My New/Old Agenda

Dave MacQueen
WG 2.8, Aussois, 2013

From Ryerson Hall

To Silicon Valley

My To Do List

1 Maintain and improve SML/NJ

2 Writing projects

3 Reviving Standard ML Definition

Thoughts on Simplicity

Maintain and Improve SML/NJ

- type checking improvements
  - mostly improved error messages
- module system internals rewrite
- FLINT middle end cleanup or (better) replacement
- modernizations
  - 64 bit
  - unicode
  - LLVM (replacing MLRISC)
- regular bi-annual releases planned

Writing Projects

- Induction tutorial
  - from first principles
  - full and precise induction proofs
  - seeking further good examples (small, challenging induction proofs)

- PL course notes packaging for the web
  - open source supplementary textbook
  - full and precise proofs as in the Induction Tutorial
  - simplified variant of Harper's book

- Historical survey of module design and semantics
  - with prototype implementations of several key module papers

- Modules vs Type Classes (supervised Master's thesis)
  - critical comparison: Haskell classes, SML modules, modular type classes

Reviving the Definition of Standard ML, Revised (SML '97)

Negotiated permission from MIT Press to open the Definition for free online distribution.

1. recreate SML '97 sources and make them available online as pdf version and github source repository.

   SML '90 available now at: https://github.com/SMLDefinition

2. make corrections and minor revisions that have been suggested over the years (e.g., clarify rules for overloading resolution).

Reviving the Definition of Standard ML, Revised (SML '97)

3. Definition and language evolution, e.g.:
    - cost semantics
    - higher-order functors
    - module definitions within the core language
    - modular type classes
    - parallel ML (Manticore)

    See Bob Harper's "Future of SML" talk at the 2013 ML Workshop:
    http://www.cs.cmu/~rwh/talks/mlw13.pdf

Serious revisions will require new formalisms.
- e.g. higher-order functors incompatible with current Defn and probably with Harper-Stone style "type-theoretic" semantics
- revised definitions should ideally be mechanized

What to call a revised language? Don't want to use "Standard" anymore.
"Successor ML"? "ML2K"? "ML"?

Thoughts on Simplicity in Language Design

- Language should be simple enough that competent programmers can understand and use the whole language (without subsetting)
  ==> simple and accurate programmer's working model.

- Simple enough to have a workable, pragmatic cost model.

- [types] Simple enough to have "obvious" ("transparent") type checking
  -- should be possible to do type checking **in your head** as you write/read programs.

- Simple enough to be able to teach/learn the language to undergrad CS students (working programmers) in a reasonable amount of time
  (e.g. << a 10 week quarter, say 3 weeks or 10 hours of lectures + homework)

- Simple enough to "reason about" programs (informally)

- Simple enough to have a reasonably accessible formal definition

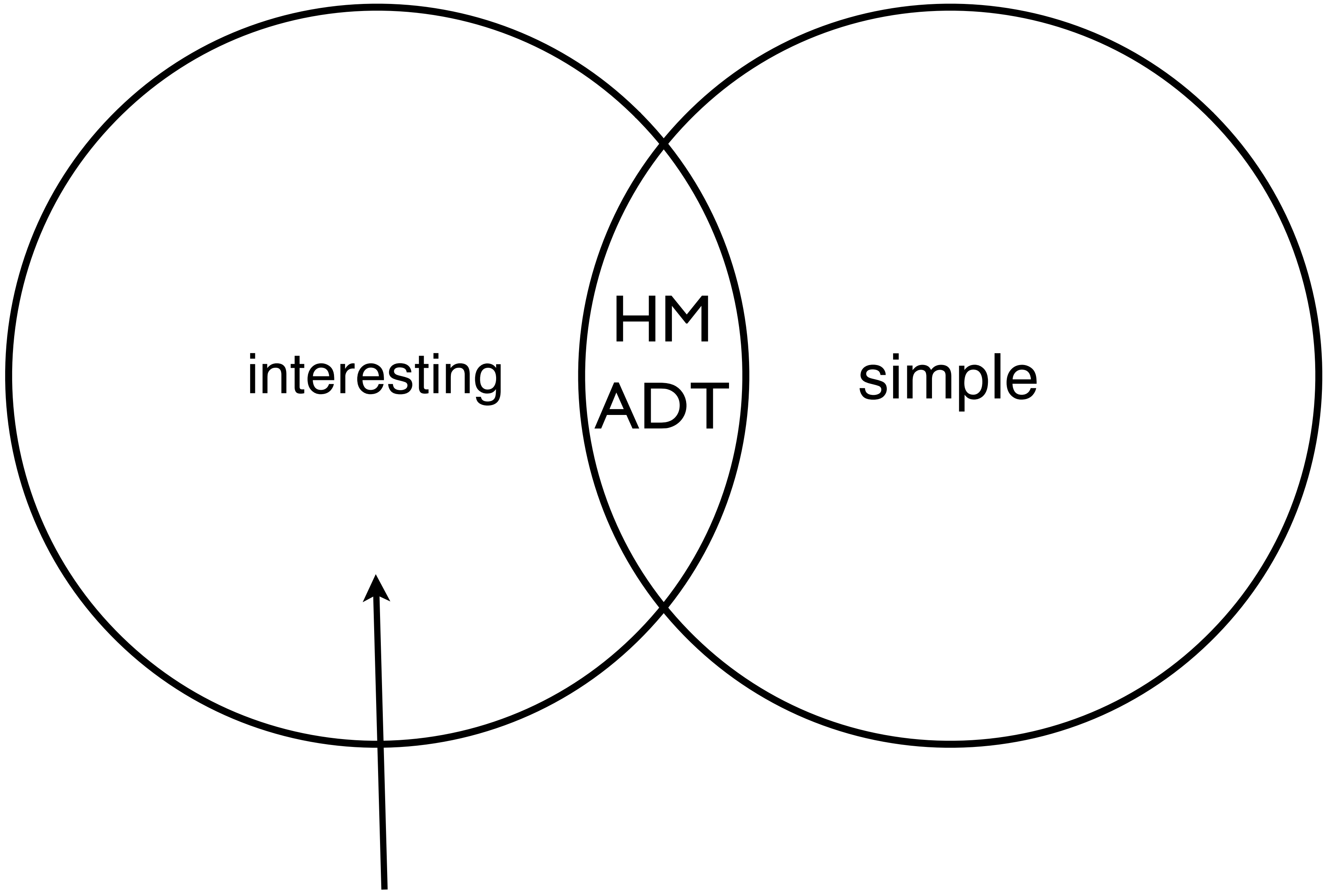
Design evolution tends to greater complexity

- Very hard to eliminate features used in any existing code (no matter how little code exists, even potential code!)

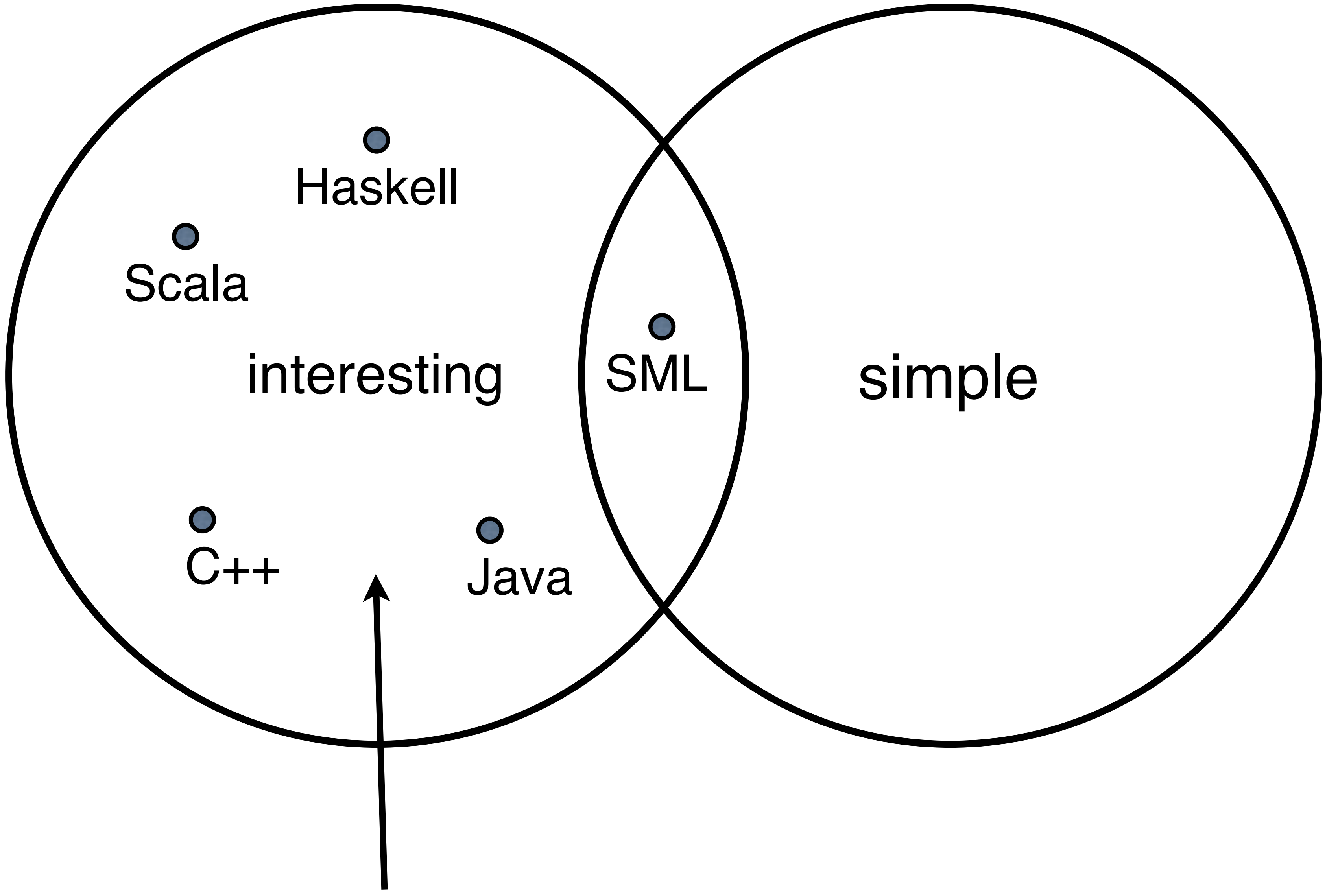
- Hence, once a feature is added, it will almost never be removed, or even improved/simplified. Instead, even more features may be added to try to correct deficiencies in earlier features, leading to

    - redundancy
        ** e.g. Lisp Machine lisp. Many ways to do the same thing, layered historically (like a software archeological tell)

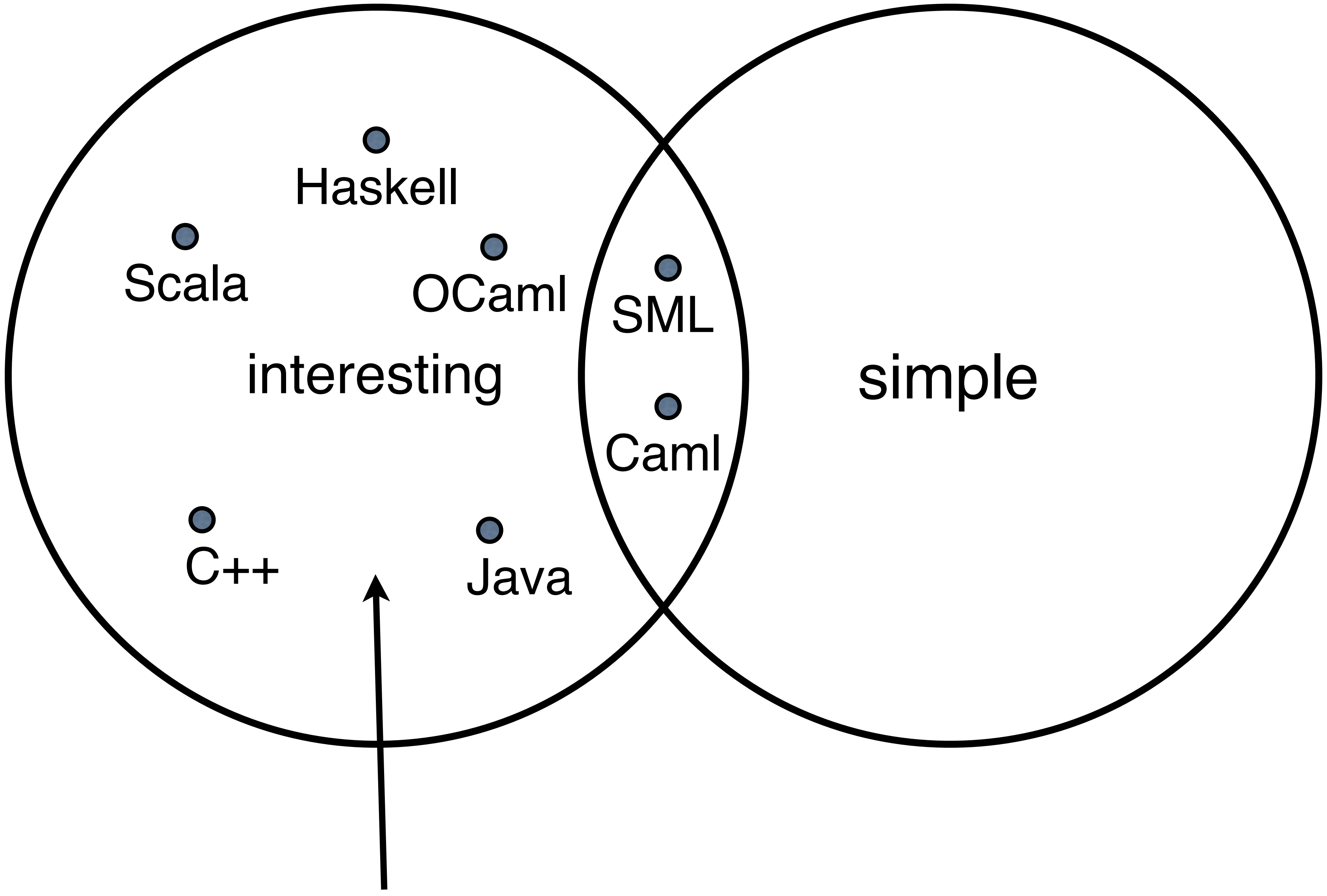
    - unintended and undesirable feature interactions



the domain of the mathematician programmer



the domain of the mathematician programmer



the domain of the mathematician programmer