Peer-to-peer Affine Commitment using Bitcoin

Karl Crary and **Michael J. Sullivan**

Carnegie Mellon University
PLDI '15, Portland

June 17, 2015

Massively Multiplayer Online Linear Logic

Karl Crary and **Michael J. Sullivan**

Carnegie Mellon University
PLDI '15, Portland

June 17, 2015

Typecoin

- A general peer-to-peer commitment mechanism - using the language of linear logic
- Implemented on top of the Bitcoin network
- With applications for proof-carrying authorization

Proof-carrying authorization

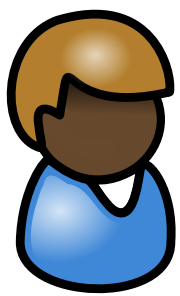
- Idea: represent authorization as logical propositions (Appel and Felten 1999)

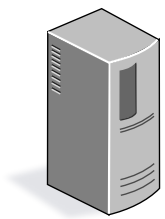
Proof-carrying authorization

- Idea: represent authorization as logical propositions (Appel and Felten 1999)
- ... in a logic with a notion of affirmation
- $\langle K \rangle A$ means "the principal K says A"

Proof-carrying authorization

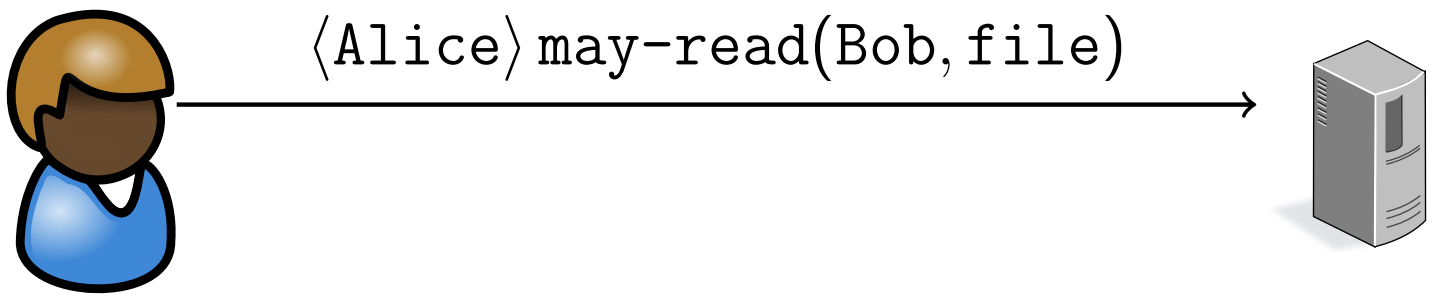
- Alice wants to give access to a file, so affirms:
  - $\langle \texttt{Alice} \rangle\, \texttt{may-read}(\texttt{Bob}, \texttt{file})$
  - $\langle \texttt{Alice} \rangle\, \texttt{may-read}(\texttt{Charlie}, \texttt{file})$





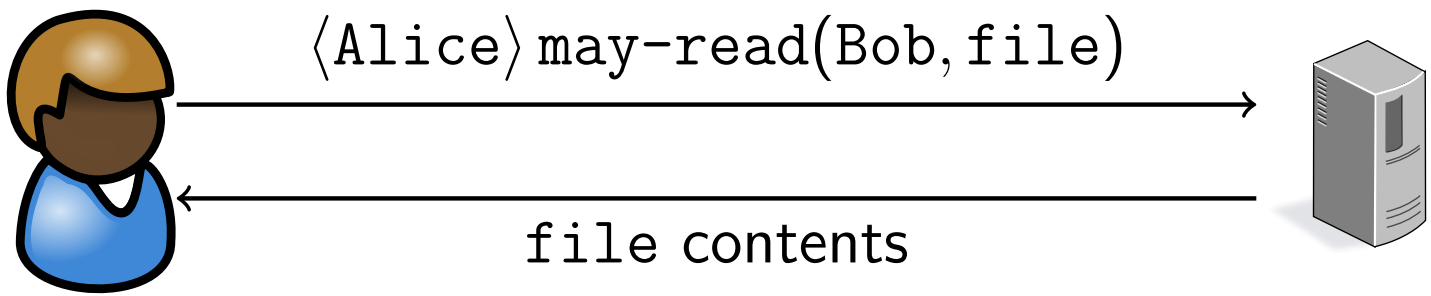
Proof-carrying authorization

- Alice wants to give access to a file, so affirms:
  - $\langle \texttt{Alice} \rangle\, \texttt{may-read}(\texttt{Bob}, \texttt{file})$
  - $\langle \texttt{Alice} \rangle\, \texttt{may-read}(\texttt{Charlie}, \texttt{file})$



Proof-carrying authorization

- Alice wants to give access to a file, so affirms:
  - $\langle \texttt{Alice} \rangle\, \texttt{may-read}(\texttt{Bob}, \texttt{file})$
  - $\langle \texttt{Alice} \rangle\, \texttt{may-read}(\texttt{Charlie}, \texttt{file})$



Proof-carrying authorization - higher order use

- Much more flexible policies are possible:

$$\langle \texttt{Alice} \rangle \forall K.\, \langle \texttt{Registrar} \rangle\, \texttt{in-Alice's-class}(K) \supset \texttt{may-read}(K, \texttt{file})$$

Proof-carrying authorization - higher order use

- Much more flexible policies are possible:

$$\langle \texttt{Alice} \rangle \forall K.\, \langle \texttt{Registrar} \rangle\, \texttt{in-Alice's-class}(K) \supset \texttt{may-read}(K, \texttt{file})$$

- Then can derive:

$$\forall K.\, \langle \texttt{Registrar} \rangle\, \texttt{in-Alice's-class}(K) \supset \langle \texttt{Alice} \rangle \texttt{may-read}(K, \texttt{file})$$

Implementing proof-carrying authorization

- Straightforward to make work even in a decentralized/peer-to-peer system
- Proofs are self-contained
- Digital signatures used for affirmation

Consumable credentials

What if we want *one time use* authorization?

Linear logic

- Garg et al. 2006; *linear* proof-carrying authorization
- Linear logic treats hypotheses as scarce resources that must be used once

For logicians

Linear logic allows *exchange*, but not *weakening* or *contraction*

Linear logic

- Garg et al. 2006; *linear* proof-carrying authorization
- Linear logic treats hypotheses as scarce resources that must be used once
- Good for modeling state change:

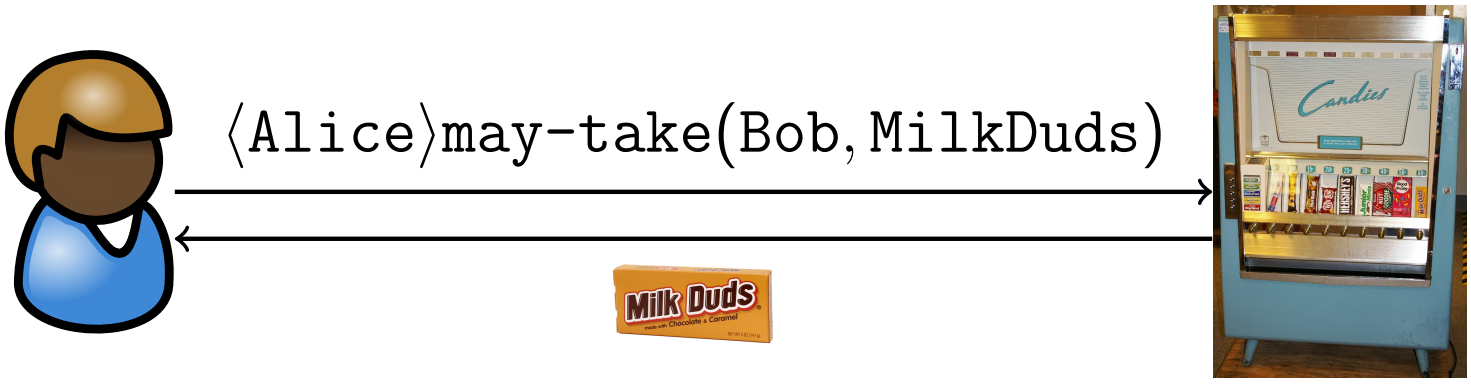
$$\texttt{bread} \otimes \texttt{ham} \multimap \texttt{ham sandwich}$$

$$\forall i.\, \texttt{counter}(i) \multimap \texttt{counter}(i+1)$$

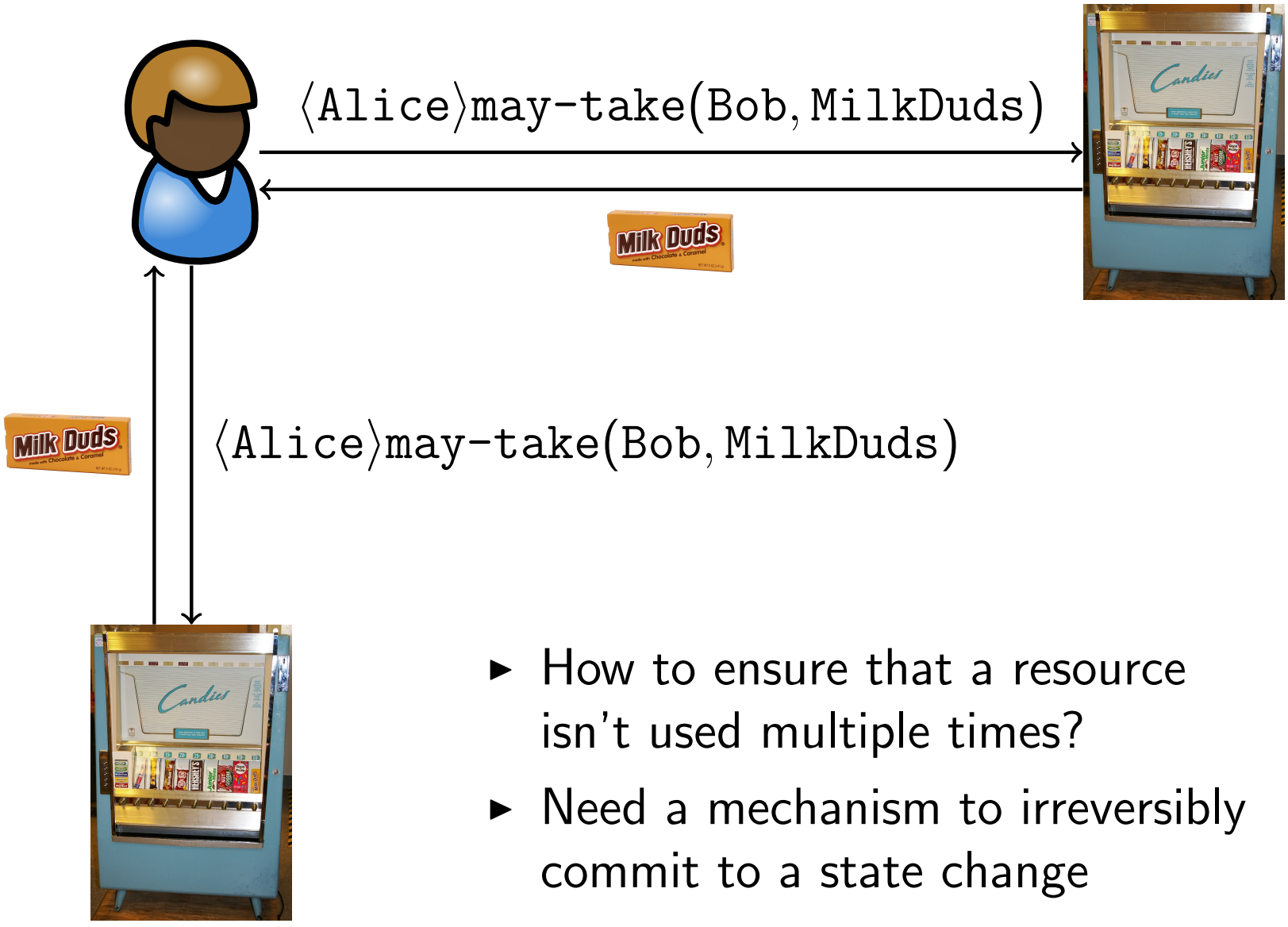
For logicians

Linear logic allows *exchange*, but not *weakening* or *contraction*

Linear authorization



Linear authorization



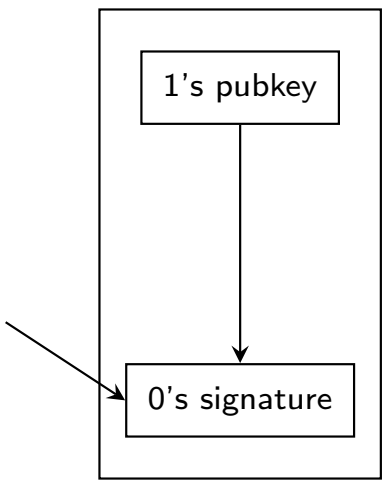
- How to ensure that a resource isn't used multiple times?
- Need a mechanism to irreversibly commit to a state change

Bitcoin

- On a completely different note: consider designing a decentralized digital currency

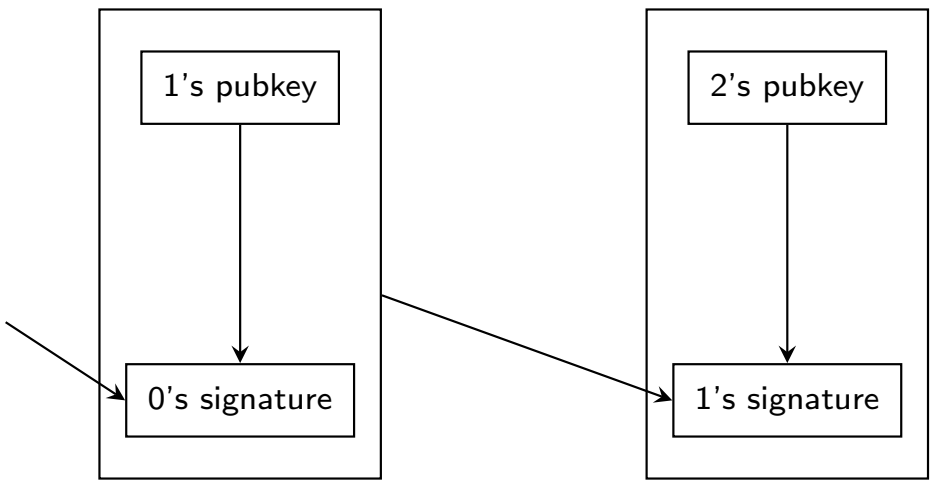
Bitcoin

- On a completely different note: consider designing a decentralized digital currency
- A coin is a chain of digital certificates
- A coin is spent by signing it over to somebody else



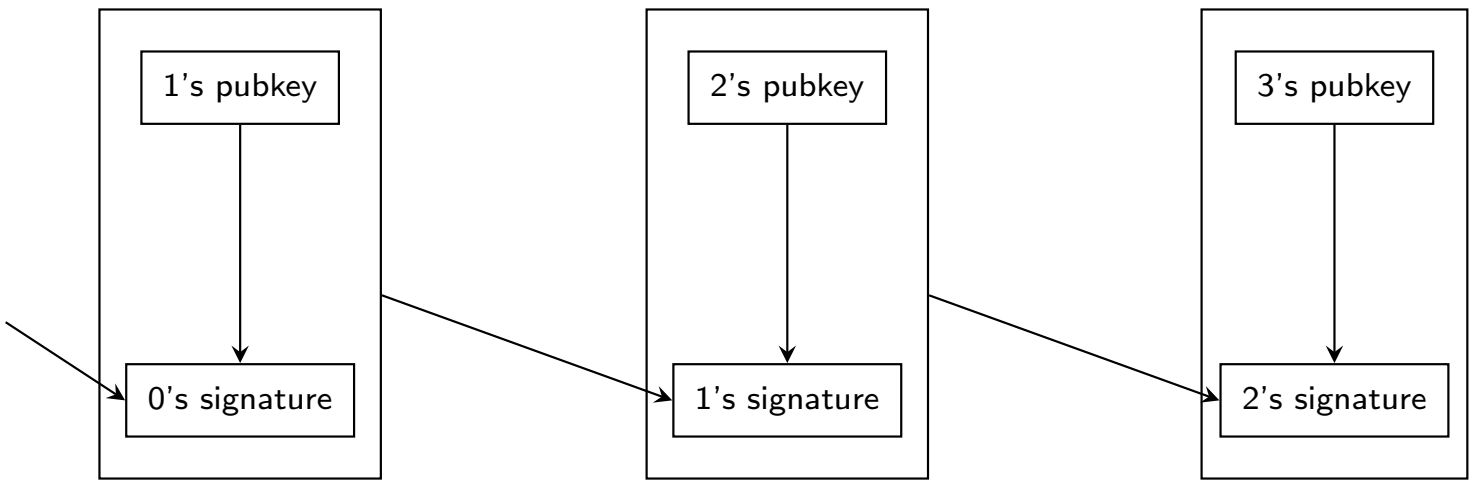
Bitcoin

- On a completely different note: consider designing a decentralized digital currency
- A coin is a chain of digital certificates
- A coin is spent by signing it over to somebody else

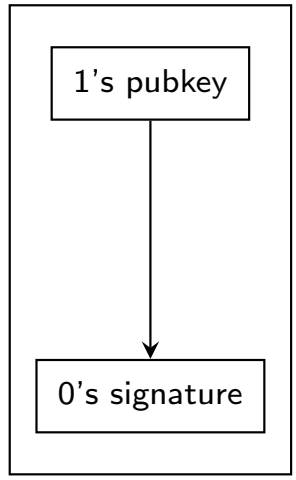


Bitcoin

- On a completely different note: consider designing a decentralized digital currency
- A coin is a chain of digital certificates
- A coin is spent by signing it over to somebody else

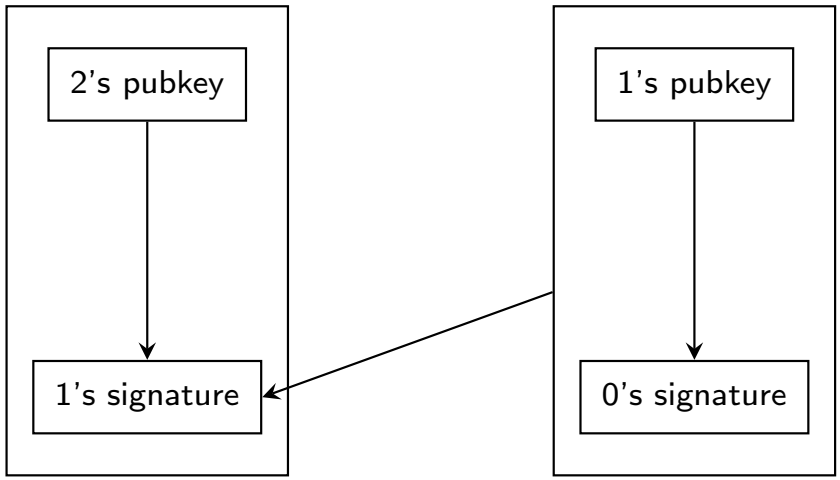


Bitcoin - the catch



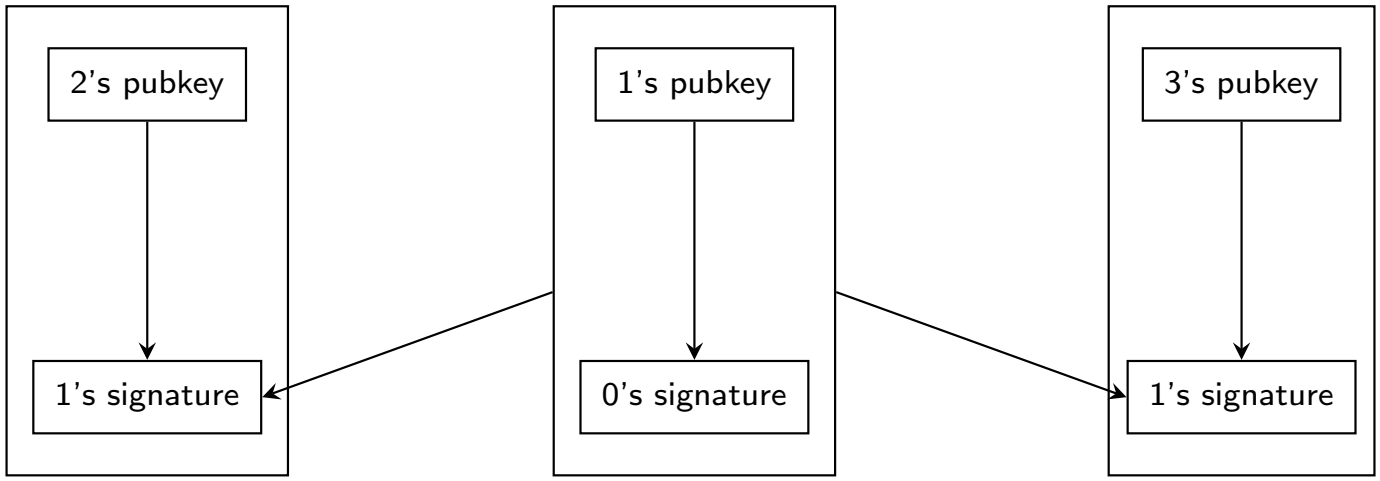
- But how do we prevent an owner from spending a coin multiple times?

Bitcoin - the catch



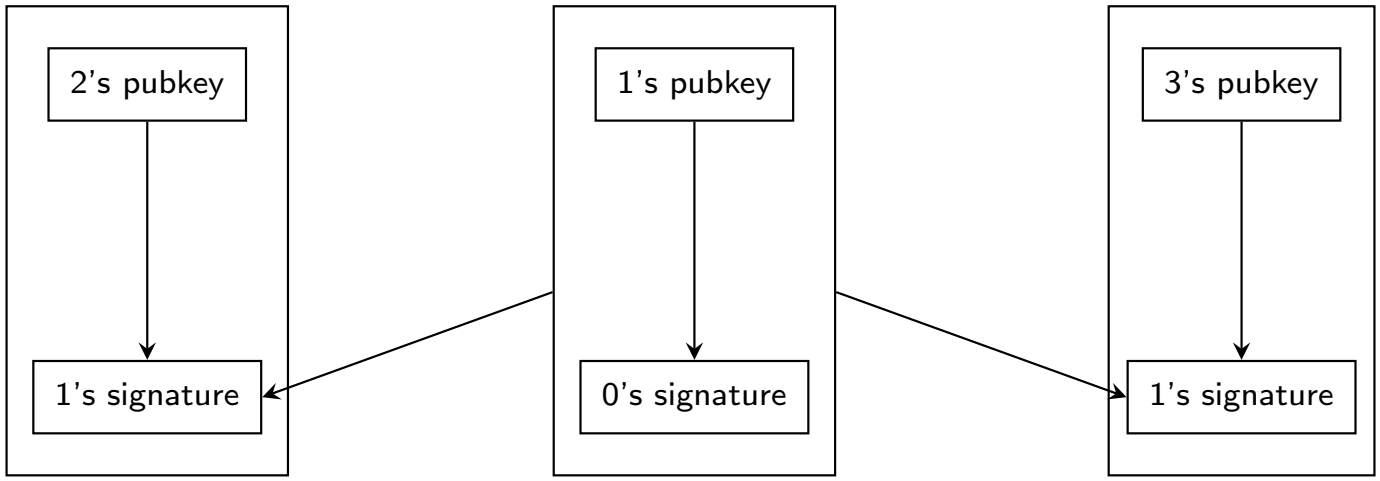
- But how do we prevent an owner from spending a coin multiple times?

Bitcoin - the catch



- But how do we prevent an owner from spending a coin multiple times?

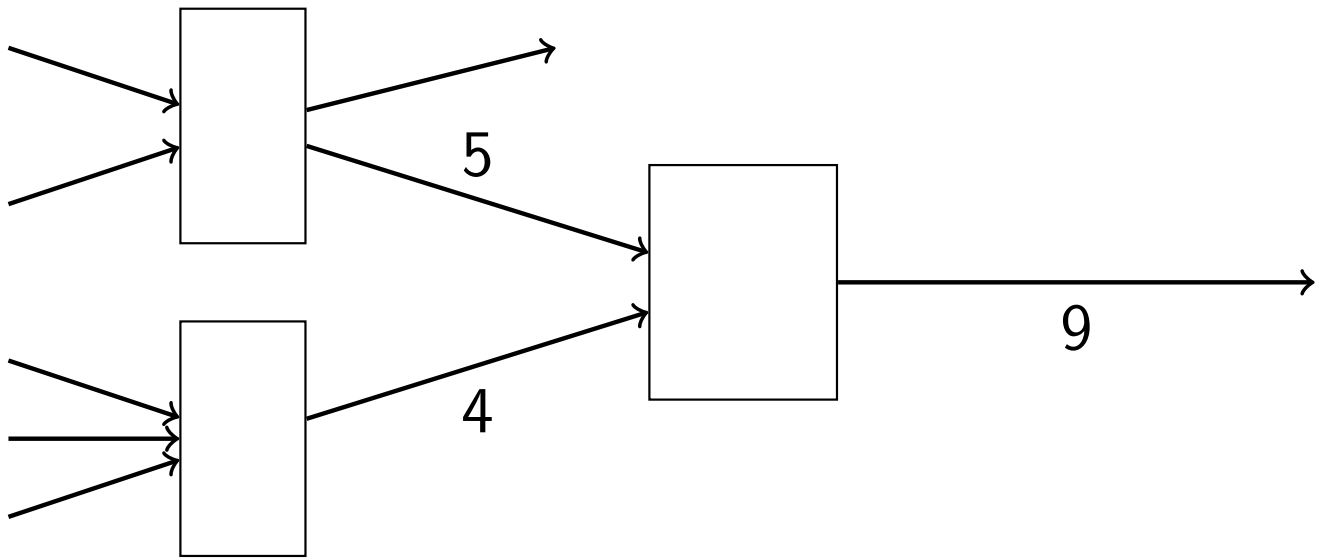
Bitcoin - the catch

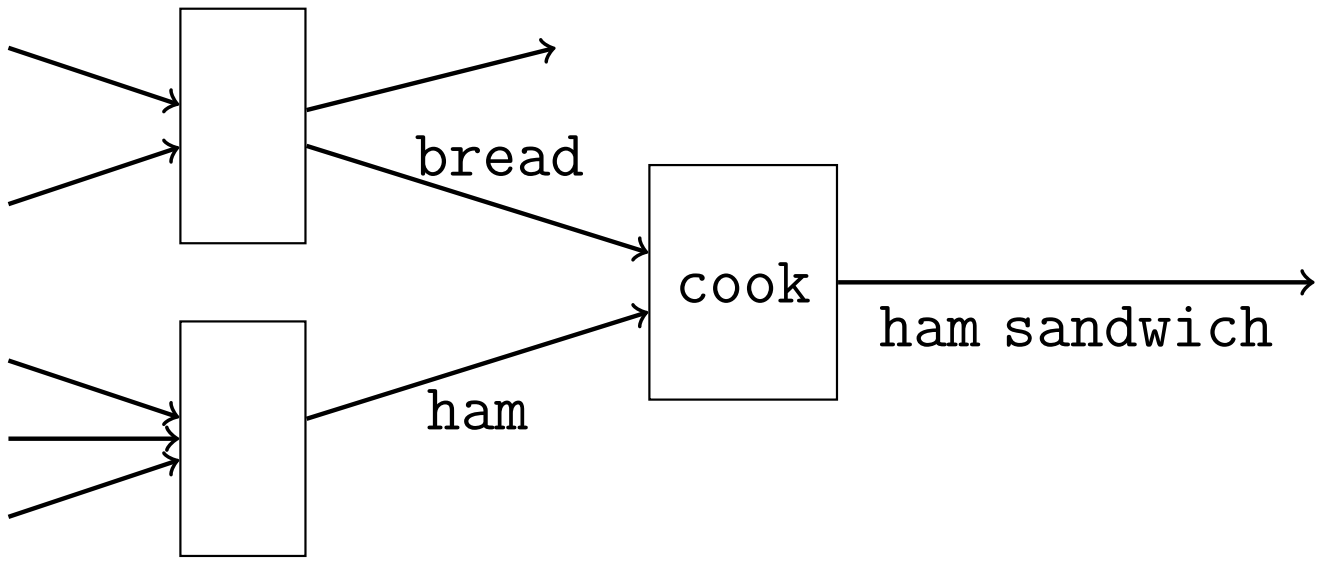


- But how do we prevent an owner from spending a coin multiple times?
- Need a mechanism to irreversibly commit to a state change

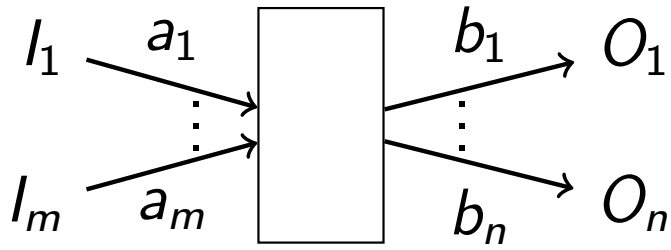
Bitcoin implementation

- Bitcoin (Nakamoto 2008) does this with a global ledger of all transactions - the "blockchain"
- Ledger maintained by distributed process called "mining"


5
4
9

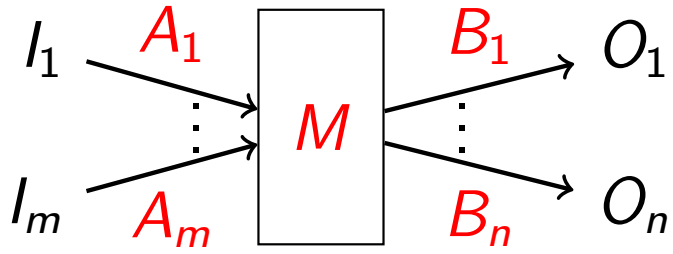

bread
cook
ham sandwich
ham

From Bitcoin to Typecoin - transactions



- $a_1 + \cdots + a_m = b_1 + \cdots + b_n$

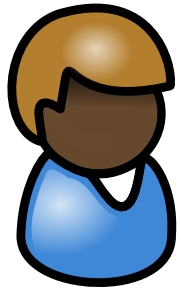
From Bitcoin to Typecoin - transactions



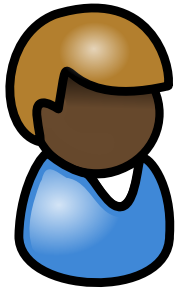
- $\vdash M : (A_1 \otimes \cdots \otimes A_m) \multimap (B_1 \otimes \cdots \otimes B_n)$
- Carry linear logic[1] propositions instead of numbers

[1] actually affine logic

Authorization example

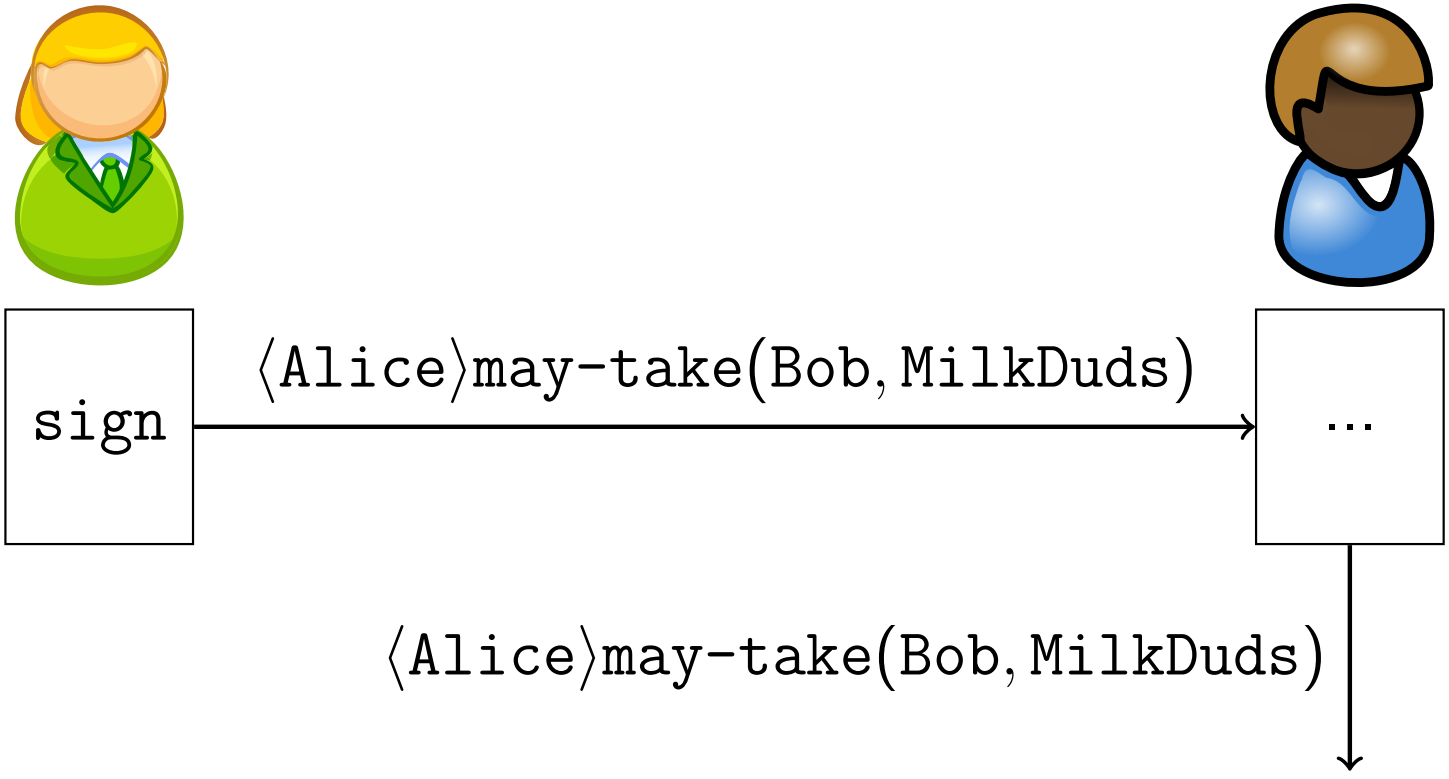


Authorization example

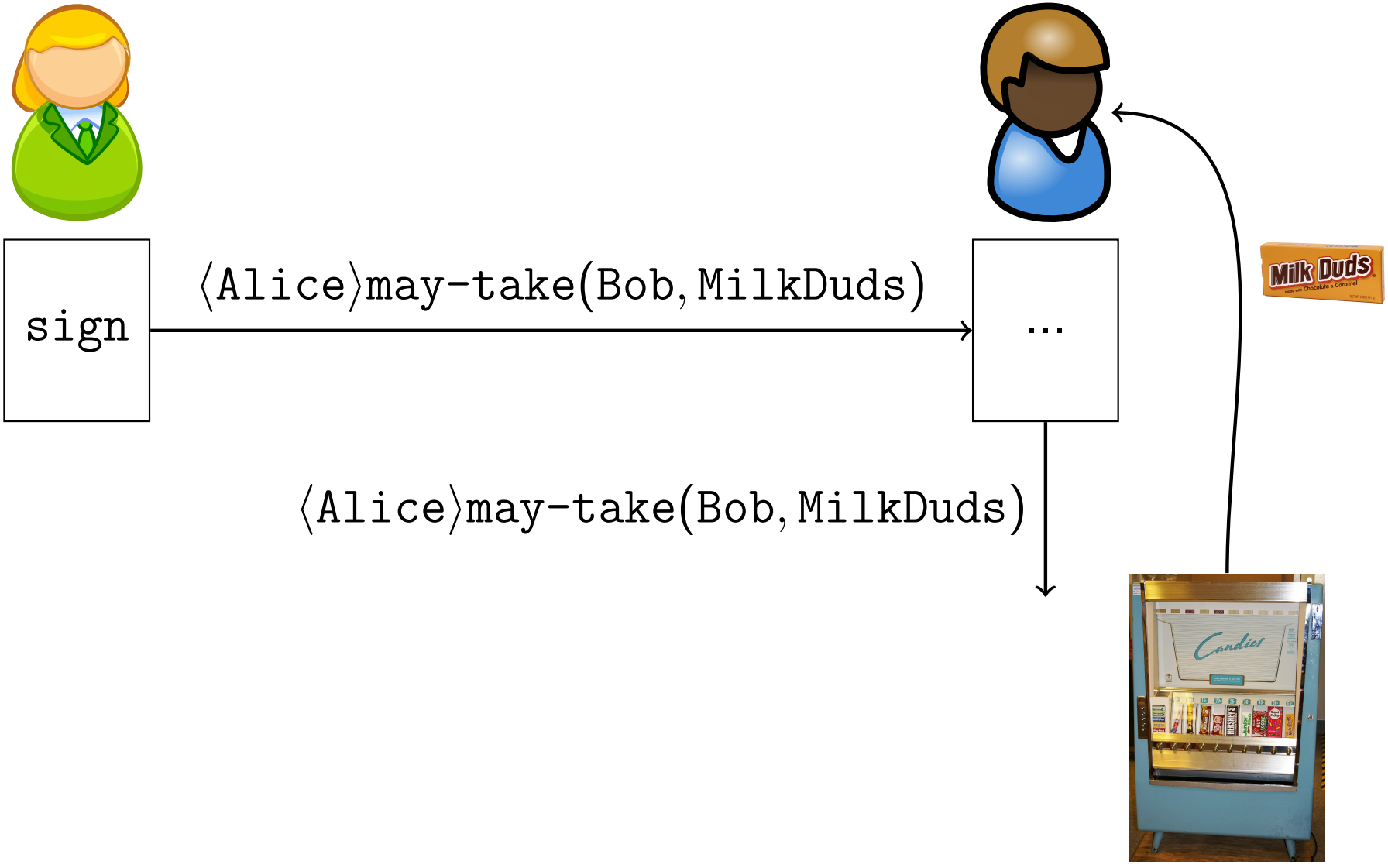


`sign` $\xrightarrow{\langle \texttt{Alice} \rangle \texttt{may-take}(\texttt{Bob}, \texttt{MilkDuds})}$

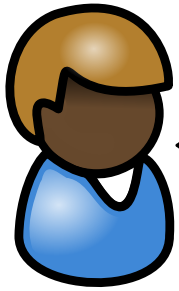
Authorization example



Authorization example



Authorization example



sign

$\langle \texttt{Alice} \rangle \forall K.\, \texttt{may-take}(K, \texttt{MilkDuds})$

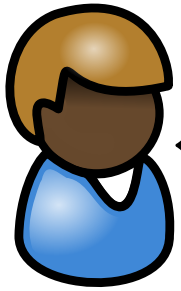
...

$\langle \texttt{Alice} \rangle \texttt{may-take}(\texttt{Bob}, \texttt{MilkDuds})$

- Quantification allows transferable permissions

Authorization example

$\langle \texttt{Alice} \rangle \forall K.\, \texttt{may-take}(K, \texttt{MilkDuds})$
$\& \, \texttt{may-take}(K, \texttt{Hershey's})$



$\texttt{sign}$

$\ldots$

$\langle \texttt{Alice} \rangle \texttt{may-take}(\texttt{Bob}, \texttt{MilkDuds})$

- Quantification allows transferable permissions
- External choice ("with") allows choice

Declarations

- Where do `may-take`, `MilkDuds`, etc. come from?

Declarations

- Where do `may-take`, `MilkDuds`, etc. come from?
- Transactions can declare types and propositions

$$\texttt{may-take} \quad : \quad \text{principal} \rightarrow \texttt{candy} \rightarrow \text{prop}$$

- Can turn Typecoin back into a currency (S-coins)

$$
\begin{array}{rcl}
\texttt{coin} & : & \mathsf{nat} \to \mathsf{prop} \\
\texttt{merge} & : & \forall\, N, M : \mathsf{nat}. \\
 & & \texttt{coin}\, N \otimes \texttt{coin}\, M \multimap \texttt{coin}\, N + M \\
\texttt{split} & : & \forall\, N, M, P : \mathsf{nat}. \\
 & & \texttt{coin}\, N + M \multimap \texttt{coin}\, N \otimes \texttt{coin}\, M
\end{array}
$$

- Need some way to mint a new S-coin

$$\begin{array}{lcl} \texttt{print} & : & \mathsf{nat} \to \mathsf{prop} \\ \texttt{issue} & : & \forall N{:}\mathsf{nat}.\, \langle \texttt{Janet} \rangle (\texttt{print}\, N) \multimap \texttt{coin}\, N \end{array}$$

How to implement?

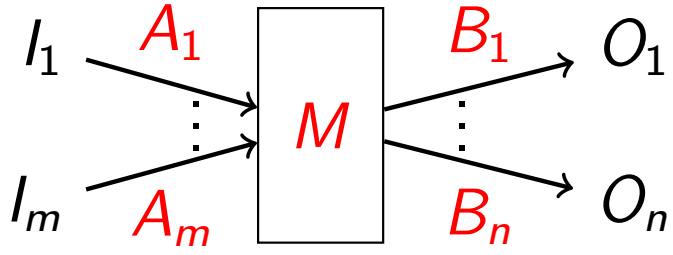
- We *could* build Typecoin in a standalone way
- Use adapted versions of the Bitcoin mining algorithms and protocol
- Could typecheck transactions before they enter the chain

How to implement?

- How to incentivize people to mine on a Typecoin chain?
- Bitcoin already has a lot of mining power
- Typechecking transactions in the chain not an obvious win: proofs might be big or not public

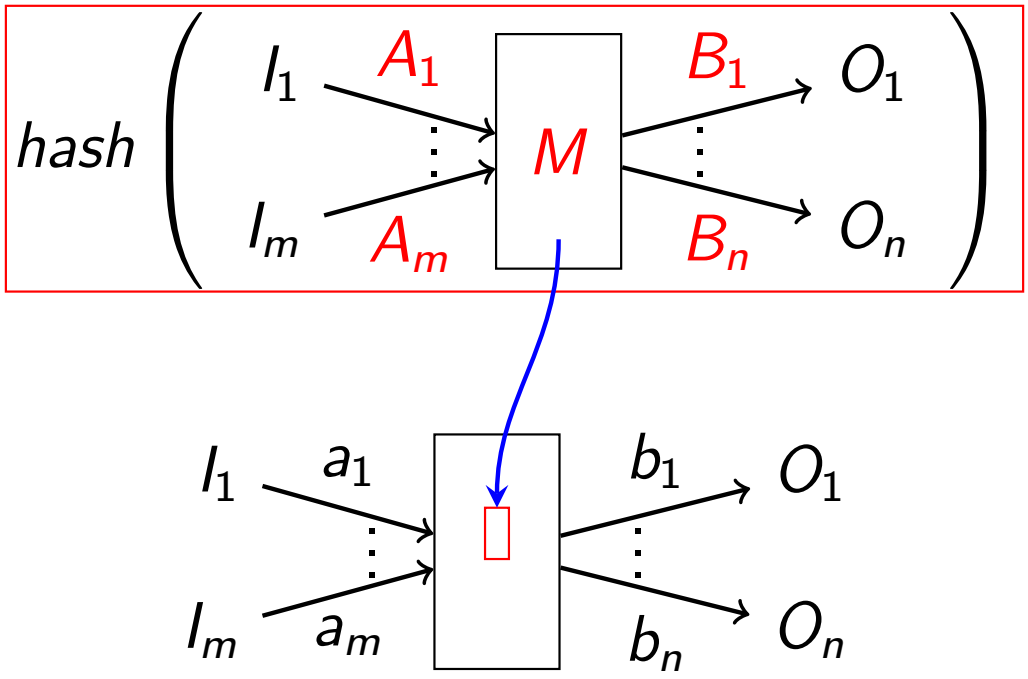
Overlaying on Bitcoin

- New plan: actually overlay on top of Bitcoin



Overlaying on Bitcoin

- New plan: actually overlay on top of Bitcoin



- Embed a hash in the metadata of the Bitcoin transaction
- Send the Typecoin transactions to interested parties

Metadata in Bitcoin

- Bitcoin historically lacked a nice place to put metadata - on principle
- (Nodes would not forward transactions that used the straightforward methods)
- Paper describes a somewhat hacky workaround

Metadata in Bitcoin

- Bitcoin historically lacked a nice place to put metadata - on principle
- (Nodes would not forward transactions that used the straightforward methods)
- Paper describes a somewhat hacky workaround
- But the Bitcoin developers have since caved

Receipts

- Receipts that attest to outputs: $\mathsf{receipt}(A \twoheadrightarrow \mathit{addr})$

$$\langle \texttt{Alice} \rangle (\mathsf{receipt}(\texttt{coin}(5) \twoheadrightarrow \texttt{Alice}) \multimap \forall K.\, \texttt{may-take}(K, \texttt{MilkDuds}))$$

Expiration/revocation

- Conditional modality permits revocation and expiration:

  $\mathsf{if}(\mathsf{before}(\text{July } 10)), \texttt{may-write}(\texttt{Alice}, \texttt{POPL-paper}))$

Implementation

- Implemented in Standard ML
- With a new Bitcoin client, in SML

Related Work

- Bowers et al. 2007; consumable credentials
- Rosenfeld 2013; colored coins
- Wood 2014; Ethereum

Conclusion

- Typecoin is a flexible peer-to-peer logical commitment mechanism
- Based on generalizing Bitcoin to carry logical propositions
- Actually implemented on top of Bitcoin
- Details on the logic are in the paper

Thank you!

Why not linear?

- Typecoin sort of *fundamentally* affine - can always throw away an output
- Allowing rule declarations in signatures makes it trivial
- `trash` : $\top \multimap 1$

Why not linear?

- Typecoin sort of *fundamentally* affine - can always throw away an output
- Allowing rule declarations in signatures makes it trivial
- $\texttt{trash} : \top \multimap 1$
- Prohibit $\top$? $\texttt{trash} : \texttt{A} \multimap 1$
- Prohibit proving 1? $\texttt{dummy} :$ prop. $\texttt{trash} : \texttt{A} \multimap !\texttt{dummy}$
- Prohibit consuming A? *trash* : $\langle K \rangle \texttt{dummy} \multimap !\texttt{dummy}$, sign $\langle K \rangle (A \multimap \texttt{dummy})$

Metadata: "m-of-n" outputs

- An "*m*-of-*n*" output lists *n* public keys
- To spend it, provide signatures using *m*
- 2-of-3 outputs useful for two-party escrow
- We use 1-of-2 outputs to embed metadata
- One public key is the real destination
- The other is actually the hash of our transaction