The design of SecreC for Sharemind 3

Peeter Laud, Jaak Randmets

Dan Bogdanov, Roman Jagomägis, Jaak Ristioja

Estonian CS Theory Days

Kubija

27.-29.01.2012

This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.
"A" (Approved for Public Release, Distribution Unlimited)

The virtual computer

- Stores a bunch of values. Each value has
  - a protection domain kind (PDK)
  - a data type
- Implements a bunch of protocols for various operations on data of various types in various PDKs
  - Each protocol / algorithm identified as a syscall
- Learns from its program, which syscalls to invoke on which data in which order

Features of the high-level language

- Imperative language (While) with procedures
- PDKs, protection domains (PD)
- Data structures (currently arrays only)
- Data types annotated with PDs
  - only atomic data
- **PD-polymorphism** for procedures
- Procedure overloading and specialization for particular PDKs
- Control flow is public

A basic code example

```
kind public;
domain public public;
kind additive3p;
domain private additive3p;
```

```
template<domain T1:additive3p>
  T1 int read_int() =
  "additive3p_read_int"
```

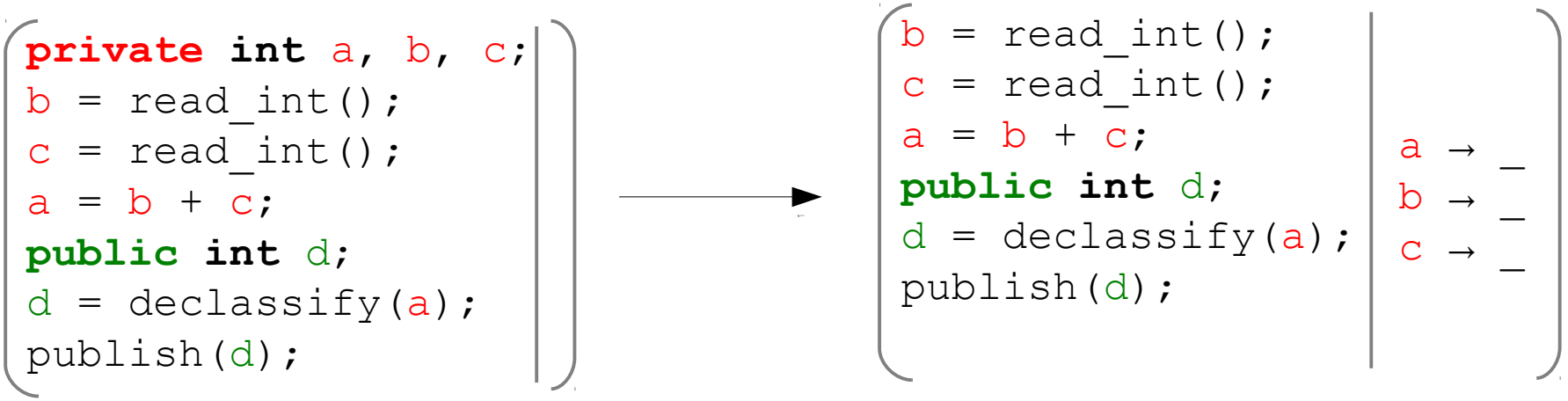
```
void main () {
  private int a, b, c;
  b = read_int();
  c = read_int();
  a = b + c;
  public int d;
  d = declassify(a);
  publish(d);
}
```

```
template<domain T1:additive3p>
  T1 int operator+(T1 int x, T1 int y) =
  "additive3p_add_ints"
```

```
template<domain pub:public>
  void publish(pub int x) =
  "print_int"
```

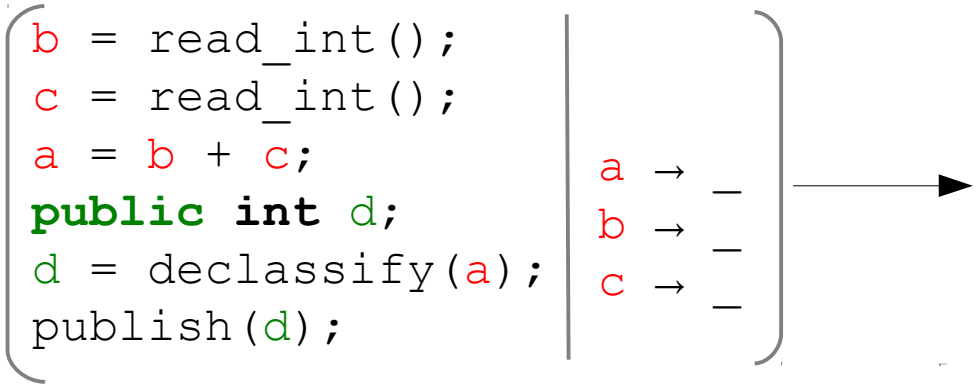
```
template<domain T1:additive3p, domain T2:public>
  T2 int declassify(T1 int x) =
  "declassify_additive3p_public_int"
```

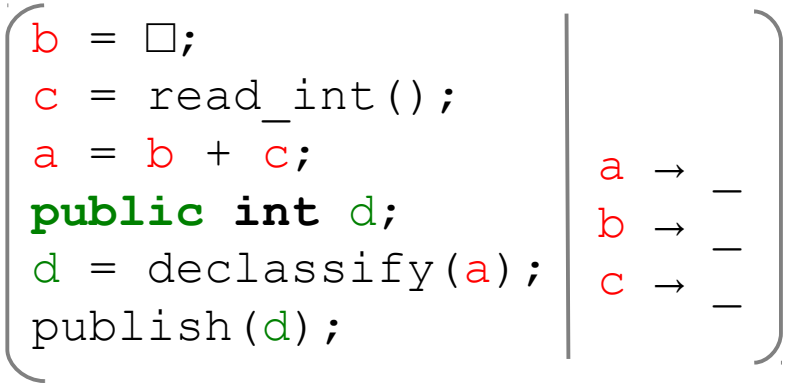
Small-step semantics



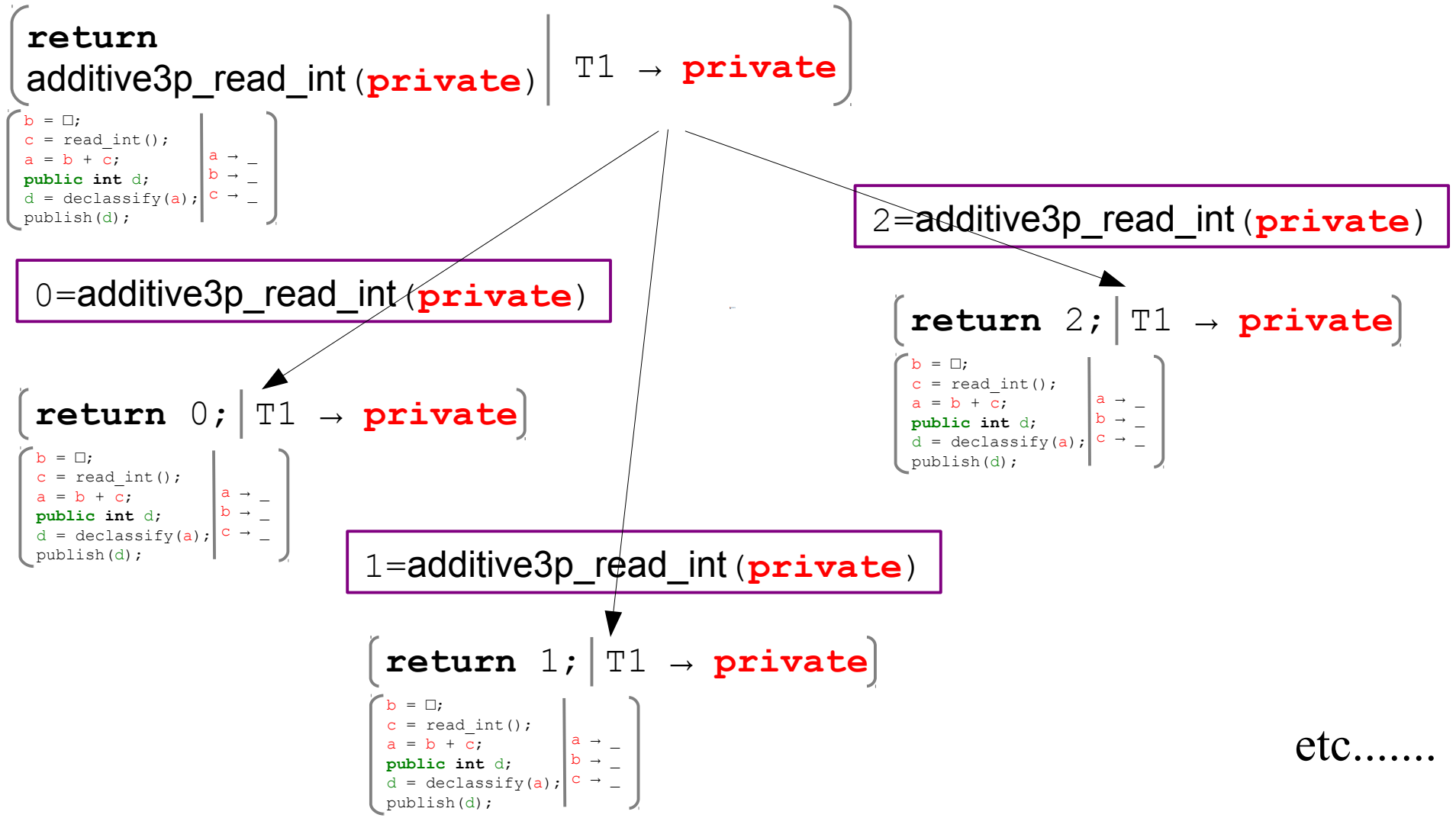
```
template<domain T1:additive3p>
  T1 int read_int() =
  "additive3p_read_int"
```

Small-step semantics

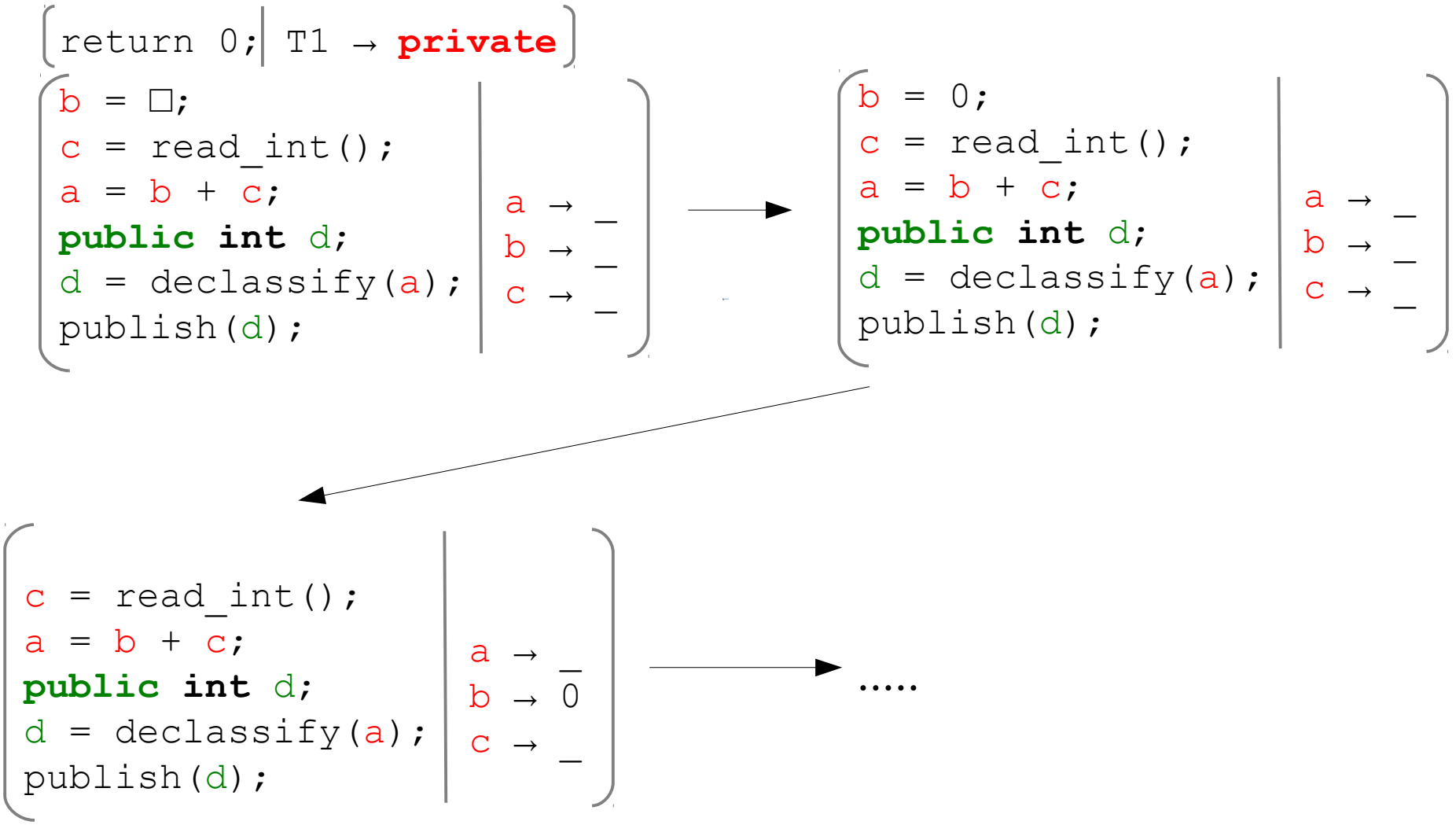




Small-step semantics



Small-step semantics



Feature: PD-polymorphism

```
template<domain T1, domain T2, domain T3>
T1 int [[1]] operator* (T2 int[[1]] x, T3 int[[1]] y) {
  T1 int [[1]] result (size(x));
  public int i;
  assert(size(x) == size(y));
  for (i = 0; i < size(x); i++) {
    result[i] = x[i] * y[i];
  }
  return result;
}
```

- In runtime, `T1,T2,T3` are instantiated with concrete protection domains
- These are used to locate the correct `T1 int operator*(T2 int, T3 int)`
- The success of locating is determined at compile-time

Feature: procedure overloading

```
template<domain T1:public>
T1 int operator*(T1 int x, T1 int y)= “public_mult”

template<domain T1:additive3p>
T1 int operator*(T1 int x, T1 int y)= “additive3p_mult_int”

template<domain T1:fhe>
T1 int operator*(T1 int x, T1 int y)= “fhe_mult_int”

template<domain T1:additive3p, domain T2:public>
T1 int operator*(T1 int x, T2 int y)=
“additive3p_mult_int_const”
```

Issues with overloading

```
template<domain T1:fhe, domain T2:fhe>
T1 int reclassify(T2 int x) = "fhe_reclassify_int"

template<domain T1>
T1 int reclassify(T1 int x) { return x; }
```

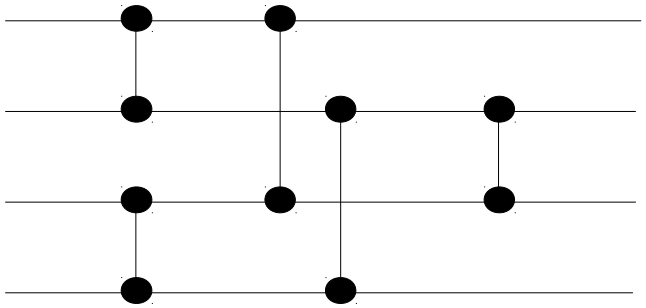
```
kind fhe;
domain private fhe;
.....
private int x, y;
.....
y = reclassify(x);
```

- Which `reclassify` should we call?
- We introduce priorities in an ad-hoc manner

Expected use of polymorphism in development

- For each PDK, a signature file associates arithmetic and other operations with syscalls
- Each PDK may have a different set of available operations
- A “standard library” provides low-priority implementations for most operations
- Most code is written PD-polymorphically
- Main program fixes PDKs of data
- Compiler checks that necessary operations are present in used PDKs

Example: sorting



- A comparator is c(x,y) = (min(x,y),max(x,y))
- c(x,y) ≡ **let** b = lt(x,y) **in** (x*b+y*(1-b),x*(1-b)+y*b)
  - Here lt(x,y)=1 if x<y; lt(x,y)=0 if x≥y
- Sorting networks are built from comparators
  - Performed comparisons are independent of starting order
- Best practical sorting networks have $O(n \log^2 n)$ comparators in $O(\log^2 n)$ rounds
- Let `sortnet(n)` output a description of a sorting network for `n` inputs

Generic sorting routine

```
template<domain PD>
PD int[[1]] sort(PD int[[1]] src) {
  public int alength, i;
  public int[[2]] sn;
  PD int x, y, b;

  alength = size(src);
  sn = sortnet(alength);
  for(i=0; i < size(sn[0]); i = i+1) {
    x = src[sn[i,0]];
    y = src[sn[i,1]];
    b = lt(x, y);
    src[sn[i,0]] = x*b + y*(1-b);
    src[sn[i,1]] = x*(1-b) + y*b;
  }
  return src;
}
```

Sorting in additive3p PDK

- additive3p has a particularly efficient shuffle
  - template<domain PD> PD int[[1]] shuffle(PD int[[1]] src) = “additive3p_shuffle_ints”
- In a shuffled array, comparison results are random
  - Hence it's OK to make them public
  - The control flow can then depend on them

Sorting routine for additive3p

```
template<domain PD:additive3p>
PD int[[1]] sort(PD int[[1]] src) {
  ... variable declarations ...

  src = shuffle(src);

  Sort src. For comparing positions a and b use
     declassify(lt(src[a],src[b]))

  return src;
}
```

- Quicksort can be done with O(n log n) work in O(log n) rounds.

Semantics

- Configuration: list of stack frames
- Stack frame: <program part, values of var.s>
- Labeled transition system over configurations
  - "most" transitions have empty labels
  - Only transitions corresponding to syscalls have non-empty labels
  - Syscall "f" with arguments $v_1,...,v_n$ in PDs $d_1,...,d_n$ returning a value in PD d_0 has a label

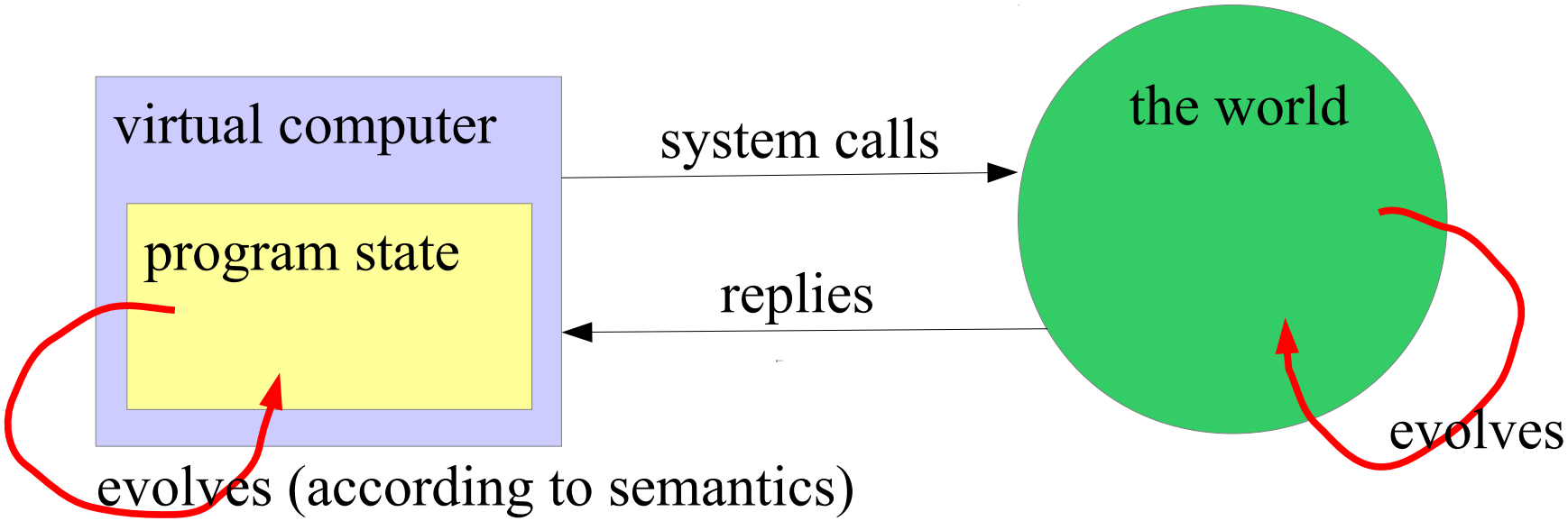
$$w = f(v_1,...,v_n,d_0,d_1,...,d_n)$$

  - w picked non-deterministically

Collecting syscalls

- The collecting semantics [[P]] of a program P is a set of sequences of labels
- Each sequence corresponds to a possible trace in the LTS given by the operational semantics of P
  - Empty labels are invisible in this sequence

Implementation of syscalls



$\mathcal{W}$ — set of all possible states of the world

the semantics of an n-ary syscall $f(v_1, \ldots, v_n; d_0, d_1, \ldots, d_n)$ has the type $[\![f]\!] : \mathbf{Val}^n \times \mathbf{PD}^{n+1} \times \mathcal{W} \to \mathbf{Val} \times \mathcal{W}$

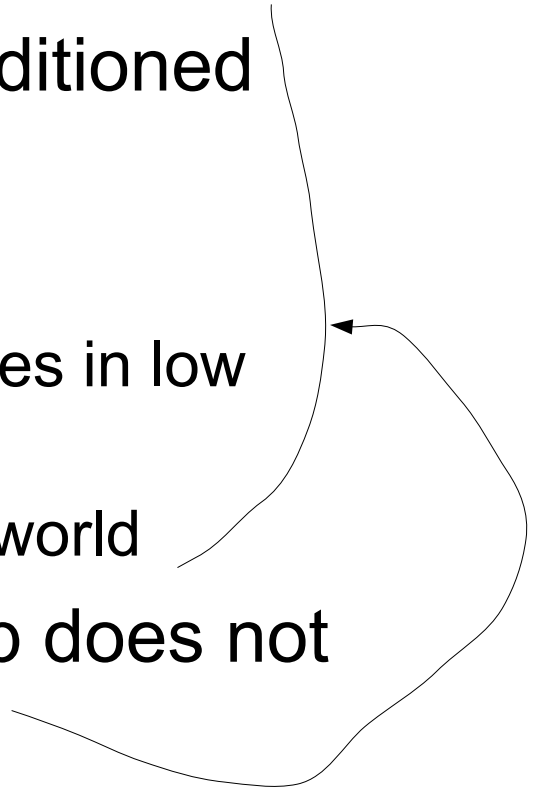
The evolution of the system is determined by the initial state of the world

Security of information flow

- Let $\mathfrak{PD}$ be partitioned to $\mathfrak{PD}_L$ [$\mathfrak{PD}_H$
- DW – distribution of the init. state of the world
  - $\mathcal{W} \cong \mathcal{W}_{\mathrm{L}} \times \mathcal{W}_{\mathrm{H}}$
  - DW keeps low and high component independent
- If $((w'_{\mathrm{L}}, w'_{\mathrm{H}}), v_0) = [\![f]\!](v_1, \ldots, v_n; d_0, \ldots, d_n; (w_{\mathrm{L}}, w_{\mathrm{H}}))$ then
  - w^0_L depends only on w_L and v_i, where $d_i 2 \mathfrak{PD}_L$
  - v_0 depends on w_H only if $d_0 2 \mathfrak{PD}_H$
- A syscall is declassifying if $d_0 2 \mathfrak{PD}_L$ and $d_i 2 \mathfrak{PD}_H$ for some i.

Security of information flow

- Let the program be running
- Consider the distribution DW_H conditioned over the *current* low-knowledge:
  - the program path
  - the current and past values of variables in low domains
  - the current and past low-state of the world
- **Theorem**. A non-declassifying step does not change the conditional distribution



Features of the intermediate language

- Imperative language (While) with procedures
- PDKs
- Data structures (currently arrays only)
- Data types annotated with PDKs
  - only atomic data
- Monomorphic procedures
- In syscalls: string constants instead of PDs
- **No interesting features really...**

Semantics of intermediate language

- SOS of While with procedures
- Syscalls are labeled and nondeterministic:

$$\texttt{syscall}(f; v_1, \ldots, v_n; d_0, \ldots, d_n) \xrightarrow{w=f(v_1,\ldots,v_n;d_0,\ldots,d_n)} w$$

  - w ranges over all possible values in the return type of f
  - d_i are strings here
- Other steps have empty labels
- We again get a set of sequences of labels
  - the world assigns a probability to each sequence

Translating from high-level to intermediate language

- variables / arguments of type `string` record the protection domains of variables / arguments / return values
- PDK-polymorphic functions are translated into several copies

Correctness of translation

- Semantics – set of sequences of labels
- This is preserved during translation

Preservation of security

- Secure information flow in terms of PDs cannot easily be defined for the intermediate language because there are no PDs
- **But:** our translation preserves the set of sequences of labels
- No world can distinguish a program from its translation
- In this sense, the noninterference properties are preserved