Karlsruher Institut für Technologie

Blockchains und Cryptofinance

Übung 3 und Übung 4

Lehrstuhl für Financial Engineering und Derivate

Infos zu Projekt 1

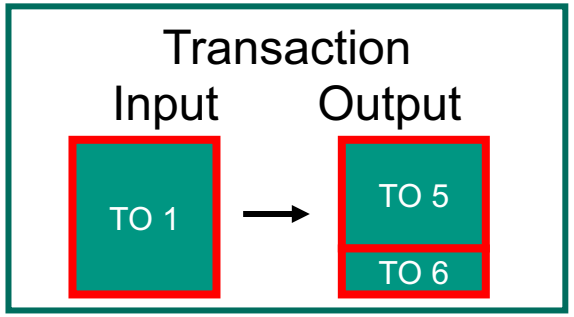
- Am 16.11.2018 (morgen) wird die Liste mit den Projektzuordnungen im Ilias veröffentlicht.
- Materialien zu Projekt 1 werden ebenfalls am 16.11.2018 im Ilias veröffentlicht. Alle Details zur Bearbeitung befinden sich im Notebook des Projekts.
- Projekt 1 muss per Mail and marcel.mueller@kit.edu **bis 27.11.2018, 23:59 Uhr** abgegeben werden. Details zur Abgabe ebenfalls im Notebook.
- Die Projektübung mit Anwesenheitspflicht für mögl. Erhalt der Bonuspunkte findet am **29.11.2018** zum gewohnten Termin statt
- Am 22.11.2018 können in der Übung noch Fragen zum Projekt 1 geklärt werden. Die Frage müssen präzise sein und bis spätestens 21.11.2018 bei mir per Mail eingereicht werden.
- Projekte 2 und 3 werden noch nicht veröffentlicht!

Agenda

- Transaktionen (von letzter Woche)
  - Signieren der Transaction
- Block
  - Struktur
  - Zugriff auf TransactionOutputs
- Blockchain
  - Struktur
  - Genesis Block
  - Hinzufügen von Blöcken
- Verifizierung von Transaktionen
- Hinzufügen von Transaktionen zum Mempool
- Verifizierung von Blöcken
- Mining

Link: TransactionOutput – Transaction

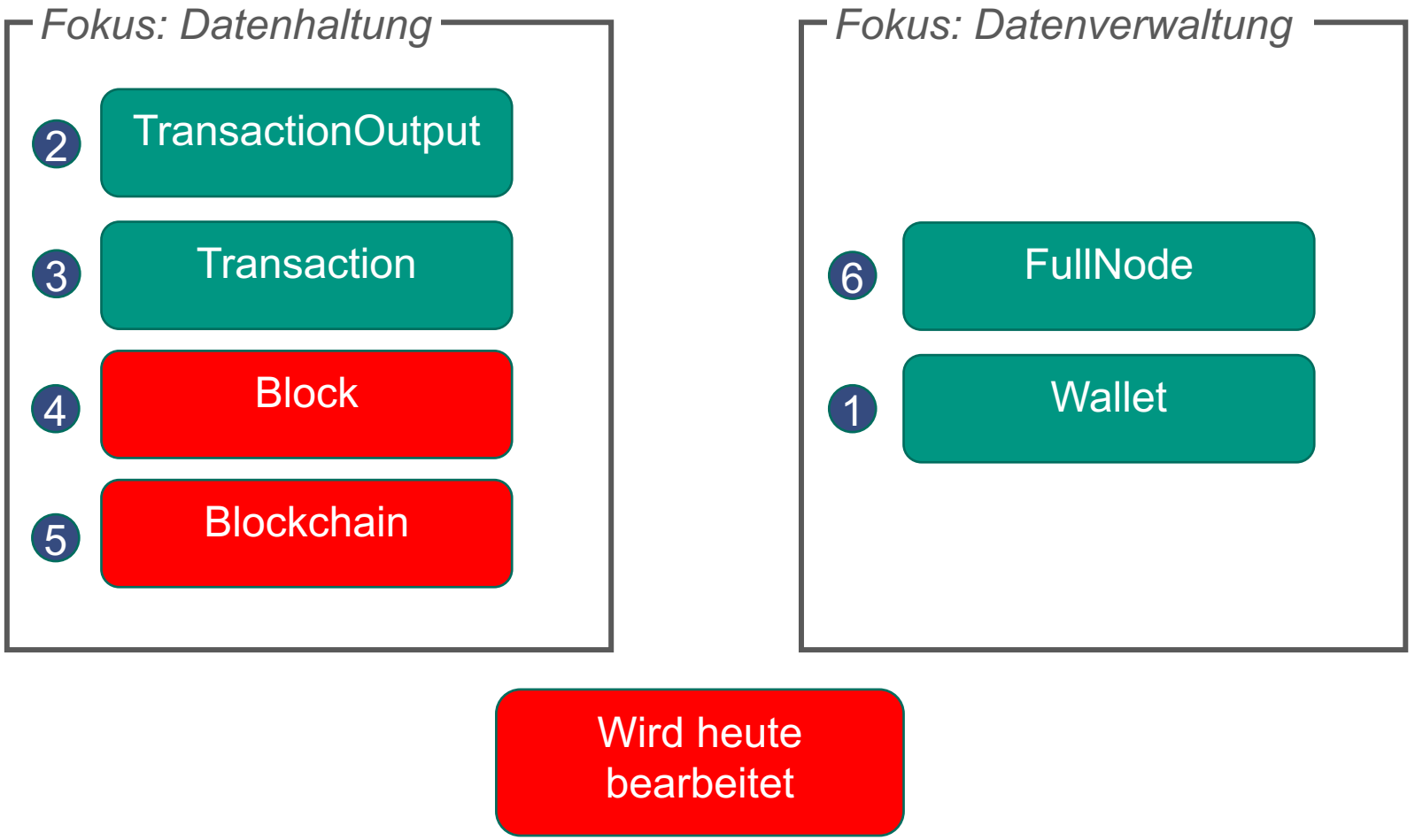
- Wichtig: Jede Instanz eines Objekts hat eine eindeutige HashId (String)



- Transaction besteht u. A. aus:
  1. List mit Verweis auf verwendetet Inputs (TransactionOutput.id)
  2. OrderedDict mit den neuen TransactionOutputs (TransactionOutput.id:TransactionOutput)
  3. Sender Public Key
  4. Signatur des Senders (signieren kann nur, wer Private Key besitzt)

  → Valide: a.) 3. muss zu 1. passen (Inputs in Besitz von Public Key aus 3.)
  b.) Summe aus 1, muss Summe aus 2. entsprechen
  c.) 4. muss zu 3. passen (und auch zu 1. und 2.)

FEDCoin – Übersicht der Pythonklassen



BLOCK UND BLOCKCHAIN

Verschachtelung – Übersicht

Blockchain

- chain→OD(Block.id:Block)

Block

- transactions→OD(Transaction.id:Transaction)
- output_transaction_mapping→OD(TransactionOutput.id:Transaction.id)

Transaction

- inputs→List(TransactionOutput.id)[NUR Referenz]
- outputs→OD(TransactionOutput.id:TransactionOutput)

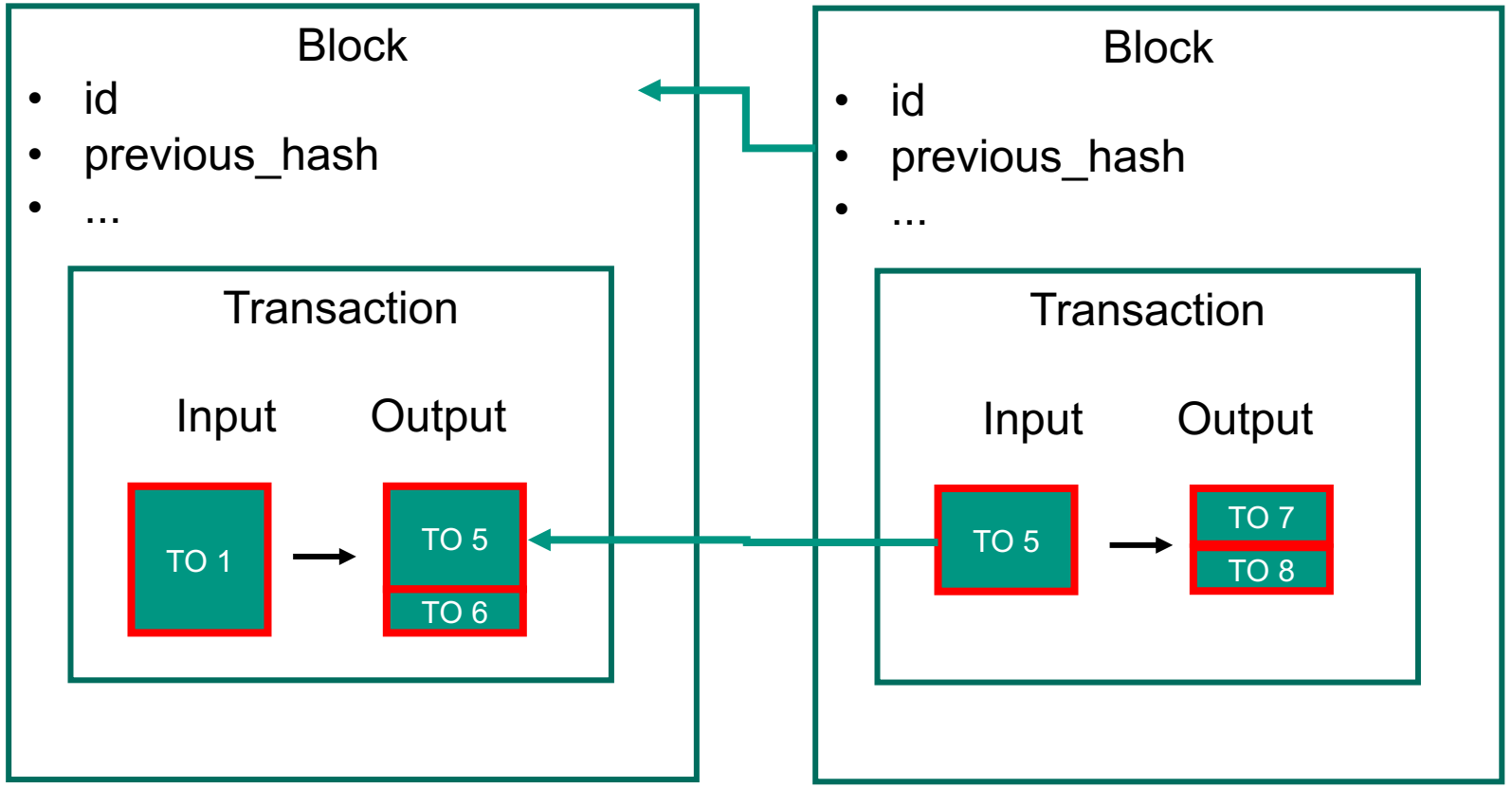
TransactionOutput

Block

- timestamp
- previous_hash
- transactions (OD mit Transaction.id:Transaction)
- output_transaction_mapping
  (OD mit TransactionOutput.id:Transaction.id)
  →Lediglich um effizienten Zugriff auf TransactionOutput zu gewährleisten, keine zusätzlicher „Inhalt“
- nonce (integer) → Relevant fürs Mining
- id (Hashwert)

→ Einfache Containerklasse

Link: Block und Blockchain



Blockchain

- Klassenvariablen:
  - initial_difficulty (integer)
  - initial_block_reward (integer)
  - half_time_in_blocks (integer)
  - max_block_size_in_transactions (integer)
- chain (OD -- Block.id:Block)
- mempool (OD -- Transaction.id:Transaction)
- UTXOs (OD -- TransactionOutput.id:Block.id)
- mempool_UTXOs (OD -- TransactionOutput.id:Transaction.id)
- pow (integer) → Gesamter investierter Proof-of-Work der Chain

Genesis Block

- 1. Block jeder Blockchain Instanz → Wird durch Konstruktor angelegt.

- Inhalt des Genesis Blocks ist eine spezielle Transaktion ohne Input und mit einem TransactionOutput (value = initial_block_reward) Empfänger ist die Genesis Wallet.

- Der Genesis Block wird *hard-coded* hinterlegt

→ Ohne Genesis Block wäre keine einzige Transaktion möglich

Hinzufügen von Blöcken

- Nachdem ein Block der Chain hinzugefügt wurde, müssen folgende Aufgaben erledigt werden:

1. **Update pow**

2. **Update UTXOs**: Durch den neuen Block entstehen neue UTXOs und alte UTXOs werden verwendet

3. **Update Mempool**: Transaktionen, die im neuen Block enthalten sind, sollten aus dem Mempool entfernt werden

VERIFIZIERUNG UND MINING

Verifizierung von Transaktionen

- Wird innerhalb der Transaction Klasse implementiert
- Folgende Verifizierungsschritte notwendig:

1. Verifizierung der Signatur (bereits implementiert)
2. Verifizierung der Transaction ID (passt HashID zum Inhalt) → Der Schritt ist eigentlich redundant, mein Fehler bei der Implementierung. Lassen wir aber trotzdem drin ;-)
3. Verifizierung der verwendeten Inputs → Gehört verwendeter Input auch dem Sender der Transaction
4. Verifizierung, ob Summe Inputs = Summe Outputs

Hinzufügen von Transactions zum Mempool

- process_transaction:

1. Überprüfe, ob Transaktion valide gegeben der aktuelle Zustand der Chain

2. Falls 1. valide, füge die Transaktion dem mempool hinzu.

3. Aktualisiere mempool_UTXOs

Verifizierung von Blöcken

- Folgende Kriterien müssen erfüllt werden, damit Block valide:

1. Alle Transactions müssen valide sein

2. Max. Anzahl der zulässigen Transactions pro Block darf nicht überschritten werden

3. BlockId muss überprüft werden (hier nicht redundant!, da Block nicht signiert wird und außer den Transactions noch zusätzlichen relavanten Inhalt hat)

Mining – Coinbase Transaction

- Entlohnung der Miner
- 1. Transaction in jedem Block
- Value entspricht aktuellem Coinbase Reward (basierend auf initial_block_reward, half_time_in_blocks und Länge der Chain)
- Keine Inputs, nur ein TransactionOutput

→ Wird vom Miner erzeugt und dem Block hinzugefügt. Wenn Block langfristig in die Blockchain kommt → Miner wird entlohnt.

Anmerkung: Genesis Transaction war Coinbase Transaction (des 1. Blocks)

Mining – Proof-of-Work

- Idee: Miner muss einen neuen validen Block erzeugen und diesen an Netzwerk posten.
- Valide bedeutet:

1. Formale Kriterien müssen eigehalten werden (z.B. # der Transactions)
2. Alle Transactions müssen valide sein (inkl. Coinbase Transaction)
3. Die BlockId muss mit einer durch initial_difficulty festgelegten Anzahl an Nullen beginnen.

→Schlachtplan Mining:

1. Coinbase Transaction erstellen
2. Transactions aus Mempool bündeln
3. PoW Rätsel lösen