FORWARDING PLANE REALITIES

FOR PROTOCOL DESIGNERS

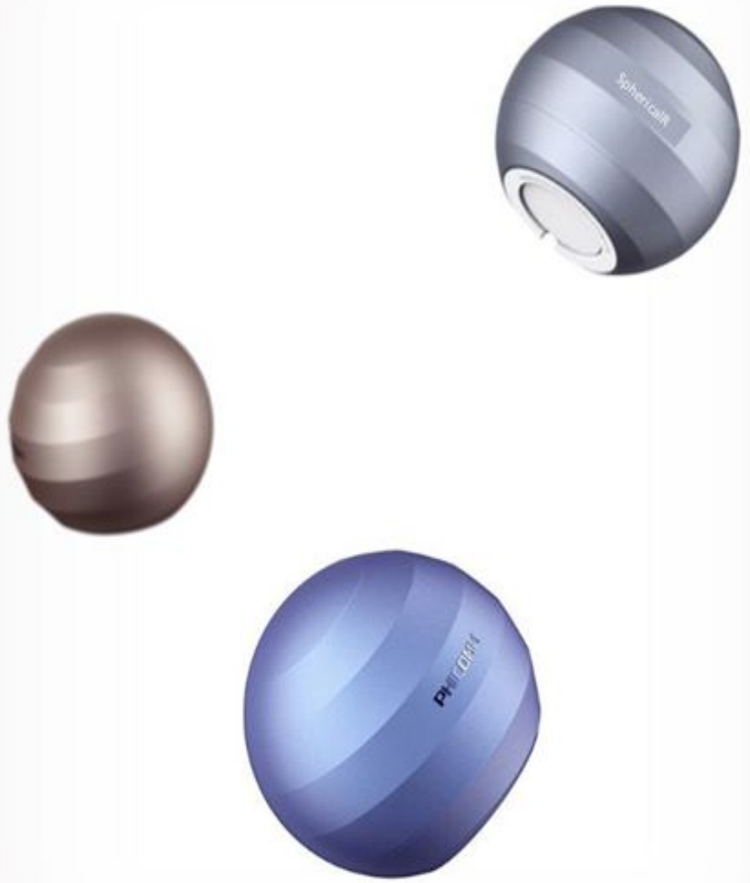
John Scudder, Juniper Networks

Frank Brockners, Cisco

Toerless Eckert, Huawei USA

IETF 104, Prague, Wednesday, March 27, 2019

Brian Petersen of Juniper Networks authored an earlier deck this one was based on.



Spherical Routers

do whatever protocols want them to do;
see also “spherical cows”

AGENDA

- Router Taxonomy
- Pipeline Characteristics
- Considerations for Protocol Designers

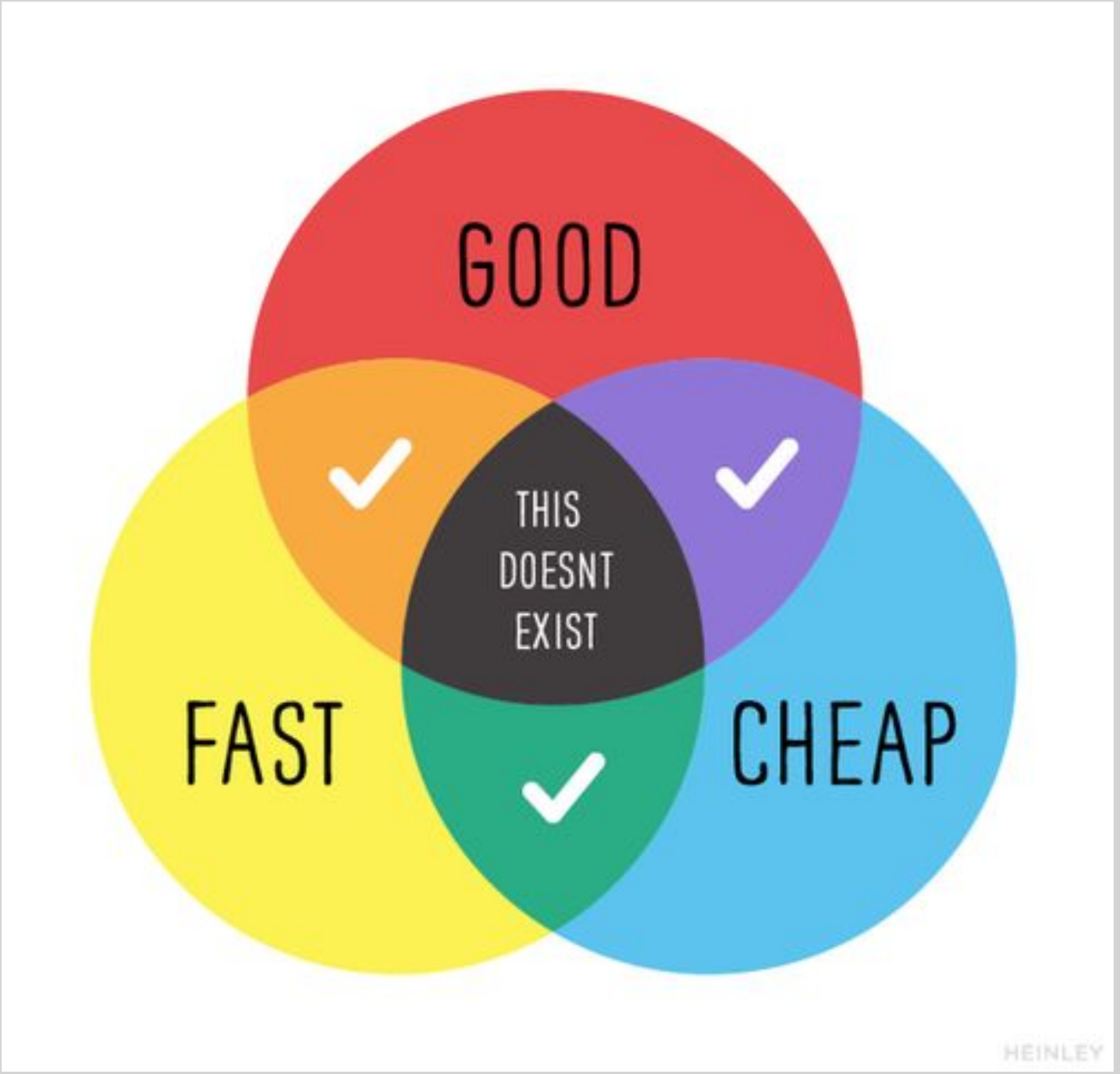


Image from InReach Solutions

ROUTER TAXONOMY

The Landscape

Metrics:

Gbps/$ - include $/RU (rack unit), $/KW (power source, cooling), power for bits (radio)

Where/how does the Control Plane / IETF factor in ?

Multi-stage programmable forwarding

CPU-forwarding

Distribution/Core
Fabric/Multi-chassis

Data Center

Campus/Industrial Broadband
L3/L2 *switching*

Low-end/ IoT

High-End Edge/SD-WAN/NFV
Multi-core forward
ODP/OFP/FD.IO/OVS/..

Open consumption / P4

Advanced/Programmable QoS/TM ??

Telemetry ?

CPU centric IETF protocol design

Cheap/high-speed Inflexible ASIC

Future:
Chip combining various elements of control and forwarding
Cores, prog. pipelines, NPU, Inference Accelerators, DSP

IEEE & Friends

Disclaimers

- Just one common type of router
- Non-distributed example
  - Distributed adds more challenges

Good reference to start design protocols for

If all the components are designed well

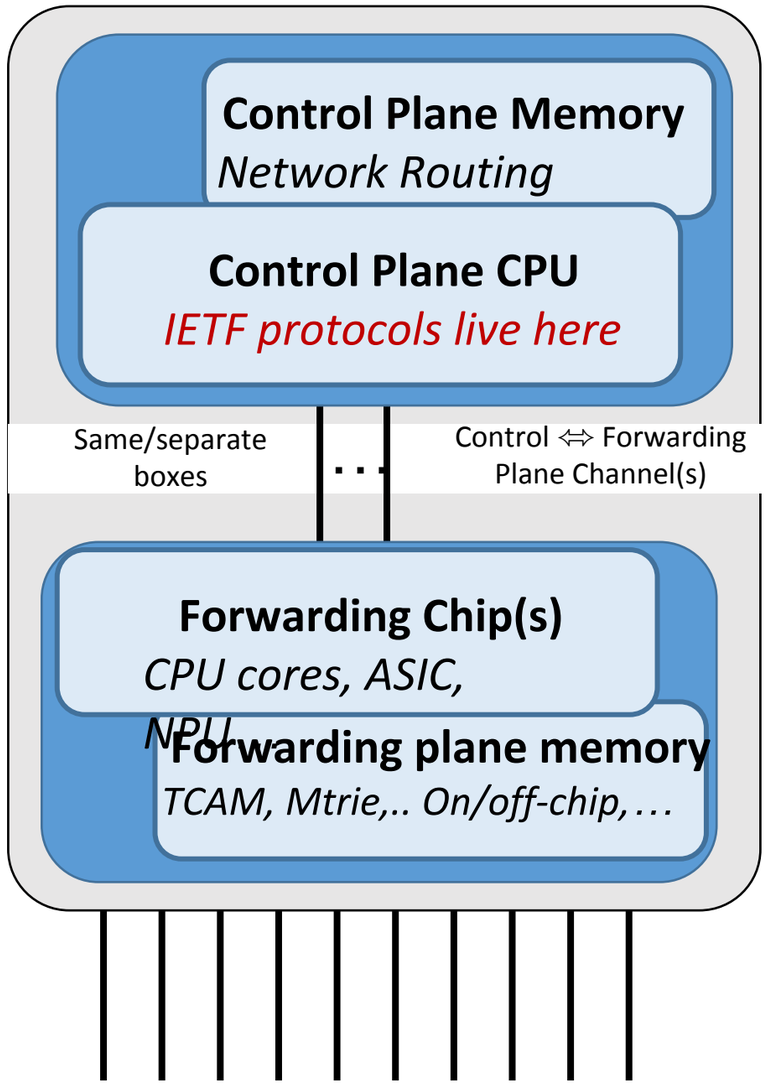
When did a customer last bought a router and asked for control plane performance ???

Highly differs by market segment

For the unpredicted future feature ?

RFP & deploy/forget or future proofing ?

SW vs. HW thinking



e.g: 100 * 100 Gbps external interfaces / neighbors

Look! 4 CPU cores (all slow)

You sell memory by the MEGAbyte ?

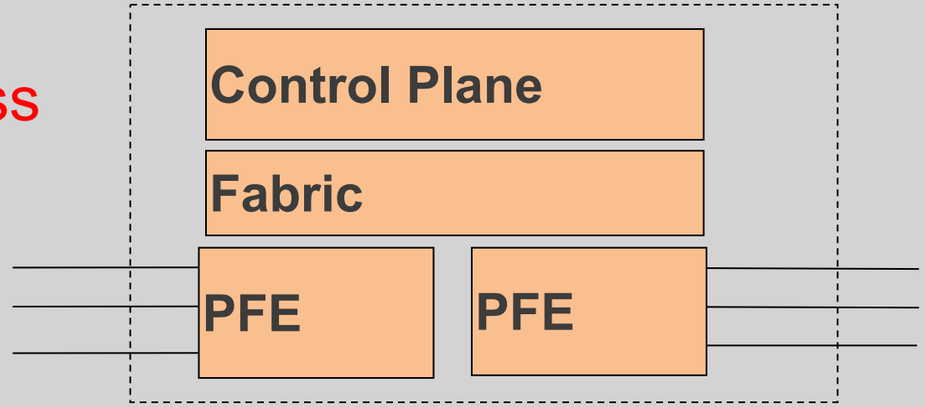
Forward for difficult packets ??? (punt)

Hmm.. 1 Gbps / 100 neighbor IGP updates

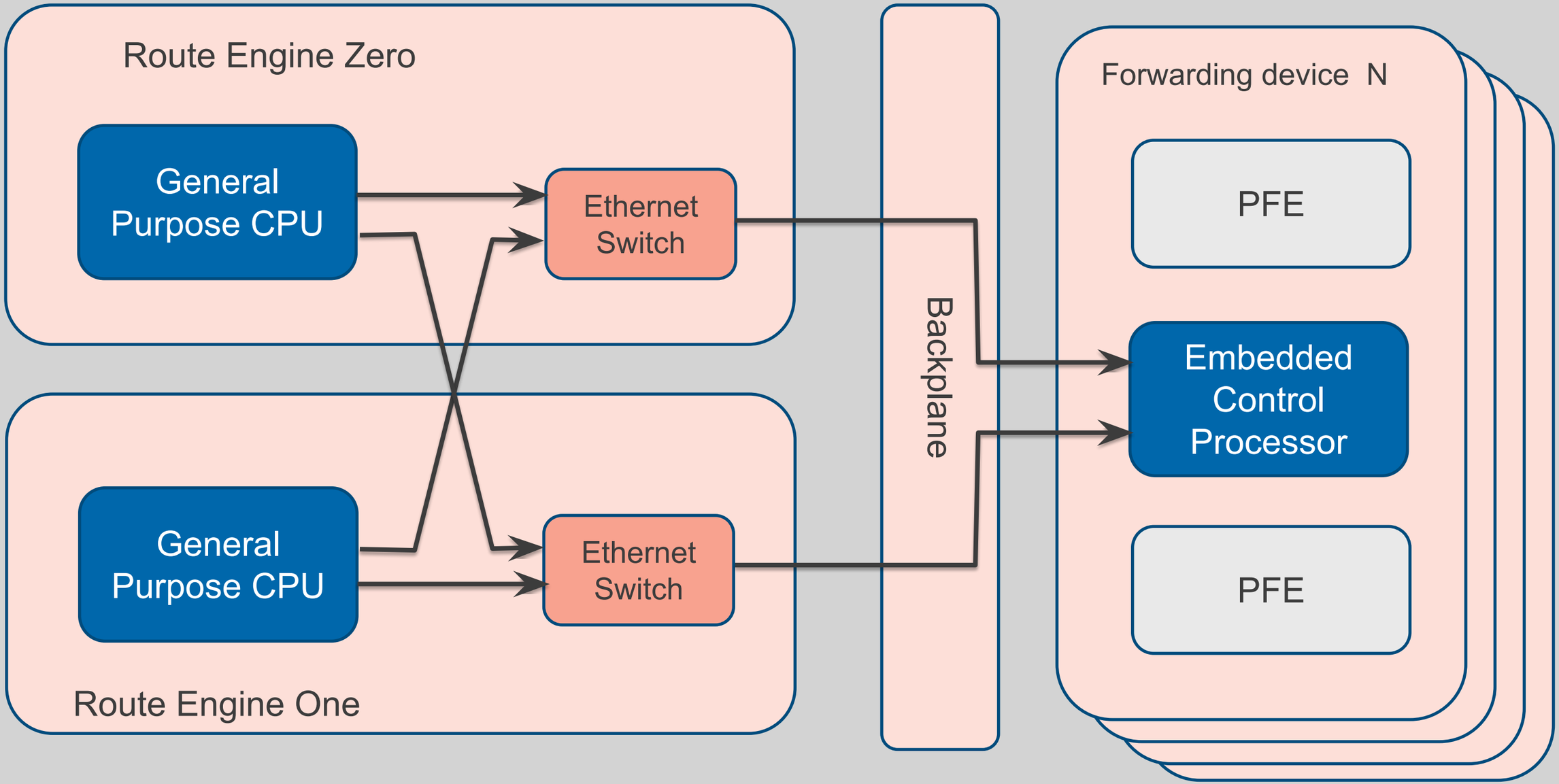
Fast but inflexible
Short TCAM length/depth
Fixed processing pipeline
…

KEY ELEMENTS

- High-scale routers comprise several key elements:
- Control Plane
  - Responsible for managing routing tables, authenticating subscribers, configuring interfaces
- Packet Forwarding Engine(s) (PFE)
  - Responsible for forwarding each packet (address lookup, queues, access lists, etc)
  - Flexibility varies greatly (x86.. to ..hyper-optimized)
- Fabric
  - Responsible for moving packets from one forwarding device (e.g. line-card, NPU, ..) to another

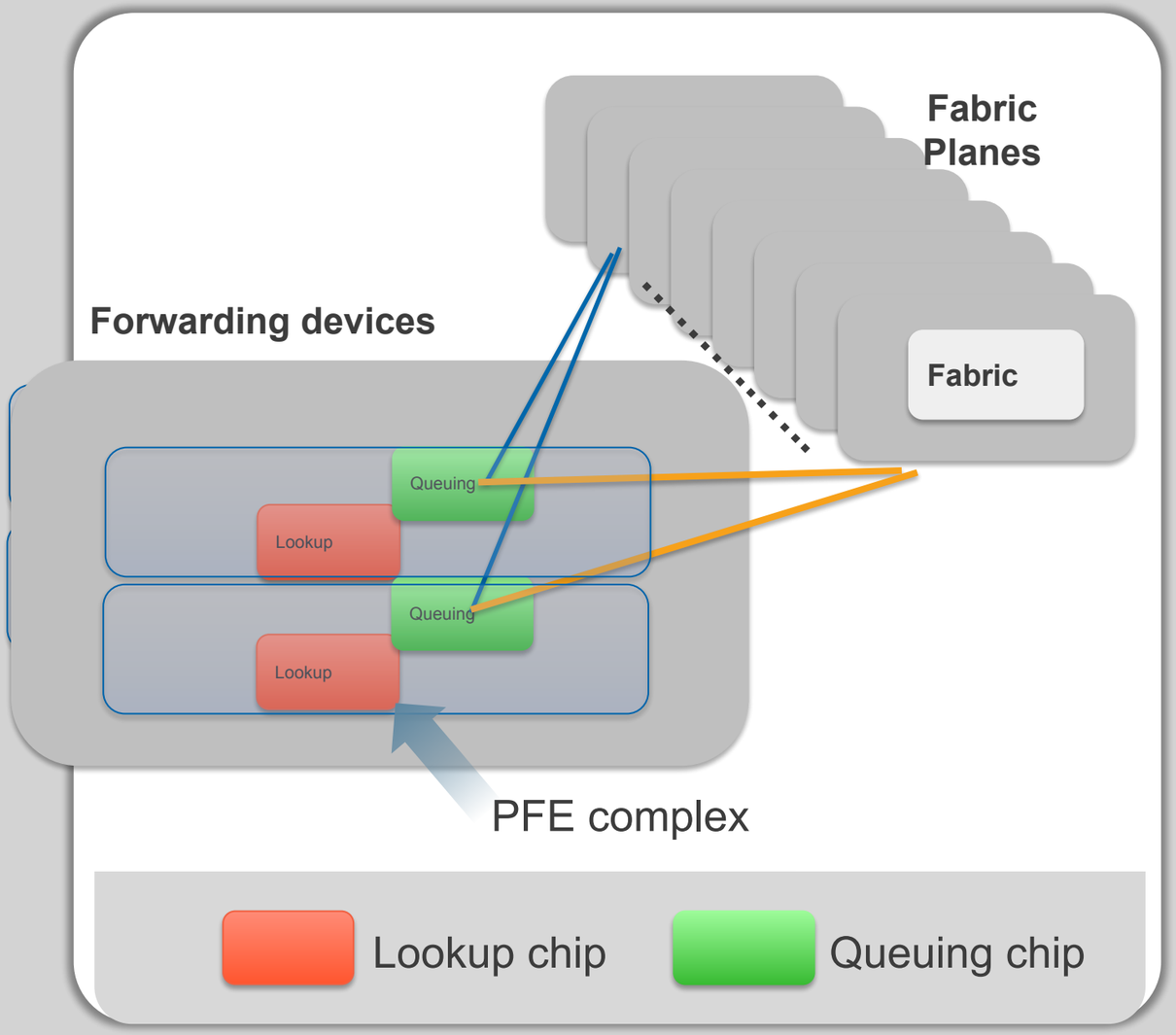


ROUTER CONTROL PLANE



FABRIC-BASED ARCHITECTURE

- Most high-scale routers are fabric-based
  - Multiple line cards, each containing PFEs
  - A chassis-wide interconnect fabric transfers traffic from ingress to egress devices/line cards

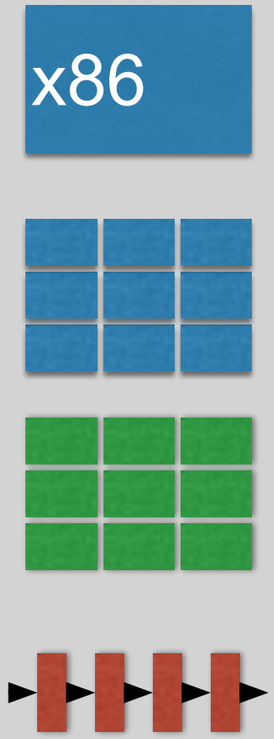


PACKET FORWARDING ENGINE (PFE)

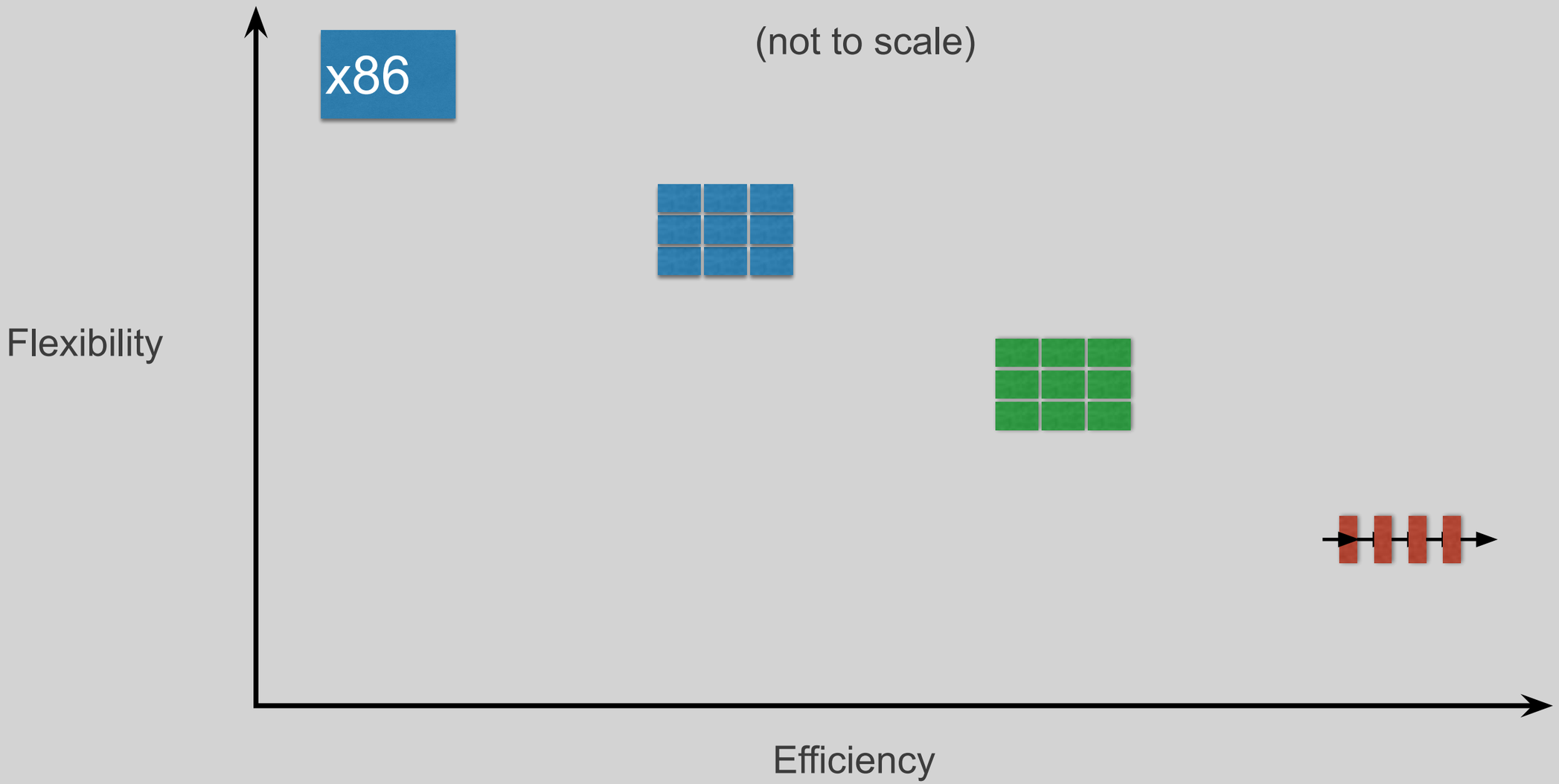
- PFEs do the work to move packets from ingress to egress
- Key functions:
  - L2 & L3 analysis & features
    - Figure out whose packet it is, what should happen to it, where it should go.
  - Packet buffering
    - Store the packet in buffer memory until there's room to transmit it
  - Queuing & scheduling
    - Decide which packets should go in what order to achieve fairness and real-time delivery guarantees.
- PFEs may be micro-programmable, table-driven or hard-coded
  - It's the old cost/performance/flexibility trade-off matrix…
- PFEs may be totally integrated (the features, buffering, and scheduling may all be on a single chip) or they may be separated into different physical devices

SOME ROUTER ARCHITECTURE TYPES

- General-Purpose Processor
- Sea of General-Purpose Processors
- Sea of Special-Purpose Processors
- Flexible Multi-Stage / Pipeline



TRADEOFFS



TRADE-OFFs

HIGH-SCALE ROUTERS VS. GENERAL-PURPOSE COMPUTERS

- Traditional computer architectures (e.g., x86) are “infinitely” flexible
  - … at a cost
- High-performance routers trade flexibility for other important attributes
  - Example tradeoff: Access to packet Data
    - General-Purpose Processors are presented with a buffer containing an entire packet
    - Pipeline (et al) are presented with the first *n* bytes of a packet
- The trick is to only trade away flexibility you didn’t need anyway
  - But predicting the future is hard (“wait, you want to look *how* deep?”)
  - This is where protocol designers can help

WHY SPECIFIC FORWARDING HARDWARE?

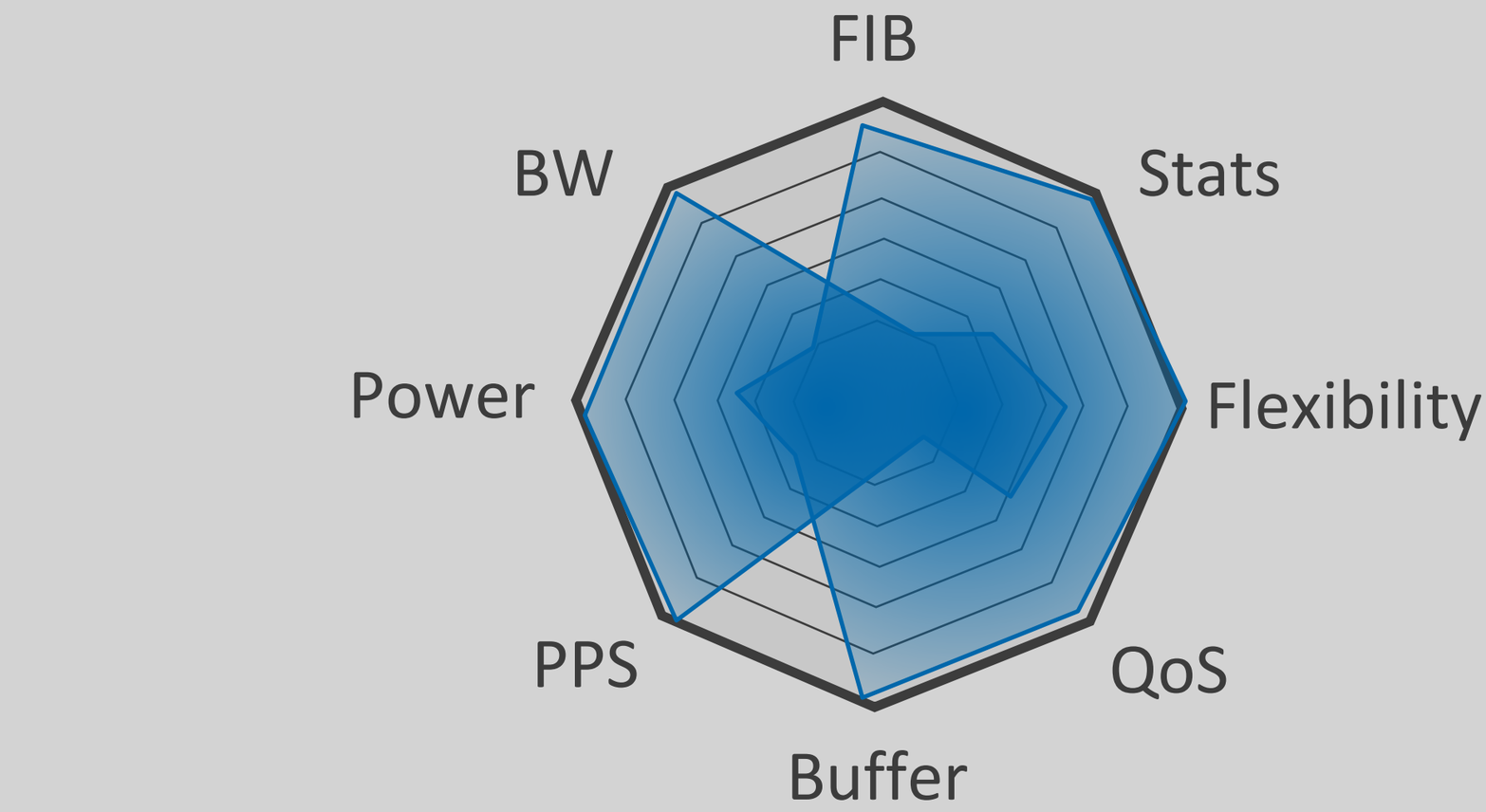
- If dedicated multi-stage architectures (like e.g. pipelines) are limited, why bother?
- In a word: efficiency
  - Operations per packet per Watt is far higher
  - Throughput per unit volume is far higher
  - It is not uncommon for a pipeline to sustain 500M–1B packets per second
  - That’s 50–100 times faster than an x86

SO WHAT?

- Okay, multi-stage architectures are a thing. Why should I care?
- Because of their efficiency, multi-stage architectures like pipelines are in widespread use
- Multi-stage architectures like pipelines have particular characteristics
- Certain protocol design characteristics are awkward fits

TRADE-OFFs

What do you use your available surface for?



POWER

- Real Money – recurring bill (and carbon footprint)
- Power availability / cost differs by location / total amount
- Power leads to cooling - a downstream design problem
- Power and cooling have physical limits within a platform and a given space (at least for forced air)

MEMORY – QoS / Buffer / FIB

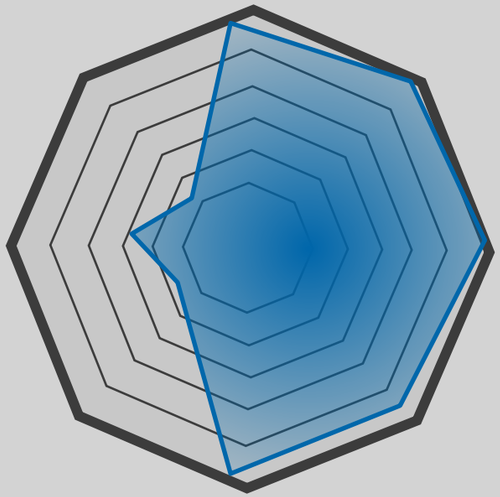
- Memory access (on-chip vs. off-chip / multi-chip);
- Surface for customer interfaces vs. surface for internal ports
- Speed of memory access (on-chip vs. off-chip)
- Amount of memory available at these speeds
- Memory structures for queues – number of operations for (hierarchical) queuing
- See “power” (related)

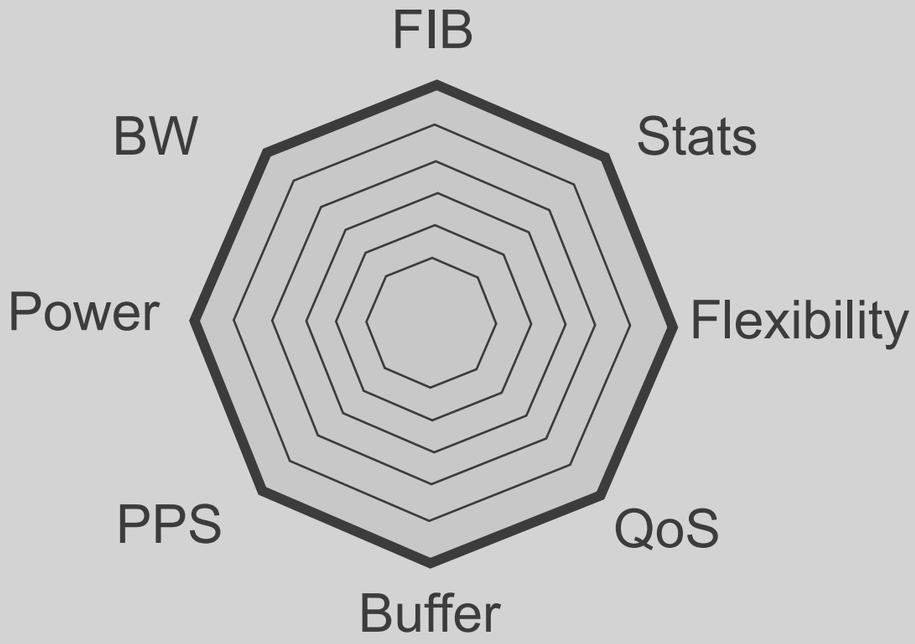
STATS

- Stats collection
  - Collecting stats requires effort;
  - Spend gates/ops on collecting stats or to forward packets?
  - See BW (related)
- Stats export
  - How much, how frequent?
  - Aggregated stats?
  - Shadow collection infra to identify top talkers requires memory…

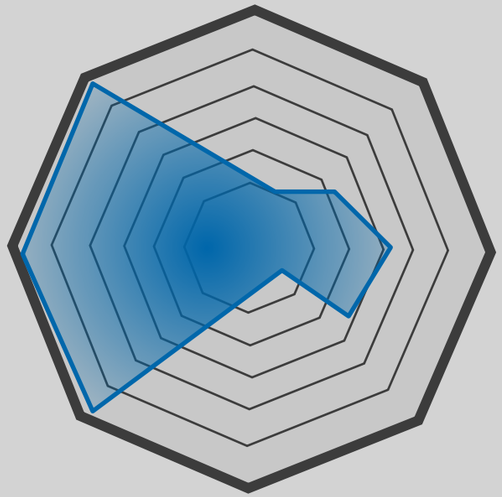
TRADE-OFFs & DEPLOYMENT DOMAINs

SP-Style





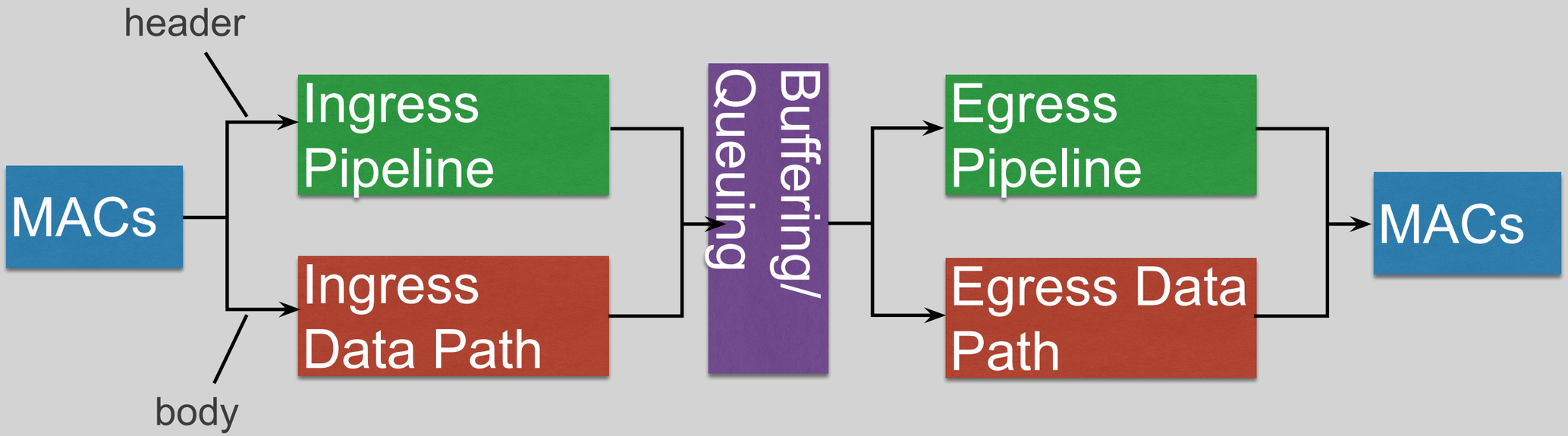
DC-style



MULTI-STAGE ARCHITECTURES: CHARACTERISTICS

Using Pipeline-Architectures As An Example

PIPELINE ROUTER BLOCK DIAGRAM



HEADER VS. BODY

- Processing
  - Lots of stages
  - Variable delay, looping, out-of-order execution, etc.
  - Expensive (power, area, complexity) to transport long header chains
- Data Path
  - RAM-based
  - Optimized for temporary storage of variable length byte strings

PIPELINE STAGES

- Pipelines break down processing into bite-sized chunks
  - For example:
    - Parsing, receive context, destination lookup, etc.
  - Each stage performs a specific operation and delivers results to the next stage
  - Stages *can* be made more complex - at the expense of trading speed for complexity
- Ideally, the stages work on header data sequentially
  - Out-of-order parsing or loops in the parse tree (e.g. overloaded next header type in MPLS) can pose significant challenges

INSTRUCTIONS VS. GATES

- Instructions live in RAM and are infinitely flexible, but relatively slow
- Logic gates are fast and massively parallel
- Complex logic and math can operate at a blazing speeds
- But… gates are hard-wired and cannot be changed
- Replacing logic gates with RAM-based tables enables flexibility, but decreases efficiency

HIGHLY SIMPLIFIED HARDWARE VIEW

- PFE design is always a set of tradeoffs between complexity and speed
  - Processing with a STACK and loadable instruction sets are really flexible, but also really slow.
  - Processing everything in a tightly integrated pipeline is really fast, but much less flexible (or totally inflexible at the end-case)
- In reality, all NPU designs are a series of tradeoffs trying to optimize around some very precious resources:
  - Transistor counts: Every gate I use for processing a packet I cannot use for another interface (i.e. a port-density for features tradeoff)
  - Memory ***depth*** (how many whatzits can I store?) and access rate – to process a BPPS where each requires 10 lookups means 10 billion memory lookups per second.

HEADER VOCABULARY

- Pipelines generally have a fixed header vocabulary
- Accommodating new ways of stacking and using headers might be okay
  - Though it might well be that even if the NPU knows about the header formats, they might be pipelined in a way that differs from the original design, so that things become slow or even impossible.
- Accommodating entirely new headers may require new silicon

AGAIN, SO WHAT?

- Protocol and header design greatly impacts hardware design
- Certain protocol design choices increase hardware complexity and delay adoption
- Given the ubiquity of pipeline-based designs, it behooves the protocol designer to consider their characteristics to help foster widespread adoption and deployment.

CONSIDERATIONS

Things to keep in mind when designing a new protocol

HEADER SIZE LIMITS

- Pipelines generally split header data from packet bodies
- A long series of large headers may not fit in the allotted space; Looking deeper in the packet can slow things down or even make things impossible
  - Some NPUs have hard limits on how deep they can parse
- Minimize the forwarding address size (48, or 64 bit lookup is much nicer than 128)
  - Smaller, byte-aligned header formats are almost always faster/easier for hardware

HEADER SPACING

- Fixed header lengths make parsing easier
- Fields lengths aren’t an issue (up to a point), but making header lengths multiples of 32 bits improve hardware efficiency
  - But don’t inflate a 16 bit header to 32 bits just for the sake of it

RESPECT THE HIERARCHY

- Ideally, headers are processed in the order that they appear in the packet
- Out-of-order header processing adds significant hardware complexity; Minimize header manipulation like reordering headers within a header stack
- Processing that spans multiple headers increases complexity and eliminates opportunities for optimizations
- Make headers self-contained units of information
- Specifically disallow using fields from other layers in processing the current layer

USE EXISTING HEADERS, TUNNELS

- Whenever possible, use an existing header instead of inventing a new one
- However, avoid using well-defined fields in ways that are unrelated to the header's original definition
- Tunnel entry and exit – find ways to minimize going twice through the forwarding pipe for tunnel entry and exit

FLOW IDENTIFICATION

- Flow identification is necessary for ECMP and LAG load balancing
- Make it clear which fields are reliable for flow identification
- Provide a robust means for carrying flow entropy fields to obviate deep parsing
- The further apart your hash inputs are, both from a literal number of bits and in terms of the protocol stack, the harder or more expensive it is to hash these things

- Future: Can we avoid flow identification and accept packets and fragments reordering?

CHECK VALUES

- Make checksums optional/experimental
- Don’t have header checksums span multiple headers
- Avoid checksums that span the entire packet
- Don’t require nested CRCs

STATE ON FORWARDING DEVICES

- Avoid or minimize keeping state in a forwarding node
- Consider degrading Multicast to Unicast as early as possible

IN CONCLUSION…

- Quite simply, don’t assume complete processing flexibility.

- Trade-offs differ between “deployment domains”, though “deployment domains” might change
  - what is niche today, can be broad tomorrow

Personal Perspective

Bad code, bad specs, simple vs. intelligent forwarding planes...

Example: Router Alert experience

- Router Alert: RFC2113(IPv4)/RFC2711(IPv6)

  Hop-by-hop “inspect” packet by routers

- Example: PGM – reliable multicast transport (RFC3208)

  PGM-router-assist: Signaling and retransmitted data packets use router alert
  for constrained retransmission to receivers who missed the original data.
  Problem: punted of router-alert packets with distinguishing by next-protocol or Value
  Entirely killed PGM router-assist: No-PGM router punted PGM RA packets and died
  Host stacks introduced (non PGM RFC compliant) messaging without router-alert

- IMHO: Router Alert would perfectly work
  - But IETF was/is? unwilling to write “internal behavior” requirements to make it work
  - *Routers supporting router alert MUST process RA packets for any combination of (next-protocol/value) that it does not support or not currently enables at the same speed as packets without RA (IPv4) or with unsupported hop-by-hop-header (IPv6)*
    - Aka: MUST only punt/slow-path on a per (next-proto,value) basis, or not support RA at all.
  - But: Would need new ext-header today to do this now
    - Existing RA burned by bad code in the field. Old routers must ignore new version.

The trend for simpler fastpath and control-plane

- Punting = packets passed from fastpath to slowpath
  - Fastpath = accelerated forwarding plane
  - Slowpath = control plane CPU abused for data path operations
    that are too difficult for accelerated forwarding plane
  - Slowpath became ever more useless with growing fastpath speed
    - E.g.: Terrabit DC switches have “atom class” CPU.
  - Push back on “Difficult packets/processing”
    - RA / hop-by-hop headers, fragmentation/reassembly,…
- “Data-triggered-events”
  - Data-packet proceeds in fastpath, but control plane needs notification
  - “punt signalling” ?!
  - “Netflow forwarding”, BEHAVE “state build/refresh”, Firewall behavior,...
  - IP Multicast protocols 20 year evolution to minimize data-triggered-events
    - DVMRP -> PIM-DM -> PIM-SM -> PIM-SSM -> Bidir-PIM (good implementation: none)
  - Overall: this did NOT necessarily lead to easier to use protocols
    - Just easier to support / scale across variety of HW

How about intelligent fastpaths ?

- Many fast-paths can be ever more intelligent (CPU/NPU/”P4”/...)
  - IMHO: Future value in programmable forwarding is to run ”protocols” in fast-path
  - But those protocols need e.g.: simpler state processing to allow this
    - Therefore sometimes better called “actions” than “protocols”
  - Examples:
    - Class of “iOAM” solutions
    - “NPU based reservation setup/maintenance” *draft-han-tsvwg-ip-transport-qos*
- How to deal with this conflict in further IETF work ?
  - Not all solutions will be best supporting whole spectrum of fastpaths
    - Lowest common denominator (fixed asic) fastpath hardware
    - Intelligent fastpath hardware… Multicore CPU fastpath.
- Data-triggered-events can lead to elegant solutions
  - Example: MAC-address learning in IEEE 802.1, Interesting SPF research/work in that area too.
  - Could IETF even standardize something like this ? (visible protocol on wire ?)