Today’s Paper

- Summarize
- Trace-based compilation (why? How?)
  - Mixed mode
  - Two assumptions
Today’s Paper

- Summarize
- Trace-based compilation (why? How?)
  - Mixed mode
  - Two assumptions
    - Most time spent in hot loops
    - Hot loops are mostly type-stable
  - Traces are extended bb’s – they have multiple exits
    - Single entry (at the top)
      - No joins within (makes optimizing much simpler)
    - Functions inlined (a bit complex, b/c we have to still record frames)
Today’s Paper

• Summarize
• Trace-based compilation (why? How?)
  ■ Mixed mode
• Contributions
Today’s Paper

• Summarize
• Trace-based compilation (why? How?)
  ▪ Mixed mode
• Contributions
  ▪ Focus on tracing loops only
    ▸ Nested trace trees
    ▸ Why? How does an outer get to an inner trace?
Today’s Paper

• Summarize

• Trace-based compilation (why? How?)
  ■ Mixed mode

• Contributions
  ■ Focus on tracing loops only
    ‣ Nested trace trees
    ‣ Why? How does an outer get to an inner trace?
    ‣ Option1: trace inner (hot first), continue tracing through outer back to inner (code duplication for outer for every side exit and type)
      ◆ Will overflow the code cache
    ‣ Option2: give up and only trace inner’s
    ‣ New option: keep track and separate inner and outer; have outer “call” inner
Today’s Paper

• Summarize

• Trace-based compilation (why? How?)
  ■ Mixed mode

• Contributions
  ■ Focus on tracing loops only
    ▶ Nested trace trees
    ▶ Why? How does an outer get to an inner trace?
  ■ Real implementation

• What loop is considered hot?
Today’s Paper

• Summarize

• Trace-based compilation (why? How?)
  ■ Mixed mode

• Contributions
  ■ Focus on tracing loops only
    ▸ Nested trace trees
    ▸ Why? How does an outer get to an inner trace?
  ■ Real implementation

• What loop is considered hot?

• Monitor, Recorder

• Guards
  ■ Different control path or different types used on path
Today’s Paper

- Optimizations
  - Floating point to integer emulation on non-fp architectures
  - CSE
  - Simplification, constant folding, strength reduction
  - Source-language optimizations (Double -> int)
  - Dead stack stores, call stack stores, code elimination
- Blacklisting (Why? What? How?)
Today’s Paper

- Optimizations
  - Floating point to integer emulation on non-fp architectures
  - CSE
  - Simplification, constant folding, strength reduction
  - Source-language optimizations (Double -> int)
  - Dead stack stores, call stack stores, code elimination

- Blacklisting (Why? What? How?)
  - Some traces won’t finish (exceptions)
    - So recording them is a waste
    - Set a counter so that you don’t record them for awhile
    - Try again, then if untraceable, mark in bytecode (bad loop header)
    - Trace lookup in blacklist is time consuming