The Age of Deep Specification

Benjamin C. Pierce
University of Pennsylvania

May, 2015

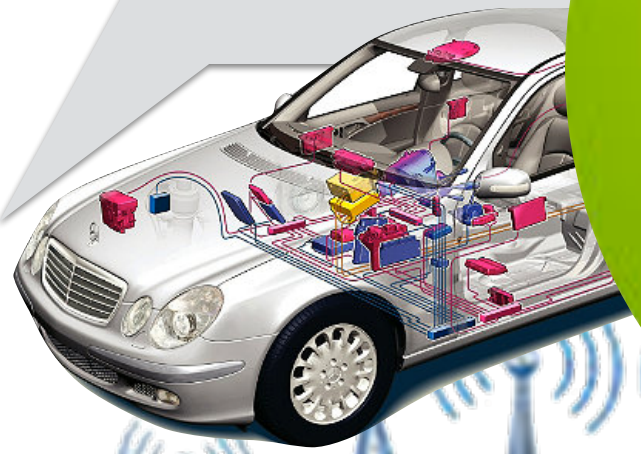
“We can’t build software that works…”

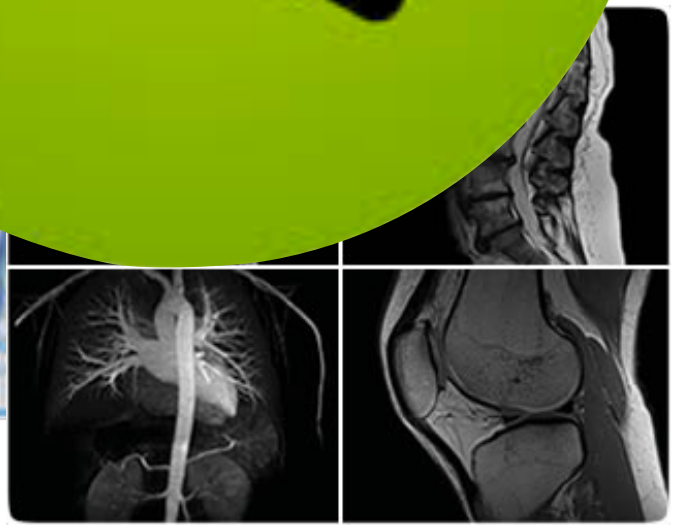
PWNED!

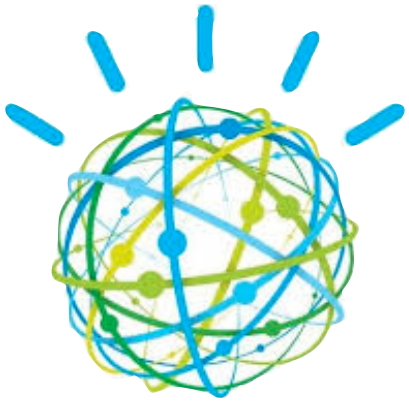
"We can't build software that works…"

But just look at all the software that *does* work!

But ju ll the
softw work!







How did that happen?

Lots of ways!

Lots of ways!

- Better software development methodology

Lots of ways!

- Better software development methodology
- Better programming languages
  - Basic *safety guarantees* built in
  - Powerful mechanisms for *abstraction* and *modularity*

Lots of ways!

- Better software development methodology
- Better programming languages
  - Basic *safety guarantees* built in
  - Powerful mechanisms for *abstraction* and *modularity*
- Better testing

Lots of ways!

- Better software development methodology
- Better programming languages
  - Basic *safety guarantees* built in
  - Powerful mechanisms for *abstraction* and *modularity*
- Better testing
- Better use of specifications

Lots of ways!

- Better software development methodology
- Better programming languages
  - Basic *safety guarantees* built in
  - Powerful mechanisms for *abstraction* and *modularity*
- Better testing
- Better use of specifications

I.e., descriptions of what software does (as opposed to the instructions for how to do it)

Lots of ways!

- Better software development methodology
- Better programming languages
  - Basic *safety guarantees* built in
  - Powerful mechanisms for *abstraction* and *modularity*
- Better testing
- Better use of specifications

I.e., descriptions of what software does (as opposed to the instructions for how to do it)

Why are specifications useful?

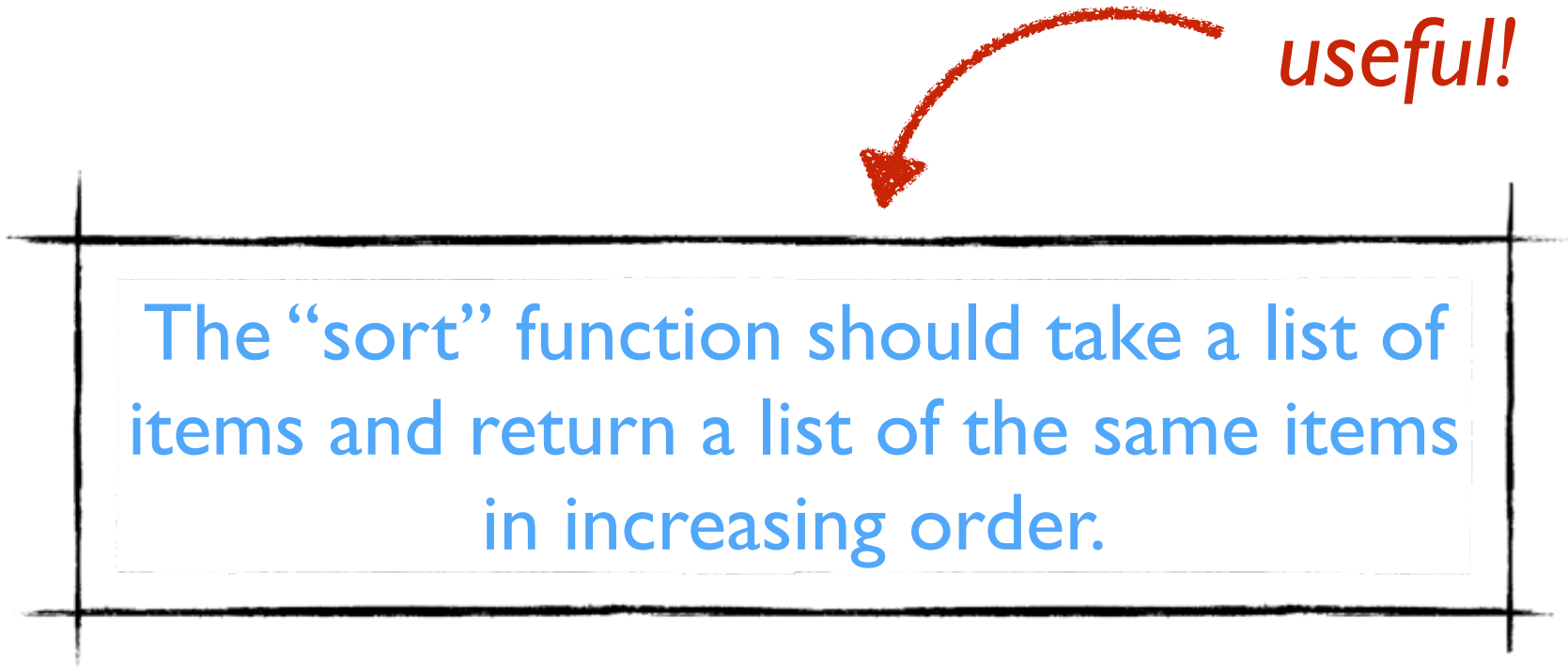
Why are specifications useful?

If you want to build software that works, it is helpful to know what you mean by "works"!

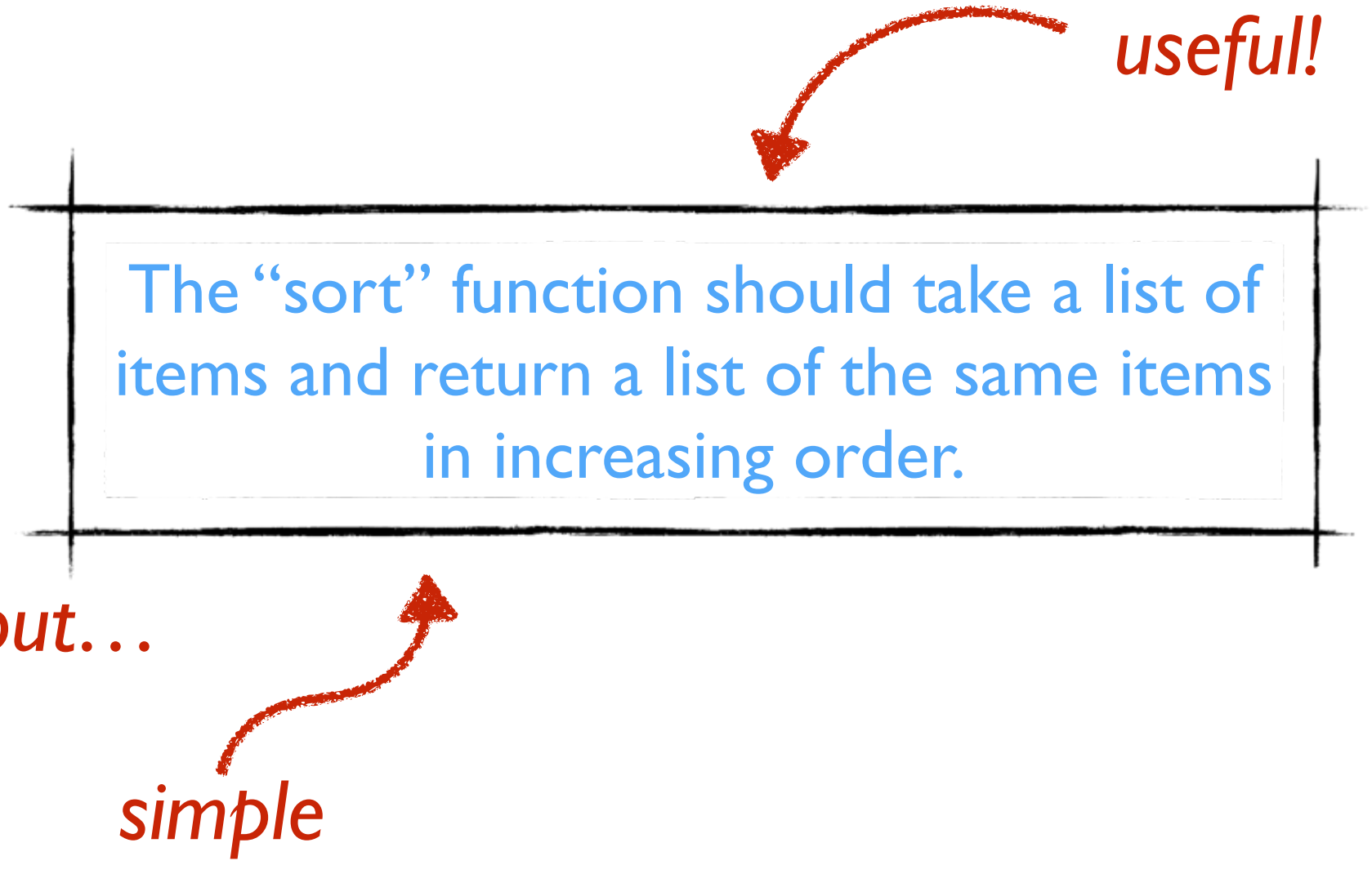
A Specification:

The "sort" function should take a list of items and return a list of the same items in increasing order.

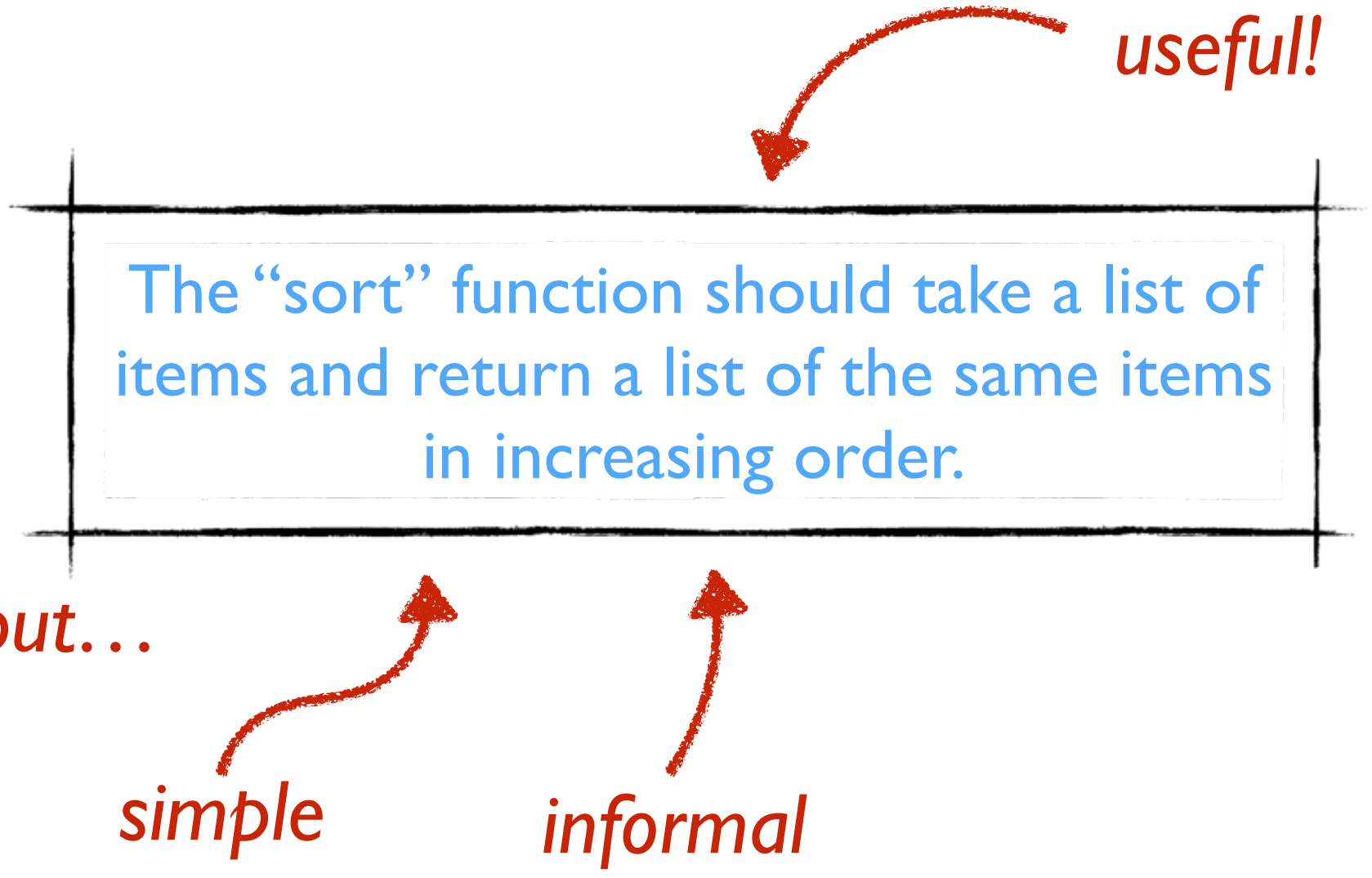
A Specification:



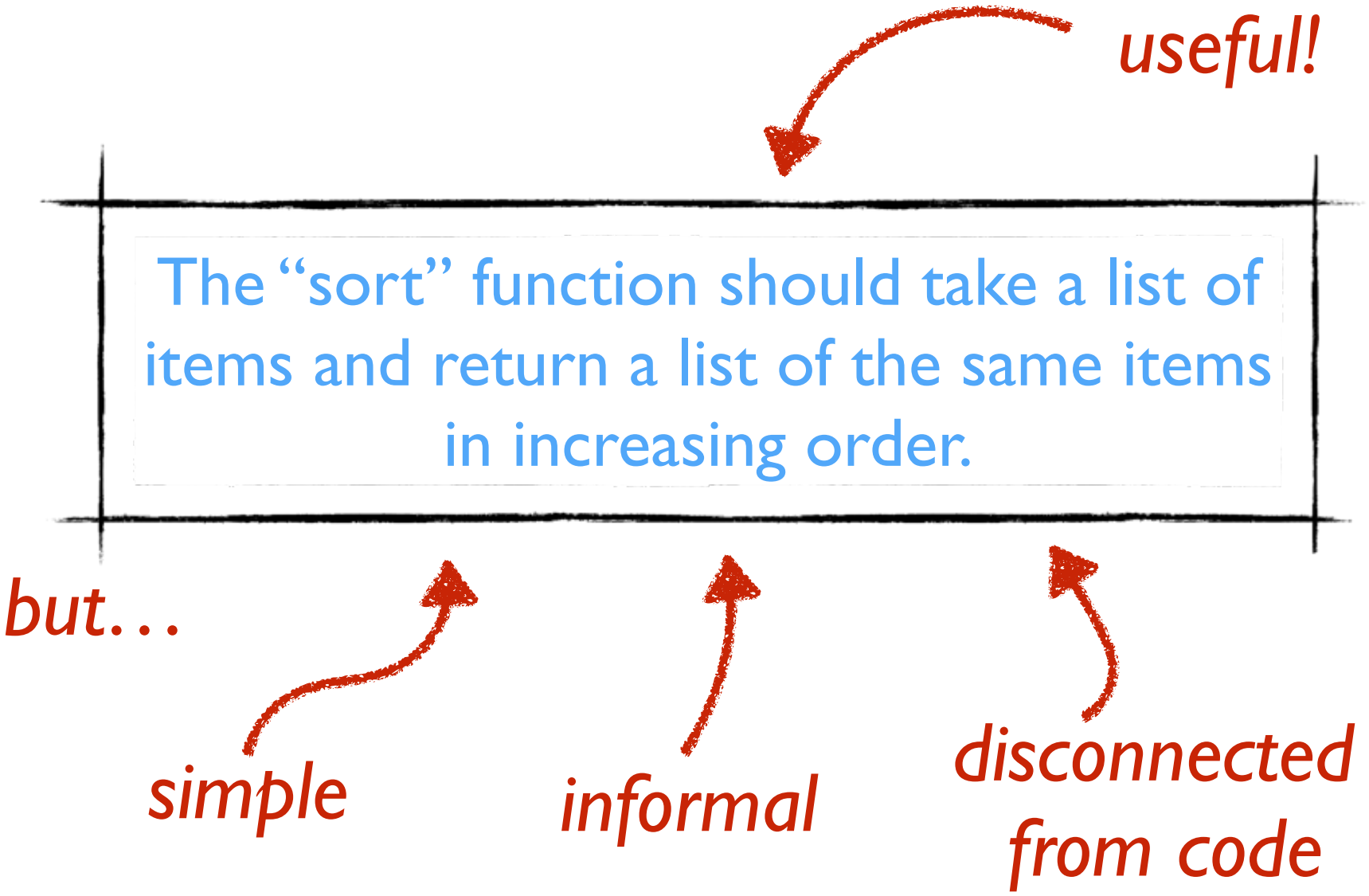
A Specification:

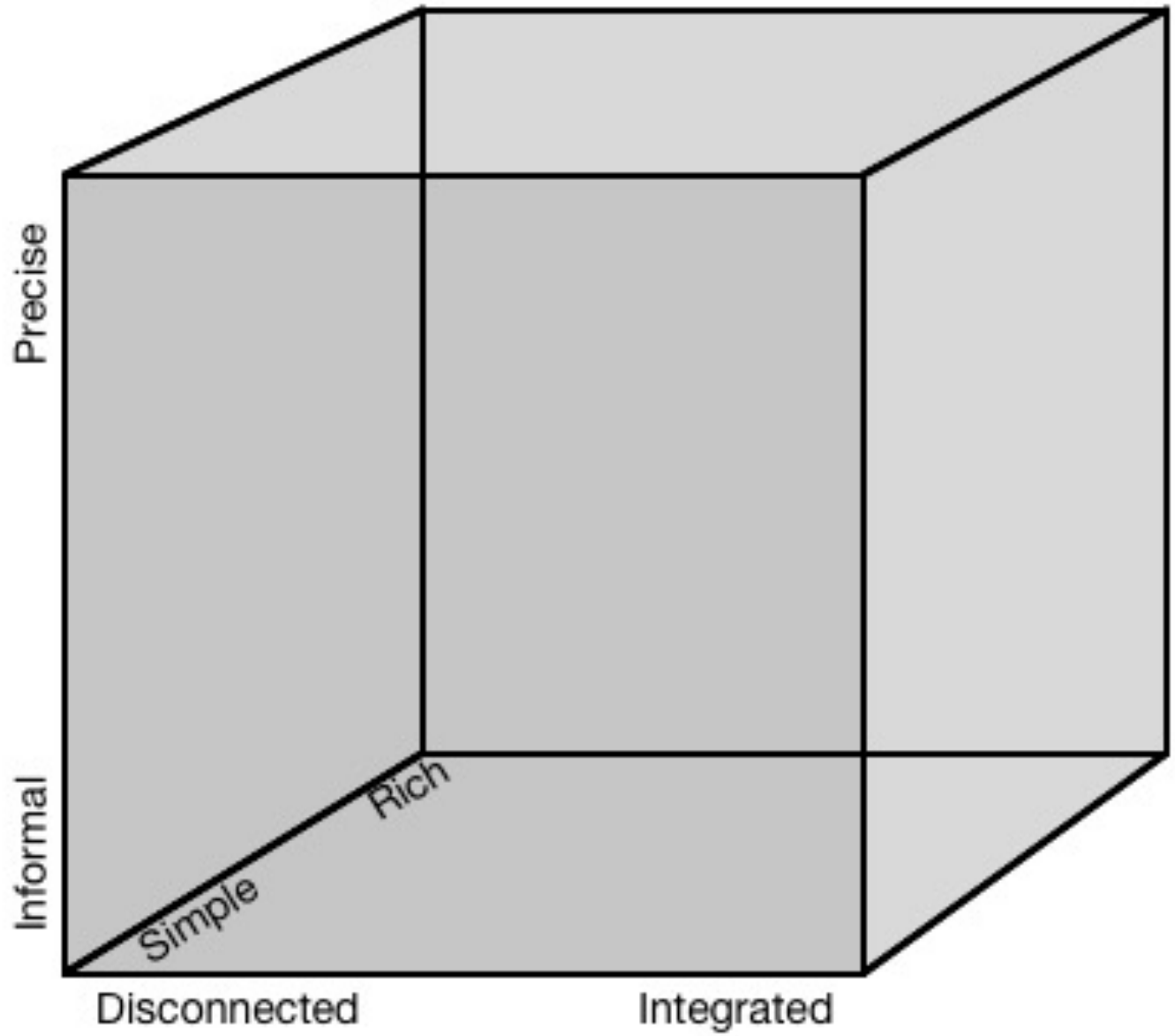


A Specification:



A Specification:




Precise
Informal
Simple
Rich
Disconnected
Integrated

Simple ⟶ Rich

- C Language Reference
  - 592 pages
  - also Java (792 pages), C++ (1354 pages, etc.
- x86 CPU reference
  - 1499 pages
- AUTOSAR standardized automotive architecture
  - 3000 pages

AUTOSAR

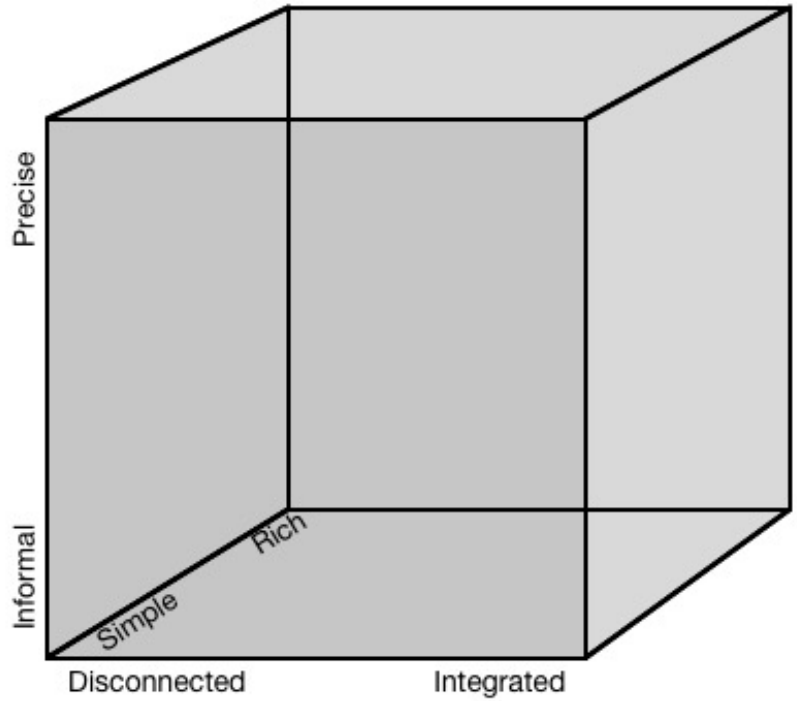
Informal ⟶ Precise

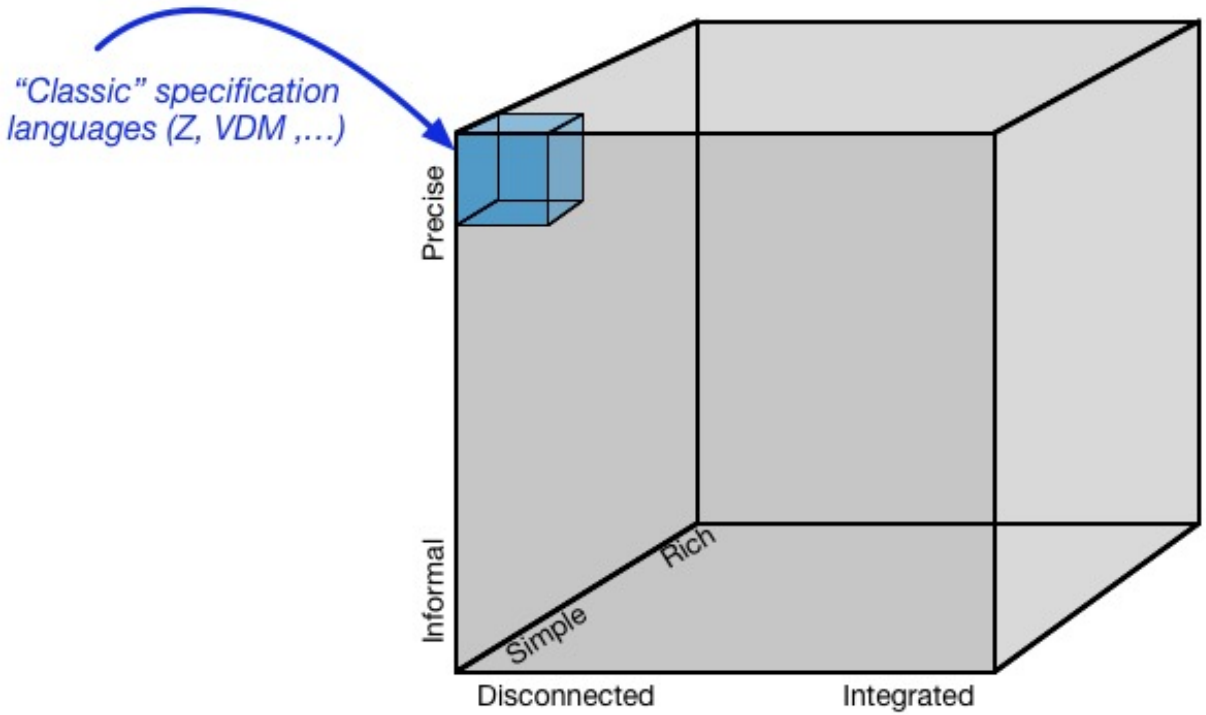
- Z, Alloy, VDM, ACL2, Coq, Isabelle, …
  - x86 instruction set (and many others)
  - Ada, Java virtual machine, C, JavaScript, …
  - …

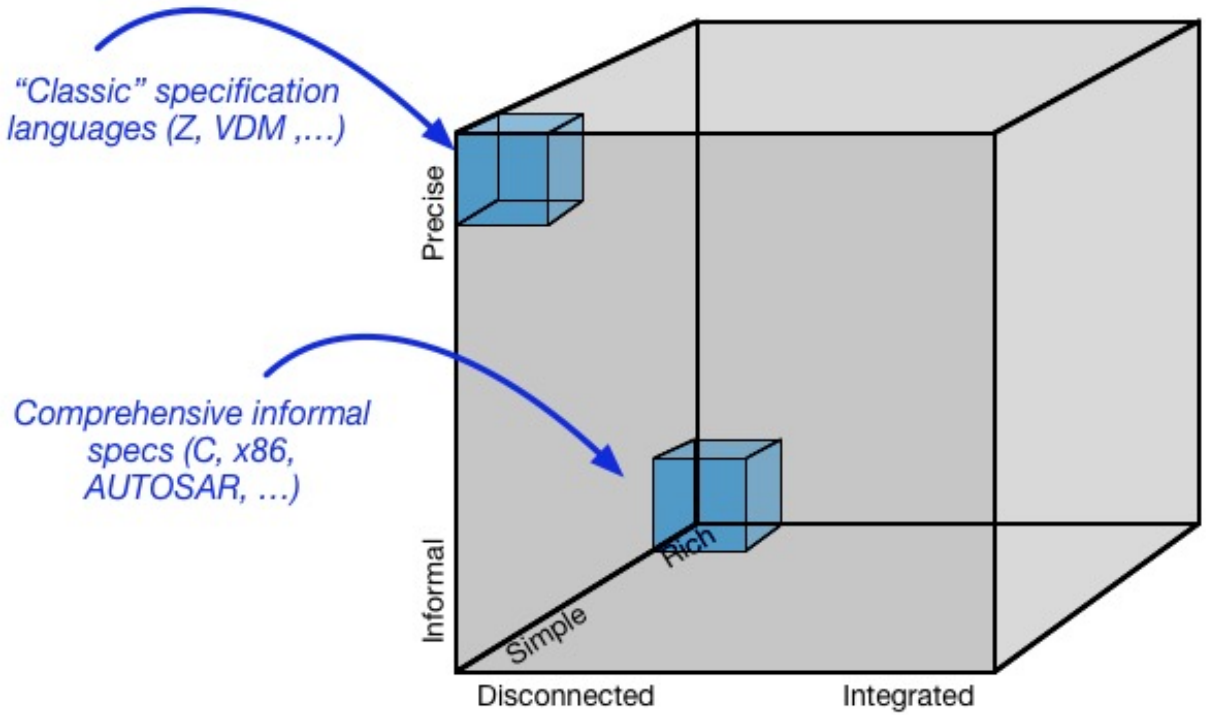
Formal *specification languages*

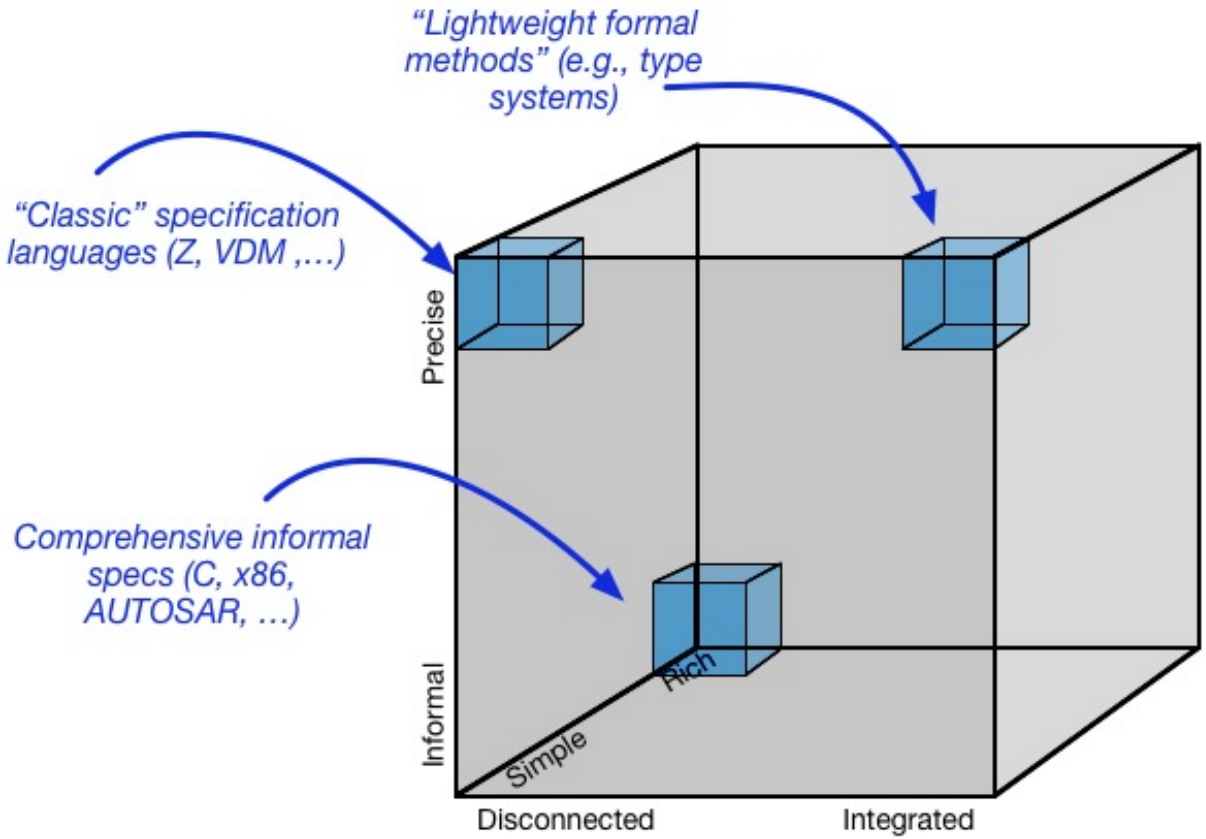
Disconnected ⟶ Integrated

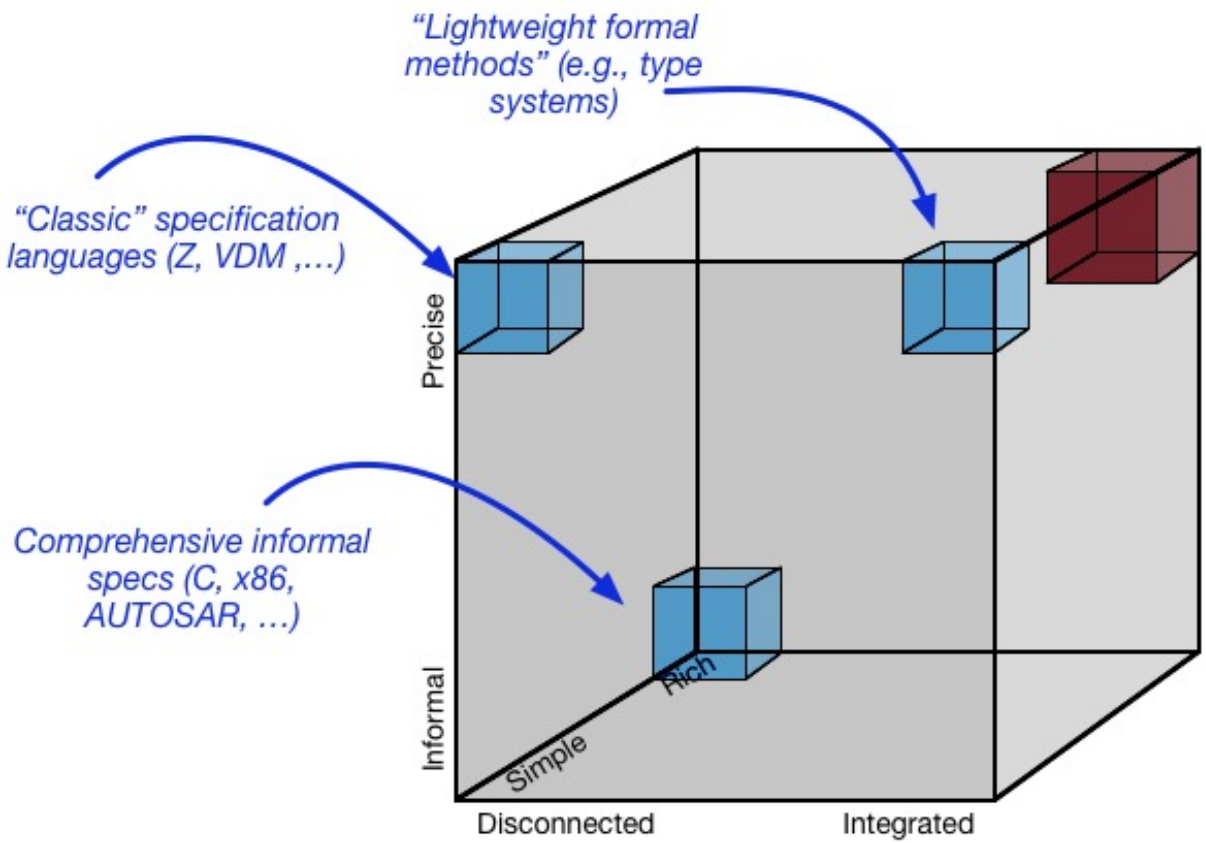
- Formal verification tools
  - Human constructs "proof script"; computer checks it
  - Capable in principle of establishing connections between arbitrary specifications and code
  - Challenging to use at scale
- Type systems
  - Highly successful "lightweight formal methods"
  - Built into programming languages
  - Limited expressiveness, but "always on"

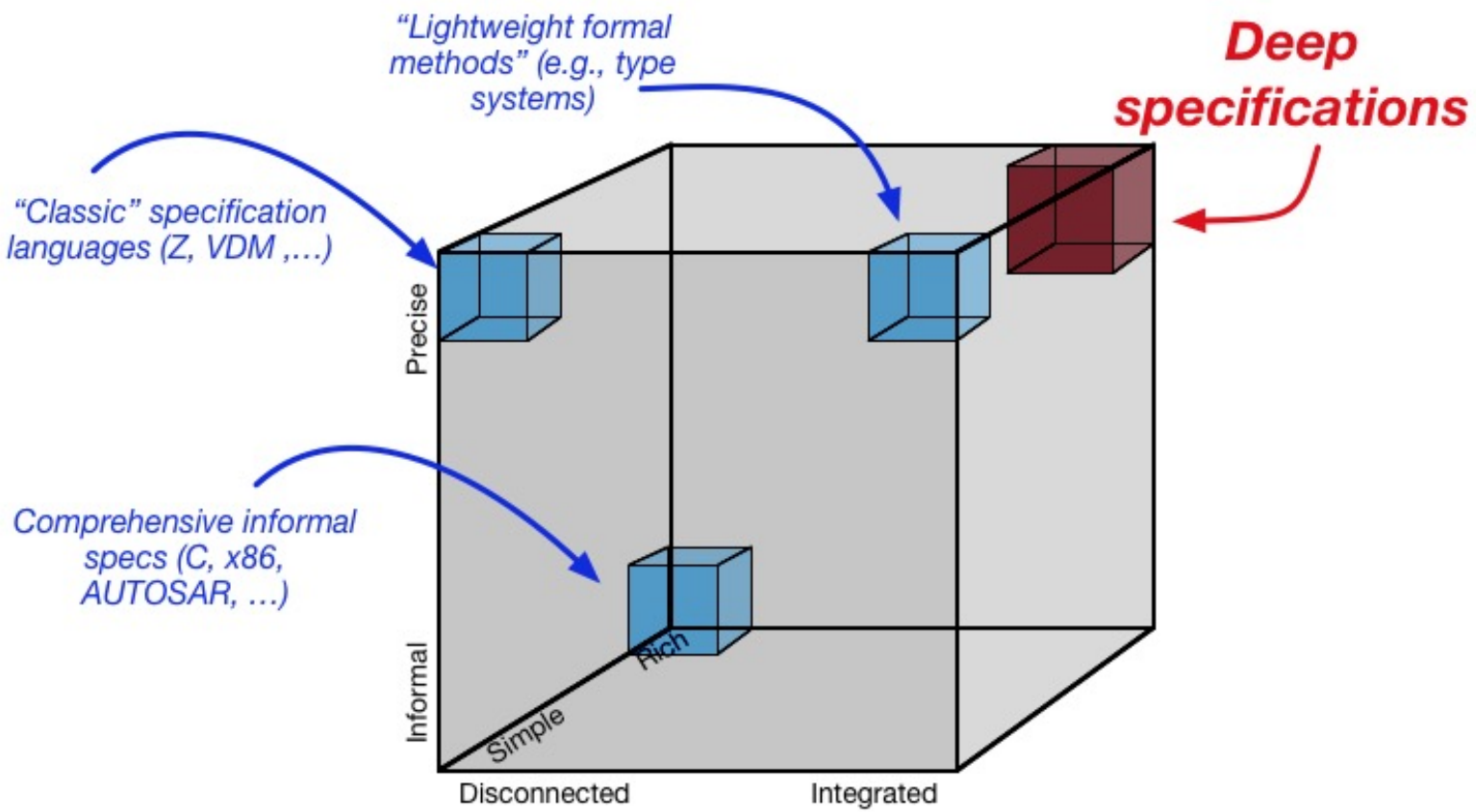

Precise
Informal
Simple
Rich
Disconnected
Integrated


"Classic" specification languages (Z, VDM ,…)
Precise
Informal
Simple
Rich
Disconnected
Integrated


"Classic" specification languages (Z, VDM ,…)
Comprehensive informal specs (C, x86, AUTOSAR, …)
Precise
Informal
Simple
Rich
Disconnected
Integrated


"Lightweight formal methods" (e.g., type systems)
"Classic" specification languages (Z, VDM ,…)
Comprehensive informal specs (C, x86, AUTOSAR, …)
Precise
Informal
Simple
Rich
Disconnected
Integrated


"Lightweight formal methods" (e.g., type systems)
"Classic" specification languages (Z, VDM ,…)
Comprehensive informal specs (C, x86, AUTOSAR, …)
Precise
Informal
Rich
Simple
Disconnected
Integrated

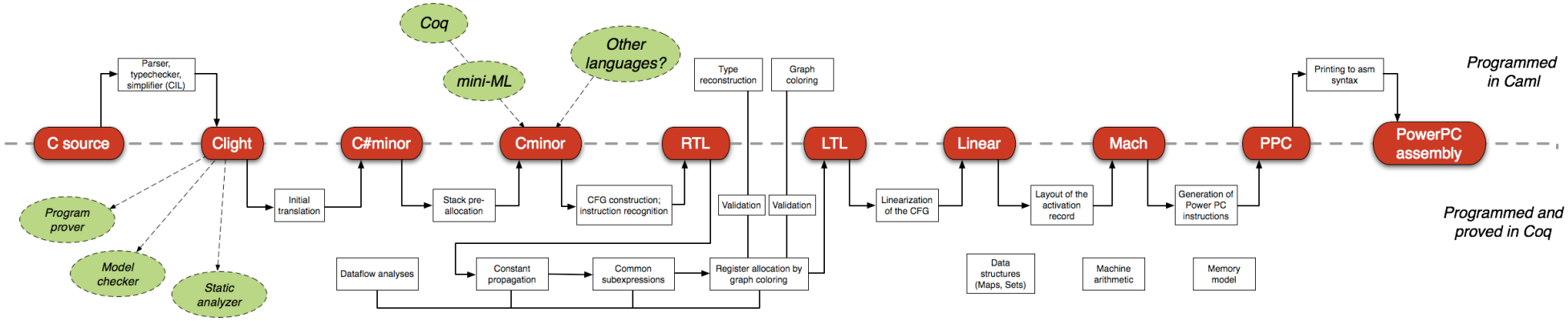

"Lightweight formal methods" (e.g., type systems)
Deep specifications
"Classic" specification languages (Z, VDM ,…)
Comprehensive informal specs (C, x86, AUTOSAR, …)
Precise
Informal
Rich
Simple
Disconnected
Integrated

Deep specifications

1. Rich
2. Formal
3. Integrated with code

early *tours de force*…

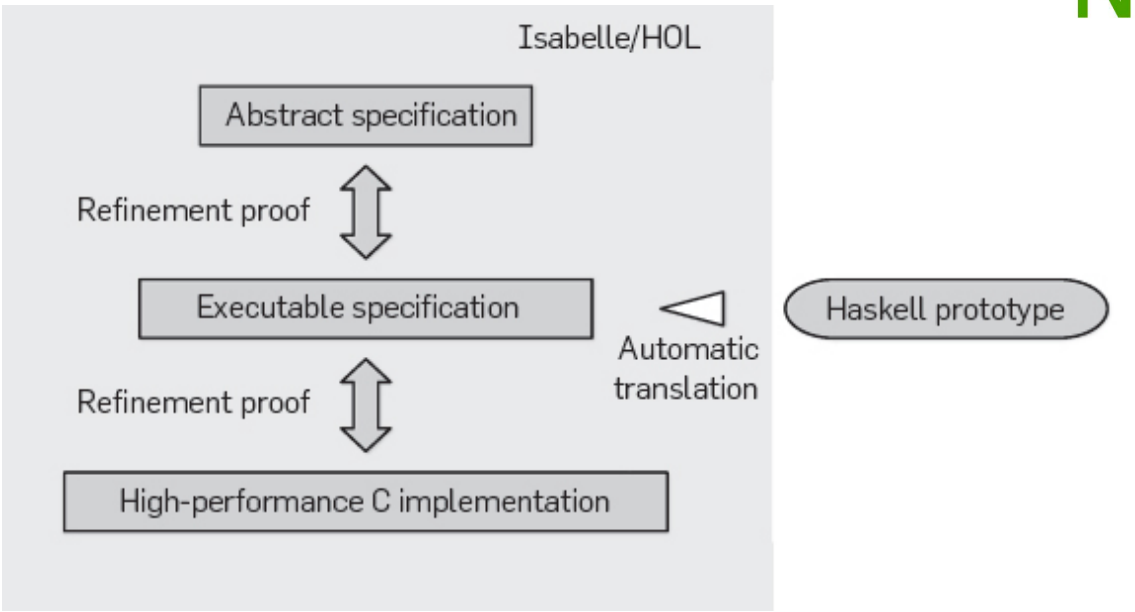
CompCert C compiler



- Fully verified translator from C to machine code
- Accepts most of ISO C 99
- Produces machine code for PowerPC, ARM, and IA32 (x86 32-bit) architectures
- 90% of the performance of GCC (v4, opt. level 1)

seL4

NICTA



Real-world operating-system kernel with an end-to-end proof of implementation correctness and security enforcement

Emerging trends…

New specification / verification tools

- Coq
- Isabelle
- ACL2
- …

Powerful
proof assistants and
program logics

- F*
- Dafny
- Boogie
- …

Quasi-automatic verifiers
based on SMT solvers

Formal verification of real software

- Verified TLS implementation
  - (Core technology for secure web communications)
- Verified compilers
  - CakeML, Bedrock, CompCertTSO, …
- Verified distributed systems
  - Verdi, …
- Verified operating systems and OS components
  - CertiKOS, Ironclad Apps, Jitk, …
- Verified cryptography
- …

Expressive type systems

- security types
- session types
- component types / object types / module systems
- generalized abstract datatypes
- …

Property-based random testing

- TCP networking protocol suite [Sewell et al., Cambridge]
- Testable AutoSAR model [Quviq, Göteborg]
  - Found >200 faults in AUTOSAR Basic Software, including >100 inconsistencies in the informal standard
- Testable model of Dropbox and other synchronization frameworks [ongoing work with Quviq]
- …

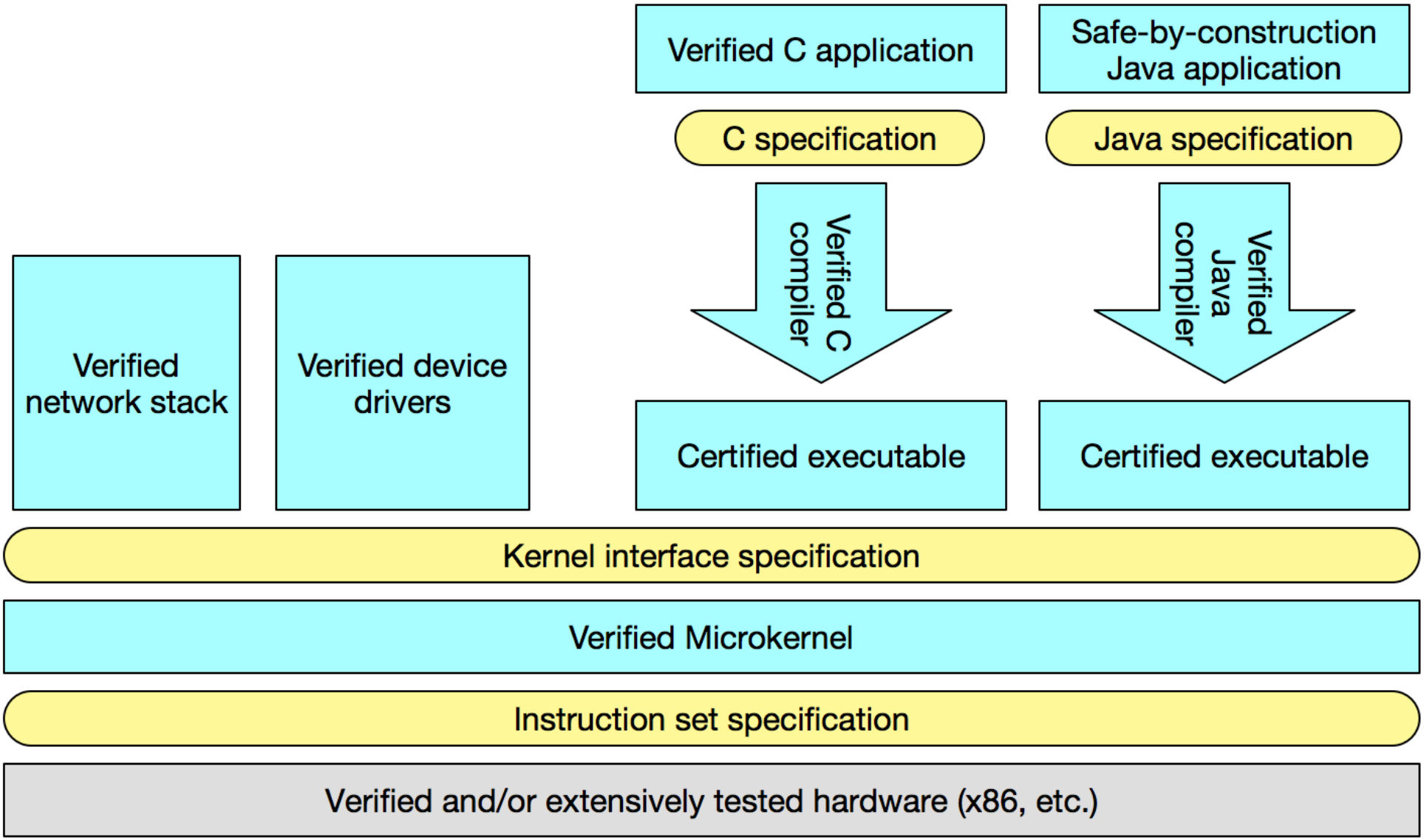
Where are we going?

Where are we going?

Where are we going?

One possibility…

A zero-vulnerability software stack



A zero-vulnerability software stack

Thank you!

(Any questions?)