A case for a faster and simpler driver

NIR on the Mesa i965 backend

Track: Graphics devroom
Room: K.3.401
Day: Sunday
Start: 11:00
End: 11:50

Eduardo Lima Mitev
elima@igalia.com

igalia

FOSDEM '16
Brussels 30 & 31 January 2016
www.fosdem.org »

TL; DR

During Q2 and Q3 2015,

the graphics team at Igalia

worked on the i965 Mesa backend

to replace the existing vector-based GLSL-IR compiler

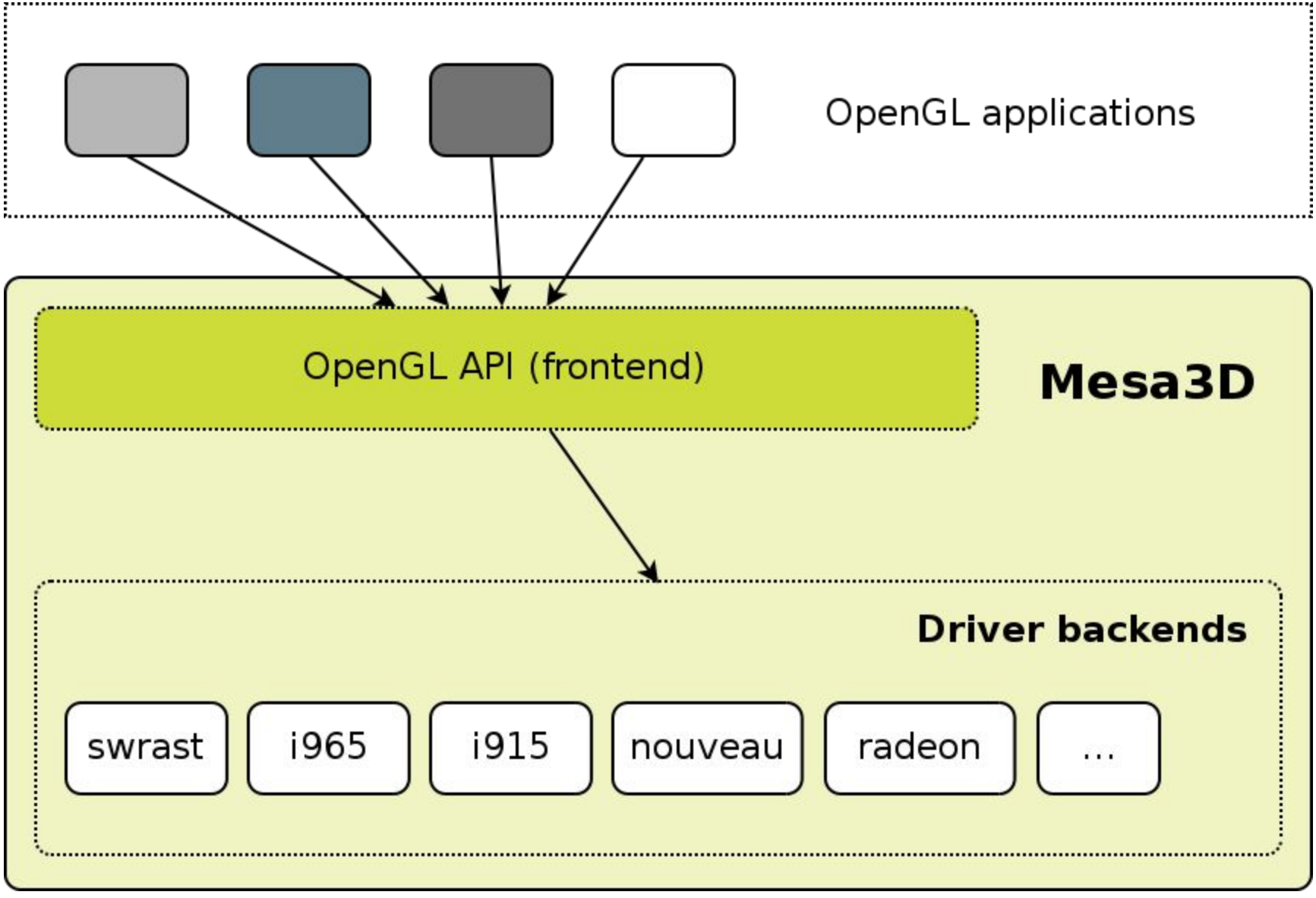
by a new one based on NIR,

resulting in performance improvements and driver simplification

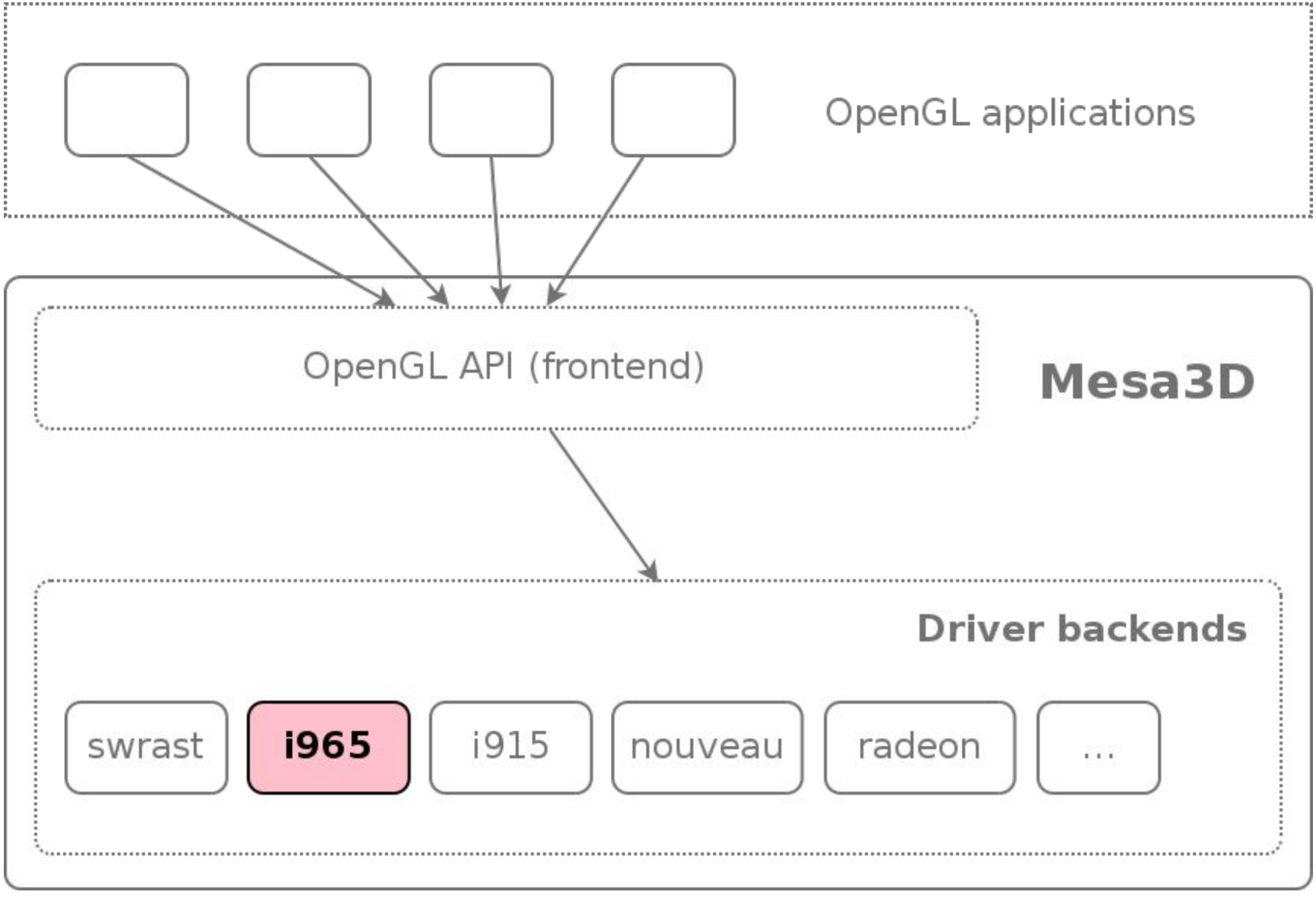
Some basic terminology...

- I use “Mesa” and “Mesa3D” interchangeably
- When I use “OpenGL” or “GL”, it also includes OpenGL-ES
- “genX” means the generation X of Intel(R)’s processor family
- a “pass” in shader compilation context refers to a transformation in the IR

(Super-simplified) Architecture of Mesa



Focus on the i965 backend



What is a Mesa backend?

Piece of the driver that handles all aspects of a particular rendering device.

Normally a specific piece of HW. i.e, (a family of) graphics cards.

What does a Mesa backend do?

- Initialize and configure the device for rendering
- Allocate and manage device resources
- Compile shader programs to device’s native code

What does a Mesa backend do?

- Initialize and configure the device for drawing
- Allocate and manage device resources
- Compile shader programs to device’s native code

Compiling a shader to native code

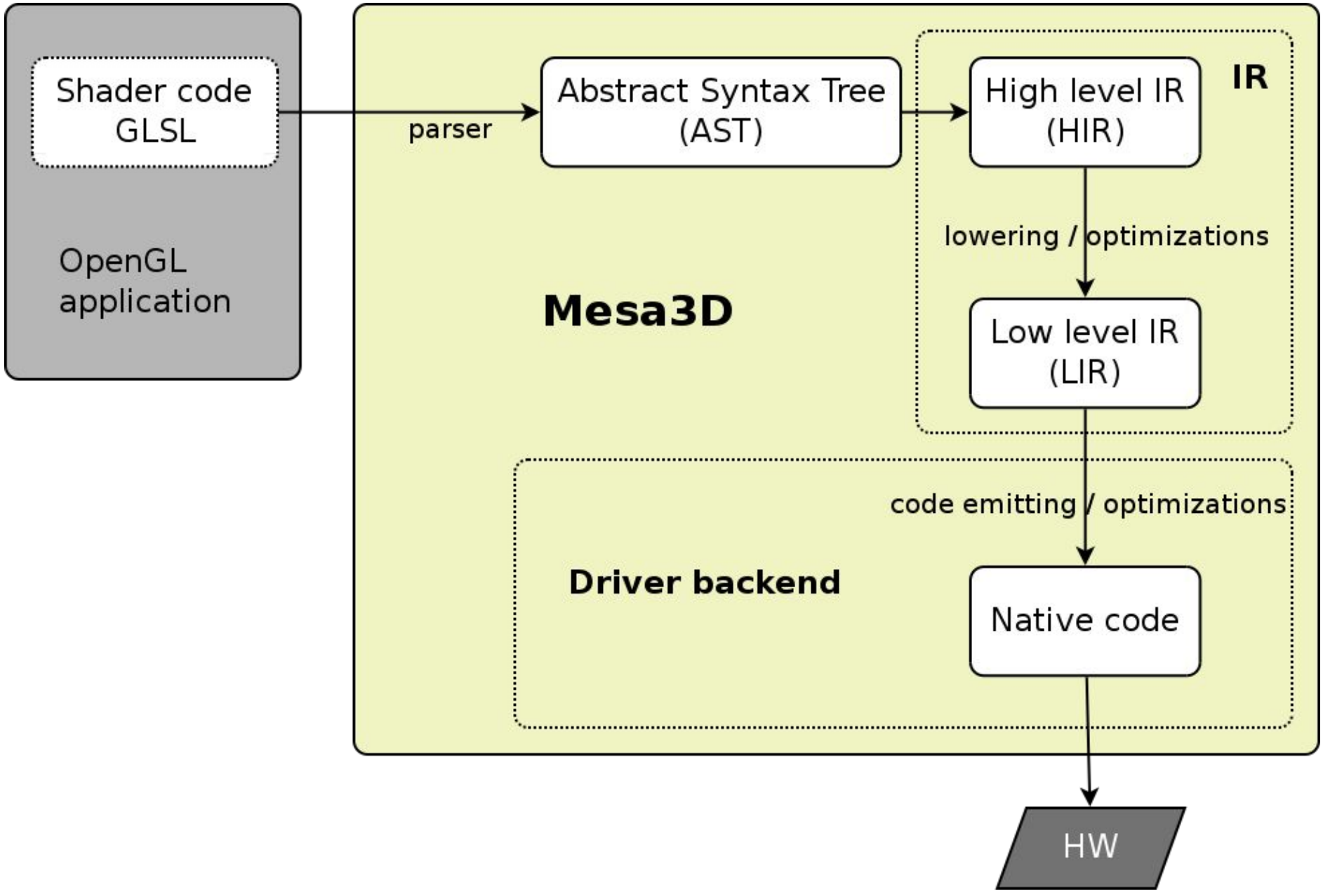
ex. GLSL vertex shader

```
#version 300 es

layout (location = 0) in vec2 attr_pos;
layout (location = 1) in vec3 attr_color;

out vec4 color;

void main() {
  gl_Position = vec4(attr_pos.yx, 0.0, 1.0);
  color = vec4(attr_color, 1.0);
}
```

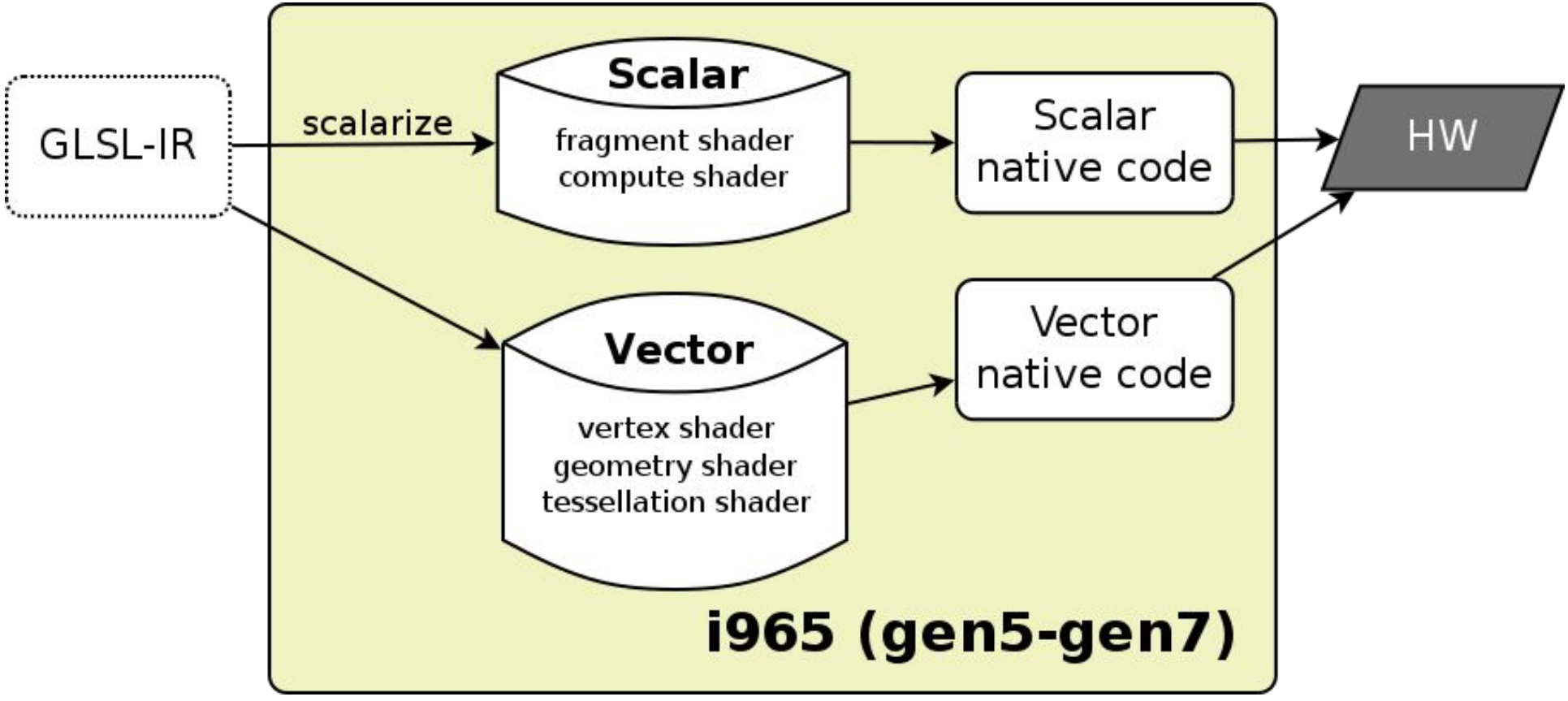
```
mov(8)     g115<1>.zUD     0x00000000UD                { align16 NoDDClr 1Q };
mov(8)     g116<1>.xyzD    g2<4,4,1>.xyzzD             { align16 NoDDClr 1Q };
mov(8)     g114<1>UD       0x00000000UD                { align16 1Q compacted };
mov(8)     g115<1>.wD      1065353216D                 { align16 NoDDClr,NoDDChk 1Q };
mov(8)     g116<1>.wD      1065353216D                 { align16 NoDDChk 1Q };
mov(8)     g115<1>.xyD     g1<4,4,1>.yxxxD             { align16 NoDDChk 1Q };
mov(8)     g113<1>UD       g0<4,4,1>UD             { align16 WE_all 1Q };
or(1)      g113.5<1>UD     g0.5<0,1,0>UD   0x0000ff00UD      { align1 WE_all };
send(8)    null            g113<4,4,1>F
                urb 0 write HWord interleave complete mlen 5 rlen 0 { align16 1Q EOT };
nop
```

resulting i965 native (HSW)

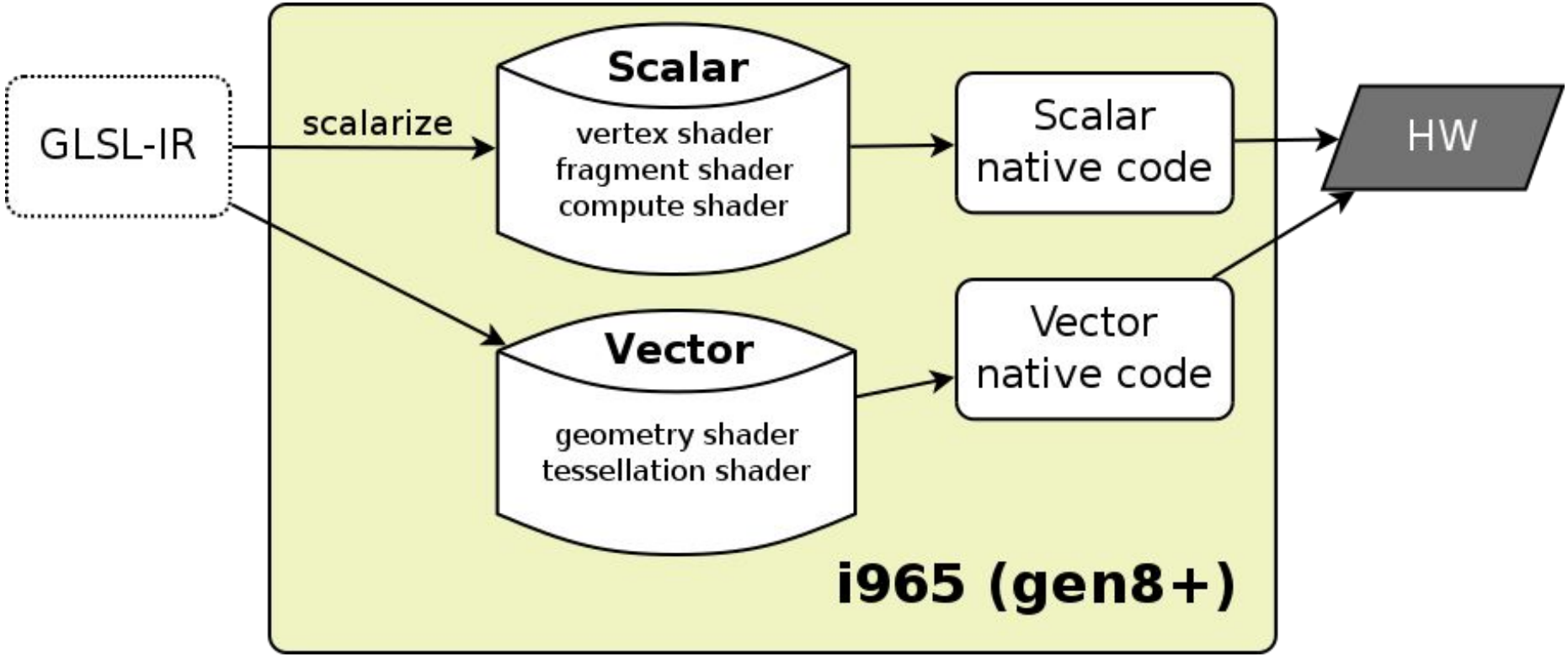
Mesa shader compilation pipeline



i965 shader pipeline (gen5 to gen7)



i965 backend shader pipeline (gen8+)

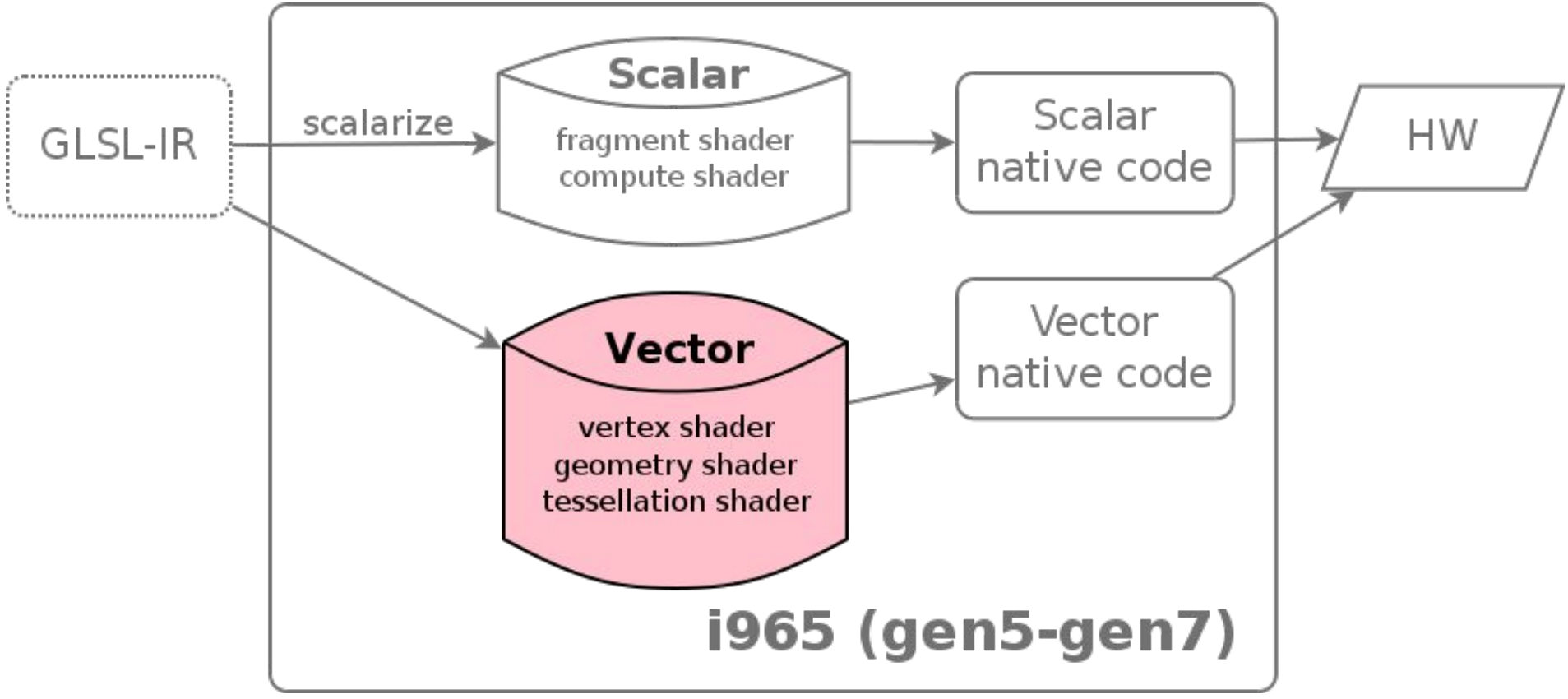


Scalar vs. vector-based

- Scalar: 1 component per register
- Vector: 4 components per register (addressable from one instruction)
- Vector: Instructions operate on vectors

For example (in i965):

```
add(8)          g116<1>.xyzF    g2<4,4,1>.xyzzF  0.5F
mov(8)          g115<1>.xyD     g1<4,4,1>.yxxxD
```

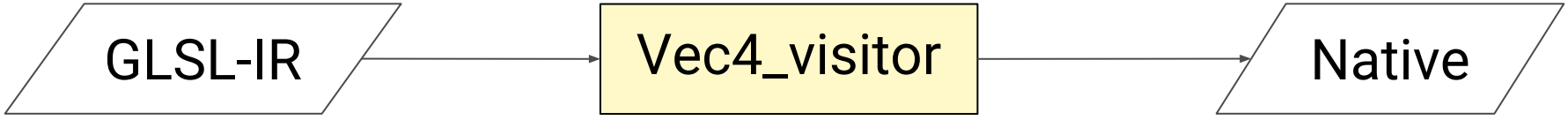


We will talk about the i965 vector pass

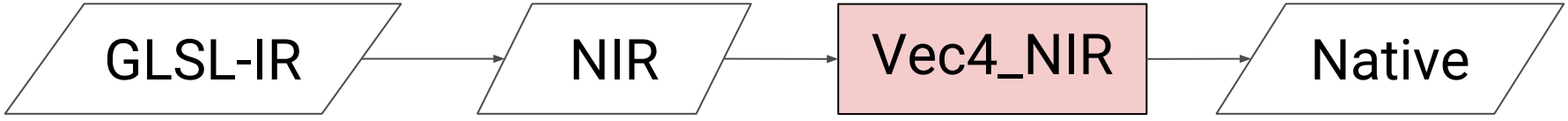
Vec4-NIR

A new IR-to-native vector pass

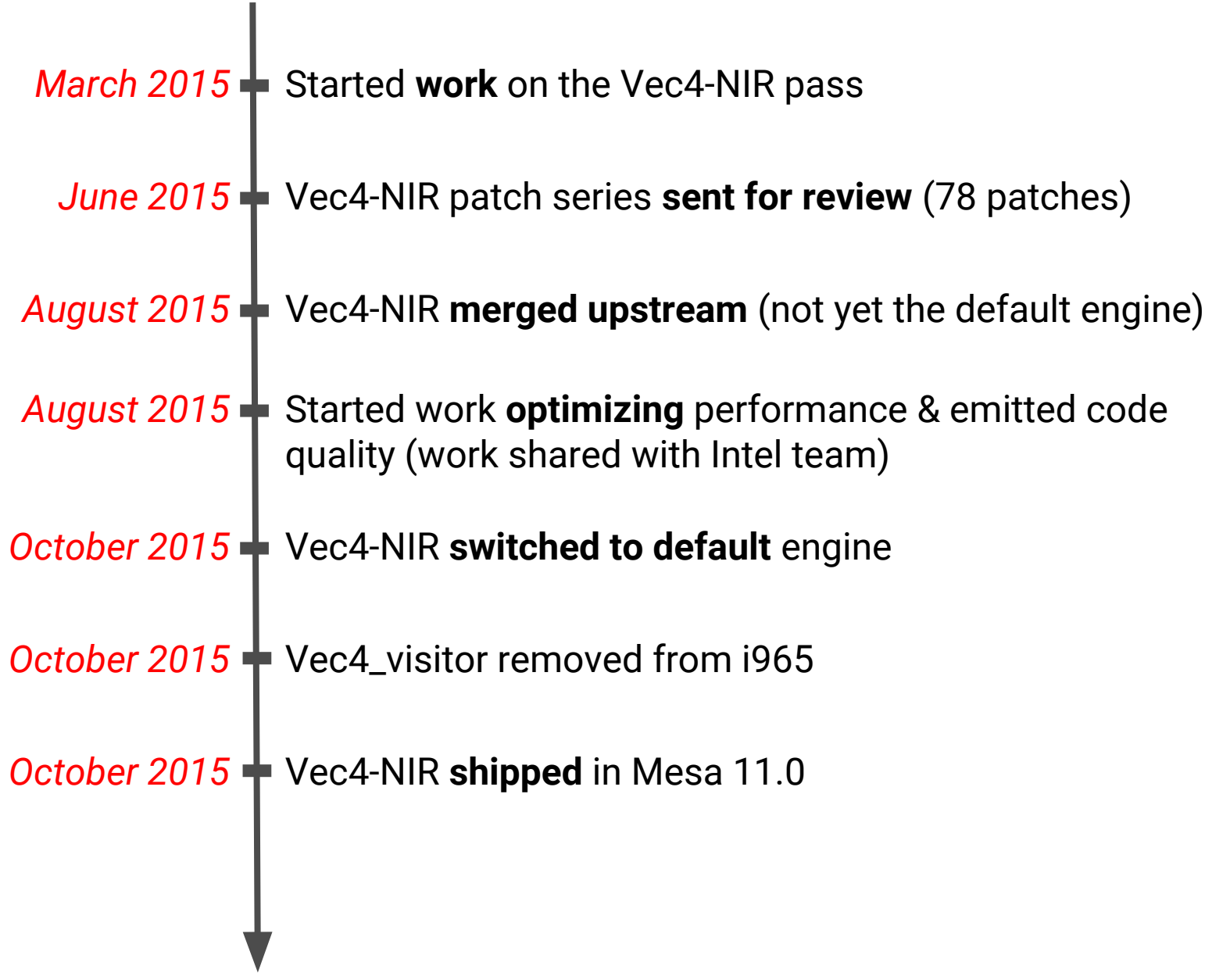
Before:



After:



Timeline


March 2015
Started **work** on the Vec4-NIR pass
June 2015
Vec4-NIR patch series **sent for review** (78 patches)
August 2015
Vec4-NIR **merged upstream** (not yet the default engine)
August 2015
Started work **optimizing** performance & emitted code quality (work shared with Intel team)
October 2015
Vec4-NIR **switched to default** engine
October 2015
Vec4_visitor removed from i965
October 2015
Vec4-NIR **shipped** in Mesa 11.0

What's NIR?

- A (N)ew (I)ntermediate (R)epresentation in Mesa
- Implements Single Static Assignment (SSA) natively
- Also, a data-structure and API available to backends
- Was already used in i965's scalar pass
- There is a GLSL-IR to NIR translator

GLSL-IR versus NIR

- GLSL-IR is typed, NIR is typeless
- GLSL-IR is based on expression trees, NIR is a flat IR
- More explicit control-flow in NIR

NIR makes it much easier to write lowering/optimization passes on the IR

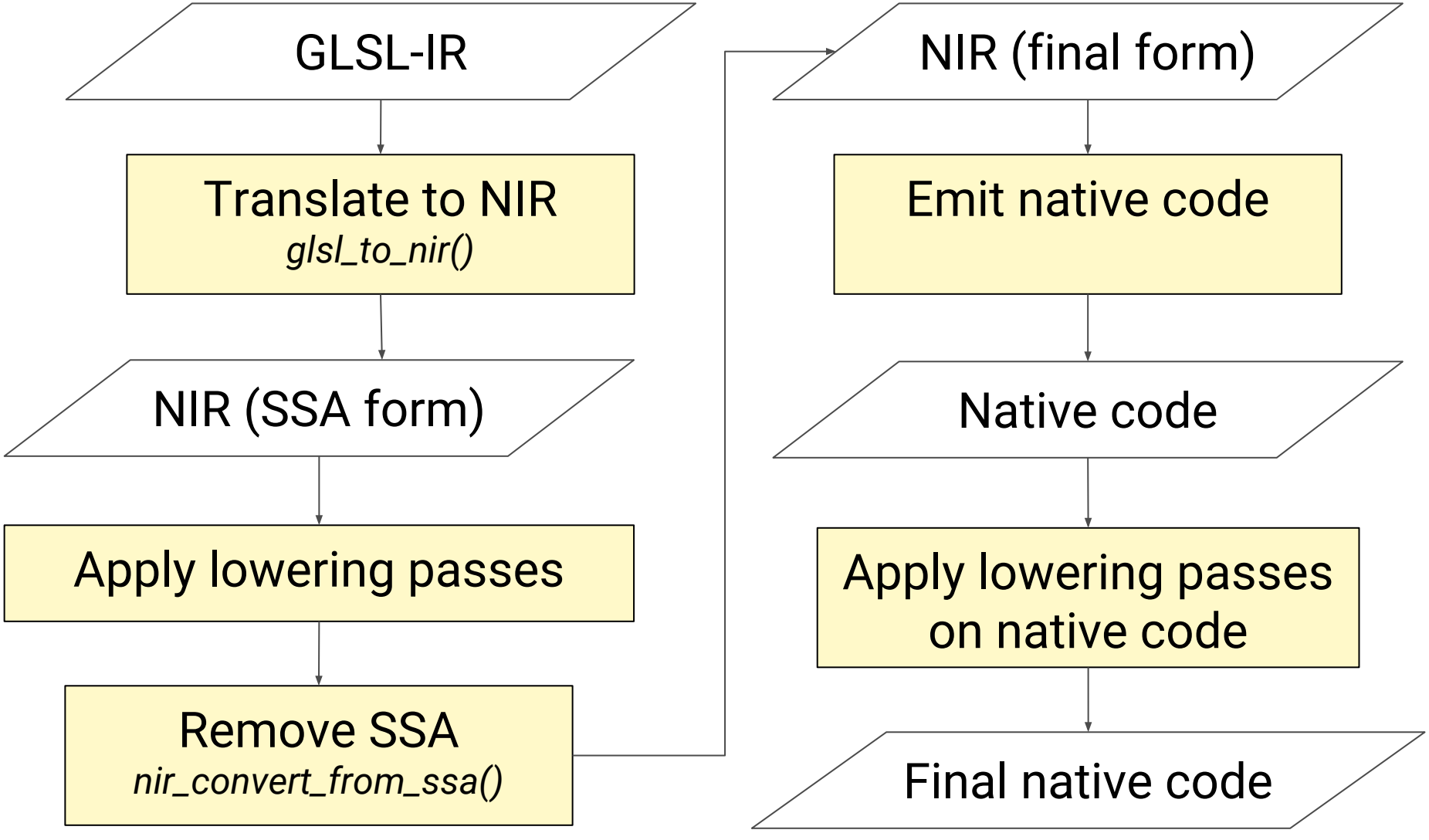
GLSL-IR vertex shader example

```
(declare (location=0 shader_out ) vec4 gl_Position)
(declare (location=26 shader_out ) vec4 color)
(declare (location=17 shader_in ) vec2 attr_pos)
(declare (location=18 shader_in ) vec3 attr_color)
( function main
  (signature void
        (parameters
        )
        (
        (declare (temporary ) vec4 vec_ctor)
        (assign  (zw) (var_ref vec_ctor)  (constant vec2 (0.000000 1.000000)) )
        (assign  (xy) (var_ref vec_ctor)  (swiz yx (var_ref attr_pos) ))
        (assign  (xyzw) (var_ref gl_Position)  (var_ref vec_ctor) )
        (declare (temporary ) vec4 vec_ctor@2)
        (assign  (w) (var_ref vec_ctor@2)  (constant float (1.000000)) )
        (assign  (xyz) (var_ref vec_ctor@2)  (var_ref attr_color) )
        (assign  (xyzw) (var_ref color)  (var_ref vec_ctor@2) )
        ))
)
)
```

NIR vertex shader example

```
decl_var shader_in INTERP_QUALIFIER_NONE vec2 attr_pos (VERT_ATTRIB_GENERIC0, 0)
decl_var shader_in INTERP_QUALIFIER_NONE vec3 attr_color (VERT_ATTRIB_GENERIC1, 1)
decl_var shader_out INTERP_QUALIFIER_NONE vec4 gl_Position (VARYING_SLOT_POS, 0)
decl_var shader_out INTERP_QUALIFIER_NONE vec4 color (VARYING_SLOT_VAR0, 26)
decl_overload main returning void

impl main {
    decl_reg vec4 r0
    decl_reg vec4 r1
    block block_0:
    /* preds: */
    vec1 ssa_0 = load_const (0x3f800000 /* 1.000000 */)
    vec2 ssa_1 = load_const (0x00000000 /* 0.000000 */, 0x3f800000 /* 1.000000 */)
    vec2 ssa_2 = intrinsic load_input () () (0)    /* attr_pos */
    r0.xy = imov ssa_2.yx
    r0.zw = imov ssa_1.xy
    vec3 ssa_4 = intrinsic load_input () () (1)    /* attr_color */
    r1.xyz = imov ssa_4
    r1.w = imov ssa_0.x
    intrinsic store_output (r0) () (0)    /* gl_Position */
    intrinsic store_output (r1) () (26)    /* color */
    /* succs: block_1 */
    block block_1:
}
```

Anatomy of a NIR backend



Emitting native code

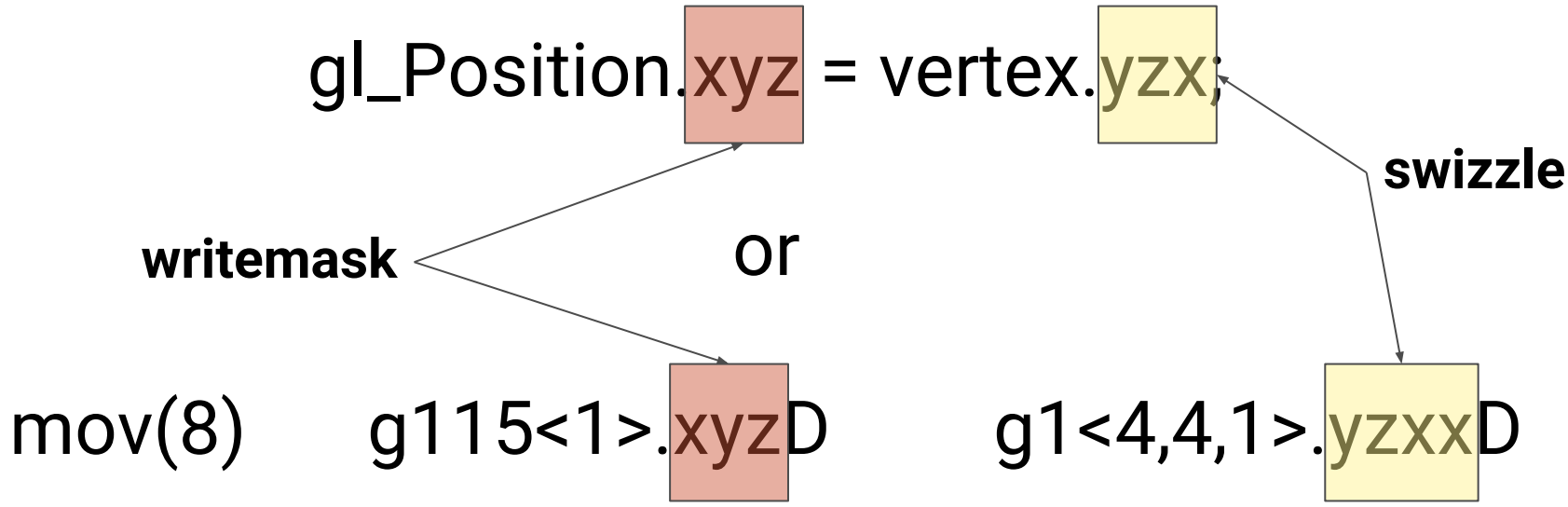
emit_nir_code()

1. Setup global registers (e.g, to hold shader input/output)
2. Find the entry-point function (“main”)
3. Setup local registers
4. Walk the body of the function, emitting control-flow instructions (*if*, *loop*, *block*, *function*)
5. For *block* instructions, walk the instructions inside
6. Emit native code for each instruction

Types of instructions

- Control flow
- ALUs
- Intrinsics
- Load constant
- Texture operations
- SSA-related (ssa_undef, phi, etc)

Writemask and swizzle



Challenges

A complex domain

- Understand the architecture of a modern GPU
- Locate and extract relevant information from large PRMs

A moving target

- NIR, a fairly new thing, constantly evolving
- Our reference code, the FS-NIR backend, also changing

Technical

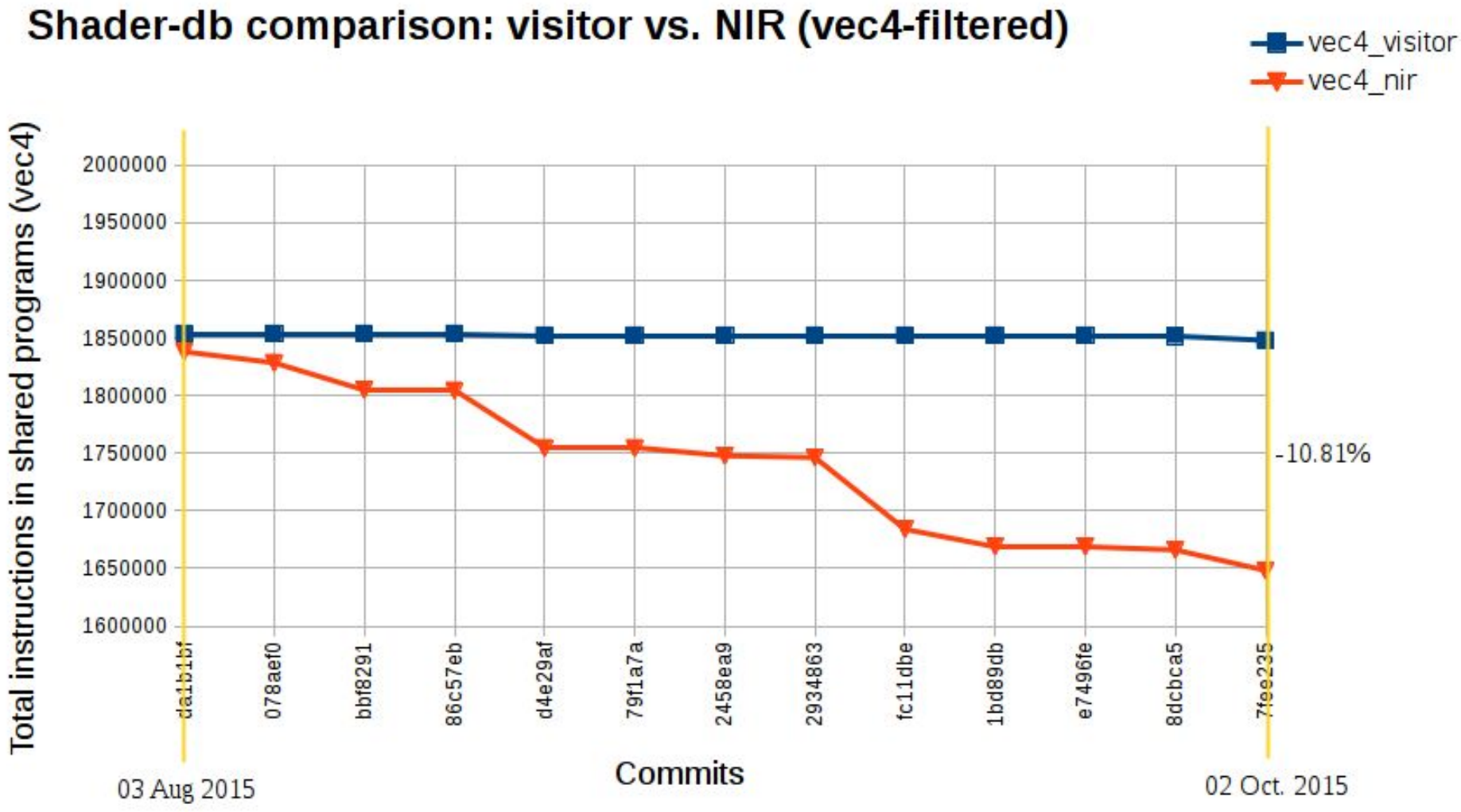
- Lots of work to get even a simple shader working
- NIR lack of support for some vector operations (e.g, I/O lowering passes)
- Bugs in NIR or its lowering passes

Technical (II)

- New emitted code broke some i965 native optimization cases
- Dealing with vectors: writemasks and swizzles
- GPU hangs!

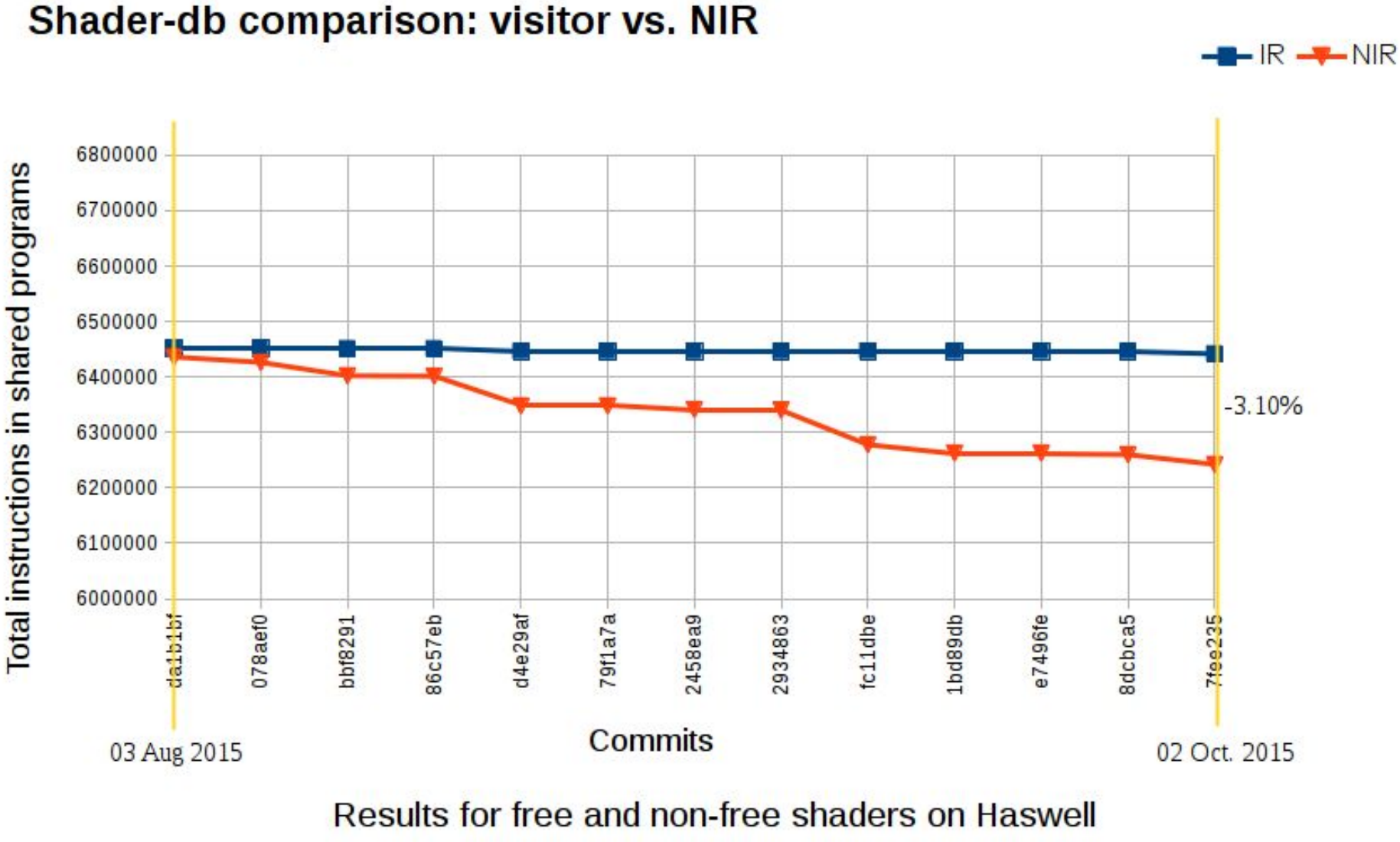
Performance

Shader-db results for vec4 shaders



Results for free and non-free shaders on Haswell

Shader-db results for all shaders



Vec4_visitor vs. Vec4_NIR

Bulk analysis of generated native code (shader-db) suggests a conservative **improvement of 5% to 10%** against old vec4_visitor

- GL benchmarks (Unigine Heaven, some GFXBench tests) show similar results for both
- Benchmarking real world apps would be necessary (e.g, games) for a more relevant assessment
- But this comparison not important anymore: vec4_visitor was removed from backend

Backend code simplification

Some immediate gains

- **Consistency**: now both Scalar and Vector passes use NIR
- **Simplicity**: we moved from a recursive pass to a linear one
- **Maintainability**: new code is much, much easier to read and reason about

A bunch of code was removed

```
Author:     Jason Ekstrand <jason.ekstrand@intel.com>
AuthorDate: Mon Sep 21 11:03:29 2015 -0700
Commit:     Jason Ekstrand <jason.ekstrand@intel.com>
CommitDate: Fri Oct 2 14:19:34 2015 -0700

    i965/vec4: Delete the old ir_visitor code

    Reviewed-by: Matt Turner <mattst88@gmail.com>

1       72      src/mesa/drivers/dri/i965/brw_vec4.h
0       45      src/mesa/drivers/dri/i965/brw_vec4_gs_visitor.cpp
0       3       src/mesa/drivers/dri/i965/brw_vec4_gs_visitor.h
121     1994    src/mesa/drivers/dri/i965/brw_vec4_visitor.cpp
1       30      src/mesa/drivers/dri/i965/gen6_gs_visitor.cpp
0       2       src/mesa/drivers/dri/i965/gen6_gs_visitor.h
```

123 lines in, 2146 lines out

more here too

```
Author:     Jason Ekstrand <jason.ekstrand@intel.com>
AuthorDate: Mon Sep 21 11:07:32 2015 -0700
Commit:     Jason Ekstrand <jason.ekstrand@intel.com>
CommitDate: Fri Oct 2 14:19:36 2015 -0700

    i965/vec4: Delete the old vec4_vp code

    Reviewed-by: Matt Turner <mattst88@gmail.com>

0       1       src/mesa/drivers/dri/i965/Makefile.sources
0       1       src/mesa/drivers/dri/i965/brw_vec4.h
0       9       src/mesa/drivers/dri/i965/brw_vec4_gs_visitor.cpp
0       1       src/mesa/drivers/dri/i965/brw_vec4_gs_visitor.h
0       649     src/mesa/drivers/dri/i965/brw_vec4_vp.cpp
0       1       src/mesa/drivers/dri/i965/brw_vs.h
0       5       src/mesa/drivers/dri/i965/test_vec4_copy_propagation.cpp
0       5       src/mesa/drivers/dri/i965/test_vec4_register_coalesce.cpp
```

0 lines in, 672 lines out

But the new pass added up almost the same amount of lines over time.

However, we effectively moved code out of the backend (i965), and into the common layer (Mesa).

But more importantly

- Future looks bright:
  - Much easier to write new optimization passes
  - All is set for a SPIR-V to NIR pass

Final words

- Challenging but very interesting work
- Awesome to hack a driver we ourselves use everyday
- Vital support from Intel’s Mesa team (specially Jason Ekstrand, Matt Turner and Kenneth Graunke) **Thank you!**
- Mesa: an great project and community

Thank you!

Q & A