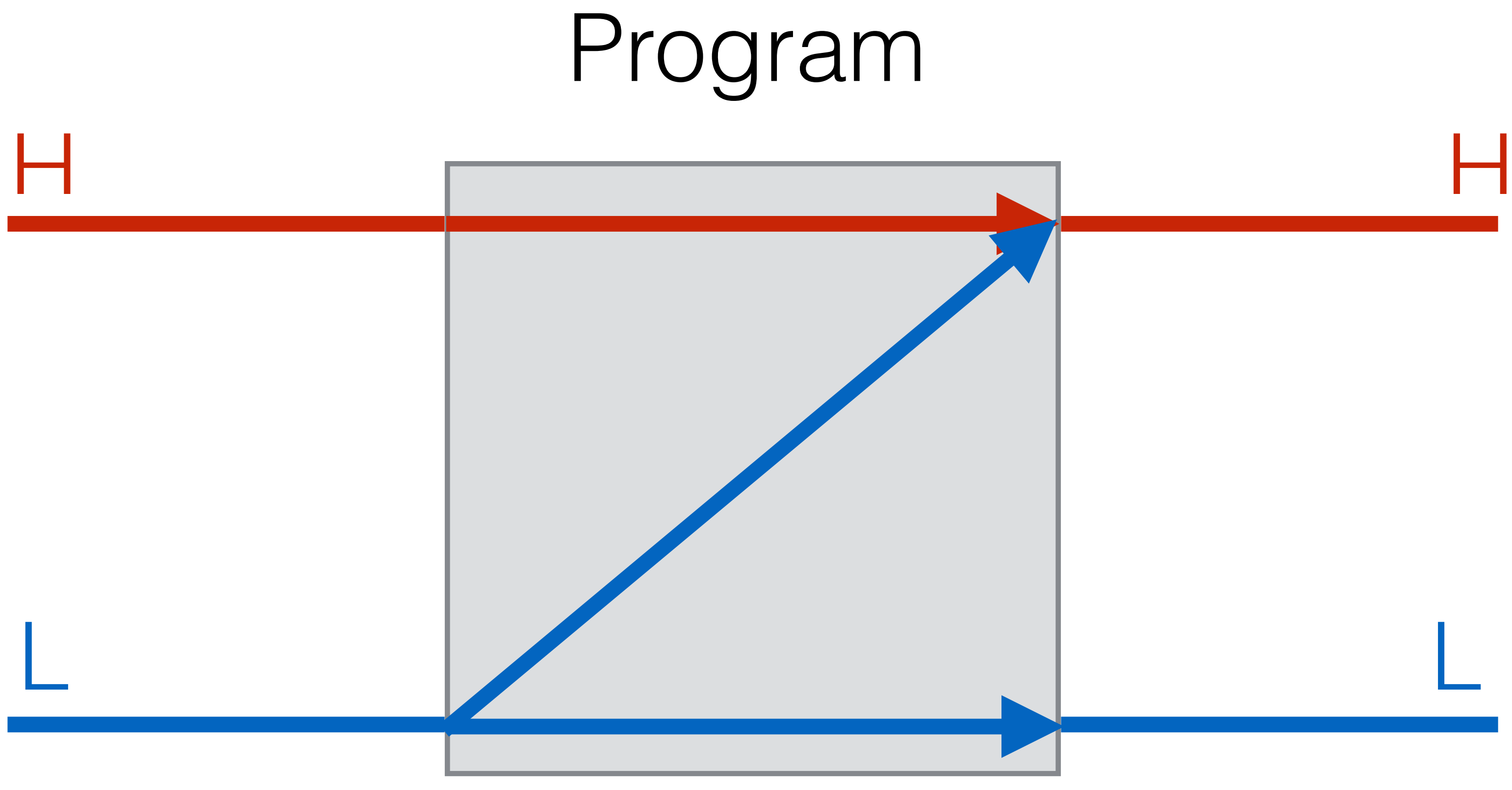
Verifying and **Synthesizing Constant-Resource** Implementations with **Types**

Van Chan Ngo
Mario Dehesa-Azuara
Matt Fredrikson
Jan Hoffmann

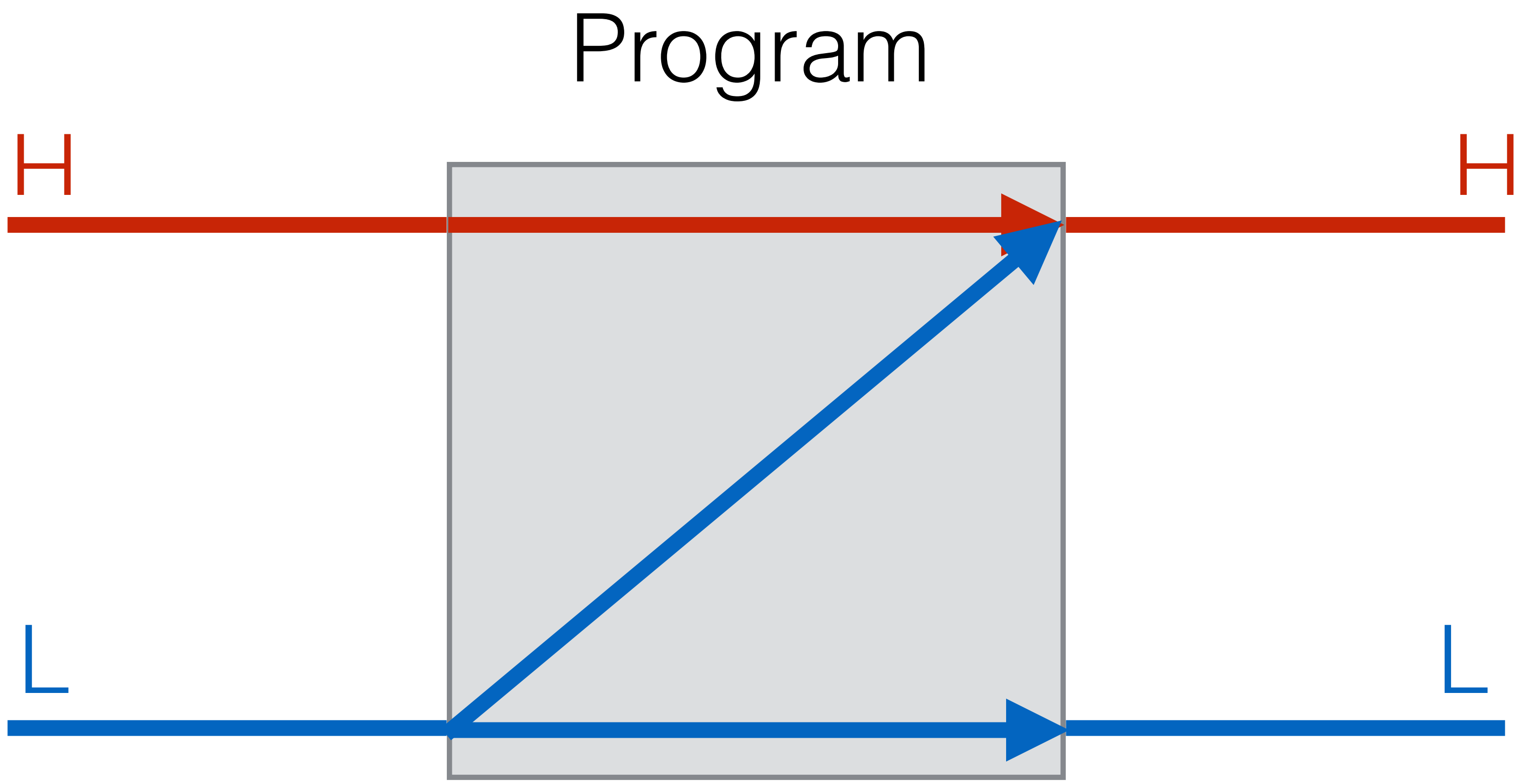
Carnegie Mellon University

S&P Oakland 2017

Classic non-interference

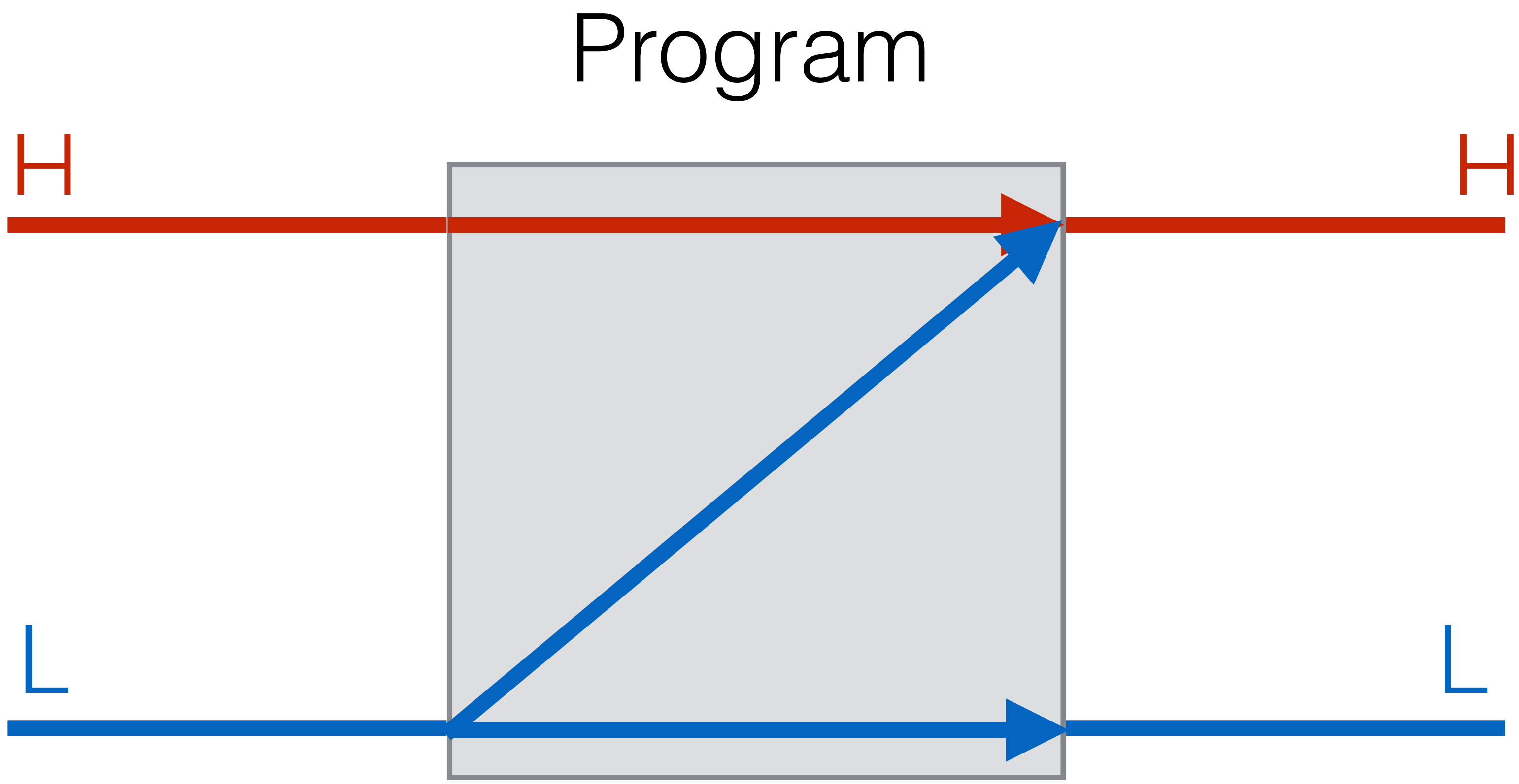


Classic non-interference



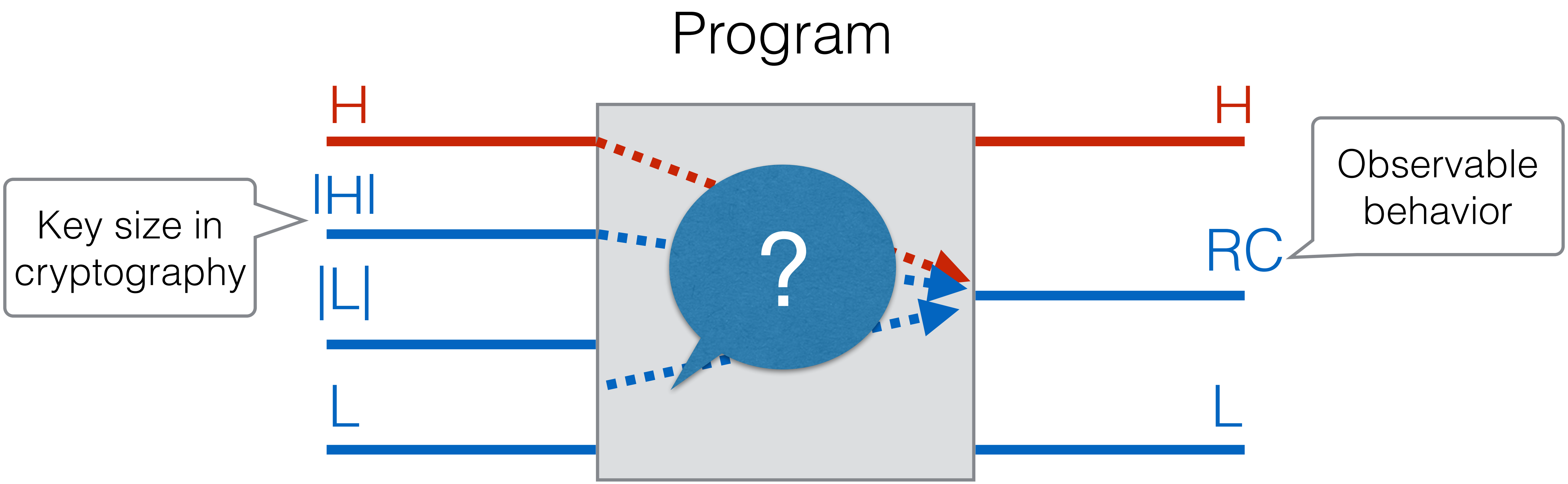
- No high security (H) flows to low security (L)

Classic non-interference

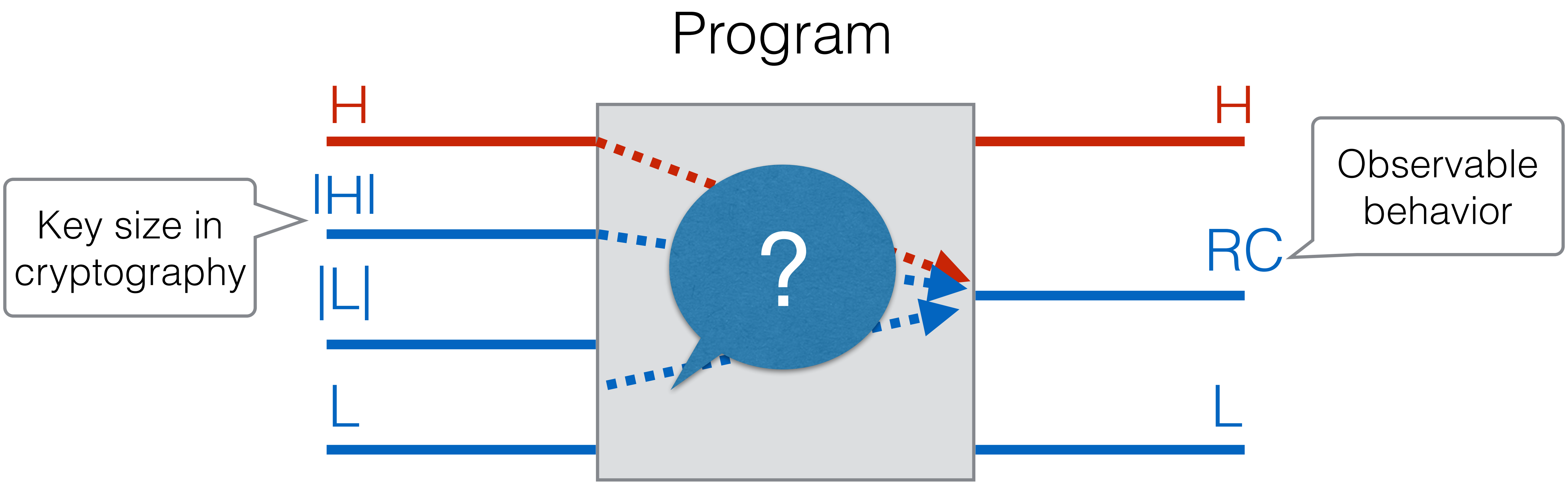


- No high security (H) flows to low security (L)
- High security does not affect low security

Flow to resource consumption

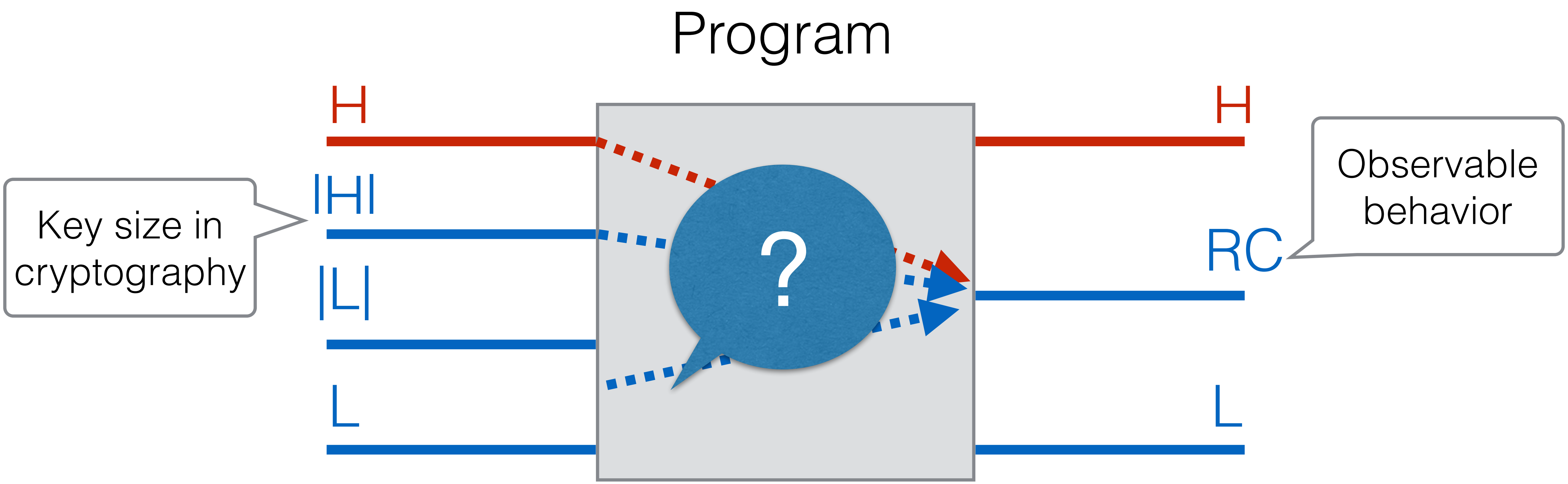


Flow to resource consumption



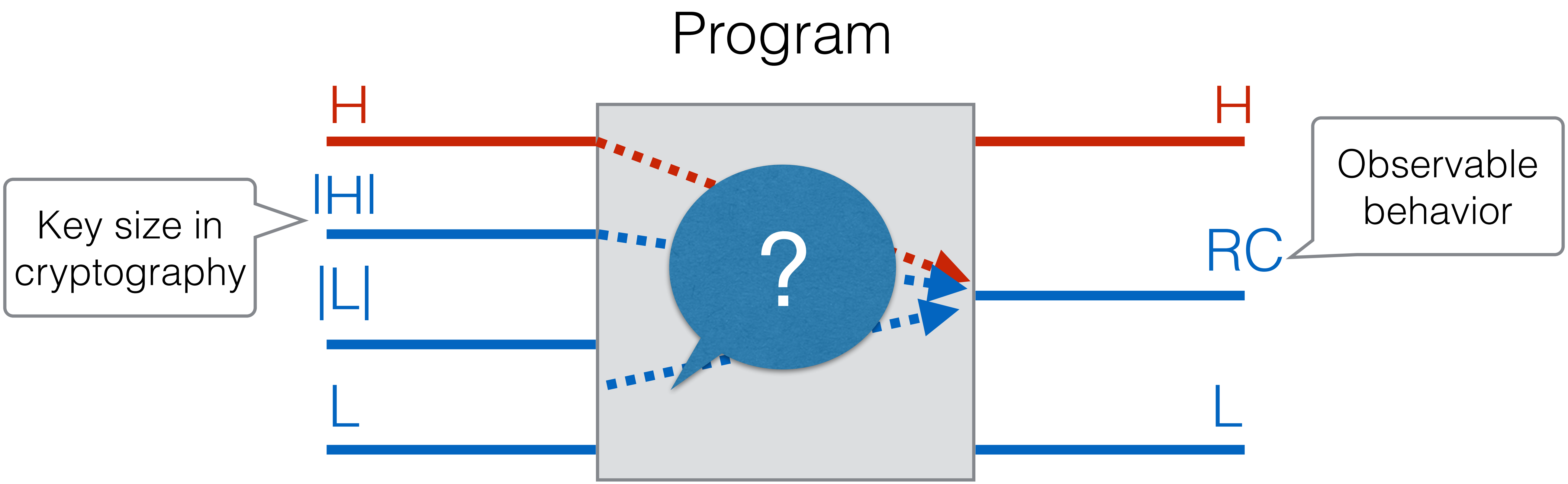
- Sizes of H (|H|) and L (|L|) and resource consumption (RC) are low securities

Flow to resource consumption



- Sizes of H (|H|) and L (|L|) and resource consumption (RC) are low securities
- Nothing about the information flow to RC

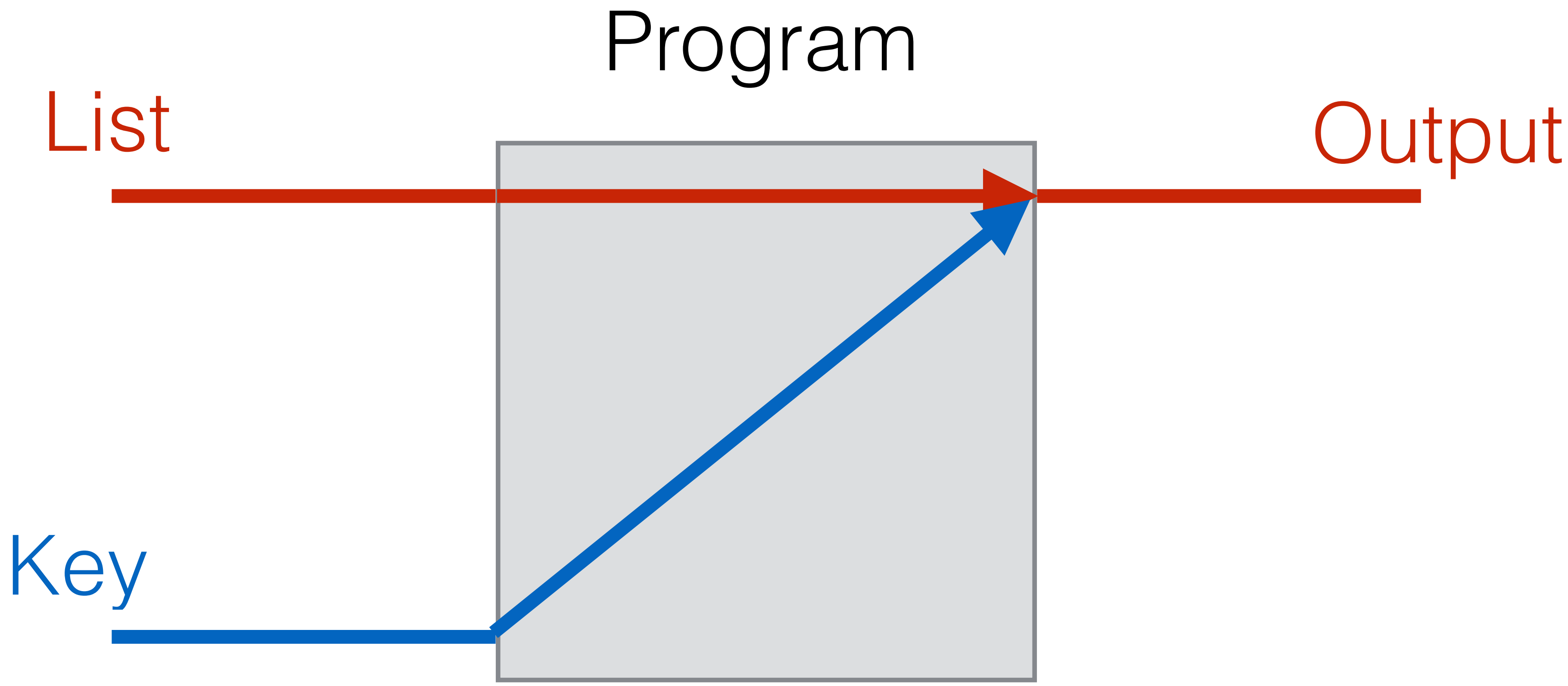
Flow to resource consumption



- Sizes of H (|H|) and L (|L|) and resource consumption (RC) are low securities
- Nothing about the information flow to RC
  - Can H and |H| flow to RC (affect RC) ?

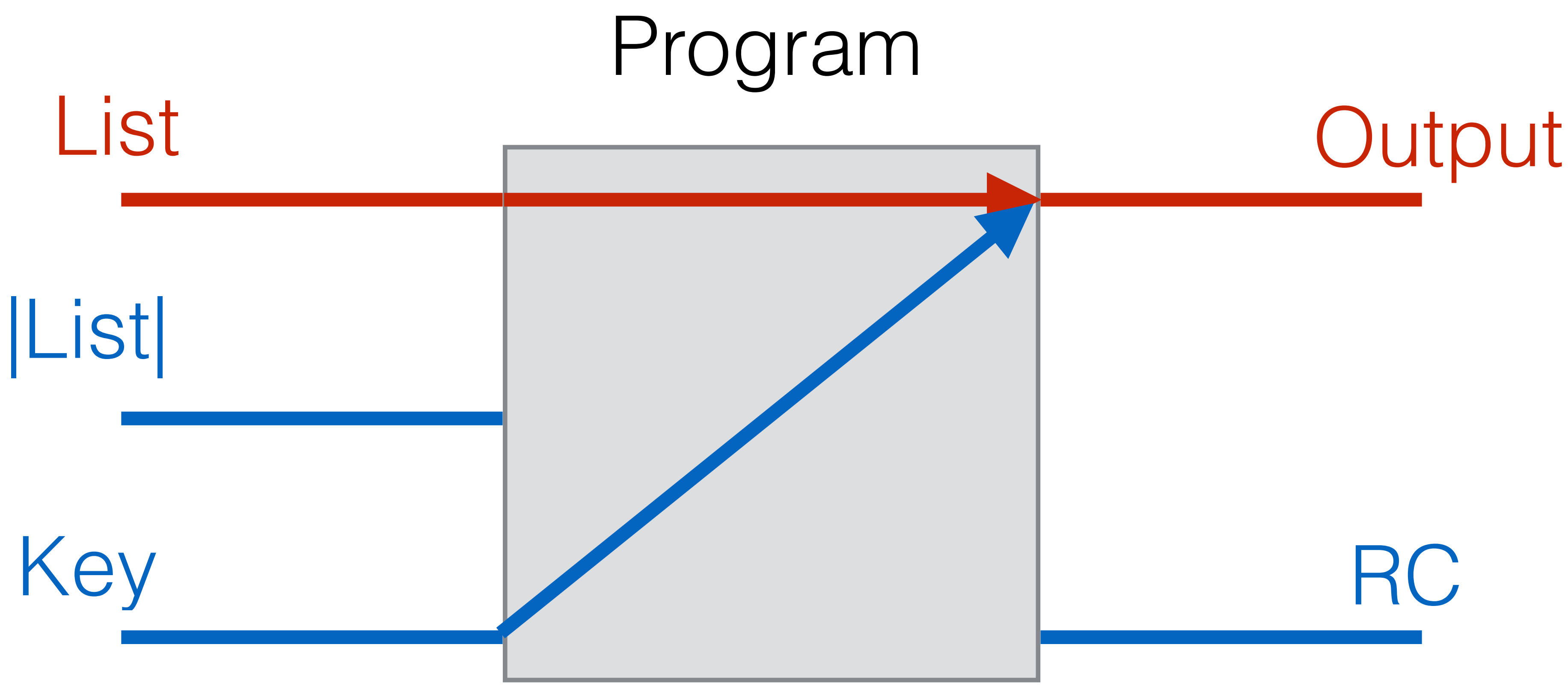
Example: Sequential search

Checks sequentially the list of items until a match for the key is found



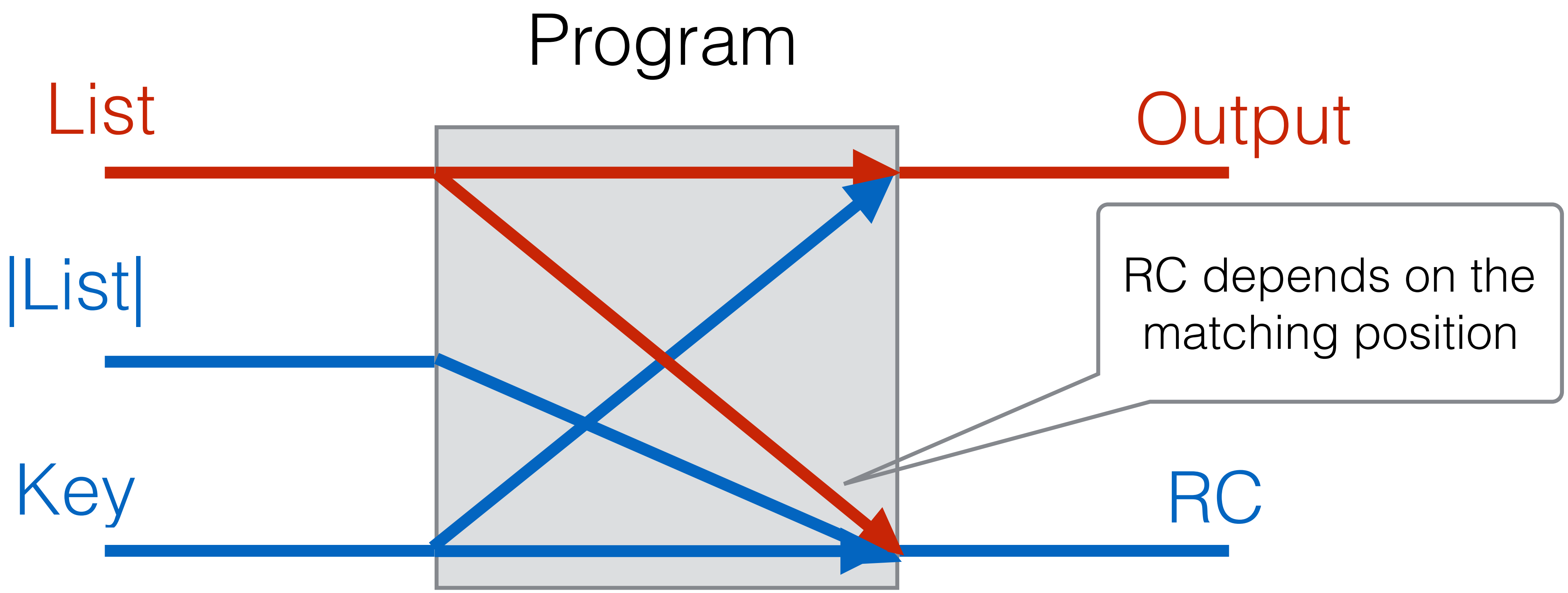
Example: Sequential search

Checks sequentially the list of items until a match for the key is found



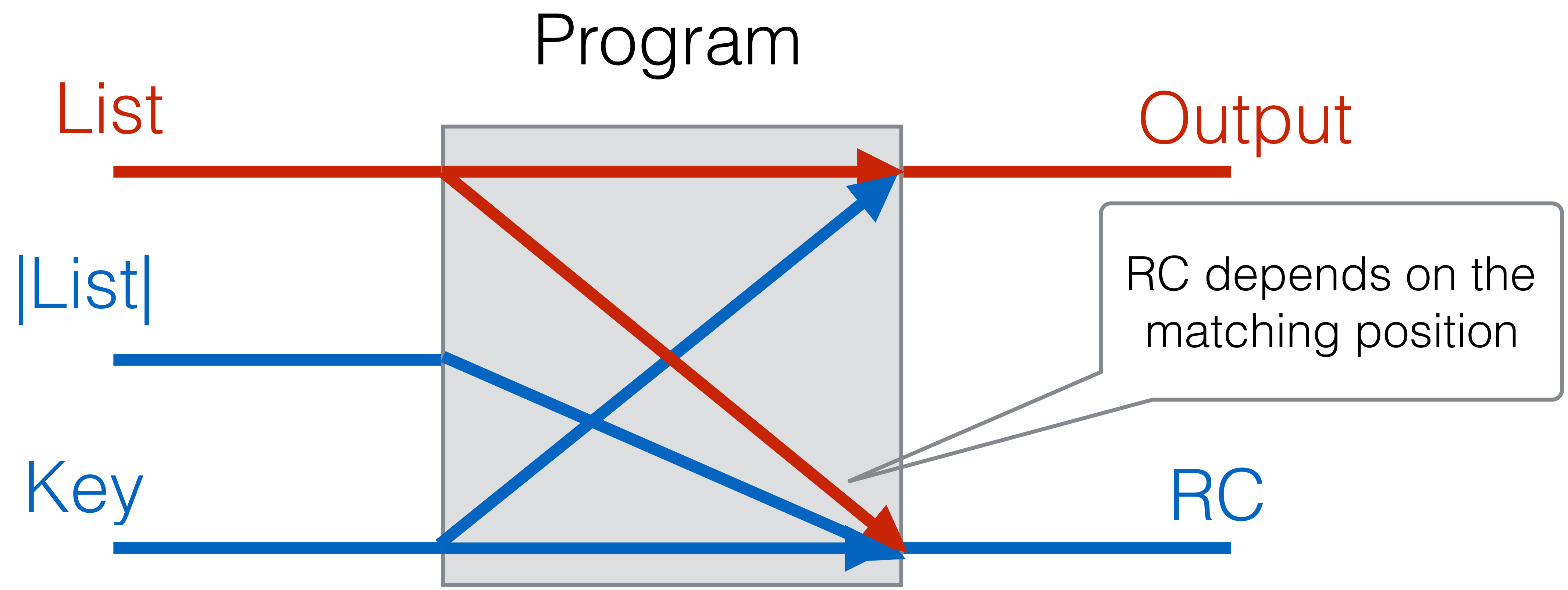
Example: Sequential search

Checks sequentially the list of items until a match for the key is found



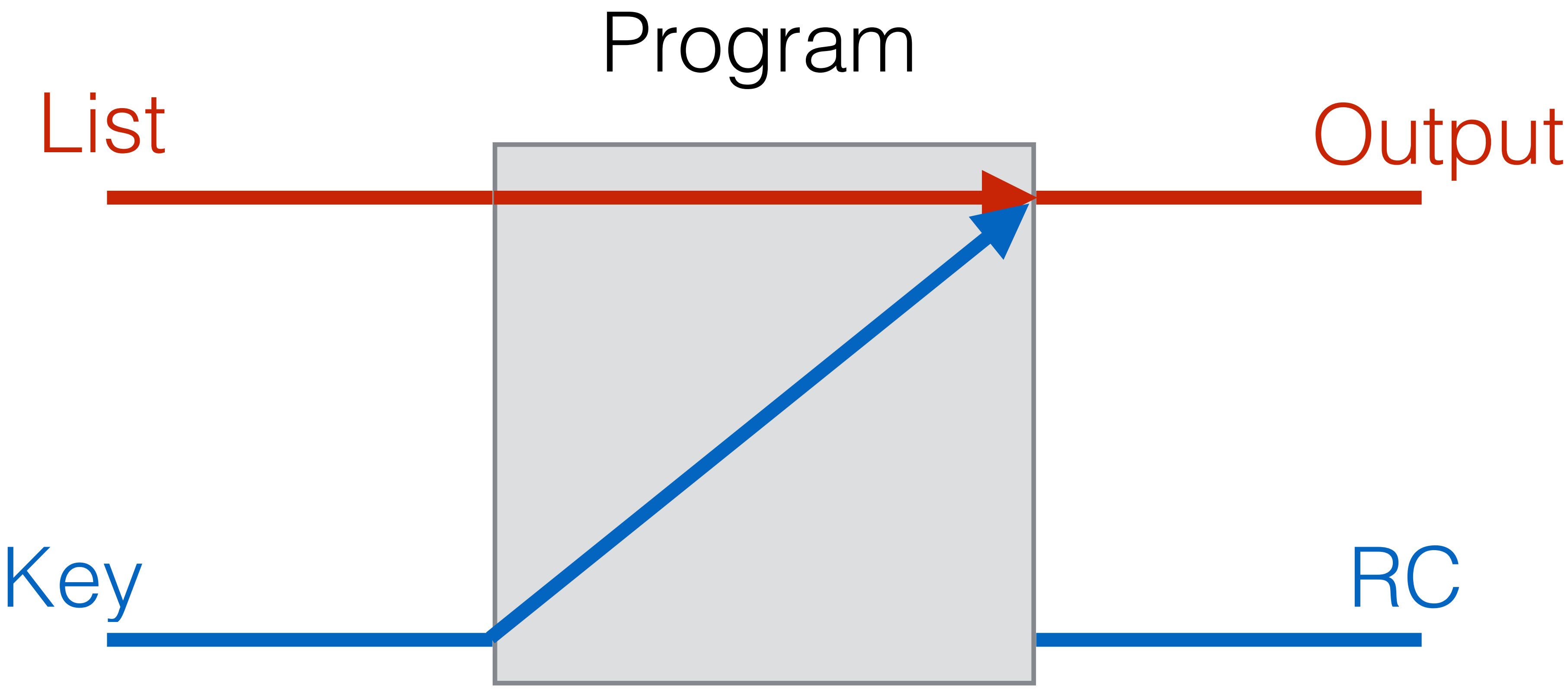
Example: Sequential search

Checks sequentially the list of items until a match for the key is found

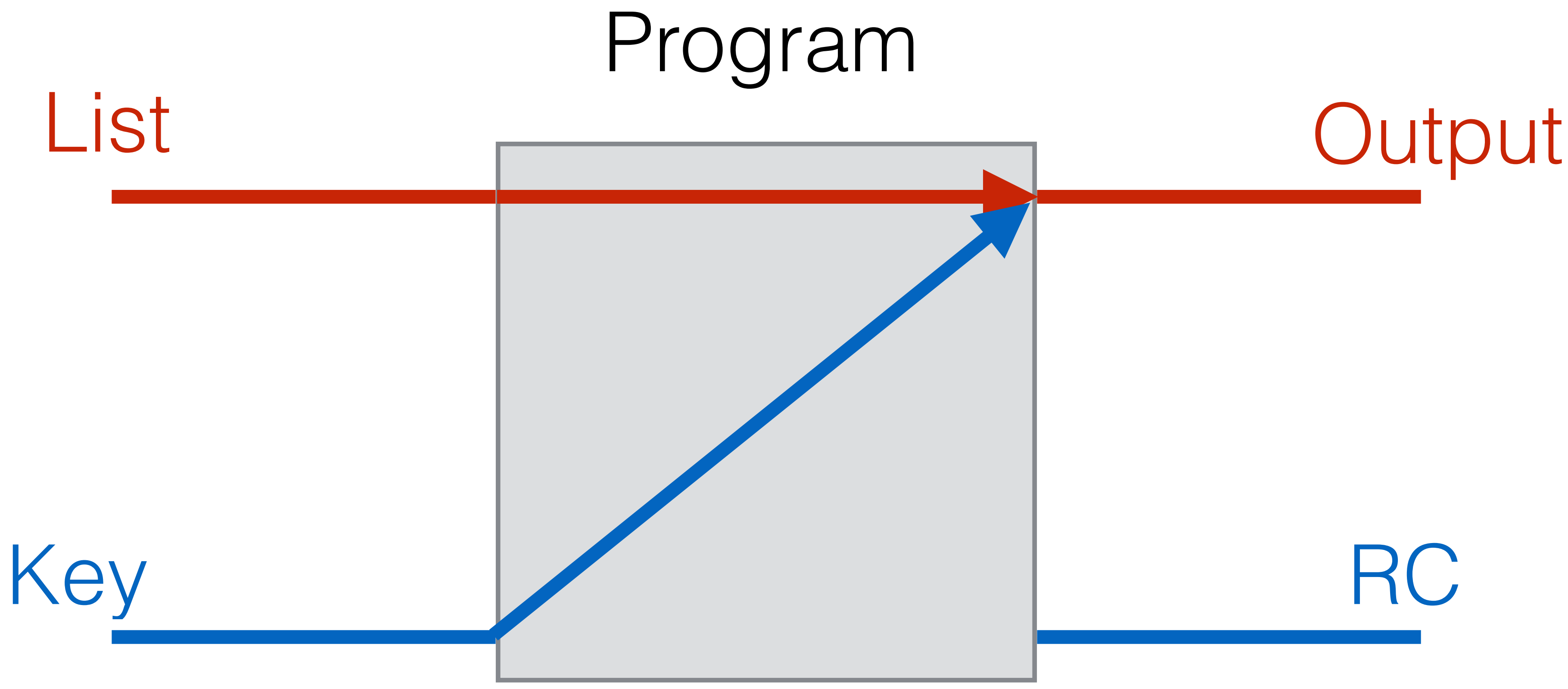


- **Problem**: High security (List) affects low security (RC)
- **Side channel attacks**: By observing RC, List can be learned

Resource-aware non-interference

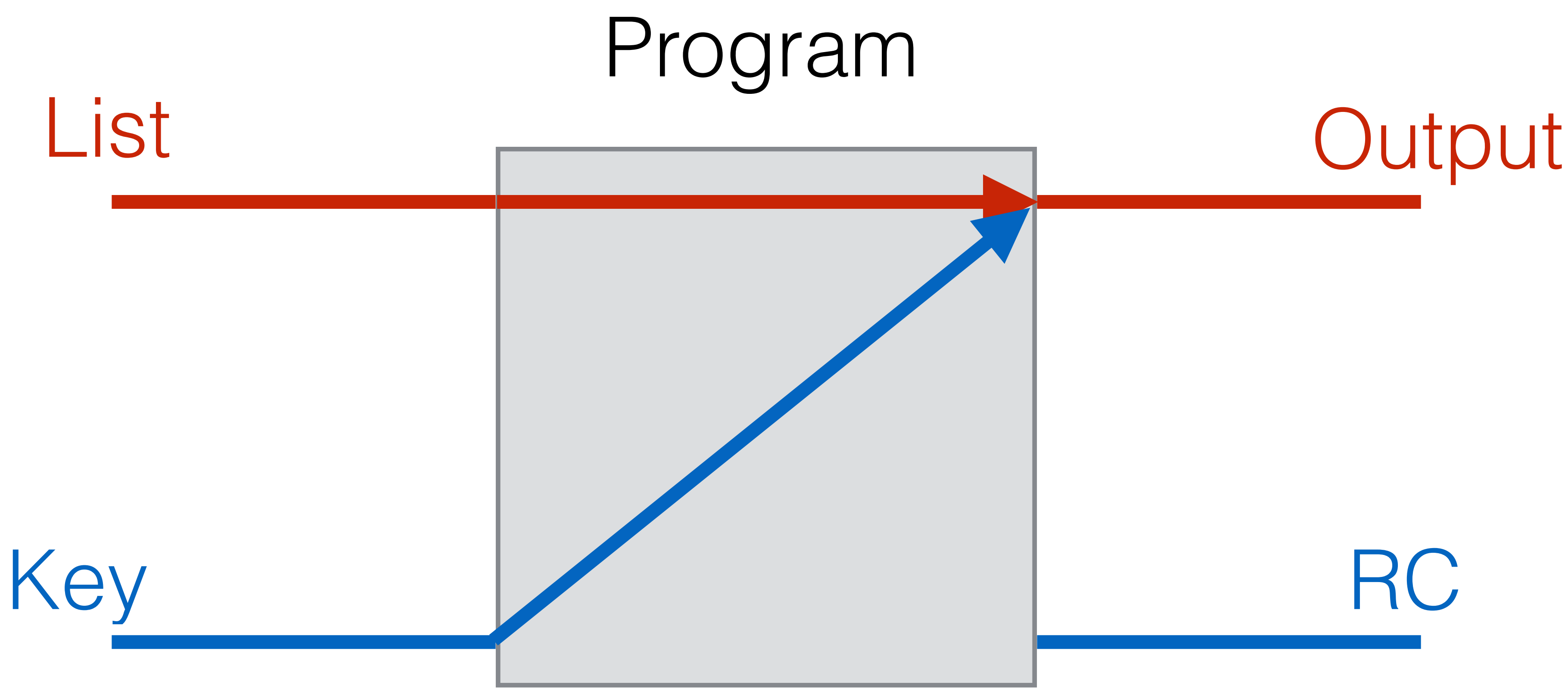


Resource-aware non-interference



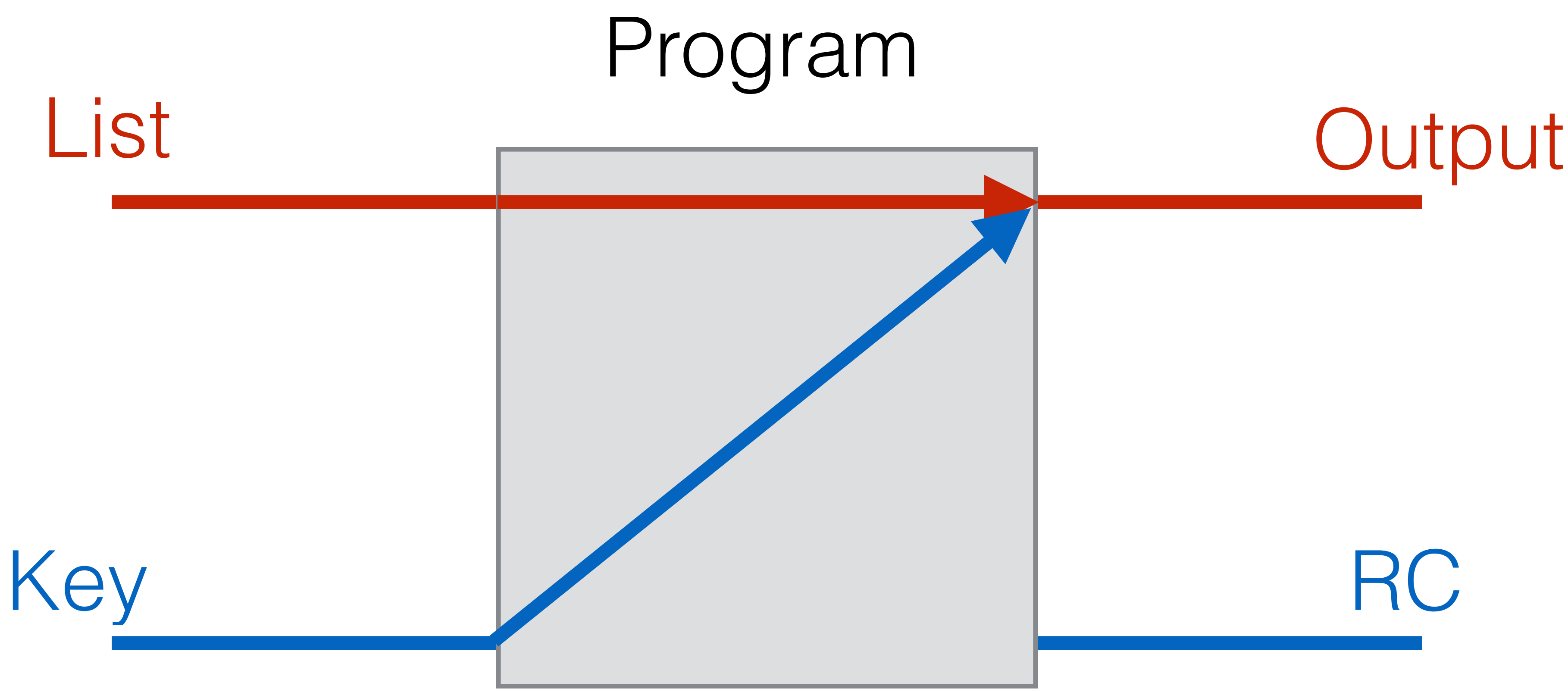
- No high security (H) flows to low security (L)

Resource-aware non-interference



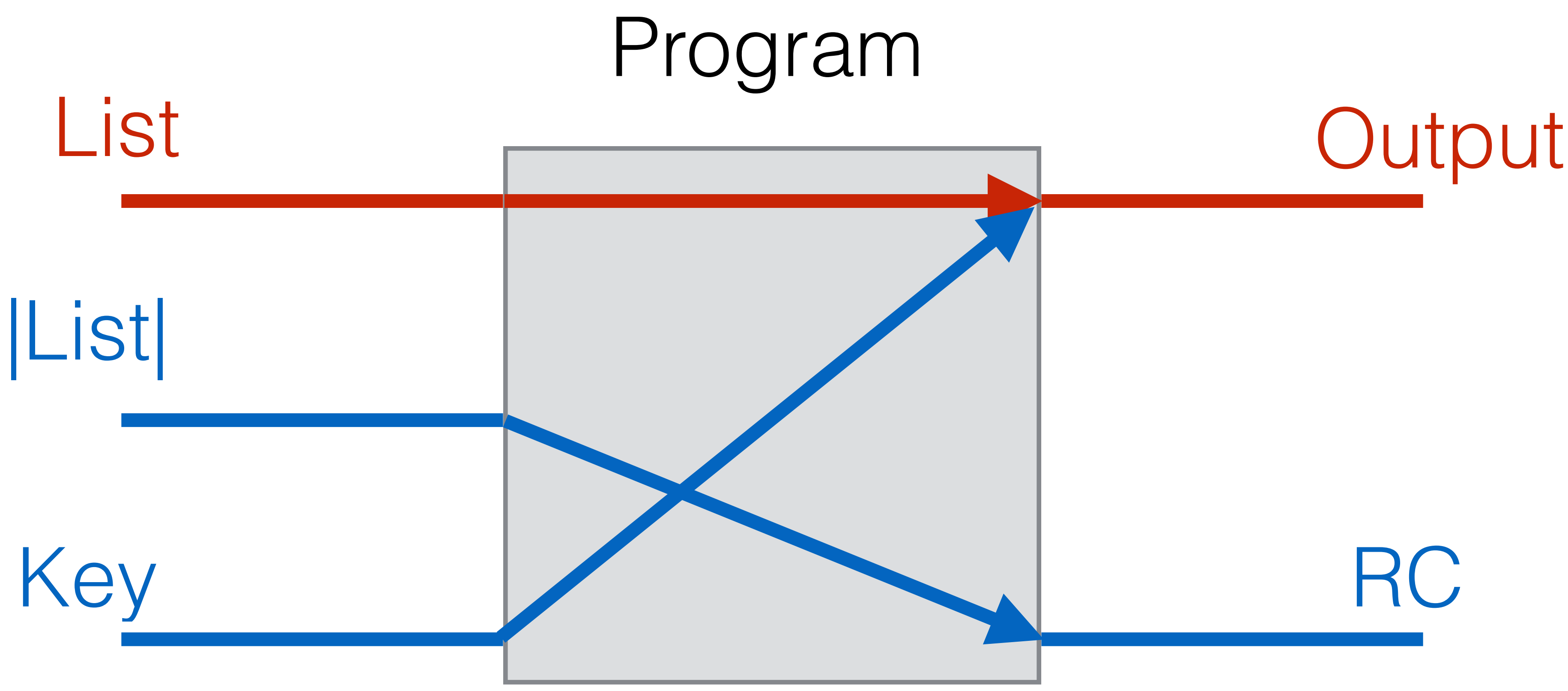
- No high security (H) flows to low security (L)
- All executions where **sizes of high securities are fixed** produce (total) constant-resource consumption (Observing RC tells nothing about Hs)

Resource-aware non-interference



- No high security (H) flows to low security (L)
- All executions where **sizes of high securities are fixed** produce (total) constant-resource consumption (Observing RC tells nothing about Hs)
- Thus, only sizes of Hs affect RC

Resource-aware non-interference



- No high security (H) flows to low security (L)
- All executions where **sizes of high securities are fixed** produce (total) constant-resource consumption (Observing RC tells nothing about Hs)
- Thus, only sizes of Hs affect RC

Resource-aware sequential search

```
let find(k,l) =
  let rec aux(k,l,res) =
    match l with
    | [] -> res
    | x::xs ->
      if x = k then
        tick(1); aux(k,xs,true)
      else
        tick(1); aux(k,xs,res)
  in aux(k,l,false)
```

Resource-aware sequential search

```
let find(k,l) =
  let rec aux(k,l,res) =
    match l with
    | [] -> res
    | x::xs ->
      if x = k then
        tick(1); aux(k,xs,true)
      else
        tick(1); aux(k,xs,res)
  in aux(k,l,false)
```

Metric: function call

Resource-aware sequential search

Sequentially checks all items, total # of functional calls is length(l)

```
let find(k,l) =
  let rec aux(k,l,res) =
    match l with
    | [] -> res
    | x::xs ->
      if x = k then
        tick(1); aux(k,xs,true)
      else
        tick(1); aux(k,xs,res)
  in aux(k,l,false)
```

Metric: function call

Resource-aware sequential search

Well-typed program

Sequentially checks all items, total # of functional calls is length(l)

```
let find(k,l) =
  let rec aux(k,l,res) =
    match l with
    | [] -> res
    | x::xs ->
      if x = k then
        tick(1); aux(k,xs,true)
      else
        tick(1); aux(k,xs,res)
  in aux(k,l,false)
```

Metric: function call

Solution: Quantitative language-based approach

Solution: Quantitative language-based approach

- **Resource type system** proves that resource consumption is constant if input sizes are fixed

Solution: Quantitative language-based approach

- **Resource type system** proves that resource consumption is constant if input sizes are fixed
- **Security type system** co-operating with resource type system enforces resource-aware non-interference

Solution: Quantitative language-based approach

- **Resource type system** proves that resource consumption is constant if input sizes are fixed
- **Security type system** co-operating with resource type system enforces resource-aware non-interference
- Quantification of information leakage of non-constant-resource programs

Solution: Quantitative language-based approach

- **Resource type system** proves that resource consumption is constant if input sizes are fixed
- **Security type system** co-operating with resource type system enforces resource-aware non-interference
- Quantification of information leakage of non-constant-resource programs
- Interactive and automatic program repair

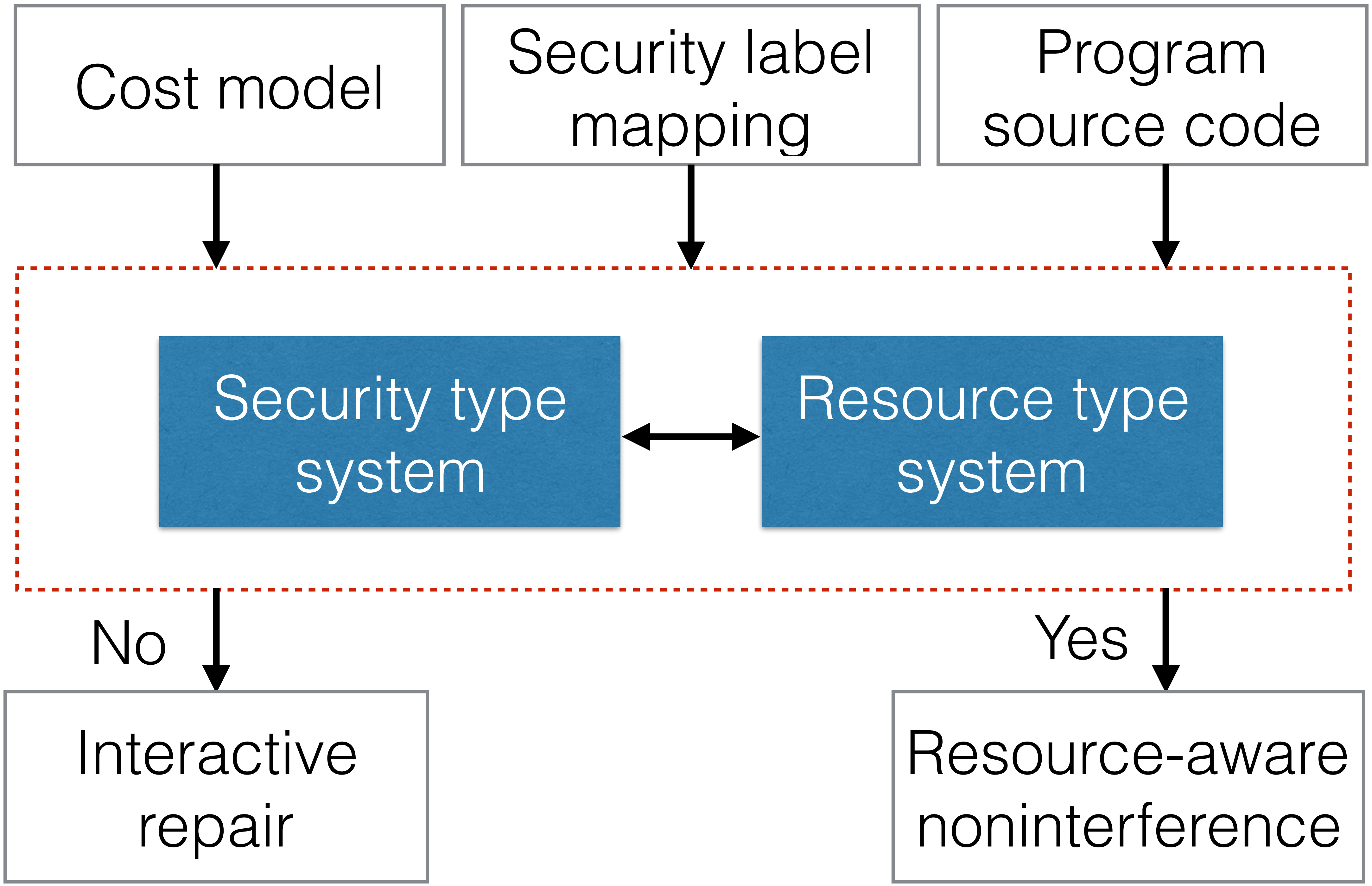
Solution: Quantitative language-based approach

- **Resource type system** proves
  consumption is constant if in

Focus of this talk

- **Security type system** co-operating with resource type system enforces resource-aware non-interference
- Quantification of information leakage of non-constant-resource programs
- Interactive and automatic program repair

Overview



Cost model
Security label mapping
Program source code
Security type system
Resource type system
No
Yes
Interactive repair
Resource-aware noninterference

Security type system

Judgement:

$$\mathrm{pc}, \Gamma \vdash^{\mathrm{const}} e : S$$

$$\mathrm{pc}, \Gamma \vdash e : S$$

- Under security setting Γ and pc, e has type S and it has non-interference property
- If judgements have const annotations then e satisfies resource-aware non-interference

Security type system

Program counter

Judgement: $\mathrm{pc}, \Gamma \vdash^{\mathrm{const}} e : S$

$$\mathrm{pc}, \Gamma \vdash e : S$$

- Under security setting Γ and pc, e has type S and it has non-interference property
- If judgements have const annotations then e satisfies resource-aware non-interference

Security type system

Program counter

Security context

Judgement: $\mathrm{pc}, \Gamma \vdash^{\mathrm{const}} e : S$

$$\mathrm{pc}, \Gamma \vdash e : S$$

- Under security setting Γ and pc, e has type S and it has non-interference property
- If judgements have const annotations then e satisfies resource-aware non-interference

Security type system

Program counter

Security context

Judgement: $\mathrm{pc}, \Gamma \vdash^{\mathrm{const}} e : S$

$$\mathrm{pc}, \Gamma \vdash e : S$$

Expression security type (annotated with security labels)

- Under security setting Γ and pc, e has type S and it has non-interference property
- If judgements have const annotations then e satisfies resource-aware non-interference

Security type system

Program counter

Security context

Indicate resource-aware noninterference

Judgement:

$$\mathrm{pc}, \Gamma \vdash^{\mathrm{const}} e : S$$

$$\mathrm{pc}, \Gamma \vdash e : S$$

Expression security type (annotated with security labels)

- Under security setting Γ and pc, e has type S and it has non-interference property
- If judgements have const annotations then e satisfies resource-aware non-interference

Enforcing resource-aware non-interference

Enforcing resource-aware non-interference

- Two extreme ways: **globally** and **locally** enforcing

Enforcing resource-aware non-interference

- Two extreme ways: **globally** and **locally** enforcing
- Global reasoning: using the resource type system to check the whole program is constant-resource

Enforcing resource-aware non-interference

- Two extreme ways: **globally** and **locally** enforcing
- Global reasoning: using the resource type system to check the whole program is constant-resource
    - *Sound but requires to reason about parts **not affected** by high securities*

Enforcing resource-aware non-interference

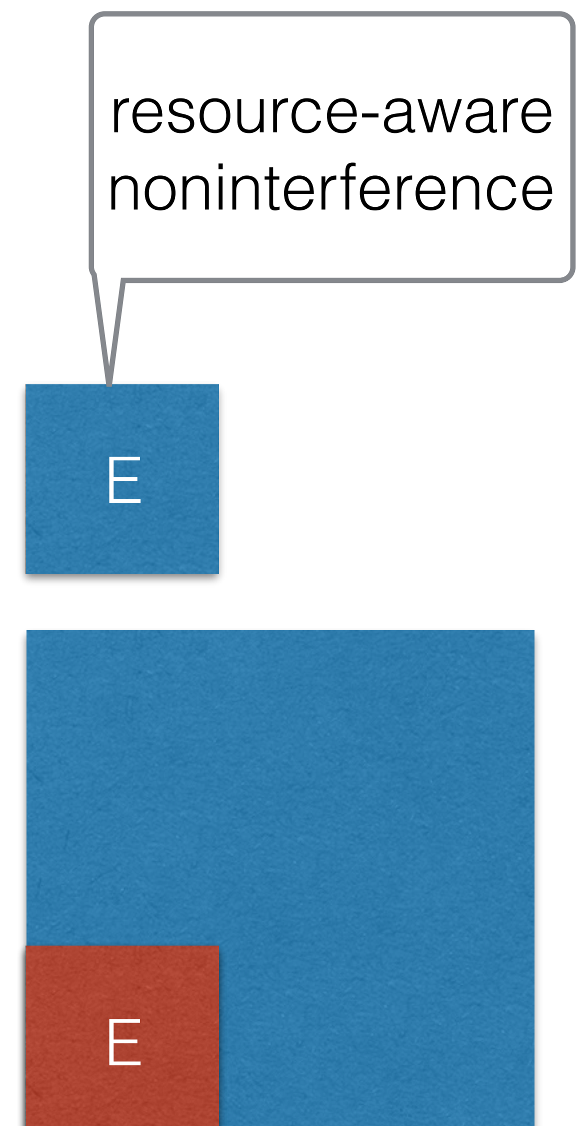
- Two extreme ways: **globally** and **locally** enforcing
- Global reasoning: using the resource type system to check the whole program is constant-resource
  - *Sound but requires to reason about parts **not affected** by high securities*
- Local reasoning: ensuring every condition branching on high security is constant-resource

Enforcing resource-aware non-interference

- Two extreme ways: **globally** and **locally** enforcing
- Global reasoning: using the resource type system to check the whole program is constant-resource
  - *Sound but requires to reason about parts **not affected** by high securities*
- Local reasoning: ensuring every condition branching on high security is constant-resource
  - *Not sufficient and efficient (**rejects valid programs**)*

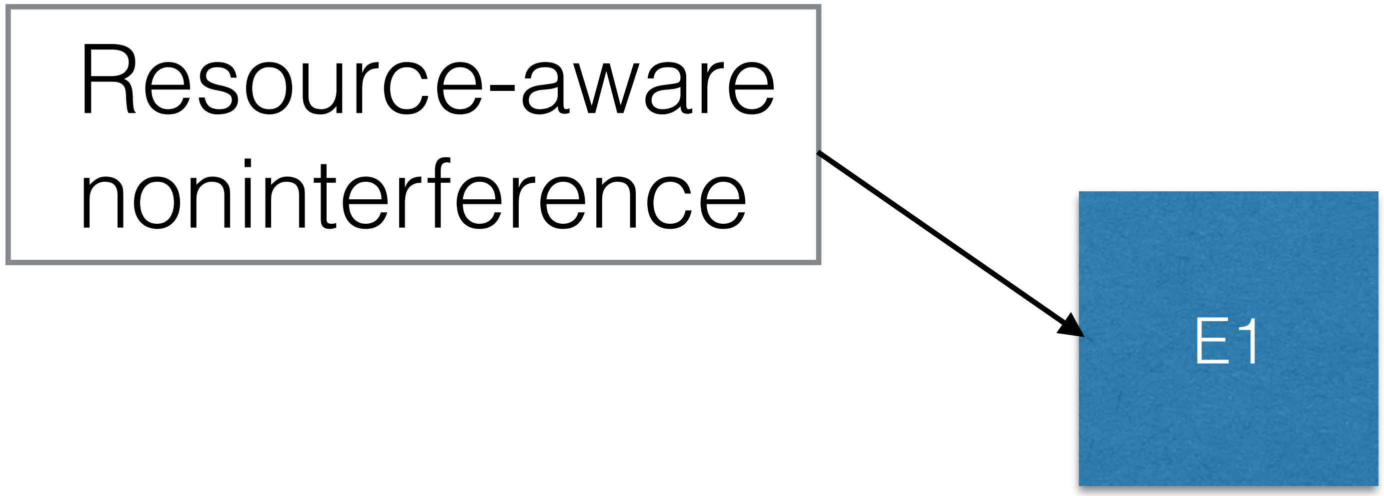
Global and local reasoning

- Security type system uses a mix of **global** and **local reasoning**
- Ensure that every expression affected by high security is
  - a resource-aware non-interference, or
  - a part of a resource-aware non-interference (Total resource usage is constant)

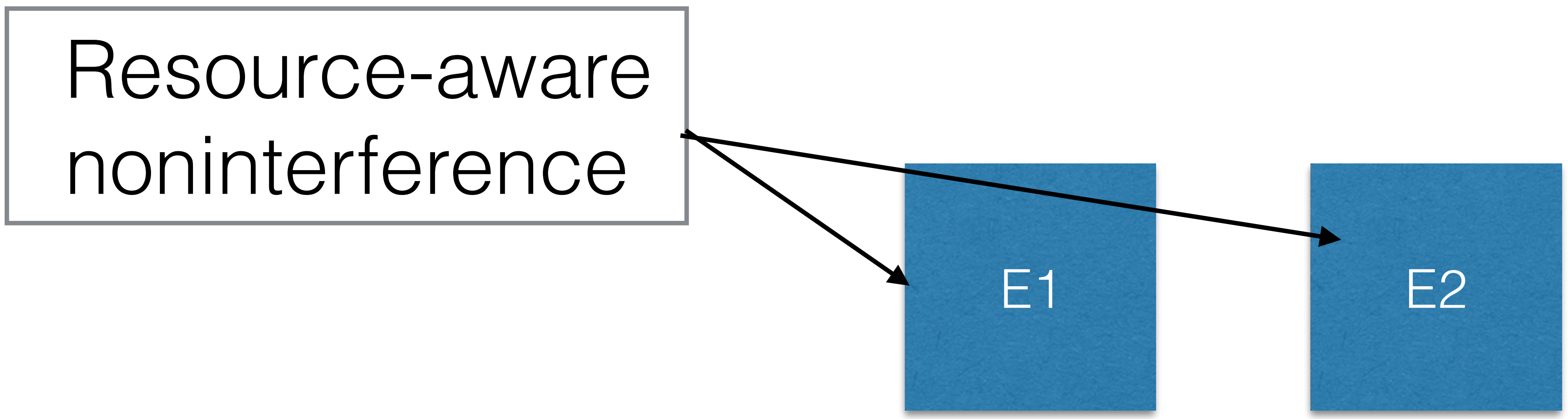


Local reasoning

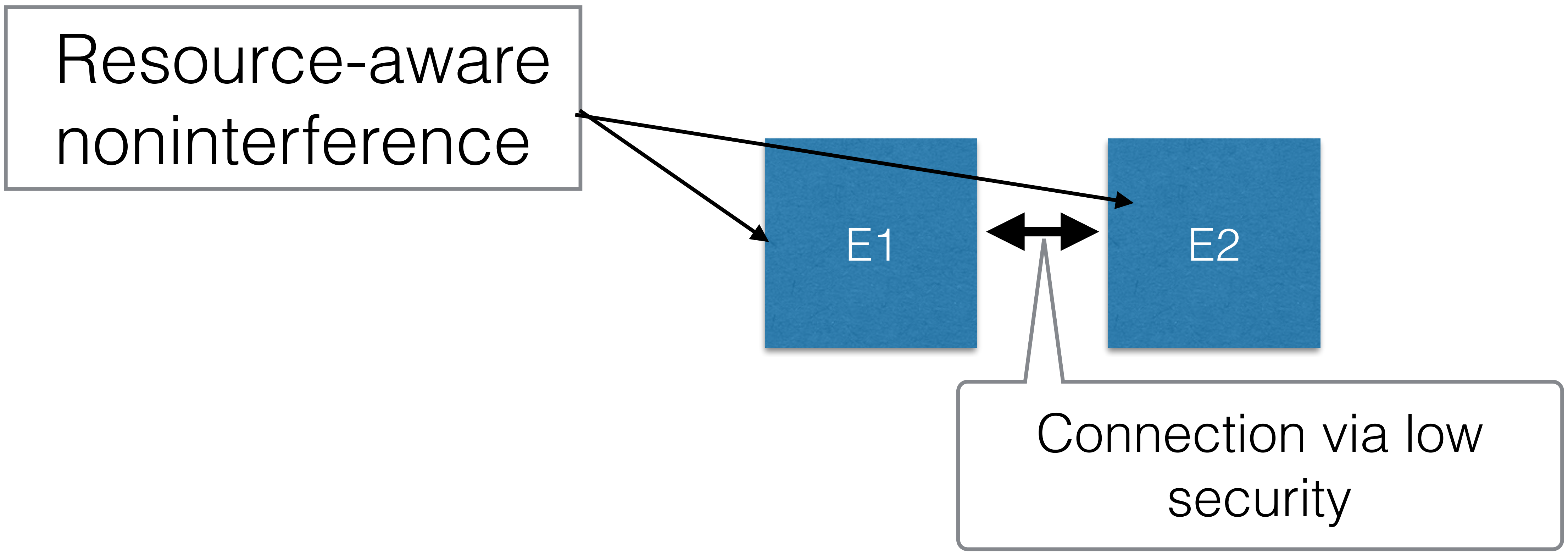
Local reasoning



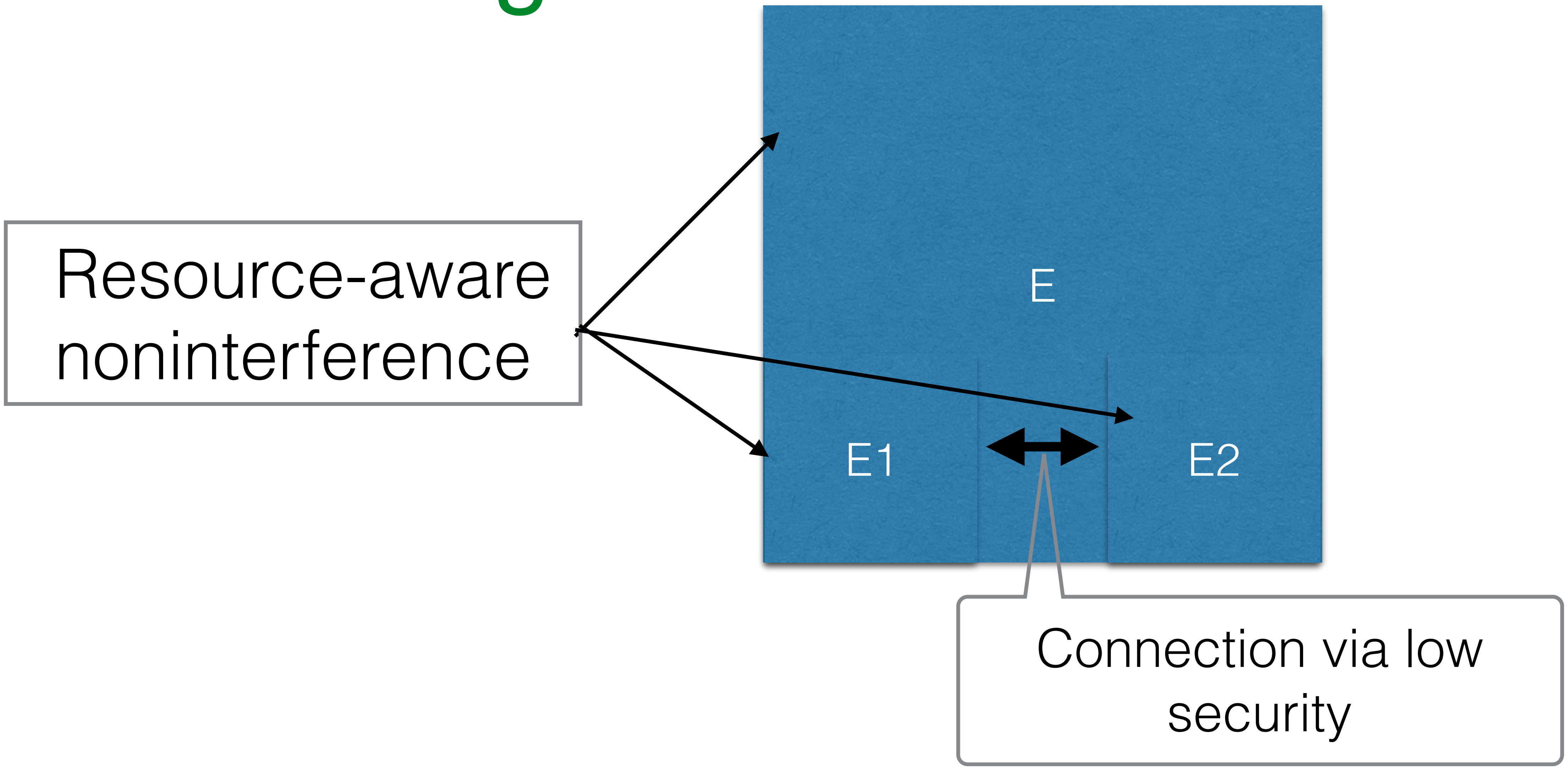
Local reasoning



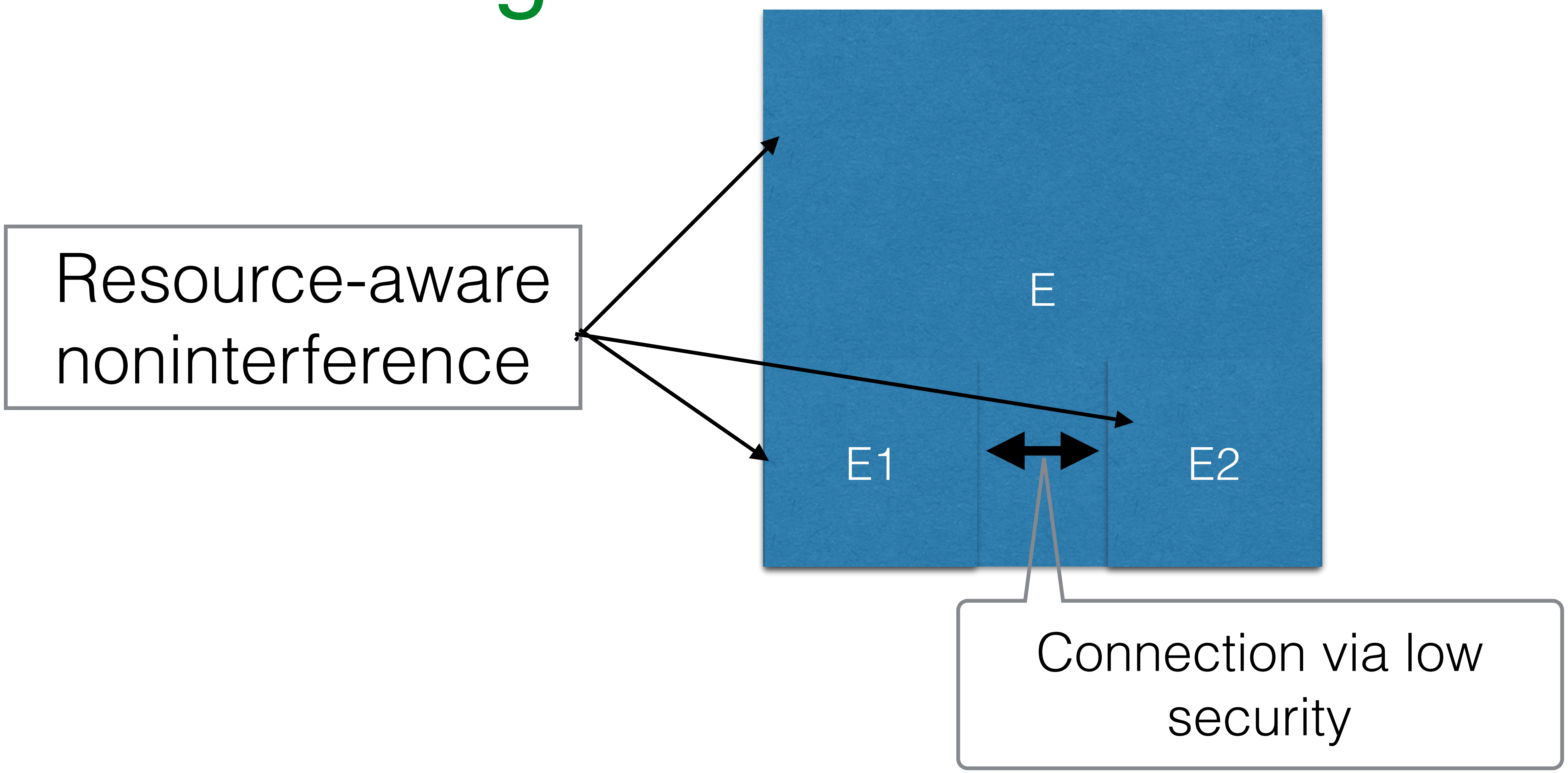
Local reasoning



Local reasoning



Local reasoning



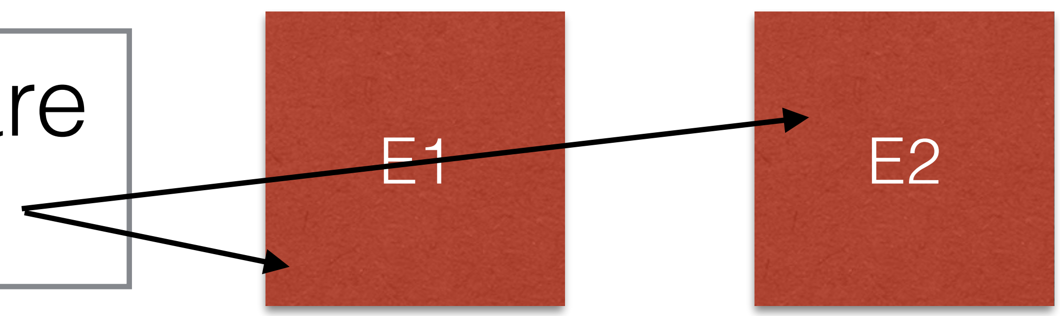
Example: rule for low security condition

$$\frac{\cdots \overset{\text{const}}{\vdash} e_t : S \qquad \cdots \overset{\text{const}}{\vdash} e_f : S \qquad k_x \sqsubseteq h}{\cdots \overset{\text{const}}{\vdash} \text{if}(x, e_t, e_f) : S}$$

Global reasoning

Global reasoning

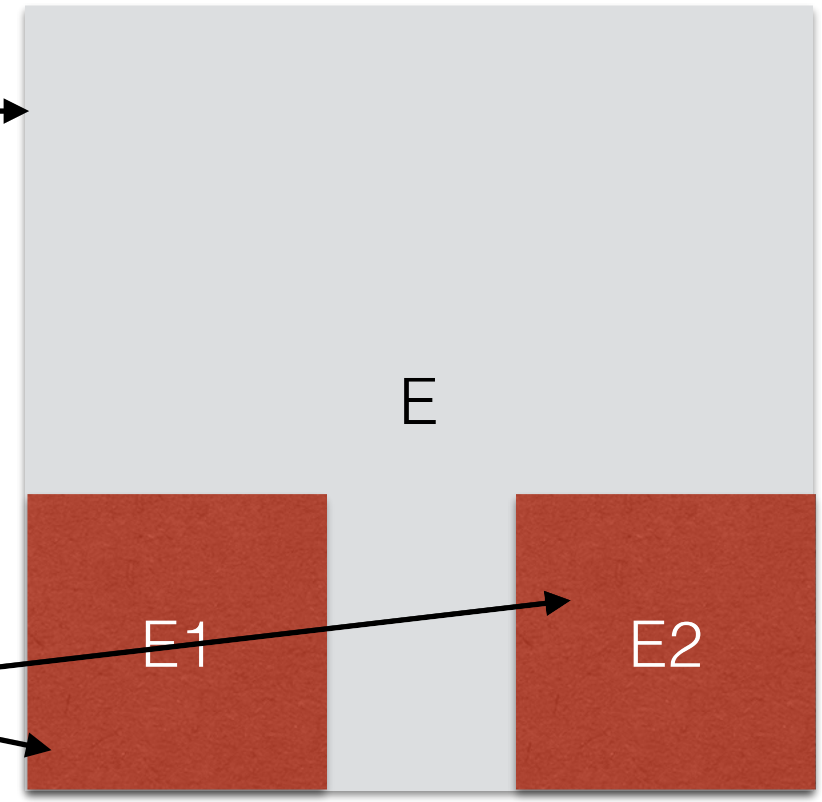
Not have resource-aware noninterference



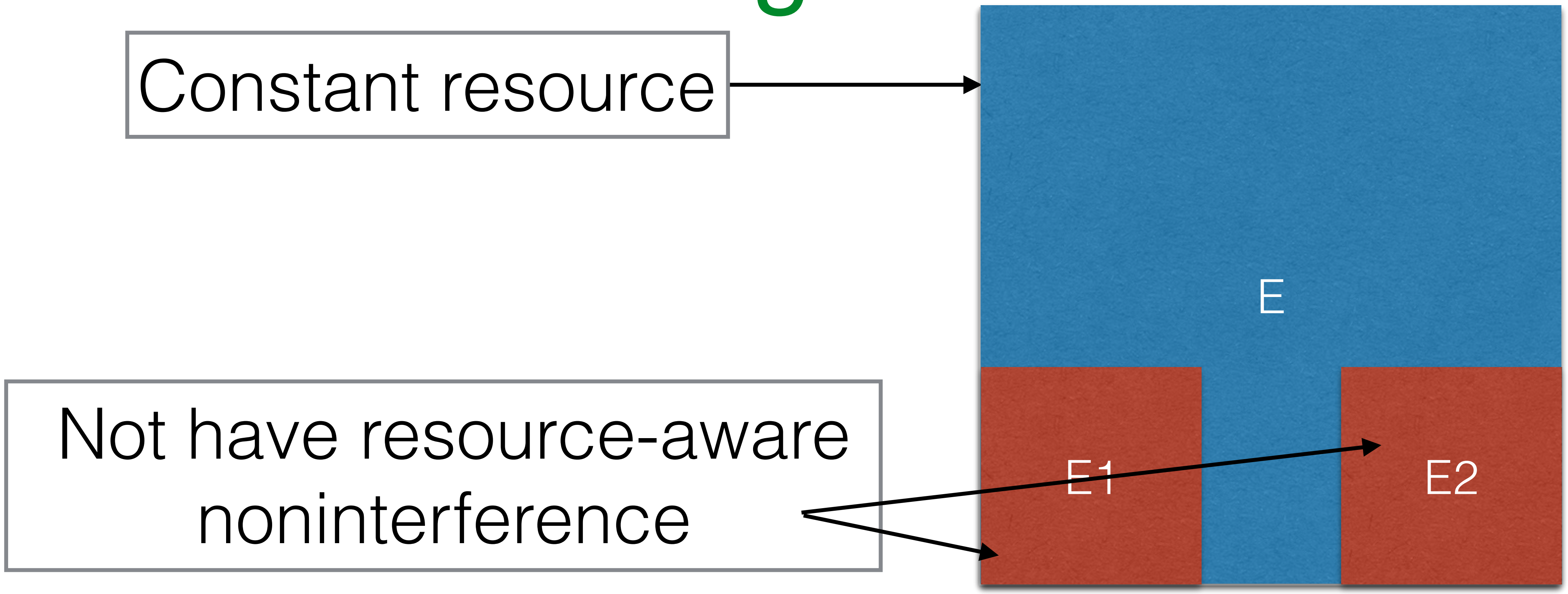
Global reasoning

Constant resource

Not have resource-aware noninterference



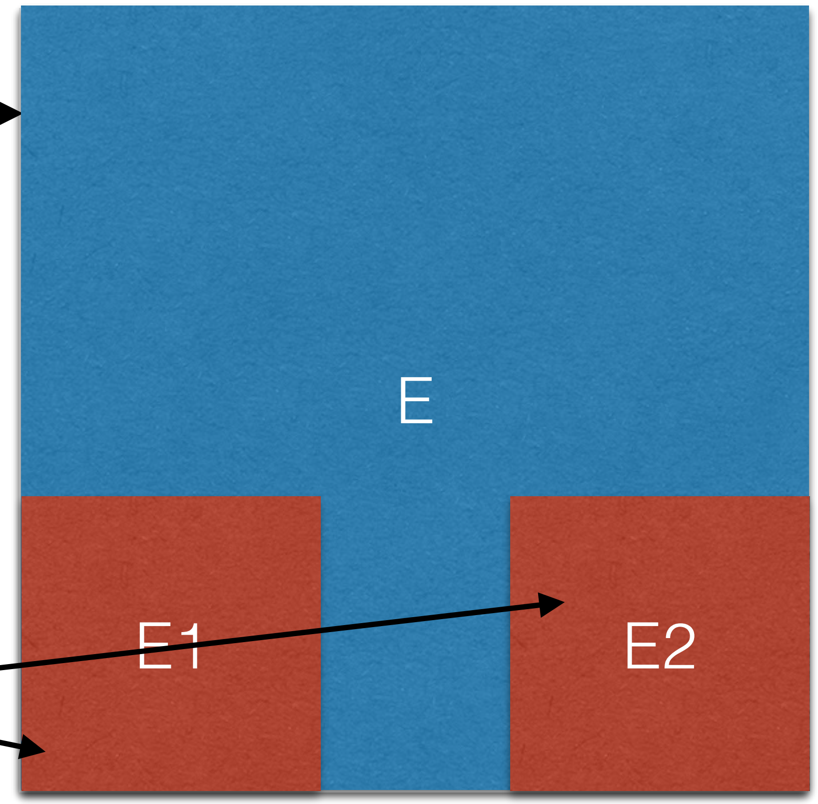
Global reasoning



Global reasoning

Constant resource

Not have resource-aware noninterference



Example: rule for cooperating with resource type system

$$\frac{\cdots \vdash e : S \qquad \mathbf{const}(e)}{\cdots \vdash^{\mathbf{const}} e : S}$$

Proving constant-resource

Affine resource type system

Relevant resource type system

Can waste potential	Worst-case resource usage
Can create potential	Best-case resource usage
Cannot waste or create potential	Exact resource usage

Linear resource type system

- Based on the existing type system using potential method (potential encoded in program state to ‘pay’ resource consumption)
- If final potential is zero then initial potential gives the constant-resource usage

Evaluation

Constant Function	LOC	Metric	Resource Usage	Time
cond_rev : $(L(\text{int}), L(\text{int}), \text{bool}) \to \text{unit}$	20	steps	$13n+13x+35$	0.03s
trunc_rev : $(L(\text{int}), \text{int}) \to L(\text{int})$	28	function calls	$1n$	0.06s
ipquery : $L(\text{logline}) \to (L(\text{int}), L(\text{int}))$	86	steps	$86n+99$	0.86s
kmeans : $L(\text{float, float}) \to L(\text{float, float})$	170	steps	$1246n+3784$	8.18s
tea_enc : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z+32nxz+1184nz+96n+128z+96$	13.73s
tea_dec : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z+32nxz+1184nz+96n+96z+96$	14.34s

Evaluation

Common primitive functions

Constant Function	LOC	Metric	Resource Usage	Time
cond_rev : $(L(\text{int}), L(\text{int}), \text{bool}) \to \text{unit}$	20	steps	$13n{+}13x{+}35$	0.03s
trunc_rev : $(L(\text{int}), \text{int}) \to L(\text{int})$	28	function calls	$1n$	0.06s
ipquery : $L(\text{logline}) \to (L(\text{int}), L(\text{int}))$	86	steps	$86n{+}99$	0.86s
kmeans : $L(\text{float}, \text{float}) \to L(\text{float}, \text{float})$	170	steps	$1246n{+}3784$	8.18s
tea_enc : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z{+}32nxz{+}1184nz{+}96n{+}128z{+}96$	13.73s
tea_dec : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z{+}32nxz{+}1184nz{+}96n{+}96z{+}96$	14.34s

Evaluation

Common primitive functions

Constant Function	LOC	Metric	Resource Usage	Time
cond_rev : $(L(\text{int}), L(\text{int}), \text{bool}) \to \text{unit}$	20	steps	$13n+13x+35$	0.03s
trunc_rev : $(L(\text{int}), \text{int}) \to L(\text{int})$	28	function calls	$1n$	0.06s
ipquery : $L(\text{logline}) \to (L(\text{int}), L(\text{int}))$	86	steps	$86n+99$	0.86s
kmeans : $L(\text{float}, \text{float}) \to L(\text{float}, \text{float})$	170	steps	$1246n+3784$	8.18s
tea_enc : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z+32nxz+1184nz+96n+128z+96$	13.73s
tea_dec : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z+32nxz+1184nz+96n+96z+96$	14.34s

Constant-time block encryption algorithm

Evaluation

Common primitive functions

Database query functions

Constant Function	LOC	Metric	Resource Usage	Time
cond_rev : $(L(\text{int}), L(\text{int}), \text{bool}) \to \text{unit}$	20	steps	$13n+13x+35$	0.03s
trunc_rev : $(L(\text{int}), \text{int}) \to L(\text{int})$	28	function calls	$1n$	0.06s
ipquery : $L(\text{logline}) \to (L(\text{int}), L(\text{int}))$	86	steps	$86n+99$	0.86s
kmeans : $L(\text{float}, \text{float}) \to L(\text{float}, \text{float})$	170	steps	$1246n+3784$	8.18s
tea_enc : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z+32nxz+1184nz+96n+128z+96$	13.73s
tea_dec : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z+32nxz+1184nz+96n+96z+96$	14.34s

Constant-time block encryption algorithm

Evaluation

Common primitive functions

Database query functions

Constant Function	LOC	Metric	Resource Usage	Time
cond_rev : $(L(\text{int}), L(\text{int}), \text{bool}) \to \text{unit}$	20	steps	$13n{+}13x{+}35$	0.03s
trunc_rev : $(L(\text{int}), \text{int}) \to L(\text{int})$	28	function calls	$1n$	0.06s
ipquery : $L(\text{logline}) \to (L(\text{int}), L(\text{int}))$	86	steps	$86n{+}99$	0.86s
kmeans : $L(\text{float}, \text{float}) \to L(\text{float}, \text{float})$	170	steps	$1246n{+}3784$	8.18s
tea_enc : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z{+}32nxz{+}1184nz{+}96n{+}128z{+}96$	13.73s
tea_dec : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	306	ticks	$128n^2z{+}32nxz{+}1184nz{+}96n{+}96z{+}96$	14.34s

Constant-time block encryption algorithm

Cost models

Summary

- Notion of resource-aware noninterference
- Security type system: combination of classic information flow and resource type systems
- Interactive repair procedure
- Implementation of both linear and polynomial resource consumption

Future work

- Reason about effects of compilation tools and hardware platforms