Entries

move_to
third_party/move/move-vm/runtime/src/interpreter.rs

```
let gv = Self::load_resource(resolver, data_store, gas_meter, addr, ty)?;
match gv.move_to(resource) {
```

load_resource
third_party/move/move-vm/runtime/src/interpreter.rs

```
match data_store.load_resource(
```

load_resource
third_party/move/move-vm/runtime/src/data_cache.rs

```
resource_resolver.get_resource_bytes_with_metadata_and_layout(
```

StorageAdapter

get_resource_bytes_with_metadata_and_layout
aptos-move/aptos-vm/src/data_cache.rs

```
self.get_any_resource_with_layout(address, struct_tag, metadata, maybe_layout)
```

get_any_resource_with_layout
aptos-move/aptos-vm/src/data_cache.rs

```
if let Some(resource_group) = resource_group {
    self.resource_group_view.get_resource_from_group(&key, struct_tag, maybe_layout)?;
} else {
    self.executor_view.get_resource_bytes(&state_key, maybe_layout)?;
```

LatestView

get_resource_bytes
aptos-move/block-executor/src/data_cache.rs

```
let maybe_state_value = self.get_resource_state_value(state_key, maybe_layout)?;
```

get_resource_state_value
aptos-move/block-executo/src/view

```
self.get_resource_state_value_impl(
```

get_resource_state_value_impl
aptos-move/block-executo/src/view

```
let state = self.latest_view.get_resource_state();
let mut ret = state.read_cached_data_by_kind(
if matches!(ret, ReadResult::Uninitialized) {
    let from_storage =
        TransactionWrite::from_state_value(self.get_raw_base_value(state_key)?);
```

get_raw_base_value
aptos-move/block-executo/src/view

```
let ret = self.base_view.get_state_value(state_key).map_err(|e| {
```

CachedStateView/database

get_state_value
storage-interface/src/state_store/state_view/cacehed_state_view.rs

```
if let Some(value_with_version_opt) = self.memorized.get_cloned(state_key) {
let value_with_version_opt = self.get_unmemorized(state_key)?;
Ok(self.memorized.try_insert(
```

get_unmemorized
storage-interface/src/state_store/state_view/cacehed_state_view.rs

```
let ret = if let Some(update) = self.speculative.get_state_update(state_key) {
        MemorizedStateRead::from_speculative_state(update)
} else if let Some(update) = self.hot.get_state_update(state_key)? {
        MemorizedStateRead::from_hot_state_hit(update)
} else if let Some(base_version) = self.base_version() {
        self.cold.get_state_value_with_version_by_version(state_key, base_version)?;
```

get_state_update
storage-interface/src/state_store/state_delta.rs

```
self.shards[state_key.get_shard_id() as usize].get(state_key)
```

get_state_update
storage/aptosdb/src/state_store/hot_state.rs

```
Ok(self.get(state_key))
```

get_state_value_with_version_by_version
storage/aptosdb/src/db/include/aptosdb_reader.rs

```
self.state_store.get_state_value_with_version_by_version(state_key, version)
```

get_state_value_with_version_by_version
storage/aptosdb/src/state_store/mod.rs

```
self.state_kv_db.get_state_value_with_version_by_version(state_key, version)
```

get_state_value_with_version_by_version
storage/aptosdb/src/state_kv_db.rs

```
let mut iter = self
    .db_shard(state_key.get_shard_id())
    .iter_with_opts::<StateValueSchema>(read_opts)?;
iter.seek(&(state_key.clone(), version))?;
iter.next().transpose()?
```

TransactionDataCache(move-vm) => StorageAdapter(aptos-move) => LastestView => CachedStateView => Aptos-db => RockDB