A Business Case for Functional Programming

John Launchbury

HCSS 2005

galois

Outline

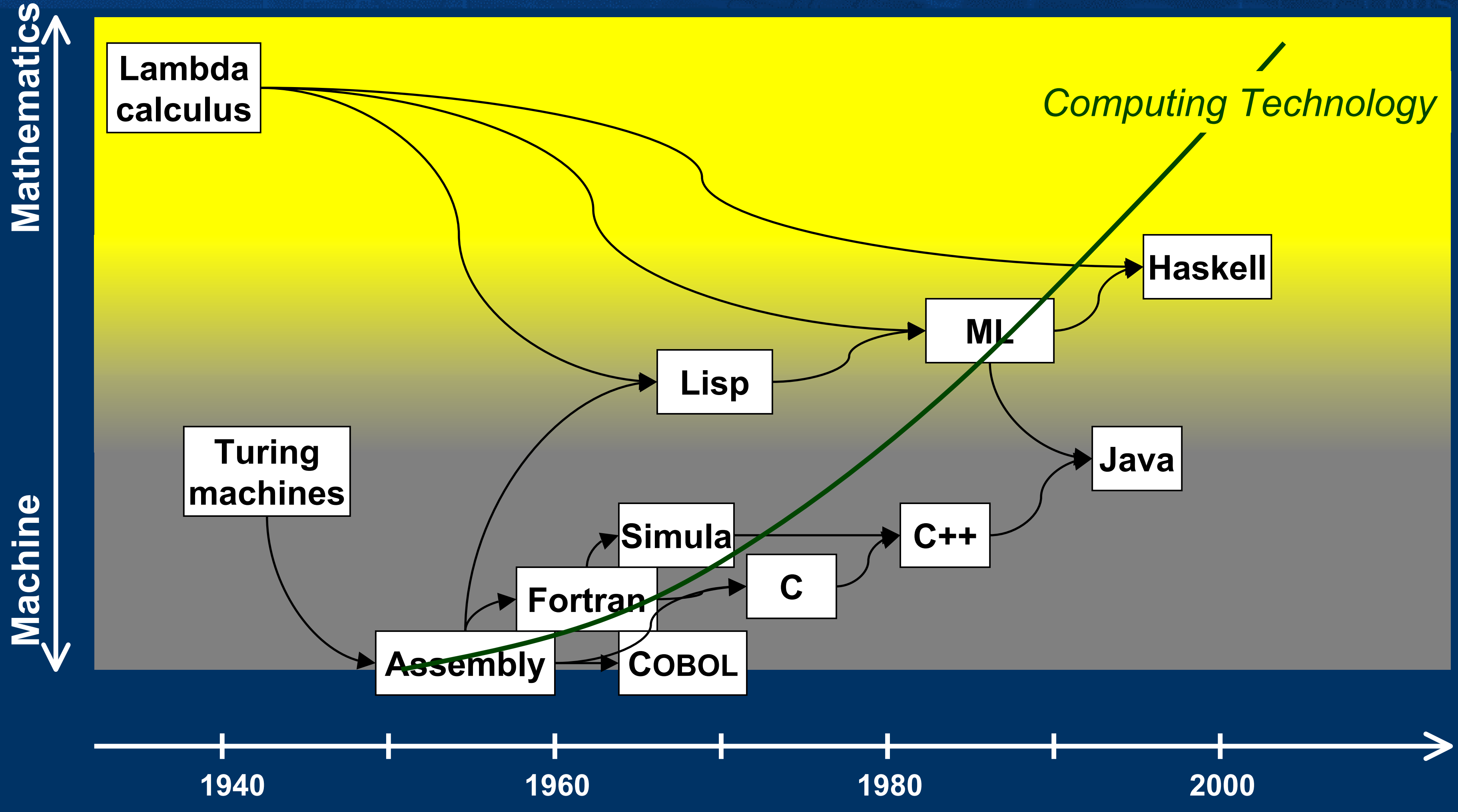
- Functional languages come of age
- Making a business from functional languages
- Focus and brand
- High assurance development
- Looking forward

Language Evolution

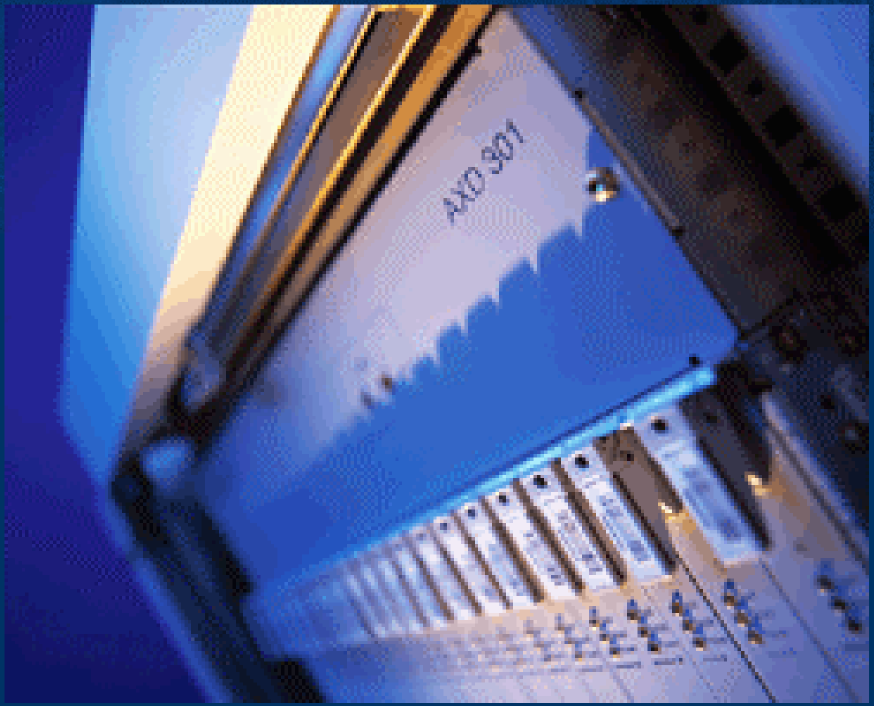
- Growth in
  - Capabilities
  - Expressiveness
  - Manageability

Abstraction
=
Say what needs to be said, nothing more

Computer Language Pedigrees



Ericsson



- **Problem**
  - Build 40Gbit per sec internet/telephone switch
  - C++ project collapsed
- **Solution**
  - Functional language: Erlang Open Telecom Platform

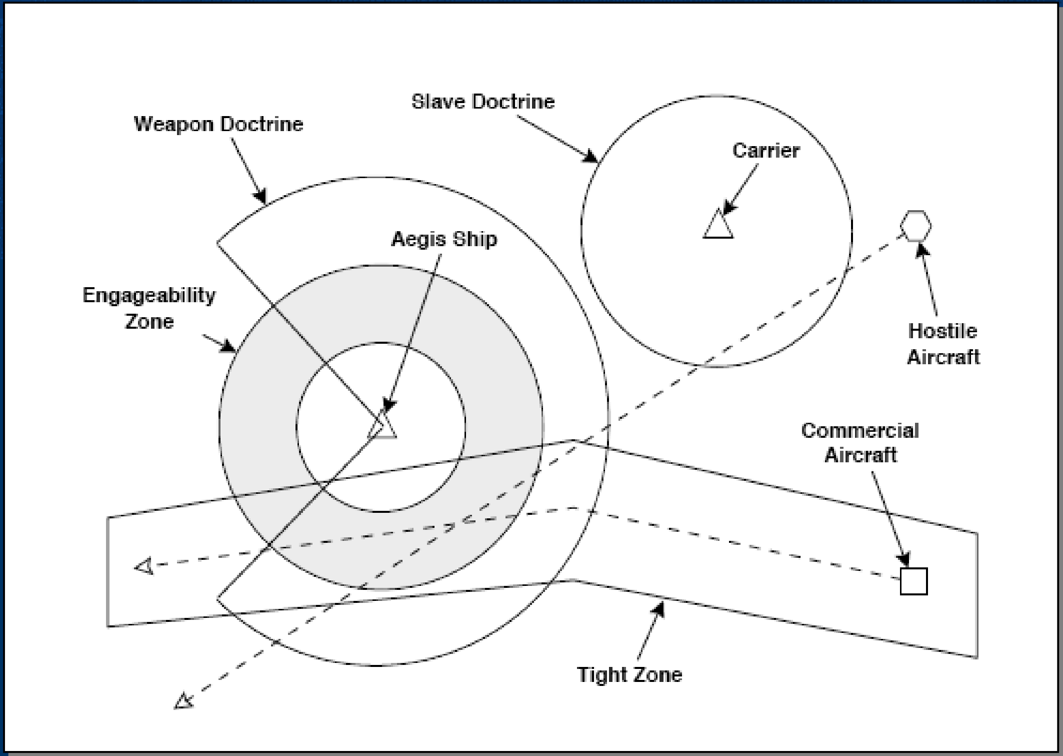
Metrics: 6x increase in productivity
4x increase in product quality

AXD301 now powers Europe's three largest transit switches

galois

Published Experiment

- Haskell vs. Ada vs. C++ vs. Awk vs. ...
- An Experiment in Software Prototyping Productivity
  - Paul Hudak, Mark P. Jones



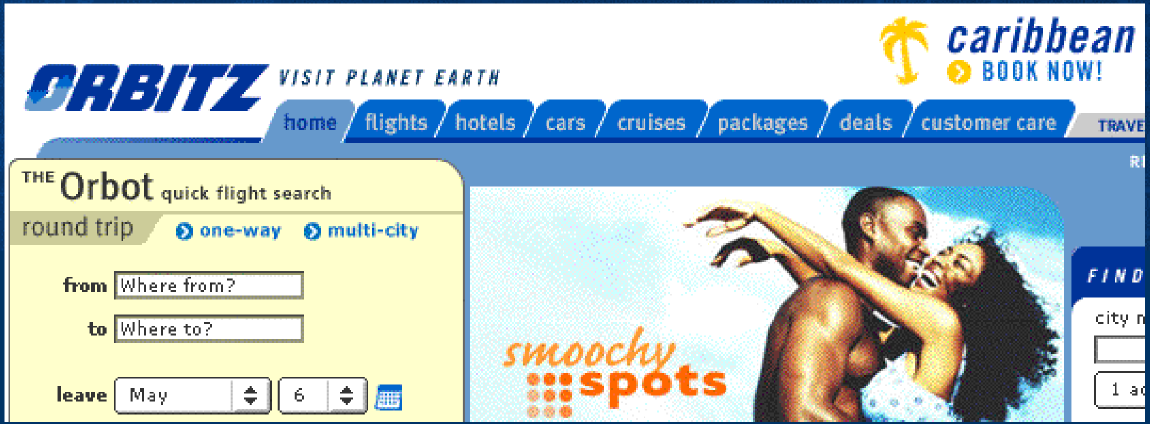
- Experiment designed and conducted by Naval Surface Warfare Center (NSWC)
  - GEO server problem
  - Component of a larger AEGIS system
  - NSWC software development staff has many years experience developing large, complex, software systems
  - Problem tackled by experts in each programming language

| galois |

Summary of Metrics

Language	Lines of code	Lines of documentation	Development time (hours)
Haskell	85	465	10
Ada	767	714	23
Ada9X	800	200	28
C++	1105	130	unreported
Awk/Nawk	250	150	unreported
Griffin	251	0	34+
Proteus	293	79	26
Relational Lisp	274	12	3
[Haskell	156	112	8]

Orbitz.com



- **Problem**
  - Find good travel deals
  - 30 yr old mainframe systems
- **Solution: Orbitz web server search engine**

Metrics: 1,000,000 queries per day
Substantial gain in market share
PCs instead of Mainframe

Functional languages:
smarter search algorithms,
fraction of the development cost

galois

Third Party View of FLs

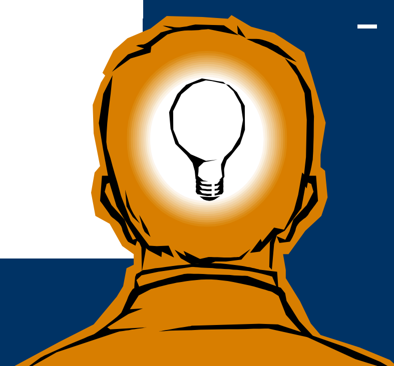
- **Basic data structures are easier to work with**
  - Vectors, many kinds of trees, etc.
- **The compiler doesn't need much hand-holding**
  - In "x + 1.2" the type of x is obvious from context — it must be a floating point value
  - If you use x inconsistently, the compiler will complain
- **Polymorphism**
  - Code that works on the shape of a binary tree can operate on trees containing vectors, or strings, or integers, etc.
    - Like templates in C++, without extra fussing or syntax
  - One routine can be used in a variety of situations.
    - Bottom line: you write less code

Third Party View …

- **More advanced concepts built-in**
  - "Imagine passing a point to a routine and having it return a function that moves a creature one step toward that point"
  - Bizarre? Or just not in the usual C++ toolbox
  - "Programming languages teach you not to want what they cannot provide,"
    [Paul Graham, ANSI Common Lisp]
- **Safety**
  - No wild pointers
  - No overrunning array bounds
- **Interactivity**
  - The seemingly unachievable goal of C debuggers is to code up a function and then immediately test it interactively …

galois

Business beginning…



- 1999/2000
- Service focus
- Customers
  - Government
  - Local industry

Making a Business out of FP

- ~~Build cool things that people should want~~
  - ~~Find sales people to sell it~~

- Sales *is always* the business
  - Technology is a support department for sales

Marketing identifies the right product for *Technology* to build so that *Sales* will be able to sell

Sales 101

1. Earn the right
2. Develop the need
3. Salesman aware
4. Customer aware
5. Offer solutions
6. Close

Rapport
Believability

Open ended questions
Most urgent problems?

Listen.
Wait to see if bigger issues are around the corner

Restate the problem.
Don't assume the customer sees it clearly.

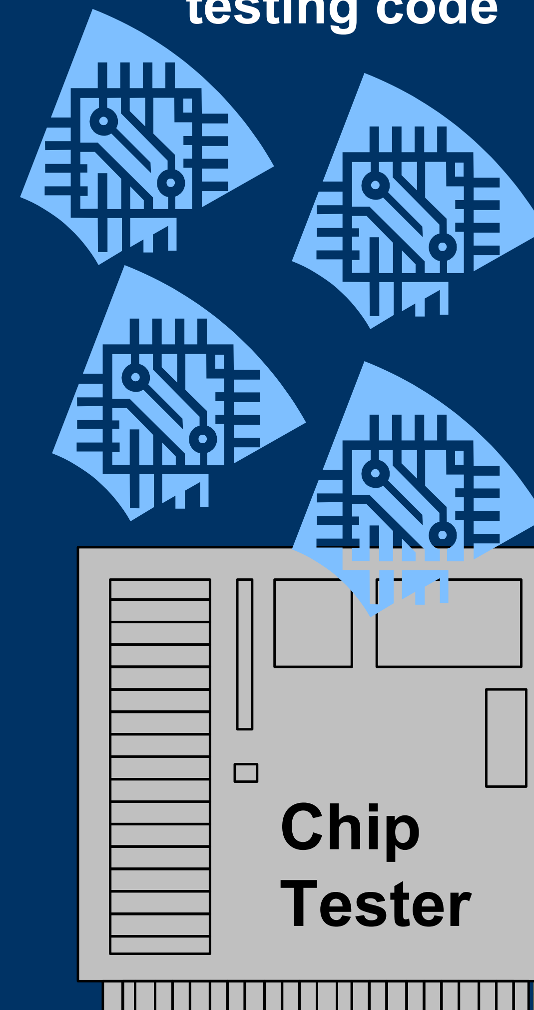
What we can provide.
How valuable?
More analysis?

When should we start?
When would you pay us?
What are the barriers?

Automated Test Equipment

- ATE vendor needs to provide backwards compatibility
- Translation task
  - Code cleaning to upgrade language
  - OS migration
  - API discovery & modification
- Problem: testing code contains IP
- Requirement: the code look-and-feel to remain unchanged

Customer's chip-specific testing code



Partial Change List

- Insert missing headers (`#include`s)
- Change/add prototypes to match definition
- Add prototype declaration instead of implicit forward declaration
- Remove syntactic clutter
- Remove/change ill-behaved declarations (e.g., `static struct`, `static char *`)
- Make type casts explicit (i.e. `double` as case scrutinee; cast to `int`)
- Change now illegal identifier names (forced by ANSI changes)
- Change `return` statements for functions that now return `void`
- Make implicit variable declarations explicit (i.e., to `int`)
- Type changes (explicate `CONN` equivalences)
- Introduction & initialization of global and /or local variables
- Type changes/initialization of struct members
- Aggregate initialization (where array is given all its values at once; need to translate to explicit bit setting)
- Removal of redundant checks (no need to check for end of array; done inside API)
- Flag deprecated API elements
- Replacing malloc/free with API create/destroy
- API function name/type changes

API Discovery

- Old machine
  - Test programs use arrays as connection lists

```
b1 = *c;          /* set b1 to current bit */
b2 = *(c++);      /* set b2 to next bit, move focus */
*(c + 1) = b3;    /* set next bit to b3 */
```

- New machine
  - Requires use of API for building connection lists

```
b1 = conn_getbit(c, c_current);
b2 = conn_getbit(c, c_current++);
conn_setbit(c, c_current + 1, b3);
```

| galois |

```
/* 1. BEFORE */

debug_printf("**** DSP error in test %s,
             occurred on bit # %d -->",
             test_name (NULL),
             (*plist & ~LASTBIT) + 1);

if ((log_ptr->vector >= f_scan_st[u])
 && (log_ptr->vector < f_scan_sp[u]))
 {
  if ((log_ptr->fail_bits[0]
       == *even_ram)
   || (log_ptr->fail_bits[1]
       == *even_ram))
  {
    ficm_write(even_ram, log_ptr->vector,
               log_ptr->vector,
               "H", UNSPECIFIED, UNSPECIFIED);
    rep_str[2*u][log_ptr->vector - f_scan_st[u]] = '1';
  }
```
ois

```
/* 1. AFTER  */

debug_printf("**** DSP error in test %s,
             occurred on bit # %d -->",
             test_name (NULL),
             conn_getbit(plist, plist_local_counter) + 1);

if ((log_ptr->vector >= f_scan_st[u])
 && (log_ptr->vector < f_scan_sp[u]))
 {
   if ((log_ptr->fail_bits[0]
      == conn_getbit(even_ram, even_ram_global_counter))
    || (log_ptr->fail_bits[1]
      == conn_getbit(even_ram, even_ram_global_counter)))
   {
     ficm_write(even_ram, log_ptr->vector,
               log_ptr->vector,
               "H", UNSPECIFIED, UNSPECIFIED);
     rep_str[2*u][log_ptr->vector - f_scan_st[u]] = '1';
   }
```

```
/* 2. BEFORE */

for( pbl = 0; pbl < S_parConnPointer->nrbitl;
     pbl++ )
{
    close_mba_relays
          ( S_parConnPointer->bitl[pbl] );
    open_io_relays
          ( S_parConnPointer->bitl[pbl] );
    prim_wait( 3 MS );
    if ( MbaTest( S_parConnPointer->bitl[pbl],
                  SREXPD, SRESPD, DontDoMbaRly )
         == FAIL )
        goto finish;
    close_io_relays
          ( S_parConnPointer->bitl[pbl] );
    open_mba_relays
          ( S_parConnPointer->bitl[pbl] );

    if ( theSiteCount > 1 && aSiteFailed )
        update_parconn ( &S_tmpParConn, &p_sdbit );
}
```

```
/* 2. AFTER  */

for( pbl = 0; pbl < parconn_getcount(S_parConnPointer);
     pbl++ )
{
    close_mba_relays
          ( parconn_getconn(S_parConnPointer, pbl) );
    open_io_relays
          ( parconn_getconn(S_parConnPointer, pbl) );
    prim_wait( 3 MS );
    if ( MbaTest( parconn_getconn(S_parConnPointer, pbl),
                  SREXPD, SRESPD, DontDoMbaRly )
         == FAIL )
        goto finish;
    close_io_relays
          ( parconn_getconn(S_parConnPointer, pbl) );
    open_mba_relays
          ( parconn_getconn(S_parConnPointer, pbl) );

    if ( theSiteCount > 1 && aSiteFailed )
        parconn_update ( S_tmpParConn, p_sdbit );
}
```

Building the translator

- C-Kit in Standard ML/NJ
- Tight schedule

Lesson 1
FP technology covers over a multitude of sins

Translators!!!

- Build demos
- Visit potential customers
- Align with channel partners

Market issues

Lesson 2
Keep the blue line above the red line

galois

Analysis

- **Didn't read the market properly**
  - References
  - Budgets
- **Lost focus on our core business**

- **Needed to re-invent Galois**
  - Very challenging times

Lesson 3
It's not about technology, it's about relationships

galois

Who are we?

- Examination
  - Look at what we’ve been successful at
  - Look at our skill sets
  - Ask our clients
- Synthesize
- Define the *brand*

Lesson 4
If you don’t know who you are,
then neither does anyone else

galois

Specifications and Formal Tools

Cryptol

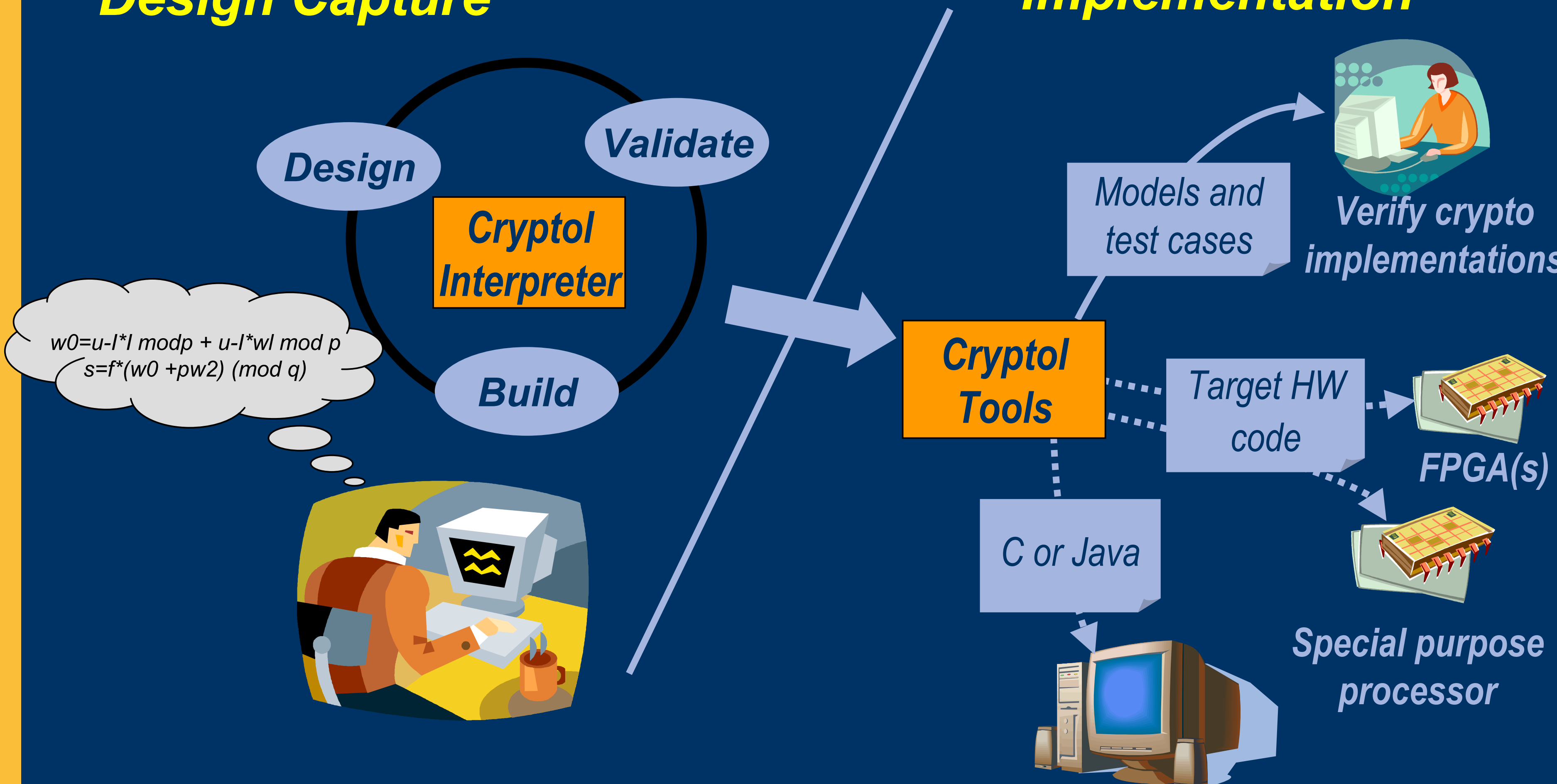
The Language of Cryptography

- Early government contract
- Declarative specification language
  - Language tailored to the crypto domain
  - Designed with feedback from cryptographers
- Execution and Validation Tools
  - Tool suite for different implementation and verification applications
  - In use by crypto-implementers

One Specification — Many Uses

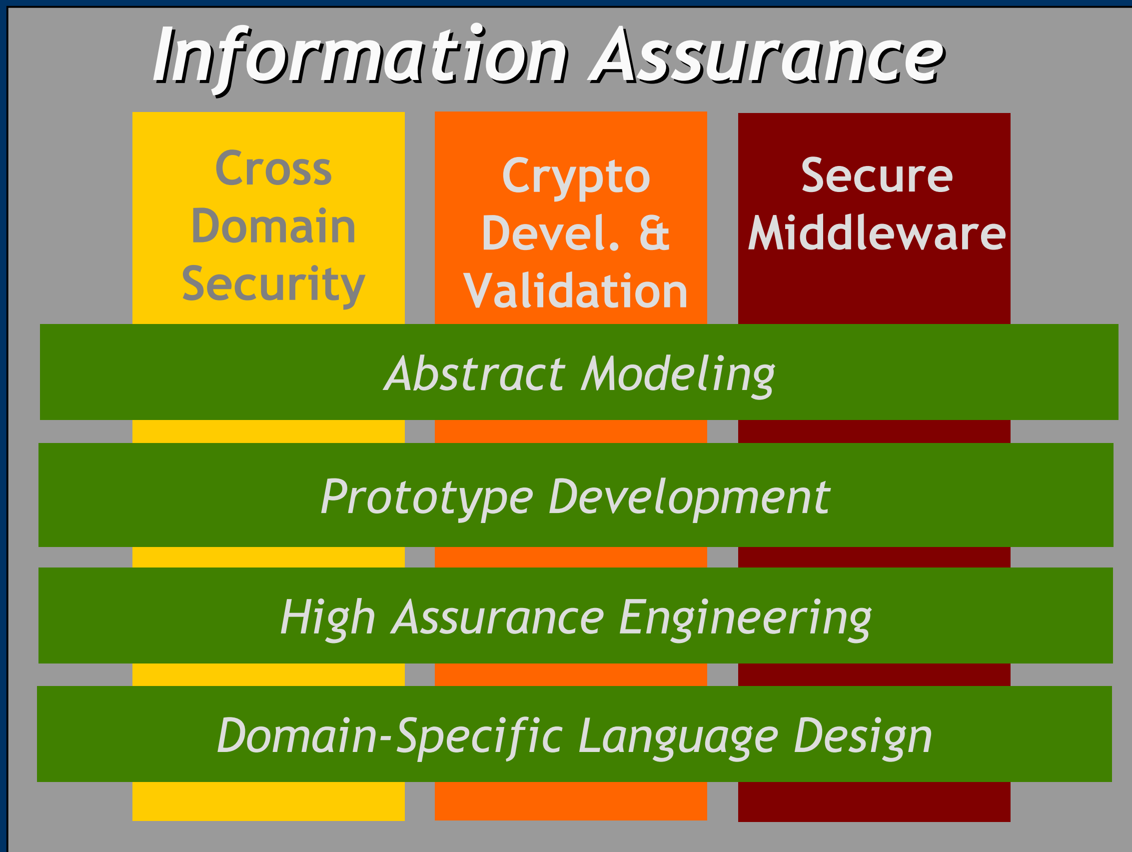
Domain-Specific Design Capture

Assured Implementation



galois

High Assurance Software

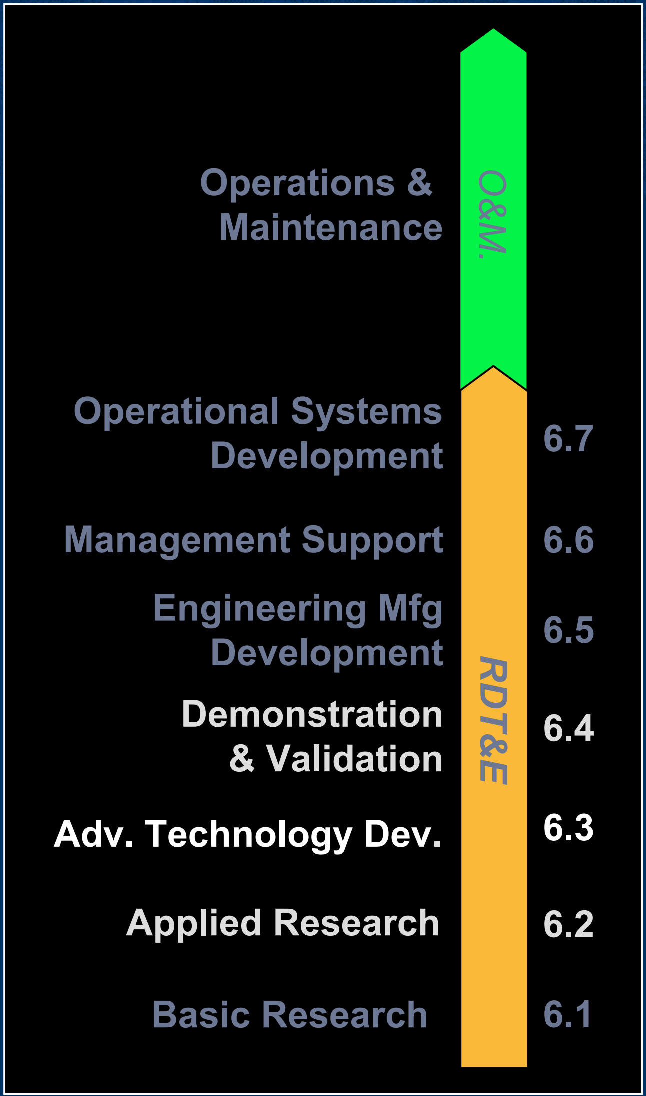


Mission: Advanced technology development for Information Assurance

galois

Technology Services

- **Advanced technology development**
  - Applied research to bring new technologies to bear
  - Demonstration & validation to ensure successful deployment
- **Market**
  - Government
  - Industry selling to government
  - Other industry
- **Business model**
  - Services and product licenses
- **Seek partnerships elsewhere**
  - Critical for client success
  - Outside of Galois' core competency



galois

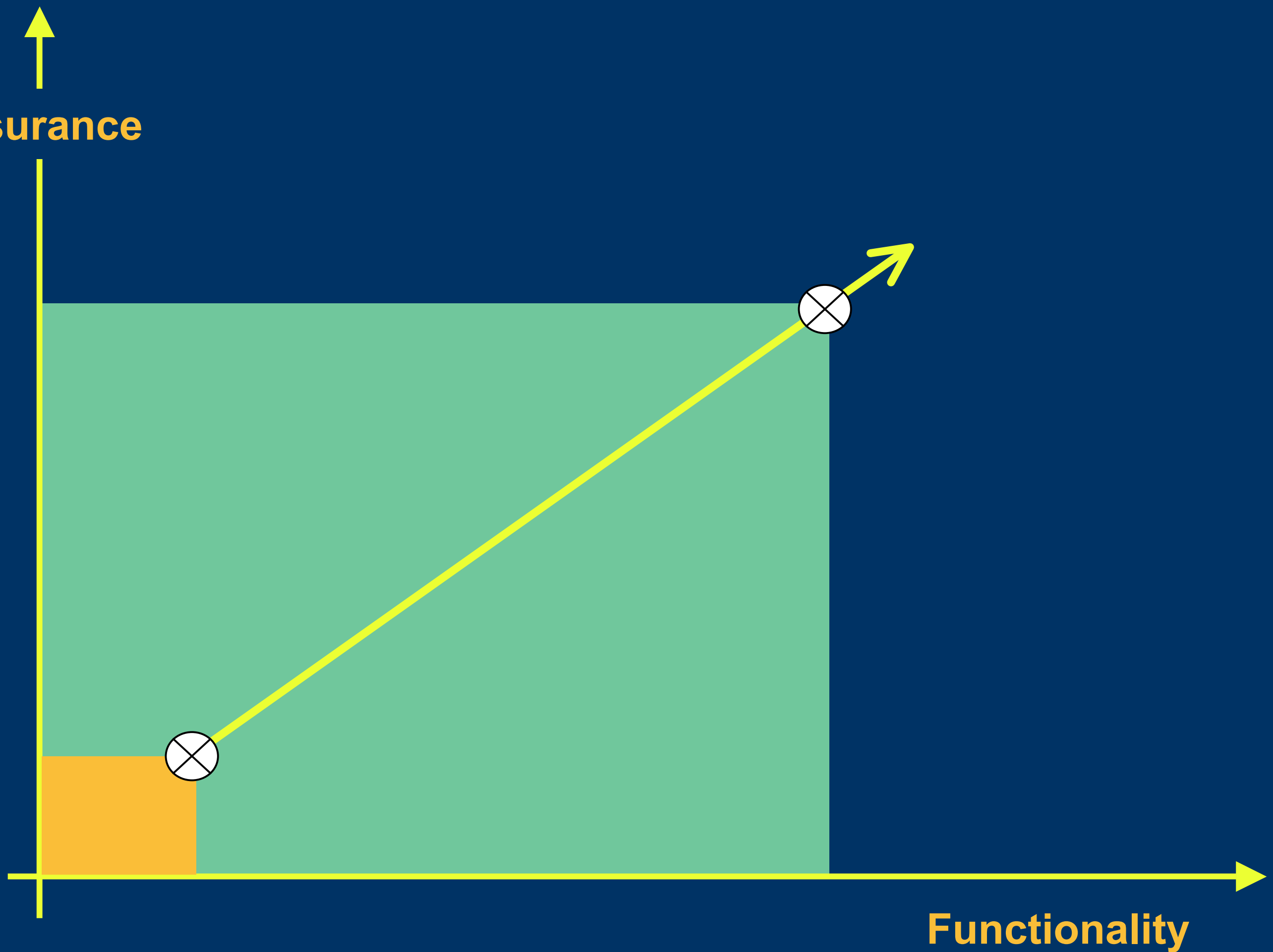
Evaluated Assurance Levels *(EAL)*

EAL1	Functionally tested	COTS
EAL2	Structurally tested	
EAL3	Methodically tested and checked	
EAL4	Methodically designed, tested & reviewed	
EAL5	Semi-formally designed and tested	Galois Focus
EAL6	Semi-formally verified design and tested	
EAL7	Formally verified design and tested	

- NSTISSP 11: Effective 1 Jul 2002
  - Acquisition of COTS IA products *limited* to NIAP validated Products, or NIST validated Crypto Modules
  - Acquisition of GOTS IA products *limited* to NSA approved

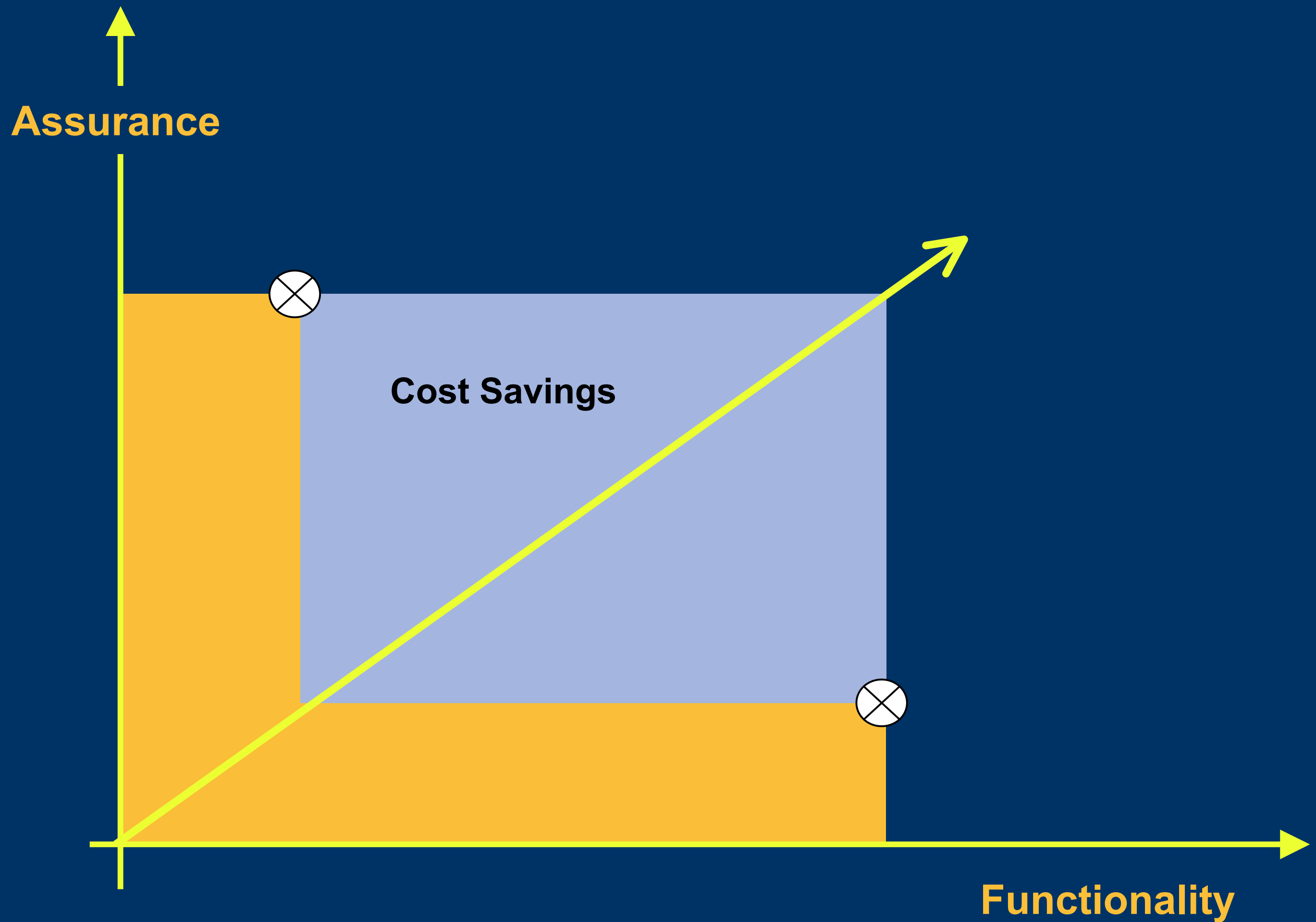
galois

Cost of Assurance/Functionality



galois

Cost of Assurance/Functionality

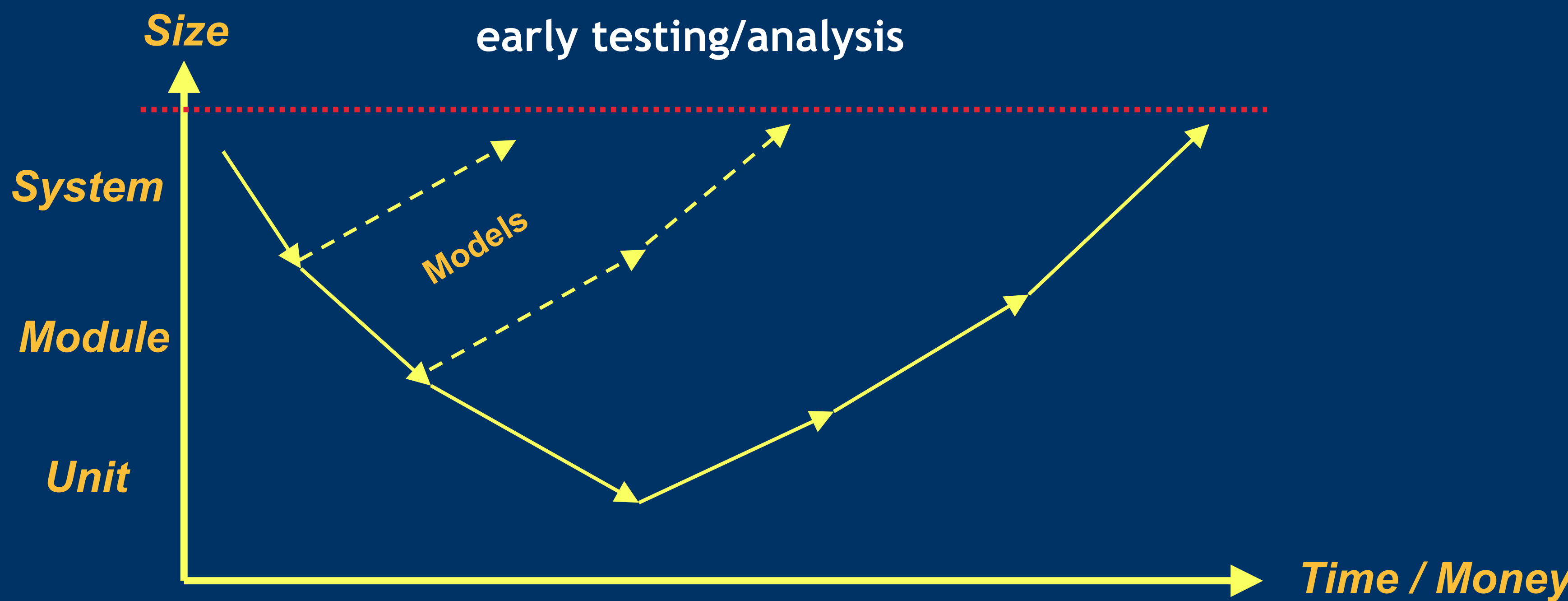


High Assurance Development

- **Non-functional, non-technical requirements**
  - Documented software process
  - Physical security of code
- **Security requirements**
  - Protection Profile (PP)
    - User statement of security requirements
  - Security Target (ST)
    - Developer statement of the security functionality of a product
- **Verification and validation**
  - Traditional testing
  - Formal and semi-formal designs and models

Early Use of Models

Profound cost-benefit of early testing/analysis

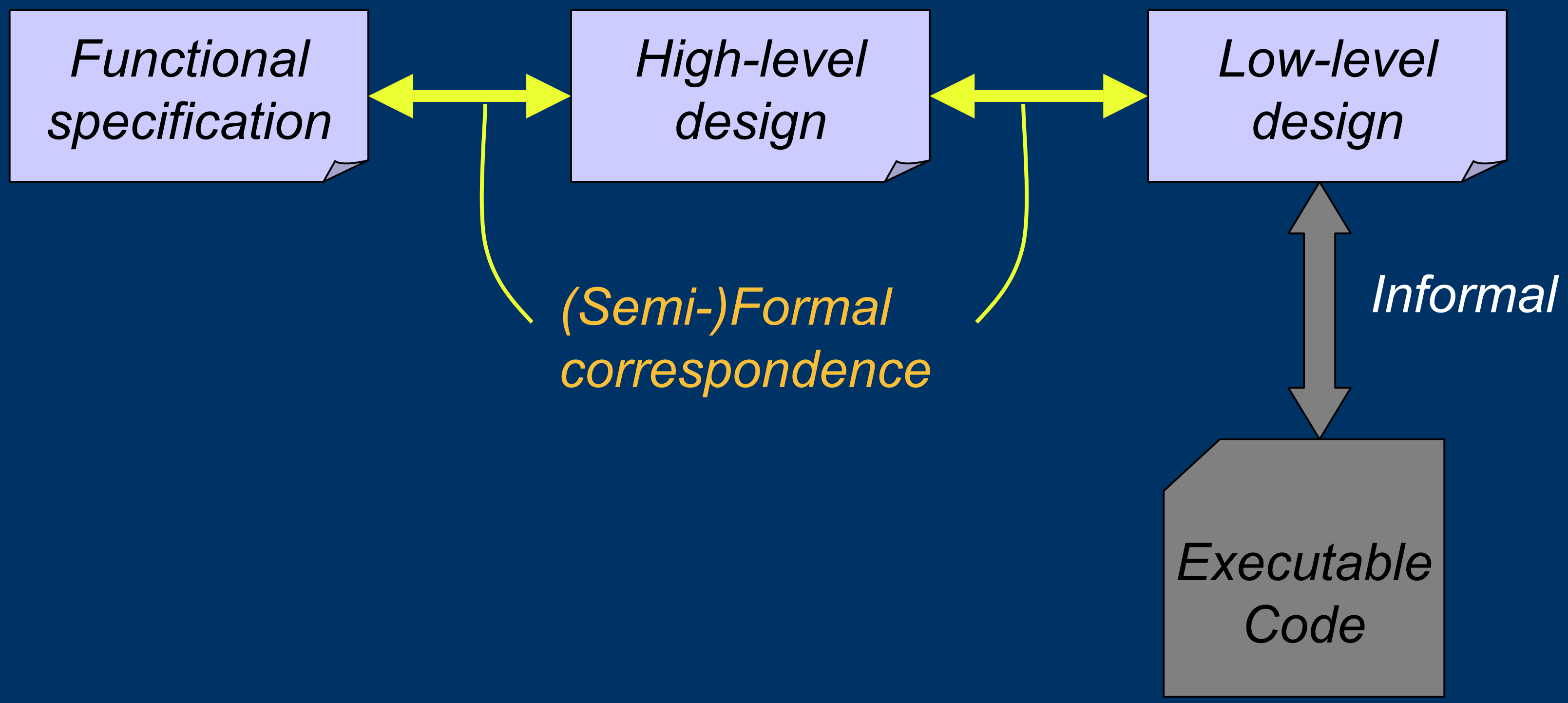


- All other branches of engineering make heavy use of mathematical models
- Mathematically meaningful models should be more prevalent in software

galois

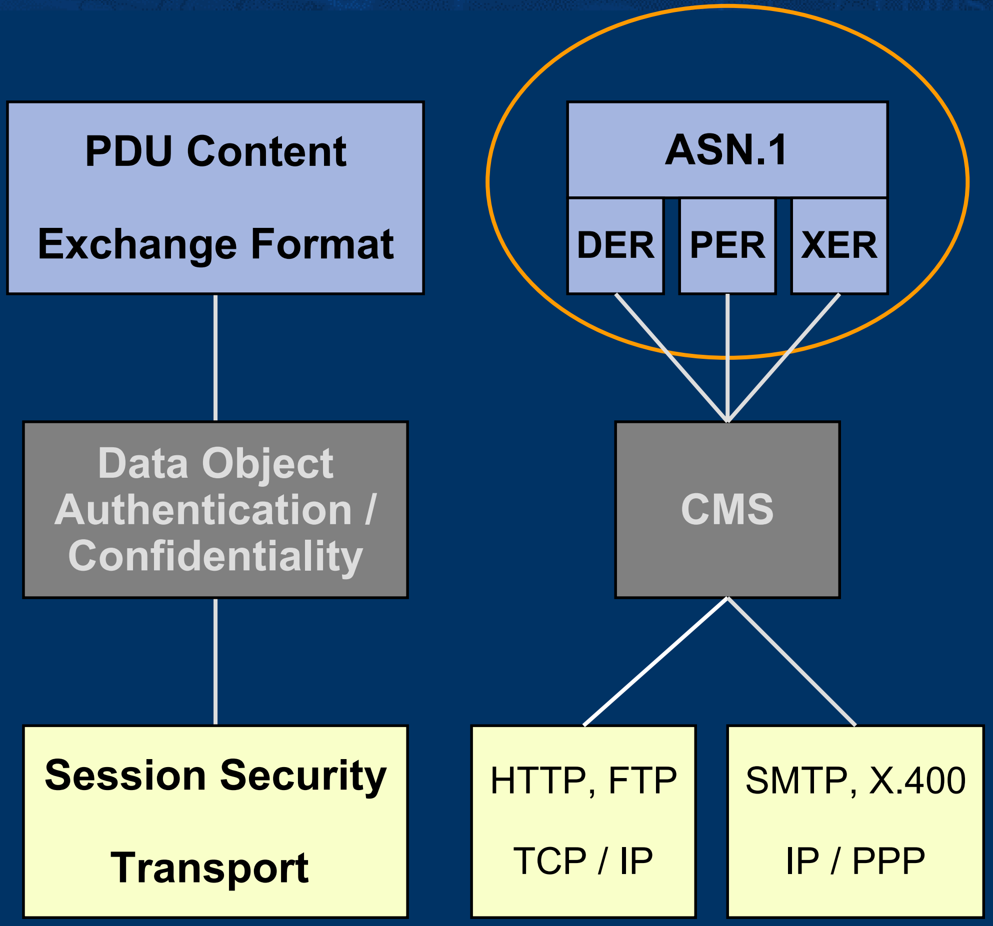
(Semi-) Formally Verified Designs

- Even EAL7 does not require formal analysis of the executable code



Role of ASN.1

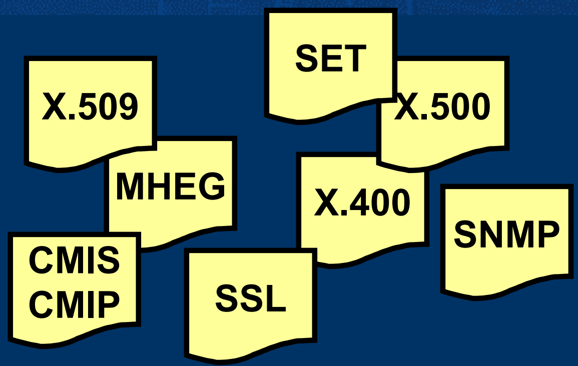
- ASN.1 is a data-description language
  - E.g. for communication protocols
- Goal
  - Platform-independent data descriptions
  - Given the same semantics by all parties
  - Mutually intelligible and interoperable
- ASN.1 used to specify
  - Structure of packet content
  - Format for data exchange



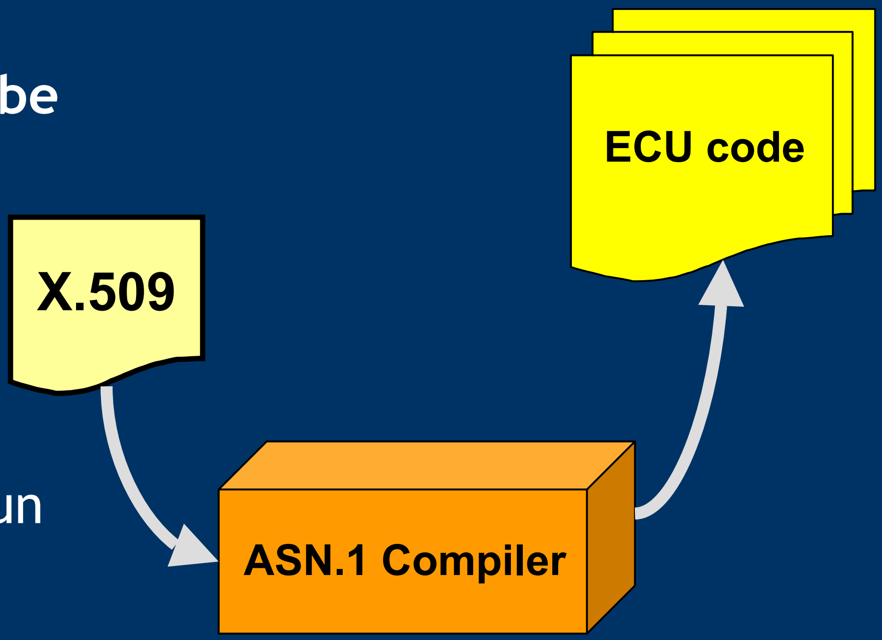
Benefits of using ASN.1

- **Interoperability**
  - Platform independence
  - Vendor independence
- **Abstract**
  - Expresses design-level concepts
  - Facilitates discussion of protocol requirements
- **Reuse**
  - Definitions from one application can be reused effectively in other contexts
- **Software flexibility**
  - Protocol layers can handle data without having to understand the content

From ASN.1 to Executable Code



- ASN.1 specifications need to be turned into executable code
  - By hand, or
  - By a compiler
- Compiler
  - Input: ASN.1 description
  - Output: Program code to run application, e.g. on ECU



The Challenge of ASN.1

- **ASN.1 is a LARGE language**
  - Many (~26) primitive types
  - Many ways to combine components (e.g., CHOICE, SET, SEQUENCE, SEQUENCE OF, SET OF, user-defined)
  - Constraints (X.680, X.682), information objects (X.681), parameterization (X.683)

- **The ASN.1 definition is very dense**
  - Precise semantics of ASN.1 is very difficult to extract
    - E.g. constraints and type equality given in terms of concrete syntax
  - Features of the language interfere with each other

- **ASN.1 executable code faces implementation challenges**
  - Numerous opportunities for overflowing machine representation
    - E.g. arbitrarily long octet streams led to recent bug Microsoft ASN.1 library
  - Common concepts get treated very differently
    - E.g., long tags vs. long lengths vs. long values

Failure of ASN.1 code

- **High impact**
  - Leads to attacker ingress, vulnerability to DoS
  - ASN.1 code often run in "privileged" mode
- **Costs of fixing ASN.1 problems estimated to be greater than Y2K[1]**
  - More equipment affected
  - Attacks lead to outages
  - Repairs must be done more quickly, more often
  - More regression testing required
    - Configuration complexity

[1] "Critical Infrastructure Protection Issues", Bill Hancock, V.P. Security and Chief Security Officer, Exodus, *ITU Workshop on Creating Trust in Critical Network Infrastructures*, May 2002

galois

Prototype High Assurance ASN.1

- **Parser grammar**
  - Almost identical to published ASN.1 definition (X.680 grammar)
  - Direct comparison feasible
- **Code generation**
  - Multiple intermediate forms (V1, V2, EnDe C)
  - Mathematically motivated transformations from one intermediate form to the next

V1: type-based specification of encode/decode

Derivation system gives a formal semantics to ASN.1

V2: lambda calculus implementation of encode/decode

Inlined, specialized version of V1

Gives a formal semantics to individual ASN.1 specifications

EnDe C: mini domain-specific language for encode/decode

Translated from V2

Gives an operational semantics to individual ASN.1 specifications

C: Final code target

Translated from EnDe C

The models form the compiler

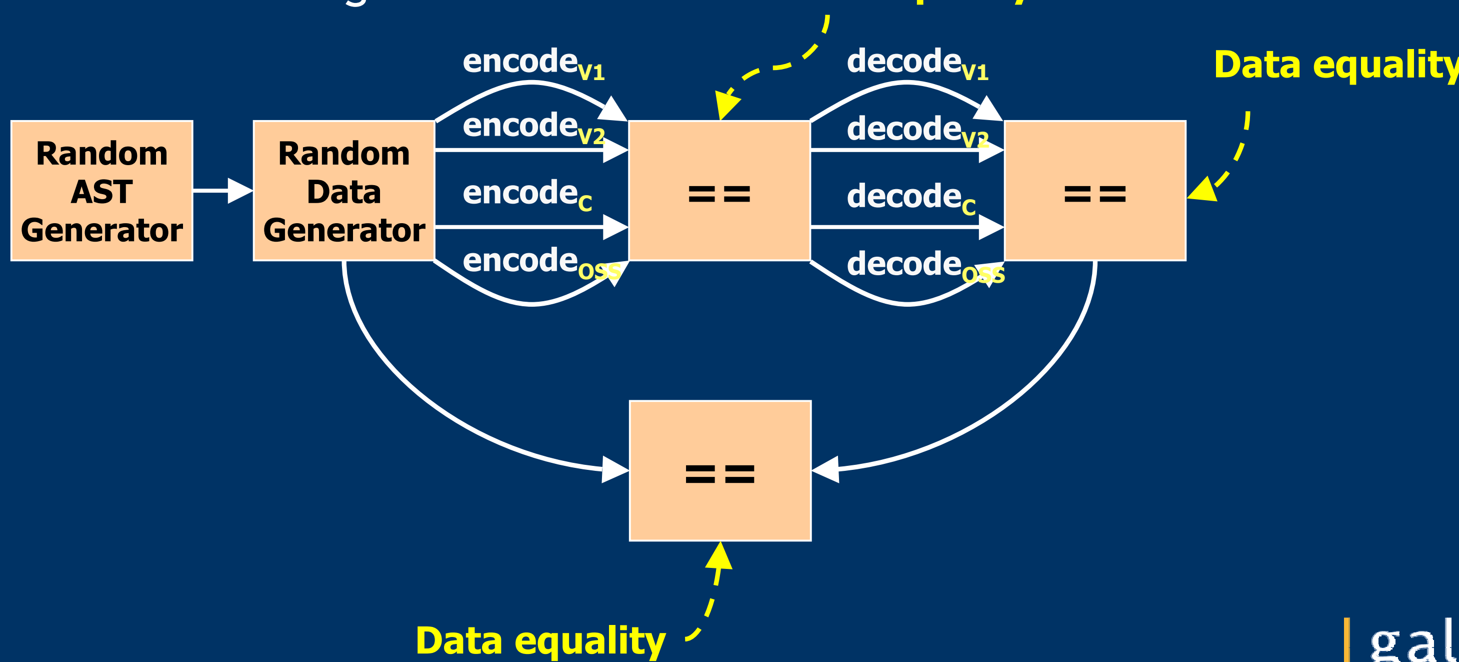
Designed For Robustness

- Mapping to C for each EnDe C construct considered in isolation
- Each mapping designed with robustness properties in mind:
  - Use ADT-style API for all types
    - The generated code handles all allocation
    - The user handles freeing
  - Encode calculates the buffer size required before encoding; allocates accordingly
  - All buffers have associated lengths
  - All `malloc`s are guarded
  - All pointer dereferences are guarded
- Run-time library designed from same principles

galois

Random Coverage Testing

- **Automatically-generated tests**
  - Developed coverage metrics for the input space
  - Tests expected behavior on valid inputs
  - Rejection behavior testing framework designed
- **Also handwritten tests**
  - Unit tests
  - Regression testing



Engineering vs Verifying

Engineering drivers

- Powerful abstraction mechanisms
  - Data, functional, behavioral, name space
- Potential non-termination
- I/O
- State
- External system interaction
- Concurrency
- Exceptions
- Execution debugging, profiling

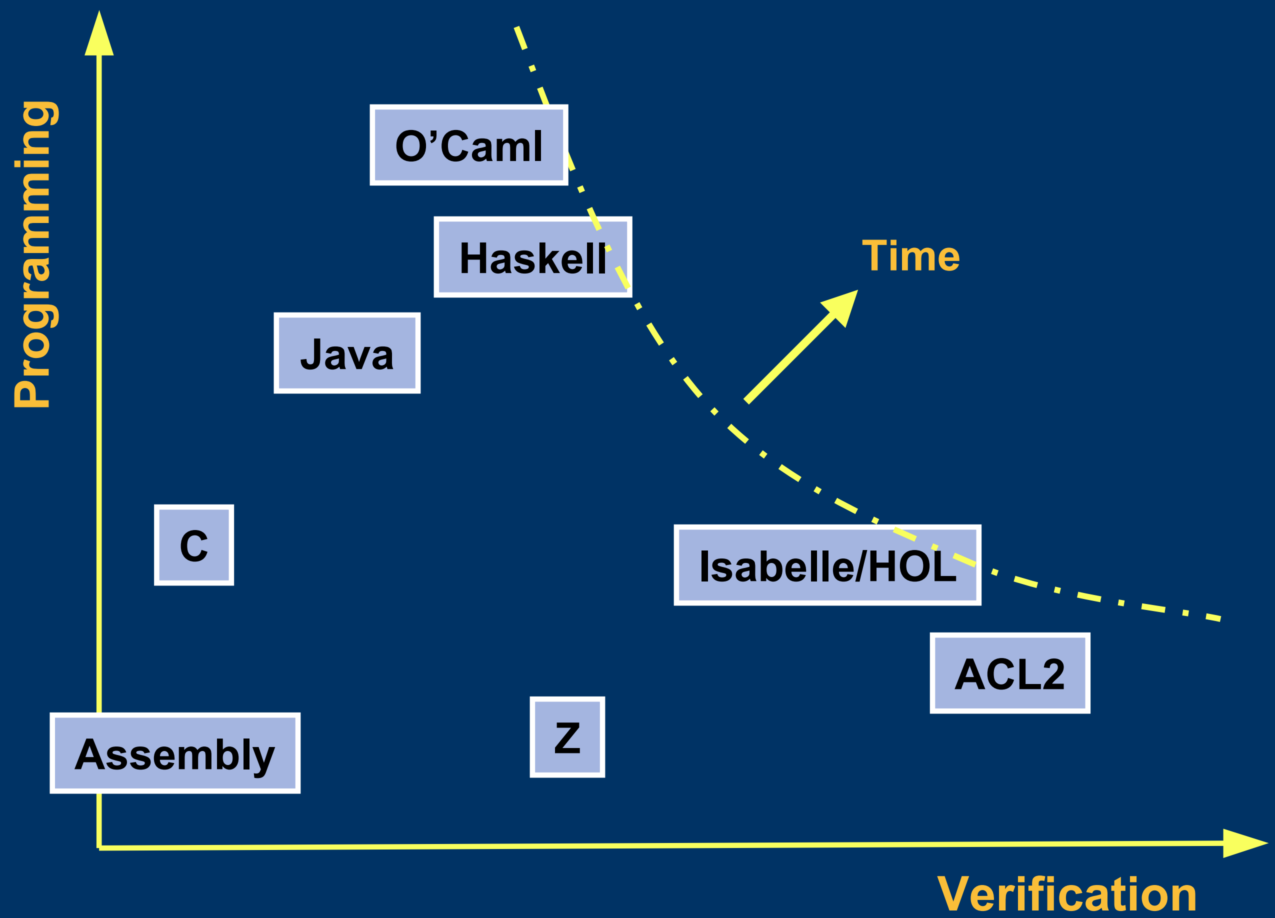
Verification drivers

- Small and simple language
- Declarative semantics
  - E.g., Set-theoretic
- Abstraction
  - Including infinitary objects
- Proof automation and debugging
- Executable
- External system/tool interaction

System and Programming Languages → ← *Verification Languages and Tools*

galois

Tradeoff and compromise



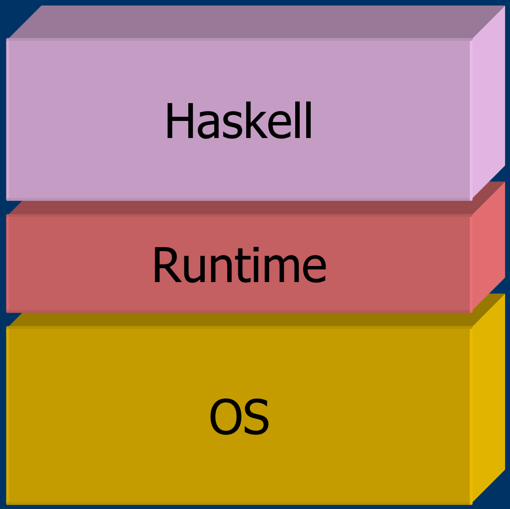
| galois |

Haskell: An Applied Formal Method

- **Specification language AND Implementation language**
  - Semantics is sets+recursion
  - Industrial-grade compiler
  - Powerful abstraction mechanisms
  - Automatic memory management
  - Effects handled explicitly
    - State, Concurrency, Exceptions
  - Flexible and safe mechanism for external interaction
- **Verification**
  - Equational reasoning
  - Type-based theorems
  - QuickCheck properties
- **In development**
  - Programatica (OGI, PSU, Oregon)
    - Haskell + properties
  - CoVer (Chalmers, Sweden)
    - Translate Haskell programs into input for theorem provers
    - Automatic QuickCheck generation
  - Other projects for linking Isabelle/HOL with Haskell

Haskell implementations

- **Any compiled program**
  - Runs in the context of a run-time system
  - Which is hosted by an operating system
  - Which runs on a hardware platform
- **Security/correctness evaluation is not just a matter of looking at the application's source code**



- ***Haskell on Bare Metal* project**
  - Eliminate operating system component
  - Demonstrate that run-time system supports and enforces Haskell semantics
  - Develop certification evidence that applies to any application

Run-Time System Semantics and Low -level Model

- **Haskell's RTS Semantics**
  - Abstract machine transition rules
  - Expressed as a set of Structural Operational Semantics (SOS)
- **Implementation model**
  - Based on the C code implementation of the RTS
  - Written in Haskell
  - Low-level enough that we can relate it to the implementation source code

Operational Semantics

ℰ ⟦M⟧ is the denotational semantics for M

V is the value of M

M wasn't already evaluated

$$\frac{\mathcal{E}[\![M]\!] = V \qquad M \not\equiv V}{\{\mathrm{E}[M]\} \;\rightarrow\; \{\mathrm{E}[V]\}}$$

If the things above the line are true, then we can deduce the thing below the line

Semantics for imperative variables

Model of the program state

$$
\begin{array}{lcll}
P, Q, R & ::= & \{M\} & \text{The main program} \\
 & | & \langle M \rangle_r & \text{An } \texttt{IORef} \text{ named } r\text{, holding } M \\
 & | & P \mid Q & \text{Parallel composition} \\
 & | & \nu x.P & \text{Restriction}
\end{array}
$$

e.g. $\nu r,s.\ (\{M\} \mid \langle 3 \rangle_r \mid \langle 89 \rangle_s)$

The main program

An IORef called r, holding 3

Semantics for imperative variables

Rules for read, write IORefs

$$\{E[\texttt{readIORef}\ r]\}\ |\ \langle M\rangle_r \quad\rightarrow\quad \{E[\texttt{return}\ M]\}\ |\ \langle M\rangle_r$$
$$\{E[\texttt{writeIORef}\ r\ N]\}\ |\ \langle M\rangle_r \quad\rightarrow\quad \{E[\texttt{return}\ ()]\}\ |\ \langle N\rangle_r$$

Uses the framework of **monads**

Low-level RTS Semantics

- The RTS executes STG-Machine code
  - Each concurrent thread has a state:

$$\langle \text{Heap}, \text{Registers}, \text{Code}, \text{Stack} \rangle_t$$

  - The Heap is shared between threads
- The SOS rules specify transitions:

$$\langle \text{Heap}, \text{Registers}, \text{Code}, \text{Stack} \rangle_t \rightarrow \langle \text{Heap}', \text{Registers}', \text{Code}', \text{Stack}' \rangle_t$$

A Low-level Example

- **MVars are Haskell's primitive form of concurrent synchronization**
  - Like semaphores-with-data
  - Three operations:
    - `newEmptyMVar`: creates an empty MVar
    - `putMVar`: stores a value in an empty MVar (blocks if the MVar is full)
    - `takeMVar`: extracts a value from a full MVar, leaving it empty (blocks if the MVar is empty)

- **Each of these is a *primitive* in the RTS**
  - They have to be, because blocking and unblocking of threads requires changing the RTS state
  - **But** they still have a clear and formal semantics…

Model for `takeMVar`

- There are three cases:
  - The MVar in question is empty, so we have to block
  - The MVar is full, and no threads are waiting to put
  - The MVar is full, and there are threads waiting to put

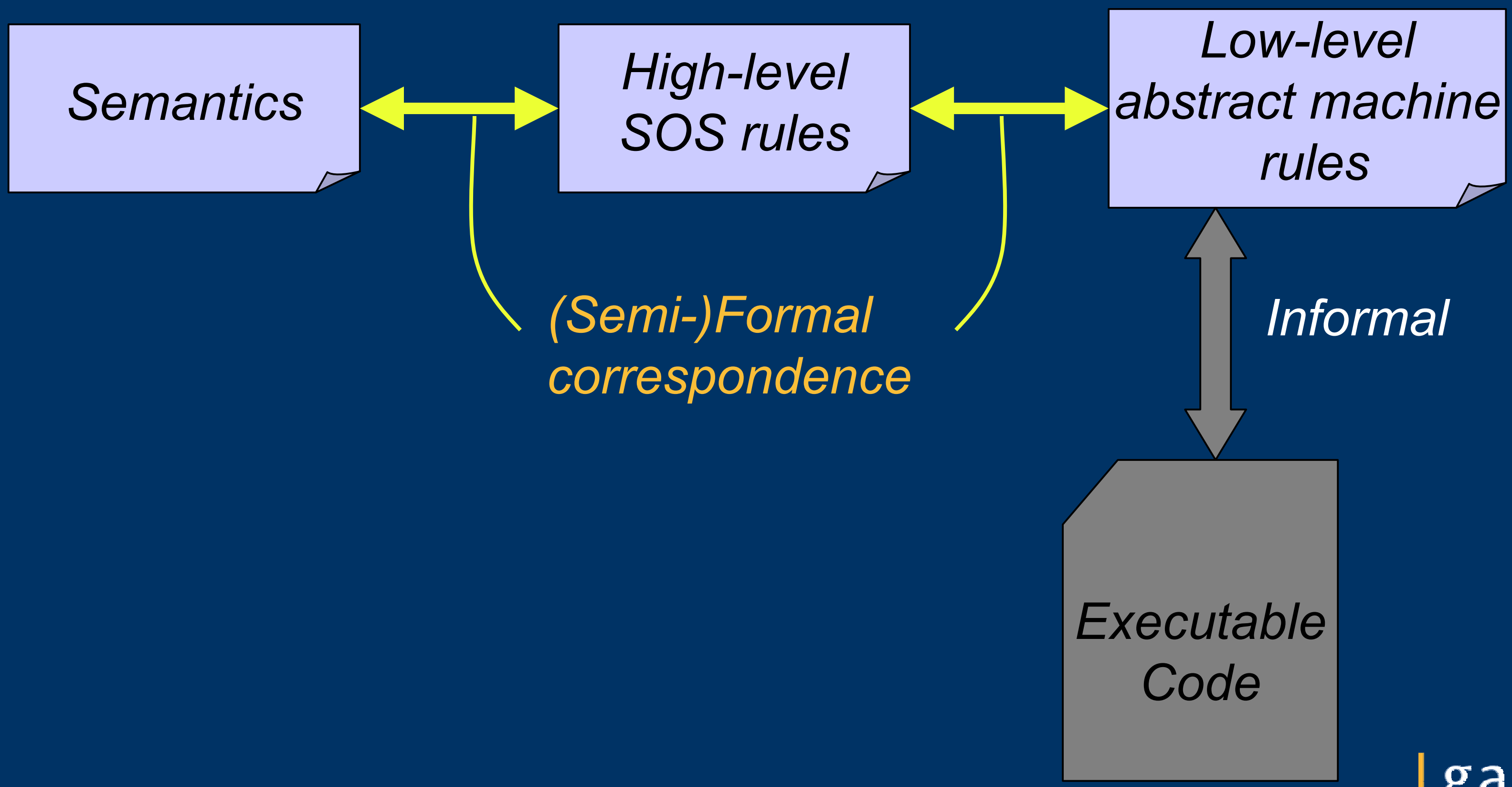
<H, Rs { R1 = MVar Nothing ts, … }, takeMVar, σ>$_t$ →
<H', Rs { R1 = MVar Nothing (t:ts) }, BLOCK, σ>$_t$

<H, Rs { R1 = MVar (Just v) [], … }, takeMVar, σ>$_t$ →
<H', Rs { R1 = MVar Nothing [] }, v, σ>$_t$

<H, Rs { R1 = MVar (Just v) (u:ts), … }, takeMVar, σ>$_t$ →
<H', Rs { R1 = MVar (Just w) ts }, v, σ>$_t$,
<_, Rs, putMVar w, σ'>$_u$

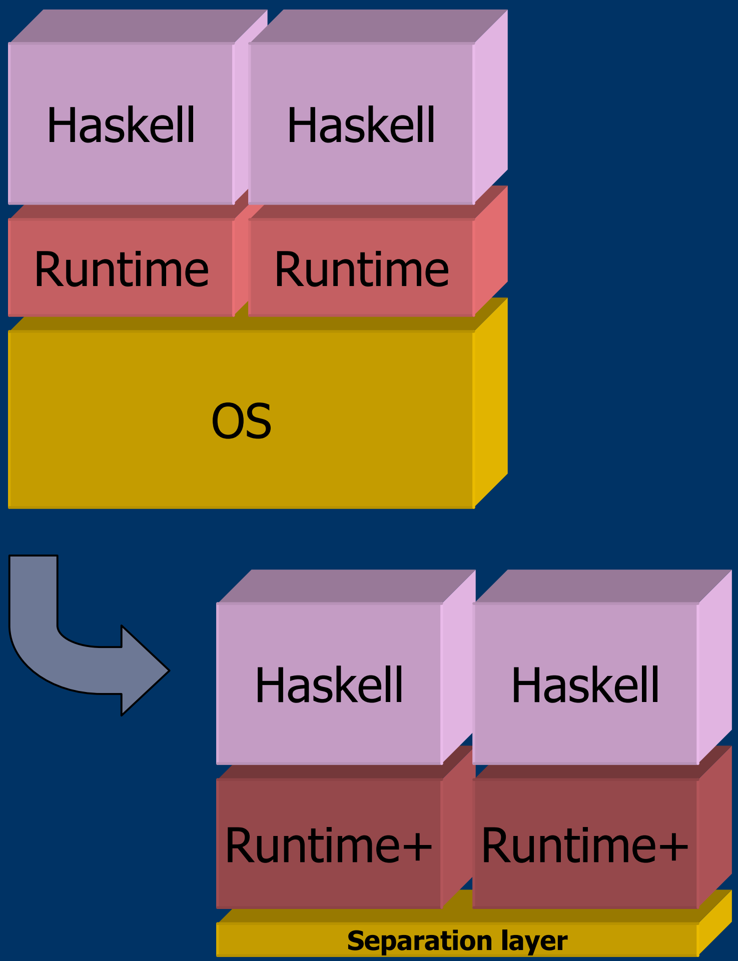
(Semi-) Formal Correspondence

- Models of Haskell's RTS

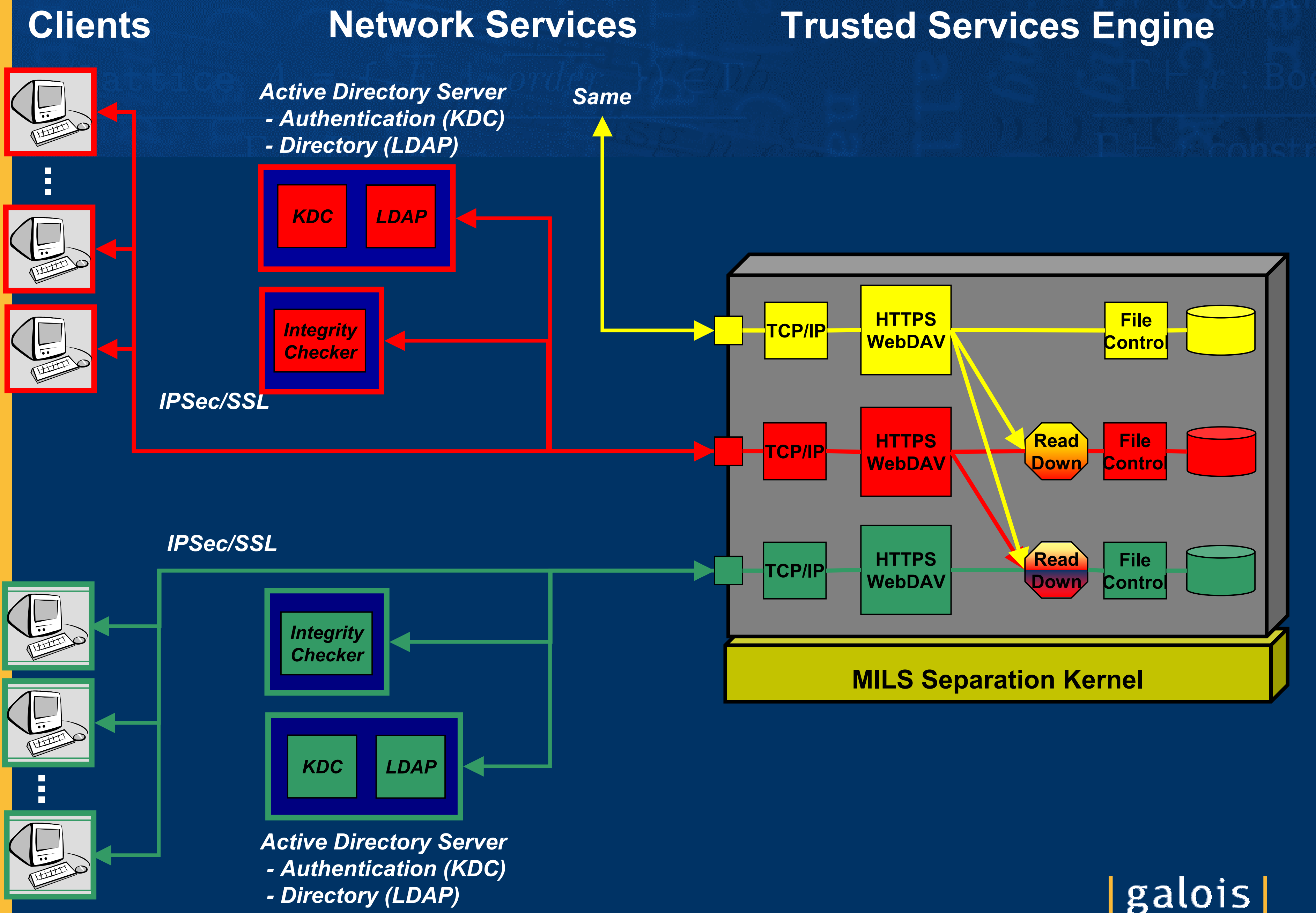


| galois |

Eliminating the OS

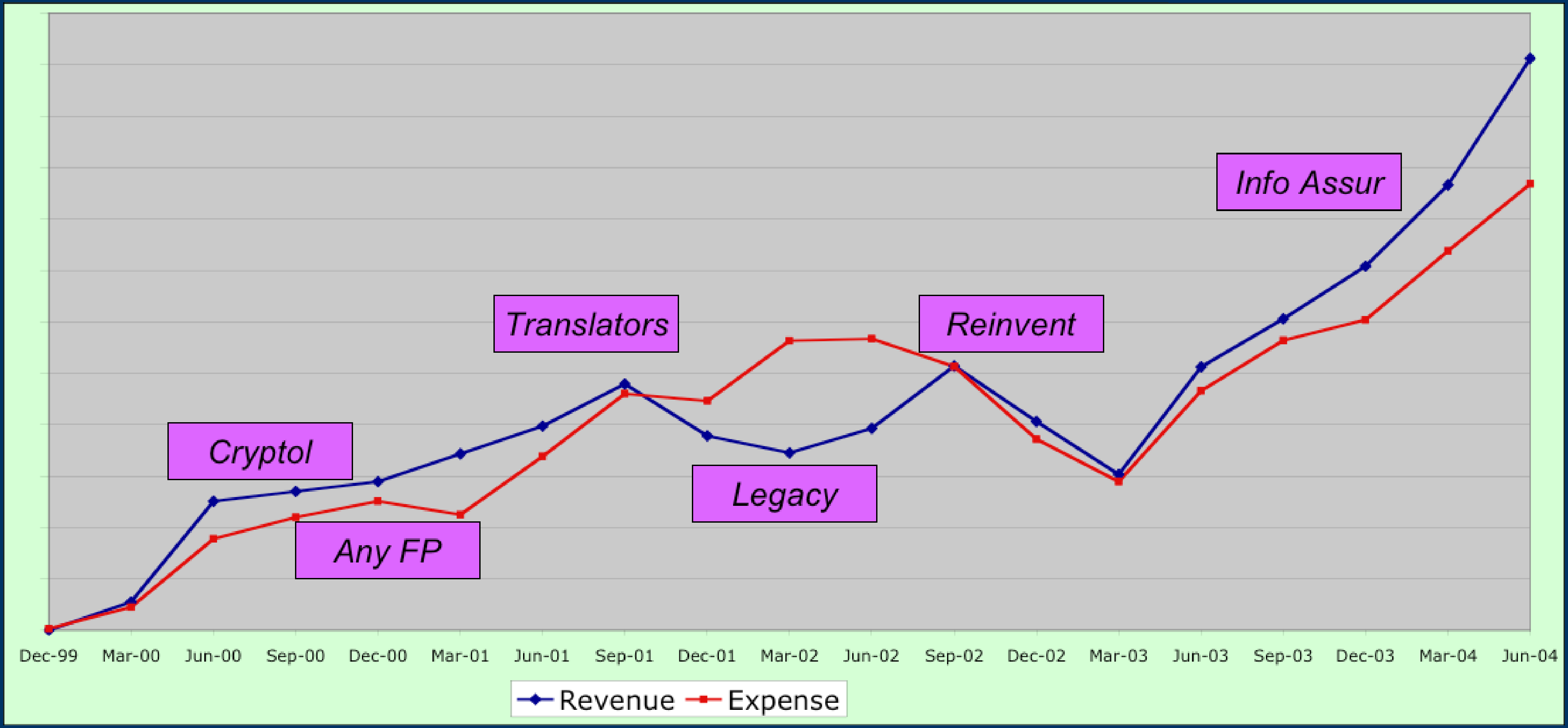


- **Minimize size of system underneath**
  - Extend RTS slightly
  - Host directly on separation layer
  - Formal model for RTS
- **High assurance platform**
  - Separation kernel provides coarse-grained security constraints
  - Haskell types provide fine-grained security assurances
- **Evaluation**
  - Evaluate RTS system once
  - Evaluation focuses on application, not infrastructure



Clients
Network Services
Trusted Services Engine
Active Directory Server
- Authentication (KDC)
- Directory (LDAP)
Same
KDC
LDAP
Integrity Checker
IPSec/SSL
TCP/IP
HTTPS WebDAV
File Control
Read Down
IPSec/SSL
Integrity Checker
KDC
LDAP
Active Directory Server
- Authentication (KDC)
- Directory (LDAP)
MILS Separation Kernel
galois

Focus, focus, focus



galois

Business Benefits of Functional Languages

High Productivity

Executable (semi)-formal method

Quality of engineers

Unique niche

Business Issues of Haskell

Debugging

Libraries

Government requirements

Support

Abstraction addiction

What is FP’s Brand?

Assurance

Types

Values

Cool

Abstraction

Expressive

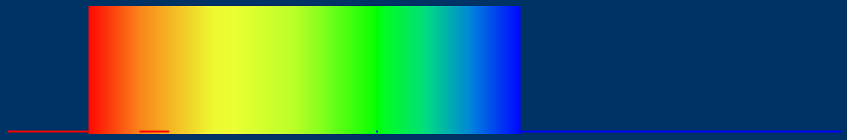
Rapid Dev

Monads

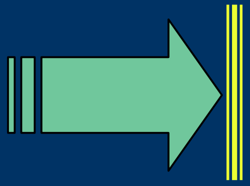
Correctness

Technology directions

- Spectrum from Haskell—HOL



- Control over sensitive values in the heap



Conclusions

- It's been an incredible experience
- FP languages are as good as we hoped
- Business and Technology can go hand in hand