Program Verification Using Separation Logic

Dino Distefano
Queen Mary University of London

Lecture 5

Today's plan

- Adaptive Shape Analysis
- Bi-Abduction and compositional analysis

Part I

Adaptive shape analysis

Analyzing Real Code

- Issues:
  - Complex data structures
  - Concurrency
  - Incomplete code, Libraries...
  - others....

Analyzing Real Code

- Issues:
  - Complex data structures
  - Concurrency
  - Incomplete code, Libraries...
  - others....

Fiction:

"no worries, device drivers use mostly lists"

IEEE 1394 Firewire

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

pMdl

pMdl

pMdl

MDL

MDL

MDL

NULL

NULL

MDL

MDL

NULL

BusResetIrp_Flink

DEVICE_E
XTENSION

BUS_RESET_IRPS

BUS_RESET_IRPS

BUS_RESET_IRPS

BUS_RESET_IRPS

BusResetIrp_Blink

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

DeviceExtension

IsochDetachData_Mdl

IsochDetachData_Mdl

IsochDetachData_Mdl

DEVICE_OBJECT

devObj

NULL

MDL

MDL

MDL

NULL

MDL

MDL

MDL

NULL

Fact

Real device drivers use lists in combination, resulting in more complicated data structures than those found in previous papers on shape analysis

```
typedef struct {
  PDEVICE_OBJECT          StackDeviceObject;
  PDEVICE_OBJECT          PortDeviceObject;
  PDEVICE_OBJECT          PhysicalDeviceObject;

  UNICODE_STRING          SymbolicLinkName;
  KSPIN_LOCK              ResetSpinLock;
  KSPIN_LOCK              CromSpinLock;
  KSPIN_LOCK              AsyncSpinLock;
  KSPIN_LOCK              IsochSpinLock;
  KSPIN_LOCK              IsochResourceSpinLock;


  BOOLEAN                 bShutdown;
  DEVICE_POWER_STATE      CurrentDevicePowerState;
  SYSTEM_POWER_STATE      CurrentSystemPowerState;

  ULONG                   GenerationCount;
  PASYNC_ADDRESS_DATA     Flink1;
  PASYNC_ADDRESS_DATA     Blink1;
  PBUS_RESET_IRP          Flink2;
  PBUS_RESET_IRP          Blink2;
  PCROM_DATA              Flink3;
  PCROM_DATA              Blink3;
  _PISOCH_DETACH_DATA     Flink4;
  _PISOCH_DETACH_DATA     Blink4;
  PISOCH_RESOURCE_DATA    Flink5;
  PISOCH_RESOURCE_DATA    Blink5;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```

```
typedef struct ASYNC_ADDRESS_DATA {
  struct ASYNC_ADDRESS_DATA*  Flink1;
  struct ASYNC_ADDRESS_DATA*  Blink1;
  _PDEVICE_EXTENSION       DeviceExtension;
  PVOID                    Buffer;
  ULONG                    nLength;
  ULONG                    nAddressesReturned;
  PADDRESS_RANGE           AddressRange;
  HANDLE                   hAddressRange;
  PMDL                     pMdl;
} ASYNC_ADDRESS_DATA, *PASYNC_ADDRESS_DATA;

typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  struct BUS_RESET_IRP *Blink2;
  PIRP              Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;

typedef struct CROM_DATA {
  struct CROM_DATA *Flink3;
  struct CROM_DATA *Blink3;
  HANDLE           hCromData;
  PVOID            Buffer;
  PMDL             pMdl;
} CROM_DATA, *PCROM_DATA;

typedef struct ISOCH_RESOURCE_DATA {
  struct ISOCH_RESOURCE_DATA *Flink5;
  struct ISOCH_RESOURCE_DATA *Blink5;
  HANDLE           hResource;
} ISOCH_RESOURCE_DATA, *PISOCH_RESOURCE_DATA;
```

```
typedef struct {
  PDEVICE_OBJECT          StackDeviceObject;
  PDEVICE_OBJECT          PortDeviceObject;
  PDEVICE_OBJECT

  UNICODE_STRING
  KSPIN_LOCK              ResetSpinLock;
  KSPIN_LOCK              CromSpinLock;
  KSPIN_LOCK              AsyncSpinLock;
  KSPIN_LOCK              IsochSpinLock;
  KSPIN_LOCK              IsochResourceSpinLock;


  BOOLEAN                 bShutdown;
  DEVICE_POWER_STATE      CurrentDevicePowerState;
  SYSTEM_POWER_STATE      CurrentSystemPowerState;

  ULONG                   GenerationCount;
  PASYNC_ADDRESS_DATA     Flink1;
  PASYNC_ADDRESS_DATA     Blink1;
  PBUS_RESET_IRP          Flink2;
  PBUS_RESET_IRP          Blink2;
  PCROM_DATA              Flink3;
  PCROM_DATA              Blink3;
  _PISOCH_DETACH_DATA     Flink4;
  _PISOCH_DETACH_DATA     Blink4;
  PISOCH_RESOURCE_DATA    Flink5;
  PISOCH_RESOURCE_DATA    Blink5;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```

around 600 loc struct definitions

```
typedef struct ASYNC_ADDRESS_DATA {
  struct ASYNC_ADDRESS_DATA*  Flink1;
  struct ASYNC_ADDRESS_DATA*  Blink1;
  _PDEVICE_EXTENSION          DeviceExtension;
  PVOID
                                                  sReturned;
  PADDRESS_RANGE              AddressRange;
  HANDLE                      hAddressRange;
  PMDL                        pMdl;
} ASYNC_ADDRESS_DATA, *PASYNC_ADDRESS_DATA;

typedef struct BUS_RESET_IRP {
  struct BUS_RESET_IRP *Flink2;
  struct BUS_RESET_IRP *Blink2;
  PIRP                 Irp;
} BUS_RESET_IRP, *PBUS_RESET_IRP;

typedef struct CROM_DATA {
  struct CROM_DATA *Flink3;
  struct CROM_DATA *Blink3;
  HANDLE           hCromData;
  PVOID            Buffer;
  PMDL             pMdl;
} CROM_DATA, *PCROM_DATA;

typedef struct ISOCH_RESOURCE_DATA {
  struct ISOCH_RESOURCE_DATA *Flink5;
  struct ISOCH_RESOURCE_DATA *Blink5;
  HANDLE           hResource;
} ISOCH_RESOURCE_DATA, *PISOCH_RESOURCE_DATA;
```

```
typedef struct {
  PDEVICE_OBJECT          StackDeviceObject;
  PDEVICE_OBJECT          PortDeviceObject;
  PDEVICE_OBJECT

  UNICODE_STRING
  KSPIN_LOCK              ResetSpinLock;
  KSPIN_LOCK              CromSpinLock;
  KSPIN_LOCK              AsyncSpinLock;
  KSPIN_LOCK              IsochSpinLock;
  KSPIN_

  BOOLEA
  DEVICE
  SYSTEM

  ULONG
  PASYNC_
  PASYNC_ADDRESS_DATA     Blink1;
  PBUS_RESET_IRP          Flink2;
  PBUS_RESET_IRP          Blink2;
  PCROM_DATA              Flink3;
  PCROM_DATA              Blink3;
  _PISOCH_DETACH_DATA     Flink4;
  _PISOCH_DETACH_DATA     Blink4;
  PISOCH_RESOURCE_DATA    Flink5;
  PISOCH_RESOURCE_DATA    Blink5;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```

around 600 loc struct definitions

many big structs (around 20 fields) mutually pointing to aeach other in several way with several fields

```
typedef struct ASYNC_ADDRESS_DATA {
  struct ASYNC_ADDRESS_DATA*  Flink1;
  struct ASYNC_ADDRESS_DATA*  Blink1;
  _PDEVICE_EXTENSION          DeviceExtension;

                                          sReturned;
  PADDRESS_RANGE              AddressRange;
  HANDLE                      hAddressRange;
  PMDL                        pMdl;
} ASYNC_ADDRESS_DATA, *PASYNC_ADDRESS_DATA;

  struct CROM_DATA *Blink3;
  HANDLE           hCromData;
  PVOID            Buffer;
  PMDL             pMdl;
} CROM_DATA, *PCROM_DATA;

typedef struct ISOCH_RESOURCE_DATA {
  struct ISOCH_RESOURCE_DATA *Flink5;
  struct ISOCH_RESOURCE_DATA *Blink5;
  HANDLE           hResource;
} ISOCH_RESOURCE_DATA, *PISOCH_RESOURCE_DATA;
```

```
typedef struct {
  PDEVICE_                 StackDeviceOb
  PDEVICE_O                rtDeviceOb
  PDEVICE_OBJ

  UNICODE_STRING
  KSPIN_LOCK
  KSPIN_LOCK

  BOOLEA
  DEVICE
  SYSTEM

  PASYNC_ADD
  PBUS_RESET_IRP
  PBUS_RESET_IRP
  PCROM_DATA
  PCROM_DATA
  _PISOCH_DETACH_DATA      nk4;
  _PISOCH_DETACH_DATA      Blink4;
  PISOCH_RESOURCE_DATA     Flink5;
  PISOCH_RESOURCE_DATA     Blink5;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```

```
typedef struct ASYNC_ADDRESS_DATA {
  st    ct ASYNC_ADD    RESS_DATA*  Flink1;
             ASY        DRESS_DATA*  Bl   ink1;
                  NSION          ceExtension;
                                           urned;
                                           e;
                                           nge;

                                           S_DATA;

                               ata;
                               ;

  CR                           ;

typedef st                 URCE_DATA {
  struct ISOCH_RES          DATA *Flink5;
  struct ISOCH_RESOUR       DATA *Blink5;
  HANDLE          hResou    ce;
} ISOCH_RESOURCE_DATA, *PISOCH_RESOURCE_DATA;
```

Nearly impossible to understand which shape the heap will have

In summary

- Problem 1 (Complex shapes): analyses dealing only with plain lists are not enough.
- Problem 2 (Automation): How to avoid understanding data definitions and their relations.

In summary

- Problem 1 (Complex shapes): analyses dealing only with plain lists are not enough.
- Problem 2 (Automation): How to avoid understanding data definitions and their relations.

No fix domain for data structures!
We need flexibility!

Main idea of the analysis

Adaptive aspect

Try to reduce complex shapes into Lists of simpler shapes by partitioning the heap (on-the-fly) into collections of building blocks

Main ingredients

High-order List predicate Is P (x,y)

Special building blocks Predicates $P = \lambda[x'_1, x'_2](\exists \vec{v}.H)$

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

pMdl

pMdl

pMdl

MDL

MDL

MDL

NULL

NULL

MDL

MDL

NULL

BusResetIrp_Flink

DEVICE_E
XTENSION

BUS_RESET_IRPS

BUS_RESET_IRPS

BUS_RESET_IRPS

BUS_RESET_IRPS

BusResetIrp_Blink

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

DeviceExtension

IsochDetachData_Mdl

IsochDetachData_Mdl

DEVICE_OBJECT

NULL

MDL

MDL

MDL

MDL

NULL

MDL

MDL

NULL

devObj

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

pMdl

pMdl

pMdl

list MDL

list MDL

list MDL

BusResetIrp_Flink

DEVICE_E
XTENSION

BUS_RESET_IRPS

BUS_RESET_IRPS

BUS_RESET_IRPS

BUS_RESET_IRPS

BusResetIrp_Blink

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

DeviceExtension

IsochDetachData_Mdl

IsochDetachData_Mdl

DEVICE_OBJECT

list MDL

list MDL

list MDL

devObj

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

pMdl

pMdl

pMdl

list MDL

list MDL

list MDL

BusResetIrp_Flink

DEVICE_E
XTENSION

list Bus_Reset_Irps

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

DeviceExtension

IsochDetachData_Mdl

IsochDetachData_Mdl

DEVICE_OBJECT

list MDL

list MDL

list MDL

devObj

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

ASYNCH_ADDRESS_DATA

pMdl

pMdl

pMdl

list MDL

list MDL

list MDL

BusResetIrp_Flink

DEVICE_E
XTENSION

list Bus_Reset_Irps

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

DeviceExtension

IsochDetachData_Mdl

IsochDetachData_Mdl

DEVICE_OBJECT

list MDL

list MDL

list MDL

devObj

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

AAD

AAD

AAD

DEVICE_E
XTENSION

BusResetIrp_Flink

list Bus_Reset_Irps

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

DeviceExtension

IsochDetachData_Mdl

IsochDetachData_Mdl

DEVICE_OBJECT

list MDL

list MDL

list MDL

devObj

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

AAD

AAD

AAD

DEVICE_E
XTENSION

BusResetIrp_Flink

list Bus_Reset_Irps

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

ISOCH_DETACH_DATA

DeviceExtension

IsochDetachData_Mdl

IsochDetachData_Mdl

DEVICE_OBJECT

list MDL

list MDL

list MDL

devObj

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

AAD

AAD

AAD

DEVICE_E
XTENSION

BusResetIrp_Flink

list Bus_Reset_Irps

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

DeviceExtension

IDD

IDD

IDD

DeviceExtension

DEVICE_OBJECT

devObj

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

list ADD

BusResetIrp_Flink

DEVICE_E
XTENSION

list Bus_Reset_Irps

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

DeviceExtension

IDD

IDD

IDD

DeviceExtension

DEVICE_OBJECT

devObj

AsynchAddressData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

AsynchAddressData_Blink

list ADD

BusResetIrp_Flink

DEVICE_E
XTENSION

list Bus_Reset_Irps

IsochDetachData_Blink

IsochDetachData_Flink

DeviceExtension

DeviceExtension

DeviceExtension

list IDD

DeviceExtension

DEVICE_OBJECT

devObj

Type-tagged Points-to

- Synthesized from the input program
- One predicate for each structure declared in the program
- They are the most basic blocks for building other predicates

```
typedef struct{
    PLIST_ENTRY *Blink;
    PLIST_ENTRY *Flink;
} LIST_ENTRY
```

Type-tagged Points-to

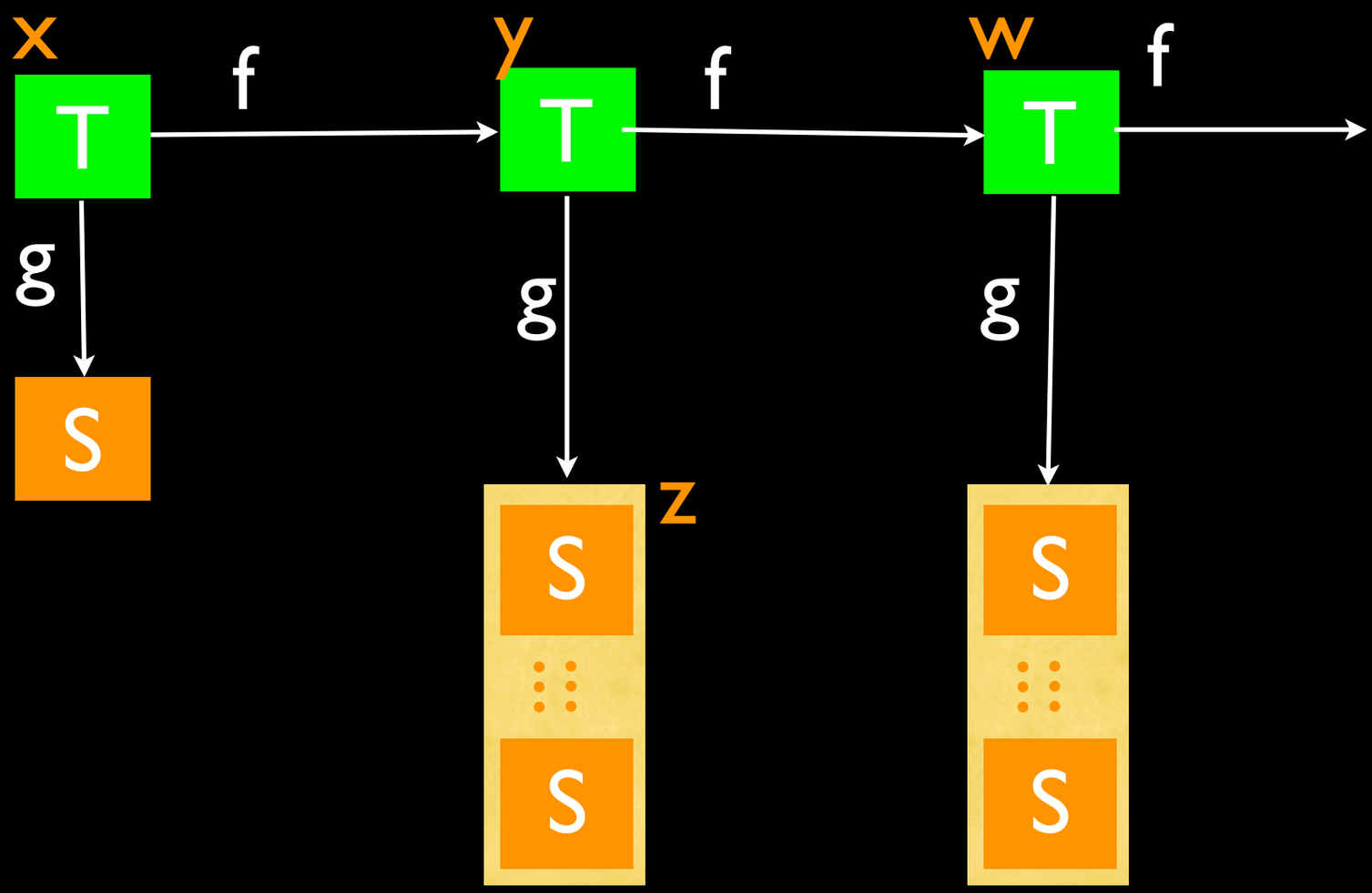
- Synthesized from the input program
- One predicate for each structure declared in the program
- They are the most basic blocks for building other predicates

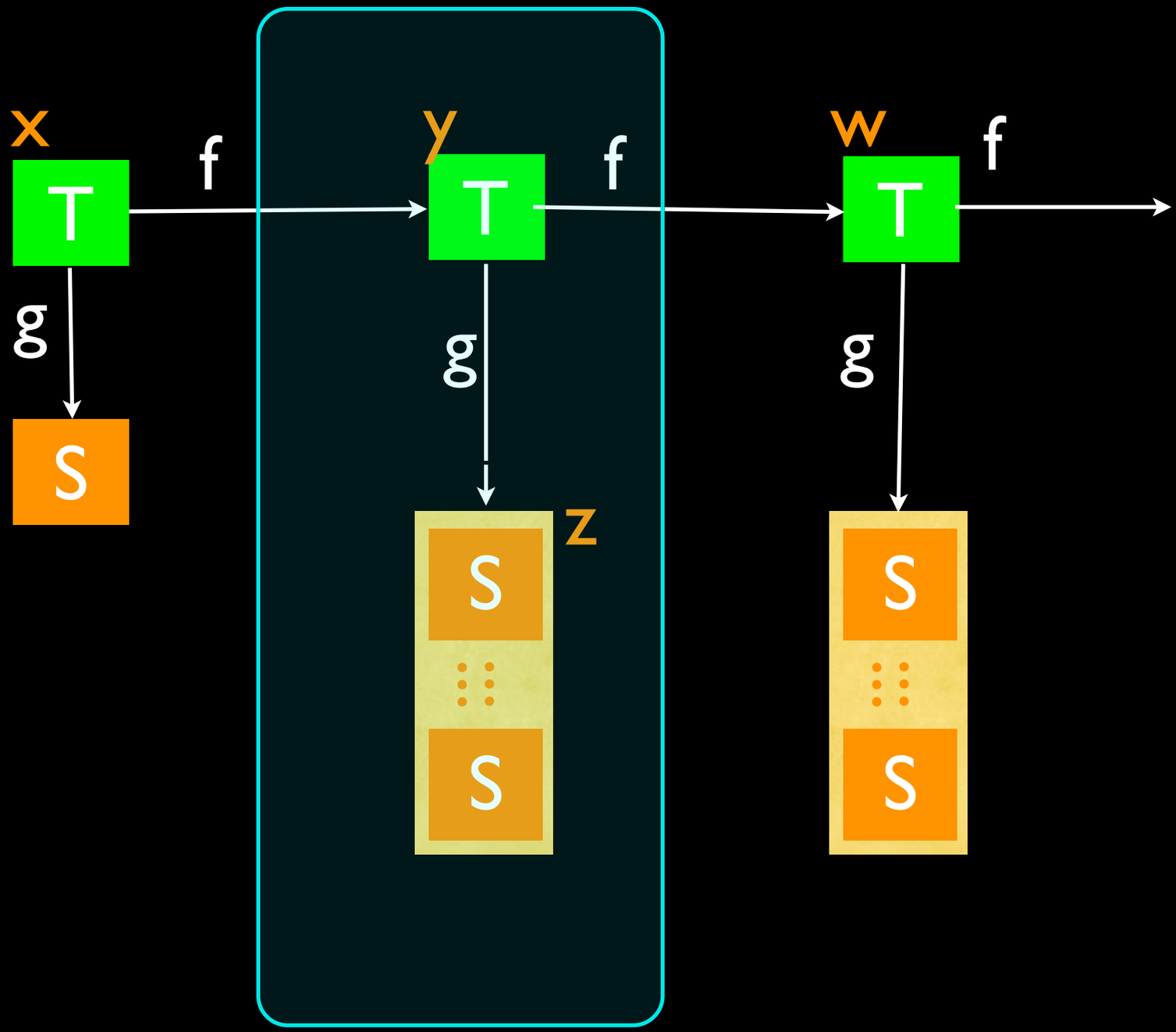
```
typedef struct{
    PLIST_ENTRY *Blink;
    PLIST_ENTRY *Flink;
} LIST_ENTRY
```

$$E \mapsto \texttt{LIST\_ENTRY}(\texttt{Blink}:E_0,\ \texttt{Flink}:E_1)$$

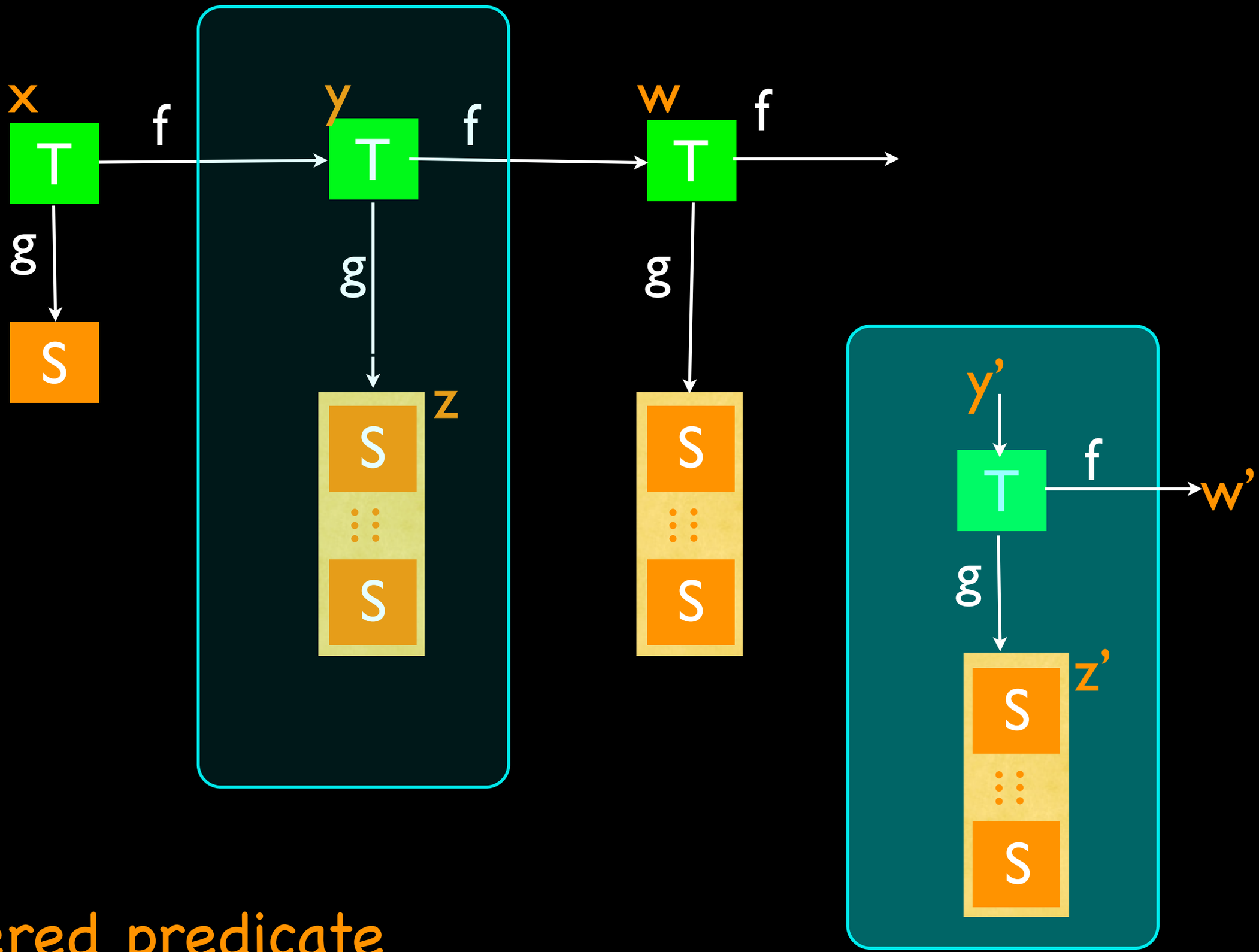
Building blocks predicates

- Predicates describing a some particular shape
- Discovered on-the-fly (i.e. the Set of predicates is not fixed).
- Strategy: new predicates are introduced only if useful to use the list abstraction


x
y
w
f
f
f
T
T
T
g
g
g
S
z
S
S
S
S

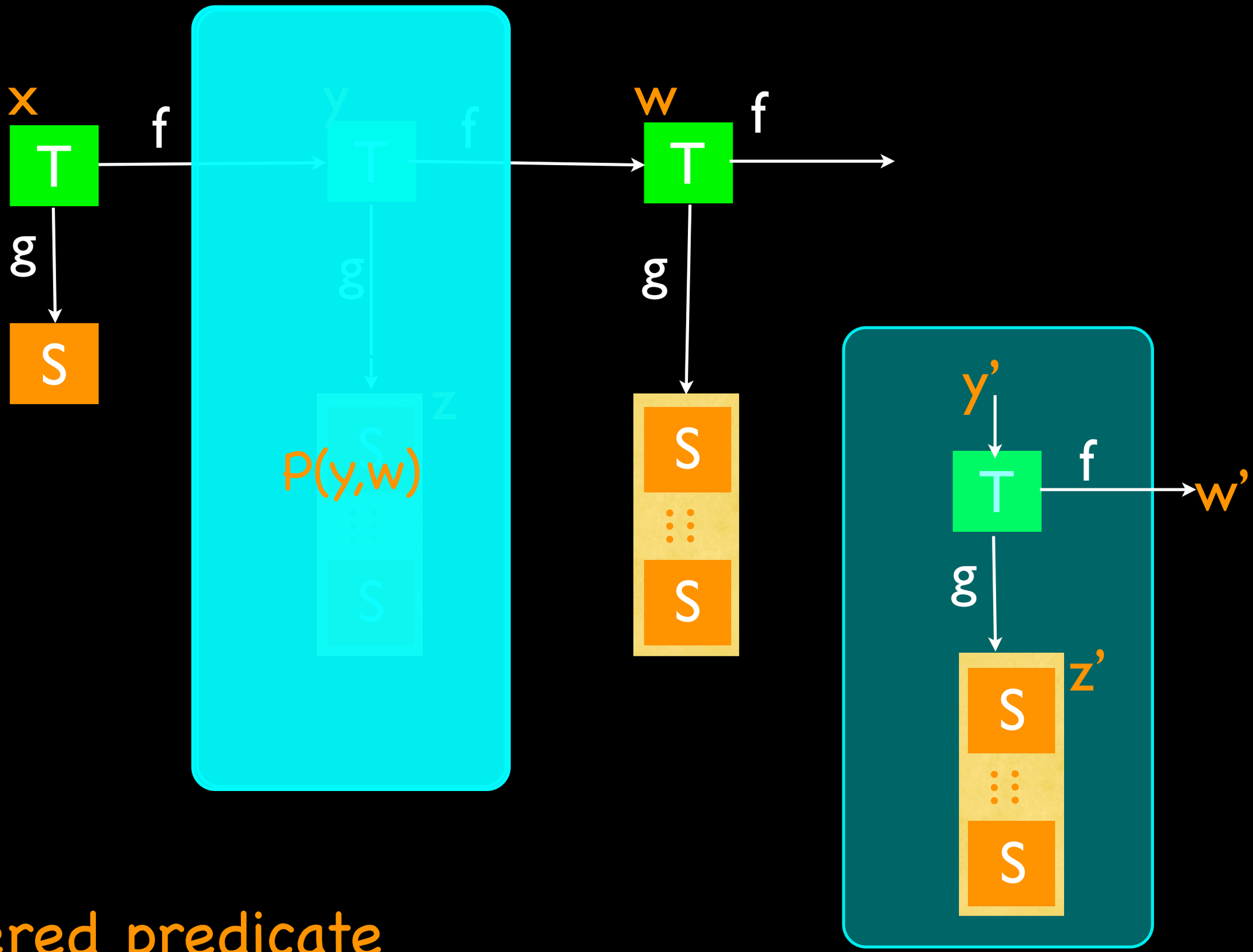


y|->T(g:z, f:w) * ls S (z,nil)



Discovered predicate

P= lambda[y',w']. exists z'. y'|->T(g:z', f:w') * ls S (z',nil)



Discovered predicate

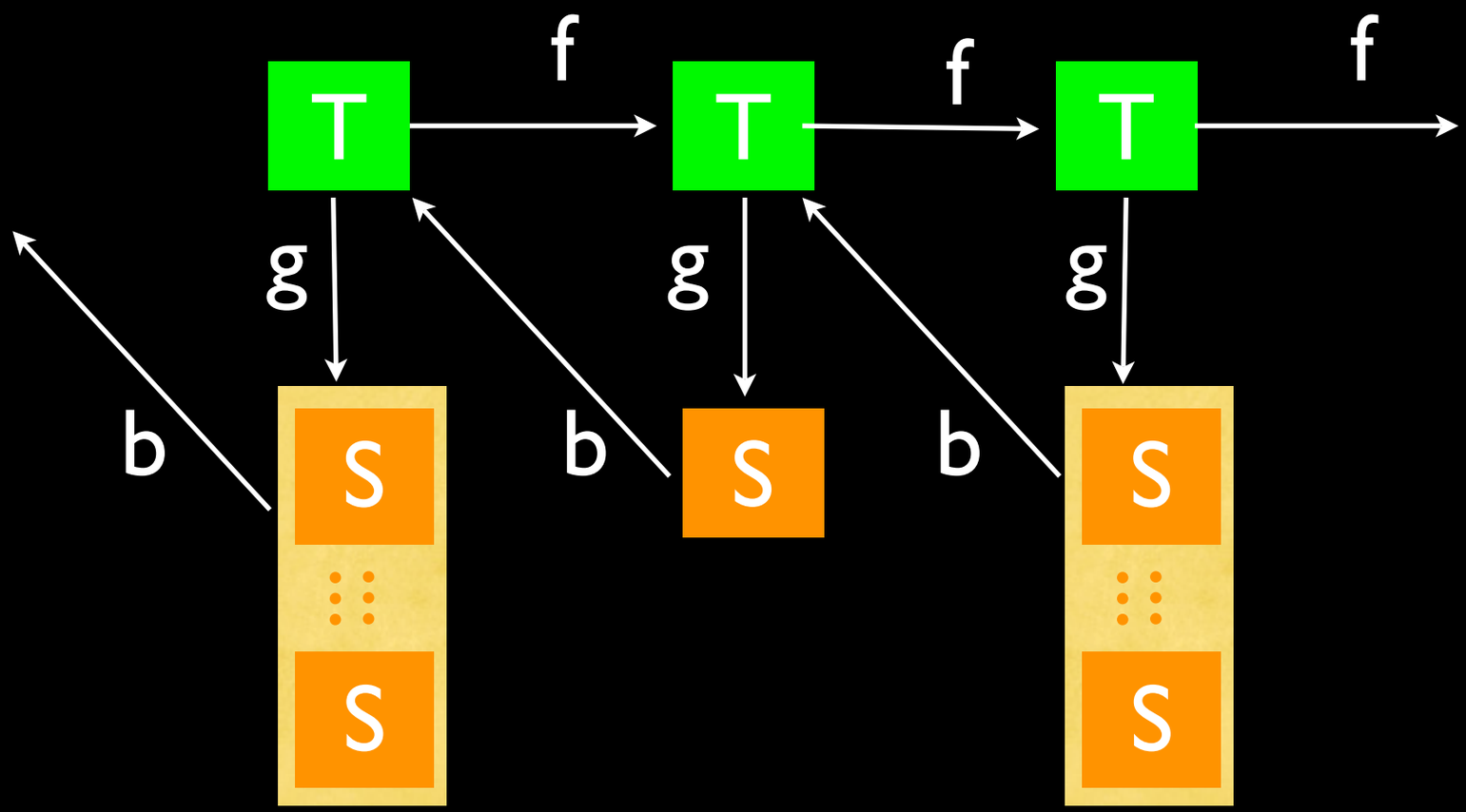
P= lambda[y',w']. exists z'. y'|->T(g:z', f:w') * ls S (z',nil)

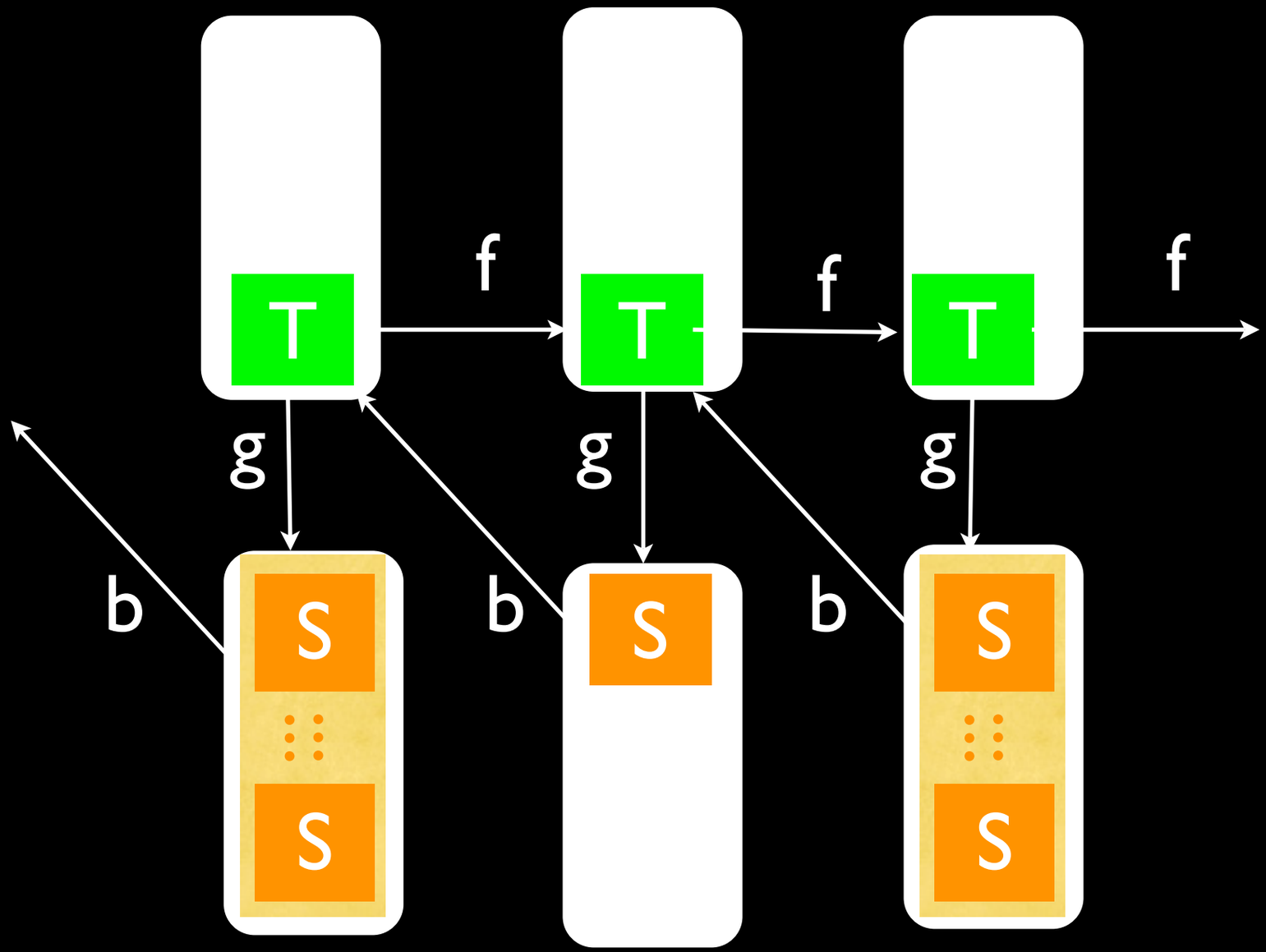
Canonicalization: High-level Algorithm

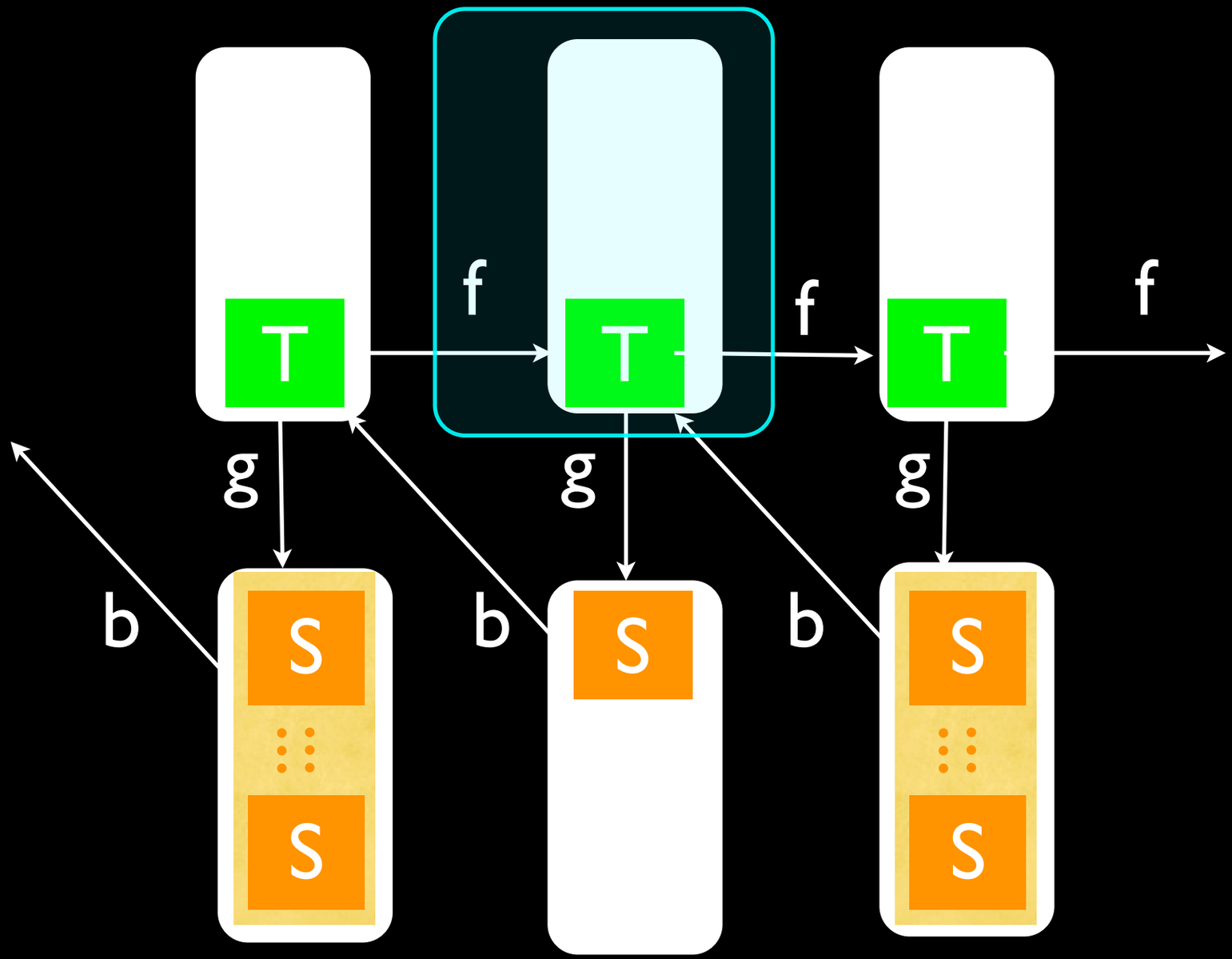
```
FUNCTION CANONICALIZATION
INPUT HEAP H
REPEAT
  IF REDUCTION RULES APPLIES THEN
     H:=REDUCE H
  ELSE IF PREDICATE SYNTHESIS SUCCEEDS THEN
    APPLY PREDICATE DISCOVERY TO H
  ELSE RETURN H
DONE
```

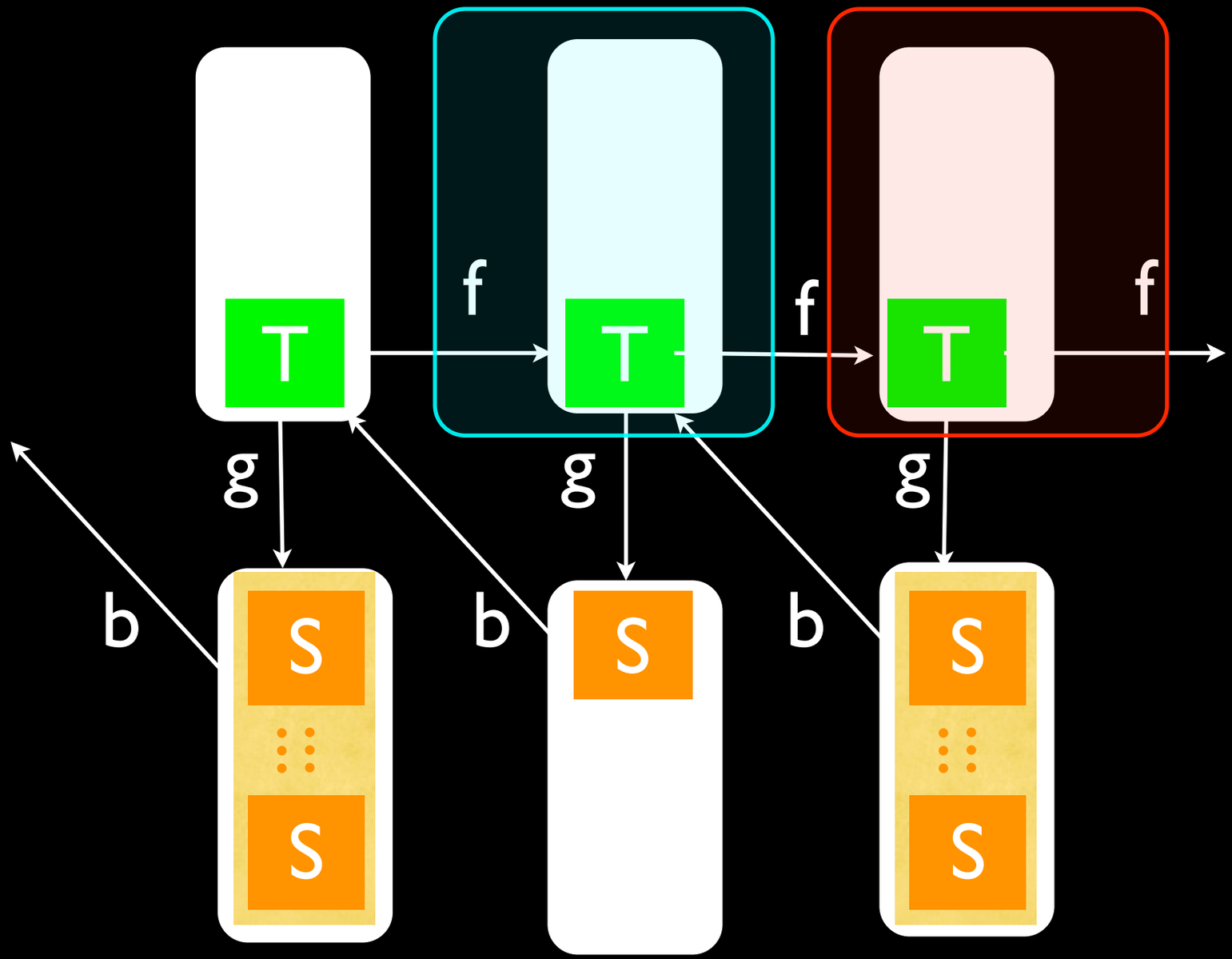
Discovery Strategy

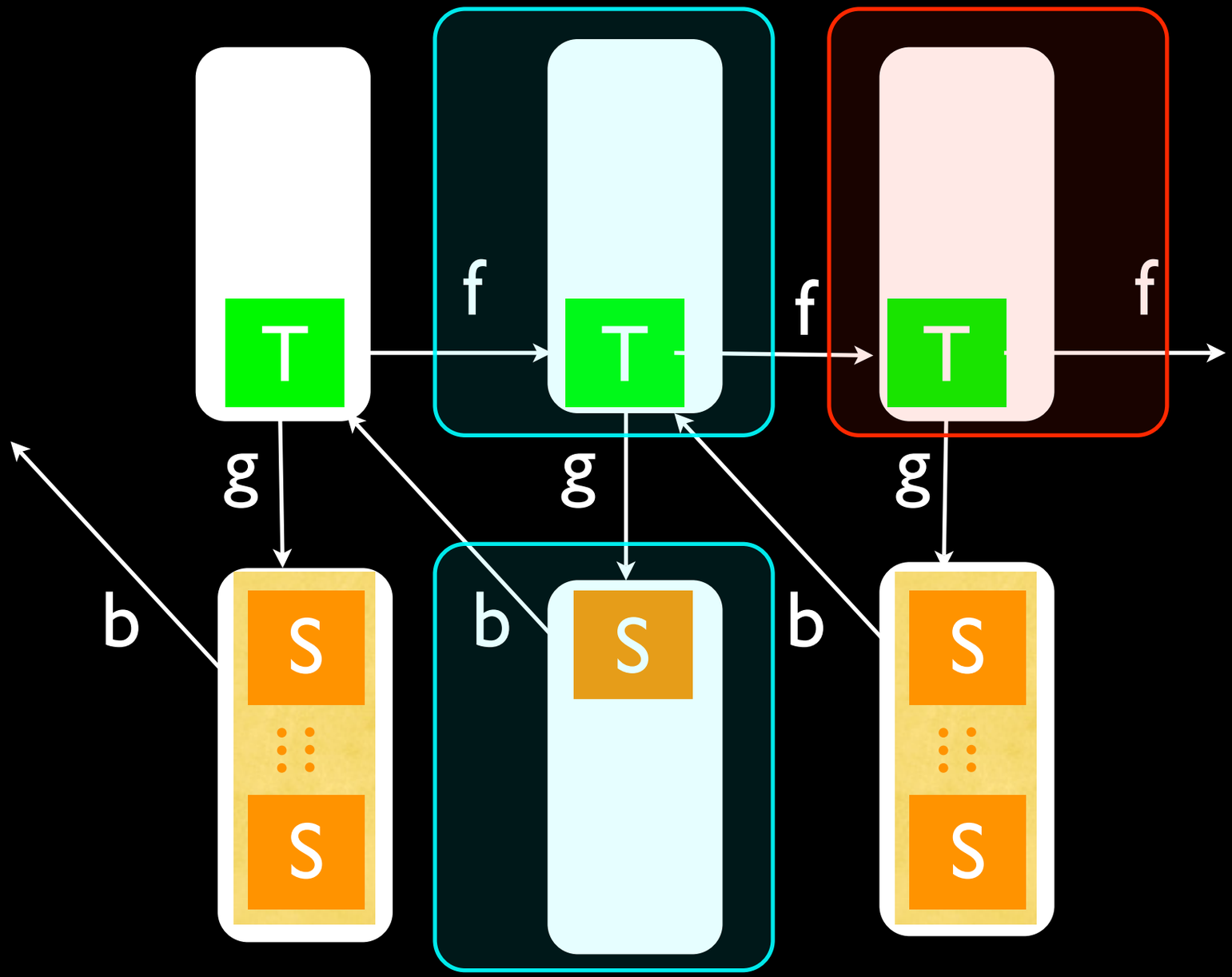
- The discovery function takes an heap H
- Traverse to find two node N and M that can form a list
- Try to get two disjoint isomorphic subgraph reachable from N and M to use as shape for the predicate

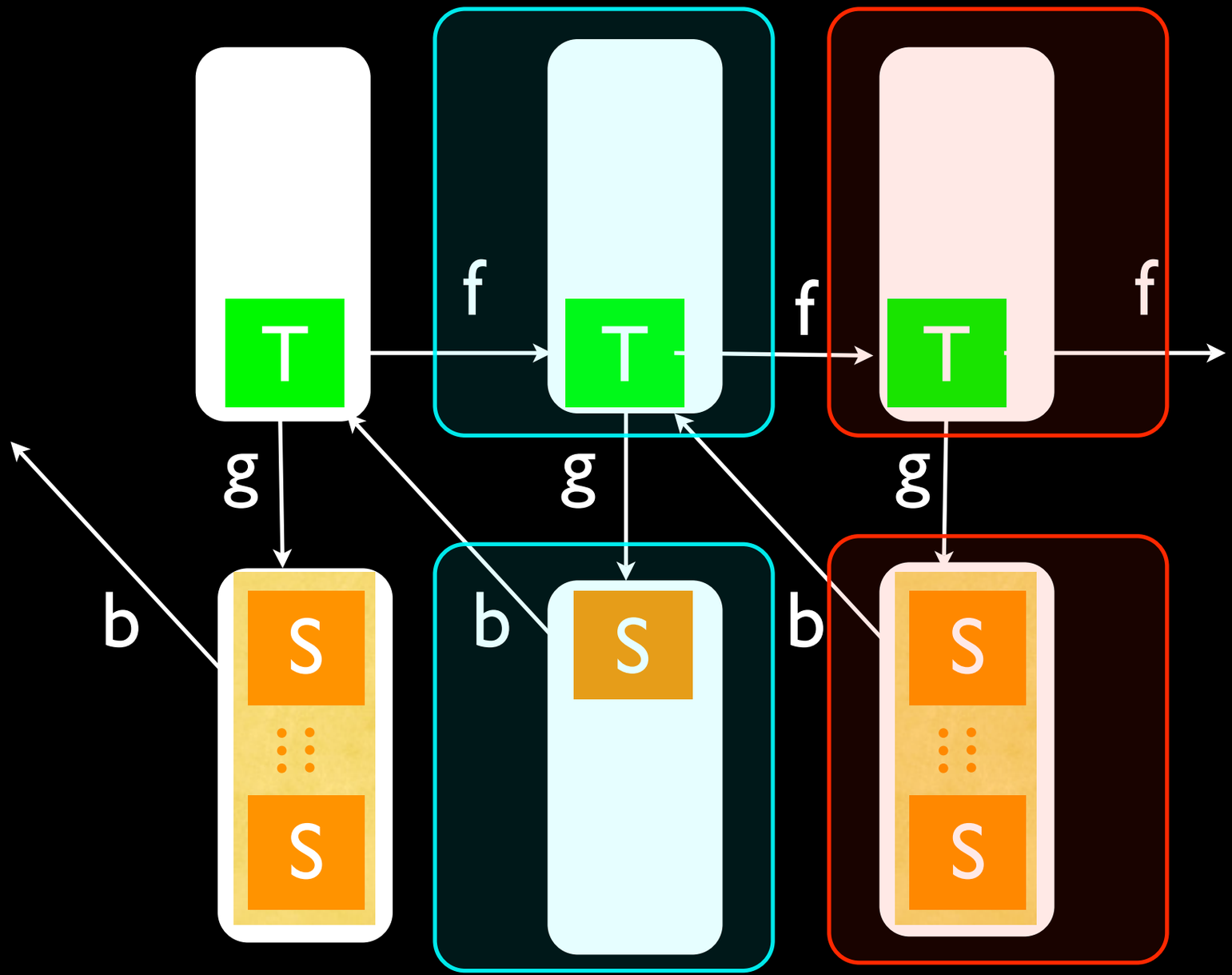

T
f
T
f
T
f
g
g
g
b
b
b
S
S
S
S
S

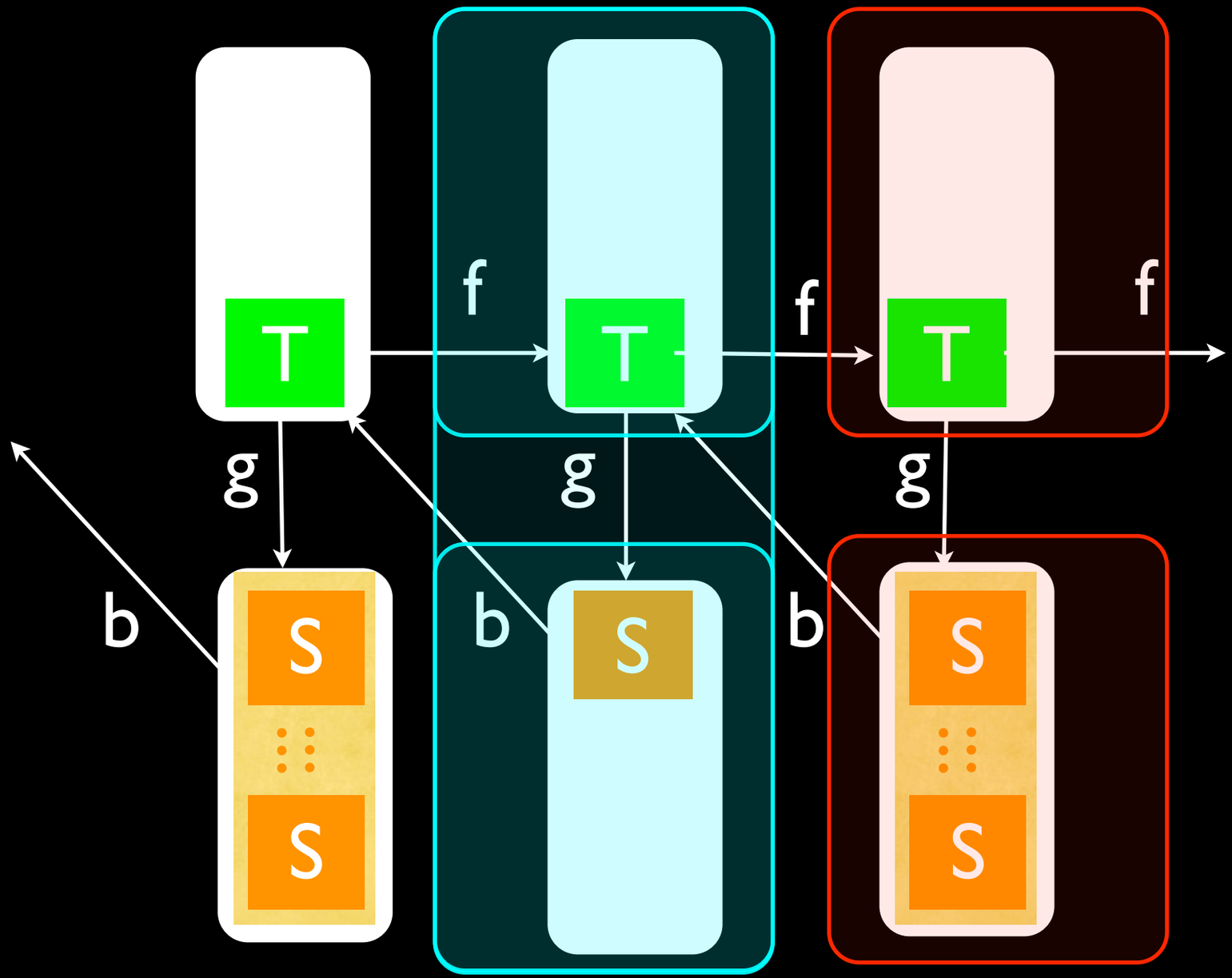

T
T
T
f
f
f
g
g
g
b
b
b
S
S
S
S
S

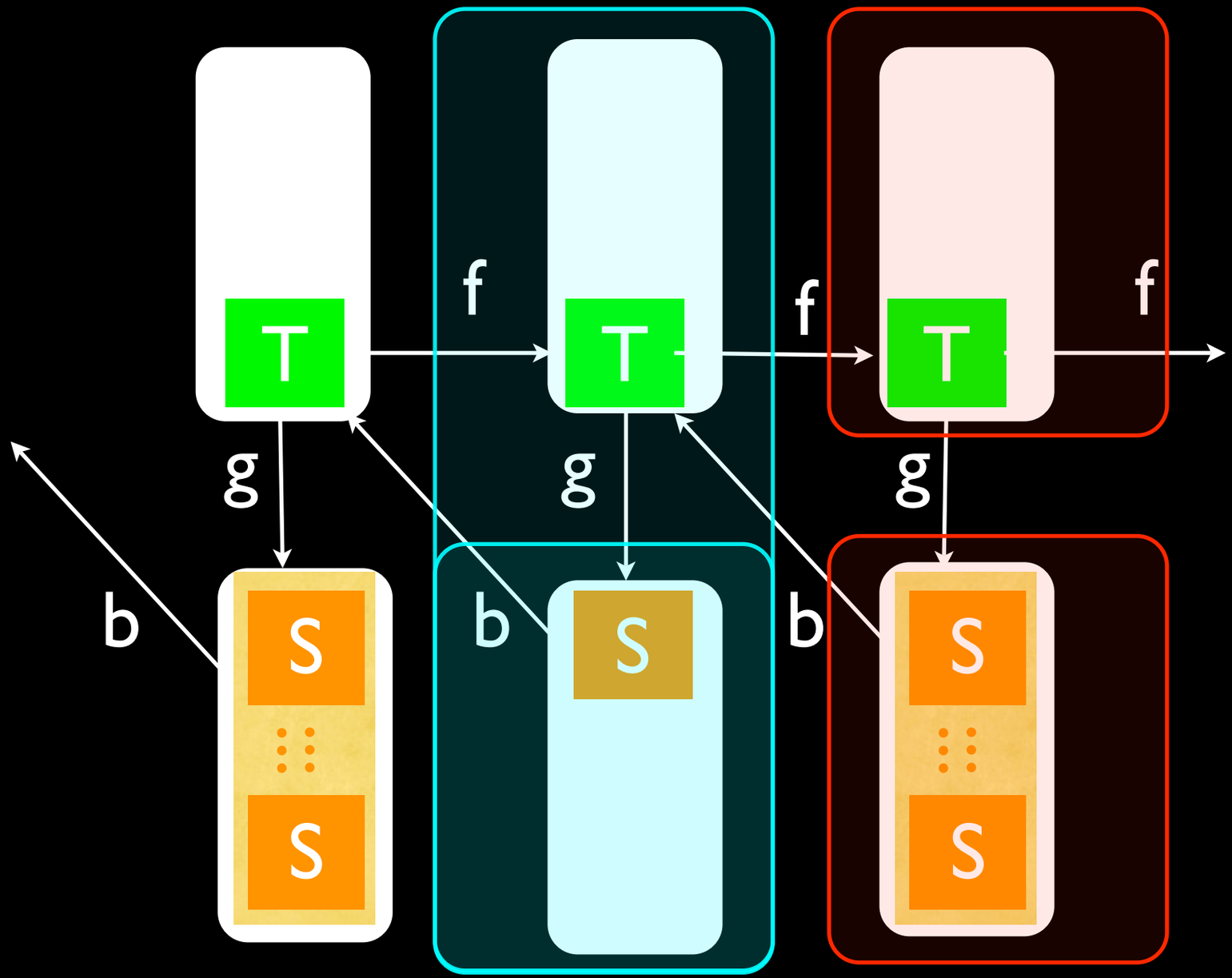

T
T
T
f
f
f
g
g
g
b
b
b
S
S
S
S
S

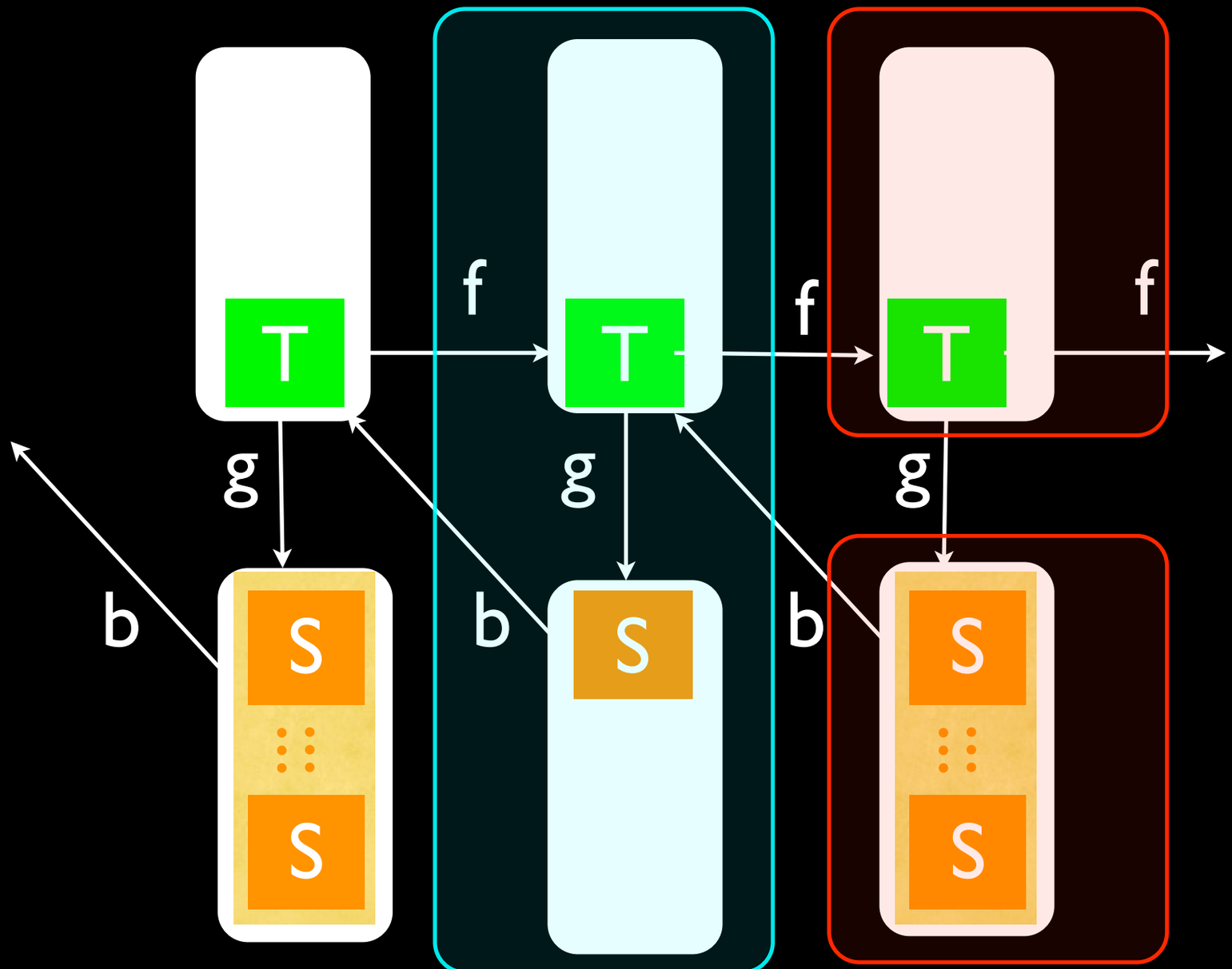

T
f
T
f
T
f
g
g
g
b
b
b
S
S
S
S
S

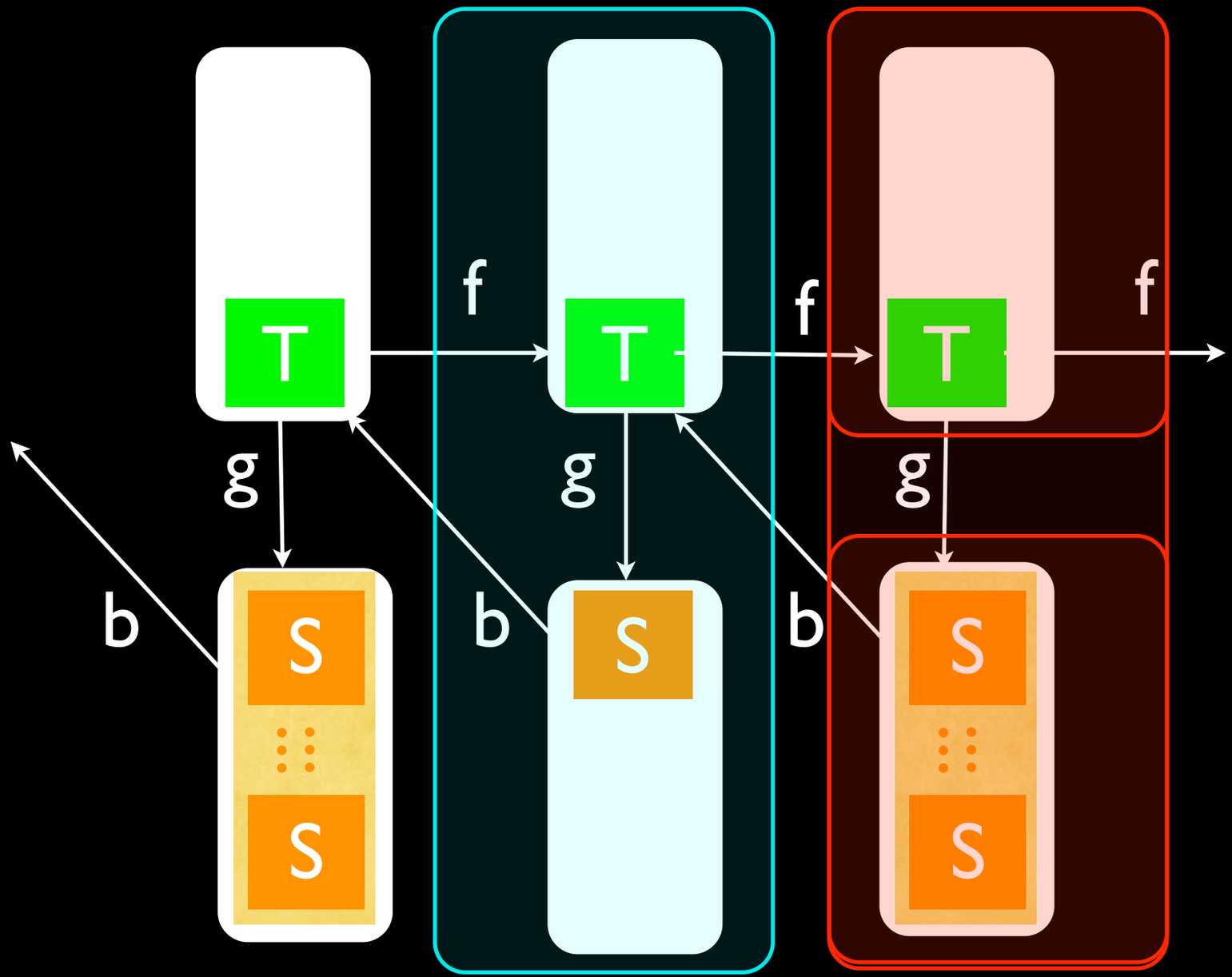

T
f
T
f
T
f
g
g
g
b
S
S
b
S
S
b
S
S

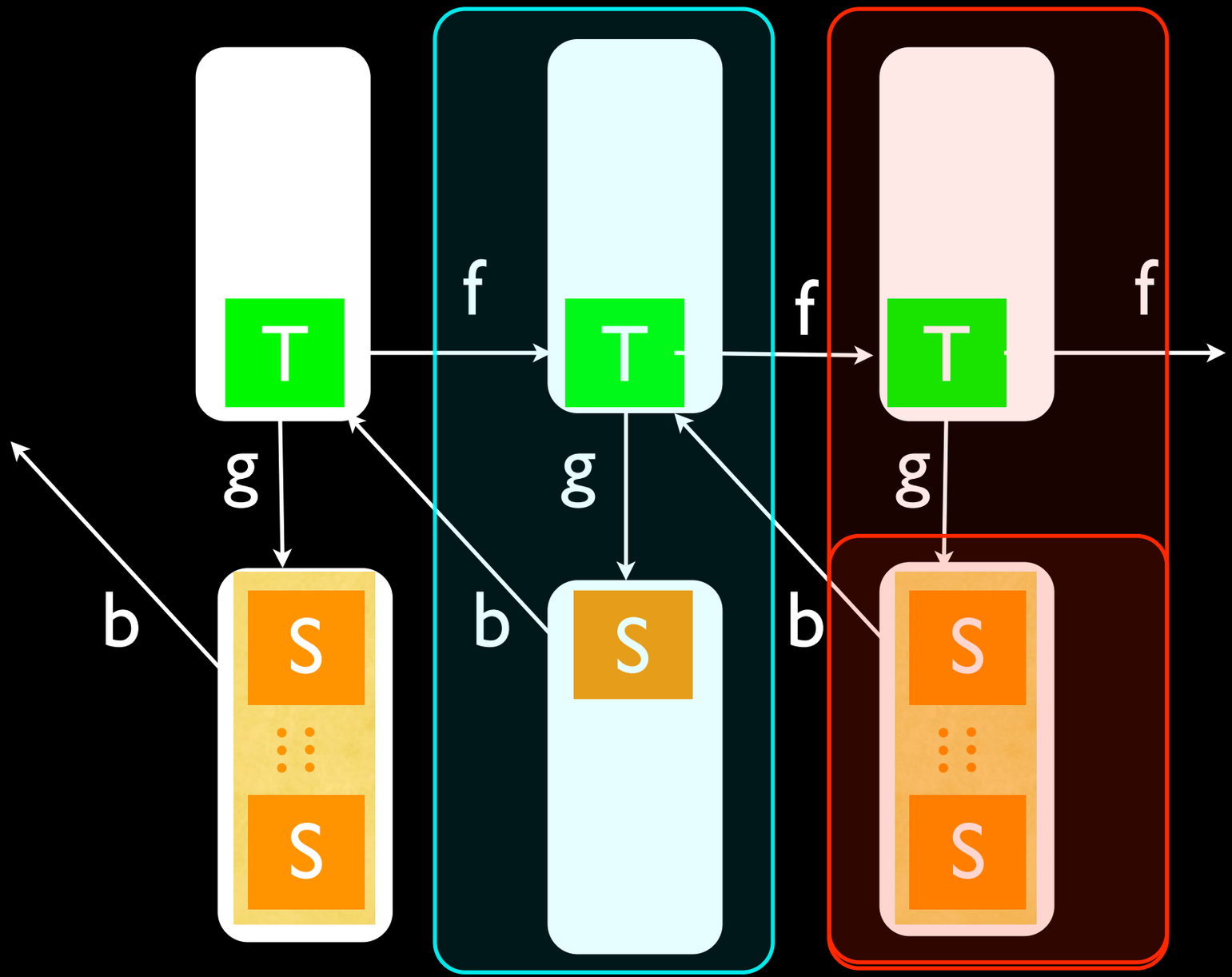

T
f
T
f
T
f
g
g
g
b
S
S
b
S
S
b
S
S

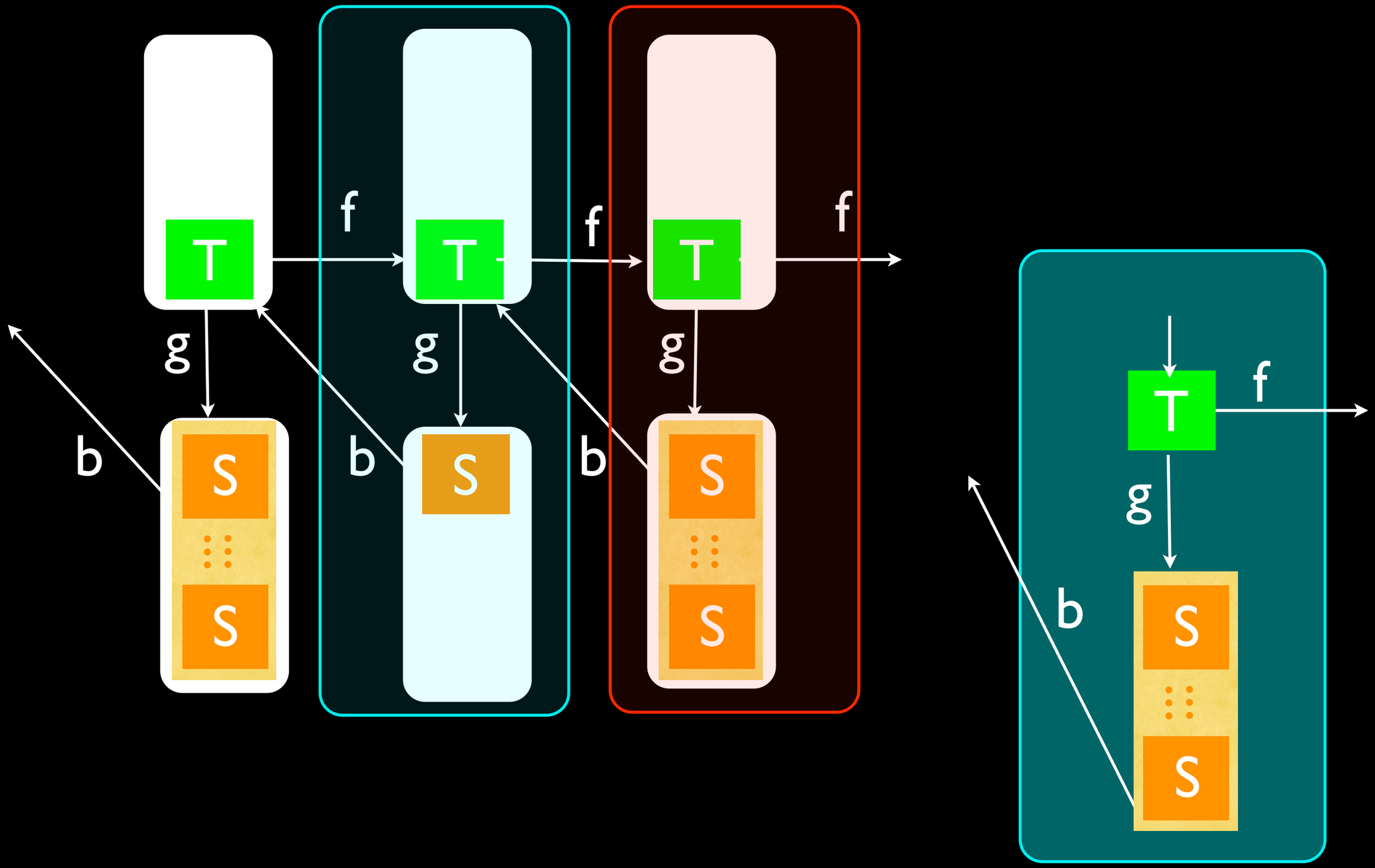

T
f
T
f
T
f
g
g
g
b
S
S
b
S
S
b
S
S


T
f
T
f
T
f
g
g
g
b
S
S
b
S
S
b
S
S


T
T
T
f
f
f
g
g
g
b
b
b
S
S
S
S
S


T
f
T
f
T
f
g
g
g
b
S
b
S
b
S
S
S


T
T
T
f
f
f
g
g
g
b
b
b
S
S
S
S
S


T
T
T
f
f
f
g
g
g
b
b
b
S
S
S
S
S
T
f
g
b
S
S

Which data structures?

- Lists
  - singly cyclic/acyclic
  - doubly cyclic/acyclic
- List of list,
- list of lists of lists of doubly circular lists of strange shapes
- etc....

What don't we do?

- Trees: currently not implemented, but the extension should be reasonably straighforward
- DAG: this sounds involved because of sharing (but interesting to try)
- General Inductive predicate: difficult, but very interesting direction to try

Conclusions

- We analyze complex data structures:
  - On-the-fly discovery of predicates (adaptive aspect)
  - High-order inductive predicates

References:

J. Berdine, C. Calcagno, B. Cook, D. Distefano, P. O'Hearn, T. Wies, H. Yang: Shape Analysis for Composite Data Structures. CAV 2007

Part II
Bi-Abduction

Literature

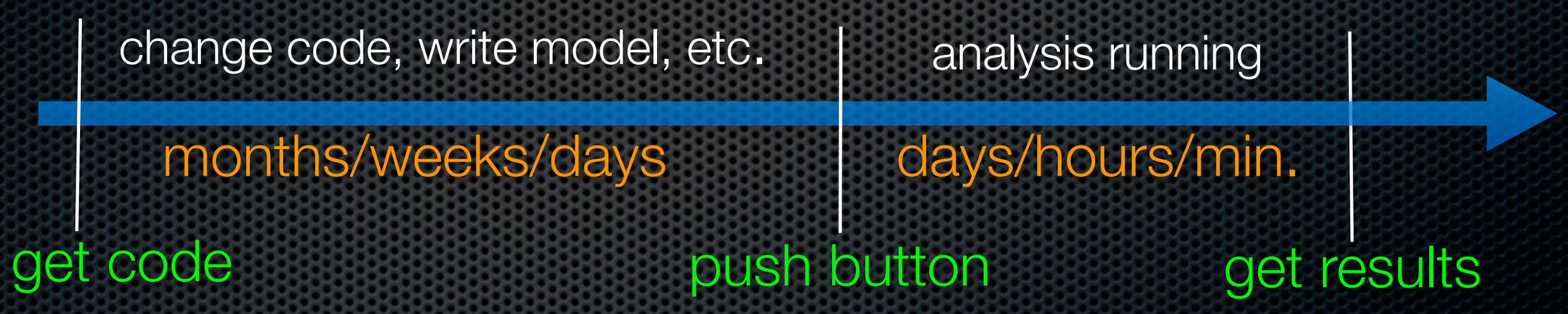
- C. Calcagno, D. Distefano, P. O'Hearn and H. Yang. Compositional Shape Analysis by Means of Bi-Abduction. POPL 2009.
- D. Distefano. Attacking Large Industrial Code with Bi-Abductive Inference. FMICS 2009

Space Invader analyzer: overview

- Inter-procedural shape analysis
- Based on Separation Logic and Abstract interpretation to infer invariants
- Builds proofs or reports possible memory faults or memory leaks

Shape Analysis and Real Code

Shape Analysis and Real Code



Shape Analysis and Real Code

Shape Analysis and Real Code

Need very high modularity

change code, write model, etc.

analysis running

months/weeks/days

days/hours/min.

get code

push button

get results

Need to handle incomplete code

Shape Analysis and Real Code

Need very high modularity

Start with something partial

change code, write model, etc.

analysis running

months/weeks/days

days/hours/min.

get code

push button

get results

Need to handle incomplete code

A compositional Space Invader

- ✓ Handles incomplete code
- ✓ Admits partial results
- ✓ Modular

A compositional Space Invader

✓ Handles incomplete code
✓ Admits partial results
✓ Modular

...demo!

Small specs

- Small specs encourage local reasoning and help to get small proofs
- When proving code involving procedures we use only their footprint

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);
Dispose(l2);
```

Spec: `{list(l)} Dispose(l) {emp}`

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);
Dispose(l2);
```

Spec: `{list(l)} Dispose(l) {emp}`

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);
{emp*list(l2)}
Dispose(l2);
```

Spec: `{list(l)} Dispose(l) {emp}`

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);
{list(l2)}
Dispose(l2);
```

Spec: `{list(l)} Dispose(l) {emp}`

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}$$ Frame Rule

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);
{list(l2)}
Dispose(l2);
{emp}
```

Spec: `{list(l)} Dispose(l) {emp}`

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Spec: `{list(l)}` `Dispose(l)` `{emp}`

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Spec: `{list(l)}` `Dispose(l)` `{emp}`

- In analysis to use the Frame Rule we need to compute R
- Frame inference problem: given A and B compute X such that $A \vdash B * X$

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Spec: `{list(l)}` `Dispose(l)` `{emp}`

- In analysis to use the Frame Rule we need to compute R
- Frame inference problem: given A and B compute X such that $A \vdash B * X$

Example:

`list(l1)*list(l2)`$\vdash$ `list(l1)* X`

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Spec: `{list(l)}` `Dispose(l)` `{emp}`

- In analysis to use the Frame Rule we need to compute R
- Frame inference problem: given A and B compute X such that $A \vdash B * X$

Example:

$$\texttt{list(l1)*list(l2)} \vdash \texttt{list(l1)*list(l2)}$$

Frame Inference

```
{list(l1)*list(l2)}
 Dispose(l1);

 Dispose(l2);
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Spec: `{list(l)}` `Dispose(l)` `{emp}`

- In analysis to use the Frame Rule we need to compute R
- Frame inference problem: given A and B compute X such that $A \vdash B * X$

Example:

$$\texttt{list(l1)*list(l2)} \vdash \texttt{list(l1)*list(l2)}$$

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);
{emp*list(l2)}
Dispose(l2);
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$$ Frame Rule

Spec: `{list(l)}` `Dispose(l)` `{emp}`

- In analysis to use the Frame Rule we need to compute R
- Frame inference problem: given A and B compute X such that $A \vdash B*X$

Example:

`list(l1)*list(l2)` $\vdash$ `list(l1)*list(l2)`

Abduction

Abductive Inference (Charles Peirce, circa 1900, writing about the scientific process)

"Abduction is the process of forming an explanatory hypothesis. It is the only logical operation which introduces any new idea"

"A man must be downright crazy to deny that science has made many true discoveries. But every single item of scientific theory which stands established today has been due to Abduction."

The Collected Papers of Charles Sanders Peirce, Volume V, Pragmatism and Pragmaticism

Abduction for Space Invader

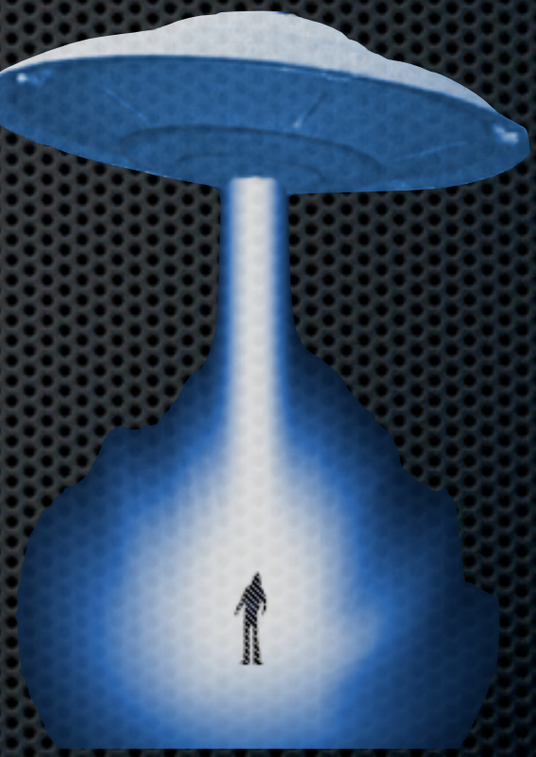
Abduction for Space Invader

Abduction for Space Invader

Abduction Inference:

given A and B compute X such that

$$A * X \vdash B$$



Abduction for Space Invader

Abduction Inference:

given A and B compute X such that

$$A * X \vdash B$$

Example:

Spec: `{list(l1)*list(l2)}` `Dispose_Two_Lists(l1,l2)` `{emp}`

`list(l1)`

Abduction for Space Invader

Abduction Inference:

given A and B compute X such that

$$A * X \vdash B$$

Example:

Spec: `{list(l1)*list(l2)}` `Dispose_Two_Lists(l1,l2)` `{emp}`

`list(l1)*X` ⊢ `list(l1)*list(l2)`

Abduction for Space Invader

Abduction Inference:

given A and B compute X such that

$$A * X \vdash B$$

Example:

Spec: `{list(l1)*list(l2)}` `Dispose_Two_Lists(l1,l2)` `{emp}`

`list(l1)*list(l2)`$\vdash$ `list(l1)*list(l2)`

Abduction is not enough

If heaps A and B are incomparable abduction and frame inference alone are not enough.

We need to synthesize both missing portion of state and leftover portion of state

Heap A

$$x \mapsto y \quad * \quad y \mapsto y'$$

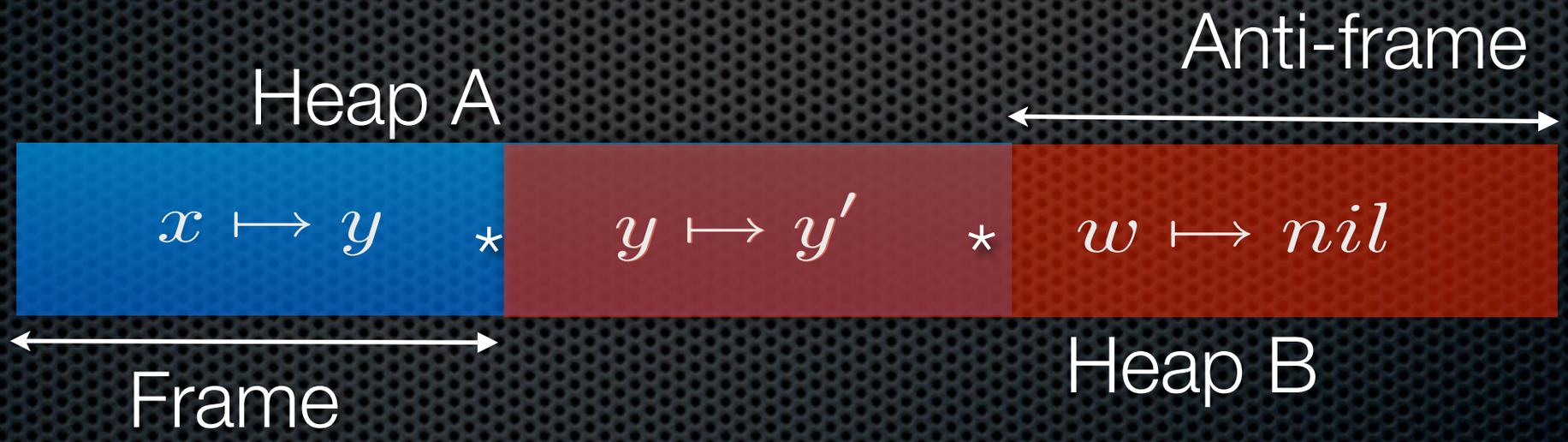
$$y \mapsto y' \quad * \quad w \mapsto nil$$

Heap B

Abduction is not enough

If heaps A and B are incomparable abduction and frame inference alone are not enough.

We need to synthesize both missing portion of state and leftover portion of state



Bi-Abduction

Synthesizing both missing portion of state (anti-frame) and leftover portion of state (frame) give rise to a new notion

Bi-Abduction:

given A and B compute ?antiframe and ?frame such that

$$A * ?antiframe \vdash B * ?frame$$

Bi-Abduction

Synthesizing both missing portion of state (anti-frame) and leftover portion of state (frame) give rise to a new notion

Bi-Abduction:

given A and B compute ?antiframe and ?frame such that

$$A * ?antiframe \vdash B * ?frame$$

Example:

$$x \mapsto 0 * z \mapsto 0 * ?antiframe \vdash \mathsf{list}(x) * \mathsf{list}(y) * ?frame$$

Bi-Abduction

Synthesizing both missing portion of state (anti-frame) and leftover portion of state (frame) give rise to a new notion

Bi-Abduction:

given A and B compute ?antiframe and ?frame such that

$$A * ?antiframe \vdash B * ?frame$$

Example:

$$x \mapsto 0 * z \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y) * z \mapsto 0$$

Bi-Abduction

Synthesizing both missing portion of state (anti-frame) and leftover portion of state (frame) give rise to a new notion

Bi-Abduction:

given A and B compute ?antiframe and ?frame such that

$$A * ?antiframe \vdash B * ?frame$$

Example:

$$x \mapsto 0 * z \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y) * z \mapsto 0$$

Our POPL'09 paper defines a theorem prover for Bi-Abduction

Compositionality

Principle of Compositionality (Frege's principle)

The meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them.

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {                                  emp
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;             emp
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

Bi-abductive prover

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

Bi-abductive prover

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-abductive prover

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

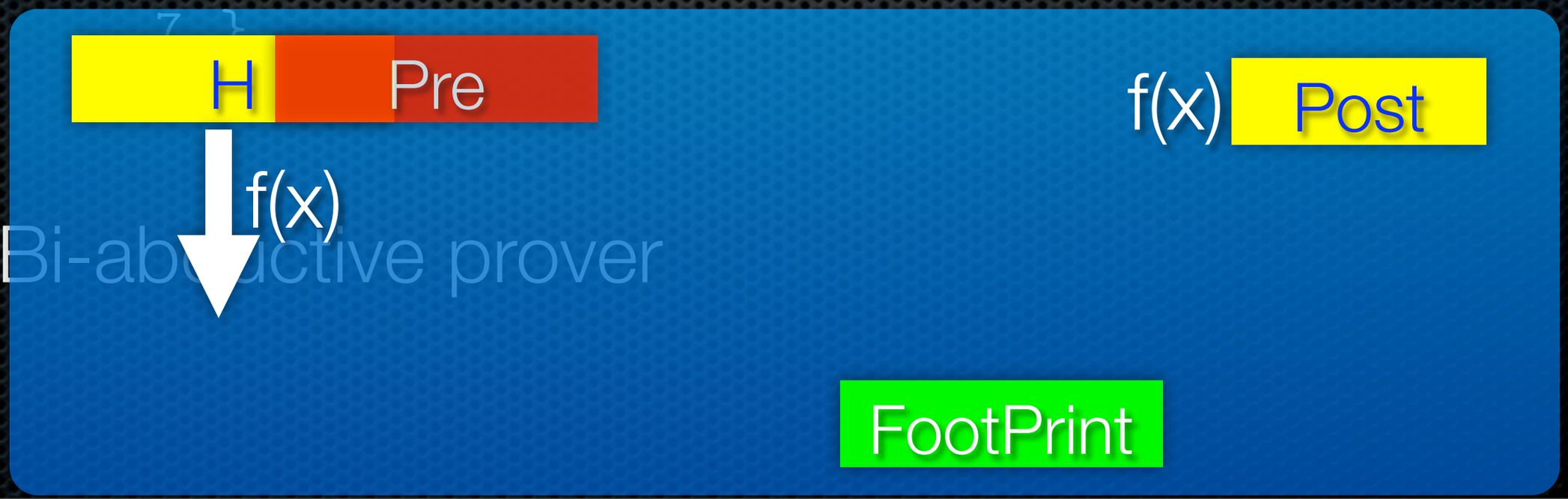
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

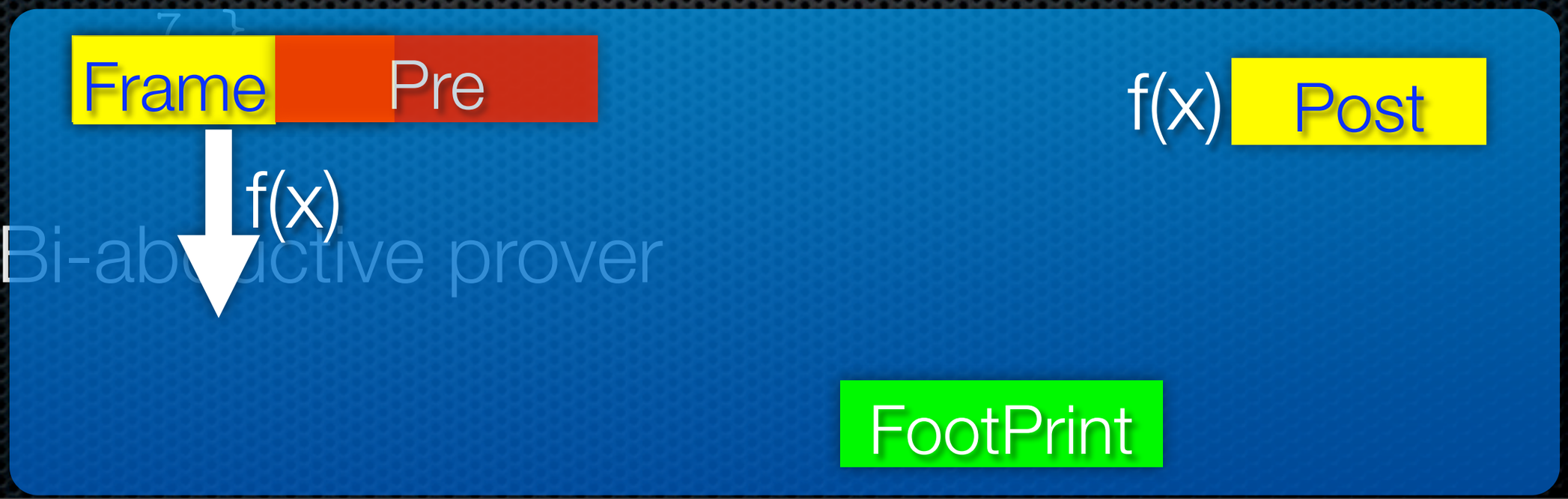
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

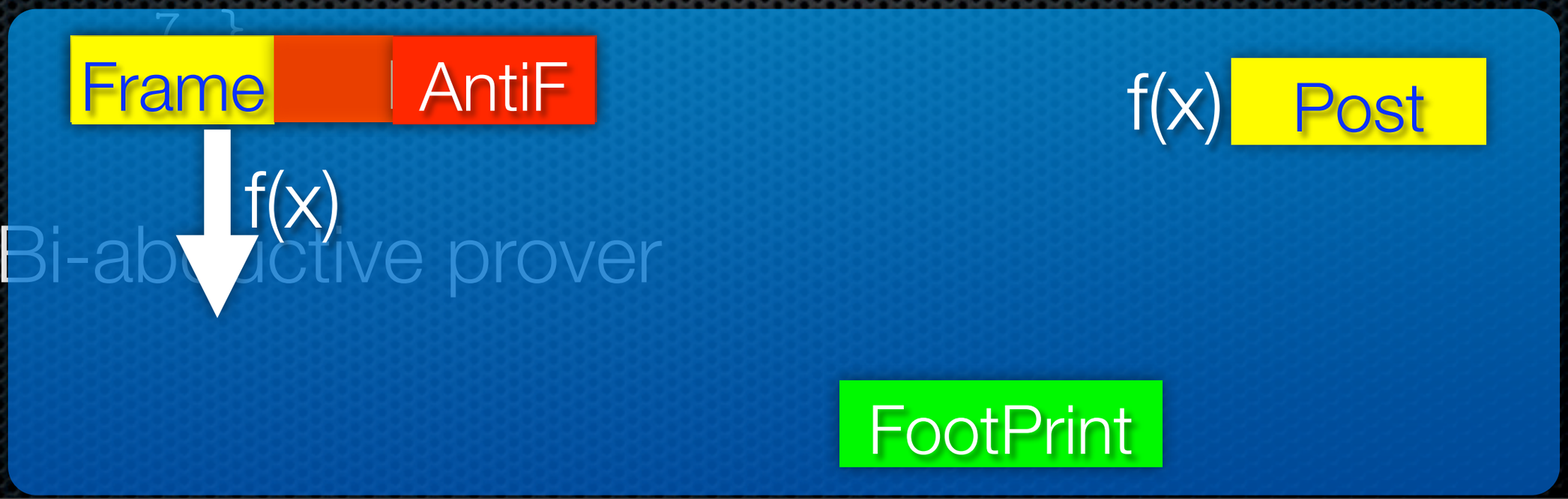
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

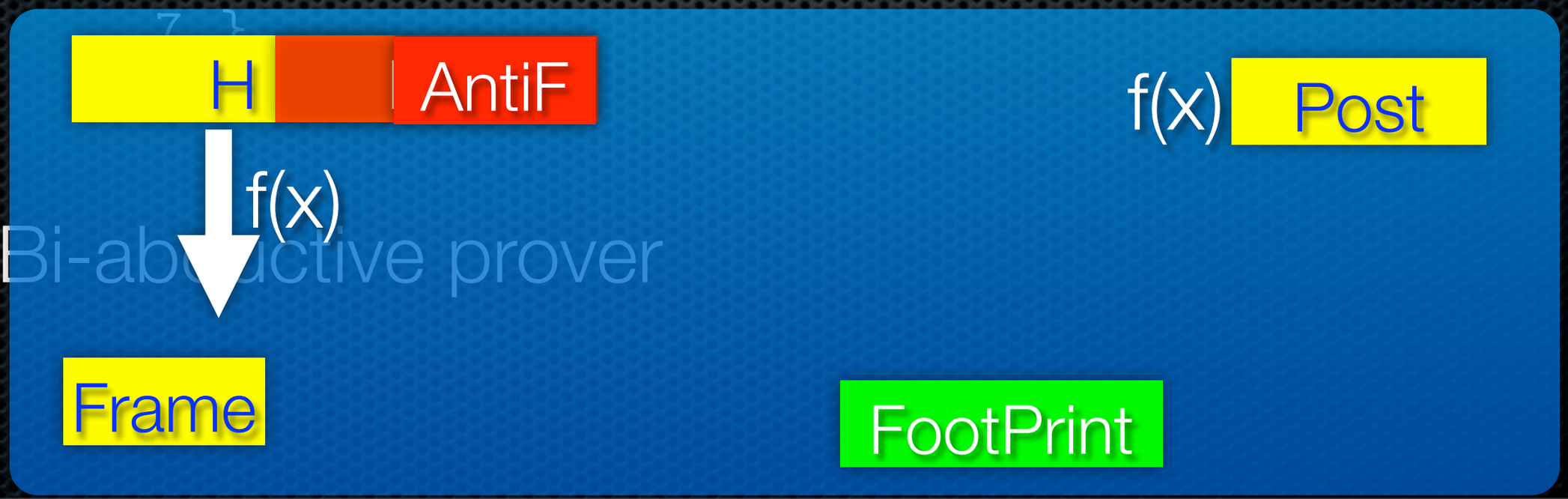
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

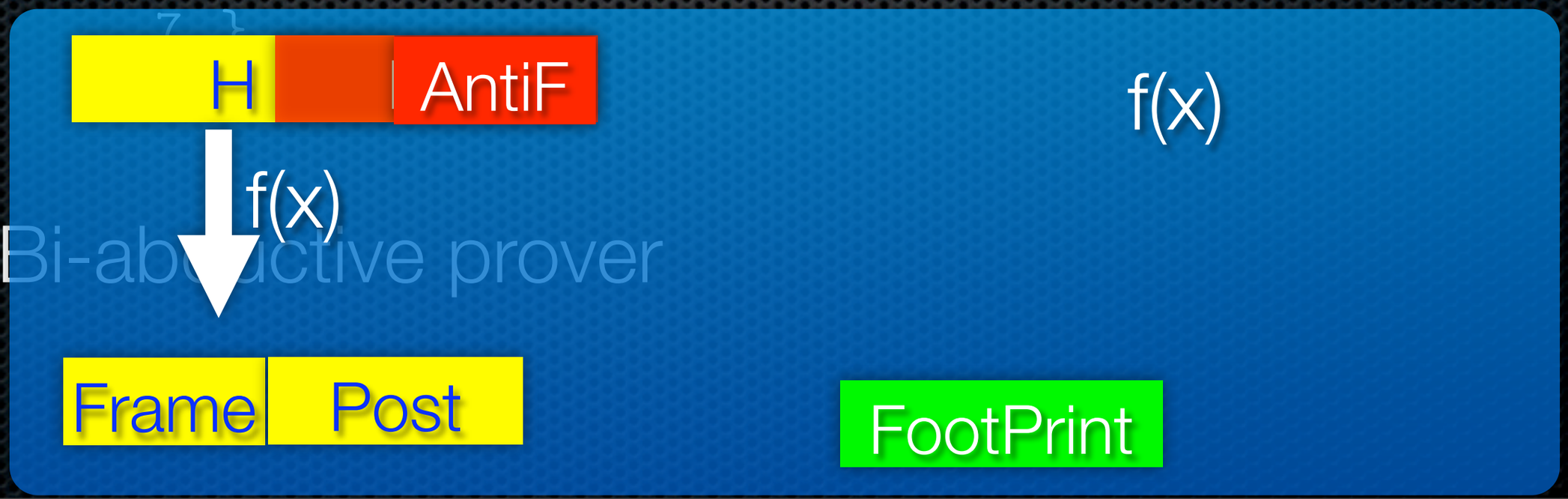
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-abductive prover

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-abductive prover

$$x \mapsto 0 * z \mapsto 0 * ?antiframe \vdash \mathsf{list}(x) * \mathsf{list}(y) * ?frame$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-abductive prover

$$x \mapsto 0 * z \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y) * z \mapsto 0$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

$\mathsf{list}(y)$

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

$\mathsf{list}(x) * z \mapsto 0$

Bi-abductive prover

$$x \mapsto 0 * z \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y) * z \mapsto 0$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

$\mathsf{list}(y)$

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

$\mathsf{list}(x) * z \mapsto 0$

Bi-abductive prover

$$\mathsf{list}(x) * z \mapsto 0 * ?antiframe \vdash \mathsf{list}(x) * \mathsf{list}(z) * ?frame$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

$\mathsf{list}(y)$

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

$\mathsf{list}(x) * z \mapsto 0$

Bi-abductive prover

$$\mathsf{list}(x) * z \mapsto 0 * \mathsf{emp} \vdash \mathsf{list}(x) * \mathsf{list}(z) * \mathsf{emp}$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail = 0;
4    z=malloc(sizeof(list_item)); z->tail = 0;
5    foo(x,y);
6    foo(x,z);
7  }
```

$\mathsf{list}(y)$

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

$\mathsf{list}(x) * z \mapsto 0$

$\mathsf{list}(x)$

Bi-abductive prover

$$\mathsf{list}(x) * z \mapsto 0 * \mathsf{emp} \vdash \mathsf{list}(x) * \mathsf{list}(z) * \mathsf{emp}$$

General Schema Compositional Analysis

For function in the program we compute tables of specs

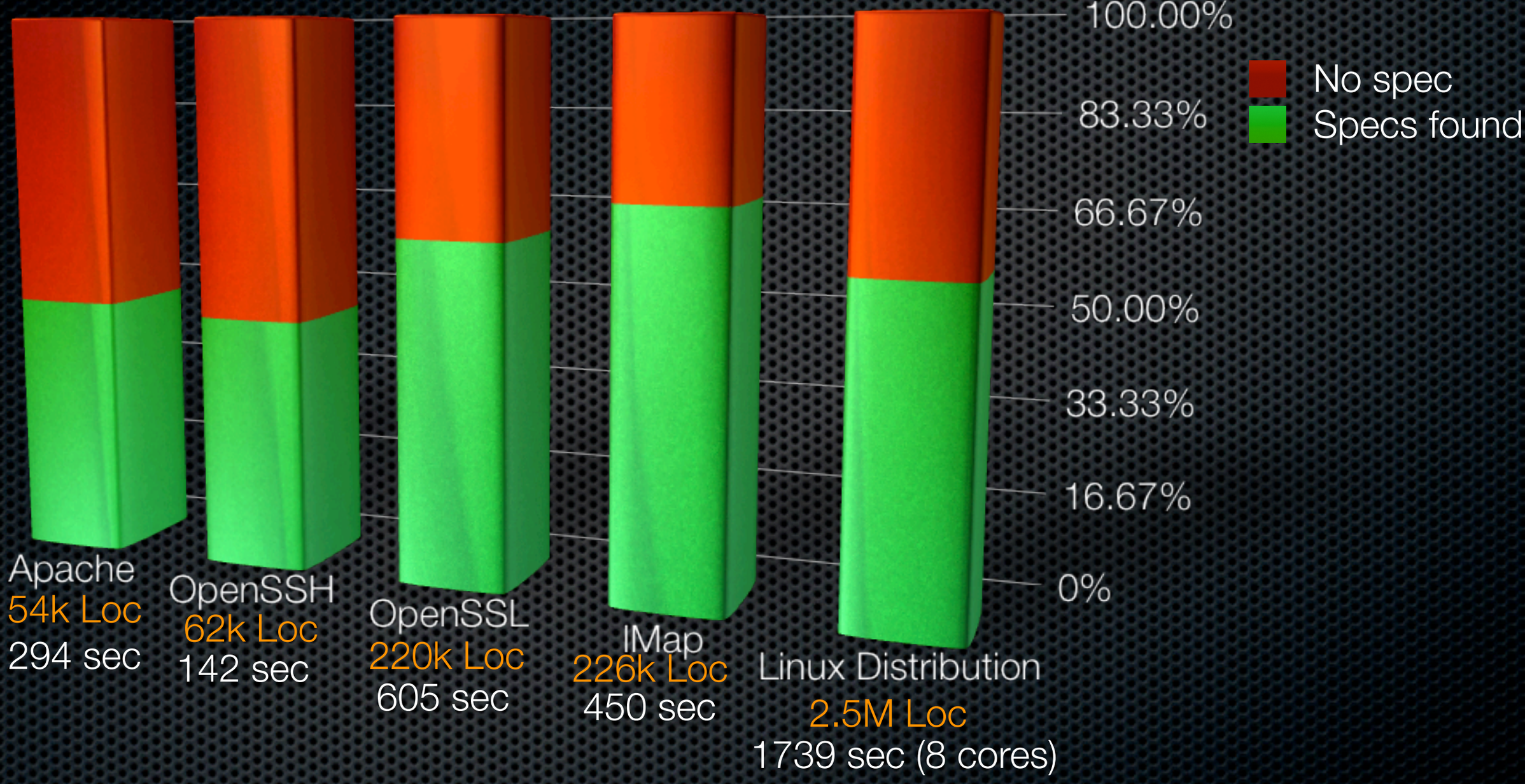
$$\{T_{f_1}, \cdots, T_{f_n}\}$$

Tables are sets of entries of type: $(pre, \{post_1, post_2, \dots\})$

The computation follows the call graph (start from leaves)

Recursive function are analyzed with an iterative method until it reaches fixed point

Running on really big code



Test for precision: run on Firewire device driver and small recursive procedures handling nested data structures

Nice features of compositional analysis

- Automatic
- Parametric on the abstract domains
  - better domain ==> better results
- Modular: Big/incomplete code
- Parallel Implementation (Multicore)
- Easily Incremental: reuse previous analyses results
- Support for other shape analyses (speculative)

The bi-abduction manifesto

- Frame inference $A \vdash B * X$ allows an analyzer to use small specs
- Abduction $A * X \vdash B$ helps to synthesize small specs
- Their combination, bi-abduction

$$A * X \vdash B * X'$$

helps to achieve compositional bottom-up analysis. Furthermore it brings the benefits of local reasoning (as introduced in Separation Logic) to automatic program verification

Canonical Form

$\Pi|\Sigma$ is in canonical form if and only if

- Π does not contain primed variables
- if $x' \in \Sigma$ then is reachable and
  - x′ is shared or
  - x′ points to a possible dangling variable or
  - x′ is possibly dangling
  - x′ is the internal point of a cycle of length 2

Proposition: If the number of program variables is finite then the set of canonical forms CSH is finite.

Homework: try to prove it.