Formal Methods and Tools for Distributed Systems

Thomas Ball

Microsoft

http://research.microsoft.com/~tball

Outline

- 20 Years at Microsoft (1999-present)
- The great work of others at Microsoft

20 Years at Microsoft

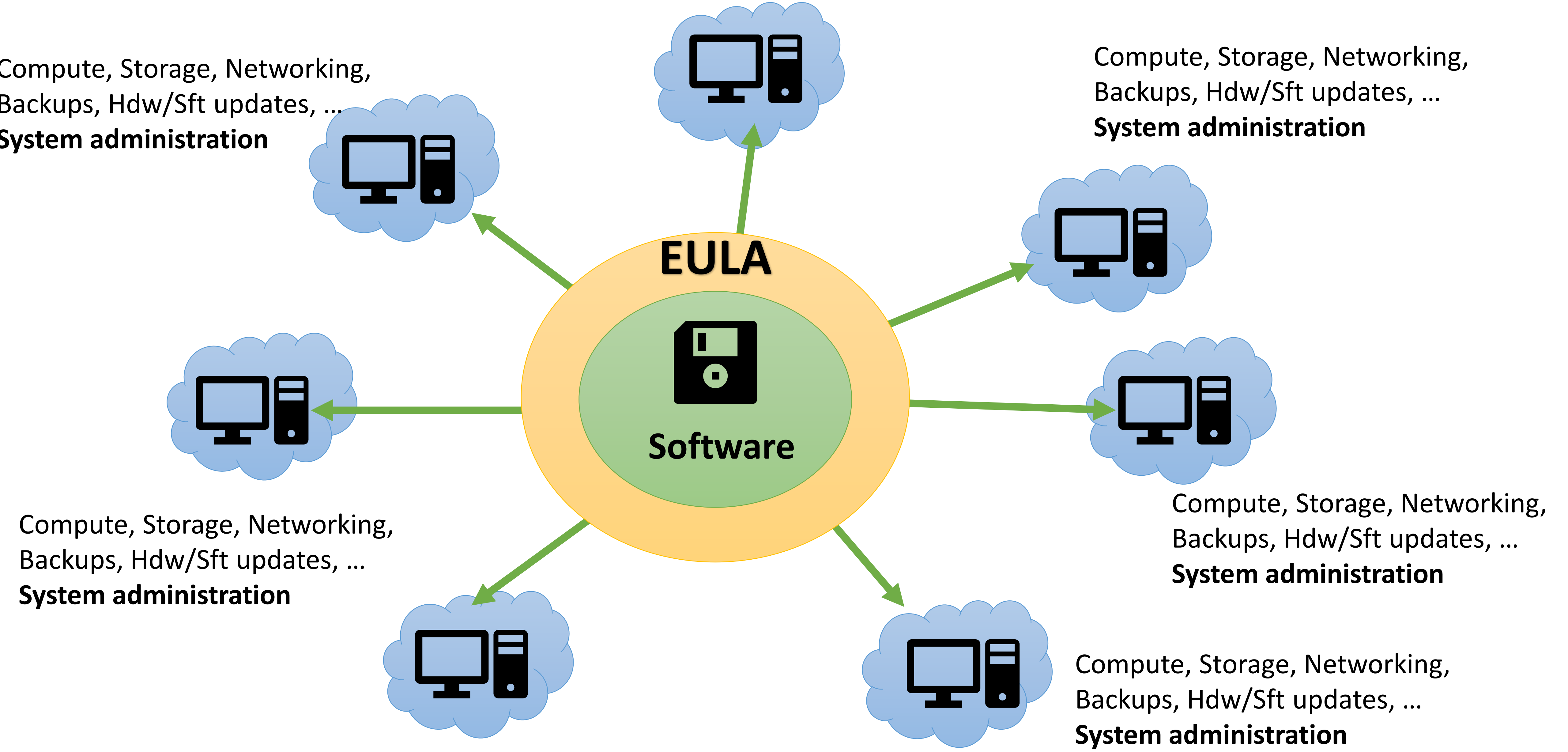
From EULA to SLA

From Bugs and Bounties to Cyberweapons

From Spec to Spec+Check

From Closed to Open

From EULA (1) to SLA



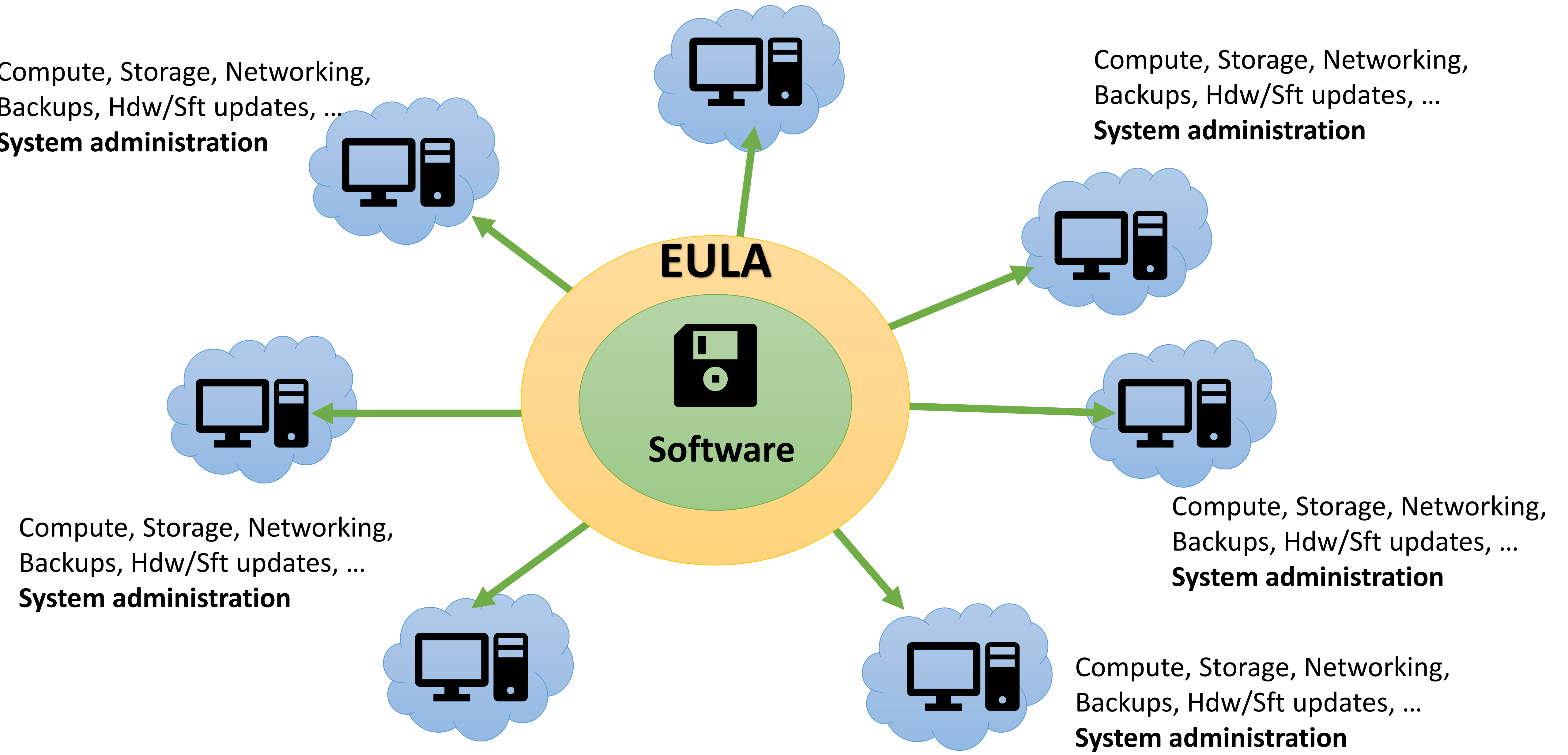
The GPL

- 11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. **THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU**. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
- 12. **IN NO EVENT** UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING **WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM** AS PERMITTED ABOVE, **BE LIABLE TO YOU FOR DAMAGES**, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES **ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM** (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

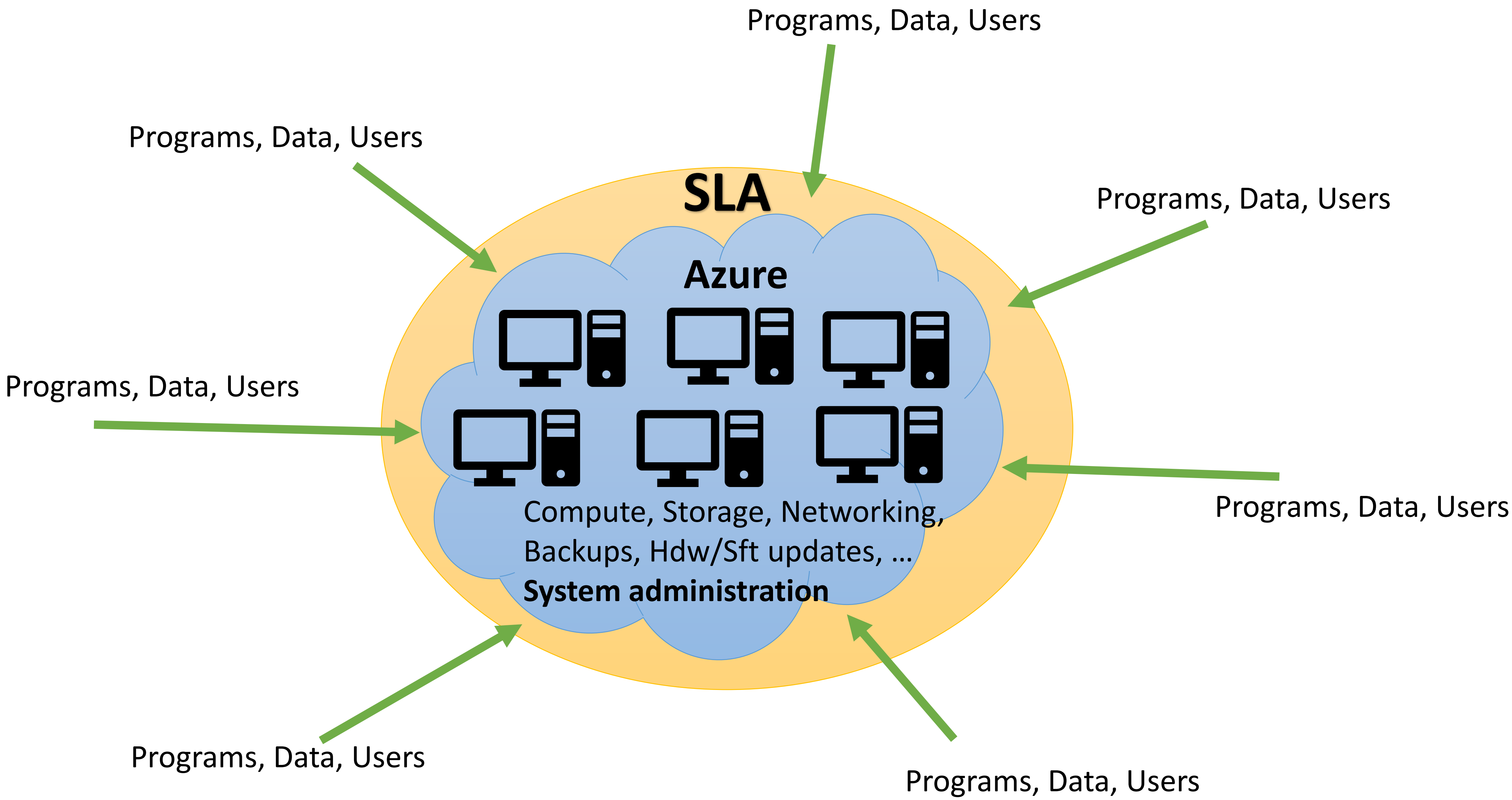
Microso

- **11**. EXCLUSIC
CERTAIN OT
PERMITTED
MICROSOFT
SPECIAL, IN
DAMAGES W
DAMAGES FC
INFORMATIC
INJURY, FOR
DUTY INCLU
FOR NEGLIG
OTHER LOSS
RELATED T
SOFTWARE
PROVIDE SU
CONNECTIO
THE EVENT
STRICT LIAB
WARRANTY
MICROSOFT
POSSIBILITY

From EULA (1) to SLA



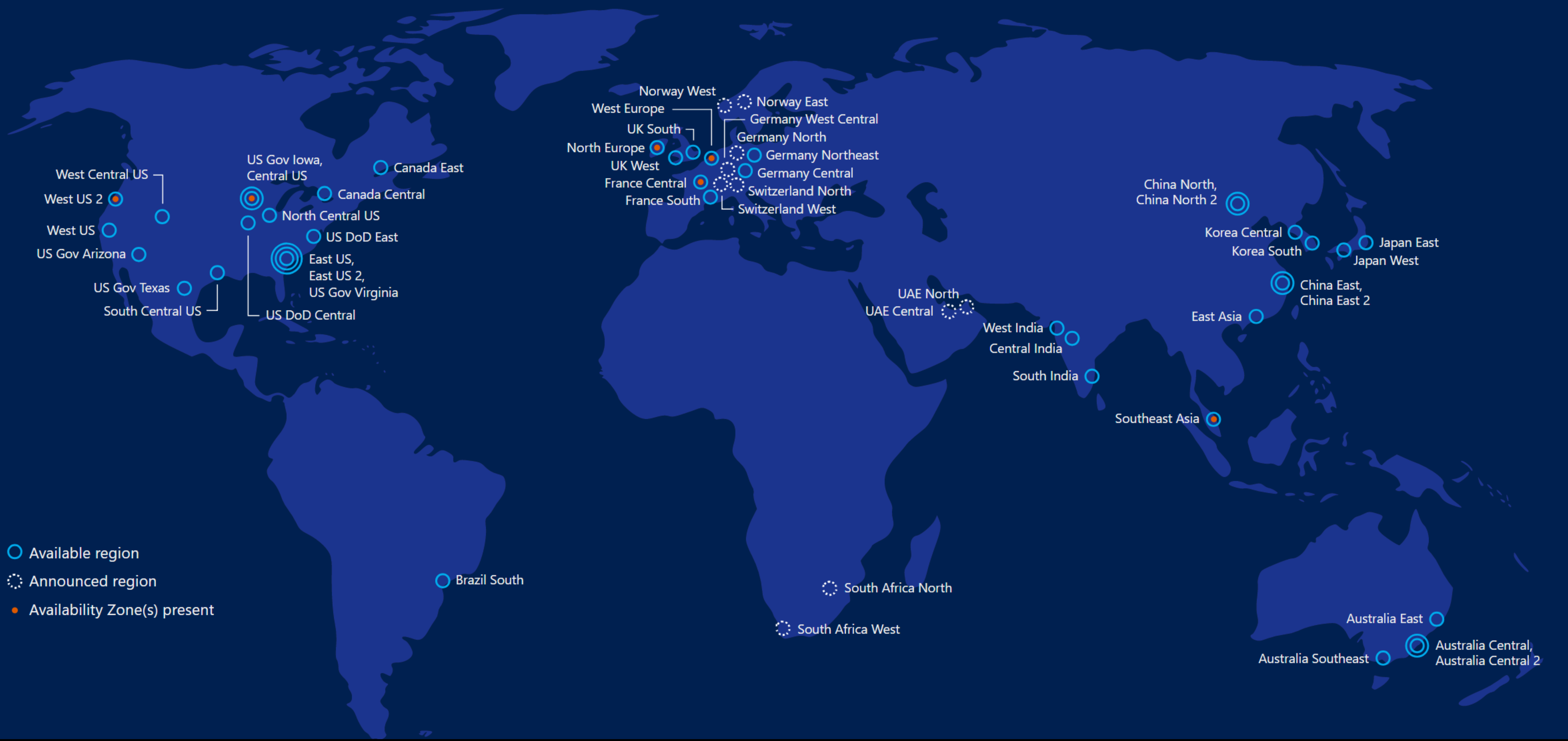
From EULA to SLA (2)




CLOUD SCALE

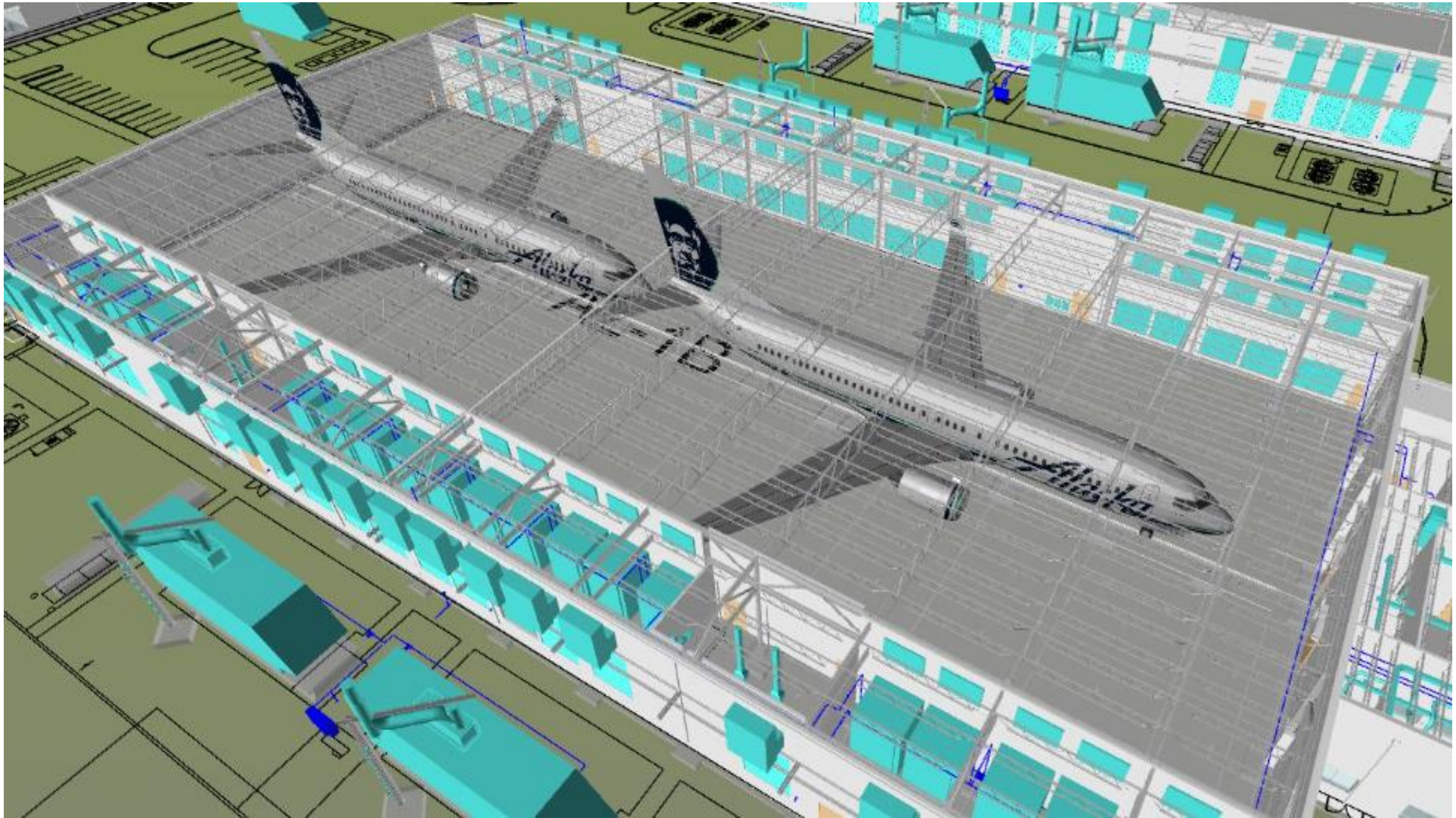
54 regions worldwide

140 available in 140 countries



Cloud Scale..

Cloud Scale….



Service Level Agreement (SLA)

"For all Virtual Machines that have two or more instances deployed in the same Availability Set, we guarantee you will have Virtual Machine Connectivity to at least one instance at least 99.95% of the time."

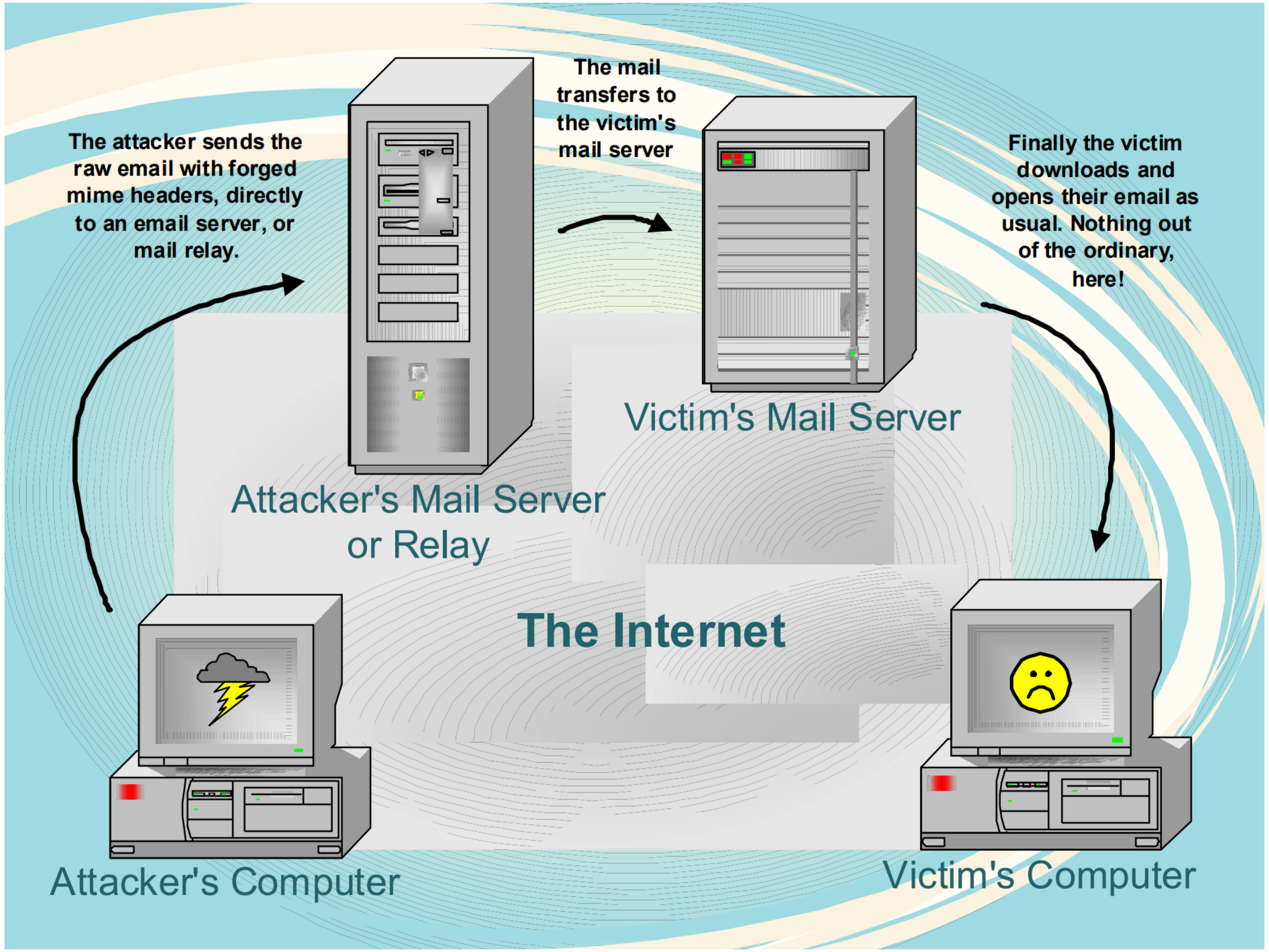
MONTHLY UPTIME PERCENTAGE	SERVICE CREDIT
< 99.95%	10%
< 99%	25%
< 95%	100%

https://azure.microsoft.com/support/legal/sla/virtual-machines/v1_8/

From Bugs and Bounties to Cyberweapons

Bugs… because there are so many more ways for things to go wrong than there are for them to go right.

Bugs (2001): Nimda



https://en.wikipedia.org/wiki/Nimda

https://www.zdnet.com/article/nimda-rampage-starts-to-slow/

https://www.cnet.com/news/microsoft-attempts-to-allay-security-fears/

https://digitalguardian.com/about/security-change-agents/code-red-and-nimda-worms

https://pen-testing.sans.org/resources/papers/gcih/automated-execution-arbitrary-code-forged-mime-headers-microsoft-inte

2002

Bill Gates' Trustworthy Computing Memo

Availability: Our products should always be available when our customers need them. System outages should become a thing of the past because of a software architecture that supports redundancy and automatic recovery. …

Security: The data our software and services store on behalf of our customers should be protected from harm and used or modified only in appropriate ways. …

Privacy: Users should be in control of how their data is used. Policies for information use should be clear to the user. Users should be in control of when and if they receive information to make best use of their time. …

https://www.wired.com/2002/01/bill-gates-trustworthy-computing/

SDL Timeline

The perfect storm (2000–2002)

- Growth of home PC's
- Rise of malicious software
- Increasing privacy concerns
- Internet use expansion

SDL ramp up (2002–2005)

- Bill Gates' TwC memo
- Microsoft security push
- Microsoft SDL released
- SDL becomes mandatory policy at Microsoft
- Windows XP SP2 and Windows Server 2003 launched with security emphasis

Setting a new bar (2005–2008)

- Windows Vista and Office 2007 fully integrate the SDL
- SDL released to public
- Data Execution Prevention (DEP) & Address Space Layout Randomization (ASLR) introduced as features
- Threat Modeling Tool

Collaboration (2008–2011)

- Microsoft joins SAFECode
- Microsoft Establish SDL Pro Network
- Defense Information Systems Agency (DISA) & National Institution Standards and Technology (NIST) specify featured in the SDL
- Microsoft collaborates with Adobe and Cisco on SDL practices
- SDL revised under the Creative Commons License

Selective tooling and Automation (2011–2018+)

- Additional resources dedicated to address projected growth in Mobile app downloads
- Industry-wide acceptance of practices aligned with SDL
- Adaption of SDL to new technologies and changes in the threat landscape
- Increased industry resources to enable global secure development adoption

2000 — 2001 — 2002 — 2003 — 2004 — 2005 — 2006 — 2007 — 2008 — 2009 — 2010 — 2011 — 2018+ →

https://www.microsoft.com/en-us/securityengineering/sdl/about

Bugs (2014): OpenSSL

"These produce wrong results. The first example does so only on 32 bit, the other three also on 64 bit."

"I believe this affects both the SSE2 and AVX2 code. It does seem to be dependent on this input pattern."

"I'm probably going to write something to generate random inputs and stress all your other poly1305 code paths against a reference implementation."

```
Poly1305 functions of openssl.

These produce wrong results. The first example d
the other three also on 64 bit.
```

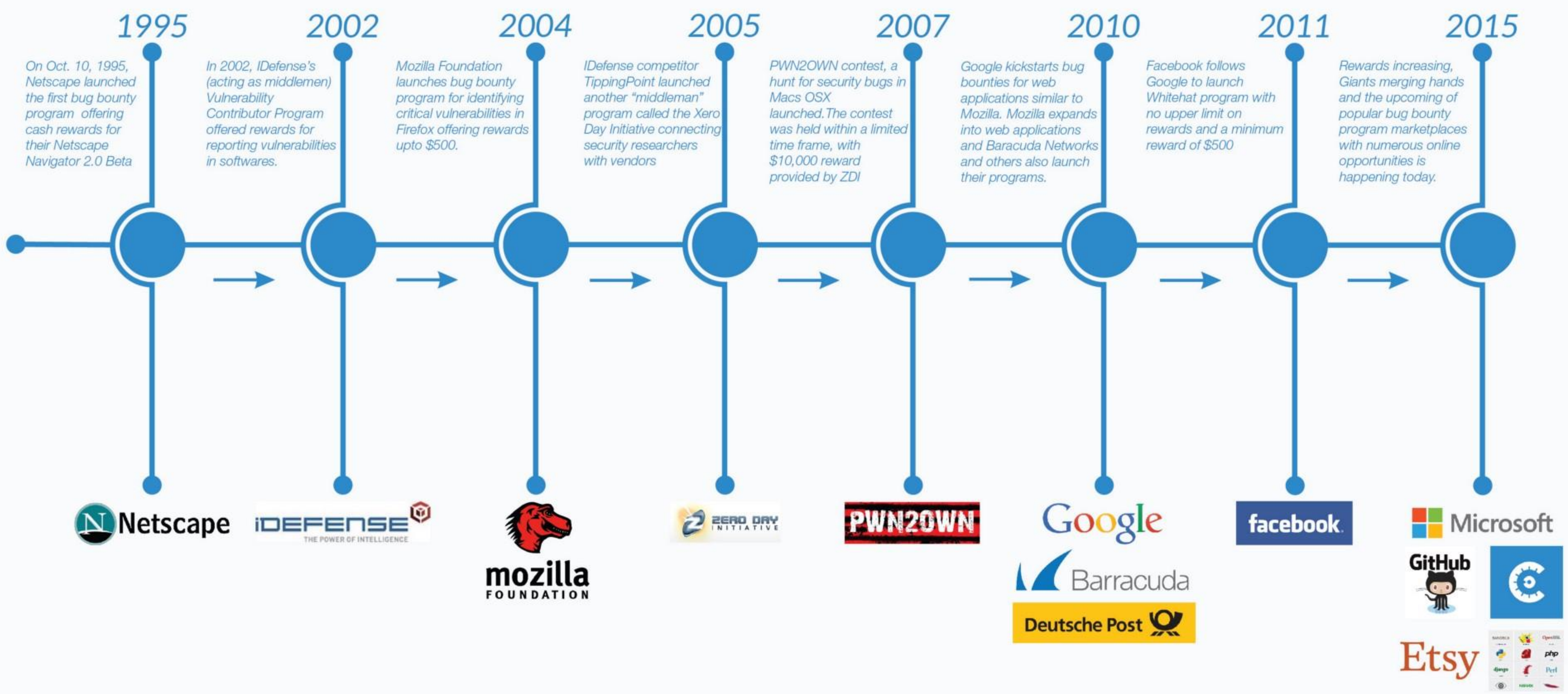
```
Hi folks,

You know the drill. See the attached poly1305_test2.c.

$ OPENSSL_ia32cap=0 ./poly1305_test2
PASS
$ ./poly1305_test2
Poly1305 test failed.
got:      2637408fe03086ea73f971e3425e2820
expected: 2637408fe13086ea73f971e3425e2820

I believe this affects both the SSE2 and AVX2 code. It does seem to be
dependent on this input pattern.

This was found because a run of our SSL tests happened to find a
problematic input. I've trimmed it down to the first block where they
```

The Impact of One Bug

"The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet."

http://heartbleed.com/

Bounties



https://blog.cobalt.io/the-history-of-bug-bounty-programs-50def4dcaab3

Cyberweapons

“**Stuxnet** is a malicious computer worm, first uncovered in 2010. Thought to have been in development since at least 2005, Stuxnet targets SCADA systems and is believed to be responsible for causing substantial damage to Iran's nuclear program.”

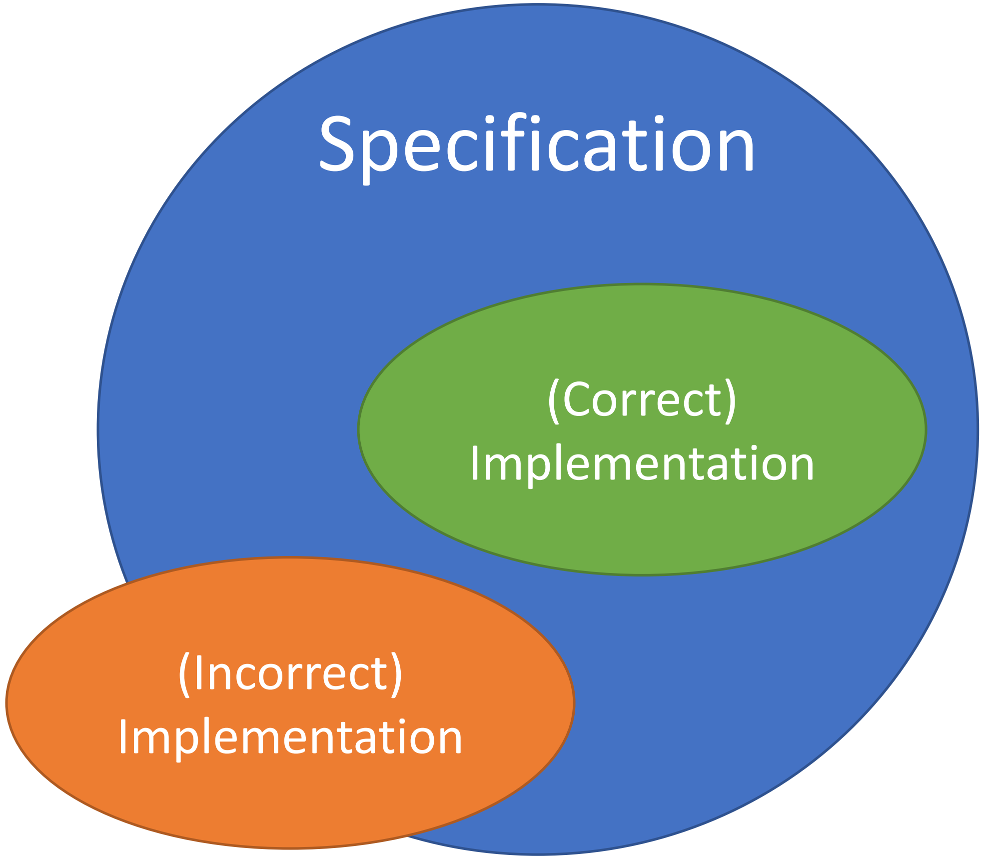
“Stuxnet attacked Windows systems using an unprecedented four zero-day attacks (…)… The number of zero-day exploits used is unusual, as they are highly valued and malware creators do not typically make use of (and thus simultaneously make visible) four different zero-day exploits in the same worm.”

https://en.wikipedia.org/wiki/Stuxnet

From Spec to Spec+Check

Formal Methods

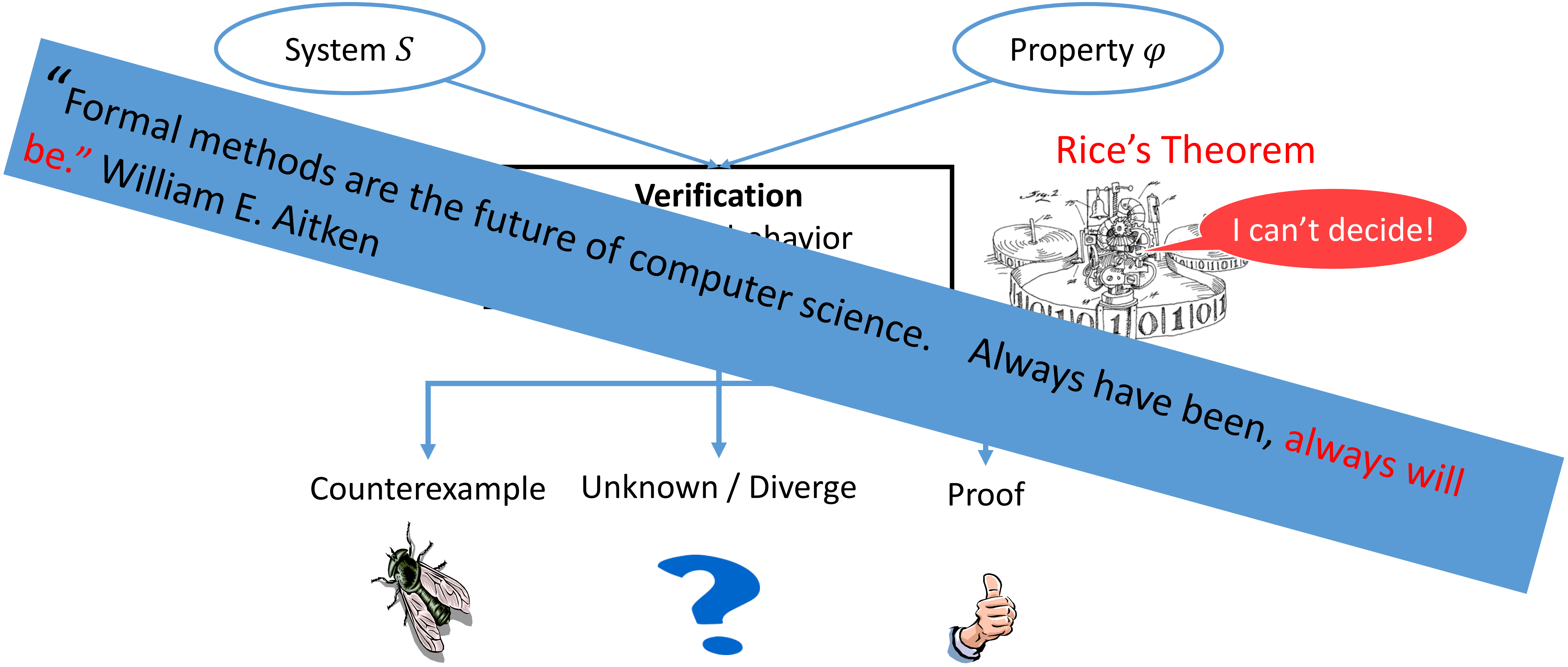
- Mathematical/logical specification of desired (correct) behavior
- Automated/interactive checking of implementation against specification



Correctness Properties

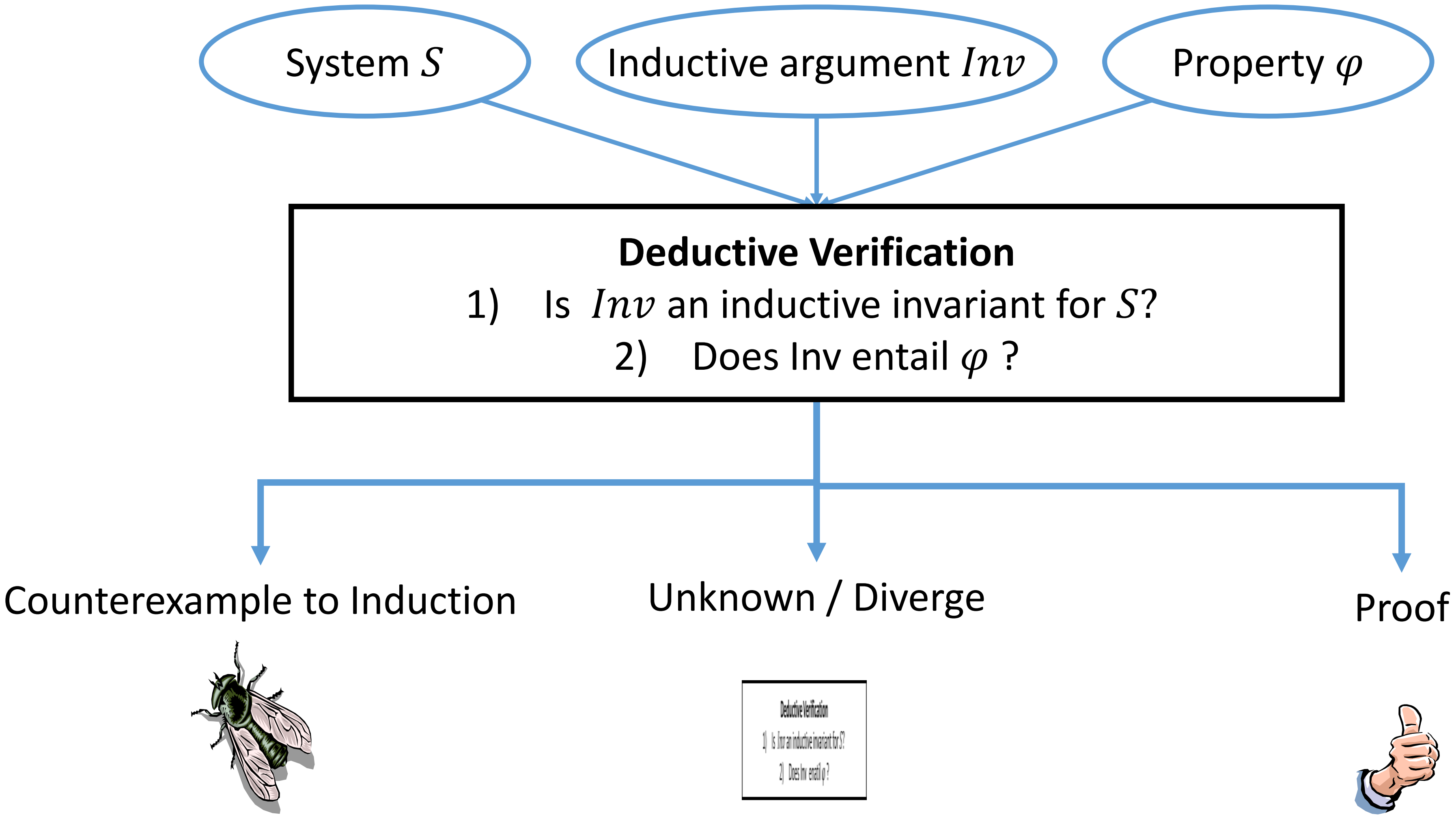
- Memory safety
- No buffer overruns
- Functional correctness
- Termination
- Minimize side-channel leaks
- Cryptographic security
- …

Automatic verification of infinite-state systems



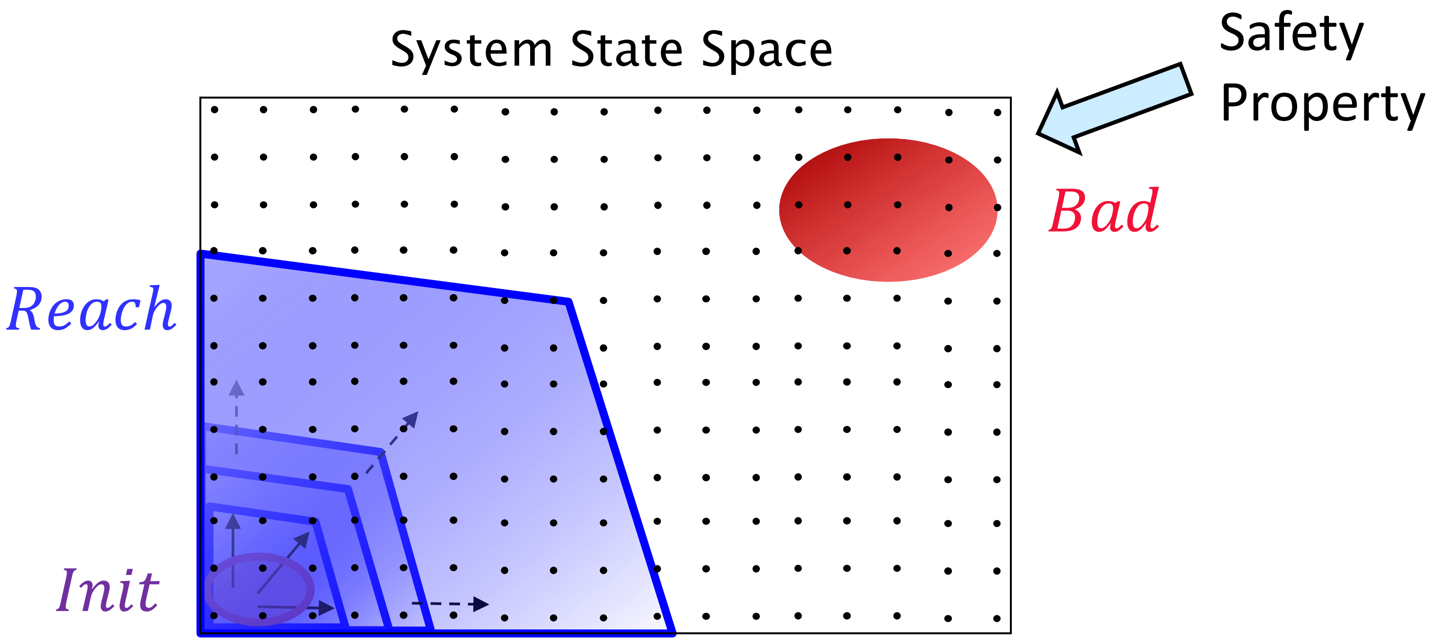
Slide from Mooly Sagiv

Deductive verification



Slide from Mooly Sagiv

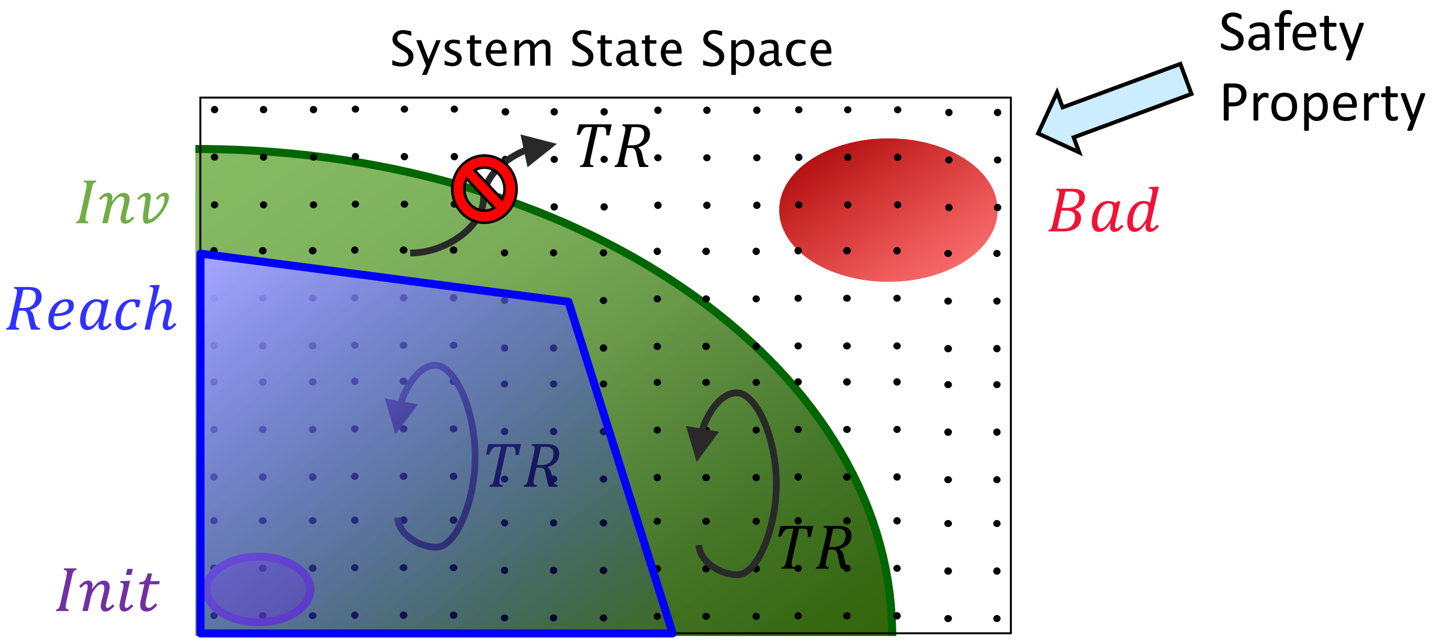
Inductive invariants



System S is **safe** if all the reachable states satisfy the property $\varphi = \neg Bad$

Slide from Mooly Sagiv

Inductive invariants



System S is **safe** if all the reachable states satisfy the property $\varphi = \neg Bad$

System S is safe iff there exists an **inductive invariant** Inv :

$Init \subseteq Inv$ (**Initiation**)

if $\sigma \in Inv$ and $\sigma \rightarrow \sigma'$ then $\sigma' \in Inv$ (**Consecution**)

$Inv \cap Bad = \emptyset$ (**Safety**)

Slide from Mooly Sagiv

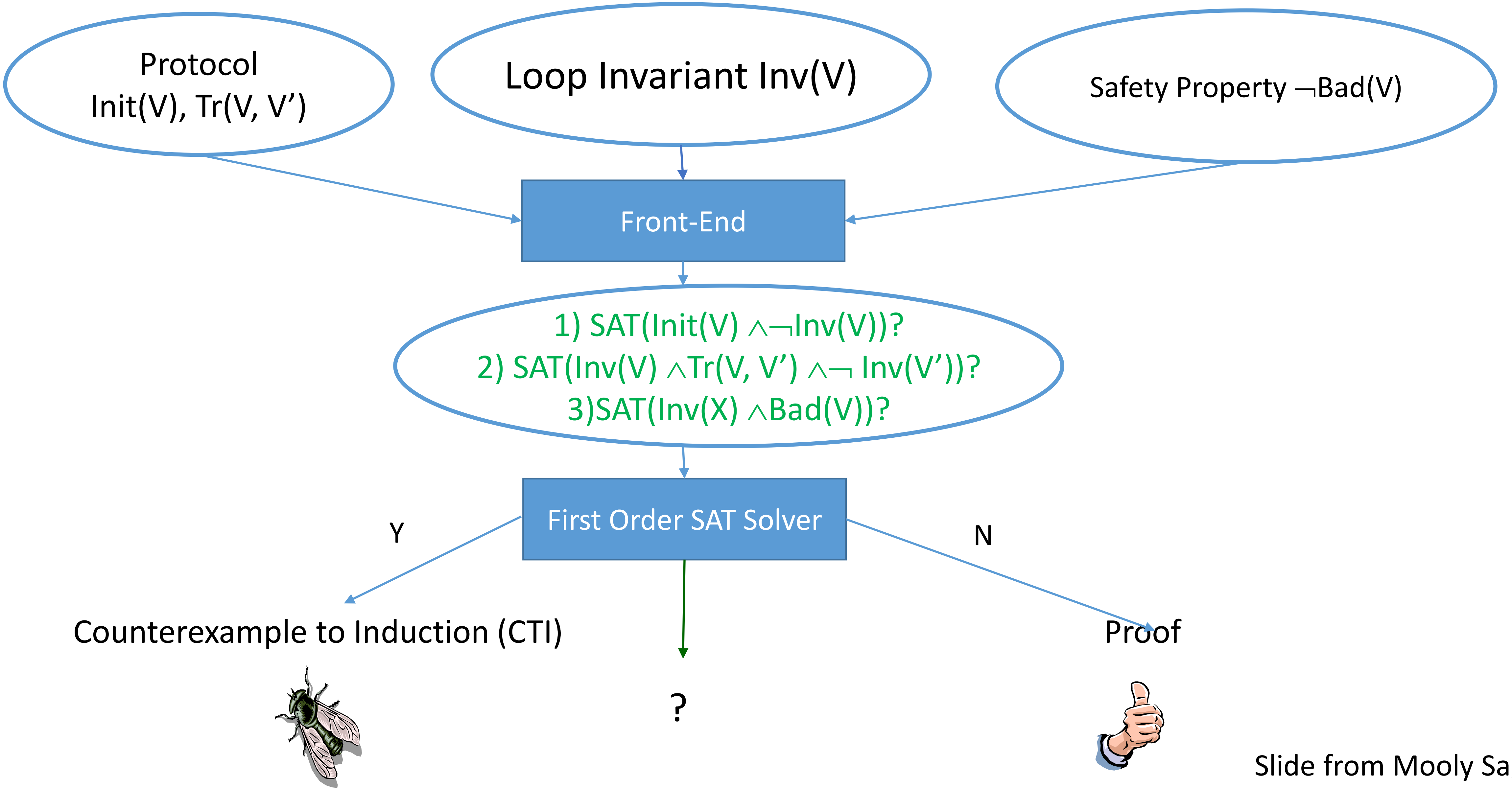
Logic-based deductive verification

- Represent $Init$, $\rightarrow$, Bad, Inv by logical formulas
  - Formula ⇔ Set of states

- Automated solvers for logical satisfiability made huge progress
  - Propositional logic (SAT) – industrial impact for hardware verification
  - First-order theorem provers
  - Satisfiability modulo theories (SMT) – major trend in software verification

Slide from Mooly Sagiv

Deductive verification by reductions to First Order Logic



Slide from Mooly Sagiv

Automated Theorem Prover

Open Source (MIT License)

https://github.com/z3prover/z3

https://rise4fun.com/Z3/tutorial

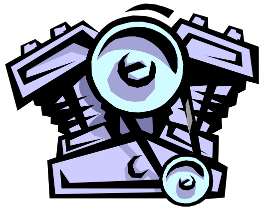
Leonardo de Moura, Nikolaj Bjorner,
Christoph Wintersteiger, …

Boolean Algebra

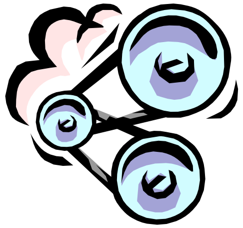
Bit Vectors

Linear Arithmetic

Floating Point



Z3 reasons over
a combination of theories



First-order Axiomitizations

Non-linear, Reals

Sets/Maps/…

Algebraic Data Types

Reduction to Logic

```
int Puzzle(int x)
{
    int res = x;
    res = res + (res << 10);
    res = res ^ (res >> 6);
    if (x > 0 && res == x + 1)
        throw new Exception("bug");
    return res;
}
```

```
(declare-const x (_ BitVec 32))
(assert (bvsgt x #x00000000))
(assert (= (bvadd x #x00000001)
  (bvxor (bvadd x (bvshl x #x0000000A))
         (bvashr (bvadd x (bvshl x #x0000000A)) #x00000006))))
(check-sat)
(get-model)
```

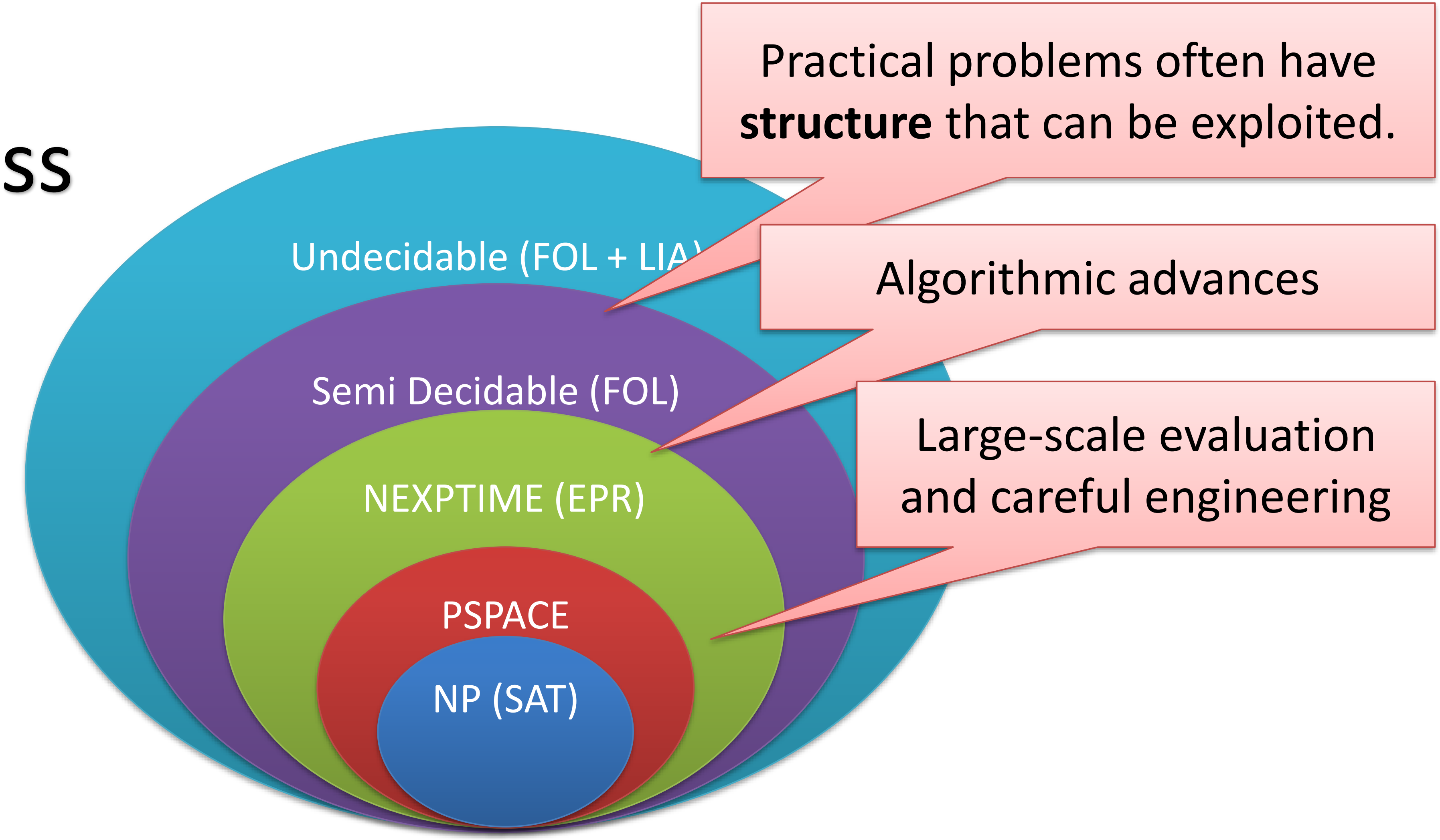
x = 389306474

https://rise4fun.com/Z3/n6ZB6

Logic/Complexity Classes

Greater Expressiveness

Greater Automation



Symbolic Analysis Tools

SecGuru

FORMULA
Modeling Foundations.

Vcc

Yogi

SLAM

SAGE

Pex

TERMINATOR

SLAYER

HAVOC

BOOGIE

SLS, floats

νZ: Opt+MaxSMT

μZ: Datalog

Generalized PDR

Existential Reals

Model Constructing SAT

CutSAT: Linear Integer Formulas

Quantified Bit-Vectors

Linear Quantifier Elimination

Model Based Quantifier Instantiation

Generalized, Efficient Array Decision Procedures

Engineering DPLL(T) + Saturation

Effectively Propositional Logic

Model-based Theory Combination

Relevancy Propagation

Efficient E-matching for SMT solvers

Z3 Internals

Formal Methods: Substantial Progress

Better Tools

- Automated + Interactive Theorem Provers
- Model Checking
- Program Analysis

Application to Real Systems

- Static Driver Verifier (Windows drivers)
- http://compcert.inria.fr/ (C compiler)
- https://sel4.systems/ (OS)
- …

From Spec to Spec+Check

Open Source: Times have changed!

“We will move to a Chromium-compatible web platform for Microsoft Edge on the desktop” https://blogs.windows.com/

- Microsoft actively contributes to and use open source
- The tools presented in this talk are open source, or have open source equivalents

20 Years at Microsoft

From EULA to SLA

From Bugs and Bounties to Cyberweapons

From Spec to Spec+Check

From Closed to Open

Formal Methods and Tools

Logic

High-level Specification
(TLA+)

Correctness of Cryptography and Protocols
(F*, Ivy, P#)

Bug Finding and Verification for C/C++
(SAGE, Corral)

Network Verification
(SecGuru)

thinking

programming

testing

verifying

Formal Methods and Tools

Logic

High-level Specification
(TLA+)

Correctness of Cryptography and Protocols
(F*, Ivy, P#)

Bug Finding and Verification for C/C++
(SAGE, Corral)

Network Verification
(SecGuru)

thinking

programming

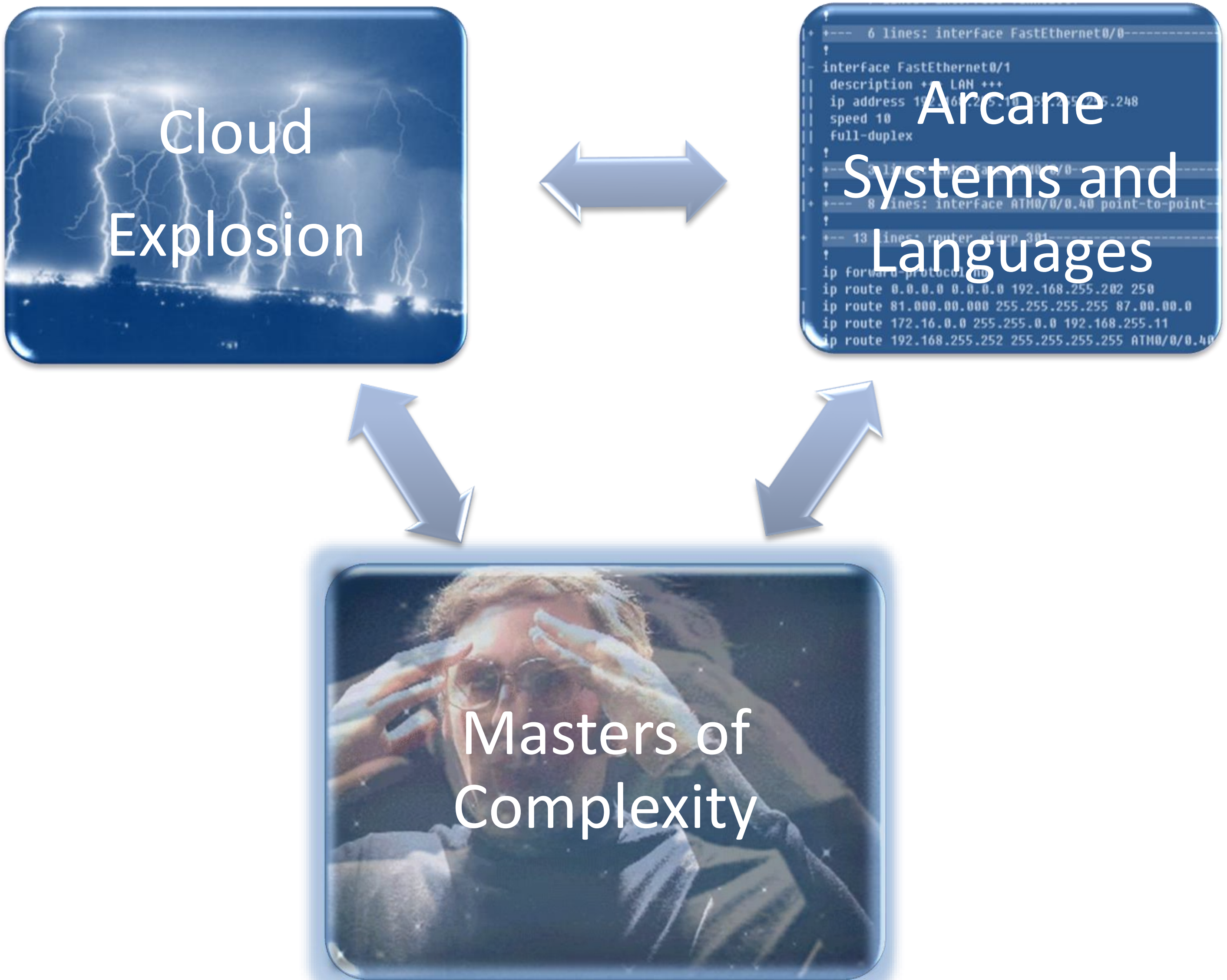
testing

verifying

SecGuru

Nikolaj Bjørner,
Karthick Jayaraman

A Cloud run by Masters of Complexity



A Cloud Harnessed by Logic/SE

Network Policies:
Complexity, Challenge and Opportunity

Several devices, vendors, formats

- Net filters
- Firewalls
- Routers

Challenge in the field

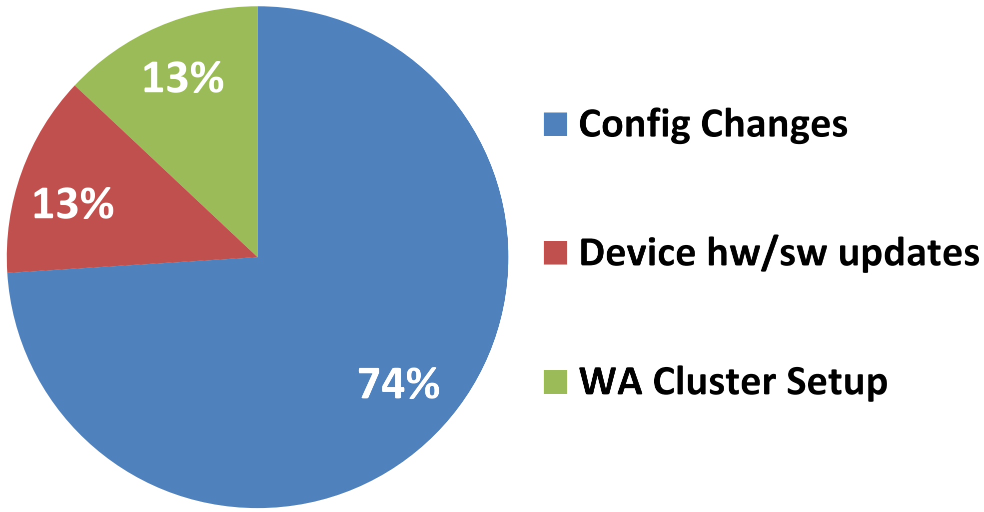
- Do devices enforce policy?
- Ripple effect of policy changes

Arcane

- Low-level configuration files
- Mostly manual effort
- Kept working by
  "Masters of Complexity"

Human errors > 4 x DOS attacks

Human Errors by Activity



Intent = Reality ?

Reality?

- Forwarding information base (FIB)
- Access Control Lists (ACL)
- Churn

Intent?

- Network Graph Service (NGS)
- Contracts derived from topology and architecture

Validation

- Continuous verification using **local validation**

Feedback

- Alerts
- Remediation

Access Control

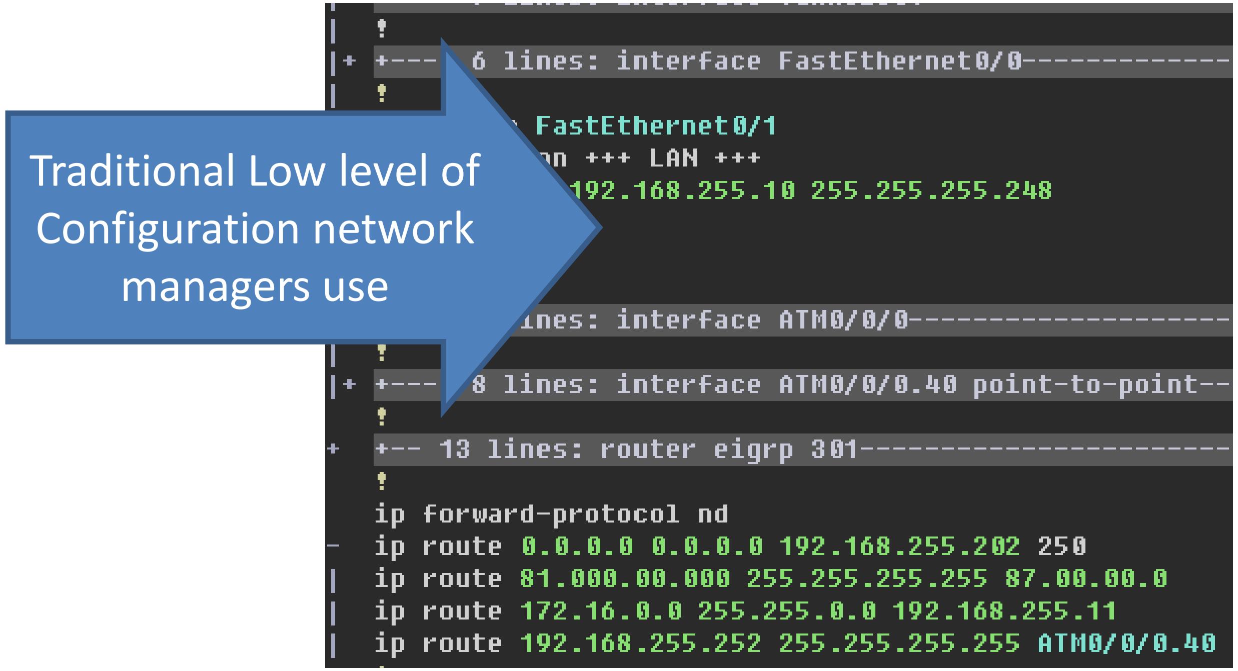
Contract:

DNS ports on DNS servers are **accessible** from tenant devices over both TCP and UDP.

Contract:

The SSH ports on management devices are **inaccessible** from tenant devices.

Policies as Logical Formulas



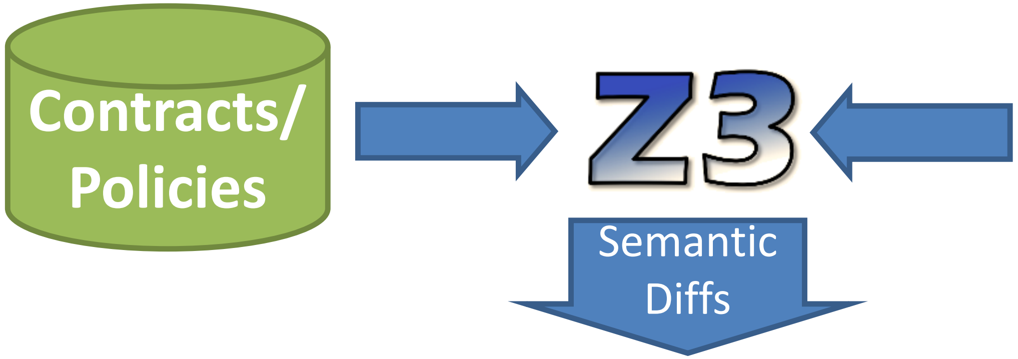
Precise Semantics as bit-vector formulas

$$Allow: \begin{array}{c}(10.20.0.0 \leq srcIp\ 10.20.31.255) \wedge \\ (157.55.252.0 \leq dstIp \leq 157.55.252.255) \wedge \\ (protocol = 6)\end{array}$$

$$Deny: \begin{array}{c}(65.52.244.0 \leq dstIp \leq 65.52.247.255) \wedge \\ (\text{protocol} = 4)\end{array}$$

Combining semantics

$$\left(\bigvee_i Allow_i\right) \wedge \left(\bigwedge_j \neg Deny_j\right)$$



Beyond Z3: a *new* idea to go from one violation to all violations

$$\left(\bigvee_i Allow_i\right) \wedge \left(\bigwedge_j \neg Deny_j\right)$$

Z3

$$\neg\left[\left(\bigvee_m Allow_m\right) \wedge \left(\bigwedge_n \neg Deny_n\right)\right]$$

Semantic Diffs

$$srcIp = 10.20.0.0/16, 10.22.0.0/16$$
$$dstIp = 157.55.252.000/24, 157.56.252.000/24$$
$$port = 80, 443$$

Representing solutions

- $2 * 2^{16} * 2 * 2^{8} * 2 = 2^{27}$ single solutions, or
- 8 products of contiguous ranges, or
- A single product of ranges

SecGuru contains optimized algorithm for turning single solutions into all (product of ranges)

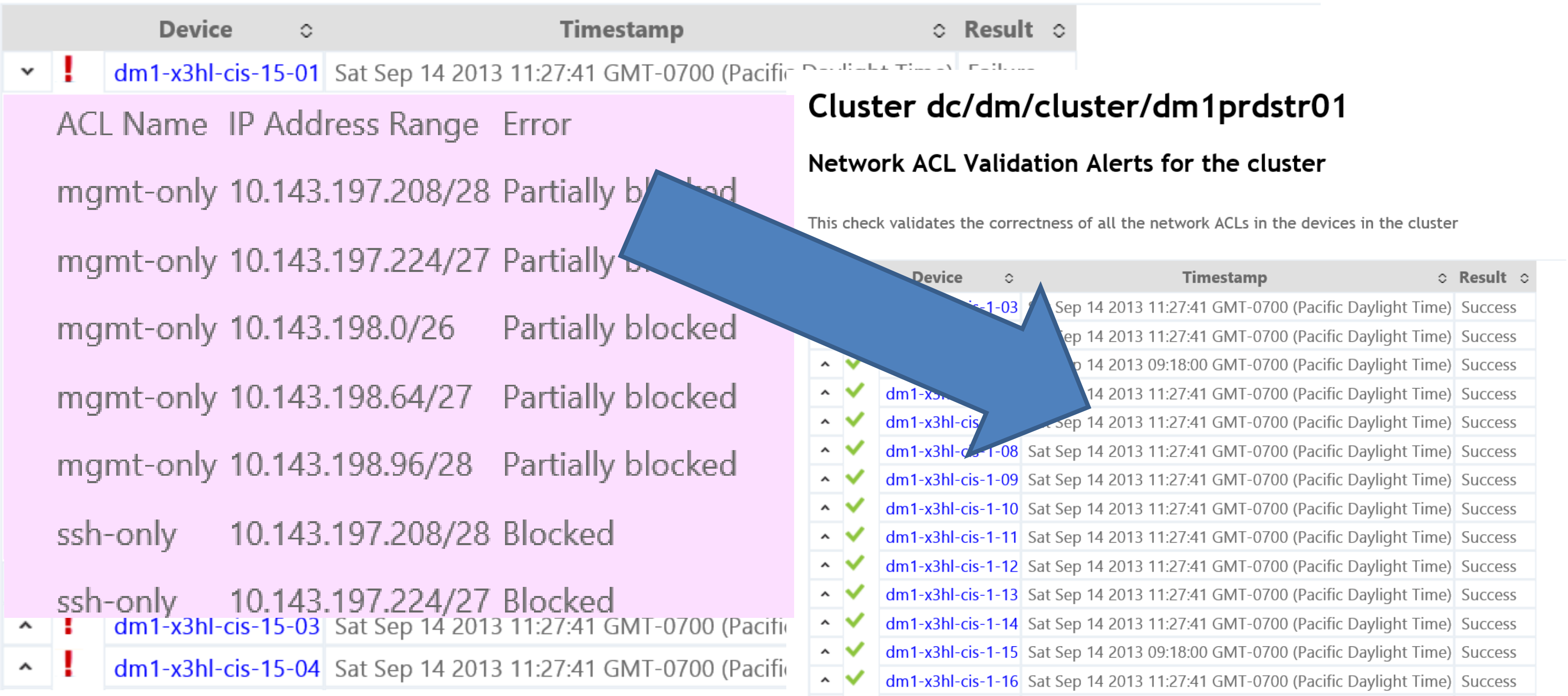
SecGuru in WANetmon

40,000 ACL checks per month
Each check 50-200ms
20 bugs/month (mostly for build-out)

Cluster dc/dm/cluster/dm1prdstr08

Network ACL Validation Alerts for the cluster

This check validates the correctness of all the network ACLs in the devices in the cluster



Self-contained Windows Firewall Checker

GitHub, Inc. [US] | https://github.com/Z3Prover/firewallchecker

Two minimal tab-separated example firewall rule files are as follows (see Examples directory):

Firewall 1:

```
Name    Enabled Action  Local Port      Remote Address  Remote Port     Protocol
Foo1    Yes     Allow   100     10.3.141.0      100     UDP
Bar1    Yes     Allow   200     10.3.141.0      200     TCP
```

Firewall 2:

```
Name    Enabled Action  Local Port      Remote Address  Remote Port     Protocol
Foo2    Yes     Allow   100     10.3.141.0      100     UDP
Bar2    Yes     Allow   200     10.3.141.1      200     TCP
```

This generates the following output from `FirewallEquivalenceCheckerCmd.exe`:

```
Microsoft.FirewallEquivalenceCheckerCmd.exe --firewall1 .\firewall1.txt --firewall2 .\firewall2.txt
Parsing first firewall...
Parsing second firewall...
Running equivalence check...
Firewalls are NOT equivalent.

Inconsistently-handled packets:
-------------------------------------------------------------------------
|  PID |     Src Address | Src Port | Dest Port | Protocol | Allowed By |
-------------------------------------------------------------------------
|    0 |      10.3.141.0 |      200 |       200 |      TCP |      First |
|    1 |      10.3.141.1 |      200 |       200 |      TCP |     Second |
-------------------------------------------------------------------------

Firewall rules matching inconsistently-handled packets:
-------------------------------------------------------------------------
|  PID | Firewall | Action | Rule Name                                  |
-------------------------------------------------------------------------
|    0 |    First |  Allow | Bar1                                       |
```

By Andrew Helwer, Azure

https://github.com/Z3Prover/FirewallChecker

Formal Methods and Tools

Logic

High-level Specification
(TLA+)

Correctness of Cryptography and Protocols
(F*, Ivy, P#)

Bug Finding and Verification for C/C++
(SAGE, Corral)

Network Verification
(SecGuru)

thinking

programming

testing

verifying

Microsoft Security Risk Detection

https://www.microsoft.com/en-us/security-risk-detection/

Security Basics

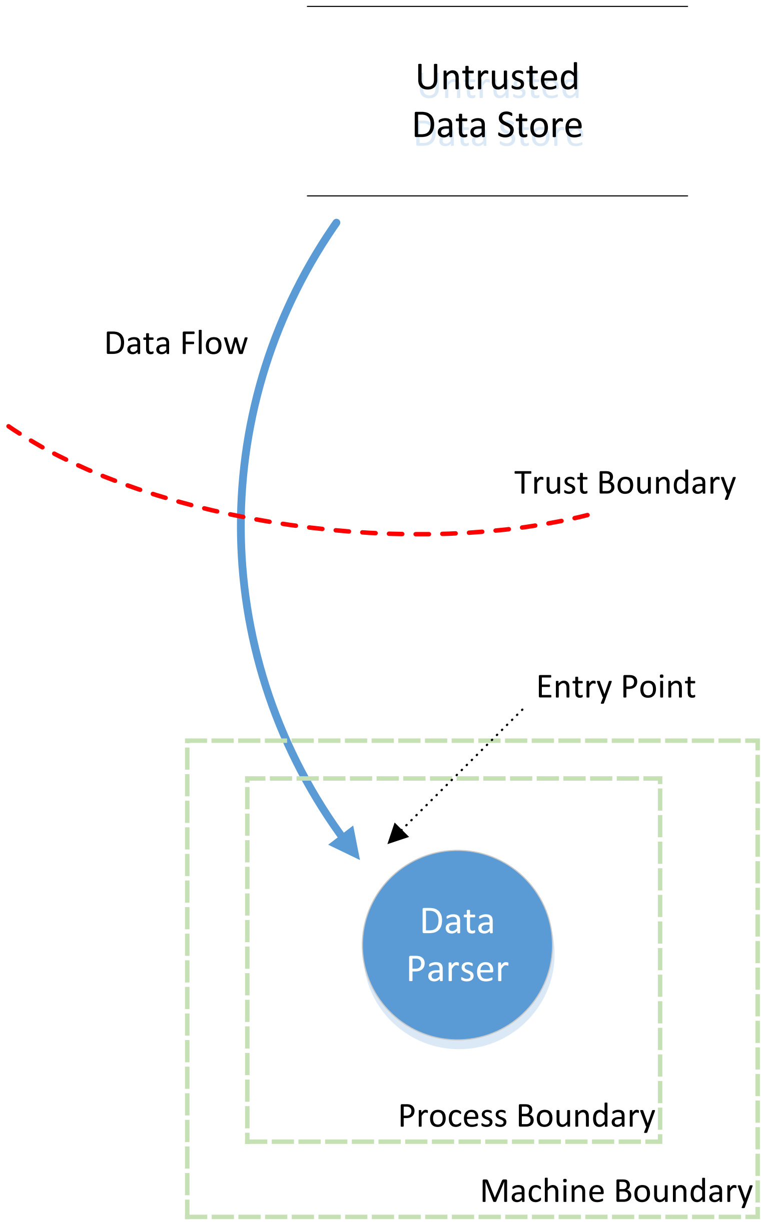
An important step in software security is identifying high-risk targets…

Dataflow, movement of bits between two network entities

Entry Point, where external data enters an entity

Trust Boundary, a dividing line across which data flows

Security Bug, any regular code or design bug



White Box Input Fuzzing

```
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 4) crash();
}
```

input = “good”

Path constraint:

I_0!='b' → I_0='b'

I_1!='a' → I_1='a'

I_2!='d' → I_2='d'

I_3!='!' → I_3='!'

Theorem prover

10+ years of
sustained investment

Z3

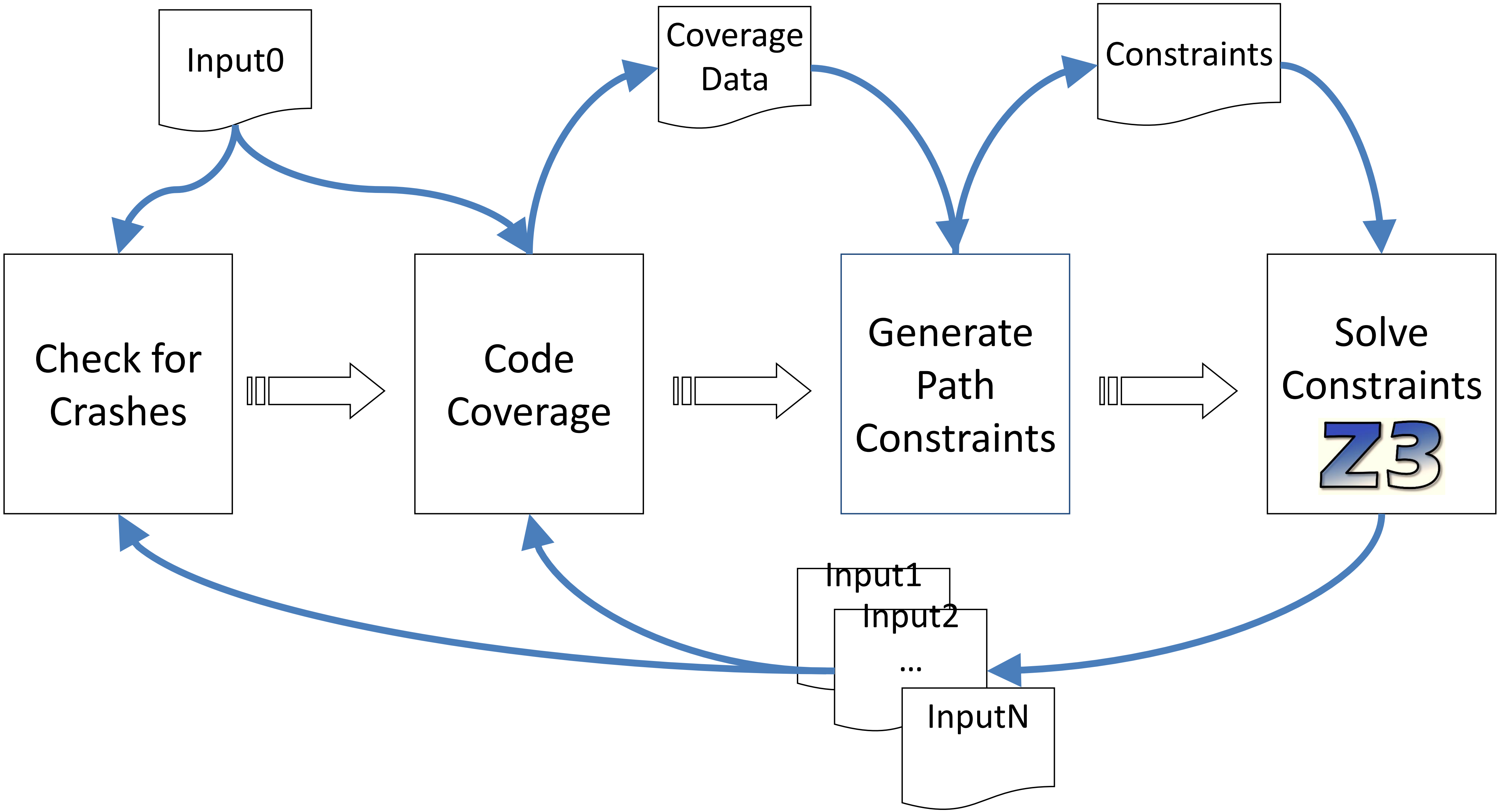
good

Gen 1 Gen 2 Gen 3 Gen 4

…

bad!

White Box Fuzzing (SAGE)



Security Risk Detection and the SDL

SAGE used internally at Microsoft to meet SDL verification requirements

White Box Fuzzing (SAGE) Results

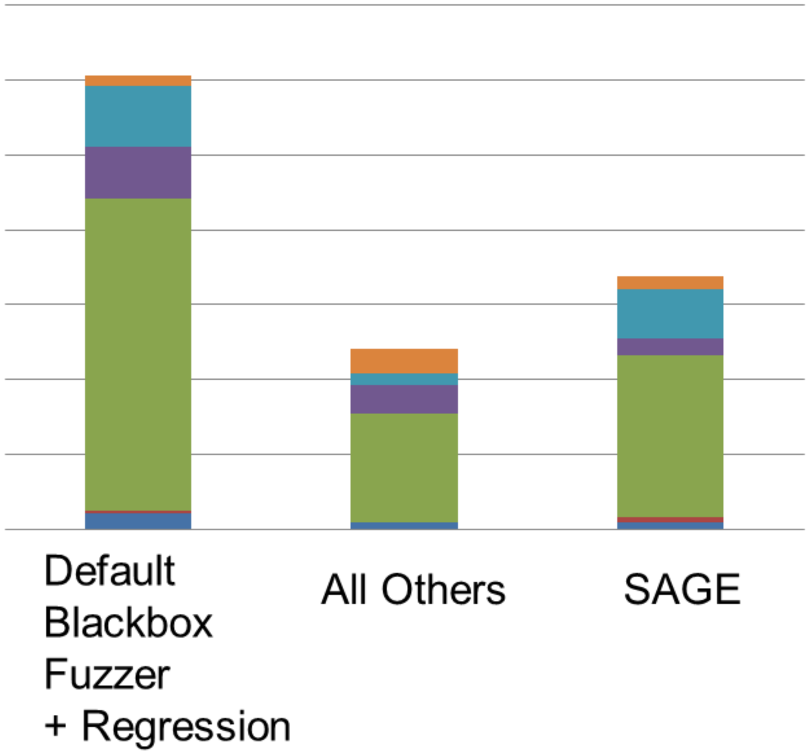
Since 2007: many new security bugs found

- Apps: decoders, media players, document processors, …
- Bugs: Write A/Vs, Read A/Vs, Crashes, …
- Many triaged as "security critical, severity 1, priority 1"

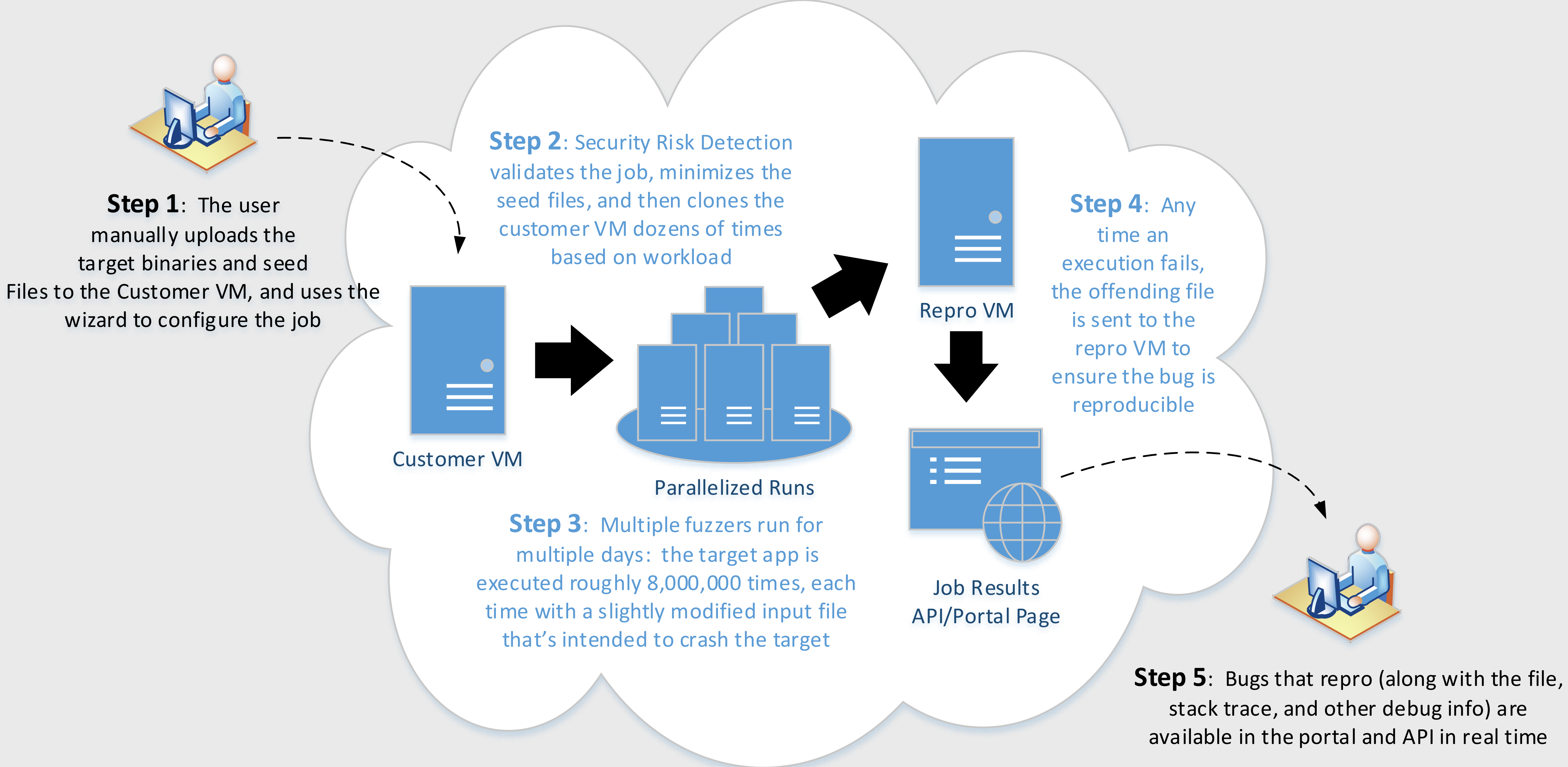
- 100s of apps, 100s of bugs
  - Bug fixes shipped quietly (no MSRCs) to 1 Billion+ PCs
  - Millions of dollars saved (for Microsoft and the world)

- "Practical Verification"
  - <5 security bulletins in SAGE-cleaned parsers since 2009

How fuzzing bugs found (2006-2009) :



Job – Cloud Workflow



More on Dynamic Symbolic Execution

For real programs, compiled through LLVM

- https://klee.github.io/

For a small subset of Python, using Z3

- https://github.com/thomasjball/PyExZ3

Hot off the press

REST-ler: Automatic Intelligent REST API Fuzzing

- Vaggelis Atlidakis, Patrice Godefroid, Marina Polishchuk
- https://arxiv.org/abs/1806.09739

Formal Methods and Tools

Logic

High-level Specification
(TLA+)

Correctness of Cryptography and Protocols
(F*, Ivy, P#)

Bug Finding and Verification for C/C++
(SAGE, Corral)

Network Verification
(SecGuru)

thinking

programming

testing

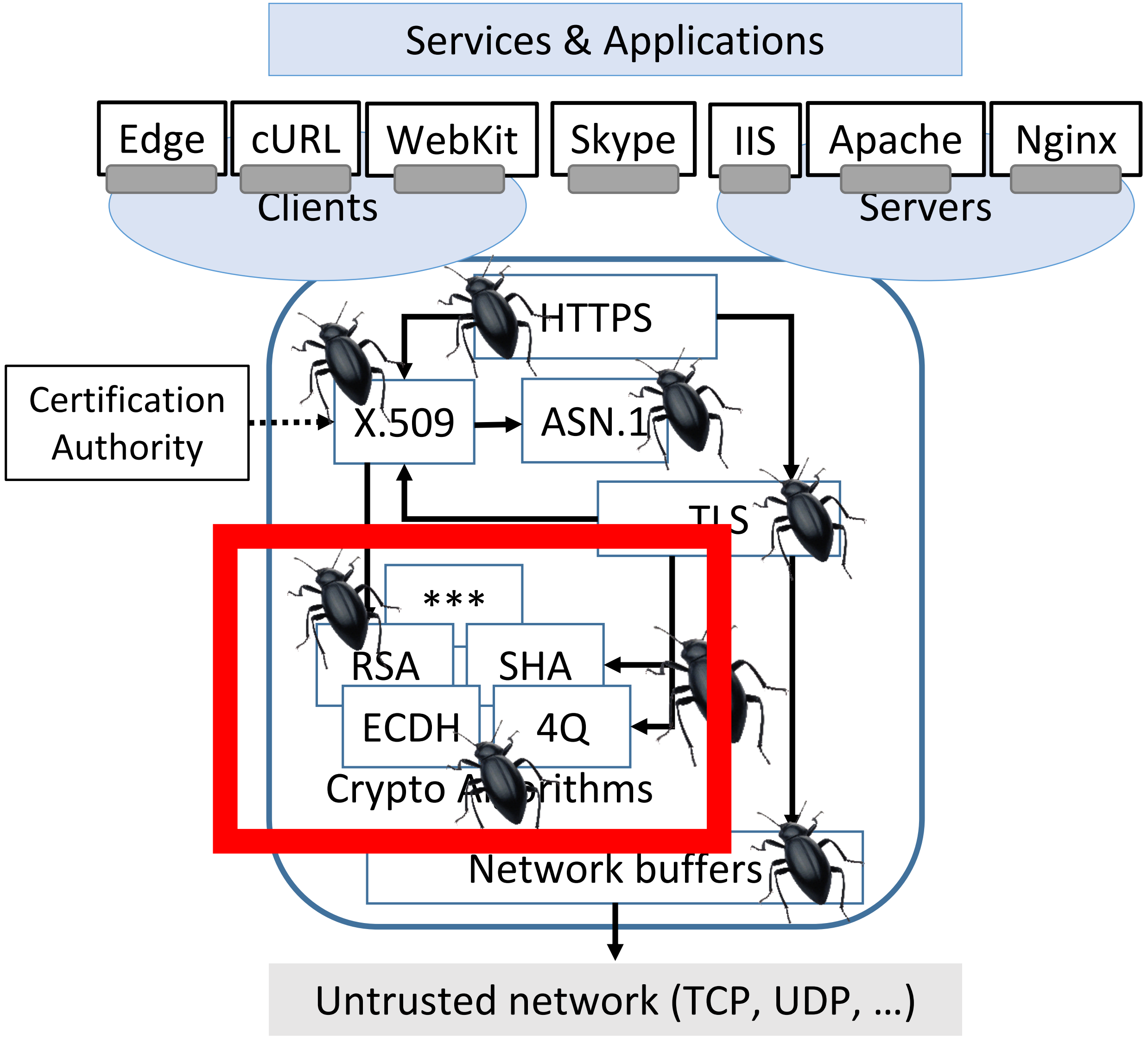
verifying

MSR’s Project Everest

Goal: verified HTTPS replacement

Challenges:

- scalability of verification
- performance
- usable tool chain



https://project-everest.github.io/

Subgoal: Verified low-level crypto

Efficient crypto requires customizations

- Poly1305: Uses the prime field with $p = 2^{130} - 5$
  - Need 130 bits to represent a number
  - Efficient implementations require custom bignum libraries to delay carries
  - On X86: use 5 32-bit words, but using only **26 bits in each word**
  - On X64: use 3 64-bit words, but using only **44 bits in each word**

Everest subgoal: generic, efficient bignum libraries

- Curve25519: Uses the prime field with $p = 2^{255} - 19$
  - On X64: use 5 64-bit words, but using only **51 bits per word**

- OpenSSL has 12 unverified bignum libraries optimized for each case

A generic bignum library

Bignum code can be **shared** between Curve25519, Ed25519 and Poly1305, which all use different fields

Only modulo is specific to the field (optimized)

Consequently:

- write once
- verify once
- extract three times

```
module Hacl.Bignum.Curve25519.Constants
let prime = pow2 255 - 19
let word_size = 64
let len = 5
let limb_size = 51
```

```
module Hacl.Bignum.Poly1305.Constants
let prime = pow2 130 - 5
let word_size = 64
let len = 3
let limb_size = 44
```

Prove correct in F*, extract to efficient C

```
val poly1305_mac: tag:nbytes 16 →
                  len:u32 →
                  msg:nbytes len{disjoint tag msg} →
                  key:nbytes 32 {disjoint msg key ∧ disjoint tag key} →
                   ST unit
        (requires (λ h → msg ∈ h ∧ key ∈ h ∧ tag ∈ h))
        (ensures (λ h0 _ h1 →
             let r=Spec.clamp h0.[sub key 0 16] in
             let s=h0.[sub key 16 16] in
             modifies {tag} h0 h1 ∧
             h1.[tag] == Spec.mac_1305 (encode_bytes h0.[msg]) r s))
```

Mathematical spec in F*

`poly1305_mac`: (1) computes a polynomial in GF(2^{130}-5),
(2) stores the result in `tag`,
(3) does not modify anything else

```c
void
poly1305_mac(uint8_t *tag, uint32_t len, uint8_t *msg, uint8_t *key)
{
  uint64_t tmp [10] = { 0 };
  uint64_t *acc = tmp
  uint64_t *r = tmp + (uint32_t)5;
  uint8_t s[16] = { 0 };
  Crypto_Symmetric_Poly1305_poly1305_init(r, s, key);
  Crypto_Symmetric_Poly1305_poly1305_process(msg, len, acc, r);
  Crypto_Symmetric_Poly1305_poly1305_finish(tag, acc, s);
}
```

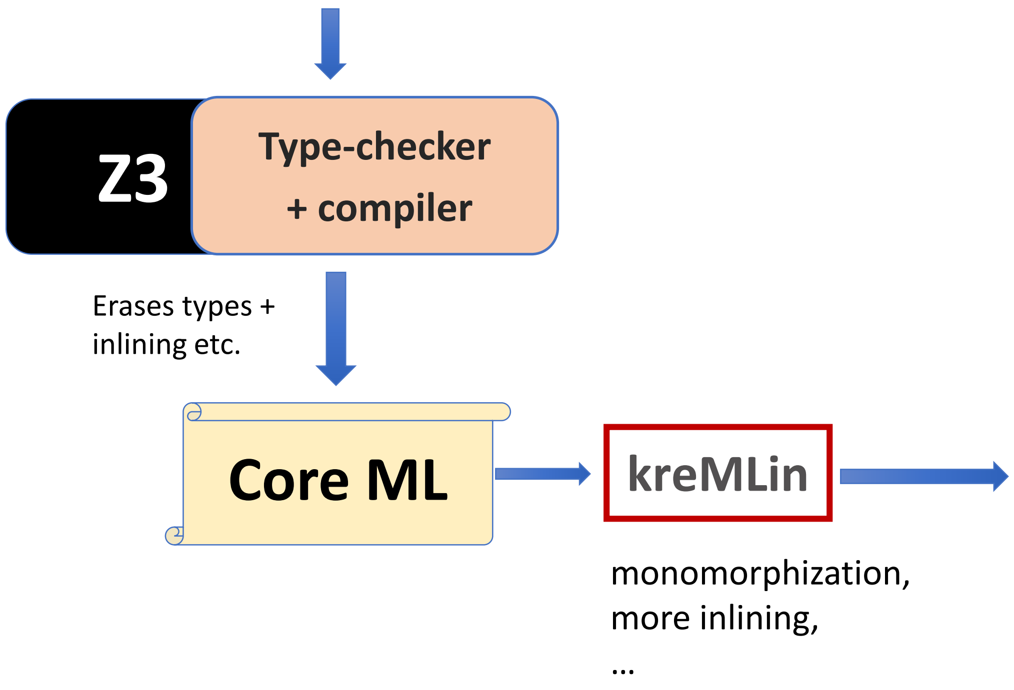
Efficient C implementation

Verification imposes no runtime performance overhead

Sample code Poly1305 MAC

F* source: core-ML with dependent types and effects

```
let poly1305_mac: tag:nbytes 16 →
              len:u32 →
              msg:nbytes len{disjoint tag msg} →
              key:nbytes 32 {disjoint msg key ∧ disjoint tag key} →
      ST unit
        (requires (λ h → msg ∈ h ∧ key ∈ h ∧ tag ∈ h))
        (ensures (λ h0 _ h1 → … )) = …
```



C source, tuned for readability, compliance with C linters etc.

```
void
poly1305_mac(uint8_t *tag, uint32_t len,
             uint8_t *msg, uint8_t *key)
{
  uint64_t tmp [10] = { 0 };
  uint64_t *acc = tmp
  uint64_t *r = tmp + (uint32_t)5;
  uint8_t s[16] = { 0 };
  Crypto_Symmetric_Poly1305_poly1305_init(r, s, key);
  Crypto_Symmetric_Poly1305_poly1305_process(msg, len, acc, r);
  Crypto_Symmetric_Poly1305_poly1305_finish(tag, acc, s);
}
```

clang, gcc,
msvc, CompCert, …

https://fstar-lang.org/tutorial/

Performance of Everest's High Assurance Crypto Library (HACL*)

Algorithm	Spec (F* loc)	Code+Proofs (Low* loc)	C Code (C loc)	Verification (s)
Salsa20	70	651	372	280
Chacha20	70	691	243	336
Chacha20-Vec	100	1656	355	614
SHA-256	96	622	313	798
SHA-512	120	737	357	1565
HMAC	38	[illegible]	[illegible]	[illegible]
Bignum-lib	-	[illegible]	[illegible]	[illegible]
Poly1305	45	[illegible]	[illegible]	[illegible]
X25519-lib	-	[illegible]	[illegible]	[illegible]
Curve25519	73	1901	798	246
Ed25519	148	7219	2479	2118
AEAD	41	309	100	606
SecretBox	-	171	132	62
Box	-	188	270	43
Total	801	22,926	7,225	9127

Table 1: HACL* code size and verification times

Verification enables using 64x64 bit multiplications, without fear of getting it wrong

- Several complete TLS ciphersuites
- *Verification can scale up!*

Algorithm	HACL*	OpenSSL
SHA-256	13.43	16.11
SHA-512	8.09	10.34
Salsa20	6.26	-
ChaCha20	6.37 (ref) 2.87 (vec)	7.84
Poly1305	2.19	2.16
Curve25519	154,580	358,764
Ed25519 sign	63.80	-
Ed25519 verify	57.42	-
AEAD	8.56 (ref) 5.05 (vec)	8.55

cycles/ECDH

- With performance as good as or better than hand-written C https://blog.mozilla.org/security/2017/09/13/verified-cryptography-firefox-57/

"Mozilla has partnered with INRIA and Project Everest (Microsoft Research, CMU, INRIA) to bring components from their formally verified HACL* cryptographic library into NSS, the security engine which powers Firefox.

Project Everest: Open Source

- https://www.github.com/FStarLang/FStar
- https://www.github.com/FStarLang/kremlin

- https://www.github.com/mitls/mitls-fstar
- https://www.github.com/mitls/hacl-star

- https://www.github.com/project-everest/vale

Formal Methods and Tools

Logic

High-level Specification
(TLA+)

Correctness of Cryptography and Protocols
(F*, Ivy, P#)

Bug Finding and Verification for C/C++
(SAGE, Corral)

Network Verification
(SecGuru)

thinking

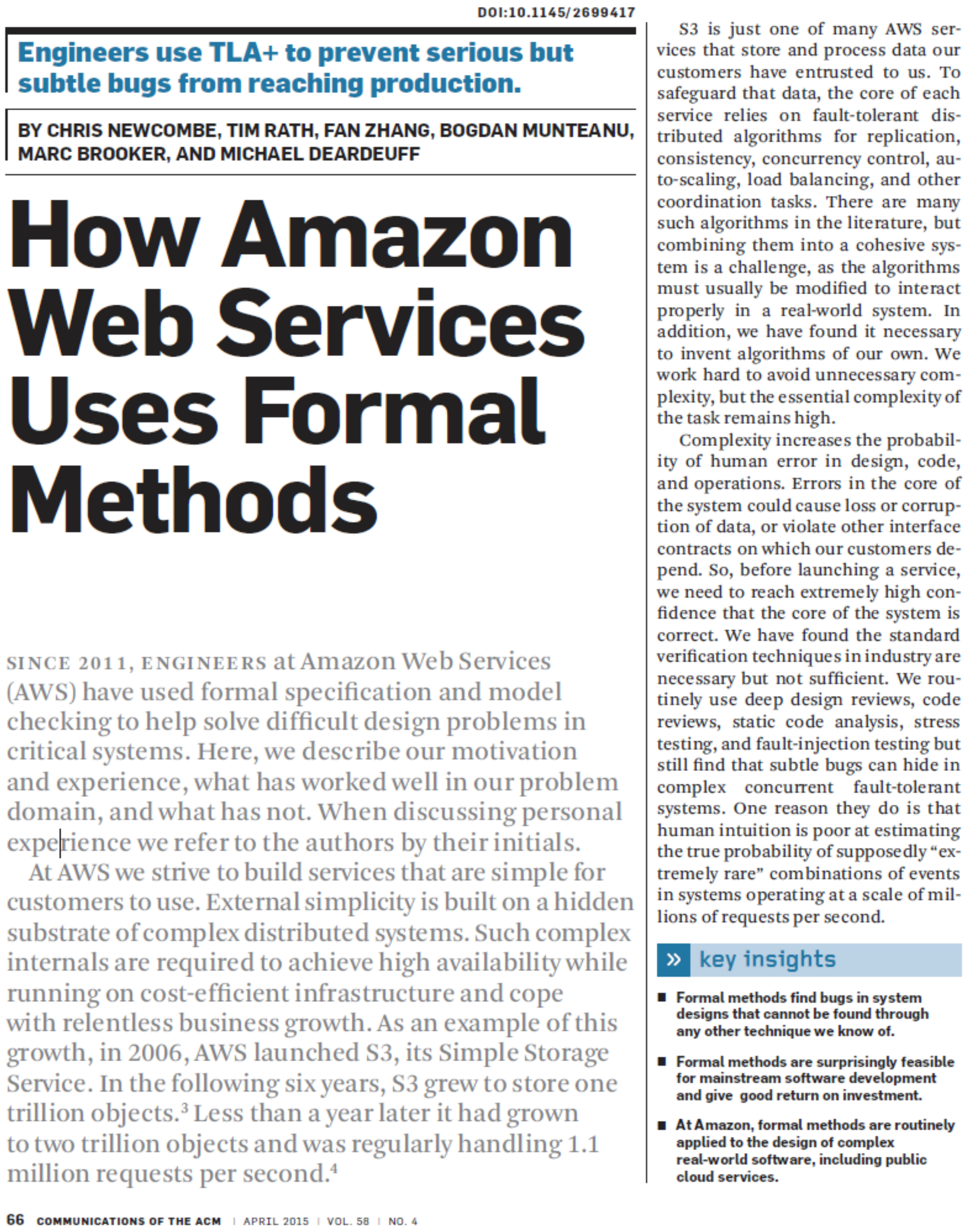
programming

testing

verifying

TLA+ (Leslie Lamport)

- A language for high-level modelling of digital systems, especially concurrent and distributed systems
- Tools for checking the models (TLC)
- IDE for end-to-end experience (Toolbox)
- https://github.com/tlaplus



DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

66 COMMUNICATIONS OF THE ACM | APRIL 2015 | VOL. 58 | NO. 4

Chris Newcombe, AWS

- Formal methods find bugs in system designs that cannot be found through any other technique we know of
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

Chris Newcombe, AWS

- Formal r … e found through…
- Formal … oftware develop…
- At Ama… esign of comple… es.

“TLA+ is the most valuable thing that I've learned in my professional career. It has changed how I work, by giving me an immensely powerful tool to find subtle flaws in system designs. It has changed how I think, by giving me a framework for constructing new kinds of mental-models, by revealing the precise relationship between correctness properties and system designs, and by allowing me to move from `plausible prose' to precise statements much earlier in the software development process.”

Formal Methods and Tools

Logic

High-level Specification
(TLA+)

Correctness of Cryptography and Protocols
(F*, Ivy, P#)

Bug Finding and Verification for C/C++
(SAGE, Corral)

Network Verification
(SecGuru)

thinking

programming

testing

verifying