PartEmu: Enabling Dynamic Analysis of Real-World TrustZone Software Using Emulation

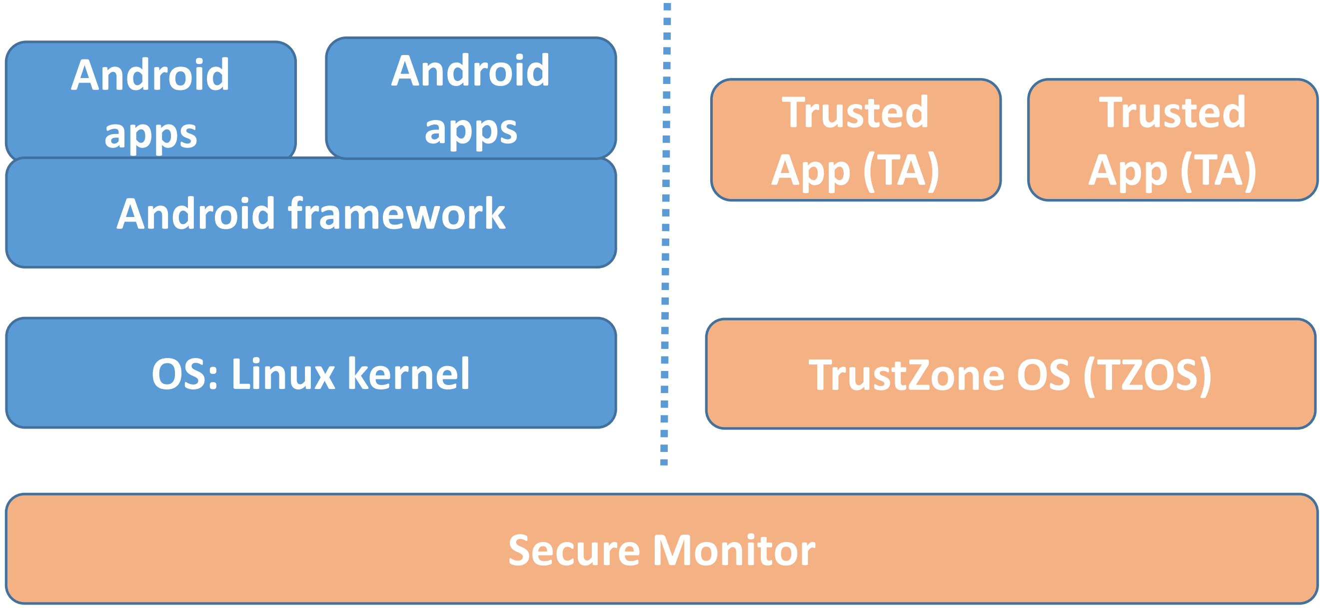
Lee Harrison, Hayawardh Vijayakumar, Michael Grace

Samsung Knox, Samsung Research America

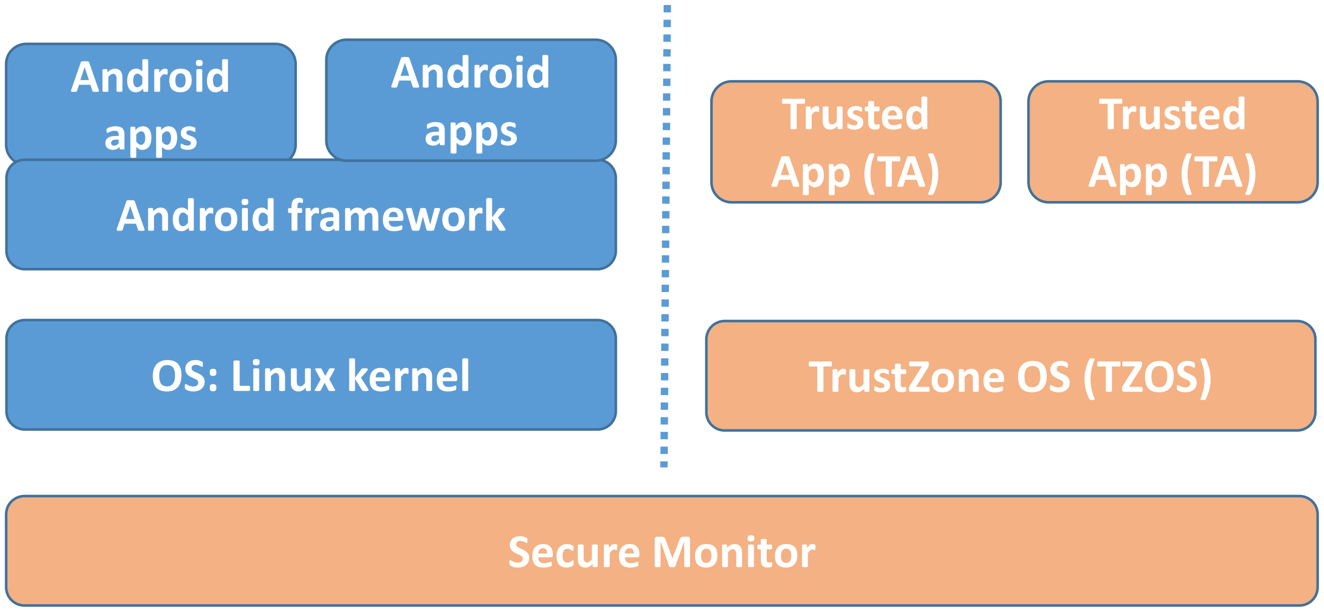
Rohan Padhye, Koushik Sen

EECS Department, University of California, Berkeley

The “Hidden” Software Stack: TrustZone

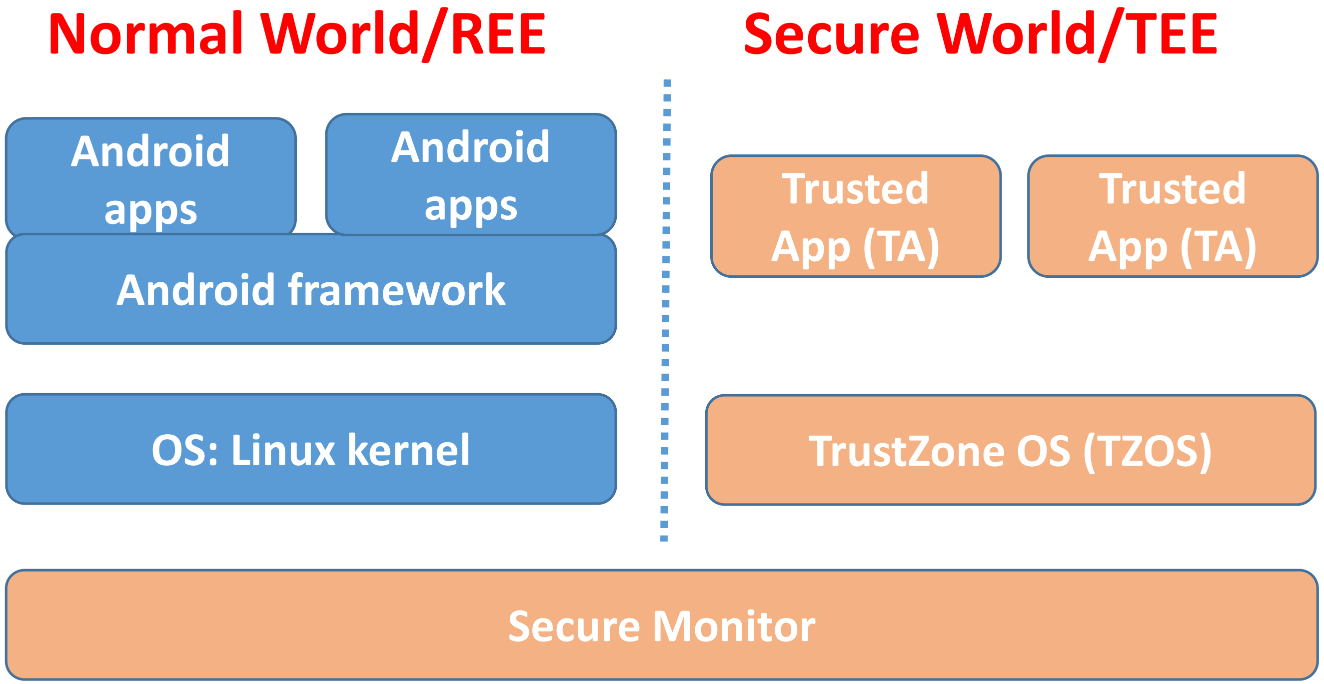


The "Hidden" Software Stack: TrustZone



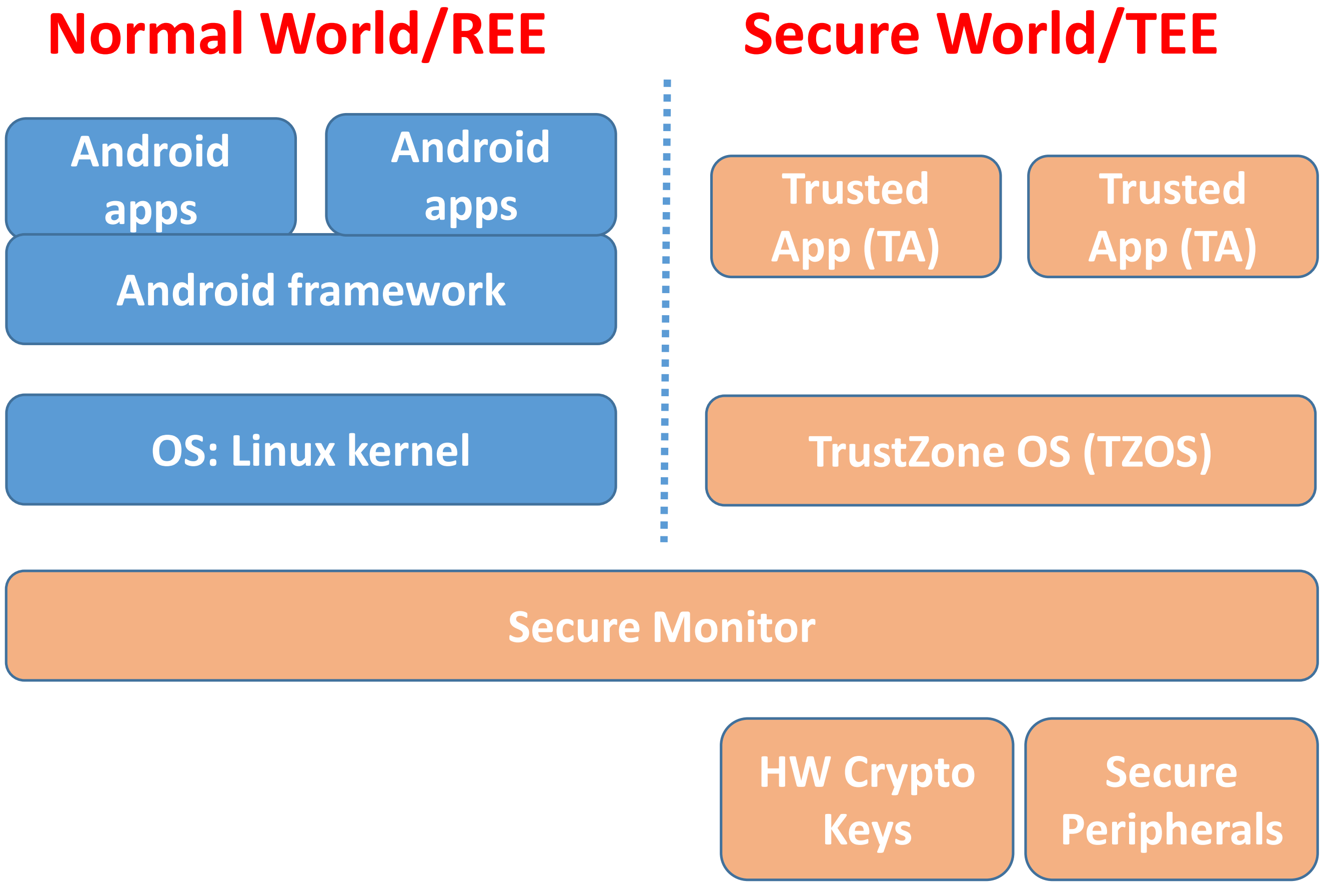
- Separate software stack
  - Trusted applications (TAs)
  - TrustZone OS (TZOS)

The “Hidden” Software Stack: TrustZone



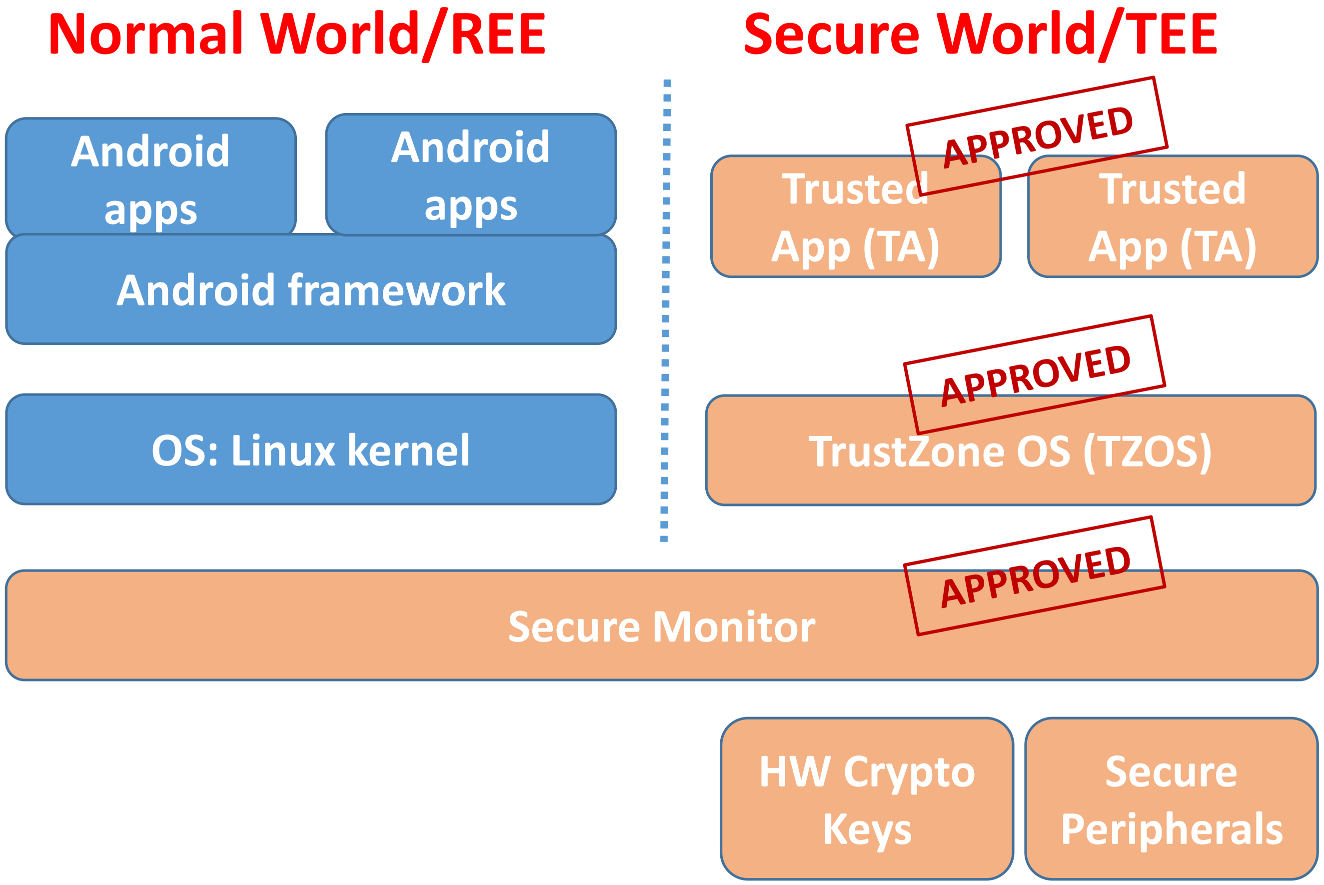
- Separate software stack
  - Trusted applications (TAs)
  - TrustZone OS (TZOS)
  - TEE/REE

The “Hidden” Software Stack: TrustZone



- Separate software stack
  - Trusted applications (TAs)
  - TrustZone OS (TZOS)
  - TEE/REE
- Basis for security: Has access to hardware keys

The “Hidden” Software Stack: TrustZone



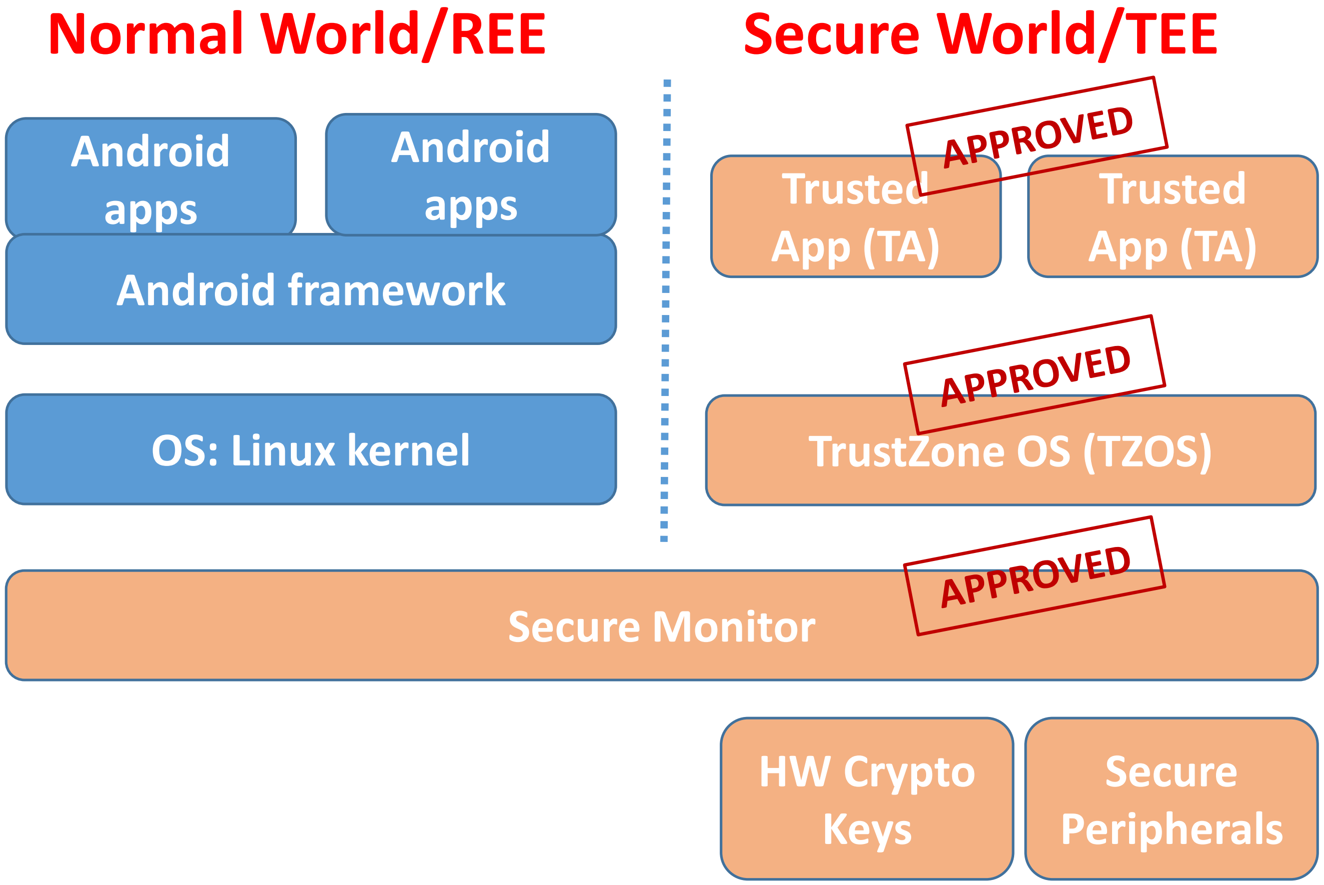
- Separate software stack
  - Trusted applications (TAs)
  - TrustZone OS (TZOS)
  - TEE/REE
- Basis for security: Has access to hardware keys
- Access to TZ locked down: Only signed software can run

Problem: Dynamic analysis of TZ is hard!

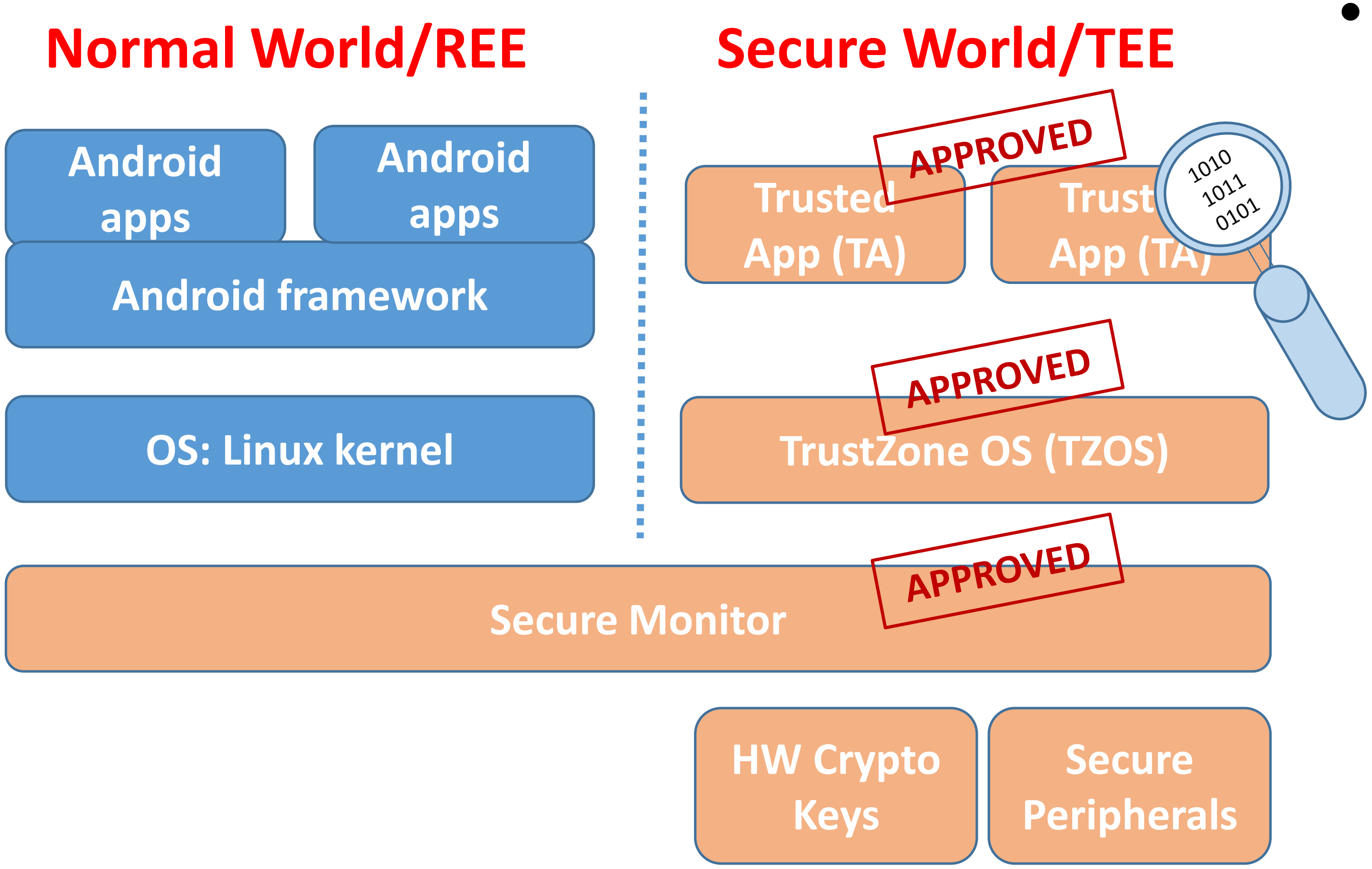
Approach

Results: What did we learn?

Problem: Dynamic Analysis of TZ is Hard

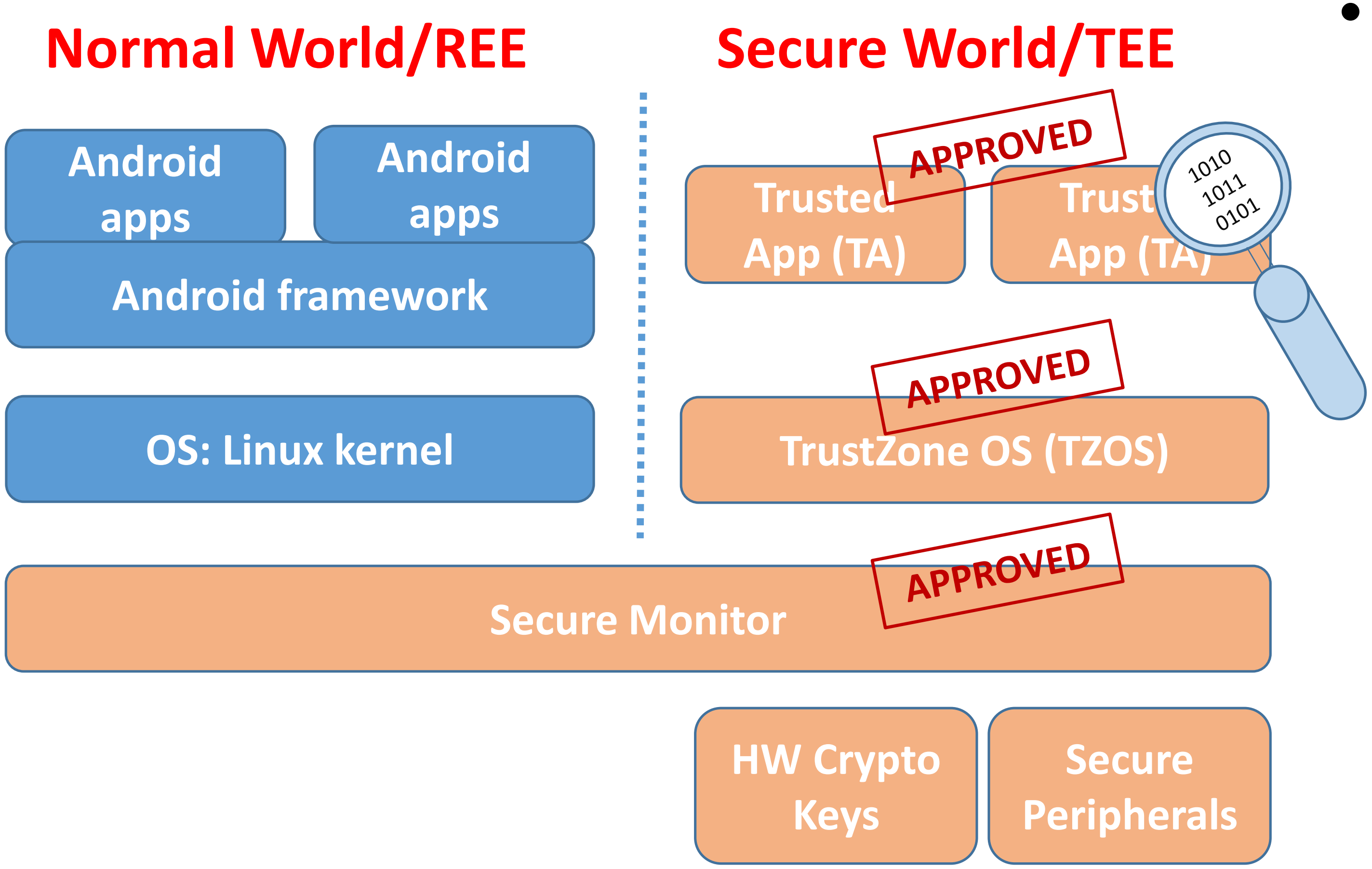


Problem: Dynamic Analysis of TZ is Hard



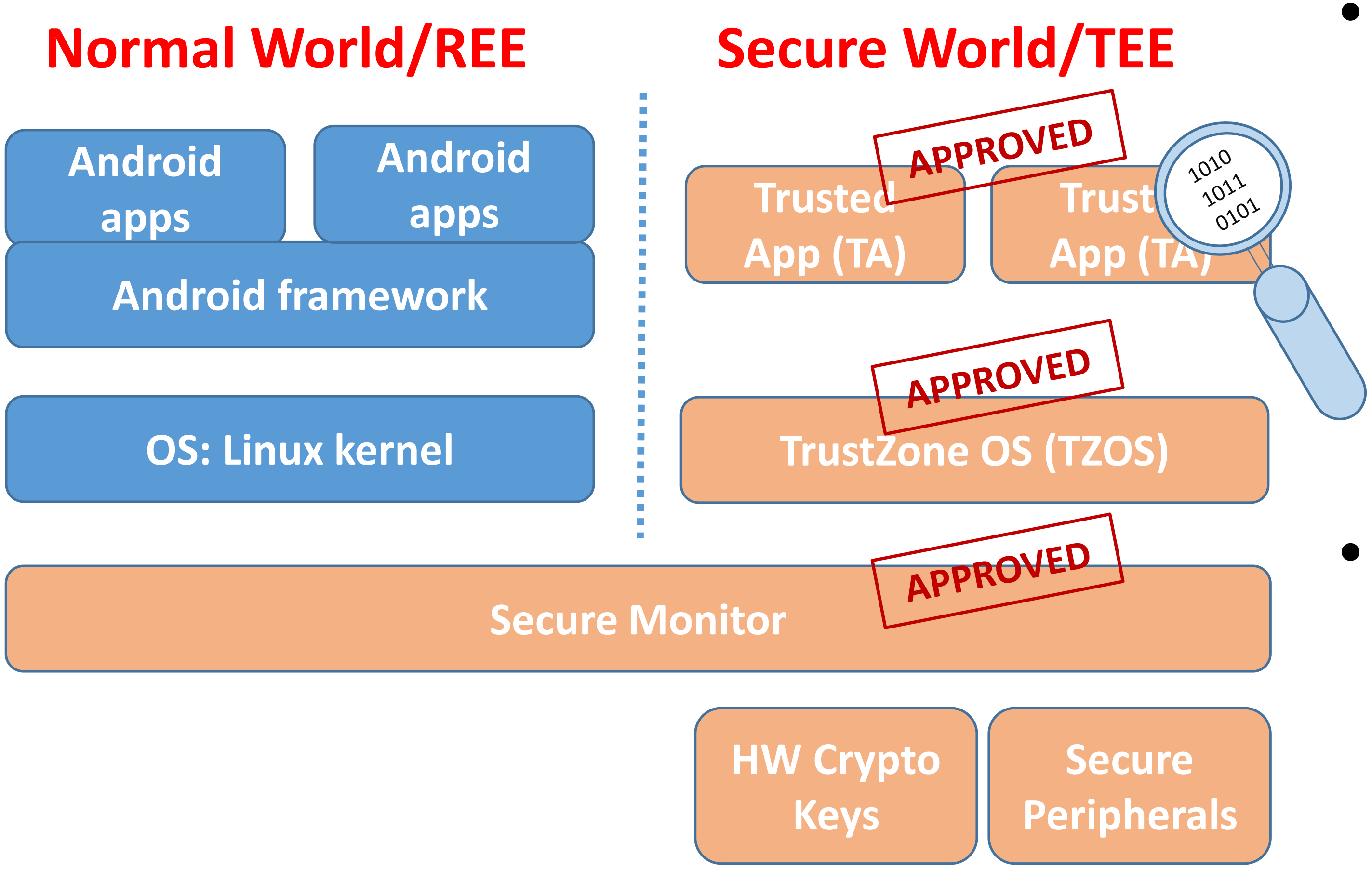
- Dynamic analysis needs ability to monitor target

Problem: Dynamic Analysis of TZ is Hard



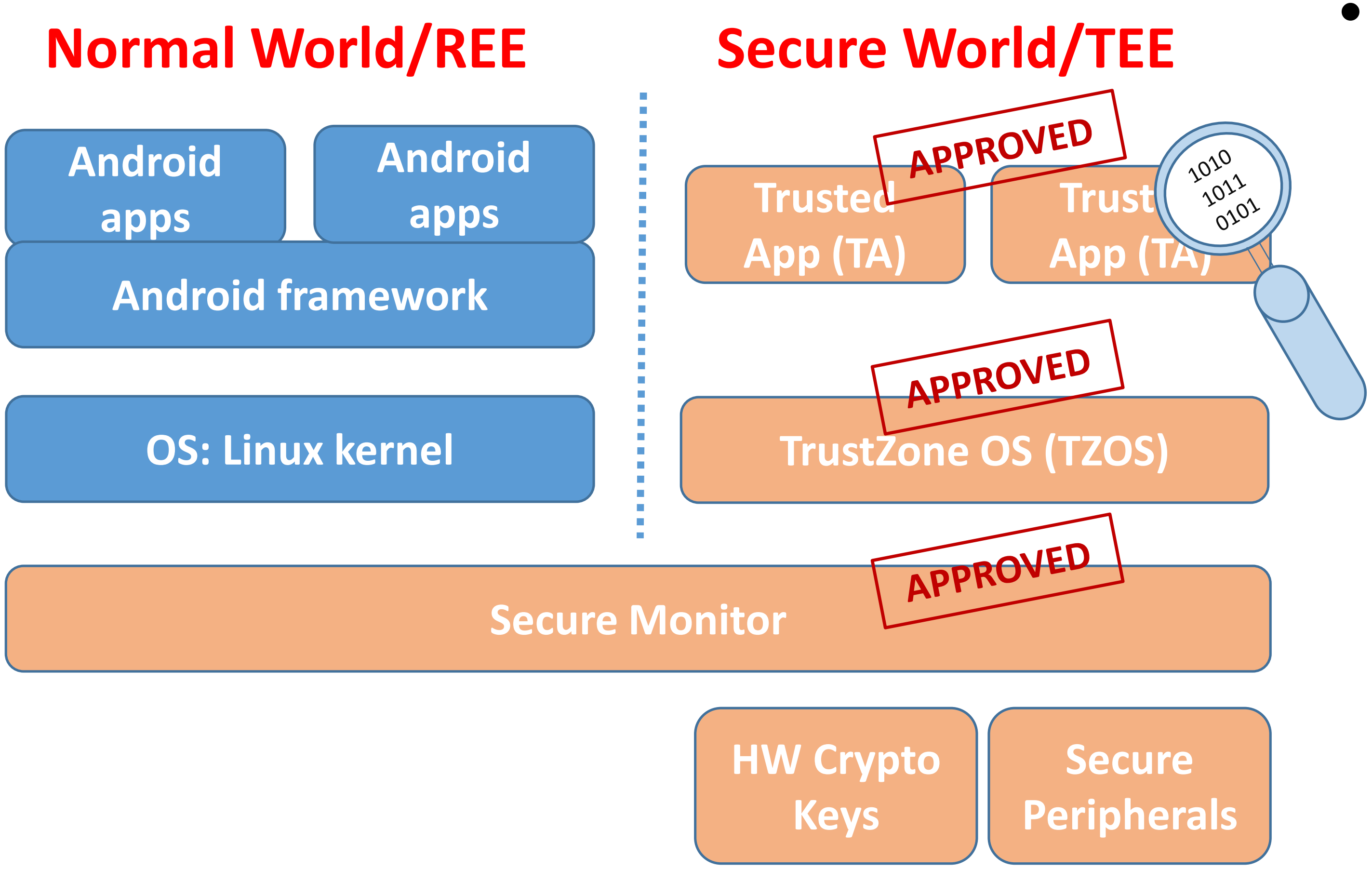
- Dynamic analysis needs ability to monitor target
  - Debugging – needs memory/registers
  - Feedback-driven fuzz testing – needs list of basic blocks covered

Problem: Dynamic Analysis of TZ is Hard



- Dynamic analysis needs ability to monitor target
  - Debugging – needs memory/registers
  - Feedback-driven fuzz testing – needs list of basic blocks covered
- However, cannot instrument TZ software or monitor TZ memory due to signing!

Problem: Dynamic Analysis of TZ is Hard



- Prior dynamic analysis approaches limited!
  - TA/TZOS binary reverse engineering
  - Fuzz testing without feedback

Solution: Dynamic Analysis By Emulation

- We build an emulator that runs real-world TZOSes and TAs

Solution: Dynamic Analysis By Emulation

- We build an emulator that runs real-world TZOSes and TAs
- Emulation enables dynamic analysis
  - Allows introspection and monitoring of TZ execution

Solution: Dynamic Analysis By Emulation

- We build an emulator that runs real-world TZOSes and TAs
- Emulation enables dynamic analysis
  - Allows introspection and monitoring of TZ execution
- We support four widely-used real-world TZOSes:
  - Qualcomm’s QSEE
  - Trustonic’s Kinibi
  - Samsung’s TEEGRIS
  - Linaro’s OP-TEE

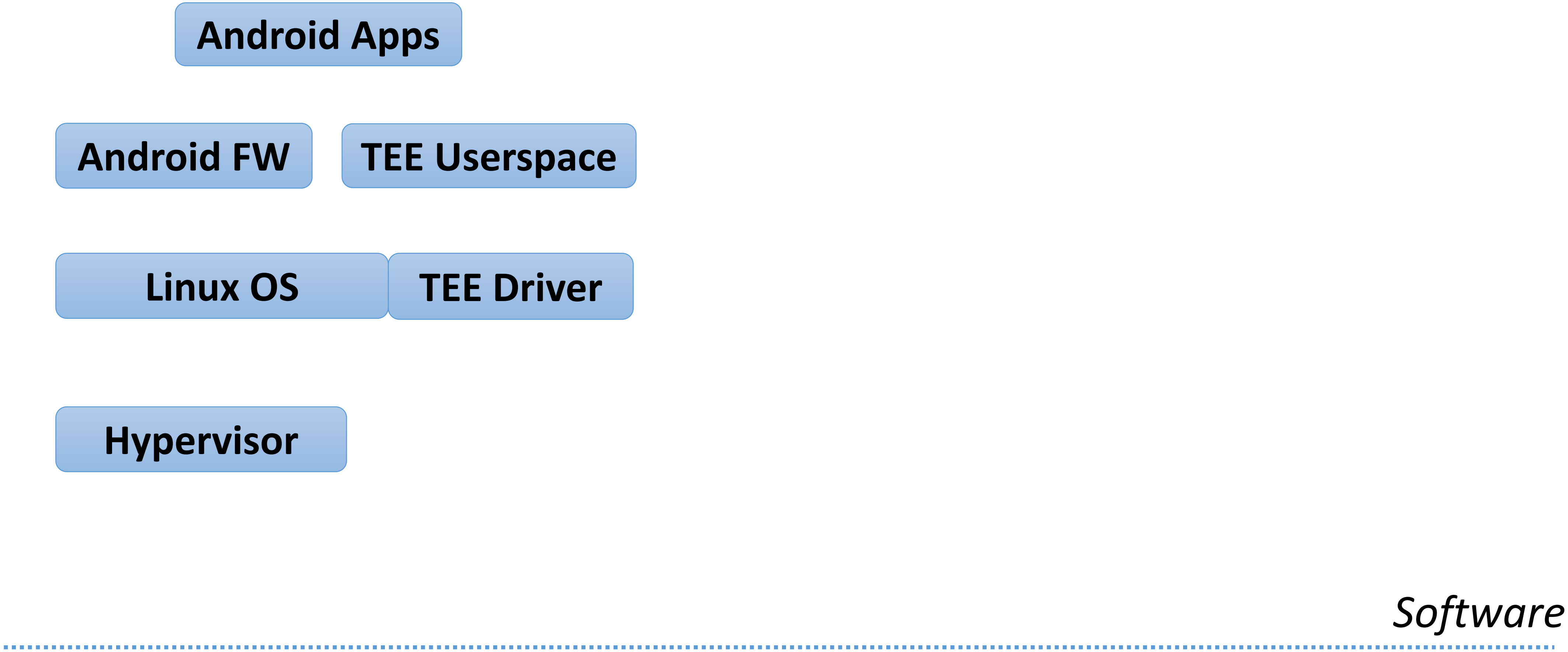
Problem: Dynamic analysis of TZ is hard!

Approach: How did we run TZ in an emulator?

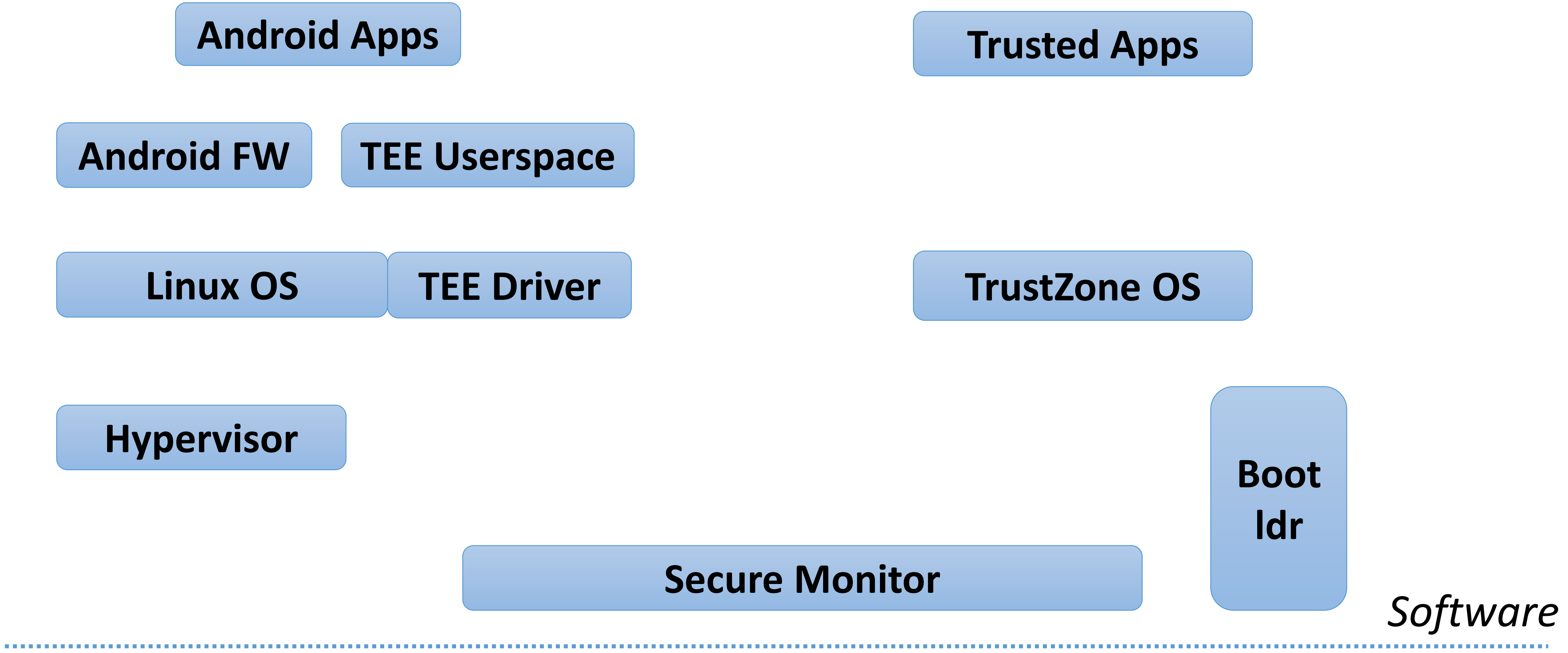
Results: What did we learn?

Challenge: Large Number of Components

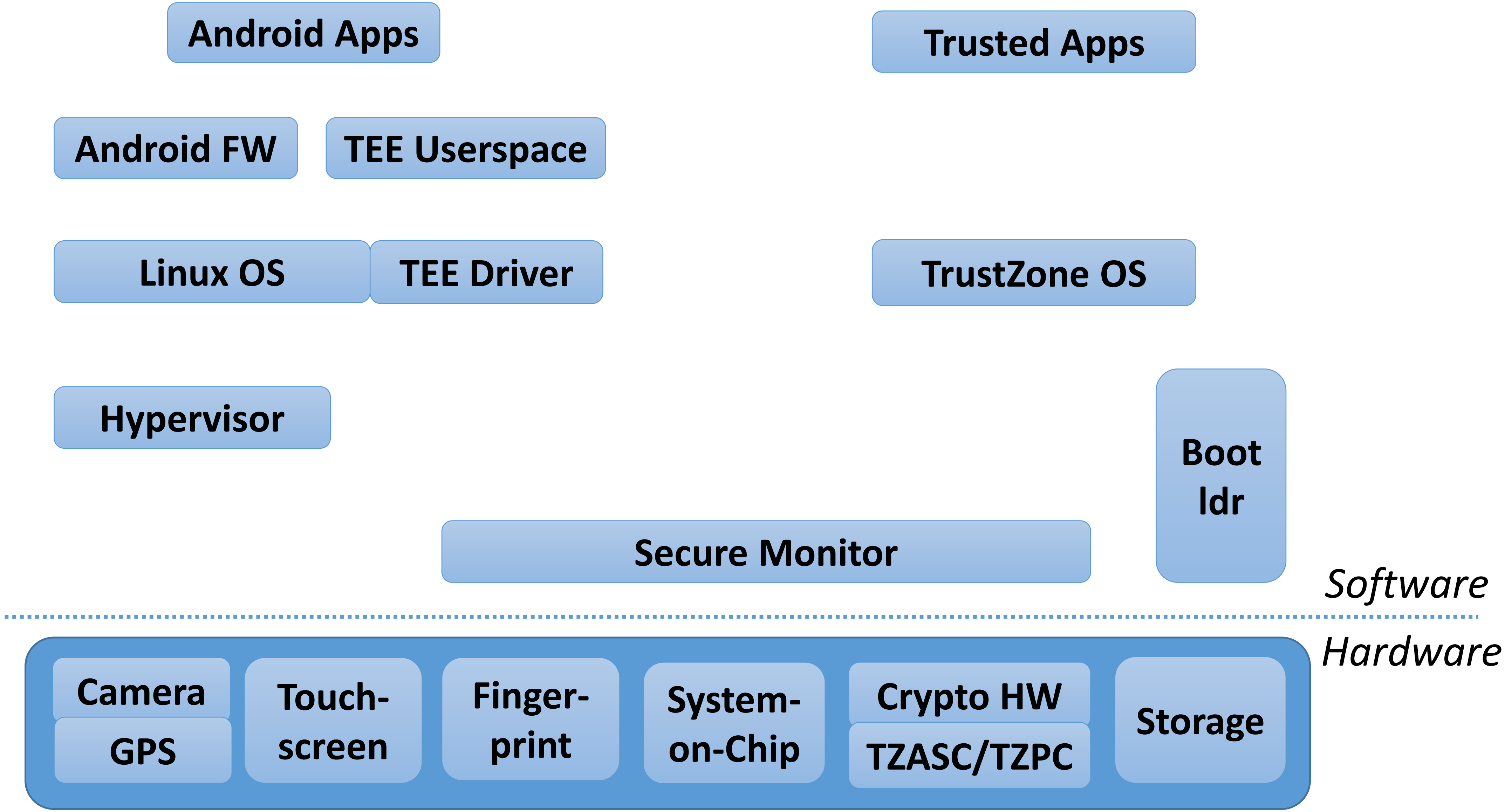
Challenge: Large Number of Components



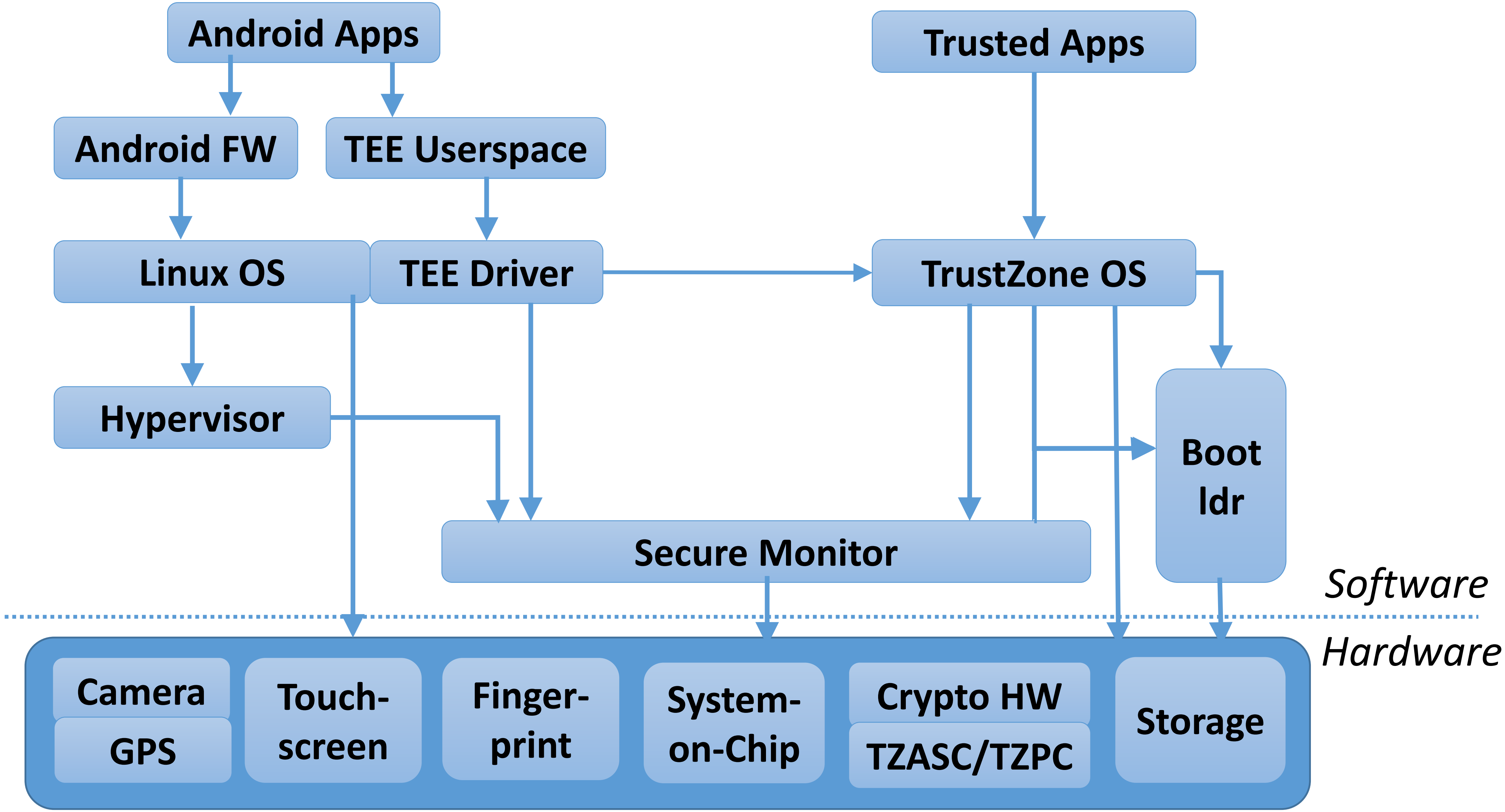
Challenge: Large Number of Components



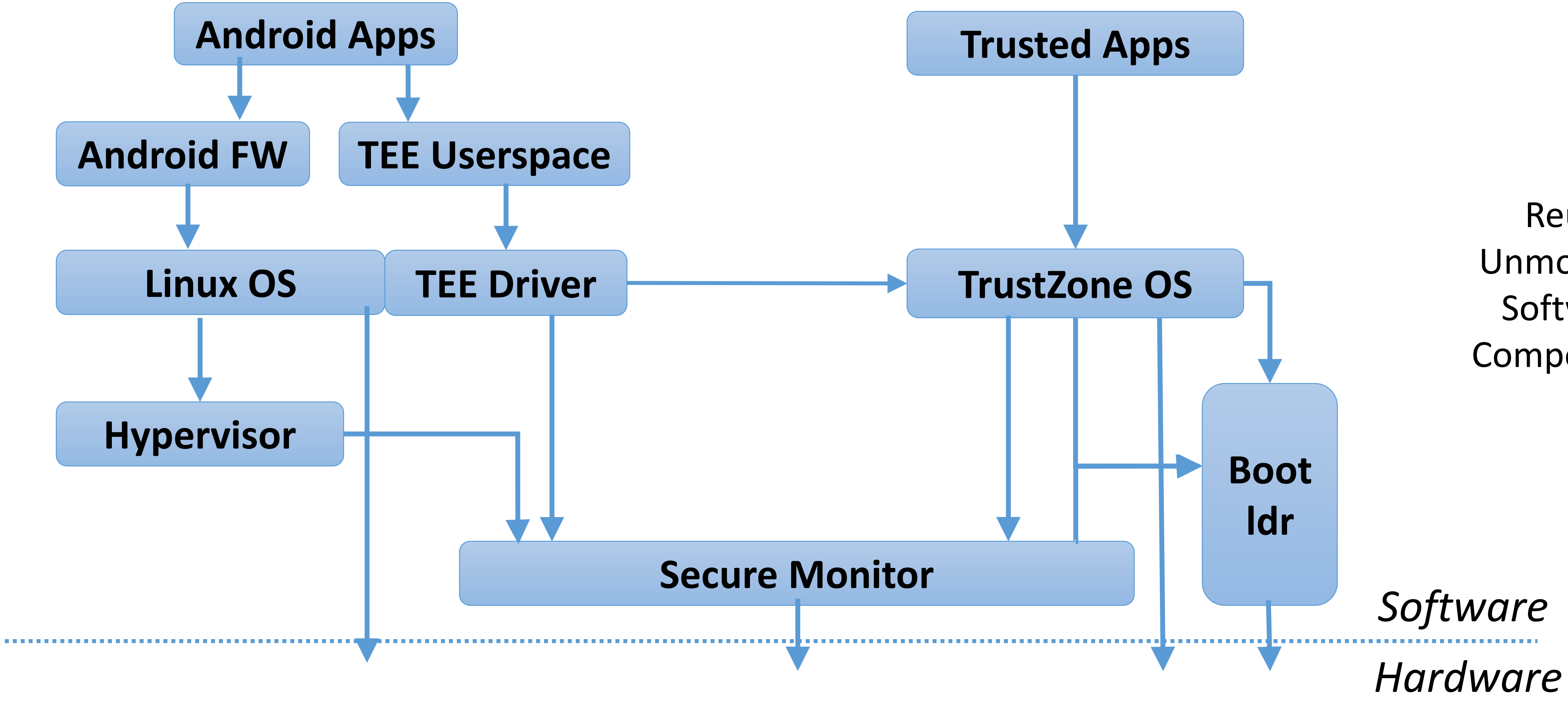
Challenge: Large Number of Components



Challenge: Large Number of Components

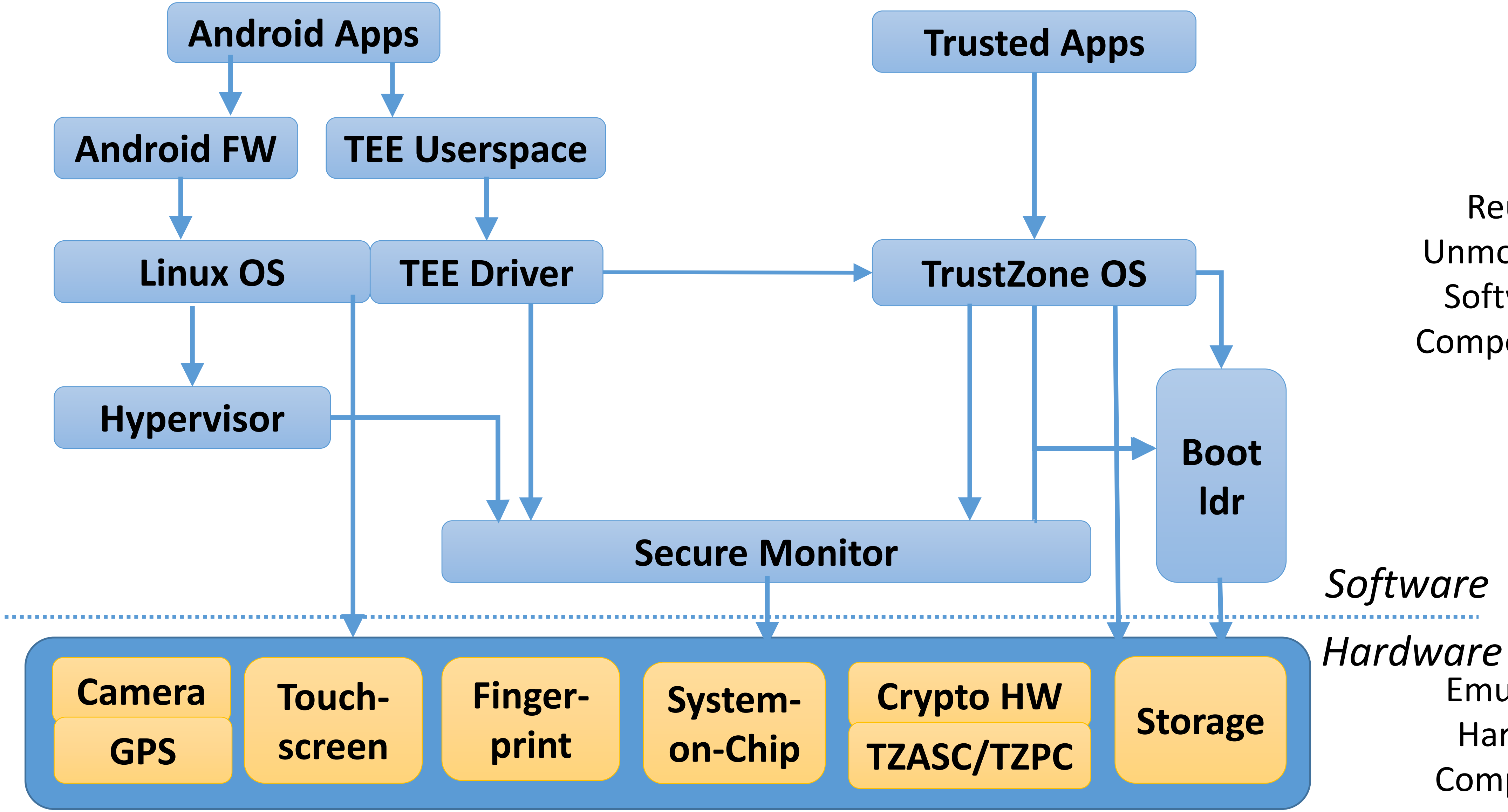


Traditional Approach: Emulate all HW



Reuse Unmodified Software Components

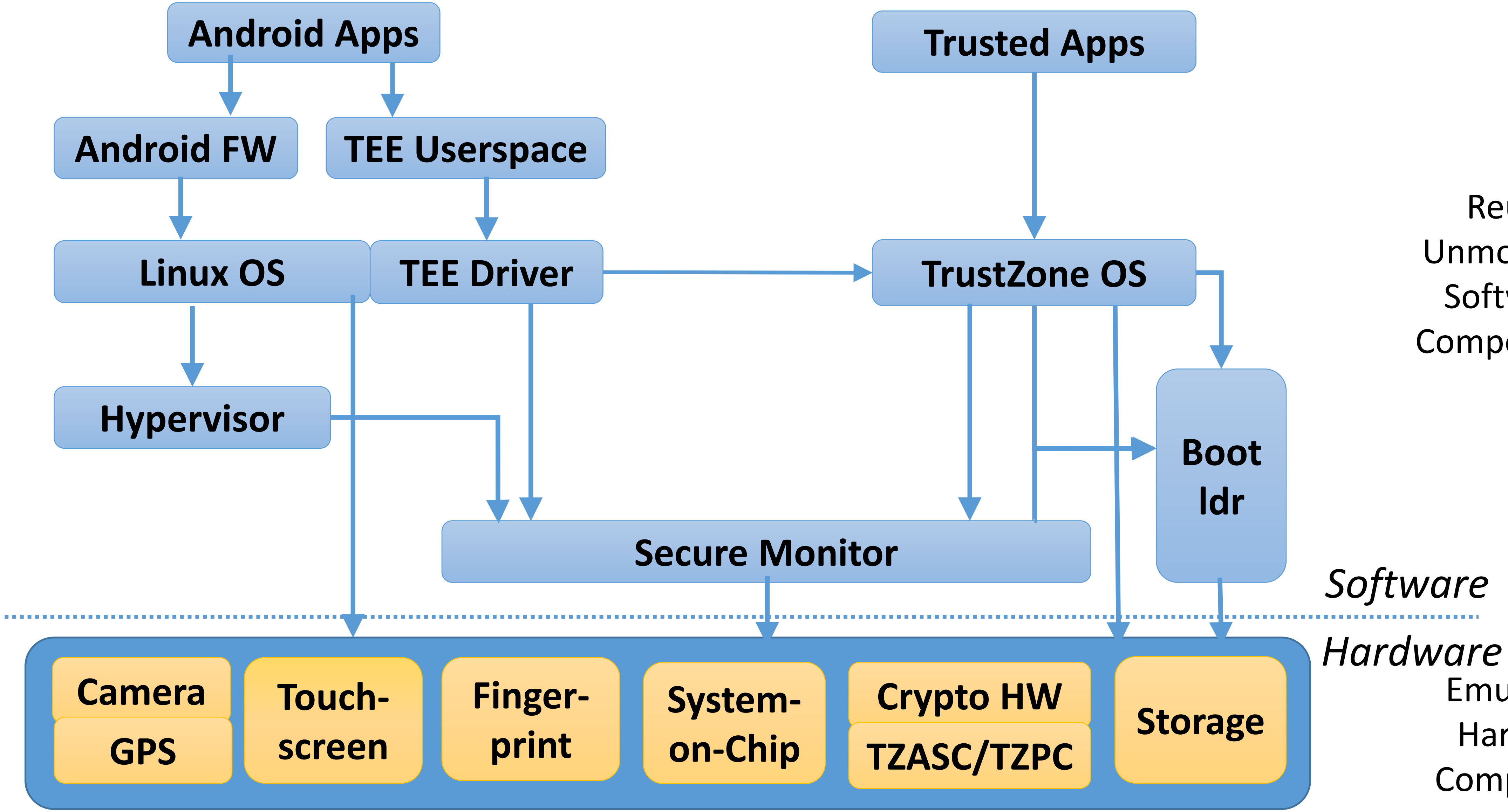
Traditional Approach: Emulate all HW



Reuse Unmodified Software Components

Emulate All Hardware Components

Traditional Approach: Emulate all HW



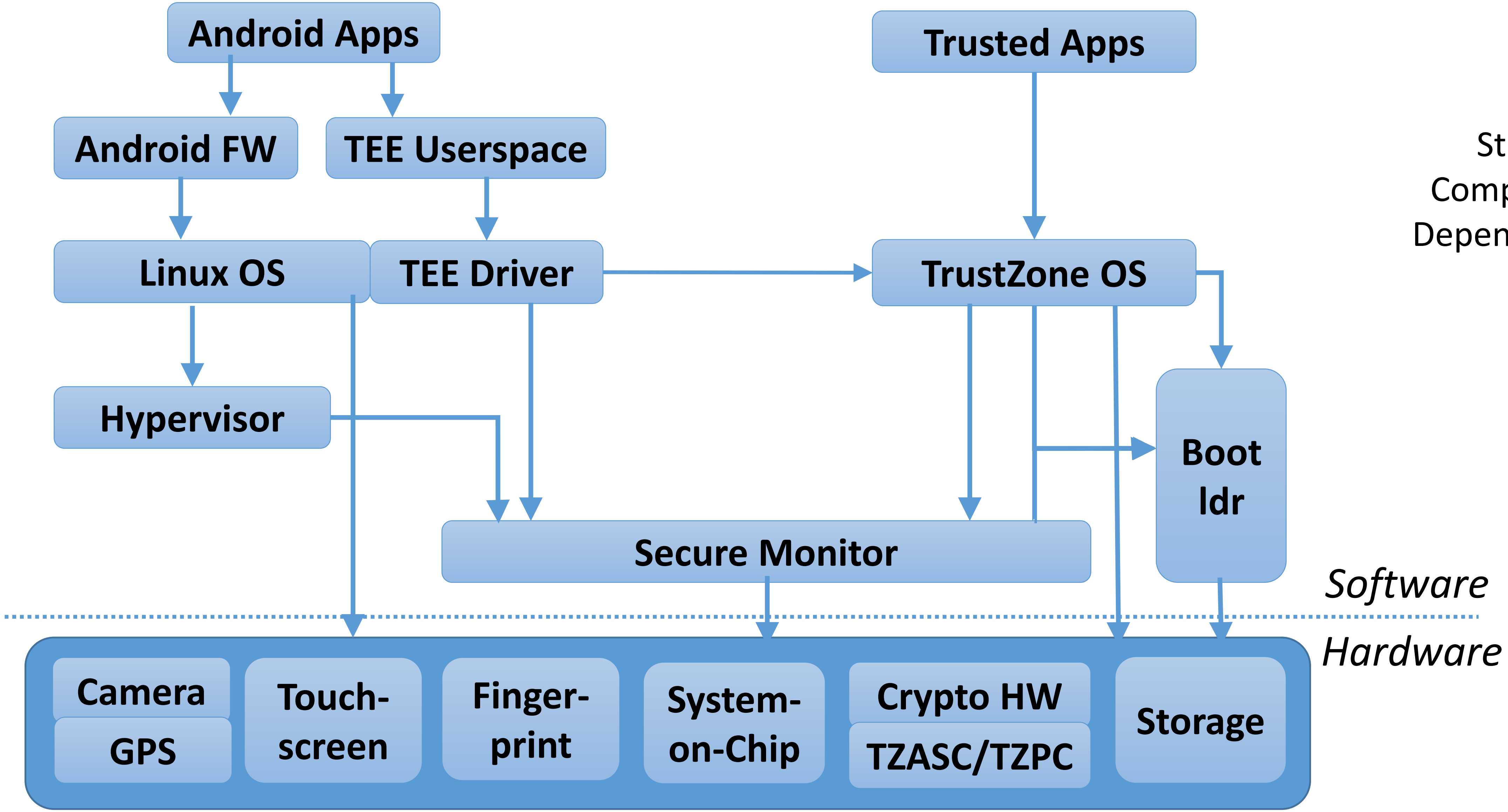
Reuse Unmodified Software Components

Emulate All Hardware Components

Impractical to emulate all hardware

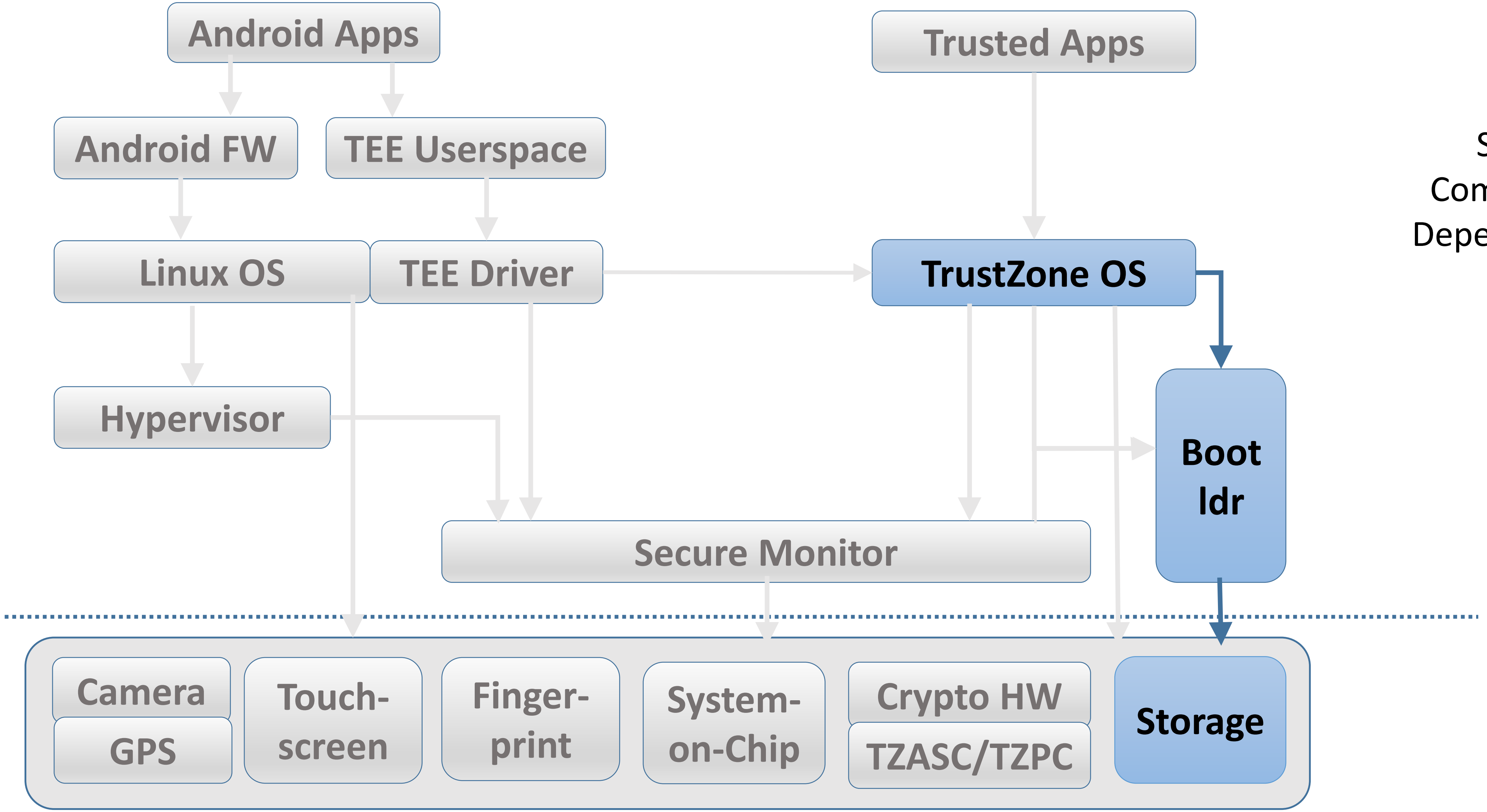
Our Approach: Emulate Subset of HW and SW

Our Approach: Emulate Subset of HW and SW



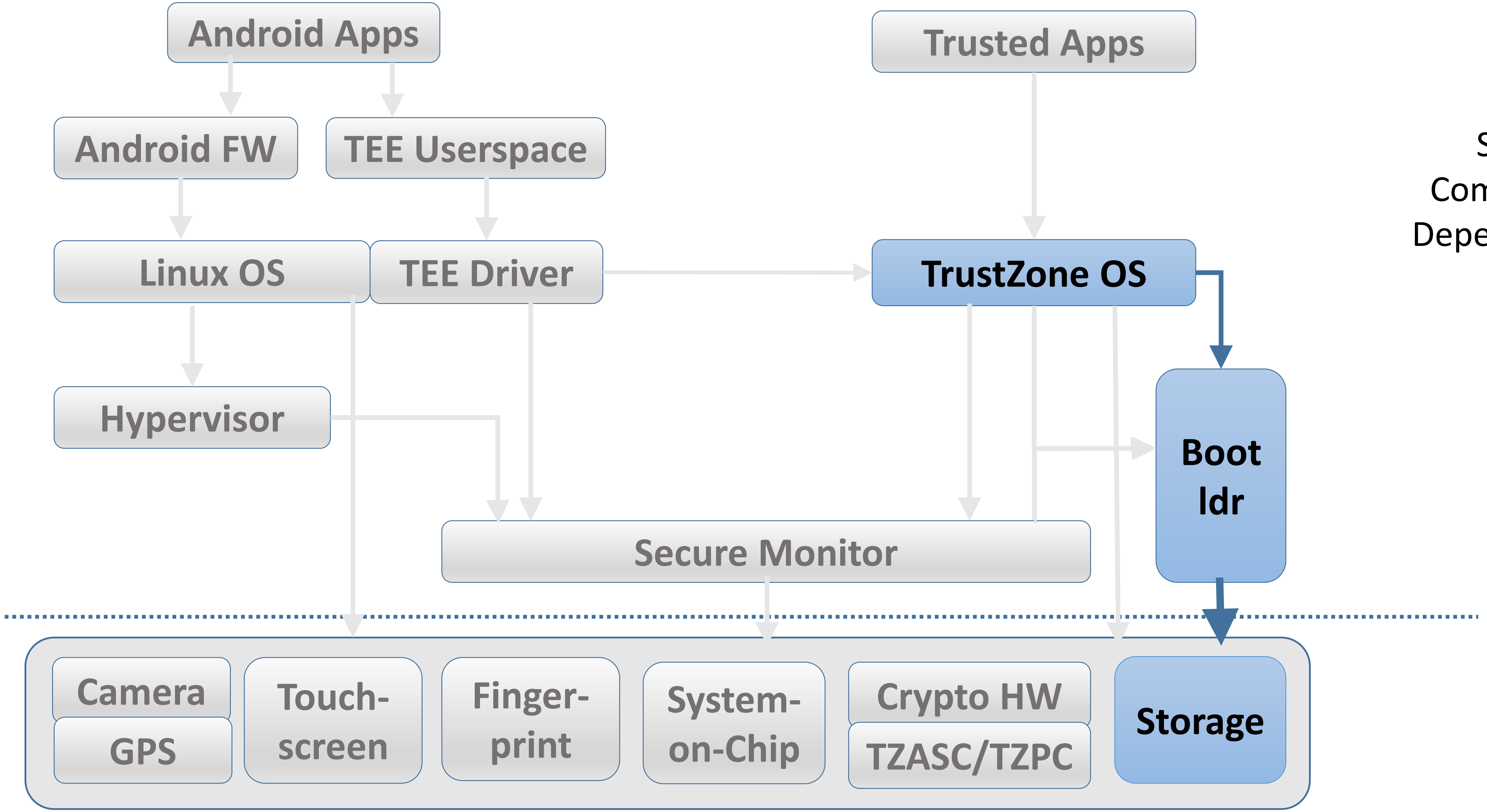
Study Component Dependencies

Our Approach: Emulate Subset of HW and SW



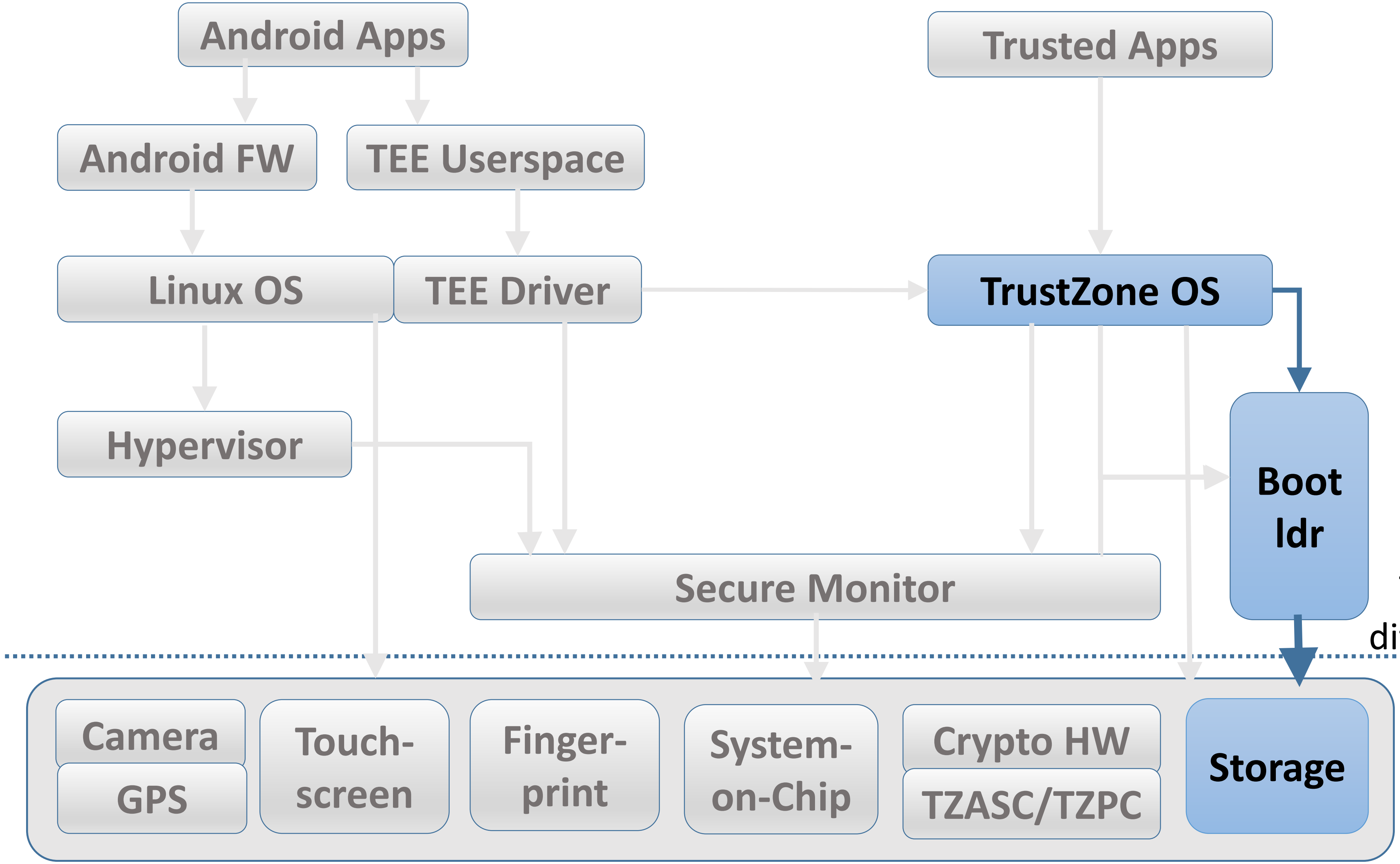
Study Component Dependencies

Our Approach: Emulate Subset of HW and SW



Study Component Dependencies

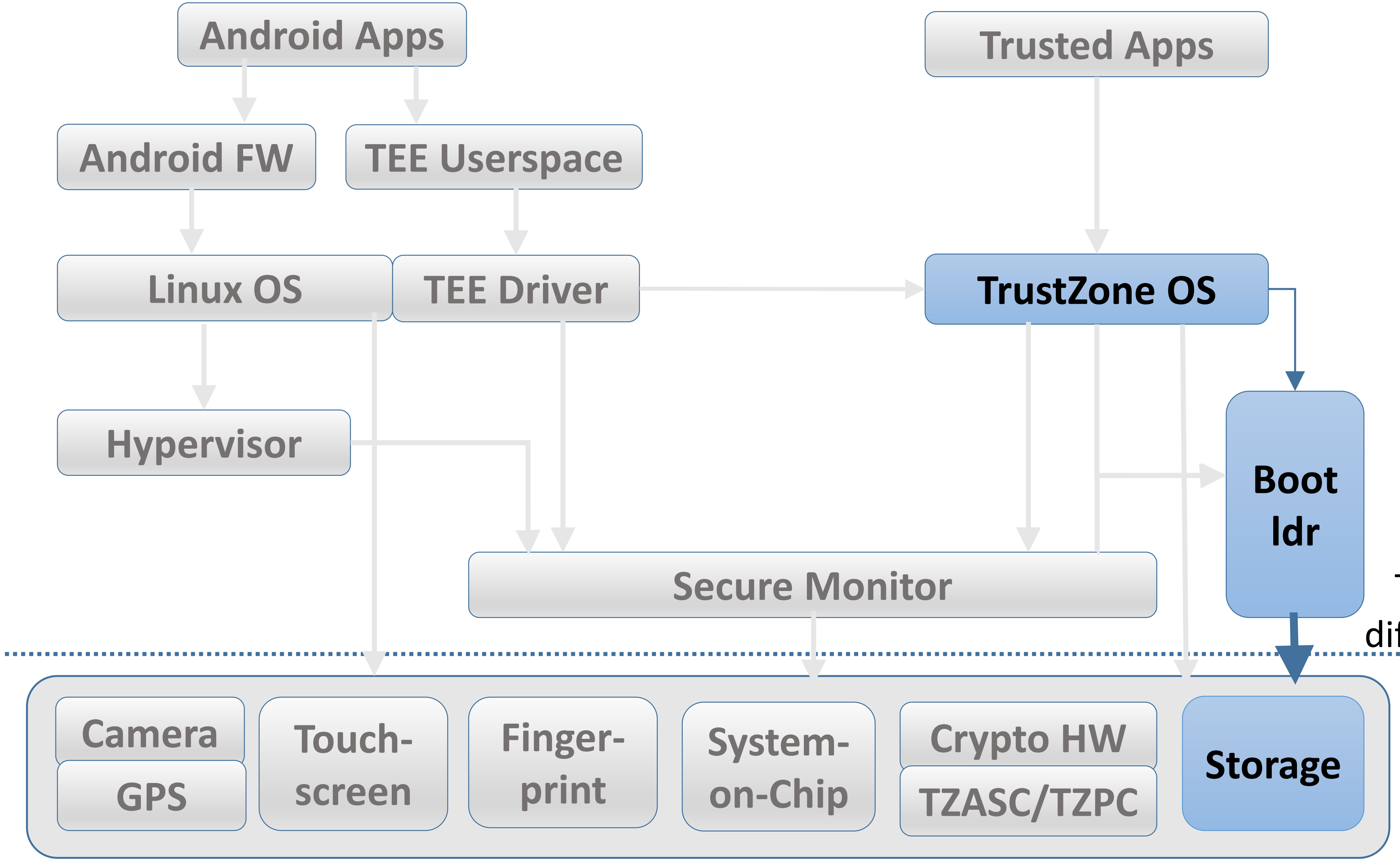
Our Approach: Emulate Subset of HW and SW



Study Component Dependencies

Tighty coupled, difficult to emulate dependency

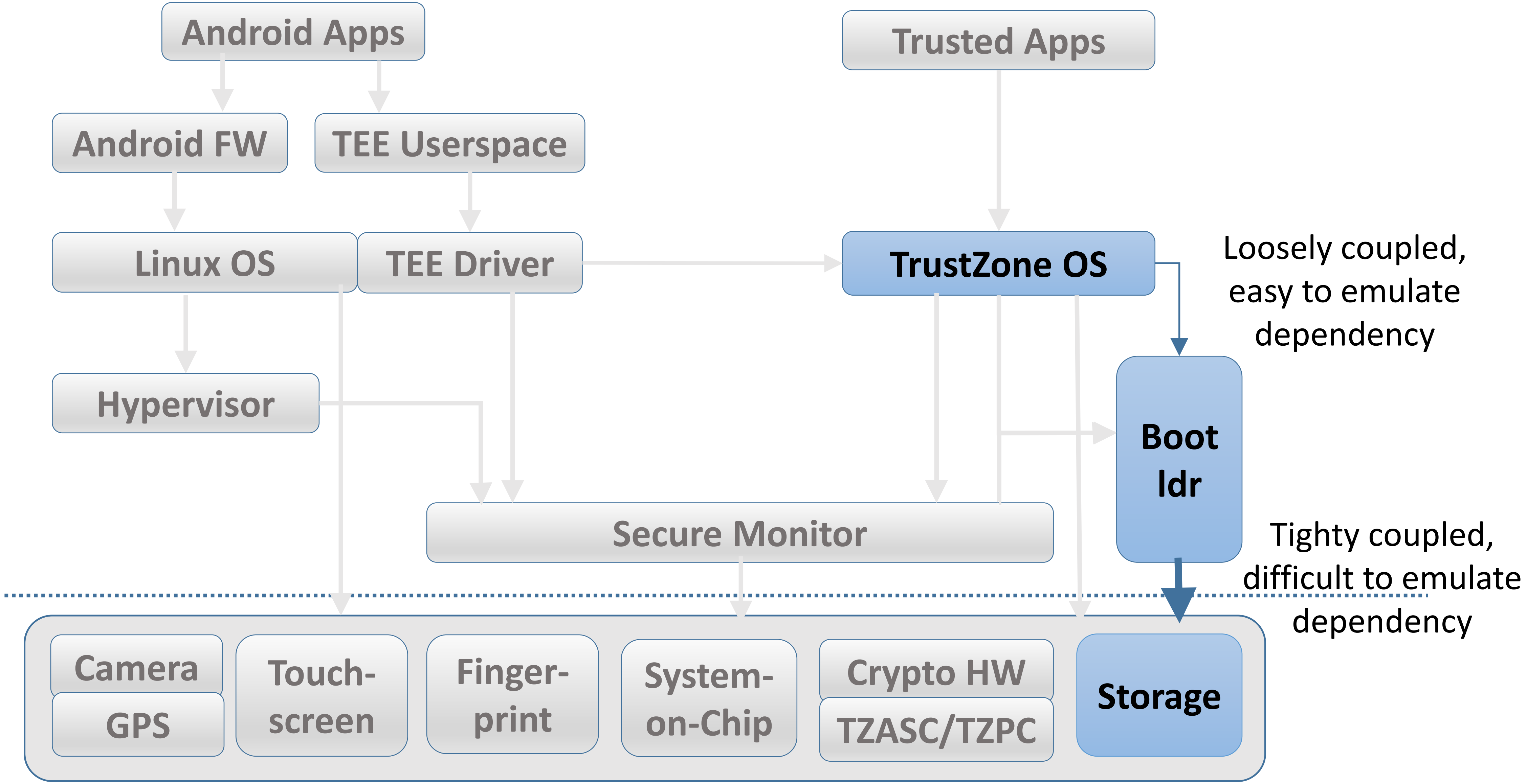
Our Approach: Emulate Subset of HW and SW



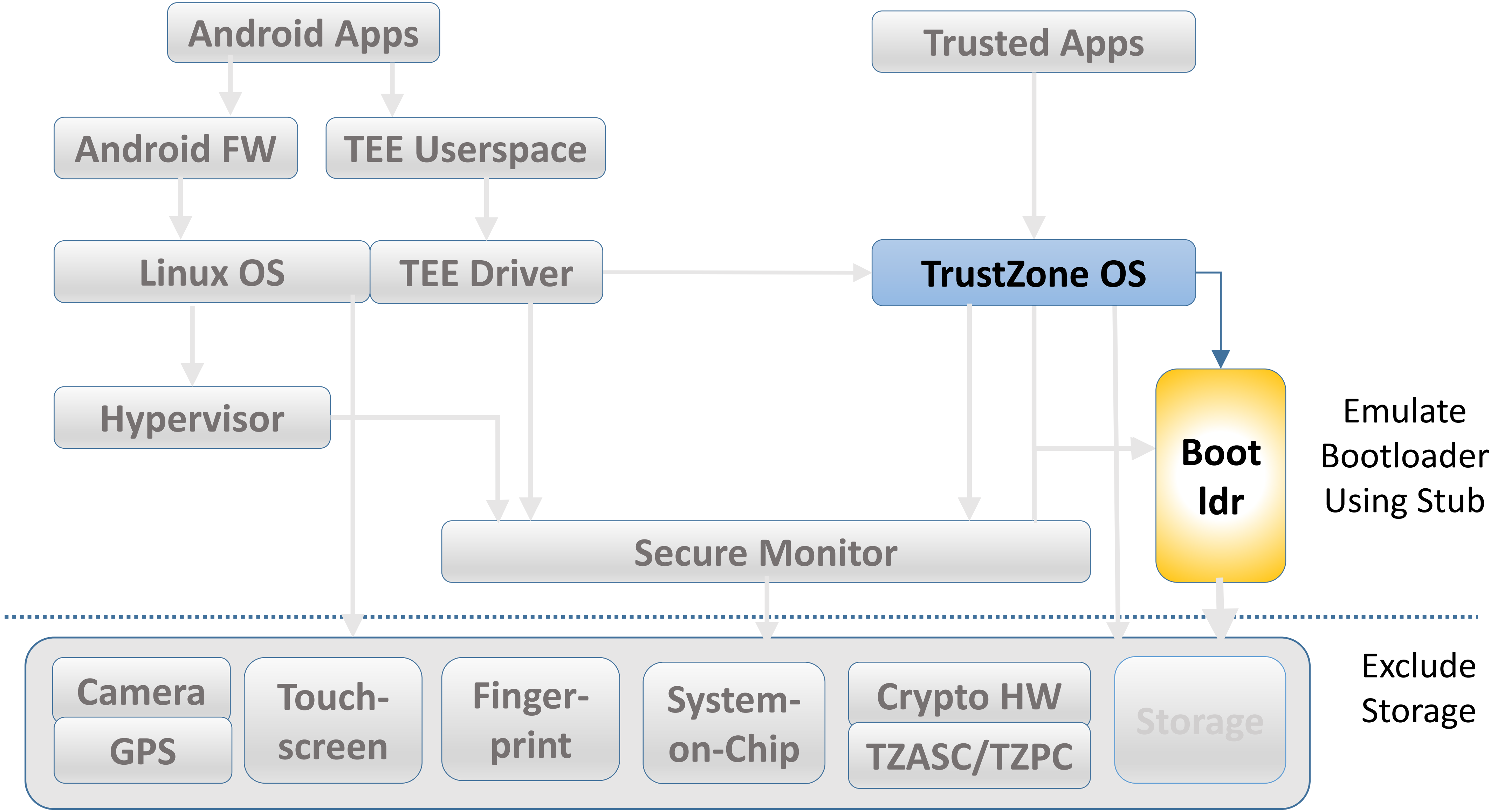
Study Component Dependencies

Tighty coupled, difficult to emulate dependency

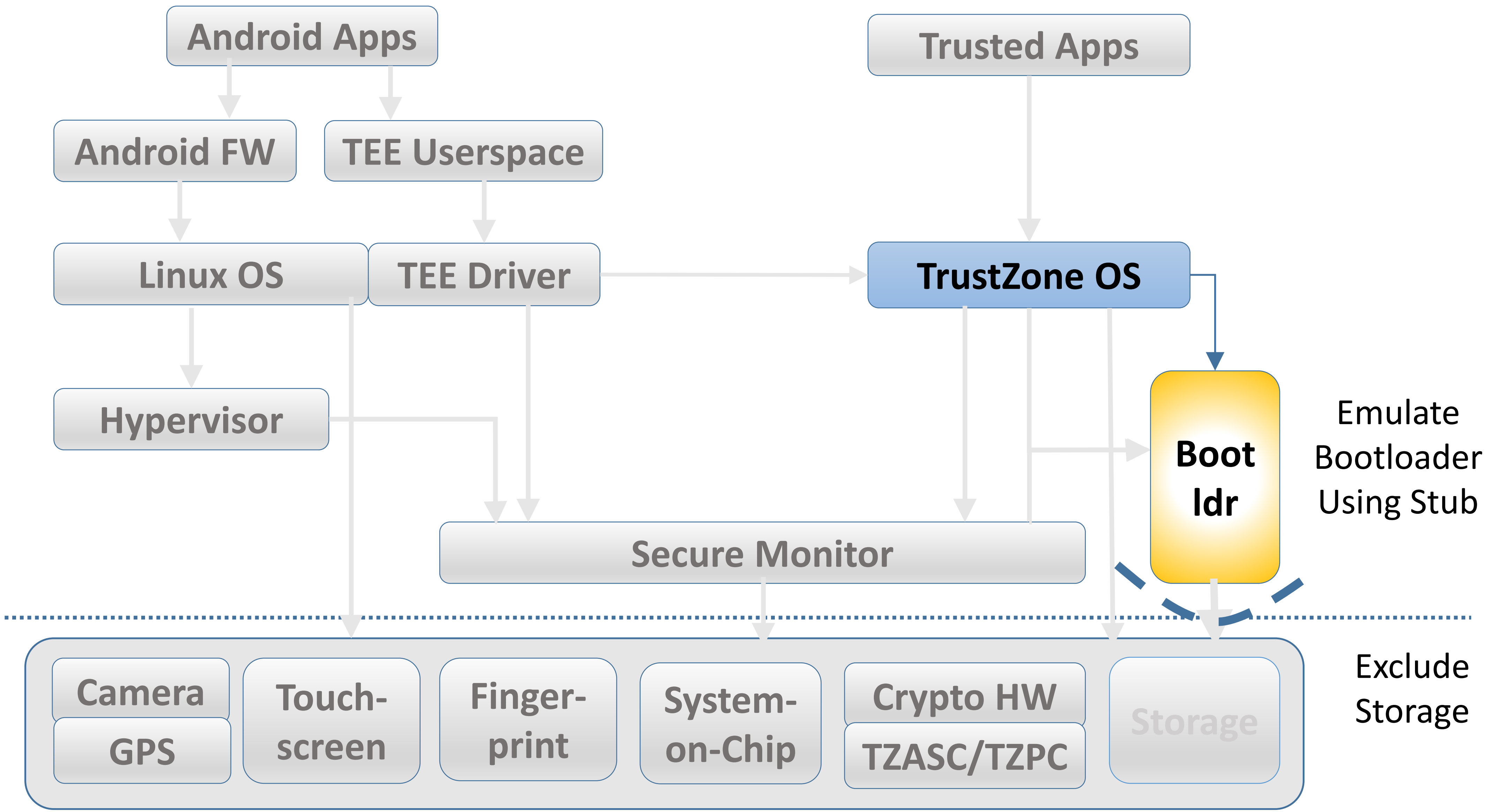
Our Approach: Emulate Subset of HW and SW



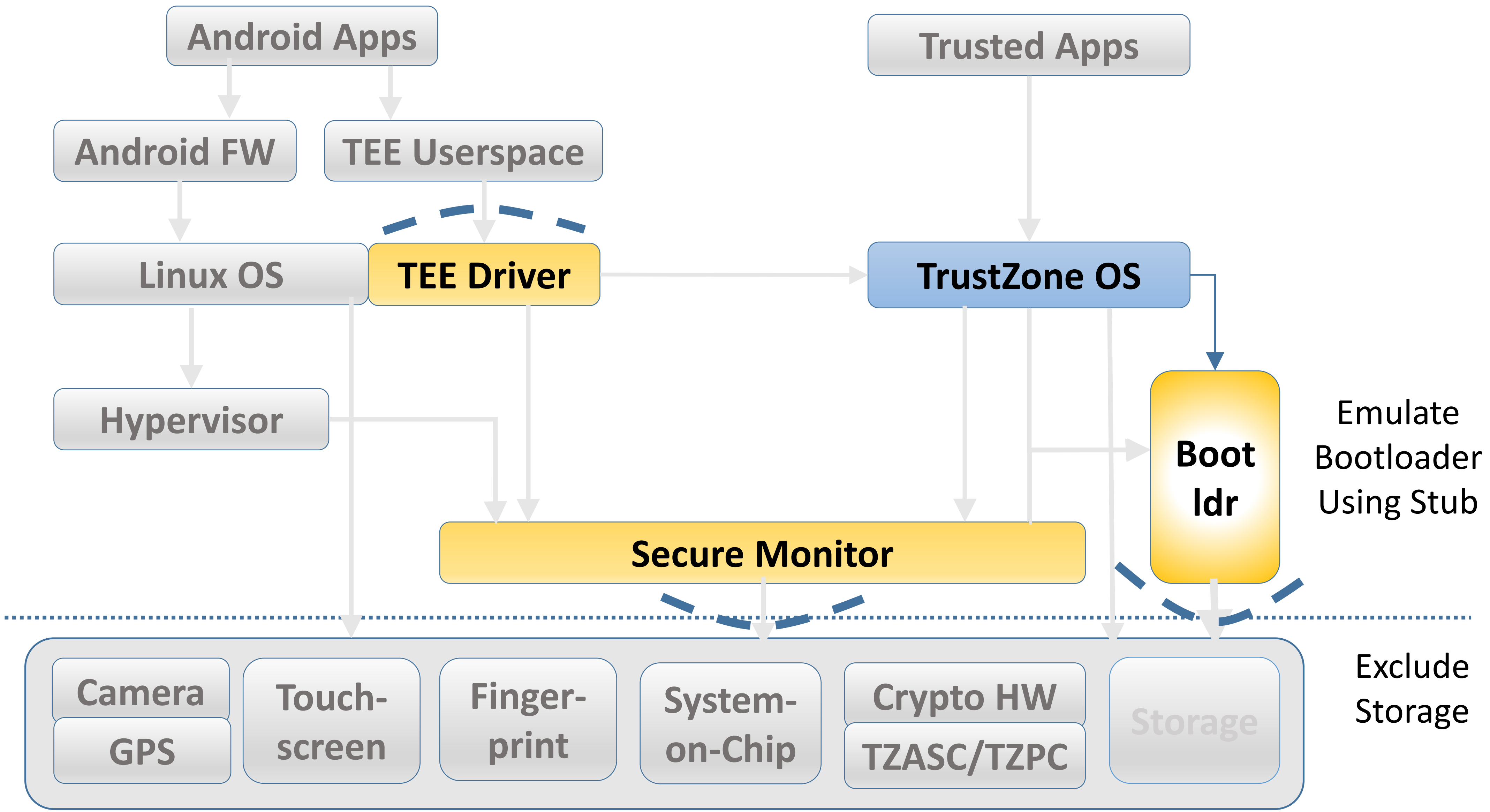
Our Approach: Emulate Subset of HW and SW



Our Approach: Emulate Subset of HW and SW



Our Approach: Emulate Subset of HW and SW



Emulation Effort Feasible Using Patterns

Emulation Effort Feasible Using Patterns

- Patterns to Emulate Hardware (MMIO Loads and Stores)

```
# Constant read (CONSTANT_READ_REG)
v = read(CONSTANT_READ_REG);
if (v != VALID_VALUE)
    fail();

# Read-write (READ_WRITE_REG)
write(READ_WRITE_REG, v1);
v2 = read(READ_WRITE_REG);
if (v2 != v1)
    fail();

# Increment (INCR_REG)
v = read(INCR_REG);
if (read(INCR_REG) < v)
    fail();

# Poll (POLL_REG)
while (read(POLL_REG) != READY);
```

```
# Random (RAND_REG)
v1 = read(RAND_REG)
v2 = read(RAND_REG)
if (v1 == v2)
    fail();

# Shadow (SHADOW_REG1, SHADOW_REG2)
# Commit (COMMIT_REG)
# Target (TARGET_REG1, TARGET_REG2)
write(SHADOW_REG1, v1)
write(SHADOW_REG2, v2)
write(COMMIT_REG, COMMIT_VALUE)
v3 = read(TARGET_REG1)
v4 = read(TARGET_REG2)
if ((v1 != v3) or (v2 != v4))
    fail();
```

Emulation Effort Feasible Using Patterns

- Patterns to Emulate Hardware (MMIO Loads and Stores)

```
# Constant read (CONSTANT_READ_REG)
v = read(CONSTANT_READ_REG);
if (v != VALID_VALUE)
    fail();

# Read-write (READ_WRITE_REG)
write(READ_WRITE_REG, v1);
v2 = read(READ_WRITE_REG);
if (v2 != v1)
    fail();

# Increment (INCR_REG)
v = read(INCR_REG);
if (read(INCR_REG) < v)
    fail();

# Poll (POLL_REG)
while (read(POLL_REG) != READY);
```

```
# Random (RAND_REG)
v1 = read(RAND_REG)
v2 = read(RAND_REG)
if (v1 == v2)
    fail();

# Shadow (SHADOW_REG1, SHADOW_REG2)
# Commit (COMMIT_REG)
# Target (TARGET_REG1, TARGET_REG2)
write(SHADOW_REG1, v1)
write(SHADOW_REG2, v2)
write(COMMIT_REG, COMMIT_VALUE)
v3 = read(TARGET_REG1)
v4 = read(TARGET_REG2)
if ((v1 != v3) or (v2 != v4))
    fail();
```

Emulation Effort Feasible Using Patterns

- Patterns to Emulate Hardware (MMIO Loads and Stores)

```
# Constant read (CONSTANT_READ_REG)
v = read(CONSTANT_READ_REG);
if (v != VALID_VALUE)
    fail();

# Read-write (READ_WRITE_REG)
write(READ_WRITE_REG, v1);
v2 = read(READ_WRITE_REG);
if (v2 != v1)
    fail();

# Increment (INCR_REG)
v = read(INCR_REG);
if (read(INCR_REG) < v)
    fail();

# Poll (POLL_REG)
while (read(POLL_REG) != READY);
```

```
# Random (RAND_REG)
v1 = read(RAND_REG)
v2 = read(RAND_REG)
if (v1 == v2)
    fail();

# Shadow (SHADOW_REG1, SHADOW_REG2)
# Commit (COMMIT_REG)
# Target (TARGET_REG1, TARGET_REG2)
write(SHADOW_REG1, v1)
write(SHADOW_REG2, v2)
write(COMMIT_REG, COMMIT_VALUE)
v3 = read(TARGET_REG1)
v4 = read(TARGET_REG2)
if ((v1 != v3) or (v2 != v4))
    fail();
```

Emulation Effort Feasible Using Patterns

- Patterns to Emulate Software APIs

Category	Difficulty	K	Q	T	O
Emulated Boot Information Structure					
Constants	Low	13	8	2	3
Any value	Low	1	3	0	0
Simple value	Low	2	1	14	2
Complex values	High	2	1[note a]	0	0
Total	-	**18**	**13**	**16**	**5**
Emulated Secure Monitor Calls[note b]					
Return simple value	Low	0	-	3	-
Return constant	Low	1	-	5	-
Store/retrieve values	Low	1	-	2	-
Control transfer	High	3	-	2	-
Total	-	**5**	**-**	**12**	**-**

Emulation Effort Feasible Using Patterns

- Patterns to Emulate Software APIs

Category	**Difficulty**	**K**	**Q**	**T**	**O**
		Emulated Boot Information Structure			
Constants	Low	13	8	2	3
Any value	Low	1	3	0	0
Simple value	Low	2	1	14	2
Complex values	High	2	1 [note a]	0	0
Total	-	**18**	**13**	**16**	**5**
		Emulated Secure Monitor Calls [note b]			
Return simple value	Low	0	-	3	-
Return constant	Low	1	-	5	-
Store/retrieve values	Low	1	-	2	-
Control transfer	High	3	-	2	-
Total	-	**5**	**-**	**12**	**-**

Emulation Effort Feasible Using Patterns

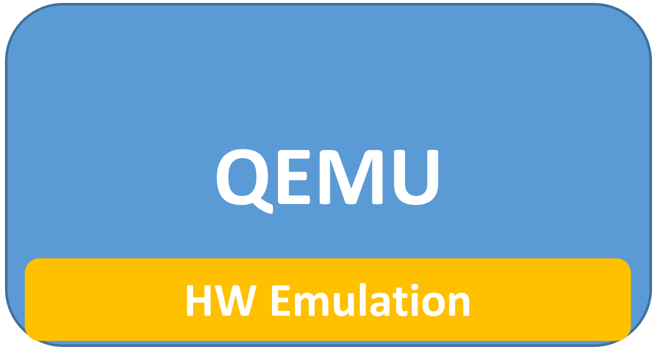
- Patterns to Emulate Software APIs

Category	**Difficulty**	**K**	**Q**	**T**	**O**
Emulated Boot Information Structure					
Constants	Low	13	8	2	3
Any value	Low	1	3	0	0
Simple value	Low	2	1	14	2
Complex values	High	2	1 [note a]	0	0
Total	-	**18**	**13**	**16**	**5**
Emulated Secure Monitor Calls [note b]					
Return simple value	Low	0	-	3	-
Return constant	Low	1	-	5	-
Store/retrieve values	Low	1	-	2	-
Control transfer	High	3	-	2	-
Total	-	**5**	**-**	**12**	**-**

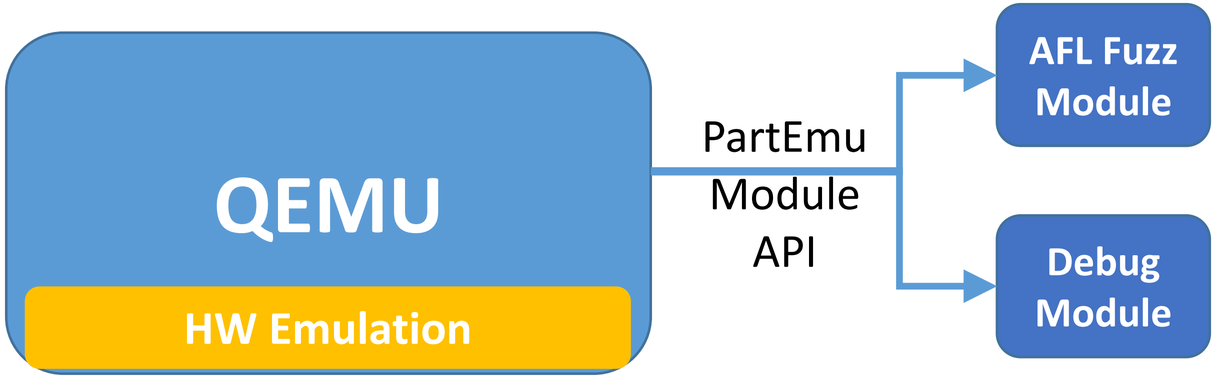
Implementation

QEMU

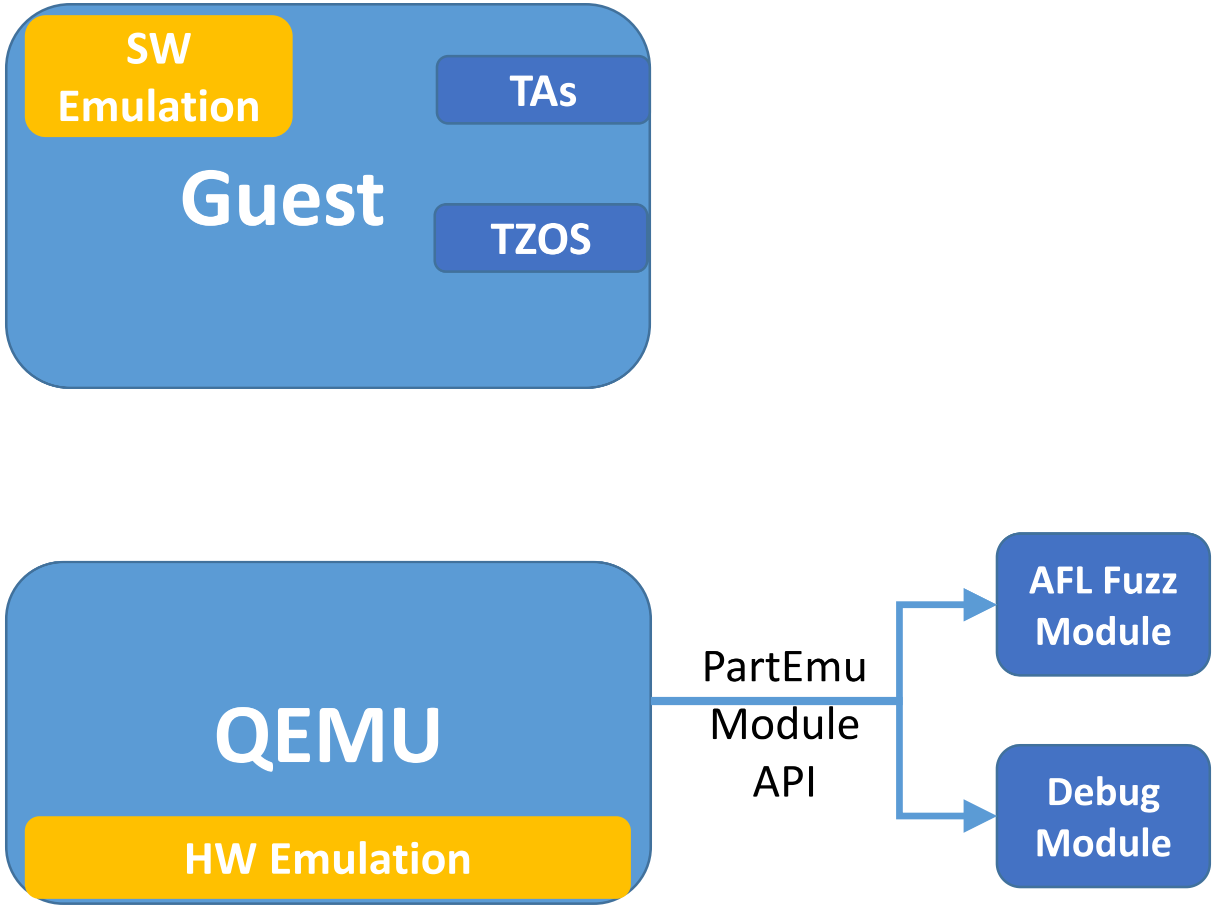
Implementation



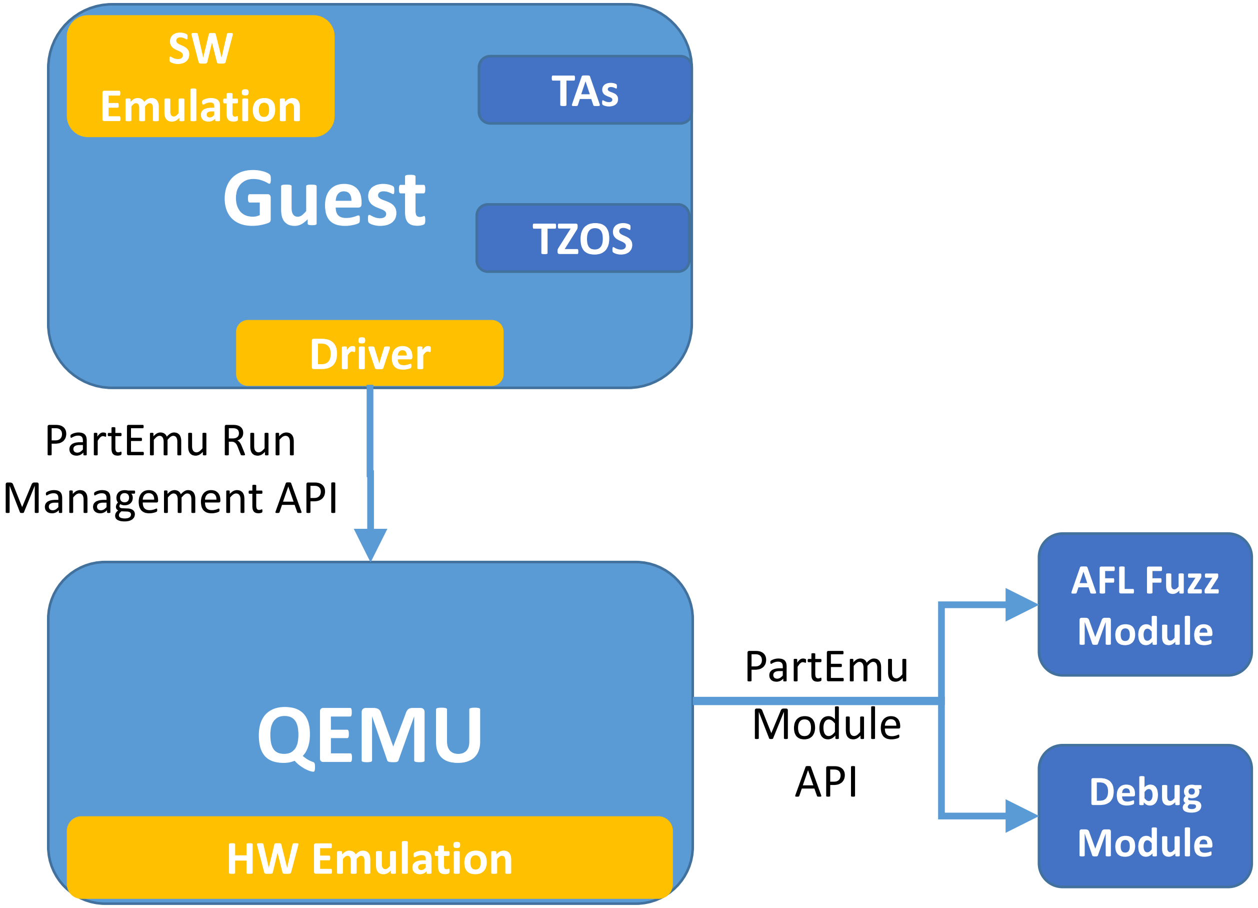
Implementation



Implementation



Implementation



Problem: Dynamic analysis of TZ is hard!

Approach: How did we run TZ in an emulator?

Results: What did we learn?

Fuzz Testing TAs Using AFL

16 Firmware Images

Fuzz Testing TAs Using AFL

16 Firmware Images

12 Smartphone / IoT vendors

Fuzz Testing TAs Using AFL

16 Firmware Images

12 Smartphone / IoT vendors

196 Unique TAs

Fuzz Testing TAs Using AFL

16 Firmware Images

12 Smartphone / IoT vendors

196 Unique TAs

AFL Crashed **48** TAs

Fuzz Testing TAs Using AFL

16 Firmware Images	**12** Smartphone / IoT vendors	**196** Unique TAs	AFL Crashed **48** TAs

- Found TZ-specific coding anti-patterns that led to crashes

Anti-Pattern 1:
Assumptions about Request Sequence

Anti-Pattern 1: Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

Anti-Pattern 1: Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

```
char *ptr = NULL; // global
…
switch (request) {
case INIT:
        init(ptr);
        break;
case DO_ACTION:
        do_action(ptr);
        break;
case UNINIT:
        uninit(ptr);
        break;
};
```

Anti-Pattern 1: Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

1.

```
char *ptr = NULL; // global
…
switch (request) {
case INIT:
        init(ptr);
        break;
case DO_ACTION:
        do_action(ptr);
        break;
case UNINIT:
        uninit(ptr);
        break;
};
```

Anti-Pattern 1: Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

```
        char *ptr = NULL; // global
        …
        switch (request) {
1.      case INIT:
                init(ptr);
                break;
2.      case DO_ACTION:
                do_action(ptr);
                break;
        case UNINIT:
                uninit(ptr);
                break;
        };
```

Anti-Pattern 1: Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

1.

2.

3.

```
char *ptr = NULL; // global
…
switch (request) {
case INIT:
        init(ptr);
        break;
case DO_ACTION:
        do_action(ptr);
        break;
case UNINIT:
        uninit(ptr);
        break;
};
```

Anti-Pattern 1:
Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

1.

2.

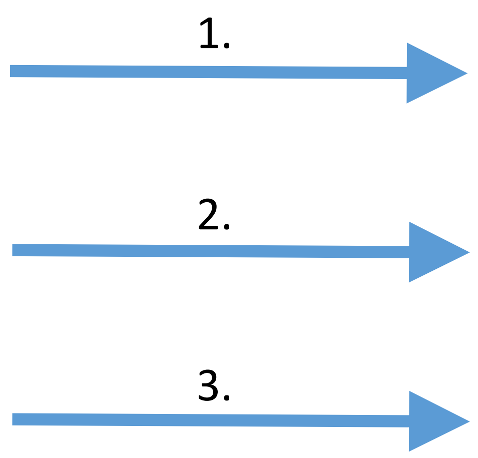
3.

1.

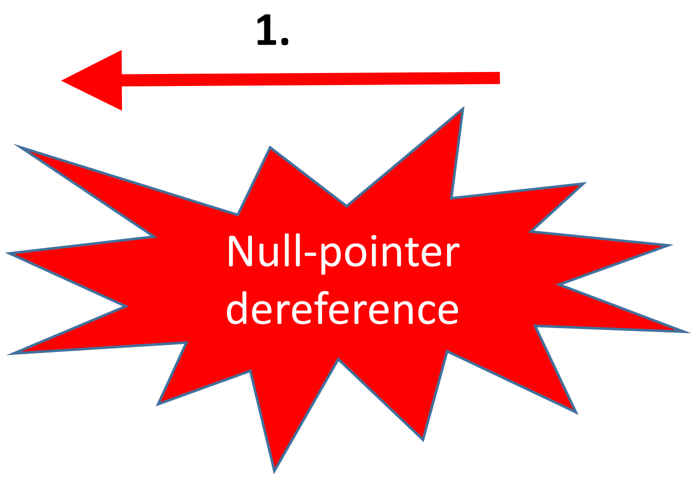
```
char *ptr = NULL; // global
…
switch (request) {
case INIT:
        init(ptr);
        break;
case DO_ACTION:
        do_action(ptr);
        break;
case UNINIT:
        uninit(ptr);
        break;
};
```

Anti-Pattern 1:
Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests




```
char *ptr = NULL; // global
…
switch (request) {
case INIT:
        init(ptr);
        break;
case DO_ACTION:
        do_action(ptr);
        break;
case UNINIT:
        uninit(ptr);
        break;
};
```



Anti-Pattern 1:
Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

```
char *ptr = NULL; // global
…
switch (request) {
case INIT:
        init(ptr);
        break;
case DO_ACTION:
        do_action(ptr);
        break;
case UNINIT:
        uninit(ptr);
        break;
};
```

1.

2.

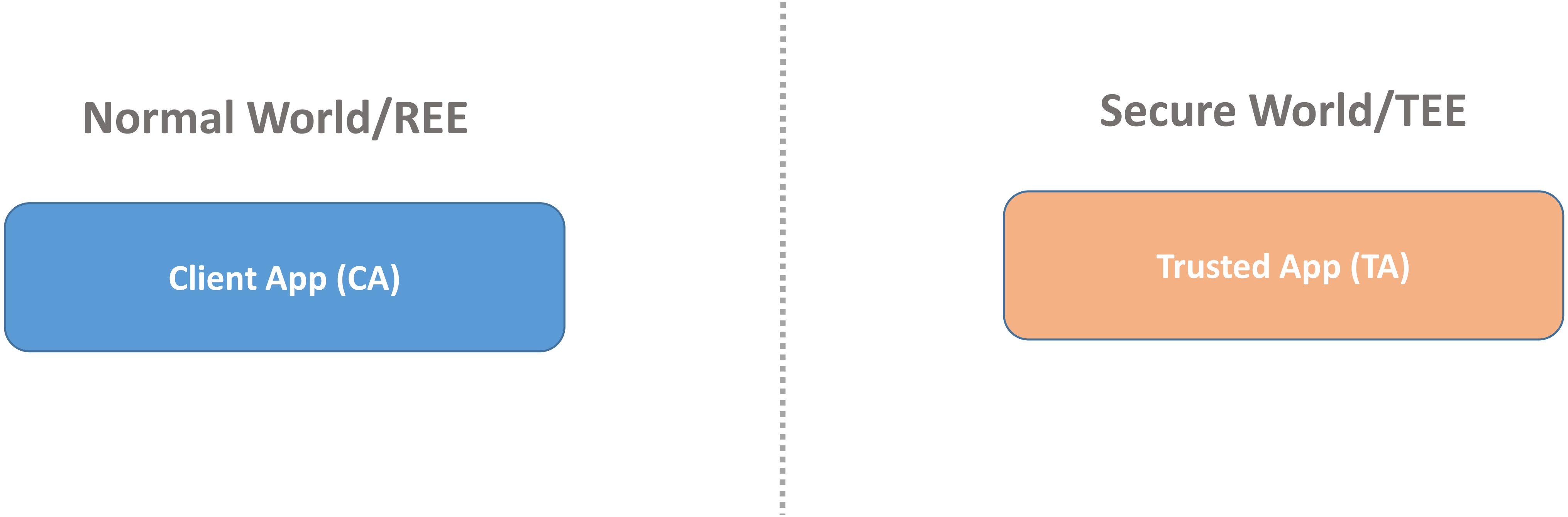
3.

1.

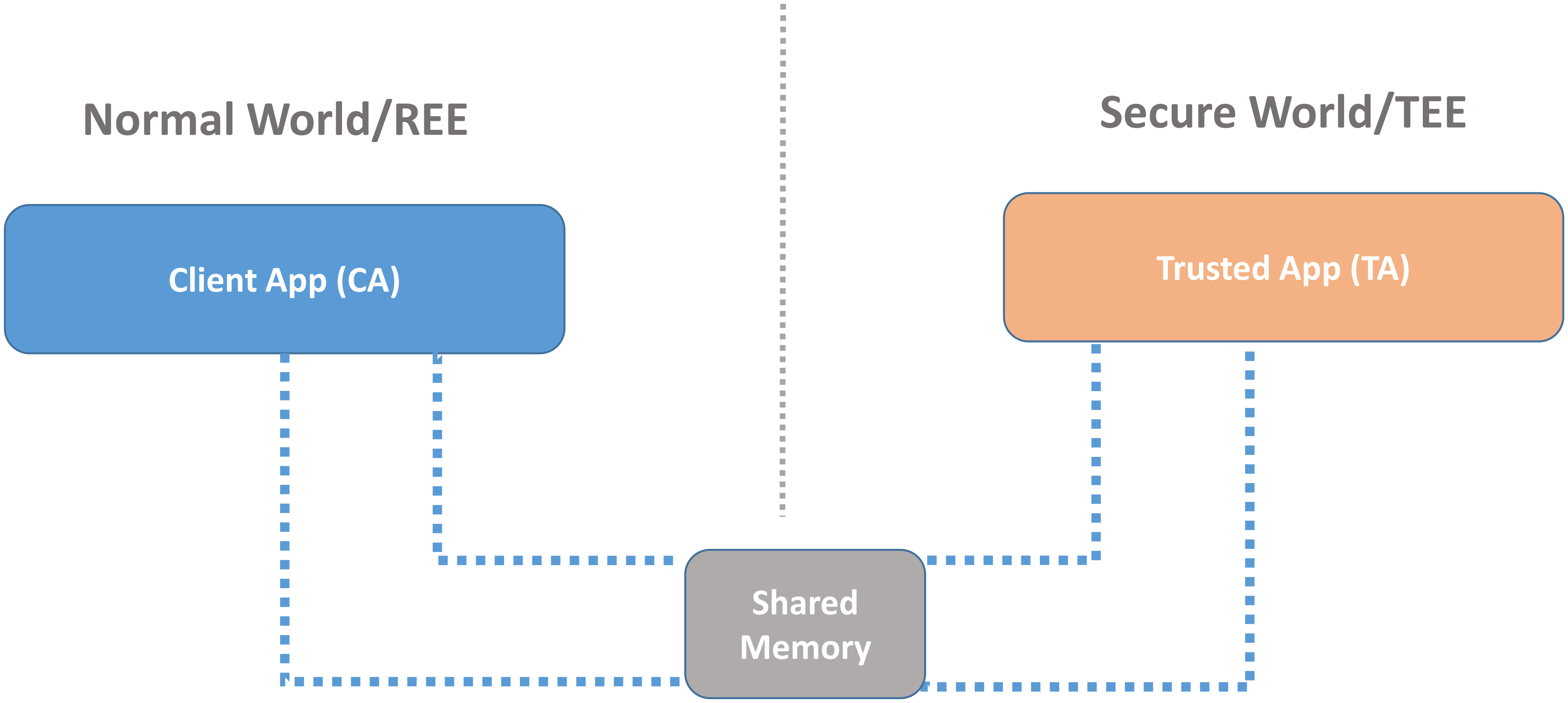
Null-pointer dereference

TA should properly handle any sequence of requests from CA

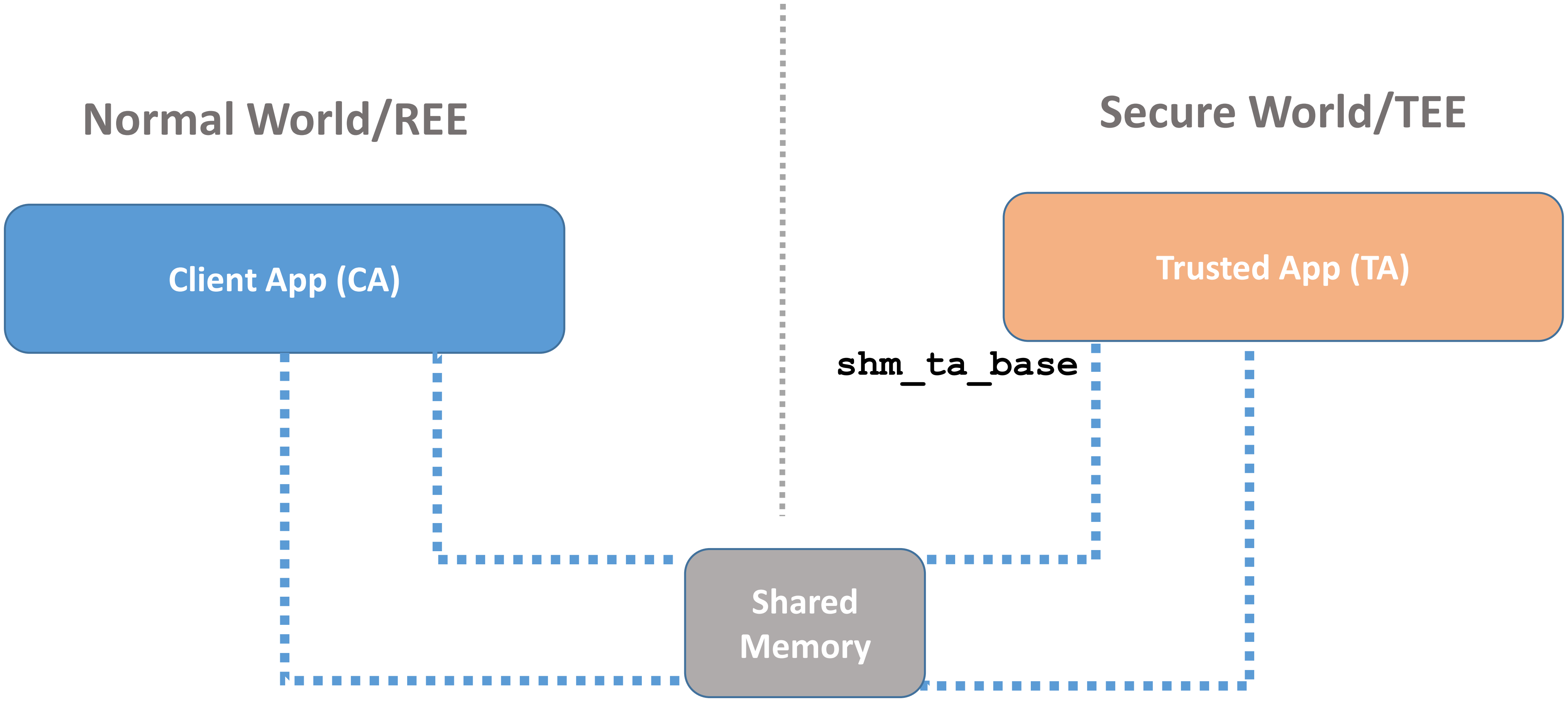
Anti-Pattern 2: Unvalidated Normal-World Pointers



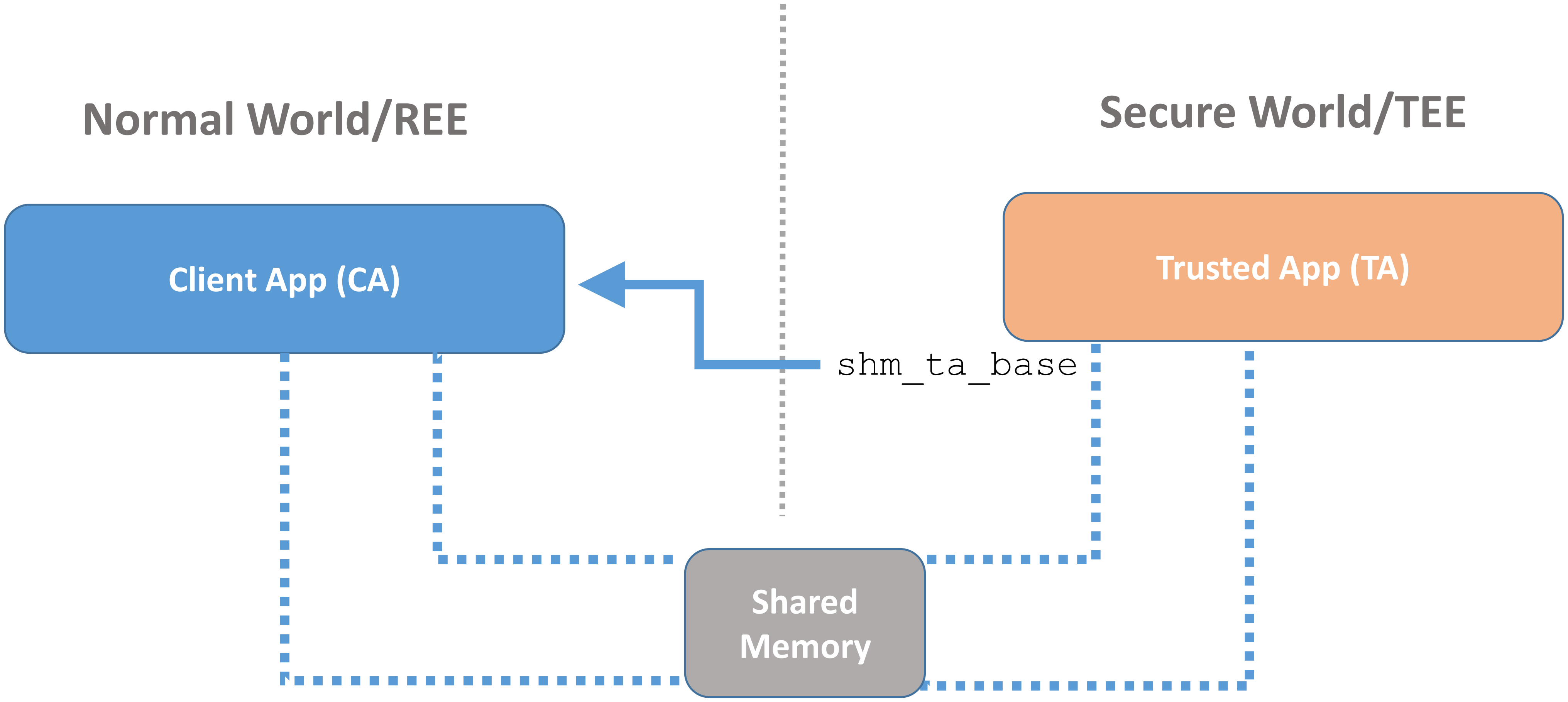
Anti-Pattern 2: Unvalidated Normal-World Pointers



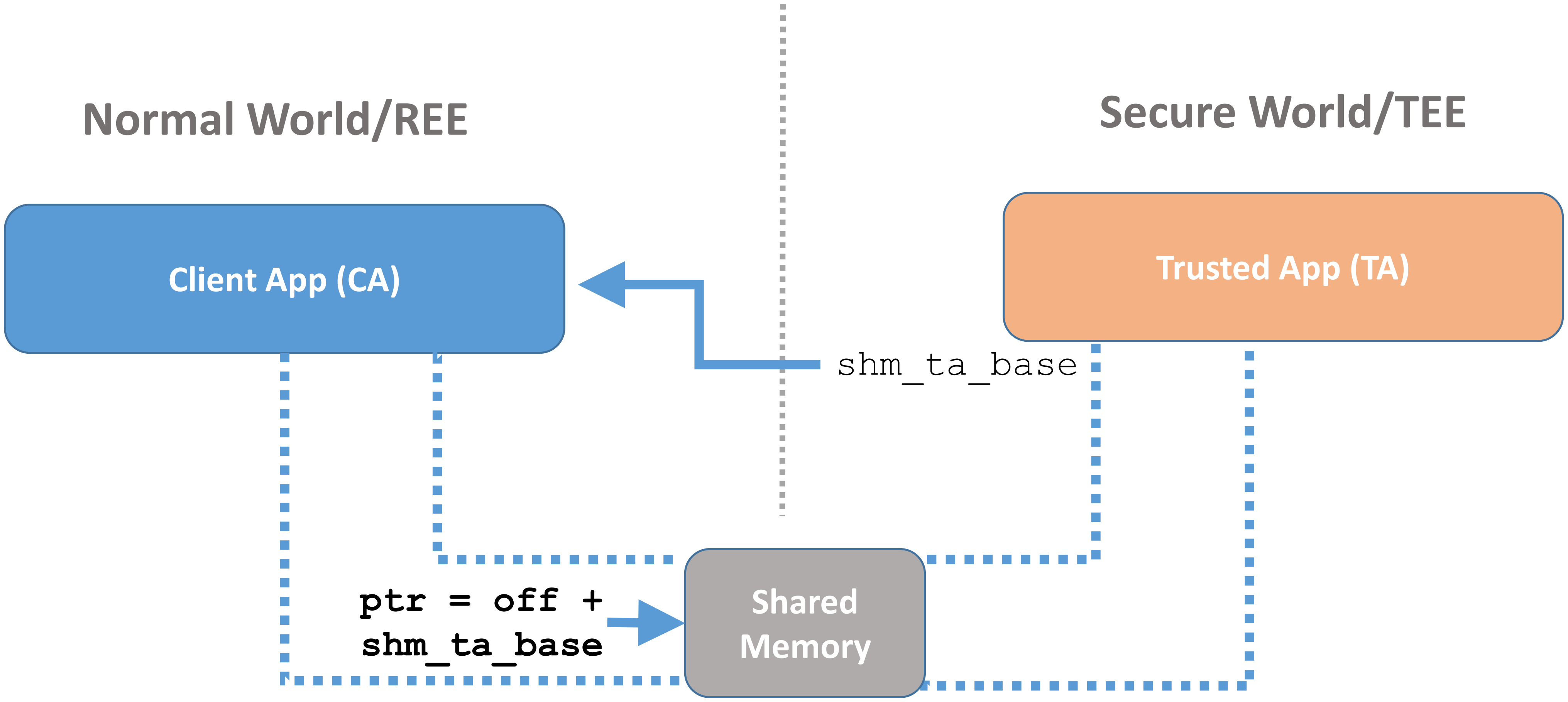
Anti-Pattern 2:
Unvalidated Normal-World Pointers



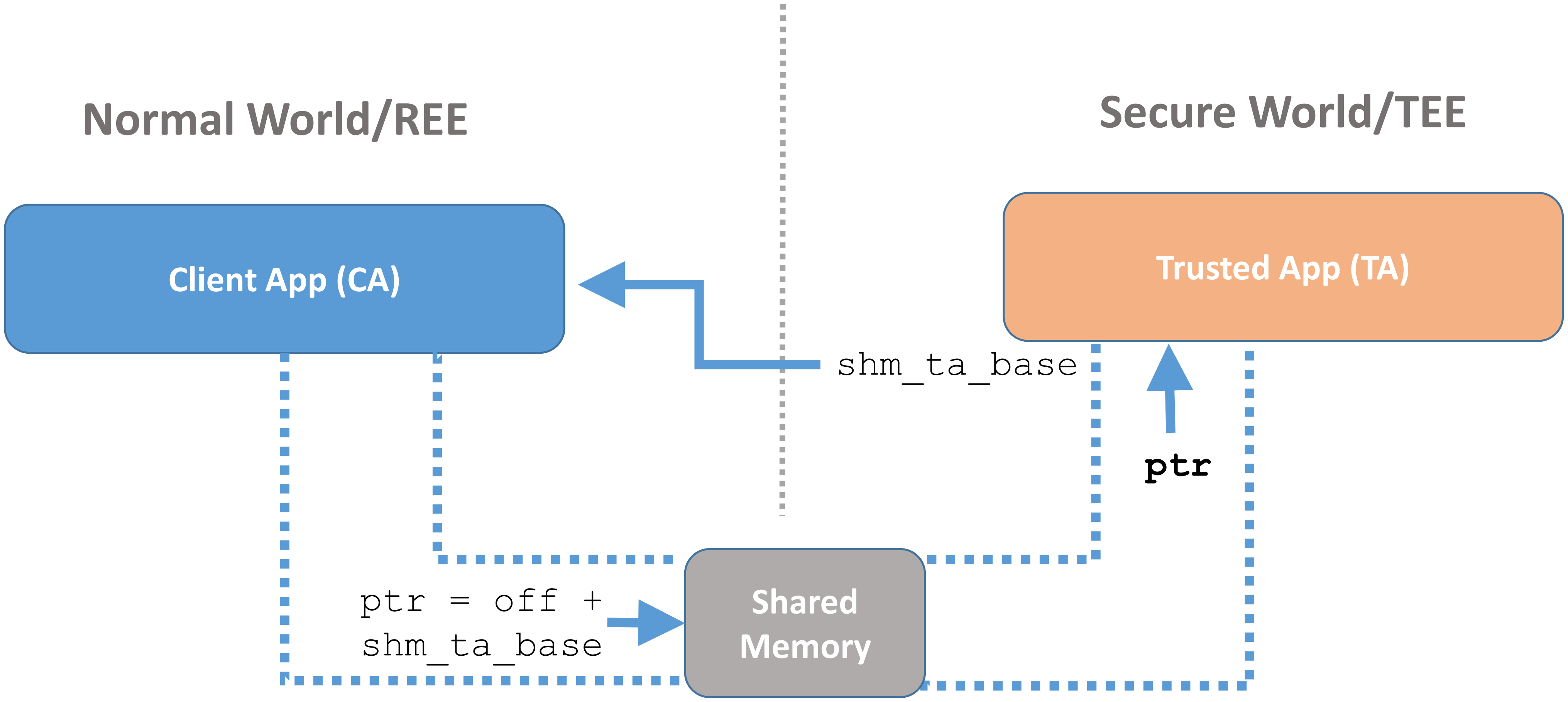
Anti-Pattern 2: Unvalidated Normal-World Pointers



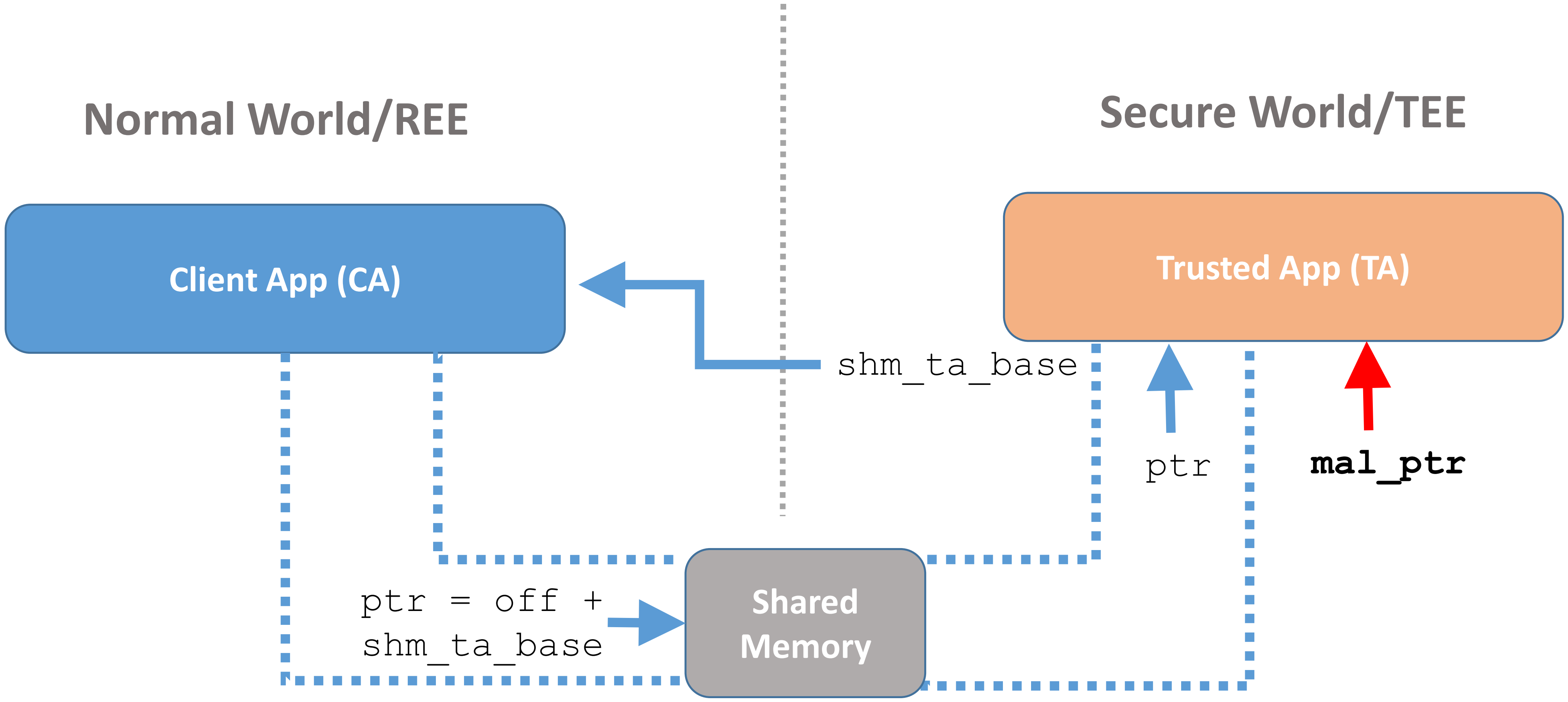
Anti-Pattern 2: Unvalidated Normal-World Pointers



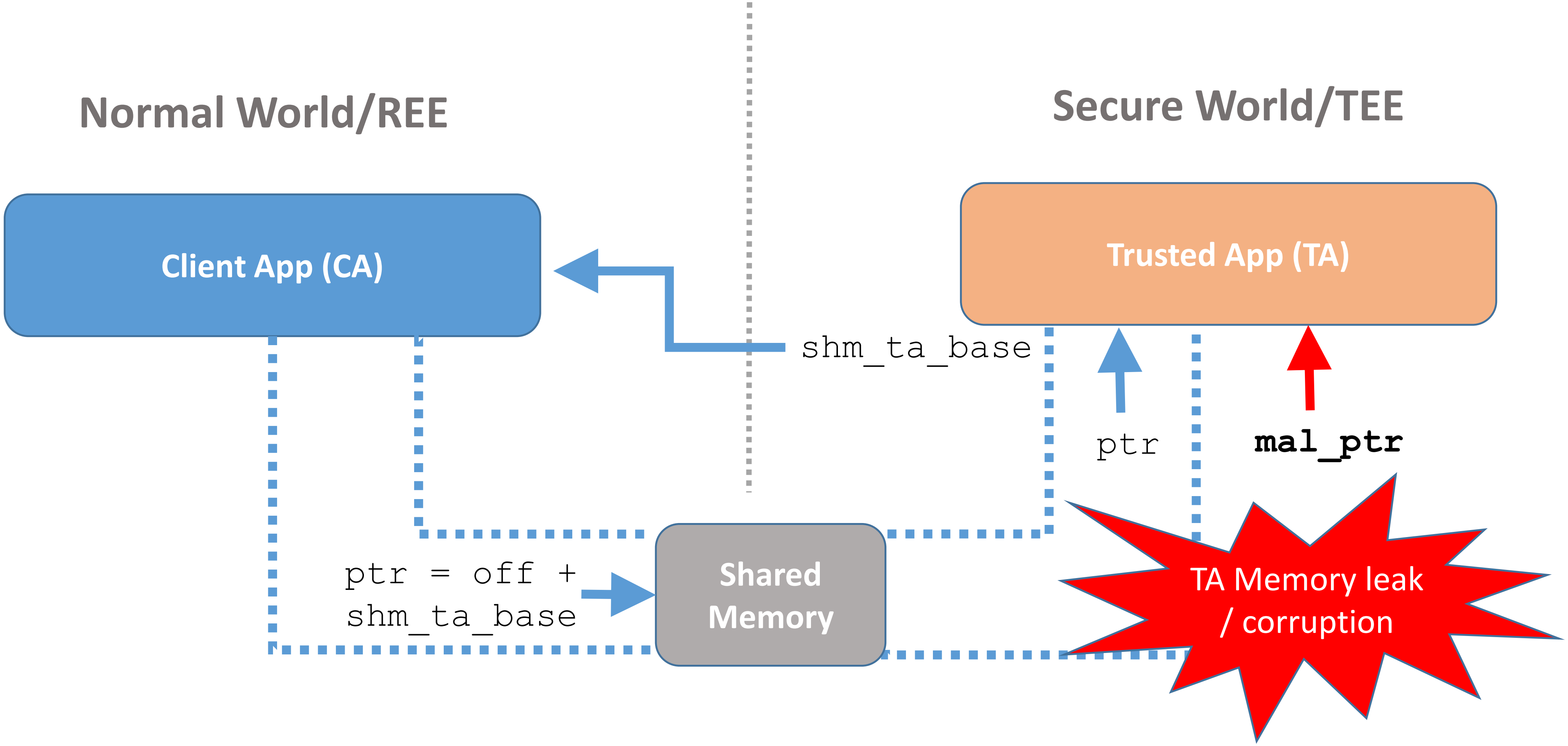
Anti-Pattern 2:
Unvalidated Normal-World Pointers



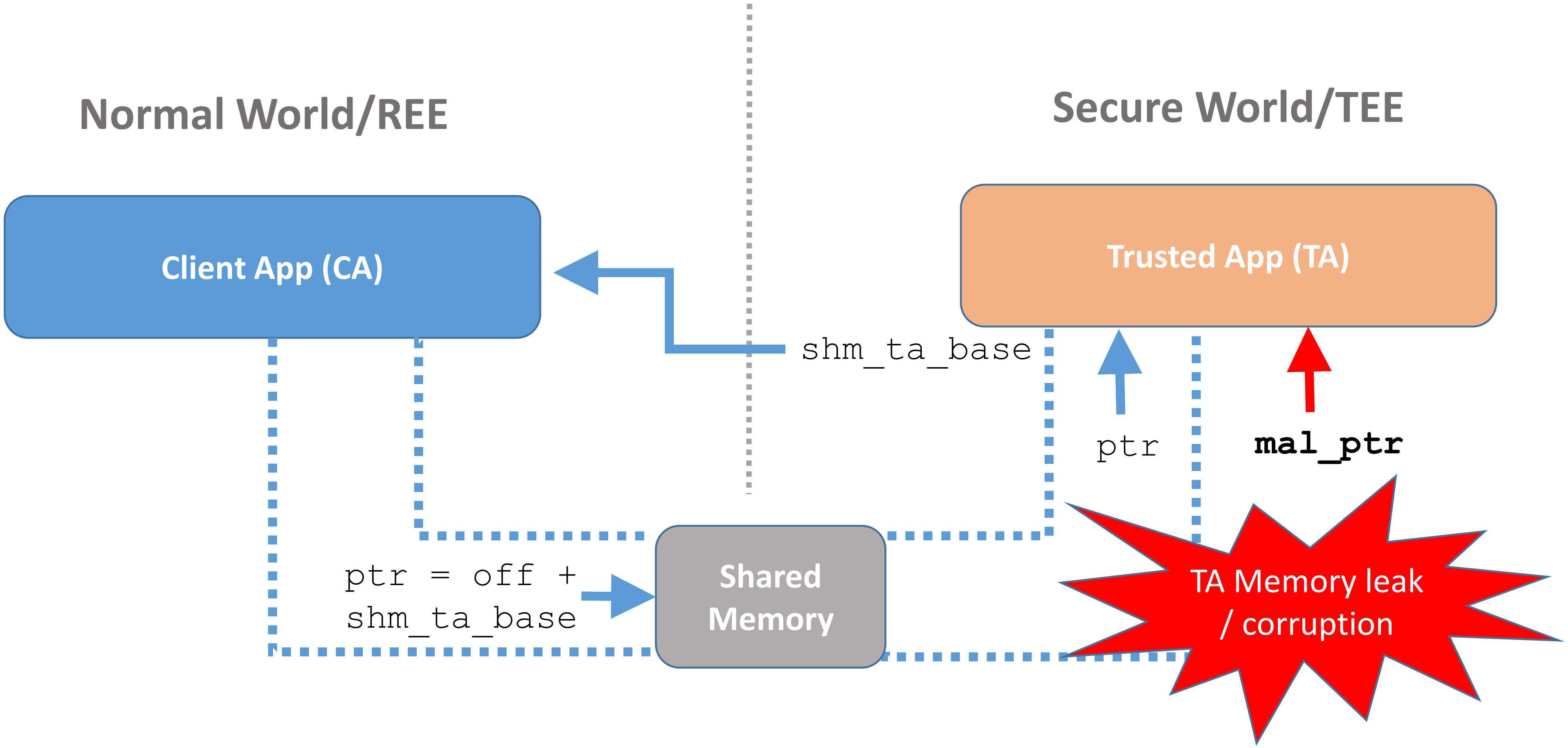
Anti-Pattern 2:
Unvalidated Normal-World Pointers



Anti-Pattern 2:
Unvalidated Normal-World Pointers



Anti-Pattern 2: Unvalidated Normal-World Pointers



TA should check that CA-supplied pointers point to shared memory

Anti-Pattern 3: Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
       uint32_t paramTypes, TEE_Params params[4])
{
       // Use params[0] as a buffer
       request_ptr = (struct request_struct *) params[0];
       switch (request_ptr->request) {
       …
       }
}
```

Anti-Pattern 3: Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
        uint32_t paramTypes, TEE_Params params[4])
{
        // Use params[0] as a buffer
        request_ptr = (struct request_struct *) params[0];
        switch (request_ptr->request) {
        …
        }
}
```

Anti-Pattern 3: Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
      uint32_t paramTypes, TEE_Params params[4])
{
      // Use params[0] as a buffer
      request_ptr = (struct request_struct *) params[0];
      switch (request_ptr->request) {
      …
      }
}
```

paramType(0) =

TEEC_MEMREF;

Anti-Pattern 3:
Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
      uint32_t paramTypes, TEE_Params params[4])
{
      // Use params[0] as a buffer
      request_ptr = (struct request_struct *) params[0];
      switch (request_ptr->request) {
      …
      }
}
```

```
paramTypes(0) =
      TEEC_MEMREF;
```

`paramTypes(0) =`
`TEEC_VALUE;`

Anti-Pattern 3:
Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
      uint32_t paramTypes, TEE_Params params[4])
{
      // Use params[0] as a buffer
      request_ptr = (struct request_struct *) params[0];
      switch (request_ptr->request) {
      …
      }
}
```

```
paramTypes(0) =
      TEEC_MEMREF;
```

```
paramTypes(0) =
      TEEC_VALUE;
```

TA Memory leak / corruption

Anti-Pattern 3: Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
        uint32_t paramTypes, TEE_Params params[4])
{
        // Use params[0] as a buffer
        request_ptr = (struct request_struct *) params[0];
        switch (request_ptr->request) {
        …
        }
}
```

```
paramTypes(0) =
      TEEC_MEMREF;
```

```
paramTypes(0) =
      TEEC_VALUE;
```

TA Memory leak / corruption

TA should check CA-supplied parameter types

Conclusion

- We showed that it is practically feasible to run real-world TZOSes and TAs in an emulator

Conclusion

- We showed that it is practically feasible to run real-world TZOSes and TAs in an emulator
- Large-scale fuzz testing of TAs using the emulator found several previously unknown vulnerabilities in TAs with high reproducibility

Conclusion

- We showed that it is practically feasible to run real-world TZOSes and TAs in an emulator
- Large-scale fuzz testing of TAs using the emulator found several previously unknown vulnerabilities in TAs with high reproducibility
- We identified vulnerability patterns unique to TA development
    - Pointing to the need for TZ-specific developer education

Conclusion

- We showed that it is practically feasible to run real-world TZOSes and TAs in an emulator
- Large-scale fuzz testing of TAs using the emulator found several previously unknown vulnerabilities in TAs with high reproducibility
- We identified vulnerability patterns unique to TA development
  - Pointing to the need for TZ-specific developer education

Thank you!