Programming Languages for High-Assurance Autonomous Vehicles

Lee Pike (speaker) leepike@galois.com
Pat Hickey, James Bielman, Trevor Elliott, Erlend Hamberg, Thomas DuBuisson, Jamey Sharp, Eric Seidel

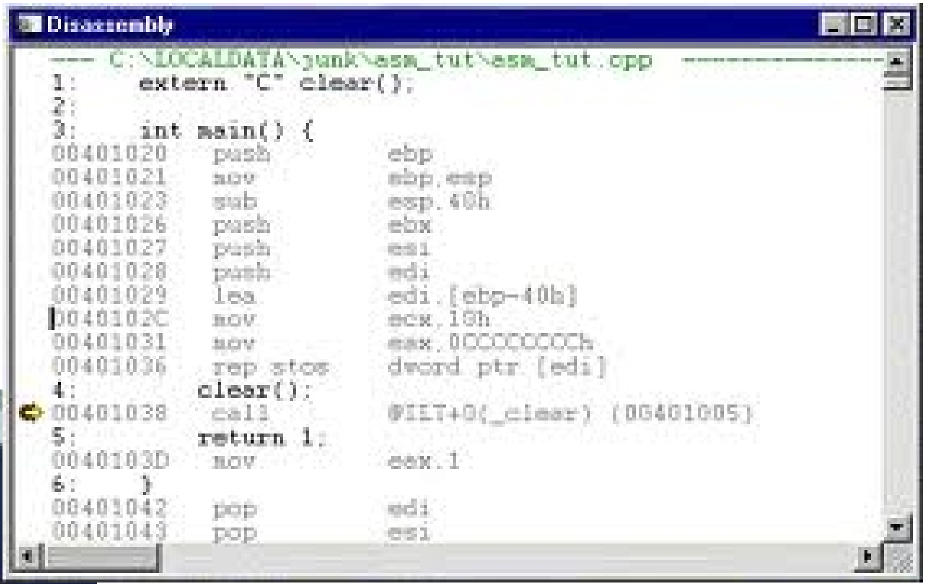
VSTTE | July 2015

Embedded Security: Where Are We At?

Embedded Programming 1970s - 2015

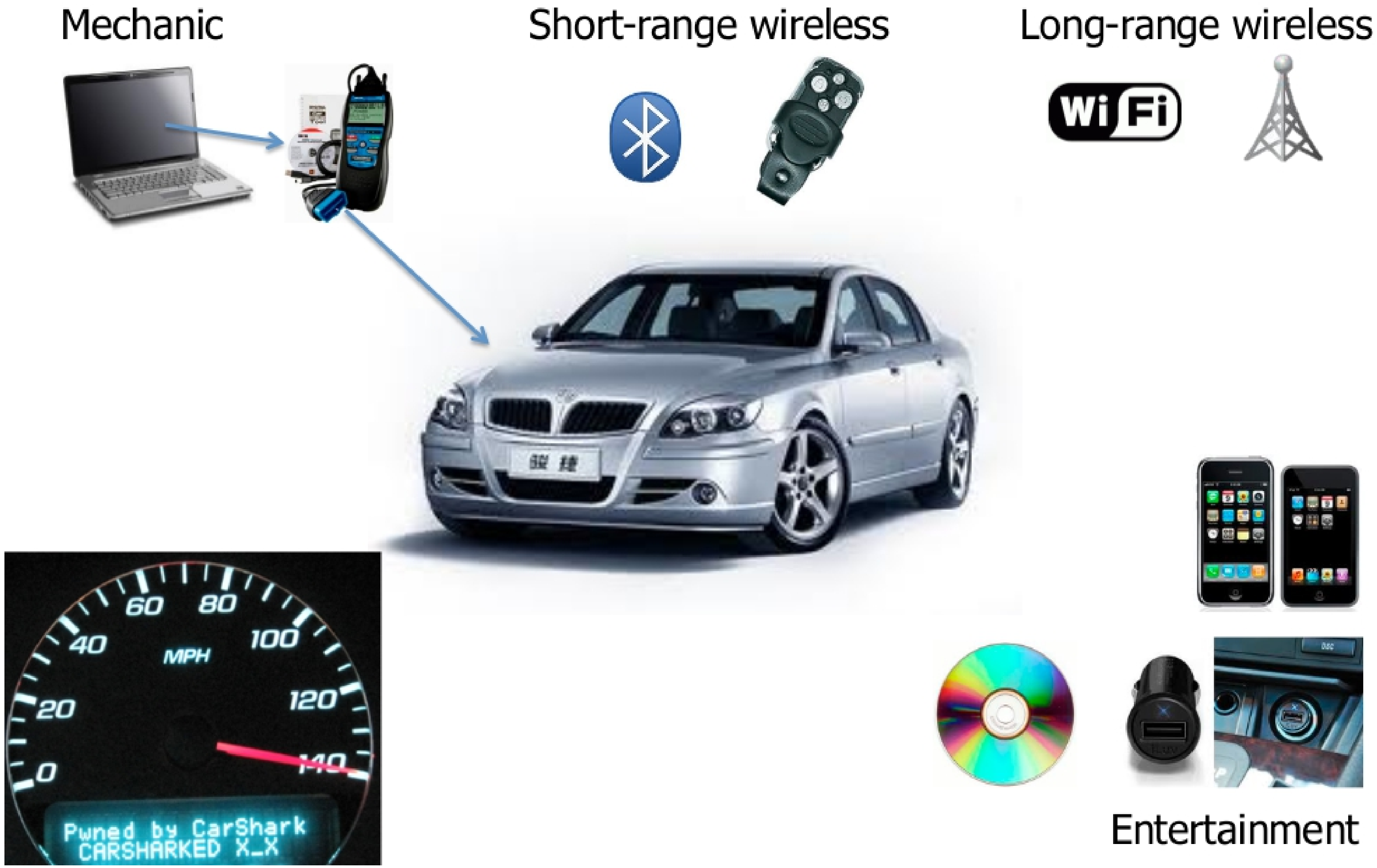
Typical tools:

- **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

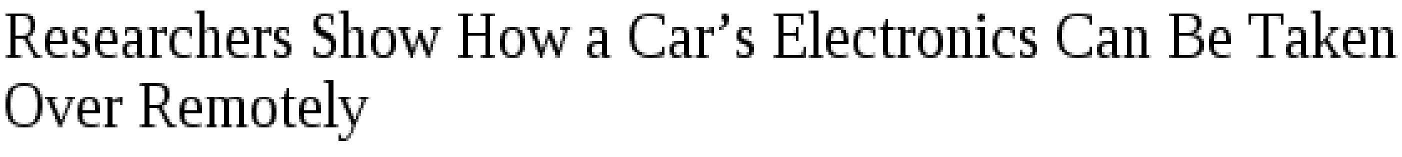
Short-range wireless

Long-range wireless

Entertainment

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

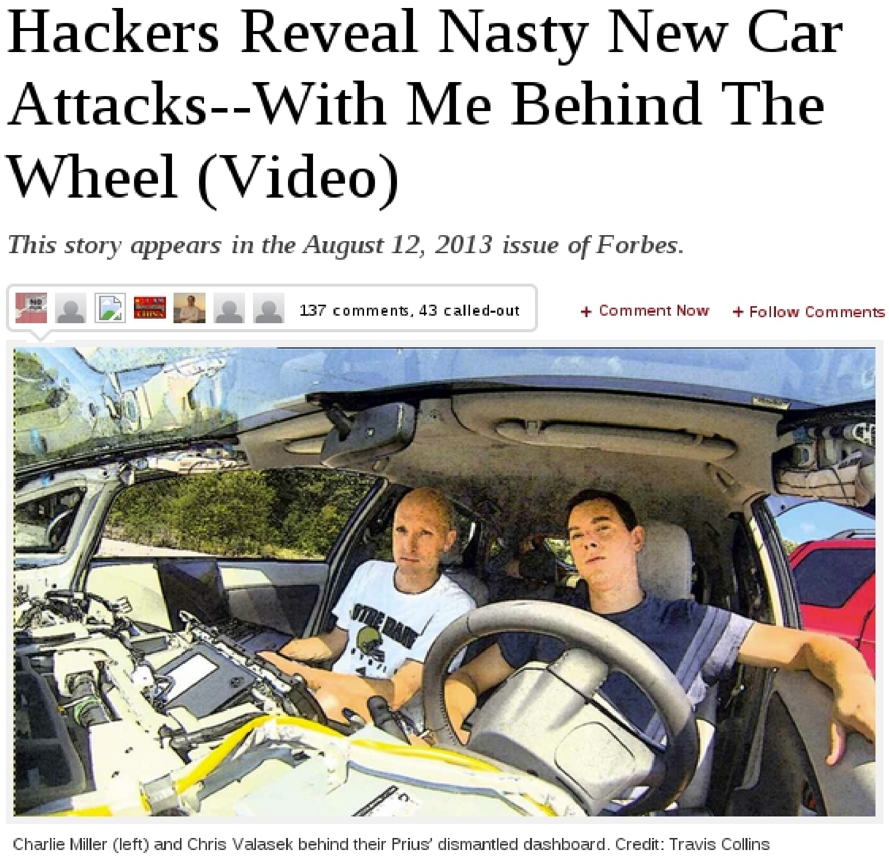
Hacking Cars



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

By JOHN MARKOFF
Published: March 9, 2011

New York Times



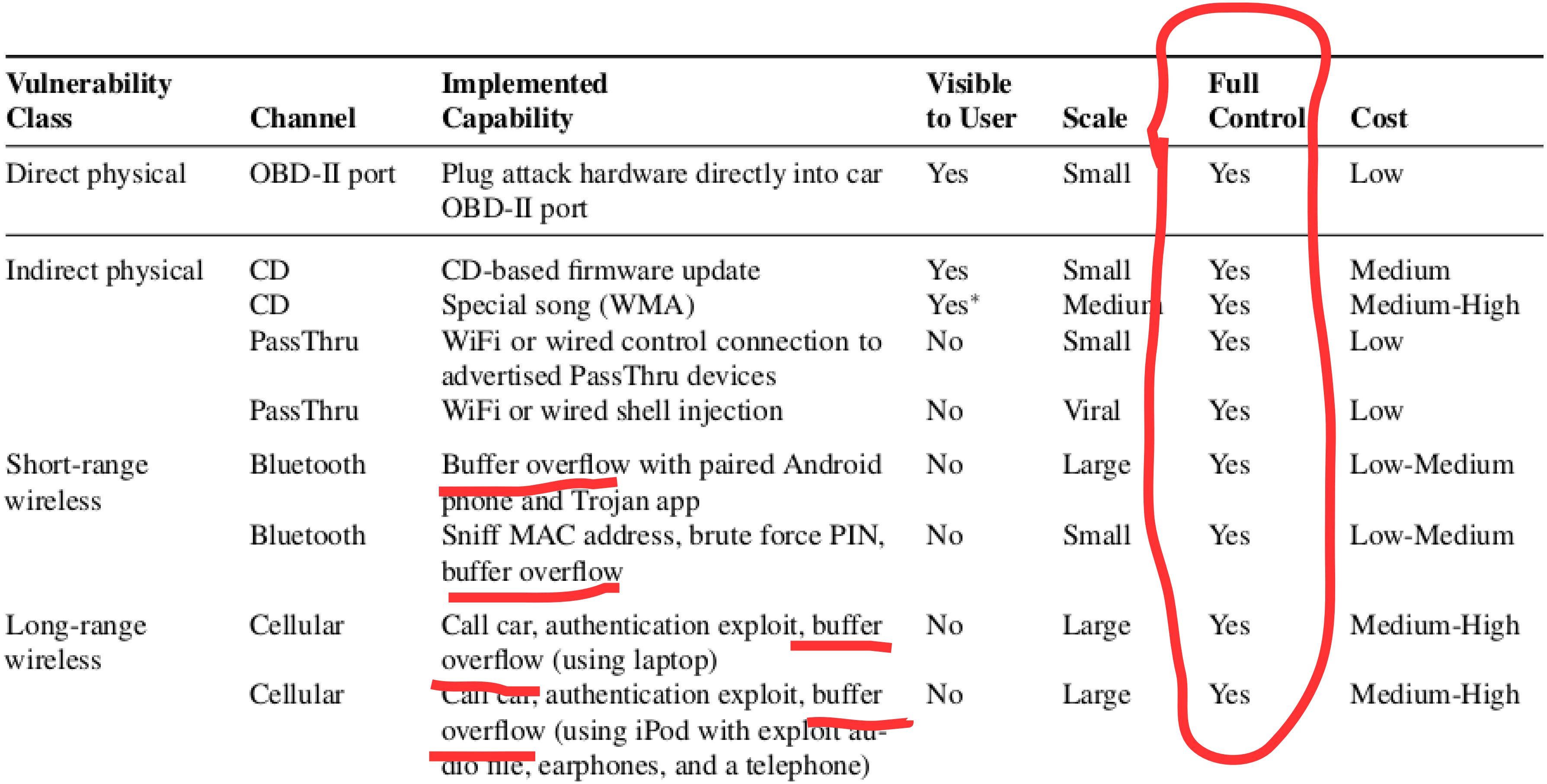
Hackers Reveal Nasty New Car Attacks--With Me Behind The Wheel (Video)

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

137 comments, 43 called-out

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

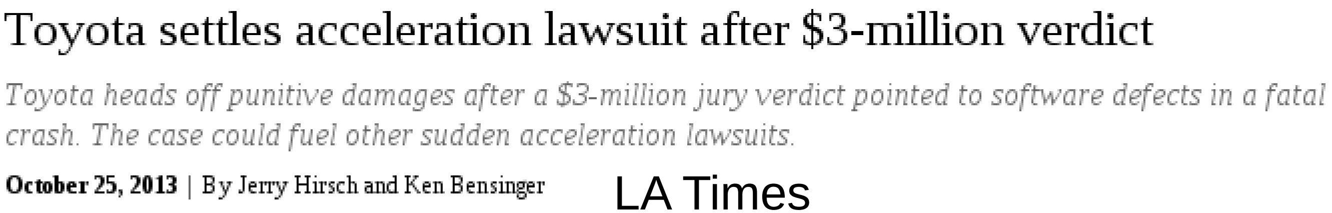
Example Attacks



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:

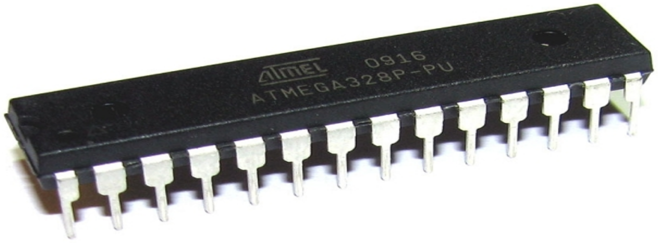
- Buffer overflows
- Unsafe casts
- Race conditions
- Recursion (makes stack analysis difficult)

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, Coq

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
- Too complex for post-hoc verification
  - Model of libc, ASM
  - Concurrency

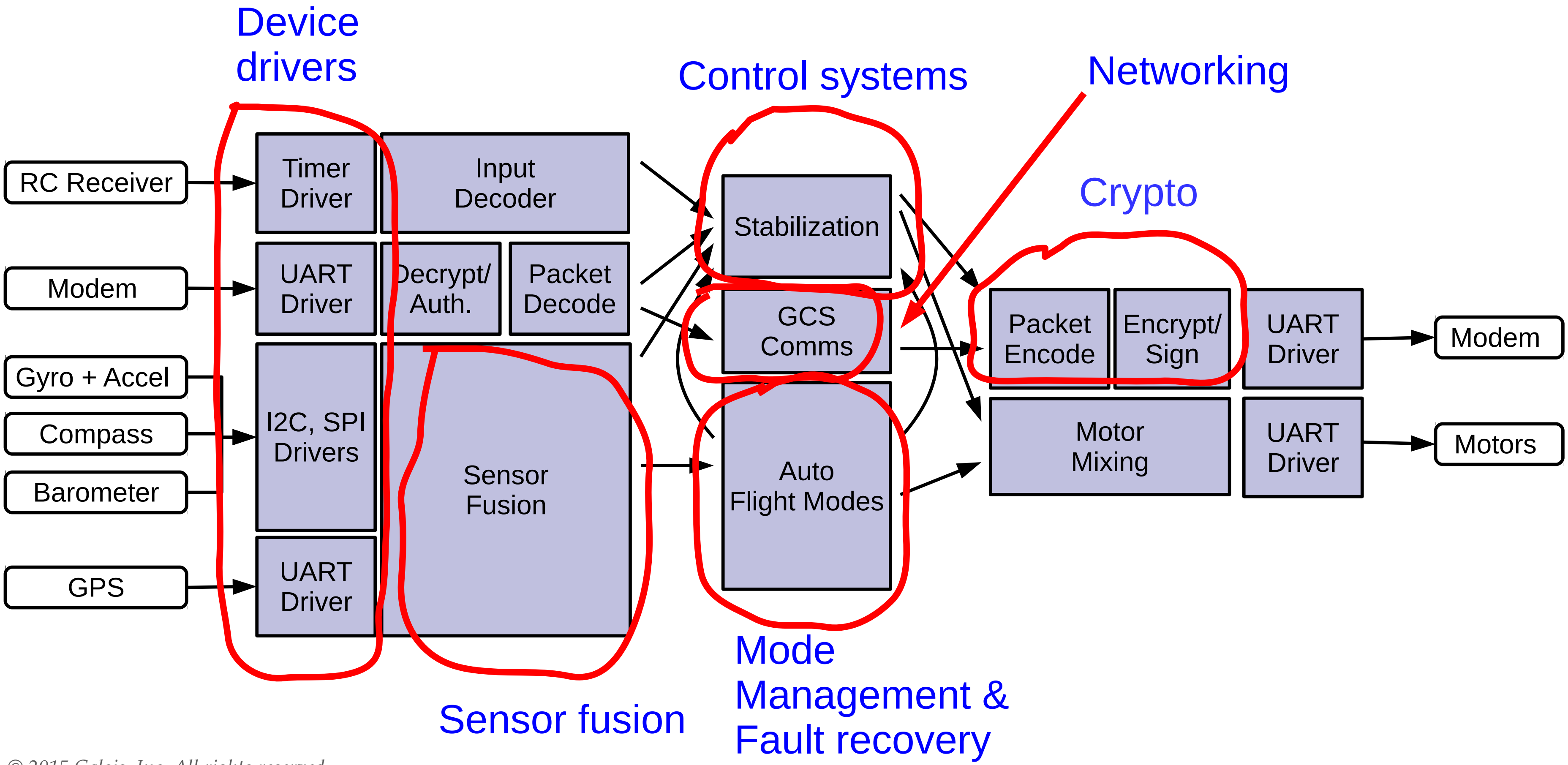


Heterogenous Embedded Systems: What are the properties?

|galois|

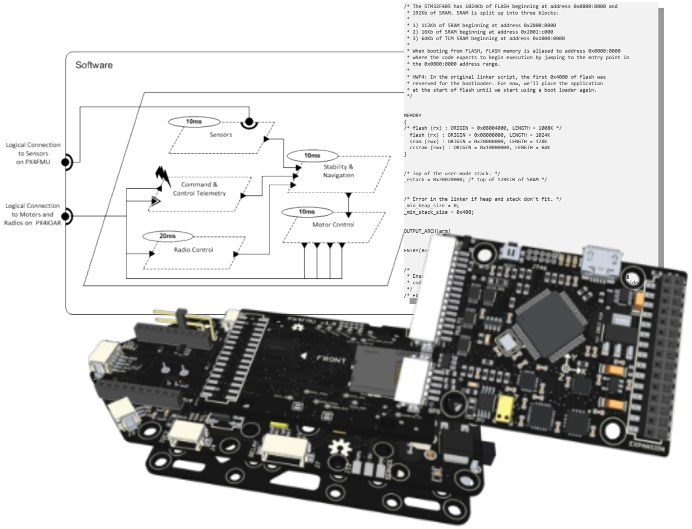
Consider an autopilot:

Not just different properties, different *kinds* of properties



Air Team Platforms

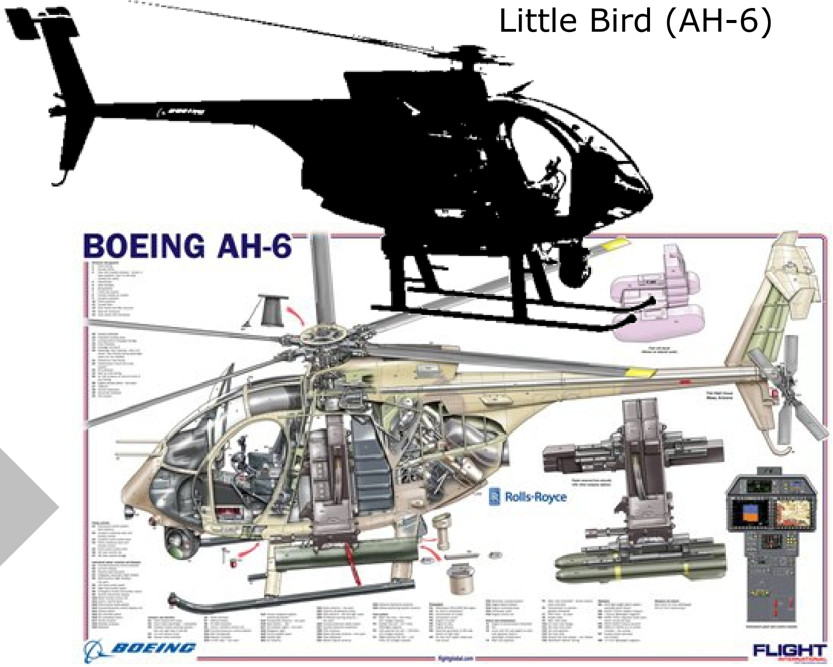
AR.drone &
ArduCopter
(SMACCMcopter)



Software
Logical Connection to Sensors on PX4FMU
Logical Connection to Motors and Radios on PX4IOAR
10ms
Sensors
10ms
Stability & Navigation
Command & Control Telemetry
10ms
Motor Control
20ms
Radio Control
FRONT

```
/* The STM32F405 has 1024Kb of FLASH beginning at address 0x0800:0000 and
 * 192Kb of SRAM. SRAM is split up into three blocks:
 *
 * 1) 112Kb of SRAM beginning at address 0x2000:0000
 * 2) 16Kb of SRAM beginning at address 0x2001:c000
 * 3) 64Kb of TCM SRAM beginning at address 0x1000:0000
 *
 * When booting from FLASH, FLASH memory is aliased to address 0x0000:0000
 * where the code expects to begin execution by jumping to the entry point in
 * the 0x0800:0000 address range.
 *
 * HWF4: In the original linker script, the first 0x4000 of flash was
 * reserved for the bootloader. For now, we'll place the application
 * at the start of flash until we start using a boot loader again.
 */

MEMORY
{
/* flash (rx) : ORIGIN = 0x08004000, LENGTH = 1008K */
  flash (rx) : ORIGIN = 0x08000000, LENGTH = 1024K
  sram (rwx) : ORIGIN = 0x20000000, LENGTH = 128K
  ccsram (rwx) : ORIGIN = 0x10000000, LENGTH = 64K
}

/* Top of the user mode stack. */
_estack = 0x20020000; /* top of 128KiB of SRAM */

/* Error in the linker if heap and stack don't fit. */
_min_heap_size = 0;
_min_stack_size = 0x400;

OUTPUT_ARCH(arm)

ENTRY(Re

/*
 * Ens
 * cod
 */
/* EX
```

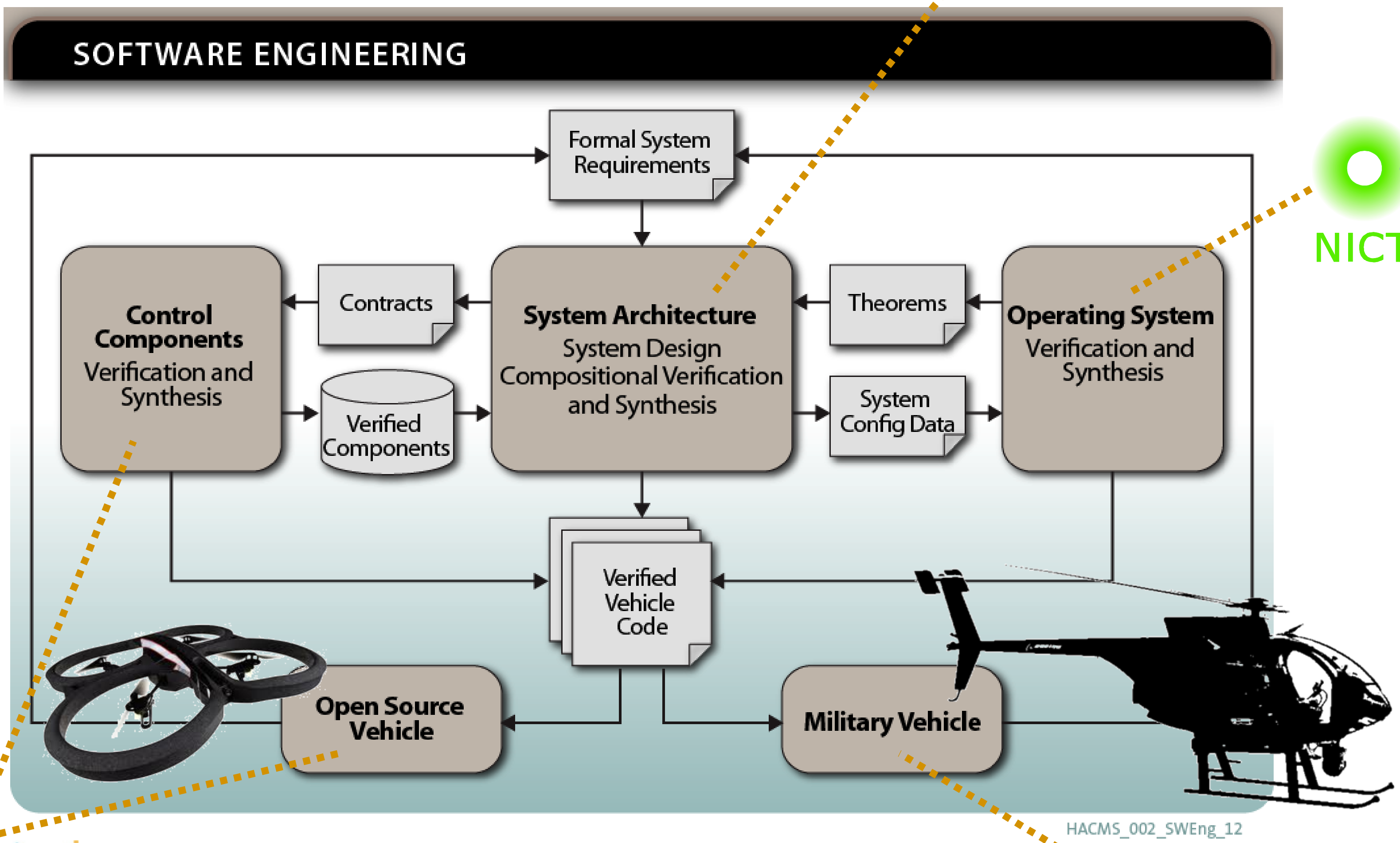
Boeing Unmanned
Little Bird (AH-6)



New electronics to
host provably secure
software

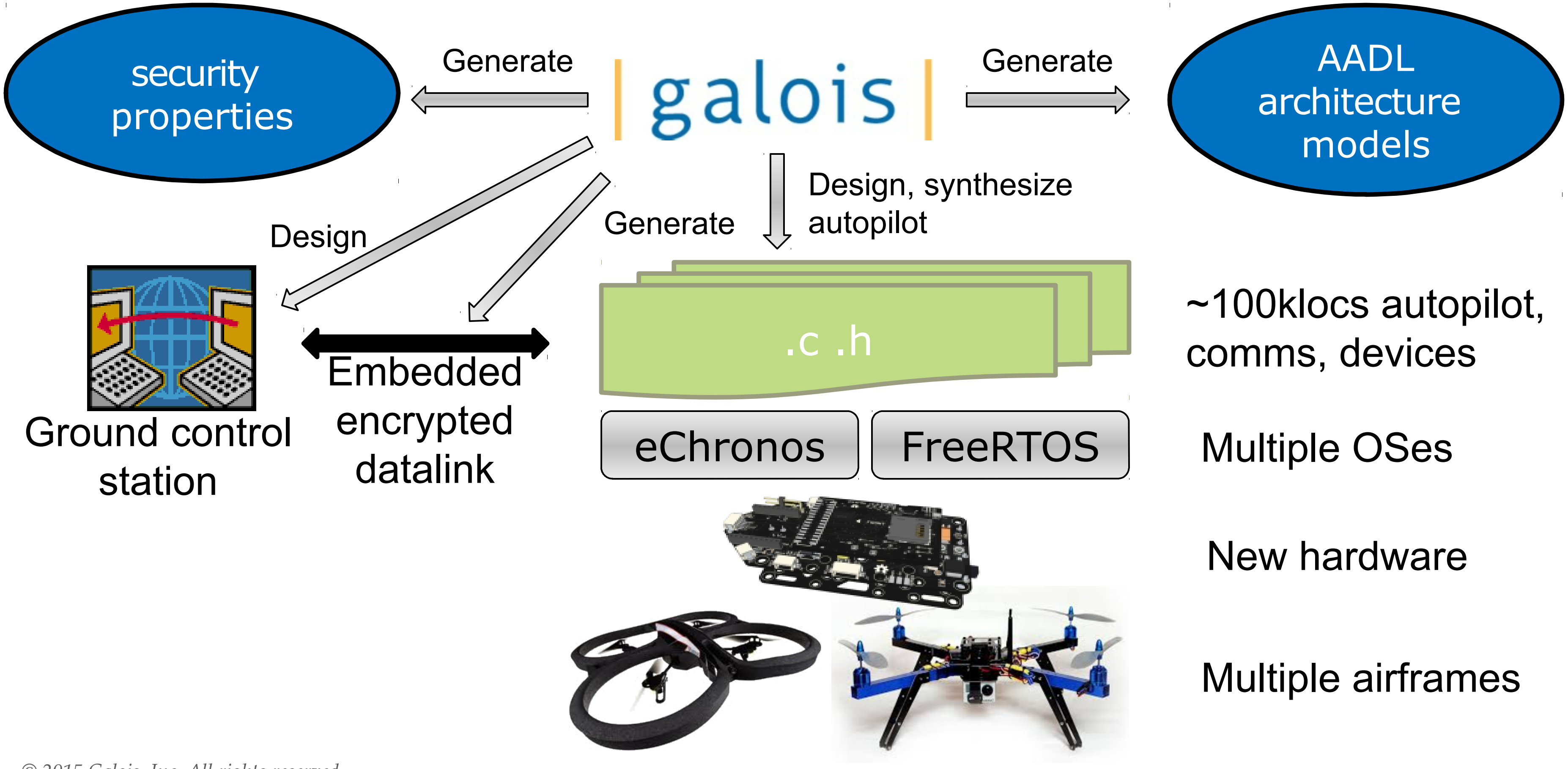
src: Rockwell Collins

Architecture-Driven Assurance



src: Rockwell Collins

“Most Secure UAV in the World”



security properties

Generate

galois

Generate

AADL architecture models

Design

Generate

Design, synthesize autopilot

Ground control station

Embedded encrypted datalink

.c .h

eChronos

FreeRTOS

~100klocs autopilot, comms, devices

Multiple OSes

New hardware

Multiple airframes

Languages for Secure Embedded Systems

The Problem(s) With C

- Memory unsafe
- Undefined behavior everywhere!
  - Dereferencing, arithmetic, casting, etc.
- Implementation-defined behavior everywhere!
  - Type sizes, signed/unsigned types, bit-fields, type-punning, etc.

Even *Defined* C is Problematic

Distilled ArduPilot bug discovered by Galois:

```
...
uint8_t a = 10;
uint8_t b = 250;
printf("Answer: %i, %i", a-b > 0, (uint8_t)(a-b) > 0);
...
```

Answer: 0, 1
Assuming `int > uint8_t`

JPL's “Power of 10” Rules

1. Simple control flow (no `setjmp`, `longjmp`, etc)
2. Loops with fixed upper bounds
3. No dynamic memory (after allocation)
4. Short functions
5. >= 2 assertions per function
6. Data objects in smallest scope
7. Check return vals/args

   (e.g., `printf`, `strlen(0)`)
8. Limit pre-processor
9. Limit pointer usage (one level of indirection, no func pointers)
10. All compiler warnings are errors

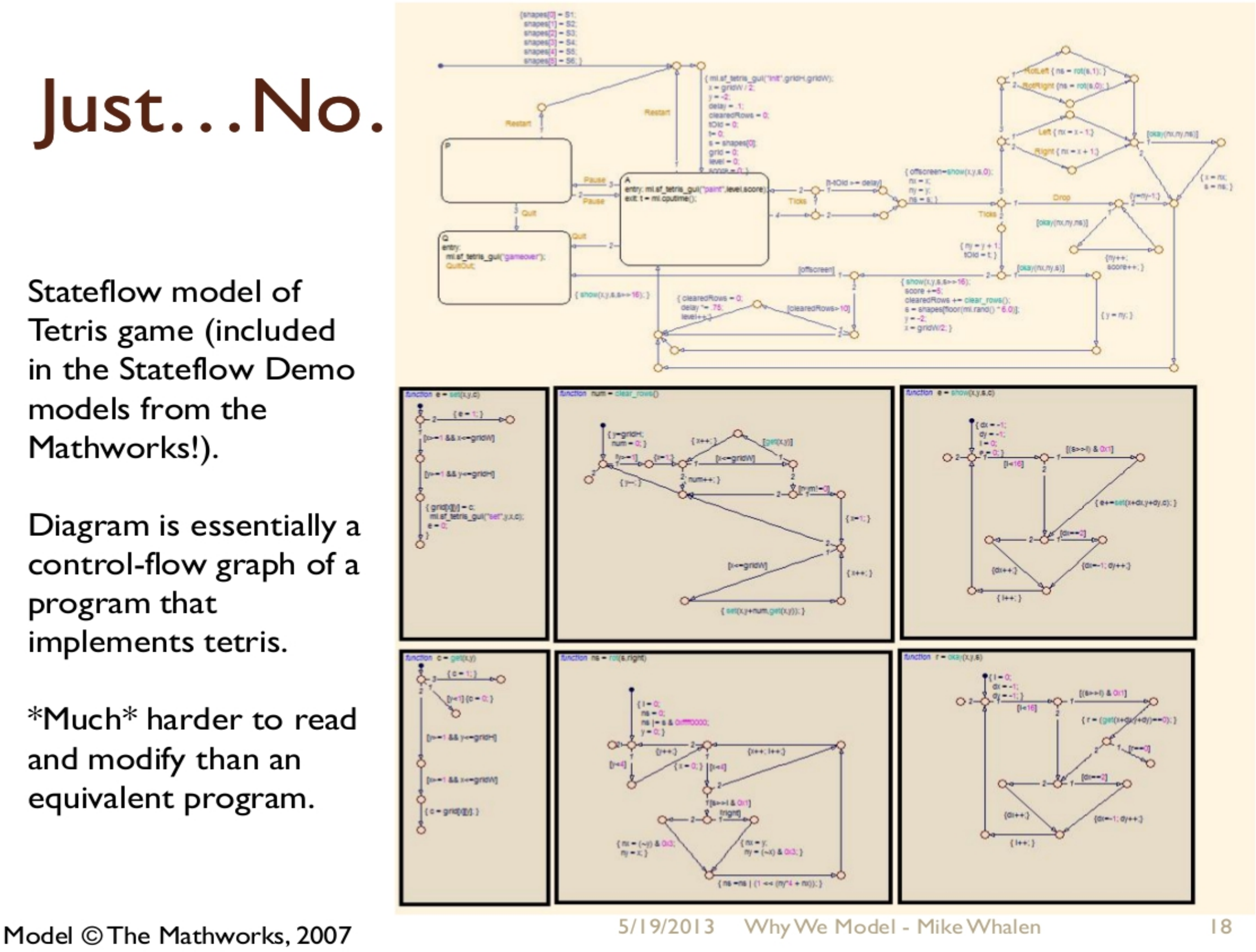
From convention to enforcement

src: http://spinroot.com/gerard/pdf/P10.pdf

Safe Low-Level Programming

- Option #1: model-based development

As Mike Whalen put it...



Just…No.

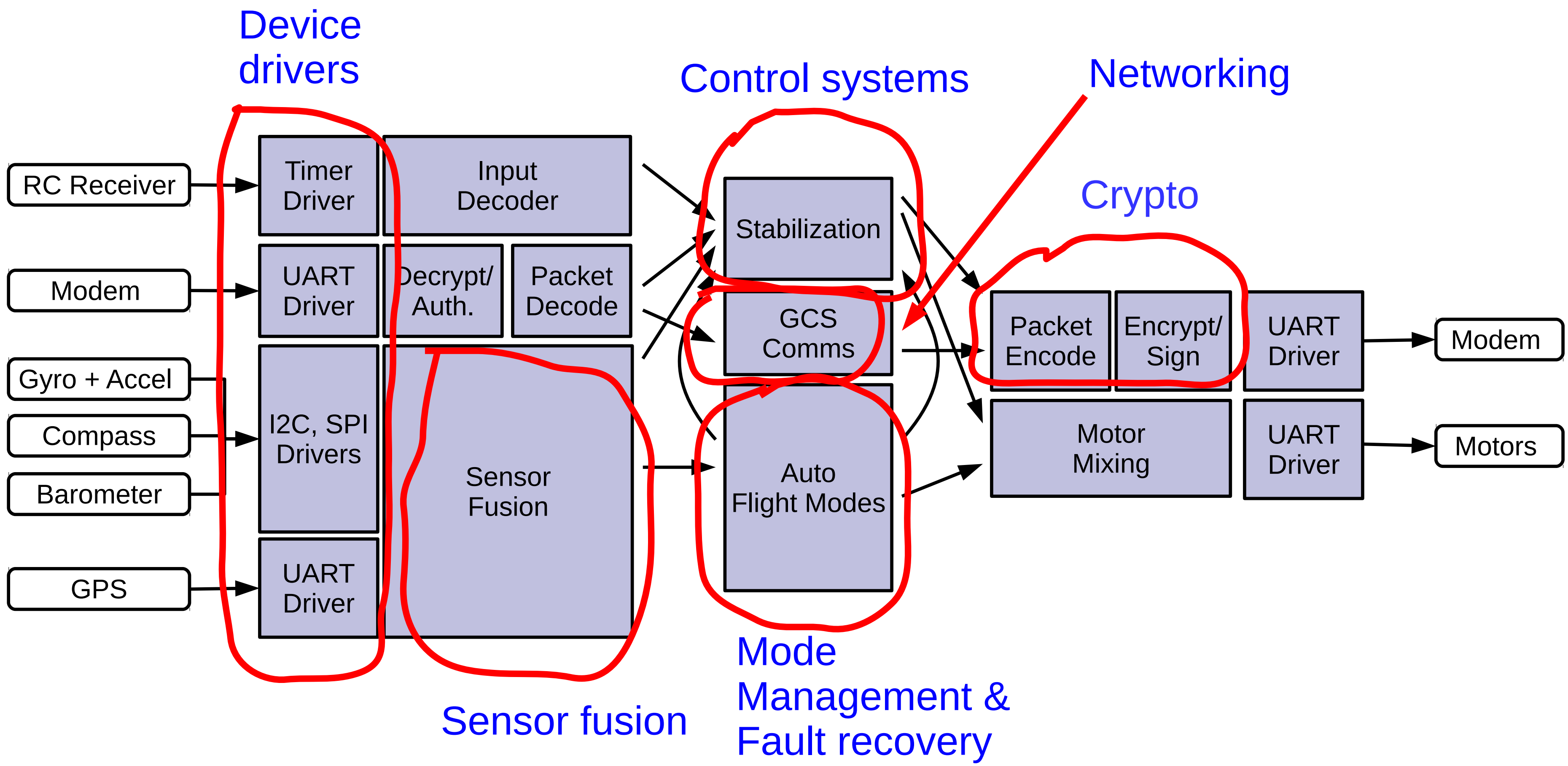
Stateflow model of Tetris game (included in the Stateflow Demo models from the Mathworks!).

Diagram is essentially a control-flow graph of a program that implements tetris.

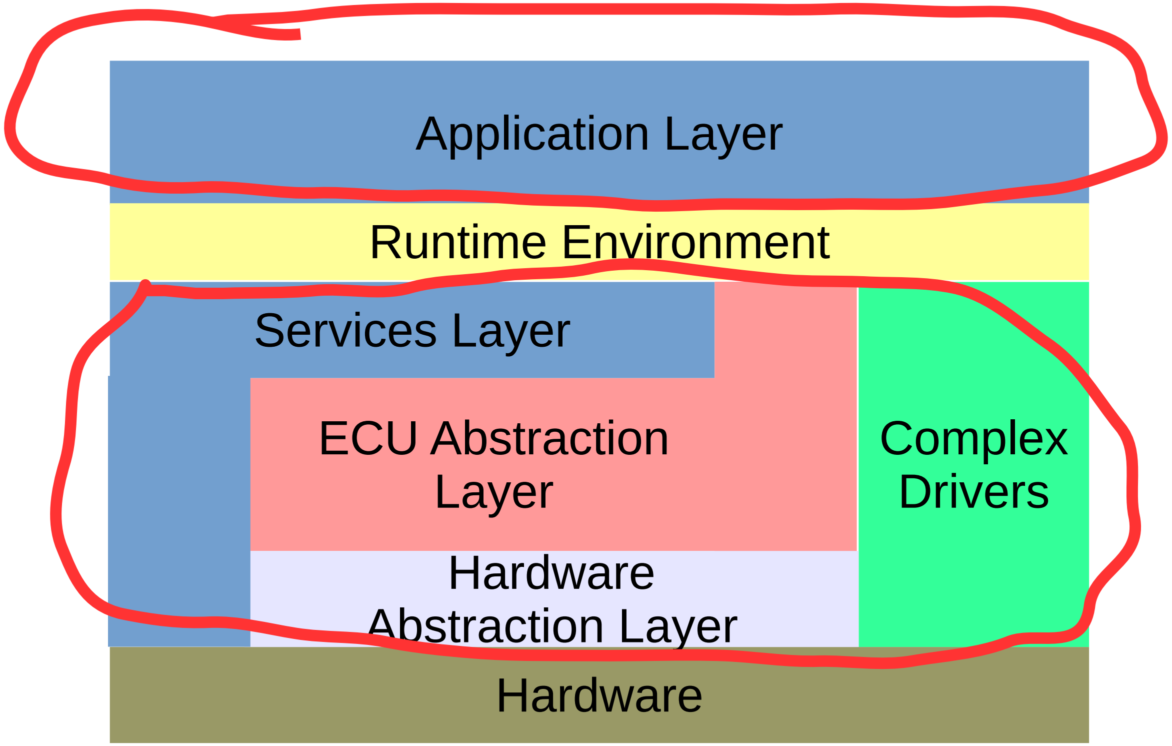
Much harder to read and modify than an equivalent program.

Model © The Mathworks, 2007

5/19/2013 Why We Model - Mike Whalen 18


Device drivers
Control systems
Networking
Crypto
RC Receiver
Modem
Gyro + Accel
Compass
Barometer
GPS
Timer Driver
Input Decoder
UART Driver
Decrypt/ Auth.
Packet Decode
I2C, SPI Drivers
Sensor Fusion
UART Driver
Stabilization
GCS Comms
Auto Flight Modes
Packet Encode
Encrypt/ Sign
UART Driver
Motor Mixing
UART Driver
Modem
Motors
Sensor fusion
Mode Management & Fault recovery

Software Stack (AUTOSAR)



Model-Based Design
(Simulink/Stateflow)

Basic Software Layer
(C/C++)

Safe Low-Level Programming

- Option #1: model-based development
- Option #2: *a posterori* verification

Safe Low-Level Programming

- Option #1: model-based development
- Option #2: *a posterori* verification
- Option #3: synthesis from a specification language

Or... model-based design for embedded systems

Haskell

- Strong, static, polymorphic type checking and inference
- Pure, higher-order language—no side effects
- Functional programming for modularity: program composition is function composition

Why Functional Programming Matters by John Hughes (1990)

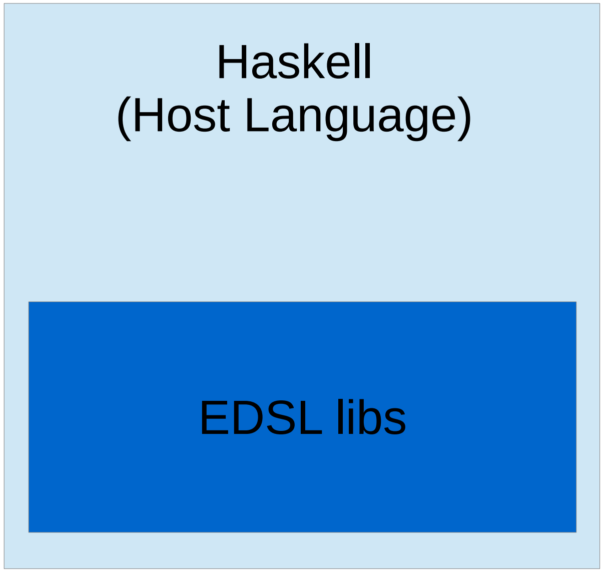
What if...

Can we have the high-level abstractions and type-safety of functional programming in embedded systems programming?

Approaches:

- Design a new FP-inspired language/compiler from scratch? **No**:
  - Would take too long
  - No library support
- Take the Haskell/Ocaml compiler and pair it down? **No**:
  - The runtime system is 50KLOCs of C/C--
  - Issues with timing, code size, etc.

Embedded Domain-Specific Language



Haskell
(Host Language)

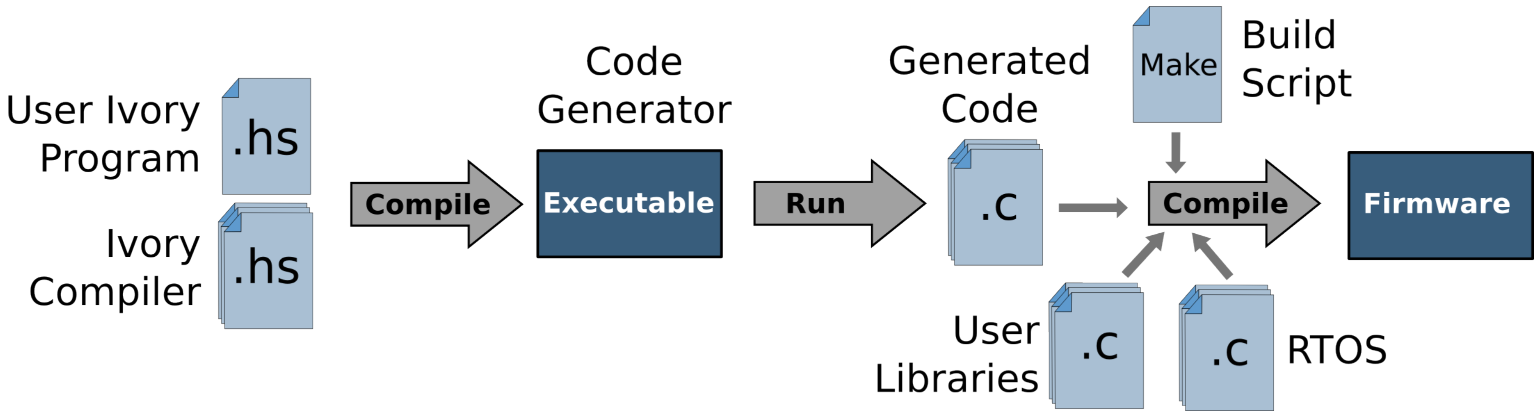
EDSL libs

EDSL language: ~10KLOCs
C backend: ~1.5KLOCs

- 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

Compiling and Running an EDSL

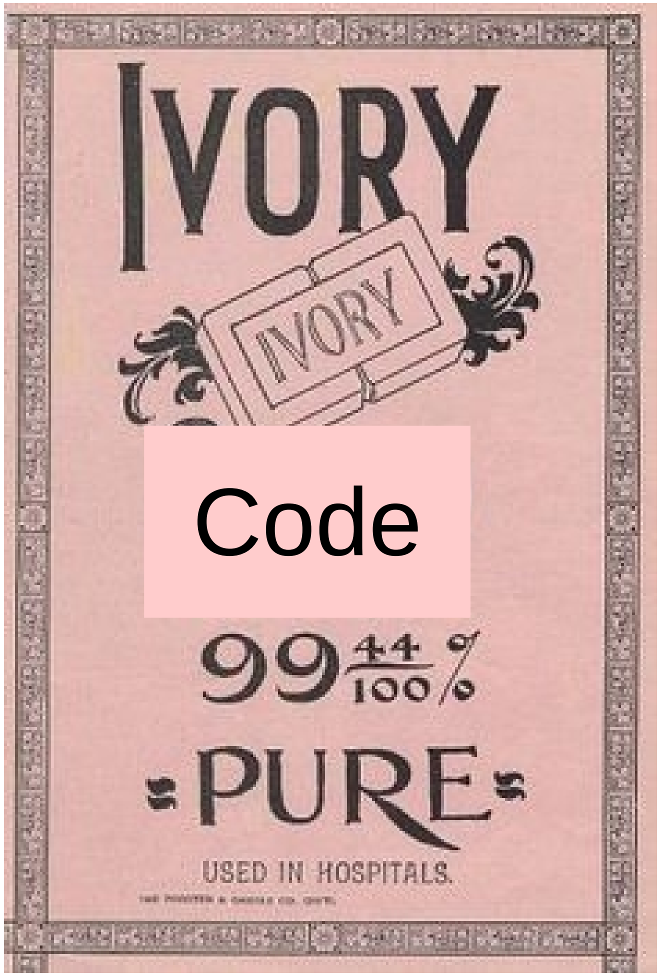


Who's Used EDSLs?

- Eaton: garbage truck controllers
- Boeing: component configuration
- Ericsson: DSP
- Xilinx: FPGA synthesis
- Soostone: high-speed trading
- ...

Ivory

- Haskell-based EDSL for embedded software
- High-level functional programming for low-level programming
- Major features:
  - Verification tool integration (SMT, ACL2, AADL)
  - Haskell as a macro language
  - Improved safety for low-level programming



Ivory: What We Added (compared to C)

- Effects
  - **Allocation effects:** This program can't (stack) allocate memory
  - **Escape effects:** No break is allowed in this program
  - **Return effects:** This program contains no return statement
- References (guaranteed non-null pointers)
- Array map/fold combinators
- Safe string operators (don't depend on null termination)
- Safe bit-data manipulation

Ivory: What's Missing (from C)

- Arbitrary heap allocation
  - The stack: world's simplest collector
- Arbitrary loops
- Pointers (replaced with references)
- Implementation-defined size-types
- Side-effecting expressions
- Most undefined behavior

Ivory Example

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

```
arrayExample :: Def('[ Ref s (Array 4 (Stored Uint8))
                     , Uint8
                     ] :-> ())
arrayExample = proc "arrayExample"
  $ \arr x -> body
  $ arrayMap
  $ \ix -> do
      v <- deref (arr ! ix)
      store (arr ! ix) (v + x)
```

Type automatically inferred

Map over the elements of the array

Guaranteed dereference `arr` at `ix`

Store `v+x` at index `ix`

Ivory's C-Like Syntax

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;
  }
}
```

C

```
void mapProc (uint32_t arr[], int len, uint32_t x) {
  for(int ix = 0; ix < len; ix++) {
    uint32_t v = arr[ix];
    arr[ix] = v + x;
  }
}
```

Syntax Matters!

- Working with Boeing to rewrite their comms stack in Ivory
- Stanag 4586 Levels of Interoperability
- Fairly direct translation of the C++ (~1kloc)

Type-Safe Macro Languages (1)
Language Extensions as Macros

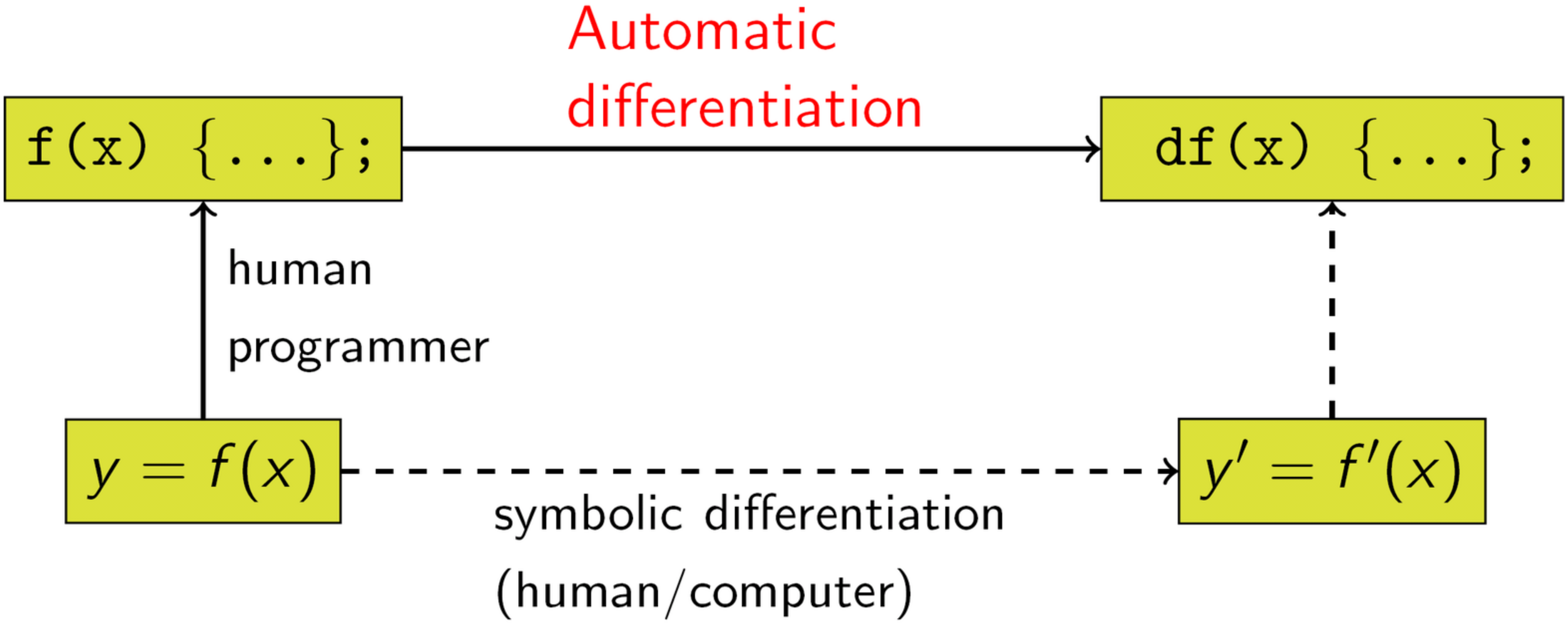
```
data Cond eff = Cond IBool (Ivory eff ())
(==>)  = Cond
cond [] = return ()
cond (Cond b f : cs) = ifte_ b f (cond cs)
```

```
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
  ]
```

Type-Safe Macro Languages (2)
AST Computations



Automatic differentiation
f(x) {...};
df(x) {...};
human programmer
$y = f(x)$
$y' = f'(x)$
symbolic differentiation (human/computer)

src: https://en.wikipedia.org/wiki/File:AutomaticDifferentiationNutshell.png

Automatic Differentiation

$$u \rightarrow u_v = \langle u,\ u' \rangle$$

- $u_v + v_v = \langle u + v,\ u' + v' \rangle$
- $\sin(u_v) = \langle \sin u,\ u' * \cos(u) \rangle$ (chain rule)
- …

Type-Safe Macro Languages (2)
AST Computations

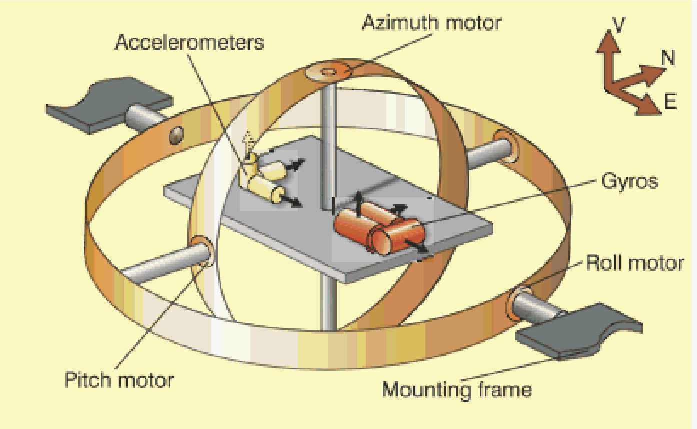
|galois|

```
class Num a where
   (+) :: a → a → a
   ...
```

```
instance Num IvoryExp where
   (+) e0 e1 = PlusExp e0 e1
   ...
```

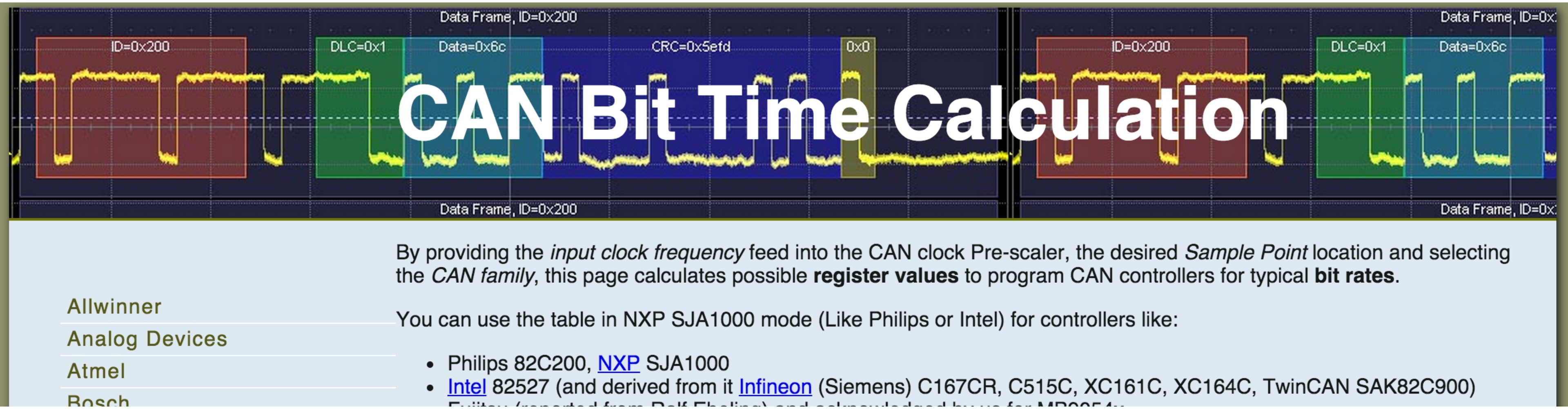
```
jacobianMatrixAD :: (..., Num a) => …
```

Upshot: inertial navigation
- defined in 100s of LOCs
- generates 10x C LOCs



src: https://leagueofextraordinarytechnicians.wikispaces.com/Inertial+Navigation+Systems+-+Operation+%26+Testing

Type-Safe Macro Languages (3)
Driver Constraints



CAN Bit Time Calculation

By providing the *input clock frequency* feed into the CAN clock Pre-scaler, the desired *Sample Point* location and selecting the *CAN family*, this page calculates possible **register values** to program CAN controllers for typical **bit rates**.

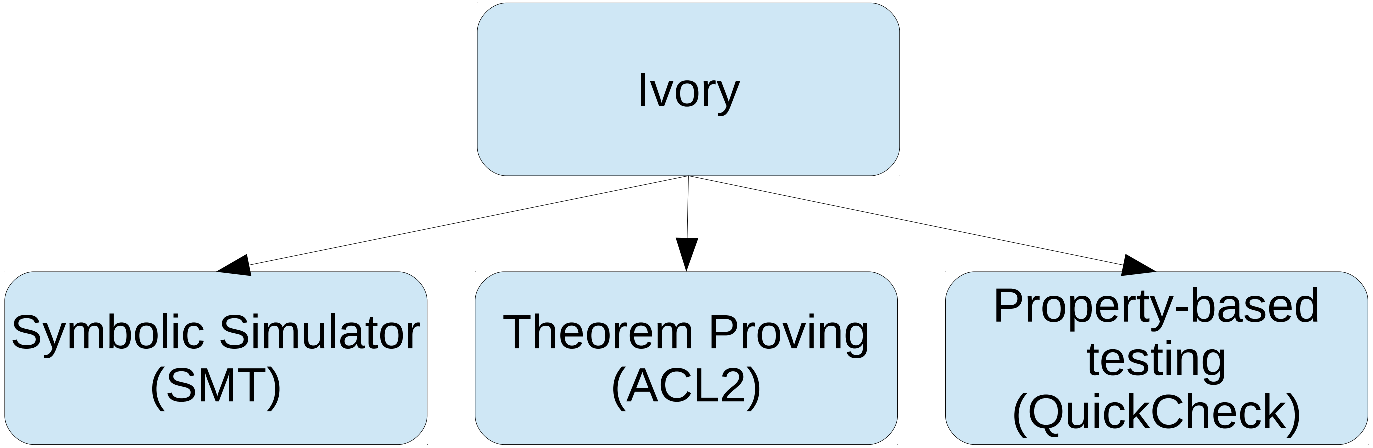
You can use the table in NXP SJA1000 mode (Like Philips or Intel) for controllers like:

- Philips 82C200, NXP SJA1000
- Intel 82527 (and derived from it Infineon (Siemens) C167CR, C515C, XC161C, XC164C, TwinCAN SAK82C900)
- Fujitsu (reported from Ralf Ebeling) and acknowledged by us for MB9054x

src: http://www.bittiming.can-wiki.info/

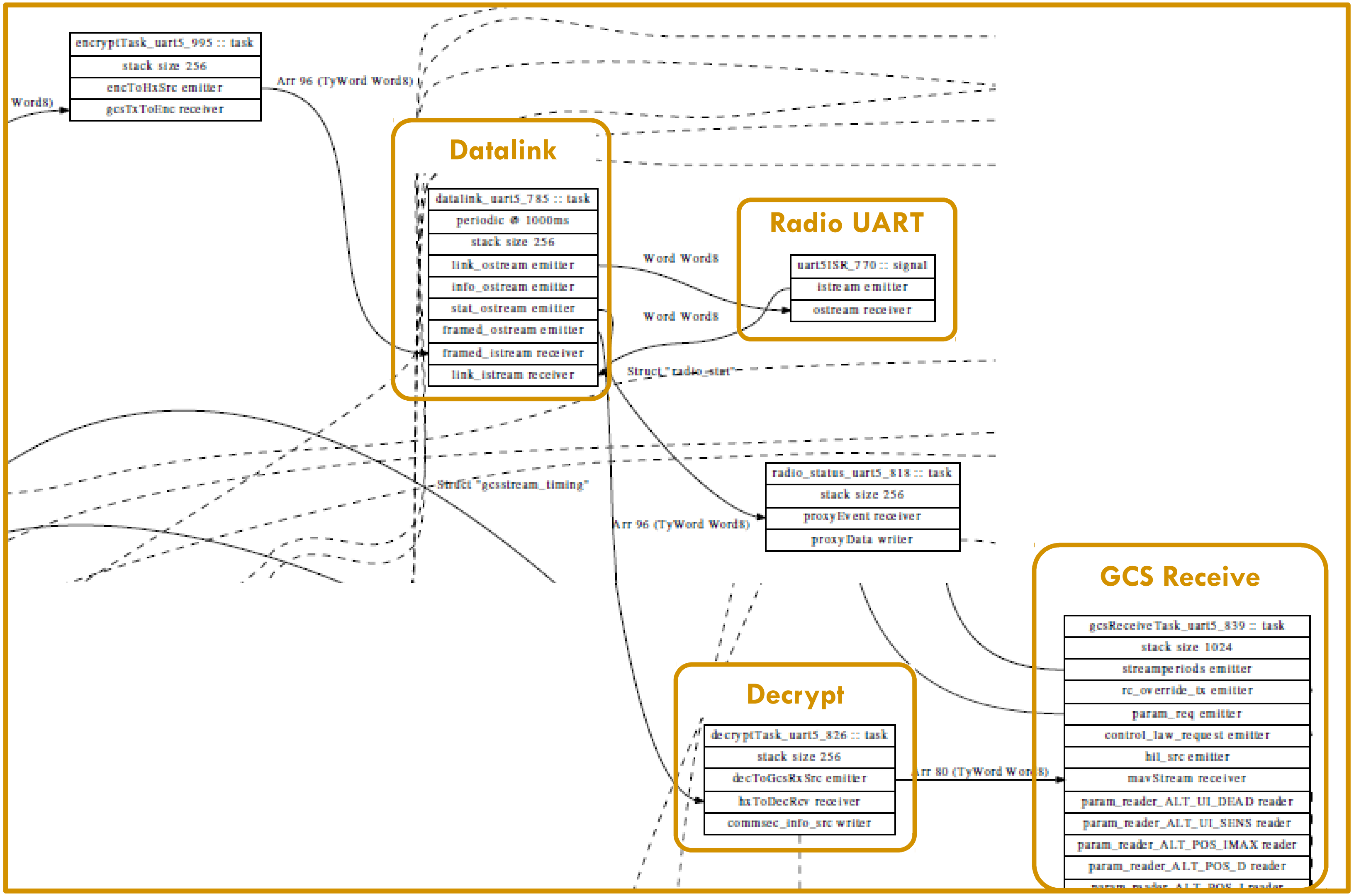
```
legalTimings clk bitrate =
  [ t | baud_prescalar ← [1..1024]
      , constraints...
  ]
```

Integrated modelling, testing, verification



Safe Concurrency

The Complexity of Concurrency



From Procedures to Architectures

Problems:

- We've got safe procedures, but what about concurrency?
- Glue code: boilerplate C for system calls, IPC, task initialization

From Procedures to Architectures

- Assume an underlying scheduler, locking, message passing
- “Just” Ivory macros so has all the type-safety guarantee of Ivory—and no new code generator!
- Also generate architectural descriptions
- Our architecture EDSL is called Tower

Concurrency Model

- Lock free thread concurrency

  No locks specified by user (implemented by backends)

- Shared-state concurrency

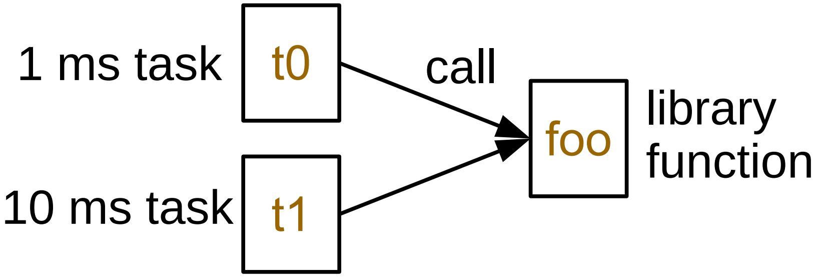
Monitors

- Monitor: thread-safe object:
  - Shared state *S*
  - Collection of handlers
  - Monadic, composable
- Handler: Given
  - Incoming channel *I* over alphabet Σ
  - Outgoing channels $O_1 \ldots O_n$ over alphabets Σ_i, respectively
  - a handler function *h*: $S \times \Sigma \rightarrow S \times \Sigma_1 \times \ldots \Sigma_n$
- Channels:
  - Active: clocks, signals
  - Passive: data types

Tower Semantics

Tower Specification

```
simpleExample = do

  (tx, rx) <- channel
  per0 <- period (1`ms`)
  per1 <- period (10`ms`)

    handler per0 “t0” $ do
      … Ivory code …
      emit tx 42

    handler per1 “t1” $ do
     … Ivory code …
     emit tx 99

   monitor "m” $ do
    s <- state ...
    handler rx “foo” $ do
     … Ivory code …
     … update s  …
```



1ms

```
task t0:
{
  lock;
  *chan_t0 = 42;
  Ivory code...
  unlock;
  foo(chan_t0);
}
```

10ms

```
task t1:
{
  lock;
  *chan_t1 = 99;
  Ivory code...
  unlock;
  foo(chan_t1);
}
```

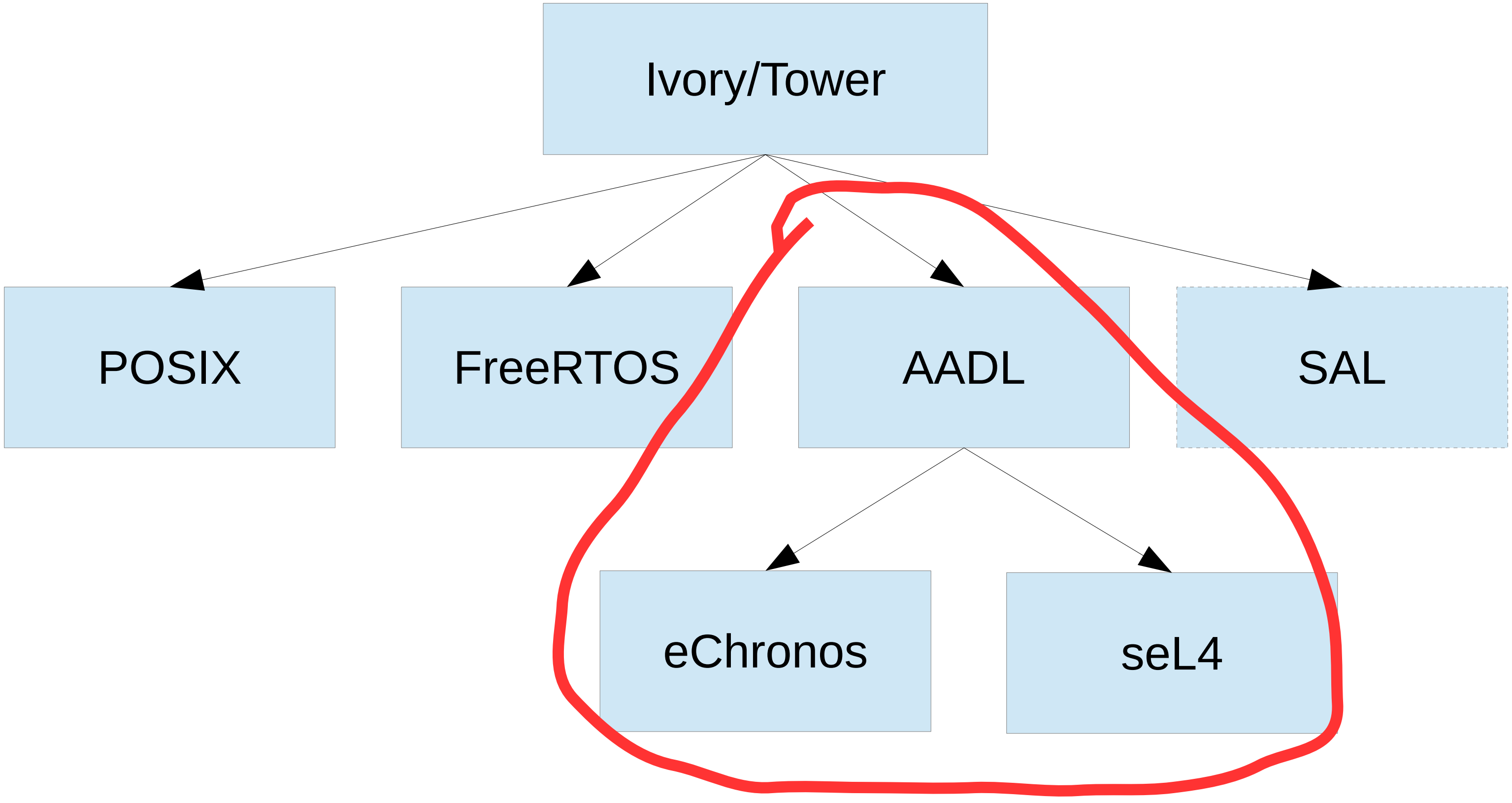
```
foo(chan) {
  lock;
  Ivory code...
  s = *chan;
  unlock;
}
```

Uniprocessor implementation

Implementation Constraints

- No channel cycles
- All monitor computation in a mutex
- Up to the programmer to keep monitors small—critical sections
  - No nested locks—allows simple priority ceiling
  - Task WCET is sum of closure of handler WCET

Backends



“Trusted Build”

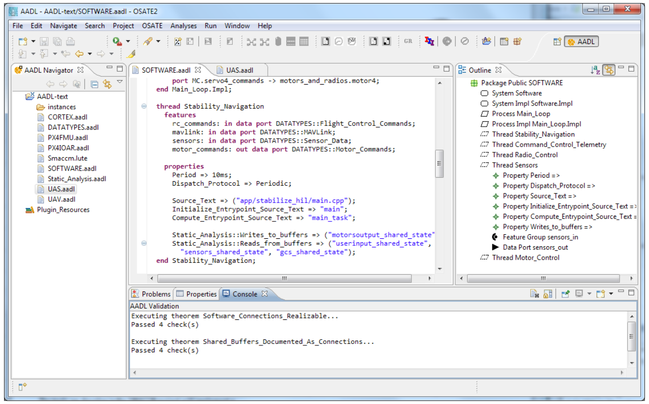
|galois|

Common tools: Formal Methods Workbench

OSATE

Trusted Build

Architecture Models



```
    port MC.servo4_commands -> motors_and_radios.motor4;
  end Main_Loop.Impl;

  thread Stability_Navigation
    features
      rc_commands: in data port DATATYPES::Flight_Control_Commands;
      mavlink: in data port DATATYPES::MAVLink;
      sensors: in data port DATATYPES::Sensor_Data;
      motor_commands: out data port DATATYPES::Motor_Commands;

    properties
      Period => 10ms;
      Dispatch_Protocol => Periodic;

      Source_Text => ("app/stabilize_hil/main.cpp");
      Initialize_Entrypoint_Source_Text => "main";
      Compute_Entrypoint_Source_Text => "main_task";

      Static_Analysis::Writes_to_buffers => ("motorsoutput_shared_state"
      Static_Analysis::Reads_from_buffers => ("userinput_shared_state",
        "sensors_shared_state", "gcs_shared_state");
  end Stability_Navigation;
```

```
Executing theorem Software_Connections_Realizable...
Passed 4 check(s)

Executing theorem Shared_Buffers_Documented_As_Connections...
Passed 4 check(s)
```

Resolute

Assurance Case

AGREE

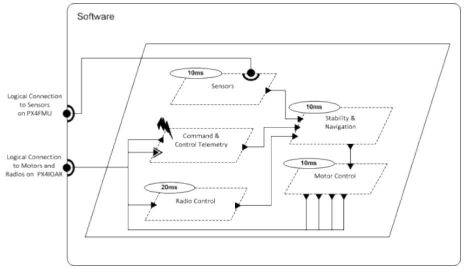
Behavioral Analysis

Lute

Structural Analysis

Architecture Analysis

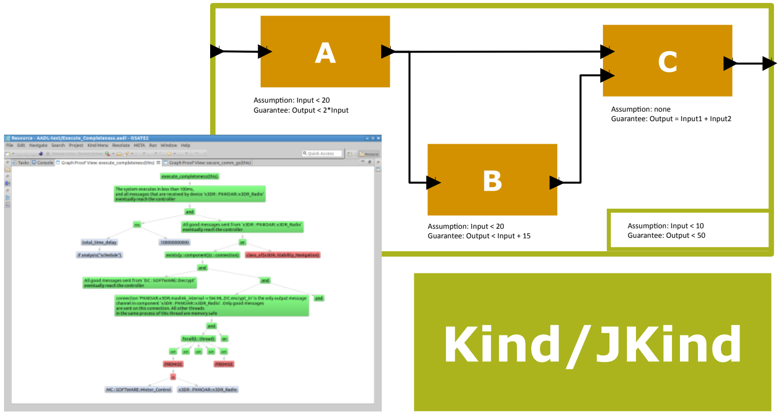
AADL


Software
Logical Connection to Sensors on PX4FMU
Logical Connection to Motors and Radios on PX4IOAR
10ms
Sensors
10ms
Stability & Navigation
Command & Control Telemetry
10ms
Motor Control
20ms
Radio Control

Architecture Translation

seL4

eChronos



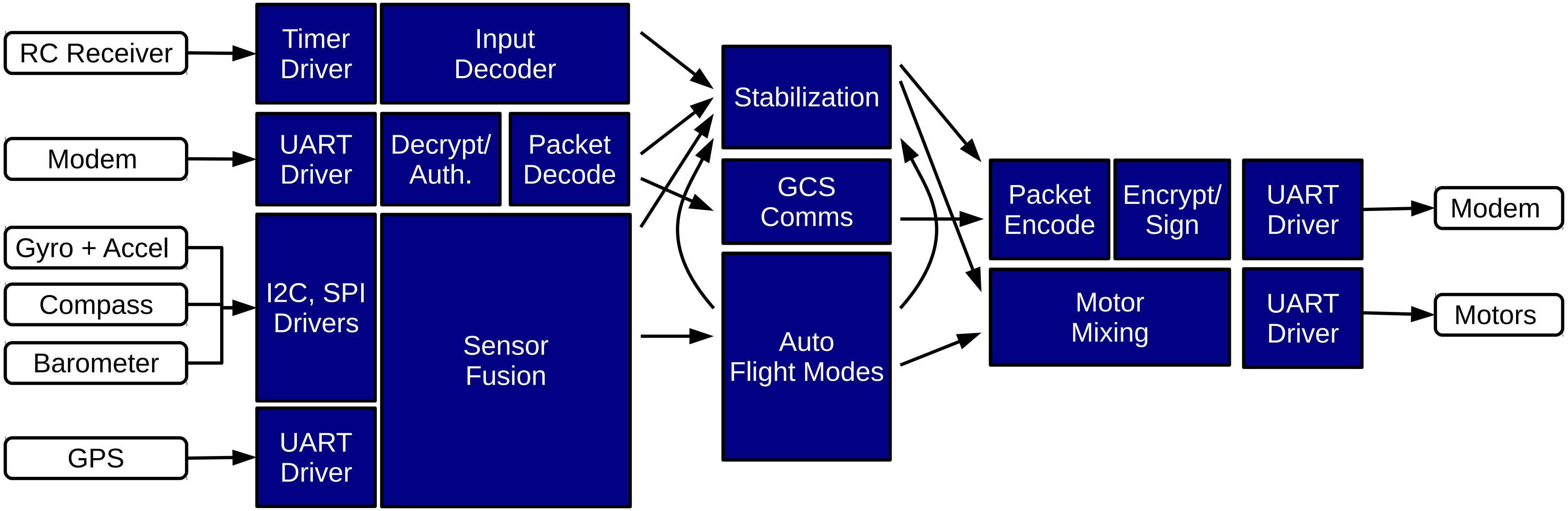
Kind/JKind

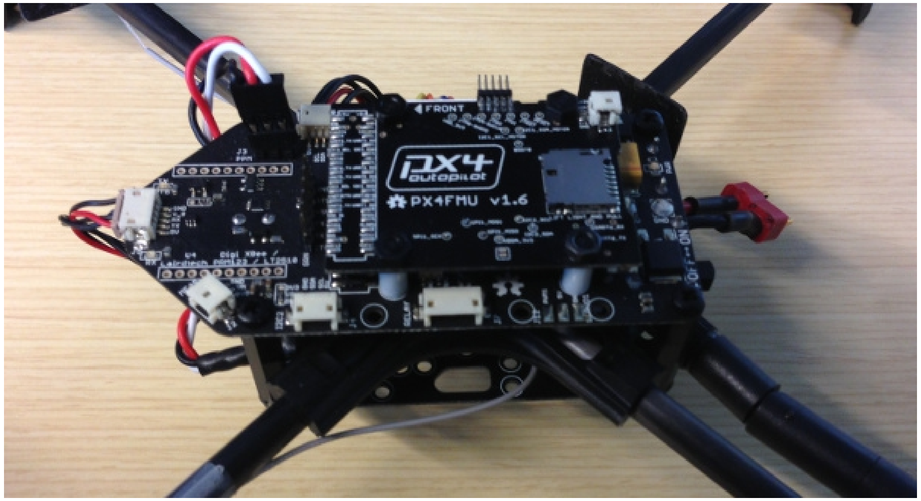
Src: Rockwell Collins

SMACCMPilot

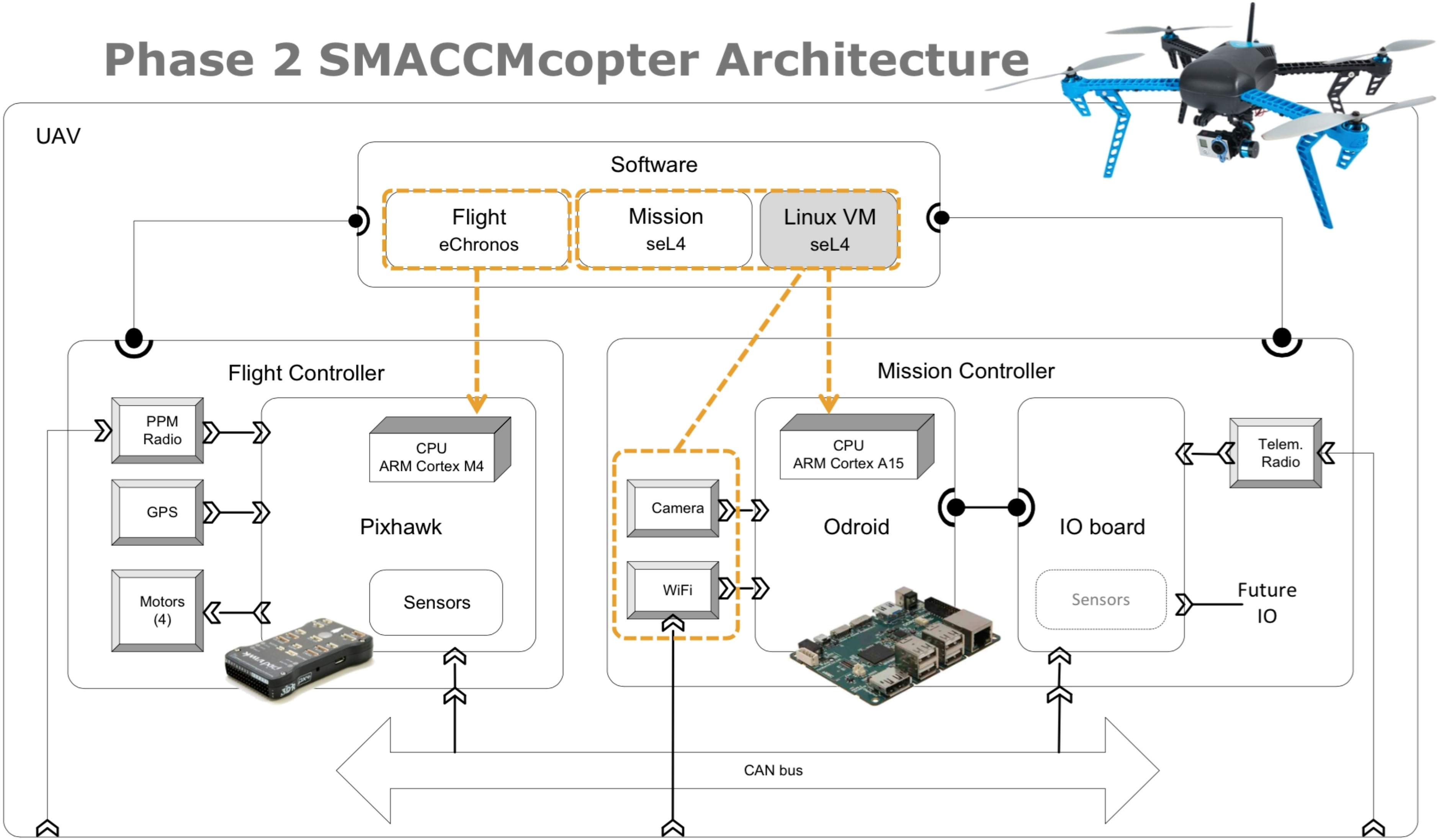
|galois|

SMACCMPilot Architecture





Phase 2 SMACCMcopter Architecture



Red Team Analysis: Baseline System

|galois|

- 3DR Radios have no security; injection and sniffing are trivial
- 3DR radios allow remote reboot into firmware update mode
- MavLink channel operates near saturation, trivial to overload channel causing effects on Mission Planner
- MavLink protocol allows read/write of internal memory
- Mission Planner DoS
- 3DR firmware retrieved from unsecure server by Mission Planner

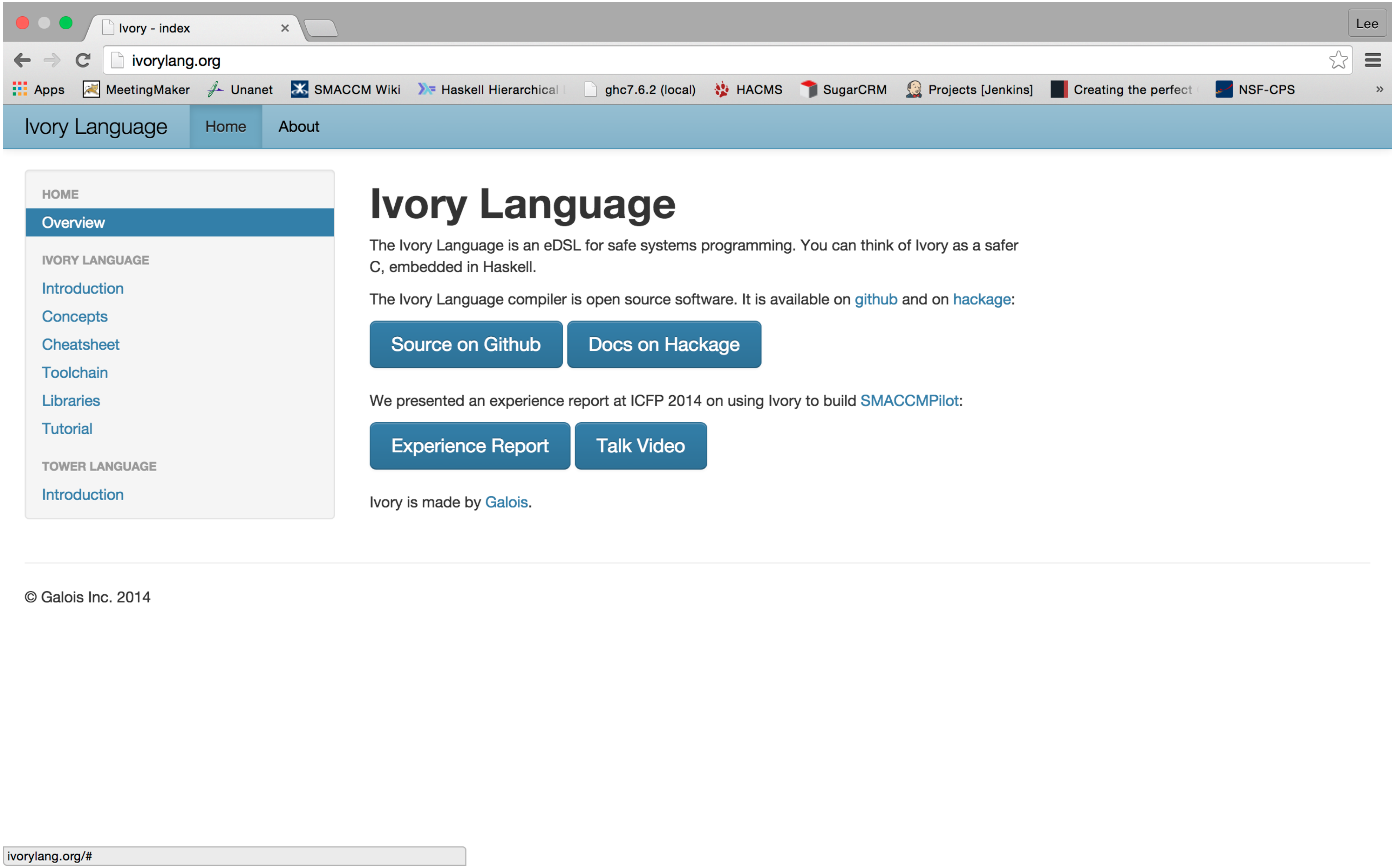
Red Team Analysis: SMACCMPilot

- ~2 months with code and vehicle (whitebox analysis)
- Main tools: code inspection, wisdom, fuzz testing
- Main result: could not penetrate the network/vehicle
- Minor issues found:
  - Replicated debugging channel left in deployed system (physical access)
  - Triggered a code-level assertion

Security for *Systems*

- Do the easy stuff
  - Regression tests, fuzz testing, nightly builds, static analysis
- Do the easy stuff, part II
  - Filter the network inputs
  - Handle all possible errors
- “Hard core” formal verification isn't useful if it's surrounded by a pile of untrusted code (microkernels aren't enough)
- Mitigations are hard
  - What to do with undefined behavior?
  - How to recover?
- Integrate tools/models into the build

ivorylang.org



Ivory Language

The Ivory Language is an eDSL for safe systems programming. You can think of Ivory as a safer C, embedded in Haskell.

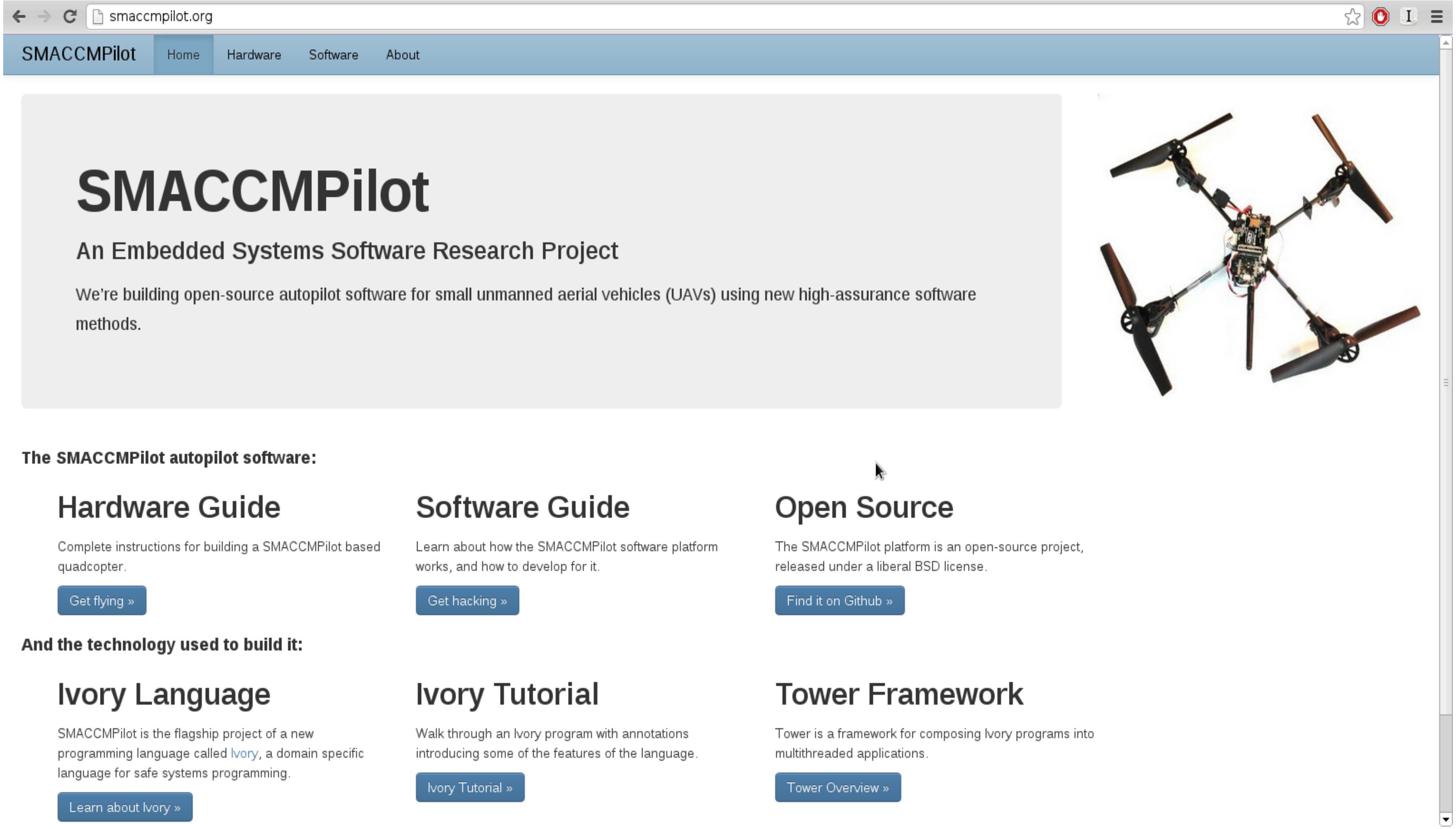
The Ivory Language compiler is open source software. It is available on github and on hackage:

We presented an experience report at ICFP 2014 on using Ivory to build SMACCMPilot:

Ivory is made by Galois.

© Galois Inc. 2014

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.

Get flying »

Software Guide

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

Get hacking »

Open Source

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

Find it on Github »

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.

Learn about Ivory »

Ivory Tutorial

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

Ivory Tutorial »

Tower Framework

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

Tower Overview »

Questions