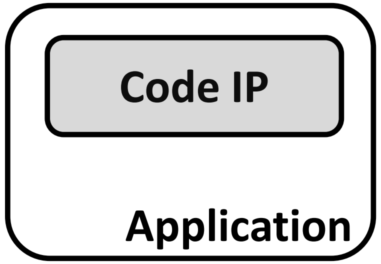
Pagoda: Towards Binary Code Privacy Protection with SGX-based Execute-Only Memory

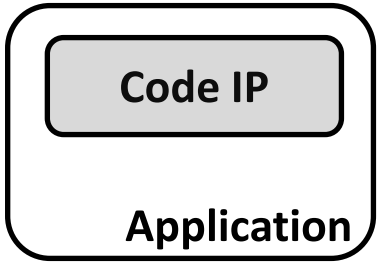
JIYONG YU[1], XINYANG GE[2], TRENT JAEGER[3], CHRIS FLETCHER[1], WEIDONG CUI[4]

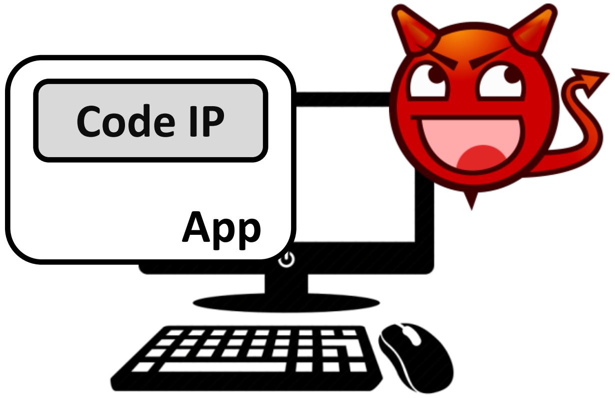
[1]UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN[1]
[2]DATABRICKS
[3]PENNSYLVANIA STATE UNIVERSITY
[4]MICROSOFT RESEARCH

Code Privacy in Untrusted Environments

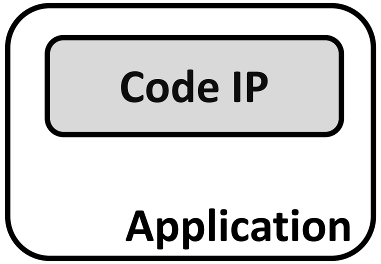


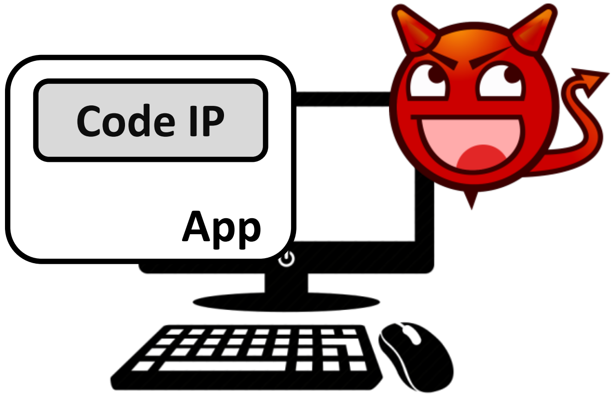
Code Privacy in Untrusted Environments

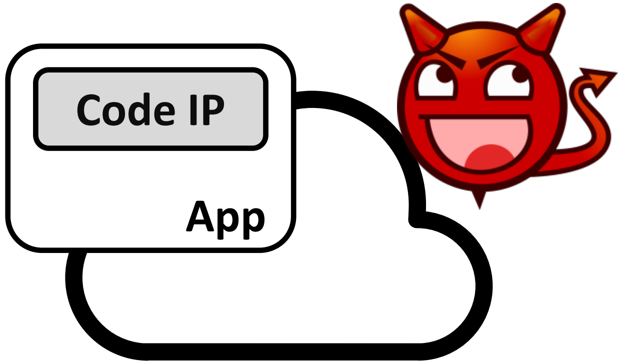




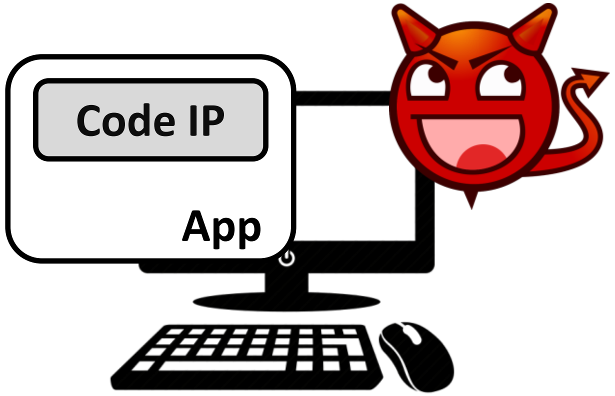
Code Privacy in Untrusted Environments

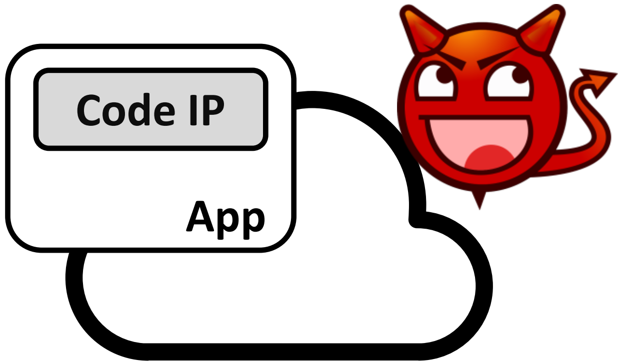


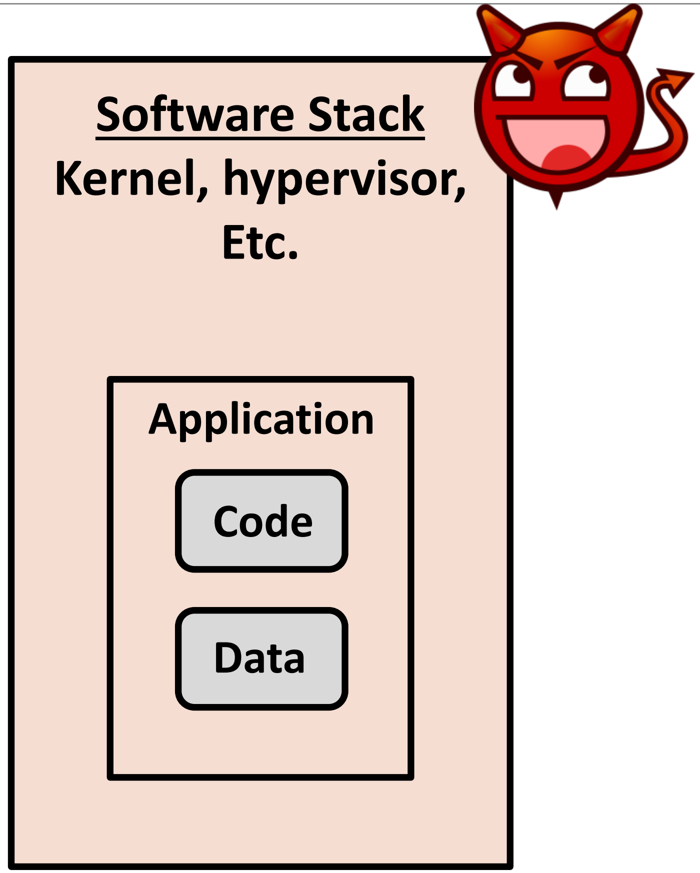




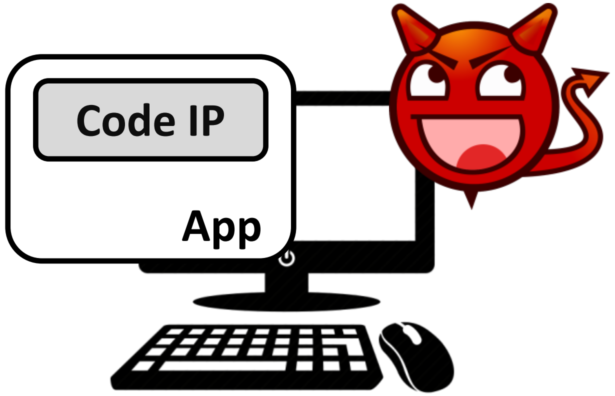
Intel SGX Comes to Rescue

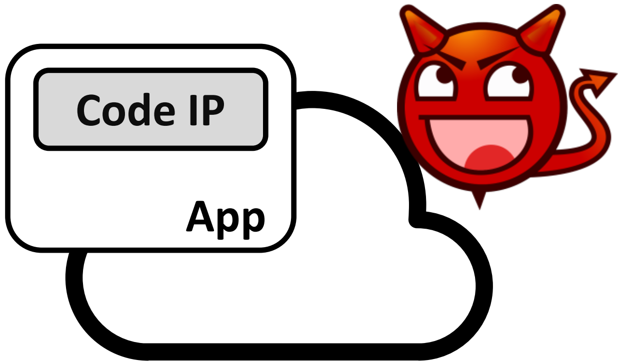


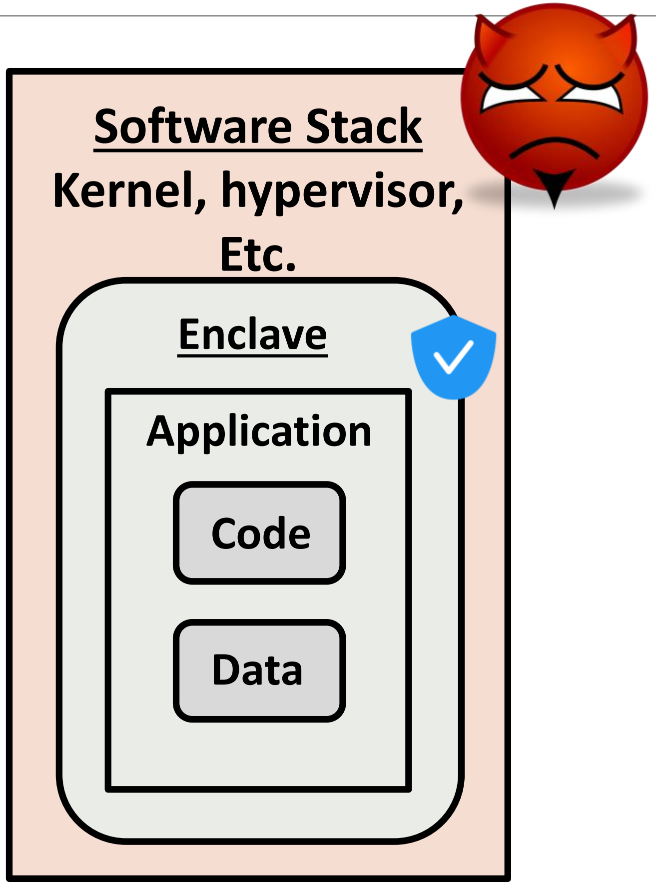




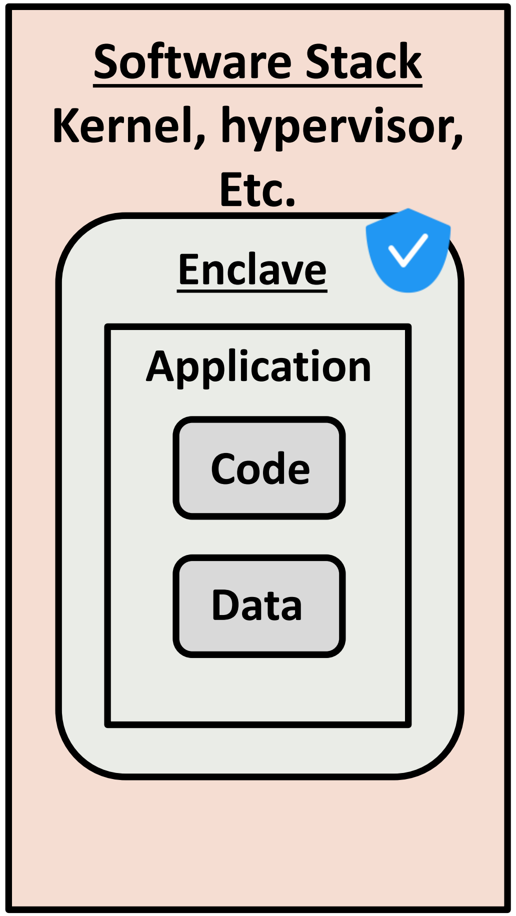
Intel SGX Comes to Rescue





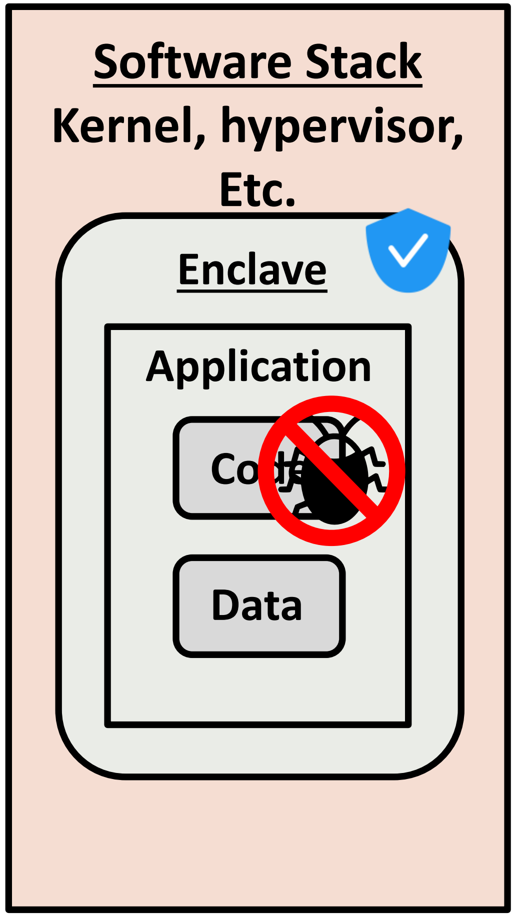


Protecting Large Applications using SGX is Challenging



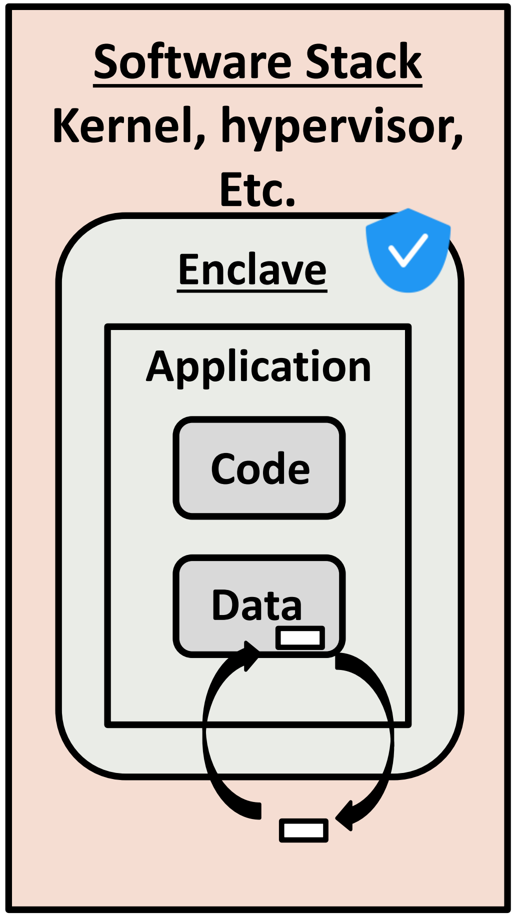
- Enclave programming model requires significant software refactoring

Protecting Large Applications using SGX is Challenging



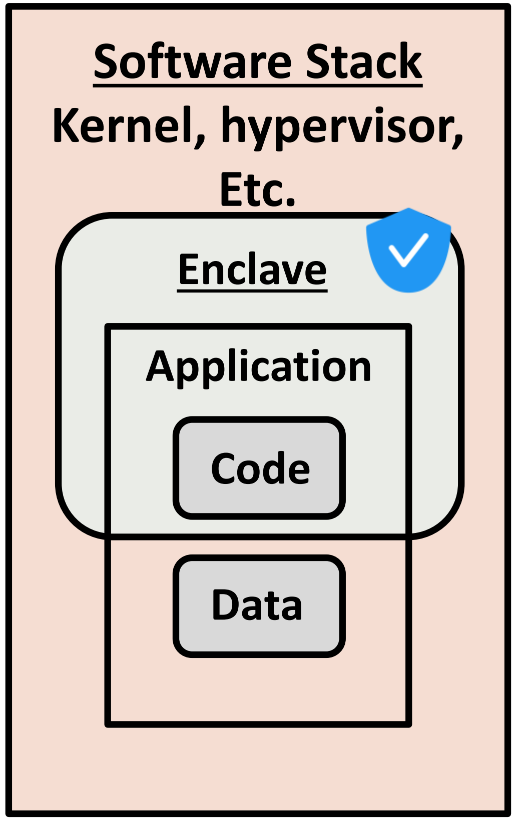
- Enclave programming model requires significant software refactoring
- Existing “plug-and-play” SGX frameworks (e.g., GrapheneSGX)
  - Assuming code is bug-free

Protecting Large Applications using SGX is Challenging



- Enclave programming model requires significant software refactoring
- Existing “plug-and-play” SGX frameworks (e.g., GrapheneSGX)
  - Assuming code is bug-free
  - Explicit data movement during system calls

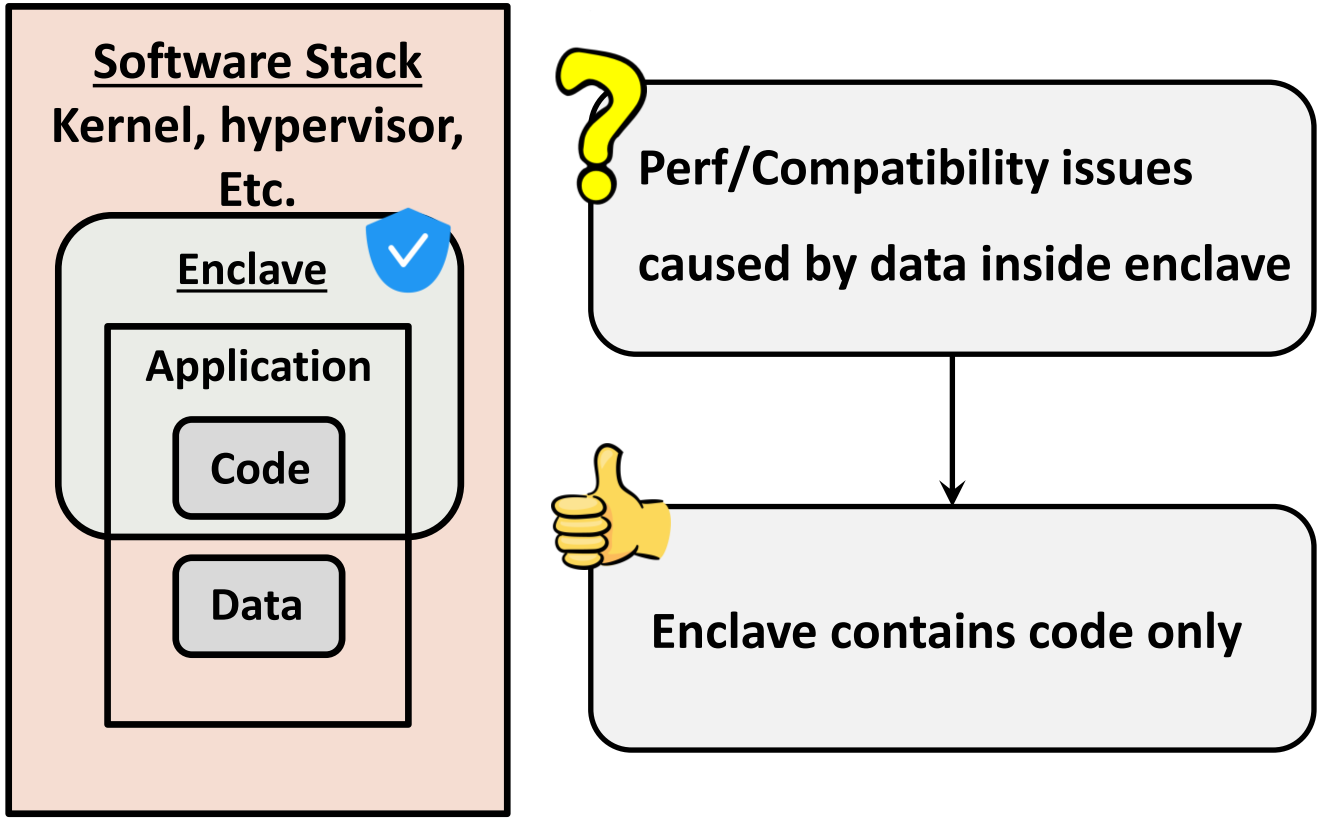
Key Idea of Pagoda



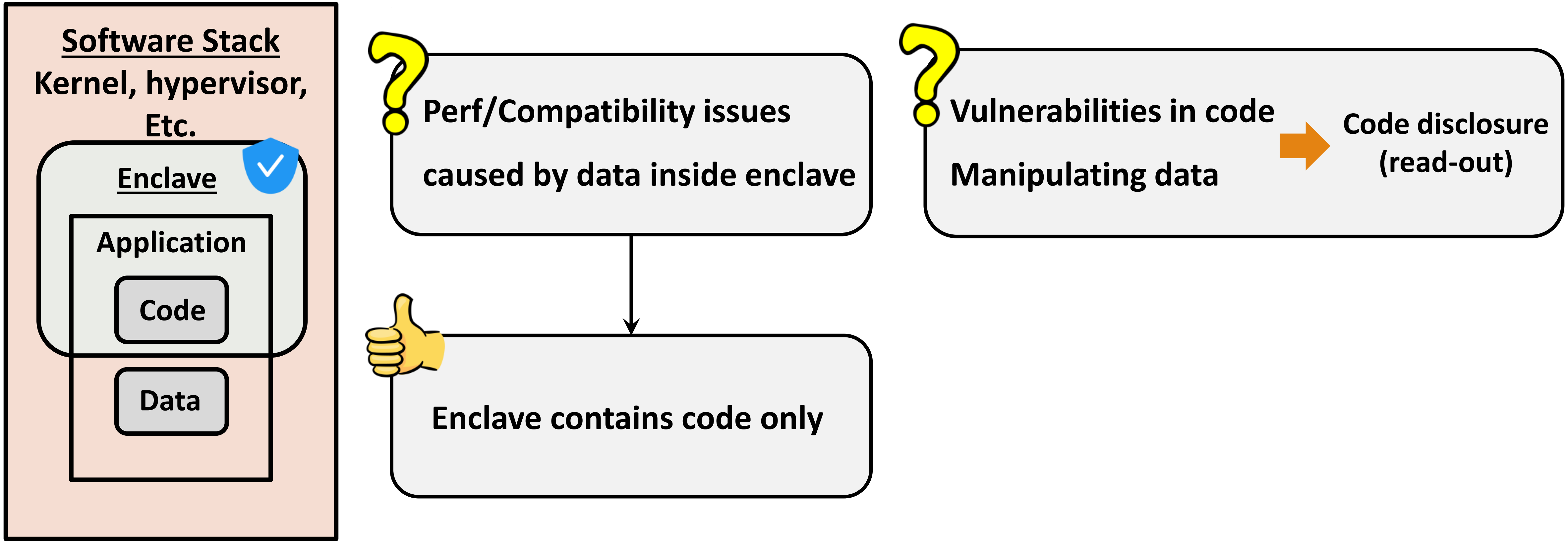
Perf/Compatibility issues

caused by data inside enclave

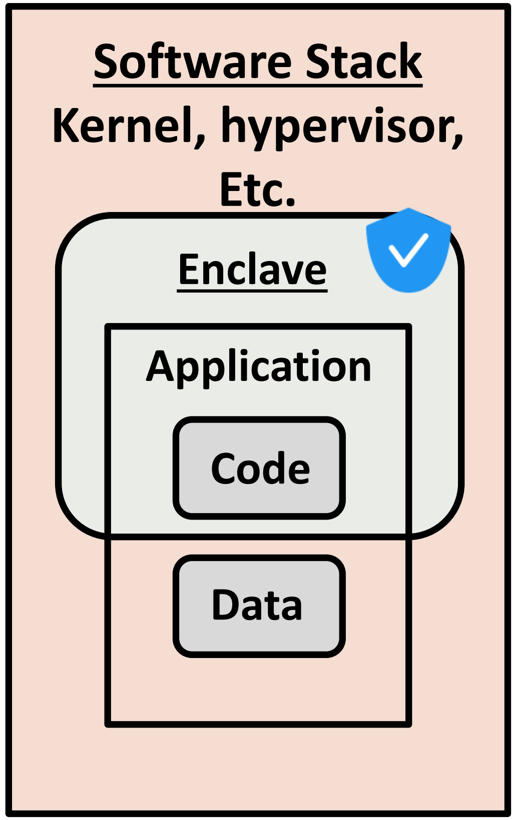
Key Idea of Pagoda



Key Idea of Pagoda



Key Idea of Pagoda



Perf/Compatibility issues caused by data inside enclave

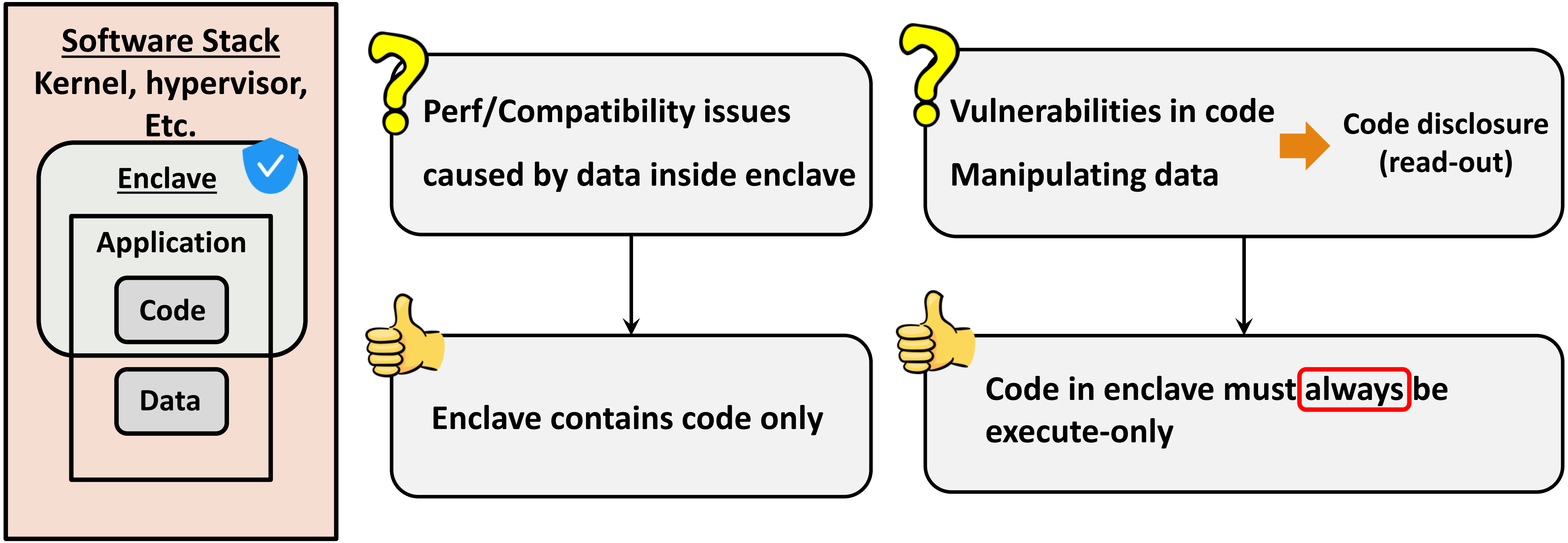
Enclave contains code only

Vulnerabilities in code Manipulating data

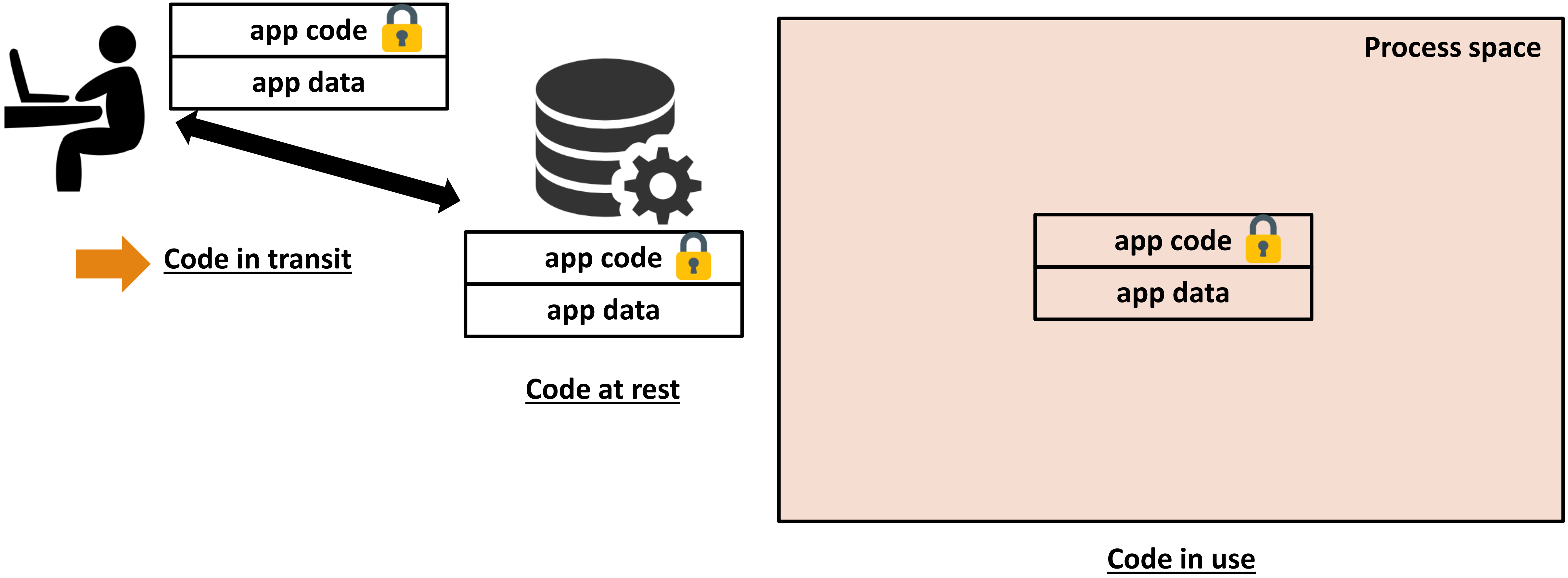
Code disclosure (read-out)

Code in enclave must always be execute-only

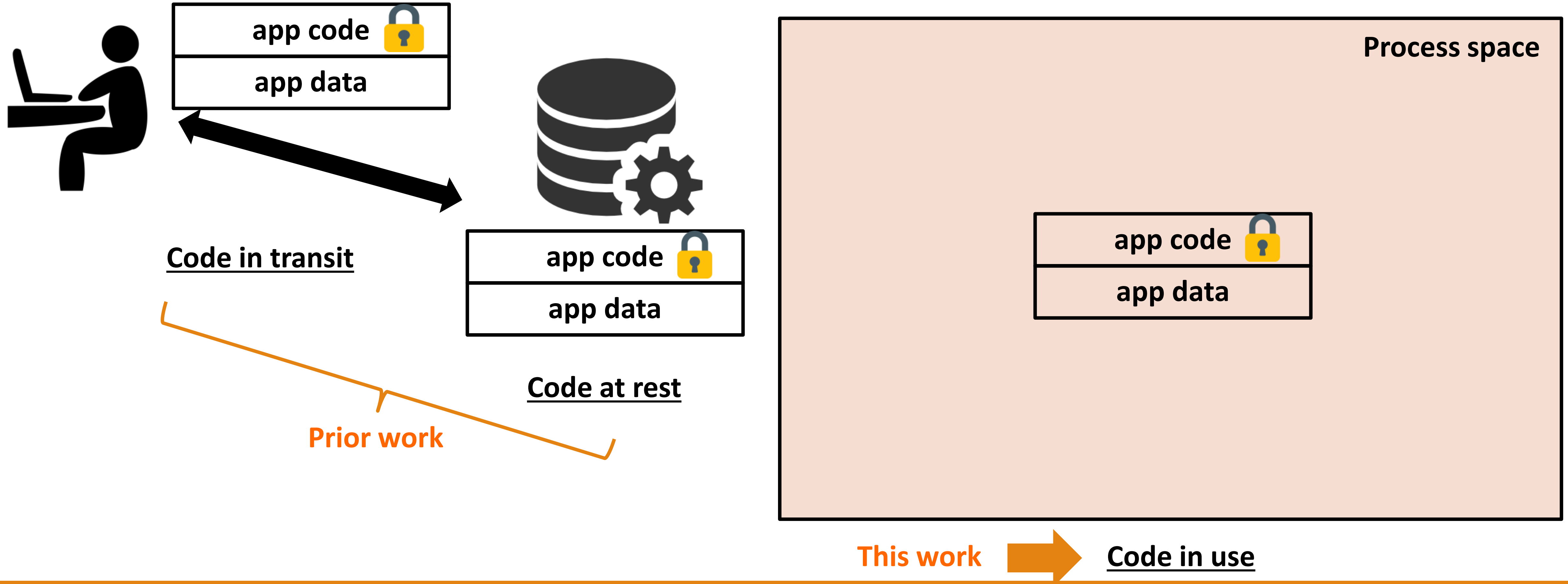
Key Idea of Pagoda



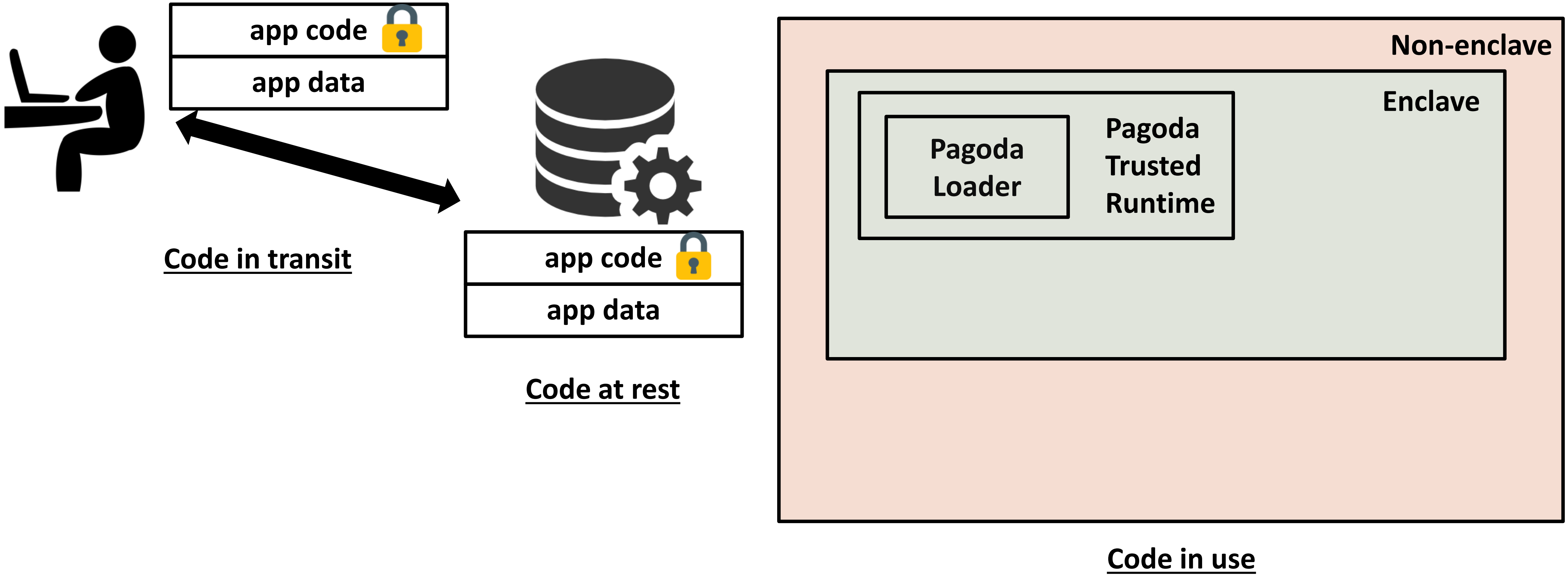
Deploying a Protected Software



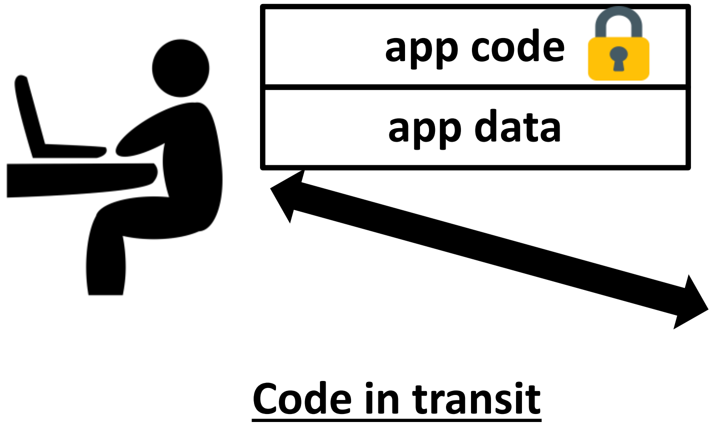
Deploying a Protected Software

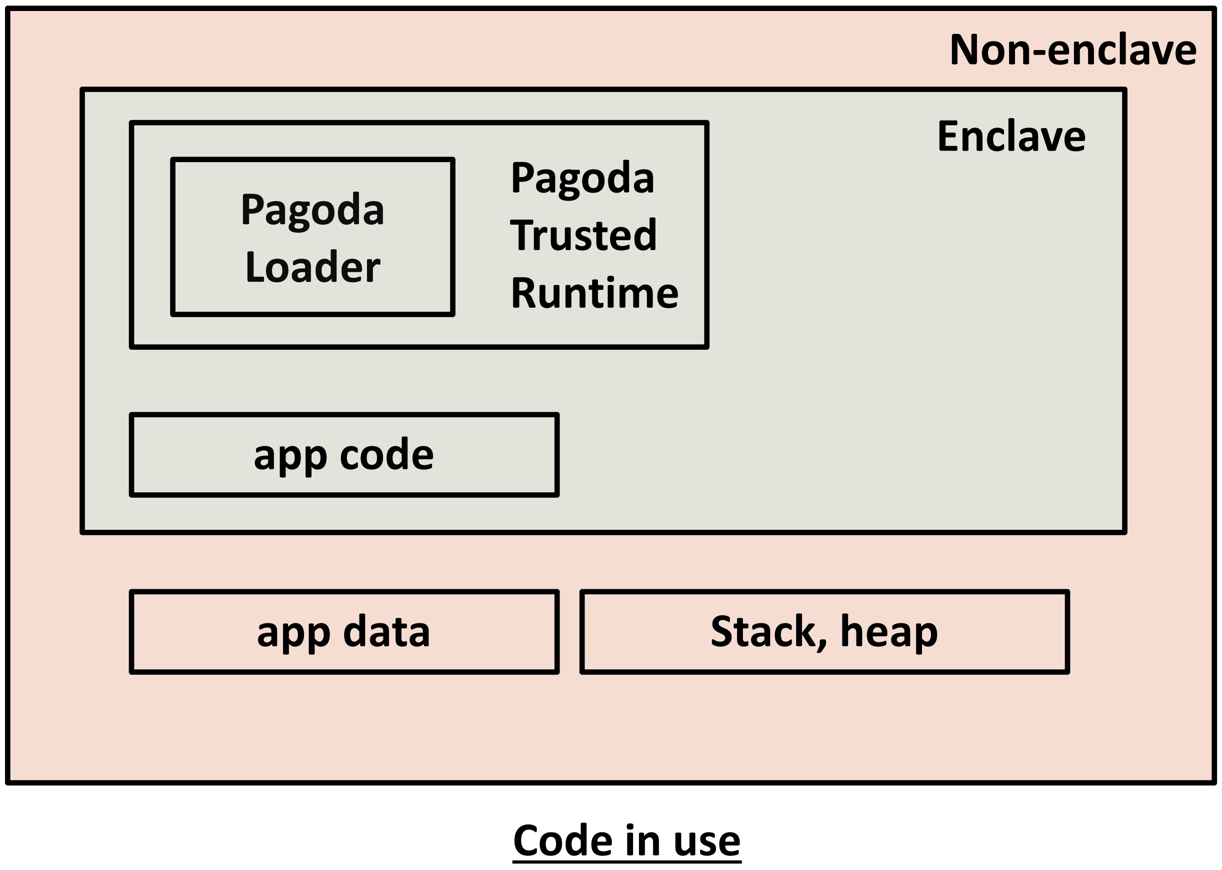


Deploying a Protected Software

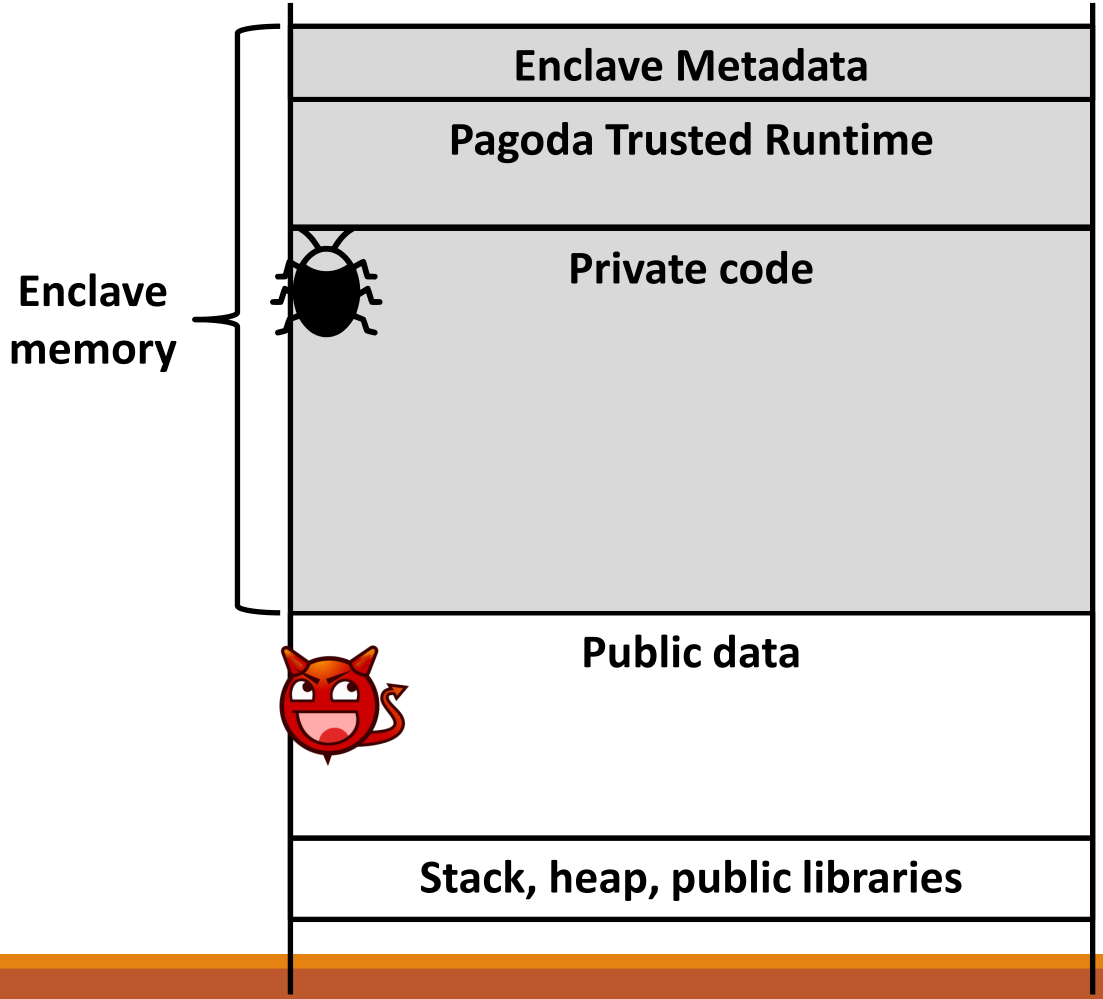


Deploying a Protected Software

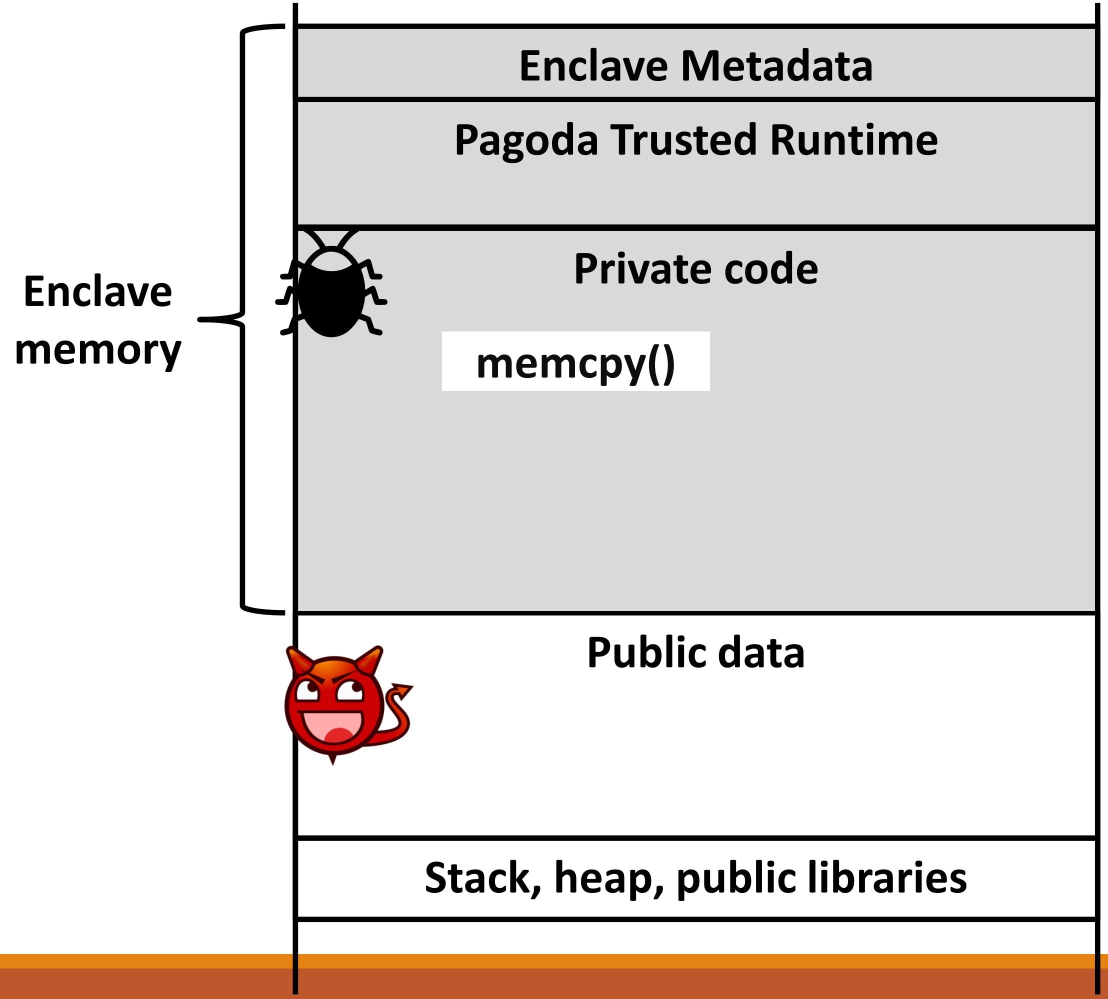




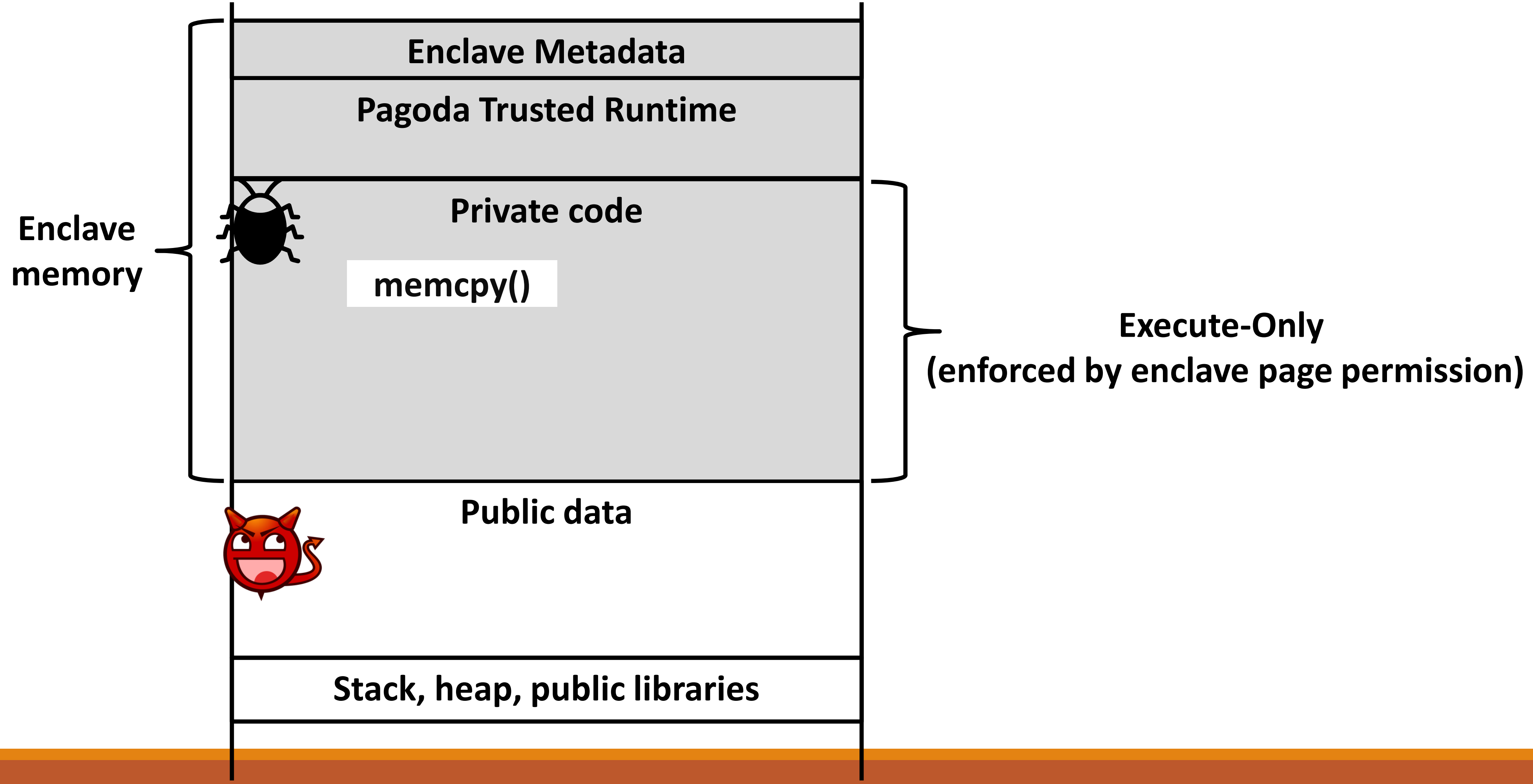
Enforcing Execute-Only Property



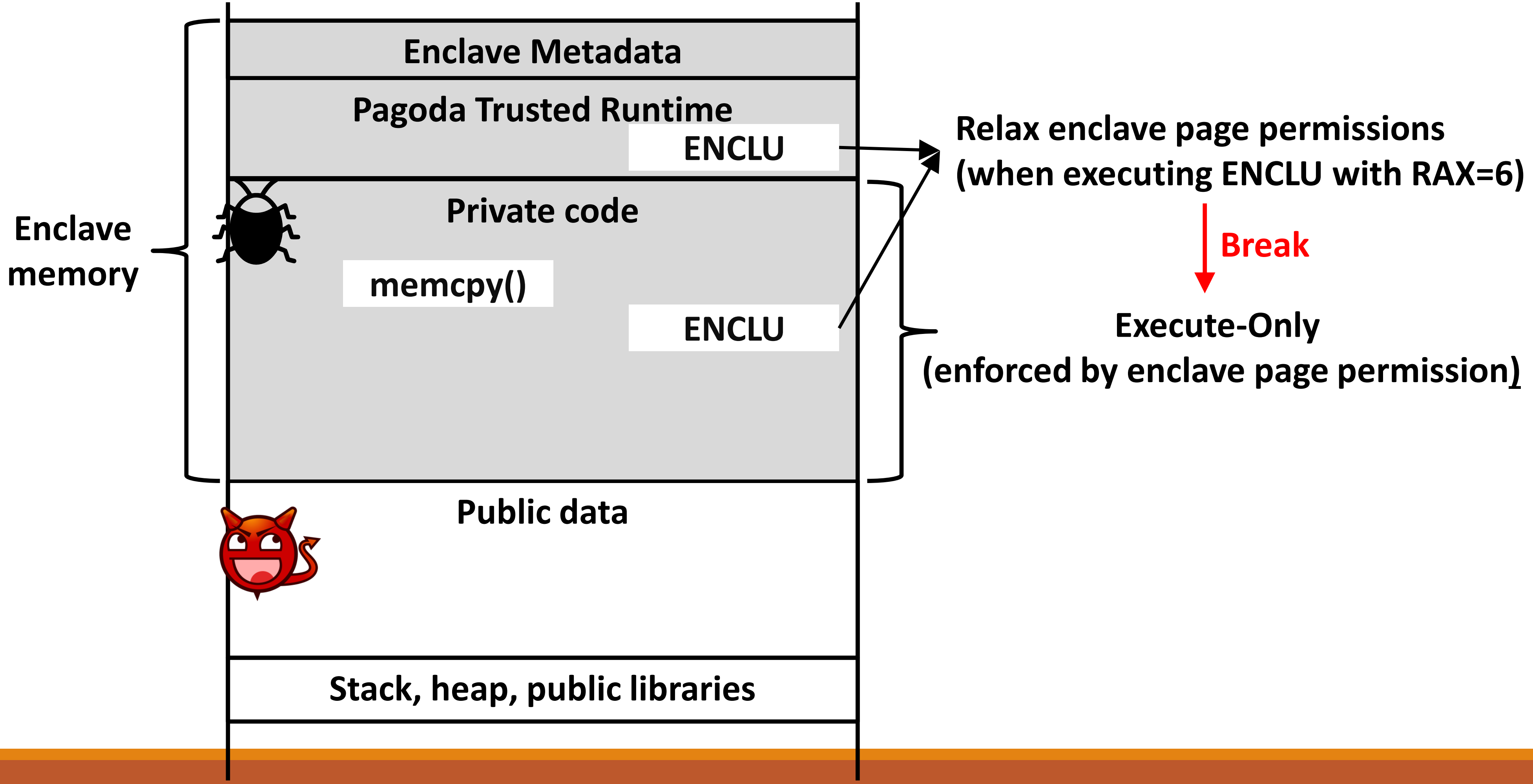
Enforcing Execute-Only Property



Enforcing Execute-Only Property

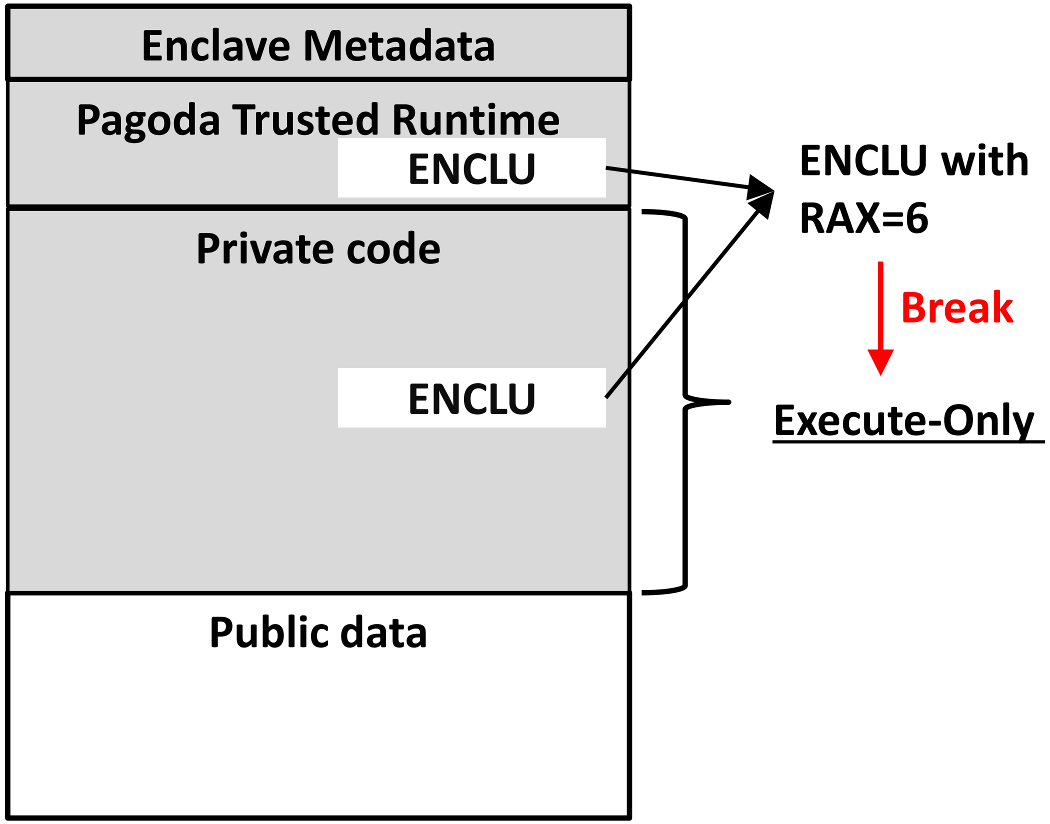


Preventing Relaxation of Execute-Only



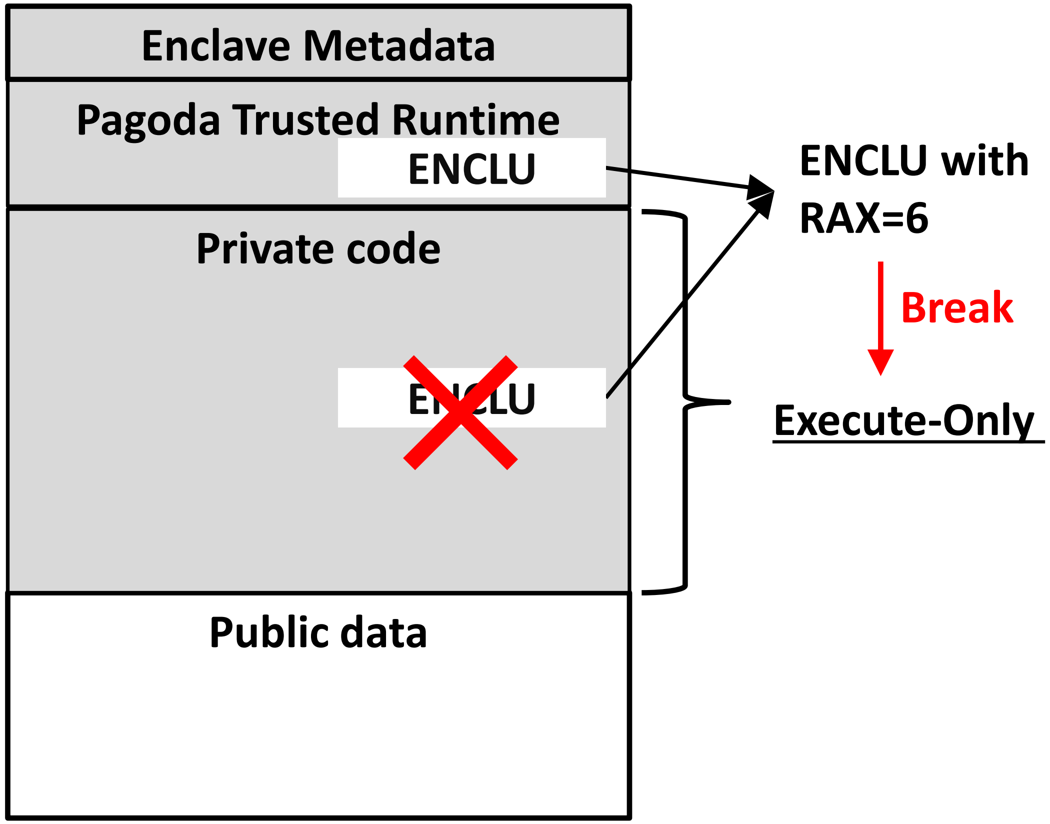
Preventing Relaxation of Execute-Only

- **Strategies:**



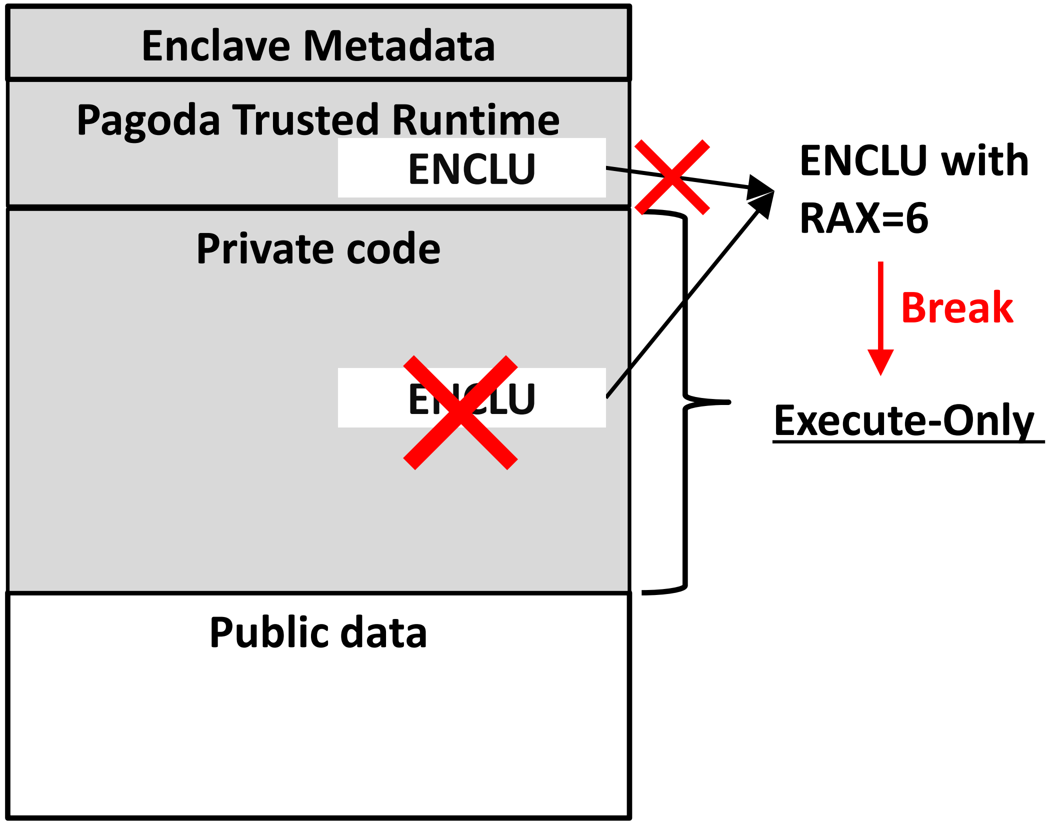
Preventing Relaxation of Execute-Only

- **Strategies:**
  1. **No ENCLU (3-byte sequence {0x0F, 0x01, 0xD7} in the application code**



Preventing Relaxation of Execute-Only

- **Strategies:**
  1. **No ENCLU (3-byte sequence {0x0F, 0x01, 0xD7} in the application code**
  2. **All ENCLU in the Pagoda Trusted Runtime cannot be used with RAX=6**



Preventing Relaxation of Execute-Only

```
/* Pagoda trusted runtime */

    enclu
    cmp rax, 6 // check if ENCLU is
               // used w/ rax=6
    jne cont
    call abort
cont:
    ……
```

Preventing Relaxation of Execute-Only

Single-threaded Applications

```
/* Pagoda trusted runtime */

    enclu
    cmp rax, 6 // check if ENCLU is
               // used w/ rax = 6
    jne cont
    call abort
cont:
    ……
```

Thread 1

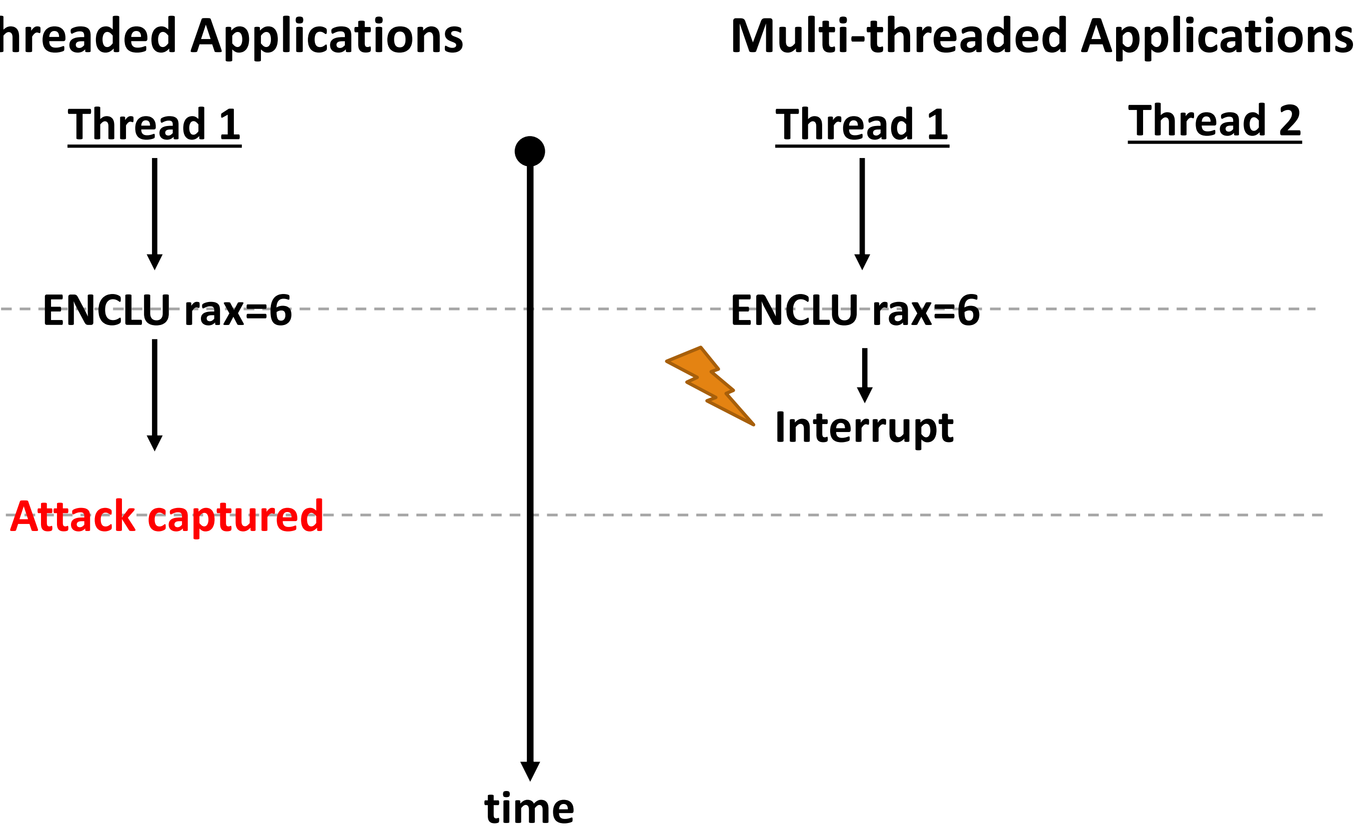
ENCLU rax=6

Attack captured

time

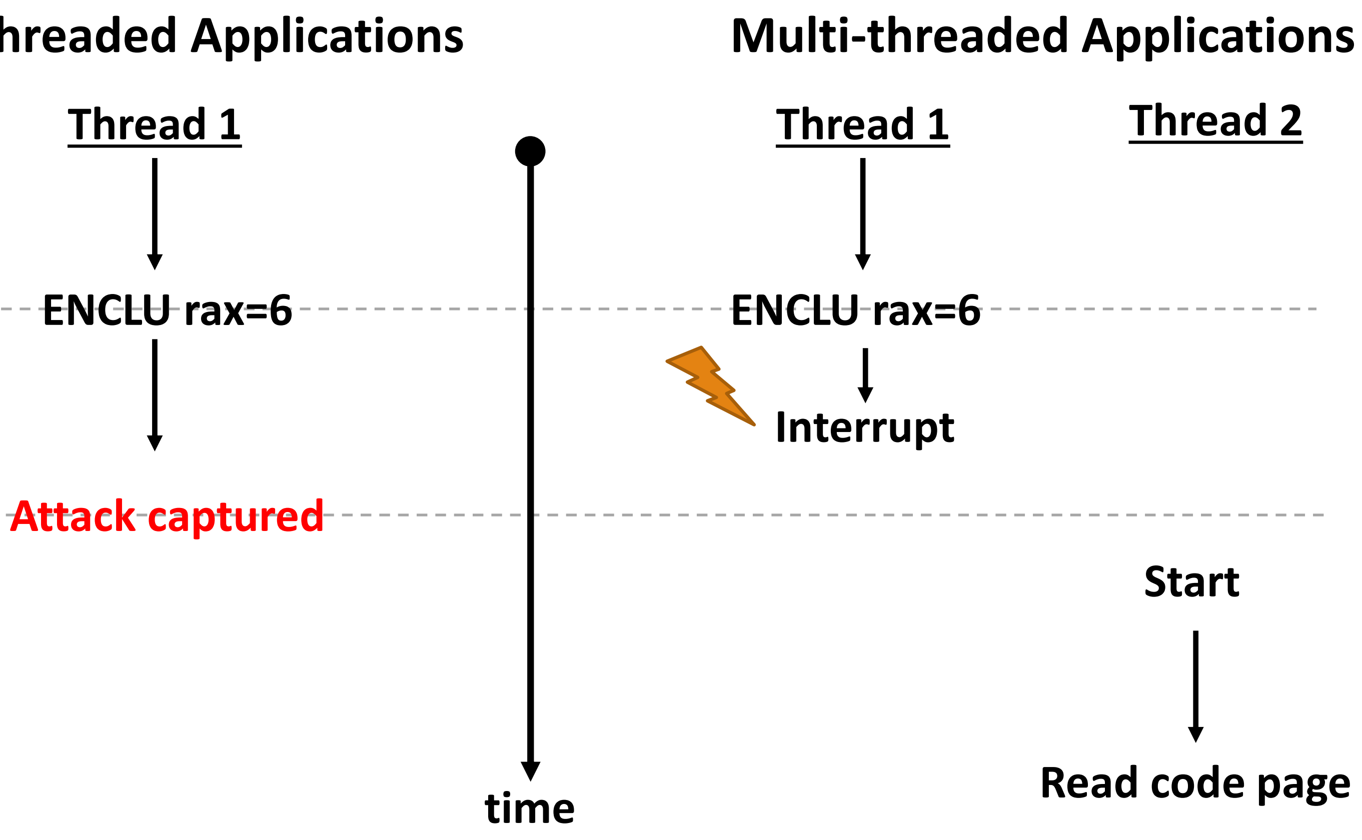
Preventing Relaxation of Execute-Only

```
/* Pagoda trusted runtime */

    enclu
    cmp rax, 6 // check if ENCLU is
               // used w/ rax = 6
    jne cont
    call abort
cont:
    ……
```



Preventing Relaxation of Execute-Only

```
/* Pagoda trusted runtime */

    enclu
    cmp rax, 6 // check if ENCLU is
               // used w/ rax = 6
    jne cont
    call abort
cont:
    ……
```



Preventing Relaxation of Execute-Only

```
/* Pagoda trusted runtime */

    enclu
    cmp rax, 6 // check if ENCLU is
               // used w/ rax = 6
    jne cont
    call abort
cont:
    ……
```

Single-threaded Applications

Thread 1

ENCLU rax=6

Attack captured

time

Multi-threaded Applications

Thread 1 **Thread 2**

ENCLU rax=6

Solution: A hardware fix (microcode update)

Enclave can permanently prohibit ENCLU rax=6

Read code page

Additional Features

- Fast cross-enclave-boundary calls
- Supporting different types of applications using dynamic linking/loading
- Multi-threading, signal-handling, etc.

Evaluation

Benchmark	Type	Performance Metric	Pagoda / Native Linux
SPEC CPU 2017	Standardized benchmark	Execution time	102.1%
Lighttpd	Server applications	Peak Throughput	28.9%
Memcached			51.1%
Quake	Desktop applications	Frame-per-second (FPS)	86.2%
Witchblast			97.0%

Conclusion

This work proposes Pagoda, the first practical and high-performance code privacy protection system based on SGX.

Q & A

Backup slides

Evaluation – SPEC CPU 2017

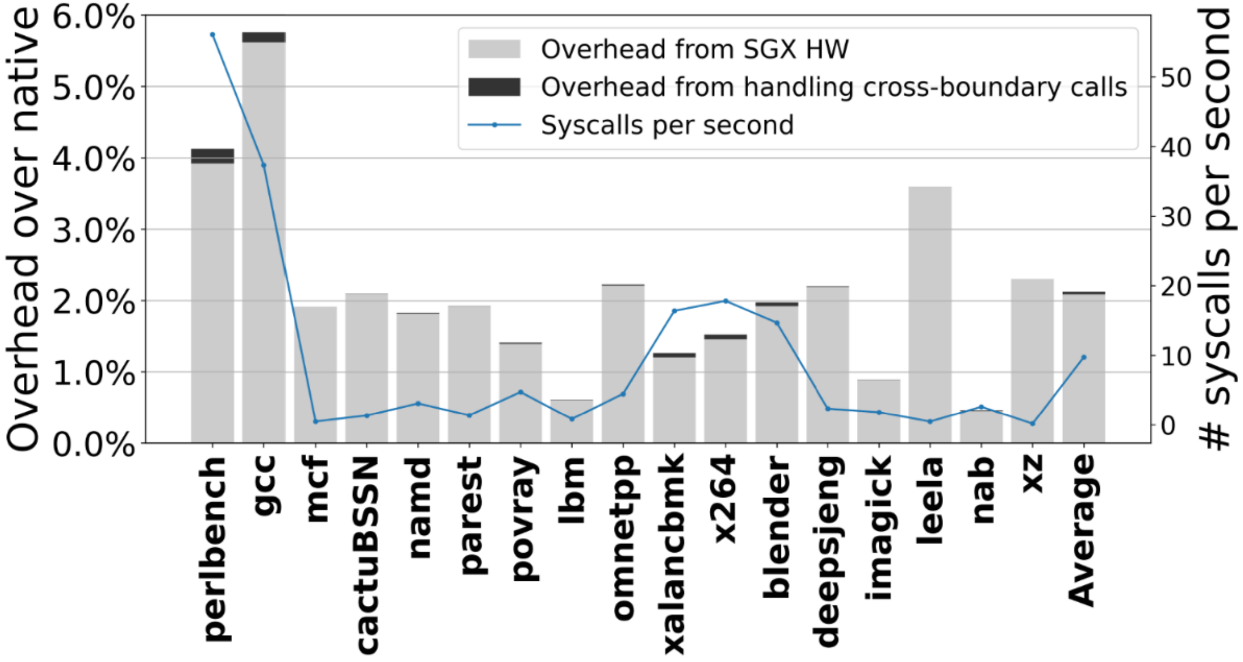


Fig. 5: The breakdown of Pagoda's performance overhead over native Linux execution for SPEC2017 benchmarks.

Evaluation – Server Applications

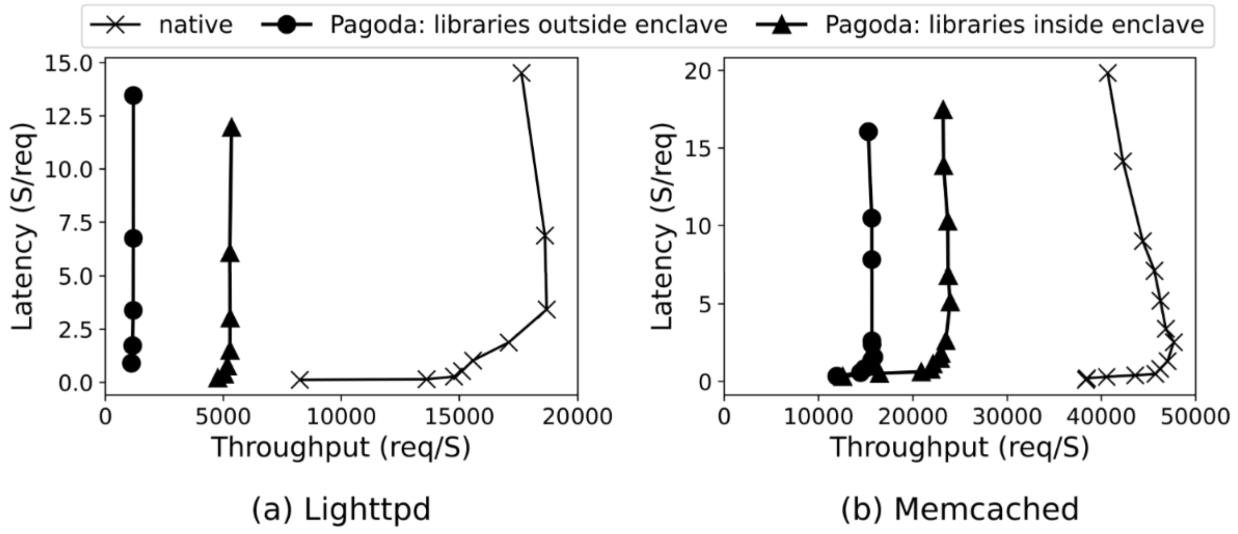


Fig. 6: Throughput vs. Latency of Lighttpd and Memcached. We run both applications with three configurations: bare-metal Linux, Pagoda with all shared libraries outside the enclave (treating all libraries as public), Pagoda with all libraries inside the enclave (treating all libraries as private).

Evaluation – Desktop (Gaming) Applications

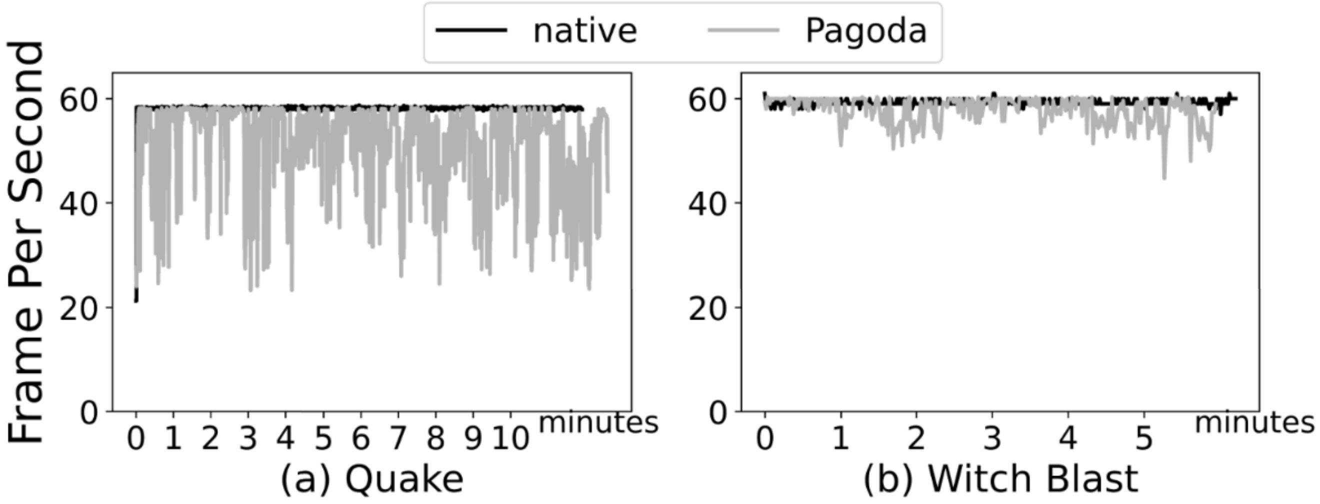
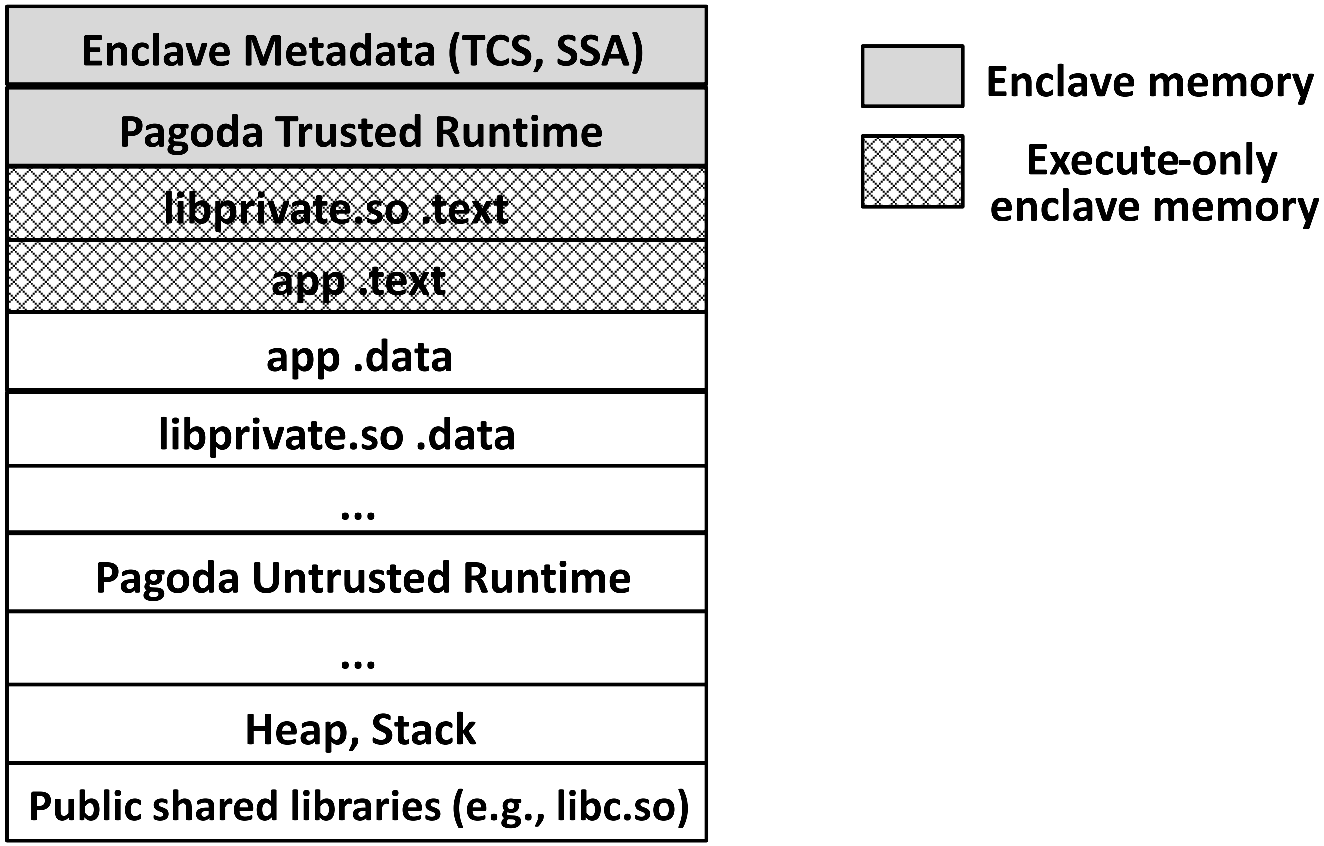


Fig. 7: Comparing the change of Frame-Per-Second over time between native Linux execution and Pagoda.

Pagoda memory layout

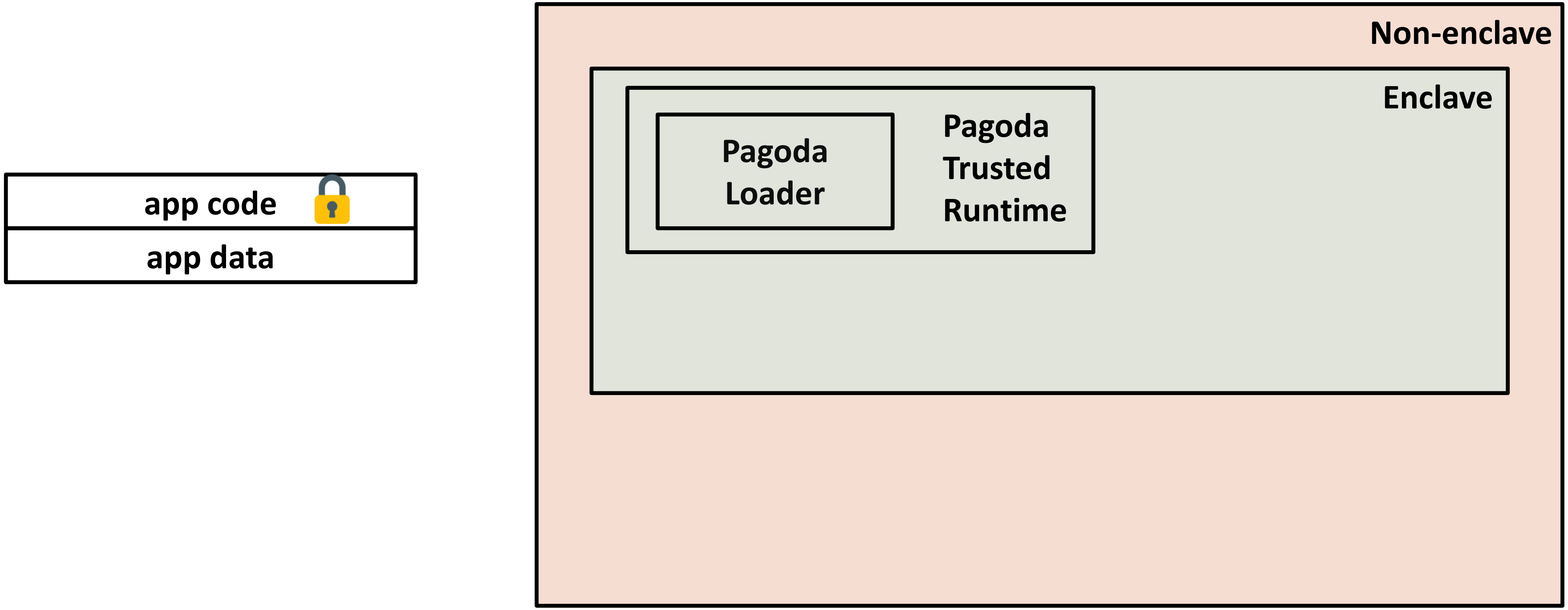


Direct vs. Indirect attack

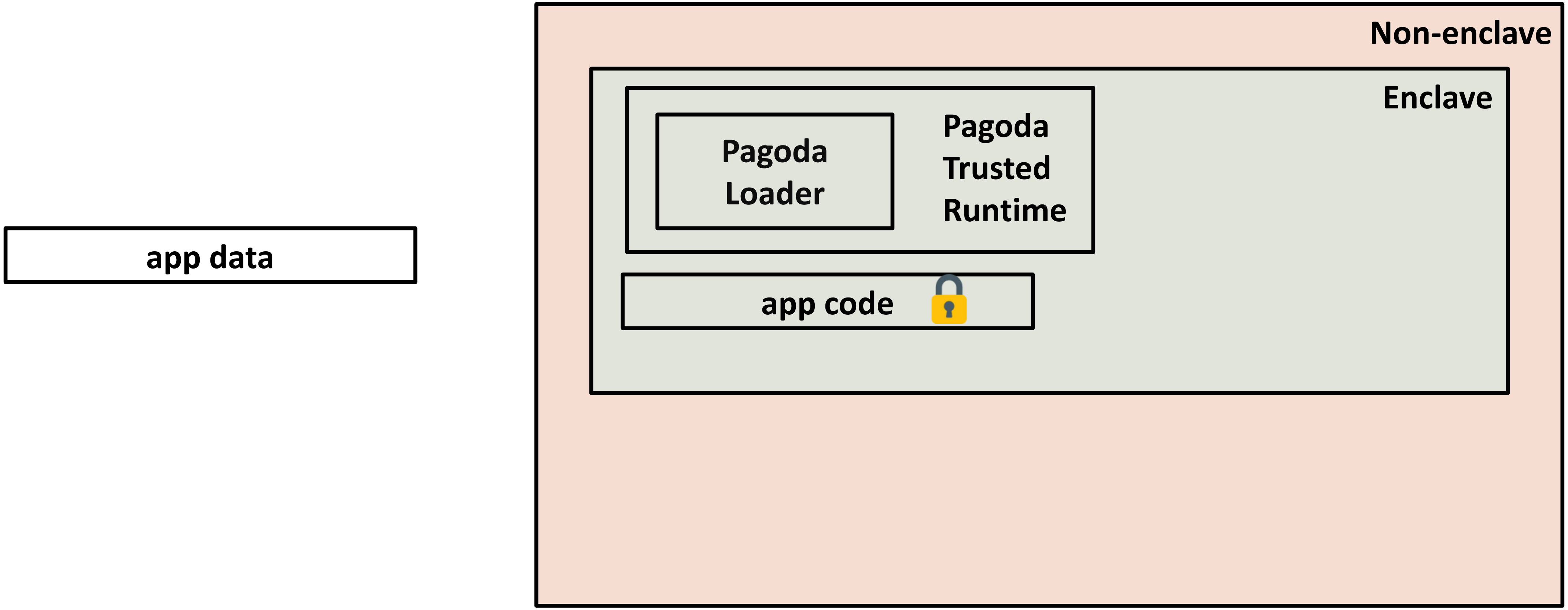
Direct attack starts w/ direct read of code. All attack discussion so far is about this type

Indirect attack starts w/ execute code & some form of observation of the system (data memory, uarch usage) This require huge effort, non-trivial, but possible (to at least reconstruct some information about the code), future work. See discussion section in the paper.

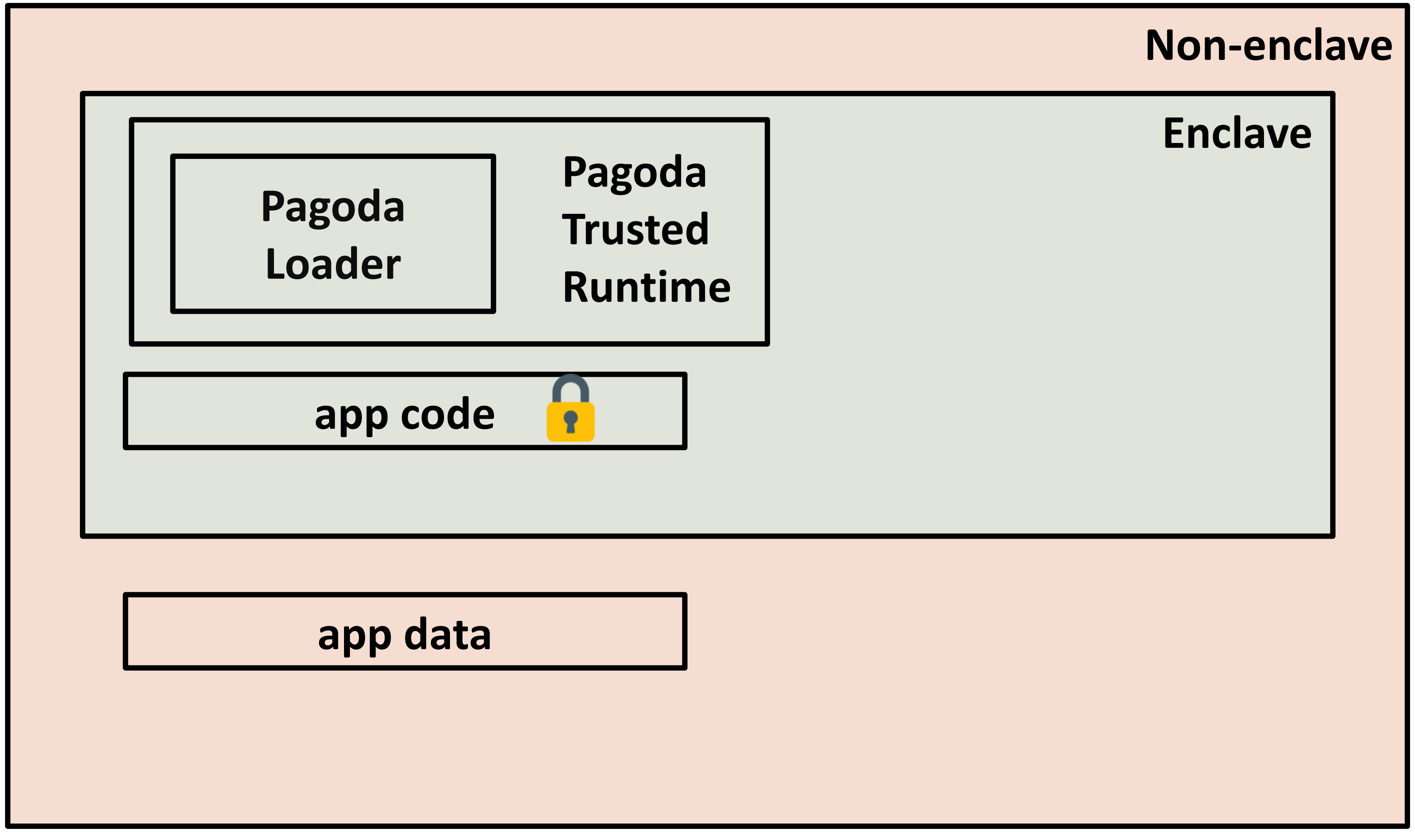
Deploying a Protected Software



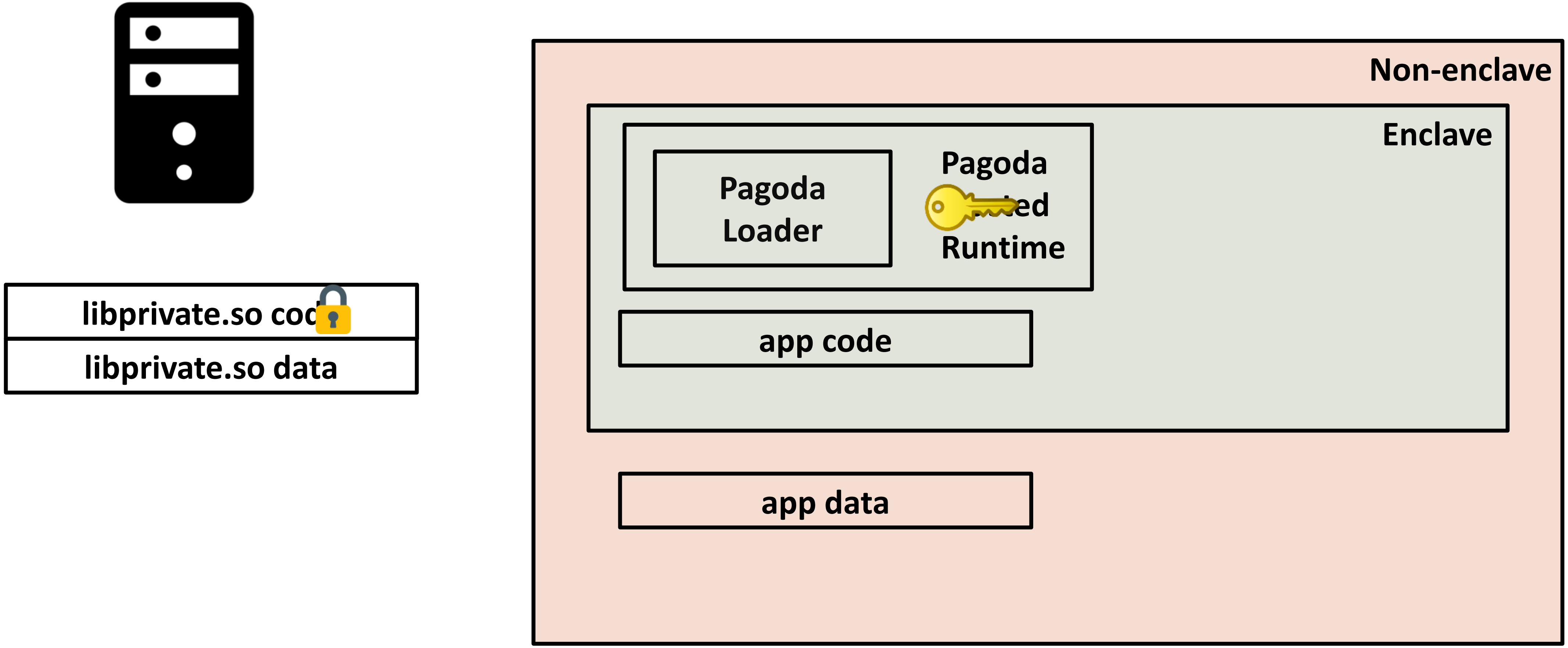
Deploying a Protected Software



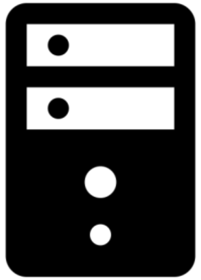
Deploying a Protected Software

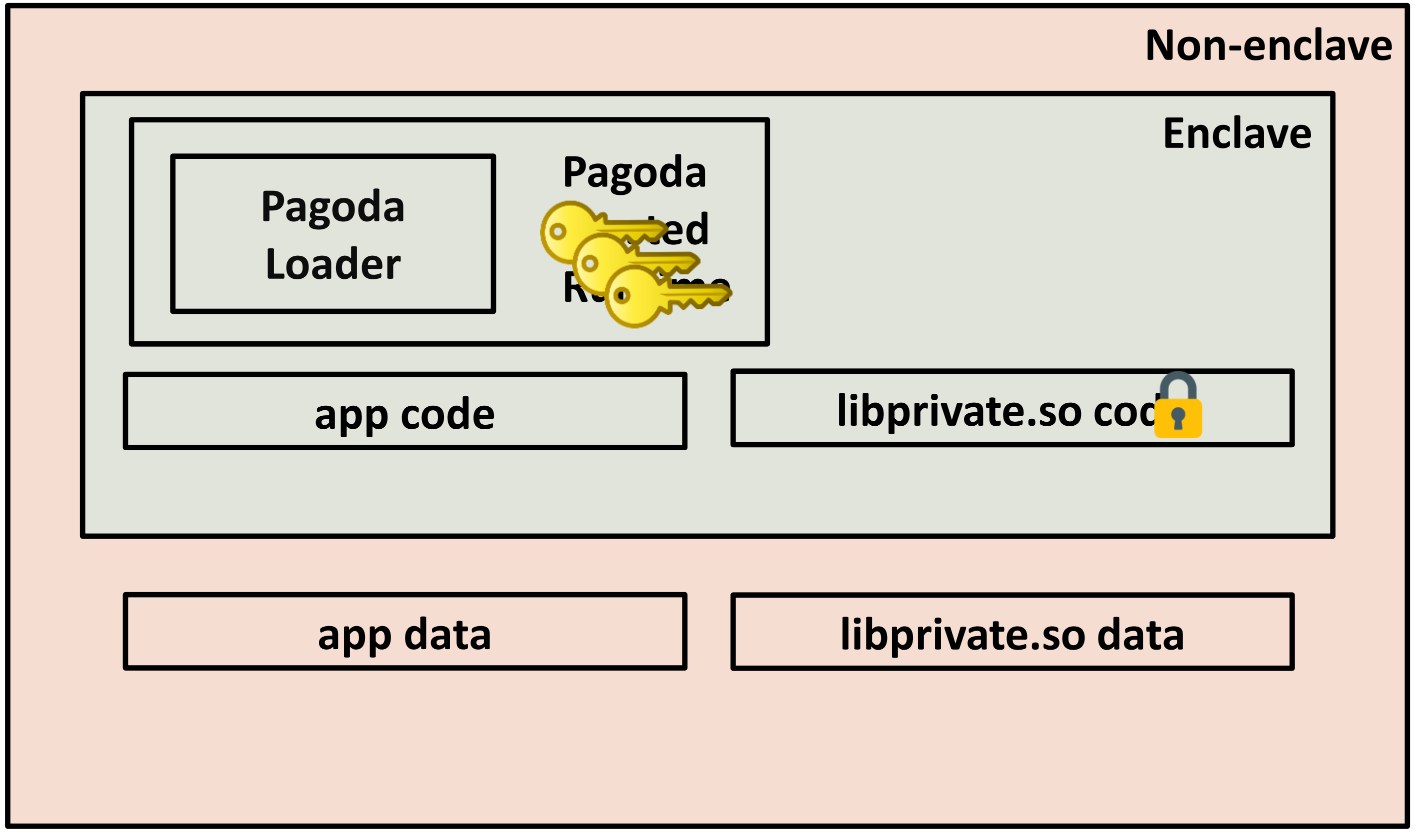


Deploying a Protected Software

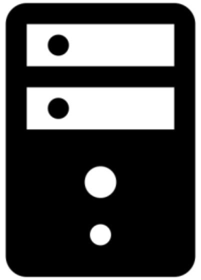


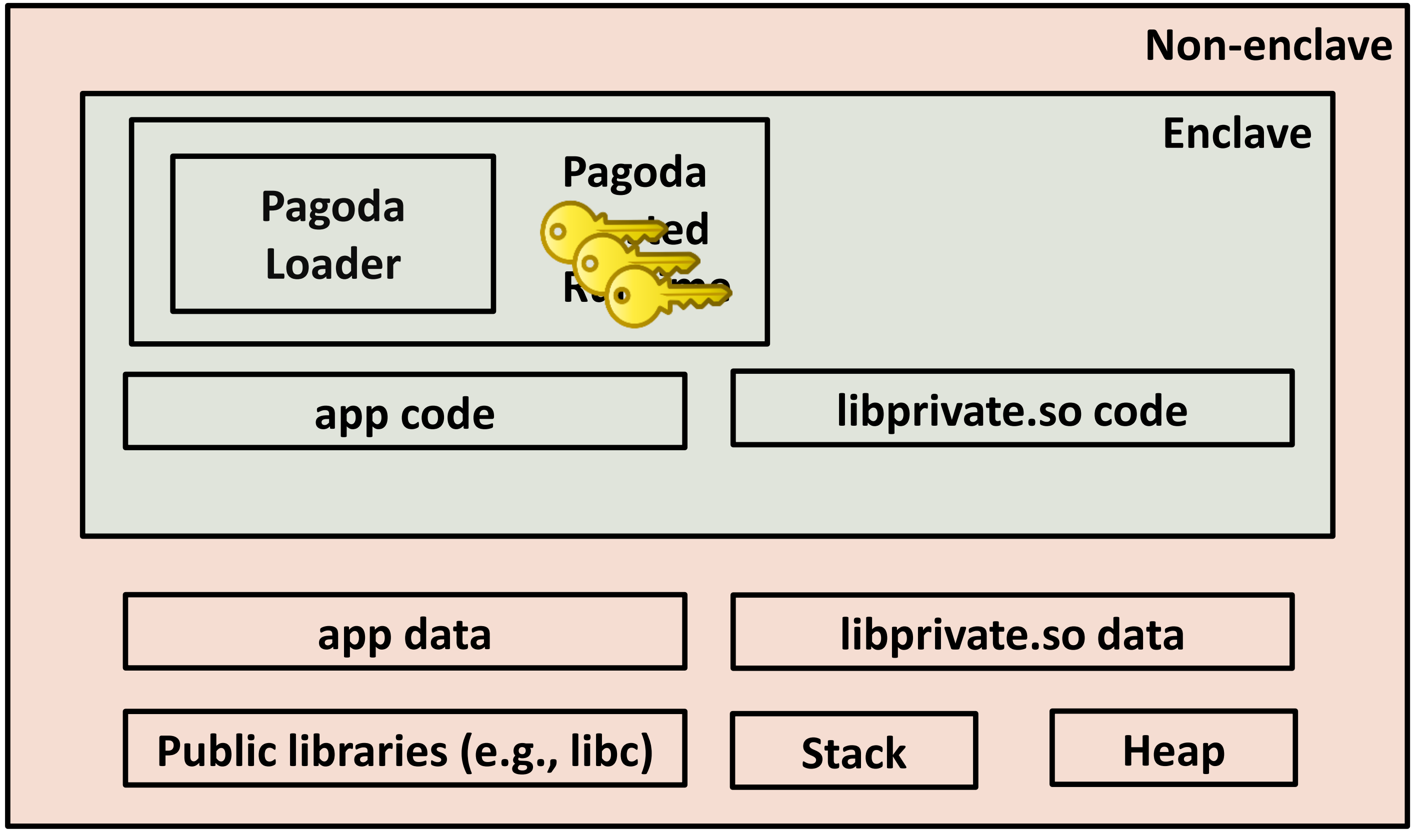
Deploying a Protected Software





Deploying a Protected Software





Meeting notes

1. Add an overview slide: objective: lifetime code privacy. Make a graph depicts in transit, at rest, in use. Say it's the standard way to fix in transit/at rest (prior works) with remote attestation/encryption. We focus on in-use in Pagoda. In 10-min version, replace all the slides on this topic with this overview slide.

2. keep trusted runtime term, add explanation what it does and loader is part of it

3. 23->24: copy the figure, when saying ½, point to the enclu in the figure, make it clear that we eliminate enclu in app binary, and regulate the enclu in Pagoda runtime

4. cut eval slide details (NUC…)

5. server app graph: more requests along the curve

6. backup slide #1: detailed memory layout, explain what we do for making this layout

7. backup slide #2: discussion section, direct/indirect attack

Direct attack starts w/ direct read of code. All attack discussion so far is about this type

Indirect attack starts w/ execute code & some form of observation of the system (data memory, uarch usage) This require huge effort, non-trivial, but possible (to at least reconstruct some information about the code), future work. See discussion section in the paper.

Question: what's the contribution of the paper:

1) insight: for code privacy, just put code in enclave -> great compatibility

2) for insight, we need to address some problems: a) functionality. See backup slide, and the communication stuff b) security: a practical assumption (vulnerable code), we tackle all vulnerability problems to use of XOM. Also boil down all ways to break XOM into preventing the use of EMODPE. And study how to solve this problem for single/multi-thread app

Chris’s notes

- minor: since xinyang was at MSR when he did pagoda, not sure we want the databricks logo
- slide 4: say for both settings, the software stack is untrusted. you said "platform", which could also mean HW.
- slide 9: I don't see the most important thing on the slide "enclave code is in TCB"
- slide 12: it's not just moving data/in out. that doesn't really cover the syscall benefits.
- slide 12: it's not as straightforward as XOM. you still need to ensure XOM is maintained throughout app lifetime. we should allude to this being a problem we will have to solve, but not say (yet) how we will solve it.
- slide 18: suggested organization: say we need to protect code throughout its lifetime = in use + in transit + at rest. here is how we deal with each: ... (and make it clear that code in transit/at rest isn't our contribution)
- slide 25: the red X is a bit vague. maybe spell out "attack detected"?
- slide 31: you might want to say something about syscalls, because you don't mention earlier how they are performed.
- slide 32: why are the graphs bending backwards on themselves!?
- slide 33: it's a bit counter-intuitive that the overcall to syscalls would create such a large dip.

Weidong, Xinyang