Push, Pull, Partner

A Few Models for Working with Industry

Thomas (Tom) Ball

Research Manager, Principal Researcher

Microsoft Research

http://icsa-conferences.org/2018/

My experience

- Programming languages
- Program analysis
- Formal methods
- Software engineering

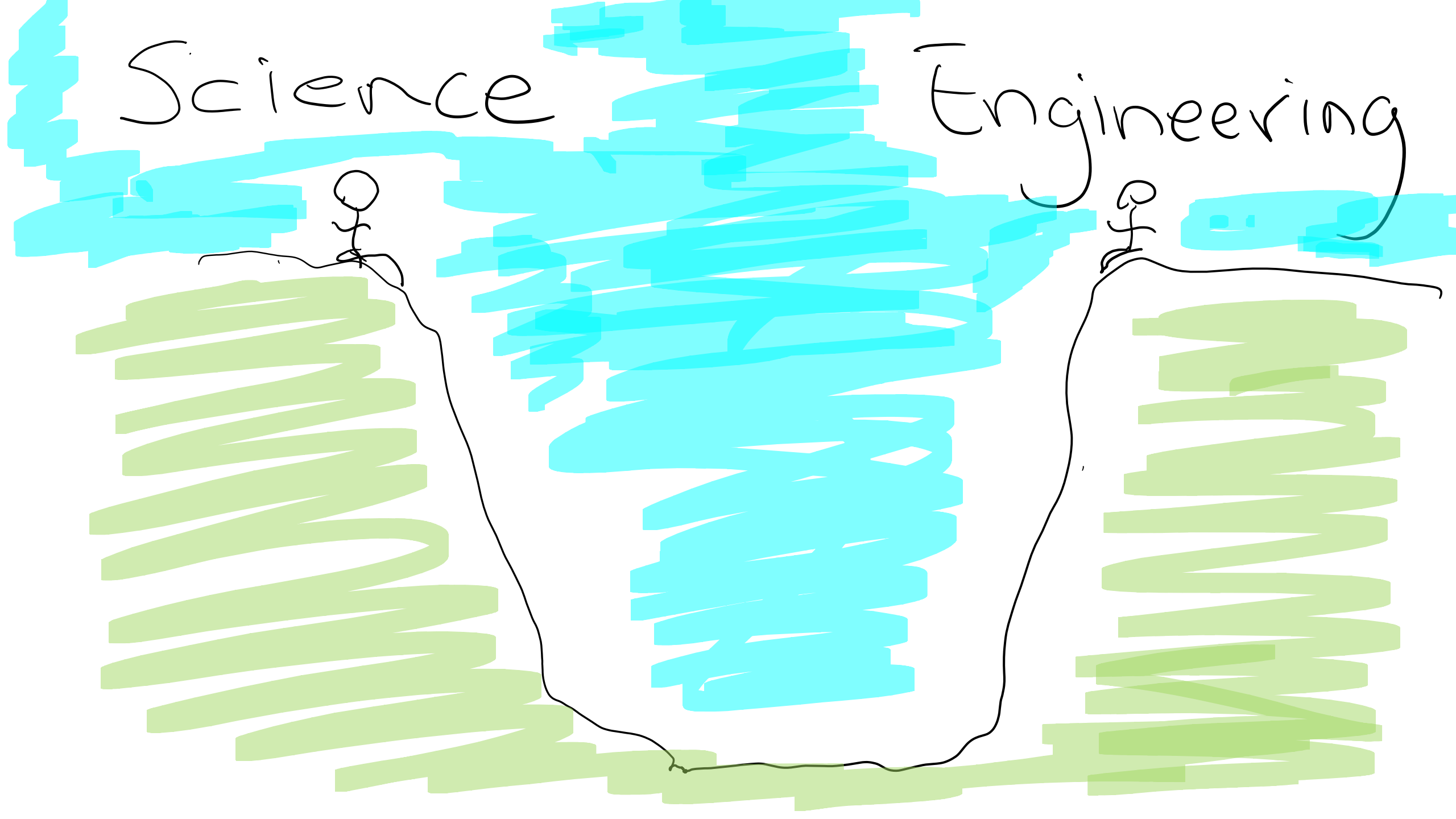
See Research in Software Engineering home page for more

Overview

- Science and Engineering, via Sir Tony Hoare
- The Industrial Research Cycle

- Three Examples: Push, Pull, Partner
  - Device Driver Quality: http://research.microsoft.com/slam
  - Formally Verified Systems: https://project-everest.github.io/
  - CS Education: http://makecode.com

- Conclusions


Science
Engineering

Timescale

Posterity

Client with deadline/budget

Idealism

Ideals for their own sake

Compromise: solving in presence of constraints

Certainty

High degree of assuredness

Managing risk

Generality

Abstraction, unifying theory

Particulars and specific solutions

Separation of Concerns

Isolation, experimentation

Resolution of huge collection of concerns

Originality

Sources acknowledged,

plagiarism punished

Tried and true methods

Formalization

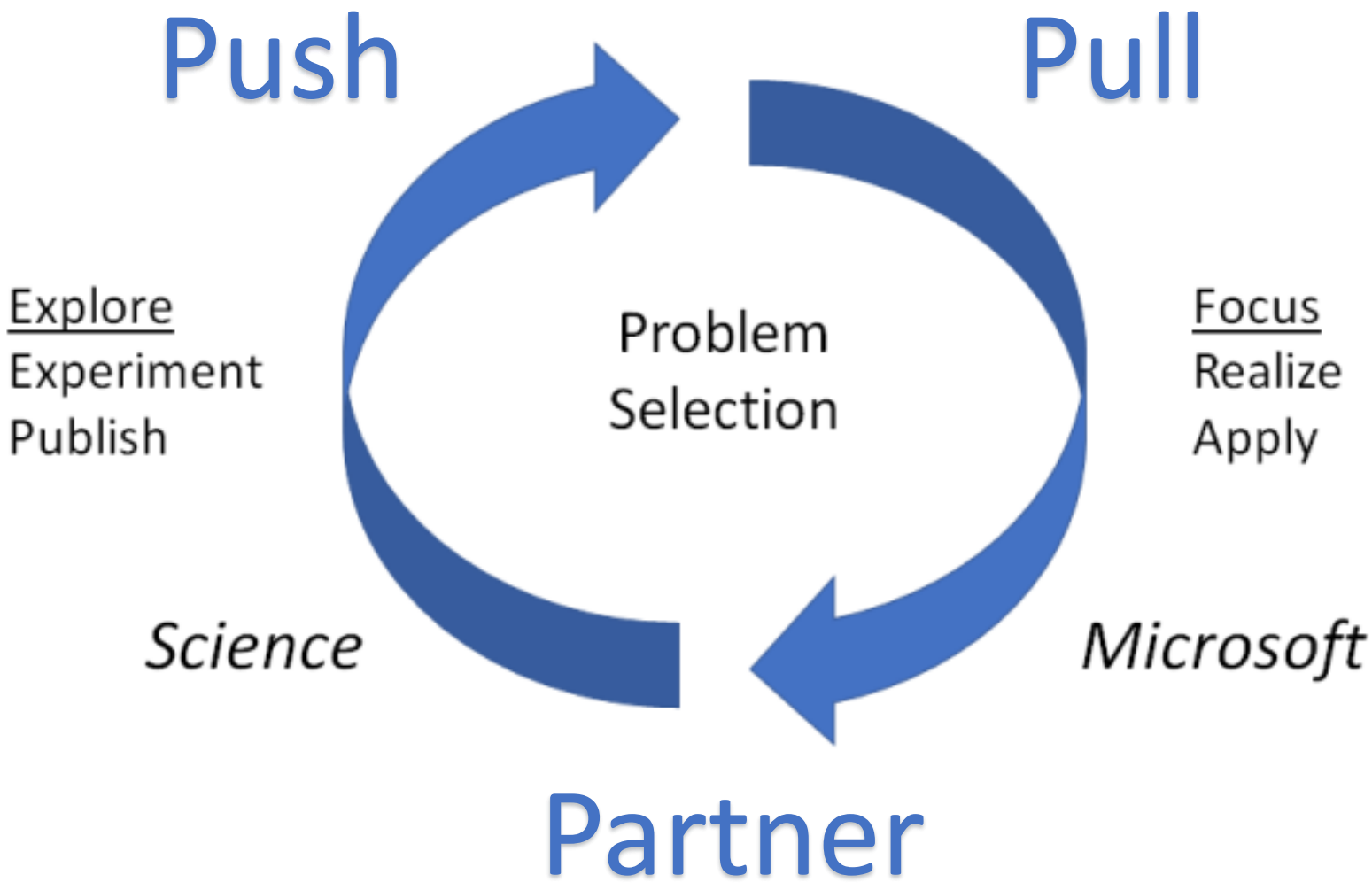
Precision and proof

Experience, engineering standards

MSR and The Industrial Research Cycle

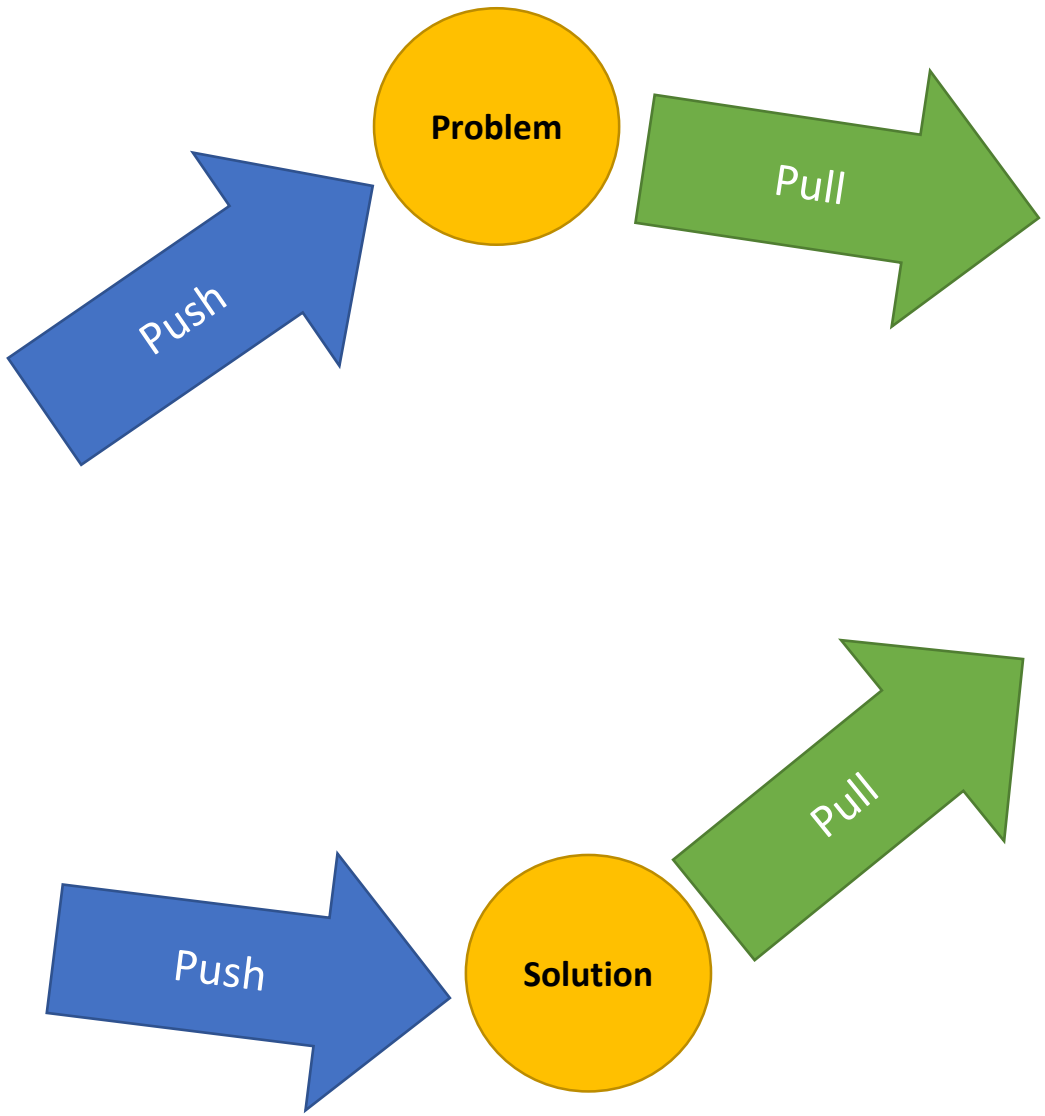
MSR gives you the freedom to explore and expand the bounds of scientific knowledge, as in academia, with the added challenge to align your scientific pursuits with company problems and to drive for impact on Microsoft, especially as you grow in seniority at the company.

The Industrial Research Cycle



Push and Pull

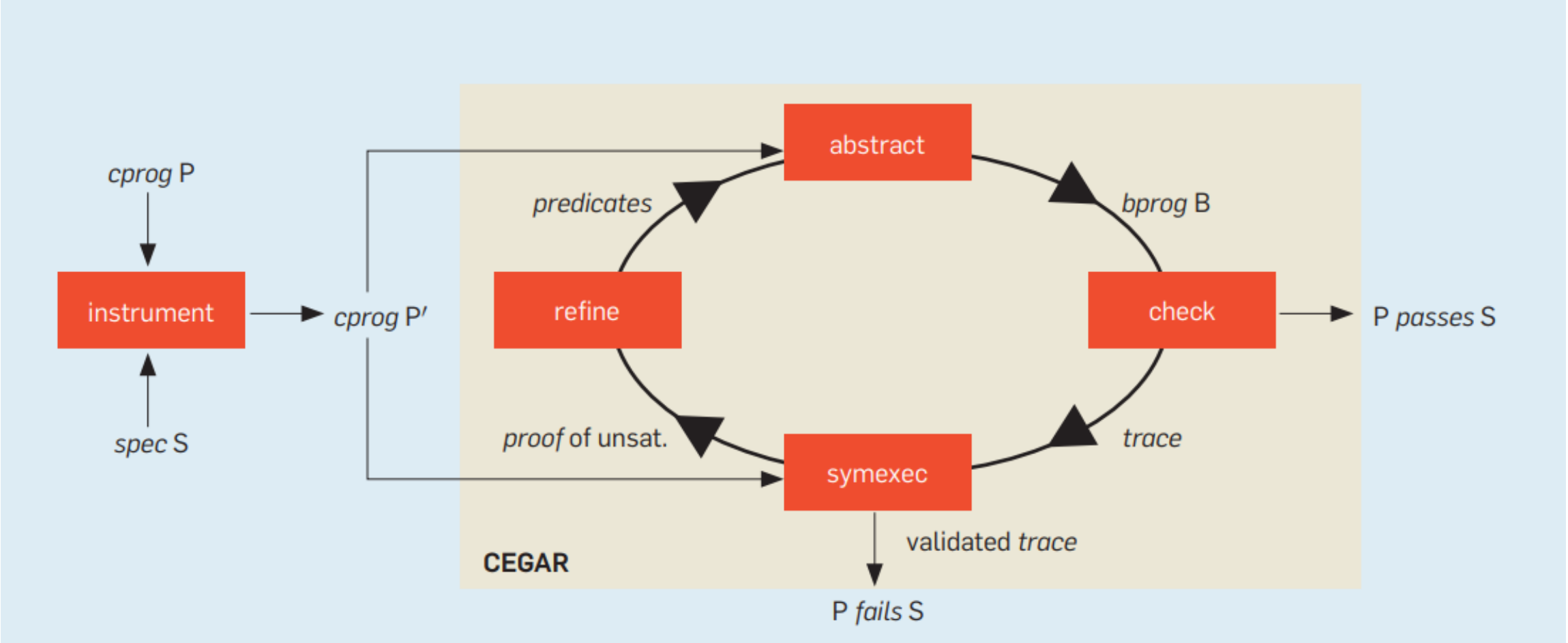
- Problem definition
- Solution space
- The environment



Push: SLAM project

- Tom Ball and Sriram Rajamani
  - Different backgrounds: program analysis, model checking
- Started SLAM project in 2000
  - automated defect detection in C programs
  - focus on device drivers, access to experts and source code in Microsoft
- First two years (2000-2001)
  - defined a new direction for automated software analysis
  - **hosted many interns!**
  - published a good number of papers
  - created a prototype tool that found non-trivial bugs in device drivers
  - attracted attention from the academic research community

Push: SLAM project



Pull: Driver Quality and Static Driver Verifier (SDV)

- 2001: Nimbda and Code Red worms take down Windows PCs
  - https://www.wired.com/2002/01/bill-gates-trustworthy-computing/
  - https://www.cnet.com/news/seeking-signs-of-microsoft-security-push/

- 2002: Windows Security Push and Security Definition Lifecycle (SDL)

- 2002: Windows Driver Quality Team formed
  - Hired Byron Cook and Vladimir Levin into Windows to help create SDV

Partnering: Measuring Progress on SDV

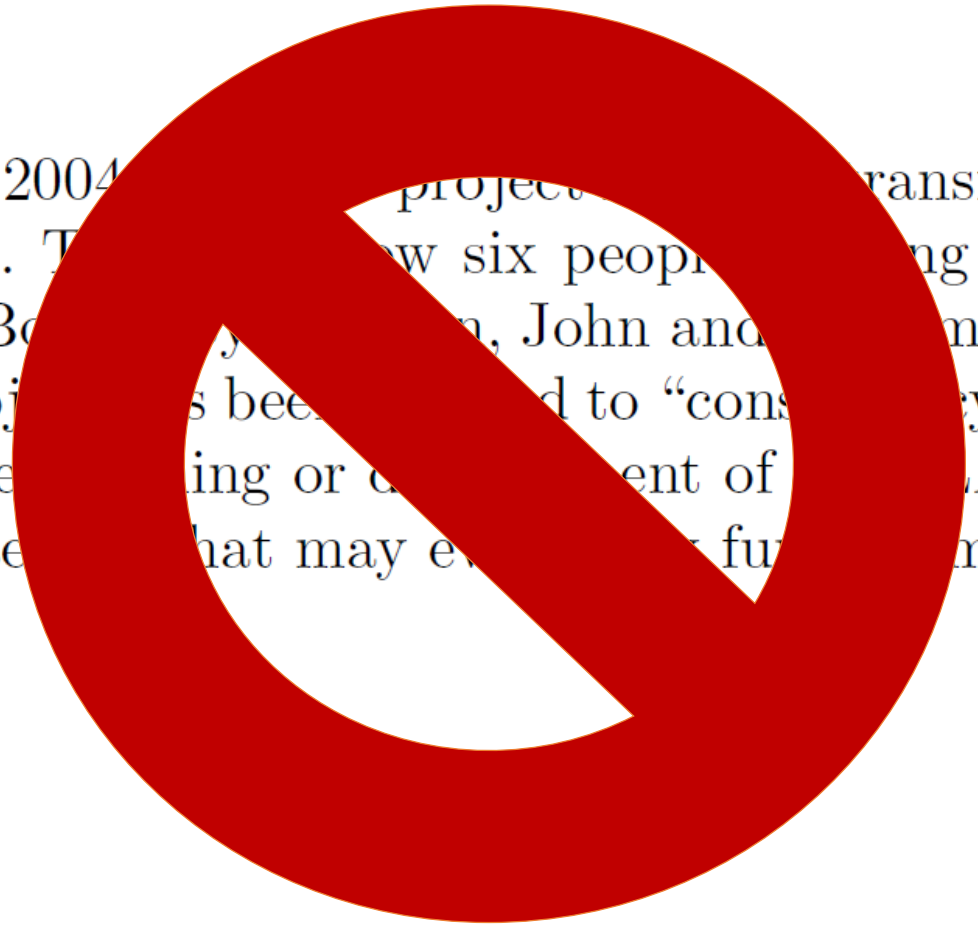
- The biggest problem in the autumn of 2002:
  - lack of a clear statement of how progress/success would be measured

- Nar Ganapathy created criteria document that listed:
  - the type of drivers that SDV needed to run on
  - specific drivers on which SDV needed to run successfully
  - performance expectations from SDV
  - allowable ratio of false errors the tool could produce

Lessons from SLAM and SDV

- *Focus on Problems not Technology*
- *Leverage Diverse Views*
- *Plan Carefully*
- *Maintain Continuity and Ownership*
- *Reflect and Assess*
- *Put Yourself in Another's Shoes*

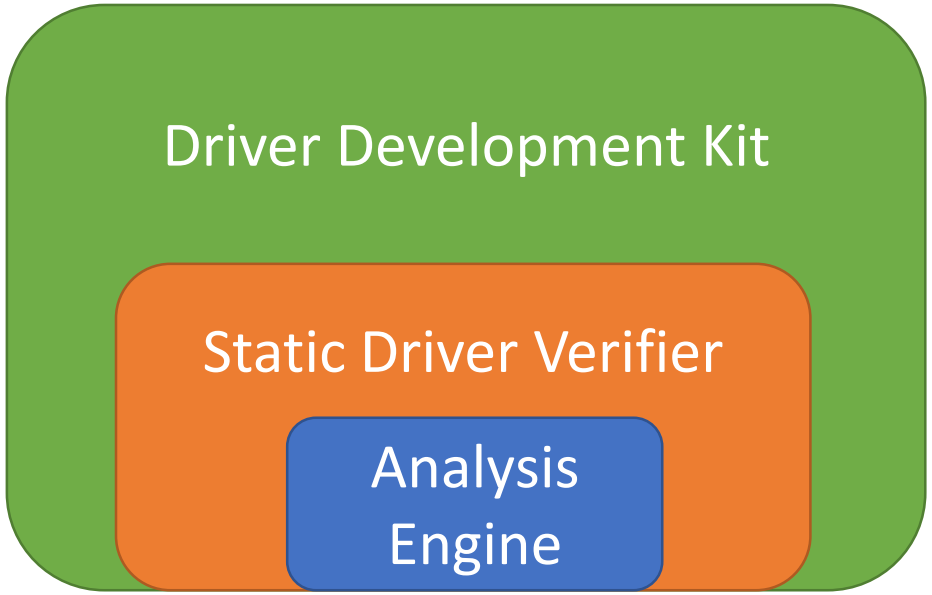
SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft



As of the beginning of 2004 [illegible] project [illegible] ransferred from Microsoft Research to Windows. T[illegible]w six peopl[illegible]ng full-time on SDV in Windows: Abdullah, Bo[illegible], John and [illegible]mir. Sriram and Tom's involvement in the proj[illegible]s bee[illegible]d to "cons[illegible]y"; they are no longer heavily involved in the [illegible]ing or d[illegible]ent of [illegible]AM/SDV technology but are continuing rese[illegible]hat may e[illegible] fu[illegible]mpact SDV .

Post-2004: Defining Partnership

- Ownership
  - Analysis engine, tool, delivery vehicle
  - Organizational values
- Idealism
  - searching for proof vs. searching for bugs
- Separation of concerns
  - Architecture was great for research, less so for production



Since 2004 SDV 1.0 Delivery

- Z3 Automated Theorem Prover (de Moura, Bjorner), 2006-

- New engines, using Z3, developed by MSR:
  - Software Property Checking via Static Analysis and Testing, TACAS 2009
  - SLAM2: static driver verification with under 4% false alarms, FMCAD 2010
  - A Solver for Reachability Modulo Theories, CAV 2012
  - Powering the Static Driver Verifier using Corral, FSE 2014
  - Angelic Verification: Precise Verification Modulo Unknowns, CAV 2015
  - Inferring annotations for device drivers from verification histories, ASE 2016
  - CloudSDV: Enabling Static Driver Verifier using Microsoft Azure, IFM 2016

Ella Bounimova, Aditya Nori, Rahul Kumar, Shaz Qadeer, Akash Lal, Shuvendu Lahiri, and many more interns!

Microsoft Research

Microsoft Research - Inria JOINT CENTRE

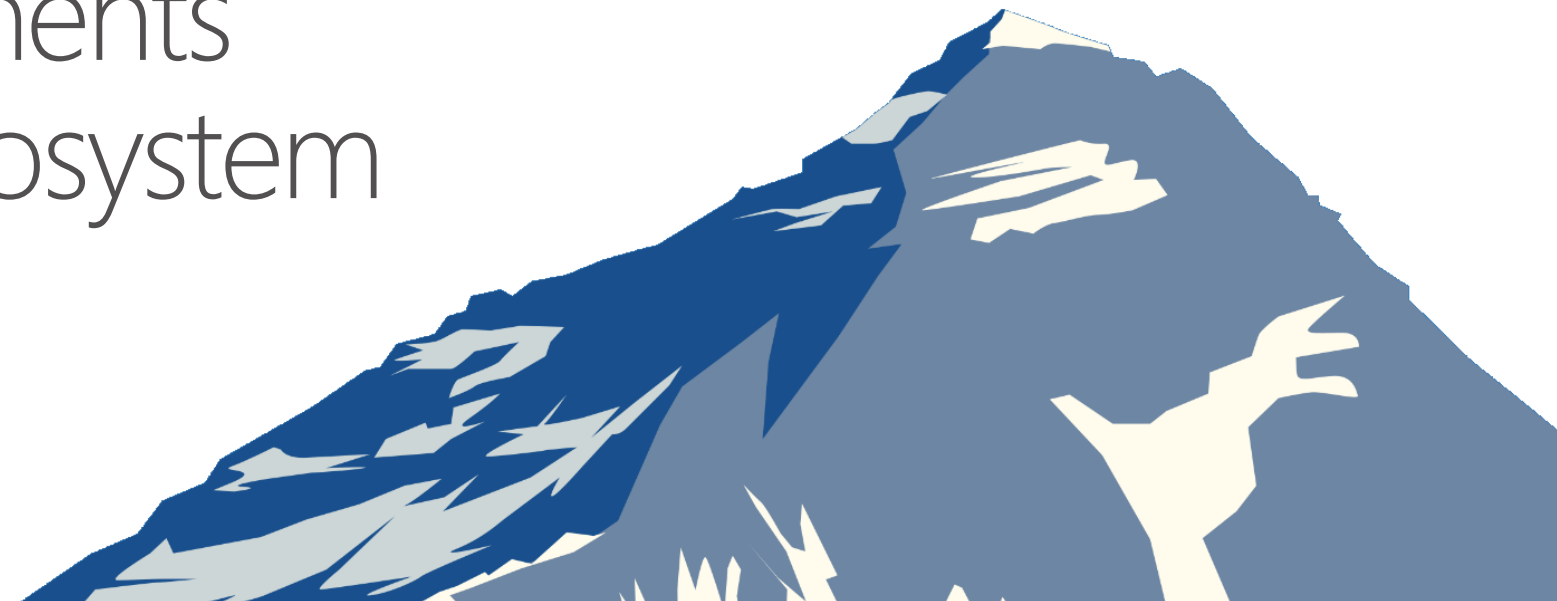
Inria INVENTEURS DU MONDE NUMÉRIQUE

Carnegie Mellon University

Microsoft

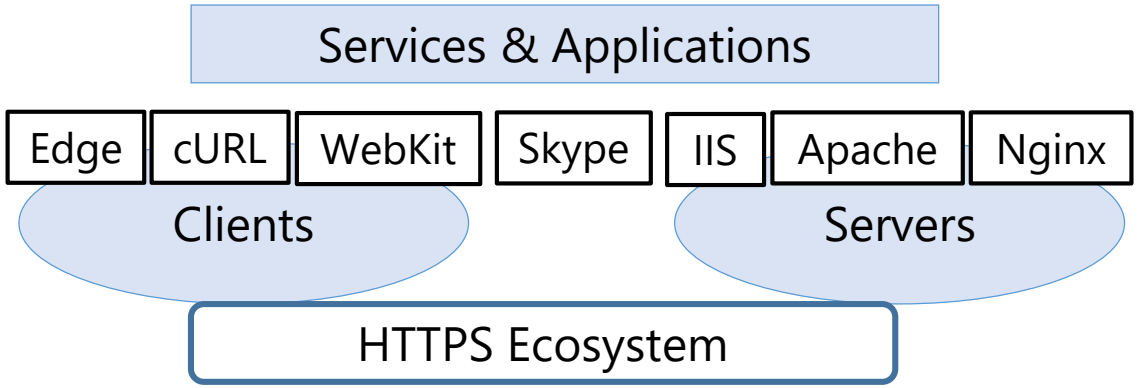
Everest

Building and Deploying Verified Components in the HTTPS Ecosystem



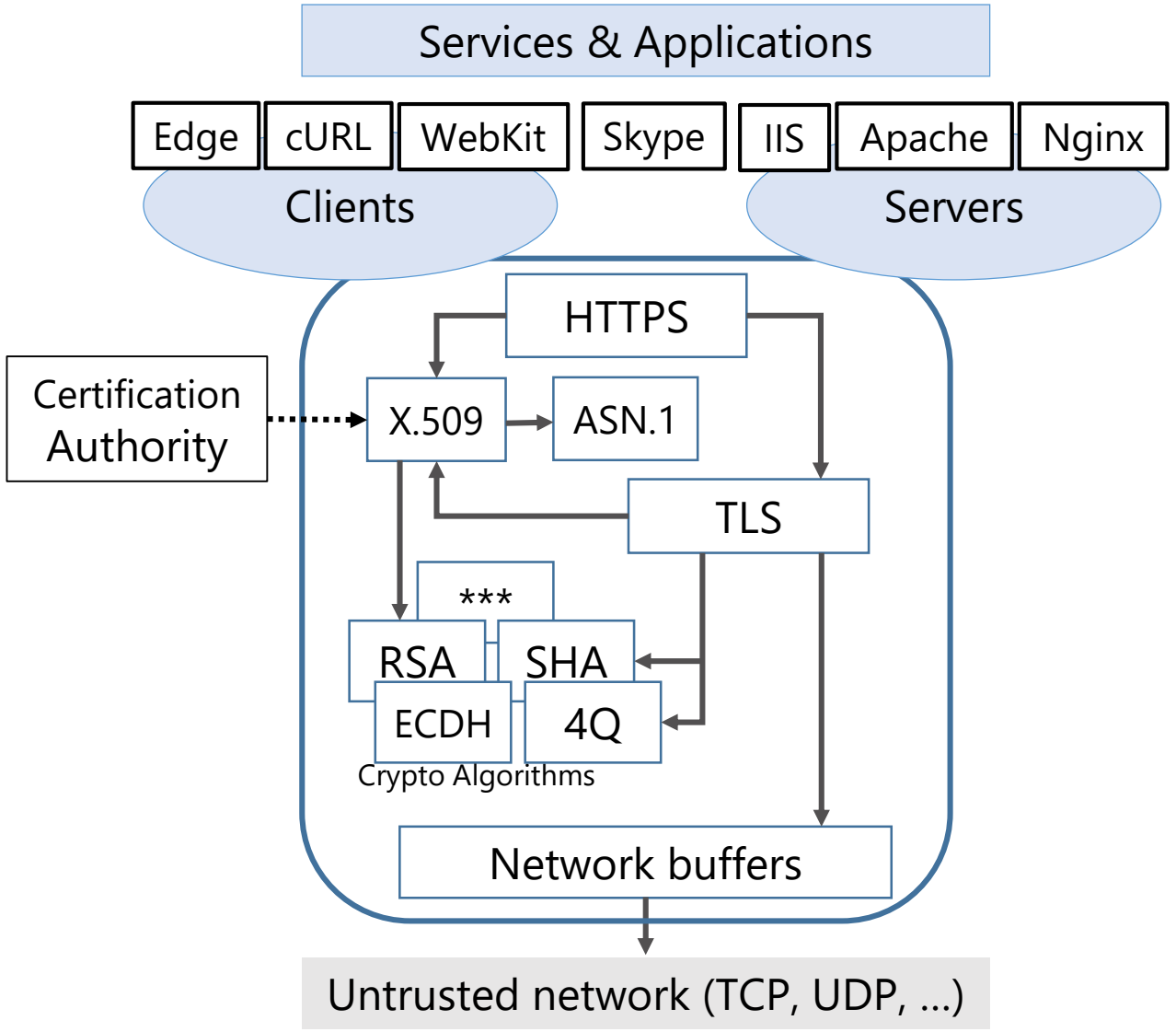
*the Everest VERified End-to-end Secure Transport

The HTTPS Ecosystem is critical



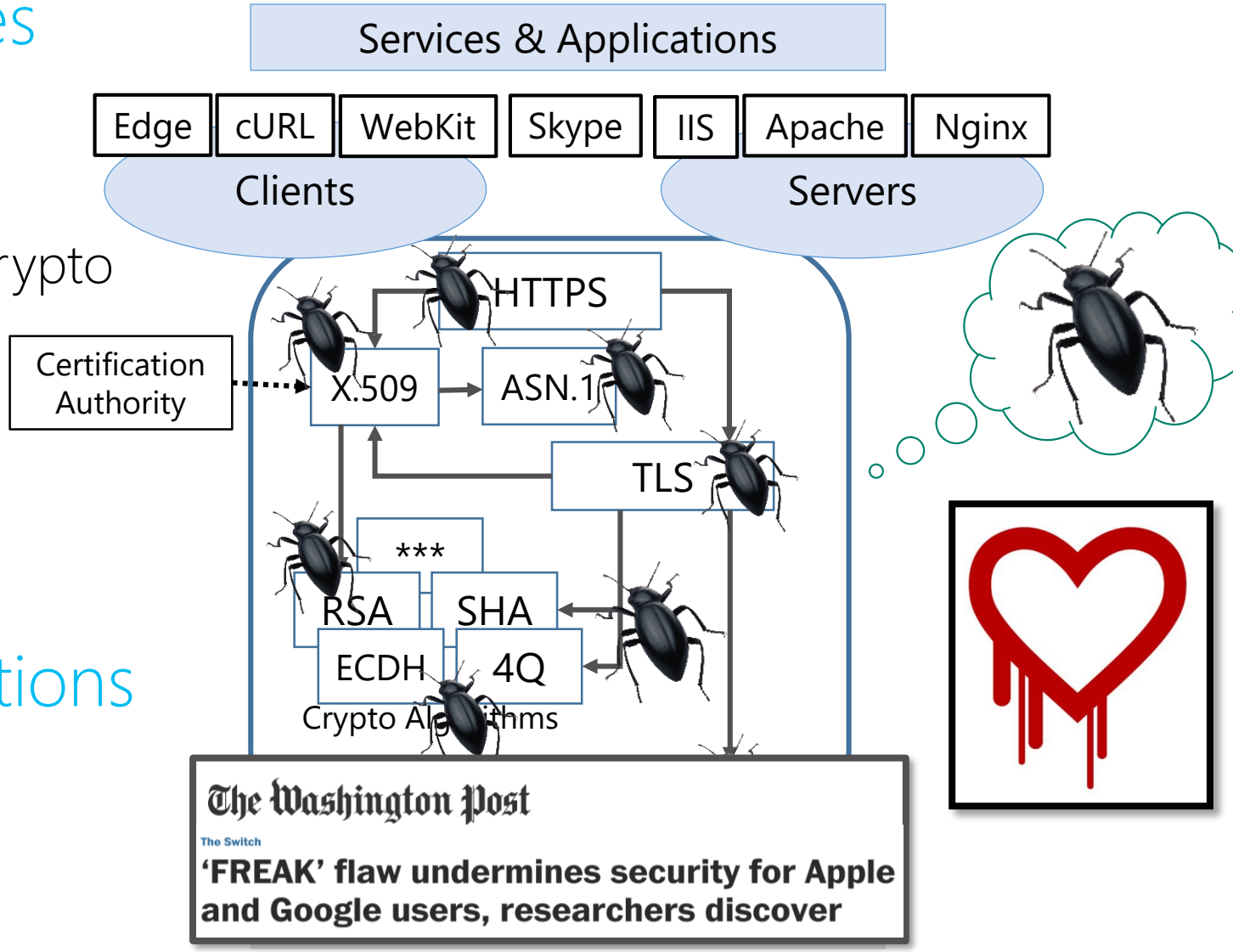
- Most widely deployed security?
  ½ Internet traffic (+40%/year)
- Web, cloud, email, VoIP, 802.1x, VPNs, …

The HTTPS Ecosystem is complex



The HTTPS Ecosystem is broken

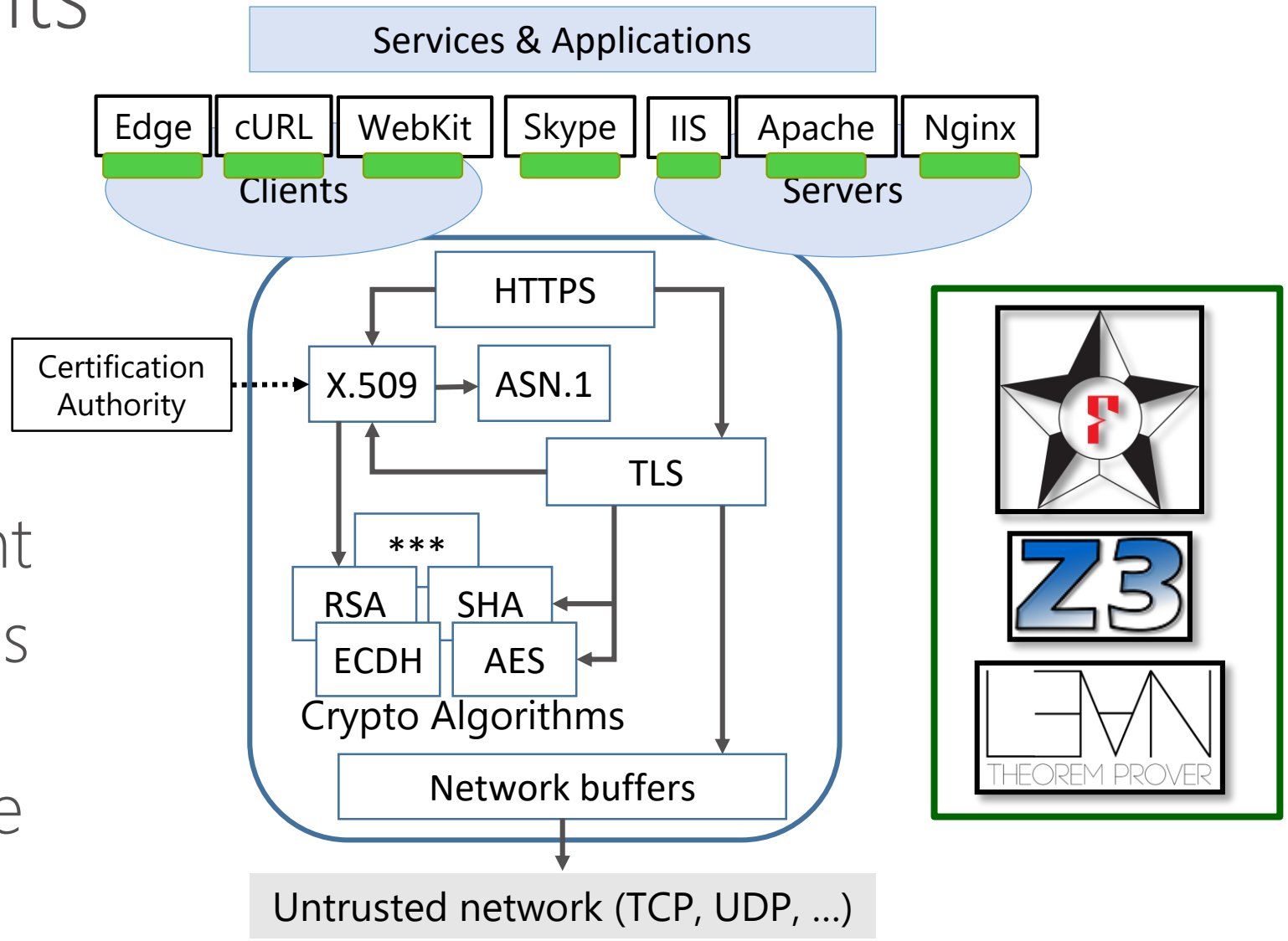
- 20 years of attacks & fixes
  - Buffer overflows
  - Incorrect state machines
  - Lax certificate parsing
  - Weak or poorly implemented crypto
  - Side channels
  - Informal security goals
  - Dangerous APIs
  - Flawed standards
- Mainstream implementations
  - OpenSSL, SChannel, NSS, …
  - Still patched every month!



Everest 2016—2021:

Verified Components for the HTTPS Ecosystem

- Strong verified security
- Widespread deployment
- Trustworthy, usable tools
- Growing expertise in high-assurance software development



Secure components: Where we are today

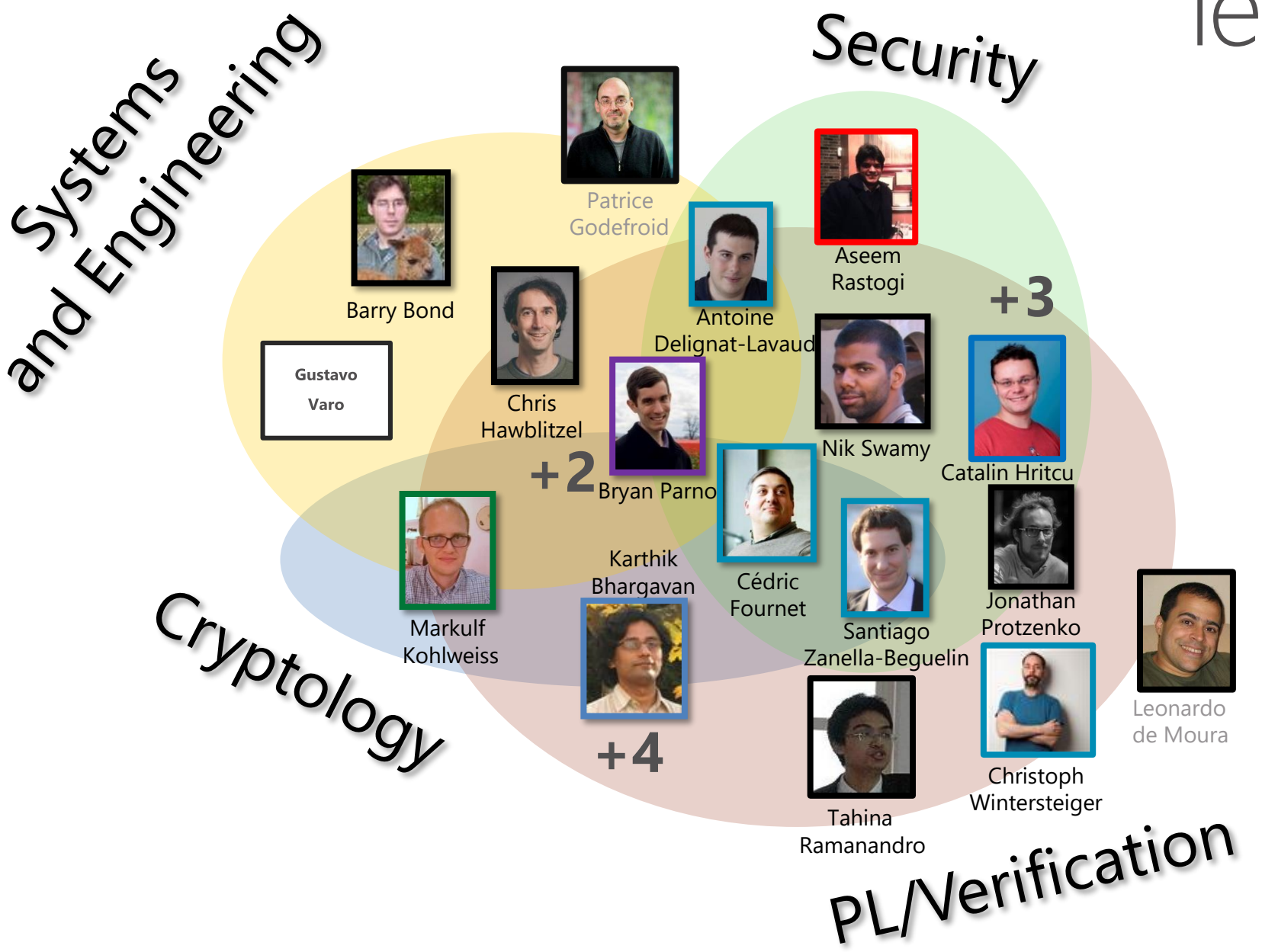
Close to production quality

- HACL*: High-assurance Crypto Library
  - Verified implementations of crypto algorithms
  - Performance comparable to hand-written C
  - Functionally correct, cryptographically secure, side-channel resistant, in Firefox Quantum
- VALE: Verified Assembly for Everest
  - Assembly level crypto, with performance comparable to OpenSSL assembly
- TLS Record Layer Protection:
  - Verified, efficient code compiled to C
  - Functionally correct, cryptographically secure, reasonably fast [used in Windows prototype]

Research prototypes

- TLS 1.3 full protocol
  - 1 year of interop with other early implementations
  - Deployments within existing clients (IE, Curl) and servers (nginx)
  - Partial verification (in progress)
- QUIC library
  - A TLS handshake library suitable for use from QUIC

Team Members



- Cambridge
- Bangalore
- Redmond
- Paris (INRIA)
- Pittsburgh (CMU)
- Edinburgh

Having an impact on industry

Developer acceptance and maintenance

- Why will people risk deploying unfamiliar verified code?
- Why choose Everest over OpenSSL?
- Who will patch Everest overnight?
- Who will maintain Everest 10 years from now?
- How do we keep deployed C/C++ code from diverging from high-level verified code?

Developer acceptance approach

- Open-source standard-compliant transparent development

  Early reference implementation of TLS 1.3
  IETF endorsement + interop and performance testing

- Compile to clean readable commented C++ code

  Allows developers to

  - Ignore verification machinery, if desired
  - Apply code reviews, auditing, or any other reasoning approaches

- Drop-in replacements in popular software (API compatibility)

  Also: interop & security testing tools for specialists (flexTLS)

  Also: lightweight deployment with minimal TCB (SGX)

- Resilience to ongoing CVEs in other implementations

Maintenance approach

- Improve usability of verification tools to the point where outsiders can adapt, improve, recompile our code
- Build a thriving open-source community
- Buy-in from Microsoft and others? See OpenSSL

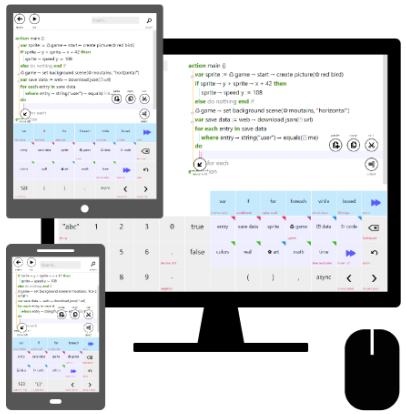
Open Source: Everest

- F*, a verification-oriented dialect of ML
- miTLS, implementation of the TLS protocol, written in F*
- KreMLin, a compiler from a subset of F* to C
- HACL*, a verified library of cryptographic primitives written in F*
- Vale, a DSL for writing verified crypto primitives in assembly
- Z3, automated theorem prover

Beginning Programming

[Touch Develop]

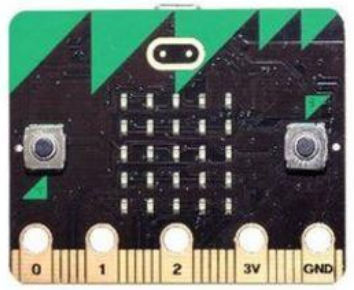
- 2010-2018



- Hobbyist, **Educator**, Researcher
- Windows Phone 7 native app
- Responsive web app
- http://github.com/microsoft/touchdevelop

BBC micro:bit

- 2015-2016 (BBC, TouchDevelop)
- 2017- (micro:bit Foundation)



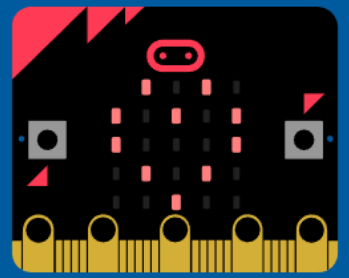
Microsoft MakeCode

About Get Inspired Learn Hardware Resources Blog Labs

www.makecode.com

Hands-on computing education

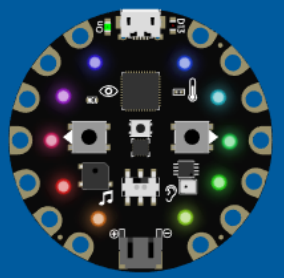
micro:bit



micro:bit

Code

adafruit



Circuit Playground Express

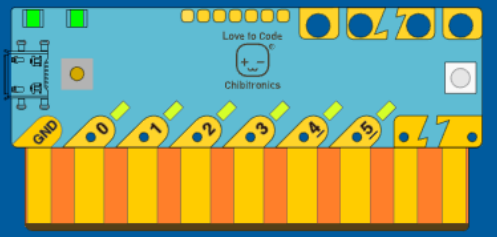
Code

Microsoft

Minecraft

Code

chibi tronics

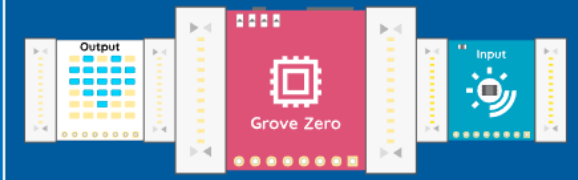


Chibi Chip

Code

seeed studio

Beta



Grove Zero

Code

wonder workshop

Cue

Apps

Open Source: MakeCode

- Framework and support
  - https://github.com/Microsoft/pxt
  - https://github.com/Microsoft/pxt-blockly
  - https://github.com/Microsoft/pxt-monaco-typescript

- Targets
  - https://github.com/Microsoft/pxt-adafruit
  - https://github.com/Microsoft/pxt-microbit
  - https://github.com/Microsoft/pxt-maker

Conclusions

- Awareness of the cultures of the scientist and the engineer
- The process of push and pull
- Working through the process can yield a long-term partnership
- **Please use Microsoft Research as resource**
  - come visit and give a talk
  - send us your best students (interns, hiring)
  - use our open source projects for your research