UNIVERSITY OF WATERLOO

CS 856: Programmable Networks

Lecture 3: Programmable Switch Architectures

Mina Tahmasbi Arashloo

Winter 2024

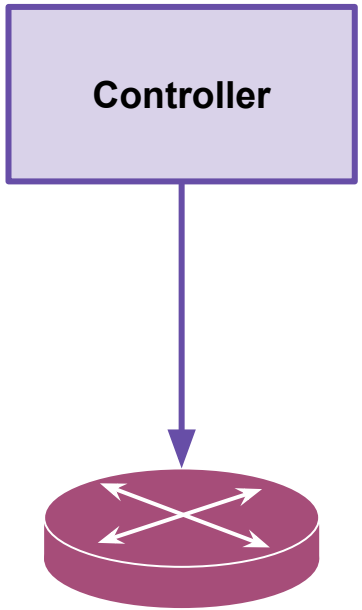
Logistics

- Reviews are due **Monday, Jan 29, at 5pm.**
- Project proposal is due **Jan 31**.
- Extra instructions for M1 and M2 chips in the assignment repo.

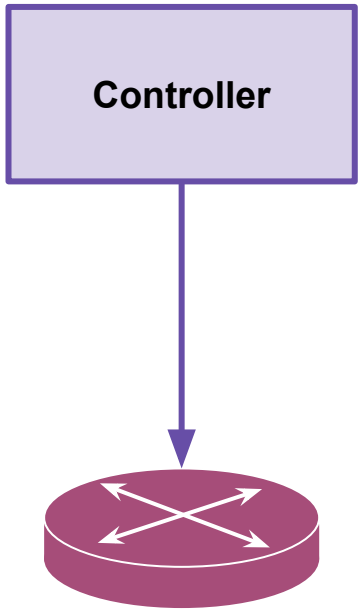
Recap: Making the data-plane "more programmable"

- OpenFlow started as a simple abstraction of the data plane
  - One big look-up table, matching on 12 fields, a handful of actions.
- It quickly grew larger
  - There was a need more fields, multiple tables, …
- Why not open the interface even more?

Controller to switch

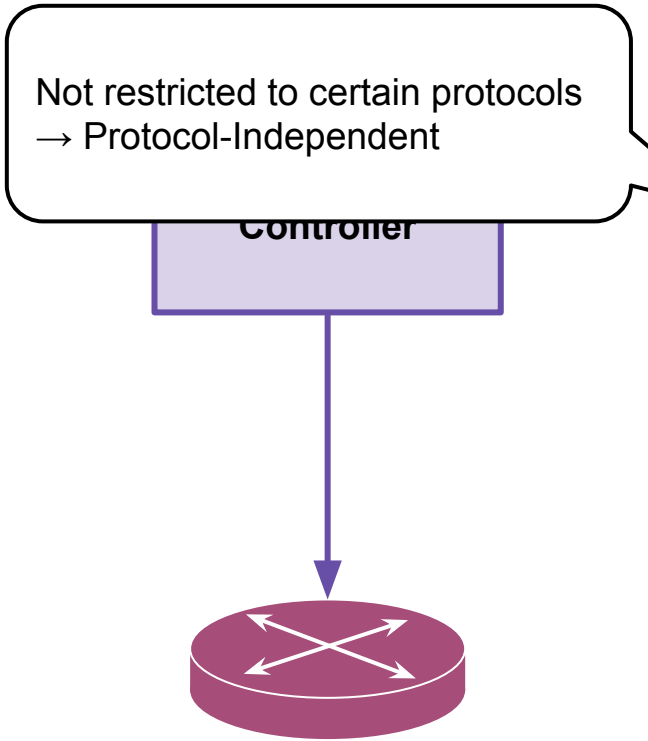


- **Runtime communication**
  - add/remove/modify table entries
  - send packet
  - request traffic statistics



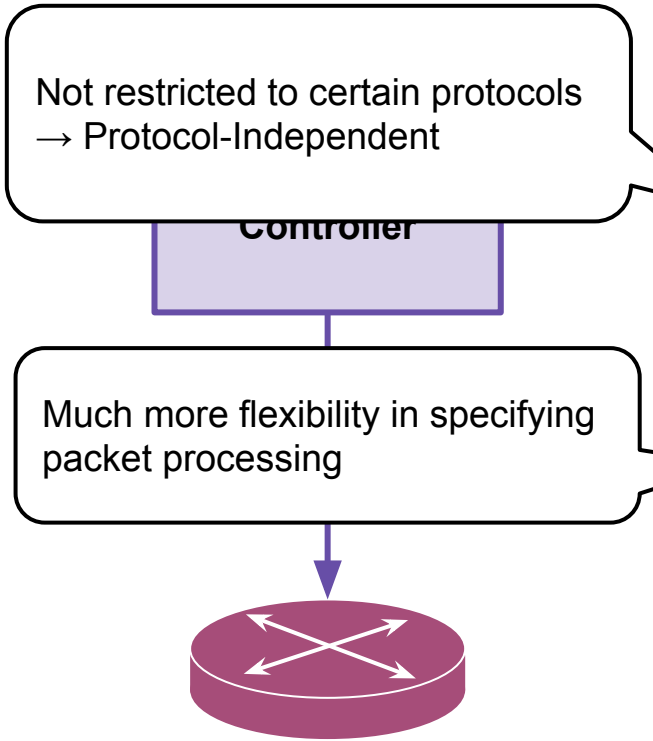
Controller to switch

- **Headers and Parsing**
  - Header X and Y look like this
  - To parse header X, look at the bytes B1 to B2 in the packet…
- **Table Configuration**
  - Table T1 should use X for match and A1 or A2 for actions.
  - Table T2 should use …
- **Runtime communication**
  - add/remove/modify table entries
  - send packet
  - request traffic statistics



Controller to switch

- **Headers and Parsing**
  - Header X and Y look like this
  - To parse header X, look at the bytes B1 to B2 in the packet…
- **Table Configuration**
  - Table T1 should use X for match and A1 or A2 for actions.
  - Table T2 should use …
- **Runtime communication**
  - add/remove/modify table entries
  - send packet
  - request traffic statistics

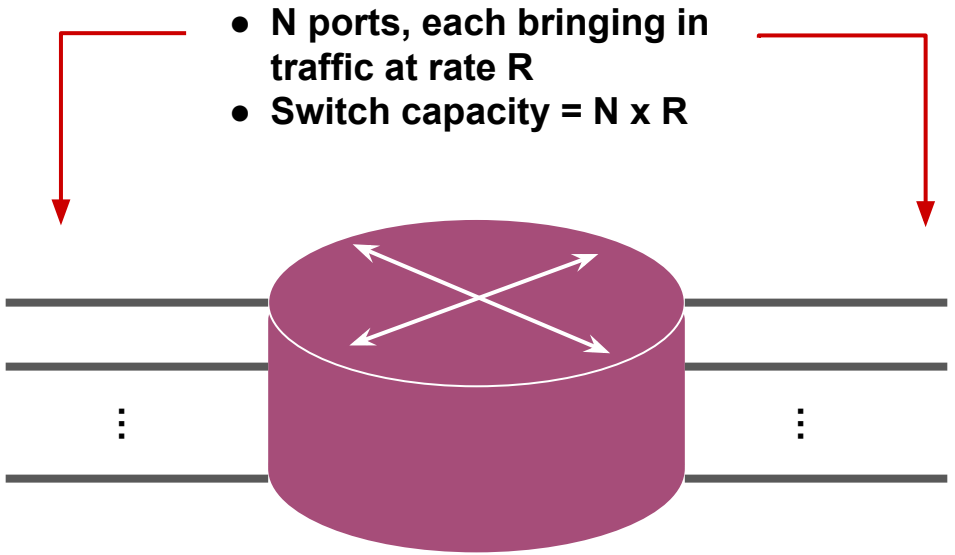


Controller to switch

- **Headers and Parsing**
  - Header X and Y look like this
  - To parse header X, look at the bytes B1 to B2 in the packet…
- **Table Configuration**
  - Table T1 should use X for match and A1 or A2 for actions.
  - Table T2 should use …
- **Runtime communication**
  - add/remove/modify table entries
  - send packet
  - request traffic statistics

Challenge: High-Speed Reconfigurable Data Plane

- Switch data planes need to process packets very fast



Challenge: High-Speed Reconfigurable Data Plane

- Switch data planes need to process packets very fast

R = 100 Gbps

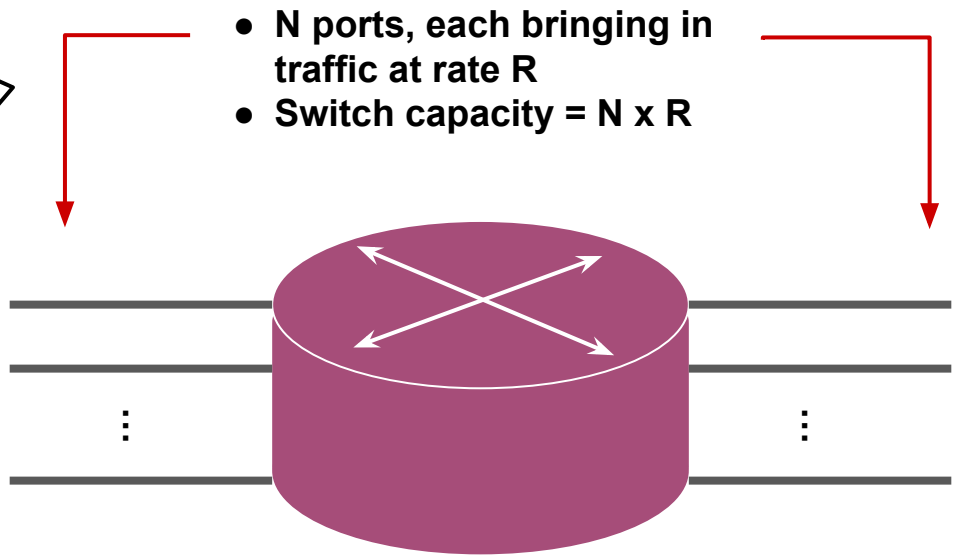
For back-to-back 64B packets, we have a packet every ~5ns.

For back-to-back 1500B packets, we have a packet every ~120ns.

N = 16

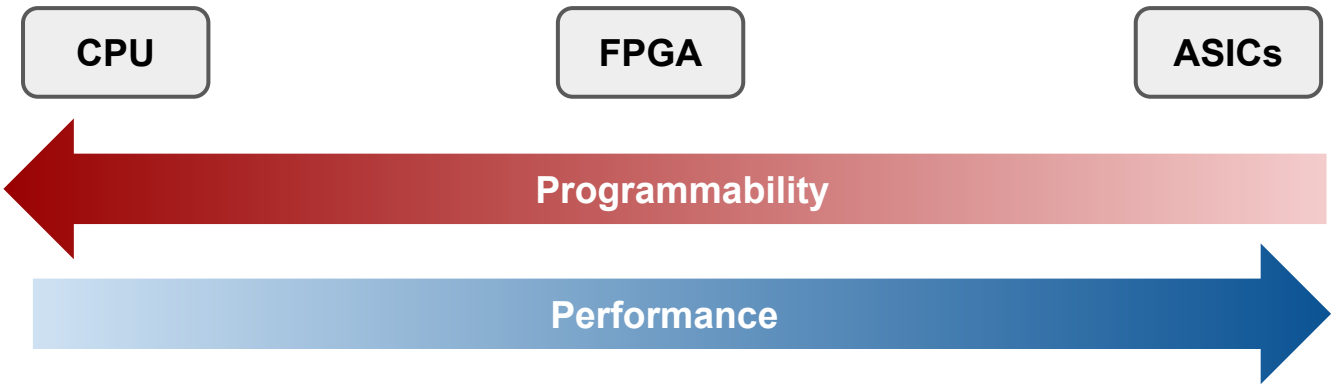
This happens concurrently on 16 ports…

N x R = 1.6 Tbps!



Challenge: High-Speed Reconfigurable Data Plane

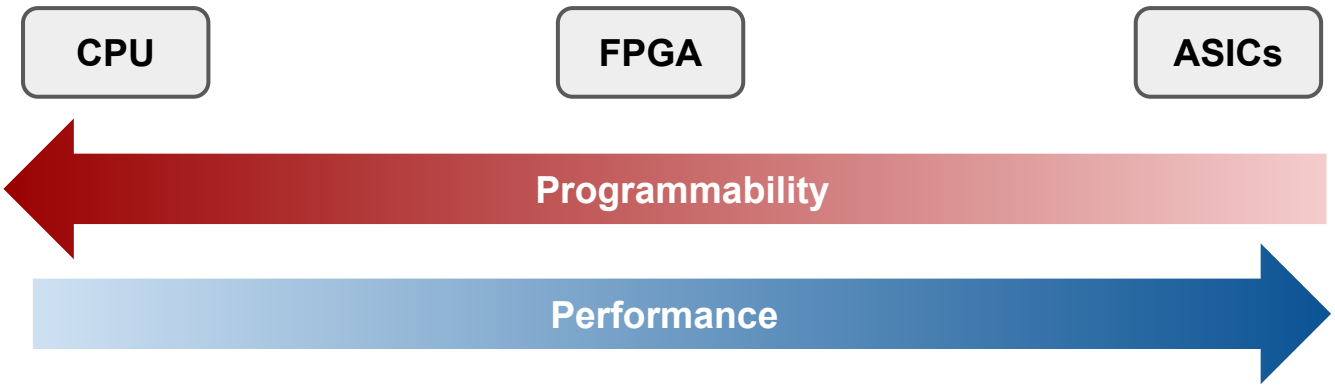
- There is a trade-off between programmability and performance



Challenge: High-Speed Reconfigurable Data Plane

- programmability and performance

General-purpose processors like CPUs can be programmed to execute any logic.

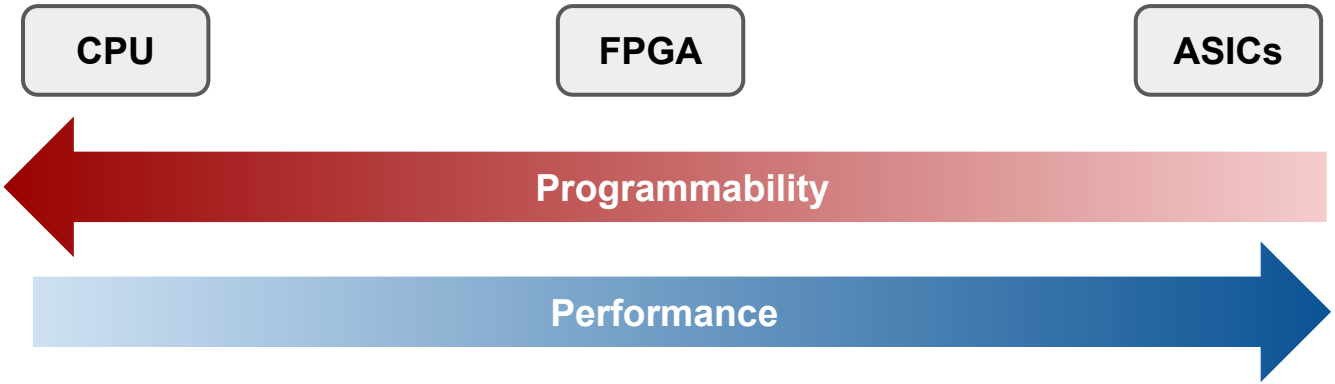


Challenge: High-Speed Reconfigurable Data Plane

- programmal

General-purpose processors like CPUs can be programmed to execute any logic.

Fixed-function ASICs are customized and optimized to for a certain kind of computation.

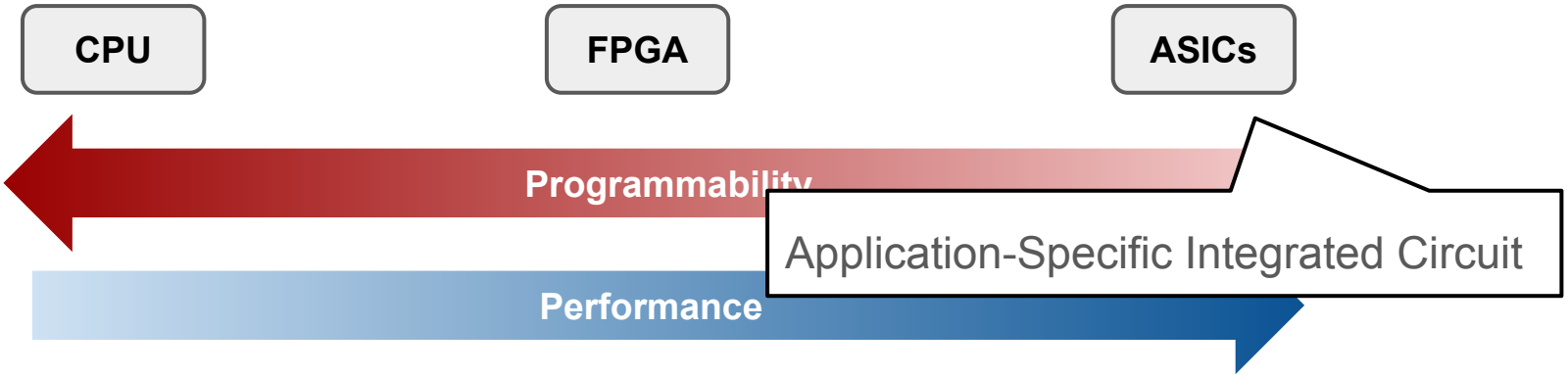


Challenge: High-Speed Reconfigurable Data Plane

- programmable

General-purpose processors like CPUs can be programmed to execute any logic.

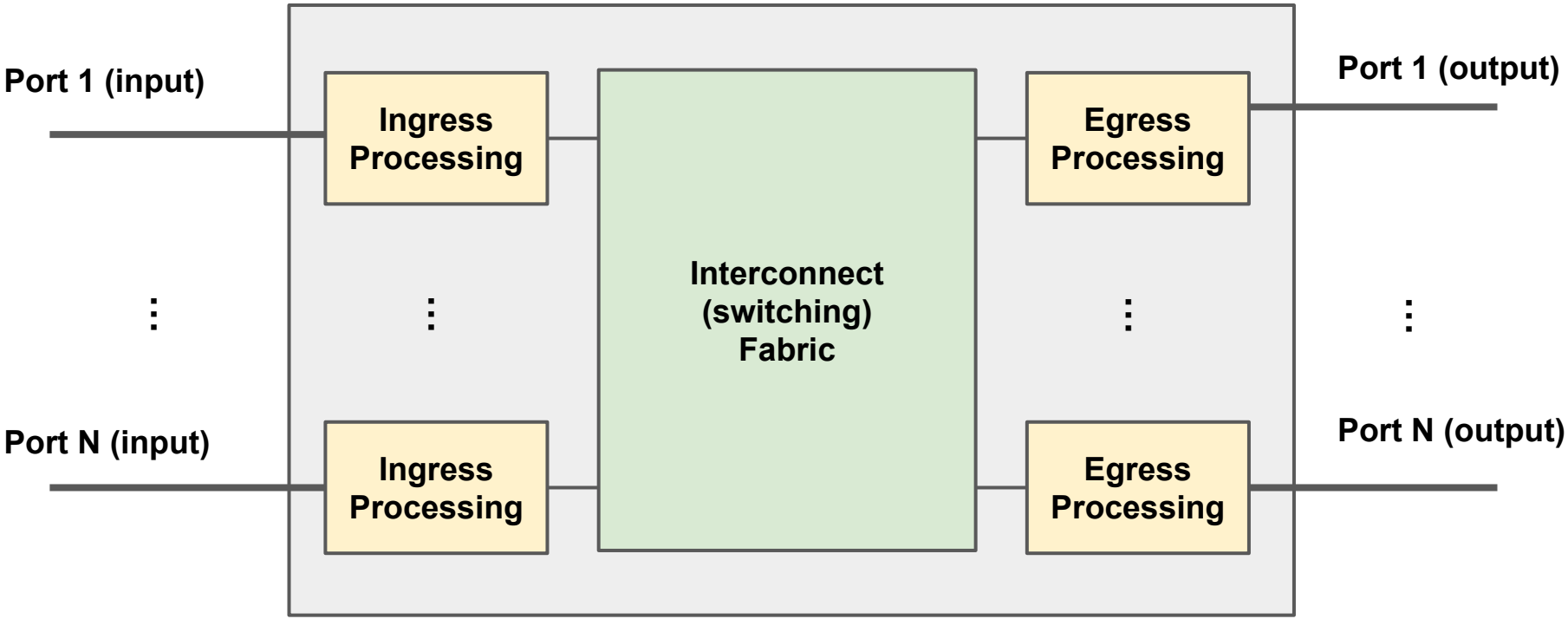
Fixed-function ASICs are customized and optimized to for a certain kind of computation.



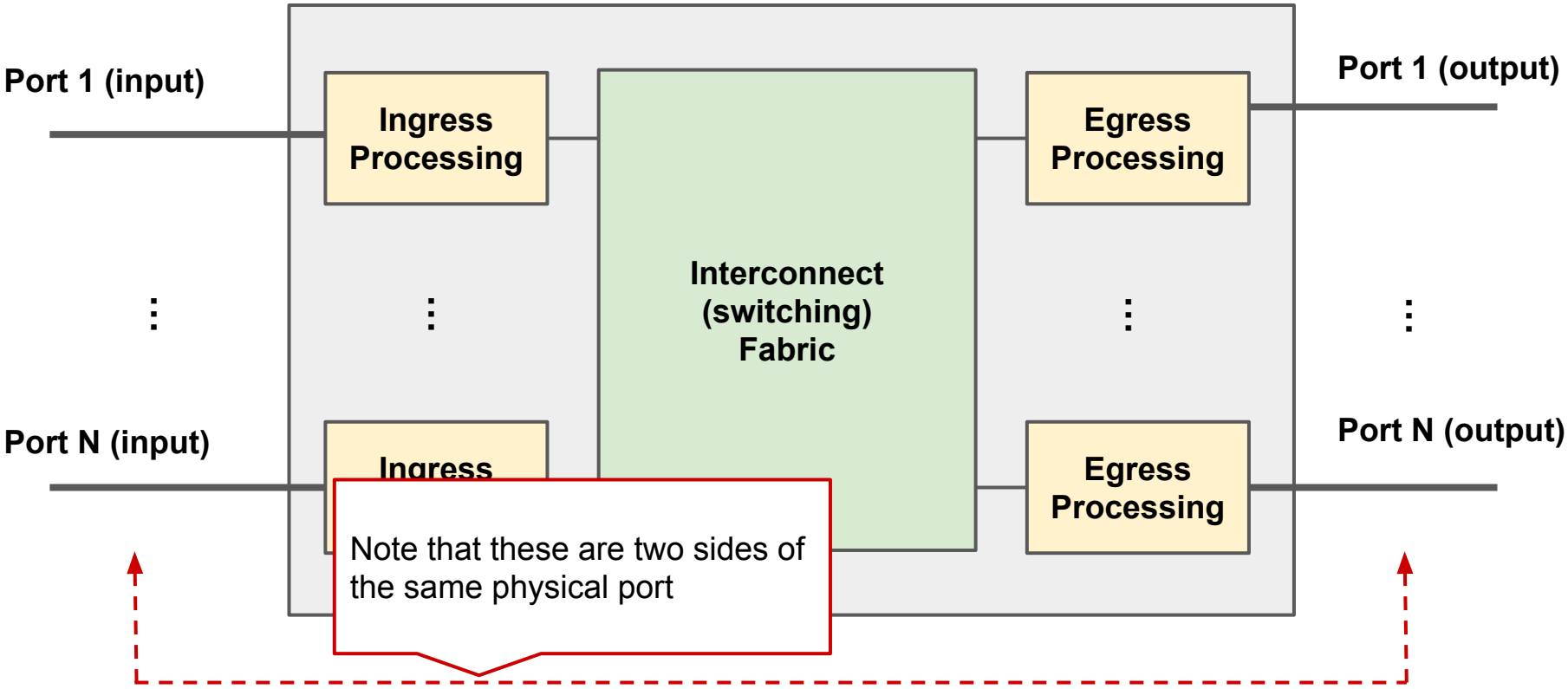
Challenge: High-Speed Reconfigurable Data Plane

- **Traditionally:** switching chips were ASICs
  - customized for packet processing, e.g., packet parsing, forwarding tables, etc.
- **The "programmability" trend:**
  - **Q1:** Is it possible to have a high-speed reconfigurable switch data plane?
  - **Q2:** How much reconfigurability can we add to the switch data plane and still be able to perform high-speed packet processing?

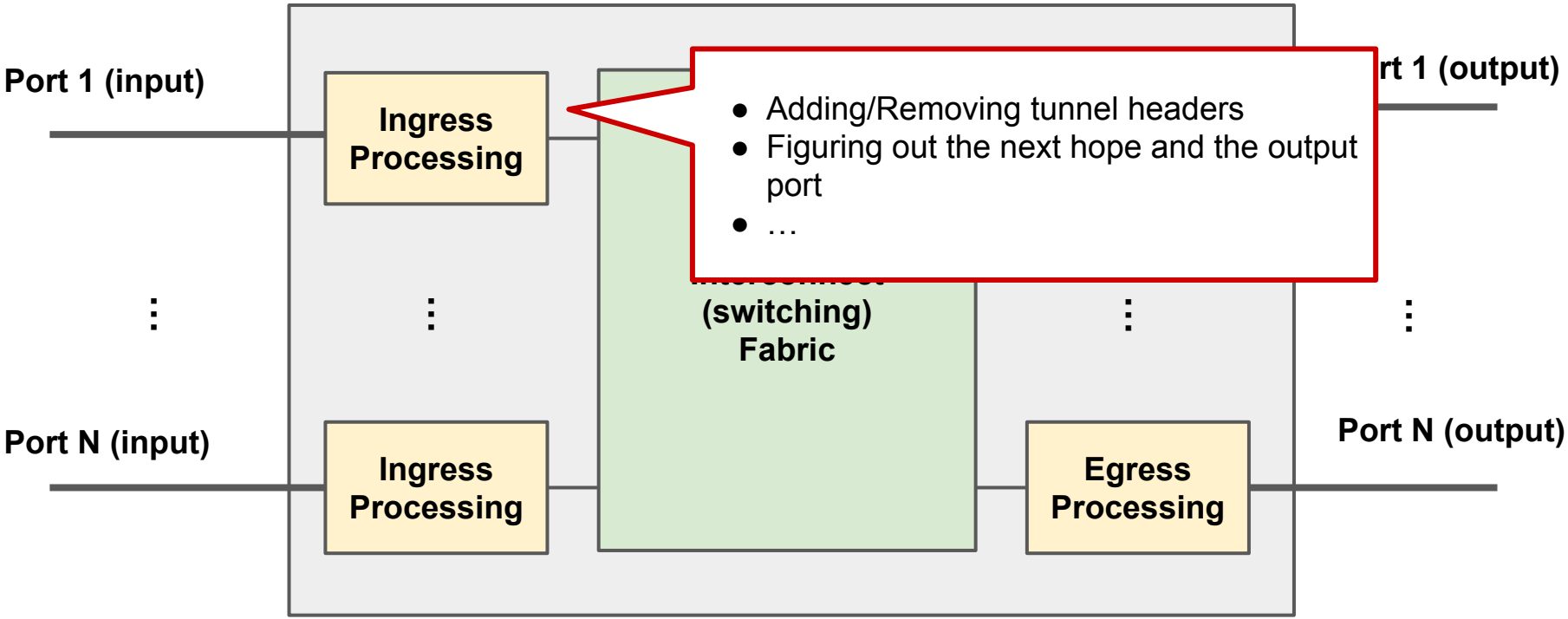
Inside a (output-queued) switch



Inside a (output-queued) switch

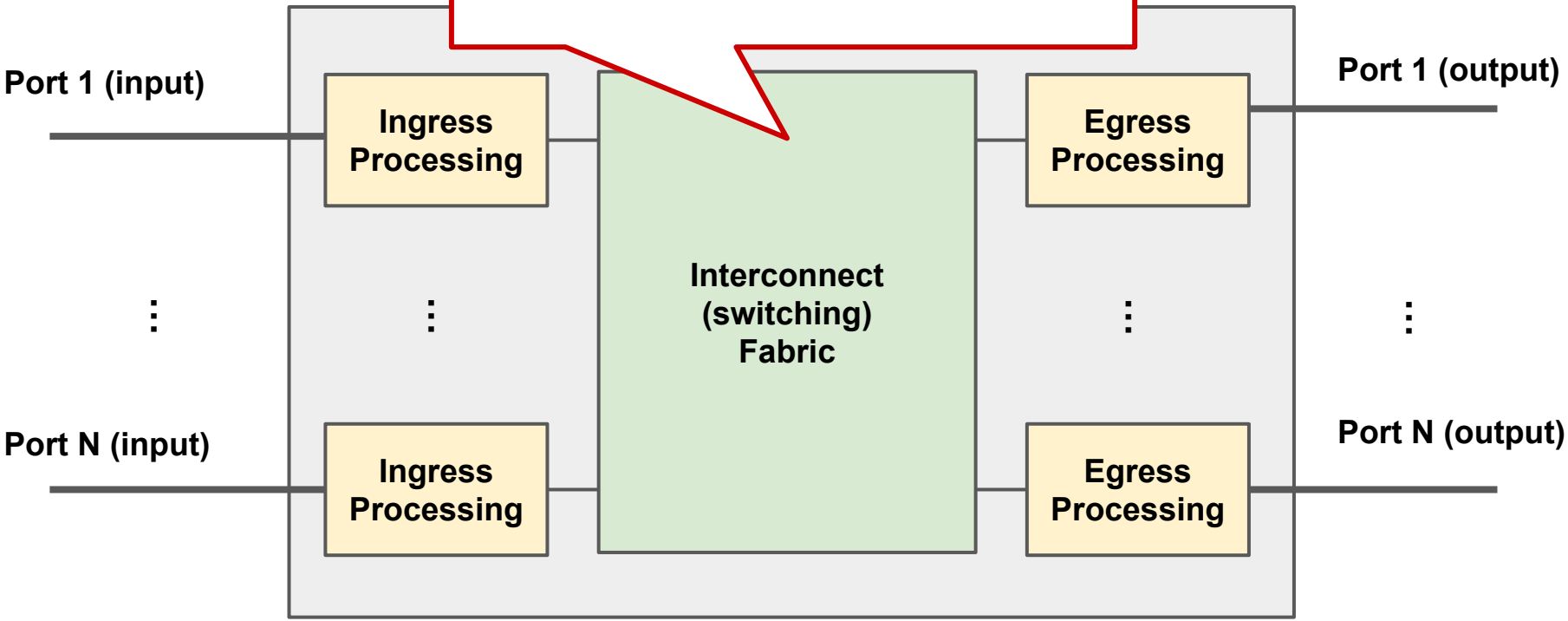


Inside a (output-queued) switch



Inside a (output-c

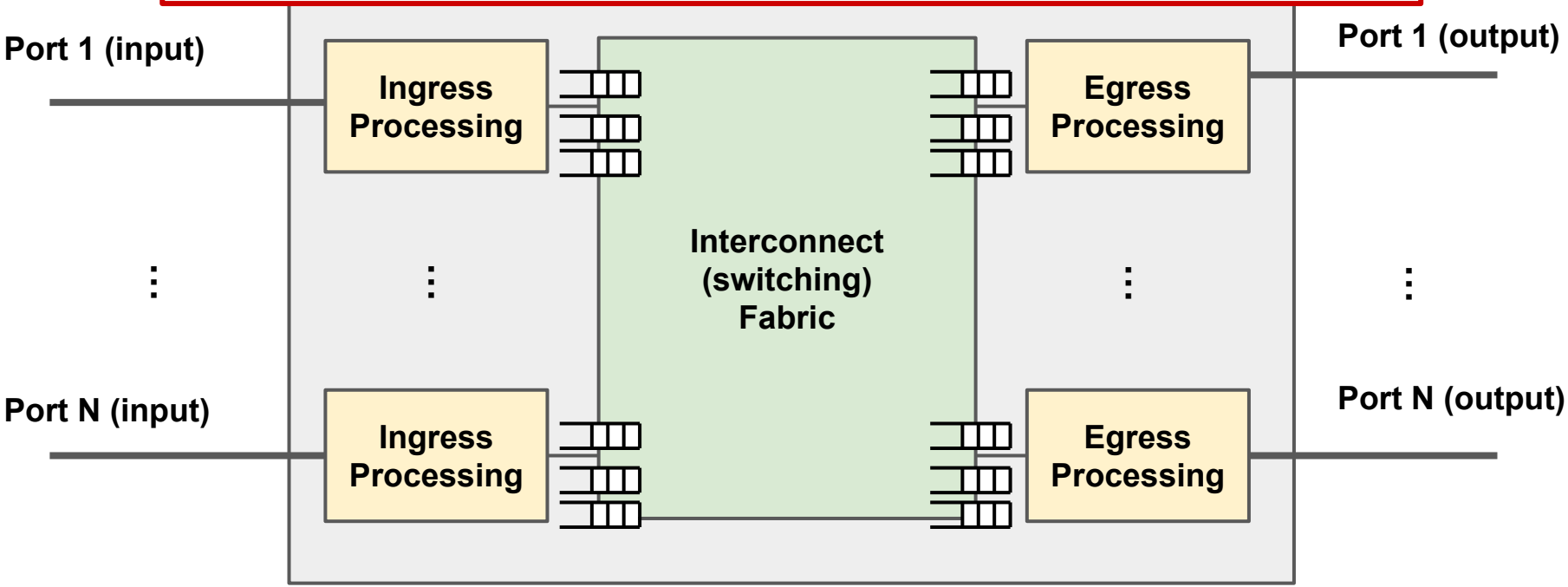
- Connects input ports to output ports
- Needs to operate at high speed (~ N times the speed of an individual port)



Inside

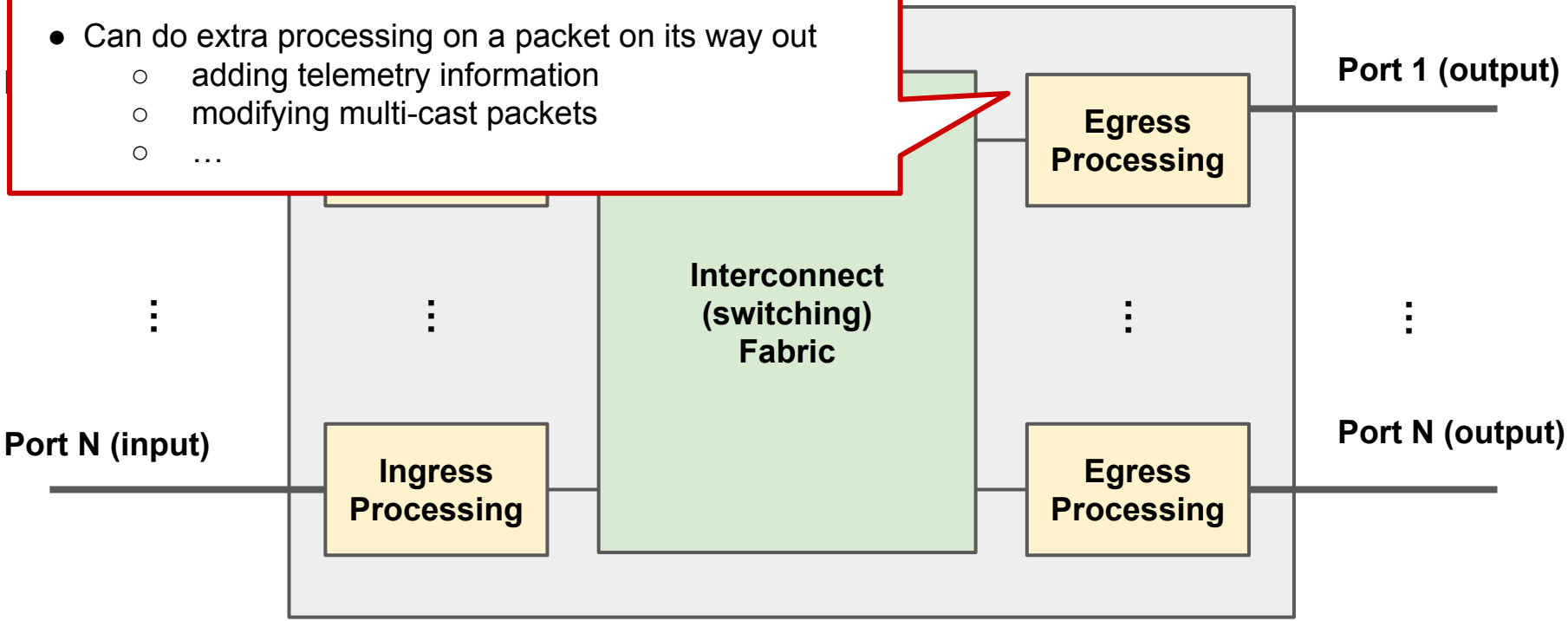
Traffic manager:

- Packets going to the same output will be buffered in a queue
  - In ingress and/or egress.
- Packet scheduling algorithms decide which packets will go out of that port next

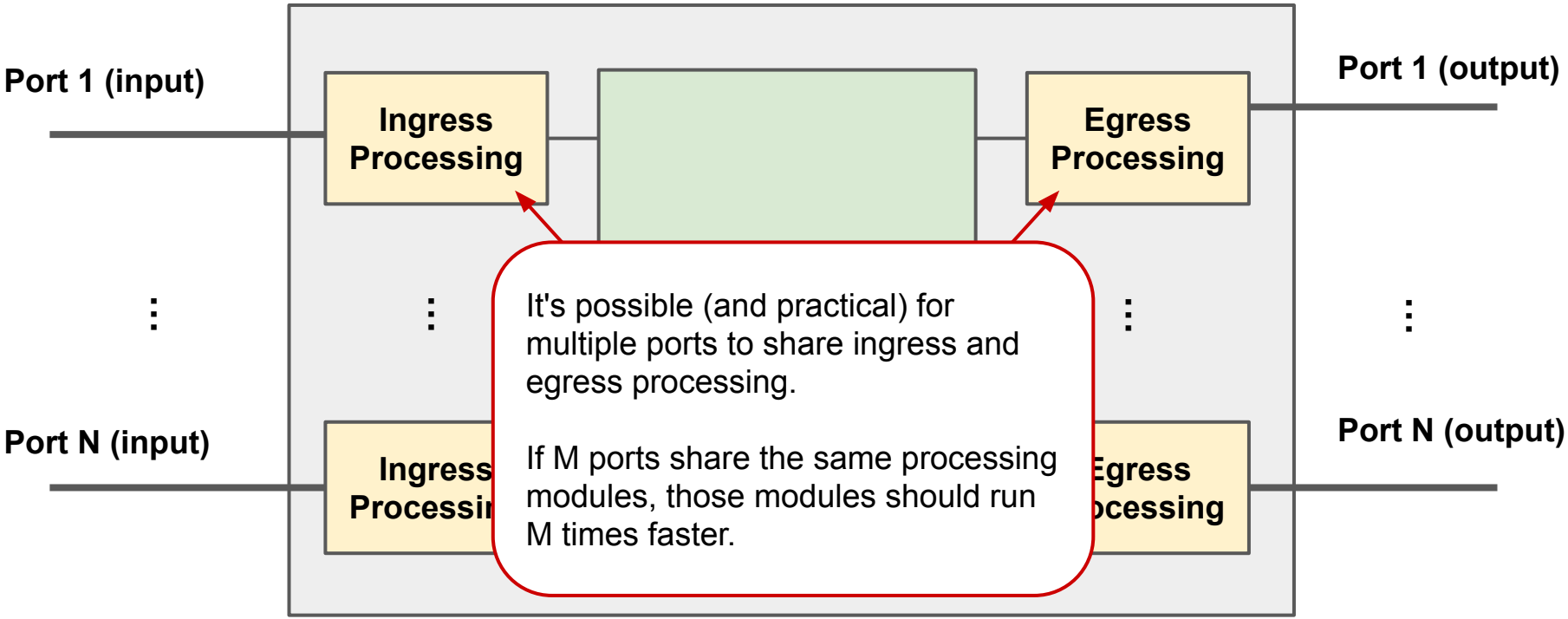


Inside a (output-queued) switch

- Can do extra processing on a packet on its way out
  - adding telemetry information
  - modifying multi-cast packets
  - …



Inside a (output-queued) switch

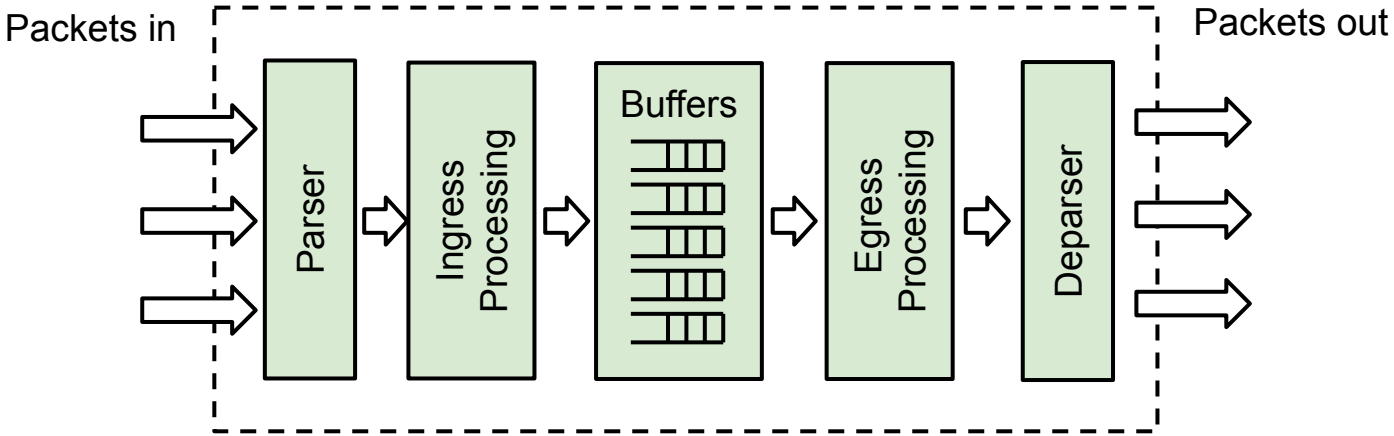


What should a "programmable" switch look like?

- We can't make everything programmable
  - the programmability-performance trade-off
- How do we decide what should be fixed and what programmable?
  - Which parts are subject to more innovation?
  - The logic of which part do we want to change more frequently?
  - Where can we afford to pay the overhead of programmability?

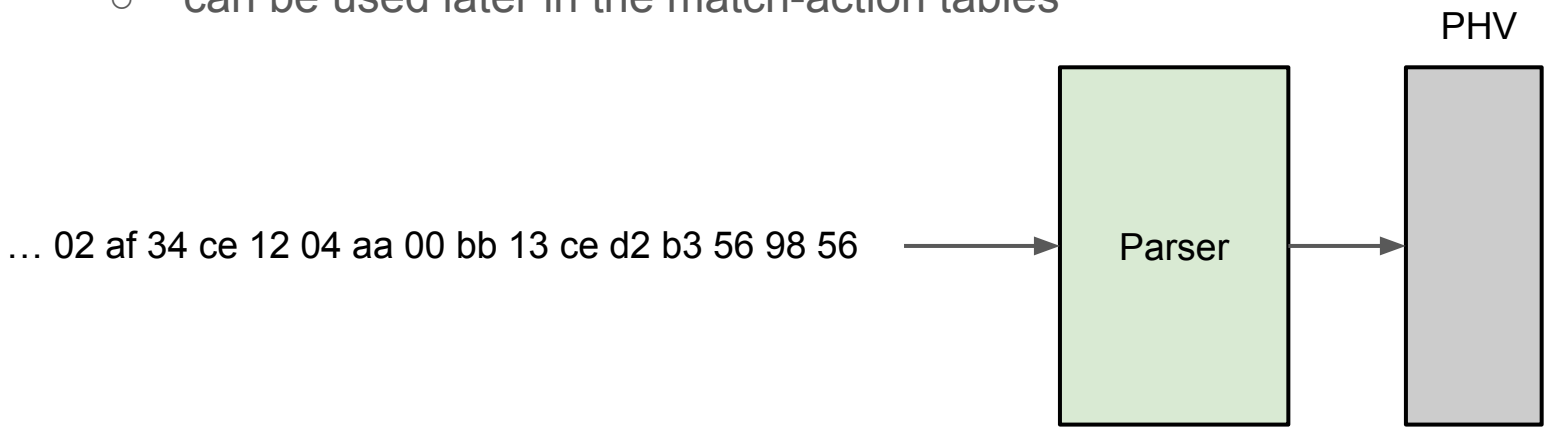
PISA: **P**rotocol-**I**ndependent **S**witch **A**rchitecture

- First academic proposal → Reconfigurable Match Tables (RMT)
- Later evolved and renamed PISA



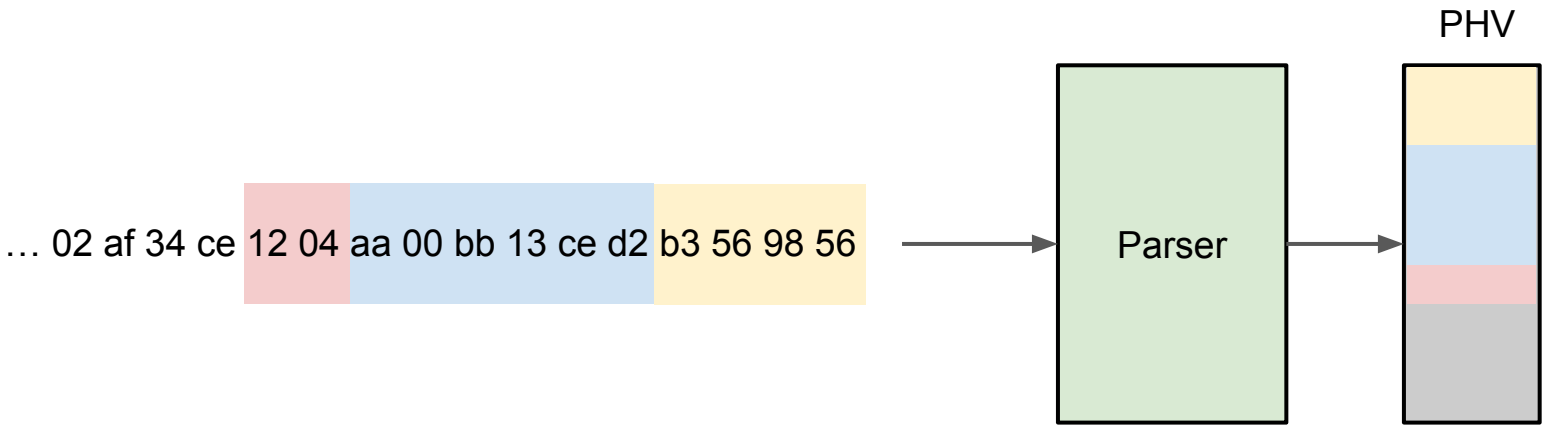
Programmable Parser

- Takes bits from a packet and outputs a *Packet Header Vector (PHV)*
- PHV: the collection of all the header fields that are
  - parsed from the packet
  - can be used later in the match-action tables

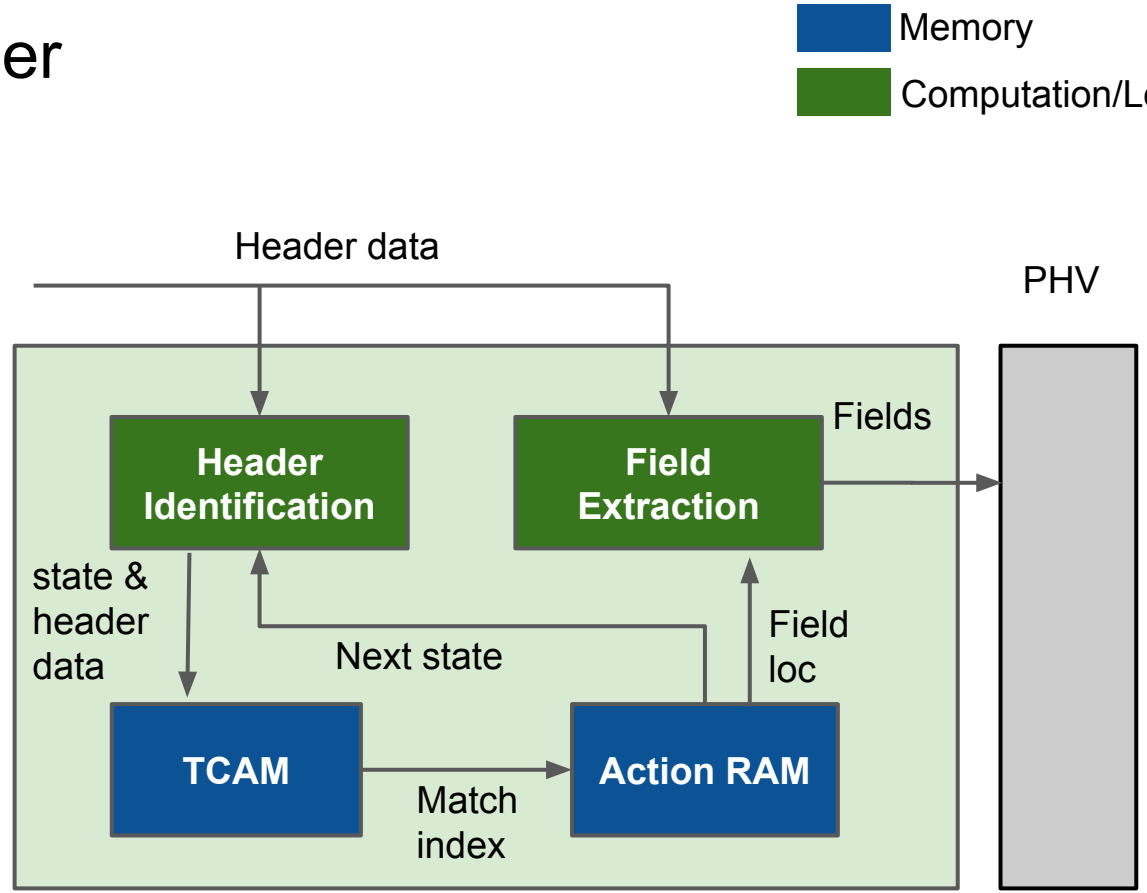


Programmable Parser

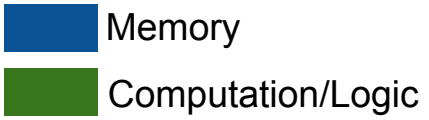
- Takes bits from a packet and outputs a *Packet Header Vector (PHV)*
- PHV: the collection of all the header fields that are
  - parsed from the packet
  - can be used later in the match-action tables



Programmable Parser

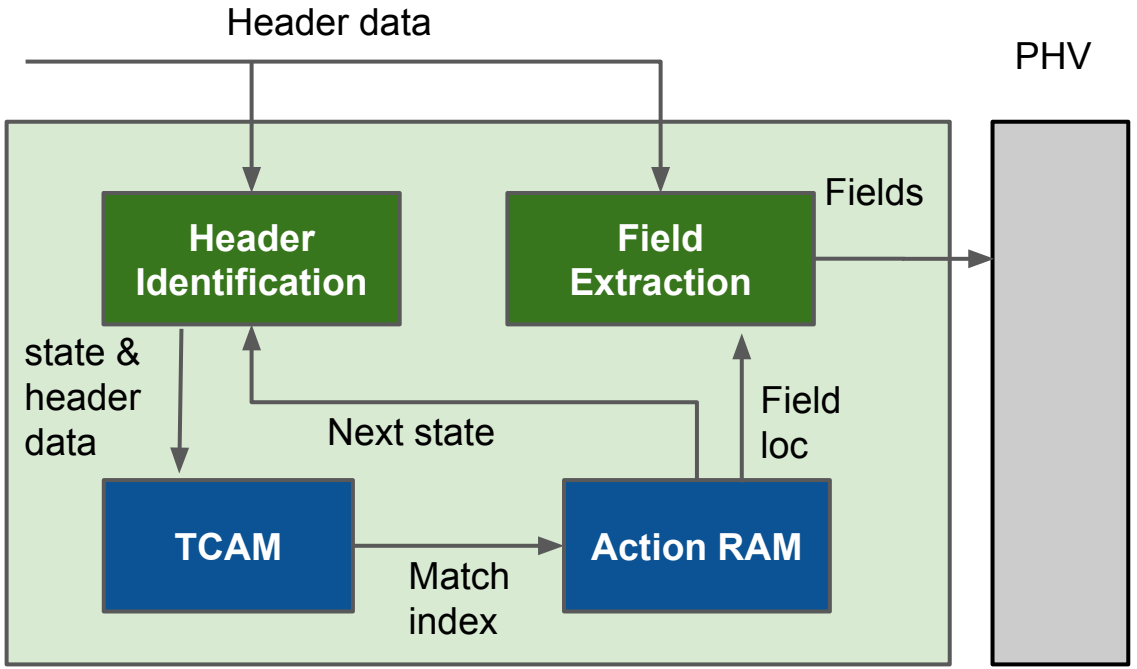


Programmable Parser

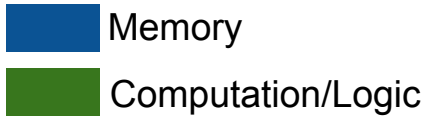


TCAM can be used to implement a match-action table

The parser is "programmed" by changing the contents of the TCAM and the RAM



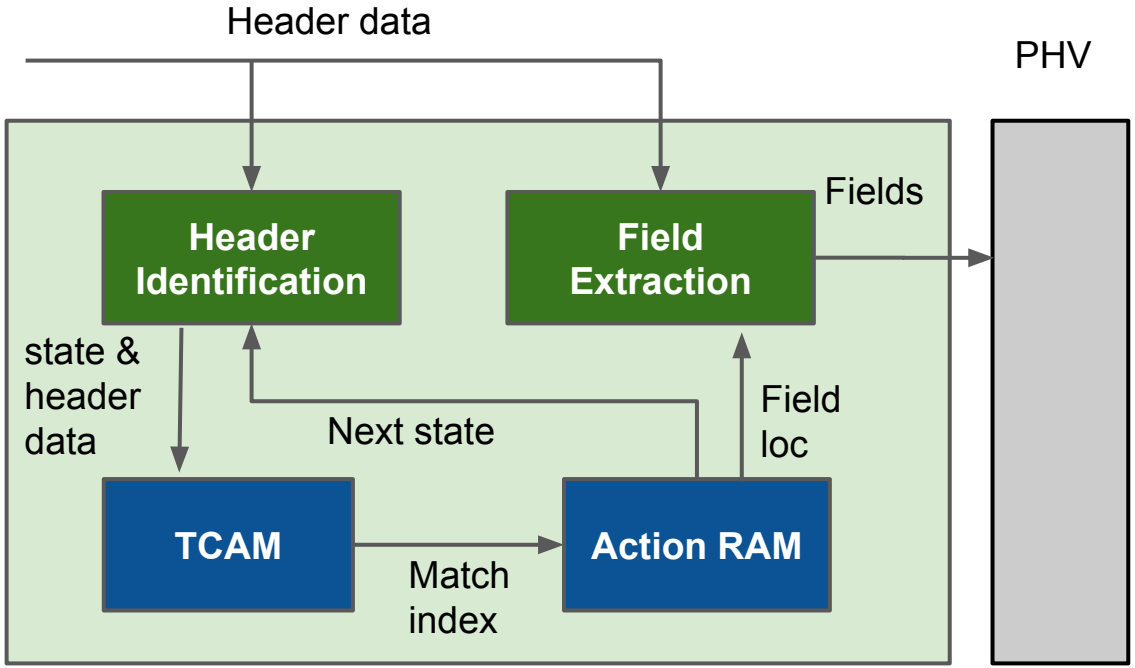
Programmable Parser



- Headers:

```
header H1 {
 bit<4> A;
 bit<1> B;
}
header H2 {
 bit<2> C;
}
```

- If B = 1, we parse H2.



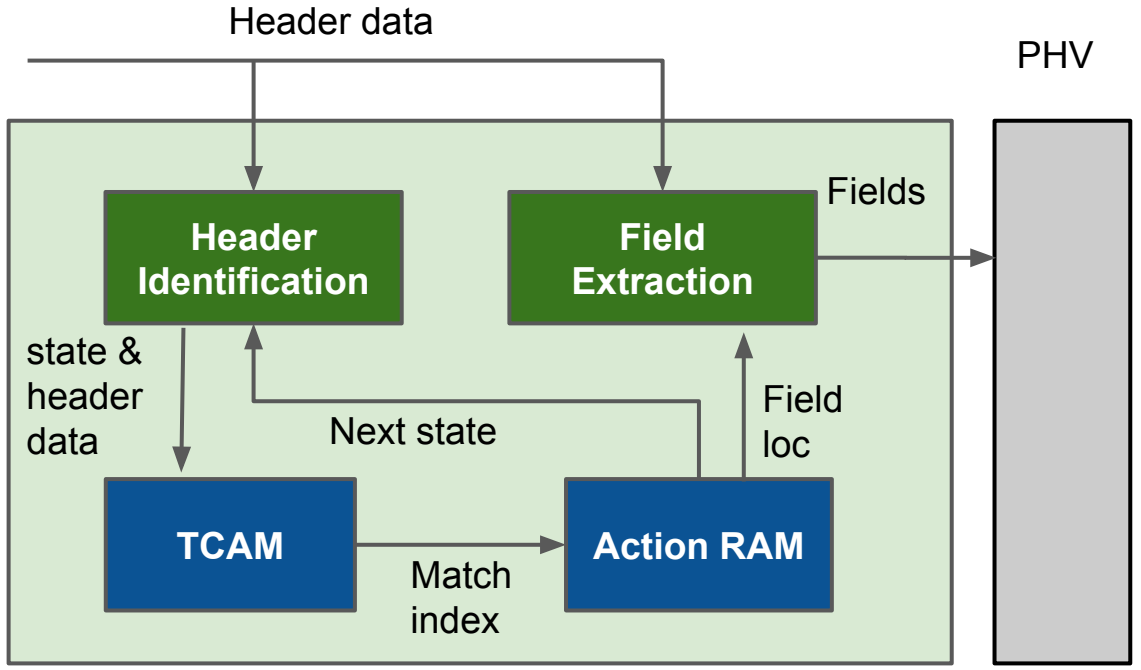
We have three states.

- s0: parse H1
- s1: parse H2
- s2: done parsing all headers

- Headers:

```
header H1 {
 bit<4> A;
 bit<1> B;
}
header H2 {
 bit<2> C;
}
```

- If B = 1, we parse H2.



The TCAM matches on
- **the state**
- **first N bits** in header data.

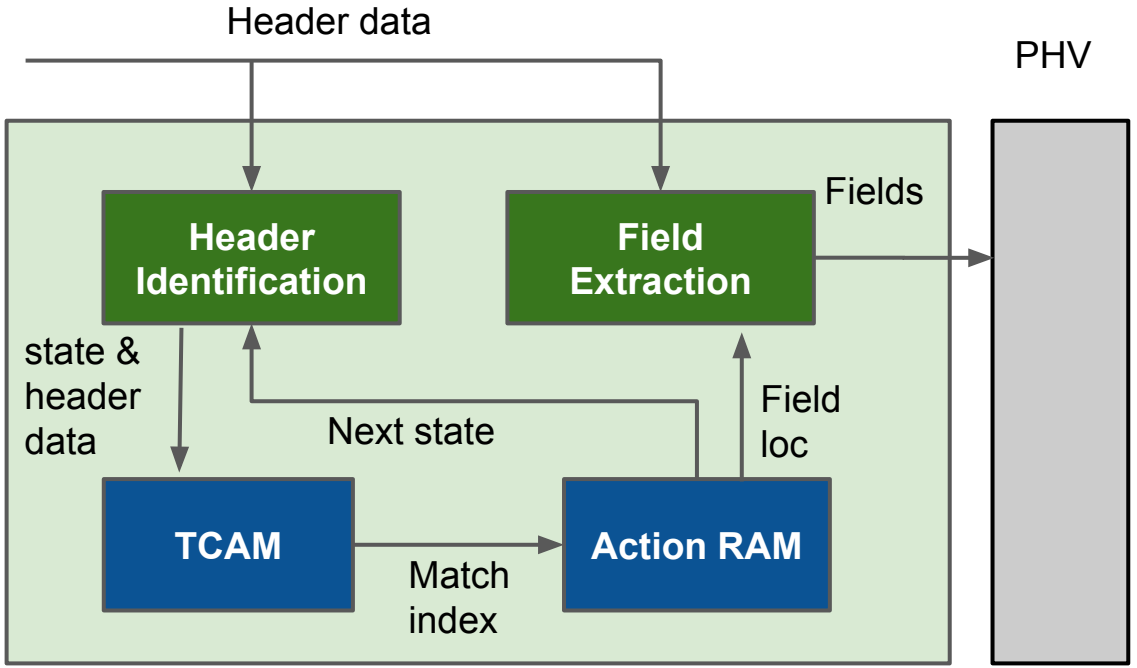
Memory

Computation/Logic

- Headers:

```
header H1 {
 bit<4> A;
 bit<1> B;
}
header H2 {
 bit<2> C;
}
```

- If B = 1, we parse H2.



We populate the TCAM with 3 entries.

Entry 1:
- match: state = s0, **header data [4] = 1**
- action: action 0

The actions are defined in the RAM:

action 0: extract 5 bits (H1) and put them in the first 5 bits of PHV, **go to s1**

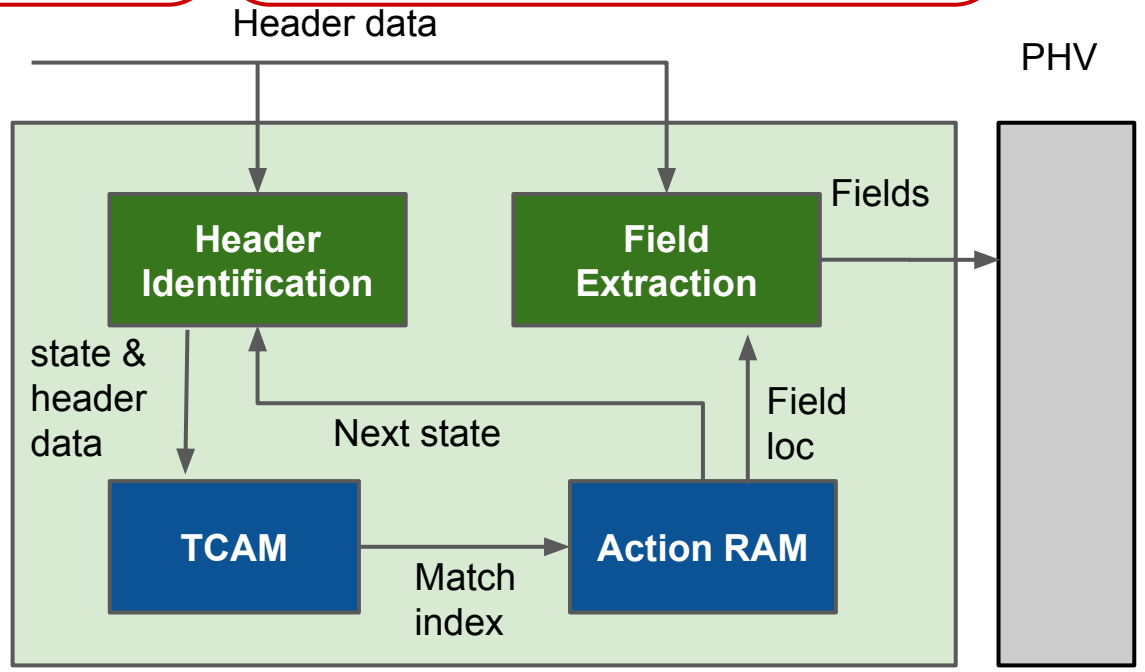
ry

utation/Logic

- Headers:

```
header H1 {
 bit<4> A;
 bit<1> B;
}
header H2 {
 bit<2> C;
}
```

- If B = 1, we parse H2.



We populate the TCAM with 3 entries.

Entry 2:
- match: state = s0, **header data [4] = 0**
- action: action 1

The actions are defined in the RAM:

action 1: extract 5 bits (H1) and put them in the first 5 bits of PHV, **go to s2**

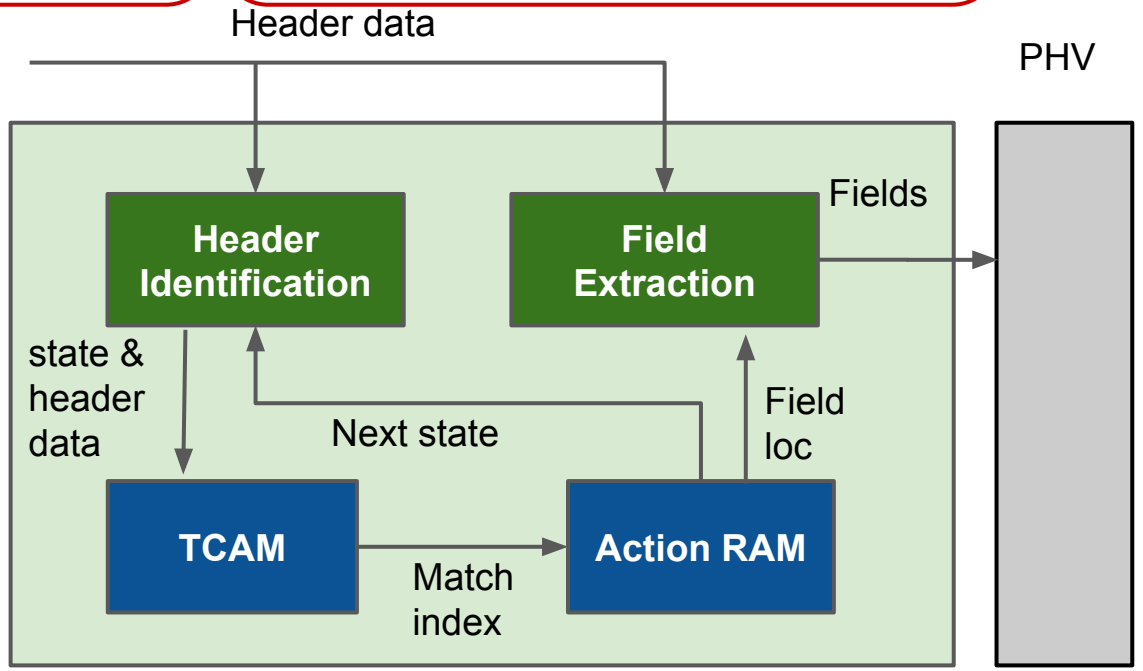
ry

utation/Logic

- Headers:

```
header H1 {
 bit<4> A;
 bit<1> B;
}
header H2 {
 bit<2> C;
}
```

- If B = 1, we parse H2.



We populate the TCAM with 3 entries.

Entry 3:
- match: state = s1, header data = *
- action: action 2

The actions are defined in the RAM:

action 2: extract 2 bits (H2) and put them in the bits 6 and 7 of PHV, **go to s2**

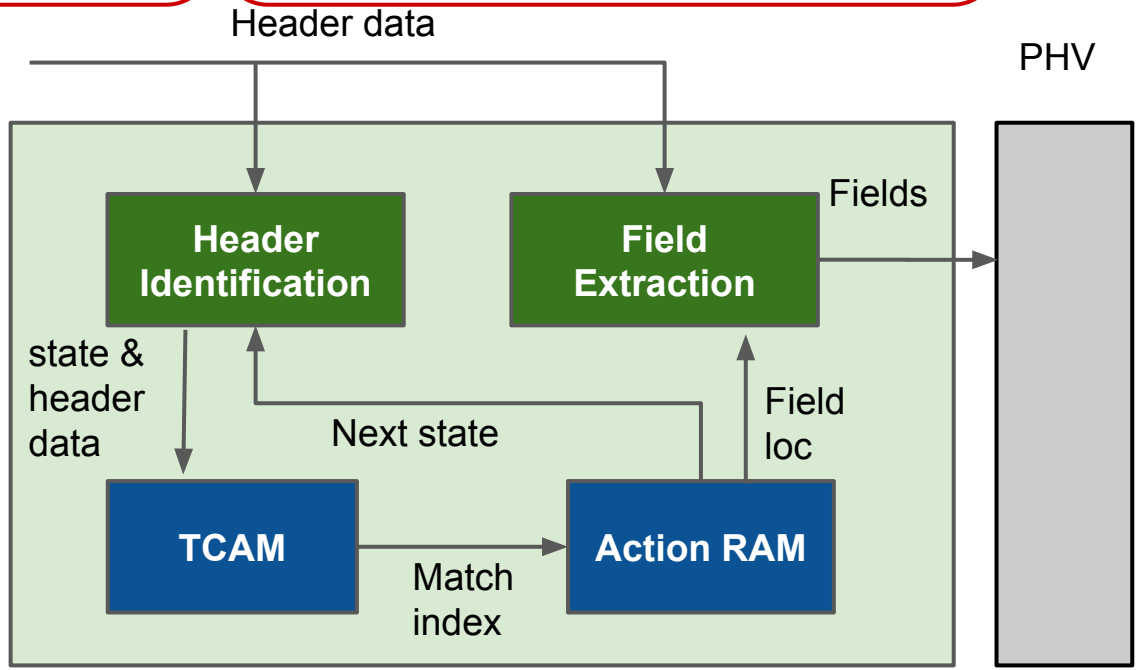
ry

utation/Logic

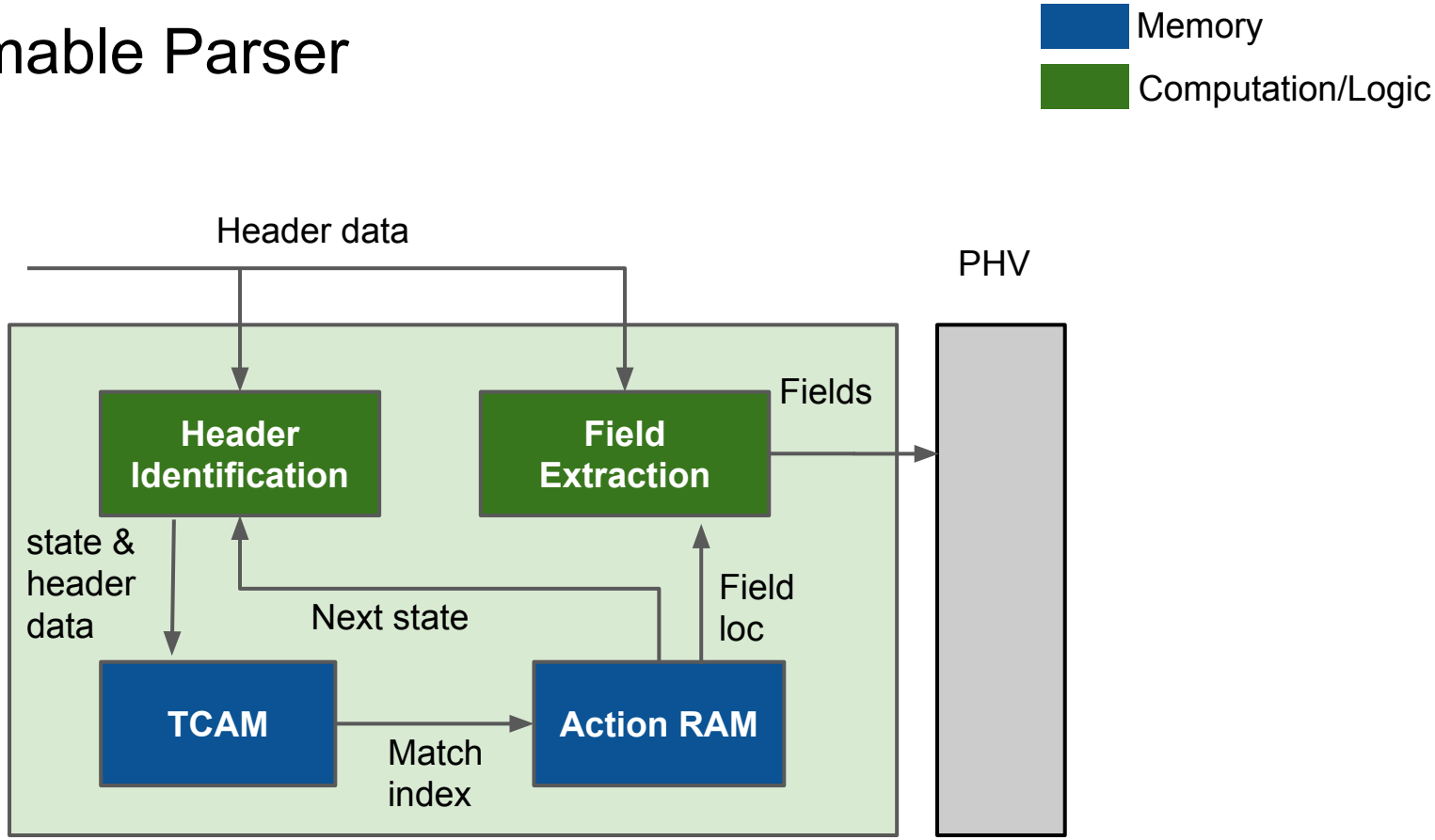
- Headers:

```
header H1 {
 bit<4> A;
 bit<1> B;
}
header H2 {
 bit<2> C;
}
```

- If B = 1, we parse H2.

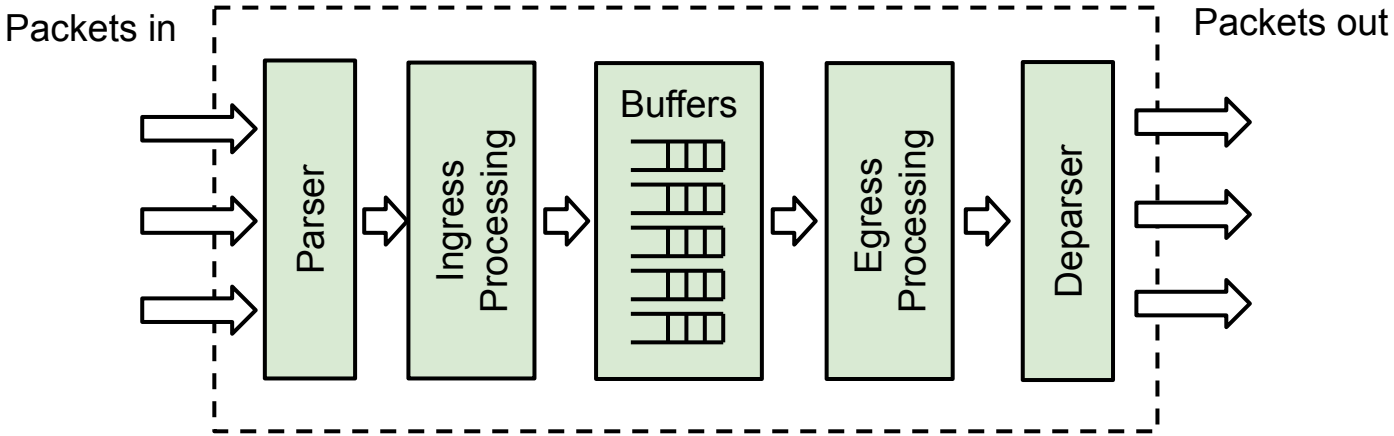


Programmable Parser

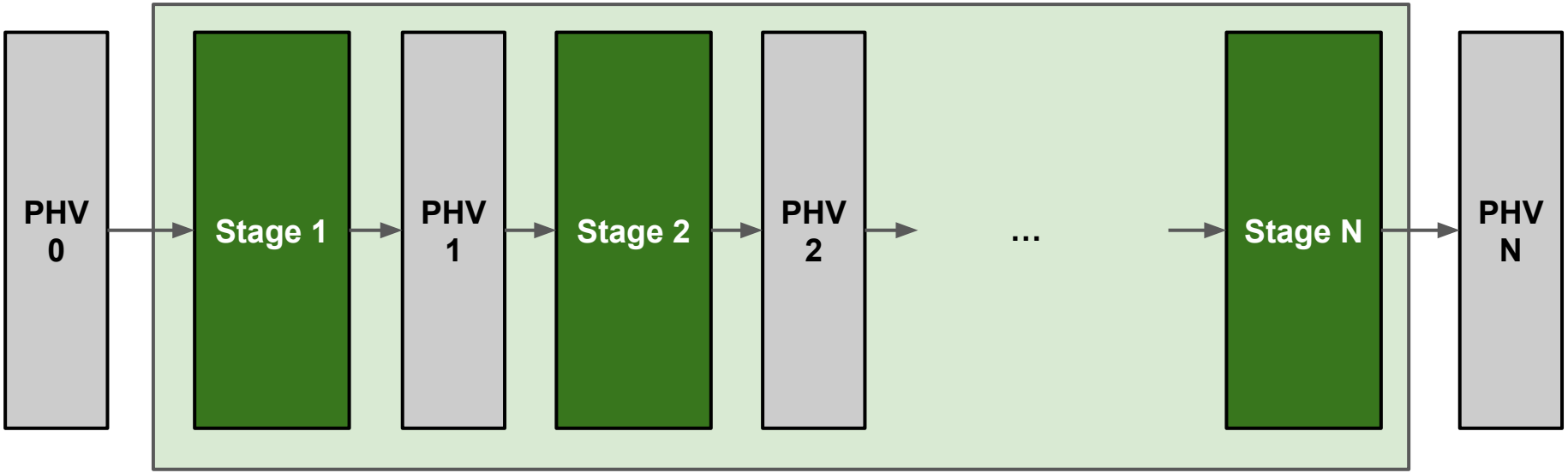


PISA: **P**rotocol-**I**ndependent **S**witch **A**rchitecture

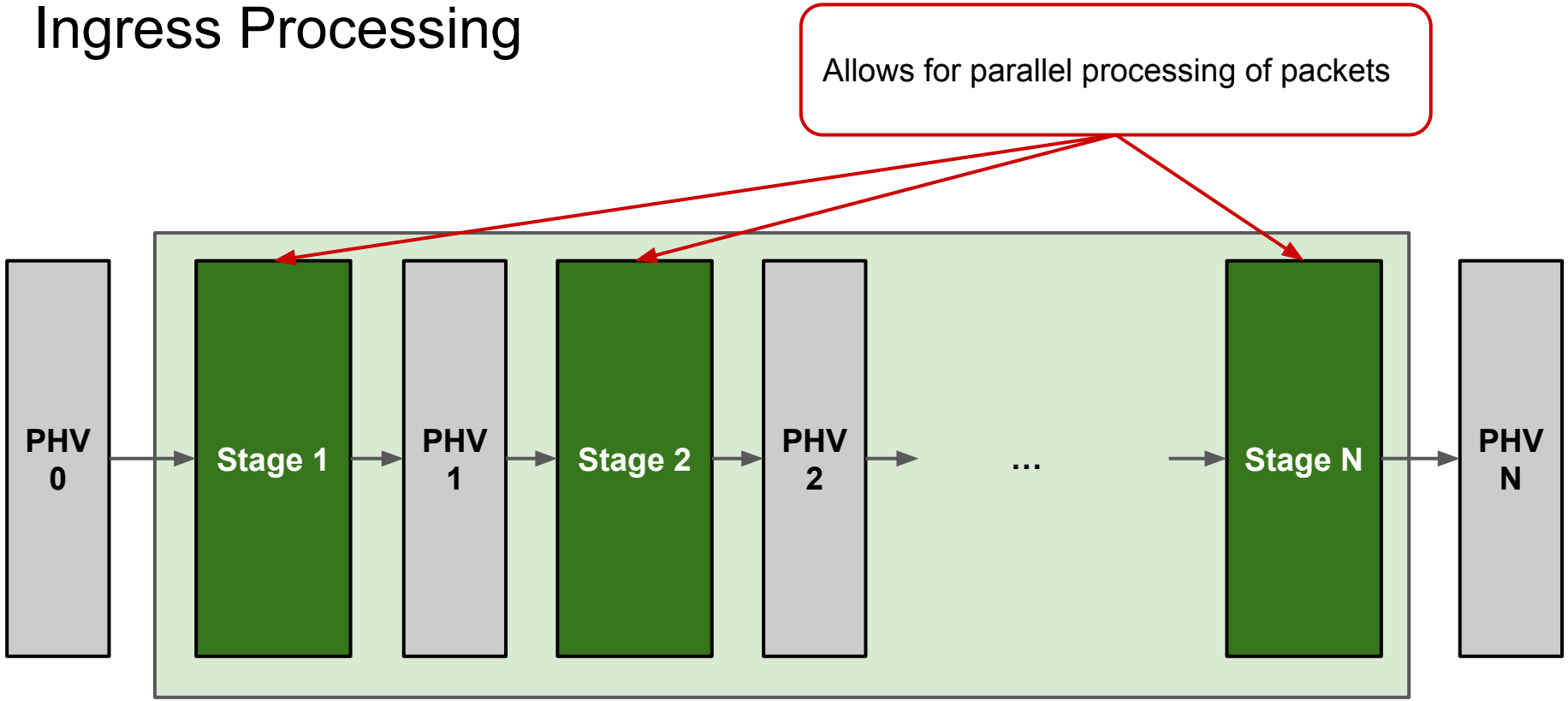
- First academic proposal → Reconfigurable Match Tables (RMT)
- Later evolved and renamed PISA



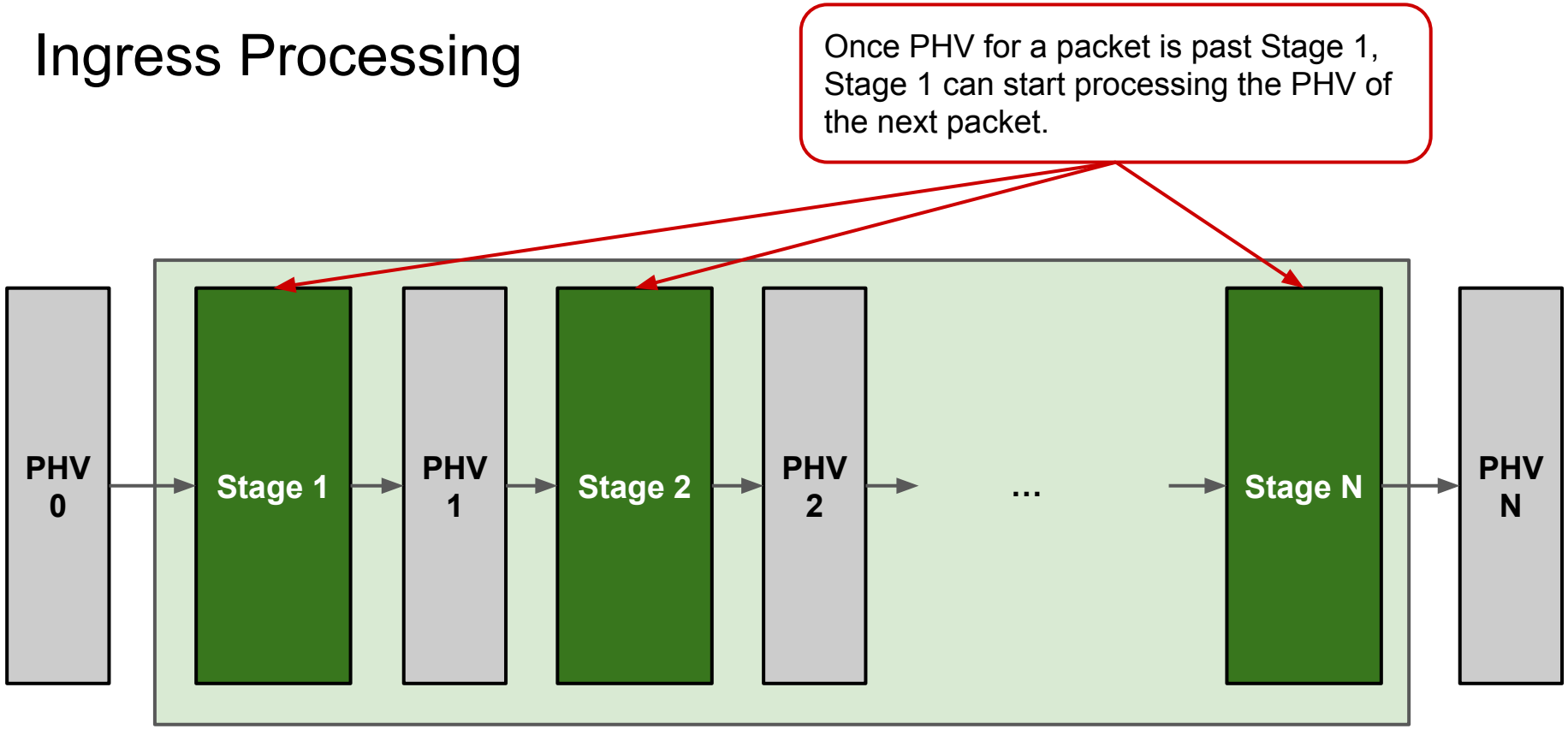
Ingress Processing



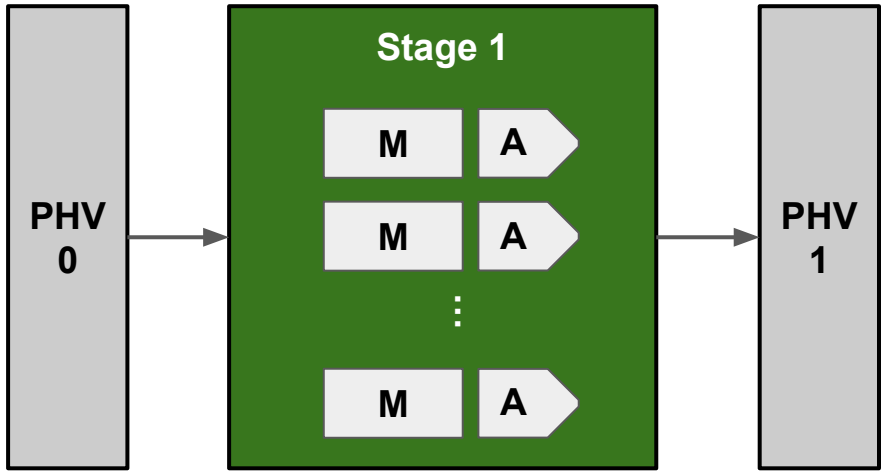
Ingress Processing



Ingress Processing



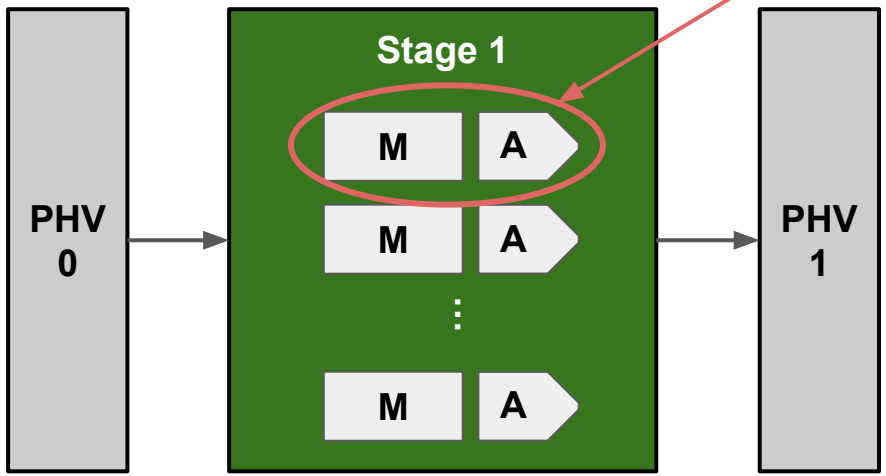
What happens inside a stage?



What happens inside a stage?

The following fours slides are adapted from Changhoon Kim's guest lecture at the "CSE 561: Computer Communication and Networks, Winter 2021" course at University of Washington

What happens inside a stage?

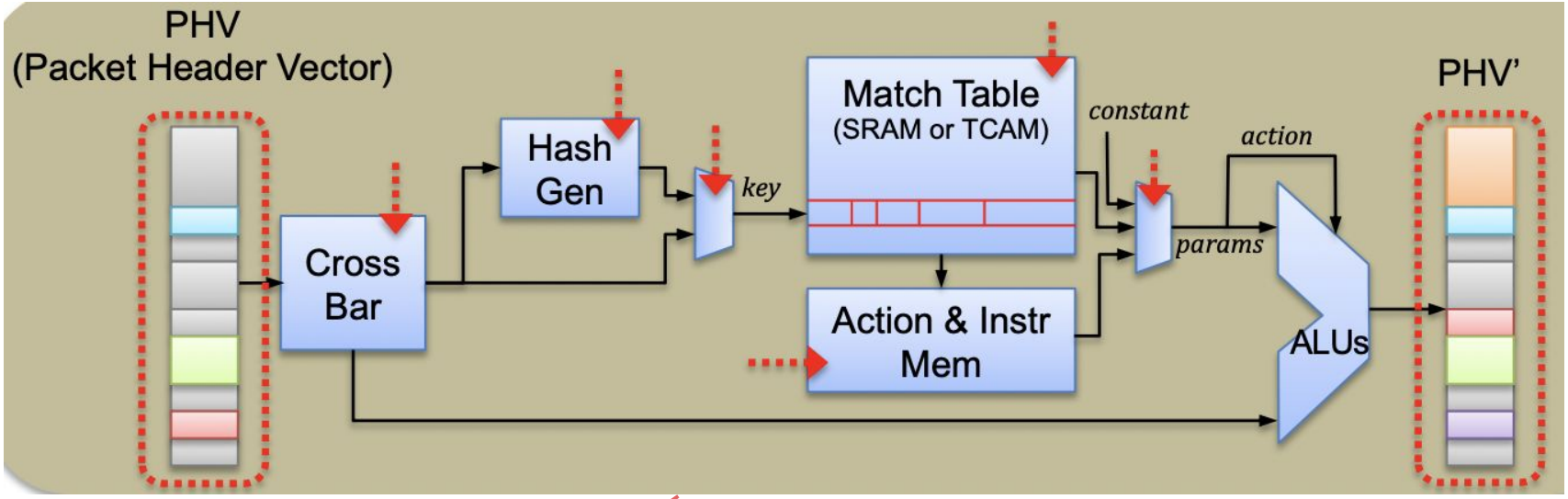


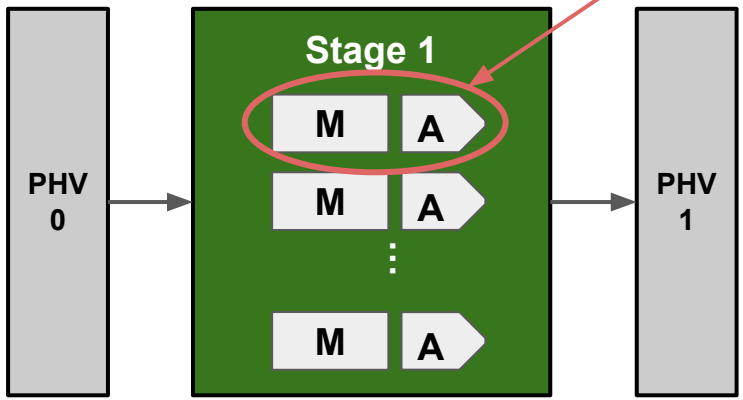
A Match-Action Unit:

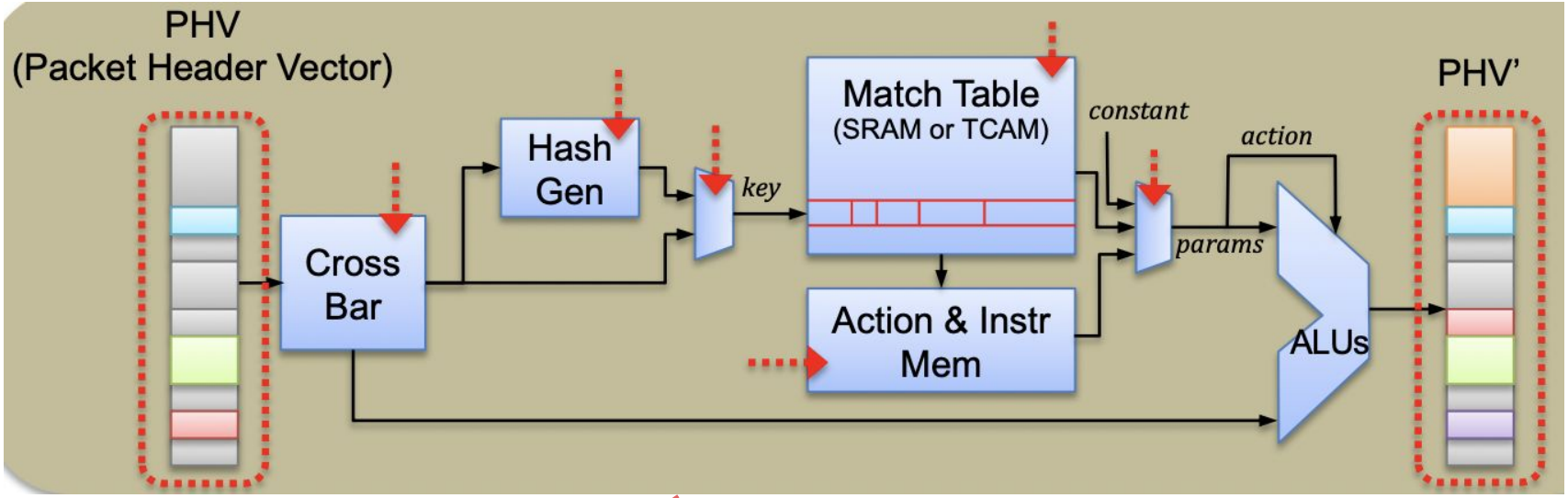
Match: SRAM or TCAM for lookup tables

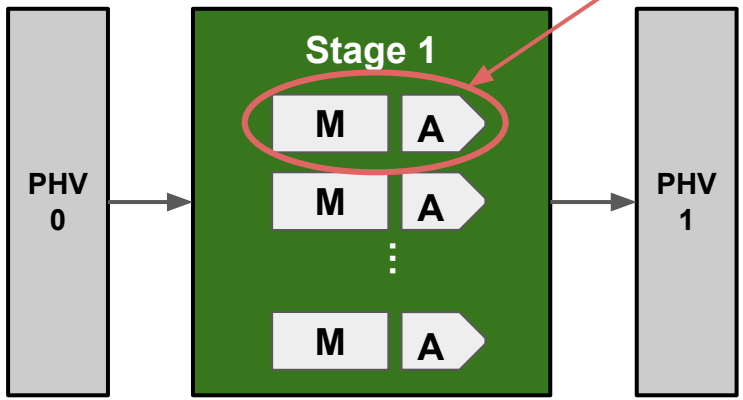
Action: ALUs for standard boolean and arithmetic operations, header modification operations, hashing operations, etc.

A stage is a collection of match-action units.

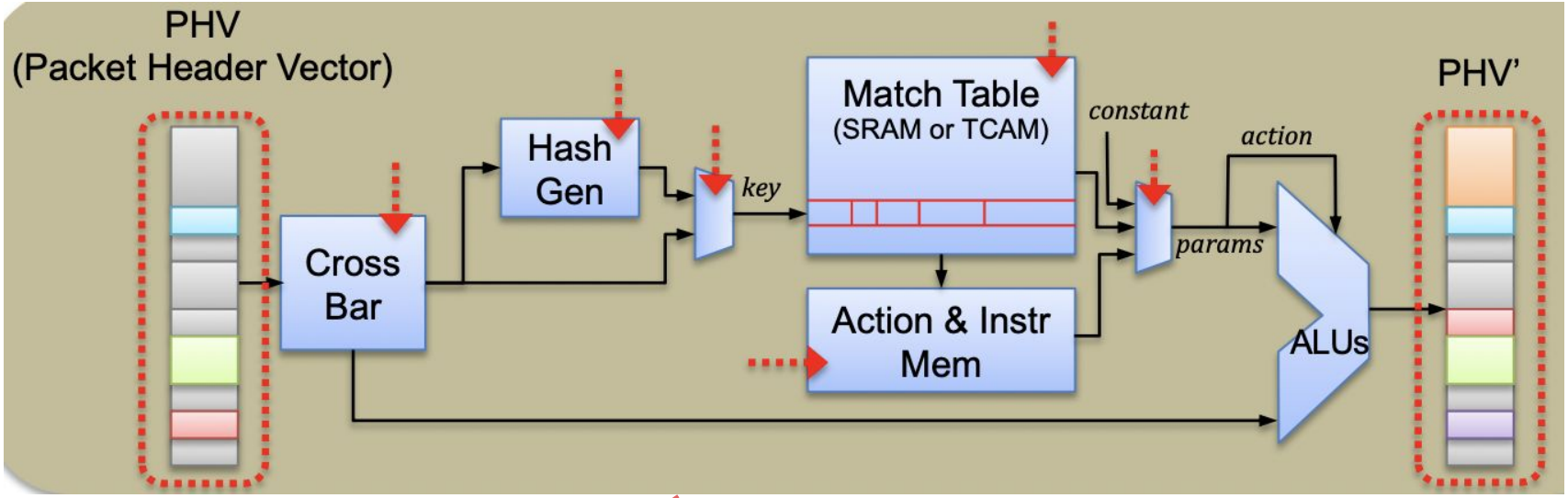

PHV
(Packet Header Vector)
Cross Bar
Hash Gen
key
Match Table
(SRAM or TCAM)
Action & Instr Mem
constant
action
params
ALUs
PHV'

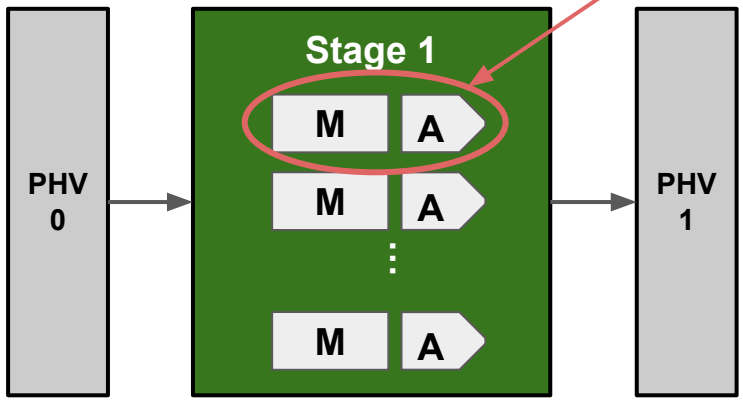

PHV 0
Stage 1
M
A
M
A
M
A
PHV 1





Match-action units are "programmed" by configuring the components marked with red dotted arrows.

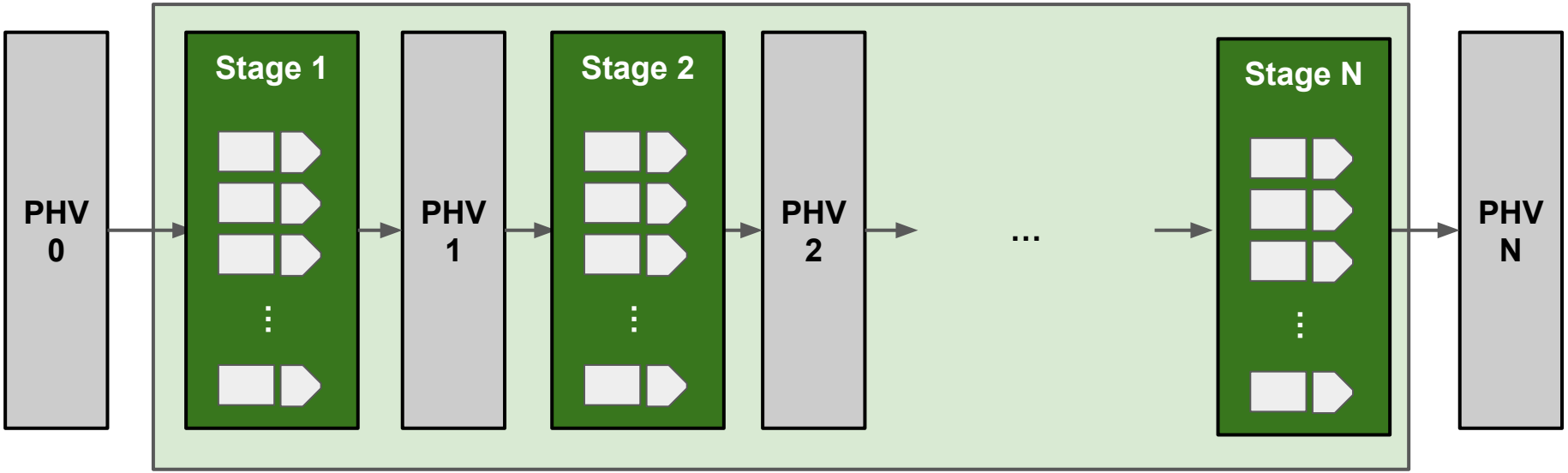




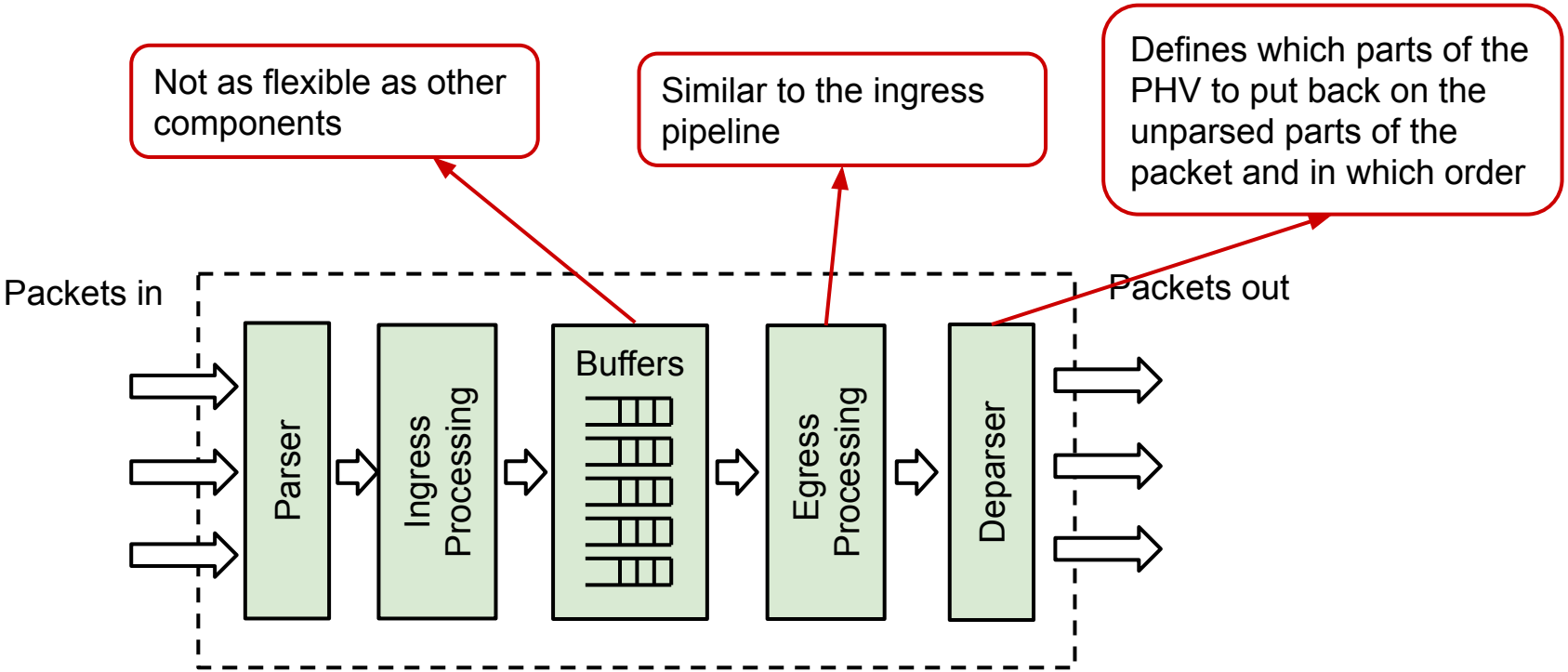
Notice the similarities with the parser.

Stages allow for more general match-action processing.

Ingress Processing



PISA: **P**rotocol-**I**ndependent **S**witch **A**rchitecture



Is PISA practical?

- The RMT paper developed a prototype and evaluated the overheads.
- Barefoot's Tofino switch was the first commercial switching chip with this architecture
  - With multiple "pipes" rather than just one.

PISA - Pros and Cons

- PISA has many advantages
  - It maintains some of the structure of high-speed switching chips
  - The architecture is amenable to high-speed implementation
  - It does a great job of identifying the kind of programmability that is needed in the networking domain (at least in the switch)
  - It was the first practical solution to providing meaningful programmability while maintaining high speed.
  - Paved the way for work on other programmable architectures

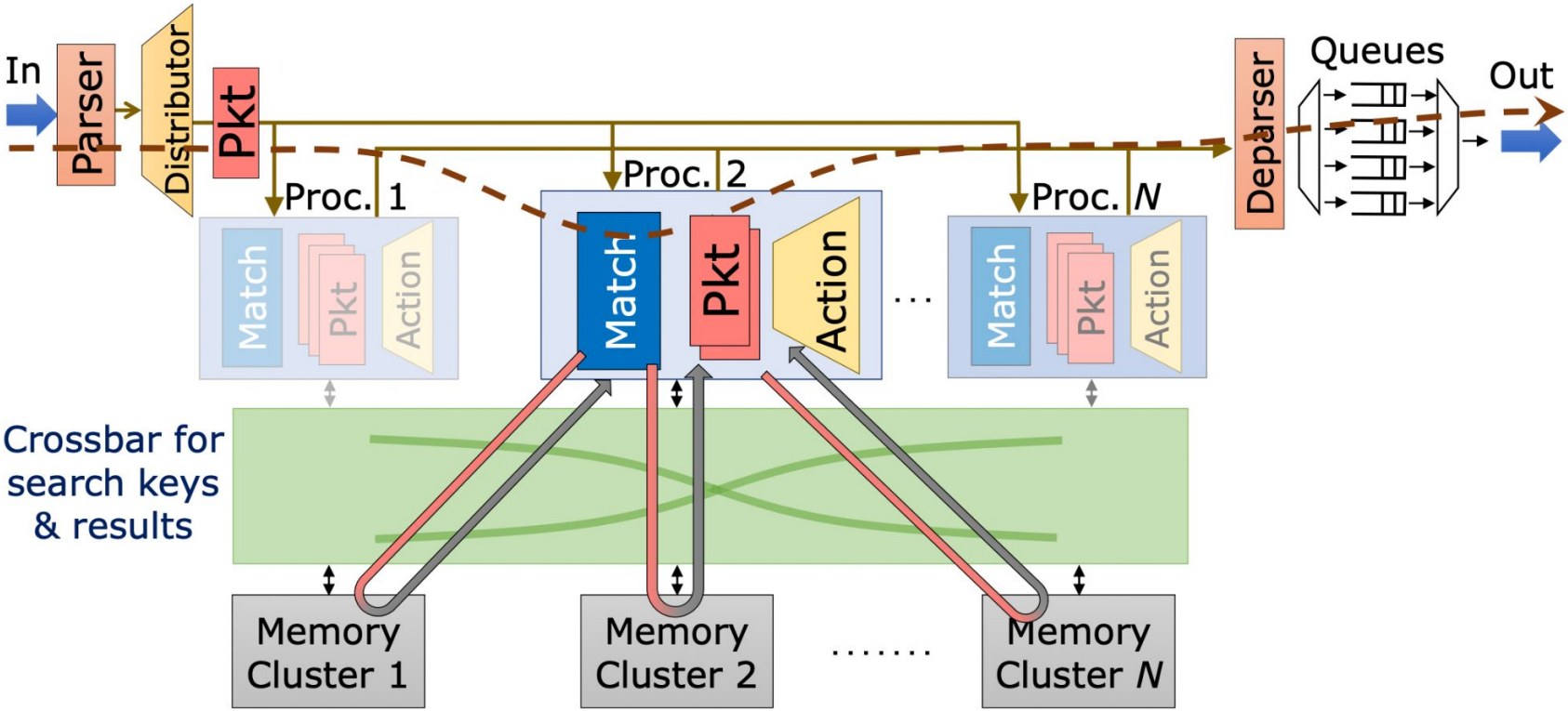
PISA - Pros and Cons

- But, it is not without disadvantages
- Resources can't be shared across stages
- The computational model is quite constrained
  - Feed forward pipeline: can't go back to previous stages
  - For each packet, you can only access the memory in each stage a limited number of times
  - The kinds of computations that the ALUs can do is also limited
- If what you want to do fits within the constraints, it runs at line rate
- If not, it doesn't run at all

Other proposed architectures: dRMT

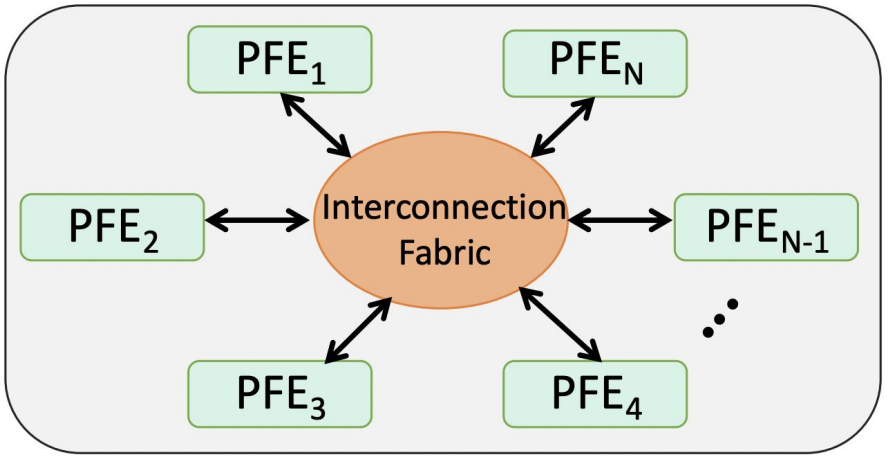
- dRMT = disaggregated RMT
  - Separate the compute and memory resources
  - schedule how packets should share their access to each resource.
- Offers advantages over RMT
  - Can use resources more flexibly and efficiently.
  - Possible to implement more complex logic but at lower performance
    - i.e., performance degradation as opposed to performance cliffs
- But, uses more area and is harder to scale.

Other proposed architectures: dRMT

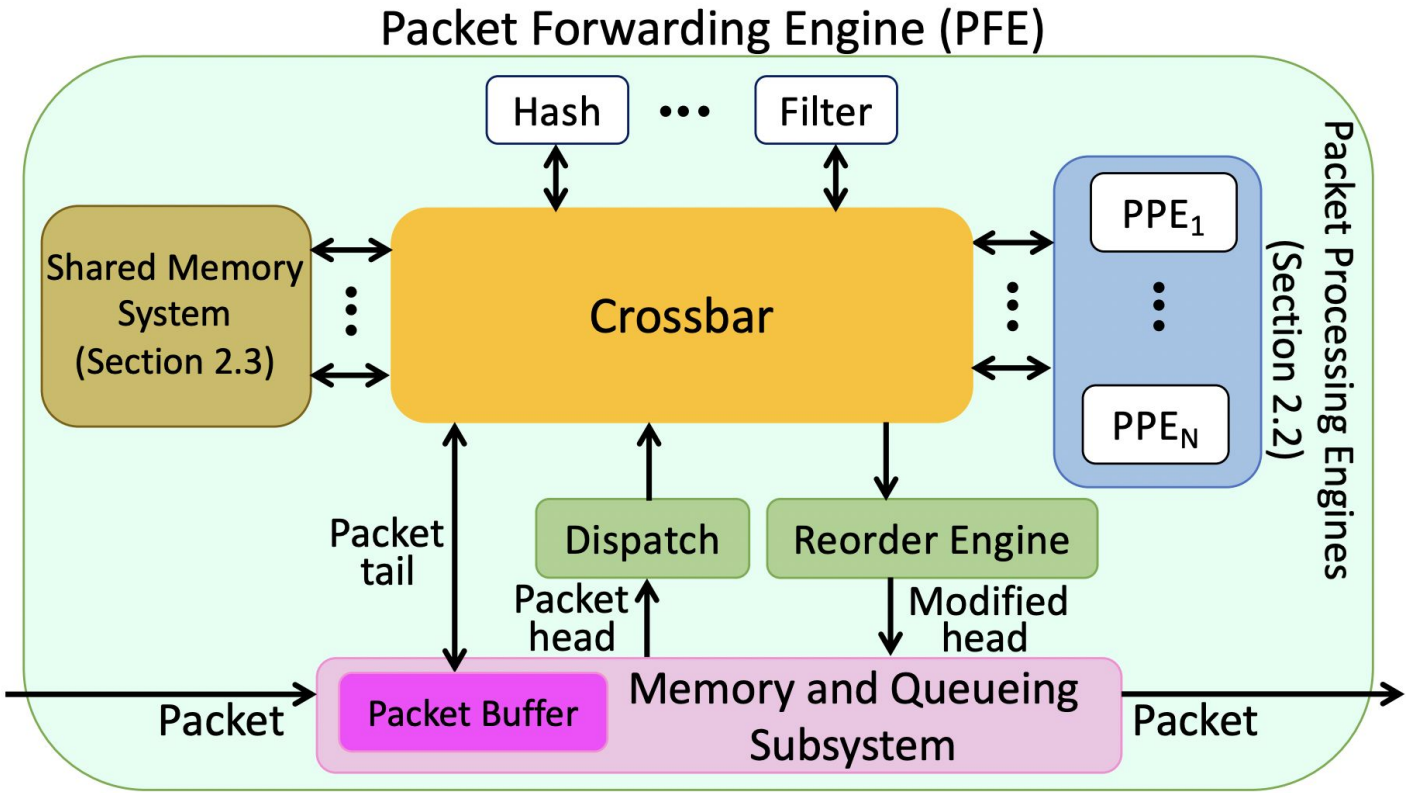


Other proposed architectures: Trio by Juniper Networks

- An interconnected collection of strong packet forwarding engines
  - as opposed to a pipeline.
- Shares similarities with dRMT but takes it further in terms of the flexibility at the architecture level



Other proposed architectures: Trio by Juniper Networks



Paper 1: Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN

- Proposes the RMT architecture, which later evolves into PISA
- Published in 2013
- P4 was published in 2014 as an abstraction for programming these kinds of chips
- Shows that building such a programmable data plane is actually feasible.

Paper 2: Compiling packet programs to reconfigurable switches

- Published in 2015
- Describes how to compile P4-like programs to RMT-like switch data planes
- RMT, P4, and this paper collectively offered an end-to-end solution for programming the data plane.
  - RMT → the underlying hardware
  - P4 → the abstraction
  - This paper → the compiler

Additional Resources

- dRMT (SIGCOMM 2017)
- Trio (SIGCOMM 2022)
- FlexCore (NSDI 2022)
  - Can we (partially) reconfigure the switch data plane without disrupting traffic?
- Menshen (NSDI 2022)
  - Isolation mechanisms for high-speed packet-processing pipelines