Software Security and Proved Compilation

Alexandre Pilkiewicz

Gallium Team

Gallium Team

Gallium Team

- Broad area: Programming languages

Gallium Team

- Broad area: Programming languages
- Specifically:
  - Design of (functional) languages
  - Compilation
  - Type system
  - Programming safety

Gallium Team

- Broad area: Programming languages
- Specifically:
  - Design of (functional) languages
  - Compilation
  - Type system
  - Programming safety

Programing Safety?

- Lots of bugs everywhere. OK? Not OK?

Programing Safety?

- Lots of bugs everywhere. OK? Not OK?

Programing Safety?

- Lots of bugs everywhere. OK? Not OK?

Software Kind

Sensitive	Network Security	Data Security		
Ordinary	PC/Phone Software			

Programing Safety?

- Lots of bugs everywhere. OK? Not OK?

Software Kind

Critical	Railways	Medical	Nuclear plants	Airplane
Sensitive	Network Security	Data Security		
Ordinary	PC/Phone Software			

Programing Safety?

- Lots of bugs everywhere. OK? Not OK?

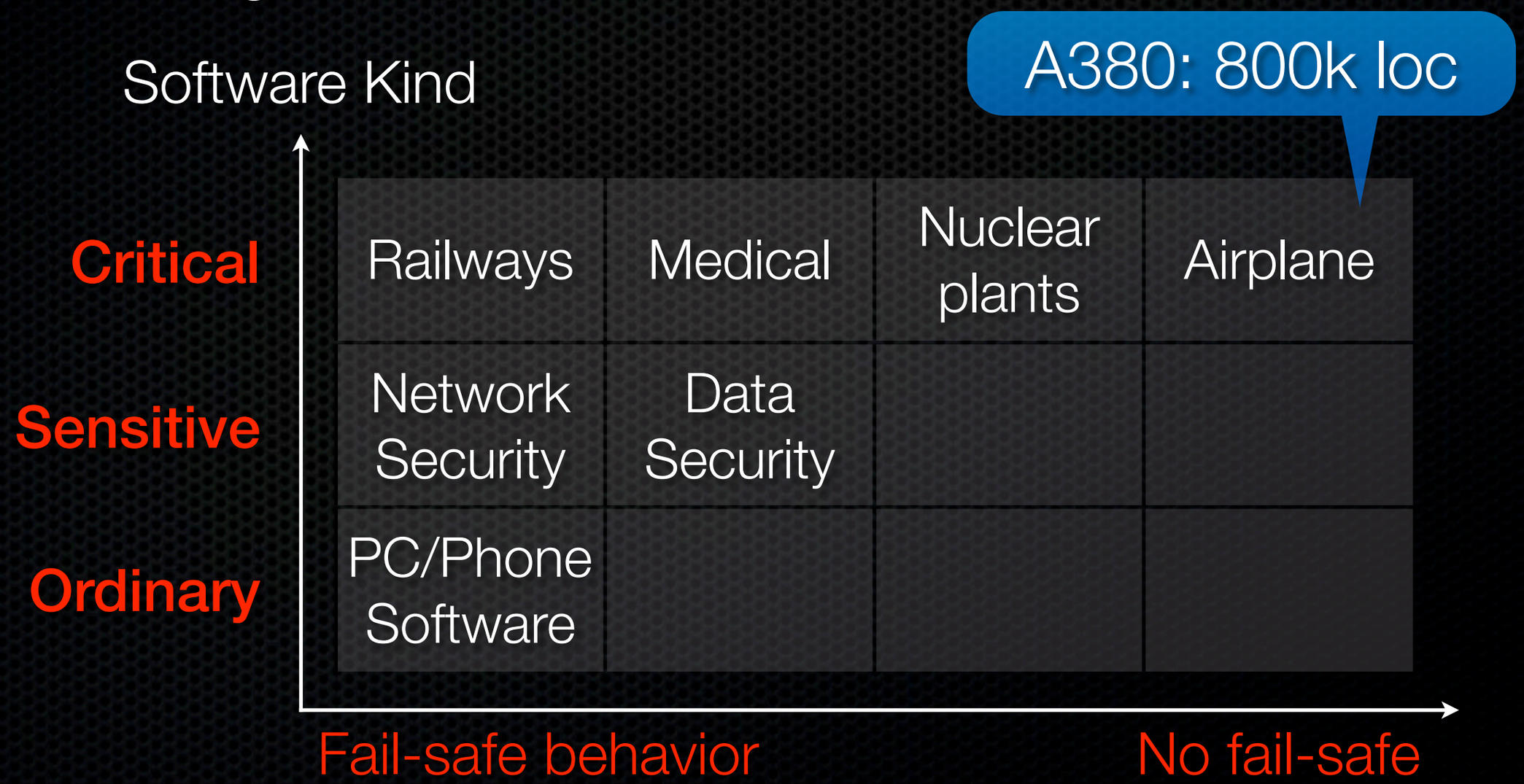
A380: 800k loc

Software Kind

Critical	Railways	Medical	Nuclear plants	Airplane
Sensitive	Network Security	Data Security		
Ordinary	PC/Phone Software			

Programing Safety?

- Lots of bugs everywhere. OK? Not OK?



Medical: Therac 25 radiation therapy machine (1985-87)

Medical: Therac 25 radiation therapy machine (1985-87)

- Two modes
  - Electron beam therapy (direct low dose)
  - Megavolt X-ray (high dose on a metallic target)

Medical: Therac 25 radiation therapy machine (1985-87)

- Two modes
  - Electron beam therapy (direct low dose)
  - Megavolt X-ray (high dose on a metallic target)
- Bug: direct high dose, no target

Medical: Therac 25 radiation therapy machine (1985-87)

- Two modes
  - Electron beam therapy (direct low dose)
  - Megavolt X-ray (high dose on a metallic target)
- Bug: direct high dose, no target

Medical: Therac 25 radiation therapy machine (1985-87)

- Two modes
  - Electron beam therapy (direct low dose)
  - Megavolt X-ray (high dose on a metallic target)
- Bug: direct high dose, no target

Transportation: easy to gain full access to a car

Checkoway et al.

Transportation: easy to gain full access to a car

Checkoway et al.

- Easy:
  - Playing a CD
  - Accessing the network of car repair shops
  - Calling the car via cell phone network

Transportation: easy to gain full access to a car

Checkoway et al.

- Easy:
  - Playing a CD
  - Accessing the network of car repair shops
  - Calling the car via cell phone network
- Full access
  - Unlock, start engine
  - Eavesdrop conversations
  - Cruise control
  - (De)activate brakes

Transportation: easy to gain full access to a car

Checkoway et al.

- Easy:
  - Playing a CD
  - Accessing the network of car repair shops
  - Calling the car via cell phone network
- Full access
  - Unlock, start engine
  - Eavesdrop conversations
  - Cruise control
  - (De)activate brakes

Solutions? Some classics:

Solutions? Some classics:

- Precise coding rules (no strcpy...)

Solutions? Some classics:

- Precise coding rules (no strcpy...)
- Careful review

Solutions? Some classics:

- Precise coding rules (no strcpy...)
- Careful review
- Testing, testing, testing

Solutions? Some classics:

- Precise coding rules (no strcpy...)
- Careful review
- Testing, testing, testing
  - Unit tests

Solutions? Some classics:

- Precise coding rules (no strcpy...)
- Careful review
- Testing, testing, testing
  - Unit tests
  - Full system tests

Solutions? Some classics:

- Precise coding rules (no strcpy...)
- Careful review
- Testing, testing, testing
  - Unit tests
  - Full system tests

DO-178

Testing is not always cheap

Testing is not always cheap

Testing is not always cheap

Testing is not always cheap

Testing is not always cheap

Testing is not always cheap

Testing is not always cheap

Ariane 5 launch
4 June 1996
$130 million direct cost

Testing is not exhaustive

Testing is not exhaustive

- Back to Therac 25 radiation machine

Testing is not exhaustive

- Back to Therac 25 radiation machine
  - Has been tested

Testing is not exhaustive

- Back to Therac 25 radiation machine
  - Has been tested
  - Sequence of keystrokes in 8 seconds -> bug

Testing is not exhaustive

- Back to Therac 25 radiation machine
  - Has been tested
  - Sequence of keystrokes in 8 seconds -> bug
  - New operator -> slow -> no bug

Testing is not exhaustive

- Back to Therac 25 radiation machine
  - Has been tested
  - Sequence of keystrokes in 8 seconds -> bug
  - New operator -> slow -> no bug
  - Better operator -> fast -> dead patient

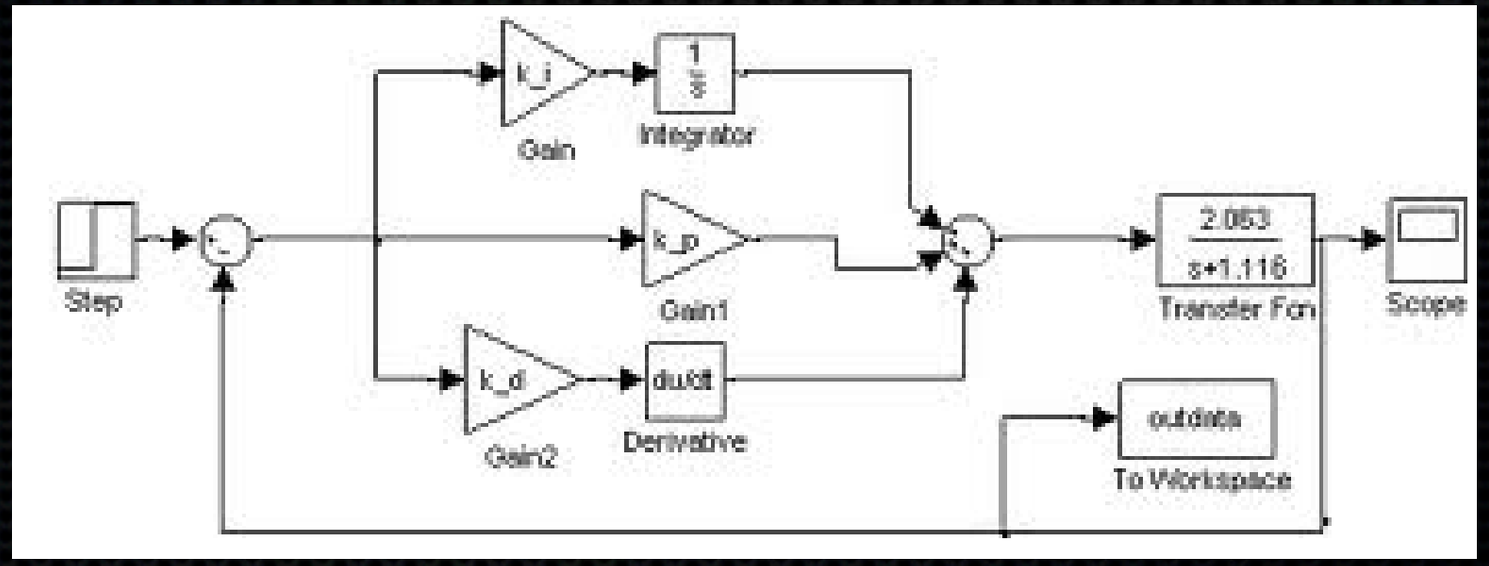
More systematic tools

More systematic tools

Simulink,
Scade

More systematic tools

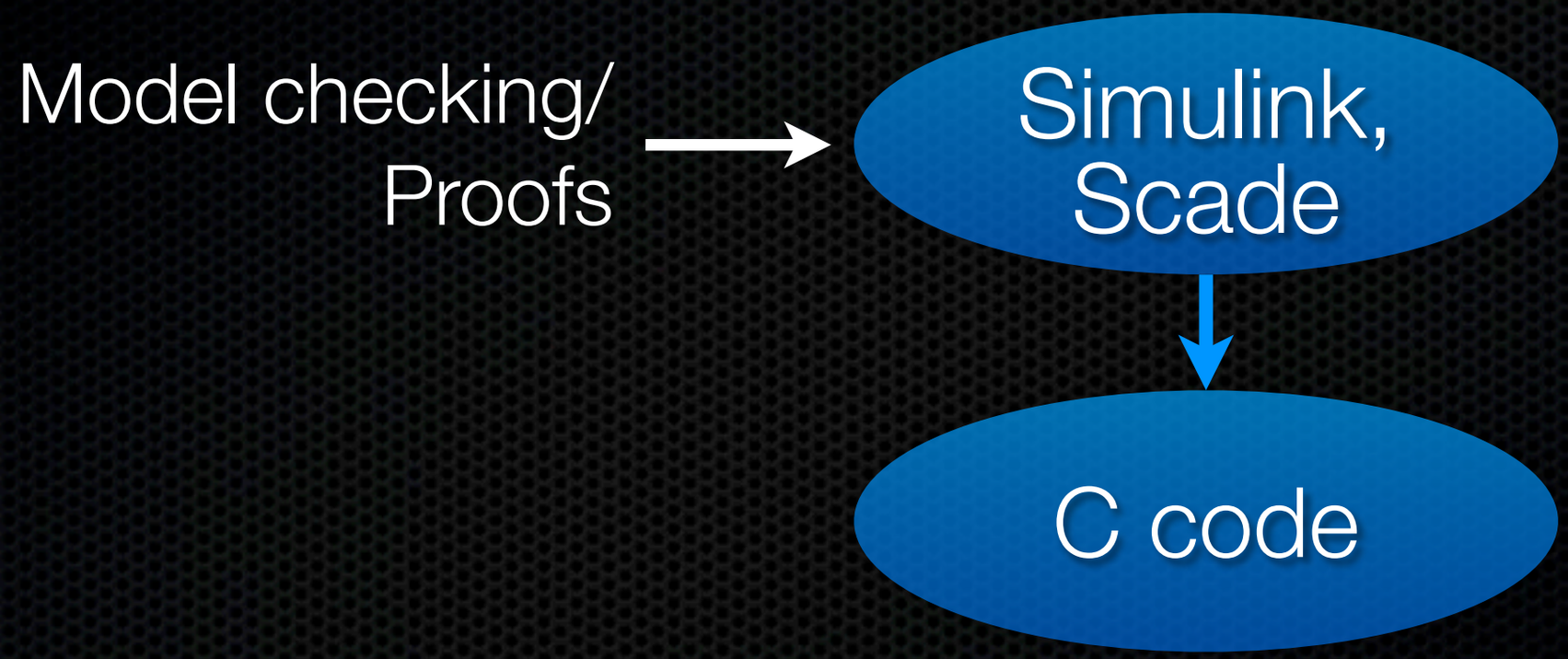
Simulink,
Scade



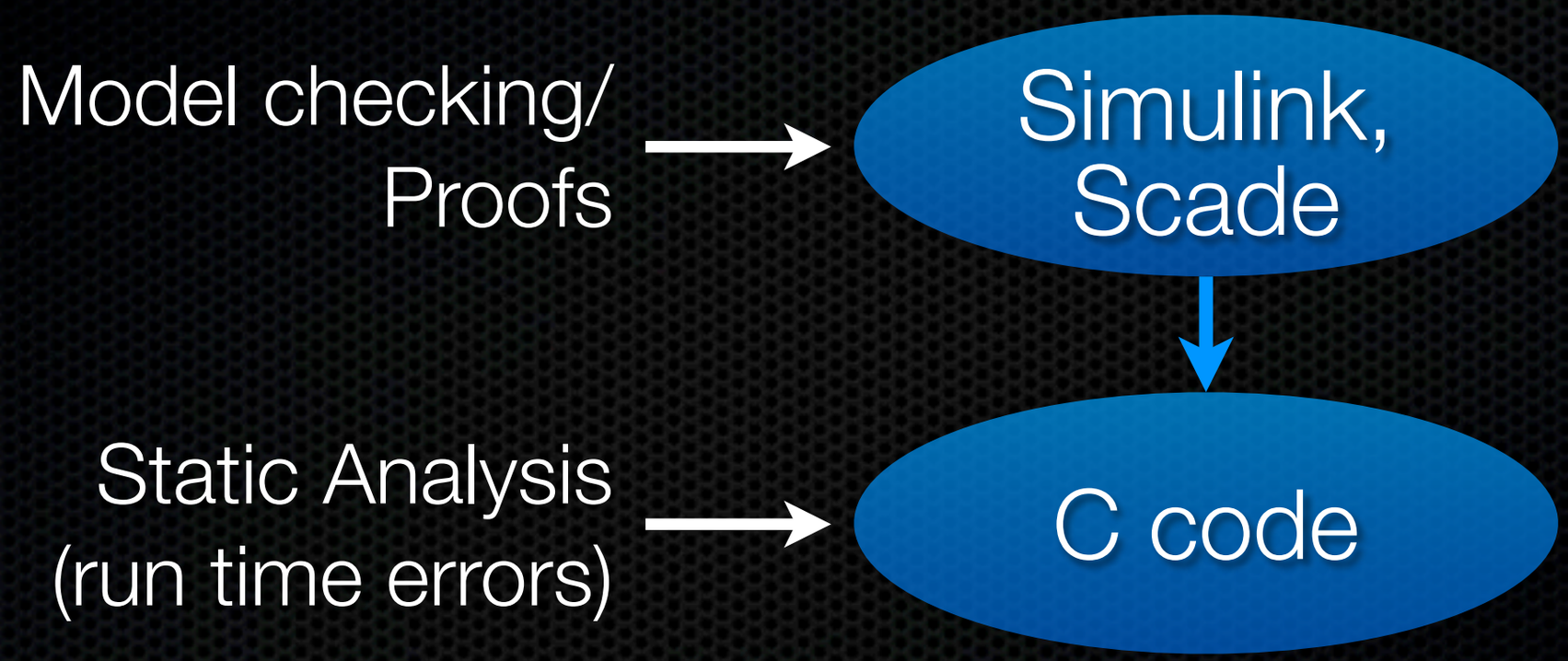
More systematic tools

Model checking/
Proofs → Simulink,
Scade

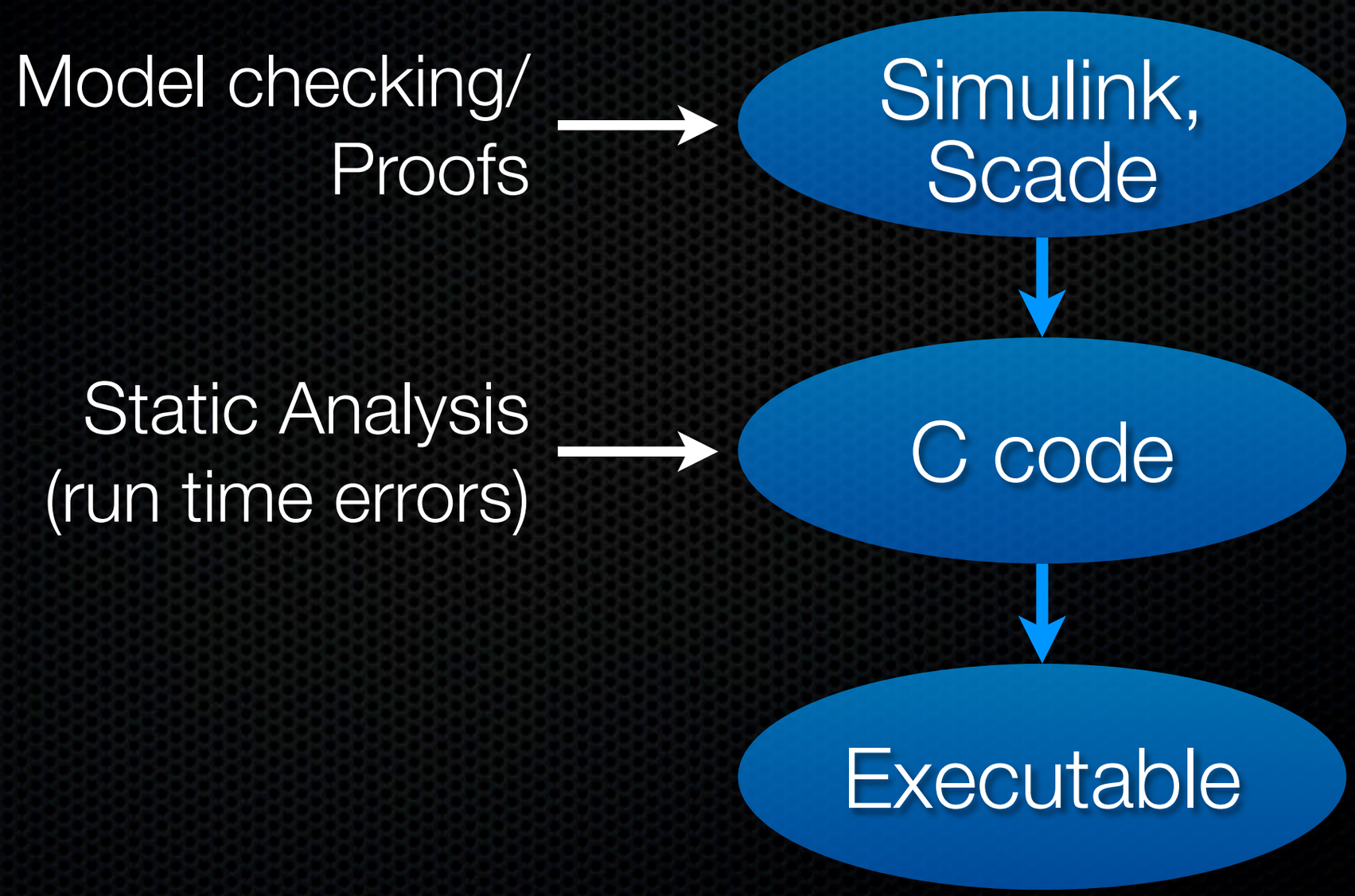
More systematic tools



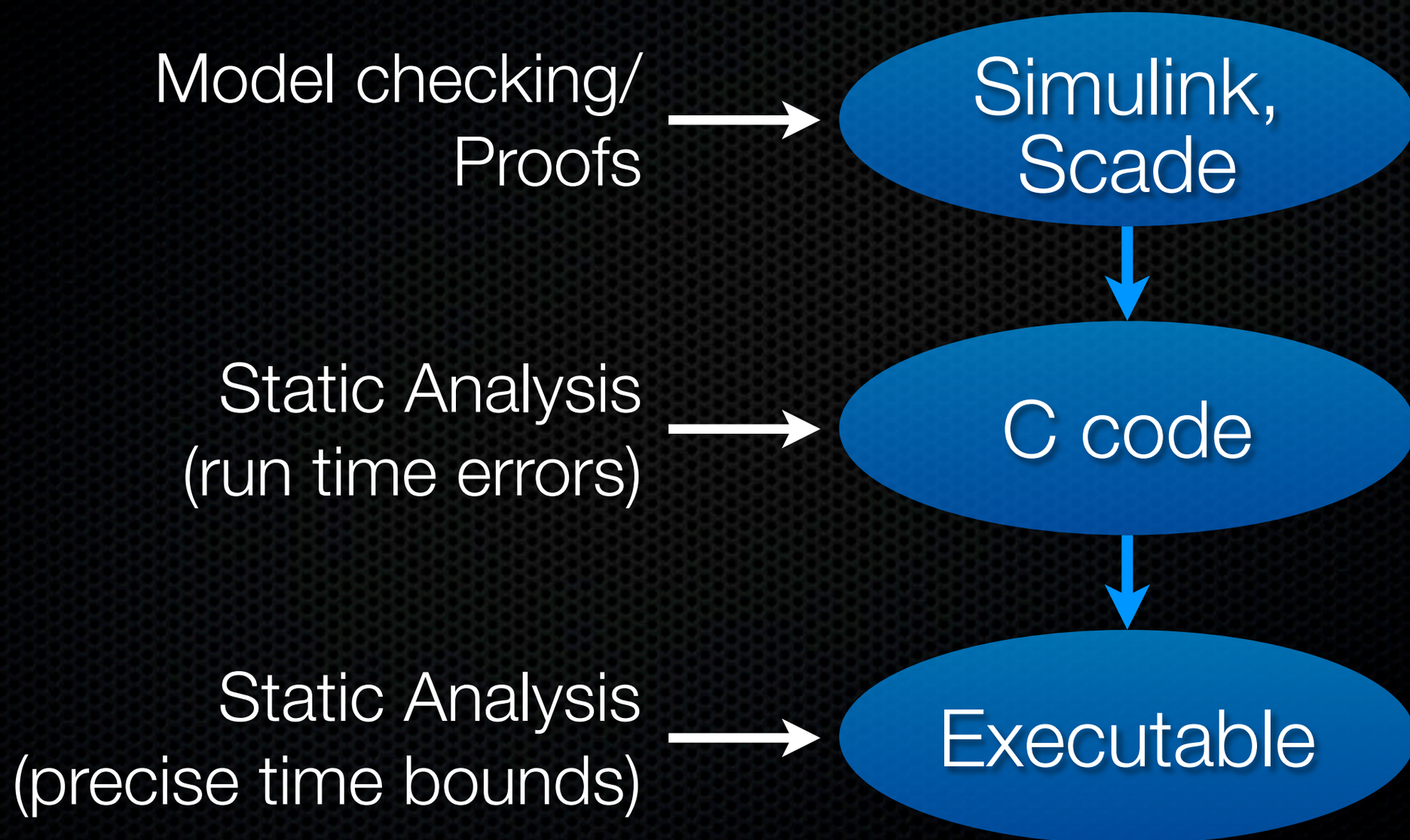
More systematic tools



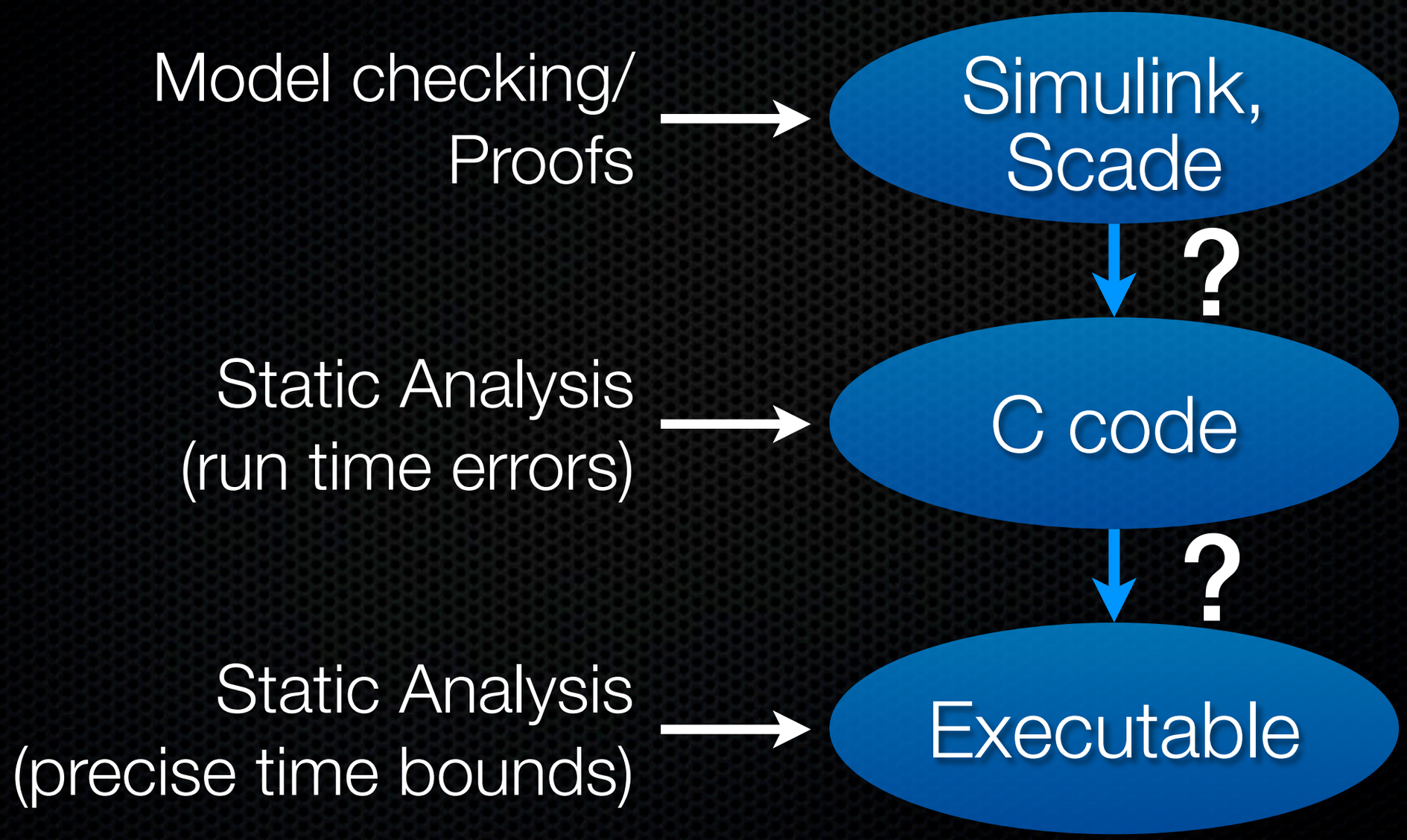
More systematic tools



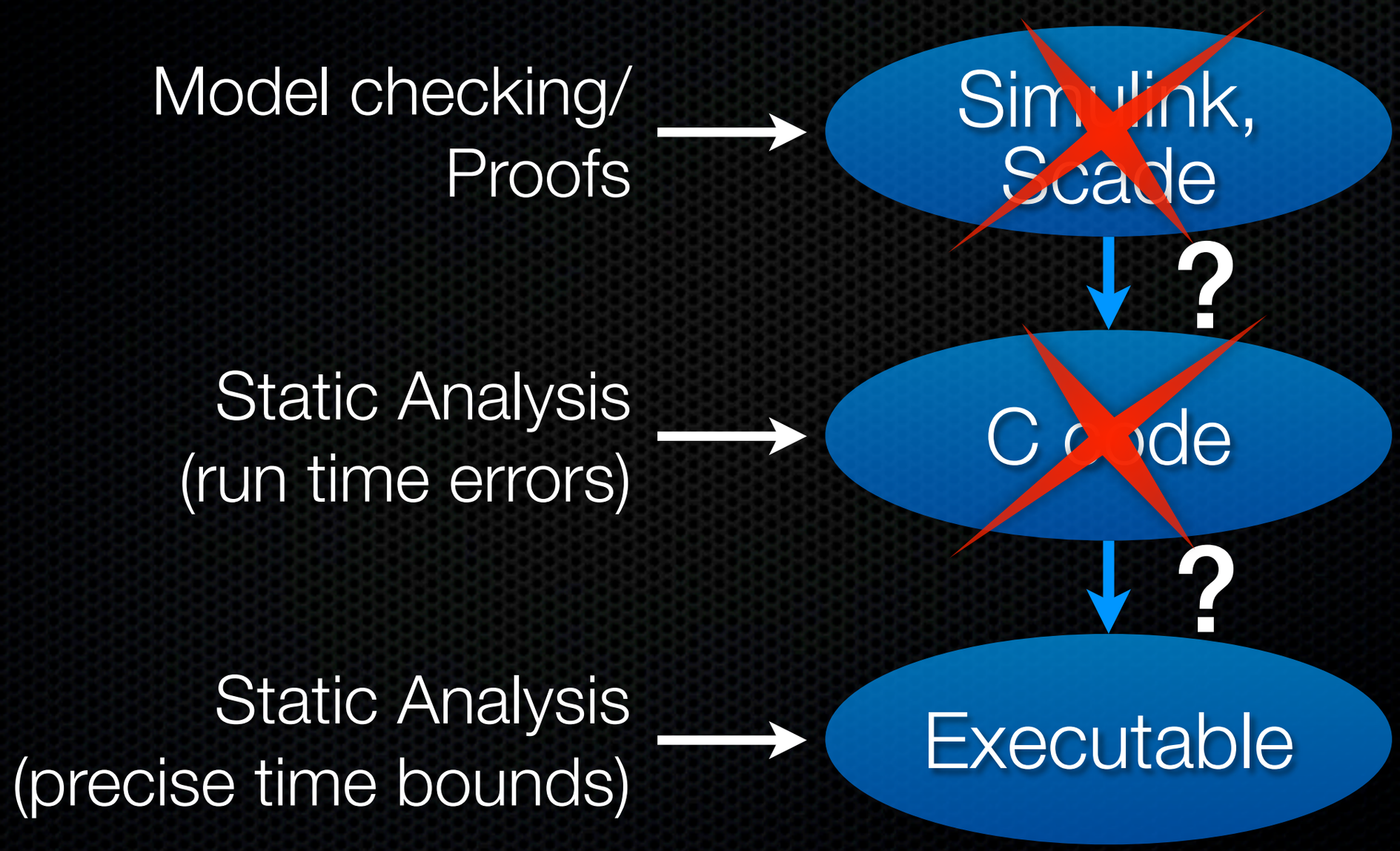
More systematic tools



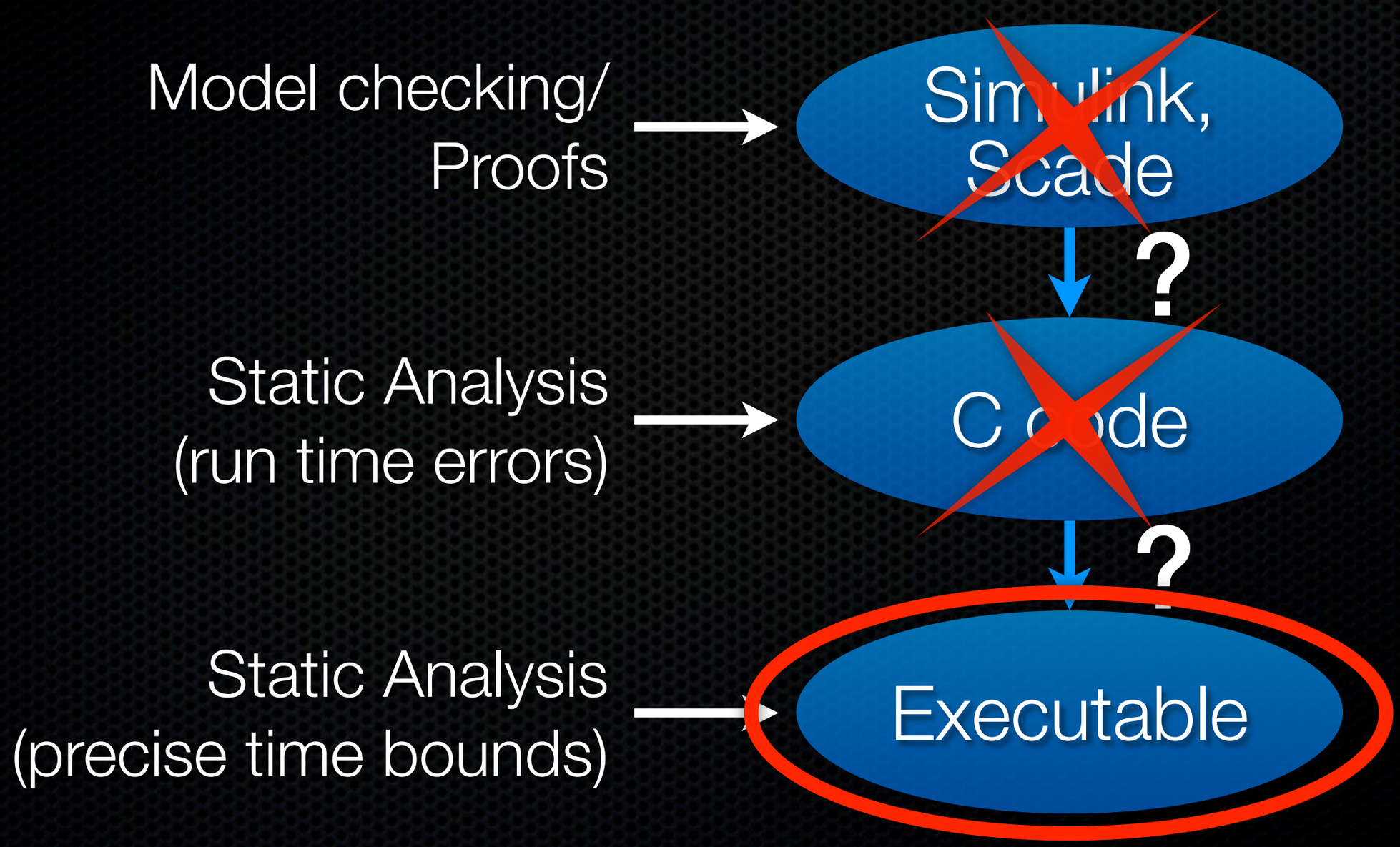
More systematic tools



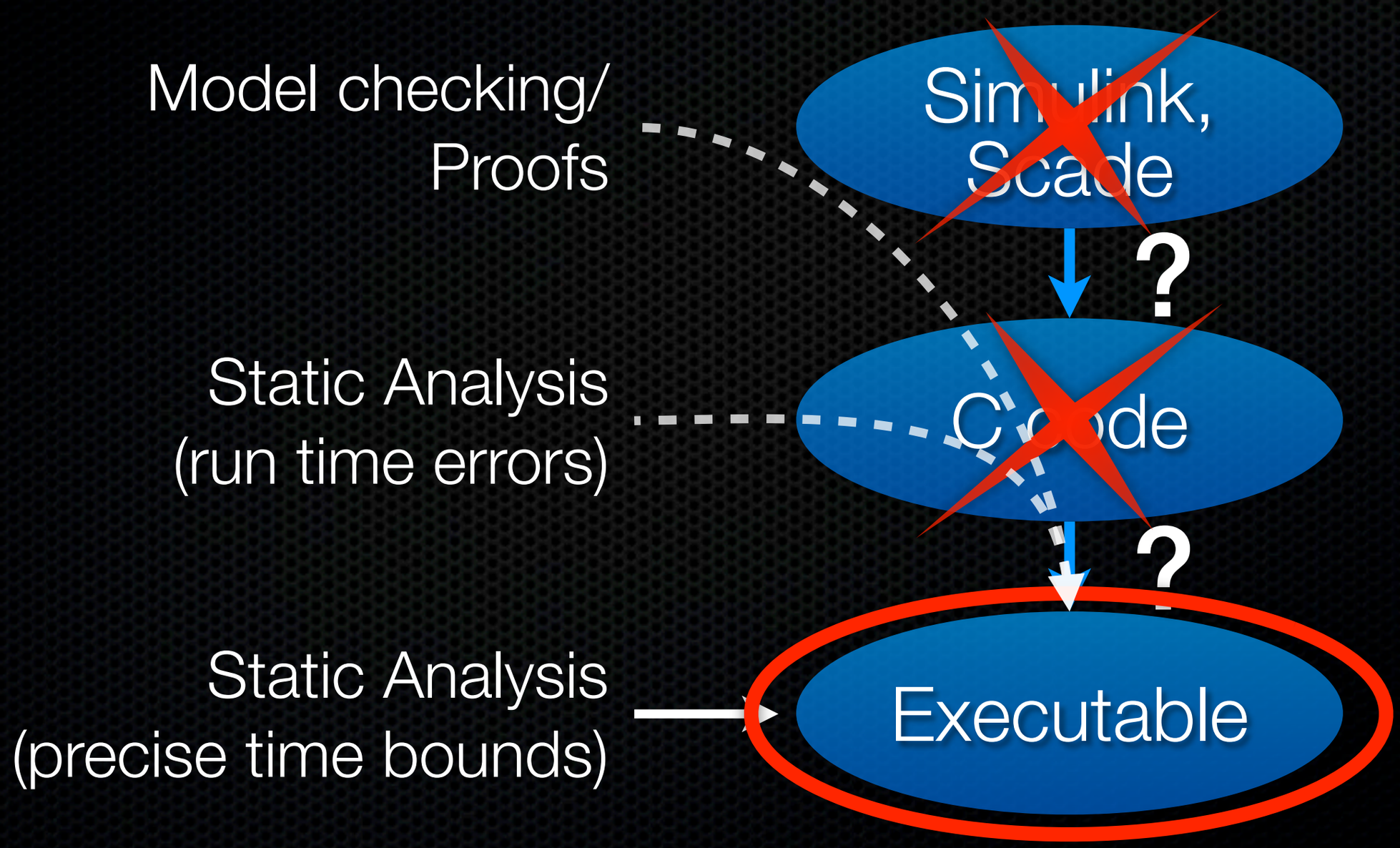
More systematic tools



More systematic tools



More systematic tools



How complicated is a compiler?

```
double dotproduct(int n, double * a, double * b) {
  double dp = 0.0;
  int i;
  for (i = 0; i < n; i++)
    dp += a[i] * b[i];
  return dp;
}
```

How complicated is a compiler?

```
double dotproduct(int n, double * a, double * b) {
  double dp = 0.0;
  int i;
  for (i = 0; i < n; i++)
    dp += a[i] * b[i];
  return dp;
}
```

$$a \cdot b = \sum_{i=0}^{n-1} a_i b_i$$

How complicated is a compiler?

```
double dotproduct(int n, double * a, double * b) {
  double dp = 0.0;
  int i;
  for (i = 0; i < n; i++)
    dp += a[i] * b[i];
  return dp;
}
```

$$a \cdot b = \sum_{i=0}^{n-1} a_i b_i$$

How complicated is a compiler?

```
double dotproduct(int n, double * a, double * b) {
  double dp = 0.0;
  int i;
  for (i = 0; i < n; i++)
    dp += a[i] * b[i];
  return dp;
}
```

$$a \cdot b = \sum_{i=0}^{n-1} a_i b_i$$

Compiled with Tru64/Unix and manually translated back to C...

How complicated is a compiler?

```
double dotproduct(int n, double * a, double * b) {
  double dp = 0.0;
  int i;
  for (i = 0; i < n; i++)
    dp += a[i] * b[i];
  return dp;
}
```

$$a \cdot b = \sum_{i=0}^{n-1} a_i b_i$$

Compiled with Tru64/Unix and manually translated back to C...

...by Xavier Leroy

```
double dotproduct(int n, double a[], double b[]) {
  dp = 0.0;
  if (n ≤ 0) goto L5;
  r2 = n − 3; f1 = 0.0; r1 = 0; f10 = 0.0; f11 = 0.0;
  if (r2 > n || r2 ≤ 0) goto L19;
  prefetch(a[16]); prefetch(b[16]);
  if (4 ≥ r2) goto L14;
  prefetch(a[20]); prefetch(b[20]);
  f12 = a[0]; f13 = b[0]; f14 = a[1]; f15 = b[1];
  r1 = 8; if (8 ≥ r2) goto L16;
L17: f16 = b[2]; f18 = a[2]; f17 = f12 * f13;
  f19 = b[3]; f20 = a[3]; f15 = f14 * f15;
  f12 = a[4]; f16 = f18 * f16;
  f19 = f29 * f19; f13 = b[4]; a += 4; f14 = a[1];
  f11 += f17; r1 += 4; f10 += f15;
  f15 = b[5]; prefetch(a[20]); prefetch(b[24]);
  f1 += f16; dp += f19; b += 4;
  if (r1 < r2) goto L17;
L16: f15 = f14 * f15; f21 = b[2]; f23 = a[2]; f22 = f12 * f13;
  f24 = b[3]; f25 = a[3]; f21 = f23 * f21;
  f12 = a[4]; f13 = b[4]; f24 = f25 * f24; f10 = f10 + f15;
  a += 4; b += 4; f14 = a[8]; f15 = b[8];
  f11 += f22; f1 += f21; dp += f24;
L18: f26 = b[2]; f27 = a[2]; f14 = f14 * f15;
  f28 = b[3]; f29 = a[3]; f12 = f12 * f13; f26 = f27 * f26;
  a += 4; f28 = f29 * f28; b += 4;
  f10 += f14; f11 += f12; f1 += f26;
  dp += f28; dp += f1; dp += f10; dp+=f11;
  if (r1 ≥ n) goto L5;
L19: f30 = a[0]; f18 = b[0]; r1 += 1; a += 8; f18 = f30 * f18;
  b += 8; dp += f18;
  if (r1 < n) goto L19;
L5: return dp;
L14: f12 = a[0]; f13 = b[0]; f14 = a[1]; f15 = b[1]; goto L18;
```

```
L17: f16 = b[2]; f18 = a[2]; f17 = f12 * f13;
  f19 = b[3]; f20 = a[3]; f15 = f14 * f15;
  f12 = a[4]; f16 = f18 * f16;
  f19 = f29 * f19; f13 = b[4]; a += 4; f14 = a[1];
  f11 += f17; r1 += 4; f10 += f15;
  f15 = b[5]; prefetch(a[20]); prefetch(b[24]);
  f1 += f16; dp += f19; b += 4;
  if (r1 < r2) goto L17;
```

```
L17: f16 = b[2]; f18 = a[2]; f17 = f12 * f13;
  f19 = b[3]; f20 = a[3]; f15 = f14 * f15;
  f12 = a[4]; f16 = f18 * f16;
  f19 = f29 * f19; f13 = b[4]; a += 4; f14 = a[1];
  f11 += f17; r1 += 4; f10 += f15;
  f15 = b[5]; prefetch(a[20]); prefetch(b[24]);
  f1 += f16; dp += f19; b += 4;
  if (r1 < r2) goto L17;
```

So, bugs in compilers?

So, bugs in compilers?

Yep

. . .

- Regehr et al (2008 and 2010):

- Regehr et al (2008 and 2010):
  - 13 production-quality C compilers tested

- Regehr et al (2008 and 2010):
  - 13 production-quality C compilers tested
  - 13 production-quality C compilers found buggy

- Regehr et al (2008 and 2010):
  - 13 production-quality C compilers tested
  - 13 production-quality C compilers found buggy
  - Around 300 previously unknown bugs

- Regehr et al (2008 and 2010):
  - 13 production-quality C compilers tested
  - 13 production-quality C compilers found buggy
  - Around 300 previously unknown bugs
- Just have a look at bug trackers...

But what exactly is a compiler bug?

But what exactly is a compiler bug?

- Crash of the compiler, wrong errors, reject correct files

But what exactly is a compiler bug?

- Crash of the compiler, wrong errors, reject correct files
- Change the meaning

But what exactly is a compiler bug?

- Crash of the compiler, wrong errors, reject correct files
- Change the meaning
  - takes a function that returns false

But what exactly is a compiler bug?

- Crash of the compiler, wrong errors, reject correct files
- Change the meaning
  - takes a function that returns false
  - produces a function that returns 42

But what exactly is a compiler bug?

- Crash of the compiler, wrong errors, reject correct files
- Change the meaning
  - takes a function that returns false
  - produces a function that returns 42

Solutions?

Solutions?

- Testing?

Solutions?

- Testing?

Solutions?

- Testing?
- Very simple compiler, manually checked executable

Solutions?

- Testing?
- Very simple compiler, manually checked executable

DO-178

Solutions?

- Testing?
- Very simple compiler, manually checked executable
- Proving it correct!

DO-178

Solutions?

- Testing?
- Very simple compiler, manually checked executable
- Proving it correct!
  - Clear and precise specification

DO-178

Solutions?

- Testing?
- Very simple compiler, manually checked executable
- Proving it correct!
  - Clear and precise specification
  - Yes, feasible! CompCert (4 person.years, Leroy & al.)

DO-178

How to prove programs?

- Method B
- Program in C, prove with Frama-C
- Program in Coq, prove in Coq
- ...

How to prove programs?

- Method B
- Program in C, prove with Frama-C
- Program in Coq, prove in Coq
- ...

Emacs@mp-41033.rocq.inria.fr

-u:-- *goals* All L1 (Co

-u:-- exemple.v Bot L6 (co -u:-- *response* All L1 (Co

Emacs@mp-41033.rocq.inria.fr

```
Definition square (x:Z): Z :=
  x * x.
```

-u:** exemple.v Bot L7 (co

-u:-- *goals* All L1 (Co

```
square is defined
```

-u:-- *response* All L1 (Co

Emacs@mp-41033.rocq.inria.fr

```
Definition square (x:Z): Z :=
  x * x.

Lemma square_positive:
  forall (x:Z), square x >= 0.
Proof.
```

-u:** exemple.v Bot L11 (co

```
1 subgoal

  ============================
   forall x : Z, square x >= 0
```

-u:-- *goals* All L4 (Co

-u:-- *response* All L1 (Co

Emacs@mp-41033.rocq.inria.fr

```
Definition square (x:Z): Z :=
  x * x.

Lemma square_positive:
  forall (x:Z), square x >= 0.
Proof.
  unfold square.
```

-u:** exemple.v Bot L12 (co

```
1 subgoal

  ============================
   forall x : Z, x * x >= 0
```

-u:-- *goals* All L4 (Co

-u:-- *response* All L1 (Co

Emacs@mp-41033.rocq.inria.fr

```
Definition square (x:Z): Z :=
  x * x.

Lemma square_positive:
  forall (x:Z), square x >= 0.
Proof.
  unfold square.
  intro x.
  destruct x.
```

-u:-- exemple.v Bot L14 (co

```
3 subgoals

  ============================
   0 * 0 >= 0

subgoal 2 is:
 Zpos p * Zpos p >= 0
subgoal 3 is:
 Zneg p * Zneg p >= 0
```

-u:-- *goals* All L4 (Co

-u:-- *response* All L1 (Co

Emacs@mp-41033.rocq.inria.fr

```
Definition square (x:Z): Z :=
  x * x.

Lemma square_positive:
  forall (x:Z), square x >= 0.
Proof.
  unfold square.
  intro x.
  destruct x.
  compute; congruence.
  compute; congruence.
```

-u:** exemple.v Bot L17 (co

-u:-- *goals* All L1 (Co

```
Proof completed.
```

-u:-- *response* All L1 (Co

Emacs@mp-41033.rocq.inria.fr

```
Definition square (x:Z): Z :=
  x * x.

Lemma square_positive:
  forall (x:Z), square x >= 0.
Proof.
  unfold square.
  intro x.
  destruct x.
  compute; congruence.
  compute; congruence.
Qed.
```

-u:** exemple.v Bot L18 (co

-u:-- *goals* All L1 (Co

```
square_positive is defined
```

-u:-- *response* All L1 (Co

Correct Compiler?

Correct Compiler?

`prog_c`

Correct Compiler?

```
compile prog_c = OK
```

Correct Compiler?

```
compile prog_c = OK executable
```

Correct Compiler?

```
  compile prog_c = OK executable
∧ Has_Behavior beh executable
```

Correct Compiler?

- Terminate or not

```
  compile prog_c = OK executable
∧ Has_Behavior beh executable
```

Correct Compiler?

- Terminate or not
- Returned value
- Input/Output

```
  compile prog_c = OK executable
∧ Has_Behavior beh executable
```

Correct Compiler?

- Terminate or not
- Returned value
- Input/Output
- System calls
- Volatile memory access

```
  compile prog_c = OK executable
∧ Has_Behavior beh executable
```

Correct Compiler?

```
  compile prog_c = OK executable
∧ Has_Behavior beh executable
→ Has_Behavior beh prog_c.
```

Correct Compiler?

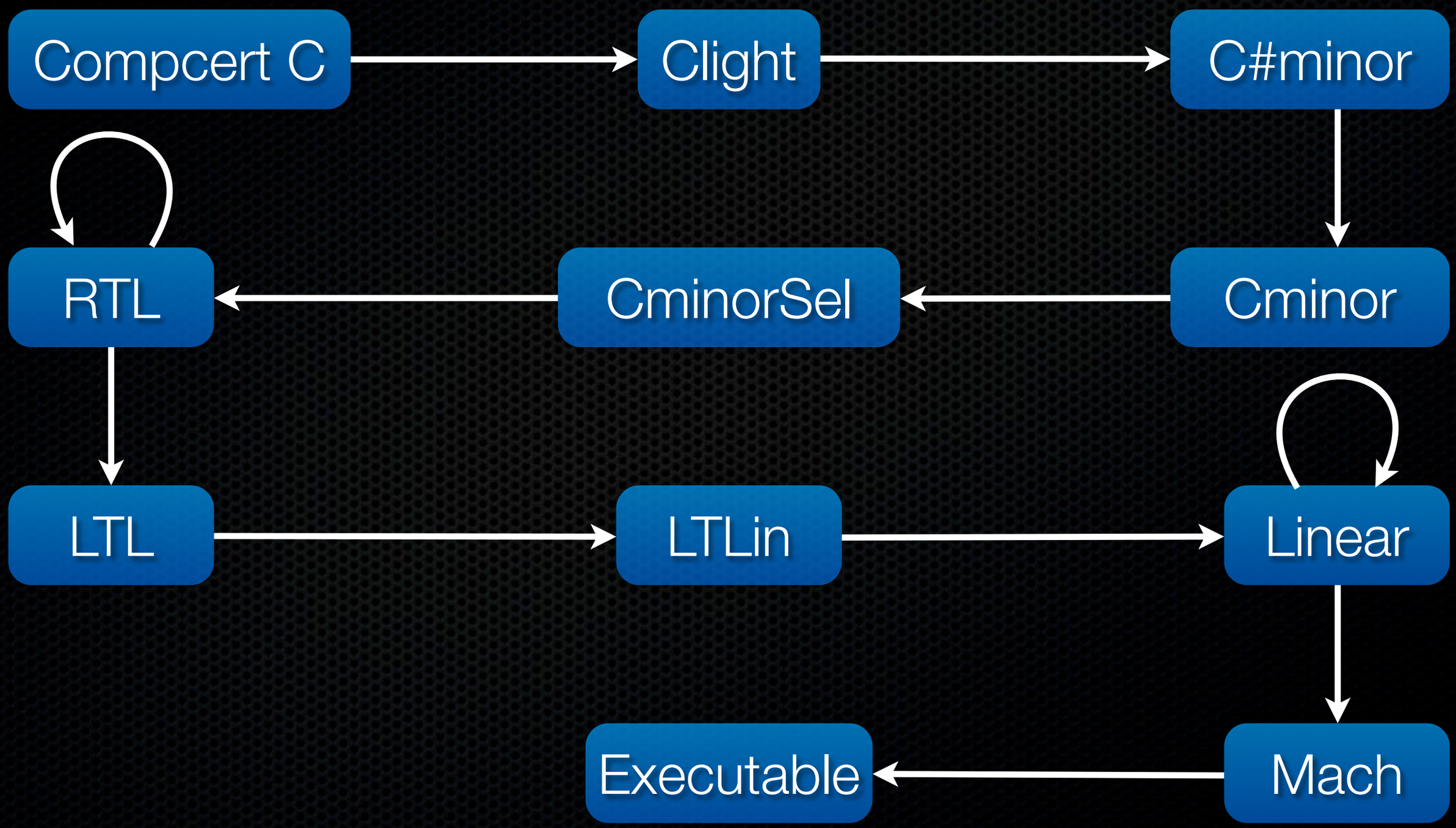
```
Theorem compiler_correct:
  ∀ prog_c executable beh,
    compile prog_c = OK executable
  ∧ Has_Behavior beh executable
  → Has_Behavior beh prog_c.
```

Everything proved at once?

Compcert C

Executable

Everything proved at once?



Proof of optimizations

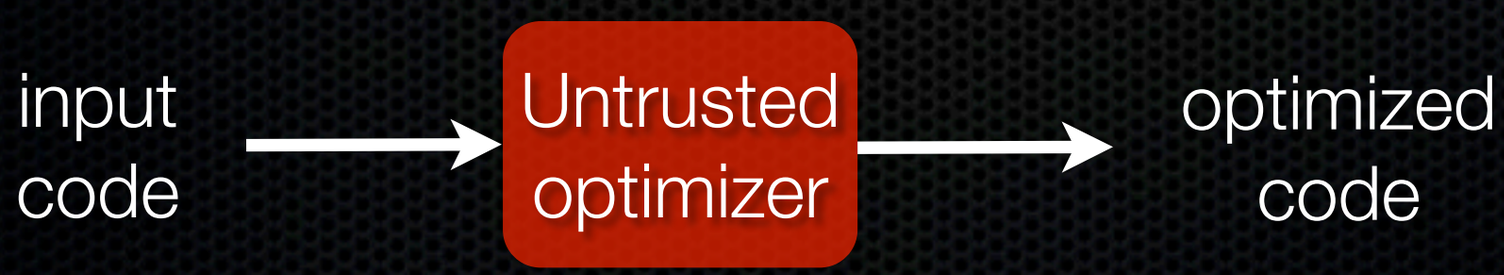
Proof of optimizations

- Implement it in Coq and prove it

Proof of optimizations

- Implement it in Coq and prove it

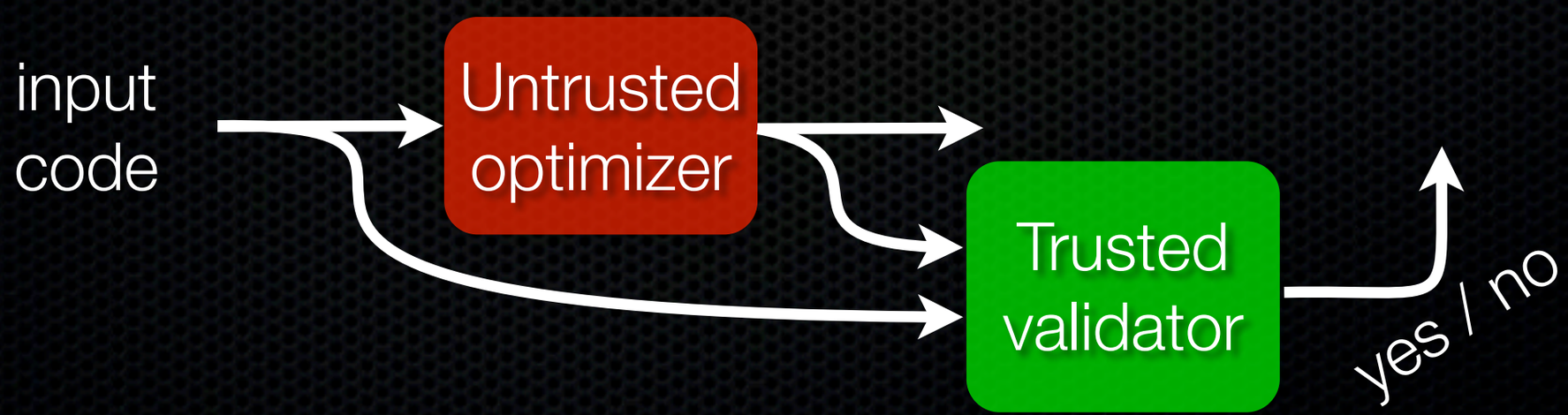
- «Off the shelf» optimizer, proved validator



Proof of optimizations

- Implement it in Coq and prove it

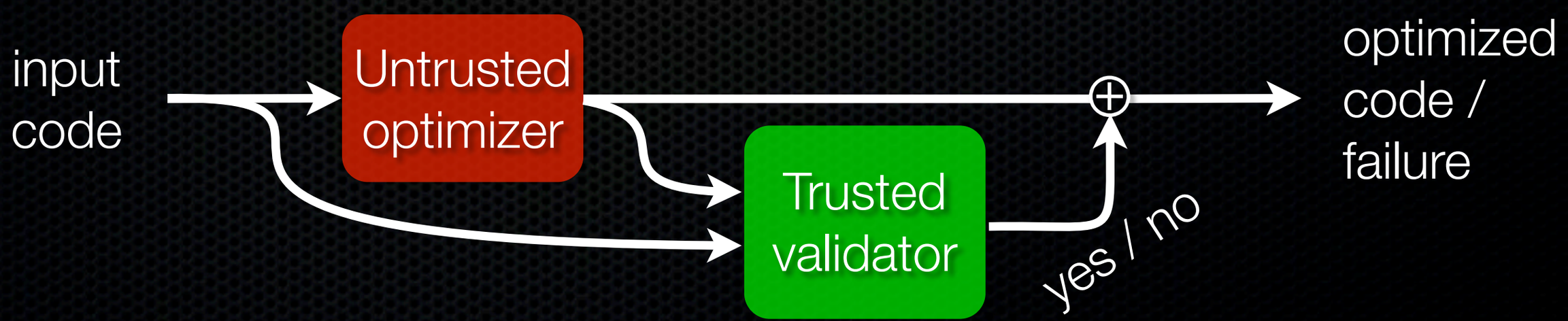
- «Off the shelf» optimizer, proved validator



Proof of optimizations

- Implement it in Coq and prove it

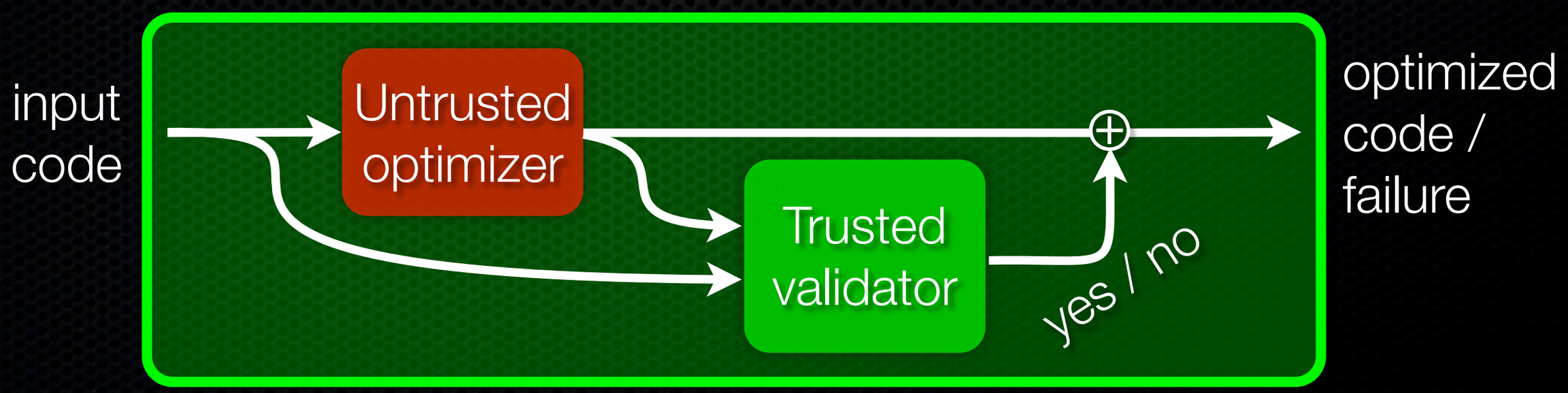
- «Off the shelf» optimizer, proved validator



Proof of optimizations

- Implement it in Coq and prove it

- «Off the shelf» optimizer, proved validator



Validated optimizations

- Lazy code motion
- Software pipelining
- ...
- Polyhedral optimizations

Validated optimizations

- Lazy code motion
- Software pipelining
- ...
- Polyhedral optimizations

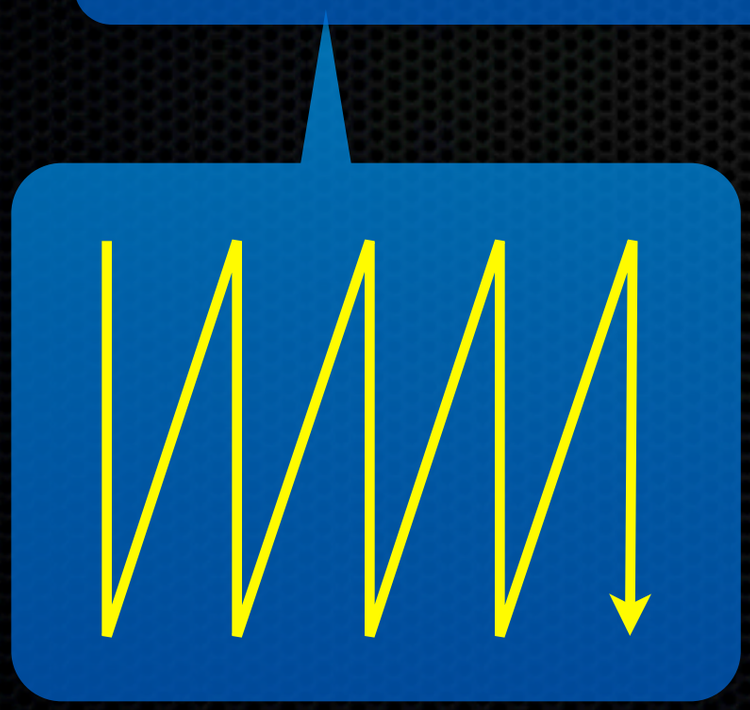
Optimizing loop nests

Polyhedral optimizations

```
for c = 0 to nb_c
  for l = 0 to nb_l
    M[l][c] = 0;
  done;
done;
```

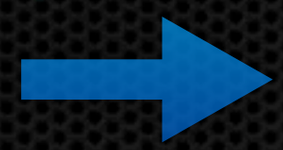
Polyhedral optimizations

```
for c = 0 to nb_c
  for l = 0 to nb_l
    M[l][c] = 0;
  done;
done;
```

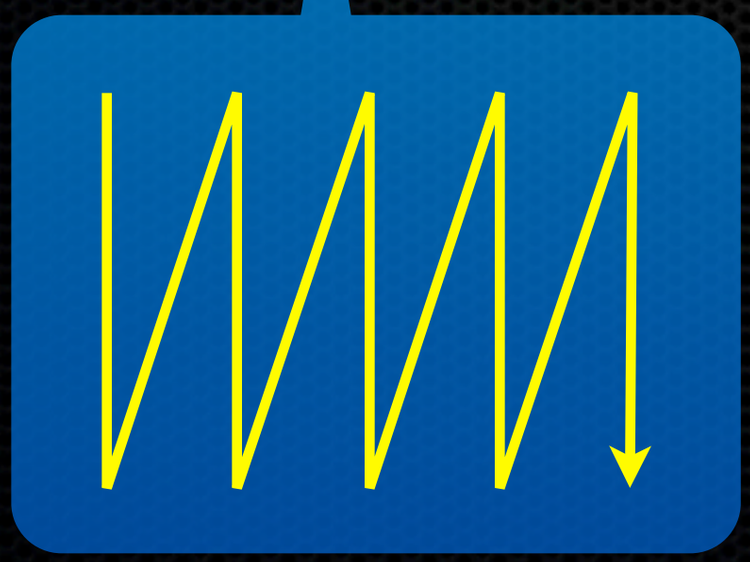


Polyhedral optimizations

```
for c = 0 to nb_c
  for l = 0 to nb_l
    M[l][c] = 0;
  done;
done;
```

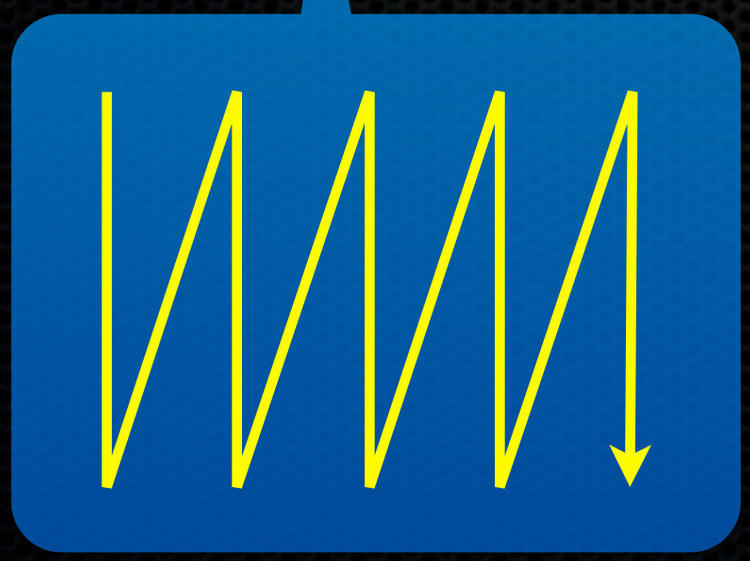


```
for l = 0 to nb_l
  for c = 0 to nb_c
    M[l][c] = 0;
  done;
done;
```

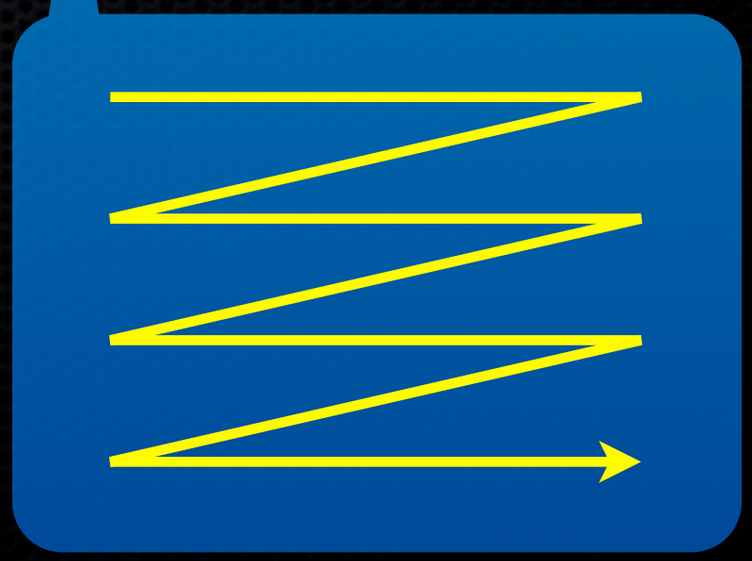


Polyhedral optimizations

```
for c = 0 to nb_c
  for l = 0 to nb_l
    M[l][c] = 0;
  done;
done;
```

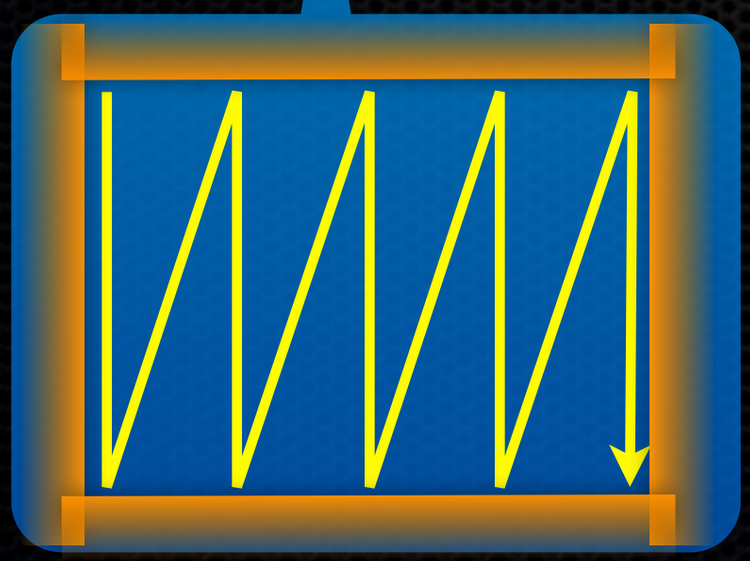


```
for l = 0 to nb_l
  for c = 0 to nb_c
    M[l][c] = 0;
  done;
done;
```

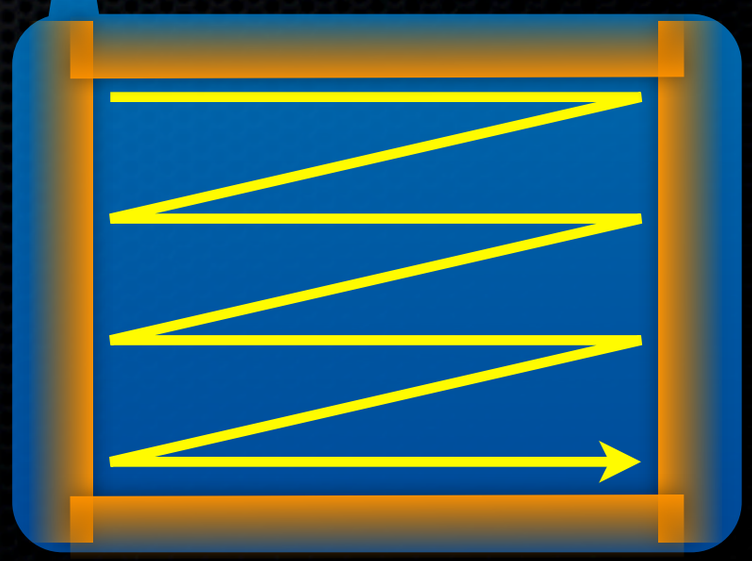


Polyhedral optimizations

```
for c = 0 to nb_c
  for l = 0 to nb_l
    M[l][c] = 0;
  done;
done;
```



```
for l = 0 to nb_l
  for c = 0 to nb_c
    M[l][c] = 0;
  done;
done;
```



Take home message

- Bugs are dangerous
- You can (and even should) prove your code
- You then need a proved compiler
- Such a compiler exists
- and is getting better!