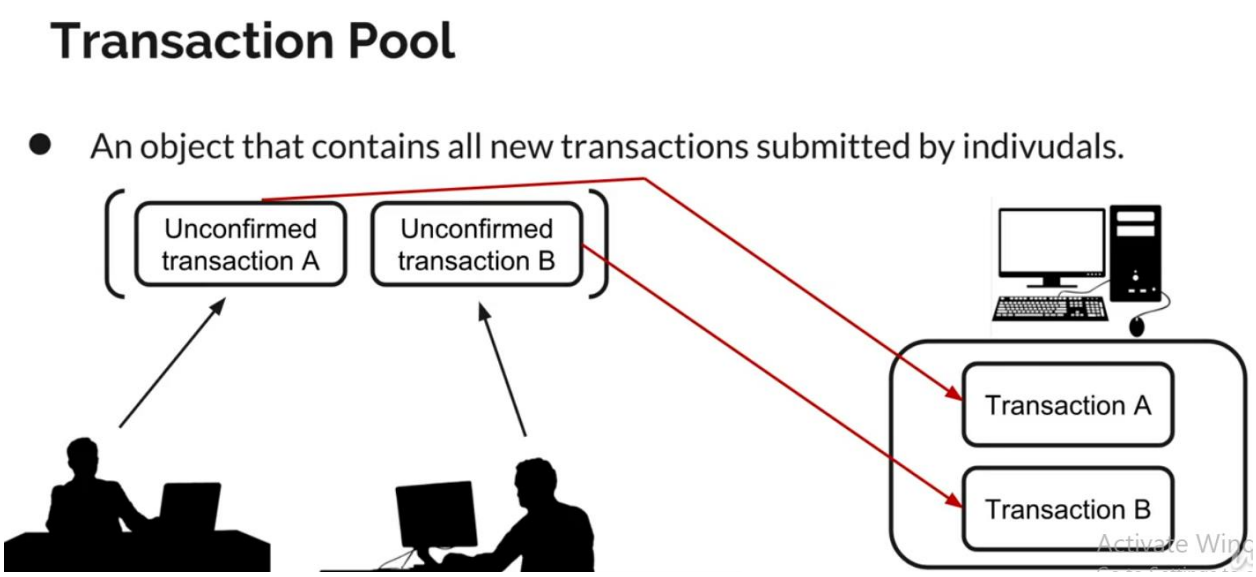
BlockChain

Collect Transactions in a Pool:

Transaction Pool:



Transaction Pool - Add Transaction:

wallet > JS transaction-pool.js > ...

```
1   class TransactionPool {
2       constructor(){
3           this.transactions = [];
4       }
5
6       updateOrAddTransaction(transaction){
7           let transactionWithId = this.transactions.find(t => t.id === transaction.id);
8
9           if(transactionWithId) {
10              this.transactions[this.transactions.indexOf(transactionWithId)] = transaction;
11          } else {
12              this.transactions.push(transaction);
13          }
14      }
15  }
16
17  module.exports = TransactionPool;
```

BlockChain

Test the Transaction Pool:

wallet > JS transaction-pool.test.js > ...

```
const TransactionPool = require('./transaction-pool');
const Transaction = require('./transaction');
const Wallet = require('./index');

describe('TransactionPool', ()=>{
    let tp, wallet, transaction;

    beforeEach(() =>{
        tp = new TransactionPool();
        wallet = new Wallet();
        transaction = Transaction.newTransaction(wallet, 'r4nd-4dr355', 30);
        tp.updateOrAddTransaction(transaction);
    });

    it('adds a transaction to the pool', () =>{
        exportAllDeclaration(tp.transactions.find(t=>t.id === transaction.id)).toEqual(transaction);
    });

    it('updates a transaction inthe pool', ()=>{
        const oldTransaction = JSON.stringify(transaction);
        const newTransaction = transaction.update(wallet, 'foo-4ddr355', 40);
        tp.updateOrAddTransaction(newTransaction);

        expect(JSON.stringify(tp.transactions.find(t => t.id === newTransaction.id)))
        .not.toEqual(oldTransaction);

    });

});
```

Create Transactions with the Wallet:

JS index.js > ...

```
createTransaction(recipient, amount, TransactionPool){
    if(amount > this.balance){
        console.log(`Amount: ${amount} exceeds current balance: ${this.balance}`);
        return;
    }

    let transaction = TransactionPool.existingTransaction(this.publicKey);

    if(transaction){
        transaction.update(this, recipient, amount);
    }else{
        transaction= Transaction.newTransaction(this, recipient, amount);
        TransactionPool.updateOrAddTransaction(transaction);
    }
    return transaction;

}
```

BlockChain

wallet > JS transaction-pool.js > TransactionPool > existingTransaction

```
15
16      existingTransaction(address){
17          return this.transactions.find(t => t.input.address === address);
18
19      }
20  }
```

Get Transactions:

wallet > JS index.js > Transaction

```
2   const Transaction = require('./transactions');
```

app > JS index.js > app.get('/transactions') callback

```
8   const Wallet = require('../wallet');
9   const TransactionPool = require('../wallet/transaction-pool');
10
```

app > JS index.js > app.get('/transactions') callback

```
35
36  app.get('/transactions', (req, res) =>{
37      res.json(tp.transactions);
38
39  })
```

Post Transactions:

JS transaction-pool.js | JS index.js app | JS index.js wallet | JS transaction-pool.test.js

app > JS index.js > ...

```
41  app.post('/transact', (req, res) => {
42      const { recipient, amount} =req.body;
43      const transaction = wallet.createTransaction(recipient, amount, tp);
44      res.redirect('/transactions');
45  });
```

BlockChain

Add the Transaction Pool to the Peer to peer Server:

app > JS p2p-server.js > P2pServer

```
55    sendChains(socket){
56        socket.send(JSON.stringify(this.blockchain.chain));
57    }
58    sendTransaction(socket, transaction){
59    socket.send(JSON.stringify({type : MESSAGE_TYPES.transaction, transaction}));
60    }
```

app > JS p2p-server.js > P2pServer

```
5    const MESSAGE_TYPES = {
6        chain: 'CHAIN',
7        transaction: 'TRANSACTION'
8    };
```

app > JS p2p-server.js > P2pServer

```
11   class P2pServer {
12       constructor(blockchain, transactionPool){
13           this.blockchain = blockchain;
14           this.transactionPool = transactionPool;
15           this.sockets = [];
16
17       }
```

```
14   const app = express();
15   const bc = new Blockchain();
16   const wallet = new Wallet();
17   const tp = new TransactionPool();
```

BlockChain

Handle Transaction Messages in the Peer to peer Server:

app > JS p2p-server.js > P2pServer > messageHandler > socket.on('message') callback

```
47      messageHandler(socket){
48          socket.on('message', message =>{
49              const data = JSON.parse(message);
50              switch(data.type){
51                  case MESSAGE_TYPES.chain:
52                      this.blockchain.replaceChain(data.chain);
53                      break;
54                  case MESSAGE_TYPES.transaction:
55                      this.transactionPool.updateOrAddTransaction(data.transaction);
56                      break;
57              }
58
59          } );
60      }
```

app > JS index.js > app.post('/transact') callback

```
40
41  app.post('/transact', (req, res) => {
42      const { recipient, amount} =req.body;
43      const transaction = wallet.createTransaction(recipient, amount, tp);
44      p2pServer.broadcastTransaction(transaction);
45      res.redirect('/transactions');
46  });
```

Public Key Endpoint:

app > JS index.js > app.get('/public-key') callback

```
47
48  app.get('/public-key', (req, res) =>{
49      res.json({publicKey: wallet.publicKey});
50  });
```