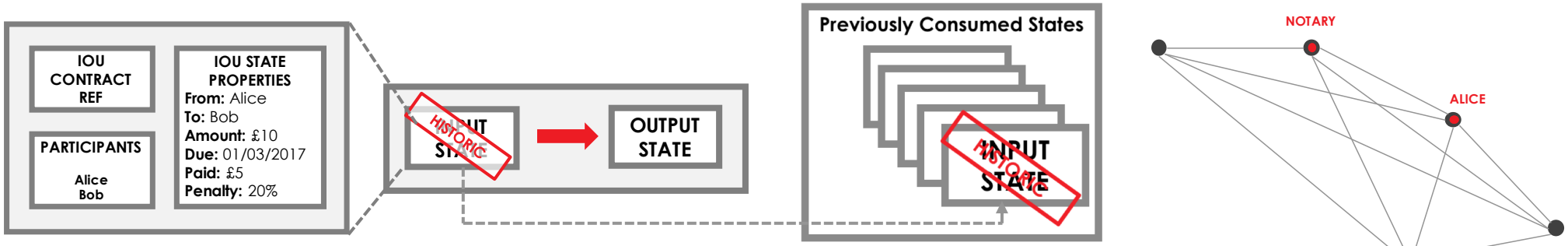
Corda Concepts

Concept to Code

Concepts to Code

Corda: Key Concepts



State Object

States are immutable objects that represent (shared) facts such as a financial agreement or contract at a specific point in time

Transaction

Transactions consume input states and create output states.

The newly created output states replace the input states which are marked as historic.

Consensus

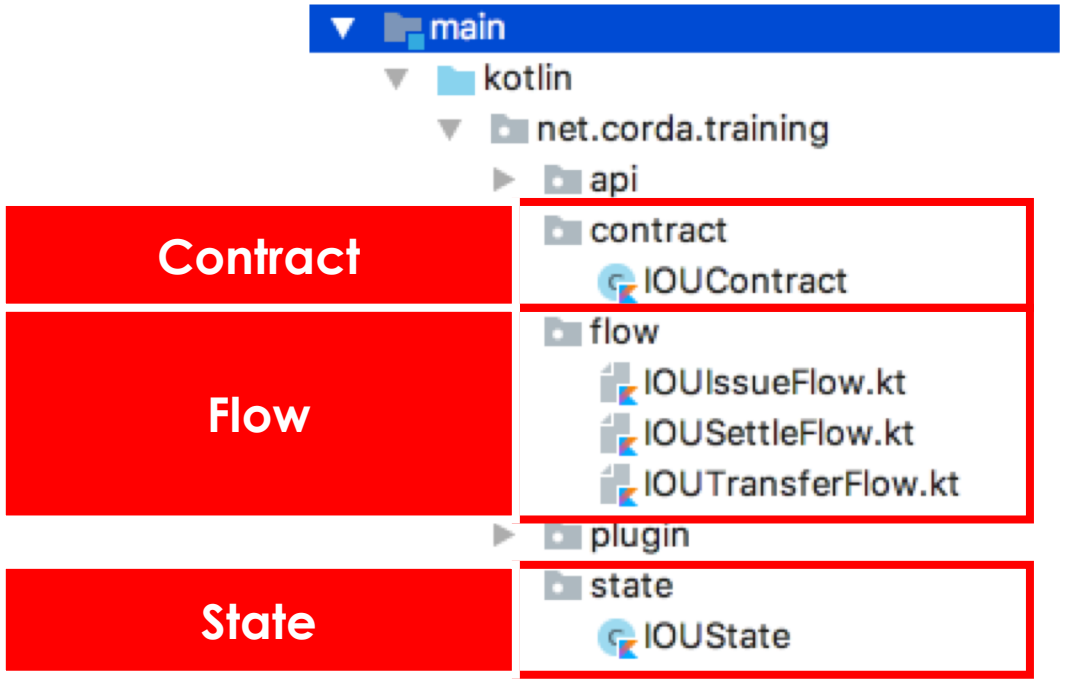
Parties reach consensus on the evolution of a shared fact. This is done by testing the validity (by way of contract code) and uniqueness (by way of the notary) of the transaction.

Flows

Flows are light-weight processes used to coordinate interactions required for peers to reach consensus about shared facts.

Concepts to Code

CODE



Concept to Code

State

Asset/Object Attributes

```
data class IOUState(val amount: Amount<Currency>,
                    val lender: Party,
                    val borrower: Party,
                    val paid: Amount<Currency> = Amount( tokenQuantity: 0, amount.token),
                    override val linearId: UniqueIdentifier = UniqueIdentifier()): LinearState {
```

Participants

```
    override val participants get() = listOf(lender, borrower)
```

Party

Parties on the network are represented using the `AbstractParty` class. There are two types of `AbstractParty`:

- `Party`, identified by a `PublicKey` and a `CordaX500Name`
- `AnonymousParty`, identified by a `PublicKey` only

Concept to Code

Contract

deterministic **Command** logic

```
 * The contract code for the [IOUContract].
 * The constraints are self documenting so don't require any additional explanation.
 */
override fun verify(tx: LedgerTransaction) {
    val command = tx.commands.requireSingleCommand<IOUContract.Commands>()
    when (command.value) {
        is Commands.Issue -> requireThat {
            "No inputs should be consumed when issuing an IOU." using (tx.inputs.isEmpty())
            "Only one output state should be created when issuing an IOU." using (tx.outputs.size == 1)
            val iou = tx.outputStates.single() as IOUState
            "A newly issued IOU must have a positive amount." using (iou.amount > Amount( tokenQuantity: 0, iou.amount.token))
            "The lender and borrower cannot have the same identity." using (iou.borrower != iou.lender)
            "Both lender and borrower together only may sign IOU issue transaction." using
                    (command.signers.toSet() == iou.participants.map { it.owningKey }.toSet())
        }
```

r3.

Concept to Code

Flow

Build Transaction

```
val notary = serviceHub.networkMapCache.notaryIdentities.first()
val issueCommand = Command(IOUContract.Commands.Issue(), state.participants.map { it.owningKey })
val builder = TransactionBuilder(notary = notary)
builder.withItems(StateAndContract(state, IOUContract.IOU_CONTRACT_ID), issueCommand)
```

Verify & Sign

```
builder.verify(serviceHub)
val ptx = serviceHub.signInitialTransaction(builder)
```

Gather Signatures

```
val sessions = (state.participants - ourIdentity).map { initiateFlow(it) }.toSet()
val stx = subFlow(CollectSignaturesFlow(ptx, sessions))
```

Finalize Transaction

```
return subFlow(FinalityFlow(stx))
```

r3.

Flow Steps - Visual