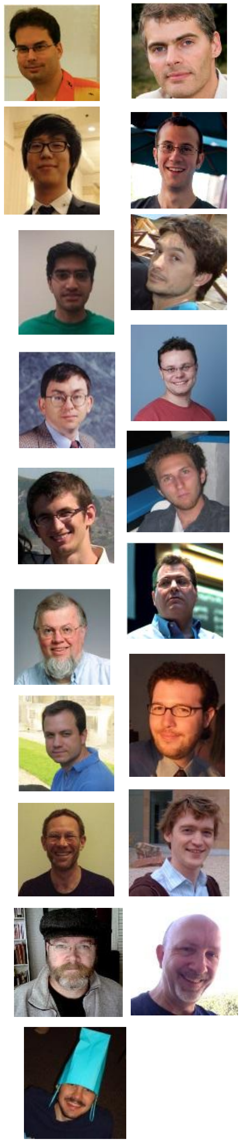
CRASH/SAFE: Clean-slate Co-design of a Secure Host Architecture

Cătălin Hrițcu

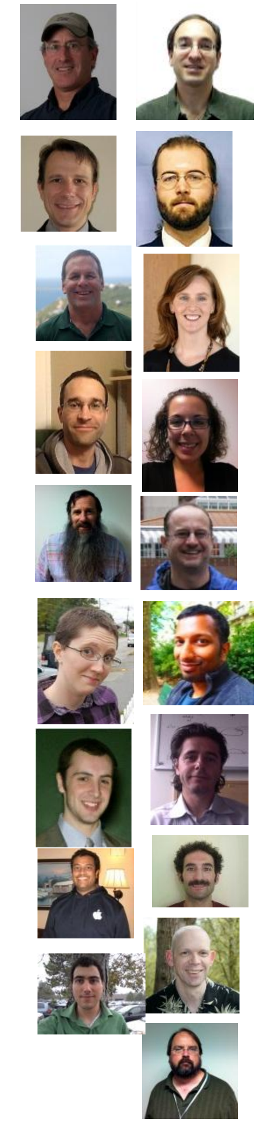
Penn
University of Pennsylvania

CRASH/SAFE project



- Funded by DARPA
  - **C**lean-Slate Design of **R**esilient, **A**daptive, **S**ecure **H**osts
- Academic partners (16):
  - **University of Pennsylvania** (11)
  - **Harvard University** (4)
  - **Northeastern University** (1)
- Industrial partners (24):
  - **BAE systems** (21) + **Clozure** (3)

40!



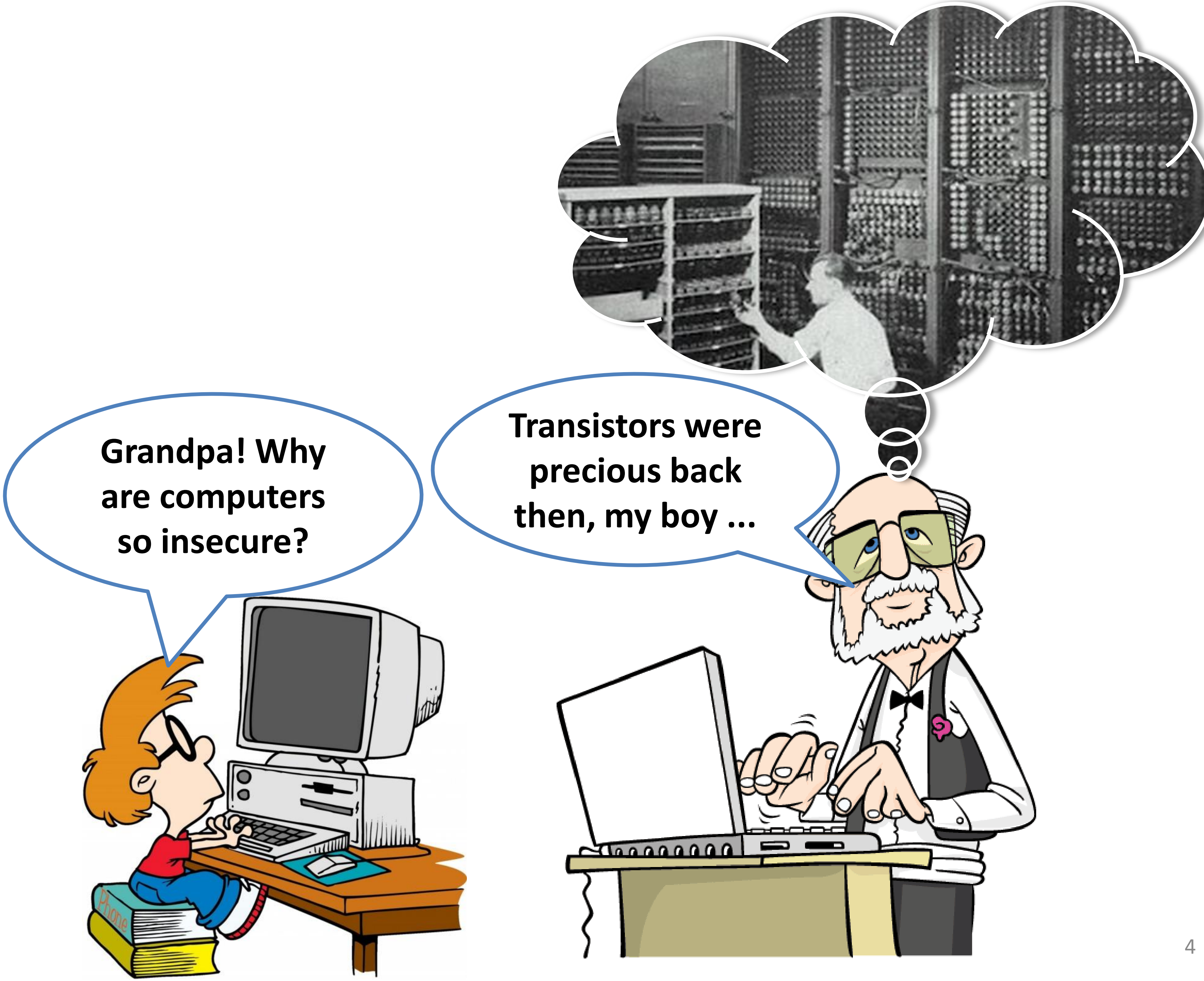
Clean-slate co-design of net host

Primary goal:
design and implement a significantly more secure architecture, without backwards compatibility concerns

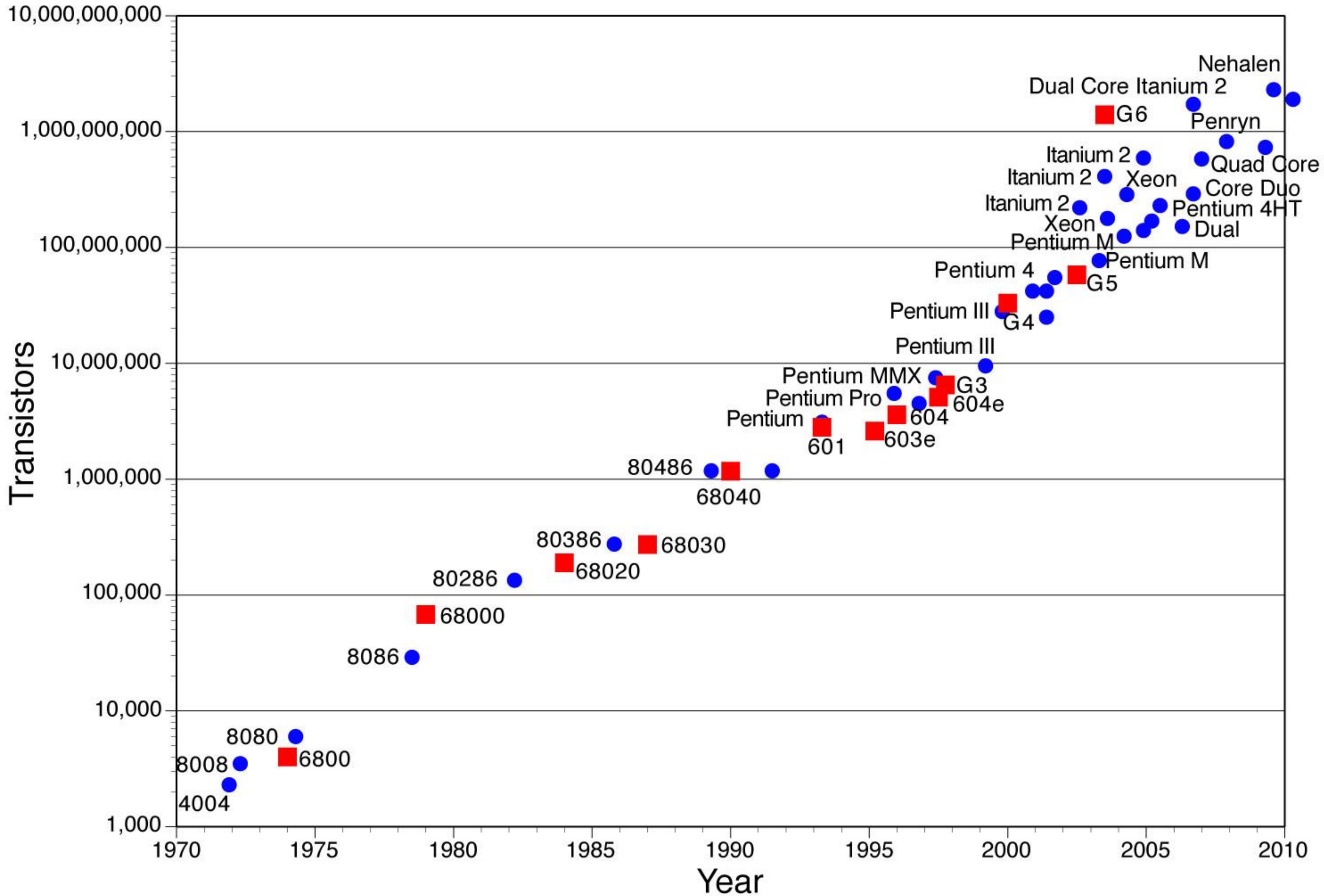
Secondary goal:
verify that it's secure (whatever that means)

New stack:

- language
- runtime
- hardware


Grandpa! Why are computers so insecure?
Transistors were precious back then, my boy ...
Phone

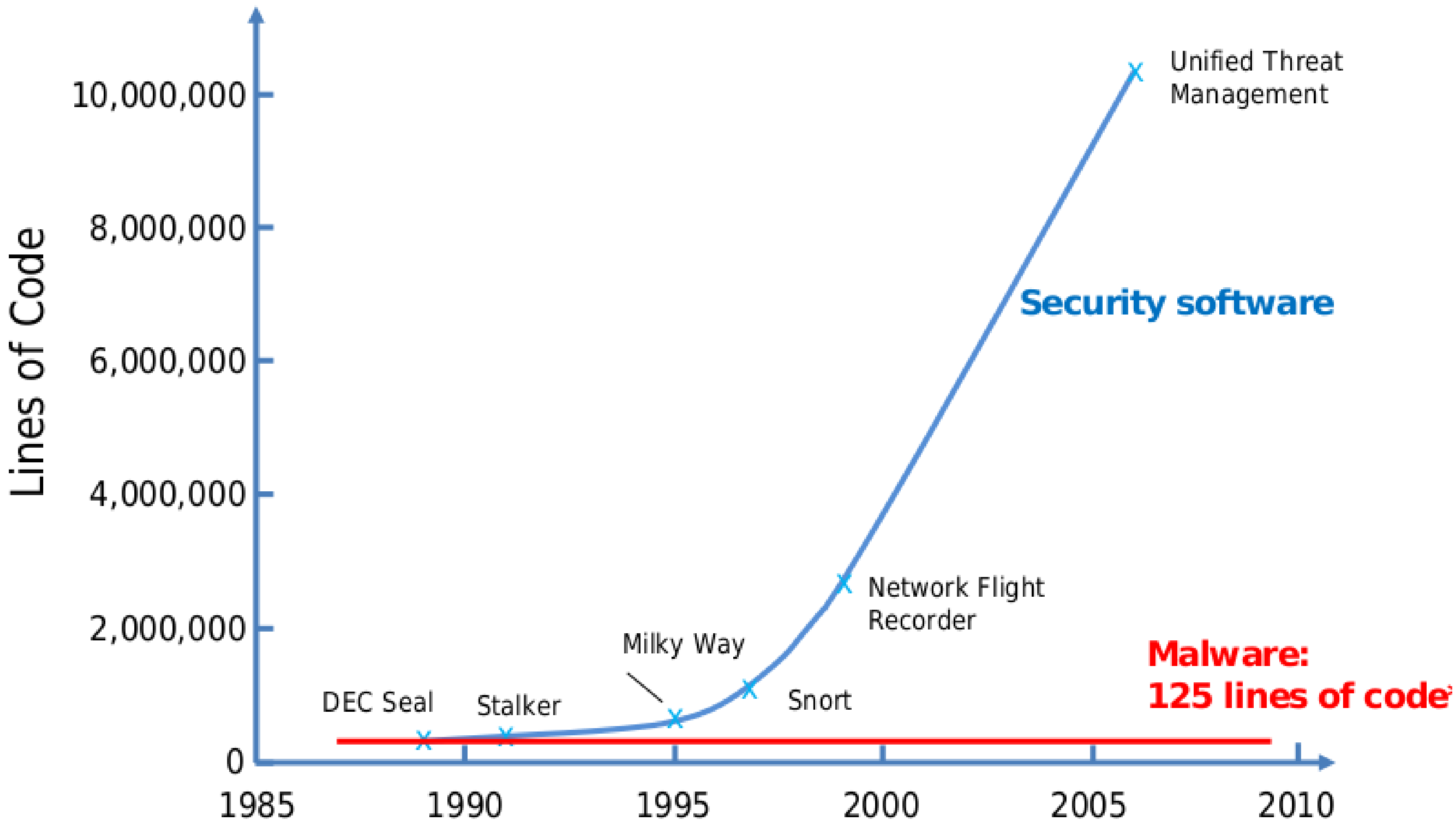
Hardware is now abundant



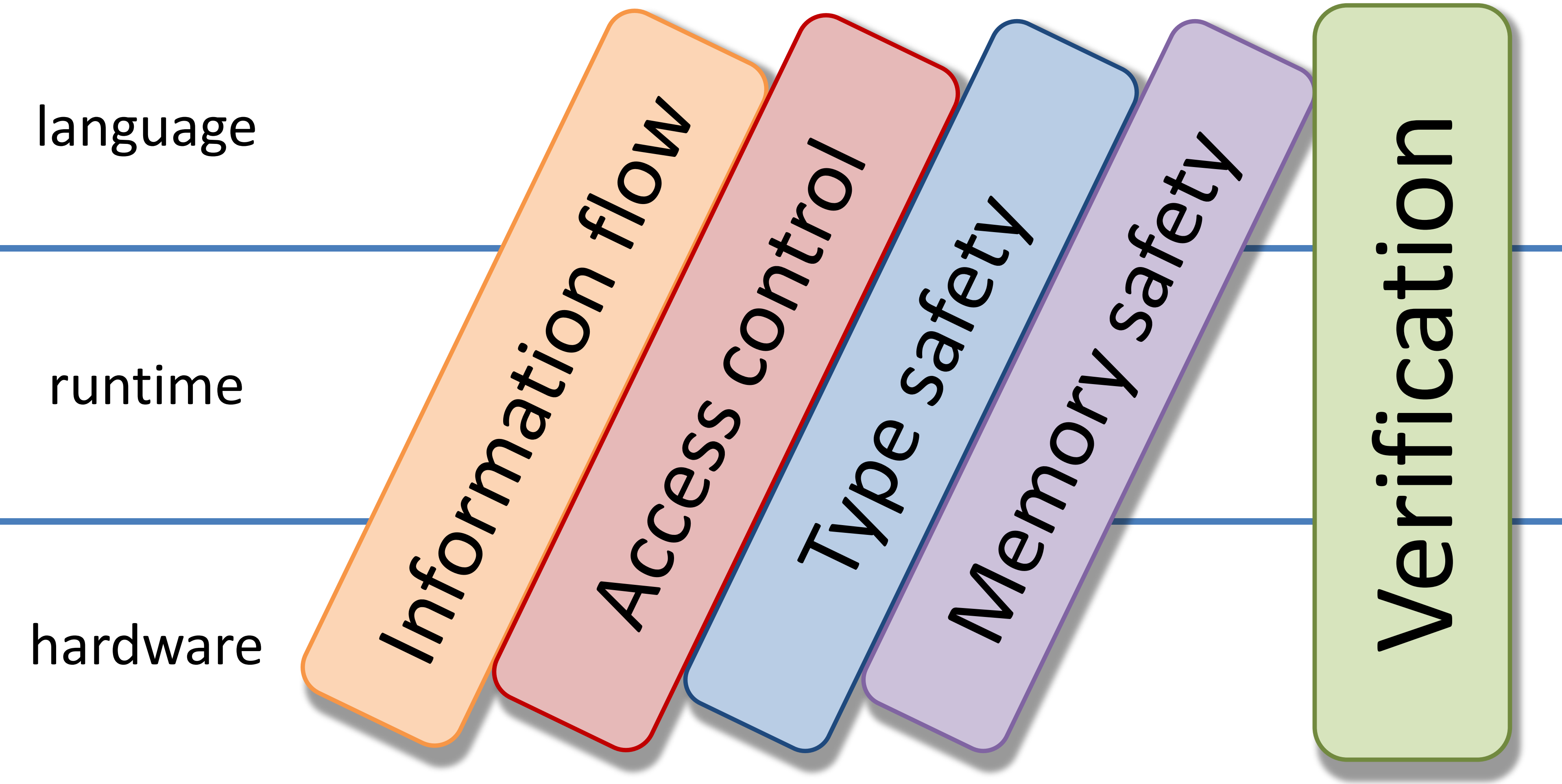
Formal methods are better now

- **random testing**
  - QuickCheck [Claessen & Hughes, ICFP'00]
- **automatic theorem provers & SMT solvers**
- **machine-checked proofs**
  - CompCert [Leroy, POPL'06]
  - seL4 [Klein et al, SOSP'09]
  - CertiCrypt [Barthe et al., POPL'09]
  - ZKCrypt [Almeida et al, CCS'12]

Security is much more important



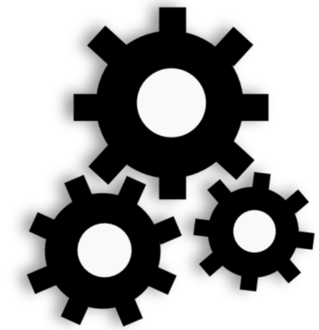
Time for a redesign!



Language (Breeze)

- testing ground for ideas we port to lower levels
- **type and memory safe** high-level language
  - **dynamically typed +** dynamically-checked contracts
- **functional core** (λ) + state(!) + concurrency (π)
  - message-passing communication (channels)
- built-in **fine-grained protection mechanisms**:
  - values are attached **security labels**
  - **dynamic information flow control** (IFC)
  - **discretionary access control** (clearance)

Runtime system

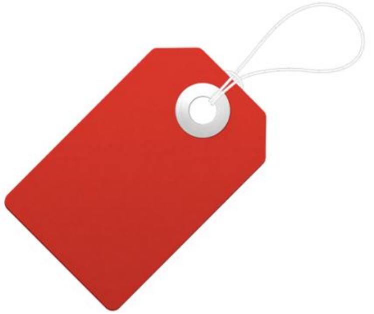


- manages:
  - **time** (scheduler)
  - **memory** (allocator, garbage collector)
  - **communication and resources** (channels)
  - **protection** (principals, authorities, and tags)
- small trusted computing base
- comparimentalized
  - a dozen mutually distrustful servers (least privilege)

Hardware

- all instructions have well-defined semantics
  - abstractions strictly enforced
- low-**fat pointers**
  - can't access/write out of frame bounds
- **dynamic types**
  - can't turn ints into pointers (unforgeable **capabilities**)
- **authority** + **closures/gates** (λ) + **protected stack**
  - fine-grained privilege separation
- programmable **tag management unit** (TMU)

Tag management



- **every word tagged** with arbitrary pointer
  – only runtime system interprets these pointers
- on **each instruction** TMU looks up tags of operands in a **hardware rule cache**
  – found → rule provides tags on results (no delay)
  – not found → trap to software (PAT server)
- **access control + IFC** enforced at lowest level

Status

- **language**:
  - stable interpreter, work-in-progress compiler
  - Coq proofs for various core calculi (non-interference)
- **runtime**:
  - detailed design, some prototype servers
  - work on testing+verifying simplified PAT server
- **hardware**:
  - working un-pipelined FPGA prototype
  - novel instruction set, simulators, debugger, ...
  - executable instruction set semantics in Coq

MY RESEARCH

All Your IFCException Are Belong To Us

Robust Exception Handling for Sound Fine-Grained Dynamic IFC

joint work with Michael Greenberg, Ben Karel, Benjamin Pierce, and Greg Morrisett

Sound dynamic IFC possible

- Non-interference can be obtained purely dynamically!
  - [Krohn & Tromer, 2009], [Sabelfeld & Russo, 2009], [Austin & Flanagan, 2009]
- Preventing implicit flows:

```
let lref = ref low false in
if h then                    pc=high
  lref := true;              potential bad flow  -> halt program
lref := false                false alarm (program non-interferent)
...
```

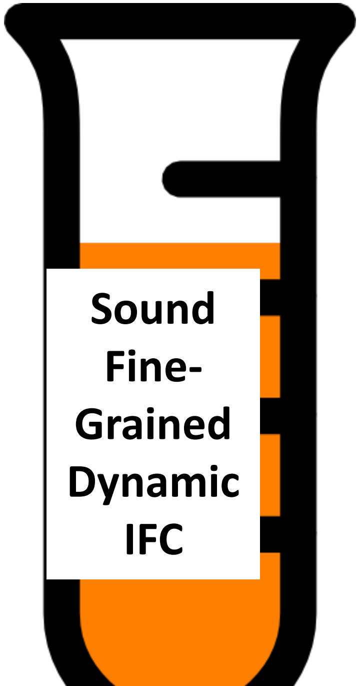
- Even functional code can leak via control flow:
  - `if h then true else false`
  - semantics of conditional:
    - **if** true@high **then** true **else** false => true@high

Exception handling

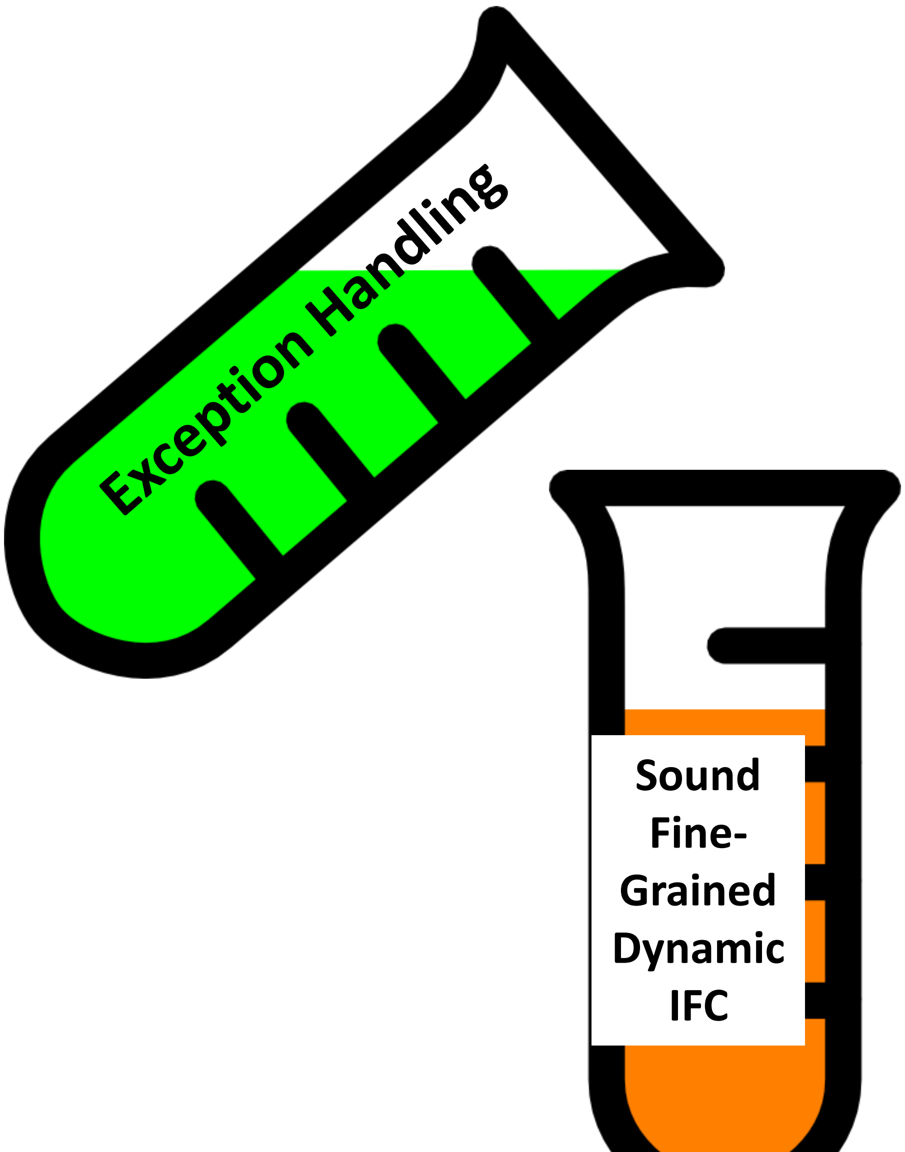
- we wanted all Breeze errors to be **recoverable**
  - **including IFC violations**
- however, existing work assumes errors are **fatal**
  - makes some things easier ... at the expense of others

    +secrecy **+integrity** **–availability**

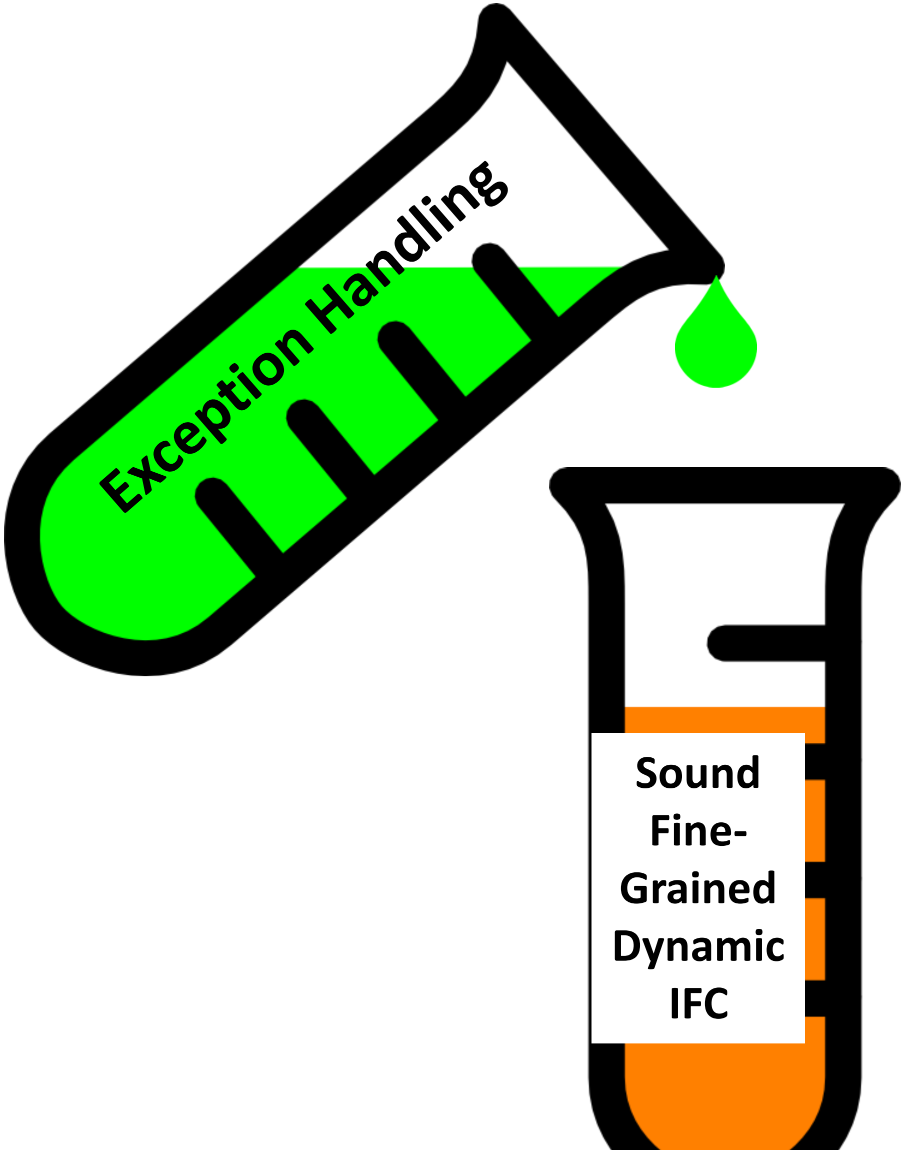
But there is a problem



But there is a problem



But there is a problem



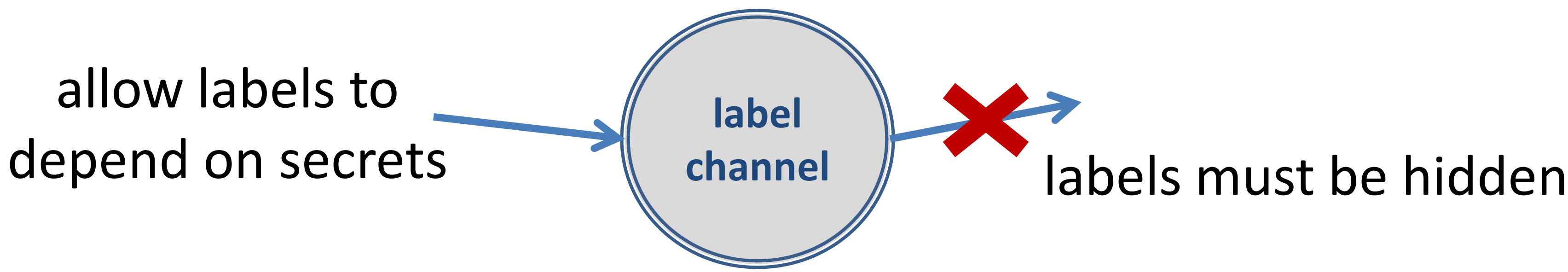
But there is a problem … **in fact two!**

Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either ***into*** or ***out of*** label channel

Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either ***into*** or ***out of*** label channel




```
if h then ()@high else ()@top
```

Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either ***into*** or ***out of*** label channel

allow labels to depend on secrets

labels must be hidden

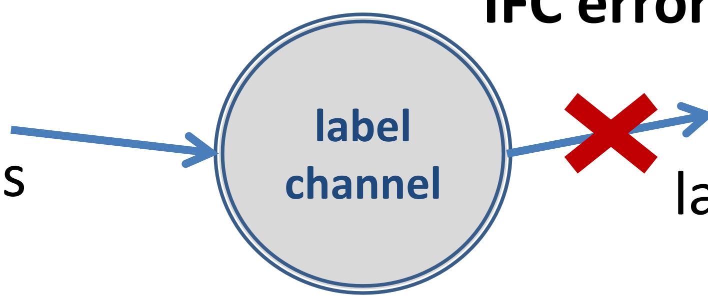
```
pc=low   if h then ()@high else ()@top => ()@{high/top}   pc=low
                   pc=high        pc=high
```

Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either ***into*** or ***out of*** label channel

IFC errors must be hidden too!

allow labels to depend on secrets

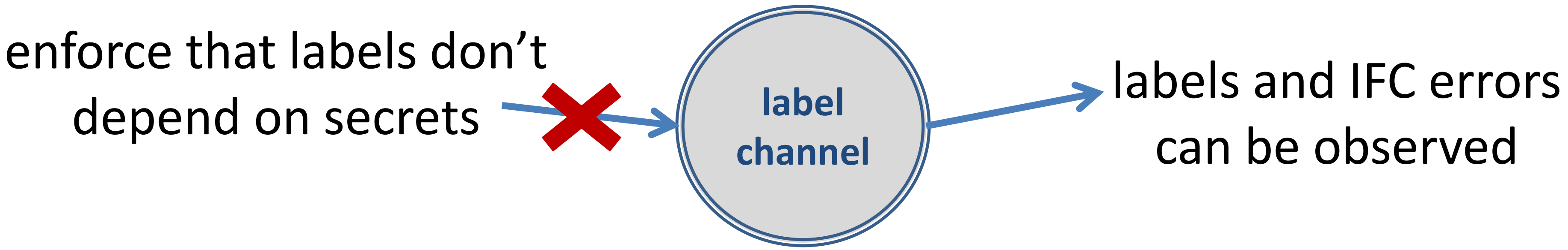


labels must be hidden

```
pc=low  if h then ()@high else ()@top => ()@{high/top}  pc=low
                  pc=high         pc=high
```

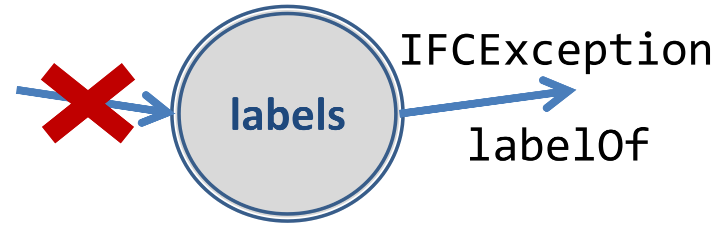
Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either ***into*** ~~or ***out of***~~ label channel



Solution #1: brackets

- **prevent labels from depending on secrets so that labels are public**



- do not automatically restore pc
  - `pc=low  if h then ()@high else ()@top => ()@{high/top}  pc=high`
- instead, restore pc manually using **brackets**
  - choose label on result before branching on secrets
  - `pc=low  top[if h then ()@high else ()@top] => ()@top  pc=low`
  - brackets are not declassification!
  - sound even when annotation is incorrect (next slide)
  - bracket annotations can be dynamically computed (labelOf)

Problem #2: exceptions destroy control flow join points

- ending brackets have to be control flow join points
  - ```
    try
      let _ = high[if h then throw Ex] in
      false
    catch Ex => true
    ```
- brackets need to delay all exceptions!
  - high[**if** true@high **then** throw Ex] => “(Inr Ex)@high”
  - high[**if** false@high **then** throw Ex] => “(Inl ())@high”
- similarly for failed brackets
  - high[42@top] => “(Inr EBracket)@high”
    ```

Solution #2: Delayed exceptions

- **delayed exceptions unavoidable**
  - still have a choice how to propagate them
- we studied **two alternatives** for error handling:
  1. **mix active and delayed exceptions** ($\lambda^{[\,]}_{throw}$)
  2. **only delayed exceptions** ($\lambda^{[\,]}_{NaV}$)
     - delayed exception = not-a-value (NaV)
     - NaVs are first-class replacement for values
     - NaVs propagated solely via data flow
     - NaVs are labeled and pervasive
     - more radical solution; implemented by Breeze

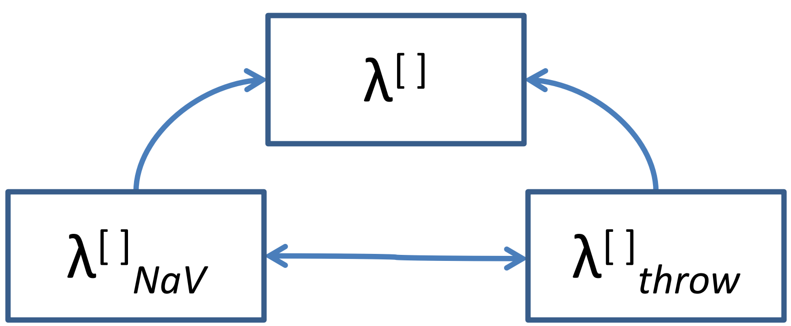
What’s in a NaV?

- error message
  - `EDivisionByZero (“can’t divide %1 by 0”, 42)
- stack trace
  - pinpoints error **origin**
    (not the billion-dollar mistake)
- propagation trace
  - how did the error make it here?

NaVs are compiler writer’s dream, especially if compiler is allowed to be imprecise about these debugging aids
(Greg Morrisett)

Formal results

- proved termination-insensitive **non-interference** in Coq for $\lambda^{[\,]}$, $\lambda^{[\,]}_{NaV}$, and $\lambda^{[\,]}_{throw}$
  - for $\lambda^{[\,]}_{NaV}$ even with all debugging aids; **error-sensitive**
- in our setting NaVs and catchable exceptions have **equivalent expressive power**
  - translations validated by QuickChecking extracted code



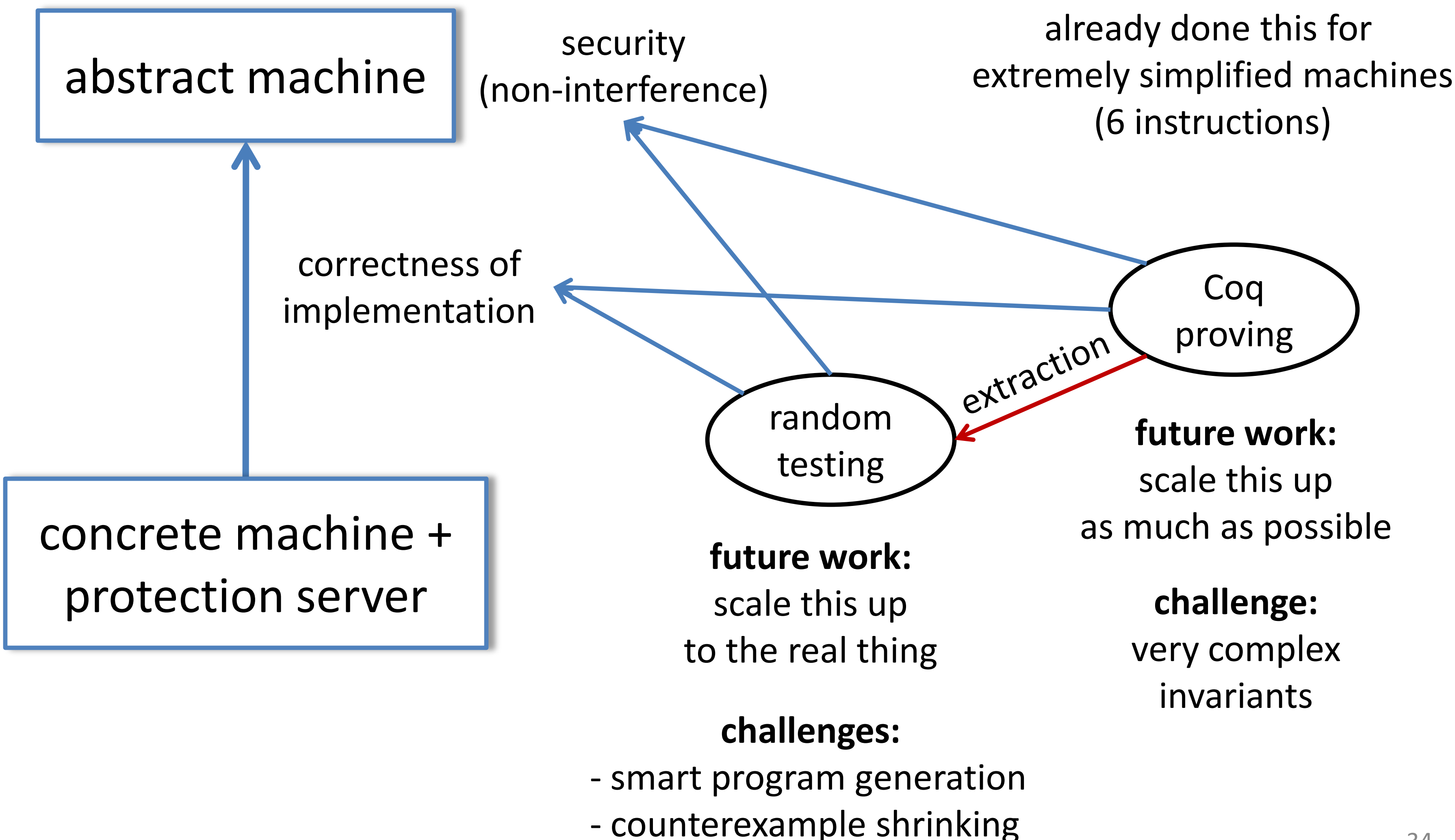
Summary for IFC exceptions

- reliable error handling *possible* even for sound fine-grained dynamic IFC systems
- we study two mechanisms ($\lambda^{[\,]}_{NaV}$ and $\lambda^{[\,]}_{throw}$)
  - **all errors recoverable**, even IFC violations
  - key ingredients:
    sound public labels (brackets) + **delayed exceptions**
  - quite radical design (not backwards compatible!)
- gathering practical experience with NaVs:
  - issues are surmountable
  - writing good error recovery code is still hard

Ongoing work

- **testing and verifying the PAT server**
- protecting data integrity with signature labels
- implementing Breeze labels cryptography

Testing and verifying PAT server



Two projects for the future

- **Software-hardware co-design for security-critical high-assurance devices**
  - electronic voting, driver assistance, medical devices
    - limited/fixed functionality
    - security more important than backwards compatibility
  - existing devices often blatantly vulnerable
  - making security analysis part of design process
  - focus on research (compared to CRASH/SAFE)
- Fine-grained access control and integrity protection for mobile devices

THE END