Glamdring: Automatic Application Partitioning for Intel SGX

Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, Pierre-Louis Aublin, and Florian Kelbert, Tobias Reiher, David Goltzsche, David Eyers, Rudiger Kapitza, Christof Fetzer, Peter Pietzuch

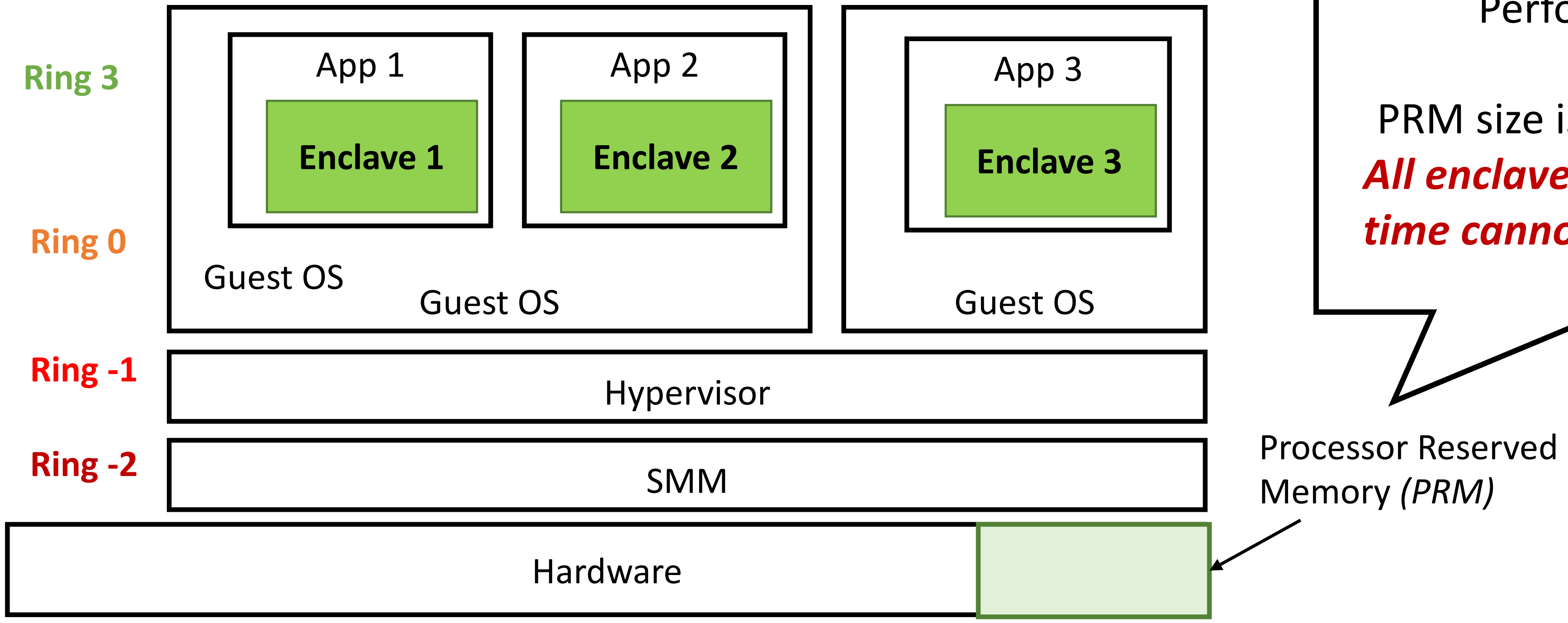
Presented by Mengjia Yan

MIT 6.888 Fall 2020

Based on slides from Divya Muthukumaran

Background On Intel SGX, Enclave

- On commodity processors starting with Skylake
- 18 CPU instructions to manage enclave lifecycle
  - ECREATE, EENTER, EEXIT, EADD, EEXTEND, EINIT, etc.



Performance issues.

PRM size is 128MB in SGX V1.0
All enclaves loaded at the same time cannot exceed said ~90MB

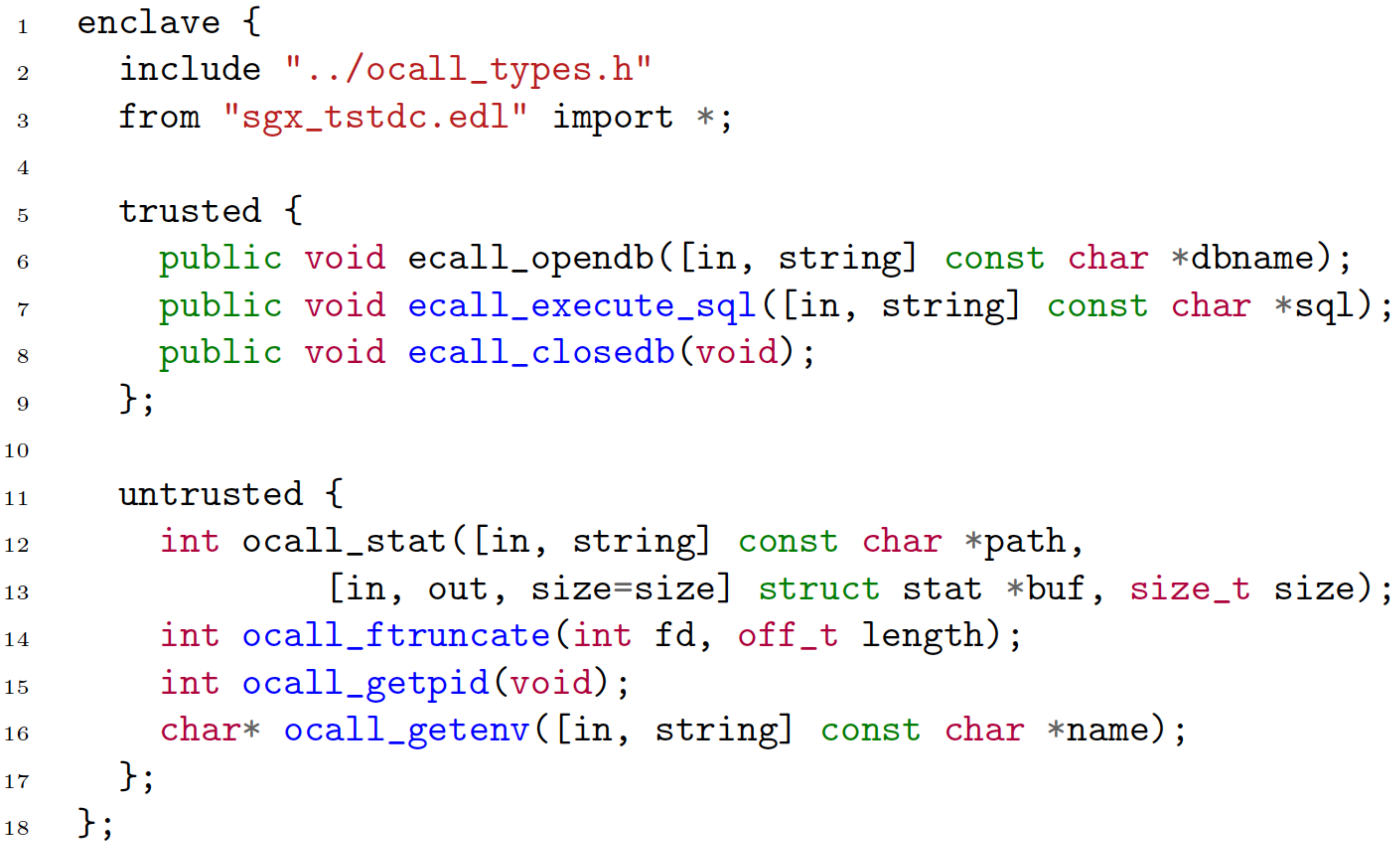
Programming Intel SGX

- Platform software (PSW)
  - SGX runtime, contains drivers, services, DLLs and ***privileged*** enclaves
  - Required to use Intel SGX

- Software Development Kit (SDK) for Linux and Windows
  - SGX libraries: Intel-custom libc and crypto libraries, sgx-specific libs
  - Tools
    - sgx_edger8r: takes an EDL file and generates glue (C code and headers)
    - sgs_sign to sign code with dev key

- Developer guide
  - https://download.01.org/intel-sgx/sgx-linux/2.11/docs/Intel_SGX_Developer_Guide.pdf

An Example EDL File



```
enclave {
  include "../ocall_types.h"
  from "sgx_tstdc.edl" import *;

  trusted {
    public void ecall_opendb([in, string] const char *dbname);
    public void ecall_execute_sql([in, string] const char *sql);
    public void ecall_closedb(void);
  };

  untrusted {
    int ocall_stat([in, string] const char *path,
            [in, out, size=size] struct stat *buf, size_t size);
    int ocall_ftruncate(int fd, off_t length);
    int ocall_getpid(void);
    char* ocall_getenv([in, string] const char *name);
  };
};
```

A part of the EDL file from SGX-SQLite

Threat Model

- Following SGX threat model
  - Attacker can be privileged software that can access or modify data in memory or disk
  - Confidentiality:
    - SGX encrypt data in DRAM
    - MMU disable accesses to PRM outside of enclave
  - Integrity:
    - SGX computes and verifies hash of data in PRM

- New attack vectors in this paper
  - Iago attacks: need to validate return value from the untrusted world

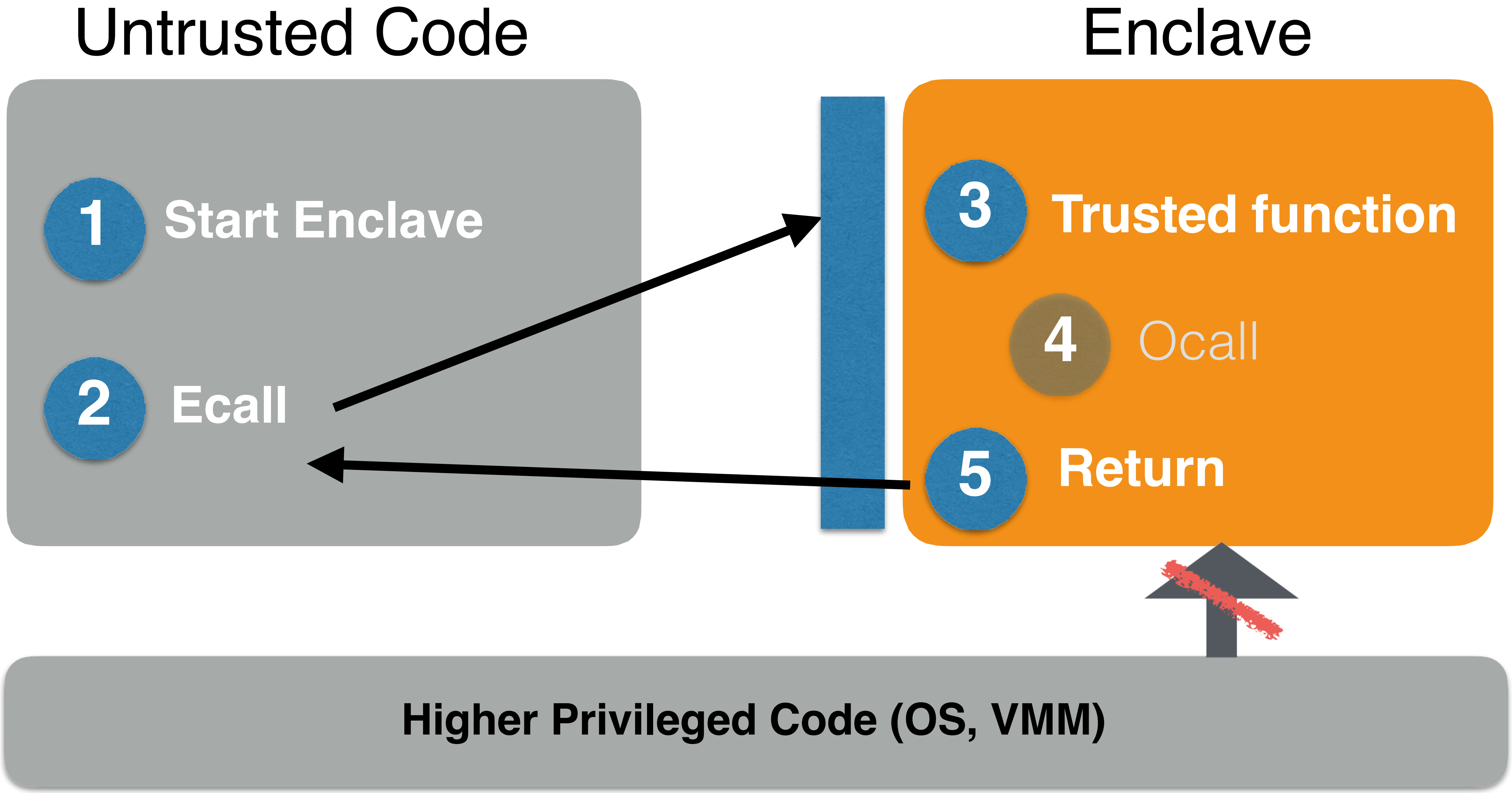
- Not considered
  - Denial of service attacks
  - Side channel attacks

Challenges of Developing Enclave Apps

1. How to partition applications into trusted and untrusted components
    - The trusted component (inside enclave) can not use syscalls and certain instructions

2. How to validate untrusted inputs (the OS cannot be trusted)

Enclave Application Lifecycle



Untrusted Code

Enclave

1 Start Enclave

2 Ecall

3 Trusted function

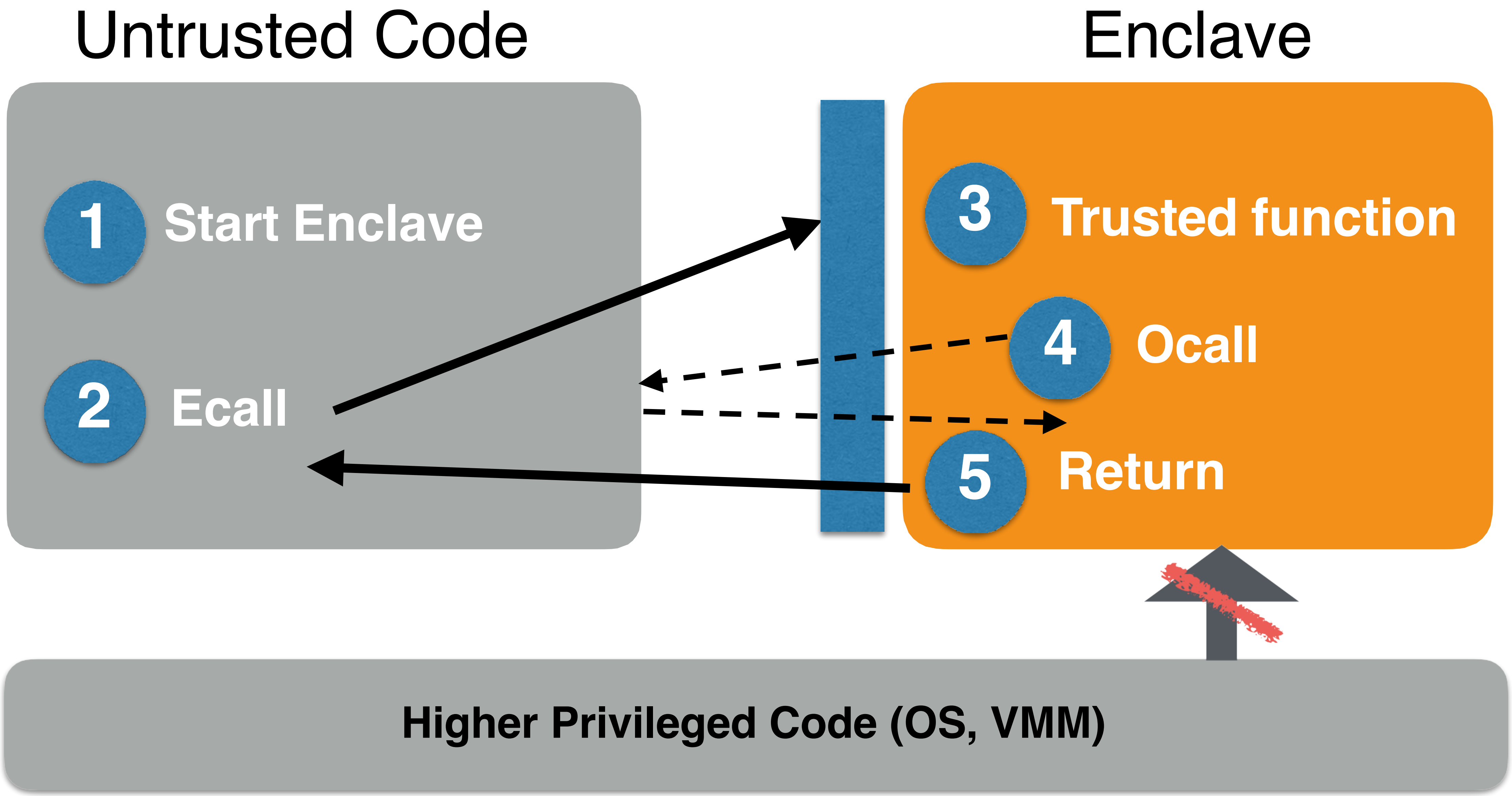
4 Ocall

5 Return

Higher Privileged Code (OS, VMM)

Enclave Application Lifecycle

Enclave crossings through ecalls and ocalls incur a performance penalty

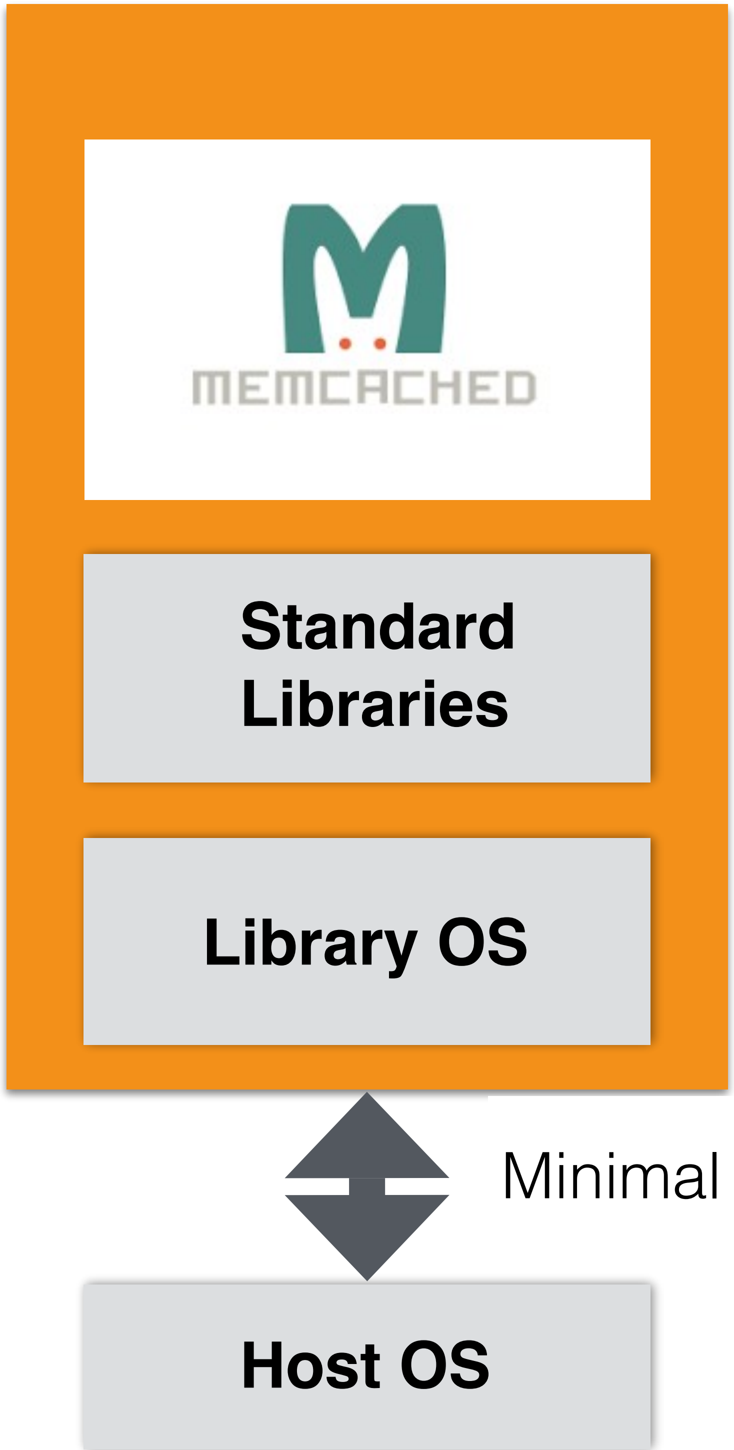


Glamdring Overview

Writing enclave applications to trade-off among

- Modification of applications (porting overhead)
- Interface complexity
- TCB size
- Performance

Library OS Inside Enclaves



Standard Libraries

Library OS

Host OS

Minimal system calls

Pros

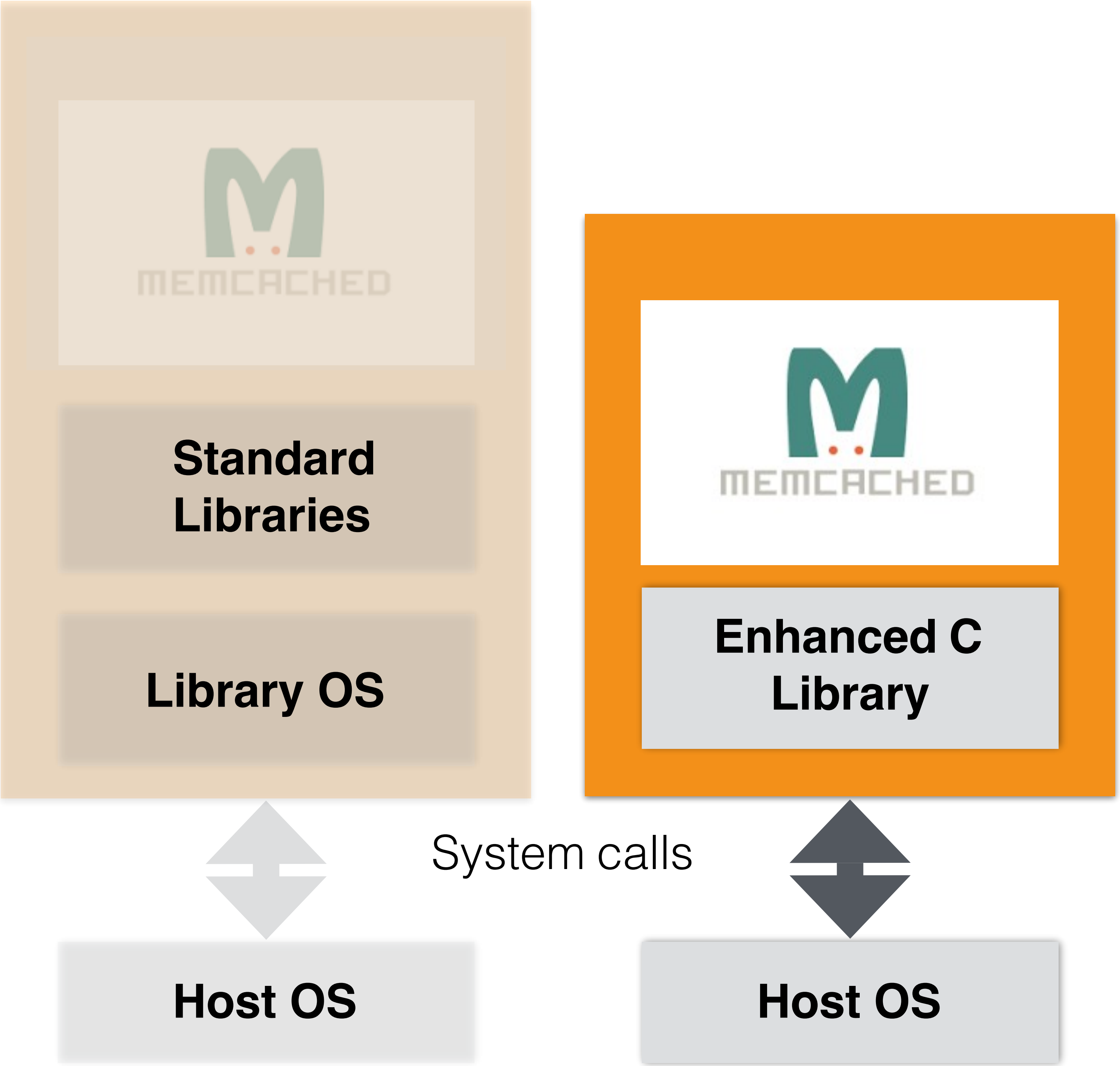
- Run unmodified applications
- Fixed shielded interface

Cons

- TCB is millions LoC!
- Performance overhead

Haven [OSDI'14]

Standard Library Inside Enclaves



Pros

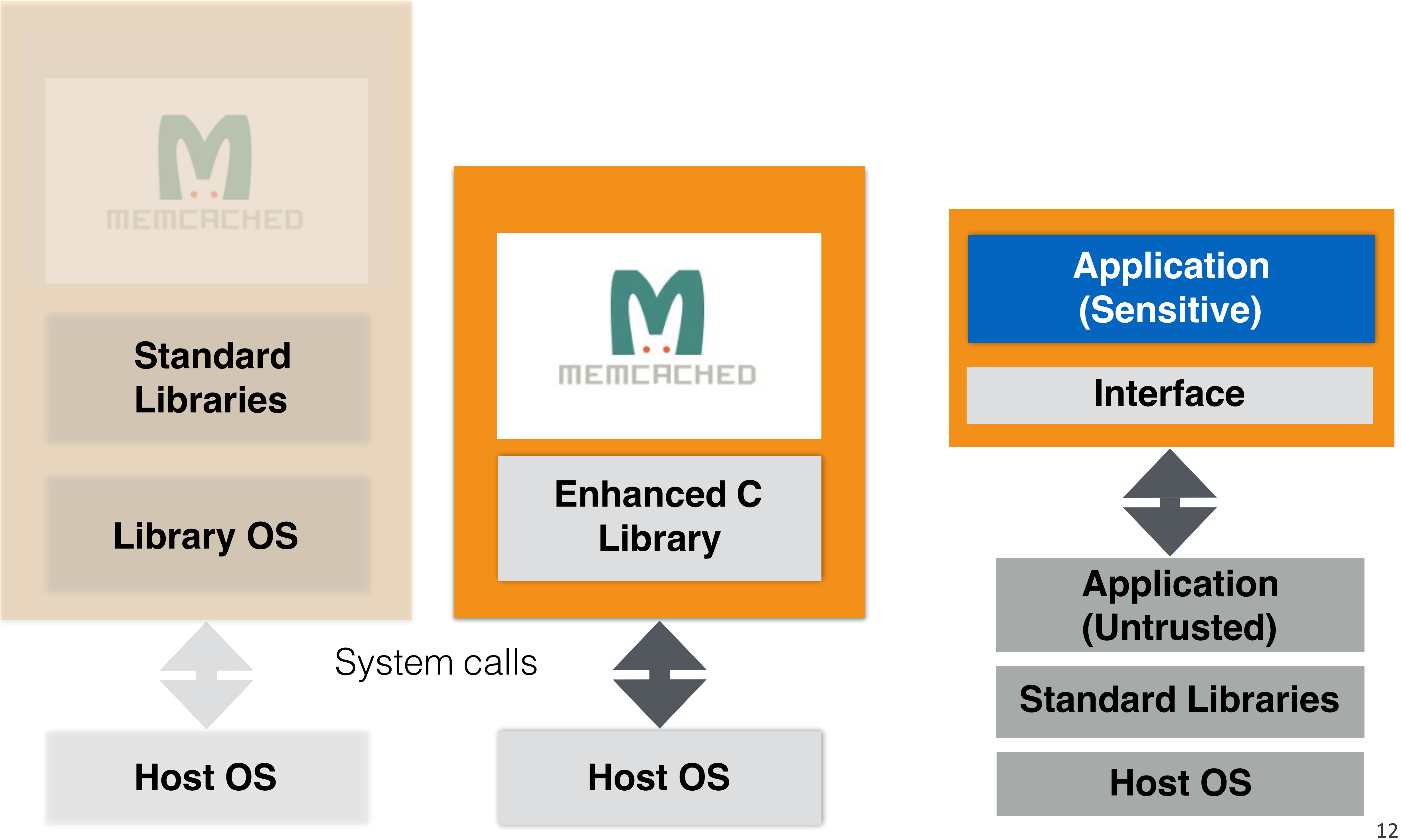
- Smaller TCB than Haven
- Fixed shielded interface

Cons

- TCB = 0.6x–2x of application size
- Recompilation needed

SCONE [OSDI'16]

Minimum TCB Inside Enclaves



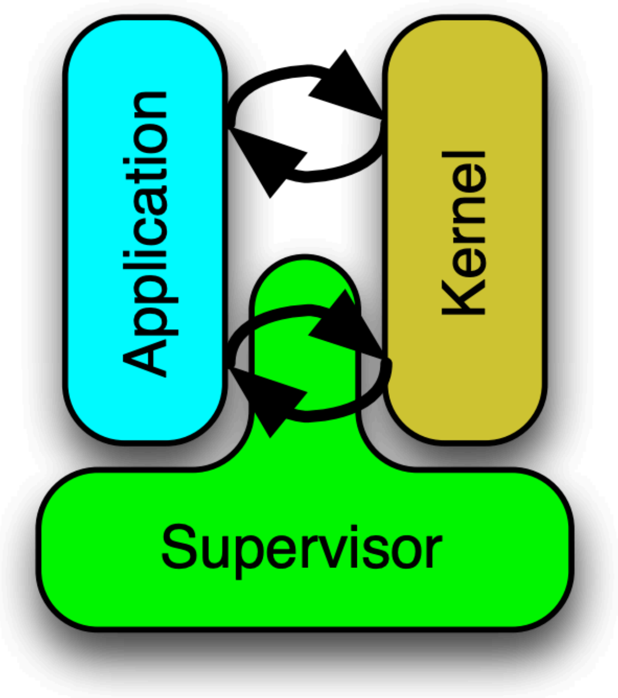
Strengths and Weakness

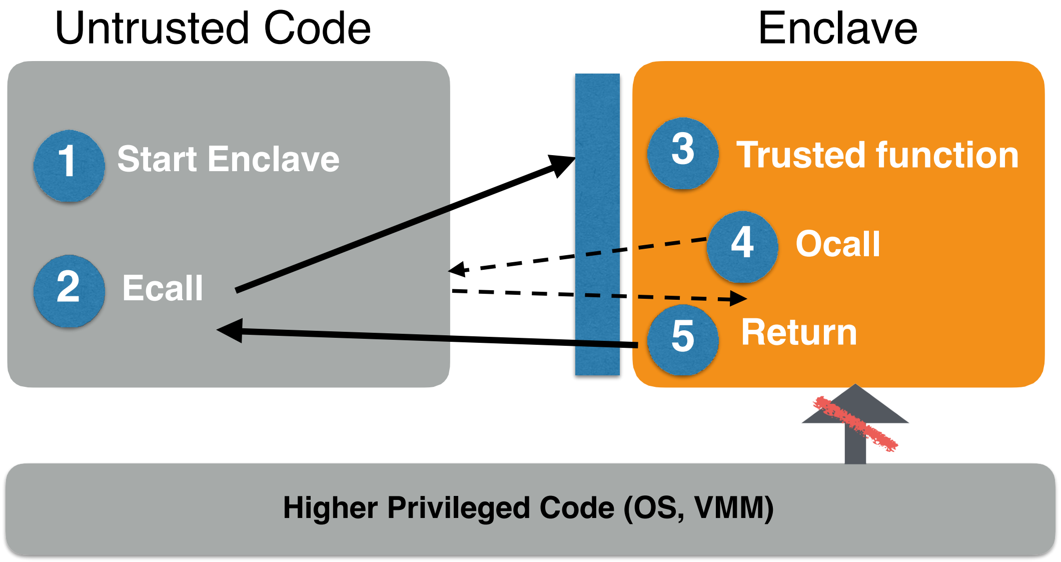
- Writing enclave applications to trade-off among
  - Modification of applications (porting overhead) ➡
  - Interface complexity ⬆
  - TCB size ⬆
  - Performance ↘

Security ?

Iago Attacks and COIN attacks

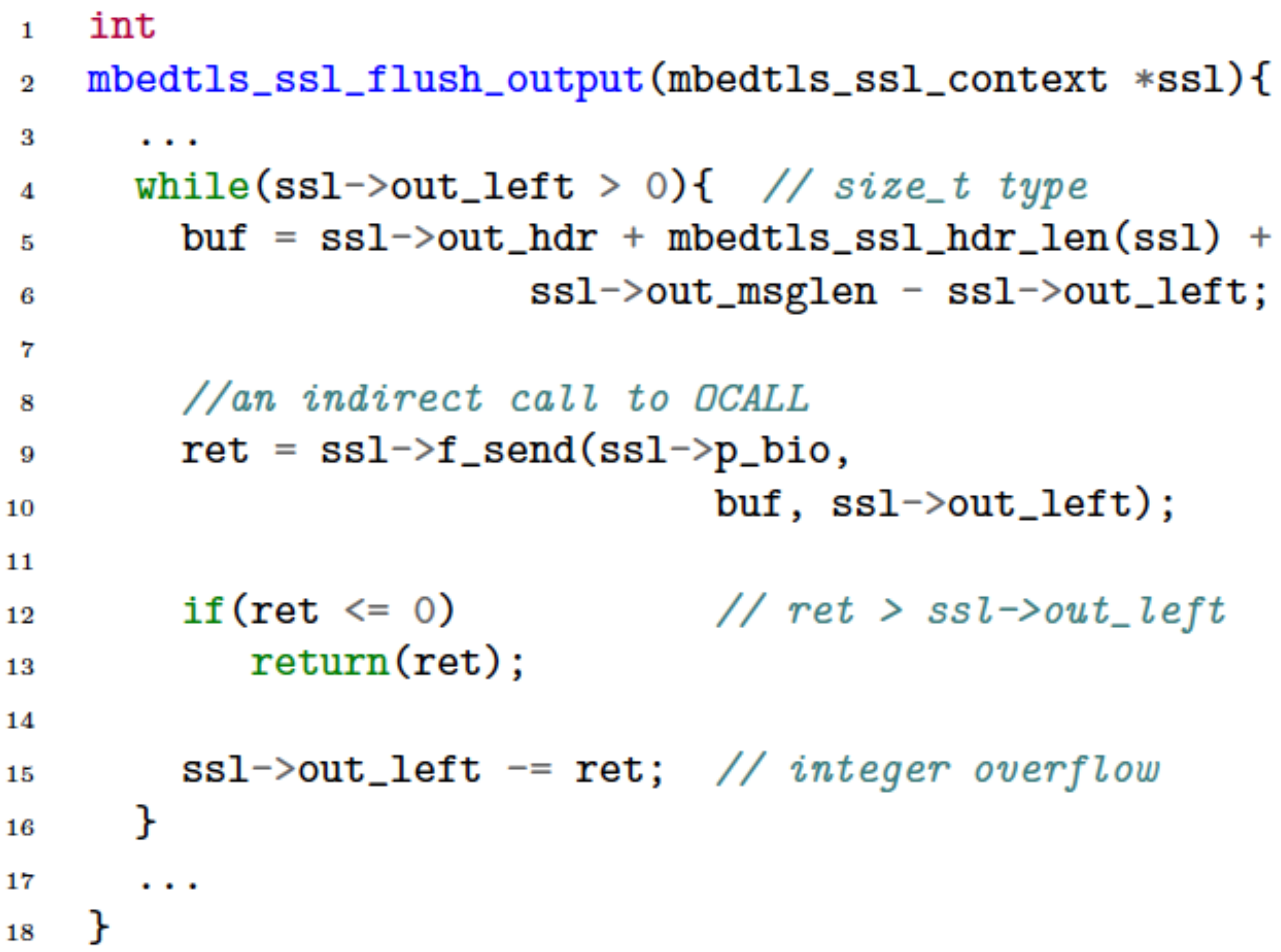
- Iago attacks: carefully chosen sequence of integer return values to Linux system calls → application executes astray
- COIN attacks: trigger ECALLs in an unexpected order and force incorrect return values of OCALLs → information leakage, control flow hijacking, etc.





Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface; Checkoway et al. ASPLOS'13
COIN Attacks: On Insecurity of Enclave Untrusted Interfaces in SGX; Khandaker et al. ASPLOS'20

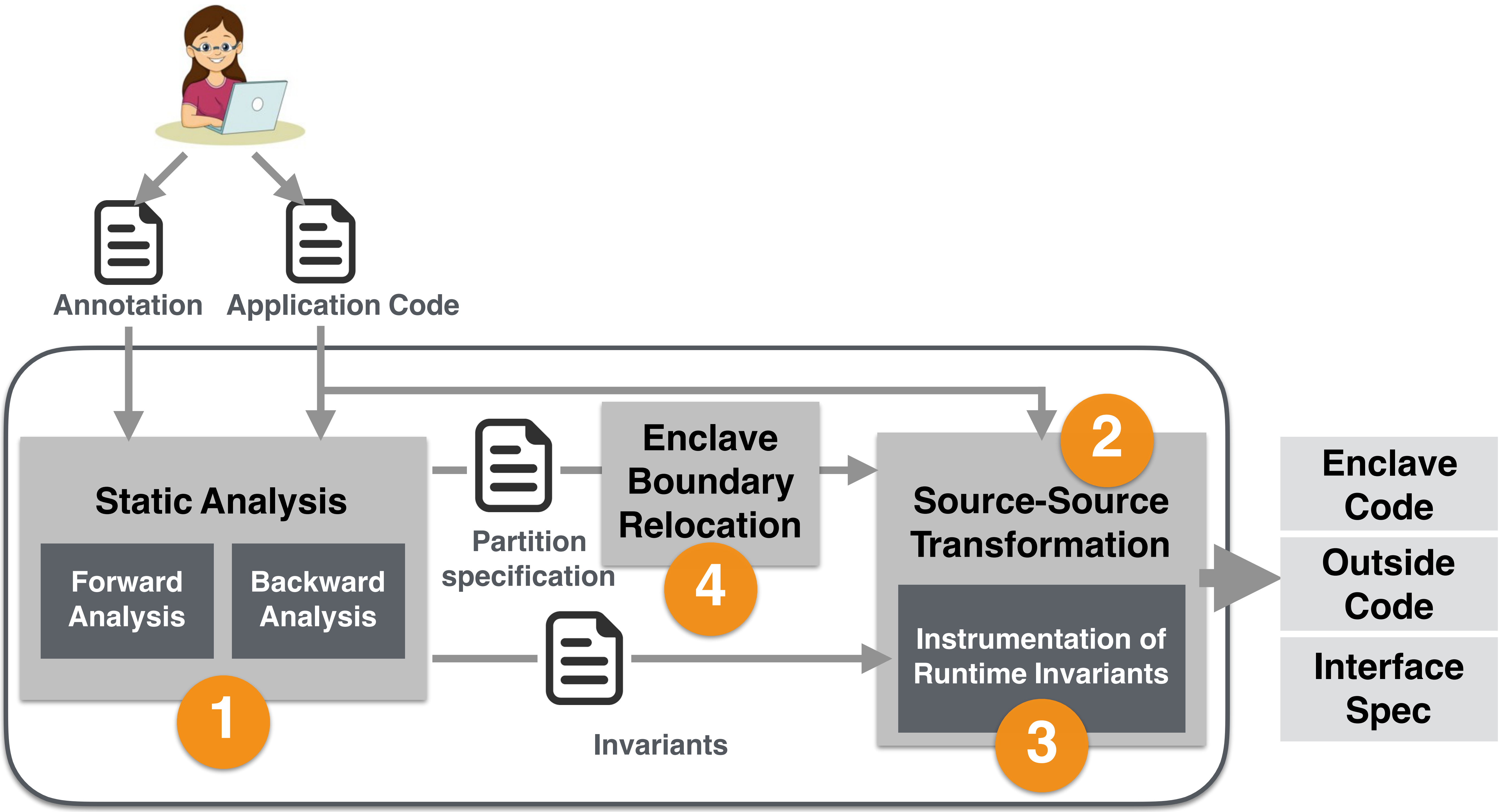
Example



```
int
mbedtls_ssl_flush_output(mbedtls_ssl_context *ssl){
  ...
  while(ssl->out_left > 0){  // size_t type
    buf = ssl->out_hdr + mbedtls_ssl_hdr_len(ssl) +
                   ssl->out_msglen - ssl->out_left;

    //an indirect call to OCALL
    ret = ssl->f_send(ssl->p_bio,
                           buf, ssl->out_left);

    if(ret <= 0)           // ret > ssl->out_left
       return(ret);

    ssl->out_left -= ret;  // integer overflow
  }
  ...
}
```

Example heap information leak from mbedTLSSGX.

Glamdring Partitioning Framework



Annotation

Application Code

Static Analysis

Forward Analysis

Backward Analysis

1

Partition specification

Enclave Boundary Relocation

4

Invariants

Source-Source Transformation

2

Instrumentation of Runtime Invariants

3

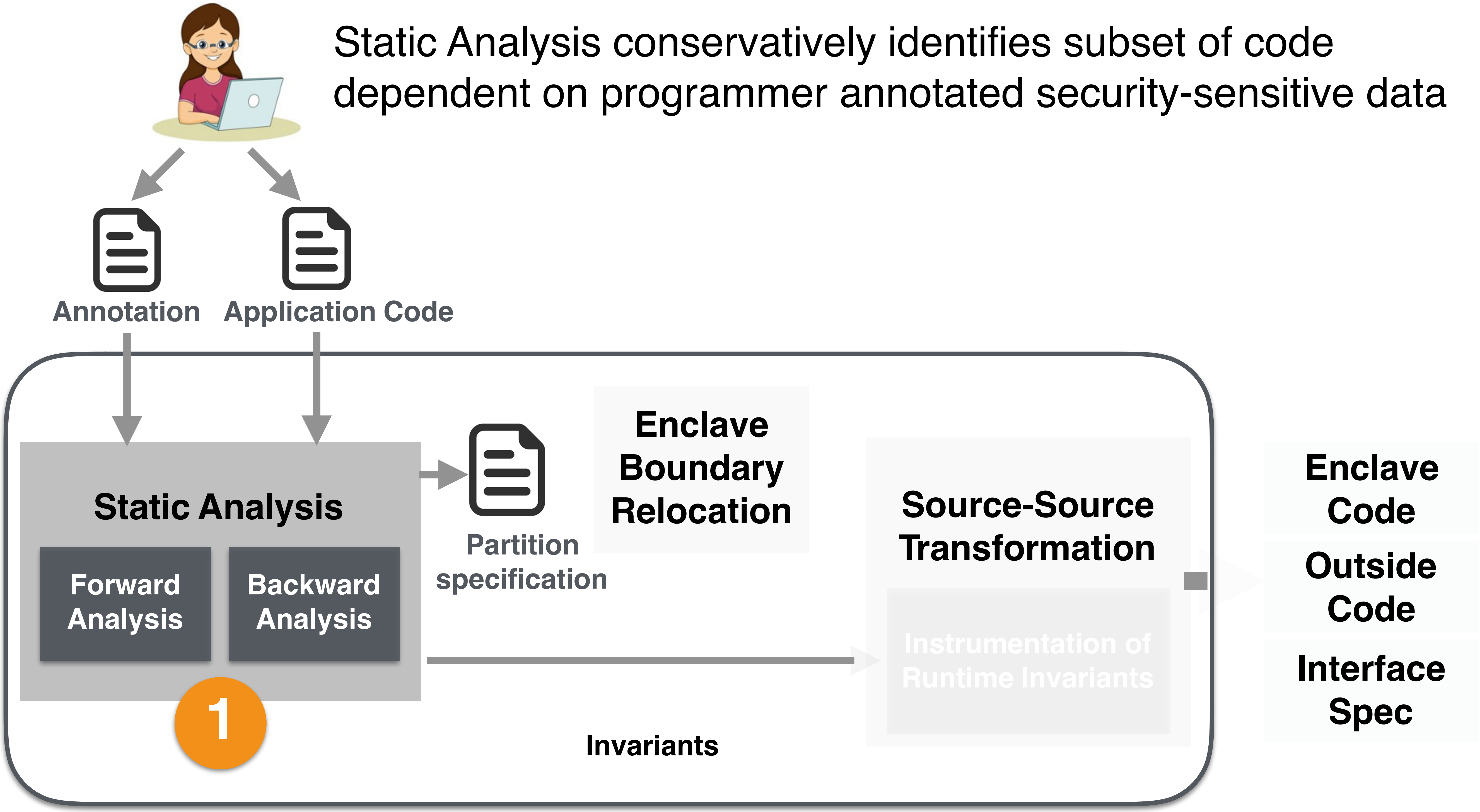
Enclave Code

Outside Code

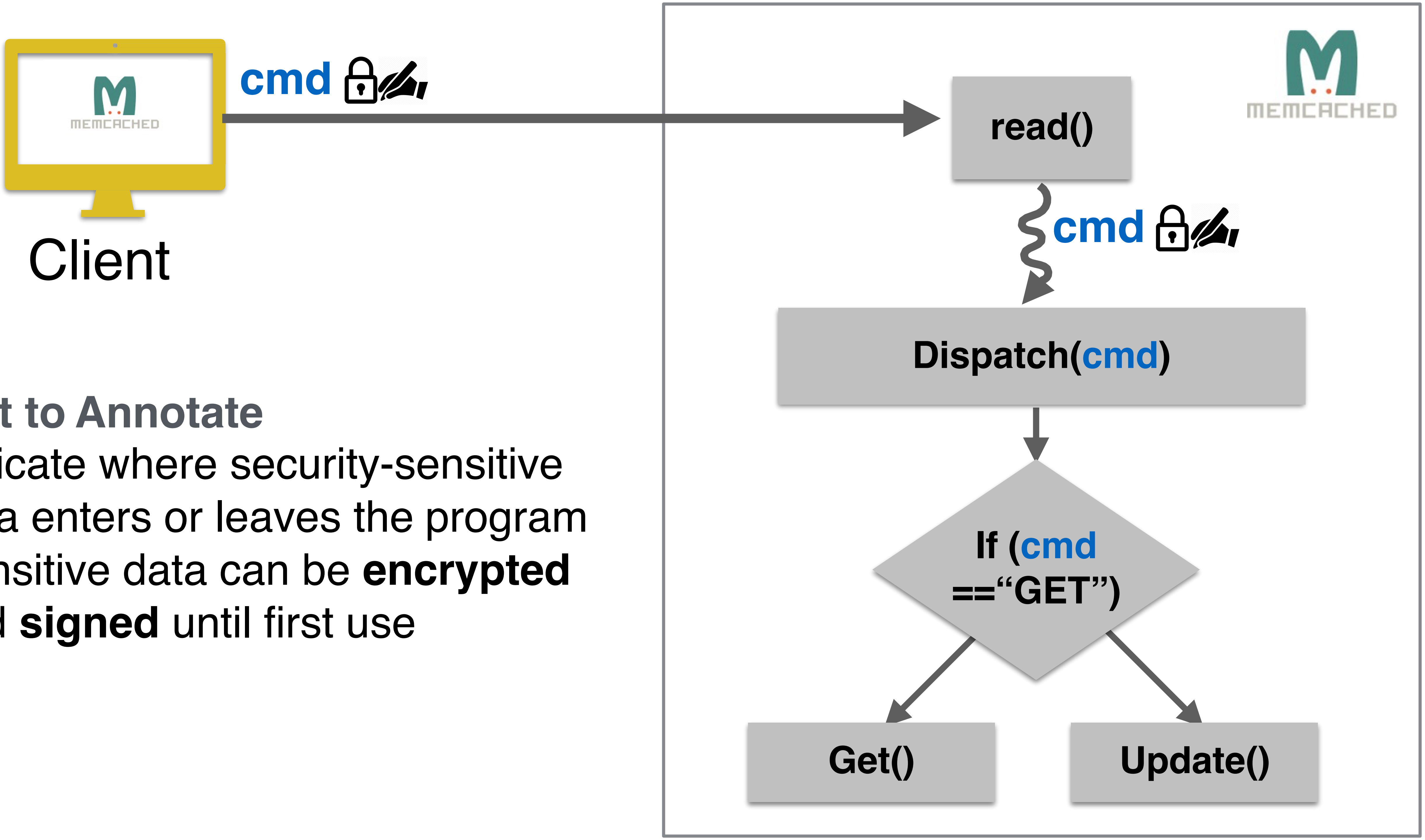
Interface Spec

1. Identify Security-Sensitive Code

Static Analysis conservatively identifies subset of code dependent on programmer annotated security-sensitive data



Annotation of Security-Sensitive Data



What to Annotate

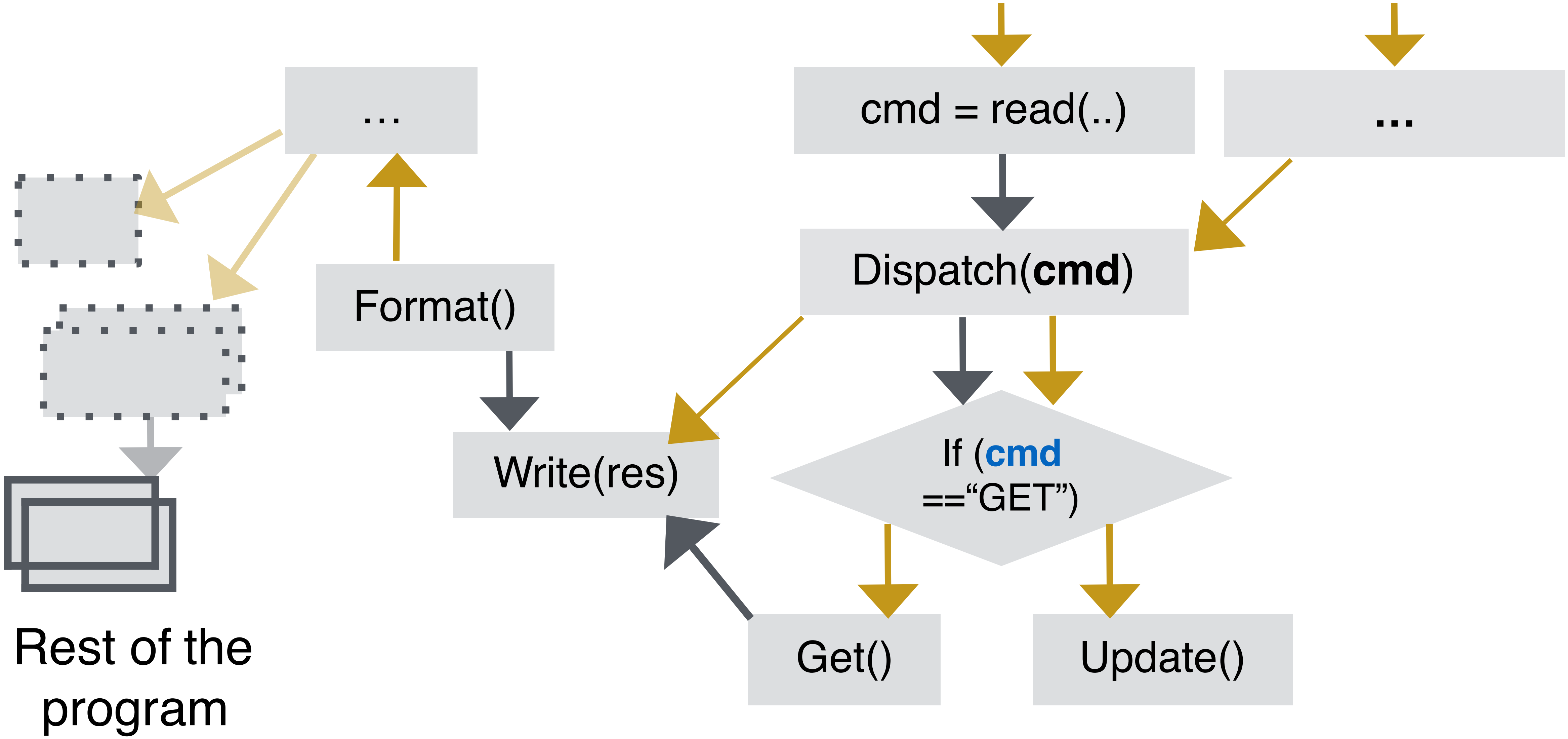
- Indicate where security-sensitive data enters or leaves the program
- Sensitive data can be **encrypted** and **signed** until first use

Static Analysis Goals

- Enforcing **Confidentiality**: Identify all functions that depend on sensitive data.
- Enforcing **Integrity**: Identify all functions on which the value of sensitive data depends
- Why Static Analysis?
  - Static Analysis is **conservative**, independent of the input to the program

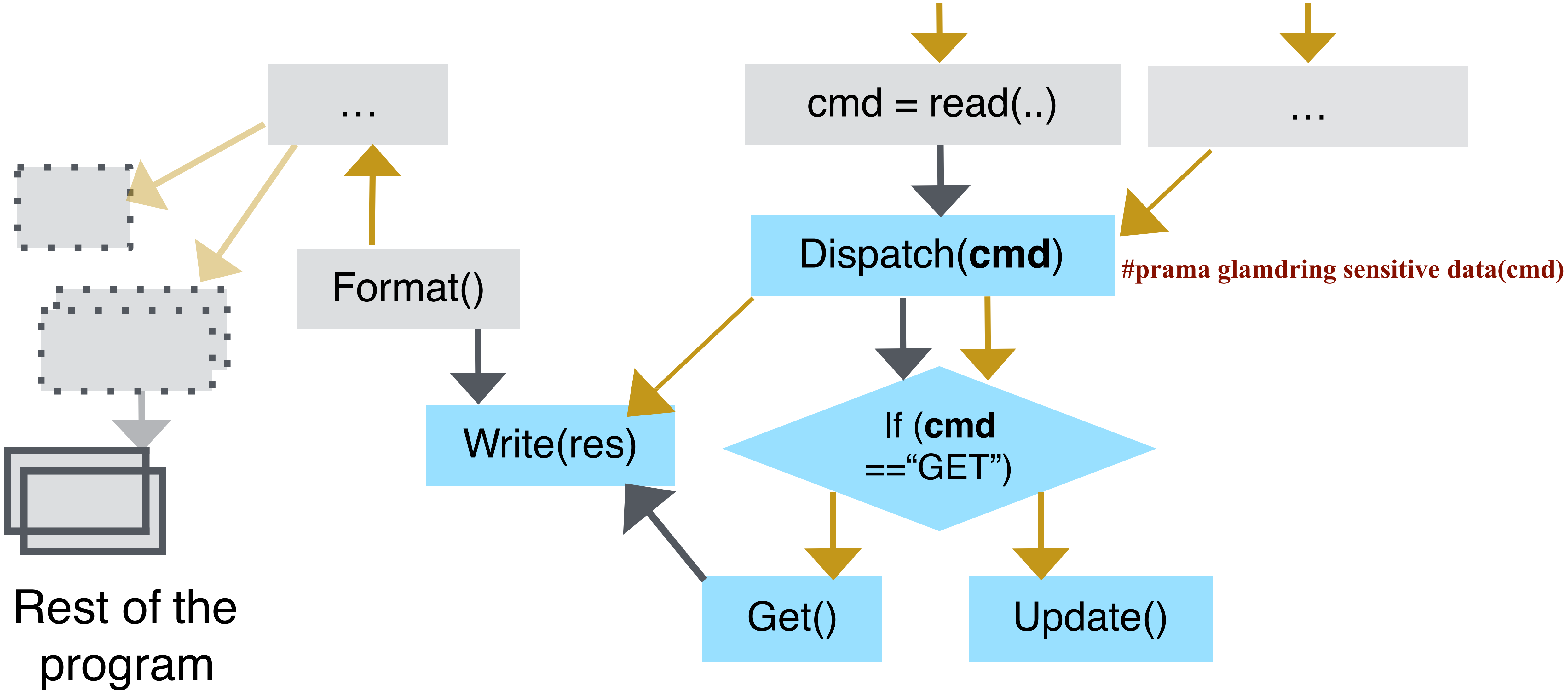
Big question on alias analysis:
Static pointer analysis for C program can be very imprecise
→ Be conservative → increase TCB size

Program Dependence Graph



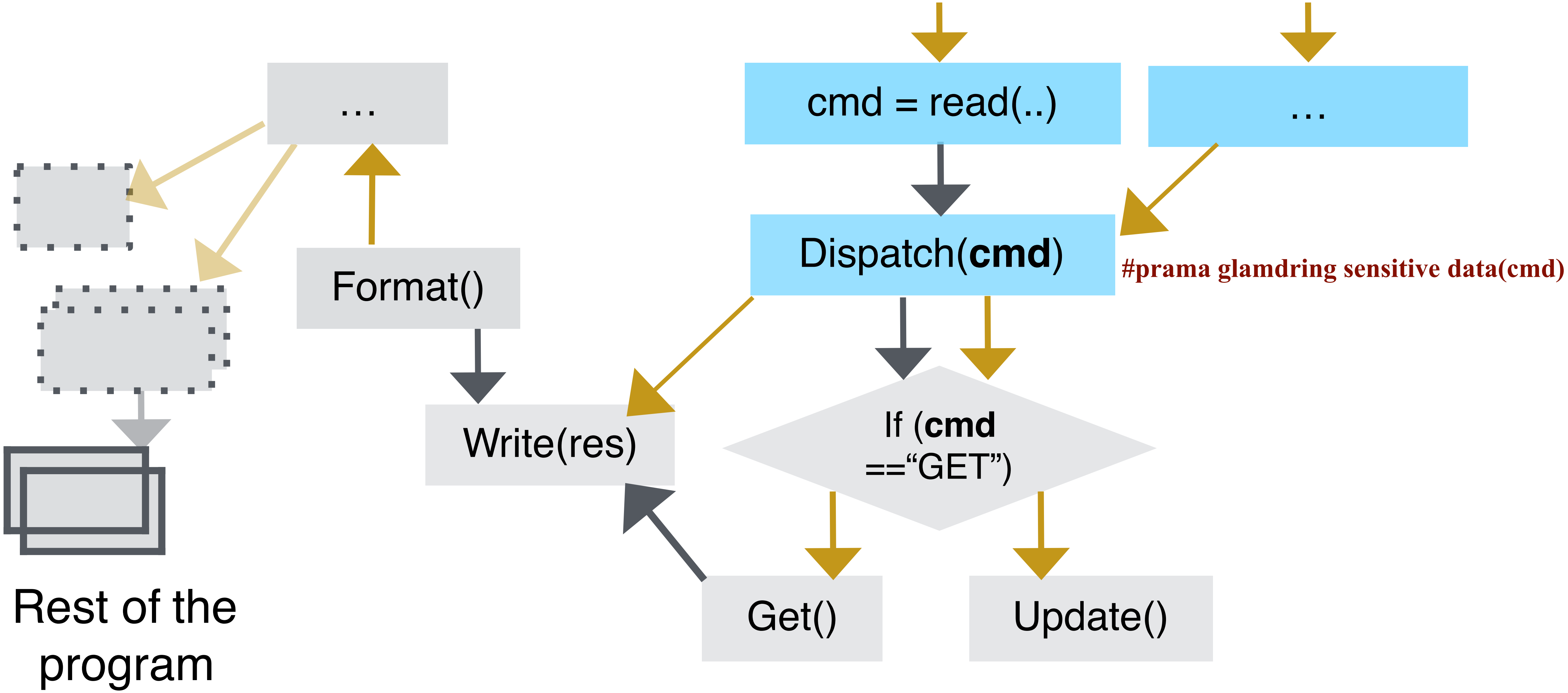
Forwards Dataflow Analysis

Confidentiality Using Graph Reachability identify all nodes with transitive control/data dependency on annotated node



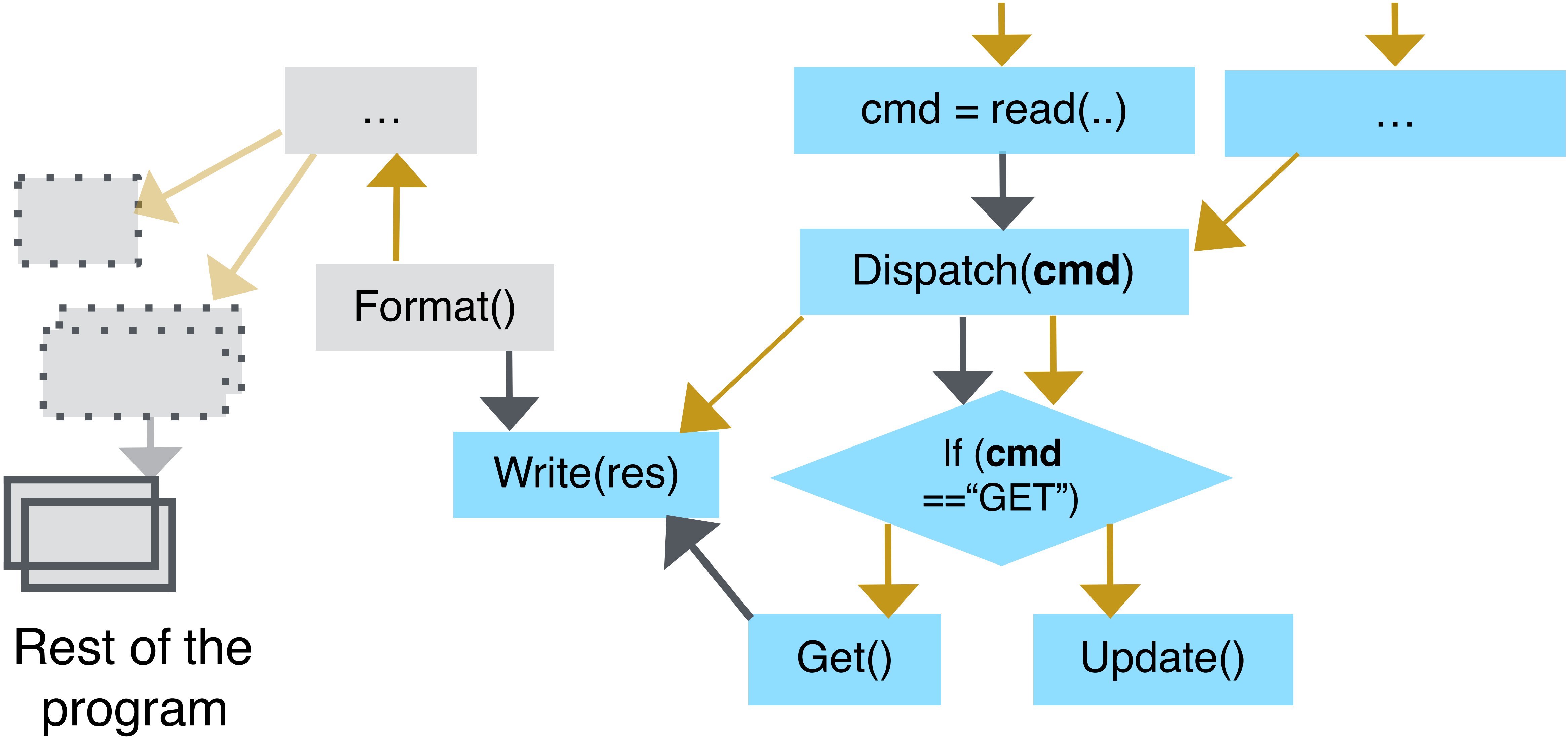
Backward Dataflow Analysis

Integrity Using Graph Reachability identify all nodes that are transitive control/data dependent on annotated node



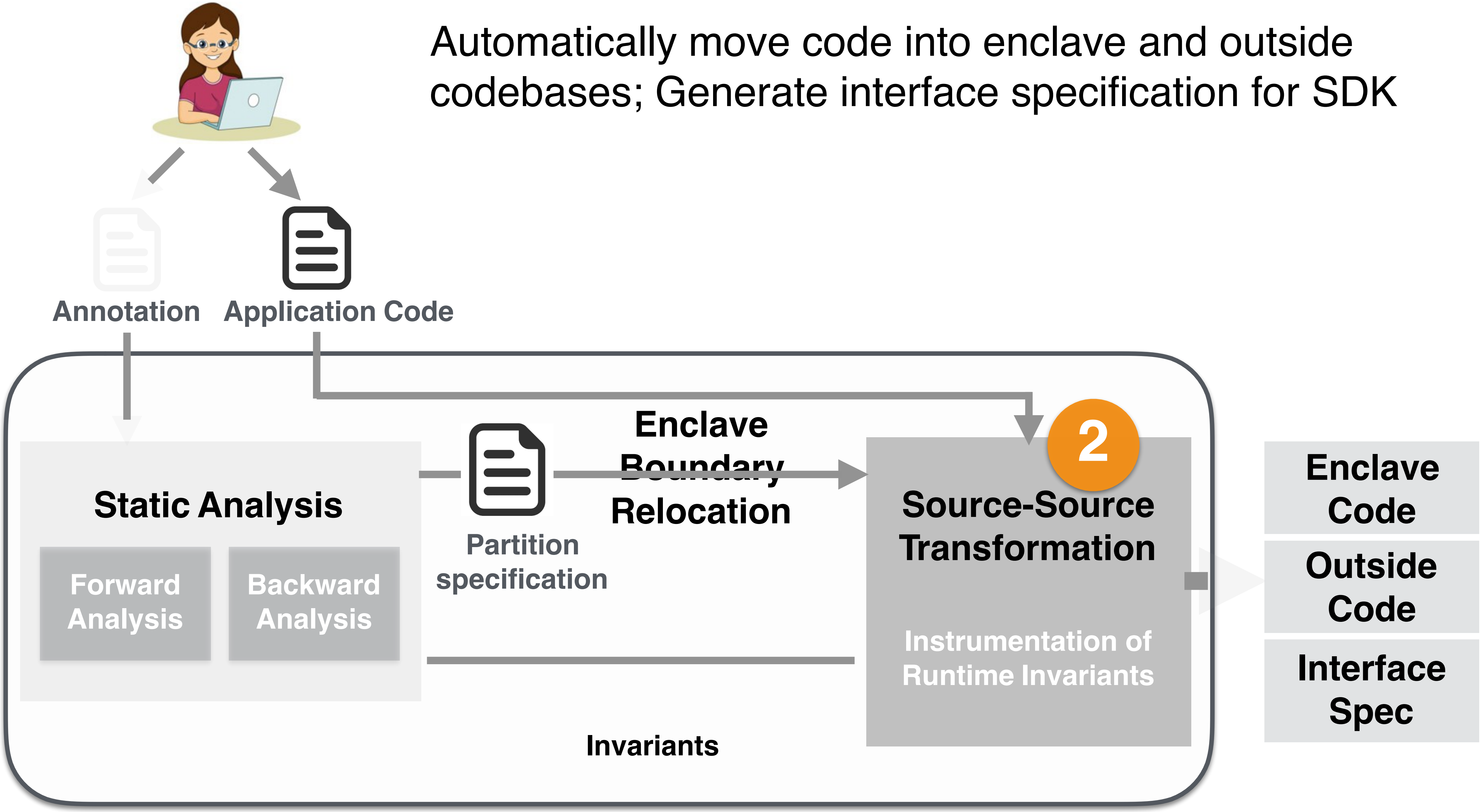
Security Sensitive Code

Union of nodes found with forwards and backwards analyses



2. Producing a Partitioned Application

Automatically move code into enclave and outside codebases; Generate interface specification for SDK



Source-Source Transformation

Partition Spec

- Enclave Functions:

  `Dispatch,`
  `Get,`
  `Update`

- Enclave Allocations:

  `malloc@241`

- Enclave Allocated Globals

  `hash_items`

Sound intuitive and easy at high level.
Many corner cases about data
accessed/modified in functions in two worlds.

Outside

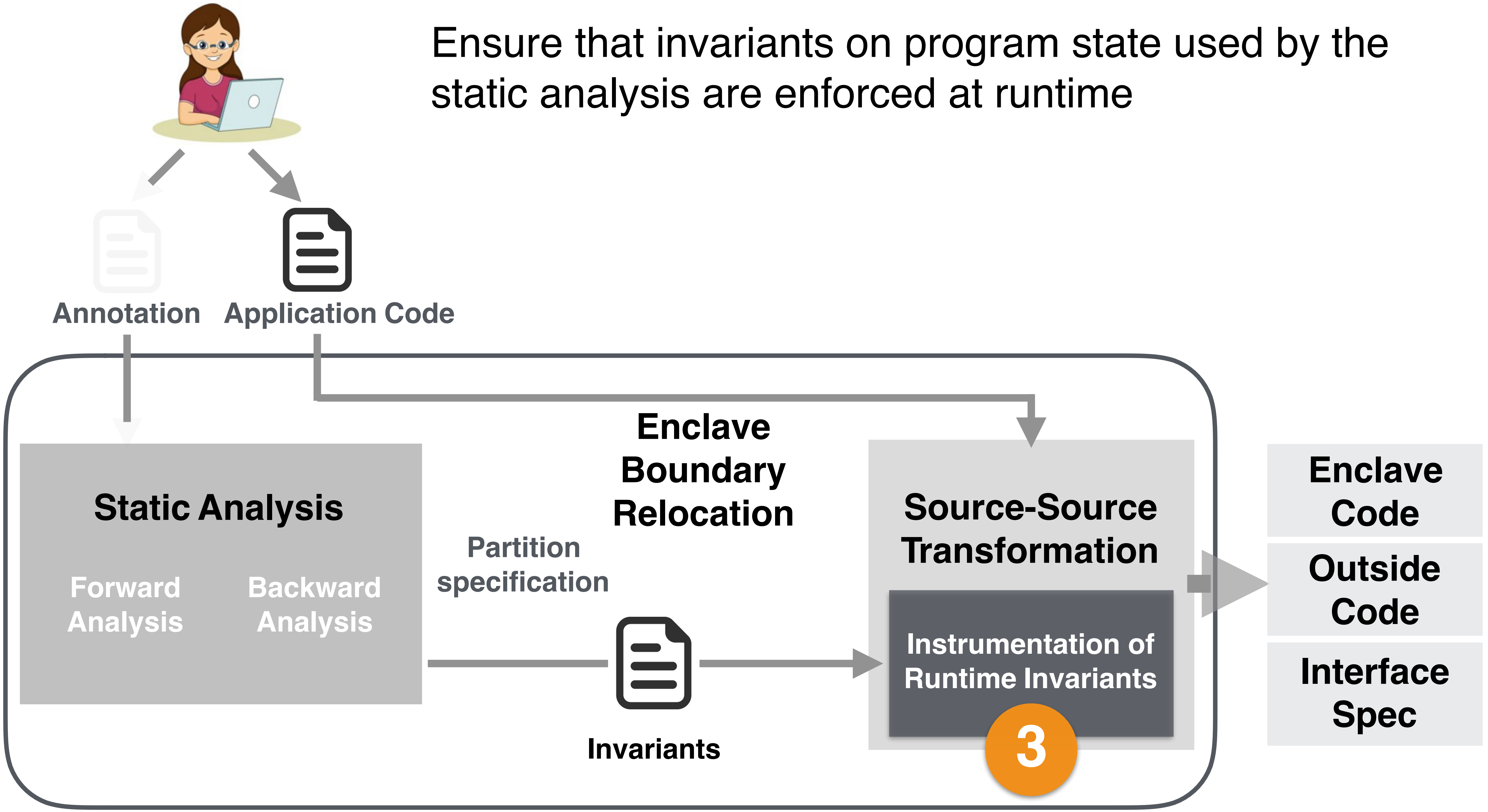
```
void Read(…) {
   ecall__Dispatch();
}
```

Enclave

```
void ecall__Dispatch(…){
…
}

void Get(…) {
…
}

void Put(…) {
…
}
```

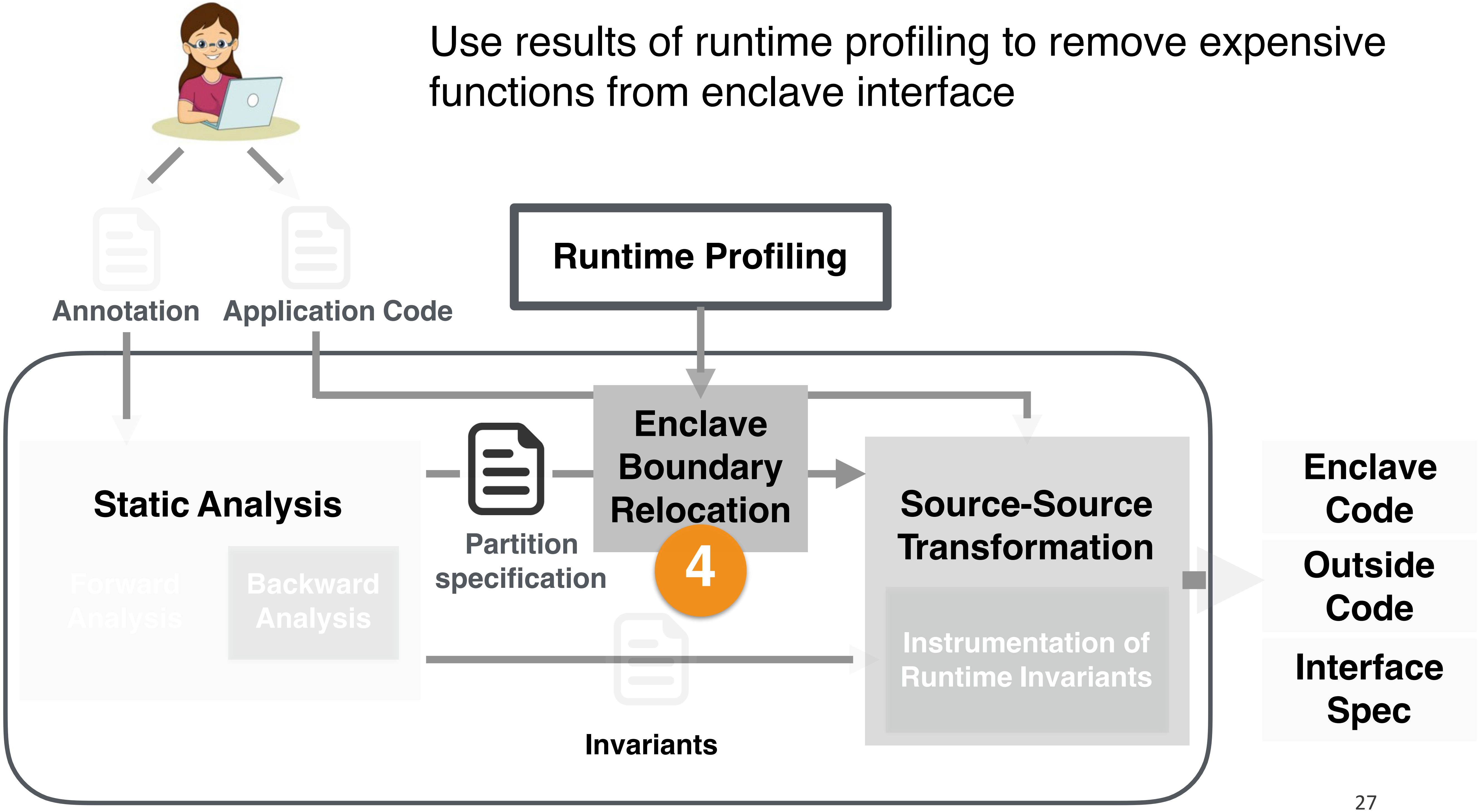
3. Upholding Static Analysis Invariants

Ensure that invariants on program state used by the static analysis are enforced at runtime



4. Improving Performance After Partitioning

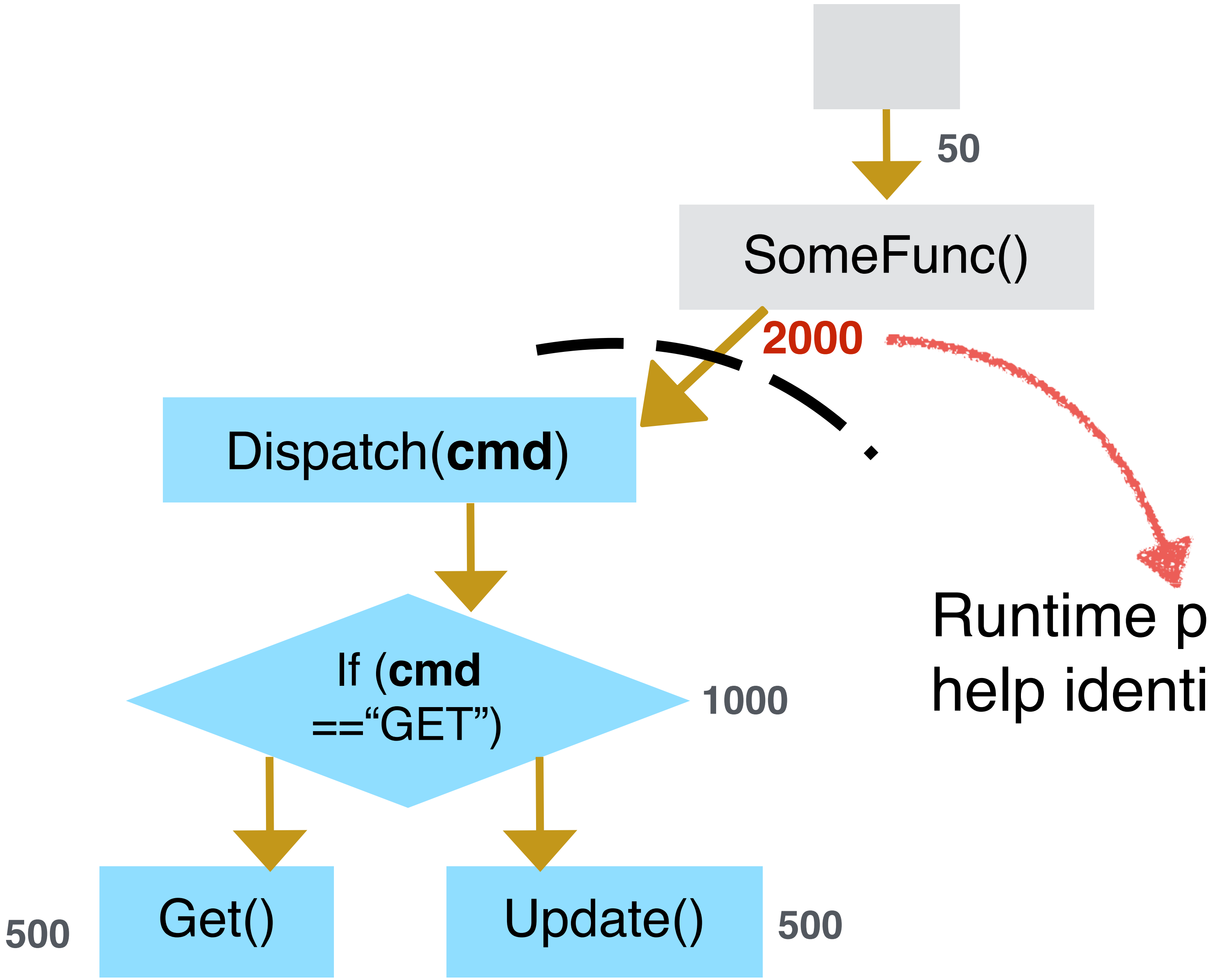
Use results of runtime profiling to remove expensive functions from enclave interface



Performance of Partitioned Applications

Expensive Interface Functions

Some of the interface functions may be ‘hotspots’ called too frequently

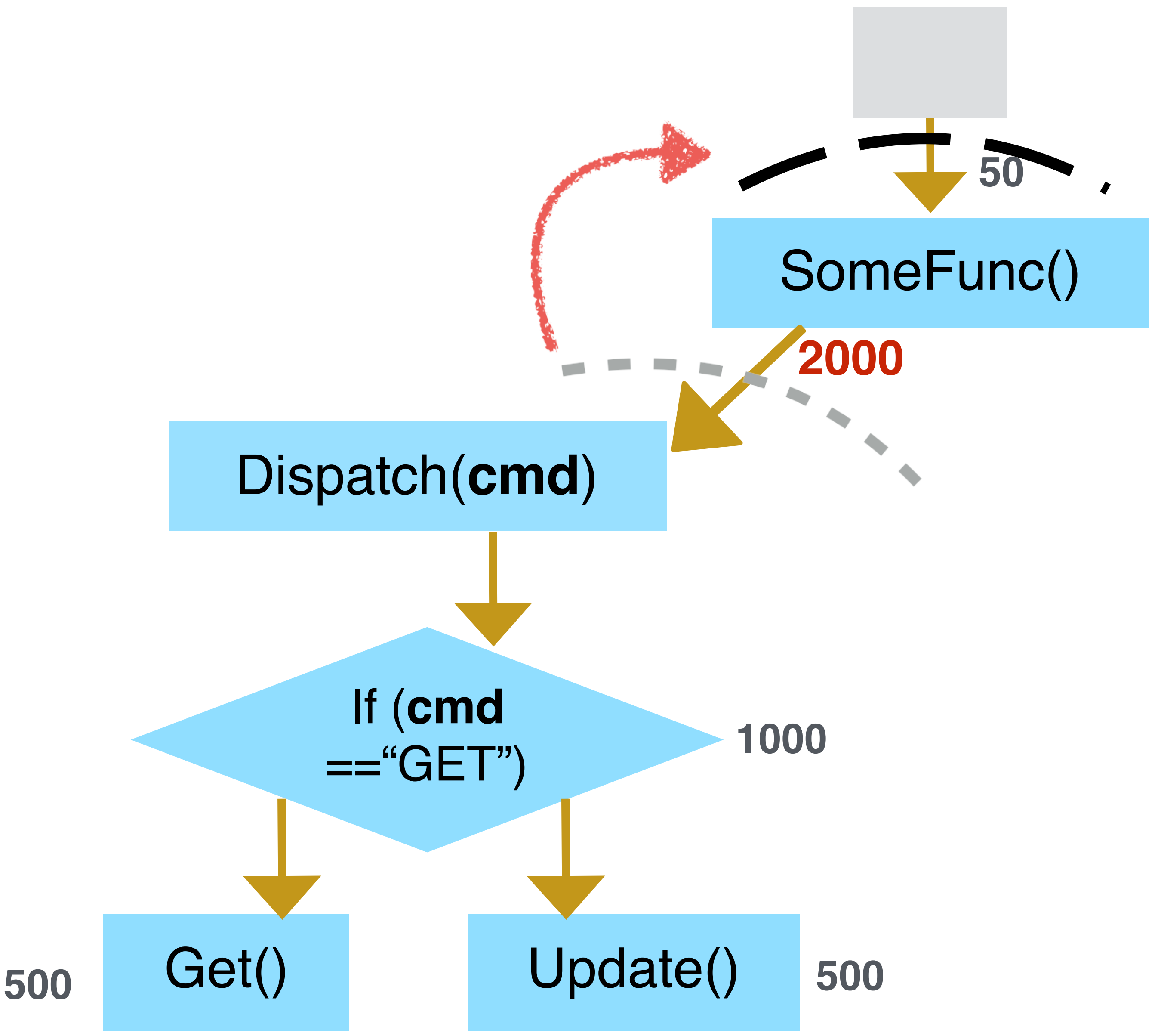


Runtime profiling can help identify hotspots

Enclave Boundary Relocation

Adding Functions to Enclave

Move additional functions into enclave to create a new interface that avoid ‘hotspots’



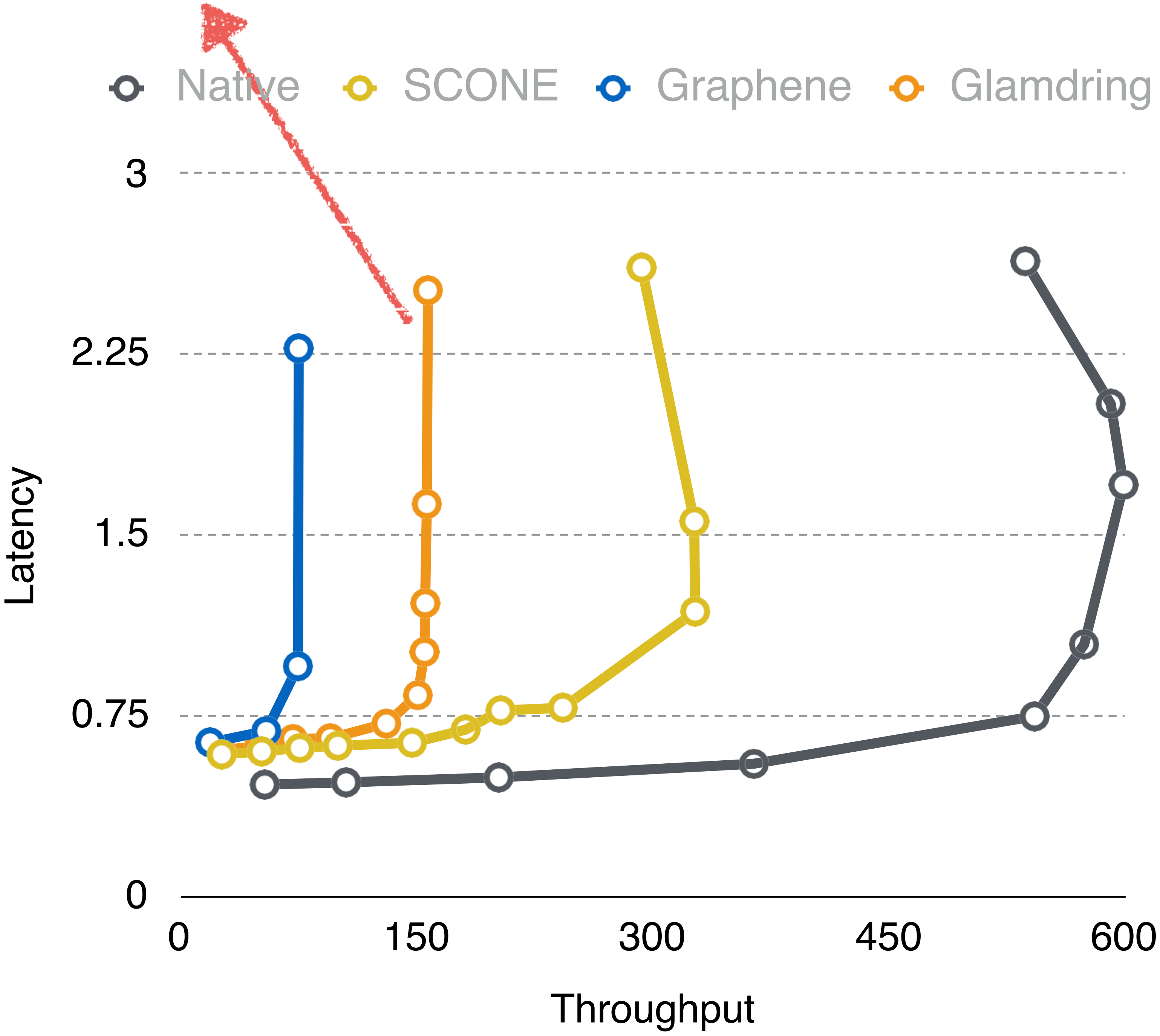
Security Evaluation – TCB size

Applications	Code Size (kLoC)	TCB size
Memcached	31	12 (**40%**)
DigitalBitbox	23	8 (**38%**)
LibreSSL	176	38 (**22%**)

TCB is less than 40% of the application size

Throughput vs Latency

Enclave transitions dominate the cost of request handling; batching requests into multi-get gets 210k req/sec



Discussion Questions on Security

- Is the possibility of side channel attacks increased with this method?

- How much does Glamdring truly reduce your TCB? Are you not just adding Glamdring's source as a TCB itself?

- The paper assumes that reducing the size of the TCB will lead to increased security, because otherwise small amounts of malicious code could enter the enclave undetected. Isn't the inverse true? With this new methodology, small amounts of secure code could be left out of the enclave by accident.

Discussion Questions on Performance/Practicality

- What are the costs to other applications interacting with glamdring? How expensive is the requirement that they encrypt/decrypt all calls.

- If the application changes anything, does the developer need to re-annotate from scratch?

- Is it fair to burden developers with the requirement to sift through the code and mark things that are security-sensitive? Could this requirement introduce subtle security bugs by omission?

- Are there any particular programs which would greatly benefit from such a partitioning scheme?