Vincent Danos (ENS-PSL, CNRS, INRIA)
Ilias Garnier (ENS-PSL)
Julien Prat (CREST, CNRS)

# off-chain consensus for real-world smart contracts

twitter:@vee3my

http://www.tokenomics2019.org

http://www.di.ens.fr/~danos/iowb.html

# 1. an example of smart contract

**Axa's fizzy**

# Axa's fizzy contract

insurance contract for delayed flights

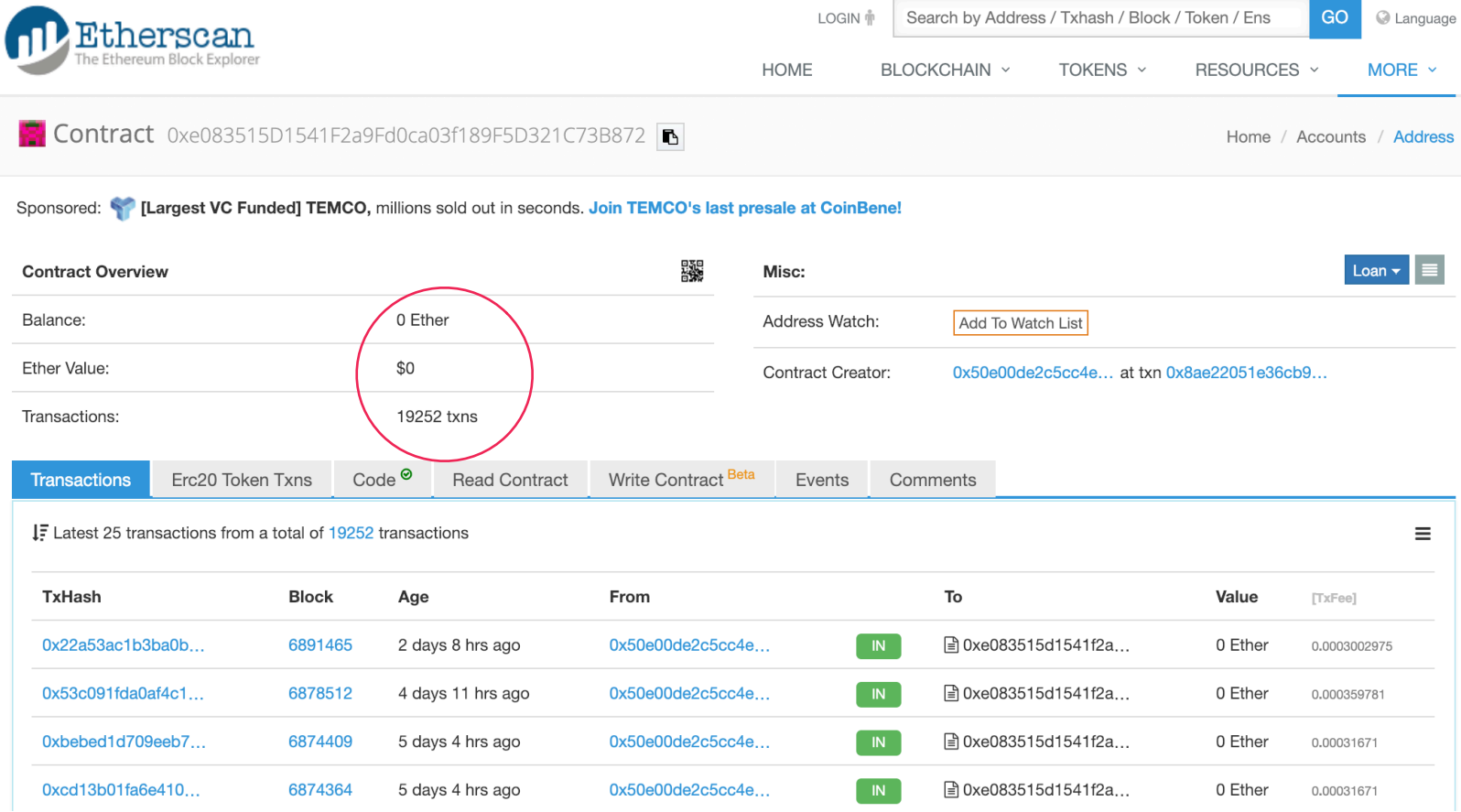
what it does (functionwise):

- logs (immutably) the succession of events per product
  - open_file [client buys insurance for flight xxx off-chain]
  - input_arrival_time of flight xxx [settlement off-chain]

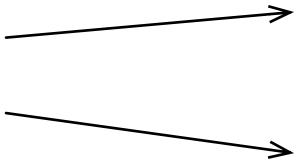
- no plausible deniability of logged items

- autonomy: commitment of decision process

it really “exists”



what it should be doing really!
- actual money transfers
- independent oracle(s)
- end-to-end evidential force



no PSP trust + cost

trustable sources for TAs
certification of pipeline to code

what it could be doing:
algorithmic provisioning for refunding
cf **etherisc** project

semantic boxing of governance
huge unseen problem

stable coin against risk of exchange

tax and consumer protection regulation

# how much can a consumer make of the immutable diary

- open_file [client buys insurance for flight xxx off-chain]

PB0 owner does not record the opening of a contract in the first place?

A: transfer of premium conditioned on the contract being opened

can only be done if the premium money goes to the contract in the first place

- input_arrival_time of flight xxx

PB1 owner records the wrong time

A: info is public and can be contested in court

PB2 owner never updates the flight status?

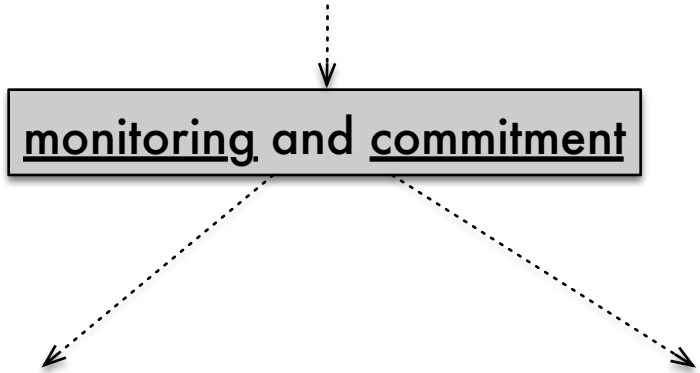
A: should be a timeout clause in the contract that transfers the agreed sum to the consumer in that case (now burden of updating the status rests on the owner of the contract).

with all of the above with have complete legal resource for client
we will say that the contract has perfect monitoring

# 2. the Szabo value equation

# The economic value of smart contracts

1. automate business agreements
2. allow players to avoid paying hidden costs due to potential litigation

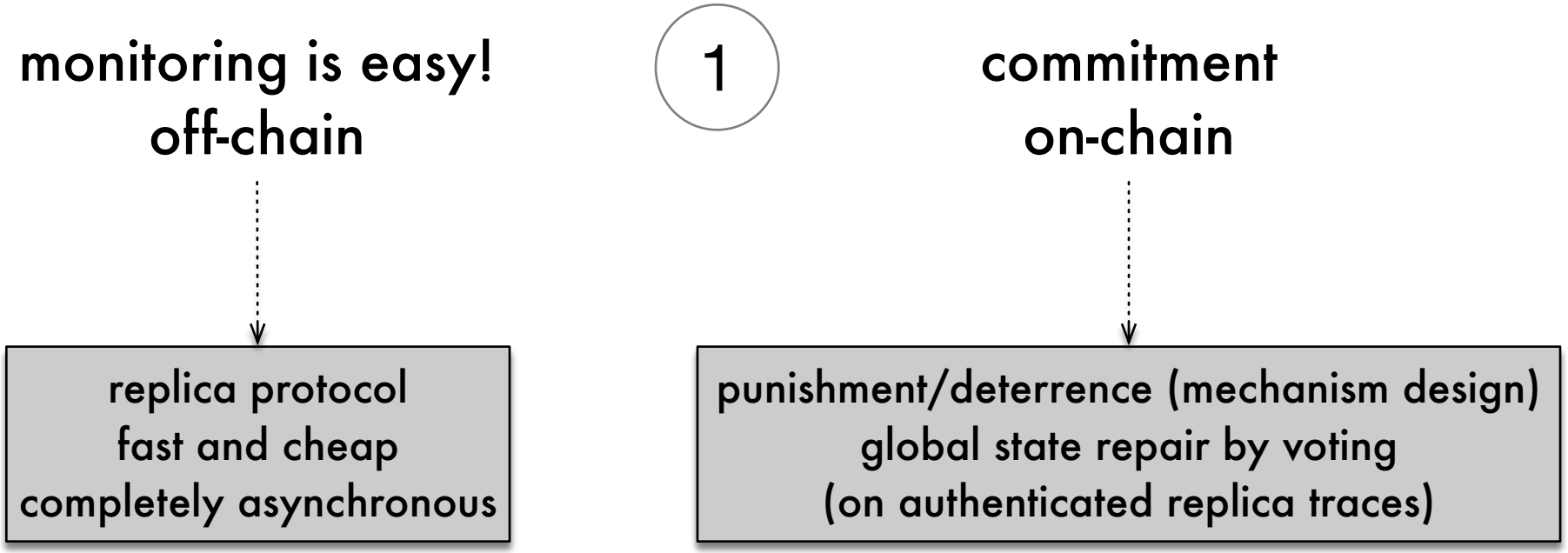



monitoring = to be sure the others are doing what they should

commitment = to punish/repair when they don't

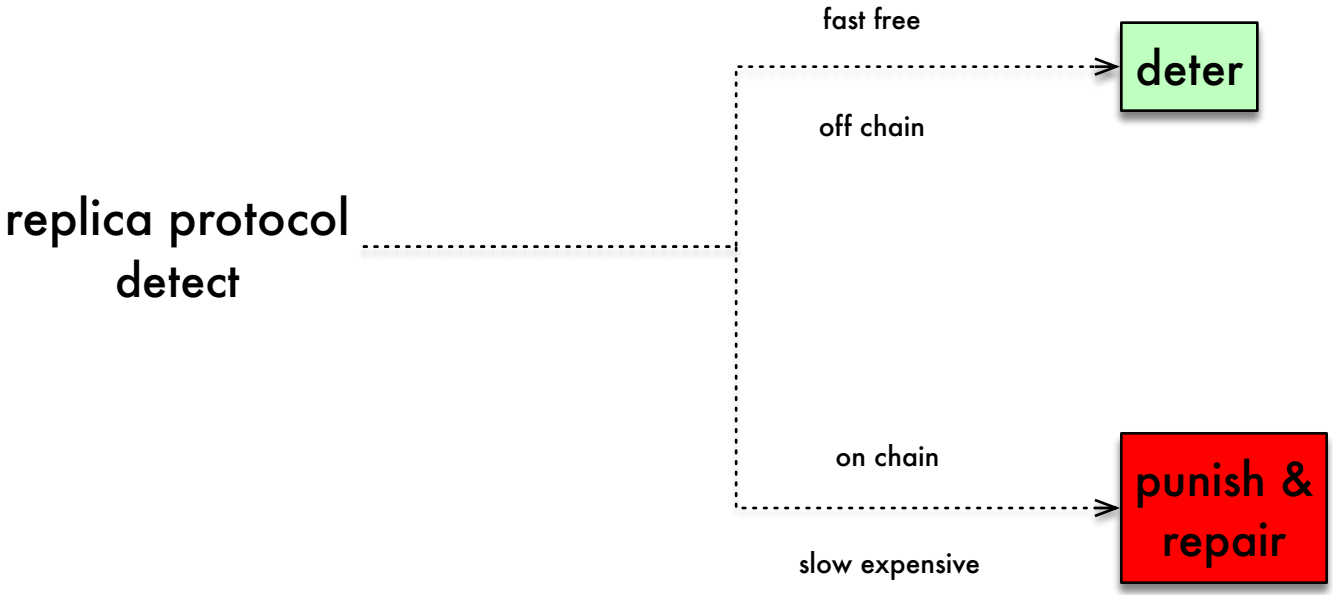
In order to carry out a market transaction it is necessary to discover who it is that one wishes to deal with, to inform people that one wishes to deal and on what terms, to conduct negotiations leading up to a bargain, to draw up the contract, to undertake the inspection needed to make sure that the terms of the contract are being observed, and so on. These operations are often extremely costly

Coase - The Problem of Social Cost
Journal of Law and Economics (1960)


monitoring is easy!
off-chain
1
commitment
on-chain
replica protocol
fast and cheap
completely asynchronous
punishment/deterrence (mechanism design)
global state repair by voting
(on authenticated replica traces)

# do not get paranoid

## B2B risk profile



most of the time everything is fine!

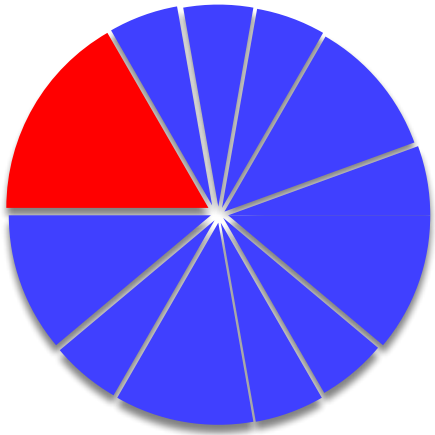
only pay for slow
and expensive BC
<u>when deterrence fails</u>

2


if your co-contractants
always get sent to the BC
for punishment maybe
change partners!

using a chain is many orders of magnitude more expensive/slow/fluctuating

so it makes sense to use it sparingly
and that is what the replica also is doing



1t/min for a year = 1c/txn
at 10% saturation

EOS' pie size is1000 tps

3

on-chain

our chain-side consensus
can be equipped with penalties
so that honest behaviour
is a (game-theoretic) equilibrium

# 3. the replica protocol

off-chain

we give one permissioned copy of the product to every player - **permissioned replicated product**

each copy is a bona fide communicating process (cSM)

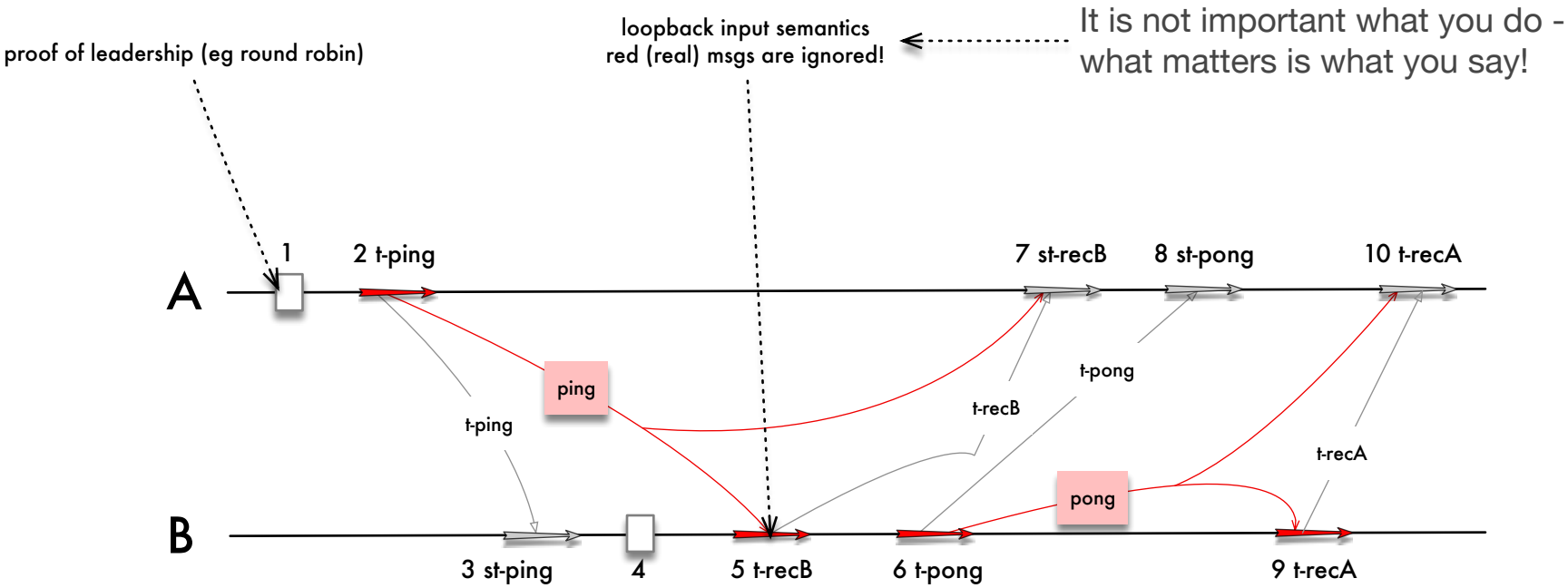
honest players run the **replica protocol**

```
let xA = ref tt in // A's state
while true do
  t-ping:
  if (!xA = tt)
    _b = "ping";     // A sends on b
    xA := ff
    or
  t-recA:
    if (!xA = ff)
    let m = a_ in    // A recvs on a
    xA := tt
done
```

```
let xB = ref ff in   // B's state
while true do
  t-pong:
    if (!xB = tt)
    _a = "pong";     // B sends on a
    xB := ff
  or
  t-recB:
  if (!xB = ff)
    let m = b_ in    // B recvs on b
    xB := tt
done
```

Figure 6: *Two processes A, B exchange messages in a cyclic fashion; their internal states have two possible values* `tt` *and* `ff`*; the joint initial state* (`tt`, `ff`) *and A has the first transition; for convenience transitions are named; those names will be used below in notifications.*

an example trace between honest players




broadcast semantics of inbound and outbound msgs (not shown)
a PRP is a process!

During the protocol a player $C$ can receive two other types of messages:

*(m1) regular signed messages of the form $\{u\}^s_A$ received on C's copy of some original input channel, with A the sender, s the round when the value was sent, and u the value sent*

*(m2) notifications of the form $\{\theta\}^t_B$ received on channel $v_{B,C}$ with B the sender, t the round when the notification was sent, and $\theta$ the transition notified.*

**I.** Suppose $A$ holds a *current* proof of leadership at time $s$. She selects an enabled transition $\theta$ for which she has permission, which she signs and notifies to all concerned as $\{\theta\}_A^s$.

cross-exam for early detection

**II.** Suppose now $B$ holds a *current* notification $\{\theta\}_A^s$. There are two cases. If $\theta$ coherently extends $B's$ current trace $\gamma_B$, that is to say, 1) $\theta$ does apply to the current state of $B$ and, 2) $\gamma_B$ generates inputs for $\theta$ using loopback input semantics, then $B$ advances, ie $\gamma_B$ += $\theta$. Else $B$ is *stuck* with a 'stucky' head $\theta$.

completely asynch!

We write $\{\theta\}_A$ for a message authenticated by $A$, as before, and $\{\theta\}_{AB}$ for a message authenticated first by $A$, then by $B$, etc. We elide time indications as all interacting messages are issued in the same round. The cross-examination protocol is as follows:[4]

> *(x1) leader A sends $\{\theta\}_A$ to all $\theta$-players*
> *(x2) [on-line comparison] B signs and resends all values of the form $\{\theta'\}_A$ (simply signed), and collects all level two values of the form $\{\theta'\}_{AB'}$ (doubly signed)*
> *(x3) [blaming] B resends and exits if he receives any locally incoherent value*

completely asynch!
non-blocking

## guarantee

We assume no cheating happens before the protocol starts, and initial states are identical across all replicas at the outset. The correctness of the protocol can thus be formulated as follows:

> *At any time, either all honest players have compatible states - or honest players will eventually discover a blame.*

additional local checks instead of the loopback semantics

# 4. on-chain: pay, punish and repair

on-chain

what to do when things go wrong

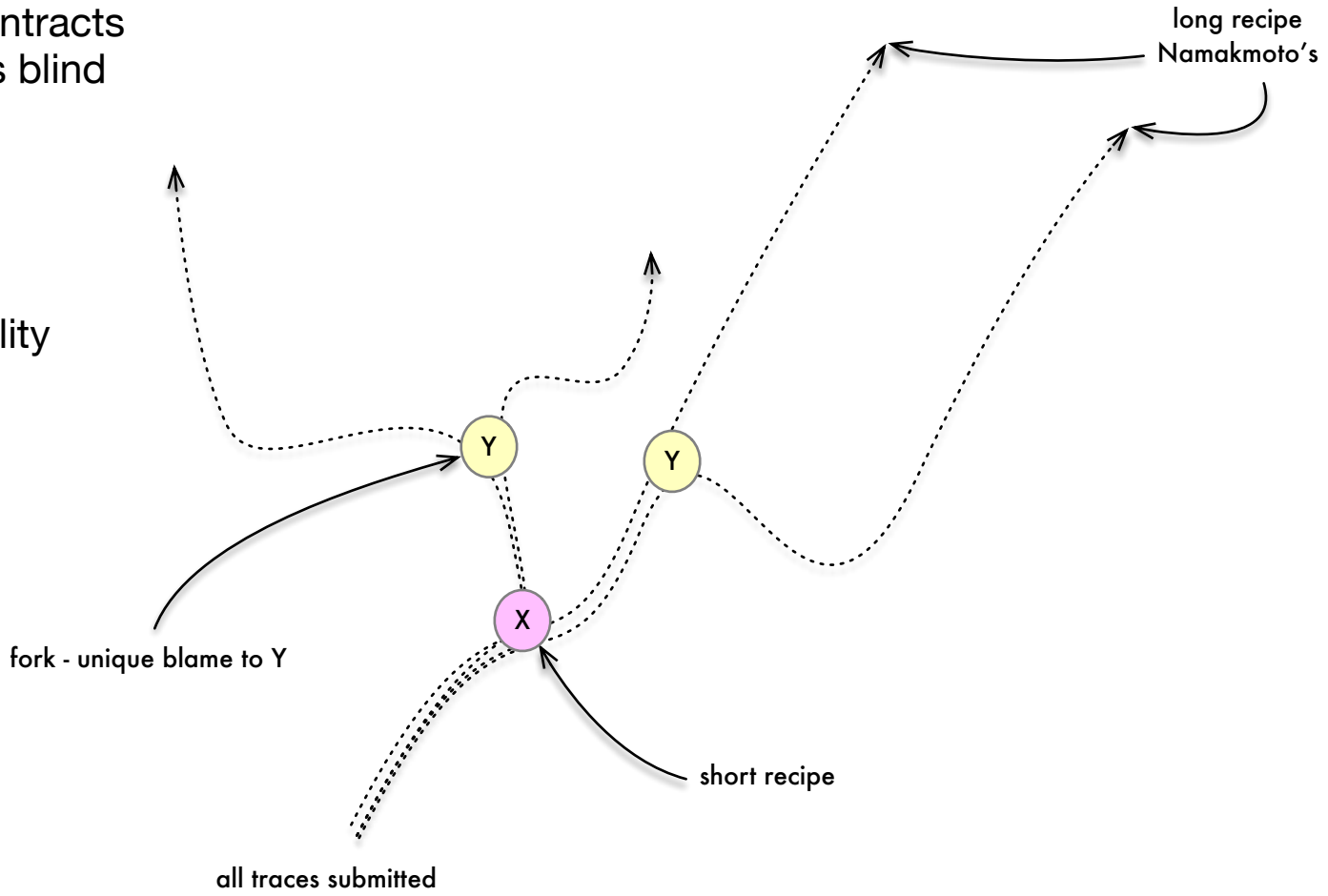
trace reconciliation (even with a 2-player contract)

players' claims are backed by authenticated traces which are compared by the MC using its recipe

all traces offered by players project to compatible sequences of leaders (by proof of leadership)
hence
forks are uniquely designating a culprit

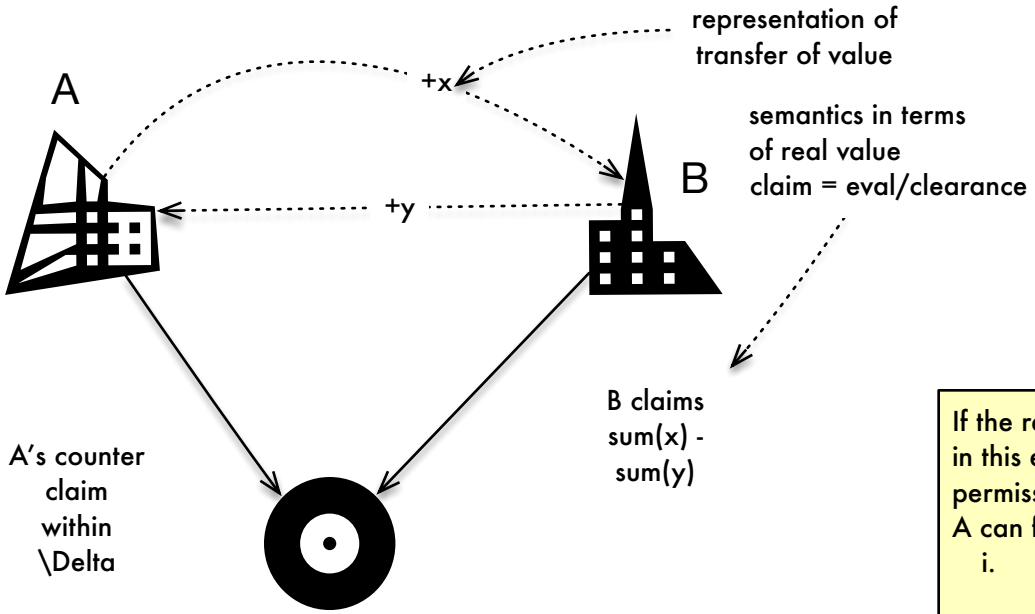
class of soft contracts
where repair is blind

confidentiality



if A forks B's trace - and the culprit is X we may keep the other player Y move or let him choose between the alternatives with a you-fork-I-choose recipe

[n/a to ≥ 3 players though]




MC mother contract contains a conflict resolution recipe

DGP principle!

game theoretic version

if A extends B's trace with X moves:
if some X=B, B loses/ie A chooses (B hid moves for no good reason - perhaps trying to omit a payment he did!)

if all X=A; there is no interest for B to not have posted this additional revenue and we take A's longer trace

If the replicande is a product (as it is in this example), and this product is permissioned in the canonical way - A can fork either

i. on an A-payment (eg , give less in her own version than she is revealing) or
ii. on a notification of a B-payment (typically and symetrically, B pretends to pay more than he does)

In this simple contract, a fork is a conflict about how much one (X say) paid to the other - but differently to ordinary life, we have evidence that X "lied".

https://github.com/igarnier/huxiang

**(in OCaml)**

collecting judiciously logs of critical transaction
can simplify and save on the
“I said, he said” game of litigation

look what happened to this 3-party contract