Shade: A Fast Instruction-Set Simulator for Execution Profiling

Background: SPARC
- What are register windows? (motivation, function)
- What are delay slots? Why are they useful?
- Why do some instructions get annulled?
- What does the CALL instruction do? How is the return address saved?
  - How is this different from a JMPL (jump and link)?
  - How is this different from a RET instruction?

Basics
- In contrast to Shade, how would a typical "classic" instruction-set simulator operate?
- Describe, in contrast, the main operation of Shade; how does it work?
  - What is the "translation cache"? What is its structure and how is it used by Shade?
  - What is the "TLB"? What is its structure and how is it used by Shade?
  - Why is the notion of a "basic block" useful?
- Figures 2a and 2b show an example translation (without tracing);
  - How fast would Shade run if each instruction were translated like this?
    (how does Shade gain back some efficiency?)
- How does Shade add tracing into translations?
  - How is this done efficiently?

Chaining
- What is chaining?
- How does chaining work given the main loop in Figure 1?
- Follow the paths through the code for chaining when…
  - … the successor was translated first
  - … the predecessor was translated first

Virtualization
- How are registers virtualized?
- What does Shade do in order to make this virtualization efficient?
- How is memory virtualized?
  - How is this different than the virtualization of memory we've discussed in other papers?
- What are condition codes?
  - What makes them hard to virtualize?
  - What is Figure 4 all about?
- What does Shade do on system calls?

Performance
- What do we learn from Figure 6?
- Figure 7?
- Figure 8?
- Anything else that interested you?

Other
- Cross-shades: what and why? Useful in VMM setting?

Overall
- What is the coolest thing about Shade?
- What would it take to make Shade a VMM?
- Does Shade still exist?