Internals of the Intel Tofino Compiler

Glen Gibb, Intel

Introduction

This presentation provides an overview of the Intel Tofino compiler's internals.

I'll be primarily focusing on four major topics:

- Custom IR nodes
- PHVInfo – a data collection/query module
- Resource allocation (PHV and table)
- Backtracking

Company Overview

Intel produces the P4-programmable Tofino switch ASICs and the Intel IPU E2100.

Intel is best-known for their microprocessors, with their Xeon, Core, and Atom product lines powering many of today's computers.

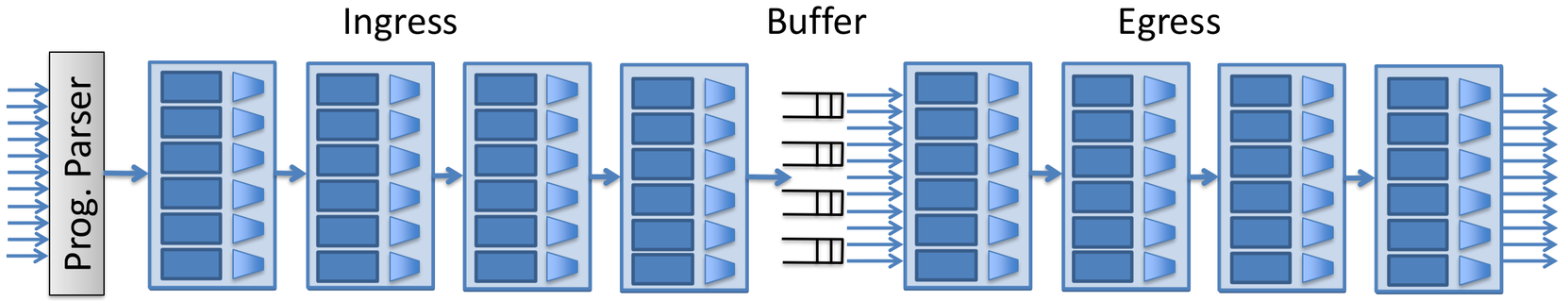
Intel is open sourcing the Tofino compiler

(+ most of the software stack)

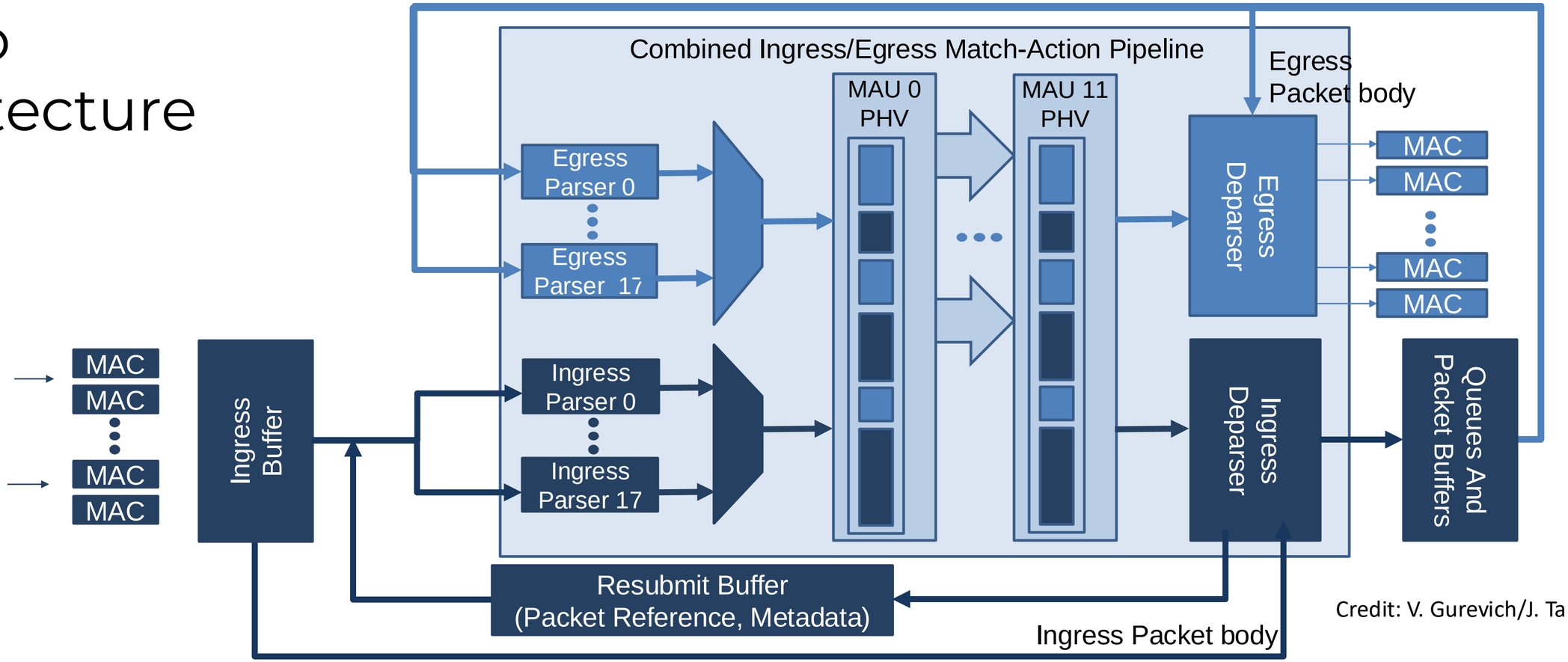
Outline

- Tofino architecture
- Midend/backend tour
- Tofino IR nodes
- PhvInfo / PHV::Field objects
- Resource allocation
- Backtracking

PISA: Protocol-Independent Switch Architecture



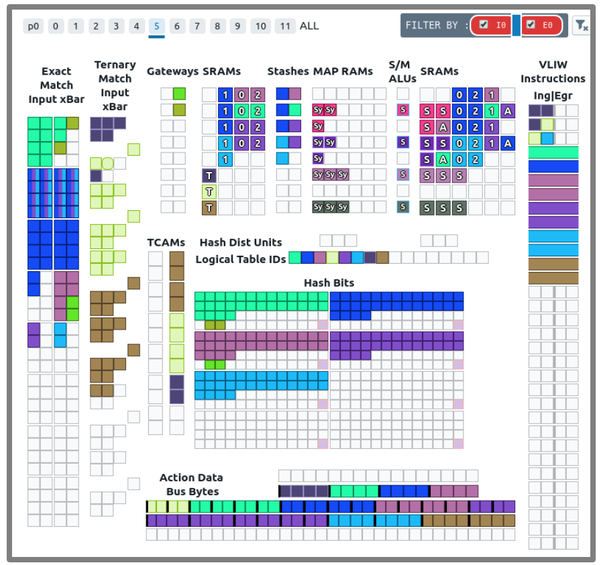
Tofino Architecture



Credit: V. Gurevich/J. Tan

Tofino architecture

- Match-Action Unit (MAU)
  - Implement multiple logical tables
  - Shared between ingress/egress
  - Allocatable resources
    - SRAMs
    - TCAMs
    - ALUs
    - Xbars
  - MAUs per pipe:
    - Tofino 1: 12
    - Tofino 2: 20



- Packet header vector (PHV)
  - Wiiiiide (4kb+)
  - 8b / 16b / 32b containers
  - Shared between ingress/egress
  - Passed between parser/MAU stages/deparser

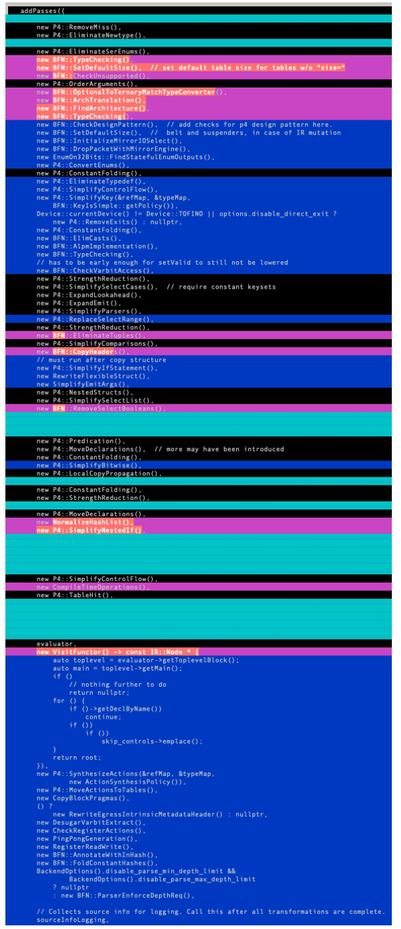
- Packet occupancy vector (POV)
  - Bit vector indicating headers
  - Stored in PHV

Midend/backend tour

MidEnd:

- Similarity to PsaSwitchMidEnd

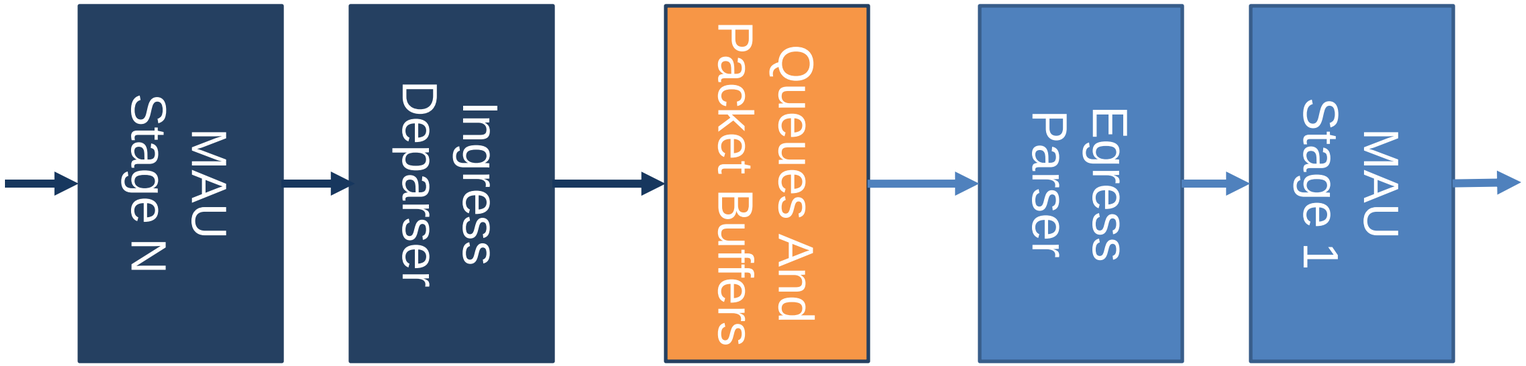
- Some specializations (P4:: → BFN::)
- Design patterns: adherence + recommendations
- Architecture translation (TNA IR nodes)
- Normalization (tna/t2na/v1model)
- Simplifications (cast elimination, control rewrites, …)
- Varbit: verify usage / desugar
- …



Midend/backend tour (cont.)

BridgedPacking:

- Runs between MidEnd and BackEnd
- Packs “bridged” metadata
  - Ingress → egress metadata
  - Prepended to packet



- Translation to backend IR nodes (MidEnd only introduces subset)

Midend/backend tour (cont.)

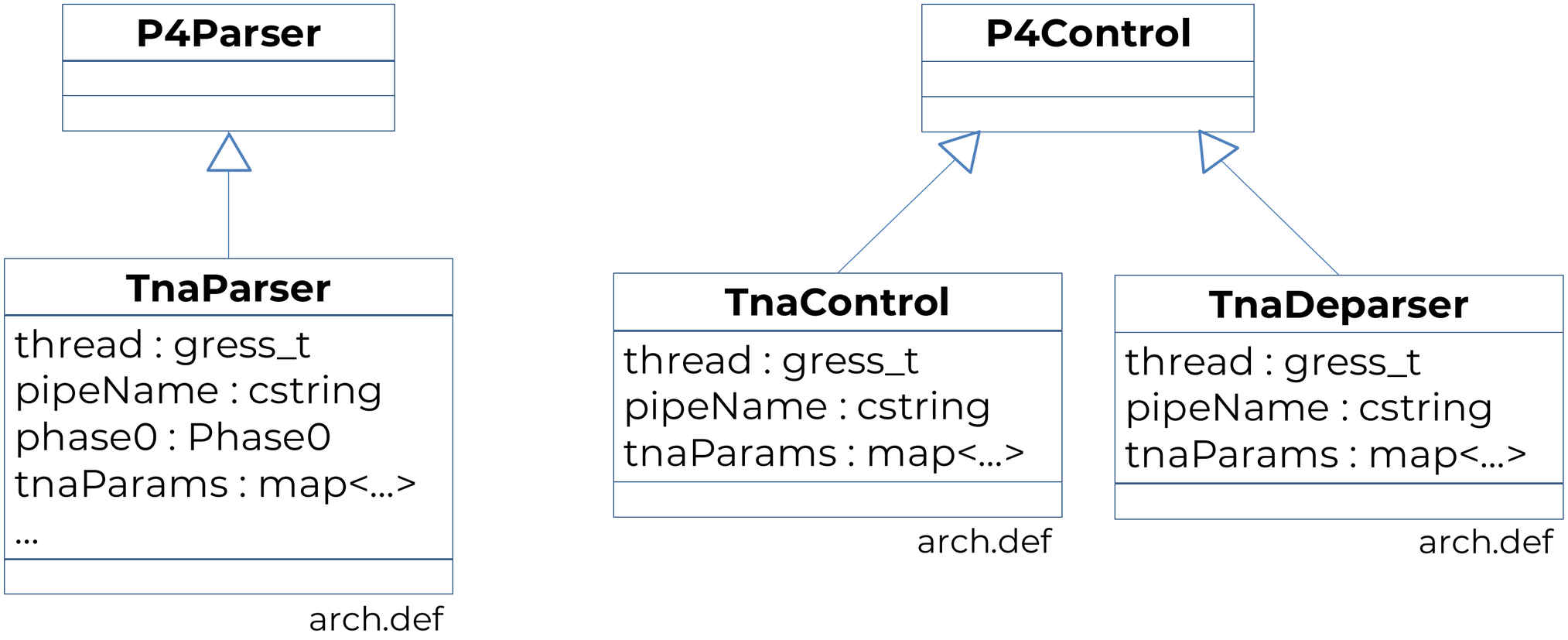
BackEnd:

- Instruction selection
- PHV allocation
- Table allocation

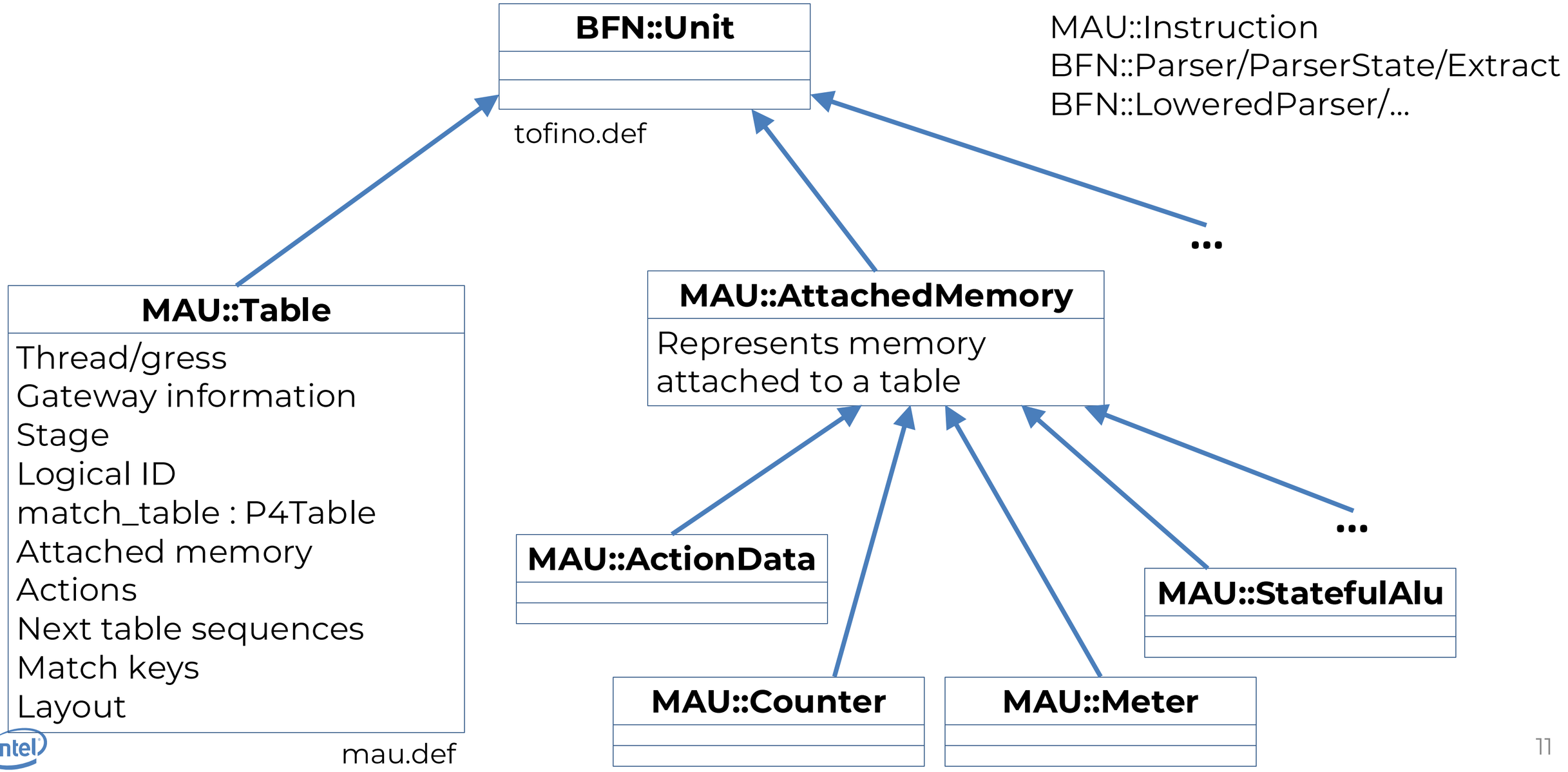
- Smaller features:
  - Aliasing (detecting two fields are equivalent → share allocation)
  - Metadata initialization (optional)
  - Pragma collection/application
  - Parser IR lowering
  - …

Tofino IR nodes

- Custom Tofino IR nodes replace some open-source IR nodes
- Reason: better represent underlying hardware



Tofino IR nodes (cont.)



PhvInfo + PHV::Field objects

PhvInfo

- Information for PHV-backed storage (fields + metadata)
- Iterators
- Query interface (e.g., field(), bits_allocated(), …)
- Populated by CollectPhvInfo pass

PHV::Field

- Basic info for a field (name, ID, size, offset within header, properties, …)
- Alignment constraints
- PHV container allocation (if any)
- Alias information
- Higher-order functions to iterate over and process allocations

Resource allocation

PHV alloc

```
collect_constraints();
vector<Cluster> clusters = create_clusters();

for (auto cluster : clusters) {
    for (auto container_group : container_groups) {
        for (auto slice : cluster) {
            vector<...> possible_allocation = try_alloc(slice, container_group);
            if (possible_allocation.size() == 0)
                break;
            Allocation best_allocation = best_allocation(possible_allocation);
            place_slice(best_allocation);
        } } }

if (unplaced_clusters())
    ::error("Not all clusters placed");
```

PHV partitioned into groups
MAU instruction:

- Operands + target field: same group
- Operands: same alignment

Try in each container at each valid alignment

Multiple tries before failing. Several different search/scoring strategies.

Resource allocation (cont.)

Table alloc

```
while (true) {
    set<Table> possible_tables = placeable_tables_due_to_dependencies();
    if (possible_tables.empty())
        break;
    vector<Allocation> possible_allocation;
    for (possible_table : possible_tables) {
        possible_allocation.push_back(find_allocation(possible_table));
    }
    Allocation best_allocation = best_allocation(possible_allocations);
    place_table(best_allocation);  // Locks in a decision
}

if (unplaced_tables())
    ::error("Not all tables placed");
```

Prefer tables in longer sequences
Prefer larger tables
Pack into earlier stages

Actual process more complex. Does internal and external backtracking.

Backtracking

- Local within table placement:
  - Record partial placements
  - If table stage > max MAU stage:
    Return to earlier point and try different choice
  - Limit on backtrack attempts

- Global from table placement to PHV allocation:
  - Table placement info fed back to PHV alloc (via MauBacktrack object)
  - PHV alloc redone – uses backtrack info

Summary

- Tofino compiler will be open sourced soon
- Some custom IR nodes
- PhvInfo: collates information about fields/metadata, query iface
- PHV::Field: information about a single field/metadata
- PHV allocation / table allocation: iterative algorithms with retry
- Backtracking: used extensively in allocation to reach a solution

Acknowledgements

Huge team effort

Thank you to all who have contributed

Thank You