Verifying C++ at Scale

Gregory Malecha

(gregory@bedrocksystems.com)

BedRock Systems, Inc

BedRock Systems Inc

Systems
Verifying ~~C++~~ at Scale

Gregory Malecha
(gregory@bedrocksystems.com)
BedRock Systems, Inc

Scaling to...

(Many dimensions of scale)

- Mainstream Languages
- Everyday developers
- Heterogeneous code-bases
- Boring problems
- Difficult problems

Scaling to...

(Many dimensions of scale)

- Mainstream Languages
- Everyday developers
- Heterogeneous code-bases
- Boring problems
- Difficult problems

BedRock
Systems Inc

Making Verification Mainstream

A "Better" Language

Convince mainstream developers to change their language of choice.

- Ada
- Rust
- ...

An Existing Language

Provide tools to verify software written in existing mainstream languages

- C (several projects)
- C++

From C to C++

Surface Complexities

- Parsing
- Type checking
- Overloading
- Syntactic sugar
- constexpr
- templates

Semantic Challenges

- Value categories
- Memory model
- Side-effects
- Modularity

Classes + Objects

- Constructors
- Destructors
- Inheritance
- Virtual dispatch

Some features of C++ are *overly* complicated.
➔ Under-approximate defined behavior.

Can support more behaviors over time.

Scaling to...

Many dimensions of scale

- Mainstream Languages
- Everyday developers
- Heterogeneous code-bases
- Boring problems
- Difficult problems

BedRock
Systems Inc

More than bug-free
(Confidence & Composition)

Implementation-level assurance

- Explainability
- Assurance

"Traditional" value-proposition of formal methods.

More than bug-free
(Confidence & Composition)

Implementation-level assurance

- Explainability
- Assurance

"Traditional" value-proposition of formal methods.

Formal methods as the **science of design**.

Design-level confidence

- Composition
- Abstraction
- Encapsulation

More than bug-free
(Confidence & Composition)

Implementation-level assurance

- Explainability
- Assurance

"Traditional" value-proposition of formal methods.

Formal methods as the **science of design**.

Design-level confidence

- Composition
- Abstraction
- Encapsulation

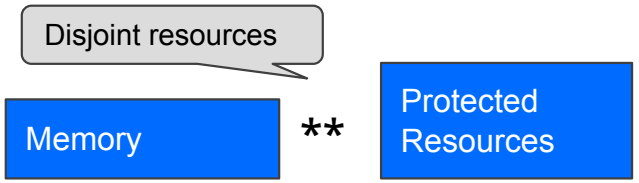
Formal methods must be more than "testing".

Separation Logic

Our ***language*** of formal methods

Separation is inherent in *our* understanding,
it should be the cornerstone of our reasoning…

What does it look like?

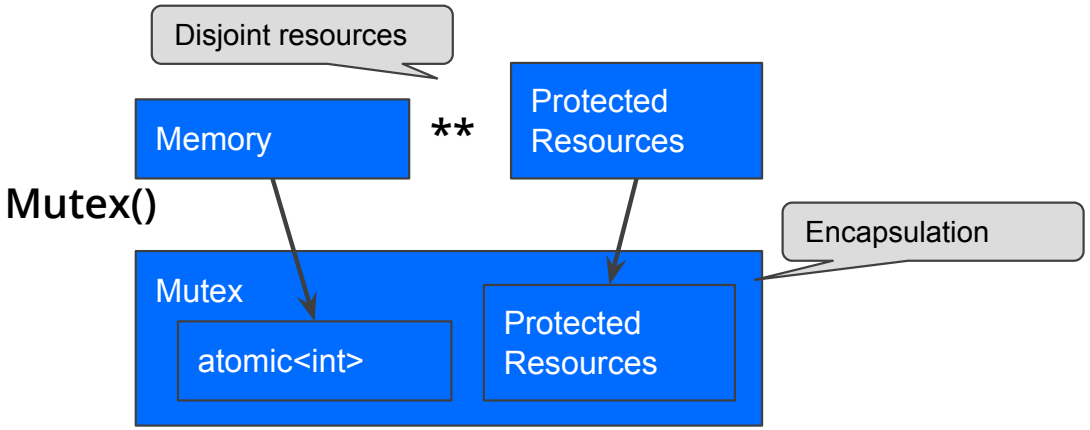


Divide the "world" into disjoint regions.

- **Disjointness enables composition**
- **Implicitly open-world reasoning**
  - New functions
  - New threads
  - ***Dynamic*** rather than static

Separation transcends the programming language, equally useful at all levels of the stack!

What does it look like?



Divide the "world" into disjoint regions.

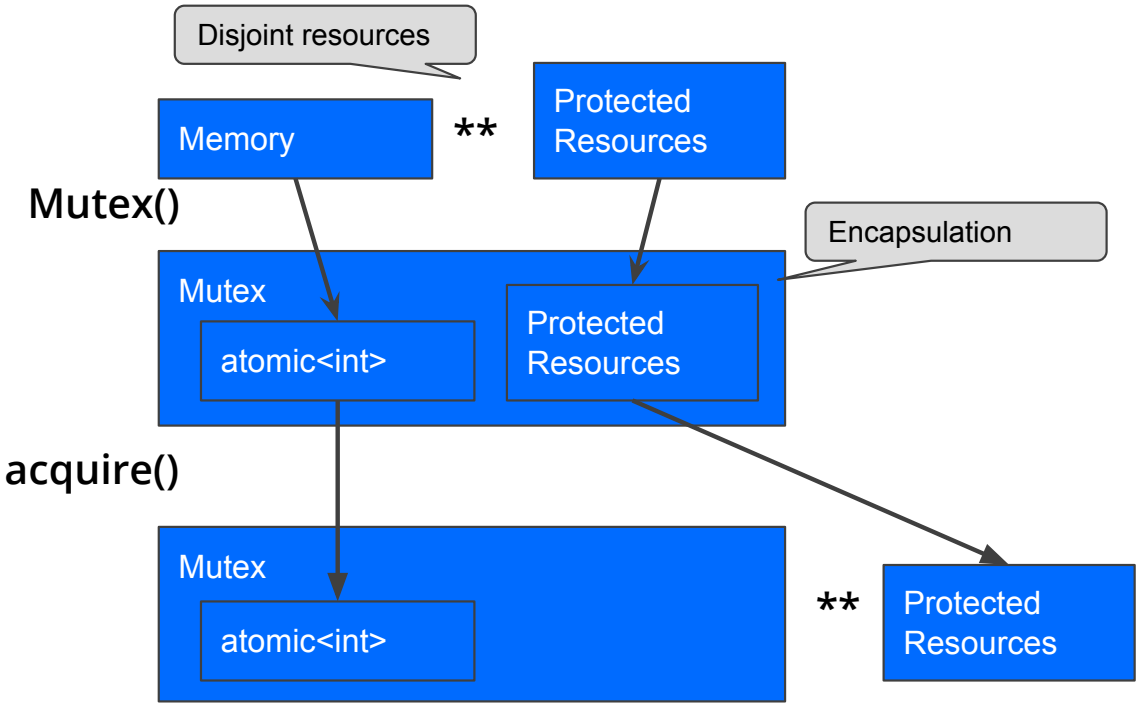
- Disjointness enables composition
- Implicitly open-world reasoning
  - New functions
  - New threads
  - ***Dynamic*** rather than static

Separation transcends the programming language, equally useful at all levels of the stack!

What does it look like?

Divide the "world" into disjoint regions.

- Disjointness enables composition
- Implicitly open-world reasoning
  - New functions
  - New threads
  - ***Dynamic*** rather than static



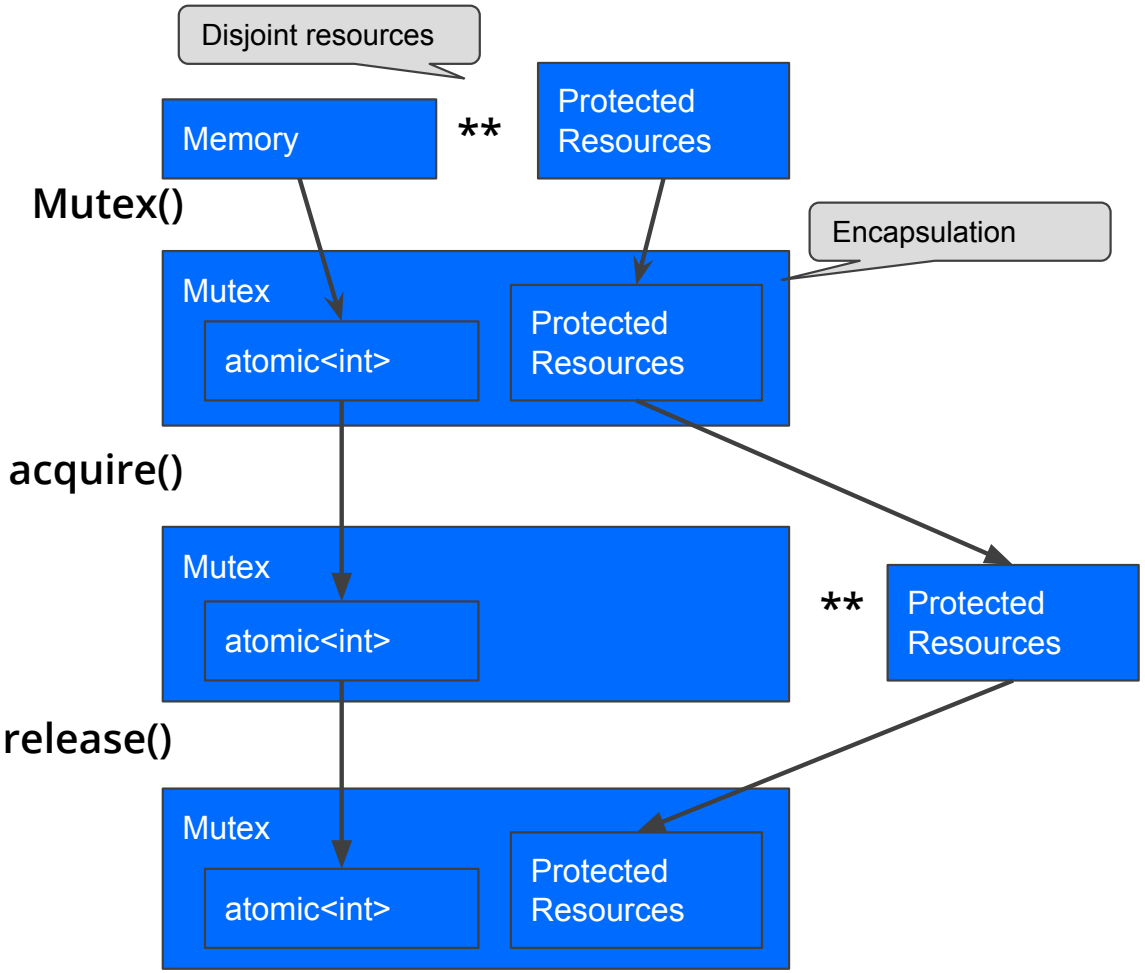
Separation transcends the programming language, equally useful at all levels of the stack!

What does it look like?

Divide the "world" into disjoint regions.

- Disjointness enables composition
- Implicitly open-world reasoning
  - New functions
  - New threads
  - ***Dynamic*** rather than static

Separation transcends the programming language, equally useful at all levels of the stack!



BedRock Systems Inc

Experiences
(clear focus on core problem)

"The specification makes the problem a lot clearer." ***~VMM Developer***

Mediating access to guest memory, races between HW, software instruction emulation, virtual devices, et.al.

- VMM guest memory management

Formal methods provides a *lens* for evaluating designs, and finding alternatives.

Experiences
(clear focus on core problem)

"The specification makes the problem a lot clearer." ***~VMM Developer***

- VMM guest memory management

Mediating access to guest memory, races between HW, software instruction emulation, virtual devices, et.al.

- ACLs for network traffic

Generalized the code-architecture of rules.
Simpler user-facing model and easier to extend.

- Console multiplexer

Clarify the protocol between the control- and data-"planes".
(reusable abstraction)

- Driver architecture

Better understanding of the design space allowed us to iron out generic specifications and evaluate alternatives.

Formal methods provides a *lens* for evaluating designs, and finding alternatives.

FM and "Best Practice" Tend to Agree

("better" code is easier to specify/understand)

- Easy explanation of move vs copy semantics

```
// copy constructor
\pre    this |-> anyR cls
     ** that |-> ClsR m
\post   this |-> ClsR m
     ** that |-> ClsR m

// move constructor
\pre    this |-> anyR cls
     ** that |-> ClsR m
\post   this |-> ClsR m
     ** ∃ m', that |-> ClsR m'
```

that does not change

that is lost (only destructable)

FM and "Best Practice" Tend to Agree

("better" code is easier to specify/understand)

- **Easy explanation of move vs copy semantics**
- **Don't return pointers/references to (certain) internal data**
- **Avoid duplicate information**
  - Potential for Inconsistent views
  - Atomic update is difficult / expensive
- **A function should only work at one level of abstraction**
  - Huge layering improvements
- **Mutation is (often) not necessary**

```
// copy constructor
\pre    this |-> anyR cls
     ** that |-> ClsR m
\post   this |-> ClsR m
     ** that |-> ClsR m

// move constructor
\pre    this |-> anyR cls
     ** that |-> ClsR m
\post   this |-> ClsR m
     ** ∃ m', that |-> ClsR m'
```

`that` does not change

`that` is lost (only destructable)

BEDROCK Systems Inc

Scaling to...

Many dimensions of scale

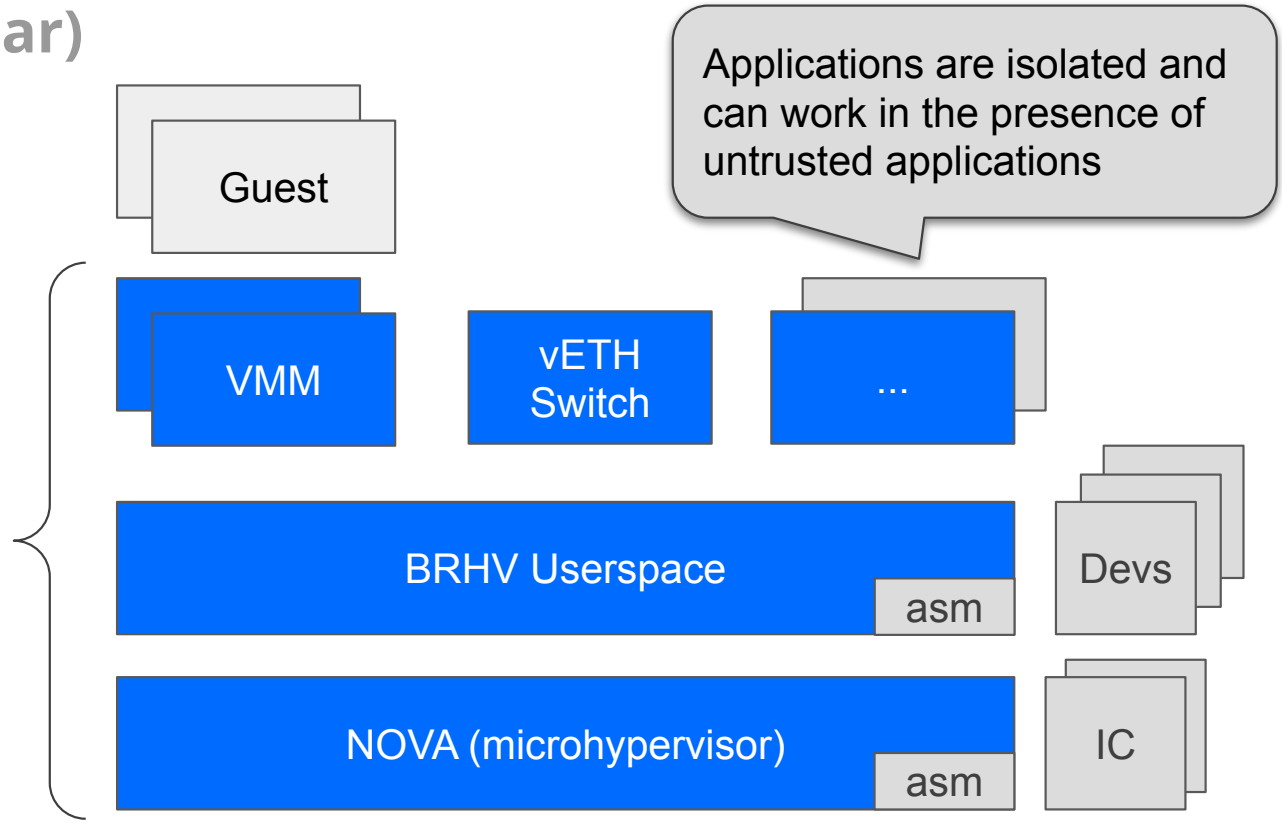
- Mainstream Languages
- Everyday developers
- Heterogeneous code-bases
- Boring problems
- Difficult problems

BedRock
Systems Inc

Verification target
(Heterogeneous & modular)

System built from many components

- Separate verification
- Significant code & proof re-use
- More than just C++
  - Assembly (x86, ARM, ...)
  - Hardware devices
  - Guest code



BEDROCK Systems Inc

Scaling to...

Many dimensions of scale

- Mainstream Languages
- Everyday developers
- Heterogeneous code-bases
- Boring problems
- Difficult problems

BedRock
Systems Inc

Easy Problems, Easy Solutions
(patterns & automation)

Formal methods are pedantic in nature.

- Proofs done in complete detail
- Precise about *everything*

Reduce "boilerplate"

Ensure that the **easy things are easy**

- Leverage the language
- Fall back on the proof assistant

Specification Generators

- ➔ Getters / setters
- ➔ Default operations
- ➔ Simple structures

Customizable Automation

- ➔ Common patterns
- ➔ Domain-specific automation

Scaling to...

Many dimensions of scale

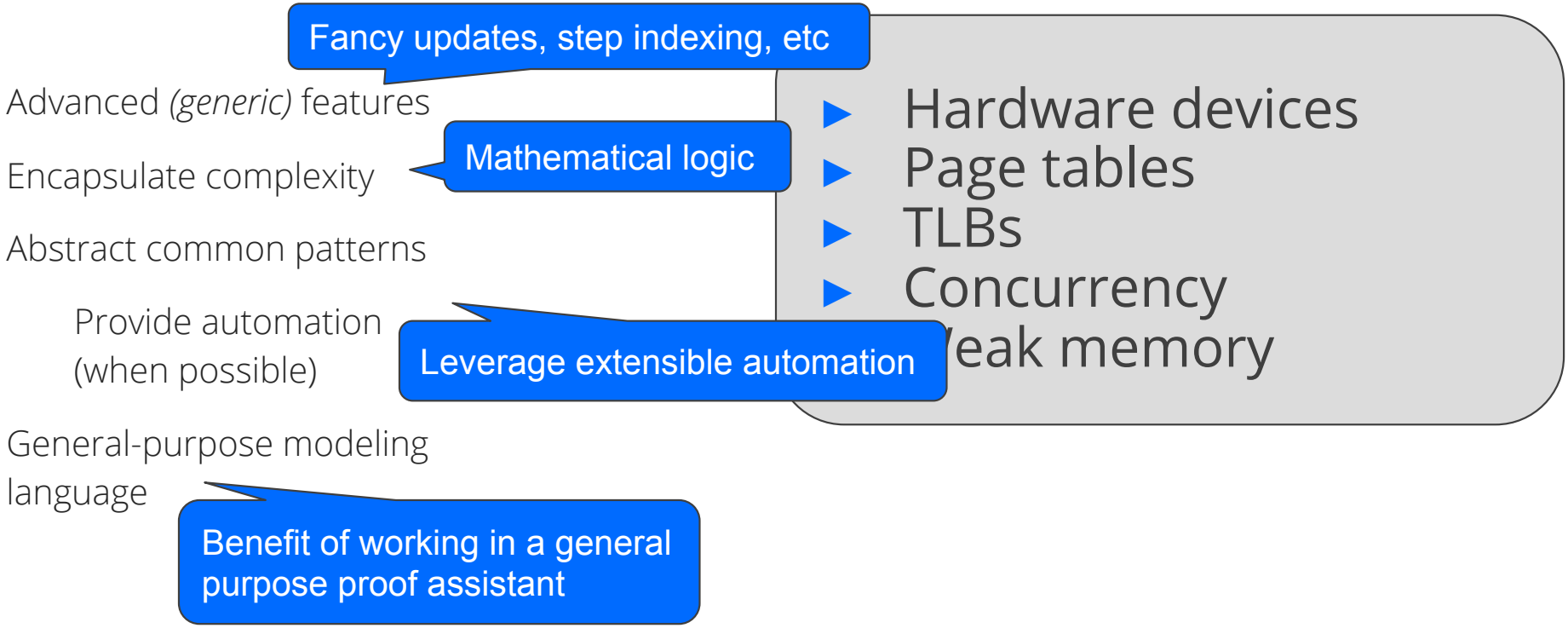
- Mainstream Languages
- Everyday developers
- Heterogeneous code-bases
- Boring problems
- Difficult problems

BedRock
Systems Inc

Hard Problems...

- Hardware devices
- Page tables
- TLBs
- Concurrency
- Weak memory

Hard Problems…
Possible Solutions
(expressivity & patterns)



BedRock Systems Inc

Scaling to...

(Many dimensions of scale)

- Mainstream Languages
- Everyday developers
- Heterogeneous code-bases
- Boring problems
- Difficult problems