Programming Languages for High-Assurance Autonomous Vehicles

Lee Pike (speaker), Pat Hickey, James Bielman, Trevor Elliott, John Launchbury, Erlend Hamberg, Thomas DuBuisson

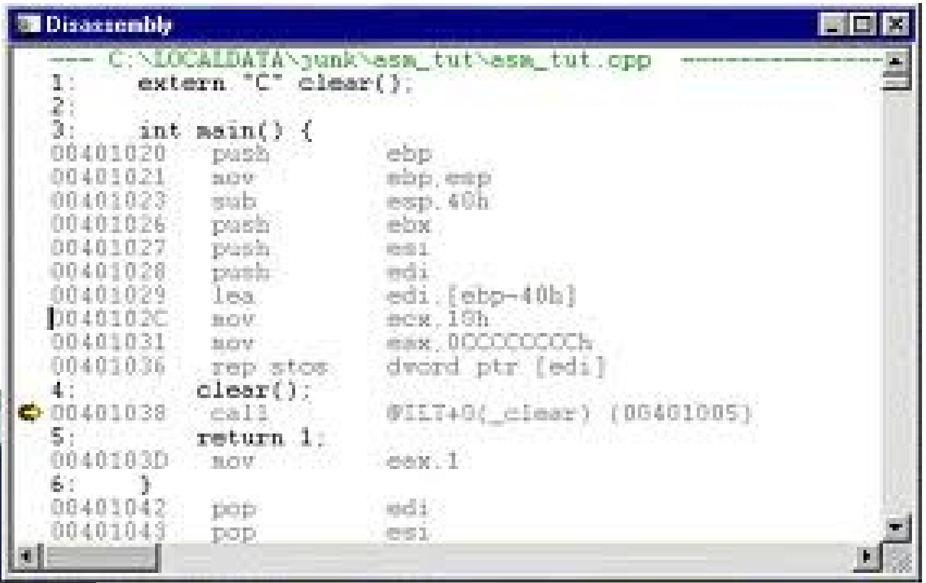
HCSS | May 2014

Embedded Security: Where Are We At?

Embedded Programming 1970s - 2014

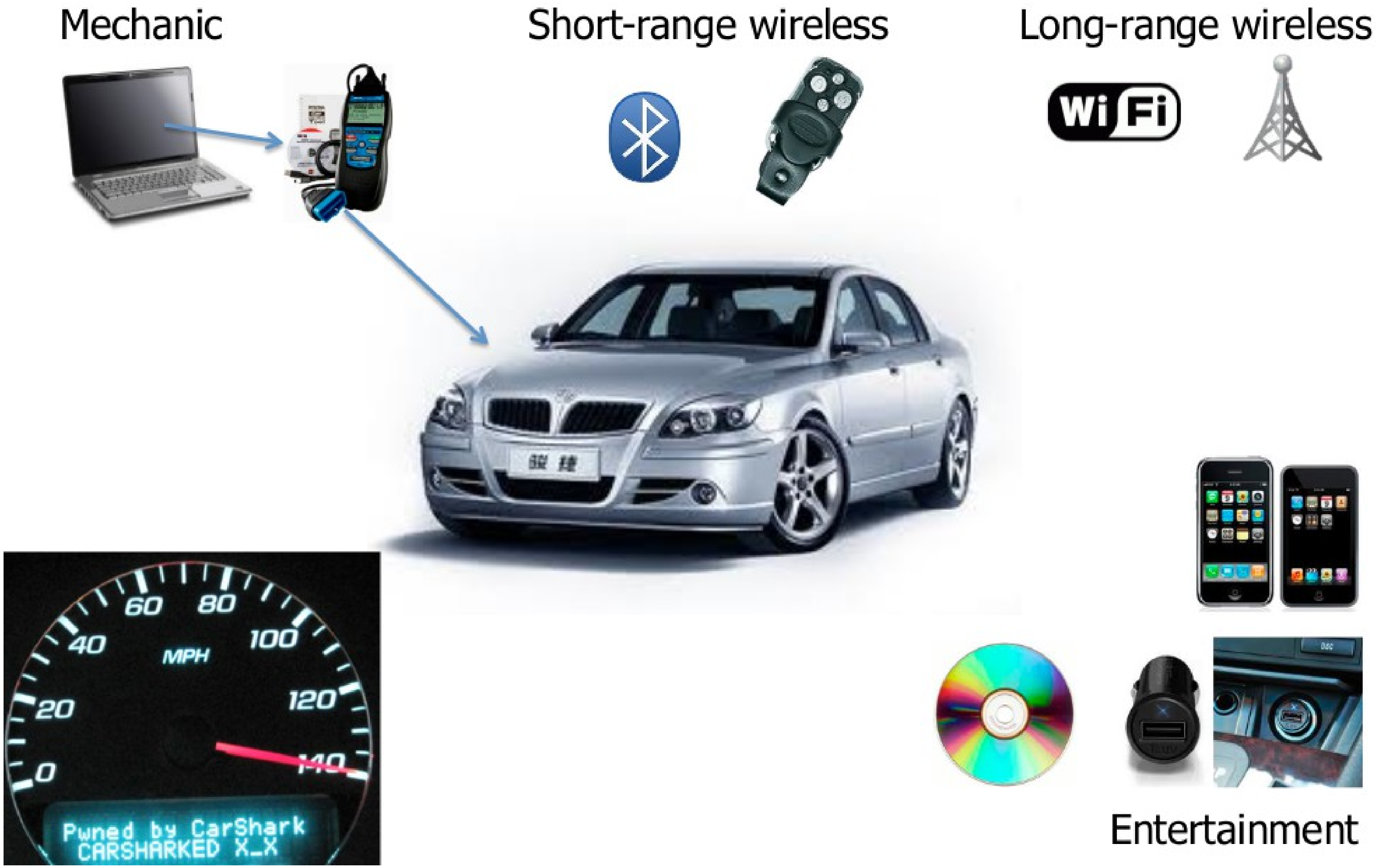
Typical tools:

- **Programming**: C/C++
- **Building**: GNU Make/GCC
- **Debugging**: GDB



```
--- C:\LOCALDATA\junk\esm_tut\esm_tut.cpp ------------------
1:      extern "C" clear();
2:
3:      int main() {
00401020   push         ebp
00401021   mov          ebp,esp
00401023   sub          esp,40h
00401026   push         ebx
00401027   push         esi
00401028   push         edi
00401029   lea          edi,[ebp-40h]
0040102C   mov          ecx,10h
00401031   mov          eax,0CCCCCCCCh
00401036   rep stos     dword ptr [edi]
4:          clear();
00401038   call         @ILT+0(_clear) (00401005)
5:          return 1;
0040103D   mov          eax,1
6:      }
00401042   pop          edi
00401043   pop          esi
```

From Embedded Systems to Cyber Physical Systems



Mechanic

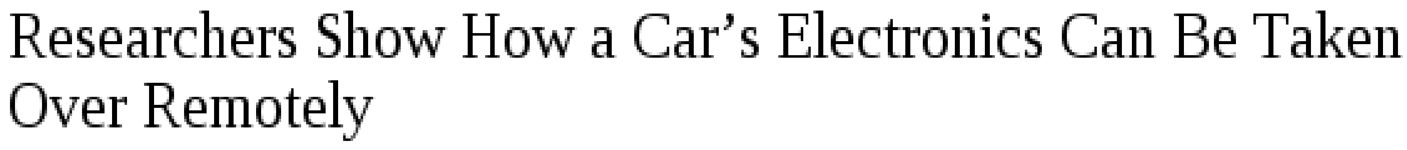
Short-range wireless

Long-range wireless

Entertainment

src: Kathleen Fisher, http://www.cyber.umd.edu/sites/default/files/documents/symposium/fisher-HACMS-MD.pdf

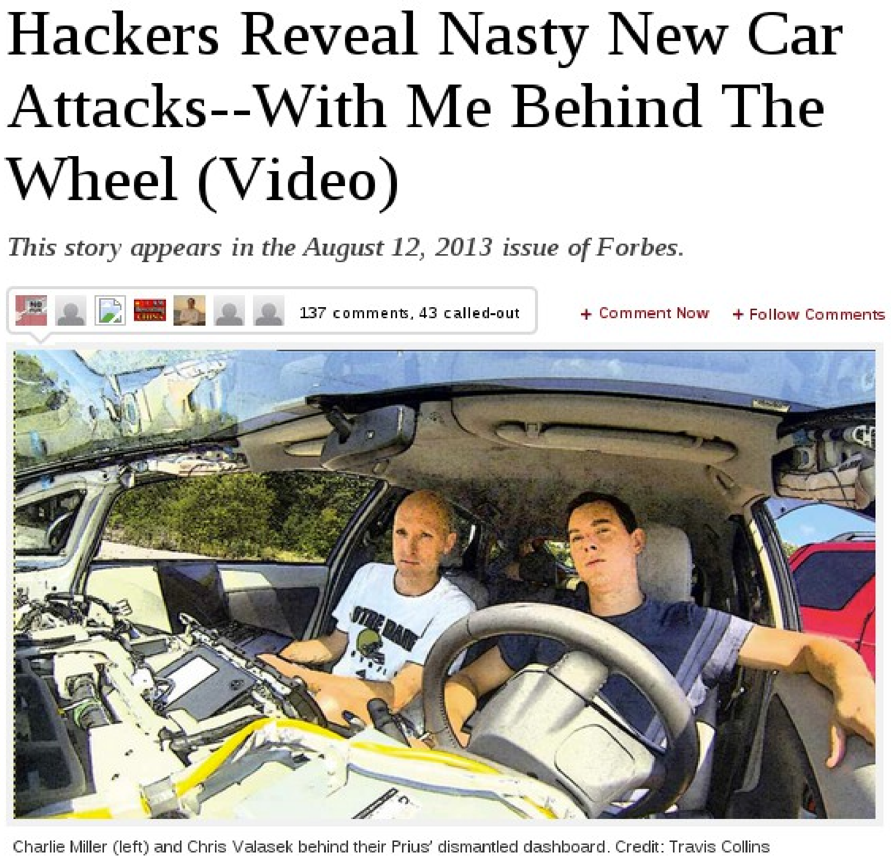
Hacking Cars



Researchers Show How a Car's Electronics Can Be Taken Over Remotely

By JOHN MARKOFF
Published: March 9, 2011

New York Times

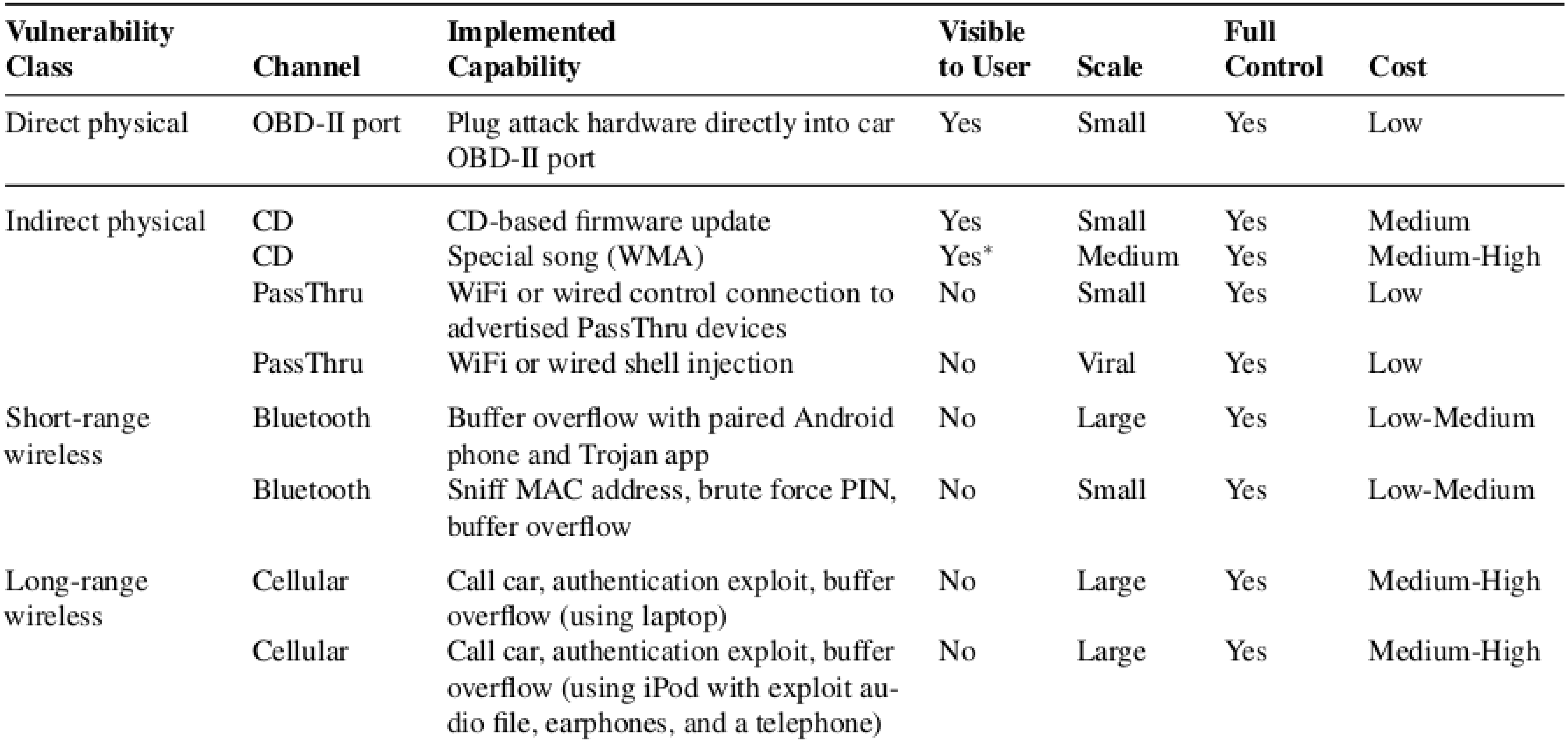


Hackers Reveal Nasty New Car Attacks--With Me Behind The Wheel (Video)

This story appears in the August 12, 2013 issue of Forbes.

Charlie Miller (left) and Chris Valasek behind their Prius' dismantled dashboard. Credit: Travis Collins

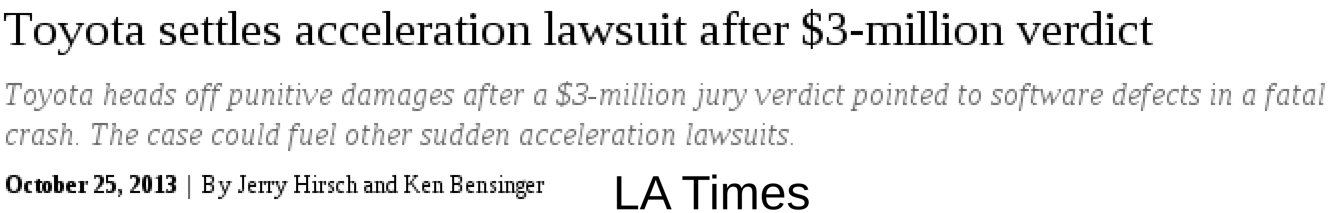
Example Attacks



Vulnerability Class	Channel	Implemented Capability	Visible to User	Scale	Full Control	Cost
Direct physical	OBD-II port	Plug attack hardware directly into car OBD-II port	Yes	Small	Yes	Low
Indirect physical	CD	CD-based firmware update	Yes	Small	Yes	Medium
	CD	Special song (WMA)	Yes*	Medium	Yes	Medium-High
	PassThru	WiFi or wired control connection to advertised PassThru devices	No	Small	Yes	Low
	PassThru	WiFi or wired shell injection	No	Viral	Yes	Low
Short-range wireless	Bluetooth	Buffer overflow with paired Android phone and Trojan app	No	Large	Yes	Low-Medium
	Bluetooth	Sniff MAC address, brute force PIN, buffer overflow	No	Small	Yes	Low-Medium
Long-range wireless	Cellular	Call car, authentication exploit, buffer overflow (using laptop)	No	Large	Yes	Medium-High
	Cellular	Call car, authentication exploit, buffer overflow (using iPod with exploit audio file, earphones, and a telephone)	No	Large	Yes	Medium-High

Comprehensive Experimental Analyses of Automotive Attack Surfaces, Stephen Checkoway et al.

Who Needs Attackers?



Toyota settles acceleration lawsuit after $3-million verdict

Toyota heads off punitive damages after a $3-million jury verdict pointed to software defects in a fatal crash. The case could fuel other sudden acceleration lawsuits.

October 25, 2013 | By Jerry Hirsch and Ken Bensinger

LA Times

Code issues:

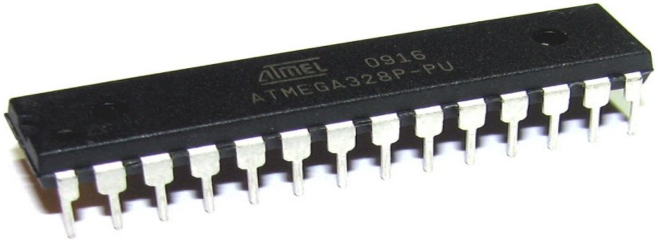
- Buffer overflows
- Unsafe casts
- Race conditions
- Recursion (makes stack analysis difficult)

Aren't These Solved Problems?

- Virtualization & sandboxes
  - E.g., Xen, Chrome Native Client
- High-level languages, powerful type systems
  - E.g., Ocaml Haskell
- Sound verification tools
  - E.g., Frama-C, CBMC

Nope.

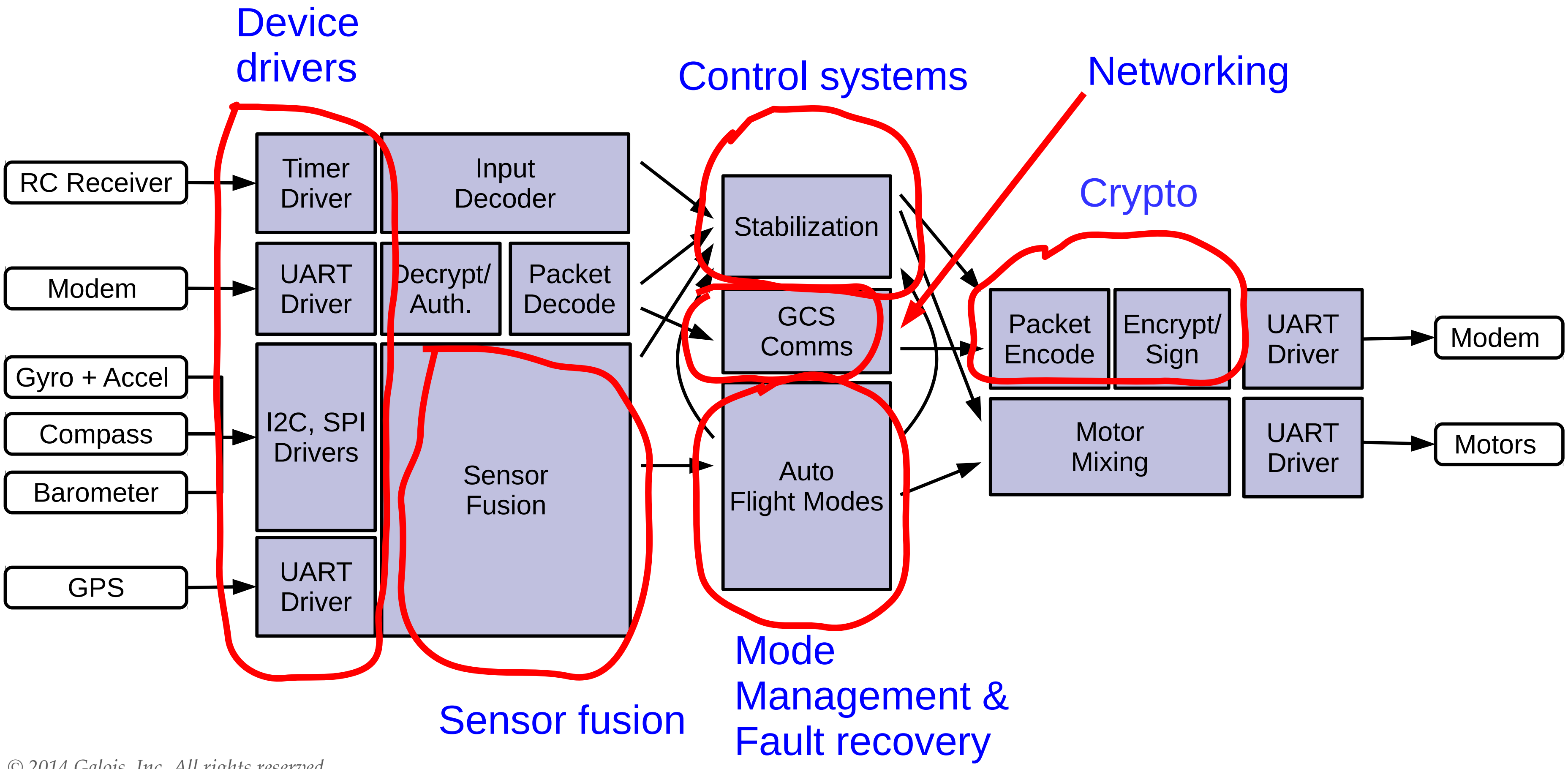
- Small, cheap hardware
  - <1MB flash, <1MB RAM, <32-bit architecture, 10s of MHz speed
  - No virtual memory
- Must control memory usage, timing
  - “Hello World” in Haskell on x86_64 requires ~1MB RAM usage, ~1MB exec
  - Can't even fit an OS sometimes
  - Unpredictable scheduling/garbage collection
- Static analysis helps, but no pancea
  - Model of libc, peripherals
  - Scaling, false-positives
  - No high-level properties,
    architectural reasoning



Heterogenous Embedded Systems: What are the properties?

Consider an autopilot:

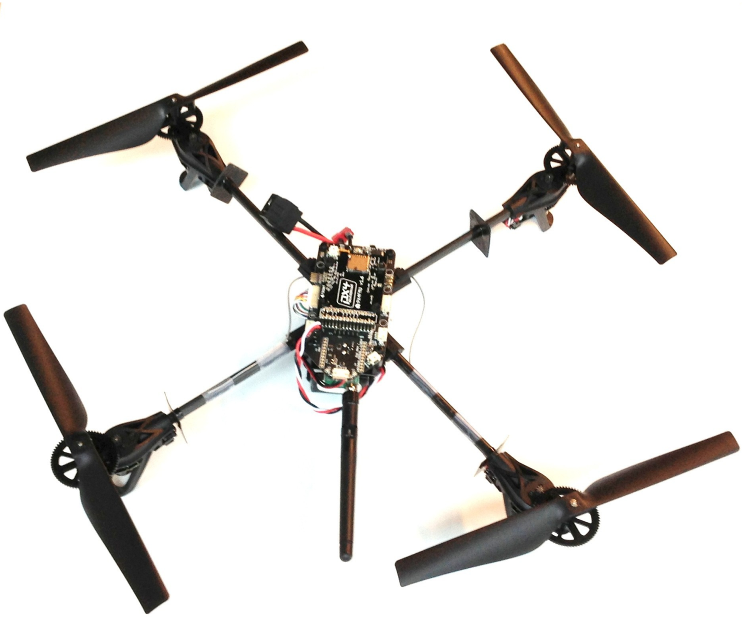
Not just different properties, different *kinds* of properties



The "Air Team"

THE A - TEAM

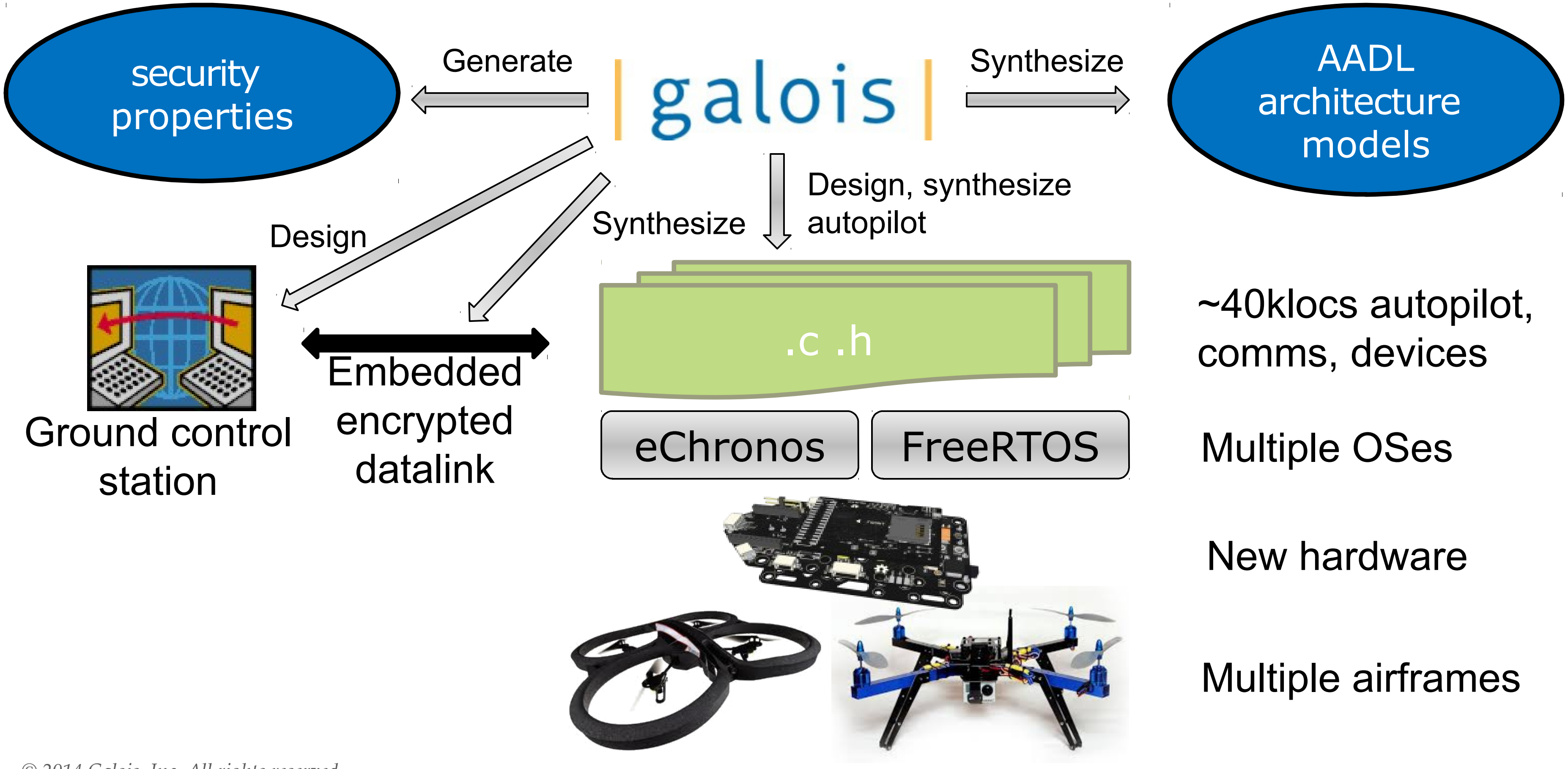
- **Boeing**: industrial-scale vehicles
- **Galois, Inc.**: research vehicle, languages
- **NICTA**: networking/operating systems



- **Rockwell Collins/Univ. Minn.**: integration and architecture
- **DRAPER/AIS/U. Oxford (Red Team)**: vulnerability analysis

The Results to Date

1.5 yrs, ~3 engineers
Need a massively more secure *and* productive approach



security properties

Generate

galois

Synthesize

AADL architecture models

Design

Synthesize

Design, synthesize autopilot

Ground control station

Embedded encrypted datalink

.c .h

eChronos

FreeRTOS

~40klocs autopilot, comms, devices

Multiple OSes

New hardware

Multiple airframes

Designing a Language for Safety and Security

- Help ensure
  - memory safety
  - timing safety (i.e., easier WCET analysis)
  - functional correctness

- While being flexible:
  - bit-data manipulation
  - memory-area manipulation
  - “escaping” to/interrop with C
  - safe user-defined abstractions
  - small and extensible
  - existing infrastructure

Memory-Safe Programming

Cyclone (AT&T, Cornell)

- Memory safety
- Garbage collection
- C look and feel

Rust (Mozilla)

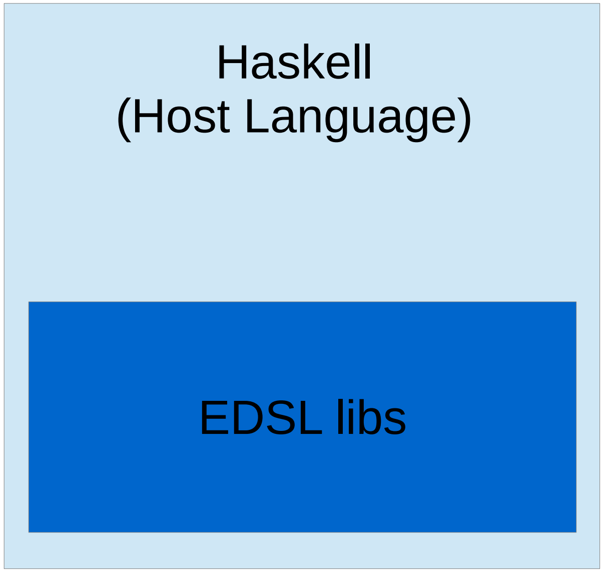
- Memory safety
- Concurrency
- C look and feel

Designing a Language for Safety and Security

- Help ensure
  - memory safety
  - timing safety (i.e., easier WCET analysis)
  - functional correctness

- While being flexible:
  - bit-data manipulation
  - memory-area manipulation
  - “escaping” to/interrop with C
  - safe user-defined abstractions
  - small and extensible
  - existing infrastructure

Embedded Domain-Specific Language



EDSL language: ~6KLOCs

- Building a new specification language is hard!
- Reduce the effort:
  - Syntax & Parser
  - Type Checker
  - Macro language is type-safe and Turing-complete

Language is “just” a powerful Haskell library

Ivory Example

Loop over an array adding `x` to each element:

Concrete Syntax

```
void mapProc(G*uint8_t[4] arr, uint8_t x) {
  map ix {
    let v = arr ! ix;
    *v = *v + x;
  }
}
```

Haskell Syntax

```
mapProc = proc "mapProc"
  $ \arr x -> body
  $ arrayMap
  $ \ix -> do
      let arrIx = arr ! ix
      v <- deref arrIx
      store arrIx (v + x)
```

Macros, Example 2

```
data Cond eff = Cond IBool (Ivory eff ())
(==>) = Cond
cond [] = return ()
cond (Cond b f : cs) = ifte_ b f (cond cs)
```

```
ifte (x >? 100)
 (store result 10)
 (ifte (x >? 50)
   (store result 5)
   (ifte (x >? 0)
     (store result 1)
     (store result 0)))
```

Type safe & for free

```
cond
  [ x >? 100 ==> store result 10
  , x >? 50  ==> store result 5
  , x >? 0   ==> store result 1
  , true     ==> store result 0
  ]
```

From Procedures to Architectures

- Goal: address the “glue code” problem: task initialization a communication
- “Just” Ivory macros so has all the type-safety guarantees o Ivory—and no new code generator!
- Also generate architectural descriptions

Ivory: What We Removed

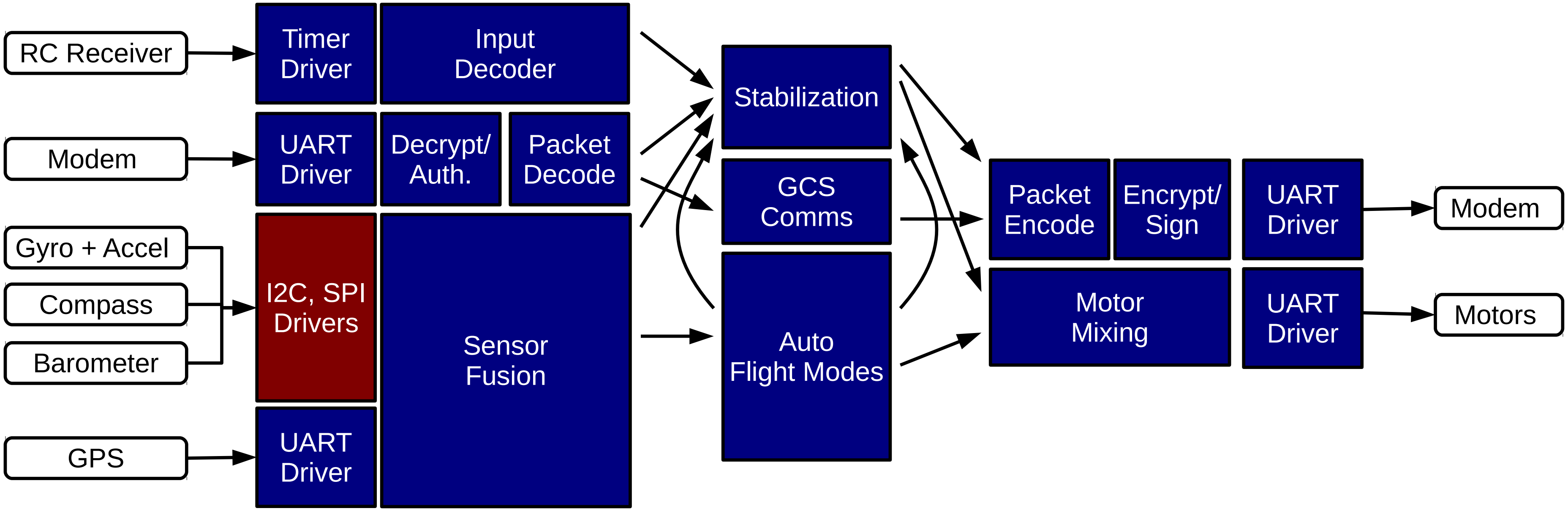
- Heap allocation
  - The stack: world's simplest collector
- Loops with user-defined termination conditions
- `void`type
- Implementation-defined size-types
- Side-effecting expressions
- Pointer arithmetic

Ivory: What We Added

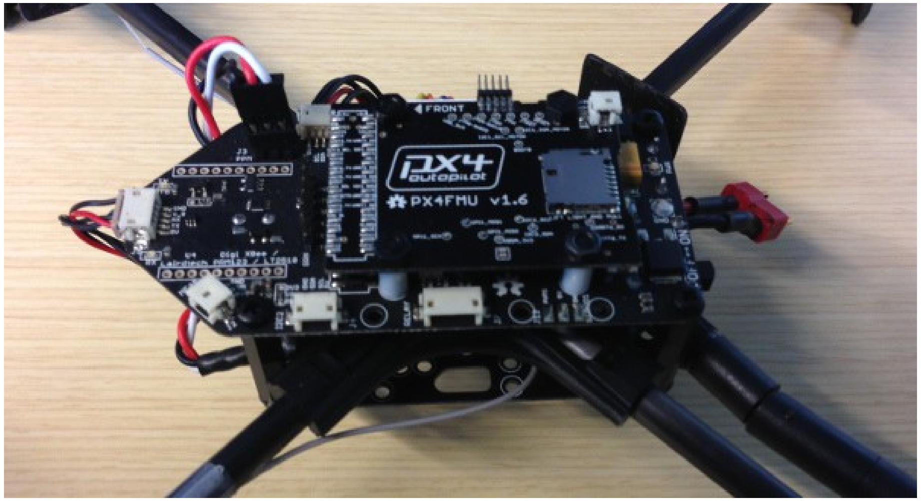
- Effect types
  - **Allocation effects:** This function can't (stack) allocate memory
  - **Escape effects:** No break is allowed in this loop
  - **Return effects:** This program fragment contains no return statement
- References (guaranteed non-null pointers)
- Array map/fold combinators
- Safe strings operators
- Safe Bit-data manipulation

SMACCMPilot

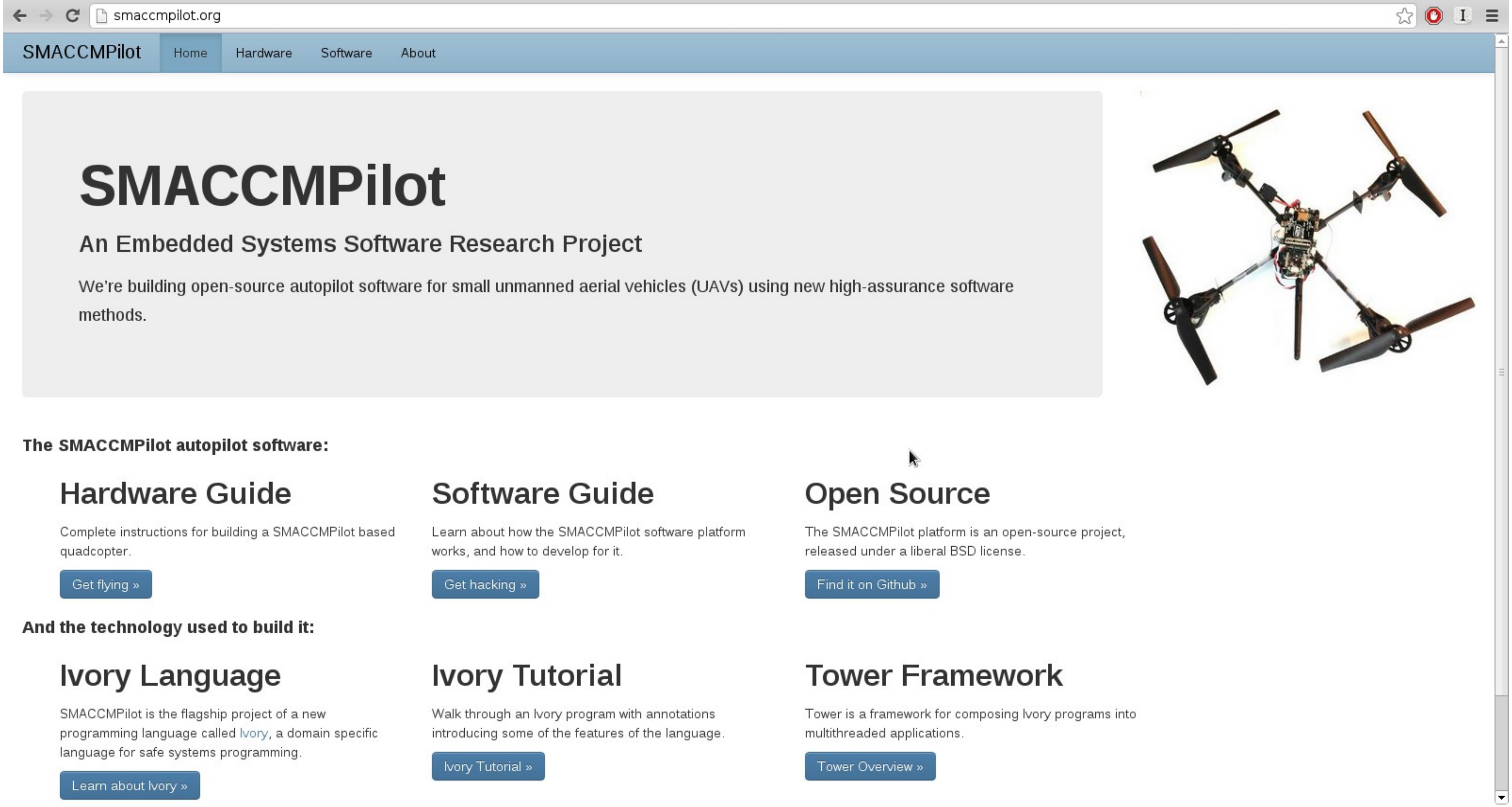
SMACCMPilot Architecture



- 15K Ivory
- 10K generated Ivory
- Generates ~45K C



smaccmpilot.org



SMACCMPilot

An Embedded Systems Software Research Project

We're building open-source autopilot software for small unmanned aerial vehicles (UAVs) using new high-assurance software methods.

The SMACCMPilot autopilot software:

Hardware Guide

Complete instructions for building a SMACCMPilot based quadcopter.

Software Guide

Learn about how the SMACCMPilot software platform works, and how to develop for it.

Open Source

The SMACCMPilot platform is an open-source project, released under a liberal BSD license.

And the technology used to build it:

Ivory Language

SMACCMPilot is the flagship project of a new programming language called Ivory, a domain specific language for safe systems programming.

Ivory Tutorial

Walk through an Ivory program with annotations introducing some of the features of the language.

Tower Framework

Tower is a framework for composing Ivory programs into multithreaded applications.

Lessons Learned

- Remove classes of bugs
  - Bugs remain, but they're the interesting ones
- Strong, static types
  - Type-checking for debug efficiency
- Small, extensible compiler
  - Instead of a growing test-suite, a growing set of checks in the compiler

Questions