Towards Certified Separate Compilation for Concurrent Programs

Hanru Jiang*   Hongjin Liang†
Siyang Xiao*   Junpeng Zha*   Xinyu Feng†

* University of Science and Technology of China
† Nanjing University

Compilers are *NOT* Trustworthy

Finding and Understanding Bugs in C Compilers

Xuejun Yang Yang Chen Eric Eide John Regehr
University of Utah, School of Computing
{jxyang, chenyang, eeide, regehr}@cs.utah.edu

[PLDI 2011]

- 11 open-source/commercial compilers were tested
- Found **325** bugs, in **EVERY** compiler!

Compilers are *NOT* Trustworthy

Finding and Understanding Bugs in C Compilers

Xuejun Yang Yang Chen Eric Eide John Regehr
University of Utah, School of Computing
{jxyang, chenyang, eeide, regehr}@cs.utah.edu

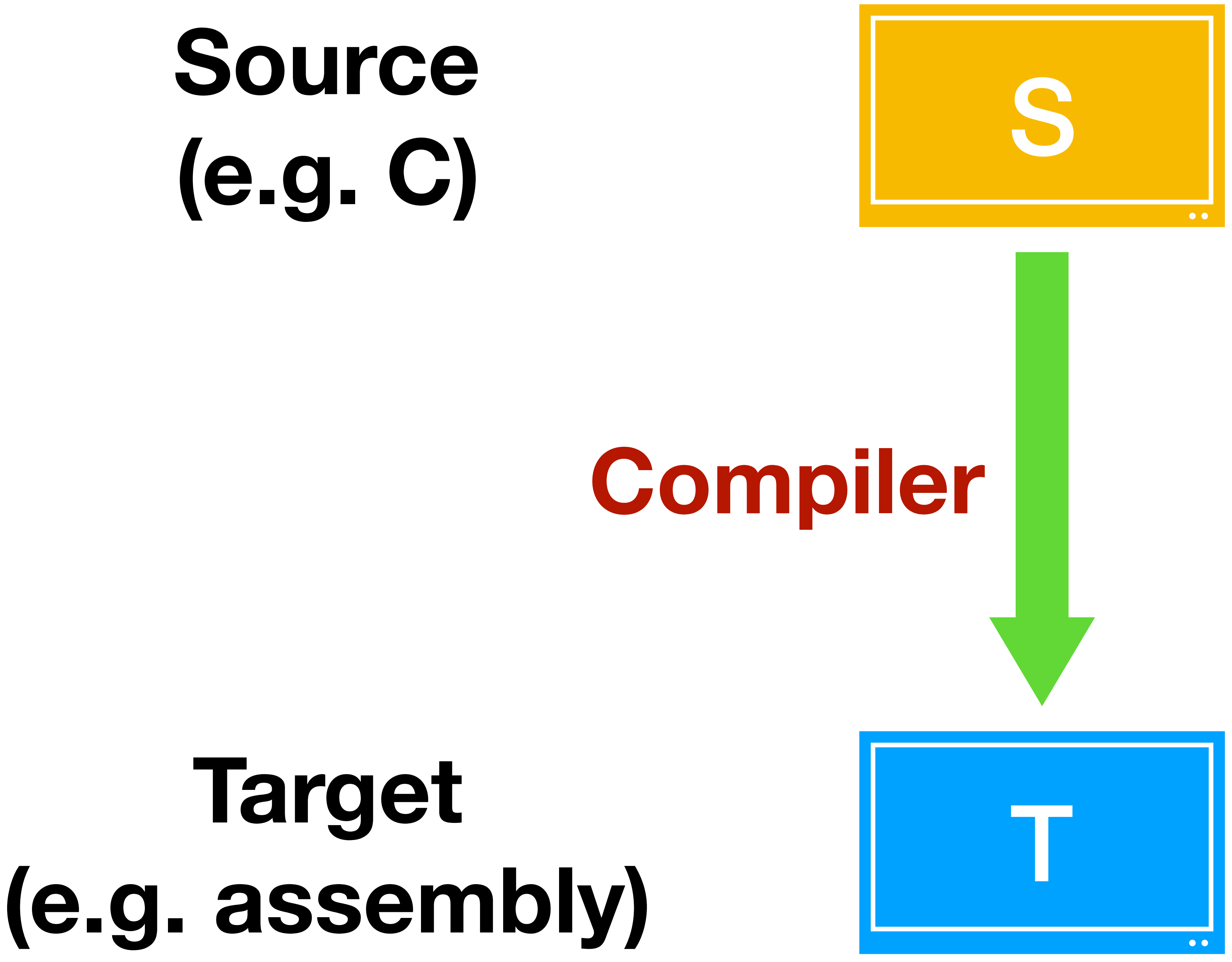
[PLDI 2011]

- 11 open-source/commercial compilers were tested
- Found **325** bugs, in **EVERY** compiler!

Verification of compiler correctness helps:

"The striking thing about our CompCert results is that the middle end bugs we found in all other compilers are absent."

Compilation Correctness



Correct(Compiler) :

$$\forall S, T\ .\ T = \mathbf{Compiler}(S) \implies T \subseteq S$$

Semantic preservation:
T has no more observable behaviors (e.g. I/O events by print) than S.

Compiler Verification

Compiler Verification

- Leroy’06: Formal certification of a compiler back-end
- Lochbihler’10: Verifying a compiler for Java threads
- Myreen’10: Verified just-in-time compiler on x86
- Sevcik et al.’11: Relaxed-memory concurrency and verified compilation
- Zhao et al.’13: Formal verification of SSA-based optimizations for LLVM
- Kumar et al.’14: CakeML: A verified implementation of ML
- Stewart et al.’15: Compositional CompCert
- Kang et al.’16: Lightweight Verification of Separate Compilation
- …

Compiler Verification

- Leroy’06: Formal certification of a compiler back-end
- Lochbihler’10: Verifying a compiler for Java threads
- Myreen’10: Verified just-in-time compiler on x86

Limited support of separate compilation and concurrency!

- Kumar et al.’14: CakeML: A verified implementation of ML
- Stewart et al.’15: Compositional CompCert
- Kang et al.’16: Lightweight Verification of Separate Compilation
- …

Separate Compilation

Source (e.g. C)

Real-world programs may consist of multiple components, which will be compiled independently.

Separate Compilation

Source
(e.g. C)


```
// Module S1
extern void g(int *x);
int f(){
  int a = 0, b = 0;
  g(&b);
  return a + b; }
```

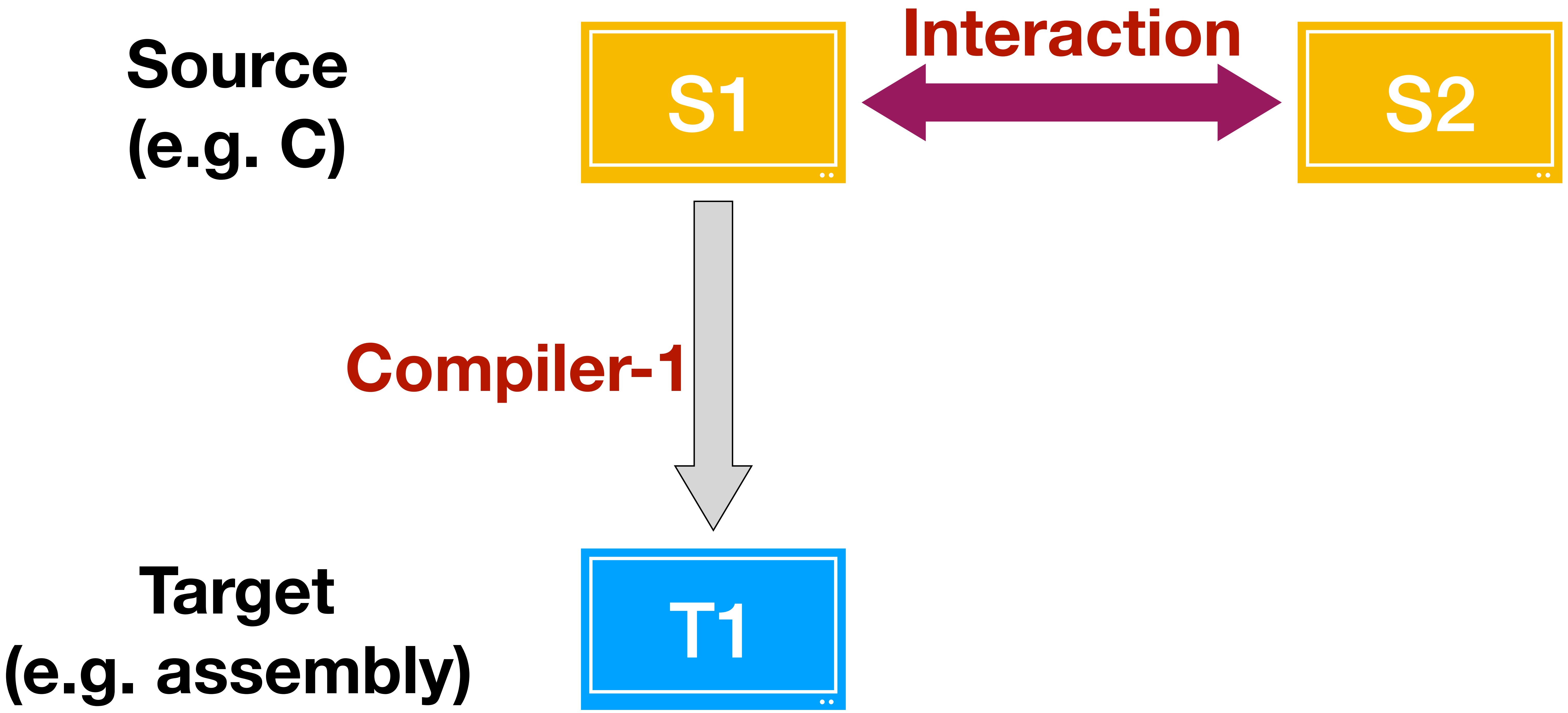
```
// Module S2
void g(int *x){
  *x = 3;
}
```

Real-world programs may consist of multiple components, which will be compiled independently.

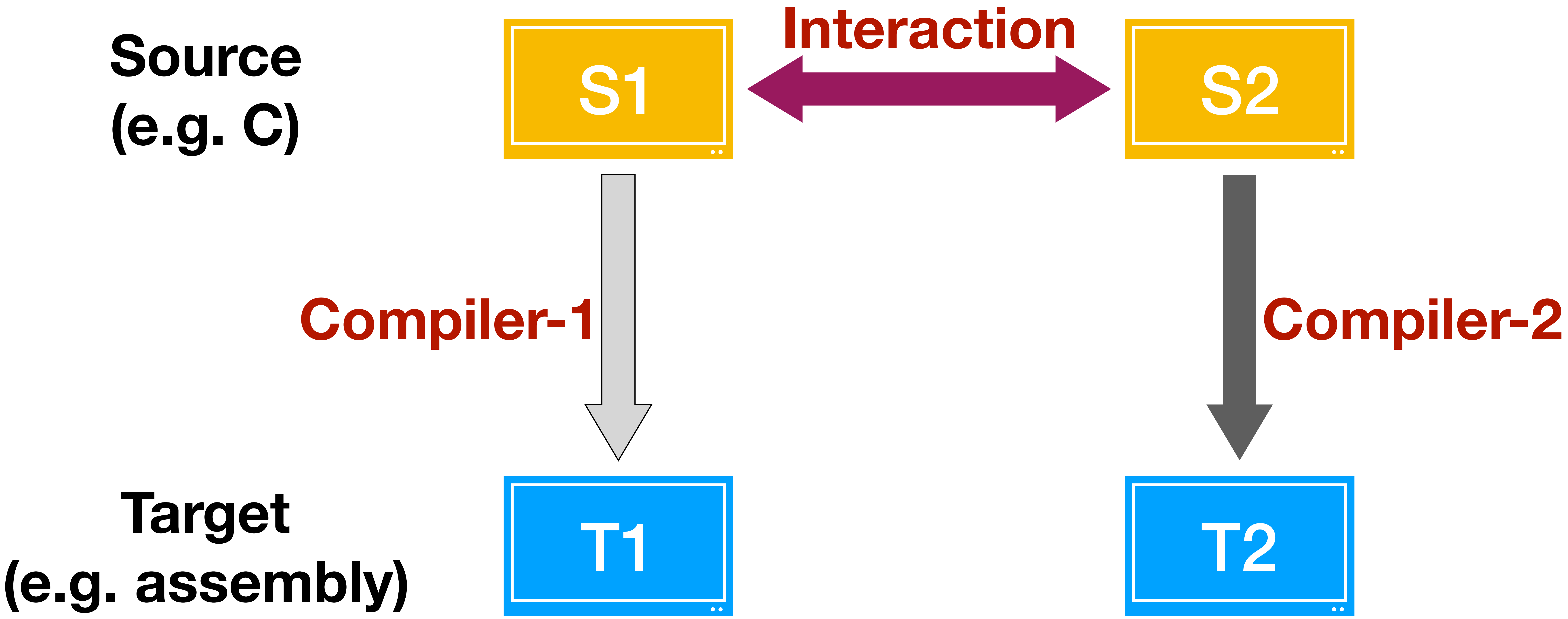
Separate Compilation

Source (e.g. C)

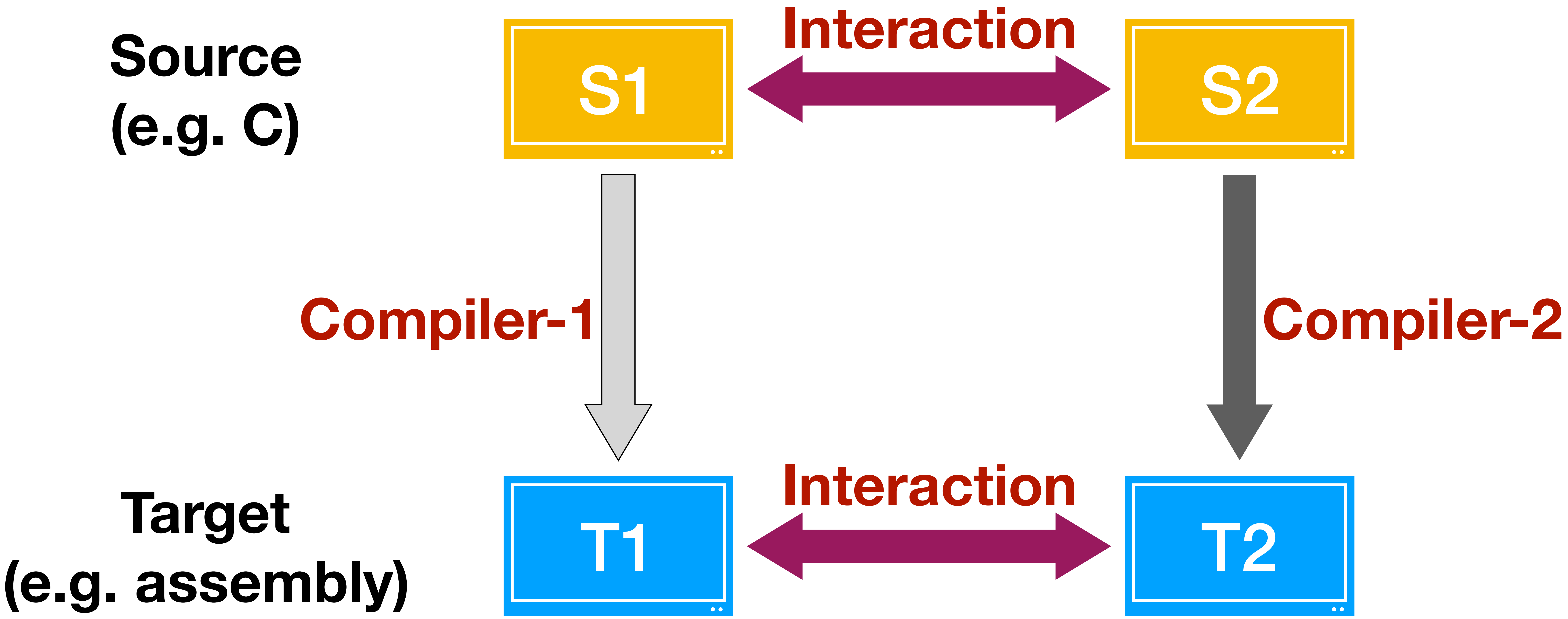
Separate Compilation



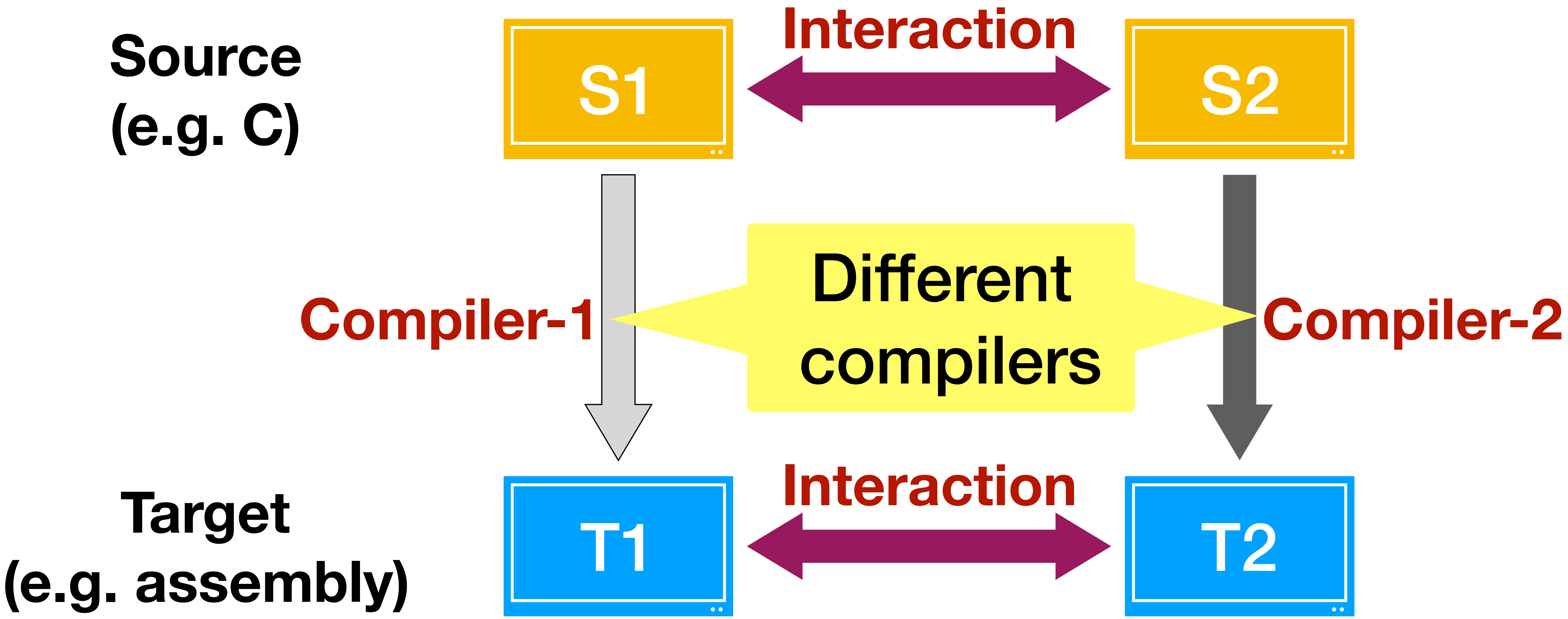
Separate Compilation



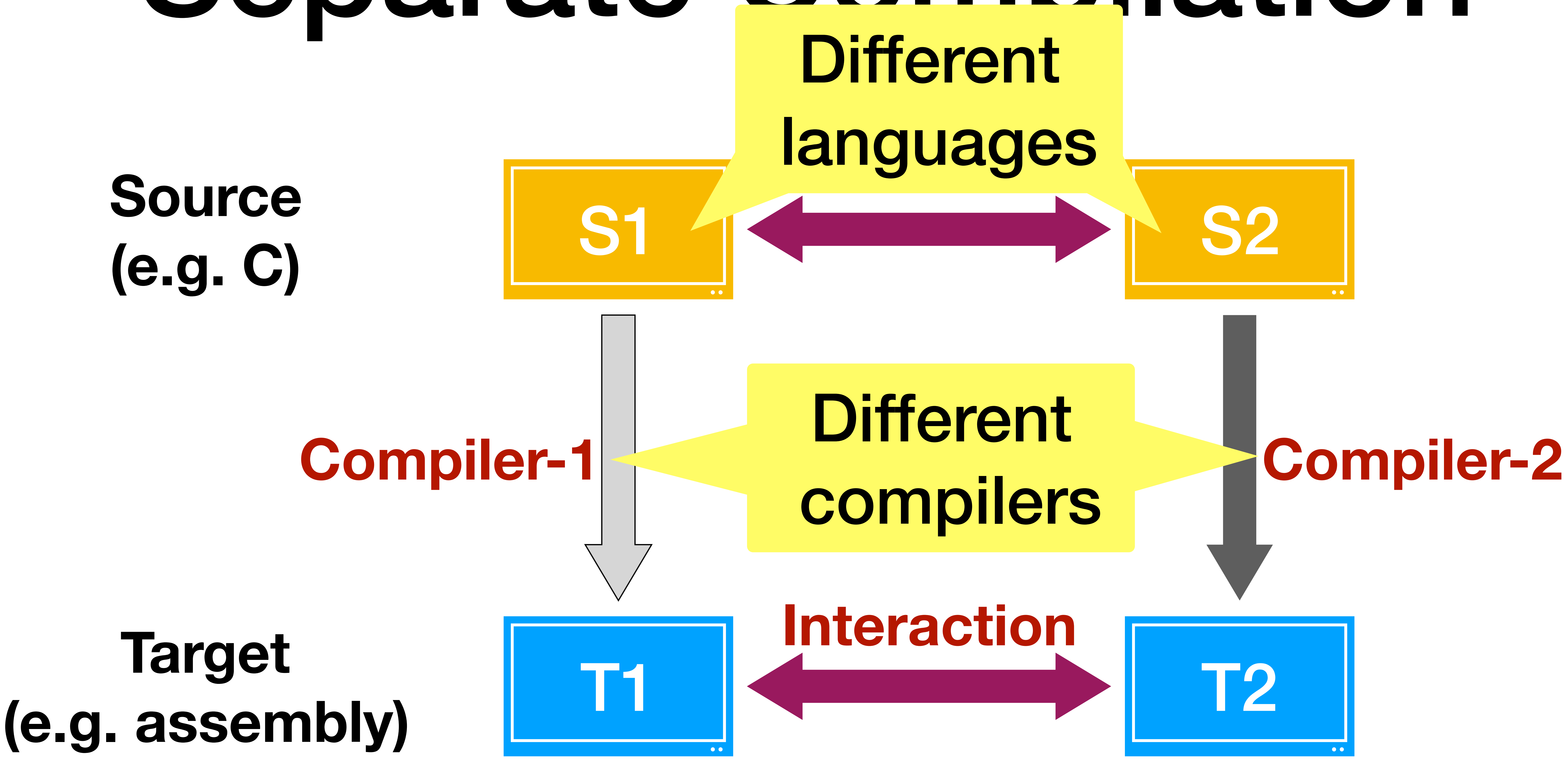
Separate Compilation



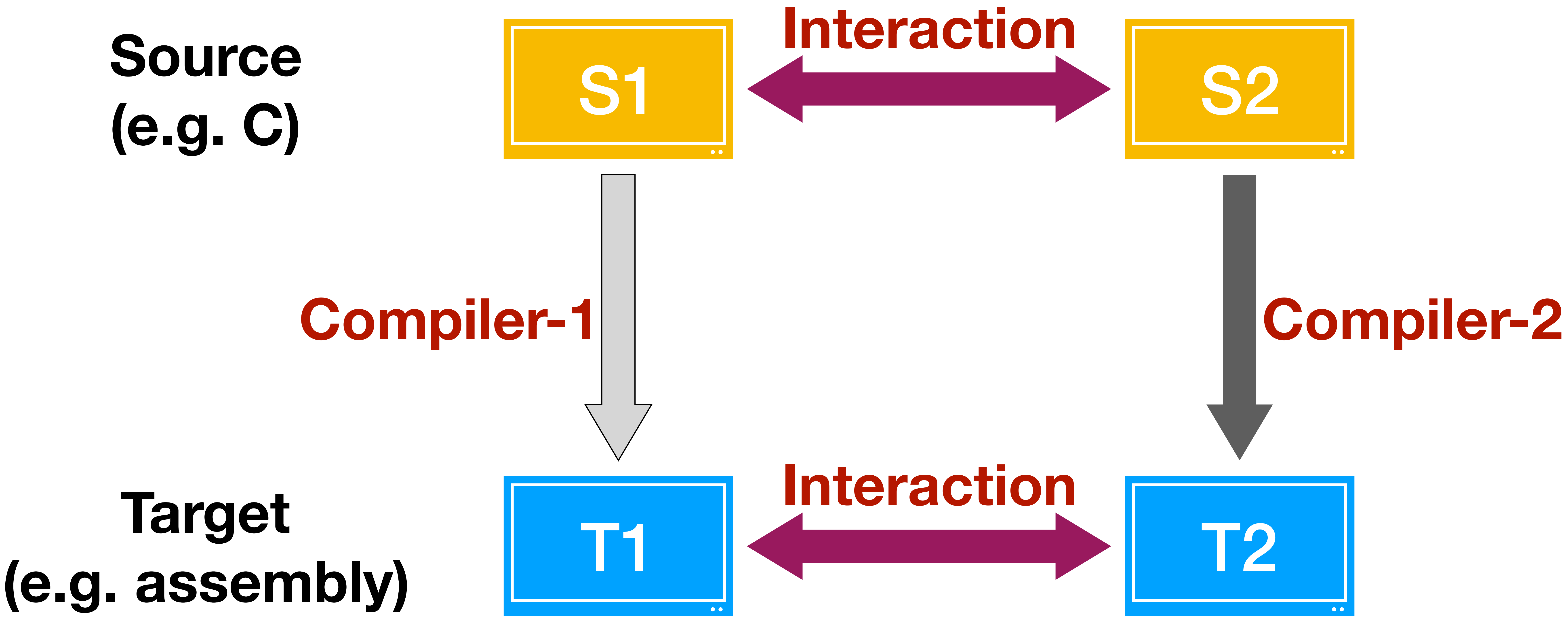
Separate Compilation



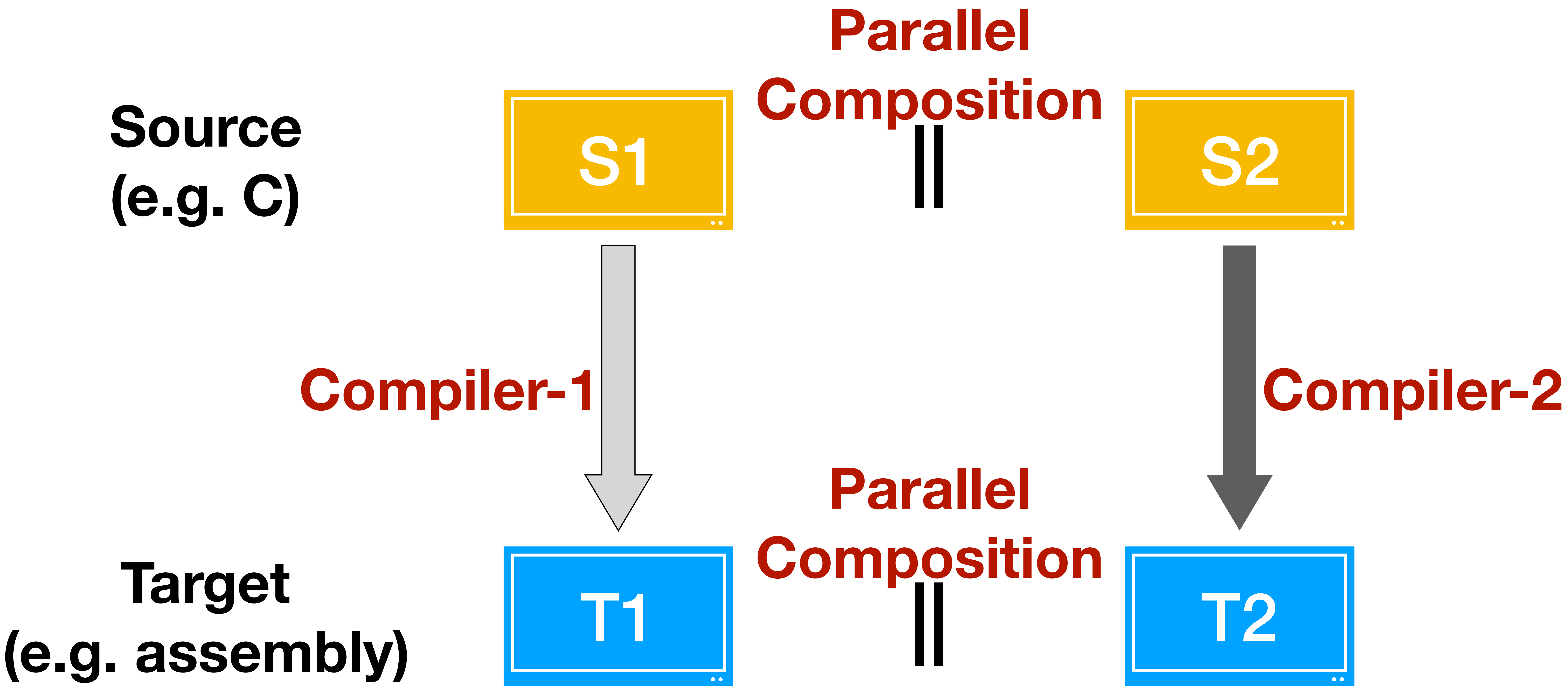
Separate Compilation



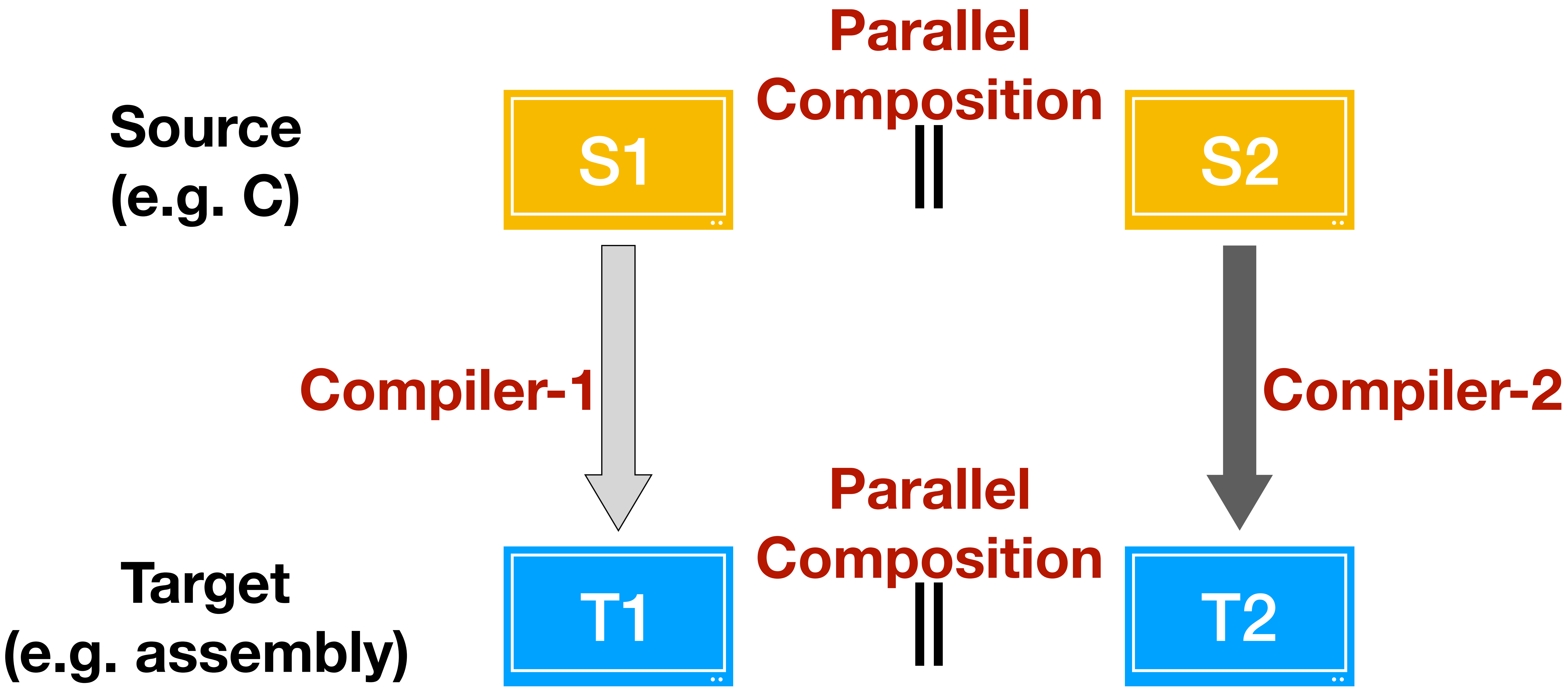
Separate Compilation of Concurrent Programs



Separate Compilation of Concurrent Programs



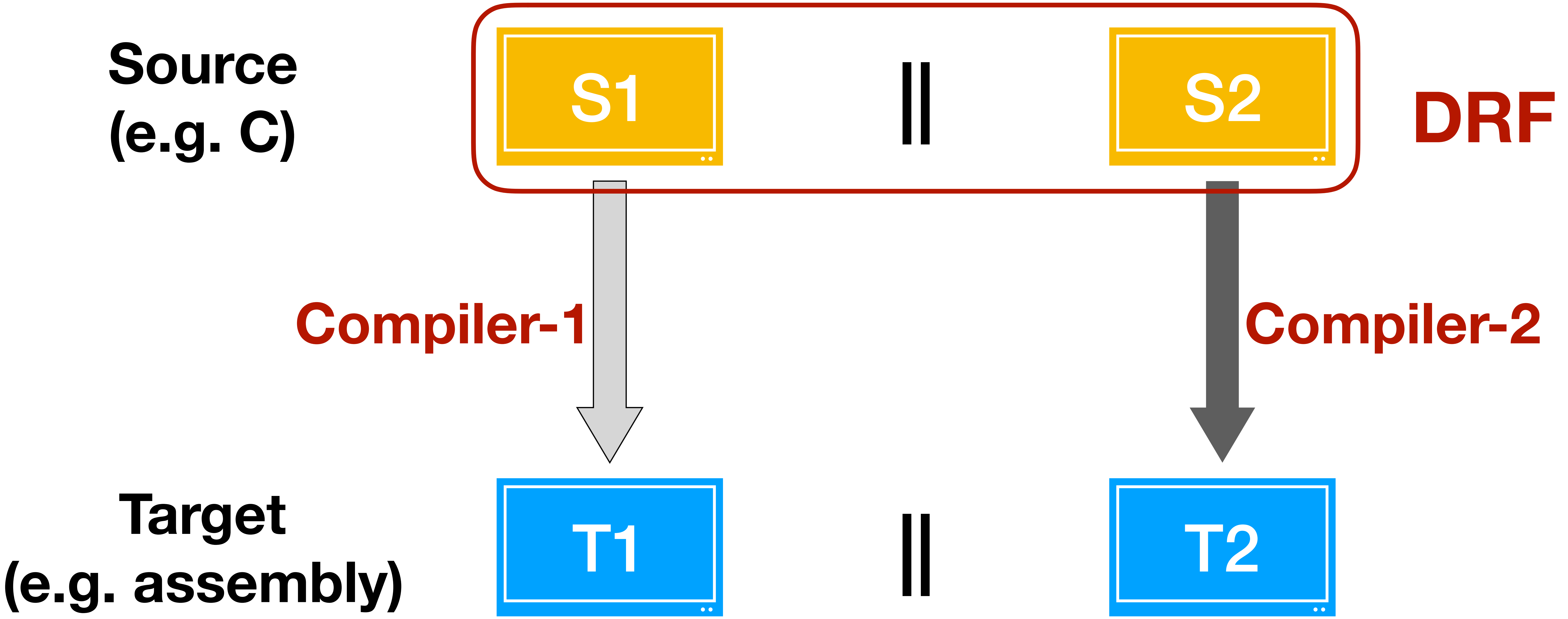
Separate Compilation of Concurrent Programs



Can we reuse existing certified compilers (e.g. CompCert) for separate compilation of concurrent programs?

Compositional CompCert's Argument…

[Stewart et al. POPL'15]



YES, for data-race-free (DRF) concurrent programs

Intuition of the Argument:
Interleaving <=> Non-preemptive for DRF Programs

```
r1 = 1;             ||   r2 = 2;
r1 = r1 + 1;        ||   r2 = r2 + 1;
lock();             ||   lock();
  x = 1;            ||     x = 2;
  y = x + 1;        ||     y = x + 1;
unlock();           ||   unlock();
```

interleaving

Intuition of the Argument:
Interleaving <=> Non-preemptive for DRF Programs

```
r1 = 1;              ||  r2 = 2;
r1 = r1 + 1;         ||  r2 = r2 + 1;
lock();              ||  lock();
  x = 1;             ||    x = 2;
  y = x + 1;         ||    y = x + 1;
unlock();            ||  unlock();
```

interleaving

No race

Intuition of the Argument: Interleaving <=> Non-preemptive for DRF Programs

```
r1 = 1;             ||  r2 = 2;
r1 = r1 + 1;        ||  r2 = r2 + 1;
lock();             ||  lock();
  x = 1;            ||    x = 2;
  y = x + 1;        ||    y = x + 1;
unlock();           ||  unlock();
```

interleaving

No race

Intuition of the Argument: Interleaving <=> Non-preemptive for DRF Programs

```
r1 = 1;              ||  r2 = 2;
r1 = r1 + 1;         ||  r2 = r2 + 1;
lock();              ||  lock();
  x = 1;             ||    x = 2;
  y = x + 1;         ||    y = x + 1;
unlock();            ||  unlock();
```

interleaving

No race

Non-preemptive: yield control at certain points only

```
sequential {  r1 = 1;        |  r2 = 2;
              r1 = r1 + 1;   |  r2 = r2 + 1;
              yield;         |  yield;
sequential {    x = 1;       |    x = 2;
                y = x + 1;   |    y = x + 1;
              yield;         |  yield;
```

Intuition of the Argument:
Interleaving <=> Non-preemptive for DRF Programs

```
r1 = 1;            ||  r2 = 2;
r1 = r1 + 1;       ||  r2 = r2 + 1;
lock();            ||  lock();
  x = 1;           ||    x = 2;
  y = x + 1;       ||    y = x + 1;
```

No race

Non-preemptive:
yield control at

Plausible, but need to address several key challenges

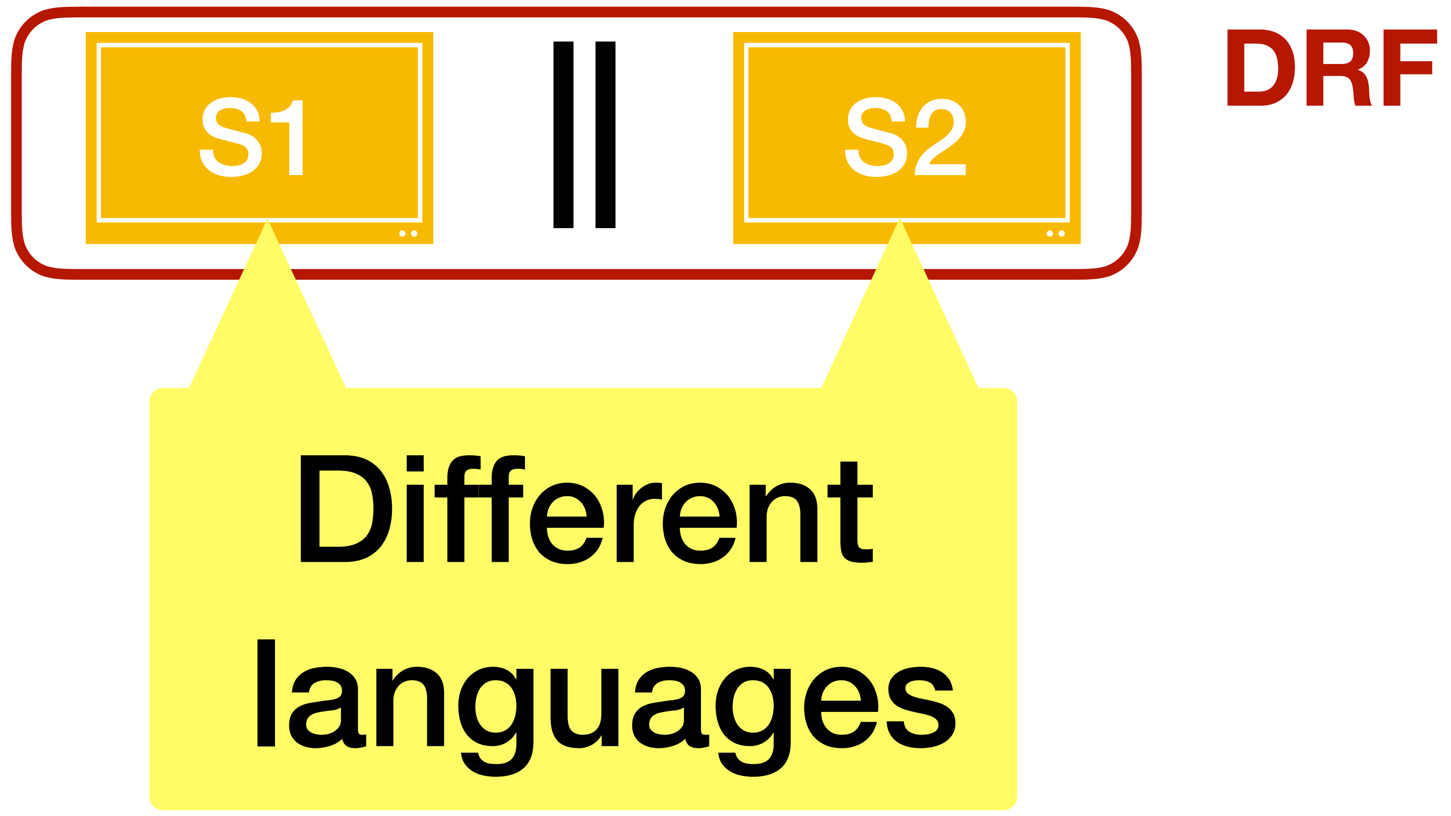
```
  r1 = r1 + 1;  |  r2 = r2 + 1;
yield;          |  yield;
  x = 1;        |    x = 2;
  y = x + 1;    |    y = x + 1;
yield;          |  yield;
```

sequential

Challenges

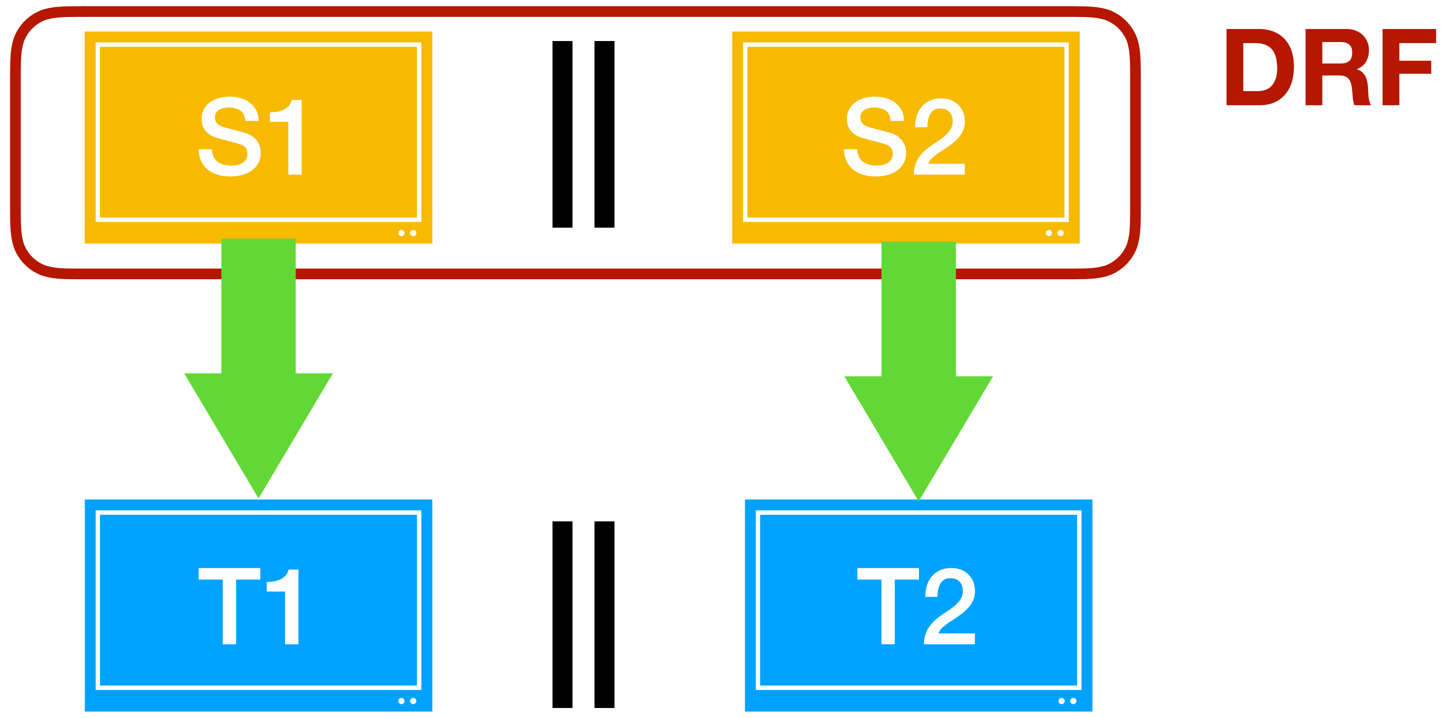
Challenges

- How to formulate **DRF** in **language independent** manner?



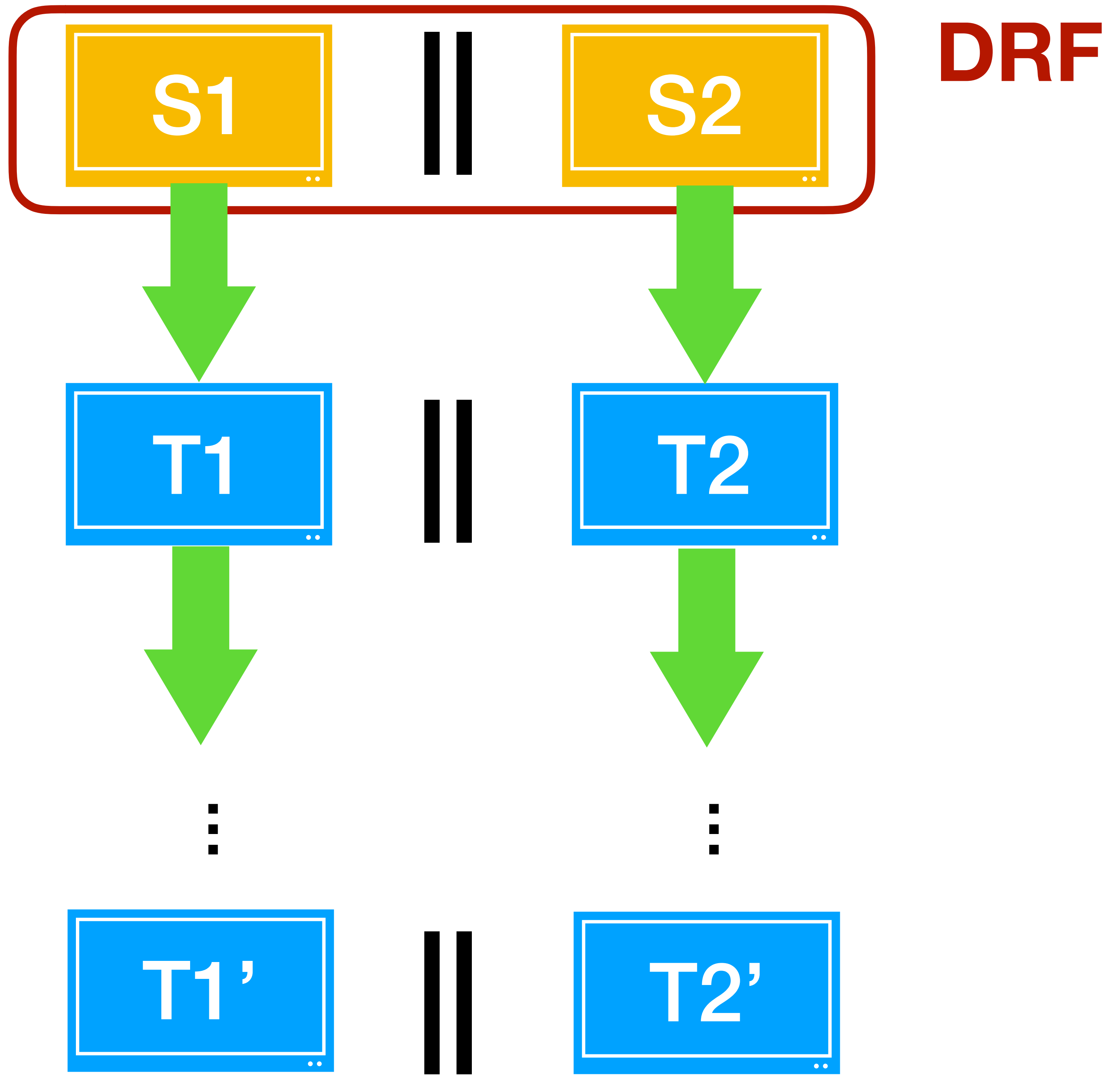
Challenges

- How to formulate **DRF** in **language independent** manner?
- How to prove **DRF-preservation**, compositionally?



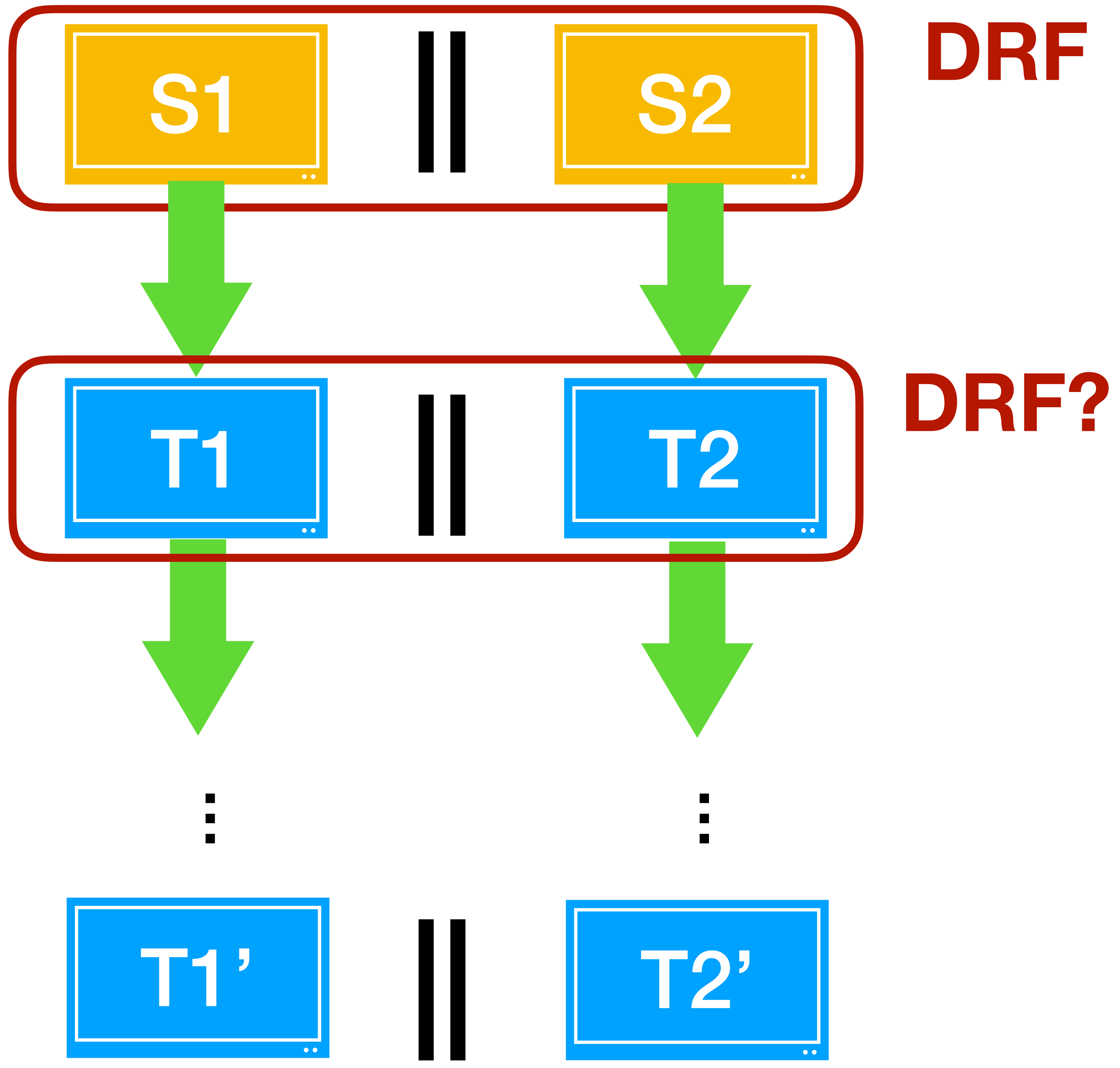
Challenges

- How to formulate **DRF** in **language independent** manner?
- How to prove **DRF-preservation**, compositionally?



Challenges

- How to formulate **DRF** in **language independent** manner?
- How to prove **DRF-preservation**, compositionally?



Challenges

- How to formulate **DRF** in **language independent** manner?
- How to prove **DRF-preservation**, compositionally?

Challenges

- How to formulate **DRF** in **language independent** manner?
- How to prove **DRF-preservation**, compositionally?
- How to support **benign-race** and **relaxed memory models**?

Challenges

- How to formulate **DRF** in **language independent** manner?
- How to prove **DRF-preservation**, compositionally?
- How to support **benign-race** and **relaxed memory models**?

"... synchronization primitives are commonly implemented with assembly code that has data races."

—— Hans-J. Boehm, HotPar'11

Our Work

Our Work

- **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
    - With both external function calls & multi-threaded code

Our Work

- **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
    - With both external function calls & multi-threaded code
- Framework extension:
  - Supports **x86-TSO** + **confined benign-races**

Our Work

- **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
    - With both external function calls & multi-threaded code
- Framework extension:
  - Supports **x86-TSO** + **confined benign-races**
- **CAS**CompCert:
  - Extends CompCert with **C**oncurrency + **A**bstraction + **S**eparate compilation
  - **Reuses** considerable amount of CompCert proofs
  - **Racy** x86-TSO impl. of locks as synchronization library

Outline of this Talk

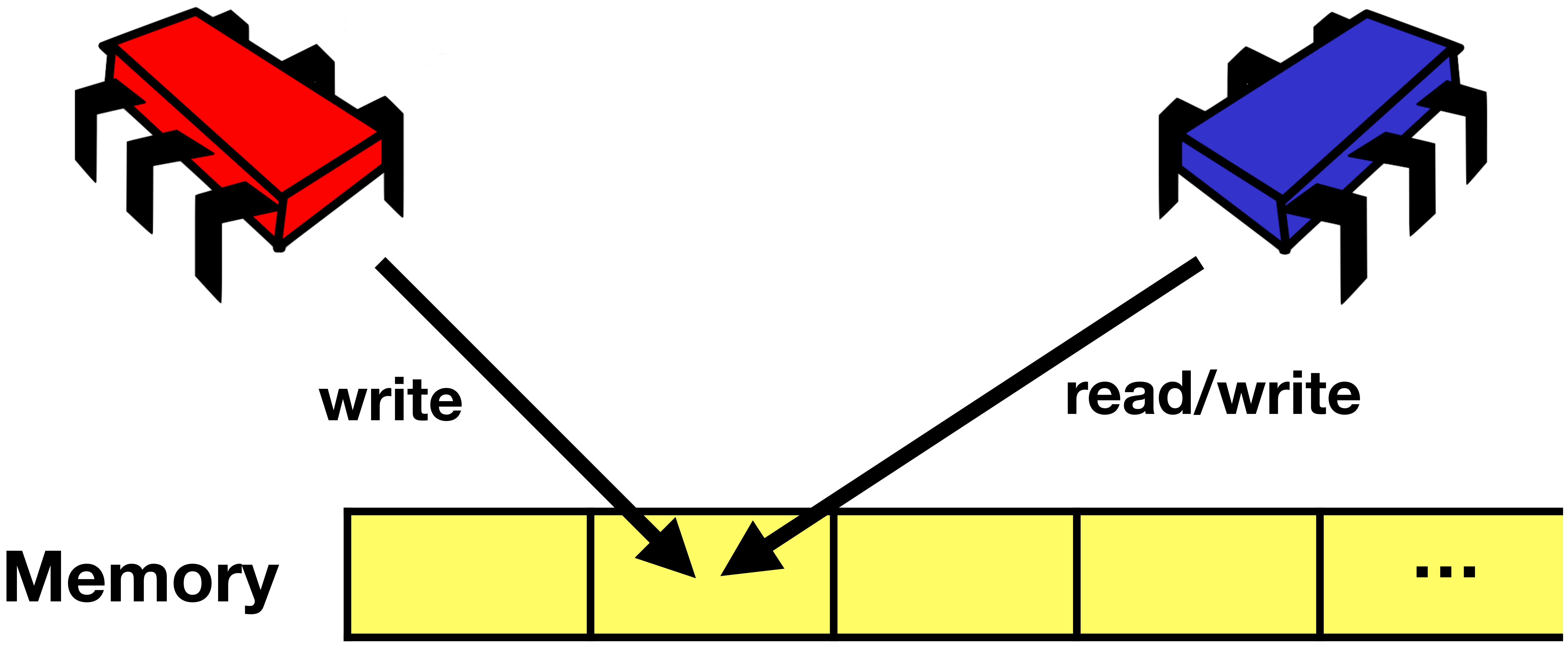
- Language-independent DRF formulation
- DRF-preservation and key proof structures
- Supporting x86-TSO and confined benign-races in CASCompCert

Outline of this Talk

- Language-independent DRF formulation
- DRF-preservation and key proof structures
- Supporting x86-TSO and confined benign-races in CASCompCert

Language-Independent DRF

Data-race: read-write / write-write conflicts



Language-Independent DRF

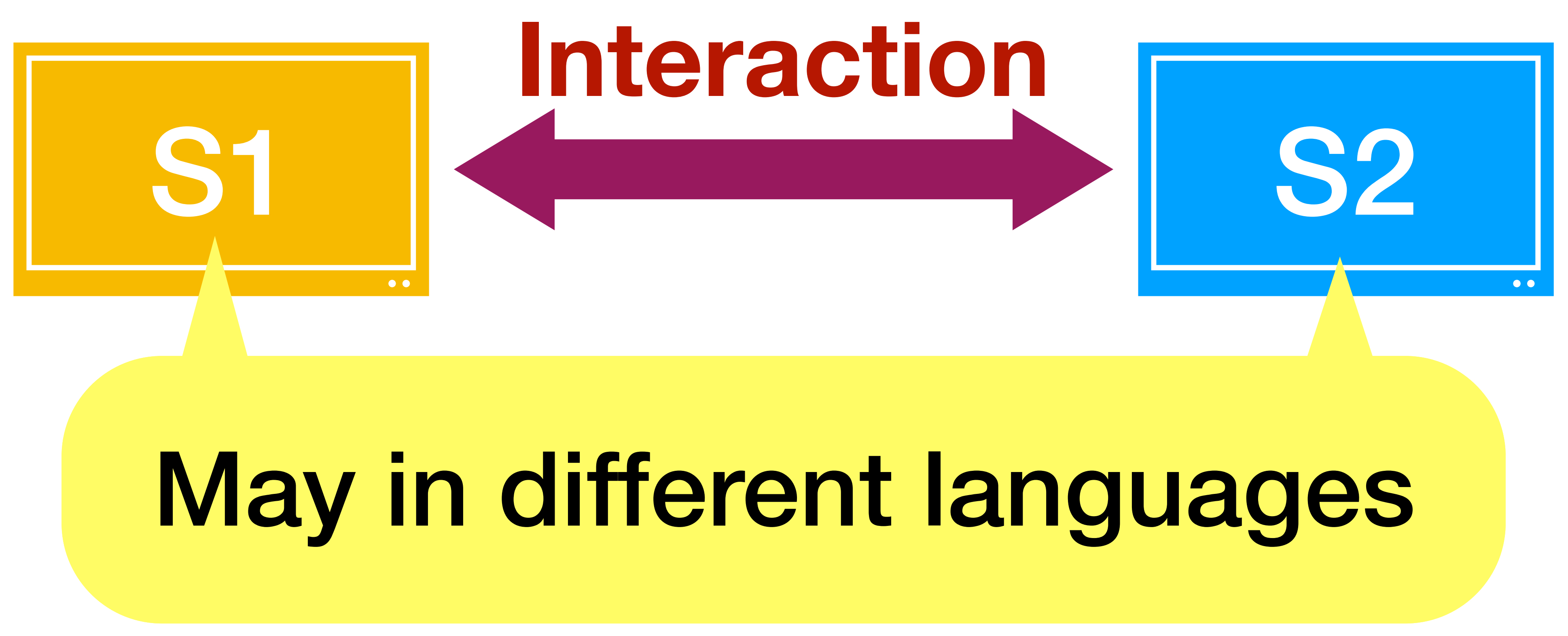
Data-race: read-write / write-write conflicts

Language-Independent DRF

Data-race: read-write / write-write conflicts

Why language-independent?

To support cross-language interaction



Language-Independent DRF

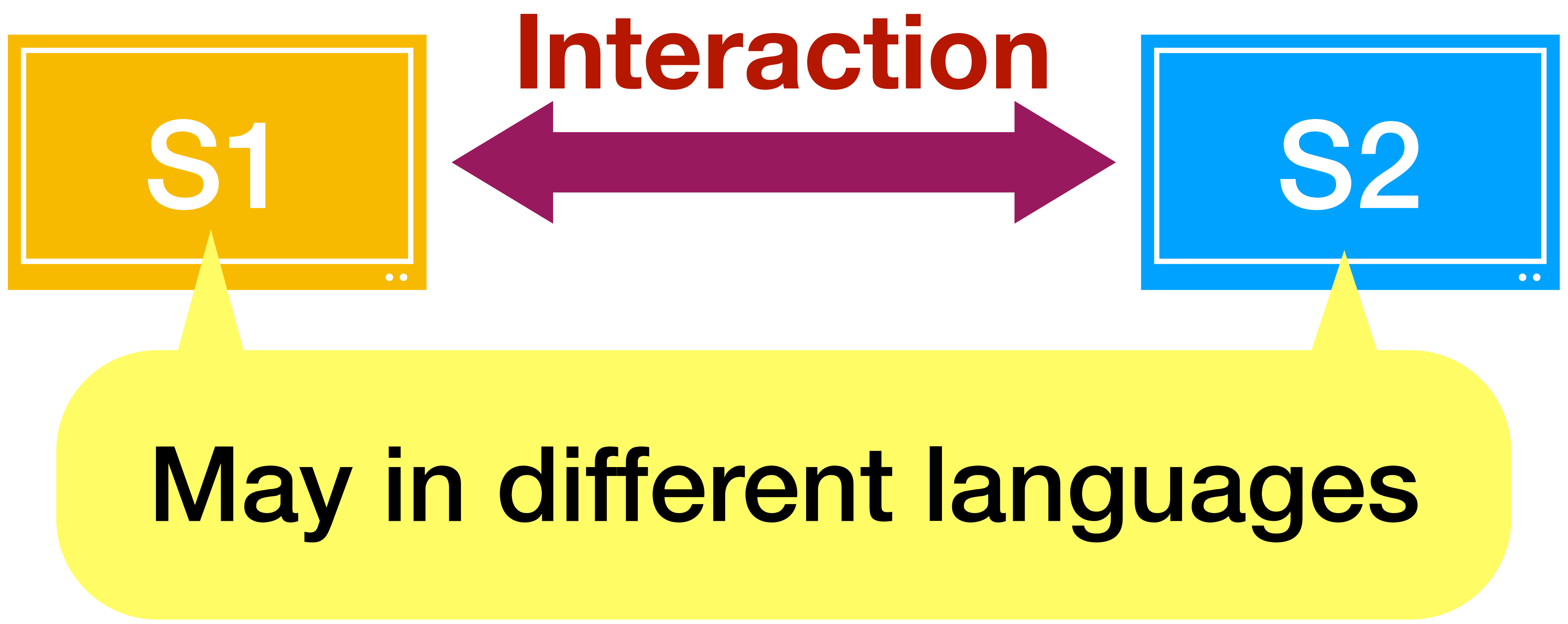
Data-race: read-write / write-write conflicts

Why language-independent?

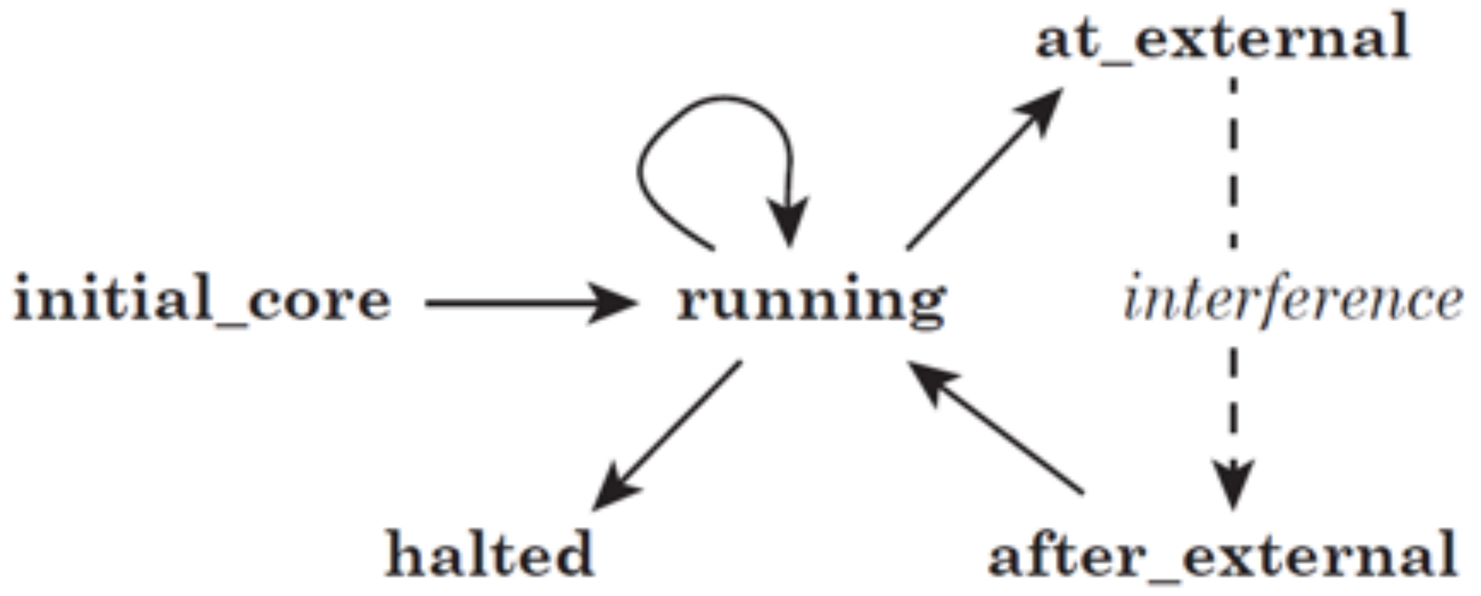
To support cross-language interaction

abstract away lang. details

e.g. interaction semantics



[Stewart et al. POPL'15]



$$\mathsf{Semantics}\ (G\ C\ M : \mathsf{Type}) : \mathsf{Type} \triangleq \left\{ \begin{array}{lcl} \mathsf{initial\_core} & : & G \to \mathcal{V} \to \mathsf{list}\ \mathcal{V} \to \mathsf{option}\ C \\ \mathsf{at\_external} & : & C \to \mathsf{option}\ (\mathcal{F} \times \mathsf{list}\ \mathcal{V}) \\ \mathsf{after\_external} & : & \mathsf{option}\ \mathcal{V} \to C \to \mathsf{option}\ C \\ \mathsf{halted} & : & C \to \mathsf{option}\ \mathcal{V} \\ \mathsf{corestep} & : & G \to C \to M \to C \to M \to \mathsf{Prop} \end{array} \right.$$

Language-Independent DRF

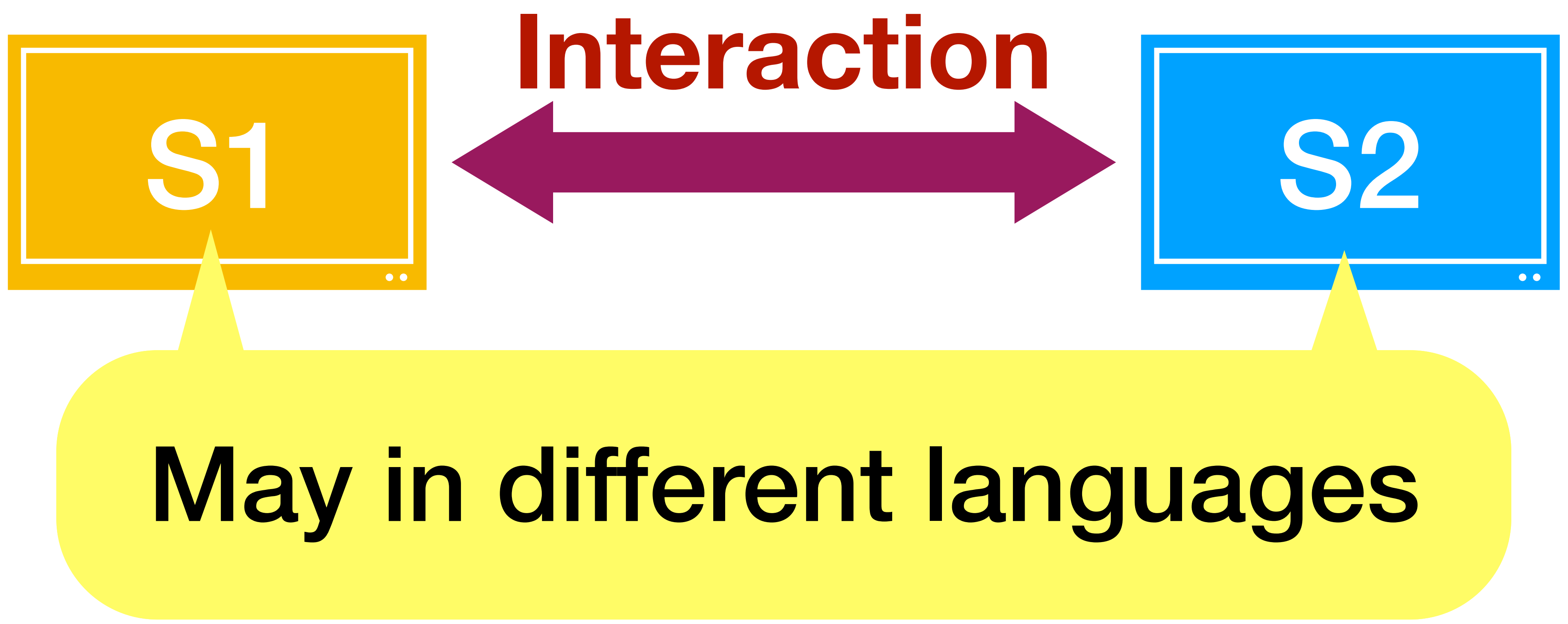
Data-race: read-write / write-write conflicts

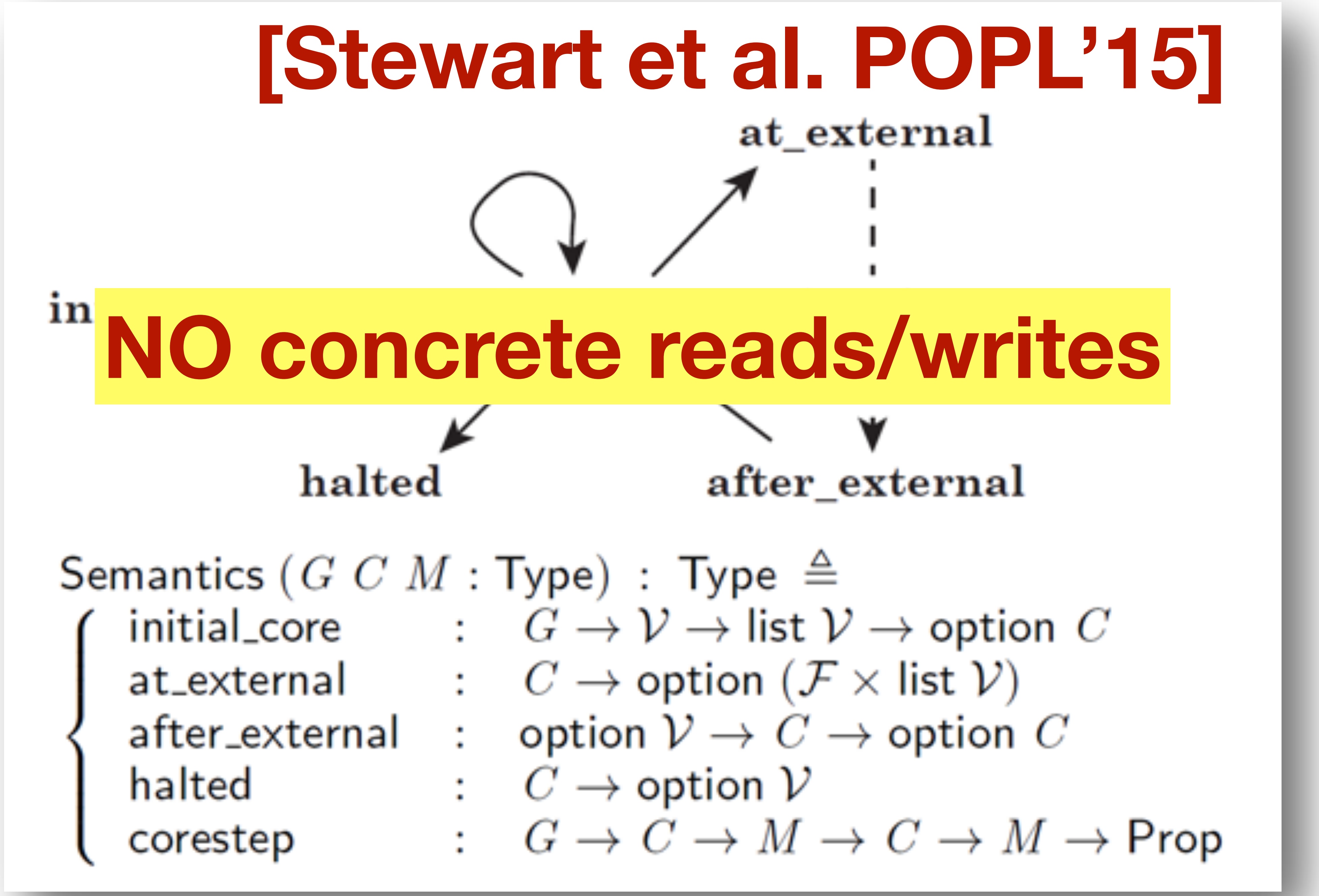
Why language-independent?

To support cross-language interaction

abstract away lang. details

e.g. interaction semantics





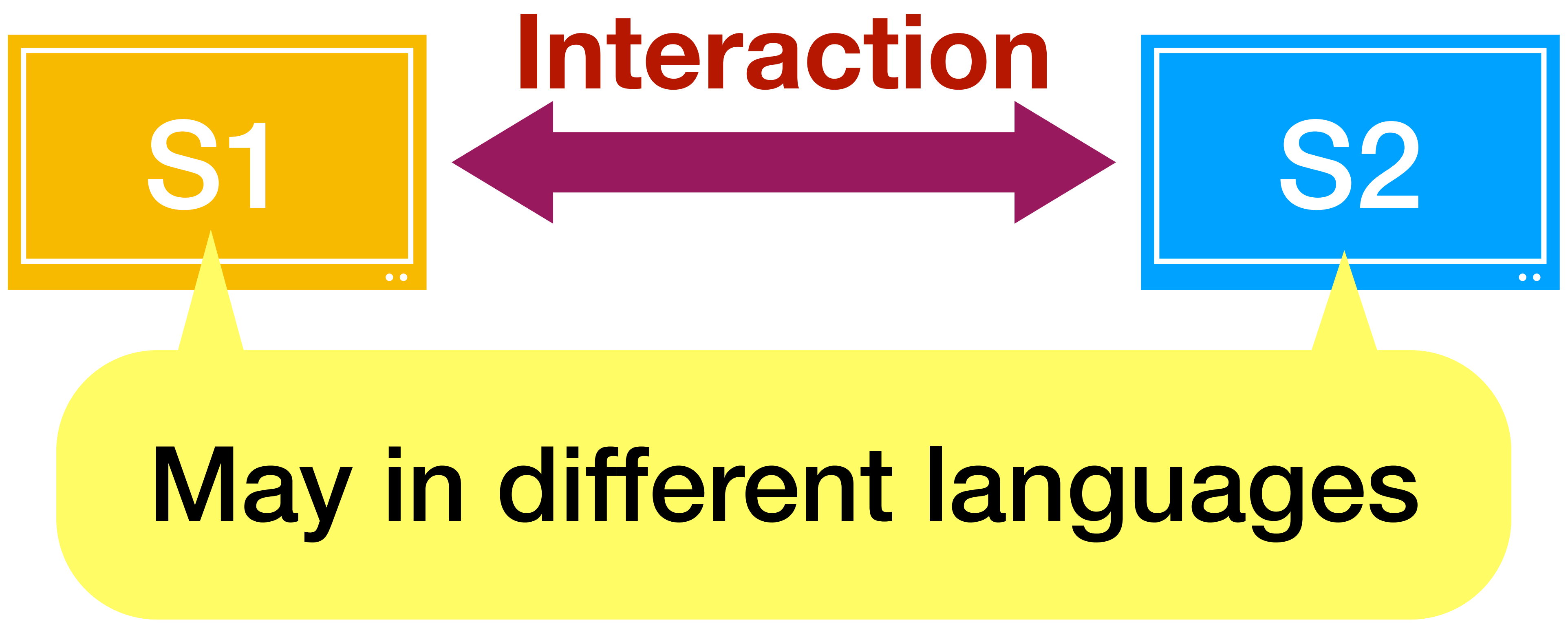
Language-Independent DRF

Data-race: read-write / write-write conflicts

Why language-independent?

How to formulate DRF

if we do not even know the concrete reads/writes?



$$
\text{Semantics}\ (G\ C\ M : \text{Type})\ :\ \text{Type} \triangleq
\begin{cases}
\text{initial\_core} & : \quad G \to \mathcal{V} \to \text{list}\ \mathcal{V} \to \text{option}\ C \\
\text{at\_external} & : \quad C \to \text{option}\ (\mathcal{F} \times \text{list}\ \mathcal{V}) \\
\text{after\_external} & : \quad \text{option}\ \mathcal{V} \to C \to \text{option}\ C \\
\text{halted} & : \quad C \to \text{option}\ \mathcal{V} \\
\text{corestep} & : \quad G \to C \to M \to C \to M \to \text{Prop}
\end{cases}
$$

Solution: Abstract Footprints

DRF(S1 || S2)

Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)

- **footprints δ ::= (rs, ws)**

read-set

write-set

Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)

- **footprints δ ::= (rs, ws)**

read-set

write-set

- **well-defined language**

extensional characterization of footprints

Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)

- **footprints δ ::= (rs, ws)**

read-set

write-set

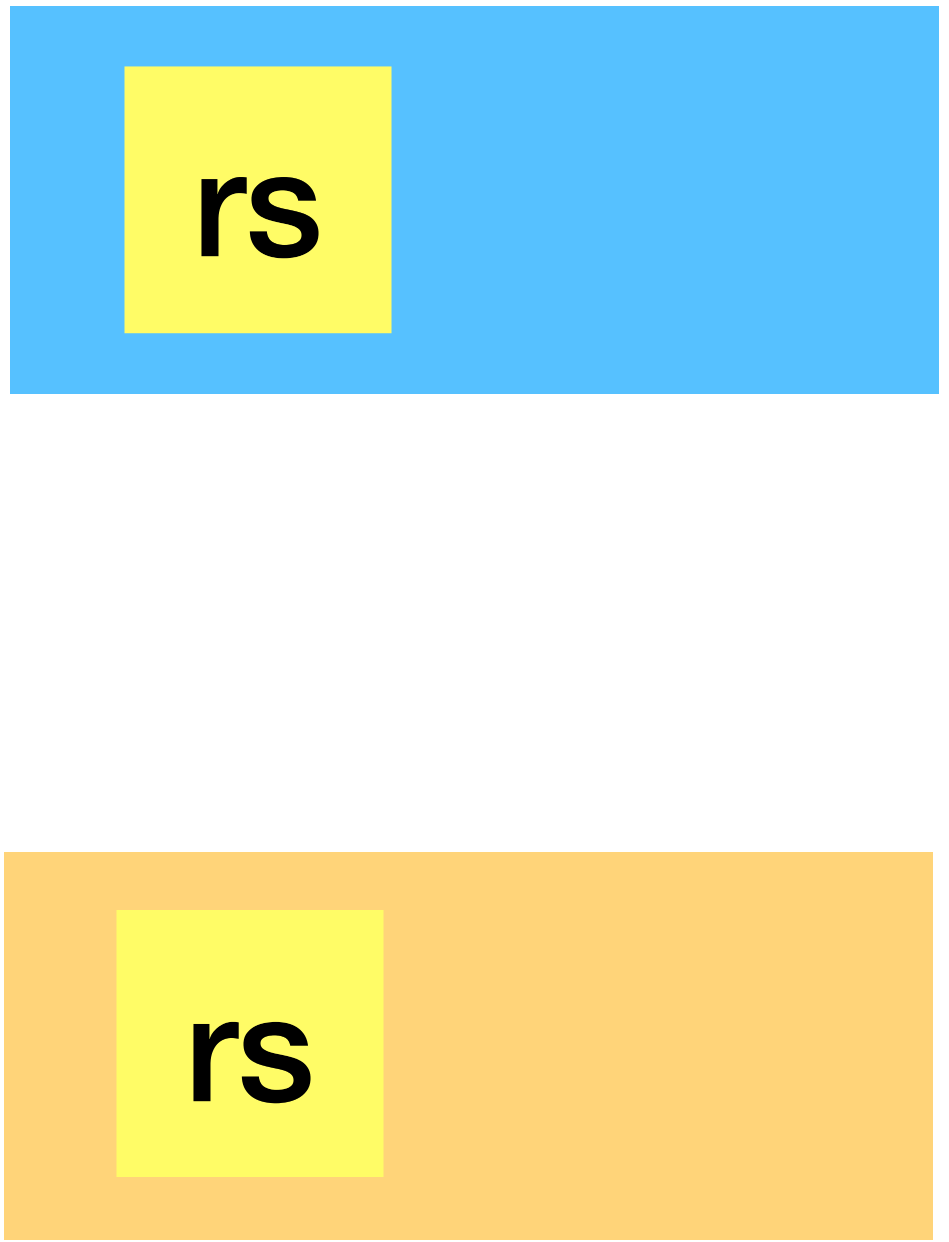
- **well-defined language**

extensional characterization of footprints

Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)



- footprints δ ::= (rs, ws)

read-set

write-set

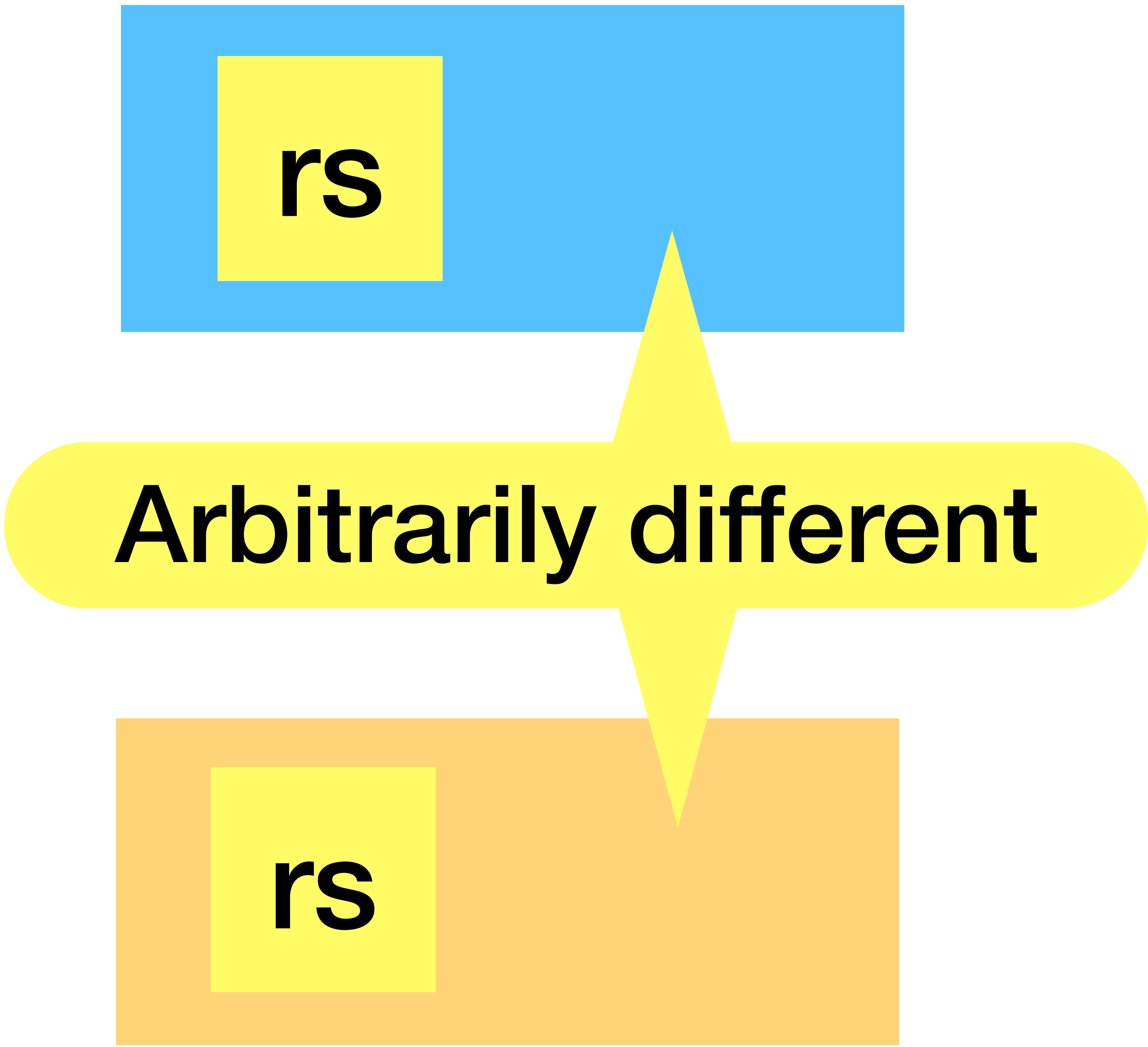
- **well-defined language**

extensional characterization of footprints

Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)



- footprints δ ::= (rs, ws)

read-set

write-set

- **well-defined language**

extensional characterization of footprints

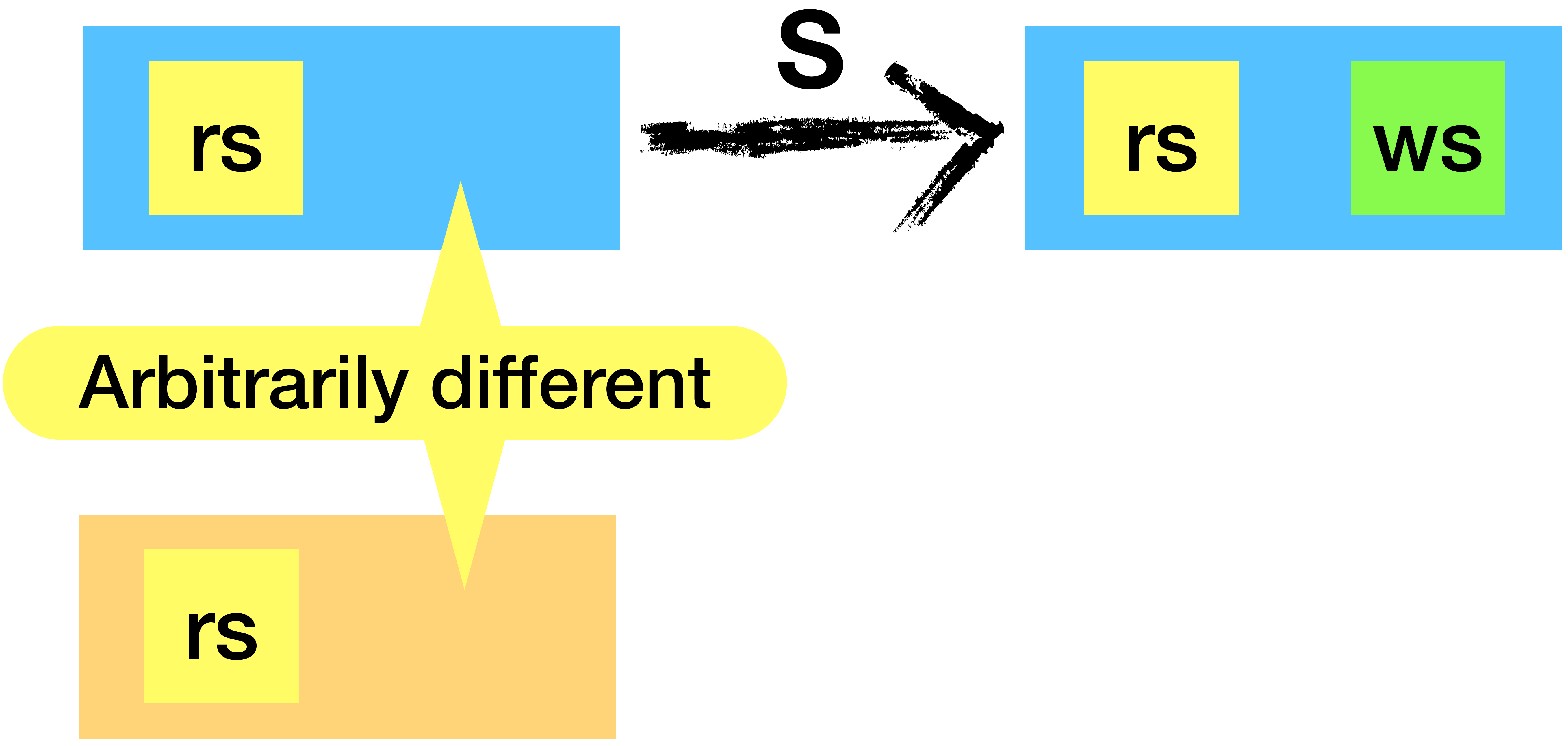
Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)

- footprints δ ::= (rs, ws)

read-set write-set



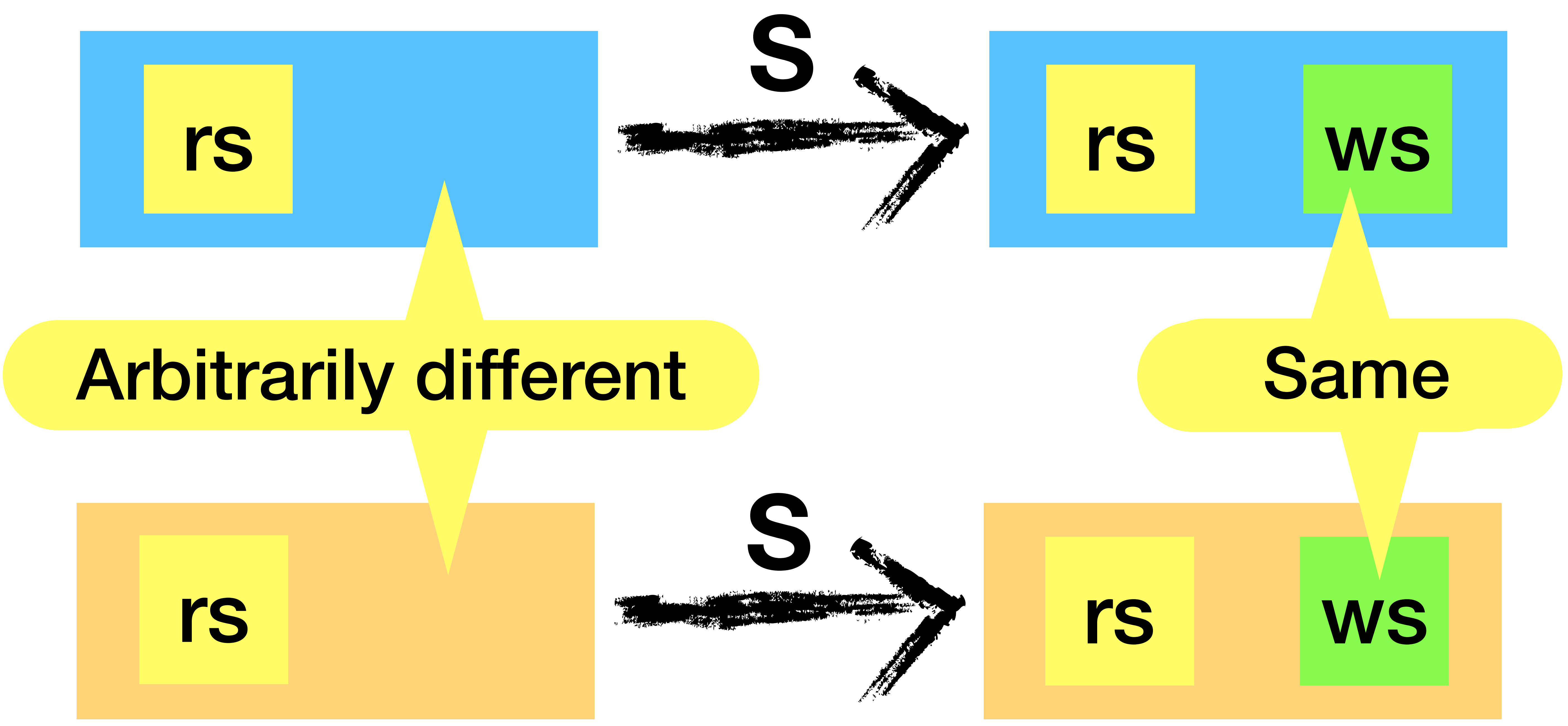
- **well-defined language**

extensional characterization of footprints

Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)



- footprints δ ::= (rs, ws)

read-set

write-set

- **well-defined language**

extensional characterization of footprints

Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)

- **footprints δ ::= (rs, ws)**

read-set

write-set

- **well-defined language**

extensional characterization of footprints

Solution: Abstract Footprints

Defined in terms of footprint disjointness

DRF(S1 || S2)

- **footprints δ ::= (rs, ws)**

read-set

write-set

- **well-defined language**

extensional characterization of footprints

Definition 1 (Well-Defined Languages). $\mathrm{wd}(tl)$ iff, for any execution step $F \vdash (\kappa, \sigma) \overset{\iota}{\underset{\delta}{\longmapsto}} (\kappa', \sigma')$ in this language, all of the following hold (some auxiliary definitions are in Fig. 6):

(1) $\mathtt{forward}(\sigma, \sigma')$;

(2) $\mathtt{LEffect}(\sigma, \sigma', \delta, F)$;

(3) For any σ_1, if $\mathtt{LEqPre}(\sigma, \sigma_1, \delta, F)$, then there exists σ_1' such that $F \vdash (\kappa, \sigma_1) \overset{\iota}{\underset{\delta}{\longmapsto}} (\kappa', \sigma_1')$ and $\mathtt{LEqPost}(\sigma', \sigma_1', \delta, F)$.

(4) Let $\delta_0 = \bigcup\{\delta \mid \exists \kappa', \sigma'.\, F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}} (\kappa', \sigma')\}$. For any σ_1, if $\mathtt{LEqPre}(\sigma, \sigma_1, \delta_0, F)$, then for any $\kappa_1', \sigma_1', \iota_1, \delta_1$, $F \vdash (\kappa, \sigma_1) \overset{\iota_1}{\underset{\delta_1}{\longmapsto}} (\kappa_1', \sigma_1') \implies \exists \sigma'.\, F \vdash (\kappa, \sigma) \overset{\iota_1}{\underset{\delta_1}{\longmapsto}} (\kappa_1', \sigma')$.

Outline of this Talk

- Language-independent DRF formulation
- DRF-preservation and key proof structures
- Supporting x86-TSO and confined benign-races in CASCompCert

Outline of this Talk

- Language-independent DRF formulation
- DRF-preservation and key proof structures
- Supporting x86-TSO and confined benign-races in CASCompCert

Compositional CompCert's Argument…

$$T1 \parallel T2 \overset{?}{\subseteq} S1 \parallel S2$$

DRF(S1 || S2)

T1 = Comp(S1)

T2 = Comp(S2)

Compositional CompCert's Argument…

$$T1 \parallel T2 \overset{?}{\subseteq} S1 \parallel S2$$

$$T1 \mid T2 \subseteq S1 \mid S2$$

Non-preemptive

DRF(S1 || S2)

T1 = Comp(S1)   T2 = Comp(S2)

Compositional CompCert's Argument…

$$T1 \parallel T2 \overset{?}{\subseteq} S1 \parallel S2$$

?

$$T1 \mid T2 \subseteq S1 \mid S2$$

Non-preemptive

DRF(S1 || S2)

T1 = Comp(S1) **T2 = Comp(S2)**

Our Ideas

$$T1 \parallel T2 \overset{?}{\subseteq} S1 \parallel S2$$

?

$$T1 \mid T2 \subseteq S1 \mid S2$$

Non-preemptive

DRF(S1 || S2)

T1 = Comp(S1)

T2 = Comp(S2)

Our Ideas

$$\begin{array}{ccc} T1 \parallel T2 & \overset{?}{\subseteq} & S1 \parallel S2 \\ \rotatebox{-90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\ T1 \mid T2 & \subseteq & S1 \mid S2 \end{array}$$

Non-preemptive

DRF(S1 || S2)

T1 = Comp(S1)

T2 = Comp(S2)

Our Ideas

$$
\begin{tikzcd}
T1 \parallel T2 \arrow[r, phantom, "\overset{?}{\subseteq}"] & S1 \parallel S2 \arrow[d, phantom, "\rotatebox{90}{$\subseteq$}"] & \text{Trivial} \\
T1 \mid T2 \arrow[u, phantom, "\rotatebox{90}{$\subseteq$}"] \arrow[r, phantom, "\subseteq"] \arrow[ru, bend right] & S1 \mid S2 &
\end{tikzcd}
$$

Non-preemptive

DRF(S1 || S2)

T1 = Comp(S1)   T2 = Comp(S2)

Our Ideas

$$\begin{array}{ccccc} & & ? & & \\ & \mathrm{T1} \parallel \mathrm{T2} & \subseteq & \mathrm{S1} \parallel \mathrm{S2} & \\ ?\ \rotatebox{-90}{$\subseteq$} & & & & \rotatebox{90}{$\subseteq$}\ \text{Trivial} \\ & \mathrm{T1} \mid \mathrm{T2} & \subseteq & \mathrm{S1} \mid \mathrm{S2} & \end{array}$$

Non-preemptive

DRF(S1 || S2)

T1 = Comp(S1)

T2 = Comp(S2)

Our Ideas

$$
\begin{tikzcd}
T1 \parallel T2 \arrow[r, phantom, "\overset{?}{\subseteq}"] \arrow[d, phantom, "\rotatebox{-90}{$\subseteq$}"] & S1 \parallel S2 \\
T1 \mid T2 \arrow[r, phantom, "\subseteq"] & S1 \mid S2 \arrow[u, phantom, "\rotatebox{90}{$\subseteq$}"]
\end{tikzcd}
$$

DRF(T1 || T2)

Trivial

Non-preemptive

DRF(S1 || S2)

T1 = Comp(S1)

T2 = Comp(S2)

Our Ideas

?

DRF(T1 || T2)

$$\begin{array}{ccc} & ? & \\ T1 \parallel T2 & \subseteq & S1 \parallel S2 \\ \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\supseteq$} \quad \text{Trivial} \\ T1 \mid T2 & \subseteq & S1 \mid S2 \end{array}$$

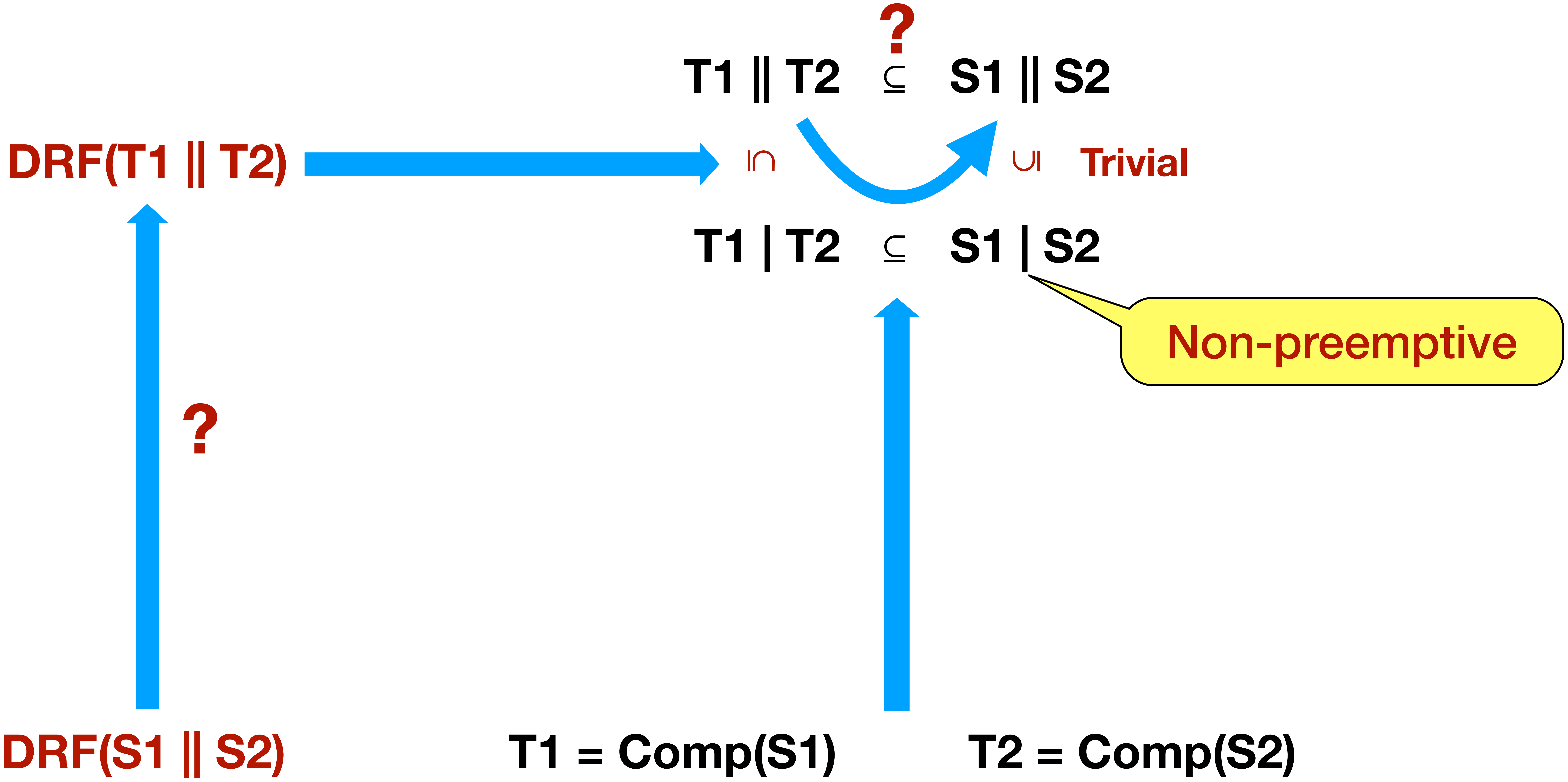
Non-preemptive

DRF(S1 || S2)

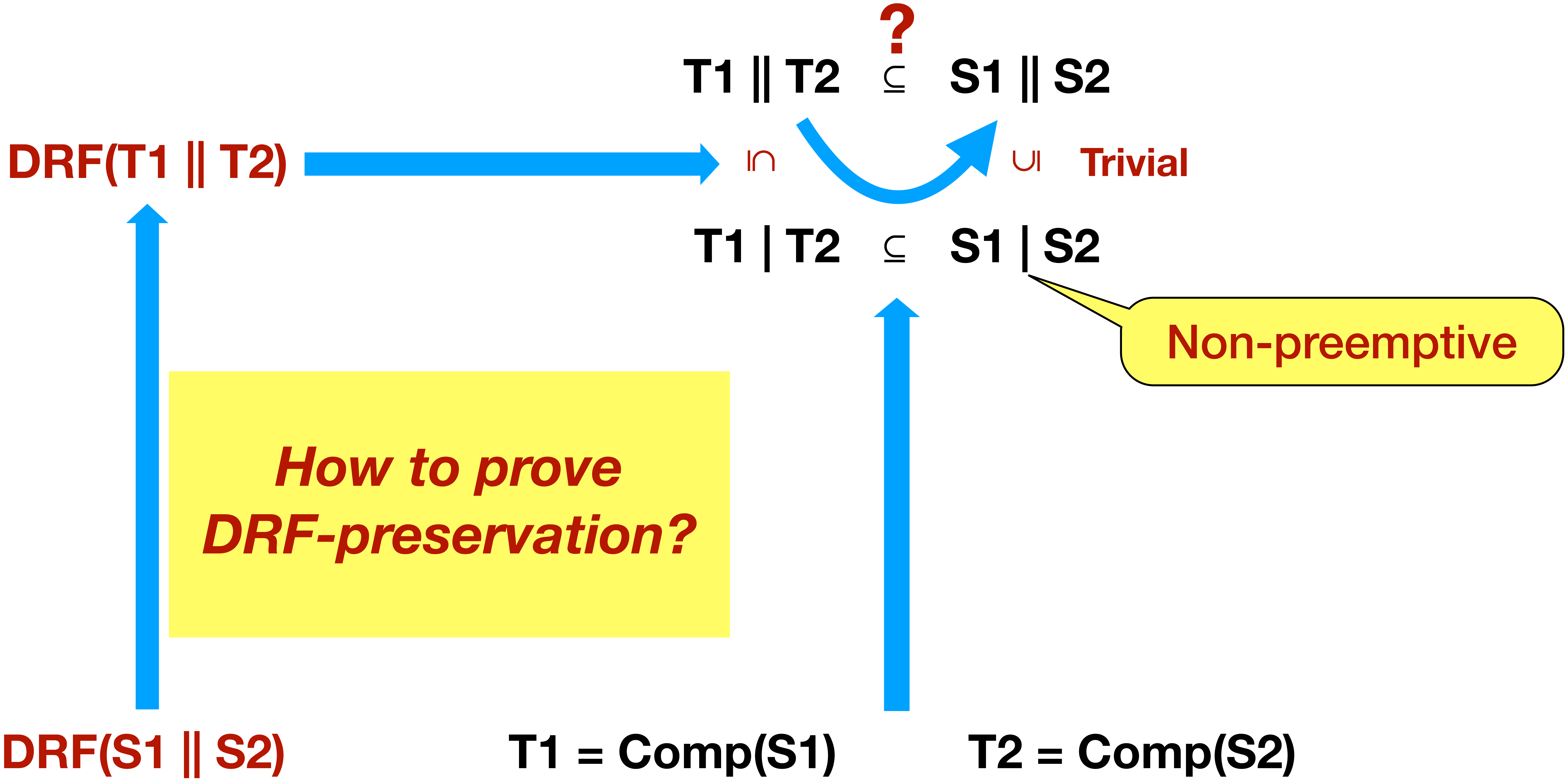
T1 = Comp(S1)

T2 = Comp(S2)

Our Ideas


?
T1 || T2 ⊆ S1 || S2
DRF(T1 || T2)
Trivial
T1 | T2 ⊆ S1 | S2
Non-preemptive
?
DRF(S1 || S2)
T1 = Comp(S1)
T2 = Comp(S2)

Our Ideas


?
T1 || T2 ⊆ S1 || S2
DRF(T1 || T2)
∩|
∪|
Trivial
T1 | T2 ⊆ S1 | S2
Non-preemptive
How to prove
DRF-preservation?
DRF(S1 || S2)
T1 = Comp(S1)
T2 = Comp(S2)

Our Ideas

$$
\begin{tikzcd}
 & & \mathbf{T1 \parallel T2} & \subseteq & \mathbf{S1 \parallel S2} & \\
\color{red}{\mathbf{DRF(T1 \parallel T2)}} \arrow[rr] & & \rotatebox{-90}{$\subseteq$} \arrow[rr, bend right] & & \rotatebox{90}{$\subseteq$} & \color{red}{\textbf{Trivial}} \\
 & & \mathbf{T1 \mid T2} & \subseteq \arrow[d, leftarrow] & \mathbf{S1 \mid S2} & \\
\color{red}{\mathbf{DRF(S1 \parallel S2)}} \arrow[uu, "\color{red}{?}"'] & & \mathbf{T1 = Comp(S1)} & & \mathbf{T2 = Comp(S2)} &
\end{tikzcd}
$$

Our Ideas

$T1 \parallel T2 \quad \subseteq \quad S1 \parallel S2$

DRF(T1 || T2) $\quad \rotatebox{90}{$\subseteq$} \qquad \rotatebox{90}{$\supseteq$}$ Triv

$T1 \mid T2 \quad \subseteq \quad S1 \mid S2$

Footprint-preserving simulation

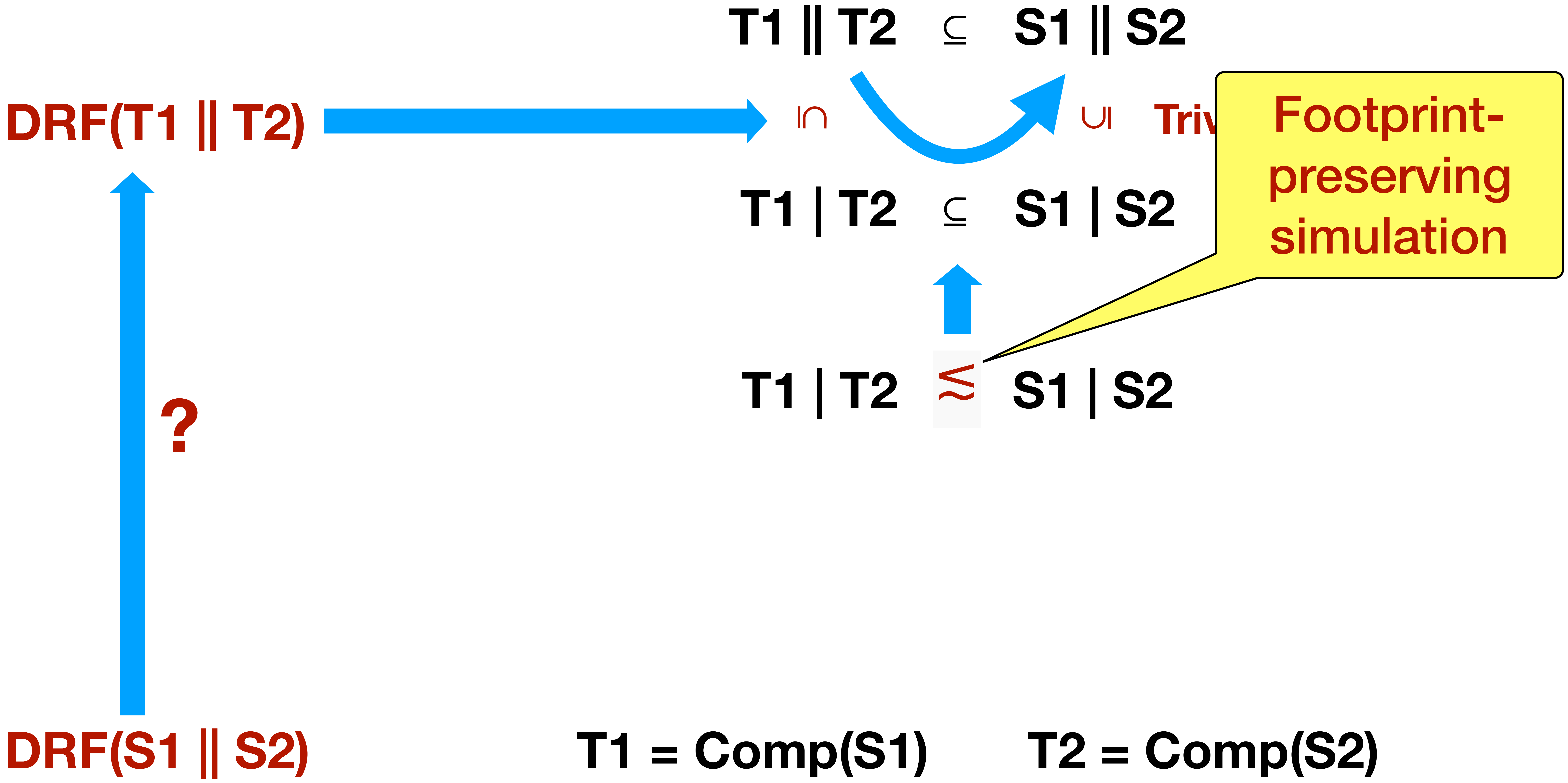
$T1 \mid T2 \quad \lesssim \quad S1 \mid S2$

?

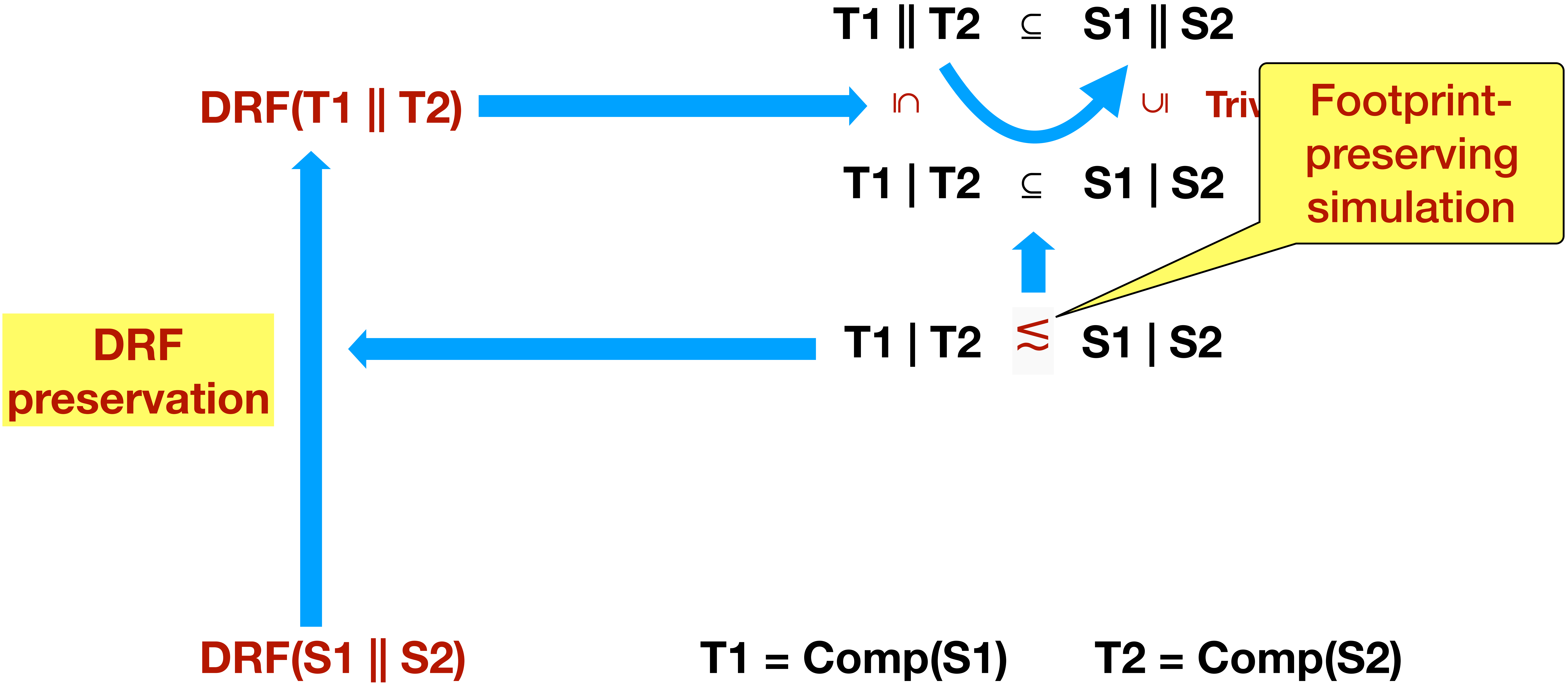
DRF(S1 || S2)

T1 = Comp(S1) T2 = Comp(S2)

Our Ideas

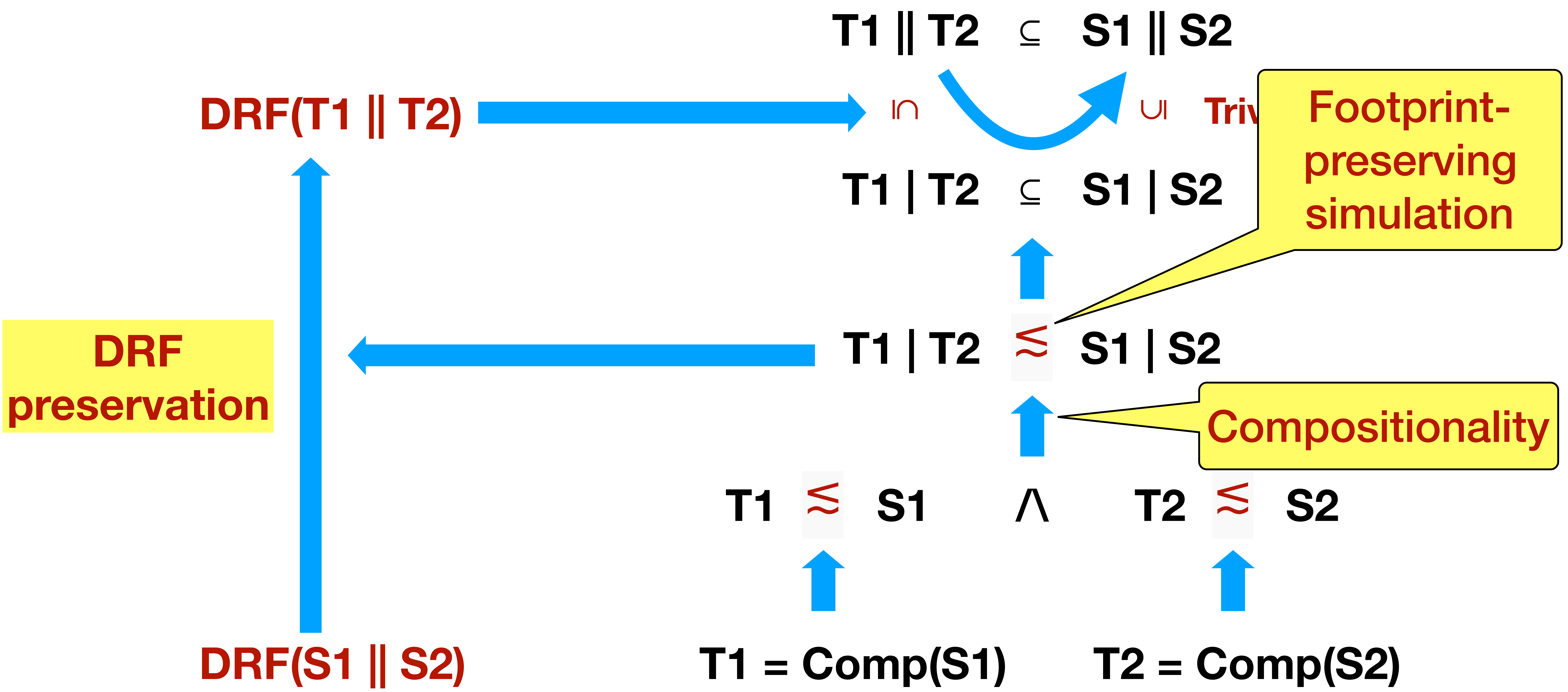

T1 || T2 ⊆ S1 || S2
DRF(T1 || T2)
⊆
⊆
Triv
Footprint-preserving simulation
T1 | T2 ⊆ S1 | S2
T1 | T2 ≲ S1 | S2
?
DRF(S1 || S2)
T1 = Comp(S1)
T2 = Comp(S2)

Our Ideas



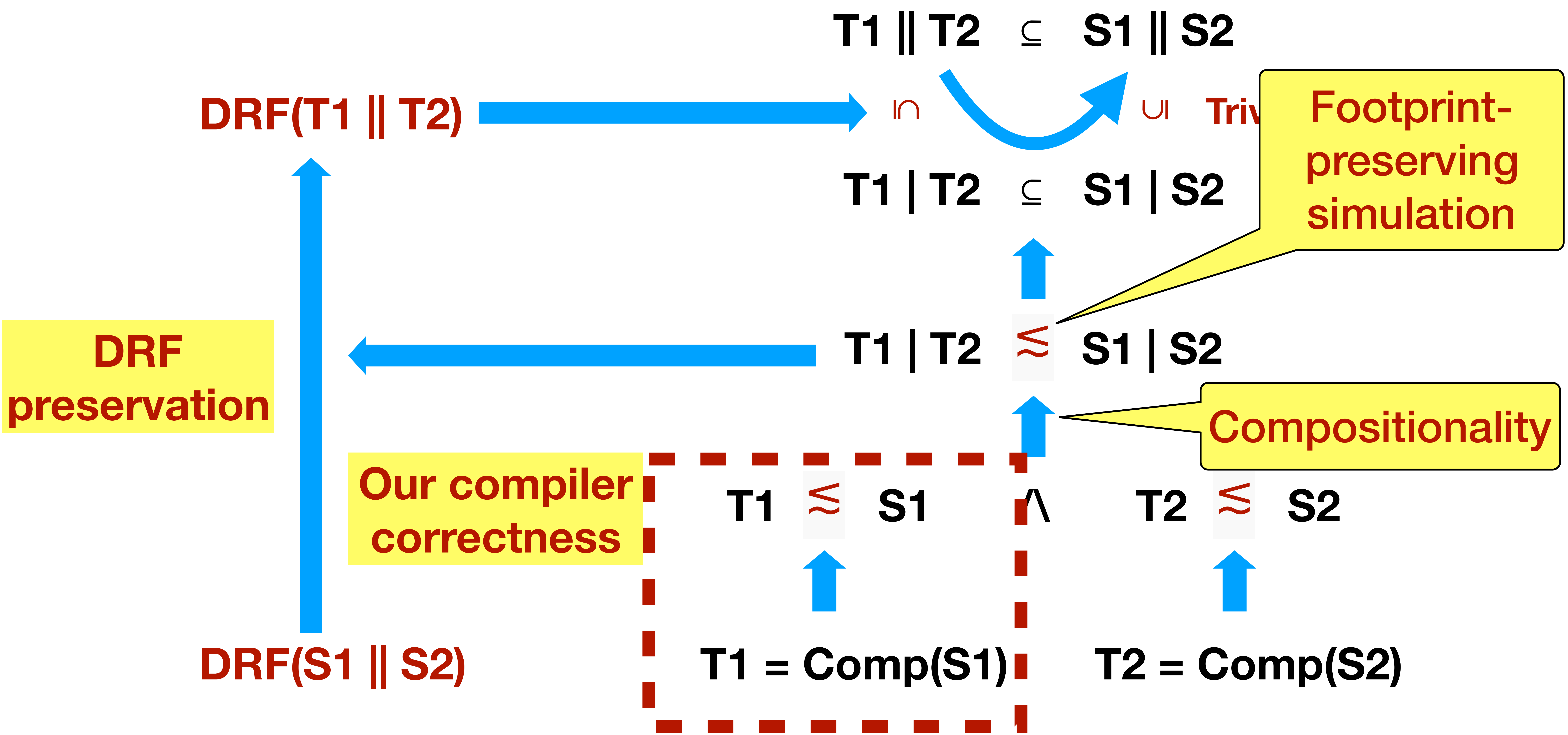
T1 || T2 ⊆ S1 || S2
DRF(T1 || T2)
⊆
⊆
Triv
T1 | T2 ⊆ S1 | S2
Footprint-preserving simulation
DRF preservation
T1 | T2 ≲ S1 | S2
DRF(S1 || S2)
T1 = Comp(S1)
T2 = Comp(S2)

Our Ideas



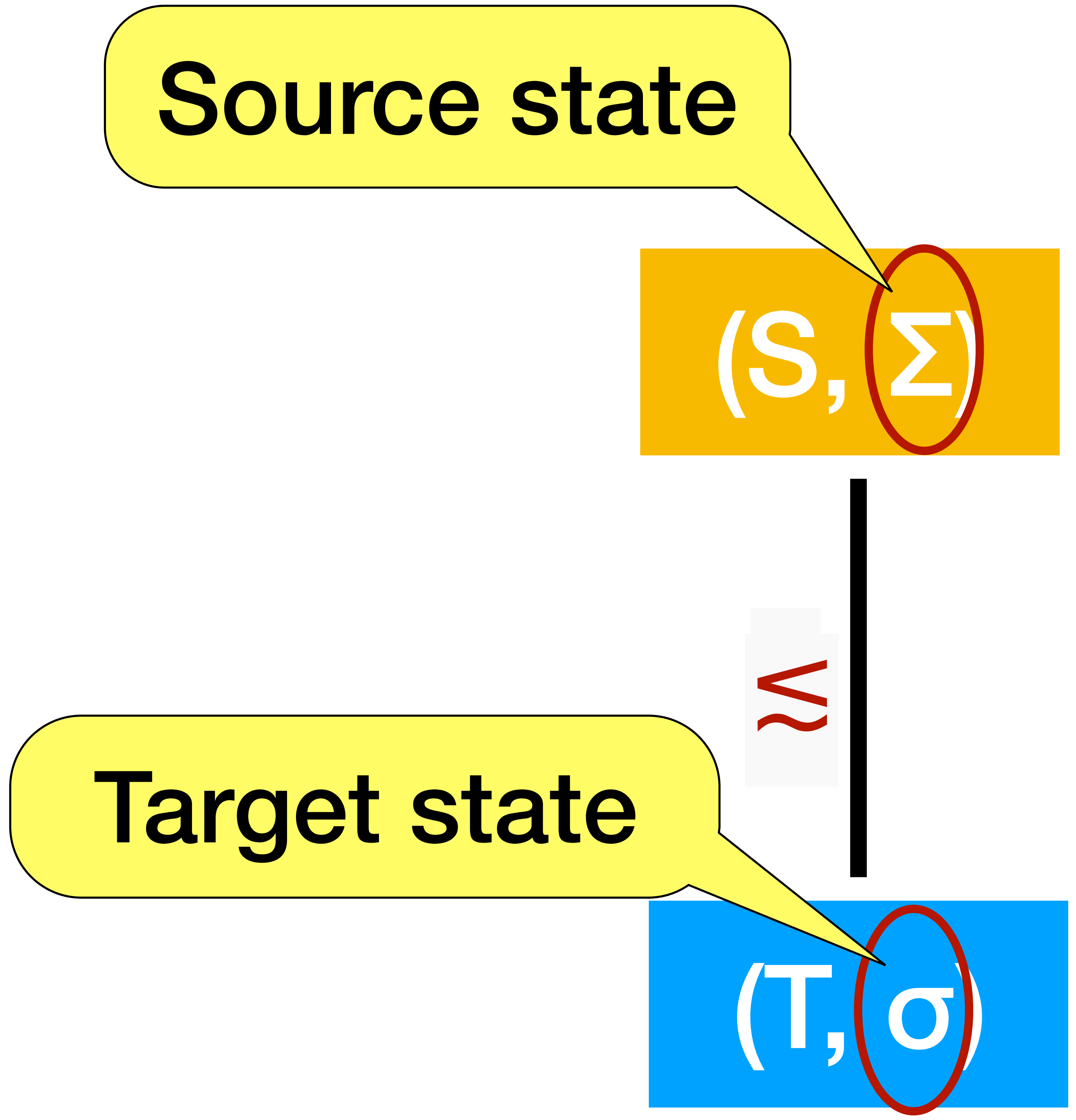
T1 || T2 ⊆ S1 || S2
DRF(T1 || T2)
⊆
⊆
Triv
Footprint-preserving simulation
T1 | T2 ⊆ S1 | S2
DRF preservation
T1 | T2 ≲ S1 | S2
Compositionality
T1 ≲ S1 ∧ T2 ≲ S2
DRF(S1 || S2)
T1 = Comp(S1)
T2 = Comp(S2)

Our Ideas



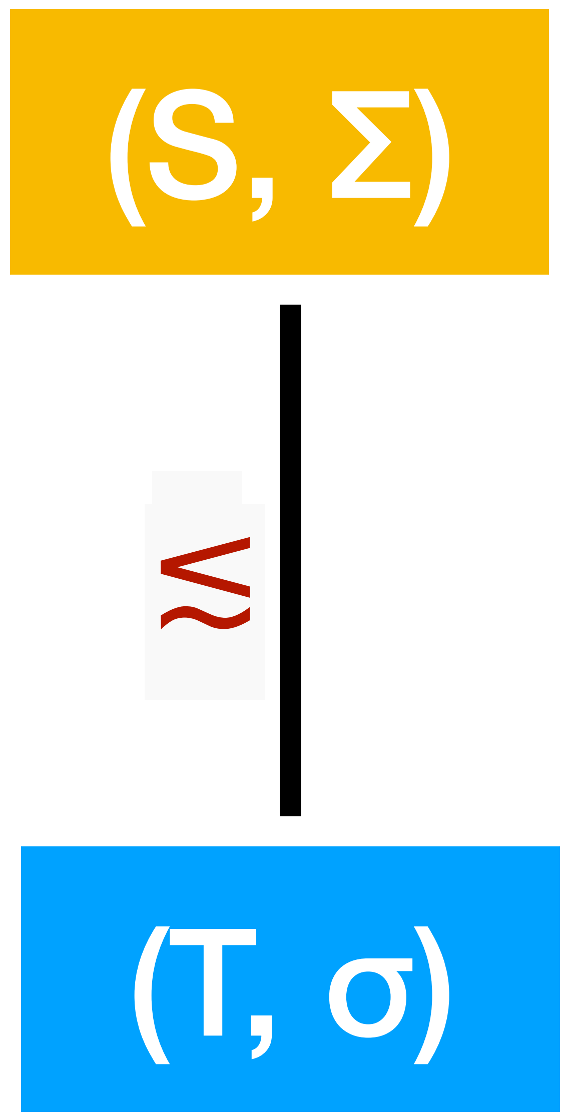
T1 || T2 ⊆ S1 || S2
DRF(T1 || T2)
⊆
⊆
Triv
Footprint-preserving simulation
T1 | T2 ⊆ S1 | S2
DRF preservation
T1 | T2 ≲ S1 | S2
Compositionality
Our compiler correctness
T1 ≲ S1
/\
T2 ≲ S2
DRF(S1 || S2)
T1 = Comp(S1)
T2 = Comp(S2)

Solution: Footprint-Preserving Simulation



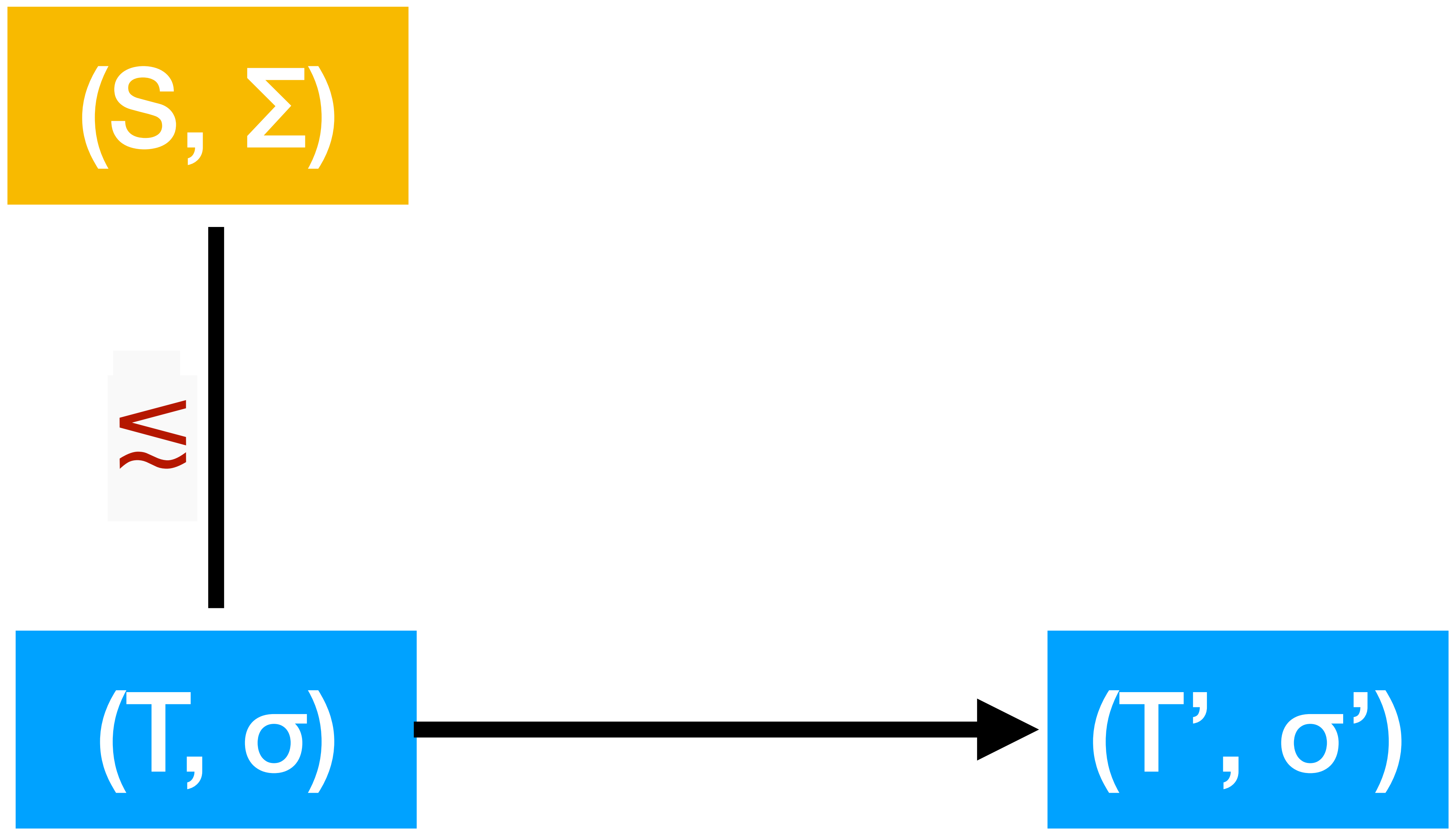
Target has smaller footprints, so cannot introduce more races

Solution: Footprint-Preserving Simulation



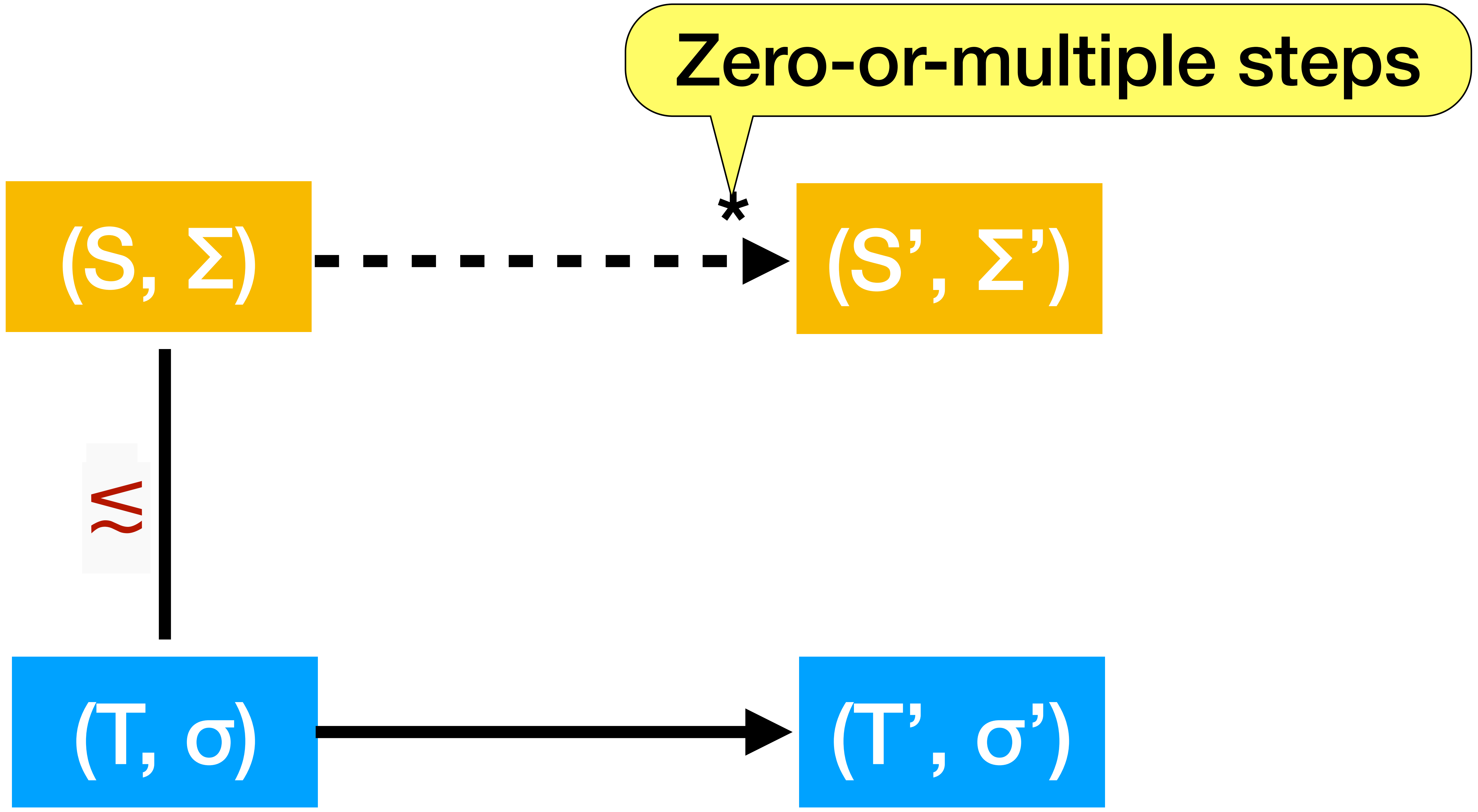
Target has smaller footprints, so cannot introduce more races

Solution: Footprint-Preserving Simulation



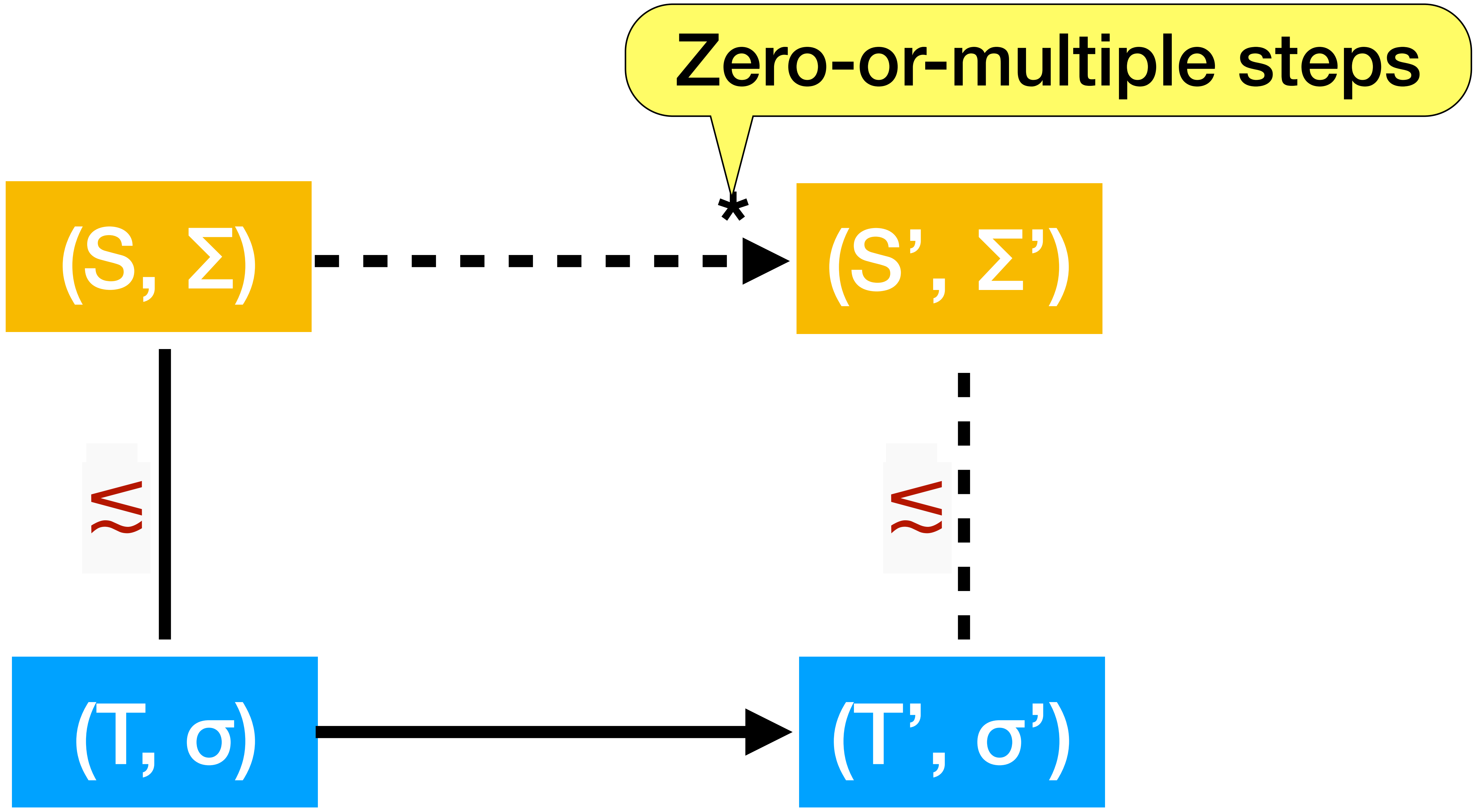
Target has smaller footprints, so cannot introduce more races

Solution: Footprint-Preserving Simulation



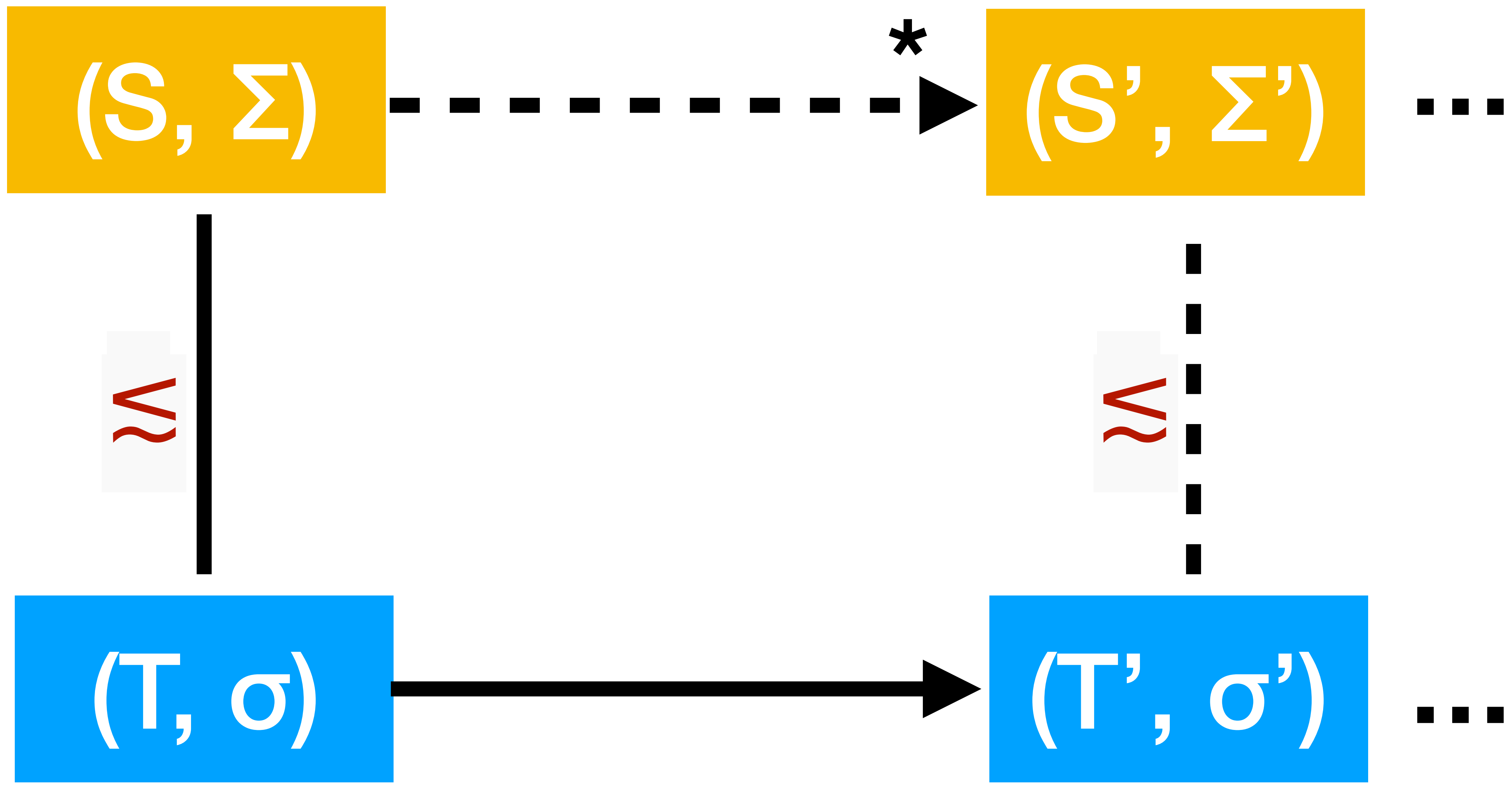
Target has smaller footprints, so cannot introduce more races

Solution: Footprint-Preserving Simulation



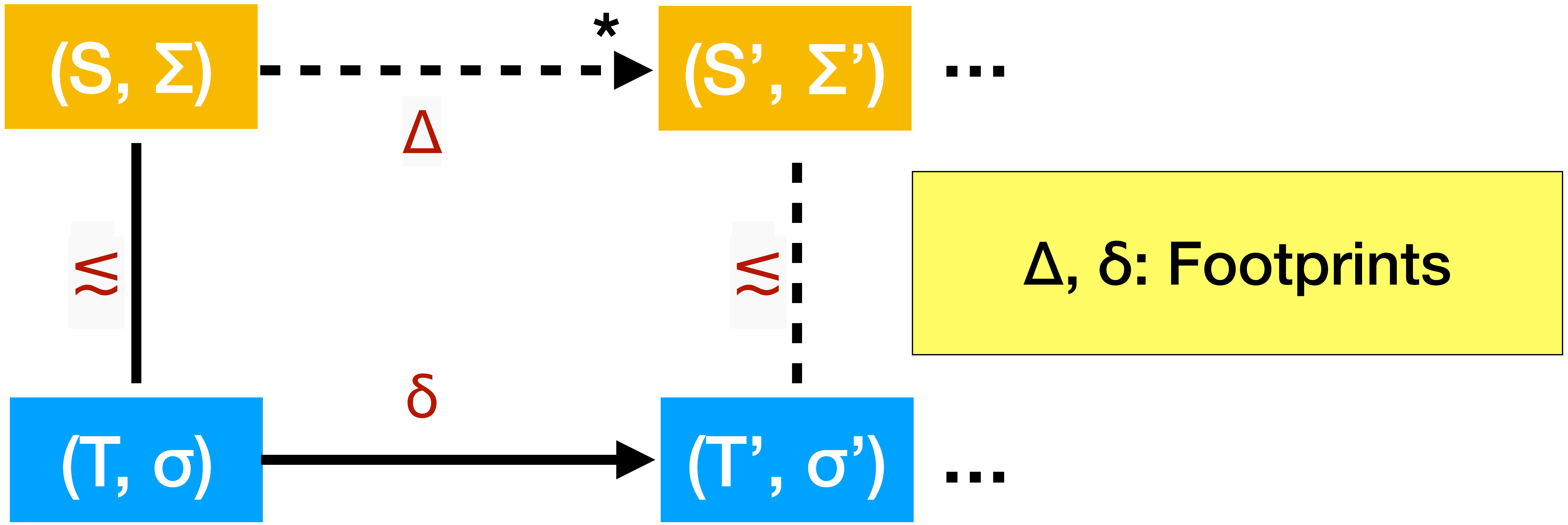
Target has smaller footprints, so cannot introduce more races

Solution: Footprint-Preserving Simulation



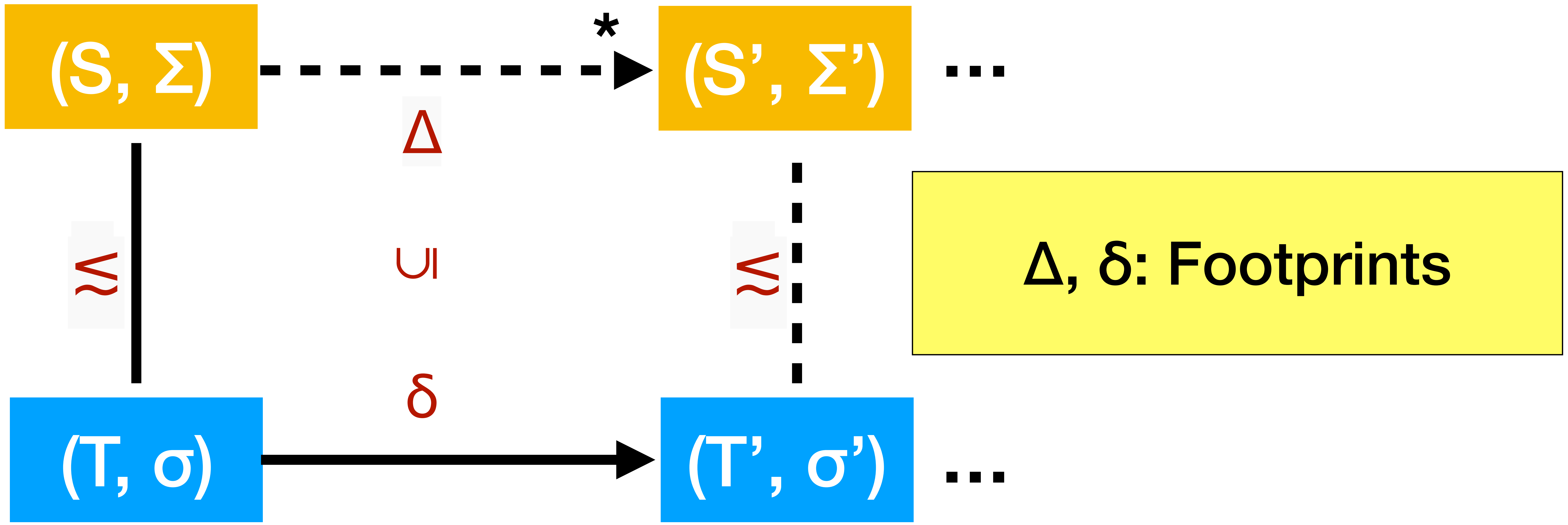
Target has smaller footprints, so cannot introduce more races

Solution: Footprint-Preserving Simulation



Target has smaller footprints, so cannot introduce more races

Solution: Footprint-Preserving Simulation



Target has smaller footprints, so cannot introduce more races

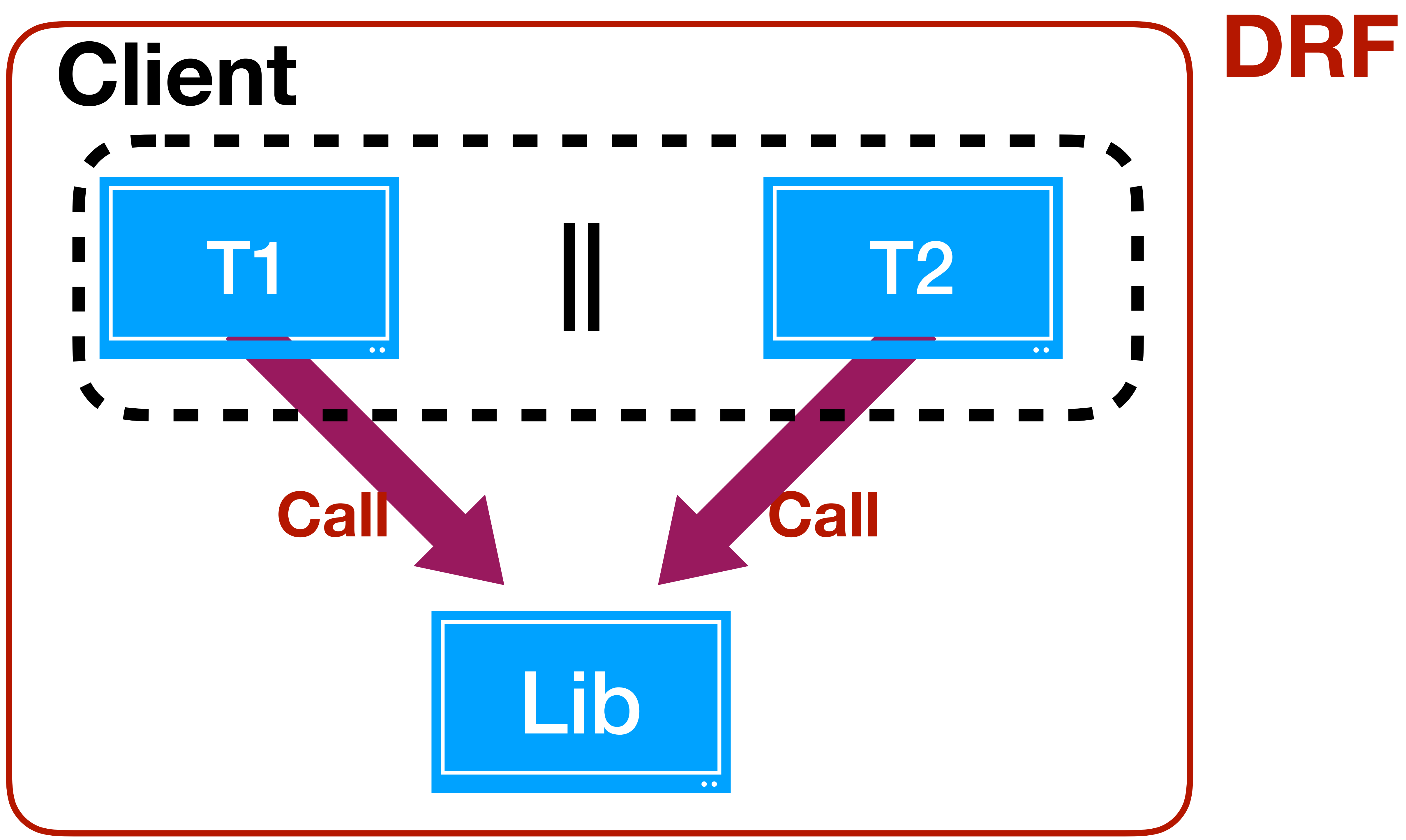
Outline of this Talk

- Language-independent DRF formulation
- DRF-preservation and key proof structures
- Supporting x86-TSO and confined benign-races in CASCompCert

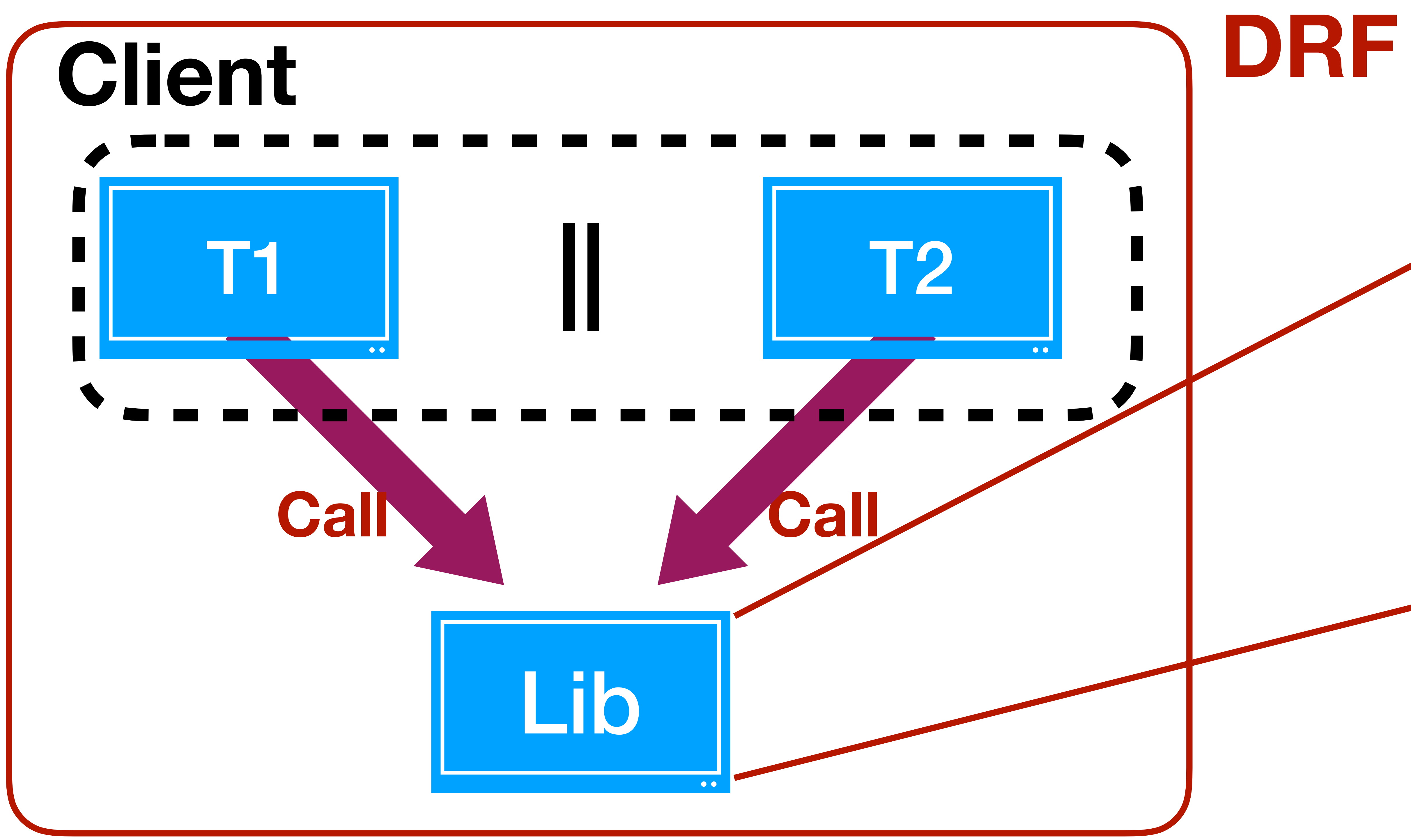
Outline of this Talk

- Language-independent DRF formulation
- DRF-preservation and key proof structures
- Supporting x86-TSO and confined benign-races in CASCompCert

DRF Imposes Strong Restriction on Libraries

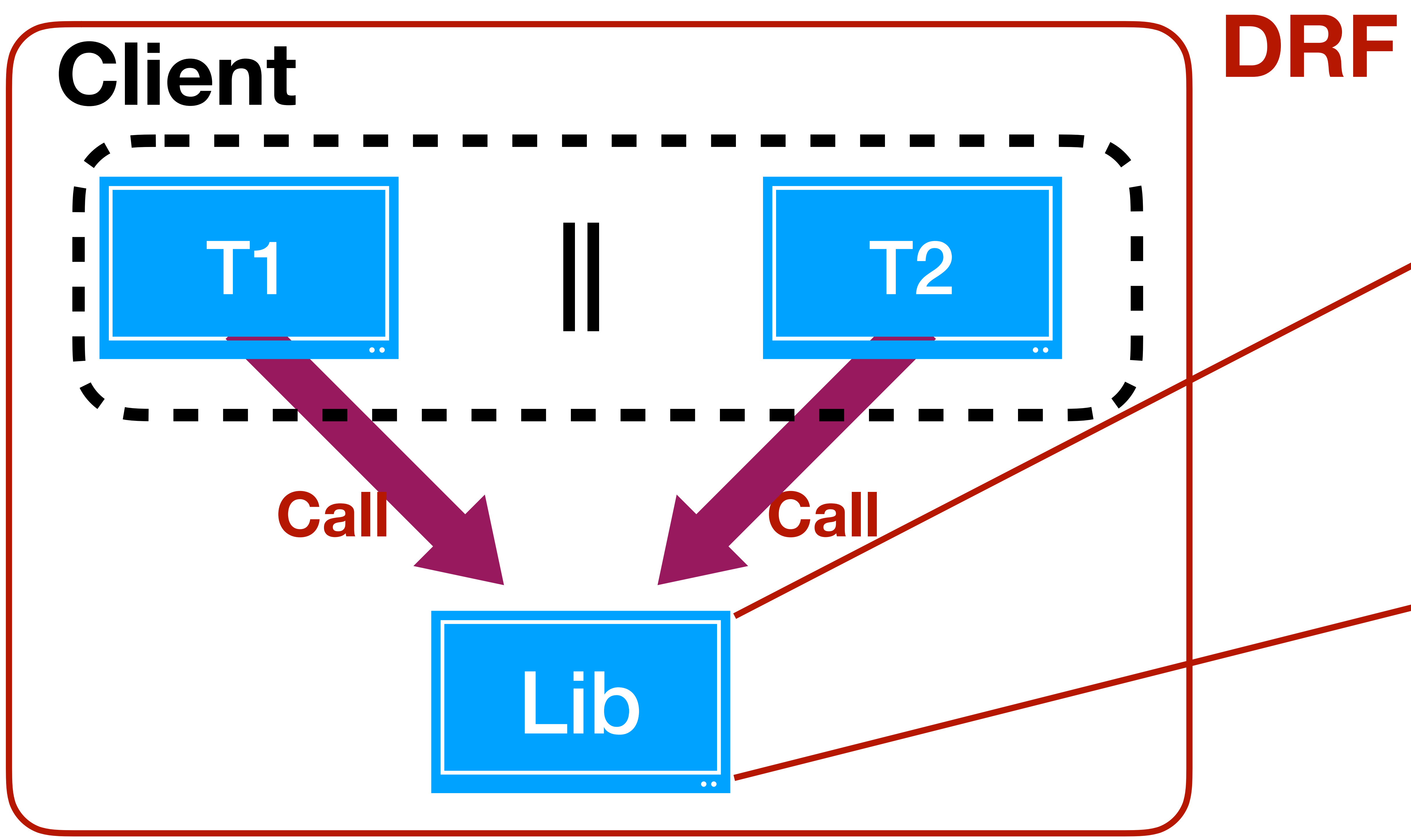


DRF Imposes Strong Restriction on Libraries




```
lock_rel:
             …
             mov    $1,    %eax
      lock   xchg   %eax,  L
             …
```

DRF Imposes Strong Restriction on Libraries



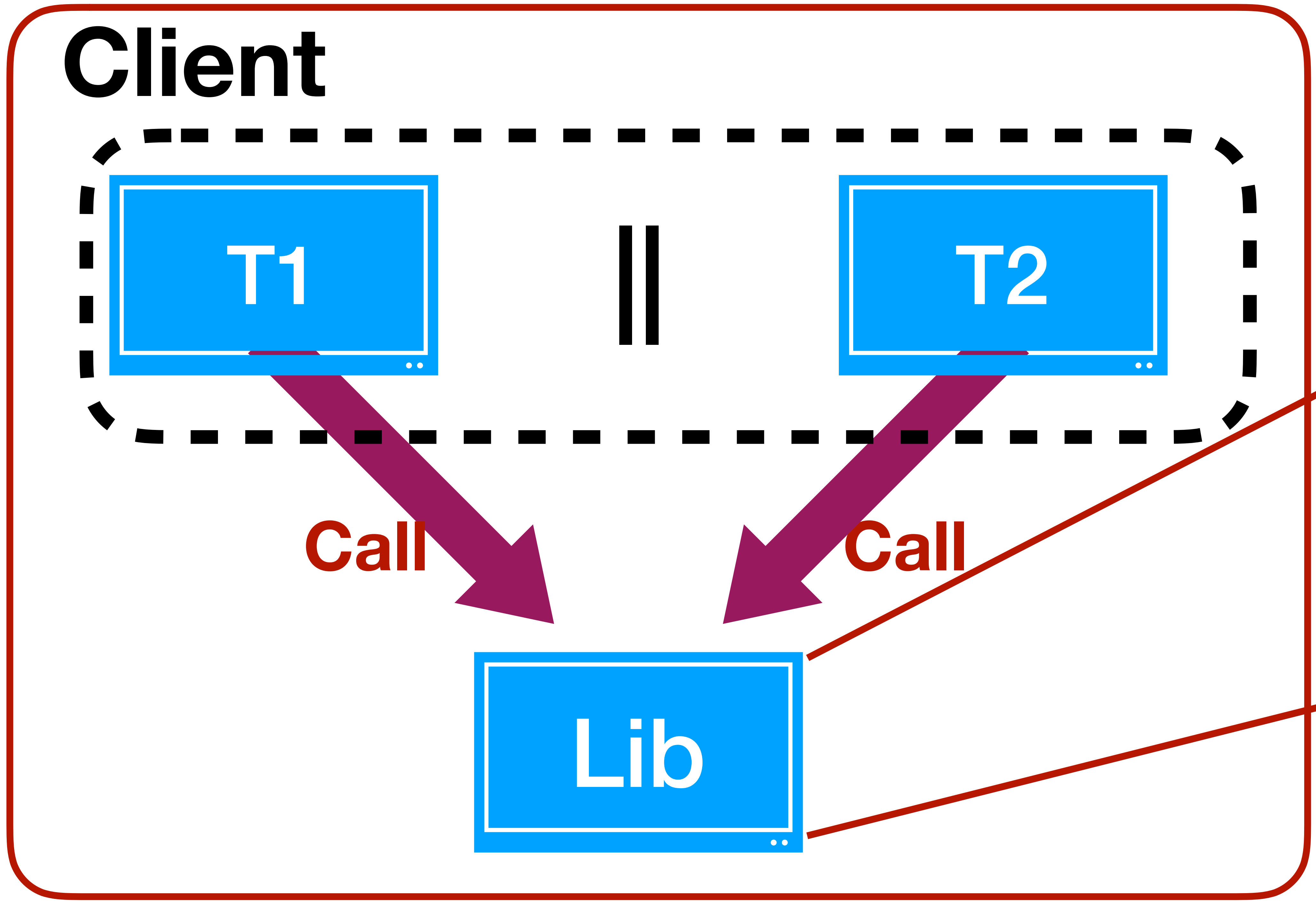
☹ inefficient ✗

```
lock_rel:
          …
          mov    $1,    %eax
  lock    xchg   %eax,  L
          …
```

```
lock_rel:
          …
          mov    $1,    L
          …
```

[spin-lock impl. in Linux 2.6]

DRF Imposes Strong Restriction on Libraries



DRF

☹ **inefficient** ✗

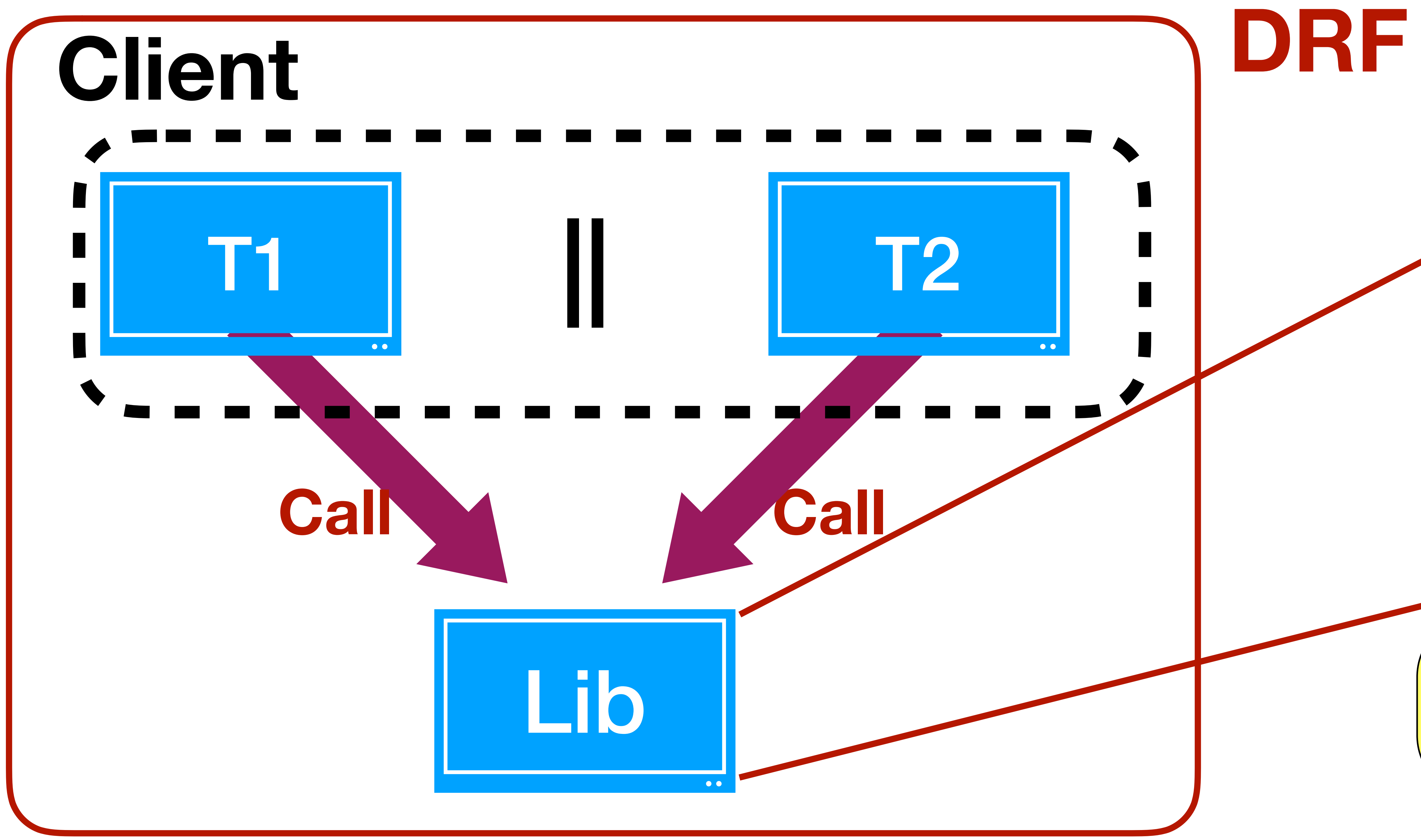
```
lock_rel:
            …
            mov    $1,     %eax
      lock  xchg   %eax,  L
            …
```

Racy

```
lock_rel:
            …
            mov    $1,     L
            …
```

[spin-lock impl. in Linux 2.6]

DRF Imposes Strong Restriction on Libraries



☹ inefficient

```
lock_rel:
            …
            mov    $1,    %eax
    lock    xchg   %eax,  L
            …
```

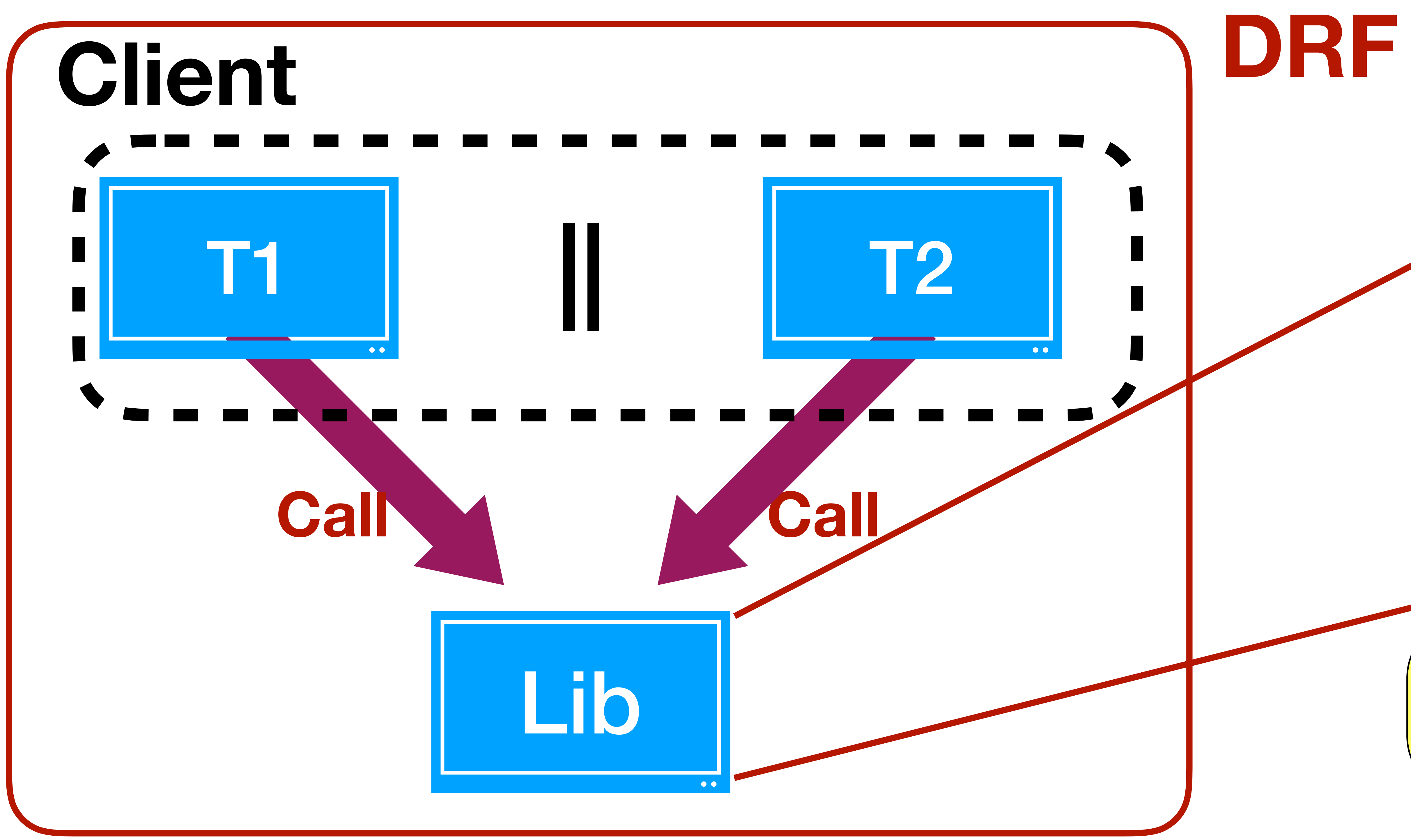
Racy

```
lock_rel:
            …
            mov    $1,    L
            …
```

Relaxed memory model, e.g. x86-TSO

[spin-lock impl. in Linux 2.6]

DRF Imposes Strong Restriction on Libraries



☹ inefficient ✘

```
lock_rel:
          …
          mov    $1,    %eax
    lock  xchg   %eax,  L
          …
```

Racy

?

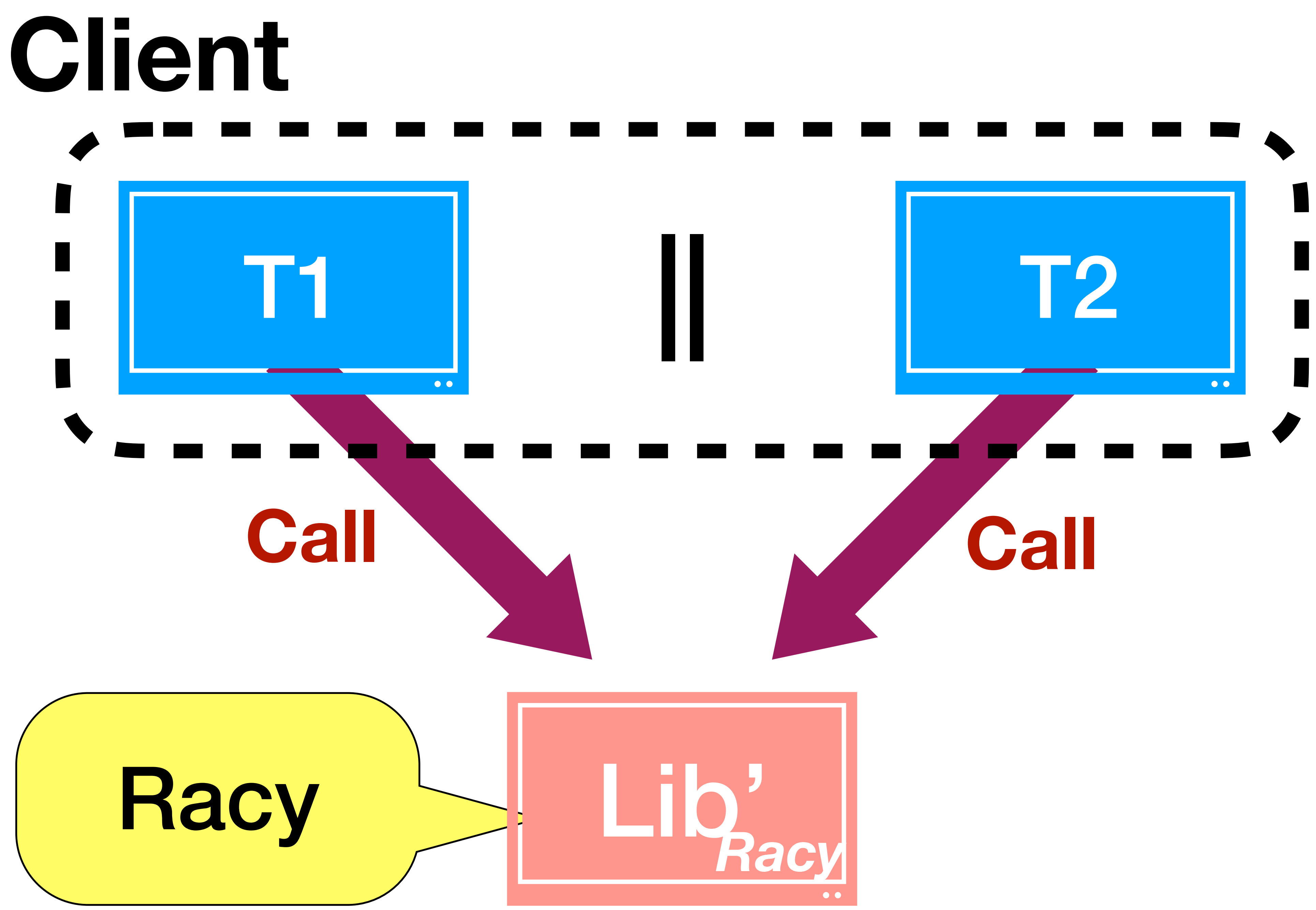
```
lock_rel:
          …
          mov    $1,    L
          …
```

Relaxed memory model,
e.g. x86-TSO

[spin-lock impl. in Linux 2.6]

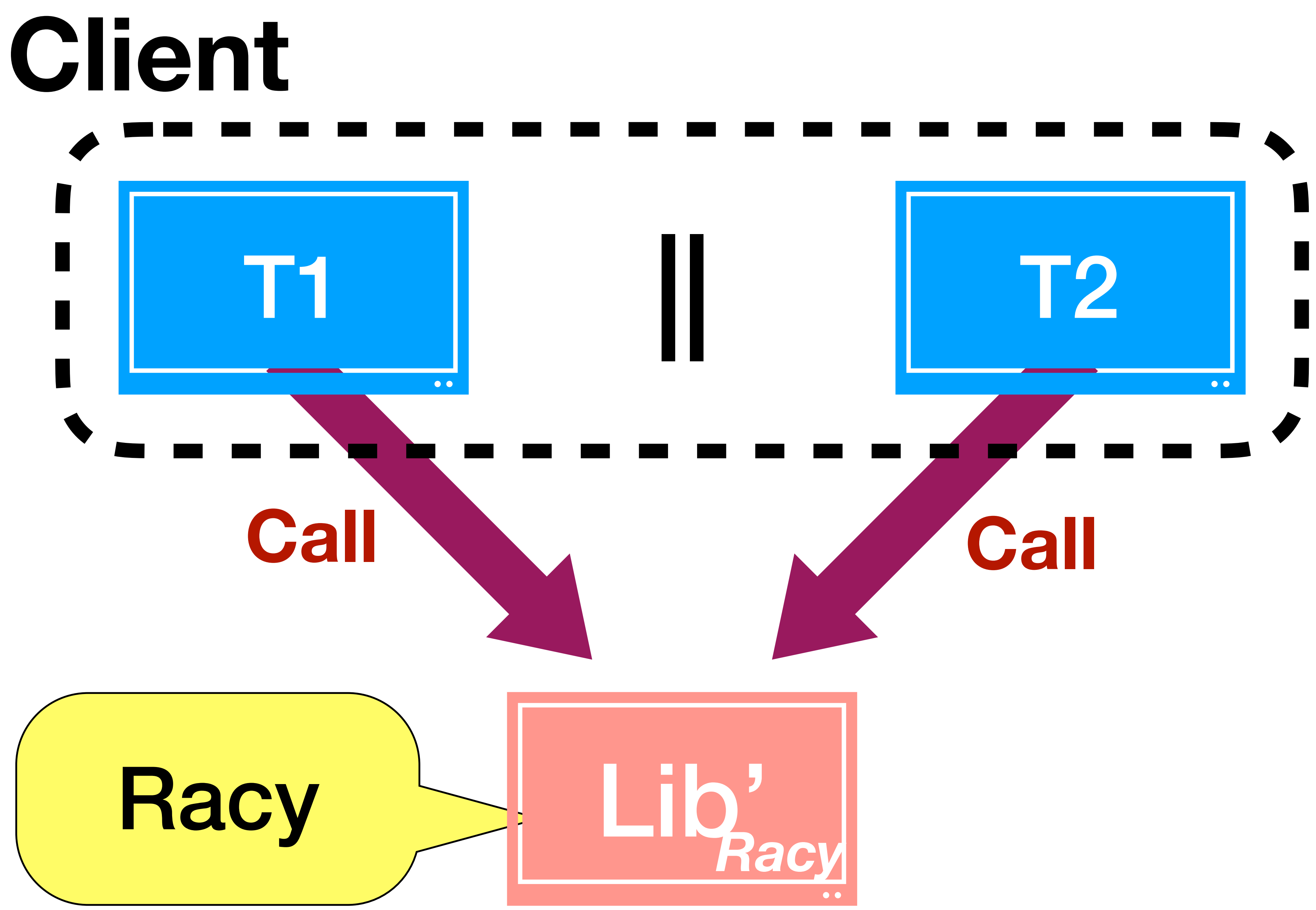
Our Idea

Our Idea



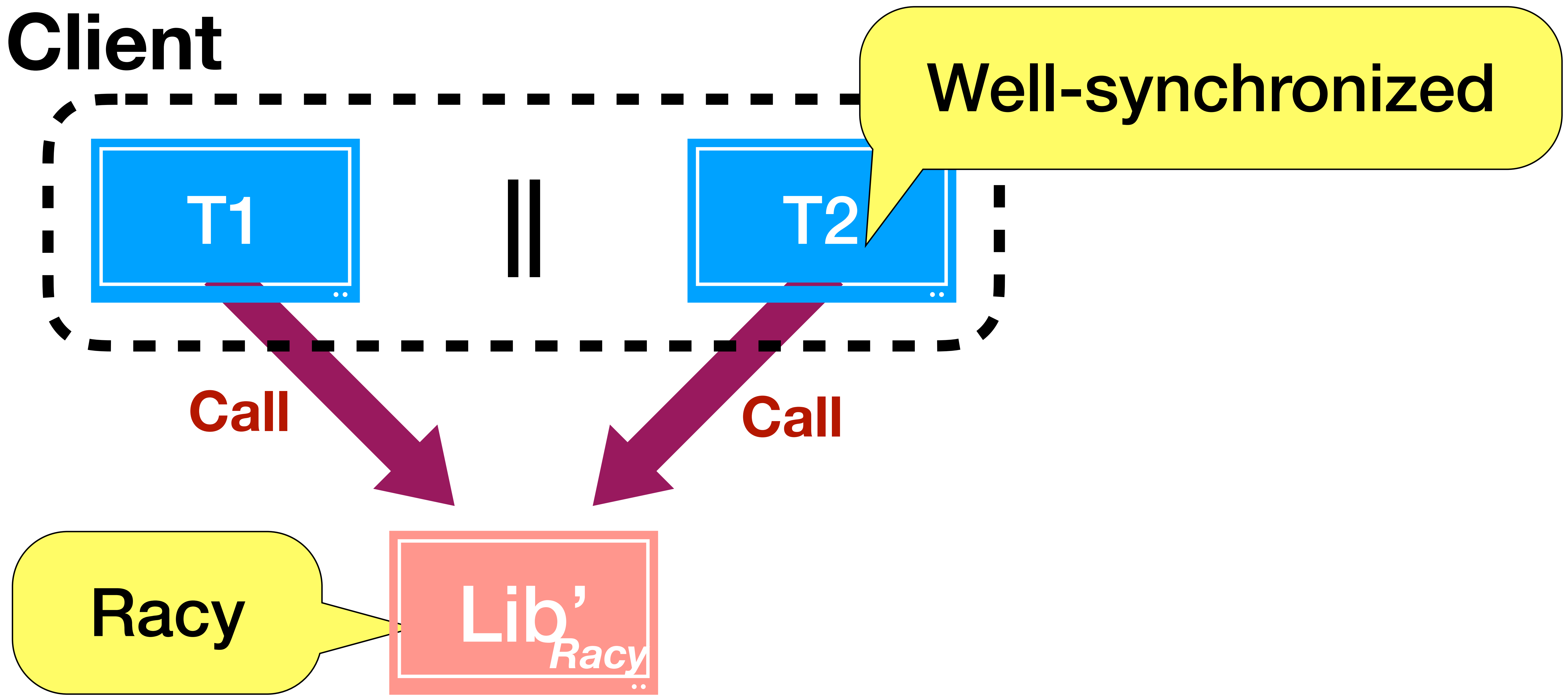
Our Idea

- Confined benign-races:
  - Racy libraries and client code run in separate memory regions



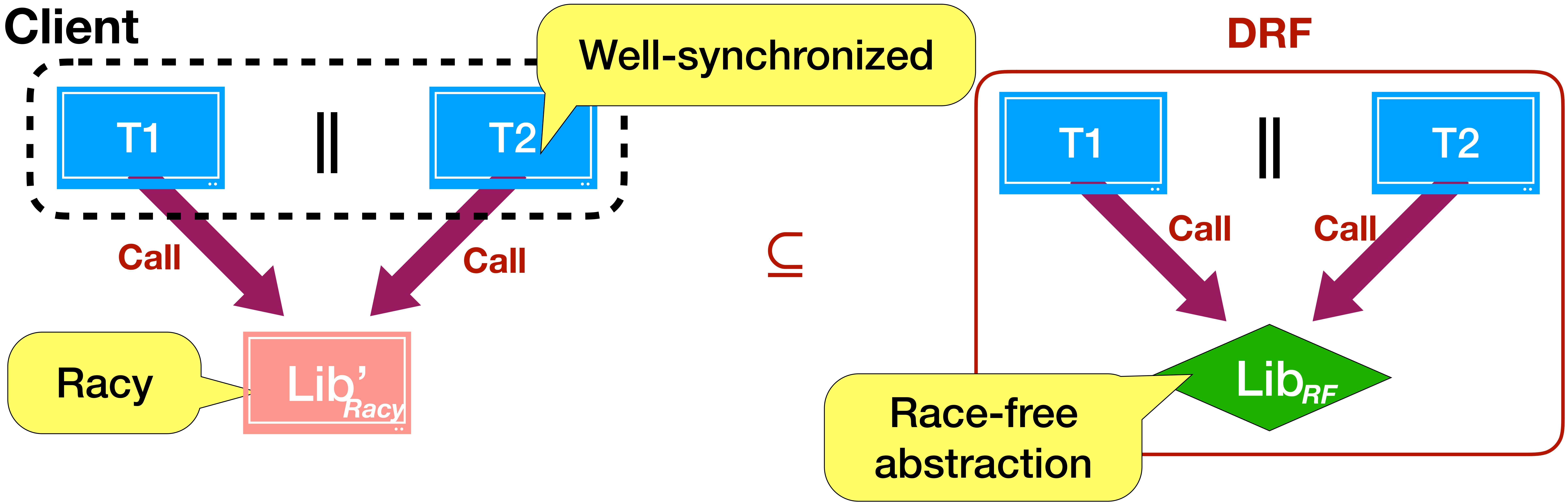
Our Idea

- Confined benign-races:
    - Racy libraries and client code run in separate memory regions
    - Client code be well-synchronized

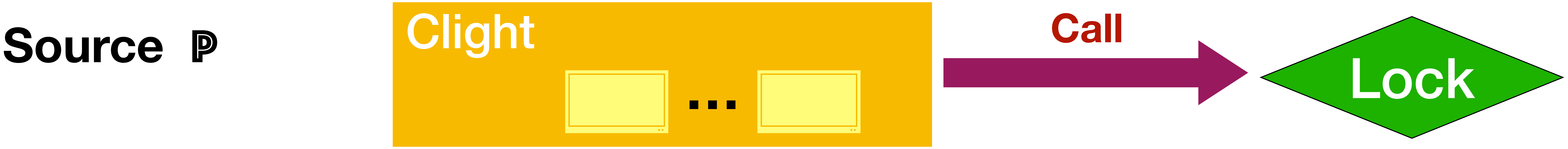


Our Idea

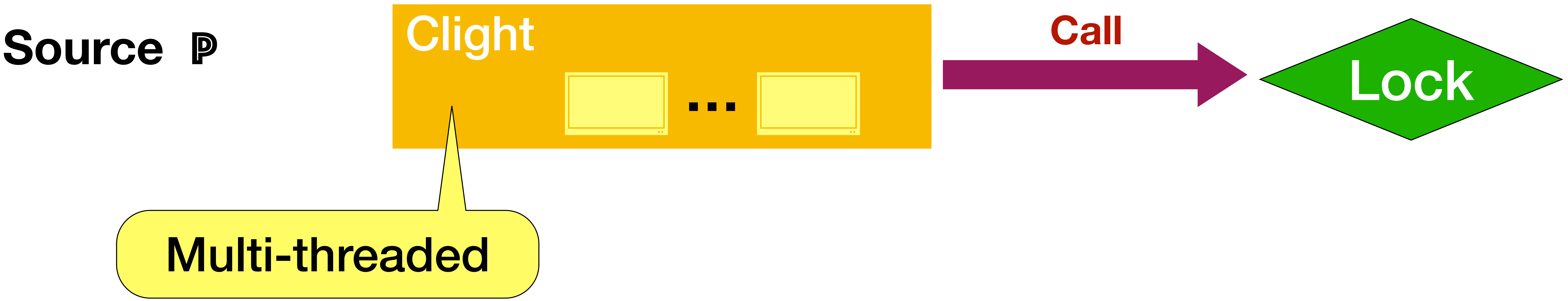
- Confined benign-races:
  - Racy libraries and client code run in separate memory regions
  - Client code be well-synchronized
  - Racy libraries have race-free abstraction



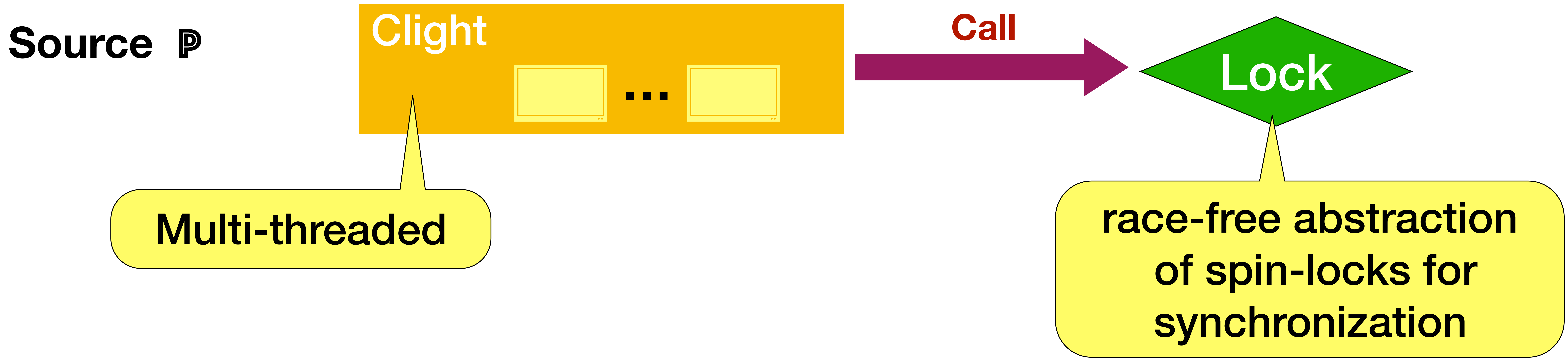
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



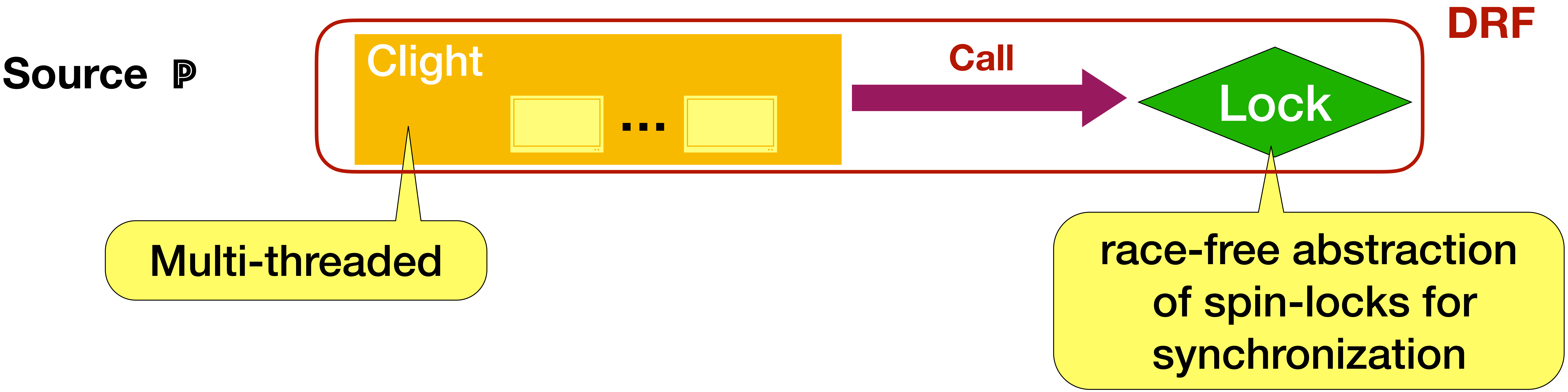
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



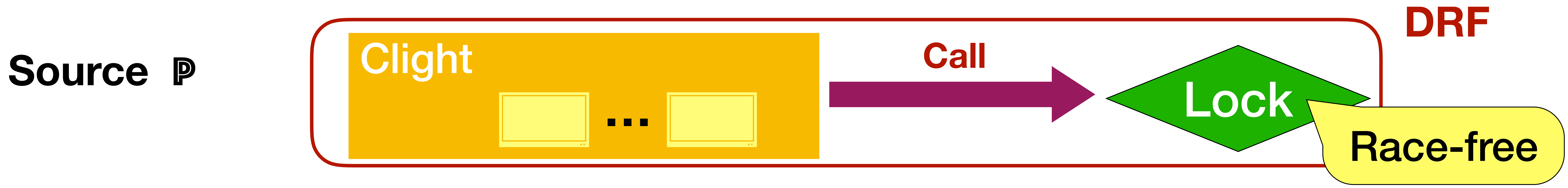
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



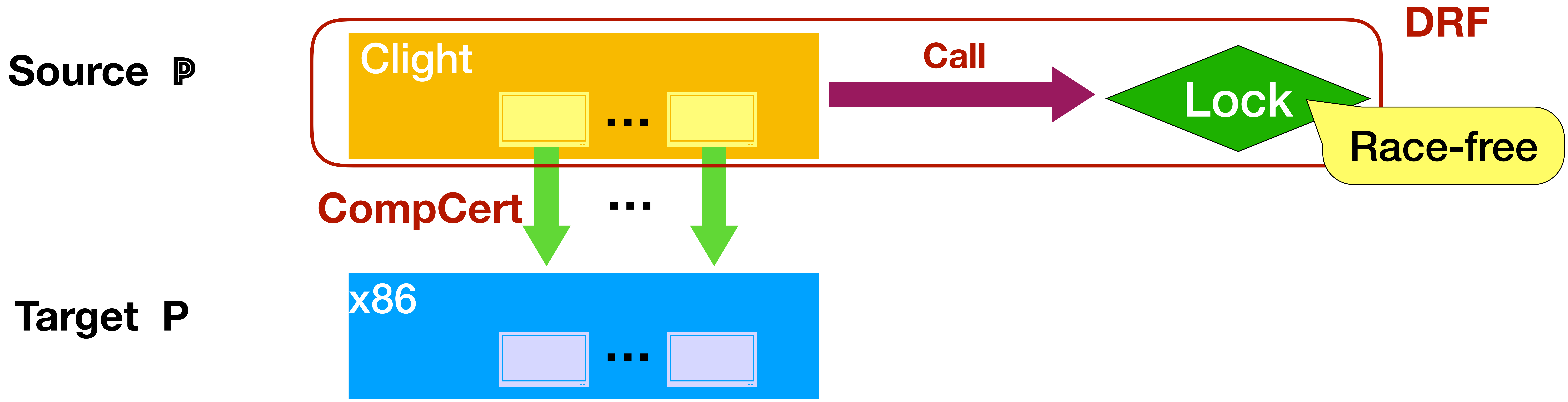
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



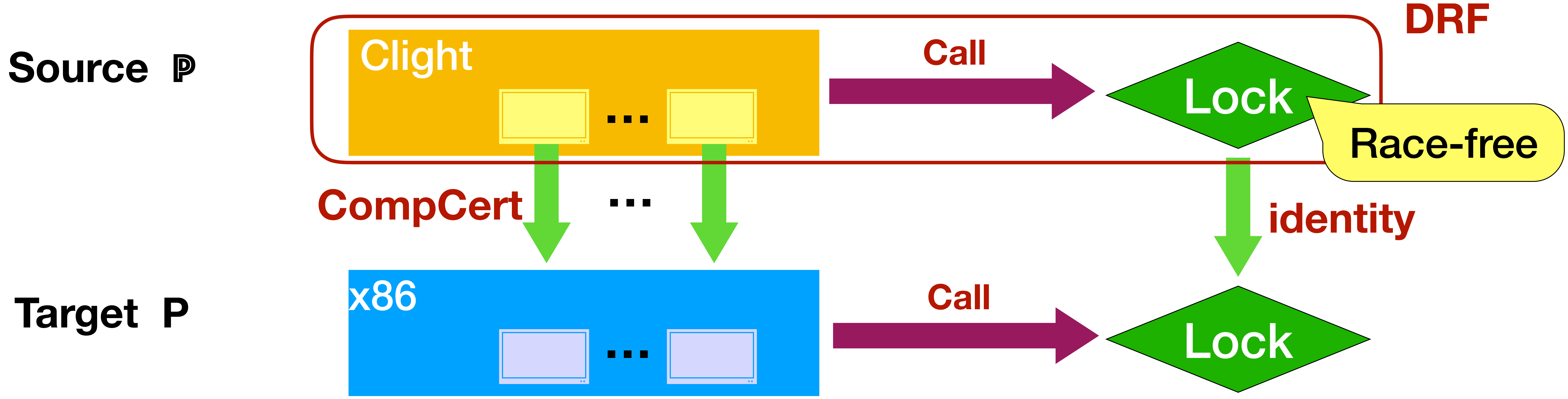
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



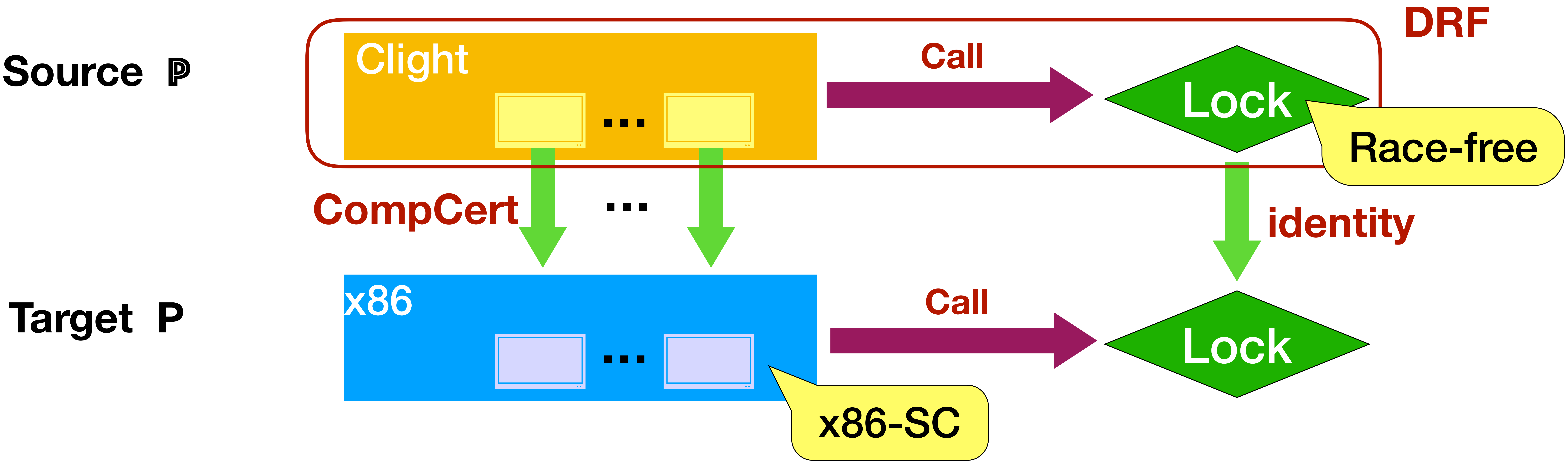
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



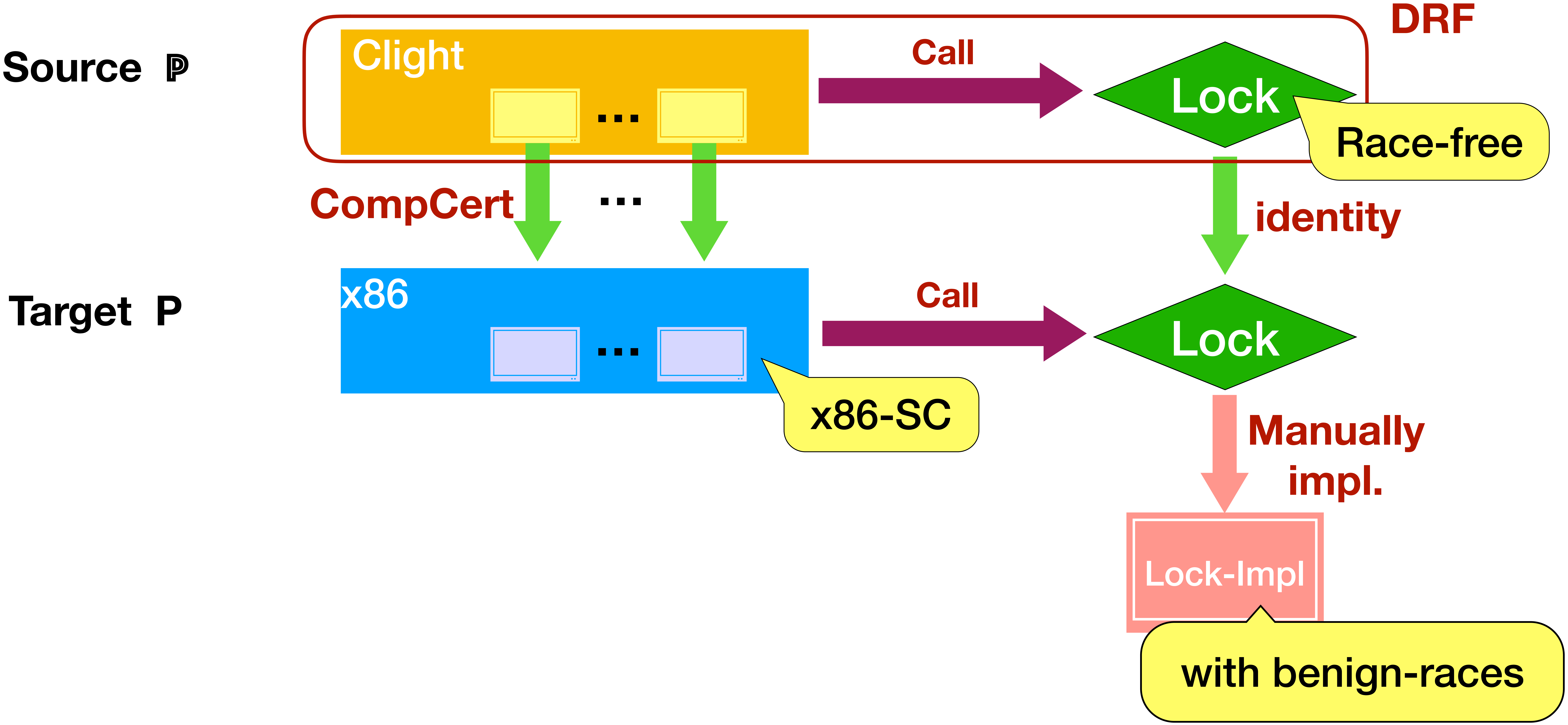
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



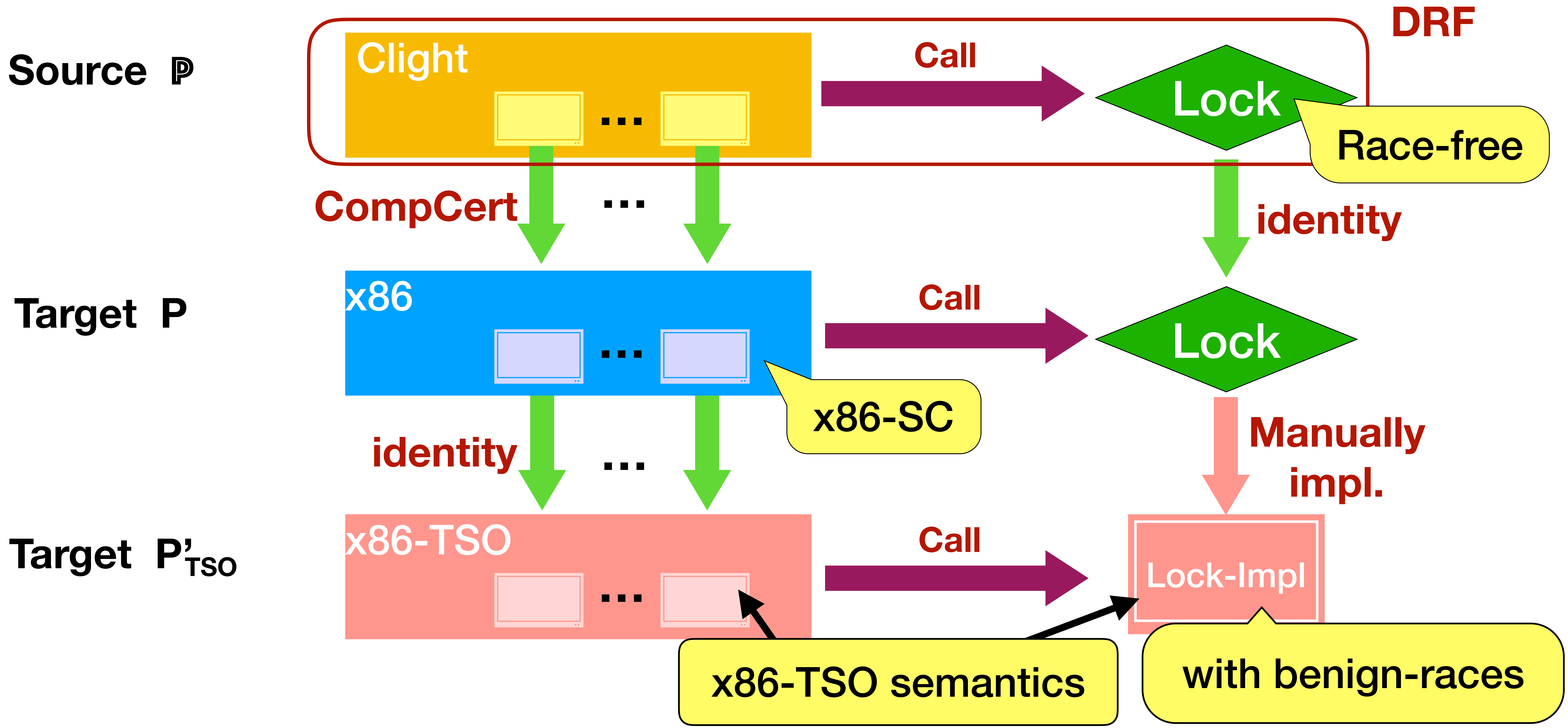
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



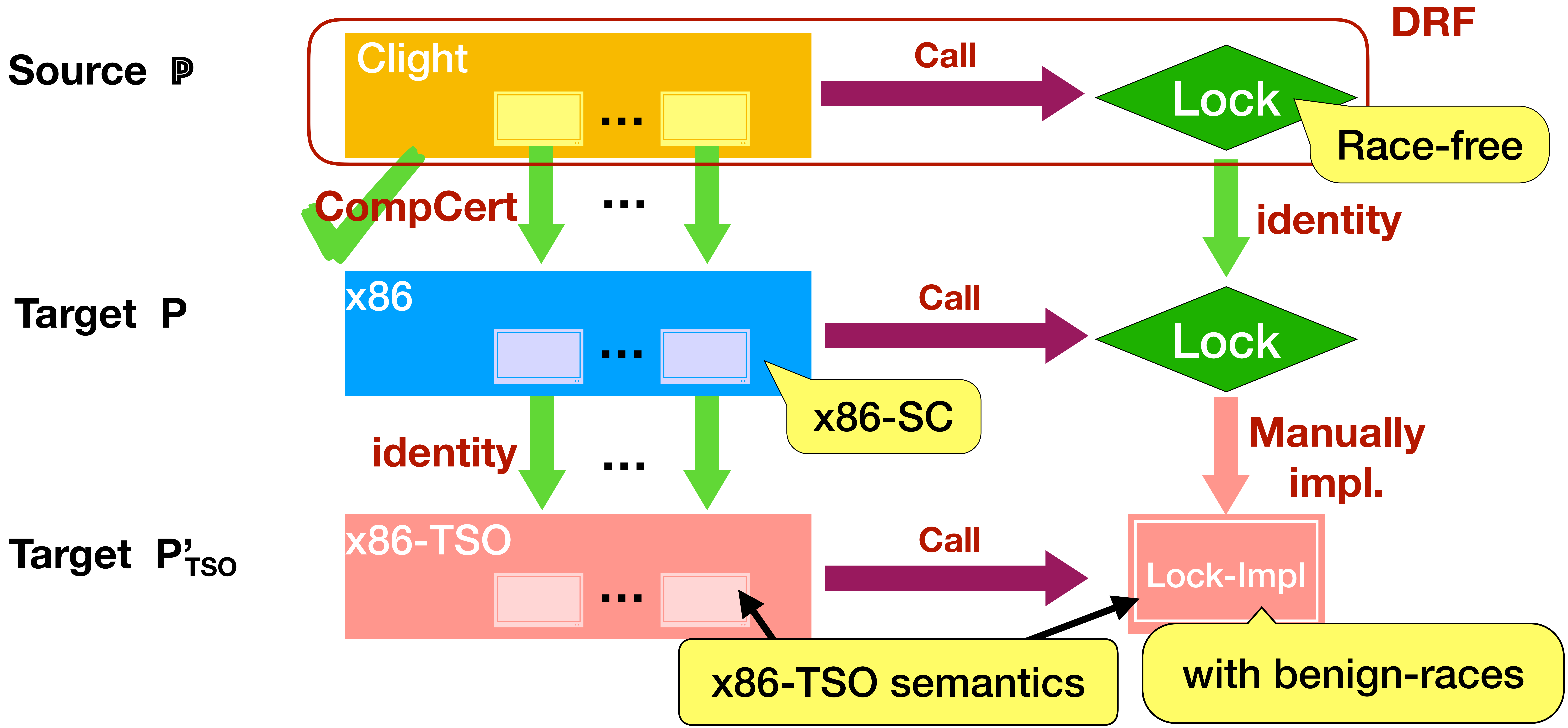
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



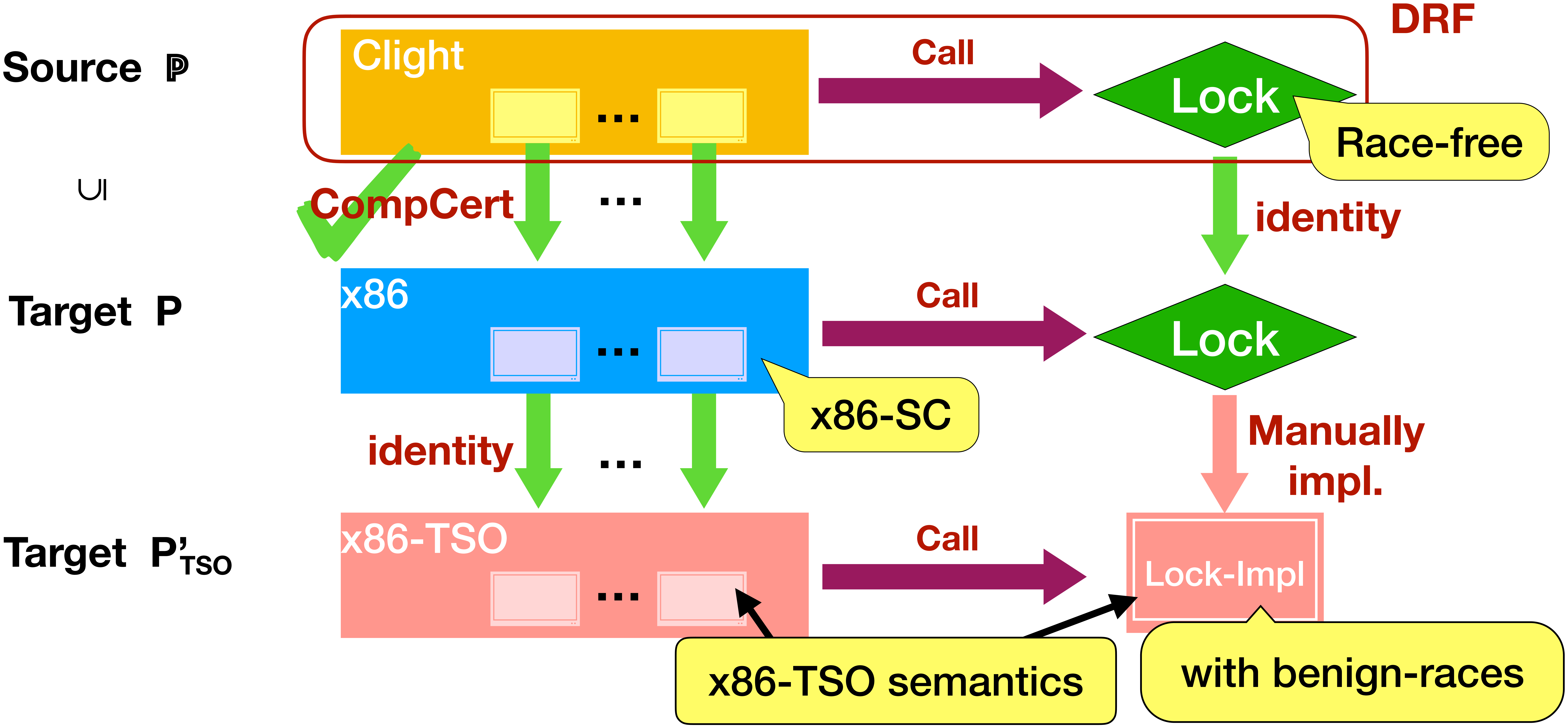
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



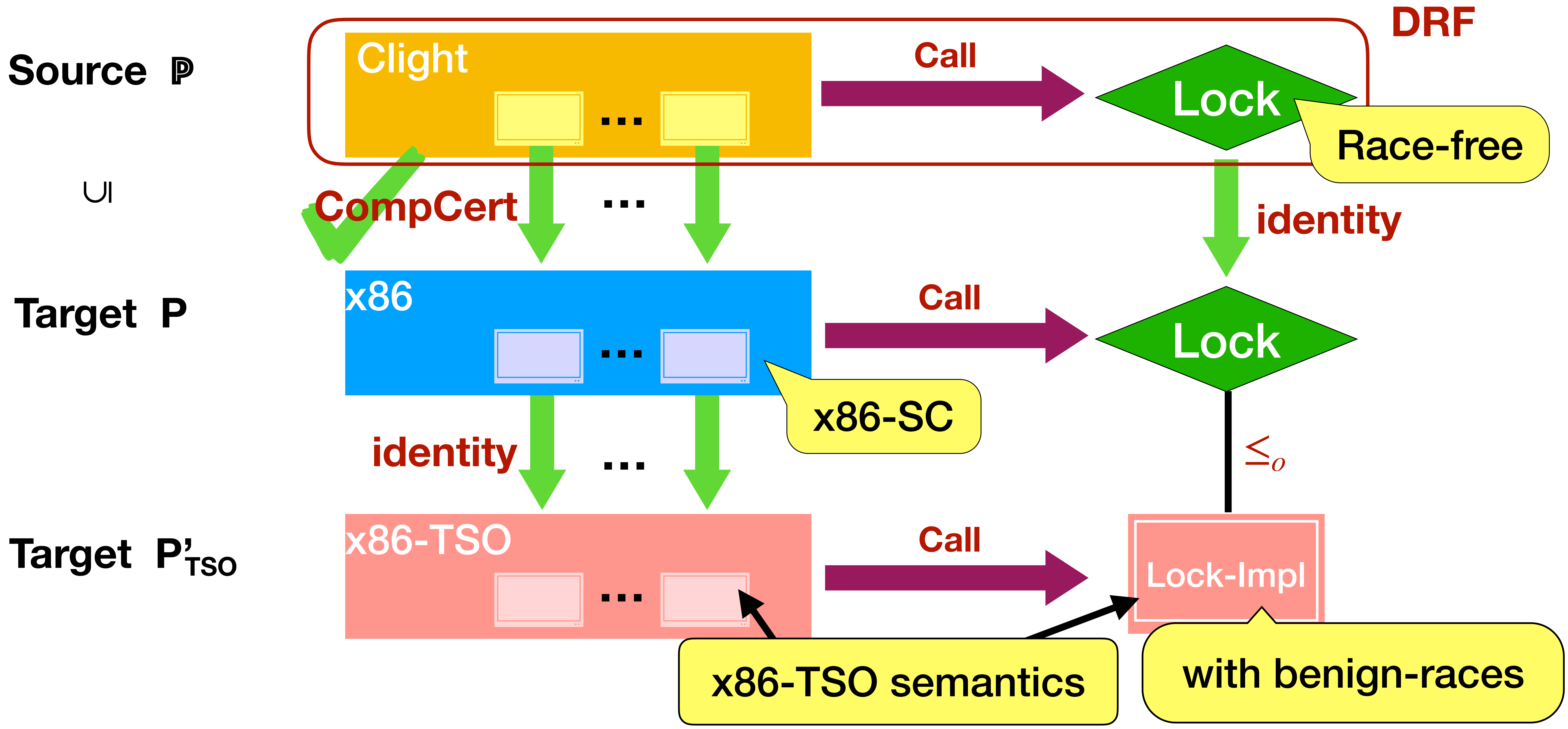
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



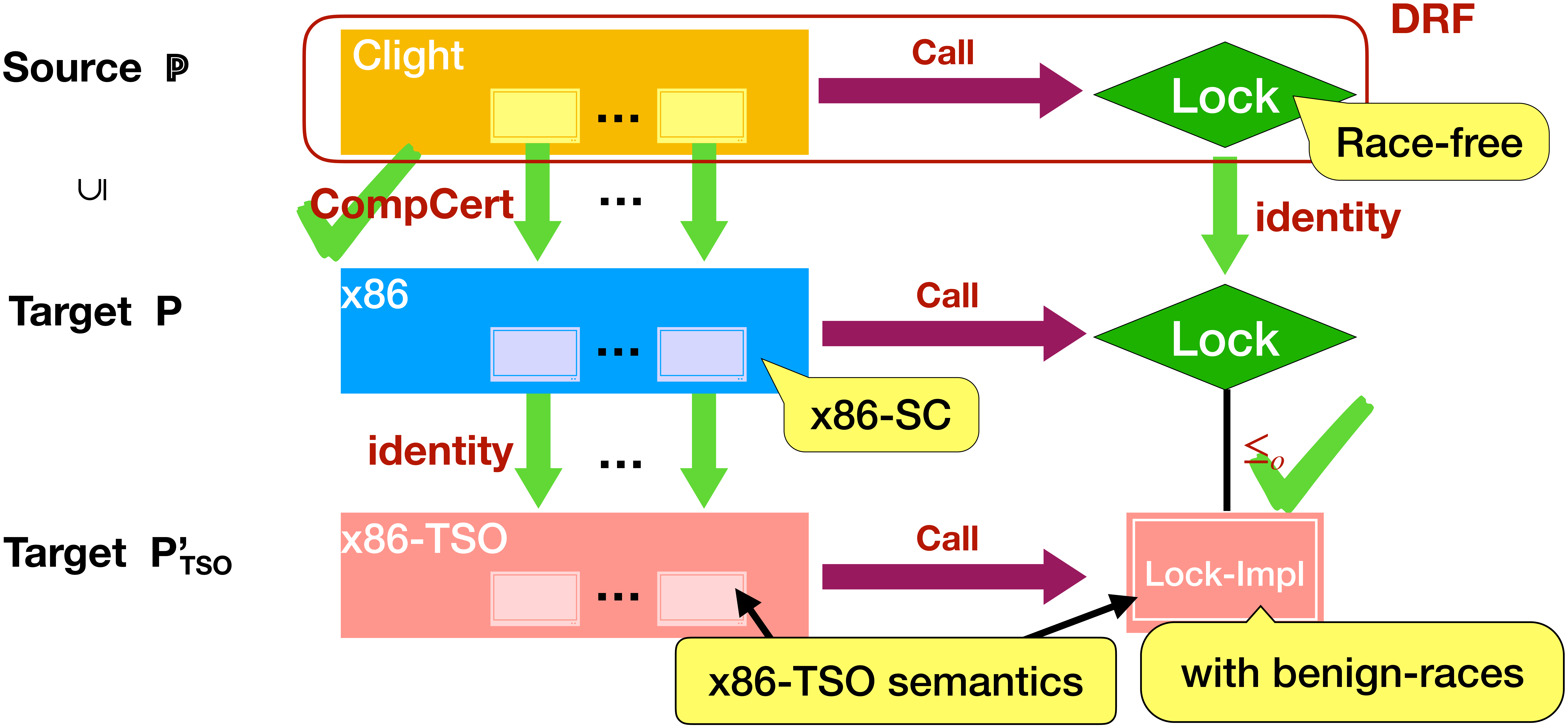
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



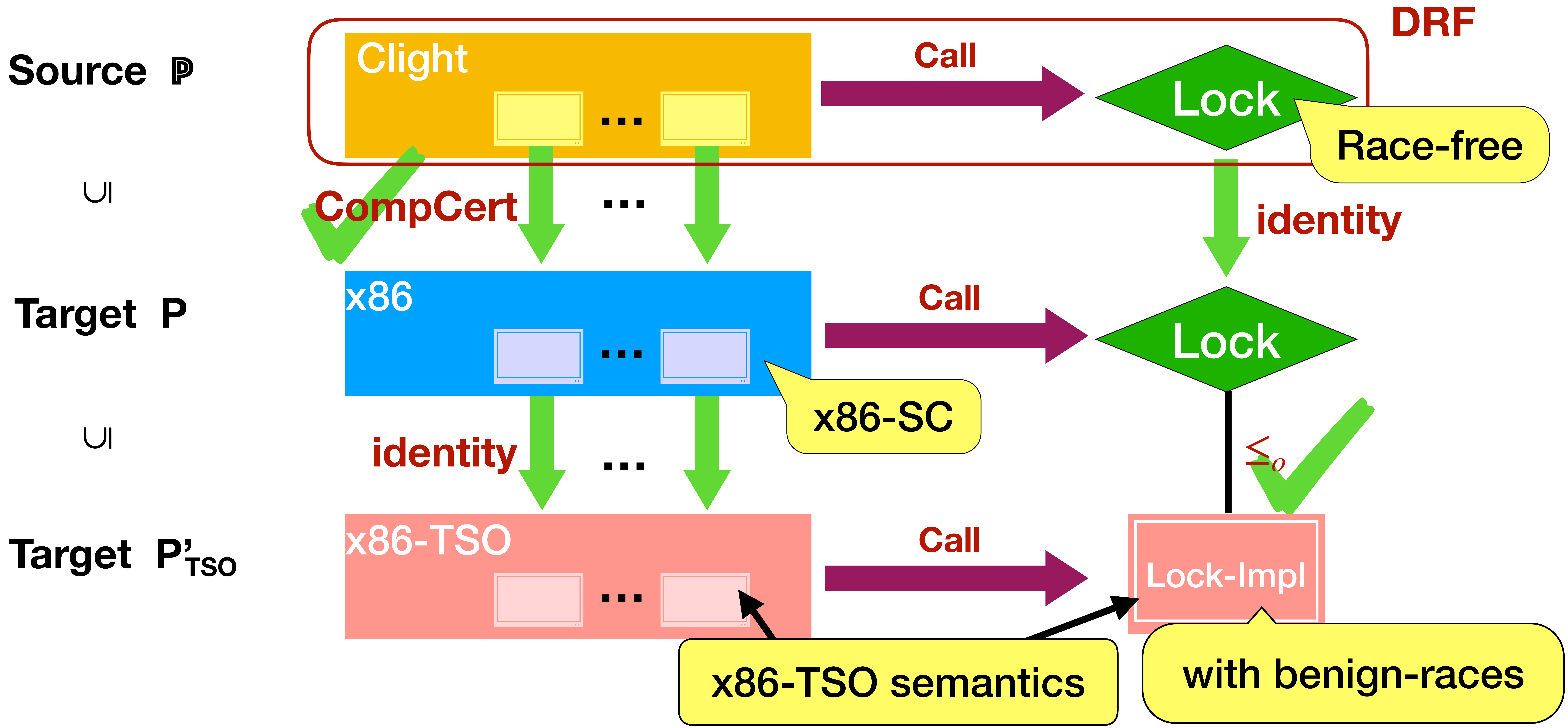
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



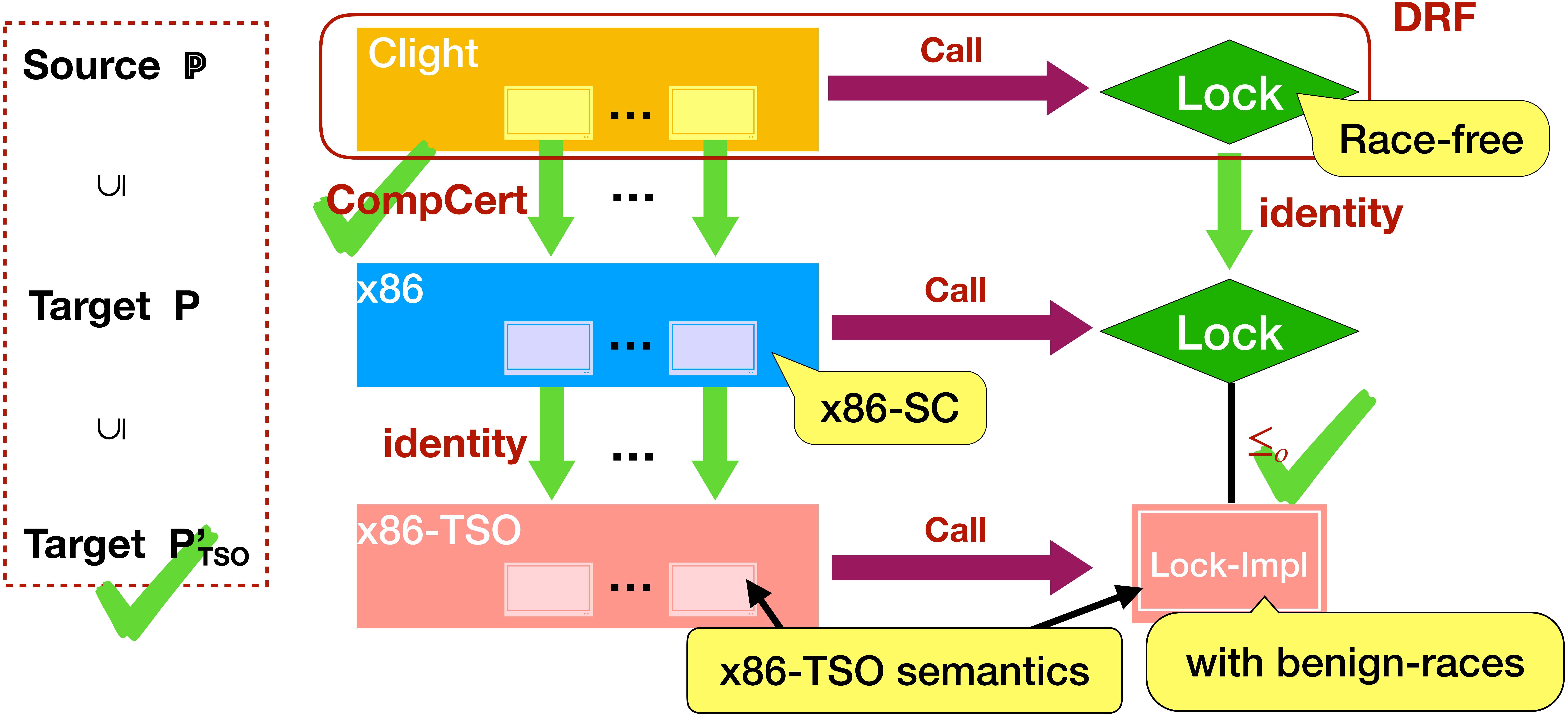
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



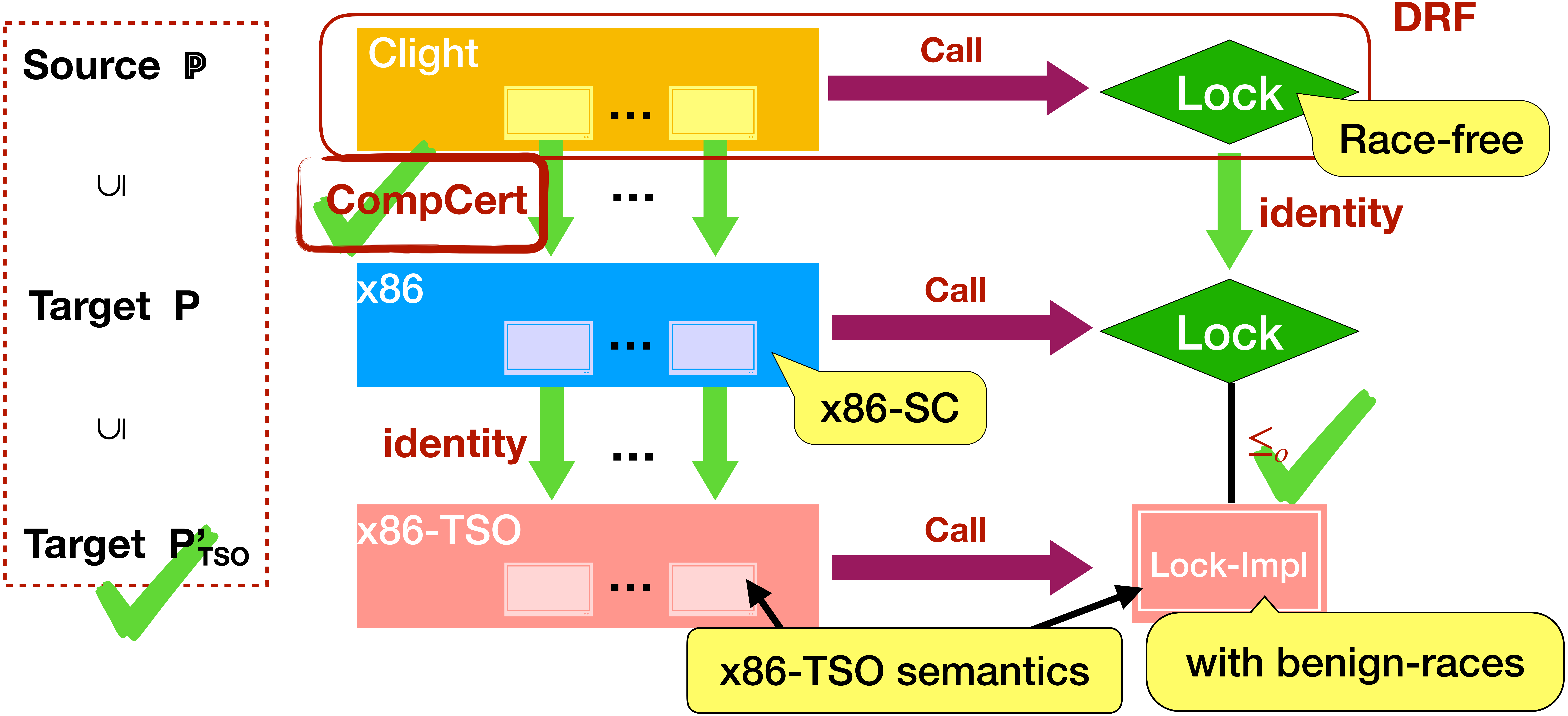
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



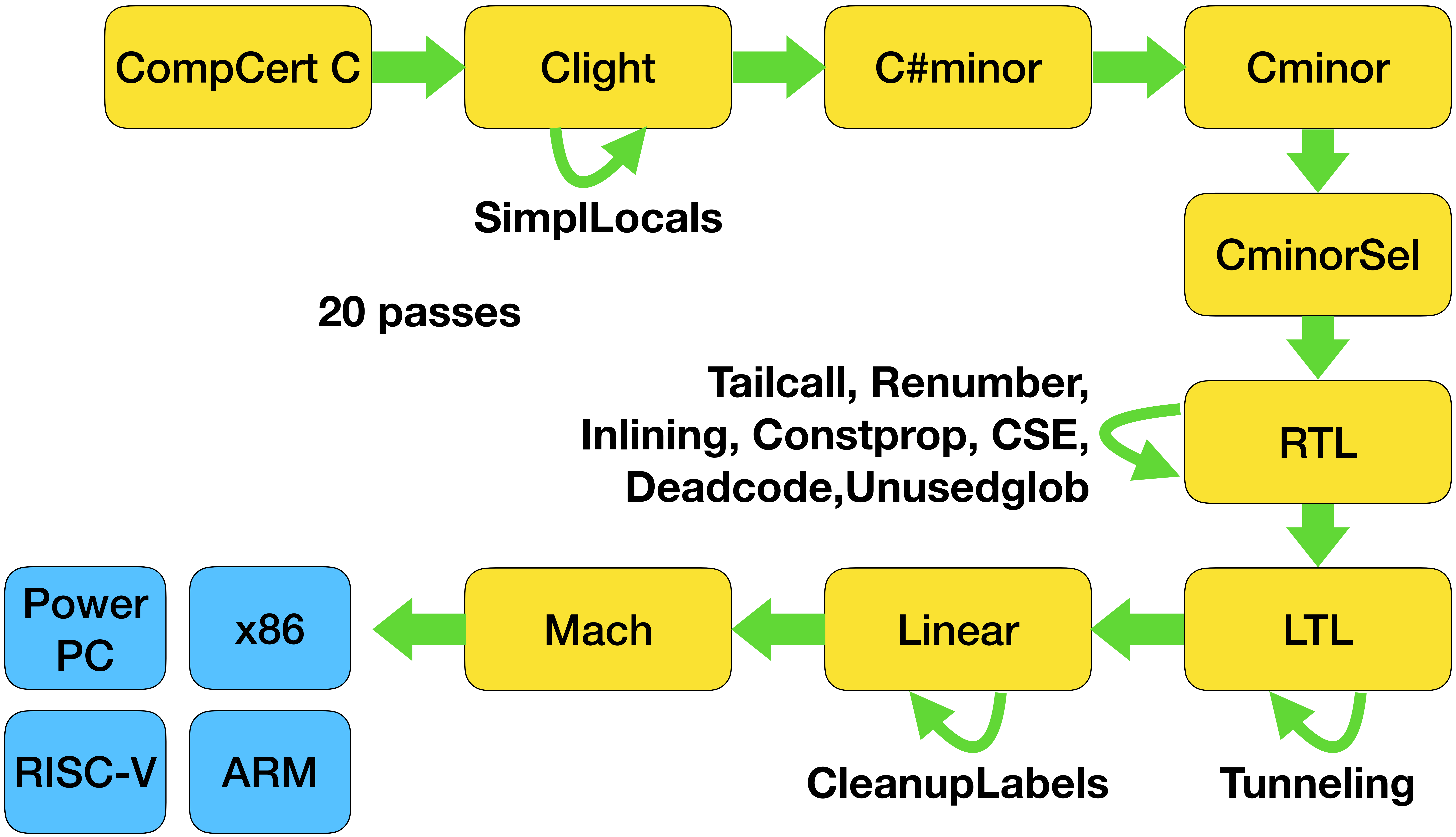
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



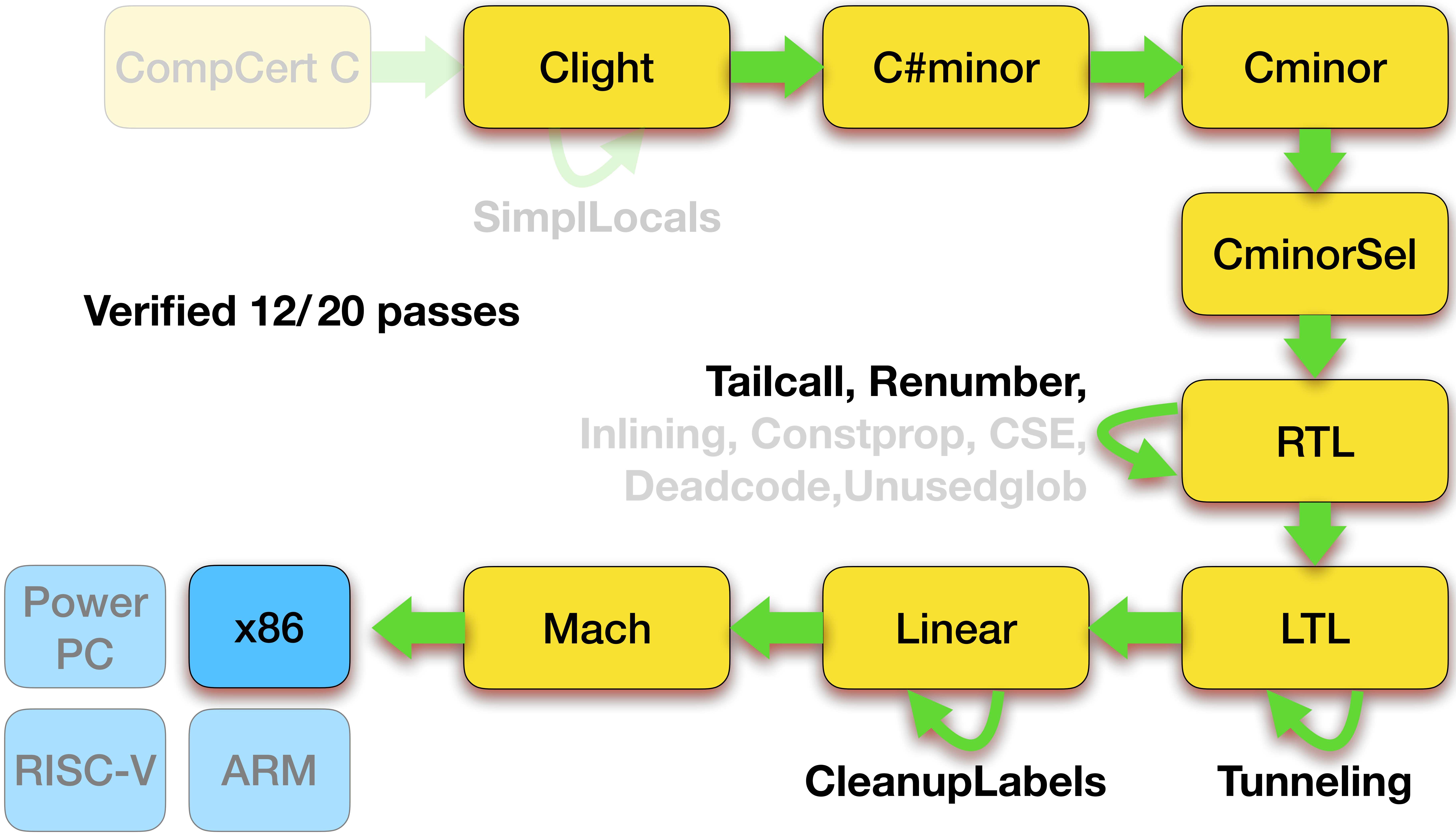
Supporting Confined Benign-Race & x86-TSO in Our CASCompCert



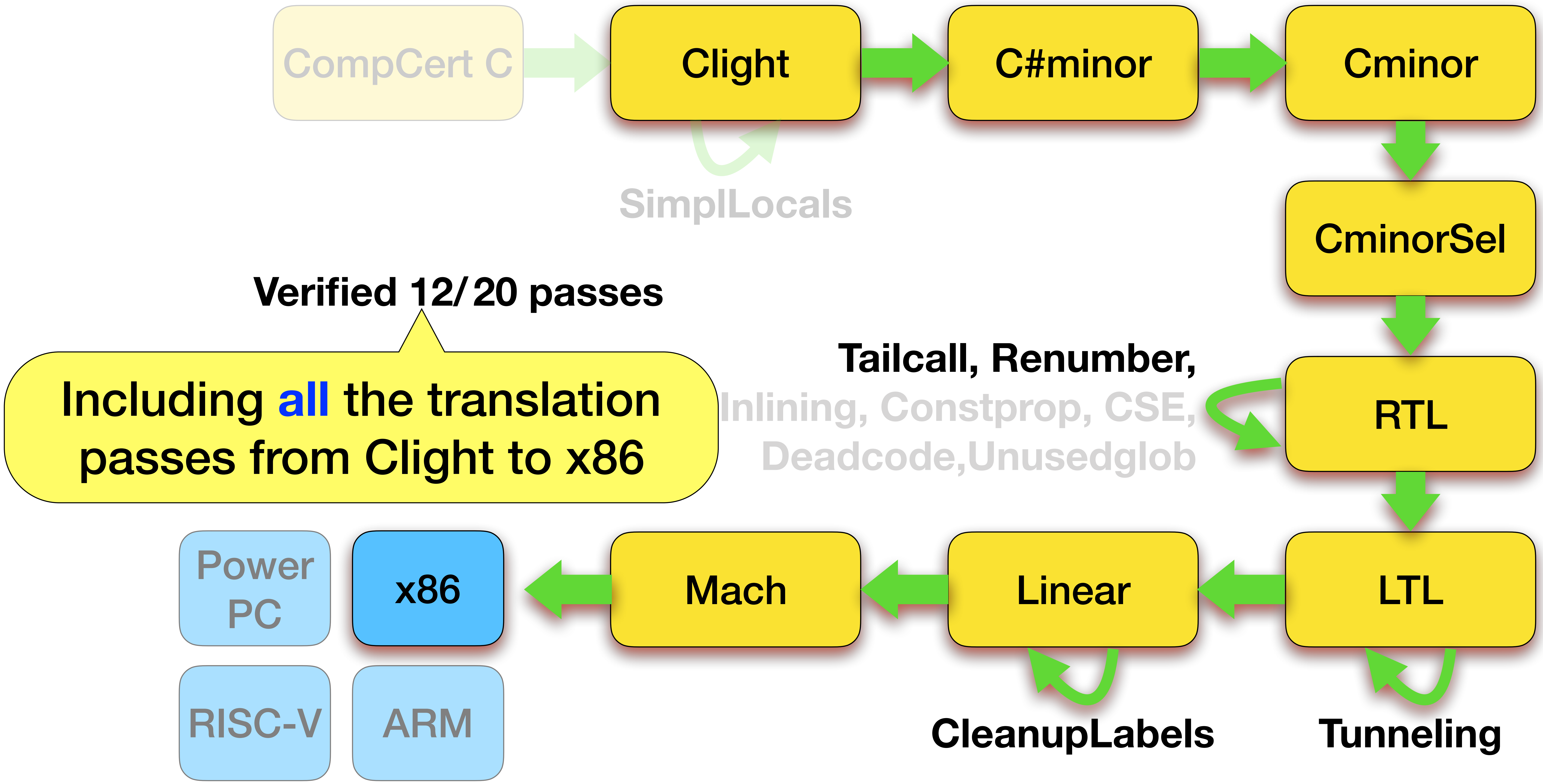
CompCert Passes



Verified 12/20 Passes in CASCompCert



Verified 12/20 Passes in CASCompCert



Reused Considerable Amount of CompCert Proofs.

Framework is Challenging to Implement.

Compilation passes	Spec		Proof	
	CompCert	Ours	CompCert	Ours
Cshmgen	515	1021	1071	1503
Cminorgen	753	1556	1152	1251
Selection	336	500	647	783
RTLgen	428	543	821	862
Tailcall	173	328	275	405
Renumber	86	245	117	358
Allocation	704	785	1410	1700
Tunneling	131	339	166	475
Linearize	236	371	349	733
CleanupLabels	126	387	161	388
Stacking	730	1038	1108	2135
Asmgen	208	338	571	1128

Reused Considerable Amount of CompCert Proofs.

Framework is Challenging to Implement.

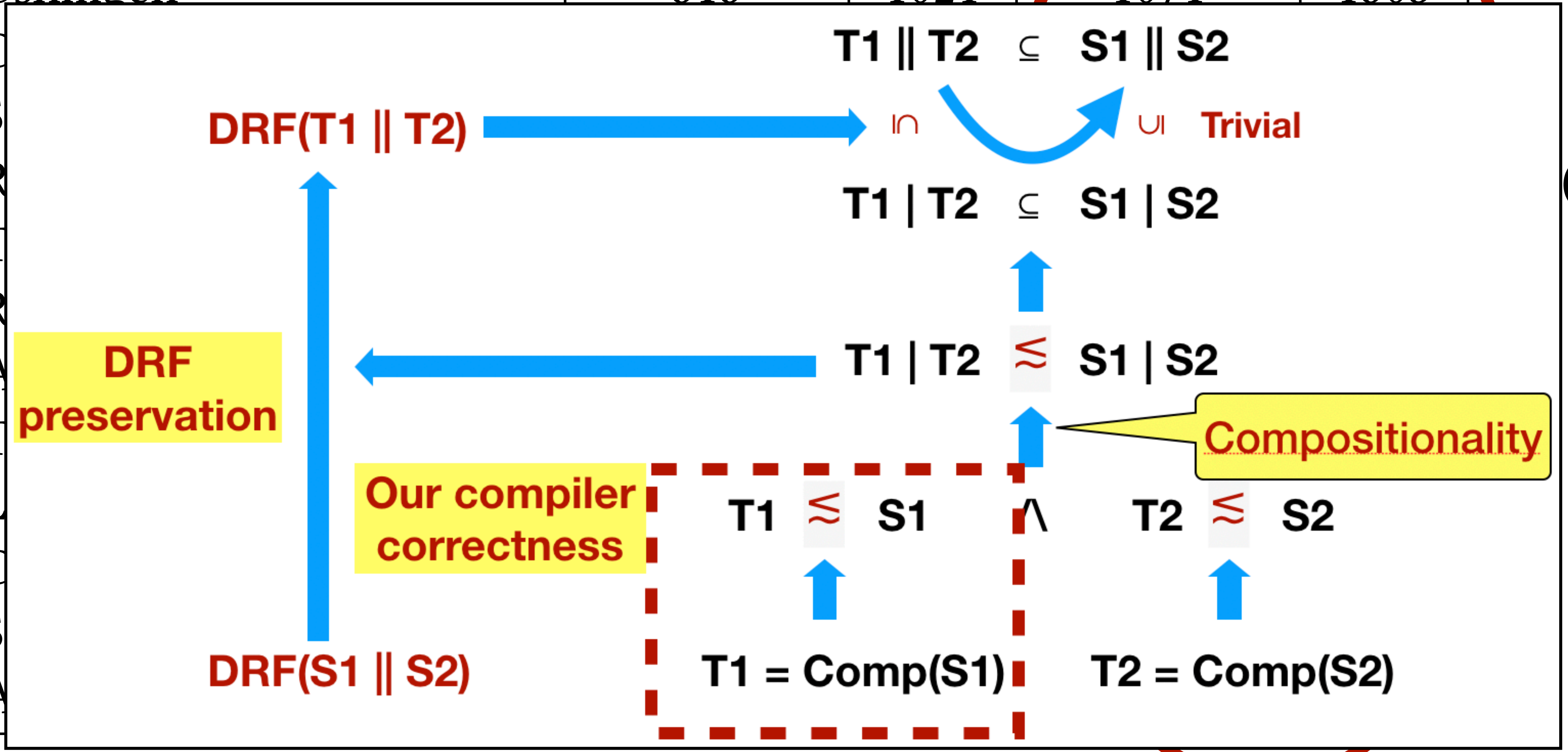
Compilation passes	Spec		Proof	
	CompCert	Ours	CompCert	Ours
Cshmgen	515	1021	1071	1503
Cminorgen	753	1556	1152	1251
Selection	336	500	647	783
RTLgen	428	543	821	862
Tailcall	173	328	275	405
Renumber	86	245	117	358
Allocation	704	785	1410	1700
Tunneling	131	339	166	475
Linearize	236	371	349	733
CleanupLabels	126	387	161	388
Stacking	730	1038	1108	2135
Asmgen	208	338	571	1128

Compiler verification:
100 - 400 more lines of Coq proof for most passes

Reused Considerable Amount of CompCert Proofs.

Framework is Challenging to Implement.

Compilation passes	Spec		Proof	
	CompCert	Ours	CompCert	Ours
Cshmgen	515	1021	1071	1503



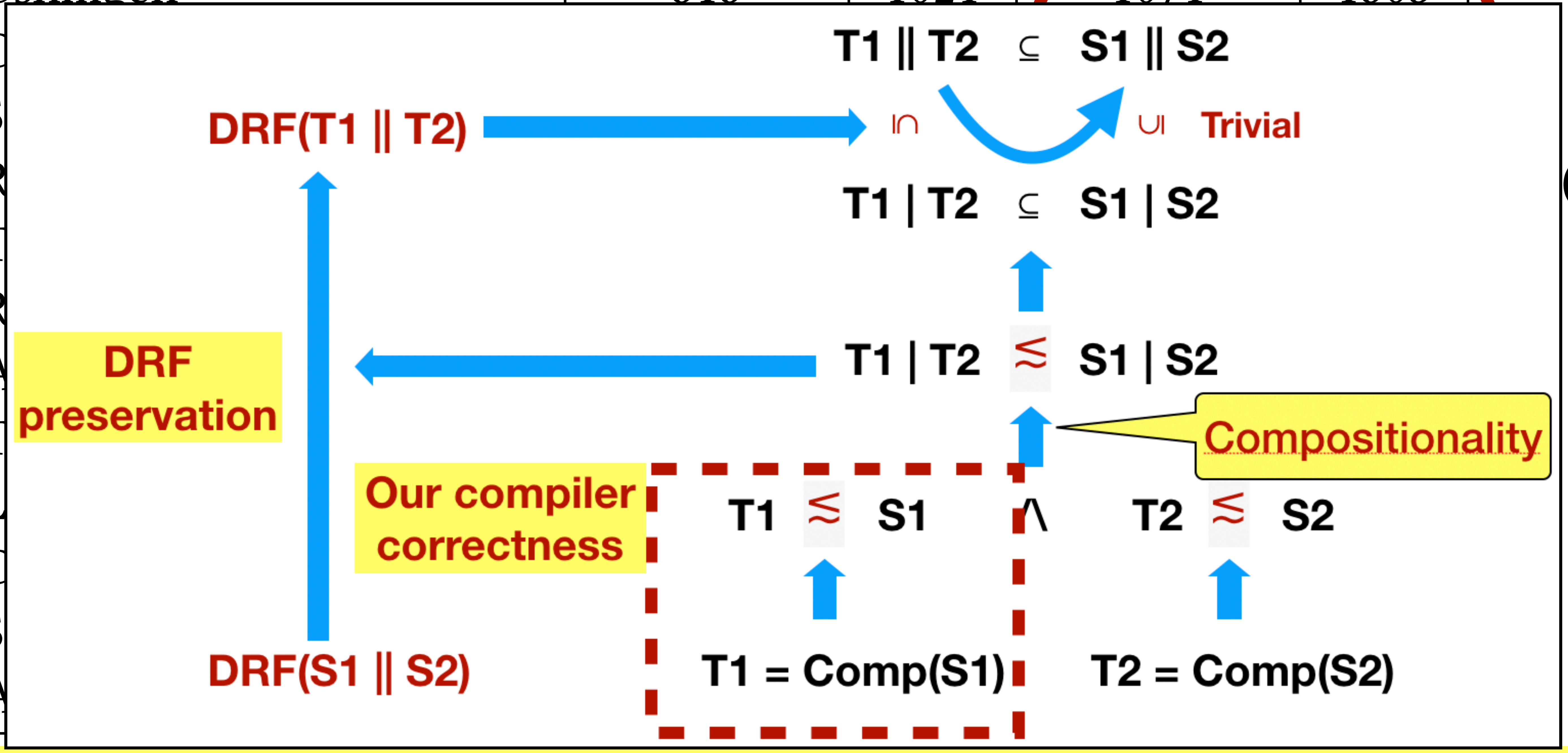
Compiler verification:

100 - 400 more lines of Coq proof for most passes

Reused Considerable Amount of CompCert Proofs.

Framework is Challenging to Implement.

Compilation passes	Spec		Proof	
	CompCert	Ours	CompCert	Ours
Cshmgen	515	1021	1071	1503
C				
S				
R				
T				
R				
A				
T				
L				
C				
S				
A				



Compiler verification:
100 - 400 more lines of
Coq proof for most passes

Framework impl.: > 60k LoC, ~ 1 person year

Conclusion

Conclusion

- **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
  - **Well-defined language** for language-independent DRF
  - **Footprint-preserving simulation** for DRF preservation

Conclusion

- **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
  - **Well-defined language** for language-independent DRF
  - **Footprint-preserving simulation** for DRF preservation
- Framework extension:
  - Support **x86-TSO** + **confined benign-races**

Conclusion

- **Language independent** verification framework
  - Key semantics components + proof structures
  - Supports separate compilation for **race-free** concurrent programs
  - **Well-defined language** for language-independent DRF
  - **Footprint-preserving simulation** for DRF preservation
- Framework extension:
  - Support **x86-TSO** + **confined benign-races**
- **CAS**CompCert:
  - Reused considerable amount of CompCert proofs
  - Racy x86-TSO impl. of locks as synchronization library

Thank you!