Do Developers Understand IEEE Floating Point?

Peter Dinda Conor Hetland
Prescience Lab
Department of EECS
Northwestern University

pdinda.org
presciencelab.org

Survey Available Here,
Please Participate!

Paper in IPDPS 2018

Paper in a Nutshell: **Not Really**

- Targeted survey
  - Aimed at practitioners likely to use FP
  - Quizzes for core, optimization, and suspicion of results
  - **First study of this kind**
- Participants do only slightly better than chance on core concepts
  - … and don't know it
  - Some factors mitigate, but none particularly well
- Participants do not understand optimization concepts
  - … and do know it
- Participants less suspicious than they should be
  - … but similar to students in a sophomore course
- Maybe systems software can do something about it

Outline

- Motivation
- Study design
- Participant selection and factors
  - Important caveat!
- Core concepts
- Optimization concepts
- Suspicion of results
- What to do?
  - What are we doing?

For a Long Time...

Developer	Focused on scientific and engineering uses, Some understanding of numerical methods Assumption/understanding of IEEE floating point
IEEE 754(-2008) Standard	Stable, pretty much universal standard since early 1980s Considerable complexity
Compiler (Optimizations)	Small set of compilers used, slow change, difficult to break IEEE compliance
Hardware (Optimizations)	Small set of hardware, IEEE compliance universal, slow change

The Concerns Now

Developer	**Dramatic expansion** in uses *(e.g., machine learning, analytics, big data, and other expanding uses of FP)* Less knowledge of numerical methods, and the standard
IEEE 754(-2008) Standard	Stable, pretty much universal standard since early 1980s Considerable complexity
Compiler (Optimizations)	**Fast evolution** *(e.g., numerous compilers, automatic precision reduction, approximate computing, optimization flag choice, automatic optimization setting search, power/energy)*
Hardware (Optimizations)	**Fast evolution** *(e.g., hardware diversity (GPUs, FPGAs, ARM), half-floats, different denorm handling, non-IEEE compliance, power/energy)*

Do Developers Understand….

Developer

IEEE 754(-2008) Standard

Compiler (Optimizations)

Hardware (Optimizations)

Dramatic expansion in uses *(e.g., machine learning, analytics, big data, and other expanding uses of FP)* Less knowledge of numerical methods, and standard

Core Focus

Stable, pretty much universal standard since early 1980s
Considerable complexity

Optimization Focus

Fast evolution *(e.g., numerous compilers, automatic precision reduction, approximate computing, optimization flag choice, automatic optimization setting search, power/energy)*

Fast evolution *(e.g., hardware diversity (GPUs, FPGAs, etc.), half-floats, different denorm handling, non-IEEE compliance, power/energy)*

… and **Suspicion**

Study Design

- **Anonymity**
- **Factor identification**
- Low time commitment

- Survey instrument (web-based)
  - Participant background (for factor analysis)
  - Core quiz
  - Optimization quiz
  - Suspicion quiz

- Closed for study reported here, but open again now
  - http://presciencelab.org/float

Study Design

- Approximation of practice
  - Pose questions that might arise during software development
- Avoid prompting or anchoring
  - Don’t test if they remember terminology, test if they can see the concept
    - In a snippet of code…
    - In a choice of optimization option...
    - In an intern’s question...

Core Quiz

- Floating point arithmetic **is not real number arithmetic**, even though it looks like it
  - Commutativity, associativity, distributivity, ordering, identity, negative zero, overflow, NaN, operation precision, denormalized numbers, signaling…
- Floating point does not behave like computer integer arithmetic either...
  - Overflow (saturation), underflow, NaN, signaling

Floating Point Questions

The following questions ask about your understanding of floating point behavior in code. The syntax used is C/C++, and the syntax in Java or C# is identical or similar.

Assume that variables a, b, and c are floating point variables.

== is the equality operator in C/C++

if a and b are numbers, it is always the case that (a + b) == (b + a)

- ○ True, it is always the case
- ○ False, sometimes it is not the case
- ○ I don't know

Optimization Quiz

- Hardware features change standard compliance
  - MADD, Flush-to-Zero
- Compiler optimizations change standard compliance
  - What's the highest `-O` level that is standard compliant?
  - Is `--fast-math` standards compliant?
- Options and features can break compliance

Floating Point Optimizations

IEEE floating point behavior is standardized to help make analysis of floating point code tractable. Hardware implementations and compilers generally obey the standard, but it is possible for them to be configured not to. This is typically done in the context of compiler optimization, where some optimizations use non-standard behavior or non-standard elements of the hardware in order to compute faster.

This set of questions tests your understanding of what is standard behavior and what is not.

The floating point fused multiply-add operation integrates multiplication and addition into a single operation for faster operation and higher precision.

- ◯ It is part of the standard
- ◯ It is not part of the standard
- ◯ I don't know

Suspicion Quiz

- Floating point condition codes can point to numeric problems
- How suspicious should you be of your results when your code produces a…
  - Overflow, underflow, precision (rounding), invalid (NaN), or denormalized result
- Lack of suspicion may mean bad results get through

Floating point condition code interpretation

IEEE floating point hardware records exceptional conditions as they occur.   Using this capability, it is possible to determine whether any of a set of possible exceptional conditions occurred one or more times during the execution of a function.

Suppose you are given the following code:

```
int scientific_simulation()
{
      clear_floating_point_exceptional_conditions();

      do_scientific_simulation();

      conditions = get_floating_point_exceptional_conditions();
}
```

For each of the following exceptional conditions, indicate how suspicious you would be of the function do_scientific simulation()'s results if the condition occurred.   This is not a ranking.

There are no "right" answers for these questions.

Overflow - the result of some floating point instruction in do_scientific_simulation() was an infinity.

	1	2	3	4	5	
Low Suspicion	○	○	○	○	○	High Suspicion

Participant Recruitment Goals

- PhD student or above
- Actively involved in software development or management for science and engineering
  - Both as main and secondary roles
- Universities, national labs, and industry

Biggest Caveat: **Not a random sample**

Participant Recruitment Process

- Standardized email sent to seed recipients
  - Relevant department chairs, center directors, faculty, postdocs, and Ph.D. students at NU
  - Highest-level personal contacts at national labs
  - Faculty contacts at >20 universities
- Request to take survey and forward email only to people relevant to our recruitment goals

Participant Background / Factors

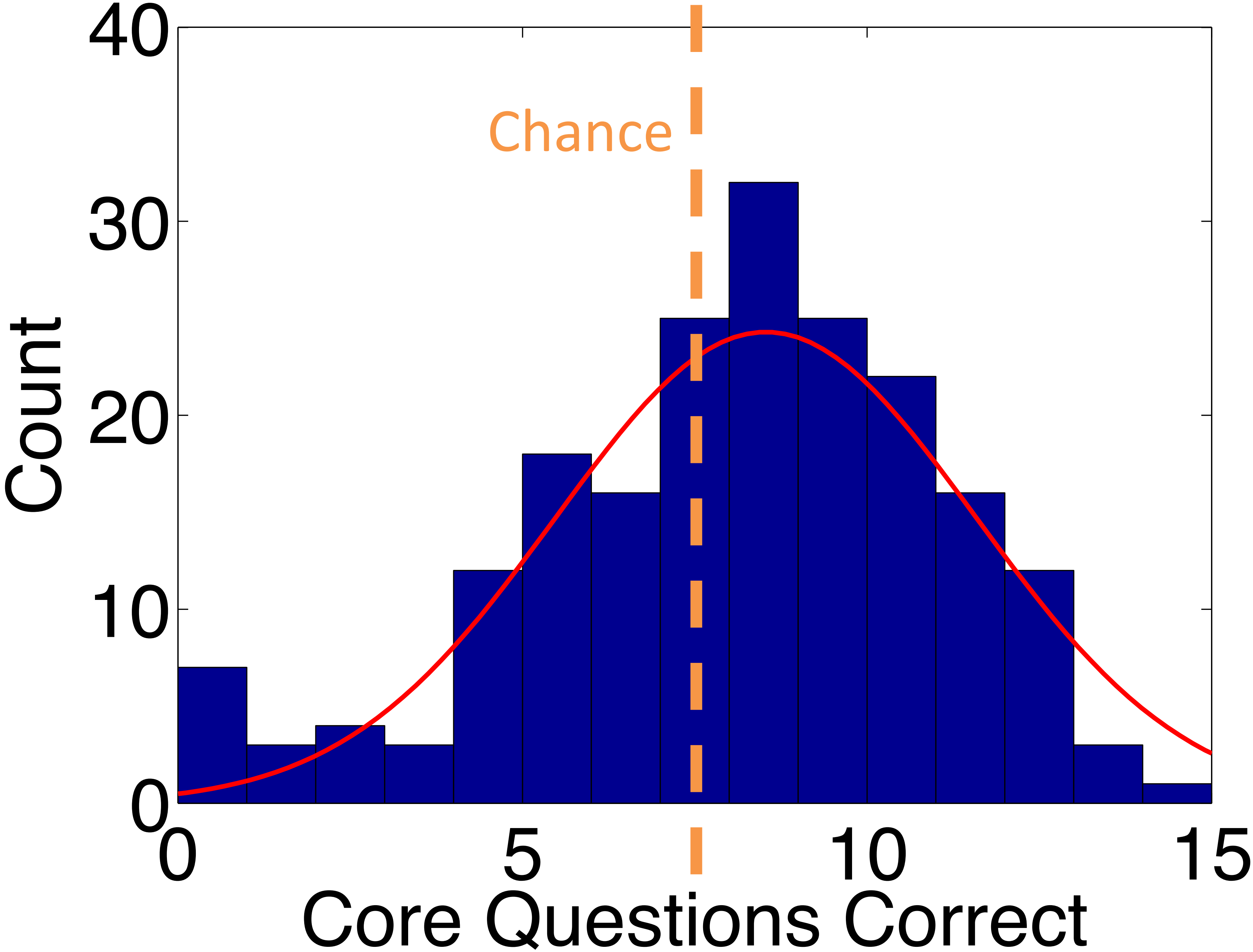
- **Anonymity**

- 199 Participants
  - Plus additional 52 undergrads for suspicion quiz

- 11 factors (self-reported)
  - 2 pages of details in paper

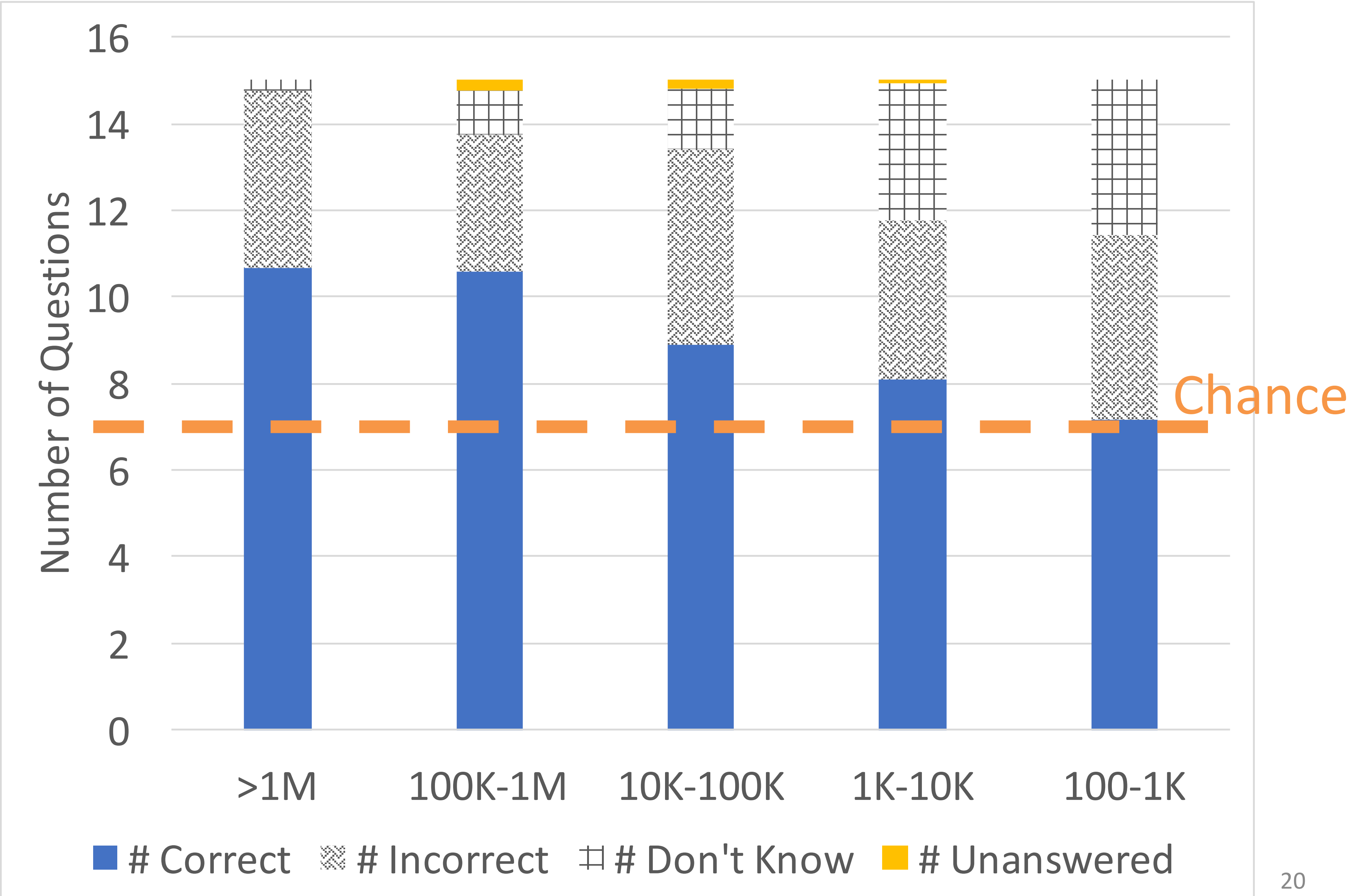
- Factors matter much less than expected
  - Will highlight a few as we go on

Prepare to be Scared

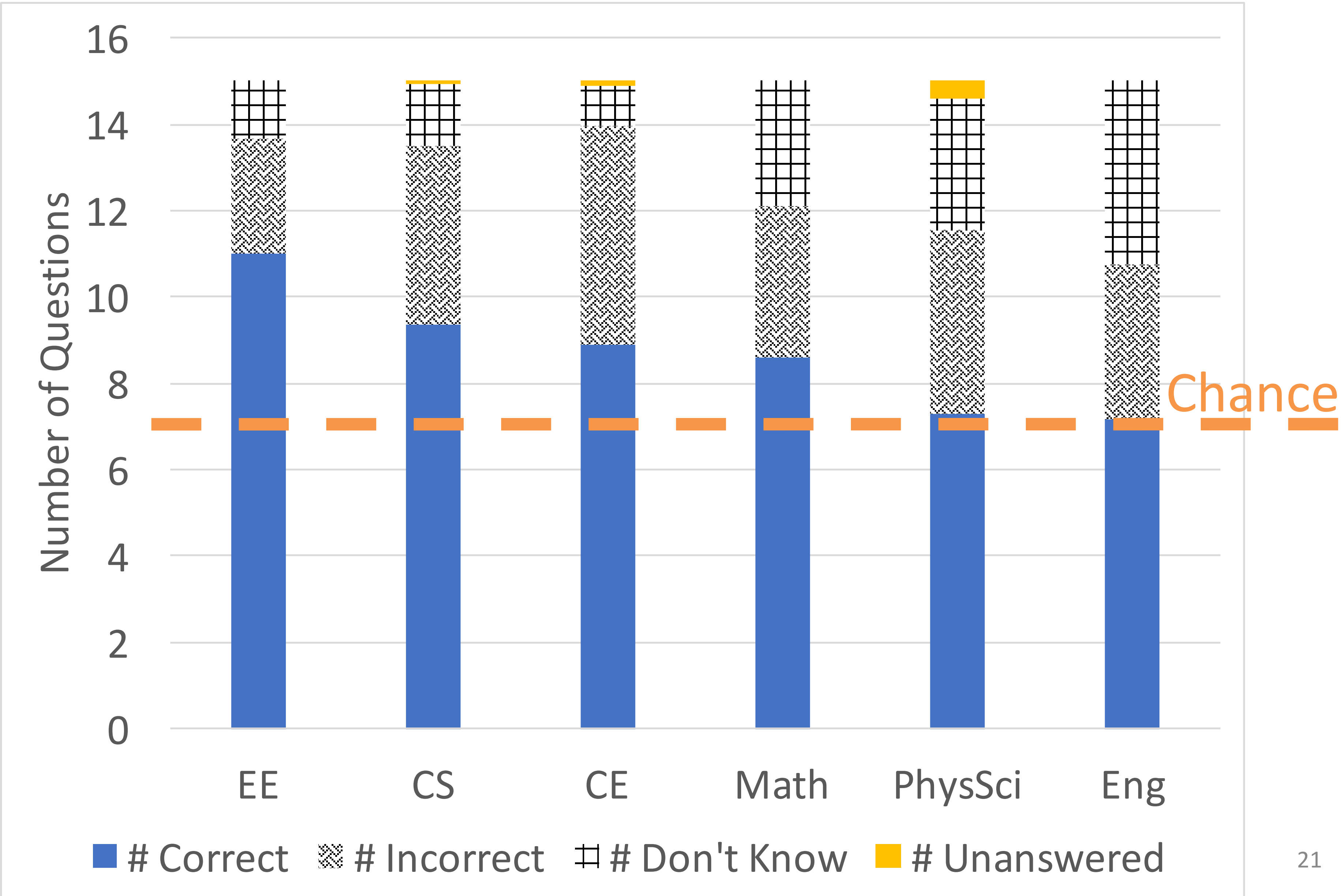


Chance
Count
0
10
20
30
40
0
5
10
15
Core Questions Correct

Experience With Code Matters (slightly)

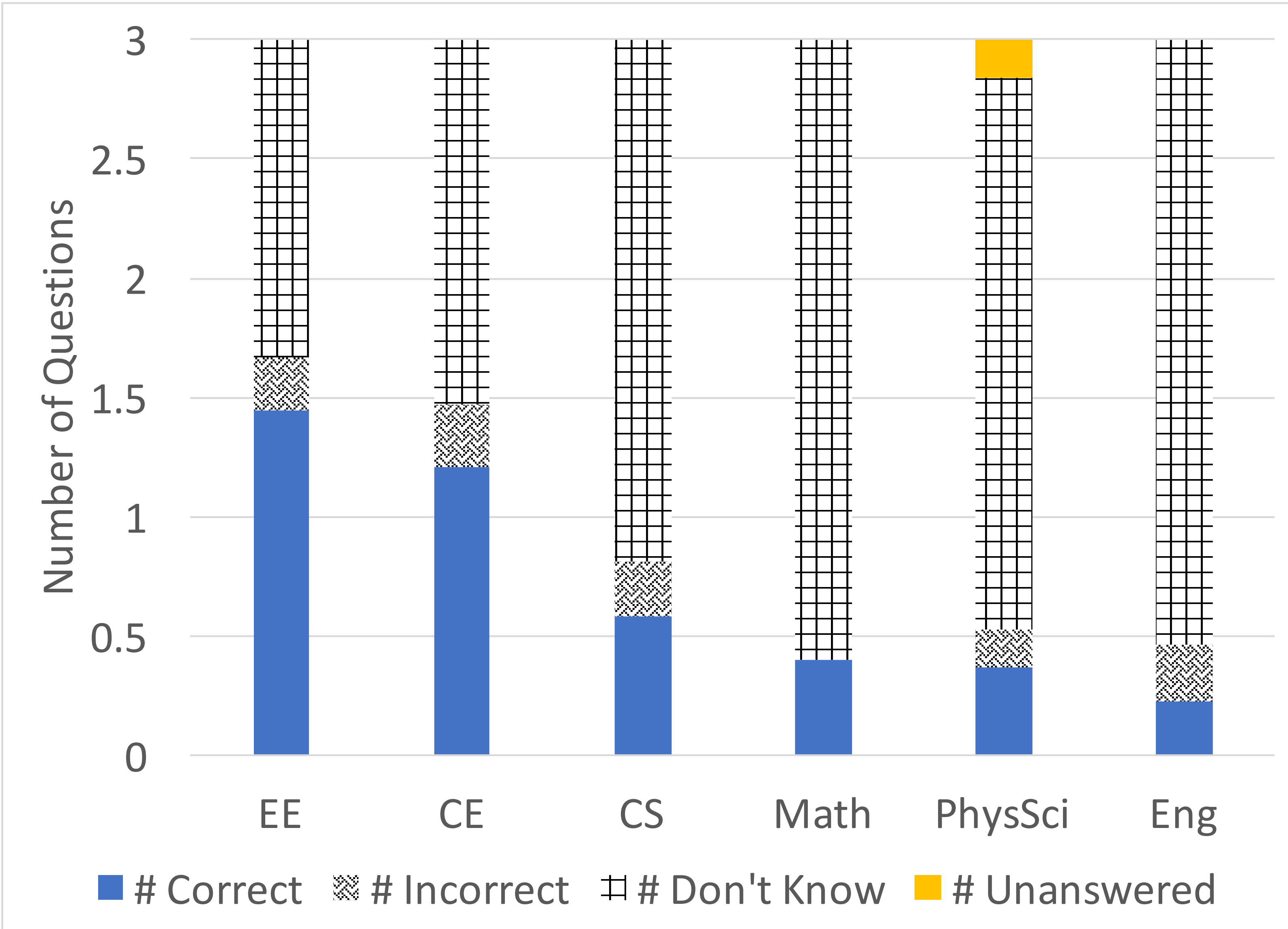


Area Matters (slightly)

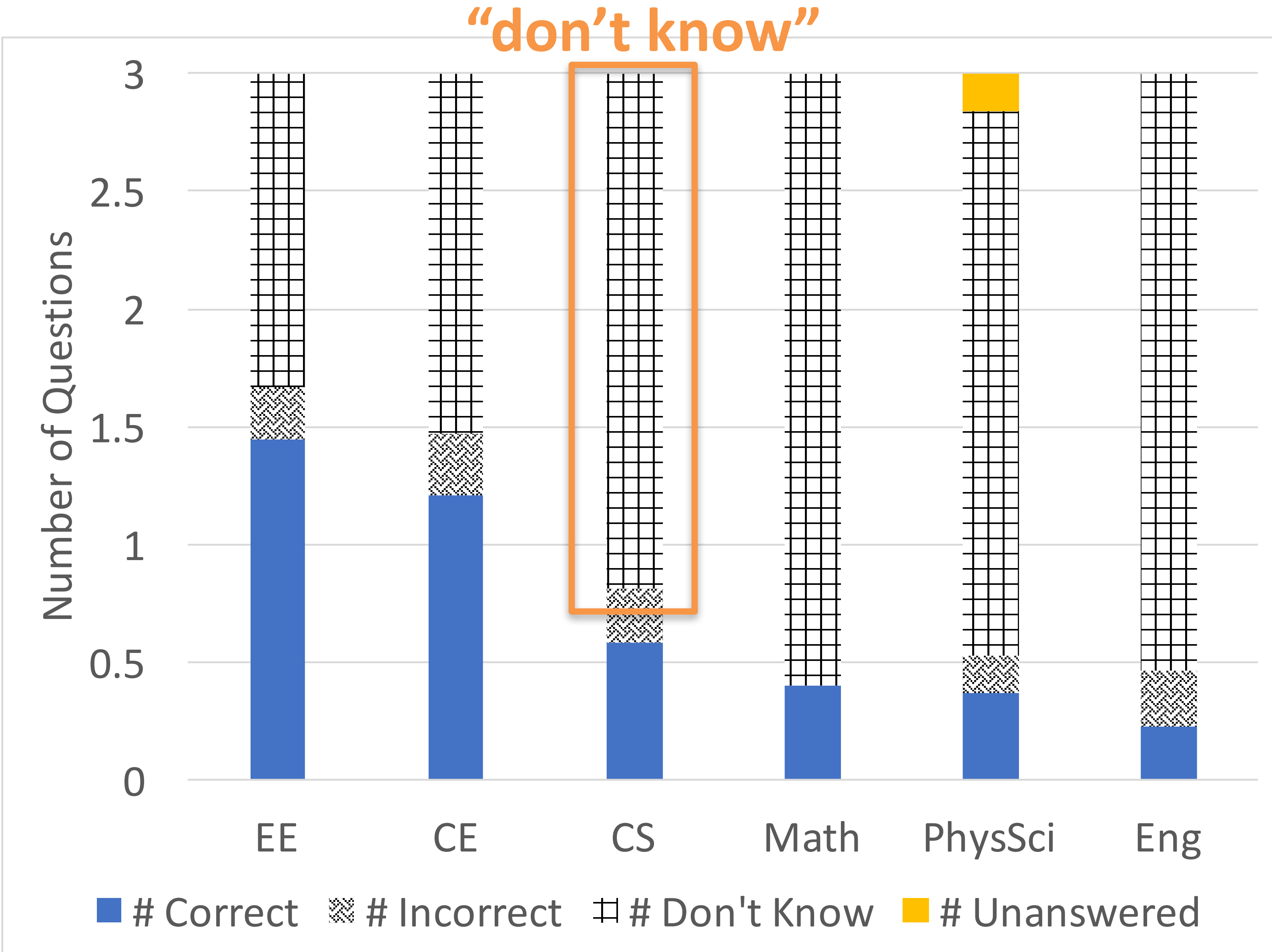


Now Some Good News for Correctness, but Bad News for Innovation

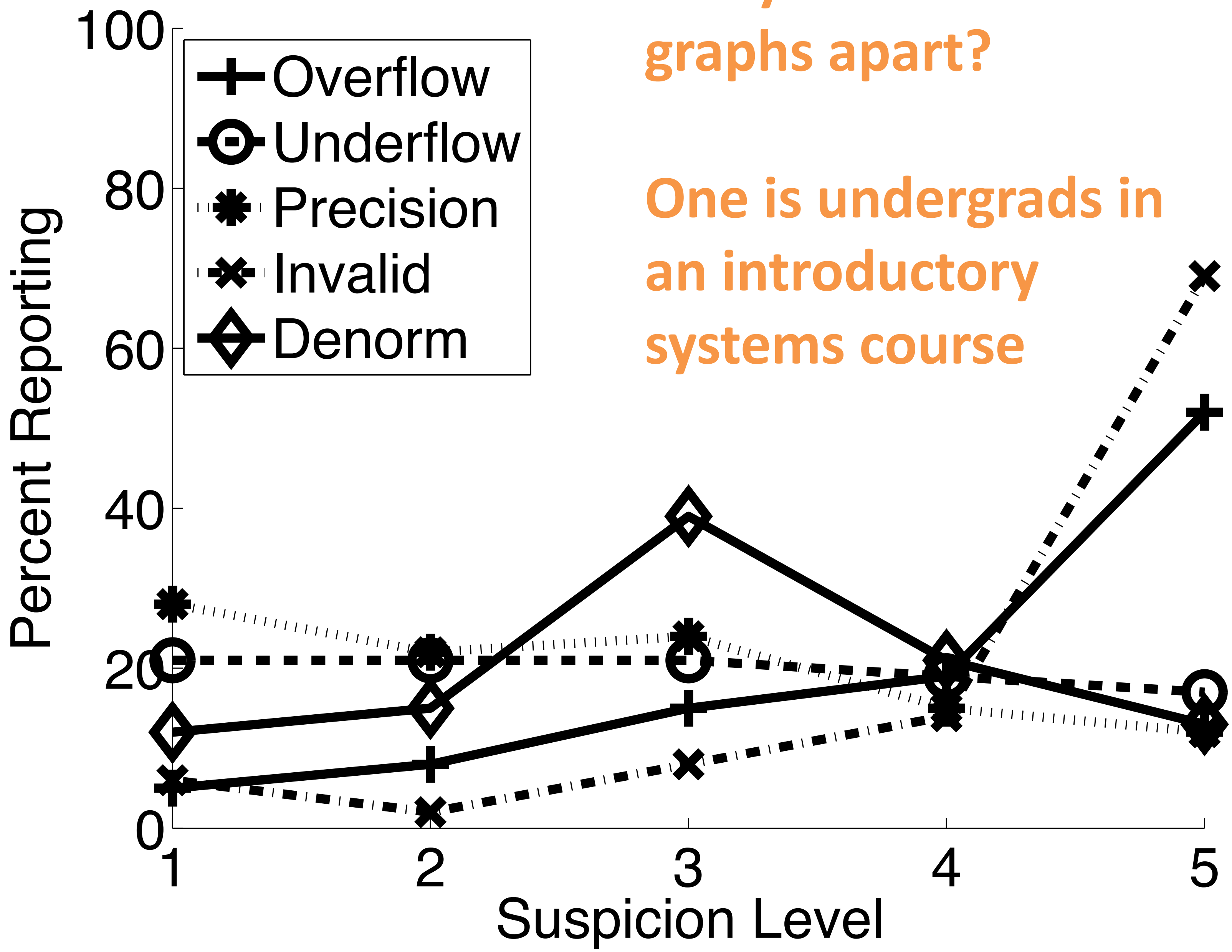
Participants Aware of Not Understanding Optimizations (HW/SW)



Participants Aware of Not Understanding Optimizations (HW/SW)

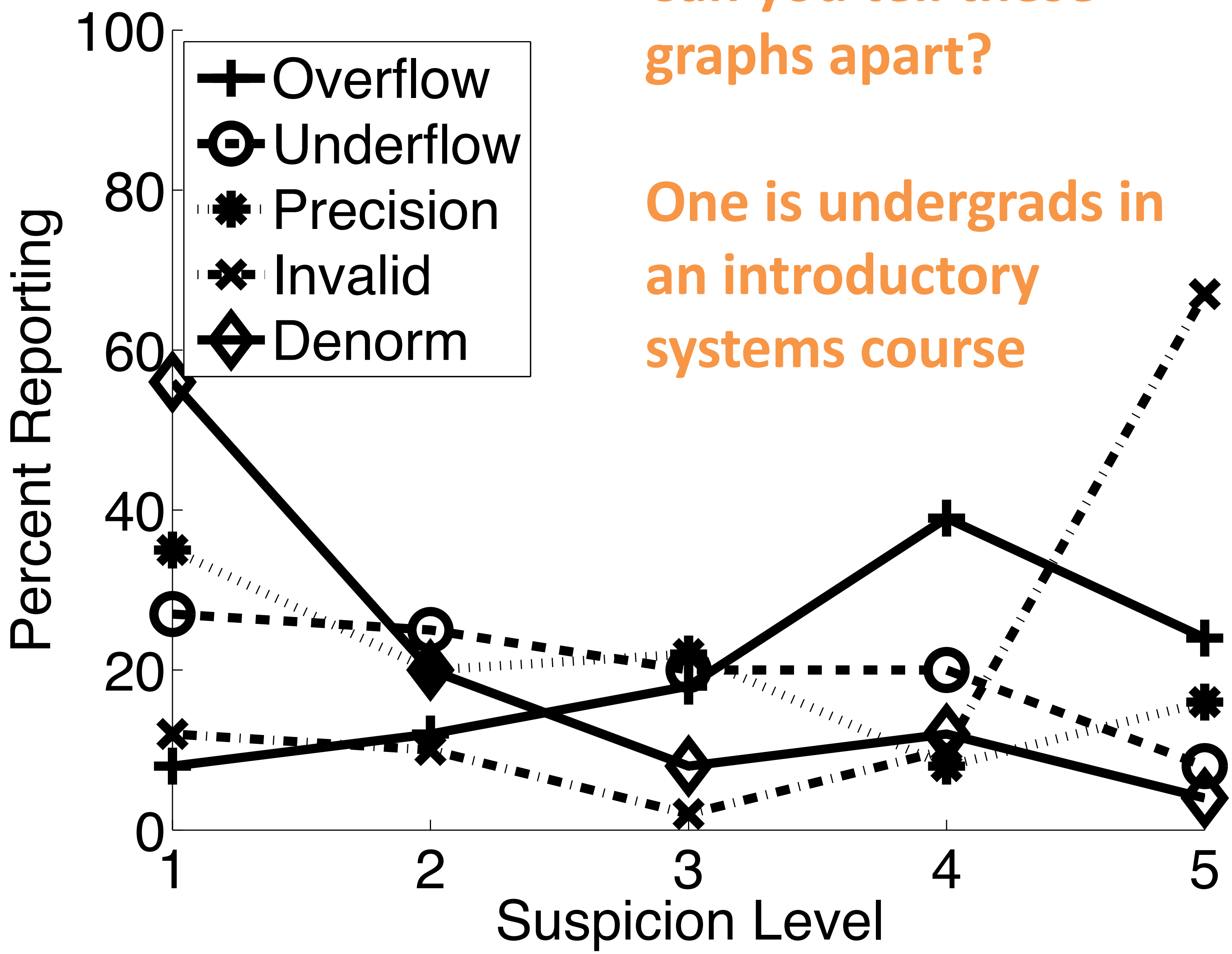


Now Some News that is Hard to Characterize



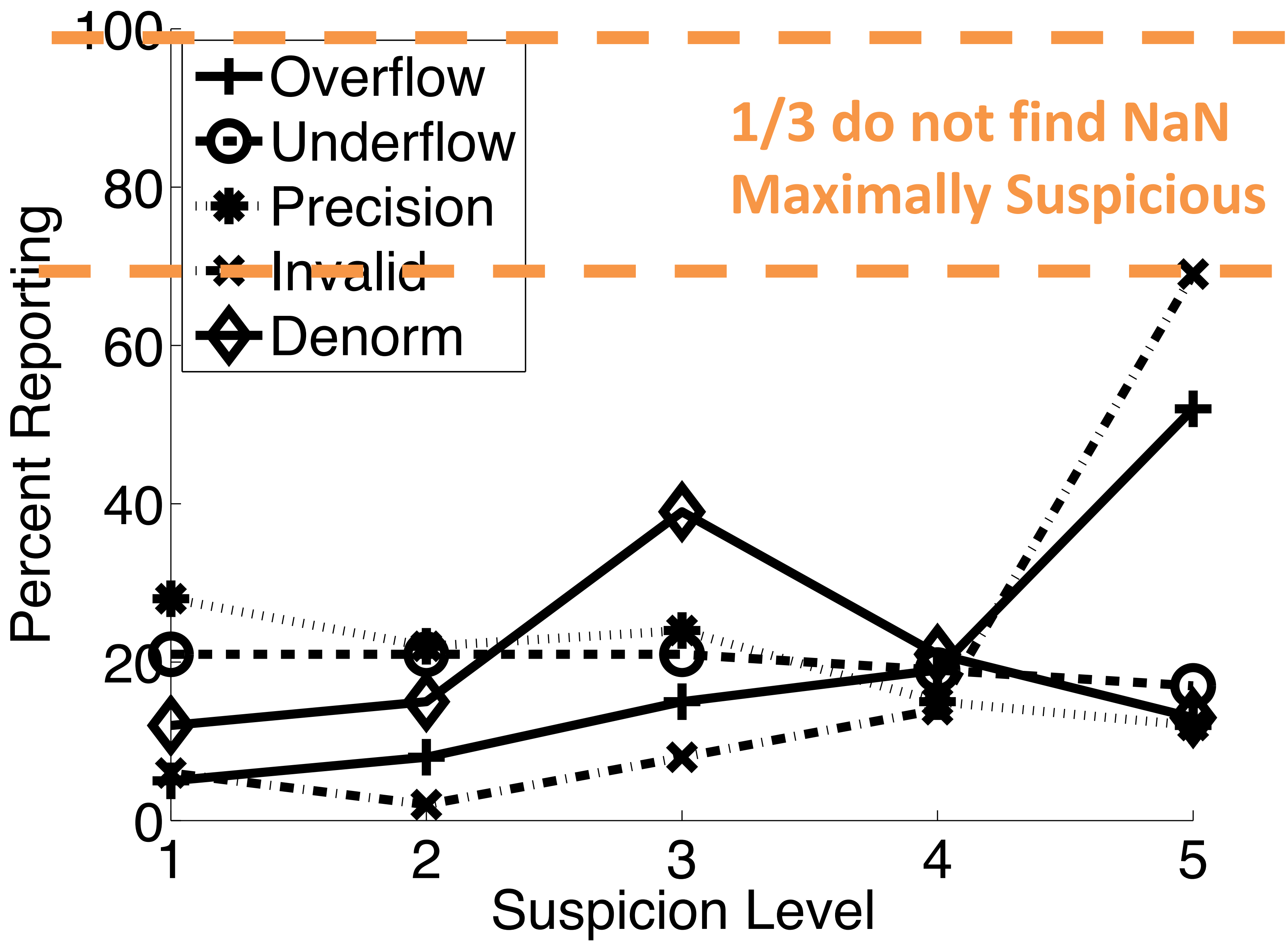
Can you tell these graphs apart?

One is undergrads in an introductory systems course



Can you tell these graphs apart?

One is undergrads in an introductory systems course



1/3 do not find NaN
Maximally Suspicious
Overflow
Underflow
Precision
Invalid
Denorm
Percent Reporting
Suspicion Level
0
20
40
60
80
100
1
2
3
4
5

Caveats

- Participants are not a random sample
- Anonymity and self-reporting
  - We cannot be sure we have hit our recruitment goals
- Confusion/lack of time for participant
  - Survey design was iterated based on feedback
- Only 199+52 data points
  - But these are users
- …

Potential Actions

- HPC community should sow suspicion
  - Much like PL and compilers community did with undefined behavior in C
- HPC community should develop better training

Potential Actions

- Better static/dynamic analysis tools
  - Work in progress
- Blurring the boundary between FP and arbitrary precision arithmetic
  - Work in progress
- Developer knowledge-limited access to software and hardware optimizations
  - “Achievement Unlocked”
  - Work in progress

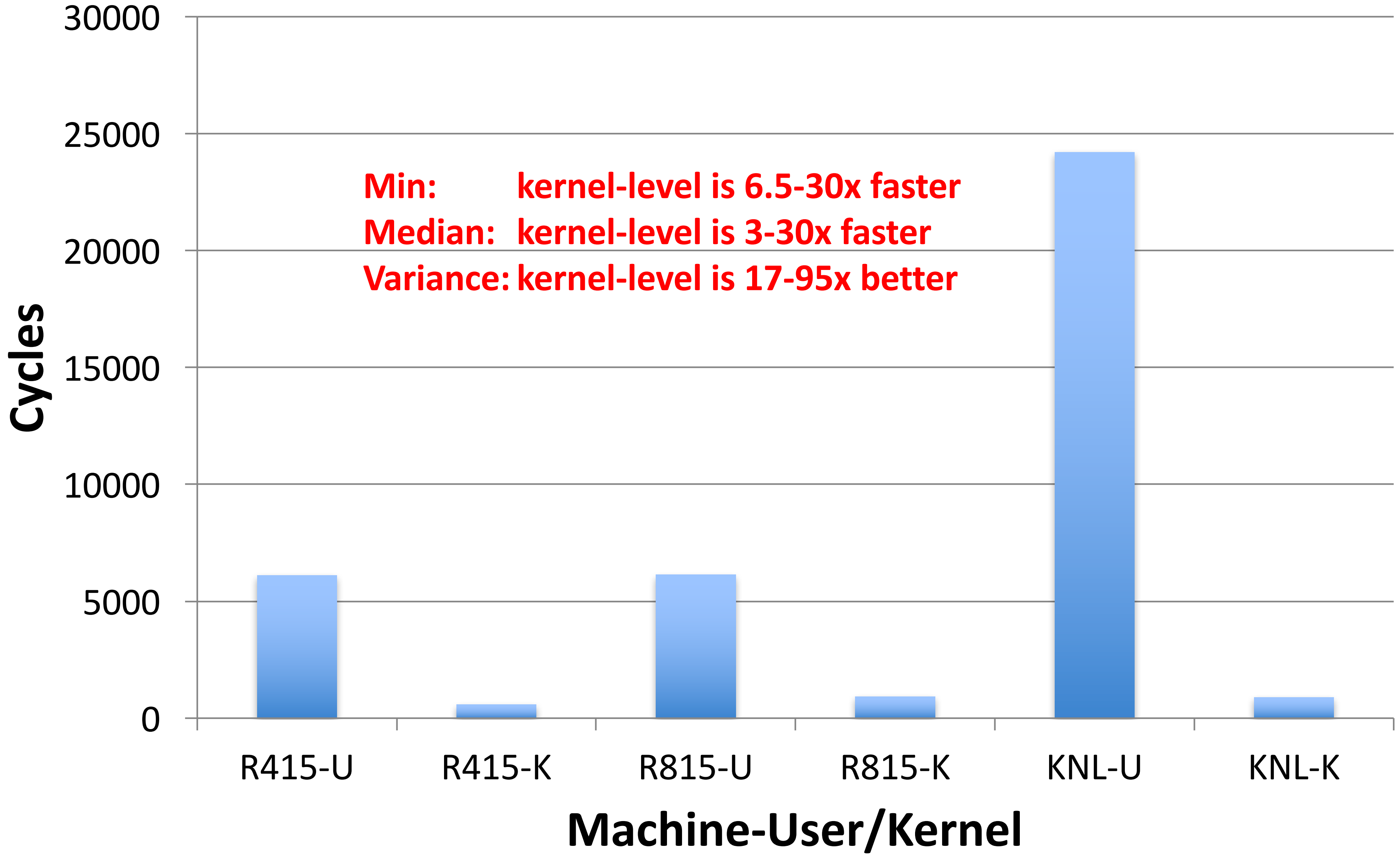
A Work in Progress: FPSpy

- User-level shim that slides underneath existing, unmodified application binary
  - Gets out of the way on conflict with application
- Uses FP hardware features, Linux FP interfaces, and debugger-style techniques to track issues
  - Aggregate mode:
    - FP condition codes set at any point during execution
    - Fast – zero overhead
  - Individual mode:
    - Instruction-level tracking of FP condition codes
    - Slower
- Current: applying FPSpy and other tools to study existing, unmodified applications
  - Does developer confusion as measured in present study manifest in codes in current use?

A Work In Progress: FPKernel

- Floating point exceptions have much lower latency and overhead in a kernel-only model
  - Like our Hybrid Run-Time (HRT) scheme and the Nautilus Kernel Framework that supports it
- Combine fixed precision hardware FP and arbitrary precision software FP to create simple arithmetic model for programmer
  - FP exceptions trigger transition to software FP
  - NaN boxing / signaling NaN for values
  - Made more practical by fast FP exceptions

Minimum Time to Floating Point Exception Handler
Linux User-level Versus Nautilus Kernel-level



Paper in a Nutshell: **Not Really**

- Targeted survey
  - Aimed at practitioners likely to use FP
  - Quizzes for core, optimization, and suspicion of results
  - **First study of this kind**

- Participants do only slightly better than chance on core concepts
  - … and don't know it
  - Some factors mitigate, but none particularly well

- Participants do not understand optimization concepts
  - … and do know it

- Participants less suspicious than they should be
  - … but similar to students in a sophomore course

- Maybe systems software can do something about it

For More Information

- Peter Dinda
  - pdinda@northwestern.edu
  - http://pdinda.org
- Conor Hetland
  - ConorHetland2015@u.northwestern.edu
- Take the survey
  - http://presciencelab.org/float
- Prescience Lab
  - http://presciencelab.org

- Acknowledgements
  - NSF, DOE