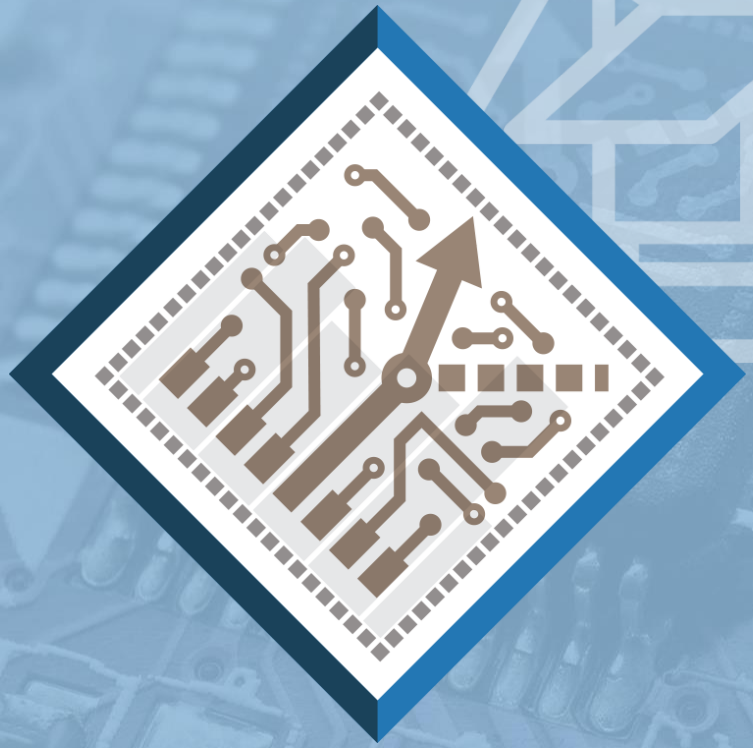
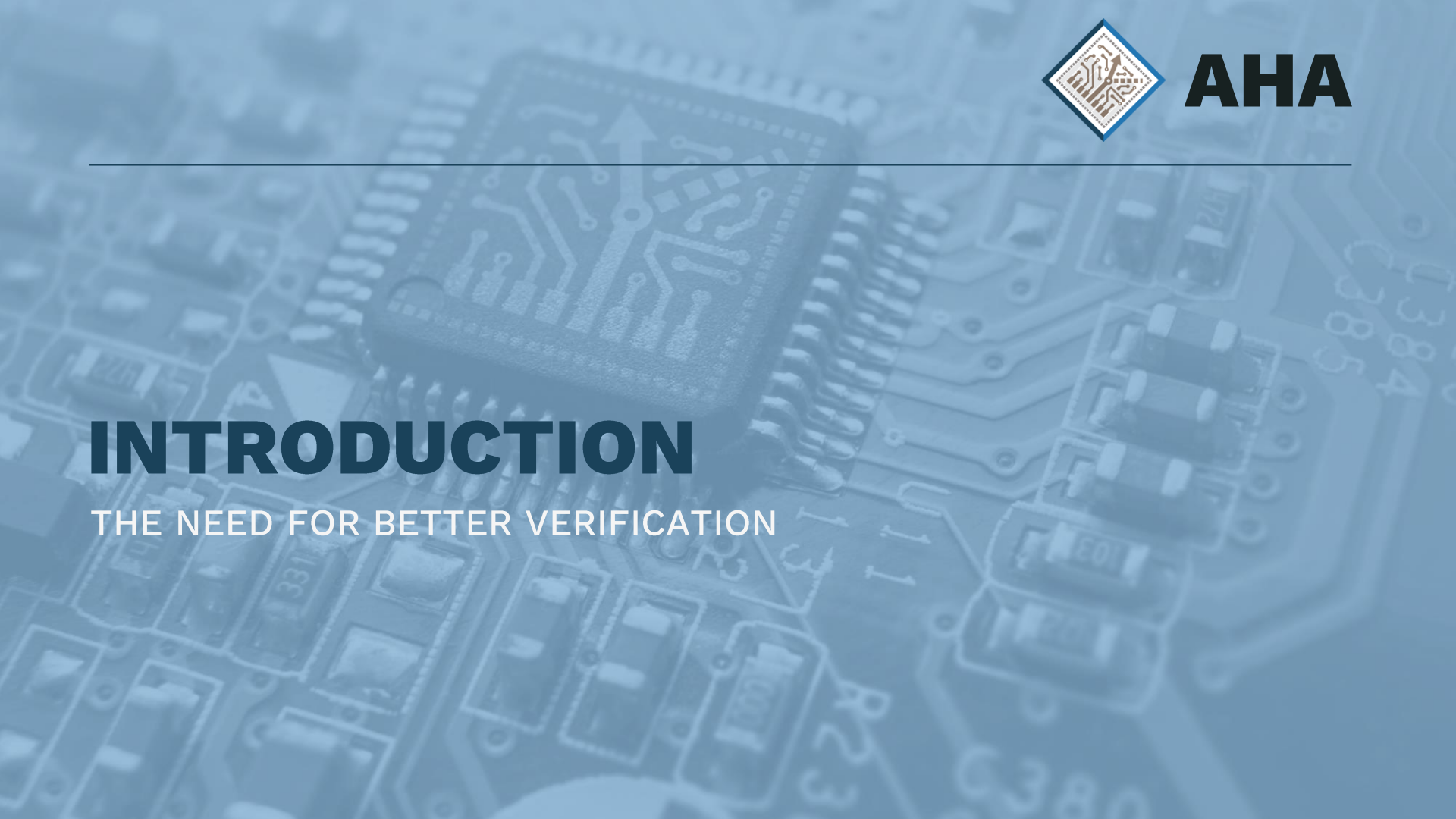
AGILE HARDWARE CENTER



FORMAL HARDWARE VERIFICATION:
CHALLENGES AND OPPORTUNITIES



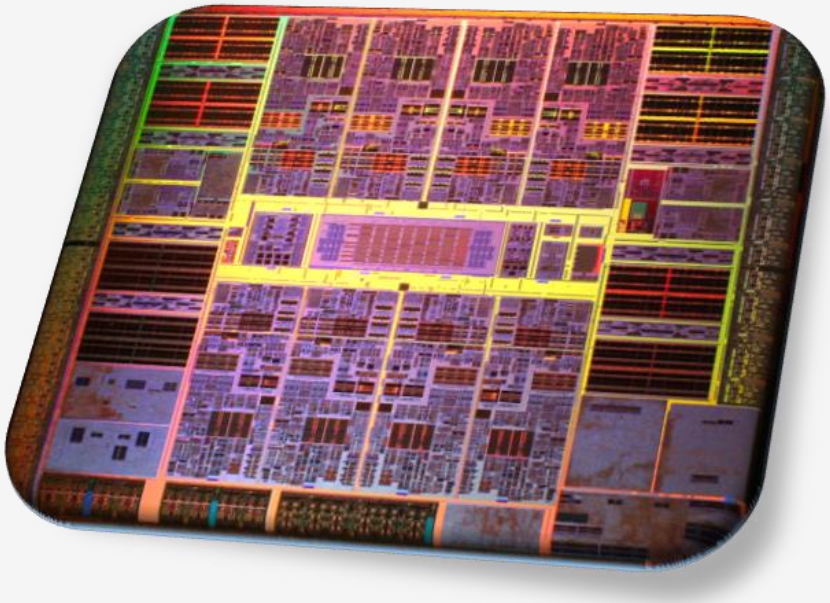
INTRODUCTION

THE NEED FOR BETTER VERIFICATION

THE VERIFICATION CHALLENGE

- **Systems on a Chip (SoCs)**
  - Growing in size and complexity
  - Single chip contains multiple cores, caches, accelerators

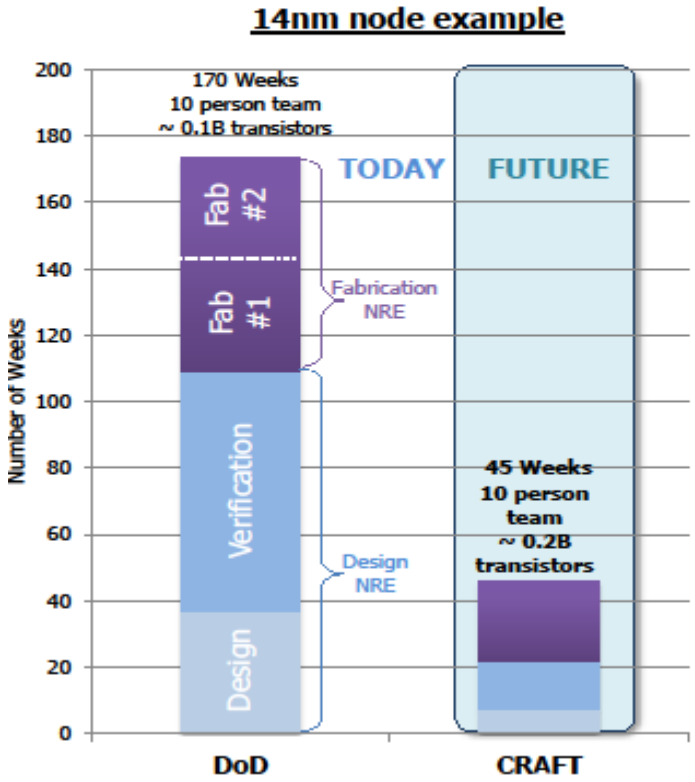
- **SoC Verification**
  - SoCs used in critical applications
  - Correctness is essential
  - Failures can be extremely costly (Intel FDIV: $500M)
  - Verification dominates design



Current design flow takes so long that it is throttling DoD access to advanced technology

Existing DoD custom IC product cycle time can take as long as **2.5 years**.

- 60%: Design (most of which is verification)
- 40%: Fabrication (20%/fab spin)



Data from industry survey by DARPA consultants

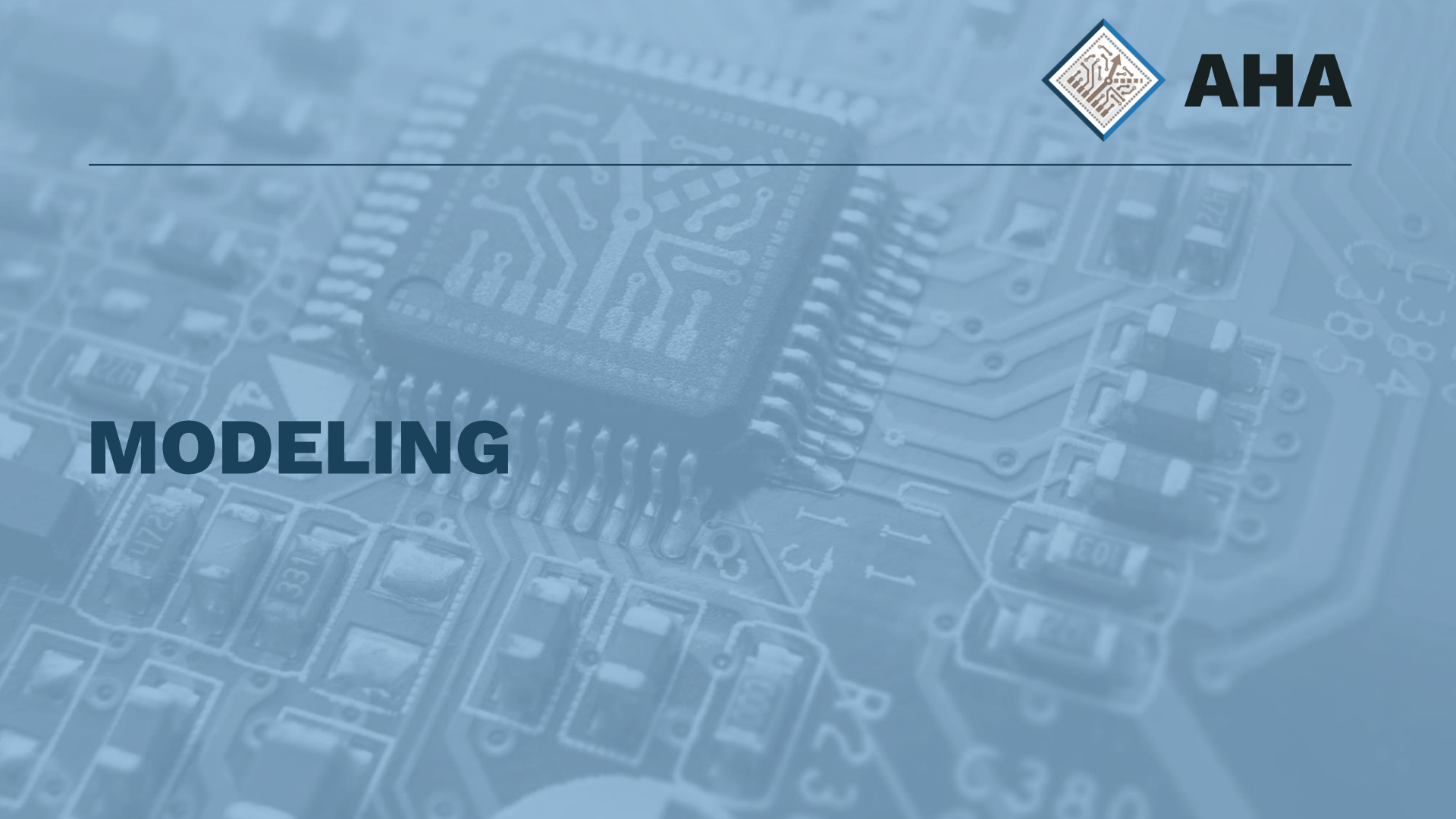
Slide from DARPA CRAFT proposer's day

FORMAL VERIFICATION

- Has potential to revolutionize verification
  - Better than testing: covers all possible cases
  - Already used extensively in industry
  - But there are many challenges

- Three main steps
  - Create a mathematical model of the system
  - Specify formally what the properties of the system should be
  - Prove that the model has the desired properties

FORMAL VERIFICATION


AHA
MODELING

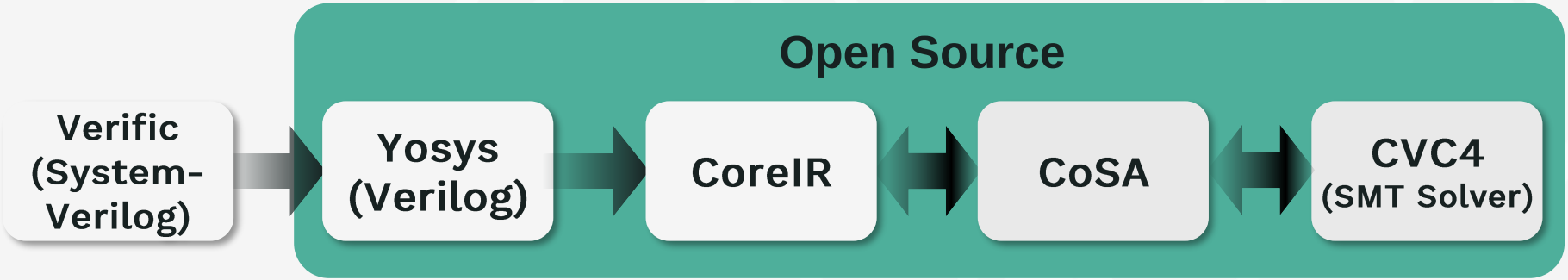
MODELING

- Good news: HDL descriptions are already a formal model

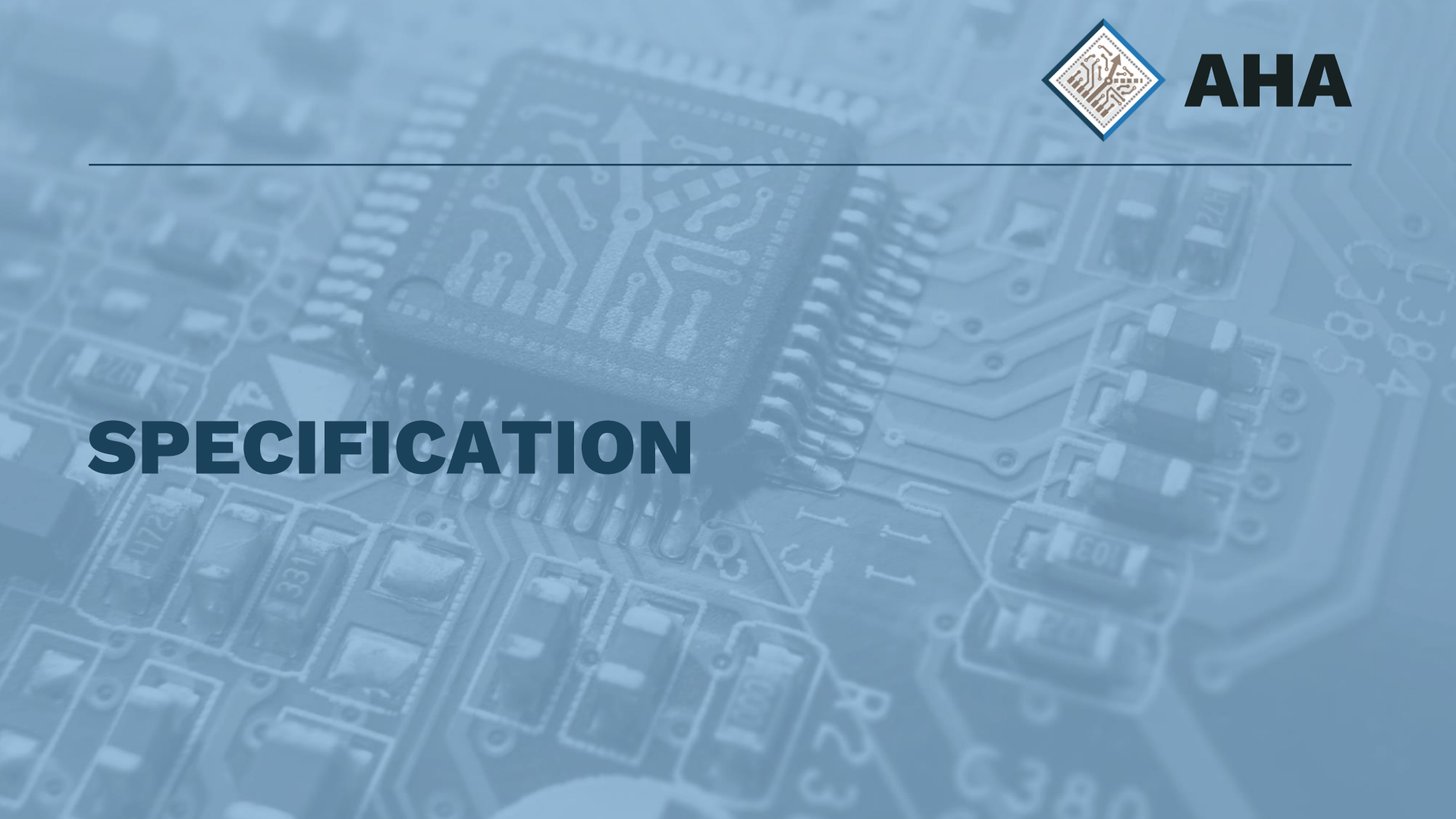
- Challenges
  - Lack of non-commercial tools that can parse modern HDL's
  - Analog / mixed-signal components
  - Closed-source IP can't be modeled precisely

- Solutions
  - New open-source toolchains are being developed
  - AMS circuits can often be approximated by digital circuits
  - New initiatives to build open-source hardware

OPEN-SOURCE FORMAL VERIFICATION FLOW



- Yosys: open-source front-end for Verilog
  - Optional (commercial) Verific front-end for full SystemVerilog support
- CoreIR: “LLVM for hardware” open intermediate representation
- CoSA (CoreIR Symbolic Analyzer): open source formal analysis (model checking, bounded model checking)
- CVC4: open-source SMT solver



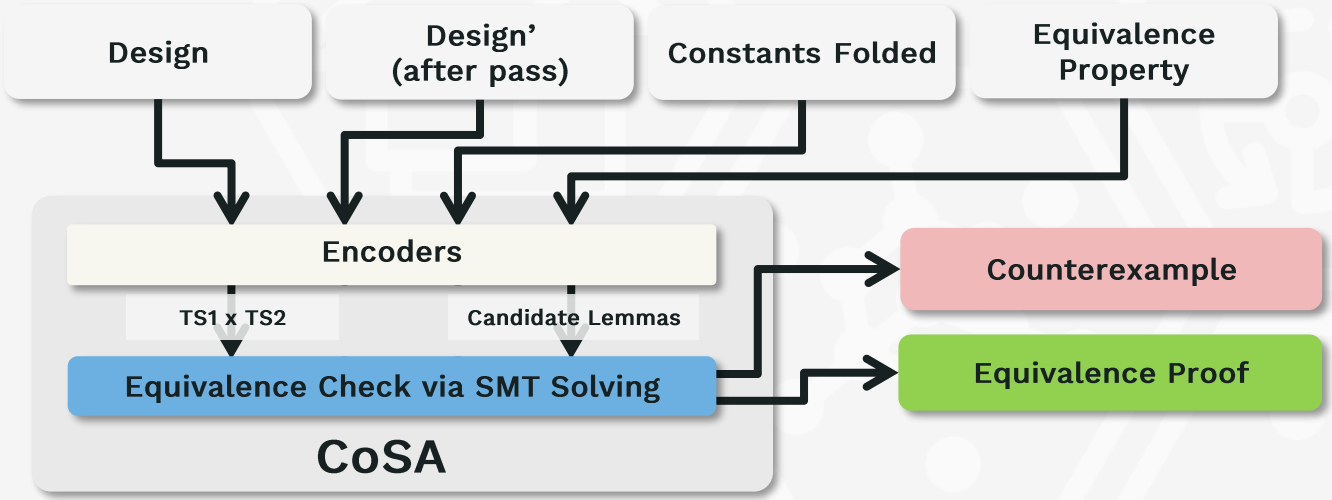
SPECIFICATION

SPECIFICATION

- Challenges for today's techniques (e.g. assertion-based verification)
  - Requires design knowledge
  - Requires manual effort
  - If incomplete, then will miss bugs

- Solutions
  - Integrated verification and design
  - Symbolic QED – no manual specification needed
    - Technique developed together with Prof. Subhasish Mitra

INTEGRATED VERIFICATION: EQUIVALENCE CHECKING



- Comparison of a CoreIR design before and after optimization pass
- CoreIR optimization pass provides information about constants folded directly to CoSA
- Significant performance improvement: 1.3 min vs. timeout (2 h)

QUICK ERROR DETECTION

- Quick Error Detection (QED)
  - Technique developed by Subhasish Mitra's group
  - Uses shadow registers and memory
  - Applies duplicate and check transformation to improve tests

	Regular	Shadow
Registers	R0-R15	R16-R31
Memory	0x10000-0x1FFFF	0x20000-0x2FFFF

```
...
LD R1,   [0x10000]
LD R2,   [0x10040]
...
```

```
LD R1,   [0x10000]
LD R2,   [0x10040]
LD R17,  [0x20000]
LD R18,  [0x20040]
CMP R1 == R17
CMP R2 == R18
```

SYMBOLIC QUICK ERROR DETECTION

- Combines formal methods with QED
  - Leverages idea of self-consistency (Jones '96)
  - Does there exist *any* sequence of instructions that would fail a QED test
  - Searches all possible sequences using bounded model checking
  - Runs automatically overnight

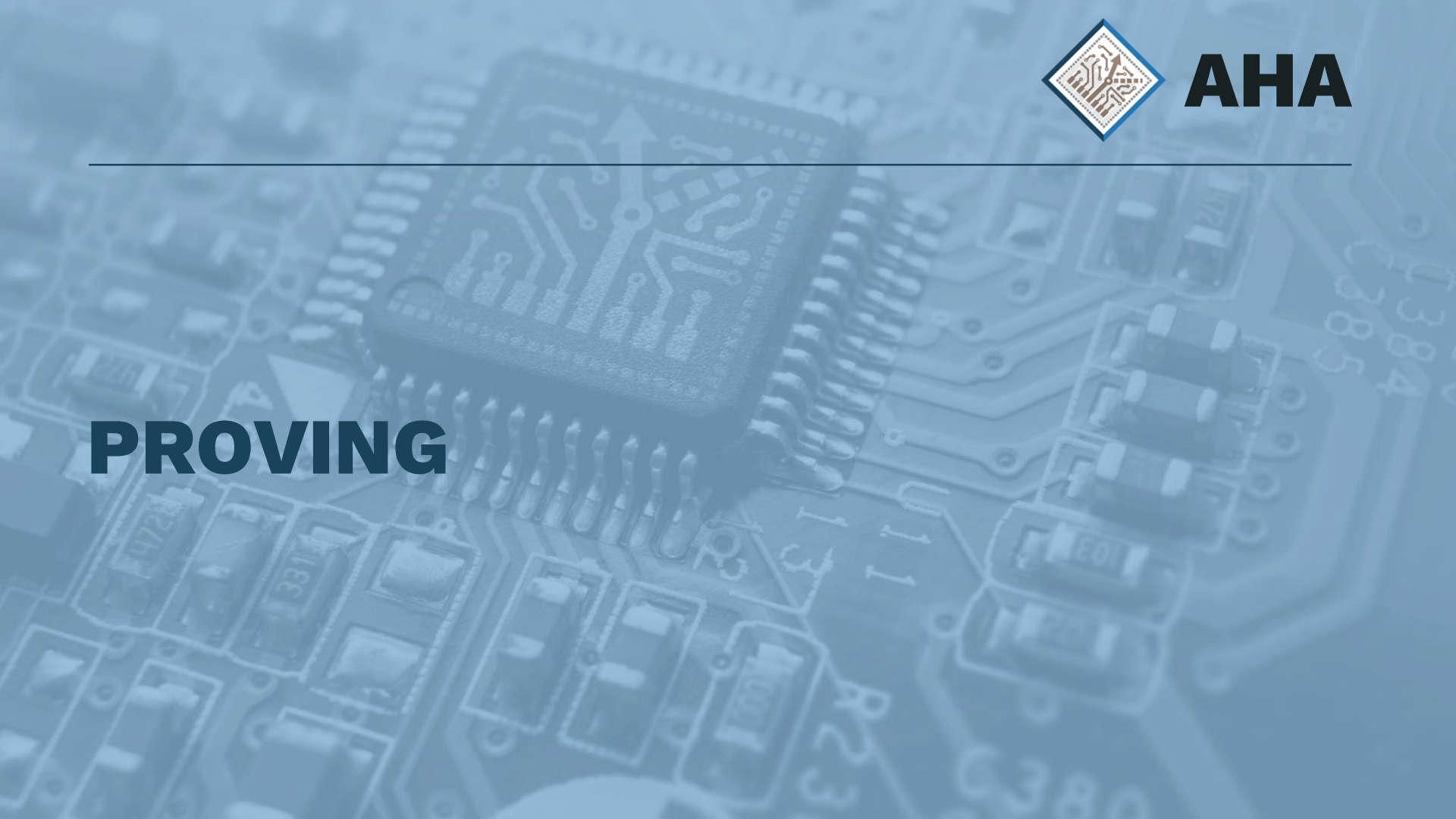
- Some results
  - OpenSPARC T2: found 92/92 tough bugs automatically
    - Few minutes to few hours each
    - Each found bug returned a bug trace of less than 10 instructions
  - RIDECORE: (open-source out-of-order Risc-V core)
    - Found previously unknown bug
    - Bug was reported and fixed

SYMBOLIC QUICK ERROR DETECTION

- Surprisingly, can be made into a complete technique
  - SQED can, in theory, find all bugs in a processor core
  - Limited only by the power of the bounded model checker

- Can be extended beyond processor cores
  - Same idea can be used for accelerators
  - Instruction-level abstraction (ILA) developed by Malik and Gupta at Princeton makes accelerators look like processors

- Techniques for scaling up (work in progress)
  - Symbolic initial states
  - Automatic design partitioning
  - Abstraction of uninteresting components


AHA
PROVING

PROVING

- Challenges
  - Manual proofs require enormous effort
  - Automated techniques limited to small designs

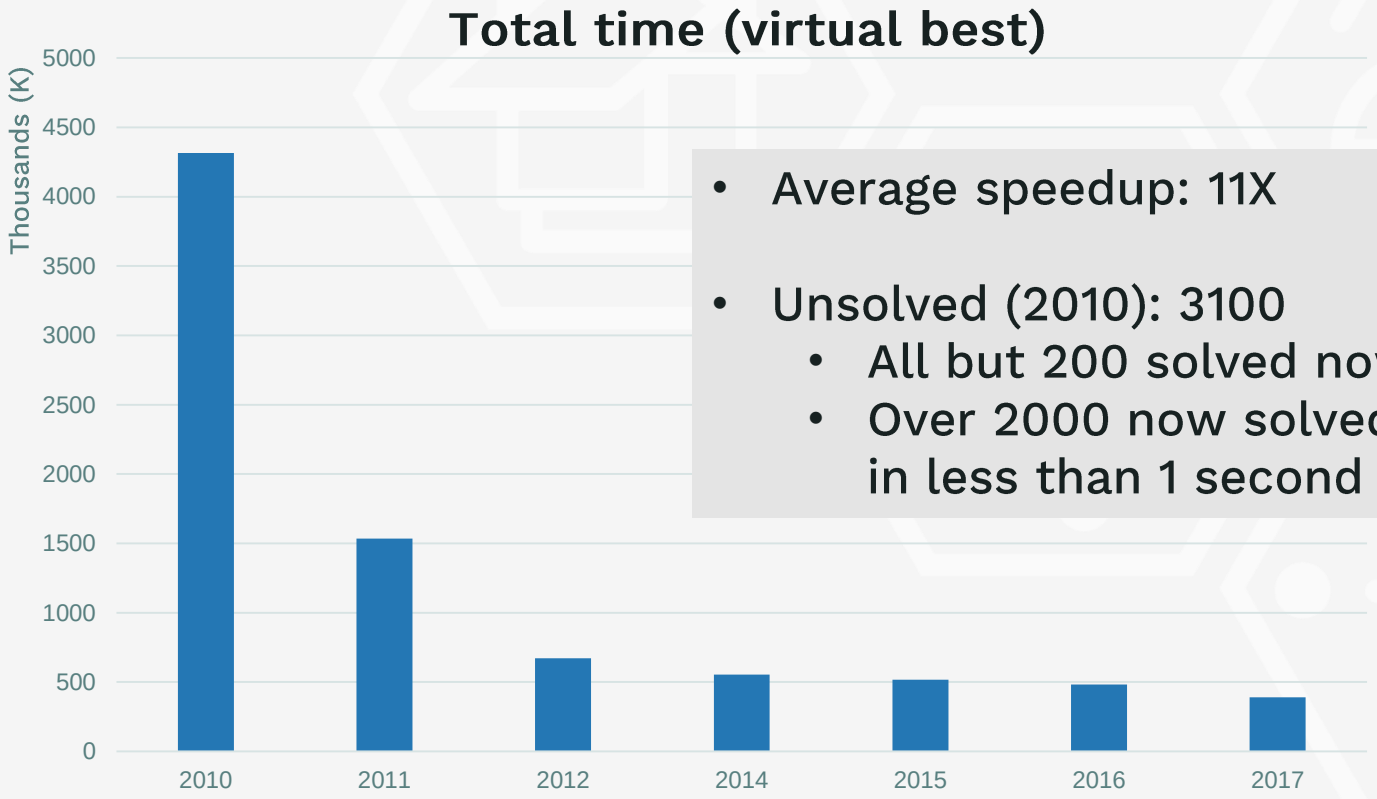
- Solution:
  - Better Solvers!

EVOLUTION OF SMT SOLVING

Quantifier-Free Bitvector (QF_BV) SMT-LIB benchmarks

- Comparison of virtual best SMT solvers since 2010
- Evaluation on 39610 benchmarks (SMT-LIB 2018)
- 41 different families of benchmarks

EVOLUTION OF SMT SOLVING



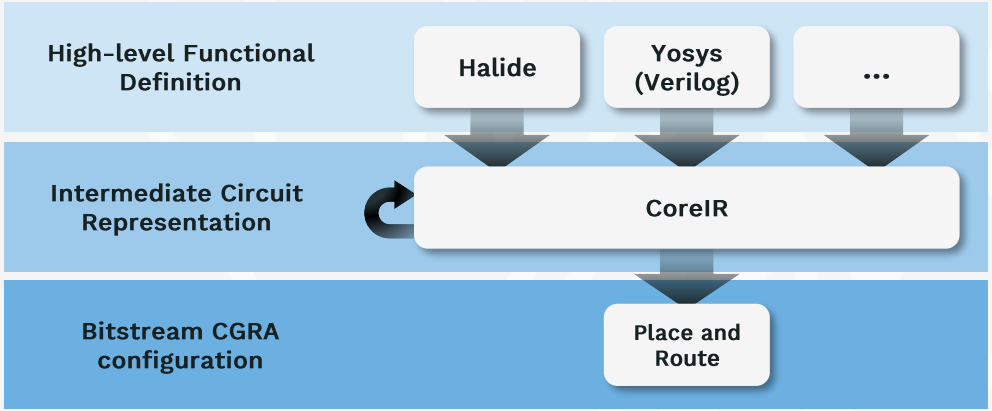
- Average speedup: 11X
- Unsolved (2010): 3100
  - All but 200 solved now
  - Over 2000 now solved in less than 1 second

HOW WILL WE GET EVEN BETTER SOLVERS?

- Leverage SAT technology
  - SAT solvers also improving each year

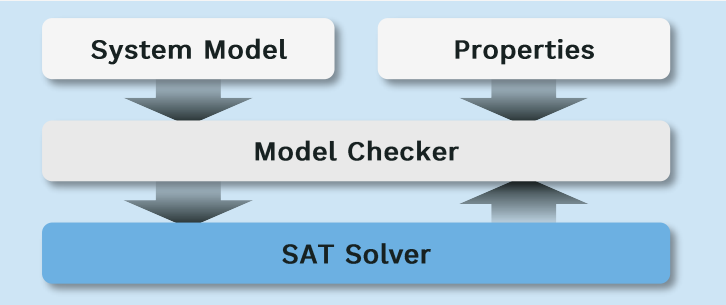
- Build better SMT solvers
  - More powerful rewriting and algebraic reasoning
  - New SMT theories
  - Specific optimizations for bounded model checking
  - Parallel computing
  - Machine learning

AGILE HARDWARE FLOW



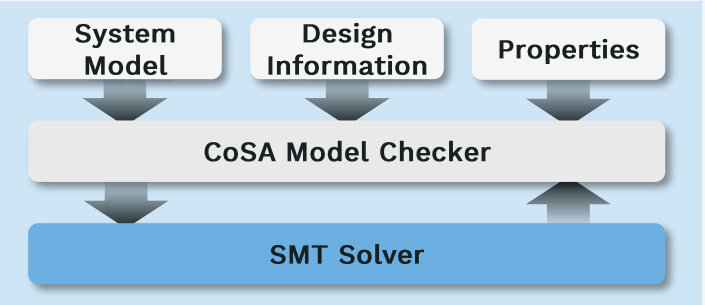
- Improve hardware design productivity with ideas from software
- Start with a high-level description language for image processing, e.g., Halide
- Uses CoreIR as a representation throughout the flow
- Allows for multiple front-ends such as Yosys
- Inspire design re-use and innovation by providing open-source toolchain

Model Checking (Commercial Tools)



- Model and Properties as single entry point to the model-checker
- Bitblasting loses word-level structure
- Black-box usage of SAT solver
- Closed source

CoreIR Symbolic Analyzer (CoSA)



- BMC and k-induction engines
- Leverages CoreIR circuit representation developed as part of Stanford agile hardware (AHA) project
- Can make use of additional design information to optimize verification
- Maintains word-level structure by using SMT theories
- Uses customizable SMT solver developed by Barrett group at Stanford
- Entire toolchain is open-source