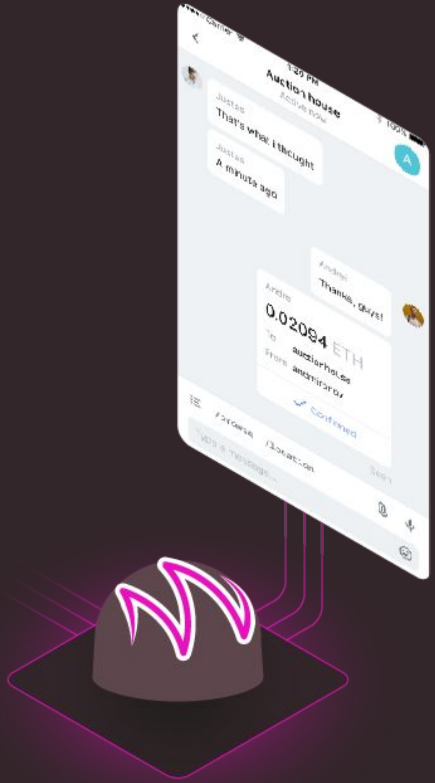
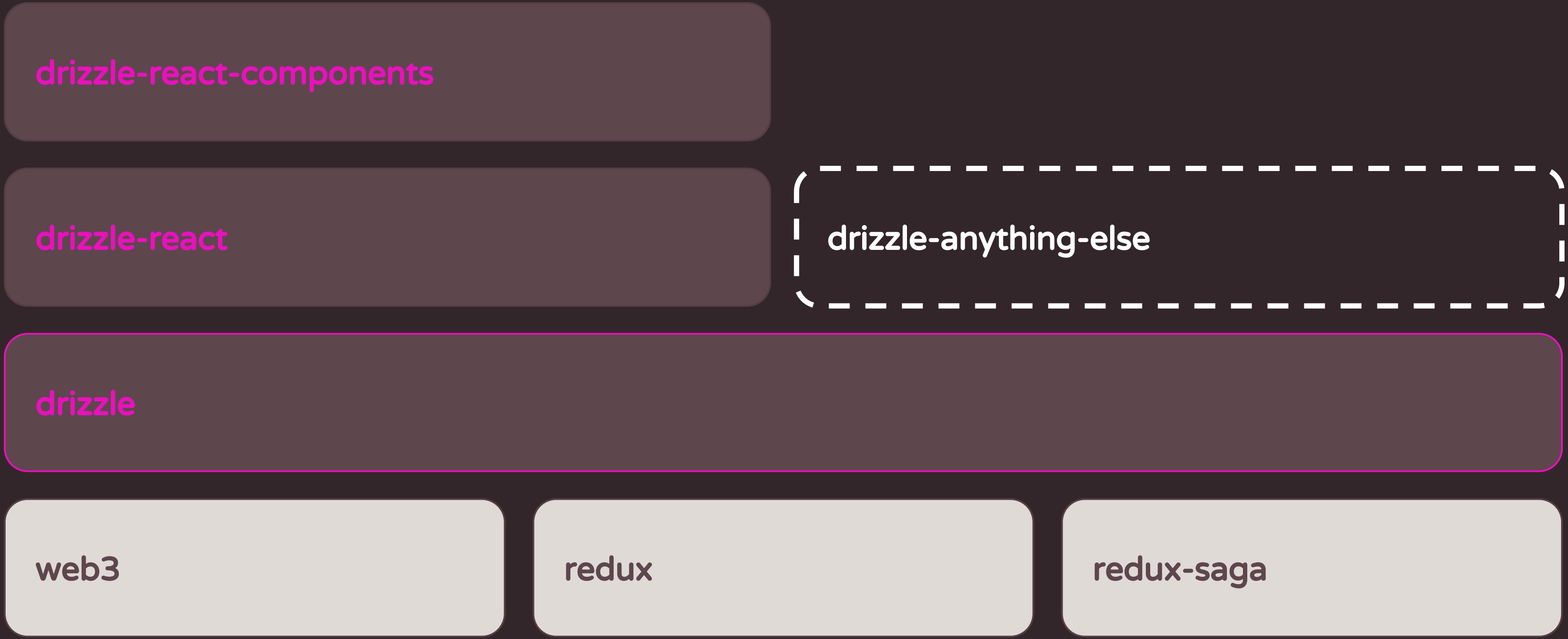


drizzle

Stateful Ethereum Datastore

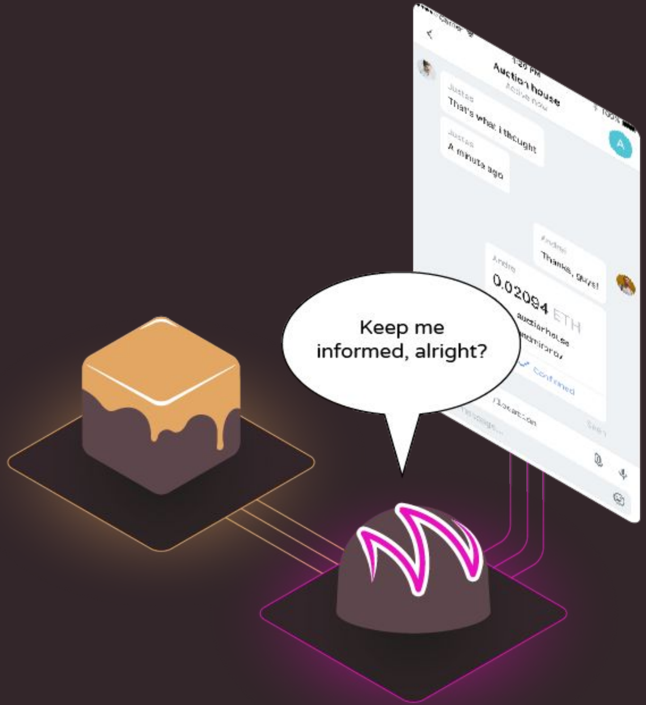
Modular


drizzle-react-components
drizzle-react
drizzle-anything-else
drizzle
web3
redux
redux-saga

The drizzle Family of Libraries

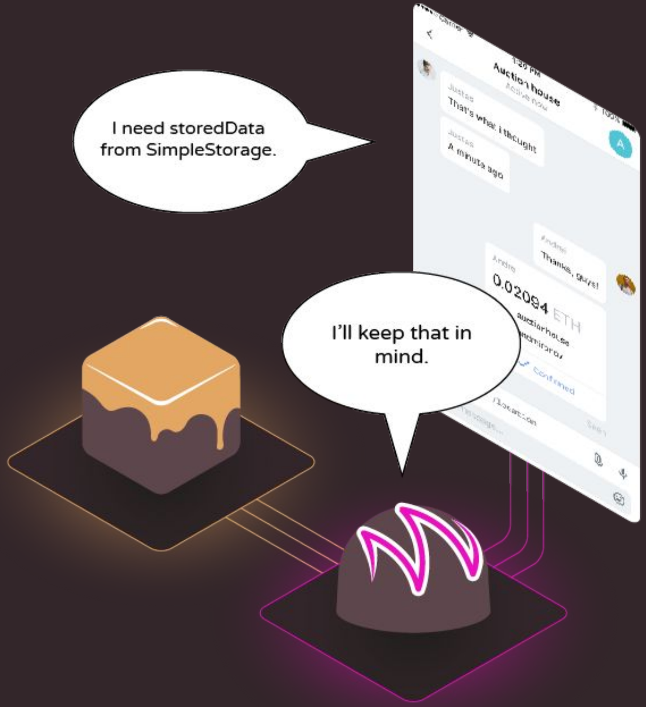
- `drizzle-react-components`
  - Library of useful components: dapps in minutes!
  - LoadingContainer, ContractData, ContractForm
- `drizzle-react`
  - Integrates drizzle with React: DrizzleProvider, drizzleConnect
- `drizzle`
  - Contains the store (generated or your own)
  - Contract, account and transaction state
  - Adds cacheCall, cacheSend functions to contract methods

Syncing: Step 1



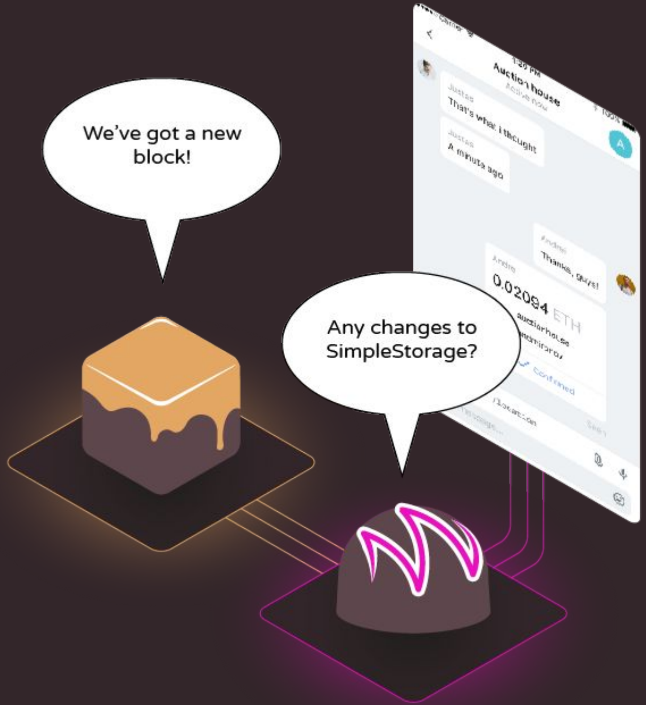
Once initialized, Drizzle instantiates web3 and our desired contracts, then observes the chain by subscribing to new block headers.

Syncing: Step 2



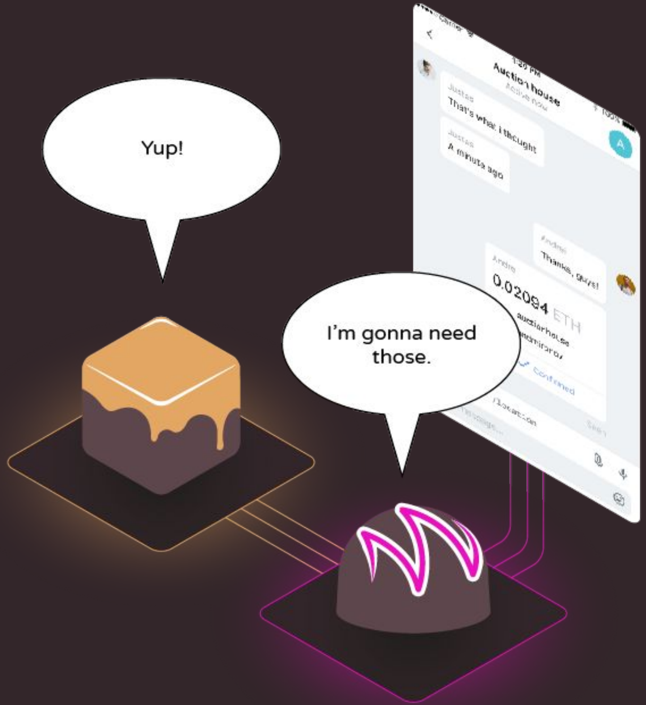
Drizzle keeps track of contract calls so it knows what to synchronize.

Syncing: Step 3



When a new block header comes in, Drizzle checks that the block isn't pending. Then, it goes through the transactions looking to see if any of them touched our contracts.

Syncing: Step 4



If they did, we replay the calls already in the store to refresh any potentially altered data. If they didn't we continue with the store data. Currently, this is granular down to the contract (moving toward granularity to the variable).

Initialization: We’ve Got Options

```
{
  contracts: [
    SimpleStorage,
    TutorialToken
  ],
  web3: {
    fallback: {
      type: 'ws',
      url: 'ws://127.0.0.1:8545'
    }
  }
}
```

cacheCall( )

```
var dataKey = drizzle.contracts.SimpleStorage.methods.storedData.cacheCall()
```

```
{this.props.contracts.SimpleStorage.storedData[dataKey].value}
```

```
SimpleStorage: {
  storedData: {
    0x0: {
      value: 1
    }
  }
}
```

```
TutorialToken: {
  balanceOf: {
    0xd3794bd...: {
      value: 11500
    },
    0xb4224ef...: {
      value: 500
    }
  }
}
```

cacheSend( )

```
var txHash = drizzle.contracts.SimpleStorage.methods.set.cacheSend(2)

{this.props.transactions[txHash].status} // pending, success, error
{this.props.transactions[txHash].confirmations}
{this.props.transactions[txHash].receipt}
{this.props.transactions[txHash].error}
```

What if it fails to broadcast and therefore gets no hash?

```
{this.props.transactionStack[id]}
```

cacheSend( ): Tx State

```
transactionStack: [
  ‘0xd3794bd...’
]
```

```
transactions: {
  0xd3794bd...: {
    status: ‘pending’,
    confirmations,
    receipt,
    error
  }
}
```

cacheSend( ): Contract State

```
SimpleStorage: {
  syncing: true
  storedData: {
    0x0: {
      value: 1
    }
  }
}
```

```
SimpleStorage: {
  syncing: false
  storedData: {
    0x0: {
      value: 2
    }
  }
}
```

Drizzle State

```
{
  accounts: {
    address: {
      balance
    }
  },
  contracts: {
    contractName: {
      events,
      initialized,
      synced,
      callerFunctionName: {
        argsHash: {
          args,
          value
        }
      }
    }
  },
  drizzleStatus: {
    initialized
  }
  transactionStack,
  transactions: {
    transactionHash: {
      confirmations,
      error,
      receipt,
      status
    }
  },
  web3: {
    status
  }
}
```