ACM SIGOPS 2026 Annual Technical Conference

ESROS2: Application Sandboxes for Secure ROS2

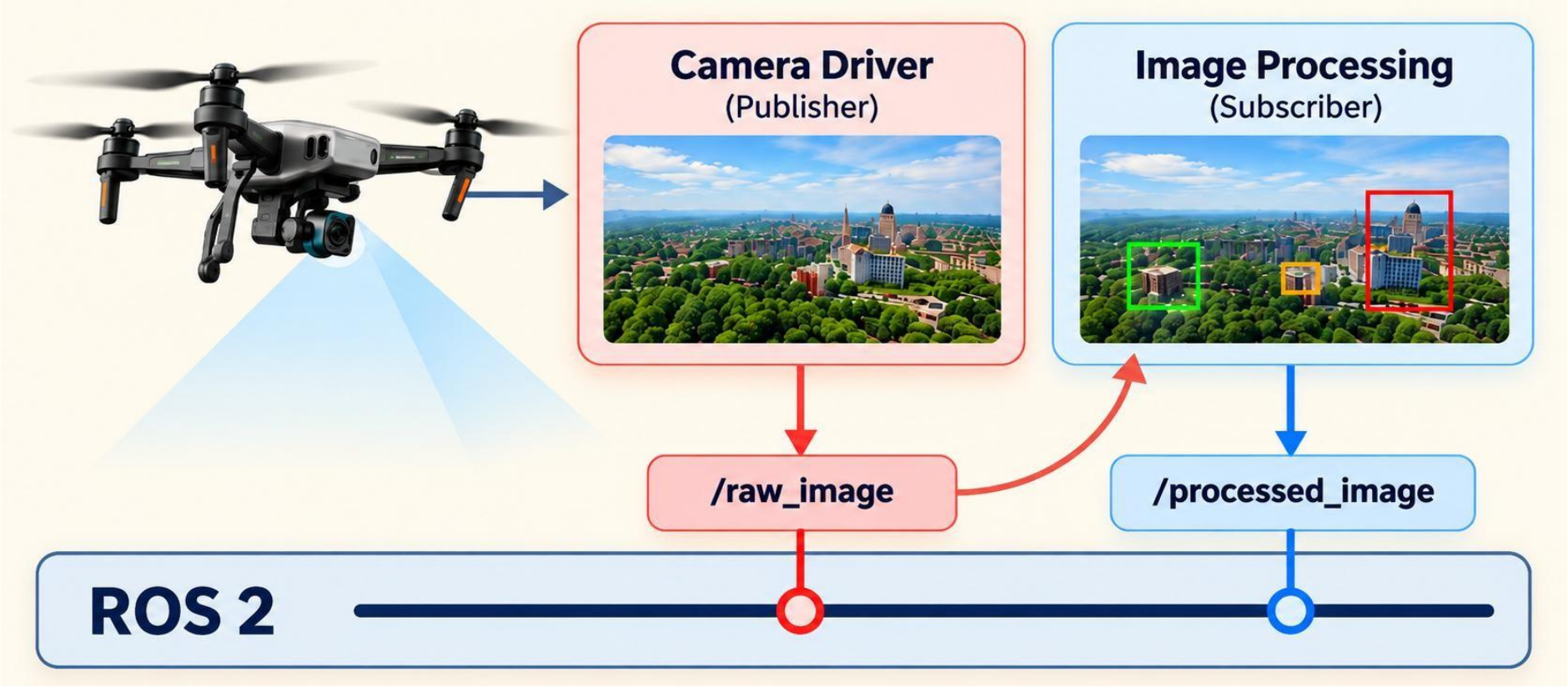
Dev Tejas Gandhi, Vinod Ganapathy



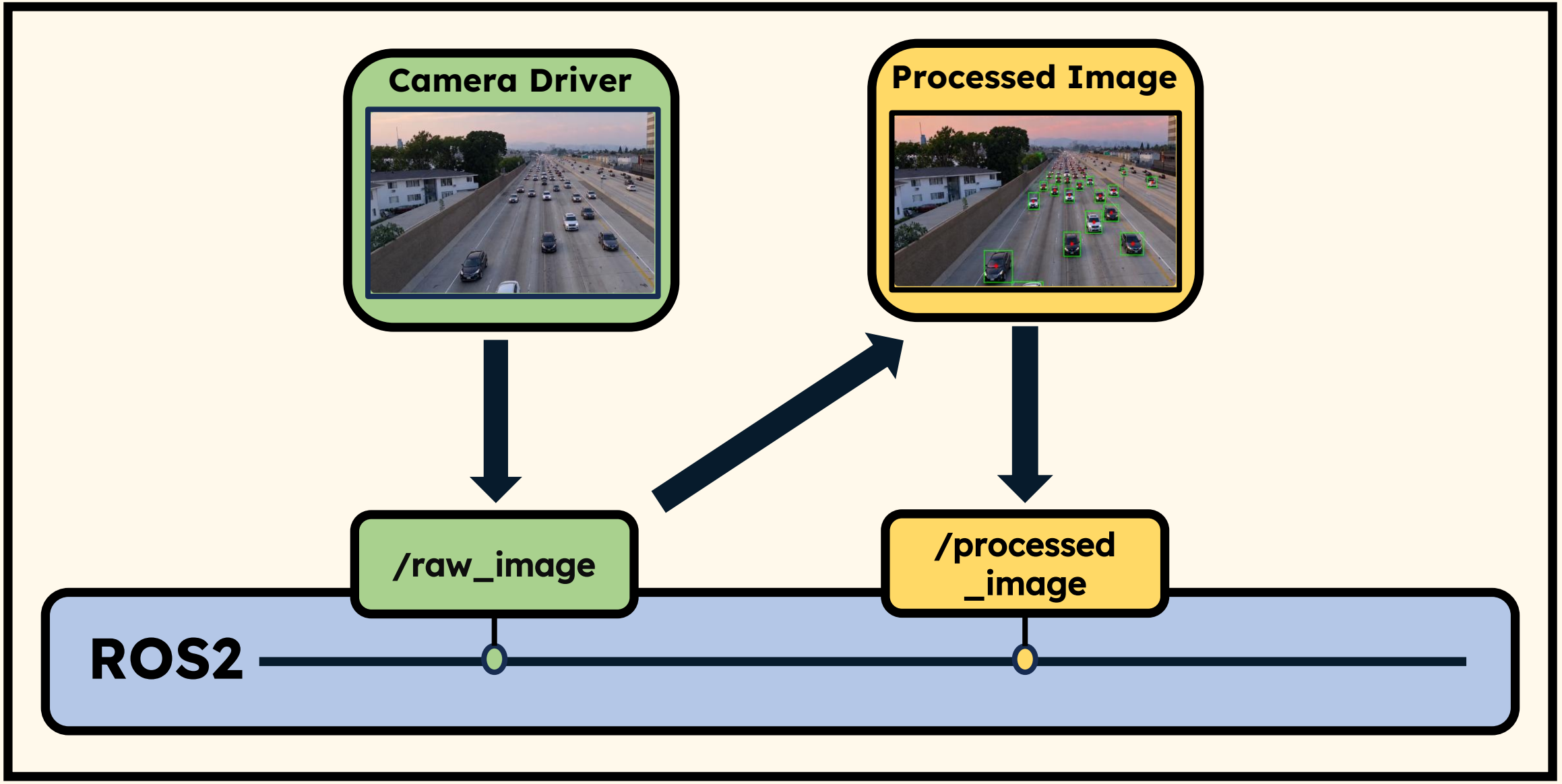
Computer Systems Security Lab,
Indian Institute of Science

ROS2 Introduction

Publish–subscribe based middleware for robotics.

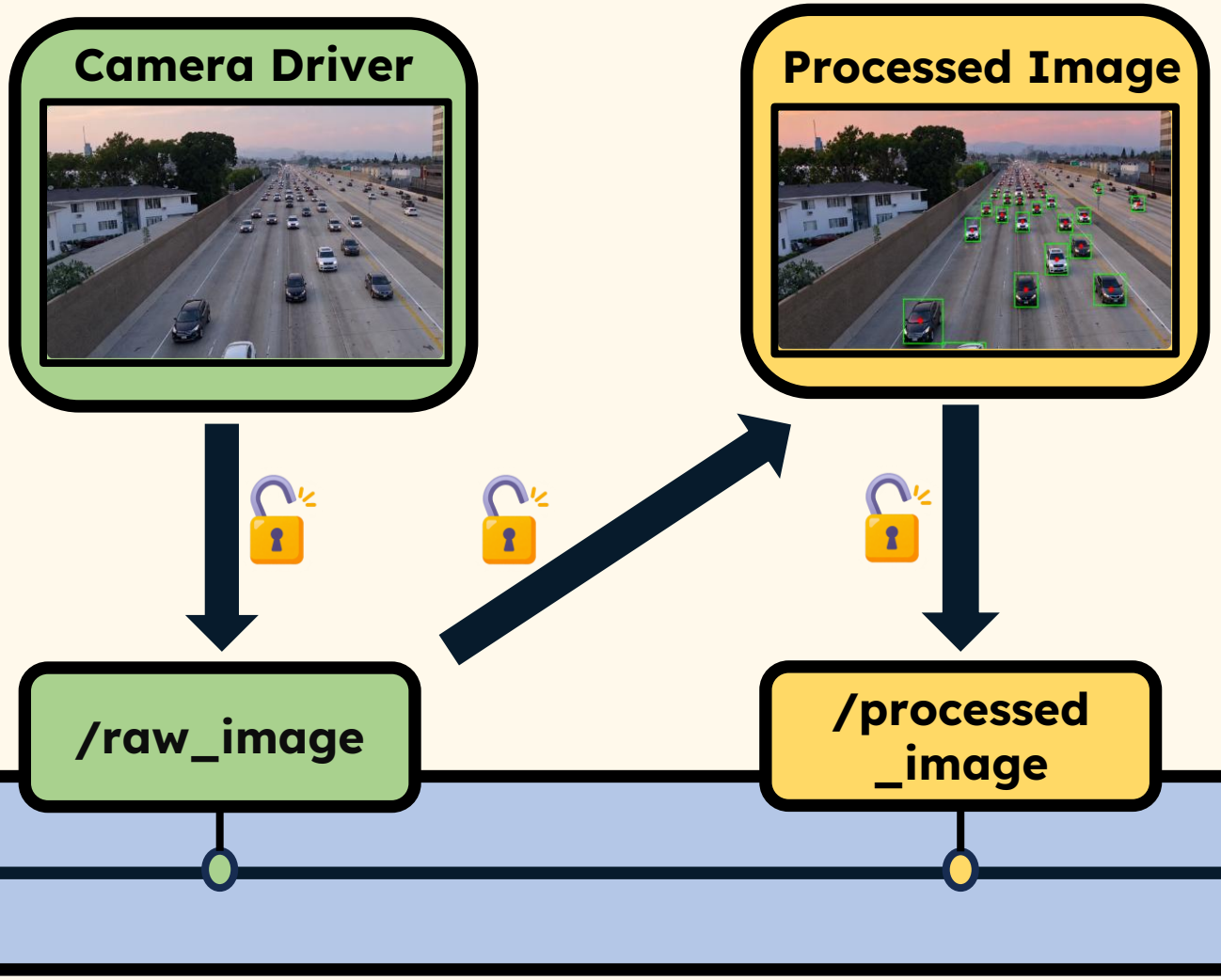


Security Challenges in ROS 2



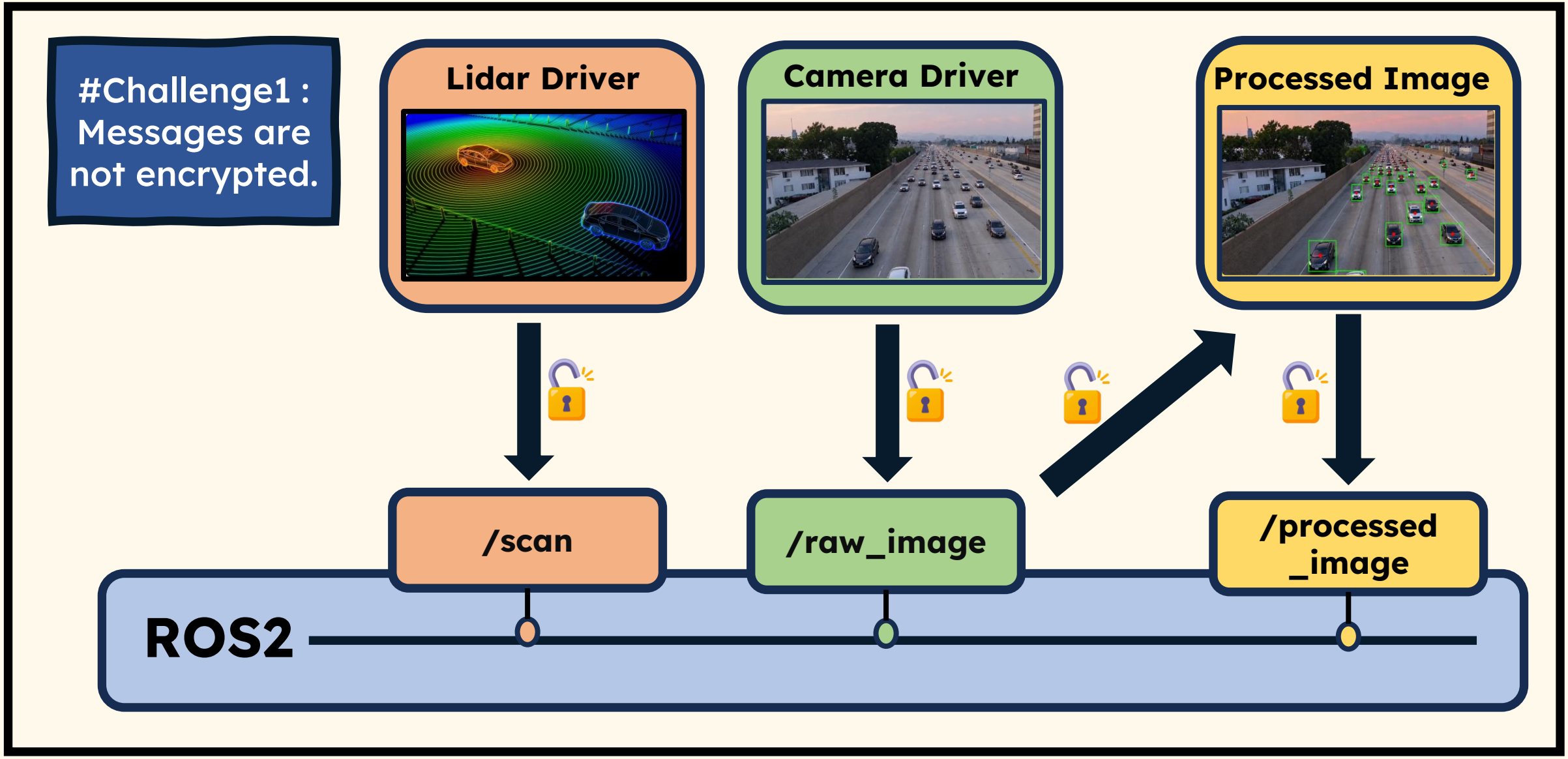
Security Challenges in ROS 2

#Challenge1 : Messages are not encrypted.



Security Challenges in ROS 2

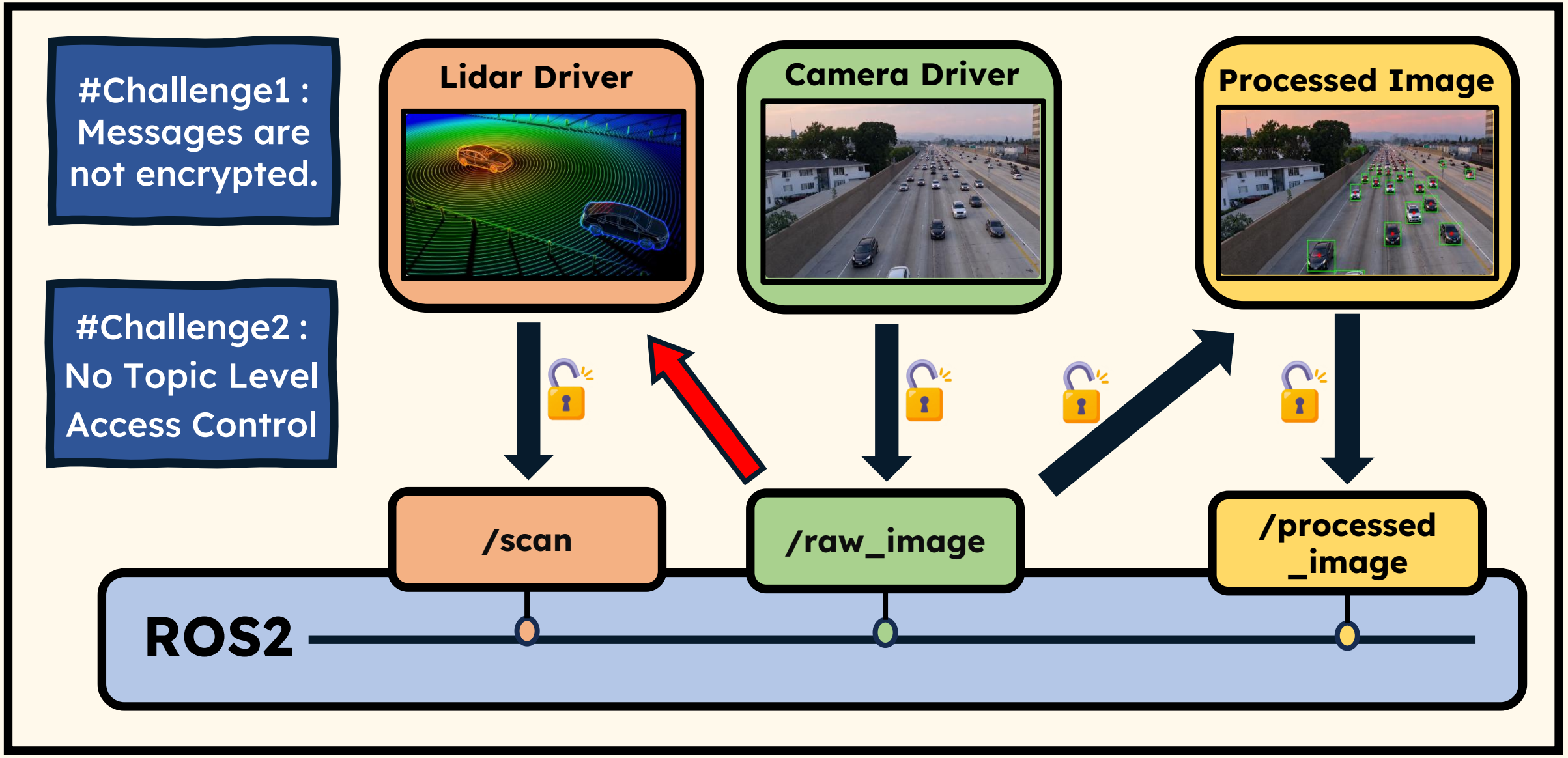
#Challenge1 : Messages are not encrypted.



Security Challenges in ROS 2

#Challenge1 : Messages are not encrypted.

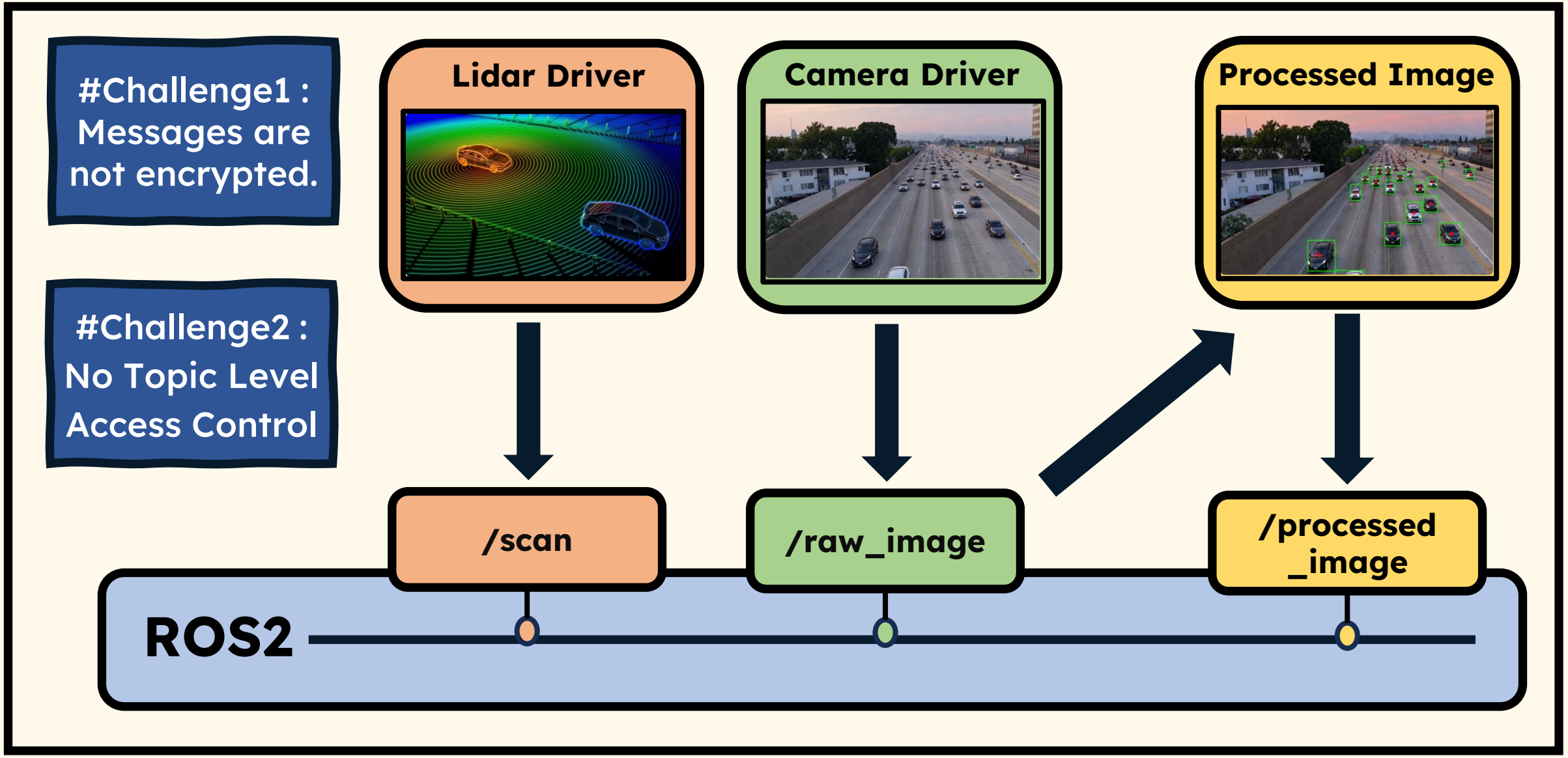
#Challenge2 : No Topic Level Access Control



Existing Solution: Secure ROS2 (SROS2)

#Challenge1 : Messages are not encrypted.

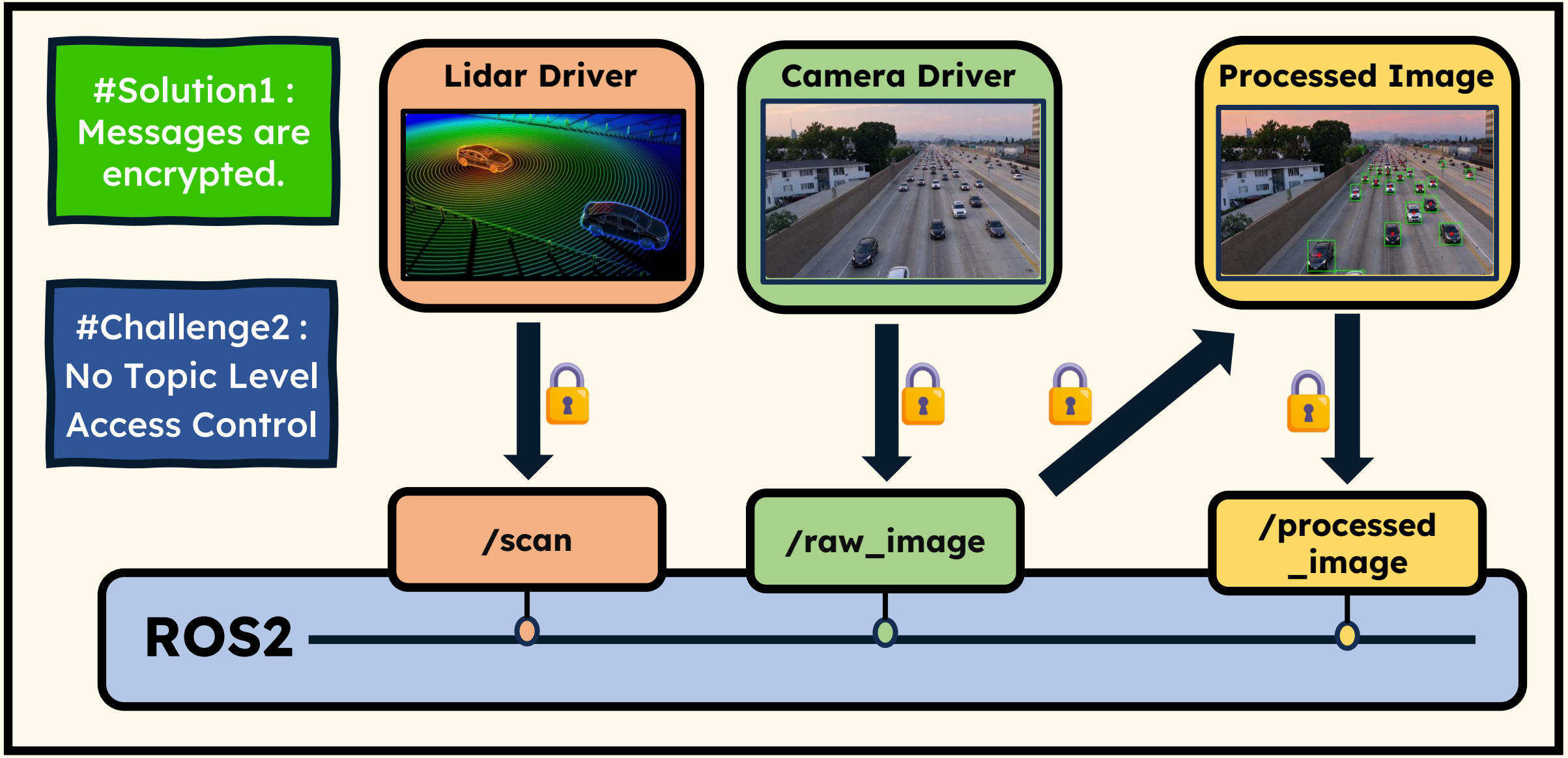
#Challenge2 : No Topic Level Access Control



Existing Solution: Secure ROS2 (SROS2)

#Solution1 : Messages are encrypted.

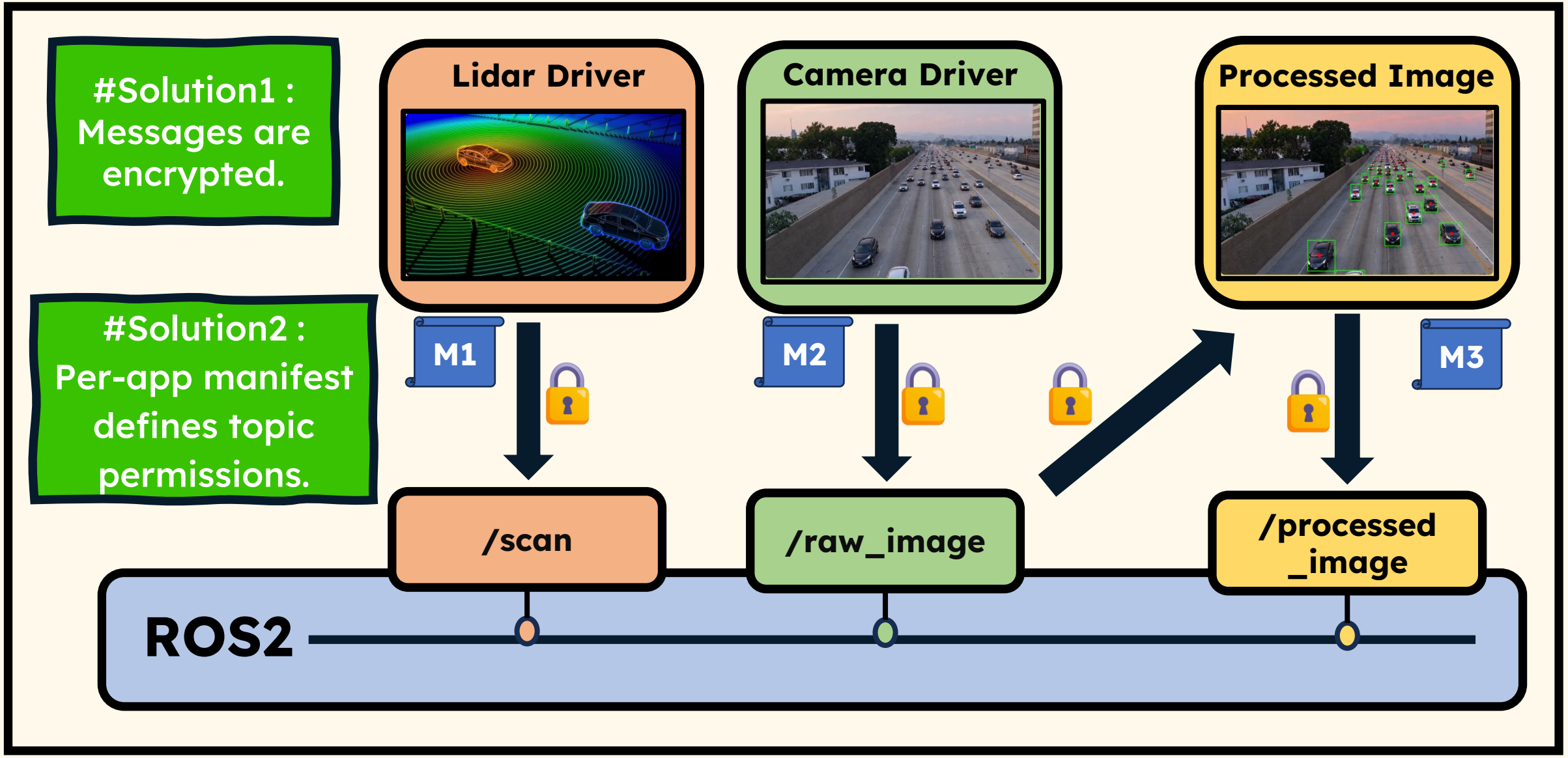
#Challenge2 : No Topic Level Access Control



Existing Solution: Secure ROS2 (SROS2)

#Solution1 : Messages are encrypted.

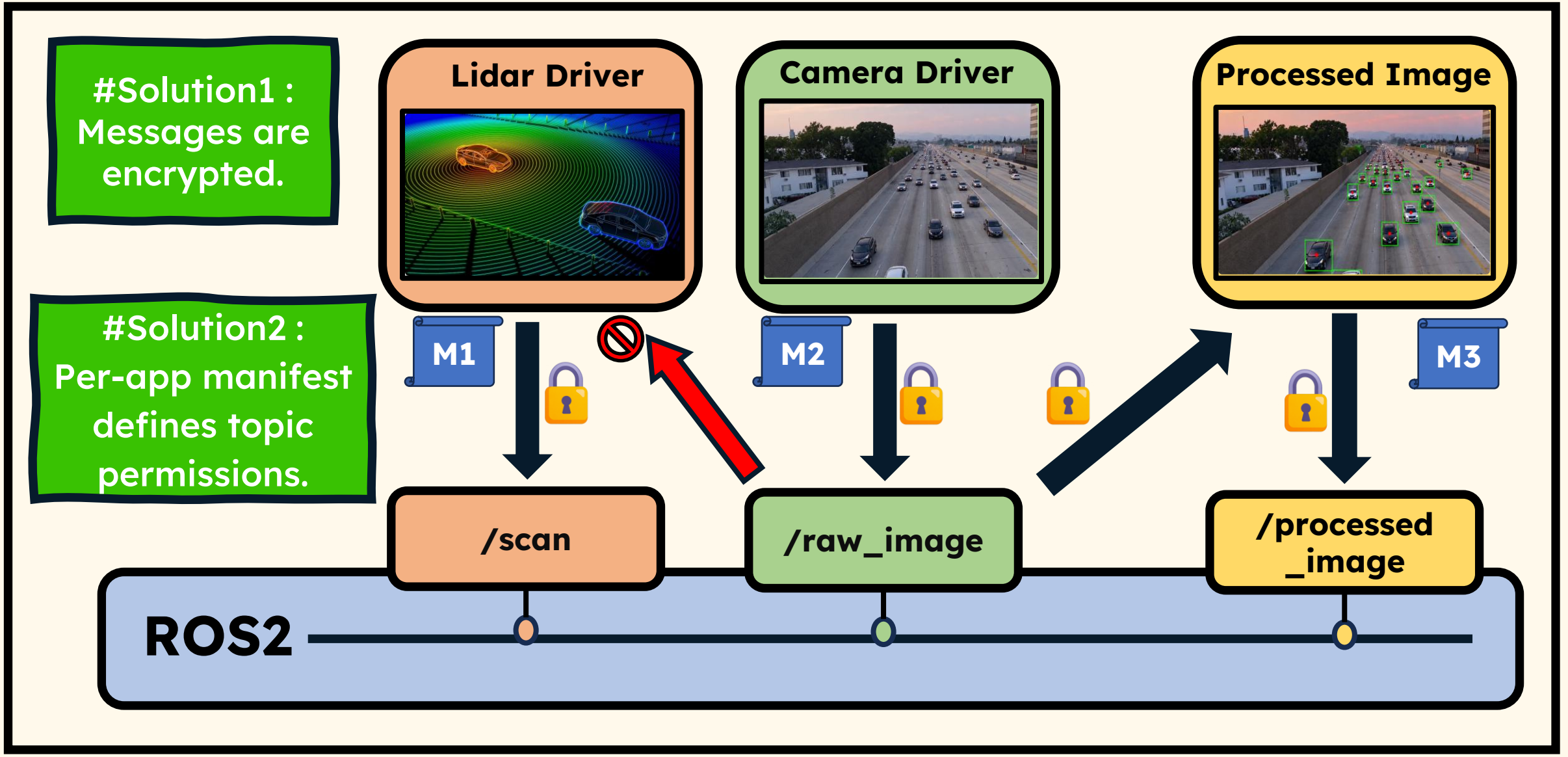
#Solution2 : Per-app manifest defines topic permissions.



Existing Solution: Secure ROS2 (SROS2)

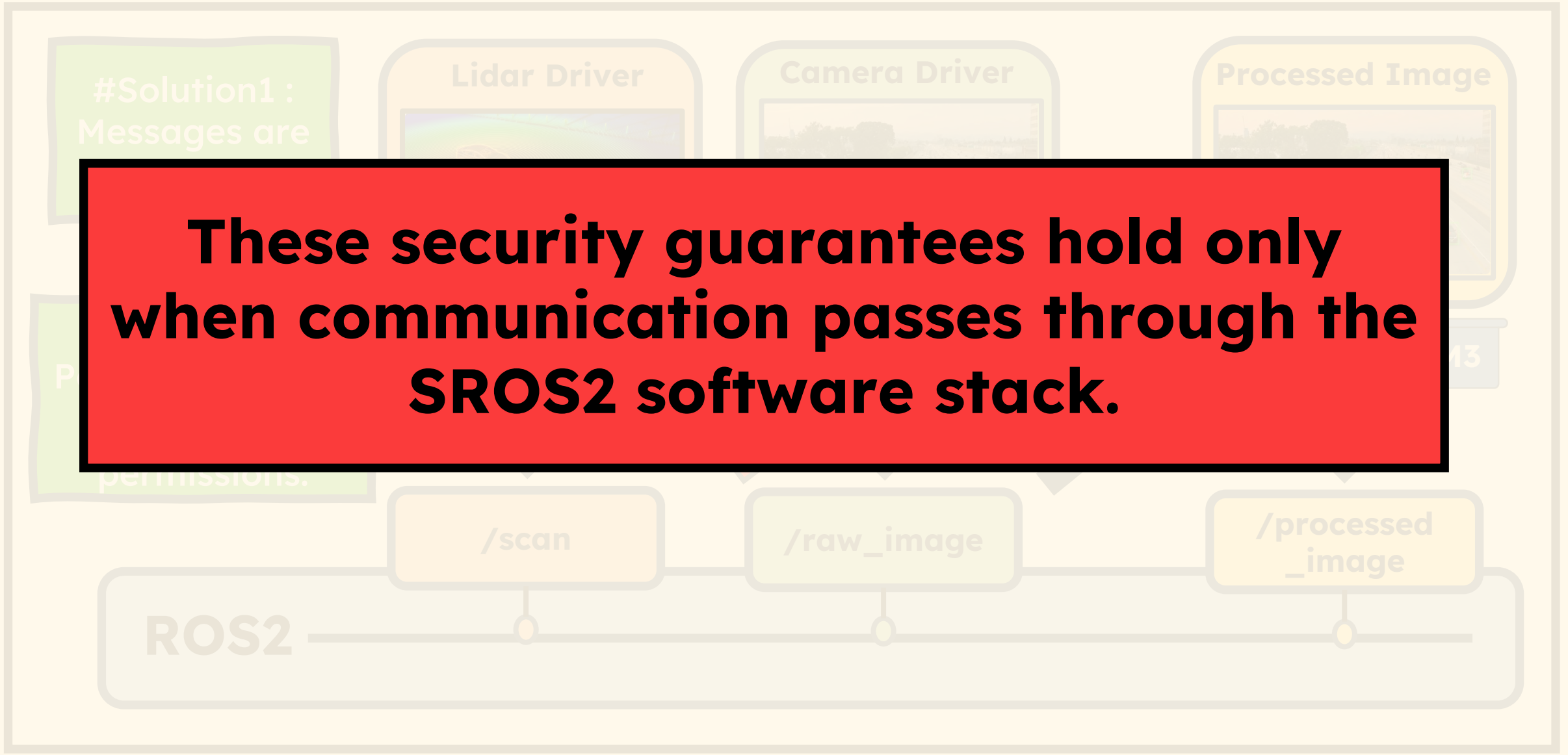
#Solution1 : Messages are encrypted.

#Solution2 : Per-app manifest defines topic permissions.



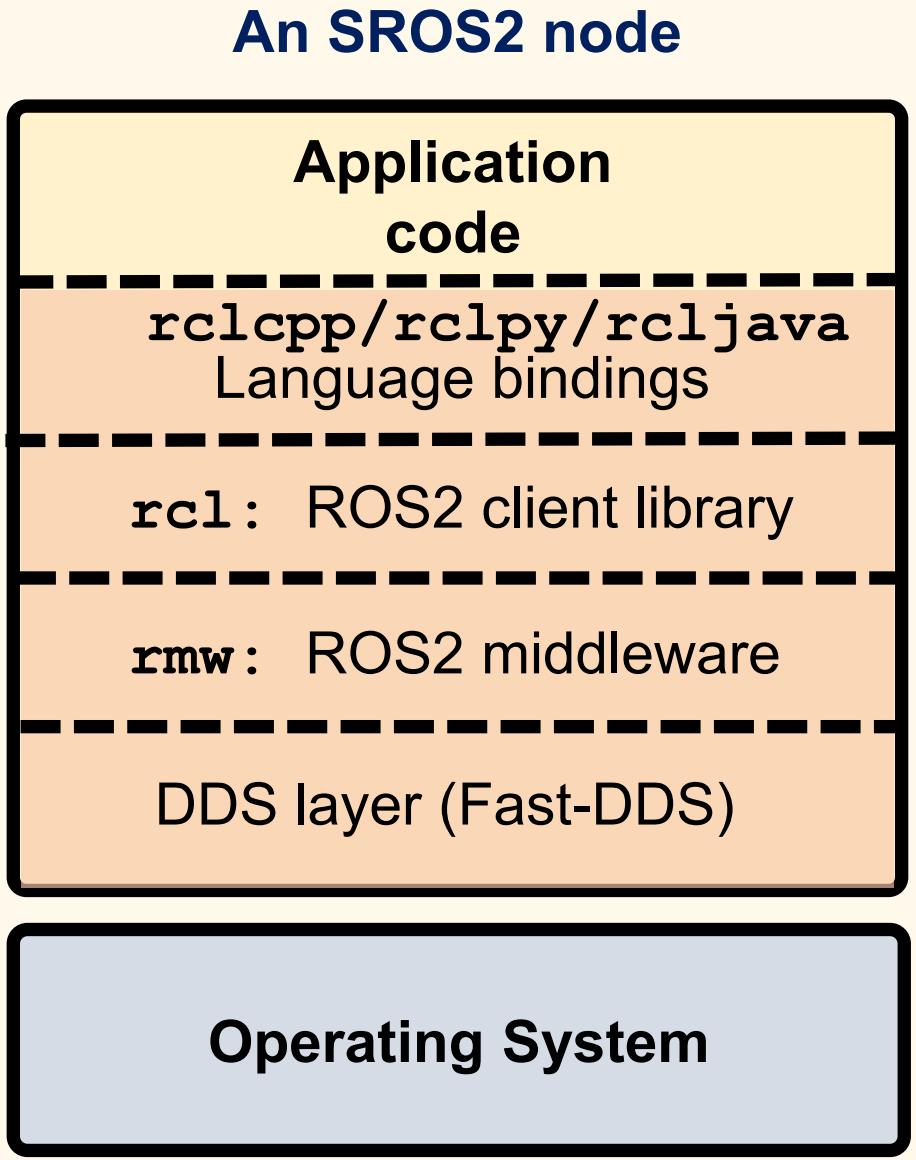
Bypassing the Secure SROS2 API

Bypassing the Secure ROS2 API

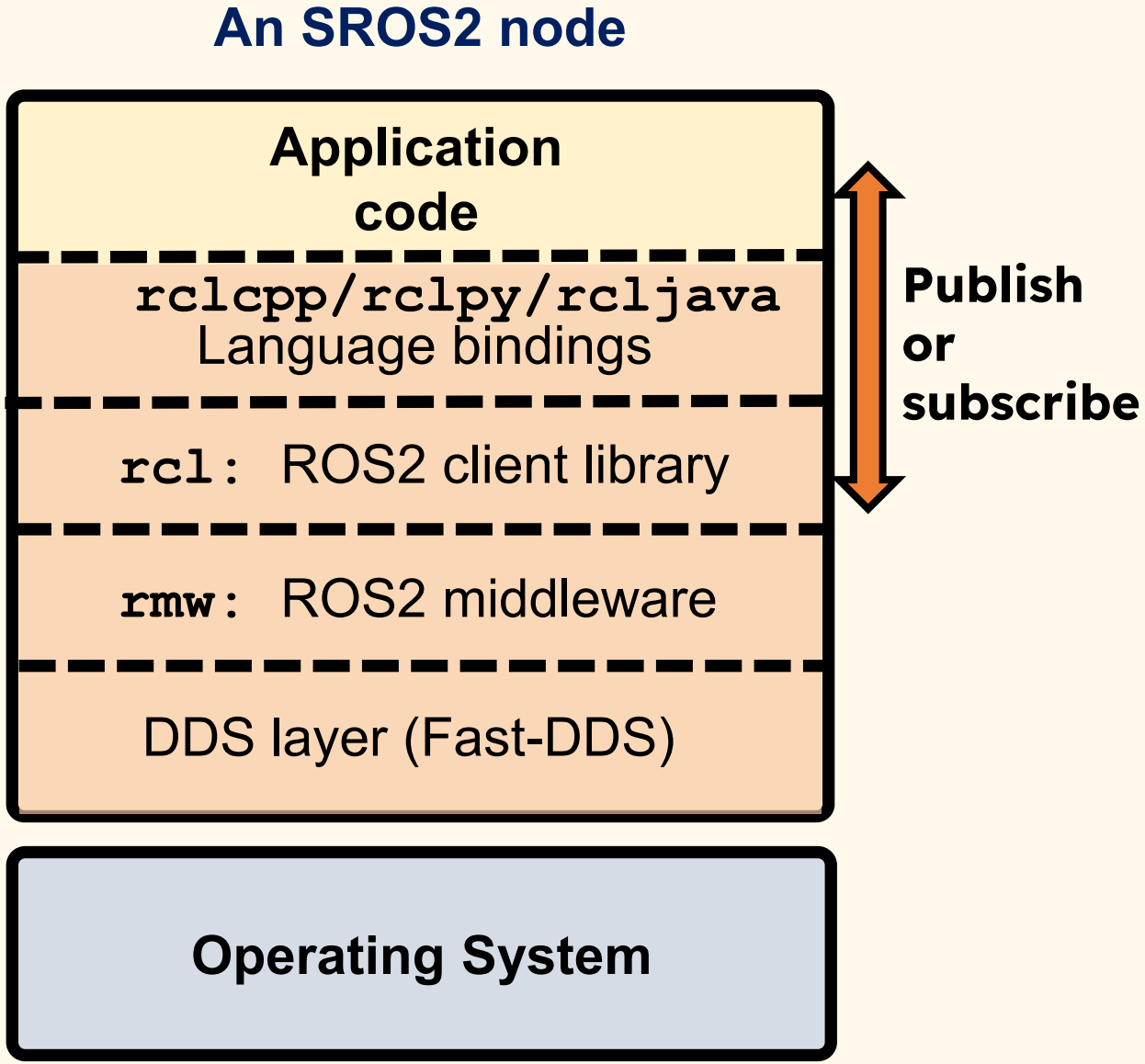


These security guarantees hold only when communication passes through the SROS2 software stack.

Bypassing the Secure ROS2 API

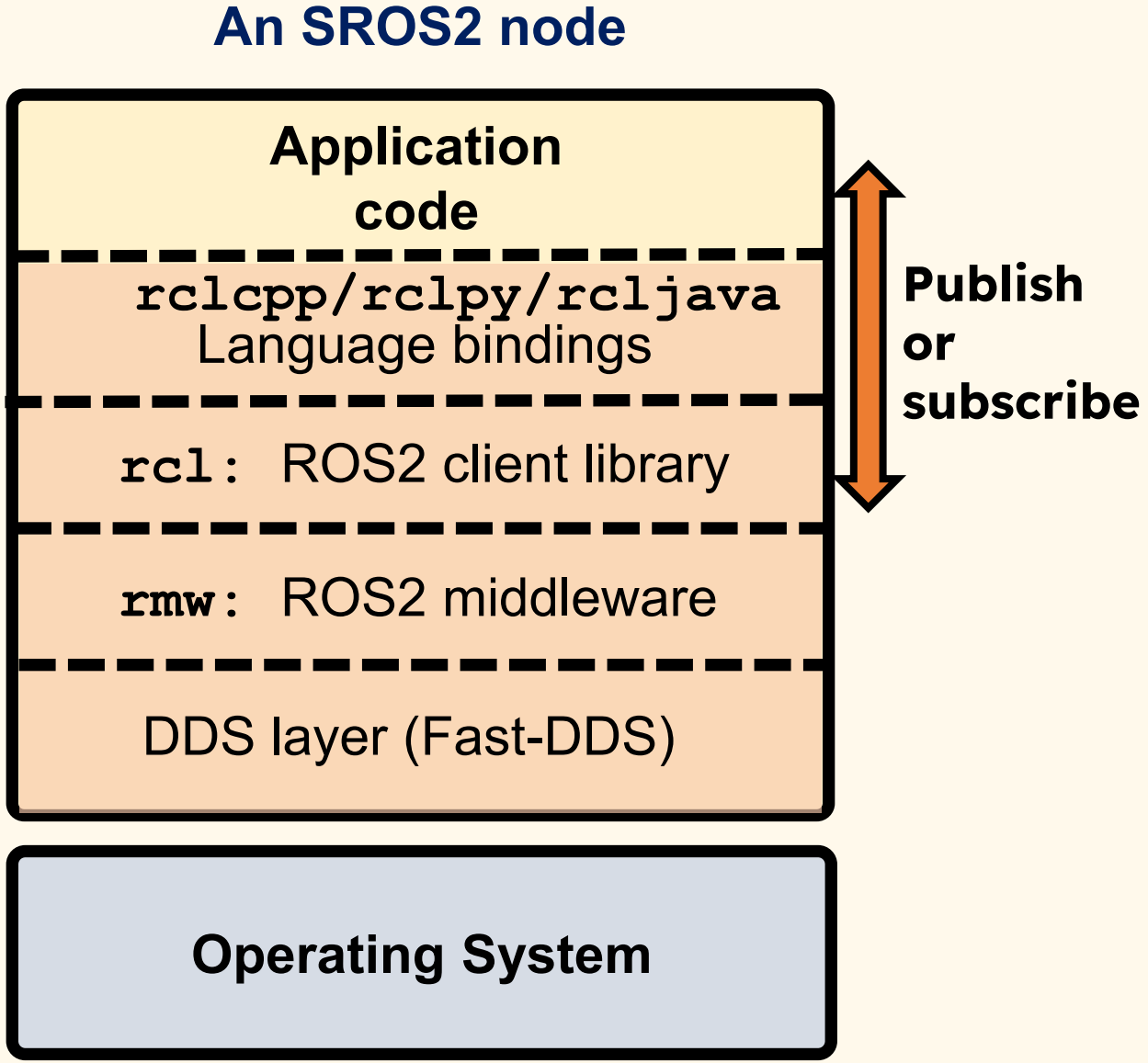


Bypassing the Secure ROS2 API



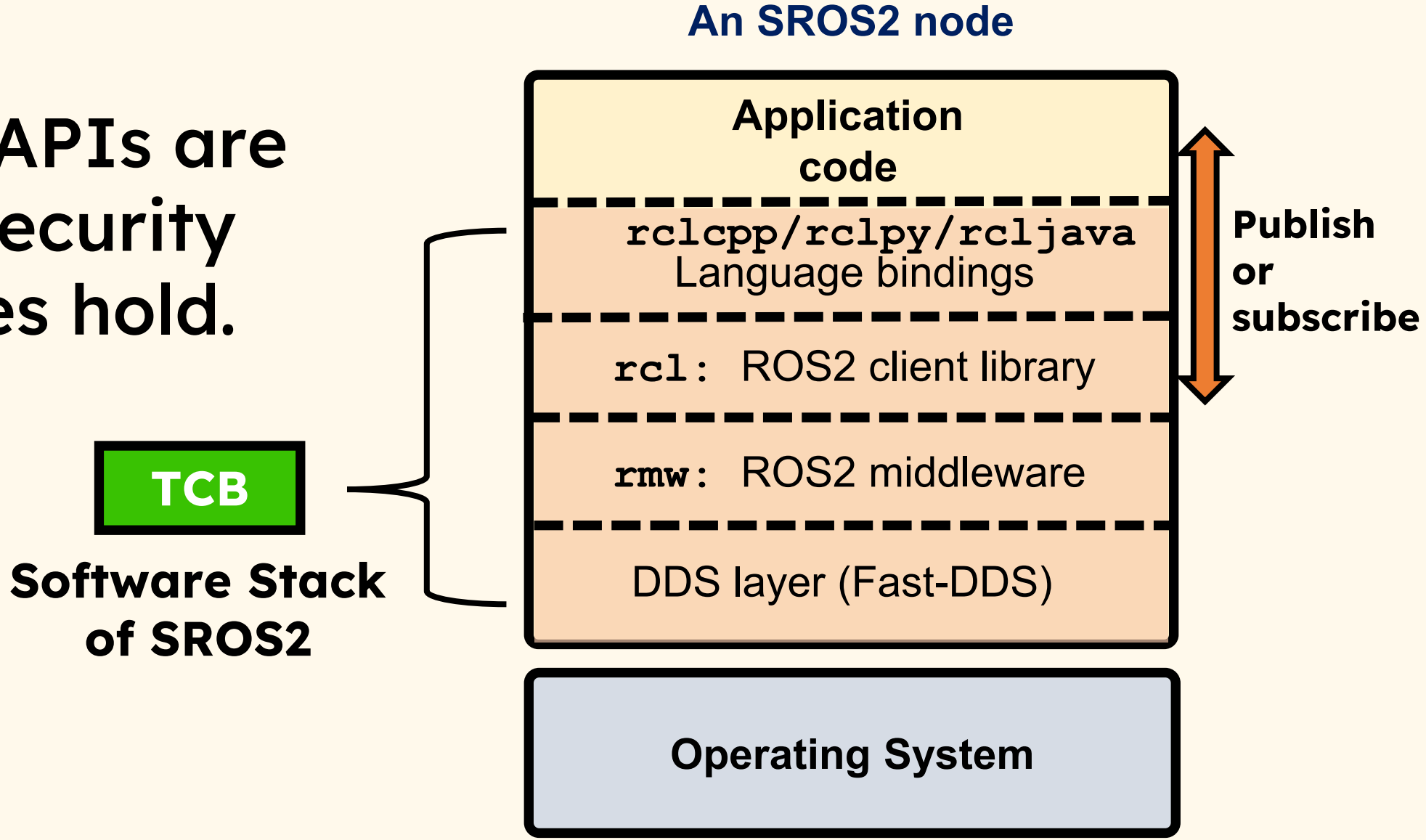
Bypassing the Secure ROS2 API

If SROS2 APIs are used, all security guarantees hold.



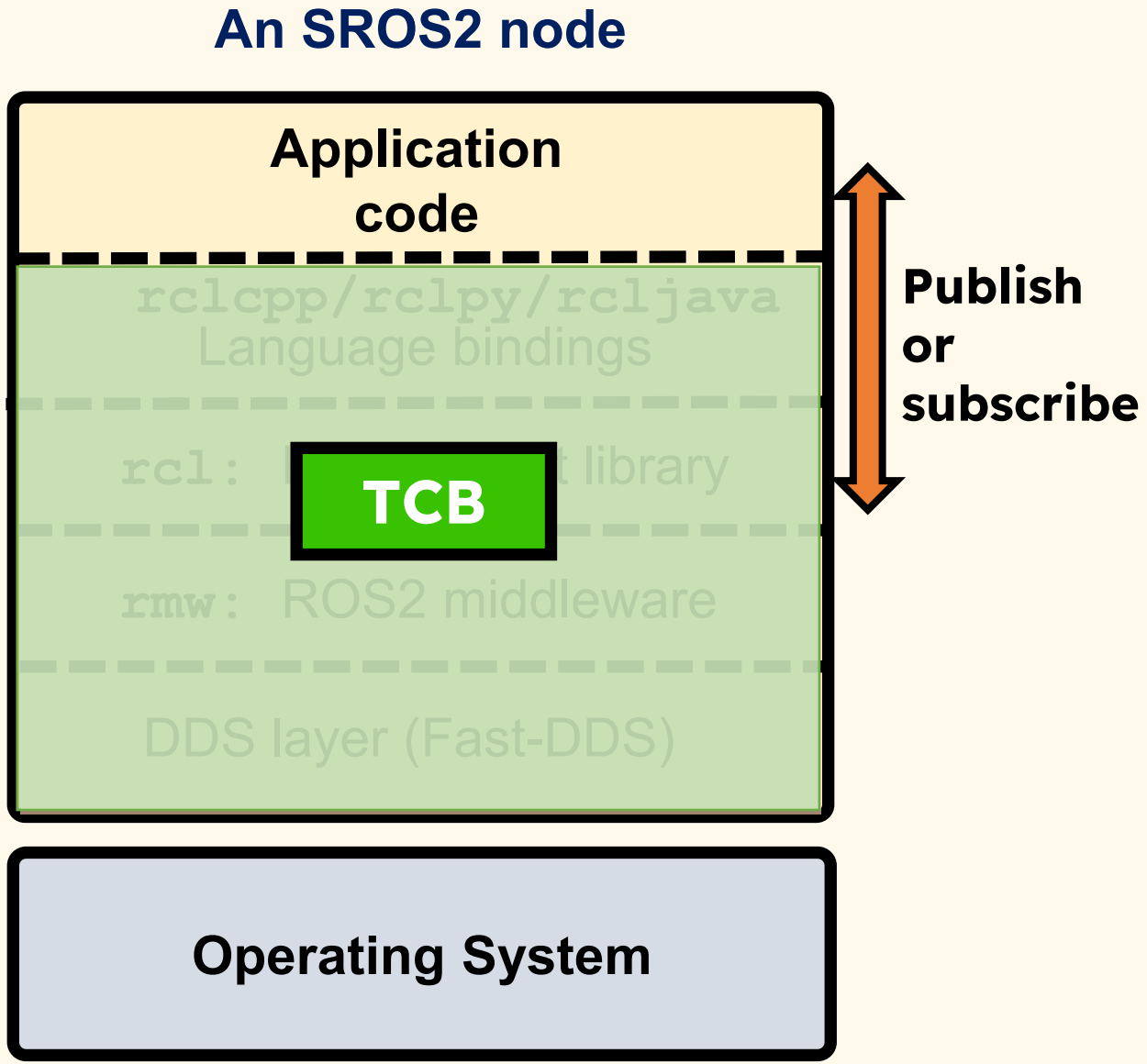
Bypassing the Secure ROS2 API

If SROS2 APIs are used, all security guarantees hold.



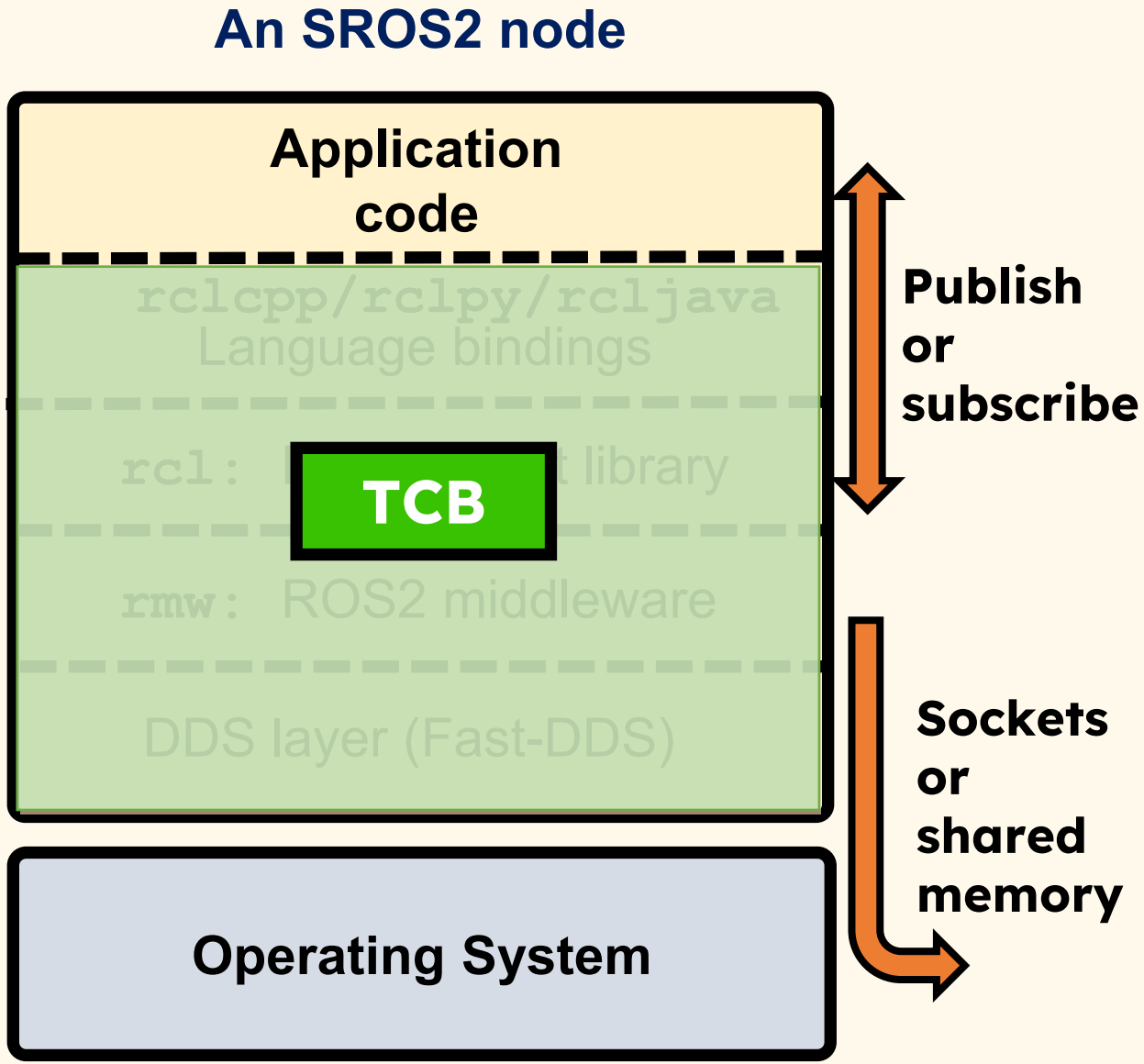
Bypassing the Secure ROS2 API

If SROS2 APIs are used, all security guarantees hold.



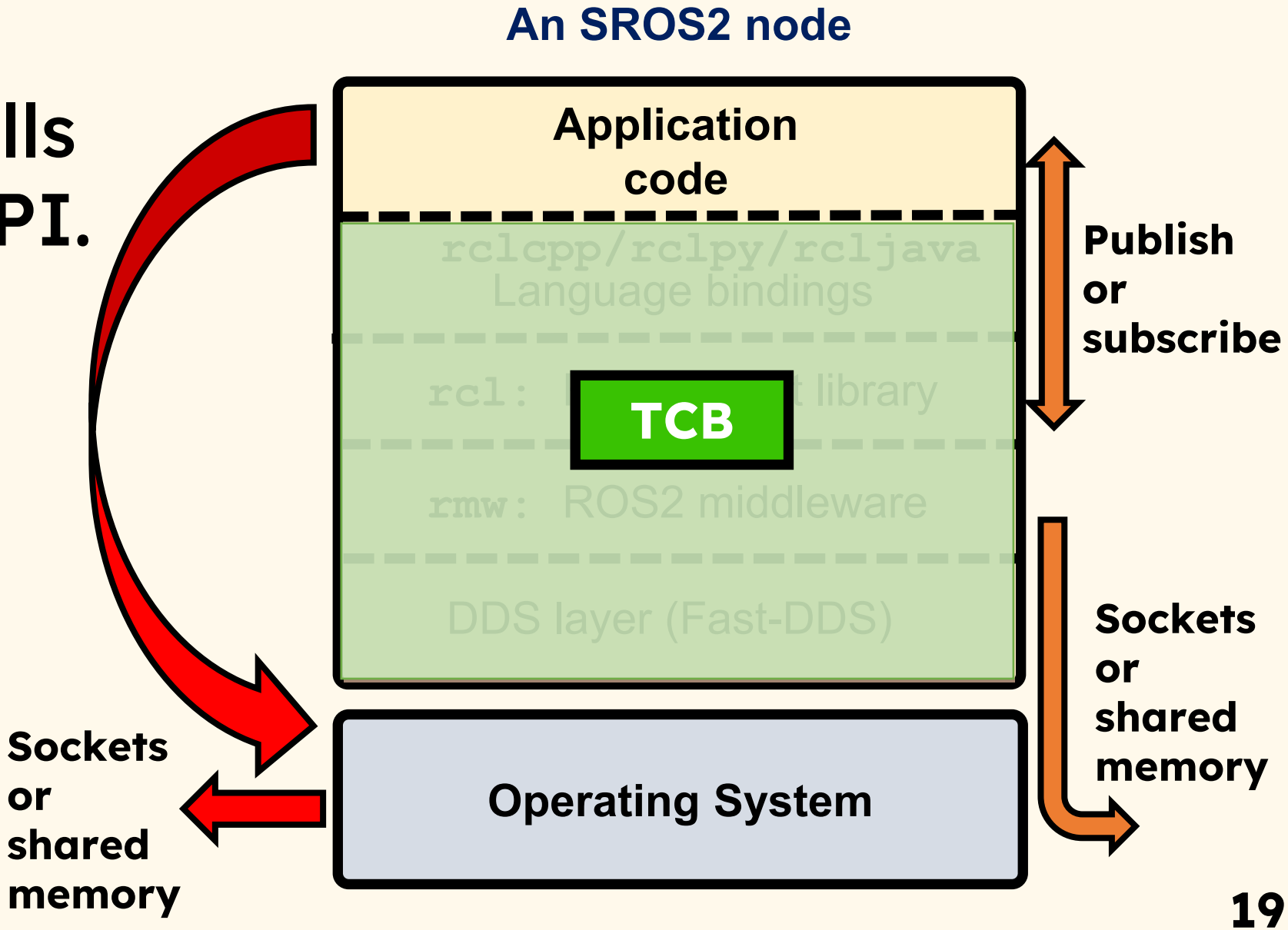
Bypassing the Secure ROS2 API

- If SROS2 APIs are used, all security guarantees hold.
- SROS2 relies on system calls to facilitate Pub-Sub communication.



Bypassing the Secure ROS2 API

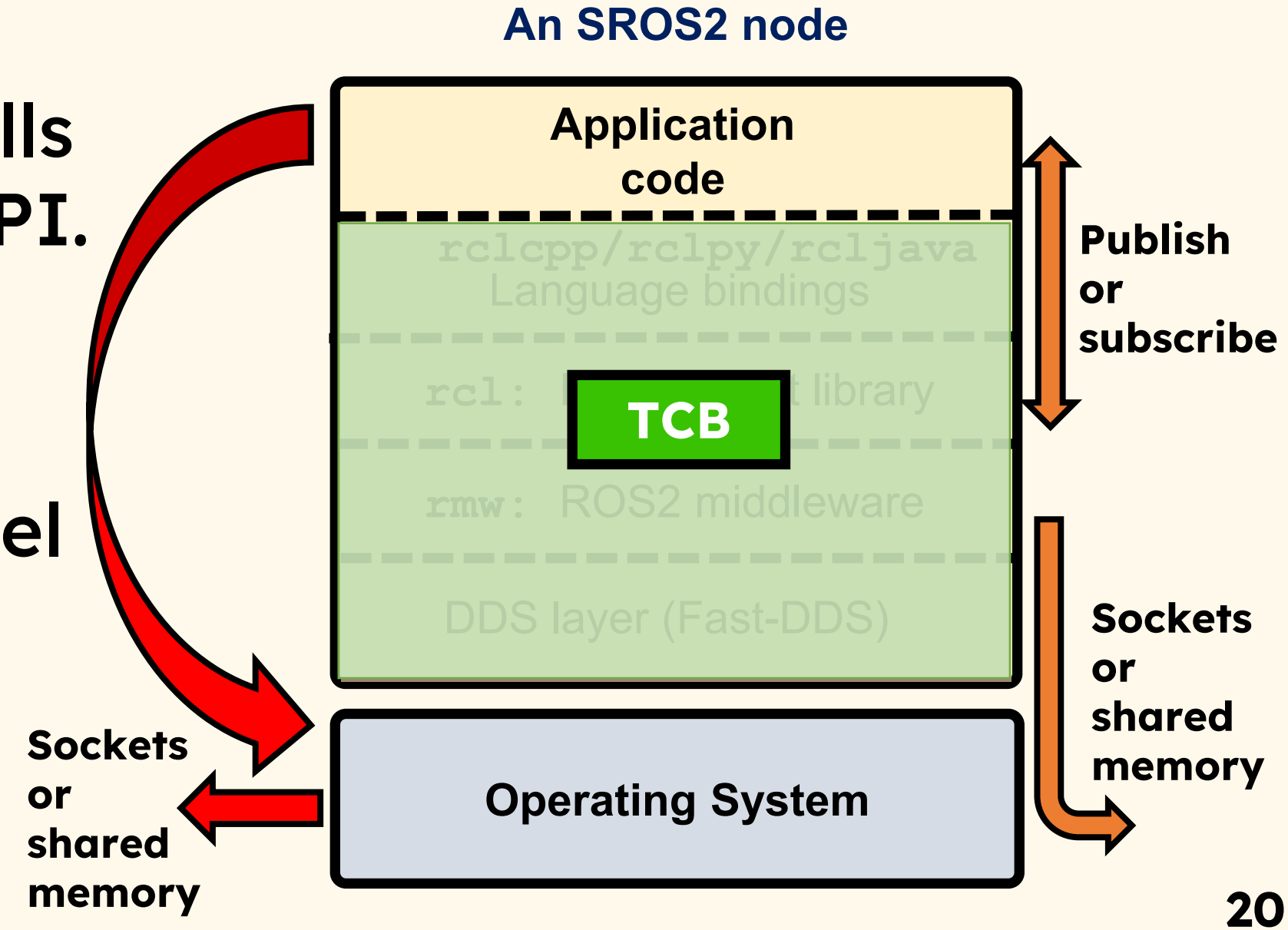
Direct system calls bypass SROS2 API.



Bypassing the Secure ROS2 API

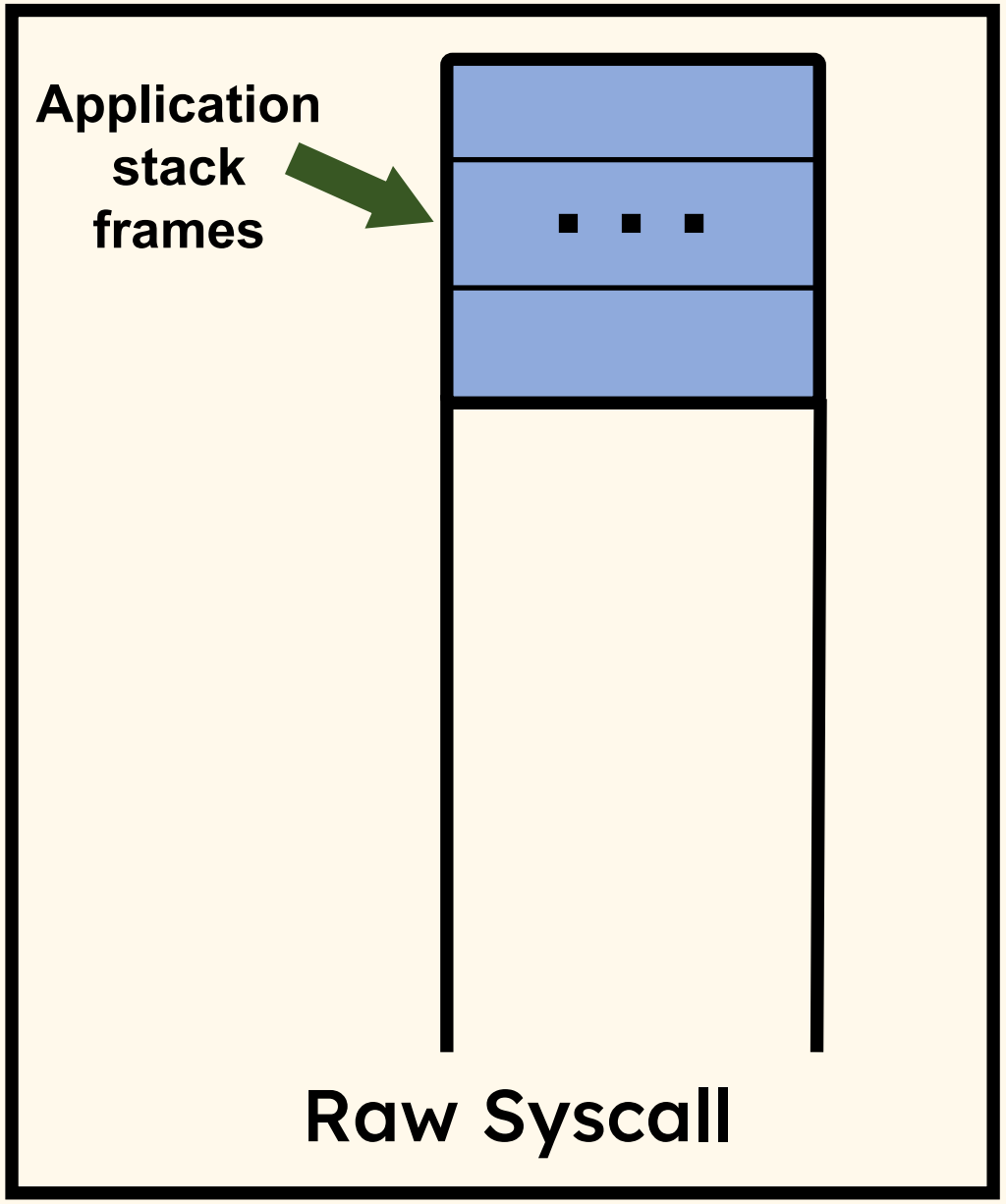
Direct system calls bypass SROS2 API.

Why? No App level sandboxing.



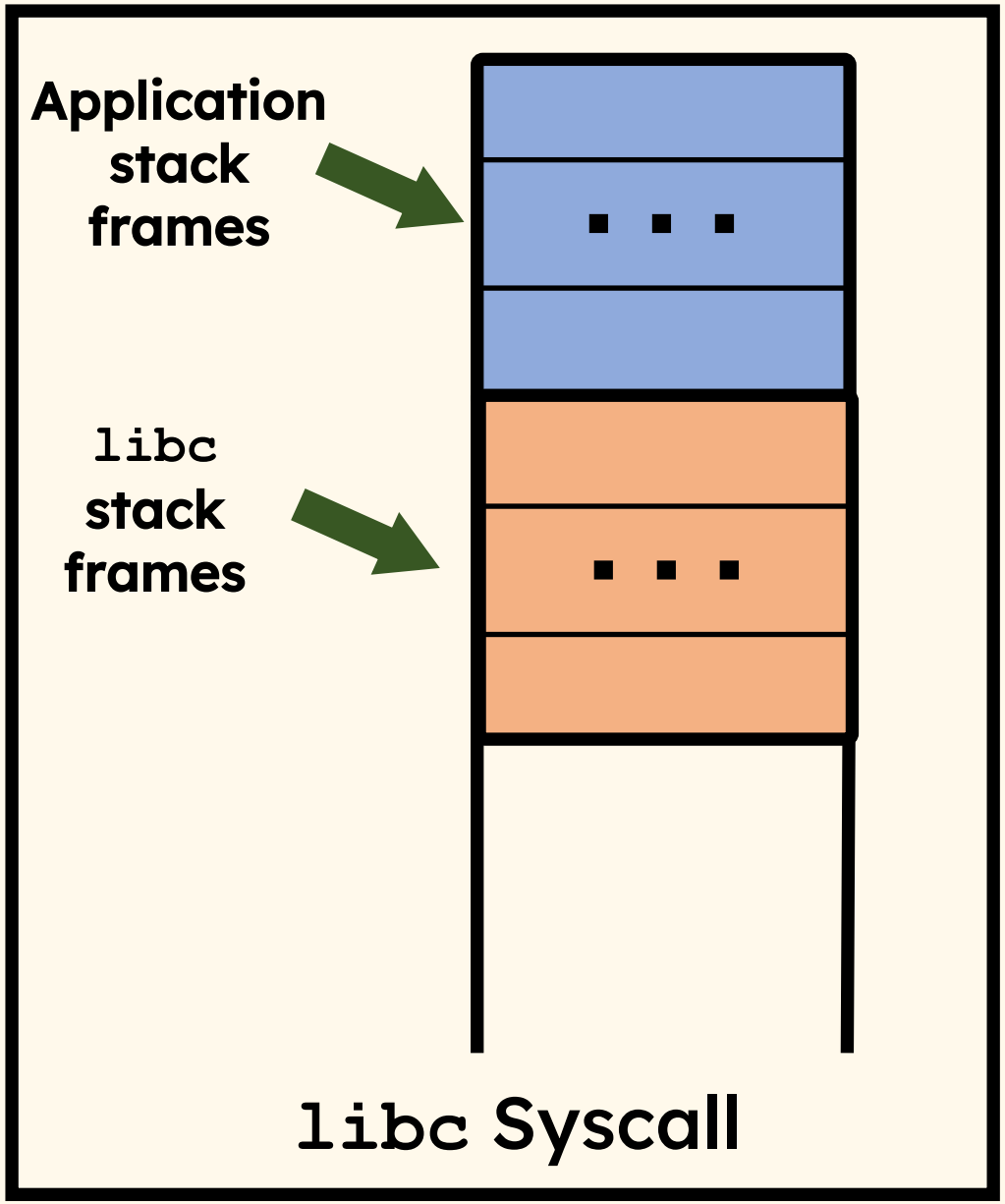
User-Space Stack During System Calls

1) **Raw Syscall:**
   When App invokes a raw system call.



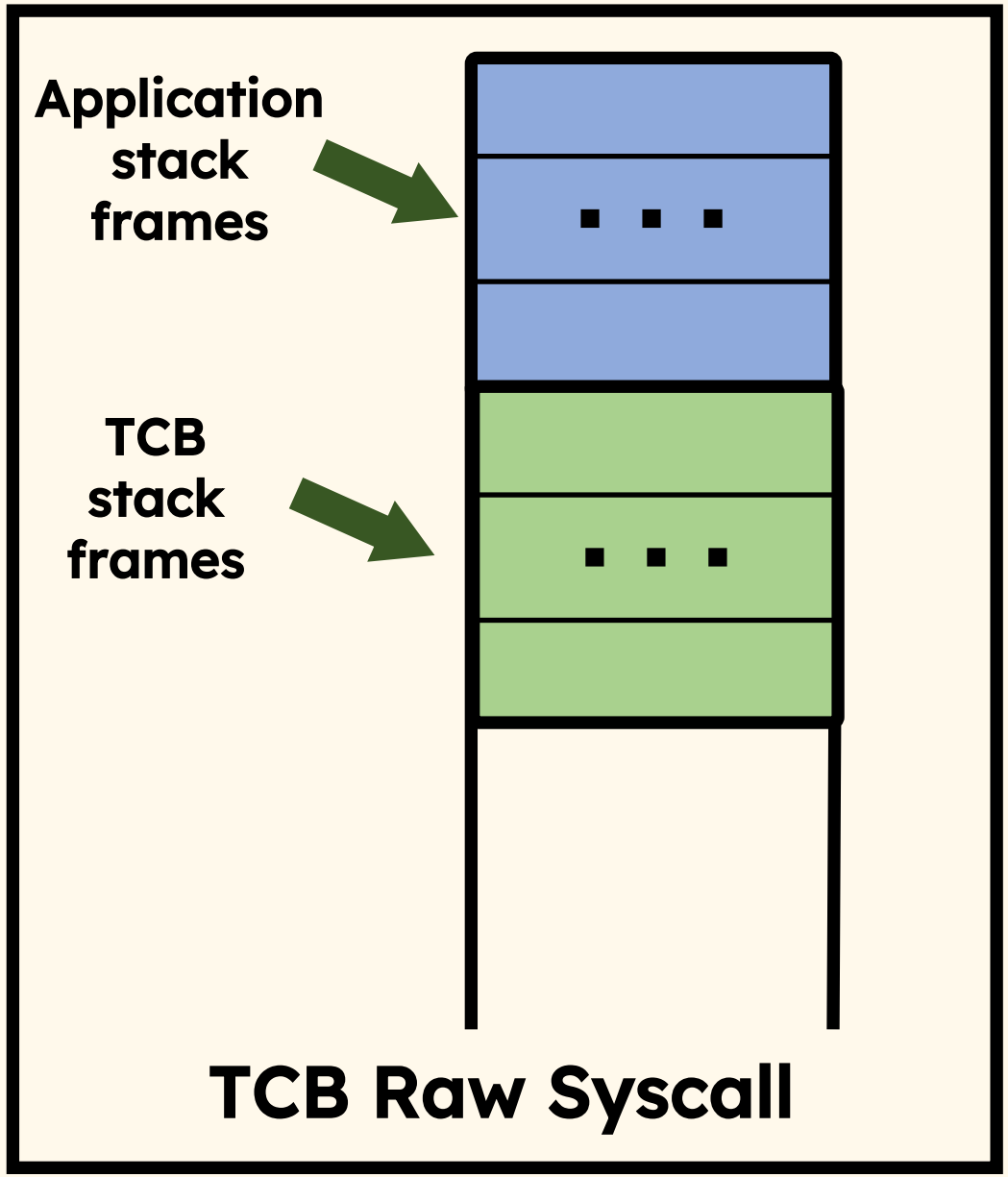
User-Space Stack During System Calls

2) `libc` Syscall: When App invokes a system call via `libc`.



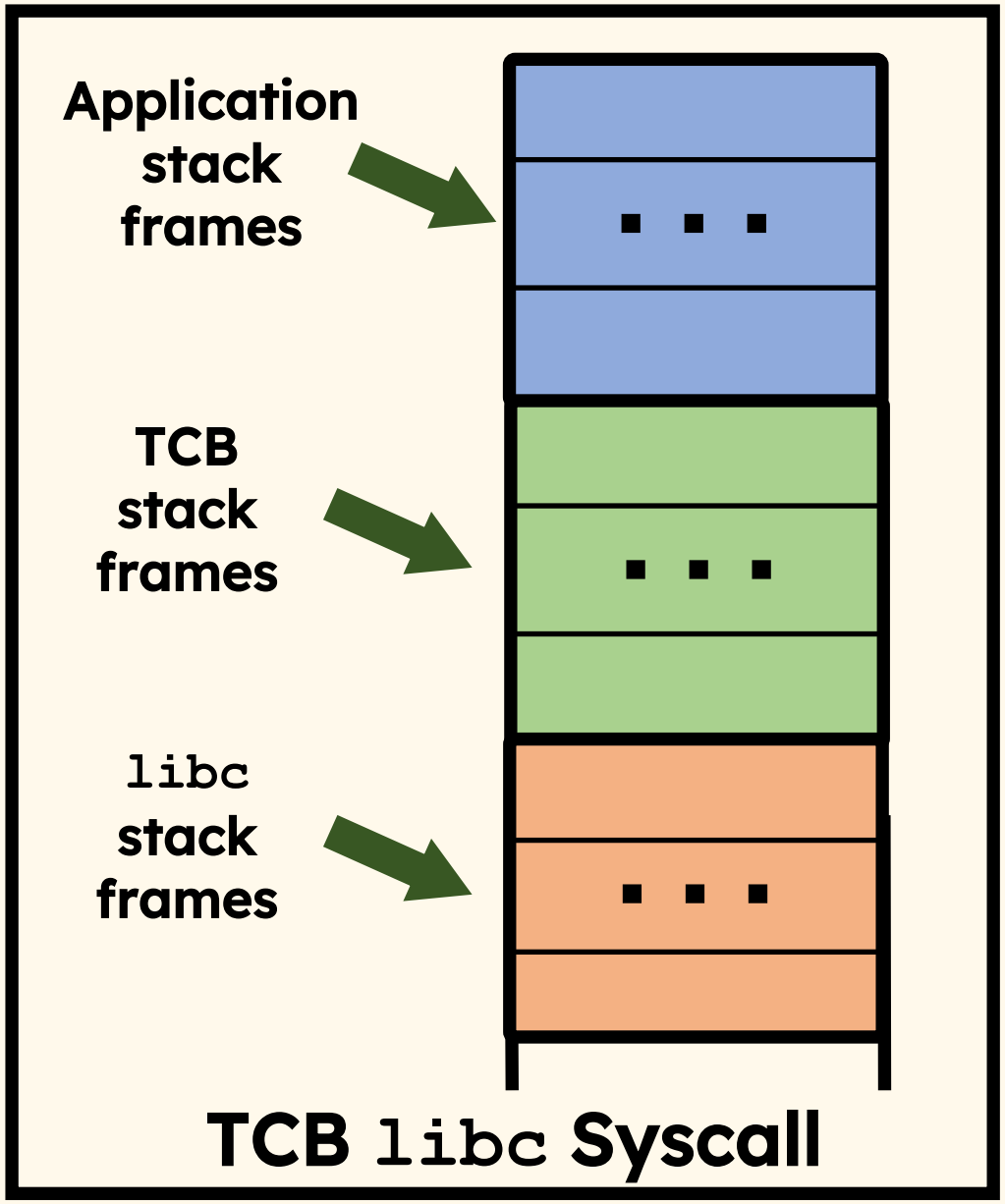
User-Space Stack During System Calls

3) TCB Raw Syscall: When TCB invokes a raw system call.

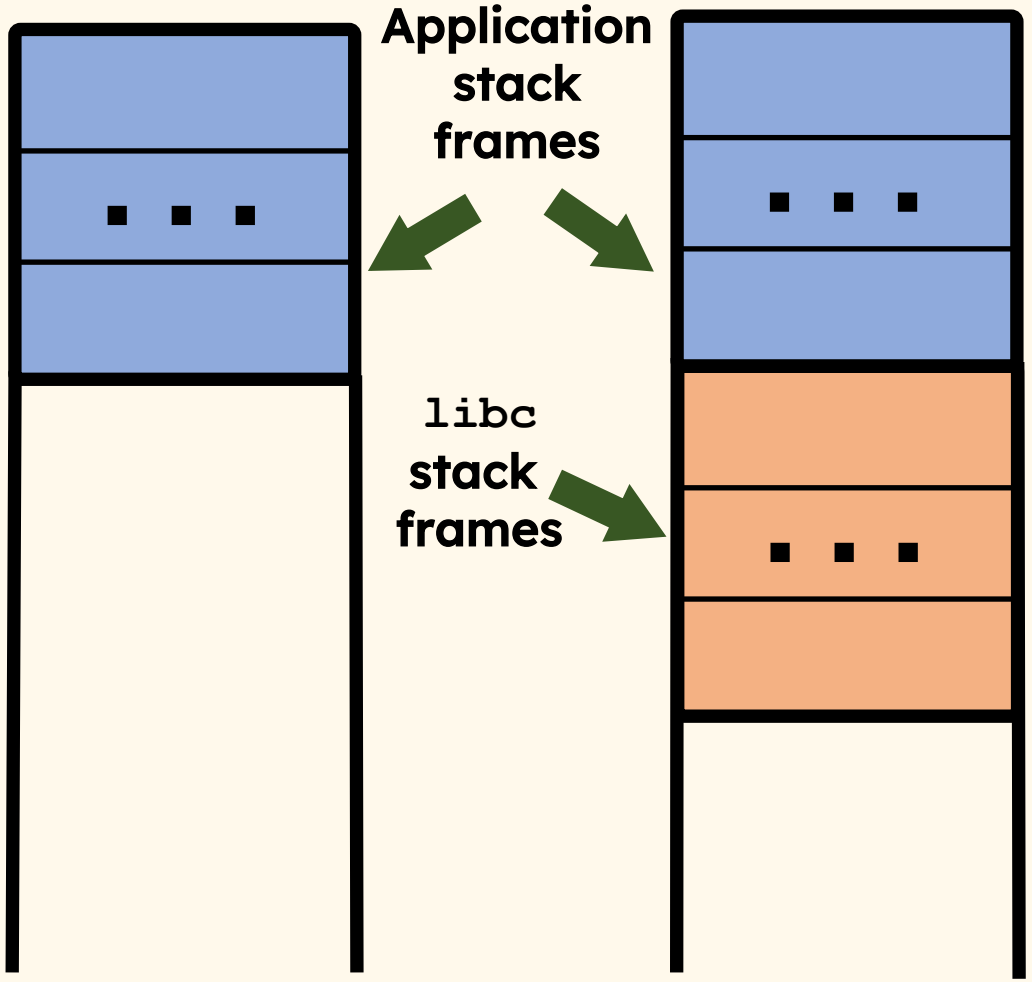


User-Space Stack During System Calls

4) TCB `libc` Syscall: When TCB invokes a system call via `libc`.

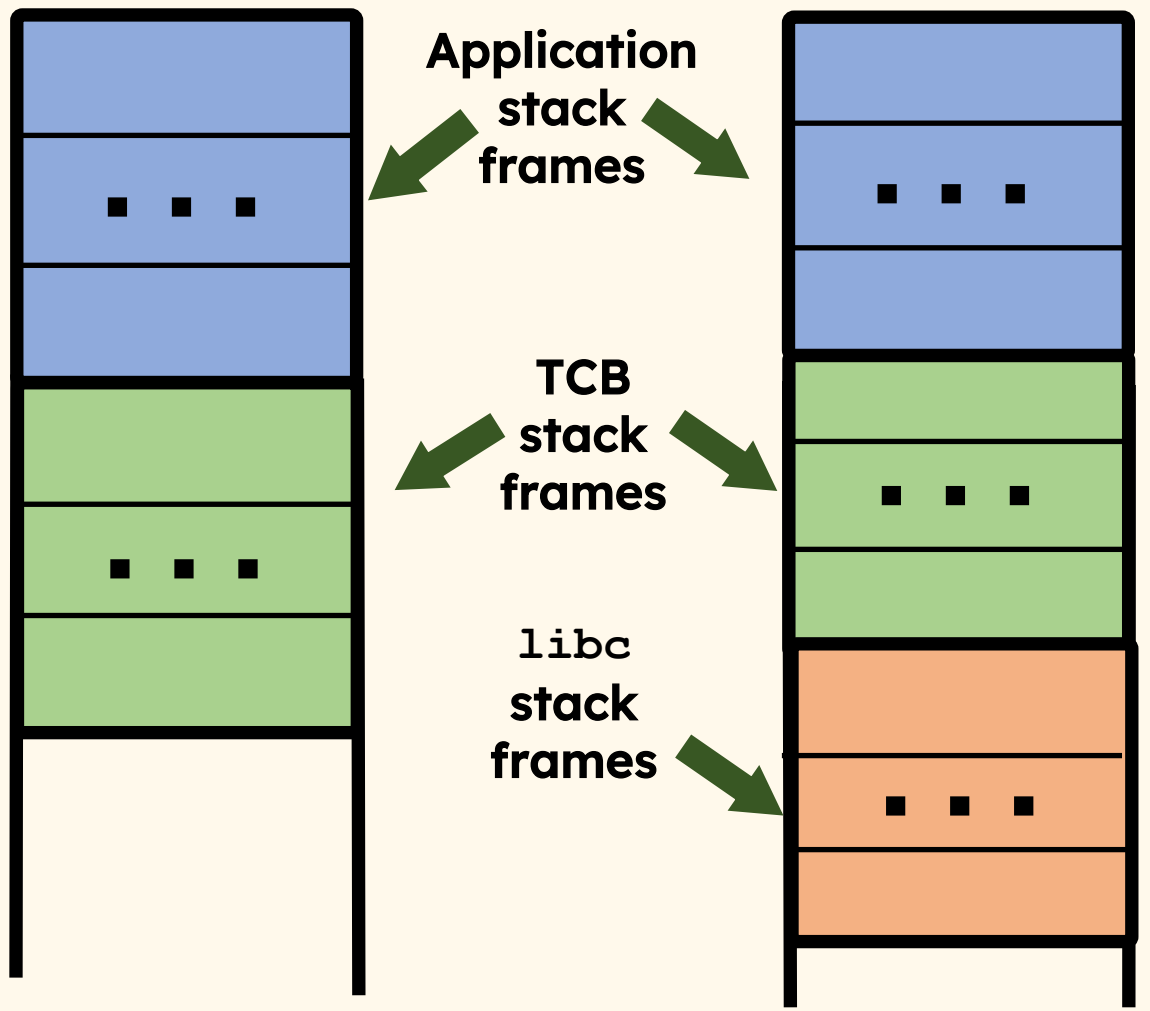


Summary of four scenarios



Raw Syscall

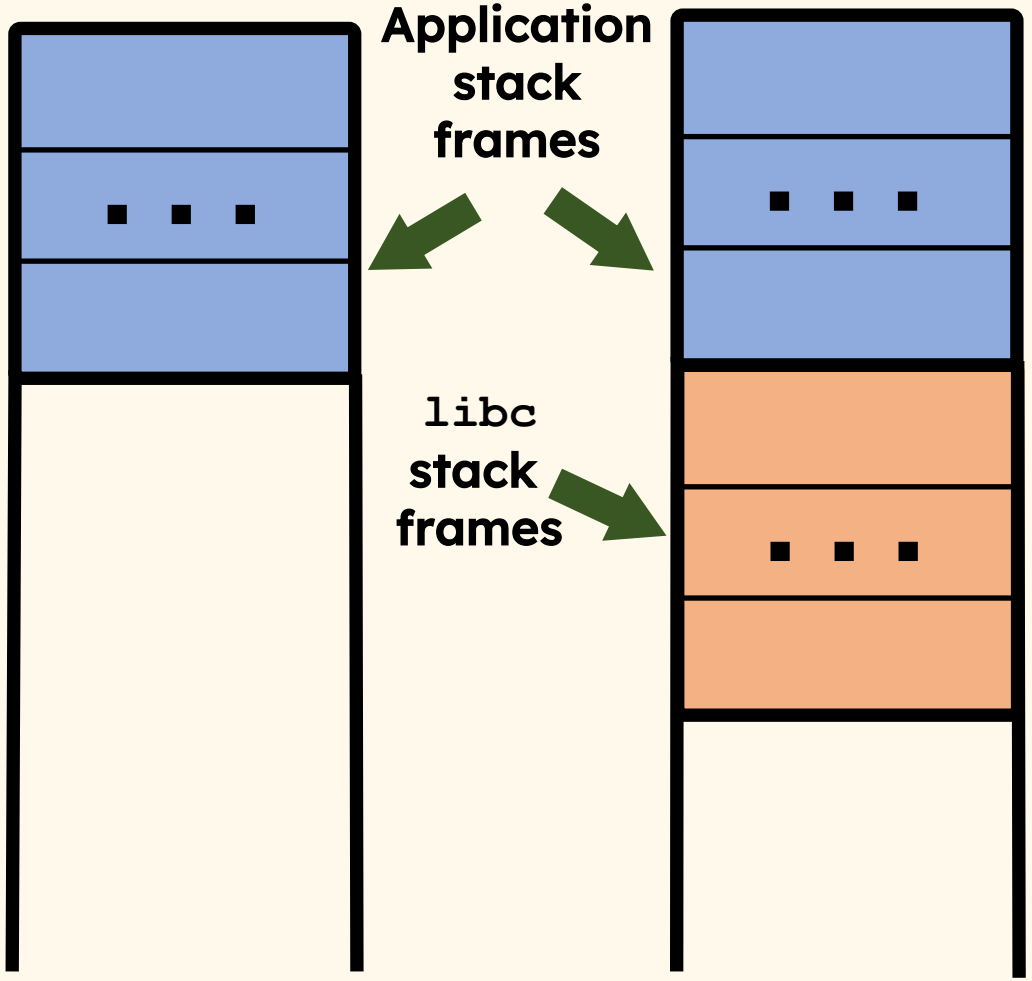
`libc` Syscall



TCB Raw Syscall

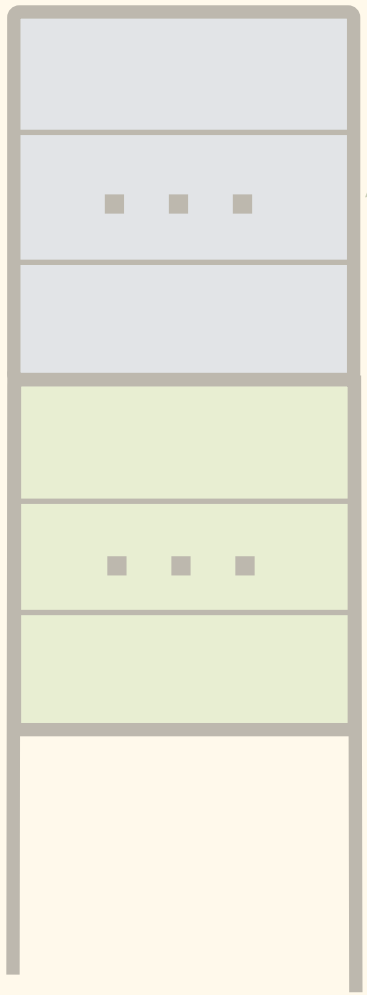
TCB `libc` Syscall

Summary of four scenarios

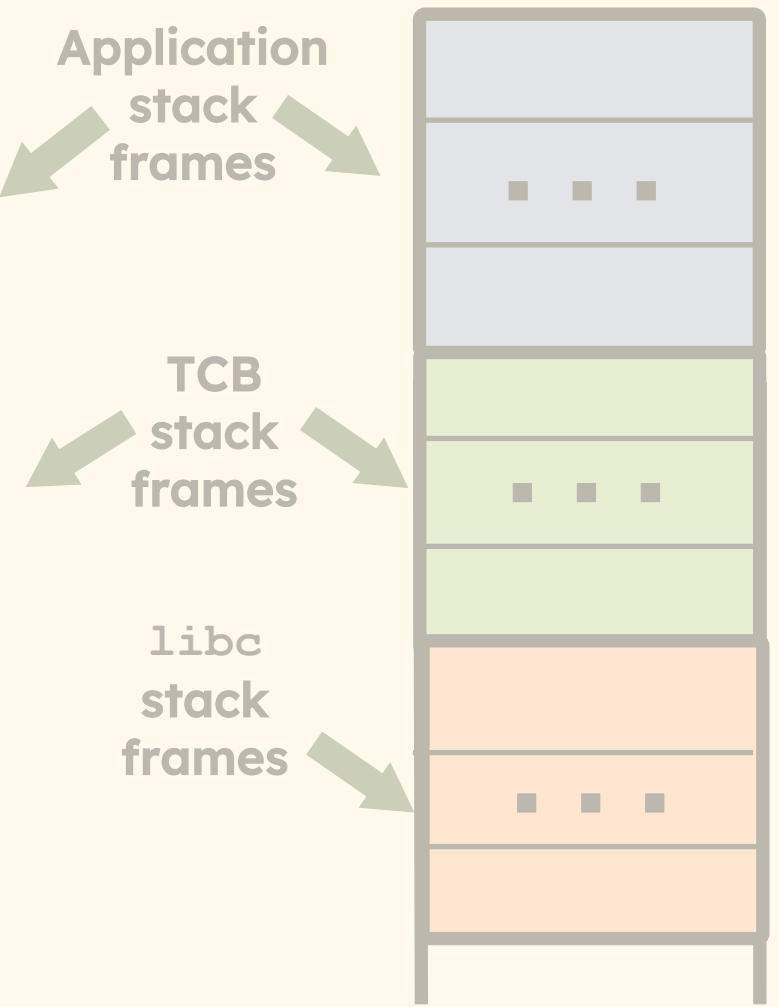


Raw Syscall

`libc` Syscall

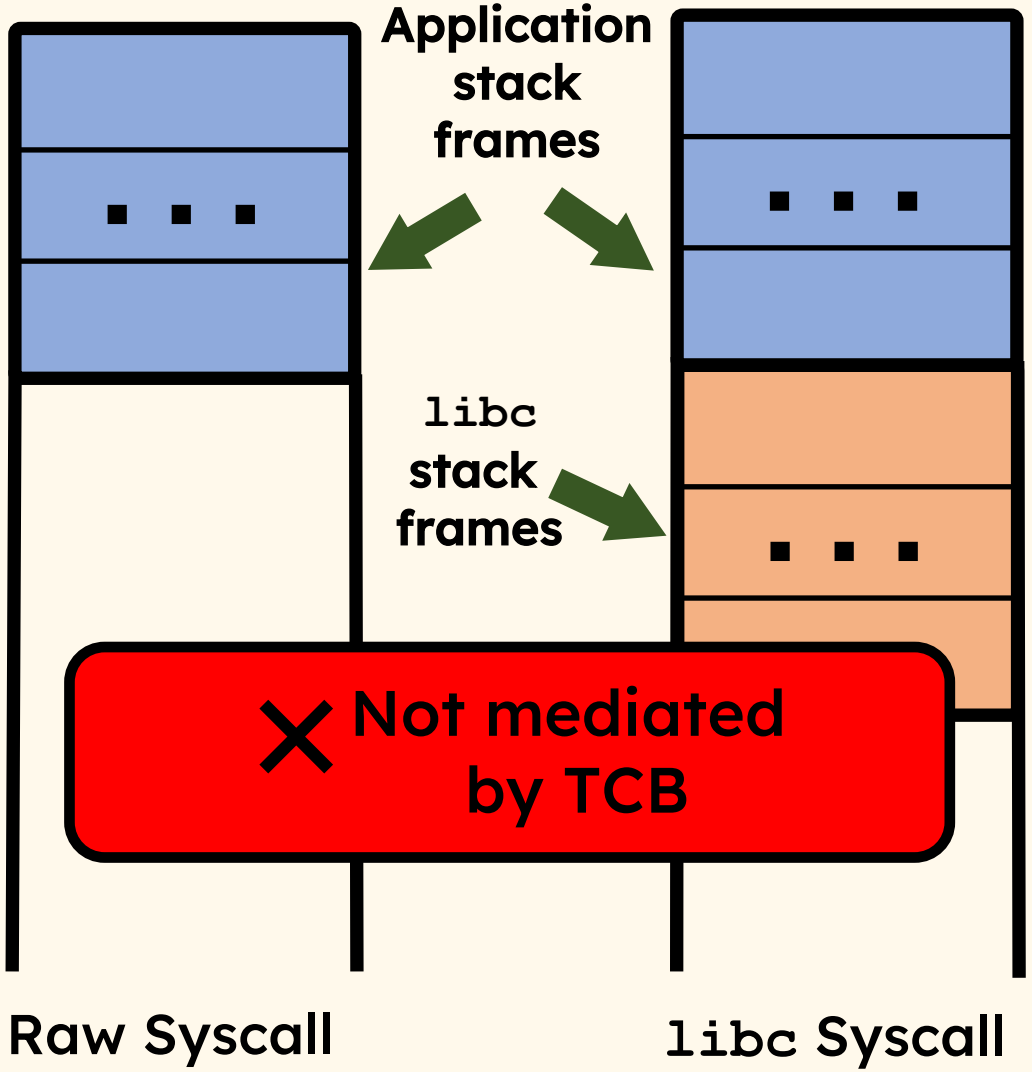


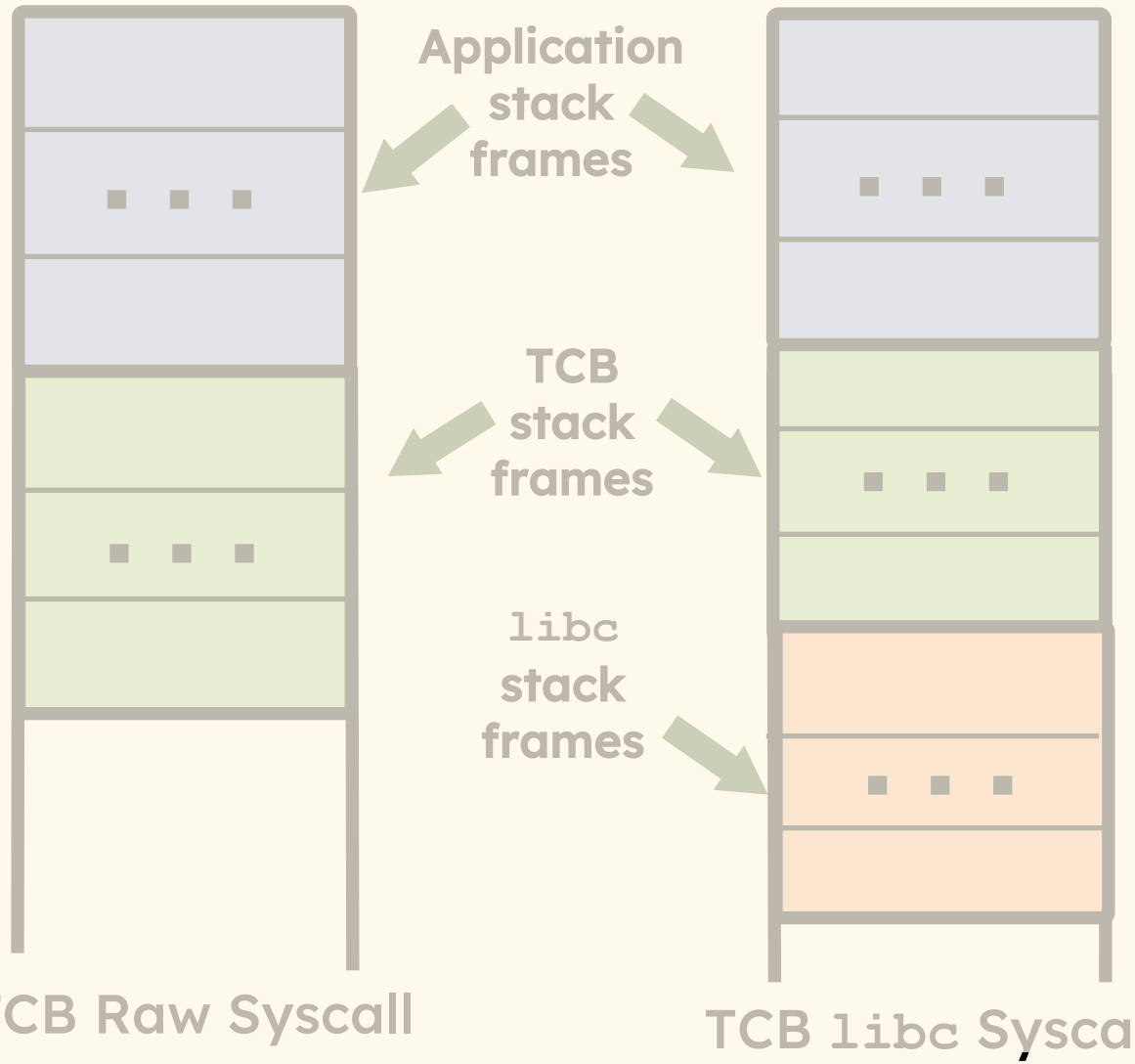
TCB Raw Syscall



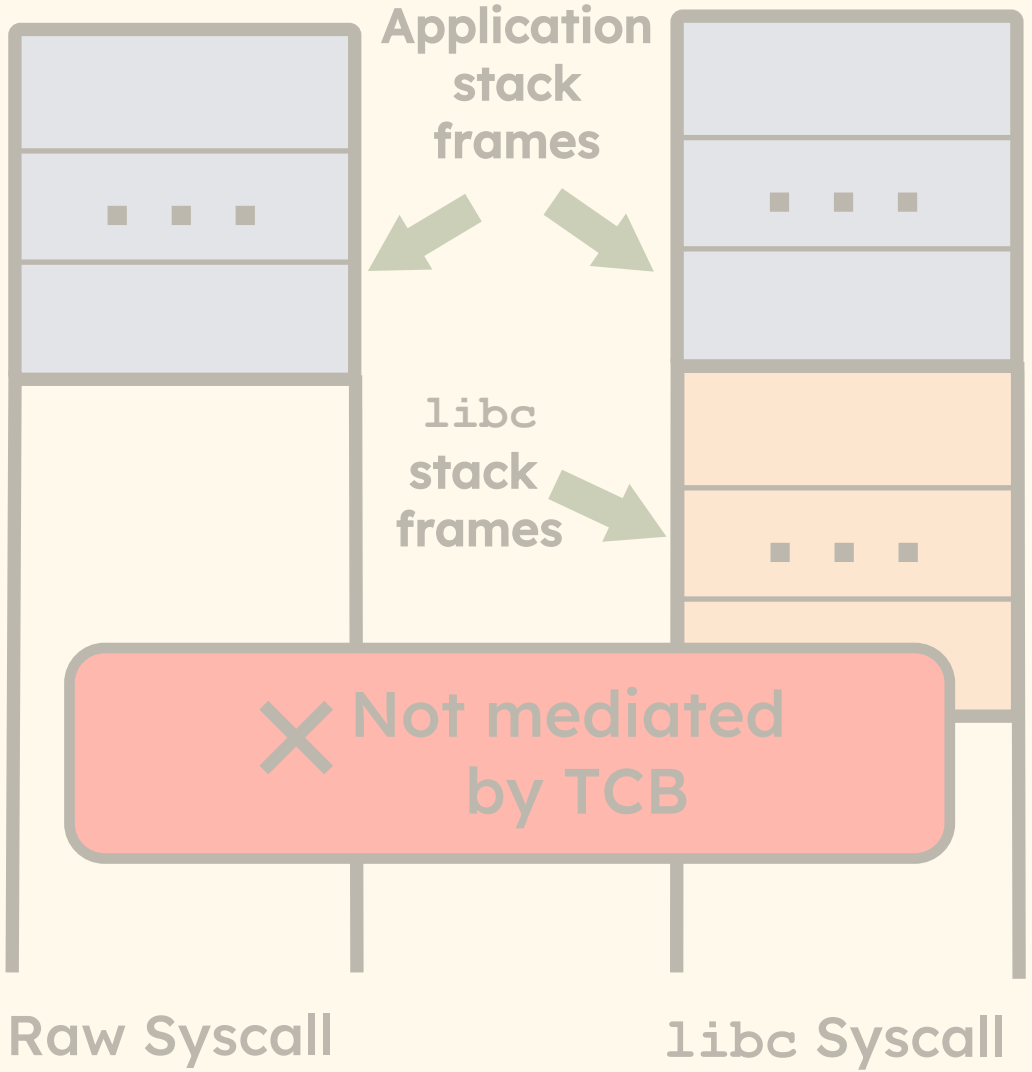
TCB `libc` Syscall

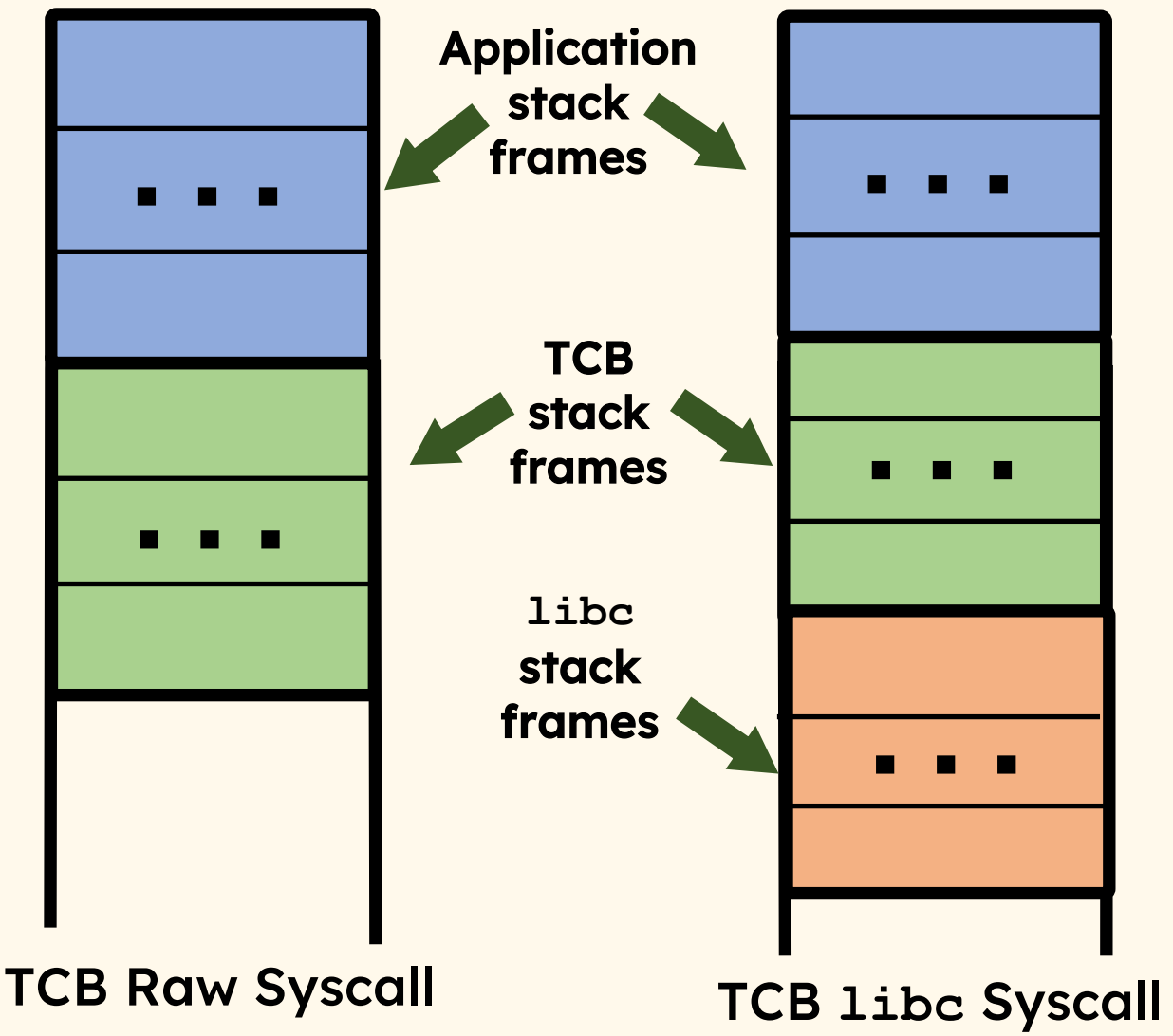
Summary of four scenarios



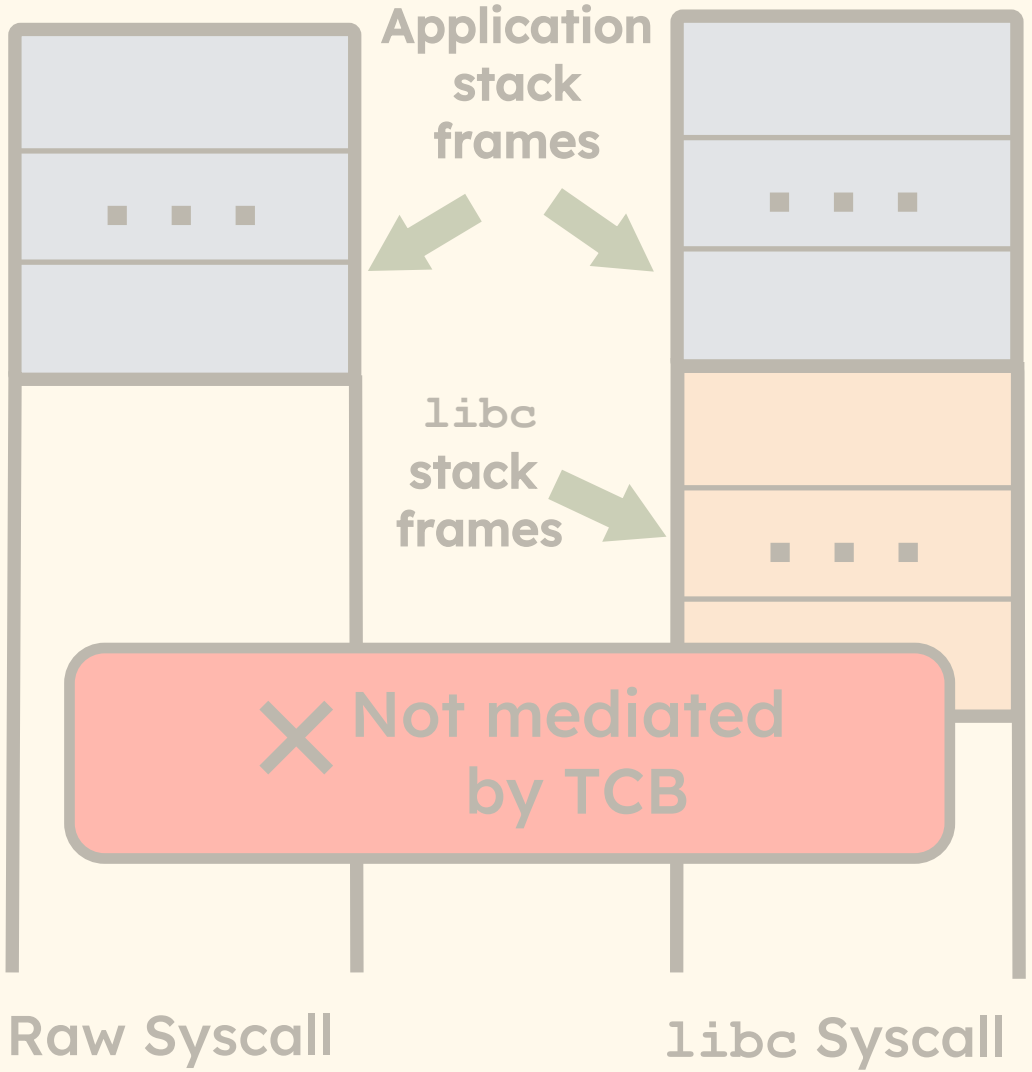


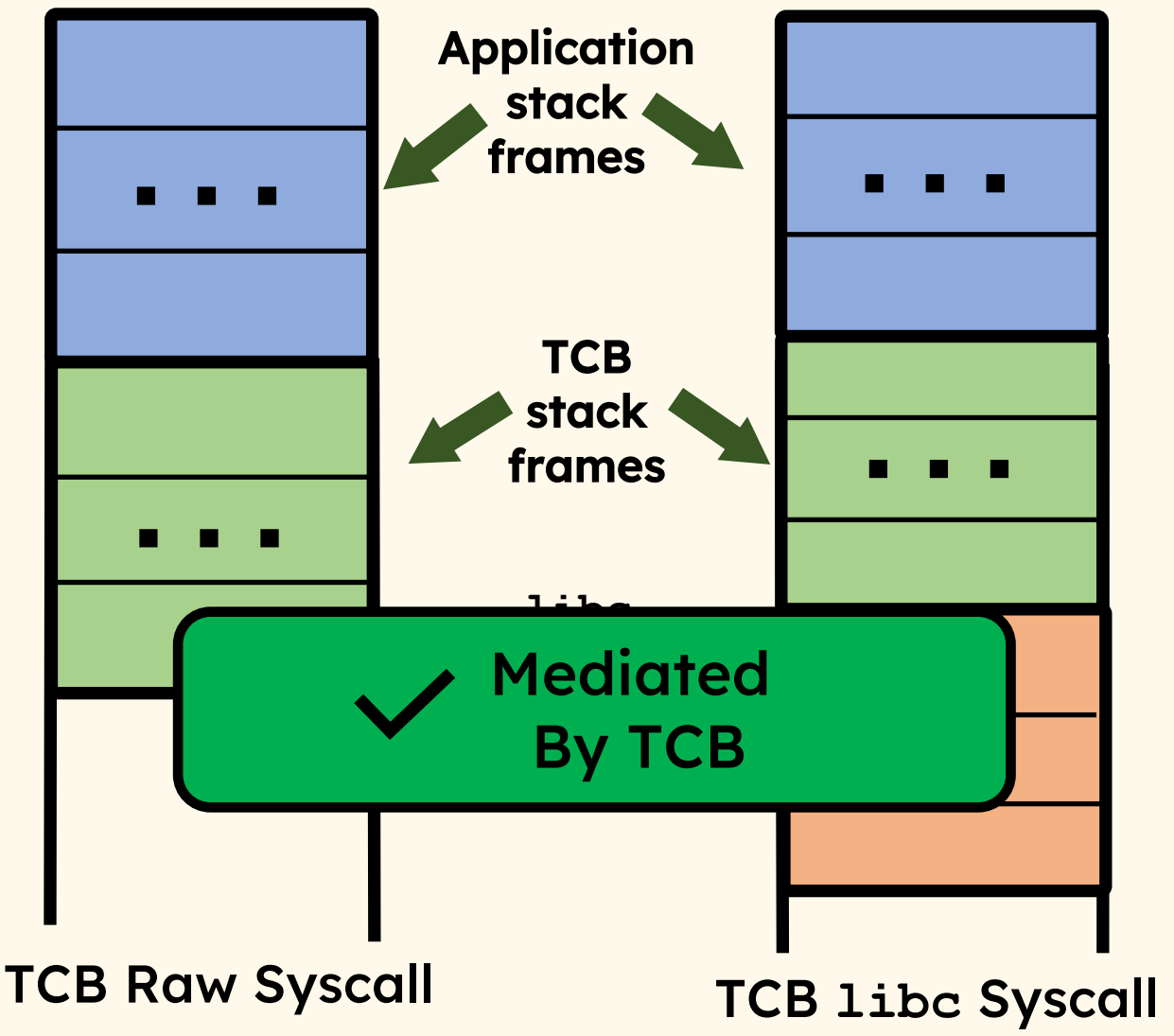
Summary of four scenarios



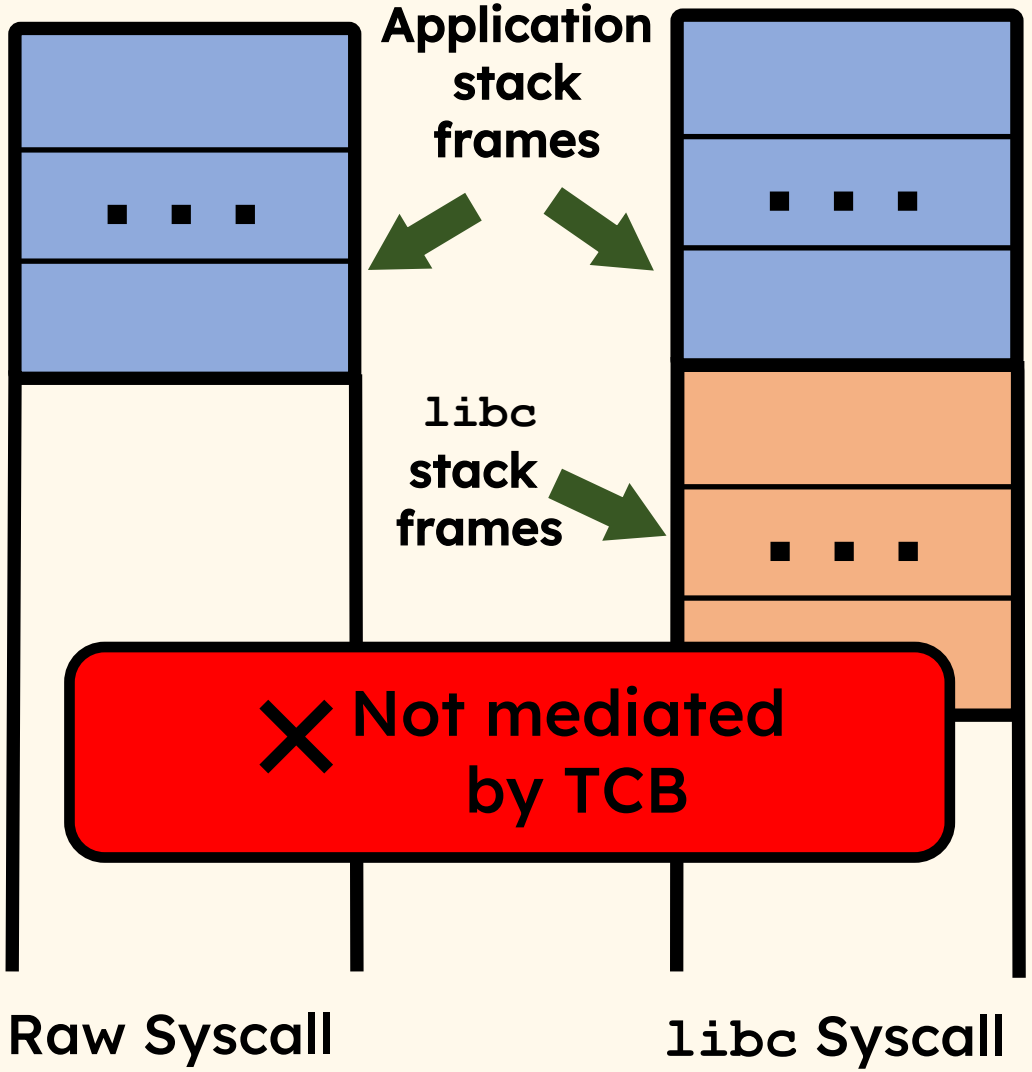


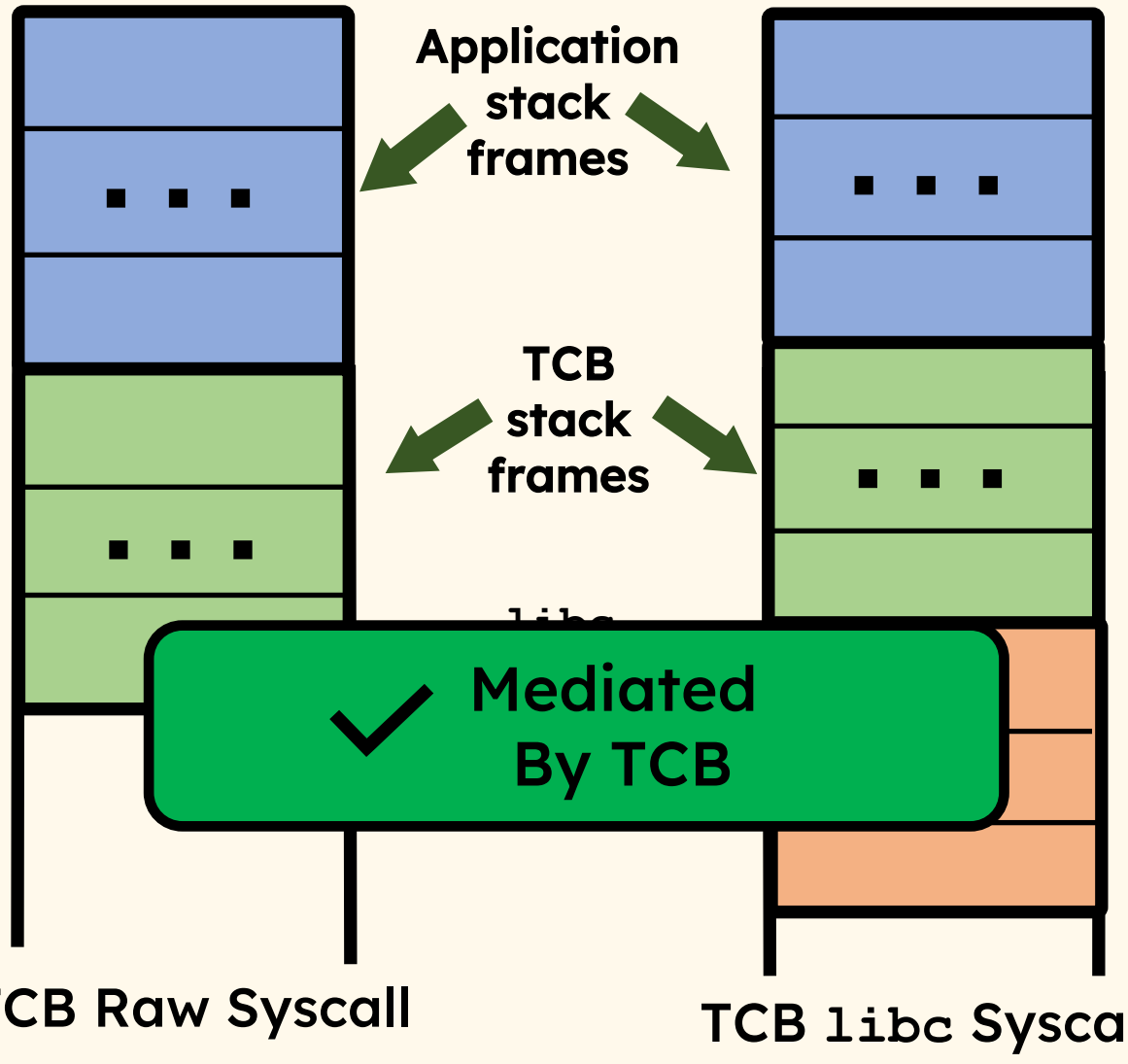
Summary of four scenarios





Summary of four scenarios





ESROS2-Invariant

When a system call is issued, the user-space stack of activation records must contain stack frames of TCB functions between the top of the stack and the stack frame of the most recently invoked entry point of the application’s logic.

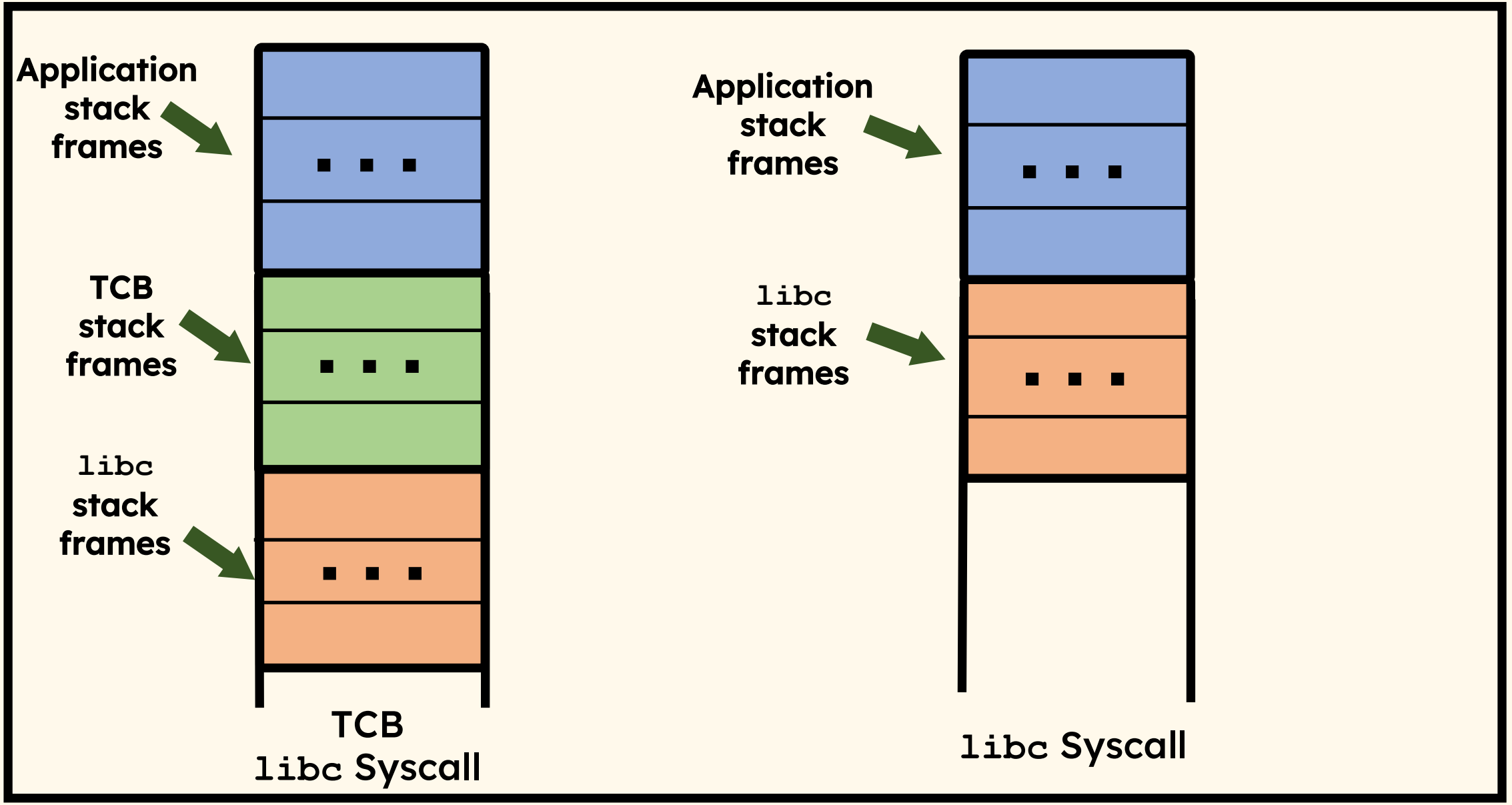
ESROS2-Invariant

When a system call is issued, the user-space stack of activation records must contain stack frames of TCB functions between the top of the stack and the stack frame of the most recently invoked entry point of the application's logic.

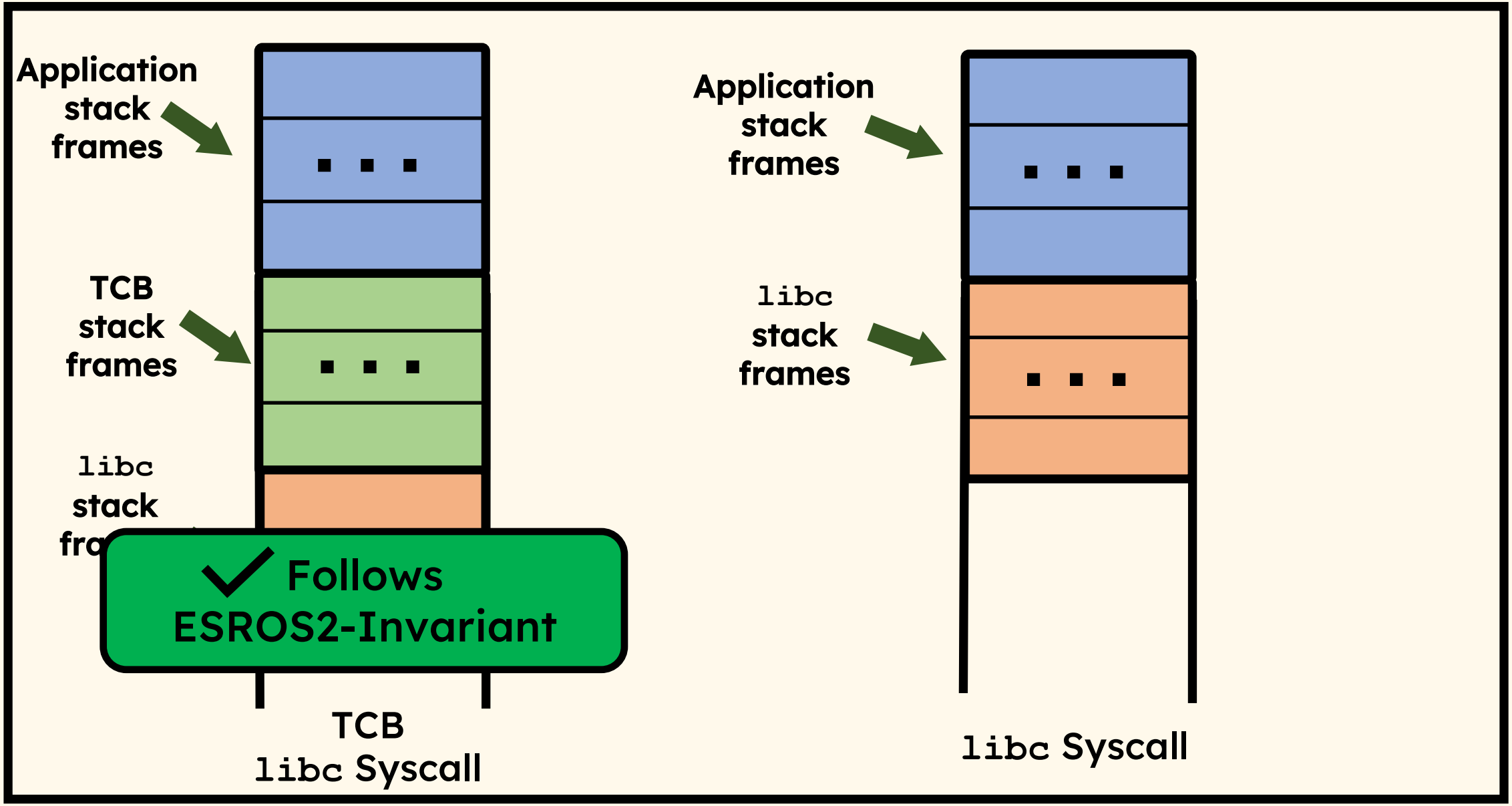
Key underlying intuition

Every system call must be mediated by the TCB.

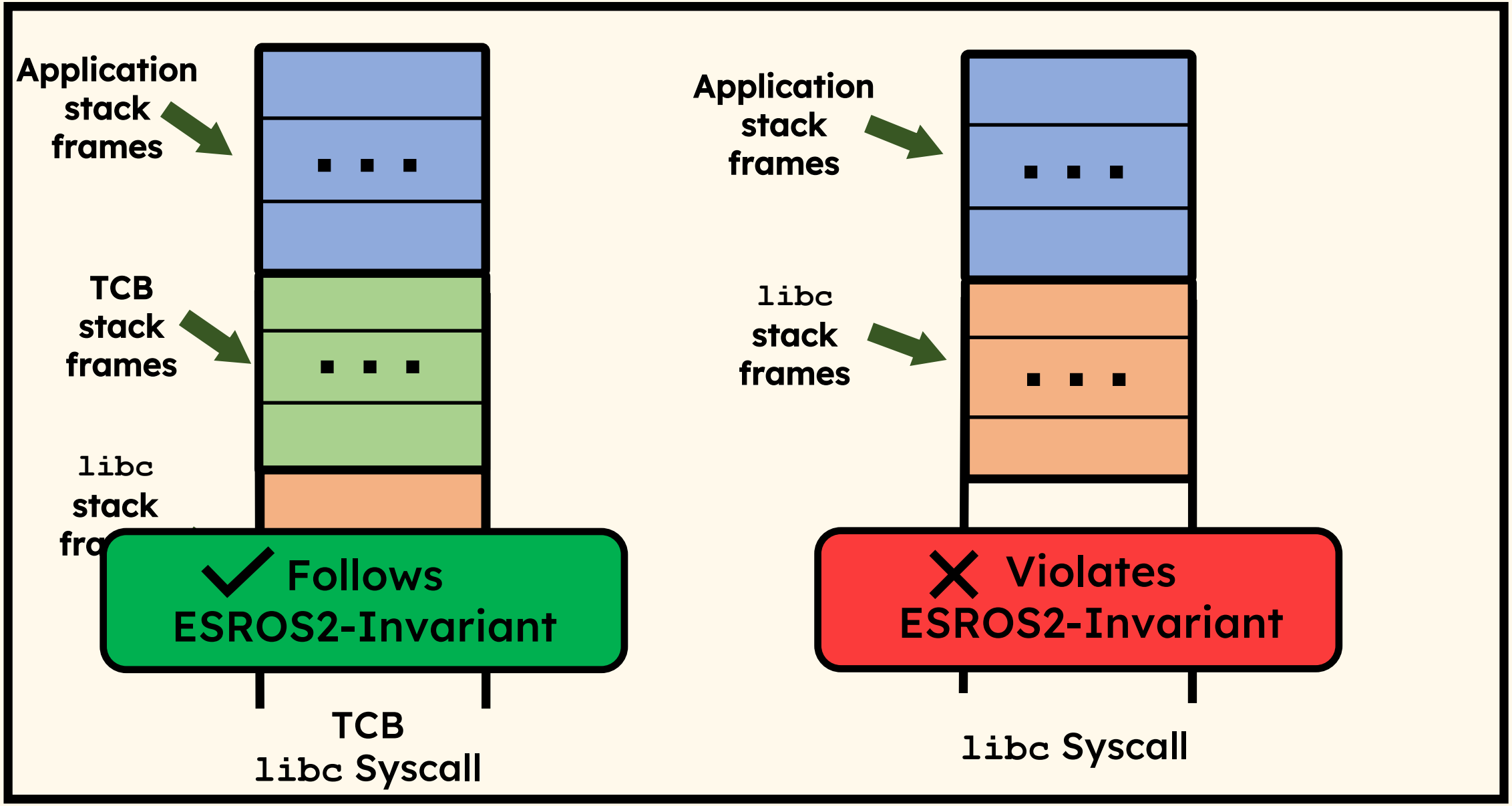
Main Challenge : `libc` usage



Main Challenge : `libc` usage

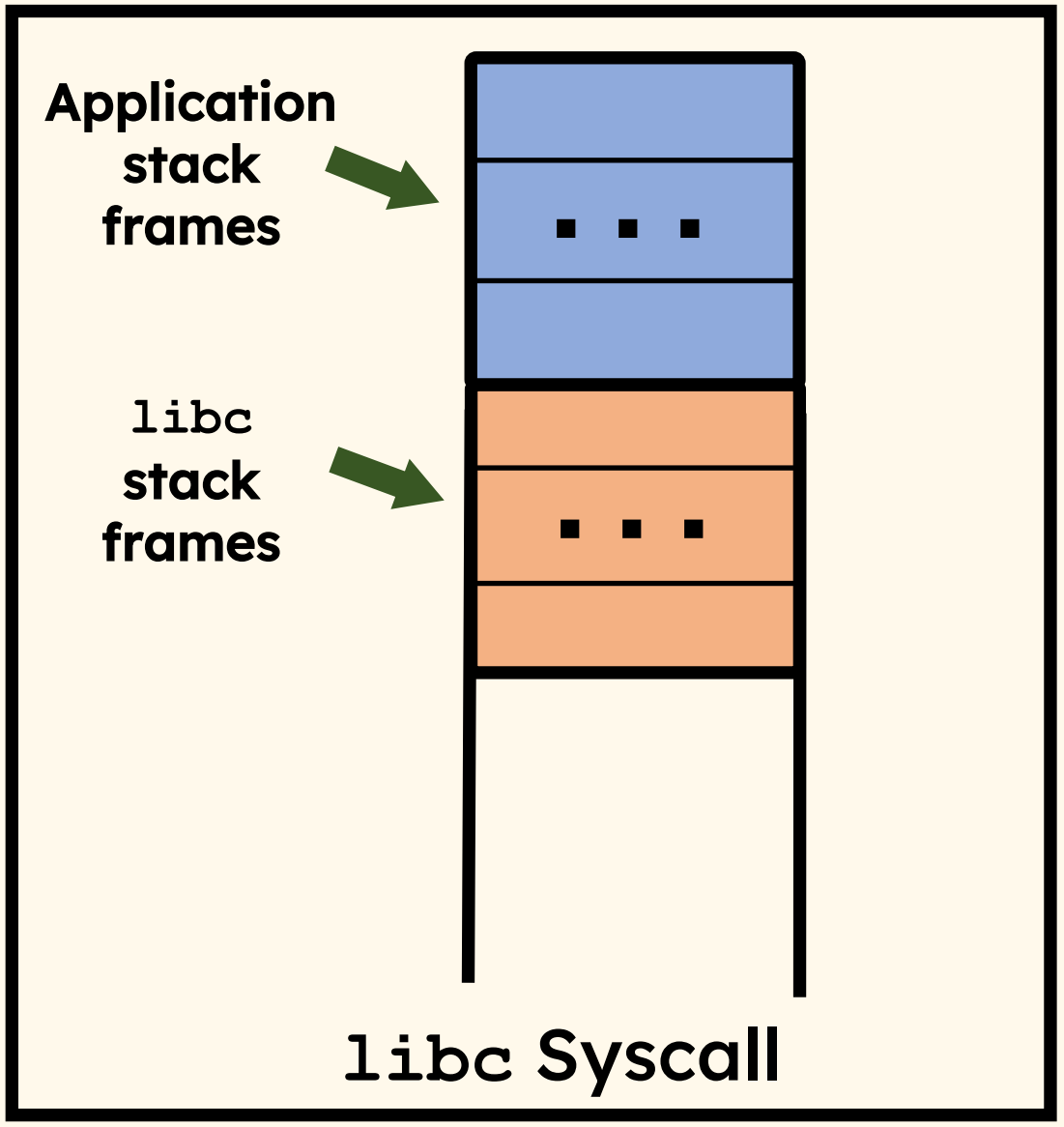


Main Challenge : `libc` usage



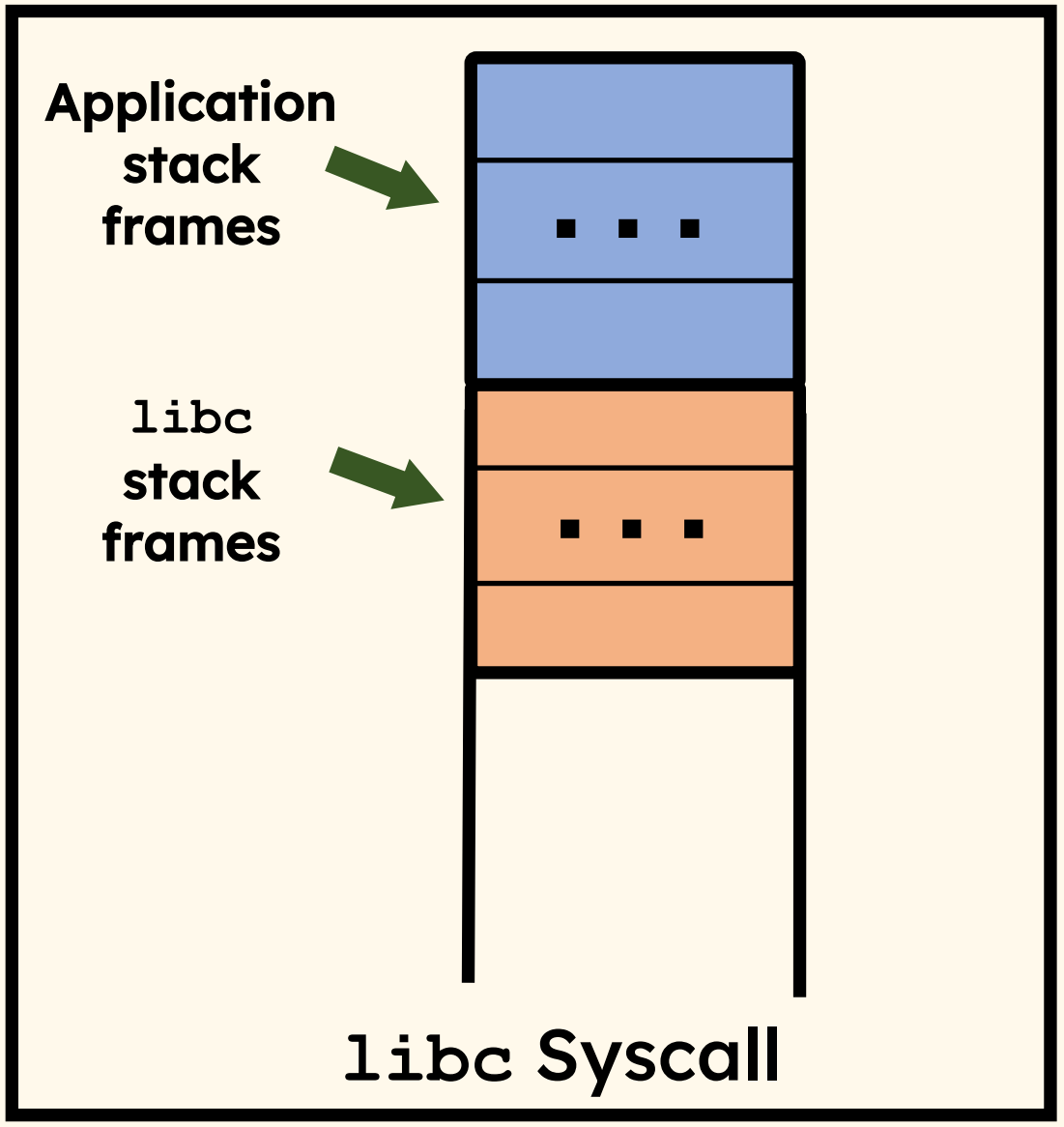
Main Challenge : `libc` usage

- Very common in practice.
- App use `libc` APIs for various operations like printf or fopen which internally invoke system calls.



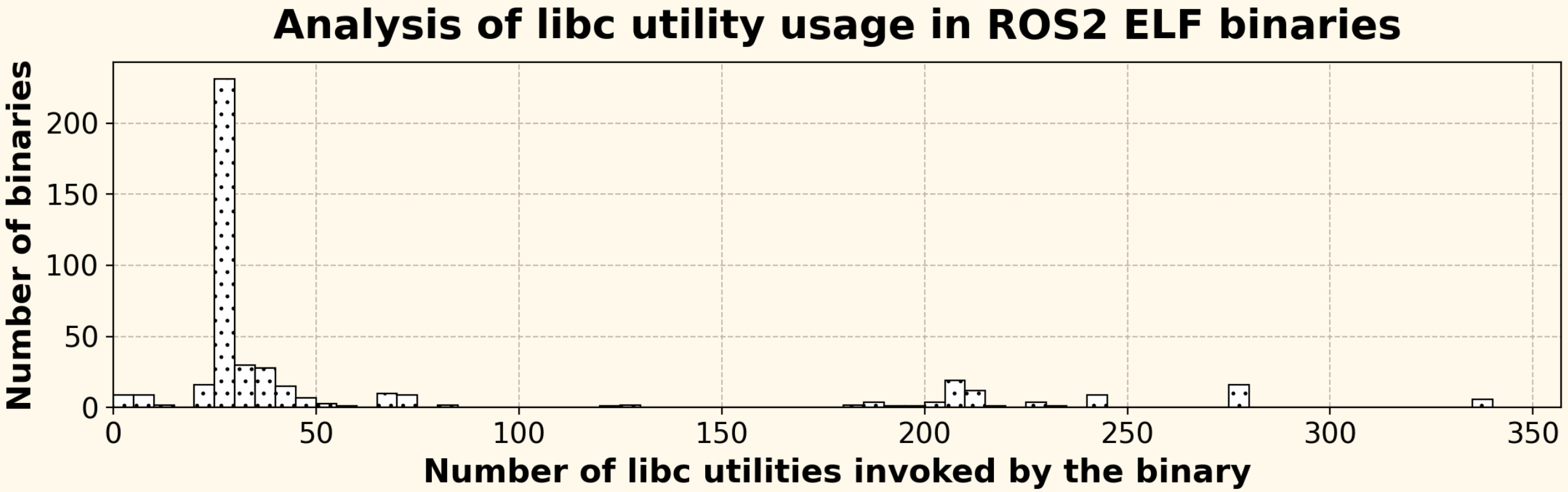
Main Challenge : `libc` usage

- Blocking all `libc` Syscall cases leads to overly restrictive sandboxing.



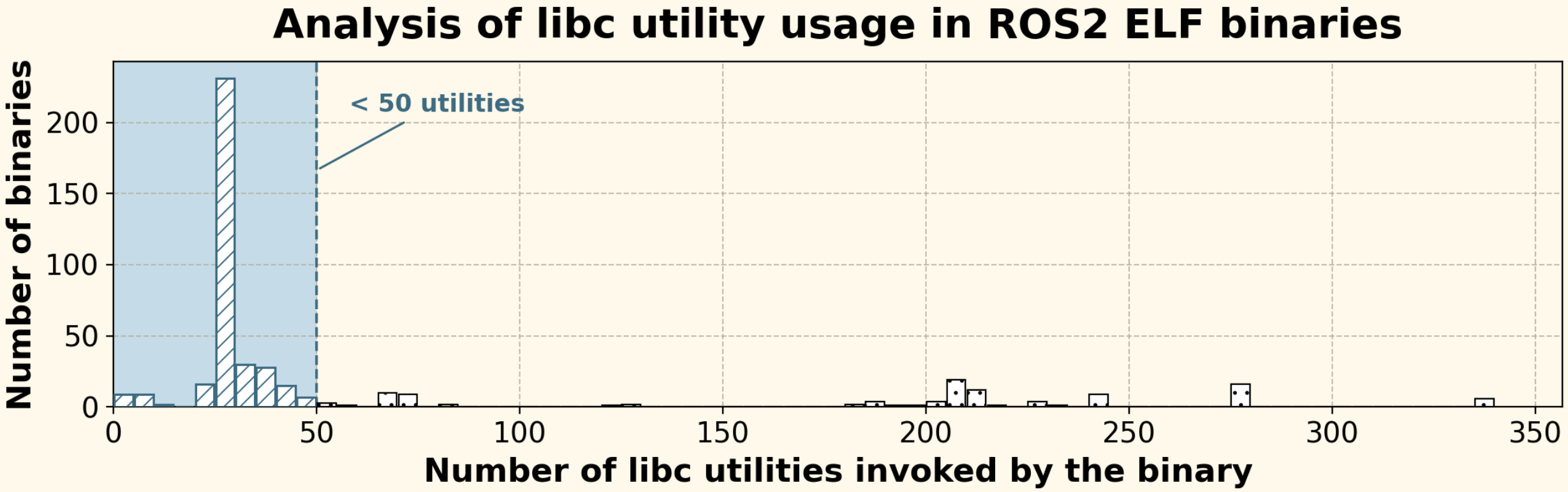
Triage of `libc` Utility Usage

Analyzed 456 ROS2 binaries to determine usage of the `libc` API.



Triage of `libc` Utility Usage

Analyzed 456 ROS2 binaries to determine usage of the `libc` API.



Leveraging Results of `libc` Utility Usage

Found that ROS2 apps use a median of just 27 `libc` calls.

Number of libc utilities invoked by the binary			
Average:	**67.22**	25th-percentile:	**26**
Std. dev.:	**81.75**	75th-percentile:	**47**
Median:	**27**	Max/Min:	**336/0**

Leveraging Results of `libc` Utility Usage

Idea: Special-case these `libc` calls via a library wrappers. Wrappers can enforce app specific policies on resource access.

Number of libc utilities invoked by the binary

Average:	**67.22**	25th-percentile:	**26**
Std. dev.:	**81.75**	75th-percentile:	**47**
Median:	**27**	Max/Min:	**336/0**

Leveraging Results of `libc` Utility Usage

Idea: Special-case these `libc` calls via a library wrappers. Wrappers can enforce app specific policies on resource access.

ESROS2-SCGate

- Implementation of `libc` wrappers.
- Wrappers are made part of TCB, because they implement policy checks

ESROS2-SCGate

Example of ESROS2-SCGate for fopen

```c
typedef int (*FdPolFn)(int fd, const char *mode);

FILE *fopen_wrapper(const char *path, const char *mode) {
    int fd = open(path, flags);
    if (fd < 0) {
        return NULL;
    }

    FdPolFn policy = get_policy_function();
    if (!policy || !policy(fd, mode)) {
        close(fd);
        return NULL;
    }

    FILE *fp = fdopen(fd, mode);
    return fp;
}
```

ESROS2-SCGate

Example of ESROS2-SCGate for fopen

```c
typedef int (*FdPolFn)(int fd, const char *mode);

FILE *fopen_wrapper(const char *path, const char *mode) {
    int fd = open(path, flags);
    if (fd < 0) {
        return NULL;
    }

    FdPolFn policy = get_policy_function();
    if (!policy || !policy(fd, mode)) {
        close(fd);
        return NULL;
    }

    FILE *fp = fdopen(fd, mode);
    return fp;
}
```

ESROS2-SCGate

Example of ESROS2-SCGate for fopen

```
1  typedef int (*FdPolFn)(int fd, const char *mode);
2
3  FILE *fopen_wrapper(const char *path, const char *mode)
4      int fd = open(path, flags);
5      if (fd < 0) {
6          return NULL;
7      }

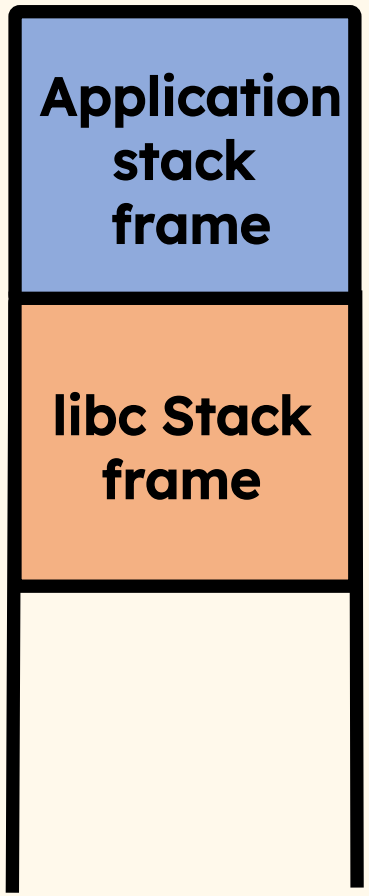
8      FdPolFn policy = get_policy_function();
9      if (!policy || !policy(fd, mode)) {
10         close(fd);
11         return NULL;
12     }

       FILE *fp = fdopen(fd, mode);
       return fp;
13 }
```

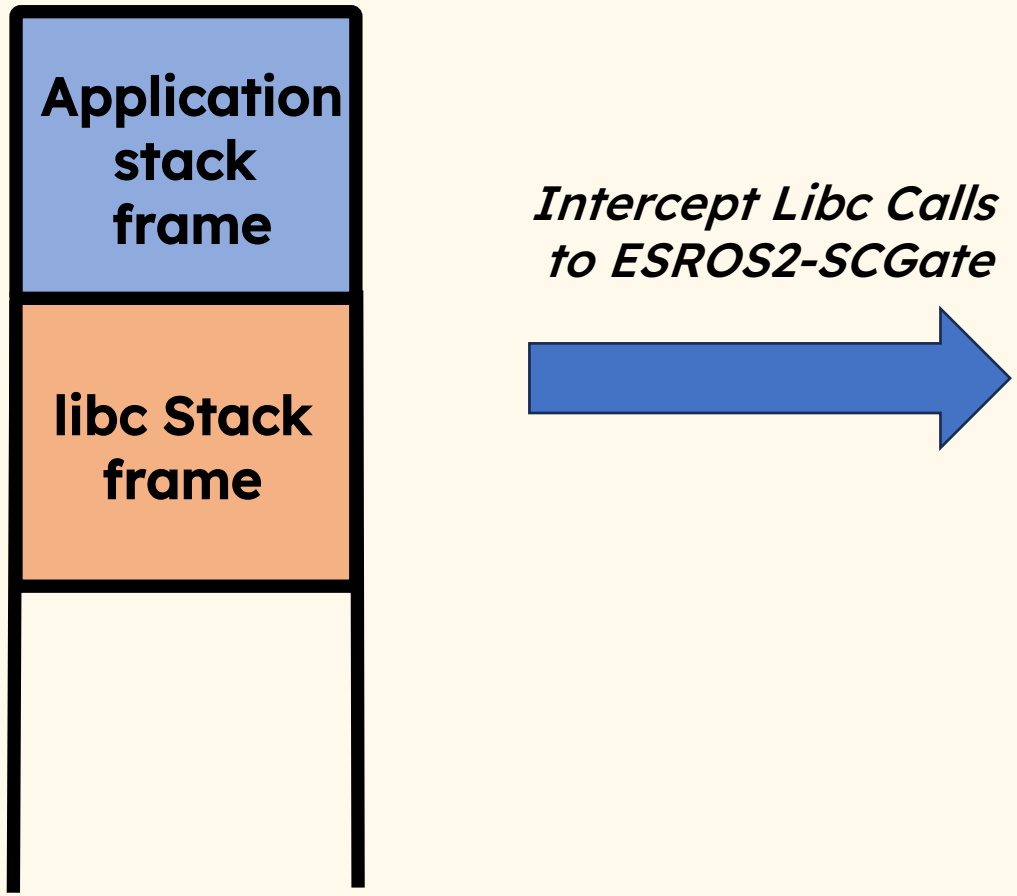
Policy

```
if (filename == "FileA" || filename == "FileB")
    ALLOW;
else
    DENY;
```

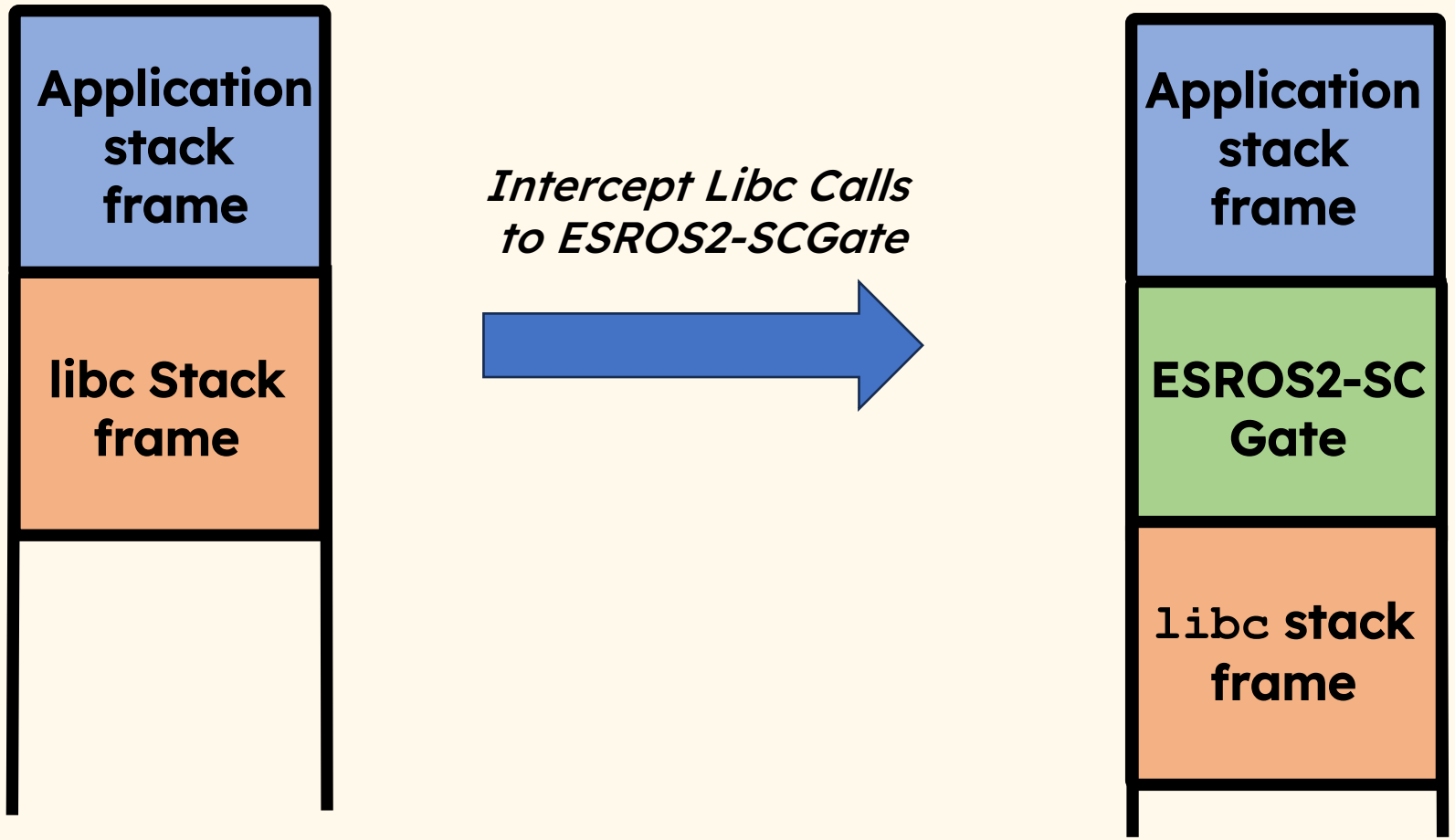
ESROS2-SCGate



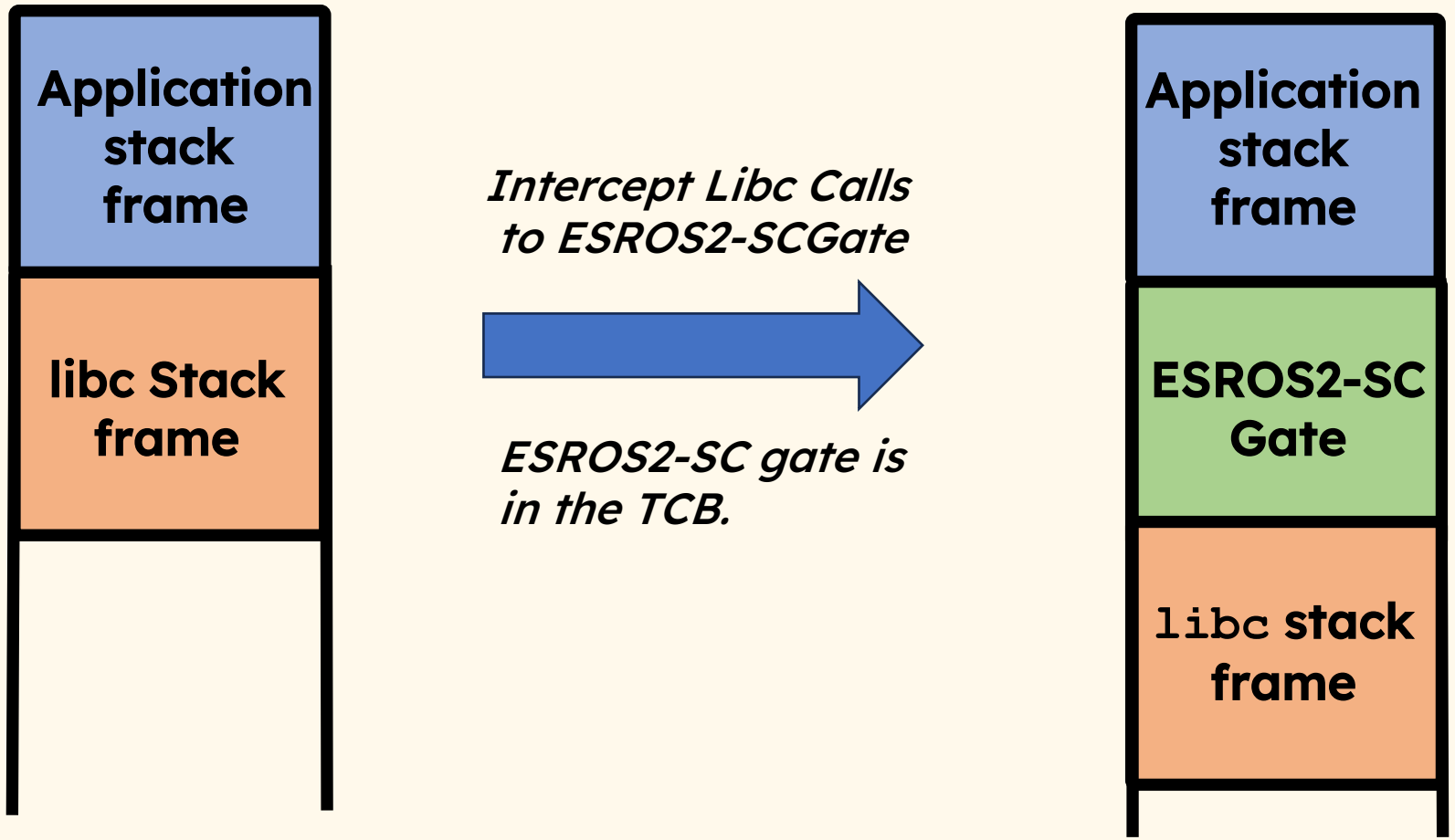
ESROS2-SCGate



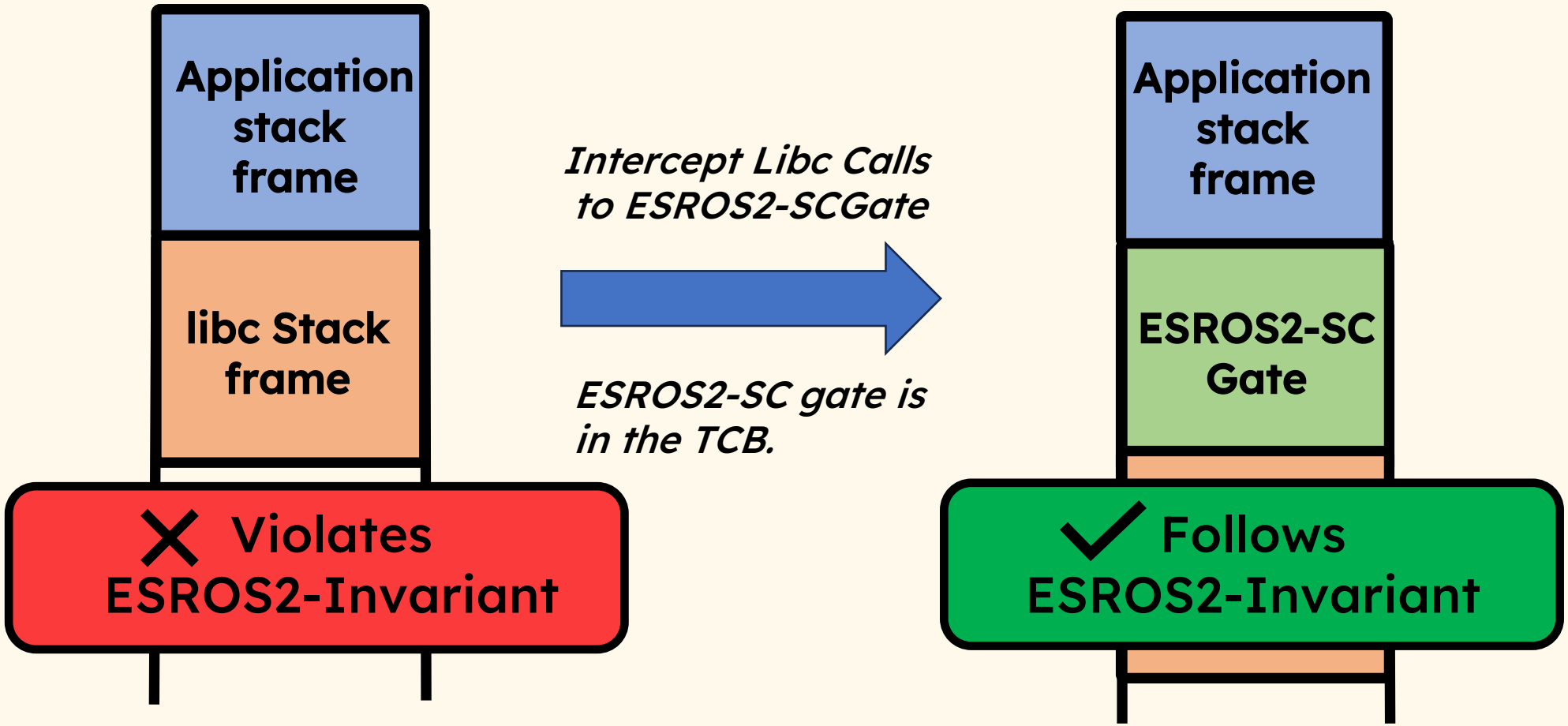
ESROS2-SCGate



ESROS2-SCGate



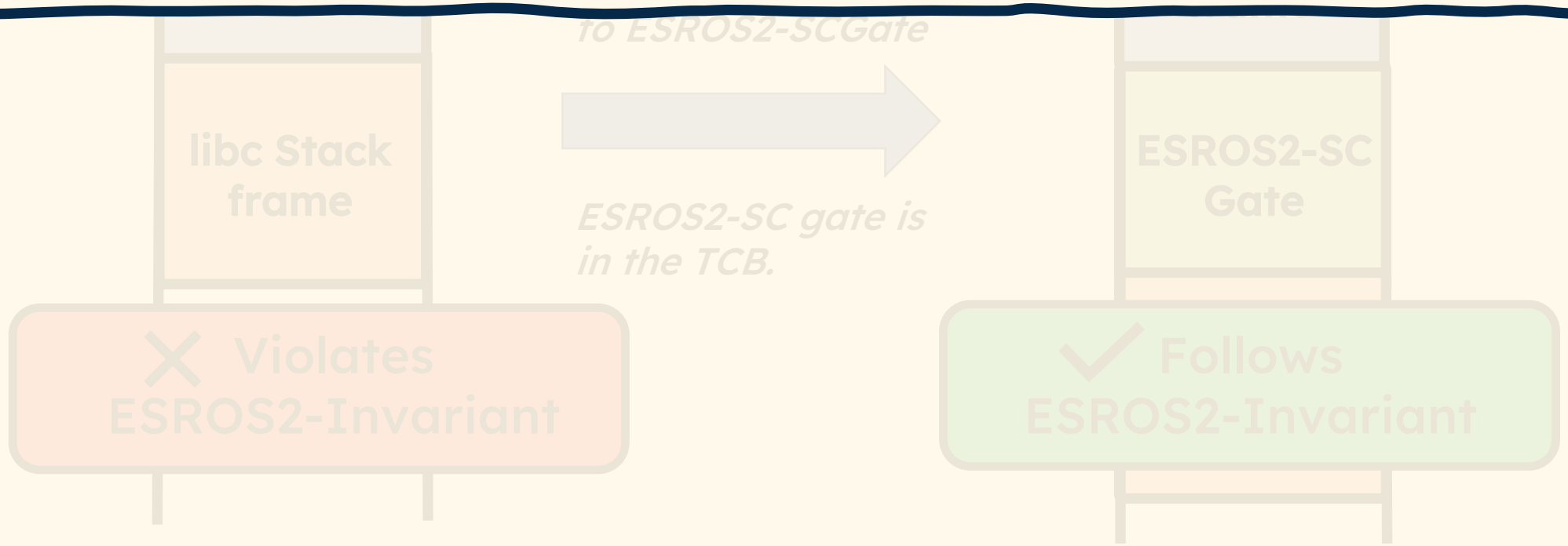
ESROS2-SCGate



ESROS2-SCGate

ESROS2-SCGate

Replaced `libc` GOT/PLT entries with ESROS2-SCGates.



ESROS2-SCGate

Replaced `libc` GOT/PLT entries with ESROS2-SCGates.

Works with off-the-shelf binaries.

ESROS2-SCGate

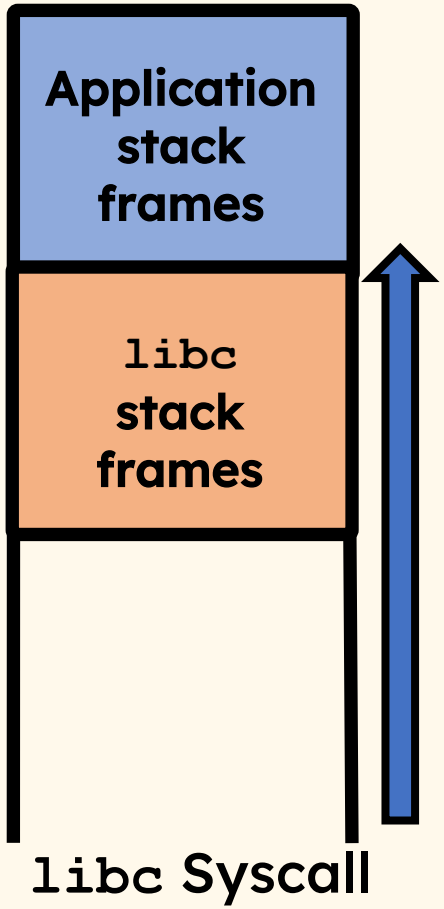
Replaced `libc` GOT/PLT entries with ESROS2-SCGates.

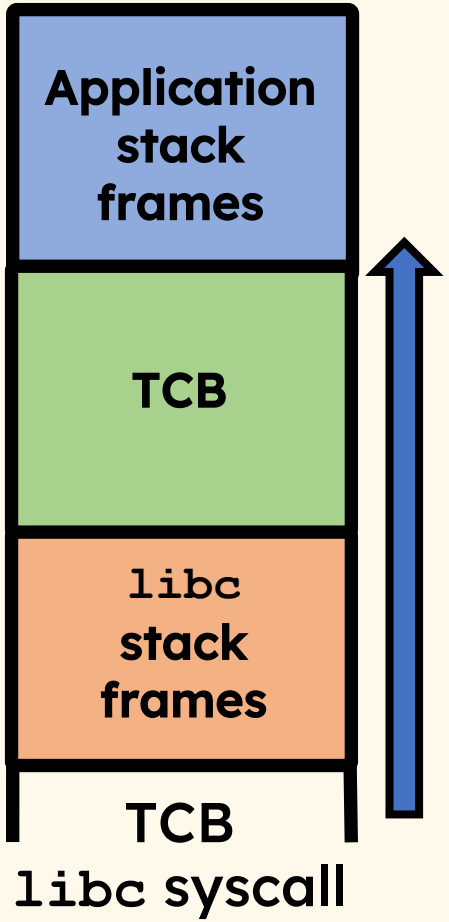
Works with off-the-shelf binaries.

Handles statically linked binaries; their security analysis is in the paper.

Enforcing ESROS2 Invariant (Strawman)

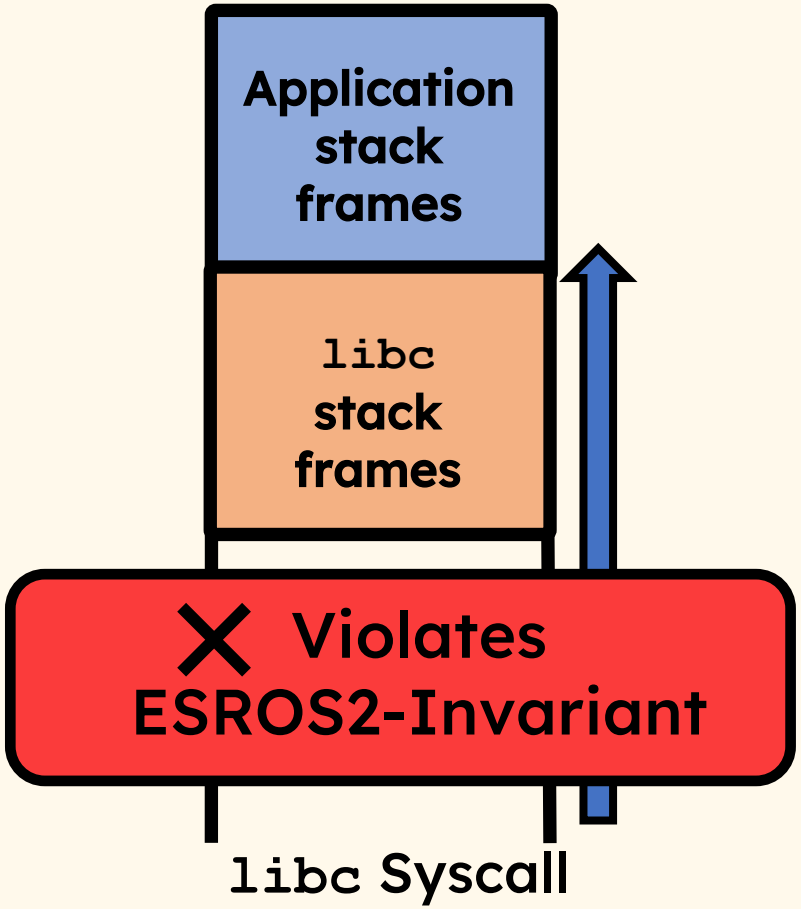
Unwind the stack at each syscall and check for an TCB frame.

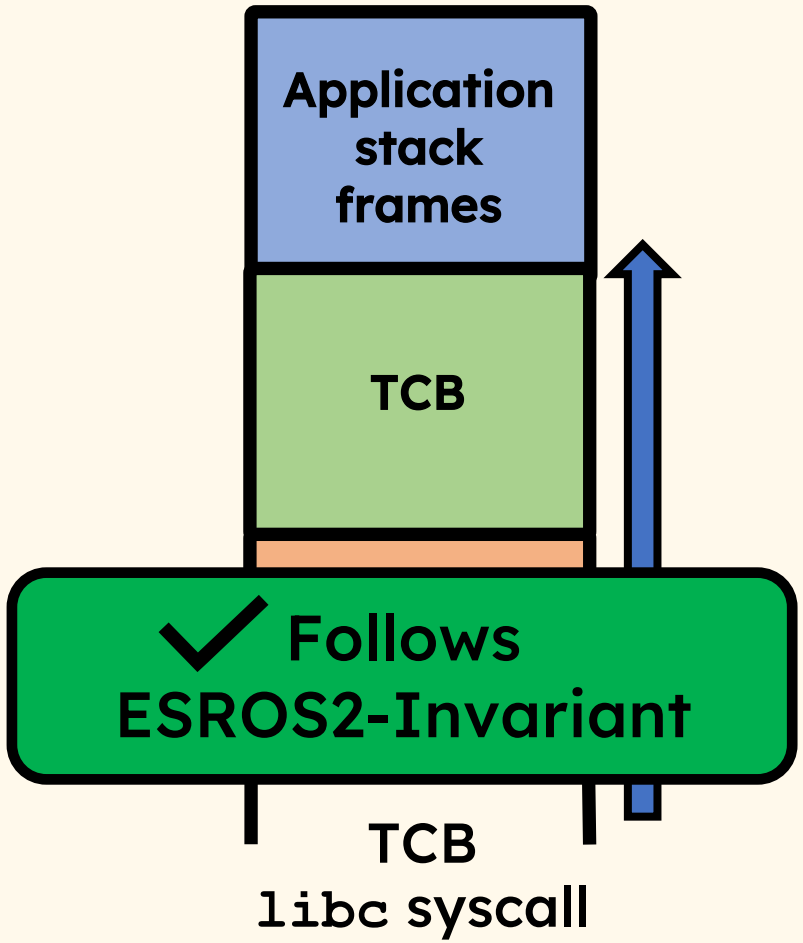




Enforcing ESROS2 Invariant (Strawman)

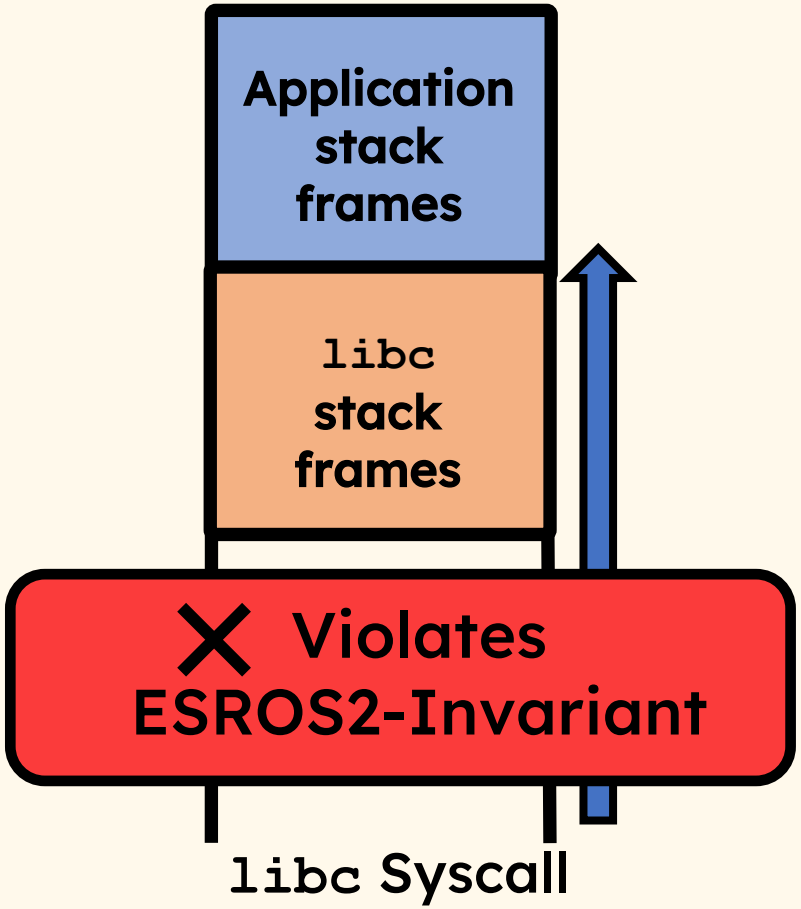
Unwind the stack at each syscall and check for an TCB frame.

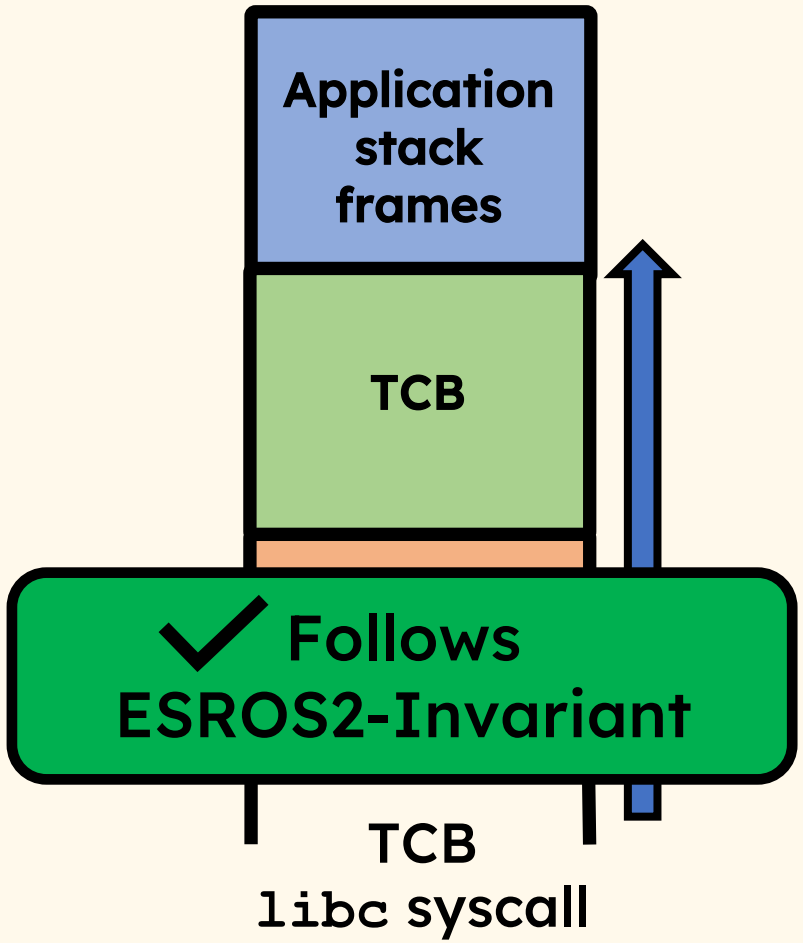




Problem With Strawman Approach

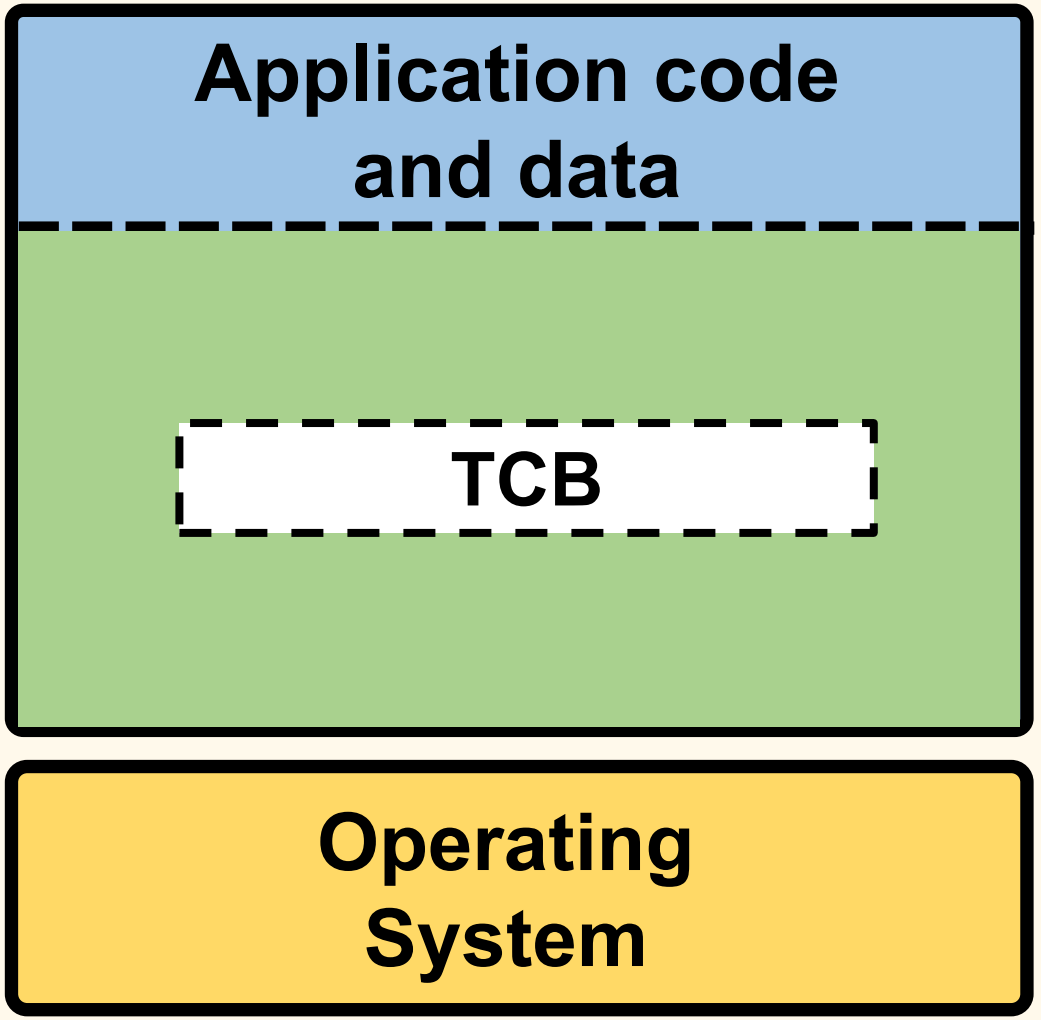
eBPF prototype: 100–150% runtime and ~325 MB memory overhead.





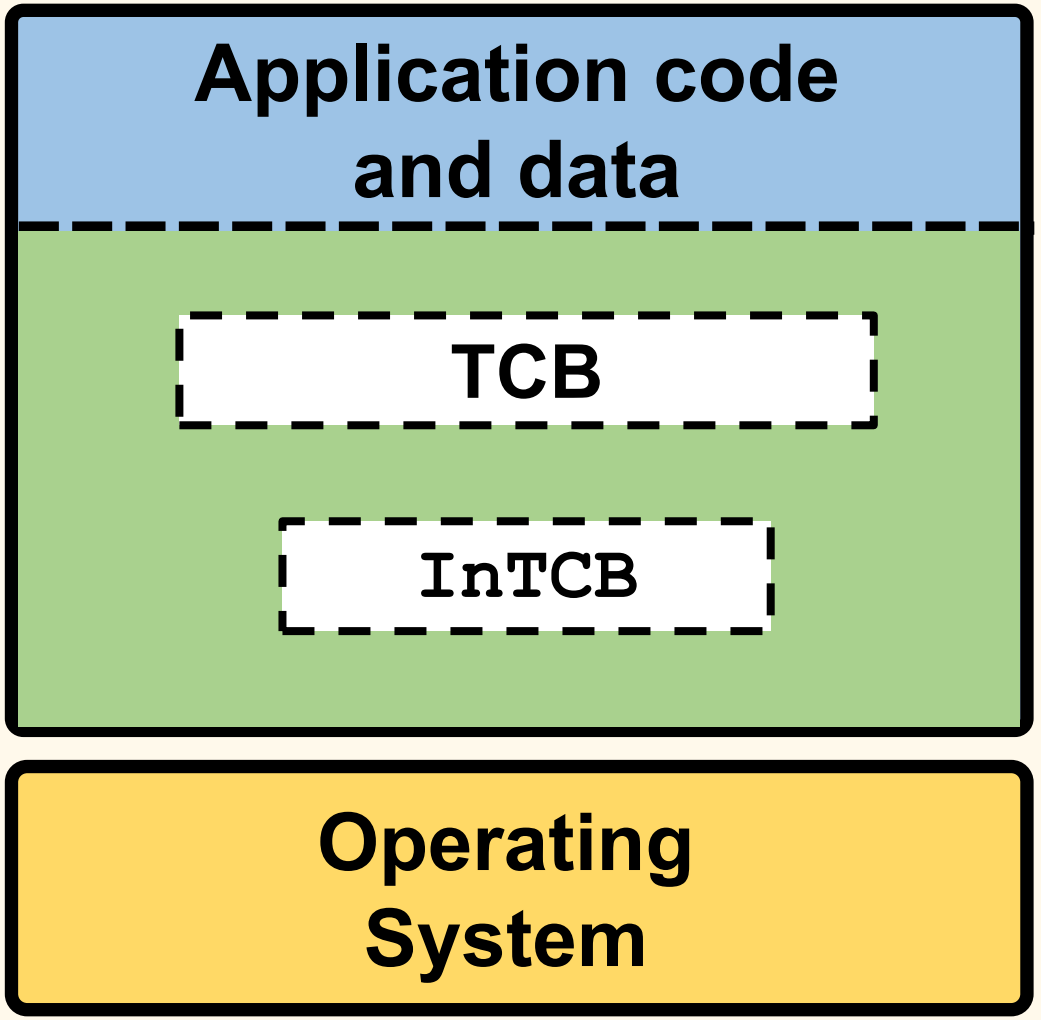
Observation: Track Stack State in User Space

Track stack state in user space to check entry into TCB.



Observation: Track Stack State in User Space

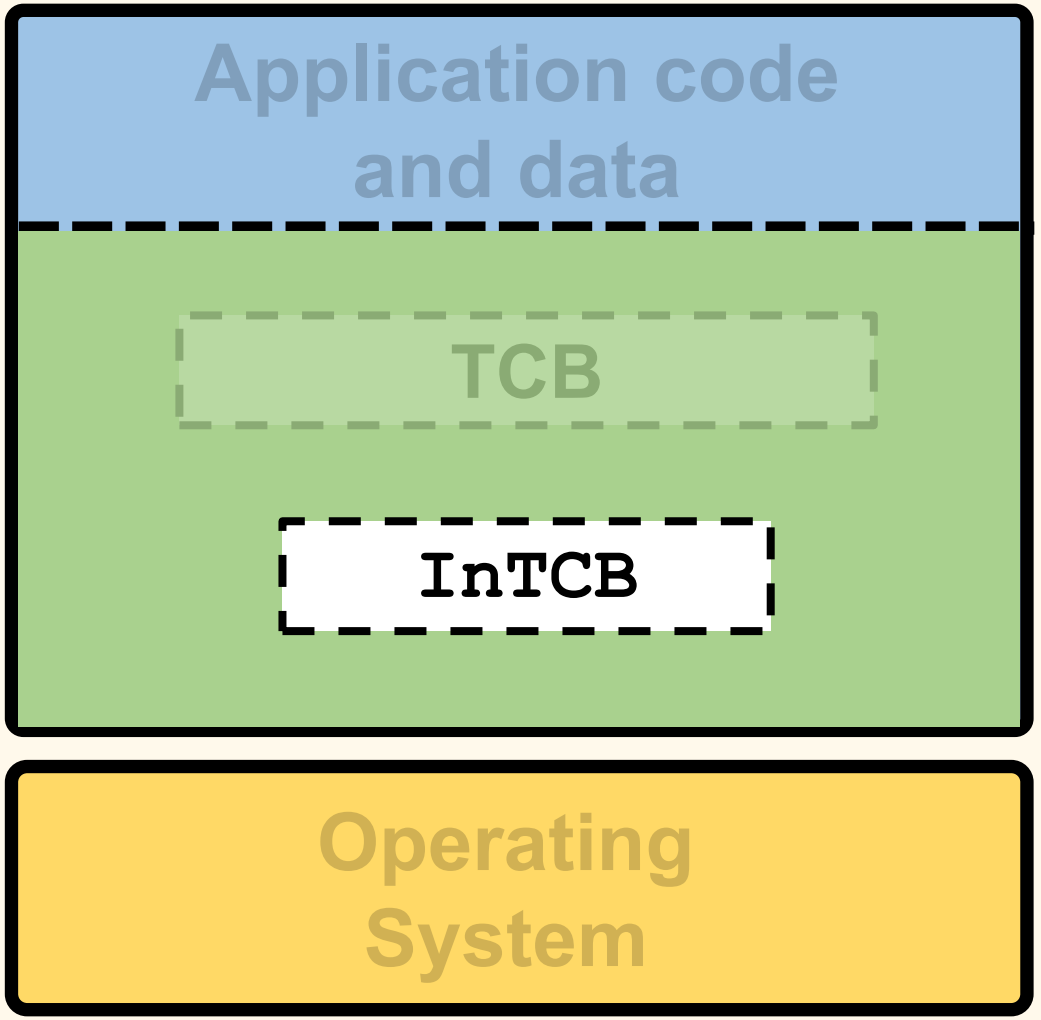
- Track stack state in user space to check entry into TCB.
- `InTCB` variable to track execution context.



Observation: Track Stack State in User Space

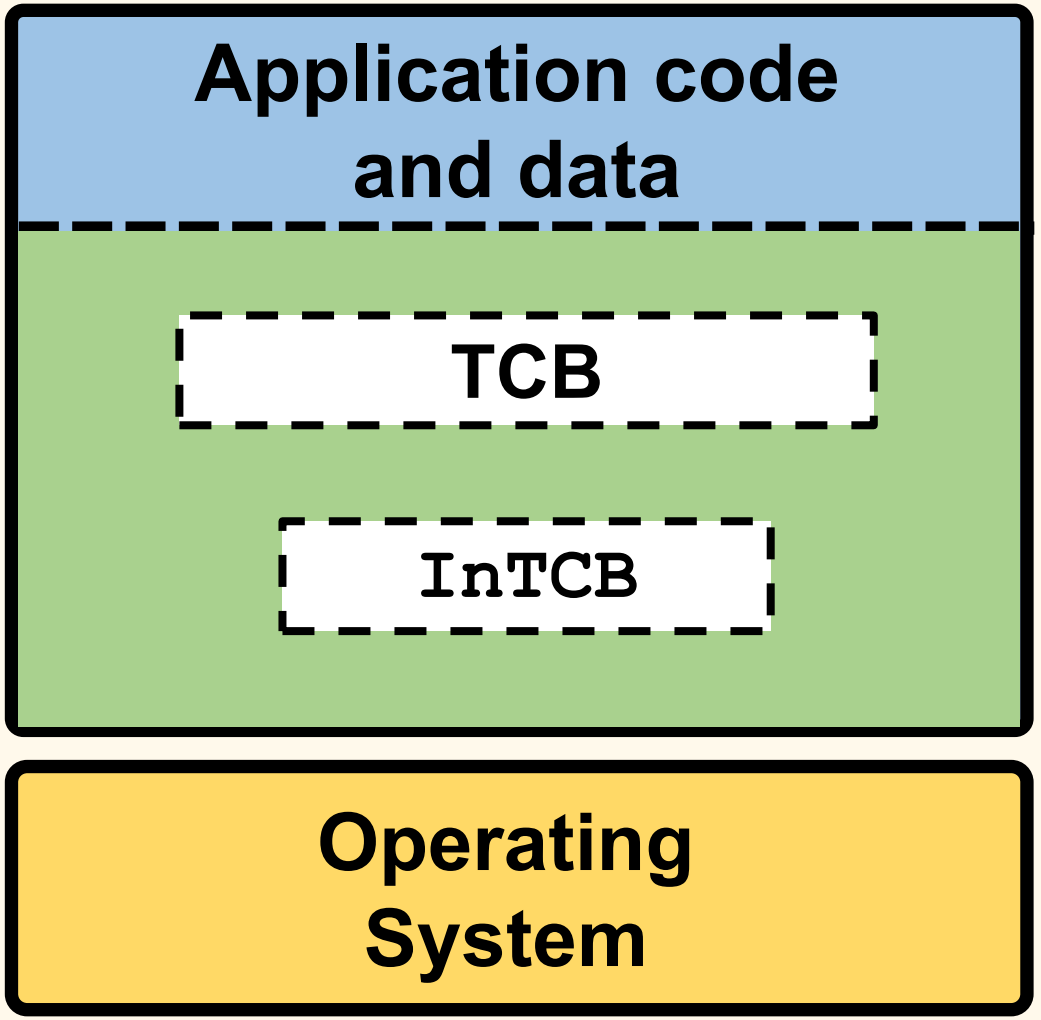
- InTCB = 0:
  Outside TCB

- InTCB = 1 :
  Inside TCB



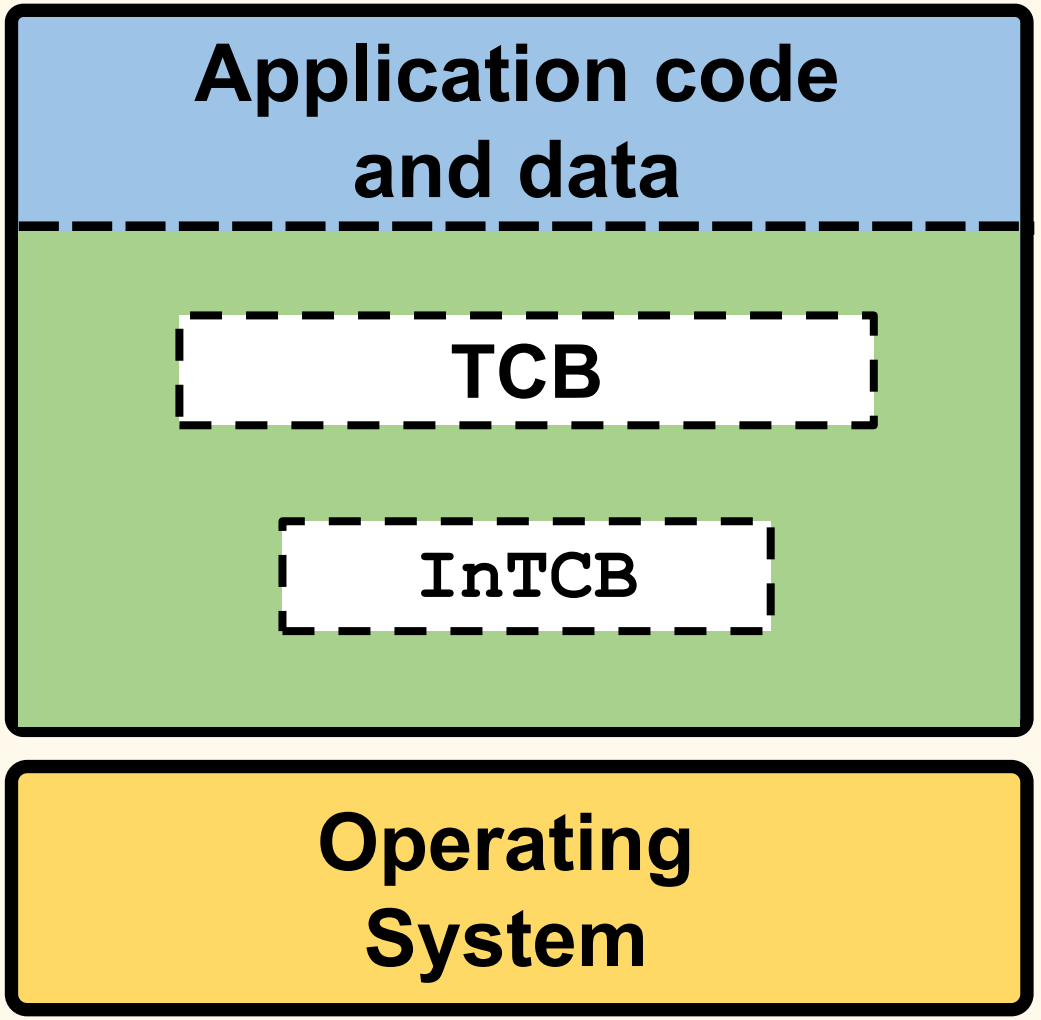
Observation: Track Stack State in User Space

User-space `InTCB` variable is not secure.



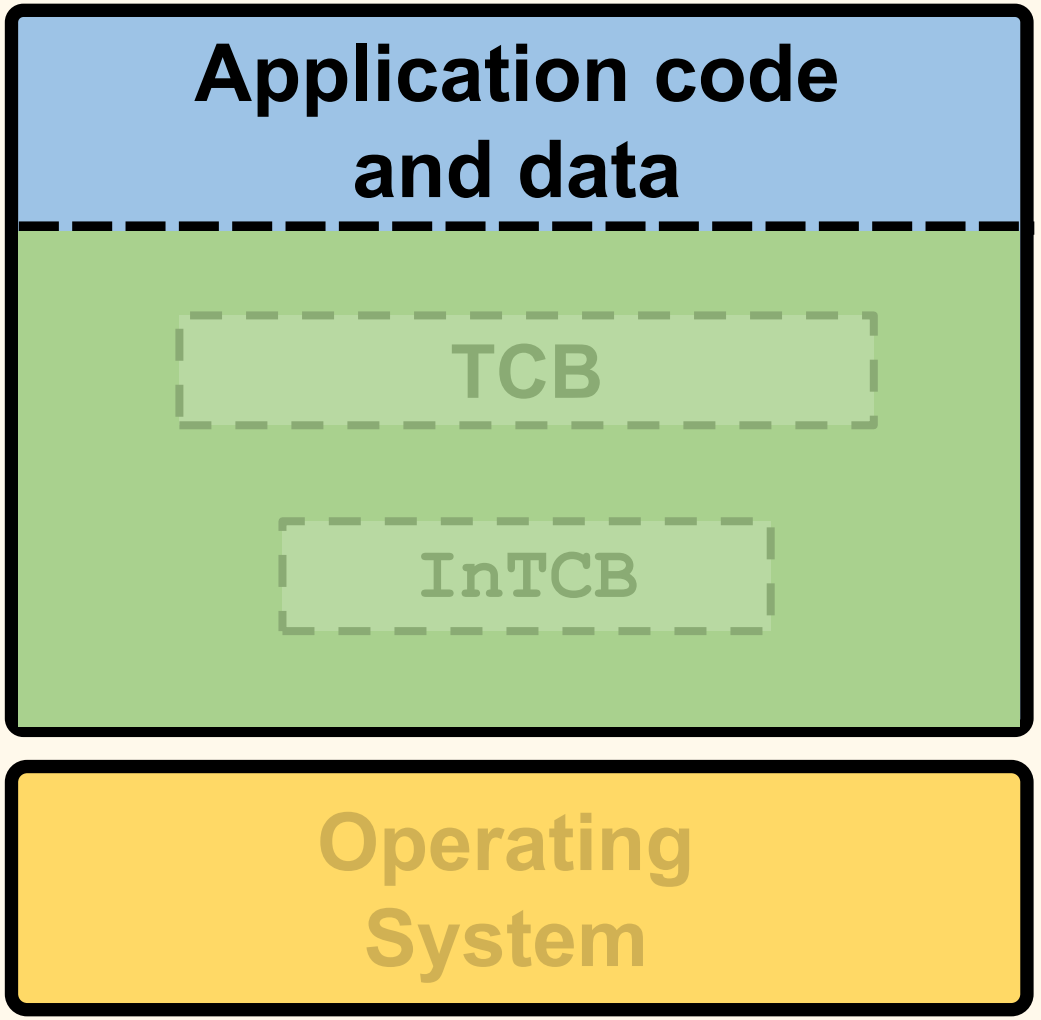
Enforcing ESROS2-Invariant

Divide the app address space using Intel MPK.



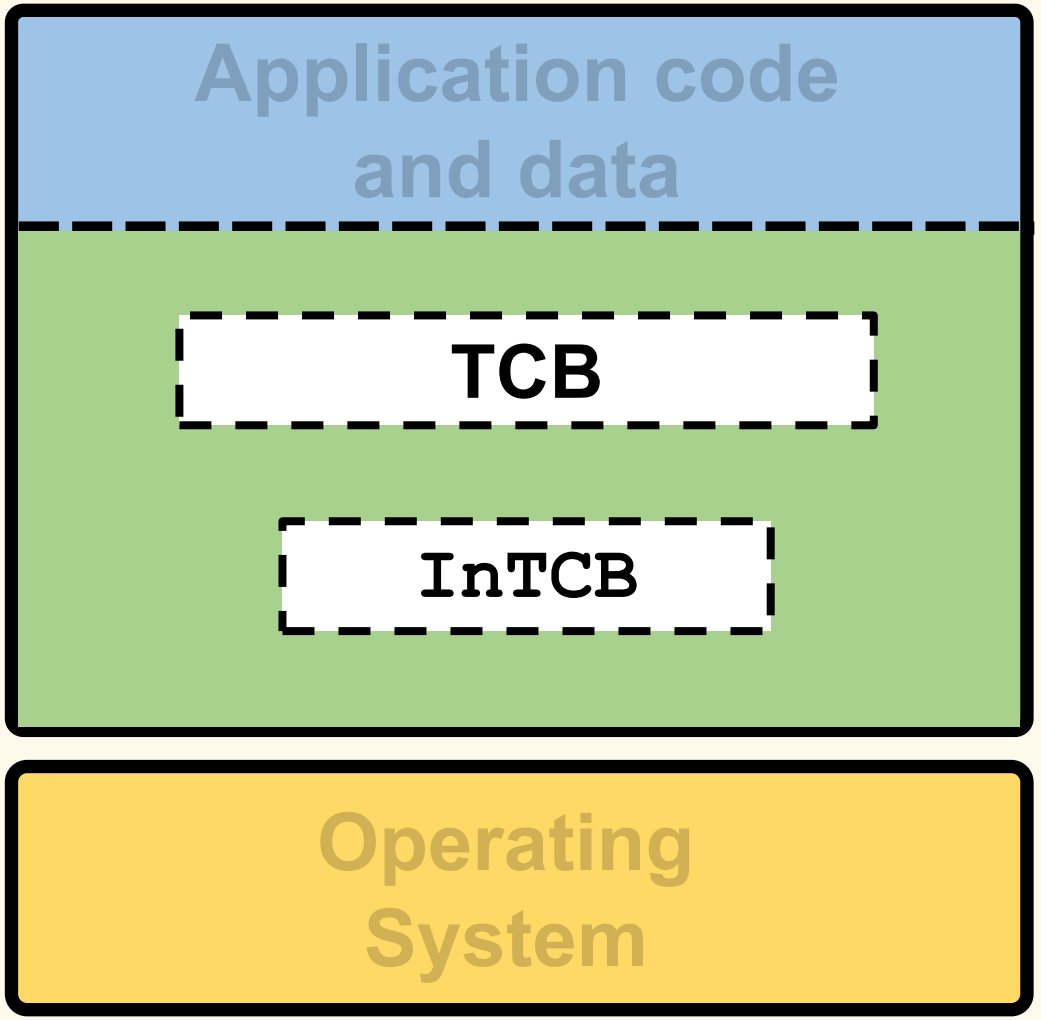
Enforcing ESROS2-Invariant

- Divide the app address space using Intel MPK.
- **A-DOM**: Application Domain



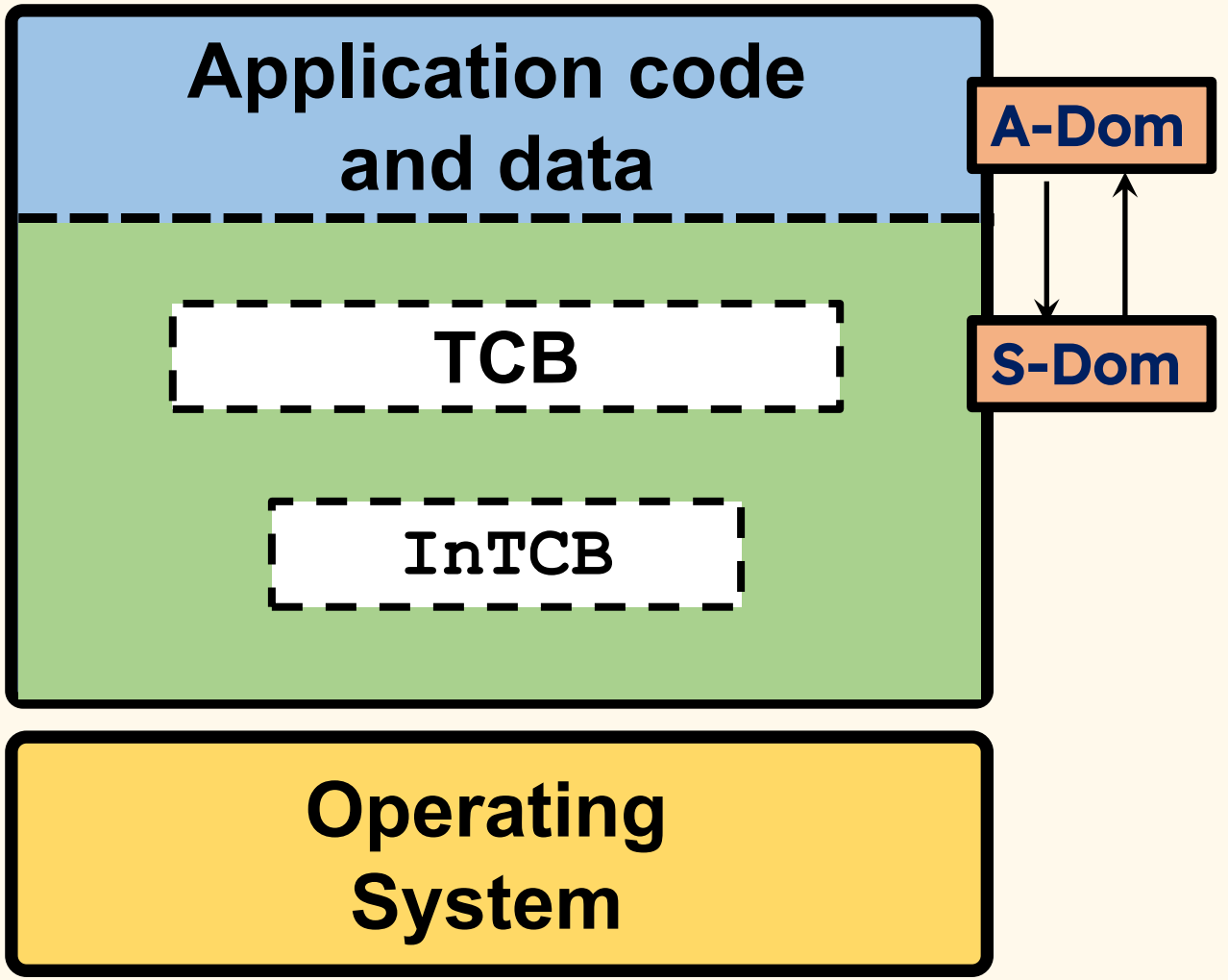
Enforcing ESROS2-Invariant

- Divide the app address space using Intel MPK.
- **A-DOM**: Application Domain
- **S-DOM**: Secure Domain



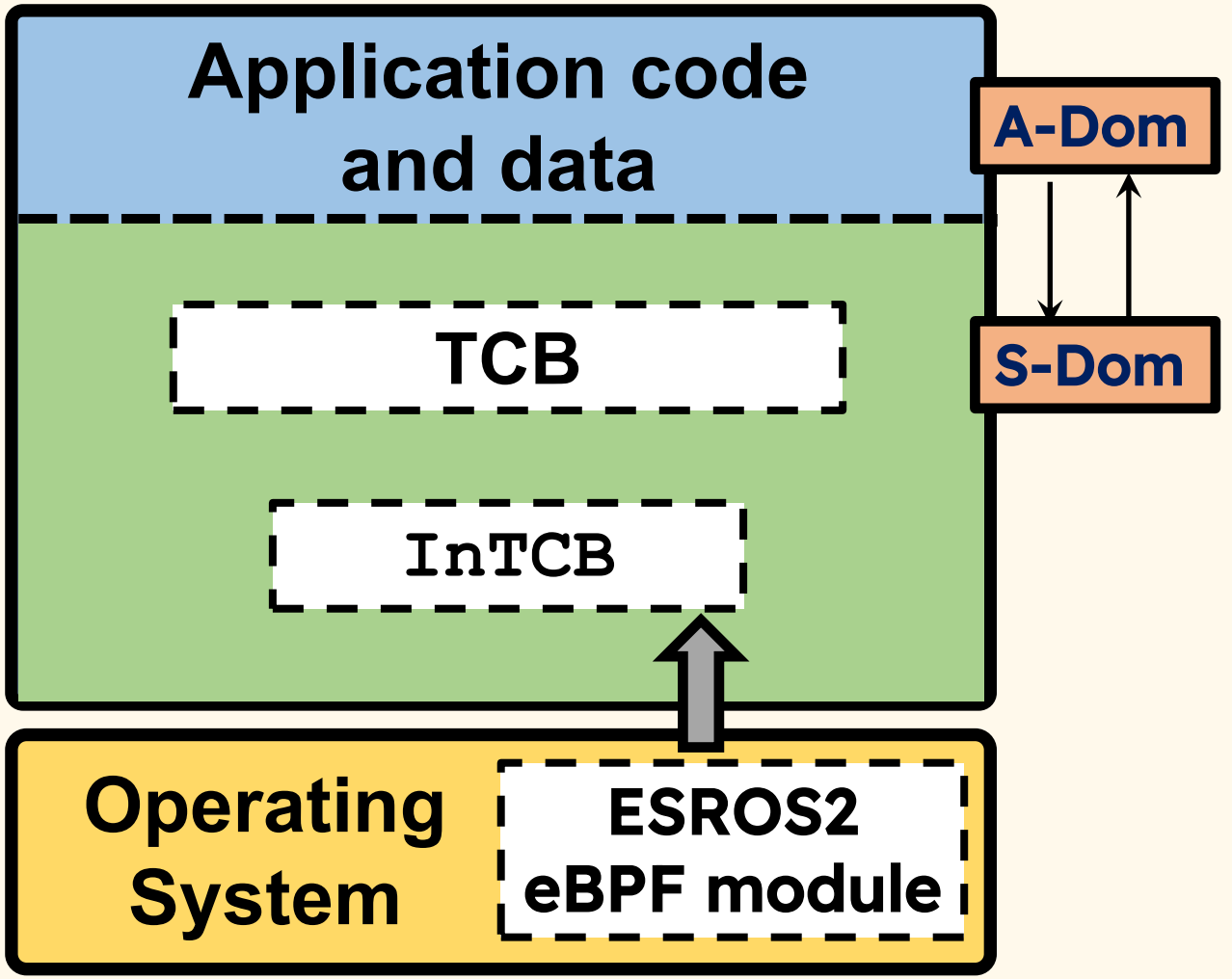
Enforcing ESROS2-Invariant

- Divide the app address space using Intel MPK.
- **A-DOM**: Application Domain
- **S-DOM**: Secure Domain



Enforcing ESROS2-Invariant

ESROS2 eBPF module checks `InTCB` to identify the app's stack domain.



Case Study : Robotic Arm

- Used a simulated Franka Emika robotic arm.
- Main goal of the case study: Evaluate runtime overhead and policy generation practicality.



Franka Emika Panda robotic arm

Case Study : Robotic Arm

Four binaries are required to run the case study.

ROS2 Binaries for Robotic Arm Case Study
move_group
robot_state_publisher
ros2_control_node
static_transform_publisher

Case Study : Robotic Arm

`libc` Utilities : Determine the unique `libc` functions invoked by each application

Robotic Arm	
ROS2 Binary	**libc Utilities**
move_group	16
robot_state_publisher	5
ros2_control_node	9
static_transform_publisher	2

Case Study : Robotic Arm

Example: Move_Group has 16 unique libc utilities, requiring 16 policies.

Robotic Arm	
ROS2 Binary	**libc Utilities**
move_group	16
robot_state_publisher	5
ros2_control_node	9
static_transform_publisher	2

Case Study : Robotic Arm

Example : Move Group uses socket() over Unix domain sockets to read configuration files.

Example Policy for Socket libc function

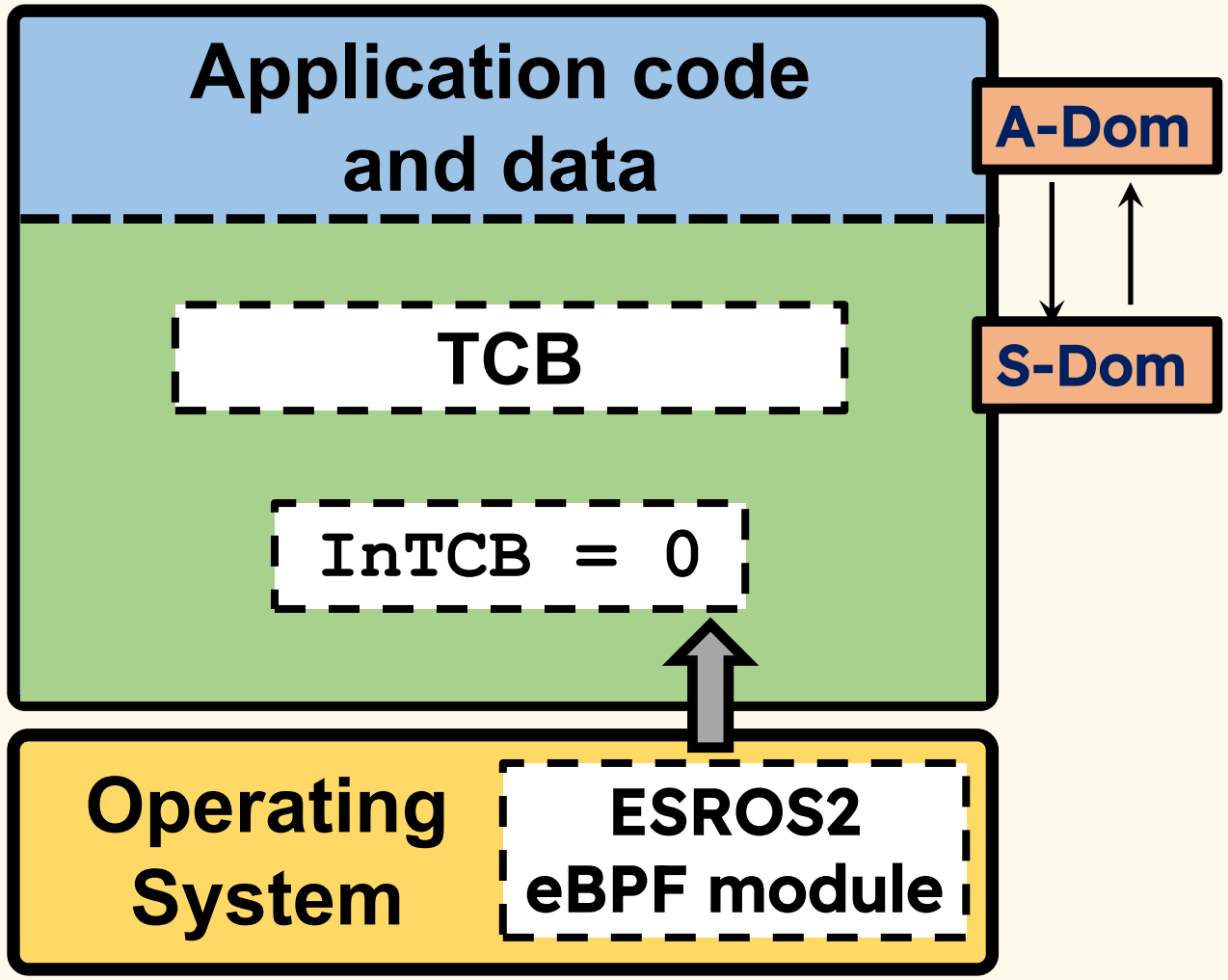
```
1  #include <sys/socket.h>
2
3  bool socket_policy(int domain, int type, int protocol) {
4      if (domain == AF_UNIX) {
5          return true;
6      }
7
8      return false;
   }
```

Case Study : Robotic Arm

- 4.16% average latency overhead observed with ESROS2 in the robotic arm case study.
- Two additional case studies and microbenchmark results are detailed in the paper.

Summary

- Identified security limitations in SROS2.
- ESROS2-Invariant Enforcement via User Space InTCB Check.
- Practical Policy Generation (Case Study Results).
- 5–7% runtime overhead across benchmarks.



Questions?





ESROS2: Application Sandboxes for Secure ROS2

Dev Tejas Gandhi, Vinod Ganapathy

Computer Systems Security Lab, Indian Institute of Science