BedRock Systems Inc

# Towards an Axiomatic Basis for C++

Gregory Malecha, Abhishek Anand, Gordon Stewart

BedRock Systems

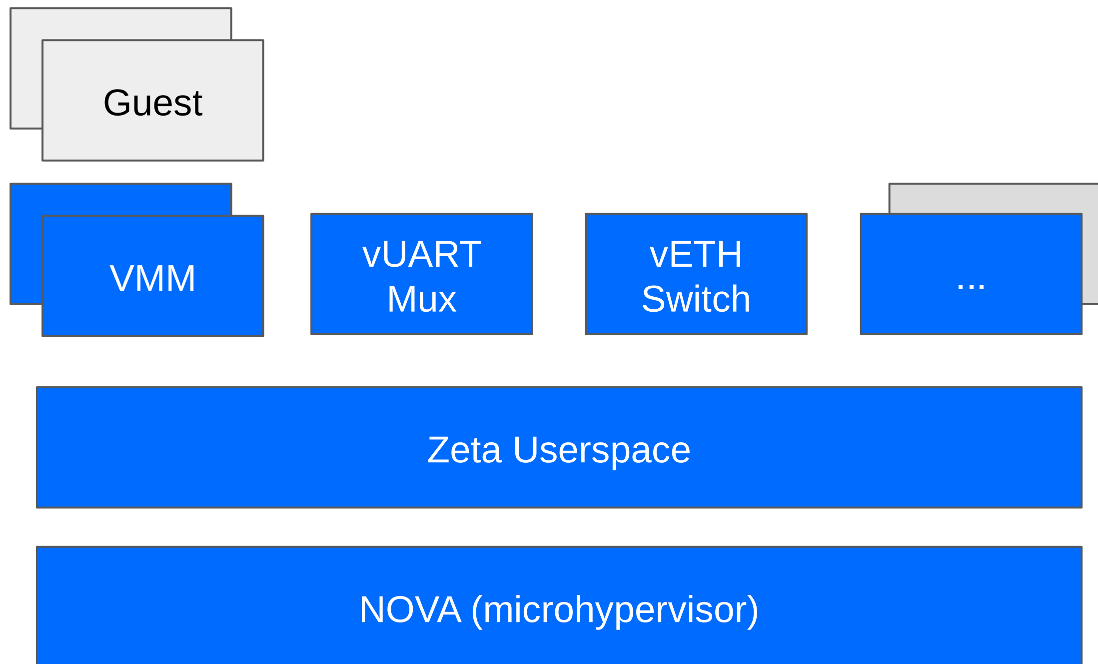
# BedRock Systems

Formally verified,
deep specifications.

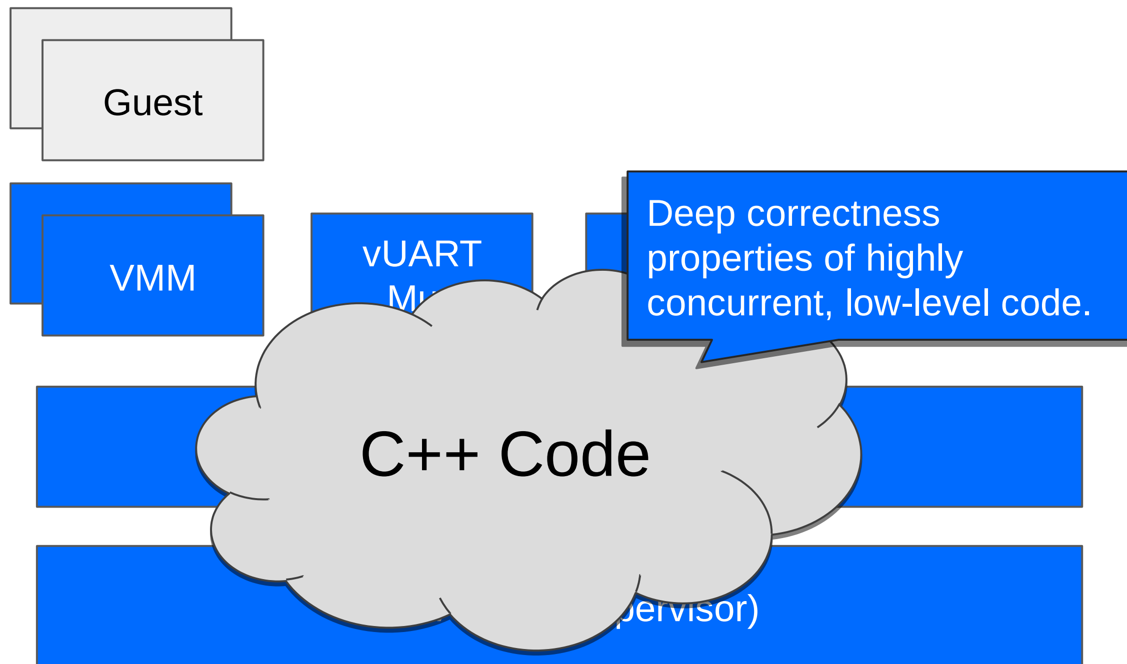
***Unbreakable Foundation for the Software Defined World***

Enable ***everyone*** to *write and share* verified code!

# Verification target



# Verification target

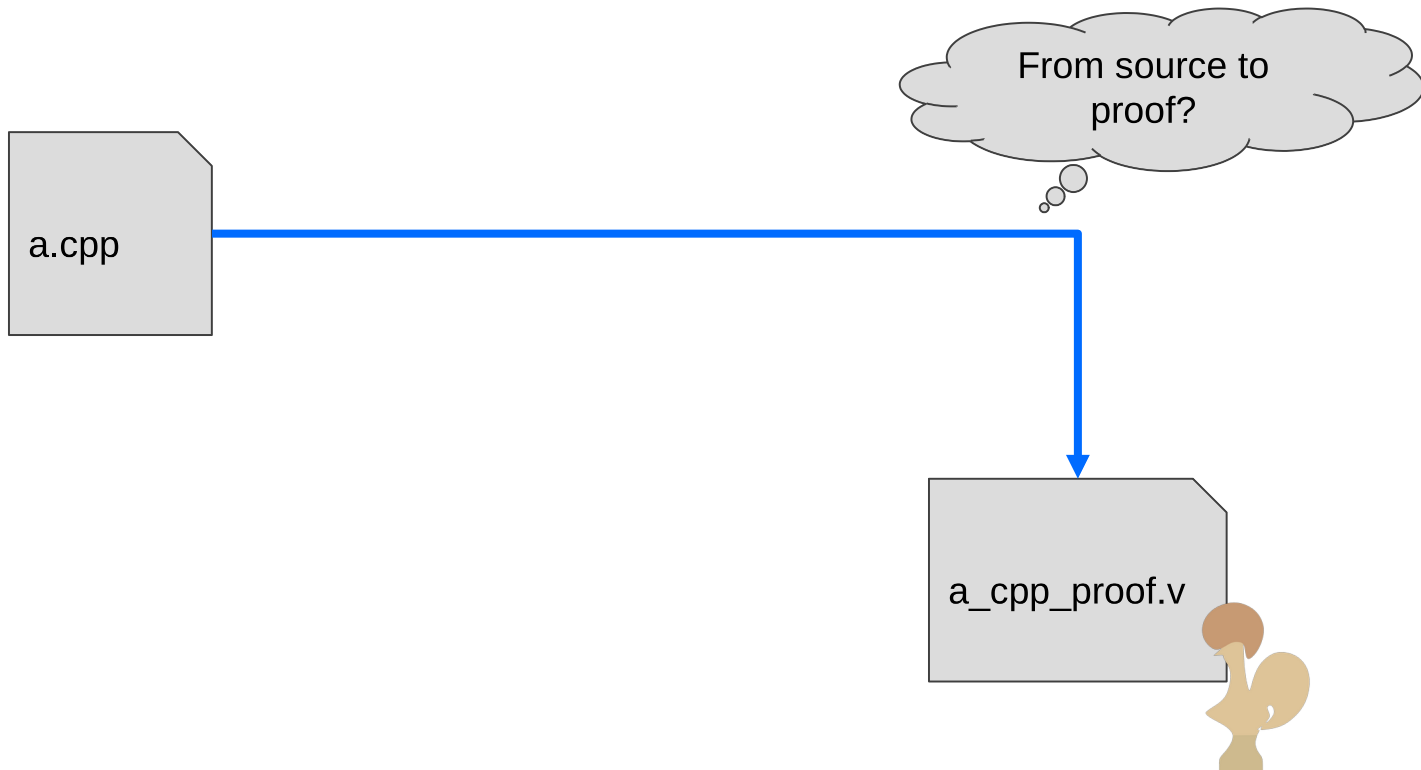





BedRock Systems Inc

# Working with C++

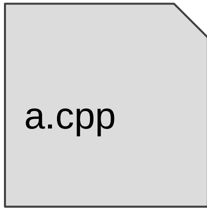
# The verification toolchain

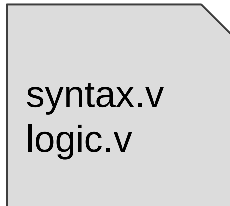


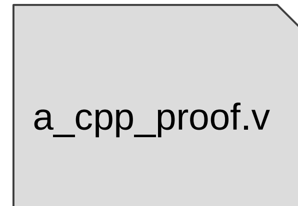



BedRock Systems Inc

# The verification toolchain



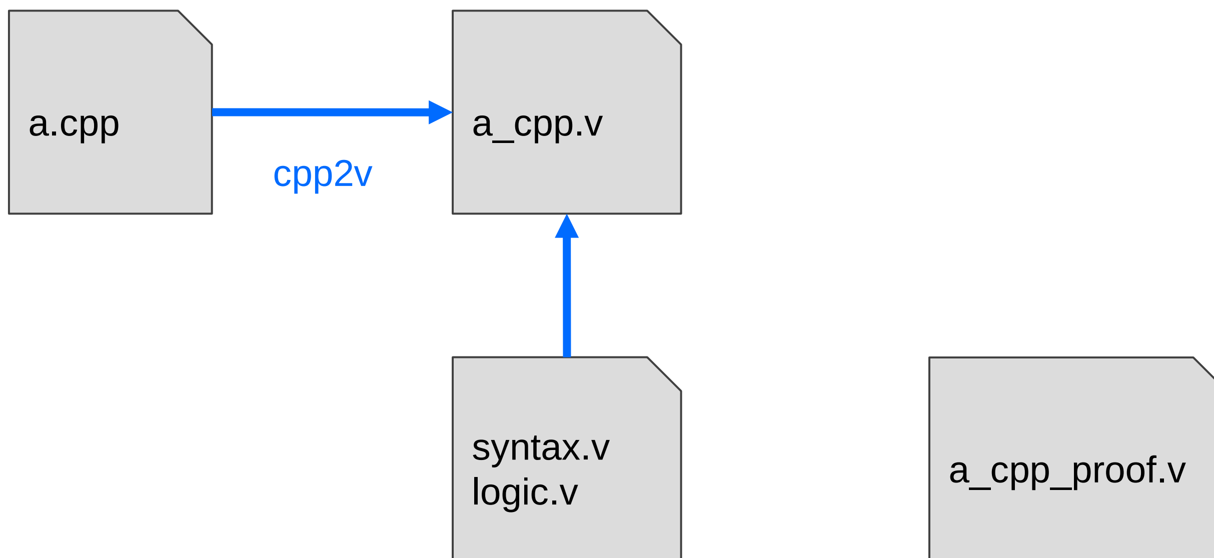



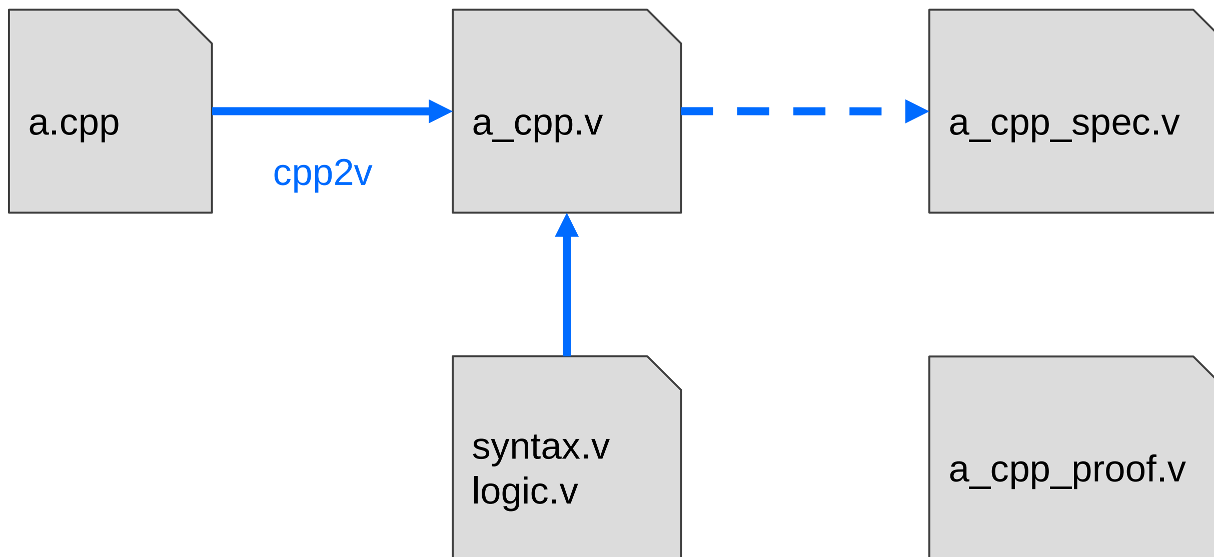



BedRock Systems Inc

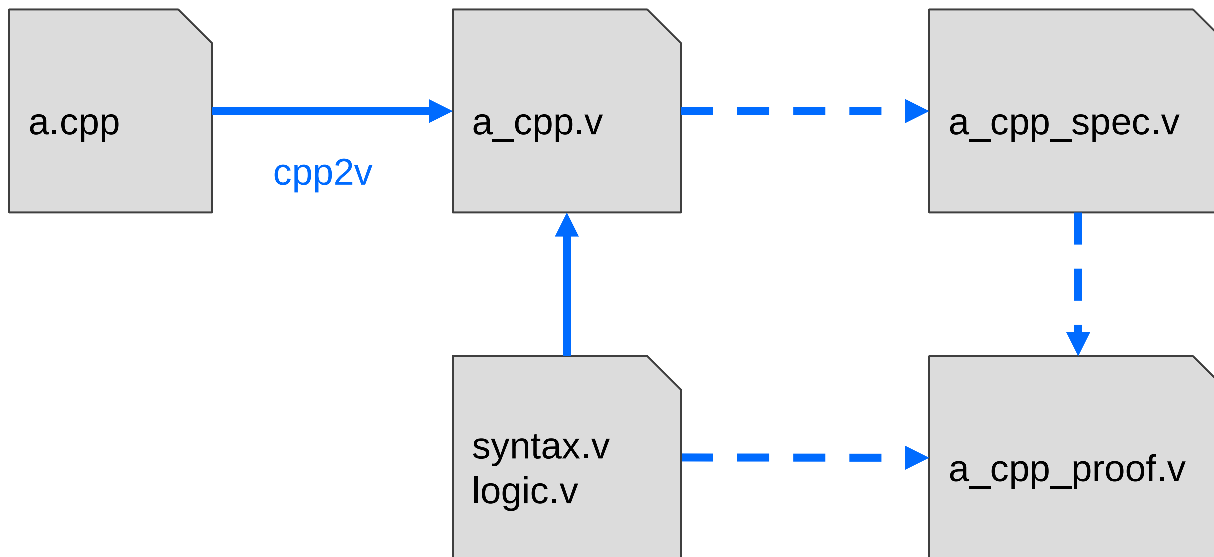
# The verification toolchain



# The verification toolchain



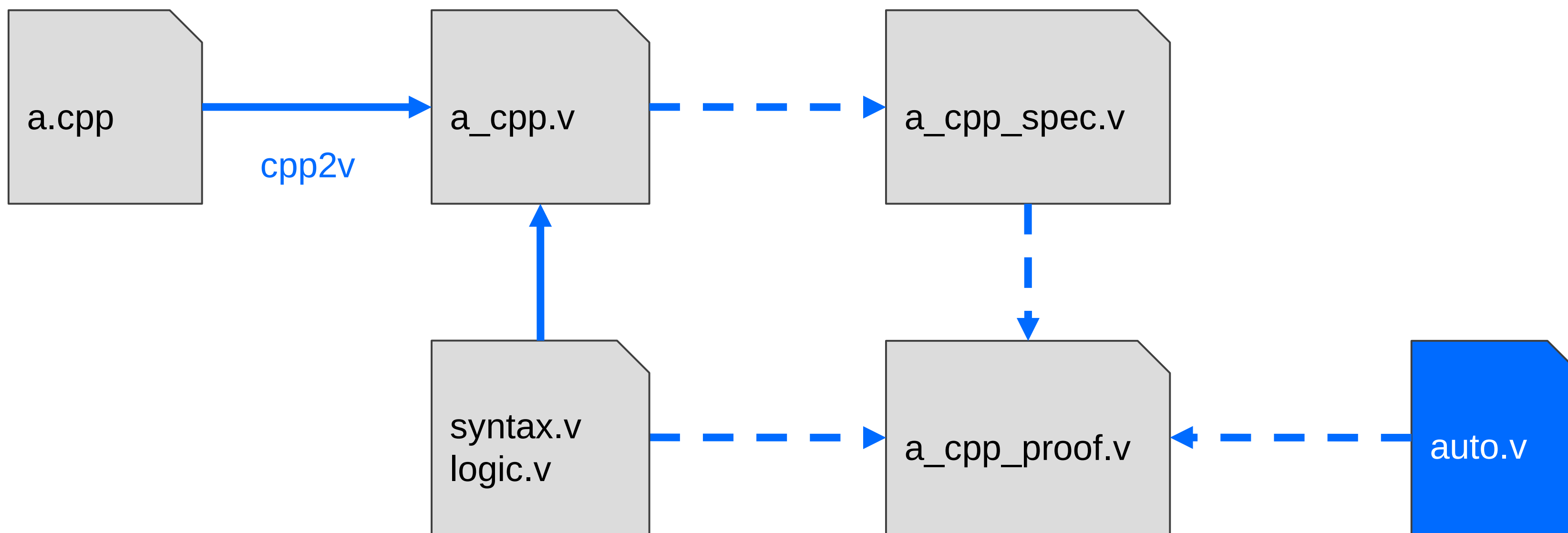
# The verification toolchain






BedRock Systems Inc

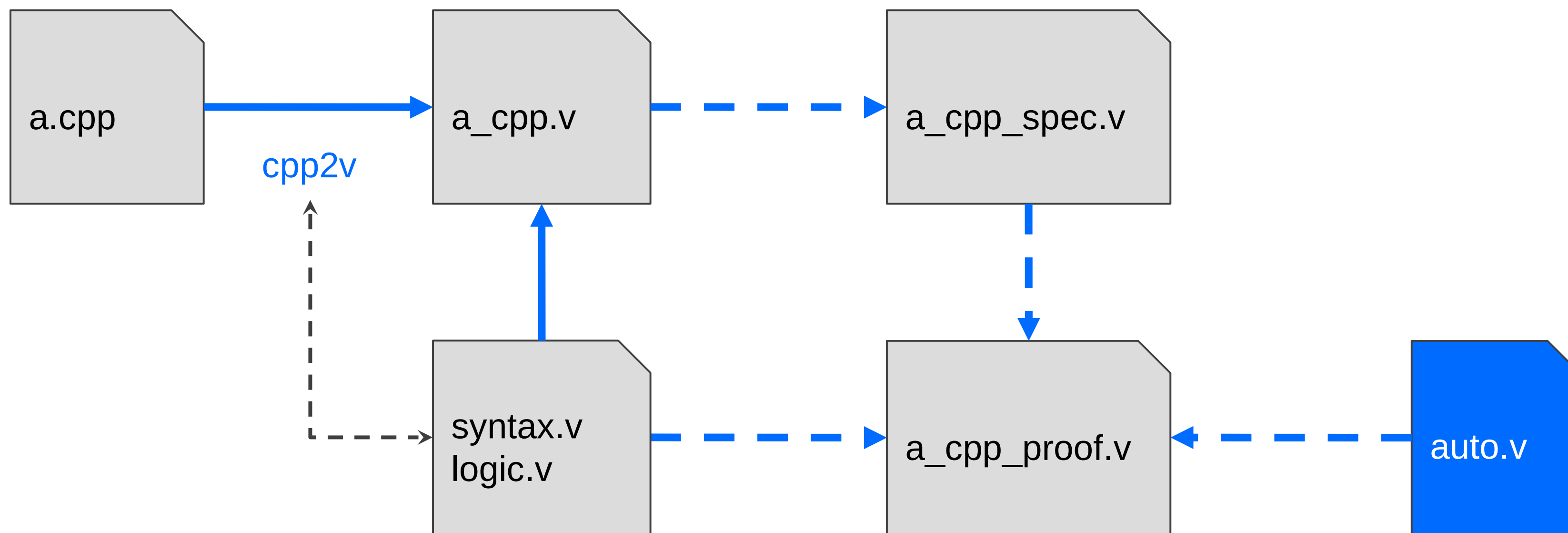
# The verification toolchain






BedRock Systems Inc

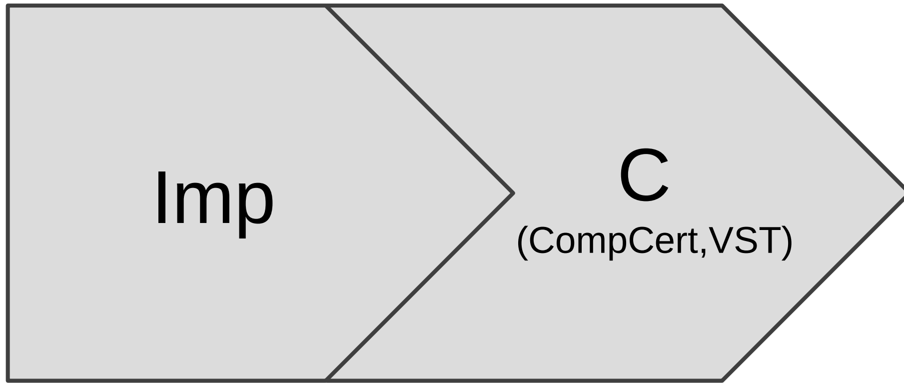
# The verification toolchain





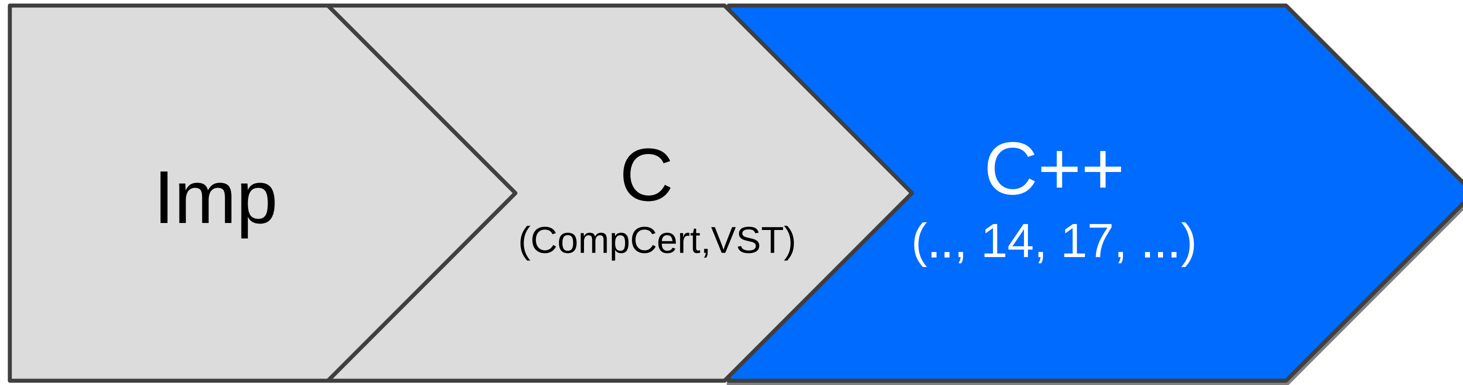

BEDROCK
Systems Inc

# Building on previous work




**Iris separation logic library**

# Building on previous work




Iris separation
logic library

# Features of C++

### Surface Complexities

- Parsing
- Type checking
- Overload resolution
- Syntactic sugar

### Semantic Challenges

- Value categories
- Side-effects
- Modularity

### Classes + Objects

- Constructors
- Destructors
- Inheritance

# Features of C++

### Surface Complexities

- Parsing
- Type checking
- Overload resolution
- Syntactic sugar

Hooking into existing tooling

### Semantic Challenges

- Value categories
- Side-effects
- Modularity

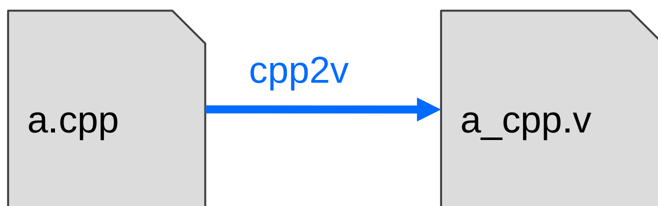
### Classes + Objects

- Constructors
- Destructors
- Inheritance



BedRock Systems Inc

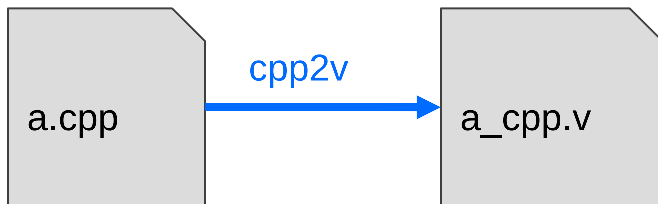
# cpp2v




**Uses** clang **to build C++ ASTs from source files.**

- First-order AST,
- embedded types

# cpp2v




**Uses** clang **to build C++ ASTs from source files.**

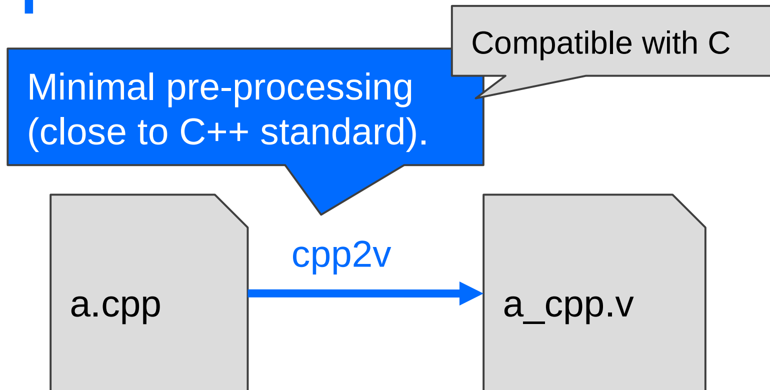
- First-order AST,
- embedded types

cpp2v -o a_cpp.v src/a.cpp -- --target=aarch64-none-elf -std=gnu++17 -O2 -fno-exceptions -fno-rtti -fno-threadsafe-statics -fno-builtin -I./include -I./include/aarch64

Standard `clang` compiler options.

Also runnable as a clang plugin.

# cpp2v




**Uses** clang **to build C++ ASTs from source files.**

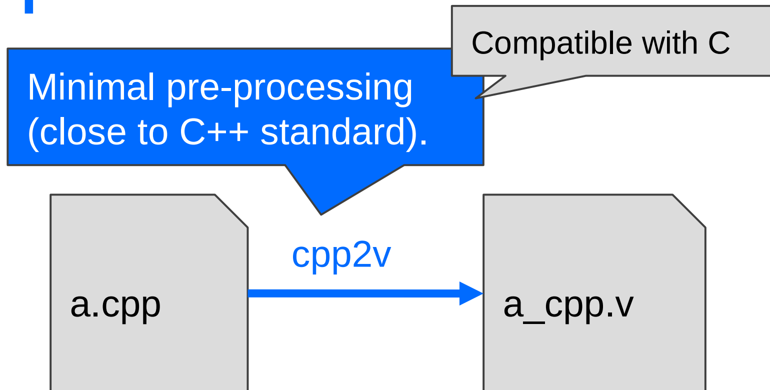
- First-order AST,
- embedded types

cpp2v -o a_cpp.v src/a.cpp -- --target=aarch64-none-elf -std=gnu++17 -O2 -fno-exceptions -fno-rtti -fno-threadsafe-statics -fno-builtin -I./include -I./include/aarch64

Standard `clang` compiler options.

Also runnable as a clang plugin.

# cpp2v




**Uses** clang **to build C++ ASTs from source files.**

- First-order AST,
- embedded types

**Include extra information to ease consumption:**

- value categories,
- types,
- implicit initializers,
- overload resolution,
- some desugaring,
- etc.

```
cpp2v -o a_cpp.v src/a.cpp -- --target=aarch64-none-elf -std=gnu++17 -O2 -fno-exceptions -fno-rtti -fno-threadsafe-statics -fno-builtin -I./include -I./include/aarch64
```

Standard `clang` compiler options.

Also runnable as a clang plugin.

# Features of C++

Surface Complexities

- Parsing
- Type checking
- Overload resolution
- Syntactic sugar

Hooking into existing tooling

Semantic Challenges

- Value categories
- Side-effects
- Modularity

Weakest precondition semantics in Iris

Classes + Objects

- Constructors
- Destructors
- Inheritance

# The program logic for C++

These are values, e.g. integers

```
(* semantics of an expression interpreted as a prvalue *)
Parameter wp_prval : ∀ {σ:genv},
    coPset → thread_info → region →
    Expr →
    (val → FreeTemps → epred) →
    mpred.
```

And for other value categories & language constructs: wp_lval, wp_xval

# The program logic for C++

```
(* semantics of an expression i            as a prvalue *)
Paramet                    ∀ {σ:genv},
    coPset → thread_info → region →
    Expr →
    (val → FreeTemps → epred) →
    mpred.
```

These are values, e.g. integers

Declarations

“Thread identifier”

Iris mask

Locals

Temporaries to destroy

And for other value categories & language constructs: wp_lval, wp_xval

# Variables & Regions

**All program state is represented uniformly as resources**

- Simple representation of stack-allocated structs
- More uniform representation predicates

```
(* variables are lvalues *)
Axiom wp_lval_lvar : ∀ ty x Q,
    Exists a, _local ρ x &~ a ** Q (Vptr a) emp
    |-- wp_lval M ti ρ (Evar (Lname x) ty) Q.
```

Location of x is a (persistent)

Mapping from names to location

```
(* left-to-right casts read locations *)
Axiom wp_prval_cast_l2r_l : ∀ ty e Q,
    wp_lval M ti ρ e (λ a free ⟹ Exists q v,
        (a |-> primR (erase_qualifiers ty) q v ** ⊤) ∧
        Q v free)
    |-- wp_prval M ti ρ (Ecast Cl2r (Lvalue, e) ty) Q.
```

*All* locations are accessed uniformly.

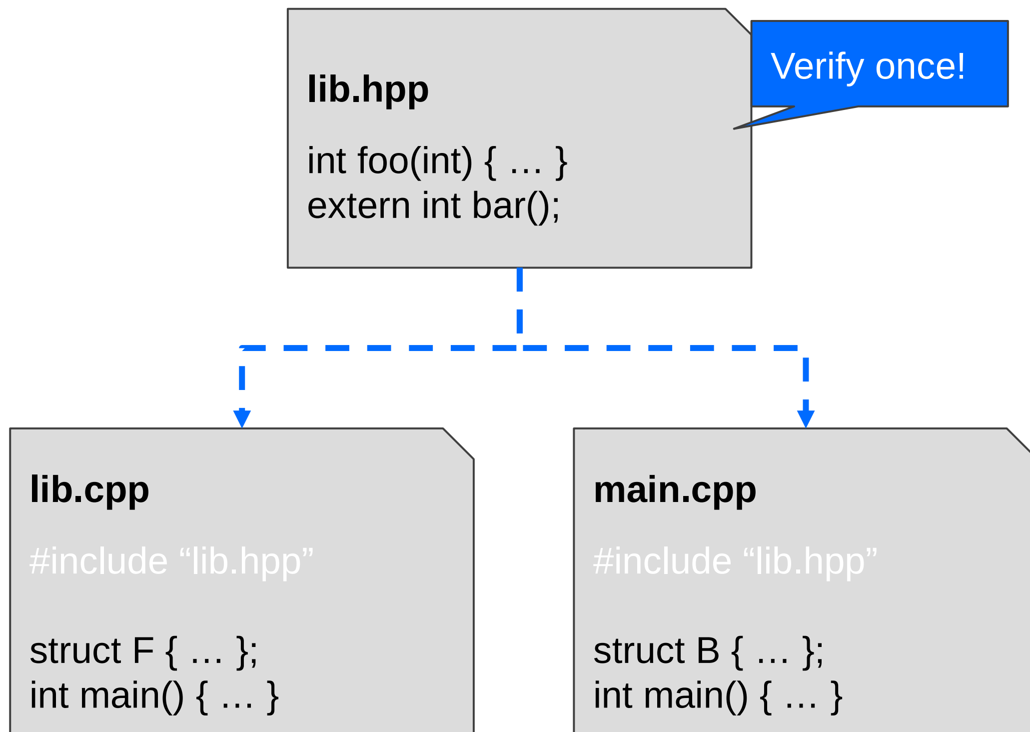


BEDROCK Systems Inc

# File-modular Verification

- **#include & macros**
  - Verification after macro expansion
  - C++ is moving away from macros towards language-based features, e.g. constexpr

- ***Lots* of code in header files.**



# File-modular Verification

- **#include & macros**
  - Verification after macro expansion
  - C++ is moving away from macros towards language-based features, e.g. constexpr
- ***Lots* of code in header files.**

lib.hpp

```
int foo(int) { … }
extern int bar();
```

*Preservation under compatible extension*

```
Axiom wp_prval_frame :
  ∀ σ1 σ2 M ti ρ e k1 k2,
    genv_leq σ1 σ2 →
    Forall v f, k1 v f -* k2 v f
    |-- @wp_prval σ1 M ti ρ e k1 -* @wp_prval σ2 M ti ρ e k2.
```

```
#include "lib.hpp"

struct F { … };
int main() { … }
```

```
#include "lib.hpp"

struct B { … };
int main() { … }
```



BedRock Systems Inc

# Features of C++

Surface Complexities

- Parsing
- Type checking
- Overload resolution
- Syntactic sugar

Hooking into existing tooling

Semantic Challenges

- Value categories
- Side-effects
- Modularity

Weakest precondition semantics in Iris

Classes + Objects

- Constructors
- Destructors
- Inheritance

Describe the object system in separation logic.

# Supporting Classes + Objects

## Classes are a pervasive addition in C++

- Constructors
- Destructors
- Member functions
- Virtual functions

**Fairly easy due to information in the AST, e.g. explicit cast nodes, etc.**



BedRock Systems Inc

# Supporting Classes + Objects

## Classes are a pervasive addition in C++

- Constructors
- Destructors
- Member functions
- Virtual functions

**Fairly easy due to information in the AST, e.g. explicit cast nodes, etc.**

## Object identity is intricate

- Track it using language-specific ghost state

```
(** [identity σ this mdc q p] state that [p] is a pointer to a (live)
    object of type [this] that is part of an object of type [mdc].
 *)
Parameter identity : ∀ {σ : genv}
    (this : globname) (most_derived : option globname),
    Qp → ptr → mpred.
```

# Supporting Classes + Objects

## Classes are a pervasive addition in C++

- Constructors
- Destructors
- Member functions
- Virtual functions

**Fairly easy due to information in the AST, e.g. explicit cast nodes, etc.**

## Object identity is intricate

- Track it using language-specific ghost state

```
(** [identity σ this mdc q p] state that [p] is a pointer to a (live)
    object of type [this] that is part of an object of type [mdc].
 *)
Parameter identity : ∀ {σ : genv}
    (this : globname) (most_derived : option globname),
    Qp → ptr → mpred.
```

**Still looking for a good abstraction for reasoning. (Do you have ideas?)**

# Features of C++

Surface Complexities

- Parsing
- Type checking
- Overload resolution
- Syntactic sugar

Hooking into existing tooling

Semantic Challenges

- Value categories
- Side-effects
- Modularity

Weakest semar

Classes + Objects

- Constructors
- Destructors
- Inheritance

e object system ation logic.

Unsupported Features

- Uninstantiated templates
- Lambda expressions
- virtual inheritance
- Exceptions
- Weak memory

# Verification for Everyone

# It helps!

Separation logic is central to this.

“Every engineer uses some form of “verification" in their head ..., formal verification simply helps putting that on paper precisely.”

~*Systems* Engineer

# It helps!

Separation logic is central to this.

"Every engineer uses some form of "verification" in their head ..., formal verification simply helps putting that on paper precisely."

~*Systems* Engineer

- **Teaching *everyone* to specify their code**
  - Very helpful to tie verification to a language they already know.
  - Systems engineers able to write first-order specifications.
  - Seems to be some cognitive benefit to classes.

# Summary

- **cpp2v is a tool for importing C++ code in Coq**
  - Built on top of the clang toolchain
- **Axiomatic semantics of (much of) C++**
  - Some interesting challenges in C++

Contributions, collaborations, and users welcome

**cpp2v**

**https://github.com/bedrocksystems/cpp2v**