Georgia Institute of Technology

Dr. Nick Feamster
Associate Professor

Software Defined Networking

In this course, you will learn about software defined networking and how it is changing the way communications networks are managed, maintained, and secured.

School of Computer Science

This Lesson: Network Assembly

- Motivation
  - Why do we need a network assembly language?
  - What can a network assembly language do?
- Example: NetASM
  - Overview
  - Brief discussion of assembly language primitives
  - Open issues

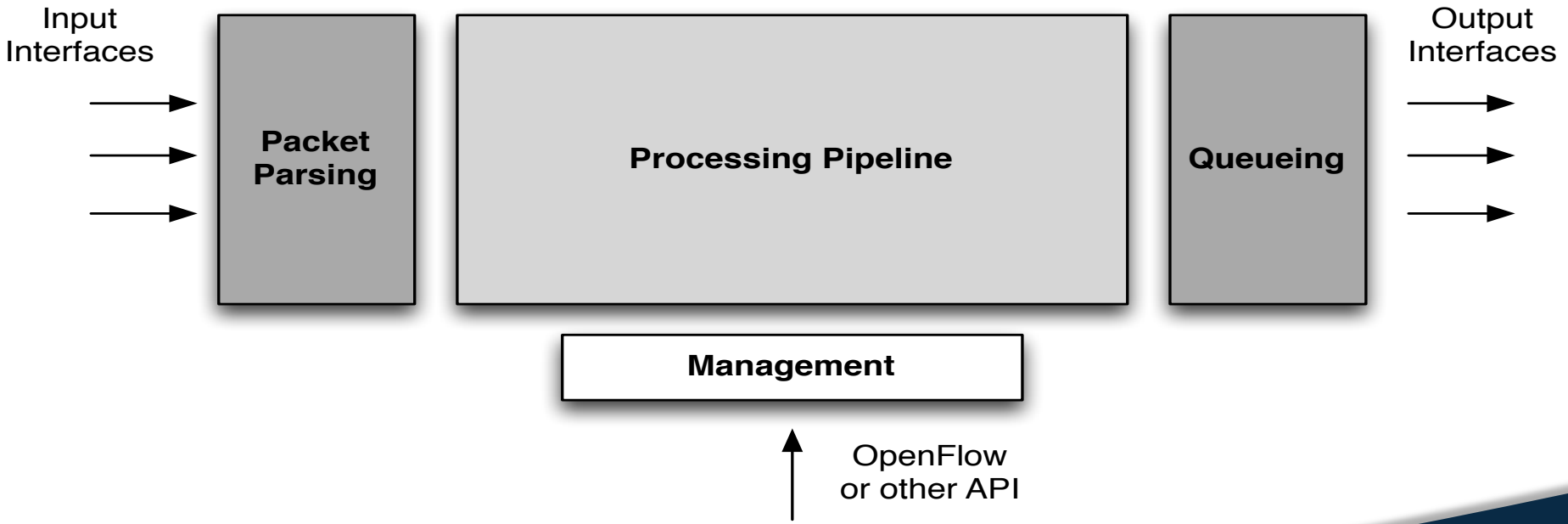
Motivation

- OpenFlow's design was motivated by the underlying device layout
  - Controller is limited in supporting new functions not supported by OpenFlow
- New chipsets (RMT, FlexPipe) are adding data plane functions
- New languages are specifying data-plane at a high level
- **What's in between?**

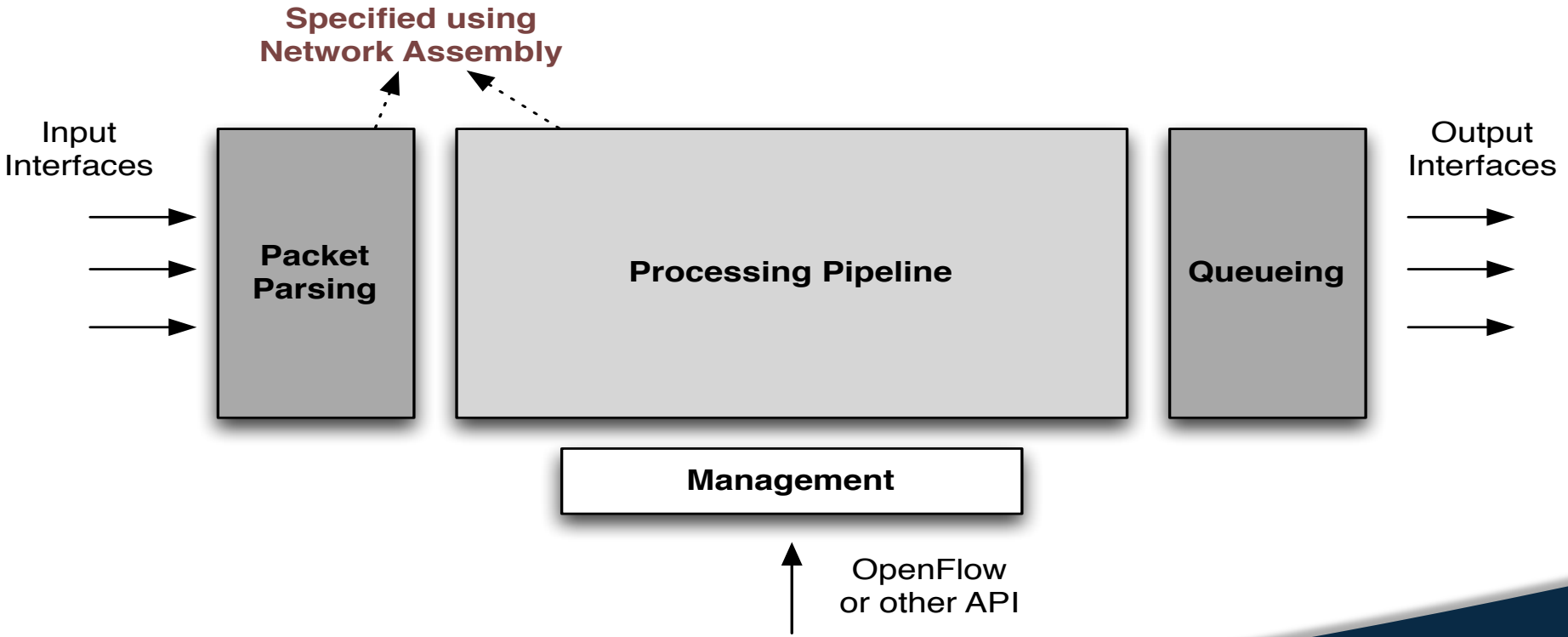
Need for Network Assembly

- A **low-level programming language** for programmable network devices
- Provides a **1-to-1 correspondence** with the underlying hardware
- Uses **well-defined constructs** to define low-level packet operations
- Enables writing highly optimized network programs

Common Hardware Architecture



Want to Specify Parsing and Processing



Programming in Network Assembly

- Explicitly describe the processing pipeline in the assembly program
  - Series of instructions as specified by sequence of operations
  - The constructs form an acyclic directed graph

- A parse graph gives a semantic meaning to the bit locations in the packet header
  - Specified by the user at the time of writing the assembly program

Three Types of Instructions

- **Initialization:** to create state elements (like tables and registers)
- **Topology:** to define how the packet is traversed and processed in the data plane
- **Control:** to provide an external control to populate the states (i.e., over OpenFlow or other interfaces)

Locally Contained Applications

- Provide the hardware pipeline the ability to update its states (registers, tables) locally
  - without referring to the controller
- Need a new construct “update”
  - to update the internal states (registers, tables)
  - has the opposite semantics to that of modify construct which updates the header/metadata
- Examples: MAC learner

Protocol Independence: Compile from Different Languages

- NetKat
- P4
- OpenState
- OpenFlow 1.x
- Flowlog

Compiler Can Optimize using Assembler

- Table
  - Composition
  - Decomposition
- Reordering
- Optimizations based on traffic profiles

Target Independence: Assemble for Different Targets

- FPGA
- Click
- NPU
- GPU
- Open vSwitch
- Open Data Plane

Key NetASM Instructions

- MKT:
  - MKT (Tbl, Tbl)
  - Initialization instruction
  - Takes two arguments (a dynamic table specification and static table with default values) and creates a new table.

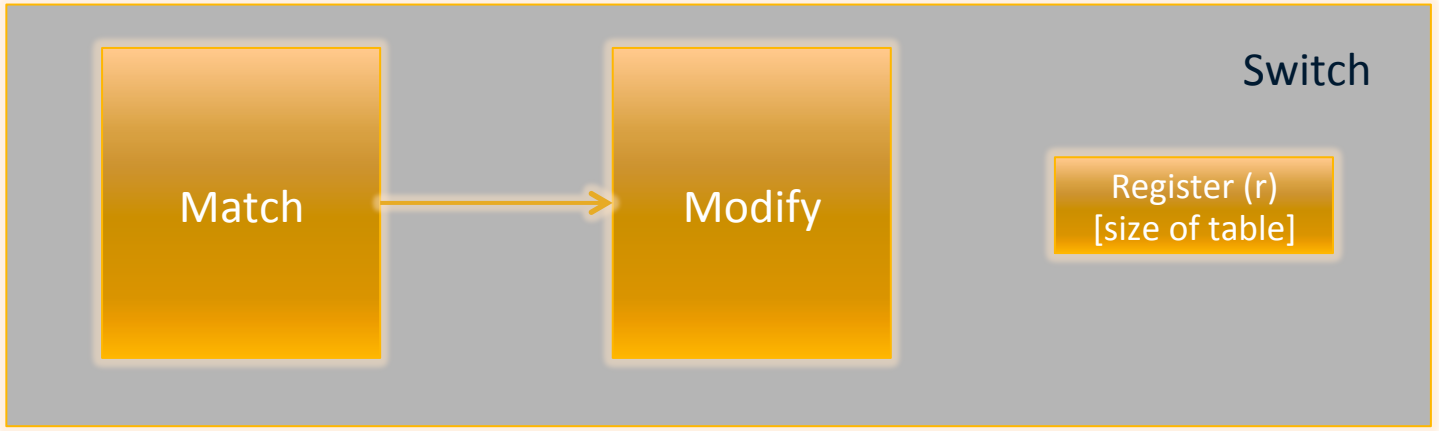
- BRTF:
  - BRTF (Tbl, Fld, Lbl)
  - Topology instruction
  - Branch to a label if the header matches with the contents of the table and set the given header field with the matched index,
    else, move to next instruction.

NetASM Instructions

- DRP:
  - DRP
  - Topology instruction
  - Marks the packet for drop.

- WRT:
  - WRT (Tbl, Ptrn, Val)
  - Control Instruction
  - Write the table with pattern at index value.

Example: Stateful MAC Learning



- Two tables: Match and modify
  - Match: matches on dst MAC, outputs index
  - Modify: modifies output port depending on index

Other Notable Aspects

- NetASM is in Haskell: Its semantics have provable, verifiable properties

- Each assembly instruction could be associated with a “cost” to allow a compiler to make intelligent compilation decisions

Conclusion

- Programmable hardware allows the data plane to evolve
  - In turn, this frees the SDN control plane from current constraints (no longer has to be OpenFlow)
- Have a high-level language specifying packet processing (P4), and an assembler (NetASM)
- Need a compiler!