BLOCKCHAIN DECONSTRUCTUCTED: CONTRACTS VERSUS SMART CONTRACTS

FRITZ HENGLEIN
UNIVERSITY OF COPENHAGEN
DEON DIGITAL AG

UNSW Blockchain Symposium
2018-02-12/13

FRITZ HENGLEIN

- Professor of programming languages and systems
  - Foundations, techniques, algorithmics, language design
  - Enterprise systems, healthcare, finance, blockchain, contract management
- Head of Research, Deon Digital AG
- Member, European Blockchain Consortium (ebcc.eu)
- Director, Research center for high-performance computing for finance (HIPERFIT.dk)
- Steering committee chair, Innovation network for Finance IT (CFIR.dk)
- Mostly academic, some industrial lab/start-up experience

Technische Universität München, Rutgers University;
New York University, Utrecht University, University of Copenhagen, IT University of Copenhagen;
IBM Research, Cornell University, University of New South Wales, University of Oxford;
Hafnium Research ApS, Deon Digital AG/Deon Digital Denmark A/S, …

A CRASH SLIDE ON BLOCKCHAIN AND SMART CONTRACTS

SMART TERM	WHAT IT MEANS (ANNO 2018)
BLOCKCHAIN	**DECENTRALIZED** APPEND-ONLY EVENT LOG (LEDGER)
SMART CONTRACT (CODE)	CLASS (IN JAVA-LIKE LANGUAGE)
SMART CONTRACT (EXECUTING)	PROCESS (OBJECT [= CLASS INSTANCE])
SMART MESSAGES	**INFORMATION** TRANSMISSION (ORDINARY MESSAGES) **RESOURCE TRANSFERS**

A CRASH SLIDE ON BLOCKCHAIN AND SMART CONTRACTS

SMART TERM	WHAT IT MEANS (ANNO 2018)
BLOCKCHAIN	**DECENTRALIZED** APPEND-ONLY **RESOURCE TRANSFER** LOG (LEDGER)
SMART CONTRACT (CODE)	CLASS (IN JAVA-LIKE LANGUAGE)
SMART CONTRACT (EXECUTING)	PROCESS (OBJECT [= CLASS INSTANCE])
SMART MESSAGES	INFORMATION TRANSMISSION (ORDINARY MESSAGES) **RESOURCE TRANSFERS**

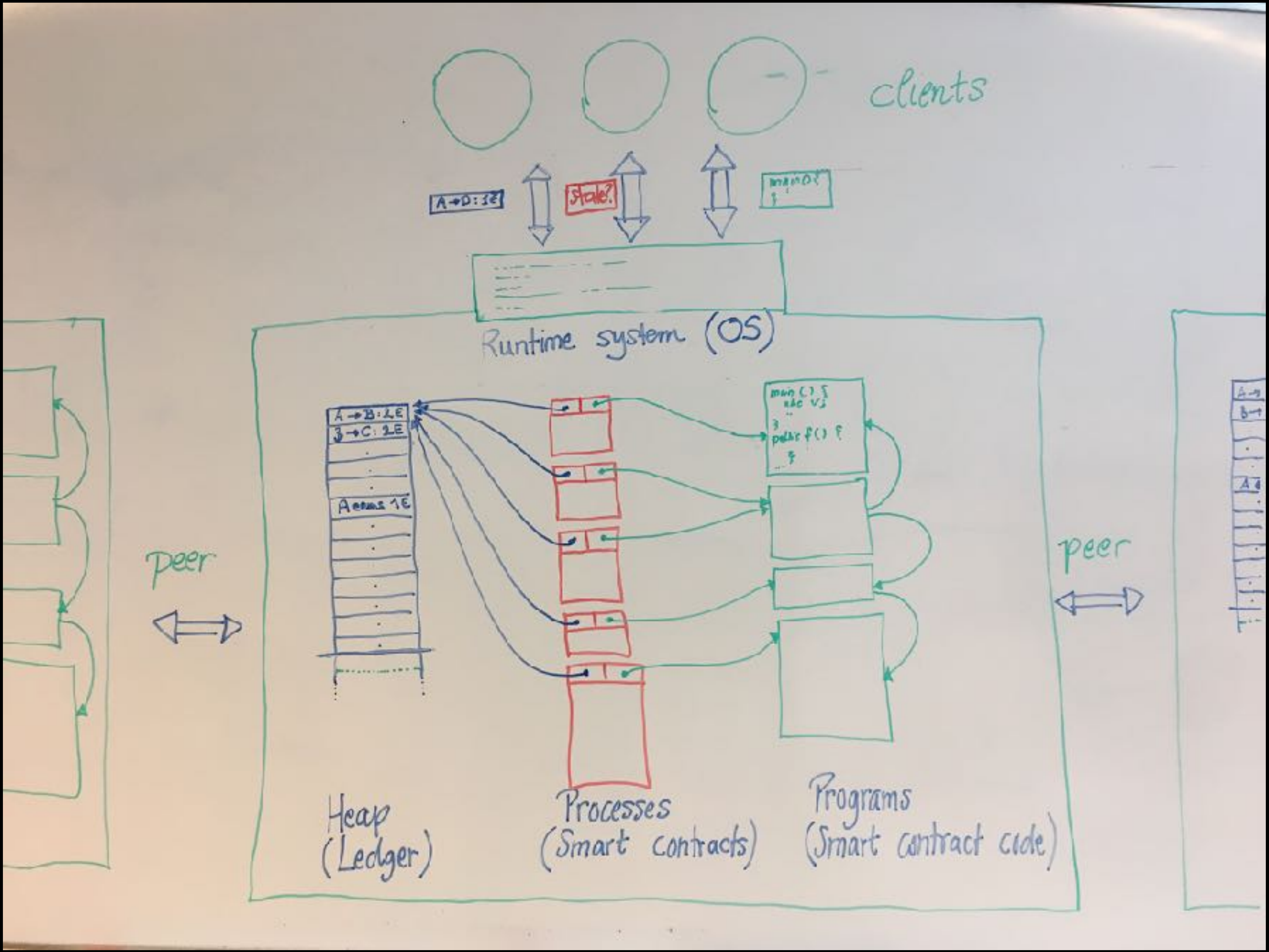
OBSERVATIONS

- Usually, only resource **balance** required for validating future events
  - Tamper-proof transfer log used for verification of balance **only**
  - Assuming infinite credit line, **all** resource transfers **commute:** order is insignificant

CONSENSUS ON LINEARIZATION NOT NECESSARY!

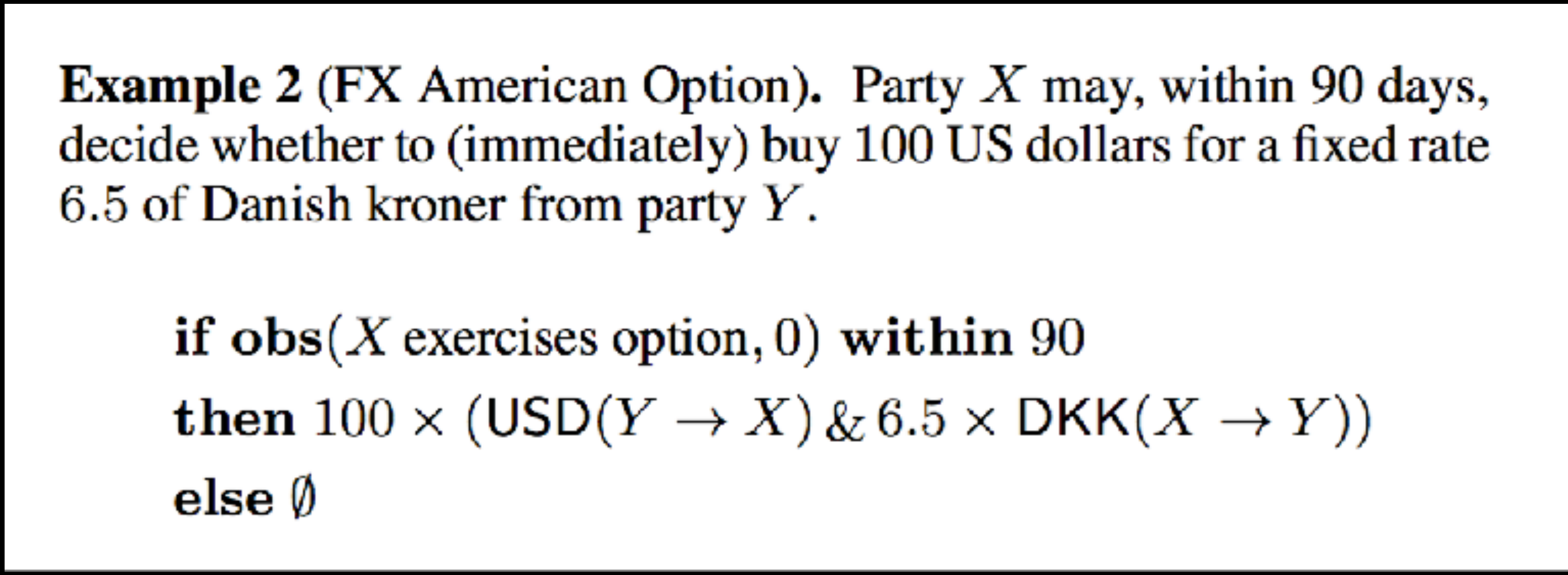
- To prevent forging, resource transfer **requires**
  - evidence of adequate resource cost by sender **or**
  - leakage of (some) information about resource transfer to a third party

BLOCKCHAIN SYSTEM ARCHITECTURE



CONTRACT VERSUS SMART CONTRACT

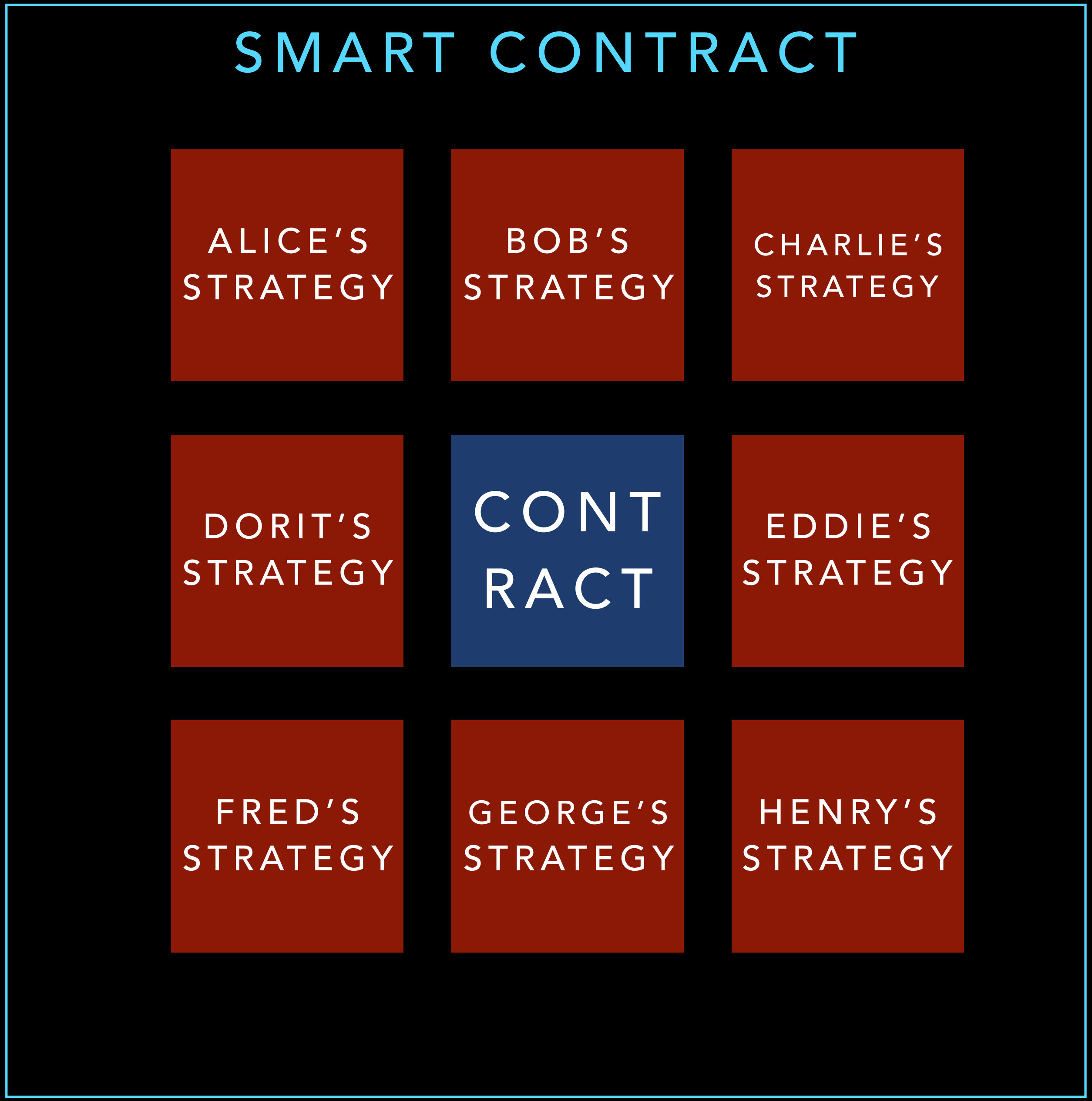
Contract:
Obligations and permissions
(rules)



Example 2 (FX American Option). Party X may, within 90 days, decide whether to (immediately) buy 100 US dollars for a fixed rate 6.5 of Danish kroner from party Y.

$$\begin{array}{l}\textbf{if obs}(X \text{ exercises option}, 0)\ \textbf{within}\ 90\\ \textbf{then}\ 100 \times (\mathsf{USD}(Y \to X)\ \&\ 6.5 \times \mathsf{DKK}(X \to Y))\\ \textbf{else}\ \emptyset\end{array}$$

CONTRACT VERSUS SMART CONTRACT

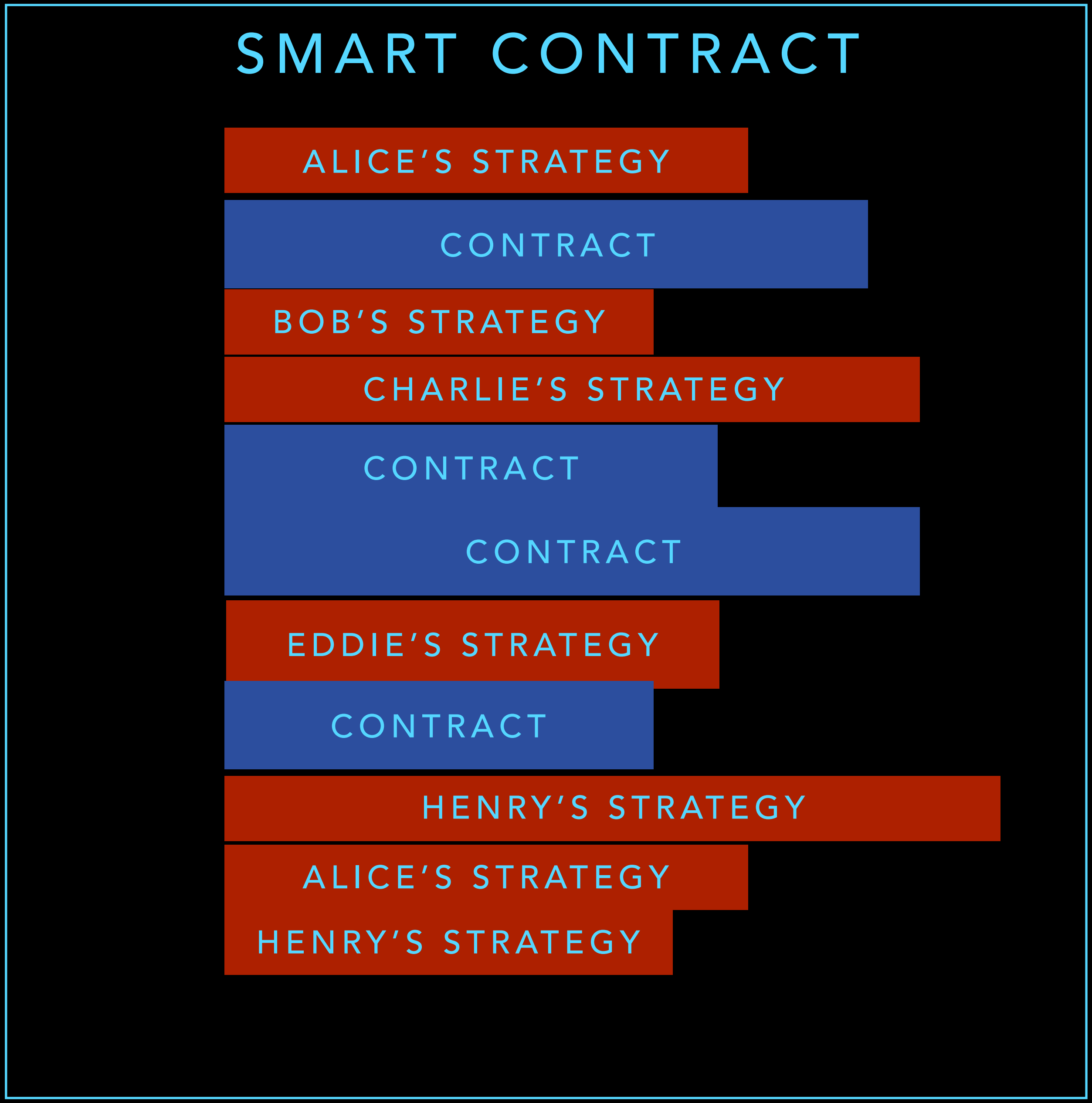


Contract:
Obligations and permissions (rules)

Strategy:
A single party's actions (actions)

Smart contract:
Rules and all parties' codified actions intermixed

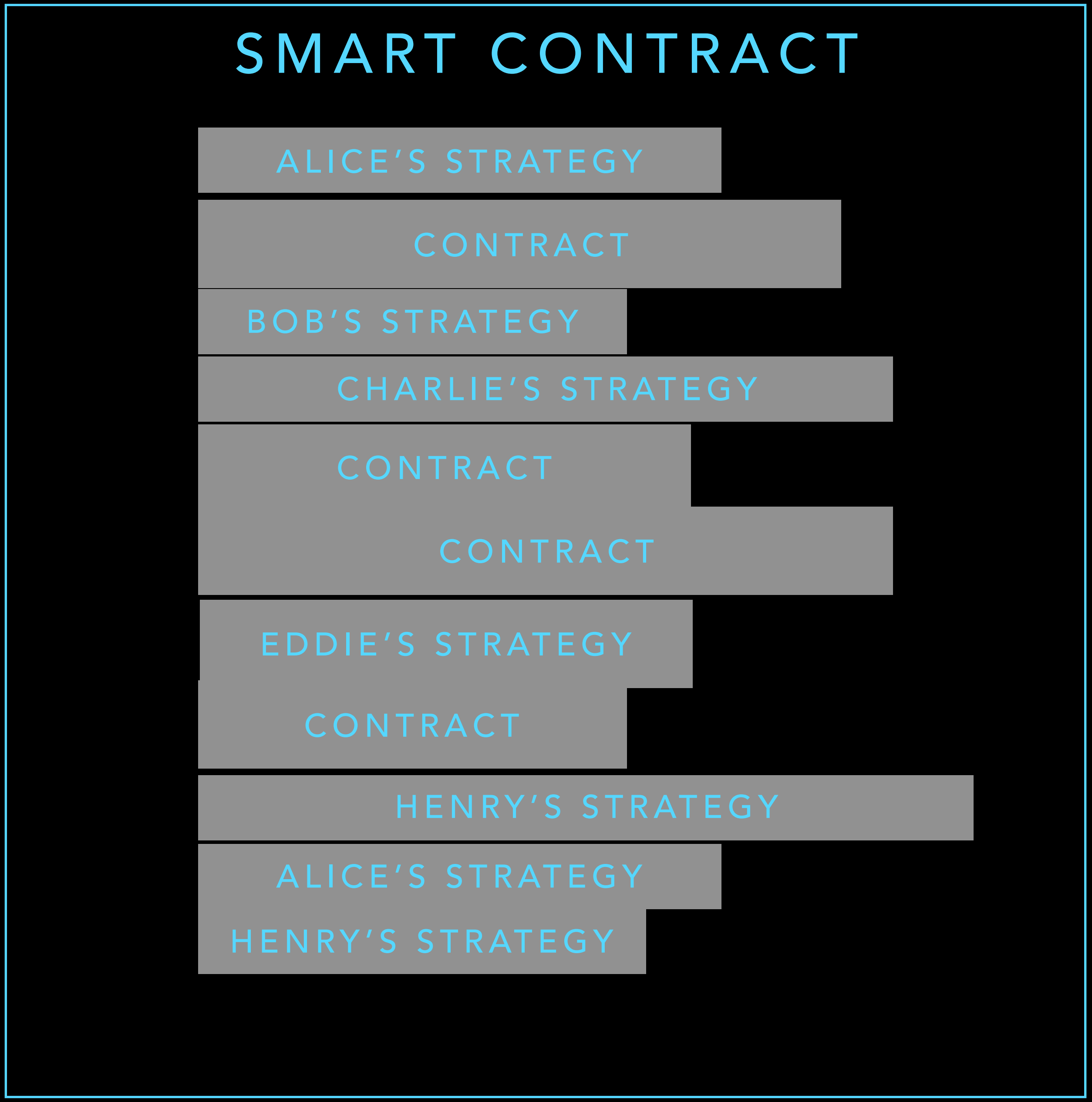
CONTRACT VERSUS SMART CONTRACT



Actually…

Contract checking and actions (strategy) mixed together in the source code

CONTRACT VERSUS SMART CONTRACT

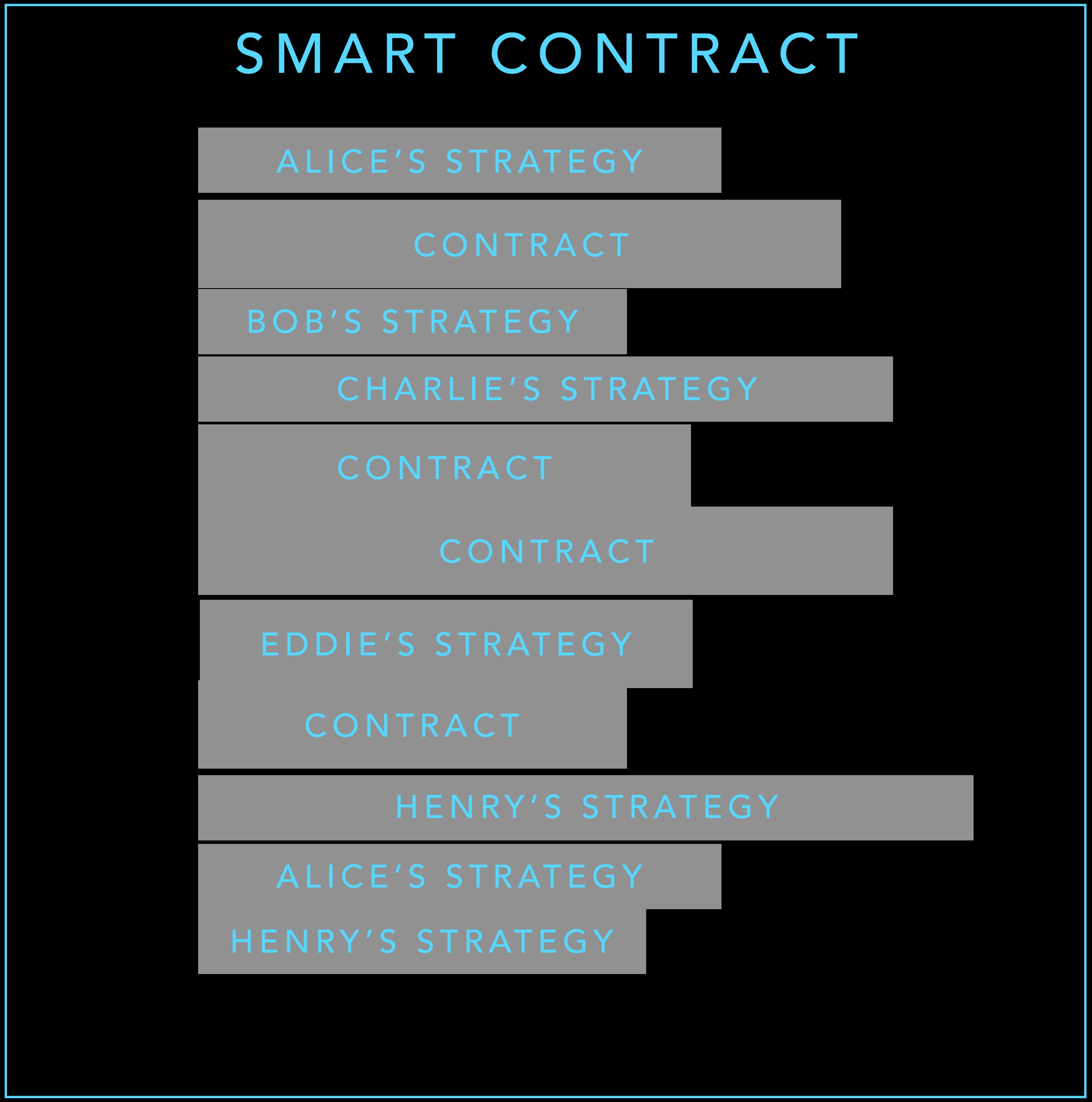


Actually…

Contract checking and actions (strategy) mixed together in the source code

and one cannot even see which is which

CONTRACT VERSUS SMART CONTRACT

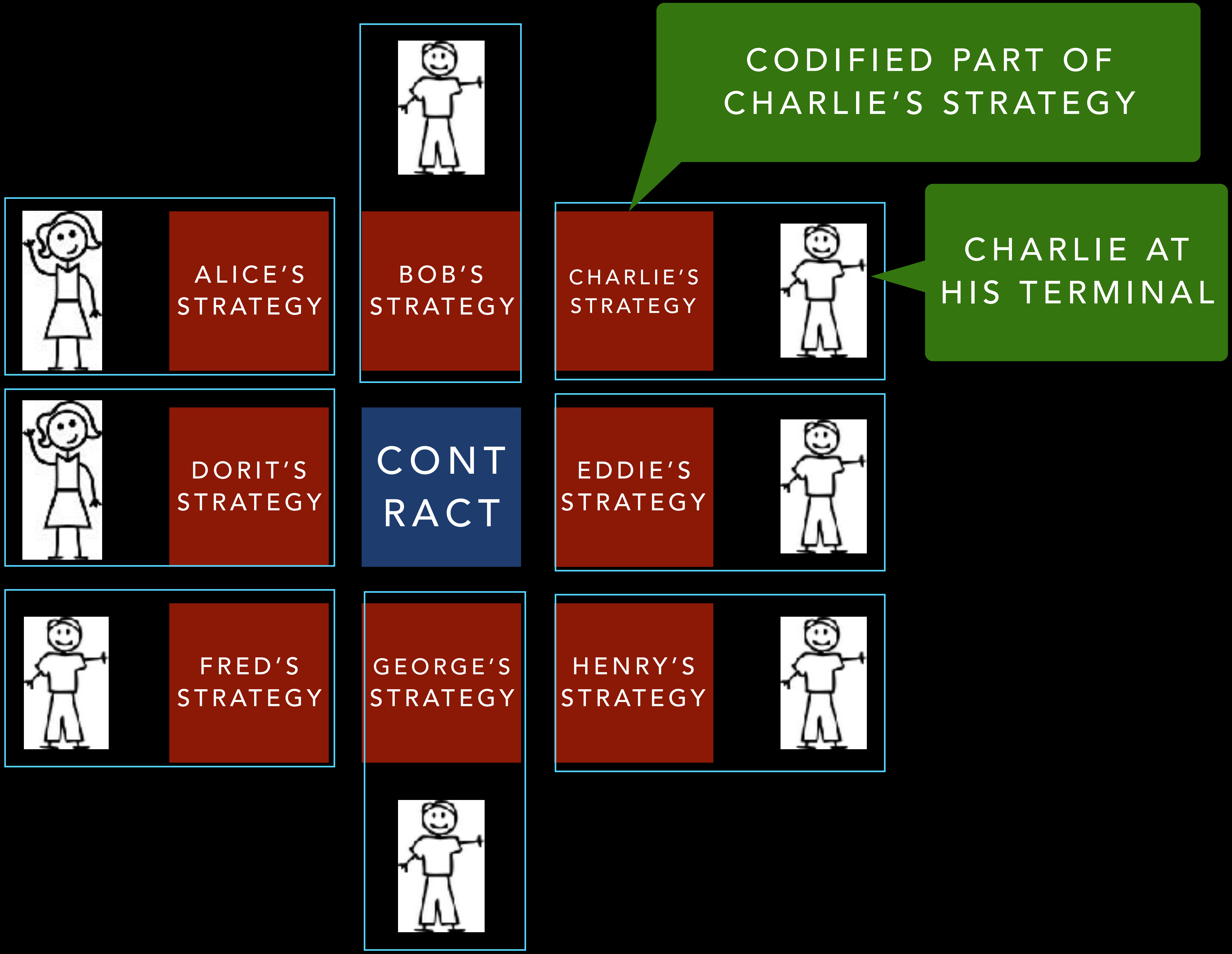


What is the **contract** and what is **strategy**?

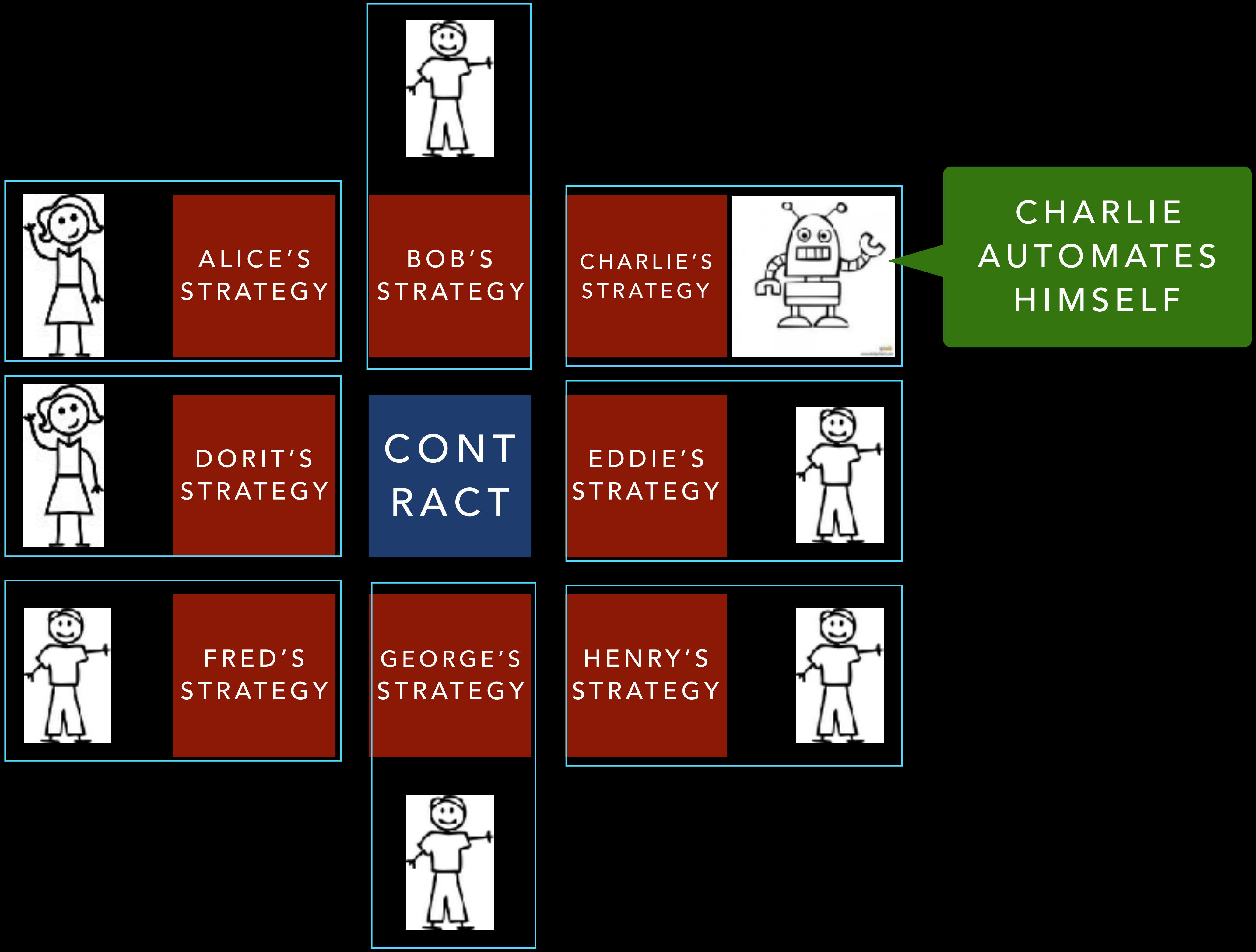
How do you **compose** contracts (by themselves)?

How do you **analyze** contracts?

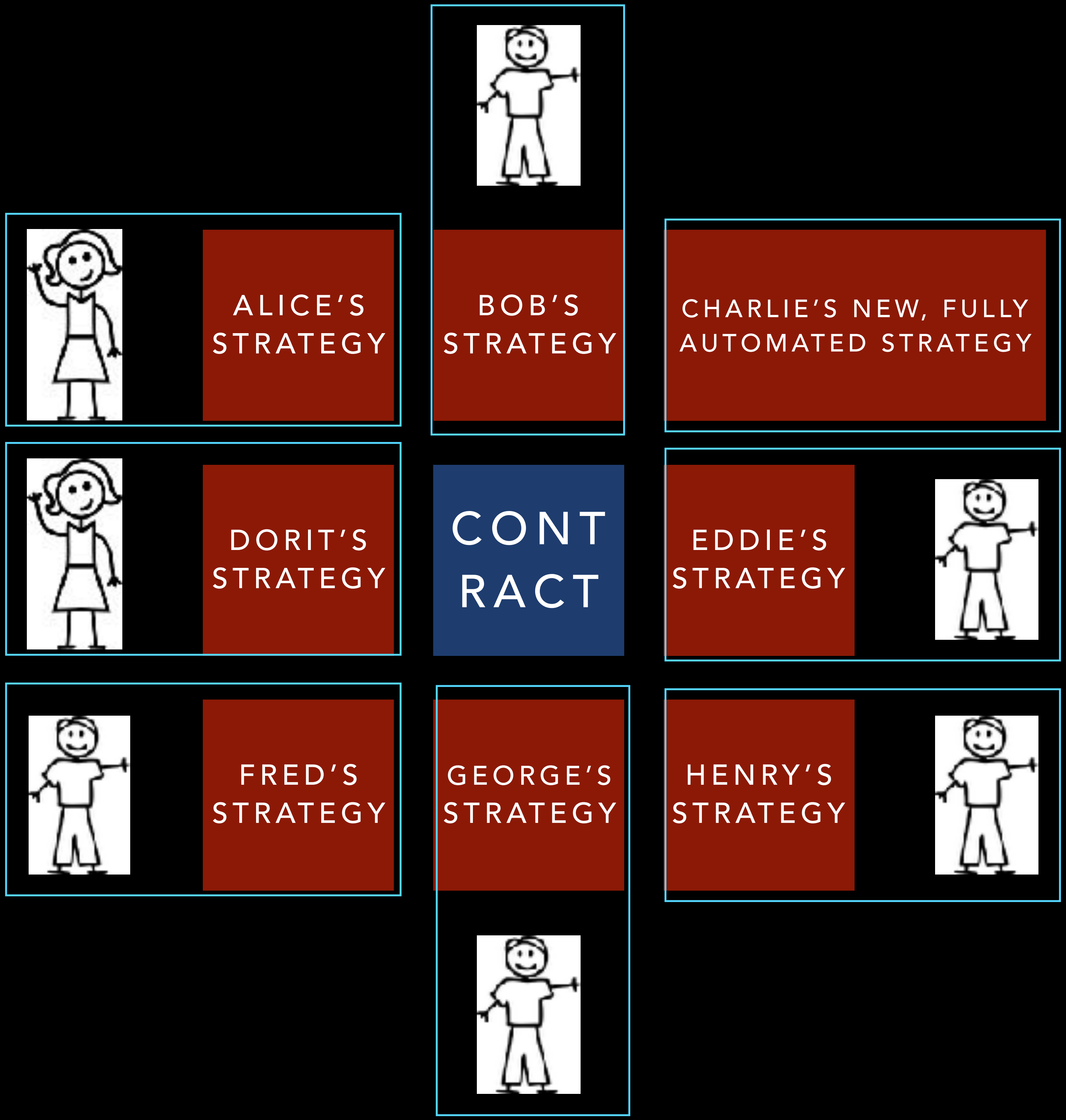
CONTRACT VERSUS SMART CONTRACT



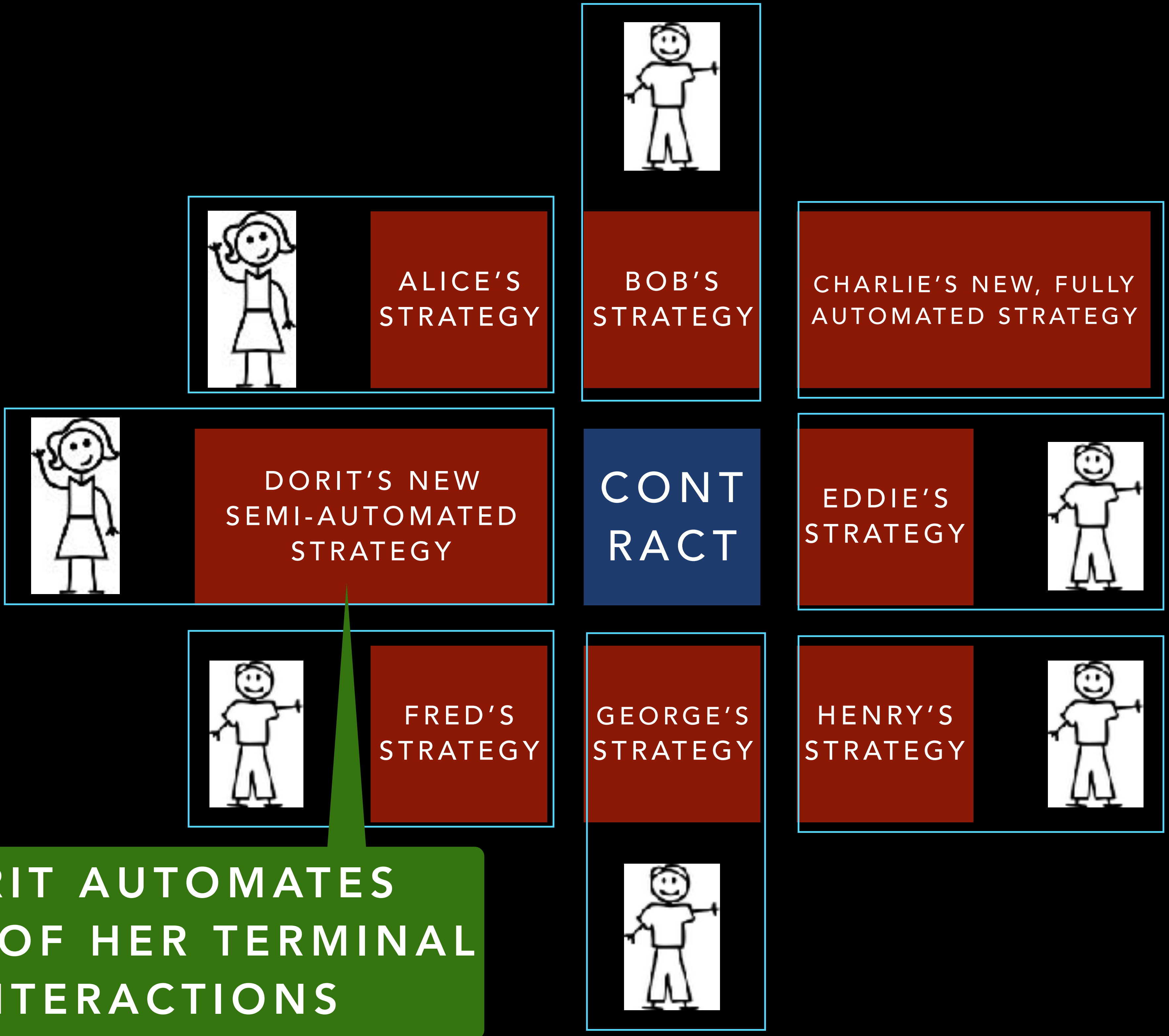
CONTRACT VERSUS SMART CONTRACT



CONTRACT VERSUS SMART CONTRACT

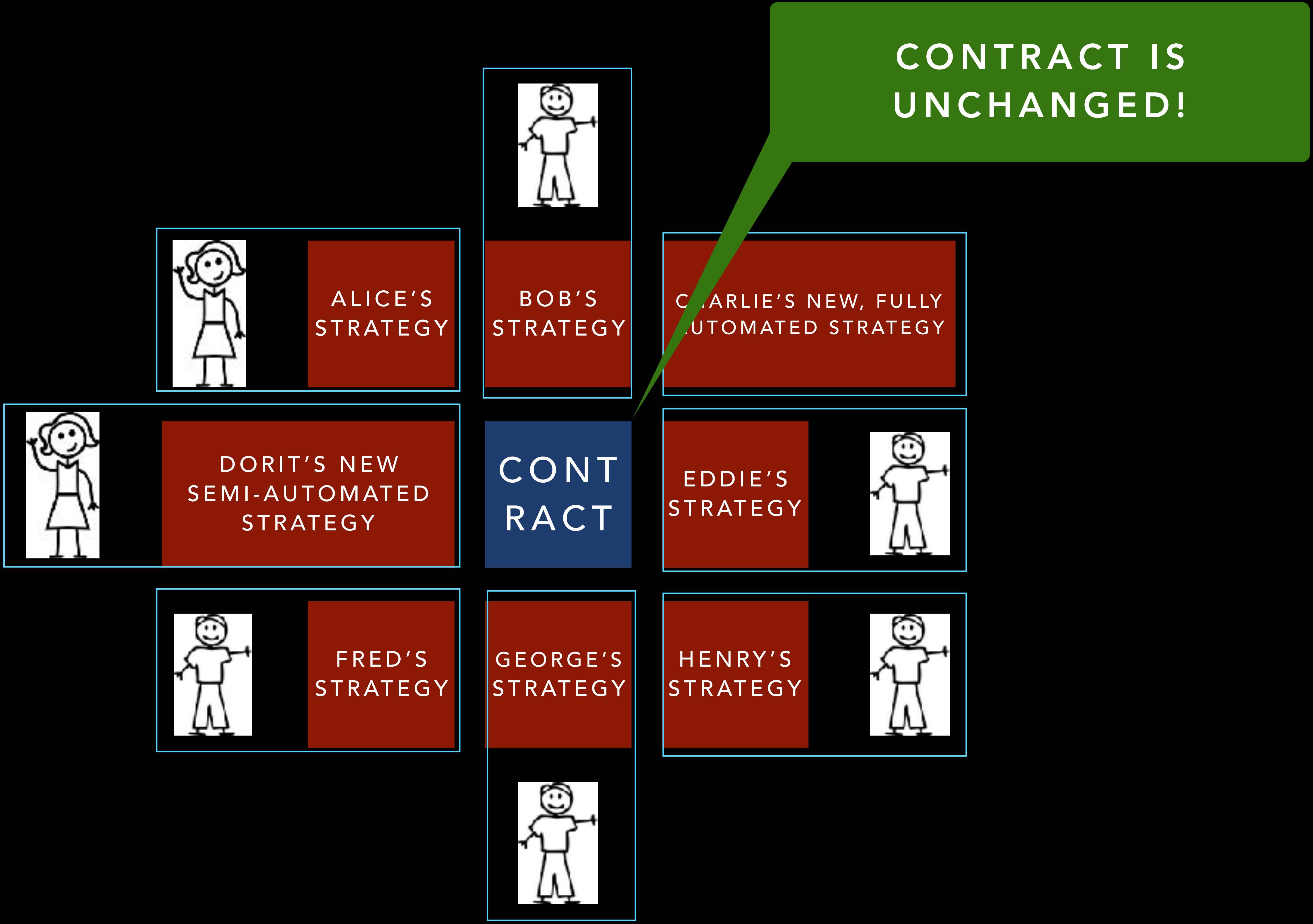


CONTRACT VERSUS SMART CONTRACT

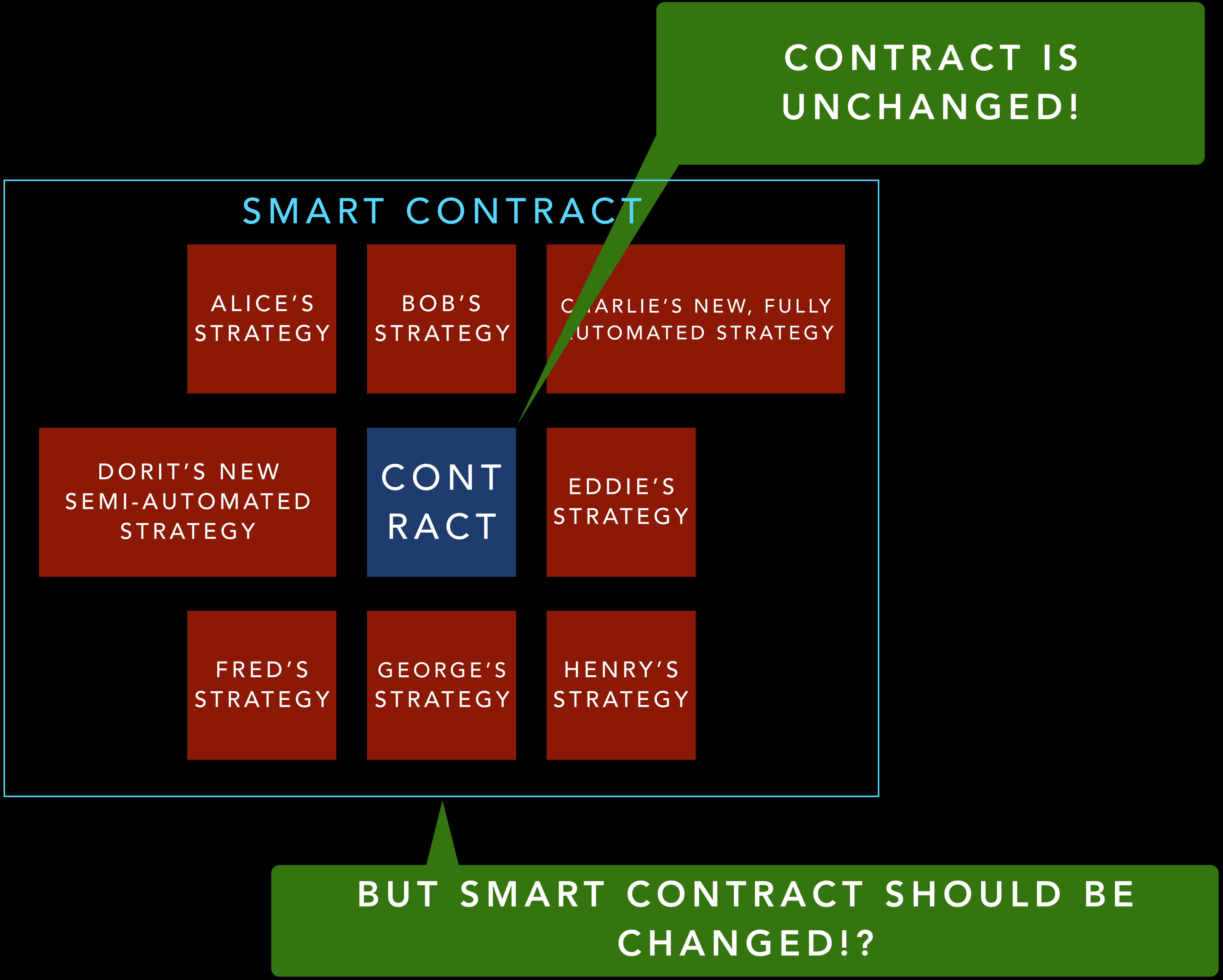


DORIT AUTOMATES PARTS OF HER TERMINAL INTERACTIONS

CONTRACT VERSUS SMART CONTRACT

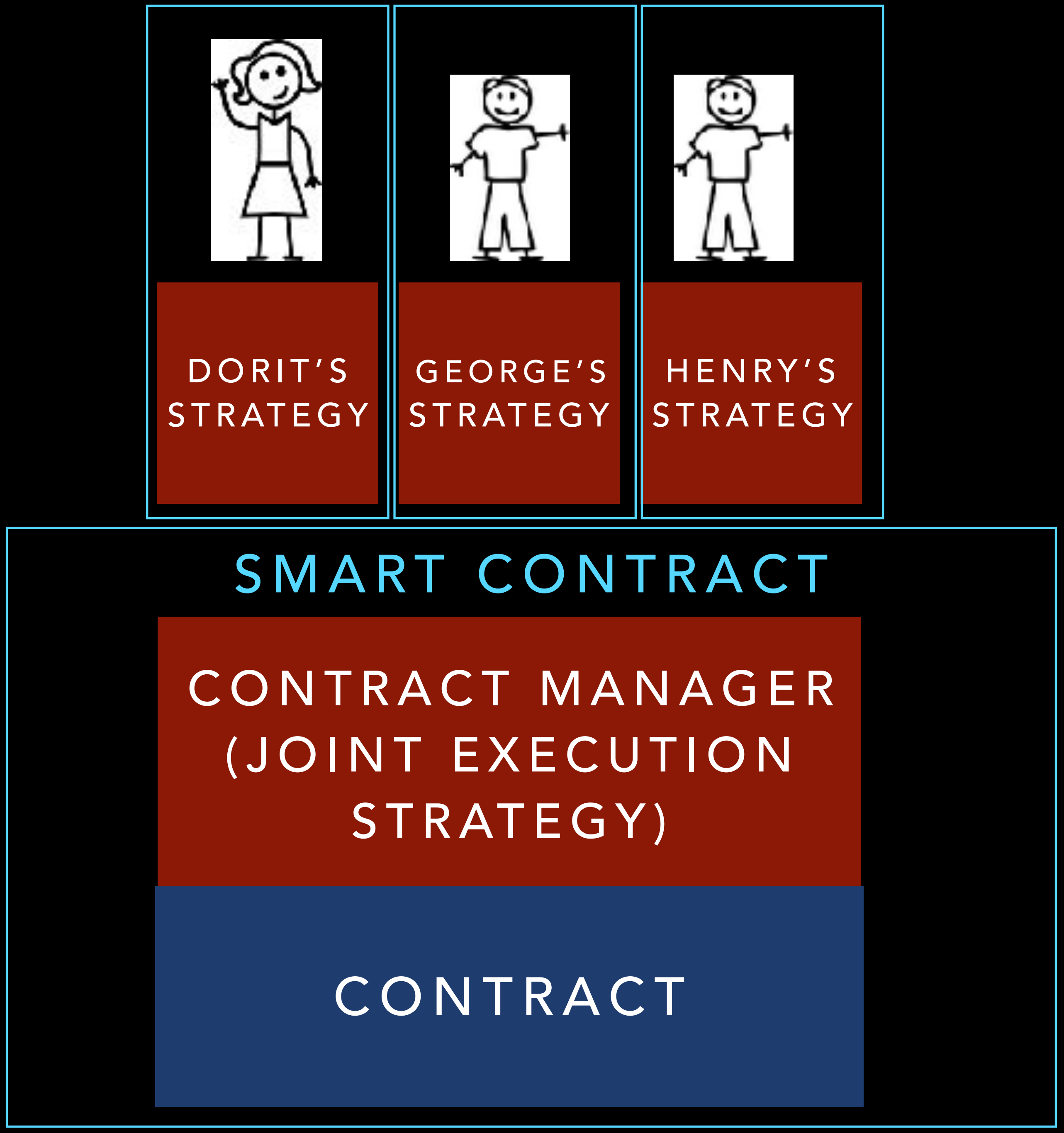


CONTRACT VERSUS SMART CONTRACT



MANAGED CONTRACT = CONTRACT + JOINT EXECUTION STRATEGY

Egelund-Müller, Elsman, Henglein, Ross, *Automated Execution of Financial Contracts on Blockchains*, BISE 2017



THE PRICE OF EXPRESSIVENESS: RICE'S THEOREM

Rice (1953)

- Smart contract: usually program, written in **Turing-complete** programming language (Ethereum, Corda, Fabric, …)
  - **+** : Expressive
  - **-** : **Undecidable properties** even with *full access to the source code*
    - Smart contracts are ultimately **unanalyzable**

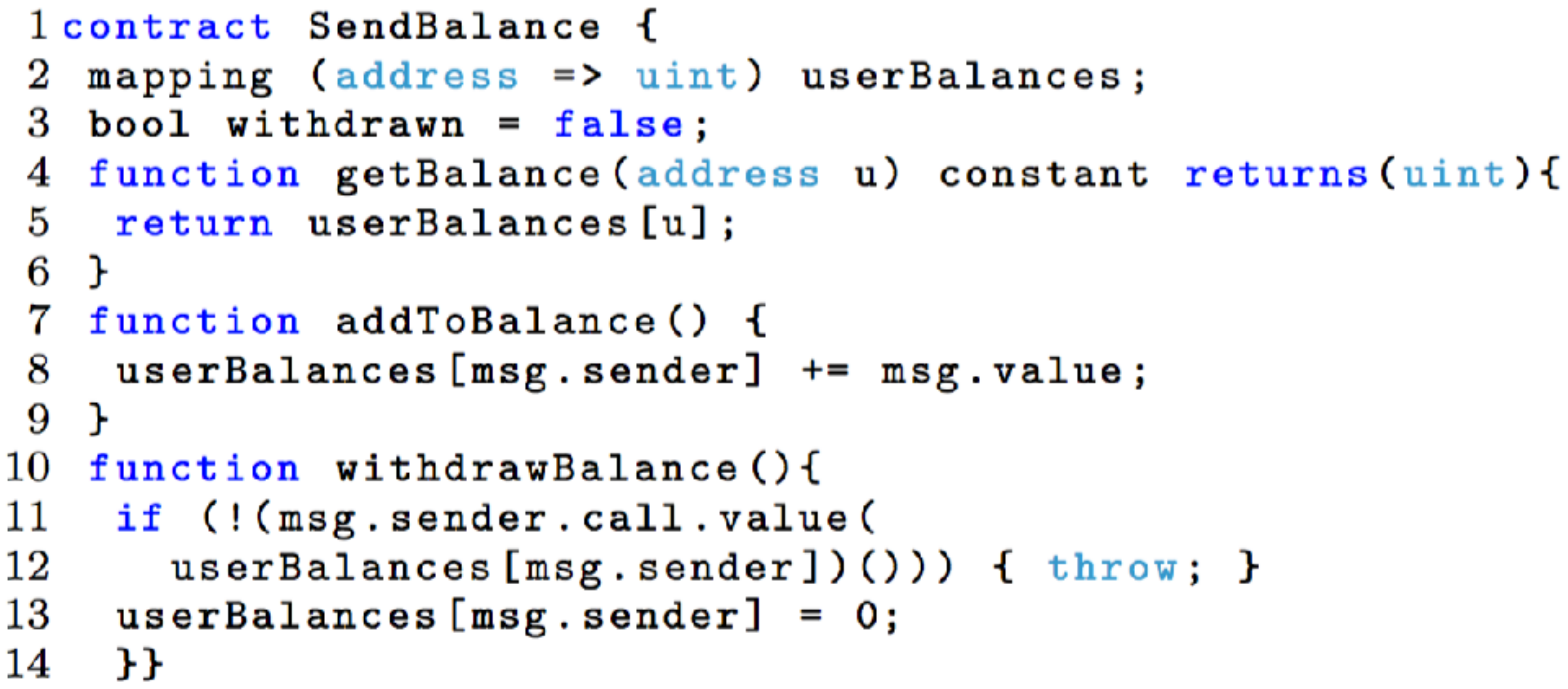
ETHEREUM VULNERABILITIES

LUU, CHU, OLICKEL, SAXENA, HOBOR, MAKING SMART CONTRACTS SMARTER (2016)

- Transaction-order dependence: Messages may have different effect depending on their order of arrival
    - Who controls the process scheduler (= message sequencer)? Some *miner*: Front-running
- Time-stamp dependence: Smart contracts may have different executions depending on the time stamp on a transaction block
    - Who controls the time stamping of transaction blocks? Some miner: Clock manipulation
- Exception handling, gas management fragility: Subtle differences in exception semantics, limited run-time stack
    - Provoking out-of-stack and gas exhaustion exceptions: Any user
- Programming language subtleties:
    - Exception handling subtleties (send vs. call)
    - Reentrancy vulnerability (DAO hack)
    - Implicit method forwarding (multi-sig exploit)

REENTRANCY VULNERABILITY

LUU, CHU, OLICKEL, SAXENA, HOBOR, MAKING SMART CONTRACTS SMARTER (2016)



```
contract SendBalance {
 mapping (address => uint) userBalances;
 bool withdrawn = false;
 function getBalance(address u) constant returns(uint){
  return userBalances[u];
 }
 function addToBalance() {
  userBalances[msg.sender] += msg.value;
 }
 function withdrawBalance(){
  if (!(msg.sender.call.value(
    userBalances[msg.sender])())) { throw; }
  userBalances[msg.sender] = 0;
  }}
```

Figure 7: An example of the reentrancy bug. The contract implements a simple bank account.

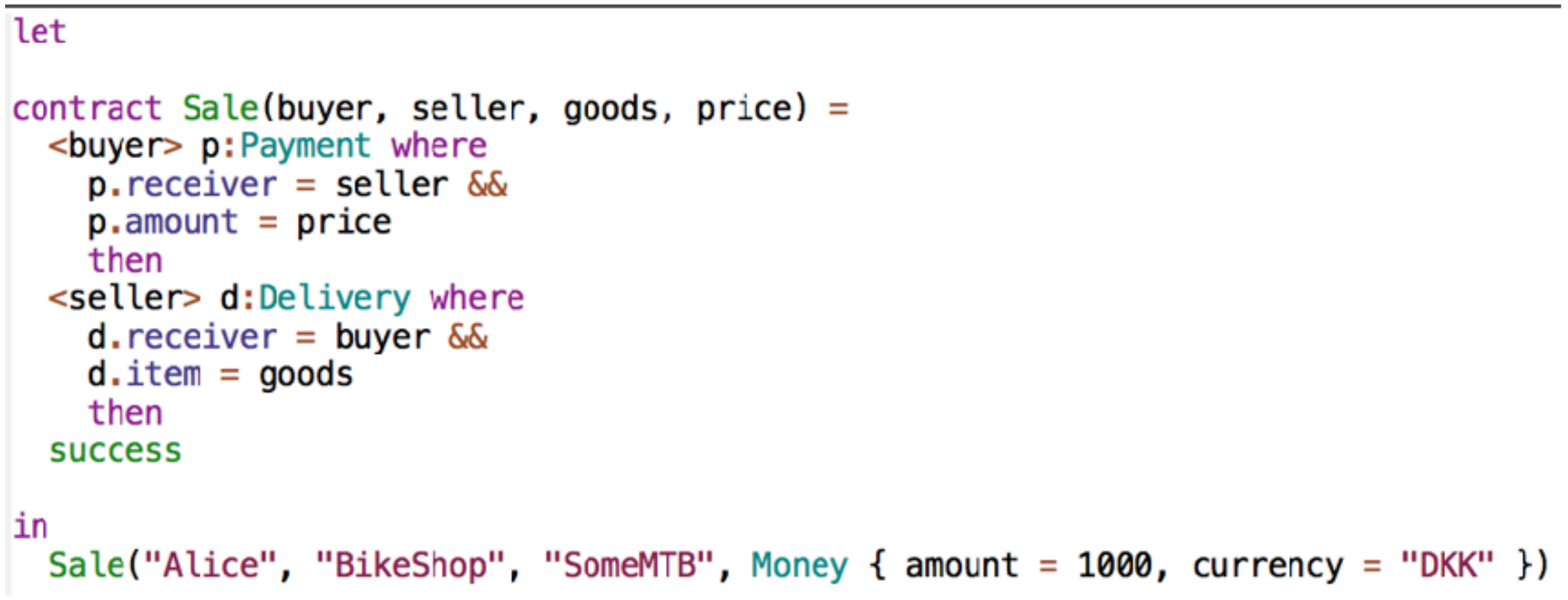
SMART CONTRACTS ARE NEITHER

- Smart contracts = self-executing contracts (programs) in complex Turing-complete programming language
  - Rules and actions intermixed:
    Not contracts
  - Hard to analyze, low-level programs:
    Not smart
- **Proposal:** Managed contract = (contract, strategy)

SEPARATED!

Compositional formal contracts

Andersen, Elsborg, Henglein, Stefansen, Simonsen (2006)

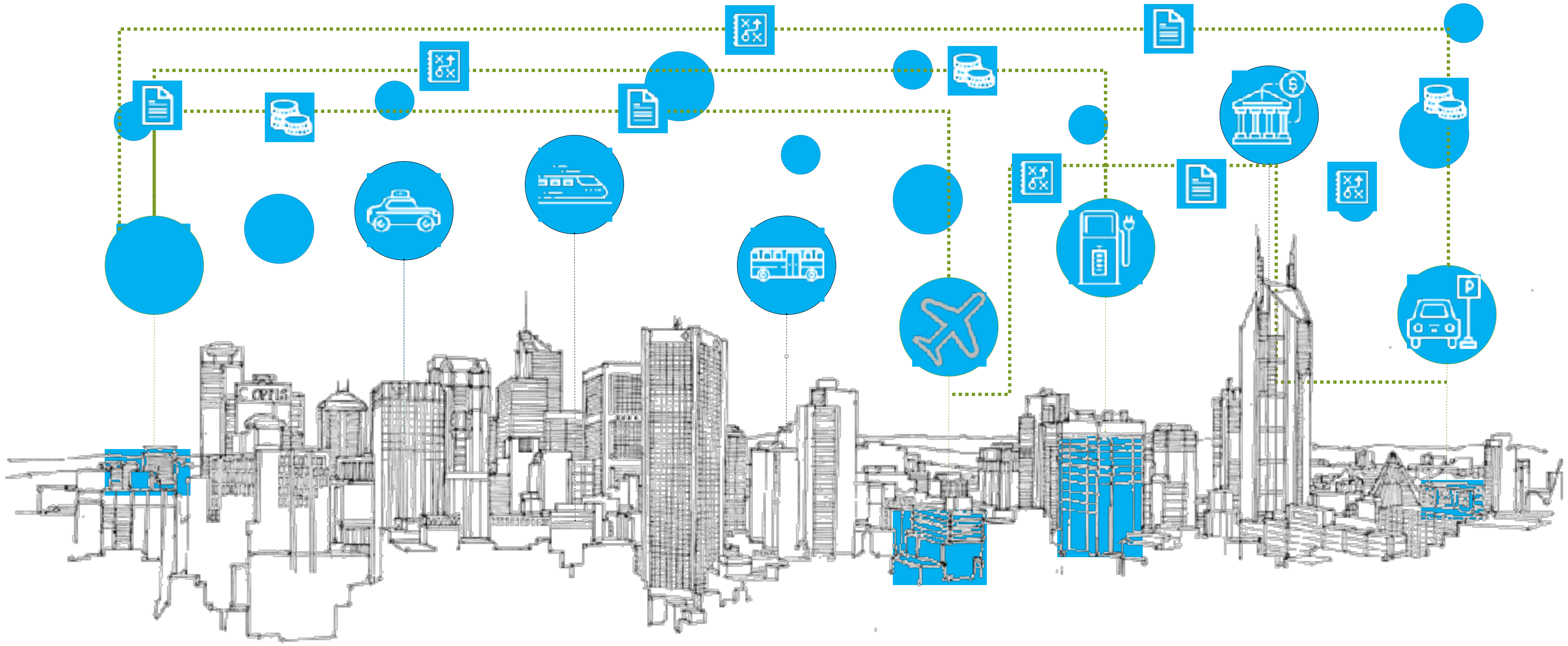


```
let

contract Sale(buyer, seller, goods, price) =
  <buyer> p:Payment where
    p.receiver = seller &&
    p.amount = price
    then
  <seller> d:Delivery where
    d.receiver = buyer &&
    d.item = goods
    then
  success

in
  Sale("Alice", "BikeShop", "SomeMTB", Money { amount = 1000, currency = "DKK" })
```

Deon Digital CSL (2017)

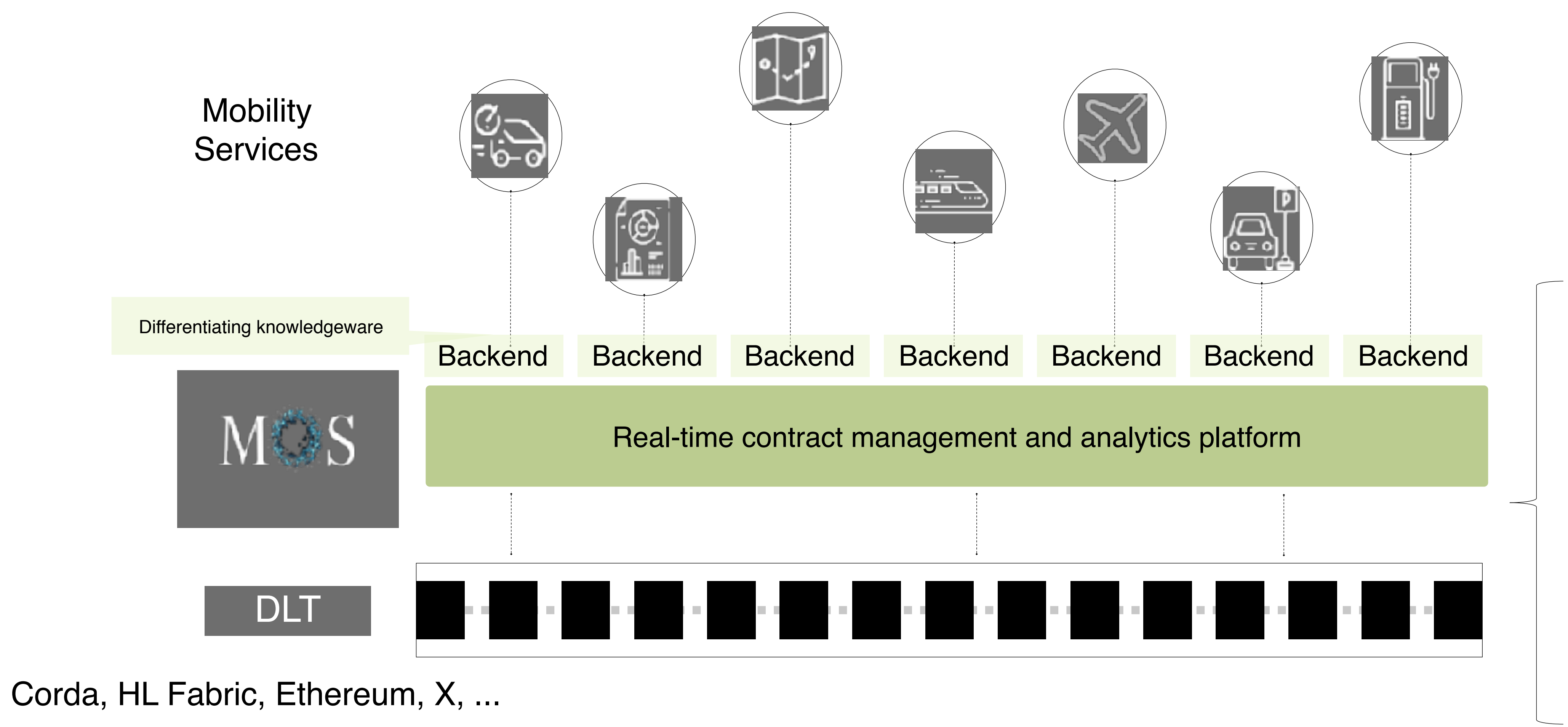
- Separation of concerns: Obligations and permissions, no self-executing actions
- Domain-oriented: No computer-oriented coding
- Analyzable
- Composable

Decentralized yet integrated mobility – how?



Mobility Operating System ©2017

Mobility Operating System (MOS)

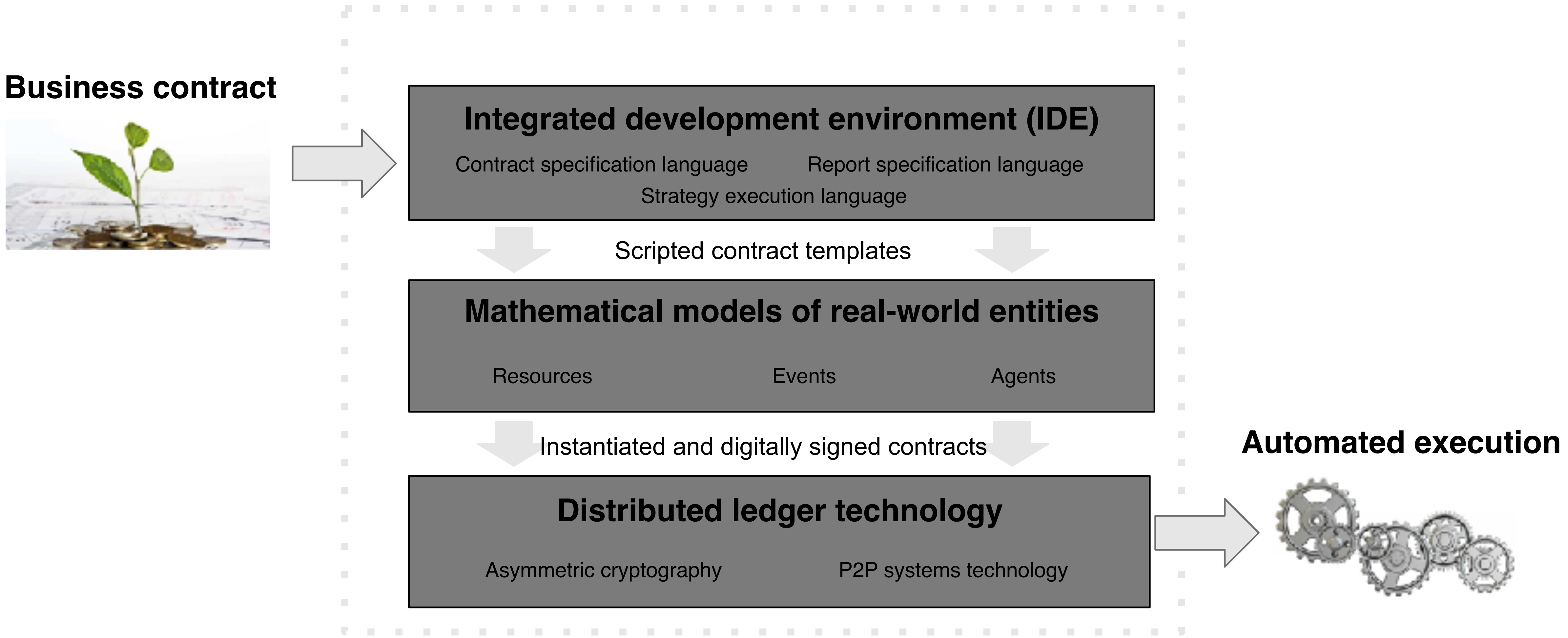

Mobility Services
Differentiating knowledgeware
Backend
Backend
Backend
Backend
Backend
Backend
Backend
MOS
Real-time contract management and analytics platform
DLT
Corda, HL Fabric, Ethereum, X, ...

- Domain Specific Language for contract specfication
- Contracts composed from reusable libraries
- Contract states and execution/monitoring on distributed ledger
- Trust and privacy: implemented according to business needs

Mobility Operating System ©2017

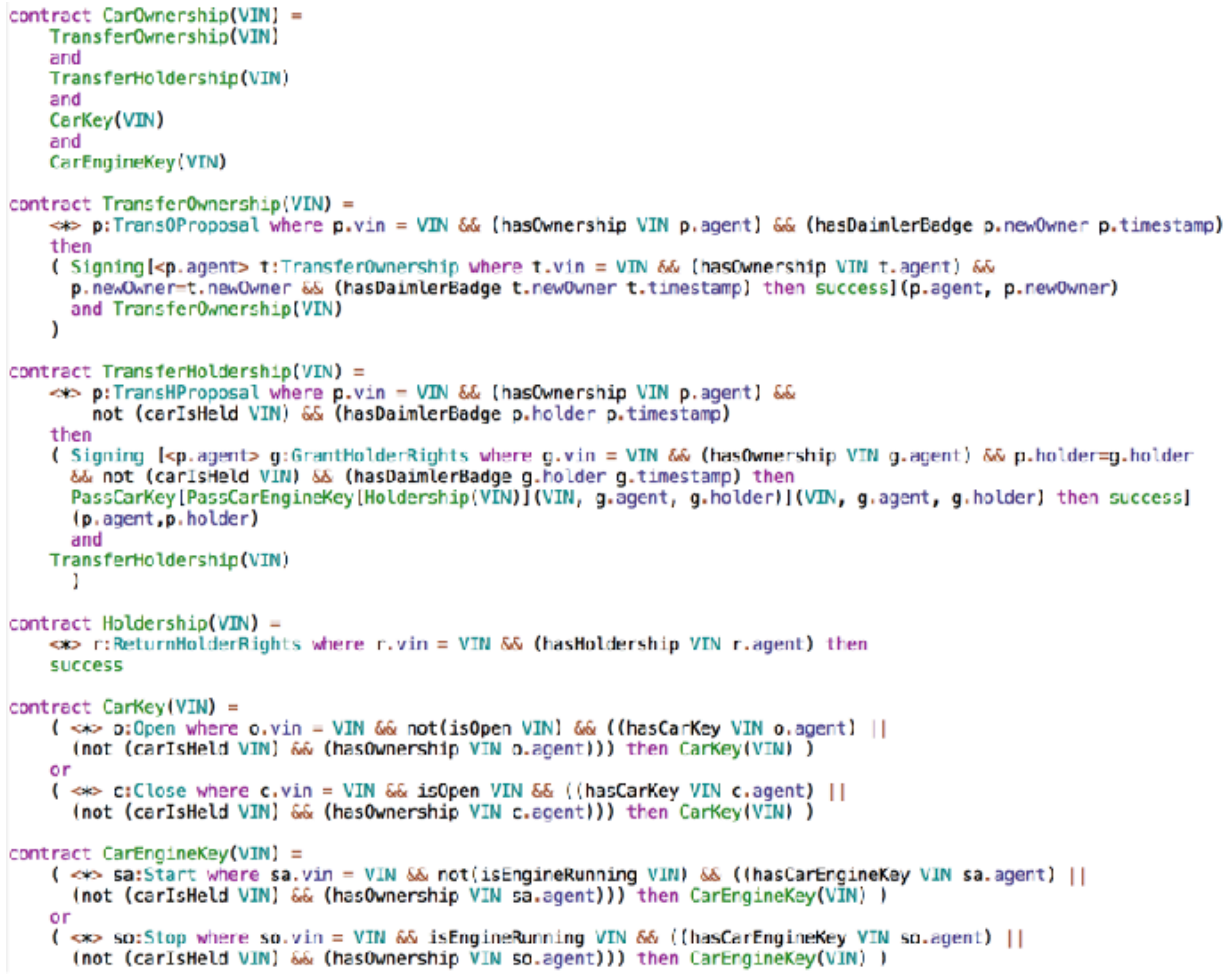
Courtesy of DEONDIGITAL

Rapid contract deployment without conventional coding


Business contract
Integrated development environment (IDE)
Contract specification language
Report specification language
Strategy execution language
Scripted contract templates
Mathematical models of real-world entities
Resources
Events
Agents
Instantiated and digitally signed contracts
Distributed ledger technology
Asymmetric cryptography
P2P systems technology
Automated execution

Courtesy of Deon Digital and Mercedes-Benz

MOS contract (example)



```
contract CarOwnership(VIN) =
    TransferOwnership(VIN)
    and
    TransferHoldership(VIN)
    and
    CarKey(VIN)
    and
    CarEngineKey(VIN)

contract TransferOwnership(VIN) =
    <*> p:TransOProposal where p.vin = VIN && (hasOwnership VIN p.agent) && (hasDaimlerBedge p.newOwner p.timestamp)
    then
    ( Signing[<p.agent> t:TransferOwnership where t.vin = VIN && (hasOwnership VIN t.agent) &&
      p.newOwner=t.newOwner && (hasDaimlerBadge t.newOwner t.timestamp) then success](p.agent, p.newOwner)
      and TransferOwnership(VIN)
    )

contract TransferHoldership(VIN) =
    <*> p:TransHProposal where p.vin = VIN && (hasOwnership VIN p.agent) &&
        not (carIsHeld VIN) && (hasDaimlerBadge p.holder p.timestamp)
    then
    ( Signing [<p.agent> g:GrantHolderRights where g.vin = VIN && (hasOwnership VIN g.agent) && p.holder=g.holder
      && not (carIsHeld VIN) && (hasDaimlerBadge g.holder g.timestamp) then
      PassCarKey[PassCarEngineKey[Holdership(VIN)](VIN, g.agent, g.holder)](VIN, g.agent, g.holder) then success]
      (p.agent,p.holder)
      and
    TransferHoldership(VIN)
      )

contract Holdership(VIN) =
    <*> r:ReturnHolderRights where r.vin = VIN && (hasHoldership VIN r.agent) then
    success

contract CarKey(VIN) =
    ( <*> o:Open where o.vin = VIN && not(isOpen VIN) && ((hasCarKey VIN o.agent) ||
      (not (carIsHeld VIN) && (hasOwnership VIN o.agent))) then CarKey(VIN) )
    or
    ( <*> c:Close where c.vin = VIN && isOpen VIN && ((hasCarKey VIN c.agent) ||
      (not (carIsHeld VIN) && (hasOwnership VIN c.agent))) then CarKey(VIN) )

contract CarEngineKey(VIN) =
    ( <*> sa:Start where sa.vin = VIN && not(isEngineRunning VIN) && ((hasCarEngineKey VIN sa.agent) ||
      (not (carIsHeld VIN) && (hasOwnership VIN sa.agent))) then CarEngineKey(VIN) )
    or
    ( <*> so:Stop where so.vin = VIN && isEngineRunning VIN && ((hasCarEngineKey VIN so.agent) ||
      (not (carIsHeld VIN) && (hasOwnership VIN so.agent))) then CarEngineKey(VIN) )
```

From: Daimler internal car sharing system

BLOCKCHAIN SYSTEM PARAMETERS

- Performance
- Availability
- Partition tolerance
- Security
- Privacy
- Expressiveness
- Analyzability

Inherent trade-offs

$\Rightarrow$

No blockchain system to rule them all

$\Rightarrow$

Need for programmable/composable blockchain systems

OBSERVATIONS AND MUSINGS

- **Blockchain/DLT** = Persistent data structure containing immutable and mutable data that
    - is organizationally and technically **decentralized** and
    - guarantees that represented (“tokenized”) resources are neither lost nor duplicated: **linearity.**
- **Smart contract** (p.t.): arbitrary unstoppable program written in complicated Turing-complete programming language with irrevocable power of attorney to manage your bank account. *Somebody* is smart here, but who?
- **Global consensus** on **particular** linear sequence of events not necessary — but popular blockchain/DLT-systems implement it. Why?

MORE INFORMATION

- hiperfit.dk: Functional high-performance computing for finance
    - Domain-specific languages for **compositional and verifiable contracts**

GOING LIVE ANY TIME NOW...

- plan-x.org: Functional programming language technology for **high-performance blockchain systems**

FUNCTIONAL PROGRAMMING = PROGRAMMING WITH ~~IMMUTABLE~~ TAMPER-PROOF DATA

EUROPEAN BLOCKCHAIN CENTER

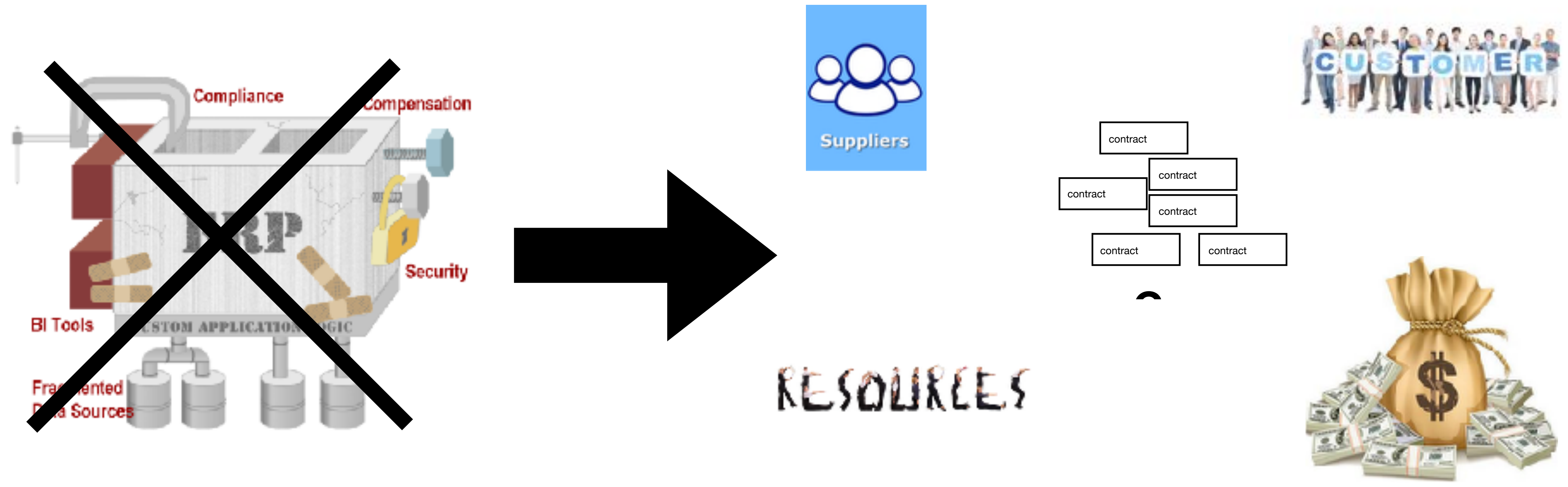
An initiative by:

IT University of Copenhagen

CBS COPENHAGEN BUSINESS SCHOOL HANDELSHØJSKOLEN

DEONDiGiTAL

Freeing business from legacy



Describe your business, not your IT systems

Thank you!

Information:

diku.dk
ebcc.eu
deondigital.com

Contact:

henglein@diku.dk
info@deondigital.com
mobility-os@daimler.com