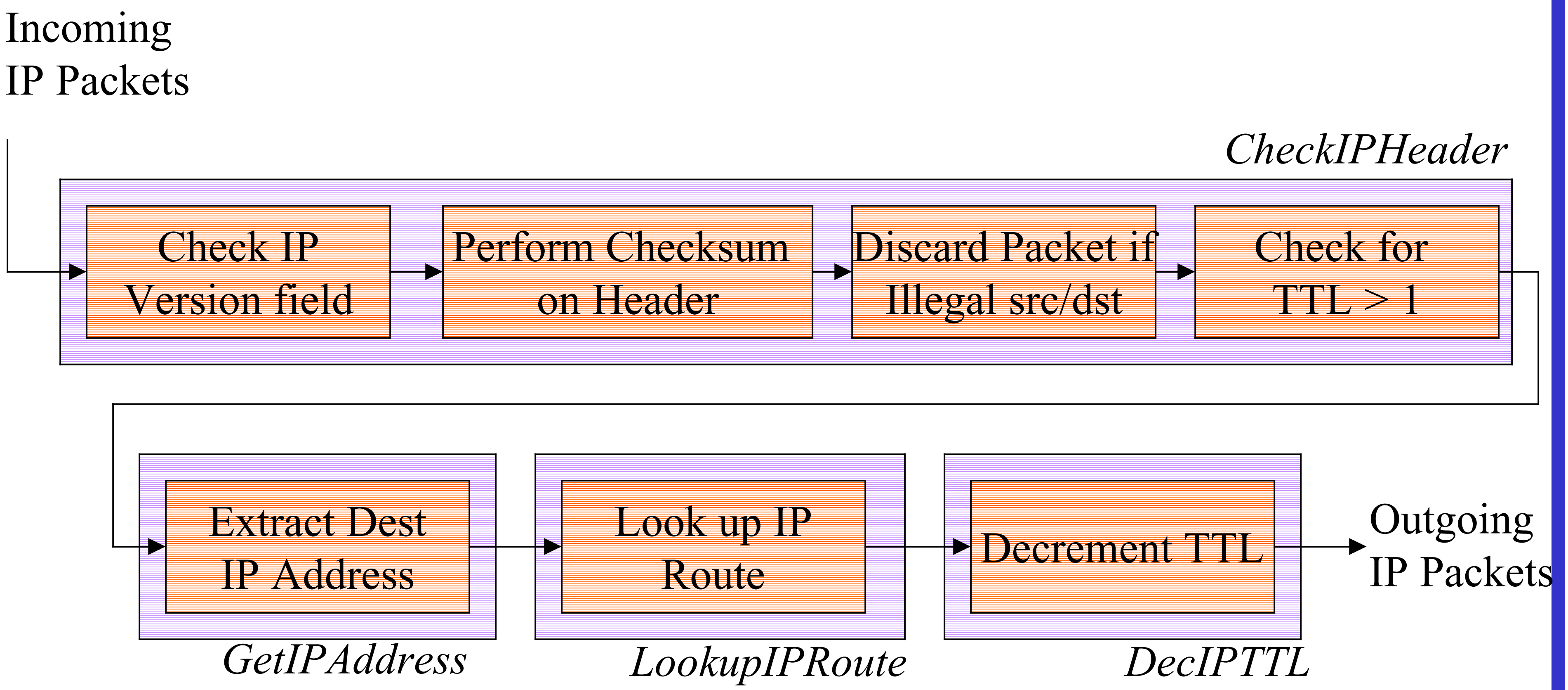
¥ Observations

¥ Summary and conclusions

IPv4 Forwarding Benchmark



Packet Forwarding Functionality



Incoming IP Packets

CheckIPHeader

Check IP Version field

Perform Checksum on Header

Discard Packet if Illegal src/dst

Check for TTL > 1

Extract Dest IP Address

GetIPAddress

Look up IP Route

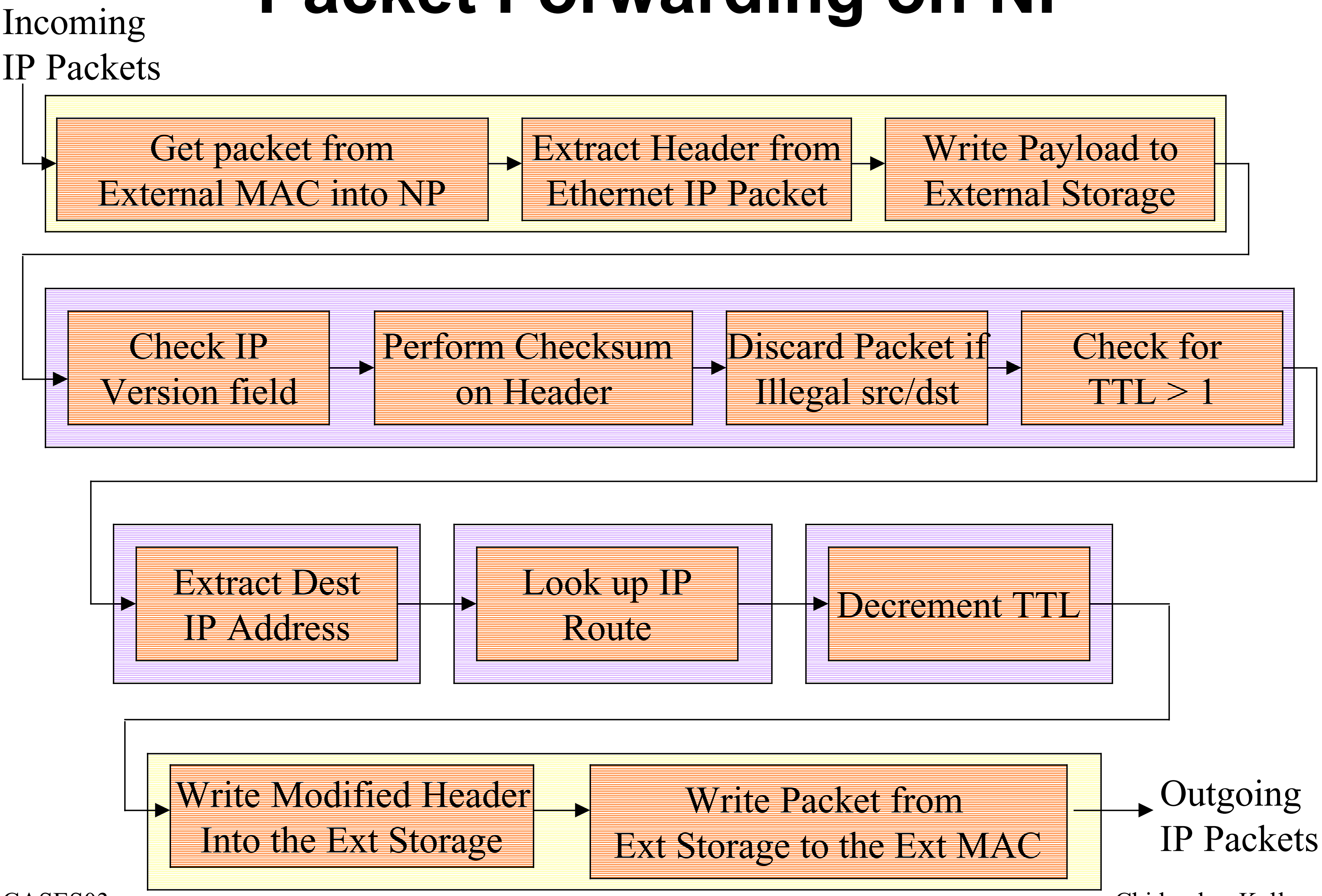
LookupIPRoute

Decrement TTL

DecIPTTL

Outgoing IP Packets

Packet Forwarding on NP



Incoming IP Packets

Get packet from External MAC into NP

Extract Header from Ethernet IP Packet

Write Payload to External Storage

Check IP Version field

Perform Checksum on Header

Discard Packet if Illegal src/dst

Check for TTL > 1

Extract Dest IP Address

Look up IP Route

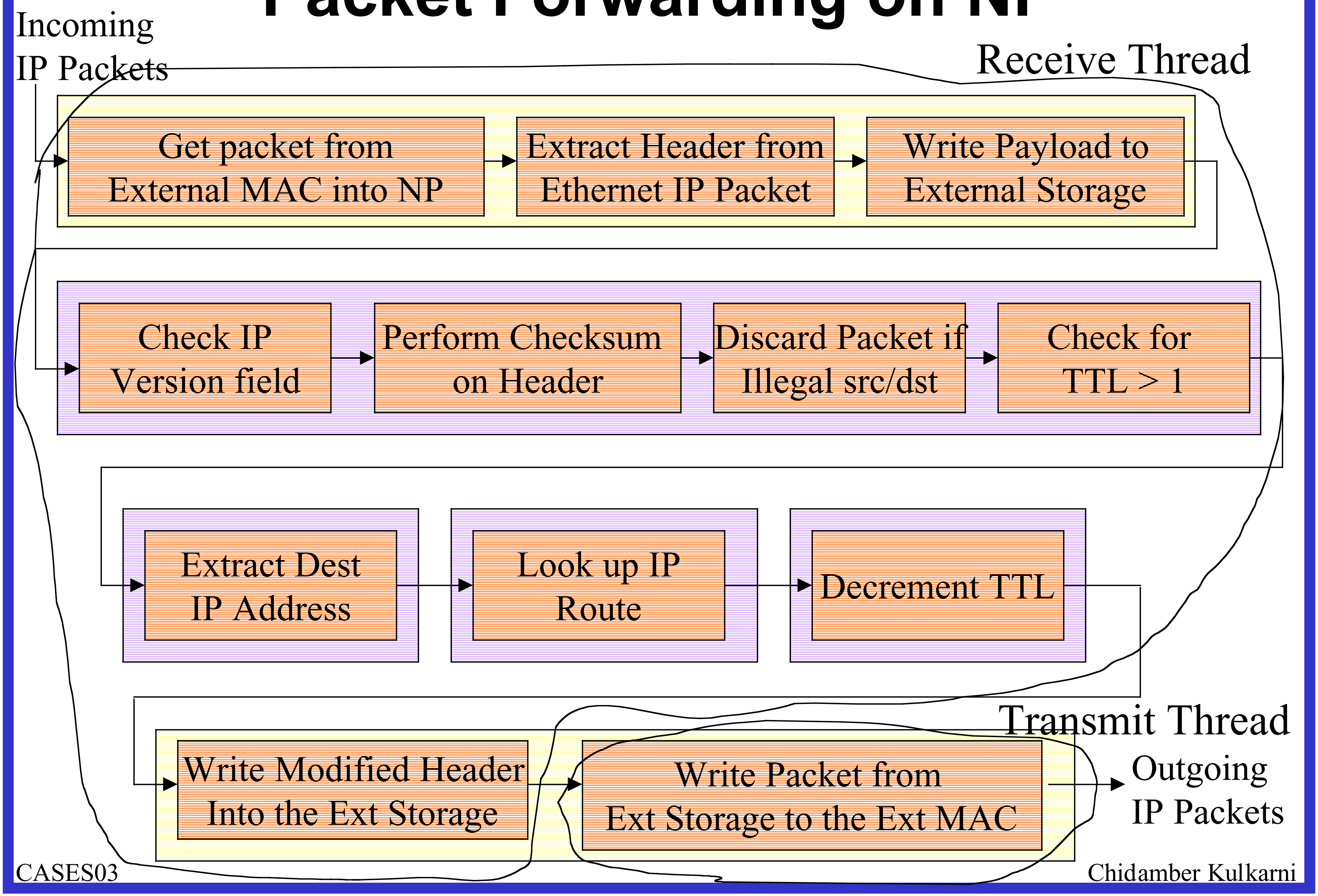
Decrement TTL

Write Modified Header Into the Ext Storage

Write Packet from Ext Storage to the Ext MAC

Outgoing IP Packets

Packet Forwarding on NP


Incoming IP Packets
Receive Thread
Get packet from External MAC into NP
Extract Header from Ethernet IP Packet
Write Payload to External Storage
Check IP Version field
Perform Checksum on Header
Discard Packet if Illegal src/dst
Check for TTL > 1
Extract Dest IP Address
Look up IP Route
Decrement TTL
Transmit Thread
Write Modified Header Into the Ext Storage
Write Packet from Ext Storage to the Ext MAC
Outgoing IP Packets

Programming IXP1200

¥ Micro-engine programming (similar to programming a RISC core without a cache)

¥ Interfaces (memories and external MAC unit)

  —Micro-engine to SDRAM, Micro-engine to SRAM, Micro-engine to Scratchpad, Micro-engine to IX bus unit

  —IX bus unit to SDRAM

  —Additional attention for non 64-byte aligned (or multiples) packets

¥ Impact of input packet sizes

¥ Scheduling freedom

¥ Partitioning

Challenges in Programming IXP1200

¥ Architecture related:

—Difficult to determine the right partitioning over different threads and micro-engines

—Programming overhead for non 64-byte multiple packets

—IX bus unit - a bottleneck for higher throughput

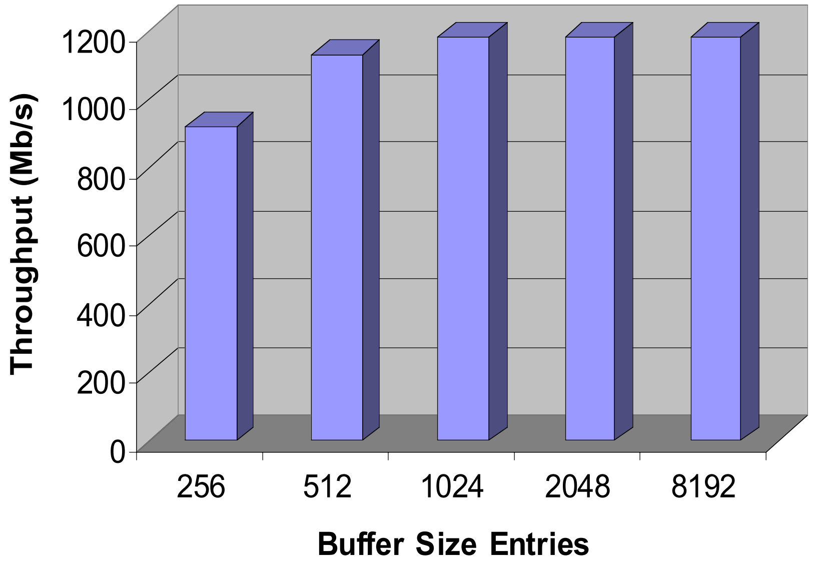
¥ Managing transmit state machine

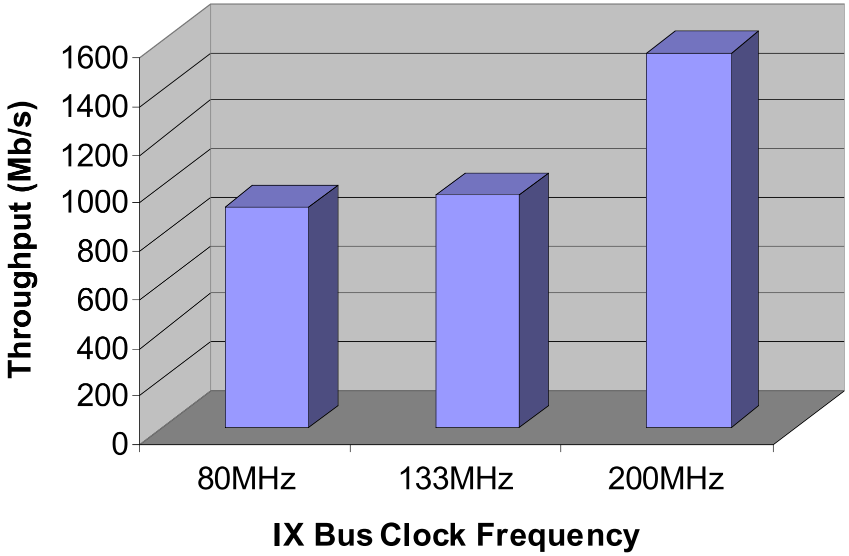
¥ Queue management issues for internet traffic mix

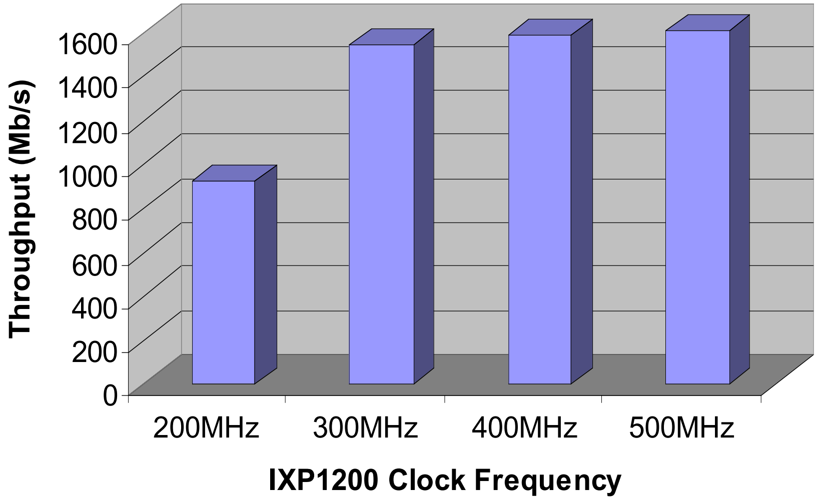
¥ Software environment related:

—IXP1200 library of elements has more layers (simpler hardware); potentially more development effort?

Influence of Interfaces on Throughput?







64 byte packets;
simple route table

Architectural Bottlenecks in IXP1200?

¥ Performance limited (probably) by

—external MAC buffer size

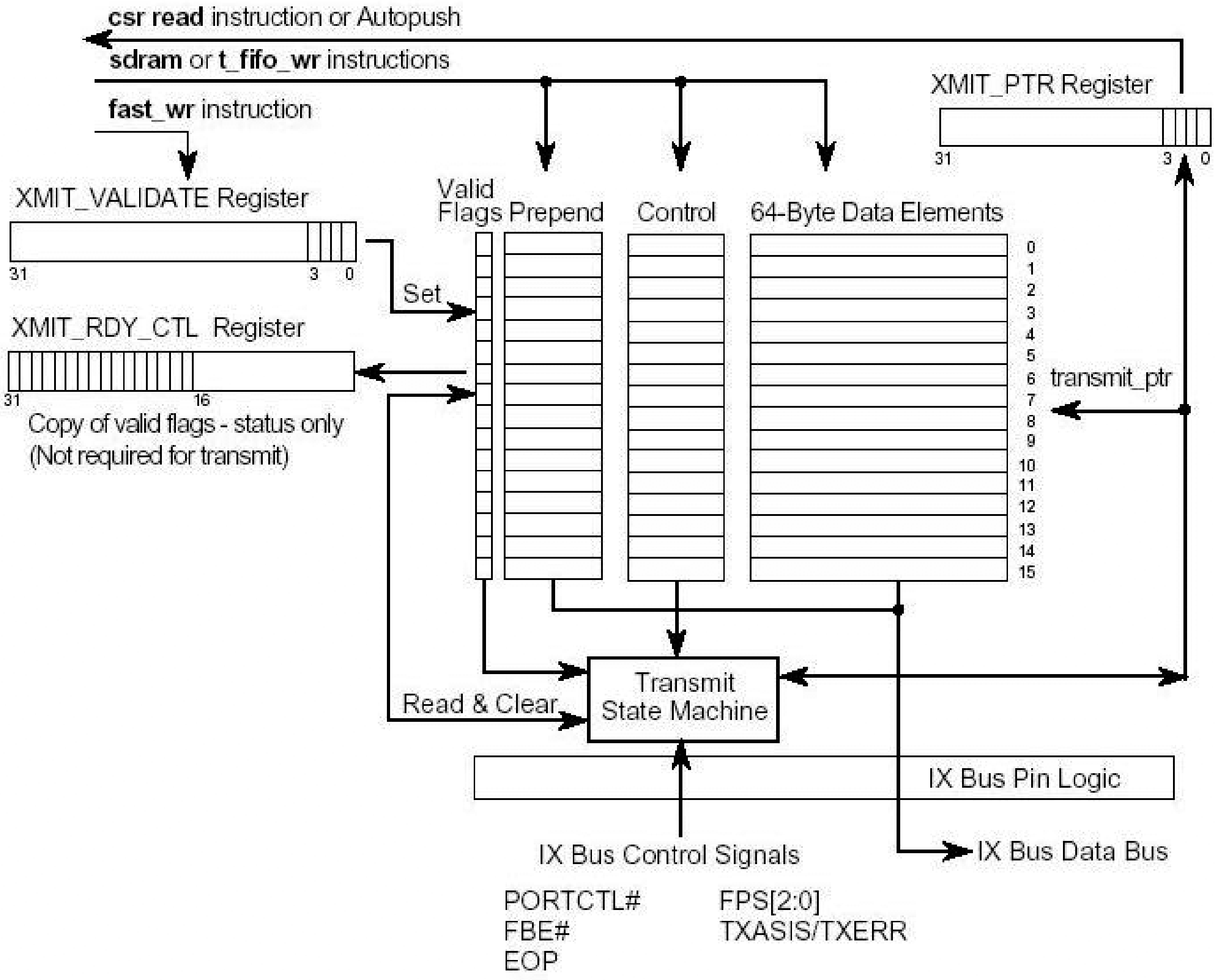
—the IX bus connecting the external MAC to the IXP1200

—IXP1200 clock frequency

—In fact - none of the above, we found it to be a

¥ Trade-off between a dynamic assignment of transmit FIFO and a static (fixed) assignment

IXP1200 Transmit State Machine



Programming C-Port C-5

¥ Channel Processor programming (similar to programming a RISC core with an SRAM)

¥ Interfaces (memories and SDP)

— CP to Queue management unit (SRAM), CP to table lookup unit (SRAM), CP to buffer management unit (SDRAM)

— CP to serial data processors (setting bits in control registers)

¥ Scheduling freedom

— Performance estimation is better due to upper/lower bounds on off-chip resource access times

¥ Partitioning

Challenges in Programming C-5

¥ Debugging micro-code for serial data processors (SDPs) can be painful (need for good libraries)

¥ Software environment needs more maturity — debugging concurrent code through gdb is difficult

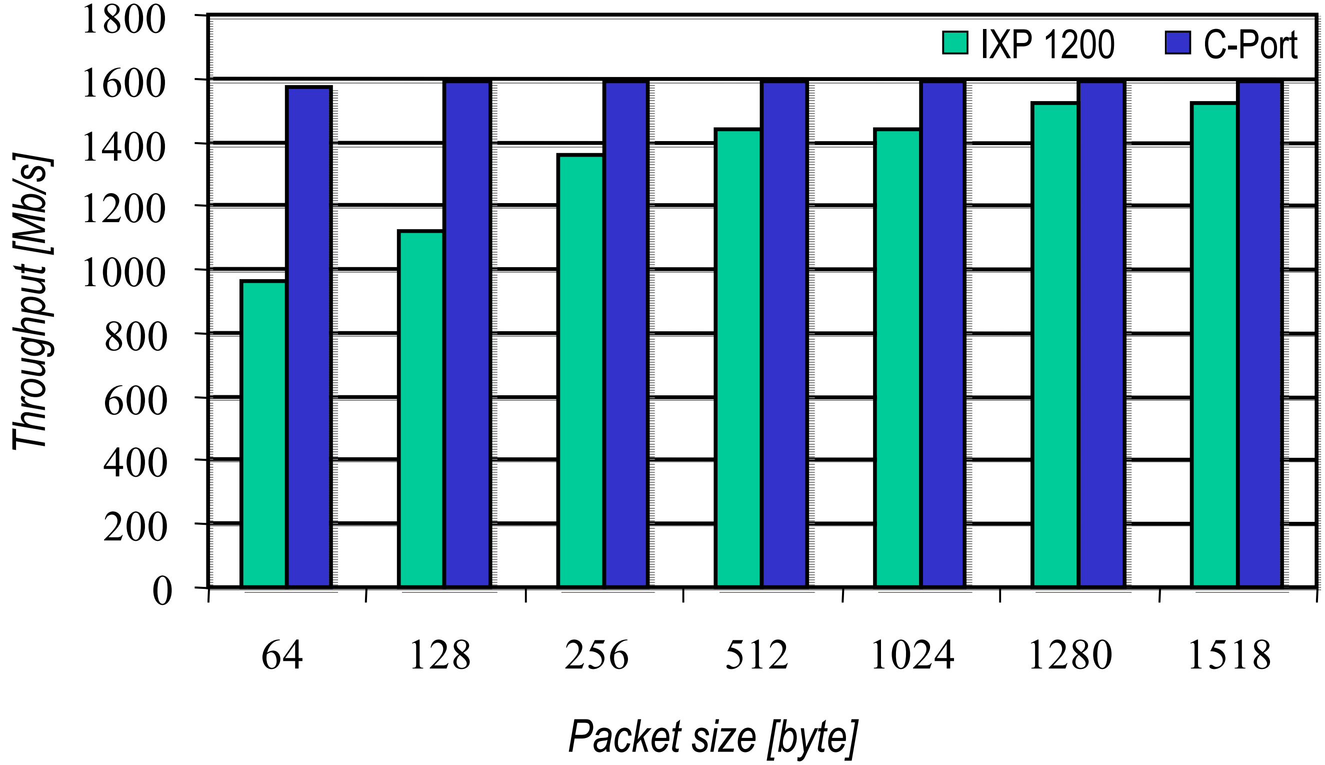
¥ API is vast — still needs some supprt for programming choices

¥ Unclear yet how to tune performance for a complex packet mix

Outline of Talk

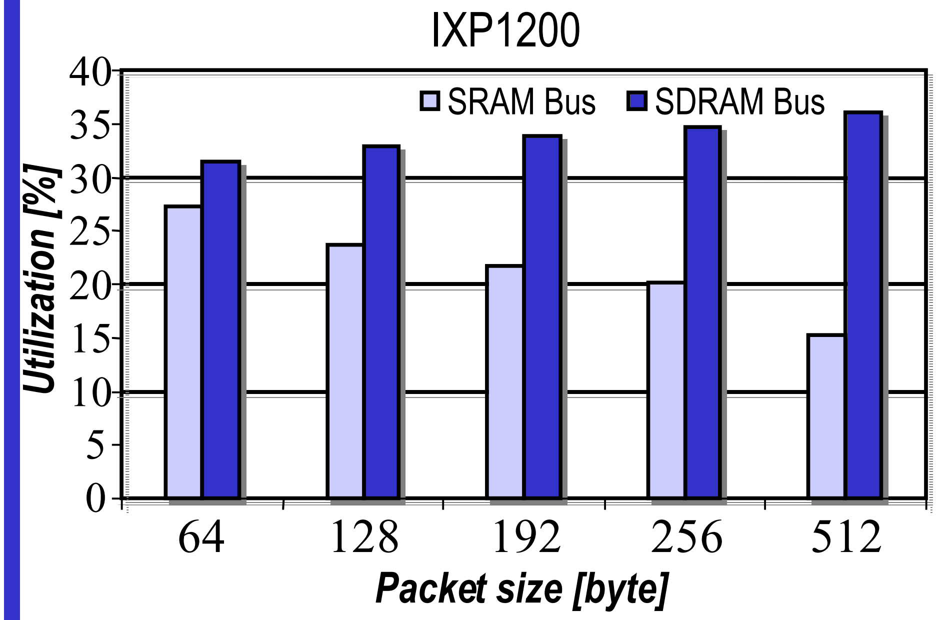
- ¥ Introduction to Network Processors
- ¥ IPv4 benchmark Implementation
  - —Intel IXP1200
  - —Motorola C-Port C-5
- ¥ Results & Analysis
  - —Throughput, Communication needs, Programming effort
- ¥ Observations
- ¥ Summary and conclusions

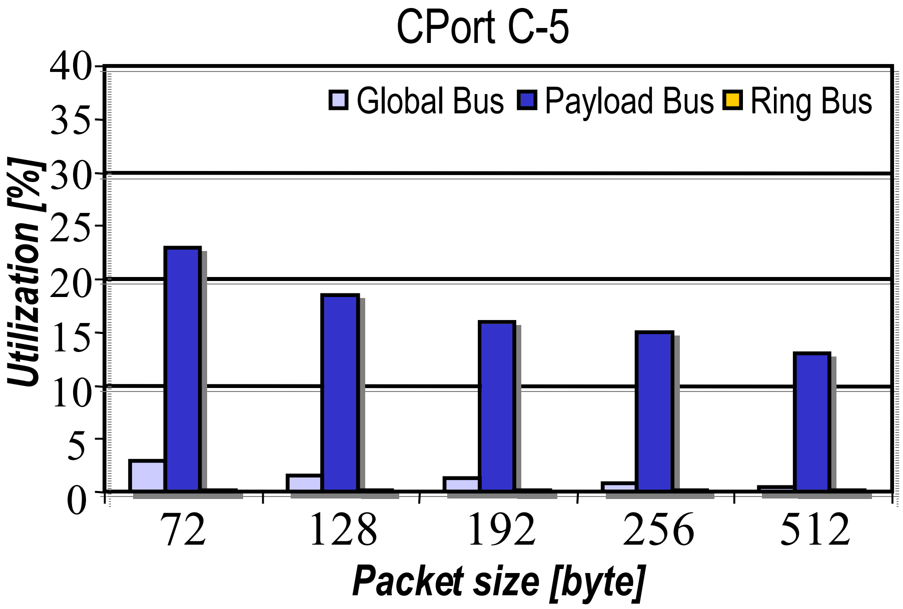
Throughput for IXP and C-5



C-5 achieves a higher throughput for all packet sizes and is less sensitive to packet sizes compared to IXP

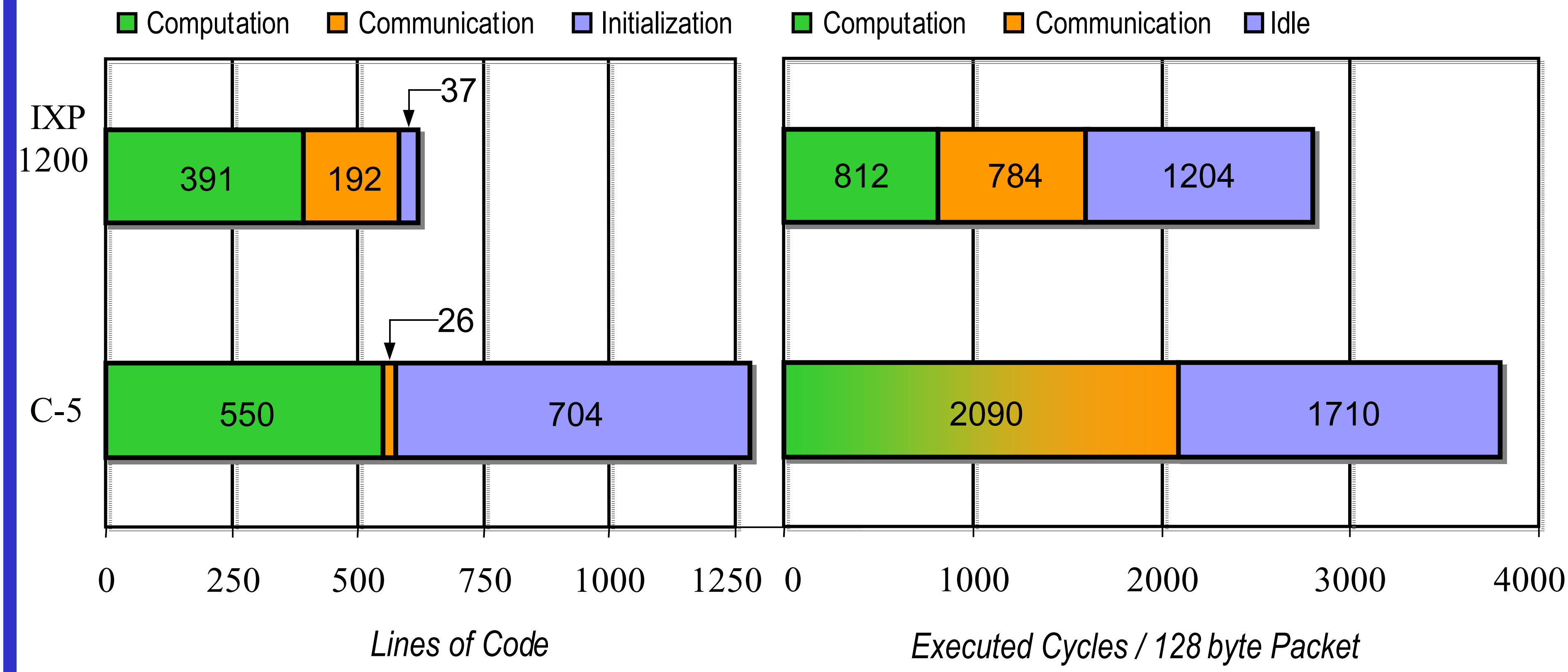
Bus load for IXP and C-5





¥ SDRAM bus in IXP has a similar utilization as the Payload bus in C-5
¥ SRAM bus in IXP however, has a much higher utilization than combination of Global and Ring bus in C-5

Programming Effort



Similar programming effort but dissimilar achieved throughputs

Comparing IXP vs C-5 Programming

¥ Functional correctness

—IXP1200 requires a larger programming effort compared to C-5

—Main reasons for the difference are:

¥ Vast API of C-5 aided by

—Specialization of interfaces

—Configurable MAC's

¥ Performance tuning

—Not yet clear since C-5 is over-powered for our benchmark as compared to IXP1200

—Mostly related to the (micro-) architectural details (nuances?) for IXP1200

¥ In summary, if possible arbitration and scheduling should be made deterministic to help programming

Outline of Talk

¥ Introduction to Network Processors

¥ IPv4 benchmark Implementation

—Intel IXP1200

—Motorola C-Port C-5

¥ Results & Analysis

—Throughput, Communication needs, Programming effort

¥ Observations

¥ Summary and conclusions

Observations on Programming Model

¥ Programming network processors

—partitioning application to (multiple) PEs/Threads

—Scheduling tasks according to packet flow

—Multi-PE and multi-resource (storage) communication

—Little support for integrated decisions for the above issues as of now

¥ Programmability

—partitioning, scheduling, cost of communication, scalability, performance determination

—simple and predictable architecture

In Summary

¥ Diverse architectures for similar problems

— For IPv4 forwarding we have a better idea

— What happens if we add all couple of more applications?

¥ Any successful adoption and deployment of NPUs depends on ease of programming

— Enable an integrated development environment that supports design space exploration and decisions related to the three key aspects

— Potential ways to achieve this

¥ Complex architectural trade-offs need to be made to enable a simple and useful programming model

¥ How about building an architecture for a given programming model (derived from application model)?