Register Analysis for General Instrumentation of AMDGPU Kernels

Hsuan-Heng Wu

Computer Sciences Department
University of Wisconsin

Scalable Tools Workshop
August 12, 2024

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Paradyn

Motivation

Binary instrumentation is a generally useful technique.

There are currently no binary instrumentation tools for the various AMD GPU architectures.

So our group started a Dyninst port for these architectures.

De-Motivation

- Oh, but wait! Conventional wisdom says that it won't work. For sophisticated kernels like in MIOpen, there's no general purpose stack and few free registers … no way to do instrumentation.
- And if you look at the various headers in the GPU code objects, this does seem to be the case.
- Folks at AMD and the national labs all said the same thing.

But careful analysis with Dyninst, including register liveness analysis, and with some clever design, and it appears that is will work.

Background - Registers

- AMDGPU implements the SIMT model
- The same kernel is executed by multiple wavefronts, each consists of 64 threads (lanes)
  - Scalar register of a wavefront shared by all lanes
  - Vector registers of a wavefront have one value per lane
  - A vector register can be used to store 64 scalar register values
- Each wavefront is allocated a fixed number of registers determined by the compiler

So How Many Registers Do We Really Need for Instrumentation?

- Global register versus local register requirement
- A register is **globally available** if it is free at all points in the code
  - Registers are allocated in multiples of 4 or 8
  - Suitable for values such as stack base & offset and heap base.
- A register is **locally available** at an instruction if it is dead
  - Suitable for temporary variables used to implement instrumentation

So How Many Registers Do We Really Need? Cont'd

The availability of local registers is less of a problem

- If stack is available, we can make more registers available by spilling

So the real question is, how many registers do we need globally available for instrumentation?

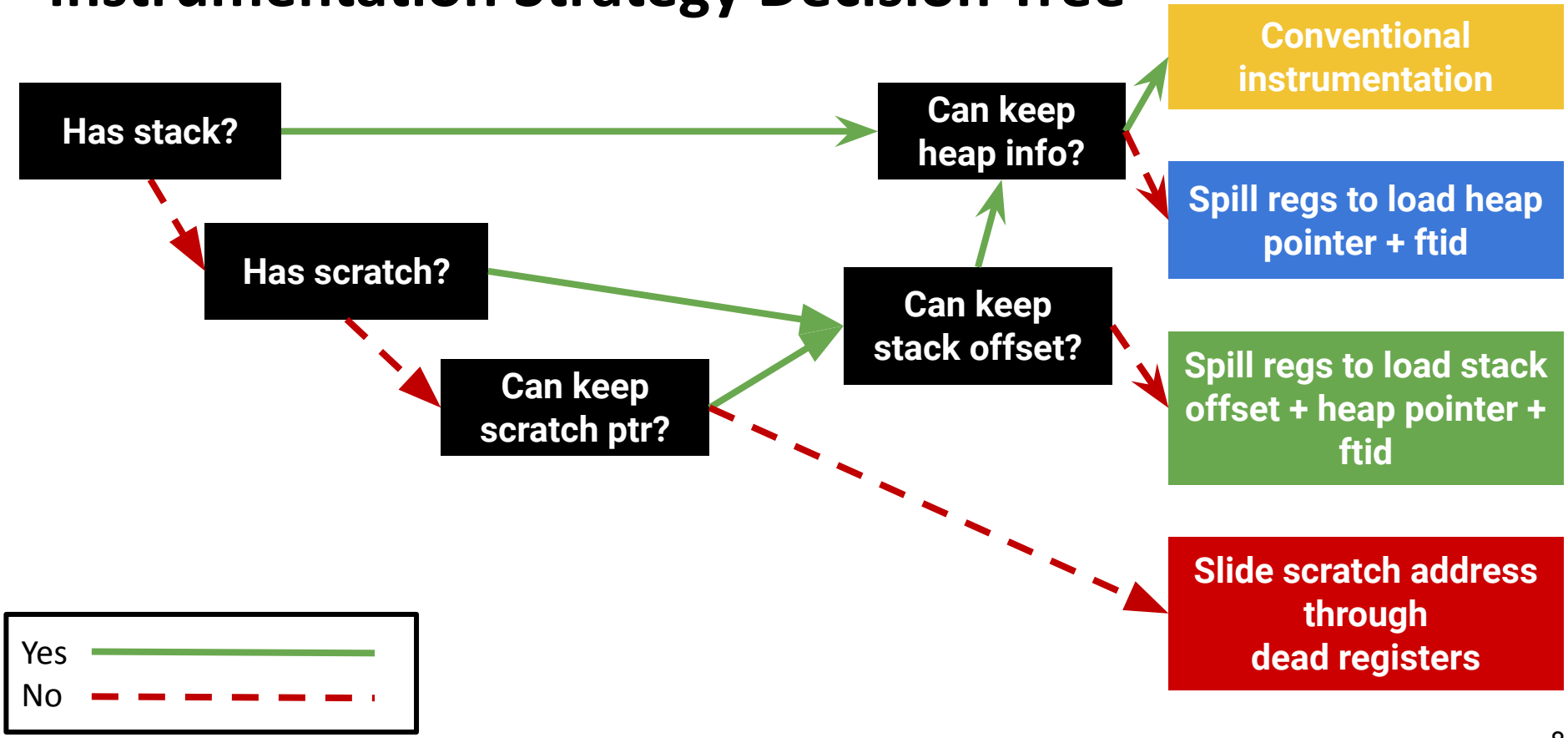
So How Many Registers Do We Really Need? Cont'd

We want enough global registers to hold

- Stack base address and offset
  - Used for function call and register spilling
- Heap base address and per-thread index
  - Used to collect instrumentation data
  - For efficiency, we want to keep the heap address in registers
  - If we don't have enough registers, we can save them in memory and restore them as necessary

Conventional instrumentation, like we use on CPUs assumes we have a general purpose stack and heap

Instrumentation Strategy Decision Tree



Instrumentation Strategy Decision Tree

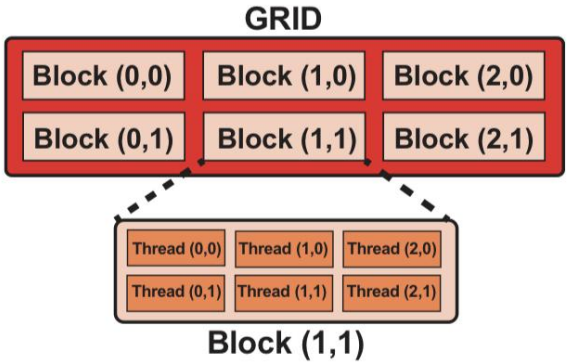
Can we keep stack info + heap info in global registers

Can we keep the stack info in registers

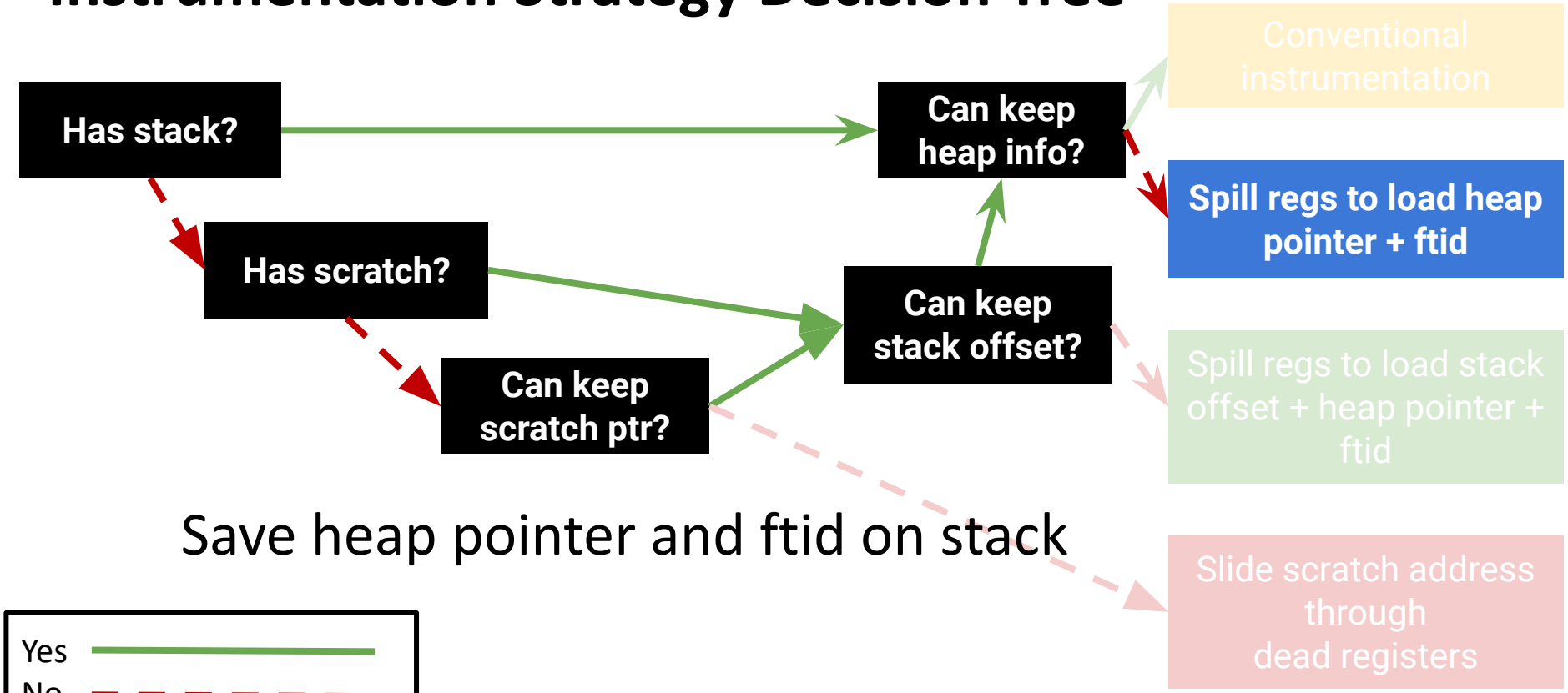
- Implemented with scratch memory
  - Per wavefront memory allocated on wavefront creation
- Need 1 pair of SGPR to hold the pointer to scratch memory (scratch pointer)
  - Each SGPR is 32 bit, and AMDGPU uses 48 bit addresses
- Need 1 SGPR to hold the offset from the scratch pointer (stack offset)

Can we keep heap info in registers?

- On launch, we allocate instrumentation memory - a fixed amount of memory per thread
- 2 SGPRs to hold the base address (heap pointer)
- 1 VGPR to hold a unique thread ID for indexing into the heap
  - Flatten the multi-dimensional tid space to 1-D **ftid**

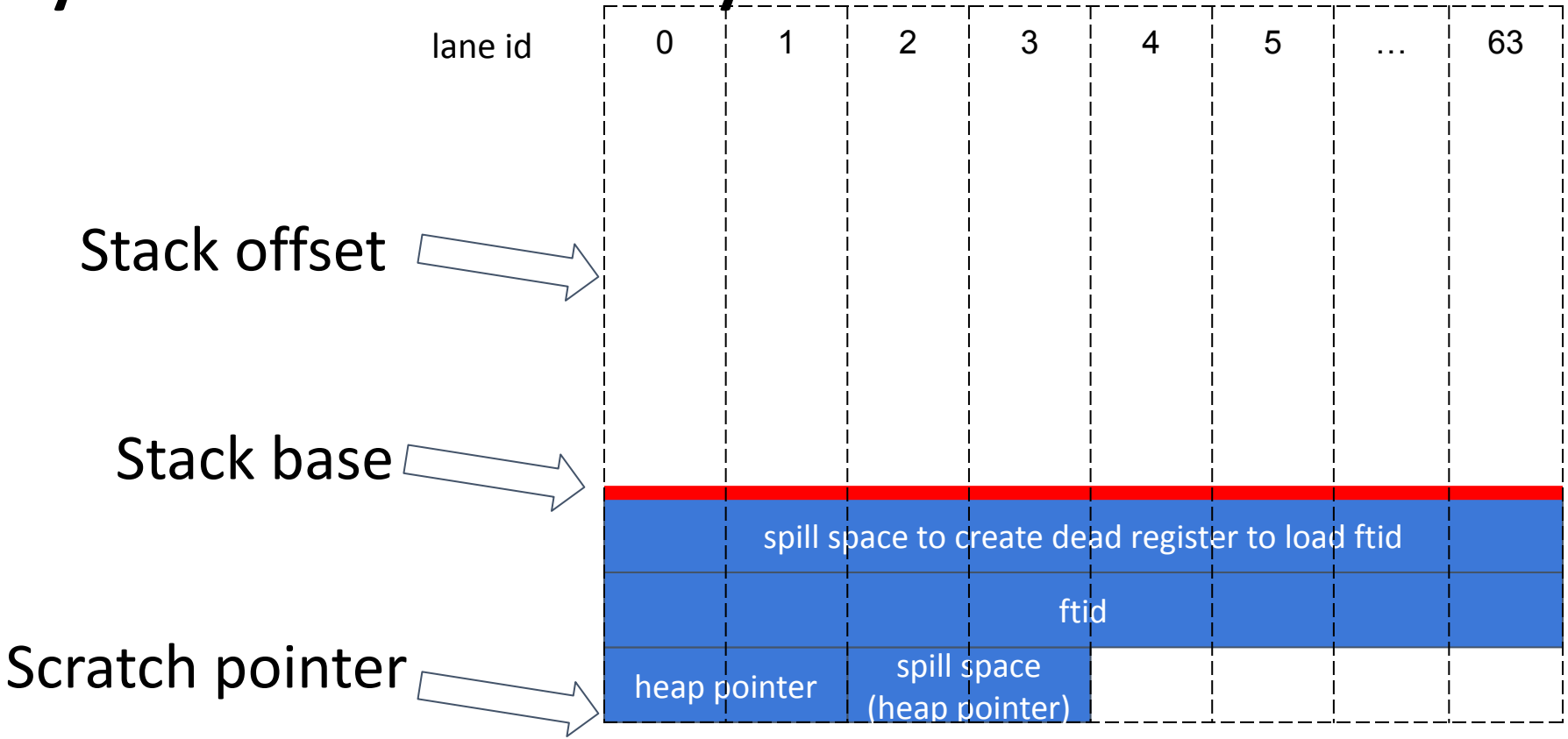


Instrumentation Strategy Decision Tree

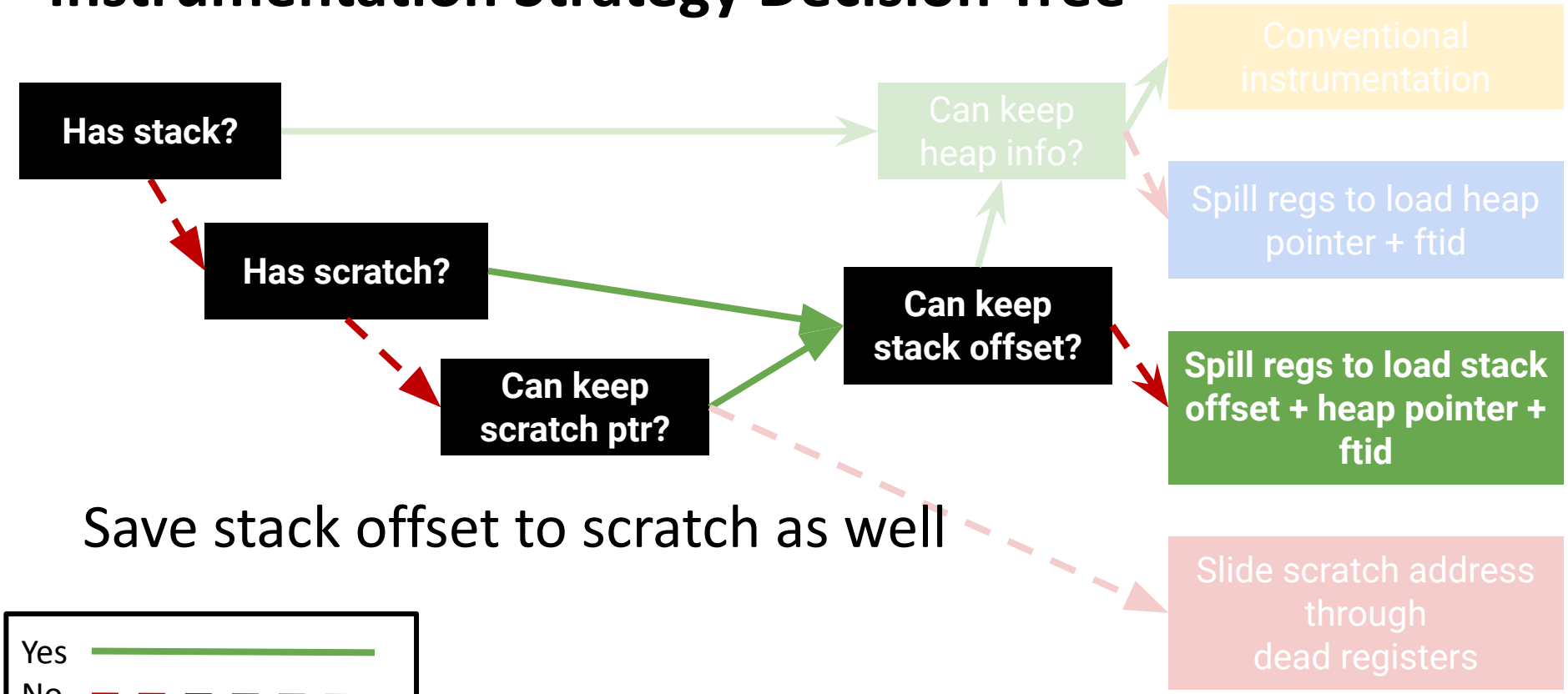


Save heap pointer and ftid on stack

Layout of Scratch Memory for a Wavefront

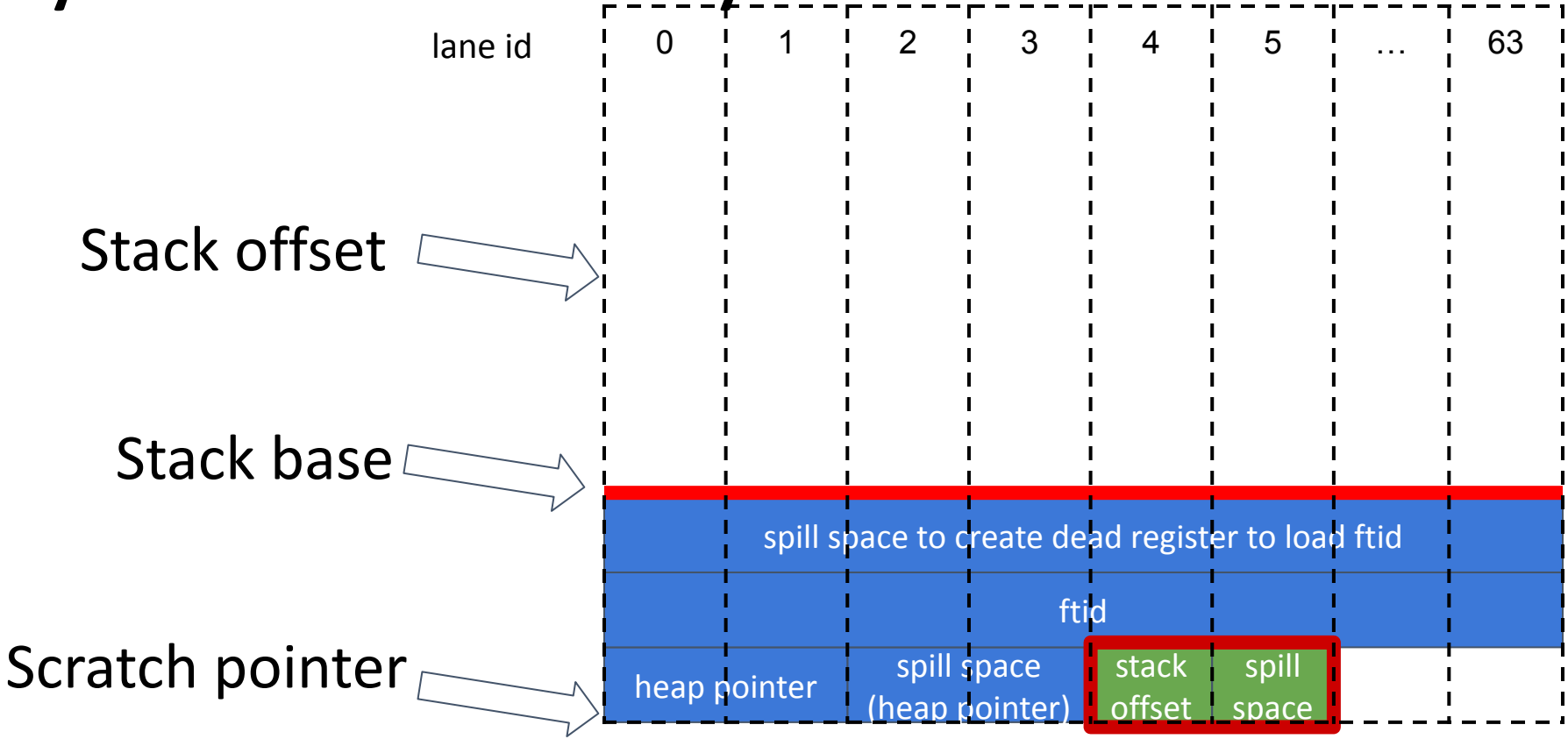


Instrumentation Strategy Decision Tree



Save stack offset to scratch as well

Layout of Scratch Memory for a Wavefront

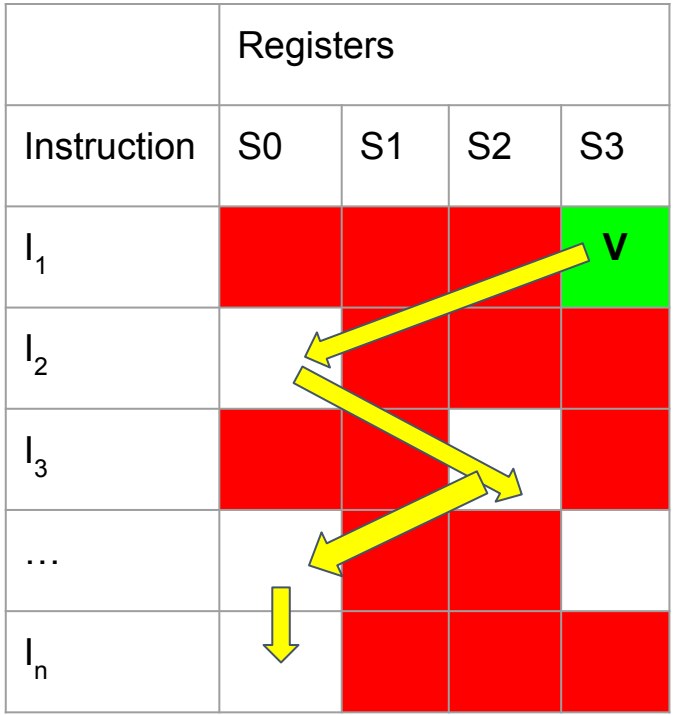


Instrumentation Strategy Decision Tree

The Sliding Tile Puzzle algorithm

- Goal is to keep the scratch address in registers throughout execution
- Requires 2 SGPRs available at every instruction, though not necessarily the same ones:
  - Can use dead registers
  - Can spill a register to the stack to free it
- Pass it through free registers
  - Use the control flow to determine where to move the value

Example: Sliding a Value Through Dead SGPR



Live

With Default Reg Allocation / With Increased Reg Alloc

Approach	Scratch		Can Reserve Stack Offset	Can Reserve Heap	Percentage of Kernels		
	Scratch Enabled	Can Enable Scratch			GFX908	GFX90A	GFX940
Conventional instrumentation	✓		✓	✓	0.0% / 0.3%	0.0% / 0.0%	0.0% / 0.0%
	X	✓	✓	✓	30.6% / 97.0%	11.2% / 97.4%	9.3% / 95.5%
Spill regs to load heap pointer and ftid	✓		✓	X	1.3% / 0.0%	1.0% / 0.0%	1.0% / 1.1%
	X	✓	✓	X	48.5% / 1.6%	68.2% / 1.5%	61.7% / 0.3%
Spill regs to load stack offset, heap pointer and ftid	✓		X	X	0.0% / 1.0%	0.0% / 0.0%	0.0% / 1.0%
	X	✓	X	X	2.4% / 0.0%	3.1% / 0.0%	4.6% / 0.3%
Slide Scratch pointer through dead regs	X	X	X	X	17% / 0.0%	16.2% / 0.0%	23.1% / 2.8%

Questions?

The minimum global register requirement

- As long as we can keep the stack base address in globally available registers
  - Save stack offset, heap pointer and ftid values at fixed offset relative to stack base
  - Spill registers at instrumentation site to create dead registers to load these values
- Otherwise, we try to slide the scratch pointer around through dead registers