Formal Methods at Microsoft

Nikolaj Bjørner, Microsoft Research, RiSE
TLA+ Conference @ StrangeLoop
September 22 - 2022

Formal Methods tools from Design to Diagnoses

TLA+

EverParse

Coyote

Bosque

LEAN

KOKA

Verus

HyperFuzzer

Torch

RESTler

F*

EverCrypt

Zen

NCVS

A tool-oriented view

I will talk about *lots of* tools:

- Scientific heritage
- Target Use
- Impact
- North Stars

Lots of material, yet very partial

I will not cover: TLA+, LƎ∀N, Koka, MakeCode, Orchestration, Parallel Computation, Synthesis, RegEx

No theorems …. but I will point to a z3 guide

Logic – Calculus of Computation

- Symbolic Model Checking
- Model-Based Testing
- Program Verification

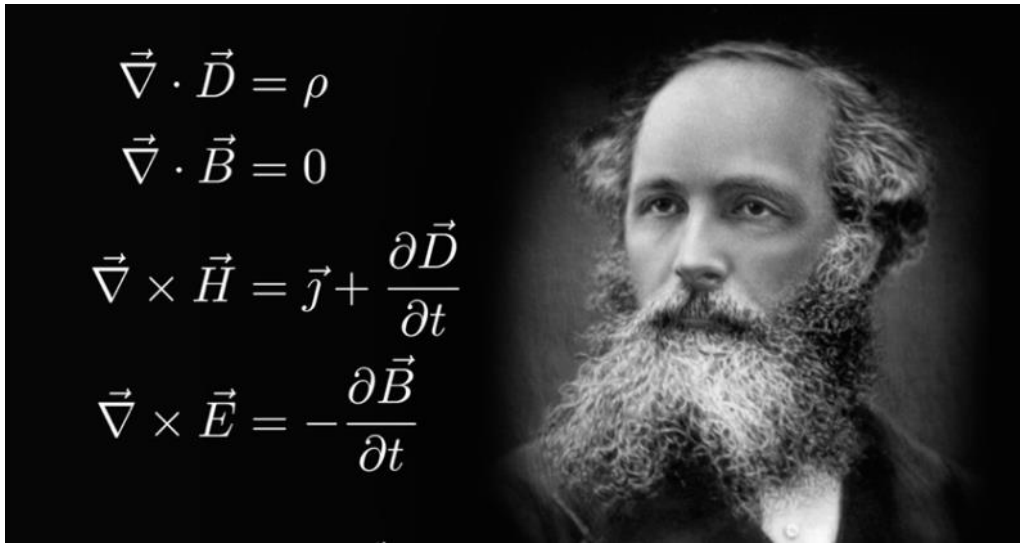
Tomography of Computation

- Concurrency Testing
- Fault Injection
- Fuzzing

Logic and Tomography

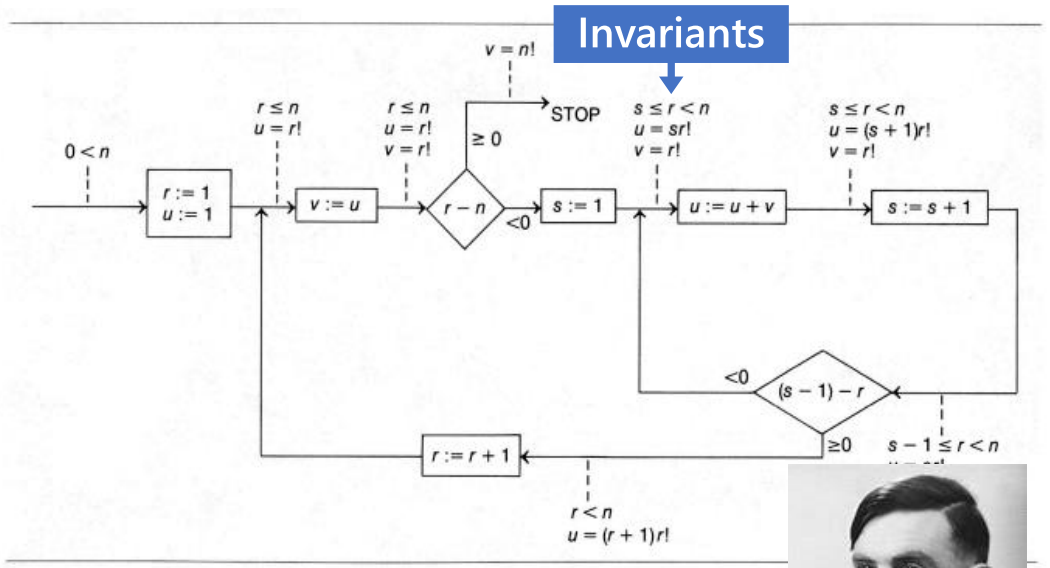
- Network Verification

Logic: The Calculus of Computation



Differential, Integral Calculus

Dynamics, Conduction,..
Matlab, Mathemetica, Simulink



Logic

Computation

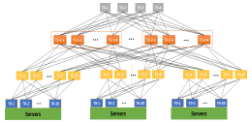
Z3

Friday, 24th June [1949]

Checking a large routine by Dr A. Turing.

Claim: Practically all modern program analysis tools involve solving logical formulas

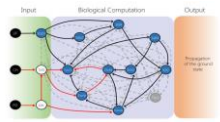
Z3 for Software +…



Azure Network Verification

Verifying C Compiler

Quantum Compilation

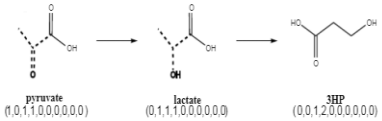


Biological Computations

Verified Crypto Libraries & Protocols

Dynamics AX

SVACE Static Analysis Engines

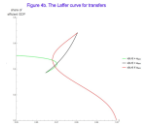


Artificial Life

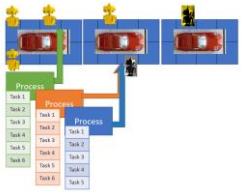
Security Risk Detection

Smart Contract Verification

ALIVE2 Translation Validation for LLVM & Visual C++

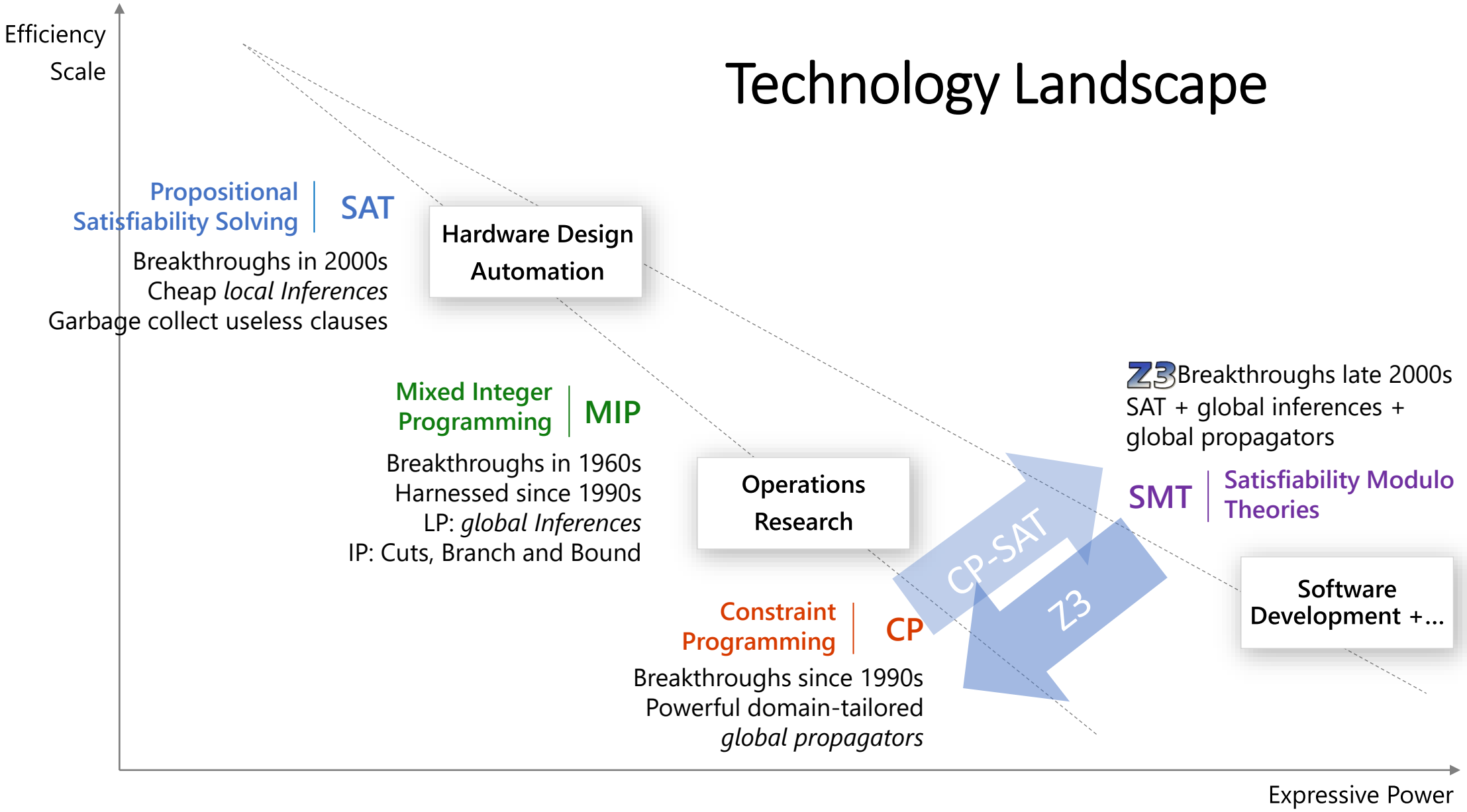


Axiomatic Economics



Assembly Line Optimization

Technology Landscape



formulas with ground terms that appear in the current search context based on *pattern annotations* on quantifiers. The pattern-based instantiation method is quite effective, even though it is inherently incomplete.

Z3 also contains a model-based quantifier instantiation component that uses a model construction to find good terms to instantiate quantifiers with; and Z3 also handles many decidable fragments.

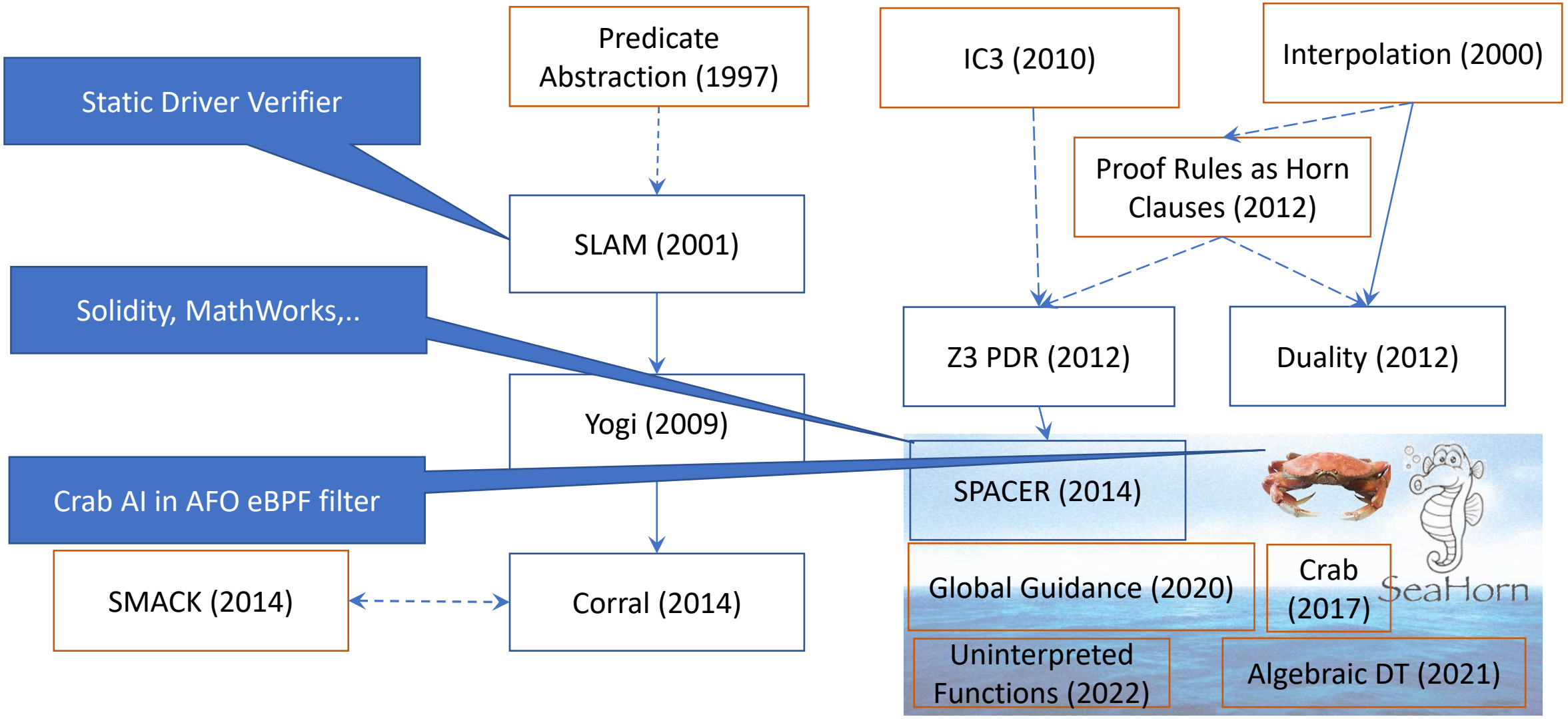
Modeling with Quantifiers

Suppose we want to model an object oriented type system with single inheritance. We would need a predicate for sub-typing. Sub-typing should be a partial order, and respect single inheritance. For some built-in type constructors, such as for array-of, sub-typing should be monotone.

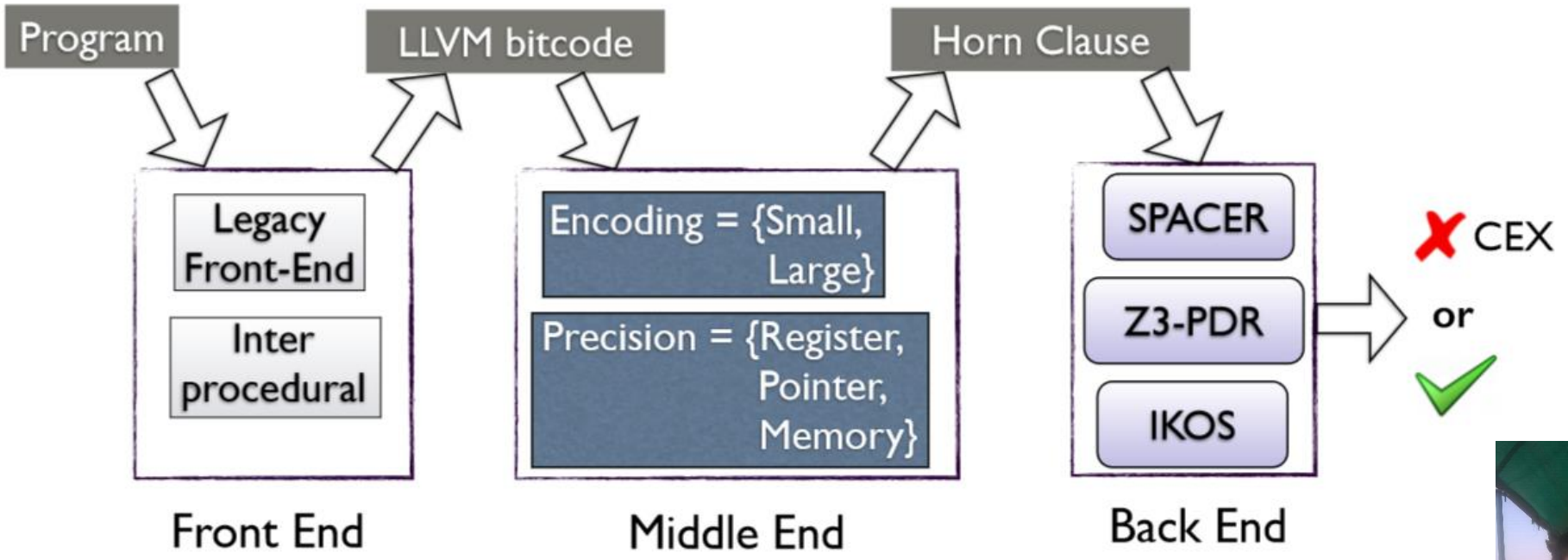
Run Discuss

```
(declare-sort Type)
(declare-fun subtype (Type Type) Bool)
(declare-fun array-of (Type) Type)
(assert (forall ((x Type)) (subtype x x)))
(assert (forall ((x Type) (y Type) (z Type))
          (= (and (subtype x y) (subtype y z))
             (subtype x z))))
(assert (forall ((x Type) (y Type))
          (= (and (subtype x y) (subtype y x))
             (= x y))))
(assert (forall ((x Type) (y Type) (z Type))
          (= (and (subtype x y) (subtype x z))
             (or (subtype y z) (subtype z y)))))
(assert (forall ((x Type) (y Type))
          (= (subtype x y)
             (subtype (array-of x) (array-of y)))))
(declare-const root-type Type)
(assert (forall ((x Type)) (subtype x root-type)))
(check-sat)
```

Symbolic Model Checking



Constrained Horn Clauses



$$\forall X.\ X > 100 \rightarrow \mathrm{mc}(X, X - 10)$$

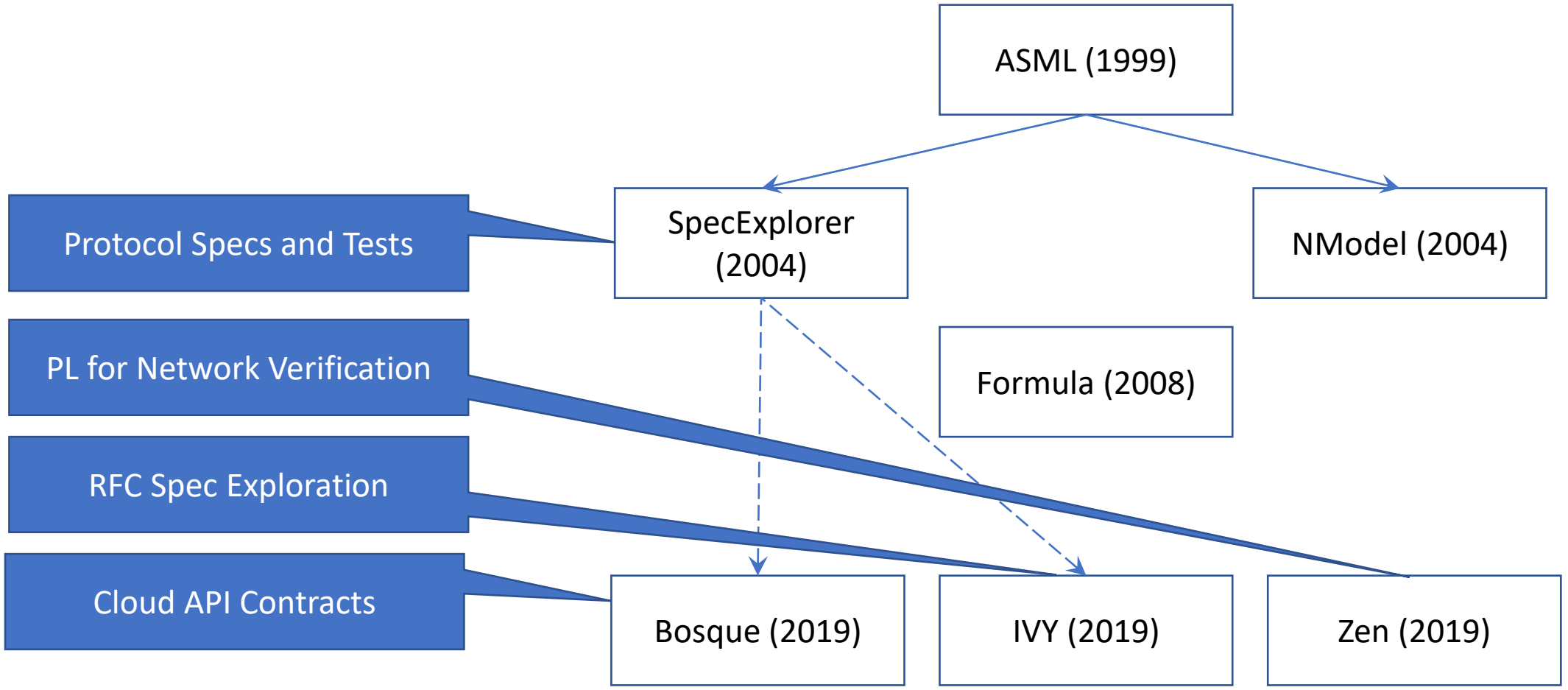
$$\forall X, Y, R.\ X \leq 100 \wedge \mathrm{mc}(X + 11, Y) \wedge \mathrm{mc}(Y, R) \rightarrow \mathrm{mc}(X, R)$$

$$\forall X, R.\ \mathrm{mc}(X, R) \wedge X \leq 101 \rightarrow R = 91$$

***Solver finds solution for* mc (McCarthy 91 function)**

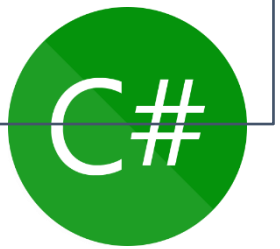
Arie Gurfinkel
U. Waterloo

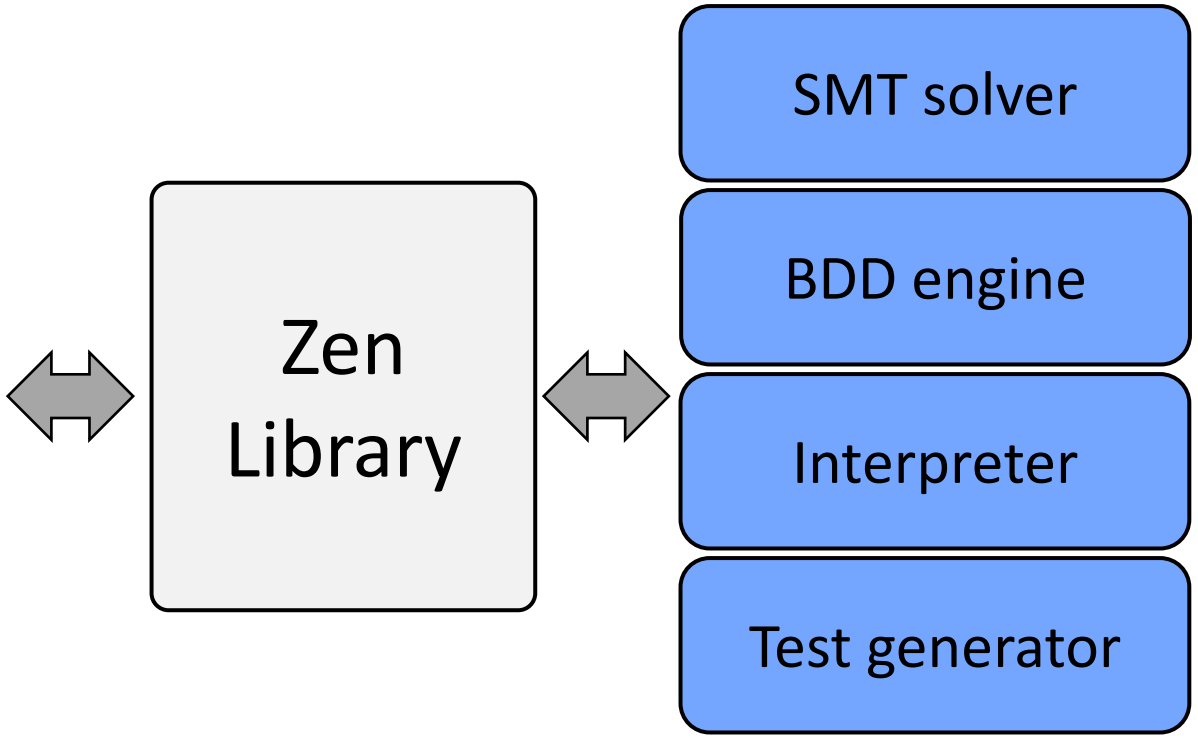
Model Based Testing and Model Programs



Zen - an intermediate policy representation

```
Zen<bool> Allow(Nsg nsg, Zen<Packet> pkt, int i) {
    if (i >= nsg.Rules.Length)
        return false;
    var rule = nsg.Rules[i];
    return If(Matches(rule, pkt),
        rule.Permit,
        Allow(nsg, pkt, i+1));
}
```





Embedded Domain-Specific Language in C# [1]
https://github.com/microsoft/zen

Backends

[1] A General Framework for Compositional Network Modeling, R. Beckett, R. Mahajan, HotNets 2020

Microsoft

Bosque - for Financial Compliance (OSFIR)

Functional IR (MorphIR)

- Referential Transparency
- Combinators: Map, Fold
- Analysis Friendly

```
typedecl ZipcodeUS = /[0-9]{5}(-[0-9]{4})?/;

function isNYCode(s: StringOf<ZipcodeUS>): Bool {
  return s.value().startsWith(/1[0-4]/);
}

isNYCode('10001'#ZipcodeUS) //true
isNYCode('87111'#ZipcodeUS) //false
isNYCode("12") //type error not a StringOf<ZipcodeUS>
isNYCode('WC1E'#PostcodeUK) //type error not a StringOf<ZipcodeUS>
```

Bosque provides

- Verification
- Test-case generation

```
typedecl Percentage = Nat & {
    invariant $value <= 100n;
}

let a = 100#Percentage;
let b = 101#Percentage;  //Runtime Error
let q = a + 25#Percentage; //Runtime Error
```

Bosque – The Future of the Cloud is APIs

Cloud Service Compositionality

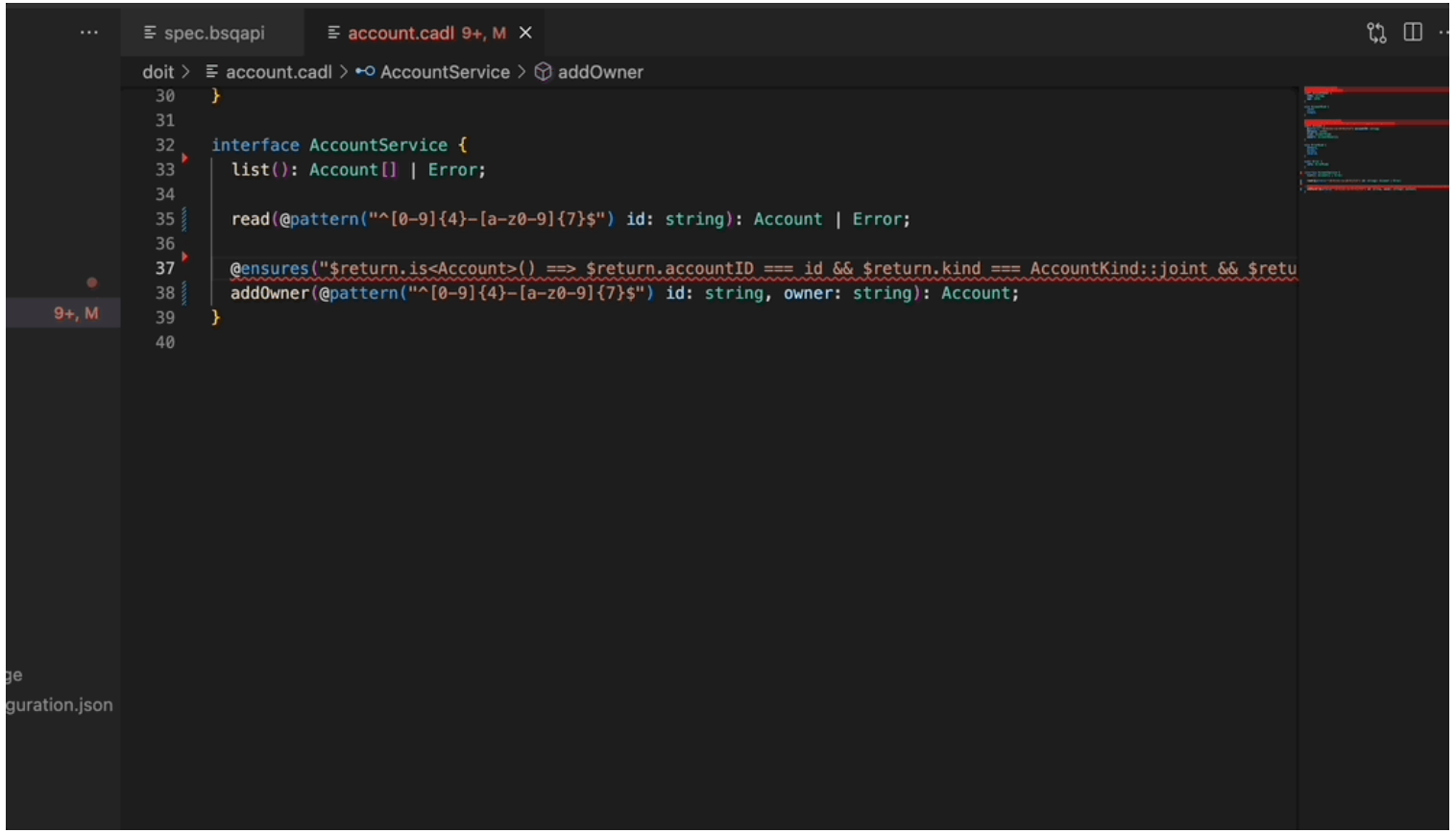
Cloud Service Contracts

- @requires
- @ensures
- @invariant

String Constraint Types

Bosque provides

- Fuzz Tests
- Auto-Mock



Systems Code Verification

FP & Logics

F# (2001)

Verve (2010)

F7 (2007)

Vale (2017)

F* (2012)

EverCrypt (2019)

EverParse (2019)

Verona (2022)

SLAyer (2008)

Steel (2020)

Verus (2022)

Armada (2020)

Separation Logic (1999)

Boogie (2004)

KISS (2004)

Spec# (2004)

HAVOC (2010)

VCC (2009)

Dafny (2010)

Chalice (2010)

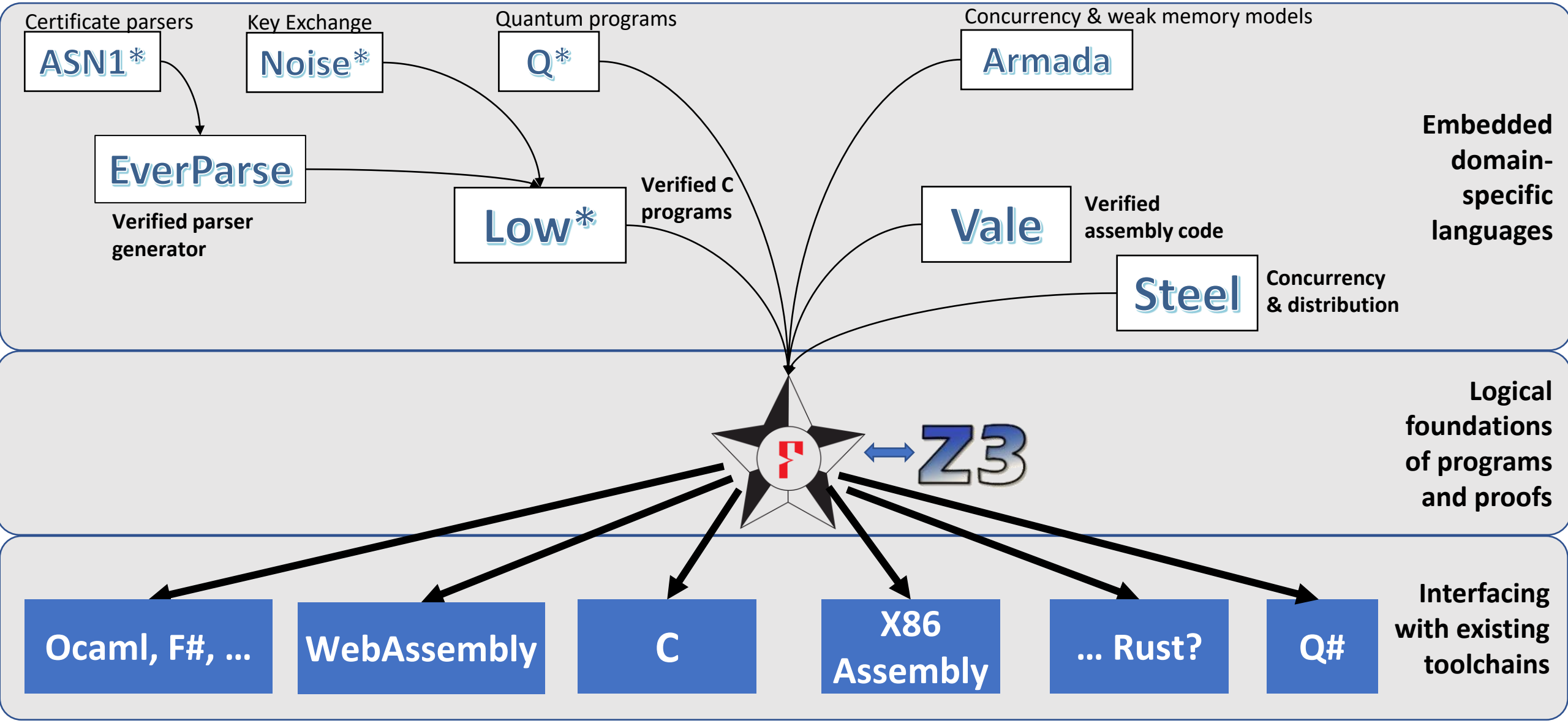
Ironfleet (2015)

CIVL (2015)

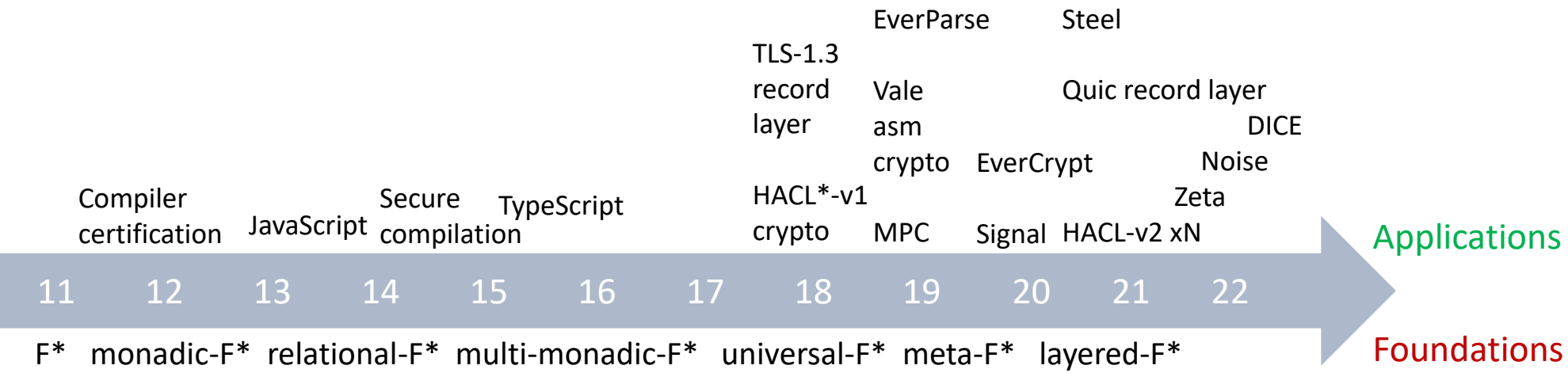
Viper (2016)

VeriSol (2018)

Languages embedded in F*



F*: A Proof-oriented Programming Language



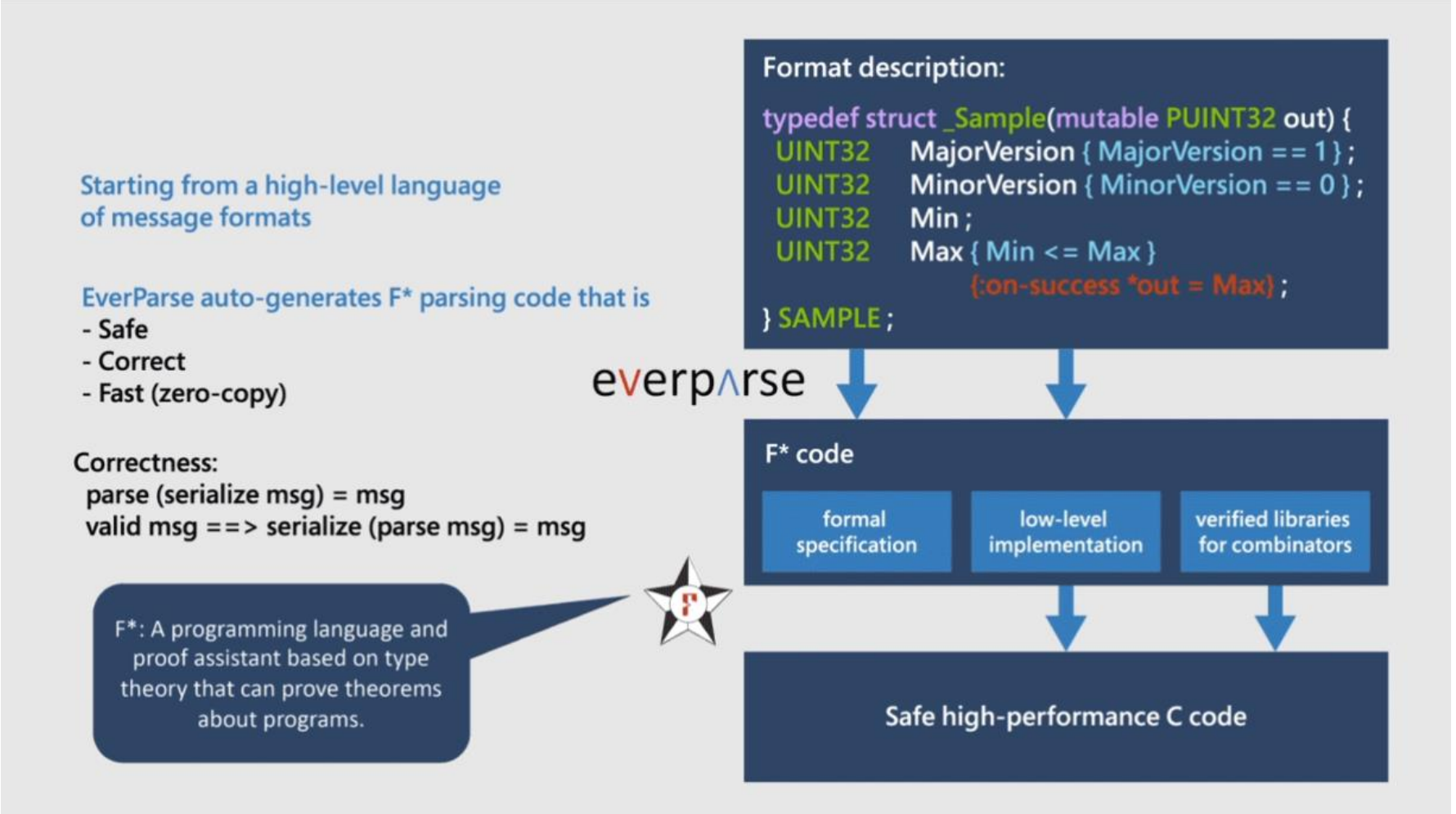
- Extensional type theory, with refinement subtyping, proof irrelevance, tactic & SMT-based proofs
- Expressiveness to state and attempt a proof of nearly any statement in mathematics (like Coq, Lean etc.)
- With a focus on programming, including features, like state, exceptions, non-determinism, concurrency, IO
- Integrated with Z3, so that many proofs are automatic,
  - but when Z3 gives up, you can fall back on manual proofs

EverParse

Mathematically proven parser generator

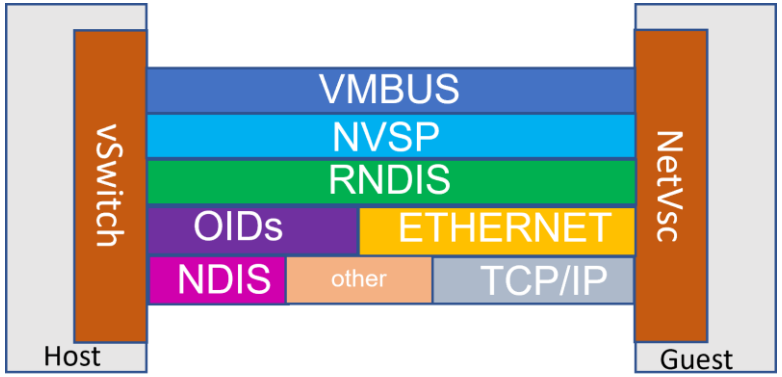
High-performance code generation

Integration into critical systems code



EverParse - Hyper-V vSwitch since 2019

- **Now in Windows 10, 11, and Azure Cloud**: Every network packet passing through Hyper-V is validated by EverParse's formally verified code
- **Hand-written alternative**: historically 30% of all Hyper-V bugs are due to parsing

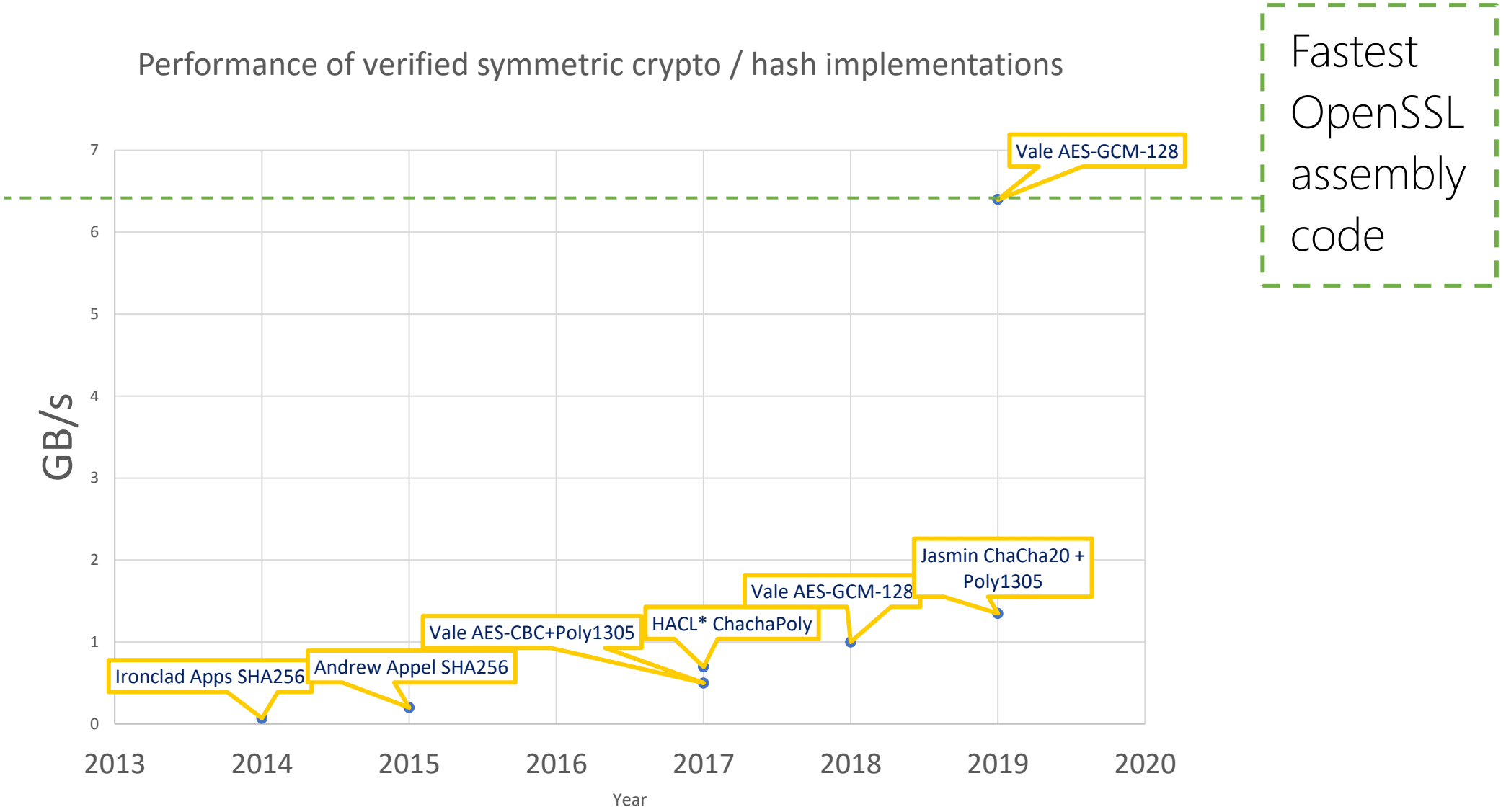


- NVSP, RNDIS, OIDs and NDIS
  - Some of which are proprietary
  - Other formats (TCP, etc.) in progress
- 5K lines of 3D specification
  - 137 structs, 22 casetypes, 30 enum types
- Verified in 82 s
- Generated 23K C code
- High performance: <2% cycles/byte overhead

A sweet spot for formal proof

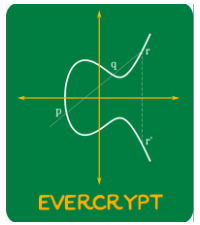
- Guarantees of memory safety and functional correctness
- Provably correct by construction: Zero user proof effort
- High-performance code generated from data format description in a high-level declarative language
- High return on investment wrt. attack surface

Vale - ***Fast*** verified crypto (via verified assembly)



Algorithm	Portable C (HACL*)	Intel ASM (Vale)	Agile API (EverCrypt)
AEAD			
AES-GCM		✔ (AES-NI + CLMUL)	✔
Chacha20-Poly1305	✔ (+ AVX,AVX2)		✔
ECDH			
Curve25519	✔	✔ (BMI2 + ADX)	
P-256	✔		
Signatures			
Ed25519	✔		
P-256	✔		
Hashes			
MD5	✔		✔
SHA1	✔		✔
SHA2-224,256	✔	✔ (SHAEXT)	✔
SHA2-384,512	✔		✔
SHA3	✔		
Blake2	✔ (+ AVX,AVX2)		
Key Derivation			
HKDF	✔	✔ (see notes below)	✔
Ciphers			
Chacha20	✔ (+ AVX,AVX2)		
AES-128,256		✔ (AES-NI + CLMUL)	
MACS			
HMAC	✔	✔ (see notes below)	✔
Poly1305	✔ (+ AVX,AVX2)	✔ (X64)	

EverCrypt



A verified high-performance cryptographic provider
- A *collection* of algorithms **(exhaustive)**
- Easy-to-use API **(CPU auto-detection)**
- Several *implementations* **(multiplexing)**
- APIs grouped by *family* **(agility)**

Clients get **state-of-the art performance.**
- 130,000 lines of Low* and 24,000 lines of Vale (F* DSLs)
- 65,000 lines of C + 15,000 lines of ASM

Proof : Code ratio ~ 2 : 1

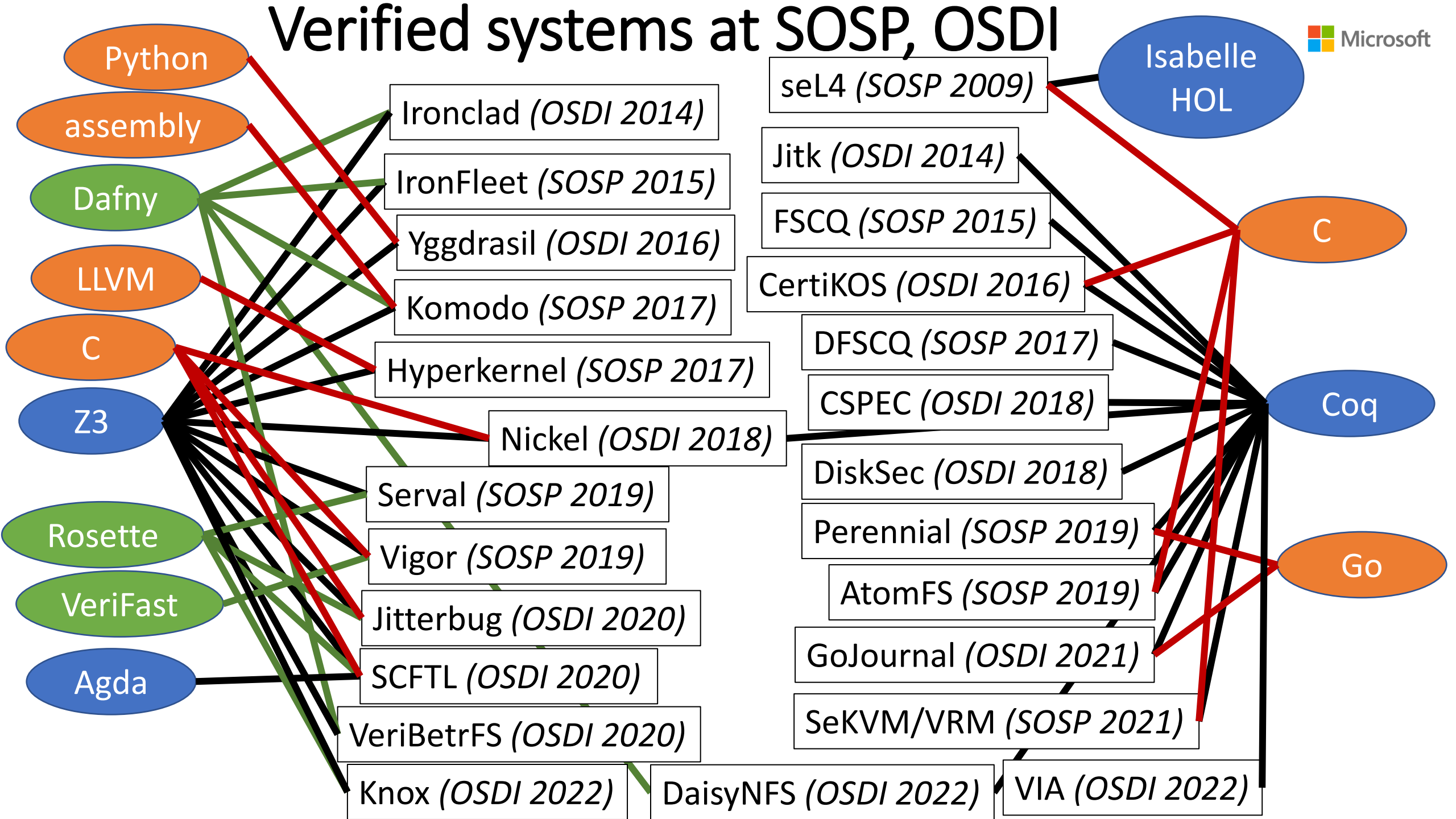
EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider; IEEE S&P 2020


Azure CCF
DICE
Signal
Windows
arm MBED
ElectionGuard

Verified cryptography in the Linux kernel

Quic transport, MSQuic in Windows, Verified crypto in Firefox, mbedTLS,

Verified systems at SOSP, OSDI



Microsoft

Languages for verifying systems code

F*

Dafny

Verified Rust?

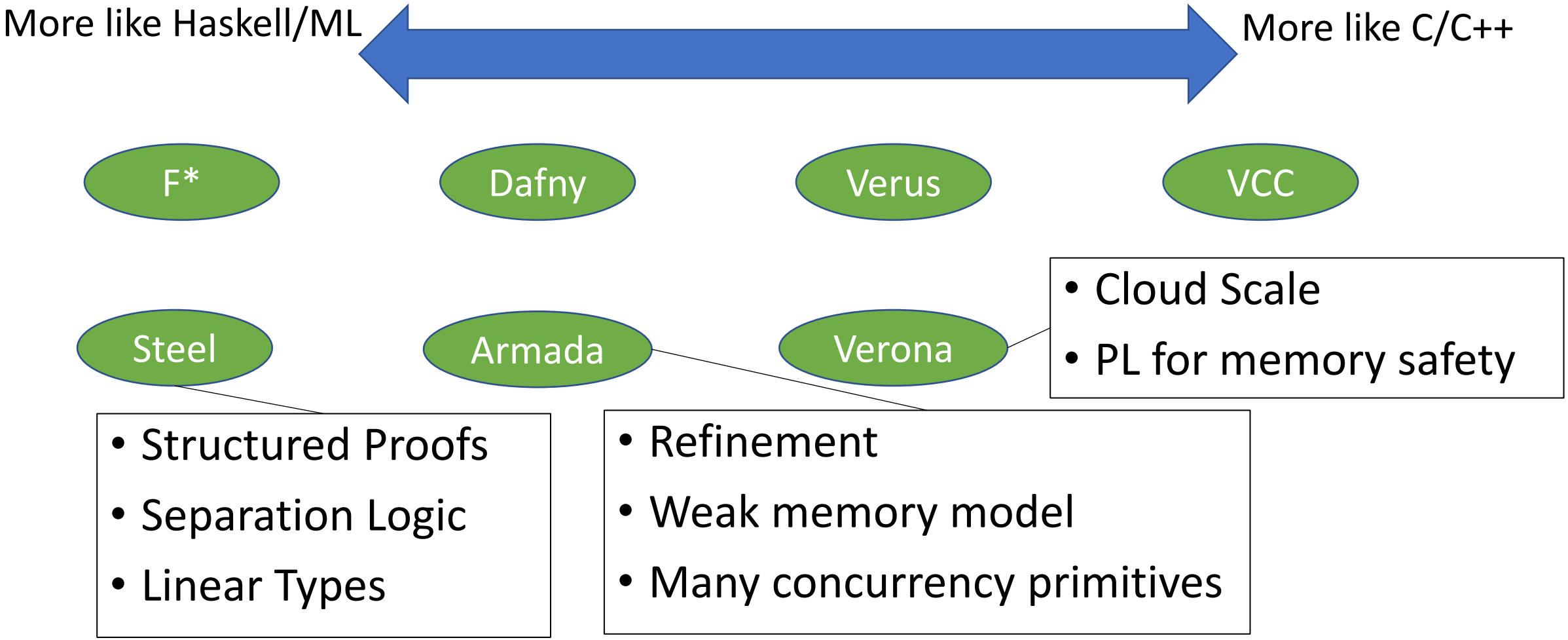
VCC

- Rust is like Haskell/ML:
  - Type-safe
  - Algebraic datatypes
  - Pattern matching
  - Side effects are restricted

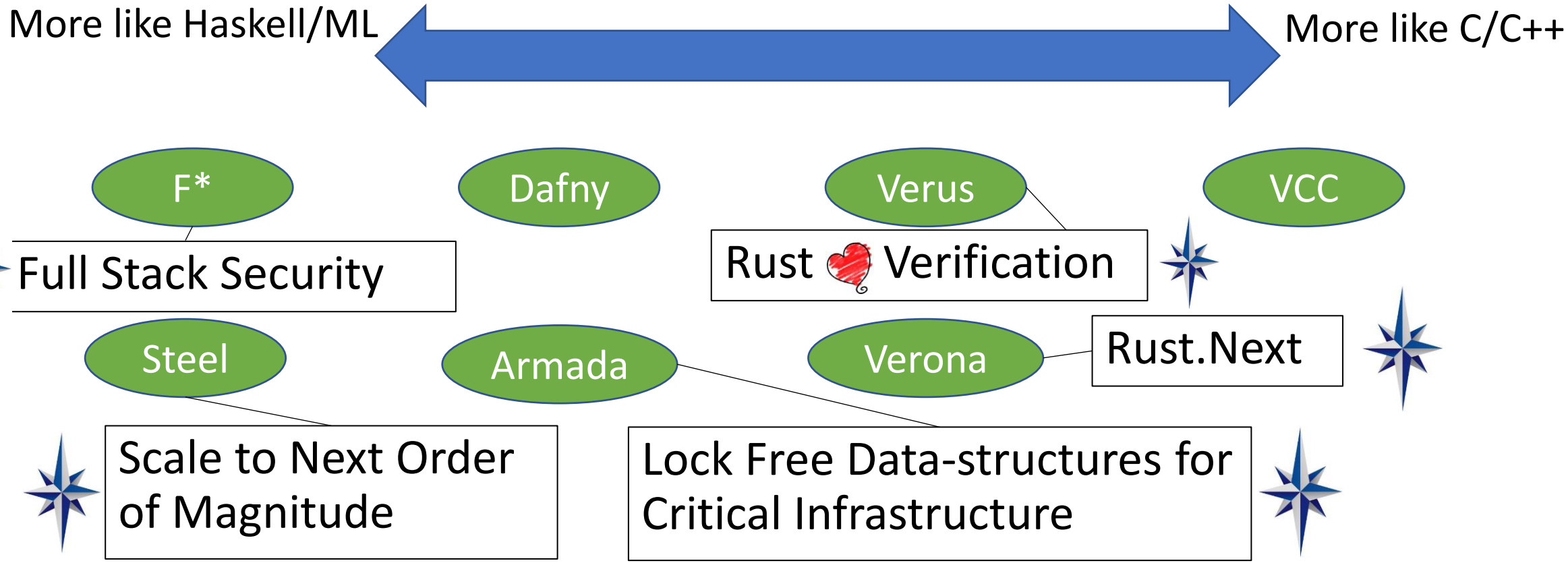
- Rust is unique:
  - Linear types
  - Borrowing

- Rust is like C/C++:
  - Low-level pointer manipulation
  - No garbage collector
  - Structs inside structs
  - In-place mutation of fields

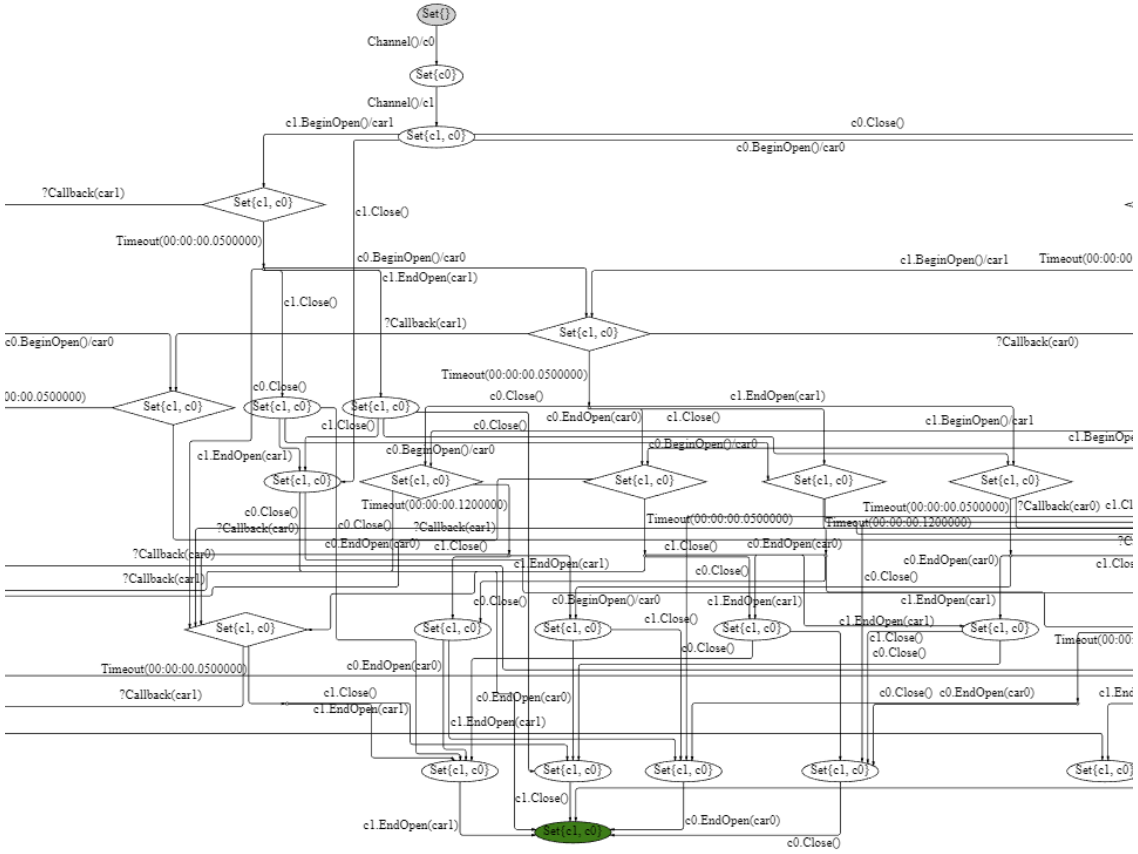
Languages for verifying systems code



Languages for verifying/safe systems code



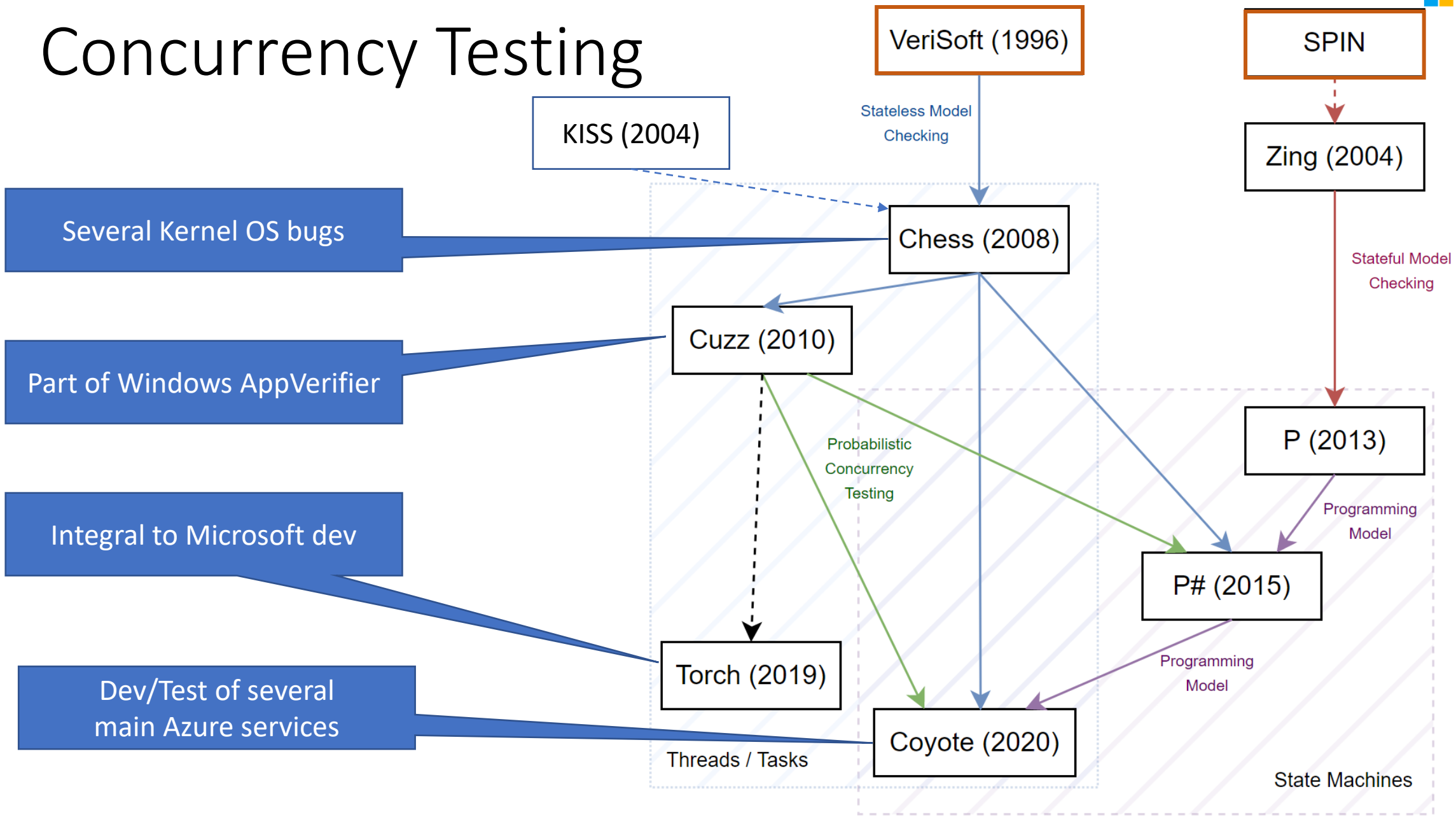
Tomography of Computation



[SVG renderer with MSAGL-JS (microsoft.github.io)]

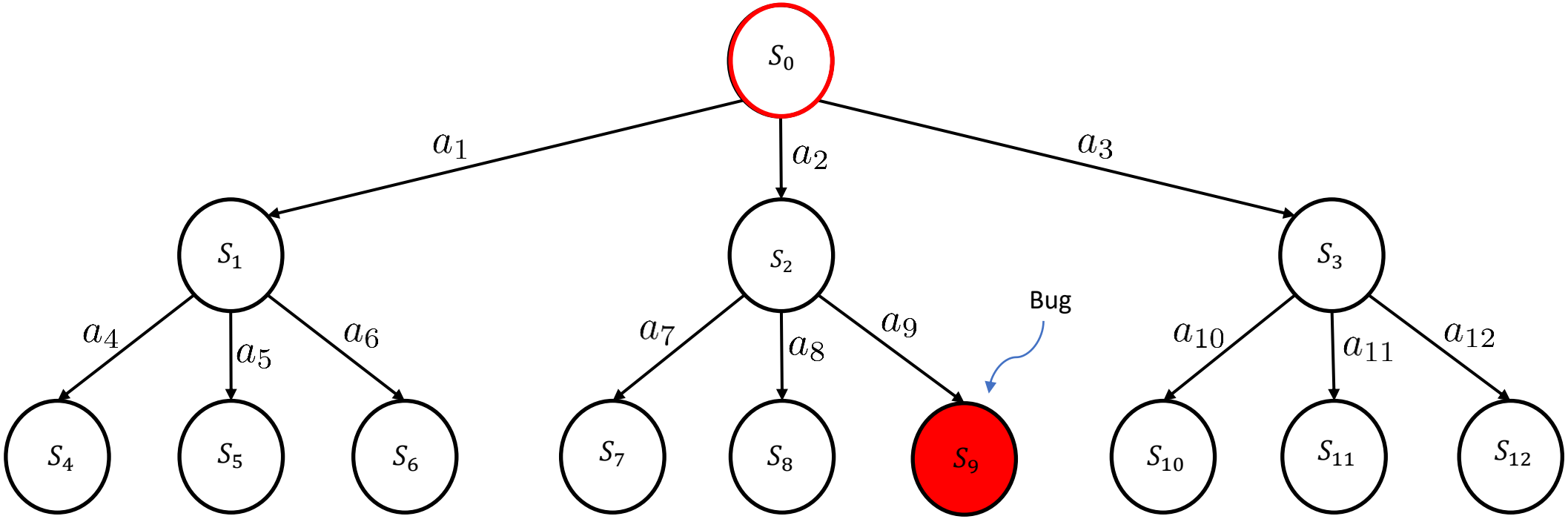
Giovanni Domenico Cassini
Topographic map of France

Concurrency Testing



Controlled Concurrency Testing

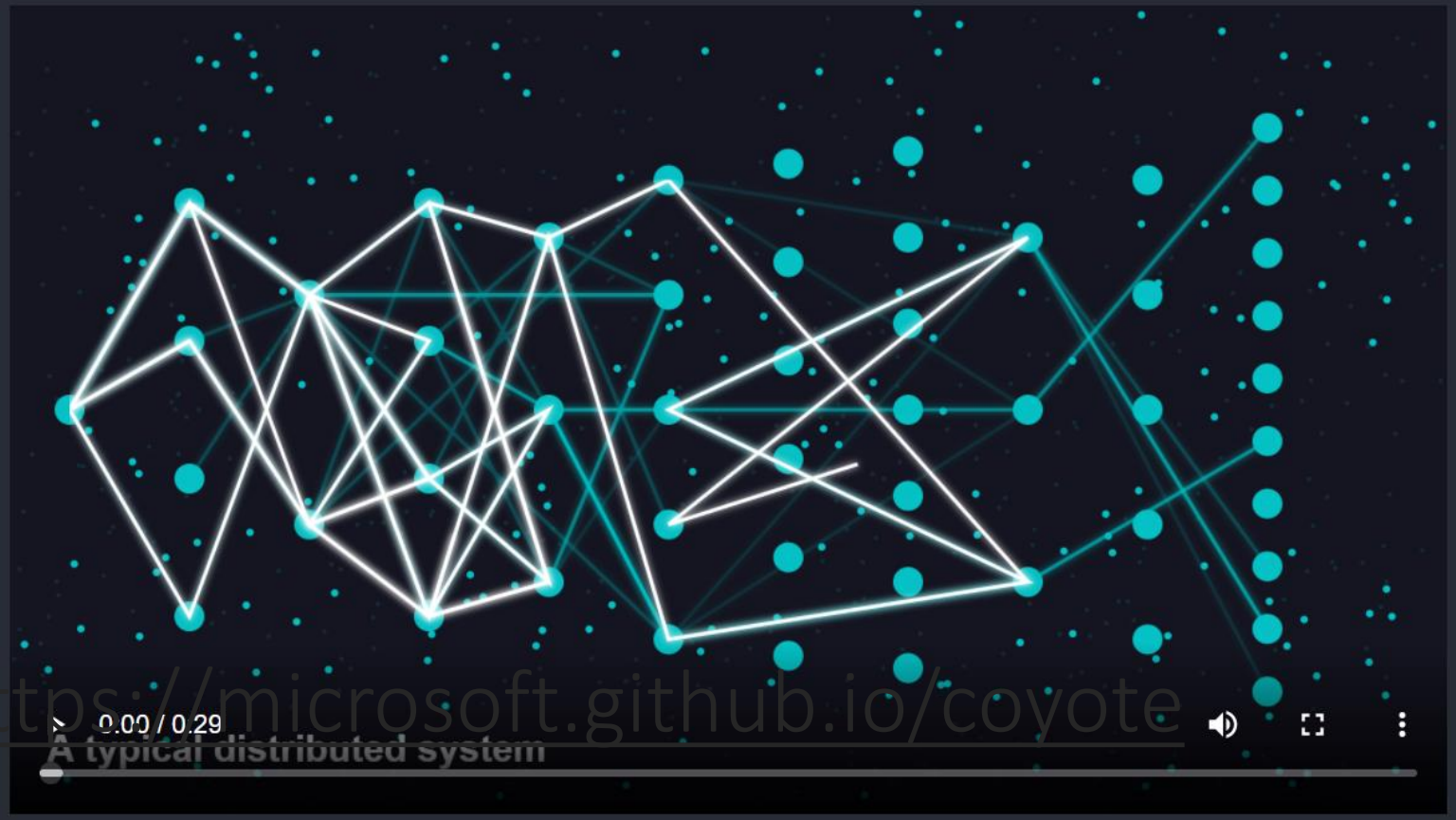
Systematically explore space of program behaviors
…by *serializing* concurrent program executions
…using a *scheduler* which resolves control non-determinism
…with *deterministic* replay



Concurrency Unit Testing with Coyote

Coyote is .NET library and tool designed to help ensure that your code is free of concurrency bugs.

Too often developers are drowning in the complexity of their own code and many hours are wasted trying to track down impossible-to-find bugs, especially when dealing with *concurrent* code or various other sources of *non-determinism* (like message ordering, failures, timeouts and so on).



Coyote helps write powerful, expressive tests for your code. We call these *concurrency unit tests*. You can declare sources of non-determinism (such as timeouts and failures) as part of your Coyote tests. The Coyote testing tool can *systematically*

Coyote @ Microsoft

Used in multiple Azure services

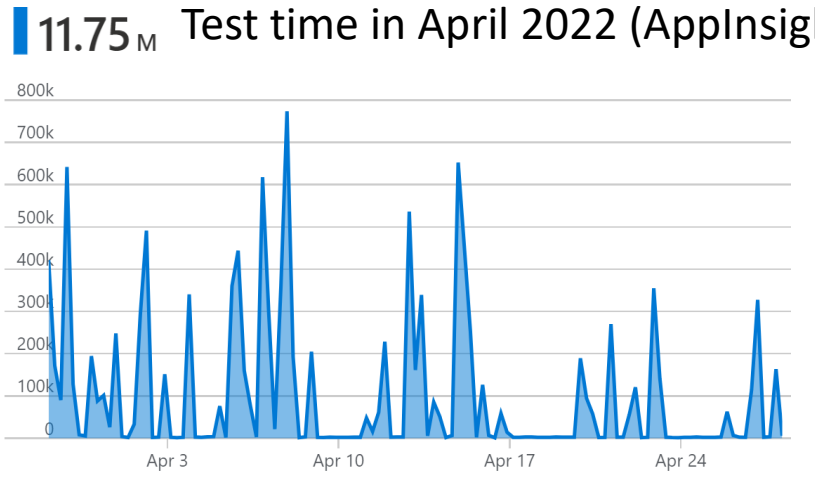
- Usage ranges from unit testing to end-to-end scenarios
- Covers failover, interleavings, timing, etc.

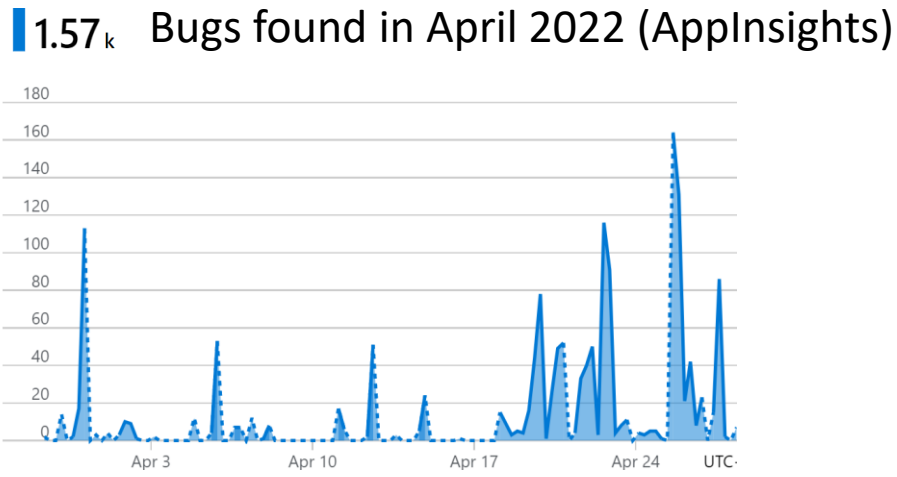
GitHub

1.2k stars

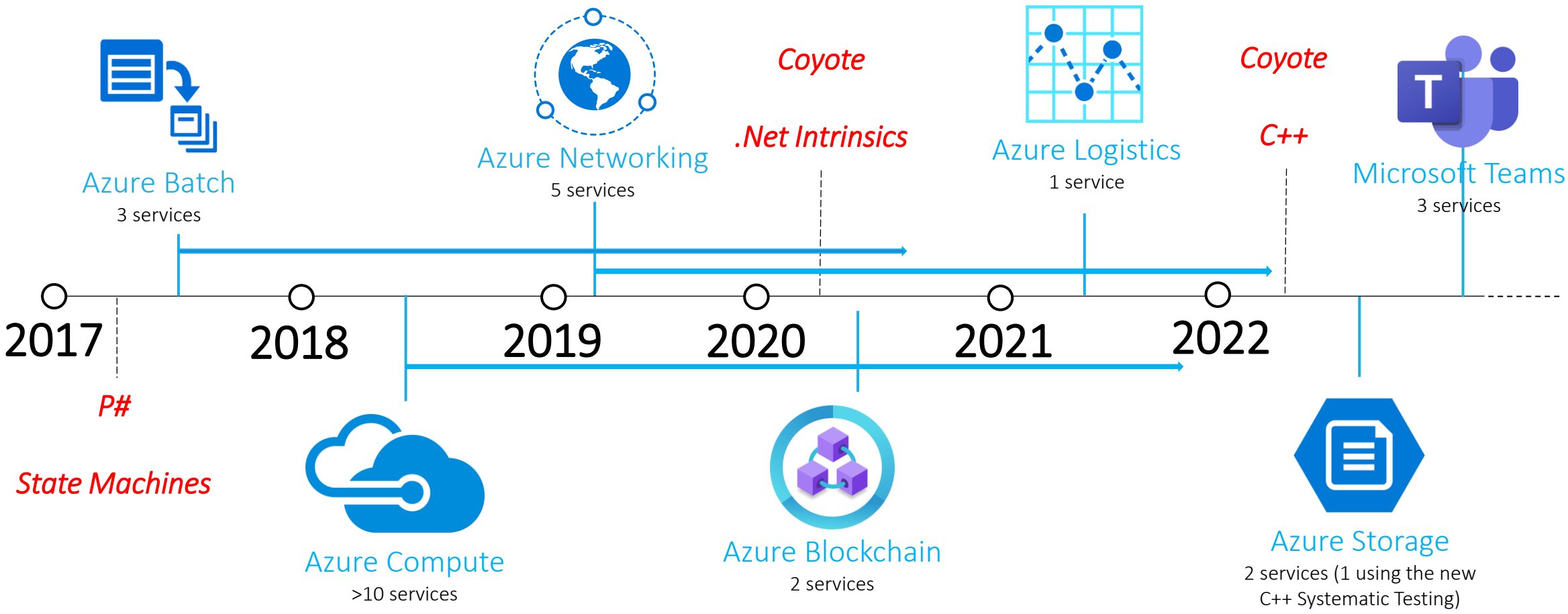
NuGet

Downloads **988.1K**





Coyote @ Microsoft



Torch: Discover Synchronization "Torchpoints"

Randomized

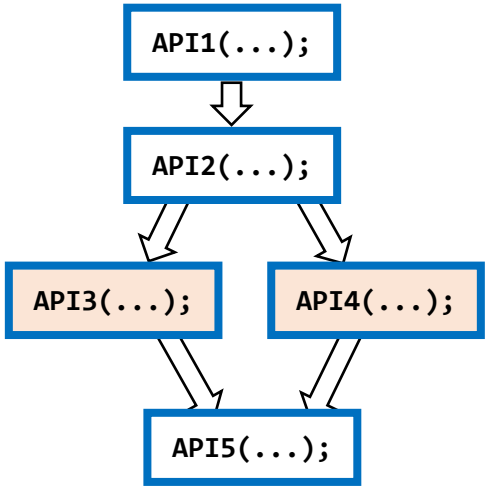
Trap at all TorchPoints, with a probability

+ Emulates a randomized priority scheduler
+ Finds each bug with a bounded probability
- Too many traps, may cause timeouts

Happens-before tracking

***Track** happens-before relationship*
Trap at independent TorchPoints

+ Fewer traps
- High tracking overhead
- Tricky to get right



Happens-before inference

***Infer** happens-before relationship at runtime*
Trap at independent TorchPoints

+ Fewer traps
+ Low tracking overhead

[SOSP'19]

Microsoft

Torch: concurrency and fault-handling bugs

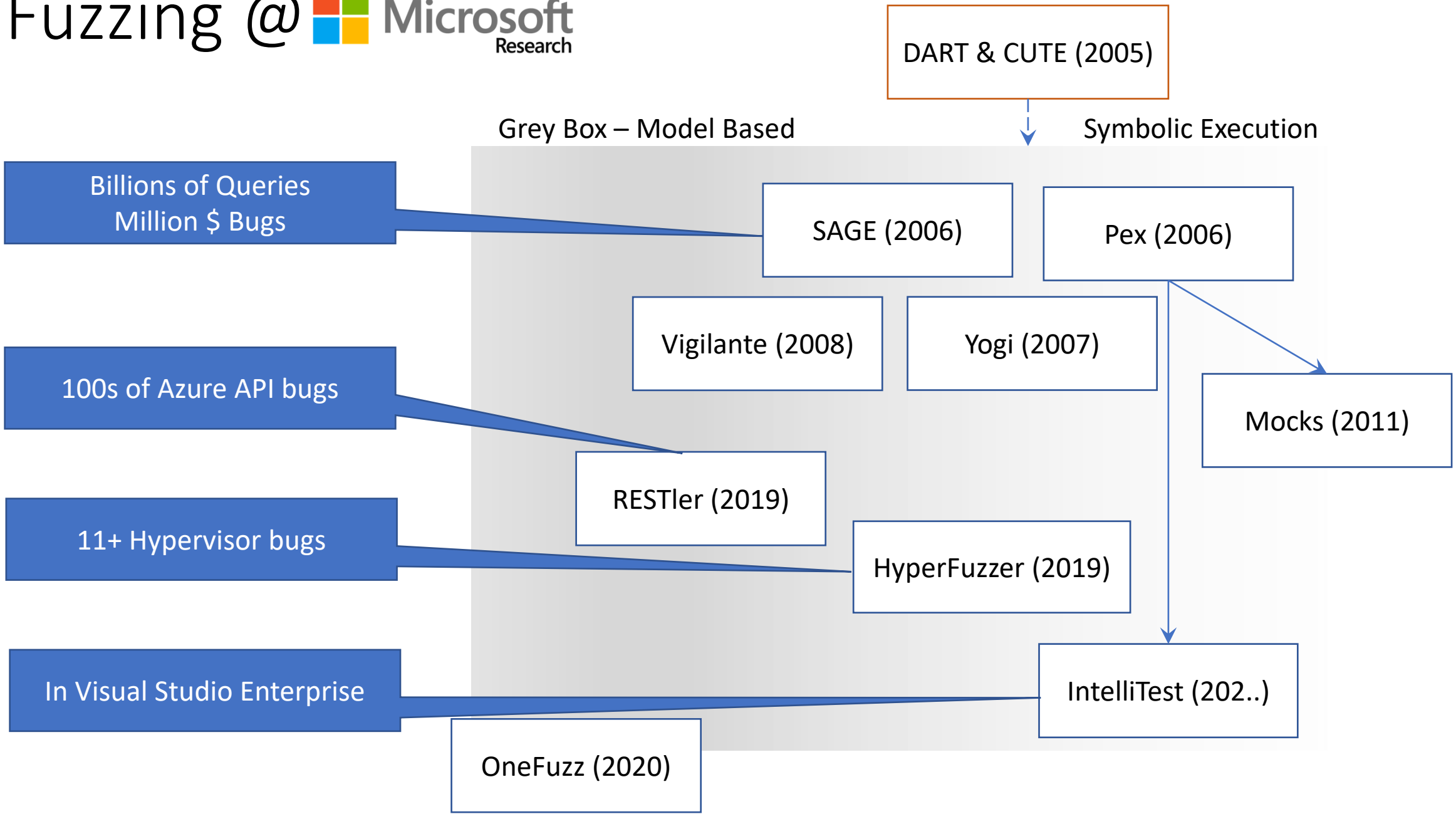
A push-button bug-finding tool for existing systems

- Uses automated instrumentation + intelligent runtime algorithms
- Reports bugs with existing tests, and without false positives
  - Concurrency bugs: due to thread-safety and order violation, interleaving, etc.
  - Fault-handling bugs: due to runtime faults

Integrated into Microsoft's CloudBuild

- Each day: ~300K tests are run with Torch, Torch reports bugs in ~1K tests
- So far: reported ~3K unique bugs in ~30 Microsoft services

Fuzzing @ Microsoft Research



Microsoft

RESTler: REST API Fuzzing

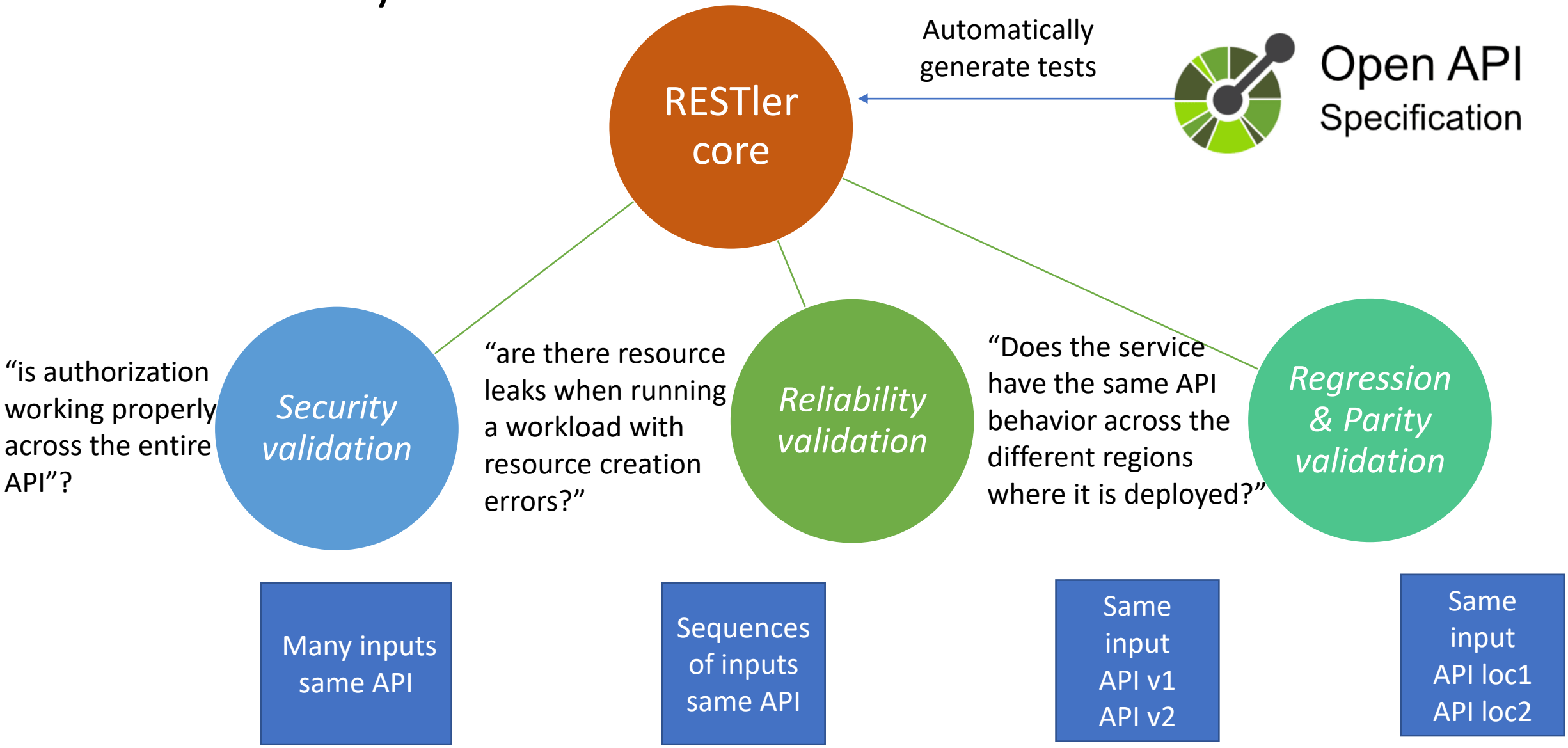
RESTler = 1st Stateful REST API Fuzzer

- Takes an API specification and automatically generates tests

- Finds *security* and *reliability* bugs in REST services
  - Systematic state-space search and learning from responses
  - Input payload (schema and value) fuzzing
  - Targeted checks for security property violations

- Open sourced on GitHub (since November 2020)

RESTler: beyond fuzz



HyperFuzzer – Fuzzing the Hypervisor

Unchartered territory

Traditional fuzzing rely on an OS layer for capturing instructions

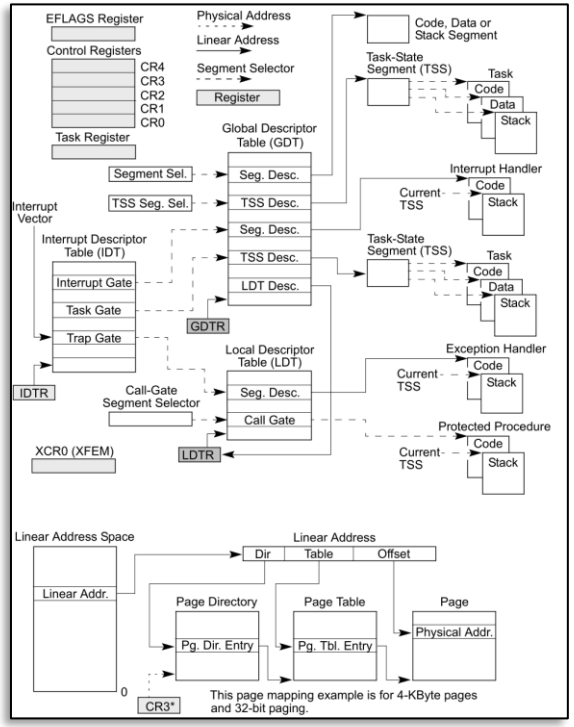
The hypervisor does not have an OS…, doh

- mixed executions between the guest and the hypervisor?
- “hidden” hardware checks?
- hypervisor internal states?

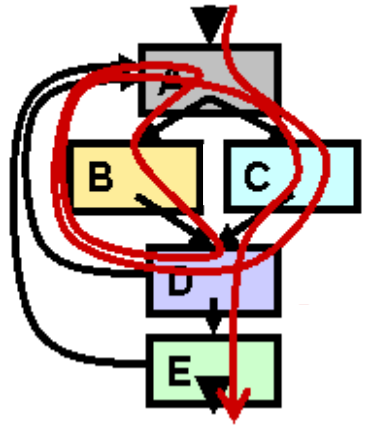
HyperFuzzer – Nimble white + Grey box fuzzing

Key insight: It's the VM state that drives the hypervisor execution

VM State



Instruction Sequence



Symbolic Execution

1000+
tests/second

95%
accuracy

11 bugs

Hypervisor bug bounty @250K$

Availability, Reliability and the 5 9’s

What is a correct-by-construction network?

Reality ≡ Intent

The ideal of verified software
Tony Hoare, 2007

The Ideal of Verified Networks

Lasting power of abstraction: Internet Protocol isolated from underlying transport
Vint Cerf

Main questions in the back of our minds

1. How is intent specified?
2. Correct by construction: How is intent translated to configurations?
3. How is drift tracked and remediated?
4. How can we prevent changes from violating intent?

$\Rightarrow$ Azure Network Verification 2011-22

Network Verification - a timeline

Prototype
Research paper
Deployment

SecGuru
DC ACLs &
Migration of Edge ACLs

NoD
Network
Optimized
Datalog
For generic
reachability in
virtual networks

Symmetries and surgeries
All-pairs
reachability
in DCs

RCDC v1
Reachability **C**hecker
Azure **D**ata **C**enters

NSG & UDR v1
Reachability for
customer Vnets
NLS v1

Jitu
Forms
Azure
NWV
Team

NSG v2
Catch live
misconfigs

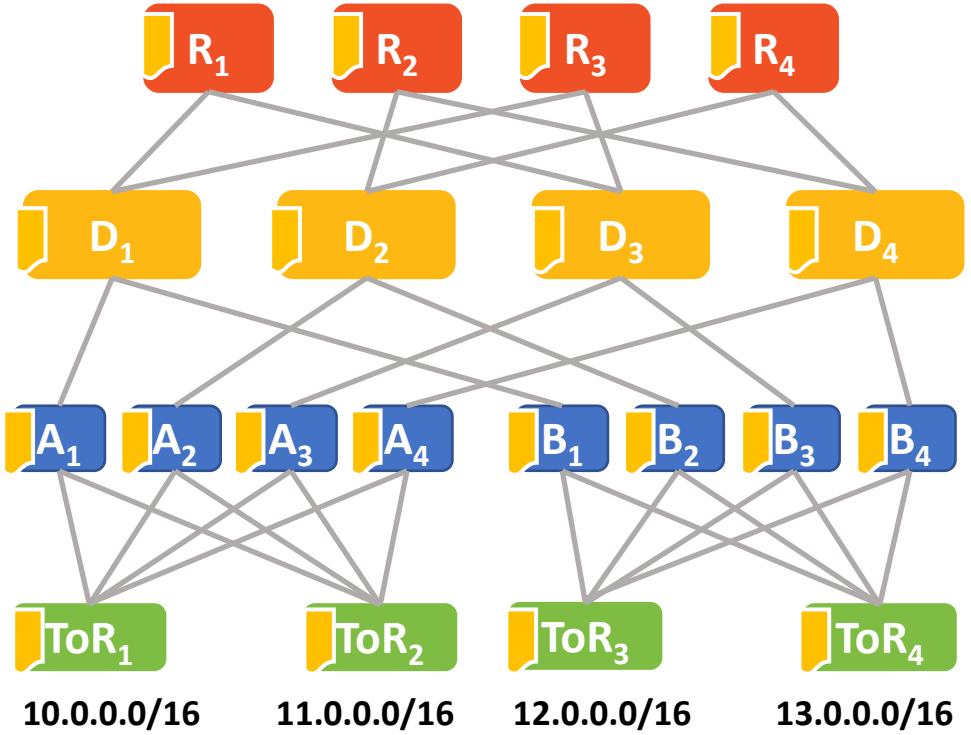
RCDC v2
Fully deployed
NCVS **Network Change Verification System**
Spock, Zen, NLS, ONE

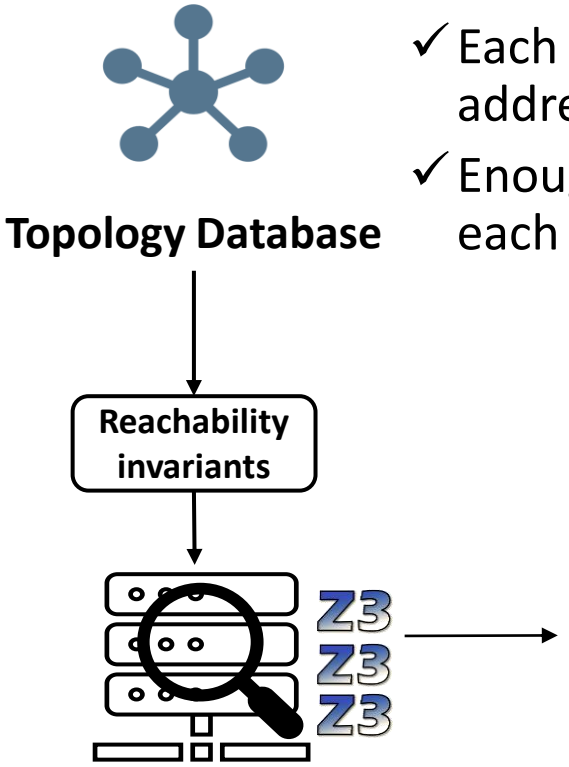
VNetVerifier v1
Reachability for
customer VNets

VNetVerifier v2
Azure Virtual
Network Manager

Live Monitoring of Forwarding Behavior

Global reachability as **local contracts**





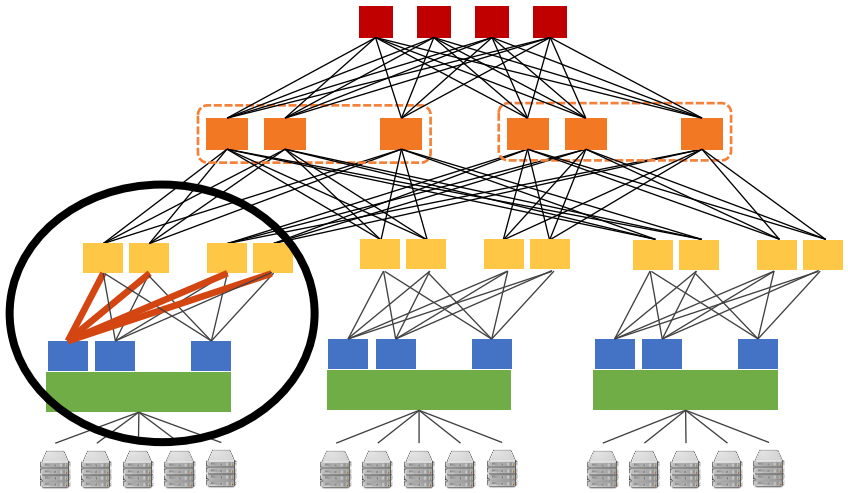
- ✓ Each router has a fixed rule for a set of addresses
- ✓ Enough to verify rule is enforced on each router

Error Reports

5 Billion Z3 queries per day

[Jayaraman et al, Sigcomm 2019]

Spock + Zen: programming local checks for each device



Correct local forwarding

```
[SpockClassInitialize]
public void Initialize(Network n) {
  this.ComputeExpectedNextHops(n.Topology)
}

[SpockTestMethod(scope: Device)]
public void LocalNextHopsTest(Device d) {
  foreach (var (prefix, nexthops) in this.expectedNextHops[d]) {
    if (prefix == "0.0.0.0/0") {
      var rule = d.Fib.GetEntry(prefix);
      Assert.AreNotEqual(rule, null);
      Assert.AreEqual(nexthops, rule.Interfaces);
    } else {
      Assert.AllPackets(
         If(ContainedBy(prefix),
         NextHops(d, hops => nexthops.SetEquals(hops))));
    }
  }
}
```

Initialization

Precompute expected next hops
For each device in the network

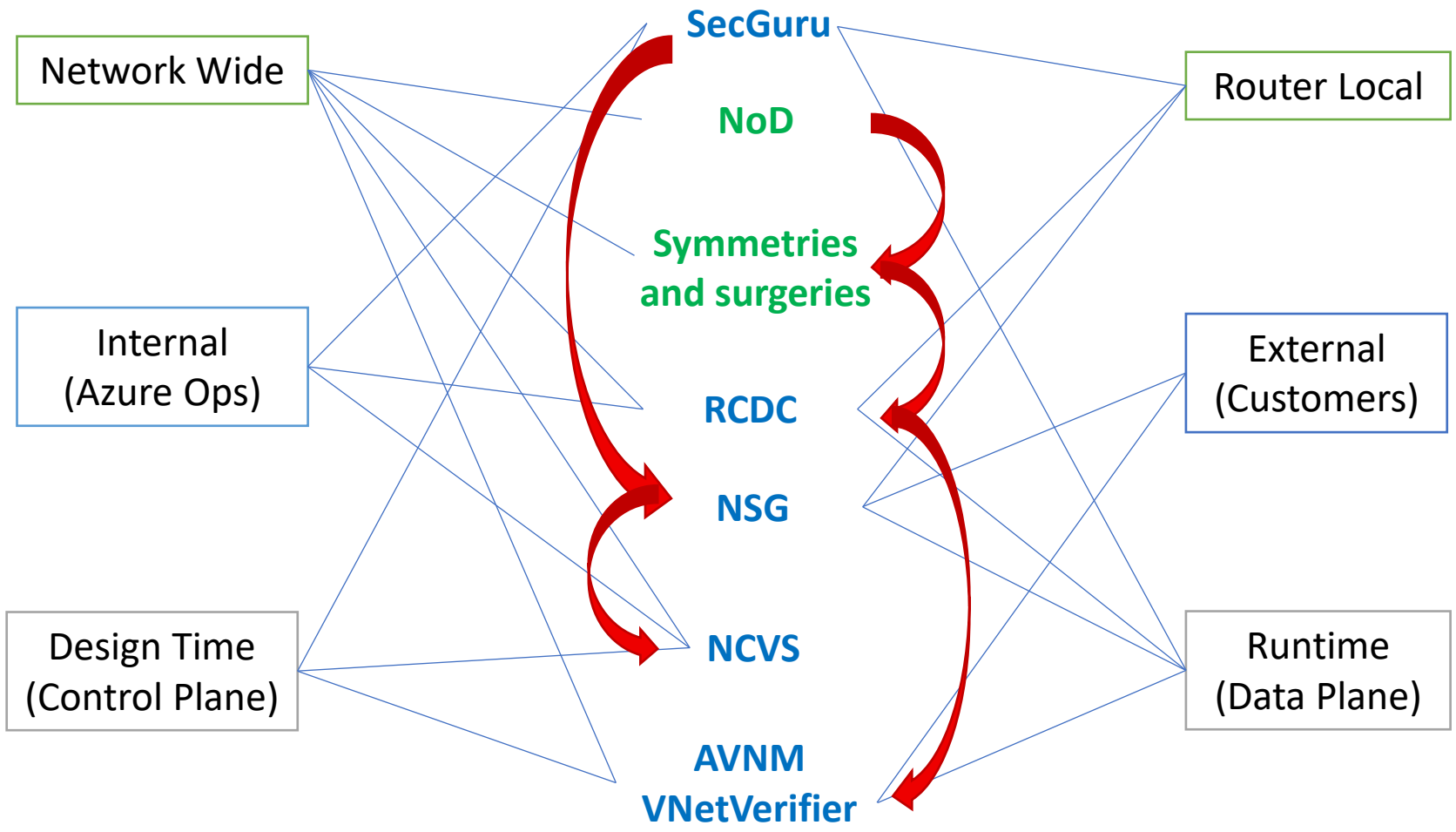
Local scope

Check each device separately, in parallel

Verification

Check that all possible packets are forwarded correctly.

Network Verification – Scope and Targets



Enablers –Automated Reasoning Engines

Leonardo de Moura

Z3 Efficient Symbolic Solver

Text and Programmatic API
Formalism support datatypes used in software

Efficient search algorithms
+
Solvers tuned for datatypes used in software

(optimal) Solution | Infeasible Core | Consequences

Zen

C# reflection:
Code and Configurations ⇒ Constraints
BDDs, Z3

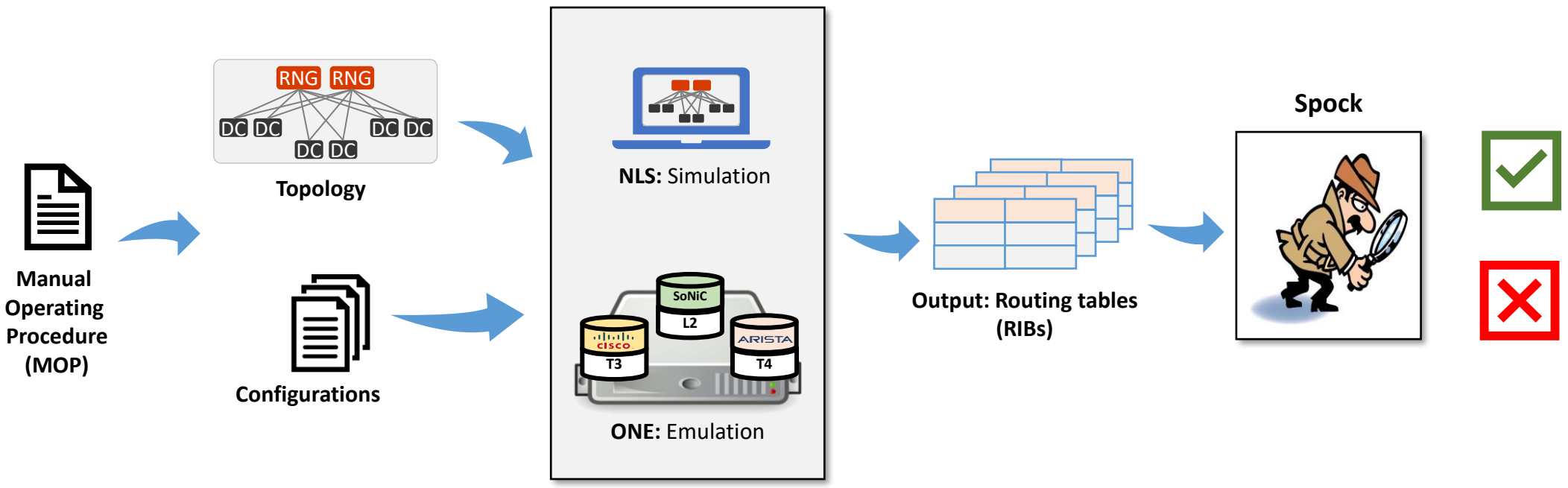
ONE

Virtual Machines
Emulation
Fidelity at Level of Router Software

NLS

Network Logic Solver
Simulation
Fidelity at Level of Standards **and** Firmware

NCVS: Network Change Verification System

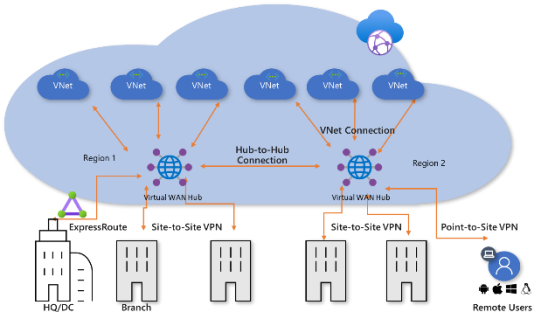


100s of migrations verified
Dozens of Sev 1 & 2 outages *prevented*

ZenGuru - Virtual Network Verifier internals

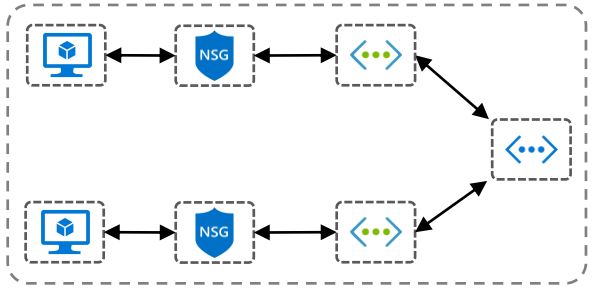
Virtual deployment

pulled via public REST APIs



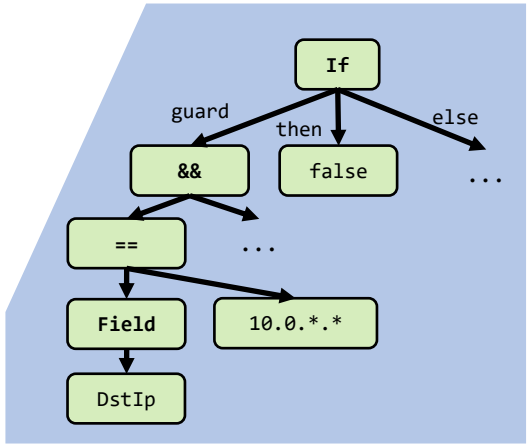
Abstract network graph

represents topology connectivity



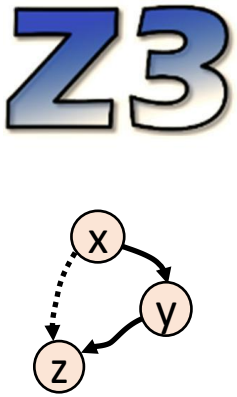
Virtual policy IR

captures policy semantics



Backend solver

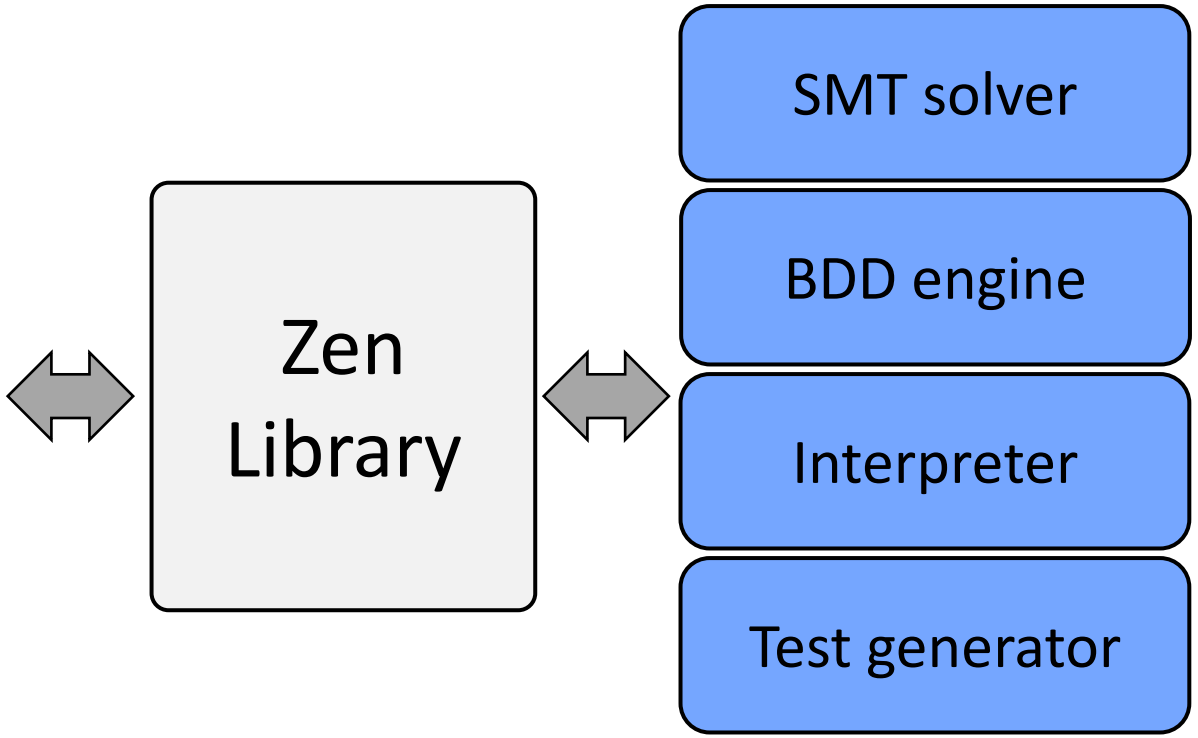
SMT, **BDDs**, etc.



Backends

Zen: an intermediate policy representation

```
Zen<bool> Allow(Nsg nsg, Zen<Packet> pkt, int i) {
    if (i >= nsg.Rules.Length)
        return false;
    var rule = nsg.Rules[i];
    return If(Matches(rule, pkt),
        rule.Permit,
        Allow(nsg, pkt, i+1));
}
```



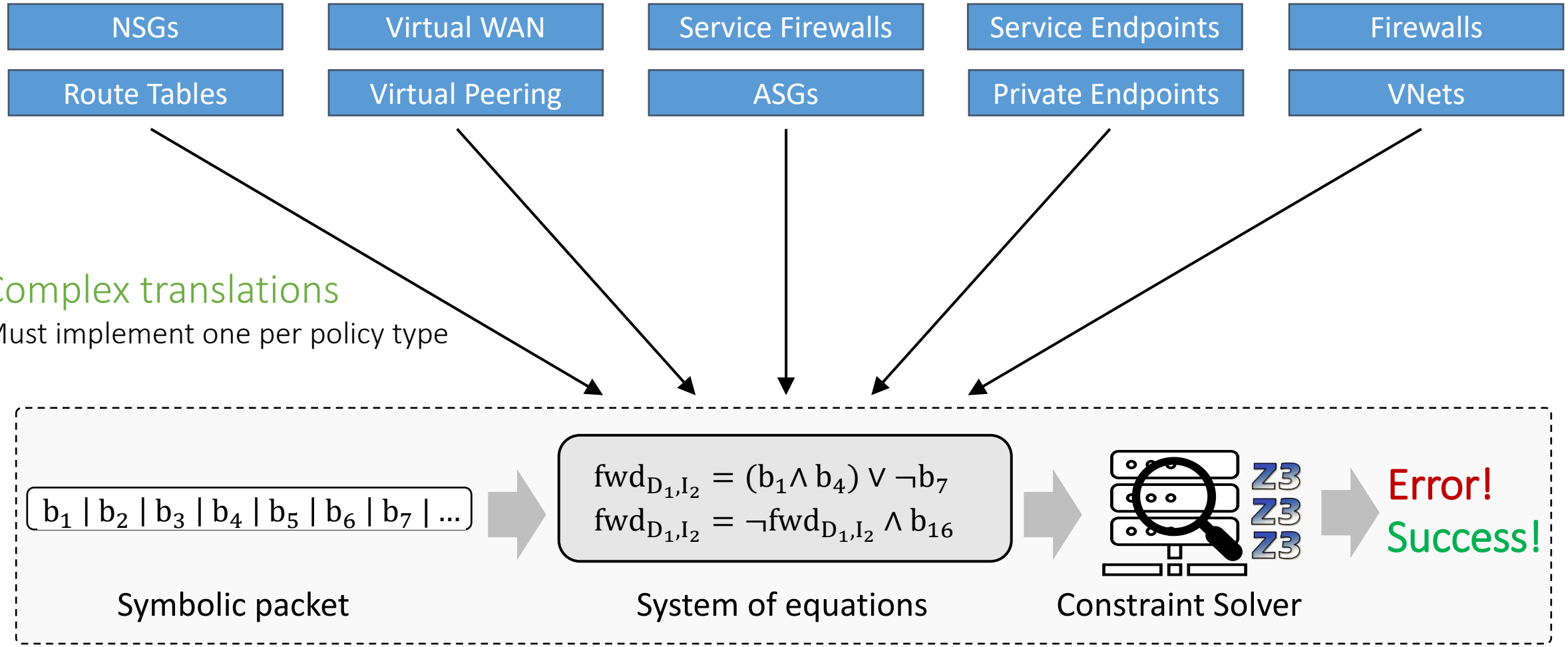
Embedded Domain-Specific Language in C# [1]

https://github.com/microsoft/zen

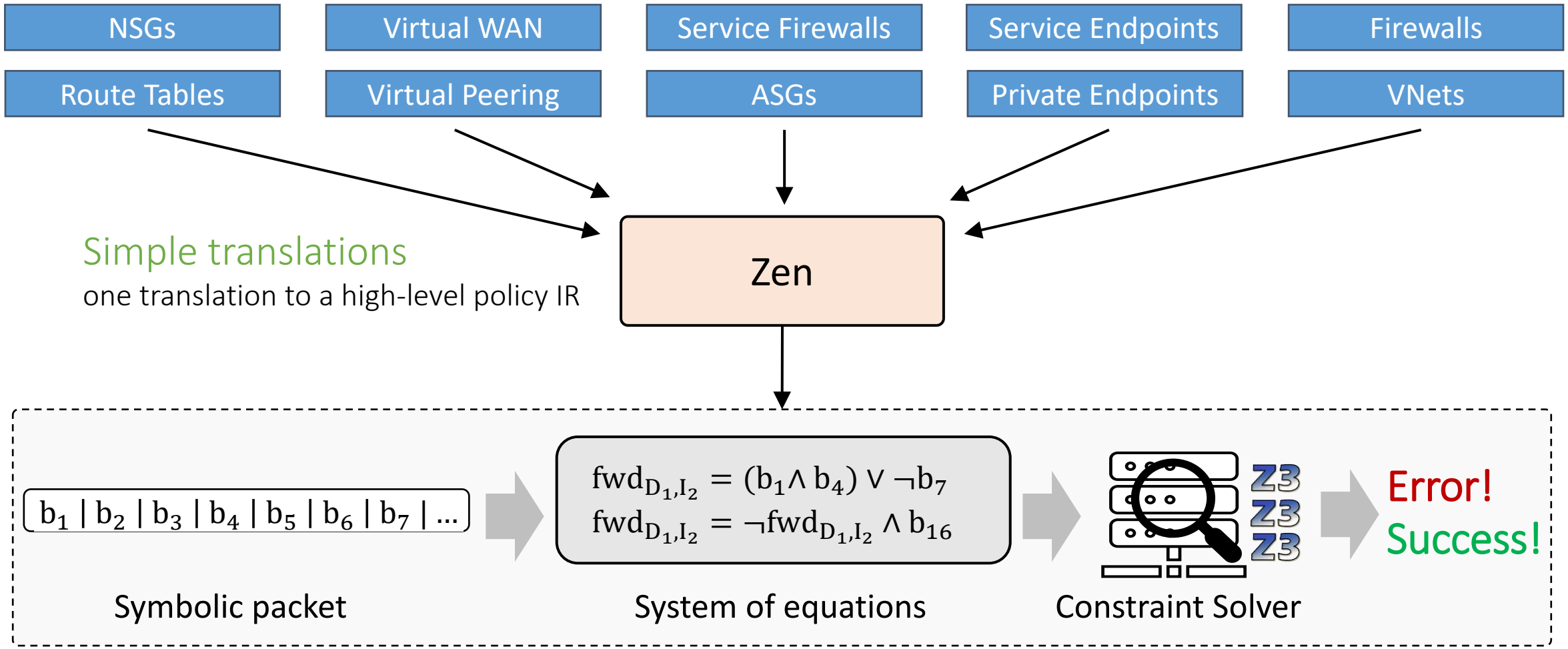
Backends

[1] A General Framework for Compositional Network Modeling, R. Beckett, R. Mahajan, HotNets 2020

Zen: an intermediate modeling language



Zen: an intermediate modeling language



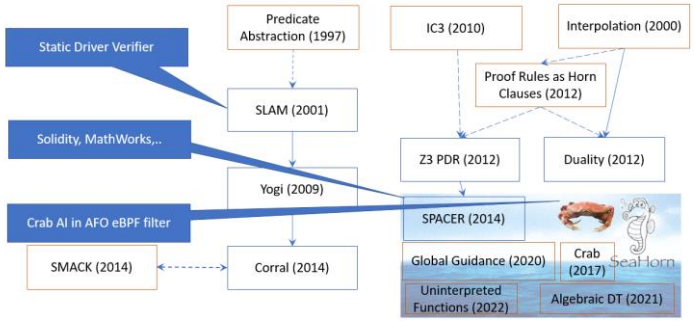
Microsoft

Confluences

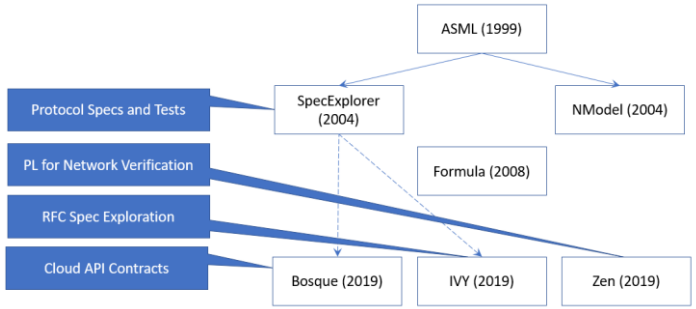
SDN	Programming Languages
Symmetries in Datacenter Networks	Bi-simulation and Congruences
IP header space vs. Forwarding Equivalence Classes	Hash-tries, Header Space Algebras and BDDs
Modularity of Forwarding Tables	Floyd-Hoare and Rely/Guarantee Proof Rules
BGP Synchronization	Abstract Dijkstra and A*
Capturing Semantics of Policies	Reflection, Meta-programming

Summary: Logic, Tomography and Networks

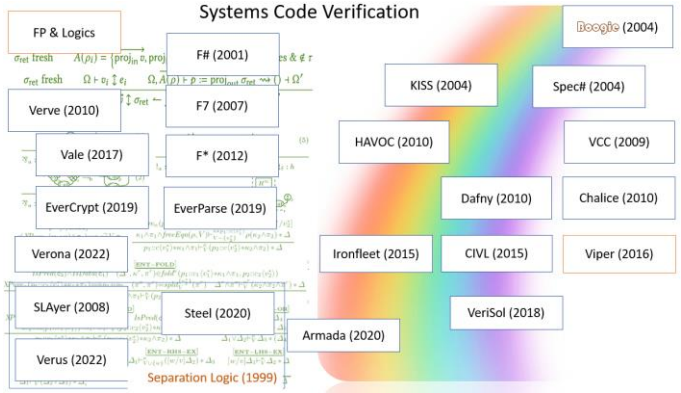
Symbolic Model Checking



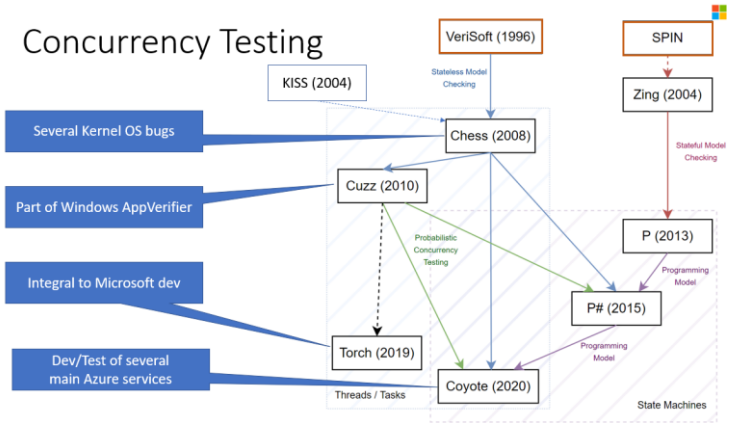
Model Based Testing and Model Programs



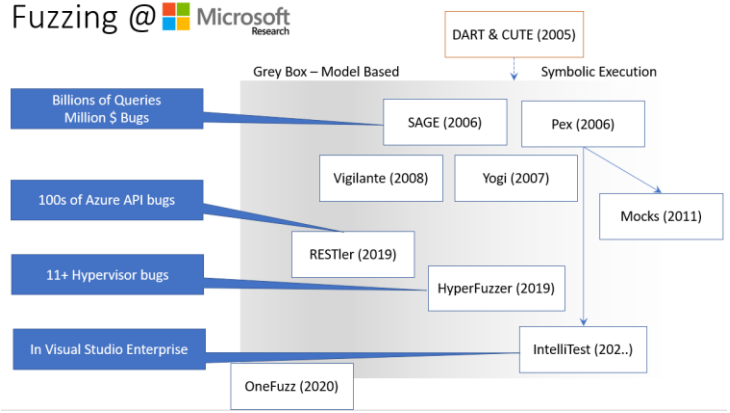
Systems Code Verification



Concurrency Testing



Fuzzing @ Microsoft Research



NCVS: Network Change Verification System