Interplay Between Language and Formal Verification

Dr. Carl Seger

Senior Principal Engineer
Strategic CAD Labs, Intel Corp.

Nov. 4, 2009

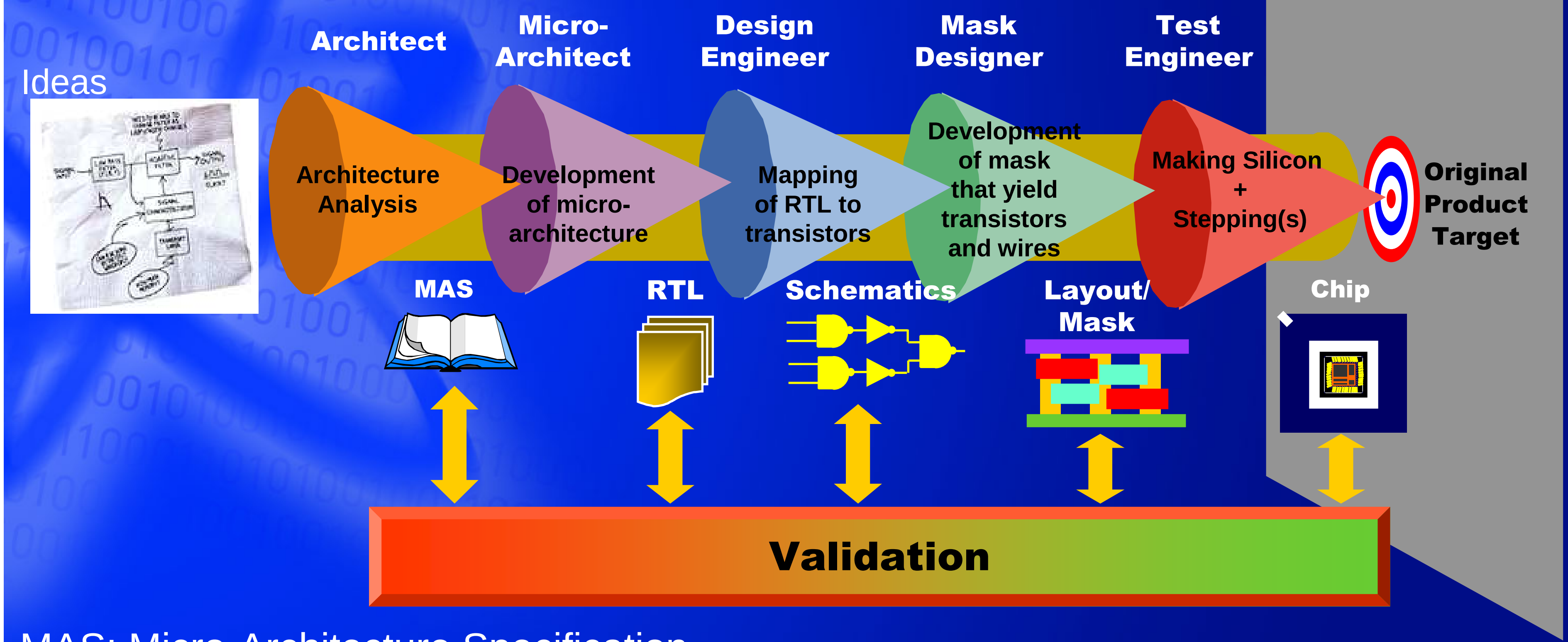
Quiz

Outline

- Context of talk
- Evolution of a custom language
  - Stage One: Scripting & implementation language
  - Stage Two: Property specification language
  - Stage Three: Term language
  - Stage Four: Modeling language
- Lessons learned

Context

The Design Process at 10,000 ft

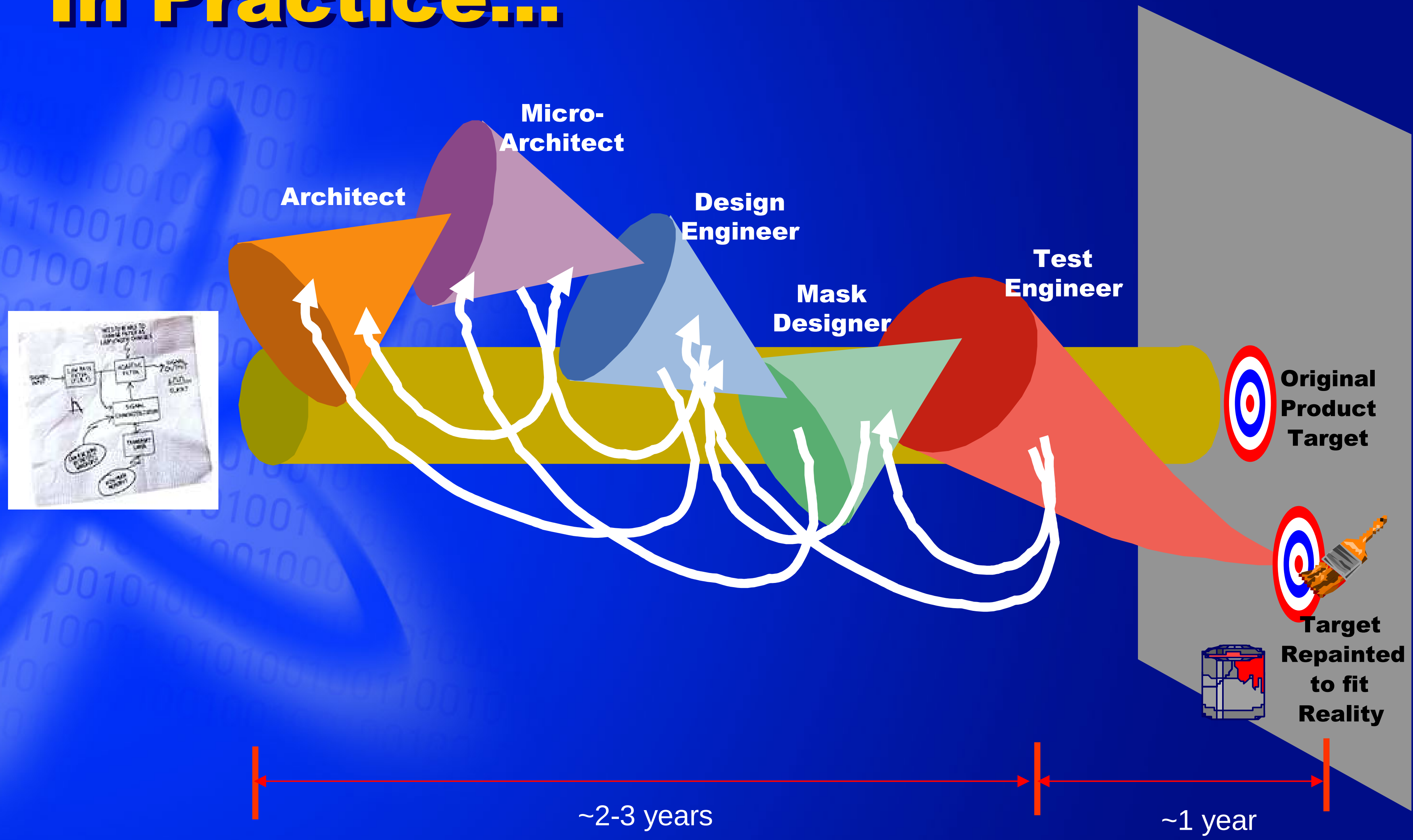


MAS: Micro-Architecture Specification

RTL: Register-Transfer Language

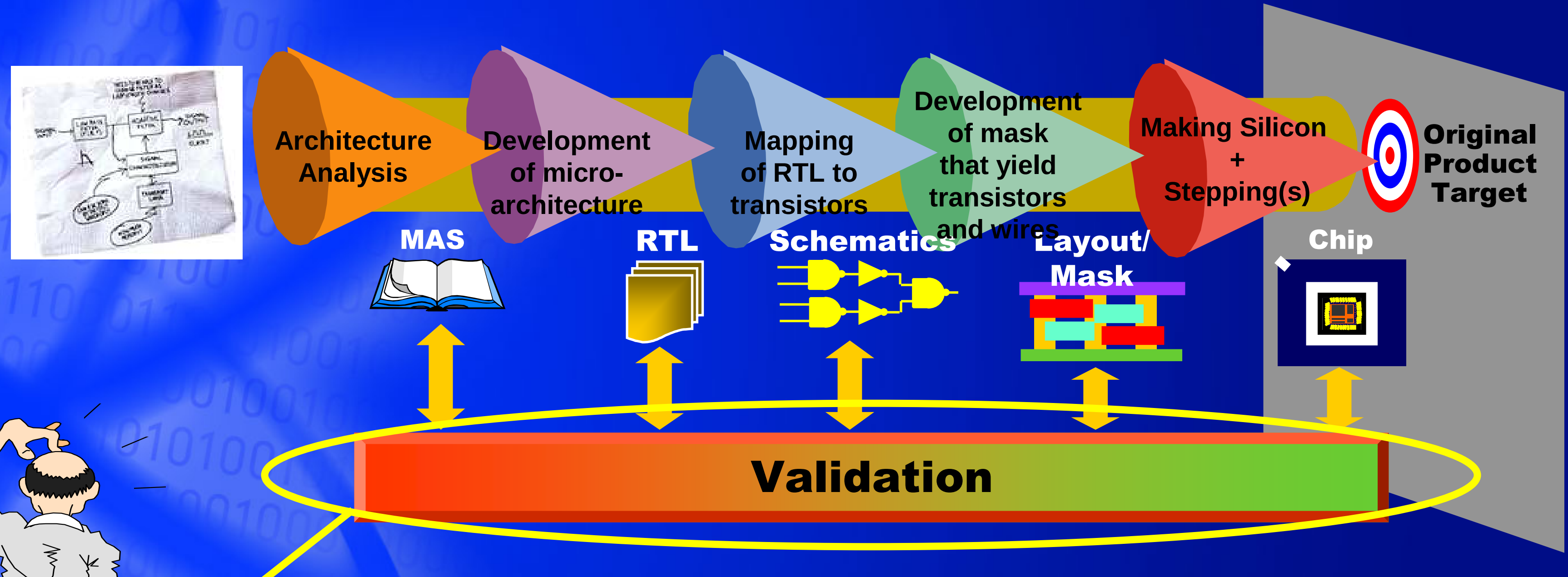
This is the theory…

In Practice...



Micro-Architect
Architect
Design Engineer
Mask Designer
Test Engineer
Original Product Target
Target Repainted to fit Reality
~2-3 years
~1 year

Validation



How to: 1) check we captured what we wanted
2) check that we did not make a mistake along the way

Evolution of a custom language

Stage One: Scripting & Implementation

- A generalized symbolic circuit simulator forms the core of our formal verification environment.
  - Symbolic Trajectory Evaluation engine
    - Combines partial order (lattice) modeling and symbolic expressions
- Binary Decision Diagrams tightly integrated into language
- An interpreted language is very helpful in driving such an engine
- We choose a pure (and very simple) lazy functional language as scripting language
  - Called fl

Example of fl usage: I

reFLect | Interrupt | Search: | Help

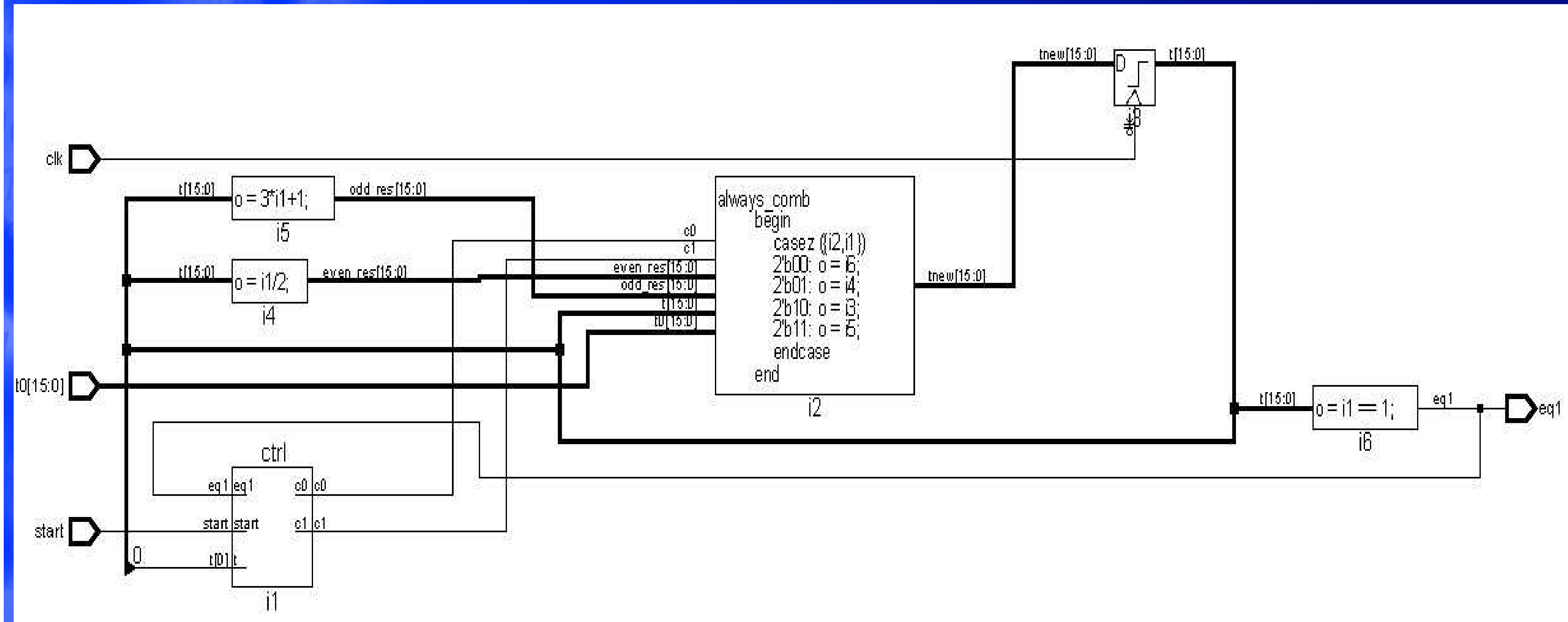
```
: let a = variable "a";

a::bool
: let b = variable "b";
b::bool
: a AND b;
b&a::bool

: let c =variable "c";
c::bool
: (a AND (b XOR c)) OR (NOT a OR b AND NOT c);
!c&b + c&!b + !a::bool

: NOT (NOT a OR NOT b) == NOT (a AND b);
F::bool

: NOT (NOT a OR NOT b) == (a AND b);
T::bool

: NOT (NOT a OR NOT b) <=> NOT (a AND b);
F::bool

: NOT (NOT a OR NOT b) <=> NOT (a XOR b);

a + b::bool
```

Example of fl usage: II

reFLect | Interrupt | Search: | Help

```
: NOT (NOT a OR NOT b) <=> NOT (a AND b);
F::bool

: NOT (NOT a OR NOT b) <=> NOT (a XOR b);

a + b::bool

:

:

:

: Quant_forall ["a","b"] (Quant_thereis ["c"] ((a = b) ==> c) <=> (a XOR b XOR c));
F::bool

: Quant_forall ["a","b"] (Quant_thereis ["c"] ((a = b) ==> c) ==> (a XOR b XOR c));
F::bool

: Quant_forall ["a","b"] (Quant_thereis ["c"] ((a = b) ==> c) <== (a XOR b XOR c));
T::bool

:

:
```

Example of fl usage: III

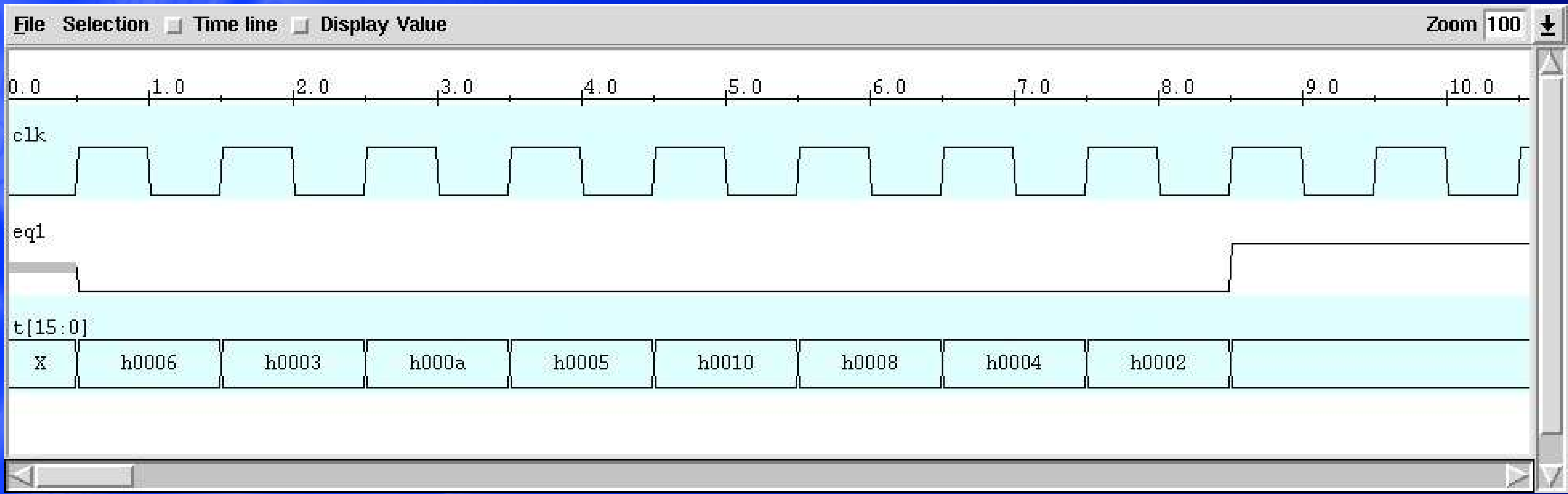


Circuit to evaluate the Collatz conjecture.

Example of fl usage: IV

```
: let start = {'start::bit};
start::bit
: let t0    = {'t0::word};
t0::word
: let ckt = pexlif2fsm (get_pexlif_in_window);
ckt::fsm
: let cycles n =
    [(T,"clk",F,2*i,2*i+1) | i in 0 upto (n-1)] @
    [(T,"clk",T,2*i+1,2*i+2) | i in 0 upto (n-1)]
;
cycles::int -> (bool # string # bool # int # int) list
: let N = 150;
N::int
: let ant = (cycles N) @
            (start isv '1 from 0 to 1)@
            (start isv '0 from 1 to (2*N))@
            (t0 isv '6 from 0 to 1)
;

ant::(bool # string # bool # int # int) list
: STE "-s" ckt [] ant [] (map (\n.n,0,2*N) (nodes ckt));
Time: 0
.Time: 1
.Time: 2
.Time: 3
```

Example of fl usage: V



Example of fl usage: VI

reFLect | Interrupt | Search: | Help

```
: let t0_var = {'a::word};
t0_var::word
: let t0_vars = word2bv t0_var;
t0_vars::bool list
: let ant = (cycles N) @
          (start isv '1 from 0 to 1)@
          (start isv '0 from 1 to (2*N))@
          (t0 isv t0_var from 0 to 1)
;
ant::(bool # string # bool # int # int) list
: STE "-s" ckt [] ant [] (map (\n.n,0,(2*N)) (nodes ckt));
Time: 0
.Time: 1
.Time: 2
.Time: 3
.Time: 4
.Time: 5
.Time: 6
.Time: 7
.Time: 8
.Time: 9
.Time: 10
.Time: 11
.GC: Marking, sweeping, done. Used=1683941(Shared=19584,Sat=1028) Freed:144802
Time: 12
```

Example of fl usage: VII

reFLect | Interrupt | Search: | Help

```
.Time: 295
.Time: 296
.Time: 297
.Time: 298
.Time: 299
.Time: 300
T::bool

: let out_at_end = get_trace_val ckt "eq1" (2*N-1);
out_at_end::bool # bool
: let fail = out_at_end = (F,T);
fail::bool
: pick_example F fail;
substitution list:

 a[15:0]: 0000000000000000
::example

: pick_example F (fail AND (t0_vars != (int2bv 0)));
substitution list:

 a[15:0]: 0000001011111011
::example

: pick_example F (fail
                AND (t0_vars != (int2bv 0))
                AND NOT (last t0_vars)
);
substitution list:

 a[15:0]: 0000010111110110
::example

:
```

Example of fl usage: VIII

reFLect | Interrupt | Search: | Help

```
:

:

: lettype model = Model
                  {rel::bool}
                  {c_vars :: bool}
                  {n_vars :: bool}
;

Model::bool -> bool -> bool -> model
  load_model::string -> bool -> string -> model
  save_model::string -> model -> bool -> model
: let cAX model set =
    let set = bdd_current_next set in
    quant_forall model:>n_vars (model:>rel ==> set)
;
cAX::model -> bool -> bool
: let cEG model set =
    letrec EGr cur =
        let new =
            let cur' = bdd_current_next cur in
            quant_forall model:>n_vars (set OR (model:>rel ==> cur'))
        in
        if new == cur then cur else EGr new
    in
    EGr F
;

cEG::model -> bool -> bool
:
```

Example of fl usage: IX

```
: // Use symbolic simulation to extract next-state relation
let model16 = Model R  c_vars n_vars;
model16::model
: let set0 = t_vars = (int2bv 1);
set0::bool
: let ok_set = cEG model16 set0;
ok_set::bool
: ok_set;
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
Iteration 11
Iteration 12
Iteration 13
Iteration 14
Iteration 15
Iteration 16
Iteration 17
Iteration 18
Iteration 19
Iteration 20
Iteration 21
Iteration 22
Iteration 23
```

Example of fl usage: X

reFLect | Interrupt | Search: | Help

```
Iteration 206
Iteration 207
Iteration 208
Iteration 209
Iteration 210
Iteration 211
Iteration 212
Iteration 213
Iteration 214
Iteration 215
Iteration 216
Iteration 217
Iteration 218
Iteration 219
Iteration 220
Iteration 221
Iteration 222
Iteration 223
Iteration 224

t[12]&t[13]&t[15] + t[11]&t[13]&t[14] + t[10]&t[13] + t[6]&t[15] + t[5]&t[13]&t[
14] OR ...
::bool

: pick_example F ((NOT ok_set) AND (t_vars != int2bv 0));
substitution list:

 t[15:0]: 0011100011100011
::example

:
```

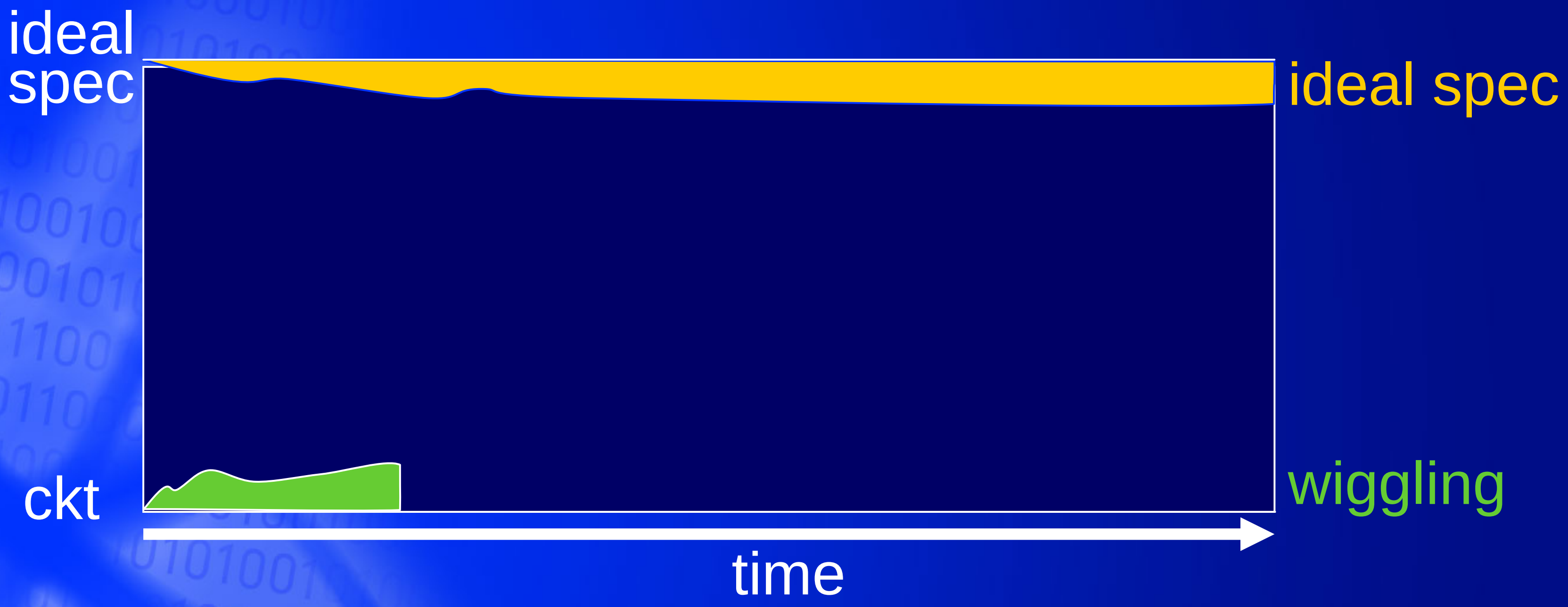
Stage Two: Property Specification

- fl with BDDs started to look like a very useful specification language as well.
- To make this even better, we extended the language by allowing conditionals to be symbolic
  - Since we could only represent Boolean functions, the “then” and “else” sides must have the same “shape”
- The extended (evolved) fl now served as:
  - Property specification language
  - Implementation language for FPV & FEV tools
  - Scripting language for the end-user

Verification Challenge

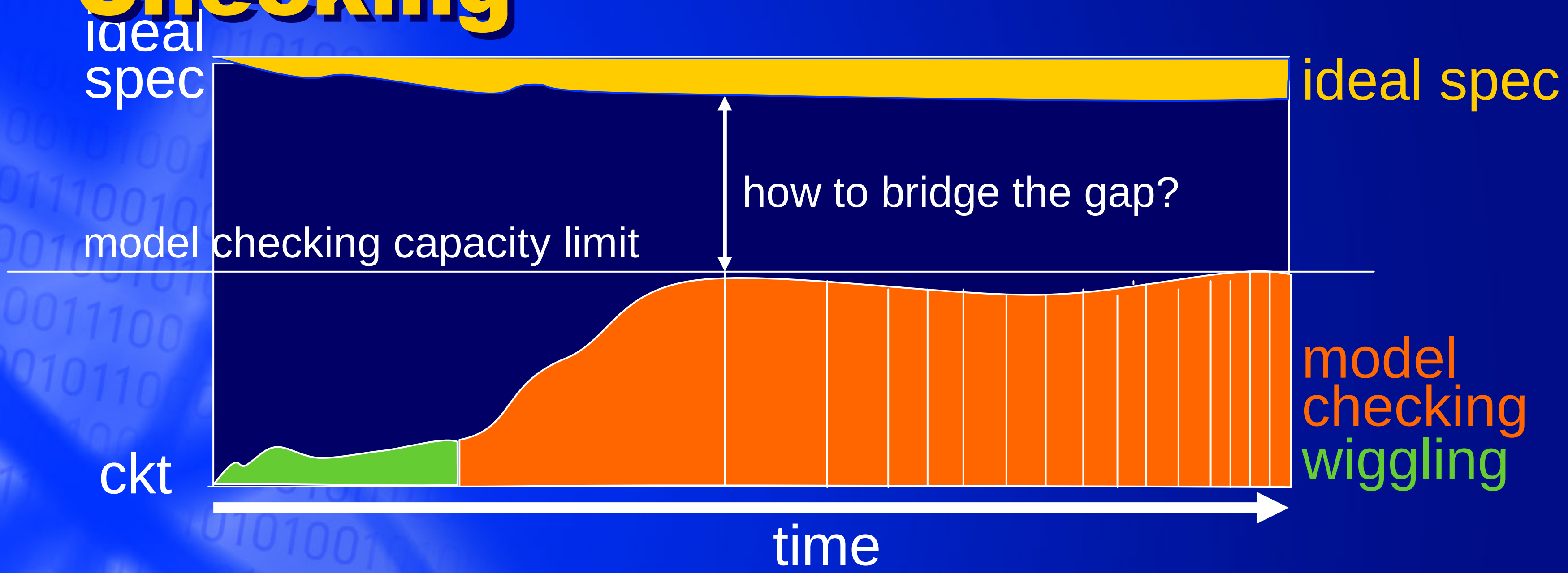
TASK: Bridge the gap from the circuit to the ideal specification in the minimum amount of time & cost.

Initial Phases of Verification



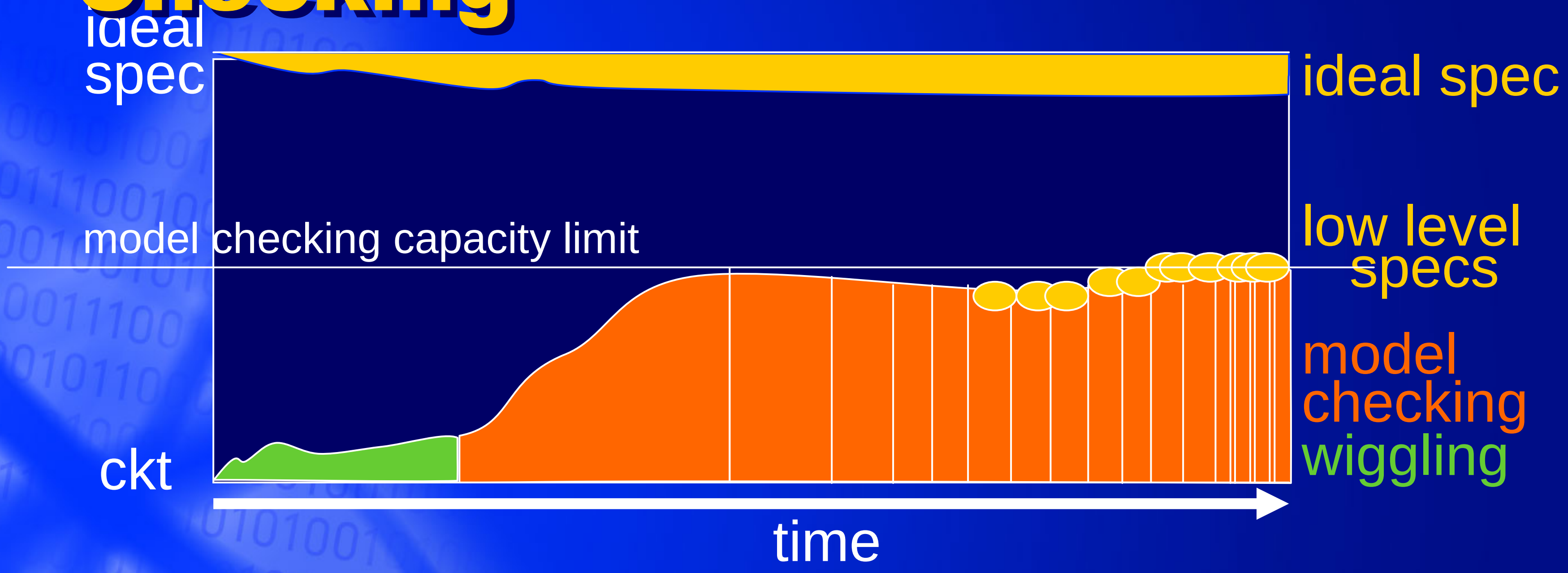
- Sketch initial specification
- Get circuit to “wiggle” (respond to simple inputs)

Verification With Only Model Checking



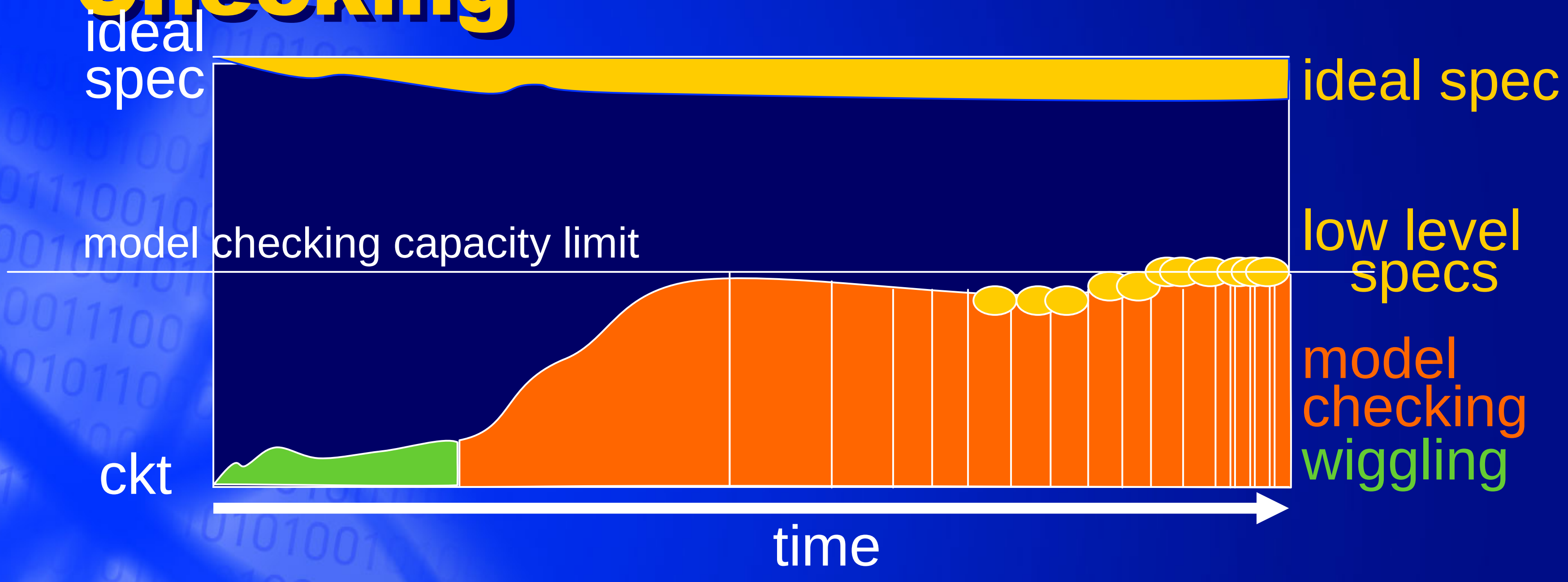
- With industrial circuits:
  very quickly encounter model checking capacity limits

Verification With Only Model Checking



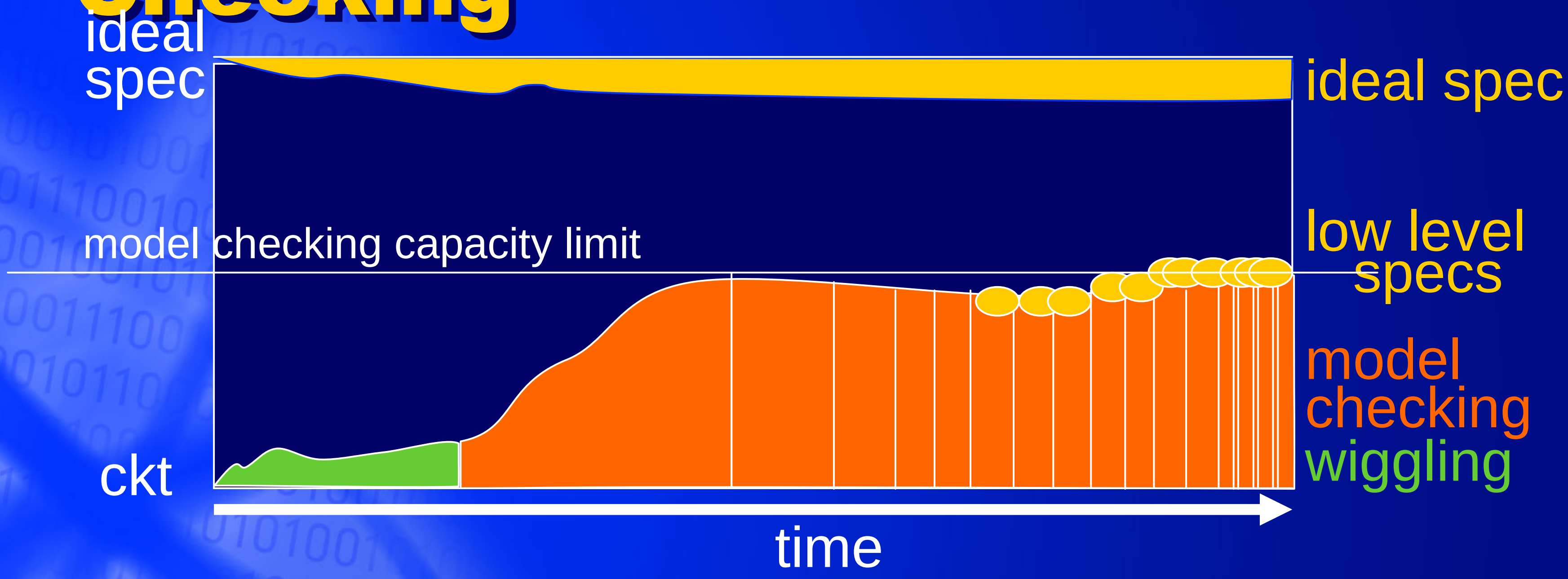
- Forced to bridge the gap with:
  - large collection of low-level specifications
  - informal checks/hand proofs against ideal specification

Verification With Only Model Checking



- Forced to bridge the gap with:
  - large collection of low-level specifications
  - informal checks/hand proofs against ideal specification
    - long tedious (uninteresting) hand proofs...

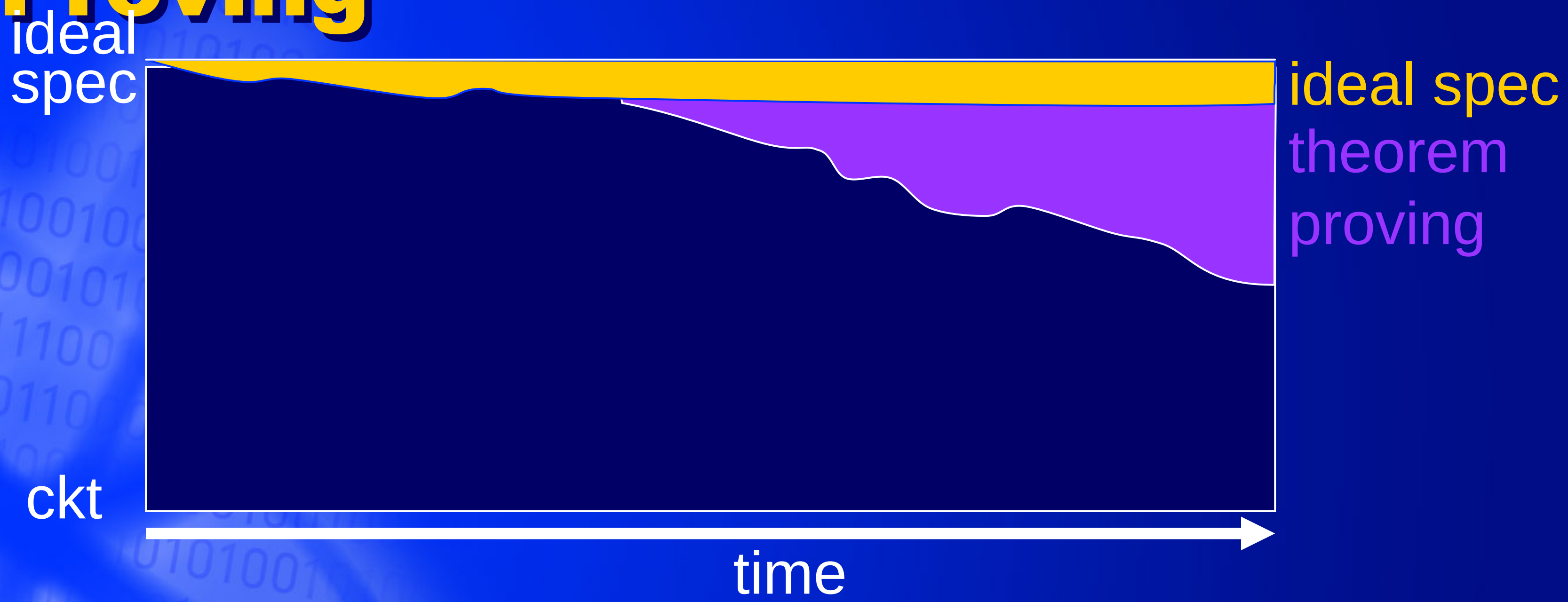
Verification With Only Model Checking



- Forced to bridge the gap with:
  - large collection of low-level specifications
  - informal checks/hand proofs against ideal specification
    - long tedious (uninteresting) hand proofs...

      ...usually wrong…

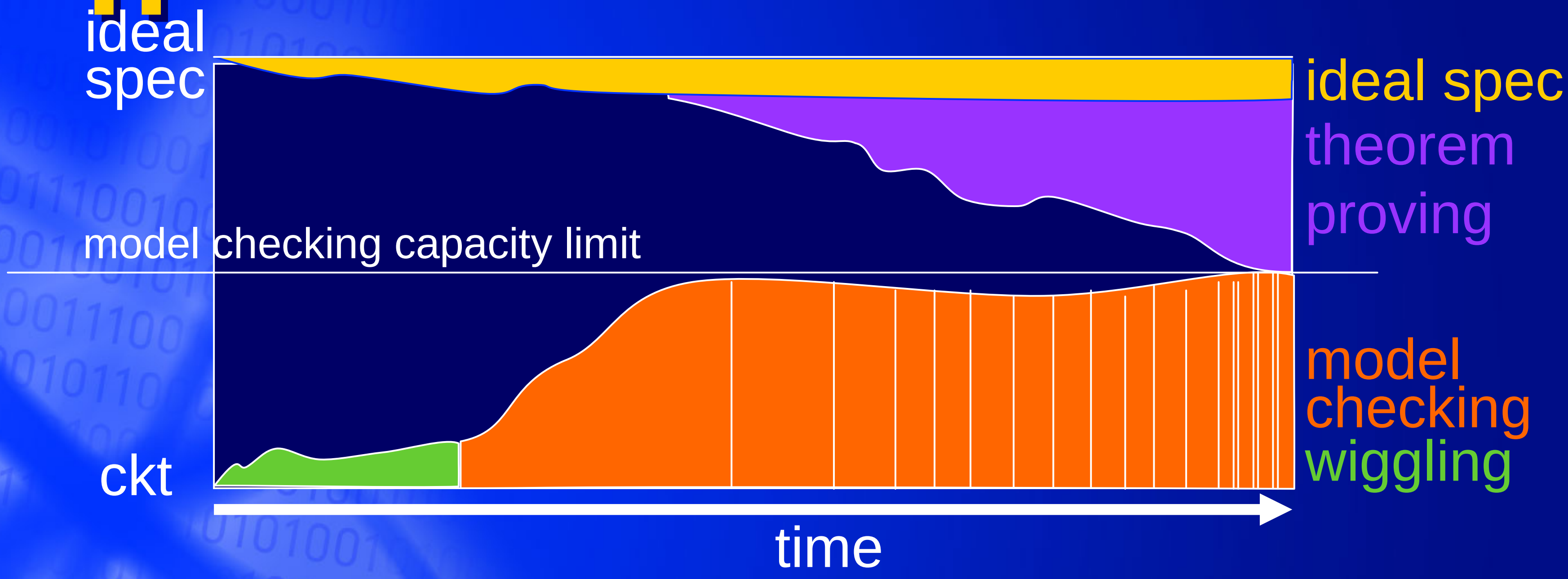
Verification with only Theorem Proving



Theorem proving (with significant manual effort) can establish correctness against abstract circuit models.

- Abstract model often significantly simpler than actual HW
- Abstract model is not verified/verifiable against actual HW

Verification with Combined MC & TP



Theorem proving provides formal link from model checking results to ideal specification.

Stage Three: Term Language

- HOL-Voss (separate theorem proving and model checking tools):
  - HOL provided TP, fl provided model checking capabilities
  - fl was used as an evaluation engine for HOL functions
  - Very difficult to use, common case slow, overkill
- VossProver (deep embedding of logic in fl)
  - Idiot-savant prover for combining model checking results
  - Easier to use, but still extra layer of interpretation
  - Very cumbersome to extend
- Reflection
  - Introduced reflection in fl so that fl programs can manipulate other fl programs.
  - No overhead for end user, trivial to extend, some “noise” in the theorem proving from fl (e.g., print statements etc.)

Stage Four: Modeling Language

- The most recent enhancement to fl has been the incorporation of more flexible syntax/semantics
- The main purpose is to make it possible to provide a practical language for High-level modeling that has an “acceptable” syntax to end users
  - Shallow embedding for efficiency
  - Reflection provides a deep embedding
  - Programmable syntax makes domain-specific language development easier
- The main challenge is error reporting!!!

Lessons Learned

Why was it successful?

- Forte provided a unified environment that made it easy to build, extend, and use FV tools in.
- There was a natural fit in the semantic model for specifications (functional)
- The performance of the interpreter was not on the critical path for most applications
- The system was easily and safely extensible by the (experienced) user.
- Forte provided a major new capability!
  - The cost of “swallowing” fl was paid back by the new capabilities.

A new language is successful only if it is part of a system that solves a previously unsolved problem.

New languages are needed regularly to solve previously unsolved problems…

Backup Slides

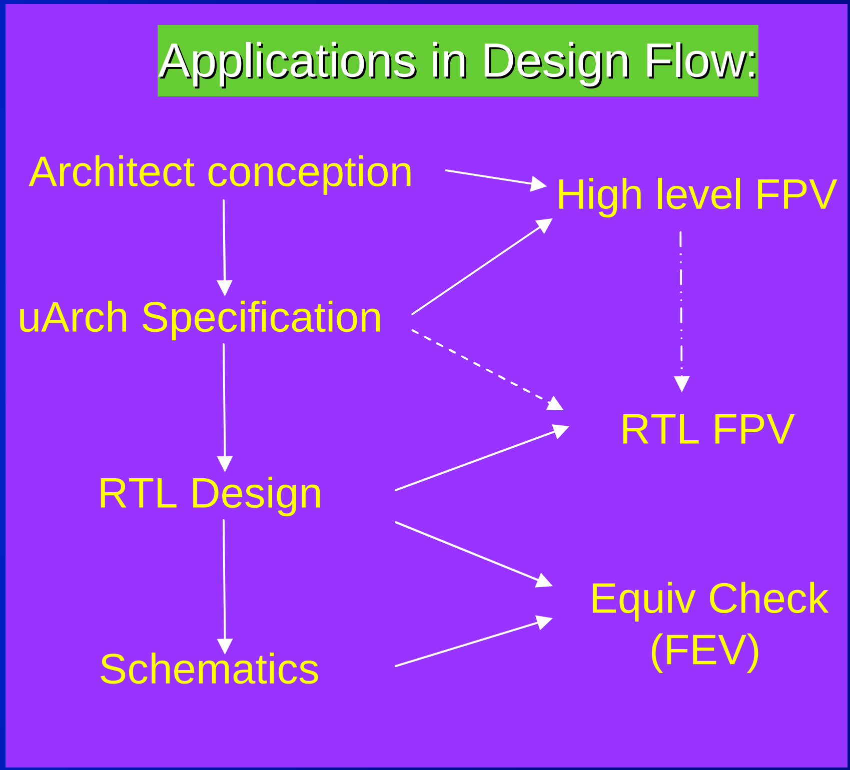
Coverage

Today's focus

Coverage		Pro	Con
100 % Covered	Formal Verification	•100% coverage •Proves absence of bugs	•Requires special skills •Constrained by complexity
	Directed Random Tests	•Targets areas most likely to be of concern •Greatly reduces cycle requirements •Develops strong uArch knowledge	•Requires strong uArch knowledge
	Generic Random Tests	•After generator created, easy to write •Requires little uArch knowledge •Can create things no one would ever think of	•Requires almost ∞ cycles / time •Difficult / impossible to avoid broken features
Low % Covered	Directed Tests	•Easy to write •Easy to understand •Easy to reuse	•Requires almost ∞ number of tests •Difficult to hit uArch conditions

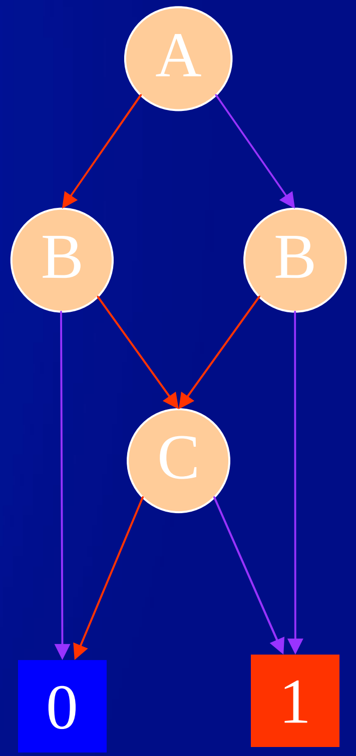
Formal Verification

- Exhaustive simulation is infeasible.
  - cannot prove the absence of bugs
- Broad classification:
  - Formal equivalence verification: FEV
    - Prove two models are the same
    - Highly automated
    - In widespread use
  - Formal property verification: FPV
    - Prove model satisfy some property
    - User driven
    - Primarily used in high risk areas



Ordered Binary Decision Diagrams: BDDs

- Canonical representation of Boolean functions
- Efficient algorithms for AND,OR,NOT, quantification, image computation, etc.
- Variable ordering critical
  - Static heuristics
  - Dynamic variable re-ordering
- Handles ~80% of all equivalence verification tasks.
- With major effort, can push to 90%
- Most modern FV tools use BDDs or a combination of BDDs and SAT solvers

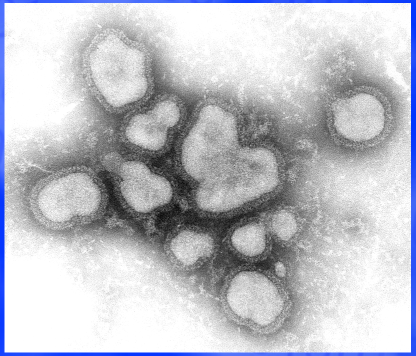


Example of Property: FP Add

```
// Feldman & Retter, Computer Architecture
// (McGraw-Hill,94) pp. 489-491
let ADDmodel pc rc in1 in2 =
  ...
  // Find the amount of shift needed
  let diff = ex2 '-' ex1  in
  let rsh = MINv diff (int2bv 68) in
  // Do the shift
  let sgf1' = srshift 68 rsh (sgf1@[F]) in
  let sgf2' = sgf2@[F] in
  // Perform the sum (or subtract)
  let add = (sign fp1 = sign fp2) in
  let sum = if add then (sgf2' '+' sgf1')
                   else (sgf2' '-' sgf1')
  // Now perform roundin
  ...
```

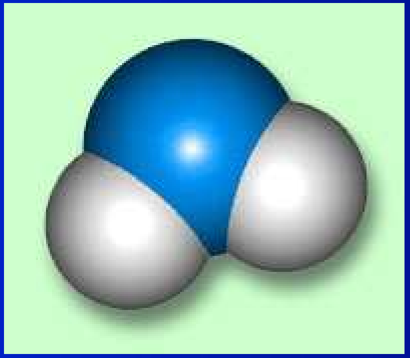
Pop Quiz

- Order the following in order of size (smallest first)

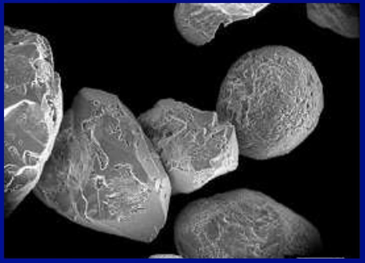


Influenza A virus

Transistor in high volume microprocessor in 2009



Water molecule



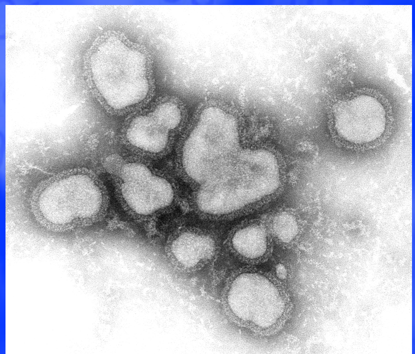
Grains of sand

Answers:

Answer to Pop Quiz

- Order the following in order of size (smallest first)

~100nm



Influenza A virus

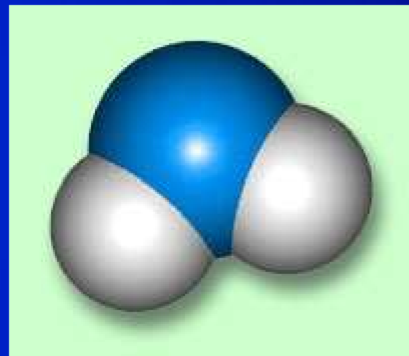
3

~30nm

Transistor in high volume microprocessor in 2009

2

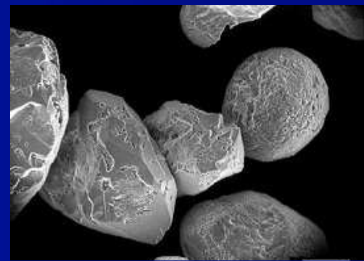
~0.3nm



Water molecule

1

~100,000nm



Grains of sand

4