Filesystems deserve verification too!

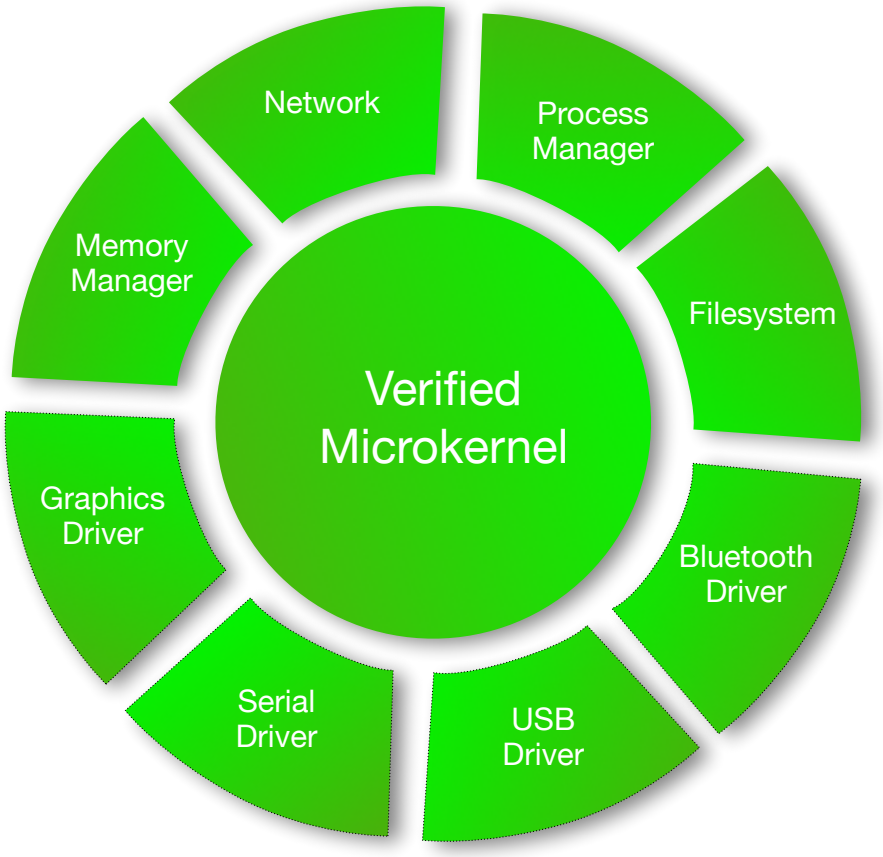
Gabriele Keller

Toby Murray, Sidney Amani, Liam O'Connor,
Zilin Chen, Leonid Ryzhyk Gerwin Klein
Gernot Heiser

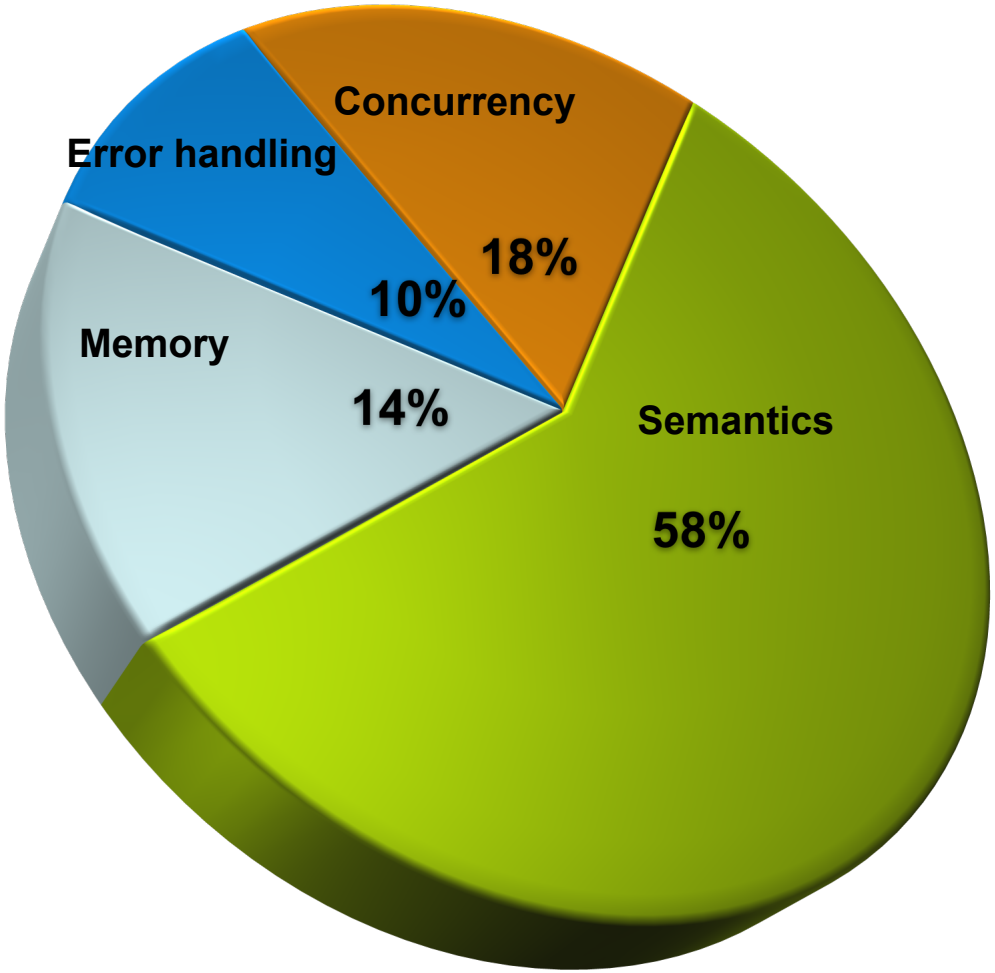
Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners
Australian National University
UNSW
NSW Trade & Investment
Victoria
The University of Melbourne
The University of Sydney
Queensland Government
Griffith University
QUT
The University Of Queensland

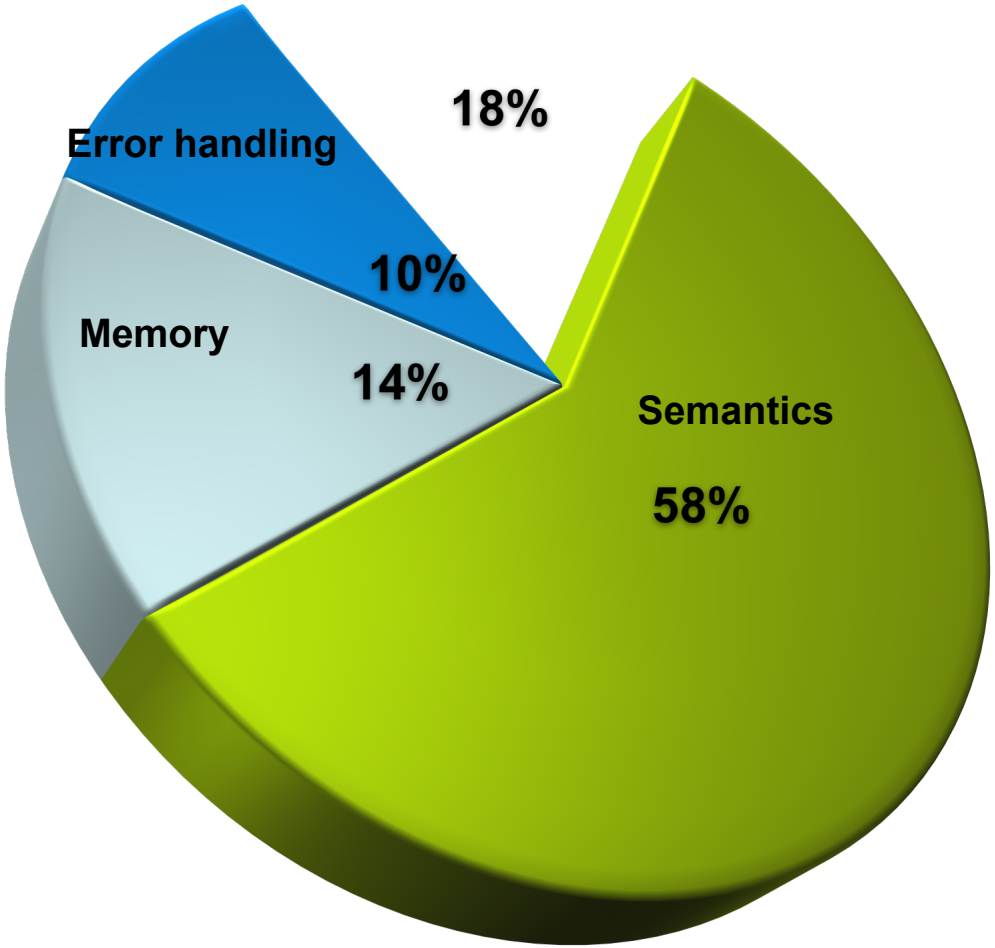
Towards a fully verified System

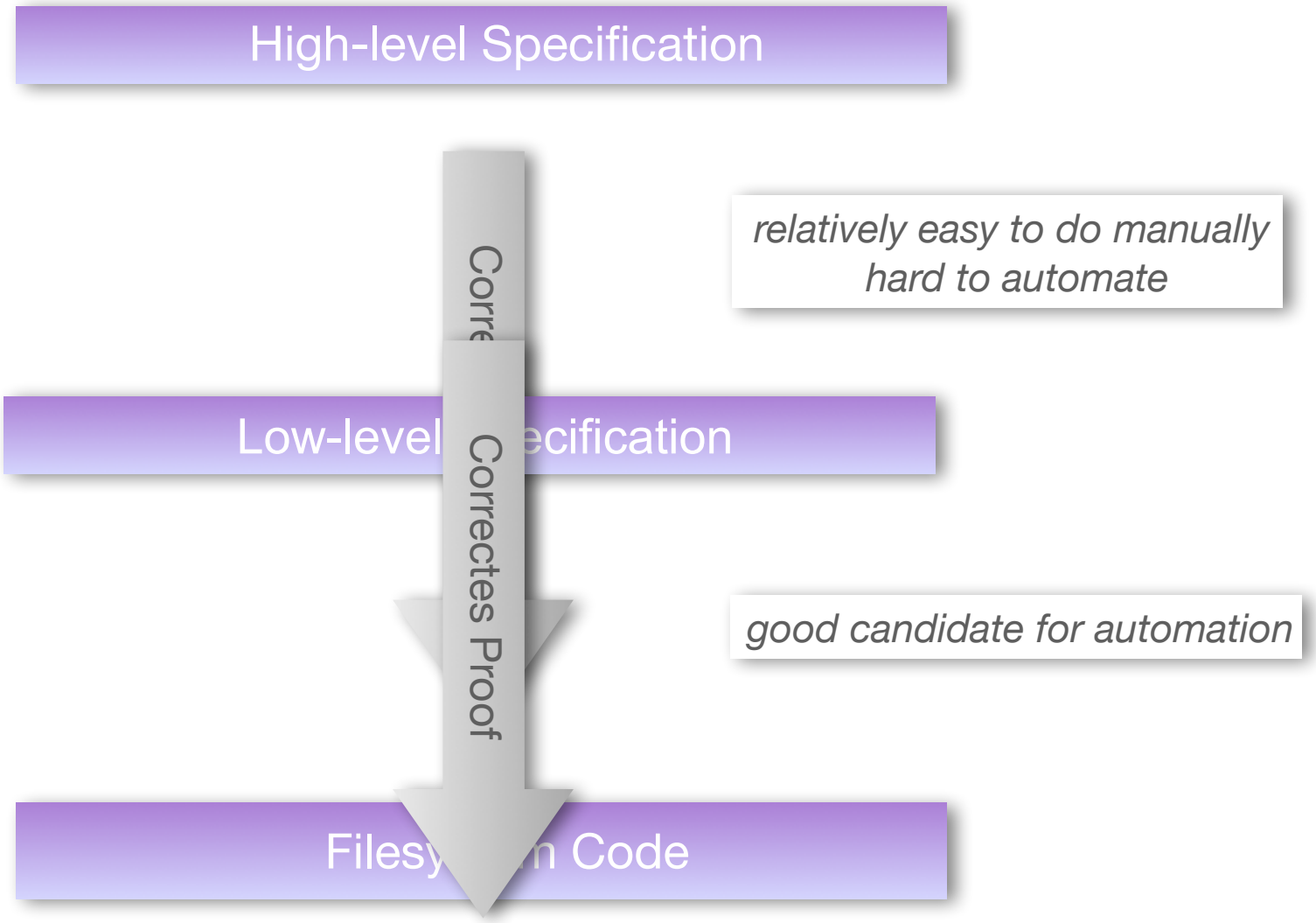


Causes of file system bugs

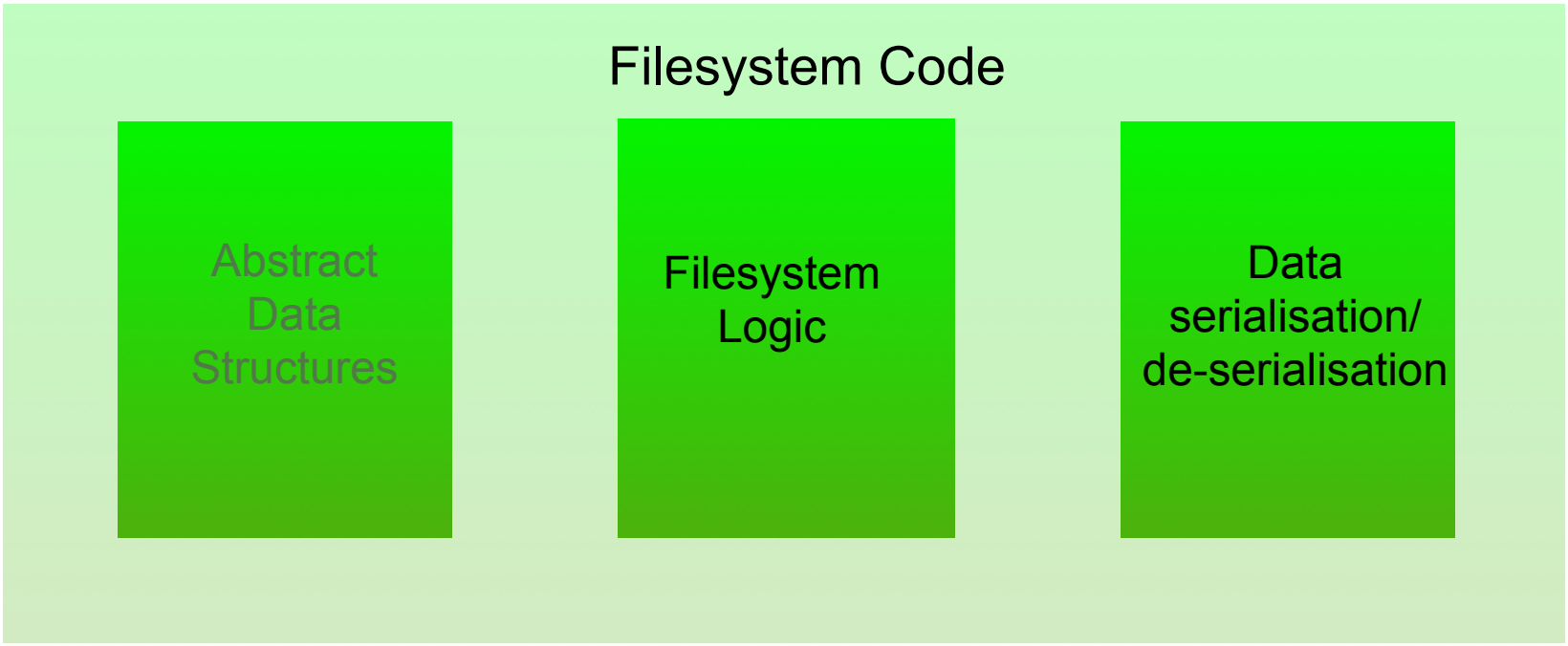


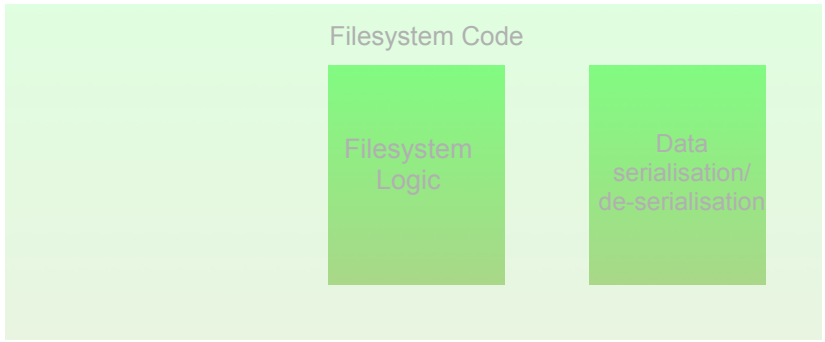
Causes of file system bugs




High-level Specification
Corres
Low-level ecification
Filesy n Code
Corrector Proof
relatively easy to do manually
hard to automate
good candidate for automation

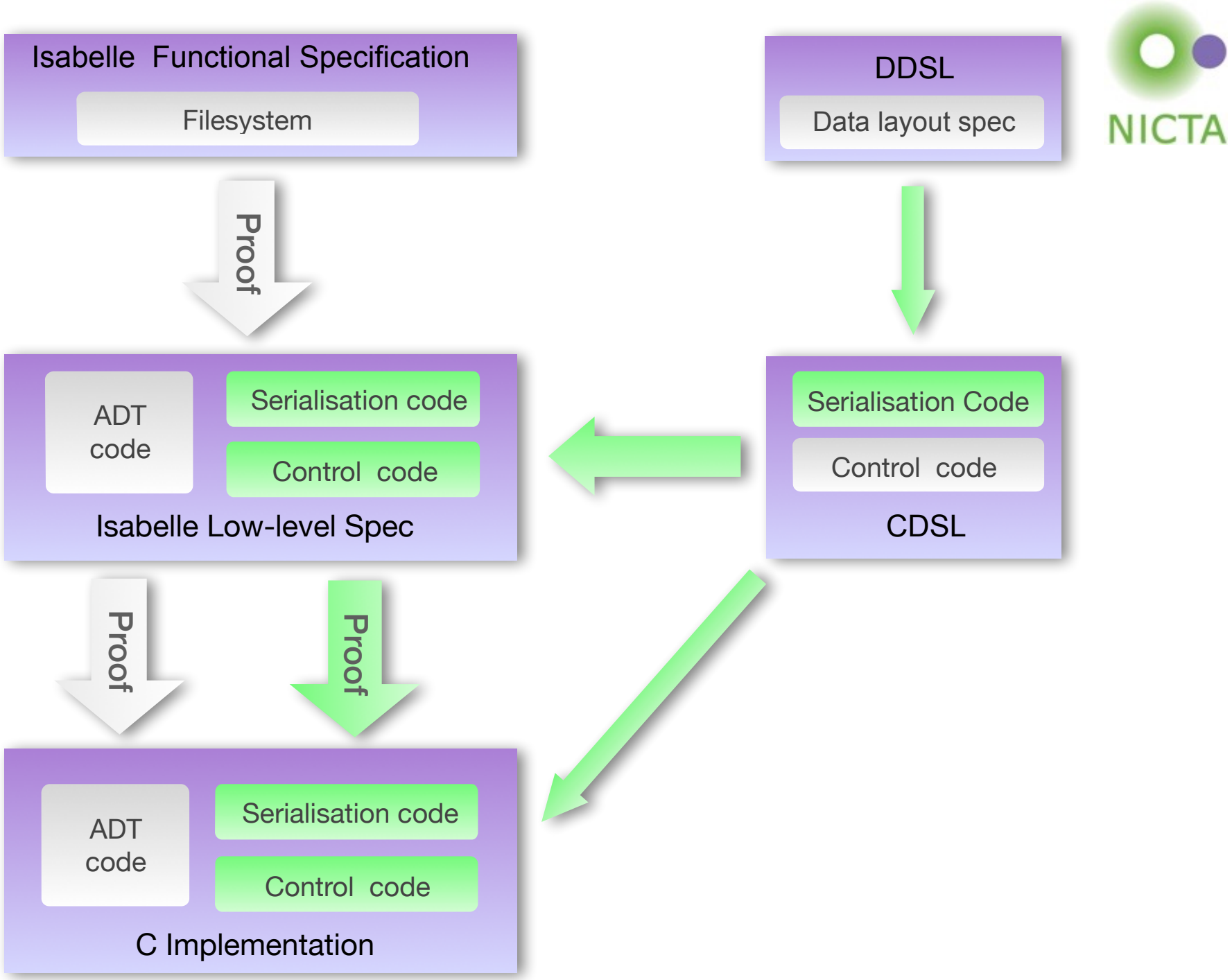

Abstract
Data
Structures
Filesystem
Logic
Filesystem Code
Data
serialisation/
de-serialisation

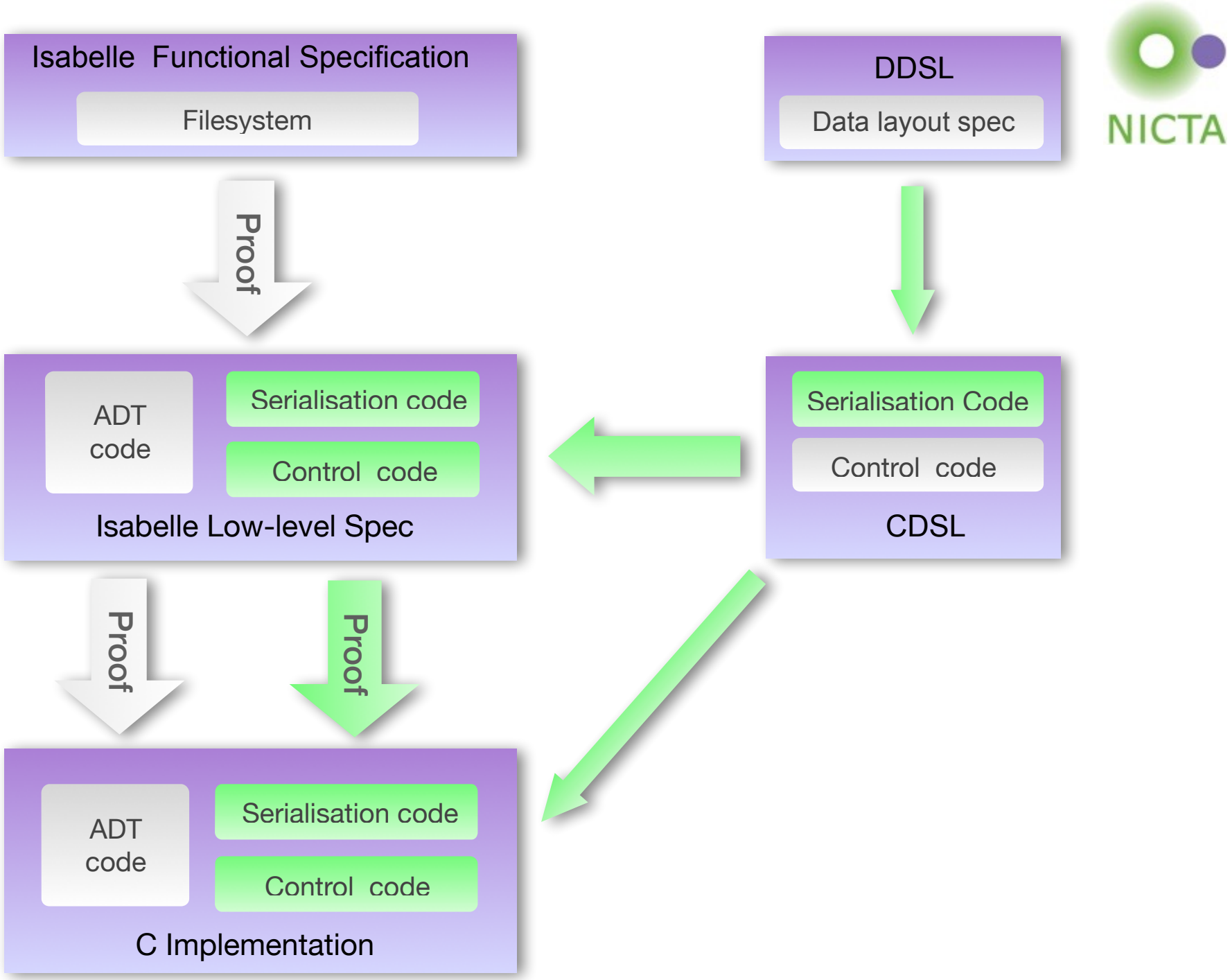


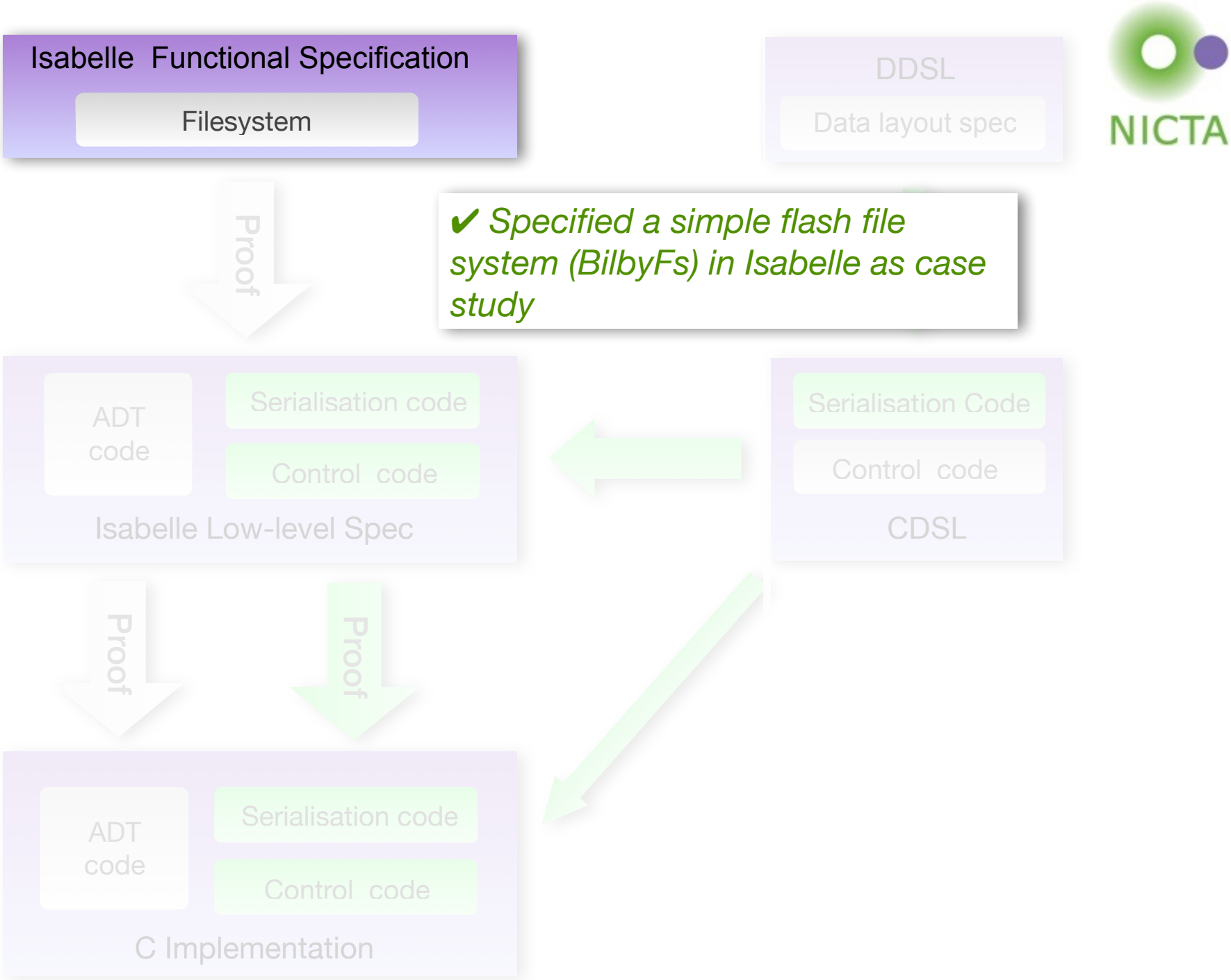


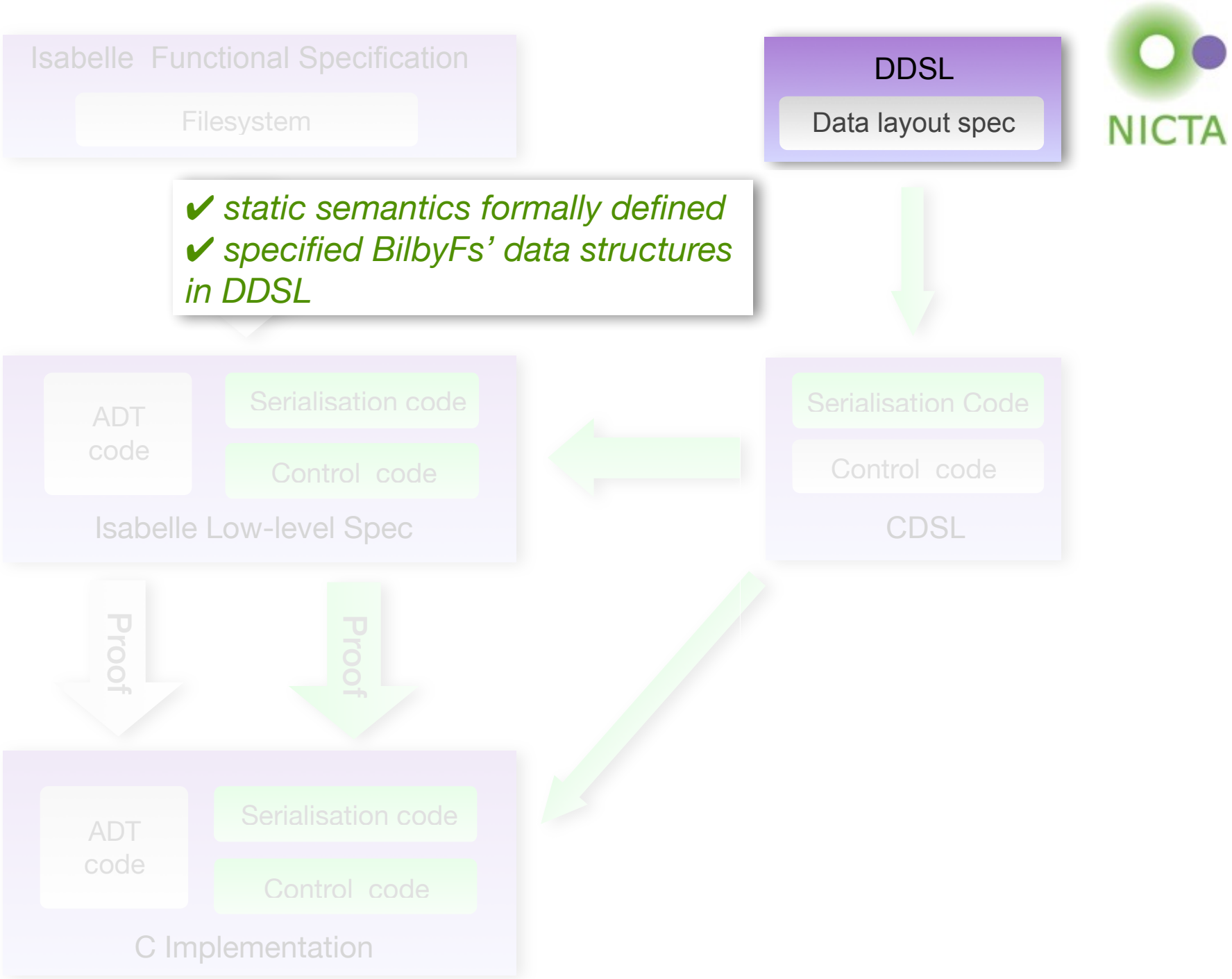
Filesystem Logic

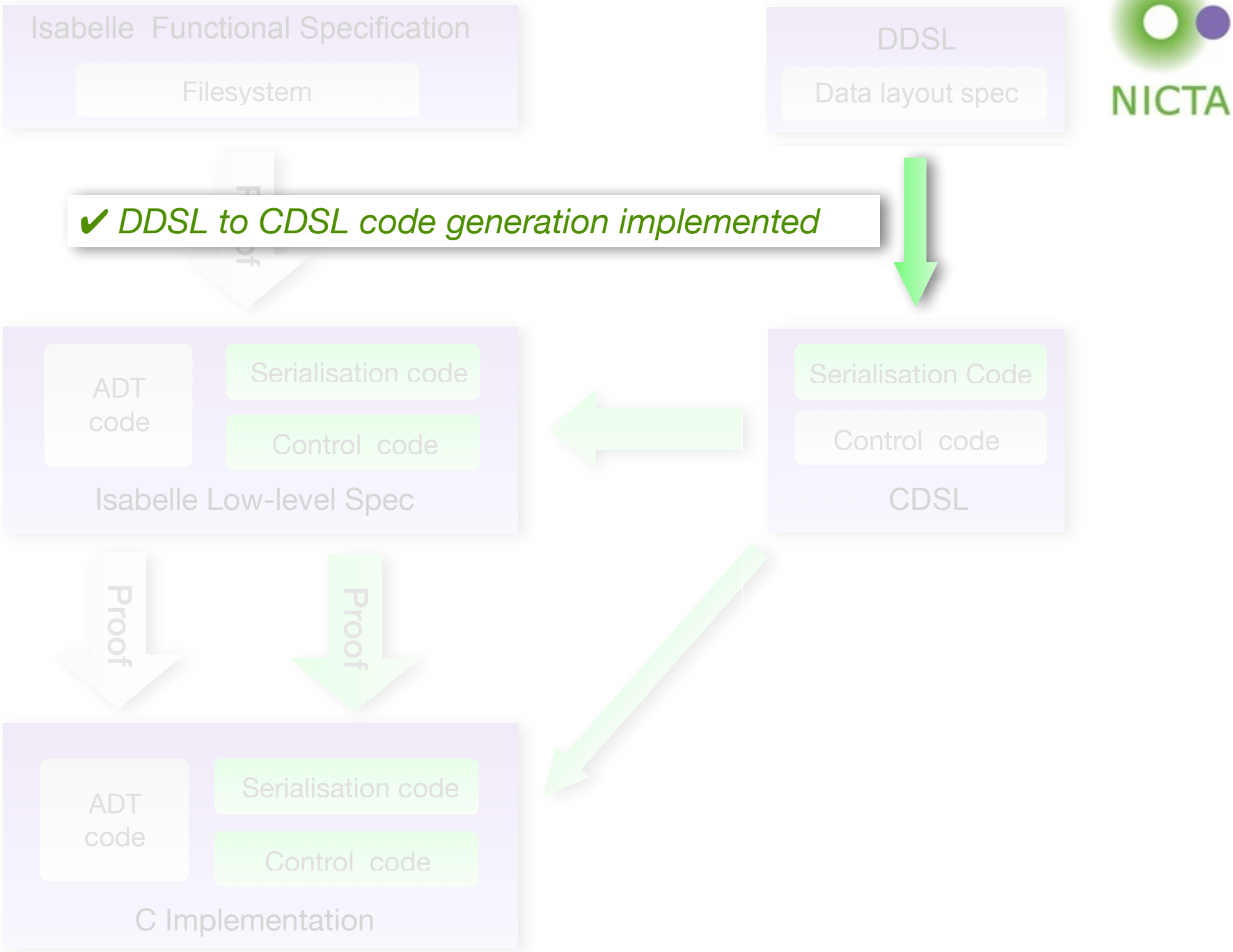
Data serialisation/ de-serialisation

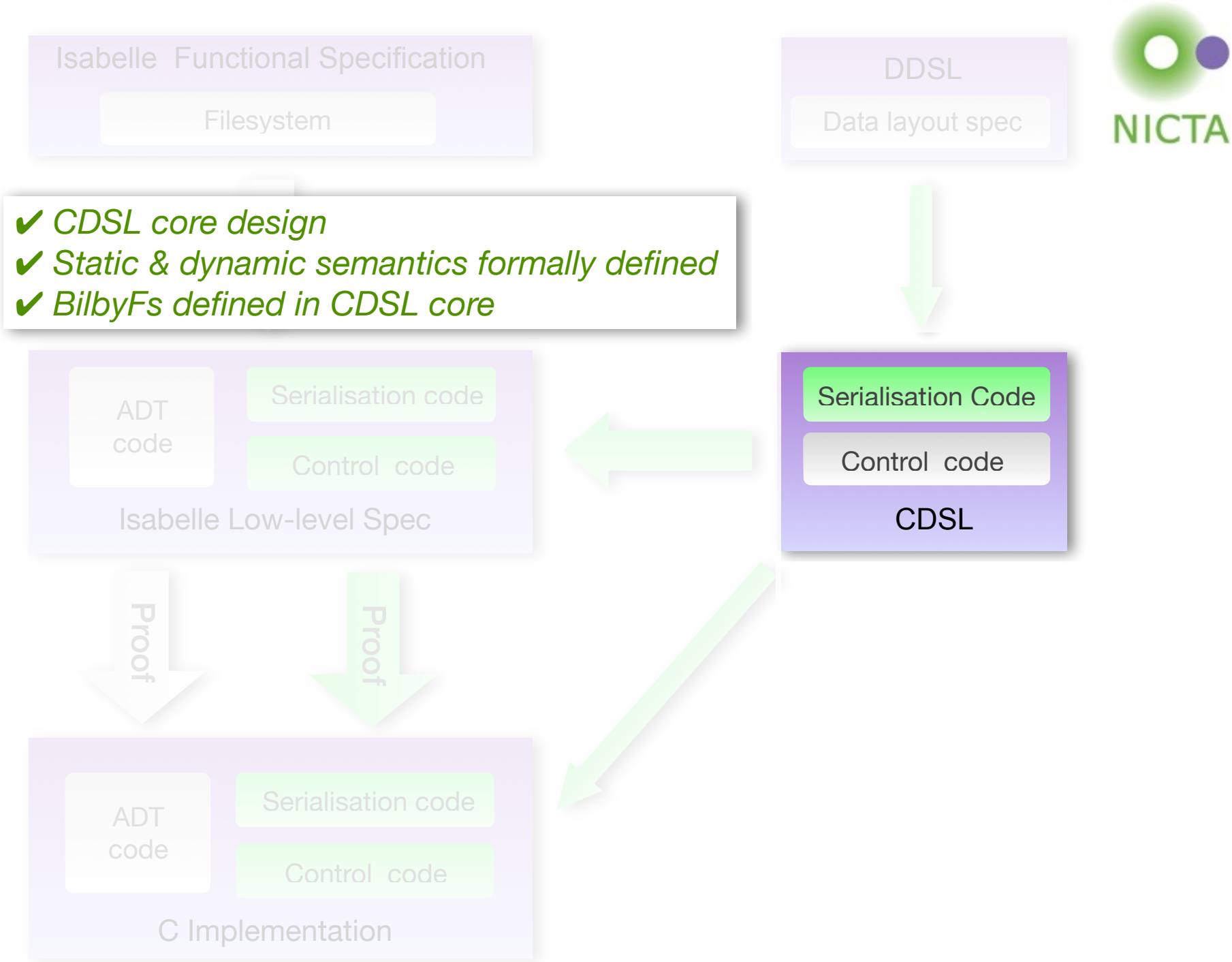

Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
NICTA
Proof
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation

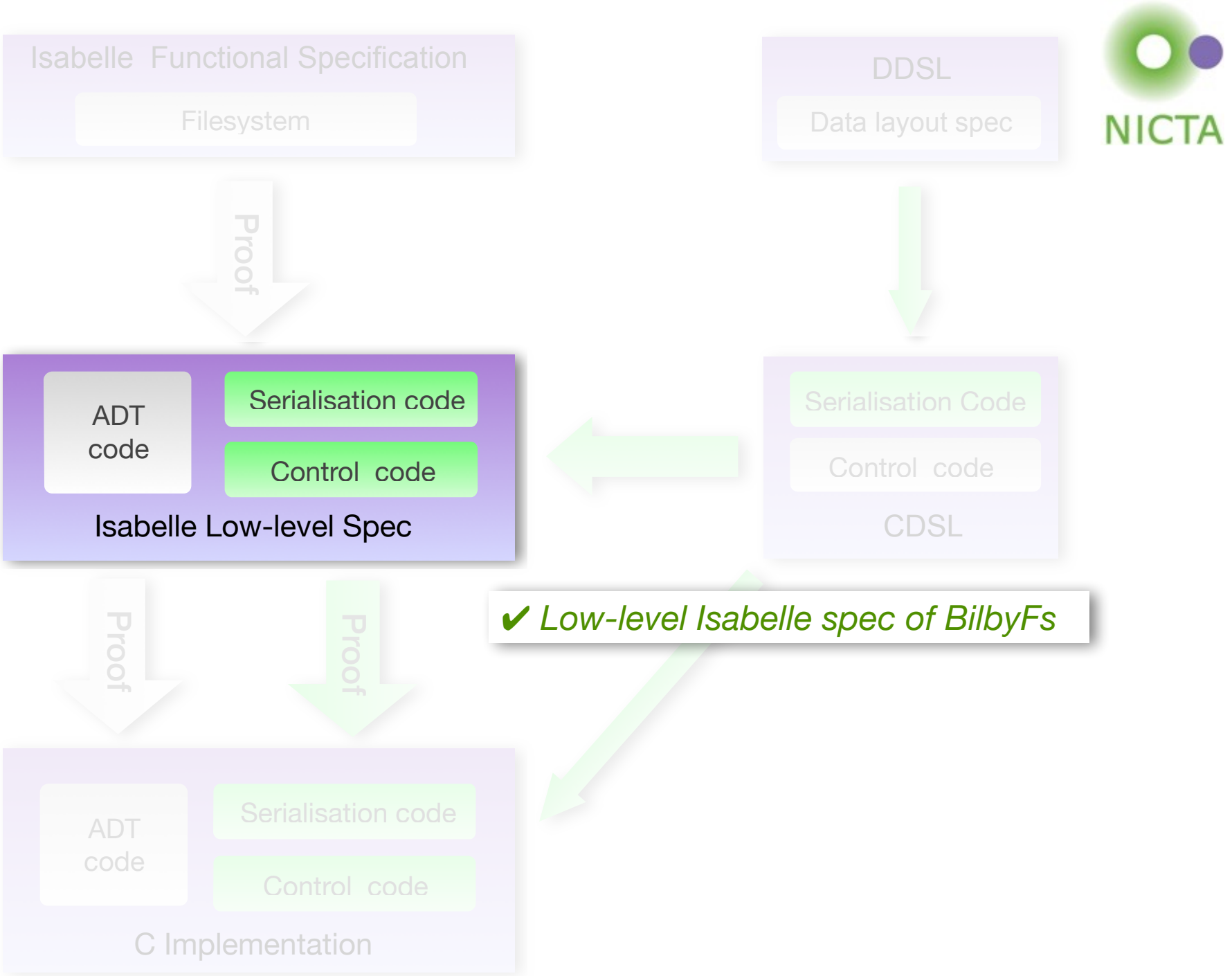

Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
NICTA
Proof
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation

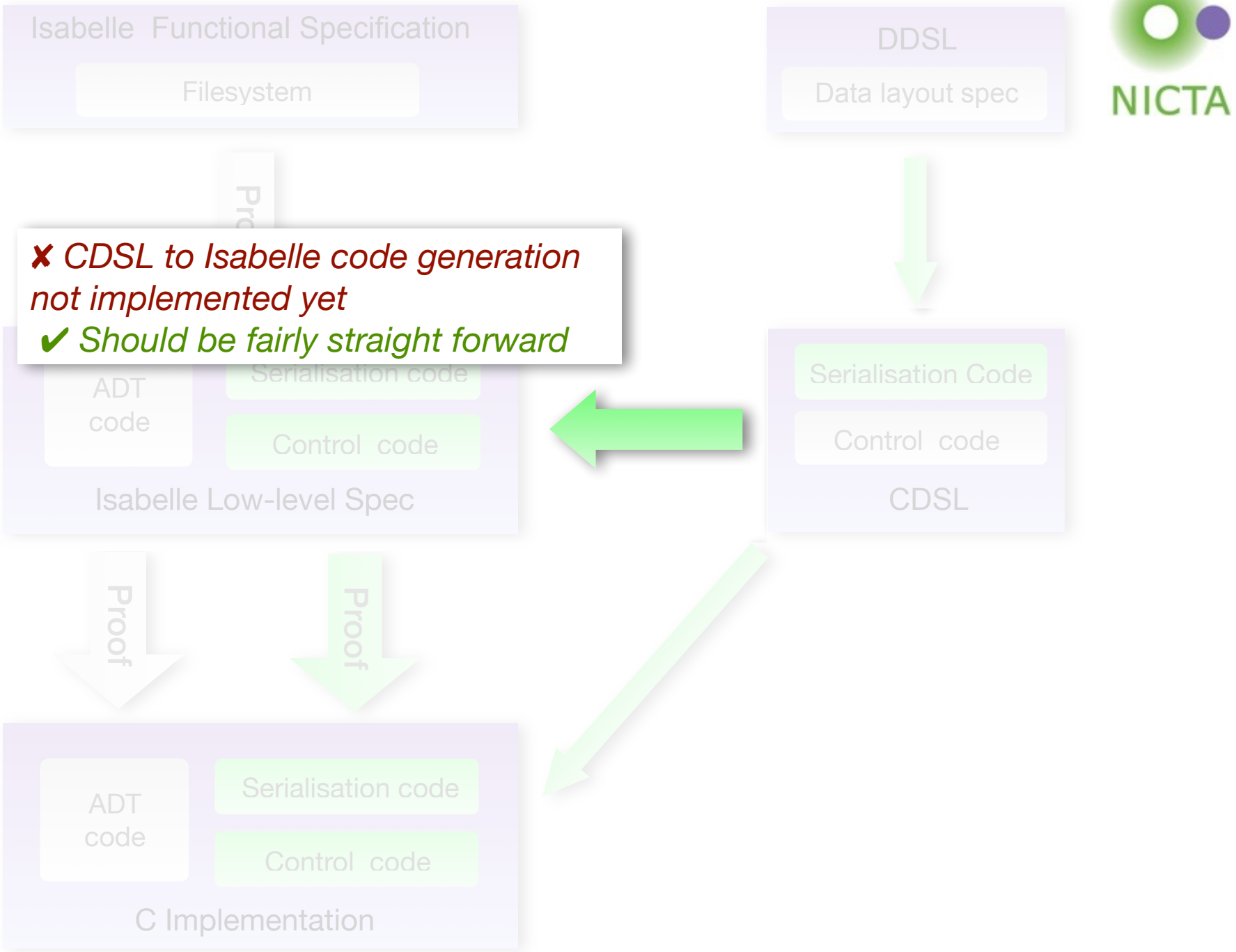

Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
NICTA
Proof
✔ Specified a simple flash file system (BilbyFs) in Isabelle as case study
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation

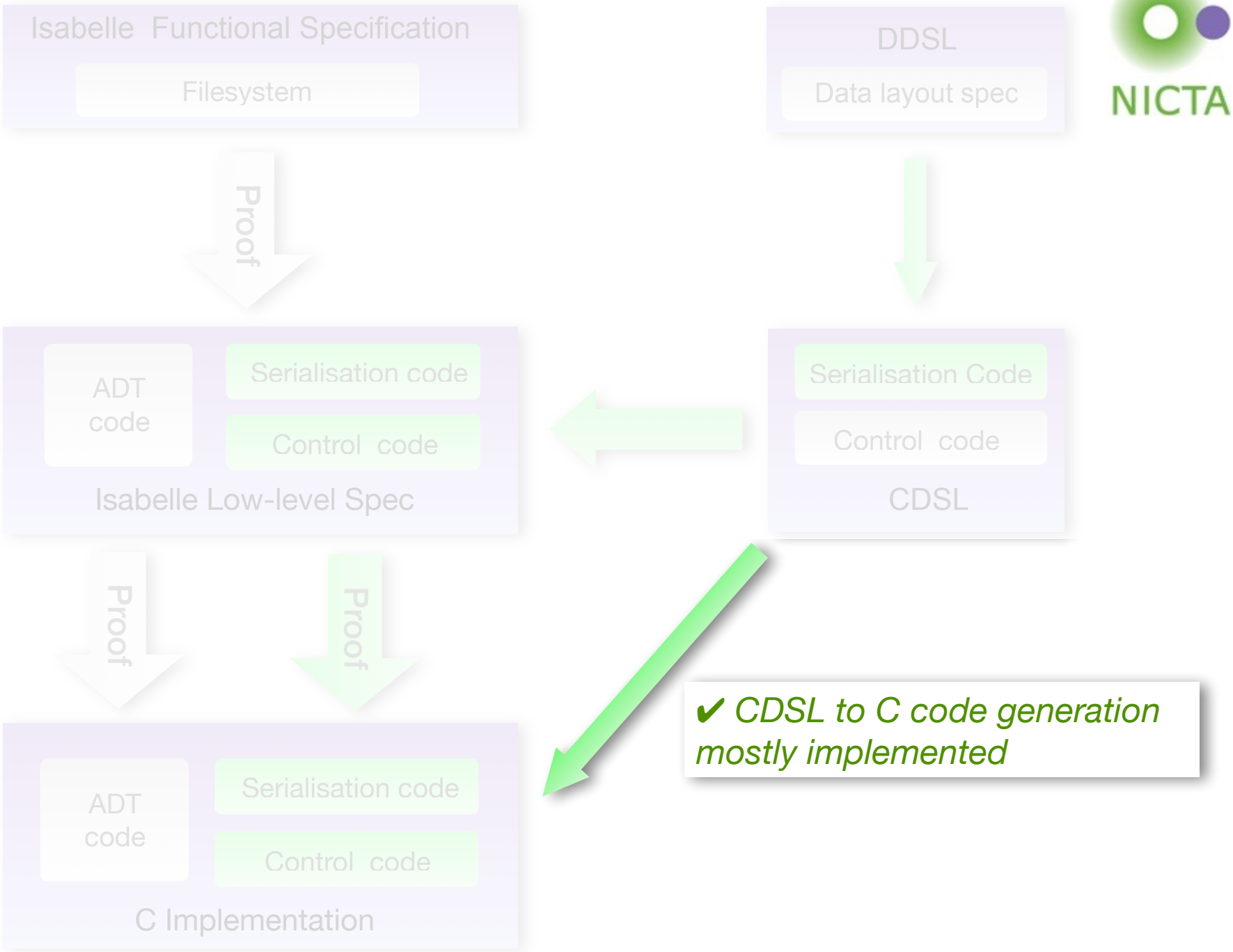

Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
NICTA
✔ static semantics formally defined
✔ specified BilbyFs' data structures in DDSL
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation

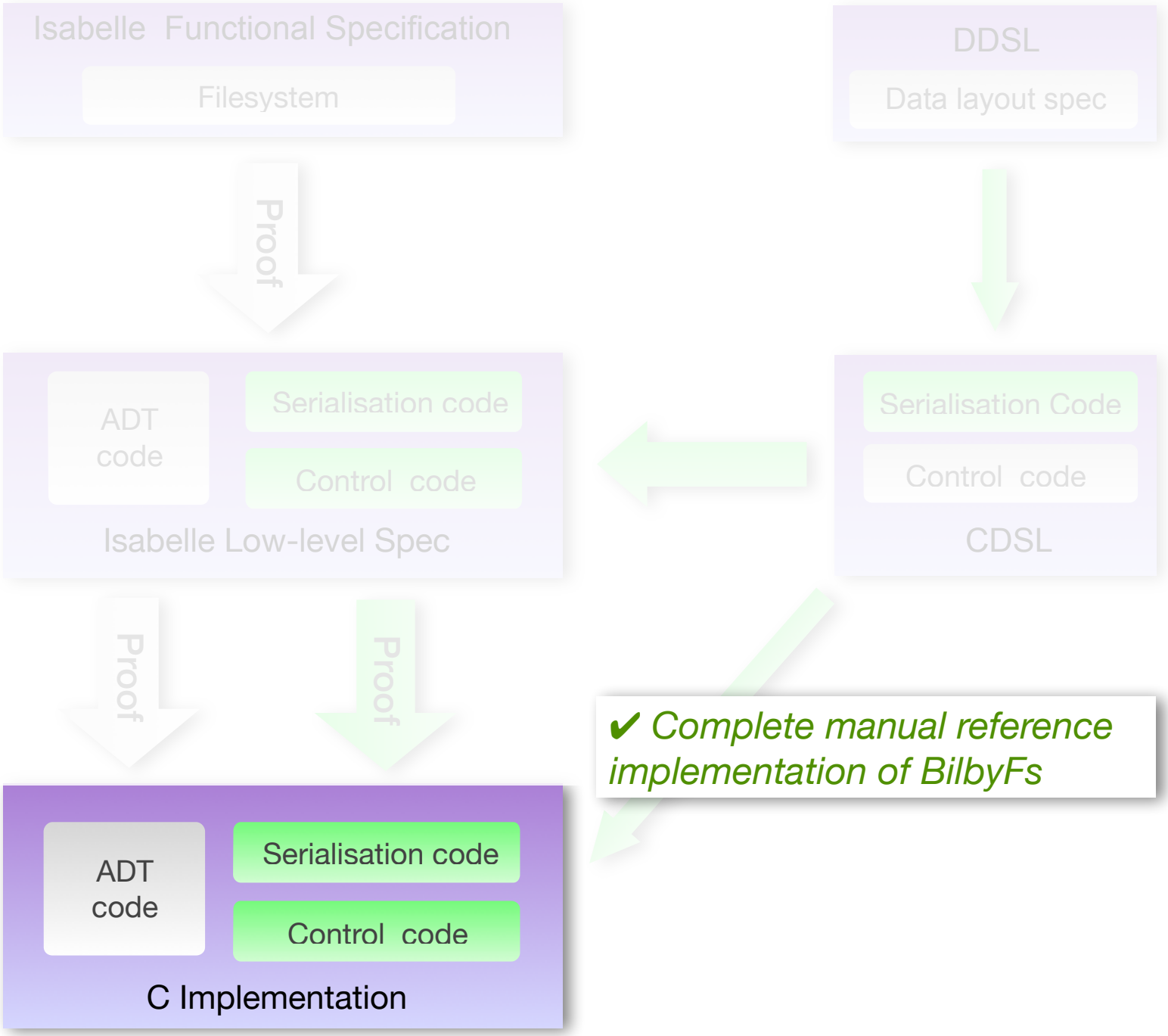

Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
✔ DDSL to CDSL code generation implemented
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation

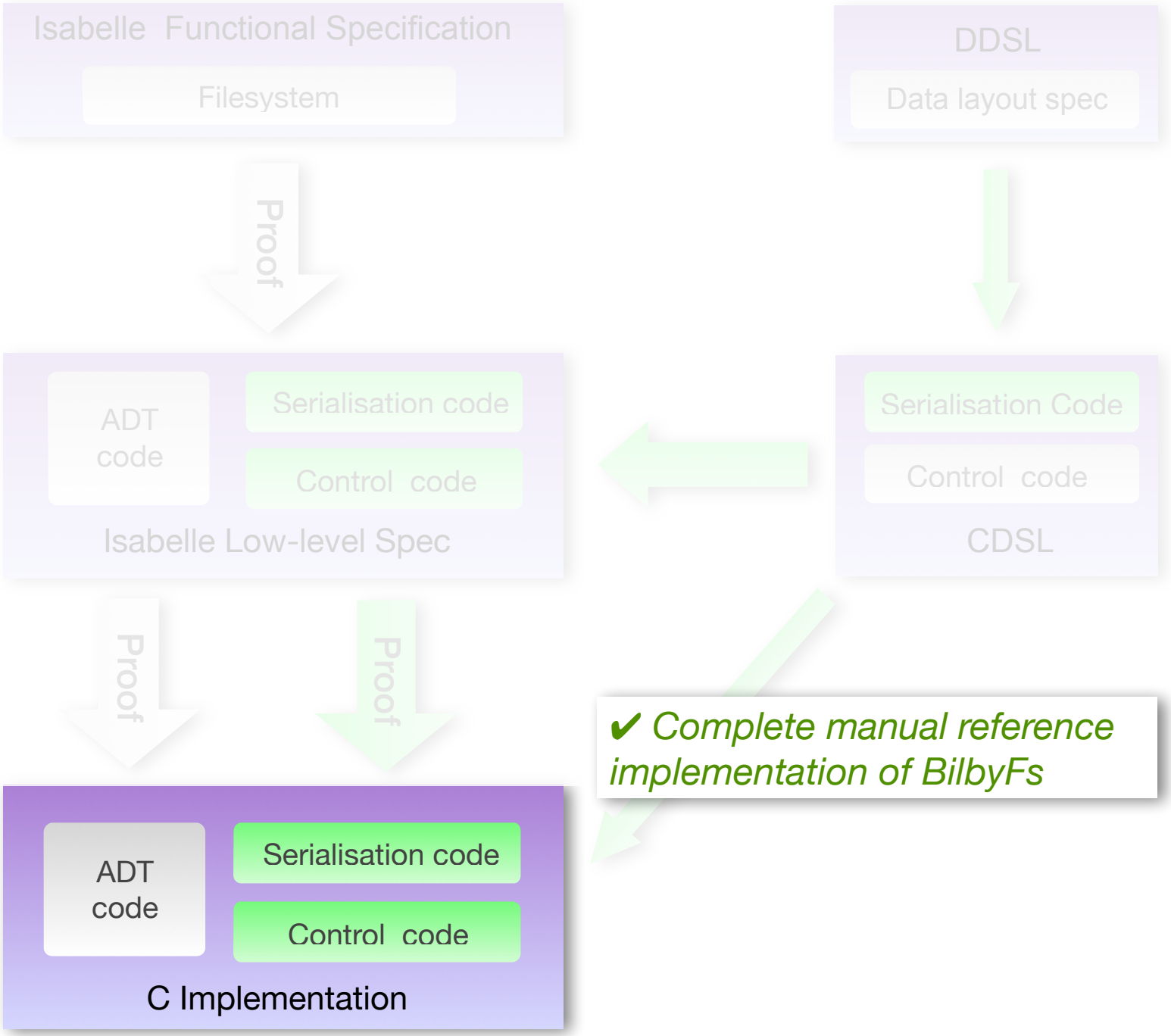

Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
NICTA
✔ CDSL core design
✔ Static & dynamic semantics formally defined
✔ BilbyFs defined in CDSL core
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation

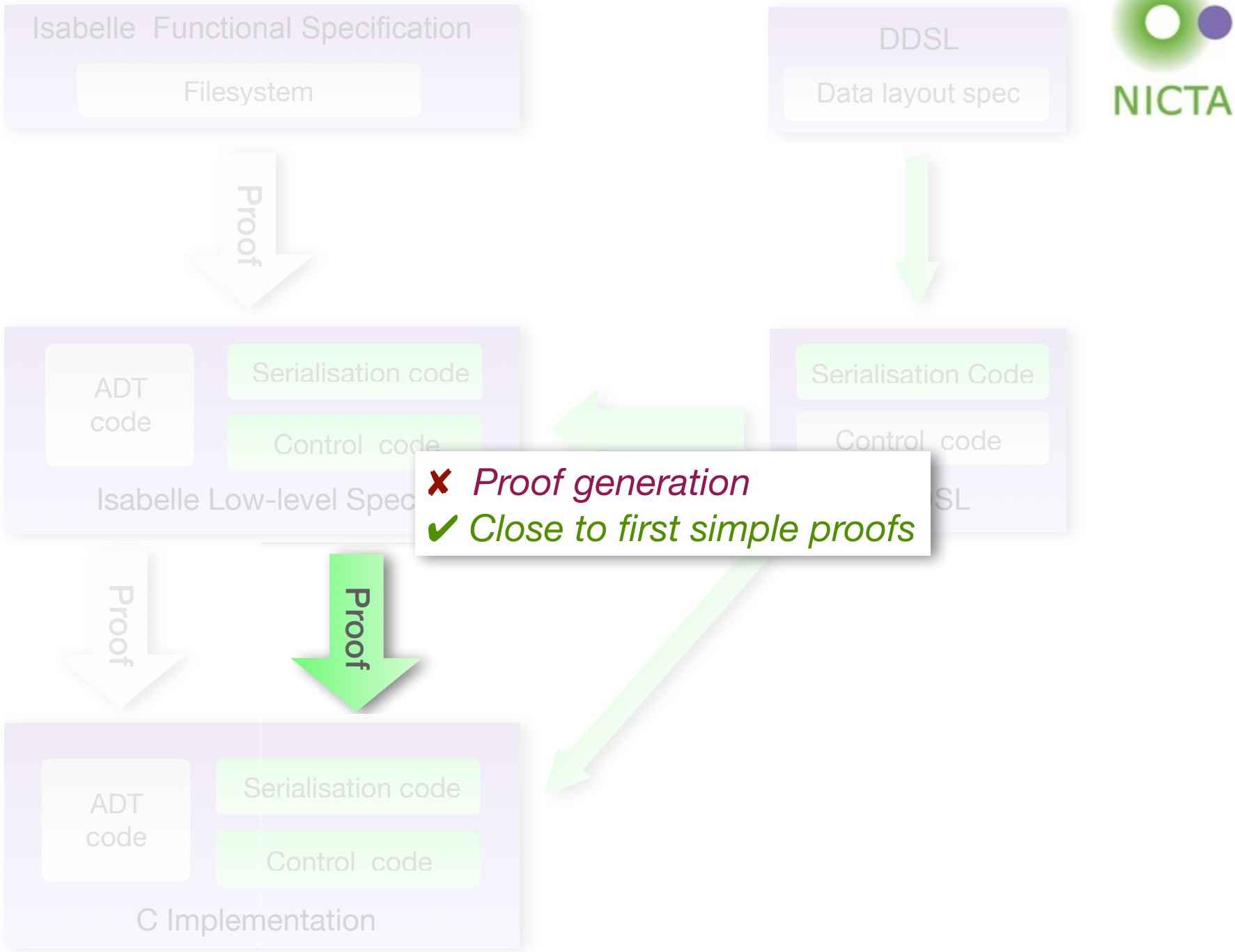

Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
Proof
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
✔ Low-level Isabelle spec of BilbyFs
ADT code
Serialisation code
Control code
C Implementation


Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
✘ CDSL to Isabelle code generation not implemented yet
✔ Should be fairly straight forward
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation


Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
NICTA
Proof
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation
✔ CDSL to C code generation mostly implemented


Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
Proof
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation
✔ Complete manual reference implementation of BilbyFs


Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
Proof
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
CDSL
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation
✔ Complete manual reference implementation of BilbyFs


Isabelle Functional Specification
Filesystem
DDSL
Data layout spec
Proof
ADT code
Serialisation code
Control code
Isabelle Low-level Spec
Serialisation Code
Control code
SL
✘ Proof generation
✔ Close to first simple proofs
Proof
Proof
ADT code
Serialisation code
Control code
C Implementation

Design of the Data Description Language

NICTA

Existing DDLs

how to specify data structures and data layout

how to express constraints on the values of the data structures

what and how to check statically

Existing verification tools

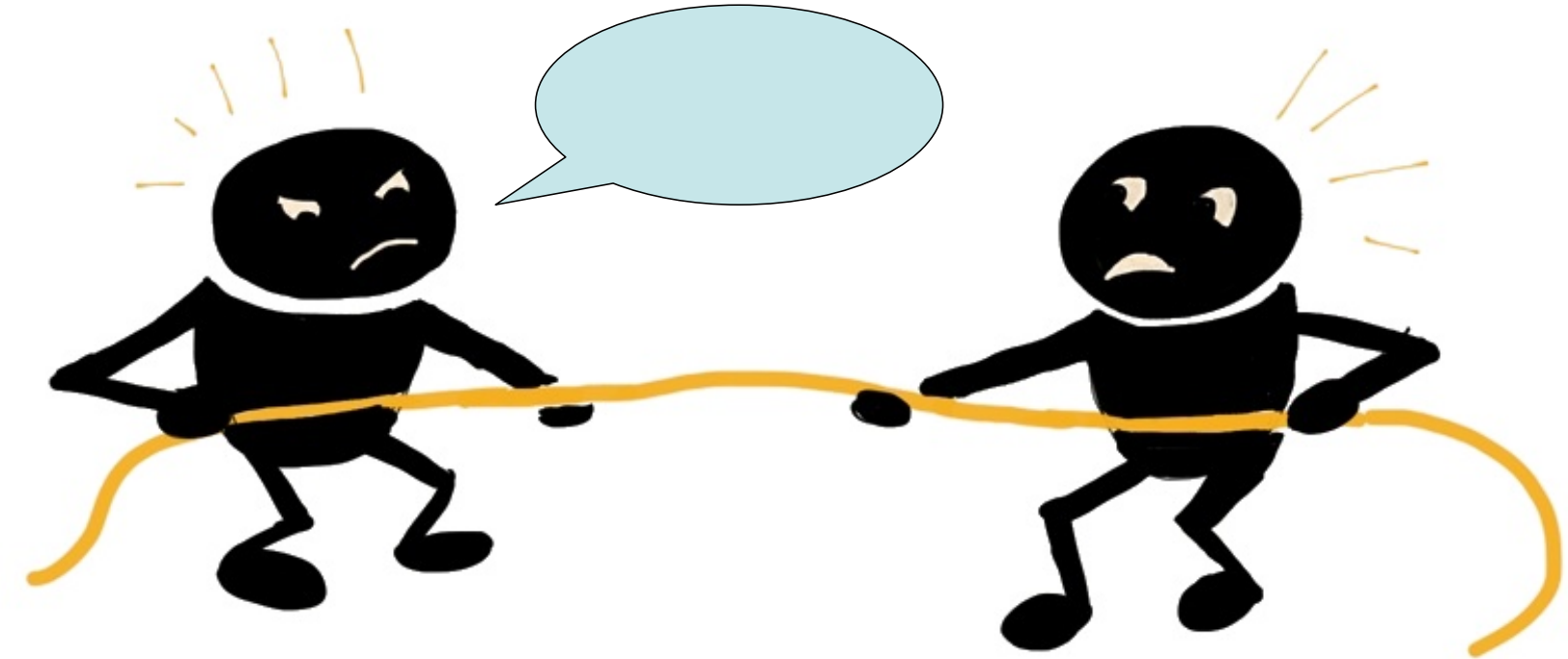
how to handle verification for tagged bitfield serialisation & de-serialisation

Properties of CDSL

NICTA

- Powerful enough to express file system operations
- Compile to efficient C Code
  - Destructive updates
  - No excessive copying
- Express verification properties like
  - Type safety
  - Memory safety
  - Termination
  - Totality

Design of the Data Description Language



Systems:
- -efficiency
- -destructive updates
- -expressiveness
- -concise

Verification:
- -controlled side effects
- -memory & type safety
- -termination
- -express assertions

Design of the CDSL

NICTA

– Linear type system allows interpretation via value as well as update semantics

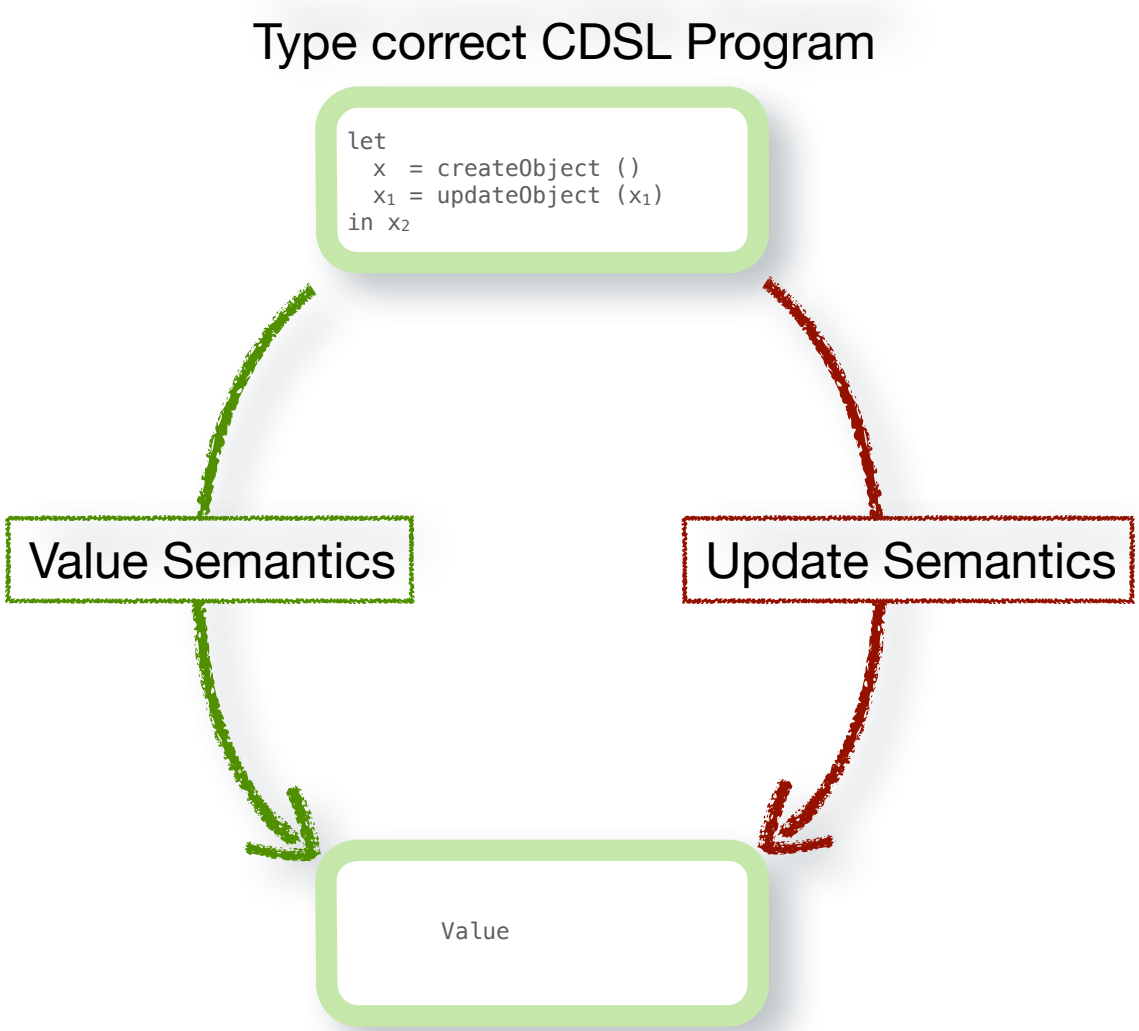
```
let
  x  = createObject ()
  x₁ = updateObject (x₁)
in x₂
```

```
let
  x = createObject ()
  x = updateObject (x)
in x
```

```
let
  x  = createObject ()
  x₁ = updateObject (x)
  _  = freeObject (x₂)
in x₂
```

```
let
  x  = createObject ()
  x₁ = updateObject (x)
in 5
```

Value and Update Semantics of CDSL



Design of the Control DSL

NICTA

- Huge design space, not much related work
  - rapid prototyping essential

- Strong type system to express static properties
  - linear type system to prevent aliasing
  - error handling enforced by type system
- Value and update semantics
  - every type correct program can be interpreted as

DDSL example

Thank You