Testing Noninterference, Quickly

Cătălin Hrițcu, John Hughes, Benjamin C. Pierce,
Antal Spector-Zabusky, Dimitrios Vytiniotis,
Arthur Azevedo de Amorim, Leonidas Lampropoulos

to appear at ICFP 2013

The SAFE project

@ Penn, Harvard, Northeastern, BAE Systems

language

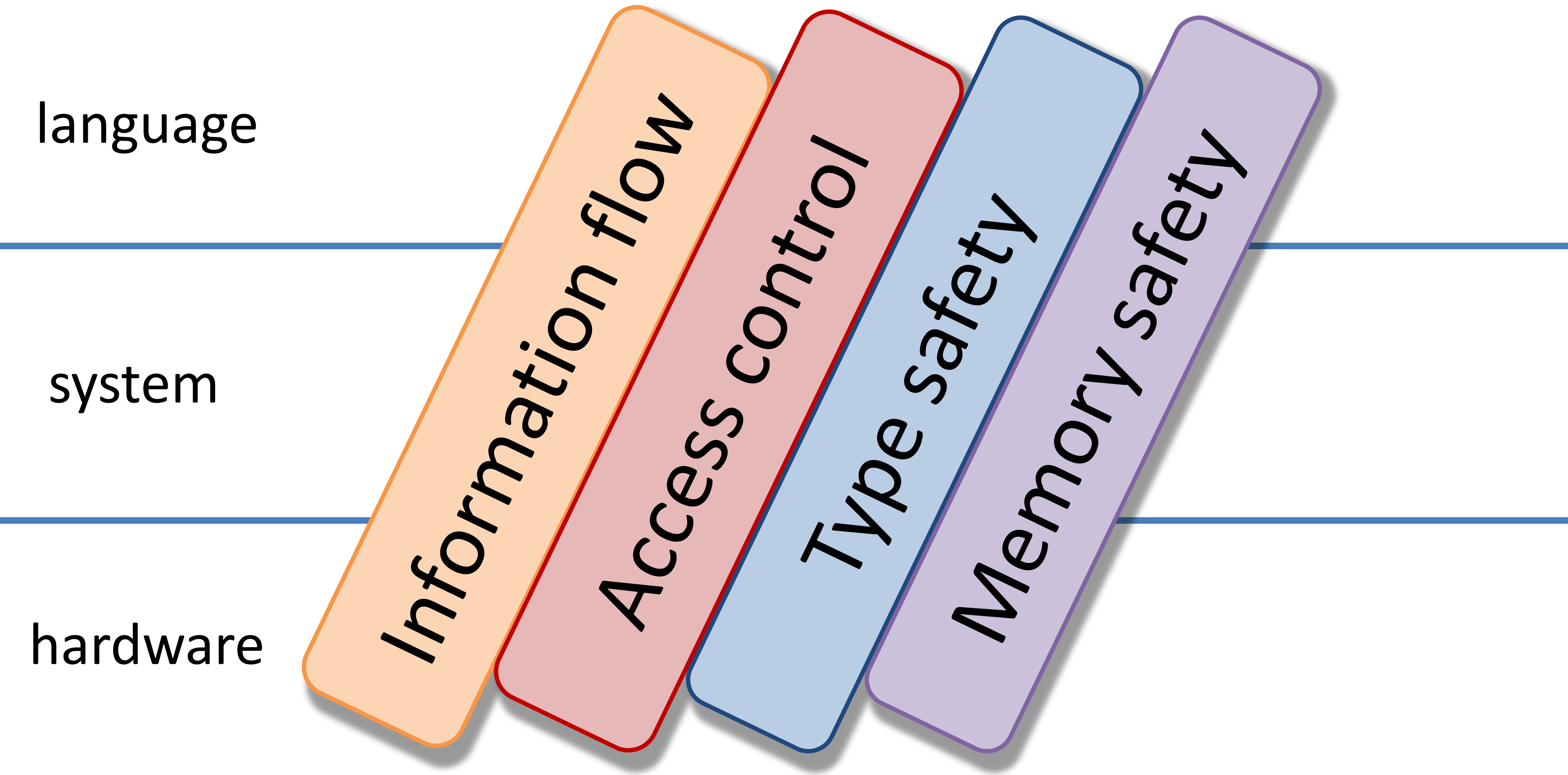
system

hardware

http://www.crash-safe.org/

The SAFE project

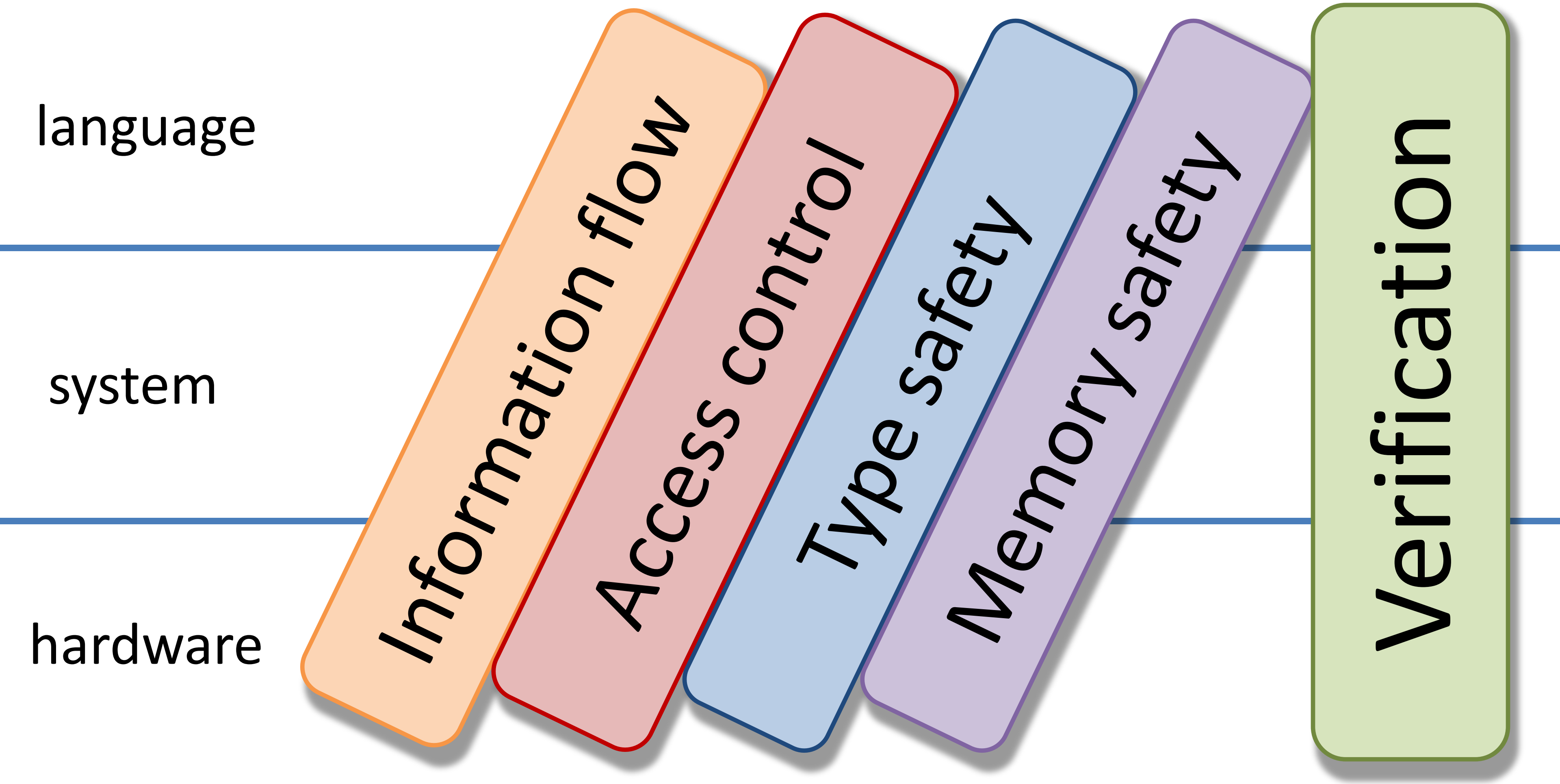
@ Penn, Harvard, Northeastern, BAE Systems



http://www.crash-safe.org/

The SAFE project

@ Penn, Harvard, Northeastern, BAE Systems



http://www.crash-safe.org/

Verifying security of the SAFE system

long term goal

Verifying security of the SAFE system

long term goal

- current status:
  noninterference in Coq for a very simplified model

Verifying security of the SAFE system

long term goal

- current status:
  noninterference in Coq for a very simplified model

Benjamin's keynote on Friday

Verifying security of the SAFE system

long term goal

- current status:
  noninterference in Coq for a very simplified model

Benjamin's keynote on Friday

- **However**…
  - Proofs for actual system will require **a lot more work**
  - Design is **still evolving**
  - **Feedback** on correctness needed ASAP

Random testing?

- Can we use property-based **random testing** for checking **noninterference**?

Random testing?

- Can we use property-based **random testing** for checking **noninterference**?
- **The experiment**
  - ***very simple* machine** (10 instructions)
  - **standard noninterference property**
  - **use QuickCheck** to generate many random programs and try to **find counterexamples**

Encouraging results

- introduced **plausible errors** in IFC rules
- **all errors found in 2-16ms** on average

Encouraging results

- introduced **plausible errors** in IFC rules
- **all errors found in 2-16ms** on average

- **However**, for these results
  we are not using QuickCheck naïvely
  - that didn't really work for us
  - **significant cleverness was needed** in 3 areas…

The 3 secret ingredients

1. Clever **program generation strategies**
   - generating only data that satisfies preconditions
   - “generation by execution”
2. **Strengthening the tested property**
   - best one: unwinding conditions
   - requires inventing (by hand!) stronger invariants
     - invariants of real SAFE machine are very complicated
3. **Shrinking** counterexamples

Getting confidence by testing

- *"testing can only show the presence of bugs, not their absence"* – Dijkstra

Getting confidence by testing

- *"testing can only show the presence of bugs, not their absence"* – Dijkstra
- **new idea**: **use old bugs to "test" the generator**
  - **if** all old bugs found fast & no new bugs found
  - **then** we do get some confidence

Getting confidence by testing

- *"testing can only show the presence of bugs, not their absence"* – Dijkstra
- **new idea**: **use old bugs to "test" the generator**
  - **if** all old bugs found fast & no new bugs found
  - **then** we do get some confidence
- **open problems**
  - how to save bugs without turning code into spaghetti?
  - or how to add all interesting bugs automatically?

Conclusion

- **property-based random testing**
  - **is a lot of fun**
  - can **inform and speed up design process**
  - can serve as **1st step towards formal verification**
    - concentrate more energy on proving correct things
    - finding the right design, properties, and invariants
  - **is not push-button ... yet**
    - but **some general tricks can help a lot**