The Billion Dollar State Machine

Processing auth stream messages with XState

Who am I?

Ian Sutherland

Architect, Head of DX and OSS at Neo Financial

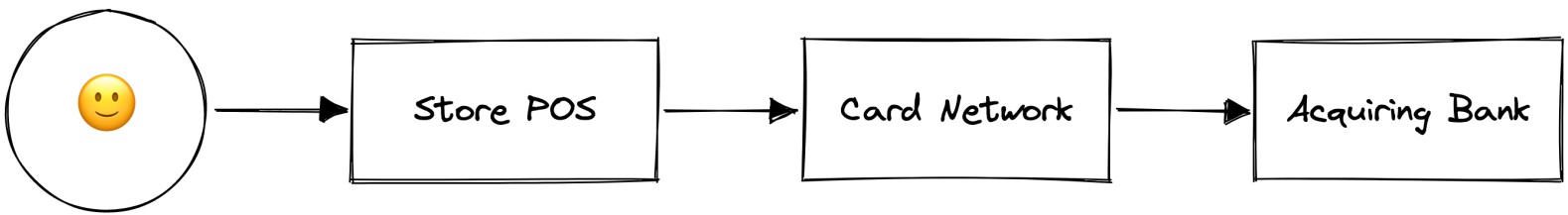
Node.js Contributor, Tooling Working Group

@iansu

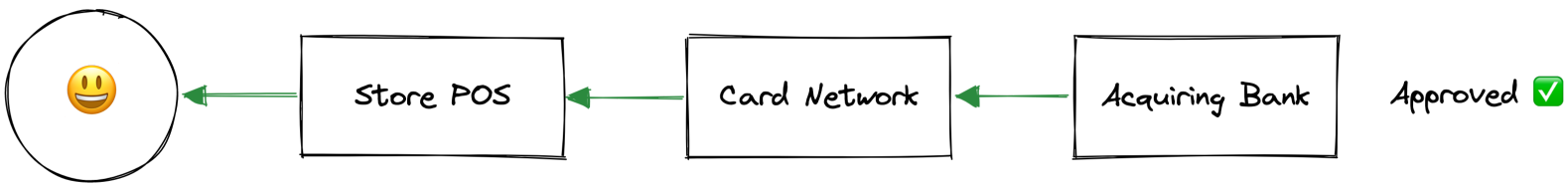
What We’re Covering Today

- How do credit cards transactions work
- What is a state machine
- How can a state machine be used to help process credit card authorization messages

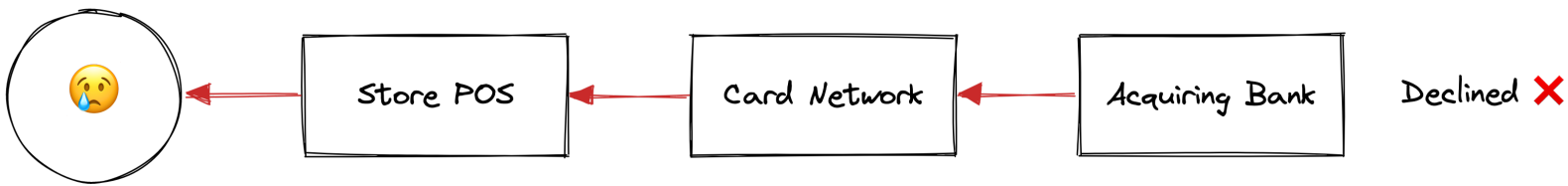
How do credit card transactions work



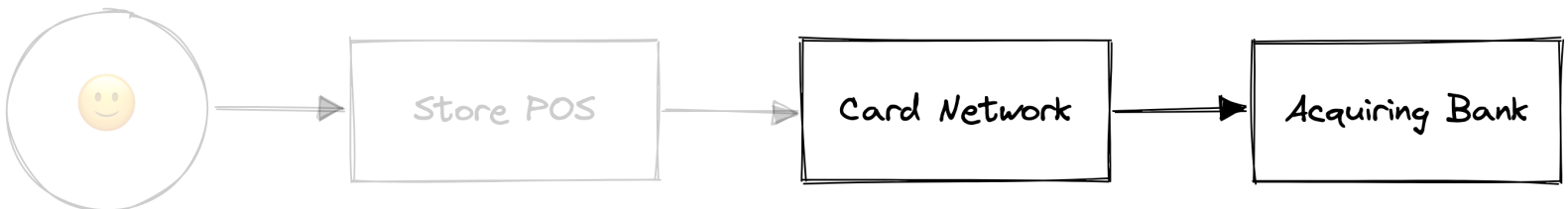
How do credit card transactions work



How do credit card transactions work



How do credit card transactions work



How are messages sent

- Messages are sent over a socket
- Not a WebSocket
- A TCP socket 😨

How are messages encoded

- Messages are encoded using a standard called `ISO 8583`

> ISO 8583 is an international standard for financial transaction card originated interchange messaging. It is the International Organization for Standardization standard for systems that exchange electronic transactions initiated by cardholders using payment cards.

- Originally created in 1987
- This version is still commonly used 😉
- There are newer versions from 1993 and 2003

How are messages encoded

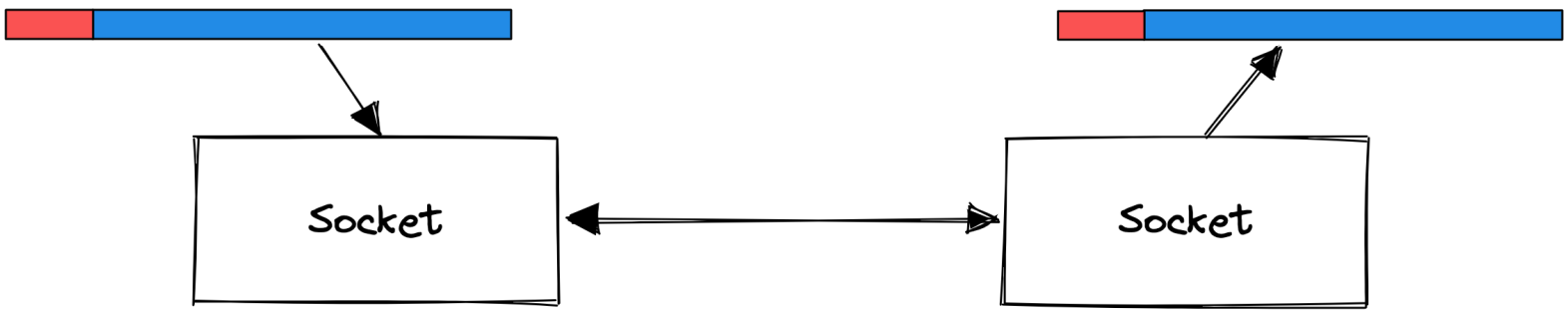
Bytes	Content
1-2	Length header
3-6	Message type
7-14	Bitmap
15-N	Data elements

How are messages encoded

The important part is the length header and then the message body

Let's visualize that like this

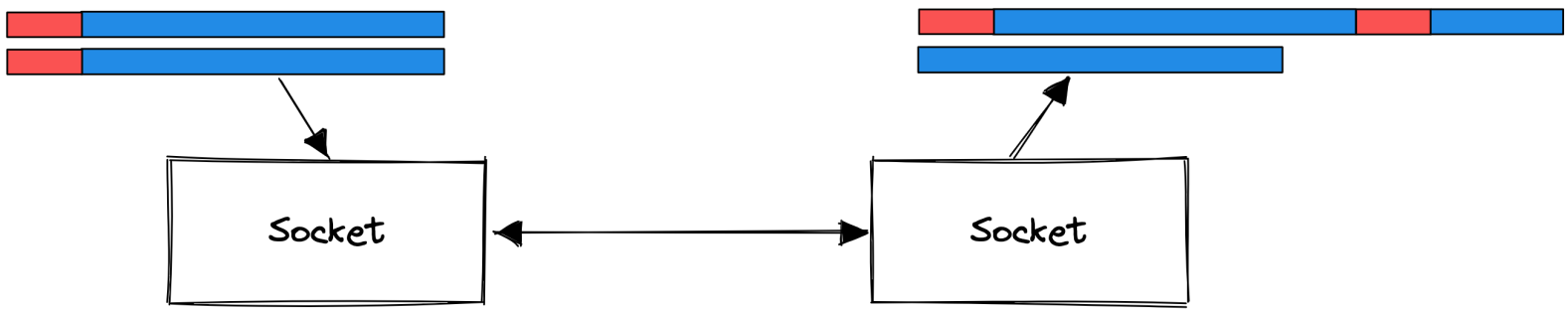
How do TCP sockets work



Handling socket messages in Node.js

```
socket.on('message', (data) => {
  // assumes data is one complete message
  handleMessage(data);
});
```

How do TCP sockets work



Handling socket messages in Node.js

```
socket.on('message', (data) => {
  // receive raw socket data, possible messages in many pieces
  receiveData(data, (message) => {
    // only pass one complete message to handleMessage at a time
    handleMessage(message);
  }
});
```

Handling socket messages in Node.js

```
let buffer = Buffer.from('');
let waitingForRestOfMessage = false;

const receiveData = (data) => {
  if (receivedPartialMessage()) {
    Buffer.concat(buffer, data);
    waitingForRestOfMessage = true;
  }
  ...
}
```

Handling socket messages in Node.js

- We’re tracking a lot of data (state)
- We’re moving from one state to another (idle, waiting for more data, etc.)
- This kind of sounds like a state machine

What is a state machine

> “A finite-state machine (FSM) or simply a state machine, is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the inputs that trigger each transition.”
>
> *–Wikipedia: Finite-state machine [https://en.wikipedia.org/wiki/Finite-state_machine]*

What is a state machine

Let’s take a look at some examples

Advantages of this approach

- We can only ever be in one known state at a time
- States are explicitly defined
- Transitions from one state to another are explicitly defined
- Easier to visualize than loops and variables

Advantages of this approach

- This is (mostly) what we actually use at Neo
- This code has processed millions of messages and transactions
- Billions of dollars worth of transactions
- Code has worked almost untouched for the past 2.5 years

Thanks for Listening!

Ian Sutherland

Architect, Head of DX and OSS at Neo Financial

Node.js Contributor, Tooling Working Group

@iansu

Thanks for Listening!

Ian Sutherland

Architect, Head of DX and OSS at Neo Financial

Node.js Contributor, Tooling Working Group

@iansu