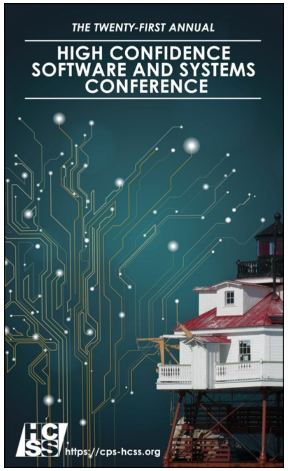


Compositional verification of modular C programs using VST and VSU

Lennart Beringer

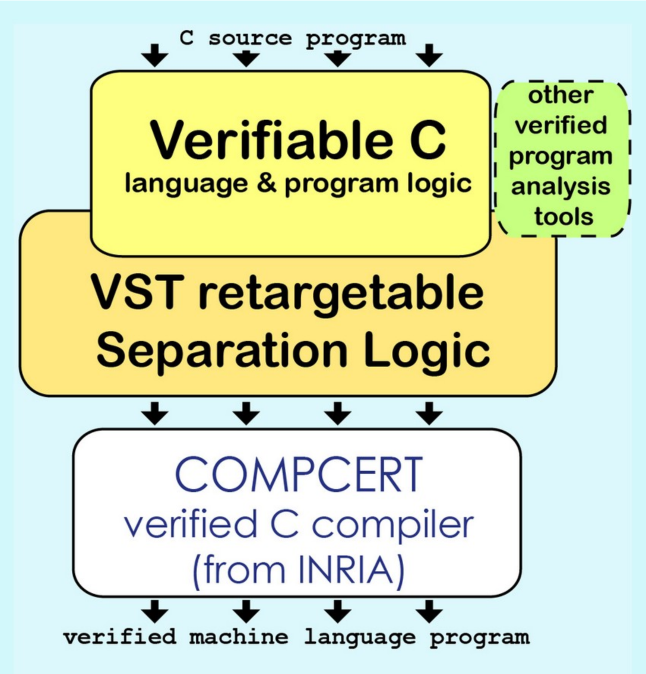
21st High Confidence Software and Systems Conference (HCSS'21)

May 3-6 2021

Awards 1005849 *Verified High Performance Data Structure Implementations* (Beringer) and 1521602 *Expedition in Computing: The Science of Deep Specification* (Appel).

Verified
Software
Toolchain

Verified Software Toolchain (VST)



C source program

Verifiable C
language & program logic

other verified program analysis tools

VST retargetable Separation Logic

COMPCERT
verified C compiler
(from INRIA)

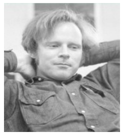
verified machine language program

Practical and **foundational**

verification for **C** using

separation logic

Realizing the vision of

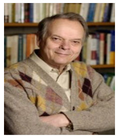


Floyd

Hoare

using the insights of

O’Hearn



Reynolds

based on foundations by

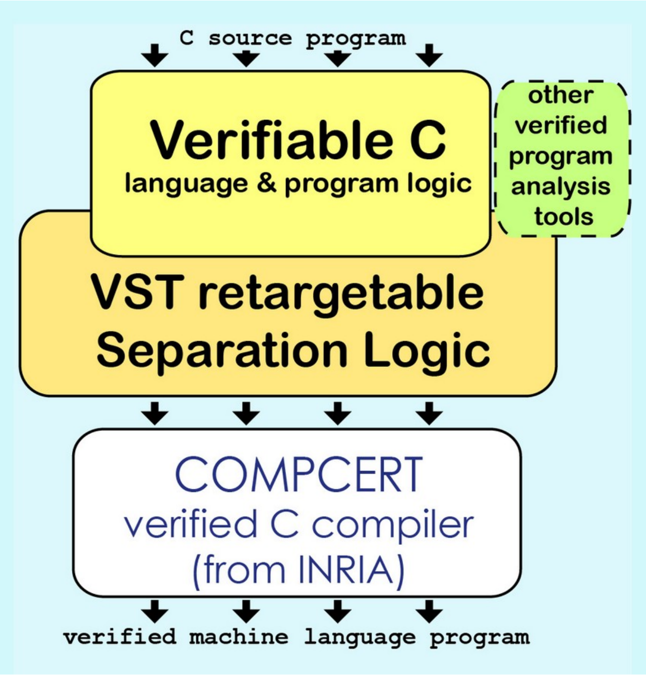
Leroy

Verified Software Toolchain (VST)

Stand-alone: crypto primitives, garbage collector, hashtables, malloc/free, mailbox communication, FEC, ...

HACMS

Connected: webserver, interaction trees, socket interface (CertiKOS)...



C source program

Verifiable C
language & program logic

other verified program analysis tools

VST retargetable Separation Logic

COMPCERT
verified C compiler
(from INRIA)

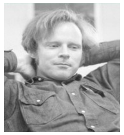
verified machine language program

Practical and **foundational**

verification for **C** using

separation logic

Realizing the vision of

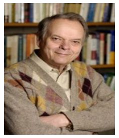


Floyd

Hoare

using the insights of

O'Hearn



Reynolds

based on foundations by

Leroy

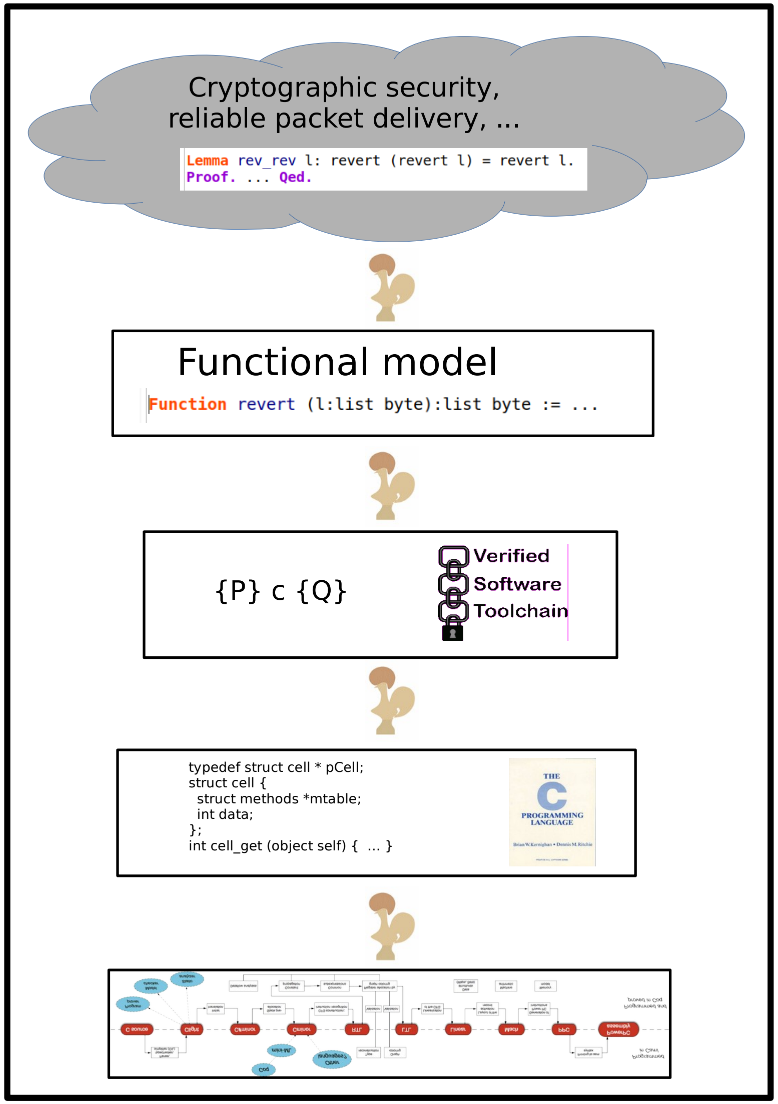
Hierarchy of formal systems

Integrate model-level-reasoning, code-level specification, verification, and compilation in a single formal environment

Benefit #1: no gaps at tool / model boundaries

Benefit #2 (“connect up): crypto,
model of network interaction

Benefit #3 (“connect down”):
OS interface (eg socket), HW



Cryptographic security,
reliable packet delivery, ...

```
Lemma rev_rev l: revert (revert l) = revert l.
Proof. ... Qed.
```

Functional model

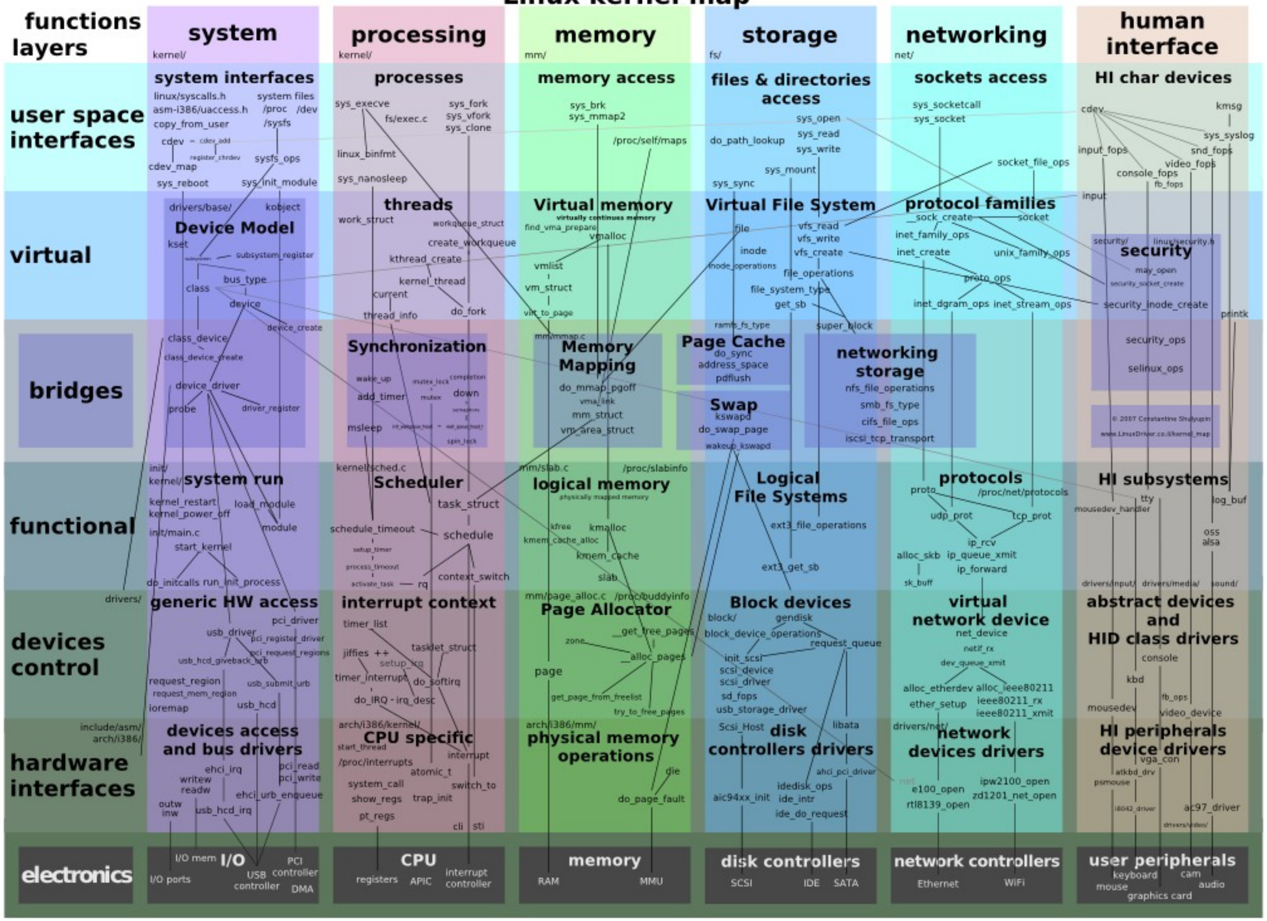
```
Function revert (l:list byte):list byte := ...
```

{P} c {Q}

```
typedef struct cell * pCell;
struct cell {
  struct methods *mtable;
  int data;
};
int cell_get (object self) {  ... }
```

Software is complex . . .

Linux kernel map



+ libraries, middleware, applications,...

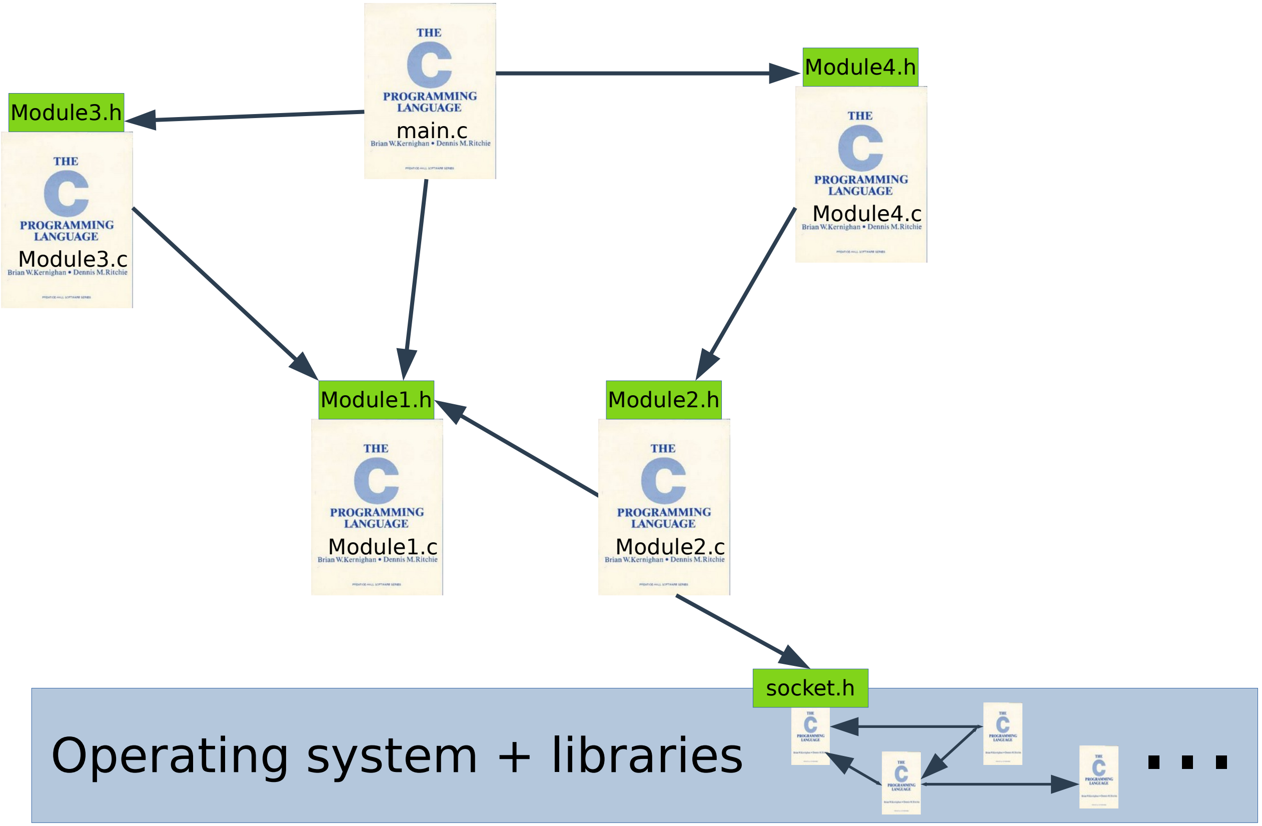
Well-organized software is modular

Mitchell

Plotkin

Burstall

Goguen



main.c

Module3.h

Module3.c

Module4.h

Module4.c

Module1.h

Module1.c

Module2.h

Module2.c

socket.h

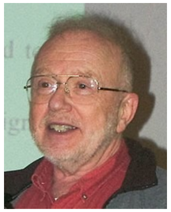
Operating system + libraries

. . .

Modular software admits **modular** specification and verification.

Liskov

Meyer



Parnas

(and many more...)

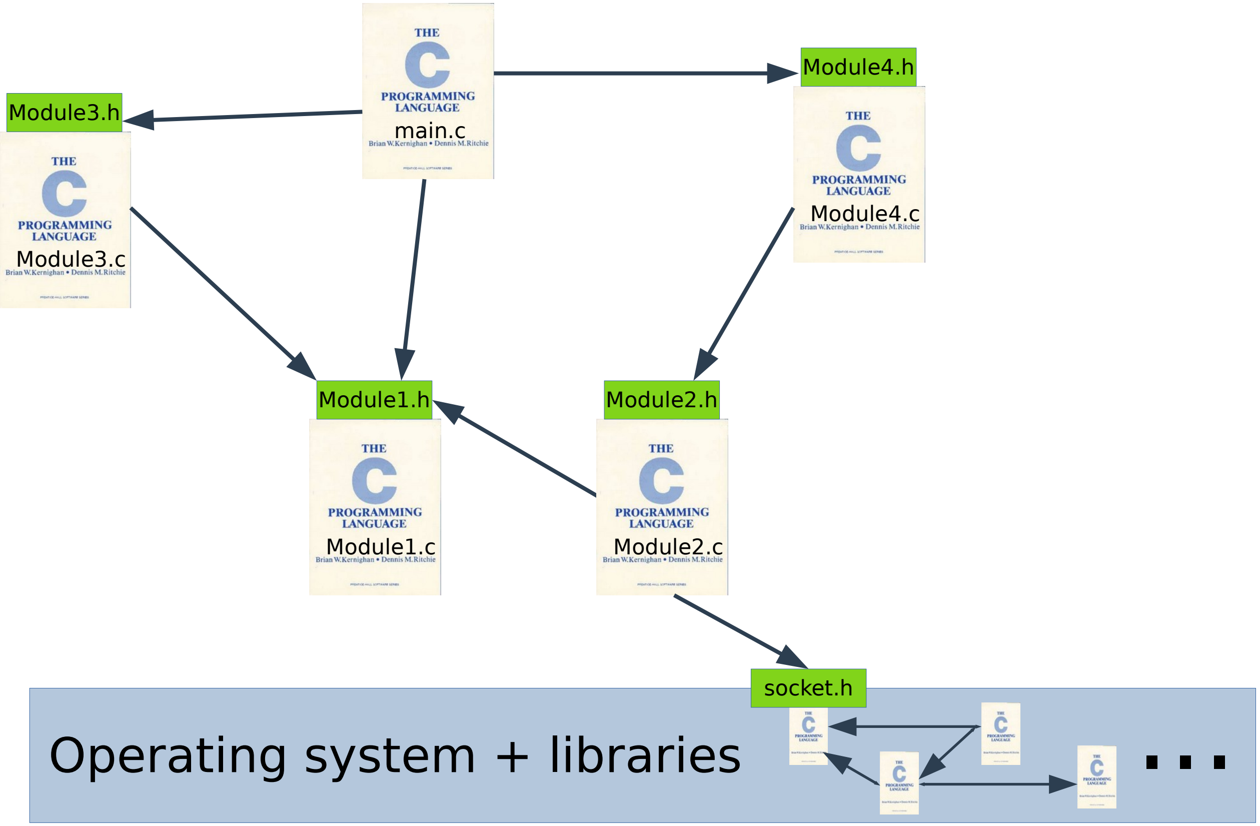
Well-organized software is modular

Mitchell

Plotkin

Burstall

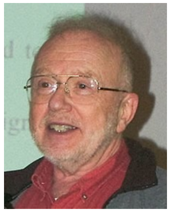
Goguen



Operating system + libraries

Liskov

Meyer


Parnas

(and
many
more...)

Modular software admits **modular** specification and verification.

Even in C ?

From VST to VSU

Theory and implementation of Verified Software Units (VSUs):

- **VST-verified compilation units for CompCert Clight**
- **composable at API-level specification interfaces**
  - compatible with syntactic composition of Clight AST's
  - provably sound w.r.t. VST's whole-program guarantee

From VST to VSU

Theory and implementation of Verified Software Units (VSUs):

- **VST-verified compilation units for CompCert Clight**
- **composable at API-level specification interfaces**
  - compatible with syntactic composition of Clight AST's ("linking")
  - provably sound w.r.t. VST's whole-program guarantee

Key idea: realize concepts from type theory/functional programming in imperative setting of C and exploit Coq's meta-logic

- **subtyping → function specification subsumption (adaptation of specifications at module boundaries)**
- **intersection types → intersection specifications (permit multiple specs, at different abstraction levels)**
- **existential (type) abstraction → information-hiding representation predicates**
- **parametricity → uniformity of specifications, strictest possible control over information leakage**

Case studies with dataless programming (ADTs, simple objects).

Example: stack ADT

Stack.h

```
// Include statements ...
// Shared constants, macros ...

struct stack;

struct stack *newstack(void);

void push (struct stack *p, int i);

int pop (struct stack *p);
```

shared data type declarations
- heap- or stack-allocated
- representation-hiding

function prototypes

Stack.c

```
// Include statements …
// Private constants, macros …
// Private data structures (“list”) and auxiliary functions …

struct stack { struct list *top; };

struct stack *newstack(void) { … }

void push (struct stack *p, int i) { … }

int pop (struct stack *p) { … }
```

data type
definition
(representation)

function definitions
(implementation)

Example: stack ADT

Stack.h

```
// Include statements ...
// Shared constants, macros ...

struct stack;

struct stack *newstack(void);

void push (struct stack *p, int i);

int pop (struct stack *p);
```

shared data type declarations
- heap- or stack-allocated
- representation-hiding

function prototypes

Stack.c

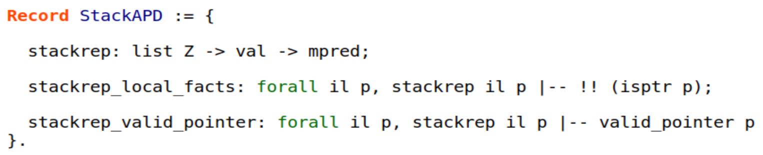
```
// Include statements ...
// Private constants, macros ...
// Private data structures and auxiliary functions ...

struct stack { struct list *top; };

struct stack *newstack(void) { … }

void push (struct stack *p, int i) { … }

int pop (struct stack *p) { … }
```

data type definition (representation)

function definitions (implementation)

Abstract predicate declarations (APD)

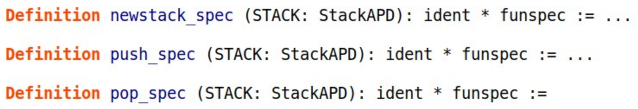
Existentially quantified representation-independent abstract predicates + axioms



```
Record StackAPD := {
  stackrep: list Z -> val -> mpred;
  stackrep_local_facts: forall il p, stackrep il p |-- !! (isptr p);
  stackrep_valid_pointer: forall il p, stackrep il p |-- valid_pointer p
}.
```

Abstract specification interface (ASI)

Representation-independent VST **specifications** of API-exposed functions, **parametric** in APDs



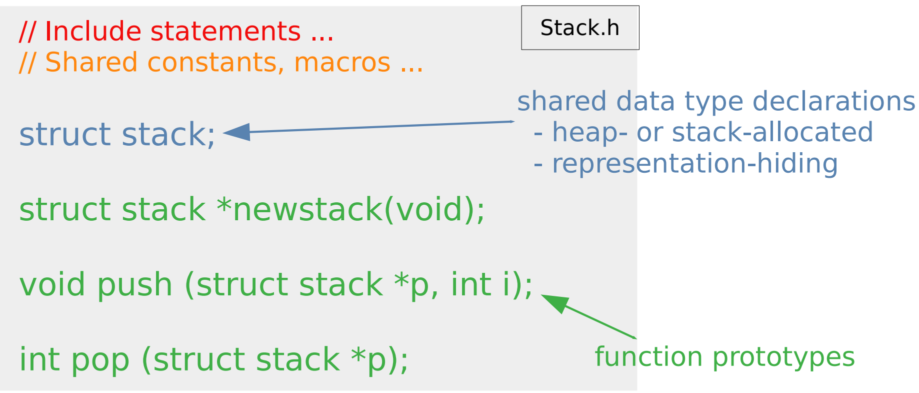
```
Definition newstack_spec (STACK: StackAPD): ident * funspec := ...
Definition push_spec (STACK: StackAPD): ident * funspec := ...
Definition pop_spec (STACK: StackAPD): ident * funspec :=
```

Verified software unit (VSU)

Representation-dependent substantiation of an ASI

- **parametric** in APDs of predicates provided by other modules
- provides **axiom-validating instantiations** of APDs for predicates provided by this module (using concrete predicates)
- VST proofs of ASI-exported **and private** functions

Example: stack ADT

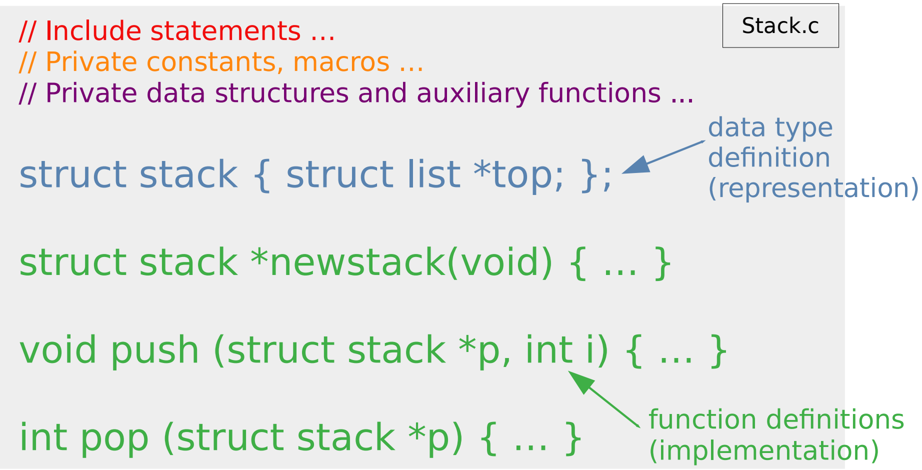


Stack.h

```
// Include statements ...
// Shared constants, macros ...

struct stack;

struct stack *newstack(void);

void push (struct stack *p, int i);

int pop (struct stack *p);
```

shared data type declarations
- heap- or stack-allocated
- representation-hiding

function prototypes



Stack.c

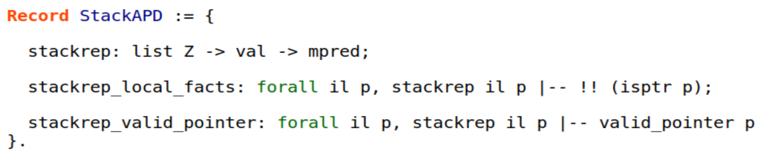
```
// Include statements ...
// Private constants, macros ...
// Private data structures and auxiliary functions ...

struct stack { struct list *top; };

struct stack *newstack(void) { … }

void push (struct stack *p, int i) { … }

int pop (struct stack *p) { … }
```

data type definition (representation)

function definitions (implementation)

Abstract predicate declarations (APD)

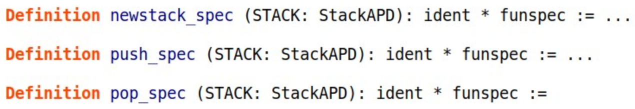
Existentially quantified representation-independent abstract predicates + axioms



```
Record StackAPD := {
  stackrep: list Z -> val -> mpred;
  stackrep_local_facts: forall il p, stackrep il p |-- !! (isptr p);
  stackrep_valid_pointer: forall il p, stackrep il p |-- valid_pointer p
}.
```

Abstract specification interface (ASI)

Representation-independent VST **specifications** of API-exposed functions, **parametric** in APDs



```
Definition newstack_spec (STACK: StackAPD): ident * funspec := ...
Definition push_spec (STACK: StackAPD): ident * funspec := ...
Definition pop_spec (STACK: StackAPD): ident * funspec :=
```

Verified software unit (VSU)

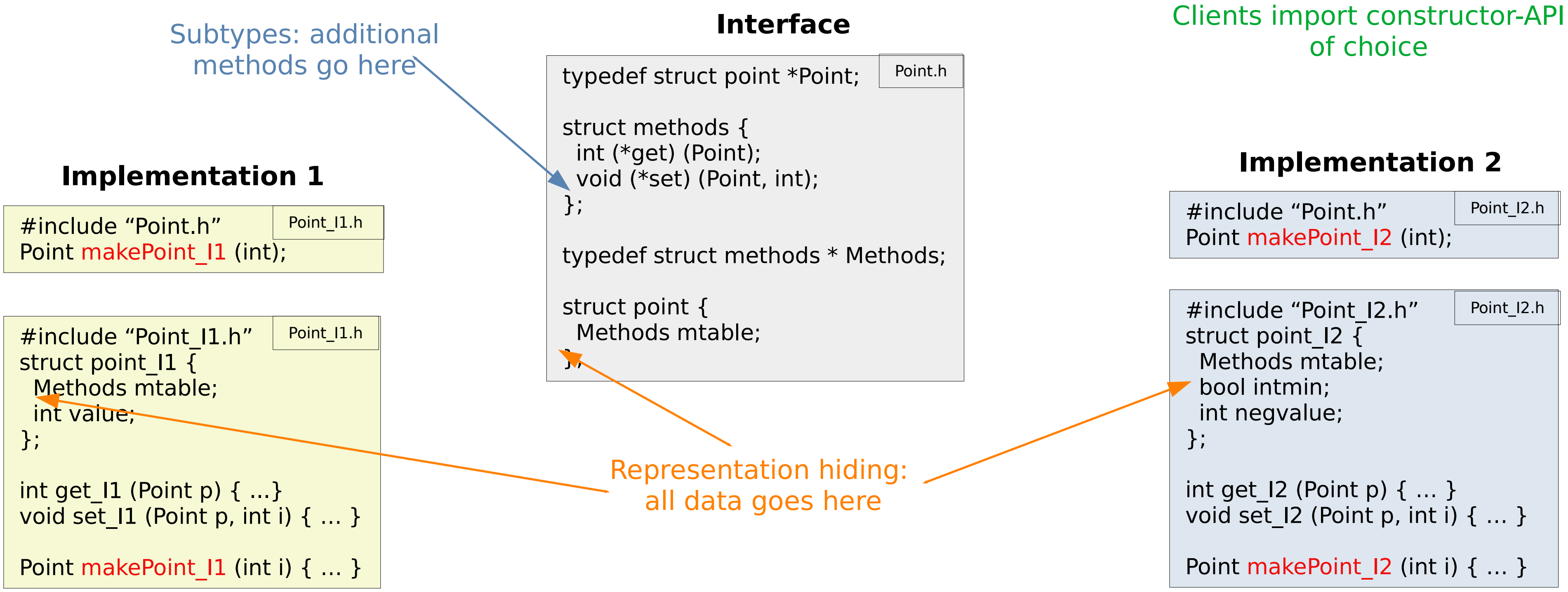
Representation-dependent substantiation of an ASI

- **parametric** in APDs of predicates provided by other modules
- provides **axiom-validating instantiations** of APDs for predicates provided by this module (using concrete predicates)
- VST proofs of ASI-exported **and private** functions

VSU_link retains abstraction and ensures whole-program soundness wrt VST

Simple objects in C

Dynamic dispatch using function pointers and struct extensions



Subtypes: additional methods go here

Interface

Point.h
```
typedef struct point *Point;

struct methods {
  int (*get) (Point);
  void (*set) (Point, int);
};

typedef struct methods * Methods;

struct point {
  Methods mtable;
};
```

Clients import constructor-API of choice

Implementation 1

Point_I1.h
```
#include “Point.h”
Point makePoint_I1 (int);
```

Point_I1.h
```
#include “Point_I1.h”
struct point_I1 {
  Methods mtable;
  int value;
};

int get_I1 (Point p) { …}
void set_I1 (Point p, int i) { … }

Point makePoint_I1 (int i) { … }
```

Implementation 2

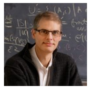
Point_I2.h
```
#include “Point.h”
Point makePoint_I2 (int);
```

Point_I2.h
```
#include “Point_I2.h”
struct point_I2 {
  Methods mtable;
  bool intmin;
  int negvalue;
};

int get_I2 (Point p) { … }
void set_I2 (Point p, int i) { … }

Point makePoint_I2 (int i) { … }
```

Representation hiding: all data goes here

One API, many **coexisting** implementations, distinguished only by API-exposed **constructors**.

Specification and verification based on **semantic objects** and **positive subtyping**.



Conclusion

- Enhanced capabilities for modular foundational verification of C code using VST
- Enforce SW engineering principles using SL + type theory + proof assistant
- Performance improvements due to lightweight representation of Clight programs
- Current & future work:
  - additional reasoning principles for objects, CertiCoq-FFI
  - applications with concurrency (fine-grained locking, lockfree, …)
  - DARPA OPS-5G: SDN control plane verification and interaction with P4
- Paper at ESOP'21, stack example in next release of Software Foundations, Volume 5

Visit https://github.com/coq/platform for installer

or https://vst.cs.princeton.edu to access git master.

Questions? See VST mailing list, stackoverflow

Verified
Software
Toolchain