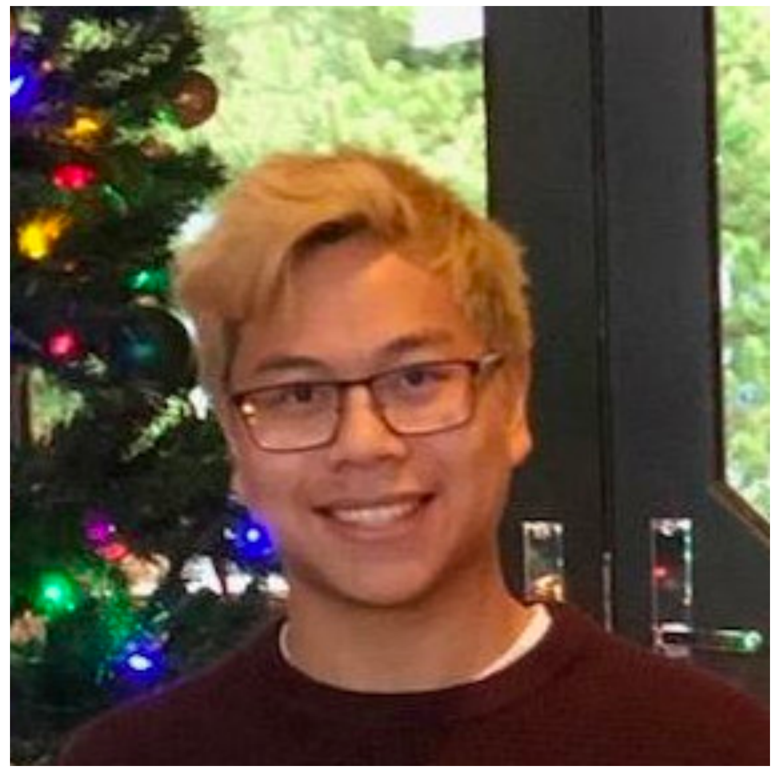
Generate Compilers from Hardware Models!

Gus Smith, PhD Candidate, University of Washington
PLARCH 2023

PAUL G.
ALLEN
SCHOOL

Ben Kushigian

Vishal Canumalla



Andrew Cheung

René Just

Zachary Tatlock

The Hardware Lottery

Sara Hooker

Google Research, Brain Team

shooker@google.com

2020

The Hardware Lottery

Sara Hooker

Google Research, Brain Team

shooker@google.com

2020

Hardware and compilers have a disproportionate role in deciding which research ideas succeed or fail.

The Hardware Lottery

Sara Hooker

Google Research, Brain Team

shooker@google.com

2020

Hardware and compilers have a disproportionate role in deciding which research ideas succeed or fail.

Takeaway for our community: new platforms (hardware + compiler stack) should be easier to build, so that the best ideas win!

The Hardware Lottery

Sara Hooker

Google Research, Brain Team

shooker@google.com

2020

Hardware and compilers have a disproportionate role in deciding which research ideas succeed or fail.

Takeaway for our community: new platforms (hardware + compiler stack) should be easier to build, so that the best ideas win!

The Hardware Lottery

Sara Hooker

Google Research, Brain Team

shooker@google.com

2020

Hardware and compilers have a disproportionate role in deciding which research ideas succeed or fail.

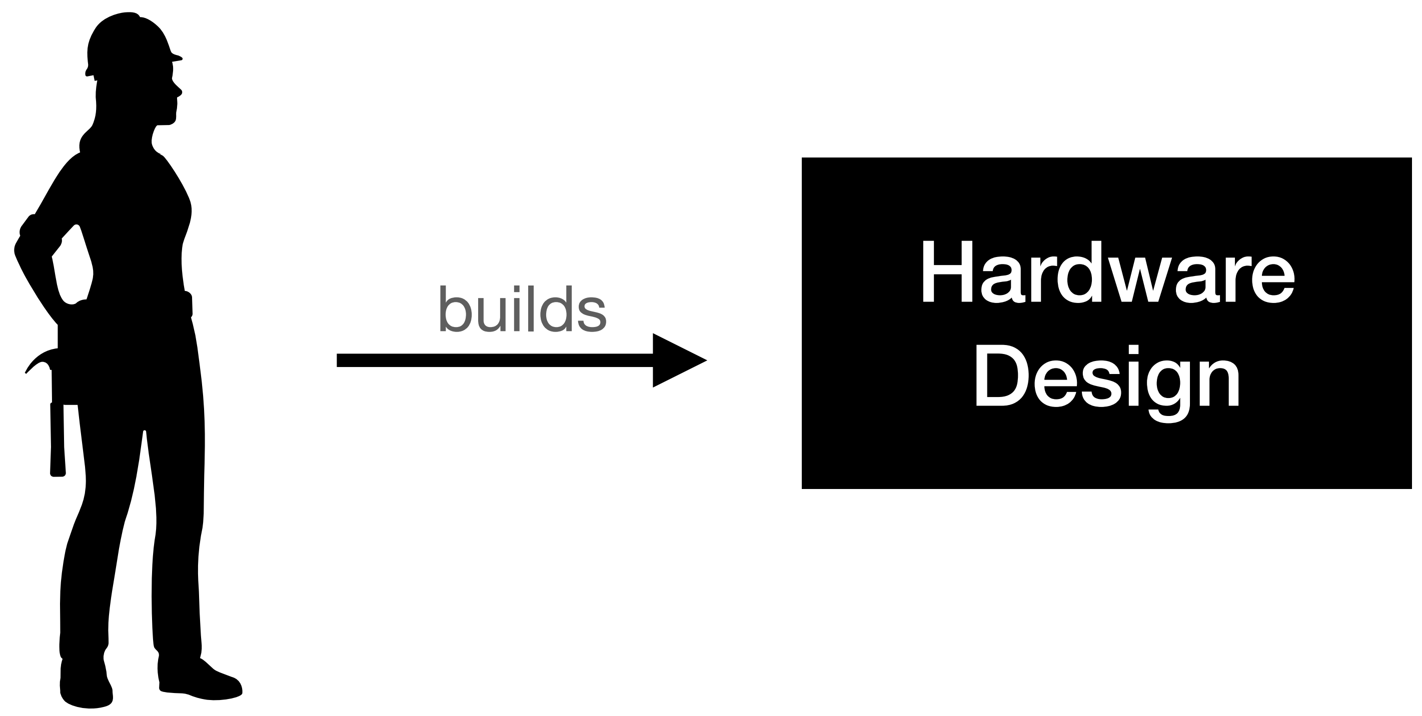
Takeaway for our community: new platforms (hardware + compiler stack) should be easier to build, so that the best ideas win!

Why are compilers hard to build?

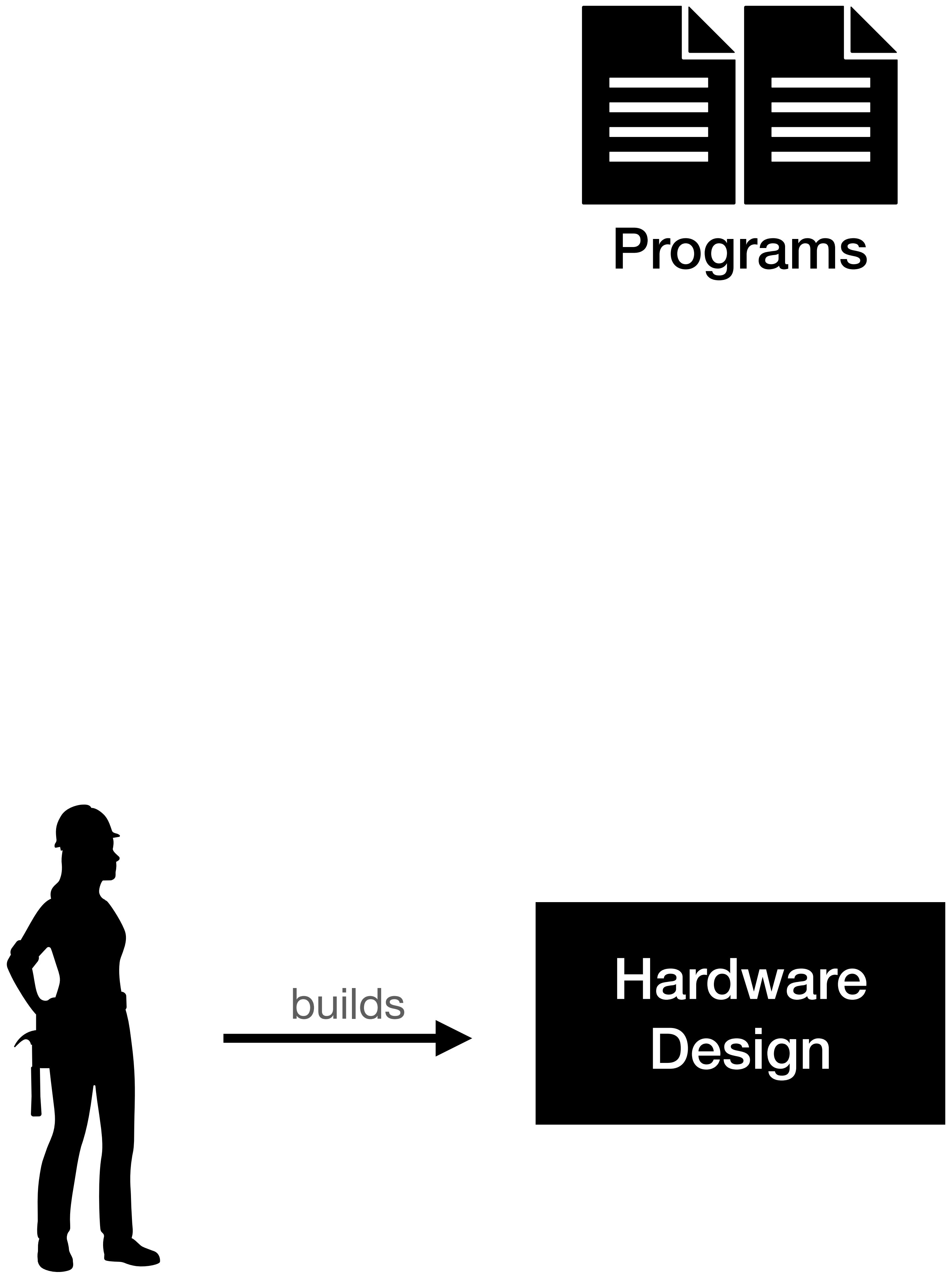
Why are compilers hard to build?

Why are compilers hard to build?

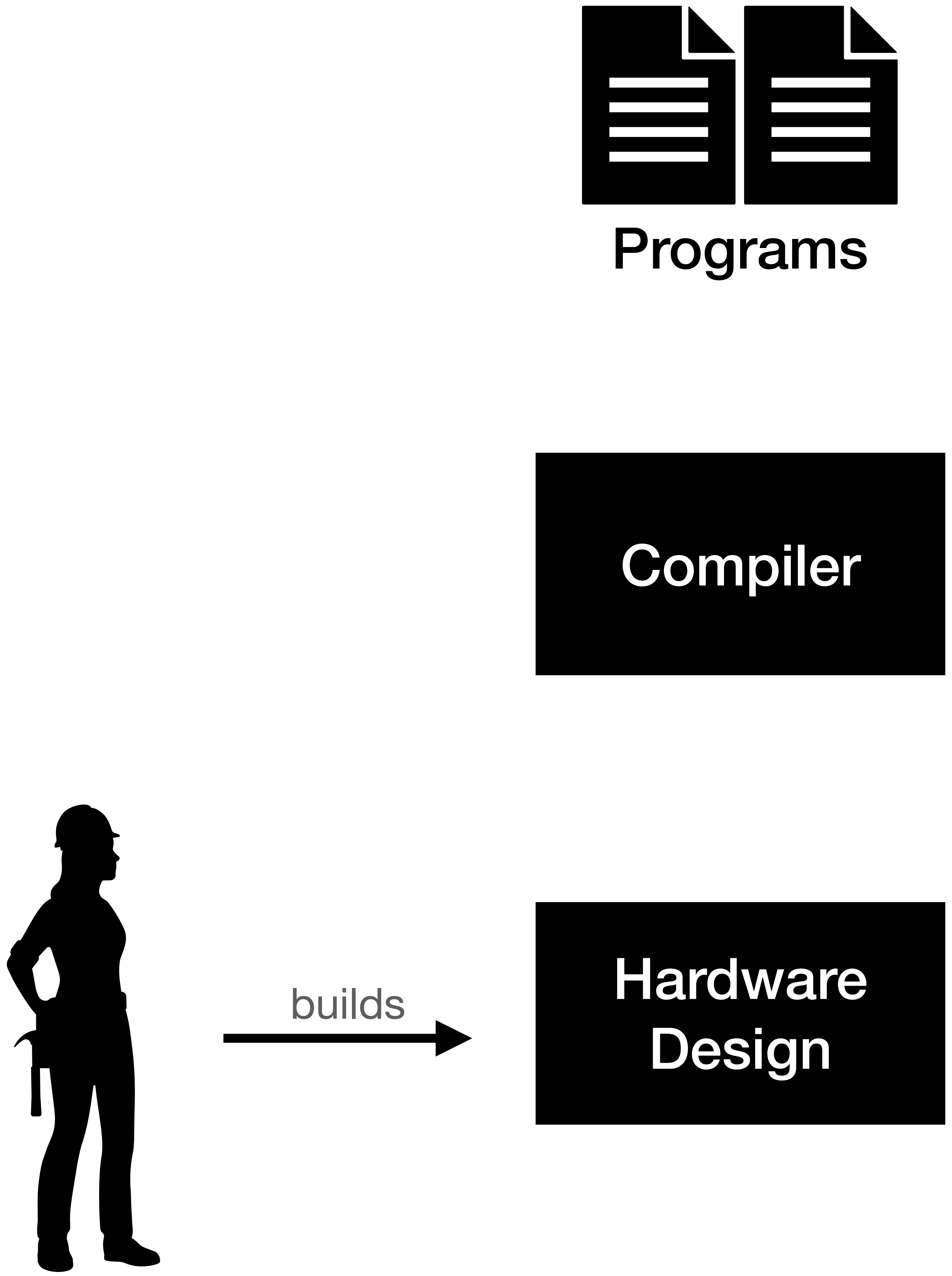
Why are compilers hard to build?



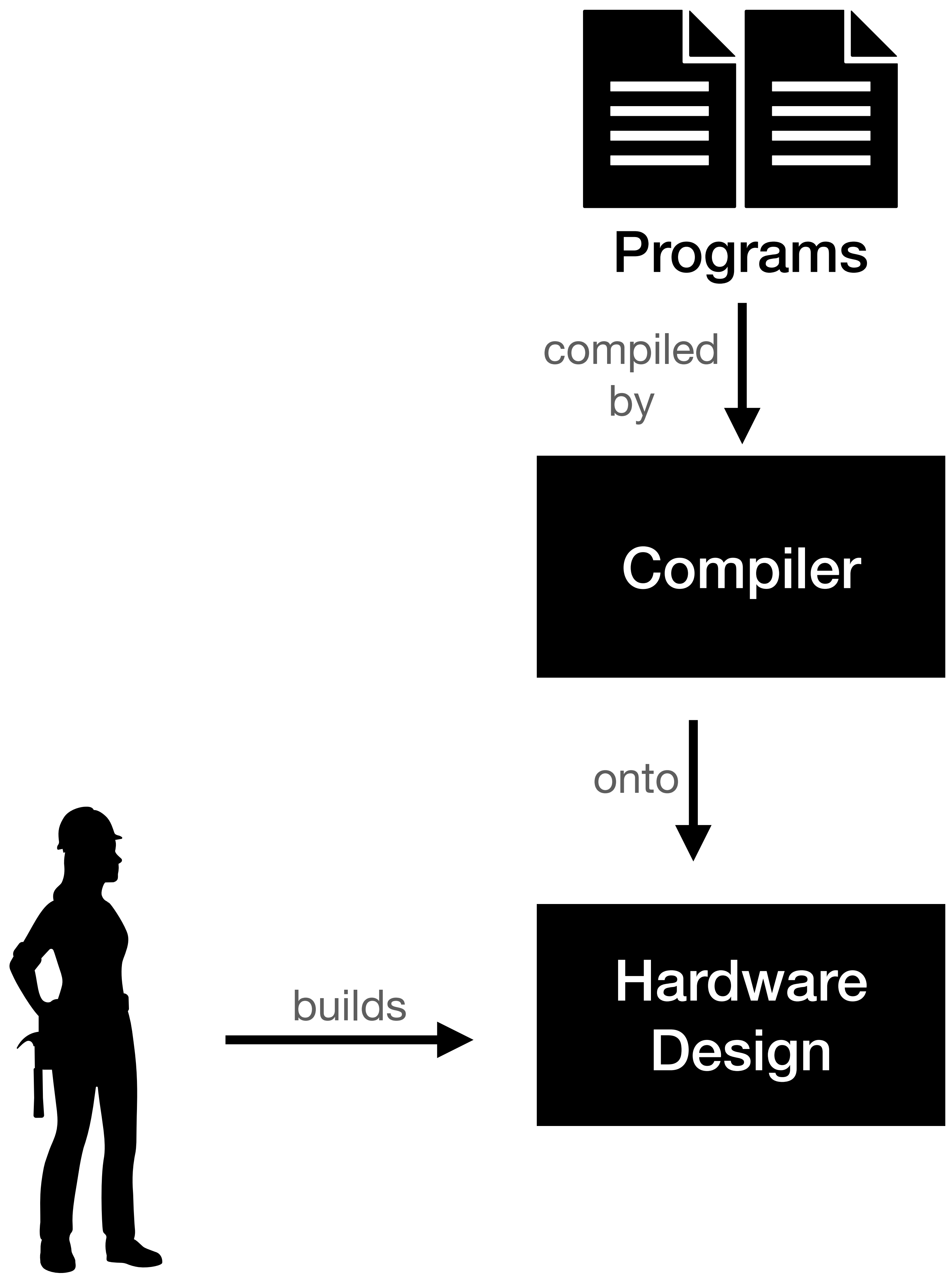
Why are compilers hard to build?



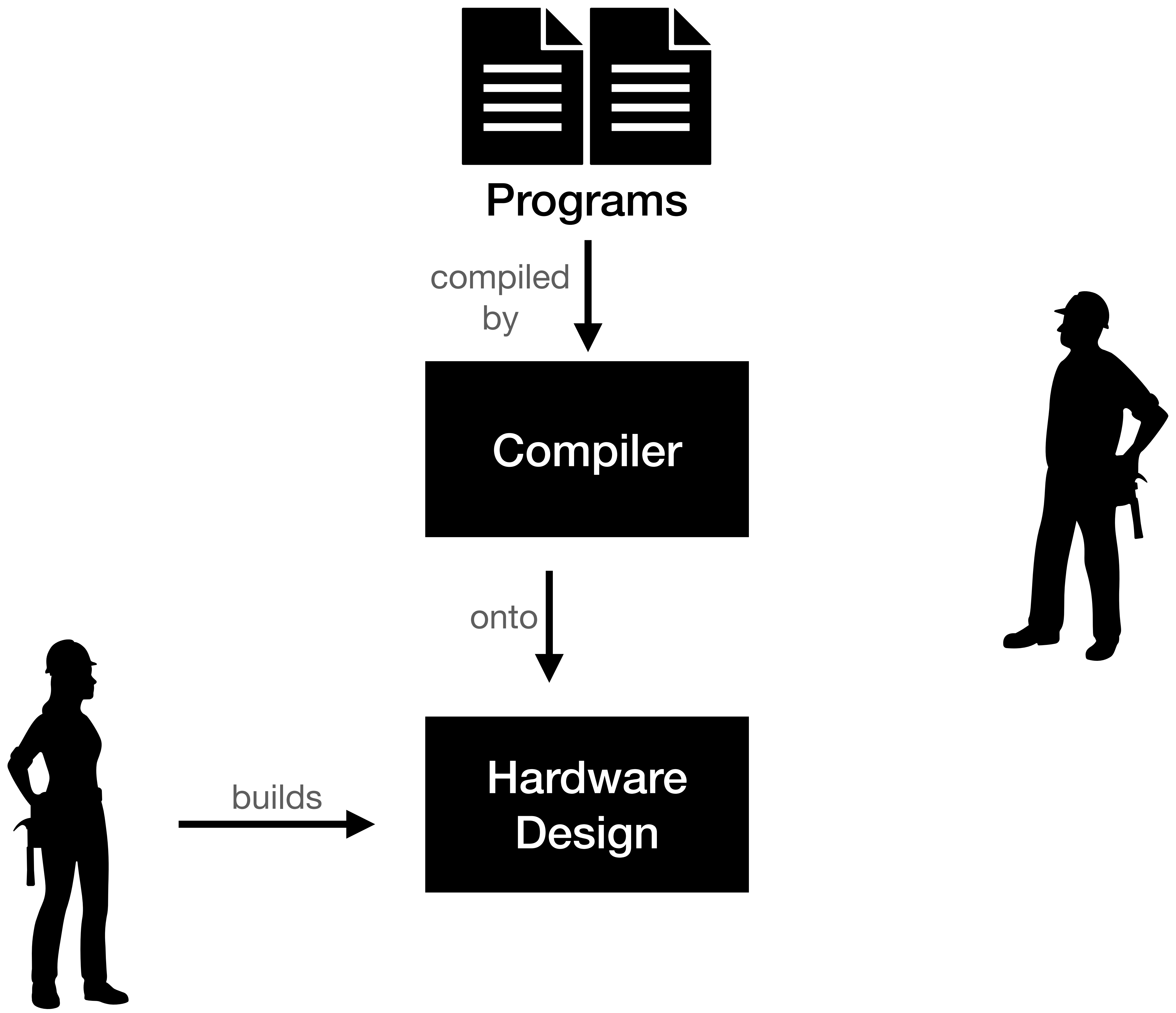
Why are compilers hard to build?



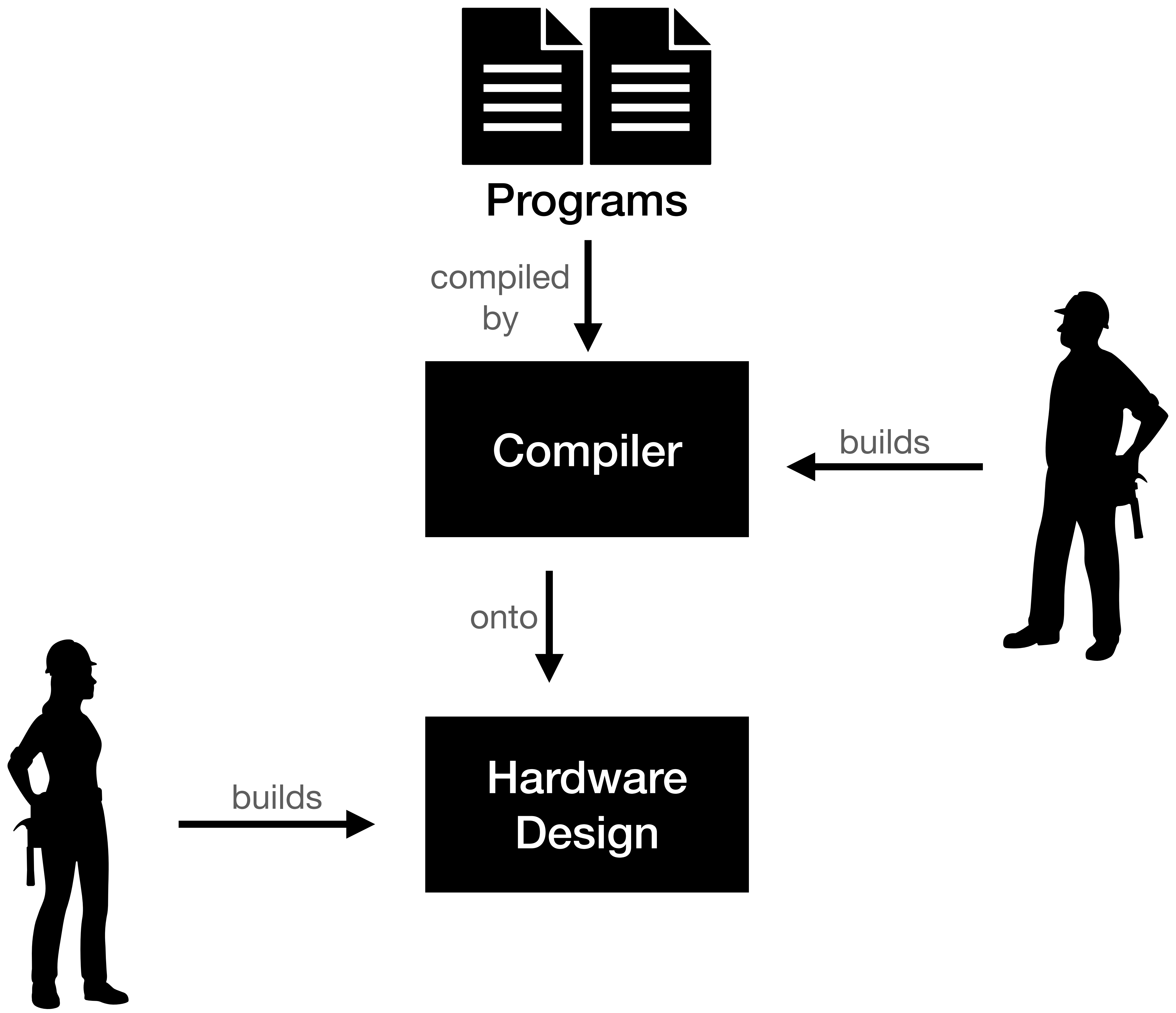
Why are compilers hard to build?

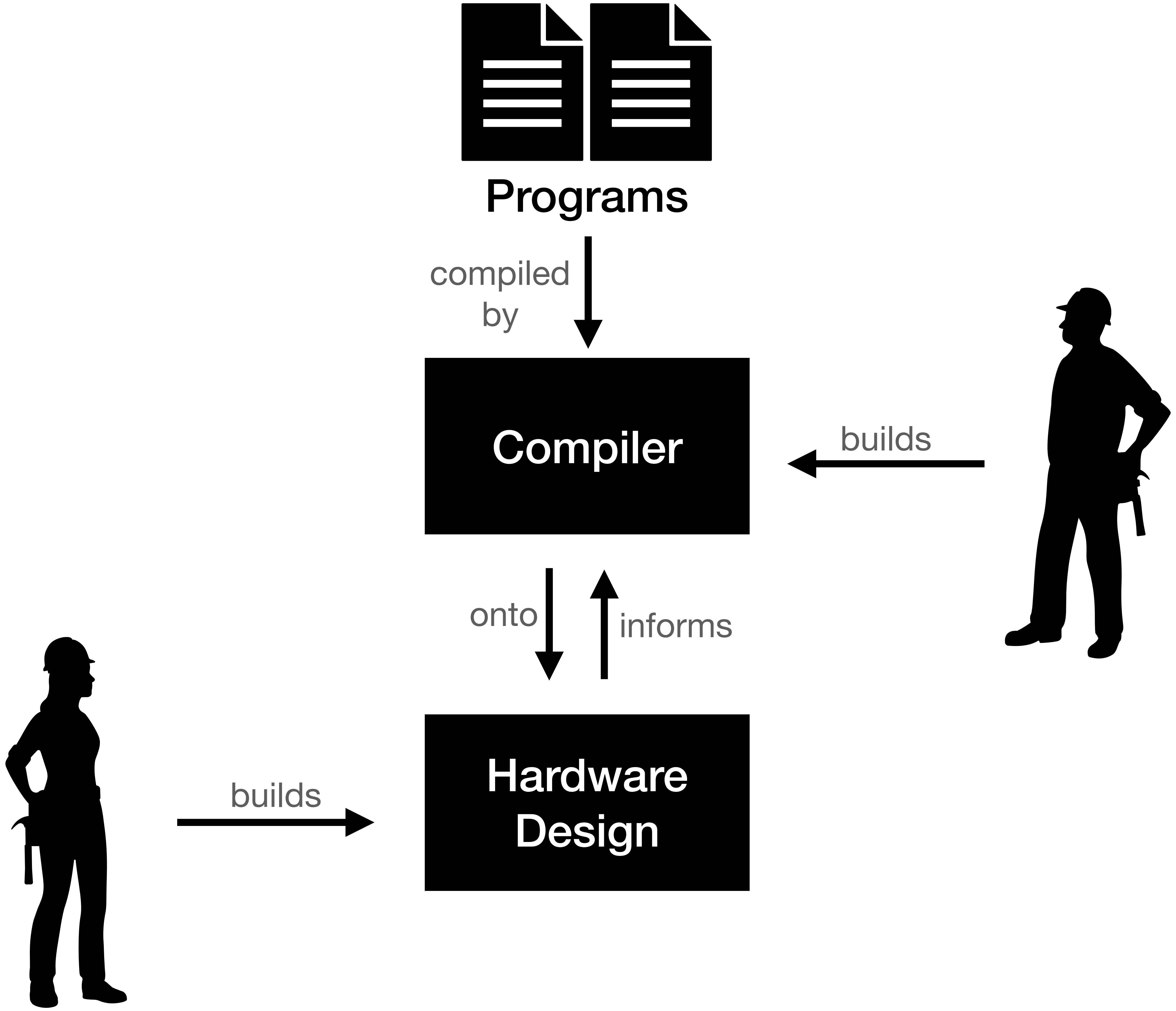


Why are compilers hard to build?

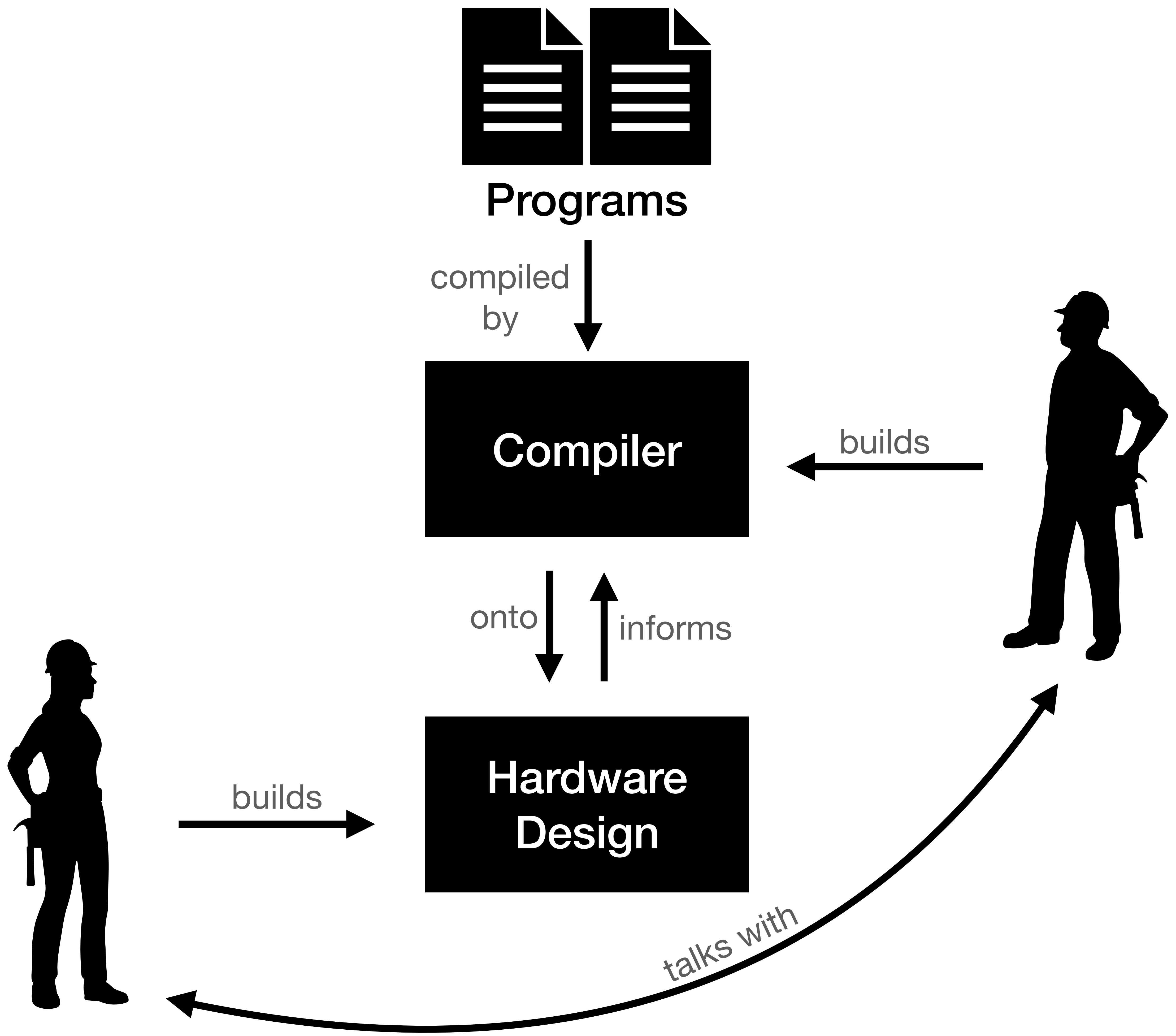


Why are compilers hard to build?

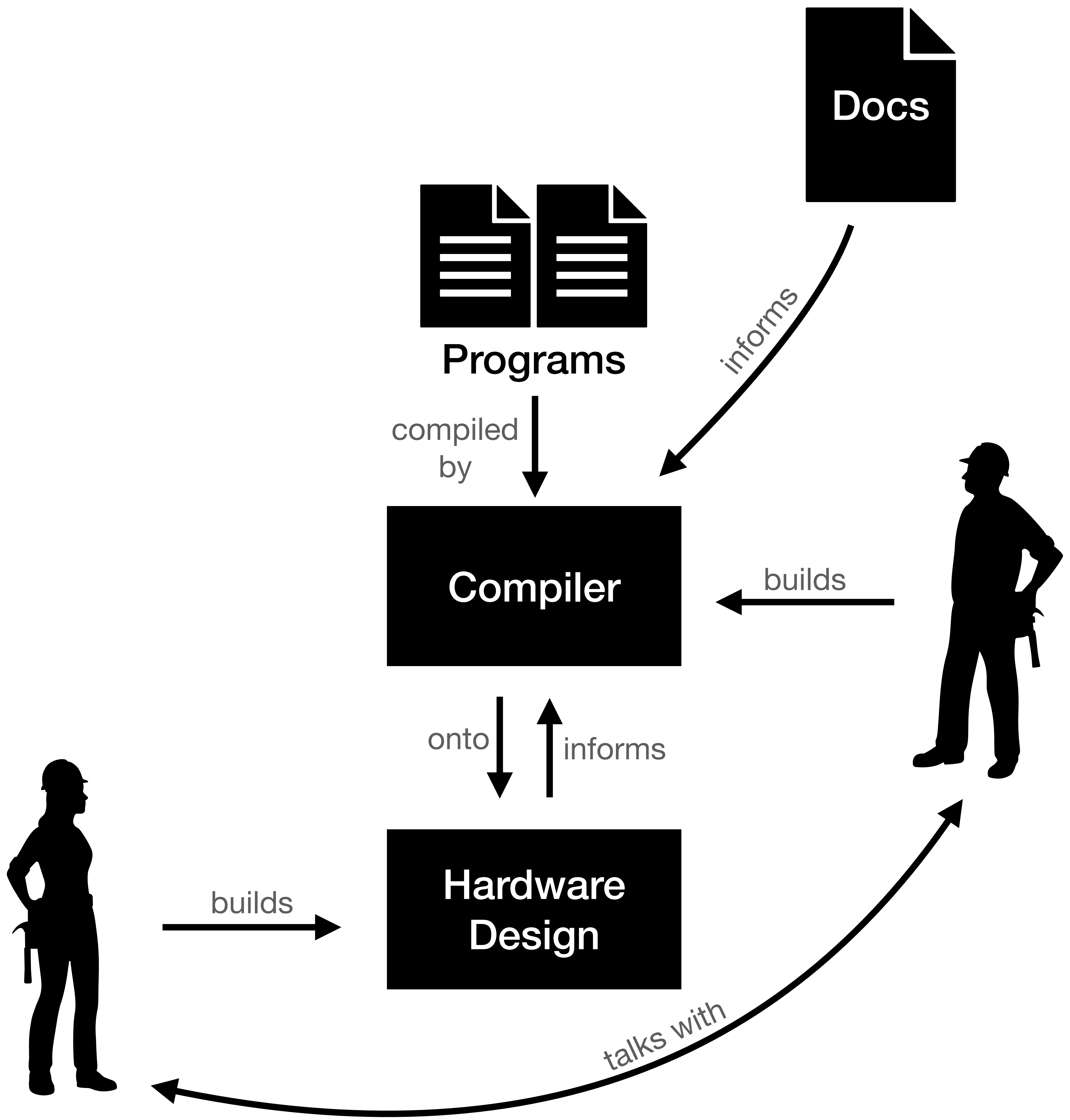



Programs
compiled by
Compiler
builds
onto
informs
Hardware Design
builds

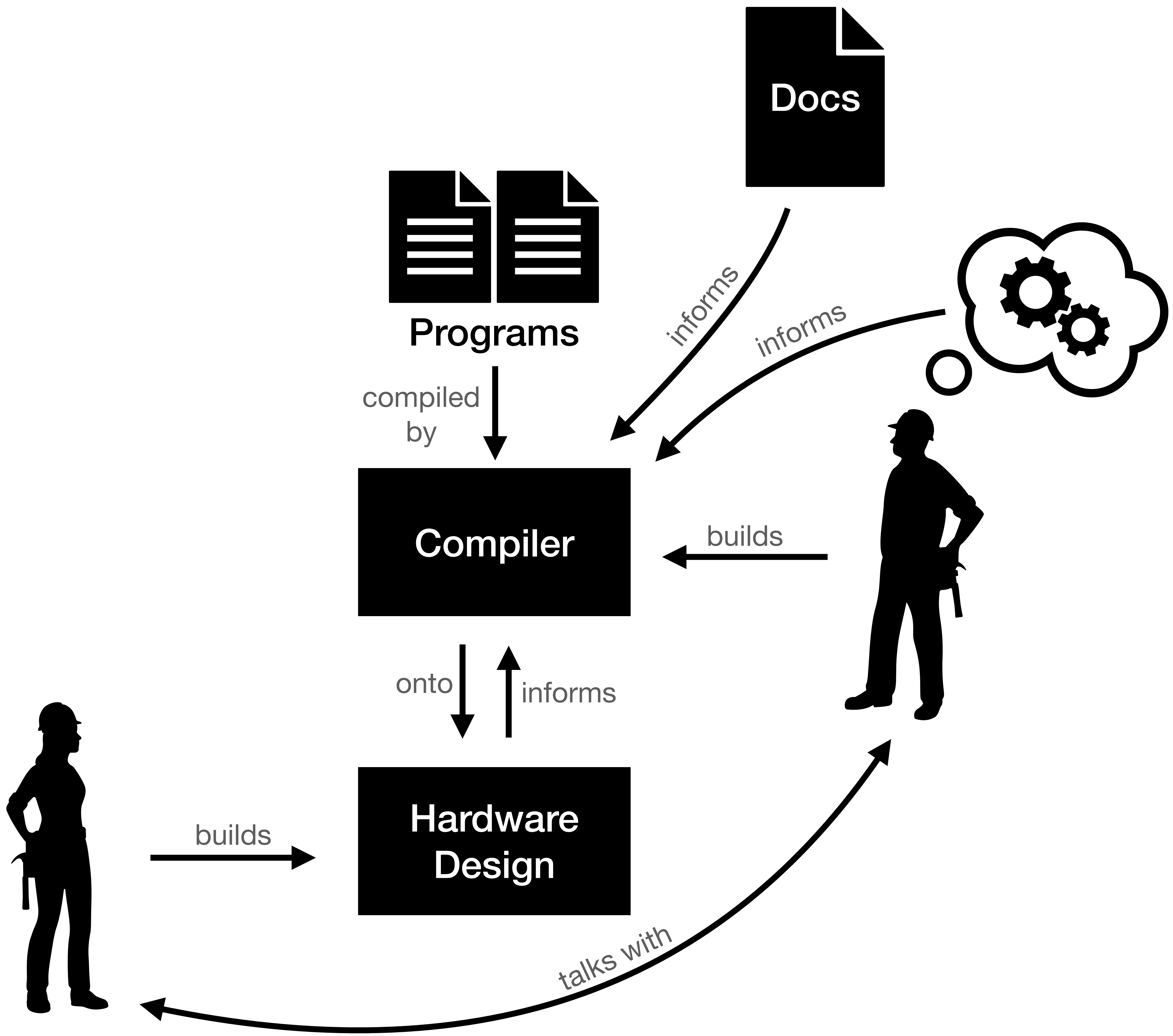
Why are compilers hard to build?



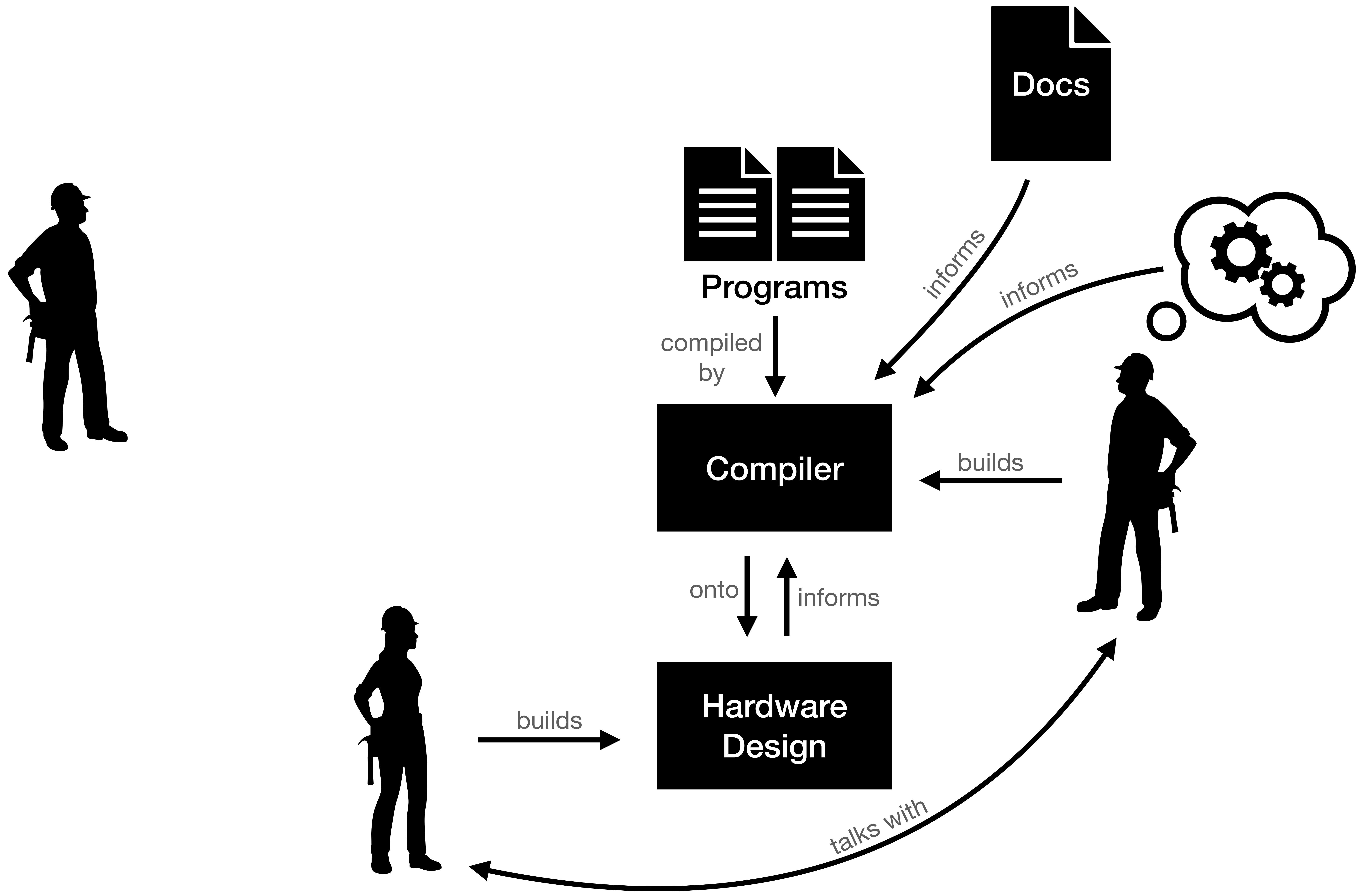
Why are compilers hard to build?



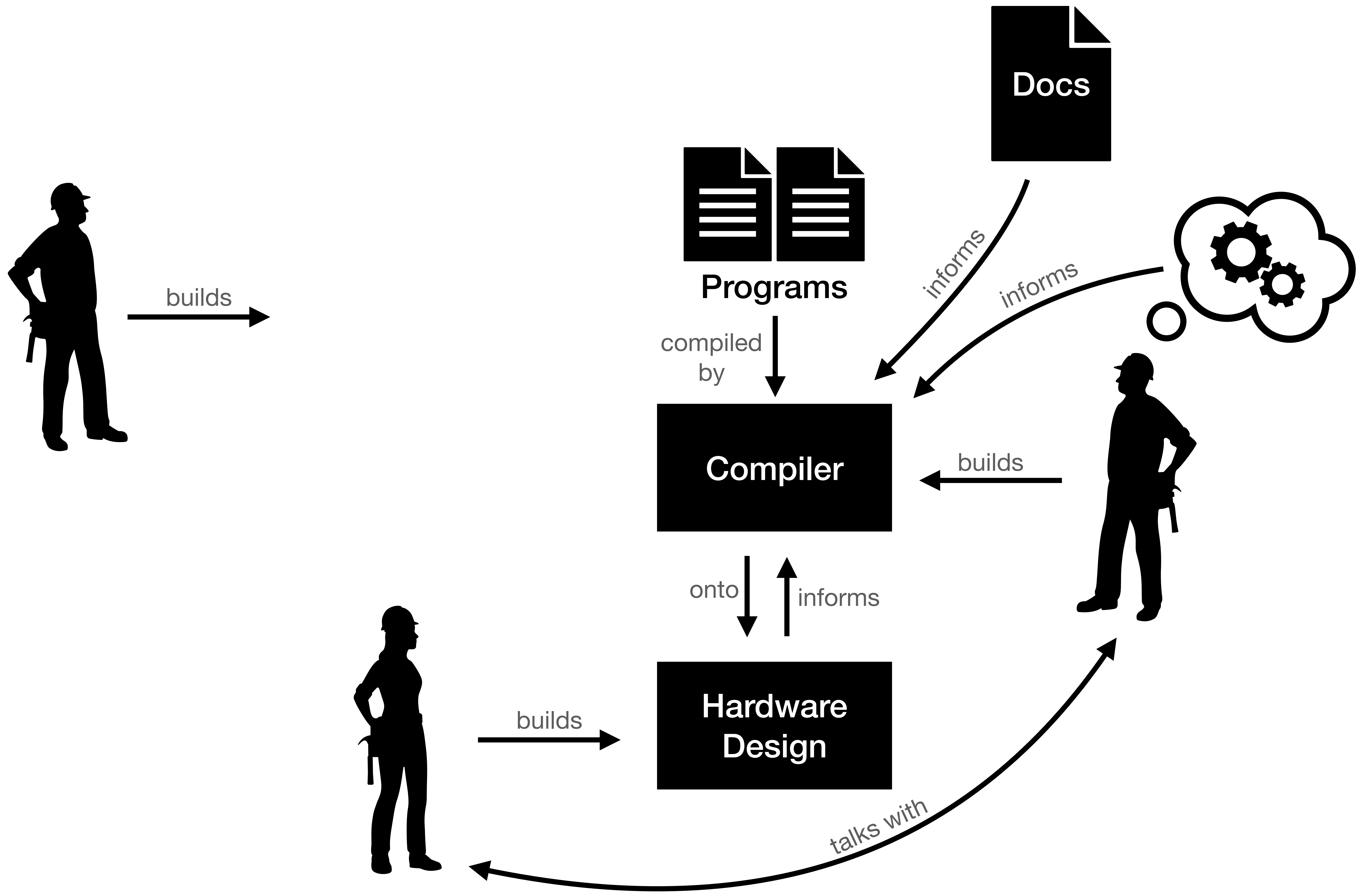
Why are compilers hard to build?



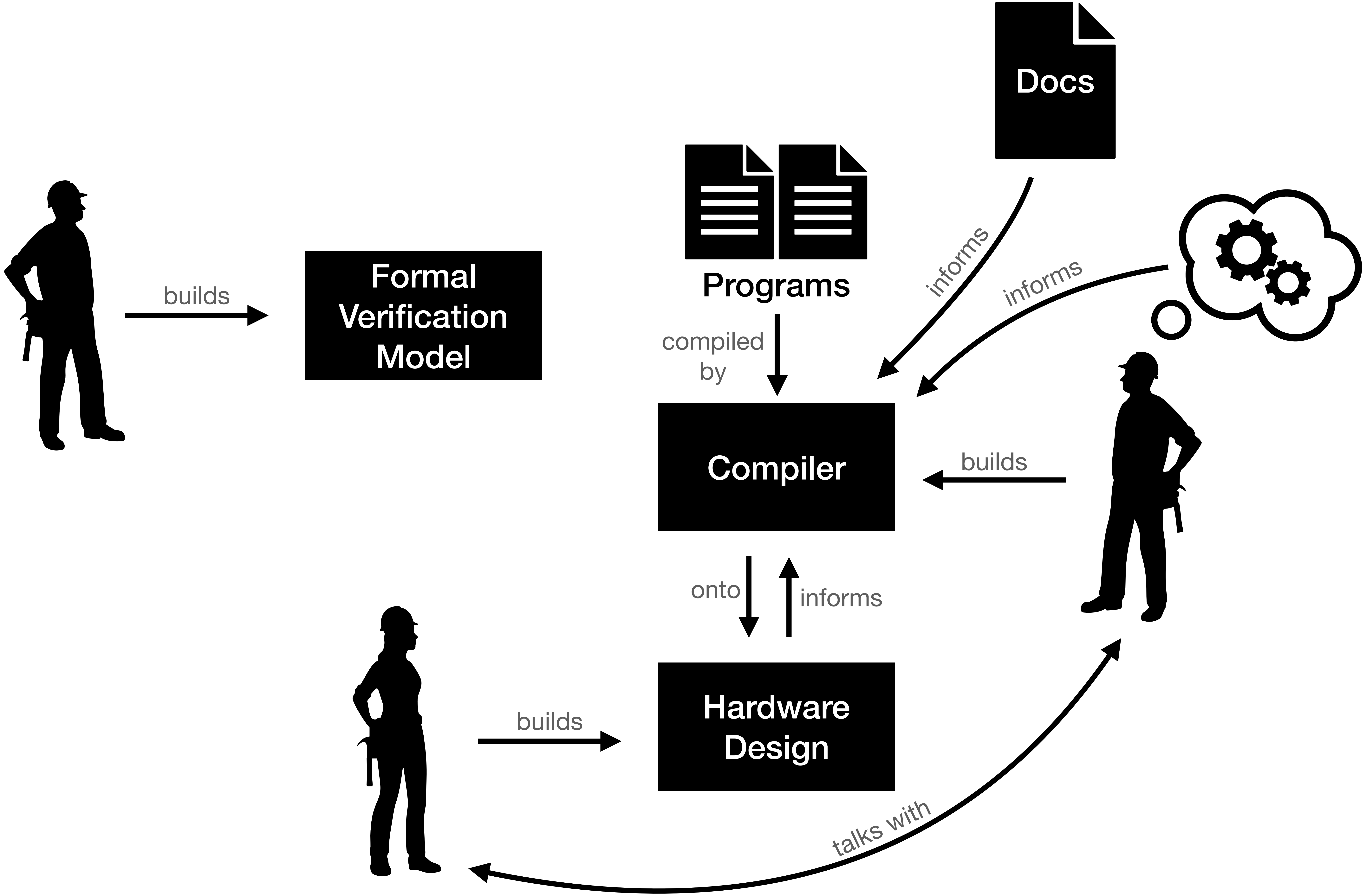
Why are compilers hard to build?



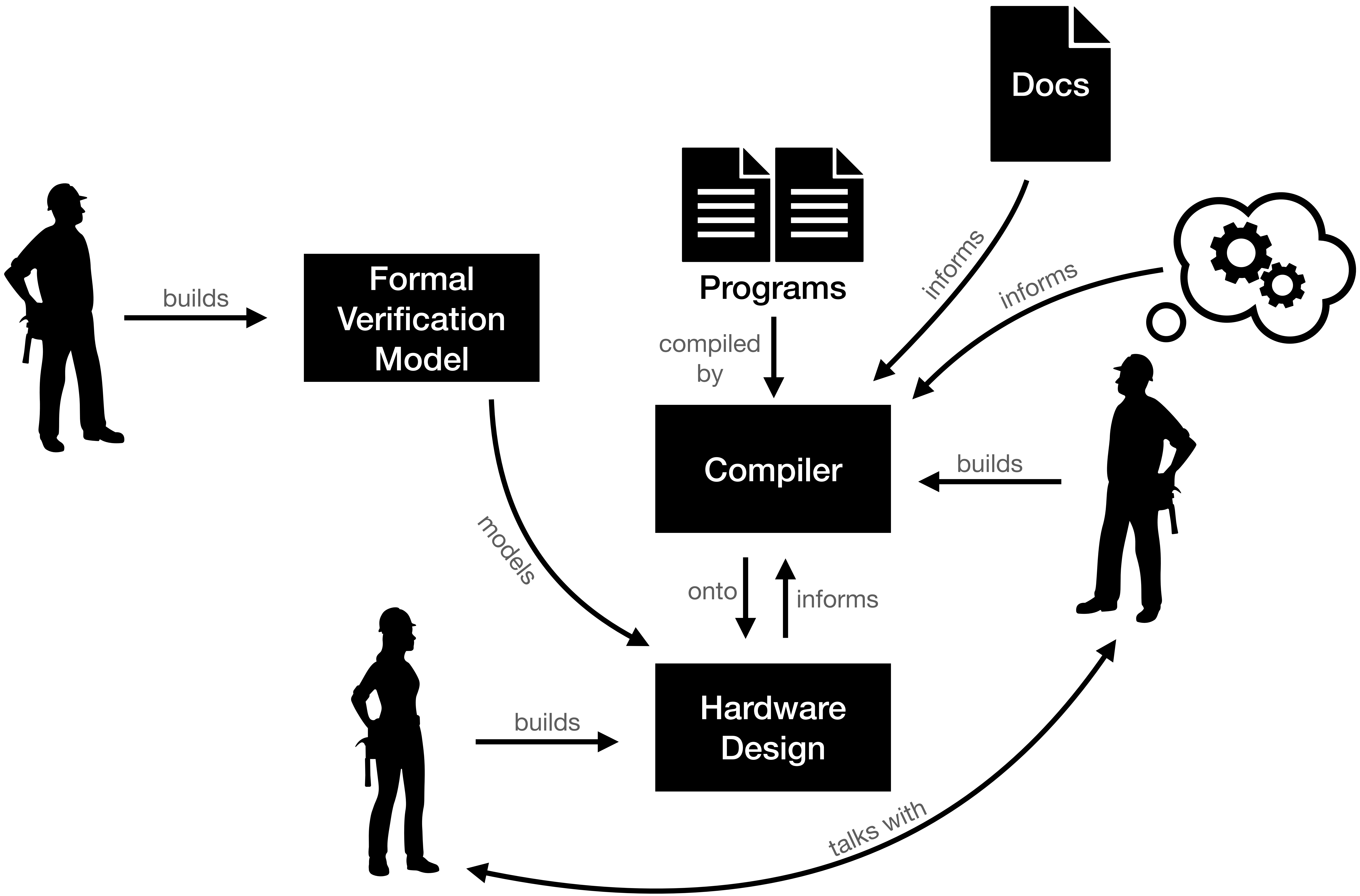
Why are compilers hard to build?



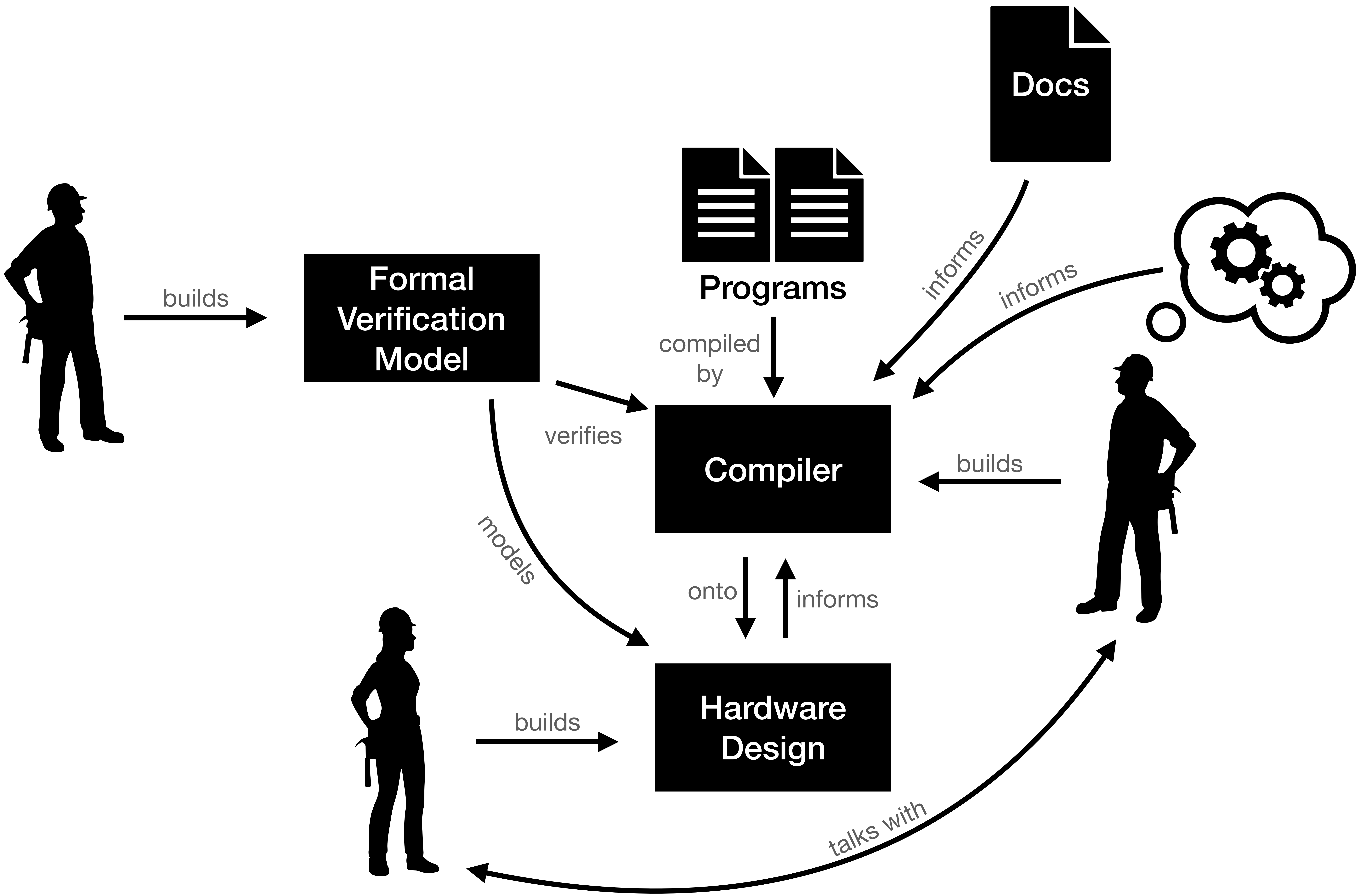
Why are compilers hard to build?



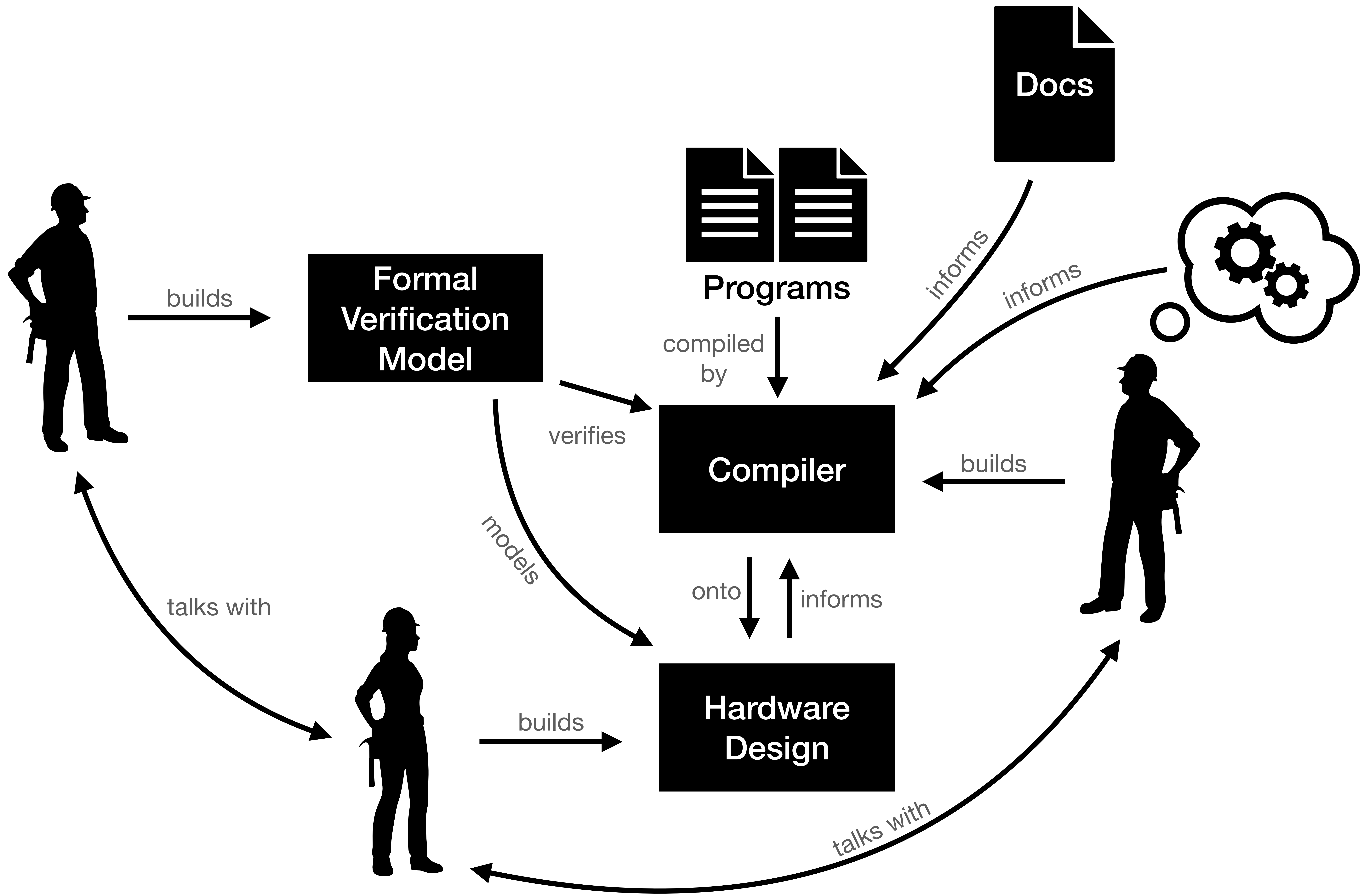
Why are compilers hard to build?



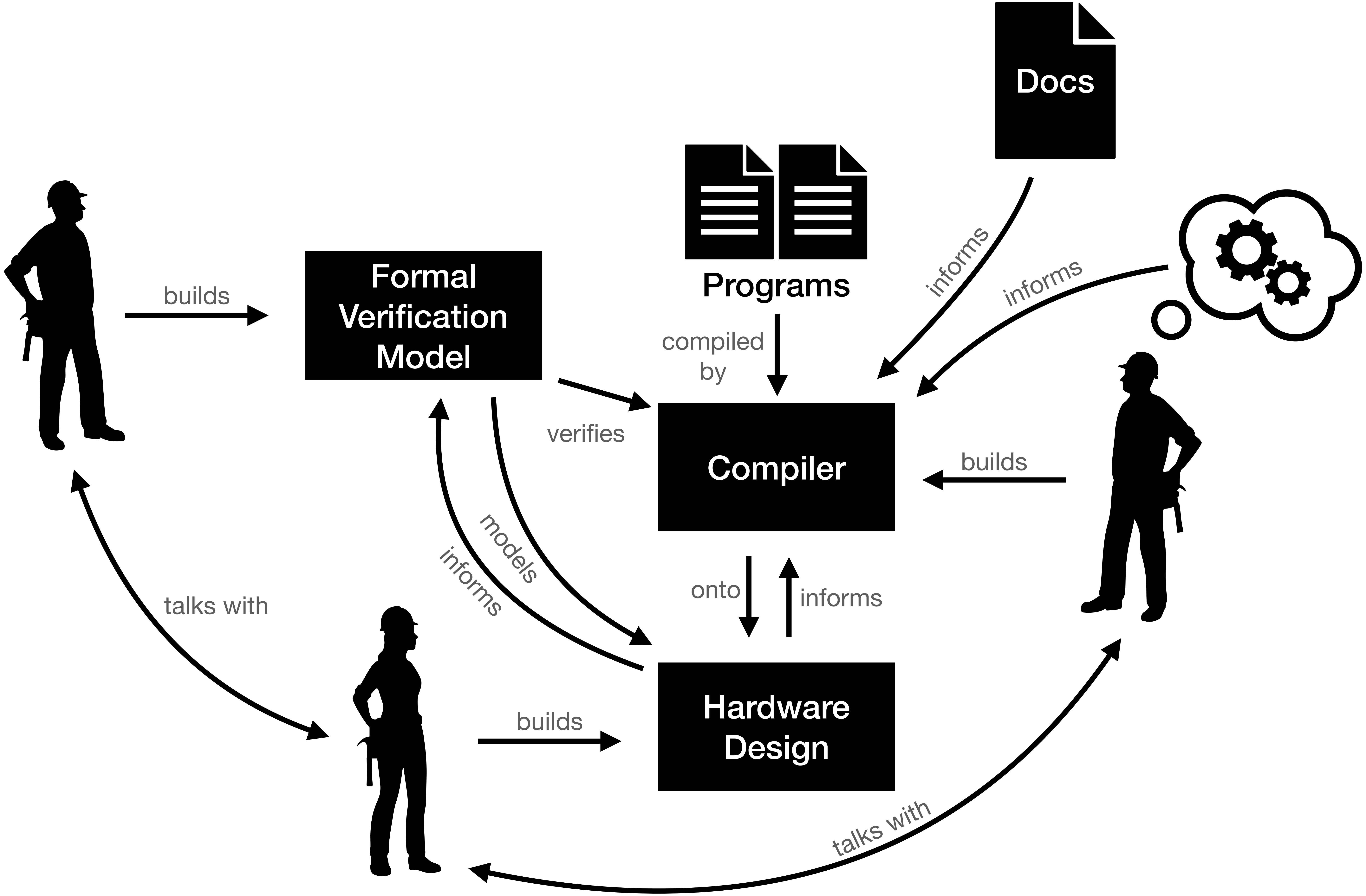
Why are compilers hard to build?



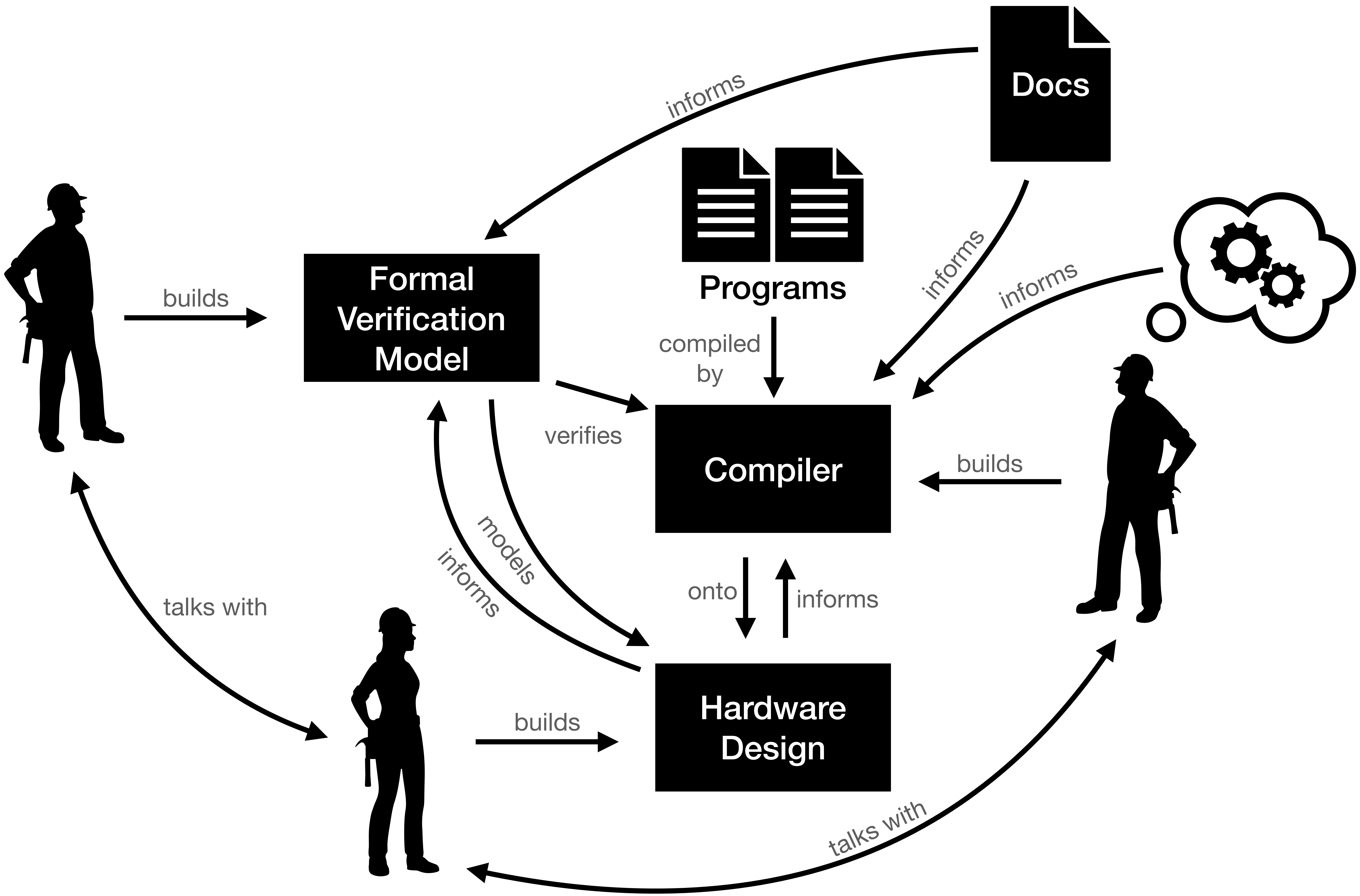
Why are compilers hard to build?

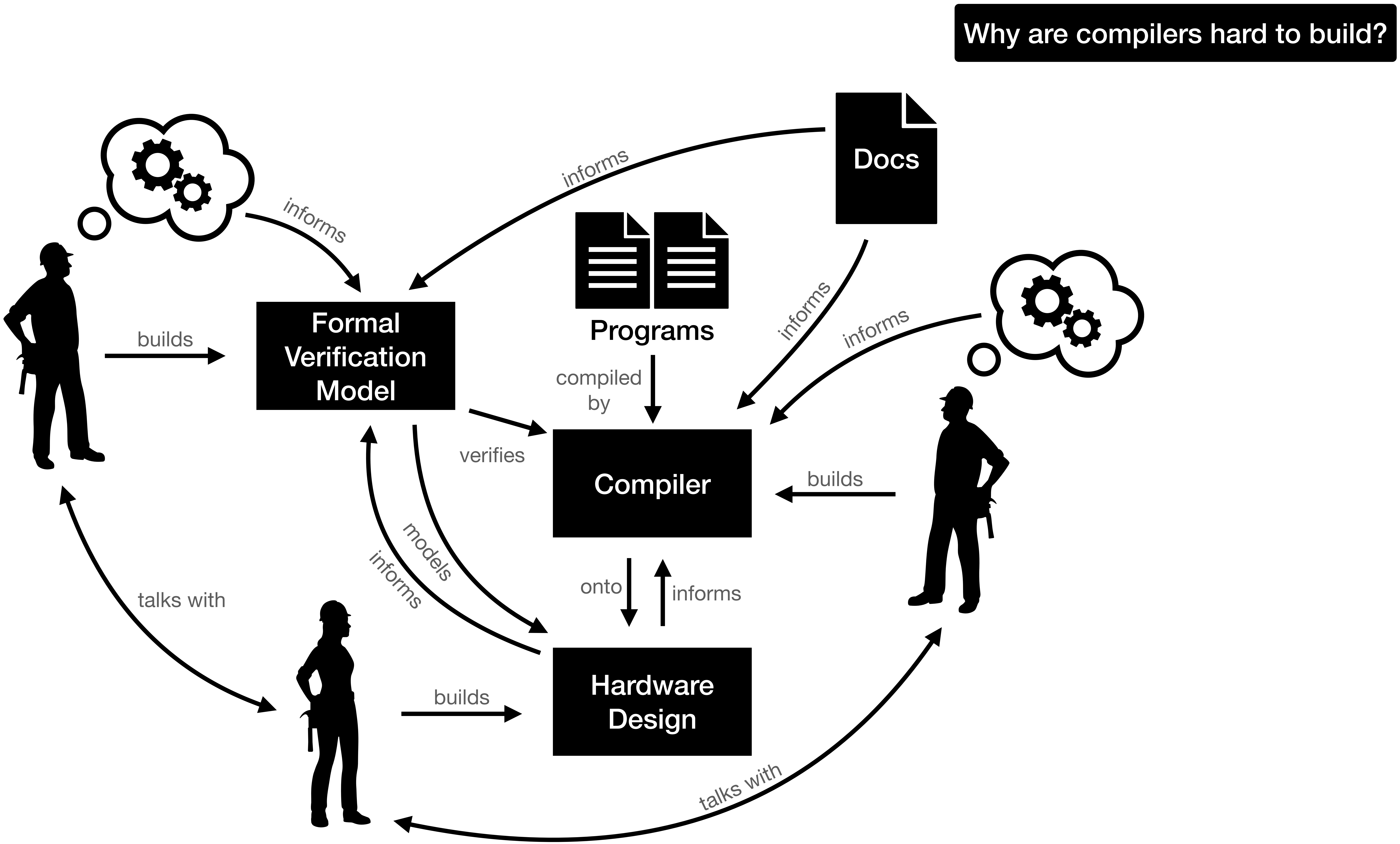


Why are compilers hard to build?

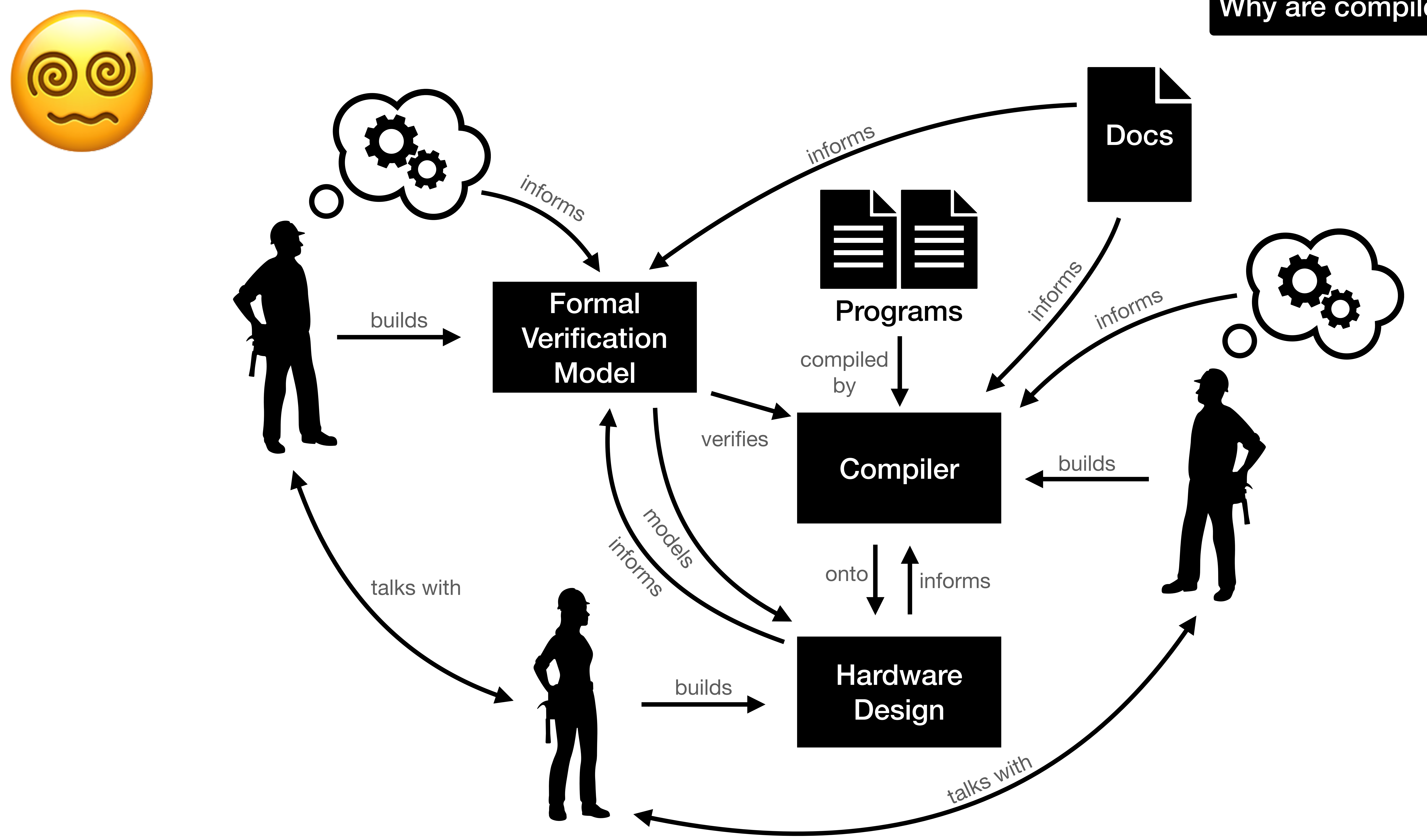


Why are compilers hard to build?

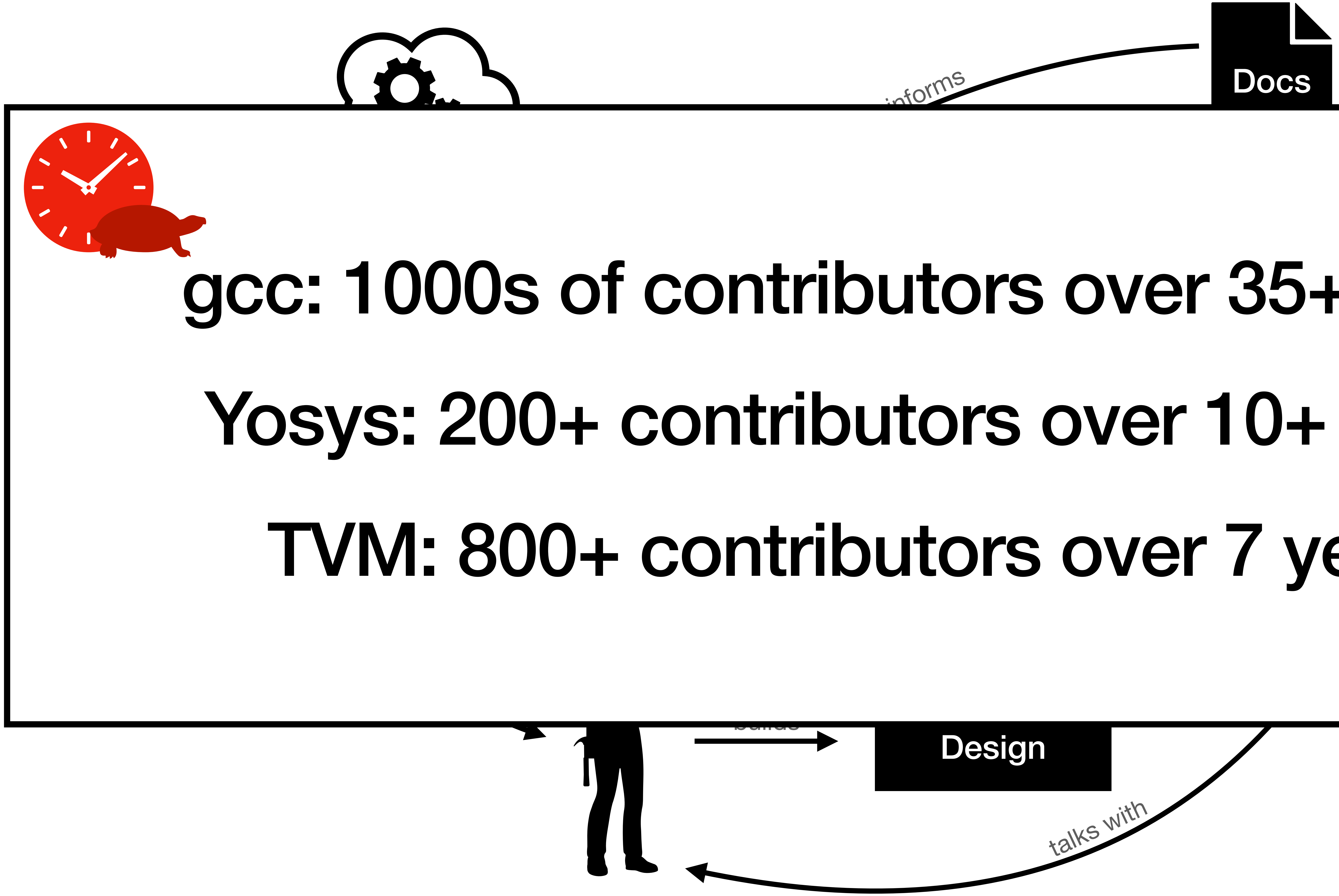



Why are compilers hard to build?
Docs
informs
Programs
informs
Formal Verification Model
builds
informs
compiled by
informs
verifies
Compiler
builds
models
informs
onto
informs
talks with
Hardware Design
builds
talks with

Why are compilers hard to build?



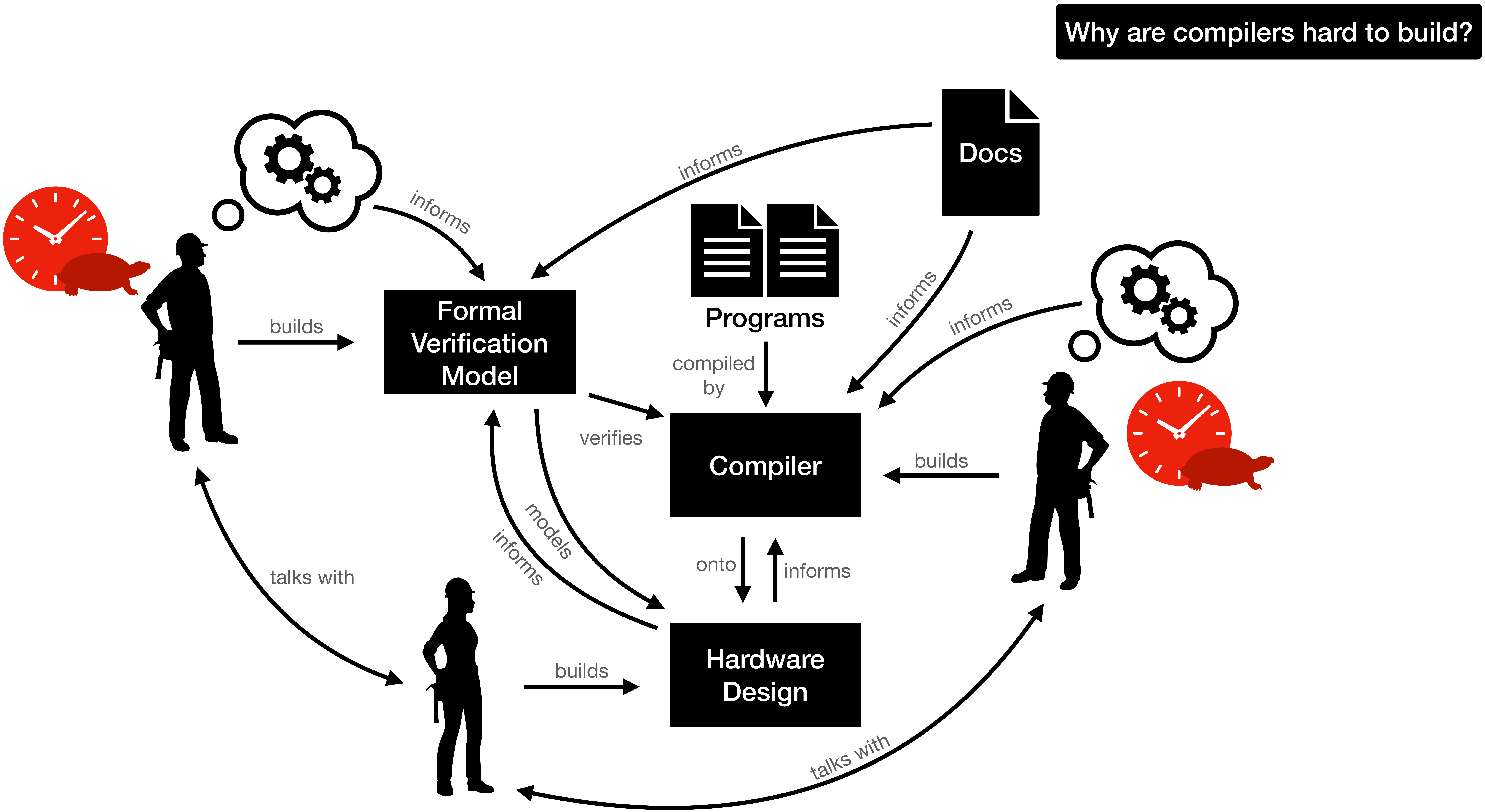

Why are compilers hard to build?
informs
Docs
informs
Programs
Formal Verification Model
builds
compiled by
informs
informs
verifies
Compiler
builds
models
informs
onto
informs
talks with
builds
Hardware Design
talks with



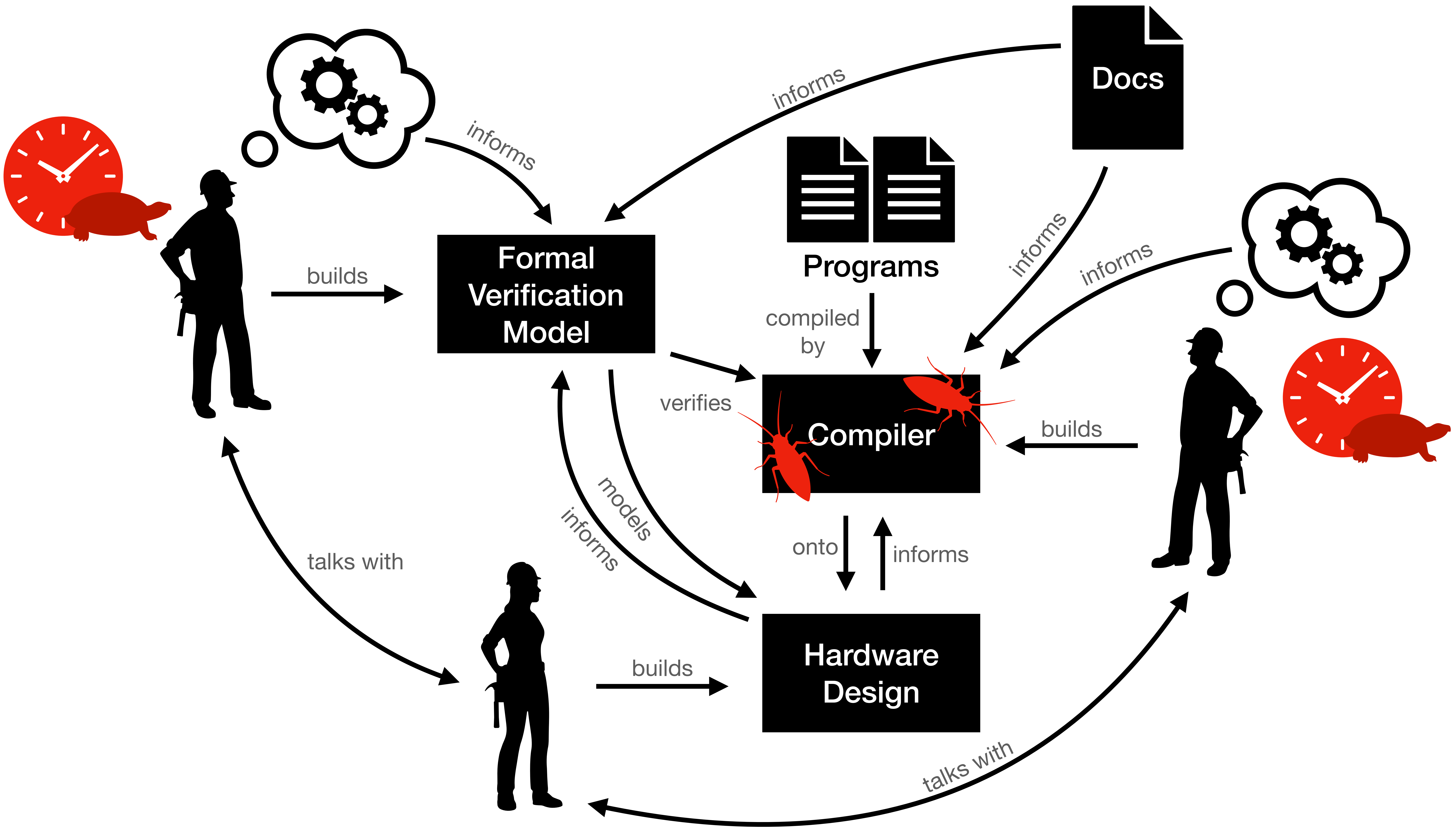
gcc: 1000s of contributors over 35+ years

Yosys: 200+ contributors over 10+ years

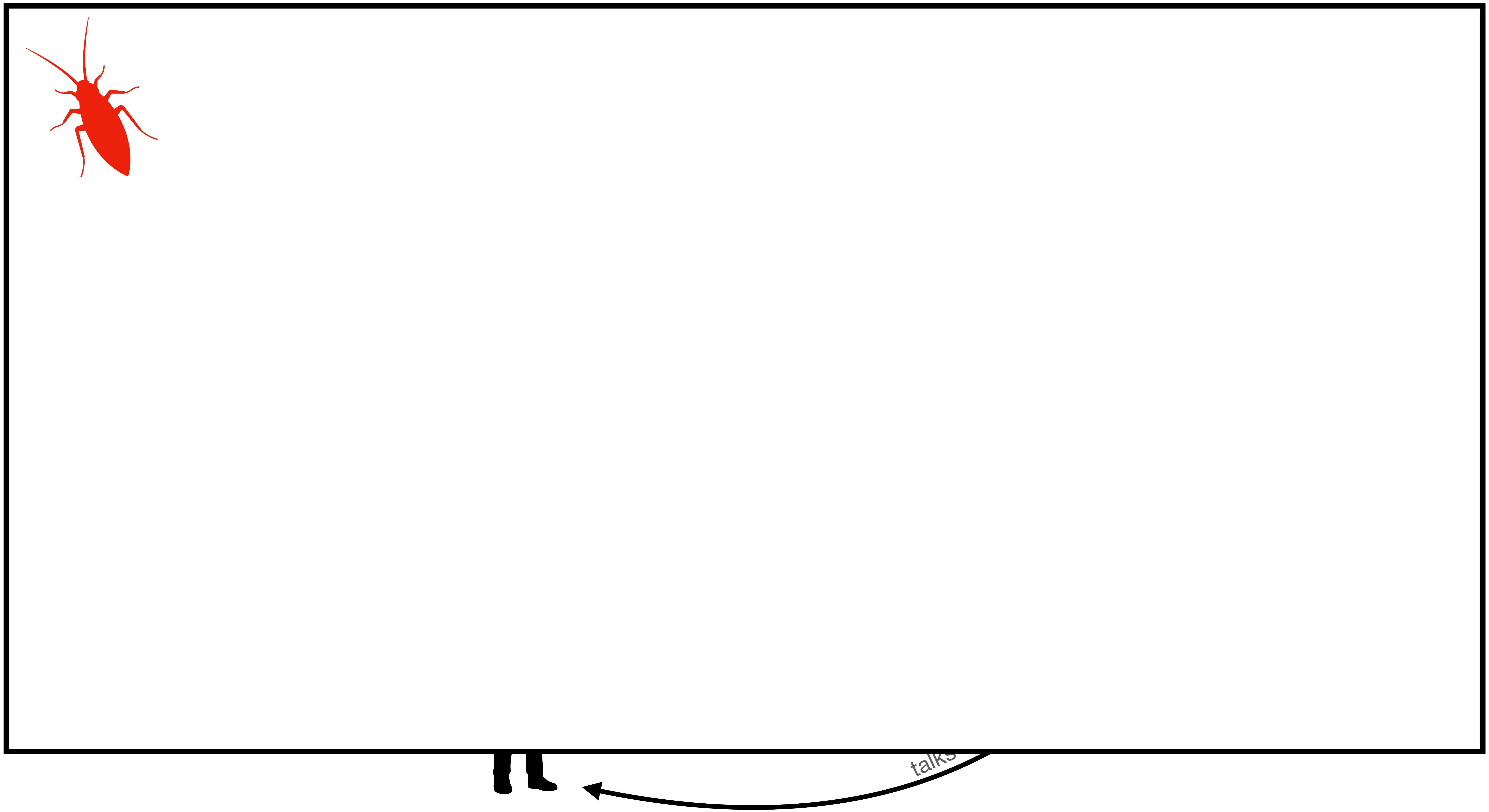
TVM: 800+ contributors over 7 years


Why are compilers hard to build?
Docs
informs
Programs
informs
informs
informs
builds
Formal Verification Model
compiled by
verifies
Compiler
builds
models
informs
onto
informs
talks with
builds
Hardware Design
talks with

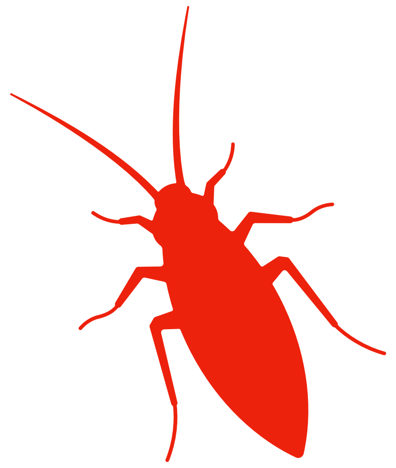
Why are compilers hard to build?



Why are compilers hard to build?



Why are compilers hard to build?



Finding and Understanding Bugs in C Compilers

Xuejun Yang Yang Chen Eric Eide John Regehr
University of Utah, School of Computing
{jxyang, chenyang, eeide, regehr}@cs.utah.edu

(Csmith)

Finds bugs in gcc

Finding and Understanding Bugs in FPGA Synthesis Tools

Yann Herklotz
yann.herklotz15@imperial.ac.uk
Imperial College London
London, UK

John Wickerson
j.wickerson@imperial.ac.uk
Imperial College London
London, UK

(Verismith)

Finds bugs in Yosys

A Comprehensive Study of Deep Learning Compiler Bugs

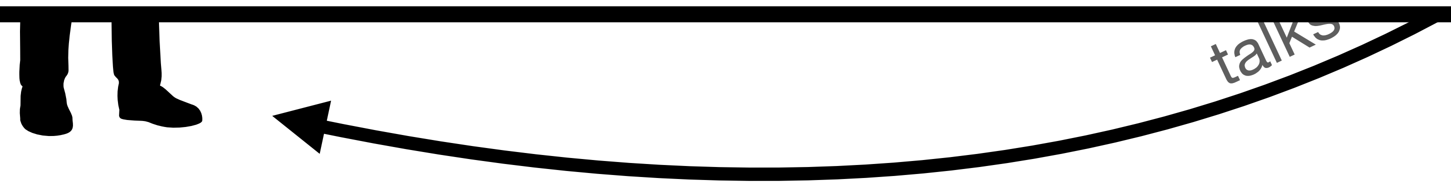
Qingchao Shen
College of Intelligence and Computing, Tianjin University
School of New Media and Communication, Tianjin University
China
qingchao@tju.edu.cn

Haoyang Ma
College of Intelligence and Computing, Tianjin University
China
haoyang_9804@tju.edu.cn

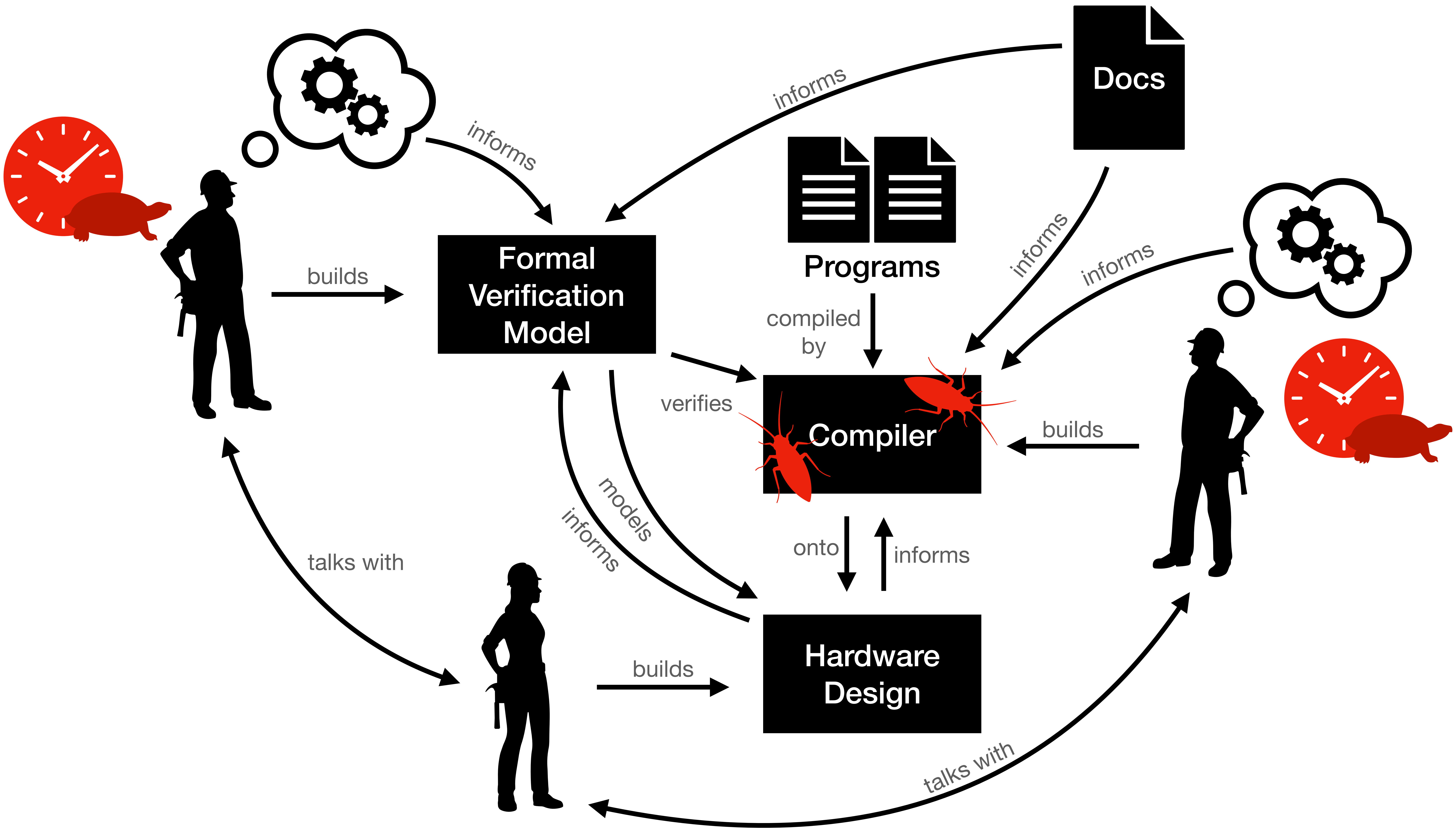
Junjie Chen*
College of Intelligence and Computing, Tianjin University
China
junjiechen@tju.edu.cn

(et al.)

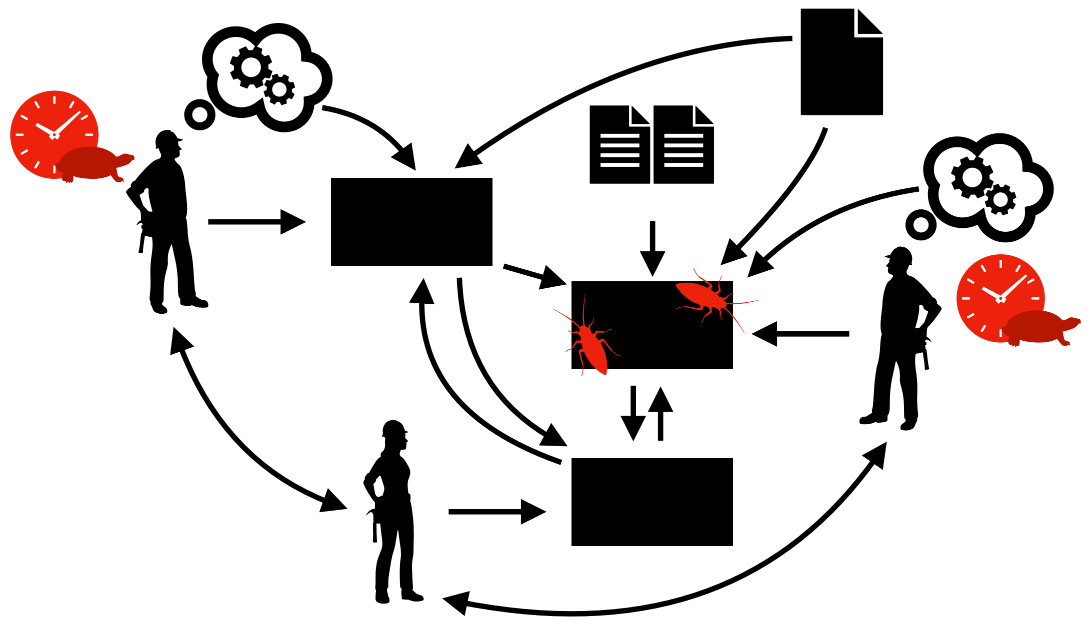
Finds bugs in TVM



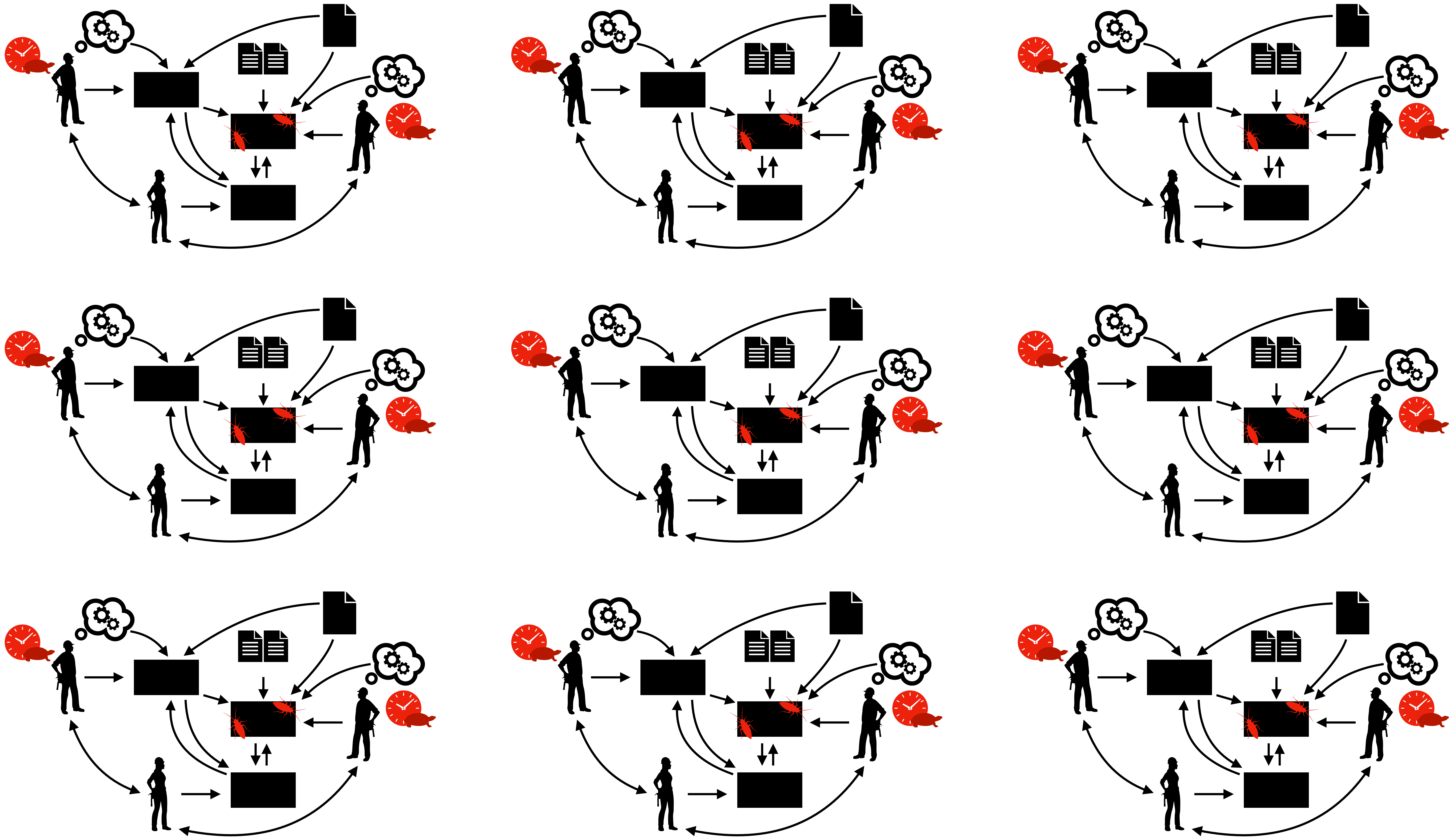
Why are compilers hard to build?



Why are compilers hard to build?



Why are compilers hard to build?



Why are compilers hard to build?

Why are compilers hard to build?

Building a compiler requires significant engineering effort and induces numerous bugs.

Why are compilers hard to build?

Building a compiler requires significant engineering effort and induces numerous bugs.

Those costs are multiplied with every new hardware design.

Why are compilers hard to build?

Building a compiler requires significant engineering effort and induces numerous bugs.

Those costs are multiplied with every new hardware design.

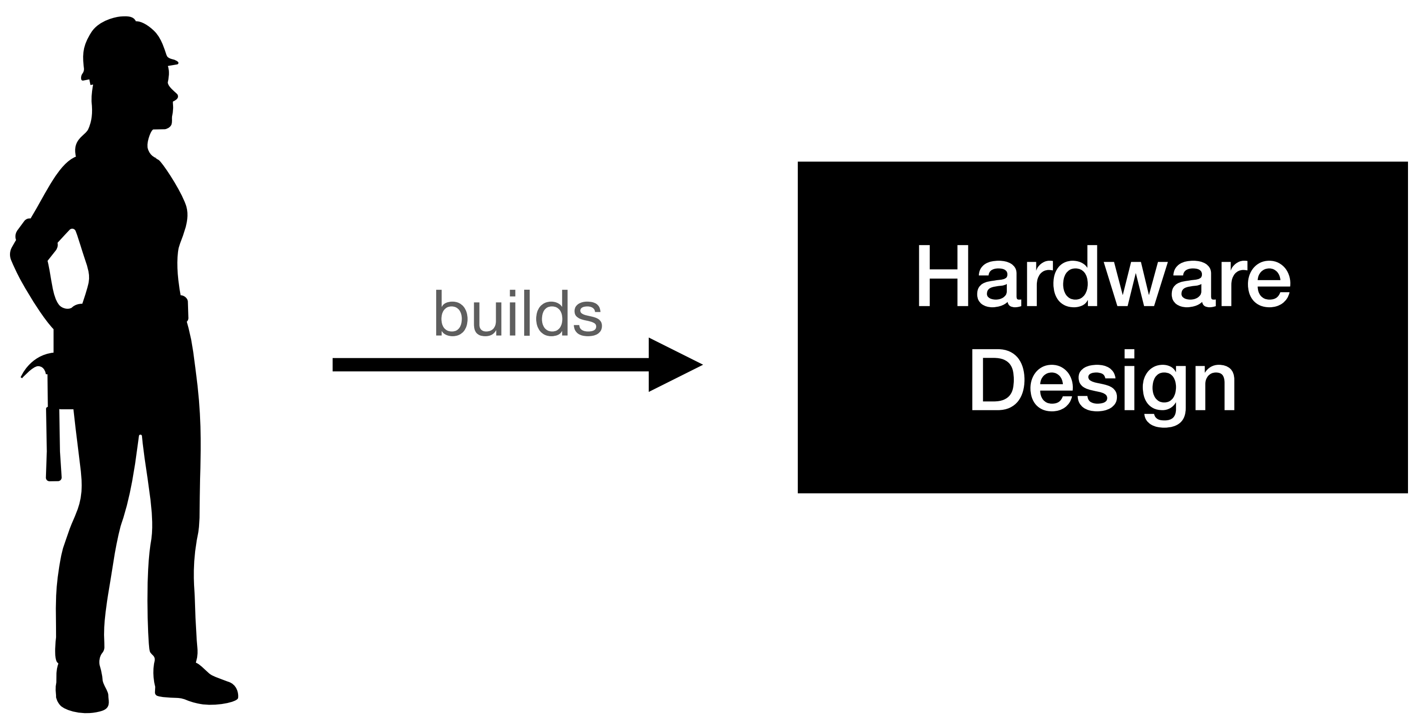
What if compilers were *synthesized?*
(i.e., automatically generated?)

Why are compilers hard to build?

What if compilers were synthesized?

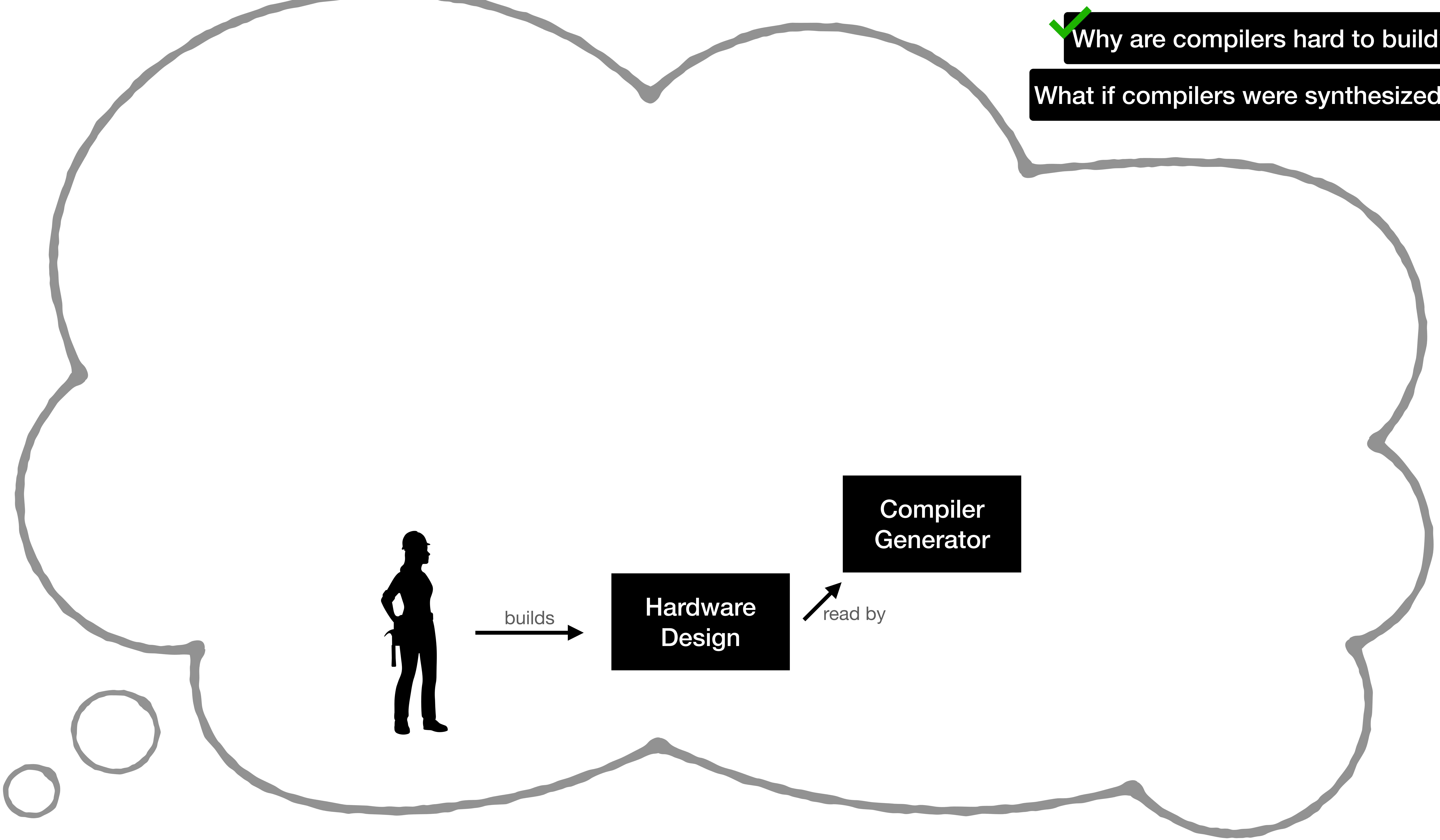
✓ Why are compilers hard to build?

What if compilers were synthesized?



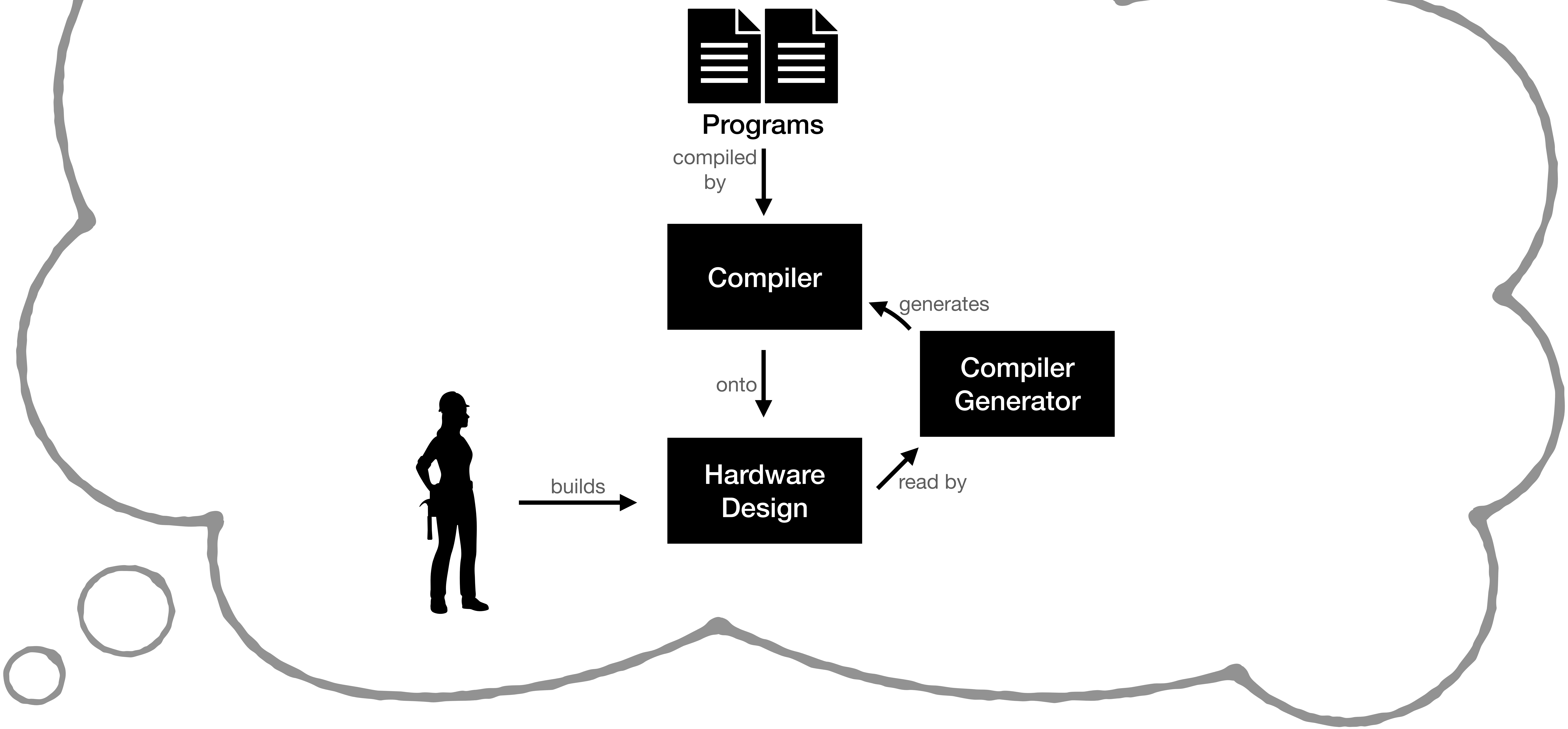
✓ Why are compilers hard to build?

What if compilers were synthesized?



✓ Why are compilers hard to build?

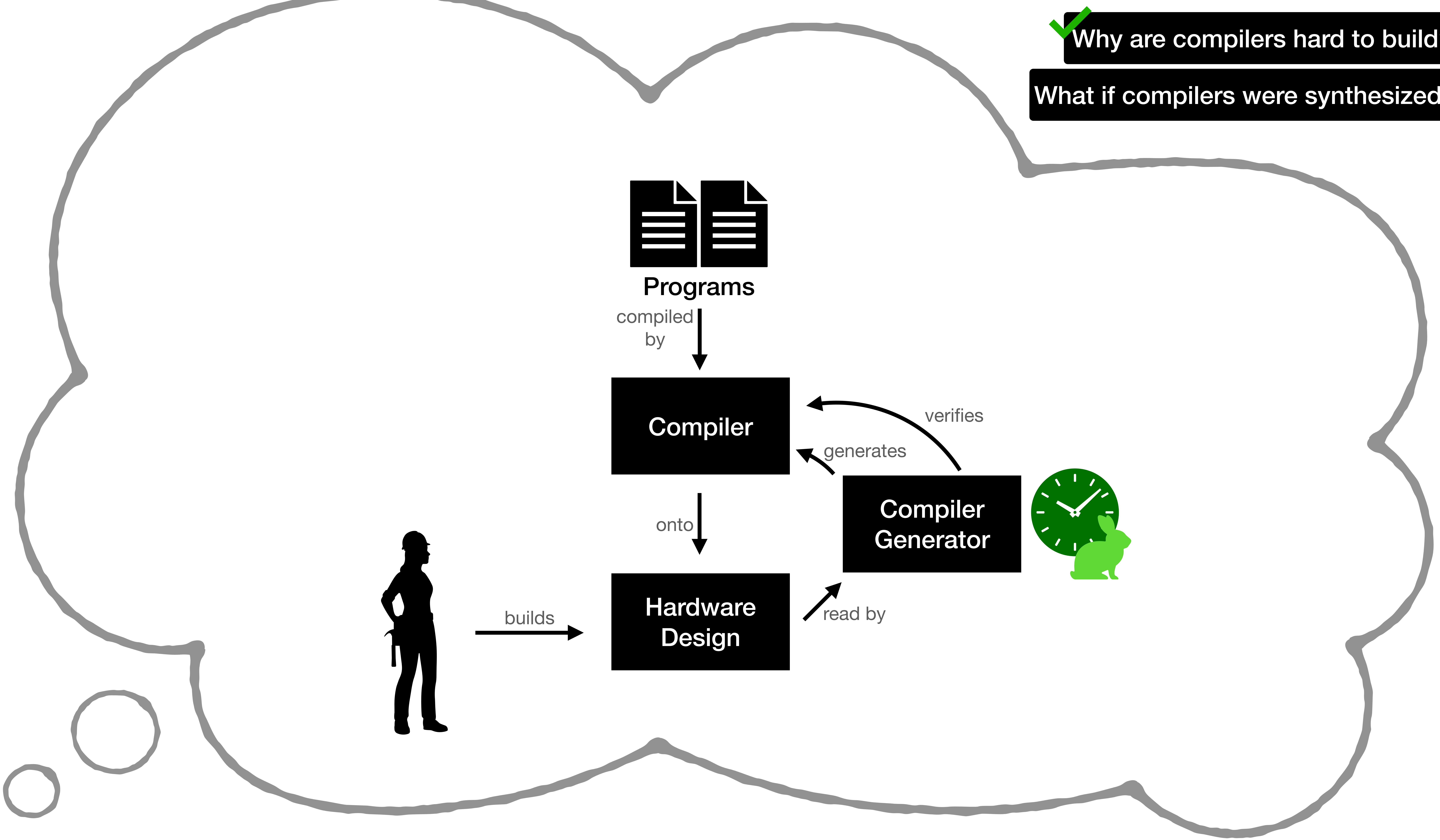
What if compilers were synthesized?




Why are compilers hard to build?
What if compilers were synthesized?
Programs
compiled by
Compiler
generates
Compiler Generator
onto
Hardware Design
read by
builds

Why are compilers hard to build?

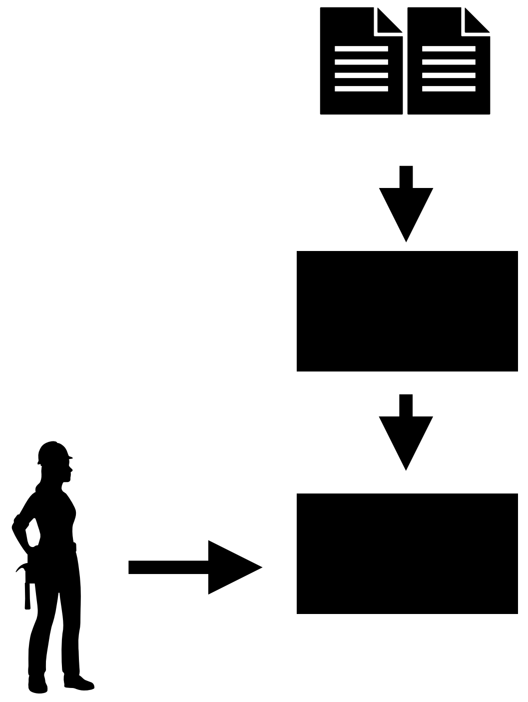
What if compilers were synthesized?




Why are compilers hard to build?
What if compilers were synthesized?
Programs
compiled by
Compiler
verifies
generates
Compiler Generator
onto
Hardware Design
builds
read by

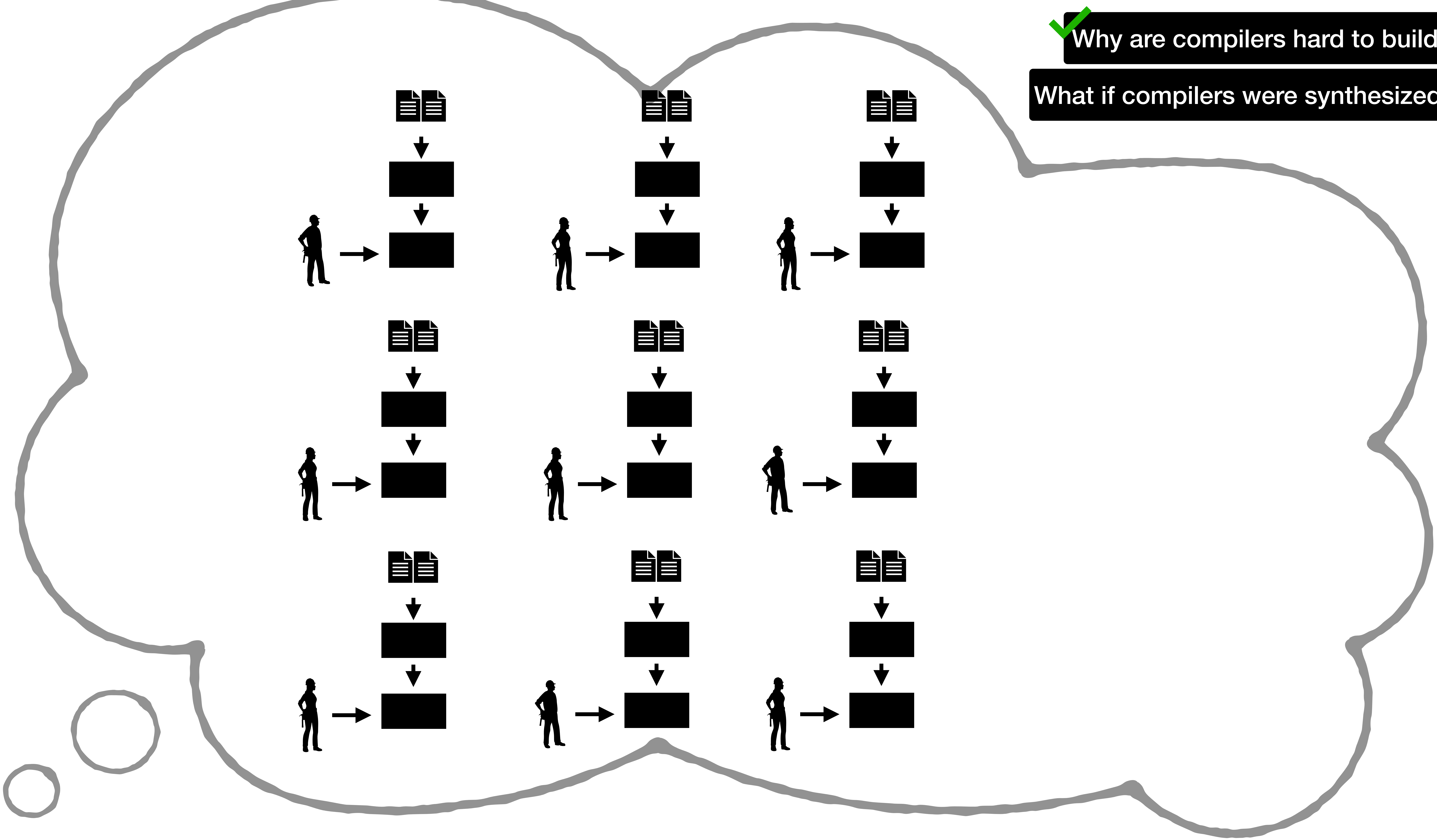
Why are compilers hard to build?

What if compilers were synthesized?



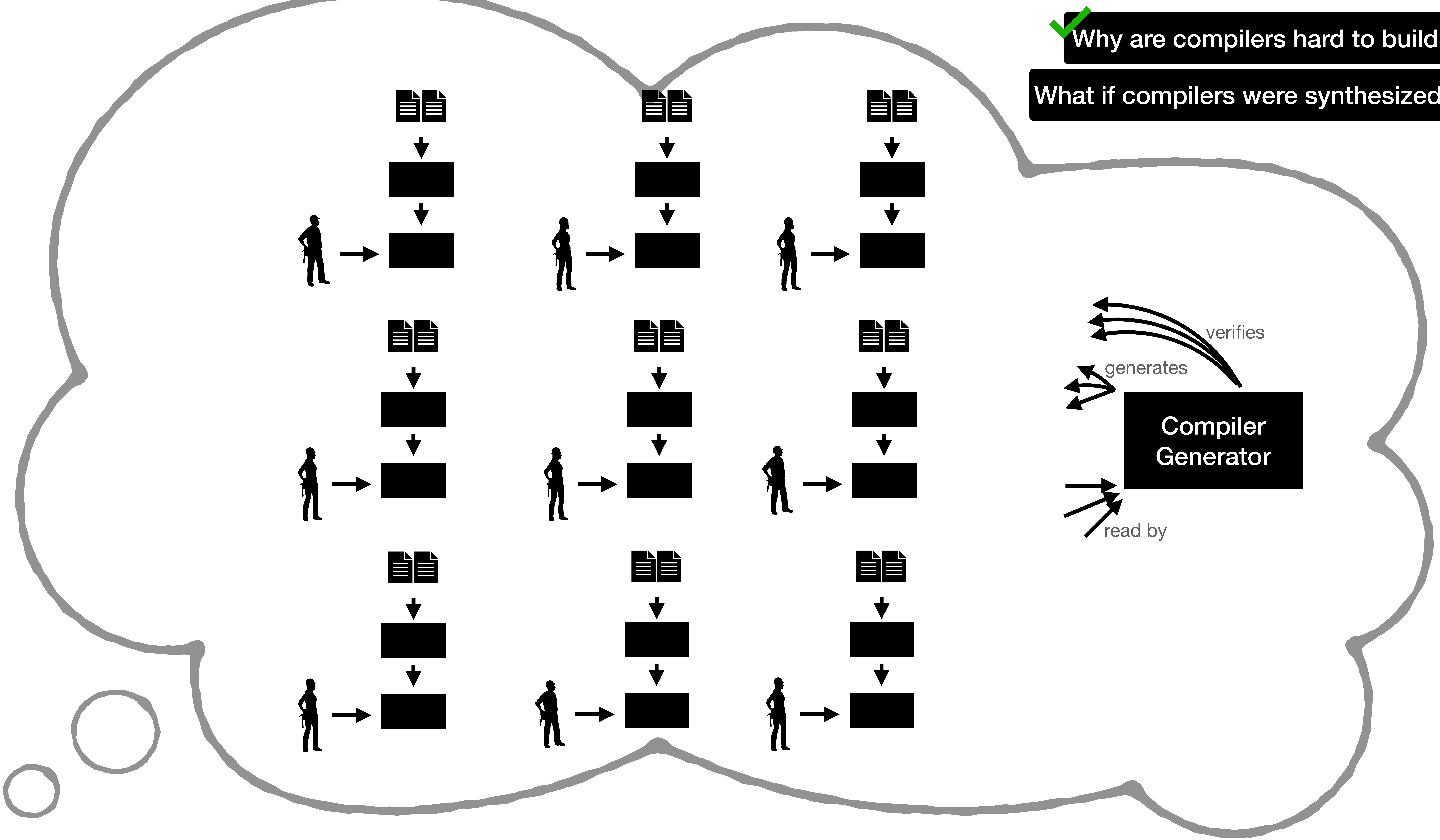
Why are compilers hard to build?

What if compilers were synthesized?



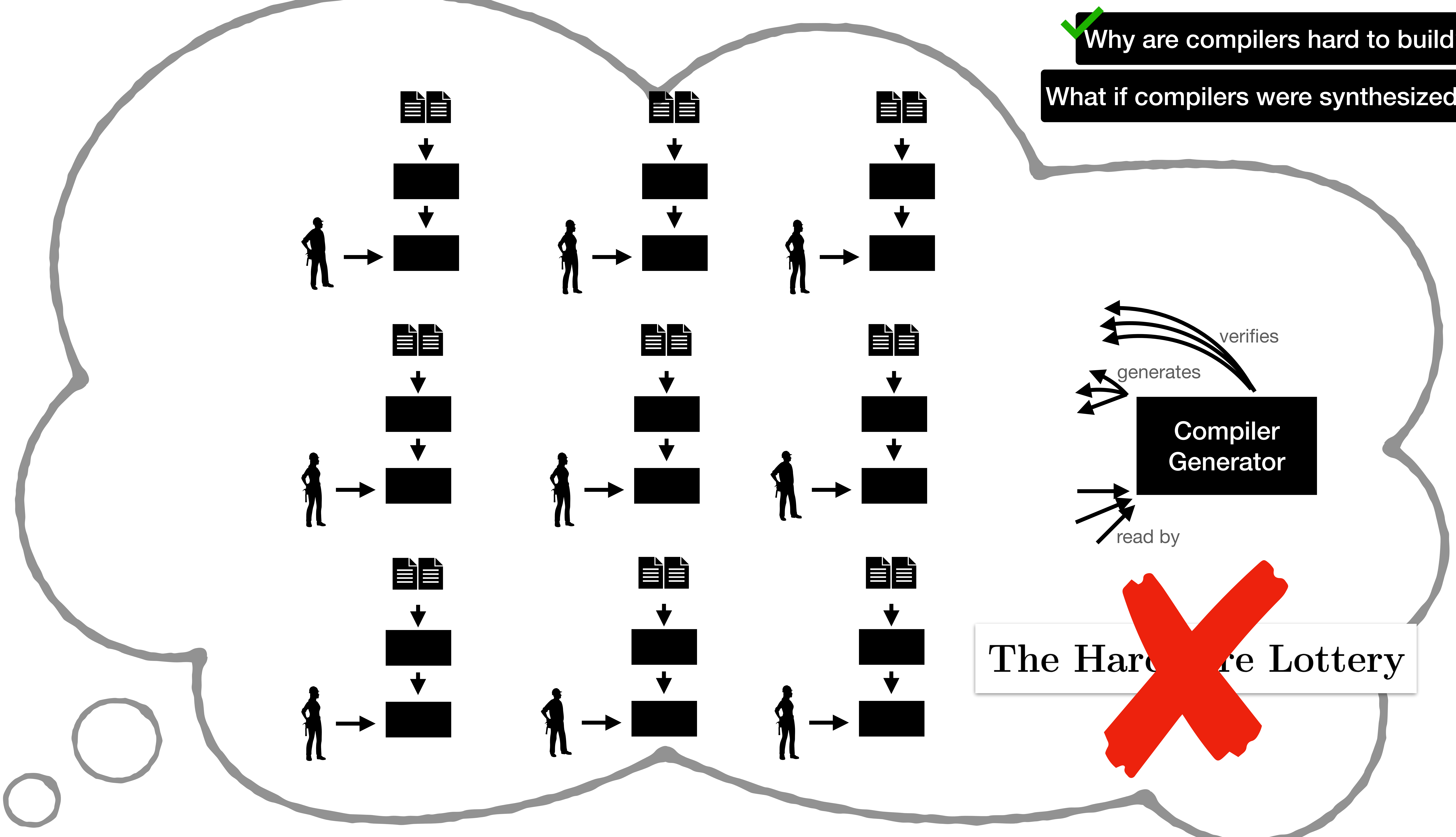
Why are compilers hard to build?

What if compilers were synthesized?



Why are compilers hard to build?

What if compilers were synthesized?



✓ Why are compilers hard to build?

What if compilers were synthesized?

What if compilers were synthesized?

Automatically generating compilers can reduce engineering effort and eliminate bugs.

What if compilers were synthesized?

Automatically generating compilers can reduce engineering effort and eliminate bugs.

Furthermore, the approach scales with new hardware designs, thus fighting against the hardware lottery!

Compilers should be generated from formal models of hardware.

Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Generating Compilers → Why Now? → Case Study: Lakeroad → Call to Action

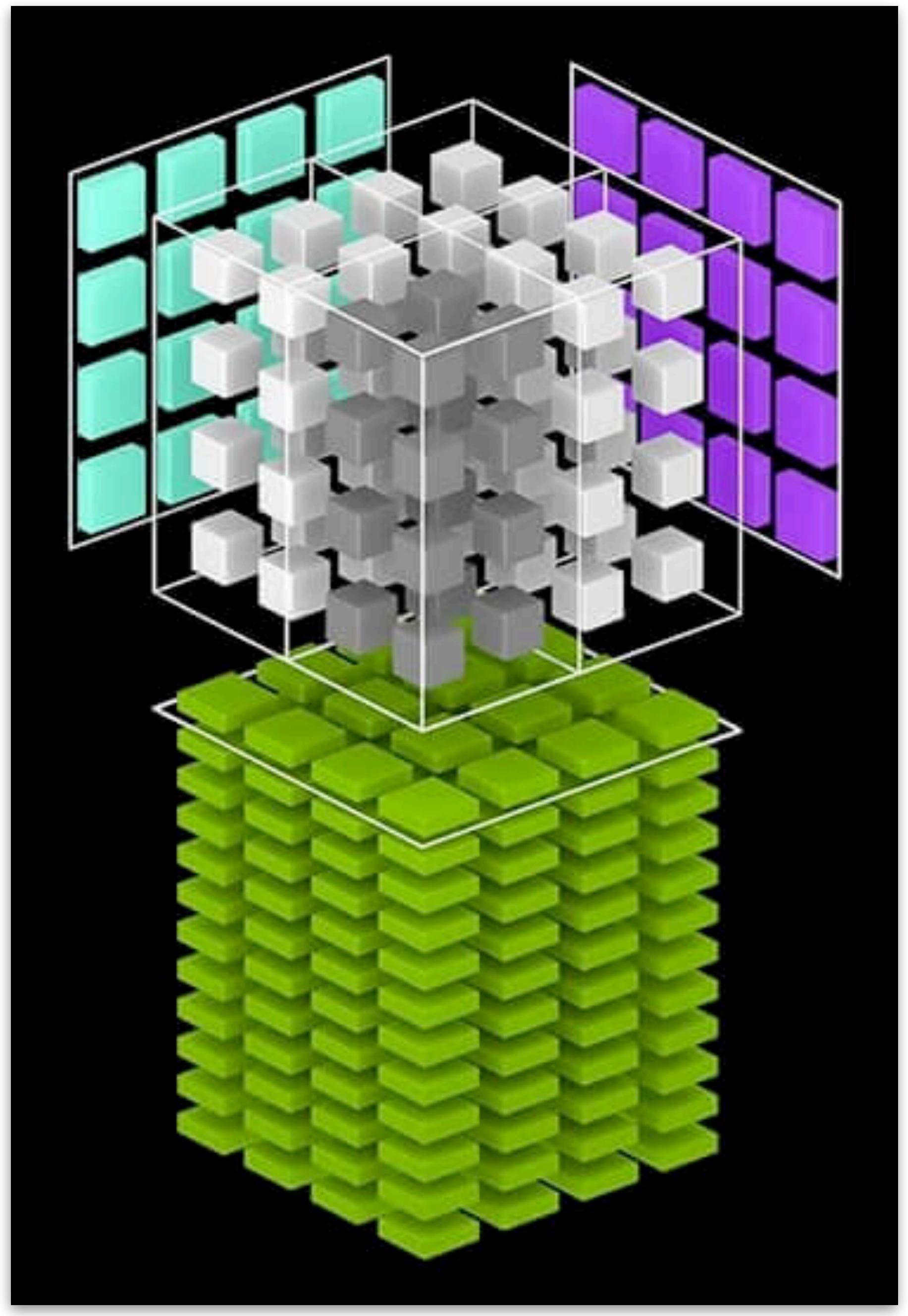
Generating Compilers → Why Now? → Case Study: Lakeroad → Call to Action

Compilers should be generated from formal models of hardware.

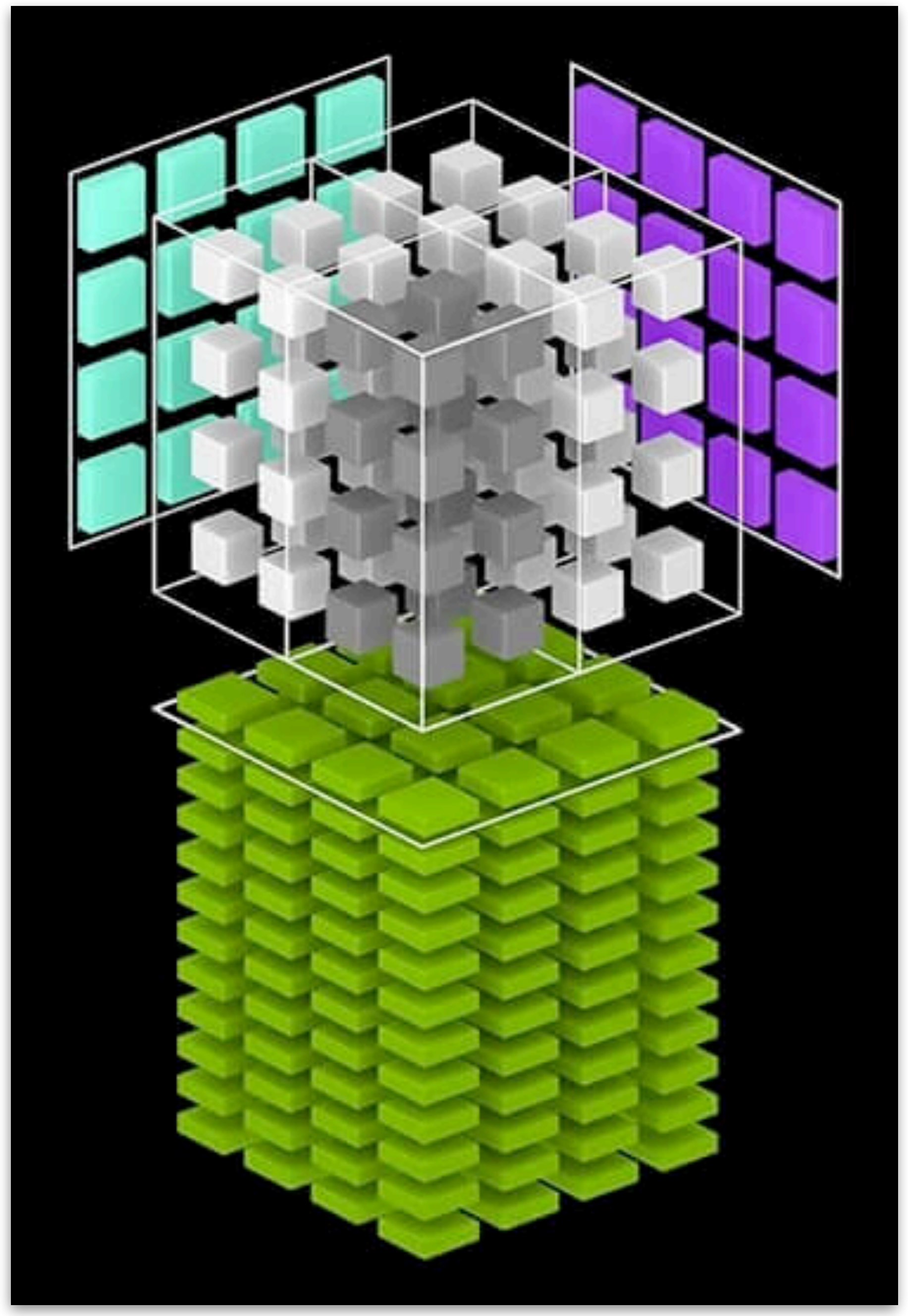
With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

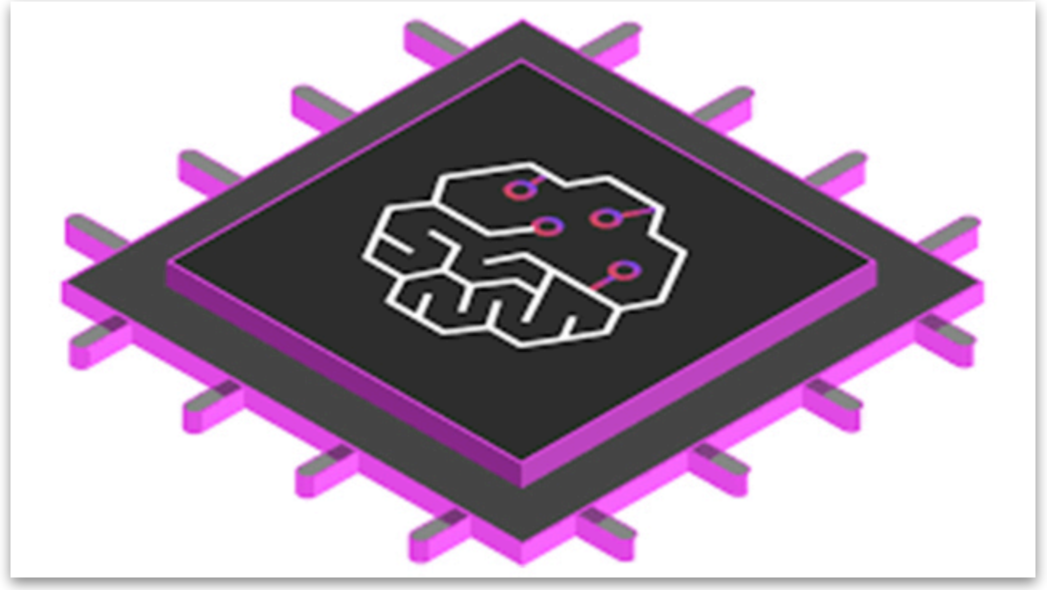


NVIDIA Tensor Cores

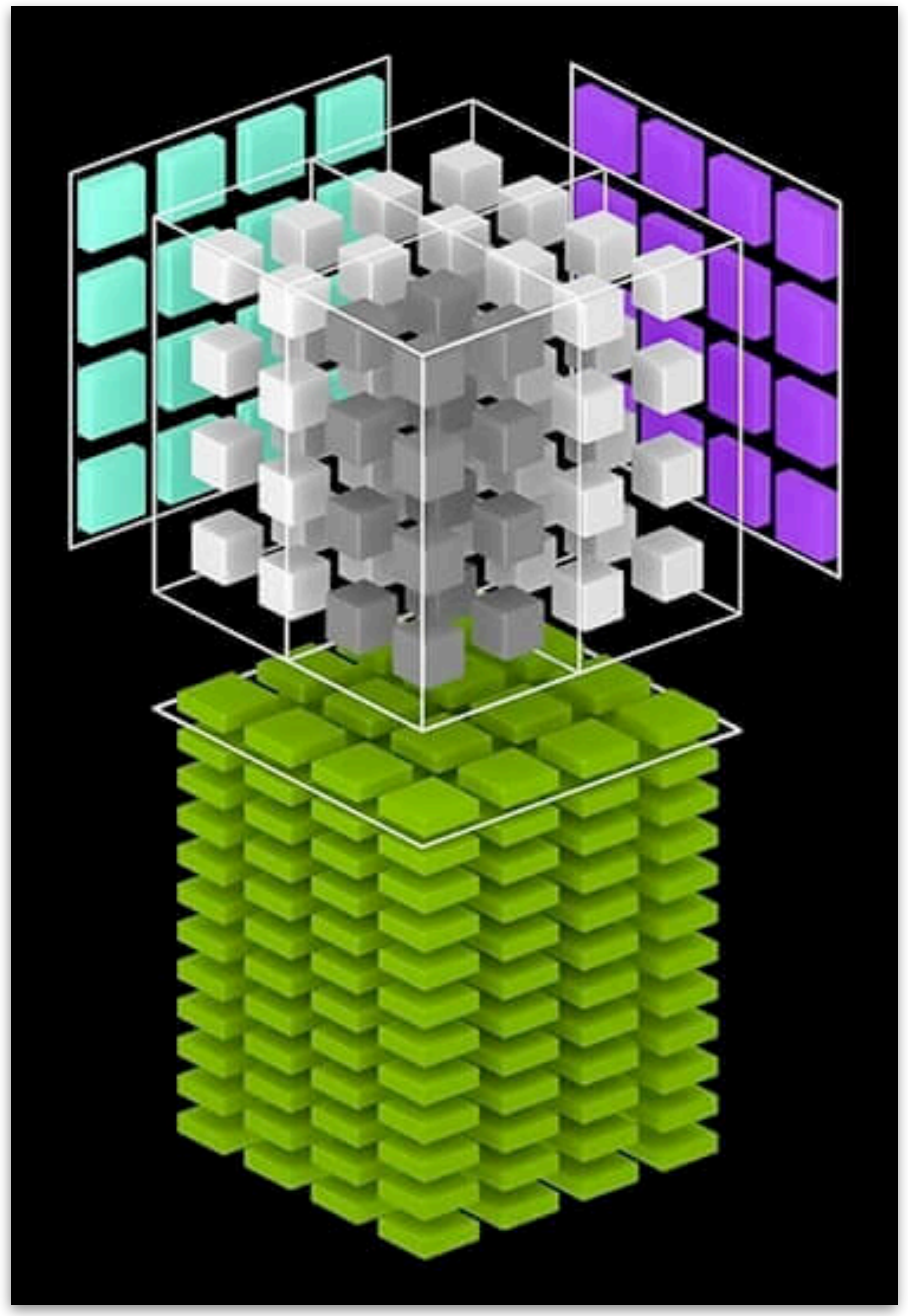


NVIDIA Tensor Cores

Google TPU

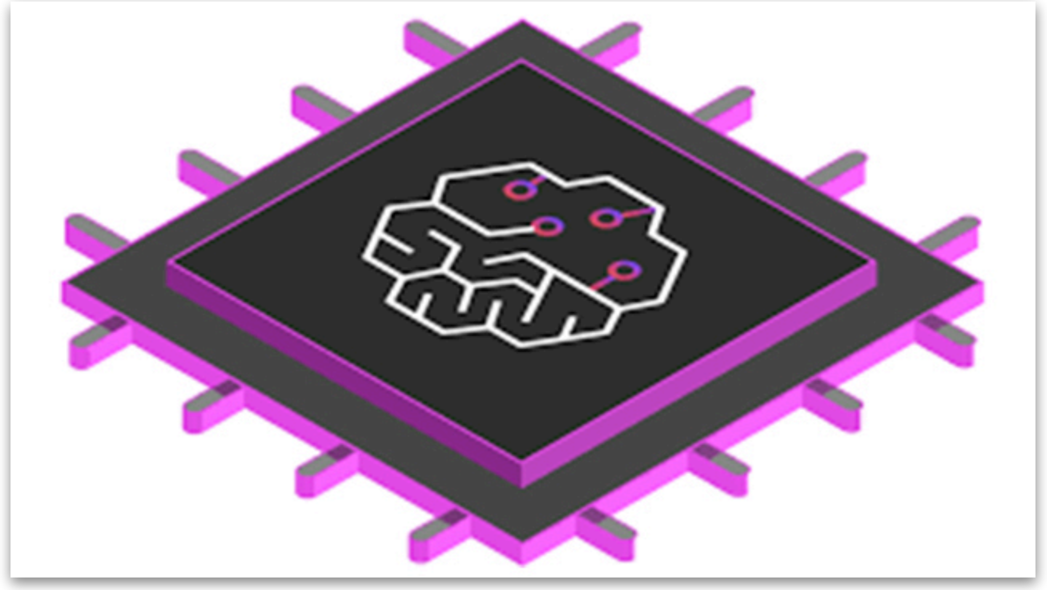


AWS Inferentia

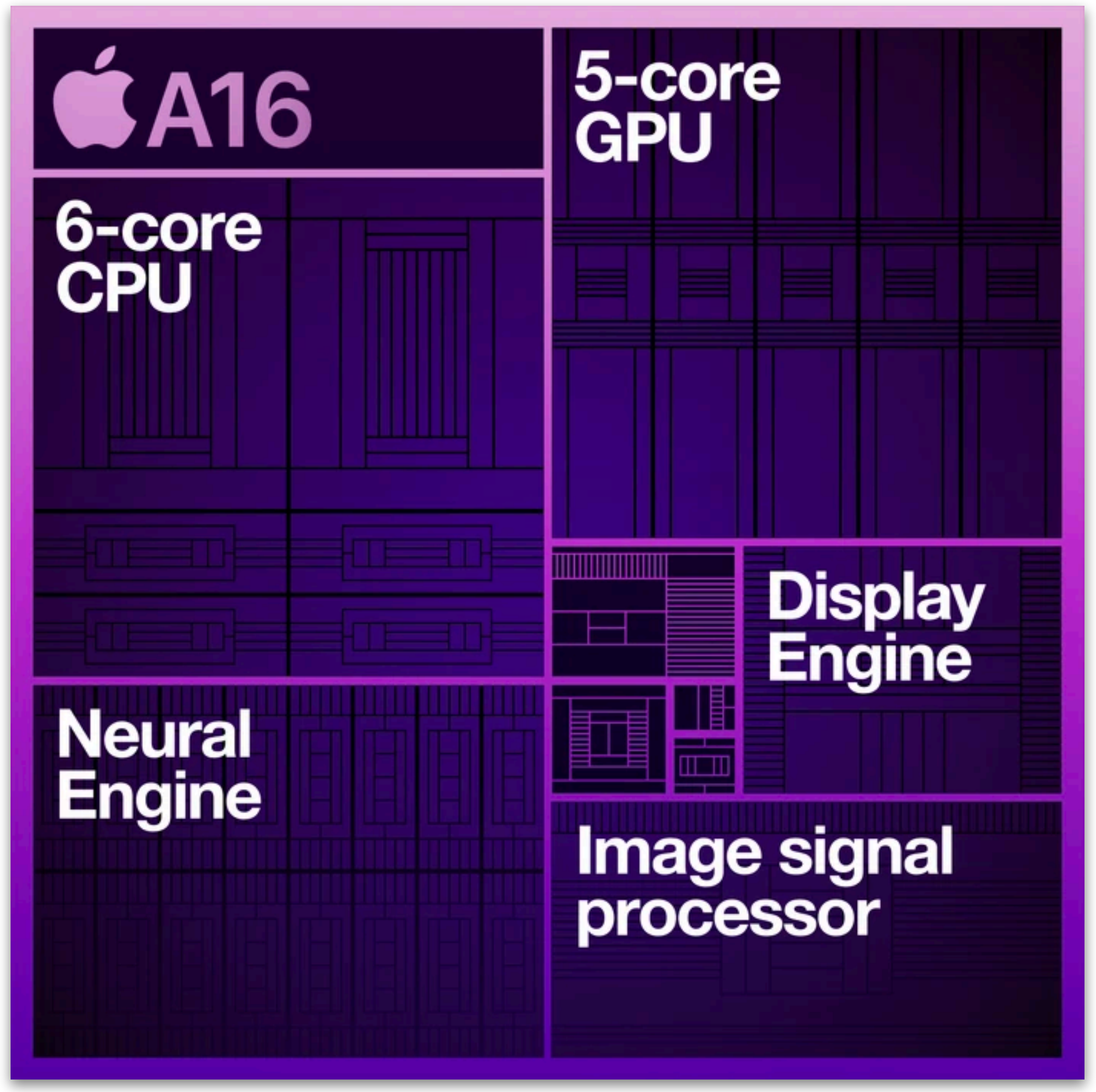


NVIDIA Tensor Cores

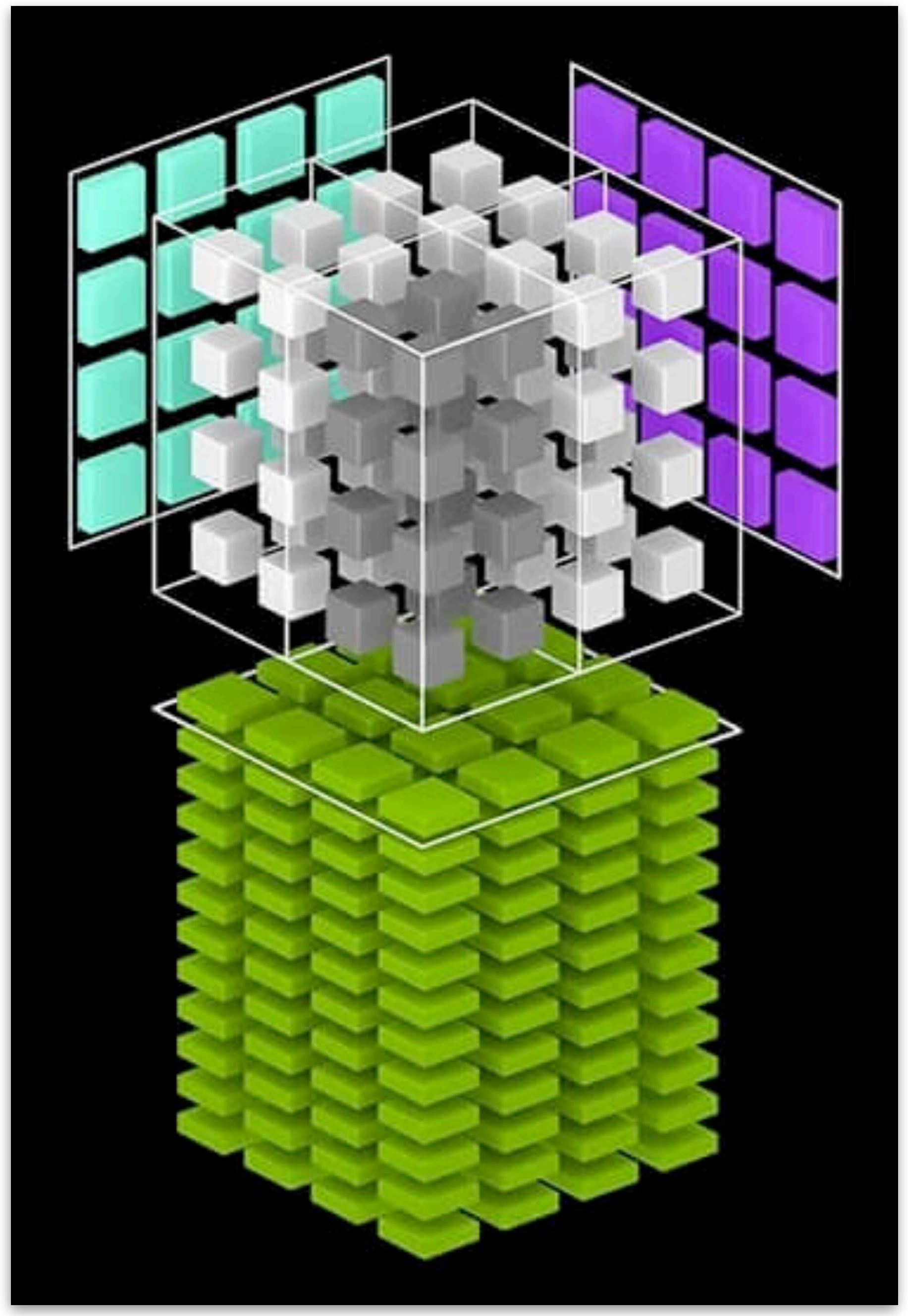
Google TPU



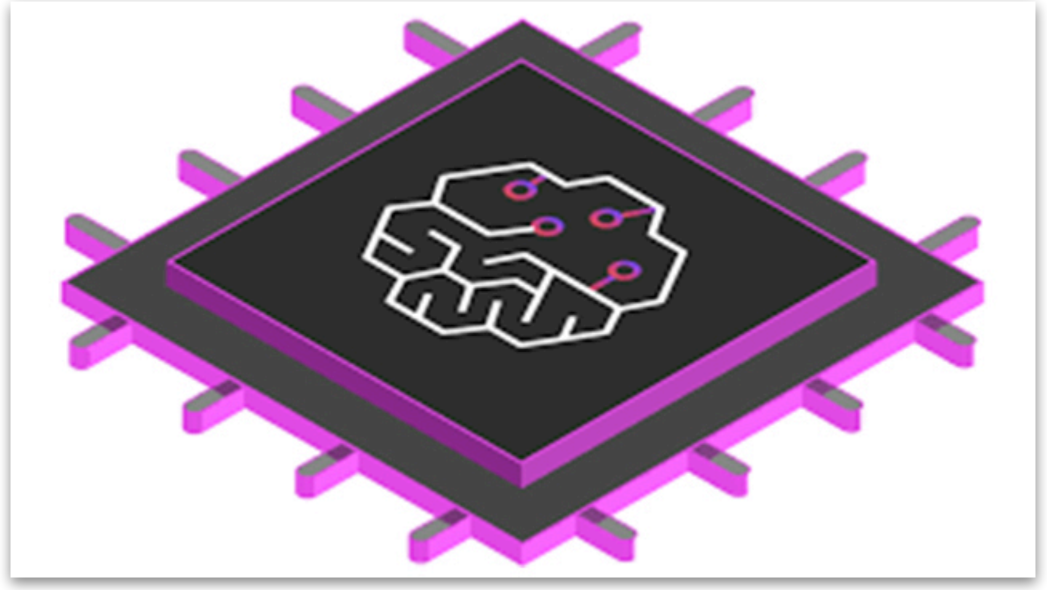
AWS Inferentia

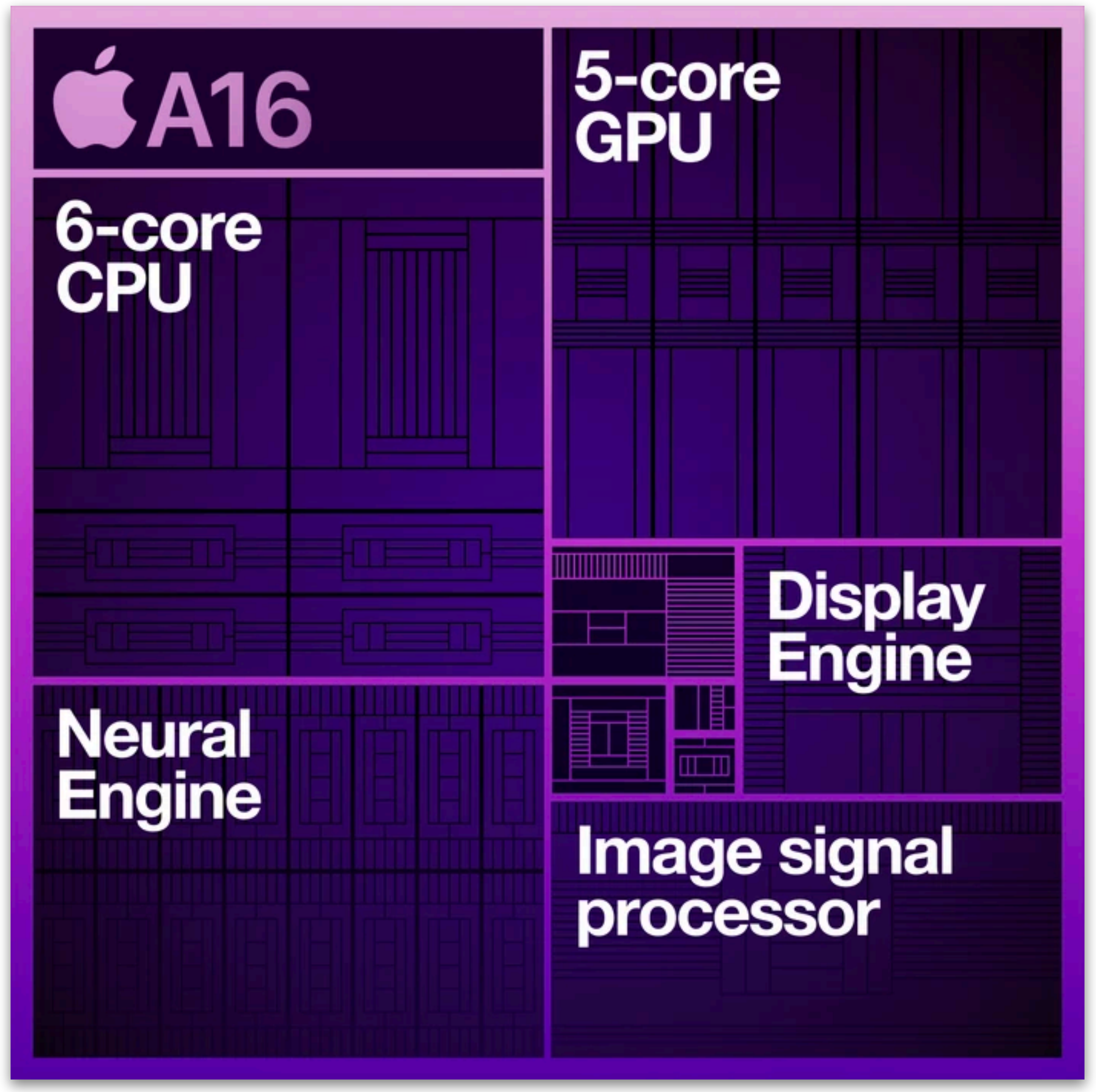


Apple A16


NVIDIA Tensor Cores

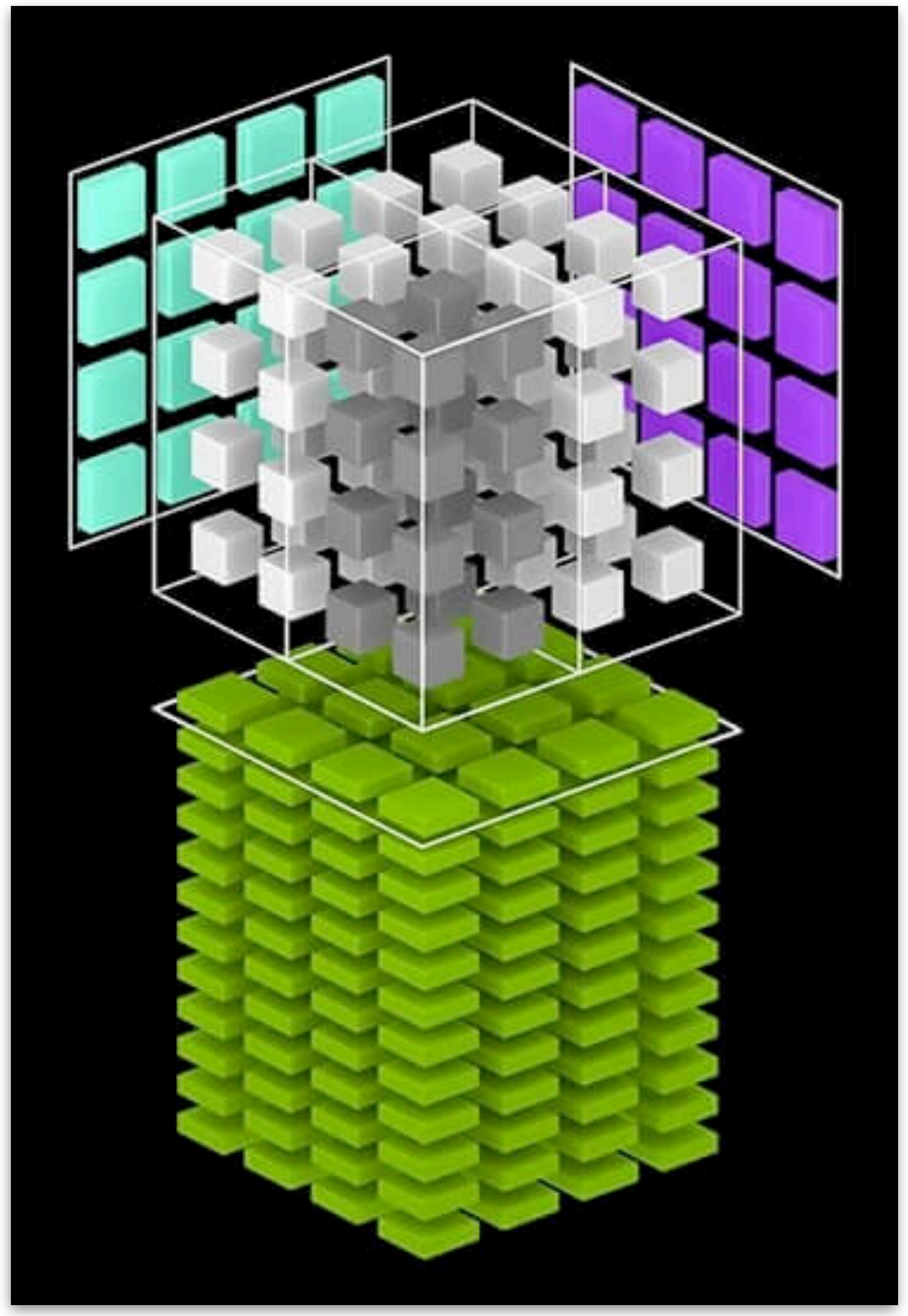

Google TPU


AWS Inferentia



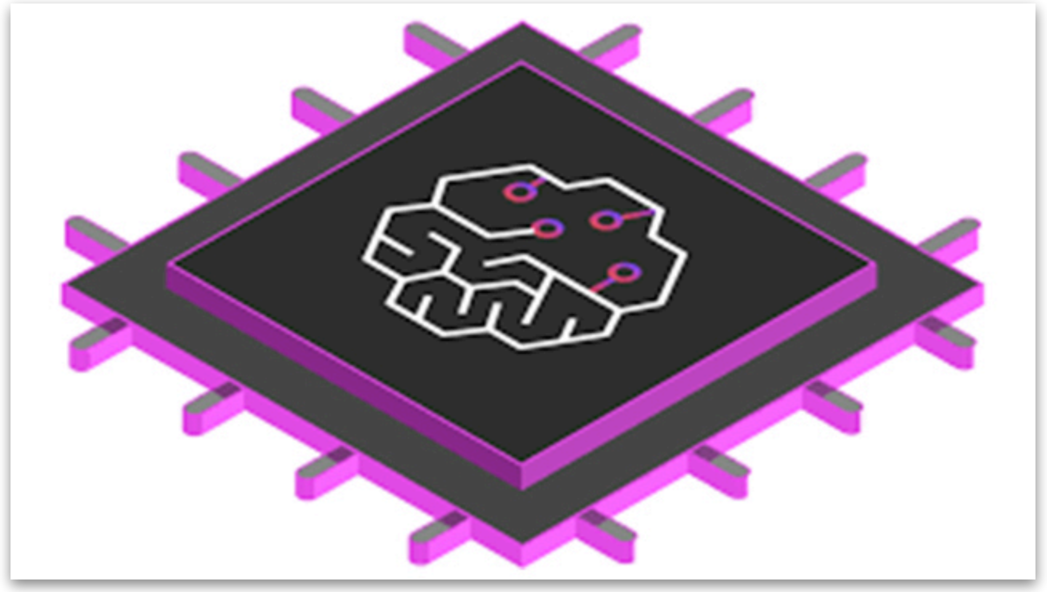
Apple A16

Xilinx Zynq

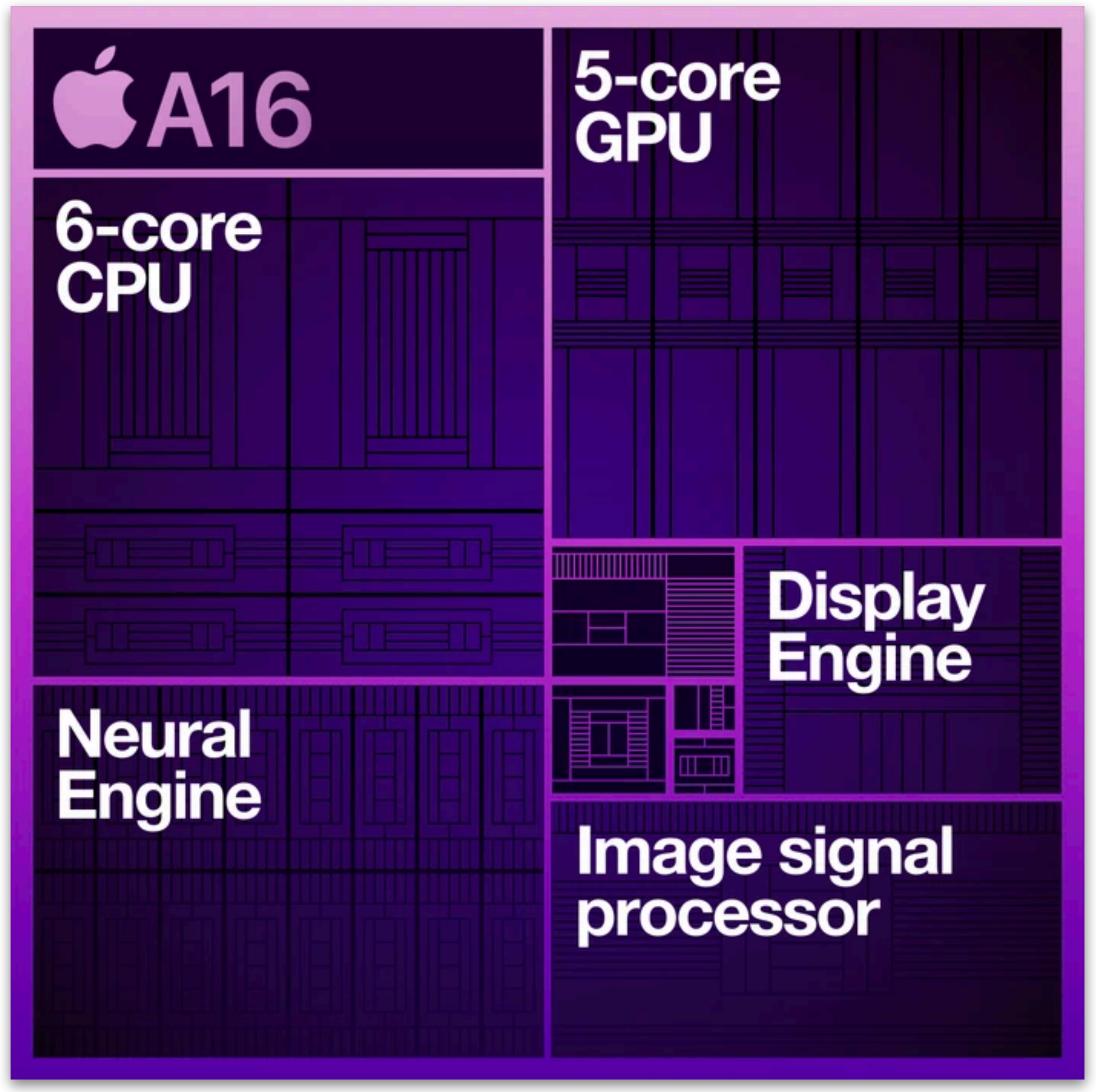


NVIDIA Tensor Cores

Google TPU

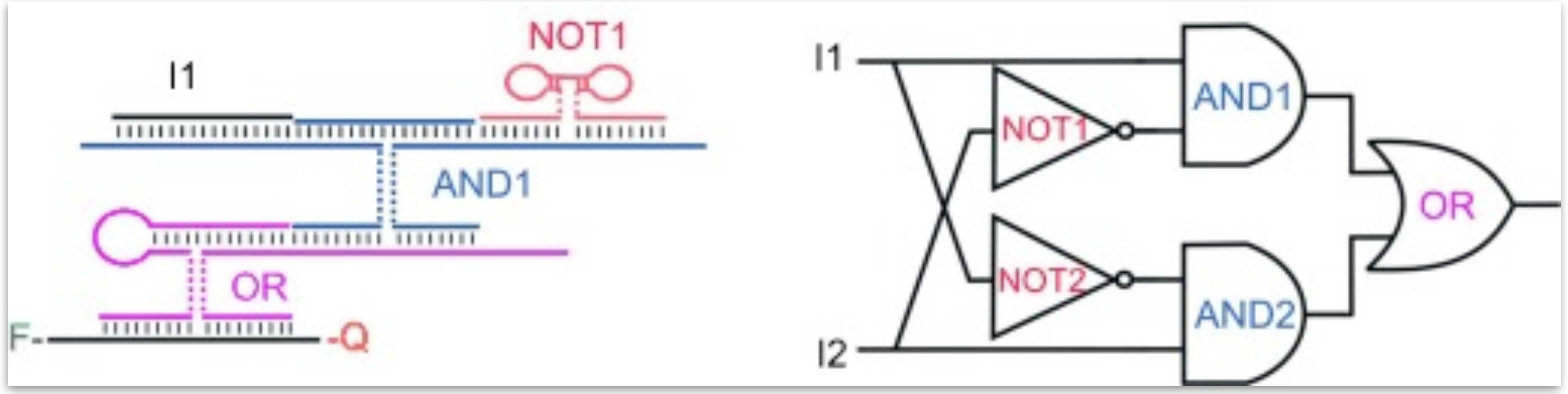


AWS Inferentia

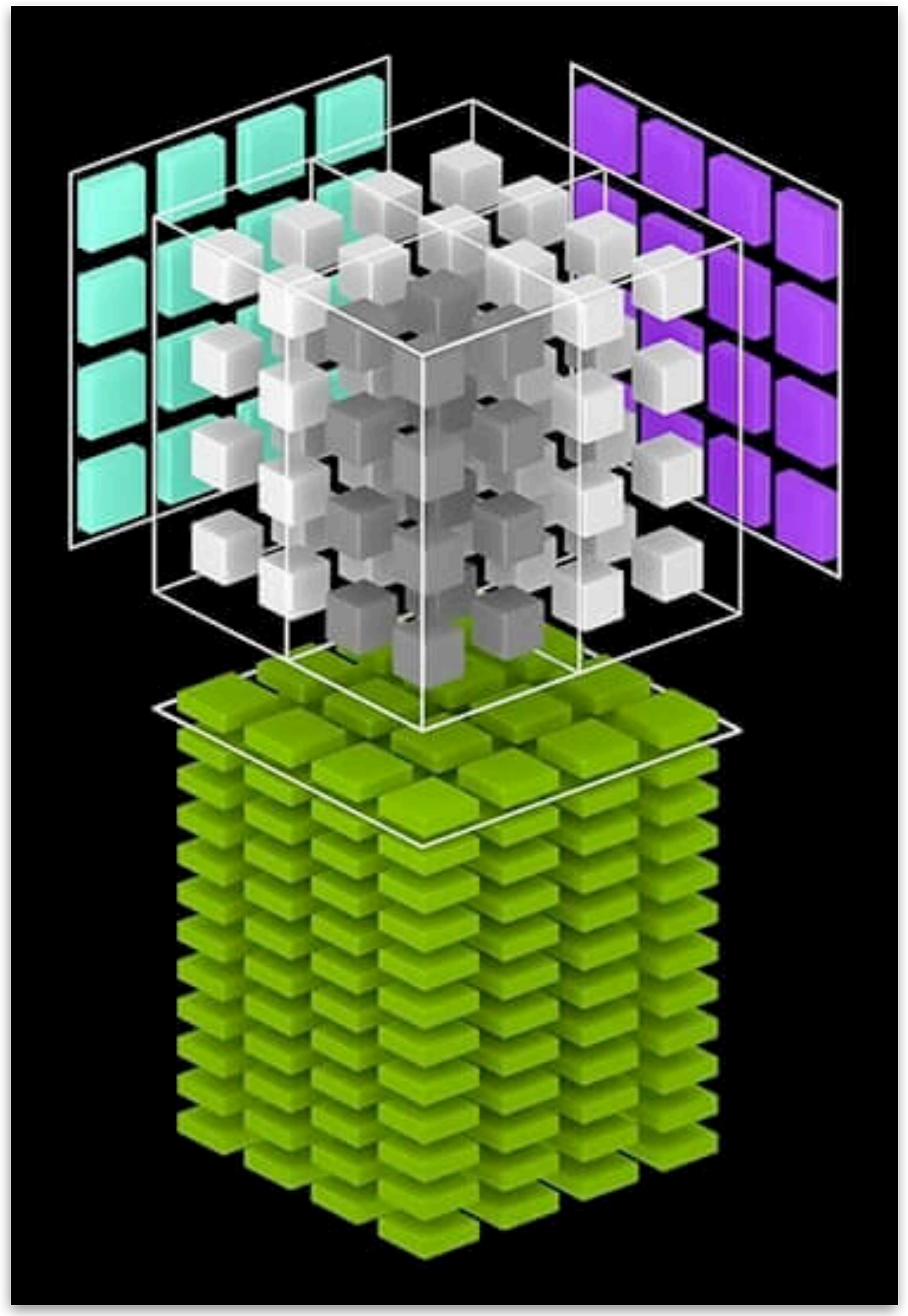


Apple A16

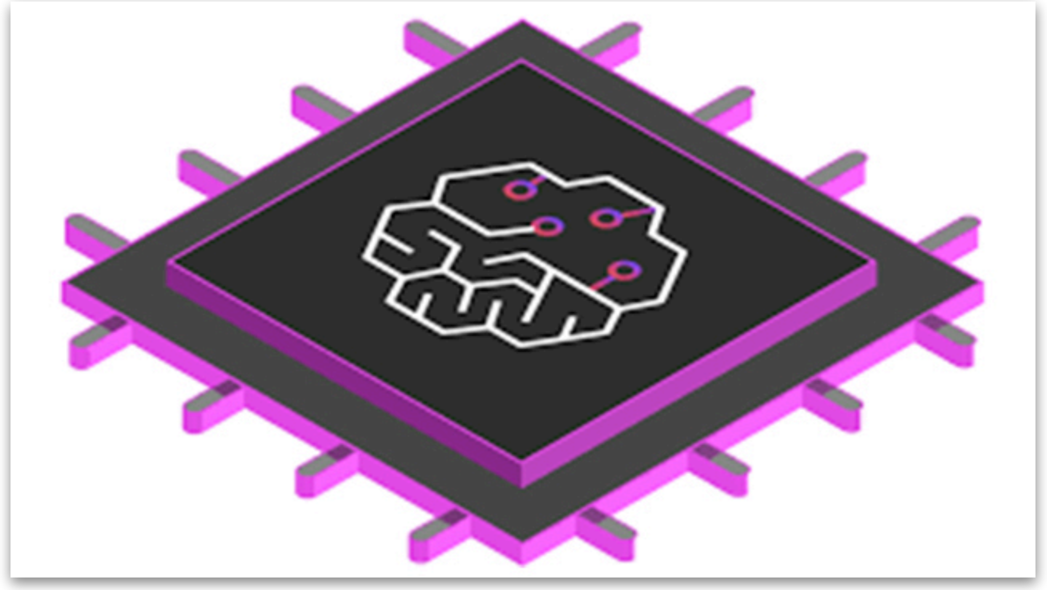
Xilinx Zynq

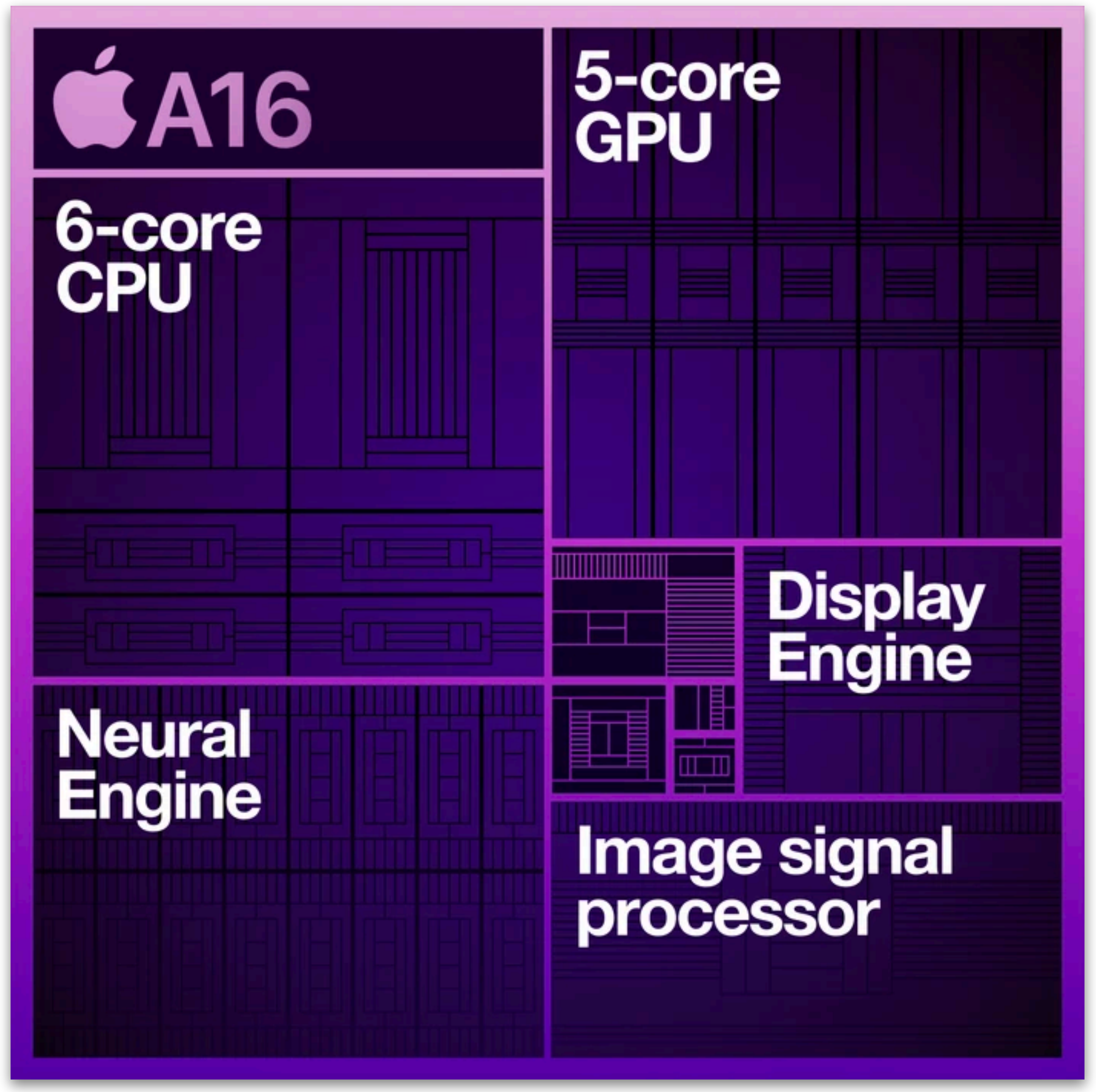


Gerasimova et al. *Connectable DNA Logic Gates: OR and XOR Logics.*


NVIDIA Tensor Cores

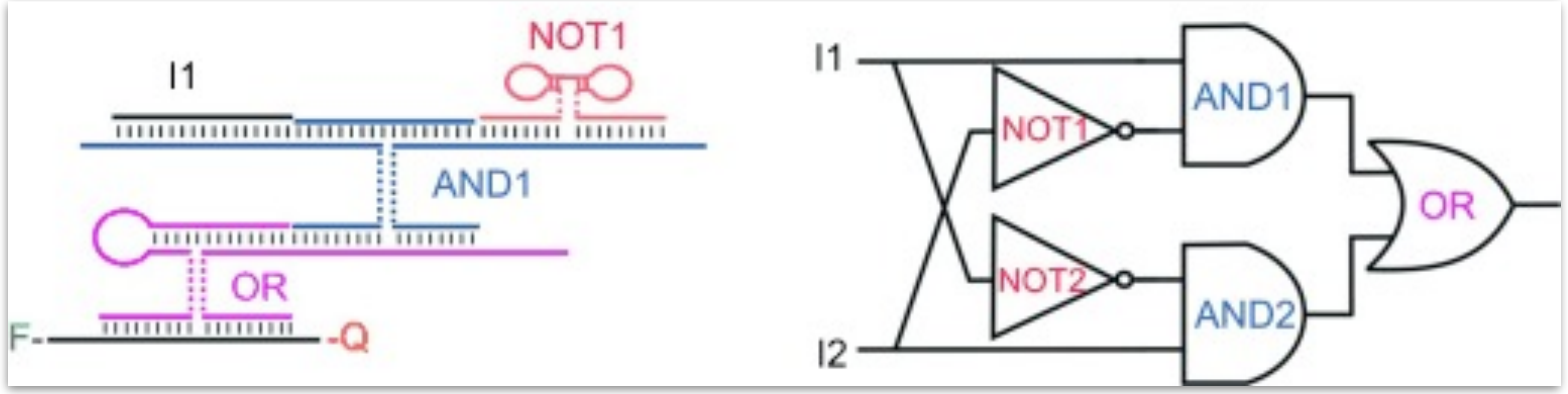

Google TPU


AWS Inferentia

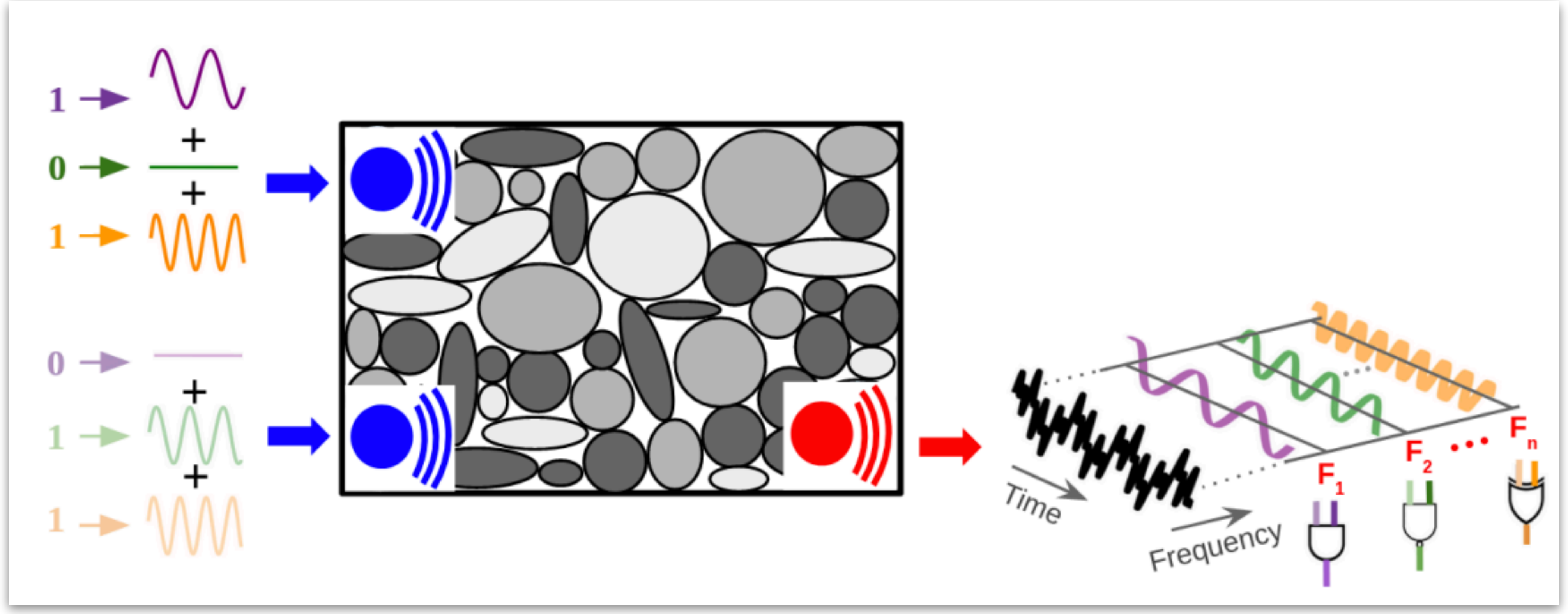


Apple A16

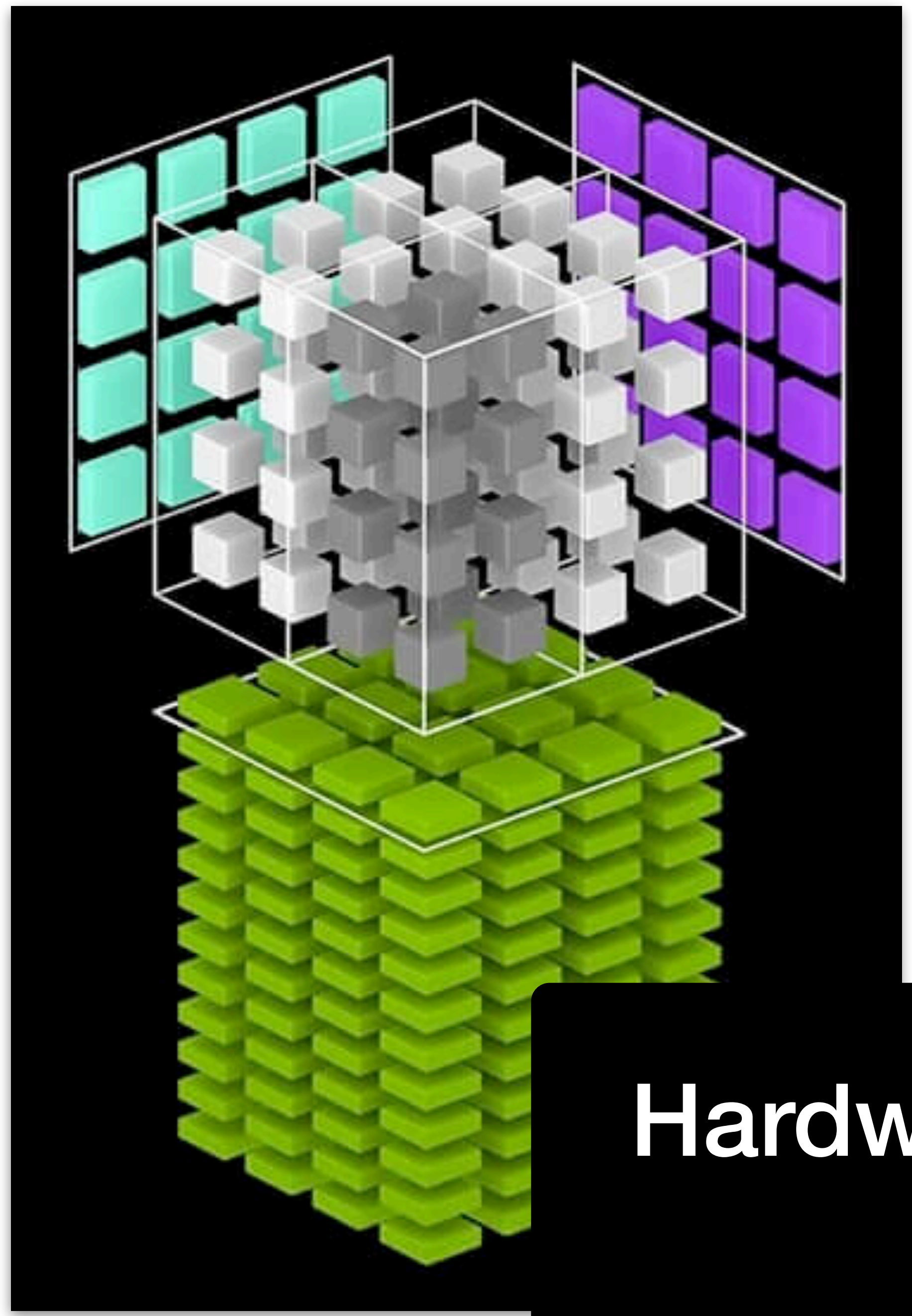
Xilinx Zynq



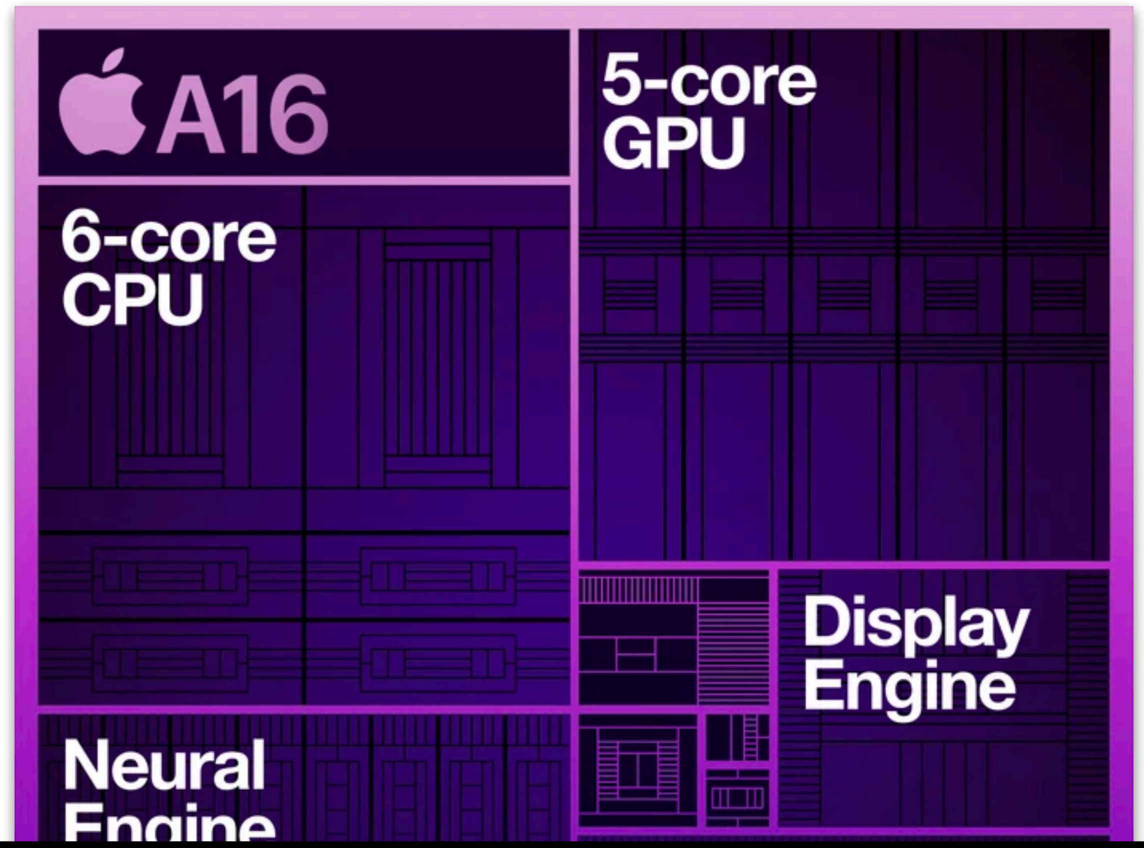
Gerasimova et al. *Connectable DNA Logic Gates: OR and XOR Logics.*



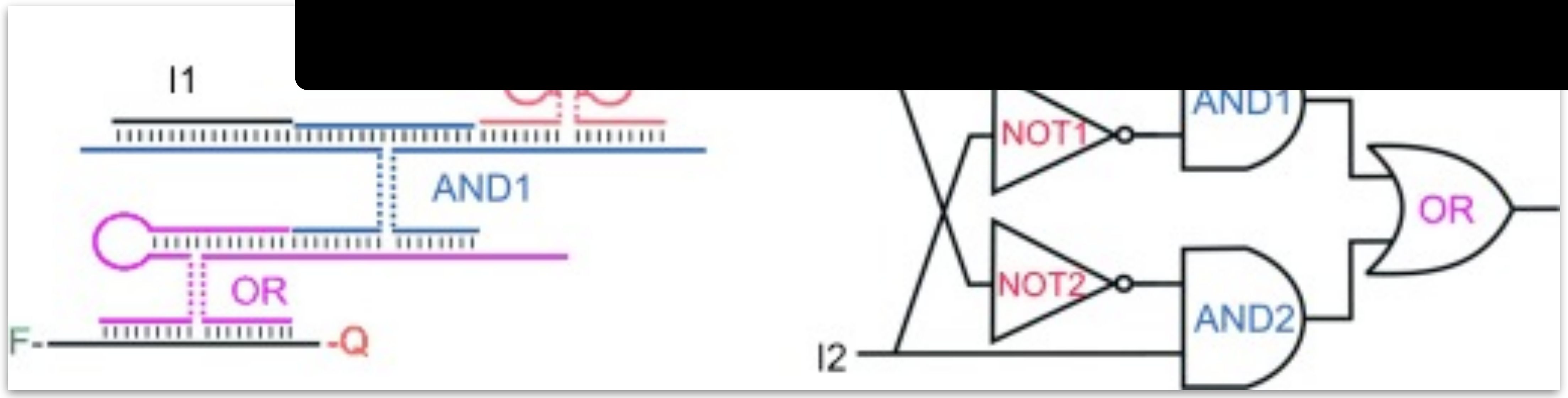
Parsa et al. *Universal Mechanical Polycomputation in Granular Matter.*


NVIDIA Tensc

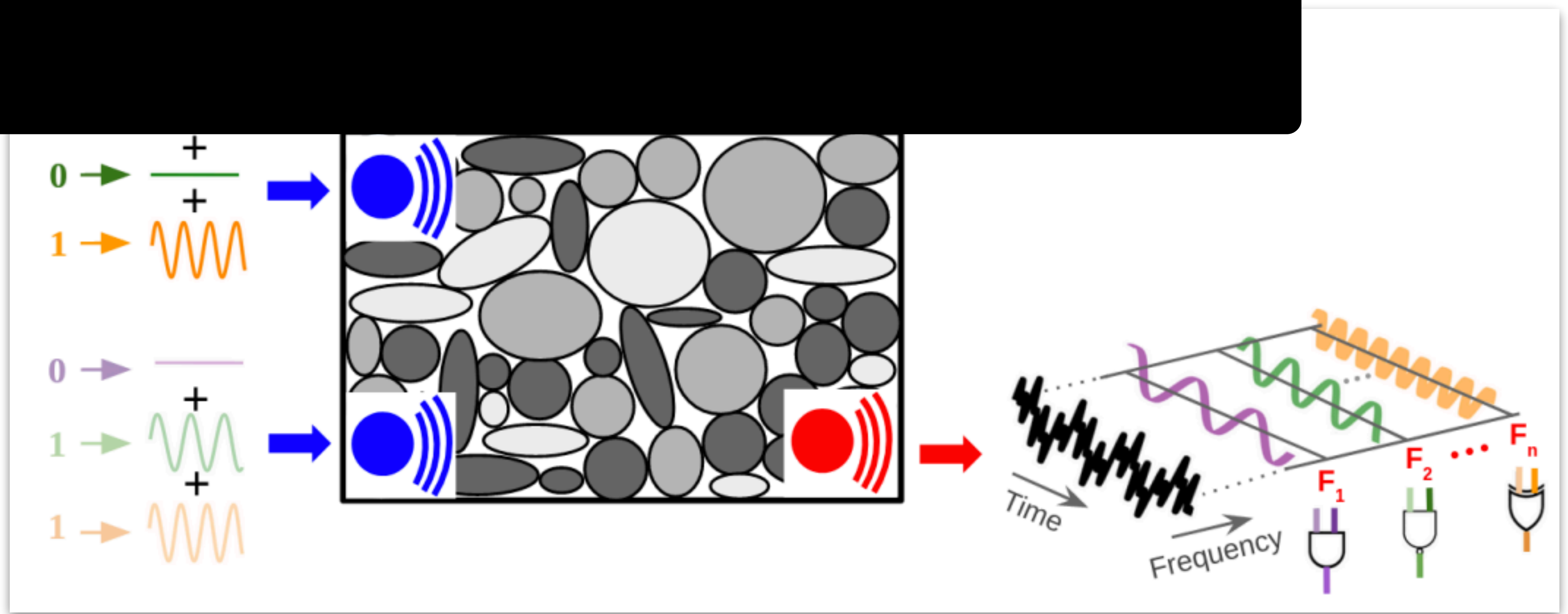

Google TPU



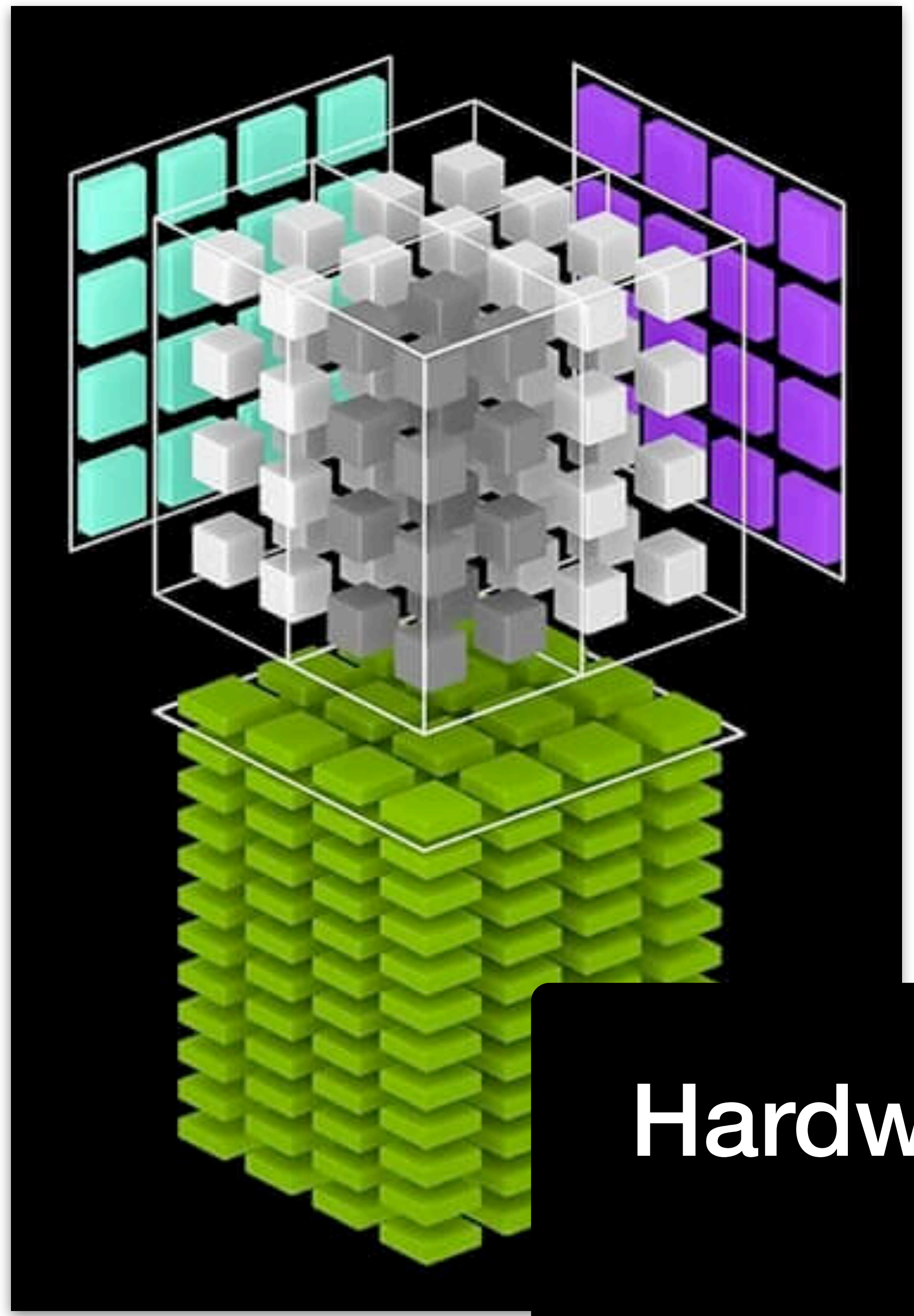
Hardware is growing more diverse; more compilers are desperately needed!



Gerasimova et al. *Connectable DNA Logic Gates: OR and XOR Logics.*

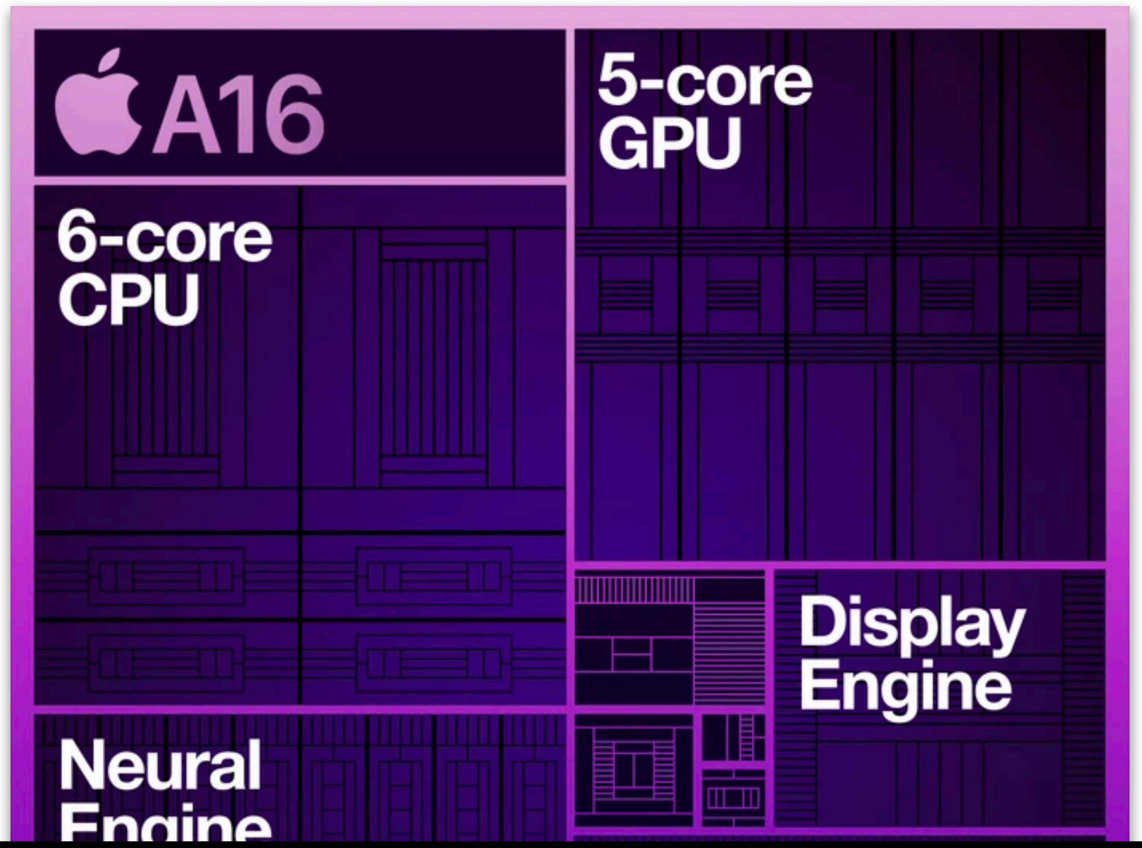


Parsa et al. *Universal Mechanical Polycomputation in Granular Matter.*



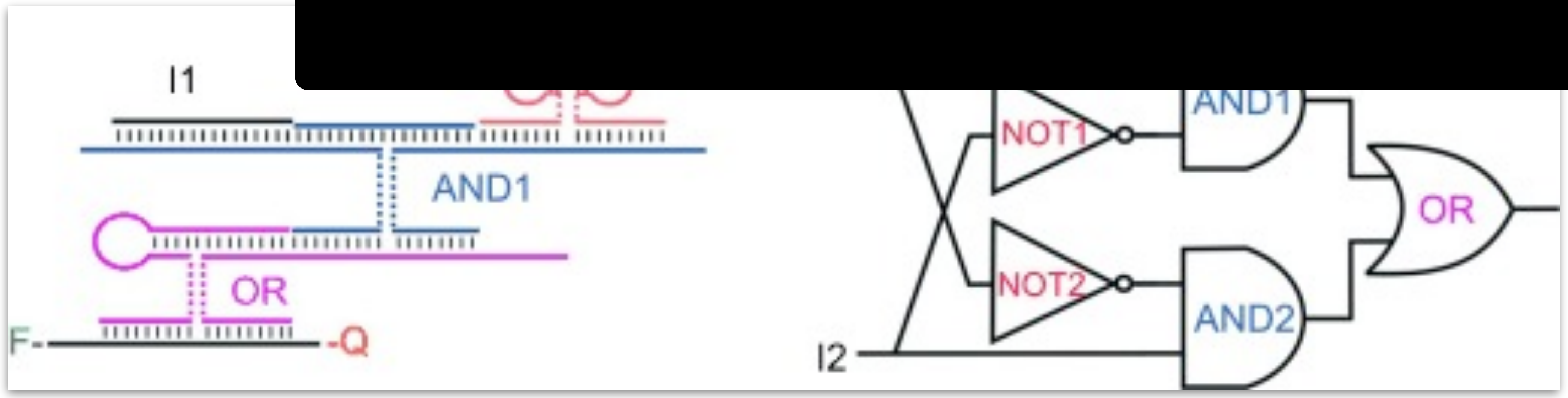
NVIDIA Tens

Google TPU

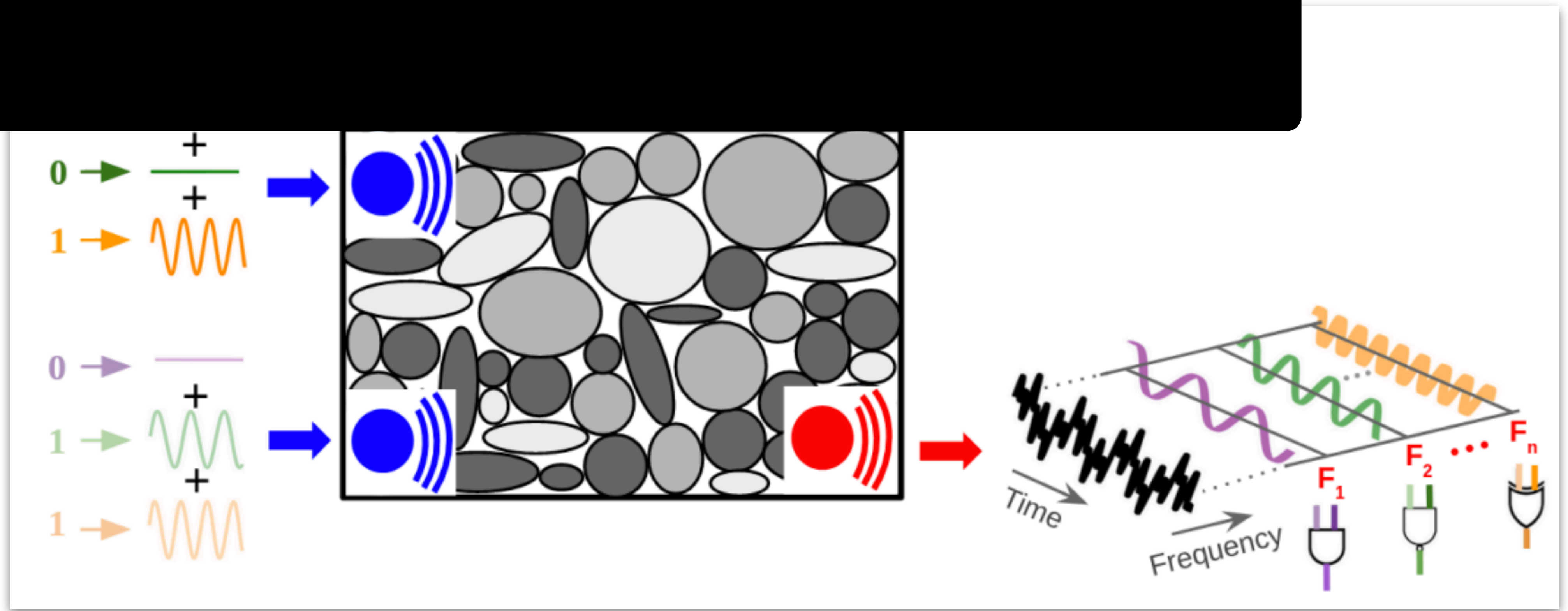


Hardware is growing more diverse; more compilers are desperately needed!

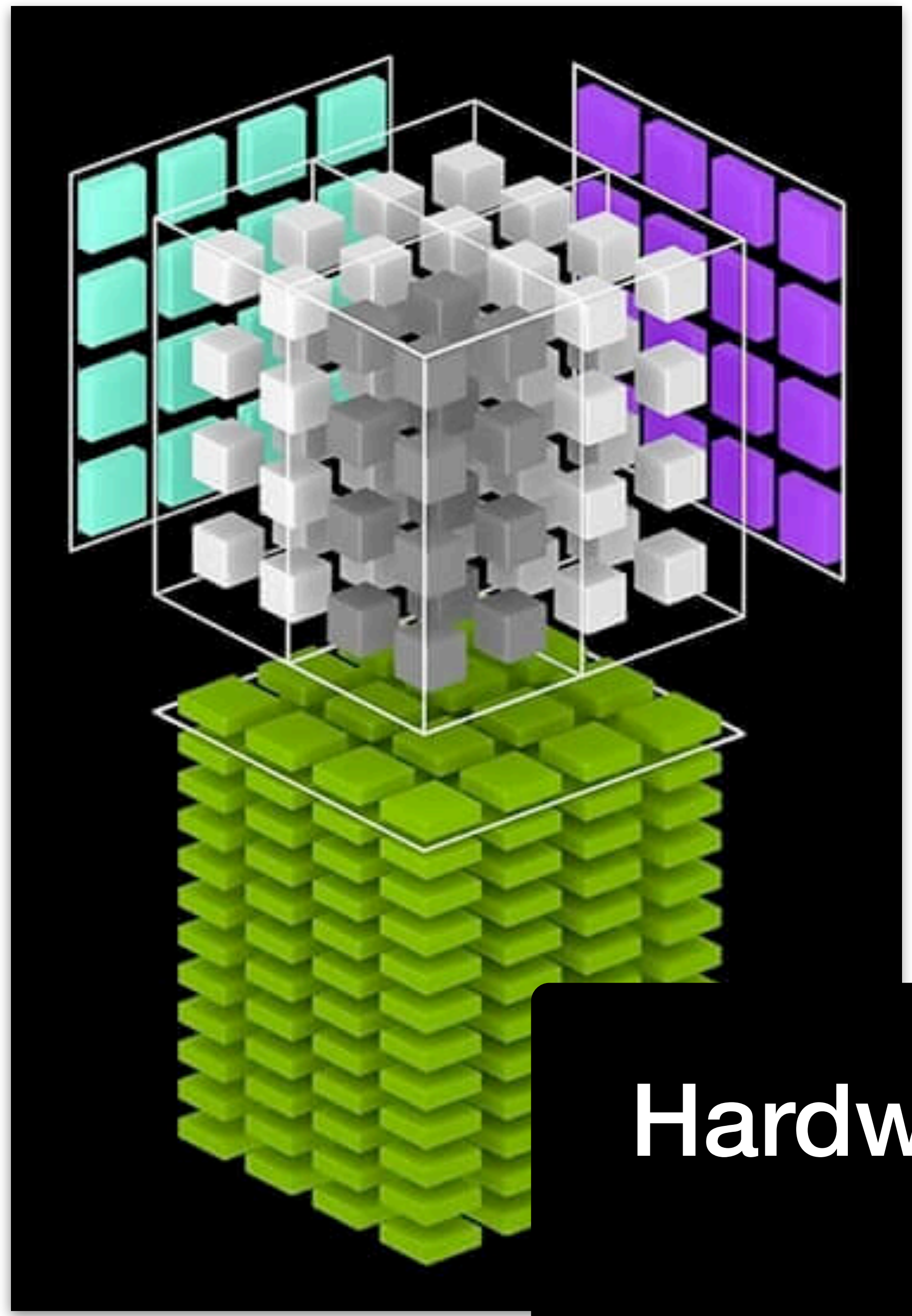
How could we possibly support all of this hardware?



Gerasimova et al. *Connectable DNA Logic Gates: OR and XOR Logics.*

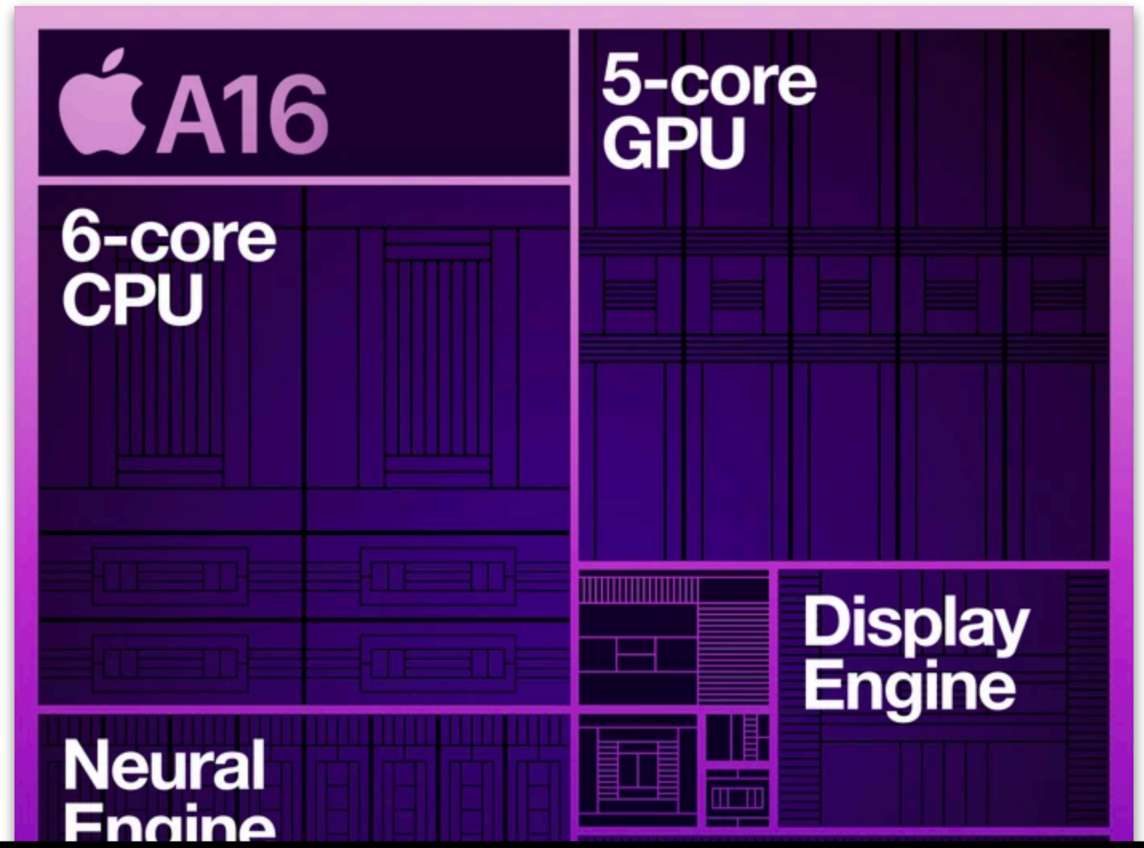


Parsa et al. *Universal Mechanical Polycomputation in Granular Matter.*



NVIDIA Tensc

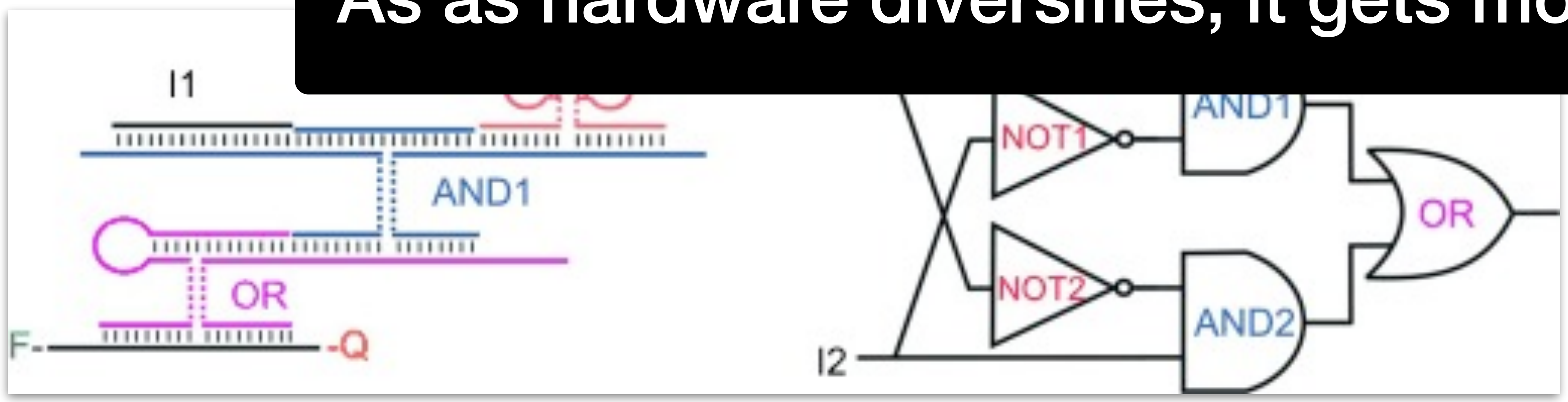
Google TPU



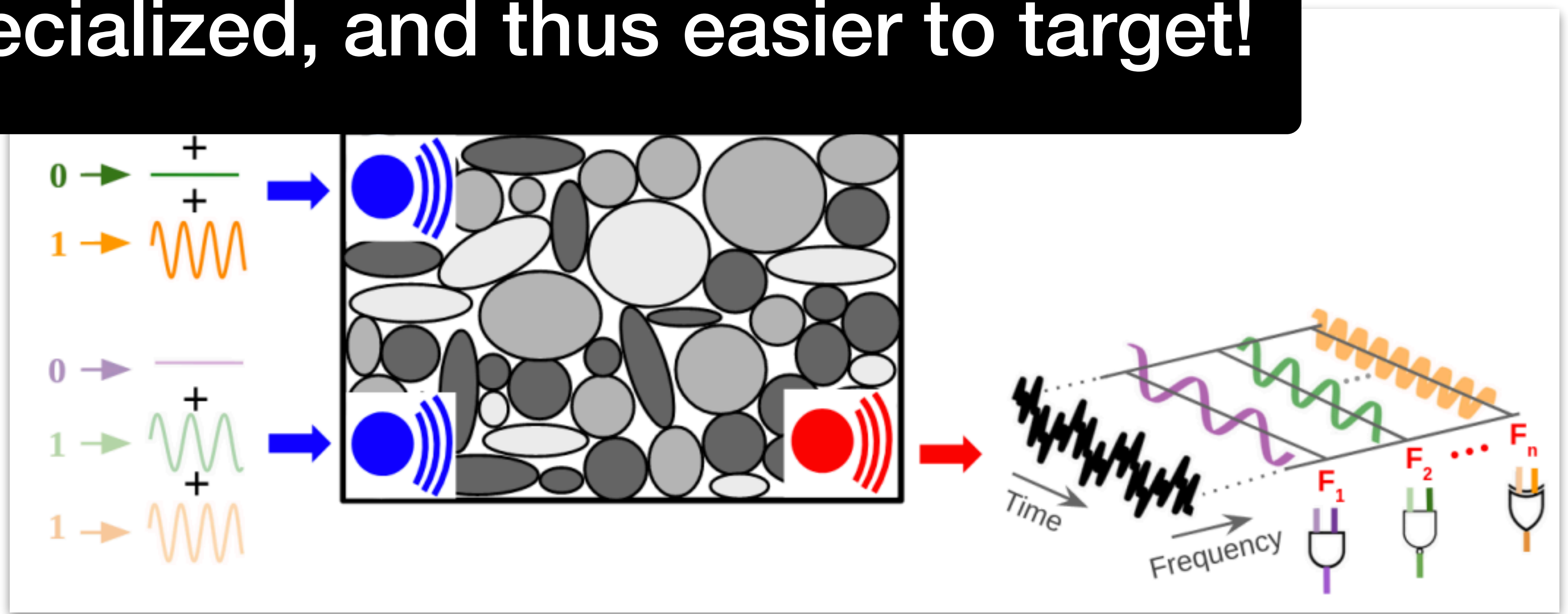
Hardware is growing more diverse; more compilers are desperately needed!

How could we possibly support all of this hardware?

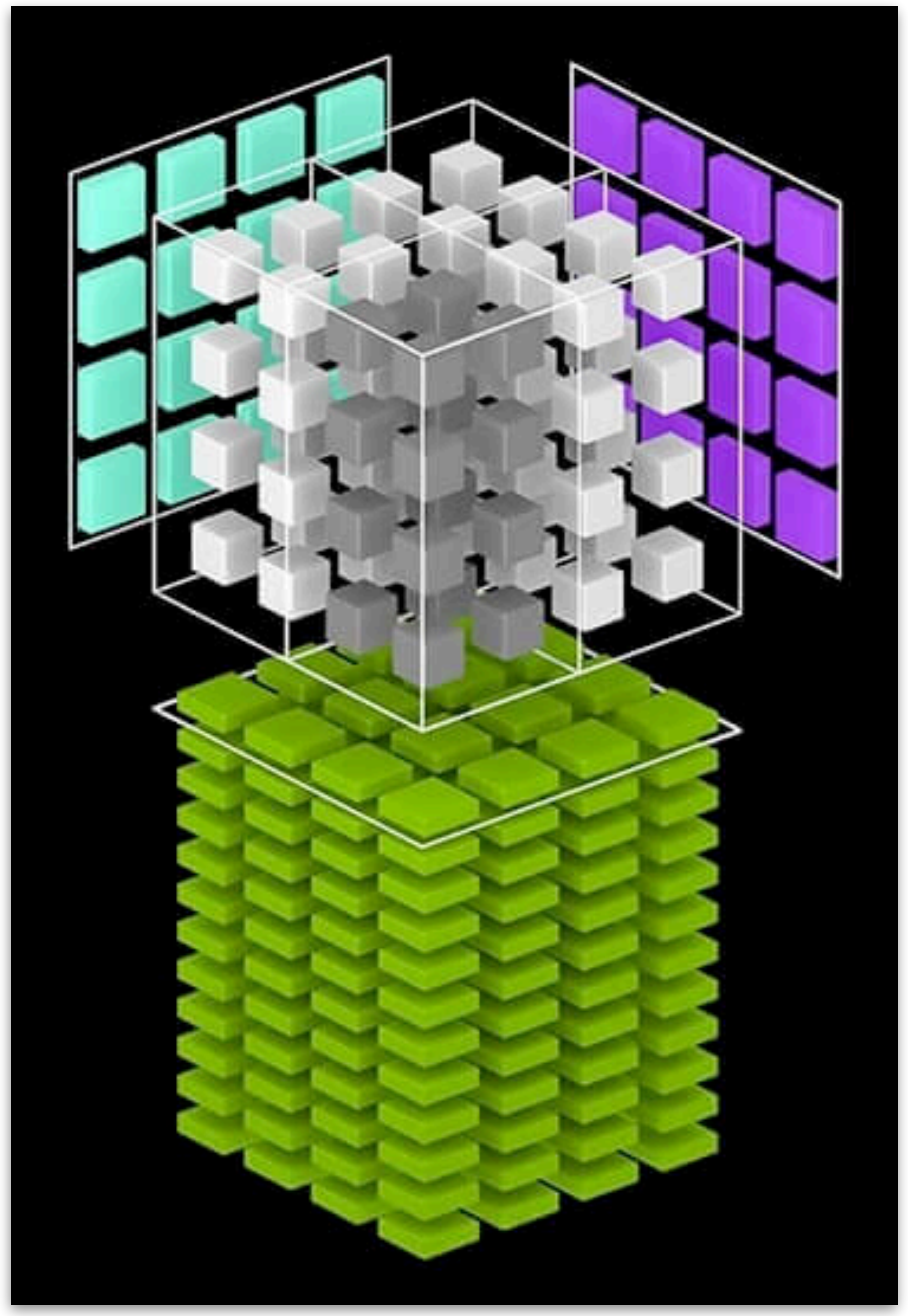
As as hardware diversifies, it gets more specialized, and thus easier to target!



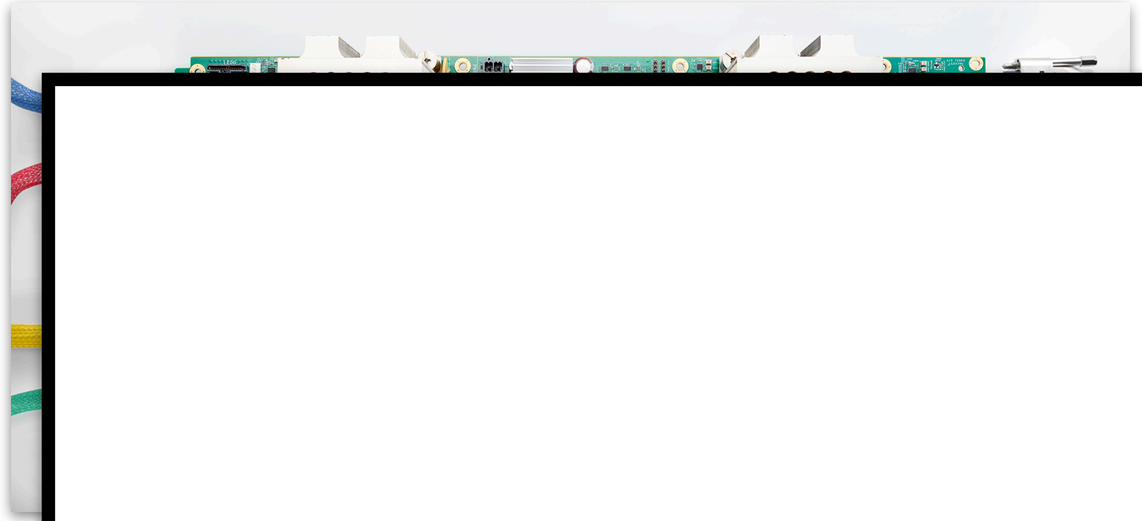
Gerasimova et al. *Connectable DNA Logic Gates: OR and XOR Logics.*



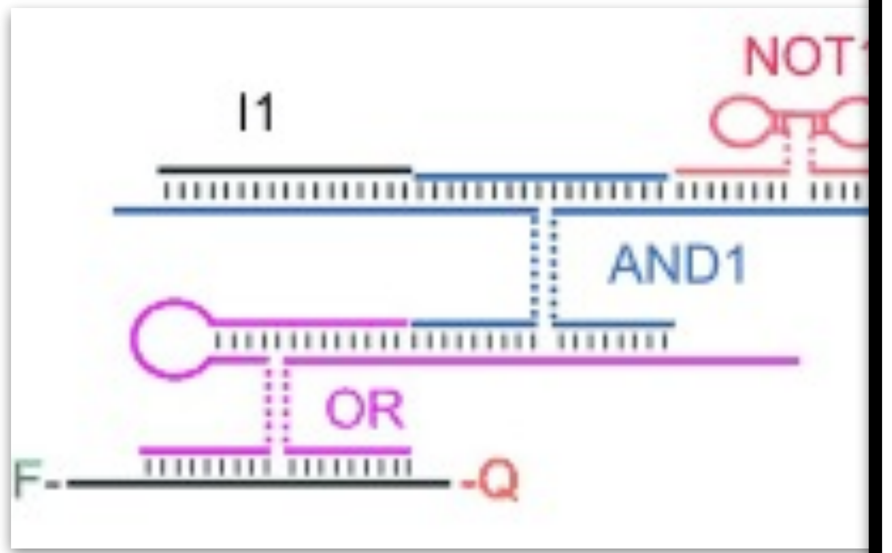
Parsa et al. *Universal Mechanical Polycomputation in Granular Matter.*



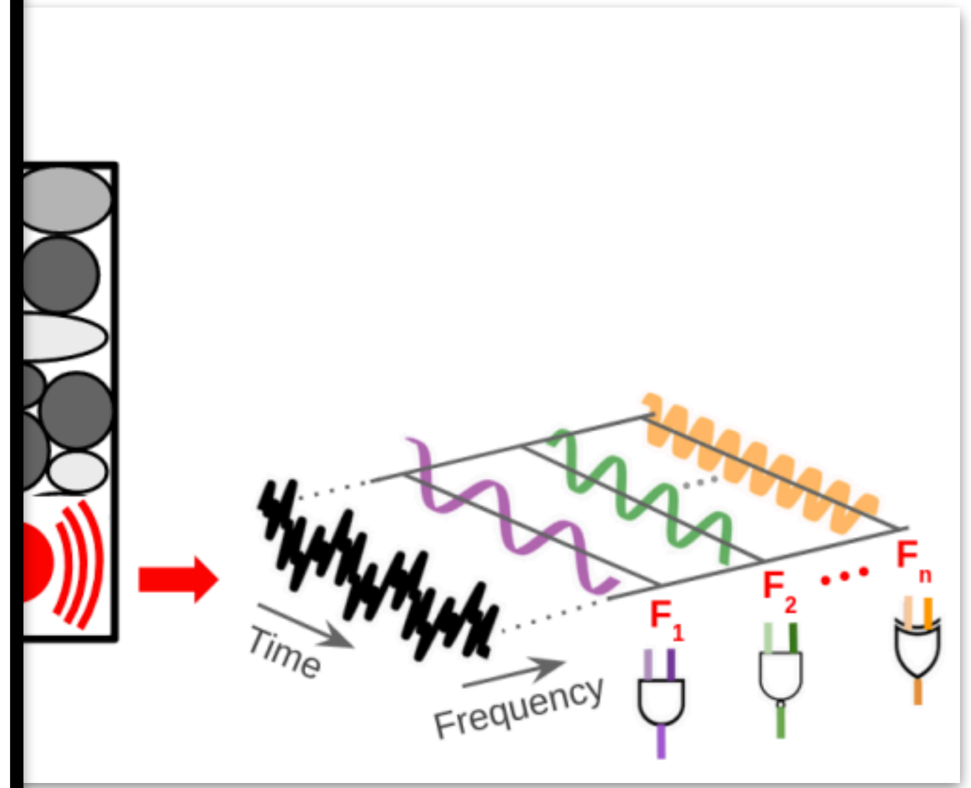
NVIDIA Tensor Cores



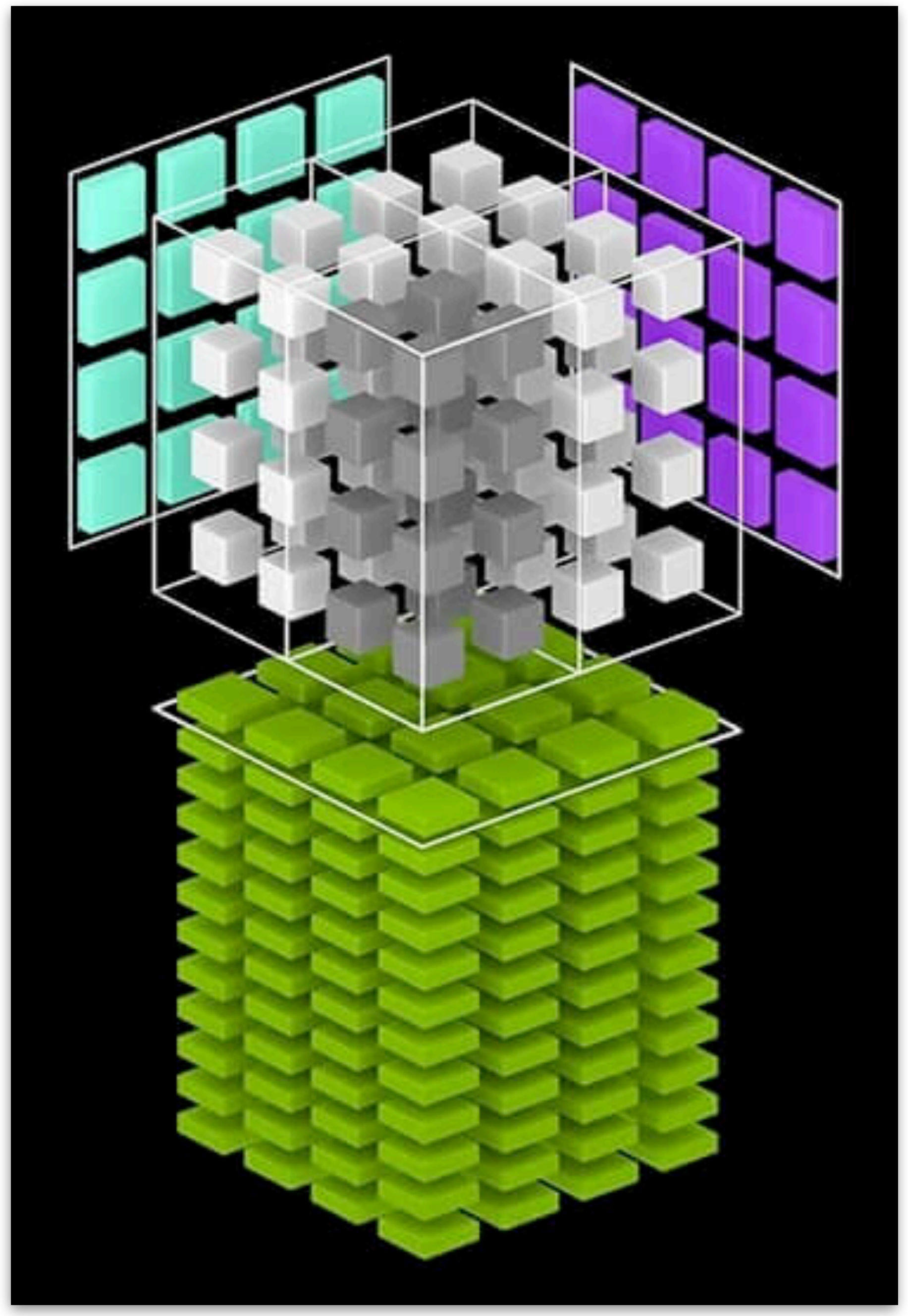
Xilinx Zynq



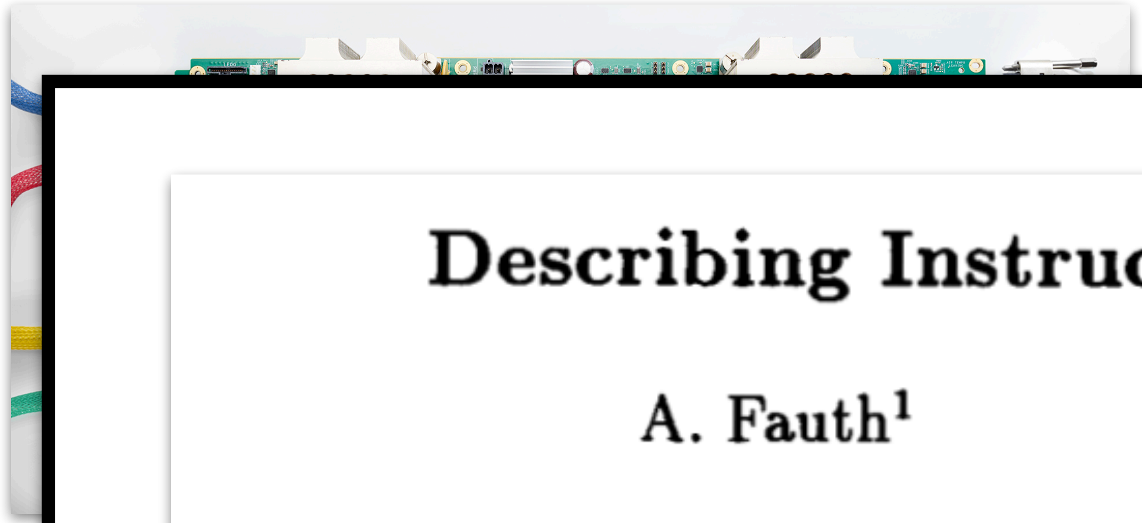
Gerasimova et al. *Connecta*



Parsa et al. *Universal Mechanical Polycomputation in Granular Matter.*



NVIDIA Tensor Cores



Describing Instruction Set Processors Using nML

A. Fauth[1] J. Van Praet[2] M. Freericks[1]

[1]Institut für Technische Informatik
Tech. Univ. Berlin, Franklinstr. 28/29
D–10587 Berlin, Germany

[2]IMEC
Kapeldreef 75
B–3001 Leuven, Belgium

1995

Retargetable Generation of Code Selectors from HDL Processor Models

Rainer Leupers, Peter Marwedel
University of Dortmund, Dept. of Computer Science 12, 44221 Dortmund, Germany
email: leupers|marwedel@ls12.informatik.uni-dortmund.de

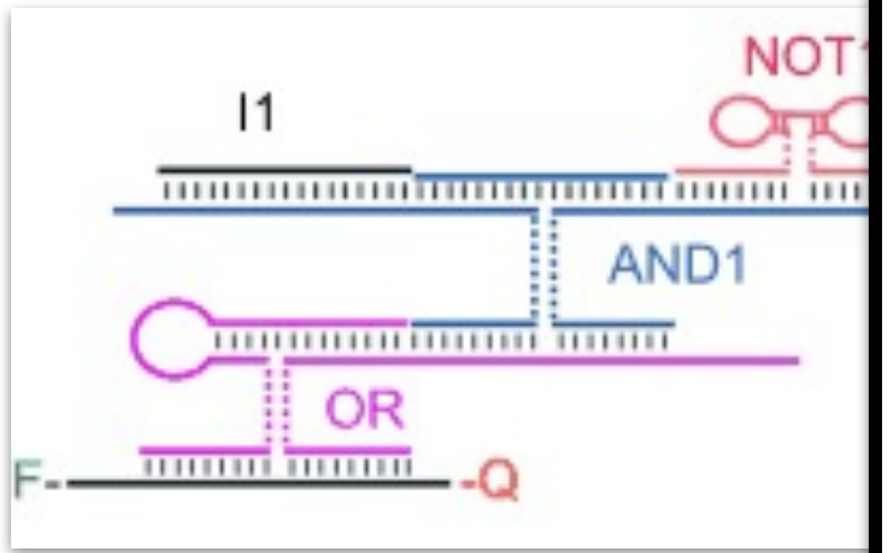
1997

Automatic Tool Generation from Structural Processor Descriptions

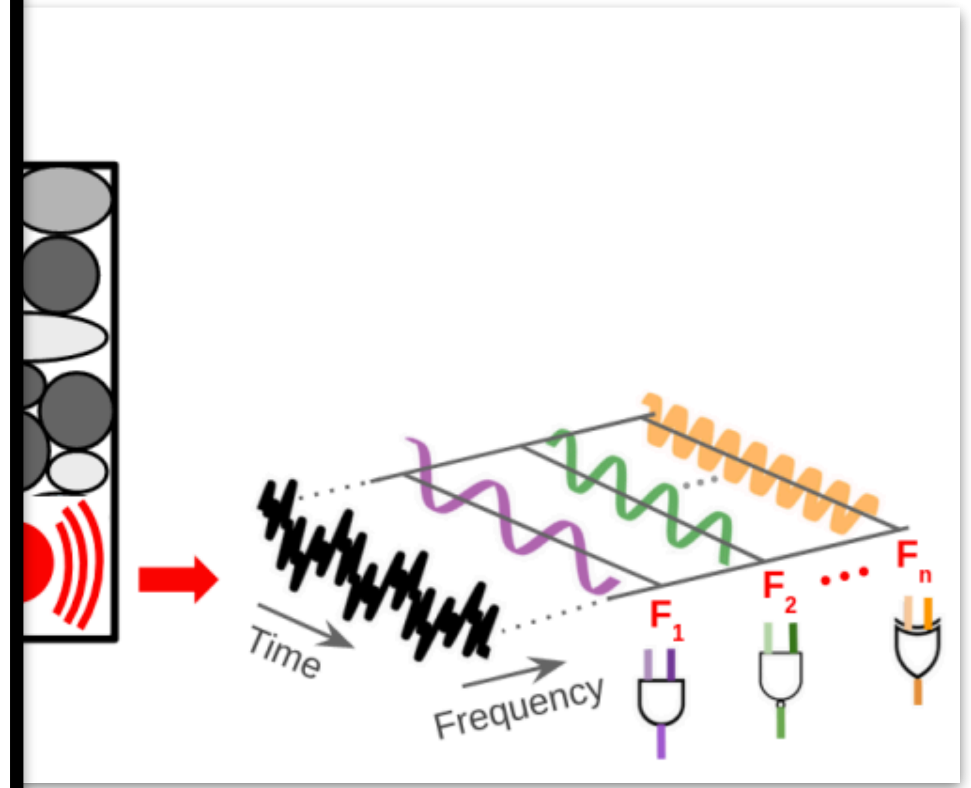
Florian Brandner

2009

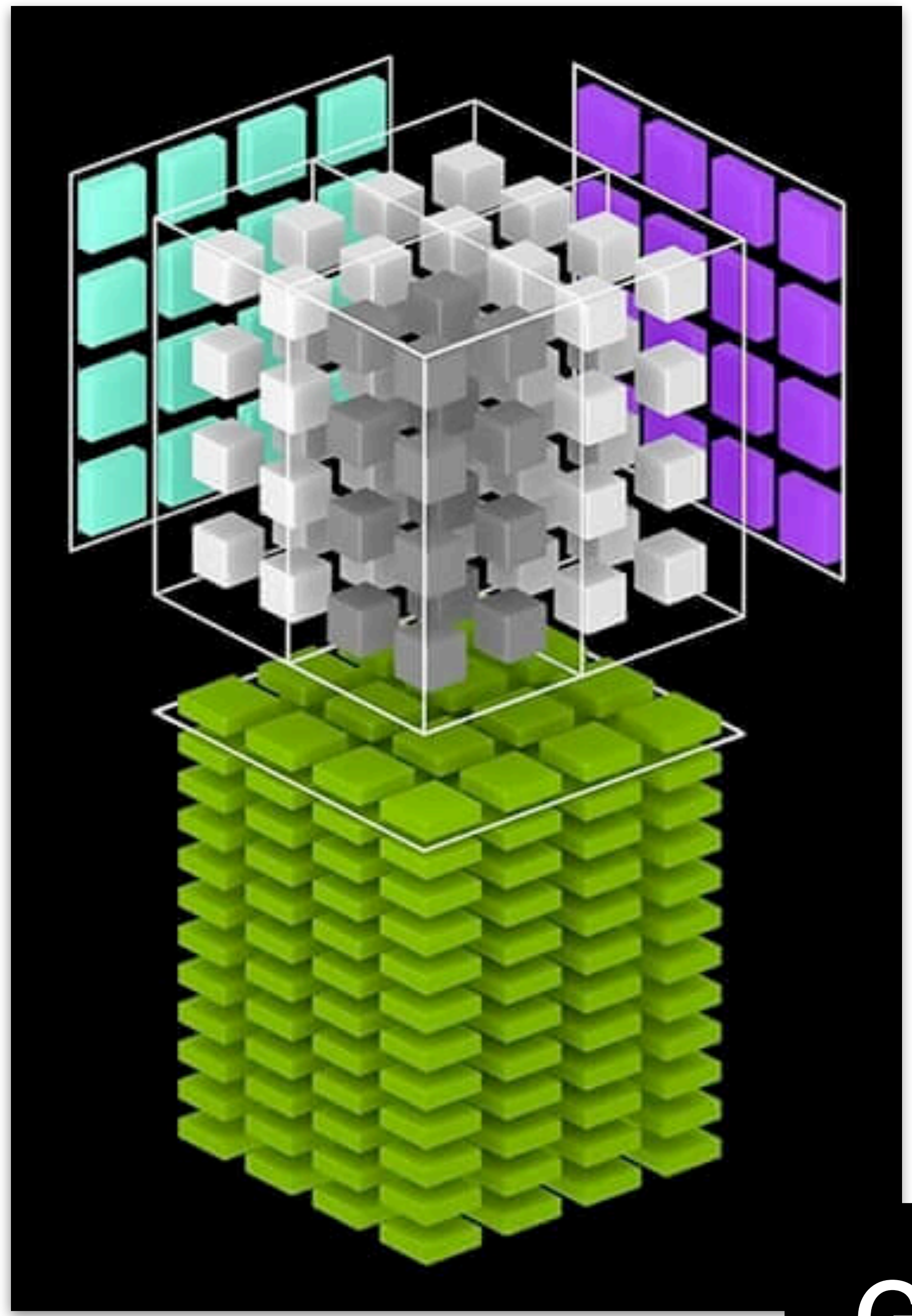
Xilinx Zynq



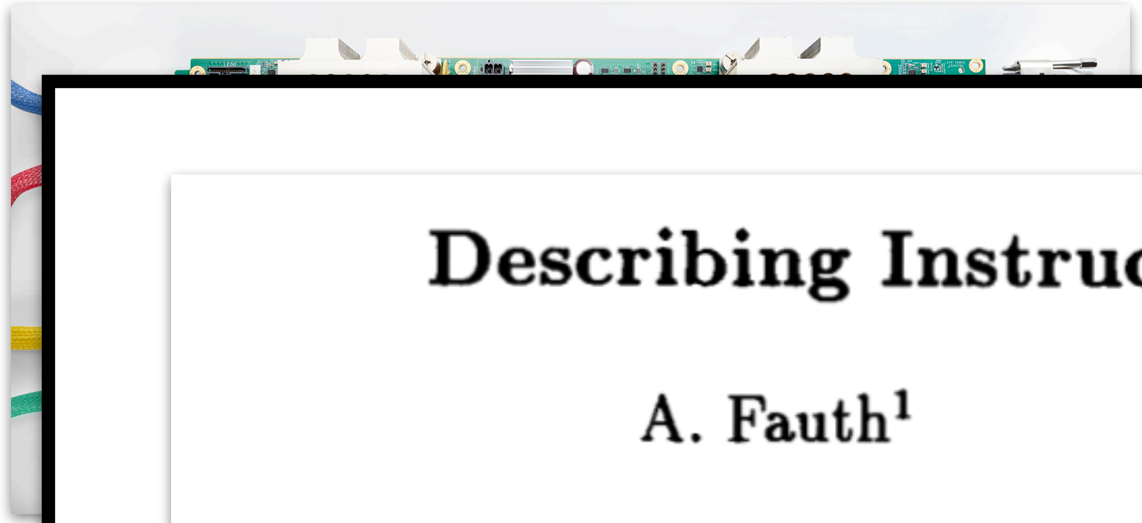
Gerasimova et al. *Connecta*



Parsa et al. *Universal Mechanical Polycomputation in Granular Matter.*



NVIDIA Tensor Cores



Describing Instruction Set Processors Using nML

A. Fauth[1] J. Van Praet[2] M. Freericks[1]

[1]Institut für Technische Informatik
Tech. Univ. Berlin, Franklinstr. 28/29
D–10587 Berlin, Germany

[2]IMEC
Kapeldreef 75
B–3001 Leuven, Belgium

1995

Retargetable Generation of Code Selectors
from HDL Processor Models

Generating compilers for general-purpose hardware is difficult.

University of Dortmund, Dept. of Computer Science 12, 44221 Dortmund, Germany
email: leupers|marwedel@ls12.informatik.uni-dortmund.de

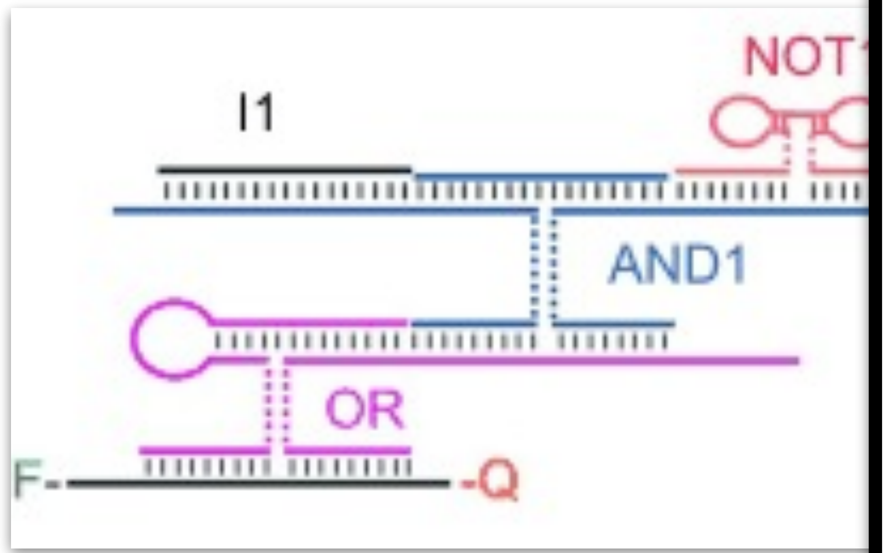
1997

Automatic Tool Generation from Structural Processor Descriptions

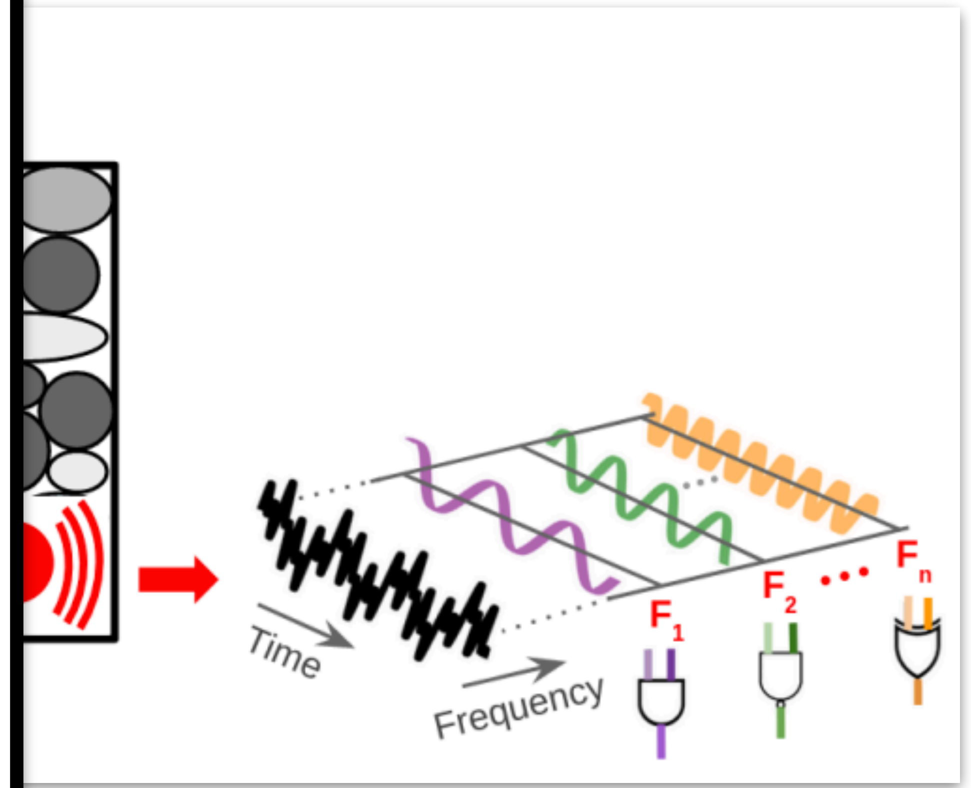
Florian Brandner

2009

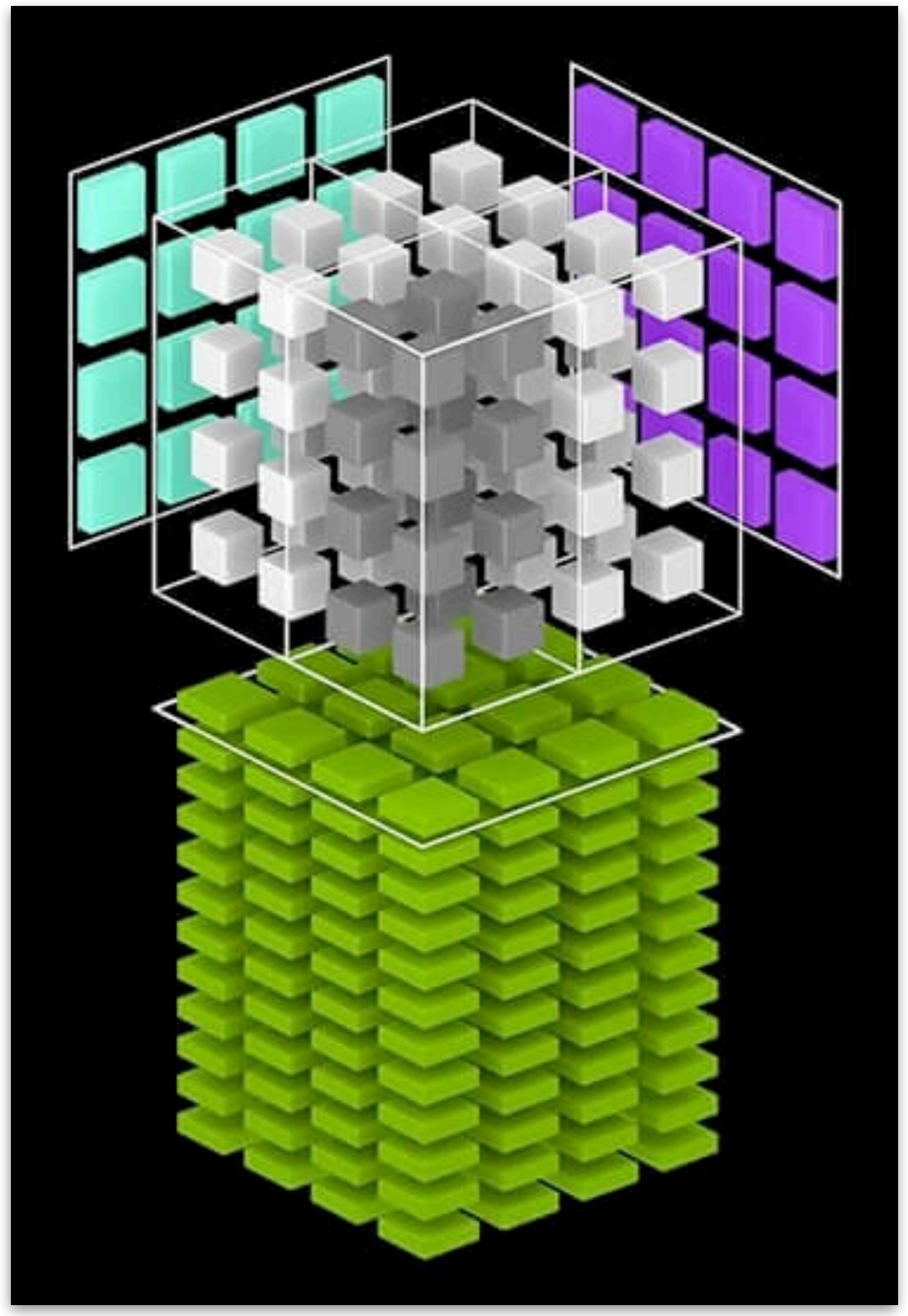
Xilinx Zynq



Gerasimova et al. *Connecta*

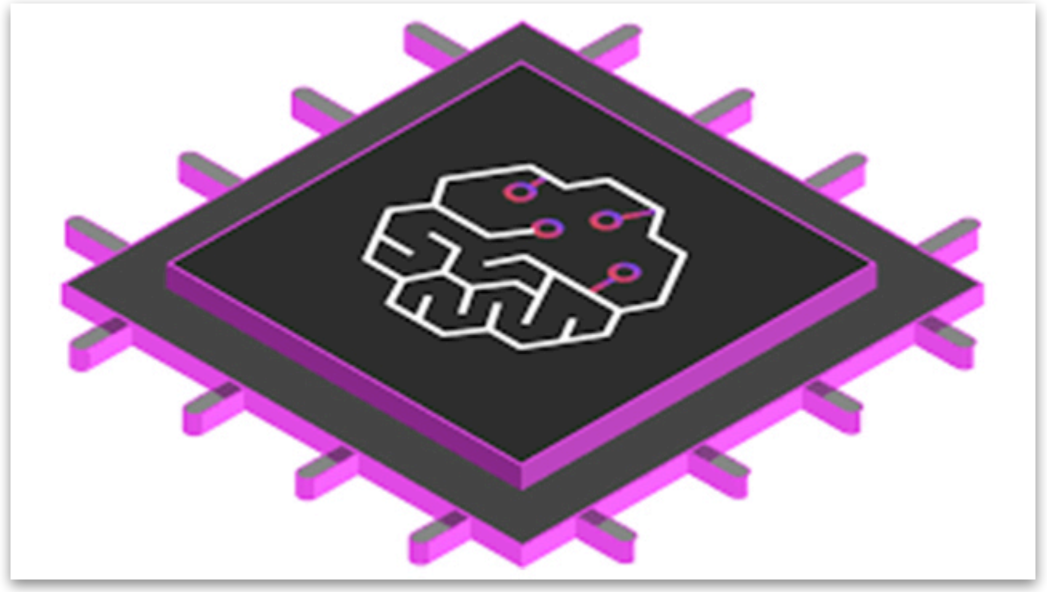


Parsa et al. *Universal Mechanical Polycomputation in Granular Matter.*

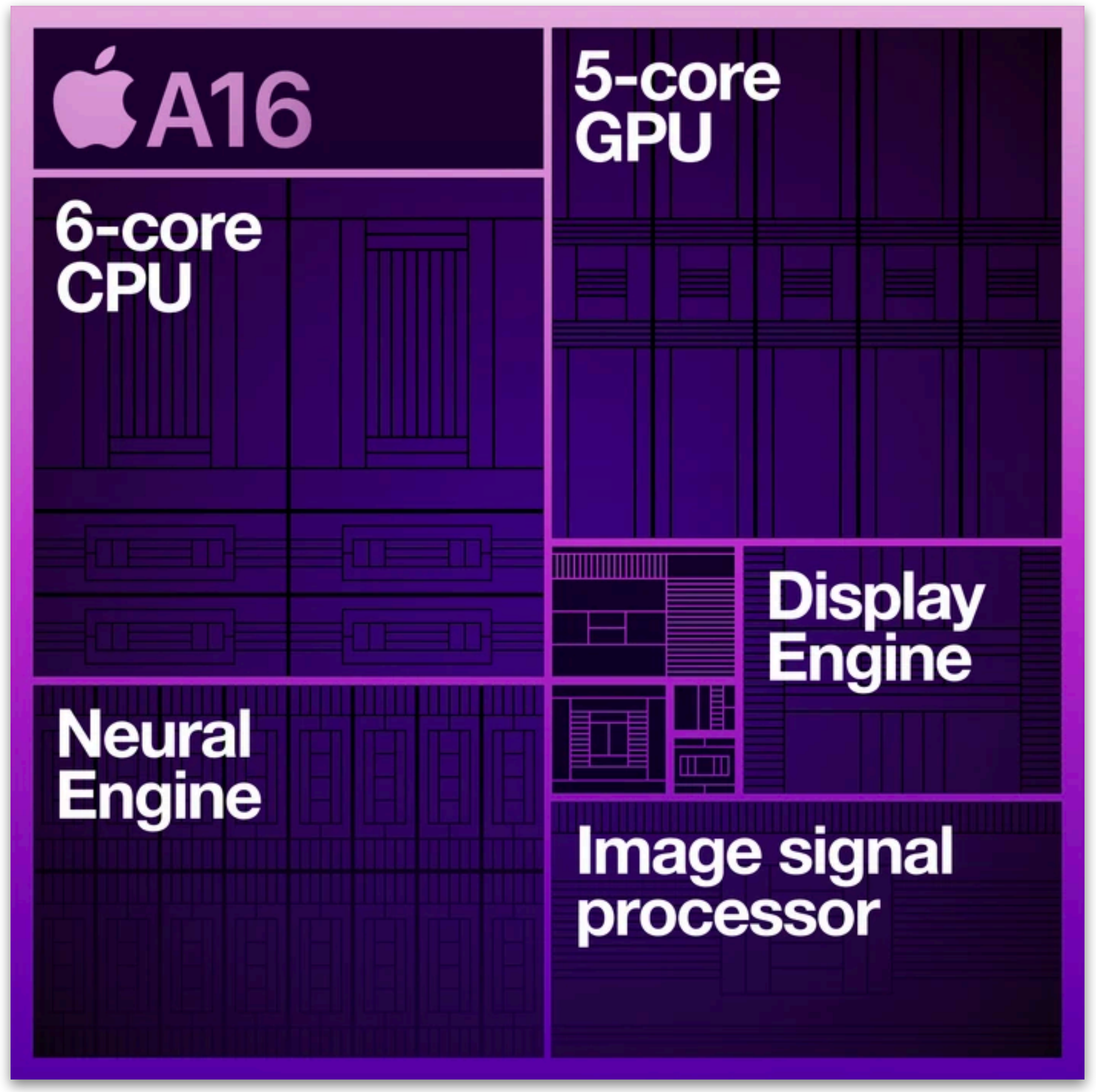


NVIDIA Tensor Cores

Google TPU

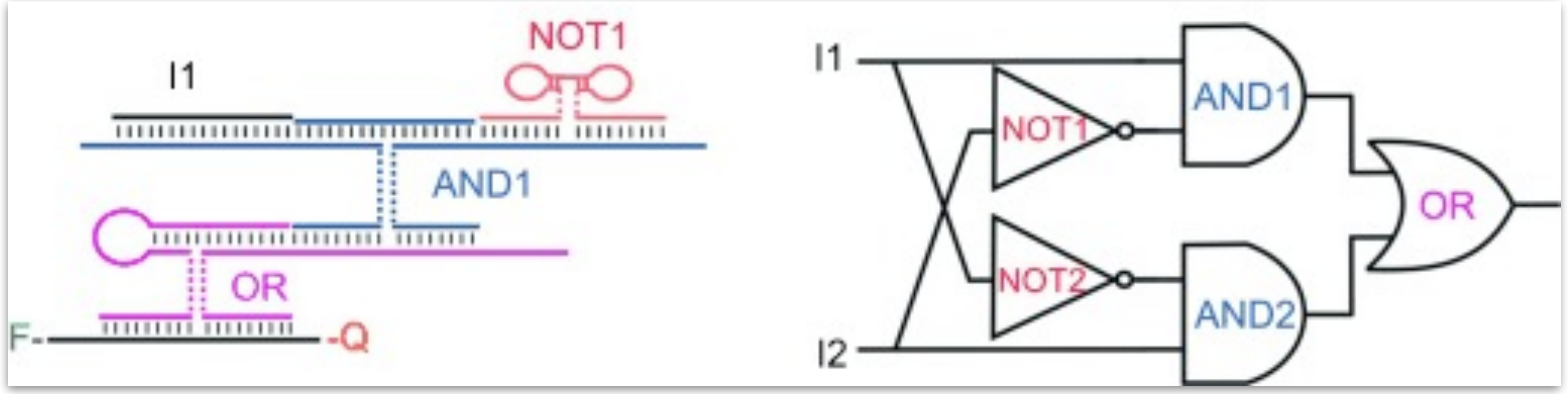


AWS Inferentia

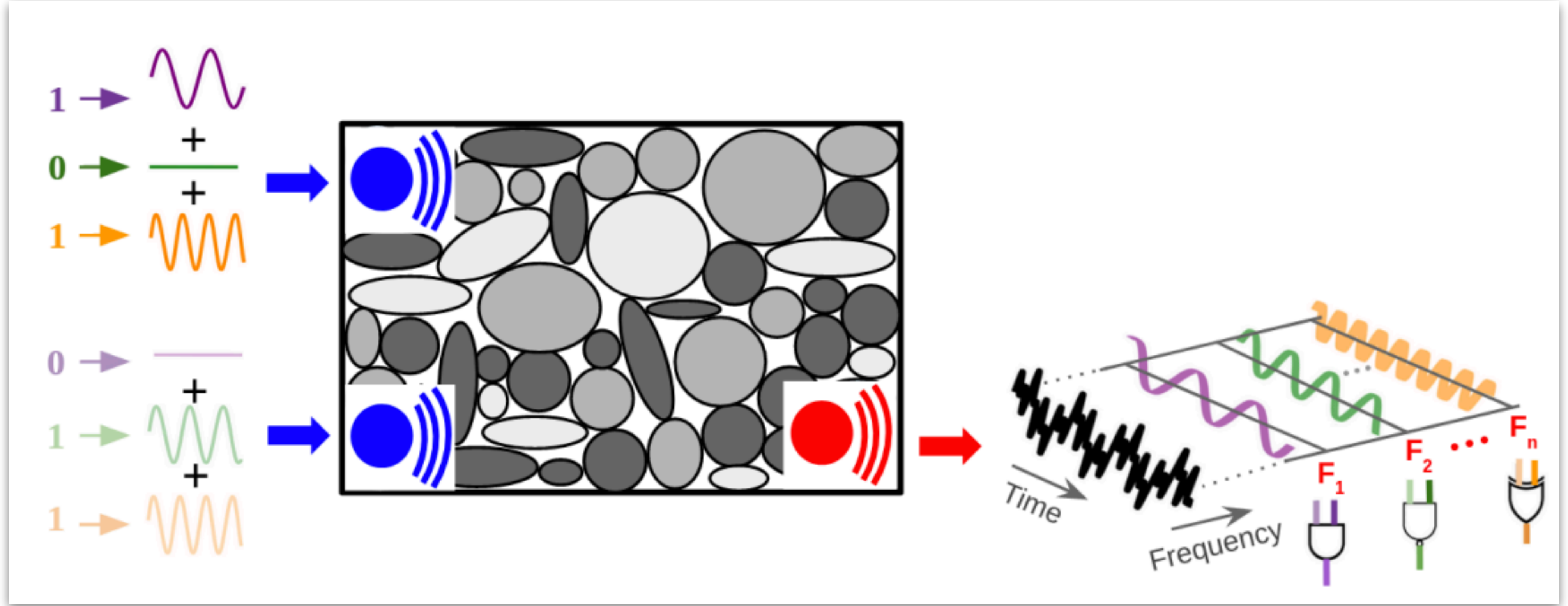


Apple A16

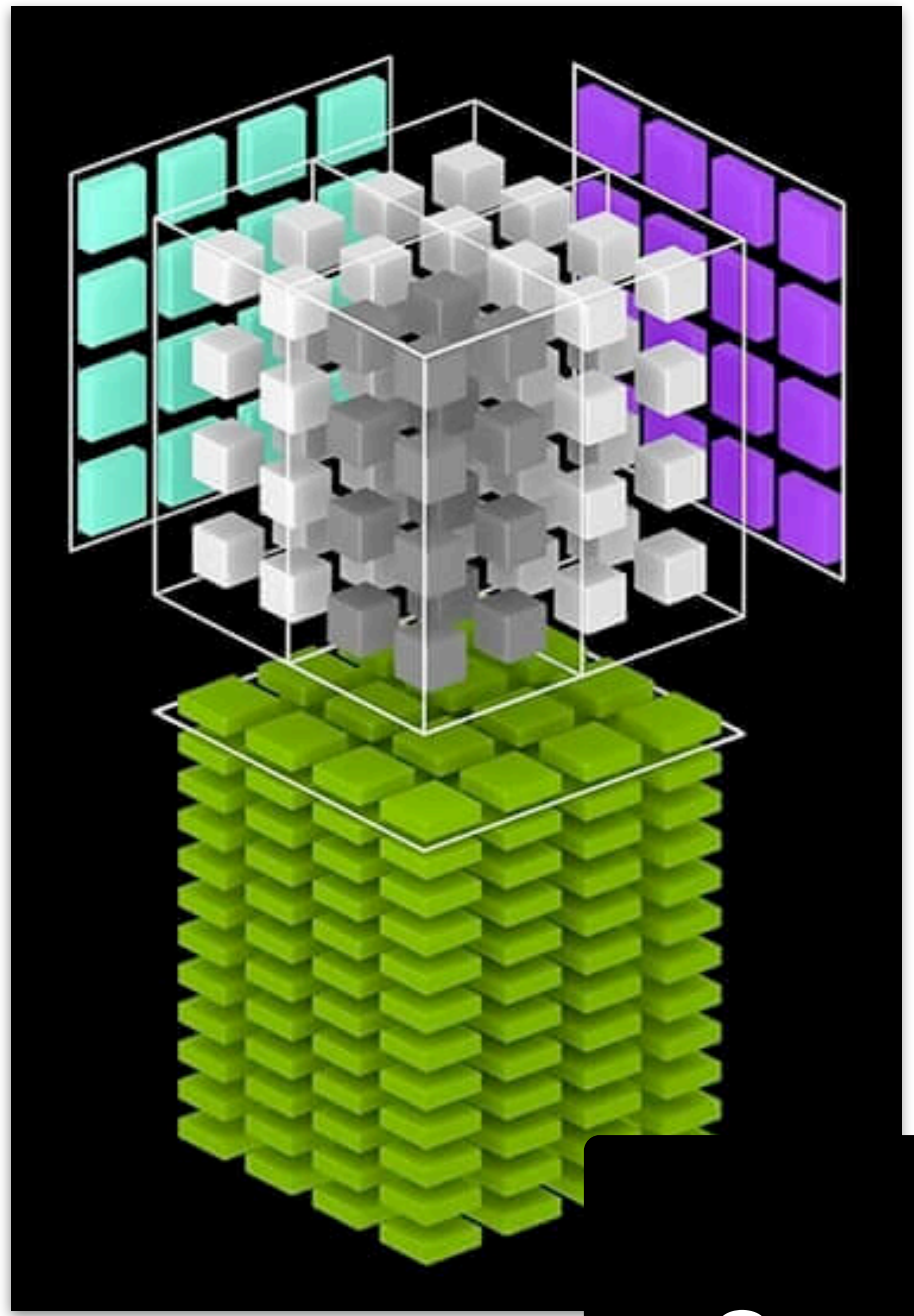
Xilinx Zynq



Gerasimova et al. *Connectable DNA Logic Gates: OR and XOR Logics.*

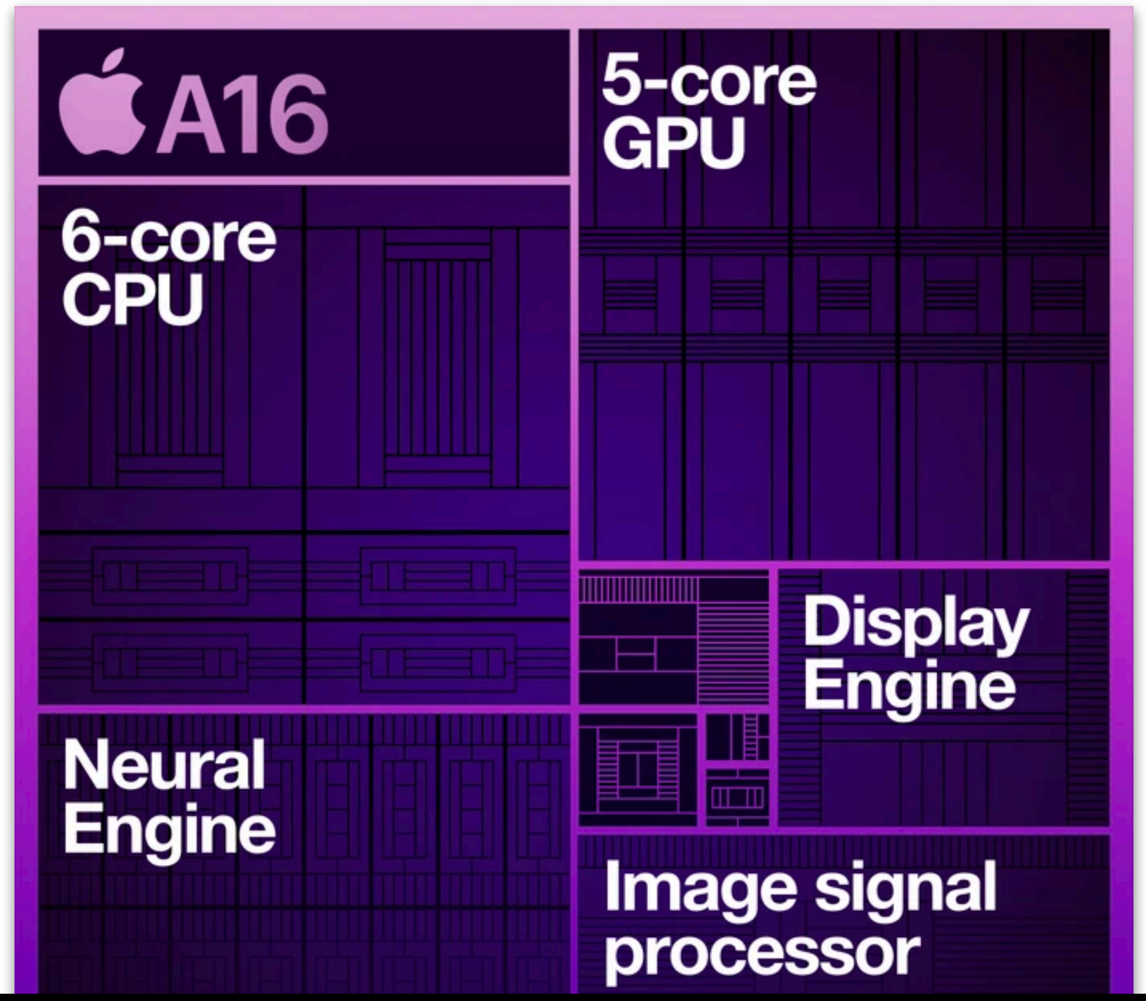


Parsa et al. *Universal Mechanical Polycomputation in Granular Matter.*



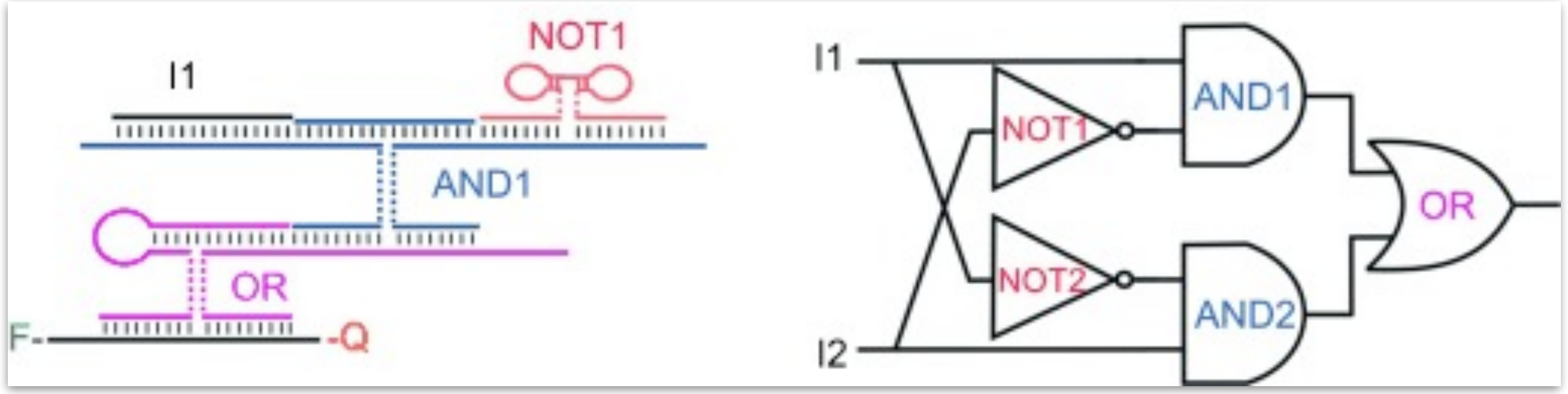
NVIDIA Tensor

Google TPU

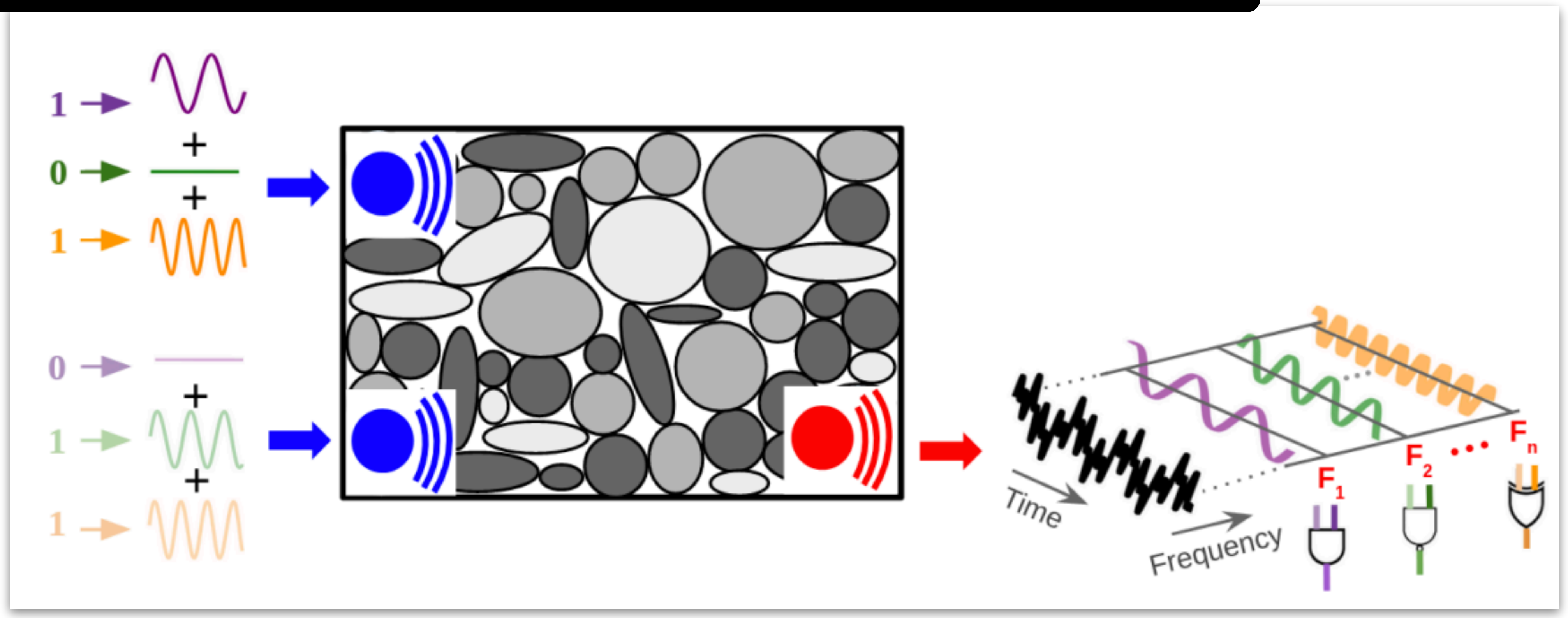


nq

Specialized hardware is easier to target with automated reasoning tools!

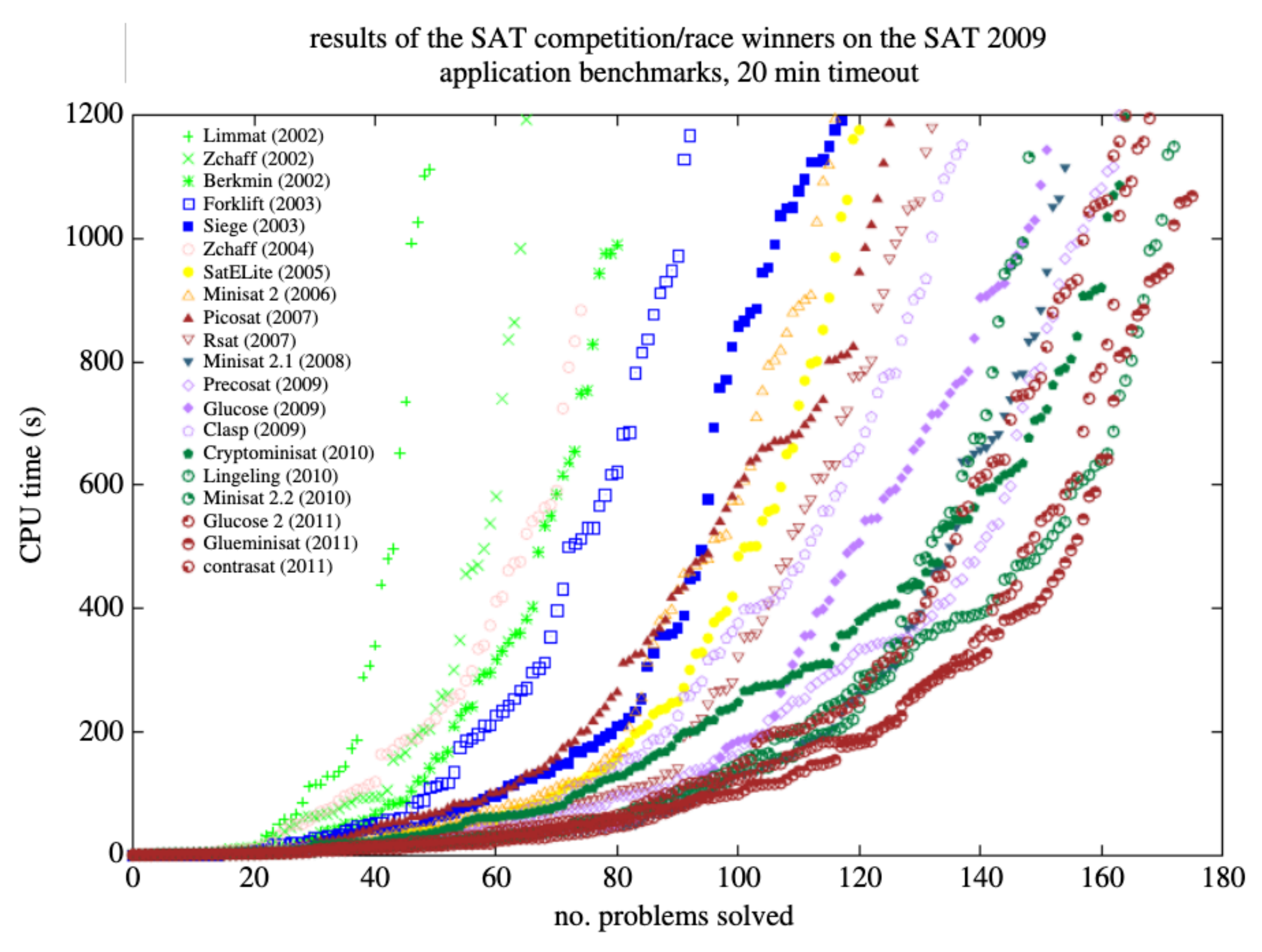


Gerasimova et al. *Connectable DNA Logic Gates: OR and XOR Logics.*



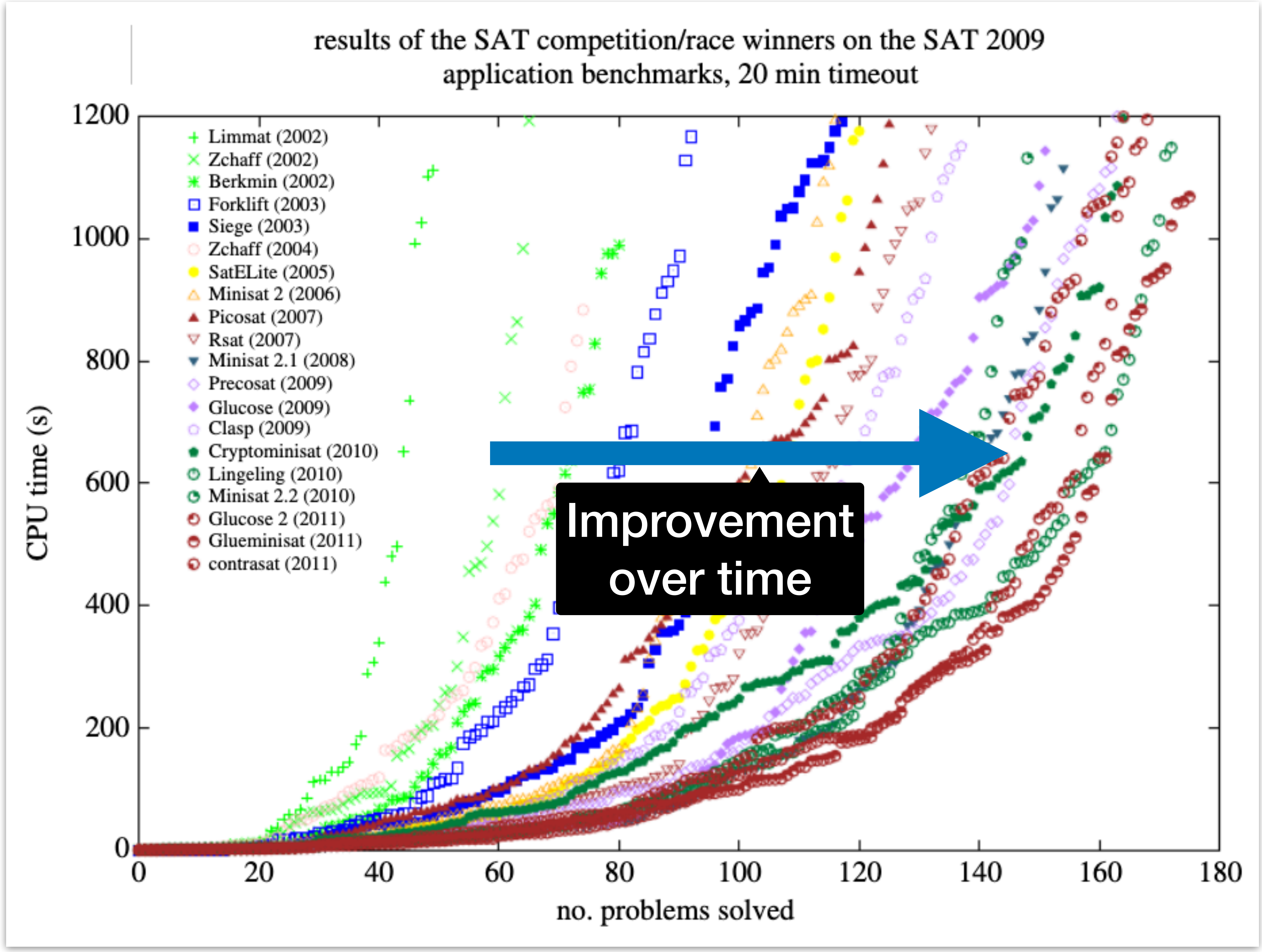
Parsa et al. *Universal Mechanical Polycomputation in Granular Matter.*

SAT/SMT



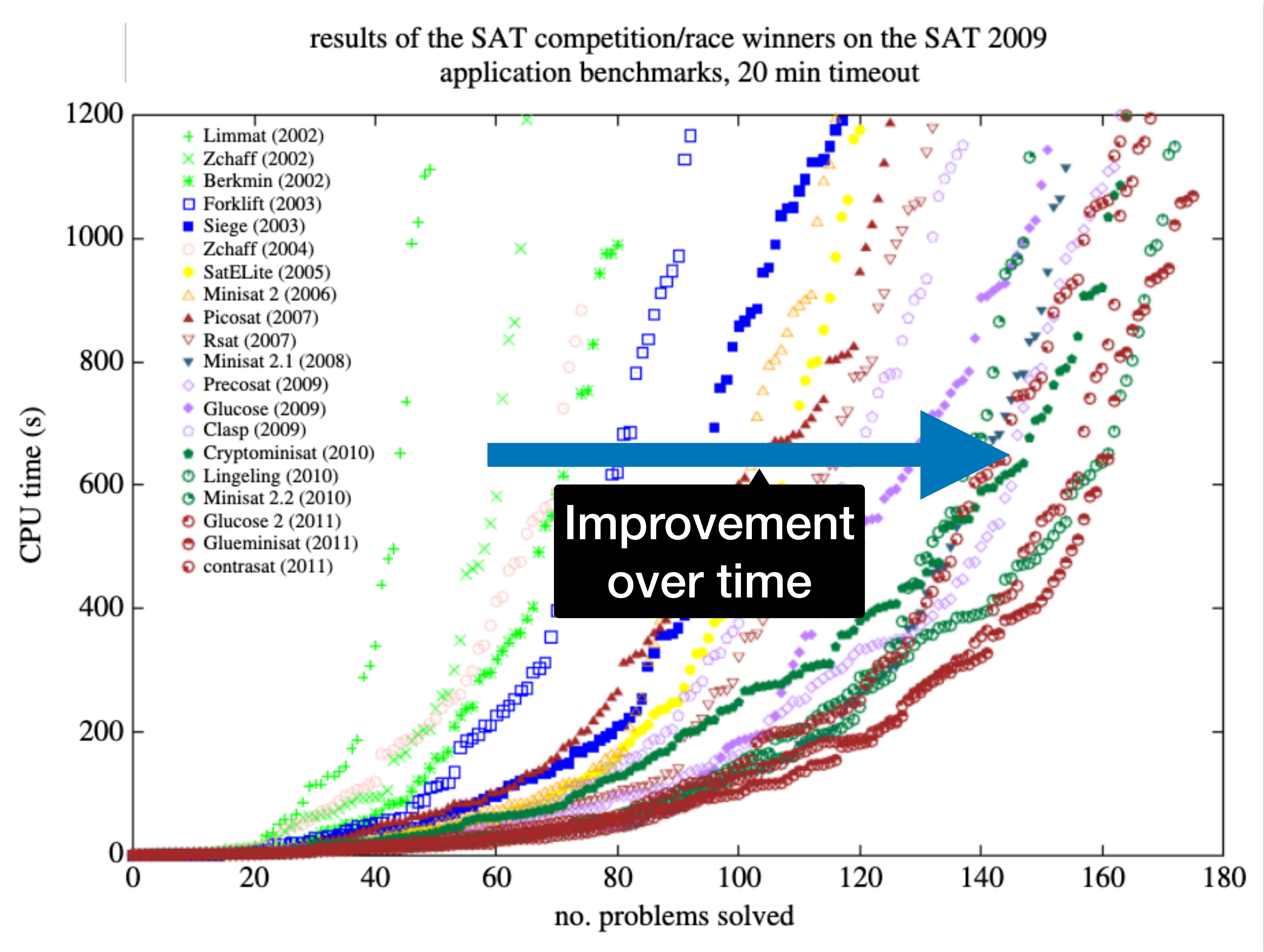
Järvisalo et al. 2012. The international SAT solver competitions.

SAT/SMT



Järvisalo et al. 2012. The international SAT solver competitions.

SAT/SMT



Järvisalo et al. 2012. The international SAT solver competitions.

Equality Saturation

Vectorization for Digital Signal Processors via Equality Saturation

Alexa VanHattum
Cornell University
Ithaca, NY, USA

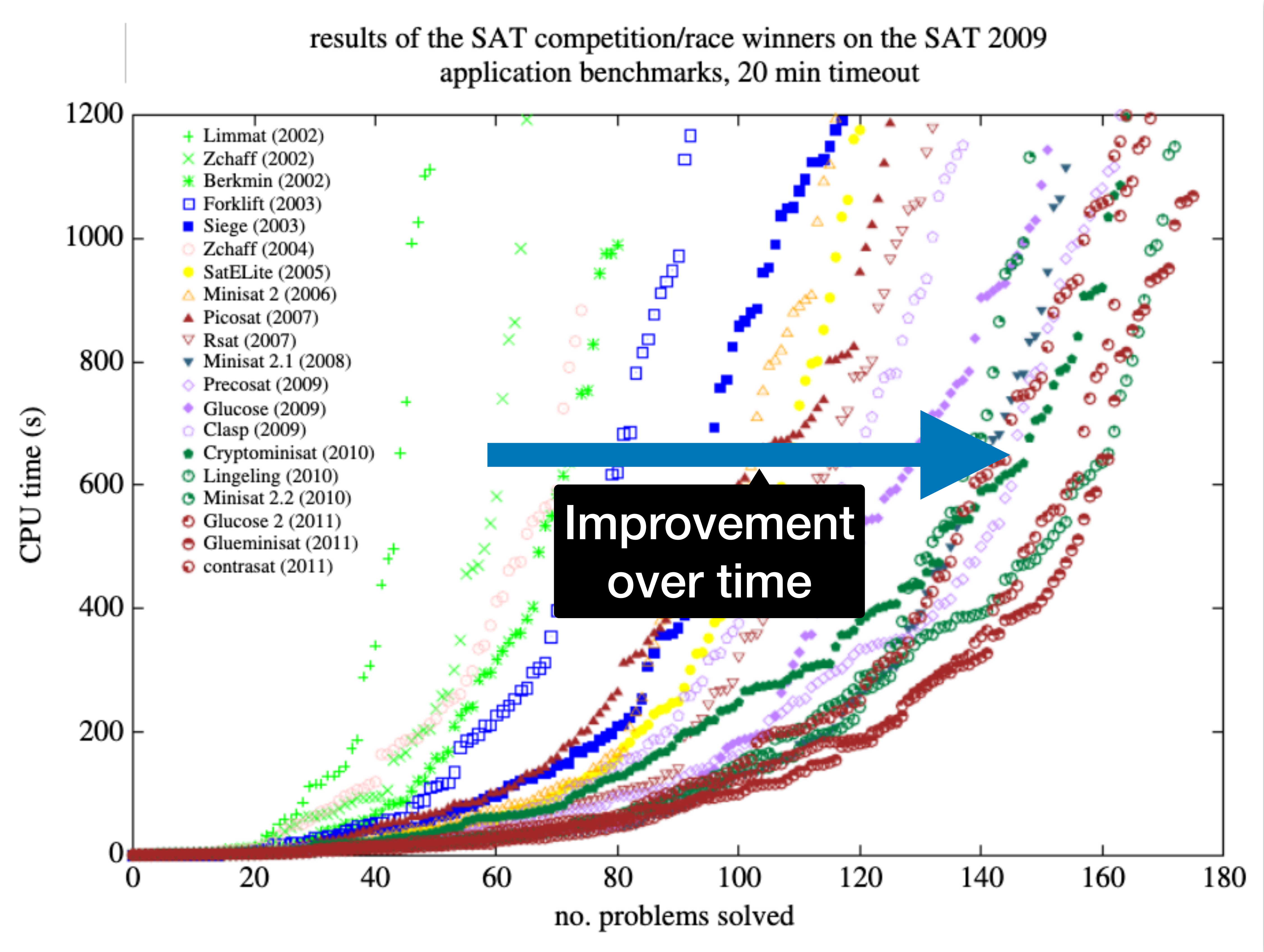
Rachit Nigam
Cornell University
Ithaca, NY, USA

Vincent T. Lee
Facebook Reality Labs Research
Redmond, WA, USA

James Bornholt
The University of Texas at Austin
Austin, TX, USA

Adrian Sampson
Cornell University
Ithaca, NY, USA

SAT/SMT



Järvisalo et al. 2012. The international SAT solver competitions.

Equality Saturation

Vectorization for Digital Signal Processors via Equality Saturation

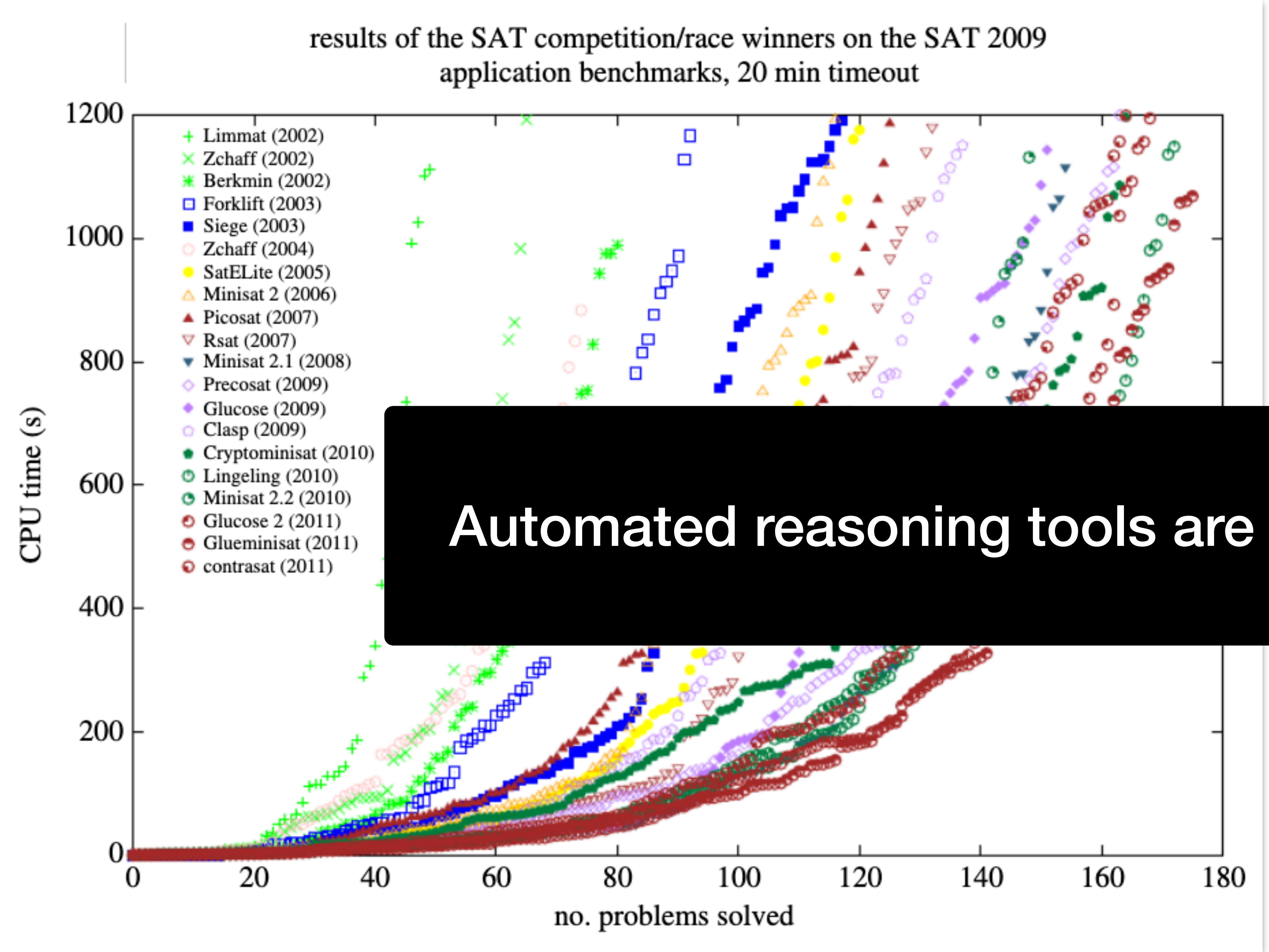
Alexa VanHattum, Cornell University, Ithaca, NY, USA
Rachit Nigam, Cornell University, Ithaca, NY, USA
Vincent T. Lee, Facebook Reality Labs Research, Redmond, WA, USA
James Bornholt, The University of Texas at Austin, Austin, TX, USA
Adrian Sampson, Cornell University, Ithaca, NY, USA

Benchmarking Large Language Models for Automated Verilog RTL Code Generation

Shailja Thakur*, Baleegh Ahmad*, Zhenxing Fan*, Hammond Pearce*,
Benjamin Tan†, Ramesh Karri*, Brendan Dolan-Gavitt*, Siddharth Garg*
*New York University, †University of Calgary

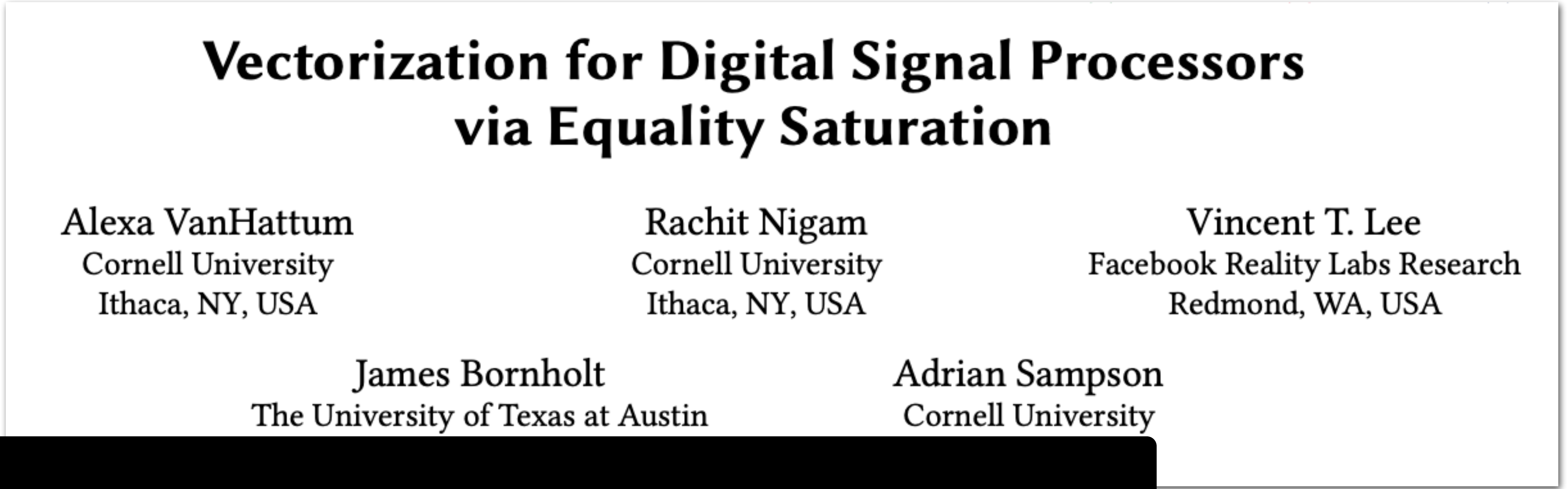
ML/LLMs

SAT/SMT



Järvisalo et al. 2012. The international SAT solver competitions.

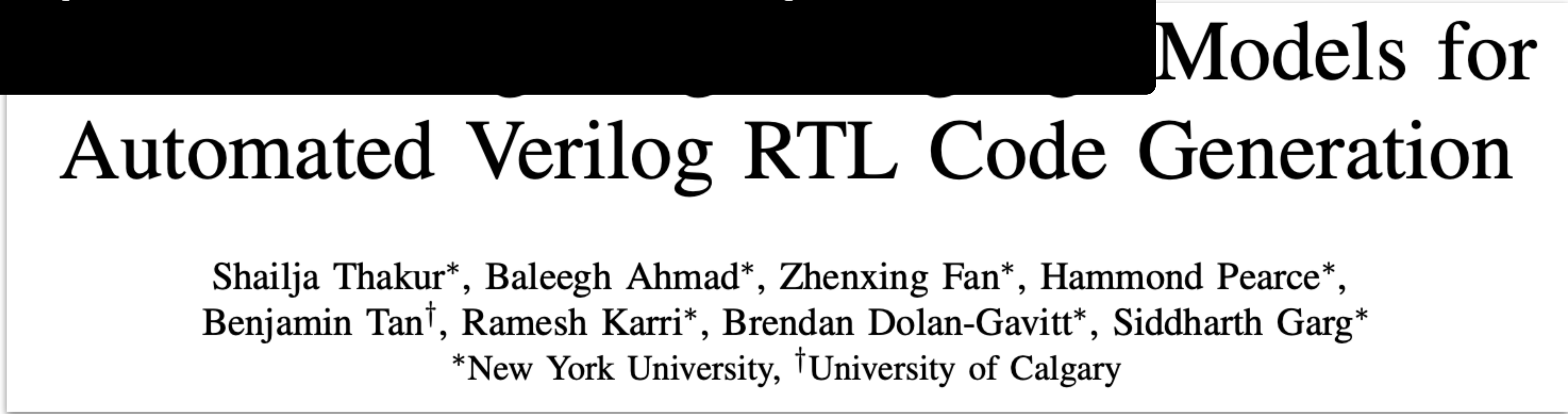
Equality Saturation



Vectorization for Digital Signal Processors via Equality Saturation

Alexa VanHattum, Cornell University, Ithaca, NY, USA
Rachit Nigam, Cornell University, Ithaca, NY, USA
Vincent T. Lee, Facebook Reality Labs Research, Redmond, WA, USA
James Bornholt, The University of Texas at Austin
Adrian Sampson, Cornell University

Automated reasoning tools are ready for the task of compiler generation.



Models for Automated Verilog RTL Code Generation

Shailja Thakur*, Baleegh Ahmad*, Zhenxing Fan*, Hammond Pearce*,
Benjamin Tan†, Ramesh Karri*, Brendan Dolan-Gavitt*, Siddharth Garg*
*New York University, †University of Calgary

ML/LLMs

Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Hardware is diversifying, and we need new compilers.

Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Hardware is diversifying, and we need new compilers.

Modern targets are more amenable to automated methods.

Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Hardware is diversifying, and we need new compilers.

Modern targets are more amenable to automated methods.

Automated reasoning tools are ready for the task of compiler generation.

Generating Compilers → Why Now? → Case Study: Lakeroad → Call to Action

Generating Compilers → Why Now? → **Case Study: Lakeroad** → Call to Action

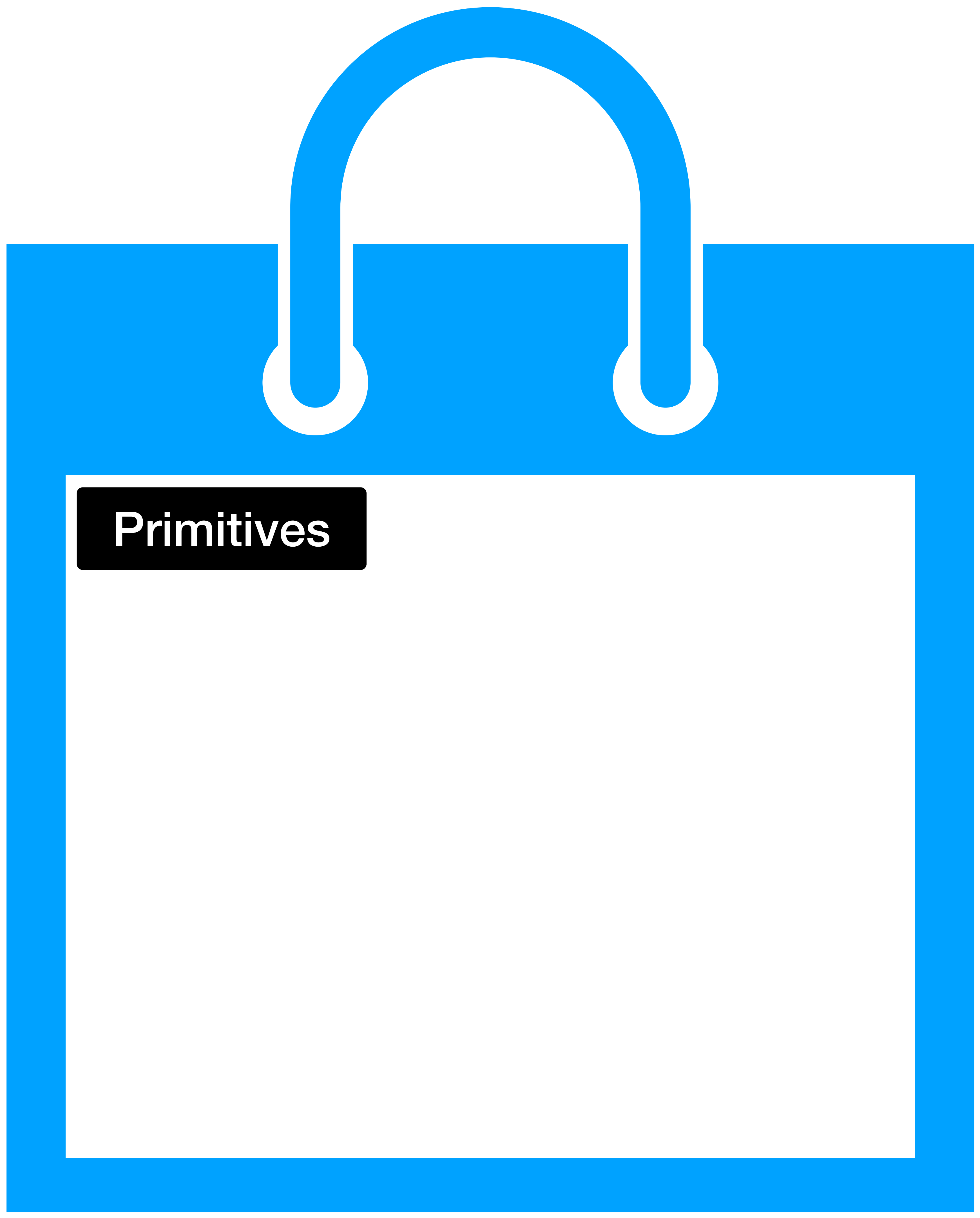
Compilers should be generated from formal models of hardware.

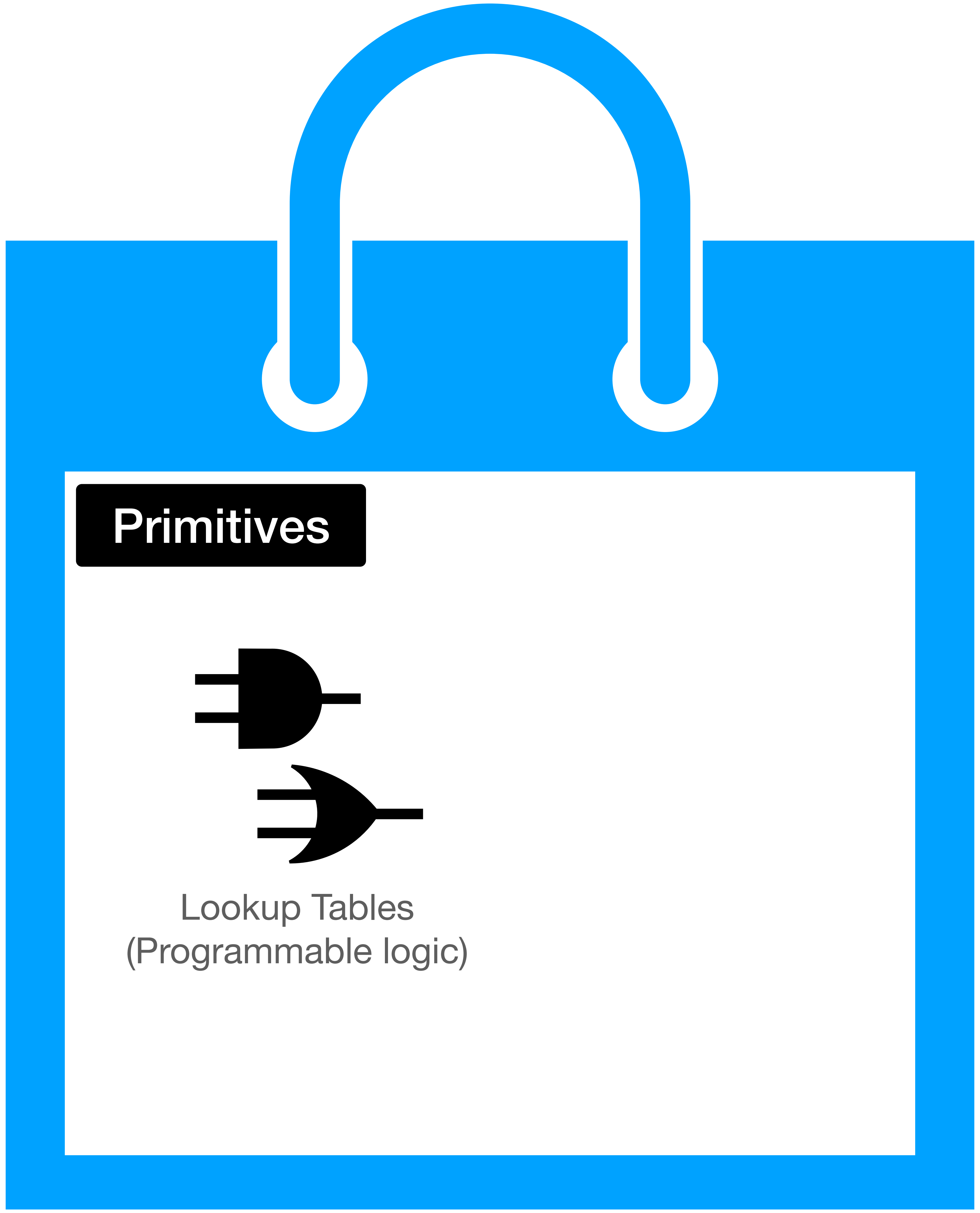
With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

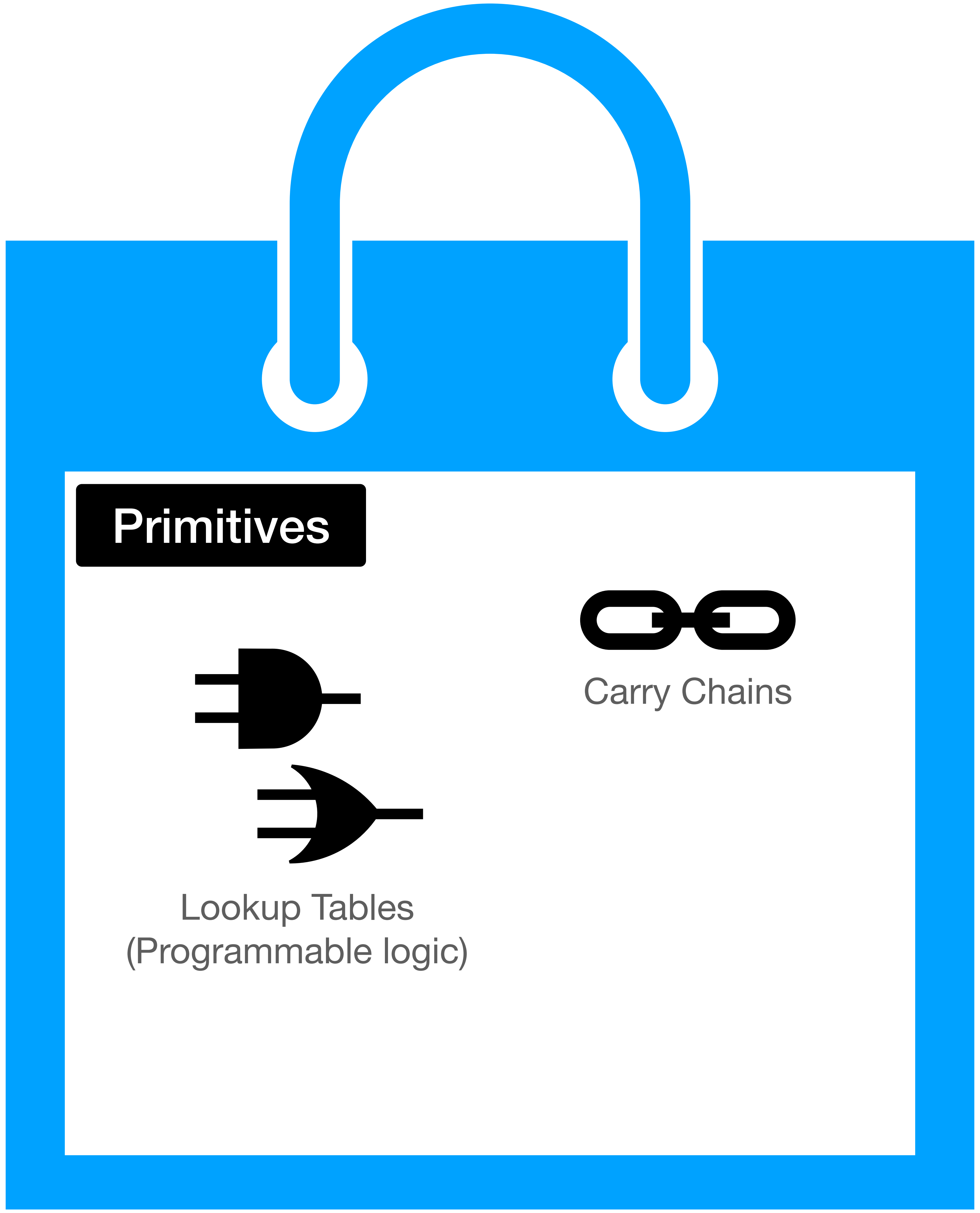
Compilers should be generated from formal models of hardware.

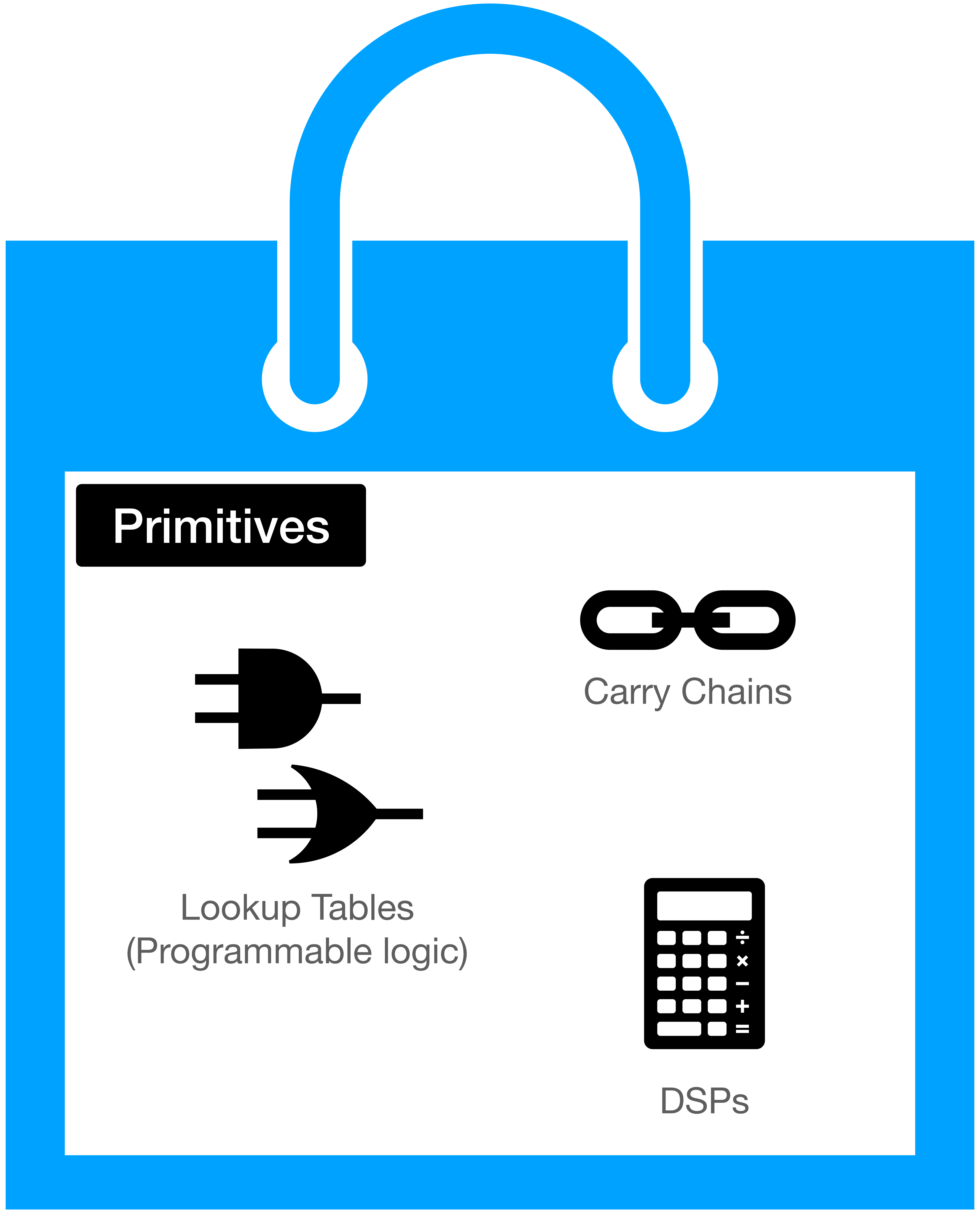
With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Let’s look at a concrete example: FPGAs.


Primitives


Primitives
Lookup Tables
(Programmable logic)


Primitives
Carry Chains
Lookup Tables
(Programmable logic)


Primitives
Carry Chains
Lookup Tables
(Programmable logic)
DSPs

Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapi… ated reasoning, … his a reality.

Even *within* FPGAs, hardware is diversifying.

Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapi… …ated reasoning, …his a reality.

Even *within* FPGAs, hardware is diversifying.

Are new primitives a challenge for FPGA compilers?

Are new primitives a challenge for FPGA compilers?

```
((d + a) * b) ^ c
```

Are new primitives a challenge for FPGA compilers?

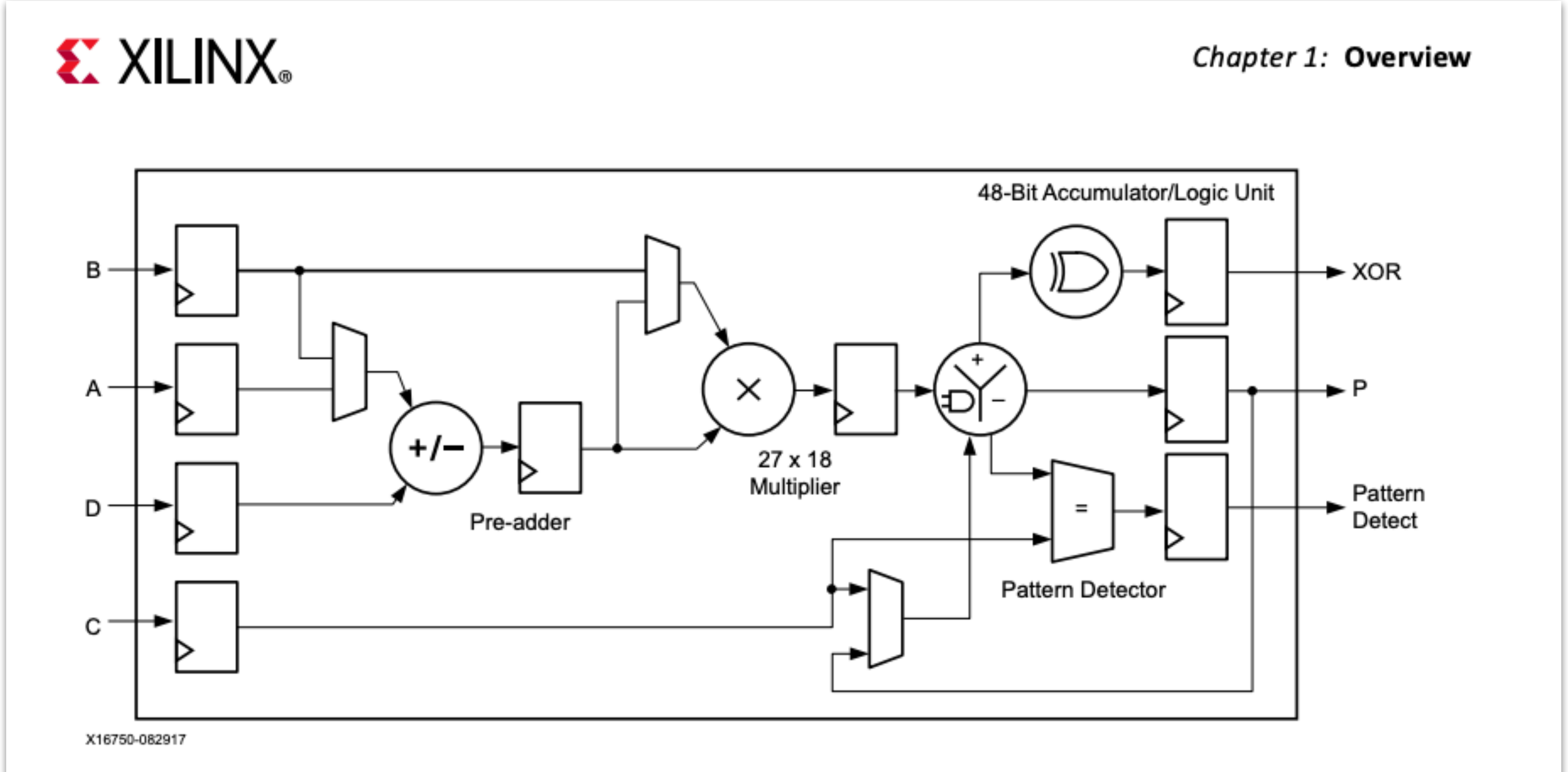


Figure 1-1: **Basic DSP48E2 Functionality**

$((d + a) * b) \wedge c$

Are new primitives a challenge for FPGA compilers?

```
Report Cell Usage:
+-------+---------+-------+
|       |Cell     |Count  |
+-------+---------+-------+
|1      |DSP48E1  |      2|
|2      |LUT2     |     10|
|3      |SRL16E   |     10|
|4      |FDRE     |     10|
+-------+---------+-------+
```

Are new primitives a challenge for FPGA compilers?

Report Cell Usage:

	Cell	Count
1	DSP48E1	2
2	LUT2	10
3	SRL16E	10
4	FDRE	10

Should use only a single DSP!

Are new primitives a challenge for FPGA compilers?

Are new primitives a challenge for FPGA compilers?

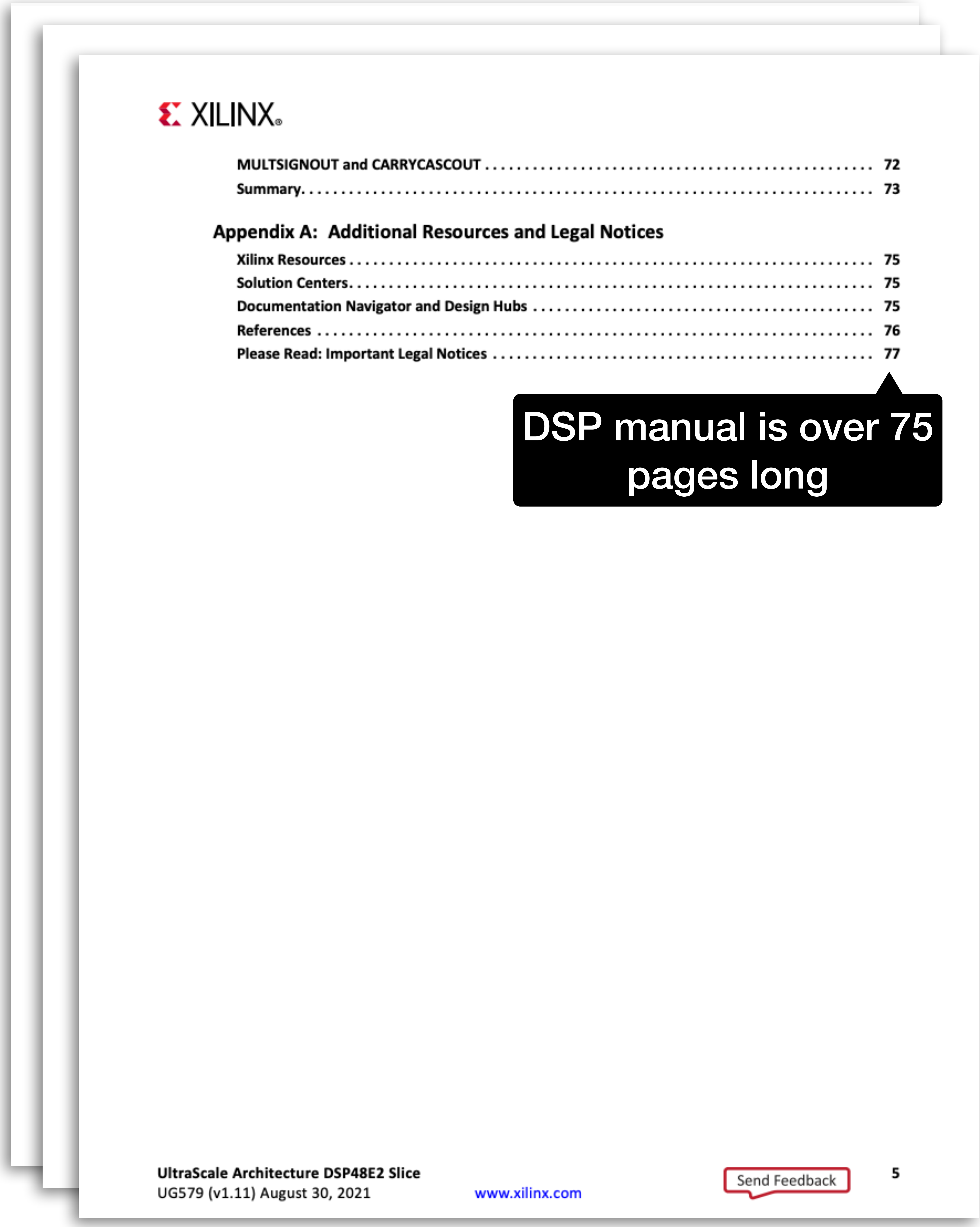
On brief inspection, yes!

Are new primitives a challenge for FPGA compilers?

On brief inspection, yes!

But this is unsurprising—DSPs are complicated.

XILINX



XILINX

```
module DSP48E2 #(
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  parameter integer ALUMODEREG = 1,
  parameter AMULTSEL = "A",
  parameter integer AREG = 1,
  parameter AUTORESET_PATDET = "NO_RESET",
  parameter AUTORESET_PRIORITY = "RESET",
  parameter A_INPUT = "DIRECT",
  ...
)(
  output [29:0] ACOUT,
  output [17:0] BCOUT,
  output CARRYCASCOUT,
  ...

  input [29:0] A,
  input [29:0] ACIN,
  input [3:0] ALUMODE,
  input [17:0] B,
  input [17:0] BCIN,
  input [47:0] C,
  ...
);
```

```
module DSP48E2 #(
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  parameter integer ALUMODEREG = 1,
  parameter AMULTSEL = "A",
  parameter integer AREG = 1,
  parameter AUTORESET_PATDET = "NO_RESET",
  parameter AUTORESET_PRIORITY = "RESET",
  parameter A_INPUT = "DIRECT",
  ...
)(
  output [29:0] ACOUT,
  output [17:0] BCOUT,
  output CARRYCASCOUT,
  ...

  input [29:0] A,
  input [29:0] ACIN,
  input [3:0] ALUMODE,
  input [17:0] B,
  input [17:0] BCIN,
  input [47:0] C,
  ...
);
```

Configuring the DSP requires setting 100+ ports and parameters

```
module DSP48E2 #(
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  parameter integer ALUMODEREG = 1,
  parameter AMULTSEL = "A",
  parameter integer AREG = 1,
  parameter AUTORESET_PATDET = "NO_RESET",
  parameter AUTORESET_PRIORITY = "RESET",
  parameter A_INPUT = "DIRECT",
  ...
)(
  output [29:0] ACOUT,
  output [17:0] BCOUT,
  output CARRYCASCOUT,
  ...

  input [29:0] A,
  input [29:0] ACIN,
  input [3:0] ALUMODE,
  input [17:0] B,
  input [17:0] BCIN,
  input [47:0] C,
  ...
);
```

Configuring the DSP requires setting 100+ ports and parameters

Table 2-4: **OPMODE Control Bits Select X Multiplexer Outputs**

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	X Multiplexer Output	Notes
xx	xxx	xx	00	0	Default
xx	xxx	01	01	M	Must select with OPMODE[3:2] = 01
xx	xxx	xx	10	P	Requires PREG = 1
xx	xxx	xx	11	A:B	48-bits wide

When either TWO24 or FOUR12 mode is selected, the multiplier must not be used, and USE_MULT must be set to NONE.

Notes:

1. When these data pins are not used and to reduce leakage power dissipation, the data pin input signals must be tied High, the input register must be selected, and the CE and RST input control signals must be tied Low. An example of unused C input recommended settings would be setting C[47:0] = all ones, CREG = 1, CEC = 0, and RSTC = 0.
2. These signals are dedicated routing paths internal to the DSP48E2 column. They are not accessible via general routing resources.
3. All signals are active High.

```
module DSP48E2 #(
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  parameter integer ALUMODEREG = 1,
  parameter AMULTSEL = "A",
  parameter integer AREG = 1,
  parameter AUTORESET_PATDET = "NO_RESET",
  parameter AUTORESET_PRIORITY = "RESET",
  parameter A_INPUT = "DIRECT",
  ...
)(
  output [29:0] ACOUT,
  output [17:0] BCOUT,
  output CARRYCASCOUT,
  ...

  input [29:0] A,
  input [29:0] ACIN,
  input [3:0] ALUMODE,
  input [17:0] B,
  input [17:0] BCIN,
  input [47:0] C,
  ...
);
```

Configuring the DSP requires setting 100+ ports and parameters

Table 2-4: **OPMODE Control Bits Select X Multiplexer Outputs**

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	X Multiplexer Output	Notes
xx	xxx	xx	00	0	Default
xx	xxx	01	01	M	Must select with OPMODE[3:2] = 01
xx	xxx	xx	10	P	Requires PREG = 1
xx	xxx	xx	11	A:B	48-bits wide

When either TWO24 or FOUR12 mode is selected, the multiplier must not be used, and USE_MULT must be set to NONE.

Notes:

1. When these data pins are not used and to reduce leakage power dissipation, the data pin input signals must be tied High, the input register must be selected, and the CE and RST input control signals must be tied Low. An example of unused C input recommended settings would be setting C[47:0] = all ones, CREG = 1, CEC = 0, and RSTC = 0.
2. These signals are dedicated routing paths internal to the DSP48E2 column. They are not accessible via general routing resources.
3. All signals are active High.

Configuring a DSP sounds a lot like writing a program!

```
module DSP48E2 #(
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  par
  par
  par
  par
  par
  par
  ...
)(
  out
  out
  out
  ...

  inp
  inp
  inp
  inp
  inp
  input [47:0] C,
  ...
);
```

Table 2-4: **OPMODE Control Bits Select X Multiplexer Outputs**

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	X Multiplexer Output	Notes

Insight #1: configuring DSPs and other complex primitives is similar to writing a program…

```
module DSP48E2 #(
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  par
  par
  par
  par
  par
  par
  ...
)(
  out
  out
  out
  ...

  inp
  inp
  inp
  inp
  inp
  input
  ...
);
```

Table 2-4: **OPMODE Control Bits Select X Multiplexer Outputs**

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	X Multiplexer Output	Notes

Insight #1: configuring DSPs and other complex primitives is similar to writing a program…

…so use *program synthesis.*

```
module DSP48E2 #(
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  ...
)(
  ...
);
```

Table 2-4: **OPMODE Control Bits Select X Multiplexer Outputs**

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	X Multiplexer Output	Notes

Insight #1: configuring DSPs and other complex primitives is similar to writing a program…

…so use *program synthesis.*

Solver-aided program synthesis: using SMT/SAT/etc. to generate programs by solving a set of constraints.

```
module DSP48E2 #(
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  parameter integer ALUMODEREG = 1,
  parameter AMULTSEL = "A",
  parameter integer AREG = 1,
  parameter AUTORESET_PATDET = "NO_RESET",
  parameter AUTORESET_PRIORITY = "RESET",
  parameter A_INPUT = "DIRECT",
  ...
)(
  output [29:0] ACOUT,
  output [17:0] BCOUT,
  output CARRYCASCOUT,
  ...

  input [29:0] A,
  input [29:0] ACIN,
  input [3:0] ALUMODE,
  input [17:0] B,
  input [17:0] BCIN,
  input [47:0] C,
  ...
);
```

Table 2-4: **OPMODE Control Bits Select X Multiplexer Outputs**

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	X Multiplexer Output	Notes
xx	xxx	xx	00	0	Default
xx	xxx	01	01	M	Must select with OPMODE[3:2] = 01
xx	xxx	xx	10	P	Requires PREG = 1
xx	xxx	xx	11	A:B	48-bits wide

When either TWO24 or FOUR12 mode is selected, the multiplier must not be used, and USE_MULT must be set to NONE.

Notes:

1. When these data pins are not used and to reduce leakage power dissipation, the data pin input signals must be tied High, the input register must be selected, and the CE and RST input control signals must be tied Low. An example of unused C input recommended settings would be setting C[47:0] = all ones, CREG = 1, CEC = 0, and RSTC = 0.
2. These signals are dedicated routing paths internal to the DSP48E2 column. They are not accessible via general routing resources.
3. All signals are active High.

```
module
   paran
   paran
   paran
   paran
   paran
   paran
   paran
   paran
   ...
)(
   outpu
   outpu
   outpu
   ...

   input
   input
   input
   input
   input
   input
   ...
);
```

DSP48E2.v

```
//////////////////////////////////////////////////////////////////
//  Copyright (c) 1995/2017 Xilinx, Inc.
//  All Right Reserved.
//////////////////////////////////////////////////////////////////
//   ____  ____
//  /   /\/   /
// /___/  \  /     Vendor      : Xilinx
// \   \   \/      Version     : 2017.3
//  \   \          Description : Xilinx Unified Simulation
//  /   /                        48-bit Multi-Functional
// /___/   /\      Filename    : DSP48E2.v
// \   \  /  \
//  \___\/\___\
//
//////////////////////////////////////////////////////////////////

`timescale 1 ps / 1 ps

`celldefine

module DSP48E2 #(
`ifdef XIL_TIMING
  parameter LOC = "UNPLACED",
`endif
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  parameter integer ALUMODEREG = 1,
  parameter AMULTSEL = "A",
  parameter integer AREG = 1,
  parameter AUTORESET_PATDET = "NO_RESET",
  parameter AUTORESET_PRIORITY = "RESET",
  parameter A_INPUT = "DIRECT",
  parameter integer BCASCREG = 1,
  parameter BMULTSEL = "B",
  ...
)(
  output [29:0] ACOUT,
  output [17:0] BCOUT,
  output CARRYCASCOUT,
  ...

  input [29:0] A,
  input [29:0] ACIN,
  input [3:0] ALUMODE,
  input [17:0] B,
  input [17:0] BCIN,
  input [47:0] C,
  ...
```

```
);

// define constants
  localparam MODULE_NAME = "DSP48E2";

// Parameter encodings and registers
  localparam AMULTSEL_A = 0;
  localparam AMULTSEL_AD = 1;
  localparam AUTORESET_PATDET_NO_RESET = 0;
  ...
`endif


  assign ACIN_in = ACIN;
  assign ALUMODE_in[0] = (ALUMODE[0] !== 1'bx) && (ALUMODE[0] ^
IS_ALUMODE_INVERTED_REG[0]); // rv 0
  assign ALUMODE_in[1] = (ALUMODE[1] !== 1'bx) && (ALUMODE[1] ^
IS_ALUMODE_INVERTED_REG[1]); // rv 0
  assign ALUMODE_in[2] = (ALUMODE[2] !== 1'bx) && (ALUMODE[2] ^
IS_ALUMODE_INVERTED_REG[2]); // rv 0
  assign ALUMODE_in[3] = (ALUMODE[3] !== 1'bx) && (ALUMODE[3] ^
IS_ALUMODE_INVERTED_REG[3]); // rv 0
  assign A_in[0] = (A[0] === 1'bx) || A[0]; // rv 1
  assign A_in[10] = (A[10] === 1'bx) || A[10]; // rv 1
  assign A_in[11] = (A[11] === 1'bx) || A[11]; // rv 1
  assign A_in[12] = (A[12] === 1'bx) || A[12]; // rv 1
  assign A_in[13] = (A[13] === 1'bx) || A[13]; // rv 1

  assign B_in[3] = (B[3] === 1'bx) || B[3]; // rv 1
  assign B_in[4] = (B[4] === 1'bx) || B[4]; // rv 1
  assign B_in[5] = (B[5] === 1'bx) || B[5]; // rv 1
  assign B_in[6] = (B[6] === 1'bx) || B[6]; // rv 1
  assign B_in[7] = (B[7] === 1'bx) || B[7]; // rv 1
  assign B_in[8] = (B[8] === 1'bx) || B[8]; // rv 1
  assign B_in[9] = (B[9] === 1'bx) || B[9]; // rv 1
  assign CARRYCASCIN_in = CARRYCASCIN;
  assign CARRYINSEL_in[0] = (CARRYINSEL[0] !== 1'bx) &&
CARRYINSEL[0]; // rv 0
  assign CARRYINSEL_in[1] = (CARRYINSEL[1] !== 1'bx) &&
CARRYINSEL[1]; // rv 0
  assign CARRYINSEL_in[2] = (CARRYINSEL[2] !== 1'bx) &&
CARRYINSEL[2]; // rv 0
  assign CARRYIN_in = (CARRYIN !== 1'bx) && (CARRYIN ^
IS_CARRYIN_INVERTED_REG); // rv 0
  assign CEA1_in = (CEA1 !== 1'bx) && CEA1; // rv 0
  assign CEA2_in = (CEA2 !== 1'bx) && CEA2; // rv 0
  assign CEAD_in = (CEAD !== 1'bx) && CEAD; // rv 0
  assign CEALUMODE_in = (CEALUMODE !== 1'bx) && CEALUMODE; // rv
0
  assign CEB1_in = (CEB1 !== 1'bx) && CEB1; // rv 0
  assign CEB2_in = (CEB2 !== 1'bx) && CEB2; // rv 0
```

```
  assign CECARRYIN_in = (CECARRYIN !== 1'bx) && CECARRYIN; // rv
0
  assign CECTRL_in = (CECTRL !== 1'bx) && CECTRL; // rv 0
  assign CEC_in = (CEC !== 1'bx) && CEC; // rv 0
  assign CED_in = (CED !== 1'bx) && CED; // rv 0
  assign CEINMODE_in = CEINMODE;
  assign CEM_in = (CEM !== 1'bx) && CEM; // rv 0
  assign CEP_in = (CEP !== 1'bx) && CEP; // rv 0
  assign CLK_in = (CLK !== 1'bx) && (CLK ^ IS_CLK_INVERTED_REG);
// rv 0
  assign C_in[0] = (C[0] === 1'bx) || C[0]; // rv 1
  assign C_in[10] = (C[10] === 1'bx) || C[10]; // rv 1
  a
  assign D_in[1] = (D[1] !== 1'bx) && D[1]; // rv 0
  assign D_in[20] = (D[20] !== 1'bx) && D[20]; // rv 0
  assign D_in[21] = (D[21] !== 1'bx) && D[21]; // rv 0
  assign D_in[22] = (D[22] !== 1'bx) && D[22]; // rv 0
  assign D_in[23] = (D[23] !== 1'bx) && D[23]; // rv 0
  assign D_in[24] = (D[24] !== 1'bx) && D[24]; // rv 0
  assign D_in[25] = (D[25] !== 1'bx) && D[25]; // rv 0
  assign D_in[26] = (D[26] !== 1'bx) && D[26]; // rv 0
  assign D_in[2] = (D[2] !== 1'bx) && D[2]; // rv 0
  assign D_in[3] = (D[3] !== 1'bx) && D[3]; // rv 0
  assign D_in[4] = (D[4] !== 1'bx) && D[4]; // rv 0
  assign D_in[5] = (D[5] !== 1'bx) && D[5]; // rv 0
  assign D_in[6] = (D[6] !== 1'bx) && D[6]; // rv 0
  assign D_in[7] = (D[7] !== 1'bx) && D[7]; // rv 0
  assign D_in[8] = (D[8] !== 1'bx) && D[8]; // rv 0
  assign D_in[9] = (D[9] !== 1'bx) && D[9]; // rv 0
  assign INMODE_in[0] = (INMODE[0] !== 1'bx) && (INMODE[0] ^
IS_INMODE_INVERTED_REG[0]); // rv 0
  assign INMODE_in[1] = (INMODE[1] !== 1'bx) && (INMODE[1] ^
IS_INMODE_INVERTED_REG[1]); // rv 0
  assign INMODE_in[2] = (INMODE[2] !== 1'bx) && (INMODE[2] ^
IS_INMODE_INVERTED_REG[2]); // rv 0
  assign INMODE_in[3] = (INMODE[3] !== 1'bx) && (INMODE[3] ^
IS_INMODE_INVERTED_REG[3]); // rv 0
  assign INMODE_in[4] = (INMODE[4] !== 1'bx) && (INMODE[4] ^
IS_INMODE_INVERTED_REG[4]); // rv 0
  assign MULTSIGNIN_in = MULTSIGNIN;
  assign OPMODE_in[0] = (OPMODE[0] !== 1'bx) && (OPMODE[0] ^
IS_OPMODE_INVERTED_REG[0]); // rv 0
  assign OPMODE_in[1] = (OPMODE[1] !== 1'bx) && (OPMODE[1] ^
IS_OPMODE_INVERTED_REG[1]); // rv 0
  assign OPMODE_in[2] = (OPMODE[2] !== 1'bx) && (OPMODE[2] ^
IS_OPMODE_INVERTED_REG[2]); // rv 0
  assign OPMODE_in[3] = (OPMODE[3] !== 1'bx) && (OPMODE[3] ^
IS_OPMODE_INVERTED_REG[3]); // rv 0

• • •
```

```
module
  paran
  paran
  paran
  paran
  paran
  paran
  paran
  paran
  ...
)(
  outpu
  outpu
  outpu
  ...
  input
  input
  input
  input
  input
  input
  ...
);
```

DSP48E2.v

```
///////////////////////////////////////////////////////////////////////
//  Copyright (c) 1995/2017 Xilinx, Inc.
//  All Right Reserved.
///////////////////////////////////////////////////////////////////////
//   ____  ____
//  /   /\/   /
// /___/  \  /     Vendor      : Xilinx
// \   \   \/      Version     : 2017.3
//  \   \          Description : Xilinx Unified Simulation
//  /   /                        48-bit Multi-Functional
// /___/   /\      Filename    : DSP48E2.v
// \   \  /  \
//  \___\/\___\
//
///////////////////////////////////////////////////////////////////////

`timescale 1 ps / 1 ps

`celldef

module D
`ifdef X
  parame
`endif
  parame
  parame
  parameter integer ALUMODEREG = 1,
  parameter AMULTSEL = "A",
  parameter integer AREG = 1,
  parameter AUTORESET_PATDET = "NO_RESET",
  parameter AUTORESET_PRIORITY = "RESET",
  parameter A_INPUT = "DIRECT",
  parameter integer BCASCREG = 1,
  parameter BMULTSEL = "B",
  ...
)(
  output [29:0] ACOUT,
  output [17:0] BCOUT,
  output CARRYCASCOUT,
  ...

  input [29:0] A,
  input [29:0] ACIN,
  input [3:0] ALUMODE,
  input [17:0] B,
  input [17:0] BCIN,
  input [47:0] C,
  ...
```

```
);

// define constants
  localparam MODULE_NAME = "DSP48E2";

// Parameter encodings and registers
  localparam AMULTSEL_A = 0;
  localparam AMULTSEL_AD = 1;
  localparam AUTORESET_PATDET_NO_RESET = 0;
  ...
`endif


  assign ACIN_in = ACIN;
  assign ALUMODE_in[0] = (ALUMODE[0] !== 1'bx) && (ALUMODE[0] ^
IS_ALUMODE_INVERTED_REG[0]); // rv 0
  assign ALUMODE_in[1] = (ALUMODE[1] !== 1'bx) && (ALUMODE[1] ^
IS_ALUMODE_INVERTED_REG[1]); // rv 0
  assign ALUMODE_in[2] = (ALUMODE[2] !== 1'bx) && (ALUMODE[2] ^
IS_ALUMODE_INVERTED_REG[2]); // rv 0
  assign ALUMODE_in[3] = (ALUMODE[3] !== 1'bx) && (ALUMODE[3] ^
IS_ALUMODE_INVERTED_REG[3]); // rv 0

  assign B_in[5] = (B[5] === 1'bx) || B[5]; // rv 1
  assign B_in[6] = (B[6] === 1'bx) || B[6]; // rv 1
  assign B_in[7] = (B[7] === 1'bx) || B[7]; // rv 1
  assign B_in[8] = (B[8] === 1'bx) || B[8]; // rv 1
  assign B_in[9] = (B[9] === 1'bx) || B[9]; // rv 1
  assign CARRYCASCIN_in = CARRYCASCIN;
  assign CARRYINSEL_in[0] = (CARRYINSEL[0] !== 1'bx) &&
CARRYINSEL[0]; // rv 0
  assign CARRYINSEL_in[1] = (CARRYINSEL[1] !== 1'bx) &&
CARRYINSEL[1]; // rv 0
  assign CARRYINSEL_in[2] = (CARRYINSEL[2] !== 1'bx) &&
CARRYINSEL[2]; // rv 0
  assign CARRYIN_in = (CARRYIN !== 1'bx) && (CARRYIN ^
IS_CARRYIN_INVERTED_REG); // rv 0
  assign CEA1_in = (CEA1 !== 1'bx) && CEA1; // rv 0
  assign CEA2_in = (CEA2 !== 1'bx) && CEA2; // rv 0
  assign CEAD_in = (CEAD !== 1'bx) && CEAD; // rv 0
  assign CEALUMODE_in = (CEALUMODE !== 1'bx) && CEALUMODE; // rv
0
  assign CEB1_in = (CEB1 !== 1'bx) && CEB1; // rv 0
  assign CEB2_in = (CEB2 !== 1'bx) && CEB2; // rv 0
```

```
  assign CECARRYIN_in = (CECARRYIN !== 1'bx) && CECARRYIN; // rv
0
  assign CECTRL_in = (CECTRL !== 1'bx) && CECTRL; // rv 0
  assign CEC_in = (CEC !== 1'bx) && CEC; // rv 0
  assign CED_in = (CED !== 1'bx) && CED; // rv 0
  assign CEINMODE_in = CEINMODE;
  assign CEM_in = (CEM !== 1'bx) && CEM; // rv 0
  assign CEP_in = (CEP !== 1'bx) && CEP; // rv 0
  assign CLK_in = (CLK !== 1'bx) && (CLK ^ IS_CLK_INVERTED_REG);
// rv 0
  assign C_in[0] = (C[0] === 1'bx) || C[0]; // rv 1
  assign C_in[10] = (C[10] === 1'bx) || C[10]; // rv 1
  a
  assign D_in[1] = (D[1] !== 1'bx) && D[1]; // rv 0
  assign D_in[20] = (D[20] !== 1'bx) && D[20]; // rv 0
  assign D_in[21] = (D[21] !== 1'bx) && D[21]; // rv 0
  assign D_in[22] = (D[22] !== 1'bx) && D[22]; // rv 0
  assign D_in[23] = (D[23] !== 1'bx) && D[23]; // rv 0
  assign D_in[24] = (D[24] !== 1'bx) && D[24]; // rv 0
  assign D_in[25] = (D[25] !== 1'bx) && D[25]; // rv 0
  assign D_in[26] = (D[26] !== 1'bx) && D[26]; // rv 0
  assign D_in[2] = (D[2] !== 1'bx) && D[2]; // rv 0

                                              MODE[0] ^
IS_INMODE_INVERTED_REG[0]); // rv 0
  assign INMODE_in[1] = (INMODE[1] !== 1'bx) && (INMODE[1] ^
IS_INMODE_INVERTED_REG[1]); // rv 0
  assign INMODE_in[2] = (INMODE[2] !== 1'bx) && (INMODE[2] ^
IS_INMODE_INVERTED_REG[2]); // rv 0
  assign INMODE_in[3] = (INMODE[3] !== 1'bx) && (INMODE[3] ^
IS_INMODE_INVERTED_REG[3]); // rv 0
  assign INMODE_in[4] = (INMODE[4] !== 1'bx) && (INMODE[4] ^
IS_INMODE_INVERTED_REG[4]); // rv 0
  assign MULTSIGNIN_in = MULTSIGNIN;
  assign OPMODE_in[0] = (OPMODE[0] !== 1'bx) && (OPMODE[0] ^
IS_OPMODE_INVERTED_REG[0]); // rv 0
  assign OPMODE_in[1] = (OPMODE[1] !== 1'bx) && (OPMODE[1] ^
IS_OPMODE_INVERTED_REG[1]); // rv 0
  assign OPMODE_in[2] = (OPMODE[2] !== 1'bx) && (OPMODE[2] ^
IS_OPMODE_INVERTED_REG[2]); // rv 0
  assign OPMODE_in[3] = (OPMODE[3] !== 1'bx) && (OPMODE[3] ^
IS_OPMODE_INVERTED_REG[3]); // rv 0

• • •
```

Simulation models provide the formal semantics of behaviors and constraints necessary for automated reasoning!

```
module DSP48E2 #(
  par
  par
  par
  par
  par
  par
  par
  par
  ...
)(
  out
  out
  out
  ...

  inp
  inp
  inp
  inp
  inp
  inp
  ...
);
```

Insight #1: configuring DSPs and other complex primitives is similar to writing a program, so use *program synthesis.*

Insight #2: we can extract the semantics necessary for automated reasoning directly from simulation models.

Lakeroad: a hardware synthesis tool utilizing program synthesis and semantics extracted from simulation models to target complex, programmable FPGA primitives.

Workload	Signed?	# Stages	Yosys	SOTA	Lakeroad
`((d + a) * b) | c`	X	1	1 DSP, 20 LUT	1 DSP, 10 LUT	1 DSP
`((d - a) * b) | c`	✓	2	1 DSP, 20 LUT	1 DSP, 10 LUT	1 DSP
`((d - a) * b) ^ c`	✓	3	1 DSP, 22 LUT	2 DSP, 11 LUT	1 DSP
`((d + a) * b) & c`	✓	3	1 DSP, 22 LUT	2 DSP, 11 LUT	1 DSP
`((d + a) * b) ^ c`	X	2	1 DSP, 18 LUT	1 DSP, 9 LUT	1 DSP

Workload	Signed?	# Stages	Yosys	SOTA	Lakeroad
`((d + a) * b)	c`	X	1	1 DSP, 20 LUT	1 DSP, 10 LUT	1 DSP
`((d - a) * b)	c`	✓	2	1 DSP, 20 LUT	1 DSP, 10 LUT	1 DSP
`((d - a) * b) ^ c`	✓	3	1 DSP, 22 LUT	2 DSP, 11 LUT	1 DSP
`((d + a) * b) & c`	✓	3	1 DSP, 22 LUT	2 DSP, 11 LUT	1 DSP
`((d + a) * b) ^ c`	X	2	1 DSP, 18 LUT	1 DSP, 9 LUT	1 DSP

Workload	Signed?	# Stages	Yosys	SOTA	Lakeroad
((d + a) * b) | c	X	1	1 DSP, 20 LUT	1 DSP, 10 LUT	1 DSP
((d - a) * b) | c	✓	2	1 DSP, 20 LUT	1 DSP, 10 LUT	1 DSP
((d - a) * b) ^ c	✓	3	1 DSP, 22 LUT	2 DSP, 11 LUT	1 DSP
((d + a) * b) & c	✓	3	1 DSP, 22 LUT	2 DSP, 11 LUT	1 DSP
((d + a) * b) ^ c	X	2	1 DSP, 18 LUT	1 DSP, 9 LUT	1 DSP

Compilers should be generated from formal models of hardware.

Compilers should be generated from formal models of hardware.

Lakeroad automatically enables compilation to FPGA primitives, given the simulation models of those primitives.

Compilers should be generated from formal models of hardware.

Lakeroad automatically enables compilation to FPGA primitives, given the simulation models of those primitives.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Compilers should be generated from formal models of hardware.

Lakeroad automatically enables compilation to FPGA primitives, given the simulation models of those primitives.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Lakeroad demonstrates that automated methods are now powerful enough address gaps in existing state-of-the-art tools.

Generating Compilers → Why Now? → Case Study: Lakeroad → Call to Action

Generating Compilers → Why Now? → Case Study: Lakeroad → Call to Action

The Hardware Lottery

Sara Hooker

Google Research, Brain Team

shooker@google.com

2020

The Hardware Lottery

Sara Hooker

Google Research, Brain Team

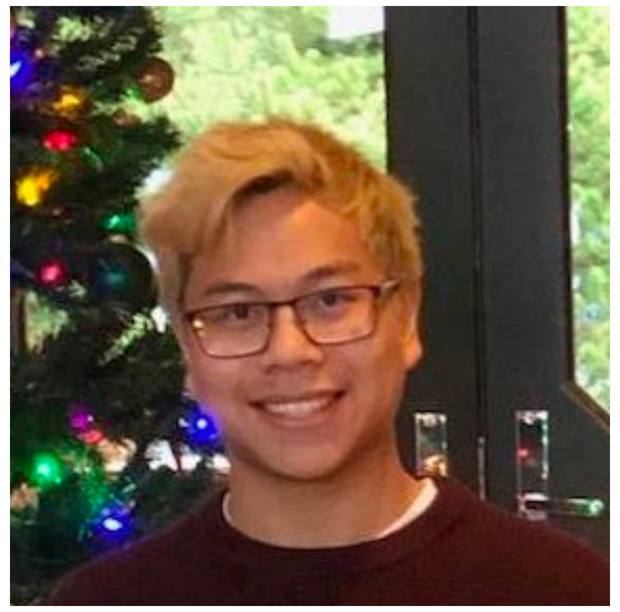
shooker@google.com

2020

It’s on on all of us to fight against the hardware lottery, by making sure that practitioners in all fields have the hardware and compilers they need to advance their research.

Thank you!



PAUL G. ALLEN SCHOOL