# SimpleScalar Tutorial

**(for tool set release 2.1)**

**Todd Austin**
**MicroComputer Research Labs**
**Intel Corporation**
**taustin@ichips.intel.com**

**Doug Burger**
**Computer Sciences Department**
**University of Wisconsin-Madison**
**dburger@cs.wisc.edu**

# Tutorial Overview

- Overview and basics
- Using the tool suite
- How to use sim-outorder
- How to build your own simulators
- How to modify the ISA
- How to use the memory extensions
- SimpleScalar limitations and caveats
- Wrapup

# Motivation for tutorial

- Free
  - Commercial restrictions apply
- Promote the tools
- Facilitate building a user community
  - Lots of exciting things happening
- Underscore limitations

# Advantages to SimpleScalar

- Extensible
  - source for compiler, libraries, simulators
  - user-extensible instruction format
- Portable
  - runs on NT and most UNIX platforms
  - target can support multiple ISAs
- Detailed
  - Interfaces support simulators of arbitrary detail
  - Multiple simulators included with distribution
- Fast (millions of instructions per second)

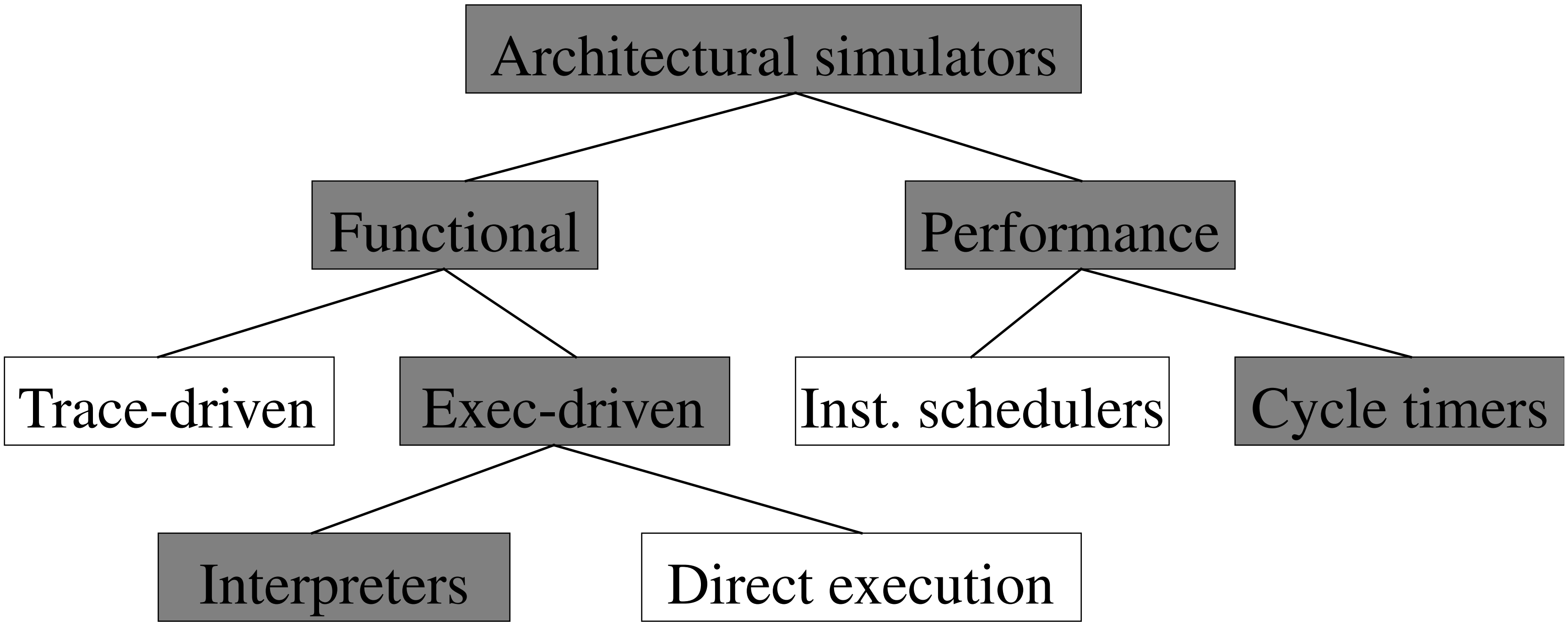
# Goal - large-scale research infrastructure

- Create large user base
  - Makes extensions available to all
- Software simulators now are ad hoc
  - Common reference platform
  - Reproducible results
  - Reduces research overhead
  - Whole community benefits

# Upcoming in SimpleScalar

- Major release (3.0) in spring/summer
  - Memory extensions
  - Multiprocessor simulator
  - Value prediction/trace caches?
- More instruction sets
- Kernel extensions (Linux?)

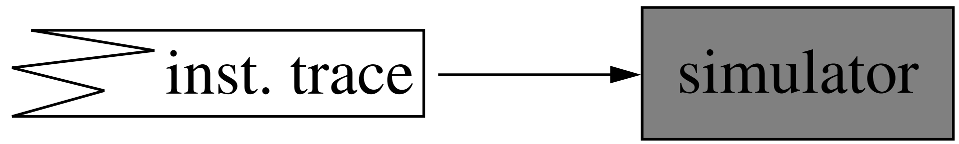
# A Taxonomy of Simulation Tools




- shaded tools are part of SimpleScalar

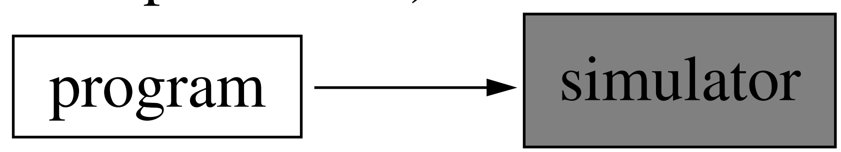
# Execution- vs. Trace-driven Simulation

- trace-based simulation:




  - reads a "trace" of insts saved from previous execution
  - easiest to implement, no functional component needed
  - no feedback into trace (e.g. mis-speculation)

- execution-driven simulation:




  - simulator "runs" the program, generating stream dynamically
  - more difficult to implement, many advantages
  - direct execution: instrumented program runs on host

# Functional vs. performance simulators

- Functional simulators implement the architecture
  - Perform the actual execution
  - Implement what programmers see
- Performance (or timing) simulators implement the microarch.
  - Model system resources/internals
  - Measure time
  - Implement what programmers do not see

# Instruction schedulers vs. cycle timers

- constraint-based instruction schedulers
  - simulator schedules instructions based on resource availability
  - instructions processed one at a time, in order
  - simpler to implement/modify, generally less detailed
- cycle-timer simulators
  - simulator tracks microarch. state each cycle
  - many instructions in various stages at any time
  - simulator state == microarch. state
  - good for detailed microarchitecture simulation

# Tool set history/acknowledgments

- Sohi's CSim begat Franklin's MSim begat SimpleScalar
  - Written by Todd Austin
  - First public release in July '96, with Doug Burger
- Other contributors:
  - Kevin Skadron (branch prediction)
  - Alain Kägi (early work on memory extensions)
  - Many others for bug fixes and suggestions
- Thanks to industrial and government funding sources

# SimpleScalar resources

- Public release (2.0 and back) available from UW-Madison
  - http://www.cs.wisc.edu/~mscalar/simplescalar.html
- Technical report:
  - *The SimpleScalar Tool Set, Version 2.0*
  - UW-Madison Tech report #1342, July, 1997.
- SimpleScalar mailing list
  - e-mail majordomo@cs.wisc.edu
  - message body contains “subscribe simplescalar”
- If all else fails, mail the maintainers (dburger, taustin)

# Goals for today

- Good high-level overview for new participants
- Details for more experienced participants
- Enumerate limitations of tools
- Answer questions - ask any time!
- FEEDBACK - how can we improve the tools and support?