Trains, Hotels, and Async

Dean Tribble
February 2019

AGORIC

Train-Hotel Problem

- Make travel arrangements with a hotel and train
  - Purchase BOTH or NEITHER ticket
  - Where the hotel and train are on different shards/chains/vat/...

- Thank you, Andrew Miller

AGORIC

The atomic approach

In a transaction

- Purchase the hotel reservation
- Purchase the train ticket
- If either fail, abort and purchase neither

- BUT distributed atomic transactions are hard
  - And likely infeasible in the byzantine context

Distributed atomicity considered harmful

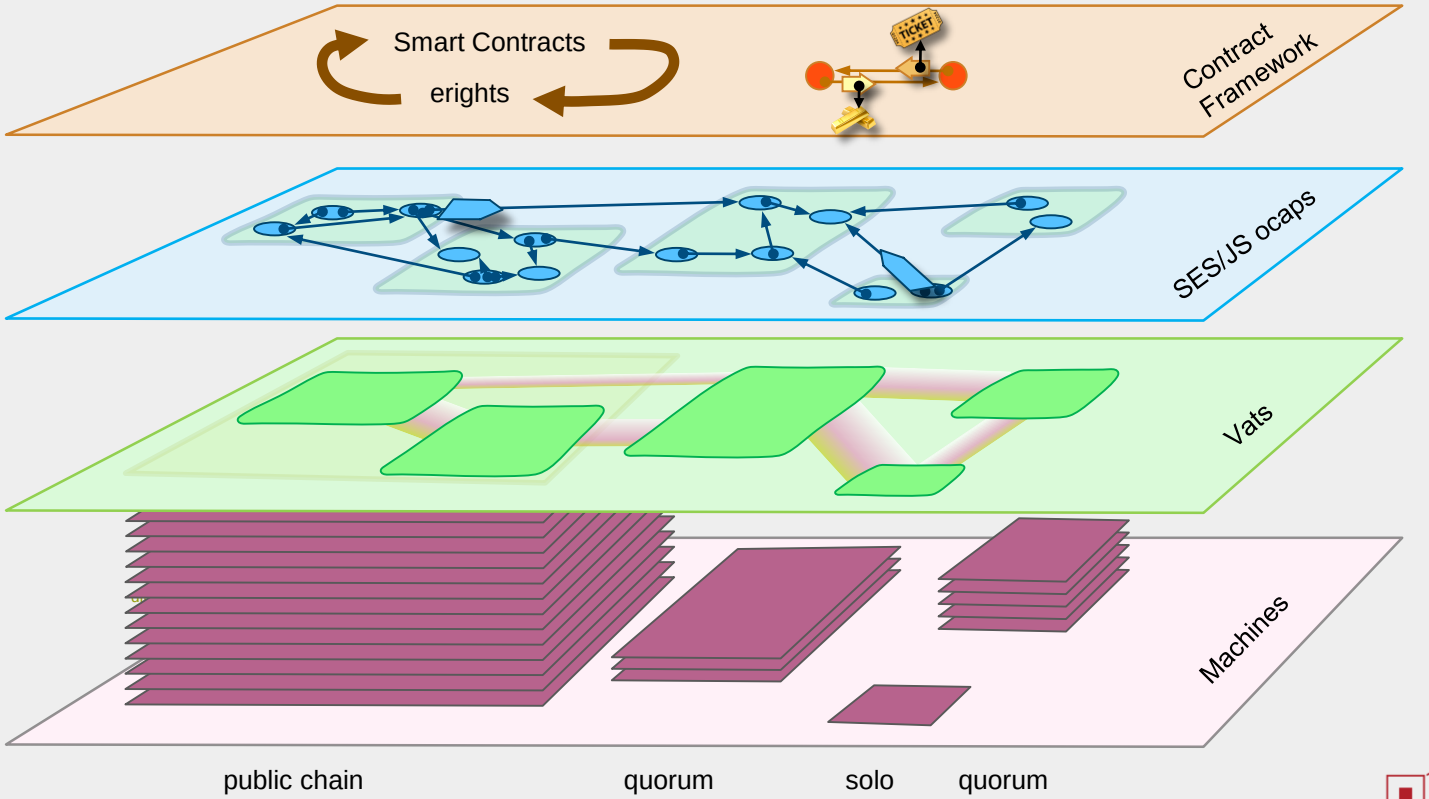
- Goal: Transfer $1 from an account in one bank to an account in another
  - Use distributed transaction
  - Lock both accounts
  - Update both balances
  - Coordinate on the commit
  - Last participant is an uncompensated option

- What do banks do?

Who are we?

- Pioneers in distributed electronic markets
  - Agoric Open Systems papers
- Driven security and async support into JavaScript
  - Promise, proxies, async/await, realms, etc.
- Built brokerage information systems
- Built multi-billion-dollar payment systems

AGORIC


Smart Contracts
erights
TICKET
Contract Framework
SES/JS ocaps
Vats
Machines
public chain
quorum
solo
quorum

AGORIC

An object-capability (ocap) is...

a transferrable, unforgeable authorization to use the object it designates

In a programming language...

Just an object reference

AGORIC

Simple ocap pattern

```
// define a gate counter
function makeEntryExitCounter() {
  let count = 0;
  return def({
    countEntry() { return ++count; },
    countExit()  { return --count; }
  });
}
```

```
// create a gate counter
const counter = makeEntryExitCounter();

// share it with the guards
entryGuard.use(counter.countEntry);
exitGuard.use(counter.countExit);
```

Simplified, of course!

AGORIC

Mints and Purses

makeMint(name) ⇒ mint
mint(amount) ⇒ Purse

Purse
  getBalance() ⇒ number
  getIssuer() ⇒ Issuer
  deposit(amount, srcPurse)

```
// setup for concert tickets
const ticketsM =
        mintMaker.makeMint("Concert");

// publish concert ticket issuer
const concertI = ticketsM.getIssuer();


// create and return a ticket
return ticketsM.mint(1);


// send a new ticket to carol
const ticketP = ticketsM.mint(1);
carol.receiveTicket(ticketP);
```

AGORIC

Issuers and exclusive transfer

Issuer
- makeEmptyPurse
- **getExclusive**(Purse)
  - ⇒ Purse

```
// carol confirms ticket and returns payment
receiveTicket(ticketP) {
  const tkt = concertI.getExclusive(ticketP);
  ...
  return paymentPurse;
}
```

```
// send a new ticket to carol and get a payment
const ticketP = ticketsM.mint(1);
const paymentP = carol.receiveTicket(ticketP);
myAccount.deposit(paymentP);
```

AGORIC

Escrow contract and market safety

```
// provide a new ticket and get a payment via a new escrow with carol
const [ticketFacet, payFacet] = EscrowExchange.make(ticketIssuer, moneyIssuer);
carol.receiveTicket(ticketFacet);
payFacet.exchange(ticketsM.mint(1), myAccount, ticketPrice);
```

AGORIC

Escrow contract and market safety

```
// provide a new ticket and get a payment via a new escrow with carol
const [ticketFacet, payFacet] = EscrowExchange.make(ticketIssuer, moneyIssuer);
carol.receiveTicket(ticketFacet);
payFacet.exchange(ticketsM.mint(1), myAccount, ticketPrice);


// carol provides the payment and gets the ticket
receiveTicket(ticketFacet) {
  const carolFacet = EscrowExchange.getExclusive(ticketFacet);
  carolFacet.exchange(paymentPurse, myTickets, 1);
}
```

AGORIC

Escrow agent contract in twenty lines

```
export function EscrowExchange(a, b) {  // a from Alice, b from Bob
 function makeTransfer(issuerP, srcPurseP, dstPurseP, amount) {
   const escrowPurseP = issuerP.getExclusive(amount, srcPurseP); // escrow the goods
   return def({
     phase1() { return escrowPurseP; },
     phase2() { return dstPurseP.deposit(amount, escrowPurseP); }, // deliver the goods
     abort()  { return srcPurseP.deposit(amount, escrowPurseP); }  // return the goods
   });
 }

 const aT = makeTransfer(a.srcP, b.dstP, b.amountNeeded); // setup transfer from alice to bob
 const bT = makeTransfer(b.srcP, a.dstP, a.amountNeeded); // setup transfer from bob to alice
 return Vow.race([Vow.all([aT.phase1(), bT.phase1()]),    // if both escrow actions succeed…
                  failOnly(a.cancellationP),
                  failOnly(b.cancellationP)])
   .then( _  => Vow.all([aT.phase2(), bT.phase2()]),      // … then complete the transaction
          ex => Vow.all([aT.abort(),  bT.abort(), ex]));  // otherwise, return the supplied goods
};
```

AGORIC

Escrow agent

- Make a new Transfer in each direction; call **Stage1**
  - Transfer - one per direction
    - **Stage1** Transfer into a new escrow purse
    - **Stage2** Transfer from escrow purse to dest
    - **Abort** Transfer from escrow purse to src
- Race:
  - If ***all*** transfer.stage1 succeed, call **Stage2**
  - If ***any*** fail or cancel, call **Abort**

Covered call option

A bounded-time right to purchase a good
the right is itself a good

- Make a new escrow for the desired transaction
- Post the digital good to the sell-facet
  - with an expiration cancelation
- Return a CoveredCall, containing the buy side
  - **exercise** – invoke the buy-facet of the escrow
  - **getExclusive** – the option is *also* a digital good

Train-Hotel Problem

- Make travel arrangements with a hotel and train
  - Purchase BOTH or NEITHER ticket
  - Where the hotel and train are on different shards/chains/vat/...

The simple async solution

- Request covered-call options for ticket & hotel
  - concurrent and asynchronous
- When enough have fulfilled to enable travel…
  - **Decide**! - locally atomic
- Exercise the selected options
  - Concurrent and asynchronous
  - Optionally, reject unused options

More general vacation

Intermediate states are visible so...

- Easy to request overlapping options from multiple sources
  - Different hotels on different shards
  - Multiple cities and/or times
  - Code can reason about response time and timeliness

Power of a framework

- Reusable components by infrastructure experts
  - Escrow agent
  - Covered call
  - Auctions
  - Multi-goods purchase!

- Dapps by domain experts

Async to the rescue!

- Remote invocation supports digital assets on other machines/chains/vats
- Distributed commerce requires
  - Acquire rights async
  - Decide locally
  - Apply consequences async

Escrow contract in twenty lines

```
export function escrowExchange(a, b) {  // a from Alice, b from Bob
 function makeTransfer(srcPurseP, dstPurseP, amount) {
   const issuerP = Vow.join(srcPurseP.getIssuer(), dstPurseP.getIssuer());
   const escrowPurseP = issuerP.getExclusive(amount, srcPurseP); // escrow the goods
   return def({
     phase1() { return escrowPurseP; },
     phase2() { return dstPurseP.deposit(amount, escrowPurseP); }, // deliver the goods
     abort()  { return srcPurseP.deposit(amount, escrowPurseP); }  // return the goods
   });
 }

 const aT = makeTransfer(a.srcP, b.dstP, b.amountNeeded); // setup transfer from alice to bob
 const bT = makeTransfer(b.srcP, a.dstP, a.amountNeeded); // setup transfer from bob to alice
 return Vow.race([Vow.all([aT.phase1(), bT.phase1()]),    // if both escrow actions succeed…
                  failOnly(a.cancellationP),
                  failOnly(b.cancellationP)])
   .then( _  => Vow.all([aT.phase2(), bT.phase2()]),      // … then complete the transaction
          ex => Vow.all([aT.abort(),  bT.abort(), ex]));  // otherwise, return the supplied goods
};
```

Questions?

AGORIC

Find out more, get involved

- https://agoric.com
- @agoric

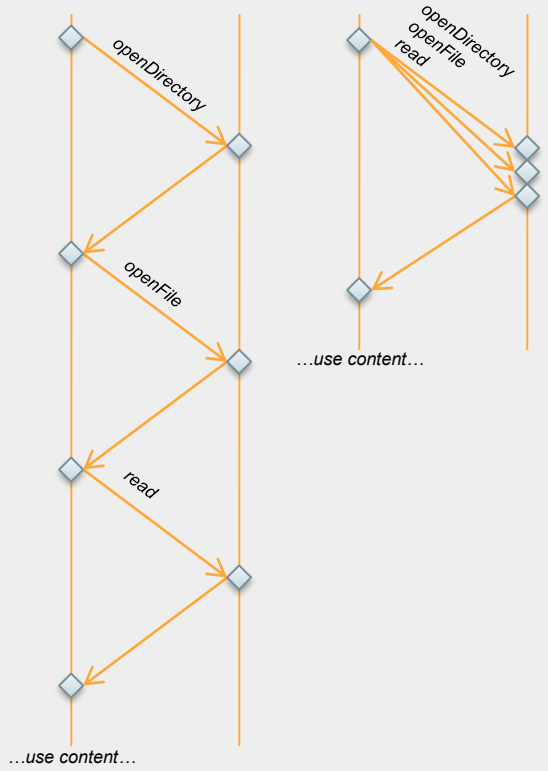
- Weekly progress Proof of Work Newsletter
  - http://proofofwork.news/
  - https://agoric.com/weekly-updates/

- Download proof of concept
  - https://github.com/Agoric/PlaygroundVat

AGORIC

Pipelining

```
// define a gate counter
const dir = storage ! openDirectory("foo");
const file = dir ! openFile("bar.txt");
const content = file ! read();
…use content…
```



Pipelining in prior systems resulted in
>100x reduction in network roundtrips

AGORIC

Web 2 breaches demonstrate the inadequacy of identity-based security for composing software systems

> *The buzzword "web3" suggests the lax, security-poor programming habits of the web.* ***When crypto or smart contracts are programmed like a web page they are doomed.*** *Sustainably successful blockchains and their apps are based on far more secure, careful, and slow programming methods."*
>
> *Nick Szabo, Feb 2018*

AGORIC

What is a smart contract?

A contract-like arrangement, expressed in code, where the behavior of the program enforces the terms of the contract