Software guards our secrets

Holistic Specifications

Sophia Drossopoulou, James Noble
Julian Mackay, Susan Eisenbach
Toby Murray, Mark S Milller

I AM the Bank Account

- Open World
- Third Party Code
- Future Modifications
- Trust
- Risk

Robustness

Traditional

Holistic

Robustness

Traditional

closed world

Holistic

open world

Robustness

Traditional

closed world

pre- and post conditions

Holistic

open world

temporal conditions

Robustness

Traditional

closed world

pre- and post conditions

sufficient conditions

Holistic

open world

temporal conditions

necessary conditions

Robustness

Traditional

closed world

pre- and post conditions

sufficient conditions

individual functions explict

Holistic

open world

temporal conditions

necessary conditions

emergent behaviour explicit

Robustness

Traditional

closed world

pre- and post conditions

sufficient conditions

individual functions explict

emergent behaviour implicit

Holistic

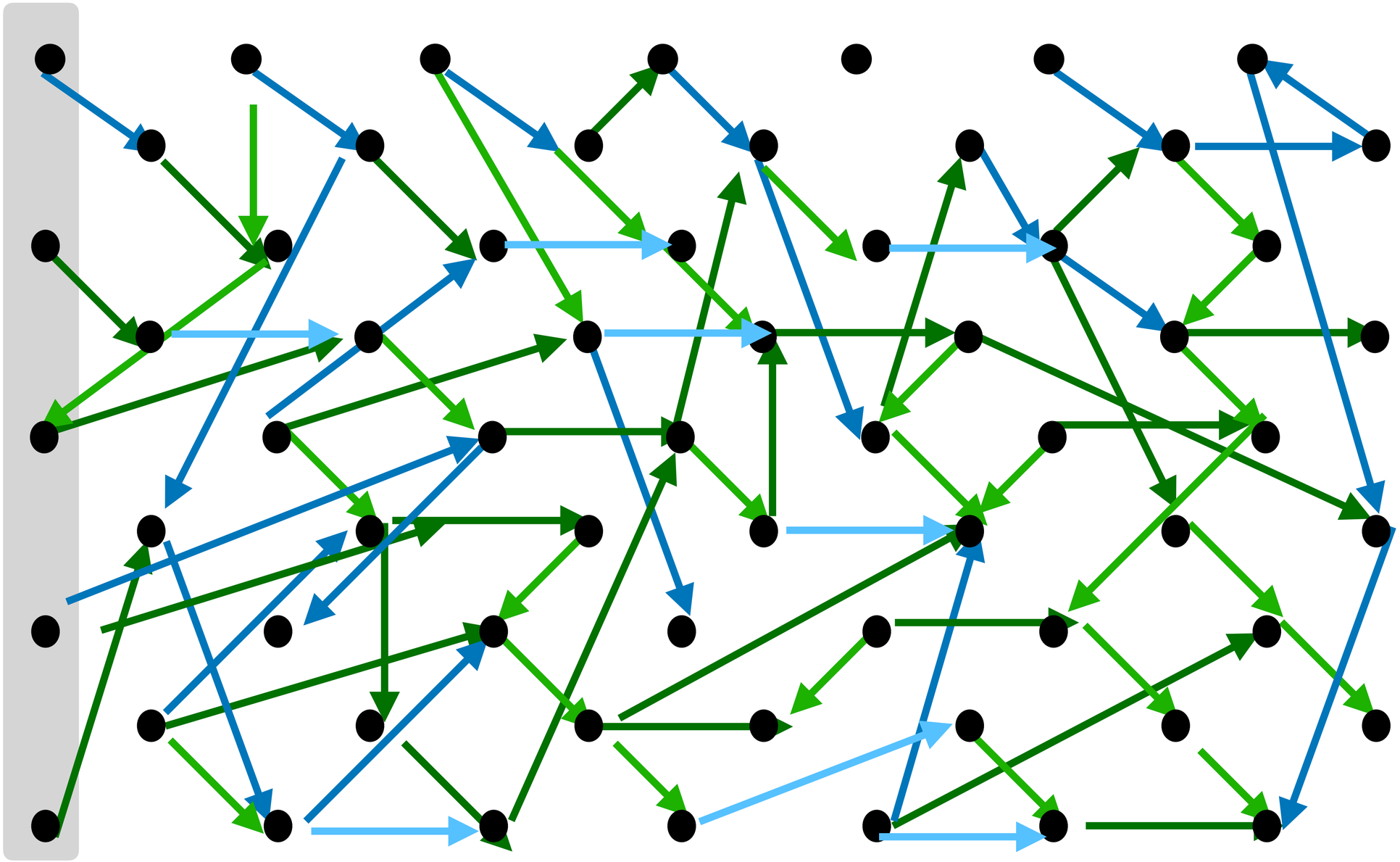
open world

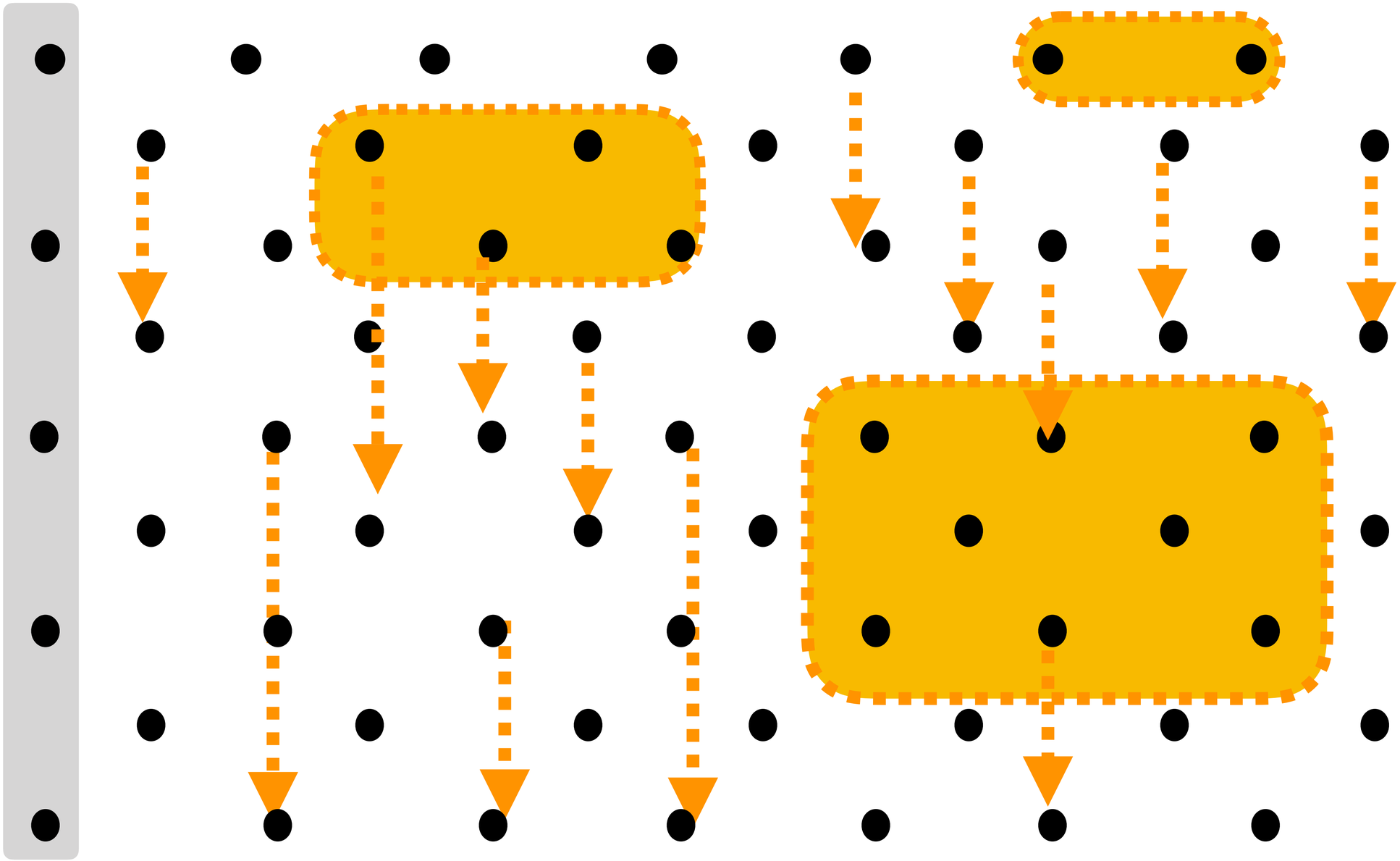
temporal conditions

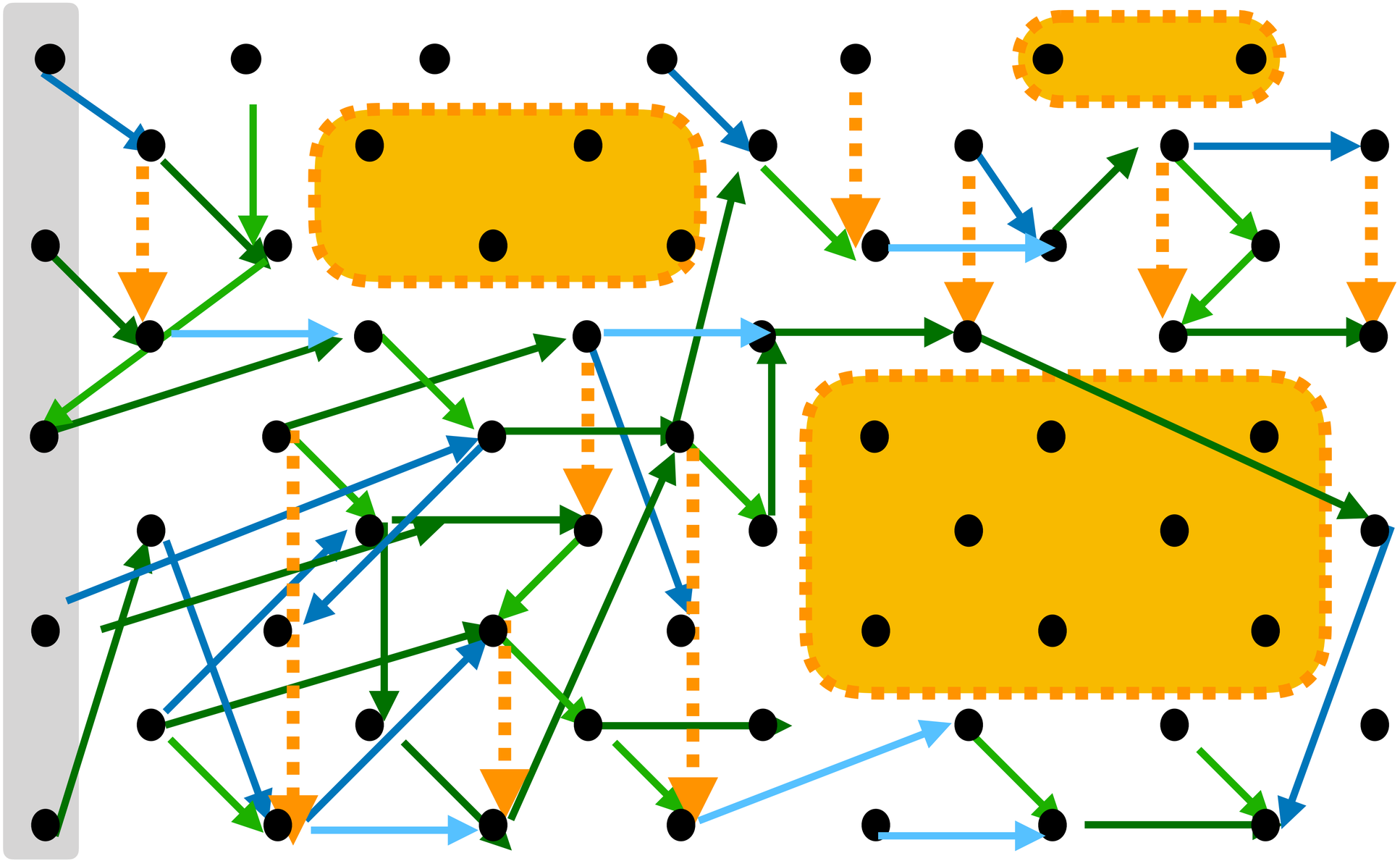
necessary conditions

emergent behaviour explicit

individual functions implicit







Assertions

$$
\begin{aligned}
A \quad ::= \; & \texttt{e} \mid \texttt{e} = \texttt{e} \mid \texttt{e} : \texttt{ClassId} \mid \texttt{e} \in \texttt{S} \mid \\
& A \to A \mid A \wedge A \mid A \vee A \mid \neg A \mid \\
& \forall \texttt{x}.A \mid \forall \texttt{S} : SET.A \mid \exists \texttt{x}.A \mid \exists \texttt{S} : SET.A \mid \\
& \langle\, x \text{ access } y \,\rangle \mid \text{changes} \langle\, e \,\rangle \mid \langle\, \texttt{x} \text{ calls } \texttt{x.m(}\, \texttt{x}^* \texttt{)} \,\rangle \\
& \text{next} \langle\, A \,\rangle \mid \text{will} \langle\, A \,\rangle \mid \text{prev} \langle\, A \,\rangle \mid \text{was} \langle\, A \,\rangle \mid \\
& \langle\, \texttt{S} \text{ in } A \,\rangle \mid \text{external} \langle\, \texttt{x} \,\rangle \mid \texttt{x} \; \textbf{\textit{obeys}} \; \texttt{S}
\end{aligned}
$$

permission **authority** **control**

temporal

space **viewpoint** **trust**

ERC 20

Ethereum contract — manages tokens for clients

transfer(dst, m) — transfer **m** tokens from ***sender*** to **dst**

approve(pxy, m) — let **pxy** spend **m** tokens on behalf of ***sender***

transferFrom(src, dst, m) — transfer **m** tokens from **src** to **dst**

balance — number of tokens held by each client

Hoare Logic

precondition

$\{C\}$

postcondition

sufficient

effect

ERC 20 - transfer

$$e : \mathtt{ERC20} \land p, p'' : \mathtt{Object} \land m, m', m'' : \mathtt{Nat} \land e.\mathtt{balance}(p) = m + m' \land e.\mathtt{balance}(p'') = m'' \land \mathtt{this} = p \; \{\, e.\mathtt{transfer}(p'', m') \,\} \; e.\mathtt{balance}(p) = m \land e.\mathtt{balance}(p'') = m'' + m'$$

sufficient effect

$$e : \mathtt{ERC20} \land p, p' : \mathtt{Object} \land m, m', m'' : \mathtt{Nat} \land e.\mathtt{balance}(p) = m \land m < m' \; \{\, e.\mathtt{transfer}(p', m') \,\} \; e.\mathtt{balance}(p) = m$$

insufficient no effect

ERC 20 - delegated transfer

$$
\begin{array}{l}
\texttt{e} : \texttt{ERC20} \wedge \texttt{p}, \texttt{p}', \texttt{p}'' : \texttt{Object} \wedge \texttt{m}, \texttt{m}', \texttt{m}'', \texttt{m}''' : \texttt{Nat} \wedge \\
\texttt{e.balance(p)} = \texttt{m} + \texttt{m}' \ \wedge \ \texttt{e.allowed(p,p}') = \texttt{m}''' + \texttt{m}' \wedge \\
\texttt{e.balance(p}'') = \texttt{m}'' \ \wedge \ \texttt{this} = \texttt{p}' \\
\{ \ \texttt{e.transferFrom(p}', \texttt{p}'', \texttt{m}') \ \} \\
\texttt{e.balance(p)} = \texttt{m} \ \wedge \ \texttt{e.balance(p}'') = \texttt{m}'' + \texttt{m}' \ \wedge \ \texttt{e.allowed(p,p}') = \texttt{m}'''
\end{array}
$$

$$
\begin{array}{l}
\texttt{e} : \texttt{ERC20} \wedge \texttt{p}, \texttt{p}' : \texttt{Object} \wedge \texttt{m}, \texttt{m}', \texttt{m}'' : \texttt{Nat} \ \wedge \ \texttt{this} = \texttt{p}' \wedge \\
(\texttt{e.balance(p)} = \texttt{m} \wedge \texttt{m} < \texttt{m}'' \vee \texttt{e.allowed(p,p}') = \texttt{m}' \wedge \texttt{m}' < \texttt{m}'') \\
\{ \ \texttt{e.transferFrom(p,p}'', \texttt{m}'') \ \} \\
\texttt{e.balance(p)} = \texttt{m} \wedge \texttt{e.allowed(p,p}') = \texttt{m}'
\end{array}
$$

ERC 20

Who stole my money?

super-client?

take 5% from each

ERC 20 — holistic transfer

effect

$$
\begin{array}{l}
\forall \mathtt{e} : \mathtt{ERC20}.\forall \mathtt{p} : \mathtt{Object}.\forall \mathtt{m}, \mathtt{m}' : \mathtt{Nat}. \\
\quad [\ \mathtt{e.balance(p)} = \mathtt{m} + \mathtt{m}' \ \wedge\ \mathsf{next}\langle \mathtt{e.balance(p)} = \mathtt{m}' \rangle \\
\qquad \longrightarrow \\
\qquad \exists \mathtt{p}', \mathtt{p}'' : \mathtt{Object}. \\
\qquad [\ \langle\, \mathtt{p}\ \mathsf{calls}\ \mathtt{e.transfer(\,p', m)}\ \rangle\ \vee \\
\qquad\quad \mathtt{e.allowed(p, p'')} \geq \mathtt{m} \wedge \langle\, \mathtt{p''}\ \mathsf{calls}\ \mathtt{e.transferFrom(\,p', m)}\ \rangle\ ] \\
\quad]
\end{array}
$$

necessary

ERC 20 — holistic authority

$$
\begin{array}{l}
\forall \mathtt{e} : \mathtt{ERC20}.\forall \mathtt{p}, \mathtt{p}' : \mathtt{Object}.\forall \mathtt{m} : \mathtt{Nat}. \\
\quad [\ \mathtt{e.allowed}(\mathtt{p}, \mathtt{p}') = \mathtt{m} \\
\qquad \longrightarrow \\
\qquad \mathcal{P}rev\langle\ \langle\ \mathtt{p}\ \textsf{calls}\ \mathtt{e.approve}(\ \mathtt{p}', \mathtt{m})\ \rangle \\
\qquad\qquad \vee \\
\qquad\qquad \mathtt{e.allowed}(\mathtt{p}, \mathtt{p}') = \mathtt{m} \wedge \\
\qquad\qquad \neg(\langle\ \mathtt{p}'\ \textsf{calls}\ \mathtt{e.transferFrom}(\ \mathtt{p}, \_)\ \rangle \vee \langle\ \mathtt{p}\ \textsf{calls}\ \mathtt{e.approve}(\ \mathtt{p}, \_)\ \rangle) \\
\qquad\qquad \vee \\
\qquad\qquad \exists \mathtt{p}'' : \mathtt{Object}.\exists \mathtt{m}' : \mathtt{Nat}. \\
\qquad\qquad [\mathtt{e.allowed}(\mathtt{p}, \mathtt{p}') = \mathtt{m} + \mathtt{m}' \wedge \langle\ \mathtt{p}'\ \textsf{calls}\ \mathtt{e.transferFrom}(\ \mathtt{p}'', \mathtt{m}')\ \rangle] \\
\qquad\ \rangle \\
\quad]
\end{array}
$$

effect

necessary

DAO

Ethereum contract — "decentralised autonomous organisation"

19 Methods

d.join(m) — ***sender*** transfers **m** into DAO

d.pay — ***sender*** wants their money back

r.send(m) — ***sender*** transfers **m** to **r**

d.balance(client) — number of tokens held by each client

DAO — holistic balance

necessary

$$\forall d : \texttt{DAO}.\forall p.\forall : m : \texttt{Nat}.$$
$$[\ \texttt{d.Balance(p)} = m \longrightarrow [\ \text{prev}\langle\langle\, p \text{ calls } \texttt{repay.d(\_)} \,\rangle\rangle \wedge m = 0 \ \vee$$
$$\text{prev}\langle\langle\, p \text{ calls } \texttt{join.d(m)} \,\rangle\rangle \ \vee$$
$$\ldots\ ]$$

effect

$$]$$

DAO — holistic repayments

covers balances

$$\forall d : \mathtt{DAO}.\forall p : \mathtt{Any}.\forall m : \mathtt{Nat}.\ [\ d.\mathtt{balances}(p) = m \longrightarrow d.\mathtt{ether} \geq m\ ]$$

pays when asked

$$\forall d : \mathtt{DAO}.\forall p : \mathtt{Any}.\forall m : \mathtt{Nat}.\ [\ d.\mathtt{balance}(p) = m \wedge \langle\, p\ \mathsf{calls}\ \mathtt{repay}.d(\_)\,\rangle \longrightarrow \mathsf{will}\langle\,\langle\, d\ \mathsf{calls}\ \mathtt{send}.p(m)\,\rangle\,\rangle\ ]$$

best effort

$$\longrightarrow\ p.\mathtt{funds} = \mathsf{prev}\langle p.\mathtt{funds}\rangle + m$$

$$\forall d : \mathtt{DAO}.\forall p : \mathtt{Any}.\forall m : \mathtt{Nat}.\ \langle\, d\ \mathsf{calls}\ \mathtt{send}.p(m)\,\rangle \longrightarrow d.\mathtt{ether} \geq m$$

eventual payments

Digested week
Boris Johnson

Sketch writing feels impossible in the age of Dom and Dommer

John Crace

As satire and reality collide I find it difficult to see the funny side of anything going on in Westminster

@JohnJCrace

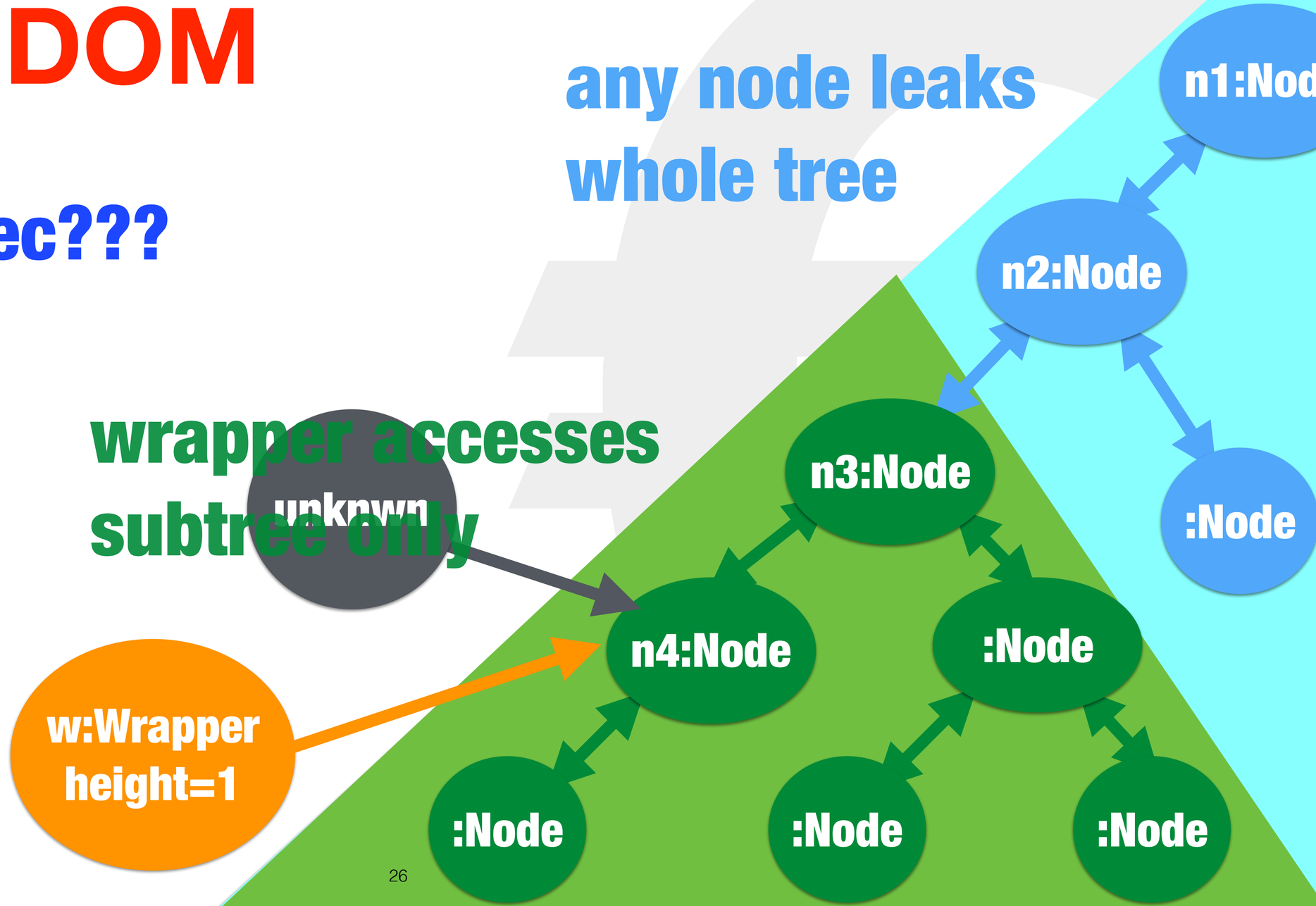
Fri 6 Sep 2019 12.30 BST

Classic DOM

classical spec???

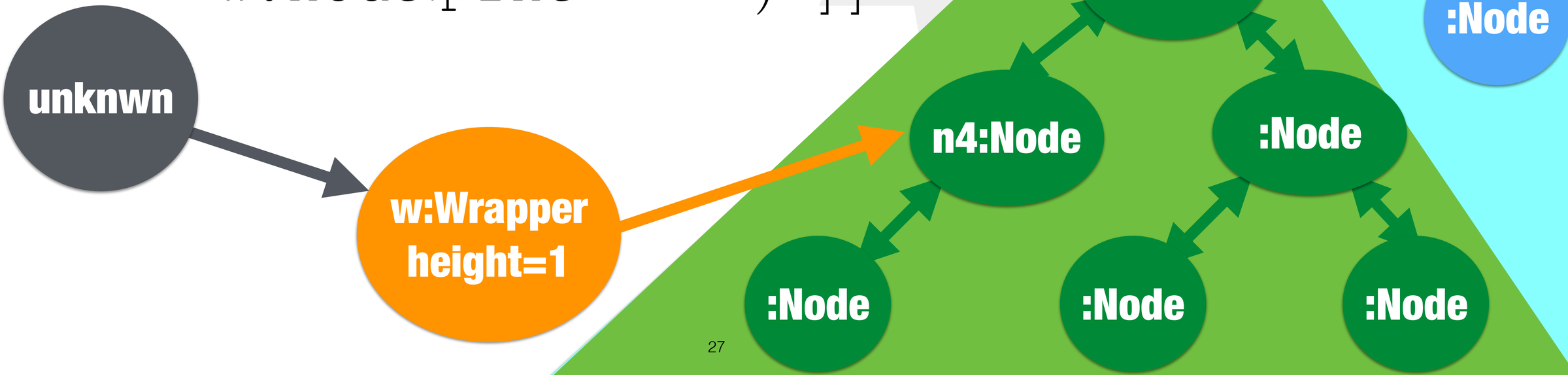
any node leaks
whole tree

wrapper accesses
subtree only



26

$$
\begin{array}{l}
\forall S : \mathtt{Set}.\forall nd : \mathtt{Node}.\forall o : \mathtt{Object}. \\
[\ \langle\, \textsf{will}\langle\, \textsf{changes}\langle\, nd.property \,\rangle\rangle\, \textsf{in}\, S \,\rangle \\
\quad \longrightarrow \\
\quad \exists o.[\, o \in S \ \wedge\ \neg(o : \mathtt{Node}) \ \wedge\ \neg(o : \mathtt{Wrapper}) \ \wedge \\
\qquad [\, \exists nd' : \mathtt{Node}.\langle\, o\ \textsf{access}\ nd' \,\rangle \ \vee \\
\qquad\quad \exists w : \mathtt{Wrapper}.\exists k{:}\mathbb{N}.(\, \langle\, o\ \textsf{access}\ w \,\rangle \\
\qquad\qquad \wedge\ nd.prnt^{k} \\
\qquad\qquad\quad = w.node.prnt^{w.hgt}) \ ]] \\
]
\end{array}
$$

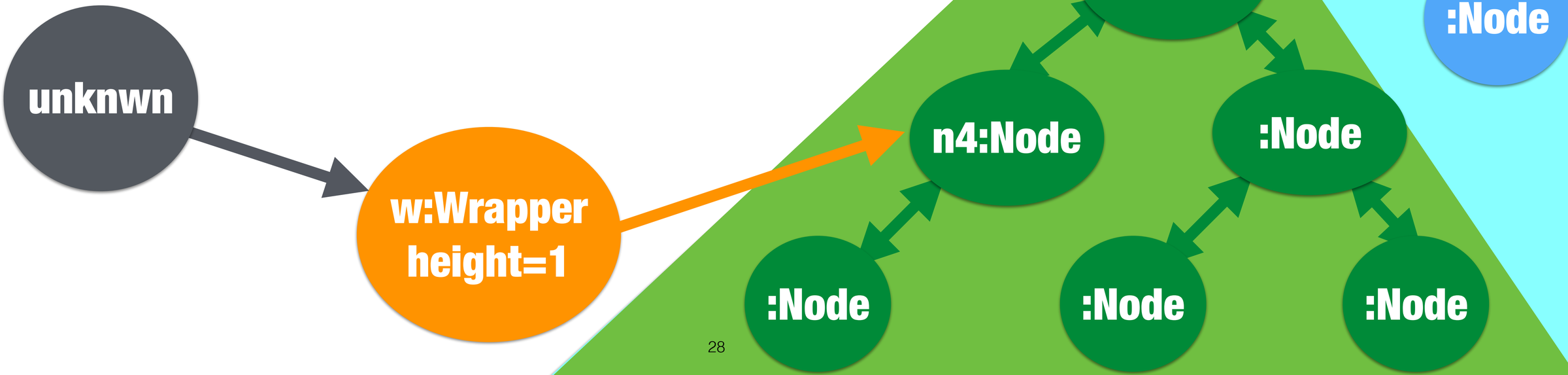


27

```
method usingWrappers(unknwn)
     {
    n1=Node(null,"fixed");
    n2=Node(n1,"robust");
    n3=Node(n2,"const");
    n4=Node(n3,"volatile");
    n5=Node(n4,"variable");
    n6=Node(n5,"ethereal");
    w=Wrapper(n5,1);
```

```
unknwn.untrusted(w);

assert n2.property=="
    robust"
...
```



Bank and Accounts

Accounts hold money

Money can be transferred between Accounts

A bank's currency = sum of balances of accounts held by bank

Object-capability example

[Miller et al, Financial Crypto 2000]

Bank and Accounts

Pol_1: With two accounts of same bank can transfer money.

Pol_2: Only someone with the Bank of a given currency can violate conservation of that currency

Pol_3: Only the bank can inflate own currency

Pol_4: No one can affect the balance of an account they do not have.

Pol_5: Balances are always non-negative.

Pol_6: A reported successful deposit can be trusted as much as one trusts the account one is depositing to.

[Miller et al, Financial Crypto 2000]

Bank and Accounts

Pol_4: No one can affect the balance of an account they do not have.

$$\forall a.\forall S : \texttt{Set}.[\ a : \texttt{Account} \wedge \langle \textsf{will}\langle\ \textsf{changes}\langle\ a\texttt{.balance}\ \rangle\rangle\ \textsf{in}\ S\ \rangle \longrightarrow \exists o.[\ o \in S \wedge \textsf{external}\langle\ o\ \rangle \wedge \langle\ o\ \textsf{access}\ a\ \rangle]\ ]$$

If an account will have a different value *(in the future)*

then some external object must be able to get to it now!

Bank and Accounts

Pol_7: Only calling deposit can affect account balances.

$$\forall a.[\ a : \texttt{Account} \wedge \textbf{changes}\langle a.\texttt{balance} \rangle \longrightarrow \exists o.[\langle o\ \textbf{calls}\ a.\texttt{deposit}(\_,\_) \rangle \vee \langle o\ \textbf{calls}\ \_.\texttt{deposit}(a,\_) \rangle]$$

If an account will have a different value *(in the future)*

either **it received a deposit request,**
or **was passed as an argument to deposit**

Chainmail

Kripke Worlds

- $\mathcal{I}nitial\langle(\psi, \chi)\rangle$, *if* ψ *consists of a single frame* ϕ *with* $dom(\phi) = \{\texttt{this}\}$, *and there exists some address* α, *such that* $\lfloor\texttt{this}\rfloor_\phi = \alpha$, *and* $dom(\chi) = \alpha$, *and* $\chi(\alpha) = (\texttt{Object}, \emptyset)$.
- $\mathcal{A}rising(\texttt{M}\,;\texttt{M}') = \{\, \sigma \mid \exists \sigma_0.\, [\, \mathcal{I}nitial\langle\sigma_0\rangle \;\wedge\; \texttt{M}\,;\texttt{M}', \sigma_0 \rightsquigarrow^* \sigma \,] \,\}$

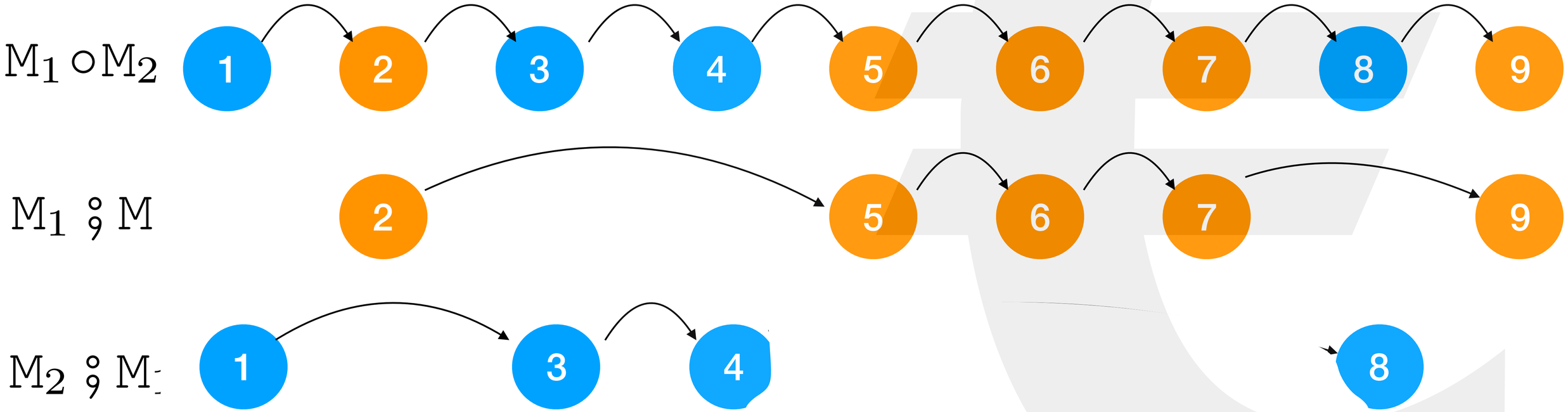
Two-module execution

- $\mathtt{M} \fatsemi \mathtt{M}', \sigma \rightsquigarrow \sigma'$ *if there exist* $n \geq 2$ *and runtime configurations* σ_1, ... σ_n, *such that*
  - $\sigma = \sigma_1$, *and* $\sigma_n = \sigma'$.
  - $\mathtt{M} \circ \mathtt{M}', \sigma_i \rightsquigarrow \sigma'_{i+1}$, *for* $1 \leq i \leq n-1$
  - $\mathcal{C}lass(\mathtt{this})_{\sigma} \notin dom(\mathtt{M})$, *and* $\mathcal{C}lass(\mathtt{this})_{\sigma'} \notin dom(\mathtt{M})$,
  - $\mathcal{C}lass(\mathtt{this})_{\sigma_i} \in dom(\mathtt{M})$, *for* $2 \leq i \leq n-2$

M — internal module

M' - external module

Two-module execution



Internal methods are treated as atomic

Assertions

$$
\begin{aligned}
A \quad ::= \; & \texttt{e} \mid \texttt{e} = \texttt{e} \mid \texttt{e} : \texttt{ClassId} \mid \texttt{e} \in \texttt{S} \mid \\
& A \to A \mid A \wedge A \mid A \vee A \mid \neg A \mid \\
& \forall \texttt{x}.A \mid \forall \texttt{S} : SET.A \mid \exists \texttt{x}.A \mid \exists \texttt{S} : SET.A \mid \\
& \langle\, x \text{ access } y \,\rangle \mid \text{changes} \langle\, e \,\rangle \mid \langle\, \texttt{x} \text{ calls } \texttt{x.m(}\, \texttt{x}^* \texttt{)} \,\rangle \\
& \text{next} \langle\, A \,\rangle \mid \text{will} \langle\, A \,\rangle \mid \text{prev} \langle\, A \,\rangle \mid \text{was} \langle\, A \,\rangle \mid \\
& \langle\, \texttt{S} \text{ in } A \,\rangle \mid \text{external} \langle\, \texttt{x} \,\rangle \mid \texttt{x} \; \textbf{\textit{obeys}} \; \texttt{S}
\end{aligned}
$$

permission **authority** **control**

temporal

space **viewpoint** **trust**

Permission

- $\mathtt{M}\,;\,\mathtt{M}', \sigma \models \langle\, \mathtt{x}\ \mathsf{access}\ \mathtt{y}\,\rangle$ *if* $\lfloor \mathtt{x} \rfloor_\sigma$ *and* $\lfloor \mathtt{y} \rfloor_\sigma$ *are defined, and*
  - $\lfloor \mathtt{x} \rfloor_\sigma = \lfloor \mathtt{y} \rfloor_\sigma$, *or*
  - $\lfloor \mathtt{x.f} \rfloor_\sigma = \lfloor \mathtt{y} \rfloor_\sigma$, *for some field* $\mathtt{f}$, *or*
  - $\lfloor \mathtt{x} \rfloor_\sigma = \lfloor \mathtt{this} \rfloor_\sigma$ *and* $\lfloor \mathtt{y} \rfloor_\sigma = \lfloor \mathtt{z} \rfloor_\sigma$, *for some variable* $\mathtt{z}$ *and* $\mathtt{z}$ *appears in* $\sigma.\mathtt{contn}$.

Control

- $\mathtt{M}\,;\,\mathtt{M}', \sigma \models \langle\, \mathtt{x}\ \mathsf{calls}\ \mathtt{y.m(}\ \mathtt{z}_1, \ldots \mathtt{z}_n\mathtt{)}\ \rangle$ *if* $\lfloor \mathtt{x} \rfloor_\sigma$, $\lfloor \mathtt{y} \rfloor_\sigma$, $\lfloor \mathtt{z}_1 \rfloor_\sigma$, ... $\lfloor \mathtt{z}_n \rfloor_\sigma$ *are defined, and*
  - $\lfloor \mathtt{this} \rfloor_\sigma = \lfloor \mathtt{x} \rfloor_\sigma$, *and*
  - $\sigma.\mathtt{contn} = \mathtt{u.m}(\mathtt{v}_1, .. \mathtt{v}_n); \_$, *for some* $\mathtt{u}, \mathtt{v}_1, \ldots \mathtt{v}_n$, *and*
  - $\lfloor \mathtt{y} \rfloor_\sigma = \lfloor \mathtt{u} \rfloor_\sigma$, *and* $\lfloor \mathtt{z}_i \rfloor_\sigma = \lfloor \mathtt{v}_i \rfloor_\sigma$, *for all* i.

Viewpoint

- $\mathtt{M} \fatsemi \mathtt{M}', \sigma \models \mathsf{external}\langle\, \mathtt{x} \,\rangle \quad \textit{if} \quad \lfloor \mathtt{x} \rfloor_\sigma \textit{ is defined and } \mathcal{C}lass(\lfloor \mathtt{x} \rfloor_\sigma)_\sigma \notin dom(\mathtt{M})$
- $\mathtt{M} \fatsemi \mathtt{M}', \sigma \models \mathsf{internal}\langle\, \mathtt{x} \,\rangle \quad \textit{if} \quad \lfloor \mathtt{x} \rfloor_\sigma \textit{ is defined and } \mathcal{C}lass(\lfloor \mathtt{x} \rfloor_\sigma)_\sigma \in dom(\mathtt{M})$

Space

Restriction:

- $\sigma{\downarrow_{\mathtt{S}}} \triangleq (\psi, \chi'), \quad \textit{if} \quad \sigma{=}(\psi, \chi), \textit{ and } dom(\chi') = \lfloor \mathtt{S} \rfloor_\sigma, \textit{and} \forall \alpha \in dom(\chi').\chi(\alpha) = \chi'(\alpha)$

Space:

- $\mathtt{M} ; \mathtt{M}', \sigma \models \langle\, A \,\mathsf{in}\, \mathtt{S} \,\rangle \quad \textit{if} \quad \mathtt{M} ; \mathtt{M}', \sigma{\downarrow_{\mathtt{S}}} \models A.$

Time

Adaptation:

- $\sigma \triangleleft \sigma' \triangleq (\phi'' \cdot \psi', \chi')$ *if*
  - $\sigma = (\phi \cdot \_, \_)$, $\sigma' = (\phi' \cdot \psi', \chi')$, *and*
  - $\phi = (\texttt{contn}, \beta)$, $\phi' = (\texttt{contn}', \beta')$, $\phi'' = (\texttt{contn}'[\texttt{zs}/\texttt{zs}'], \beta[\texttt{zs}' \mapsto \beta'(\texttt{zs})])$, *where*
  - $\texttt{zs} = dom(\beta)$, $\texttt{zs}'$ *is a set of variables with the same cardinality as* $\texttt{zs}$, *and*
  - *all variables in* $\texttt{zs}'$ *are fresh in* β *and in* β'.

Time

Temporal Operators:

- $\mathtt{M} \fatsemi \mathtt{M}', \sigma \models \mathsf{next}\langle\, A \,\rangle \quad \textit{if} \quad \exists \sigma'.[\ \mathtt{M} \fatsemi \mathtt{M}', \phi \rightsquigarrow \sigma' \ \wedge \mathtt{M} \fatsemi \mathtt{M}', \sigma \triangleleft \sigma' \models A\ ]$,
  and where ϕ *is so that* $\sigma = (\phi \cdot \_, \_)$.
- $\mathtt{M} \fatsemi \mathtt{M}', \sigma \models \mathsf{will}\langle\, A \,\rangle \quad \textit{if} \quad \exists \sigma'.[\ \mathtt{M} \fatsemi \mathtt{M}', \phi \rightsquigarrow^* \sigma' \ \wedge \mathtt{M} \fatsemi \mathtt{M}', \sigma \triangleleft \sigma' \models A\ ]$,
  and where ϕ *is so that* $\sigma = (\phi \cdot \_, \_)$.
- $\mathtt{M} \fatsemi \mathtt{M}', \sigma \models \mathsf{prev}\langle\, A \,\rangle \quad \textit{if} \quad \forall \sigma_1, \sigma_2.[\ \mathcal{I}\mathit{nitial}\langle \sigma_1 \rangle \ \wedge\ \mathtt{M} \fatsemi \mathtt{M}', \sigma_1 \rightsquigarrow^* \sigma_2 \ \wedge \mathtt{M} \fatsemi \mathtt{M}', \sigma_2 \rightsquigarrow \sigma \ \longrightarrow\ \mathtt{M} \fatsemi \mathtt{M}', \sigma \triangleleft \sigma_2 \models A\ ]$
- $\mathtt{M} \fatsemi \mathtt{M}', \sigma \models \mathsf{was}\langle\, A \,\rangle \quad \textit{if} \quad \forall \sigma_1.[\ \mathcal{I}\mathit{nitial}\langle \sigma_1 \rangle \ \wedge\ \mathtt{M} \fatsemi \mathtt{M}', \sigma_1 \rightsquigarrow^* \sigma \longrightarrow (\ \exists \sigma_2.\mathtt{M} \fatsemi \mathtt{M}', \sigma_1 \rightsquigarrow^* \sigma_2 \ \wedge\ \mathtt{M} \fatsemi \mathtt{M}', \sigma_2 \rightsquigarrow^* \sigma \ \wedge\ \mathtt{M} \fatsemi \mathtt{M}', \sigma \triangleleft \sigma_2 \models A\ )]$

Robustness

Traditional

closed world

pre- and post conditions

sufficient conditions

individual functions explict

emergent behaviour implicit

Holistic

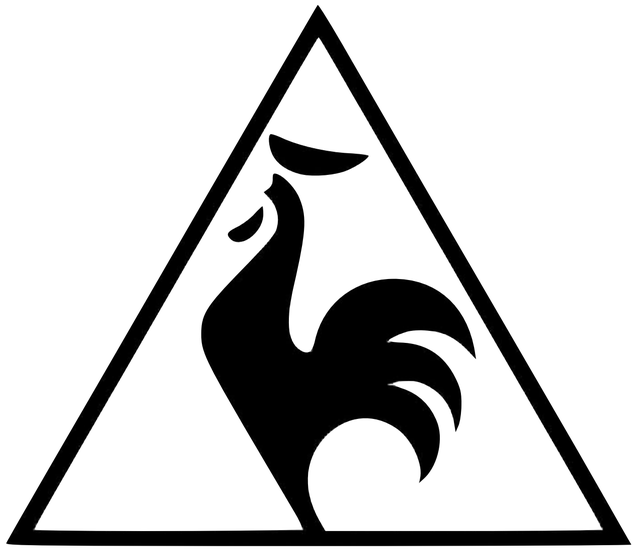
open world

temporal conditions

necessary conditions

emergent behaviour explicit

individual functions implicit



Properties of Linking	(1) moduleLinking_associative (2) moduleLinking_commutative_1 (3) moduleLinking_commutative_2 (4) linking_preserves_reduction
(1) $A \wedge \neg A \equiv \texttt{false}$ (2) $A \vee \neg A \equiv \texttt{true}$ (3) $A \vee A' \equiv A' \wedge A$ (4) $A \wedge A' \equiv A' \wedge A$ (5) $(A \vee A') \vee A'' \equiv A \vee (A' \vee A'')$	(1) sat_and_nsat_equiv_false (2) - (3) and_commutative (4) or_commutative (5) or_associative
(1) $A \wedge \neg A \equiv \texttt{false}$ (2) $A \vee \neg A \equiv \texttt{true}$ (3) $A \vee A' \equiv A' \wedge A$ (4) $A \wedge A' \equiv A' \wedge A$ (5) $(A \vee A') \vee A'' \equiv A \vee (A' \vee A'')$ (6) $(A \vee A') \wedge A'' \equiv (A \vee A'') \wedge (A' \vee A'')$ (7) $(A \wedge A') \vee A'' \equiv (A \wedge A'') \vee (A' \wedge A'')$ (8) $\neg(A \wedge A') \equiv (\neg A \vee \neg A')$ (9) $\neg(A \vee A') \equiv (\neg A \wedge \neg A')$ (10) $\neg(\exists x.A) \equiv \forall x.(\neg A)$ (11) $\neg(\exists S.A) \equiv \forall S.(\neg A)$ (12) $\neg(\forall x.A) \equiv \exists x.(\neg A)$ (13) $\neg(\forall S.A) \equiv \exists S.(\neg A)$	(1) sat_and_nsat_equiv_false (2) - (3) and_commutative (4) or_commutative (5) or_associative (6) and_distributive (7) or_distributive (8) neg_distributive_and (9) neg_distributive_or (10) not_ex_x_all_not (11) not_ex_Σ_all_not (12) not_all_x_ex_not (13) not_all_Σ_ex_not

Trust and Obey

```
1 method dealV1(
2         sellerMoney, sellerGoods,
3         buyerMoney, buyerGoods,
4         price, amt)    // price and amount
```

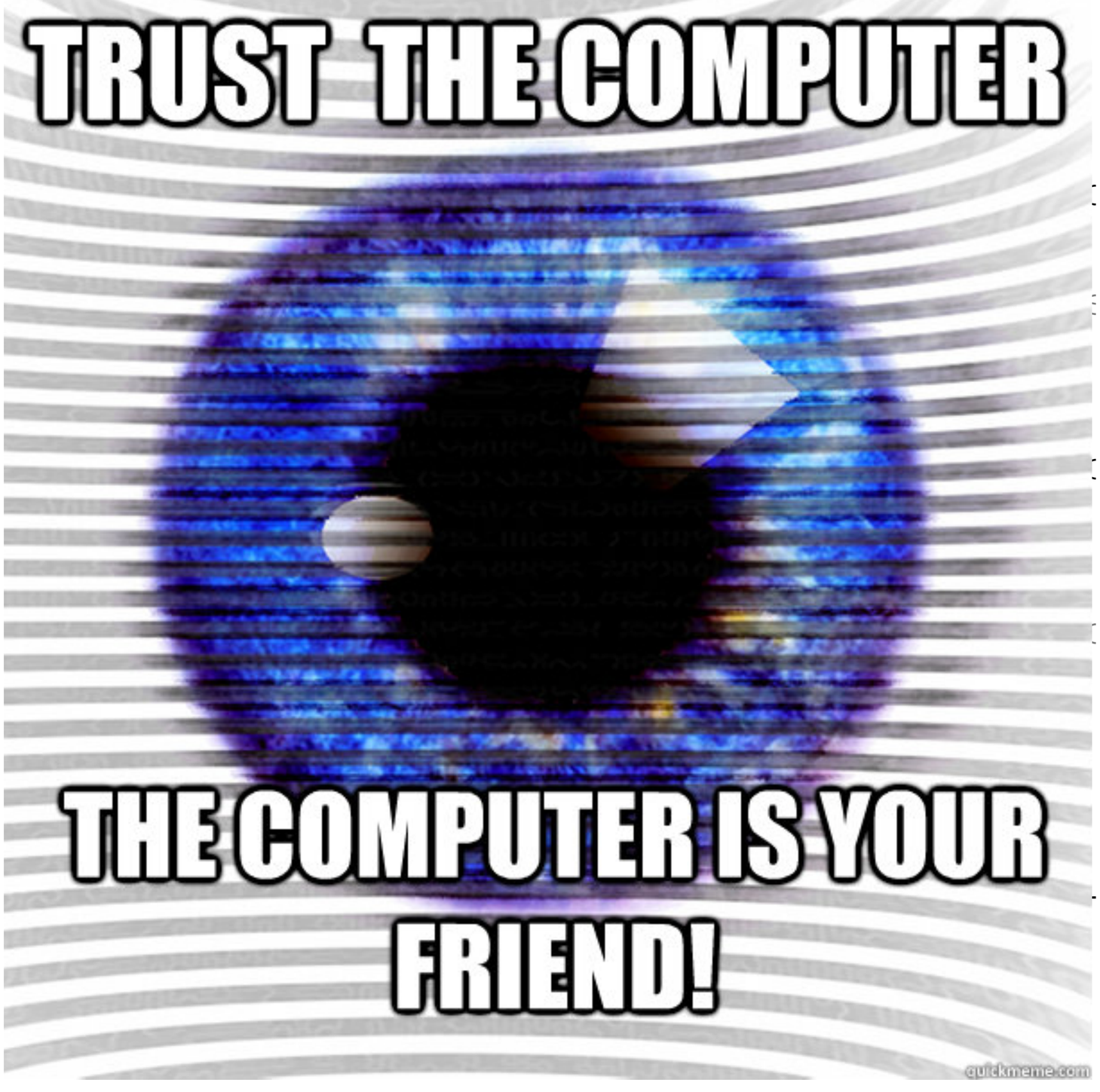
Escrow

```
1  method dealV1(
2          sellerMoney, sellerGoods,
3          buyerMoney, buyerGoods,
4          price, amt)    // price and amount
5    {
6      // make temporary money Purse
7      escrowMoney = sellerMoney.sprout
8      // make temporary goods Purse
9      escrowGoods = buyerGoods.sprout
```

Escrow

```
11    res = escrowMoney.deposit(price,buyerMoney)
12    if (!res) then
13          // insufficient money in buyerMoney
14          // or different money mints
15          { return false }
16
17    // sufficient money, same mints
18    // price transferred to escrowMoney
```

```
19      res = escrowGoods.deposit(amt, sellerGoods)
20      if (!res) then
21            // insufficient goods in sellerGoods
22            // or different goods mints
23            {  // undo the goods transaction
24               buyerMoney.deposit(price,escrowMoney)
25               return false }
26
27      // price in escrowMoney, amt in escrowGoods
28      // now complete the transaction
29      buyerMoney.deposit(price,escrowMoney)
30      sellerGoods.deposit(amt,escrowGoods)
31  }
```


TRUST THE COMPUTER
THE COMPUTER IS YOUR FRIEND!
quickmeme.com

```
19      res = escrowGoods.deposit(amt, sellerGoods)
20      if (!res) then
21          // insufficient goods in sellerGoods
22          // or different goods mixed
23          // in the goods transaction
24          buyerMoney.deposit(price, escrowMoney)
25          return false }
26
27      // price in escrowMoney, amt in escrowGoods
28      // now complete the transaction
29      buyerMoney.deposit(price, escrowMoney)
30      sellerGoods.deposit(amt, escrowGoods)
31  }
```

Who stole my money?

Grant Matcher

The Left Hand of Equals
Noble, Black, Bruce, Homer, Miller

Grant Matcher

$ $

The Left Hand of Equals
Noble, Black, Bruce, Homer, Miller

Grant Matcher

$ $

C C

The Left Hand of Equals
Noble, Black, Bruce, Homer, Miller

```
method match (
    aliceDonation : Money, aliceCharity : Charity,
    danaDonation : Money, danaCharity : Charity)
        -> Boolean {
  if (aliceDonation.amount
                .equals(danaDonation.amount)
       && aliceCharity == danaCharity)

     then {
       aliceCharity.accept(aliceDonation)
       danaCharity.accept(danaDonation)
       return true}
     else {return false}
}
```

The Left Hand of Equals

Noble, Black, Bruce, Homer, Miller

```
method match (
    aliceDonation : Money, aliceCharity : Charity,
    danaDonation : Money, danaCharity : Charity)
        -> Boolean {
  if (aliceDonation.amount
                .equals(danaDonation.amount)
       && aliceCharity.equals(danaCharity))

     then {
       aliceCharity.accept(aliceDonation)
       danaCharity.accept(danaDonation)
       return true}
     else {return false}
}
```

The Left Hand of Equals

Noble, Black, Bruce, Homer, Miller

Grant Matcher

```
class fakeCharity → Charity {
    def backPocket : Account = ...
    method equals(other : Object) { true }
    method accept(donation : Money) {
        { backPocket.accept(donation) }
}
```

The Left Hand of Equals
Noble, Black, Bruce, Homer, Miller

Bank and Accounts

Pol_1: With two accounts of same bank can transfer money.

Pol_2: Only someone with the Bank of a given currency can violate conservation of that currency

Pol_3: Only the bank can inflate own currency

Pol_4: No one can affect the balance of an account they do not have.

Pol_5: Balances are always non-negative.

Pol_6: A reported successful deposit can be trusted as much as one trusts the account one is depositing to.

[Miller et al, Financial Crypto 2000]

```
method match (
    aliceDonation : Money, aliceCharity : Charity,
    danaDonation : Money, danaCharity : Charity)
        -> Boolean {
  if (aliceDonation.amount
                .equals(danaDonation.amount)
       && aliceCharity.equals(danaCharity)
       && danaCharity.equals(aliceCharity))
     then {
       aliceCharity.accept(aliceDonation)
       danaCharity.accept(danaDonation)
       return true}
     else {return false}
}
```


The Left Hand of Equals
Noble, Black, Bruce, Homer, Miller

obeys

```
res=dest.deposit(amt, src)
```

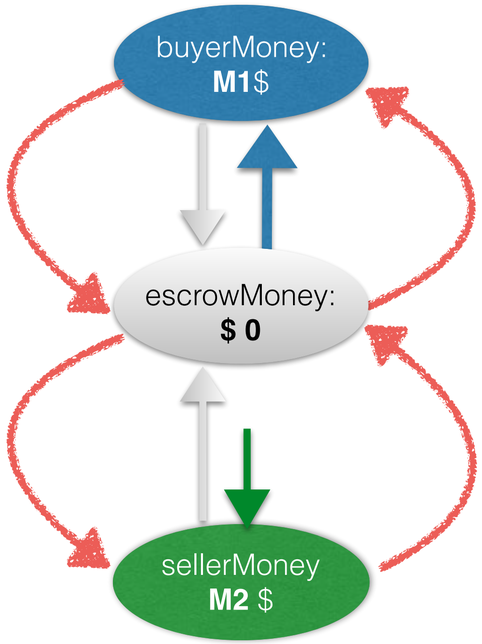
~~res → src **obeys** ValidPurse~~

absolute

res ∧ dest **obeys** ValidPurse → src **obeys** ValidPurse

conditional

Trust



escrowMoney = sellerMoney.sprout()

// sellerMoney **obeys** ValidPurse → escrowMoney **obeys** ValidPurse

res= escrowMoney. deposit (buyerMoney,**0**)

// **res=true** ∧ escrowMoney **obeys** ValidPurse

// → buyerMoney **obeys** ValidPurse

if !res then exit // sellerMoney **obeys** ValidPurse →

// ¬(buyerMoney **obeys** ValidPurse)

// sellerMoney **obeys** ValidPurse → buyerMoney **obeys** ValidPurse

res= buyerMoney. deposit (escrowMoney,**0**)

// **res=true** ∧ buyerMoney **obeys** ValidPurse

// → escrowMoney **obeys** ValidPurse

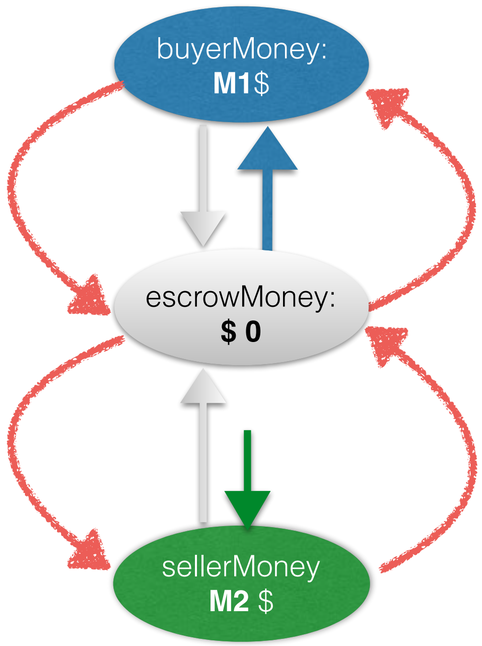
if !res then exit

res= escrowMoney. deposit (buyerMoney,**0**)

if !res then exit

// buyerMoney **obeys** ValidPurse ↔ seller **obeys** ValidPurse

Risk




```
escrowMoney =  sellerMoney.sprout()
        // ∀p. p obeys_PRE ValidPurse→
        //      [ p.balance_PRE=p.balance   ∨
        //            MayAccess_PRE(sellerMoney,p) ∧ ¬(sellerMoney obeys ValidPurse) ]
res= escrowMoney. deposit (buyerMoney,0)
        // ....

 if !res then exit        // ....

 res= buyerMoney. deposit (escrowMoney,0)
        //∀p. p obeys_PRE ValidPurse →
        //   [ p.balance_PRE=p.balance   ∨
        //     MayAccess_PRE(sellerMoney,p) ∧ ¬(sellerMoney obeys ValidPurse) ∨
        //     MayAccess_PRE(buyerMoney,p) ∧ ¬(buyerMoney obeys ValidPurse)  ]

 if !res then exit      // ....

 res= escrowMoney. deposit (buyerMoney,0)
        // ....
 if !res then exit

        //∀p. p obeys_PRE ValidPurse →
        //   [ p.balance_PRE=p.balance   ∨
        //     MayAccess_PRE(sellerMoney,p) ∧ ¬(sellerMoney obeys ValidPurse) ∨
        //     MayAccess_PRE(buyerMoney,p) ∧ ¬(buyerMoney obeys ValidPurse)  ]
```

Conclusions

- Necessary Conditions
- *Chainmail*
- Risk
- Trust