Pwning Blockchain for Fun and Profit

Exploiting an RCE Vulnerability in the Solana validator

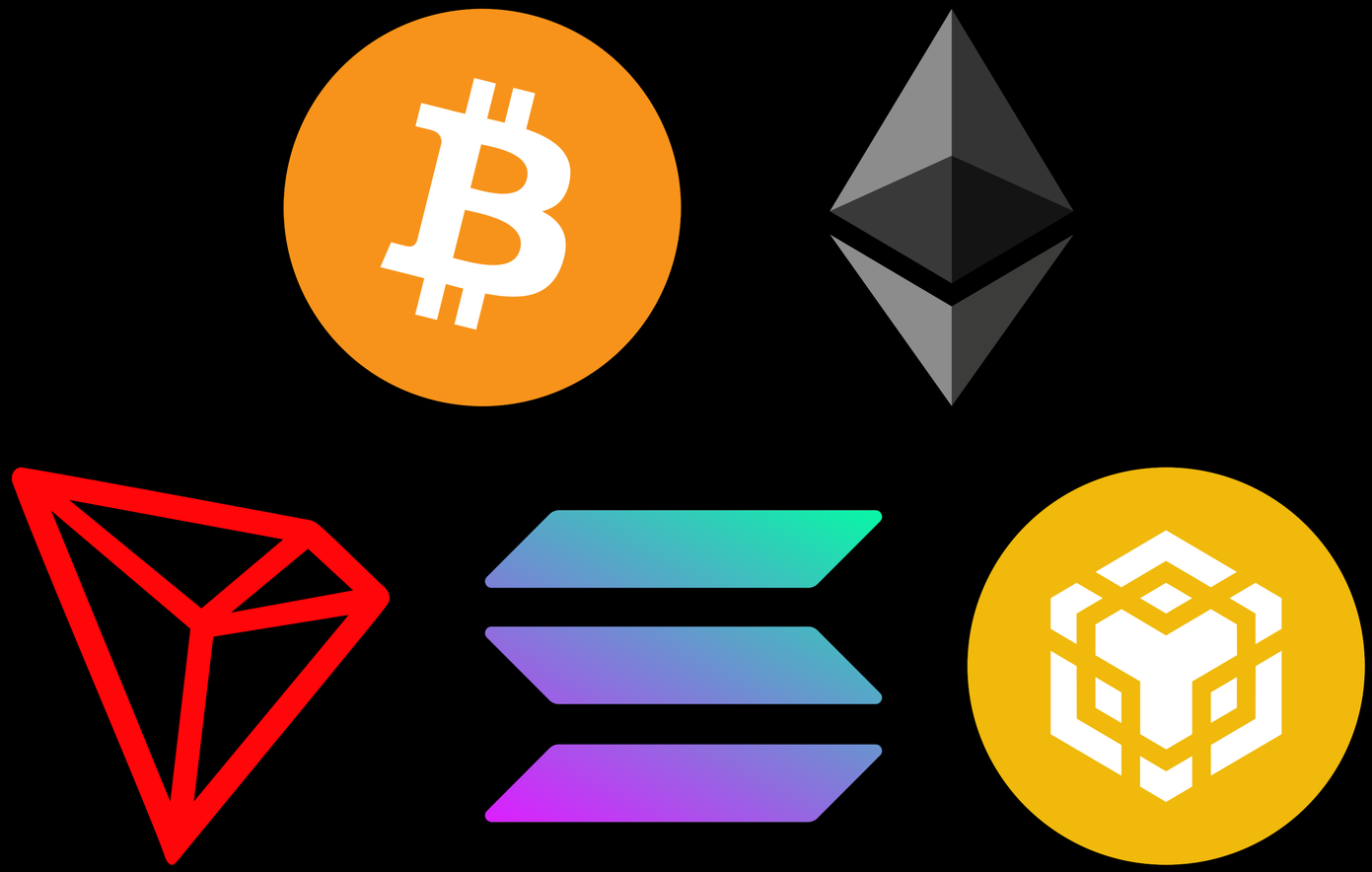
Ginoah

whoami

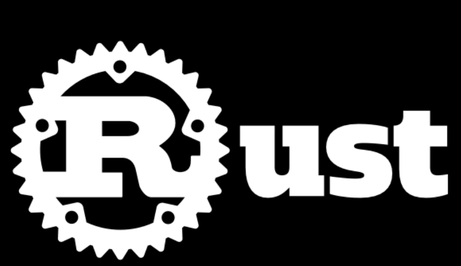
- Ginoah
- CTF with Balsn
- Bug Bounty with Anatomist

Intro

Blockchain



Solana



Versions

- New mechanism: **Direct Mapping**
  - Legacy versions: ≤ v1.14.x
  - New version: v1.16.0
- Reported and patched: June-July 2023.

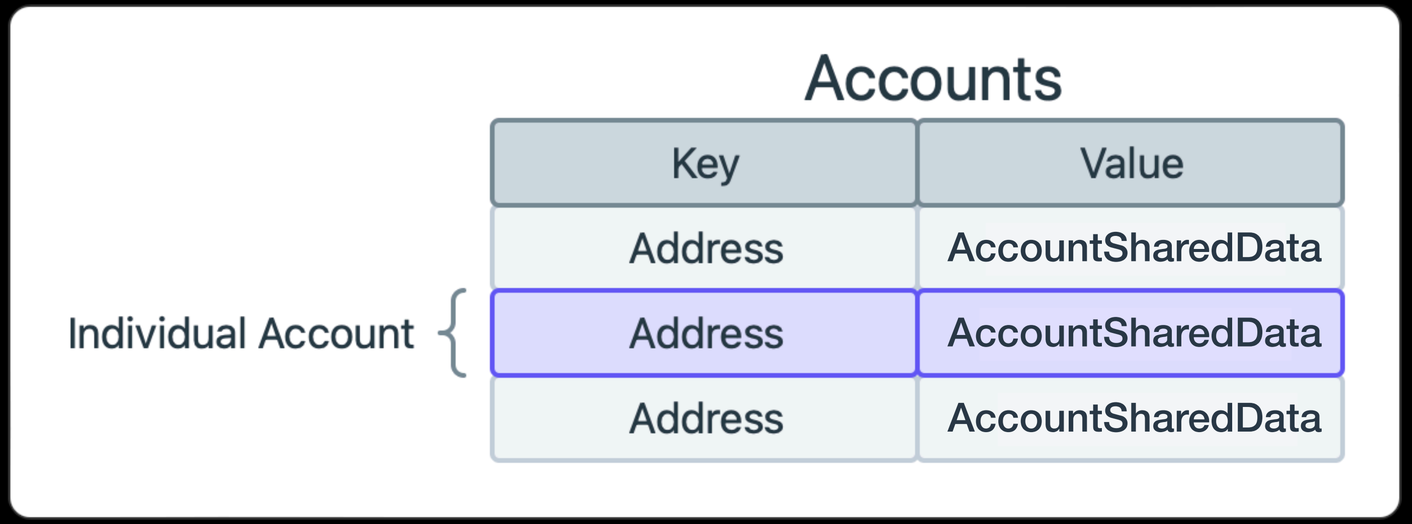
Index

- Basic Concept
- Legacy Model
- Direct Mapping
- New Model
- Bug
- Exploit

Data & Transaction Model

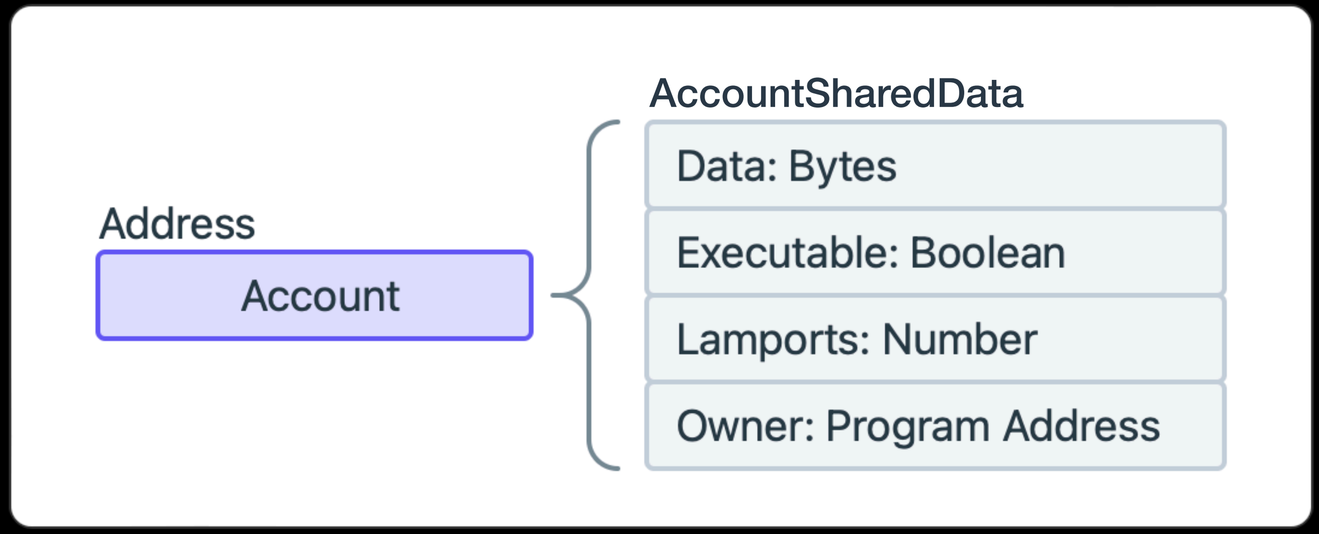
Data Storage Model

Accounts



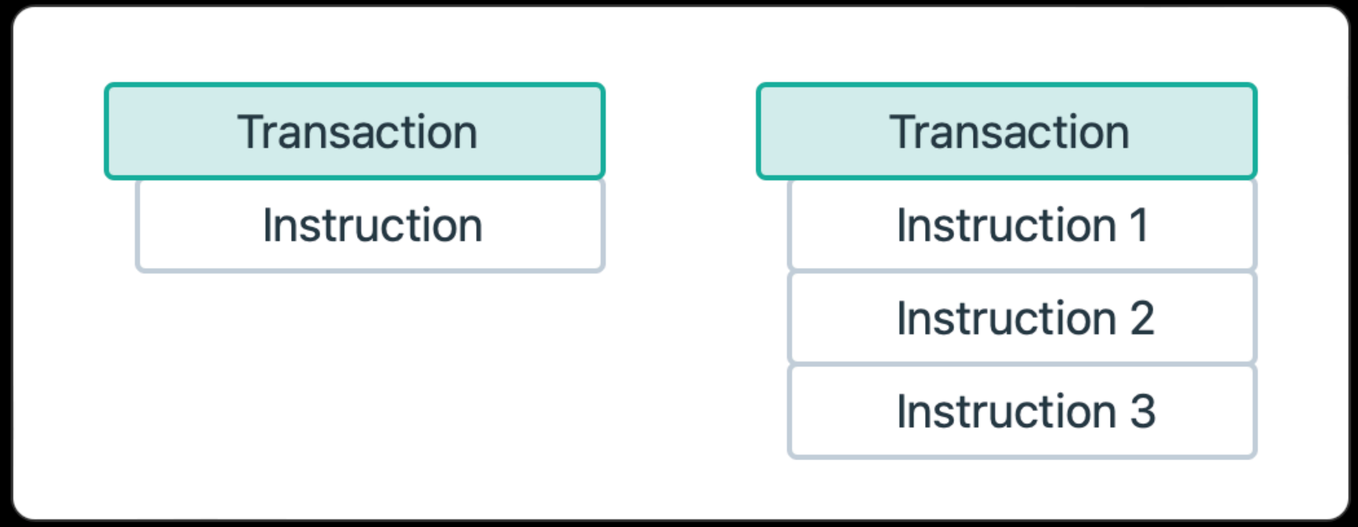
Data Storage Model

Accounts



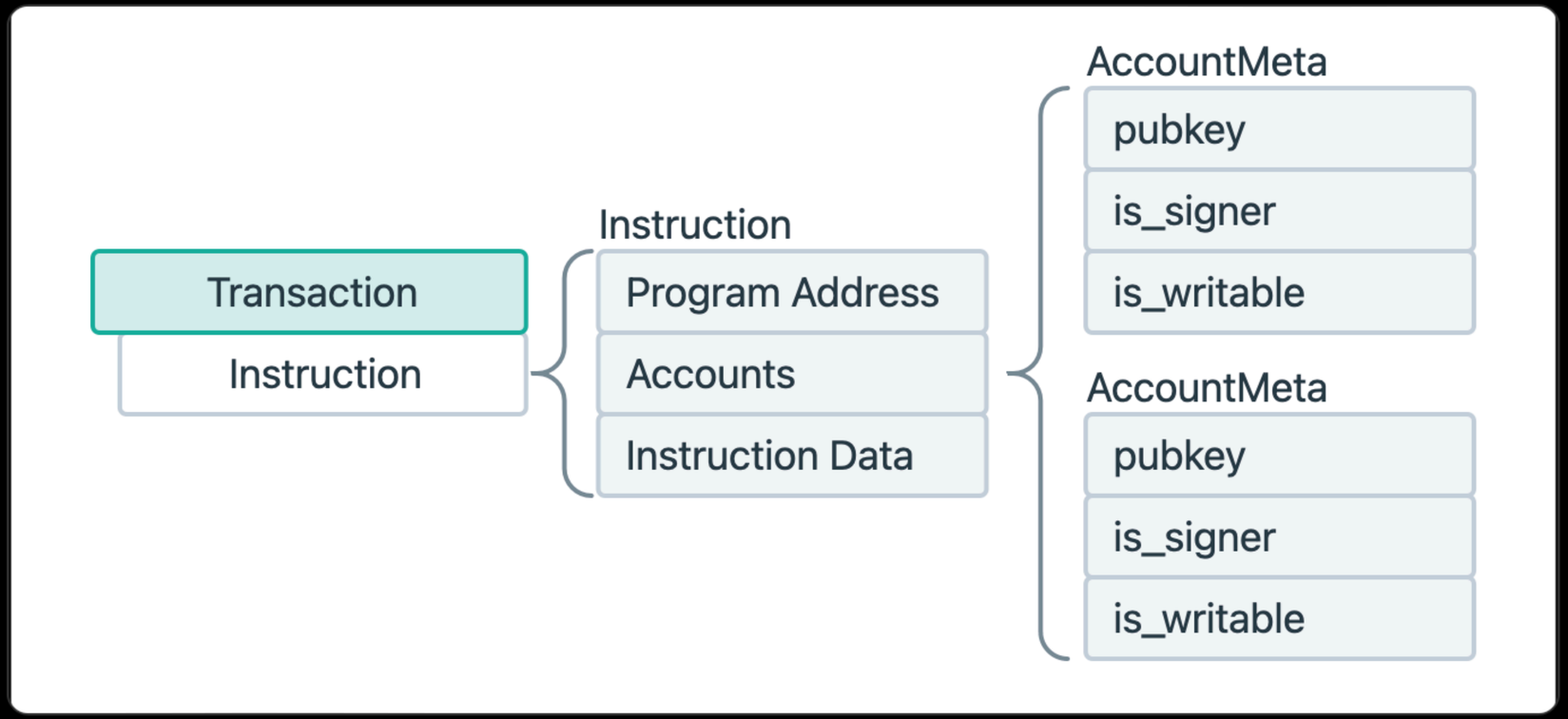
Transaction Model

Instructions & Accounts



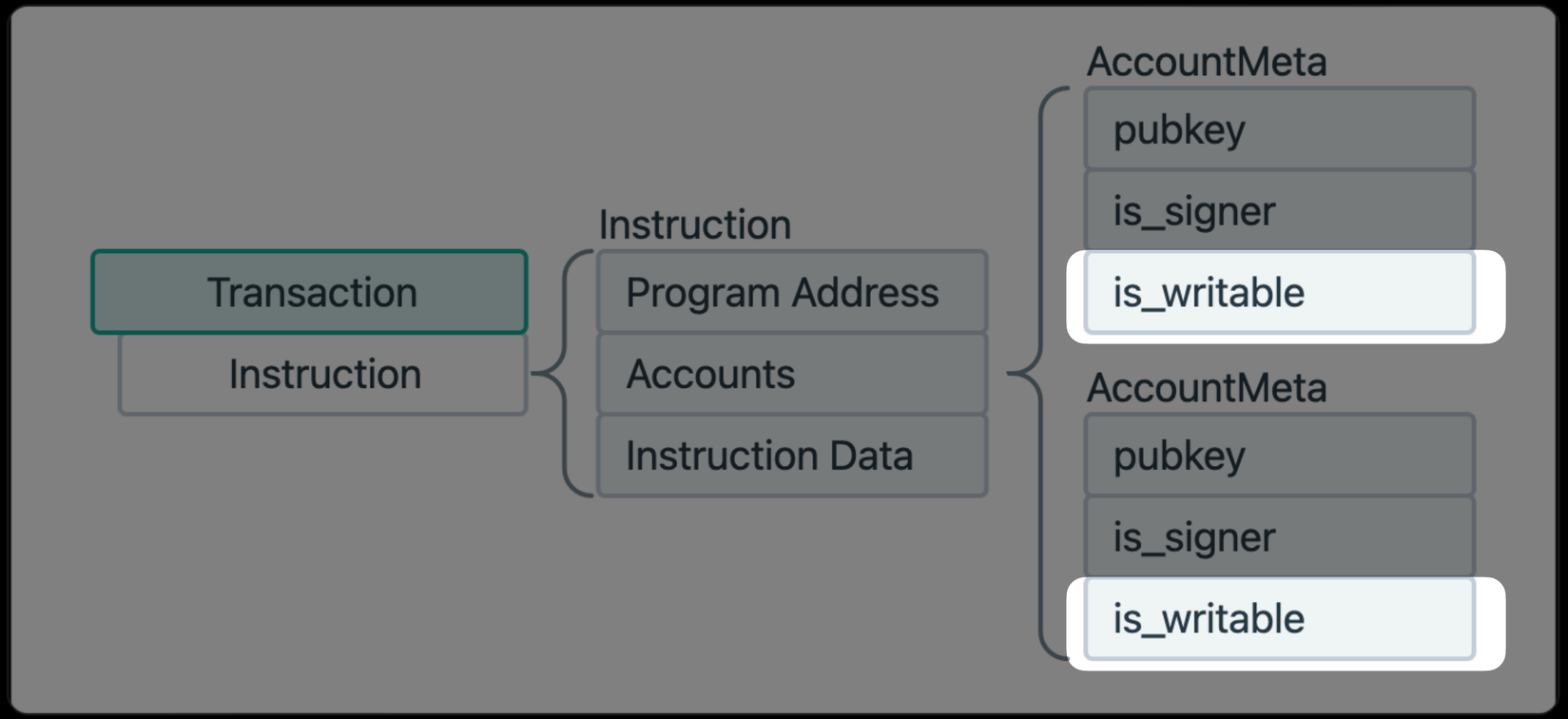
Transaction Model

Instructions & Accounts



Transaction Model

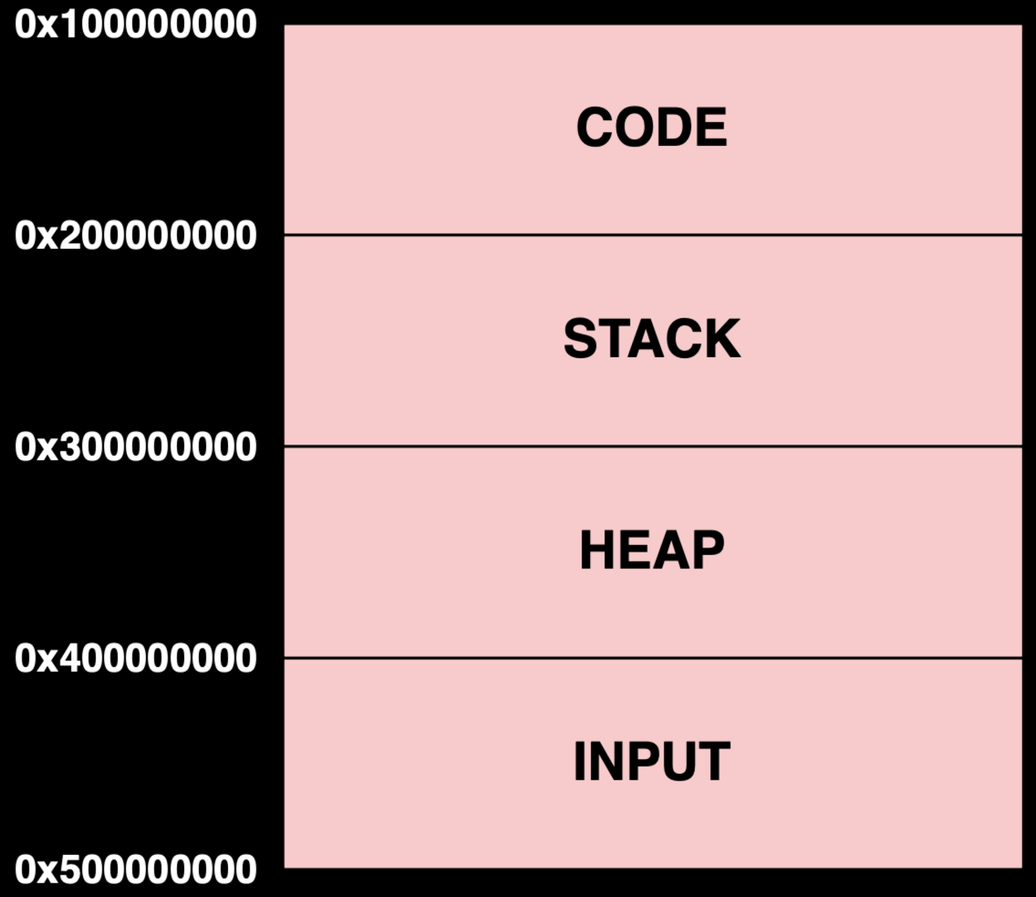
Instructions & Accounts



Legacy Data Exposure

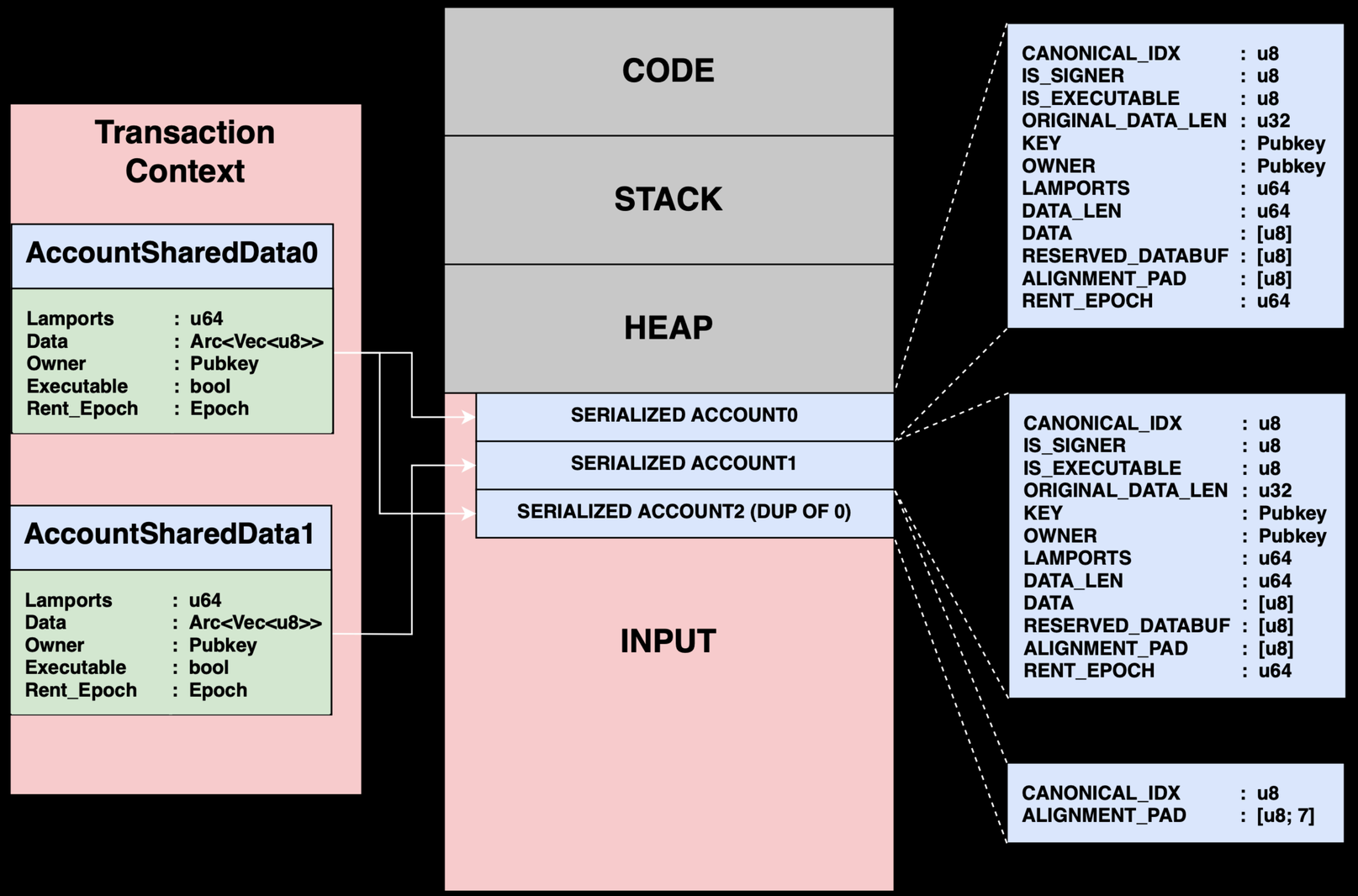
Legacy Data Exposure

rbpf runtime memory layout



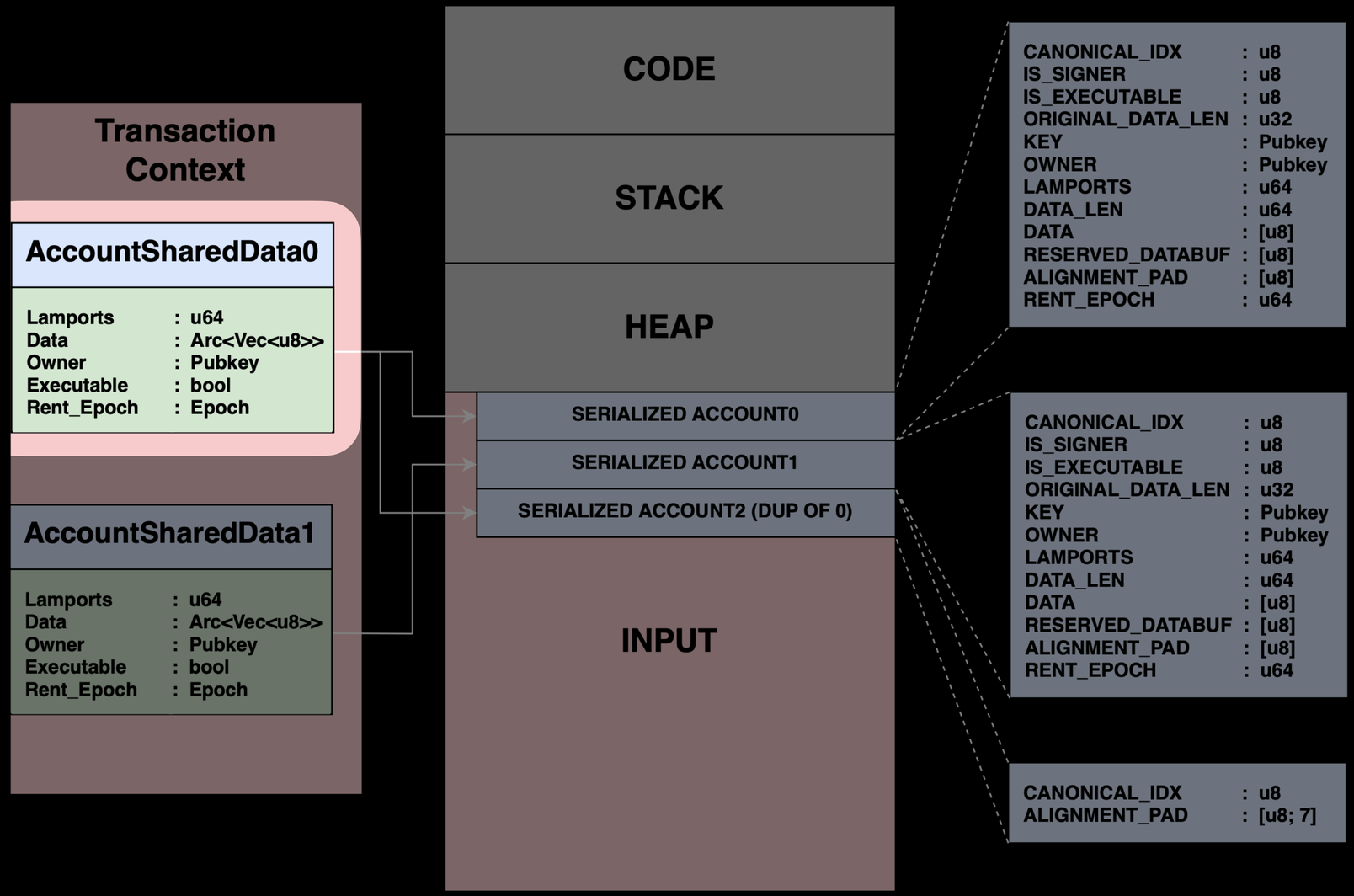
Legacy Data Exposure

Accounts serialization



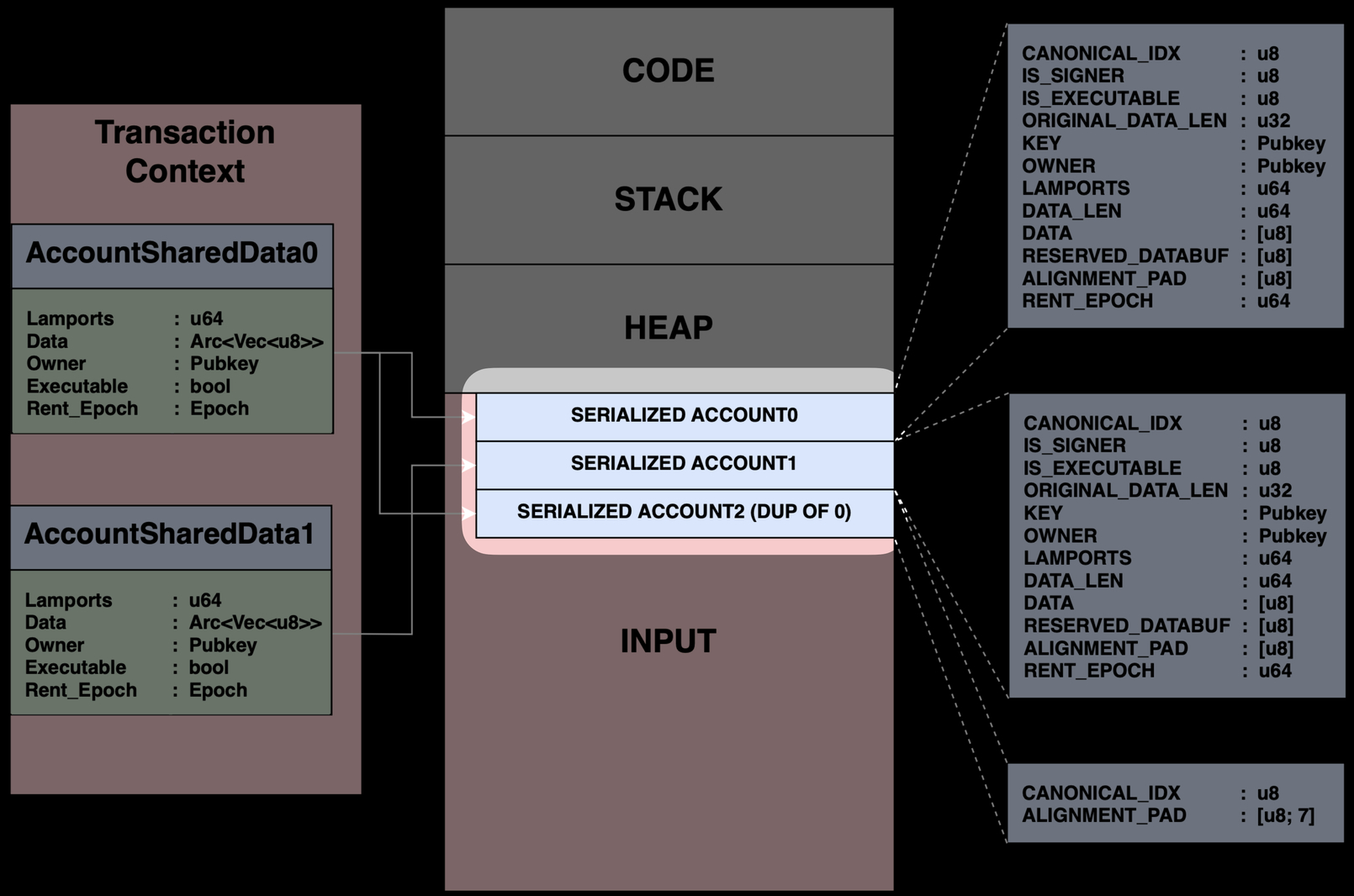
Legacy Data Exposure

Accounts serialization



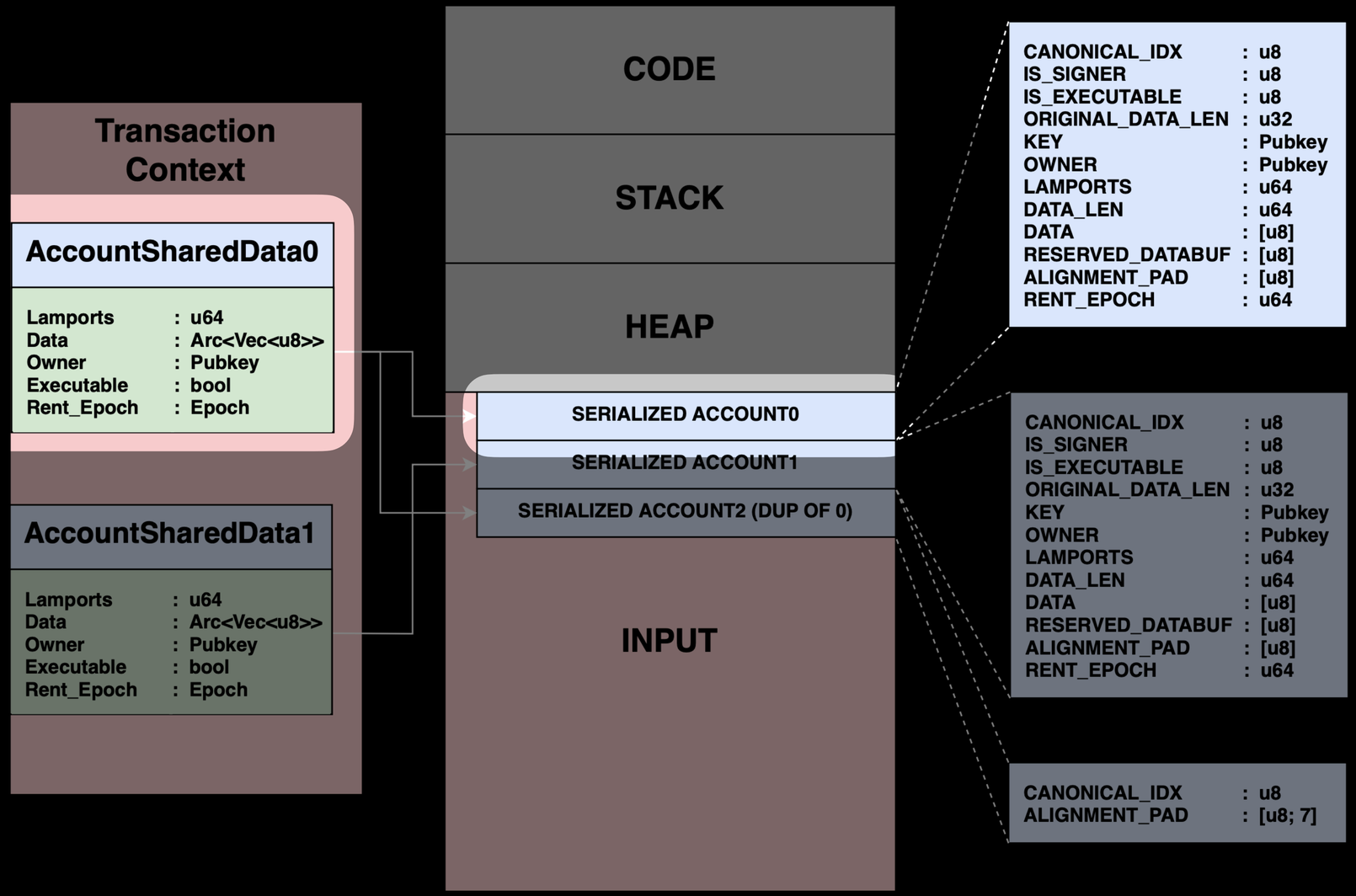
Legacy Data Exposure

Accounts serialization



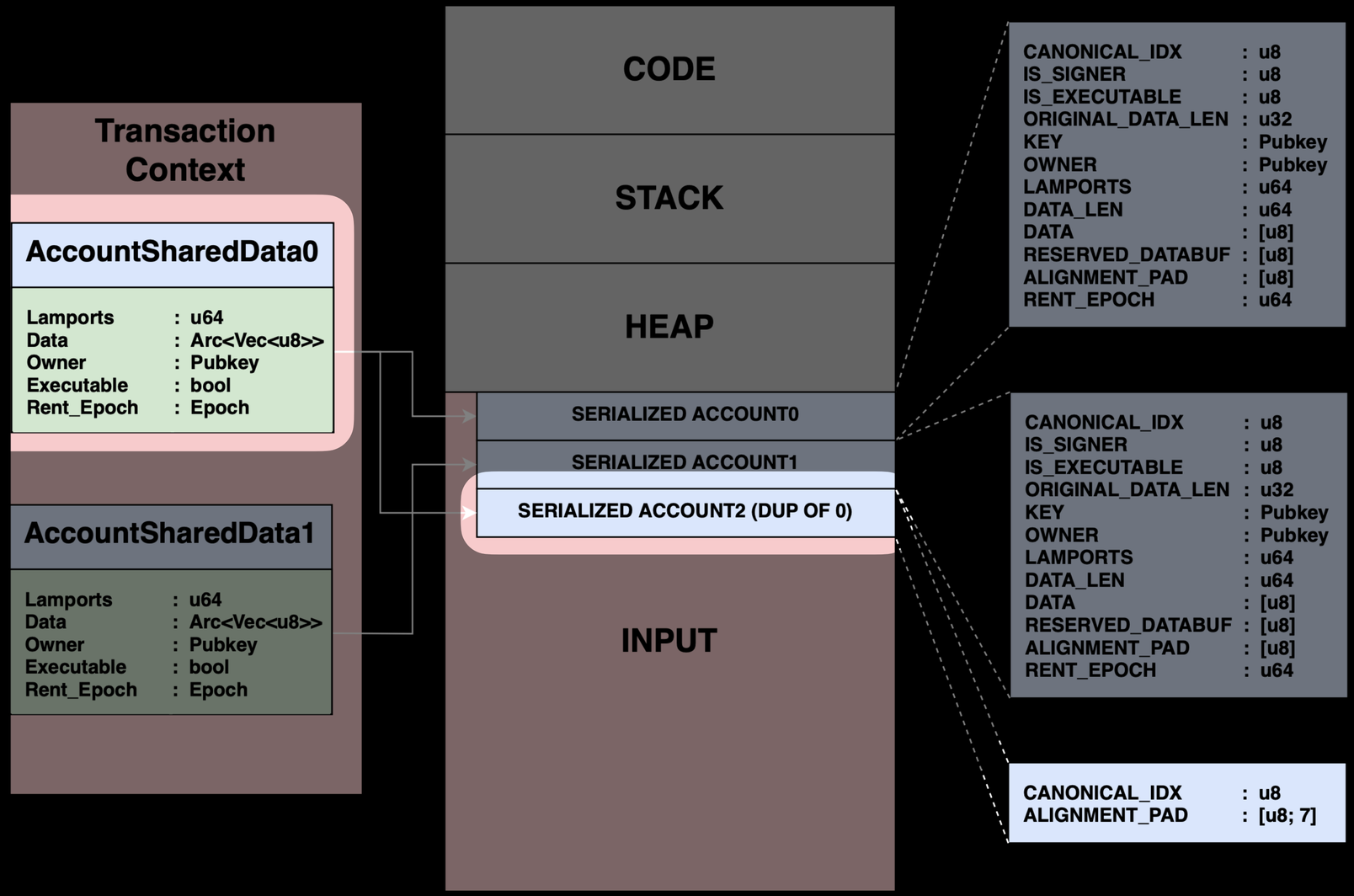
Legacy Data Exposure

Accounts serialization



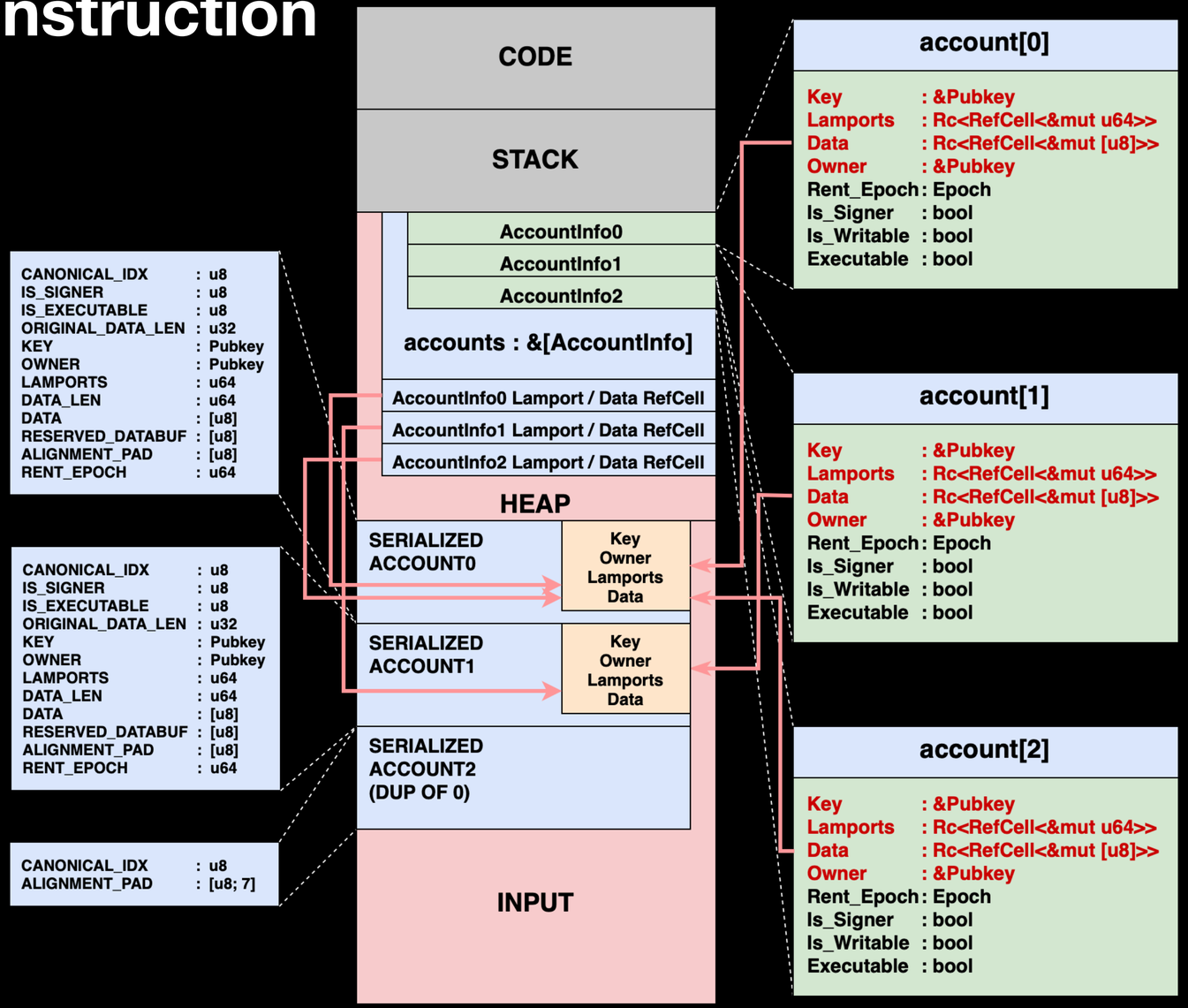
Legacy Data Exposure

Accounts serialization



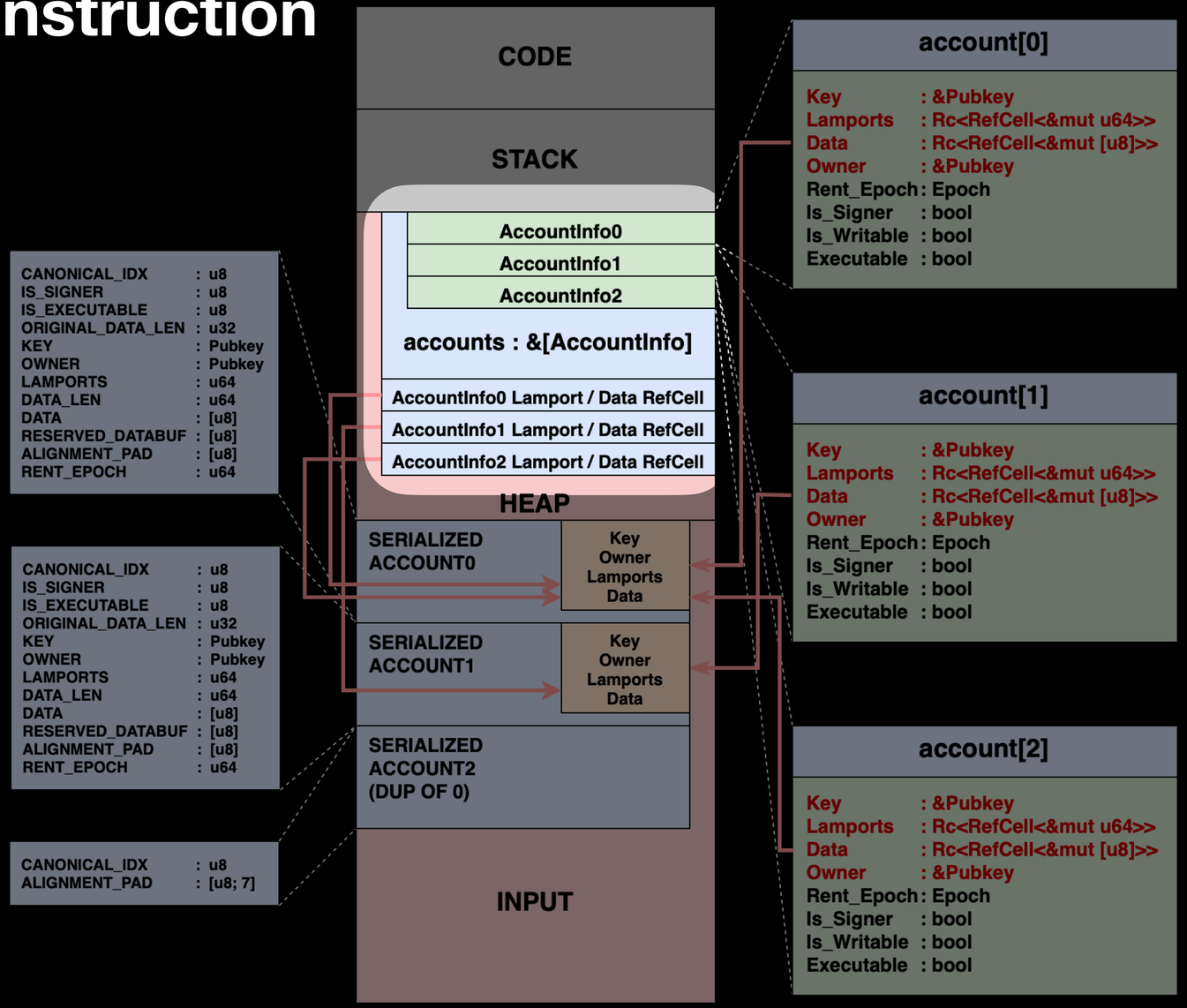
Legacy Data Exposure

AccountInfo construction



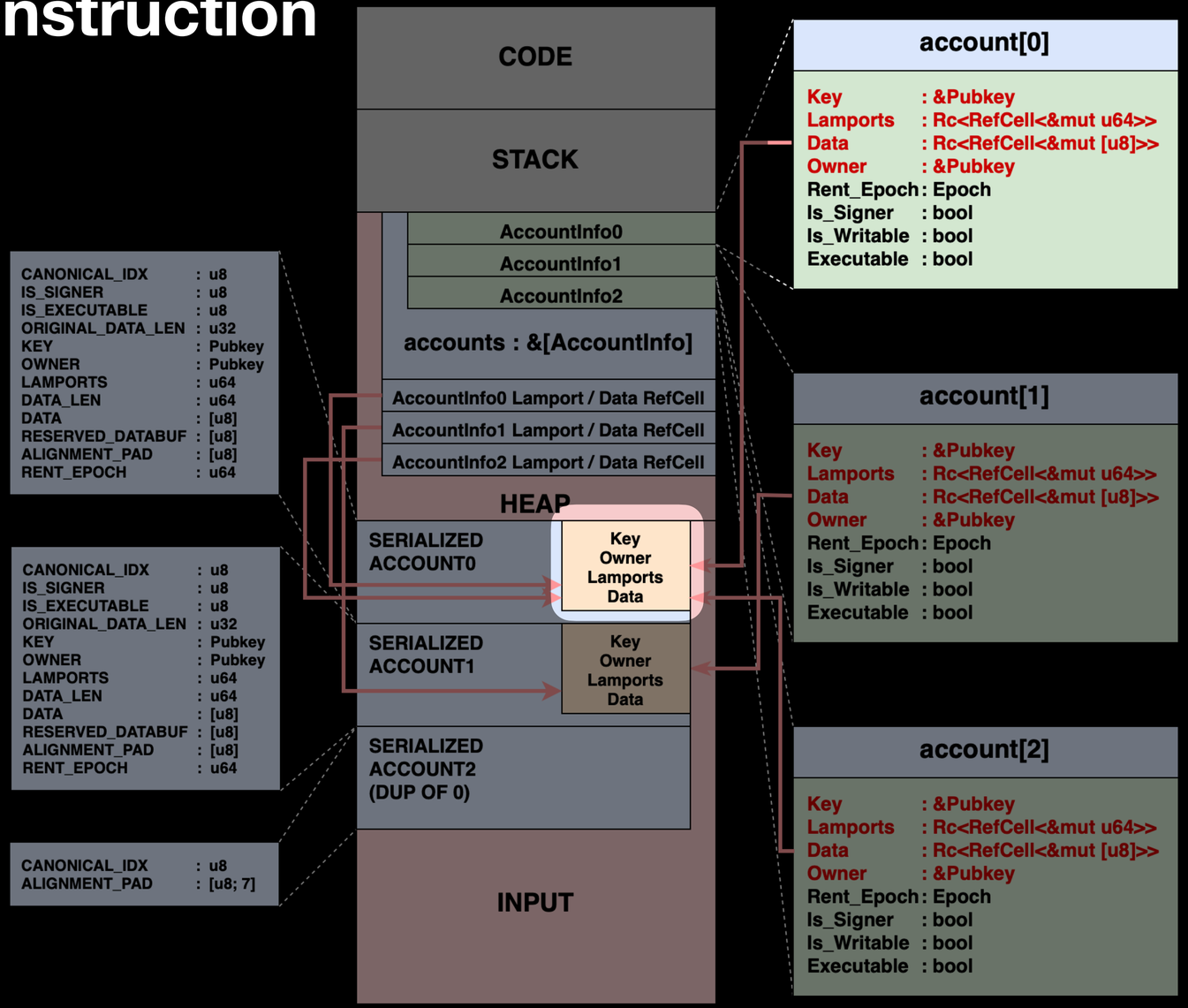
Legacy Data Exposure

AccountInfo construction



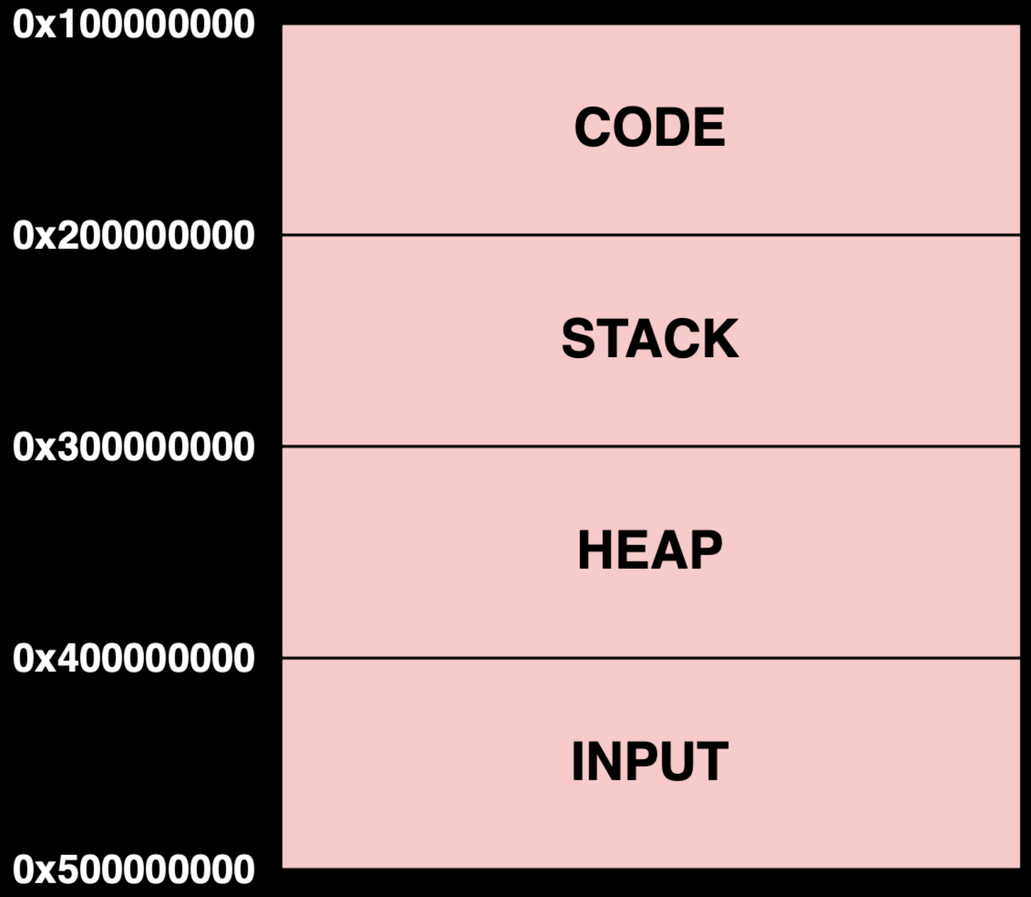
Legacy Data Exposure

AccountInfo construction



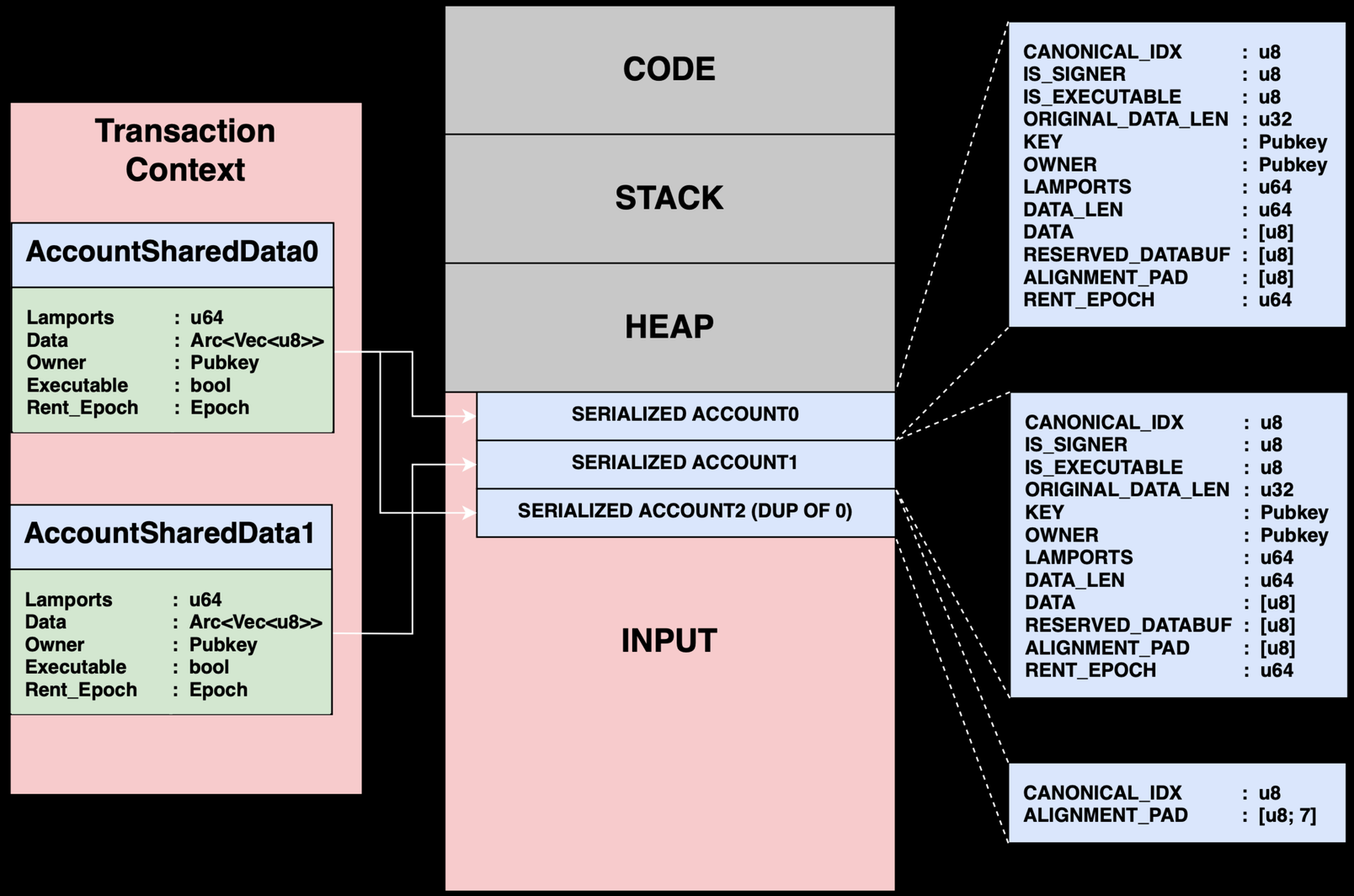
Legacy Data Exposure

Write permission check



Legacy Data Exposure

Write permission check



Legacy Data Exposure

Write permission check

- Account fields are only writable if the Account is marked as writable in the instruction
- Only the account Owner can modify Owner / Data fields
- Owner can only be modified when data == '\0' * len(data)
- Lamports can be increased by anyone
- Lamports can only be decreased by Owner
- Sum of Lamports in all accounts passed to tx must remain consistent after execution
- …

Legacy Data Exposure

Write permission check

- Account fields are only writable if the Account is marked as writable in the instruction
- Only the account Owner can modify Owner / Data fields
- Owner can only be modified when data == '\0' * len(data)
- Lamports can be increased by anyone
- Lamports can only be decreased by Owner
- Sum of Lamports in all accounts passed to tx must remain consistent after execution
- …

Legacy Data Exposure

Write permission check

- Account fields are only writable if the Account is marked as writable in the instruction
- Only the account Owner can modify Owner / Data fields
- Owner can only be modified when data == '\0' * len(data)
- Lamports can be increased by anyone
- Lamports can only be decreased by Owner
- Sum of Lamports in all accounts passed to tx must remain consistent after execution
- …

Legacy Data Exposure

Write permission check

- Account fields are only writable if the Account is marked as writable in the instruction
- Only the account Owner can modify Owner / Data fields
- Owner can only be modified when data == '\0' * len(data)
- Lamports can be increased by anyone
- Lamports can only be decreased by Owner
- Sum of Lamports in all accounts passed to tx must remain consistent after execution
- …

Legacy Data Exposure

Write permission check

- Account fields are only writable if the Account is marked as writable in the instruction
- Only the account Owner can modify Owner / Data fields
- Owner can only be modified when data == '\0' * len(data)
- Lamports can be increased by anyone
- Lamports can only be decreased by Owner
- Sum of Lamports in all accounts passed to tx must remain consistent after execution
- …

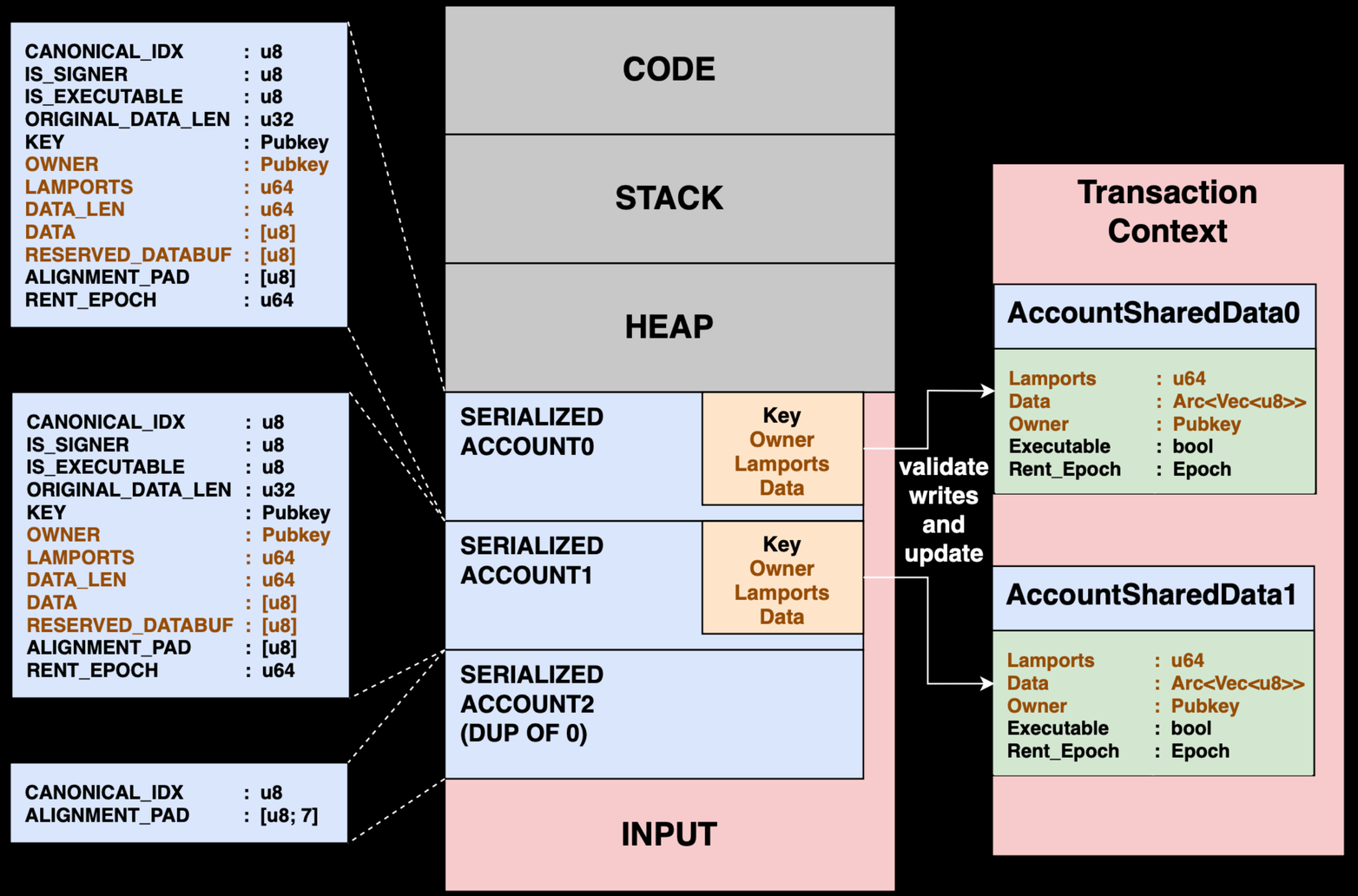
Legacy Data Exposure

Write permission check

- Account fields are only writable if the Account is marked as writable in the instruction
- Only the account Owner can modify Owner / Data fields
- Owner can only be modified when data == '\0' * len(data)
- Lamports can be increased by anyone
- Lamports can only be decreased by Owner
- Sum of Lamports in all accounts passed to tx must remain consistent after execution
- …

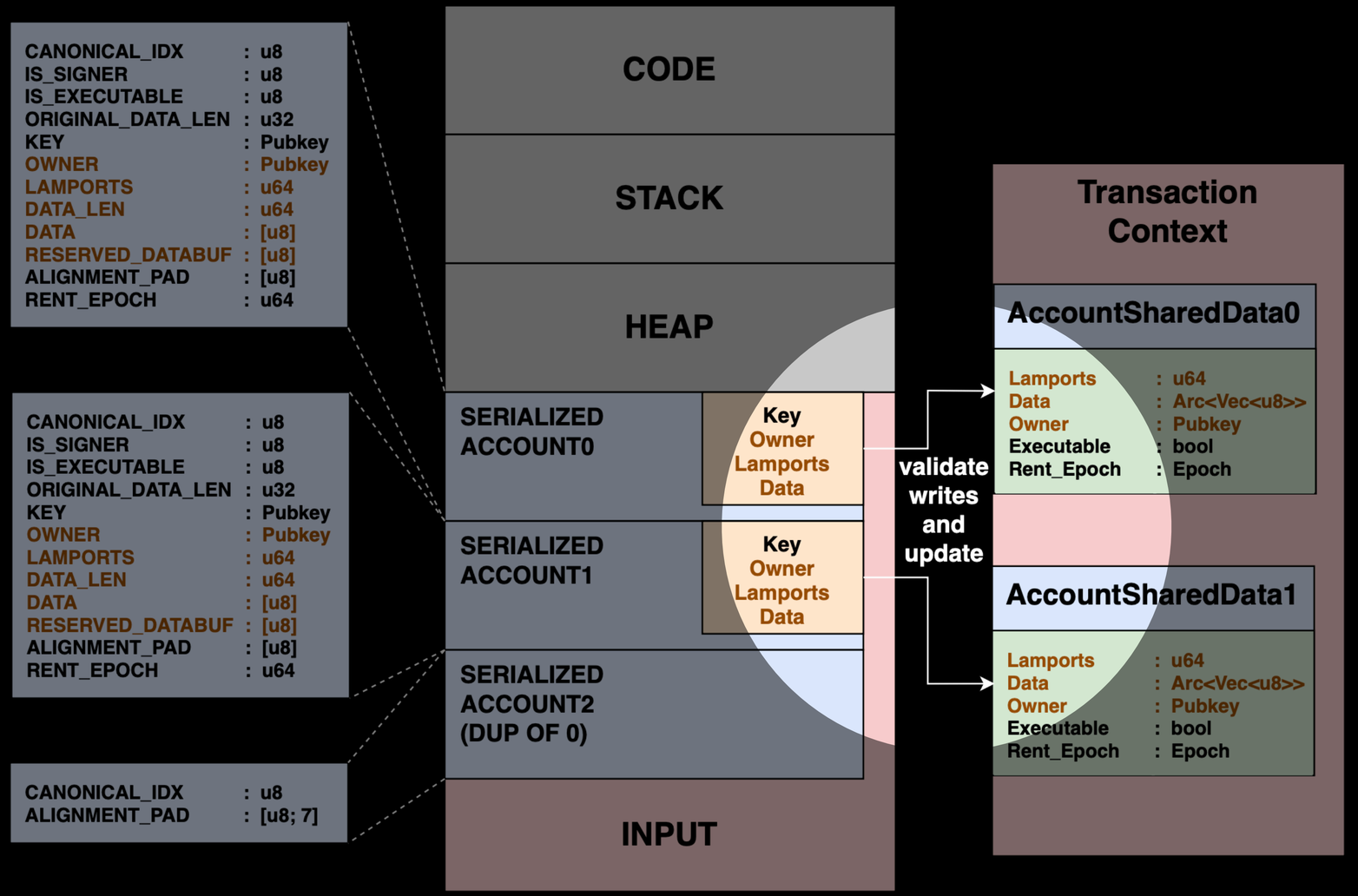
Legacy Data Exposure

Commit



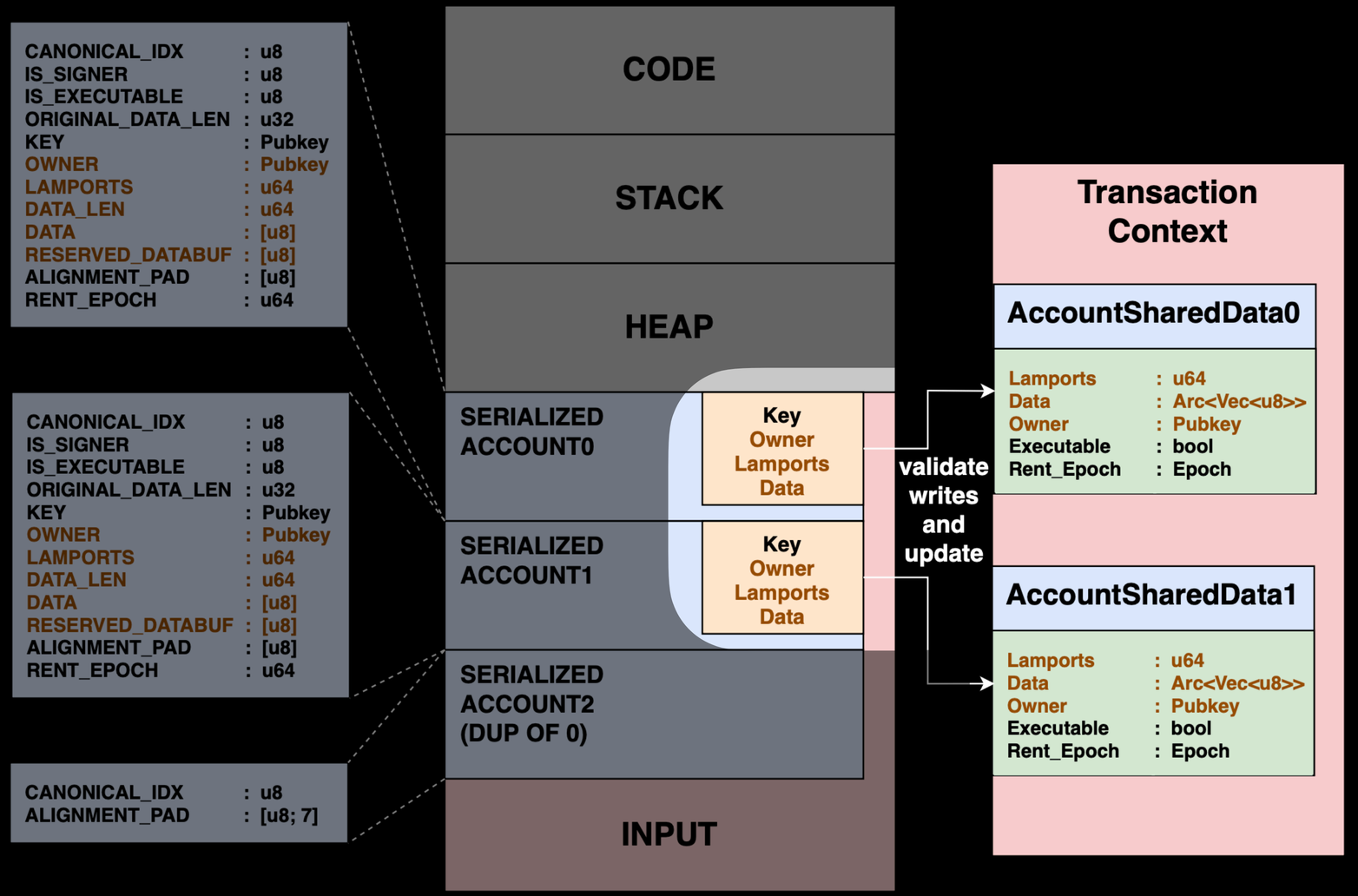
Legacy Data Exposure

Commit



Legacy Data Exposure

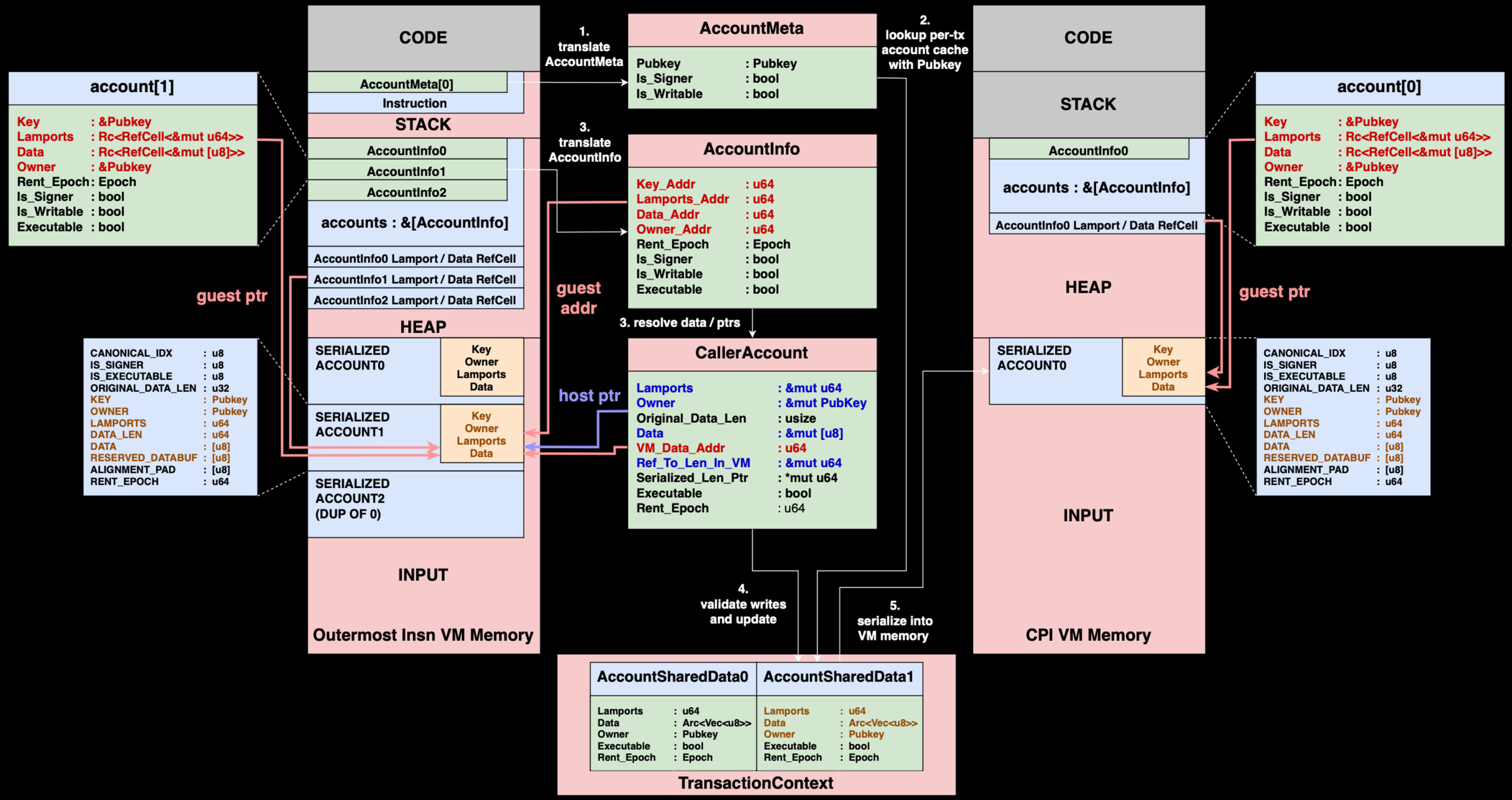
Commit



Legacy Interoperability

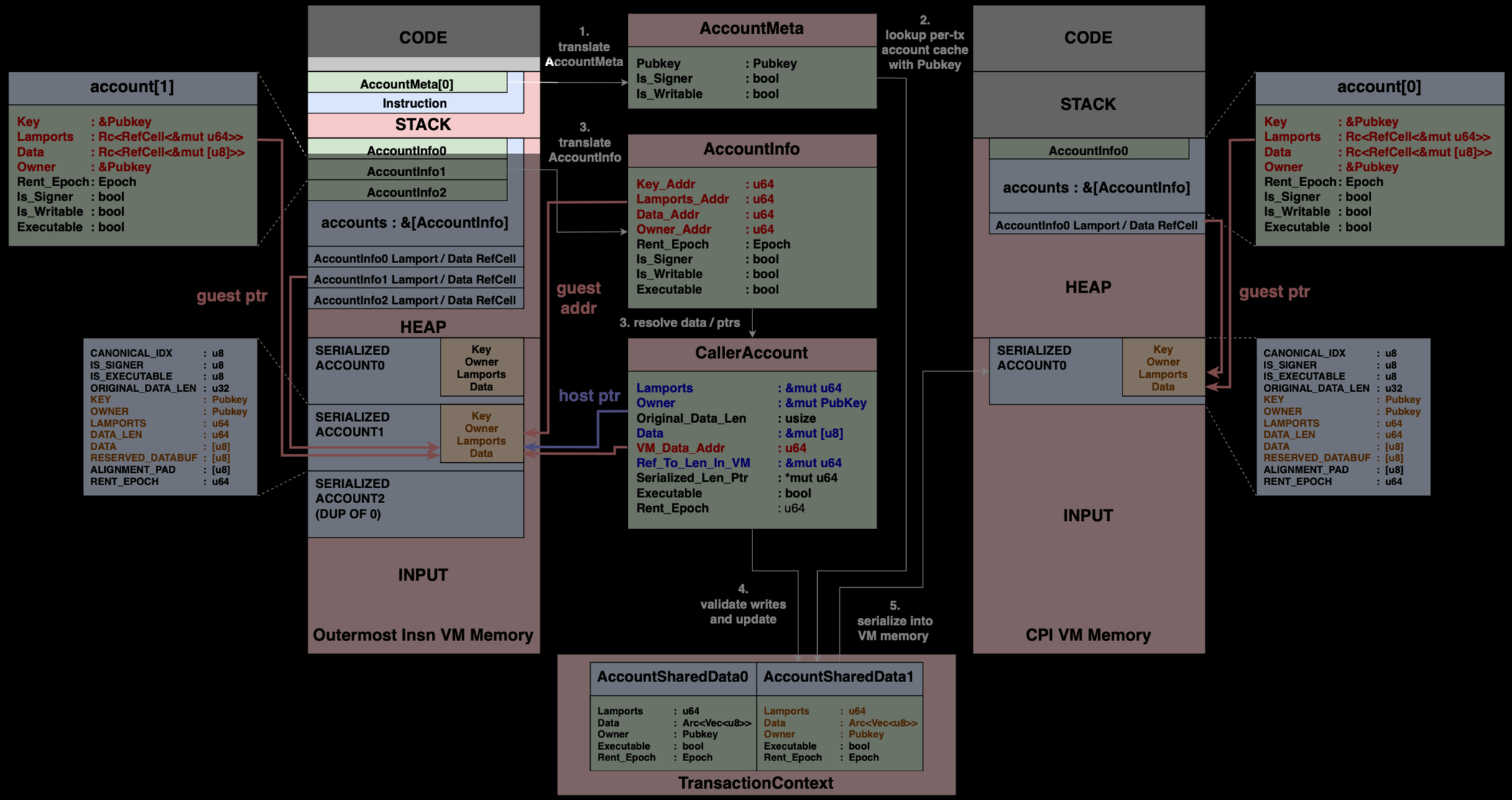
Legacy Interoperability

cpi (cross-program-invocation) call



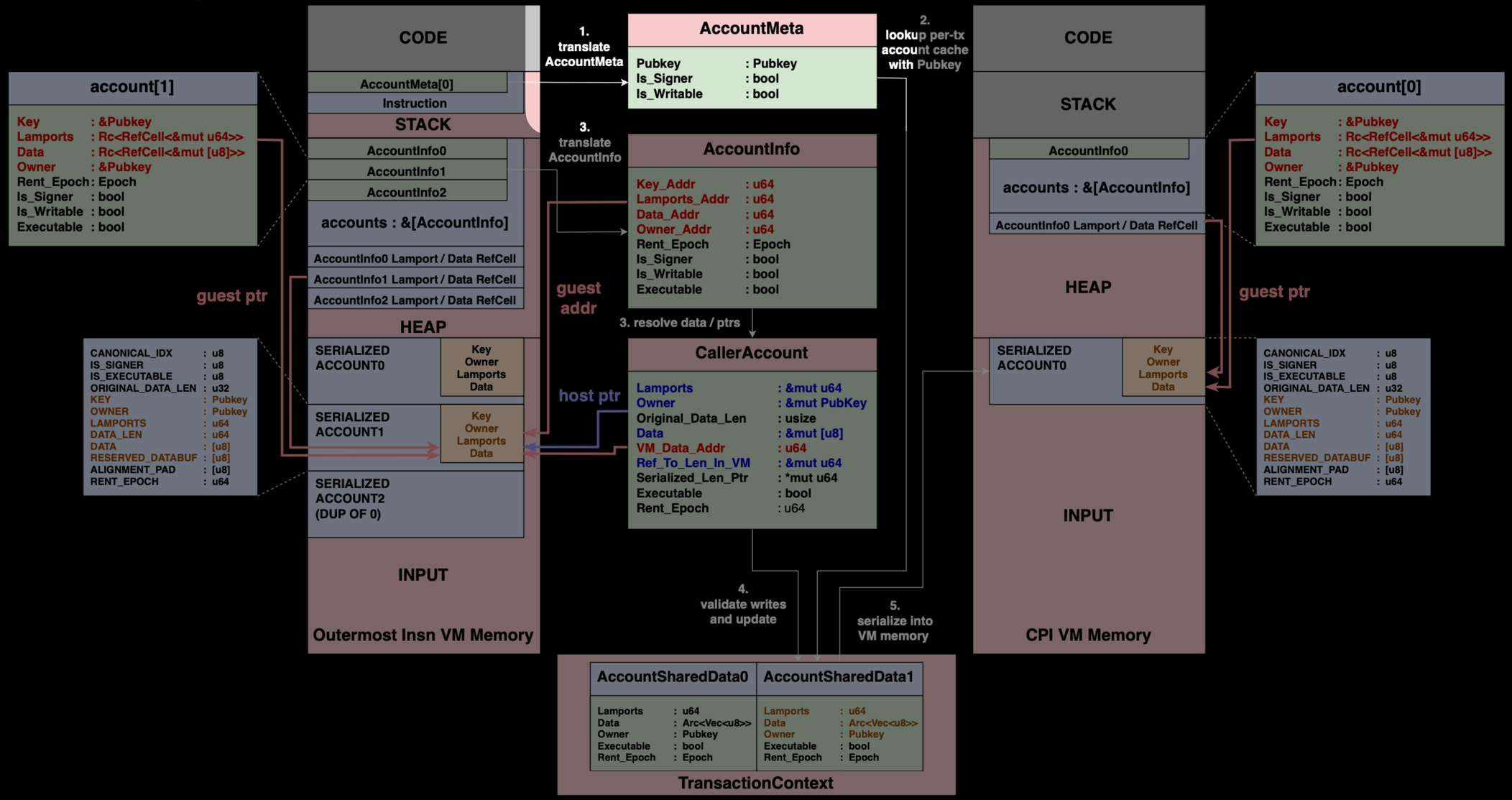
Legacy Interoperability

cpi (cross-program-invocation) call



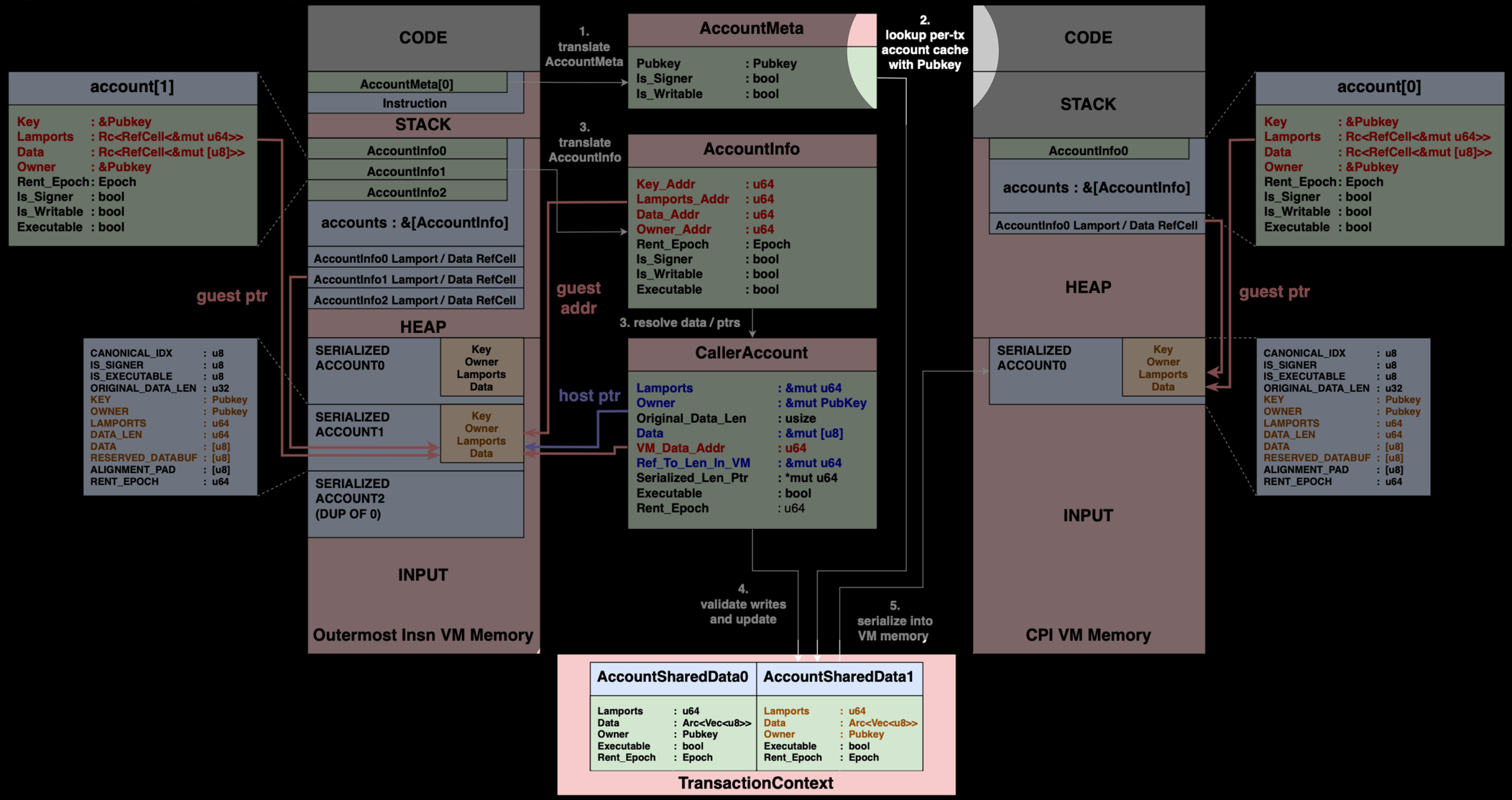
Legacy Interoperability

cpi (cross-program-invocation) call



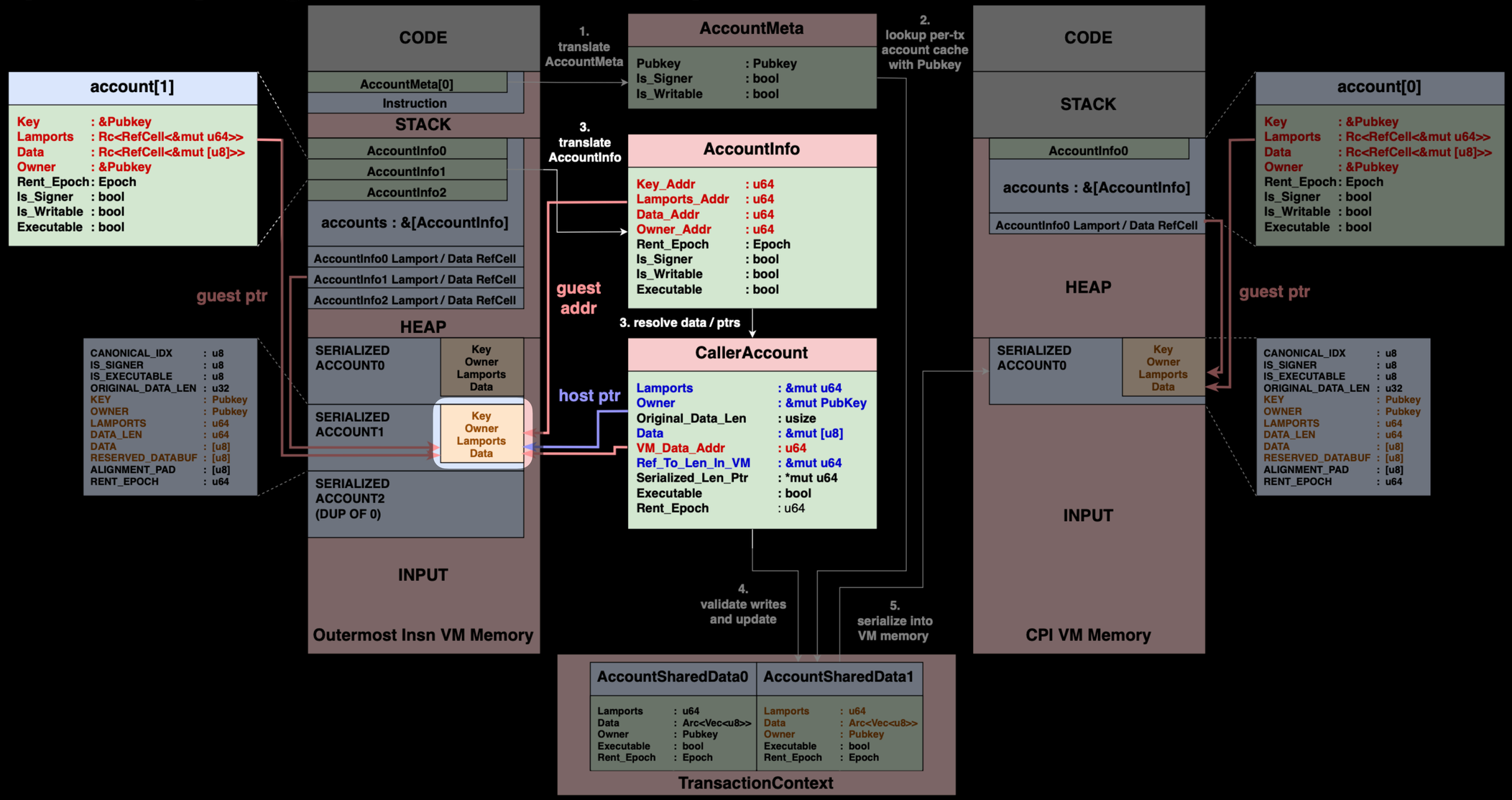
Legacy Interoperability

cpi (cross-program-invocation) call



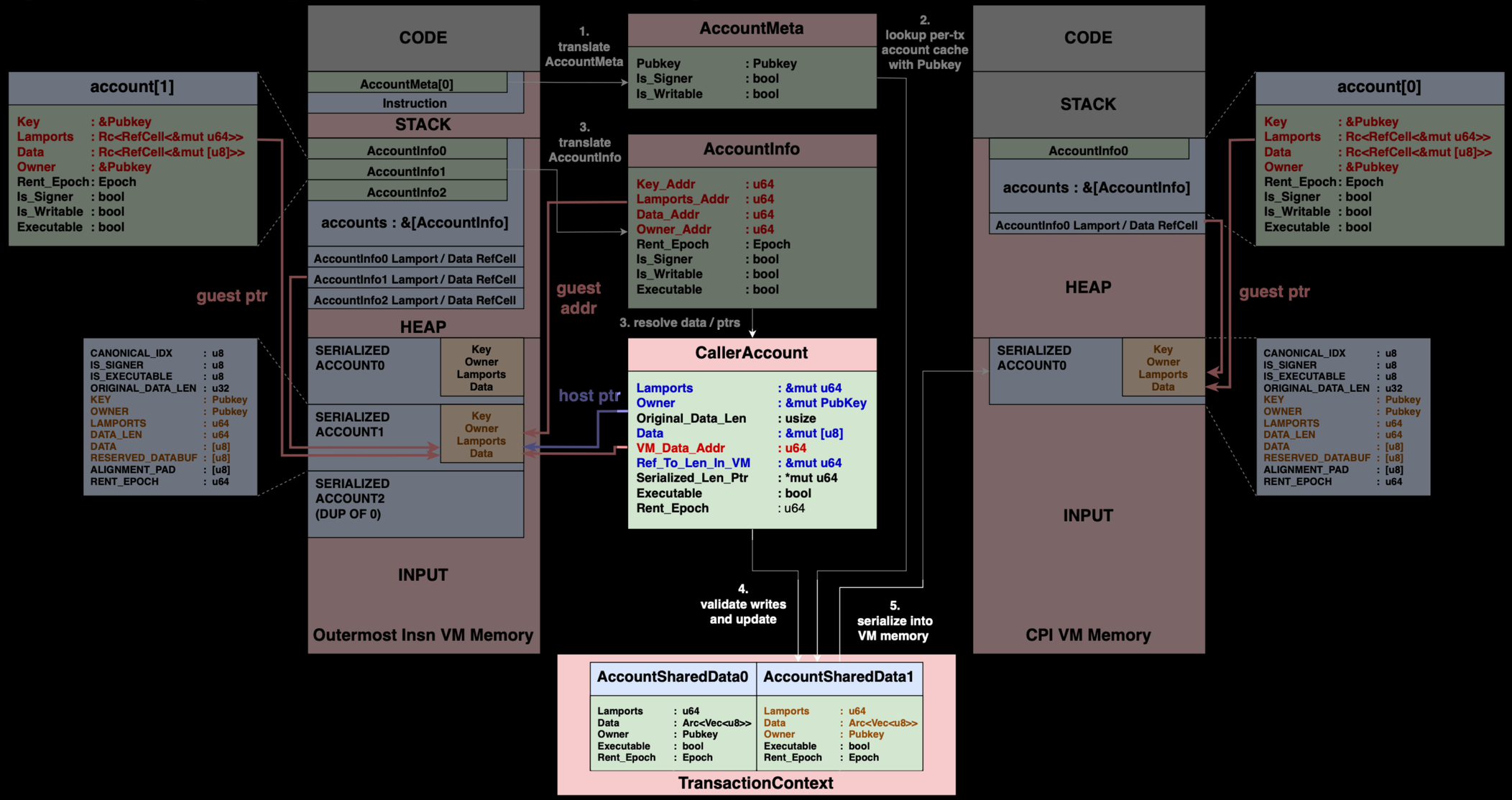
Legacy Interoperability

cpi (cross-program-invocation) call



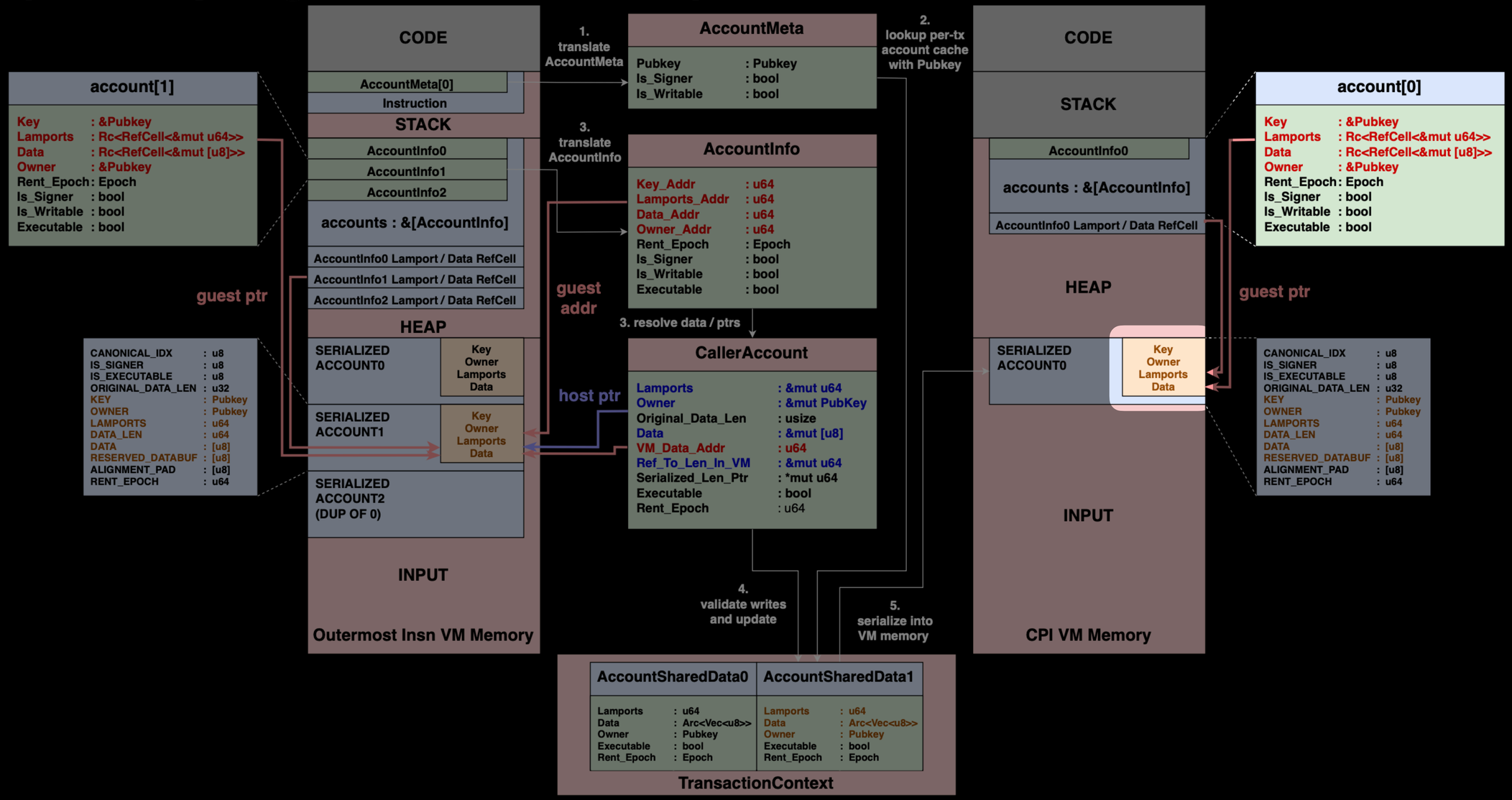
Legacy Interoperability

cpi (cross-program-invocation) call



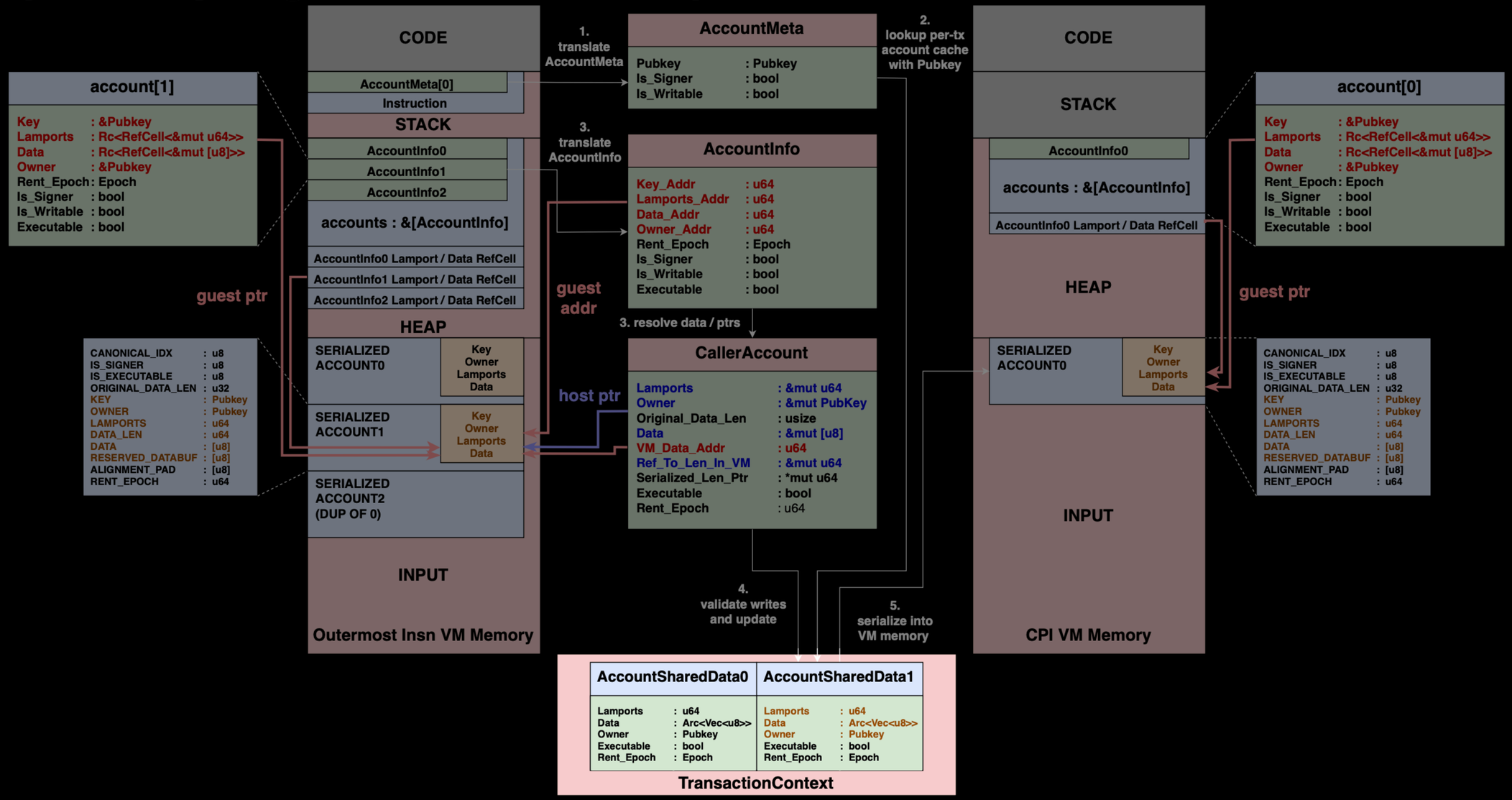
Legacy Interoperability

cpi (cross-program-invocation) call



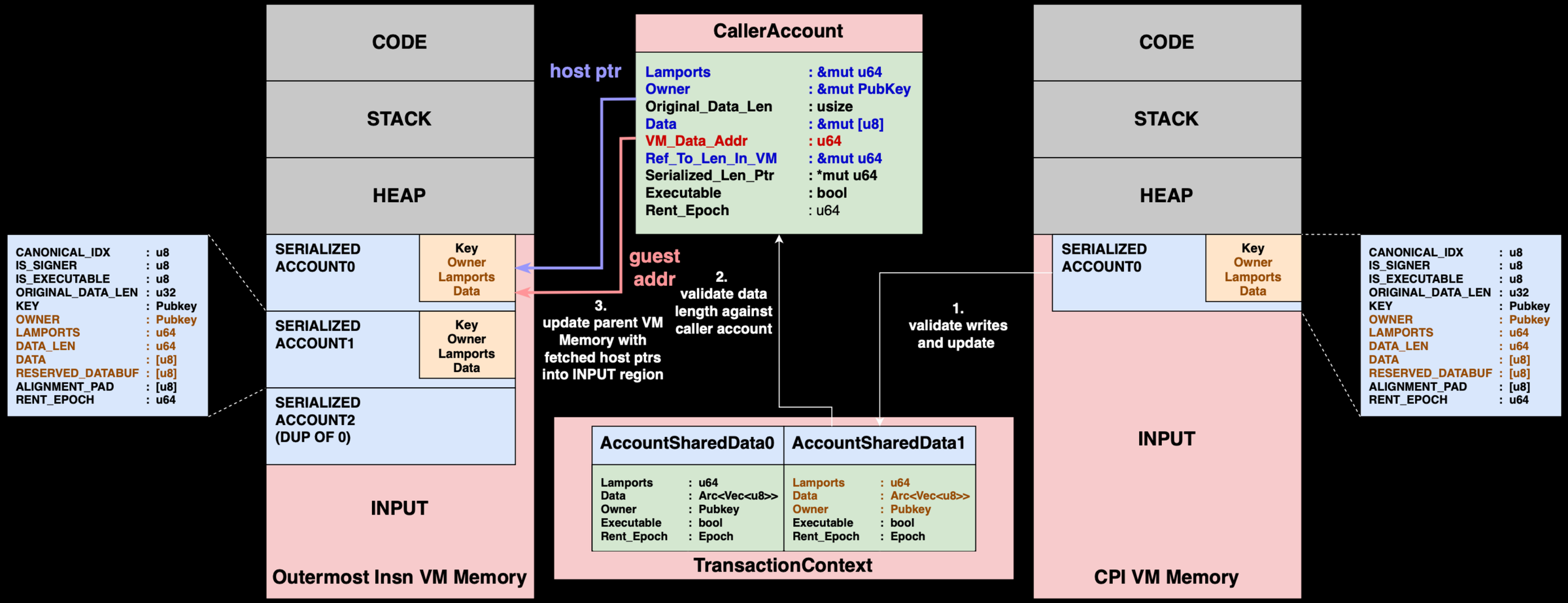
Legacy Interoperability

cpi (cross-program-invocation) call



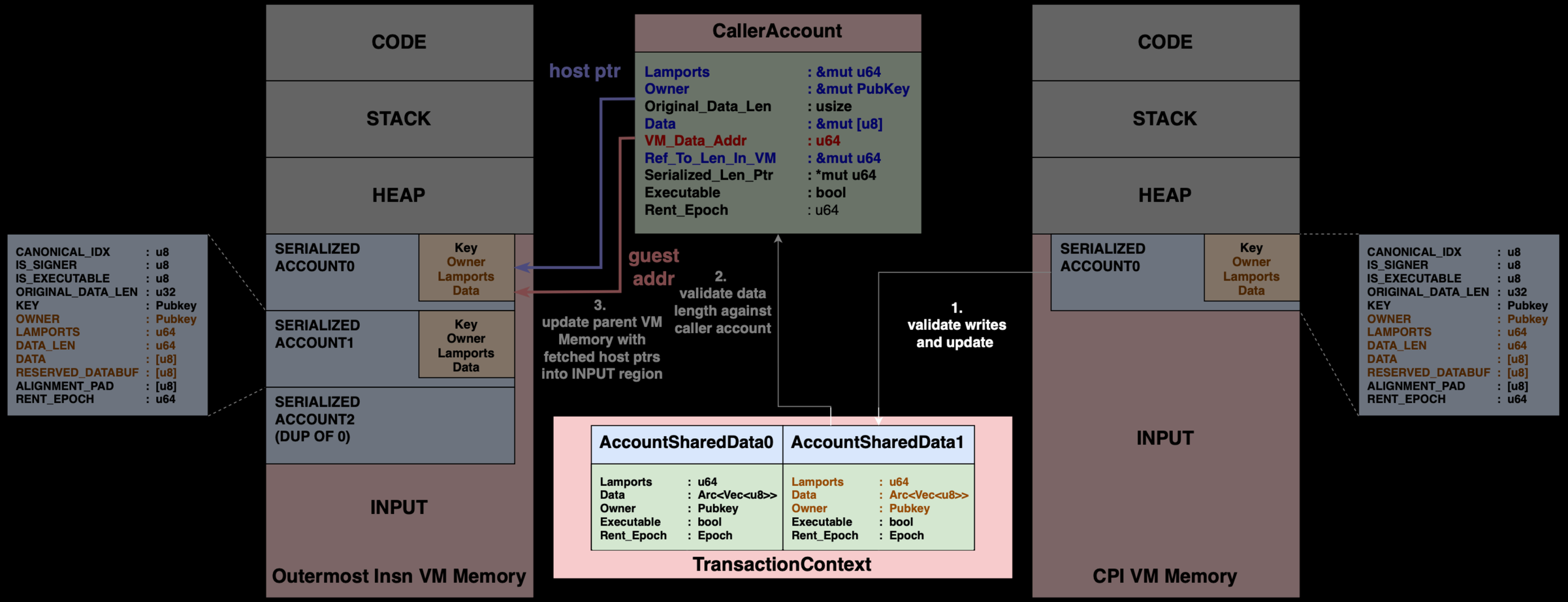
Legacy Interoperability

cpi return



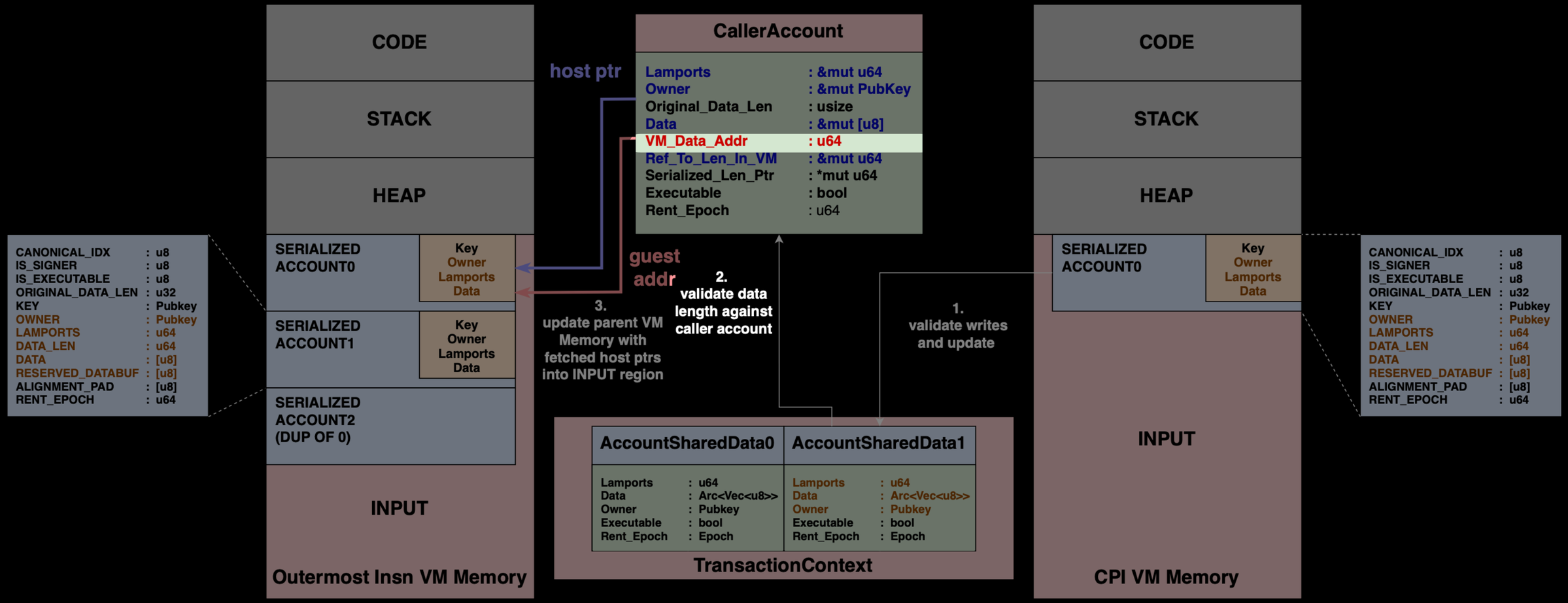
Legacy Interoperability

cpi return



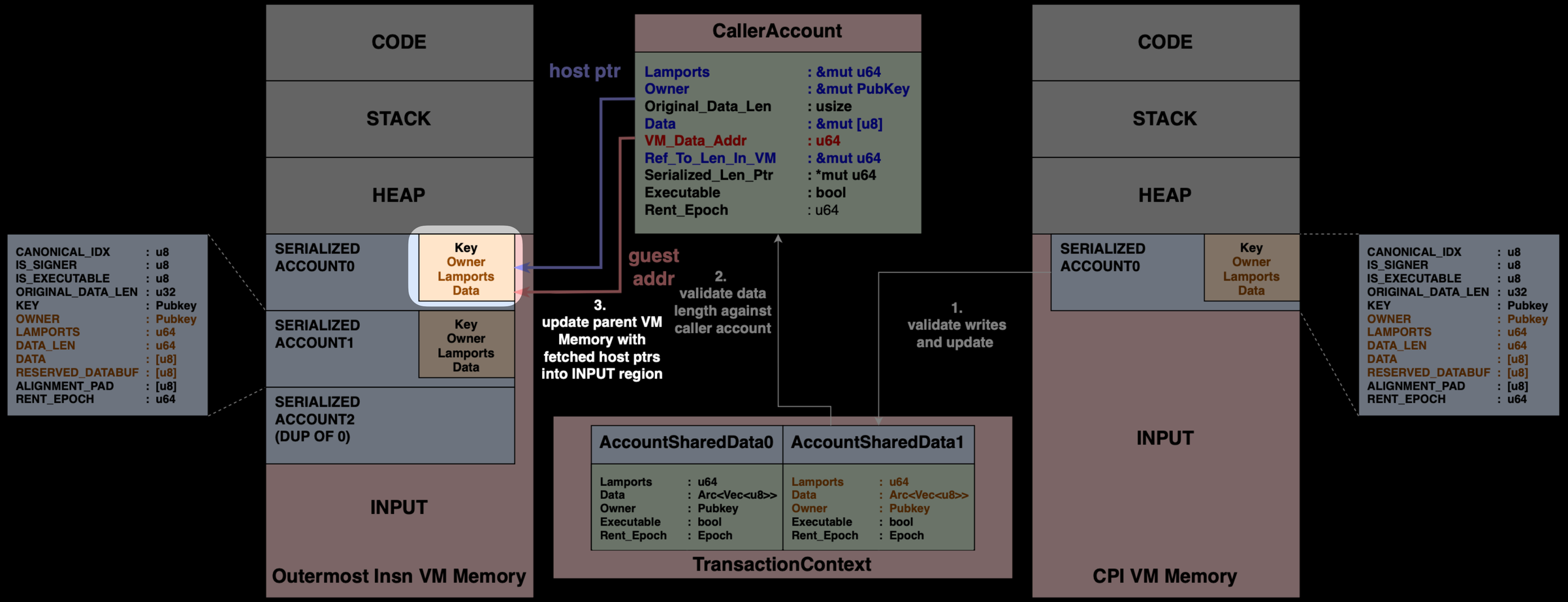
Legacy Interoperability

cpi return



Legacy Interoperability

cpi return



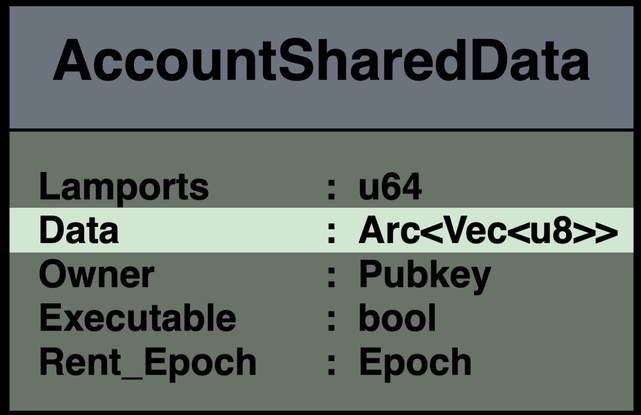
It works! But At What Cost?

Data copies are costly

Direct Mapping

Direct Mapping

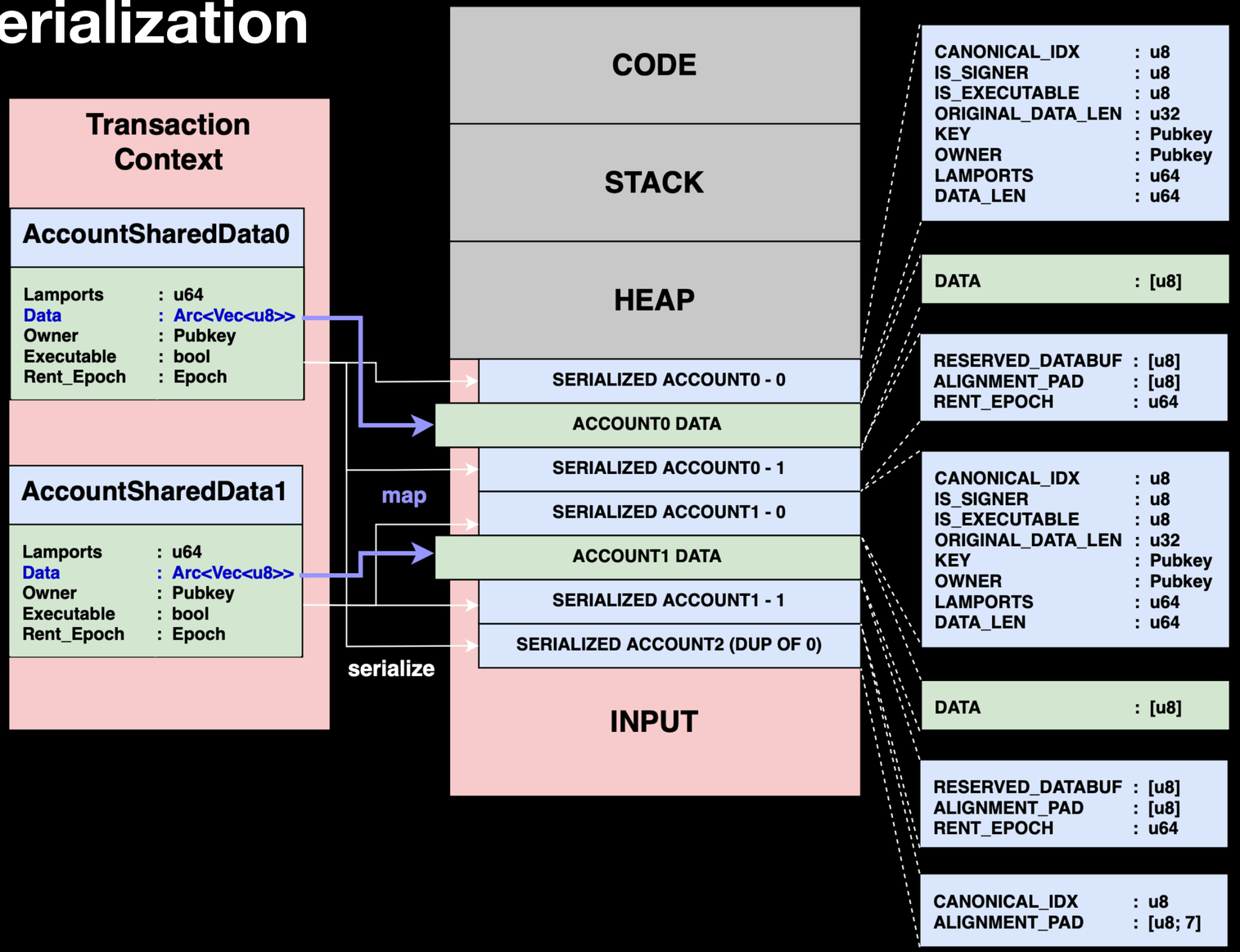
- Introduced in v1.16.0
- Avoid copying AccountSharedData.data
- Copy on Write only



New Data Exposure

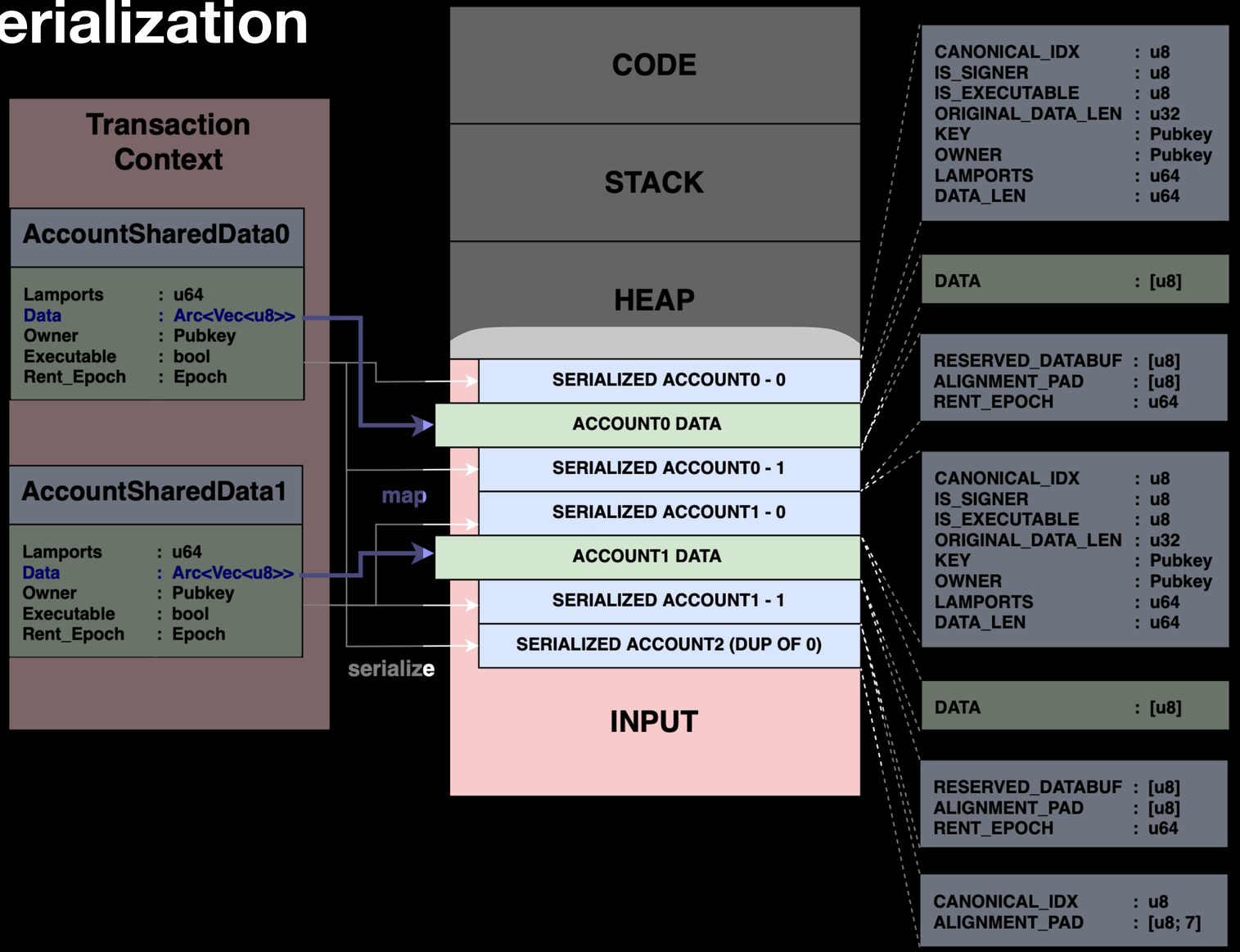
New Data Exposure

Accounts serialization



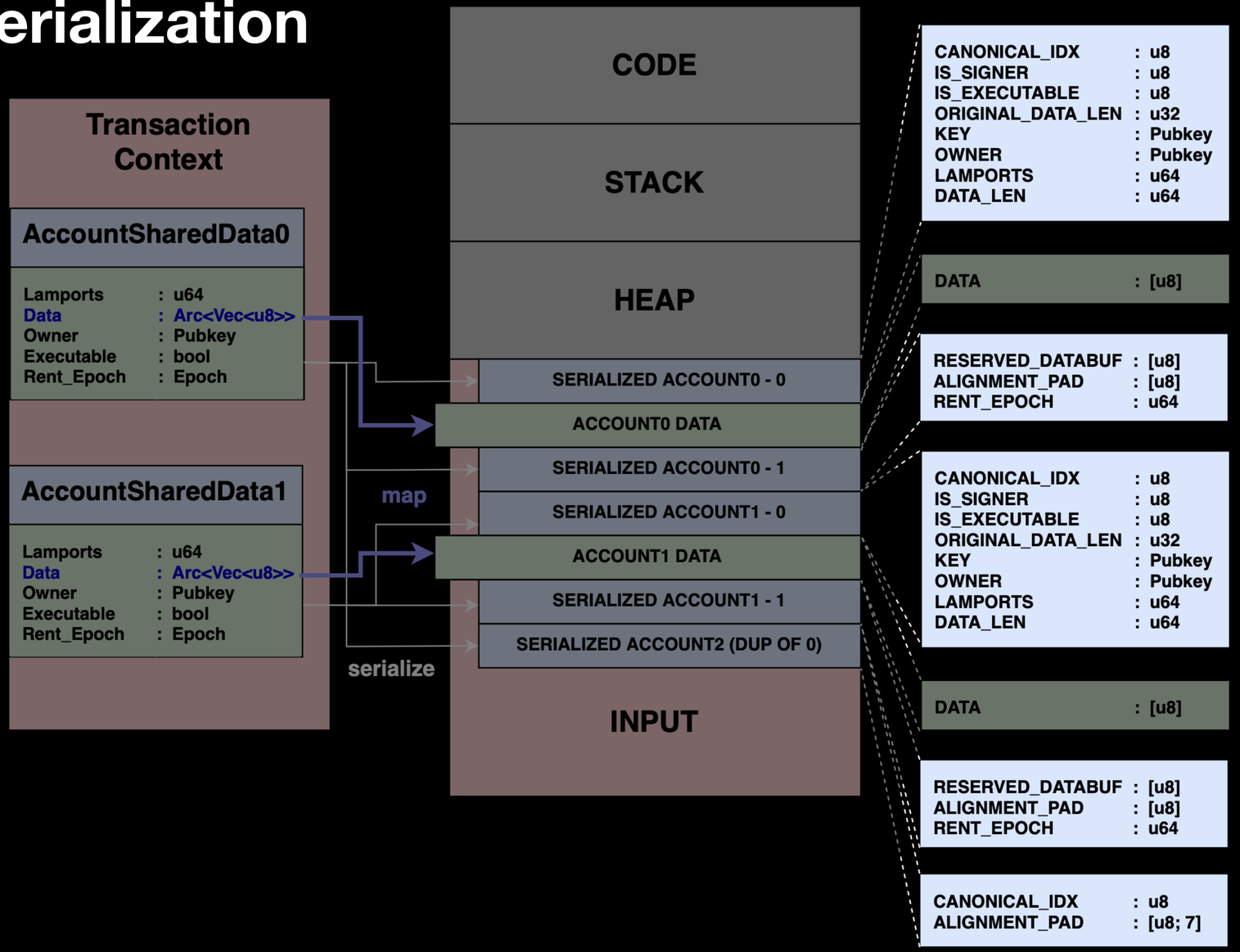
New Data Exposure

Accounts serialization



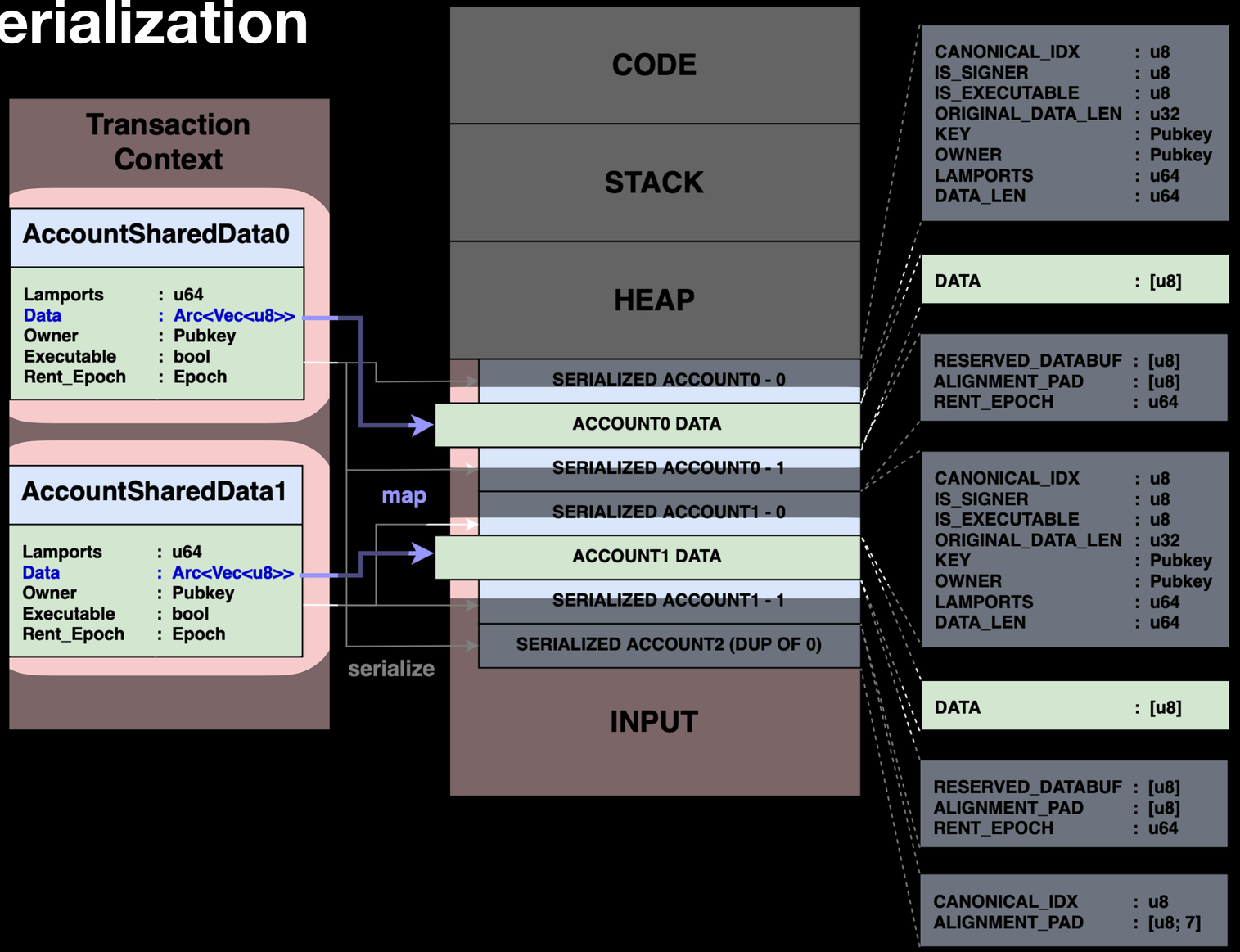
New Data Exposure

Accounts serialization



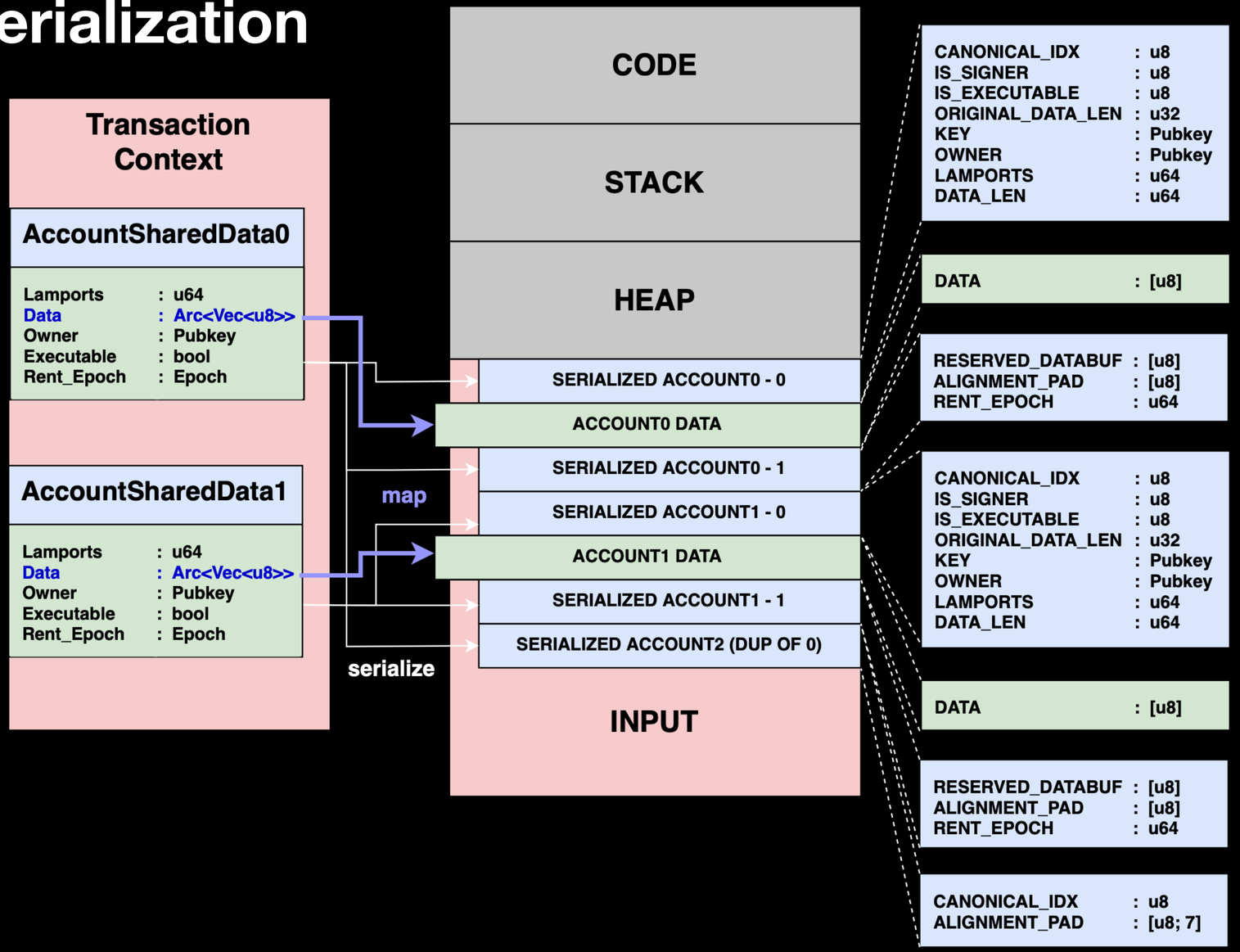
New Data Exposure

Accounts serialization



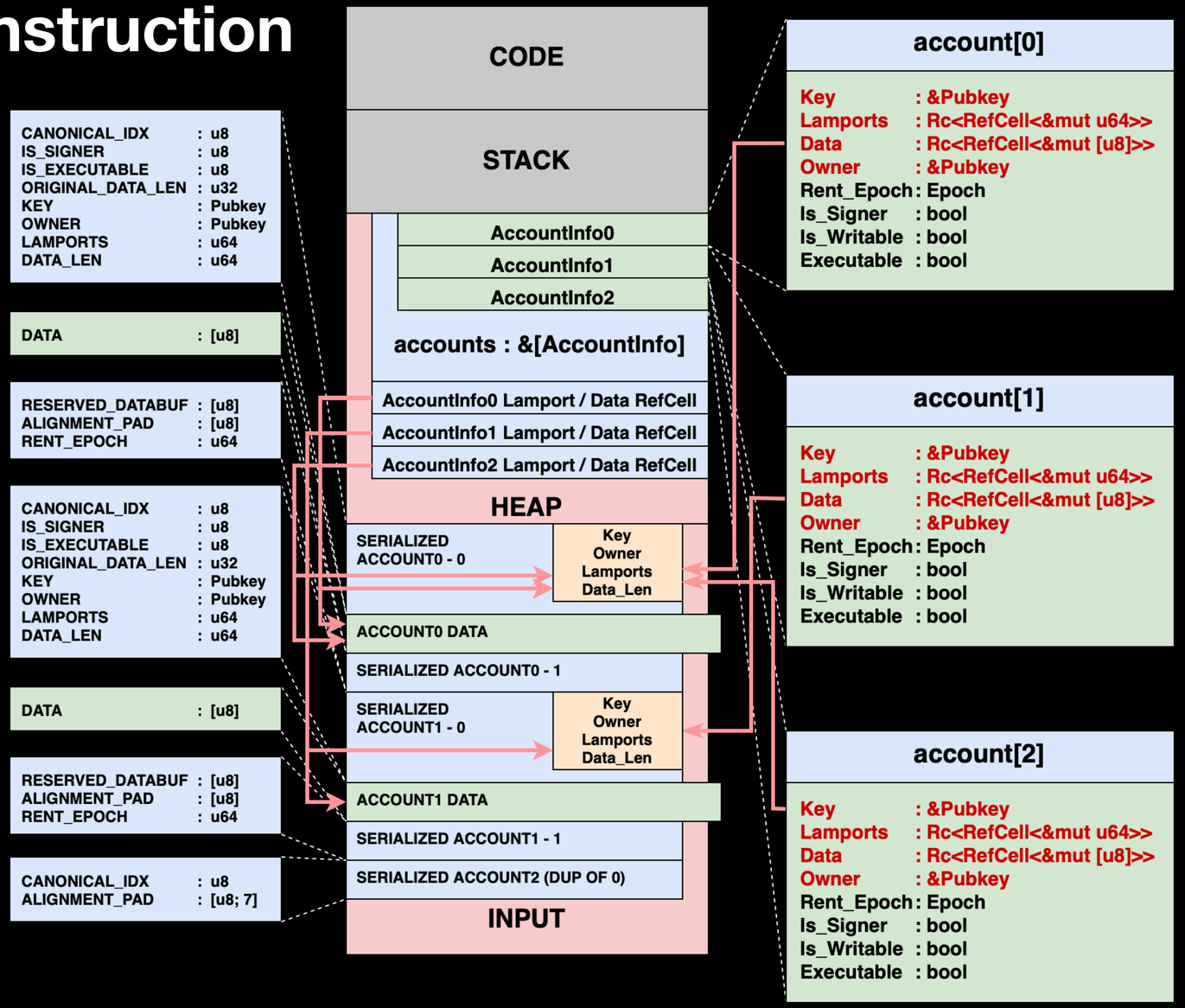
New Data Exposure

Accounts serialization



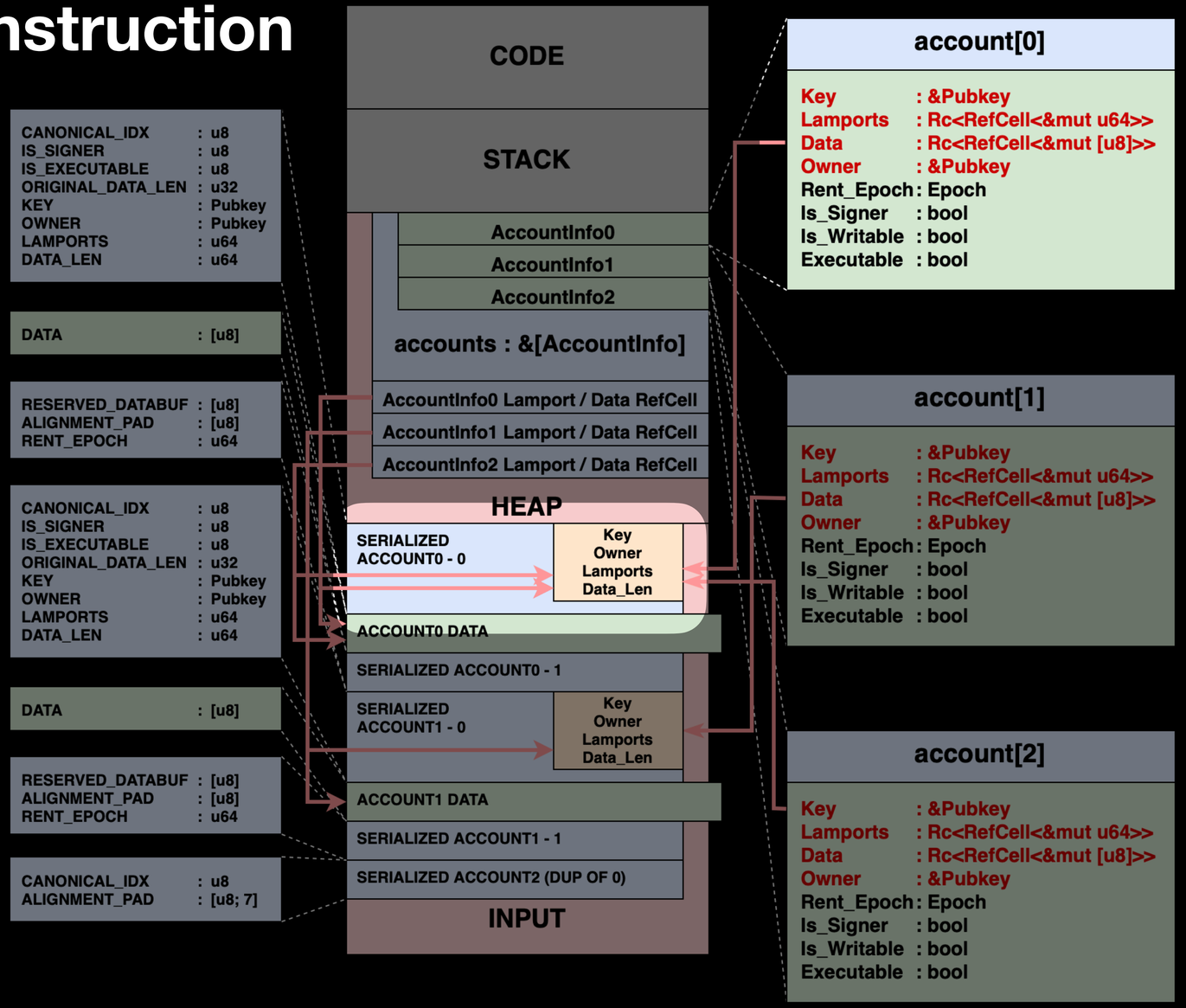
New Data Exposure

AccountInfo construction



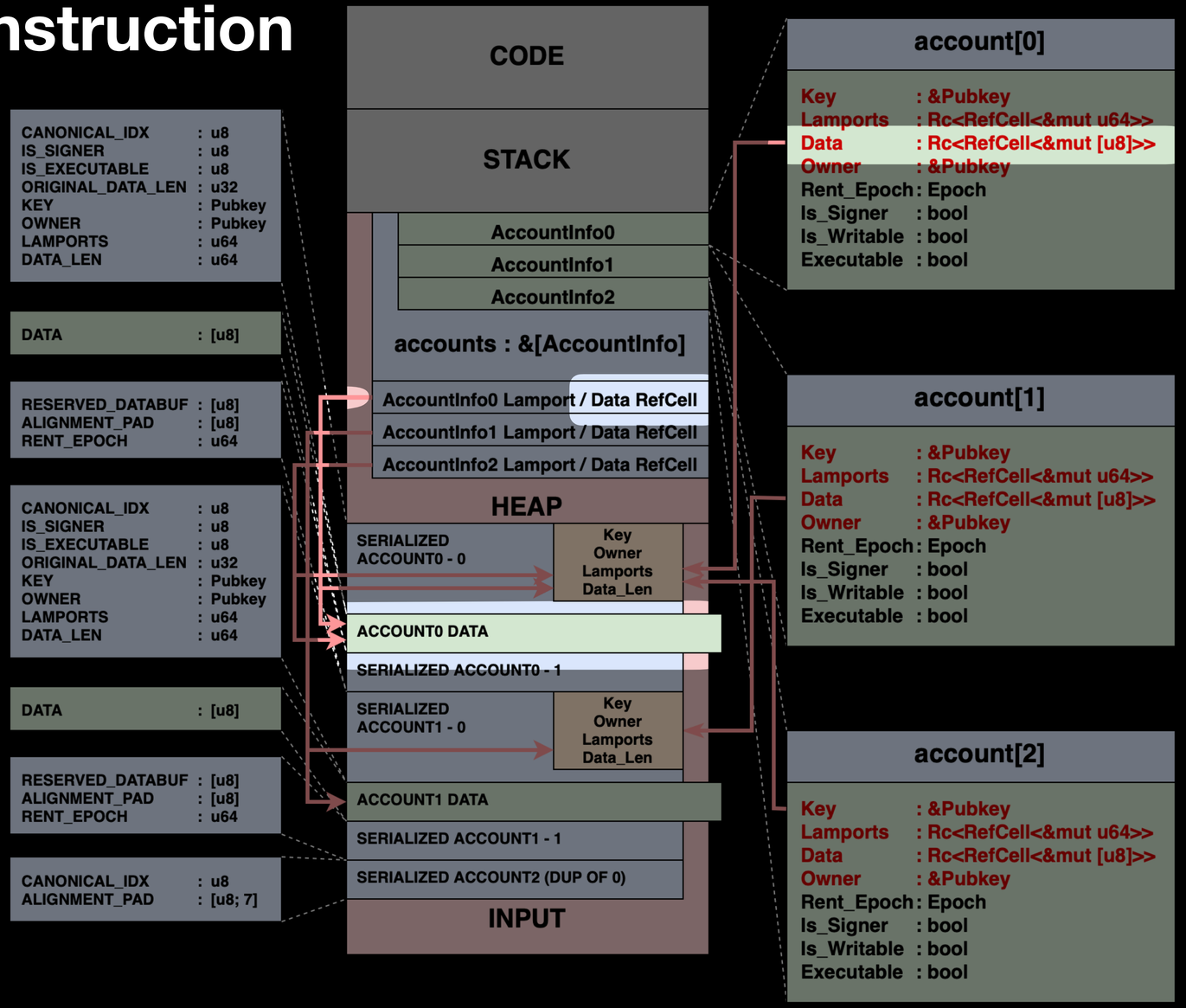
New Data Exposure

AccountInfo construction



New Data Exposure

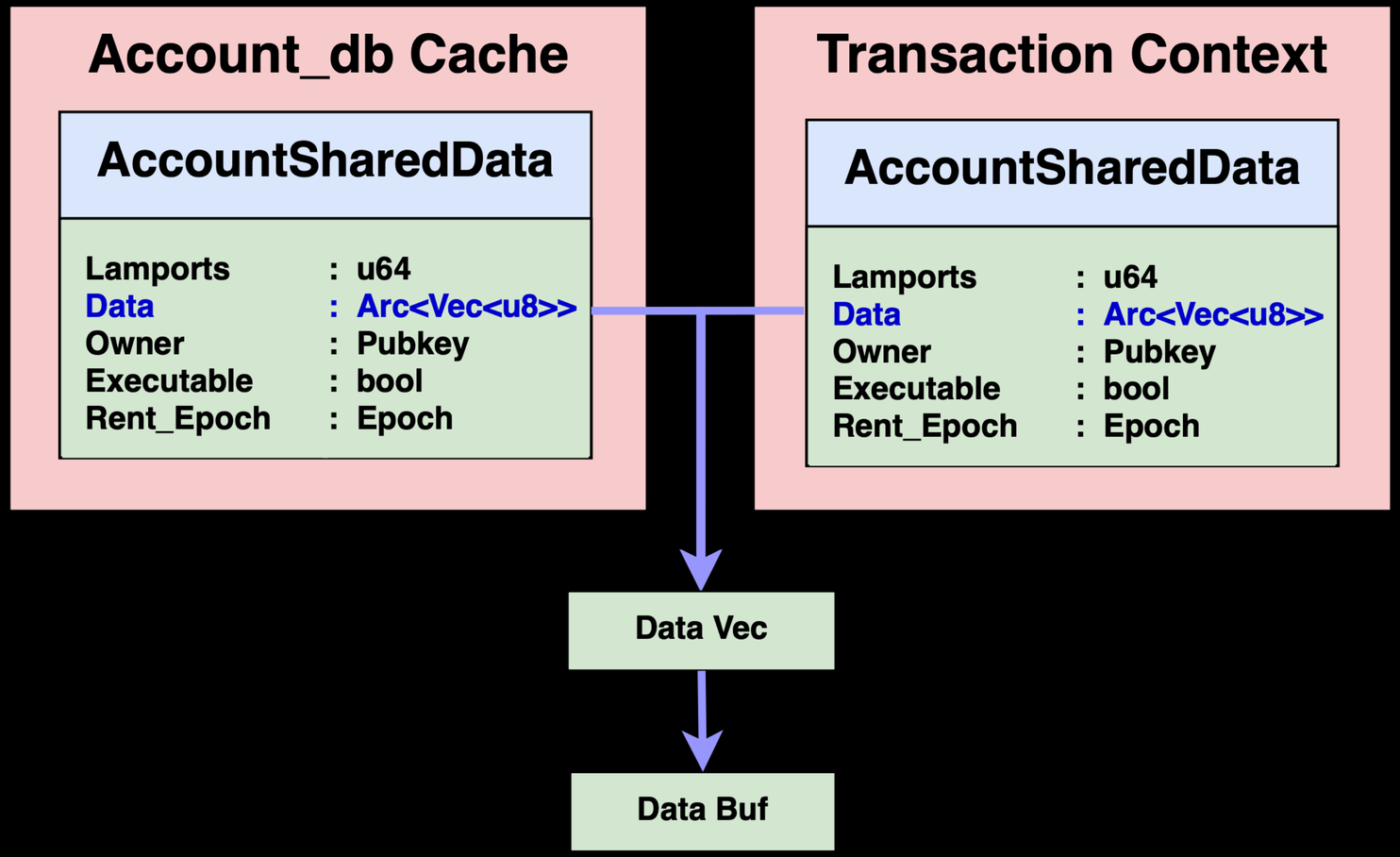
AccountInfo construction



CoW (Copy on Write)

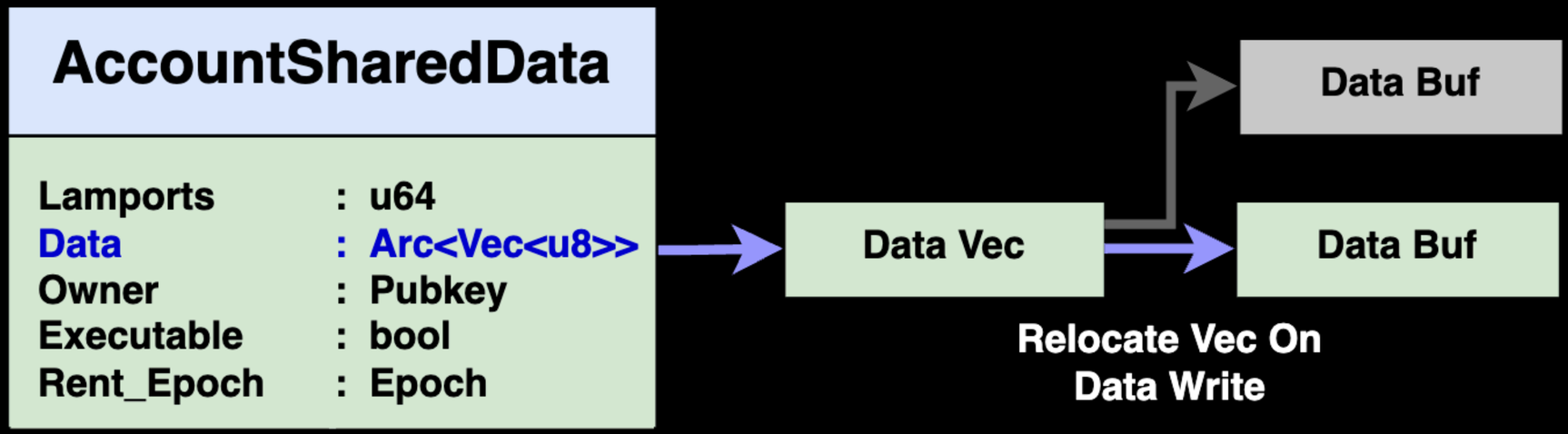
“Shared” Account Data

AccountSharedData.Data



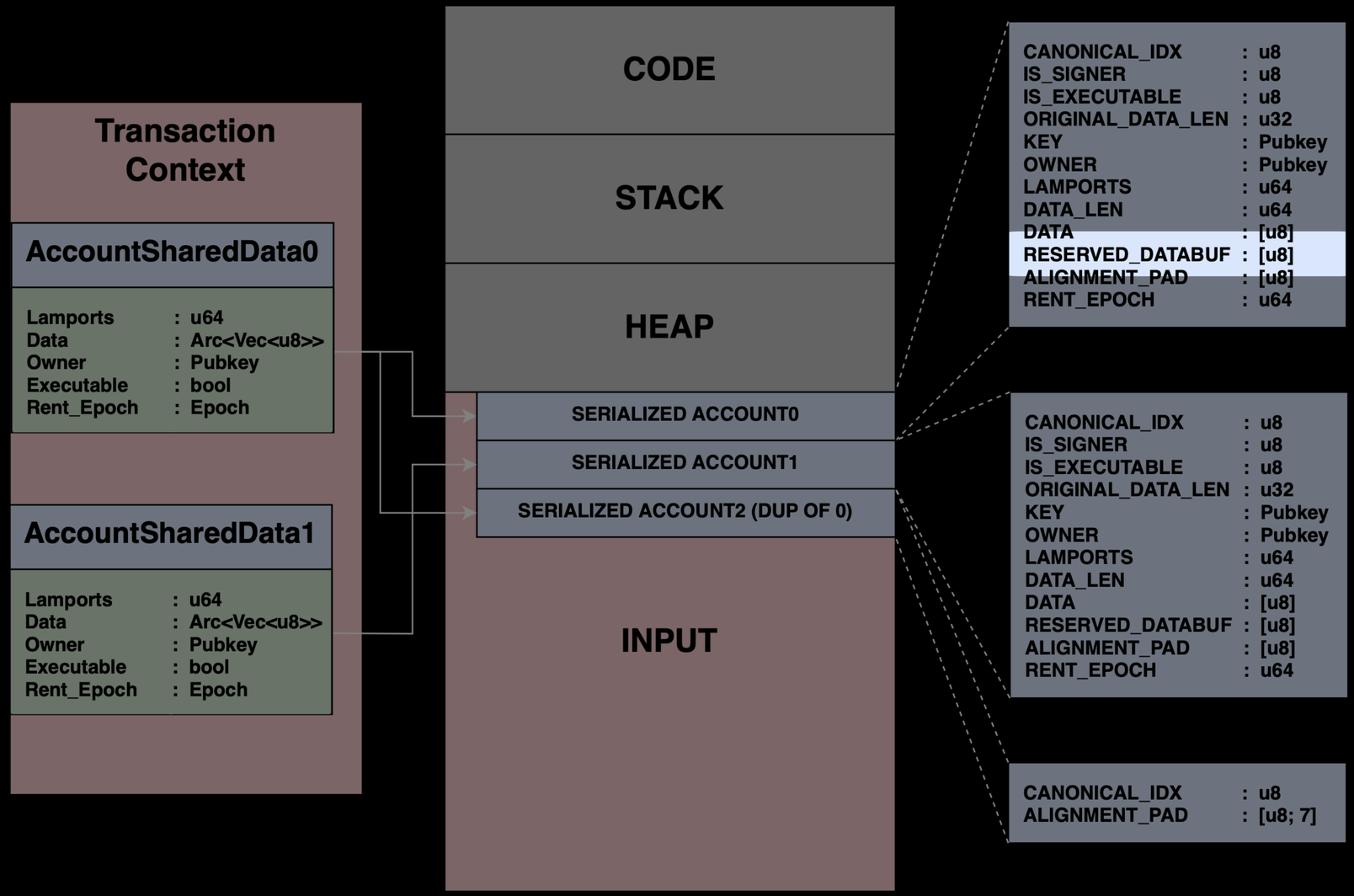
Legacy CoW

Data relocation



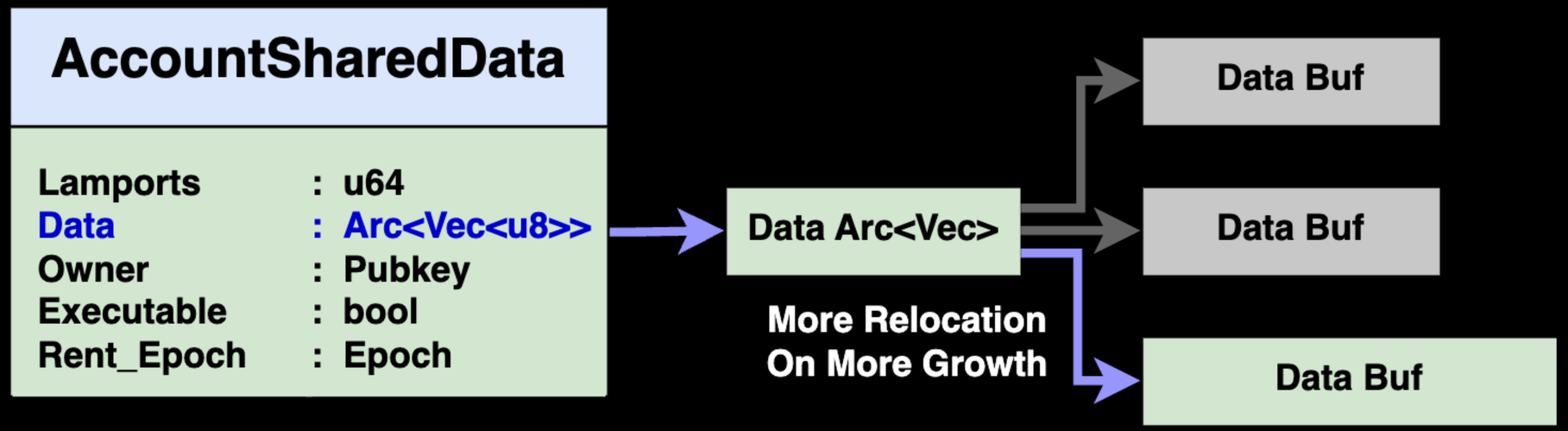
Legacy CoW

Reserved data buffer



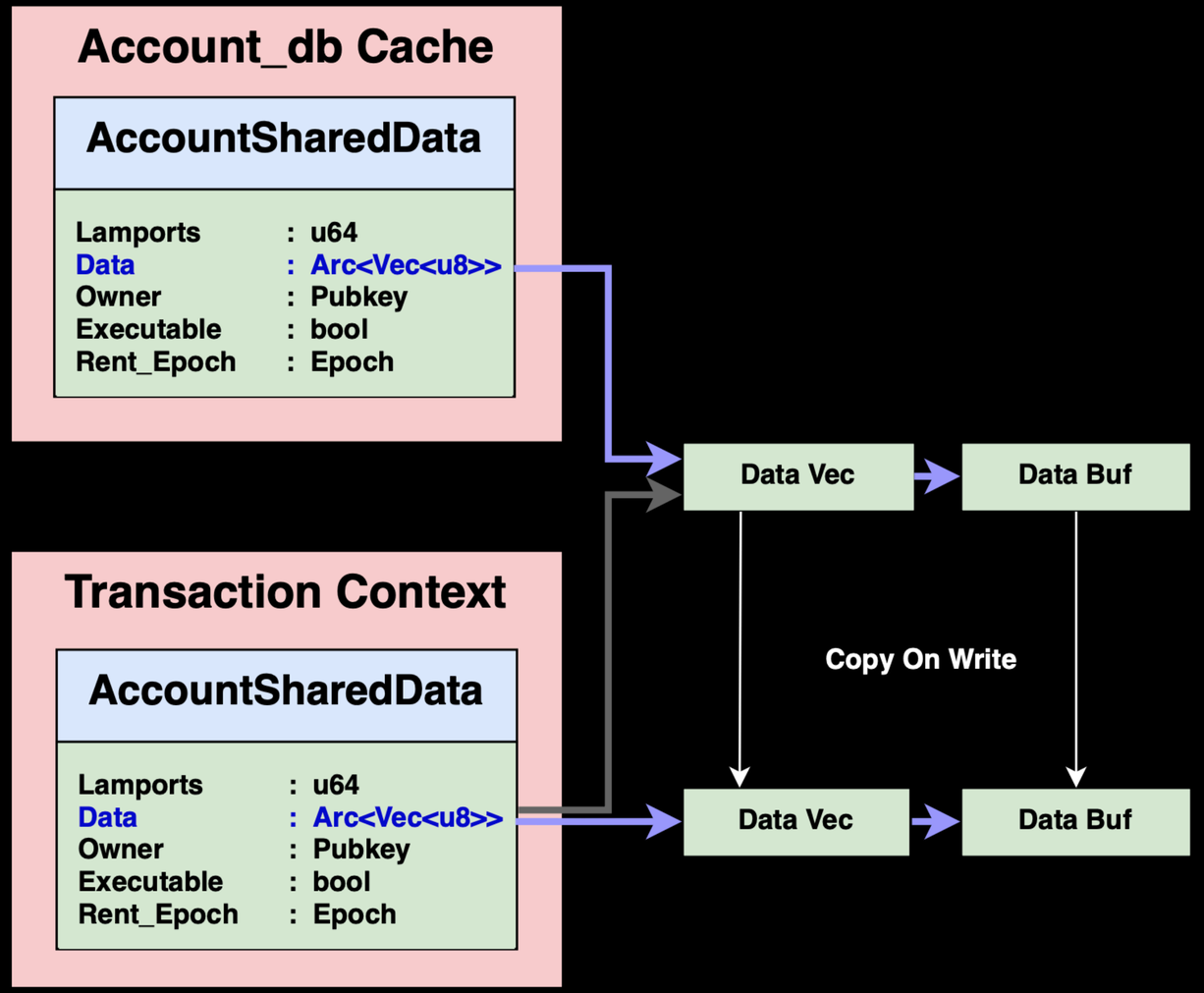
Legacy CoW

Data relocation



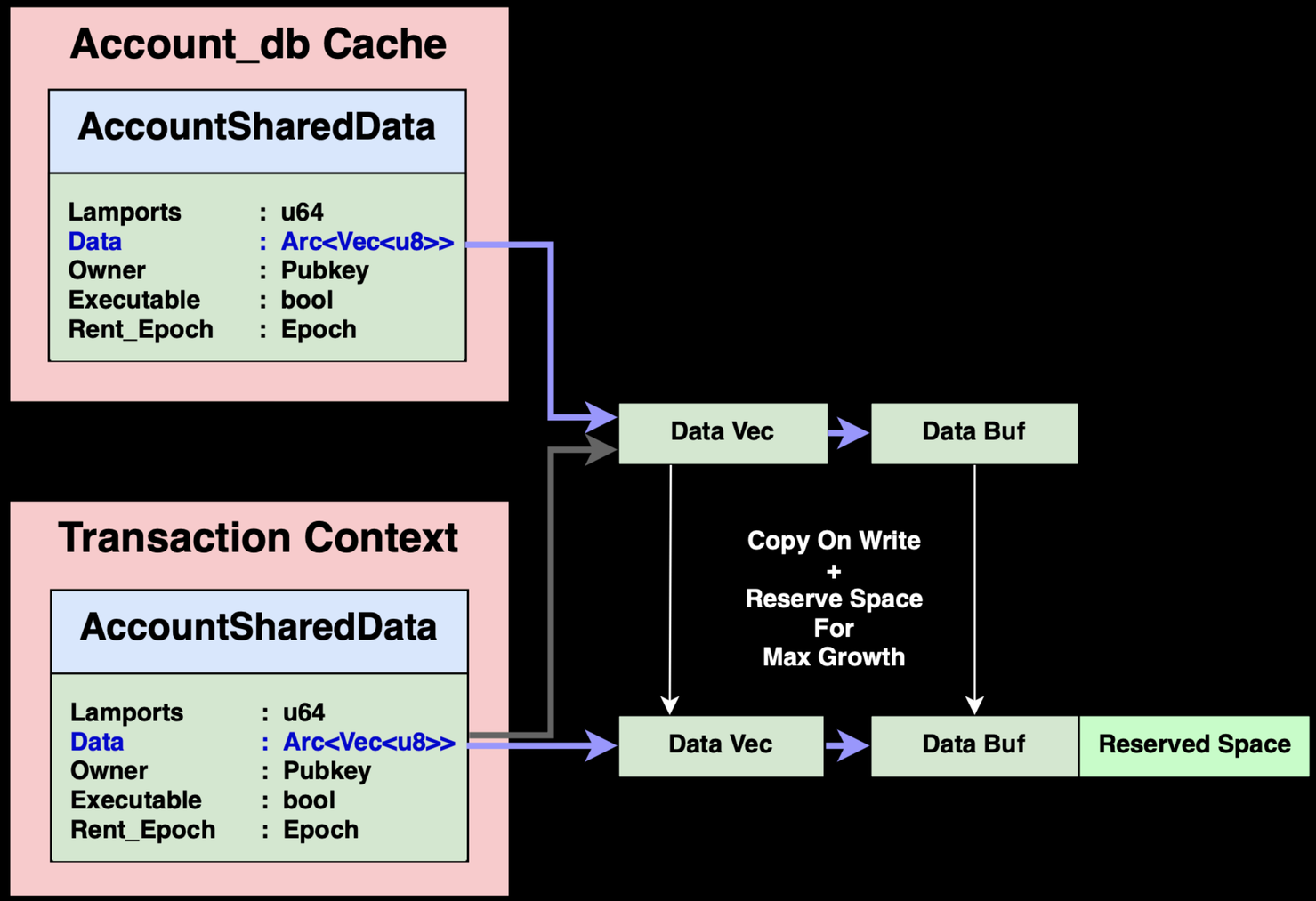
Legacy CoW

Data relocation



New CoW

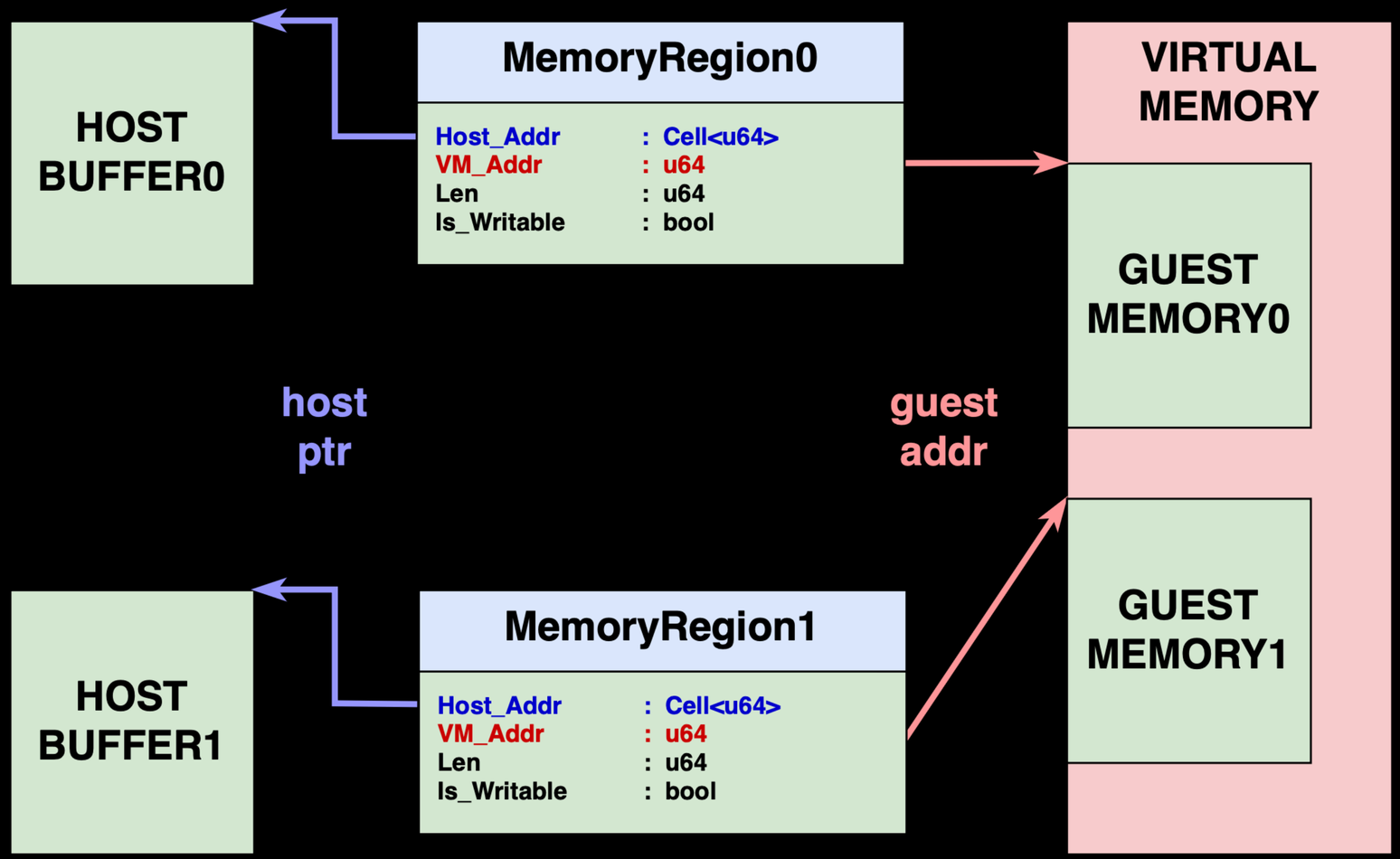
Reserved Space



VM Memory Management

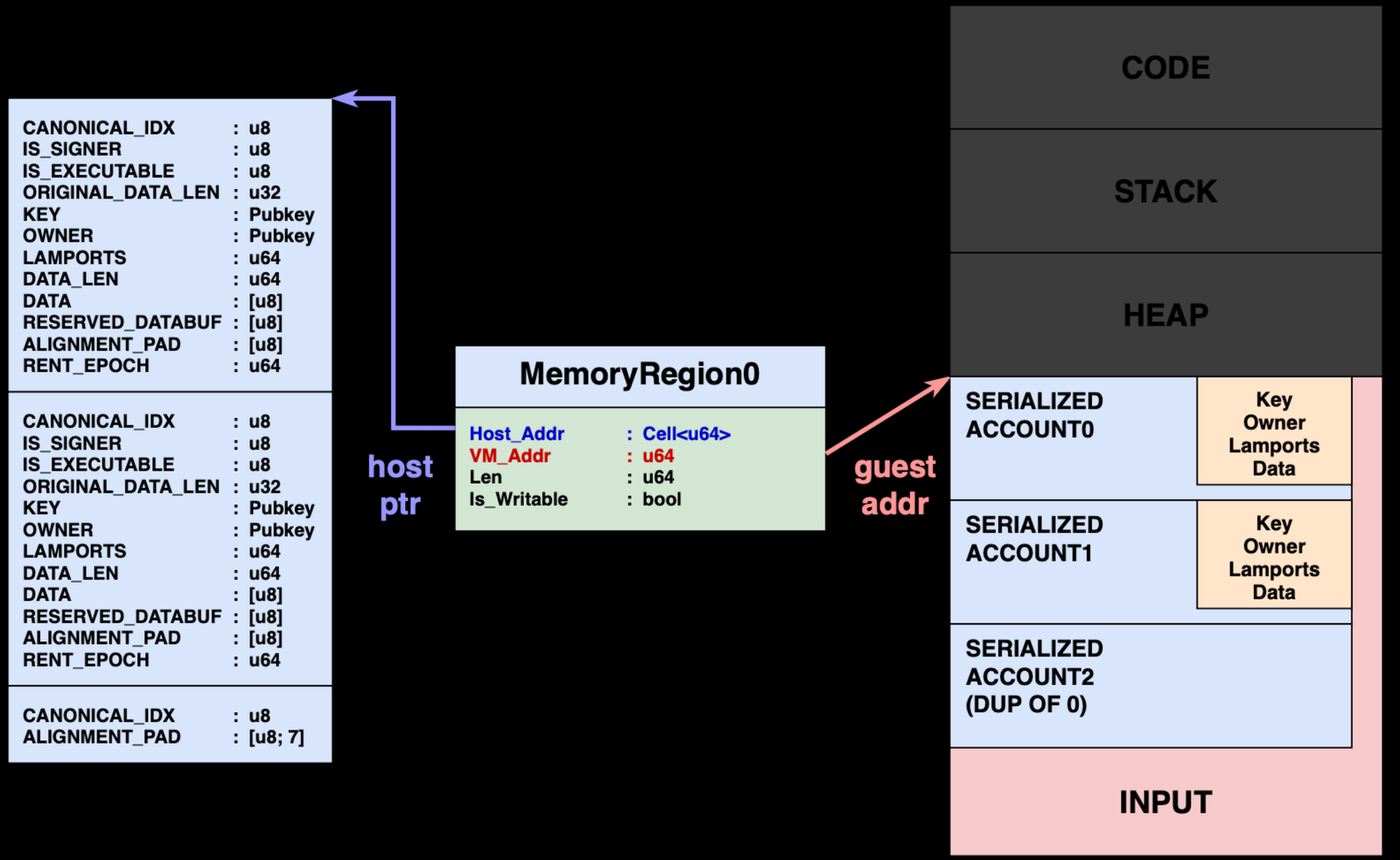
Legacy Memory Regions

Memory Regions



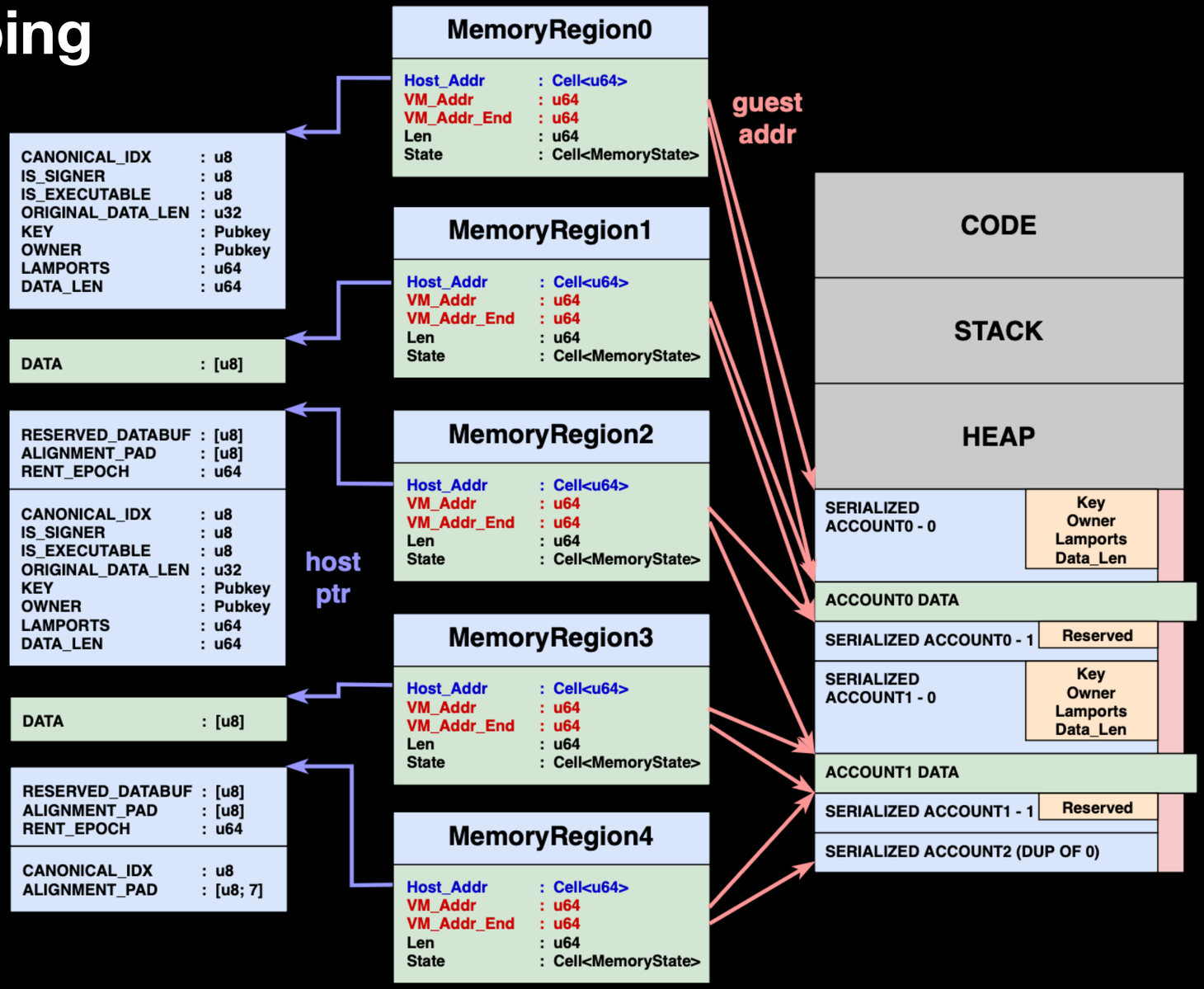
Legacy Memory Regions

Input section



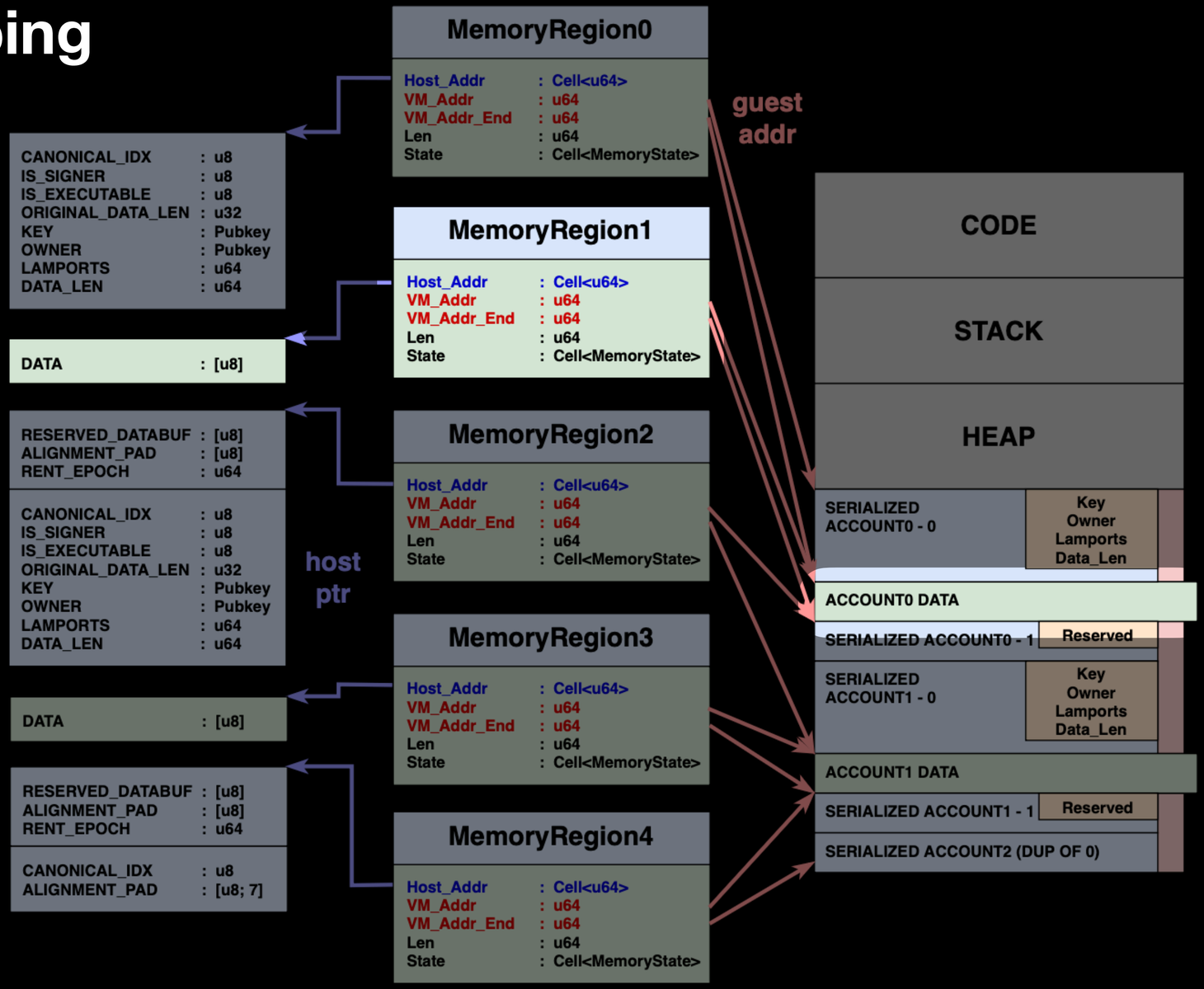
New Memory Regions

Direct Mapping



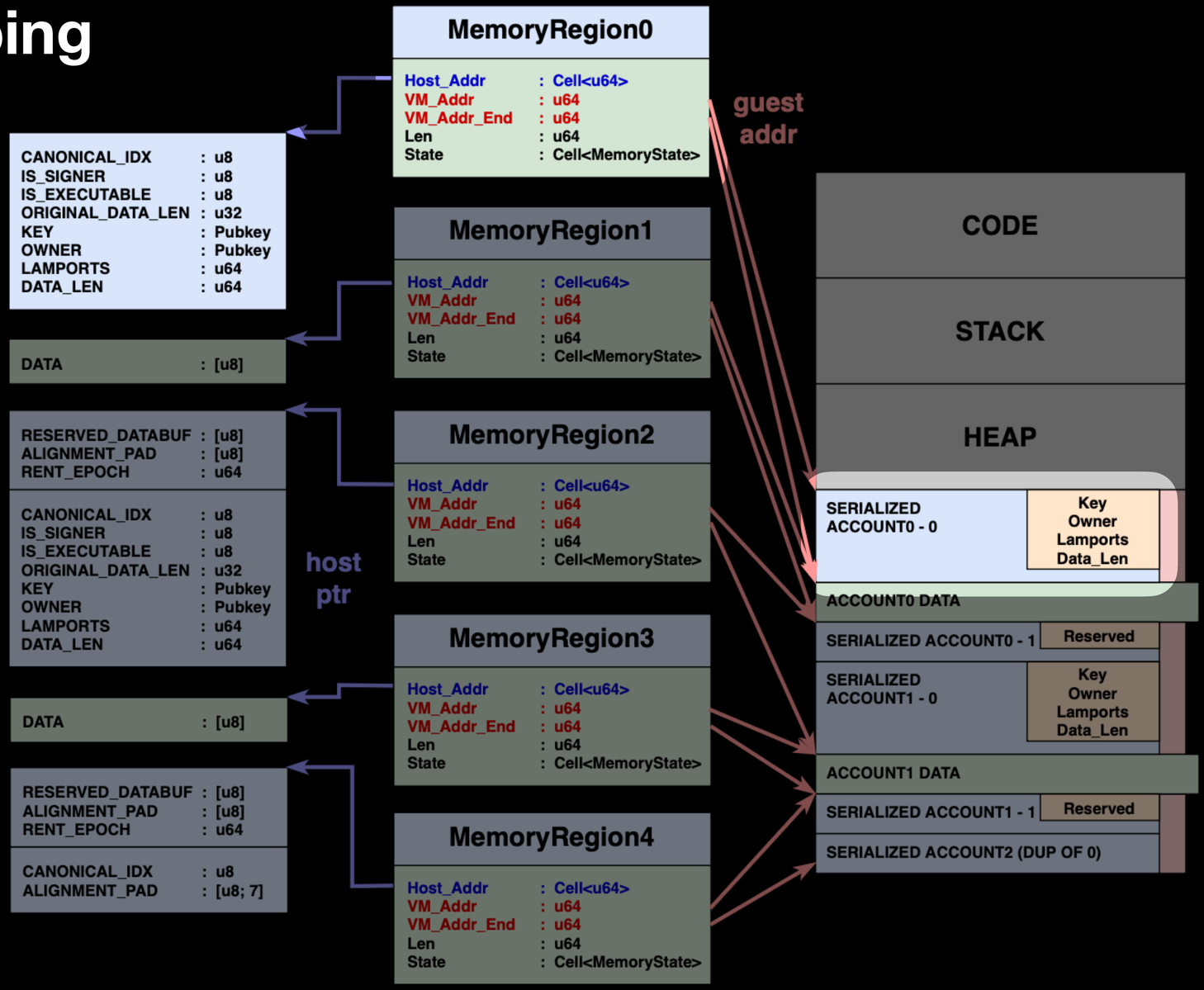
New Memory Regions

Direct Mapping



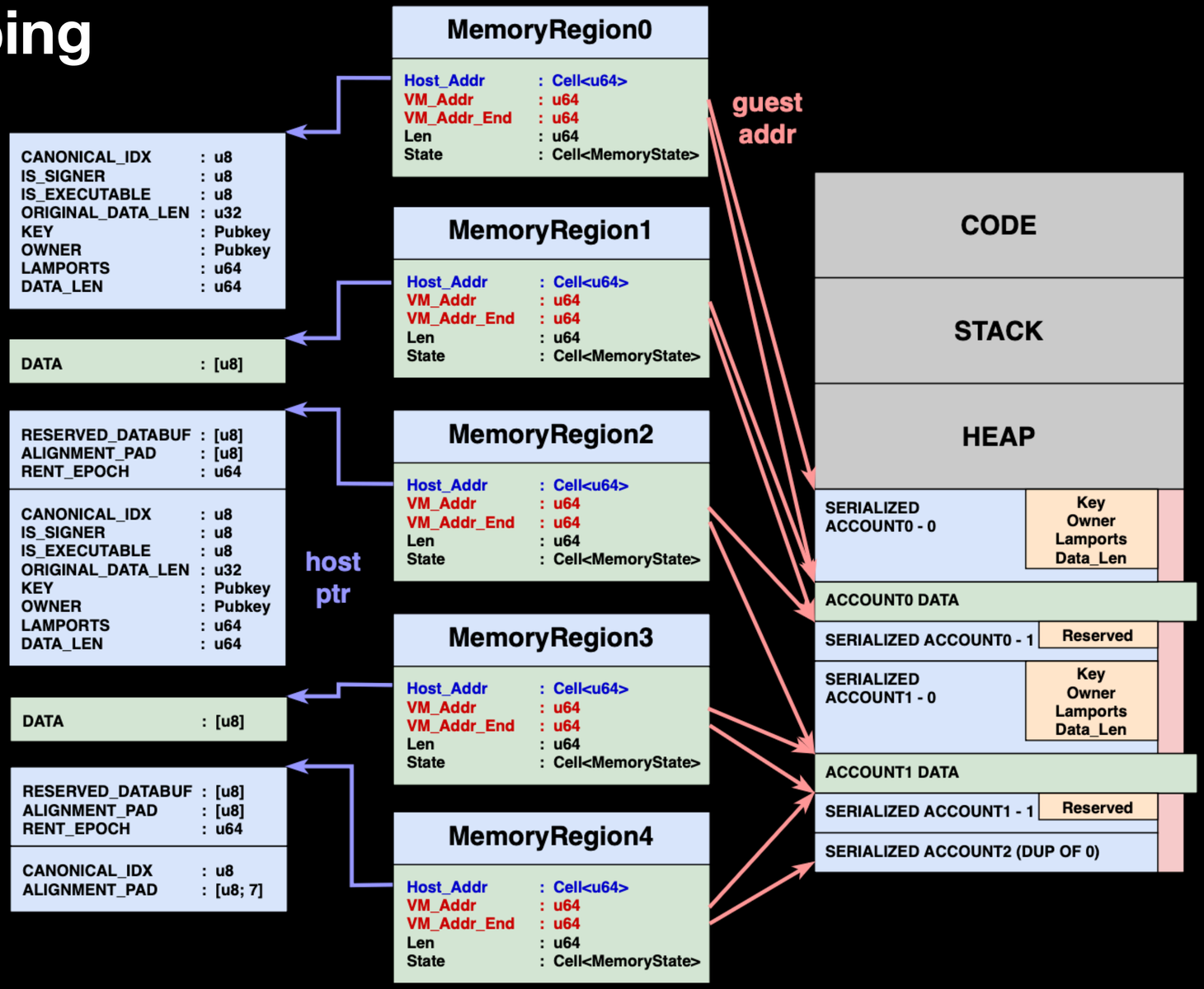
New Memory Regions

Direct Mapping



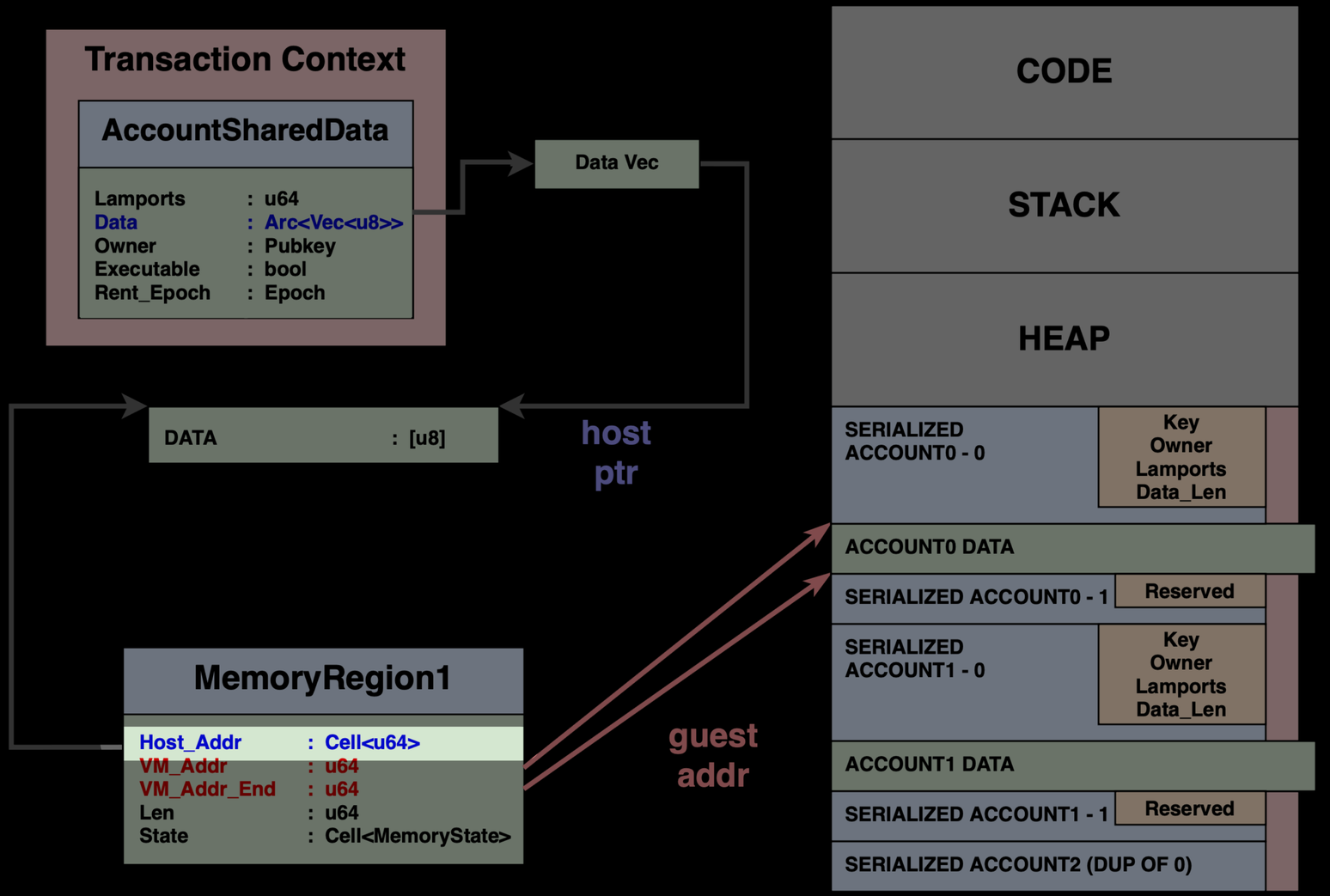
New Memory Regions

Direct Mapping



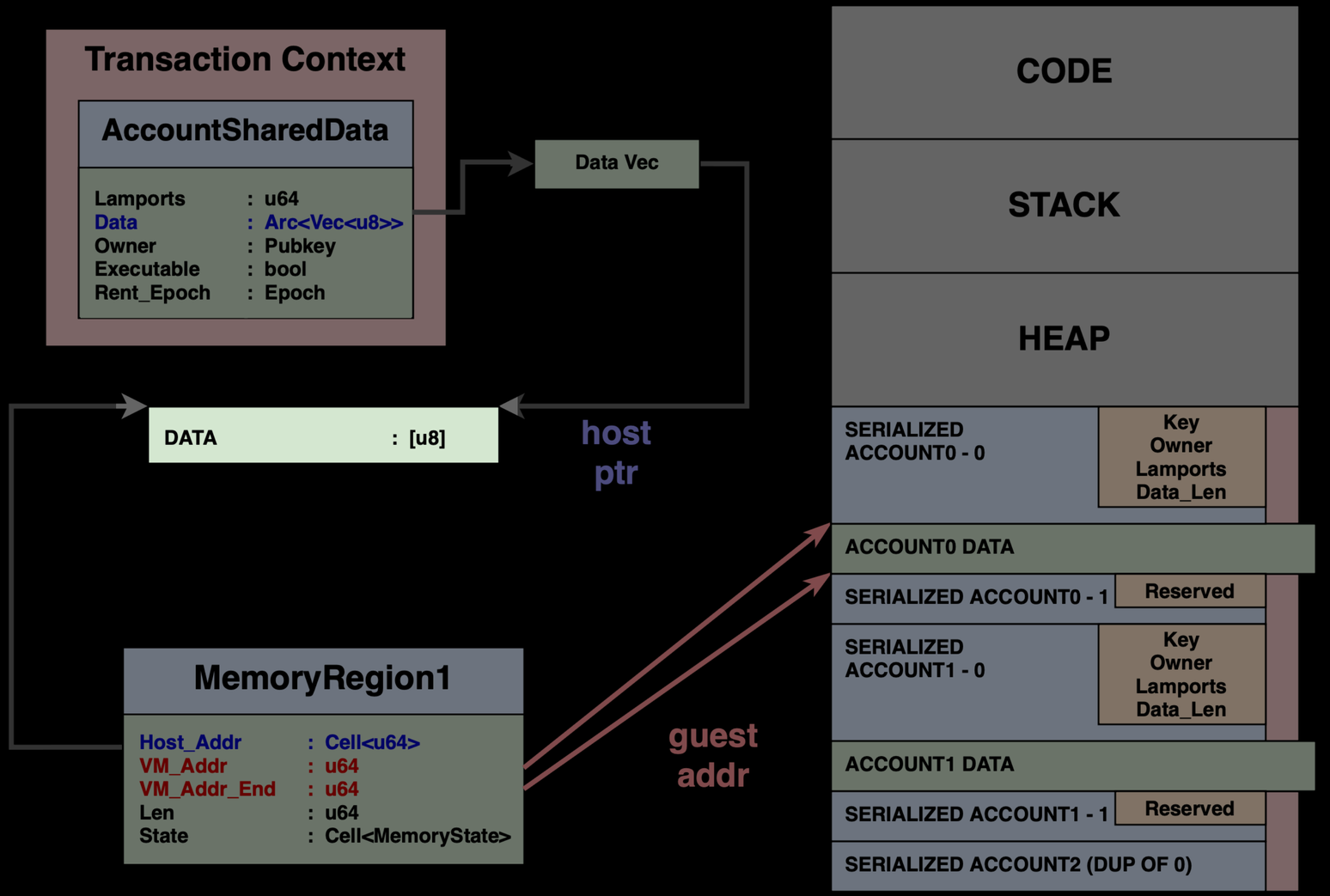
New Memory Regions

Updating Host_Addr



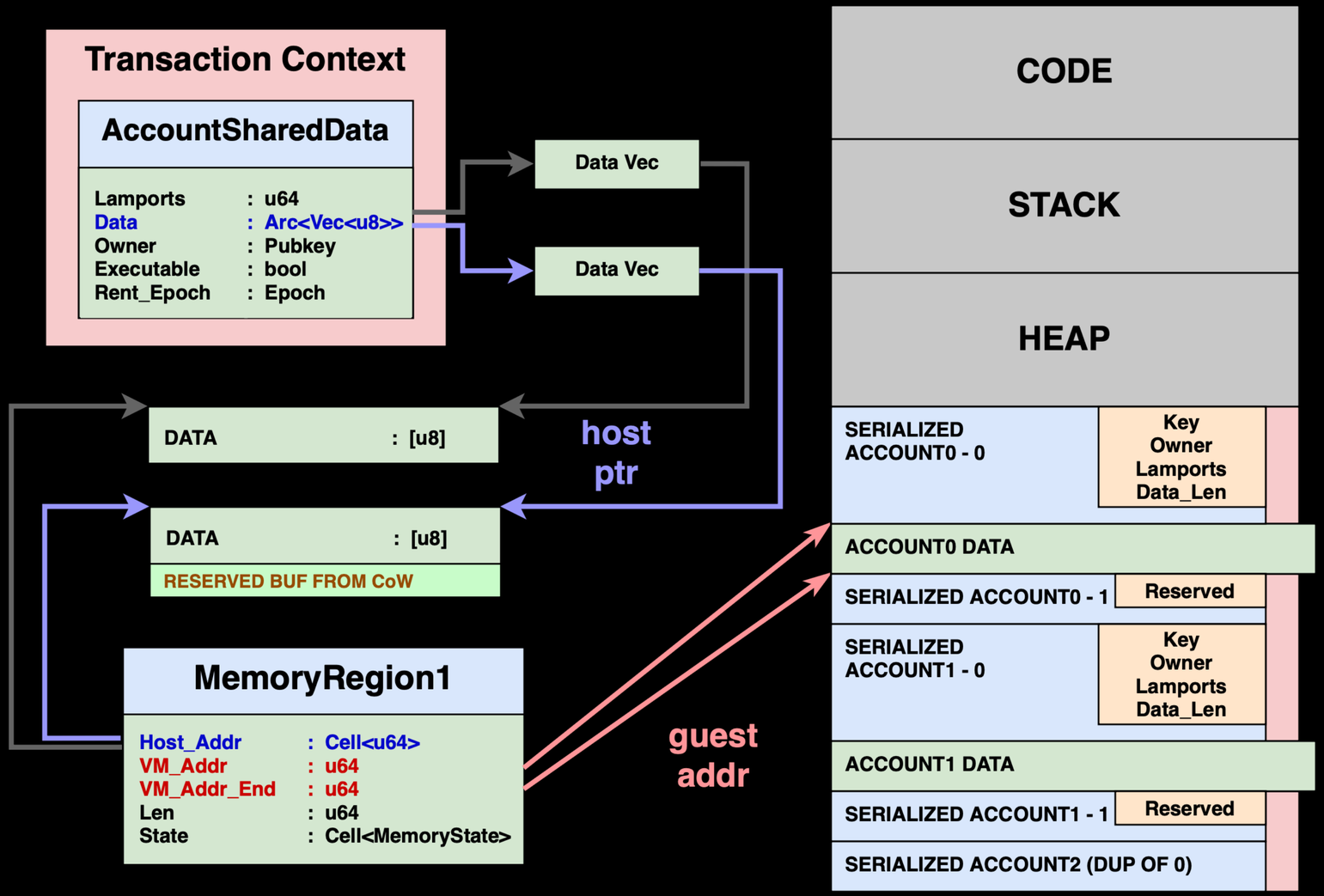
New Memory Regions

Updating Host_Addr



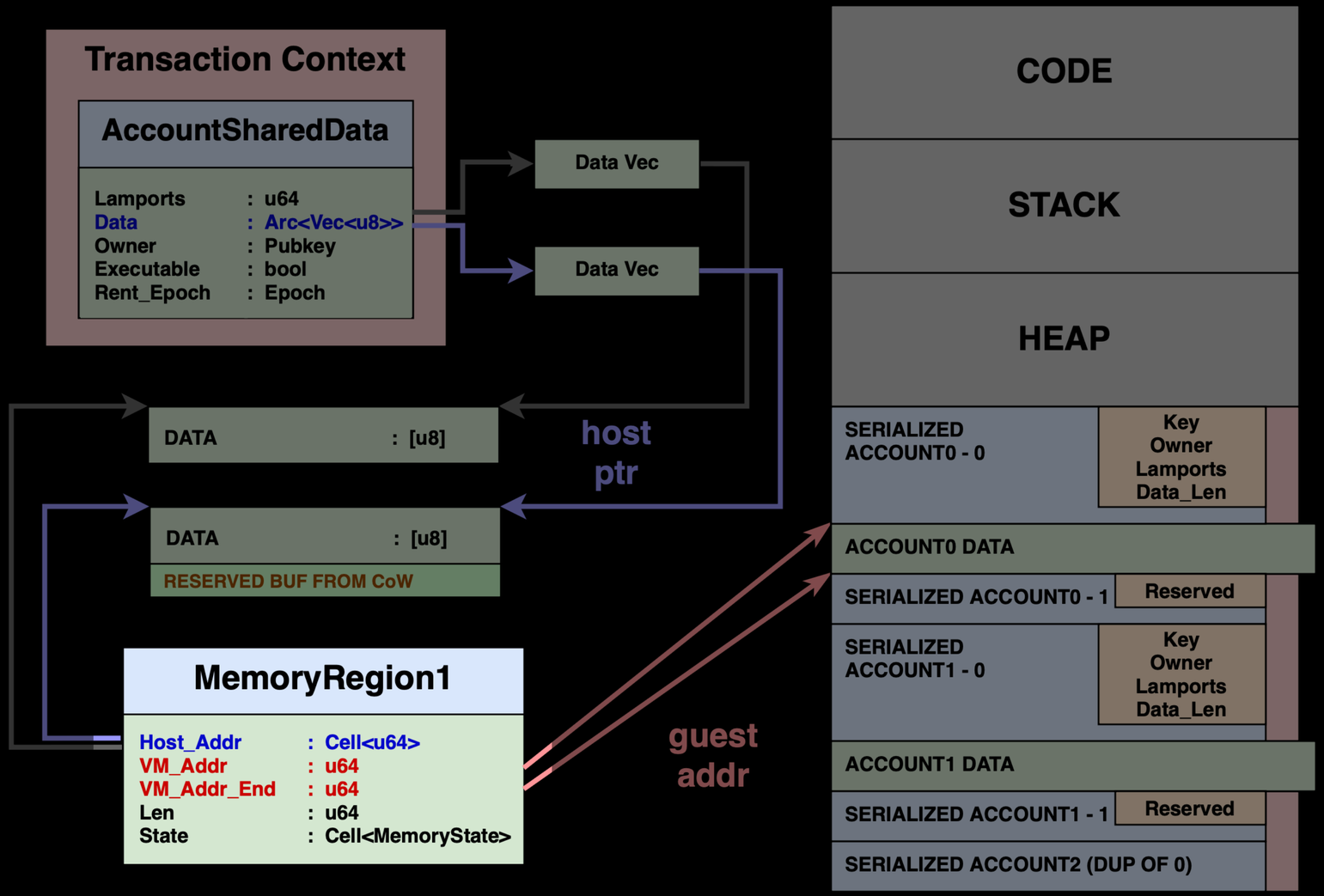
New Memory Regions

Updating Host_Addr



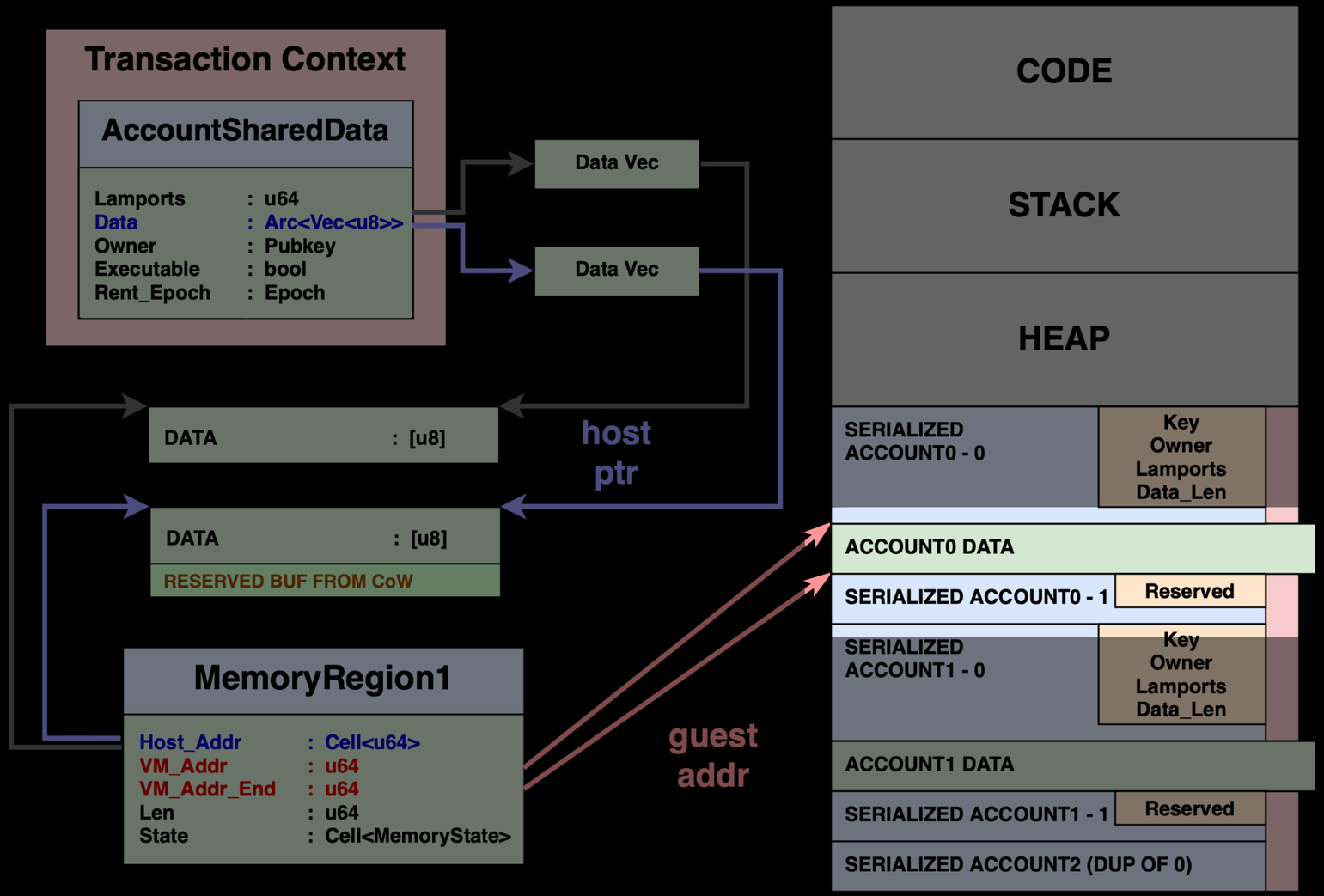
New Memory Regions

Updating Host_Addr



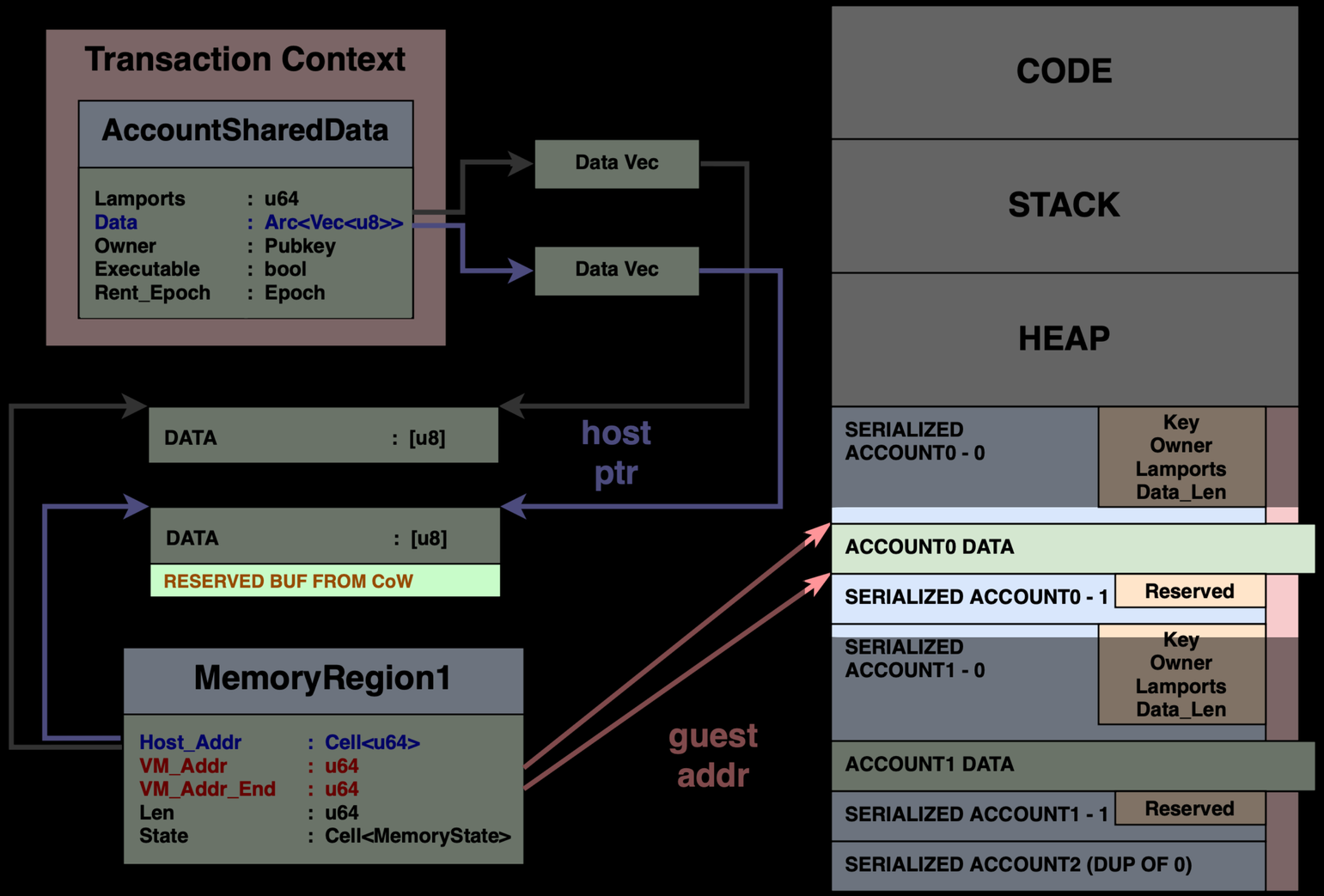
New Memory Regions

Reserved data buffer



New Memory Regions

Reserved data buffer



New Memory Regions

MemoryRegion.State

- State

MemoryRegion	
Host_Addr	: Cell<u64>
VM_Addr	: u64
VM_Addr_End	: u64
Len	: u64
State	: Cell<MemoryState>

New Memory Regions

MemoryRegion.State

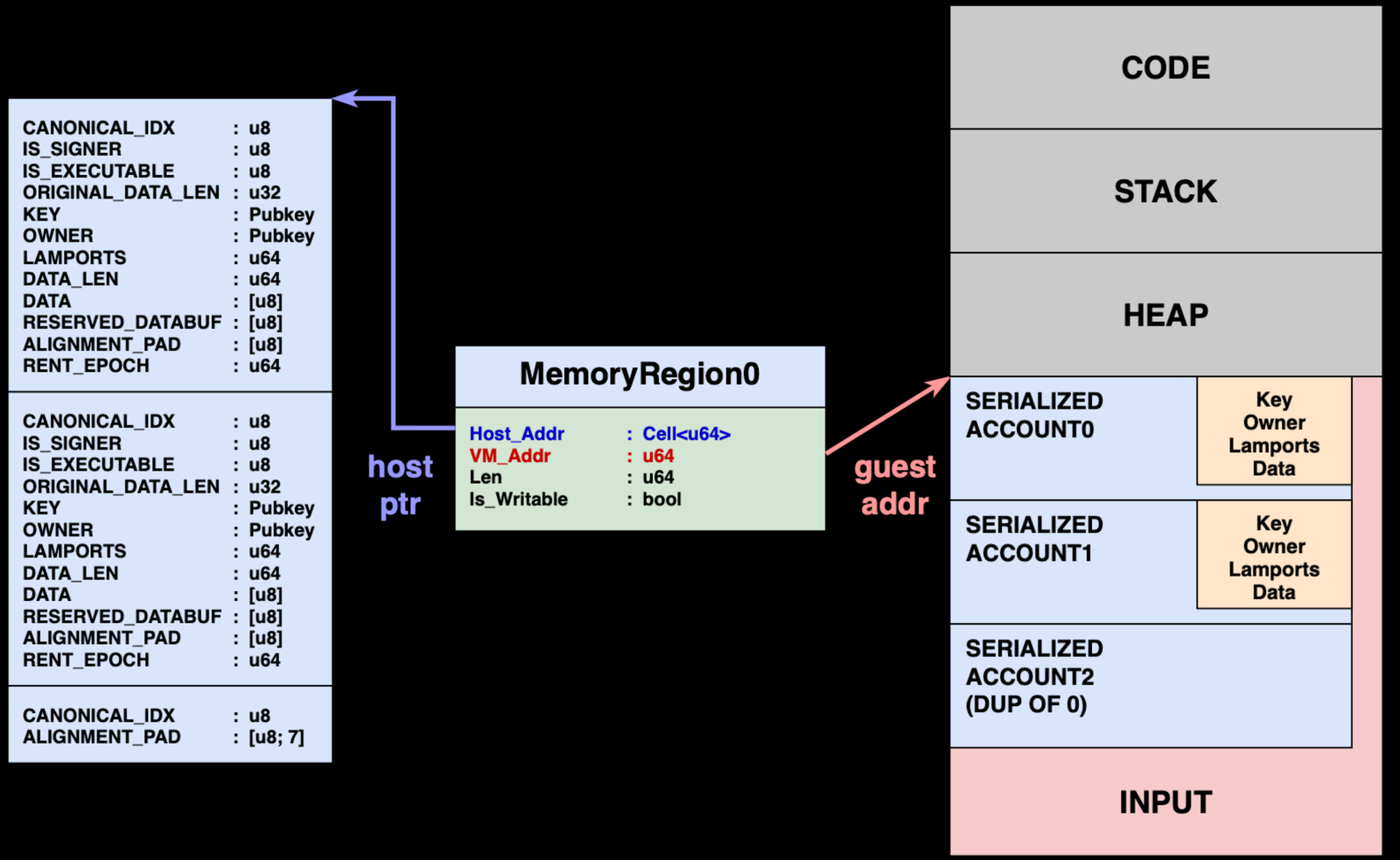
- State
  - Readable (read only)
  - Writable (read + write)
- Is_Writable
  - bool

MemoryRegion

Host_Addr	:	Cell<u64>
VM_Addr	:	u64
VM_Addr_End	:	u64
Len	:	u64
State	:	Cell<MemoryState>

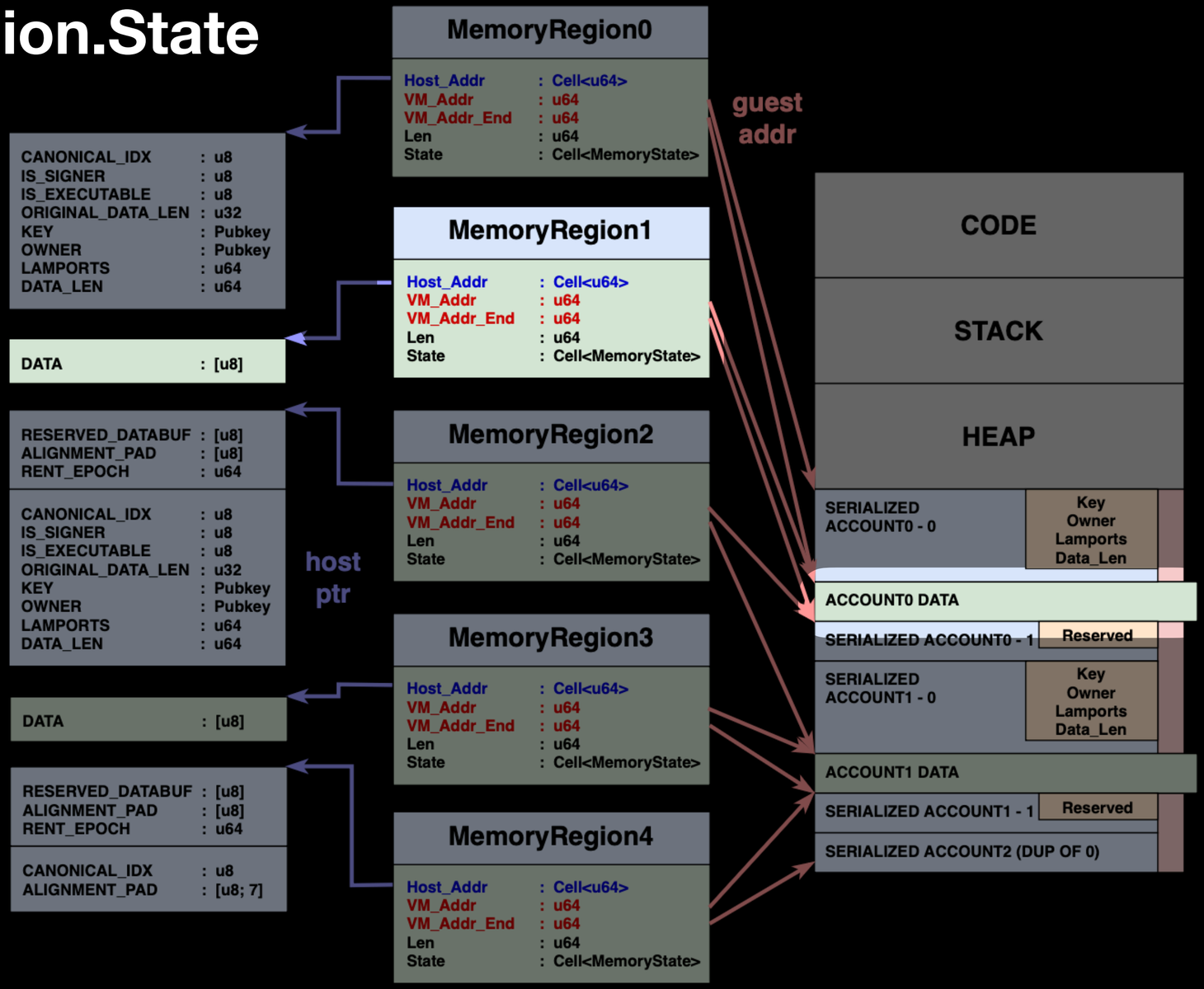
New Memory Regions

MemoryRegion.State



New Memory Regions

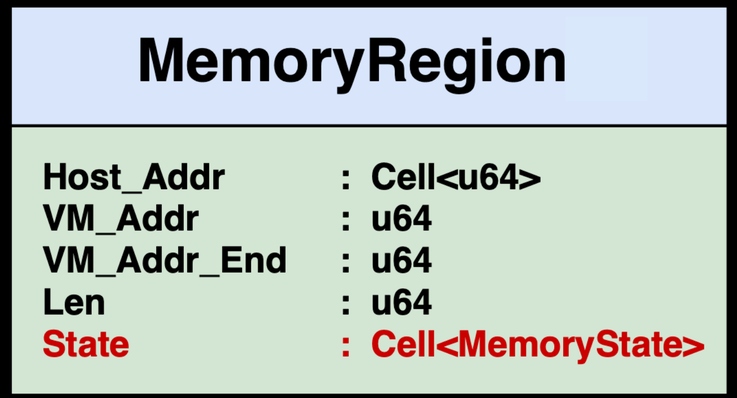
MemoryRegion.State



New Memory Regions

MemoryRegion.State

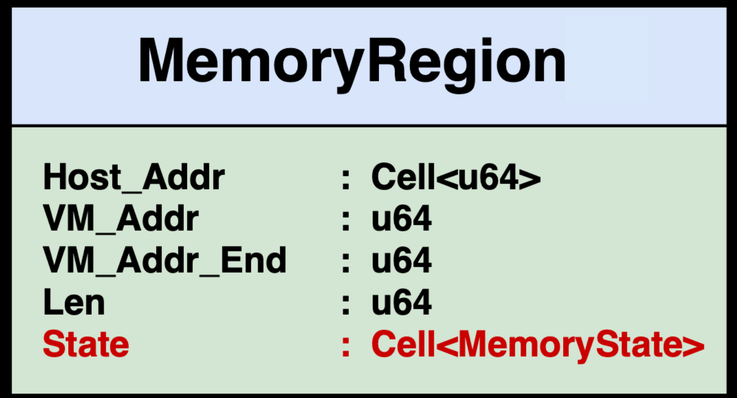
- State
  - Readable (read only)
  - Writable (read + write)



New Memory Regions

MemoryRegion.State

- State
  - Readable (read only)
  - Writable (read + write)
  - CoW (Handle Vector buffer reserve)



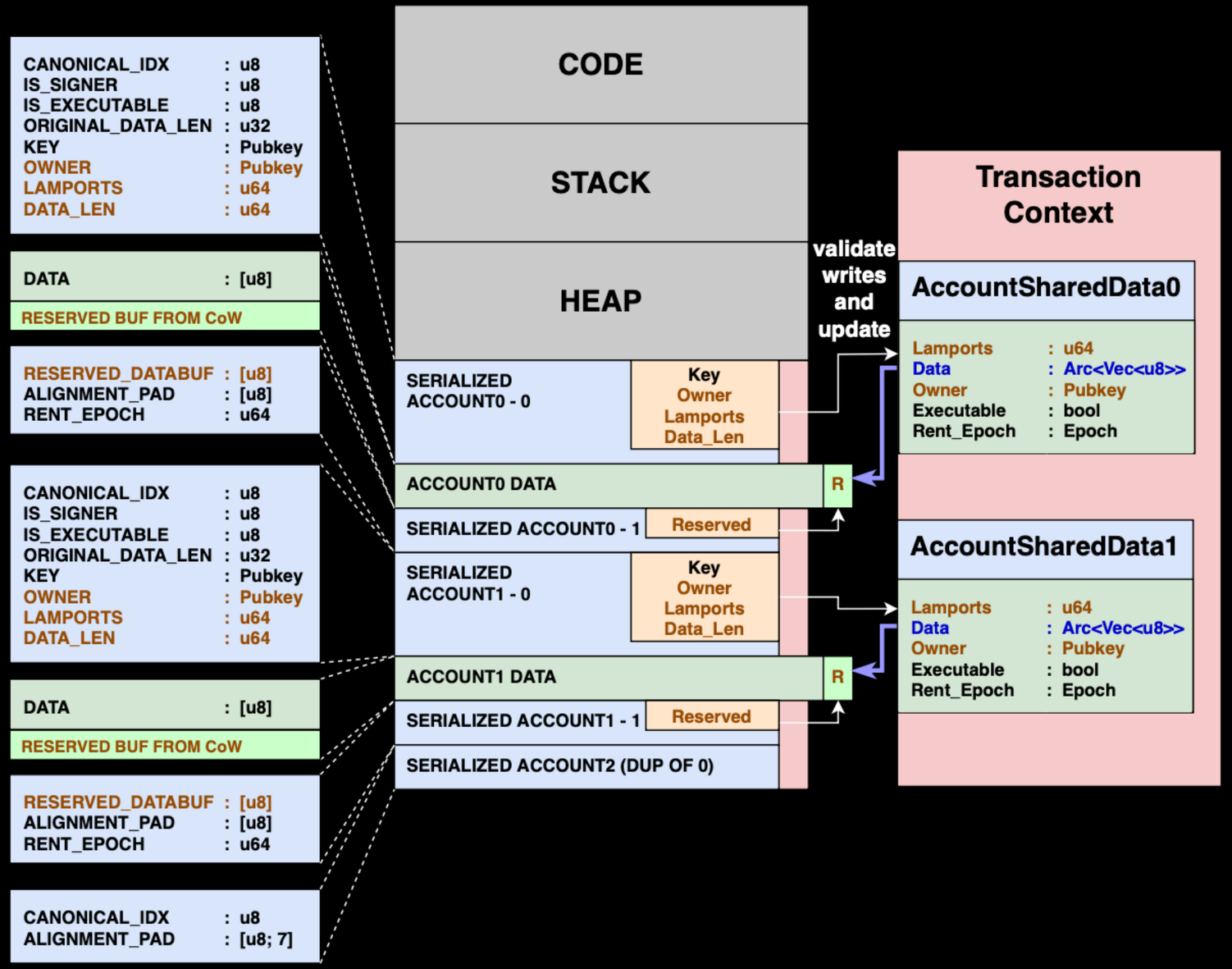
New Memory Regions

MemoryRegion.State

- Write permission check rules
  - Account fields are only writable if the Account is marked as writable in the instruction
  - Only the account Owner can modify Owner / Data fields
  - …

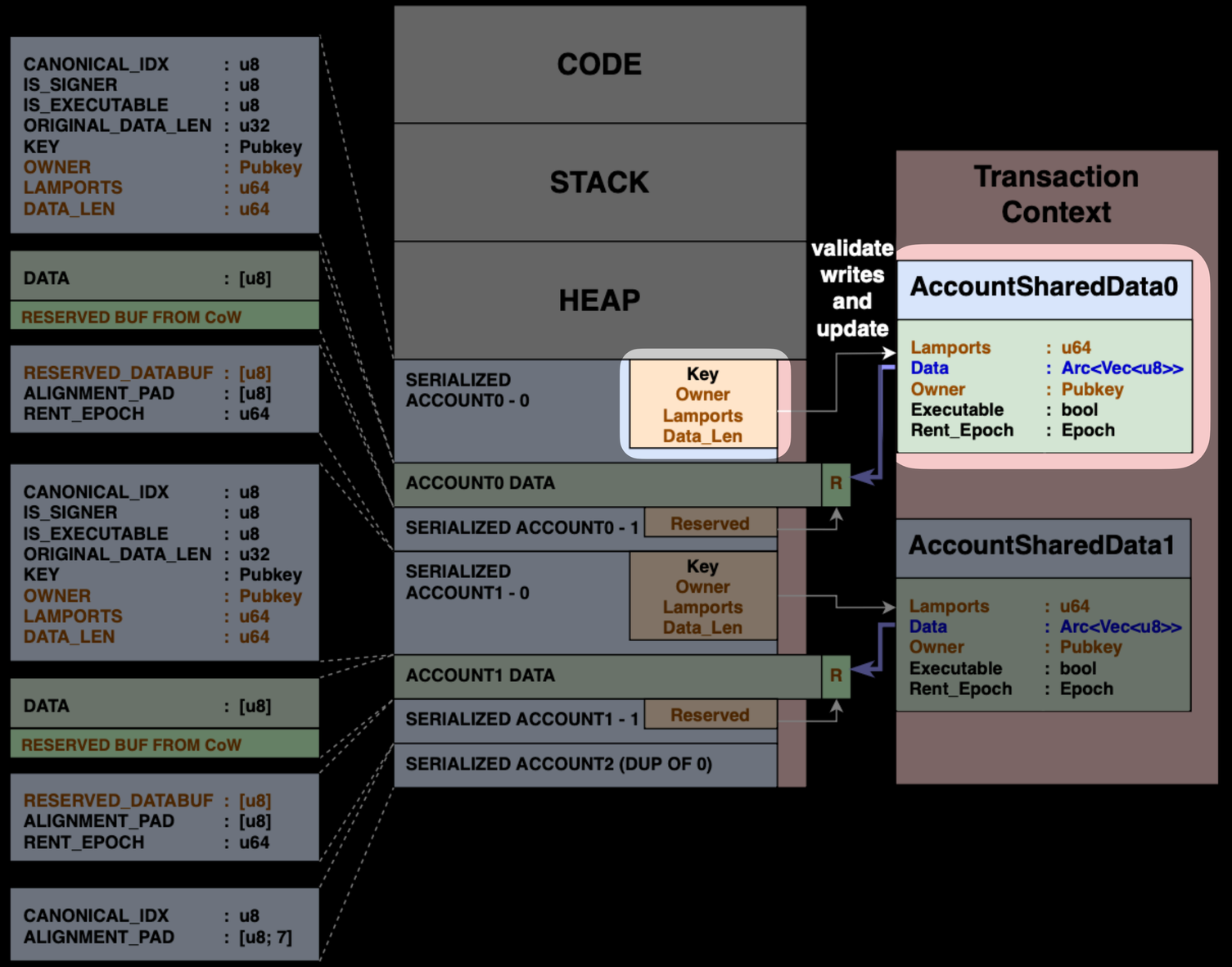
New Data Exposure

Commit



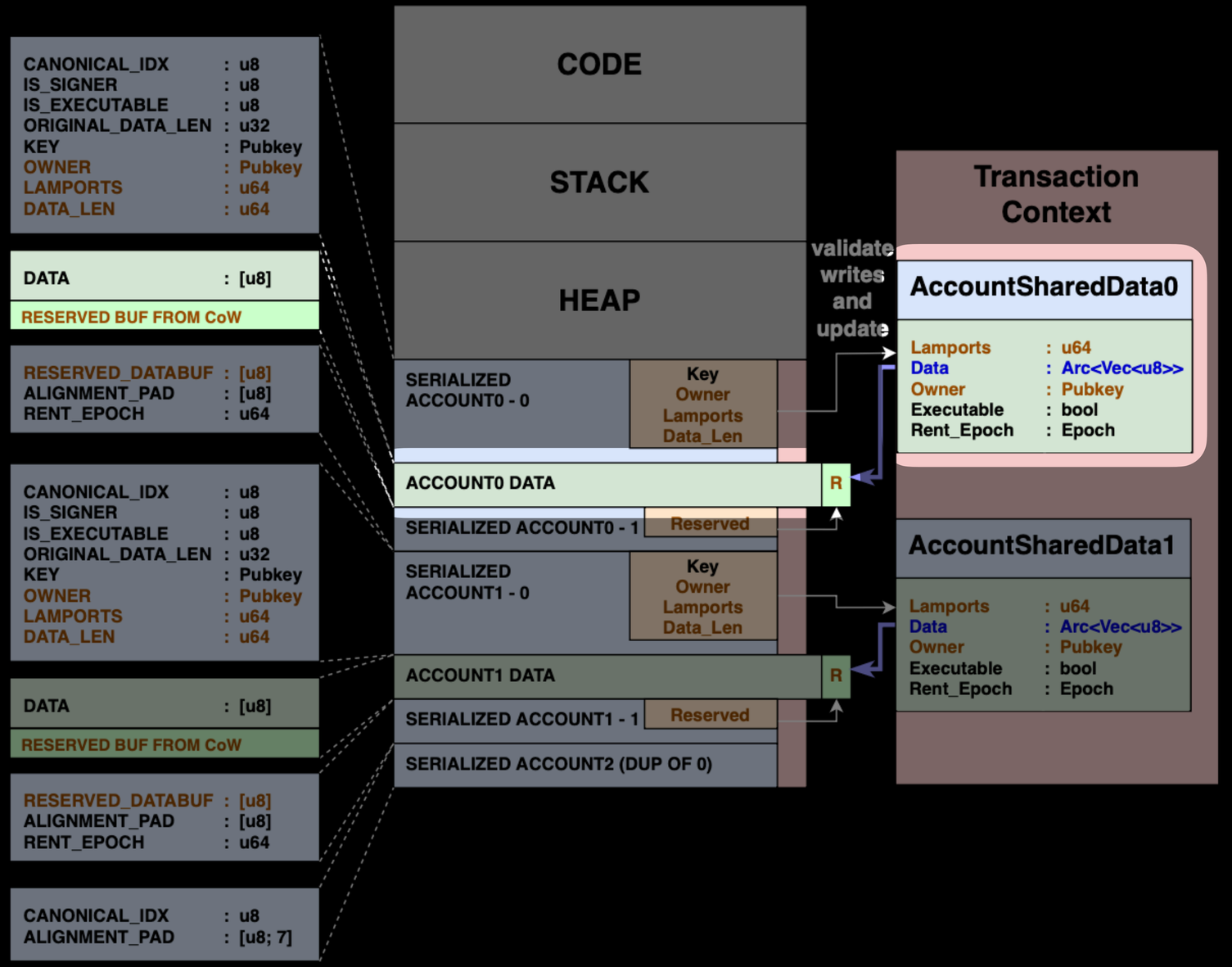
New Data Exposure

Commit



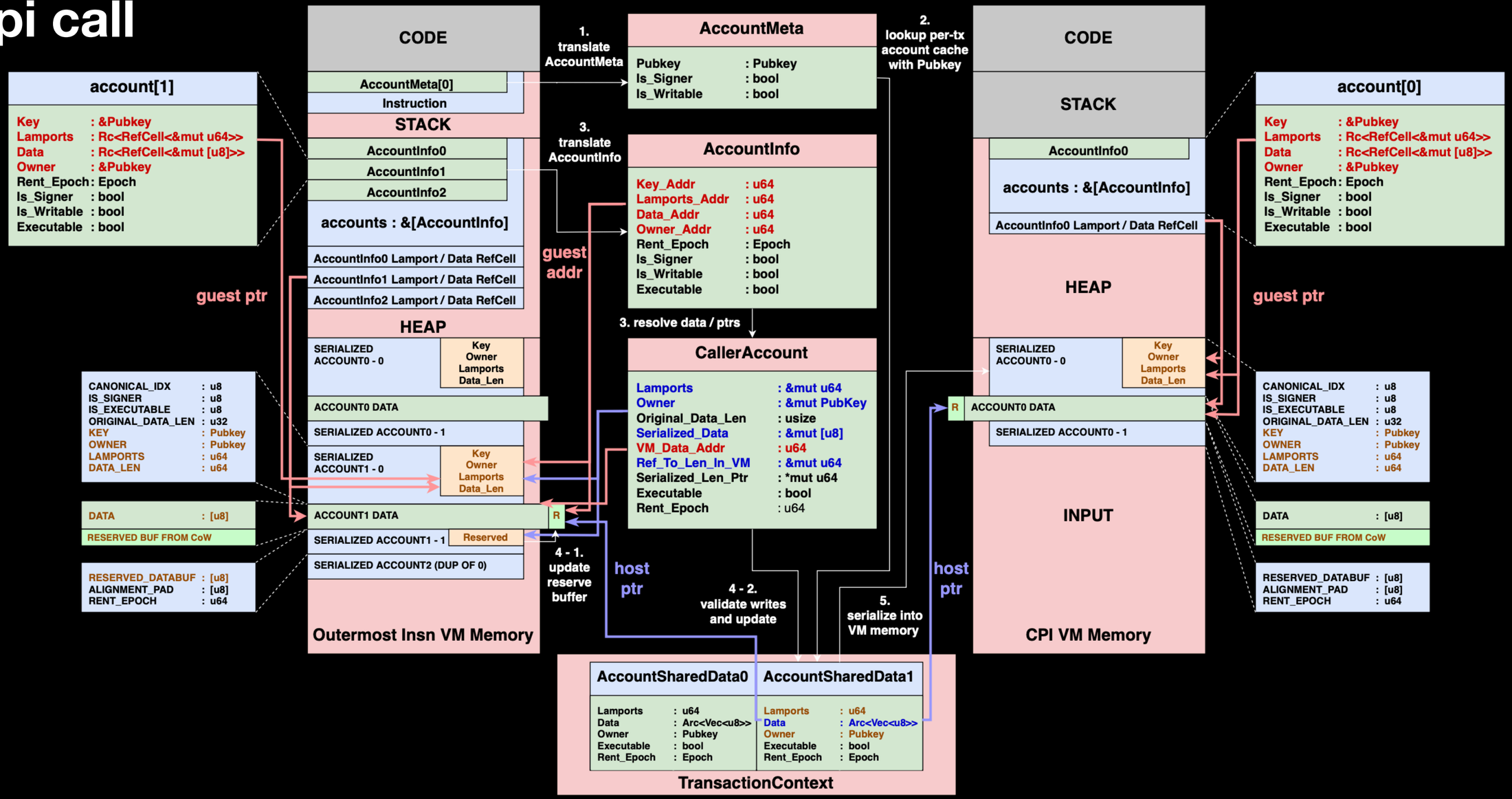
New Data Exposure

Commit



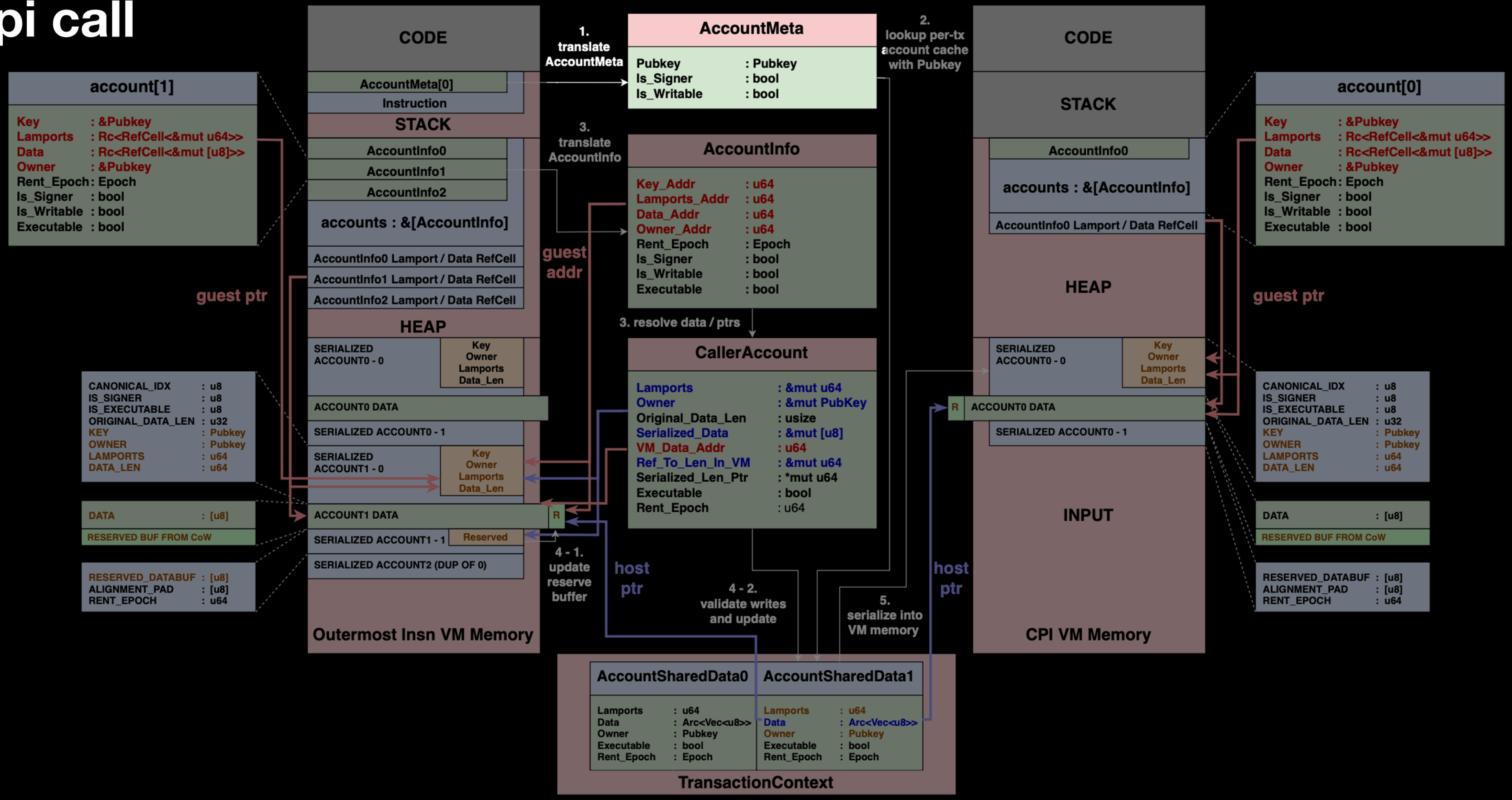
New Interoperability

cpi call



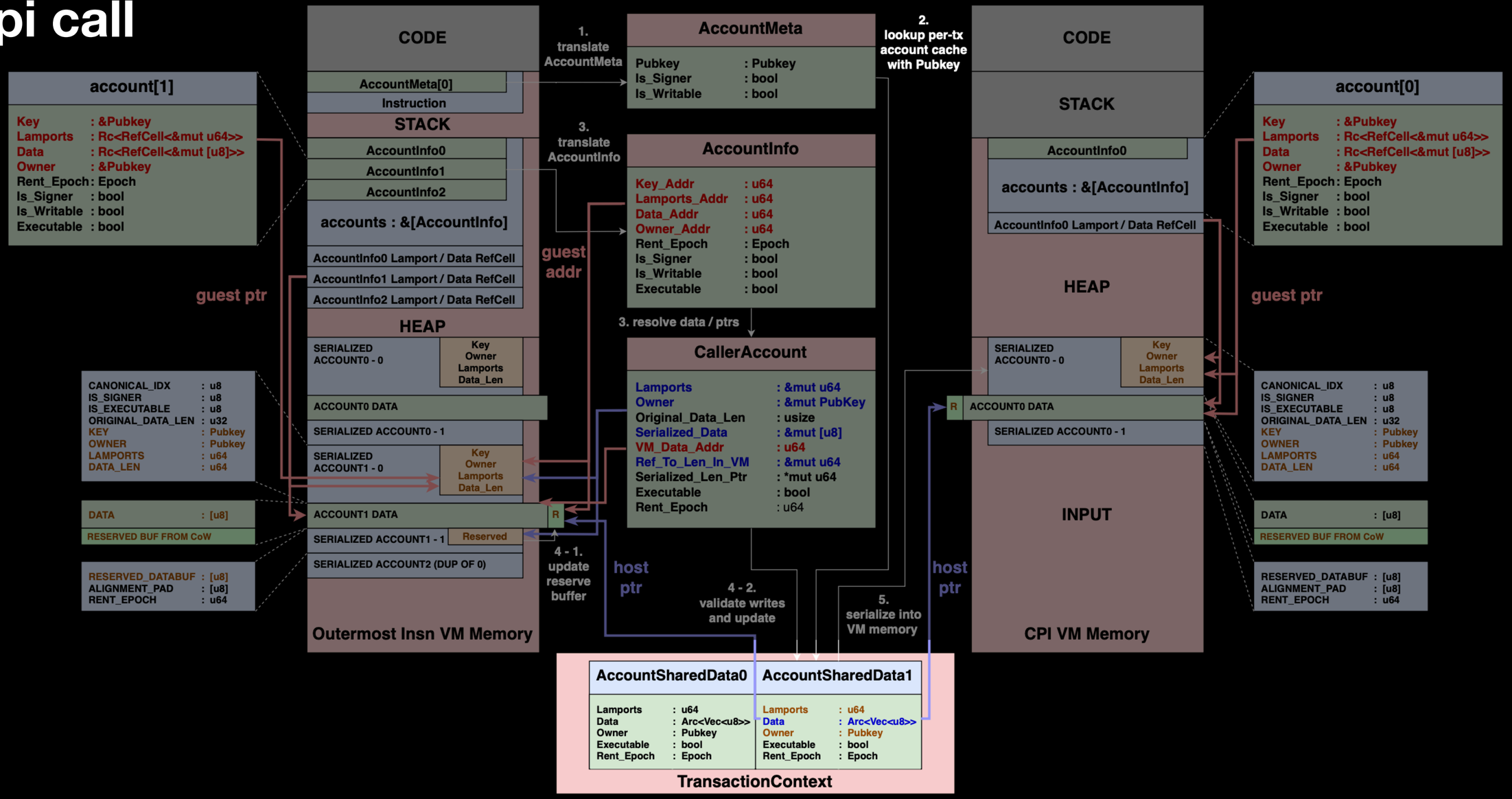
New Interoperability

cpi call



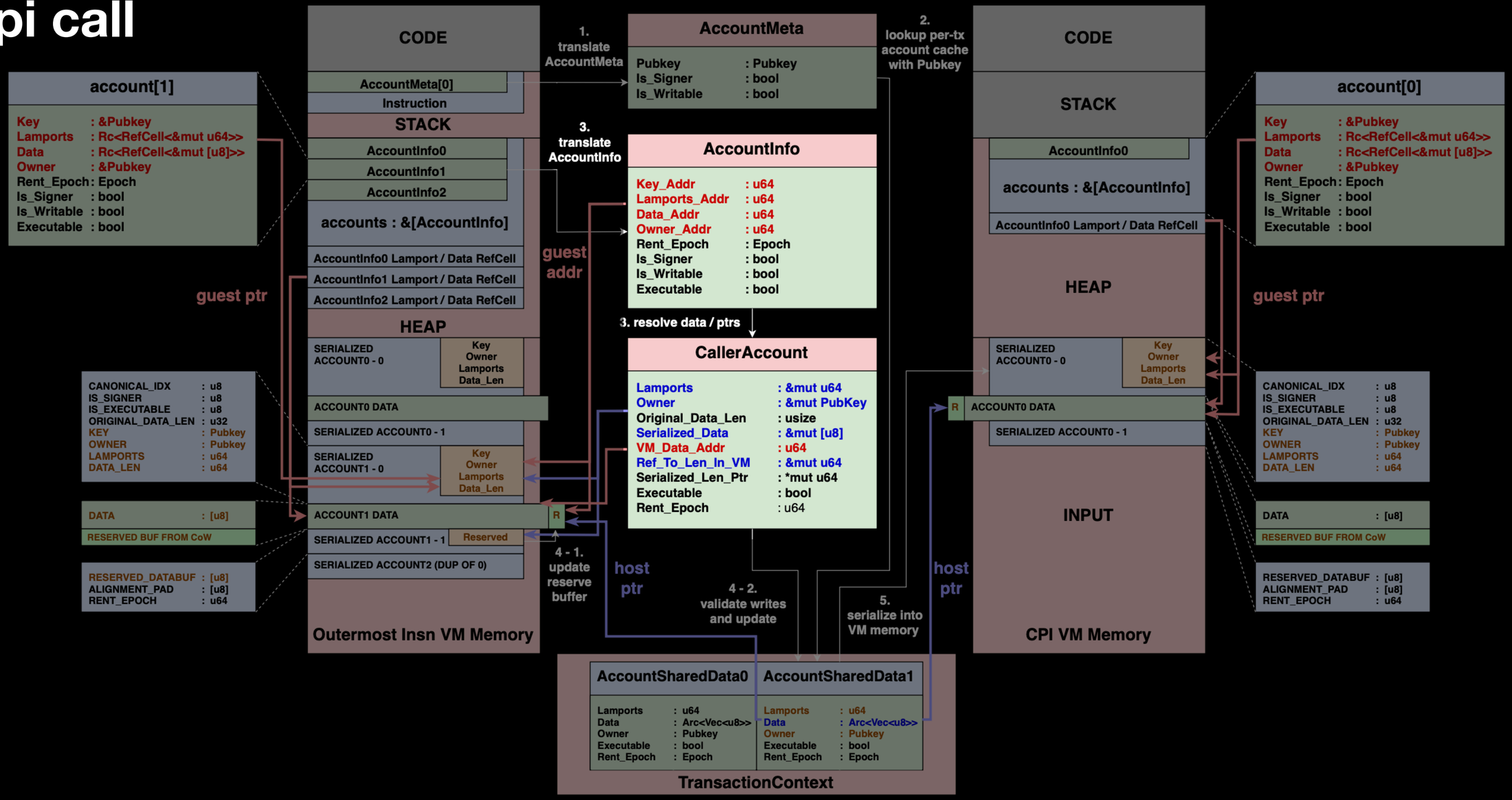
New Interoperability

cpi call



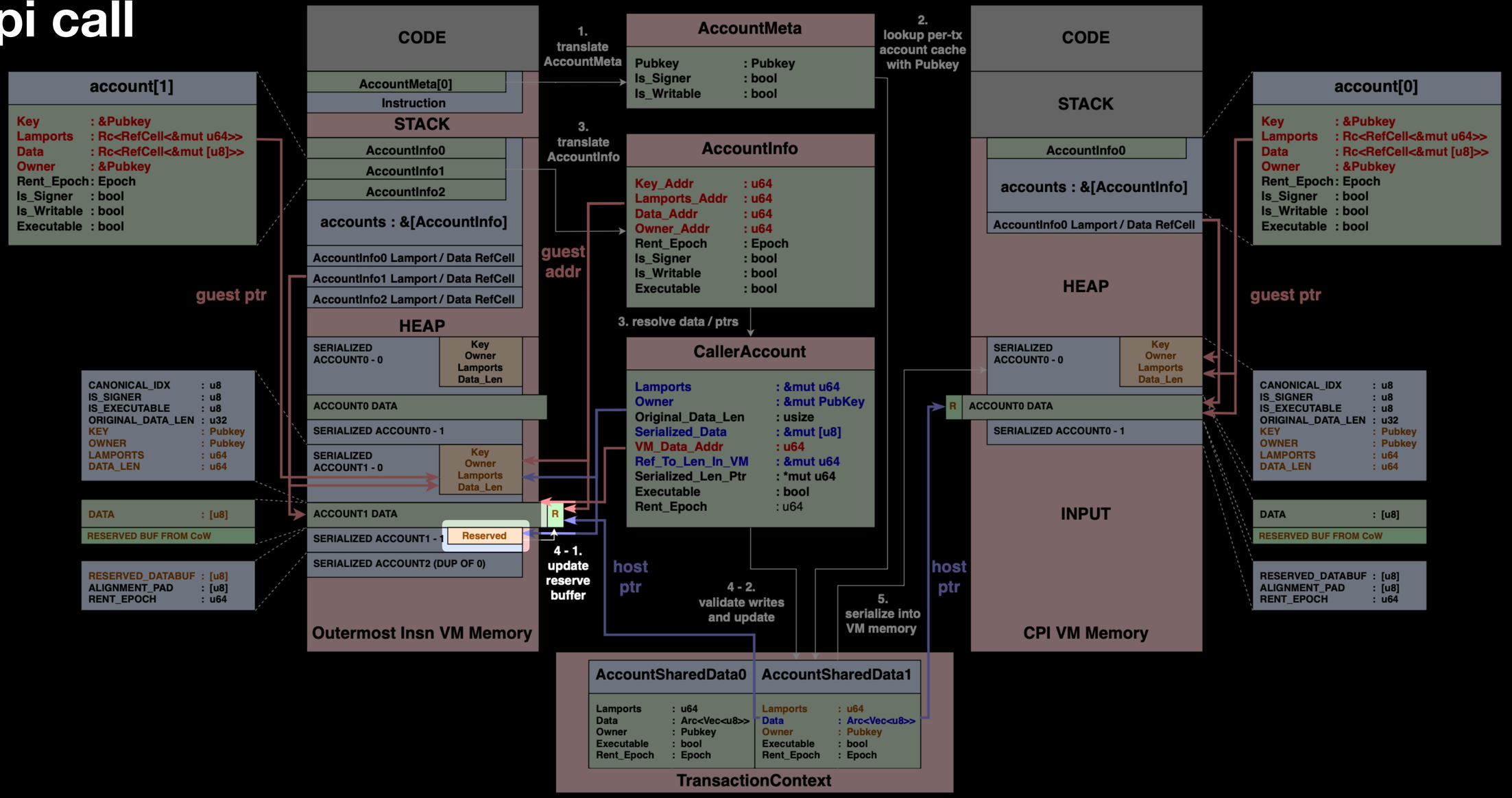
New Interoperability

cpi call



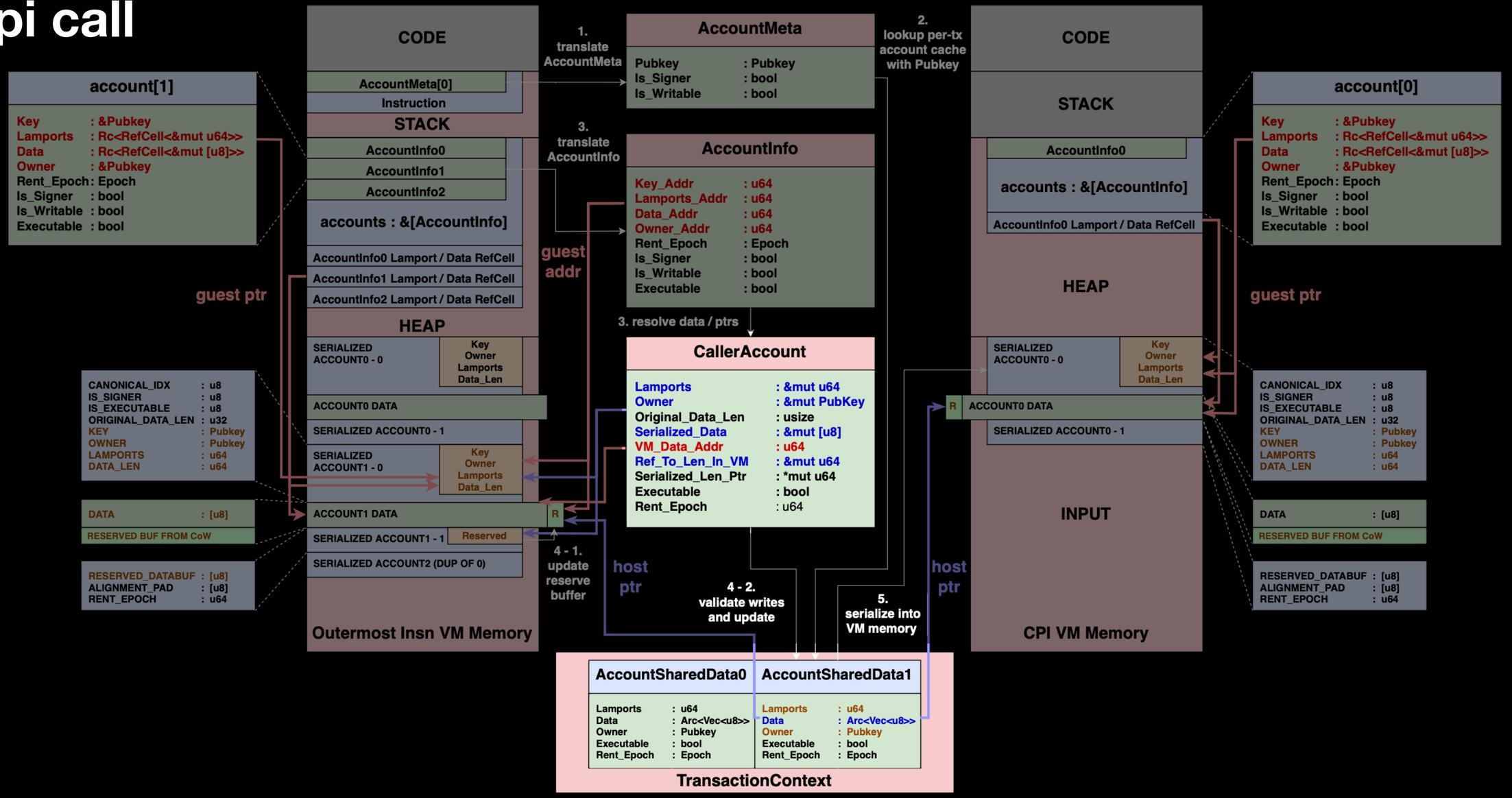
New Interoperability

cpi call



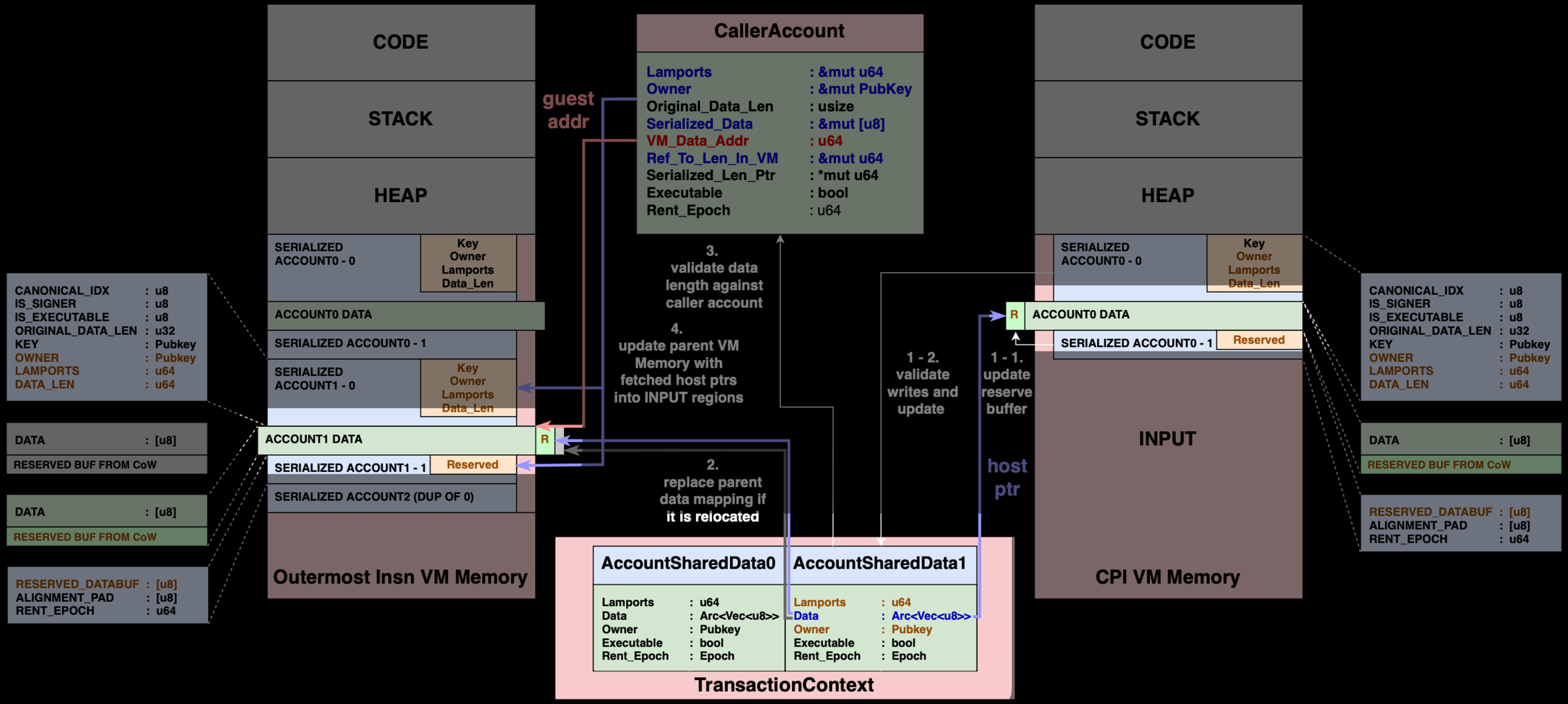
New Interoperability

cpi call



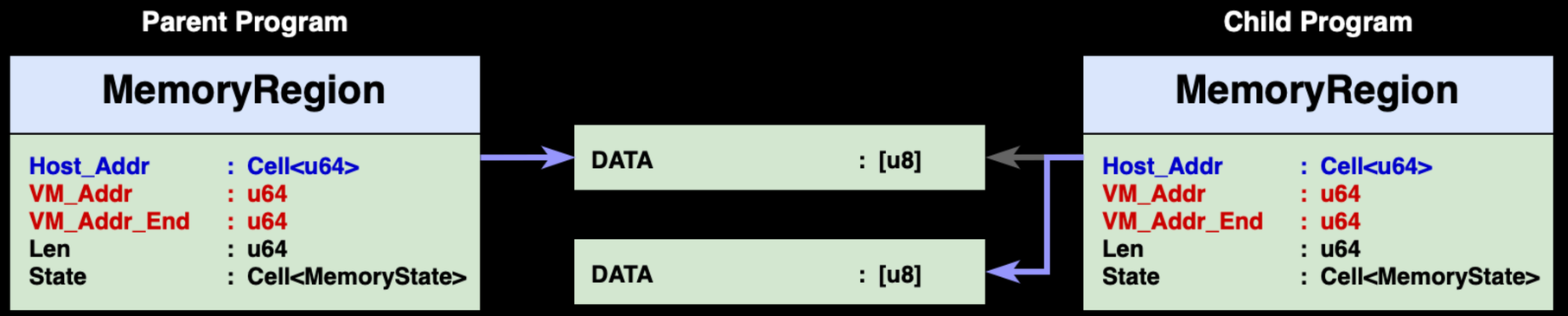
New Interoperability

cpi return



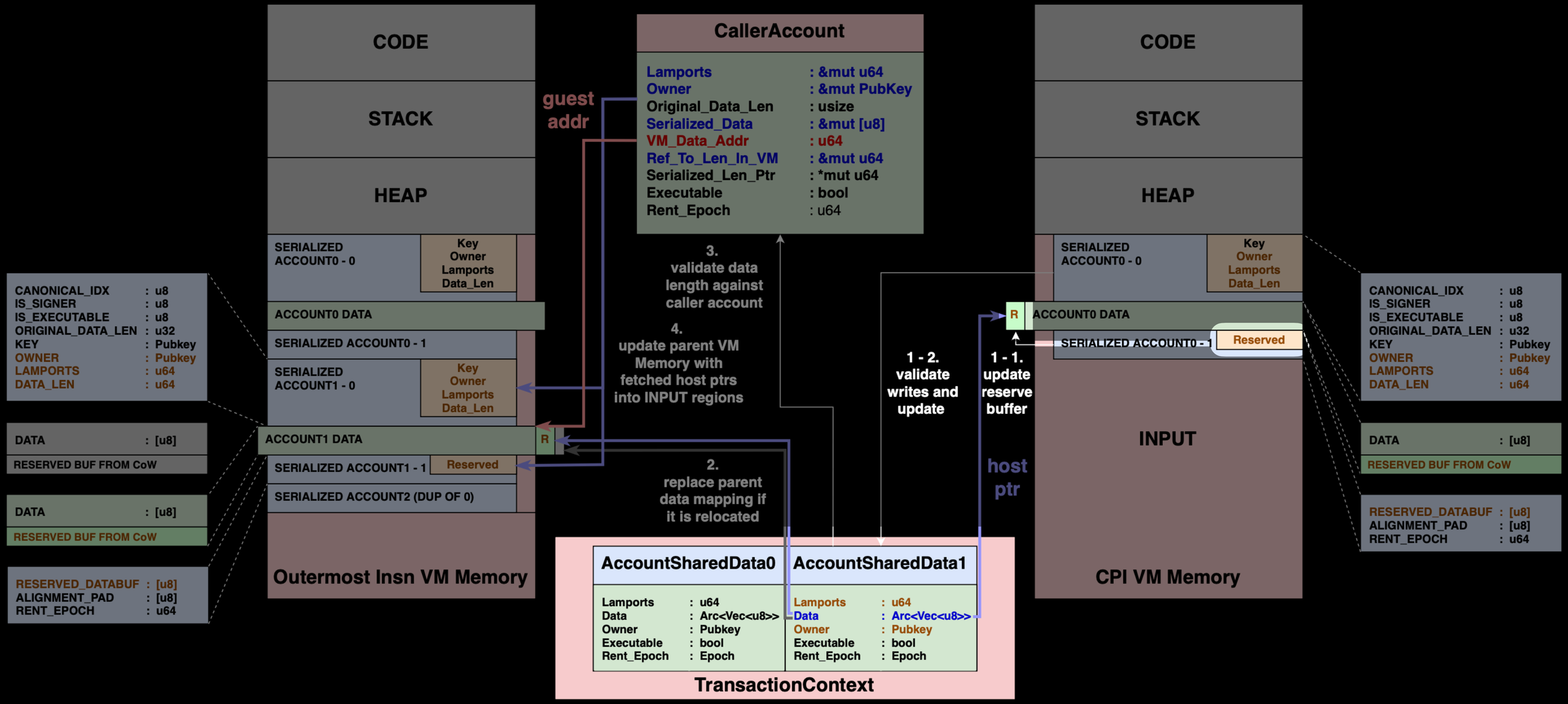
New Interoperability

Data relocation



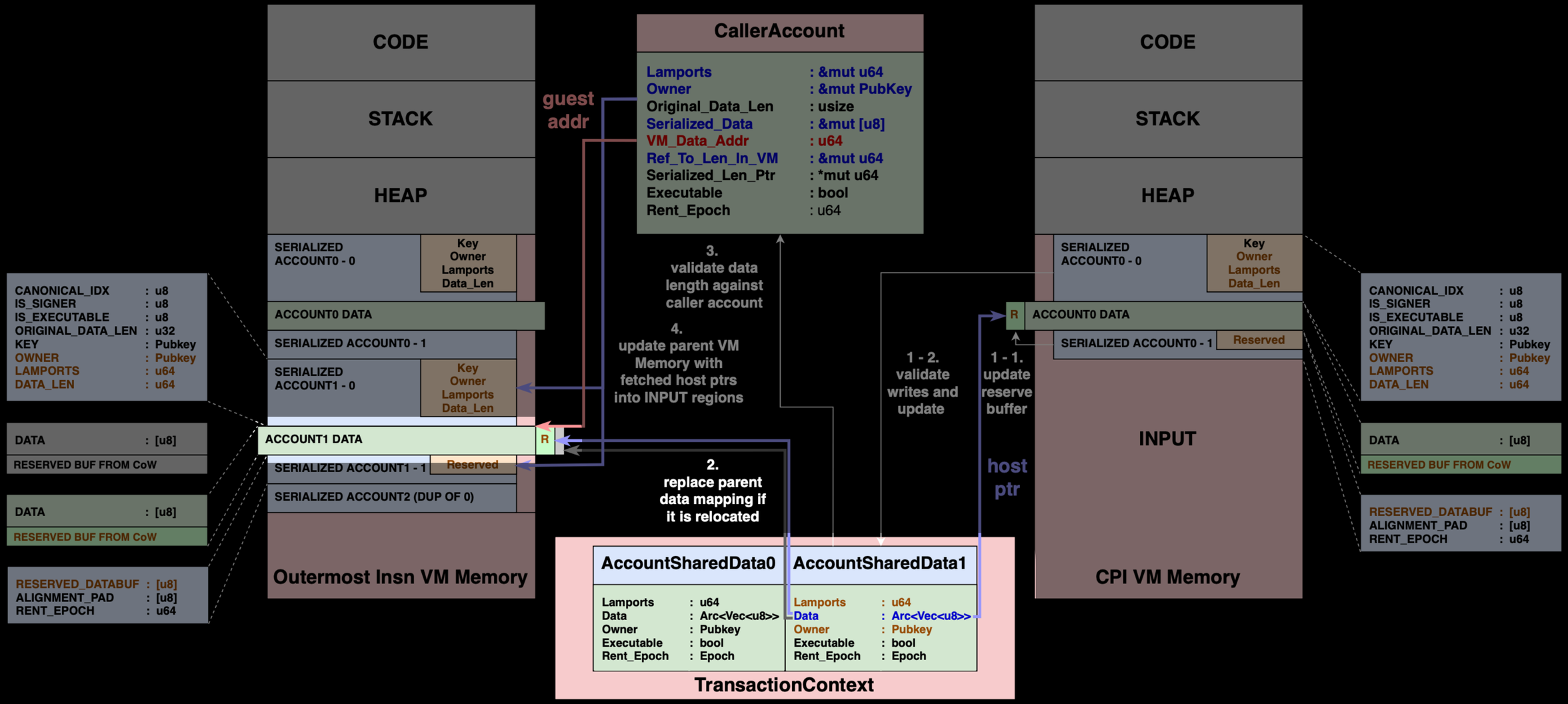
New Interoperability

cpi return



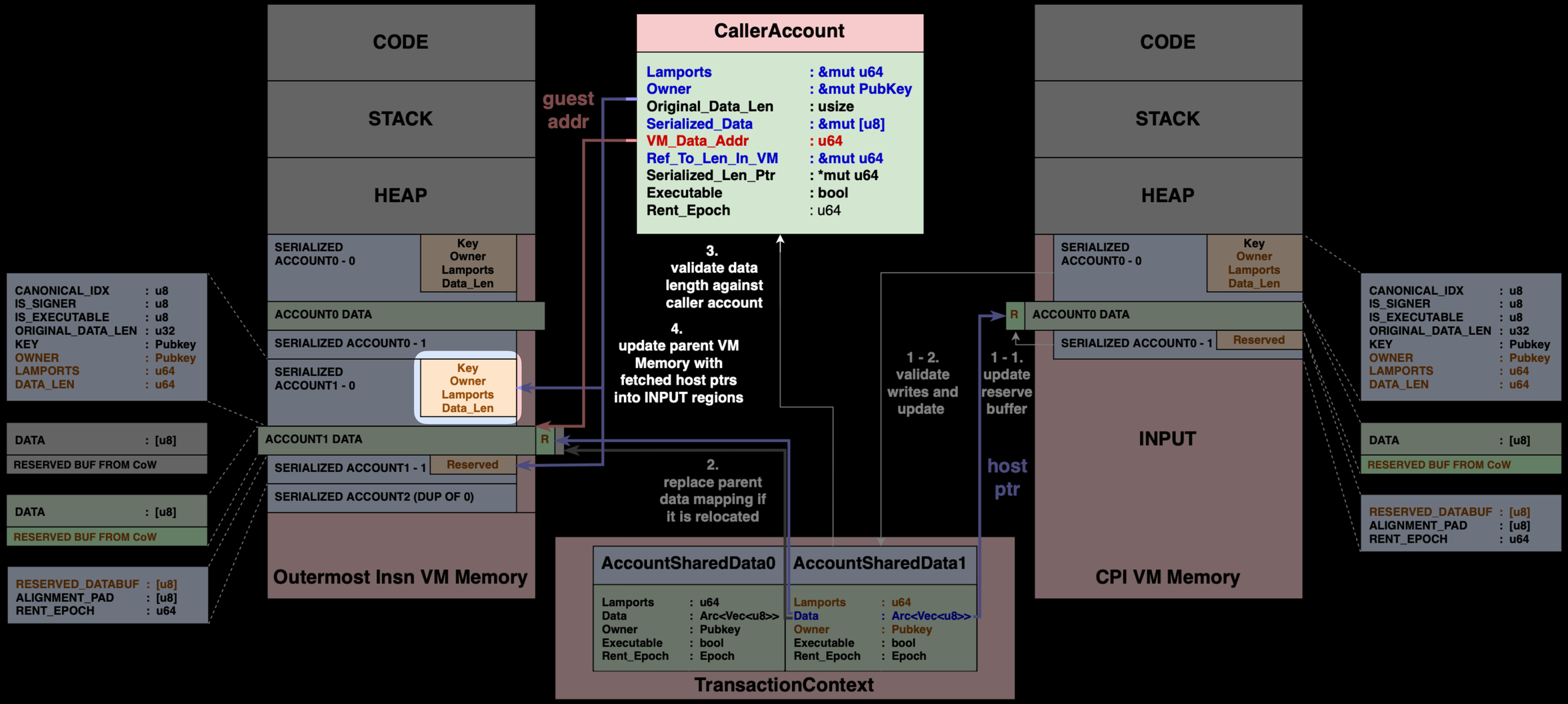
New Interoperability

cpi return



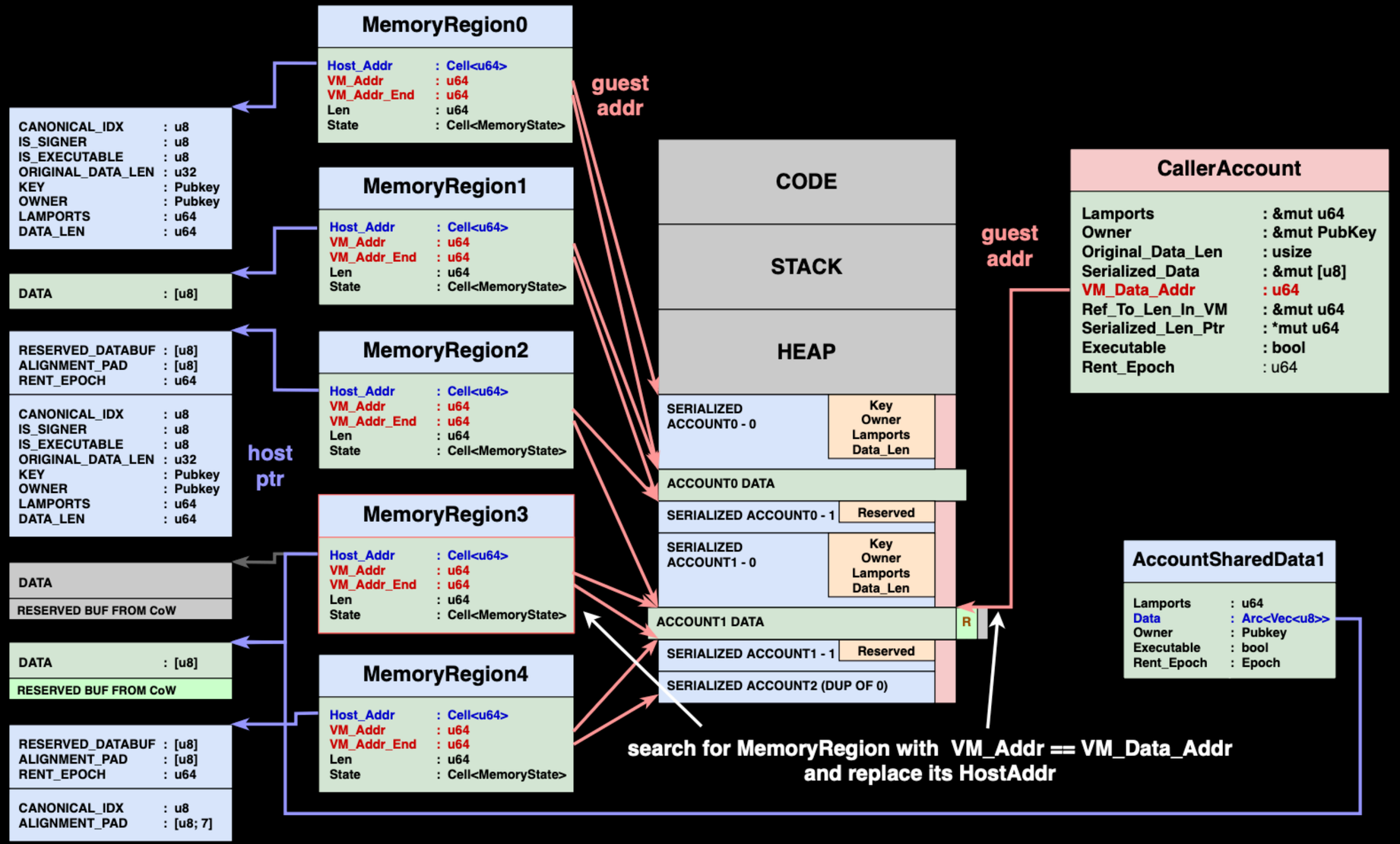
New Interoperability

cpi return



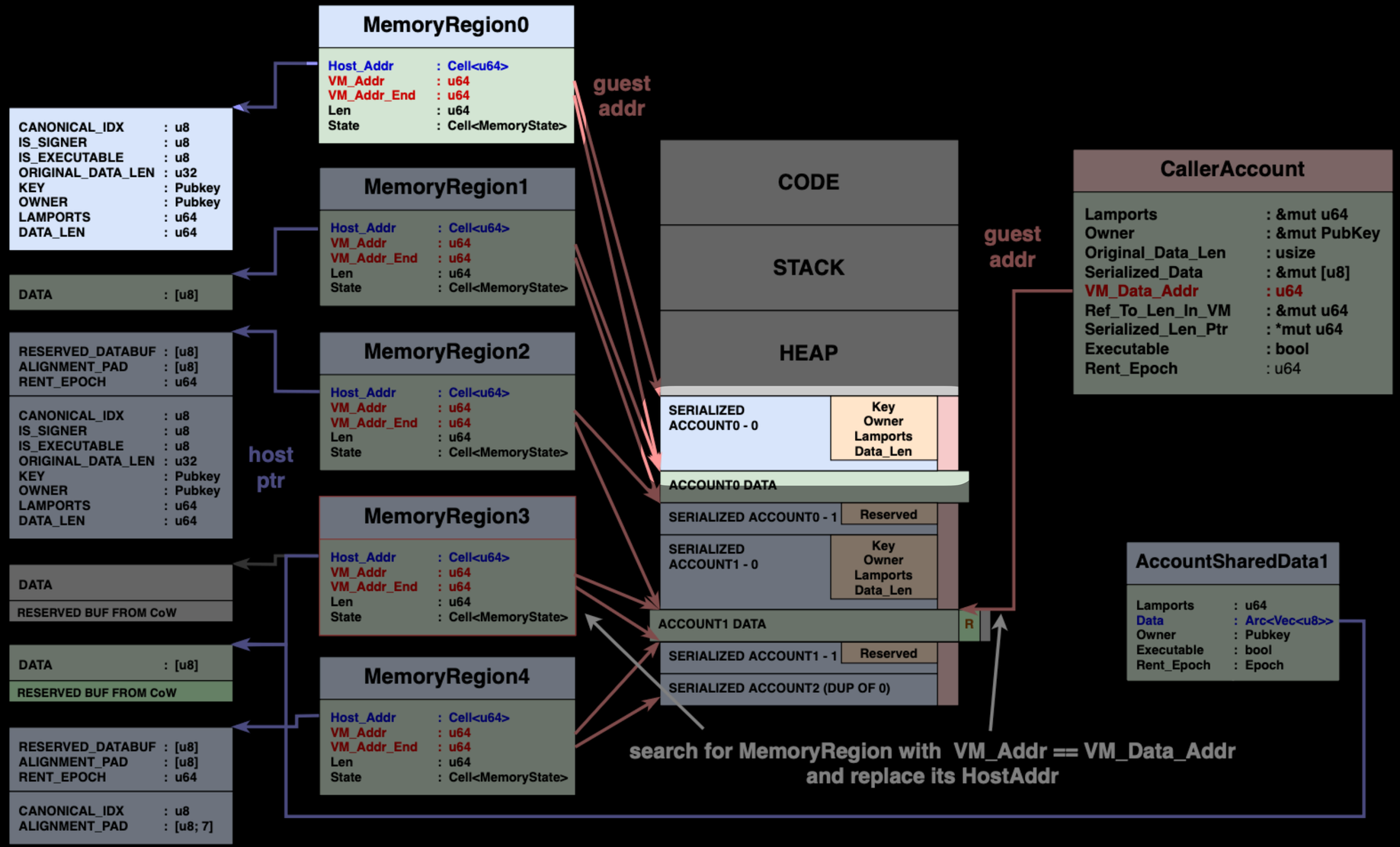
New Interoperability

Updating MemoryRegion.Host_Addr



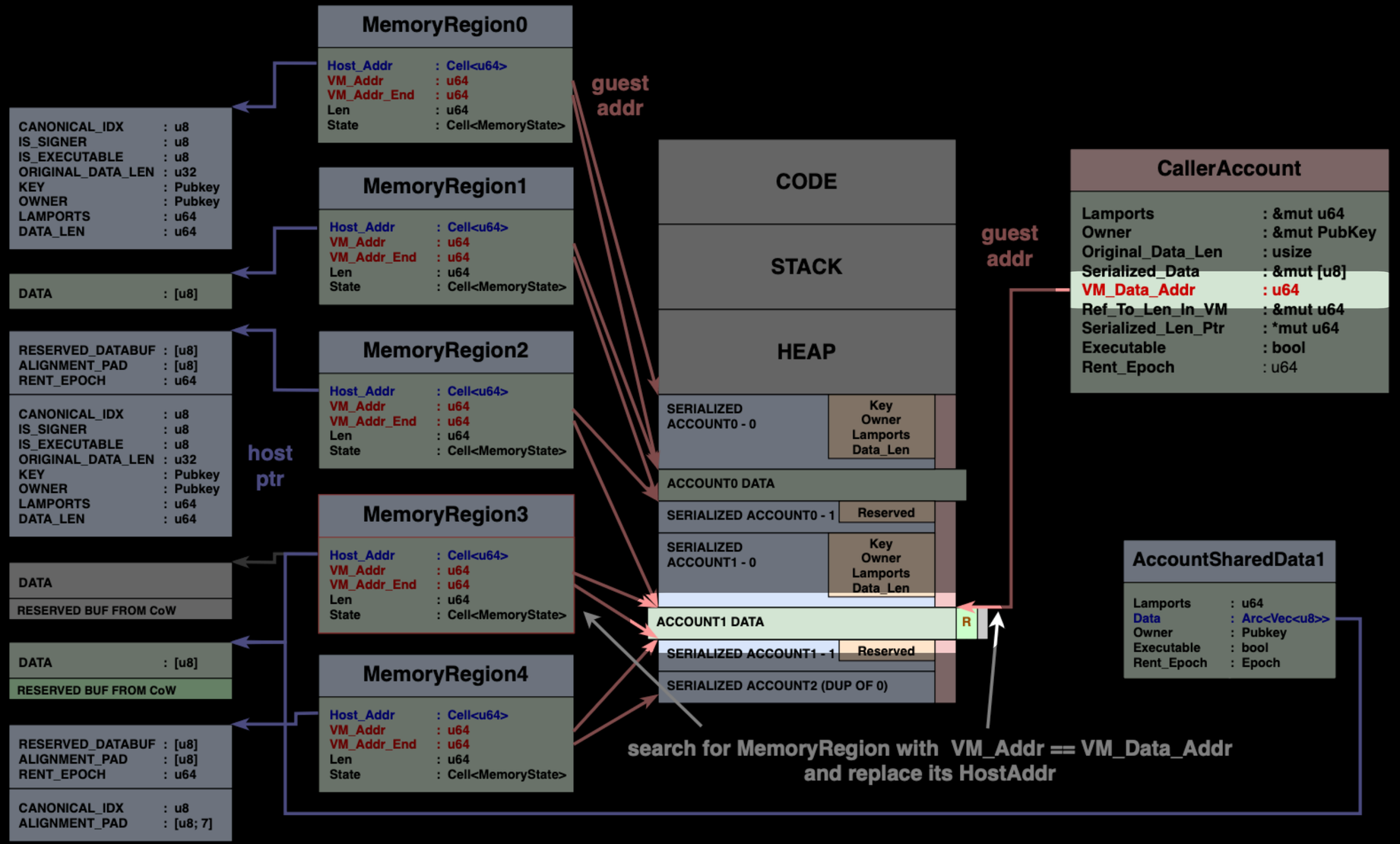
New Interoperability

Updating MemoryRegion.Host_Addr



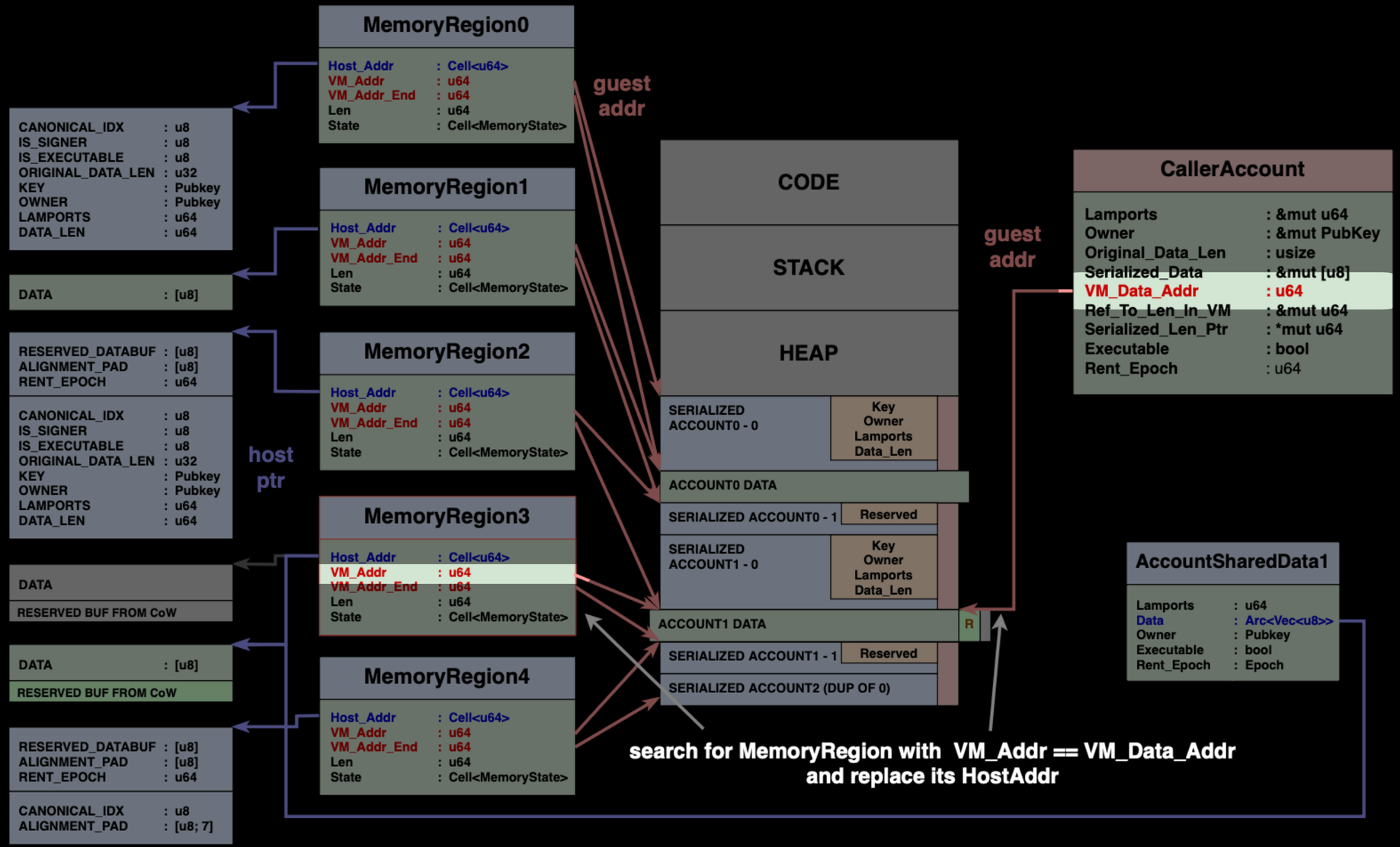
New Interoperability

Updating MemoryRegion.Host_Addr



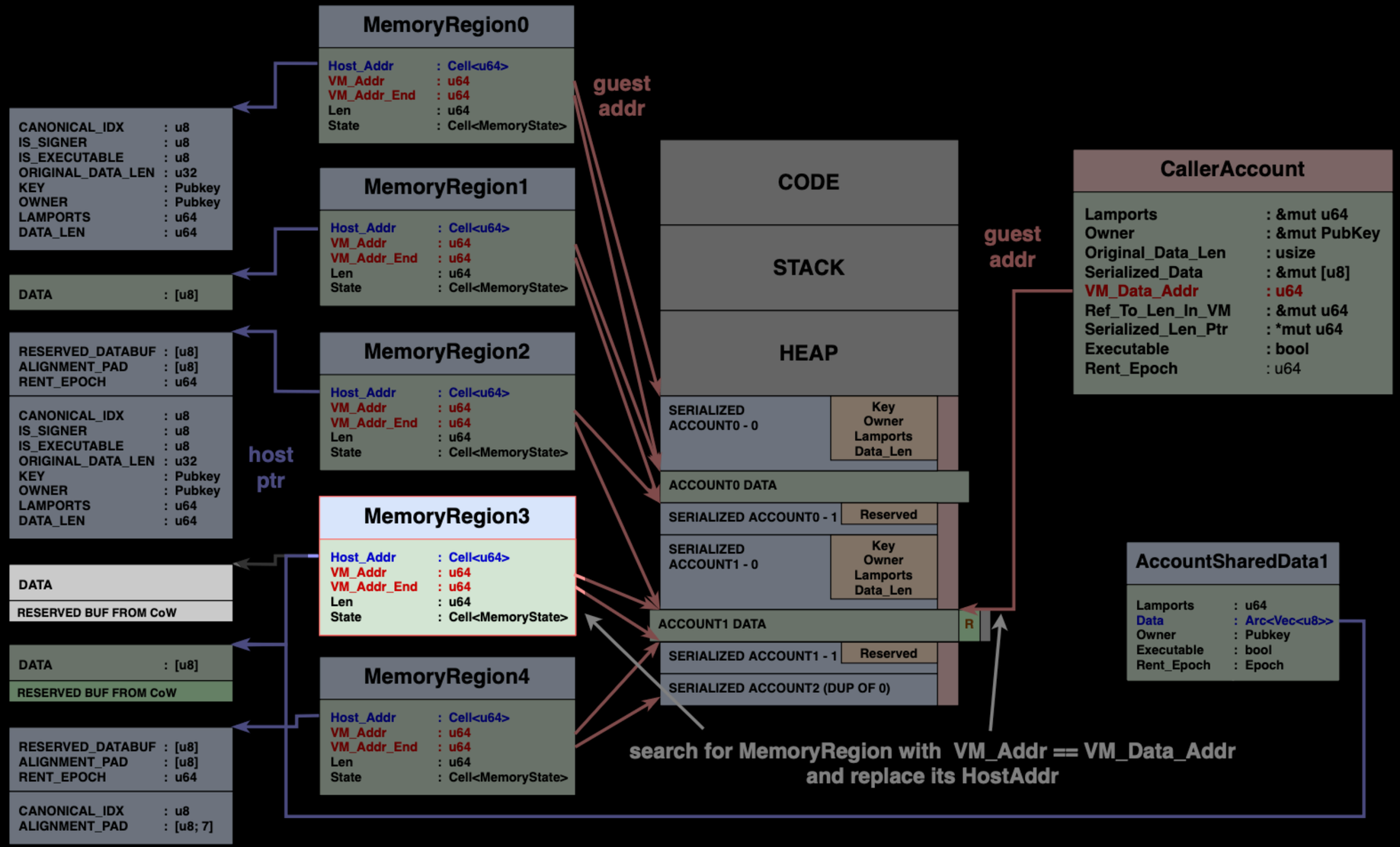
New Interoperability

Updating MemoryRegion.Host_Addr



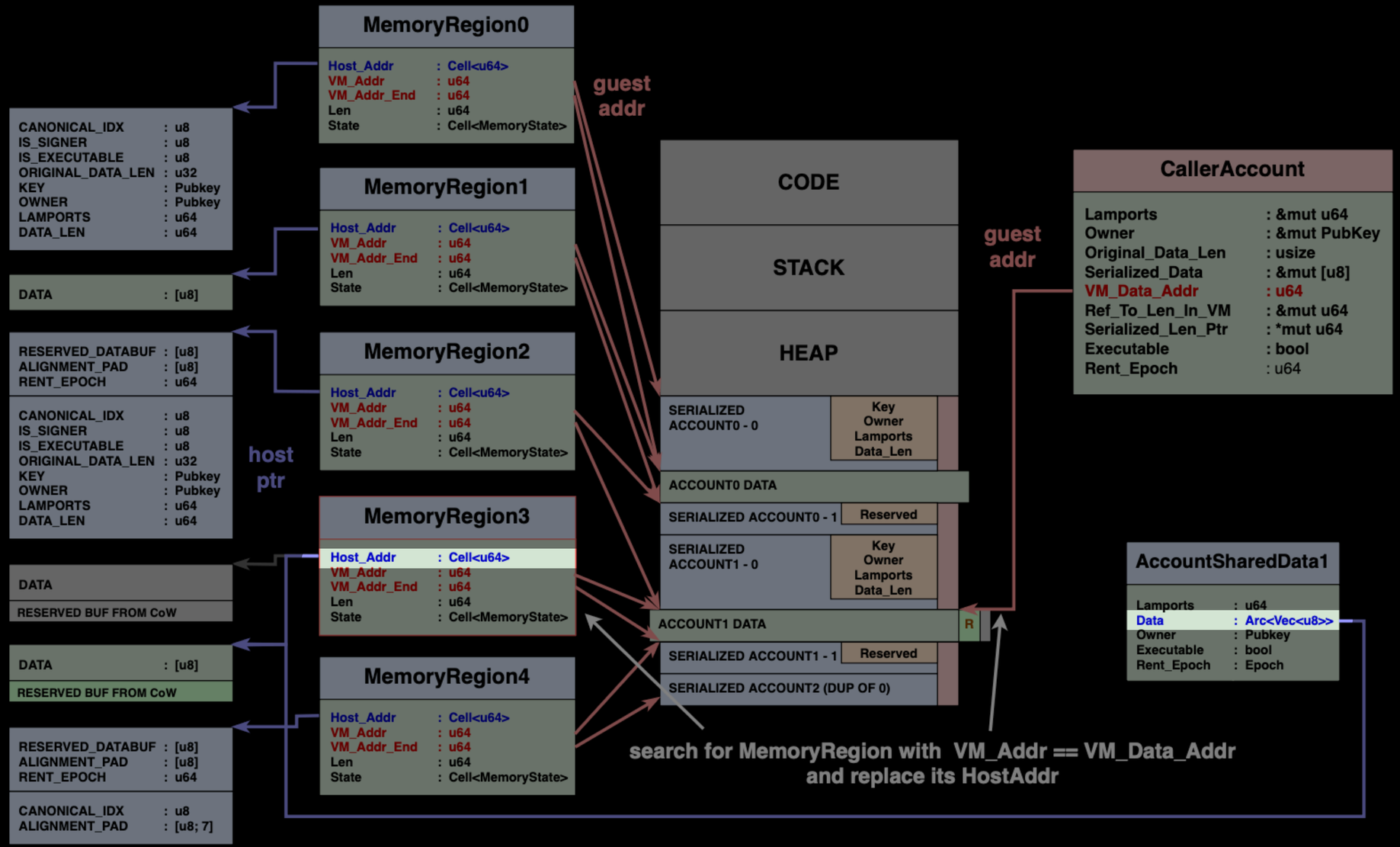
New Interoperability

Updating MemoryRegion.Host_Addr



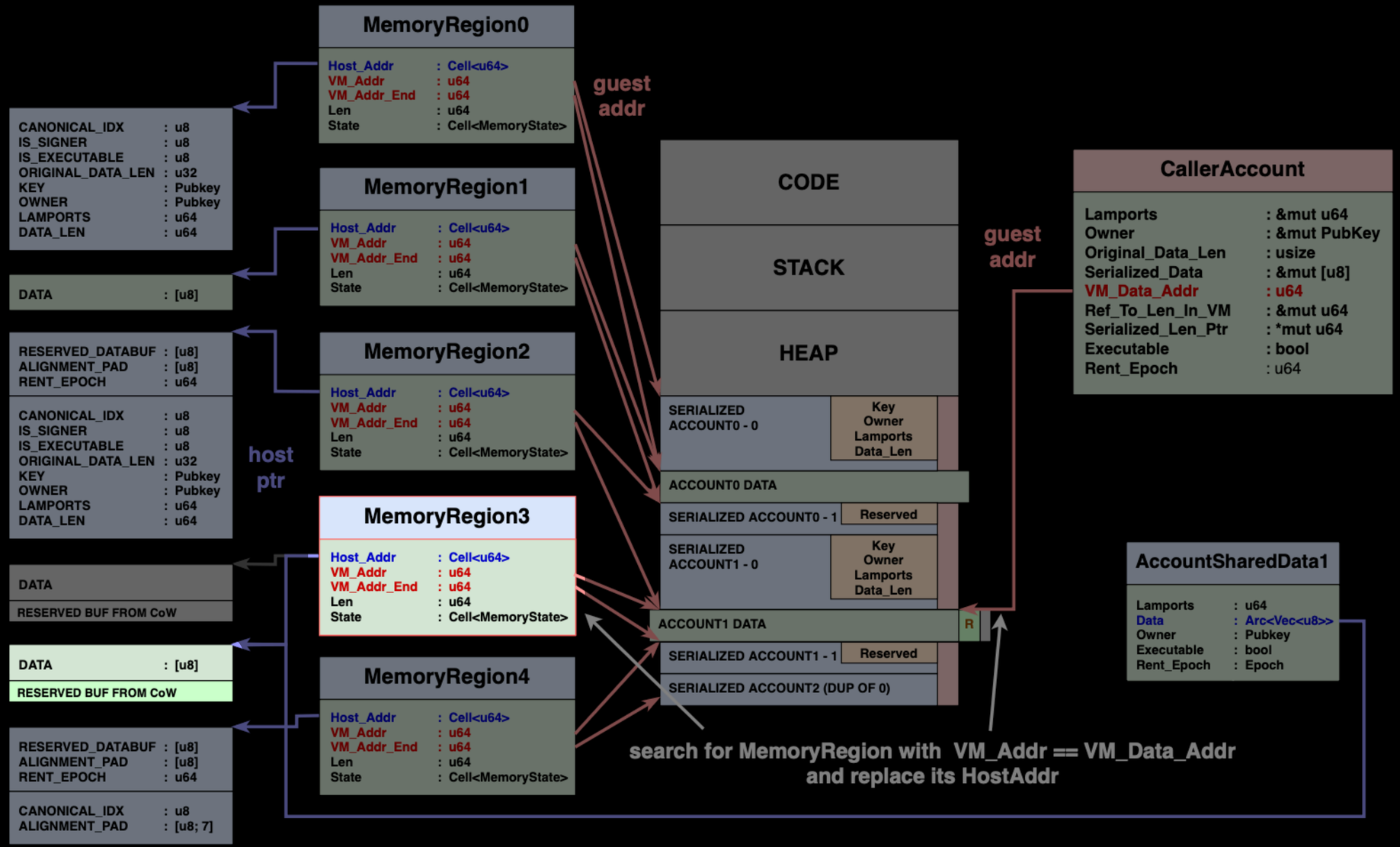
New Interoperability

Updating MemoryRegion.Host_Addr



New Interoperability

Updating MemoryRegion.Host_Addr



Bug

Bug

Missing check of guest address when updating MemoryRegion

```rust
fn update_caller_account(
    invoke_context: &InvokeContext,
    memory_mapping: &mut MemoryMapping,
    is_loader_deprecated: bool,
    caller_account: &mut CallerAccount,
    callee_account: &mut BorrowedAccount<'_>,
    direct_mapping: bool,
) -> Result<(), Error> {
    ...

    if direct_mapping && caller_account.original_data_len > 0 {
        ...
        let region = memory_mapping.region(AccessType::Load, caller_account.vm_data_addr)?;
        let callee_ptr = callee_account.get_data().as_ptr() as u64;
        if region.host_addr.get() != callee_ptr {
            region.host_addr.set(callee_ptr);
        }
    }
    ...
}
```

Bug

Missing check of guest address when updating MemoryRegion

```rust
fn update_caller_account(
    invoke_context: &InvokeContext,
    memory_mapping: &mut MemoryMapping,
    is_loader_deprecated: bool,
    caller_account: &mut CallerAccount,
    callee_account: &mut BorrowedAccount<'_>,
    direct_mapping: bool,
) -> Result<(), Error> {
    ...

    if direct_mapping && caller_account.original_data_len > 0 {
        ...
        let region = memory_mapping.region(AccessType::Load, caller_account.vm_data_addr)?;
        let callee_ptr = callee_account.get_data().as_ptr() as u64;
        if region.host_addr.get() != callee_ptr {
            region.host_addr.set(callee_ptr);
        }
    }
    ...
}
```

Bug

Missing check of guest address when updating MemoryRegion

```rust
fn update_caller_account(
    invoke_context: &InvokeContext,
    memory_mapping: &mut MemoryMapping,
    is_loader_deprecated: bool,
    caller_account: &mut CallerAccount,
    callee_account: &mut BorrowedAccount<'_>,
    direct_mapping: bool,
) -> Result<(), Error> {
    ...

    if direct_mapping && caller_account.original_data_len > 0 {
        ...
        let region = memory_mapping.region(AccessType::Load, caller_account.vm_data_addr)?;
        let callee_ptr = callee_account.get_data().as_ptr() as u64;
        if region.host_addr.get() != callee_ptr {
            region.host_addr.set(callee_ptr);
        }
    }
    ...
}
```

Bug

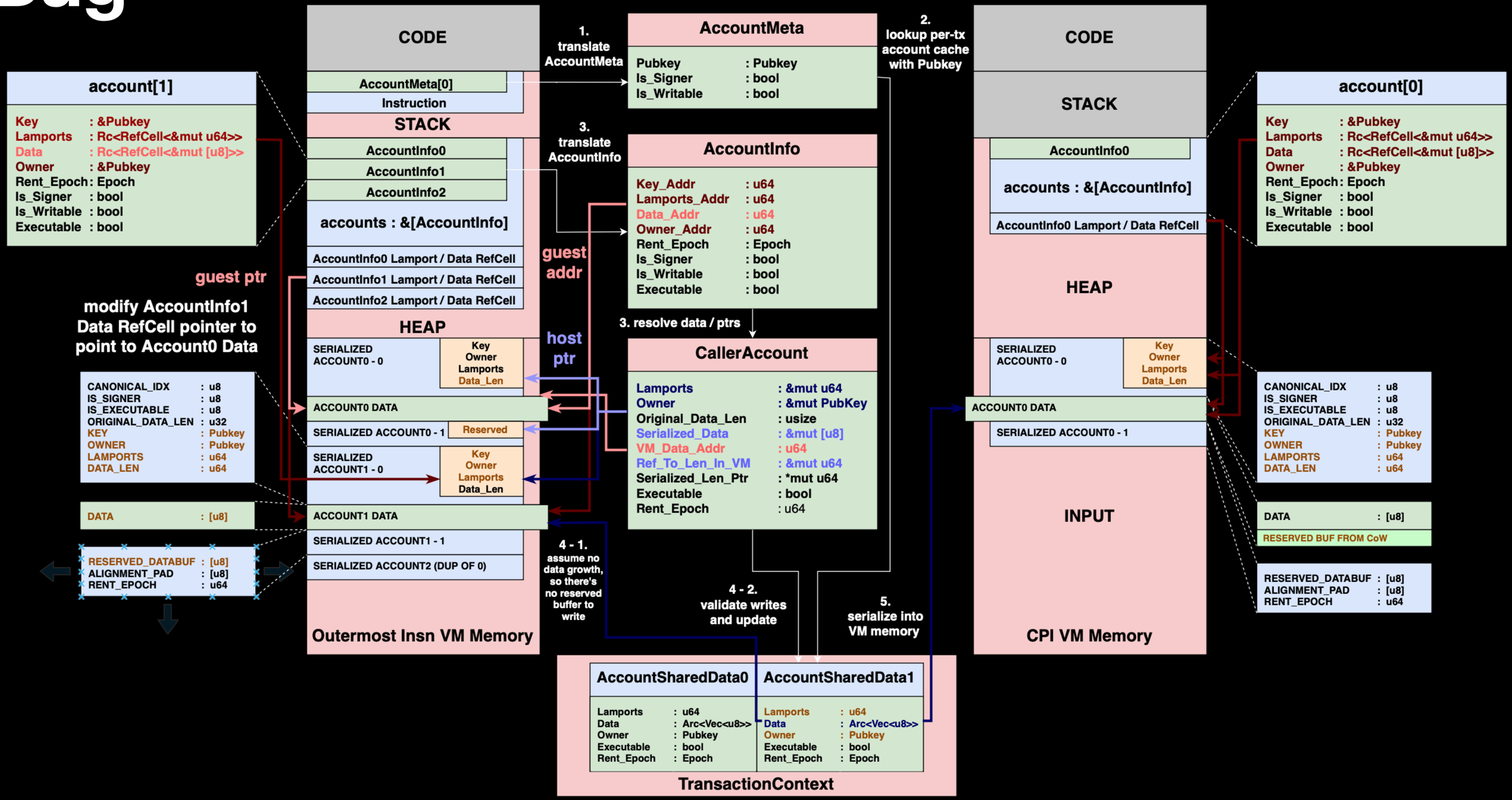
Missing check of guest address when updating MemoryRegion

- How does cpi know which region to update?
  - Search through MemoryRegions for matching virtual address
- Where does that virtual address came from?
  - CallerAccount, and users have full control over VM heap

```
if direct_mapping && caller_account.original_data_len > 0 {
    ...
    let region = memory_mapping.region(AccessType::Load, caller_account.vm_data_addr)?;
    let callee_ptr = callee_account.get_data().as_ptr() as u64;
    if region.host_addr.get() != callee_ptr {
        region.host_addr.set(callee_ptr);
    }
}
```

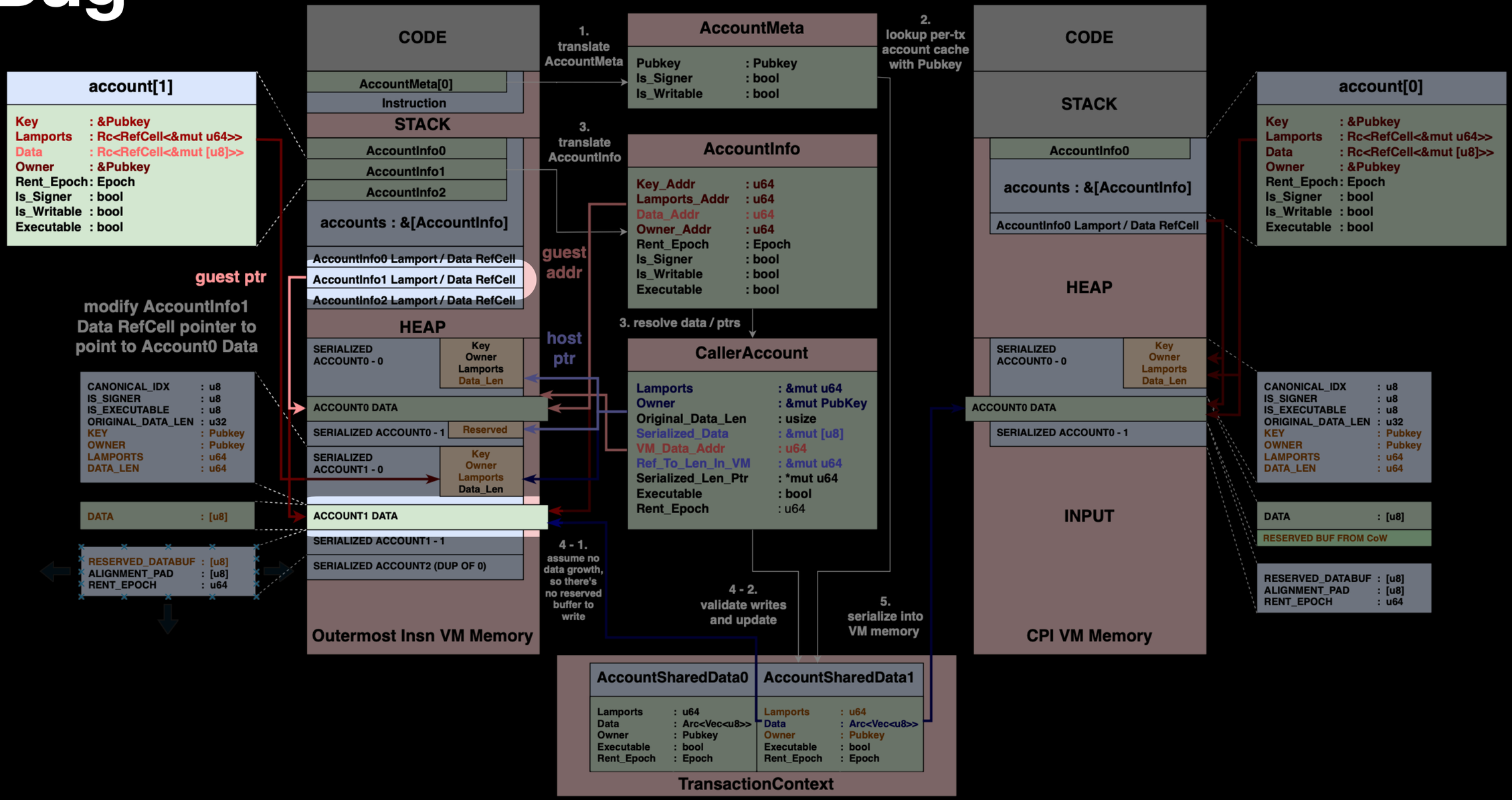
Bug

as a result, several host / guest pointers derived during CPI will also change



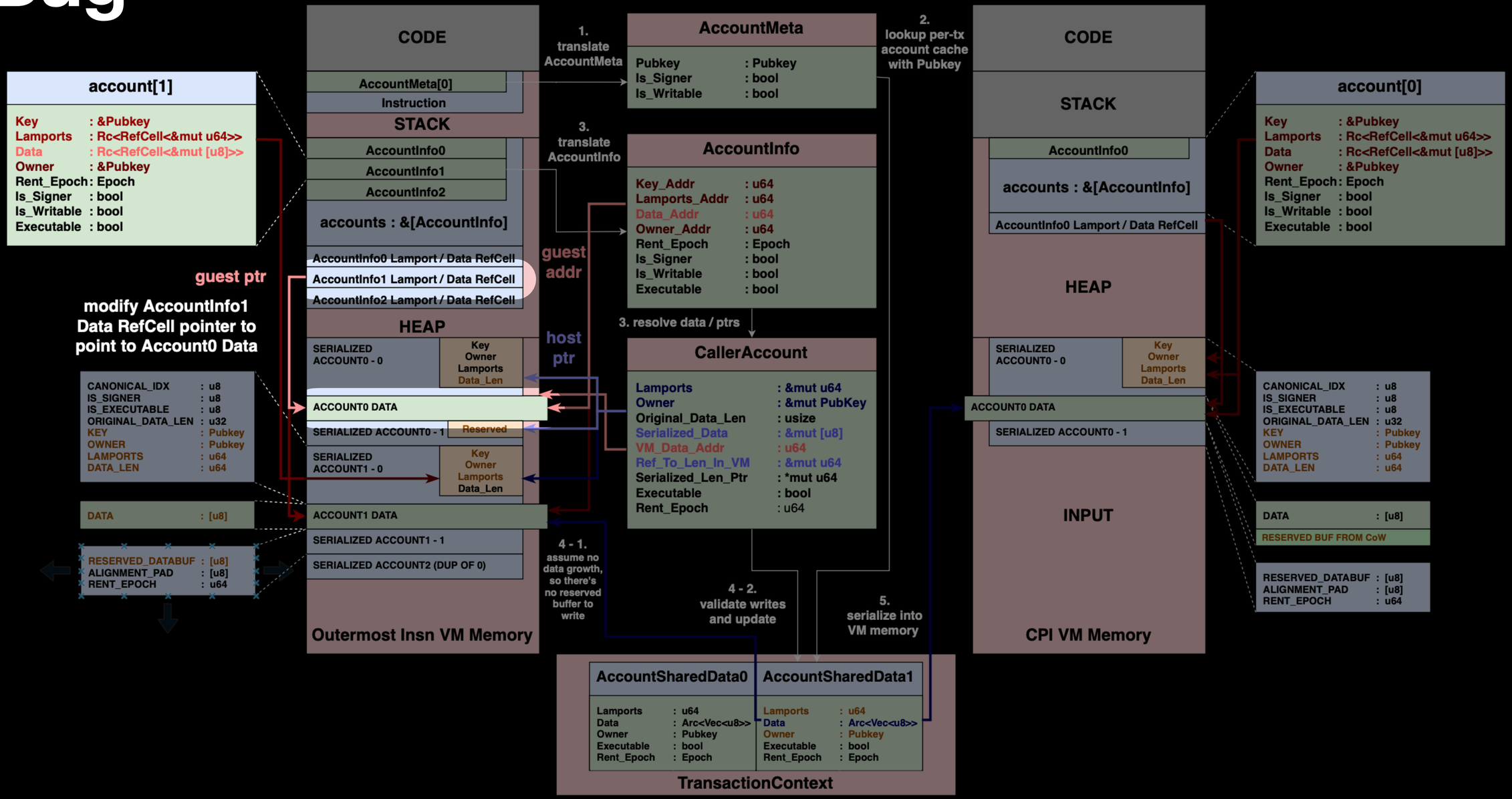
Bug

as a result, several host / guest pointers derived during CPI will also change

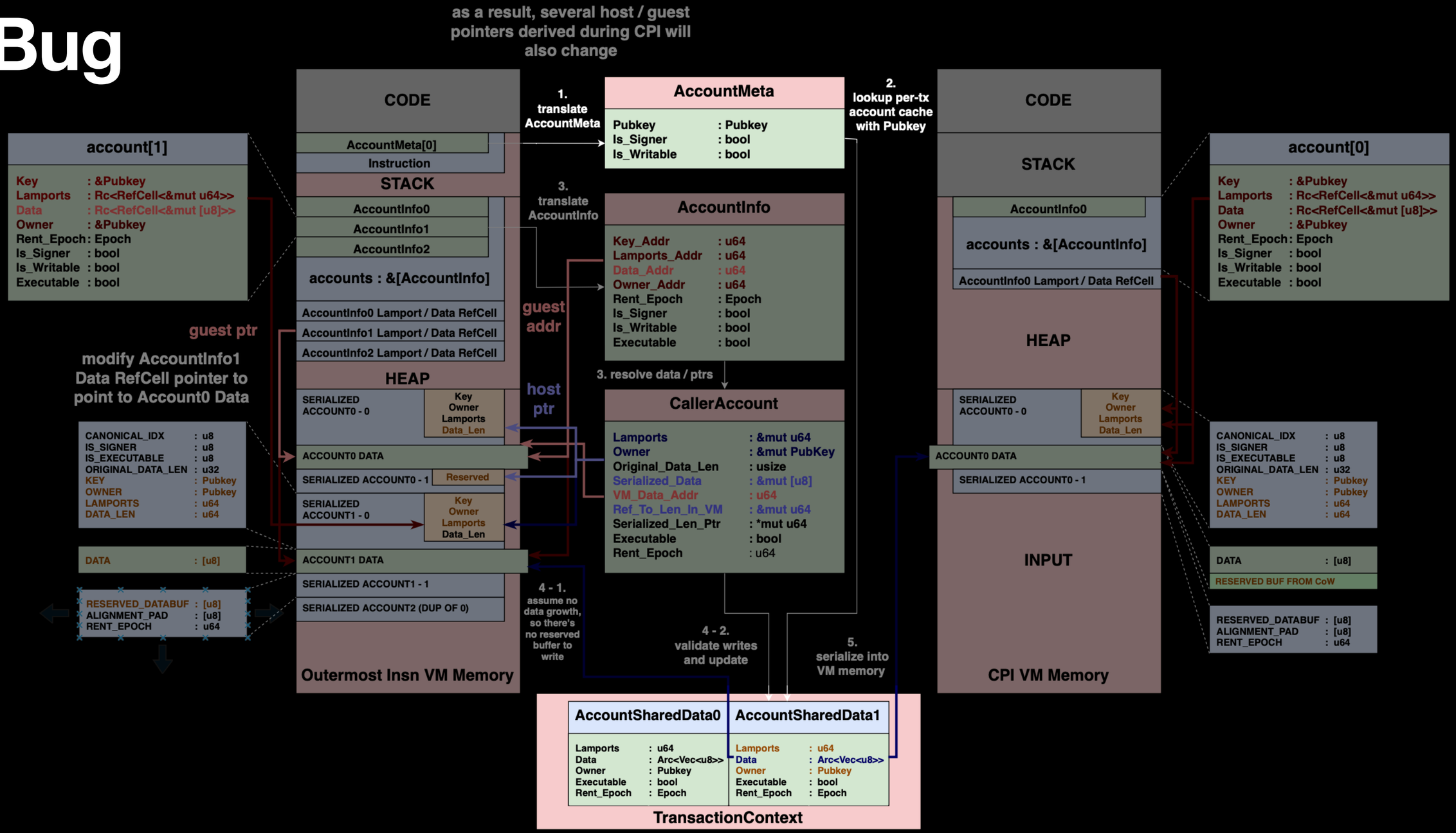


Bug

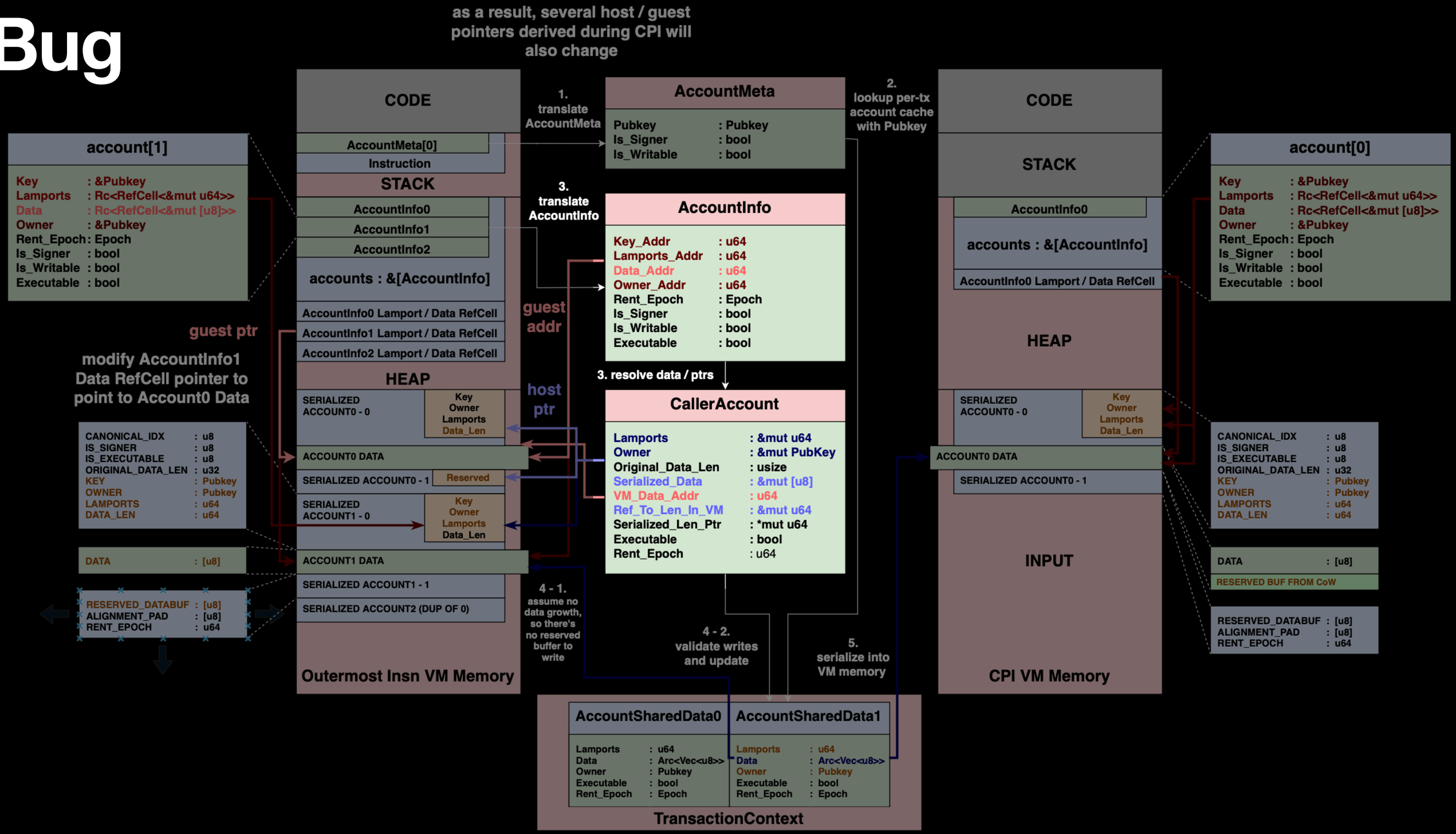
as a result, several host / guest pointers derived during CPI will also change



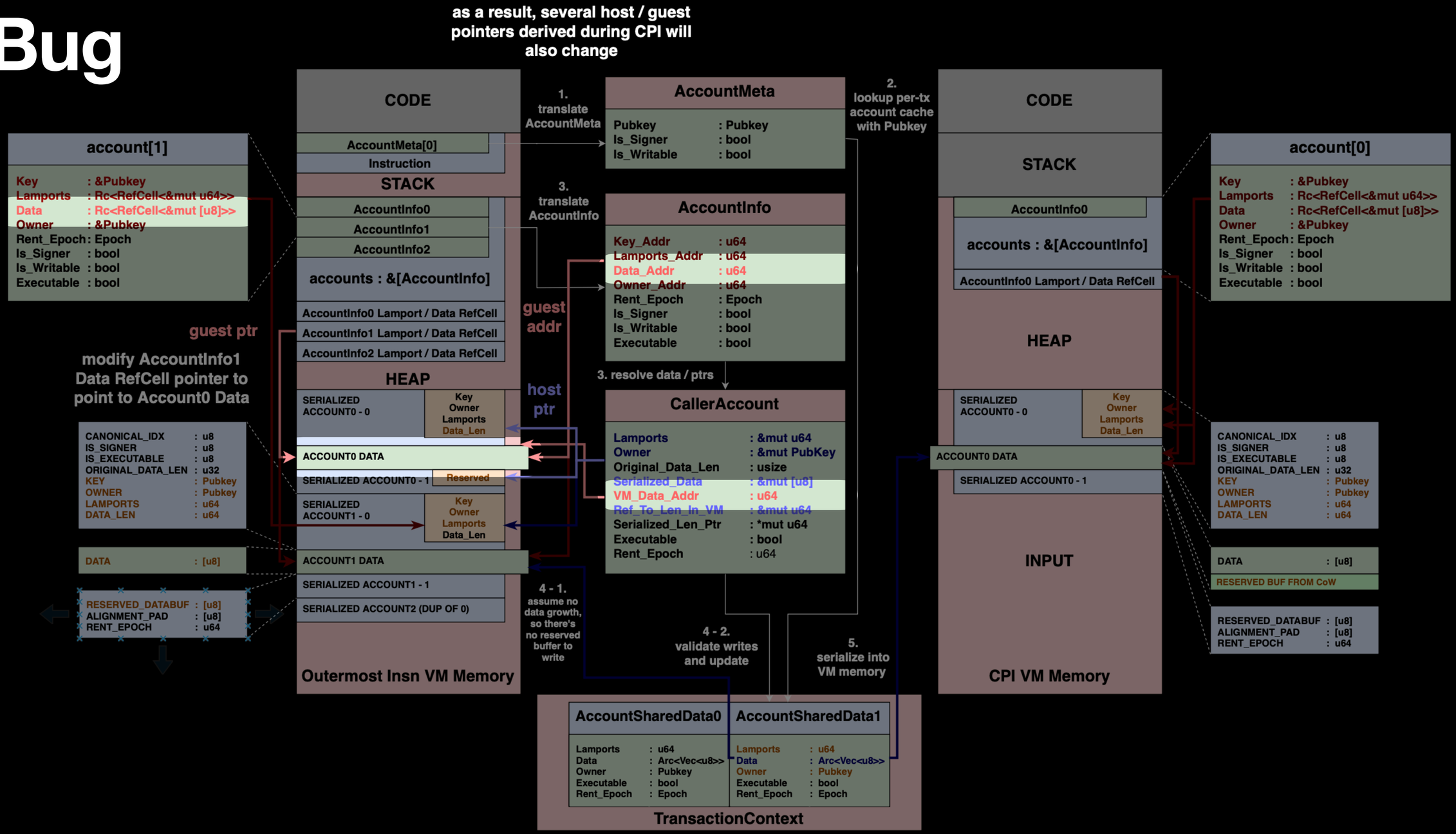
Bug


as a result, several host / guest pointers derived during CPI will also change
CODE
AccountMeta[0]
Instruction
STACK
AccountInfo0
AccountInfo1
AccountInfo2
accounts : &[AccountInfo]
AccountInfo0 Lamport / Data RefCell
AccountInfo1 Lamport / Data RefCell
AccountInfo2 Lamport / Data RefCell
HEAP
SERIALIZED ACCOUNT0 - 0
Key
Owner
Lamports
Data_Len
ACCOUNT0 DATA
SERIALIZED ACCOUNT0 - 1
Reserved
SERIALIZED ACCOUNT1 - 0
ACCOUNT1 DATA
SERIALIZED ACCOUNT1 - 1
SERIALIZED ACCOUNT2 (DUP OF 0)
Outermost Insn VM Memory
1. translate AccountMeta
AccountMeta
Pubkey : Pubkey
Is_Signer : bool
Is_Writable : bool
3. translate AccountInfo
AccountInfo
Key_Addr : u64
Lamports_Addr : u64
Data_Addr : u64
Owner_Addr : u64
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
guest addr
host ptr
3. resolve data / ptrs
CallerAccount
Lamports : &mut u64
Owner : &mut PubKey
Original_Data_Len : usize
Serialized_Data : &mut [u8]
VM_Data_Addr : u64
Ref_To_Len_In_VM : &mut u64
Serialized_Len_Ptr : *mut u64
Executable : bool
Rent_Epoch : u64
2. lookup per-tx account cache with Pubkey
CODE
STACK
AccountInfo0
accounts : &[AccountInfo]
AccountInfo0 Lamport / Data RefCell
HEAP
SERIALIZED ACCOUNT0 - 0
Key
Owner
Lamports
Data_Len
ACCOUNT0 DATA
SERIALIZED ACCOUNT0 - 1
INPUT
CPI VM Memory
account[1]
Key : &Pubkey
Lamports : Rc<RefCell<&mut u64>>
Data : Rc<RefCell<&mut [u8]>>
Owner : &Pubkey
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
guest ptr
modify AccountInfo1 Data RefCell pointer to point to Account0 Data
CANONICAL_IDX : u8
IS_SIGNER : u8
IS_EXECUTABLE : u8
ORIGINAL_DATA_LEN : u32
KEY : Pubkey
OWNER : Pubkey
LAMPORTS : u64
DATA_LEN : u64
DATA : [u8]
RESERVED_DATABUF : [u8]
ALIGNMENT_PAD : [u8]
RENT_EPOCH : u64
account[0]
Key : &Pubkey
Lamports : Rc<RefCell<&mut u64>>
Data : Rc<RefCell<&mut [u8]>>
Owner : &Pubkey
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
CANONICAL_IDX : u8
IS_SIGNER : u8
IS_EXECUTABLE : u8
ORIGINAL_DATA_LEN : u32
KEY : Pubkey
OWNER : Pubkey
LAMPORTS : u64
DATA_LEN : u64
DATA : [u8]
RESERVED BUF FROM CoW
RESERVED_DATABUF : [u8]
ALIGNMENT_PAD : [u8]
RENT_EPOCH : u64
4 - 1. assume no data growth, so there's no reserved buffer to write
4 - 2. validate writes and update
5. serialize into VM memory
AccountSharedData0
Lamports : u64
Data : Arc<Vec<u8>>
Owner : Pubkey
Executable : bool
Rent_Epoch : Epoch
AccountSharedData1
Lamports : u64
Data : Arc<Vec<u8>>
Owner : Pubkey
Executable : bool
Rent_Epoch : Epoch
TransactionContext

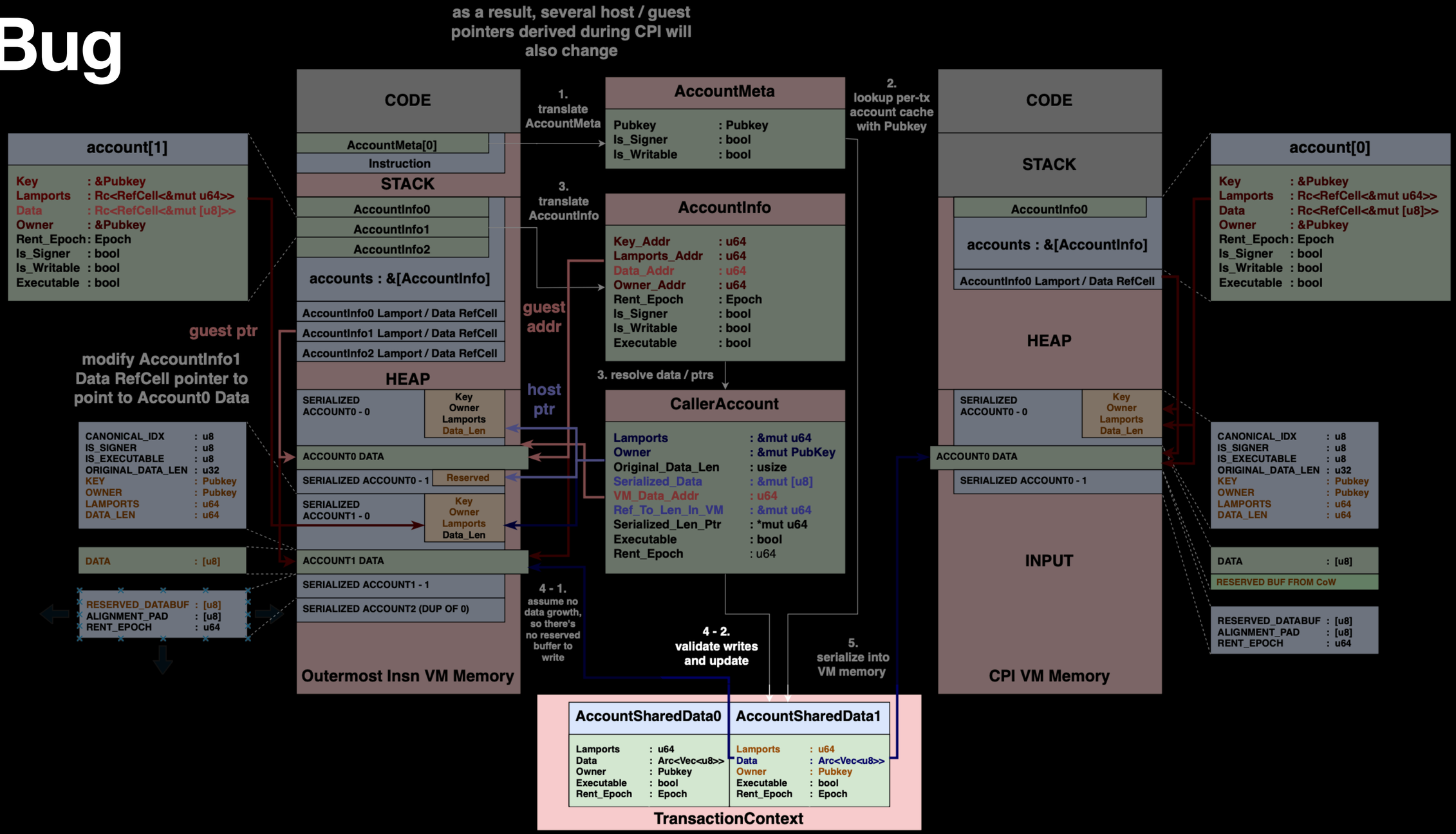
Bug


as a result, several host / guest pointers derived during CPI will also change
CODE
AccountMeta[0]
Instruction
STACK
AccountInfo0
AccountInfo1
AccountInfo2
accounts : &[AccountInfo]
AccountInfo0 Lamport / Data RefCell
AccountInfo1 Lamport / Data RefCell
AccountInfo2 Lamport / Data RefCell
HEAP
SERIALIZED ACCOUNT0 - 0
Key
Owner
Lamports
Data_Len
ACCOUNT0 DATA
SERIALIZED ACCOUNT0 - 1
Reserved
SERIALIZED ACCOUNT1 - 0
ACCOUNT1 DATA
SERIALIZED ACCOUNT1 - 1
SERIALIZED ACCOUNT2 (DUP OF 0)
Outermost Insn VM Memory
1. translate AccountMeta
AccountMeta
Pubkey : Pubkey
Is_Signer : bool
Is_Writable : bool
3. translate AccountInfo
AccountInfo
Key_Addr : u64
Lamports_Addr : u64
Data_Addr : u64
Owner_Addr : u64
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
guest addr
host ptr
3. resolve data / ptrs
CallerAccount
Lamports : &mut u64
Owner : &mut PubKey
Original_Data_Len : usize
Serialized_Data : &mut [u8]
VM_Data_Addr : u64
Ref_To_Len_In_VM : &mut u64
Serialized_Len_Ptr : *mut u64
Executable : bool
Rent_Epoch : u64
2. lookup per-tx account cache with Pubkey
CODE
STACK
AccountInfo0
accounts : &[AccountInfo]
AccountInfo0 Lamport / Data RefCell
HEAP
SERIALIZED ACCOUNT0 - 0
Key
Owner
Lamports
Data_Len
ACCOUNT0 DATA
SERIALIZED ACCOUNT0 - 1
INPUT
CPI VM Memory
account[1]
Key : &Pubkey
Lamports : Rc<RefCell<&mut u64>>
Data : Rc<RefCell<&mut [u8]>>
Owner : &Pubkey
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
guest ptr
modify AccountInfo1 Data RefCell pointer to point to Account0 Data
CANONICAL_IDX : u8
IS_SIGNER : u8
IS_EXECUTABLE : u8
ORIGINAL_DATA_LEN : u32
KEY : Pubkey
OWNER : Pubkey
LAMPORTS : u64
DATA_LEN : u64
DATA : [u8]
RESERVED_DATABUF : [u8]
ALIGNMENT_PAD : [u8]
RENT_EPOCH : u64
account[0]
Key : &Pubkey
Lamports : Rc<RefCell<&mut u64>>
Data : Rc<RefCell<&mut [u8]>>
Owner : &Pubkey
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
CANONICAL_IDX : u8
IS_SIGNER : u8
IS_EXECUTABLE : u8
ORIGINAL_DATA_LEN : u32
KEY : Pubkey
OWNER : Pubkey
LAMPORTS : u64
DATA_LEN : u64
DATA : [u8]
RESERVED BUF FROM CoW
RESERVED_DATABUF : [u8]
ALIGNMENT_PAD : [u8]
RENT_EPOCH : u64
4 - 1. assume no data growth, so there's no reserved buffer to write
4 - 2. validate writes and update
5. serialize into VM memory
AccountSharedData0
Lamports : u64
Data : Arc<Vec<u8>>
Owner : Pubkey
Executable : bool
Rent_Epoch : Epoch
AccountSharedData1
Lamports : u64
Data : Arc<Vec<u8>>
Owner : Pubkey
Executable : bool
Rent_Epoch : Epoch
TransactionContext

Bug


as a result, several host / guest pointers derived during CPI will also change
CODE
AccountMeta[0]
Instruction
STACK
AccountInfo0
AccountInfo1
AccountInfo2
accounts : &[AccountInfo]
AccountInfo0 Lamport / Data RefCell
AccountInfo1 Lamport / Data RefCell
AccountInfo2 Lamport / Data RefCell
HEAP
SERIALIZED ACCOUNT0 - 0
Key
Owner
Lamports
Data_Len
ACCOUNT0 DATA
SERIALIZED ACCOUNT0 - 1
Reserved
SERIALIZED ACCOUNT1 - 0
ACCOUNT1 DATA
SERIALIZED ACCOUNT1 - 1
SERIALIZED ACCOUNT2 (DUP OF 0)
Outermost Insn VM Memory
1. translate AccountMeta
AccountMeta
Pubkey : Pubkey
Is_Signer : bool
Is_Writable : bool
3. translate AccountInfo
AccountInfo
Key_Addr : u64
Lamports_Addr : u64
Data_Addr : u64
Owner_Addr : u64
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
guest addr
host ptr
3. resolve data / ptrs
CallerAccount
Lamports : &mut u64
Owner : &mut PubKey
Original_Data_Len : usize
Serialized_Data : &mut [u8]
VM_Data_Addr : u64
Ref_To_Len_In_VM : &mut u64
Serialized_Len_Ptr : *mut u64
Executable : bool
Rent_Epoch : u64
4 - 1. assume no data growth, so there's no reserved buffer to write
4 - 2. validate writes and update
5. serialize into VM memory
2. lookup per-tx account cache with Pubkey
CODE
STACK
AccountInfo0
accounts : &[AccountInfo]
AccountInfo0 Lamport / Data RefCell
HEAP
SERIALIZED ACCOUNT0 - 0
Key
Owner
Lamports
Data_Len
ACCOUNT0 DATA
SERIALIZED ACCOUNT0 - 1
INPUT
CPI VM Memory
account[1]
Key : &Pubkey
Lamports : Rc<RefCell<&mut u64>>
Data : Rc<RefCell<&mut [u8]>>
Owner : &Pubkey
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
guest ptr
modify AccountInfo1 Data RefCell pointer to point to Account0 Data
CANONICAL_IDX : u8
IS_SIGNER : u8
IS_EXECUTABLE : u8
ORIGINAL_DATA_LEN : u32
KEY : Pubkey
OWNER : Pubkey
LAMPORTS : u64
DATA_LEN : u64
DATA : [u8]
RESERVED_DATABUF : [u8]
ALIGNMENT_PAD : [u8]
RENT_EPOCH : u64
account[0]
Key : &Pubkey
Lamports : Rc<RefCell<&mut u64>>
Data : Rc<RefCell<&mut [u8]>>
Owner : &Pubkey
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
CANONICAL_IDX : u8
IS_SIGNER : u8
IS_EXECUTABLE : u8
ORIGINAL_DATA_LEN : u32
KEY : Pubkey
OWNER : Pubkey
LAMPORTS : u64
DATA_LEN : u64
DATA : [u8]
RESERVED BUF FROM CoW
RESERVED_DATABUF : [u8]
ALIGNMENT_PAD : [u8]
RENT_EPOCH : u64
AccountSharedData0
Lamports : u64
Data : Arc<Vec<u8>>
Owner : Pubkey
Executable : bool
Rent_Epoch : Epoch
AccountSharedData1
Lamports : u64
Data : Arc<Vec<u8>>
Owner : Pubkey
Executable : bool
Rent_Epoch : Epoch
TransactionContext

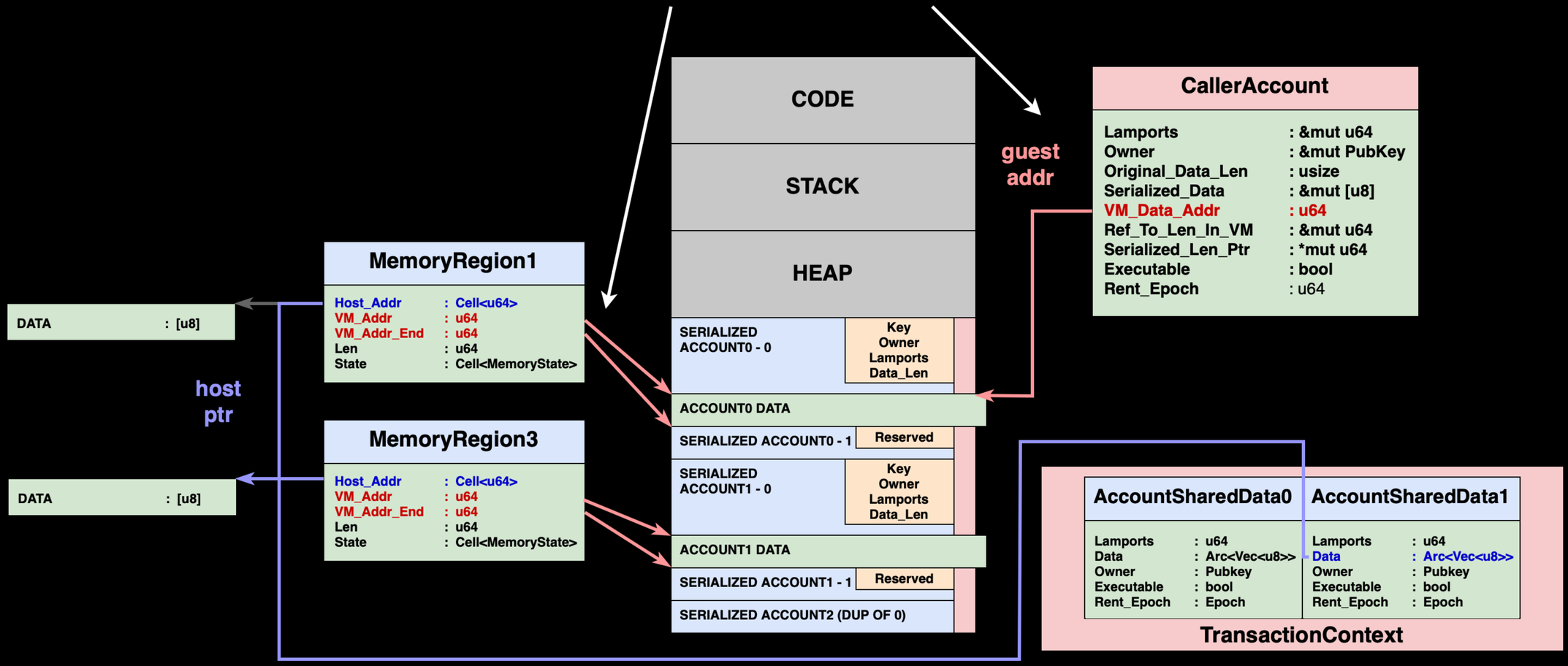
Bug


as a result, several host / guest pointers derived during CPI will also change
CODE
AccountMeta[0]
Instruction
STACK
AccountInfo0
AccountInfo1
AccountInfo2
accounts : &[AccountInfo]
AccountInfo0 Lamport / Data RefCell
AccountInfo1 Lamport / Data RefCell
AccountInfo2 Lamport / Data RefCell
HEAP
SERIALIZED ACCOUNT0 - 0
Key Owner Lamports Data_Len
ACCOUNT0 DATA
SERIALIZED ACCOUNT0 - 1
Reserved
SERIALIZED ACCOUNT1 - 0
Key Owner Lamports Data_Len
ACCOUNT1 DATA
SERIALIZED ACCOUNT1 - 1
SERIALIZED ACCOUNT2 (DUP OF 0)
Outermost Insn VM Memory
1. translate AccountMeta
AccountMeta
Pubkey : Pubkey
Is_Signer : bool
Is_Writable : bool
3. translate AccountInfo
AccountInfo
Key_Addr : u64
Lamports_Addr : u64
Data_Addr : u64
Owner_Addr : u64
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
guest addr
host ptr
3. resolve data / ptrs
CallerAccount
Lamports : &mut u64
Owner : &mut PubKey
Original_Data_Len : usize
Serialized_Data : &mut [u8]
VM_Data_Addr : u64
Ref_To_Len_In_VM : &mut u64
Serialized_Len_Ptr : *mut u64
Executable : bool
Rent_Epoch : u64
2. lookup per-tx account cache with Pubkey
CODE
STACK
AccountInfo0
accounts : &[AccountInfo]
AccountInfo0 Lamport / Data RefCell
HEAP
SERIALIZED ACCOUNT0 - 0
Key Owner Lamports Data_Len
ACCOUNT0 DATA
SERIALIZED ACCOUNT0 - 1
INPUT
CPI VM Memory
account[1]
Key : &Pubkey
Lamports : Rc<RefCell<&mut u64>>
Data : Rc<RefCell<&mut [u8]>>
Owner : &Pubkey
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
guest ptr
modify AccountInfo1 Data RefCell pointer to point to Account0 Data
CANONICAL_IDX : u8
IS_SIGNER : u8
IS_EXECUTABLE : u8
ORIGINAL_DATA_LEN : u32
KEY : Pubkey
OWNER : Pubkey
LAMPORTS : u64
DATA_LEN : u64
DATA : [u8]
RESERVED_DATABUF : [u8]
ALIGNMENT_PAD : [u8]
RENT_EPOCH : u64
account[0]
Key : &Pubkey
Lamports : Rc<RefCell<&mut u64>>
Data : Rc<RefCell<&mut [u8]>>
Owner : &Pubkey
Rent_Epoch : Epoch
Is_Signer : bool
Is_Writable : bool
Executable : bool
CANONICAL_IDX : u8
IS_SIGNER : u8
IS_EXECUTABLE : u8
ORIGINAL_DATA_LEN : u32
KEY : Pubkey
OWNER : Pubkey
LAMPORTS : u64
DATA_LEN : u64
DATA : [u8]
RESERVED BUF FROM CoW
RESERVED_DATABUF : [u8]
ALIGNMENT_PAD : [u8]
RENT_EPOCH : u64
4 - 1. assume no data growth, so there's no reserved buffer to write
4 - 2. validate writes and update
5. serialize into VM memory
AccountSharedData0
Lamports : u64
Data : Arc<Vec<u8>>
Owner : Pubkey
Executable : bool
Rent_Epoch : Epoch
AccountSharedData1
Lamports : u64
Data : Arc<Vec<u8>>
Owner : Pubkey
Executable : bool
Rent_Epoch : Epoch
TransactionContext

Bug

cpi return

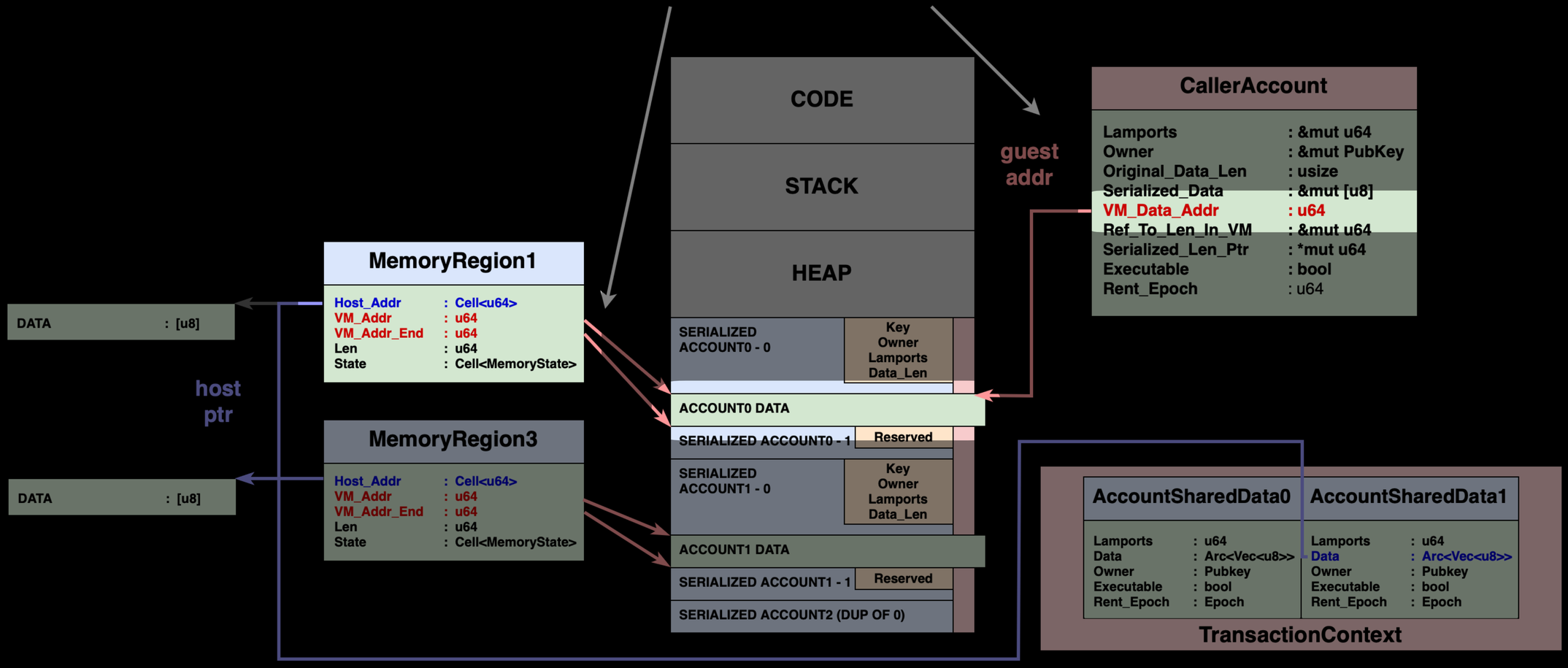
1. Search returns MemoryRegion1 (Region for Account0 Data)
2. The Host_Addr is compared against Account1 Data Buffer Addr
3. A mismatch occurs and Host_Addr is modified



Bug

cpi return

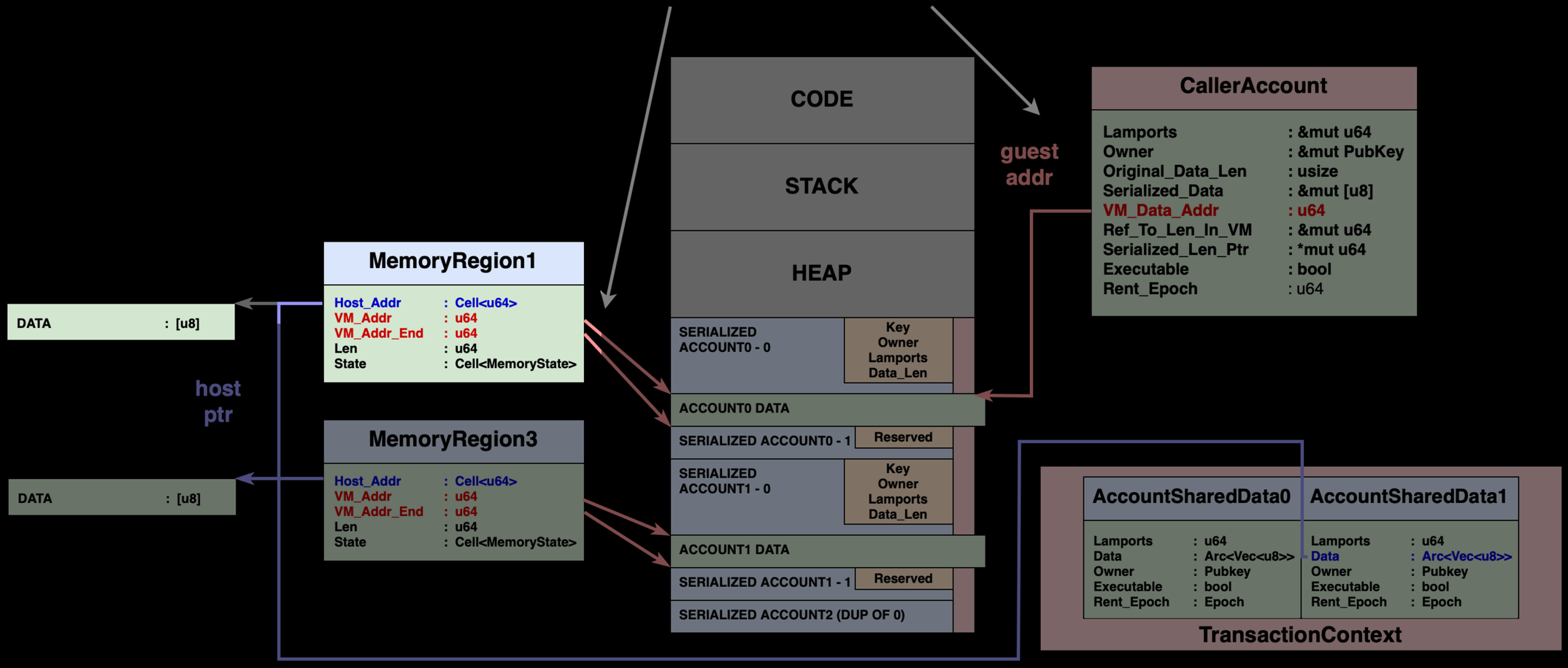
1. Search returns MemoryRegion1 (Region for Account0 Data)
2. The Host_Addr is compared against Account1 Data Buffer Addr
3. A mismatch occurs and Host_Addr is modified



Bug

cpi return

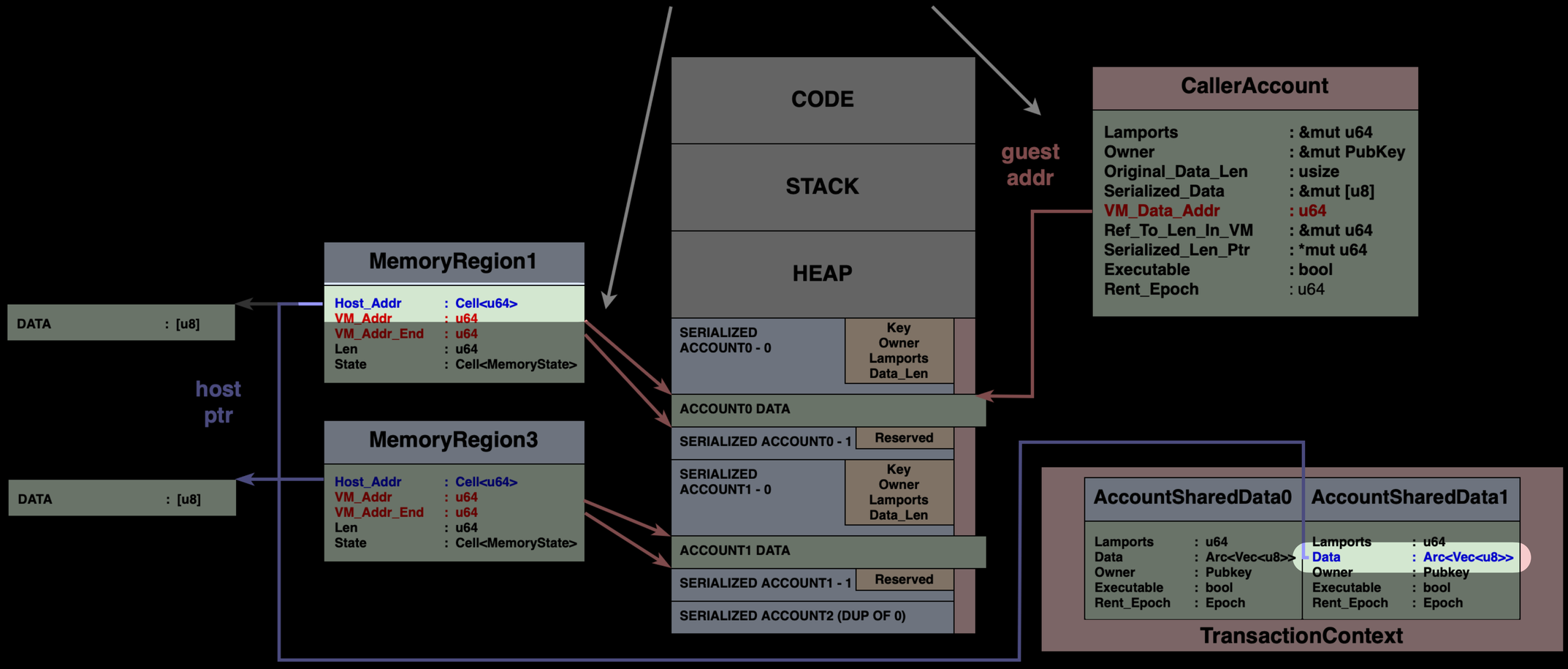
1. Search returns MemoryRegion1 (Region for Account0 Data)
2. The Host_Addr is compared against Account1 Data Buffer Addr
3. A mismatch occurs and Host_Addr is modified



Bug

cpi return

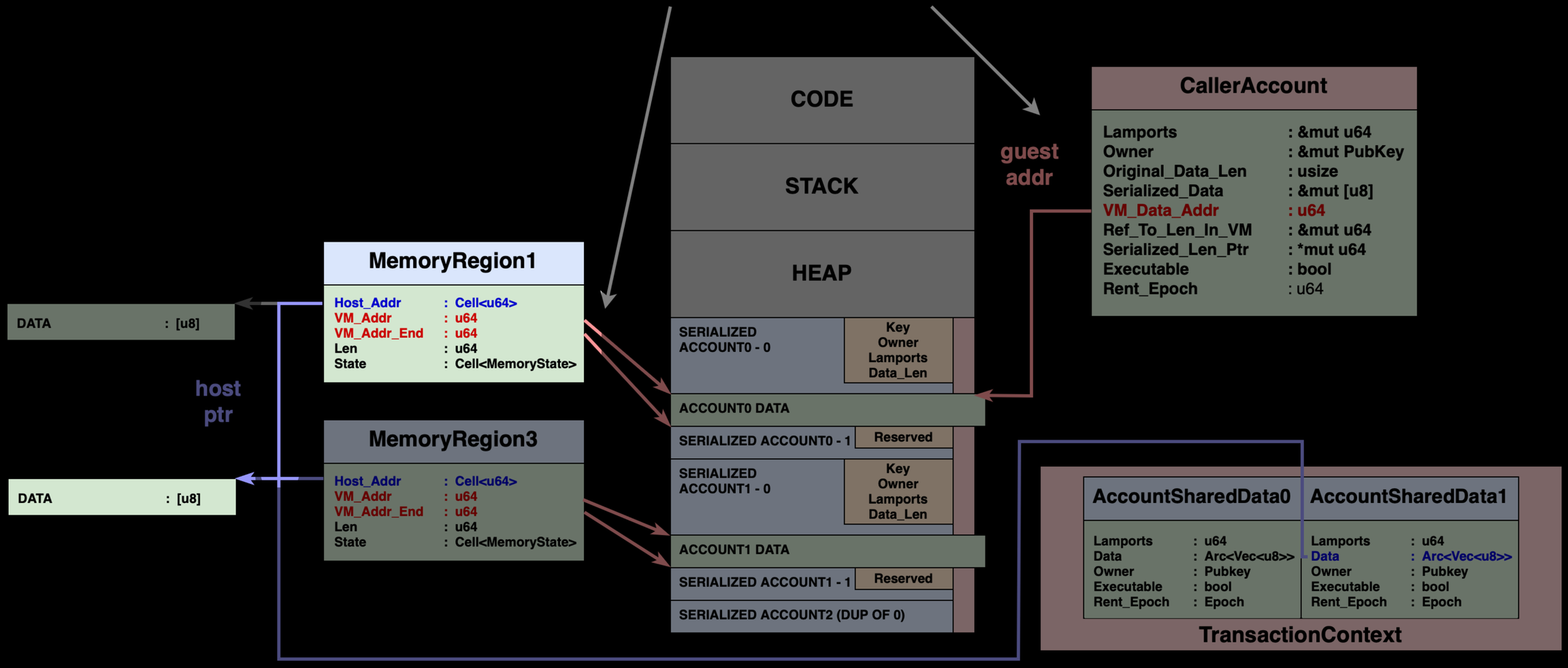
1. Search returns MemoryRegion1 (Region for Account0 Data)
2. The Host_Addr is compared against Account1 Data Buffer Addr
3. A mismatch occurs and Host_Addr is modified



Bug

cpi return

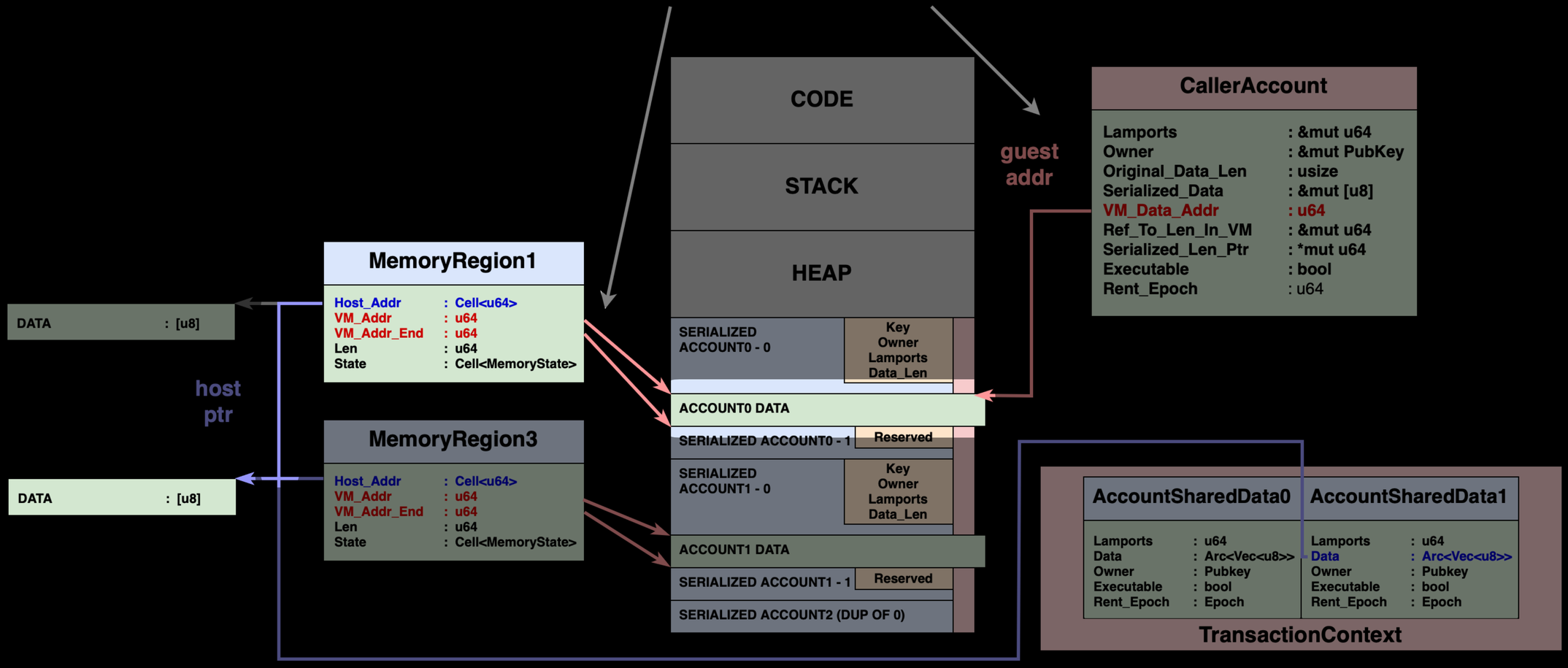
1. Search returns MemoryRegion1 (Region for Account0 Data)
2. The Host_Addr is compared against Account1 Data Buffer Addr
3. A mismatch occurs and Host_Addr is modified



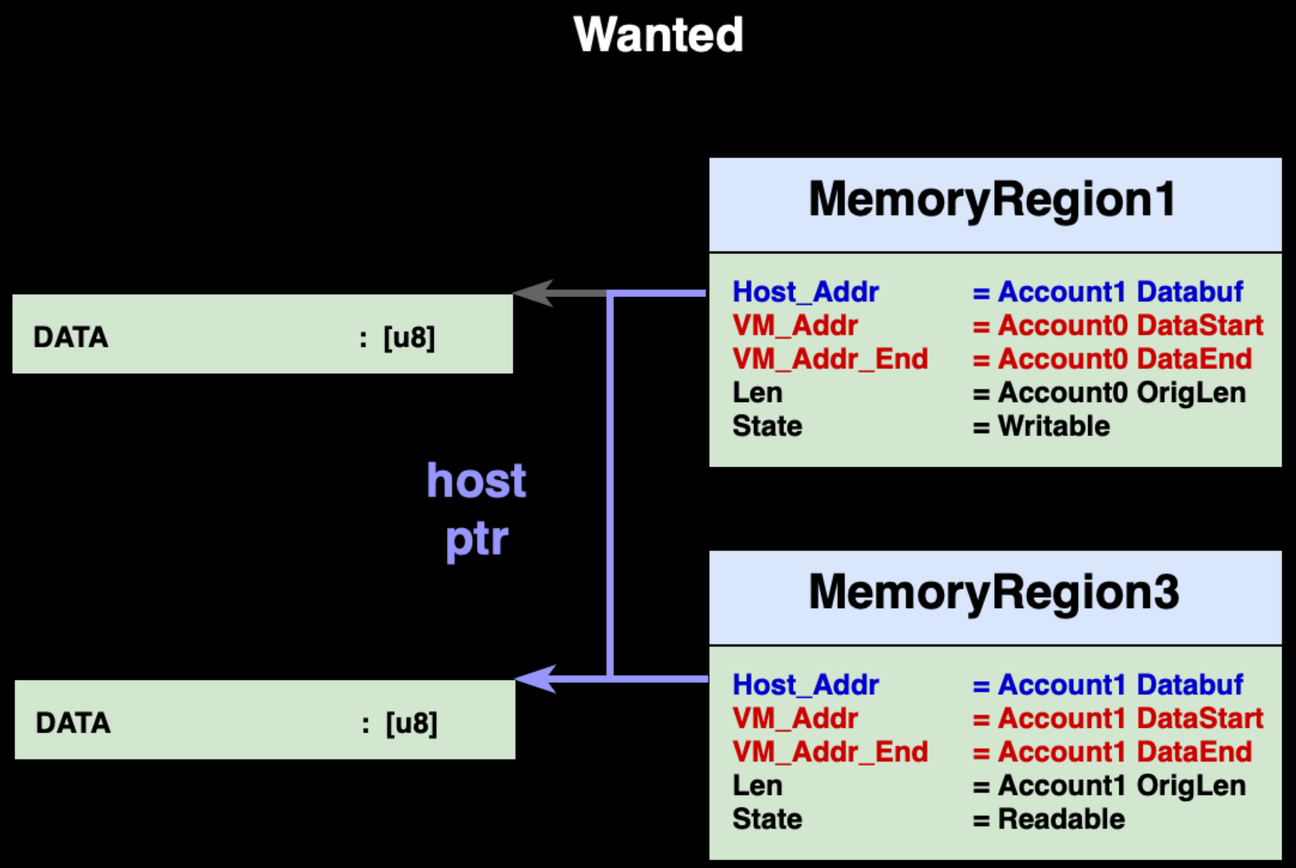
Bug

cpi return

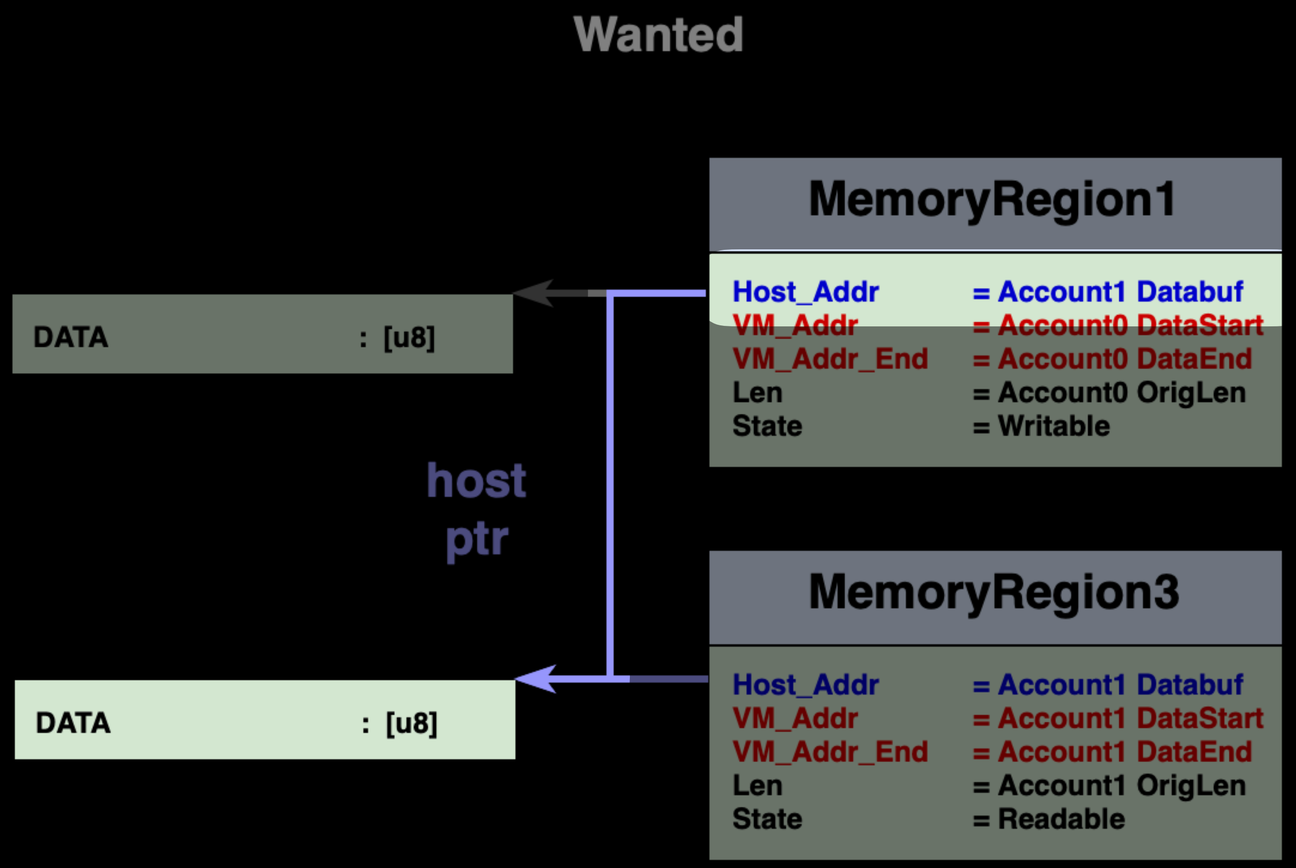
1. Search returns MemoryRegion1 (Region for Account0 Data)
2. The Host_Addr is compared against Account1 Data Buffer Addr
3. A mismatch occurs and Host_Addr is modified



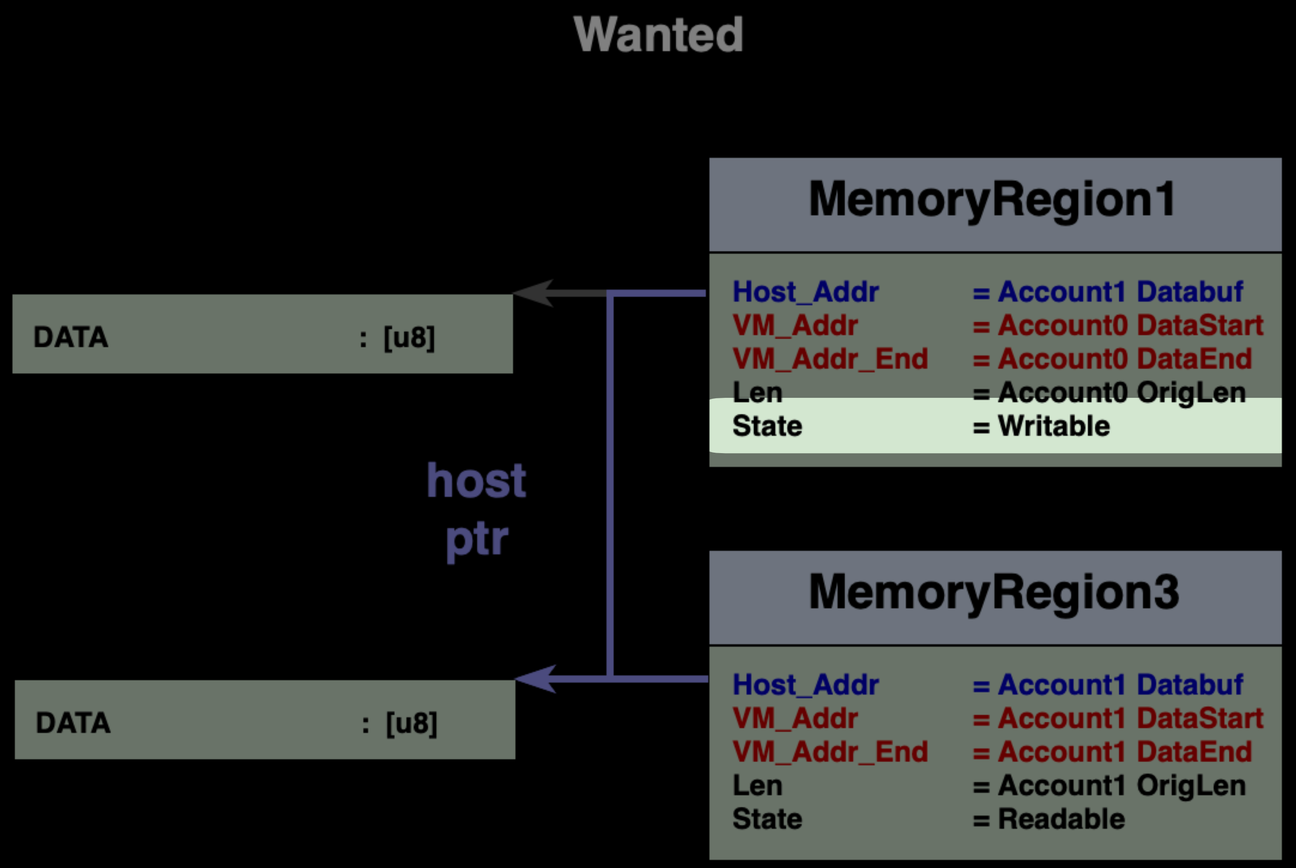
Bug
Failed plan



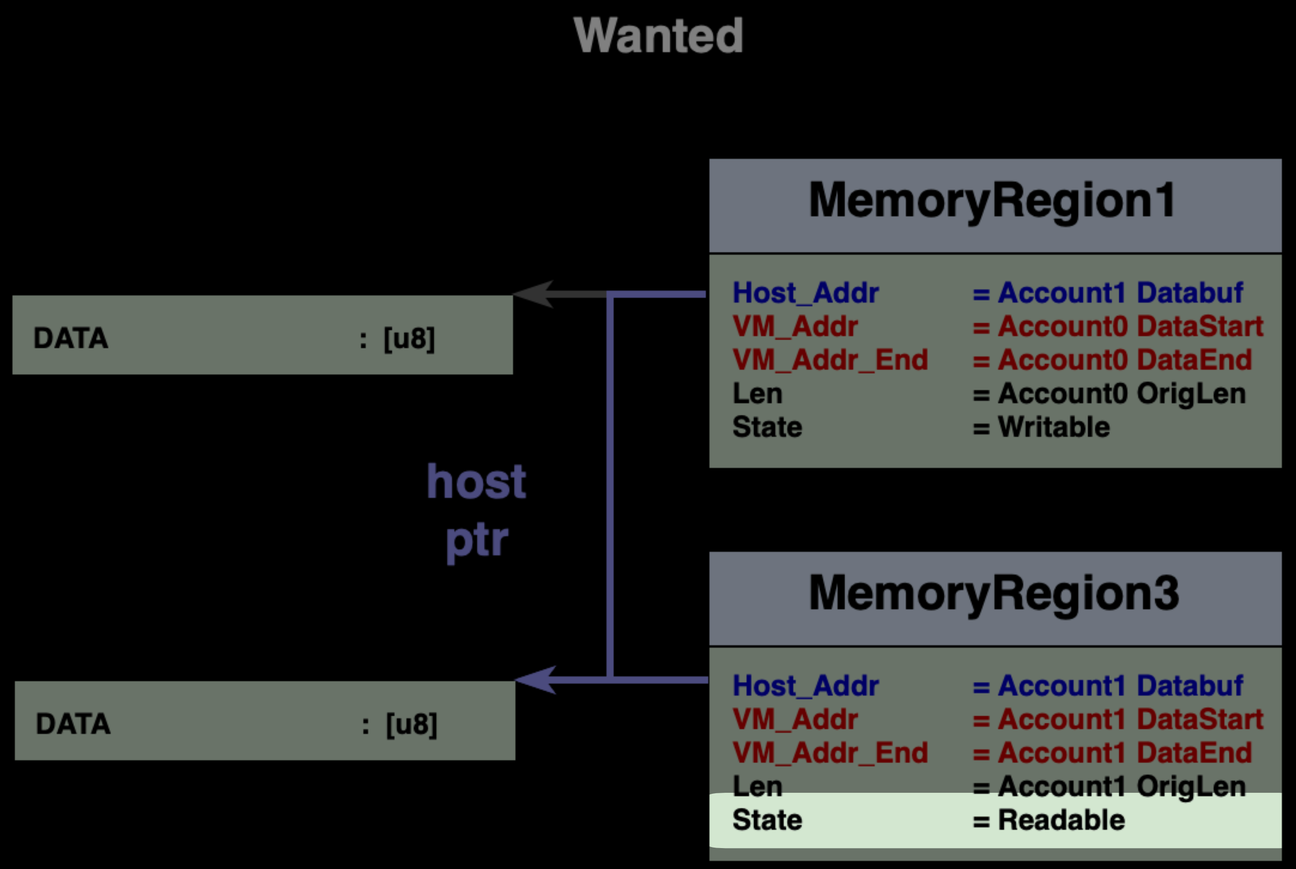
Bug
Failed plan



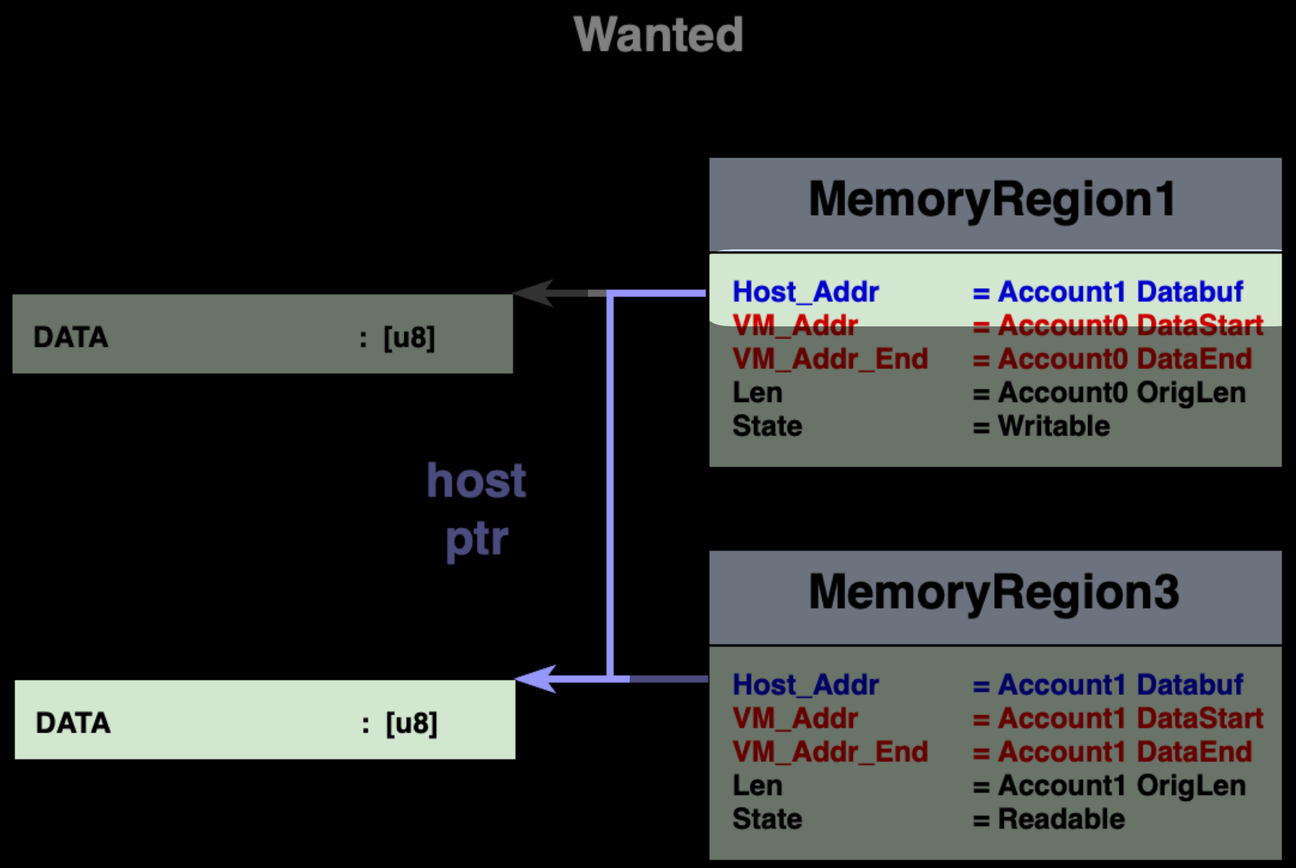
Bug
Failed plan



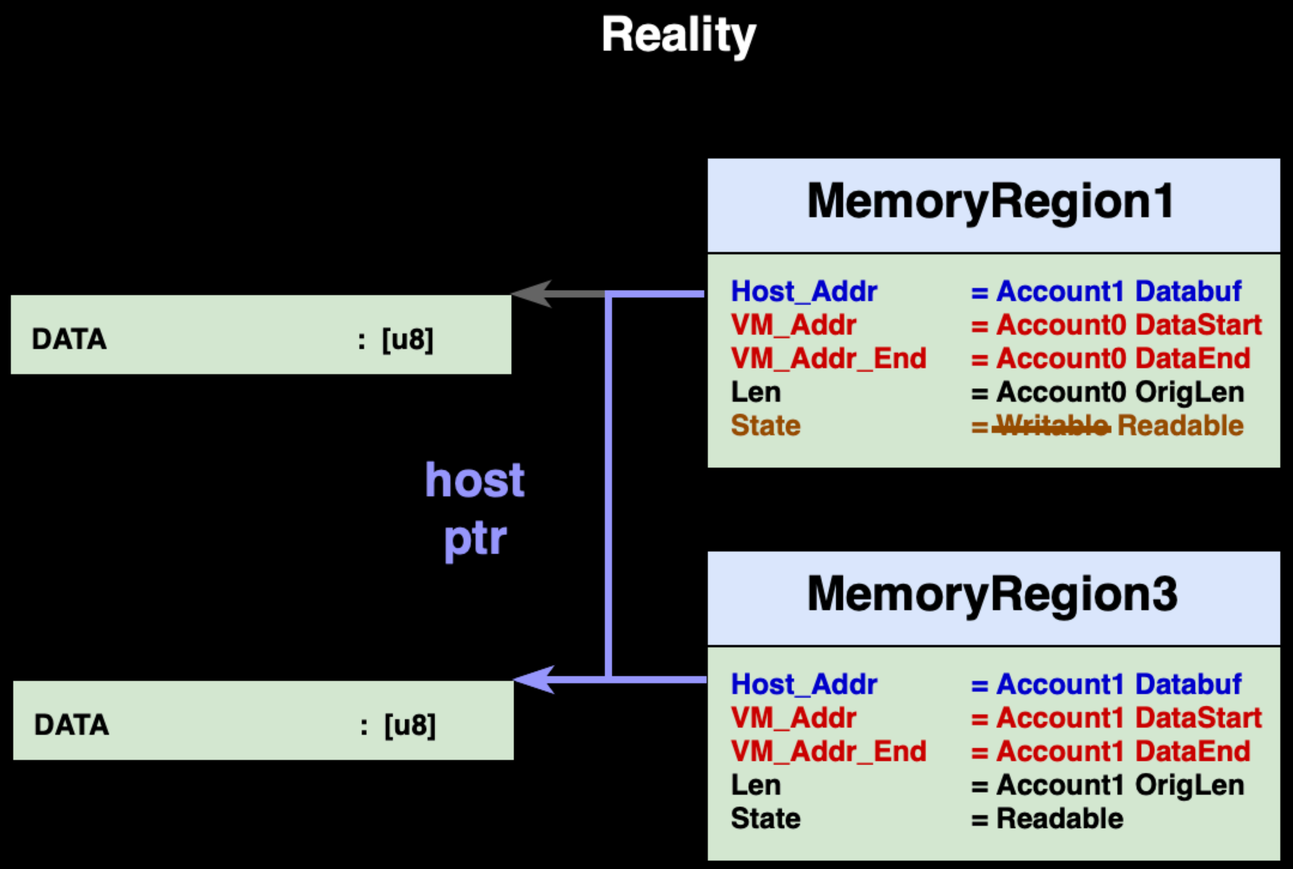
Bug
Failed plan



Bug
Failed plan

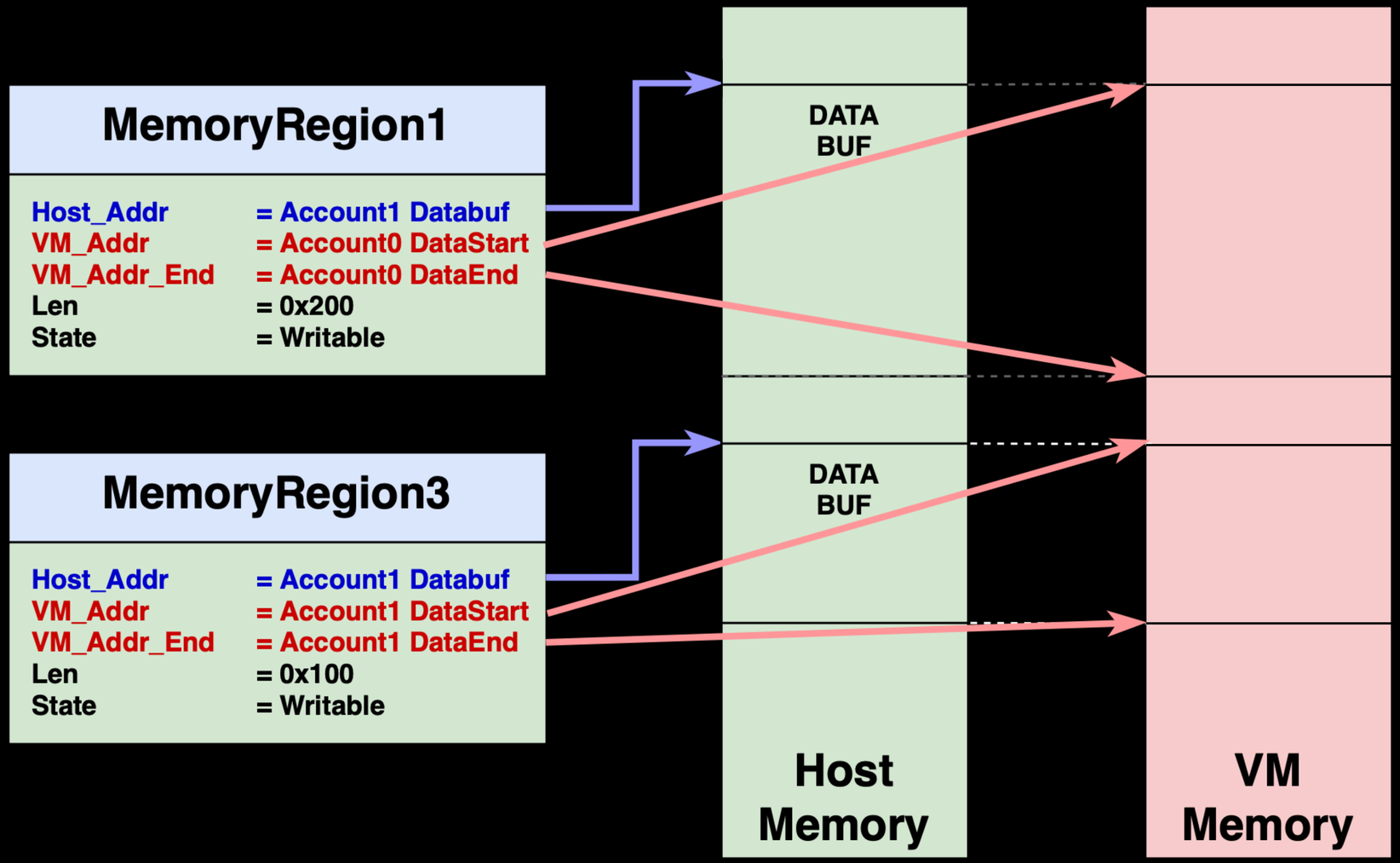


Bug
Failed plan



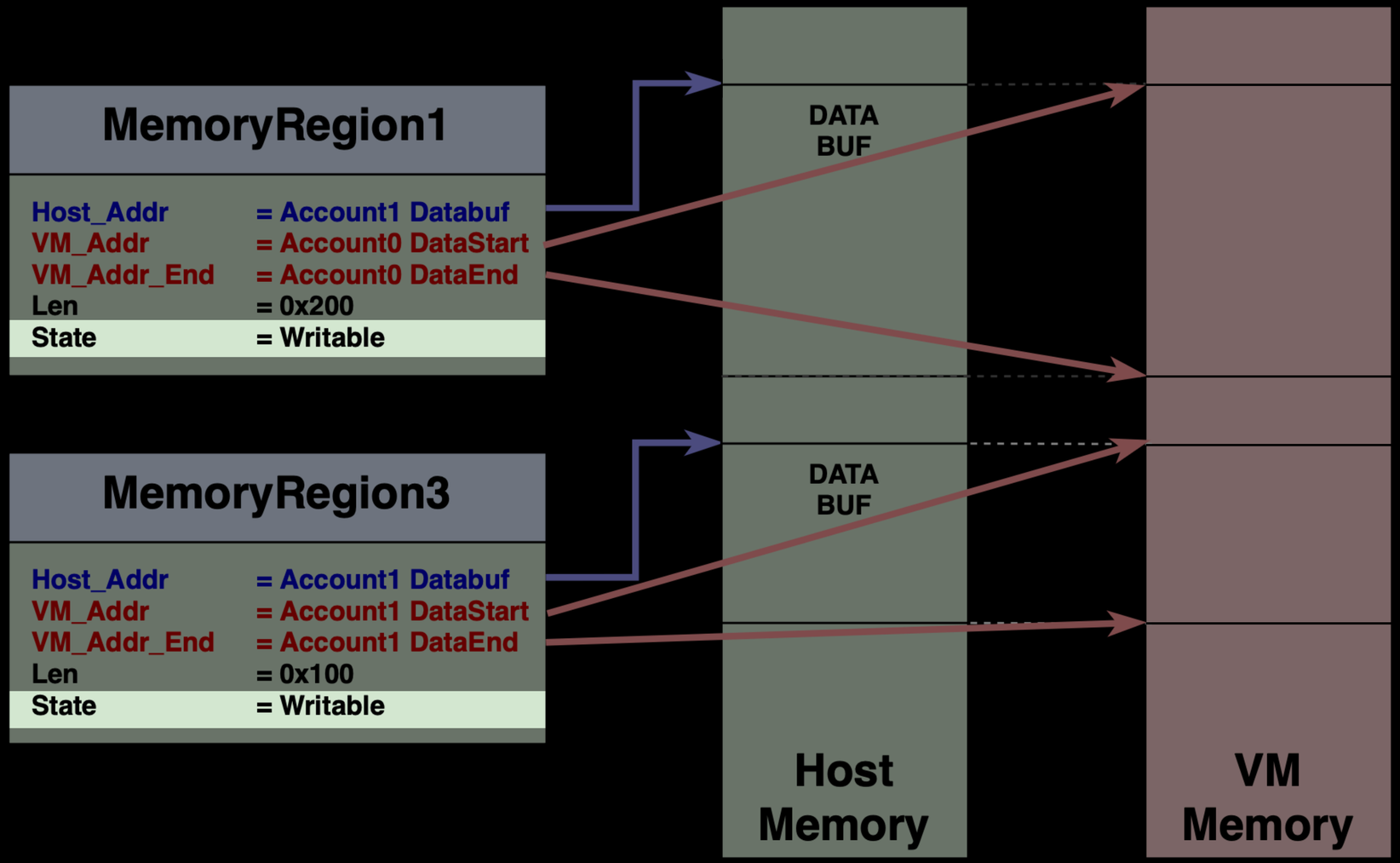
Bug

Out of bound write



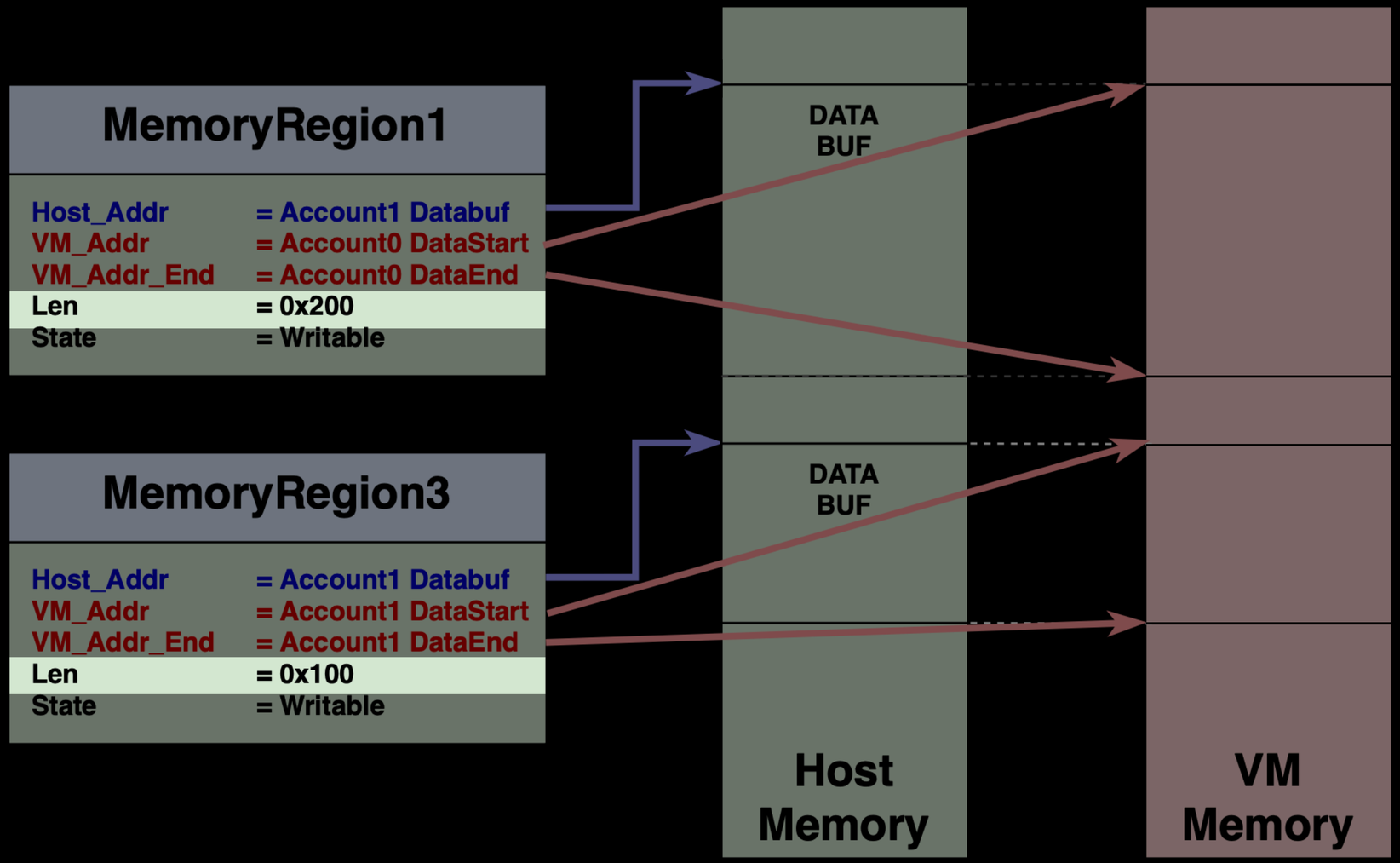
Bug

Out of bound write



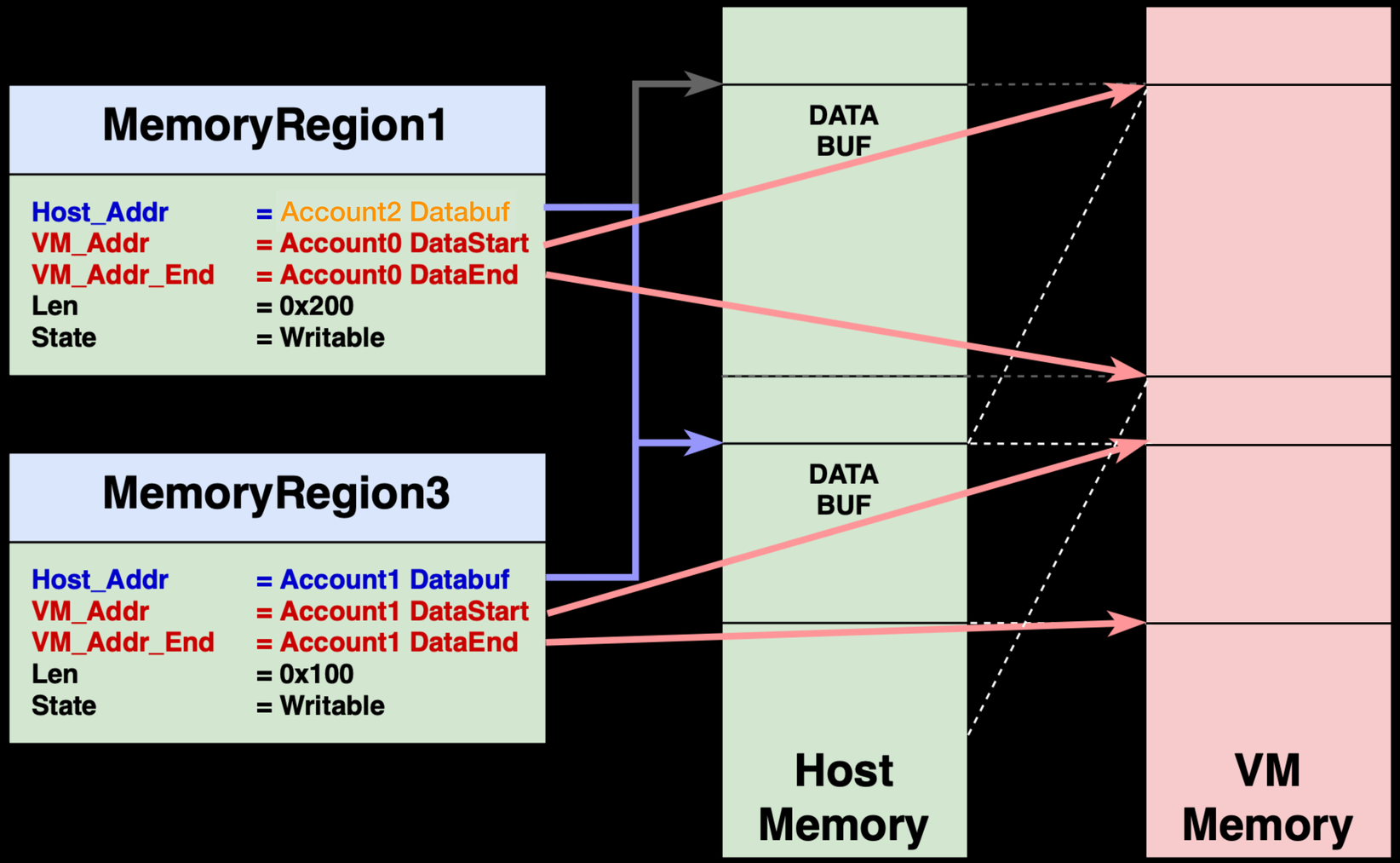
Bug

Out of bound write



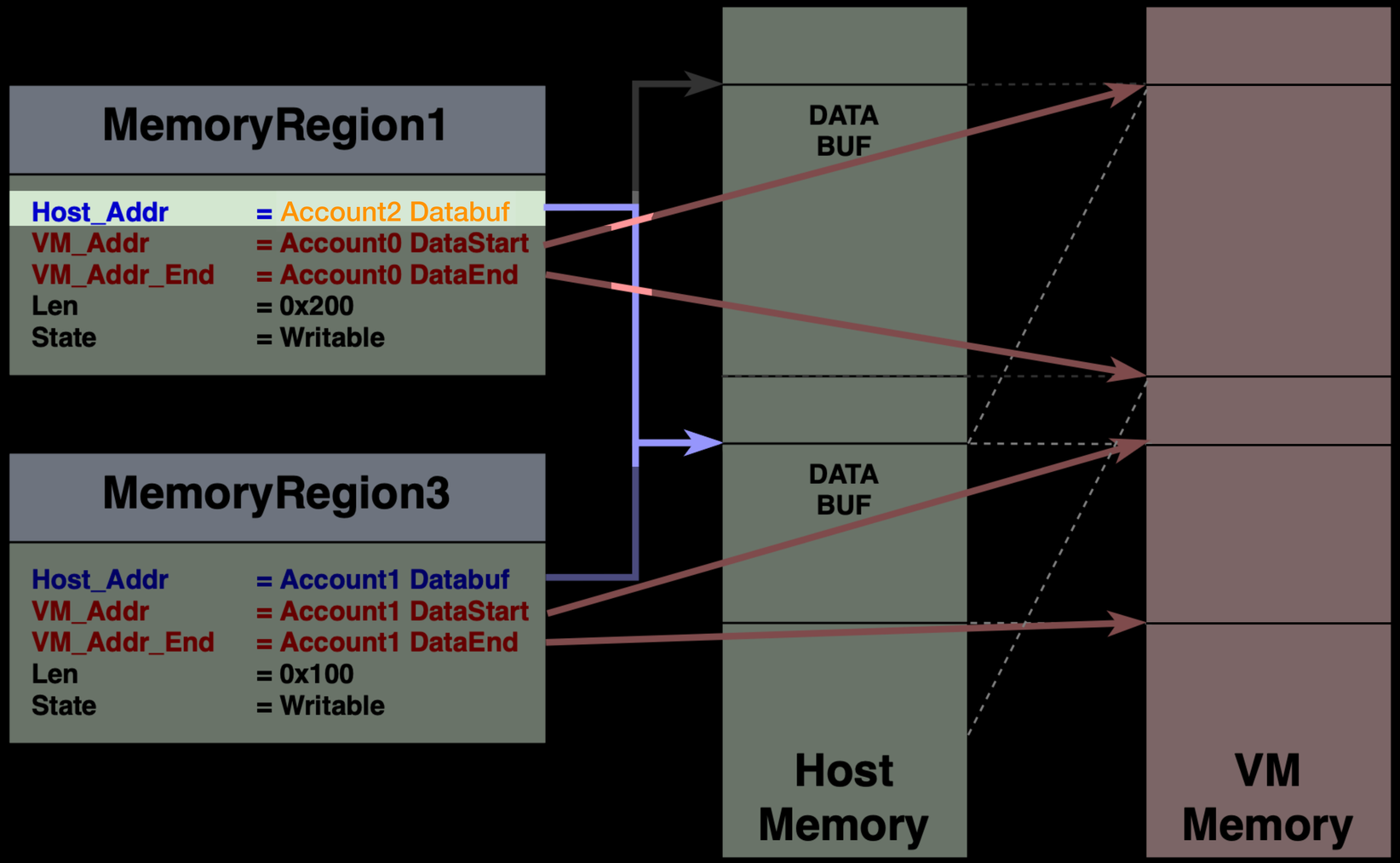
Bug

Out of bound write



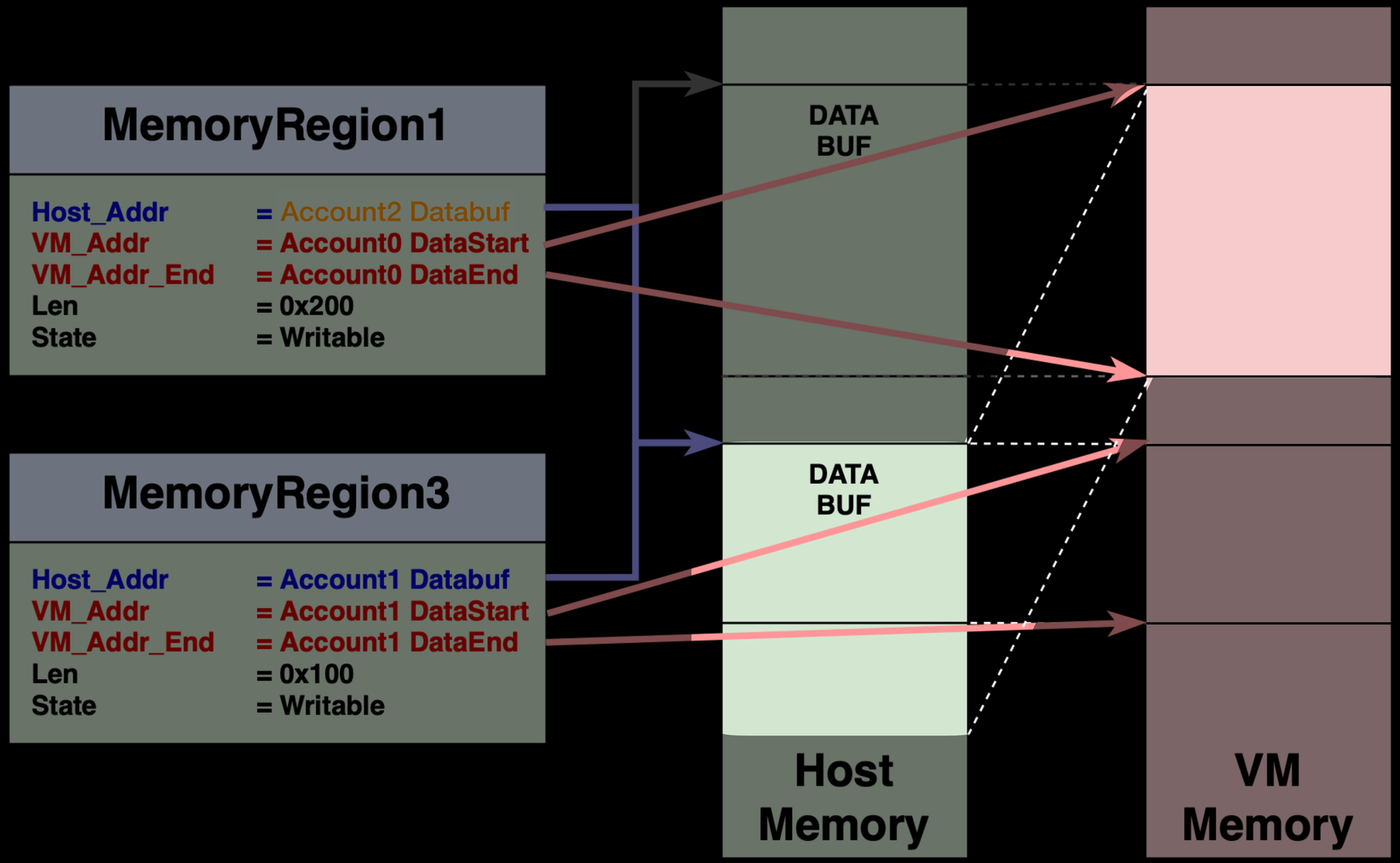
Bug

Out of bound write



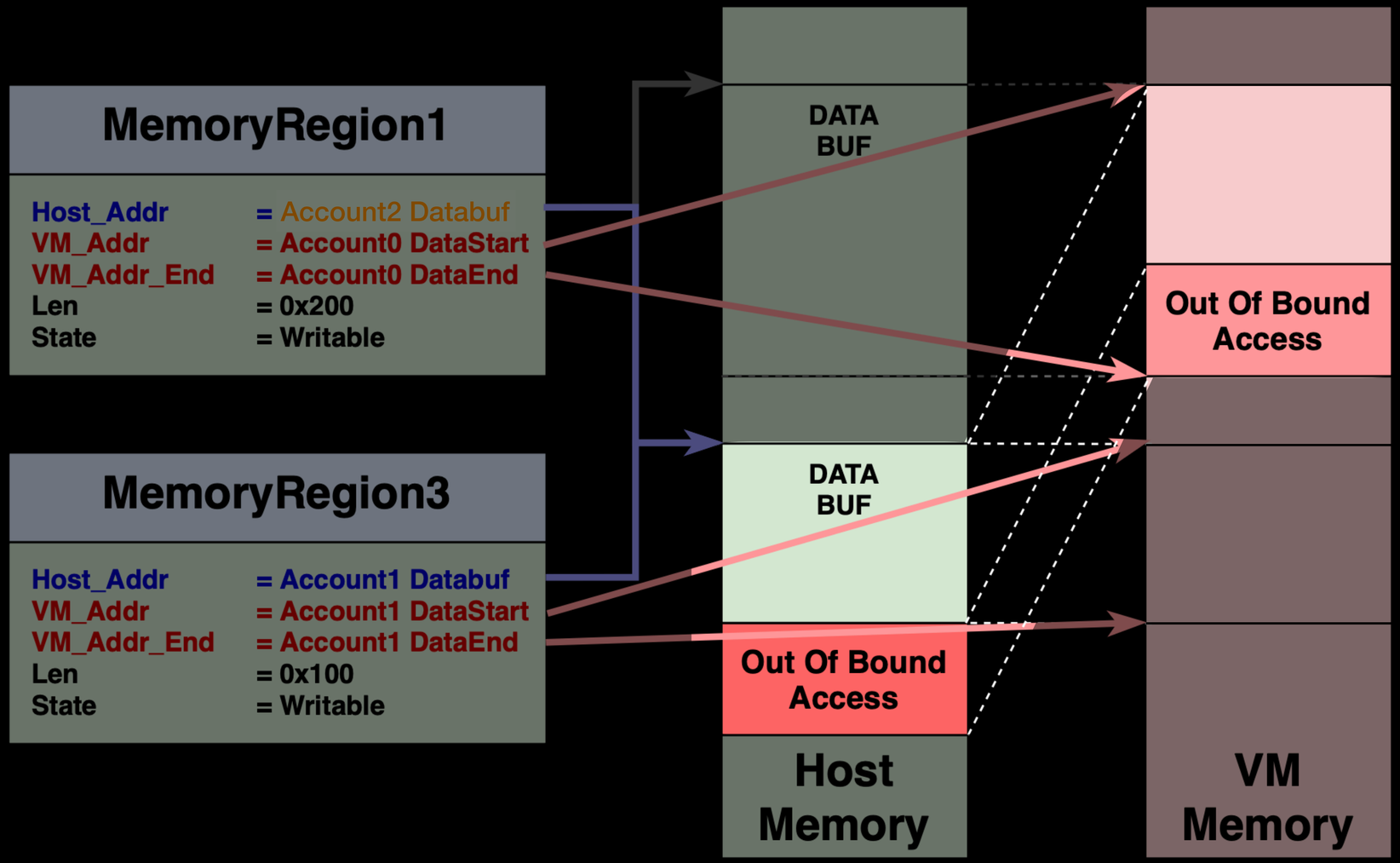
Bug

Out of bound write



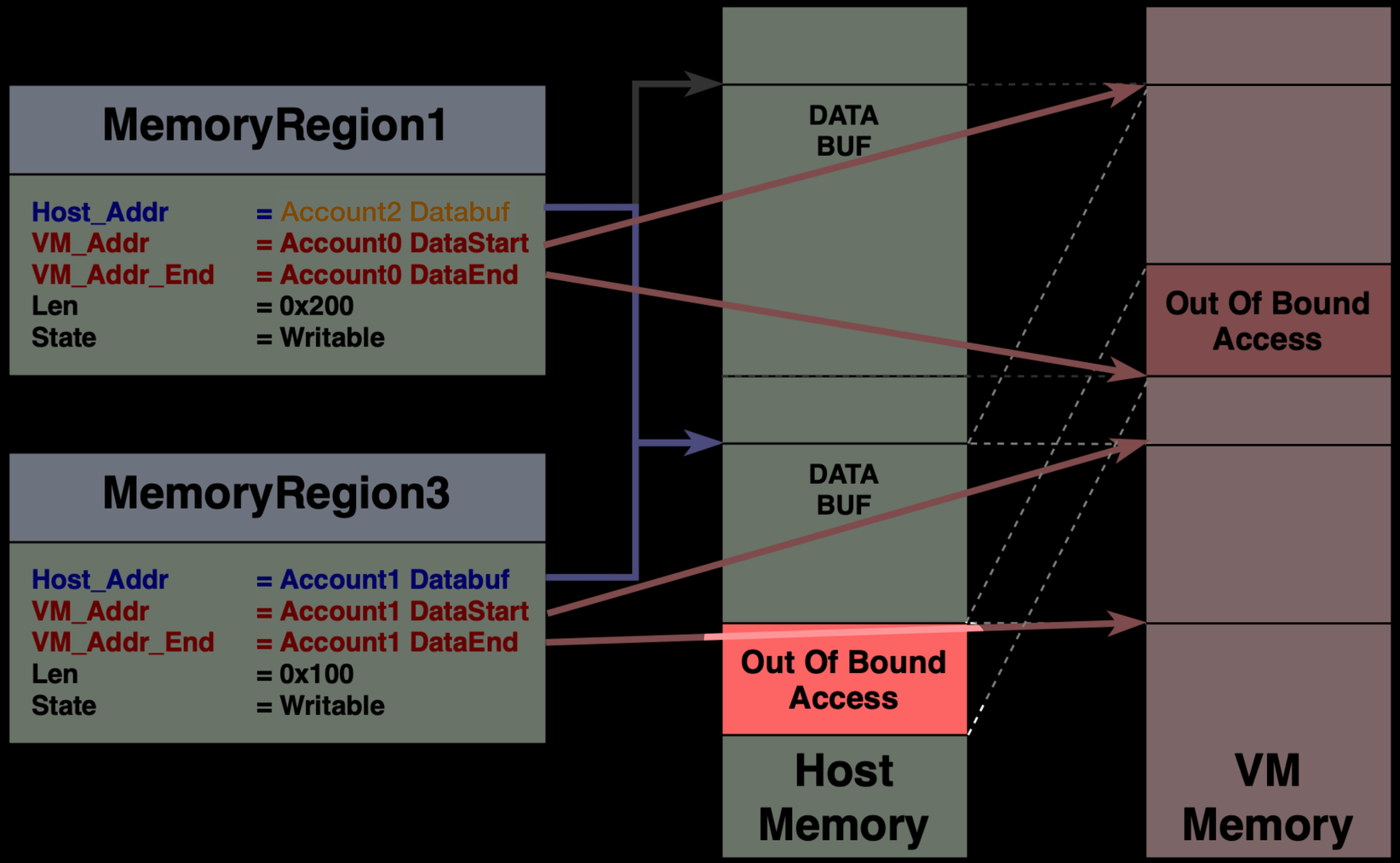
Bug

Out of bound write



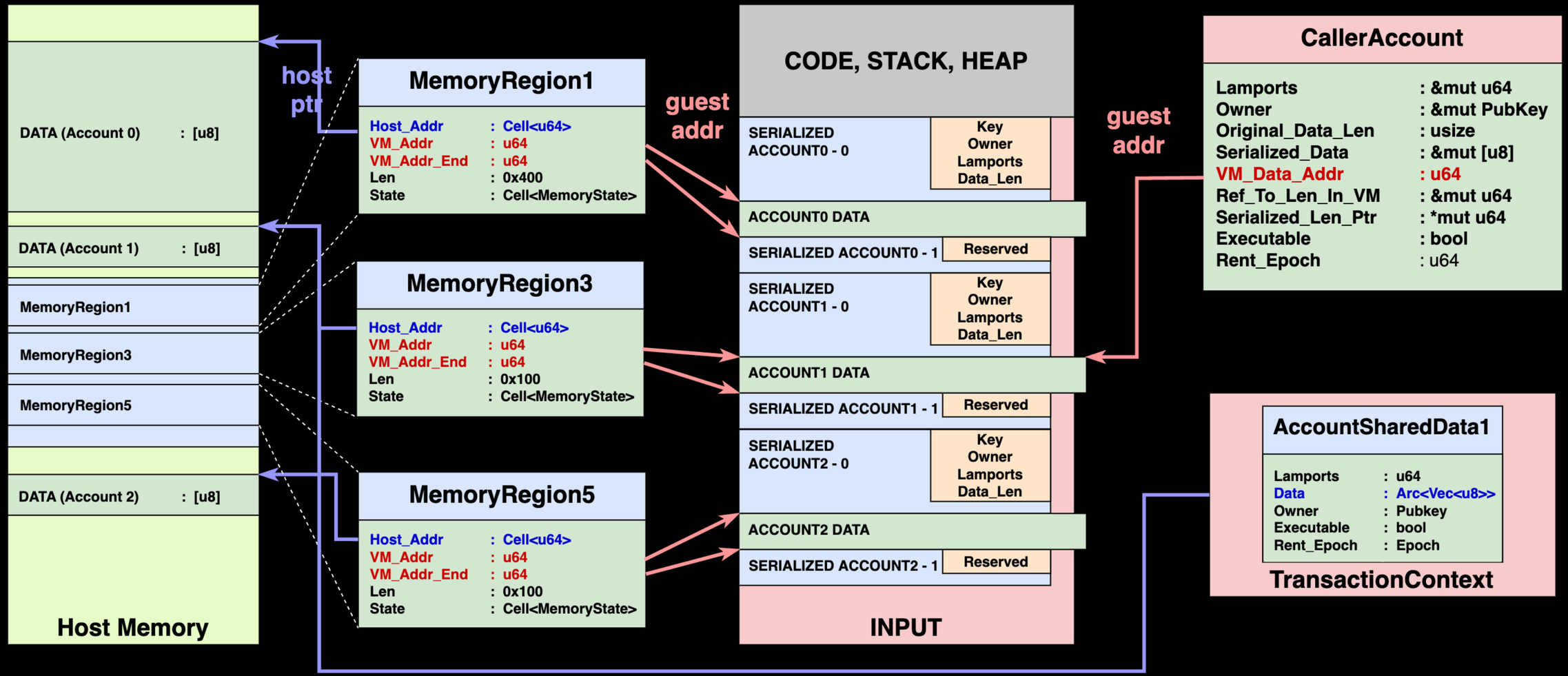
Bug

Out of bound write

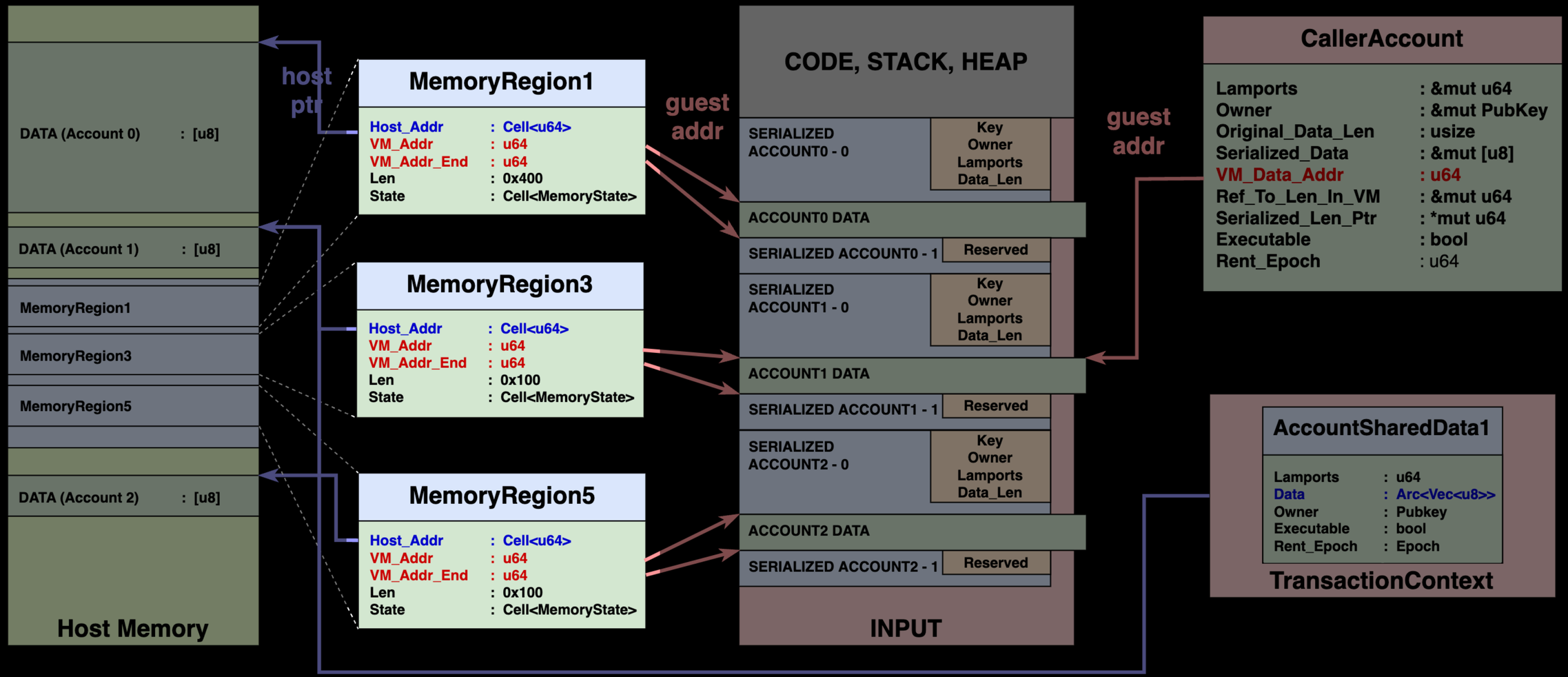


Exploit

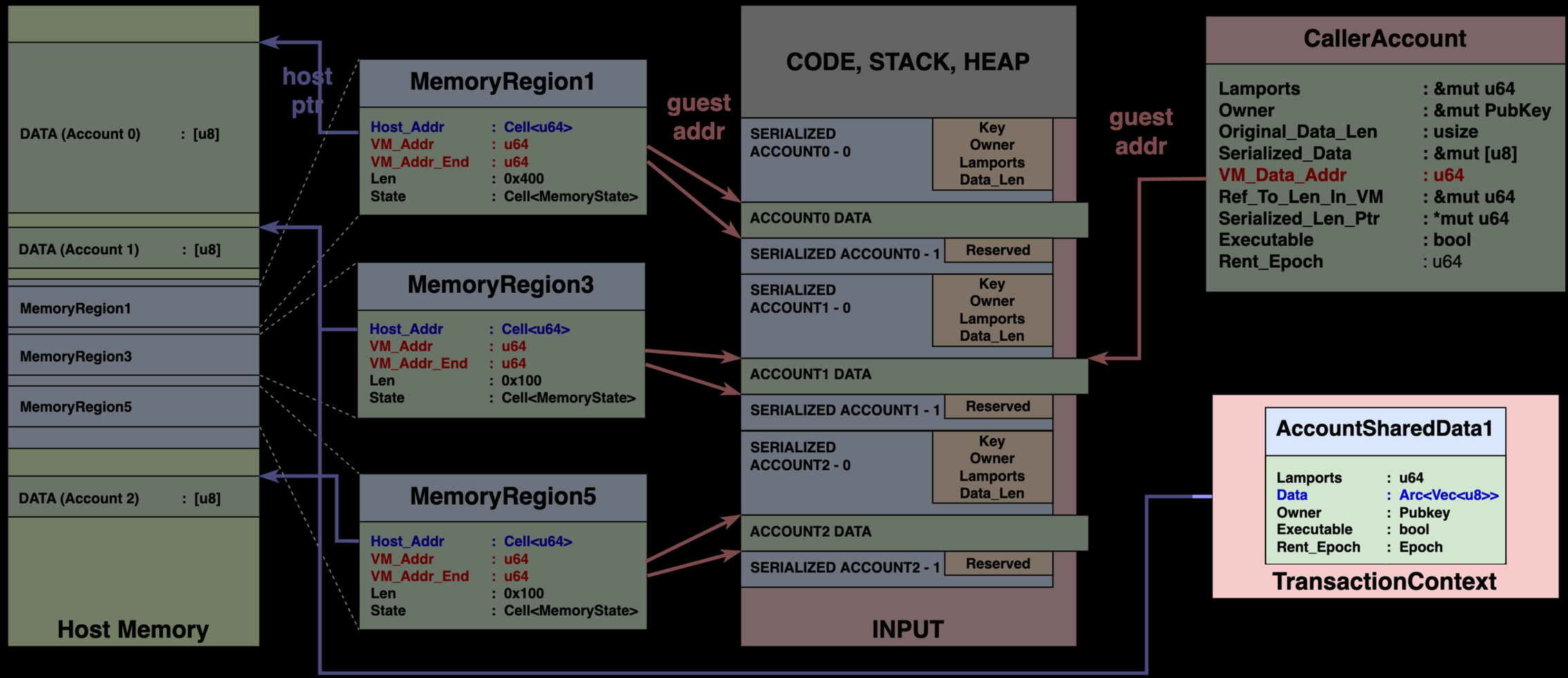
Exploit
Preparation



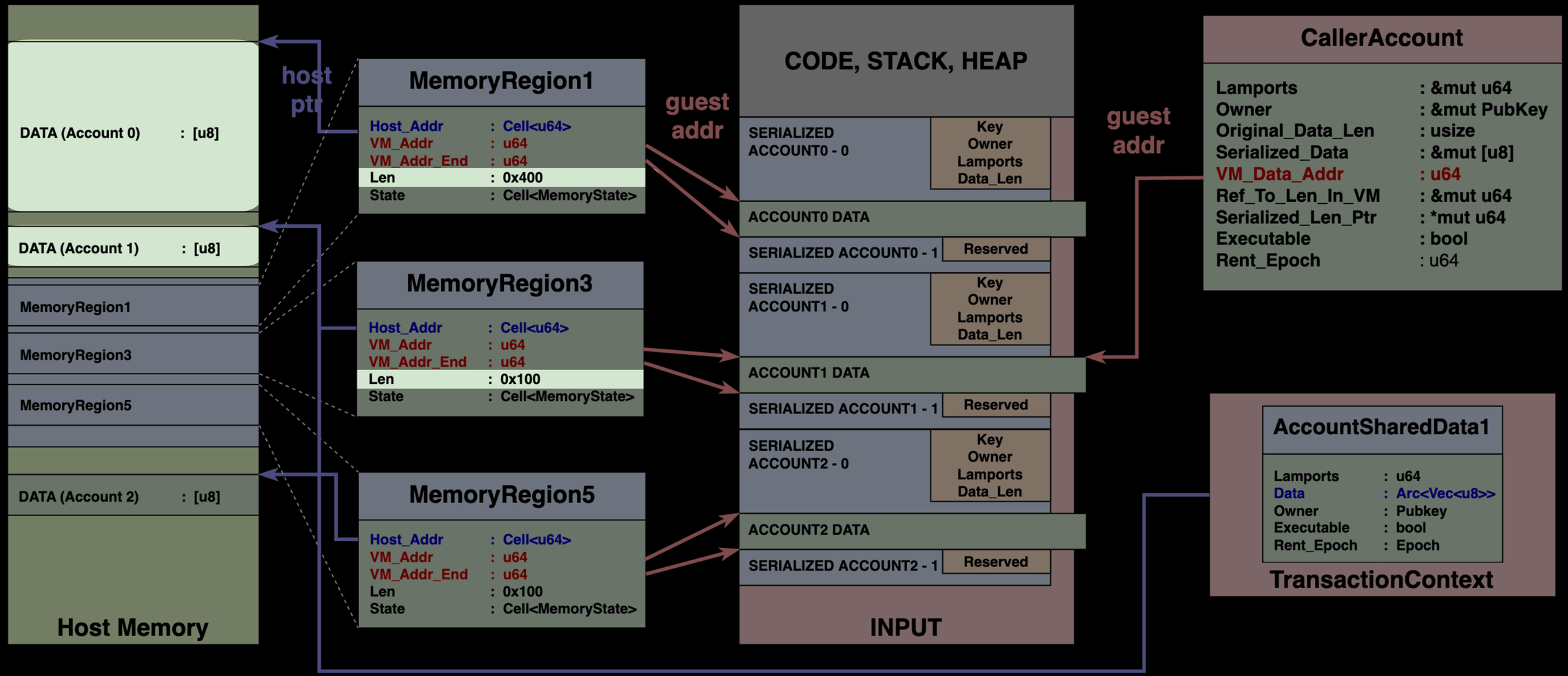
Exploit
Preparation



Exploit
Preparation

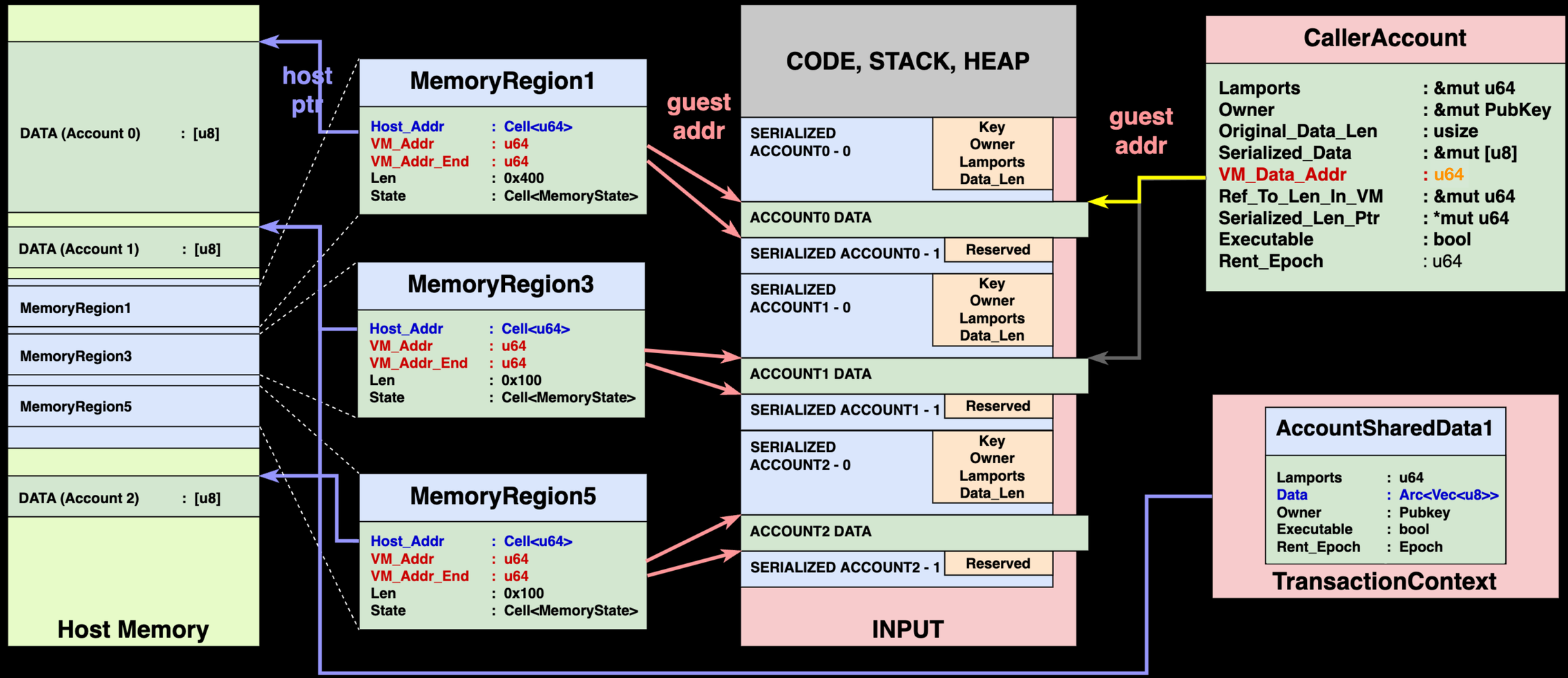


Exploit
Preparation



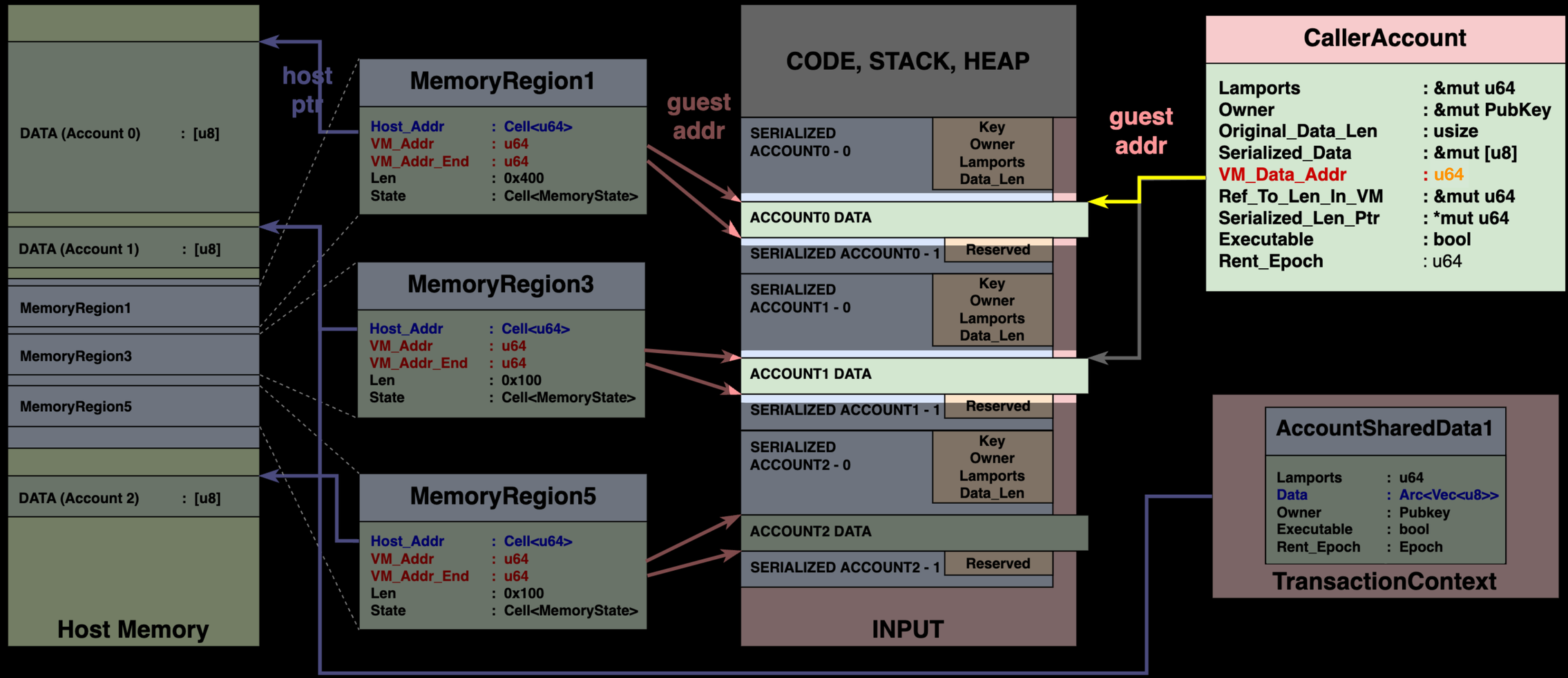
Exploit

Changing Data Pointer



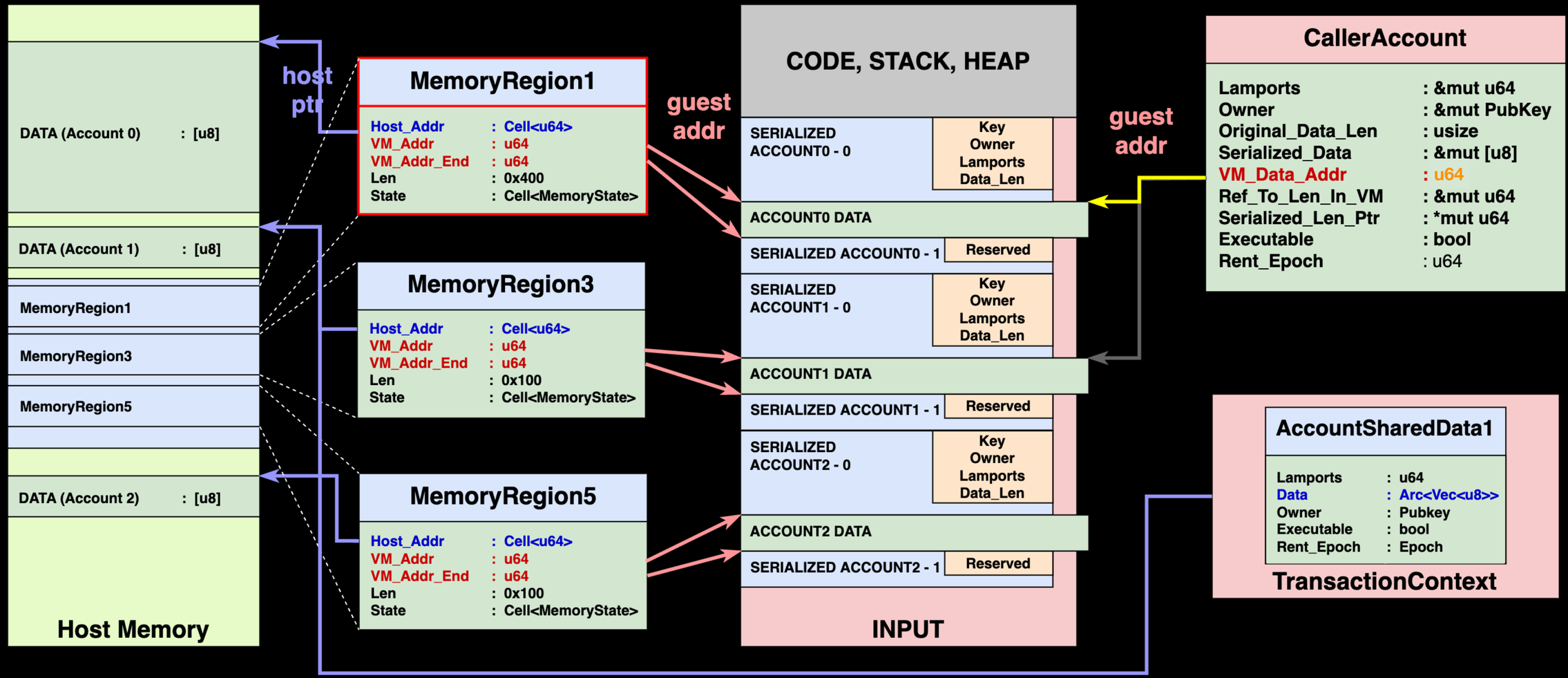
Exploit

Changing Data Pointer



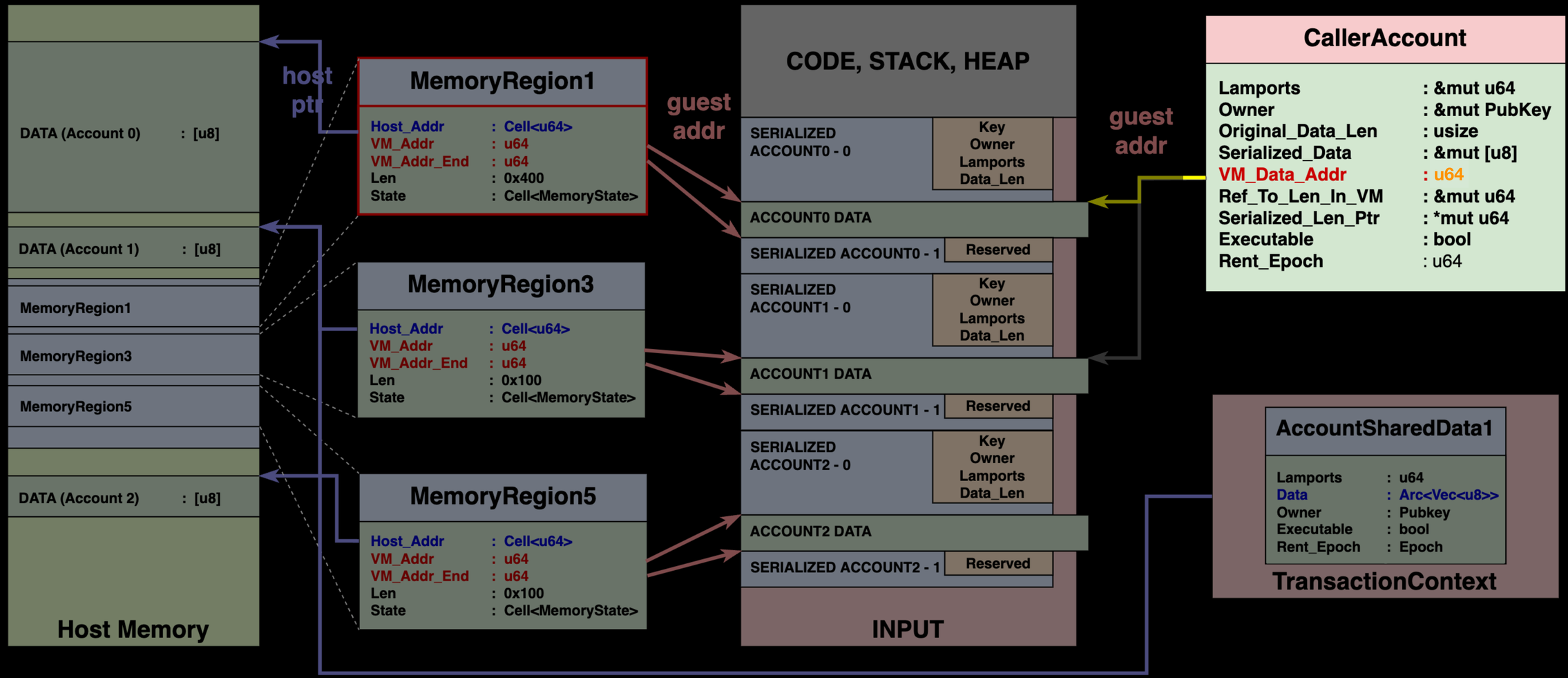
Exploit

Matching MemoryRegion



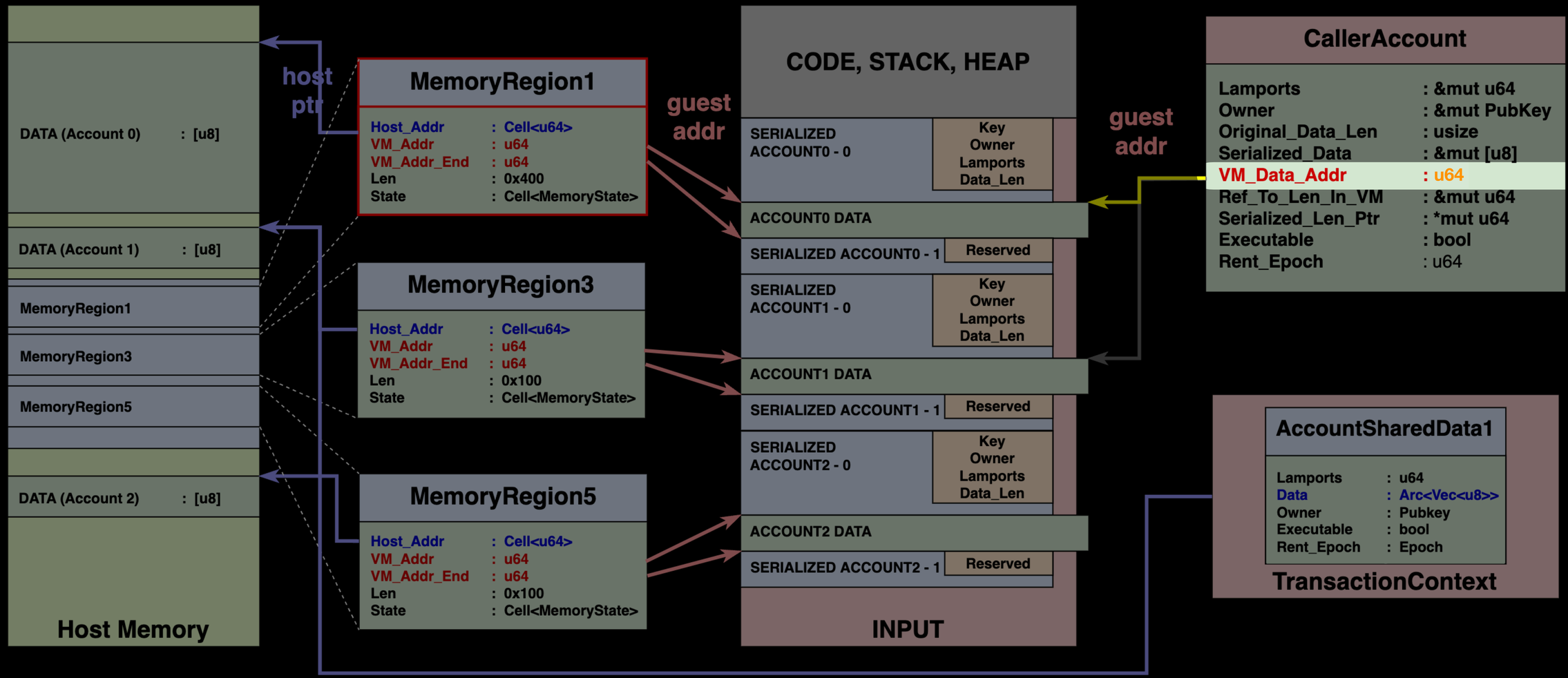
Exploit

Matching MemoryRegion



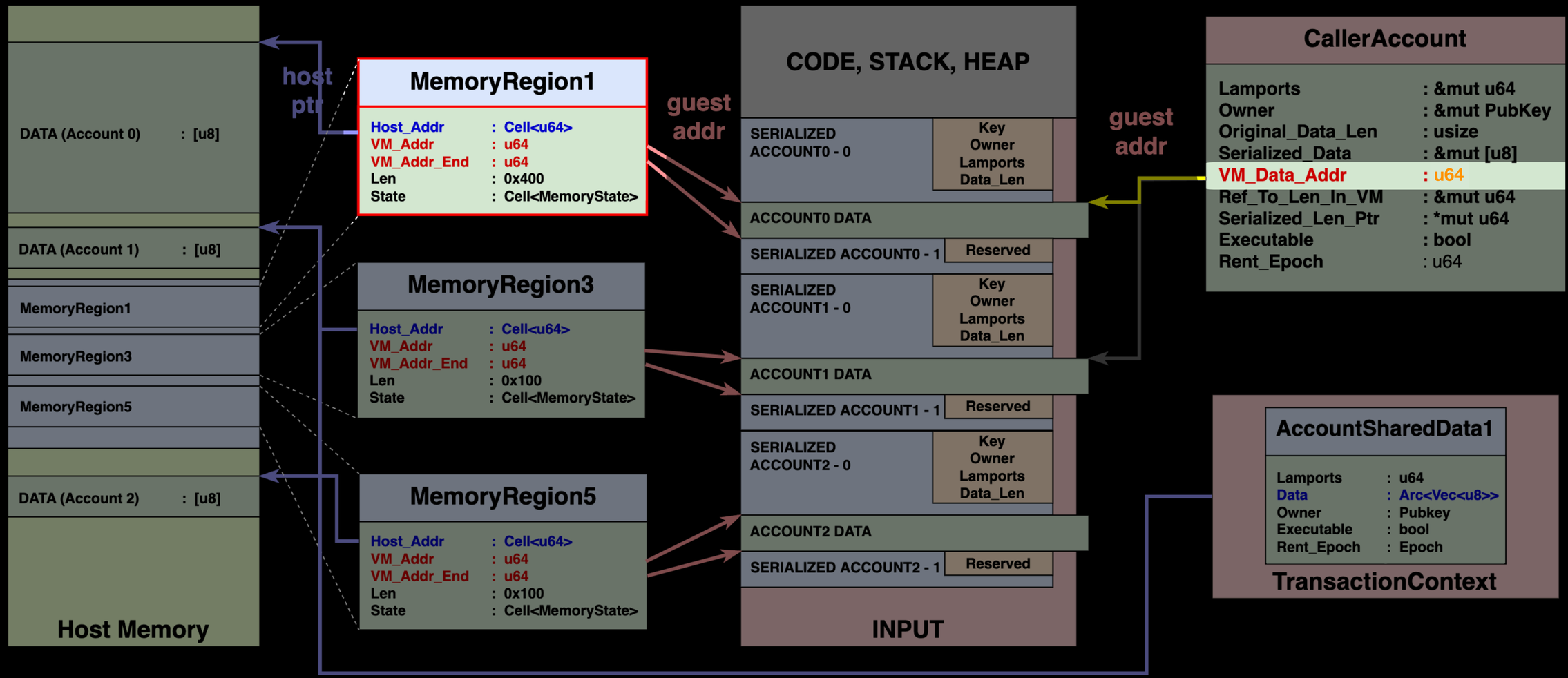
Exploit

Matching MemoryRegion



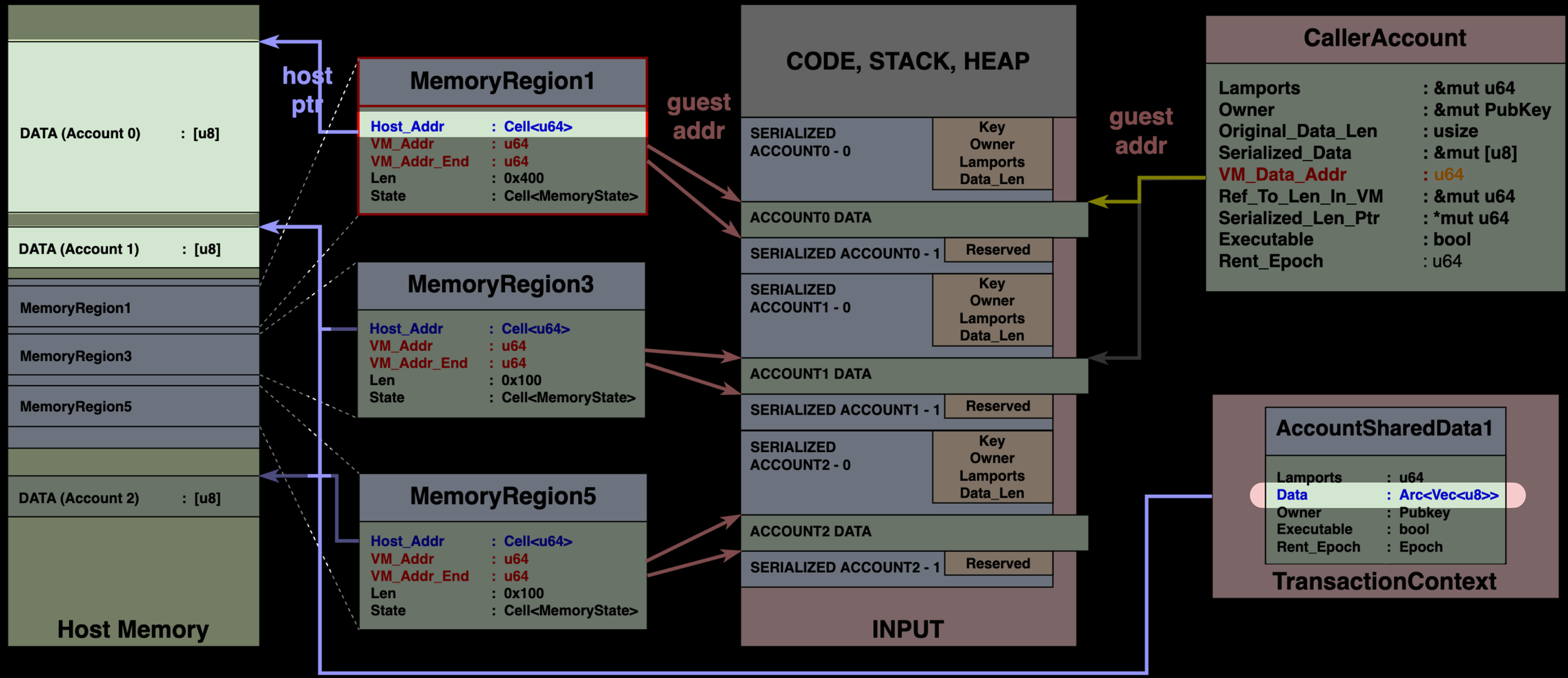
Exploit

Matching MemoryRegion



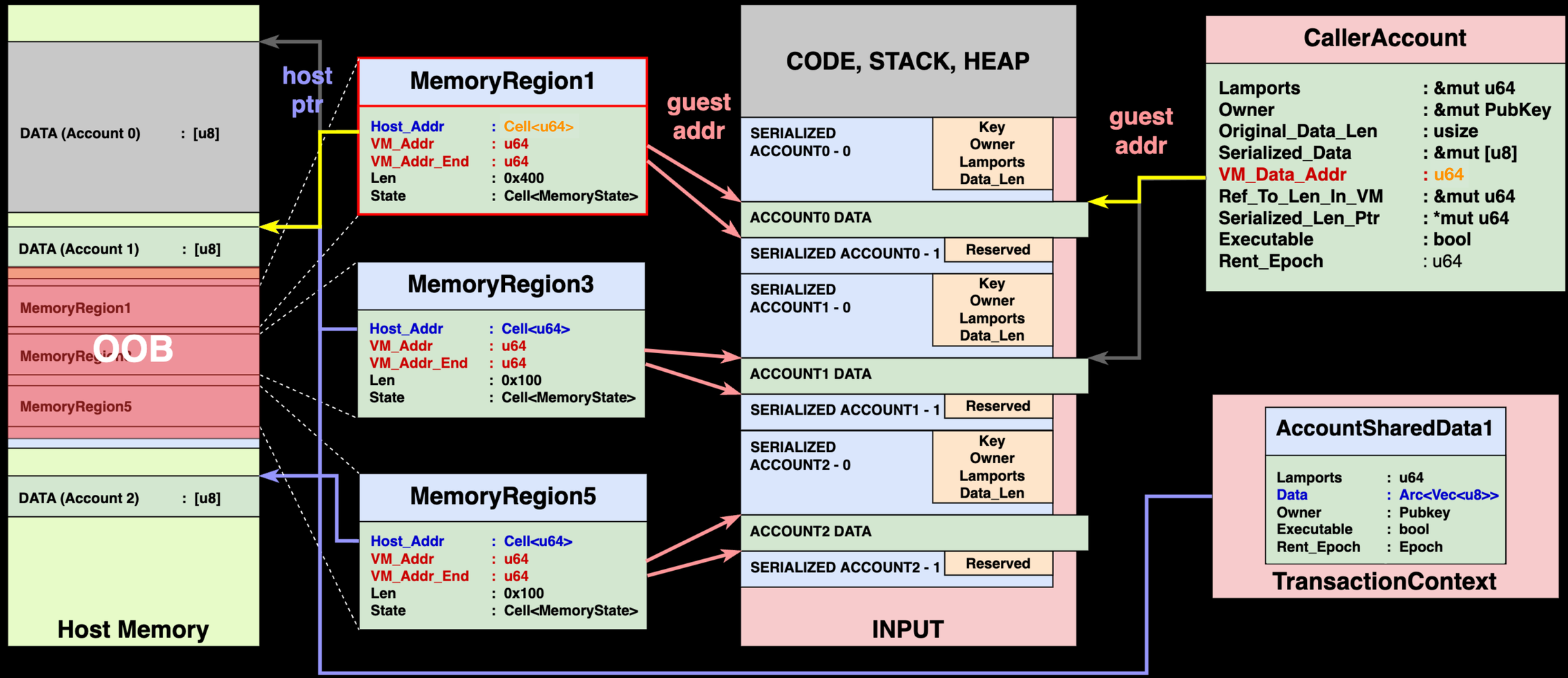
Exploit

Matching MemoryRegion



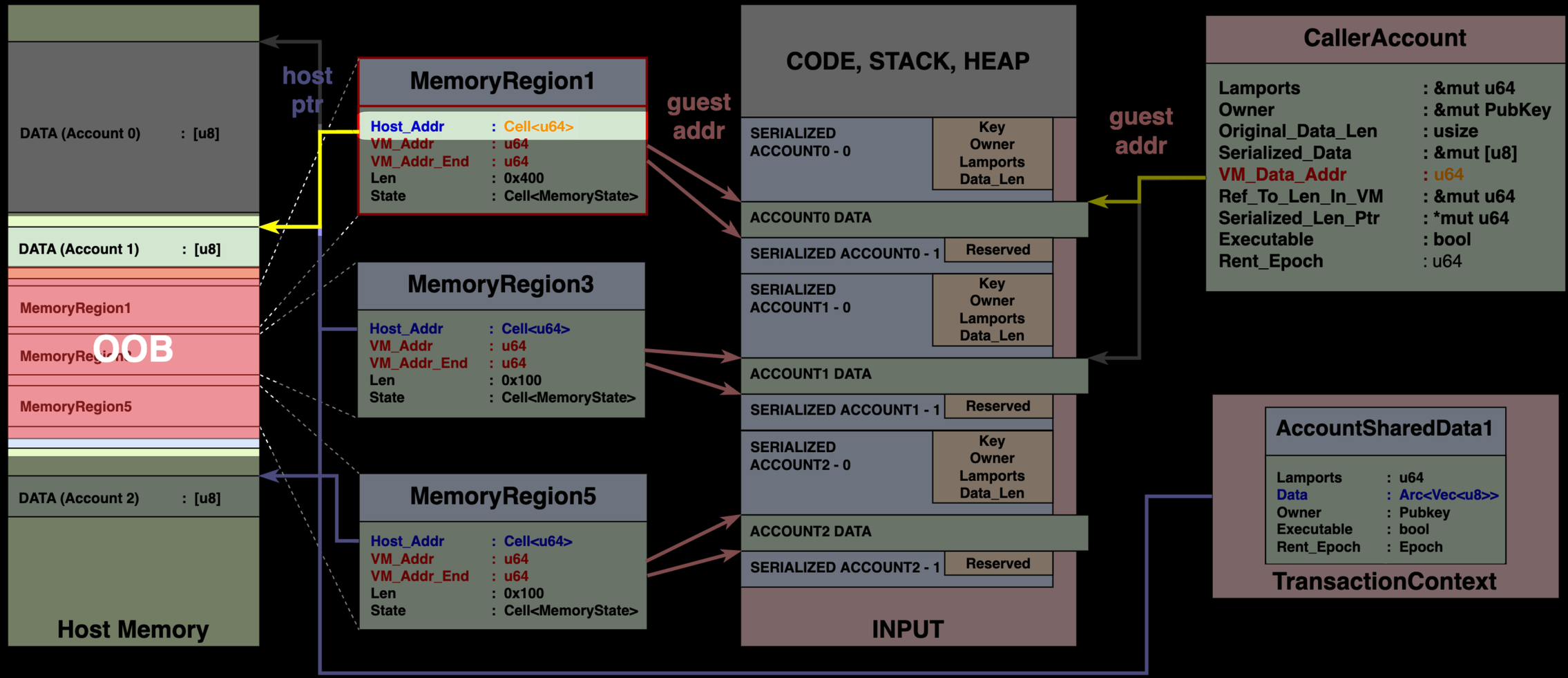
Exploit

Updating MemoryRegion.Host_Addr



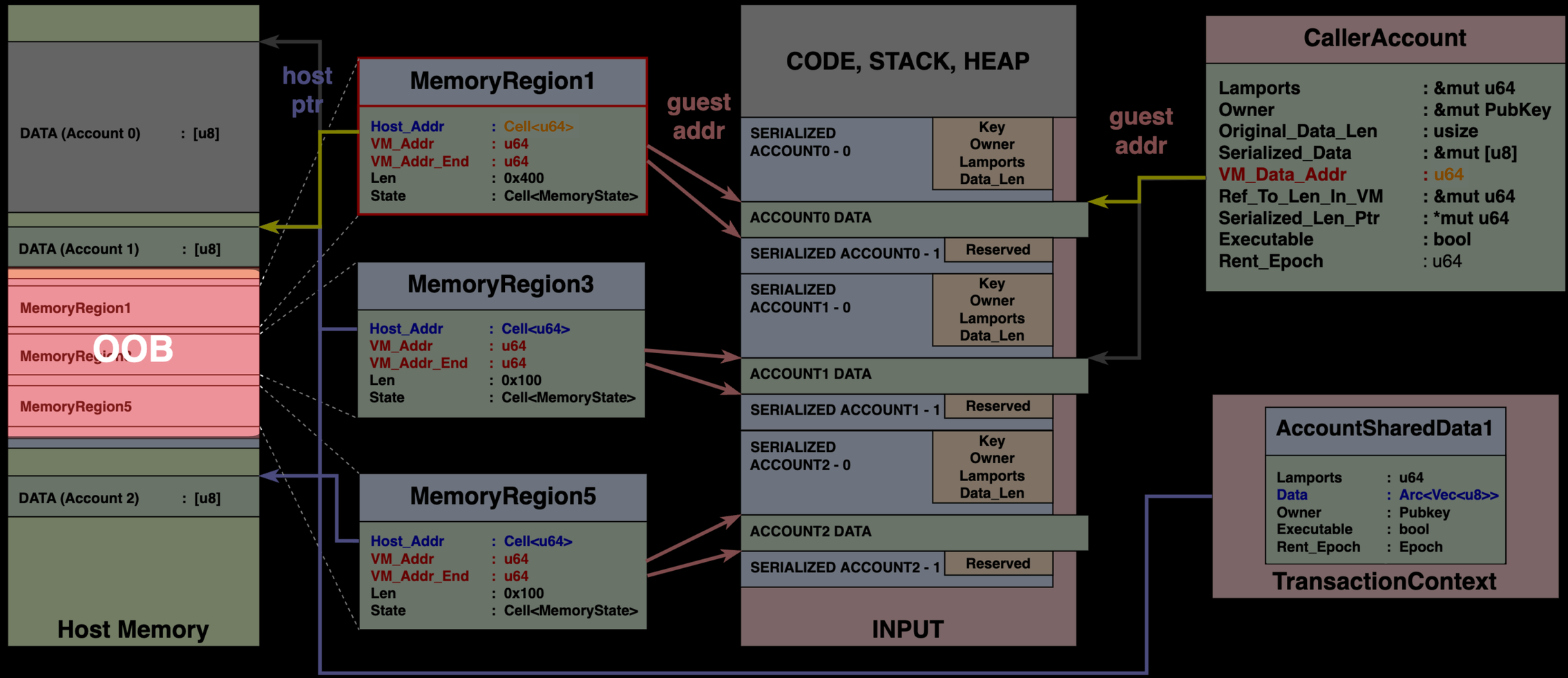
Exploit

Updating MemoryRegion.Host_Addr



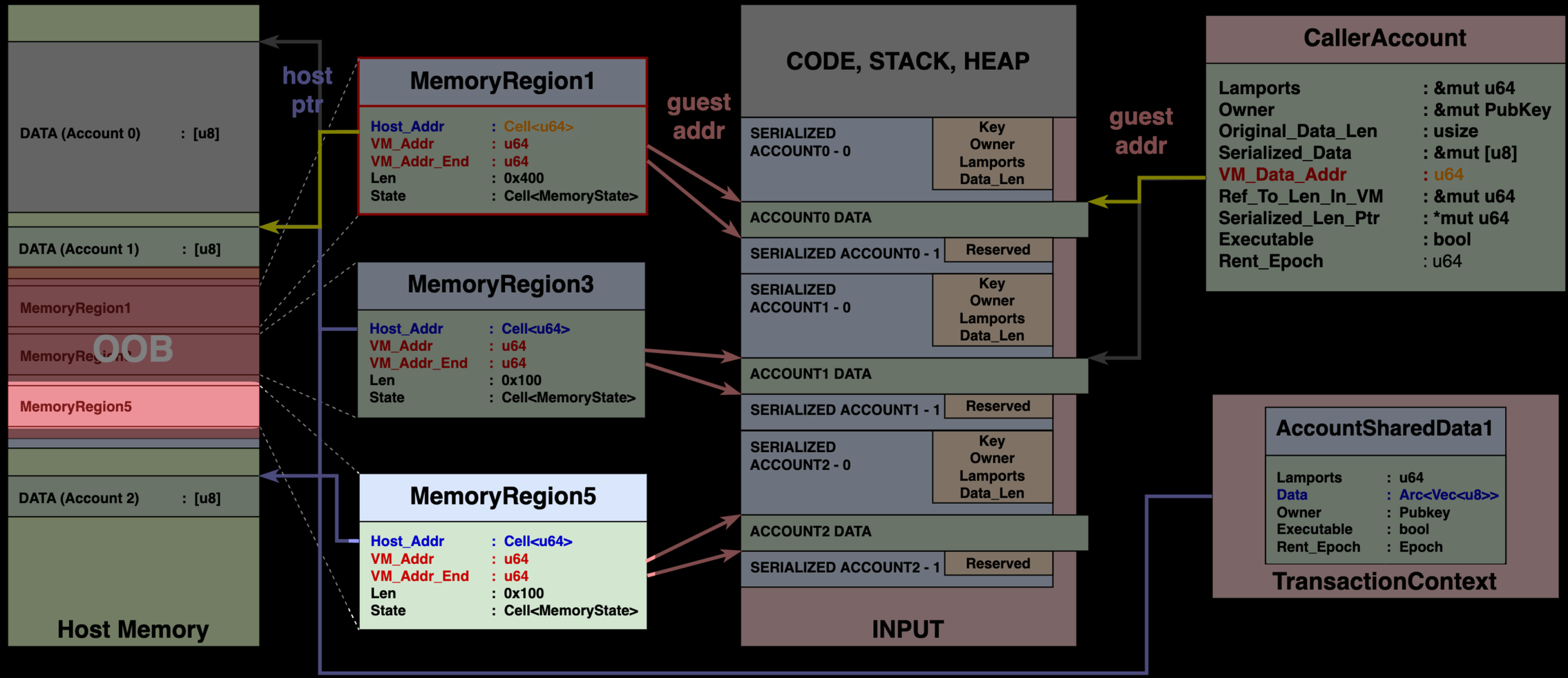
Exploit

Updating MemoryRegion.Host_Addr



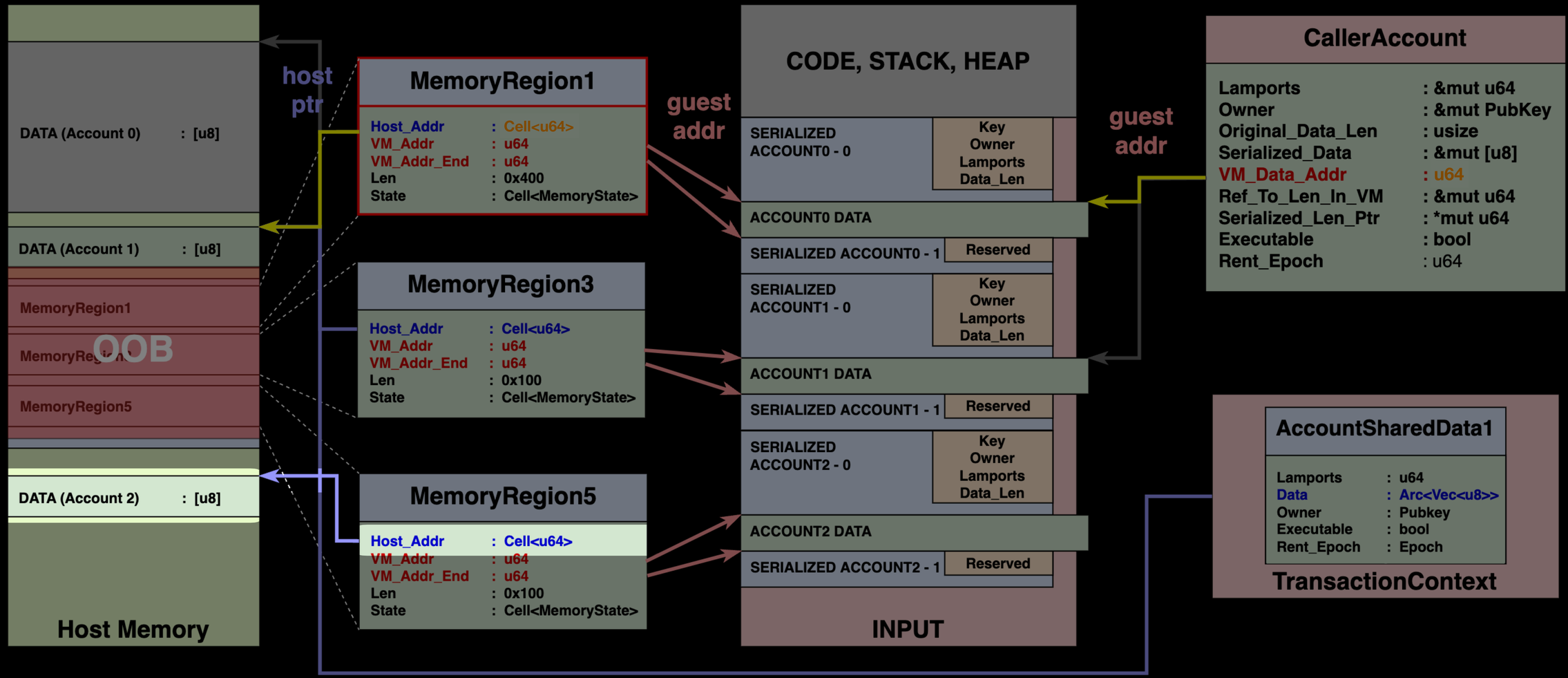
Exploit

Updating MemoryRegion.Host_Addr



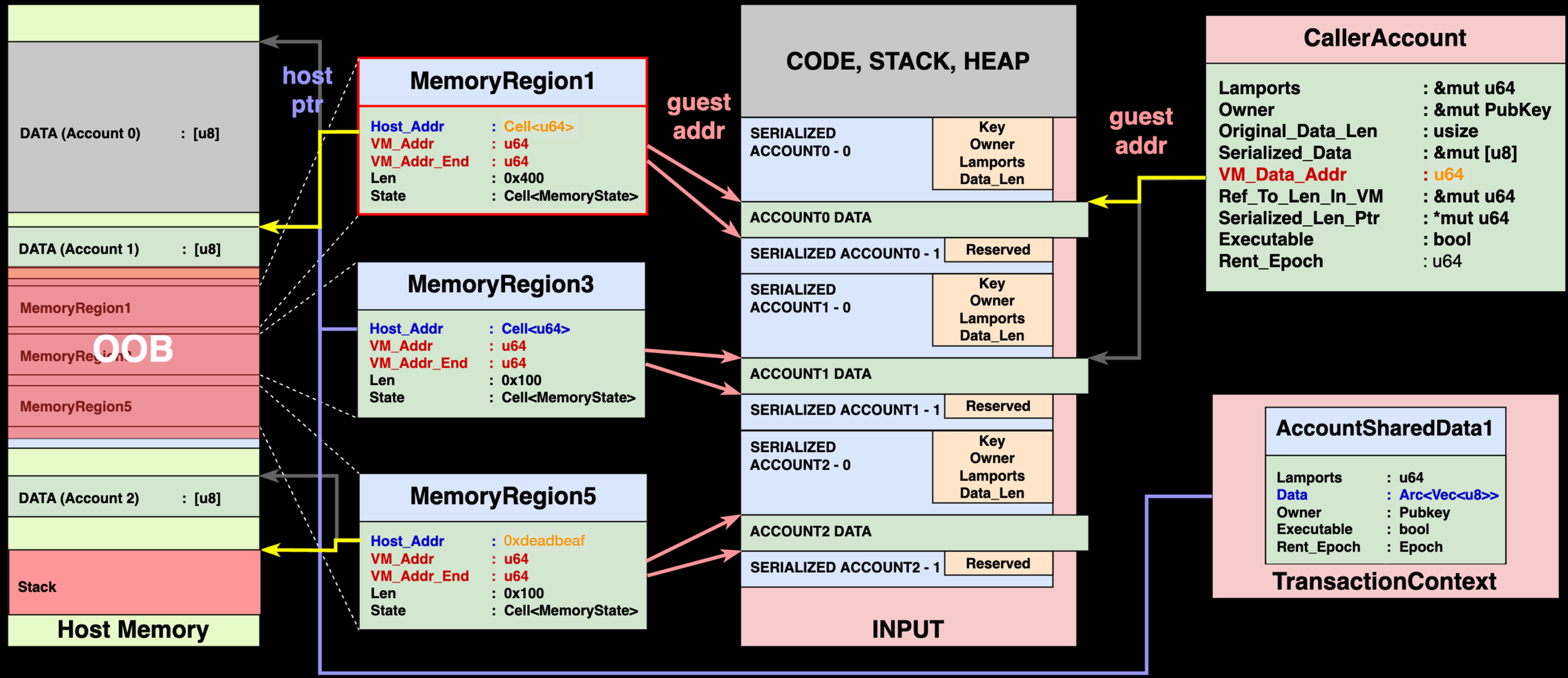
Exploit

Updating MemoryRegion.Host_Addr



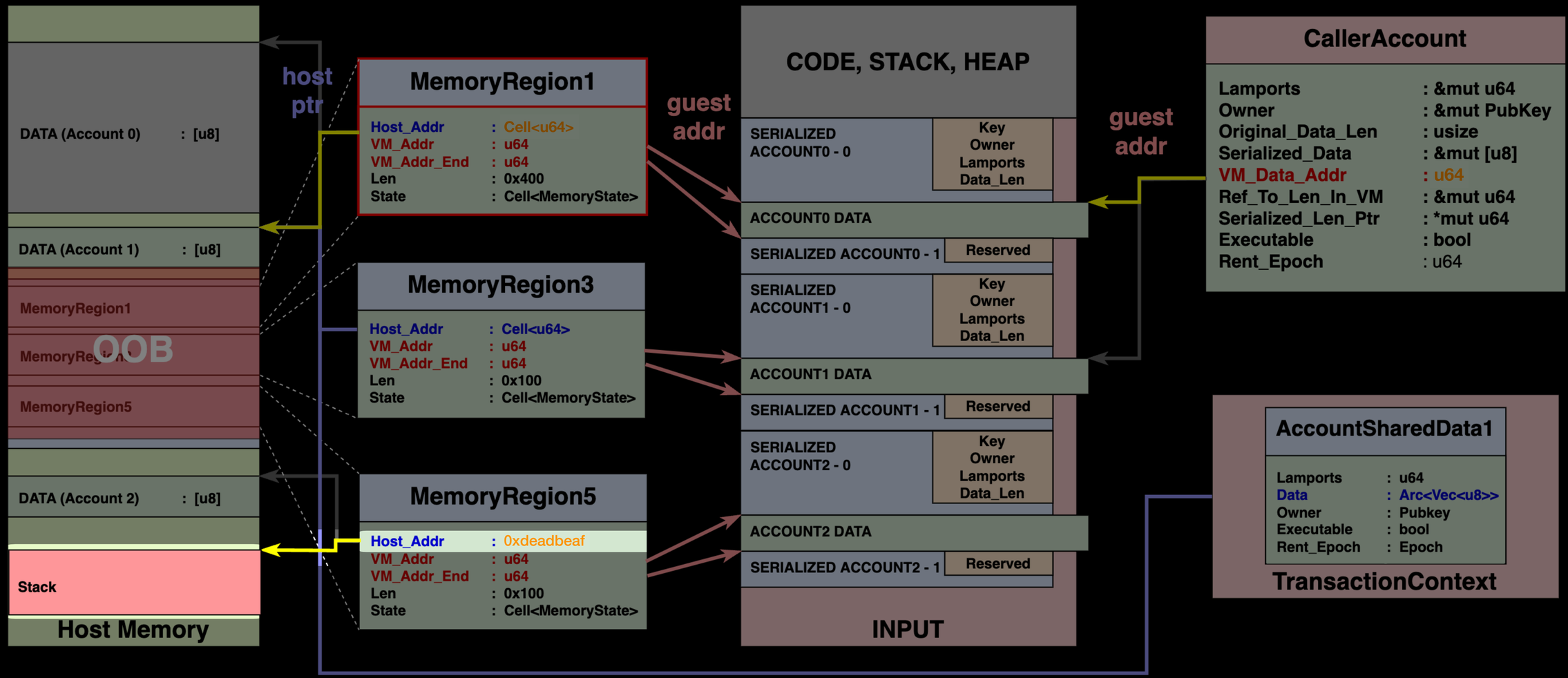
Exploit

Arbitrary Read/Write



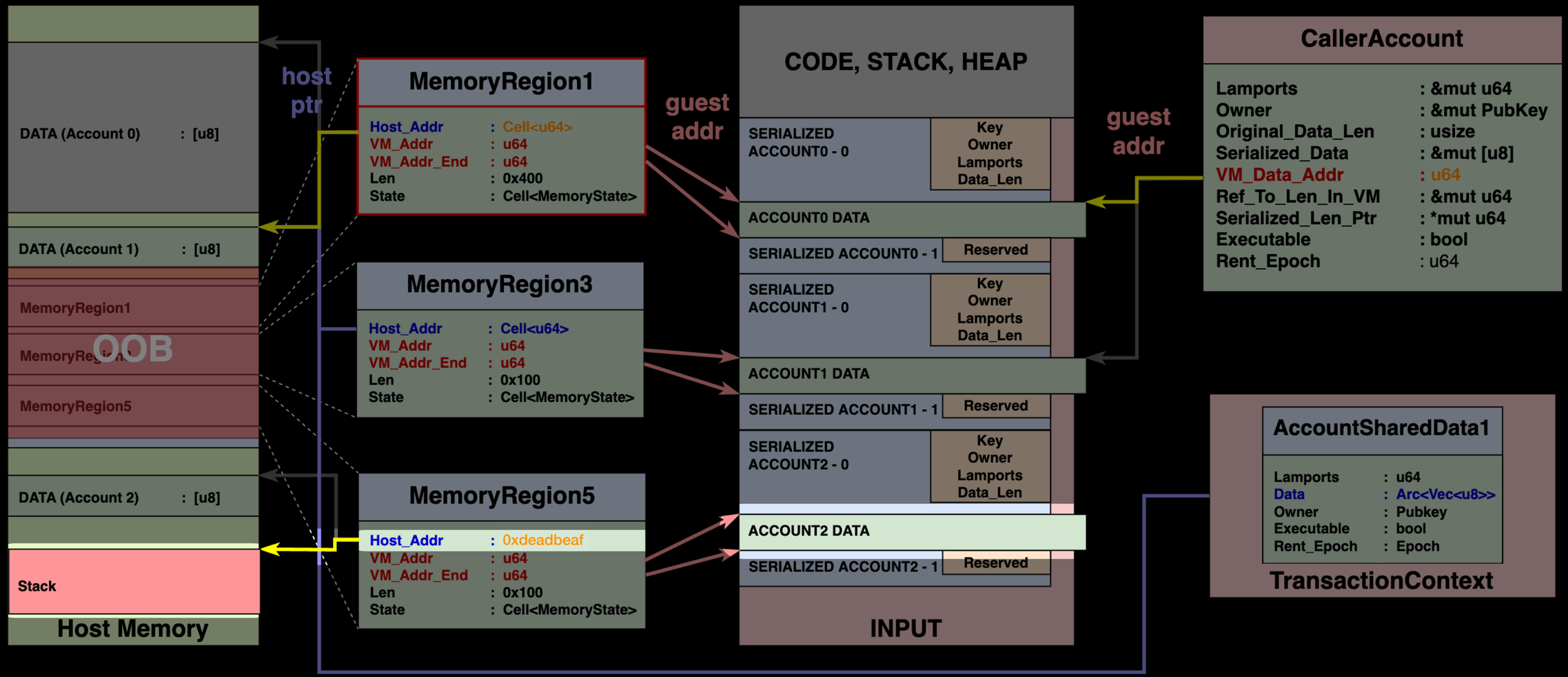
Exploit

Arbitrary Read/Write



Exploit

Arbitrary Read/Write



root@b2ec2262f745:~#

root@d705da096369:~#

root@d705da096369:~#

(0) ginoah 1:docker* 0.08 0.27 0.51 01:54`

Conclusion

Conclusion

- Not Just Simple Bugs:
  - Complex and interesting vulnerabilities exist

Conclusion

- Not Just Simple Bugs:
  - Complex and interesting vulnerabilities exist
- Modern Language ≠ Memory Safety:
  - Rust can still have memory bugs

Conclusion

- Not Just Simple Bugs:
  - Complex and interesting vulnerabilities exist
- Modern Language ≠ Memory Safety:
  - Rust can still have memory bugs
- New Research Areas:
  - Consensus algorithms, zero knowledge proofs, etc.

Thanks for Listening