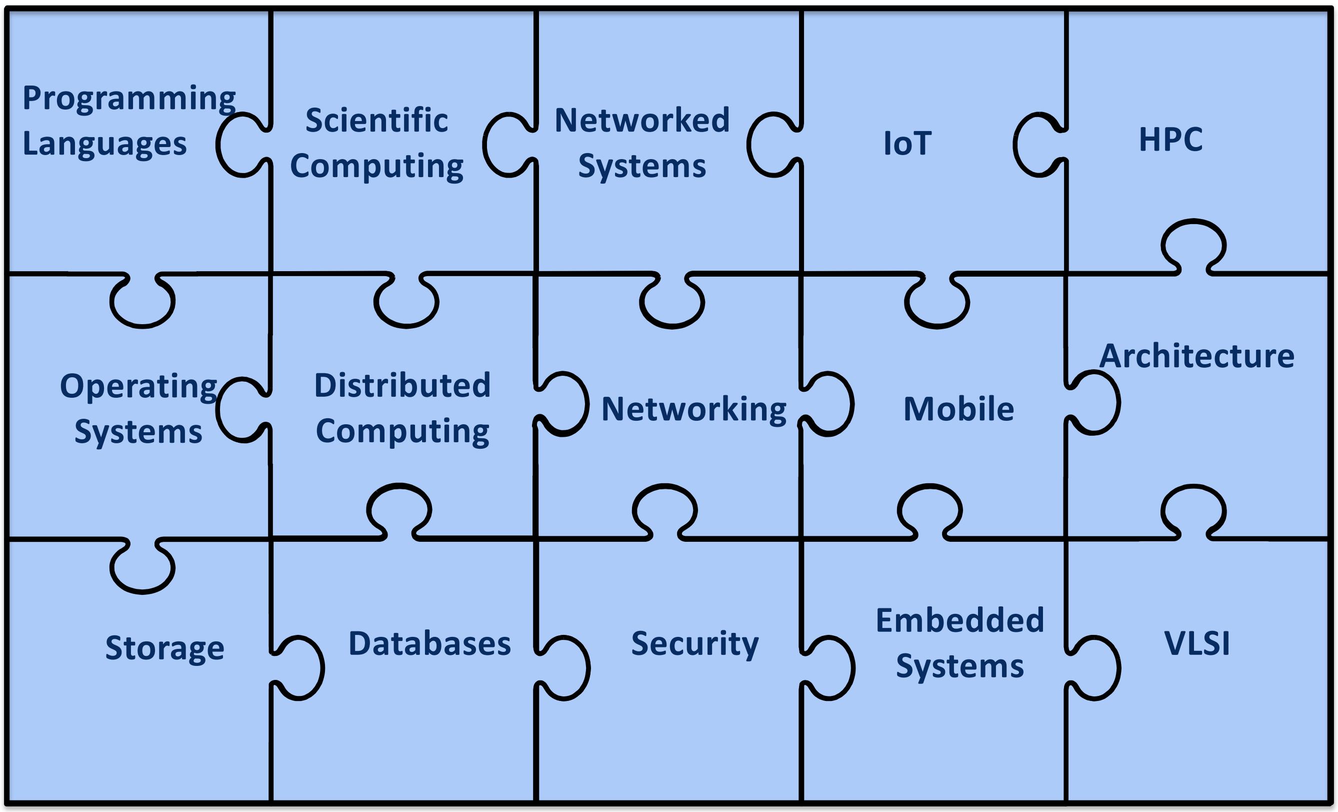
PLOS Must Save the World!

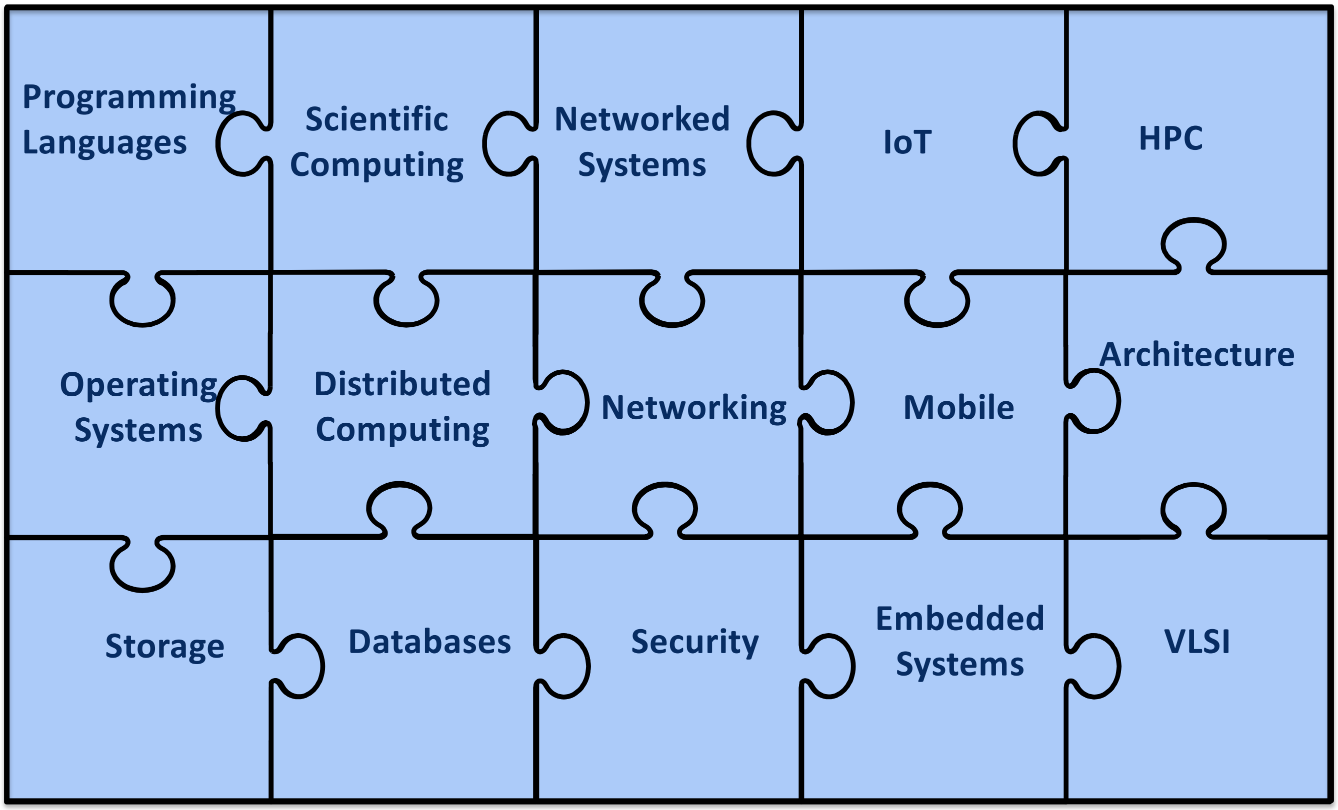
SYSTOPIA

Margo Seltzer
Canada 150 Research Chair in Computer Systems
University of British Columbia

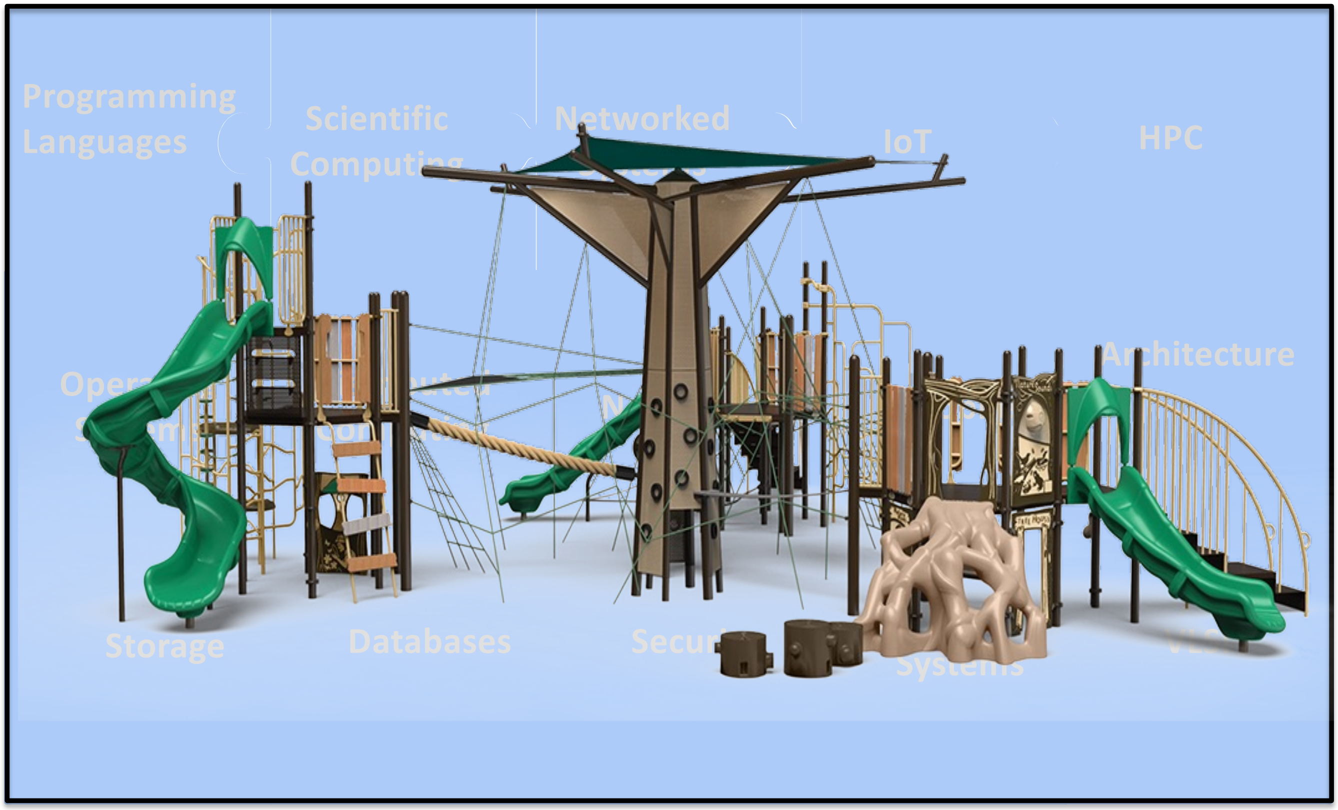
Systems -- Construed Broadly



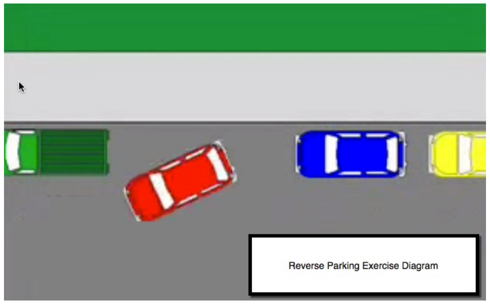
Systems -- Construed Broadly

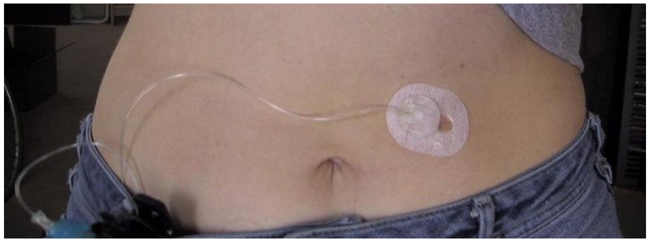


My Playground



Software Rules the World



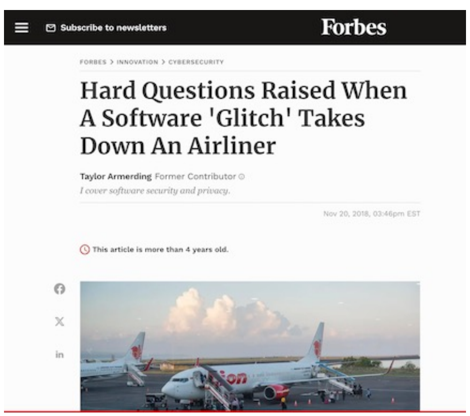


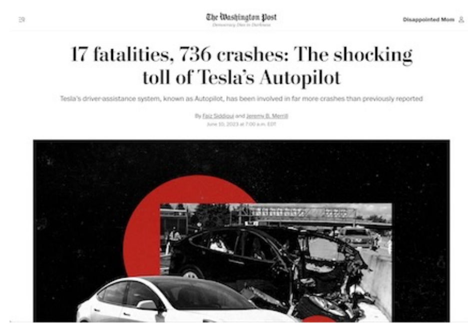
And it Fails …

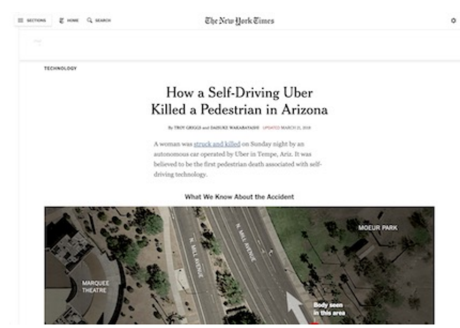

REUTERS
World Business Markets Sustainability More
Energy | White Collar Crime | Public Policy
One password allowed hackers to disrupt Colonial Pipeline, CEO tells senators
By Stephanie Kelly and Jessica Resnick-ault
JOSEPH BLOUNT


NEWS
Tech
Nest thermostat bug leaves users cold
Top Stories
Features
Get the new BBC technology newsletter
Subscribe

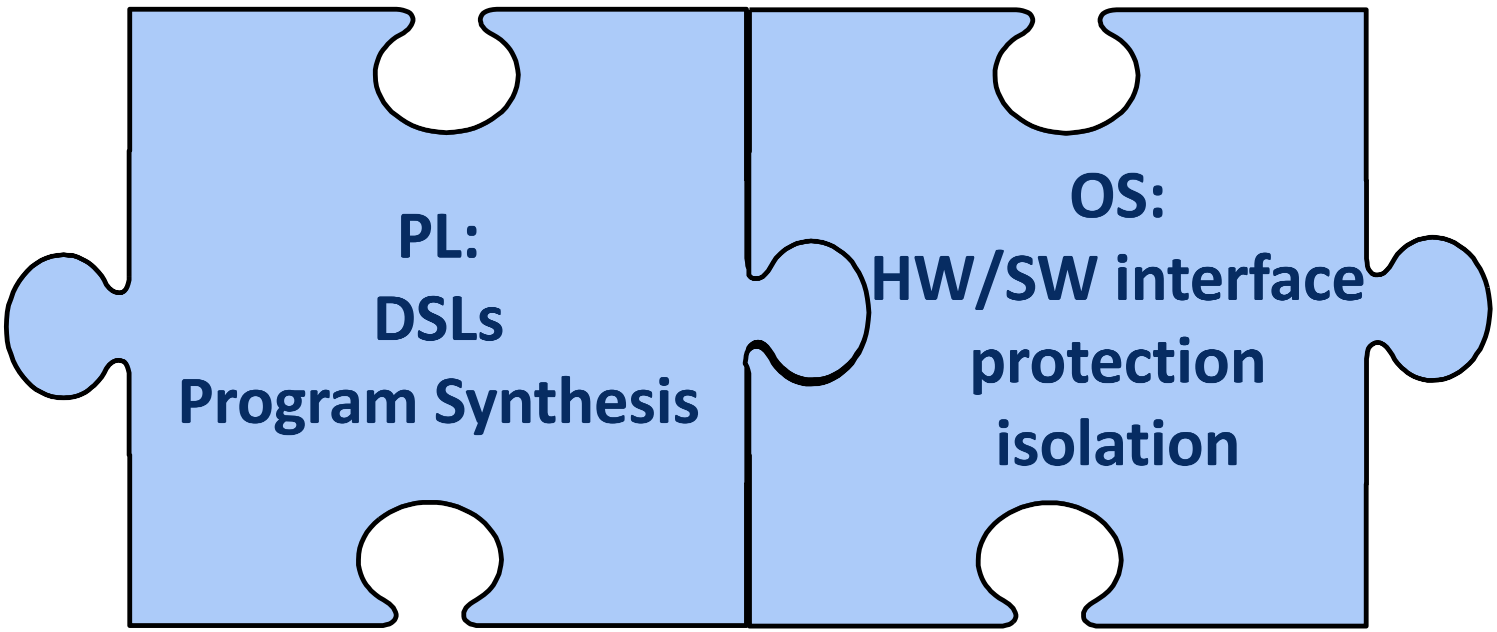

Support the Guardian
The Guardian
News
Opinion
Sport
Culture
Lifestyle
More
Coronavirus
This article is more than 1 year old
Malfunctioning NHS app for Covid vaccine status causes travel delays


Subscribe to newsletters
Forbes
FORBES › INNOVATION › CYBERSECURITY
Hard Questions Raised When A Software 'Glitch' Takes Down An Airliner
Taylor Armerding Former Contributor
I cover software security and privacy.
This article is more than 4 years old.

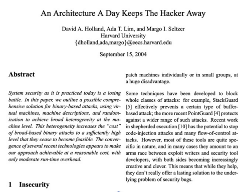

The Washington Post
17 fatalities, 736 crashes: The shocking toll of Tesla's Autopilot


The New York Times
TECHNOLOGY
How a Self-Driving Uber Killed a Pedestrian in Arizona
What We Know About the Accident

An OS/PL Partnership



The Long Road to Program Synthesis



An Architecture A Day Keeps The Hacker Away

David A. Holland, Ada T. Lim, and Margo I. Seltzer
Harvard University
{dholland,ada,margo}@eecs.harvard.edu

September 15, 2004

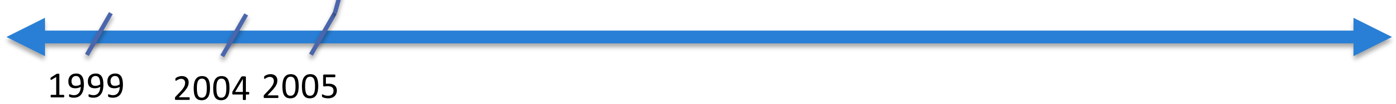
Abstract

System security as it is practiced today is a losing battle. In this paper, we outline a possible comprehensive solution for binary-based attacks, using virtual machines, machine descriptions, and randomization to achieve broad heterogeneity at the machine level. This heterogeneity increases the "cost" of broad-based binary attacks to a sufficiently high level that they cease to become feasible. The convergence of several recent technologies appears to make our approach achievable at a reasonable cost, with only moderate run-time overhead.

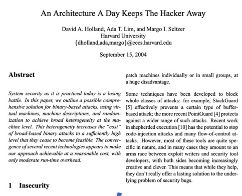
1 Insecurity

patch machines individually or in small groups, at a huge disadvantage.

Some techniques have been developed to block whole classes of attacks: for example, StackGuard [5] effectively prevents a certain type of buffer-based attack; the more recent PointGuard [4] protects against a wider range of such attacks. Recent work in shepherded execution [10] has the potential to stop code-injection attacks and many flow-of-control attacks. However, most of these tools are quite specific in nature, and in many cases they amount to an arms race between exploit writers and security tool developers, with both sides becoming increasingly creative and clever. This means that while they help, they don't really offer a lasting solution to the underlying problem of security bugs.



The Long Road to Program Synthesis



An Architecture A Day Keeps The Hacker Away

David A. Holland, Ada T. Lim, and Margo I. Seltzer
Harvard University
{dholland,ada,margo}@eecs.harvard.edu

September 15, 2004

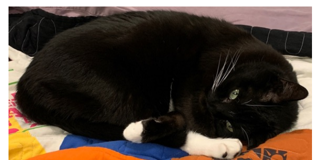
Abstract

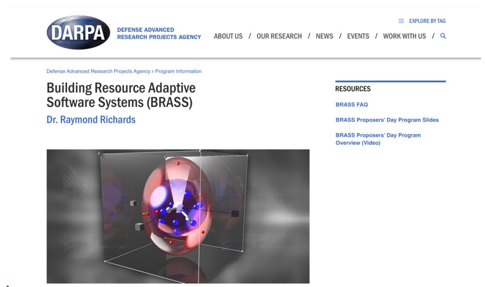
System security as it is practiced today is a losing battle. In this paper, we outline a possible comprehensive solution for binary-based attacks, using virtual machines, machine descriptions, and randomization to achieve broad heterogeneity at the machine level. This heterogeneity increases the "cost" of broad-based binary attacks to a sufficiently high level that they cease to become feasible. The convergence of several recent technologies appears to make our approach achievable at a reasonable cost, with only moderate run-time overhead.

1 Insecurity

patch machines individually or in small groups, at a huge disadvantage.

Some techniques have been developed to block whole classes of attacks: for example, StackGuard [5] effectively prevents a certain type of buffer-based attack; the more recent PointGuard [4] protects against a wider range of such attacks. Recent work in shepherded execution [10] has the potential to stop code-injection attacks and many flow-of-control attacks. However, most of these tools are quite specific in nature, and in many cases they amount to an arms race between exploit writers and security tool developers, with both sides becoming increasingly creative and clever. This means that while they help, they don't really offer a lasting solution to the underlying problem of security bugs.





DARPA DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

EXPLORE BY TAG

ABOUT US / OUR RESEARCH / NEWS / EVENTS / WORK WITH US /

Defense Advanced Research Projects Agency > Program Information

Building Resource Adaptive Software Systems (BRASS)

Dr. Raymond Richards

RESOURCES

BRASS FAQ

BRASS Proposers' Day Program Slides

BRASS Proposers' Day Program Overview (Video)

Dormant

1999 2004 2005 2015

BRASS Program

Modern-day software operates within a complex ecosystem of libraries, models, protocols and devices. Ecosystems change over time in response to new technologies or paradigms, as a consequence of repairing discovered vulnerabilities (security, logical, or performance-related), or because of varying resource availability and reconfiguration of the underlying execution platform. When these changes occur, applications may no longer work as expected because their assumptions on how the ecosystem should behave may have been inadvertently violated.

Ensuring applications can seamlessly continue to operate correctly and usefully in the face of such changes is a formidable challenge. Failure to effectively and timely adapt to ecosystem evolution can result in technically inferior and potentially vulnerable systems, but the lack of automated mechanisms to restructure and transform applications when changes do occur leads to high software maintenance costs and premature obsolescence of otherwise functionally sound systems. Neither of these outcomes is desirable and poses significant risk to economic productivity and cyber resilience.

Successfully adapting applications to an evolving ecosystem requires mechanisms to infer the impact of such evolution on application behavior and performance, automatically trigger transformations that beneficially exploit these changes and provide validation that these transformations are correct. To do so requires the ability to: (a) extract whole-system specifications over the entire software stack that can be used to define application-centric descriptions of the resources provided by the ecosystem; (b) leverage new programming abstractions, program analyses, and compilation methodologies to correlate application behavior with salient ecosystem changes; (c) develop semantics-preserving program transformations designed with adaptation in mind; and (d) exploit new runtime systems and virtual machine implementations structured to facilitate the efficient integration of these transformations.

The goal of the Building Resource Adaptive Software Systems program (BRASS) is to realize foundational advances in the design and implementation of long-lived, survivable and complex software systems that are robust to changes in the physical and logical resources provided by their ecosystem. These advances will necessitate integration of new resource-aware program abstractions and analyses, in addition to novel compiler and systems designs to trigger adaptive transformations and validate their effectiveness.

TAGS

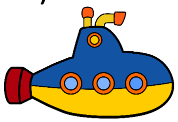
…evolving ecosystem requires mechanisms to infer the impact of such evolution on application behavior and performance, automatically trigger transformations that beneficially exploit these changes and provide validation that these transformations are correct.

extract whole-system specifications

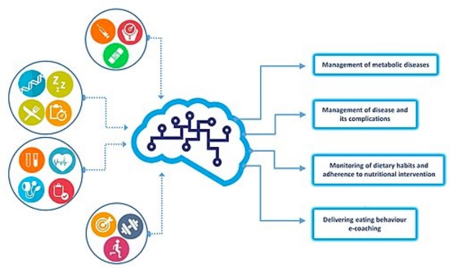
leverage new programming abstractions, program analyses, and compilation methodologies to correlate application behavior with salient ecosystem changes.

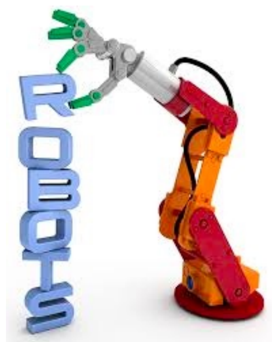
exploit new runtime systems and virtual machine implementations structured to facilitate the efficient integration of these transformations.

We (PRINCESS) are here

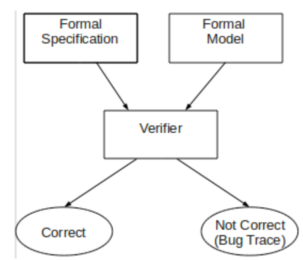


BRASS Performers

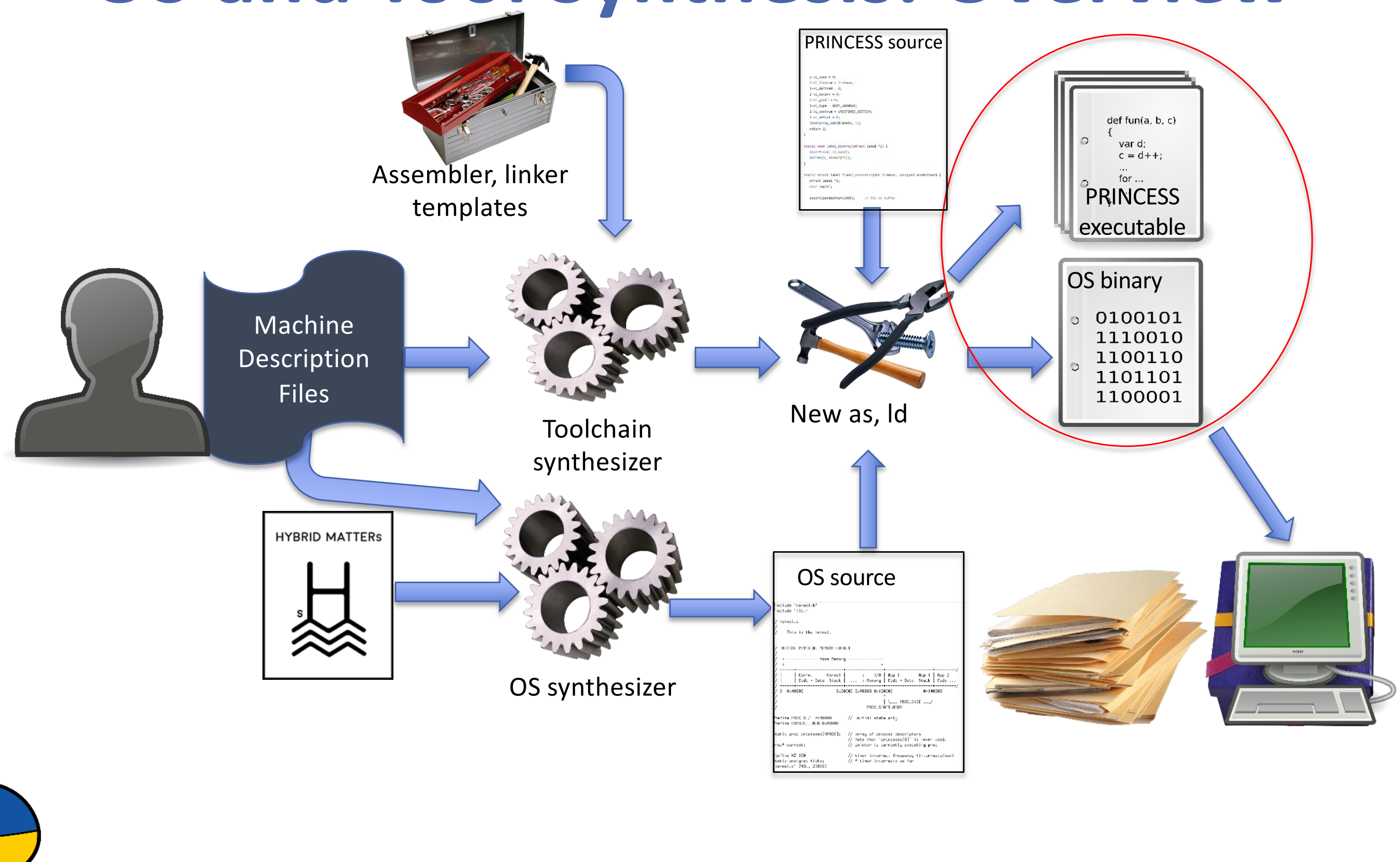




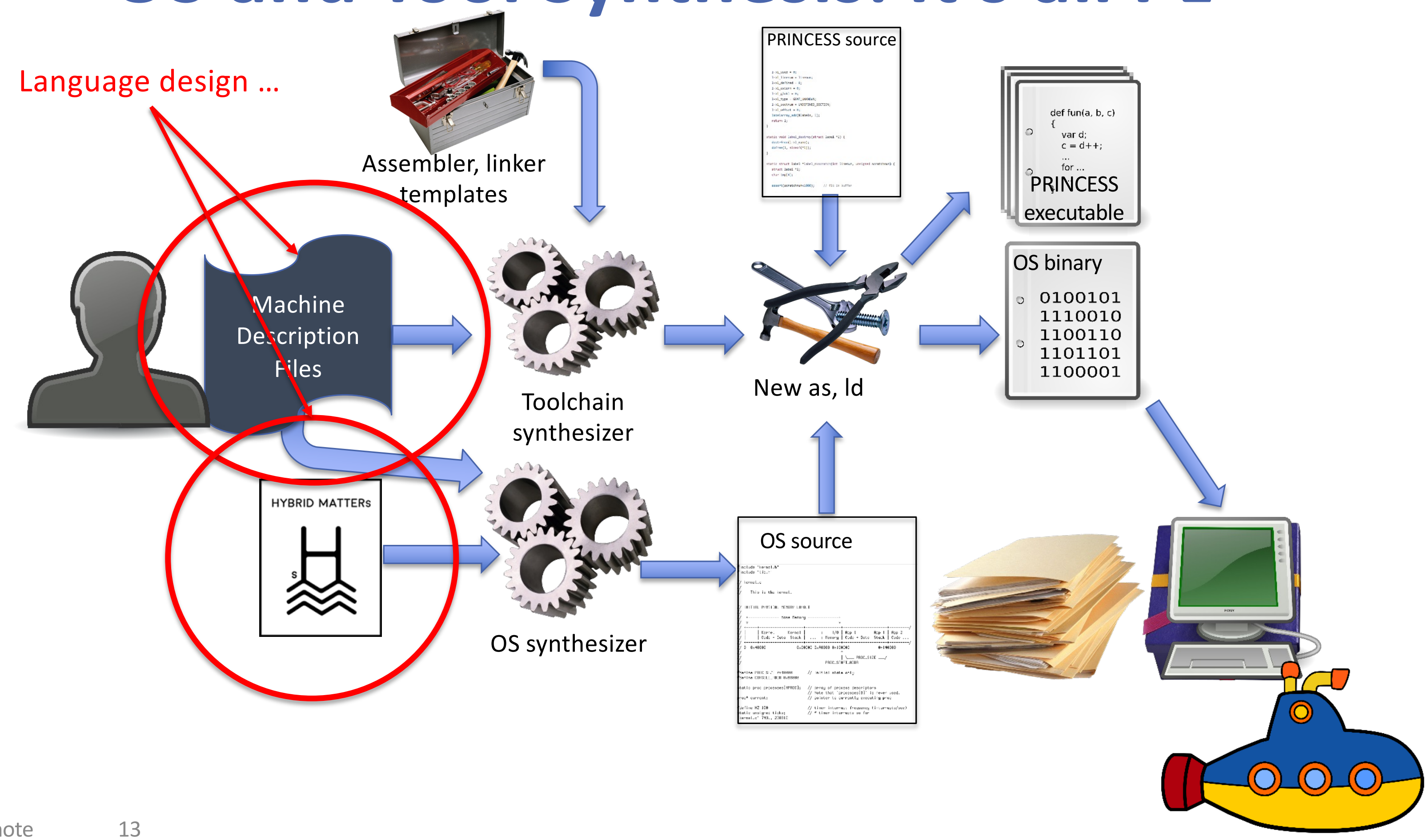
λ



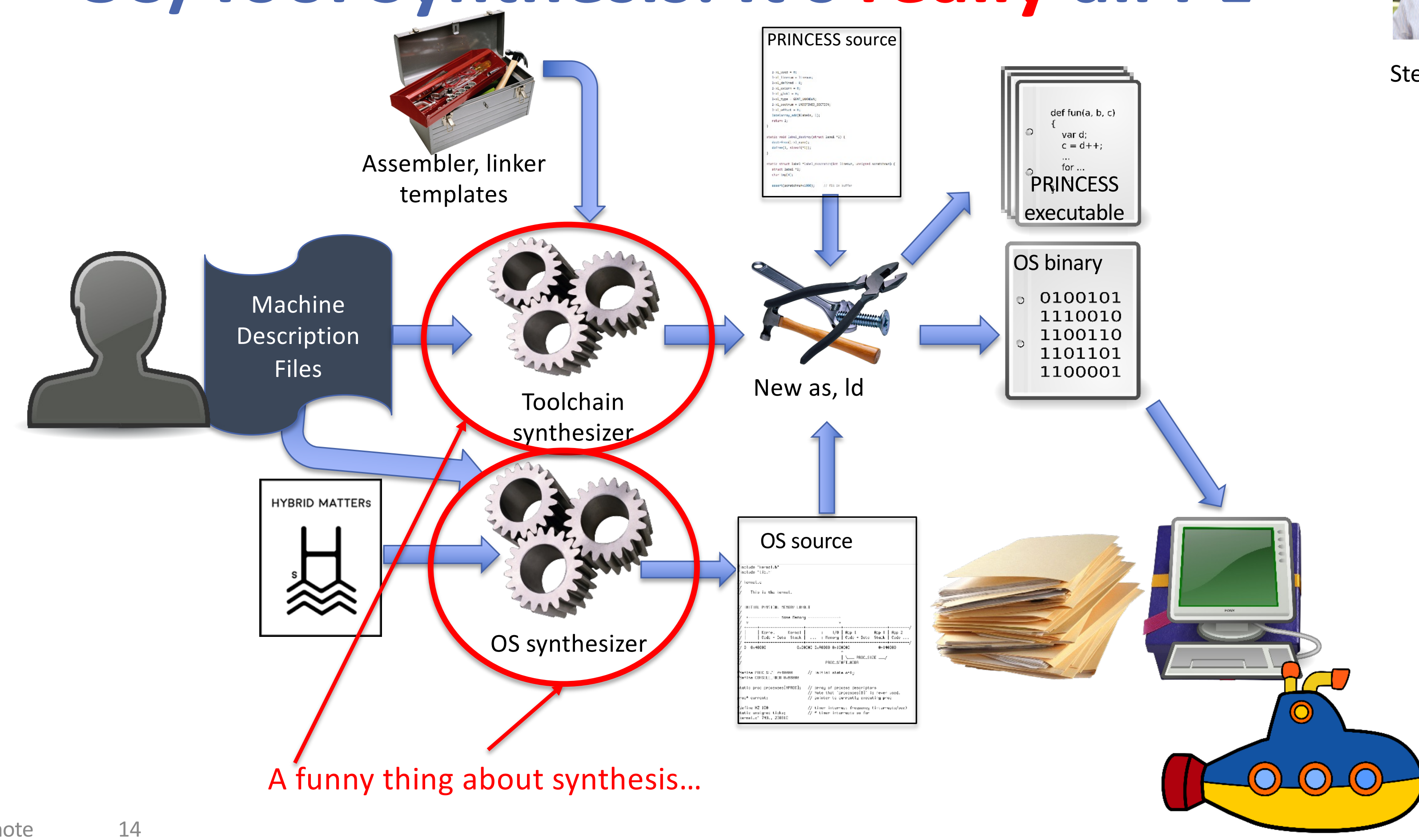
OS and Tool Synthesis: Overview



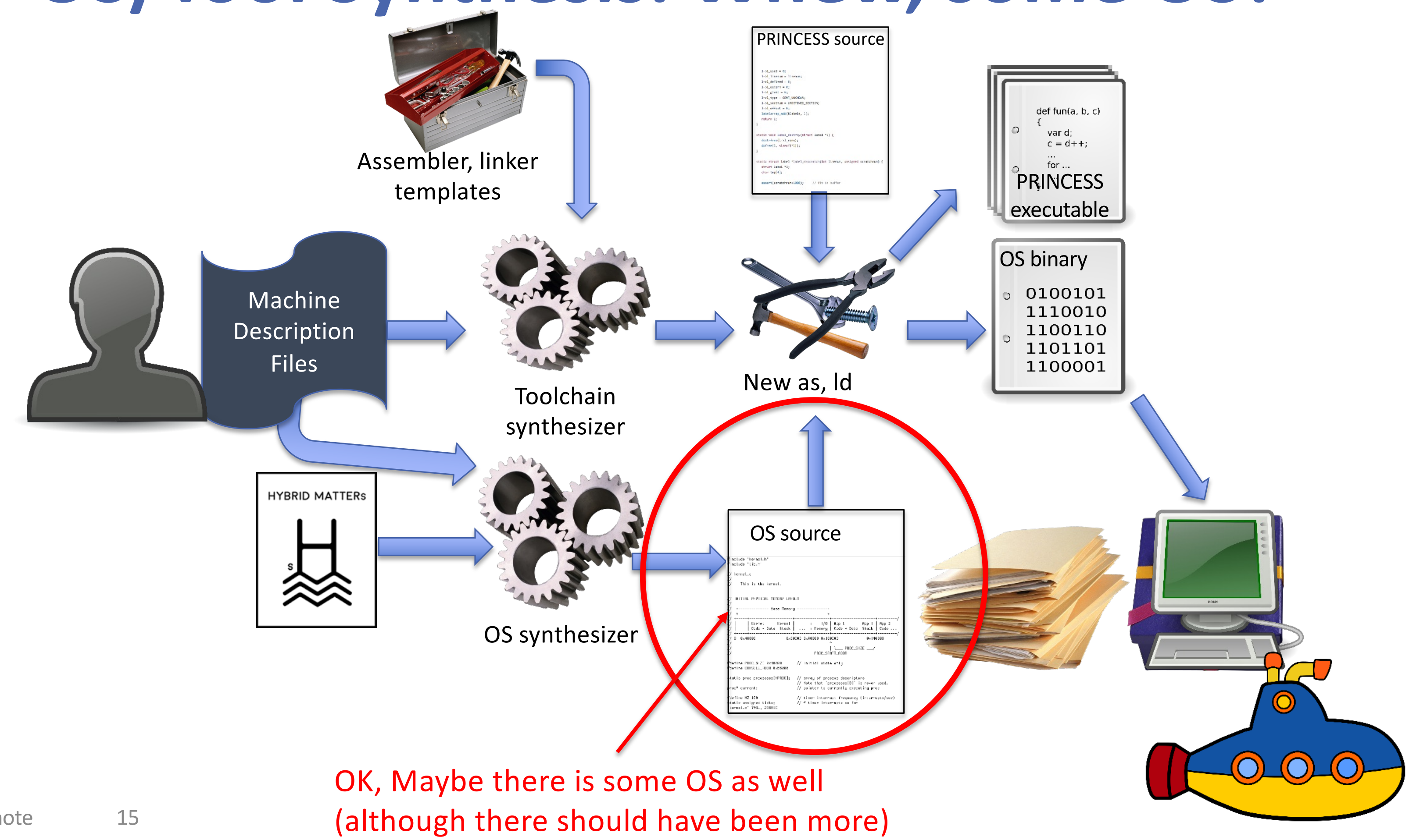
OS and Tool Synthesis: It's all PL



OS/Tool Synthesis: It's **really** all PL

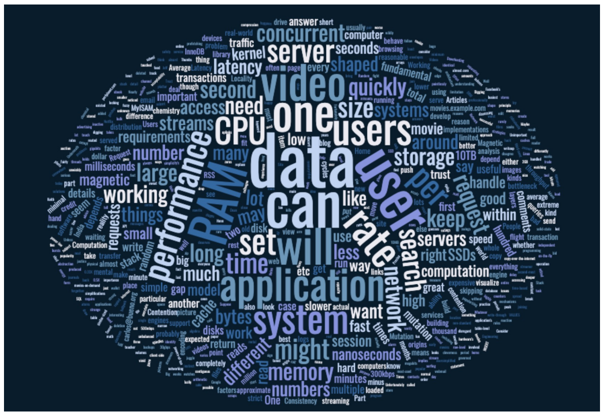


OS/Tool Synthesis: Whew, some OS!

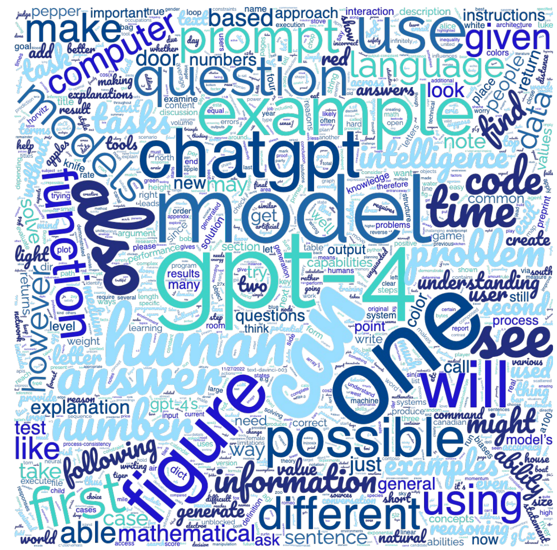


Real Systems Work!

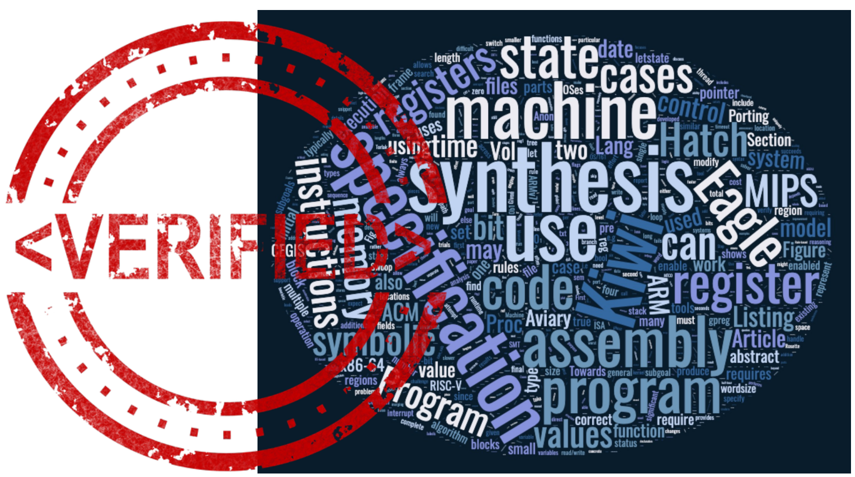
Full stack developer today



Or maybe this is now today?



Full stack developer tomorrow?

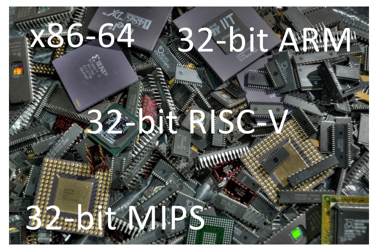


How far did we get?

Two Operating Systems

OS/161

Four Processors



17 Use Cases

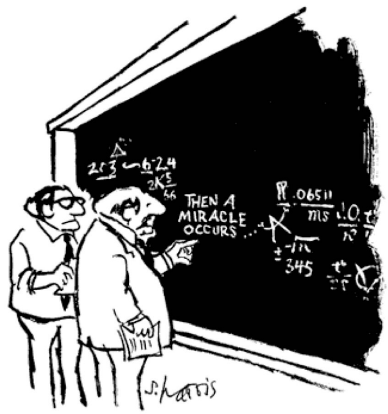
```
792415C0    55          push ebp
792415C1    89E5        mov ebp, esp
792415C3    8B45 08     mov eax, [ebp+0x08]
792415C6    DB28        fld tword [eax]
792415C8    8B4D 0C     mov ecx, [ebp+0x0C]
792415CB    DB29        fld tword [ecx]
792415CD    DEC1        faddp
792415CF    8B55 10     mov edx, [ebp+0x10]
792415D2    DB3A        fstp tword [edx]
792415D4    DB68 0A     fld tword [eax+0x0A]
792415D7    DB69 0A     fld tword [ecx+0x0A]
792415DA    DEC1        faddp
792415DC    DB7A 0A     fstp tword [edx+0x0A]
792415DF    5D          pop ebp
792415E0    C2 0C00     ret 0x000C
```

Use Case	Eagle	Verification Time (ms)				Synthesis Time (s)				Assembly (lines)			
	(lines)	ARM	MIPS	RISC-V	x86-64	ARM	MIPS	RISC-V	x86-64	ARM	MIPS	RISC-V	x86-64
SJ	9	150	260	—	130	43	140	—	13	12	12	15	9
LJ	11	180	320	—	150	—	—	—	—	(14)	(13)	(16)	(12)
CRT-i	10	46	73	55	48	0.08	2.9	1.1	0.08	0	1	1	0
CRT-s	10	53	78	60	53	6.2	9.0	11	0.50	4	2	4	2
SYS	6	12	15	13	9	0.69	2.7	1.1	0.09	1	1	1	1
IRQ	4	12	19	—	9	0.47	33	—	0.09	1	3	1	1
TS	23	1300	2600	—	1900	—	—	—	—	(23)	(26)	(30)	(20)
TS-e	7	12	15	13	9	0.90	3.1	1.3	0.10	1	1	1	1
TS-s	8	48	74	55	50	0.62	2.9	1.1	0.15	1	1	1	1
TS-l	8	48	74	55	49	1.1	2.9	1.1	0.16	1	1	1	1
TS-c	7	12	14	13	9	0.87	3.1	1.1	0.10	1	1	1	1
IS	12	130	170	140	63	4.7	14	5.8	0.18	2	2	2	1
GD	14	260	340	270	150	19	52	23	0.84	3	3	4	2
CD	12	52	77	58	53	2.9	9.7	3.9	0.23	2	2	2	1
CL	16	51	76	59	55	190	210	440	0.24	3	3	4	1
CH	16	52	76	57	54	—	13	5.1	0.24	(4)	2	2	1
SA	13	14	16	38	11	46	120	140	12	3	3	3	4

Assessment

- Was the project an unqualified success? **No!**
- Was the project a research success? **Yes!**
- Did we learn a ton? **Yes!**
- Has it informed future work? **Yes!**
- Should we have done it? **Yes!**

What Did we Learn?



"I think you should be more explicit here in step two."

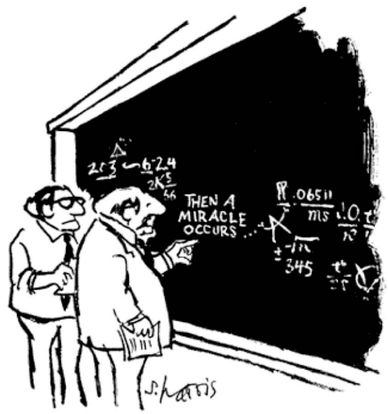
One Miracle

Experiment Early and Often

Cost of spec >= Value

Machine-Dependent != Machine Specific

What Did we Learn?



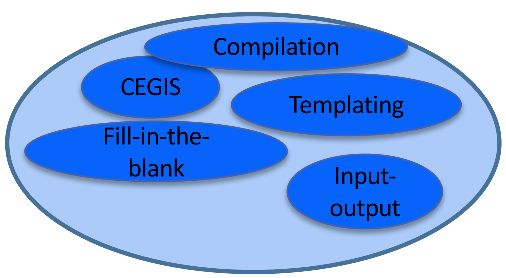
"I think you should be more explicit here in step two."

One Miracle

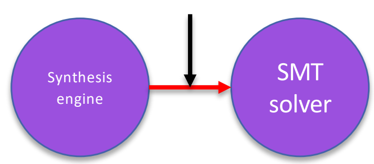
Experiment Early and Often

Cost of spec >= Value

Machine-Dependent != Machine Specific



Matters a lot!



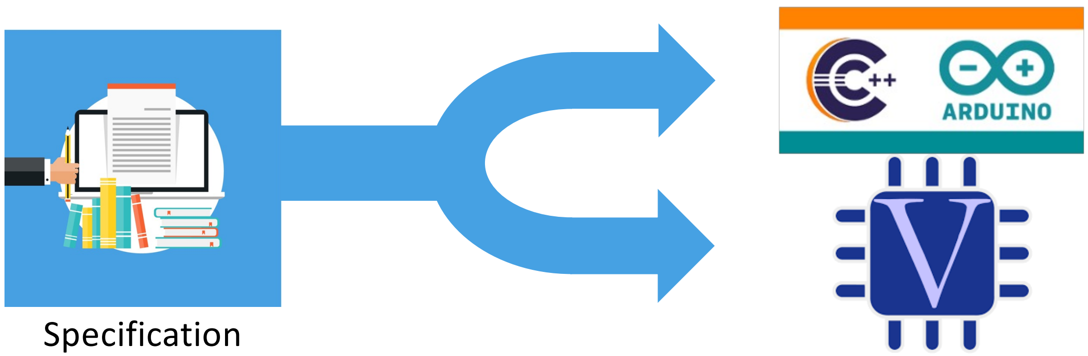
Synthesizing Concurrent Programs

Christopher Chen, with Mark Greenstreet

In the beginning: There was COMET
Tractable Reactive Program Synthesis

Memoization in
Program Synthesis

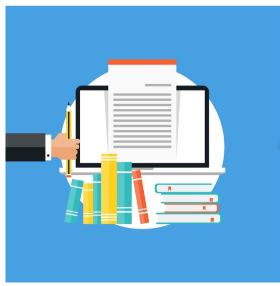
COMET



Specification

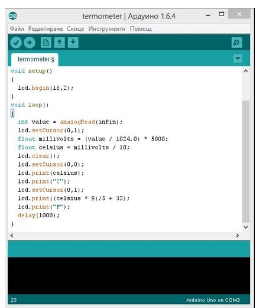
Implementations

A Solution

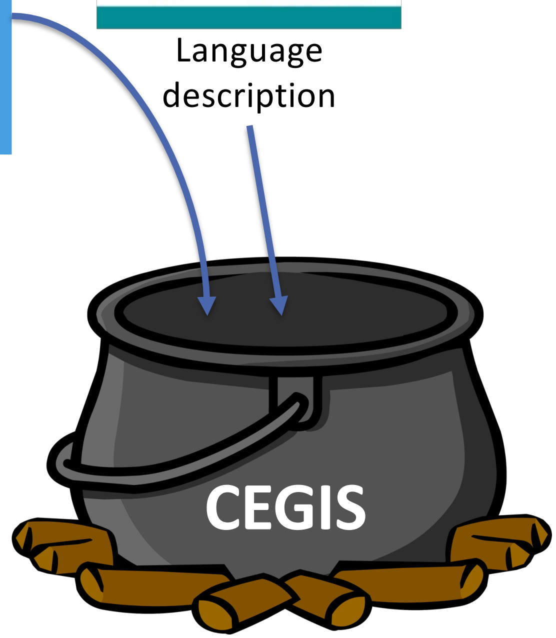


Specification

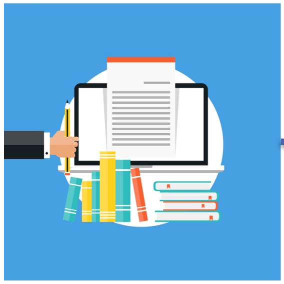
Language description



Arduino C Program



A Solution



Specification

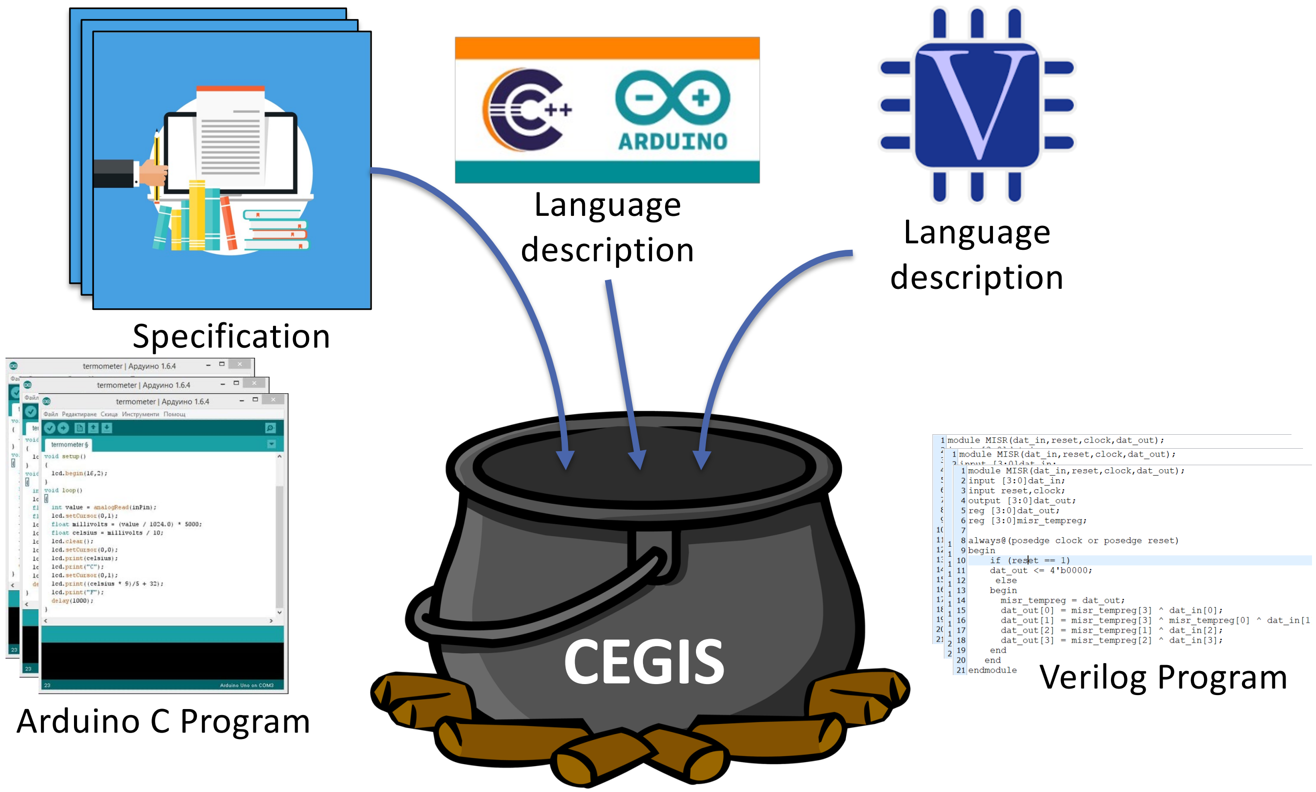
Language description


```
module MISR(dat_in,reset,clock,dat_out);
input [3:0]dat_in;
input reset,clock;
output [3:0]dat_out;
reg [3:0]dat_out;
reg [3:0]misr_tempreg;

always@(posedge clock or posedge reset)
begin
    if (reset == 1)
    dat_out <= 4'b0000;
     else
    begin
      misr_tempreg = dat_out;
      dat_out[0] = misr_tempreg[3] ^ dat_in[0];
      dat_out[1] = misr_tempreg[3] ^ misr_tempreg[0] ^ dat_in[1
      dat_out[2] = misr_tempreg[1] ^ dat_in[2];
      dat_out[3] = misr_tempreg[2] ^ dat_in[3];
    end
   end
endmodule
```

Verilog Program
(Hardware description)

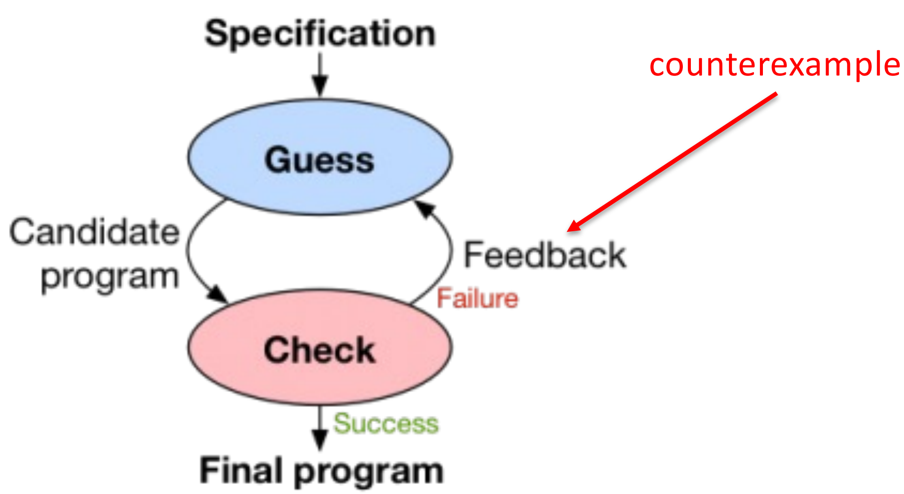
A Solution



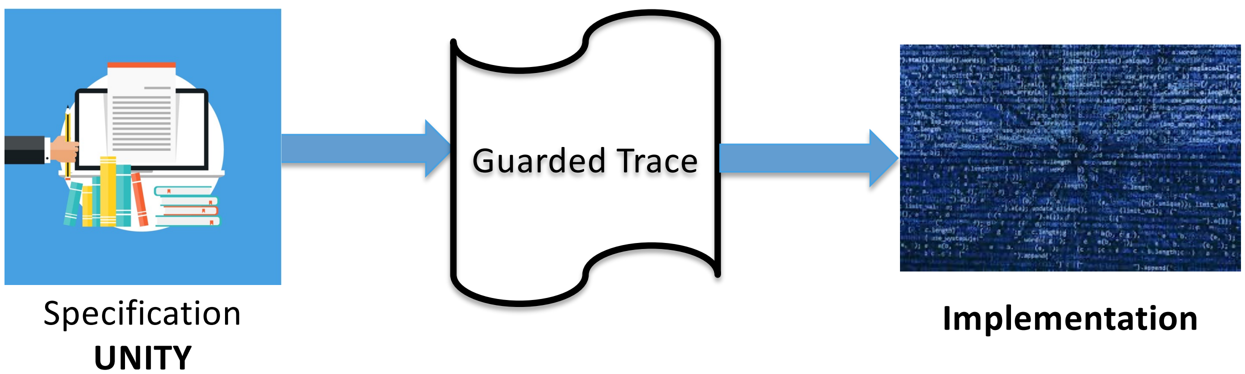
Specification
ARDUINO
Language description
Language description
CEGIS
Arduino C Program
Verilog Program

CEGIS

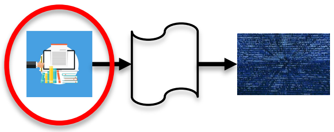
- Counterexample-Guided Inductive Synthesis
- AKA: Guess and Check



COMET Made Simple



COMET: UNITY Specifications



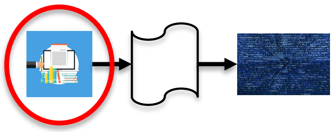
- UNITY*: Unbounded Nondeterministic Iterative Transformations

```
declare:
        in1, in2, out: integer
        cntrl: bool
initially
        out = 0
assign
        out = in1 if (cntrl)
        out = in2 if (not(cntrl))
```

conditions (guards)

*K. Mani Chandy and J. Misra. Parallel program design: a foundation, 1988.

COMET: UNITY Specifications



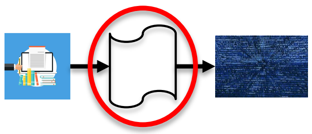
- UNITY*: Unbounded Nondeterministic Iterative Transformations

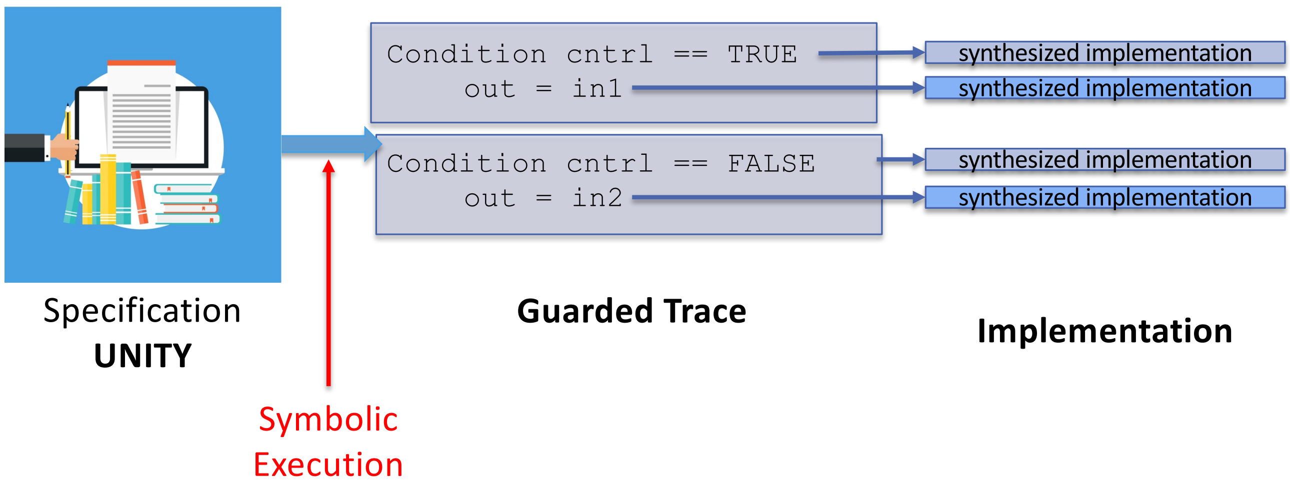
```
declare:
	in1, in2, out: integer
	cntrl: bool
initially
	out = 0
assign
	out = in1 if (cntrl)
	out = in2 if (not(cntrl))
```

assignments

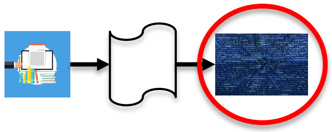
*K. Mani Chandy and J. Misra. Parallel program design: a foundation, 1988.

COMET Guarded Traces





COMET Synthesis Detail

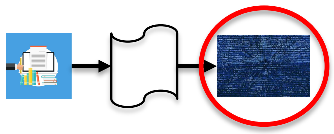


Specification Language **GOAL Language**

1. Verify that the specification language type checks
2. Type mapping between the spec and target languages
3. Use Rosette (a Racket-based framework for synthesis, verification and other things)

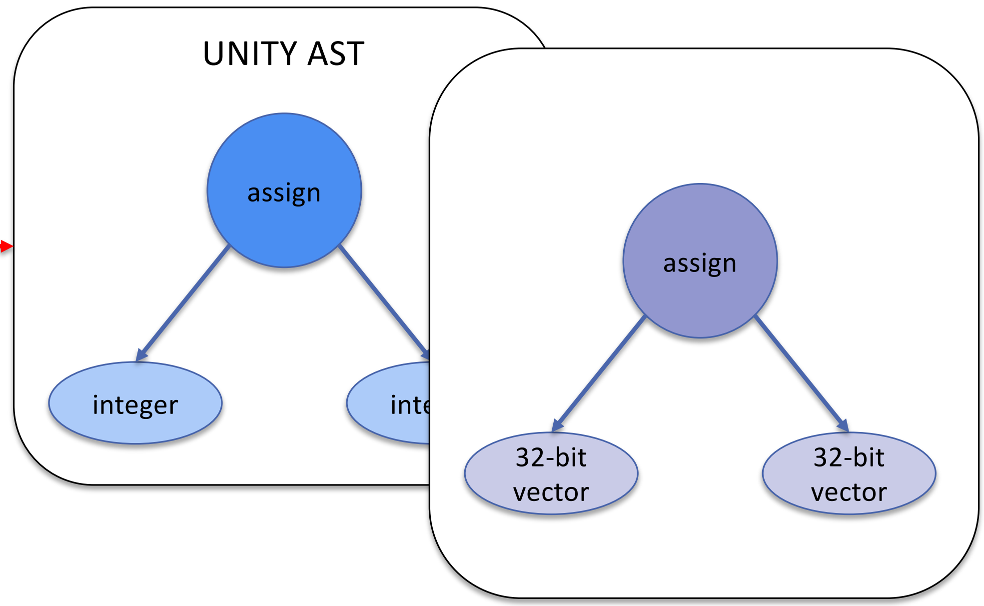
out = in1 if (cntrl)
out = in2 if (not(cntrl))

Wait! Observe the regularity in our expressions



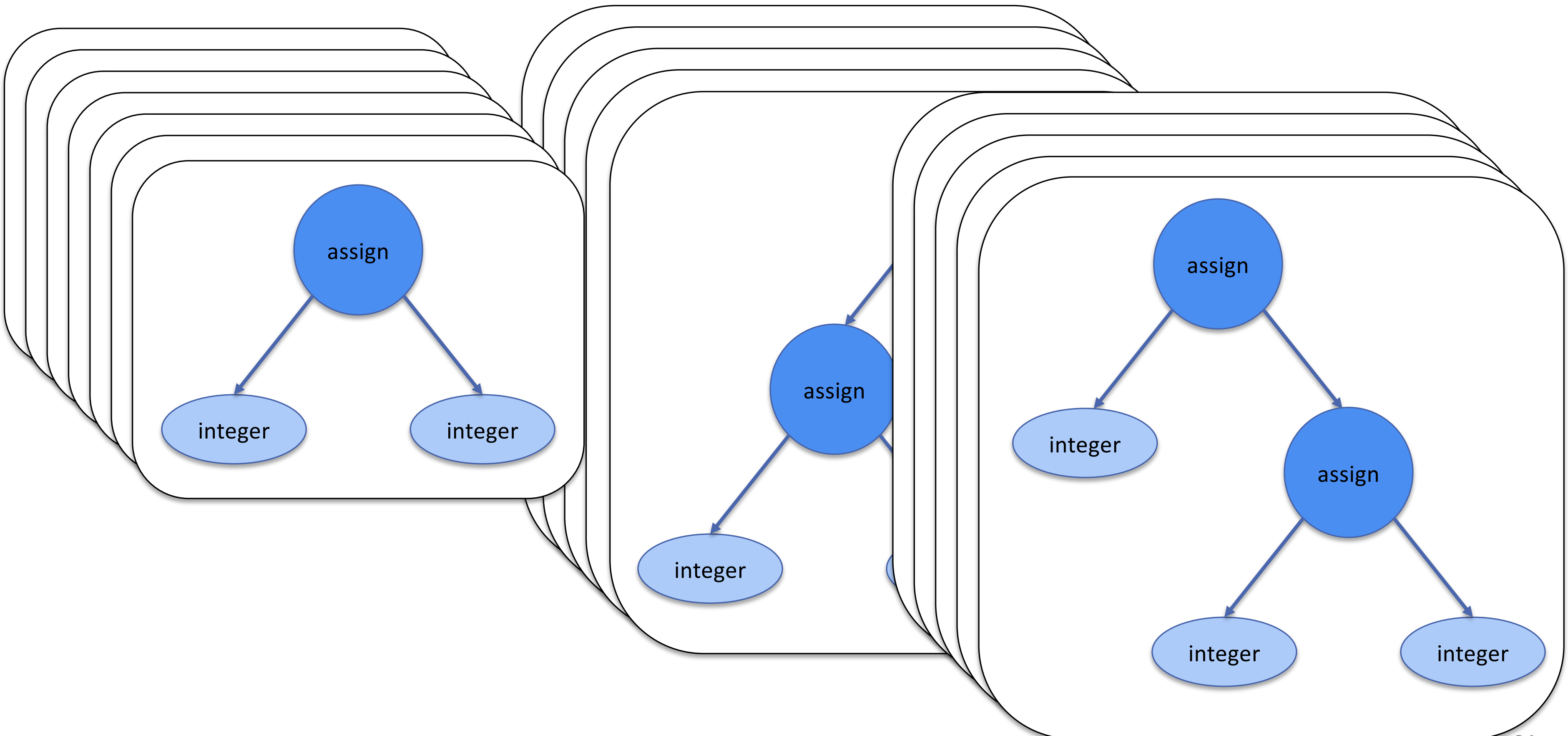
```
Condition cntrl == TRUE
    out = in1

Condition cntrl == FALSE
    out = in2
```

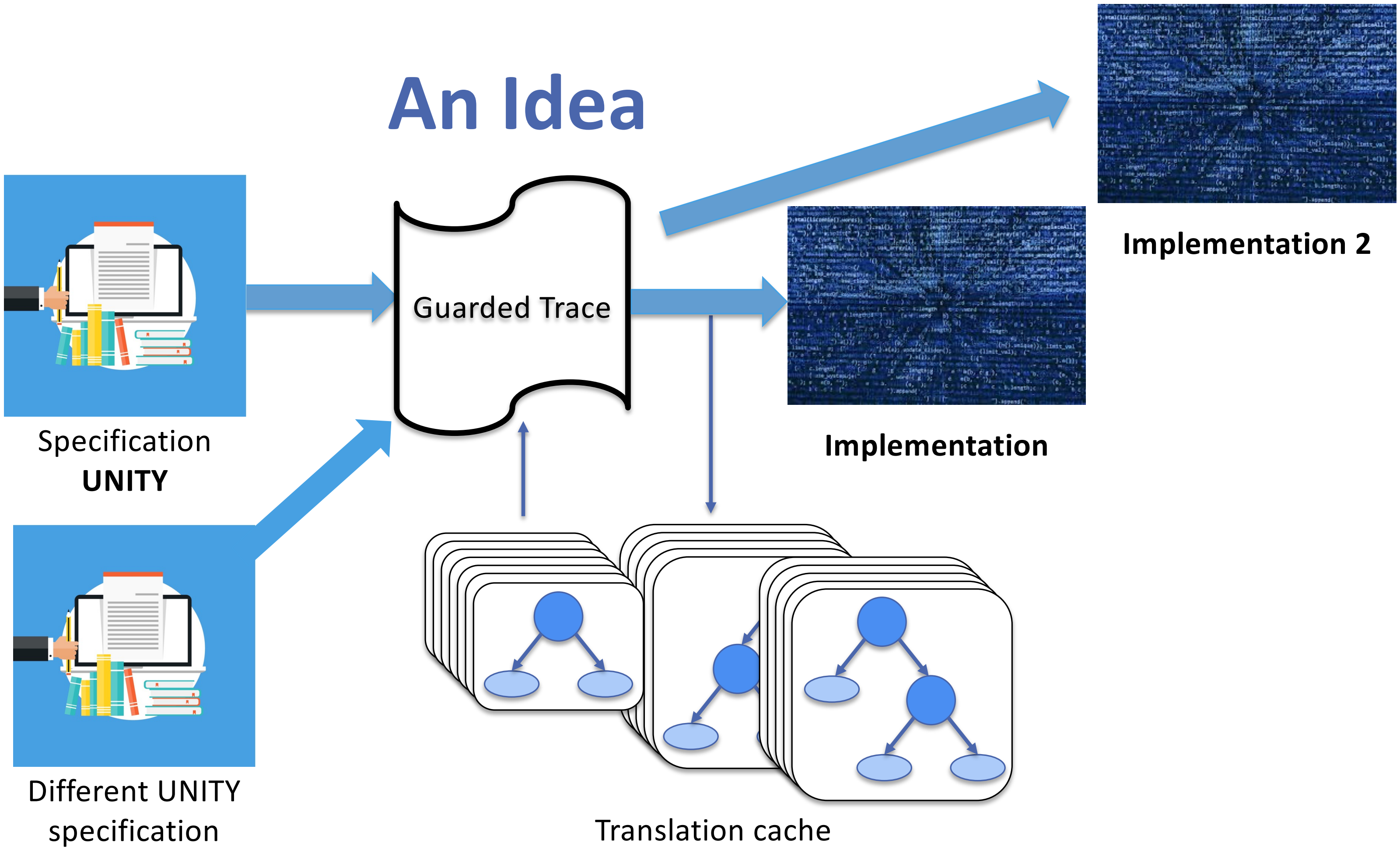


out[0:31] = in1[0:31]

But Wait! There's More …



An Idea


Guarded Trace
Implementation 2
Implementation
Specification
UNITY
Different UNITY
specification
Translation cache

Shellac: A Compiler Synthesizer for Concurrent Programs

Christopher Chen, with Mark Greenstreet

From COMET to Shellac

unity

Specification of
the UNITY
language

Specification
UNITY

Language
description

2. Type mapping between the
spec and target languages

Guarded Trace

Implementation

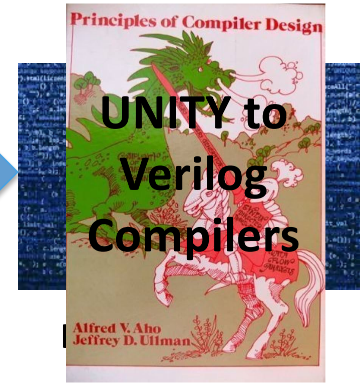
Compiler Template

Chen, C., Seltzer, M., Greenstreet, M., "Shellac: a compiler synthesizer for concurrent programs," Conference on Verified Software: Theories, Tools, and Experiments (VSTTE-2022).

From COMET to Shellac

unity

Specification of
the **UNITY**
language



Language
description

2. Type mapping between the
spec and target languages

Compiler Template

Chen, C., Seltzer, M., Greenstreet, M., “Shellac: a compiler synthesizer for concurrent programs,” Conference on Verified Software: Theories, Tools, and Experiments (VSTTE-2022).

Velosiraptor

HotOSXIX

UBC

Why Program Yourself when you can Synthesize OS CODE?

Reto Achermann, Ryan Mehri, Em Chu, Ilias Karimalis

University of British Columbia
Department of Computer Science
Systopia Research Laboratory

Synthesizing Device Drivers with GHOST WRITER

Jerry Wang, Sepehr Noorafshan, Reto Achermann and **Margo Seltzer**

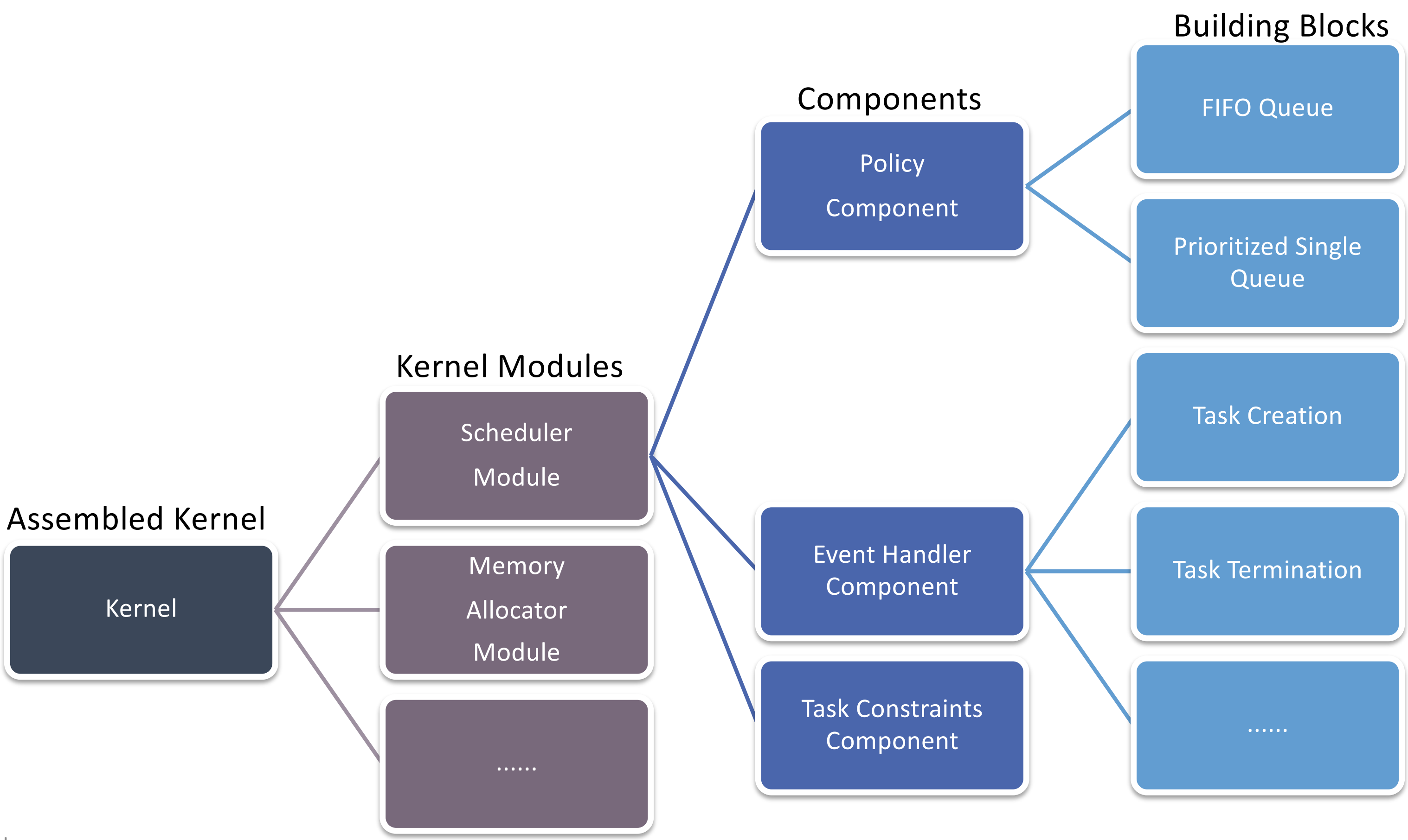
October 2023

Tinkertoy: Build your own Operating Systems for IoT Devices

- Bootloader
- Memory Allocator
- Scheduler
- IPC & RPC Primitives
- System Call
- Execution Context
- Kernel Service Routines
- Context Switcher
- Dispatcher
- Execution Model
- Service Identifier Finder
- Service Routine Mapper
- File System
- TCB Components
- Component Constraints
- Networking Support

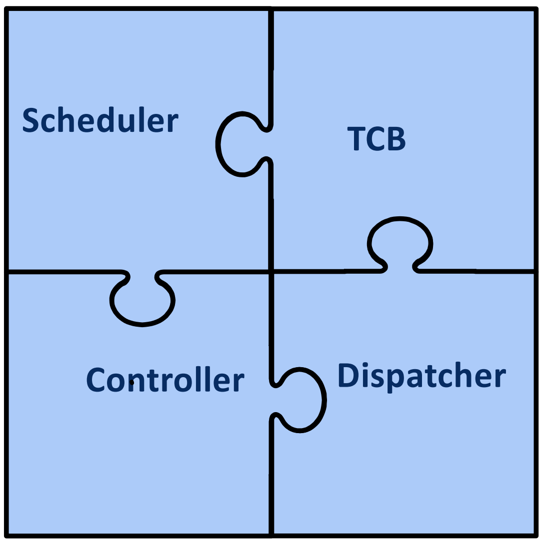
EMSOFT 2022

Tinkertoy Module Architecture

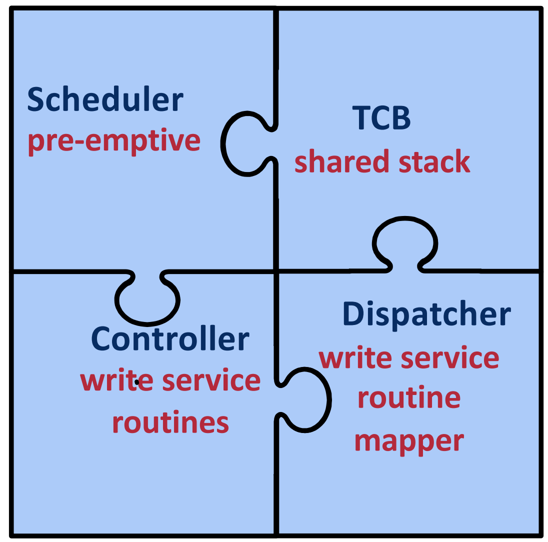


Assembling an Operating System

1. Select the modules needed.



2. Configure the modules.



3. Write startup code.

```
initUART1();
initUserStack();
initEvents();

dispatcher.dispatch();
```

Where is the Synthesis?

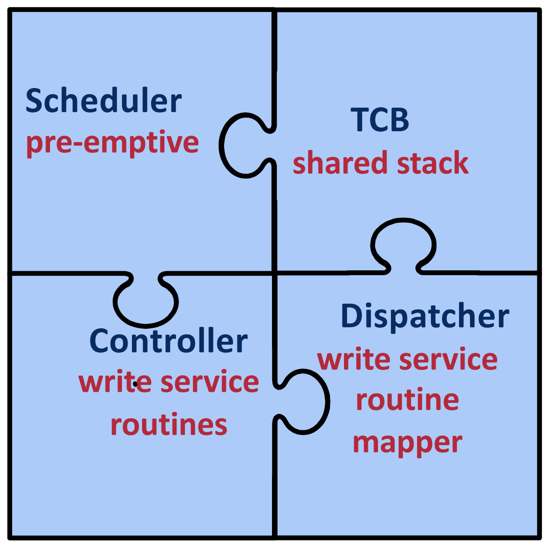
Building Blocks

- FIFO Queue
- Prioritized Single Queue
- Task Creation
- Task Termination
- ……

Where is the Synthesis?

Building Blocks

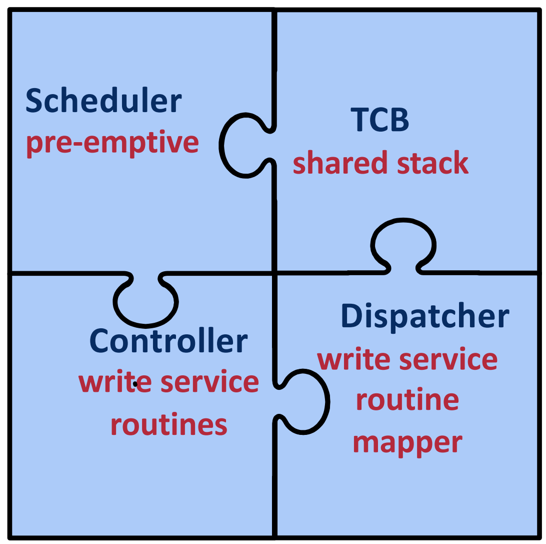
- FIFO Queue
- Prioritized Single Queue
- Task Creation
- Task Termination
- ......



Where is the Synthesis?

Building Blocks

- FIFO Queue
- Prioritized Single Queue
- Task Creation
- Task Termination
- ……




```
initUART1();
initUserStack();
initEvents();

dispatcher.dispatch();
```

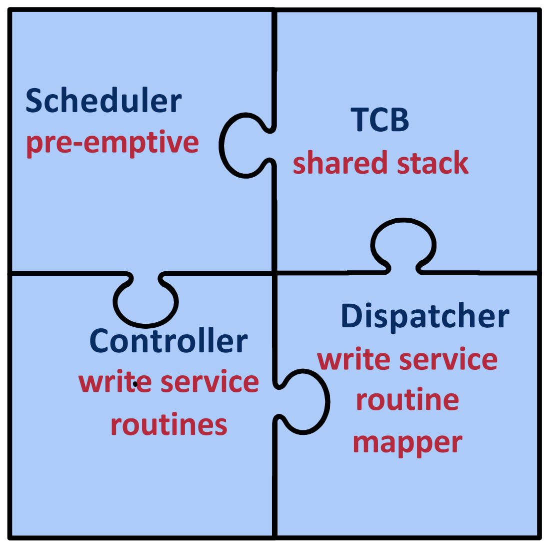
Where is the Synthesis?

Building Blocks

- FIFO Queue
- Prioritized Single Queue
- Task Creation
- Task Termination
- ……

This seems in the realm of current synthesis techniques … what tools are appropriate?

I think we need another DSL (specification language) for this too.




```
initUART1();
initUserStack();
initEvents();

dispatcher.dispatch();
```

We need a DSL for this, I think …

Why are we doing this?

Software Rules the World

And it Fails!

We need a way to build simple and correct software (from which one can build complex systems).

Thank You!

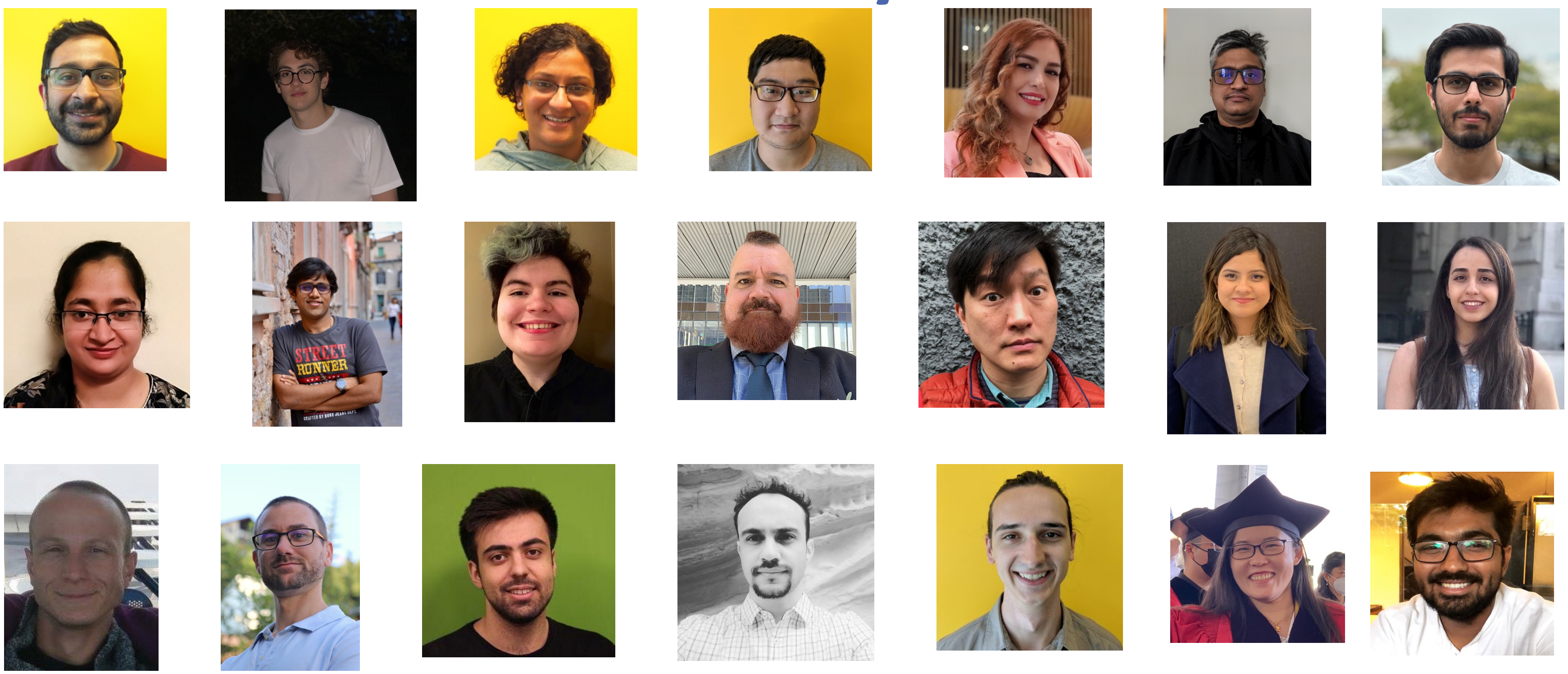
NSERC
CRSNG

DARPA

HUAWEI

ARM

Thanks to My Team



... and many, many undergraduates!