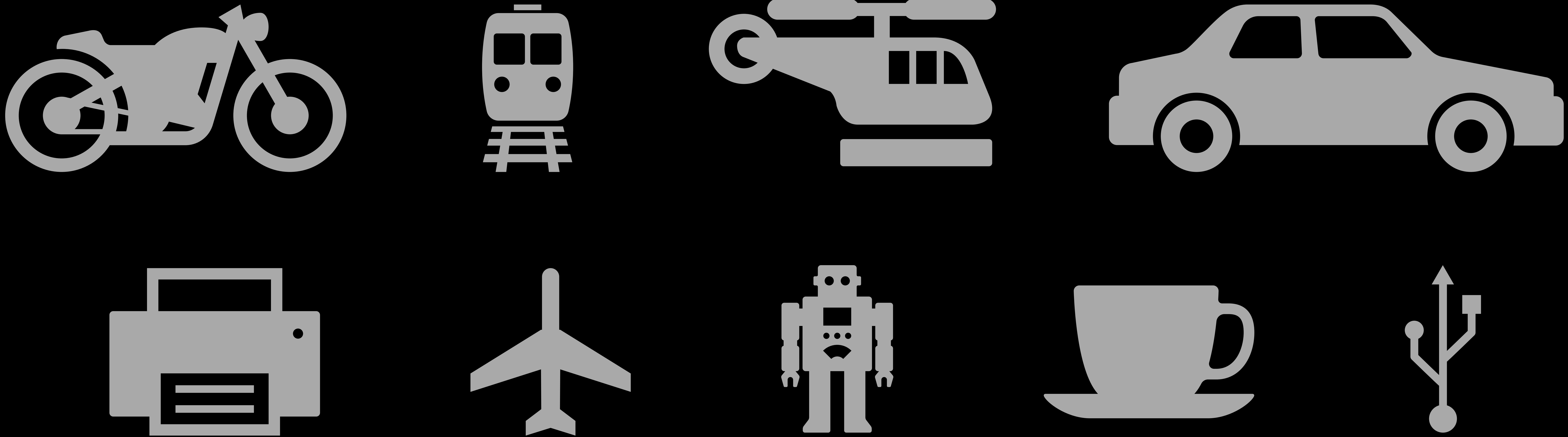
Pipit on the post

Proving pre- and post- conditions of reactive systems

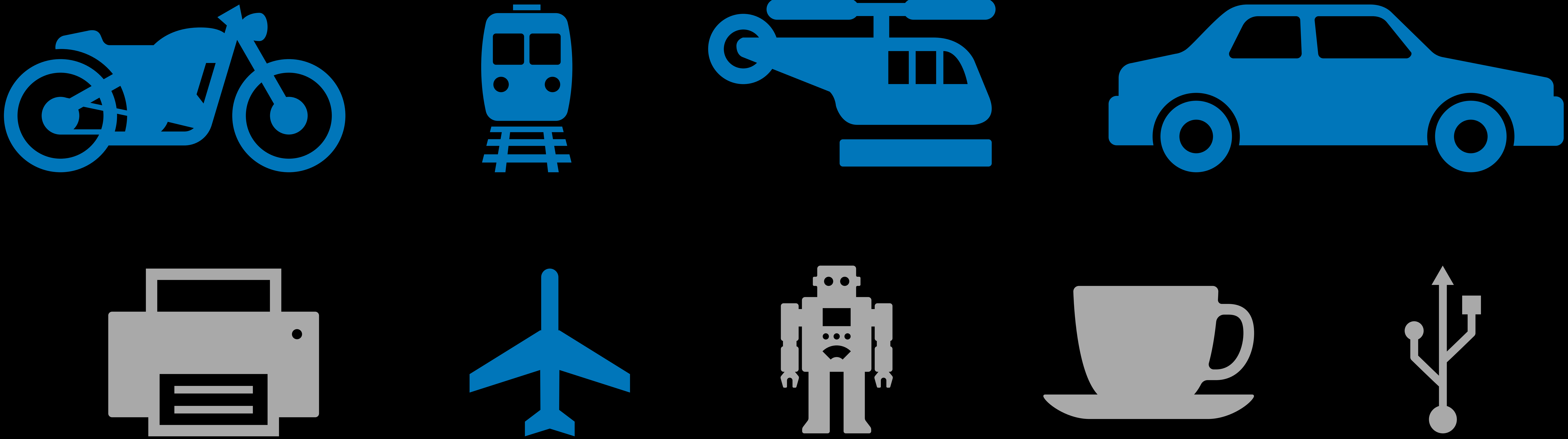
Amos Robinson, ~~Australian National University~~ -> AMD
Alex Potanin, Australian National University

photo: Australian pipit, Hexham swamp, Australia

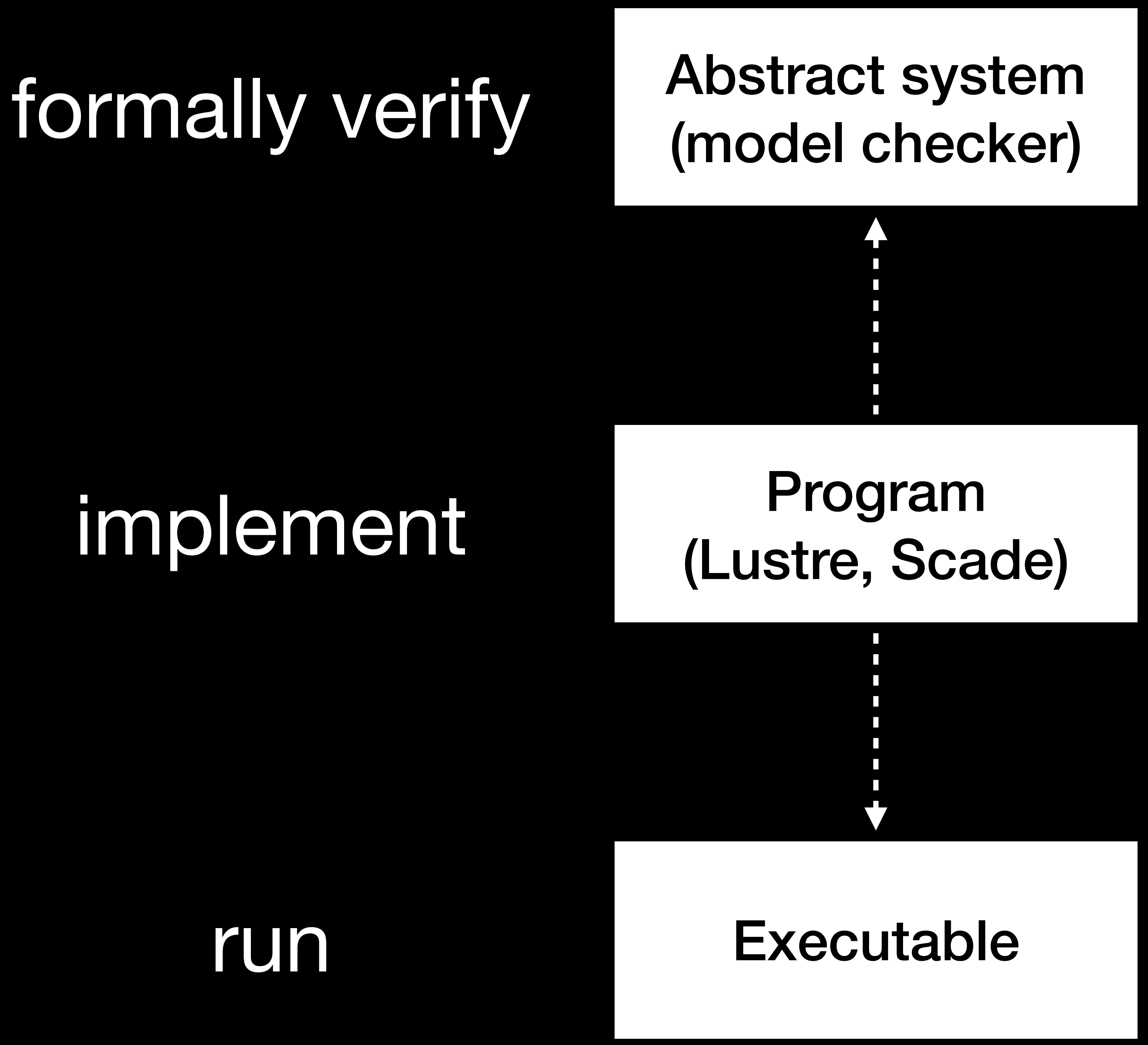
Reactive systems



Safety-critical reactive systems



Trustworthy safety-critical systems



formally verify

Abstract system
(model checker)

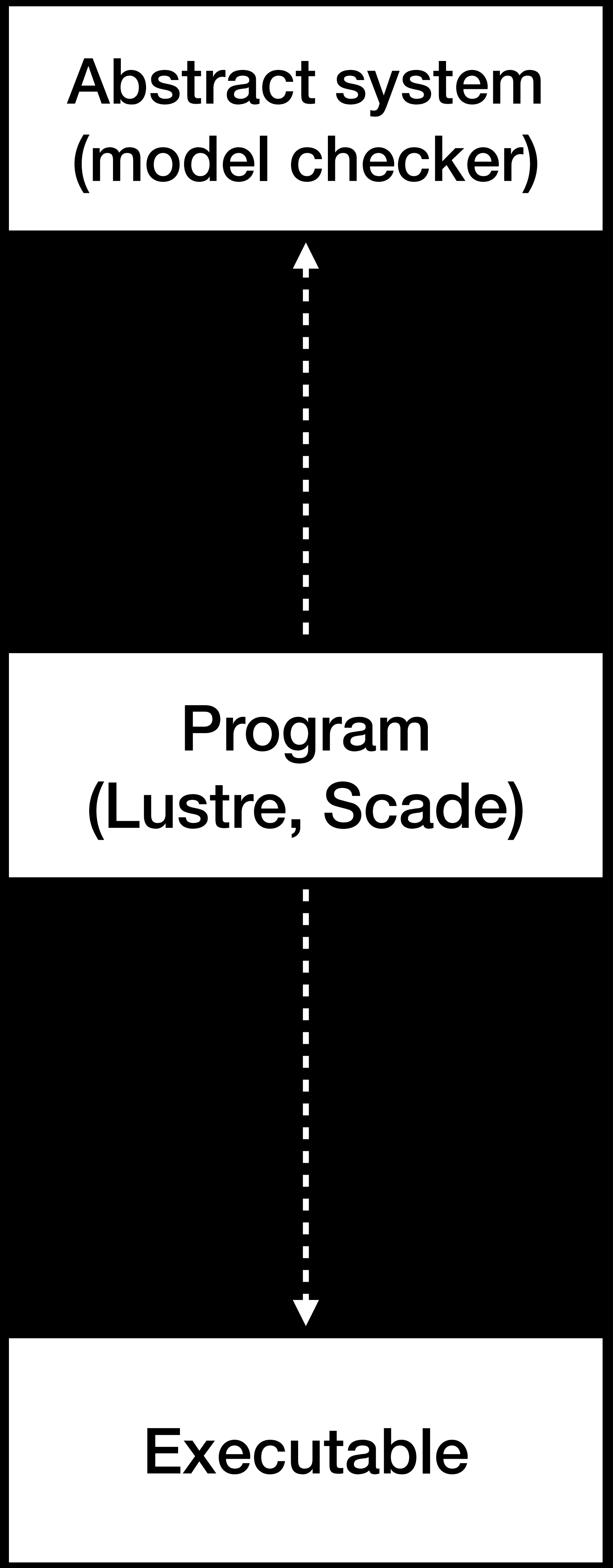
implement

Program
(Lustre, Scade)

run

Executable

Trustworthy safety-critical systems



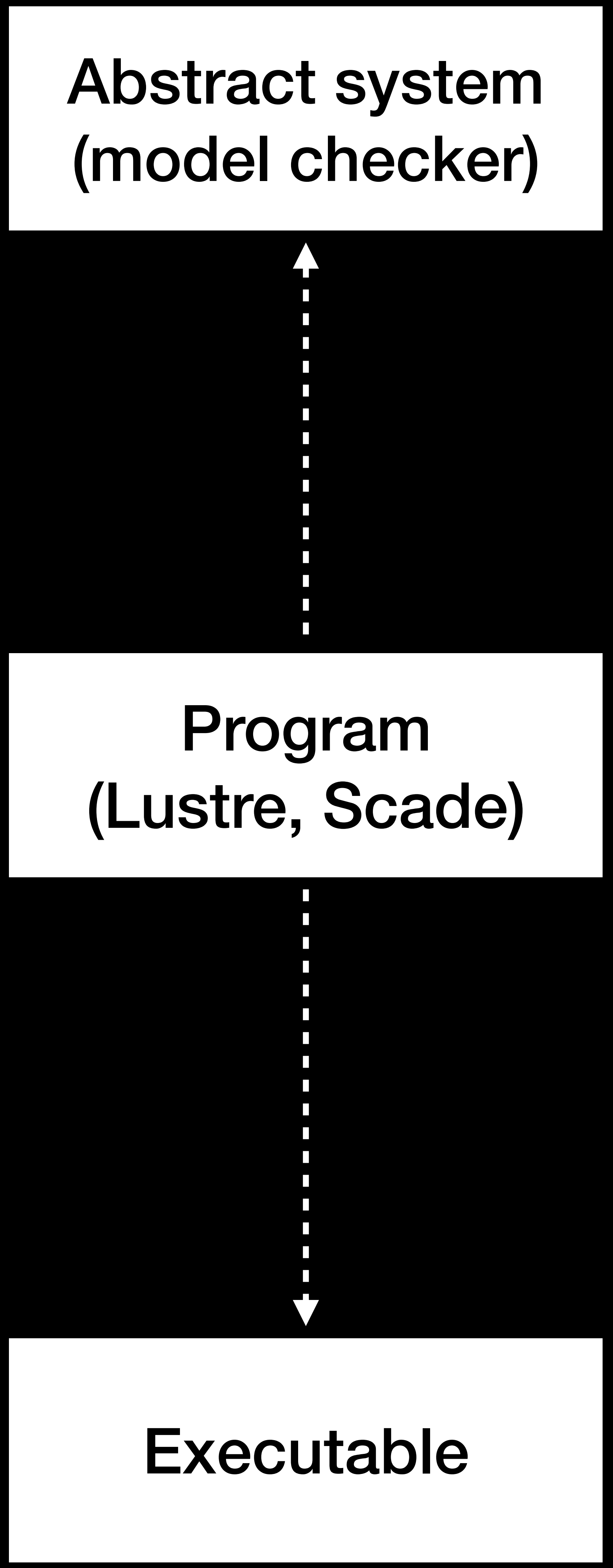
Abstract system (model checker)

Program (Lustre, Scade)

Executable

```
node div_or_default(num, den, default: int)
           returns (res: int)
var div: int;
let
    div = num / den;
    res = if den = 0 then default else div;

    --%PROPERTY den = 0 => res = default;
tel
```

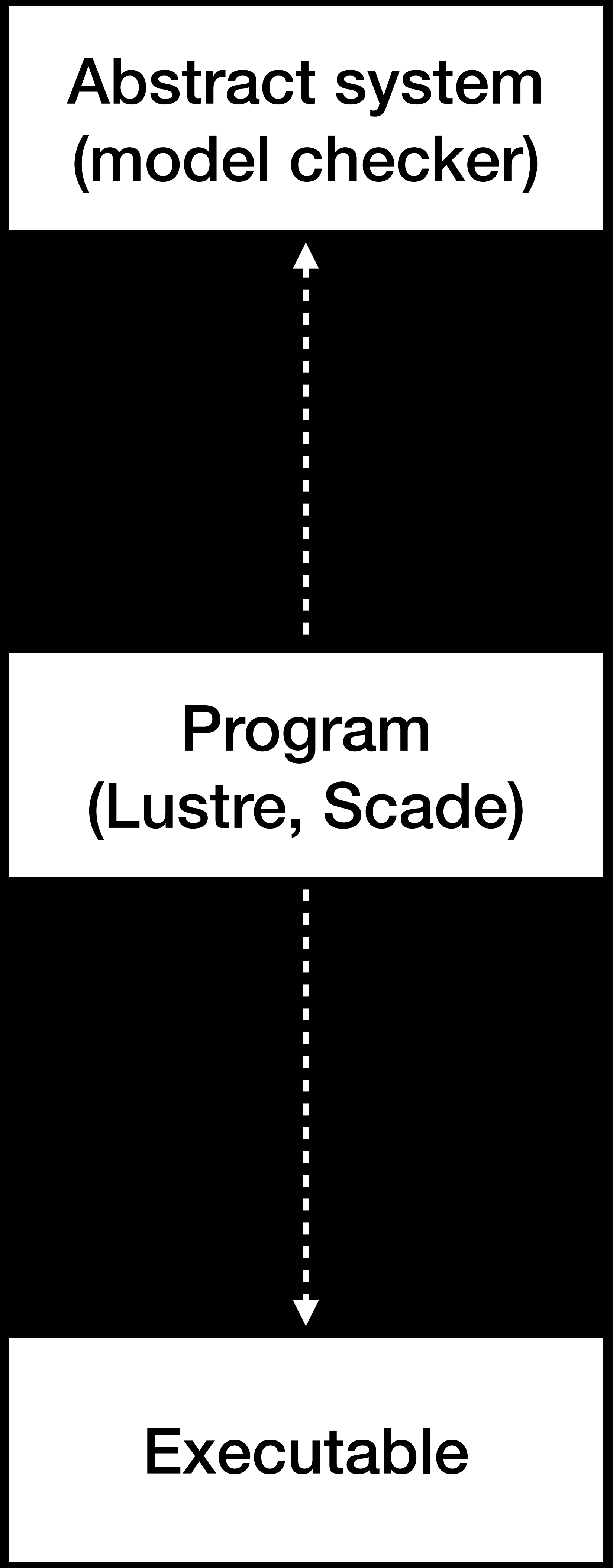
Trustworthy safety-critical systems



property proved

```
node div_or_default(num, den, default: int)
             returns (res: int)
var div: int;
let
    div = num / den;
    res = if den = 0 then default else div;

    --%PROPERTY den = 0 => res = default;
tel
```

Trustworthy safety-critical systems



Abstract system (model checker)

Program (Lustre, Scade)

Executable

property proved

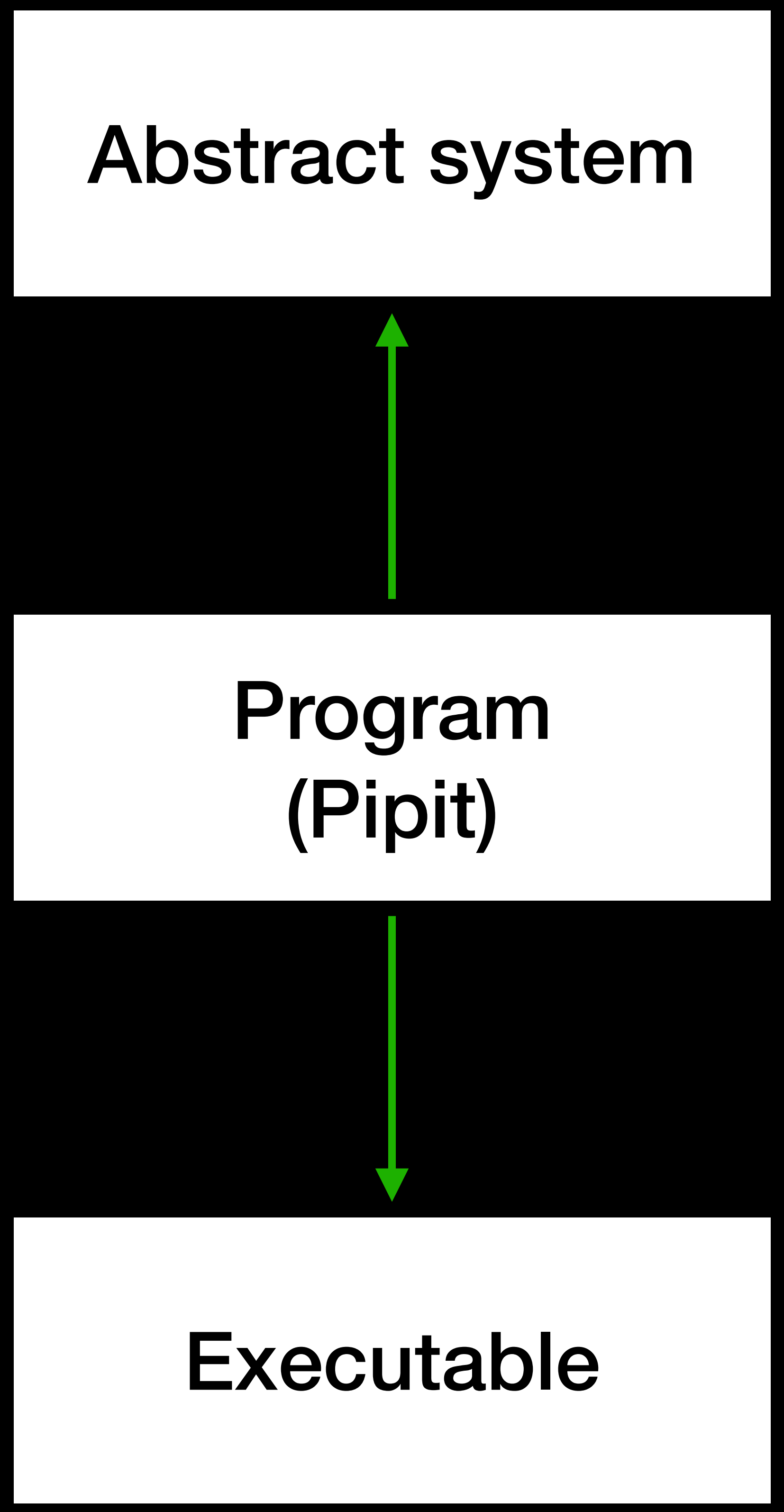
```
node div_or_default(num, den, default: int)
              returns (res: int)
var div: int;
let
    div = num / den;
    res = if den = 0 then default else div;

    --%PROPERTY den = 0 => res = default;
tel
```

error: division by zero

Trustworthy toolchain: the goals

correct compilation

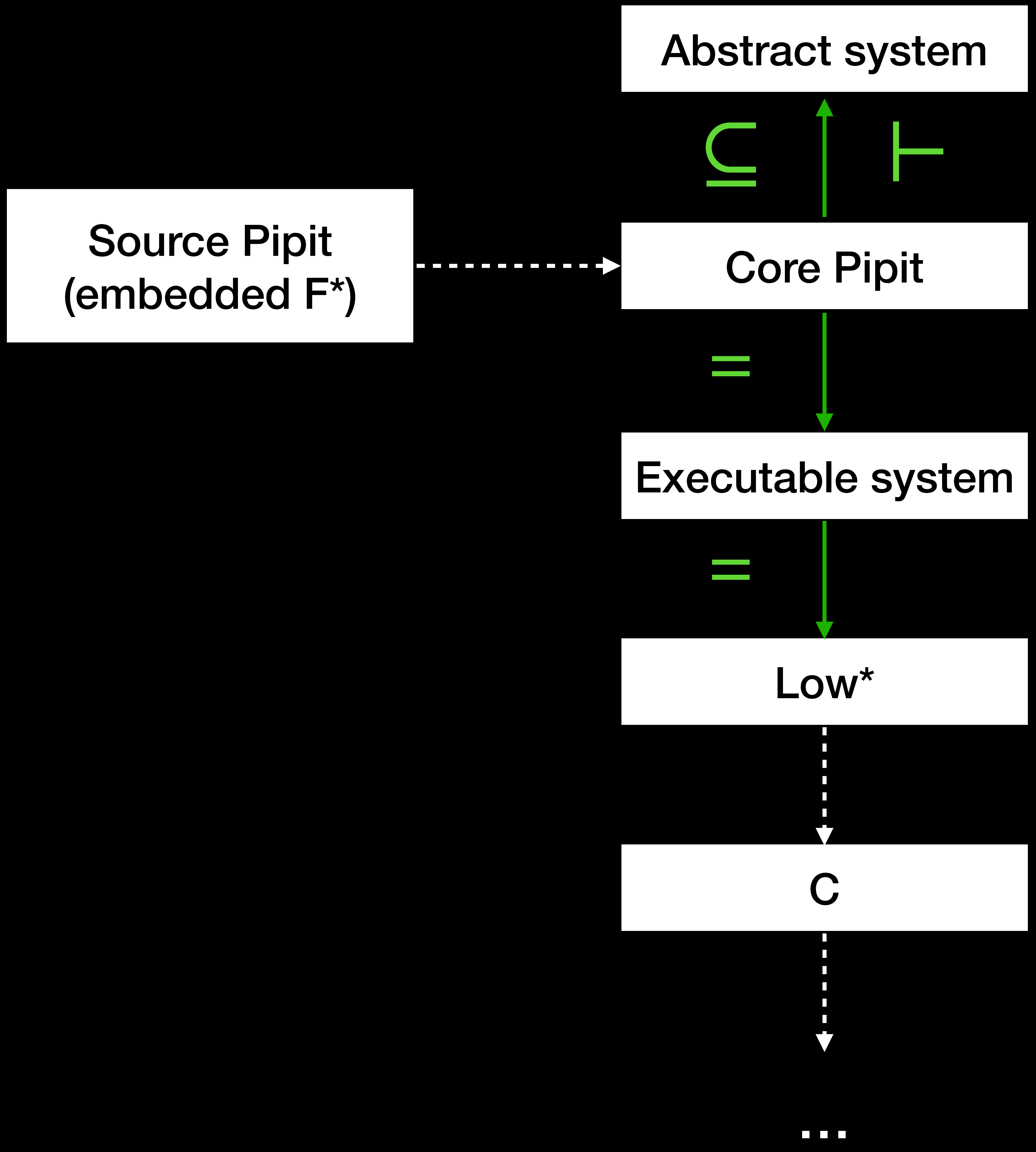
executable ⊆ abstract

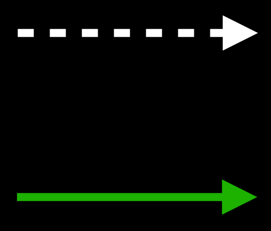


sound proofs

abstract ✓ ⊢ executable ✓

Trustworthy toolchain: reality




unverified (trusted)

formally verified

Counting with Pipit

```
let count_when (max: int) (inc: stream bool): stream int =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in
  count
```

Contracts

```
let count_when (max: int) (inc: stream bool): stream int =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in
  count

       ASSUME                      GUARANTEE
       {    } count_when max inc { c. c <= max }
```

Contracts

```
let count_when (max: int) (inc: stream bool): stream int =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in
  count

       ASSUME                           GUARANTEE
       {    } count_when max inc { c. c <= max /\
                                      c >= 0 `fby` c /\
                           not inc -> c  = 0 `fby` c /\
                               inc -> ... }
```

Inline assertions

```
let count_when (max: int) (inc: stream bool): stream int =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in
  check (count <= max);
  count
```

Pipit

- a synchronous language with
- formally verified translation and codegen and
- sound metatheory for contracts and assertions

photo: Australian pipit, Hunter Wetlands National Park, Australia