



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.

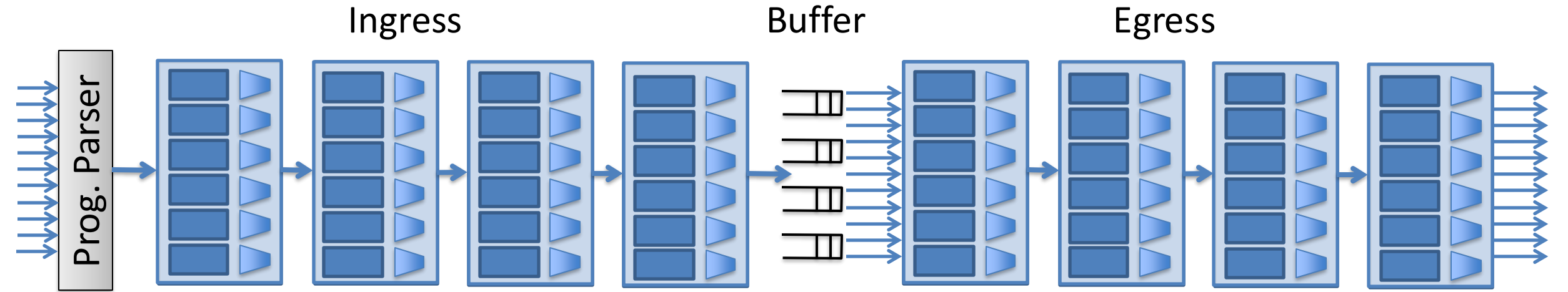
Intel is open sourcing the Tofino compiler

(+ most of the software stack)

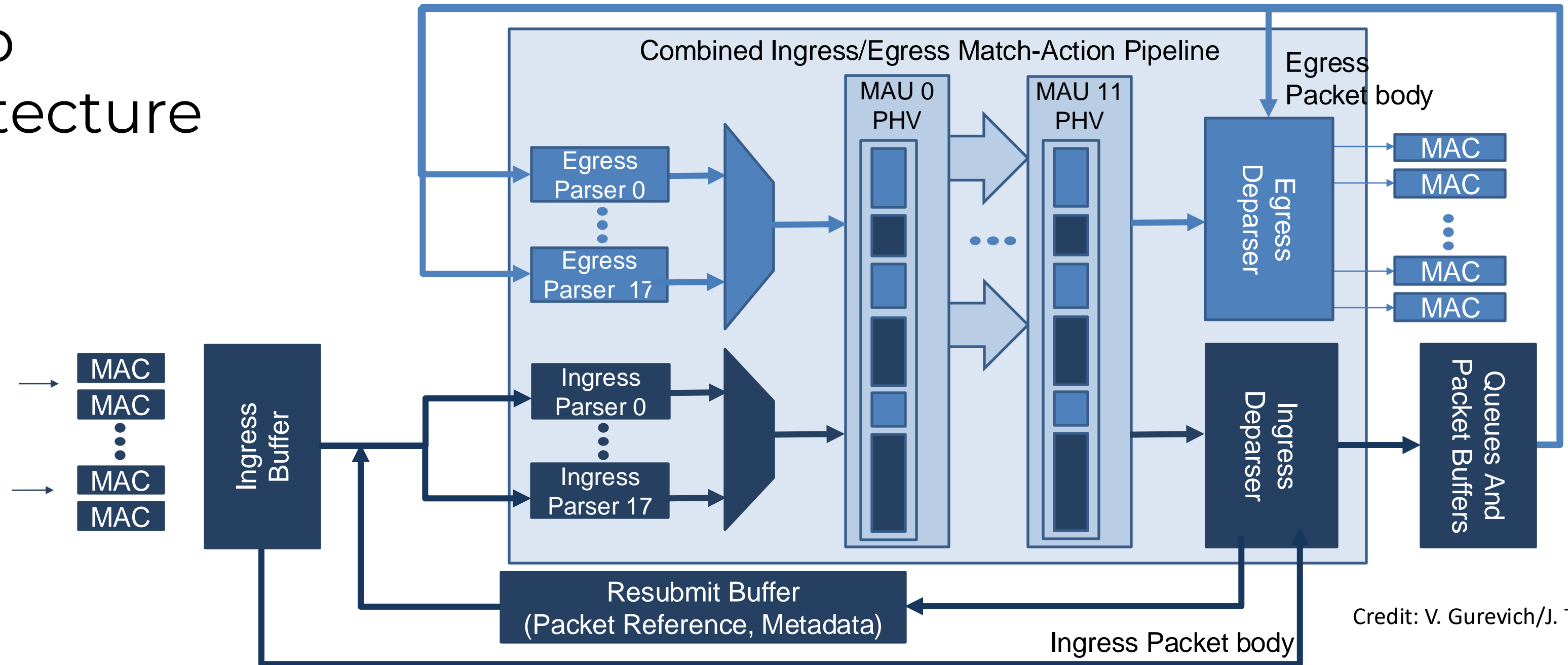
Outline

- 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)
 - Wide (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:

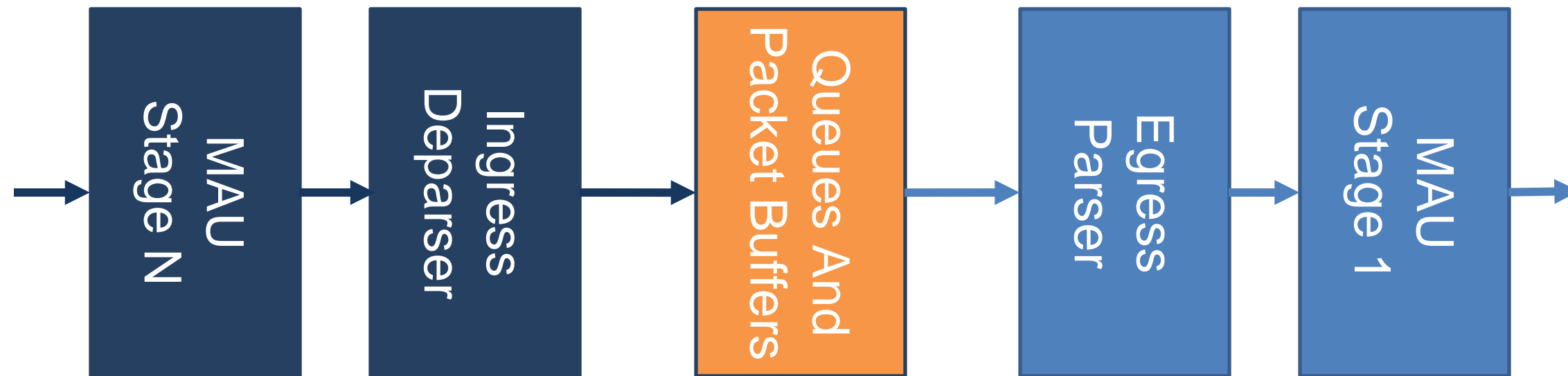
- Similarity to PsaSwitchMidEnd
- Some specializations (P4:: → BFN::)
- Design patterns: adherence + recommendations
- Architecture translation (TNA IR nodes)
- Normalization (tna/t2na/v1model)
- Simplifications (cast elimination, control rewrites, ...)
- Varbit: verify usage / desugar
- ...

```
addPasses(  
    new P4::RemoveNops(),  
    new P4::EliminateNewType(),  
    new P4::EliminateSerEnums(),  
    new BFN::TypeChecking(),  
    new BFN::SetDefaultSize(), // set default table size for tables w/o "size"  
    new BFN::ArchTranslation(),  
    new BFN::FindArchitecture(),  
    new BFN::TypeChecking(),  
    new BFN::CheckDesignPattern(), // add checks for p4 design pattern here,  
    new BFN::SetDefaultSize(), // belt and suspenders, in case of IR mutation  
    new BFN::InitialzeMirrorIOSelect(),  
    new BFN::DropBackendsMirrorEngine(),  
    new EnumOn32bits::FindStatefulEnumOutputs(),  
    new P4::ConvertTypes(),  
    new P4::ConstantFolding(),  
    new P4::EliminateTypeDef(),  
    new P4::SimplifyControlFlow(),  
    new P4::SimplifyKey(&keyMap, &typeMap,  
        BFN::KeyIsSimple::getPolicy()),  
    Device::currentDevice() != Device::TDFMO ? options.disable_direct_exit ?  
    new P4::RemoveExit(), : nullptr,  
    new P4::ConstantFolding(),  
    new BFN::ElimCasts(),  
    new BFN::AlgoImplementation(),  
    new BFN::TypeChecking(),  
    // has to be early enough for setValid to still not be lowered  
    new BFN::CheckVarbitAccess(),  
    new P4::StrengthReduction(),  
    new P4::SimplifySelectCases(), // require constant keysets  
    new P4::ExpandCobase(),  
    new P4::ExpandExit(),  
    new P4::SimplifyParser(),  
    new P4::ReorderSelectCases(),  
    new P4::StrengthReduction(),  
    new P4::SimplifyComparisons(),  
    new BFN::CopyHeader(),  
    // must run after copy structure  
    new P4::SimplifyIfStatement(),  
    new RewriteFlexibleStruct(),  
    new SimplifyExitArgs(),  
    new P4::NestedStructs(),  
    new P4::SimplifySelectList(),  
    new BFN::RemoveAccessItems(),  
    new P4::Predication(),  
    new P4::MoveDeclarations(), // more may have been introduced  
    new P4::ConstantFolding(),  
    new P4::SimplifyBitwise(),  
    new P4::LocalCopyPropagation(),  
    new P4::ConstantFolding(),  
    new P4::StrengthReduction(),  
    new P4::MoveDeclarations(),  
    new P4::SimplifyMeshList(),  
    new P4::SimplifyMeshDef(),  
    new P4::SimplifyControlFlow(),  
    new P4::TableExit(),  
    evaluator,  
    new KeyIsFunction() -> const IR::Node * f  
    auto toplevel = evaluator->getToplevelBlock();  
    if (!toplevel) {  
        // nothing further to do  
        return nullptr;  
    }  
    for (auto &block : toplevel->getChildren()) {  
        if (!block->getDeclByName())  
            continue;  
        if (!block->getDeclByName())  
            skip_controls->replace();  
    }  
    return root;  
});  
new P4::SynthesizeActions(&keyMap, &typeMap,  
    new ActionSynthesisPolicy()),  
    new P4::MoveActionsToTables(),  
    new CopyBlockPragmas(),  
    { 7  
    new RewriteExprIntrinsicMetadataHeader() : nullptr,  
    new DesugarVarbitExtract(),  
    new CheckRegisterActions(),  
    new PingPongGeneration(),  
    new RegisterReadWrite(),  
    new BFN::AnnotateWithInHash(),  
    new BFN::FoldConstantEnums(),  
    BackendOptions().disable_parse_min_depth_limit &&  
    BackendOptions().disable_parse_max_depth_limit  
    ? nullptr  
    : new BFN::ParserEnforceDepthReq(),  
    // Collects source info for logging. Call this after all transformations are complete.  
    sourceInfoLogging.
```

Midend/backend tour (cont.)

BridgedPacking:

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



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

Midend/backend tour (cont.)

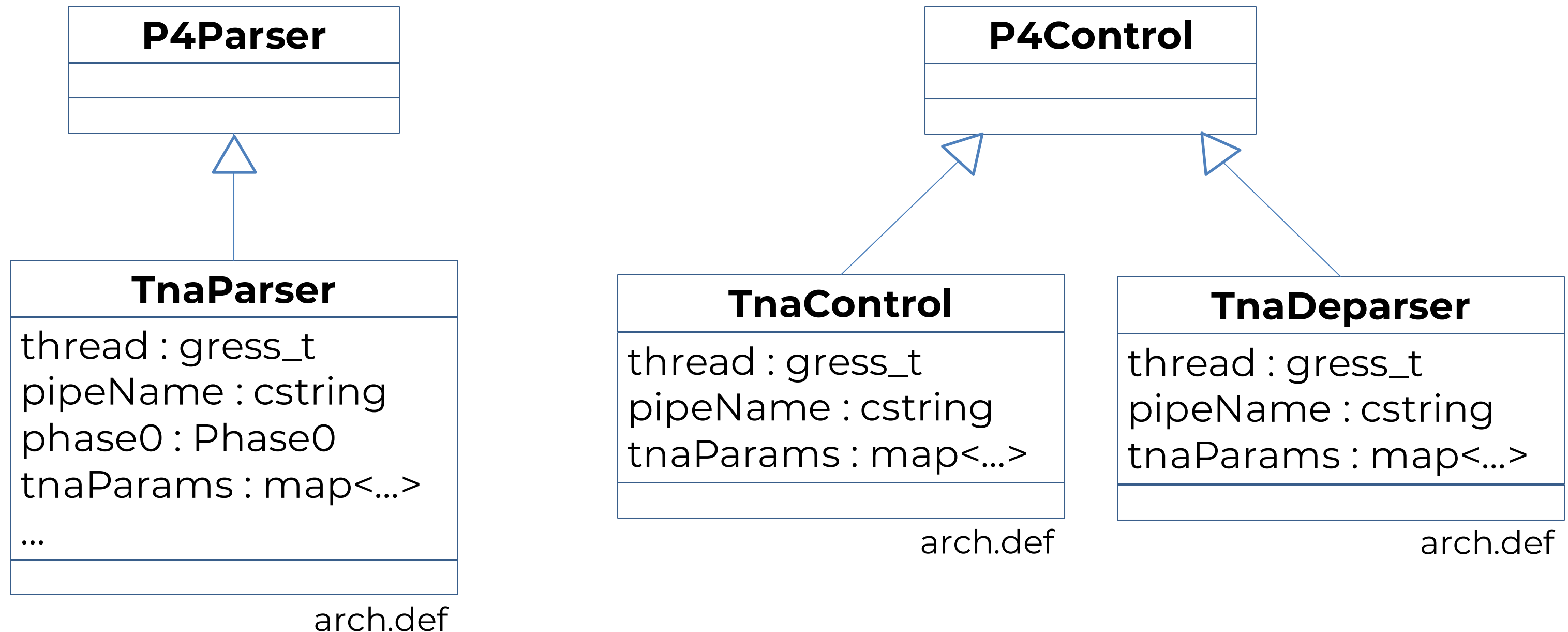
BackEnd:

- Instruction selection
- PHV allocation
- Table allocation

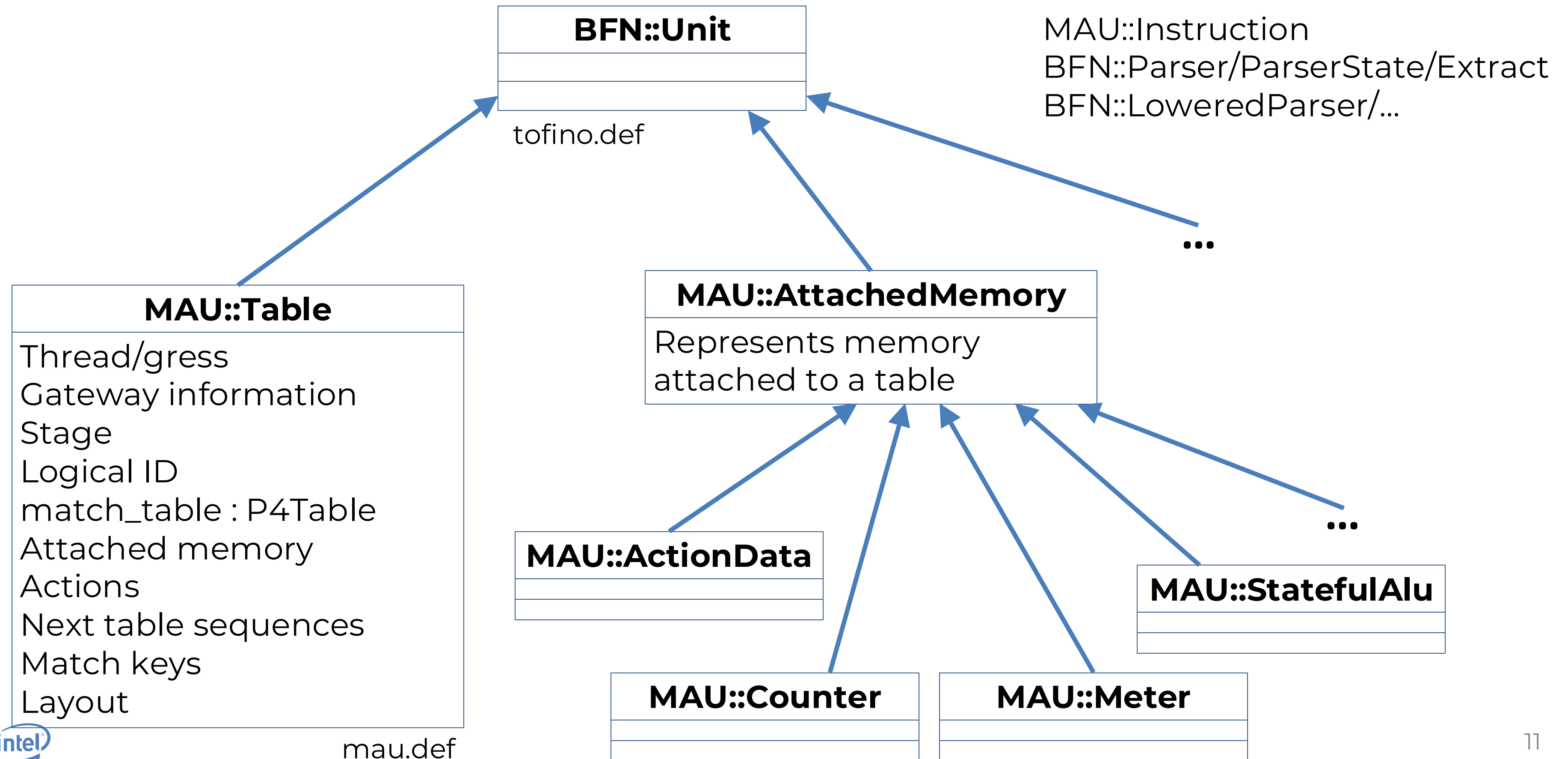
- Smaller features:
 - Aliasing (detecting two fields are equivalent → share allocation)
 - Metadata initialization (optional)
 - Pragma collection/application
 - Parser IR lowering
 - ...

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);  
        } } }  
}
```

- 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