Local temporal reasoning

Eric Koskinen
Visiting Assistant Professor
New York University

6 May 2014

Joint work with Tachio Terauchi

Making sure programs operate correctly

Making sure programs operate correctly

Safety Don’t crash! (Nothing bad happens.)

SLAM, Blast, Astrée, etc.

so many other
important properties
missing!

Making sure programs operate correctly

Safety Don’t crash! … ens.)

Prove that the program finishes.

M, Blast, Astrée, etc.

Liveness Something good eventually happens.

Terminator, ARMC, etc.

“Every time a client connects to the web server, eventually web server gives a response.”

Temporal Logic

$G(\neg \texttt{crash})$

Safety Don't crash! (Nothing bad happens.)

$F(\texttt{finish})$

Liveness Something good eventually happens.

Terminator, ARMC, etc.

"Every time a client connects to the web server, eventually web server gives a response."

$G(\texttt{request} \Rightarrow F(\texttt{response}))$

Temporal Logic

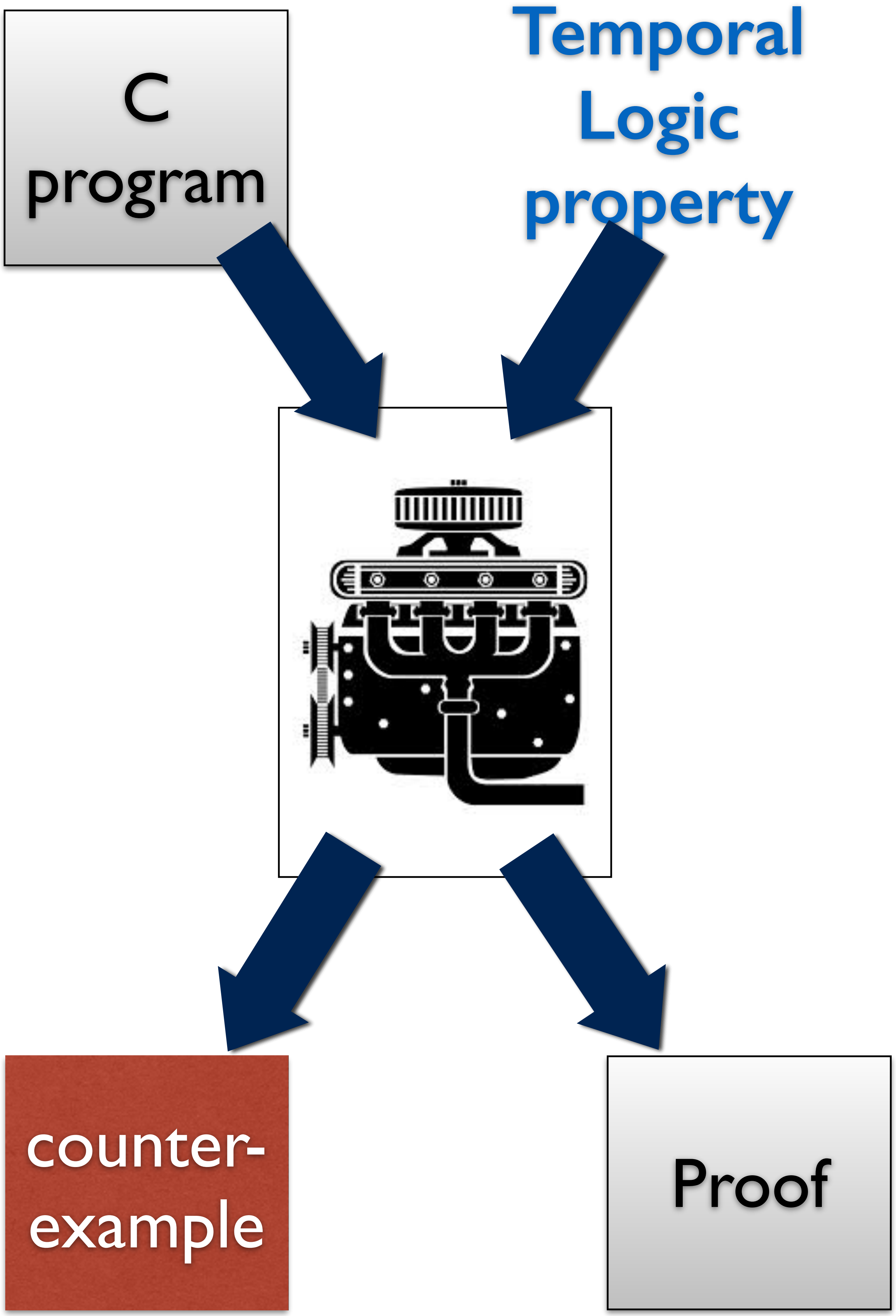
Safety Don’t crash! (Nothing bad happens.)

Liveness Something good eventually happens.

Terminator, ARMC, etc.

“Every time a client connects to the web server, eventually web server gives a response.”

Rich properties, but TL tools only for hardware verification.



C program

Temporal Logic property

counter-example

Proof

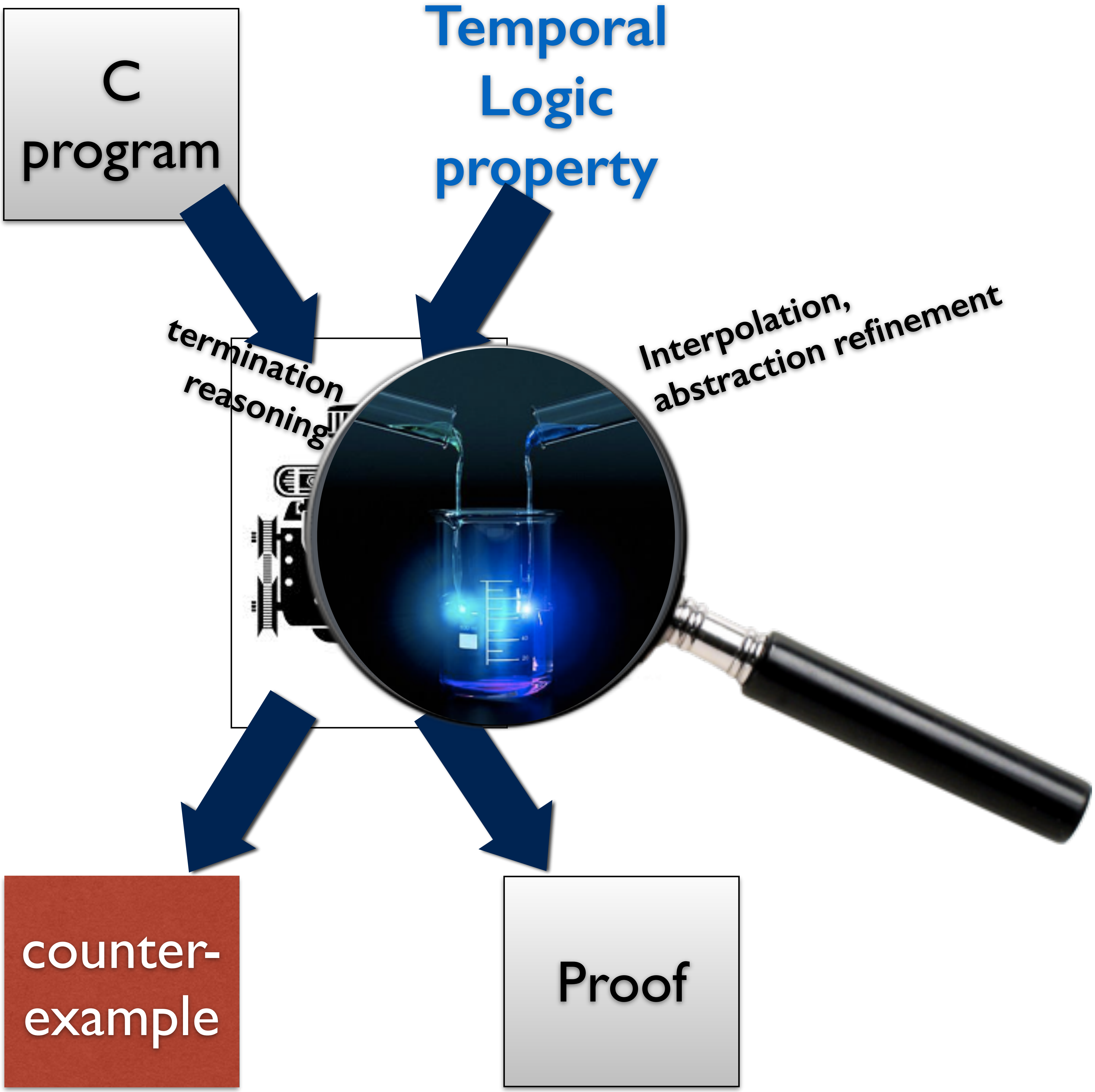
PostgreSQL StreamServer

`int StreamServerPort int family`

Proving this statically is challenging.
Reason about termination.
Reason about reachability.

```
 * StreamServerPort -- open a "listening"
 *
 * Successfully opened sockets are added t
 * at the first position that isn't -1.
 *
 * RETURNS: STATUS_OK or STATUS_ERROR
 */

int
StreamServerPort(int family, char *hostName, unsigned short portNumber,
                                 char *unixSocketName,
                                 int ListenSocket[], int MaxListen)
void body()
{
        /* Initialize hint structure */

#ifdef HAVE_UNIX_SOCKETS
        if (family == AF_UNIX)
        {
                /* Lock_AF_UNIX will also fill in sock_path. */
                /* if (Lock_AF_UNIX(portNumber, unixSocketName) != STATUS_OK) */
                /*           return STATUS_ERROR; */
                service = sock_path;
        }
        else
#endif   /* HAVE_UNIX_SOCKETS */
        {
                snprintf(1, sizeof(1), "%d", portNumber);
                service = 1;
        }

        ret = getaddrinfo_all(hostName, service, &hint, &addrs);
        if (ret || !addrs)
        {
        if (hostName) {
                        /* ereport(LOG, */

        } els
```

$$(G\ \neg \mathtt{error}) \Rightarrow F\ (\mathtt{added{>}0} \land F\ \mathtt{ret{=}OK})$$



C program

Temporal Logic property

termination reasoning

Interpolation, abstraction refinement

counter-example

Proof

C
program

Temporal
Logic
property

termination
reasoning

Interpolation,
abstraction refinement

Making Prophecies with Decision Predicates

Eric Koskinen
University of Cambridge
ejk39@cam.ac.uk

Temporal property verification as a program analysis task

Byron Cook[1], Eric Koskinen[2], and Moshe Vardi[3]

Reasoning about Nondeterminism in Programs

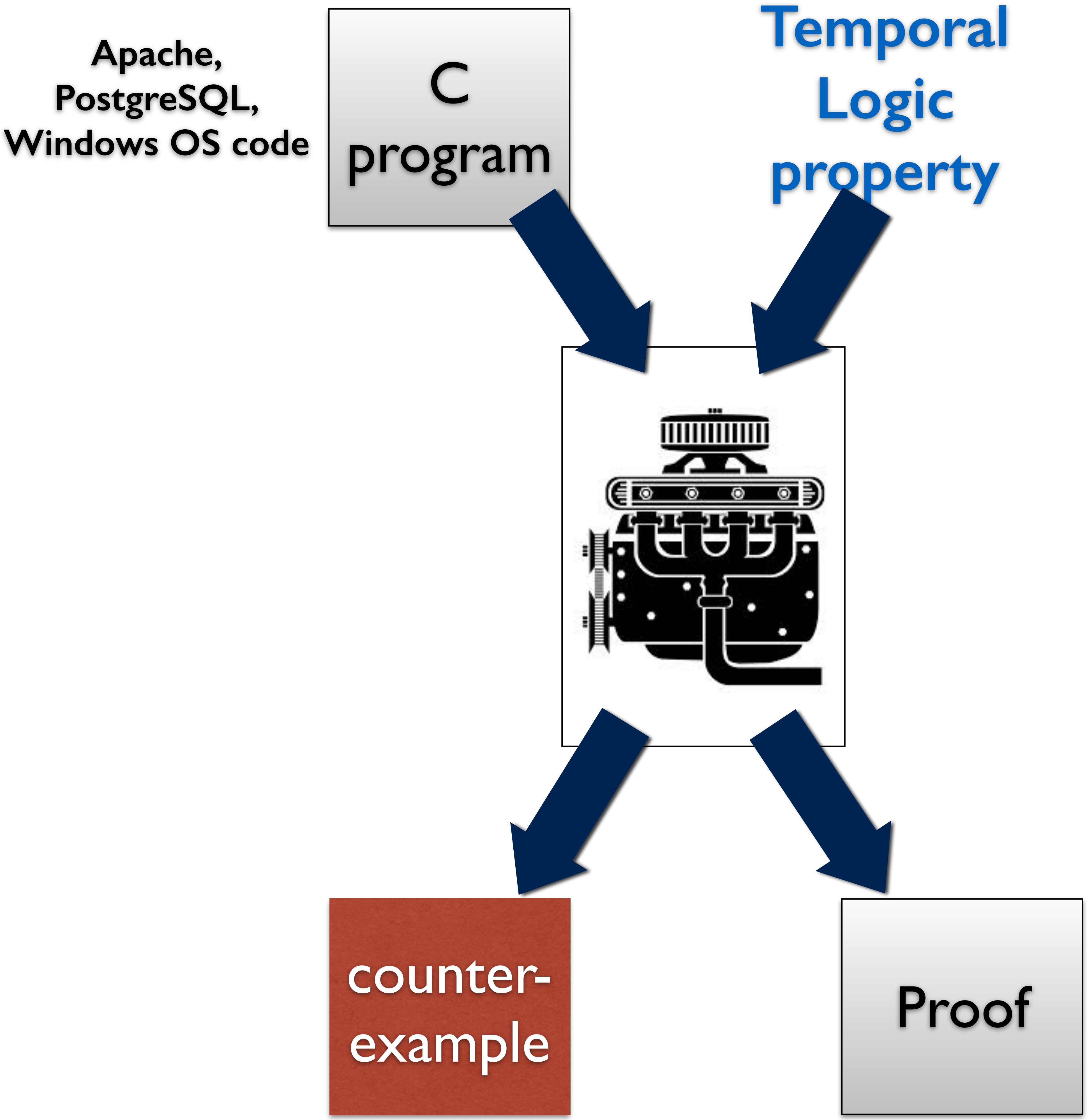
Byron Cook
Microsoft Research Cambridge
& University College London

Eric Koskinen *
New York University

PLDI'13

CAV'11
Award

POPL'11



Apache,
PostgreSQL,
Windows OS code

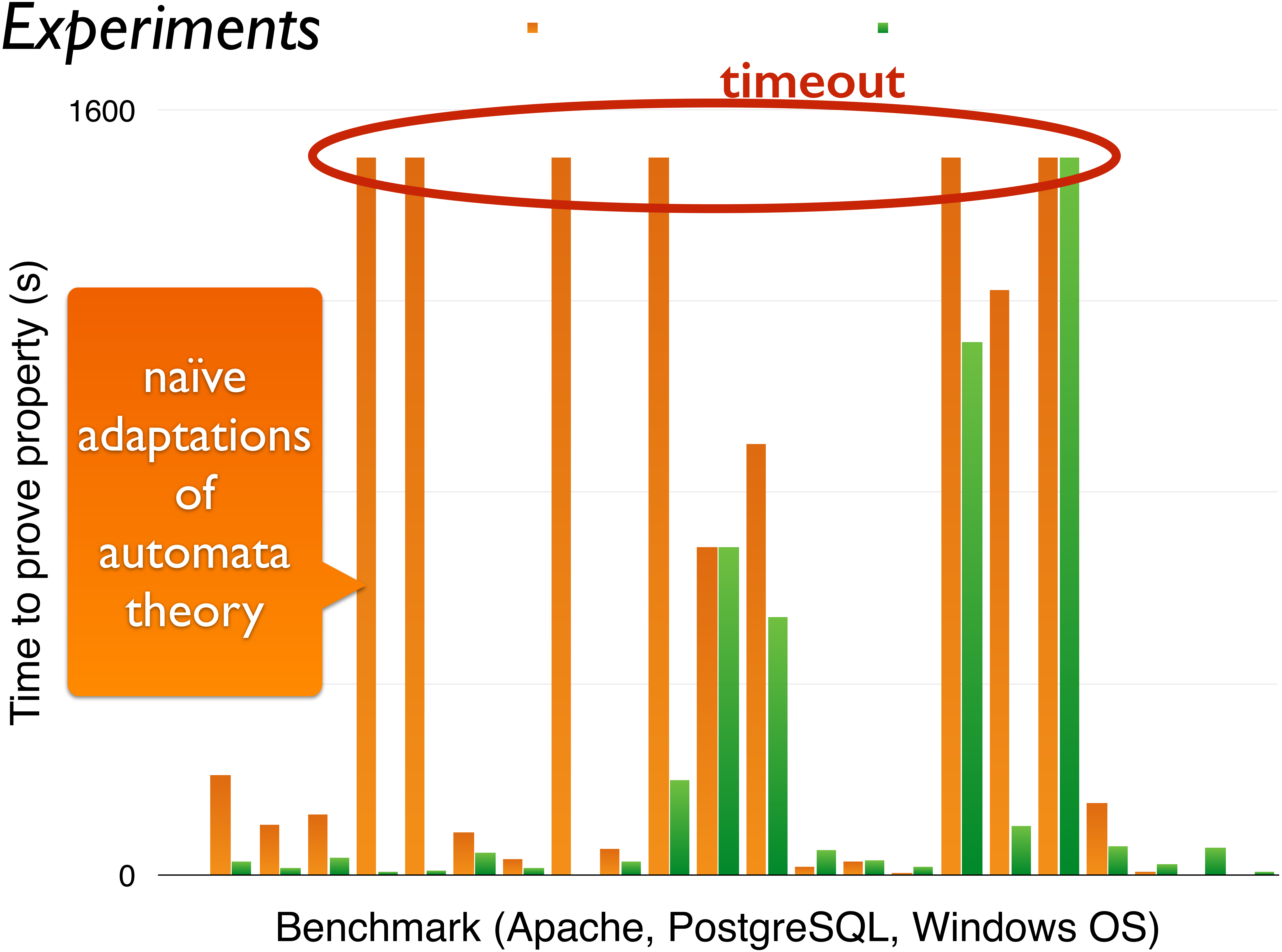
C program

Temporal Logic property

counter-example

Proof

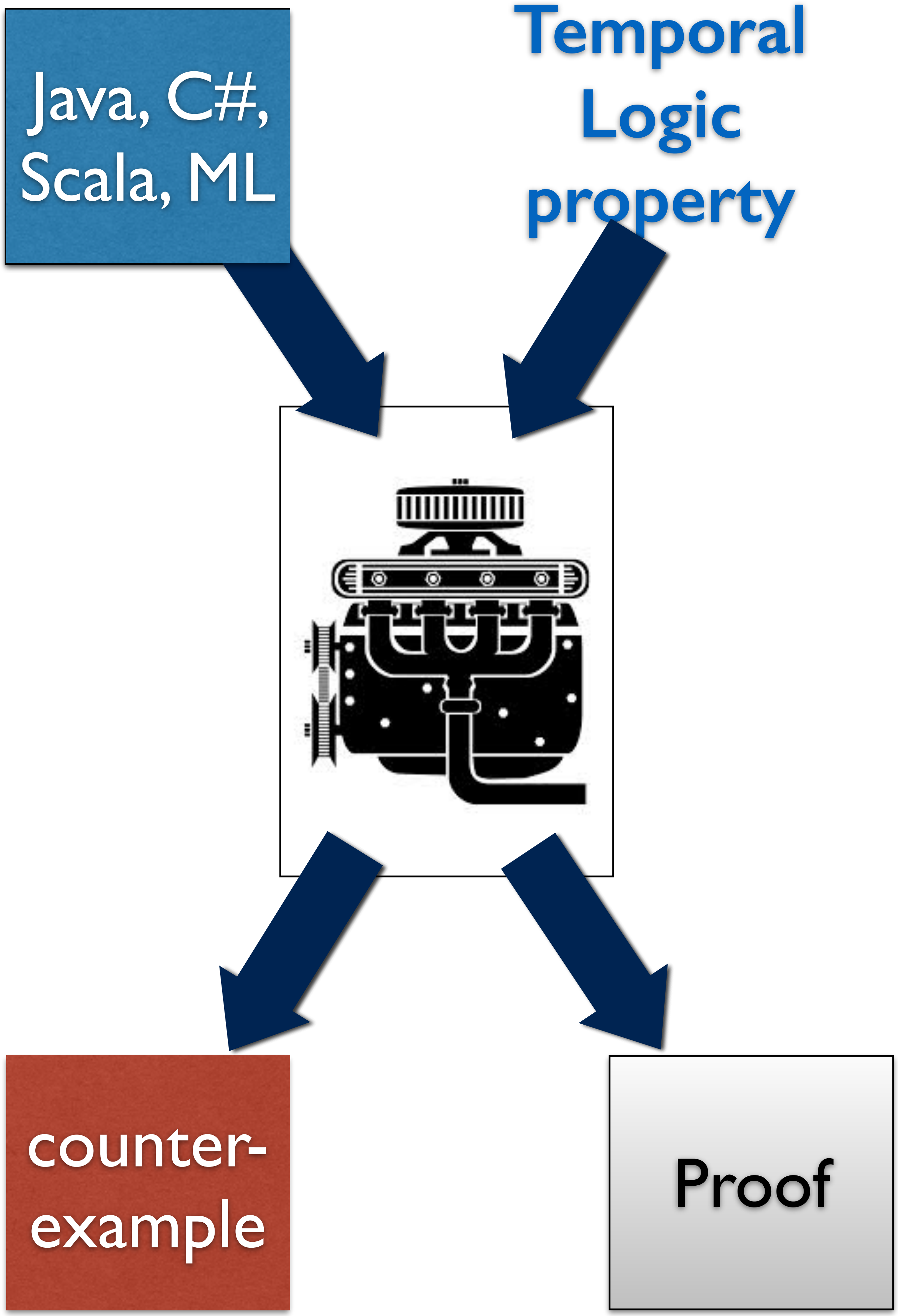
Experiments



timeout

naïve adaptations of automata theory

Higher-order programs



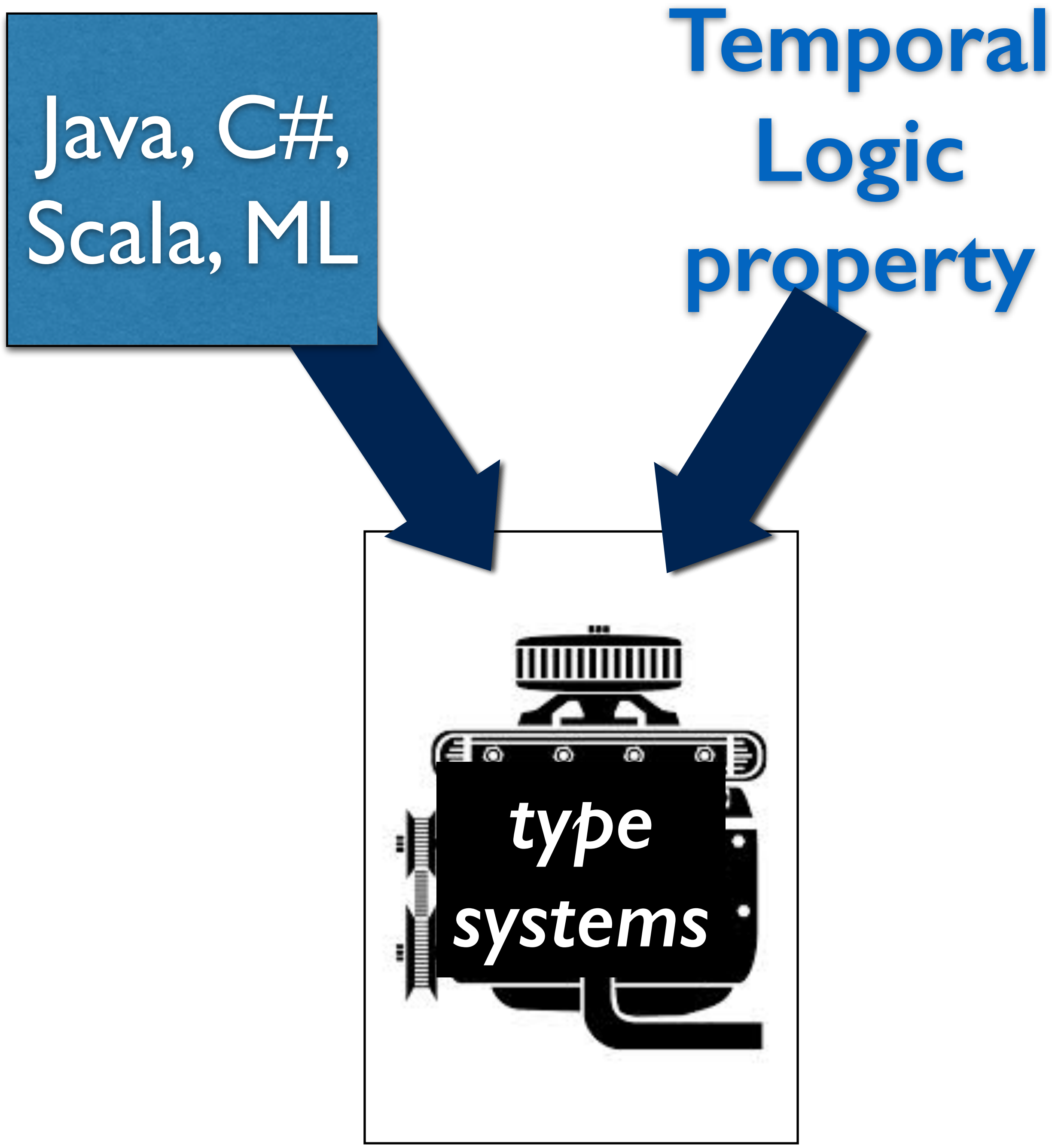
Java, C#, Scala, ML

Temporal Logic property

counter-example

Proof

Higher-order programs



Java, C#, Scala, ML

Temporal Logic property

type systems

Support from the Japan Society for the Promotion of Science

```
let rec halt _ =
  halt ()
and shrink f =
  if ( f() = 0 ) then
     halt ()
  else
     shrink (λ_. f() - 1)

and main() =
   let t = *⁺ in
     shrink (λ_. t)
```

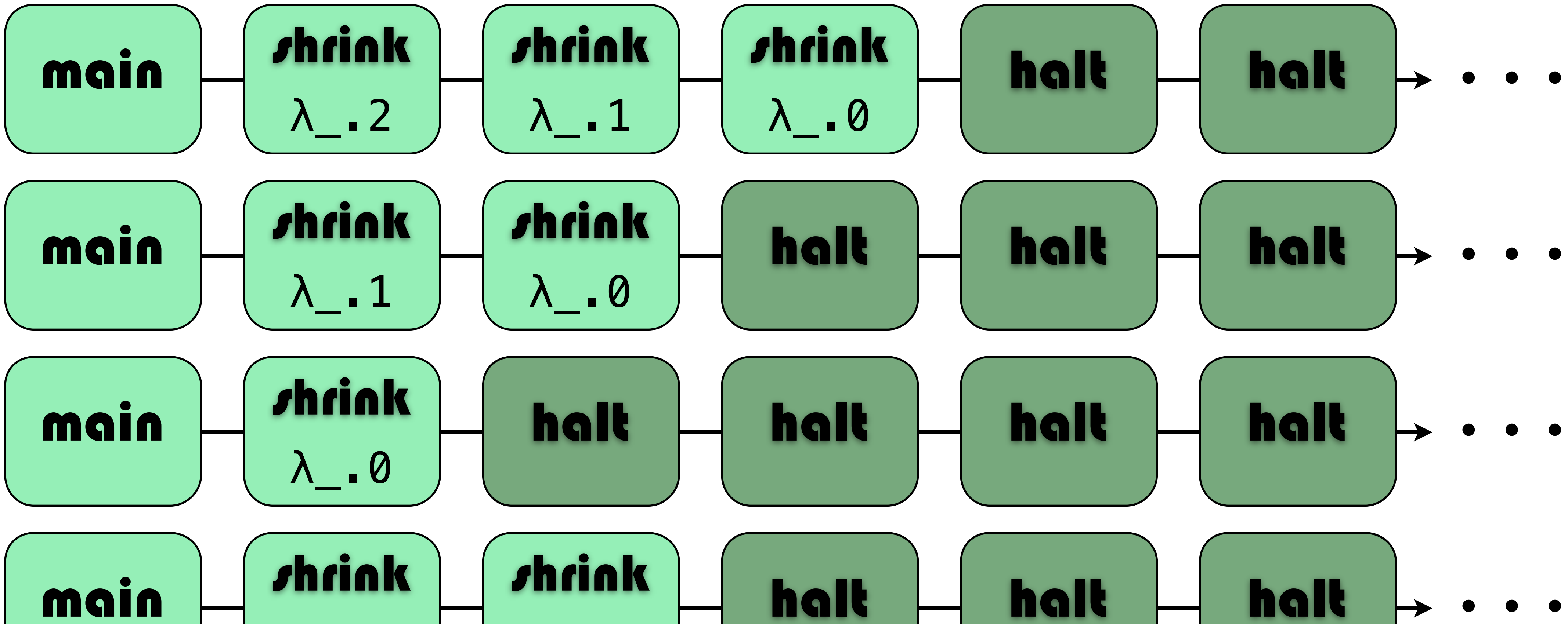
```
let rec halt _ = ev[halt];
  halt ()
and shrink f = ev[shrink];
  if ( f() = 0 ) then
     halt ()
  else
     shrink (λ_. f() - 1)

and main() = ev[main];
   let t = *+ in
     shrink (λ_. t)
```

Events

main X (shrink U halt)

```
let rec halt _ =
   halt ()
and shrink f =
   if ( f() = 0 ) then
        halt ()
   else
        shrink (λ_. f() - 1)

and main() =
    let t = *+ in
        shrink (λ_. t)
```

main X (shrink U halt)



Infinite state space

Reachability

```
let rec halt _ =
  halt ()
and shrink f =
  if ( f() = 0 ) then
     halt ()
  else
     shrink (λ_. f() - 1)

and main() =
   let t = *⁺ in
     shrink (λ_. t)
```

Higher Order

Non-determinism

main X (shrink U halt)

Termination

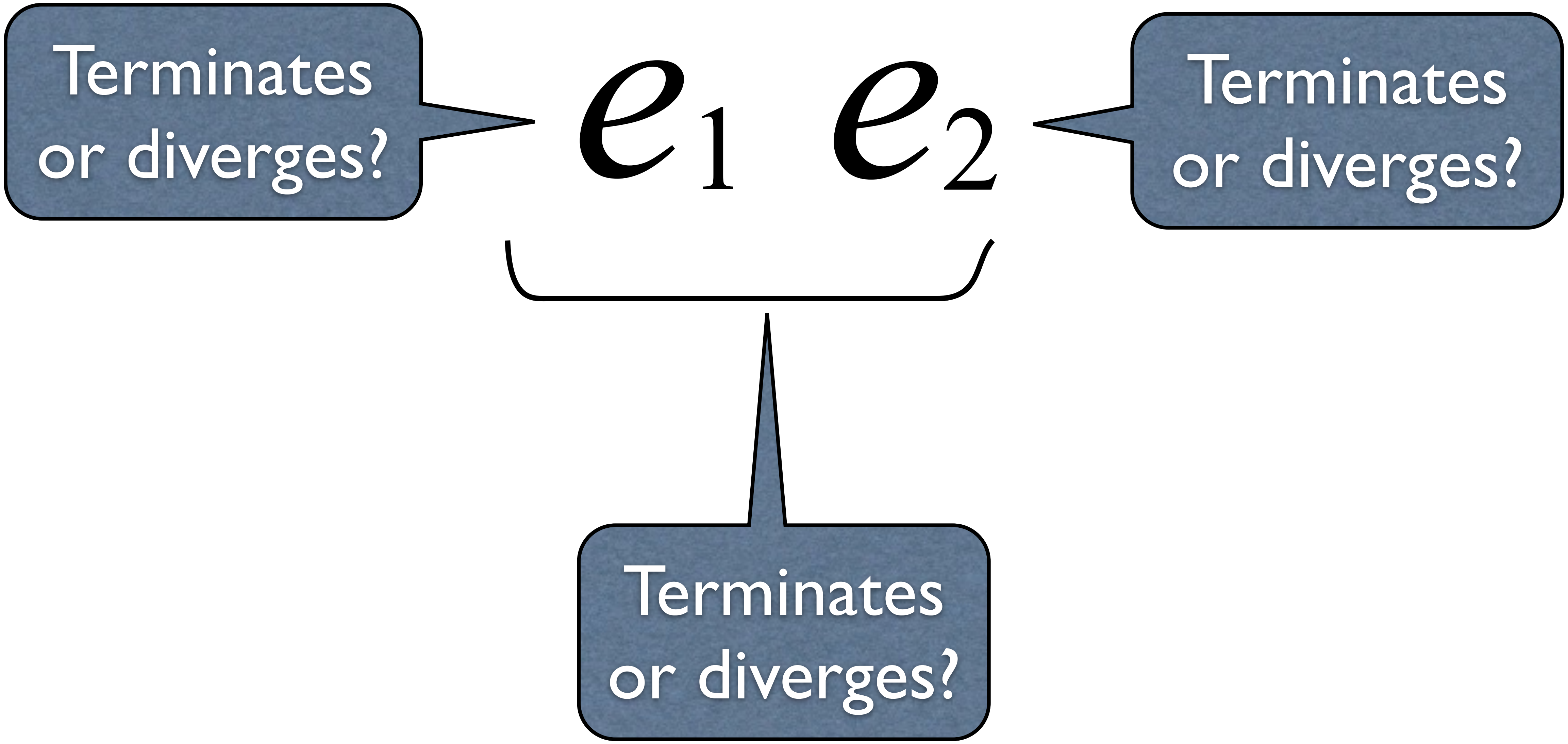
No previous technique can prove this property.

Previously:

- Expressive logics, but finite data [K/O:LICS'09]
- Infinite data, but just safety [Terauchi:POPL'10]
- Expressive logics, but first-order programs [CK:PLDI'13]

`shrink (λ_. t)`

(shrink ∪ halt)



Terminates or diverges?

$$\underbrace{e_1 \quad e_2}$$

Terminates or diverges?

Terminates or diverges?

Decompose in two ways

1. Divide up program into expressions

$$\underbrace{e_1 \; e_2}_{\Phi_3}$$

Φ_1

Temporal behavior as e_1 is reduced

Φ_2

Temporal behavior as e_2 is reduced

Φ_3

Latent behavior during application

$$\Phi_1 \cdot \Phi_2 \cdot \Phi_3$$

Decompose in two ways

1. Divide up program into expressions

Characterize temporal behavior of exprs. via type-and-effect:

$$\underbrace{\Gamma}_{\text{Typing environment}} \vdash e_1\ e_2 : \underbrace{\tau}_{\text{Dependent } \textit{Type}} \ \&\ \underbrace{\Phi}_{\text{Temporal } \textit{Effect}}$$

Decompose in two ways

1. Divide up program into expressions

Characterize temporal behavior of exprs. via type-and-effect:

$$\Gamma \vdash e_1\ e_2 : \tau \mathrel{\&} \Phi$$

Typing environment

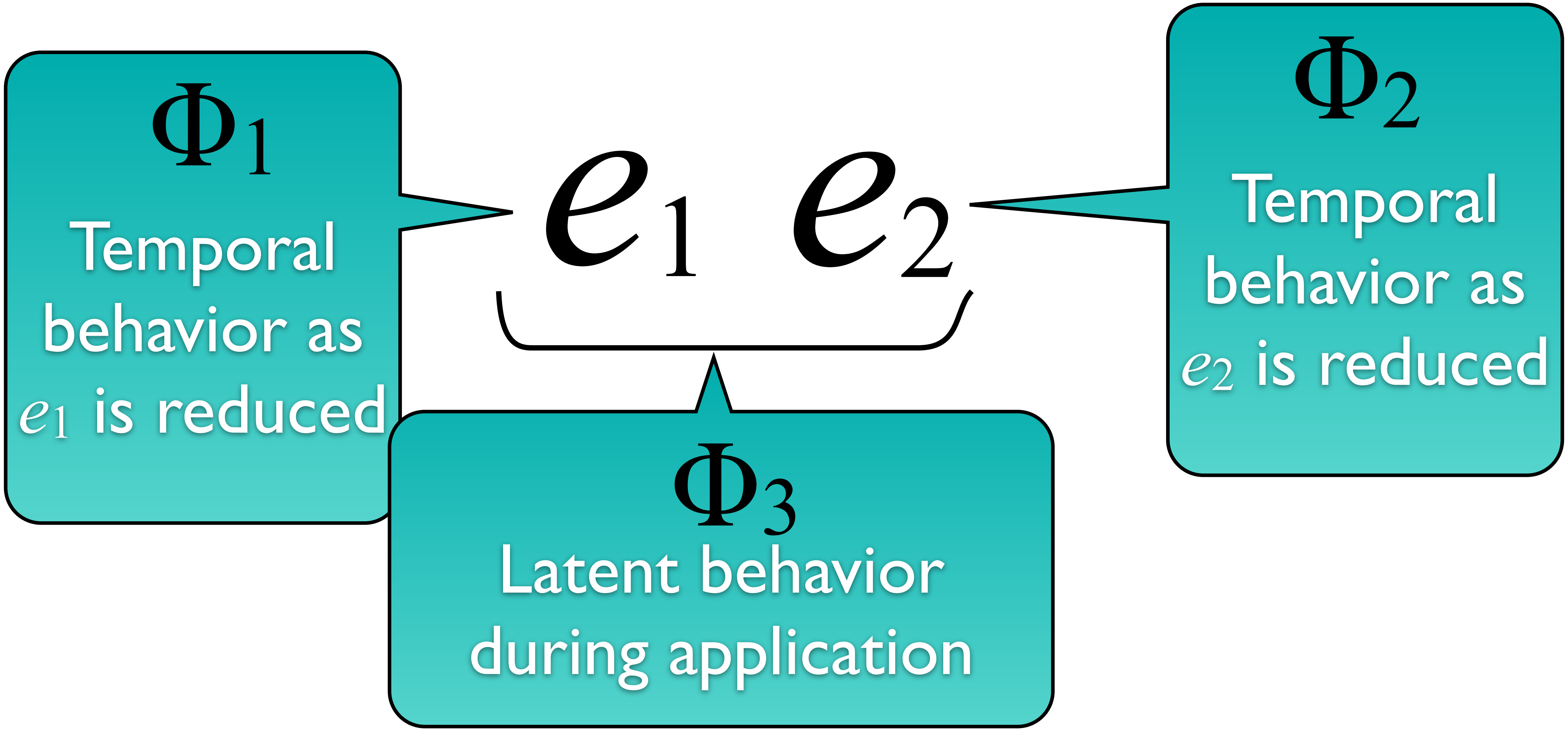
Dependent *Type*

Temporal *Effect*

Decompose in two ways

1. Divide up program into expressions

2. Track behavior of *finite* traces separate from *infinite* traces



$$\underbrace{e_1\ e_2}$$

Φ_1

Temporal behavior as e_1 is reduced

Φ_2

Temporal behavior as e_2 is reduced

Φ_3

Latent behavior during application

$$\Phi_1 \cdot \Phi_2 \cdot \Phi_3$$

Decompose in two ways

1. Divide up program into expressions

2. Track behavior of *finite* traces separate from *infinite* traces

$\Phi_1^{\text{fin}}, \Phi_1^{\text{inf}}$

Temporal behavior as e_1 is reduced

$$\underbrace{e_1\ e_2}$$

$\Phi_2^{\text{fin}}, \Phi_2^{\text{inf}}$

Temporal behavior as e_2 is reduced

$\Phi_3^{\text{fin}}, \Phi_3^{\text{inf}}$

Latent behavior during application

$$(\Phi_1^{\text{fin}}, \Phi_1^{\text{inf}}) \cdot (\Phi_2^{\text{fin}}, \Phi_2^{\text{inf}}) \cdot (\Phi_3^{\text{fin}}, \Phi_3^{\text{inf}})$$

Decompose in two ways

1. Divide up program into expressions

2. Track behavior of *finite* traces separate from *infinite* traces

$$\Gamma \vdash e : \tau \mathbin{\&} (\Phi^{\text{fin}}, \Phi^{\text{inf}})$$

```
let rec halt _ =
  halt ()
and shrink f =
  if ( f() = 0 ) then
     halt ()
  else
     shrink (λ_. f() - 1)

and main() =
   let t = ∗⁺ in
     shrink (λ_. t)
```

main X (shrink U halt)

```
let rec halt _ =
   halt ()
and shrink f =
   if ( f() = 0 ) then
        halt ()
   else
        shrink (λ_. f() - 1)

and main() =
    let t = *+ in
        shrink (λ_. t)
```

Latent Effect

$(\mathsf{shrink} \cup \mathsf{halt})$

Safety: the latent behavior when `shrink` is applied

$$\Gamma \vdash \mathsf{shrink} : (\mathtt{unit} \to \mathtt{int}) \xrightarrow{\mathsf{shrink} \vee \mathsf{halt}} \mathtt{unit} \& (\varepsilon, \varepsilon)$$

```
let rec halt _ =
   halt ()
and shrink f =
   if ( f() = 0 ) then
       halt ()
   else
       shrink (λ_. f() - 1)

and main() =
    let t = *⁺ in
       shrink (λ_. t)
```

(**shrink** U **halt**)

refinement types

“all traces exit shrink”

Safety: the latent behavior when `shrink` is applied

$$\Gamma \vdash \mathsf{shrink} : (\mathtt{unit} \rightarrow \mathtt{int}) \xrightarrow{\mathbf{shrink}\,\mathcal{W}\,\mathbf{halt}} \mathtt{unit}\&(\varepsilon, \varepsilon)$$

Liveness: the conditions under which `shrink` terminates

$$\Gamma, \mathsf{f} : \mathtt{unit} \rightarrow \{i \mid i \geq 0\} \vdash \mathsf{shrink\ f} : \mathtt{unit}\,\&\,(\top, \mathsf{F}\ \neg\mathbf{shrink})$$

```
let rec halt _ =
  halt ()
and shrink f =
  if ( f() = 0 ) then
     halt ()
  else
     shrink (λ_. f() - 1)

and main() =
   let t = *⁺ in
     shrink (λ_. t)
```

(shrink ∪ halt)

Safety

…

App

⋮

Liveness

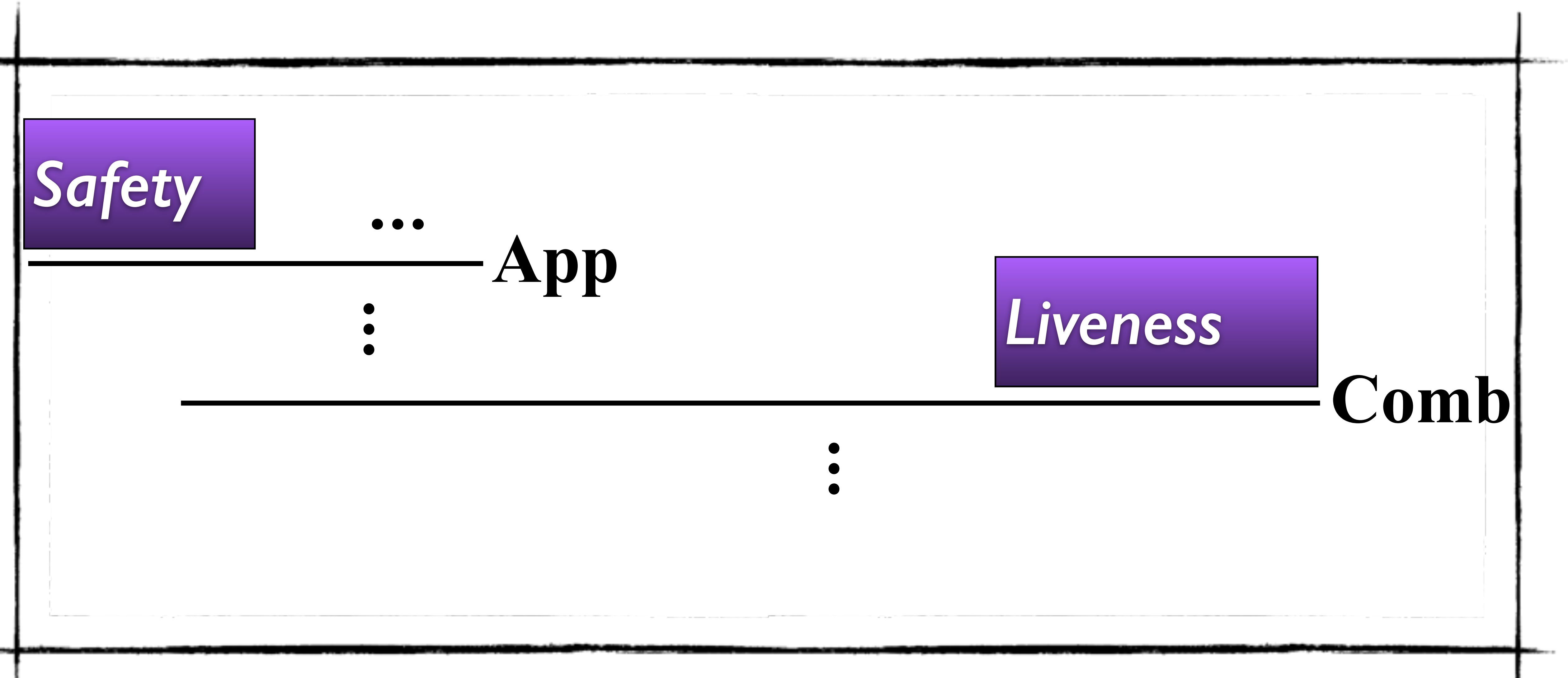
Comb

⋮

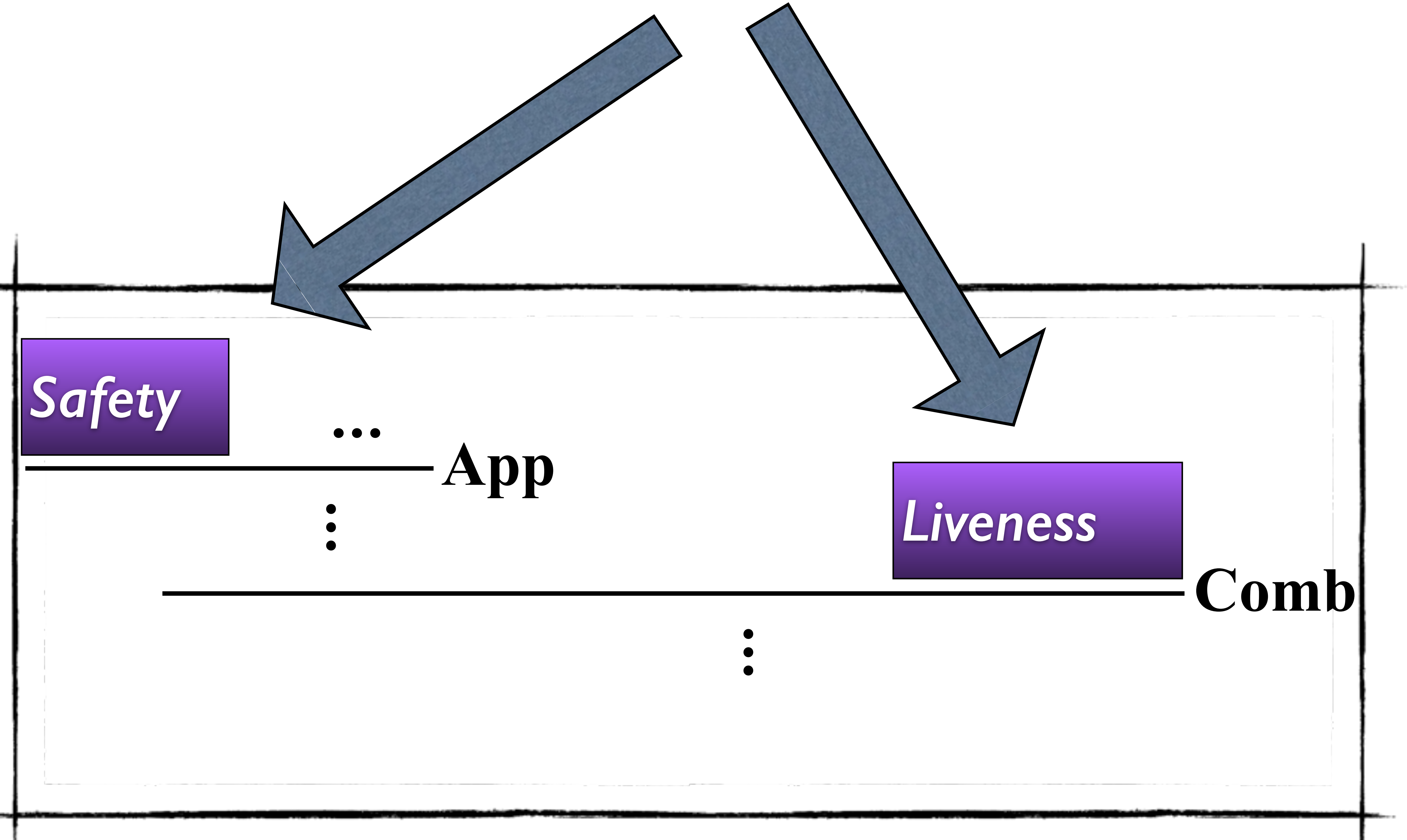
Sub

$\Gamma, \mathrm{t} : \{i \mid i \geq 0\} \vdash$ shrink (λ_. t) : unit & (shrink ∪ halt)

Liveness in a type system ???



Where do these pieces come from?



Safety

…

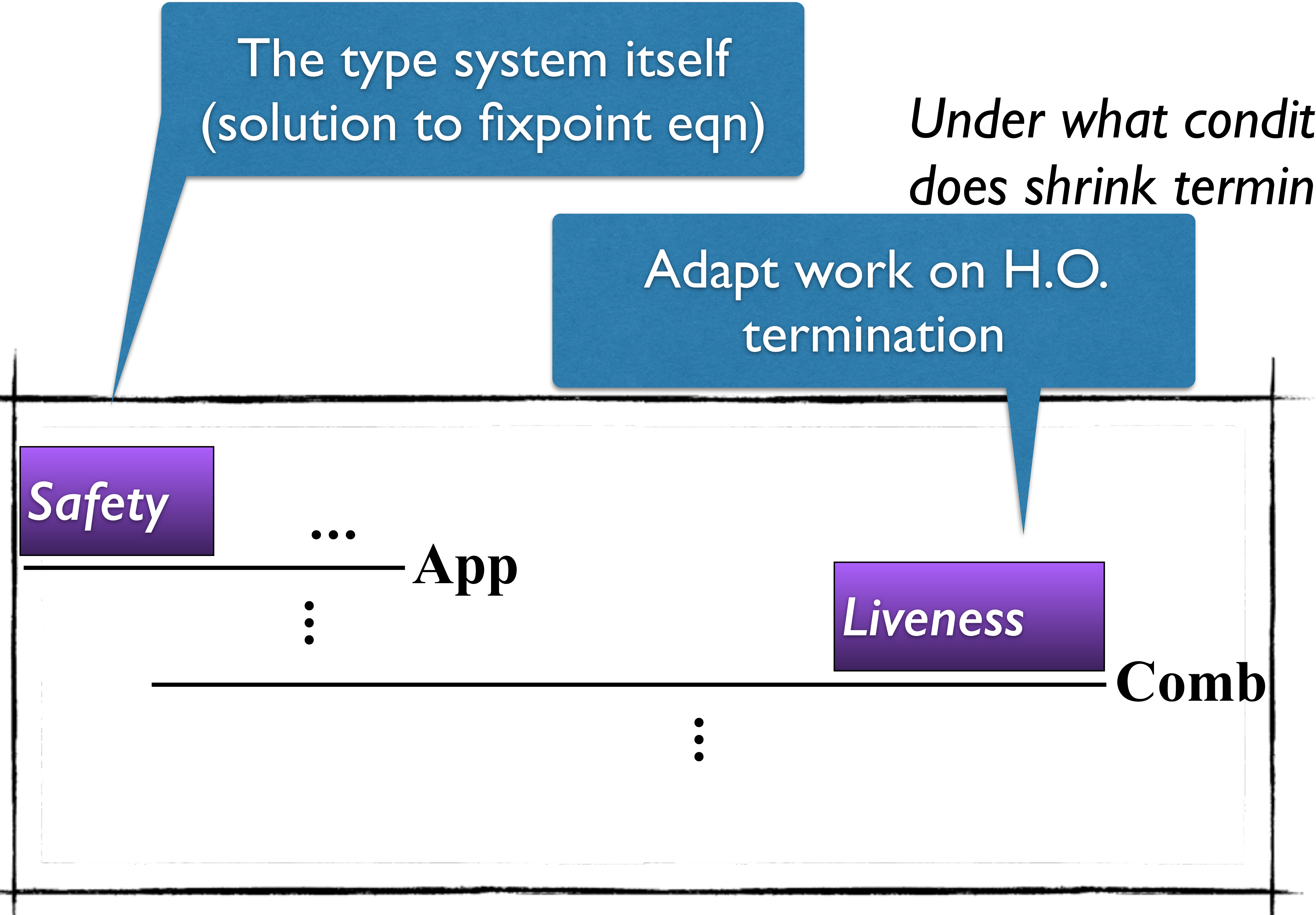
App

⋮

Liveness

Comb

⋮



The type system itself (solution to fixpoint eqn)

Under what conditions does shrink terminate?

Adapt work on H.O. termination

Safety

…

App

⋮

Liveness

Comb

⋮

Safety

$$\cfrac{\cfrac{\cdots}{\vdots}\;\textbf{App}}{\vdots}\;\textbf{Comb}$$

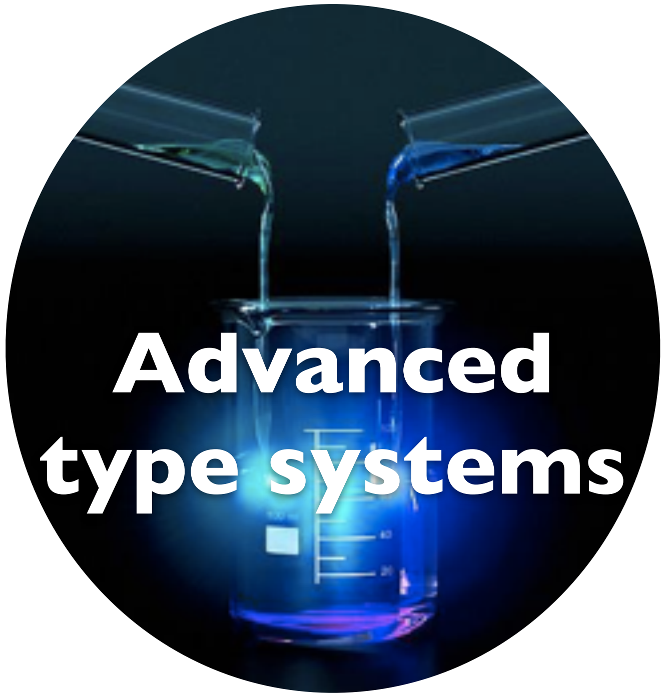
Liveness

Advanced type systems

LICS'14

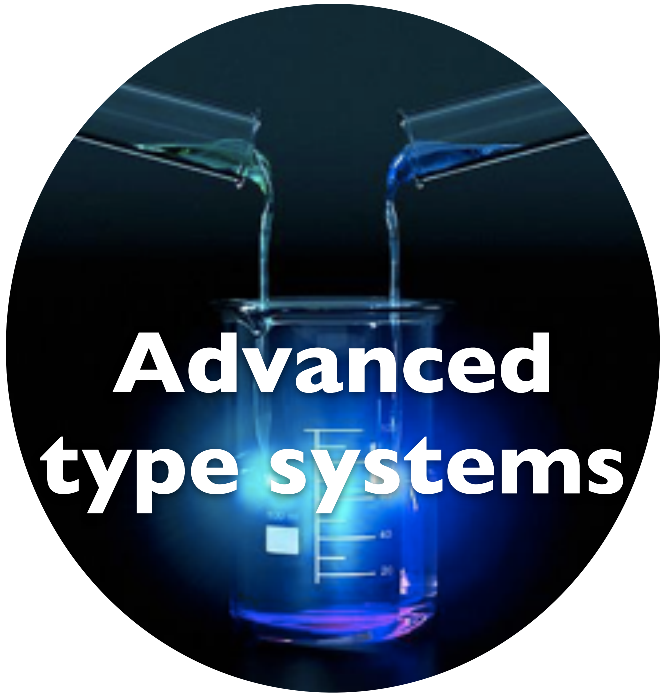
Why is this important?

- First technique for temporal properties of higher-order, infinite-data programs
- Instantiation to wide variety of spec. logics, Instantiation to type environments, Instantiation to oracles



Why is this important?

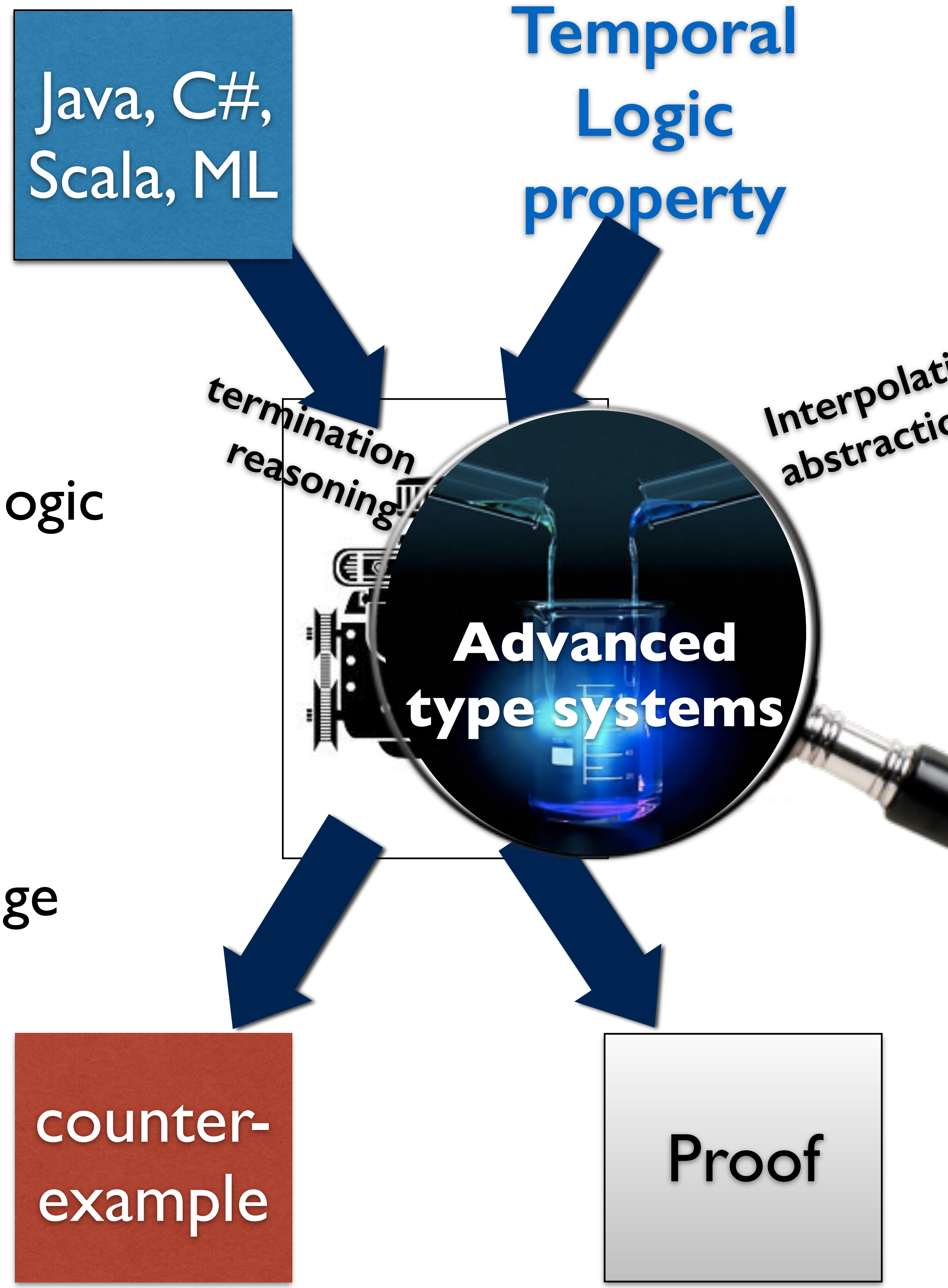
- Compositional
- Does not require input program be in continuation-passing style (CPS)
- First-order interprocedural programs



Next Steps

Innovations Needed!

- Type systems
- Formal methods, temporal logic
- Abstraction refinement
- Algorithms
- Scalable program analysis
- Systems, experiments
- Temporal Logic of Knowledge



Java, C#, Scala, ML

Temporal Logic property

termination reasoning

Interpolati abstractio

Advanced type systems

counter-example

Proof

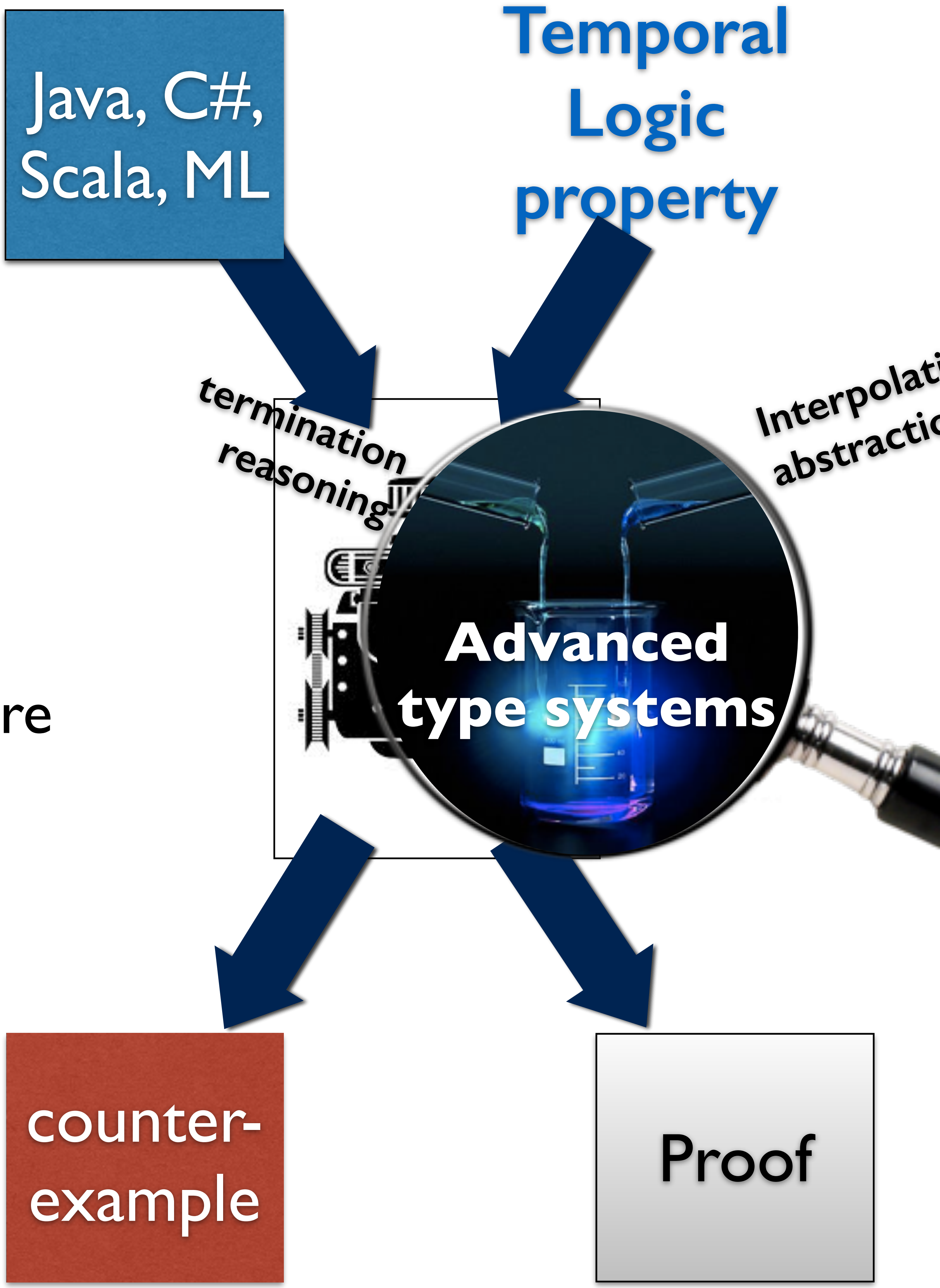
Conclusion

Temporal Logic

- Safety properties
- Liveness properties
- Mixtures
- Worked well for hardware
- Need techniques for software

2025 Languages

- Java, C#, Scala, ML



Java, C#, Scala, ML

Temporal Logic property

termination reasoning

Interpolati
abstractio

Advanced type systems

counter-example

Proof

Thank you!