PacketScope: Monitoring the Packet Lifecycle Within a Switch

Ross Teixeira (Princeton)

Rob Harrison (United States Military Academy)

Arpit Gupta (UC Santa Barbara)

Jennifer Rexford (Princeton)

Outline

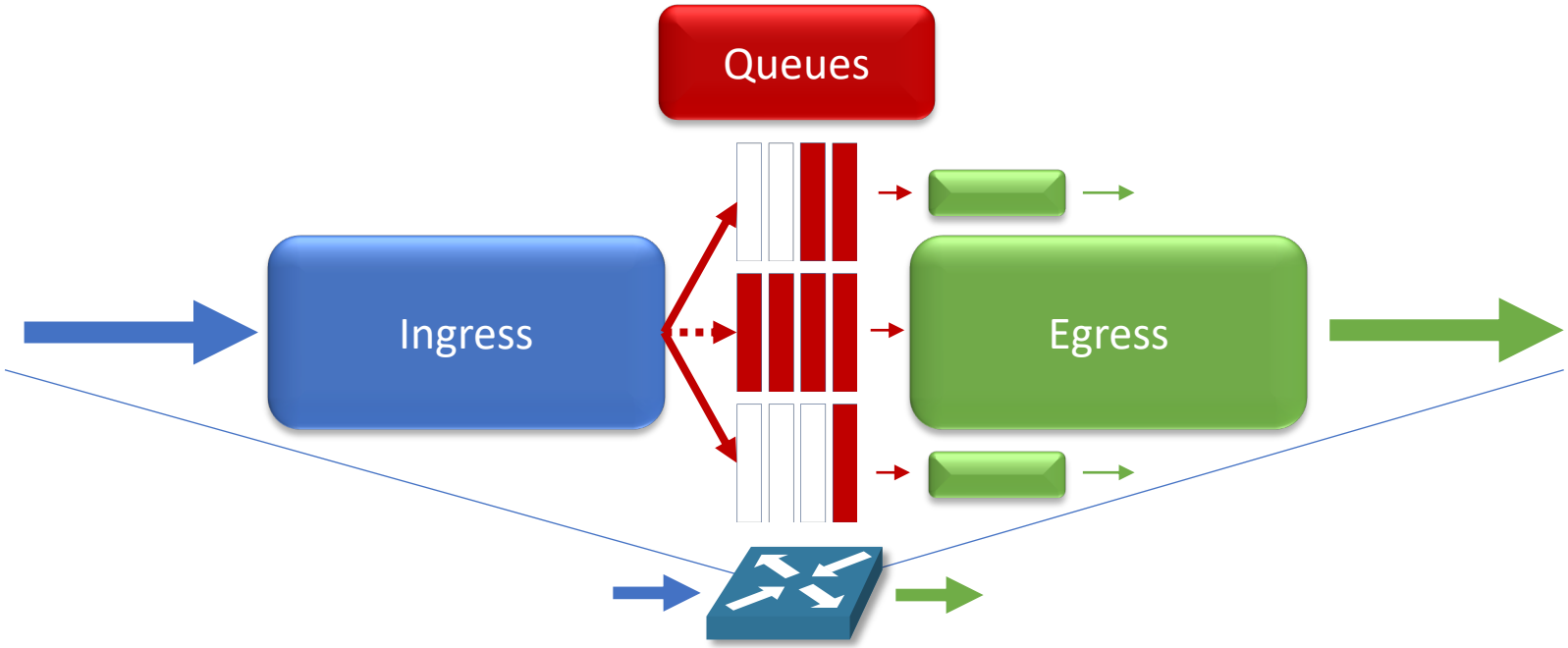
1. Peeking Inside the Switch
2. Packet Lifecycle Query Language
3. Efficient Query Compilation
4. PacketScope Prototype

Outline

1. **Peeking Inside the Switch**
2. Packet Lifecycle Query Language
3. Efficient Query Compilation
4. PacketScope Prototype

What Happens Inside a (Programmable) Switch?

- Packets are modified in the switch
  - Multiple pipelines
- Access Control List (ACL) drops
- Queues cause delays and loss



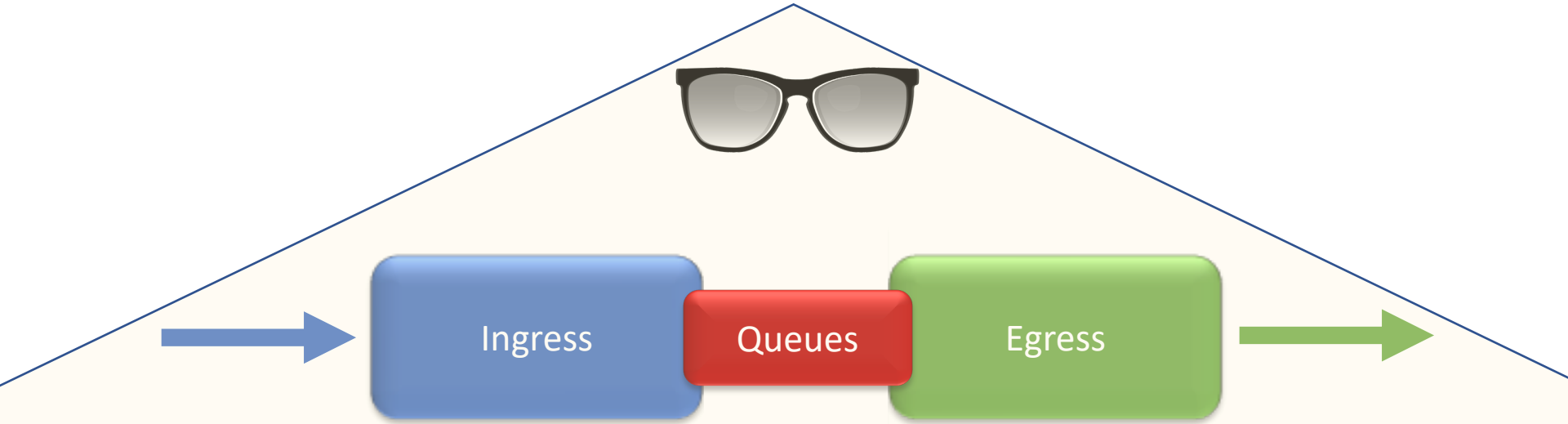
Prior Systems Don't Peek Inside

- Switch monitoring is important
- Want to adapt dataflow monitoring systems
  - map, filter, reduce operators on incoming tuples
- Prior systems only captured packets as they arrived at a switch[1,3]
  - Or only provide queuing delay info[2]

[1] Sonata (SIGCOMM '18), [2] Marple (SIGCOMM '17), [3] Gigascope (SIGMOD '03)

Introducing PacketScope

- Monitoring the packet lifecycle
  - Packet modifications
  - ACL drops
  - Queuing delays/loss



Outline

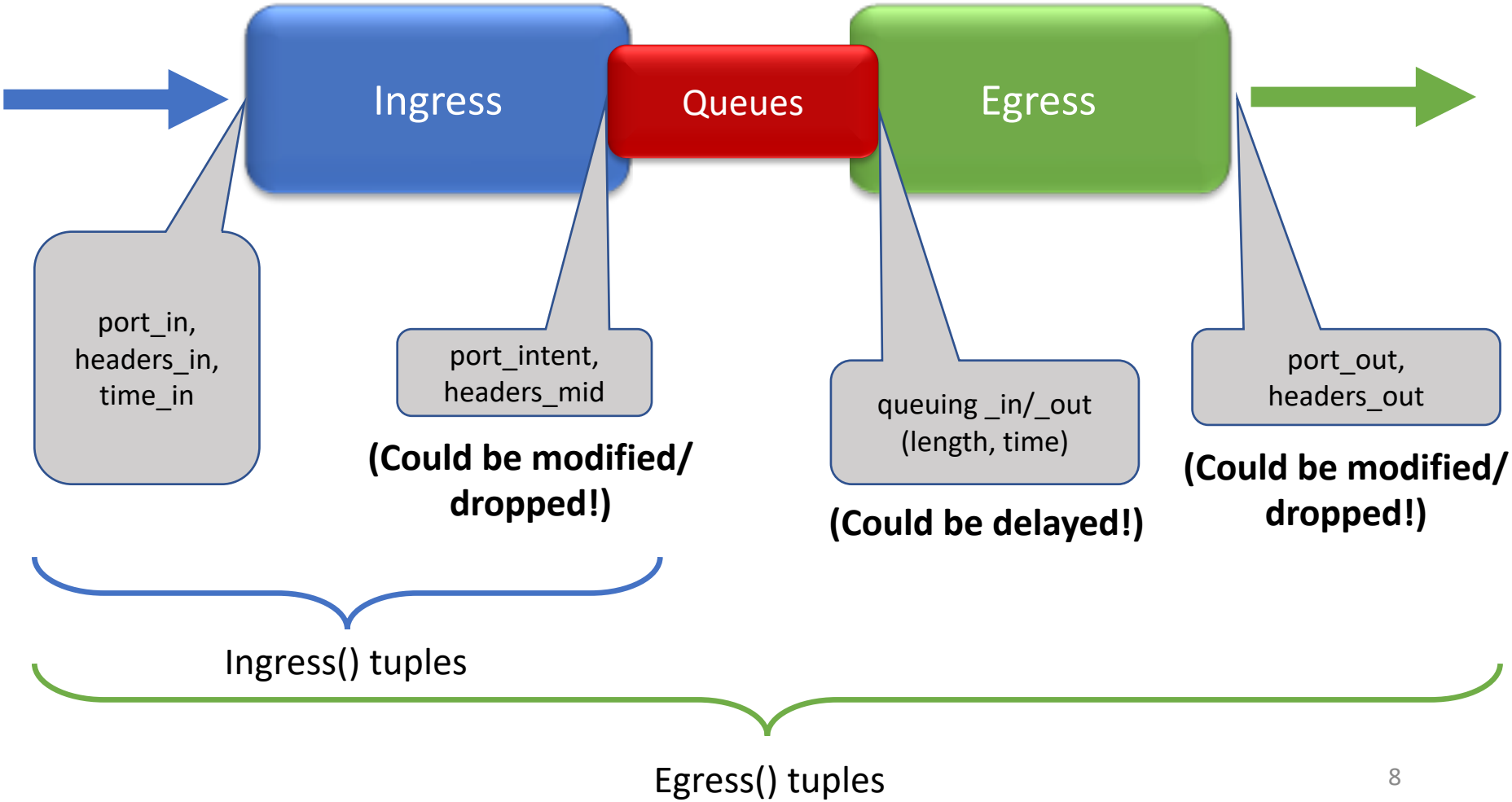
1. Peeking Inside the Switch

2. **Packet Lifecycle Query Language**

3. Efficient Query Compilation

4. PacketScope Prototype

The Life of a Packet



Example Query

- Count un-dropped SSH packets that traverse a NAT

```
1 undropped_SSH_NAT = egress()
2
3
4
```

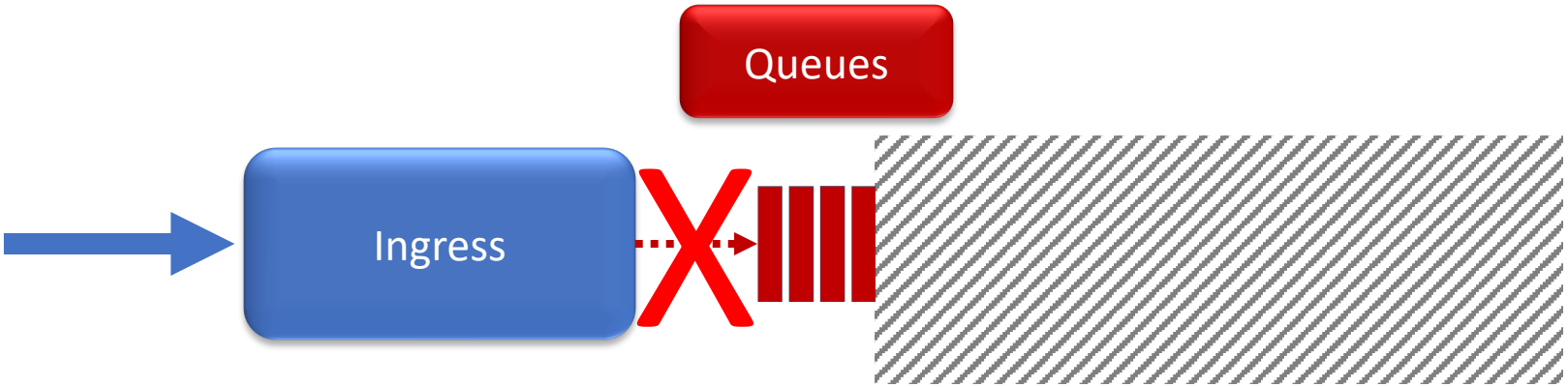
Not Lost

SSH Packets

Crossing a NAT

Not Dropped

How To Track Queuing Loss?

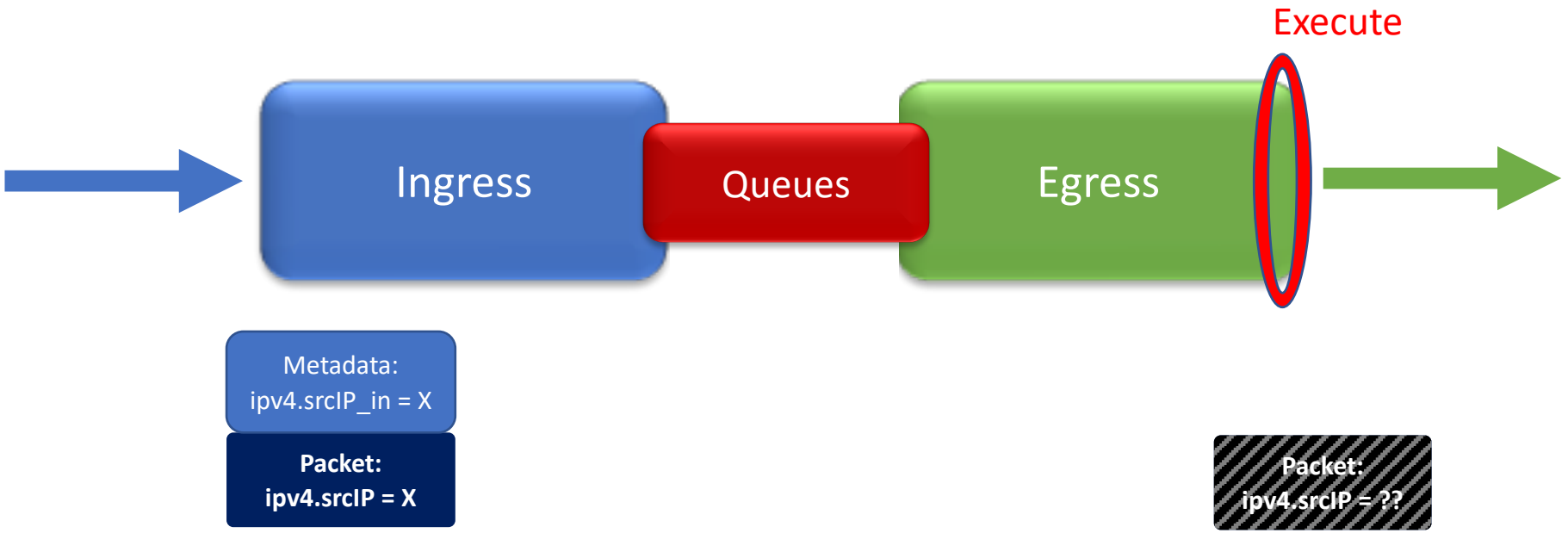


- Loss happens outside ingress/egress processing
  - We can't insert processing to capture packet
- Cannot execute query on individual packet tuples
  - But over time, we can track aggregate counts by keeping state
- *.lost*(*groupby_fields*, *epoch_ms*) operator
  - count packets grouped by *groupby_fields* every *epoch_ms*
    - Arrival time determines epoch placement

Outline

1. Peeking Inside the Switch
2. Packet Lifecycle Query Language
3. **Efficient Query Compilation**
4. PacketScope Prototype

Compilation: "**Tag Little**, Compute Early"

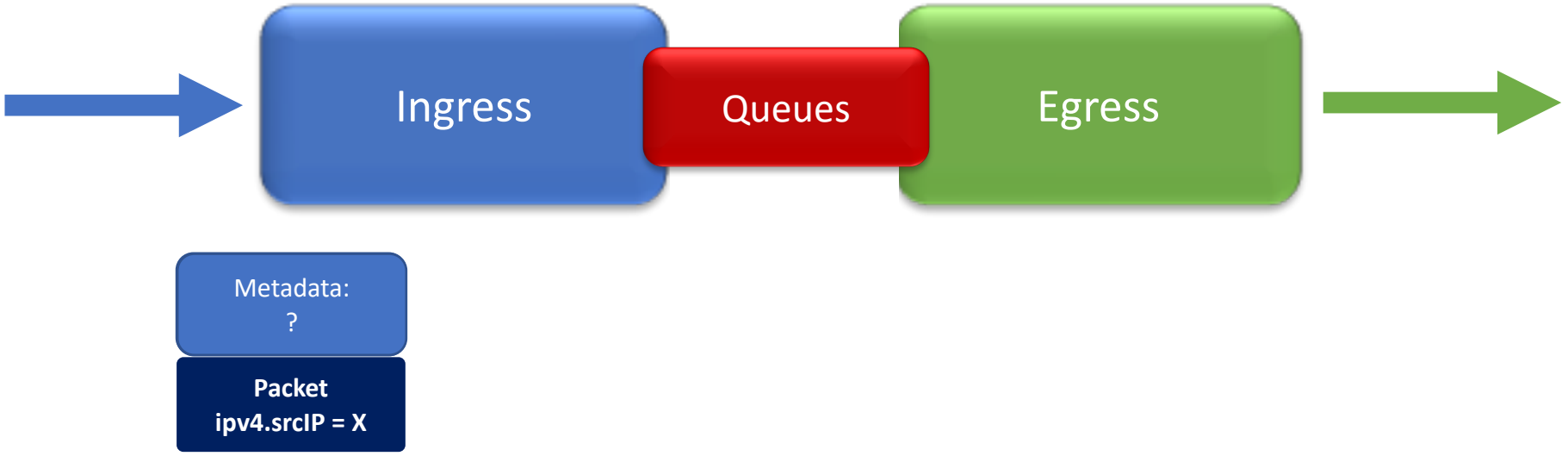


E.g. Queries across ports?

.filter(ipv4.srcIP_in != ipv4.srcIP_out)

A: Tag packet with metadata

Compilation:“Tag Little, **Compute Early**”

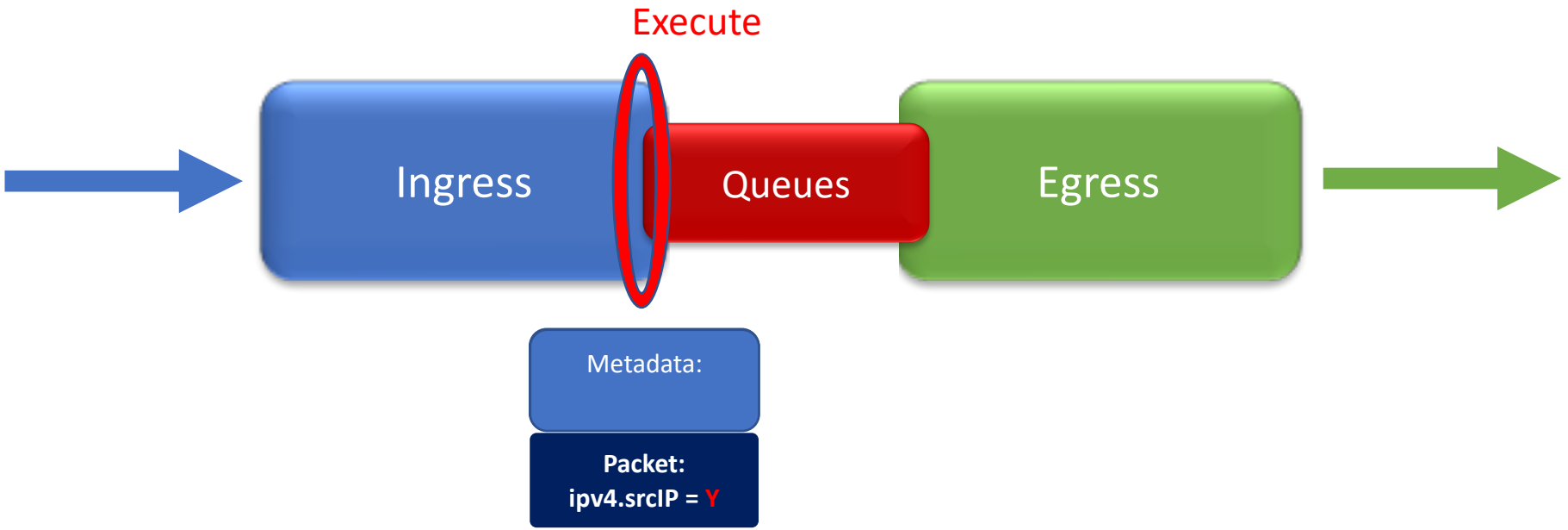


Where to place computation?

.filter(ipv4.srcIP_in != ipv4.srcIP_**mid**)…

A: As early as possible!

Compilation:“Tag Little, **Compute Early**”



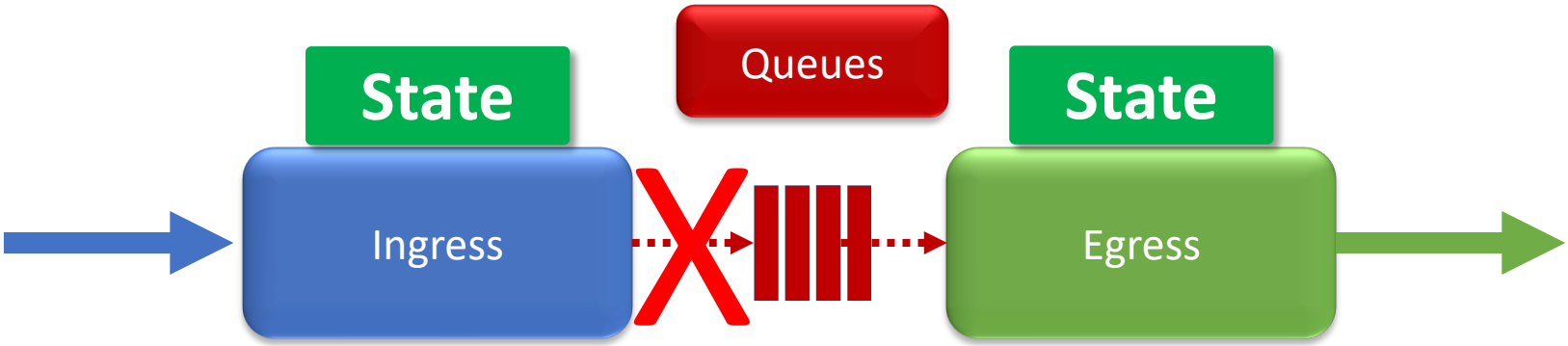
Where to place computation?

.filter(ipv4.srcIP_in != ipv4.srcIP_**mid**)…

A: As early as possible!

- Metadata can be reused for future processing.

How To Compile Lost Operator?



- *.lost([ipv4.srcIP], 10ms)*

- Compile as a *join* of two queries:
  - Count by *ipv4.srcIP* on ingress
  - Count by *ipv4.srcIP* on egress
- Report difference every *10ms of* packet arrival times

- Gory details in paper

Outline

1. Peeking Inside the Switch
2. Packet Lifecycle Query Language
3. Efficient Query Compilation
4. **PacketScope Prototype**

PacketScope Prototype

- We built a prototype[1] in Python and P4 with:
  - Support for packet modifications, queuing delays
  - Tag little, compute early compilation
- We also built a queuing loss query prototype
  - Uses the BMv2 software model
- More details and future work in paper

[1] As an extension to Sonata (SIGCOMM ‘18)

Conclusion

- PacketScope is a network telemetry system
  - Using a dataflow programming model (map, filter, reduce)
  - That supports queries on the full packet lifecycle:
    - Packet modifications
    - ACL drops
    - Queuing delays/loss
  - And compiles efficiently to programmable switches