SHADERS IN RADEONSI DYNAMIC LINKING AND NIR

NICOLAI HÄHNLE

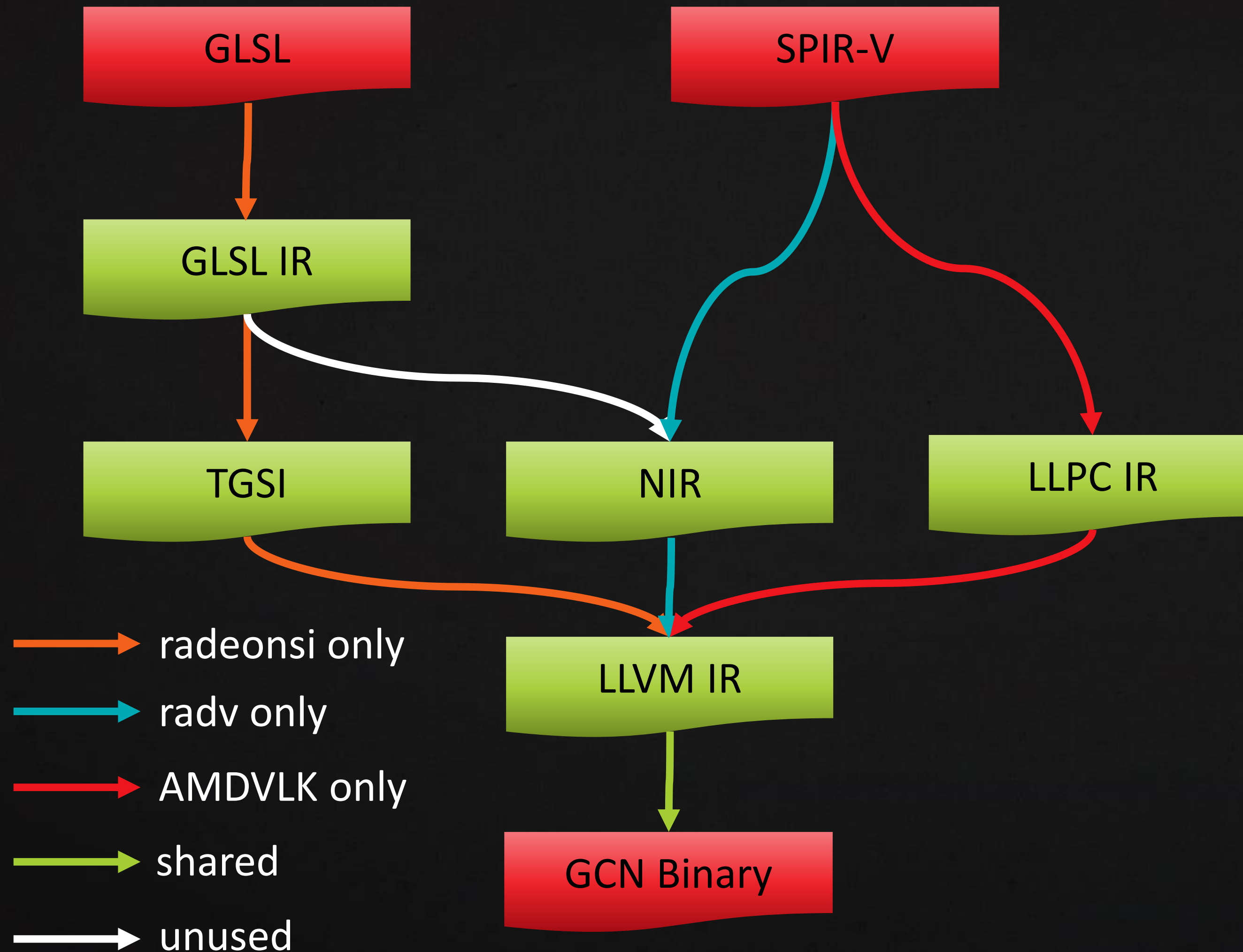
FOSDEM 2018

AMD

NIR IN RADEONSI

SHADER COMPILATION FLOWS – TODAY'S DEFAULTS

OPEN-SOURCE DRIVERS FOR AMD GPUS

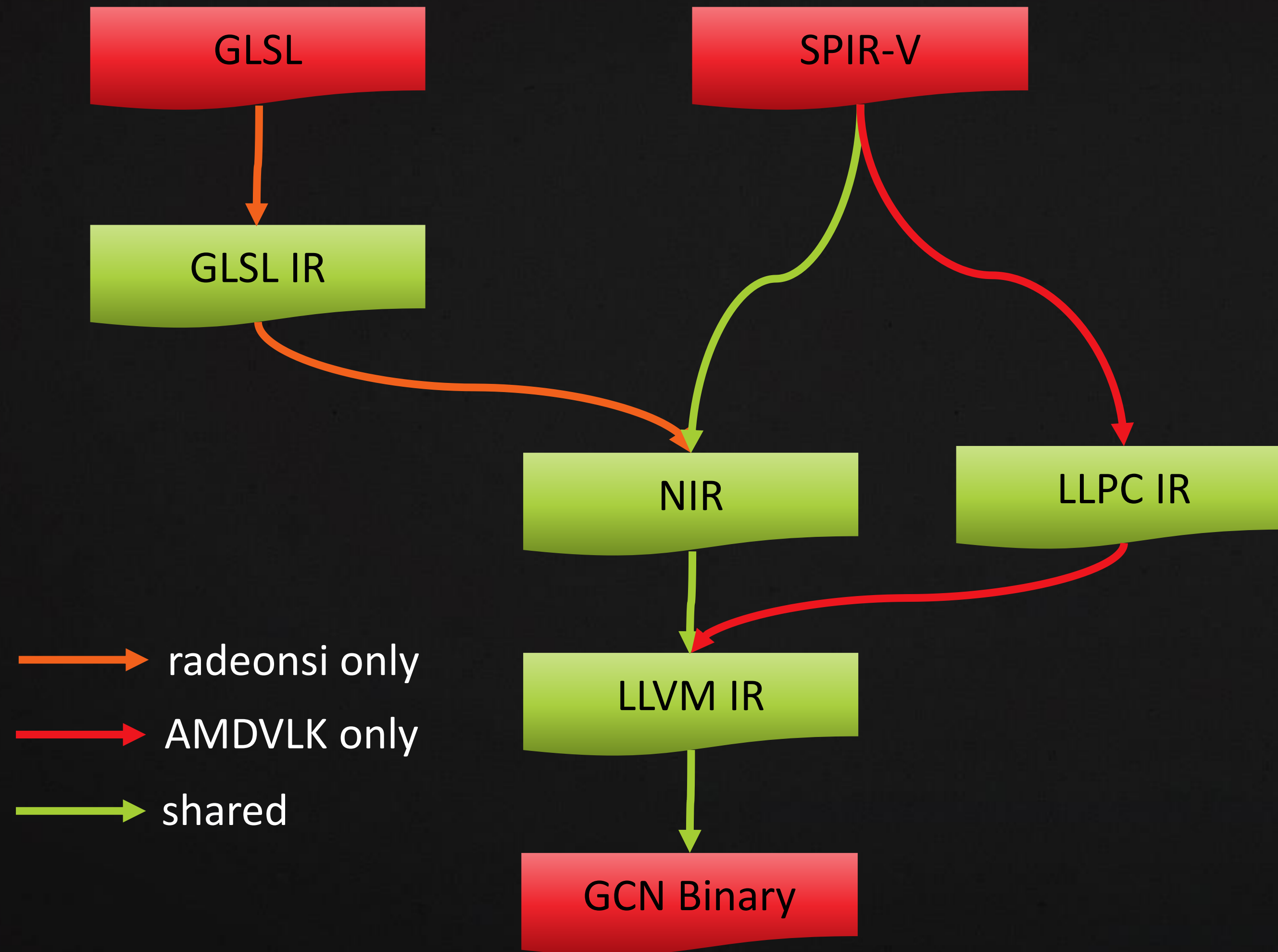


- Final code generation is shared
- Frontends not shared at all
- NIR originally developed for Intel driver
  - GLSL IR-to-NIR, SPIR-V-to-NIR used in i965 and anv, respectively
  - NIR already used by some Gallium drivers
- LLPC IR = LLVM IR with additional "intrinsics"

RADEON TECHNOLOGIES GROUP

SHADER COMPILATION FLOWS – THE PLAN

OPEN-SOURCE DRIVERS FOR AMD GPUS



- Unified frontend in Mesa
- SPIR-V-to-NIR shared for ARB_gl_spirv and OpenGL 4.6

RADEON TECHNOLOGIES GROUP

WHY NIR?

- Reduce frontend code duplication
- Leverage Vulkan work for ARB_gl_spirv and OpenGL 4.6
- NIR is better suited for representing new features (e.g. 16 bit)
- NIR is suited for code transforms
  - Simplifies driver-specific optimization flows
  - Enables hardware-specific optimization passes (e.g. for gfx9 merged shaders)

STATUS OF NIR IN RADEONSI

- Very close to feature parity
- Needs performance work
- Enable with R600_DEBUG=nir
- Kudos to Dave Airlie, Bas Nieuwenhuizen, Timothy Arceri, Samuel Pitoiset
- Future of TGSI?
  - Used by MM, nine, Gallium helpers – ttn (TGSI-to-NIR) helps
  - Used to encode shaders for virtualization (svga, virgl)

DYNAMICALLY LINKING SHADERS

WHAT AND WHY?

- LLVM generates a standard ELF object with AMDGPU-specific sections
- Driver combines multiple ELF objects to a single binary
  - –Currently ad-hoc: paste .text sections together
  - –Goal: support (some) additional sections and real relocations
- Proper dynamic linking should allow:
  - –.rodata
  - –Explicit description of LDS variables

SHADER PROLOGS AND EPILOGS

AVOIDING RECOMPILE STUTTER

- Simple pixel shader: R600_DEBUG=ps glxgears

```
s_mov_b32               m0, s9
v_interp_mov_f32        v0, p0, attr0.x
v_interp_mov_f32        v1, p0, attr0.y
v_interp_mov_f32        v2, p0, attr0.z
v_interp_mov_f32        v3, p0, attr0.w

v_cvt_pkrtz_f16_f32     v0, v0, v1
v_cvt_pkrtz_f16_f32     v1, v2, v3

exp mrt0 v0, v0, v1, v1 done compr vm
s_endpgm
```


RADEON TECHNOLOGIES GROUP

SHADER PROLOGS AND EPILOGS

AVOIDING RECOMPILE STUTTER

- Simple pixel shader: R600_DEBUG=ps glxgears

```
s_mov_b32             m0, s9
v_interp_mov_f32      v0, p0, attr0.x
v_interp_mov_f32      v1, p0, attr0.y
v_interp_mov_f32      v2, p0, attr0.z
v_interp_mov_f32      v3, p0, attr0.w
```

Load constant input attribute

```
v_cvt_pkrtz_f16_f32   v0, v0, v1
v_cvt_pkrtz_f16_f32   v1, v2, v3
```

Pack into 16-bit floating point values (round to zero)

```
exp mrt0 v0, v0, v1, v1 done compr vm
s_endpgm
```

Export to color buffer and end program

- Scalar instructions and registers
- Vector instructions and registers
- Special instructions and registers

SHADER PROLOGS AND EPILOGS

AVOIDING RECOMPILE STUTTER

- Simple pixel shader: R600_DEBUG=ps glxgears

```
s_mov_b32               m0, s9
v_interp_mov_f32        v0, p0, attr0.x
v_interp_mov_f32        v1, p0, attr0.y
v_interp_mov_f32        v2, p0, attr0.z
v_interp_mov_f32        v3, p0, attr0.w
```

Depends **only** on original GLSL source
"Main shader part"

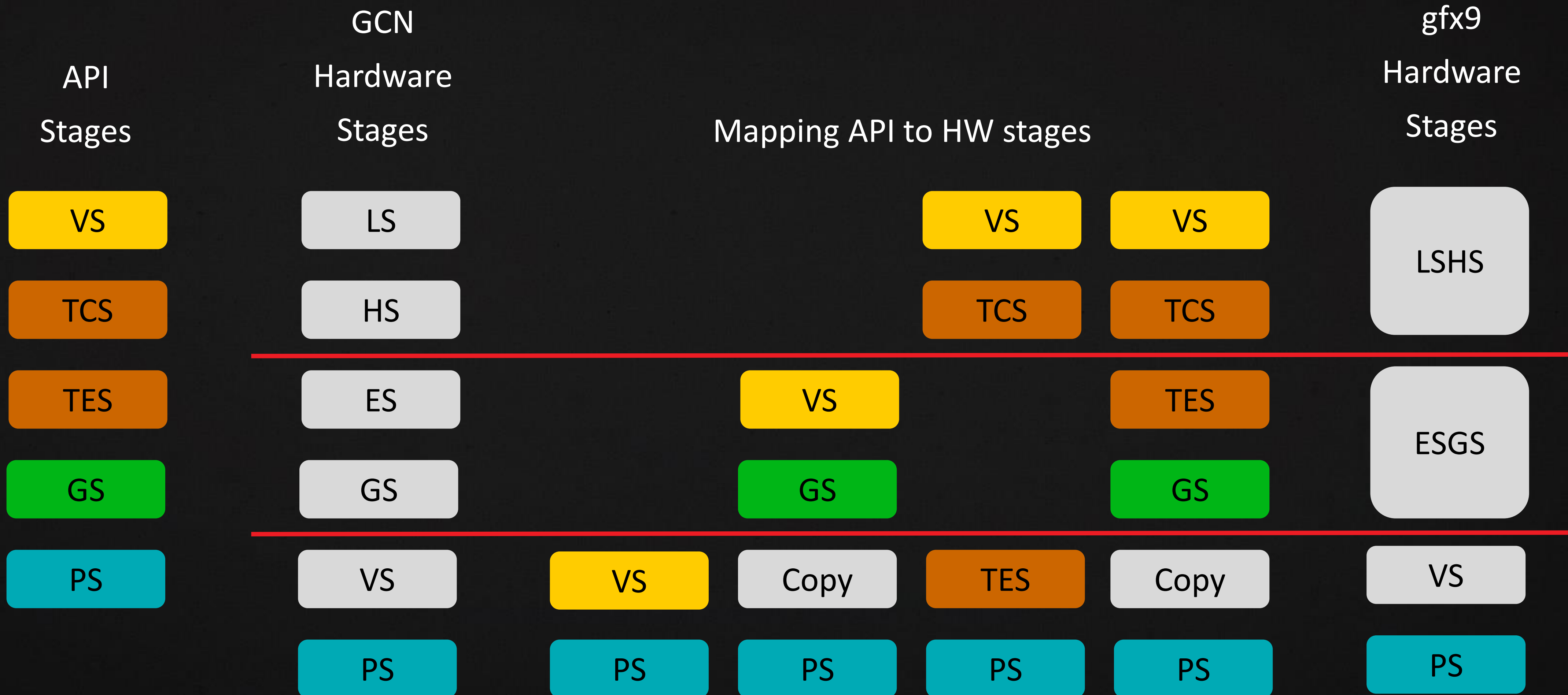
```
v_cvt_pkrtz_f16_f32     v0, v0, v1
v_cvt_pkrtz_f16_f32     v1, v2, v3

exp mrt0 v0, v0, v1, v1 done compr vm
s_endpgm
```

Depends on the current framebuffer state
"Epilog"

- Compile the main shader part once, prolog/epilog on demand
  - Much faster than recompiling entire shader on state change

RADEON TECHNOLOGIES GROUP

SHADER STAGES IN THE GRAPHICS PIPELINE



VERTEX AND GEOMETRY SHADERS AS MERGED ESGS

- Output of vertex shader:

```
out vec4 a;
out float b[5];
```

- Input of geometry shader:

```
in vec4 a[];
in float b[][5];
```

  – Arrays contain inputs for each vertex of a triangle (or other primitive)

- Transfer data from vertex lanes to primitive lanes
  – Vertex shader stores outputs in Local Data Share (LDS)
  – Geometry shader loads inputs from LDS

MERGED ESGS: LDS ADDRESSING

- Addresses are calculated manually by the frontend

v0.attr0.x	v0.attr0.y	v0.attr0.z	v0.attr0.w	v0.attr1.x	v0.attr1.y	v0.attr1.z	v0.attr1.w
(gap)	v1.attr0.x	v1.attr0.y	v1.attr0.z	v1.attr0.w	v1.attr1.x	v1.attr1.y	v1.attr1.z
v0.attr1.w	(gap)	v2.attr0.x	v2.attr0.y	v2.attr0.z	v2.attr0.w	v2.attr1.x	v2.attr1.y

- LLVM is not aware of LDS use
  - Cannot use LDS for spilling (LDS is small, but still…)
  - Cannot use LDS for dynamically indexed arrays
    - Sometimes, LDS might be more convenient than VGPR indirect addressing
  - Difficult to extend LDS use even in the frontend
  - Alias analysis may become less effective

LDS LINKING

- Goal: Explicitly represent all data in LDS
- It's not that simple for attribute memory (both ESGS and LSHS):
  - –Mismatches in the number of attributes (VS produces unused outputs)
  - –Number of waves (and thus vertices) per workgroup not known in advance
- MVP: Allow LDS variables in addition to attribute memory
  - –Attribute memory is an external variable of unknown size
  - –New ELF relocation type for patching LDS instructions
  - –Additional instructions required unless attribute memory assumed to be at 0
- Later: Representing attribute memory explicitly?
  - –Problem: it's a two-dimensional array with unknowns in both dimensions

RADEON TECHNOLOGIES GROUP

.RODATA LINKING

- Comparatively straightforward – like on CPUs
- Different ABI possibilities:
  - Free 64-bit addresses
  - .rodata restricted to a single 32-bit address space
    - Like the new constant address space for descriptors (Marek Olšák's patches)
  - .rodata and .text restricted to be in the same 4GB-aligned space
    - Fill in high 32-bits of addresses from PC_HI

SUMMARY

- Switching to NIR in radeonsi is pretty far along
- Explore proper linking of shader parts
  - .rodata
  - Local Data Share
  - What linker should we use?
    - LLD is a natural choice and embeddable, but we really “only” need a dynamic linker

RADEON TECHNOLOGIES GROUP

THANK YOU

DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners. Battlefield 4 images and logos © 2018 Electronic Arts Inc. Battlefield, Battlefield 4 and the DICE logo are trademarks of EA Digital Illusions CE AB. EA and the EA logo are trademarks of Electronic Arts, Inc.