Security for Mobile Objects

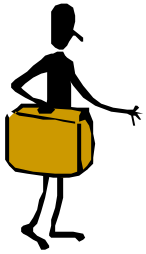
Will Harwood

Introduction

A mobile object is some code

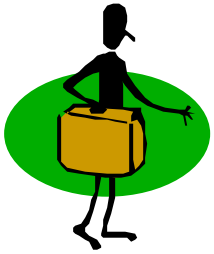
Introduction

A mobile object is some code that carries a state

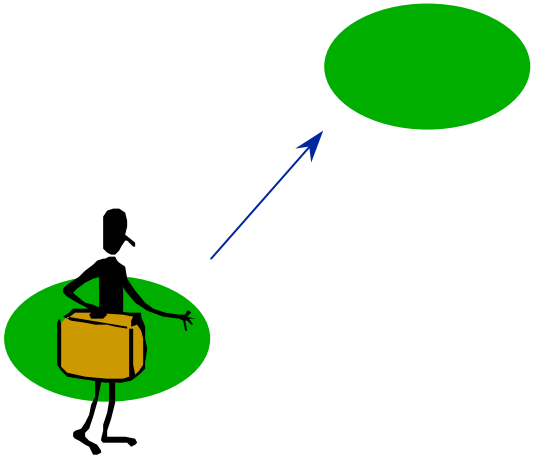


Introduction

A mobile object is some code
that carries a state
that lives on a host

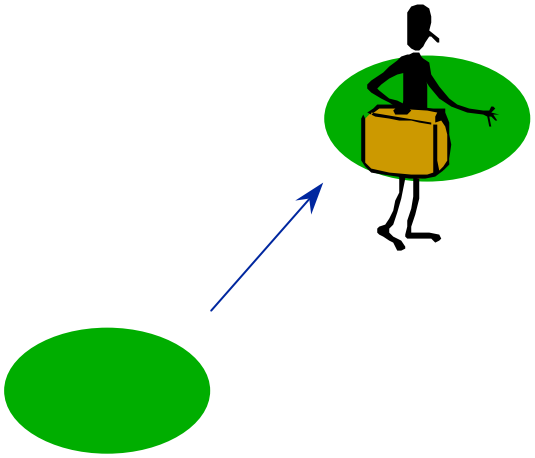


Introduction



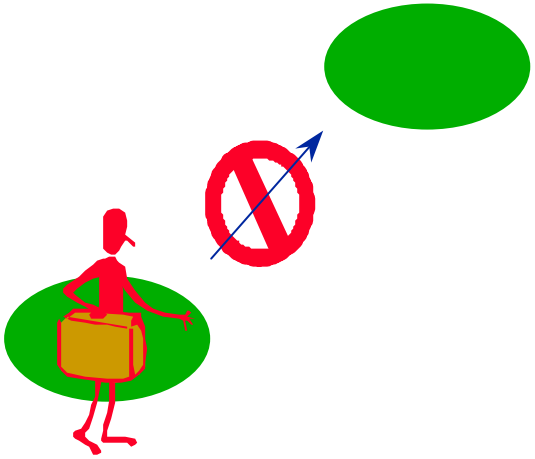
A mobile object is some code
that carries a state
that lives on a host
that visits places

Introduction



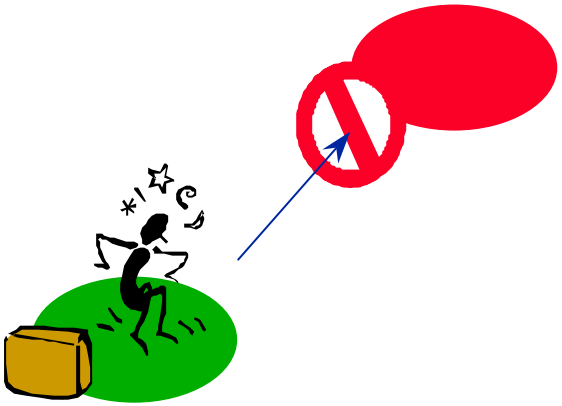
A mobile object is some code
that carries a state
that lives on a host
that visits places
which is let in when trusted

Introduction



A mobile object is some code
that carries a state
that lives on a host
that visits places
which is let in when trusted
and barred when untrusted

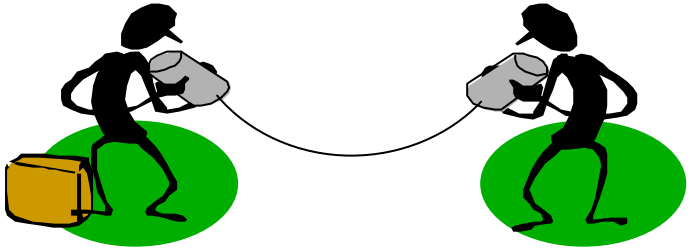
Introduction



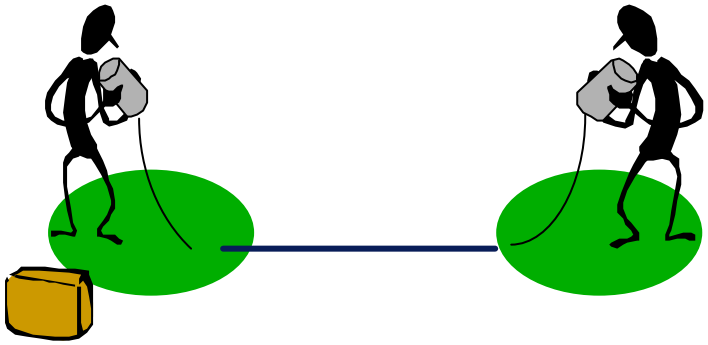
A mobile object is some code
that carries a state
that lives on a host
that visits places
which is let in when trusted
and barred when untrusted
and will refuse to go to untrustworthy places

Communication

Mobile objects can talk to their friends

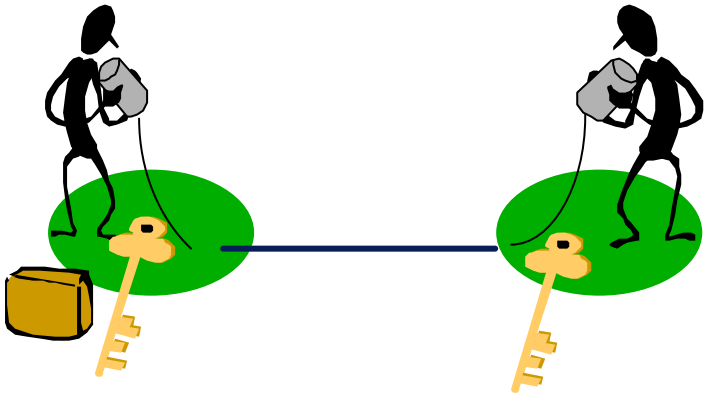


Communication



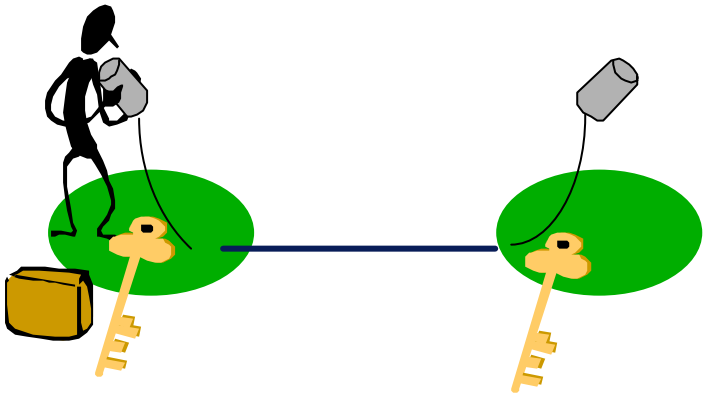
Mobile objects can talk to their friends
but only by co-operation of the hosts

Communication



Mobile objects can talk to their friends
but only by co-operation of the hosts.
In fact they expose their keys to their host.

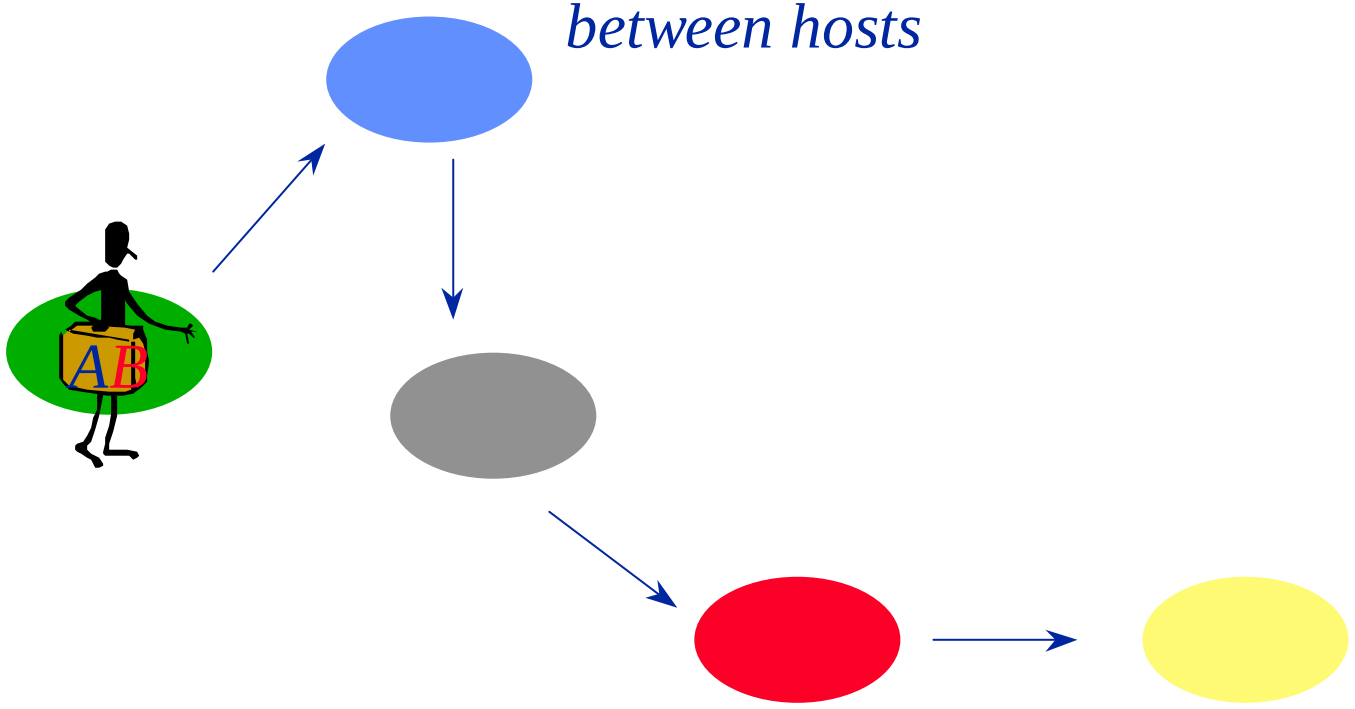
Communication



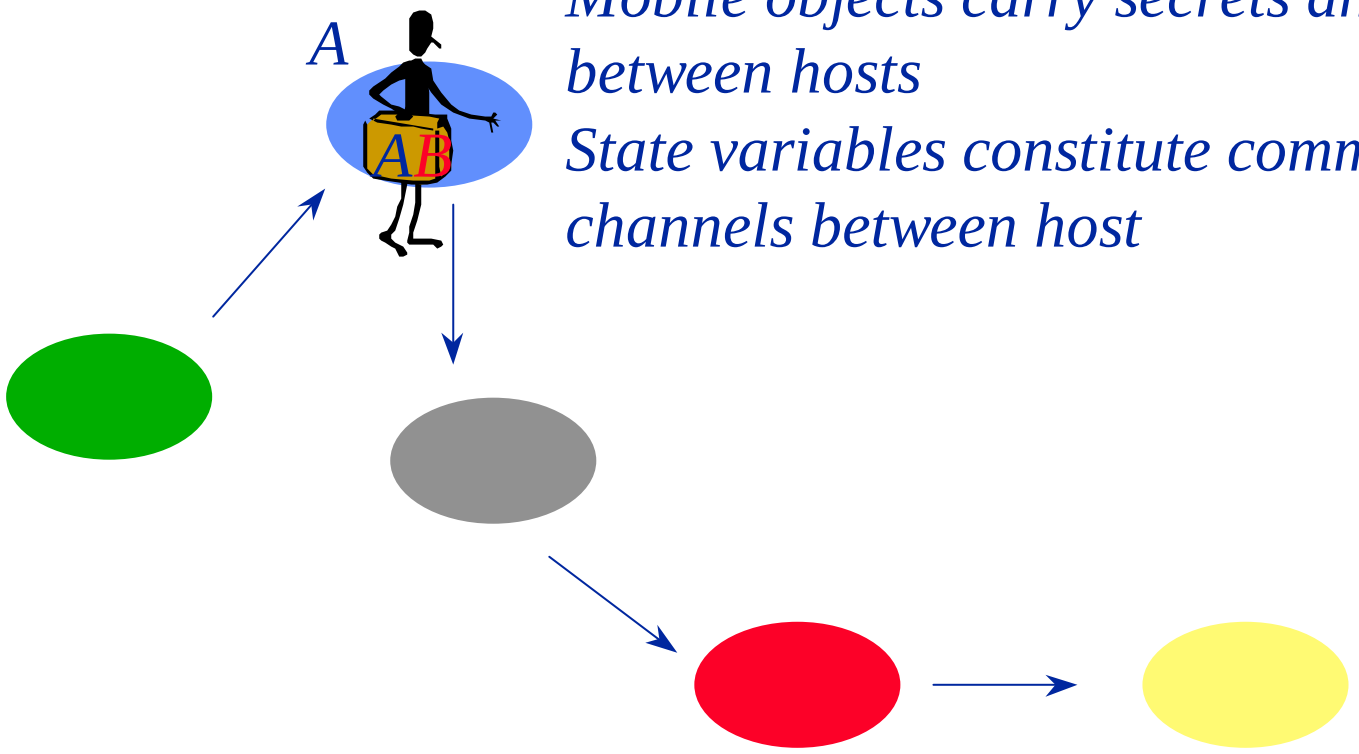
Mobile objects can talk to their friends
but only by co-operation of the hosts.
In fact they expose their keys to their host.
Which means a host can fake communication
from an object even when the object has left.

Movement, Secrets and Promises

Mobile objects carry secrets and promises between hosts



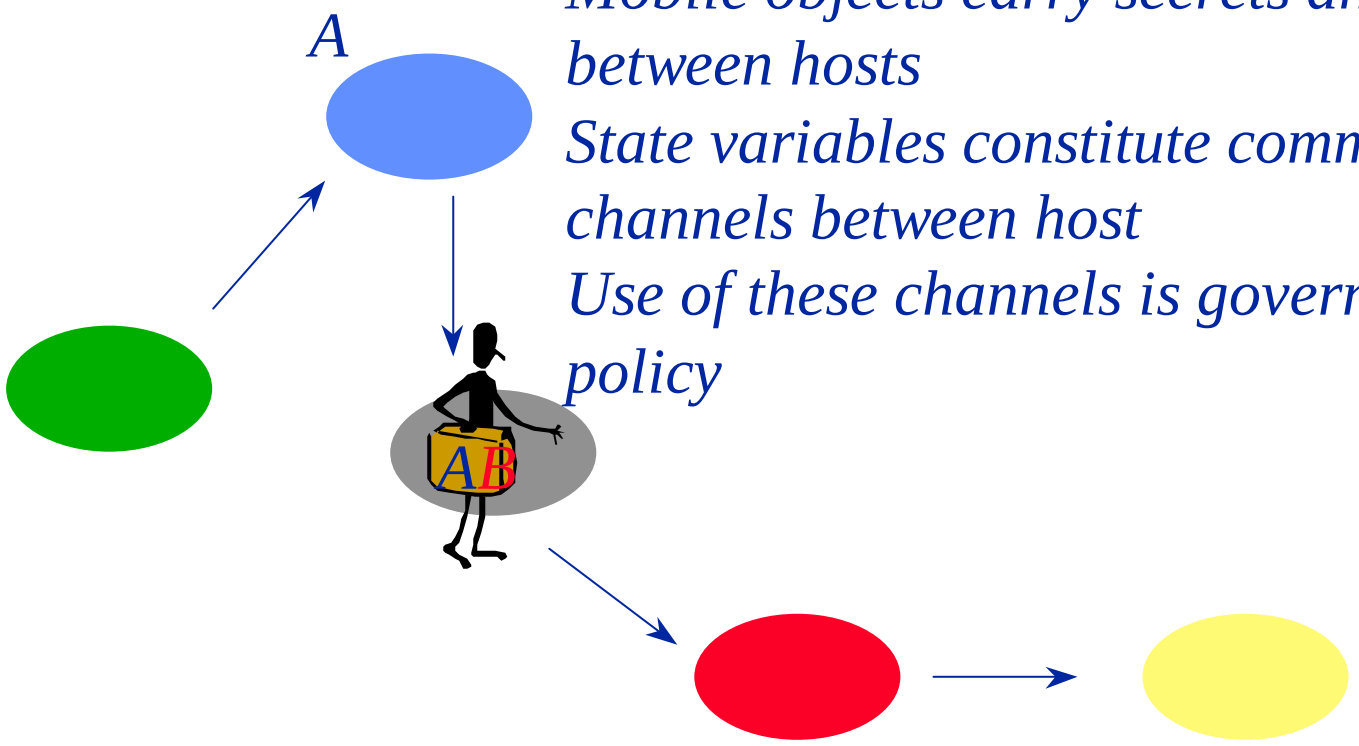
Movement, Secrets and Promises



Mobile objects carry secrets and promises between hosts

State variables constitute communication channels between host

Movement, Secrets and Promises

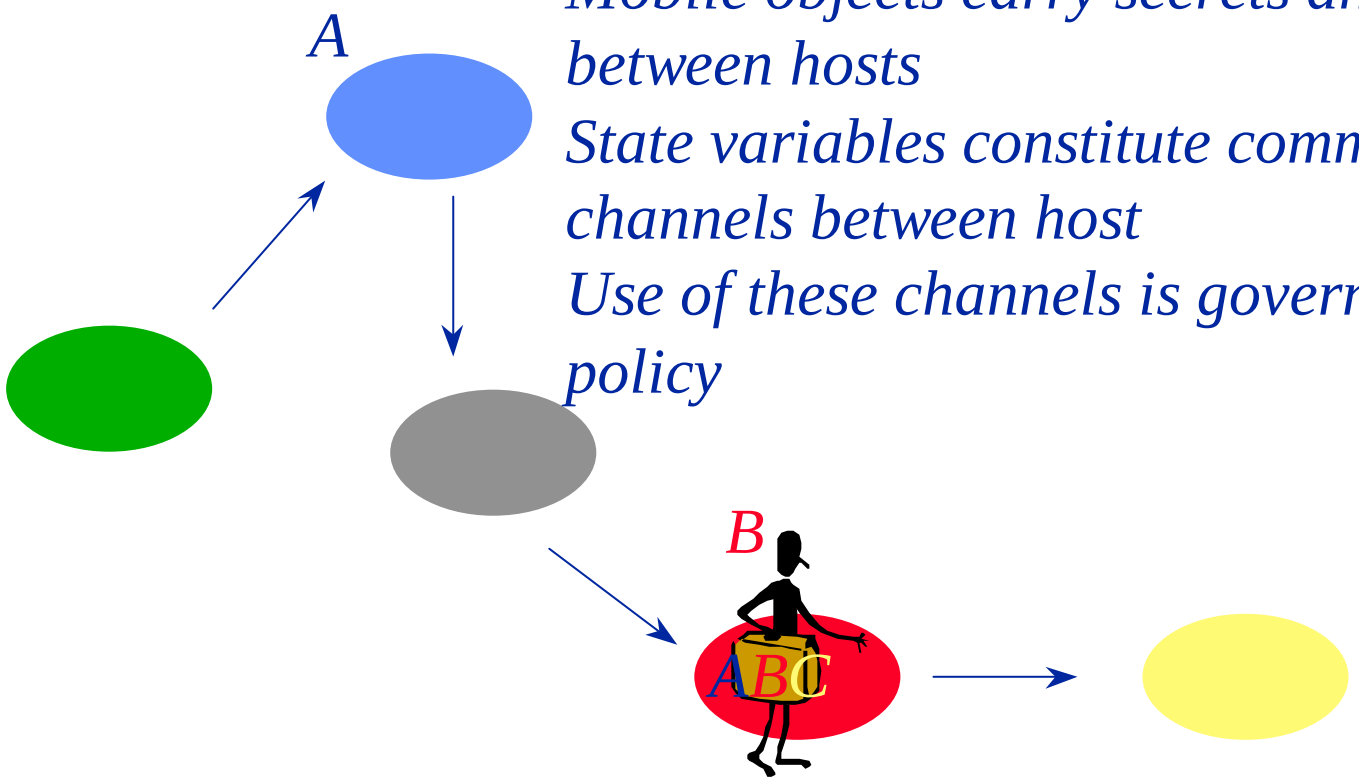


Mobile objects carry secrets and promises between hosts

State variables constitute communication channels between host

Use of these channels is governed by a policy

Movement, Secrets and Promises

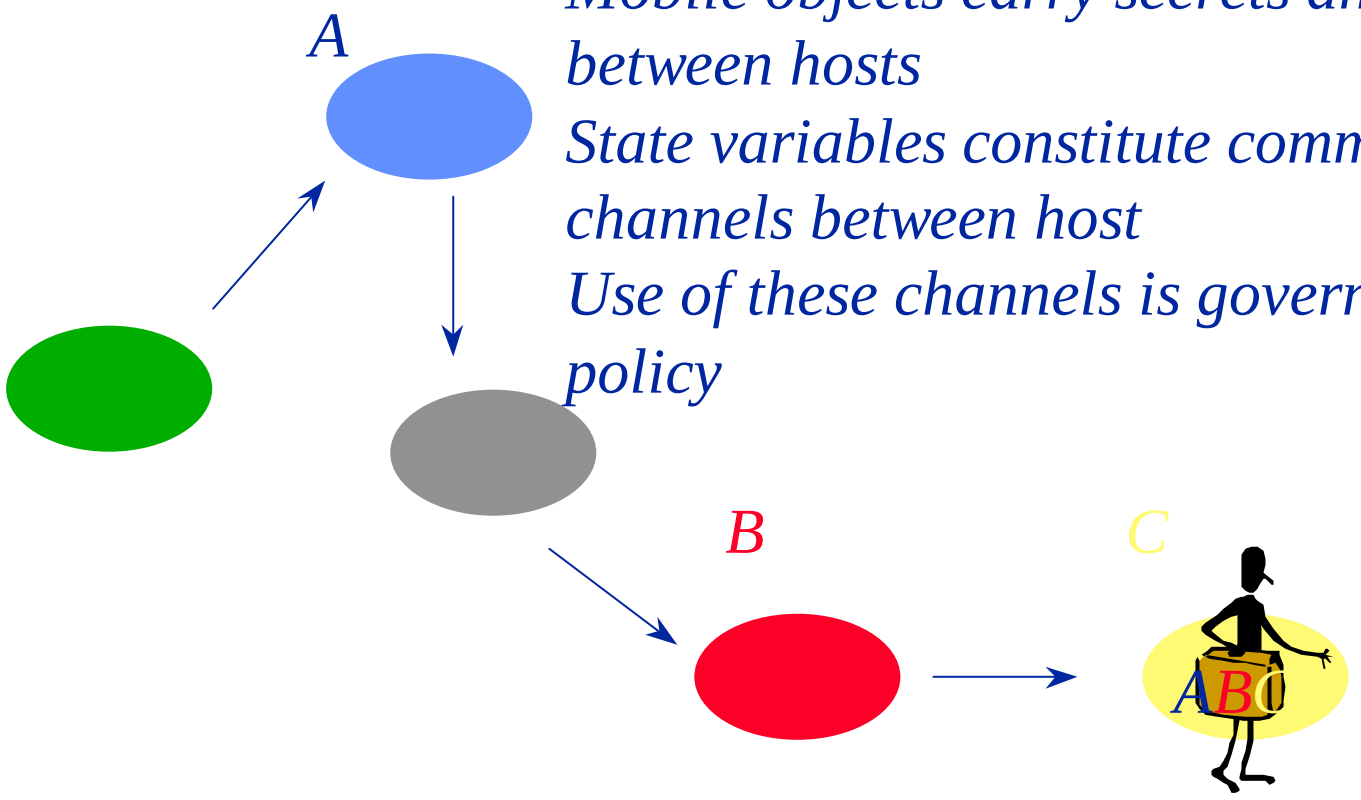


Mobile objects carry secrets and promises between hosts

State variables constitute communication channels between host

Use of these channels is governed by a policy

Movement, Secrets and Promises



Mobile objects carry secrets and promises between hosts

State variables constitute communication channels between host

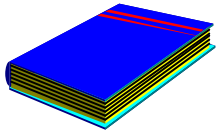
Use of these channels is governed by a policy

What's Really Going On?

What's Really Going On?

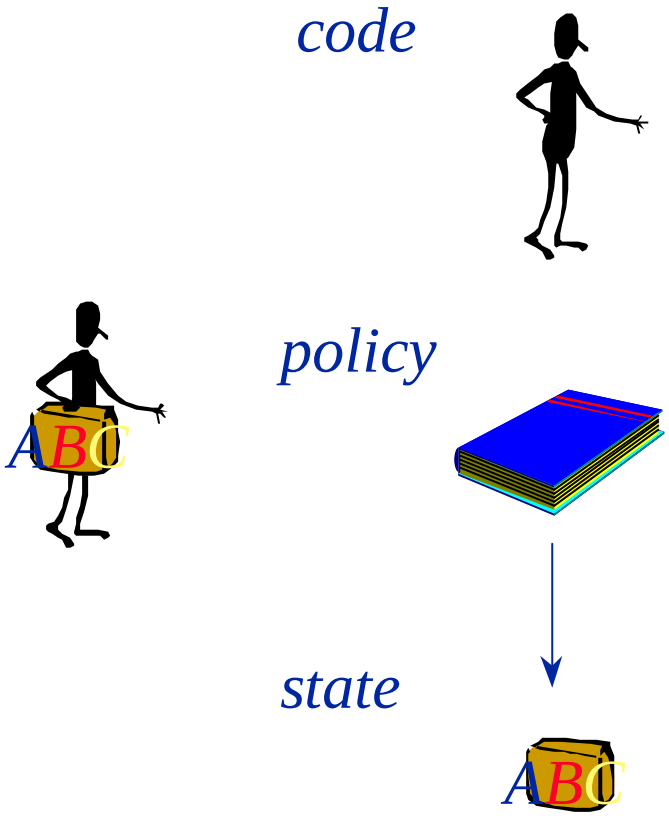
code

policy

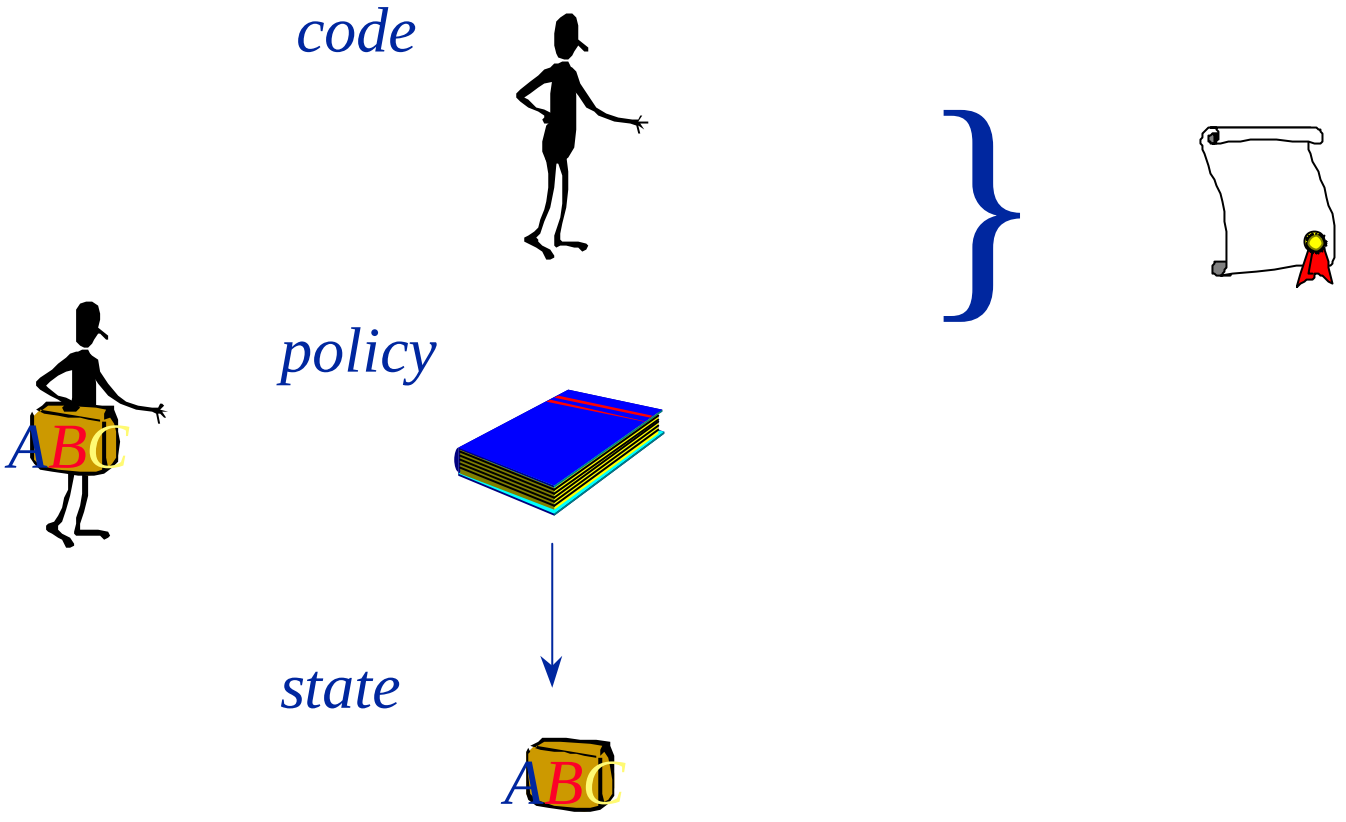


state

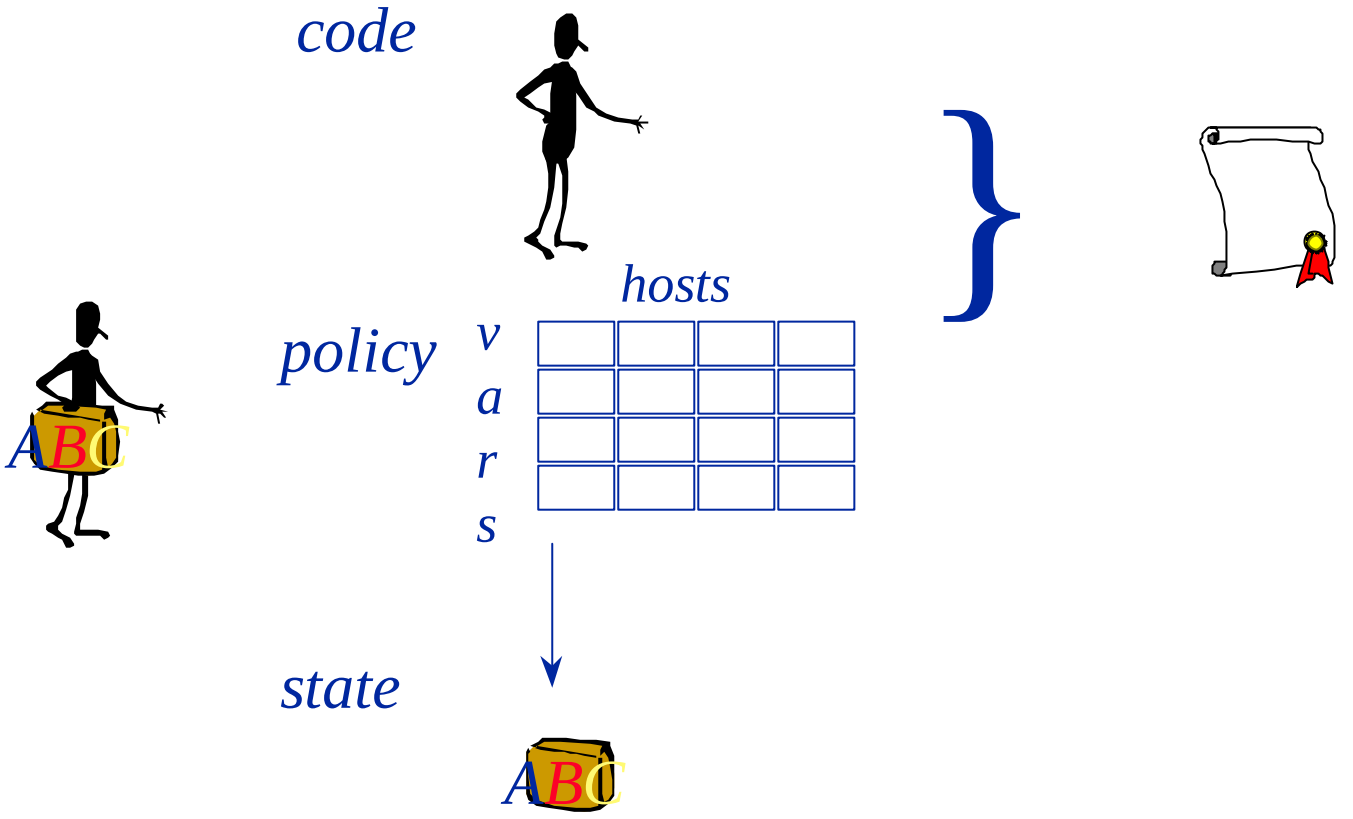
What's Really Going On?



What's Really Going On?



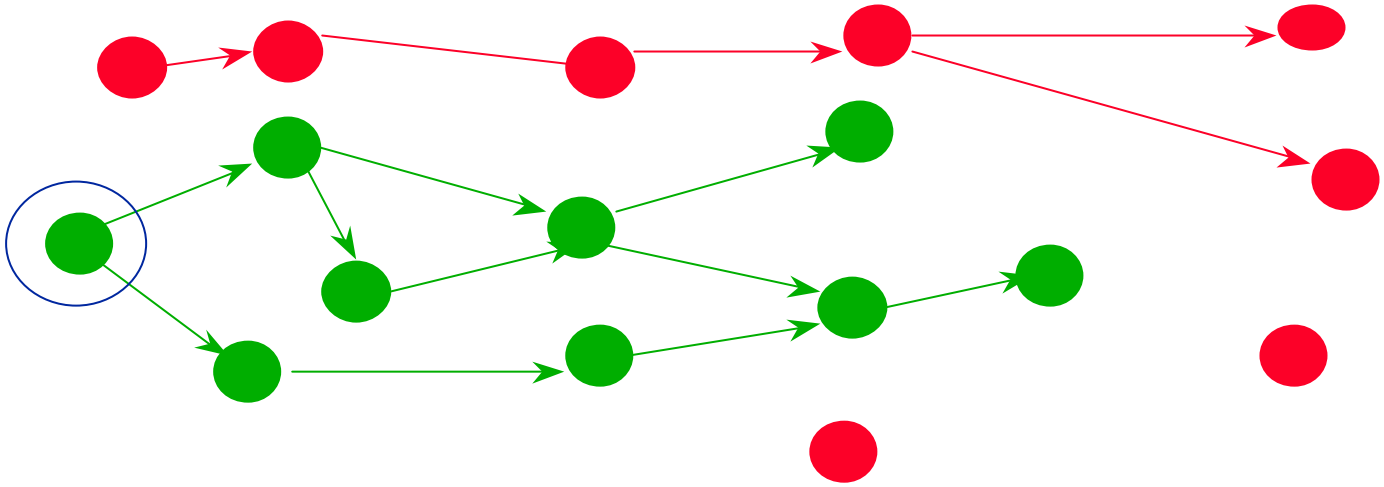
What's Really Going On?



Access Control Matrices

- Access control matrices define a set of secure transitions in a system
- A system is secure if it starts in a secure initial state and only applies secure transitions
- A state is secure if it was obtained starting from the secure initial state and applying a sequence of secure transitions

Secure Transition System



In a secure state and transition pre condition holds

go to new secure state

Problem

- In general it is not possible to look at a given state and know that it is secure without looking at how that state was constructed.

Solutions

- Find a way of imposing the pre conditions without the consent of the hosts
  - cryptographically enforced pre conditions
- Translate pre conditions into post conditions that can be checked at subsequent hosts
  - integrity post conditions
  - history conditions

Cryptographic Enforcement

- Read access to variables can be restricted by encryption of the variables contents. Read access is only granted to those hosts with keys
- Read access can be separated from write access by use of asymmetric encryption

Integrity Post Conditions

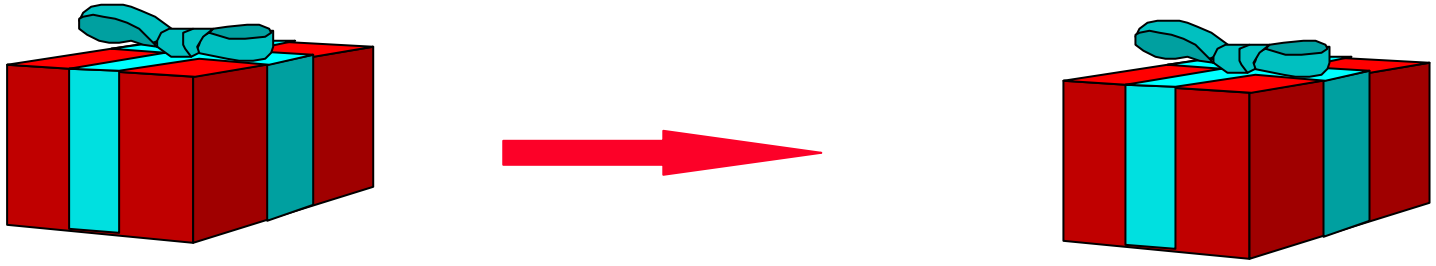
- Cryptographic signing can be used to ensure that only legitimate writers have altered a variable
  - Anonymously signed by any writer
  - can use asymmetric encryption if the data is self validating (otherwise use separate signing)

Integrity Post Conditions

- Cryptographic signing can be used to ensure that only legitimate writers have altered a variable
  - Anonymously signed by any writer
  - can use asymmetric encryption if the data is self validating (otherwise use separate signing)

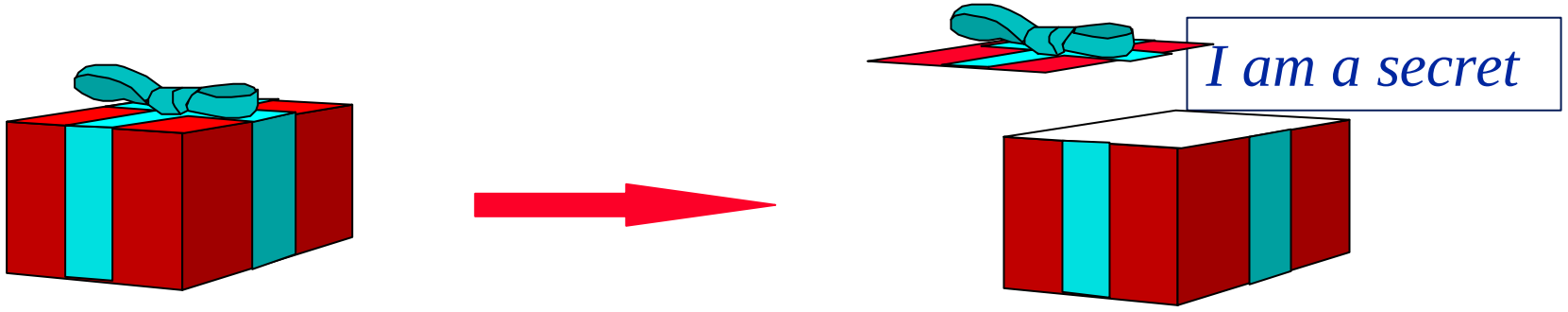
Integrity Post Conditions

- Cryptographic signing can be used to ensure that only legitimate writers have altered a variable
  - Anonymously signed by any writer
  - can use asymmetric encryption if the data is self validating (otherwise use separate signing)



Integrity Post Conditions

- Cryptographic signing can be used to ensure that only legitimate writers have altered a variable
  - Anonymously signed by any writer
  - can use asymmetric encryption if the data is self validating (otherwise use separate signing)

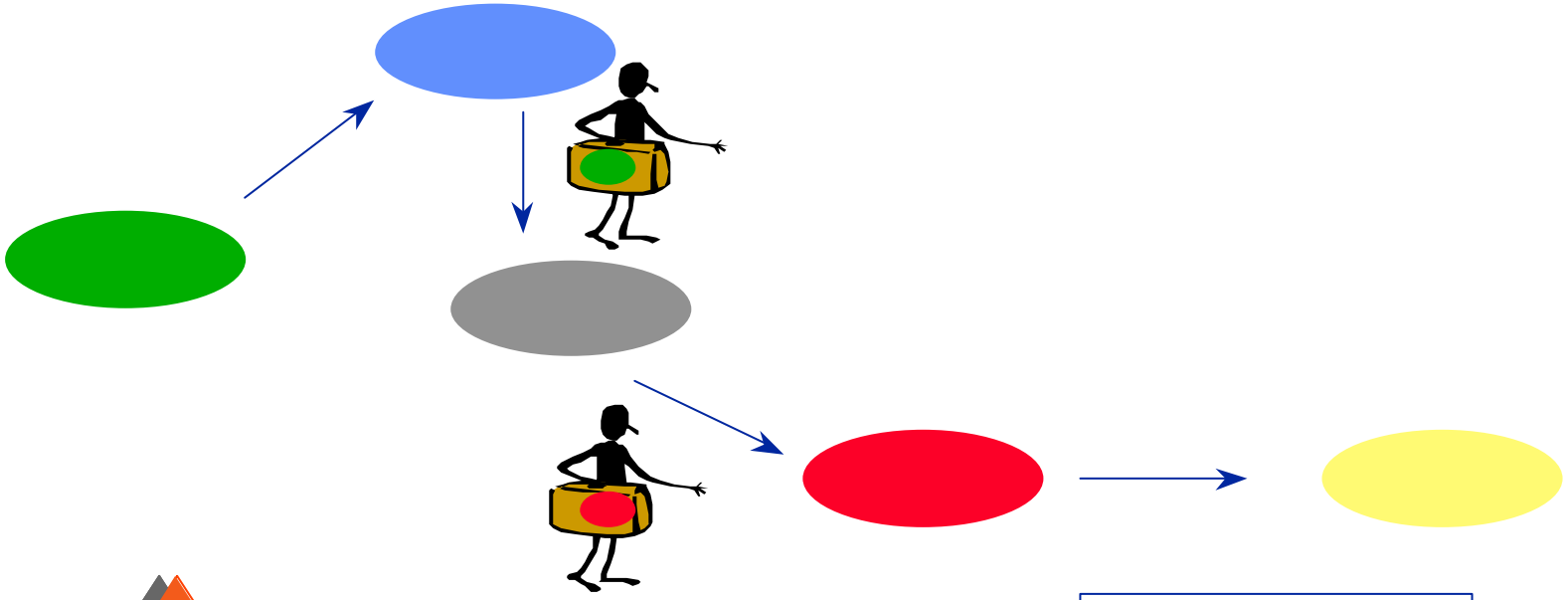


Promises

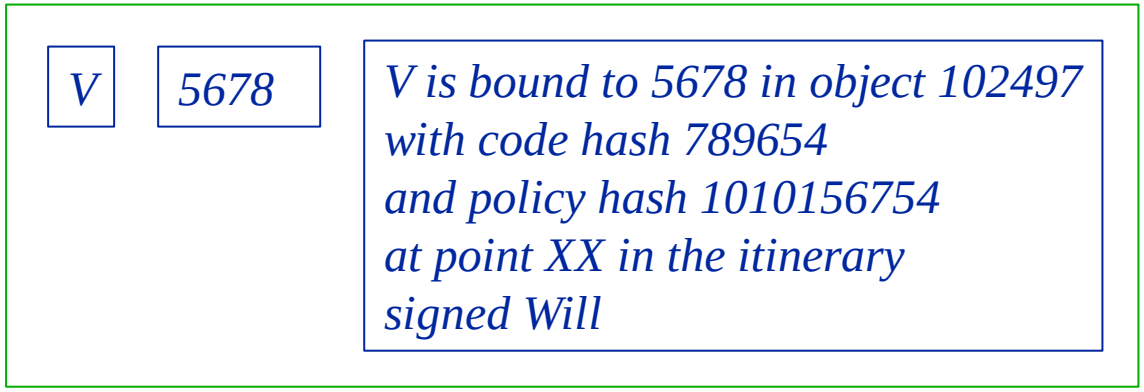
- Hosts commit themselves to the value of a variable or group of variables by signing.
- Commitments may be conditional on other hosts commitments remaining valid.

Replays

- *V* | *1234* | *V is bound to 1234 signed Will* — *yesterday*
- *V* | *5678* | *V is bound to 5678 signed Will* — *today*



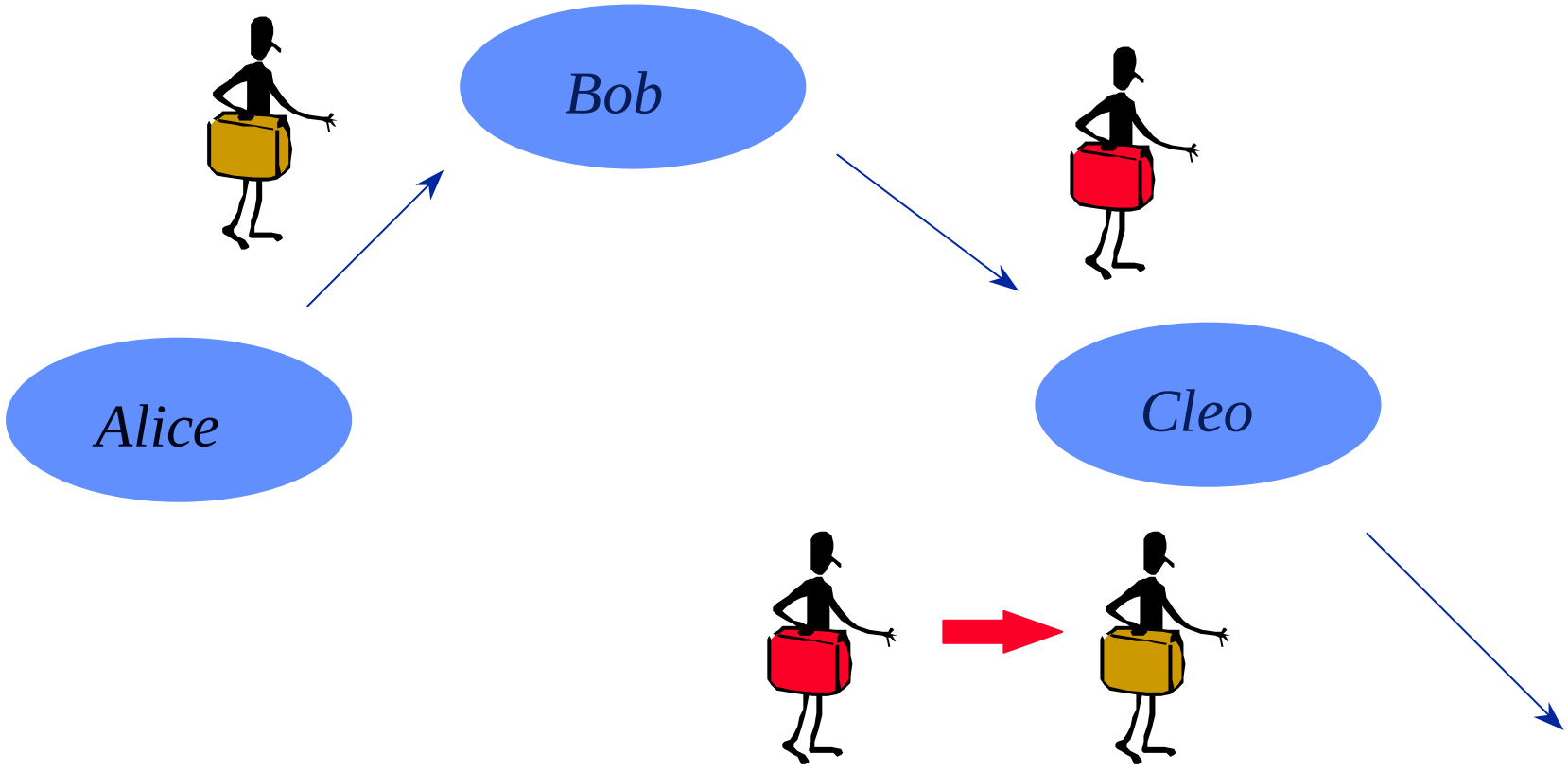
Signing

- 

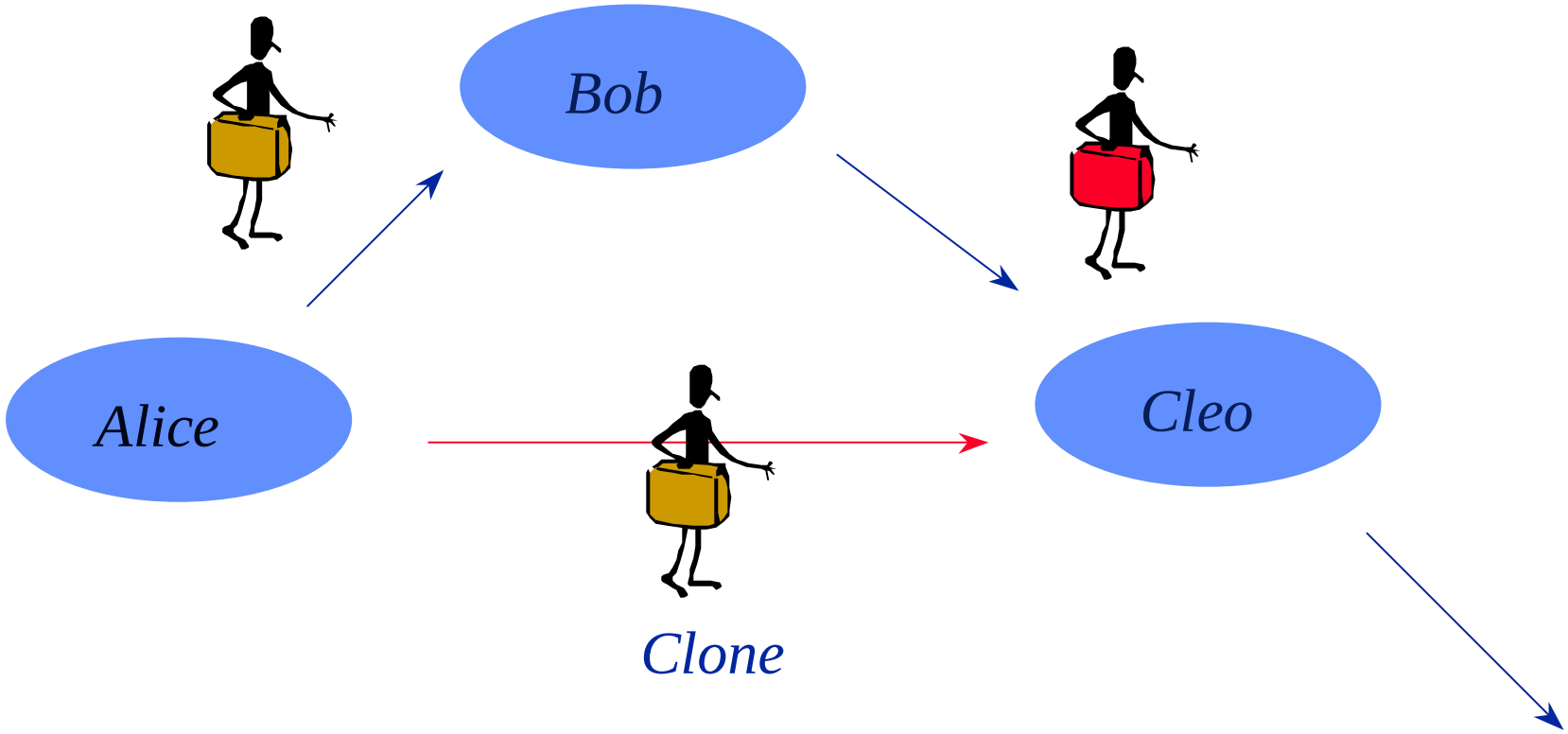
today

i.e. object is created with unique object number and signing is based on the static part of the object and the point that the signing happens in the objects history.

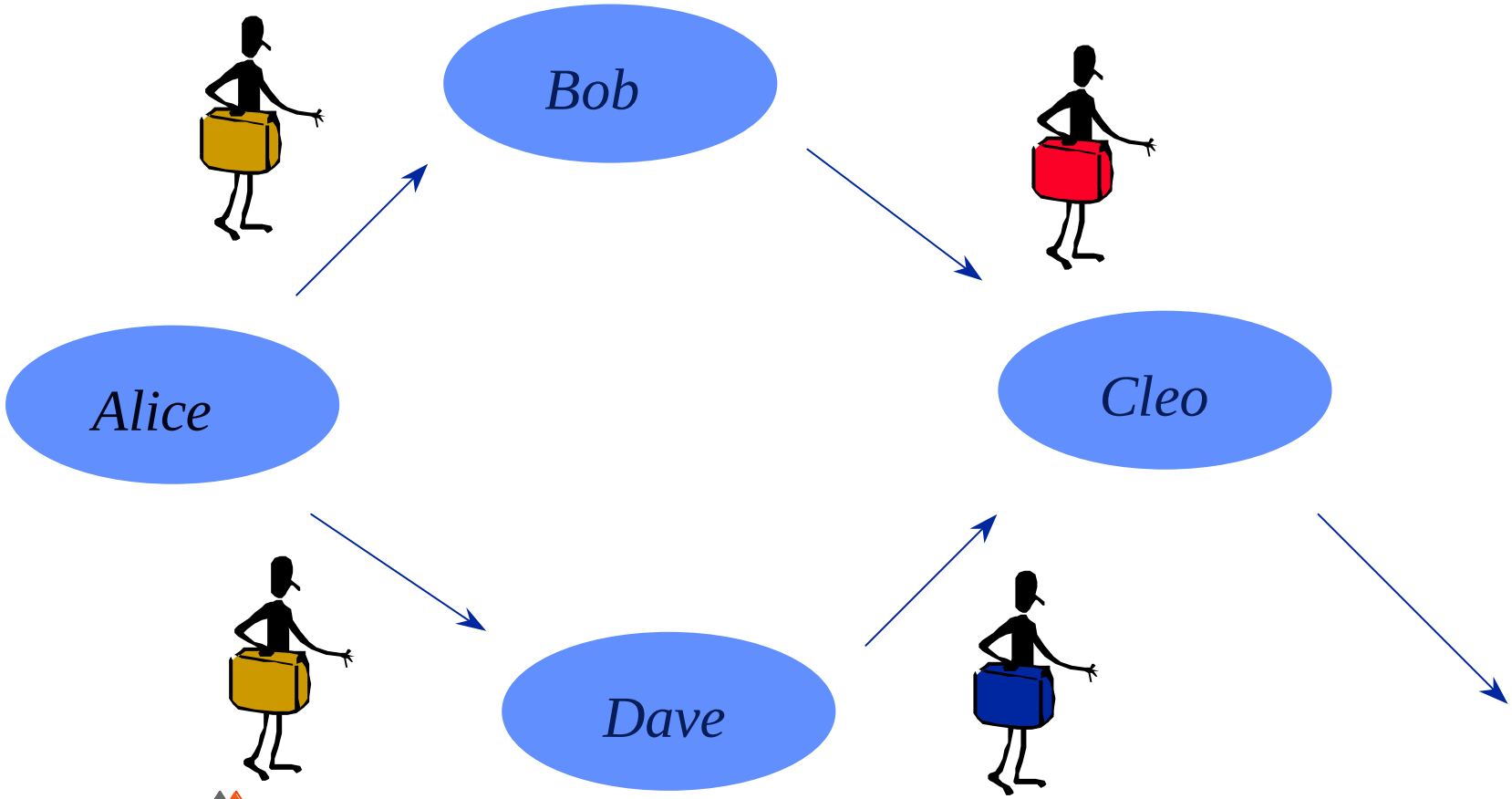
Unwinding



Piggy-in-the-Middle

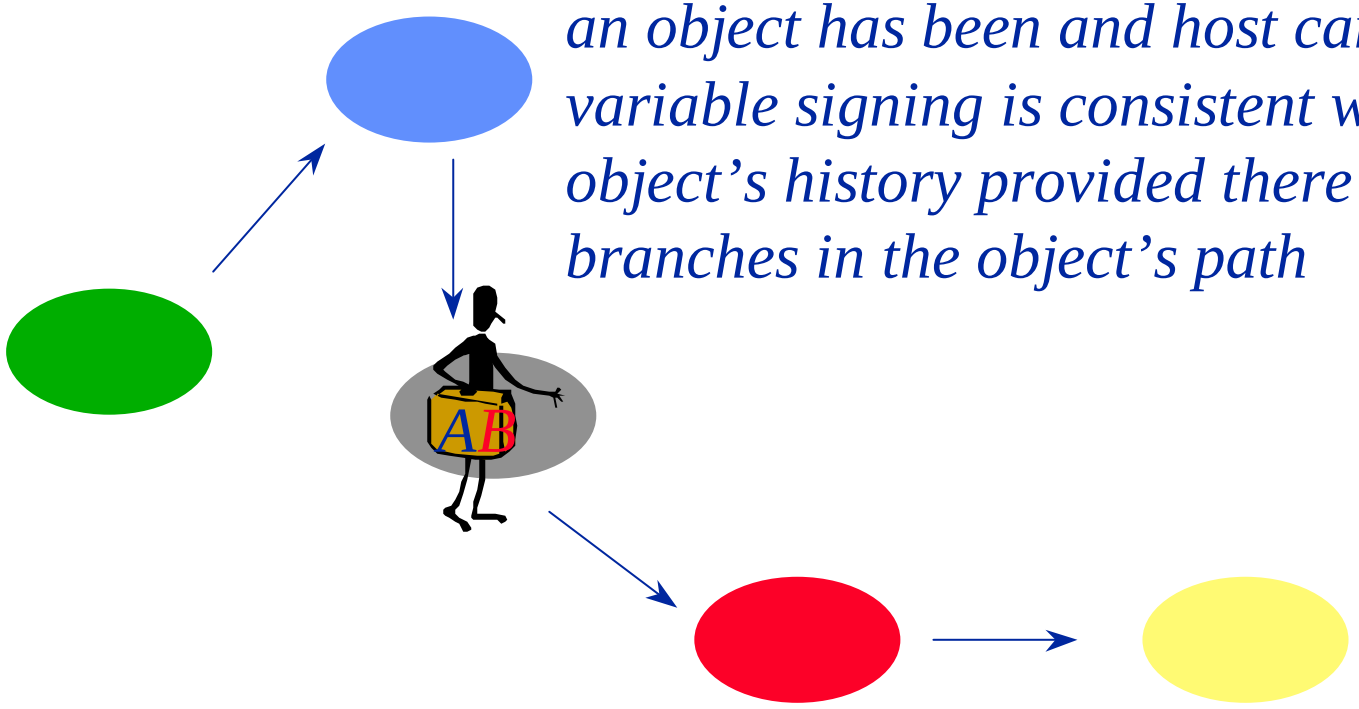


Best Branch

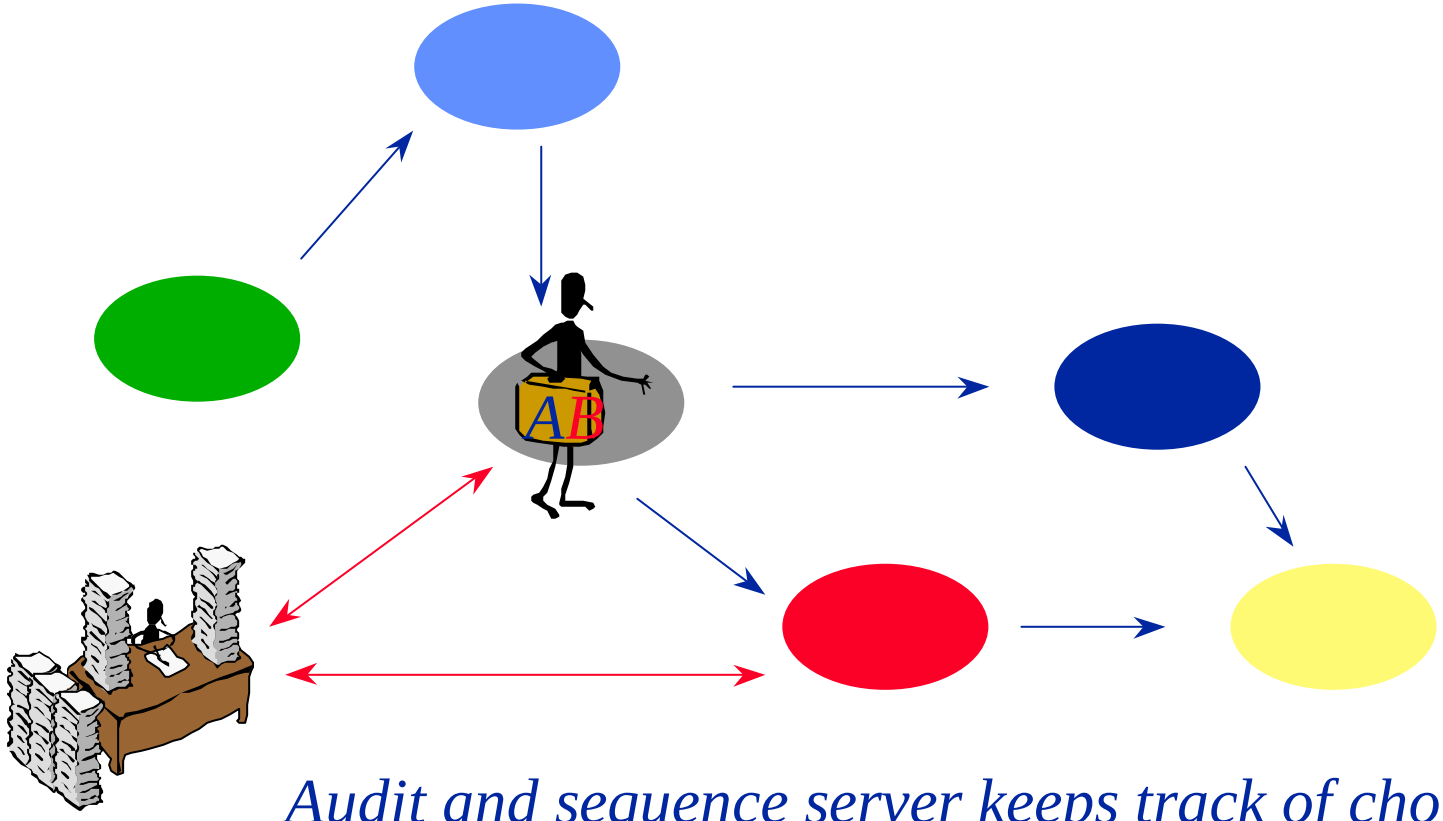


Histories

Nested signing can be used to record where an object has been and host can check that variable signing is consistent with the object's history provided there are no branches in the object's path



Audit and Sequence Server

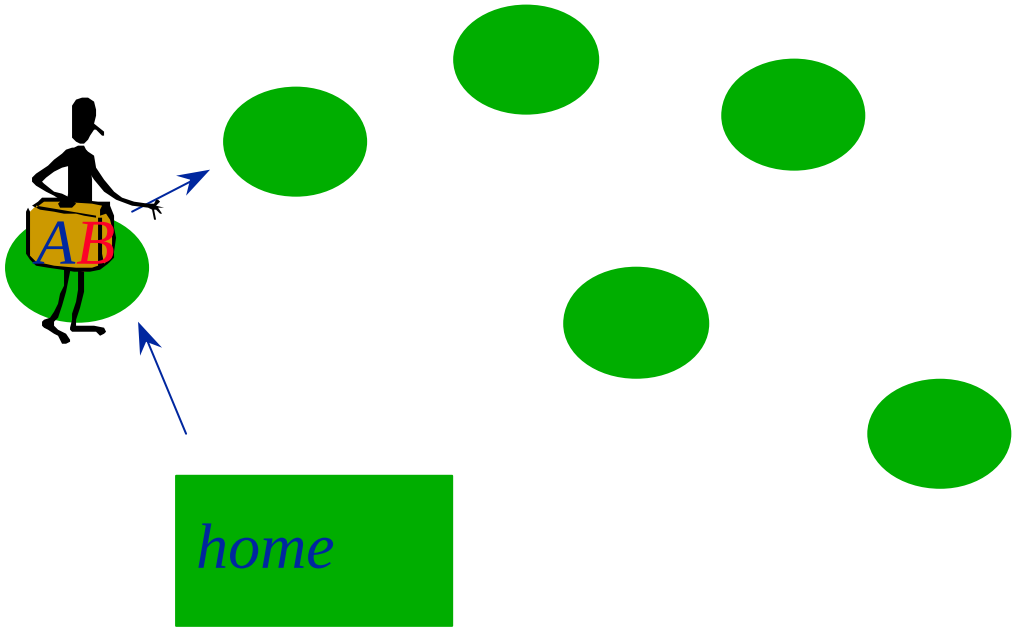


Audit and sequence server keeps track of choices

A Little Language of Behaviour

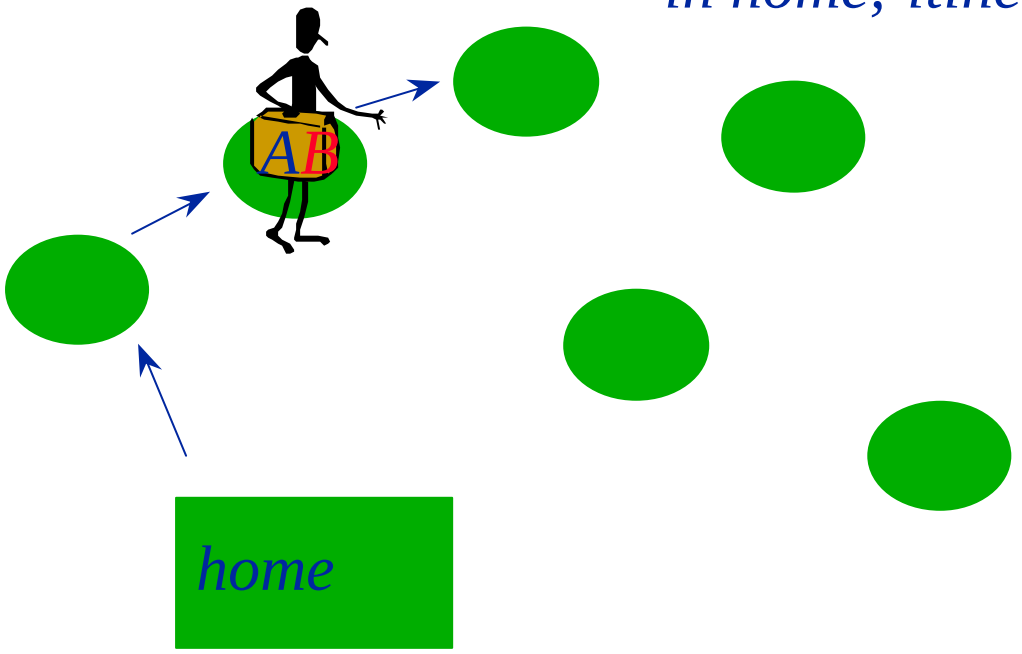
- Object movement is described by a finite state machine
  - itineraries: h;k, h+k, h[action], skip
- Each “state” corresponds to a visit to a host where the object performs actions
  - updates: !!x, !x, ^x, <update,…,update>
  - checks: h?x, h?<x,y,z>
  - conditional updates: H?x,k?y → <!!x,!y>

Information Gatherer



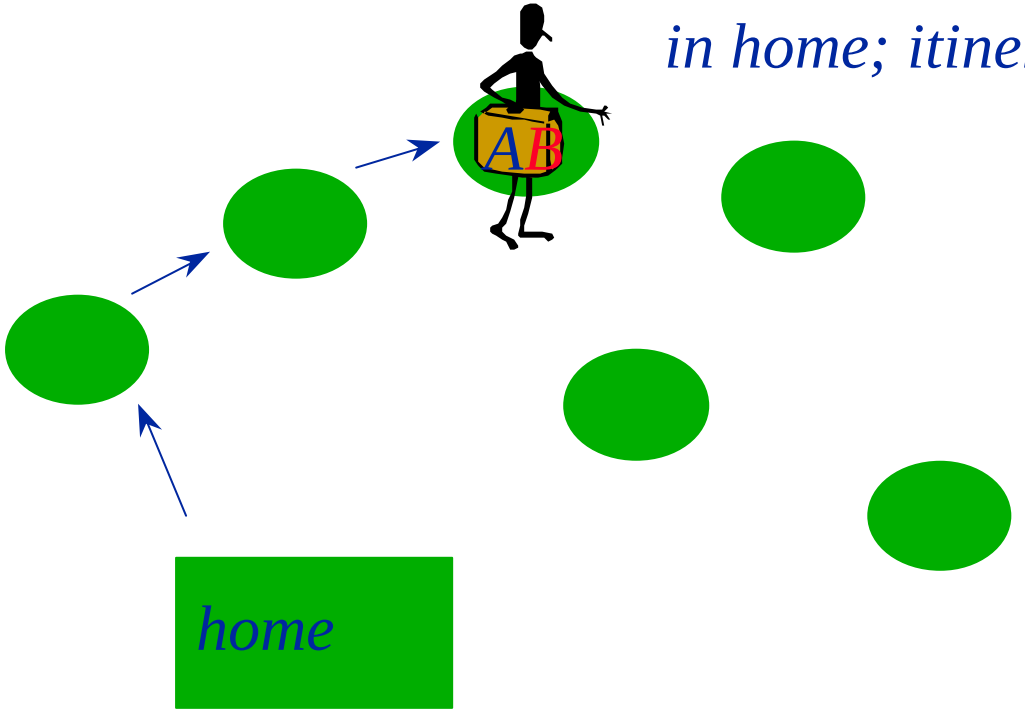
Information Gatherer

$$let\ itinerary = (P_1 + \ldots + P_n)[\hat{}stack];\ itinerary + Skip$$
$$in\ home;\ itinerary;\ home$$



Information Gatherer

let itinerary = (P_1+ … + P_n)[^stack]; itinerary
+ Skip
in home; itinerary; home

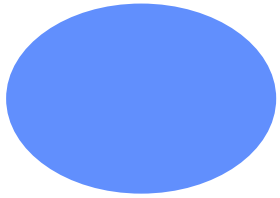


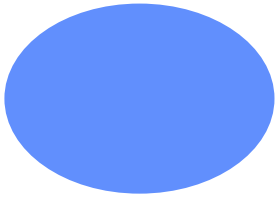
	Home	***P_i***
Stack	*R/W*	*W*

Lottery

Lottery Registrary

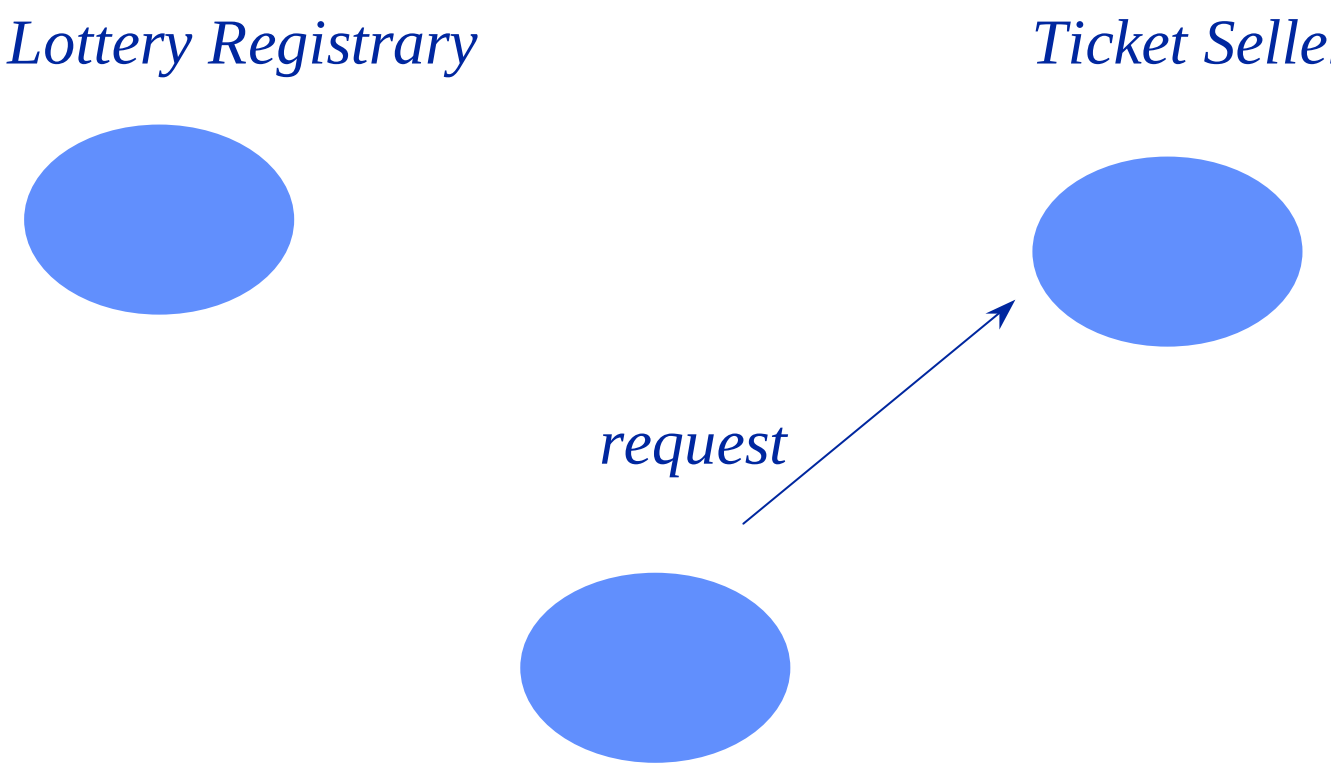
Ticket Seller



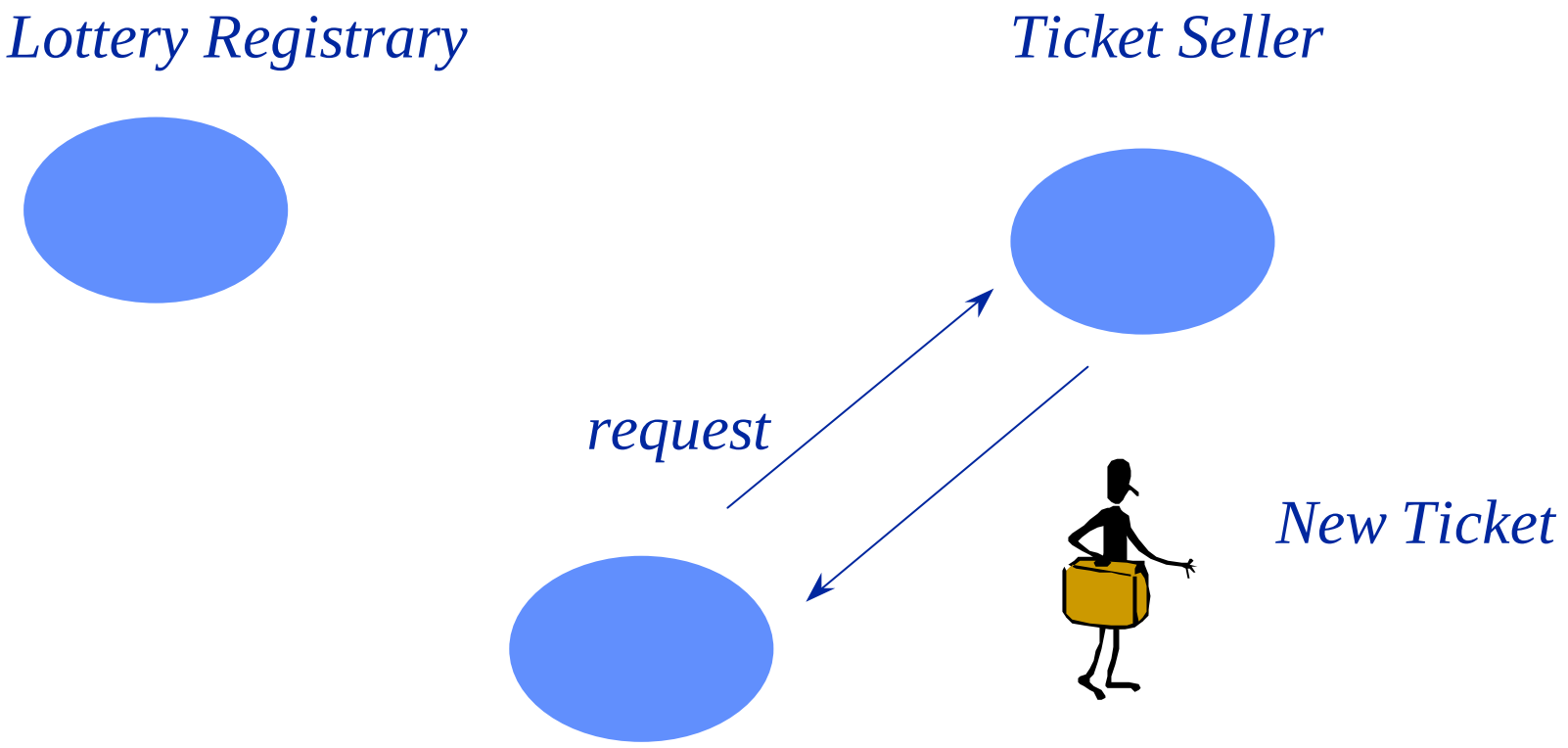


Client

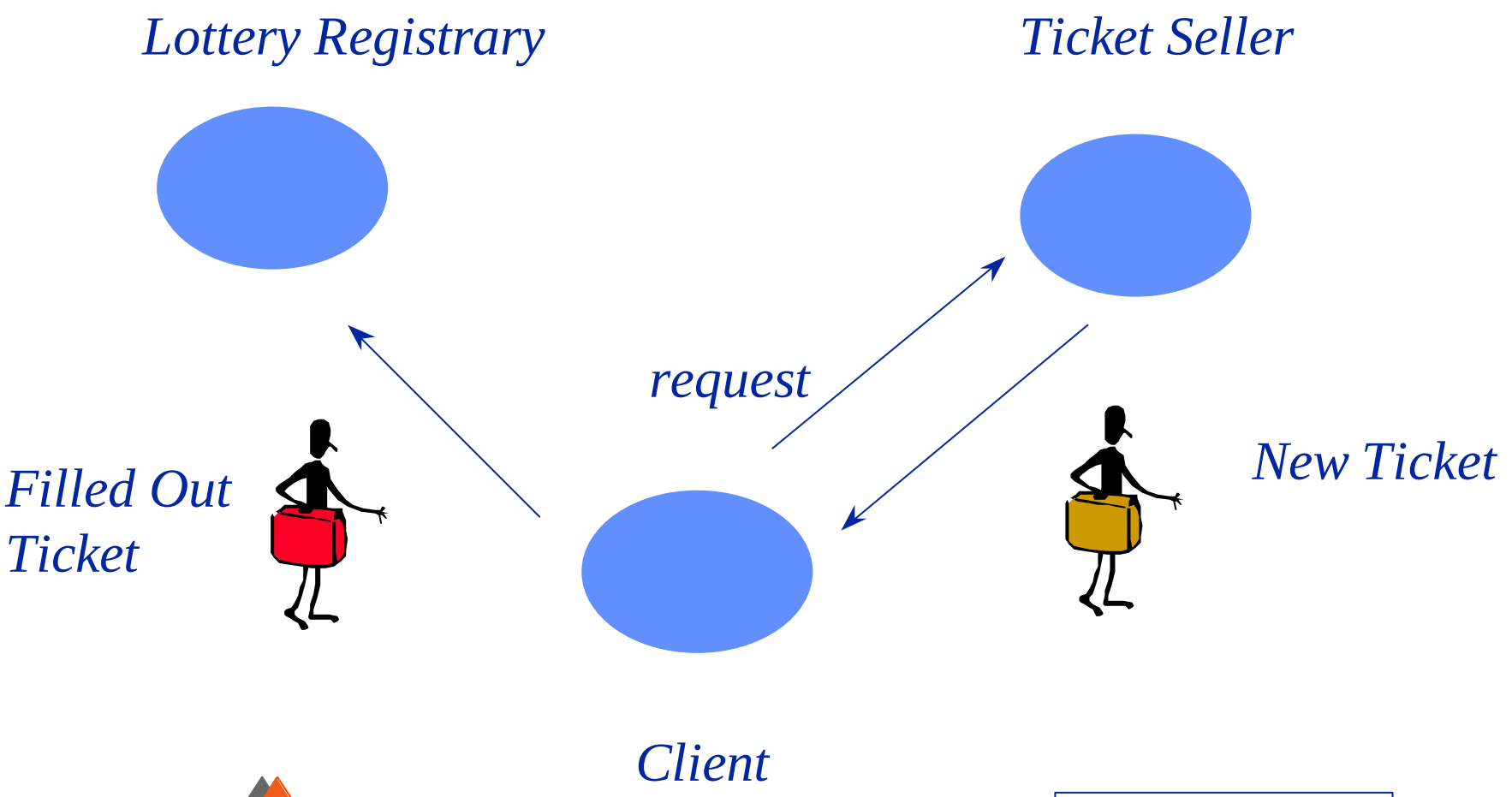
Lottery



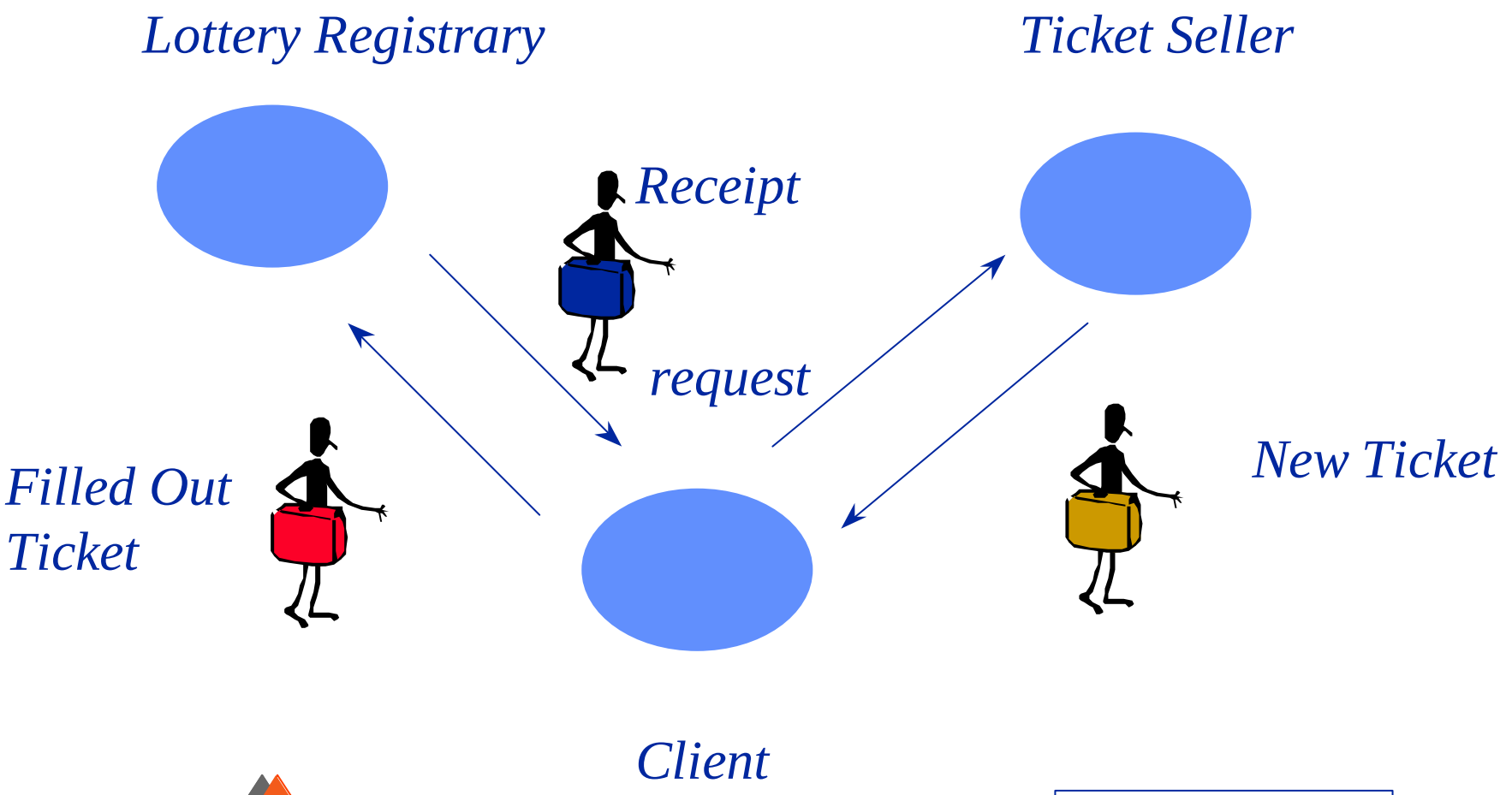
Lottery



Lottery

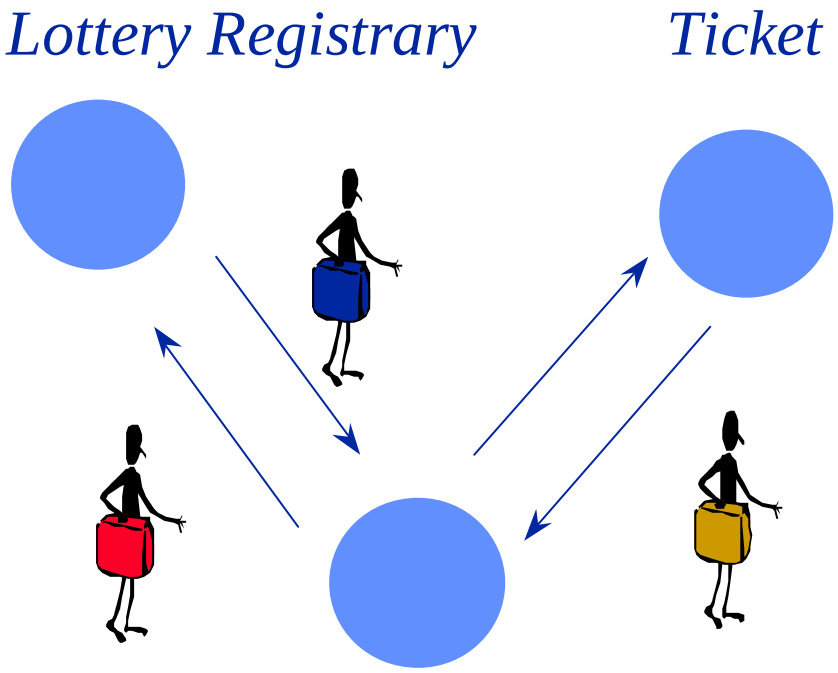


Lottery



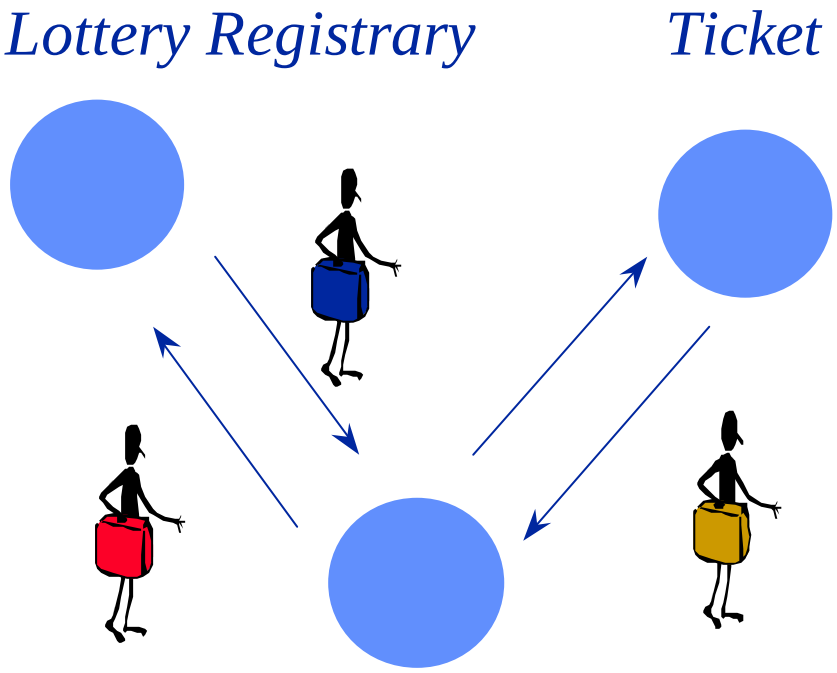
Lottery Behaviour

TS ; Client ; TR[<!TN,!Bet>] ; Client[TR?<TN,Bet>]



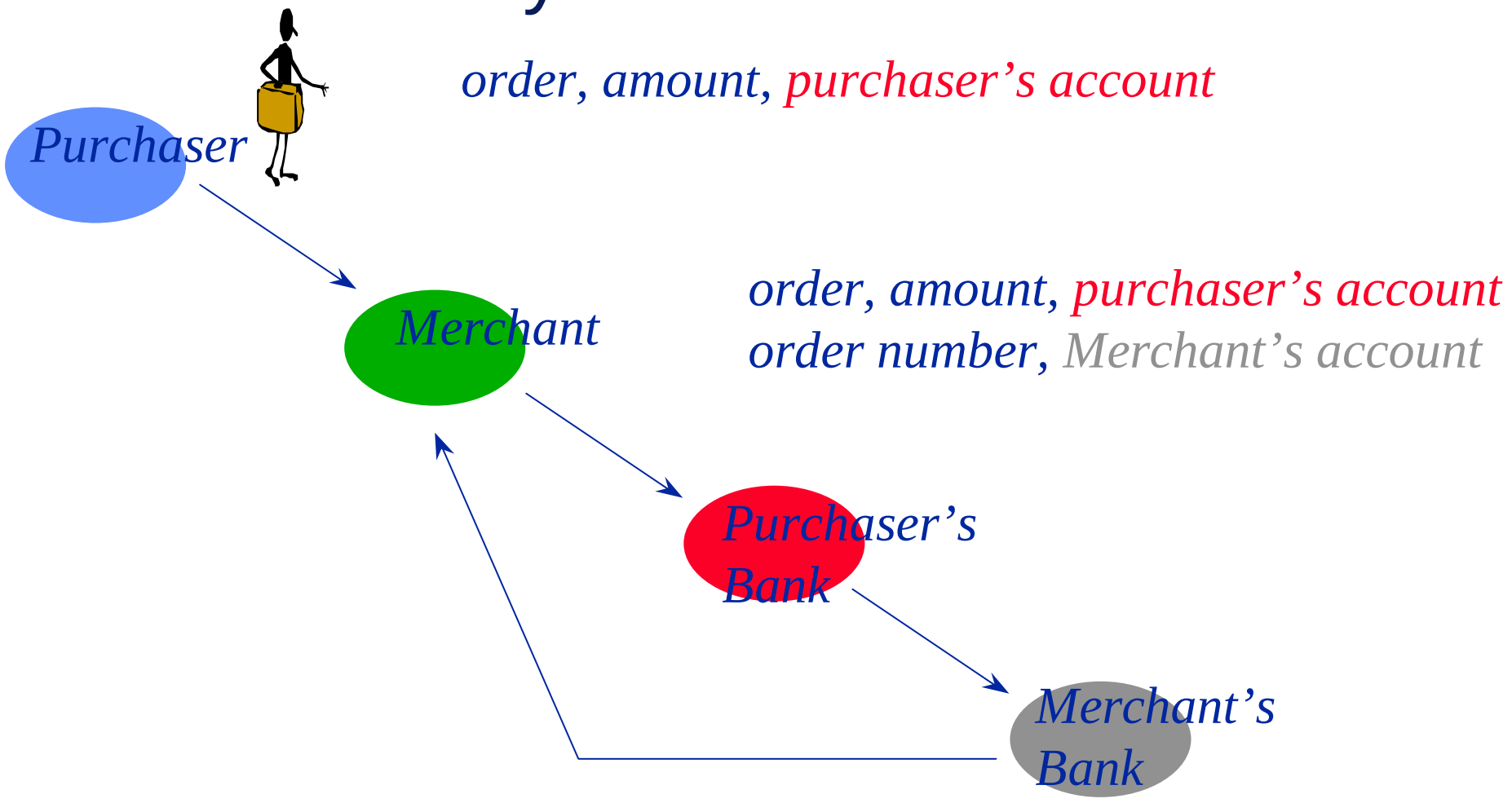
Lottery Behaviour

TS ; Client ; TR[<!TN,!Bet>] ; Client[TR?<TN,Bet>]



	TN	Bet
TS	W	
Client	R	W
TR	R	R

Payment Protocol



Payment Access Matrix

	order	*amount*	*order number*	*Purchaser's account*	*Merchant's account*
Purchaser	*W*	*W*		*W*	
Merchant	*R*	*R*	*R/W*		*W*
Purchaser's Bank		*R*	*R*	*R*	
Merchant's Bank		*R*	*R*		*R*

Payment Behaviour

P[<<!!order,!!amount,!!PurchaserAccount>];

M[P?<order,amount,PurchaserAccount> →
<!order,!amount,!!OrderNumber,!!MerchantAccount>>];

PB[M?<order,account,OrderNumber,MerchantAccount>,
P?<order,amount,PurchaserAccount> →
<!amount,!OrderNumber>];

MB[M?<amount,OrderNumber,MerchantAccount>,
PB?<amount,OrderNumber> →
<!amount,!OrderNumber>];

M[MB?<amount,OrderNumber>]