UNIVERSITY OF PADUA

Analysis of the implementation of the Stipula legal calculus using Distributed Ledger Technologies

Master's degree thesis

Supervisor

Professor Silvia Crafa

Graduand

Federico Zanardo

DIPARTIMENTO MATEMATICA

UNIVERSITY OF PADUA

Research question

- *What is a platform for running Stipula contracts?*
- **1. Architectural analysis**
  - Proposal of a distributed platform scalable from a server to a peer-to-peer network
  - Asset definition and UTXO model

UNIVERSITY OF PADUA

Research question

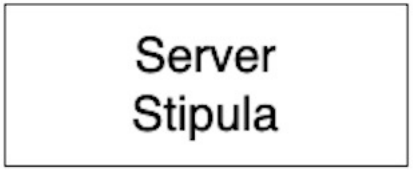
- **2. Specific implementation**
  - Compiler and Virtual Machine
  - Asset transfers
  - Obligations and event scheduling

```
stipula BikeRental {
  asset wallet:stipula_coin_asd345
  field cost, rentingTime, use_code

  agreement (Lender, Borrower)(cost, rentingTime){
      Lender, Borrower: cost, rentingTime
  } ==> @Inactive

  @Inactive Lender : offer(z)[] {
      z -> use_code
  } ==> @Proposal

  @Proposal Borrower : accept()[y]
      (y == cost) {
          y -o wallet
          use_code -> Borrower
          now + rentingTime >>
              @Using {
                  "End_Reached" -> Borrower
                  wallet -o Lender
              } ==> @End
  } ==> @Using

  @Using Borrower : end()[] {
      wallet -o Lender
  } ==> @End
}
```

UNIVERSITY OF PADUA

Possible configurations

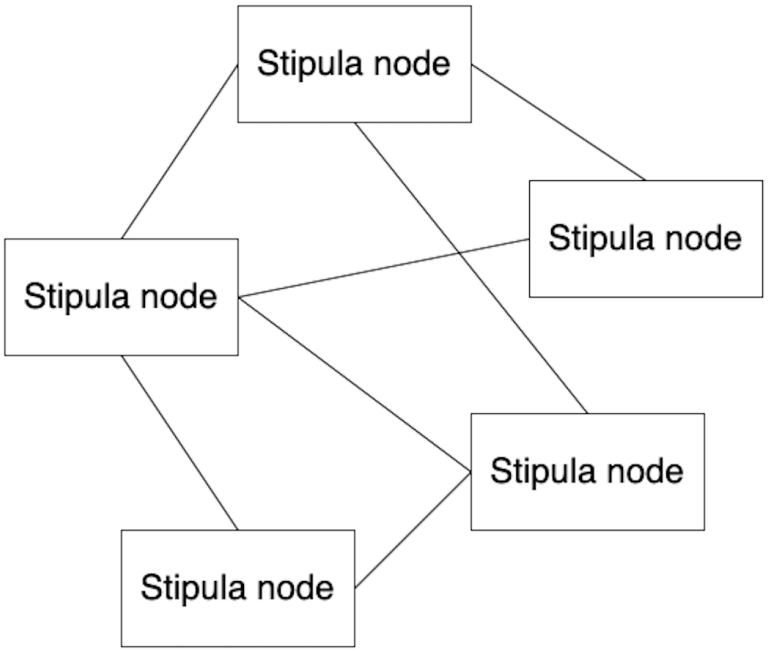
- **Stipula server**

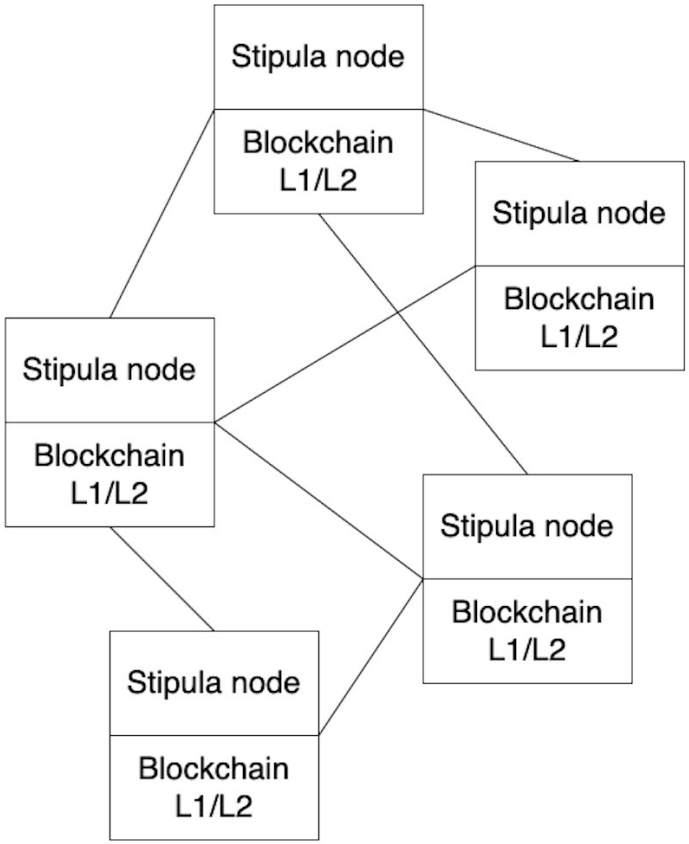


UNIVERSITY OF PADUA

Possible configurations

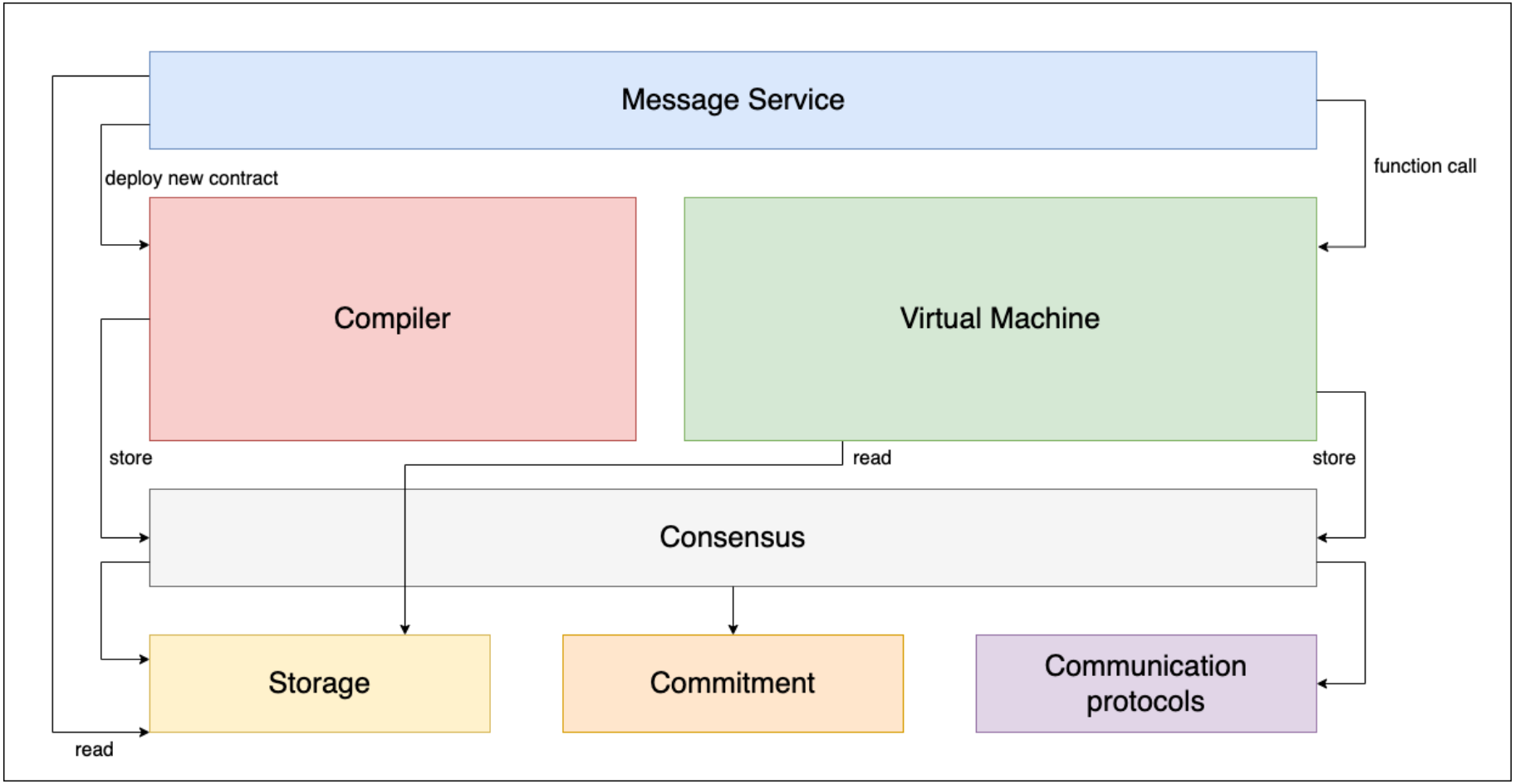
- **Stipula node**





UNIVERSITY OF PADUA

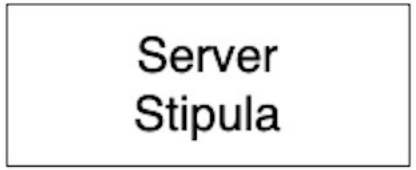
Proposed architecture

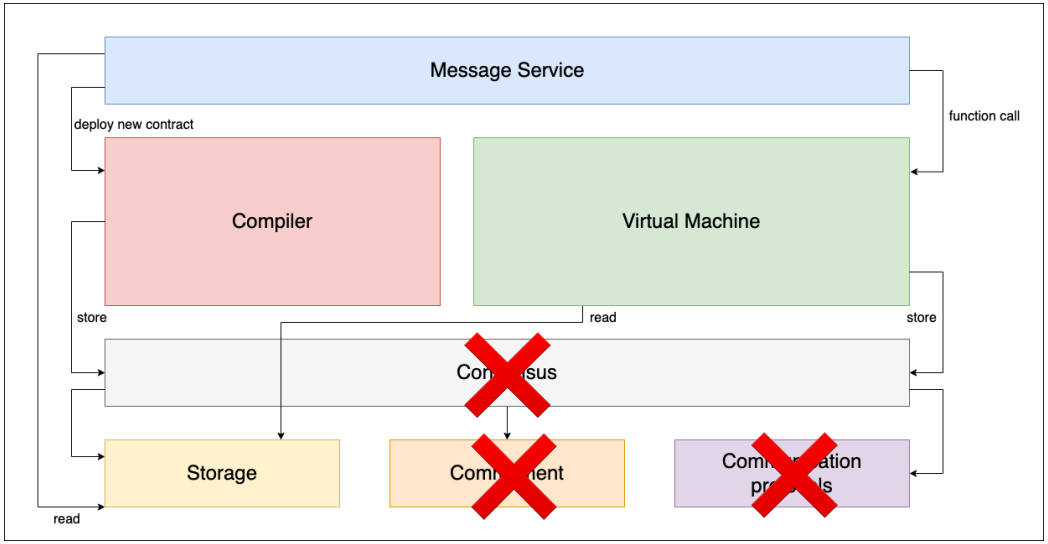


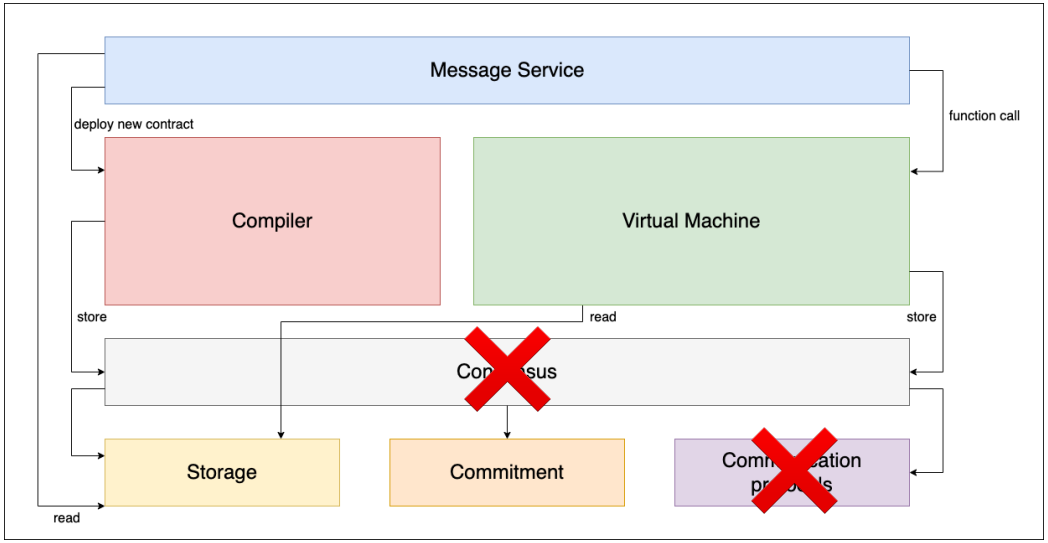
Proposed architecture

- *Compiler*
  - Efficiency
  - Correctness of contract
  - Interoperability
- *Virtual Machine*
  - Contract execution
  - Event scheduling
  - Safety of asset transfers

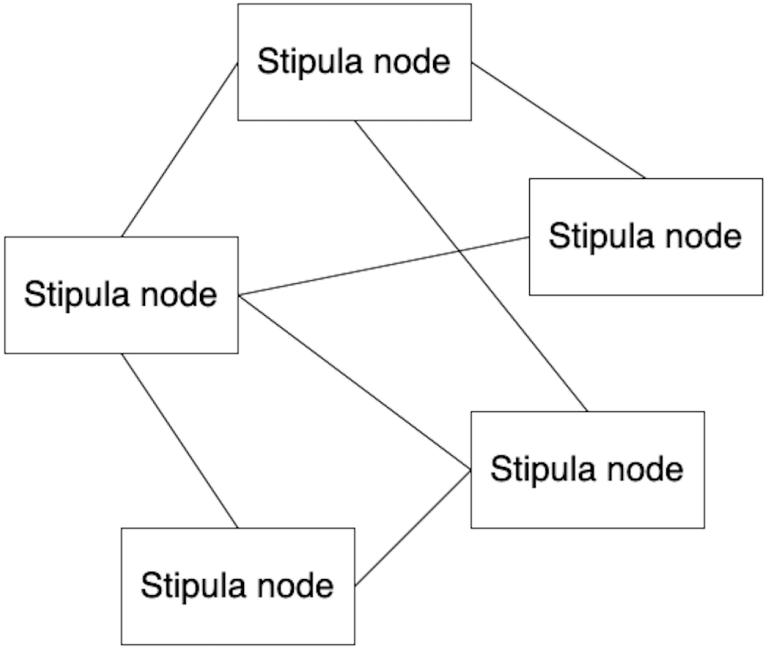
1. Architectural analysis

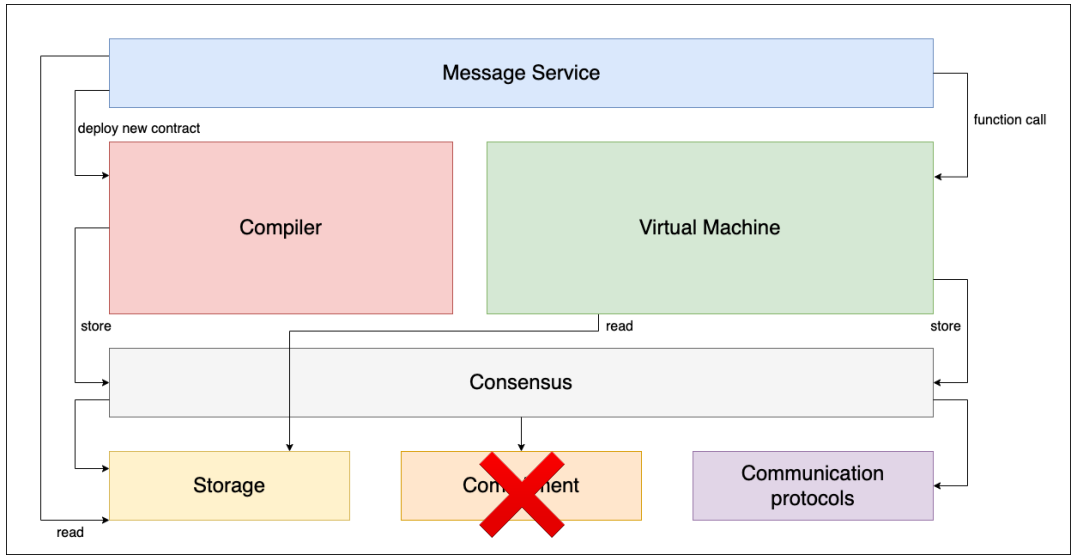




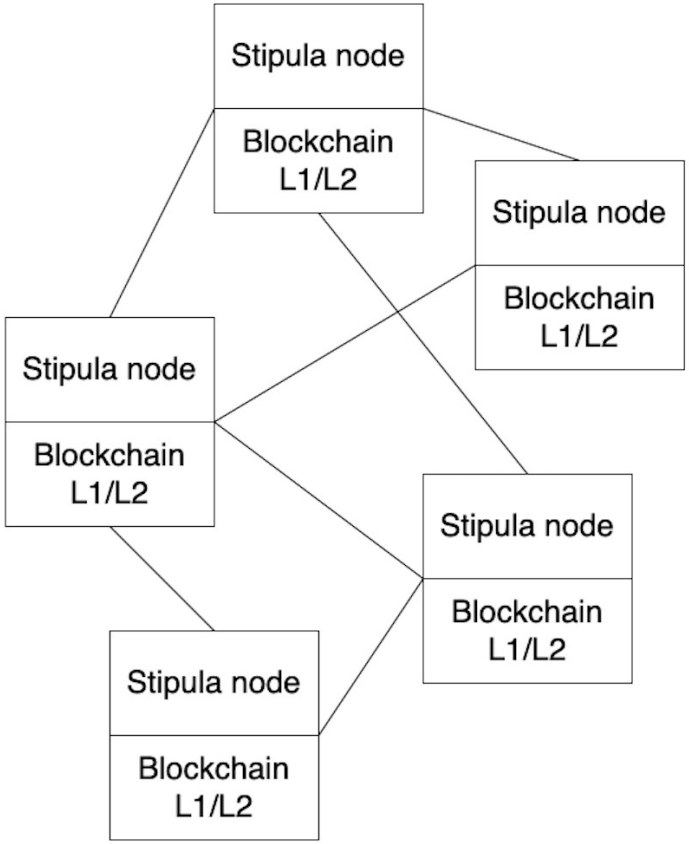


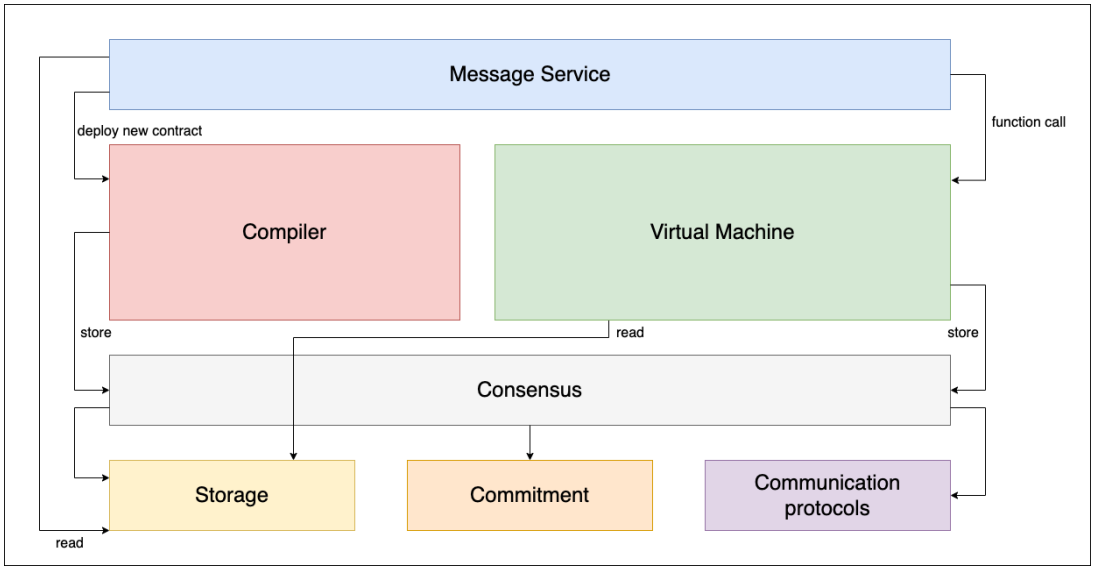
1. Architectural analysis





1. Architectural analysis





Asset models

- **Account-balance-based** and **UTXO** (**Unspent Transaction Output**)
- The UTXO model fits better the needs for the implementation

UTXO model

- A UTXO is a box that contains a certain amount of an asset
- This box is close by a **single-use-seal** which can only be broken by the owner of the quantity of assets in question
- Advantages:
  - Parallelization
  - Partially solves the double-spending problem
  - Security

Limitations of naive UTXO model

- Simply using public and private keys is *limiting*
- Examples of advanced transaction types not supported:
  - Using multisig wallets
  - Funds timelock

Adopted solution

- Definition of a **Script language** to specify the *conditions* under which a transaction can be spent in the network
- Inspired from Bitcoin Script language
- A program in Script language allows a party to provide a cryptographic proof of ownership of a UTXO

Script

- **Pay-to-Public-Key-Hash** (**P2PKH**) in Bitcoin
- `lockScript`: lock the funds
- `unlockScript`: cryptographic proof provided by the owner of the funds
- `script = unlockScript + lockScript`

UNIVERSITY OF PADUA

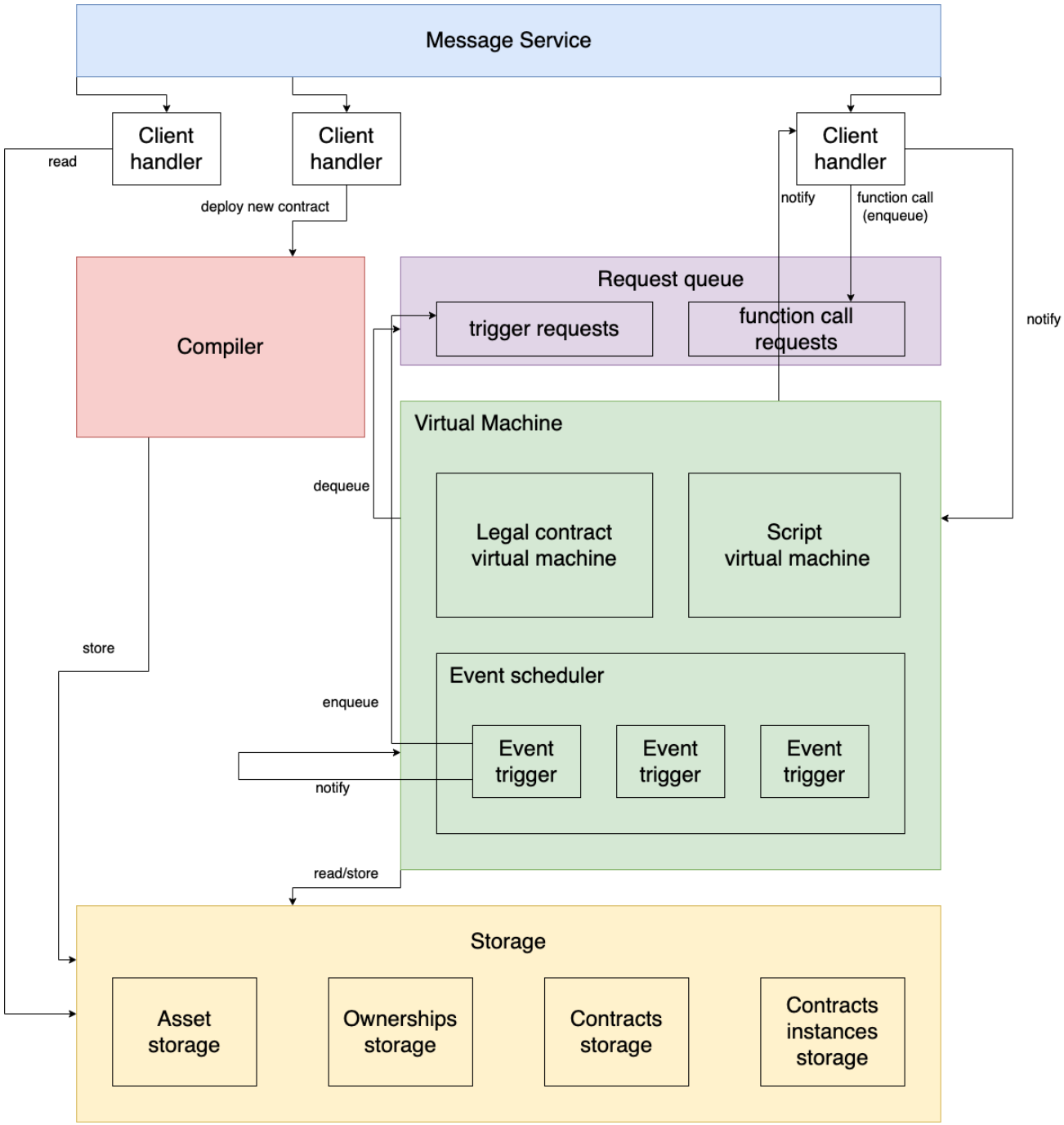
Script

Script

unlockScript

```
PUSH str <signature> PUSH str <pub_key>

+

DUP SHA256 PUSH str <pub_key_hash> EQUAL CHECKSIG
```

lockScript


Message Service
Client handler
Client handler
Client handler
read
deploy new contract
notify
function call (enqueue)
notify
Compiler
Request queue
trigger requests
function call requests
Virtual Machine
dequeue
Legal contract virtual machine
Script virtual machine
store
Event scheduler
enqueue
Event trigger
Event trigger
Event trigger
notify
read/store
Storage
Asset storage
Ownerships storage
Contracts storage
Contracts instances storage

UNIVERSITY OF PADUA

Stipula bytecode

- Compiler (ANTLR tool): lexer, parser, semantic analyzer and code generator
- This language was designed to *mirror* the features of the high-level language and to run on a stack-based virtual machine

UNIVERSITY OF PADUA

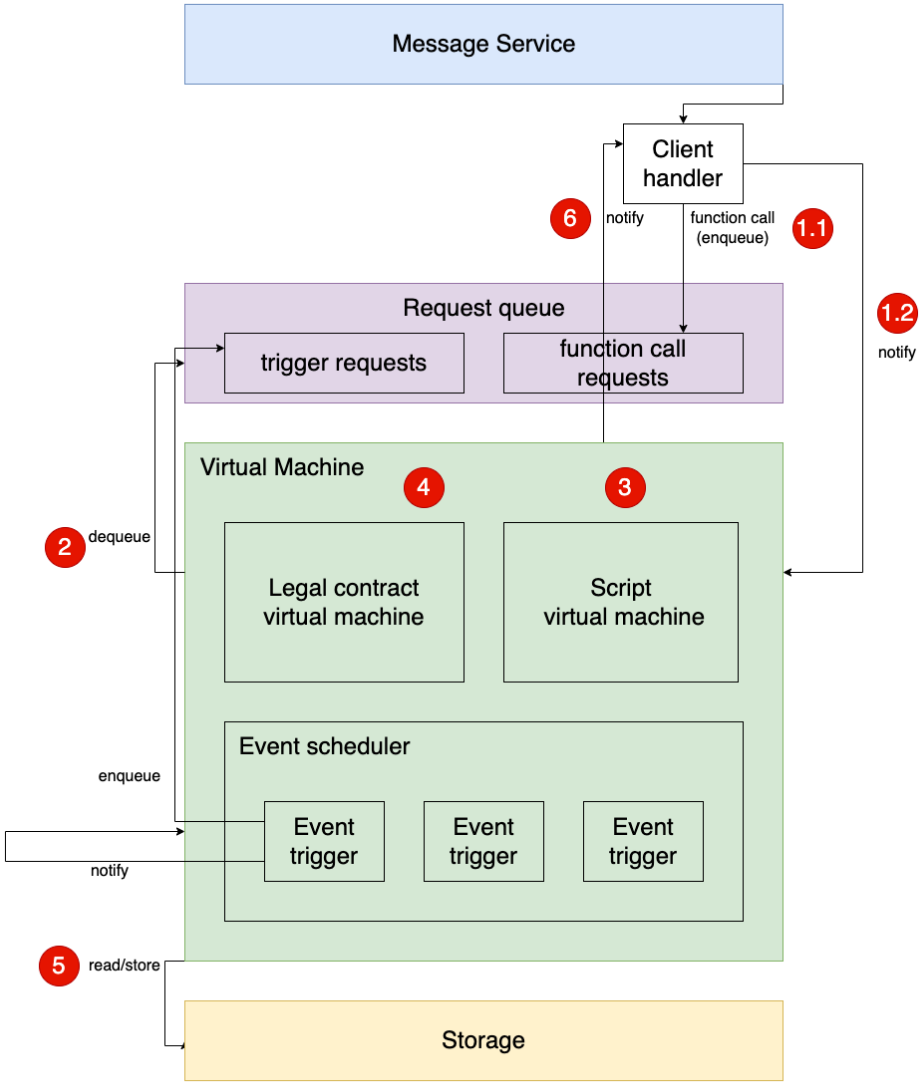
Stipula bytecode – Instructions

Instruction	Behavior
PUSH	$- \rightarrow *$
HALT	$- \rightarrow -$
ADD	$(\texttt{int}, \texttt{int}) \rightarrow \texttt{int}$, $(\texttt{real}, \texttt{real}) \rightarrow \texttt{real}$, $(\texttt{asset}, \texttt{asset}) \rightarrow \texttt{real}$, $(\texttt{asset}, \texttt{real}) \rightarrow \texttt{real}$, $(\texttt{real}, \texttt{asset}) \rightarrow \texttt{real}$, $(\texttt{time}, \texttt{time}) \rightarrow \texttt{time}$
SUB	$(\texttt{int}, \texttt{int}) \rightarrow \texttt{int}$, $(\texttt{real}, \texttt{real}) \rightarrow \texttt{real}$, $(\texttt{asset}, \texttt{asset}) \rightarrow \texttt{real}$, $(\texttt{asset}, \texttt{real}) \rightarrow \texttt{real}$, $(\texttt{real}, \texttt{asset}) \rightarrow \texttt{real}$
MUL	$(\texttt{int}, \texttt{int}) \rightarrow \texttt{int}$, $(\texttt{real}, \texttt{real}) \rightarrow \texttt{real}$, $(\texttt{asset}, \texttt{asset}) \rightarrow \texttt{real}$, $(\texttt{asset}, \texttt{real}) \rightarrow \texttt{real}$, $(\texttt{real}, \texttt{asset}) \rightarrow \texttt{real}$
DIV	$(\texttt{int}, \texttt{int}) \rightarrow \texttt{int}$, $(\texttt{real}, \texttt{real}) \rightarrow \texttt{real}$, $(\texttt{asset}, \texttt{asset}) \rightarrow \texttt{real}$, $(\texttt{asset}, \texttt{real}) \rightarrow \texttt{real}$, $(\texttt{real}, \texttt{asset}) \rightarrow \texttt{real}$

INST	$- \rightarrow -$
AINST	$- \rightarrow -$
GINST	$- \rightarrow -$
LOAD	$- \rightarrow *$
ALOAD	$- \rightarrow *$
GLOAD	$- \rightarrow *$
STORE	$* \rightarrow -$
ASTORE	$* \rightarrow -$
GSTORE	$* \rightarrow -$
AND	$(\texttt{bool}, \texttt{bool}) \rightarrow \texttt{bool}$
OR	$(\texttt{bool}, \texttt{bool}) \rightarrow \texttt{bool}$
NOT	$\texttt{bool} \rightarrow \texttt{bool}$
JMP	$- \rightarrow -$
JMPIF	$\texttt{bool} \rightarrow - \mid\mid \texttt{bool}$
ISEQ	$* \rightarrow \texttt{bool}$
ISLE	$* \rightarrow \texttt{bool}$
ISLT	$* \rightarrow \texttt{bool}$
DEPOSIT	$(\texttt{asset}, \texttt{asset}) \rightarrow -$
WITHDRAW	$(\texttt{real}, \texttt{asset}, \texttt{party}) \rightarrow -$
RAISE	$- \rightarrow \texttt{str}$
TRIGGER	$\texttt{time} \rightarrow -$

2. Specific implementation

```
@Proposal Borrower : accept()[y]
    (y == cost) {
        y -o wallet;
        use_code -> Borrower;
        now + rentingTime >>
            @Using {
                "End_Reached" -> Borrower
                wallet -o Lender
            } ==> @End
} ==> @Using
```

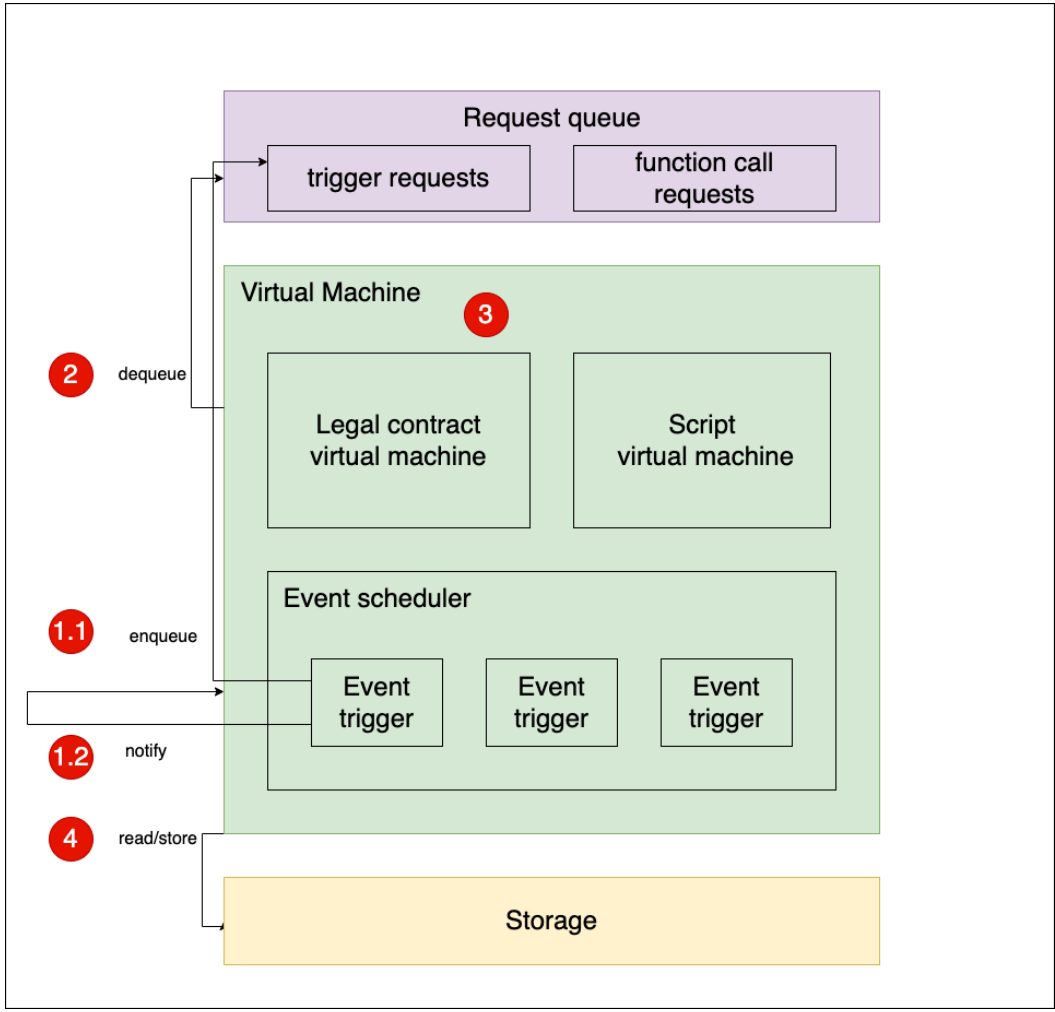


2. Specific implementation


```
fn Proposal Borrower accept Using asset
args:
PUSH asset :y
AINST asset :y
ASTORE y
start:
ALOAD y
GLOAD cost
ISEQ
JMPIF if_branch
RAISE AMOUNT_NOT_EQUAL
JMP end
if_branch:
ALOAD y
GLOAD wallet
DEPOSIT wallet
GLOAD rentingTime
PUSH time now
ADD
TRIGGER obligation_1
end:
HALT
```

2. Specific implementation

```
now + rentingTime >>
    @Using {
        "End_Reached" -> Borrower
        wallet -o Lender
    } ==> @End
```



2. Specific implementation

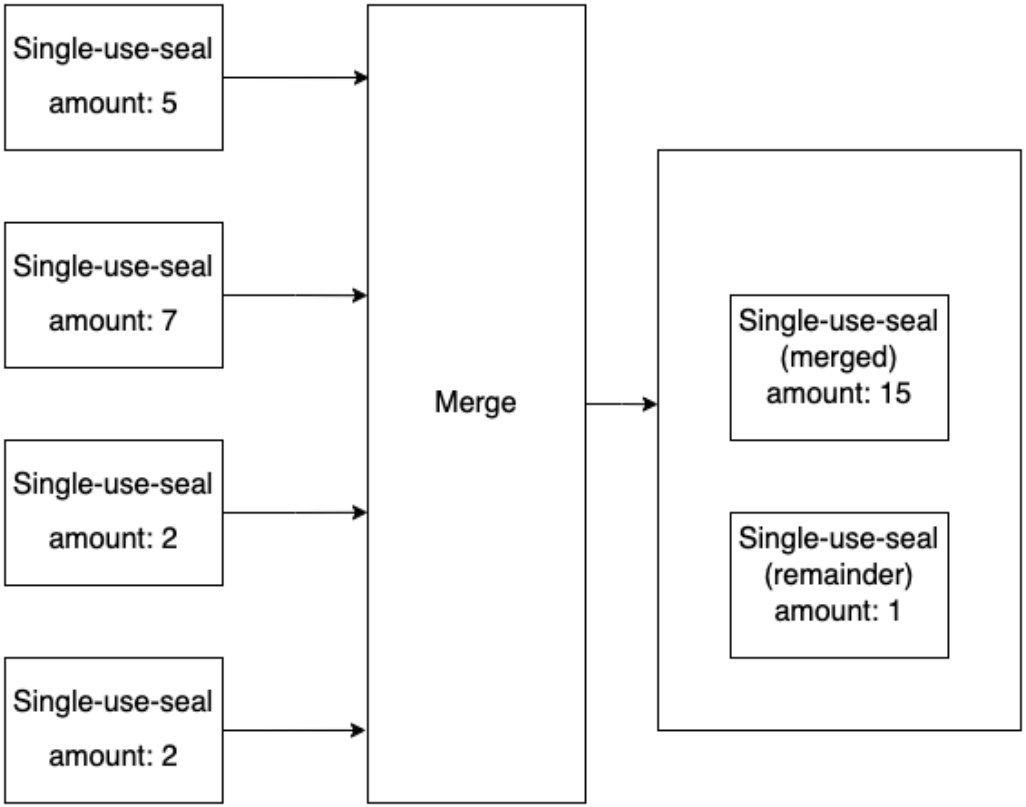

```
obligation Using obligation_1 End
start:
PUSH real 100 2
GLOAD wallet
GLOAD Lender
WITHDRAW wallet
end:
HALT
```

Single-use-seal merge

- In order to make a payment, the user must have available a single-use-seal of the *exact quantity* required by the contract
- If the user does not have a single-use-seal of the requested quantity, the user cannot make the payment

UNIVERSITY OF PADUA

Single-use-seal merge



Future improvements

- Implementation of the consensus module and communication protocols
- Implementation of the commitment module
- Script language extension
- Creation of assets and their distribution
- Implementation of additional software (i.e., SDK, wallet, …)

Challenges

- *Research question*: what is a platform for running Stipula contracts?
  - 1. Architectural analysis
  - 2. Specific implementation
- Mapped out a project direction
- Desgin an implementation-agnostic architecture

UNIVERSITY OF PADUA

Challenges

	Research	Design	Implementation
Virtual Machine, *compiler* and *Stipula bytecode*	✅	✅	✅
Asset management and *Script language*	✅	✅	✅
Distributed context and consent	✅	⚠️	

UNIVERSITY OF PADUA

Thank you for your attention!

DIPARTIMENTO MATEMATICA