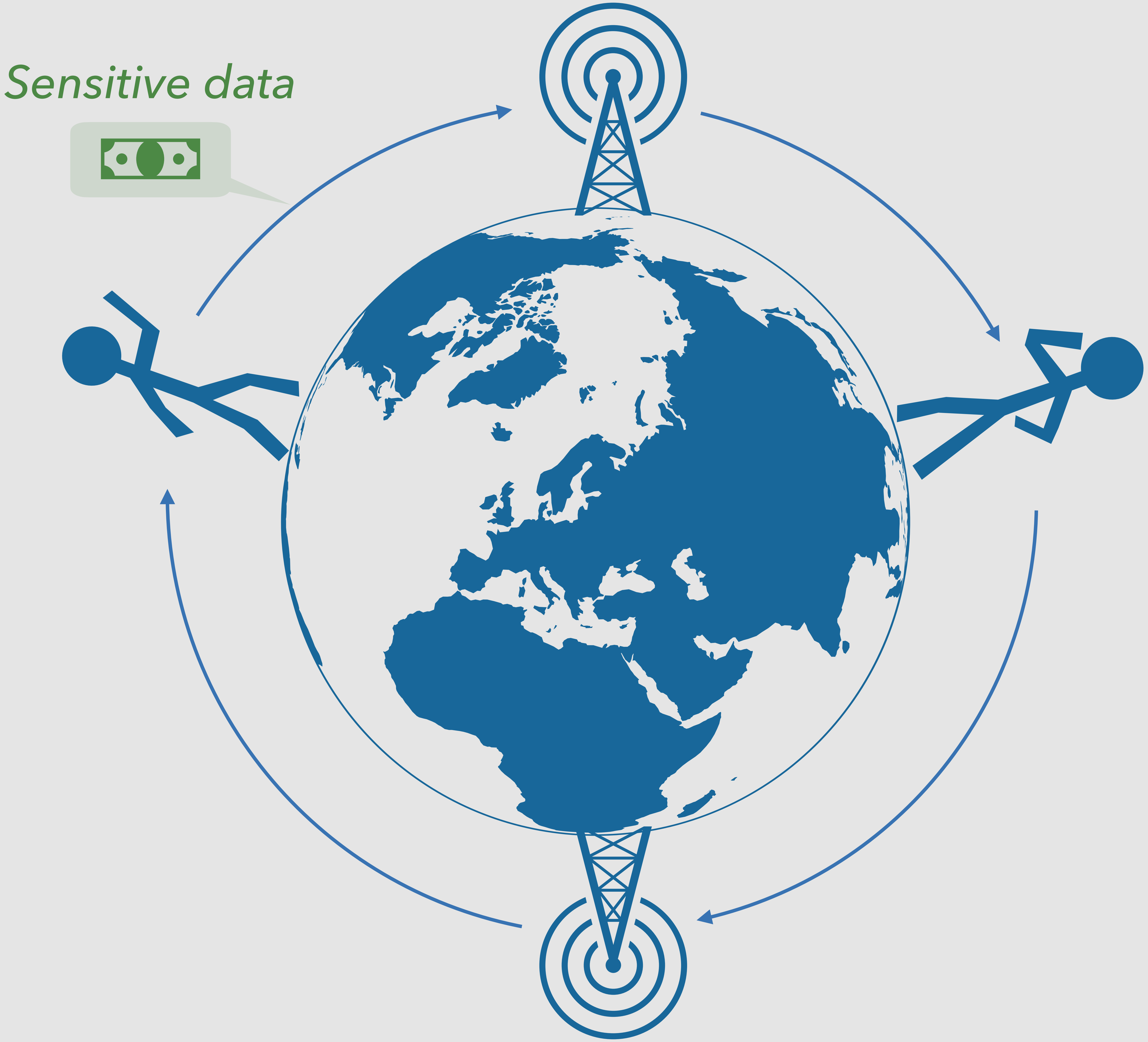
Security Going Live: Verification of Real-Time Components of Security Protocols

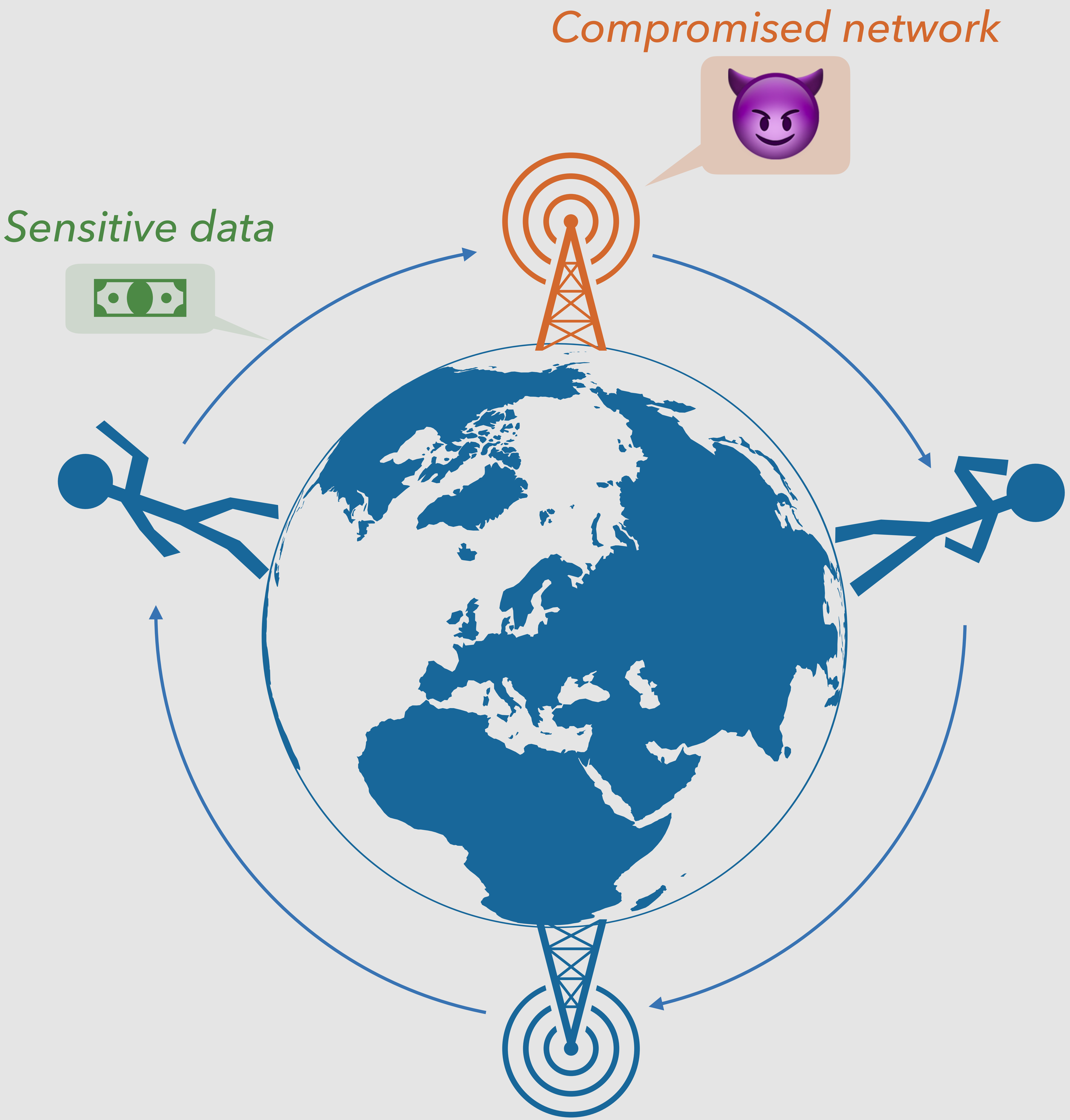
Work in progress with

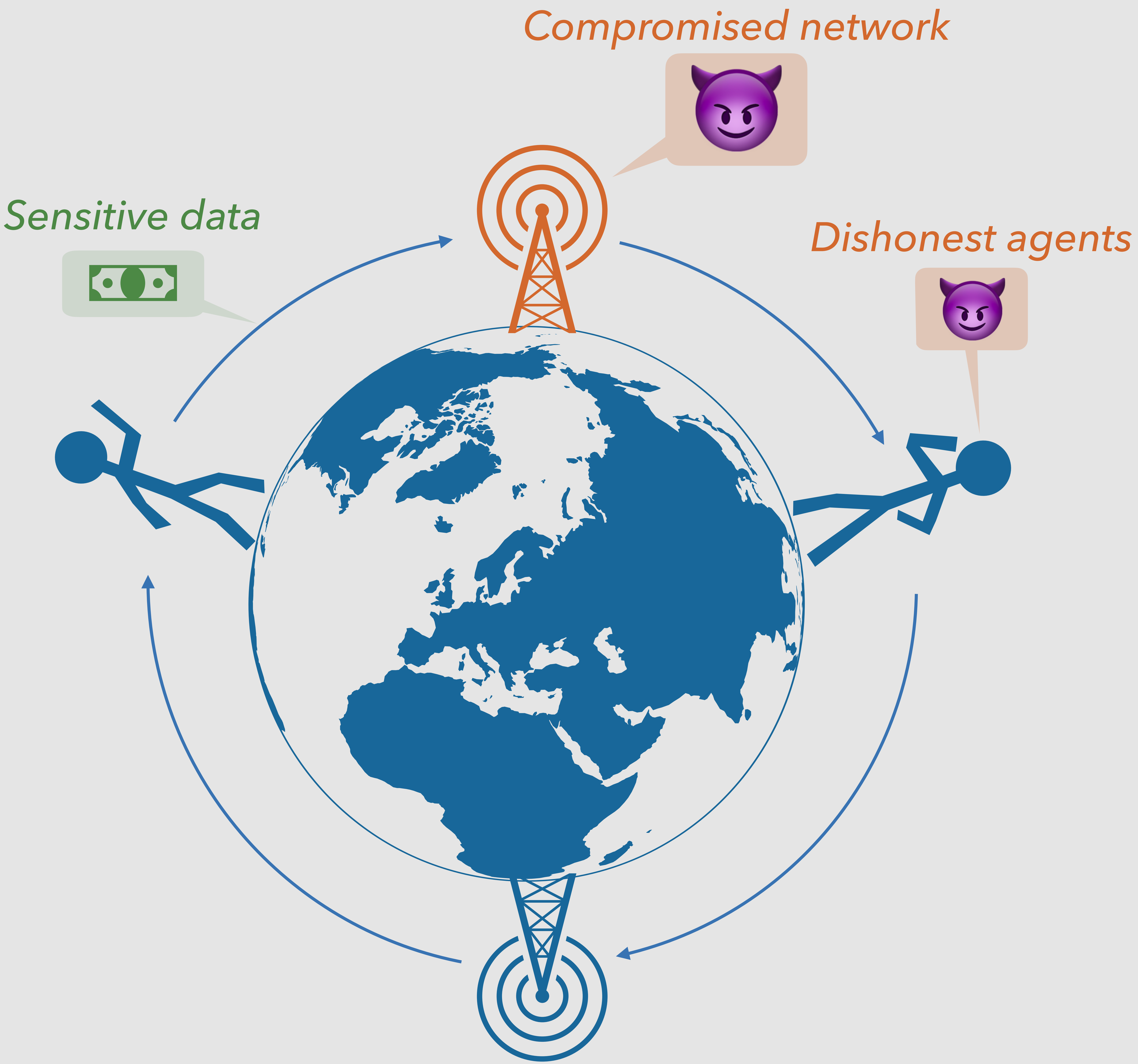
Pedro Moreno-Sanchez[1], Itsaka Rakotonirina[2], Clara Schneidewind[2]

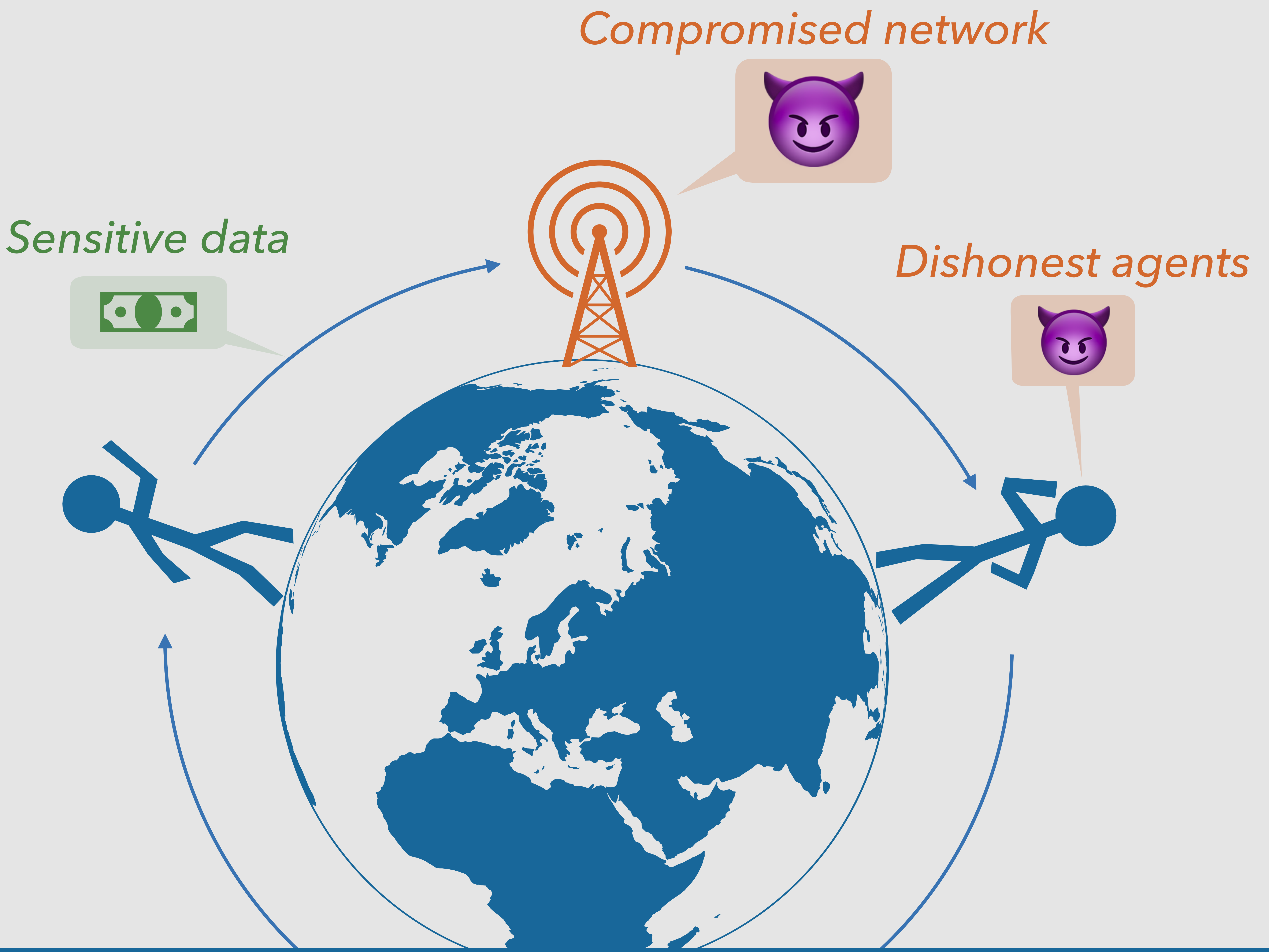
[1] IMDEA Software Institute

[2] MPI-SP


Sensitive data


Compromised network
Sensitive data


Compromised network
Sensitive data
Dishonest agents



Fairness: can one stall the protocol after having gained an "advantage"?

Theorem

Asynchronous communications are unfair.

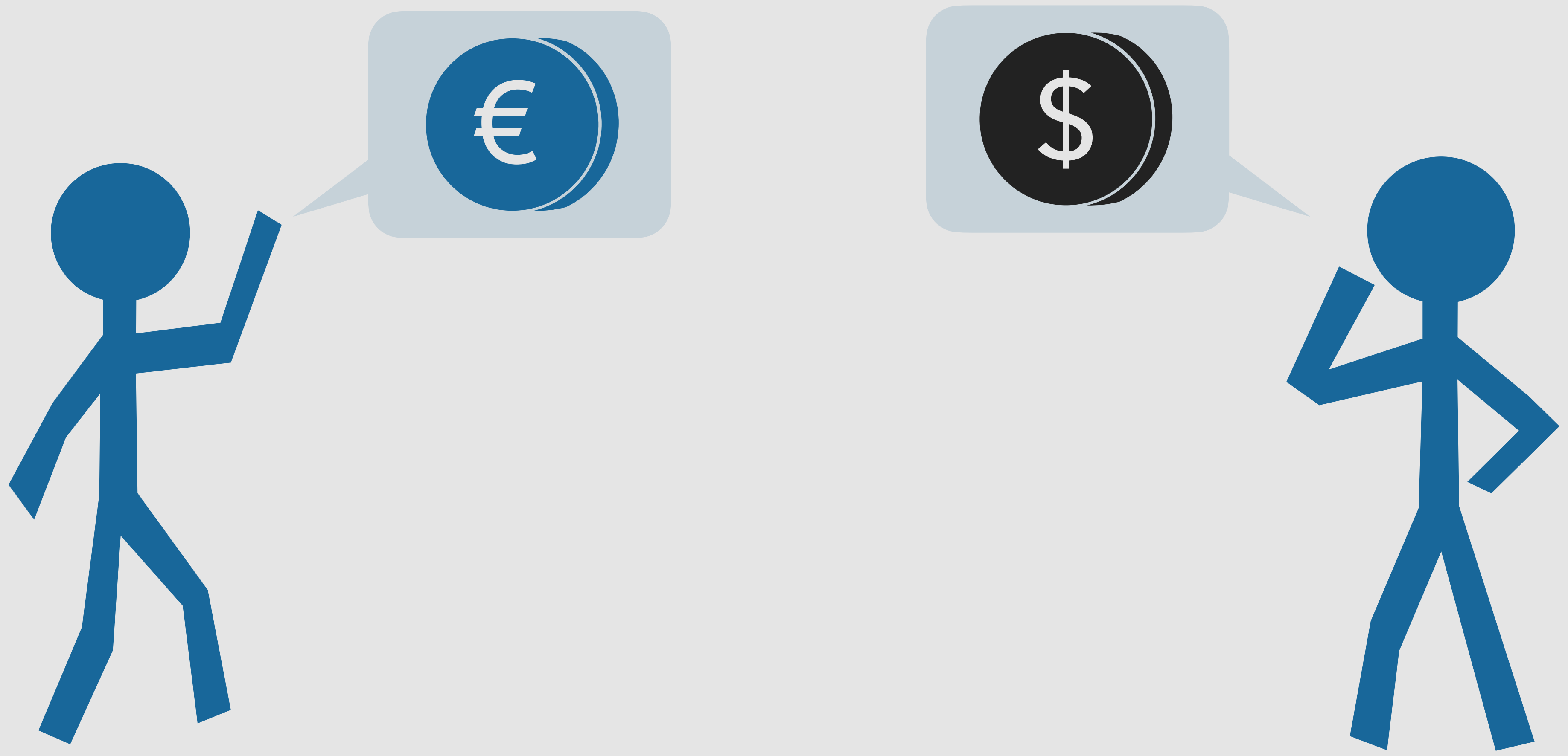
- S. Even, Y. Yacobi, 1980. *Relations among public key signature system*
- R. Cleve, 1986. *Limits on the security of coin flips when half the processors are faulty*

Theorem

Asynchronous communications are unfair.

- S. Even, Y. Yacobi, 1980. *Relations among public key signature system*
- R. Cleve, 1986. *Limits on the security of coin flips when half the processors are faulty*

Example: *asynchronous coin swapping*

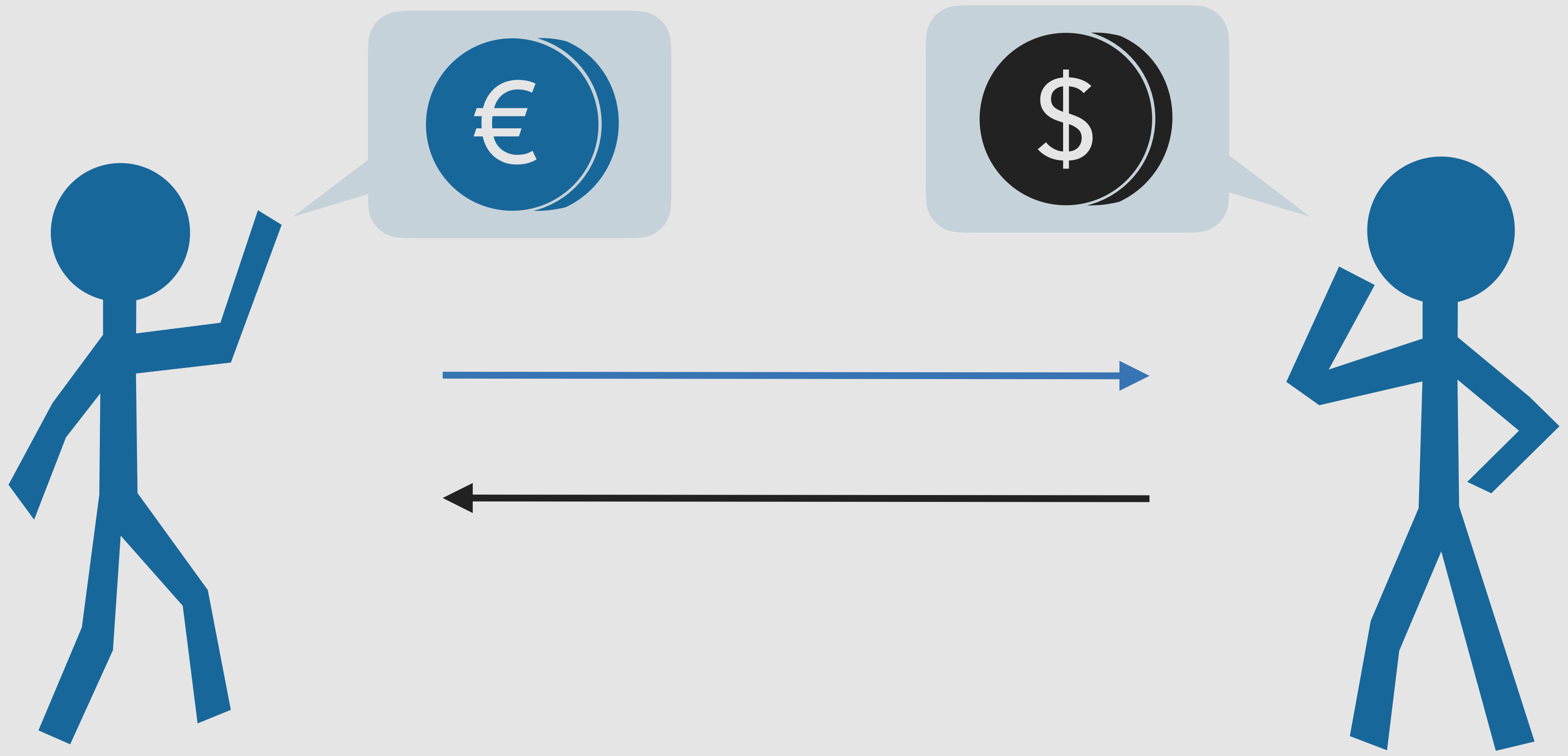


Theorem

Asynchronous communications are unfair.

- S. Even, Y. Yacobi, 1980. *Relations among public key signature system*
- R. Cleve, 1986. *Limits on the security of coin flips when half the processors are faulty*

Example: *asynchronous coin swapping*

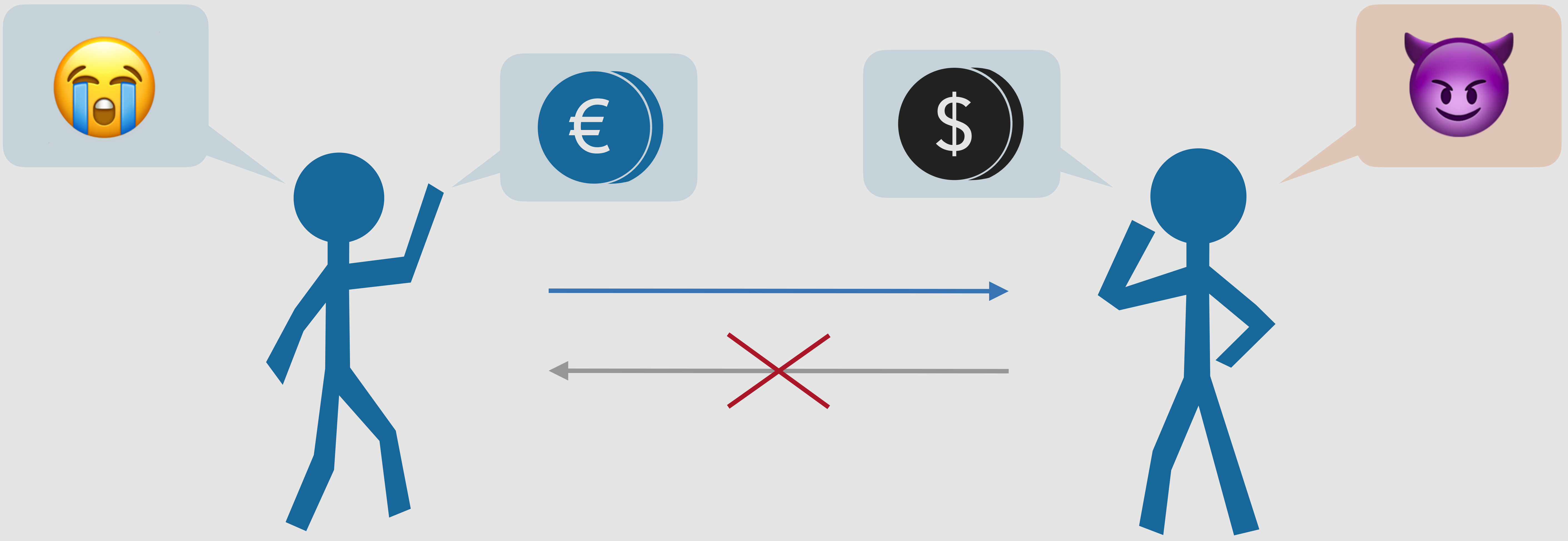


Theorem

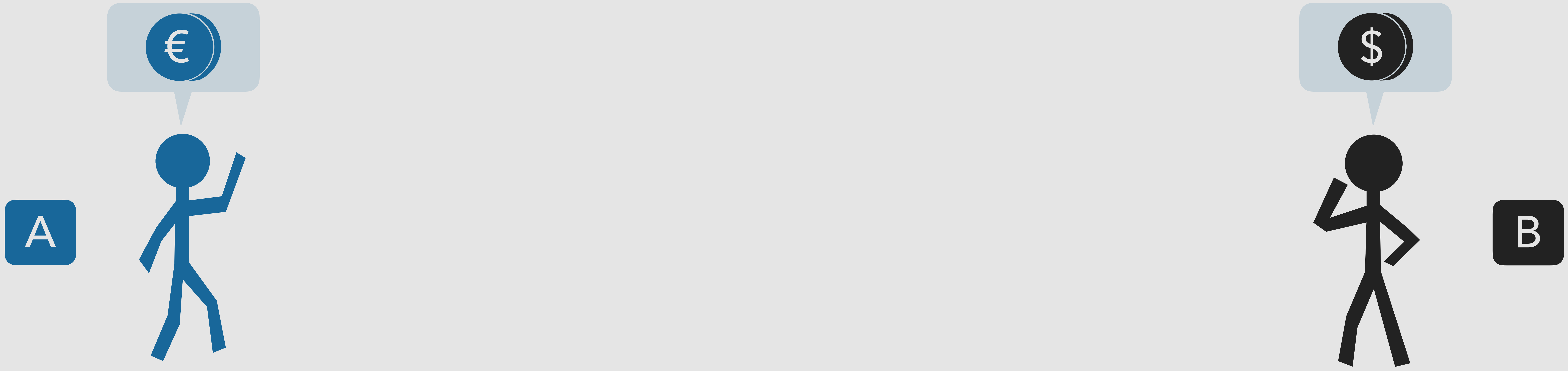
Asynchronous communications are unfair.

- S. Even, Y. Yacobi, 1980. *Relations among public key signature system*
- R. Cleve, 1986. *Limits on the security of coin flips when half the processors are faulty*

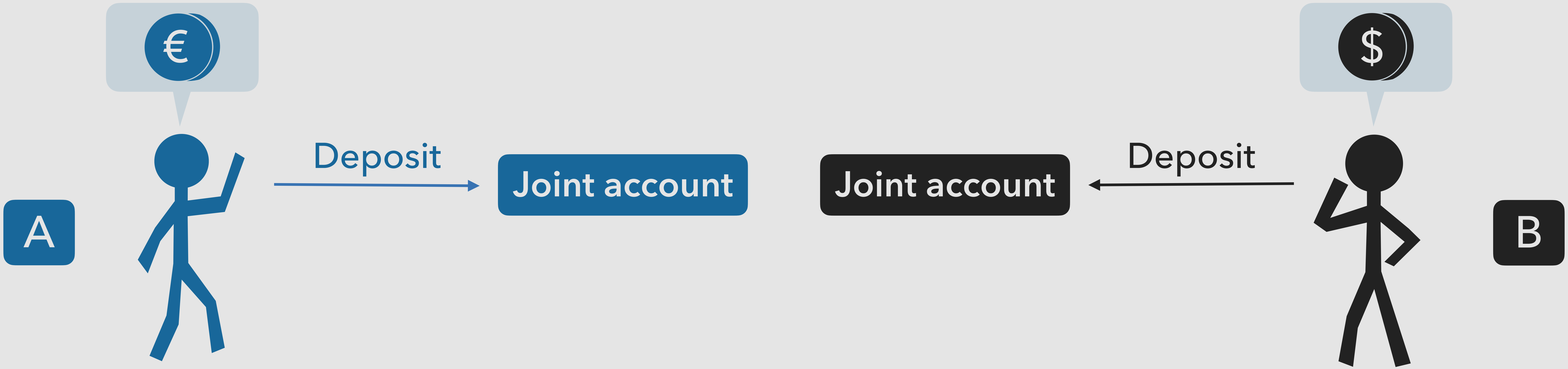
Example: *asynchronous coin swapping*



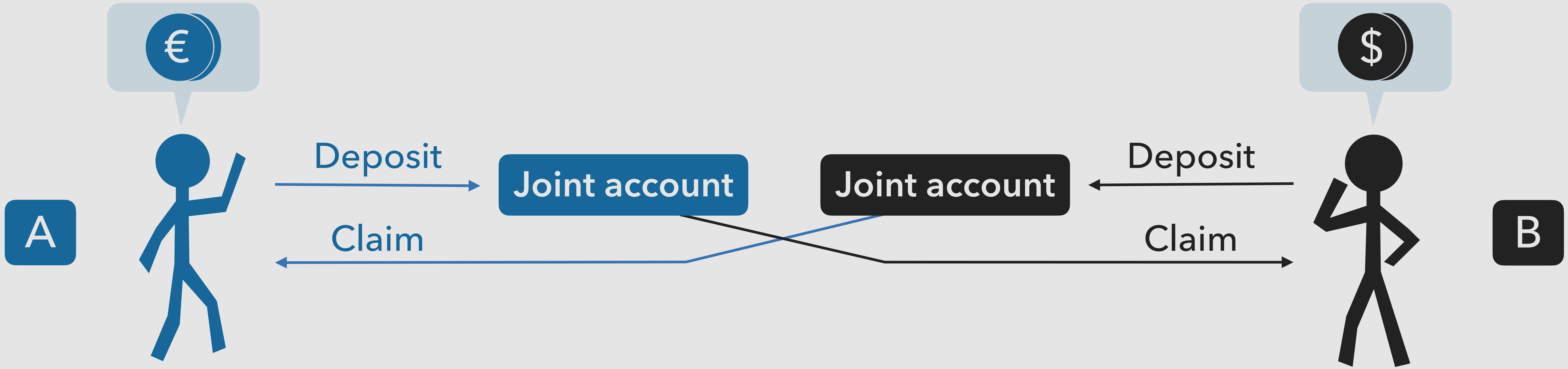
Practical solutions



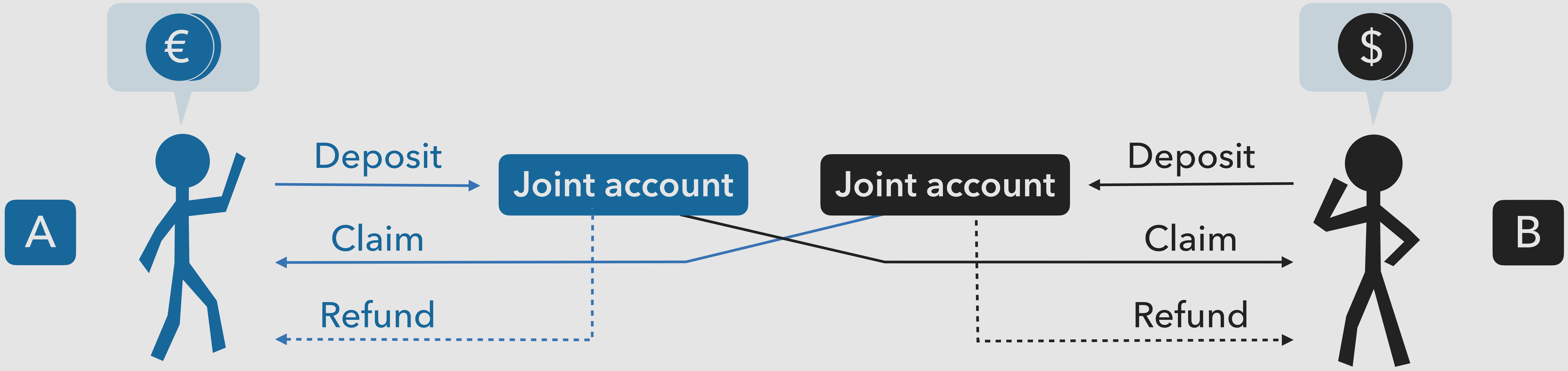
Practical solutions



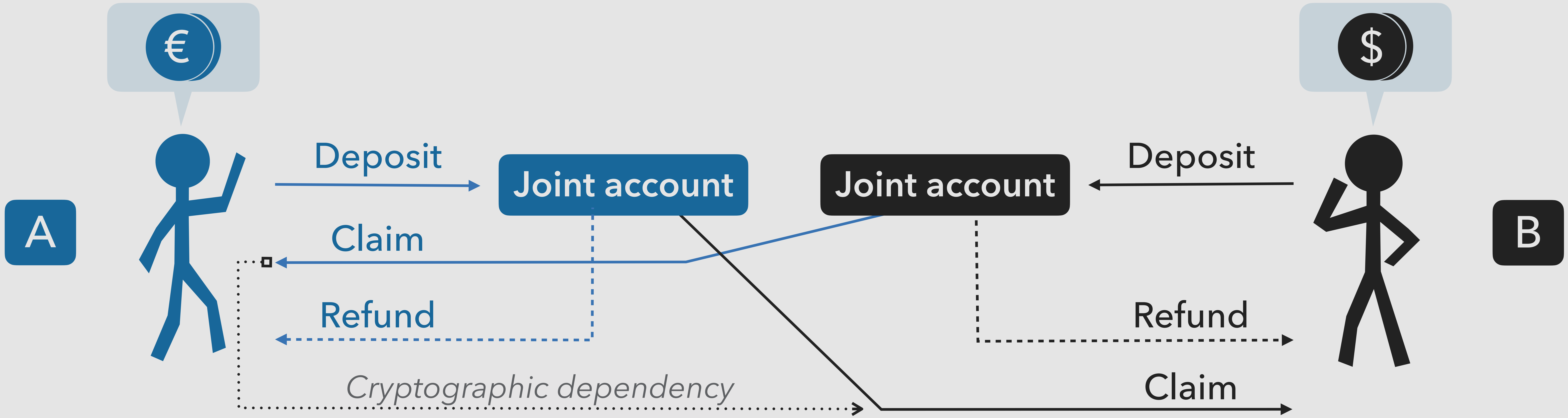
Practical solutions



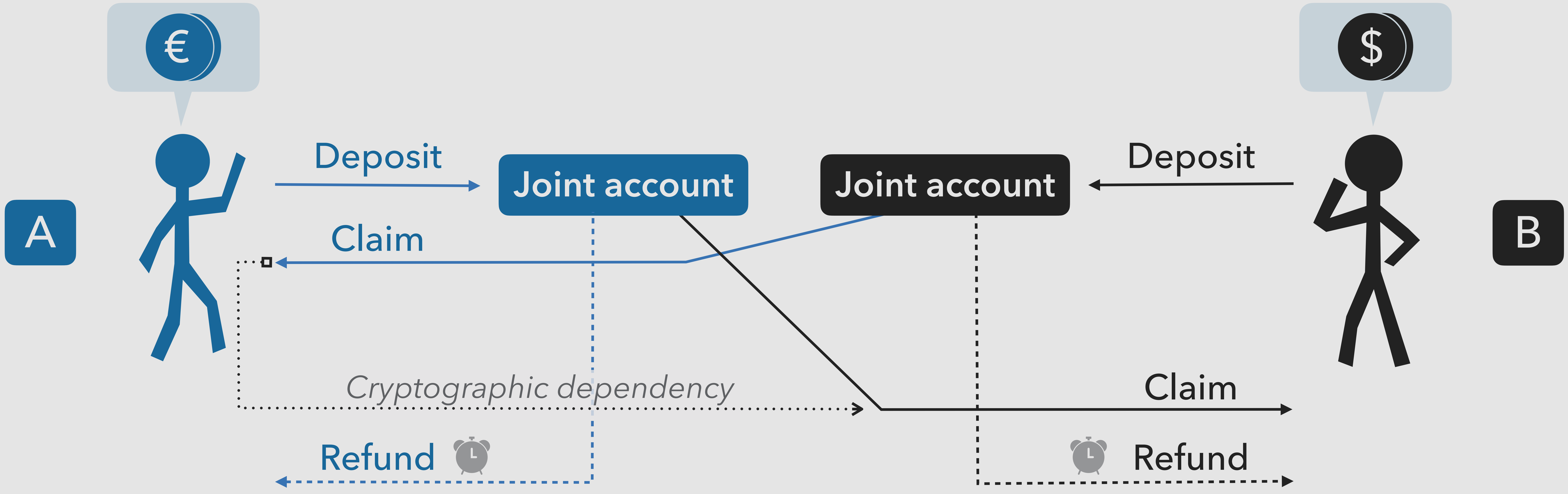
Practical solutions



Practical solutions

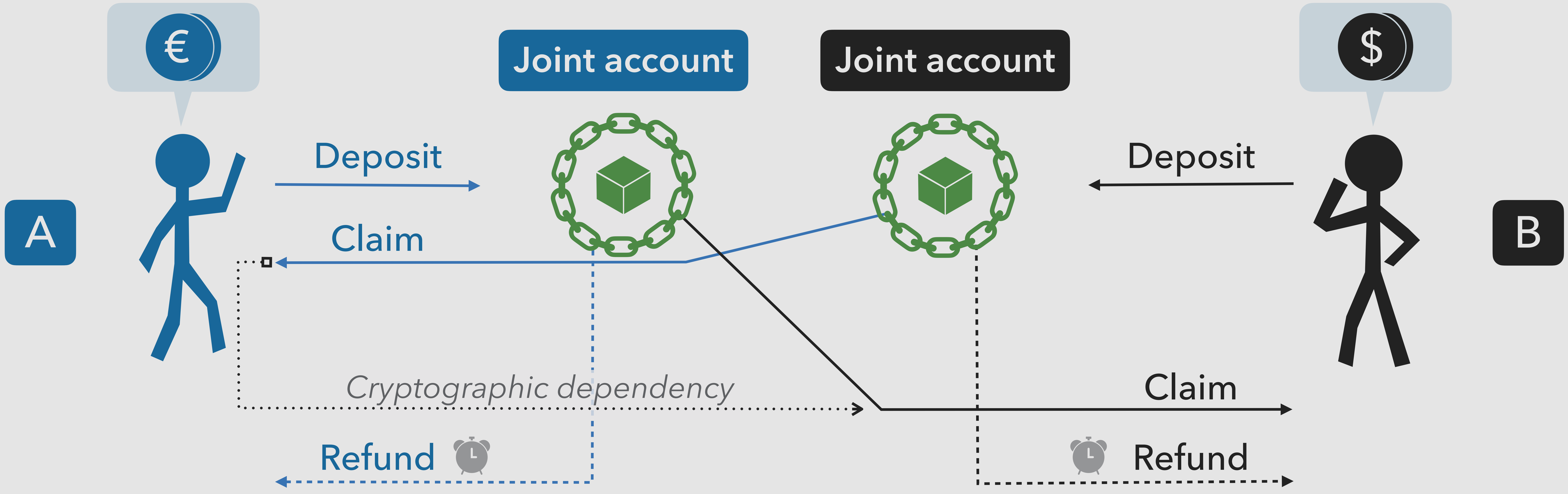


Practical solutions



Practical solutions

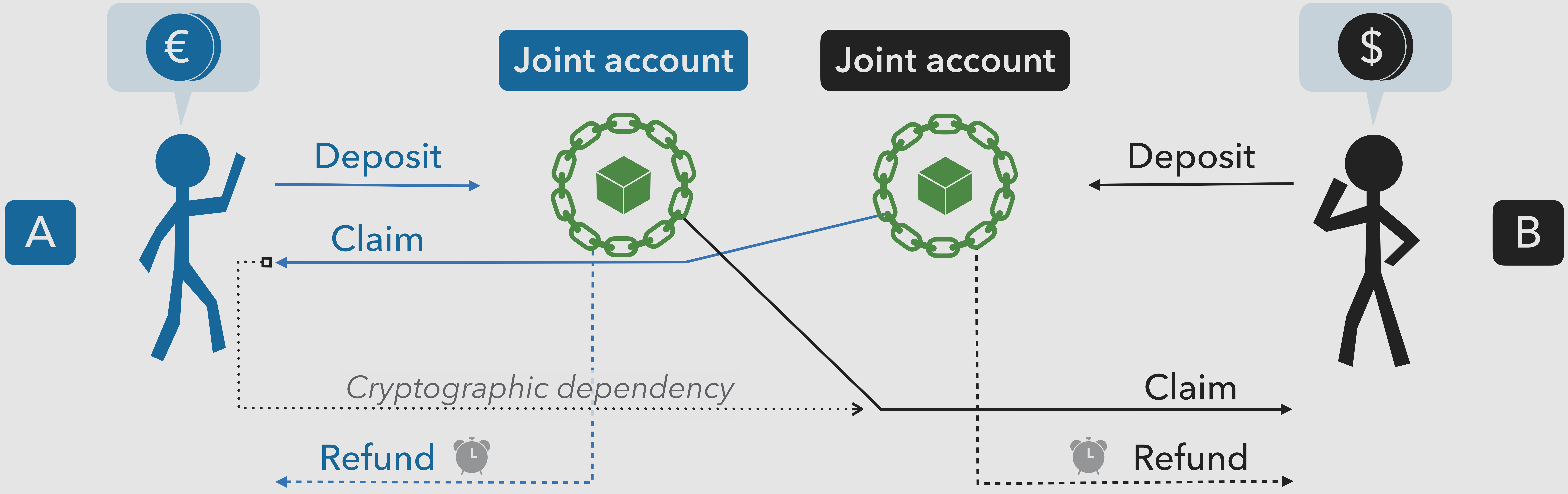
Blockchain(s): trusted public ledger(s) publishing transactions regularly



Practical solutions

Blockchain(s): trusted public ledger(s) publishing transactions regularly

Time lock: lower bound on the publication time of a transaction



Takeaway message

Enforcing fairness often requires:

Complex cryptographic interactions with a third party

Intricate real-time mechanisms

Takeaway message

Enforcing fairness often requires:

Complex cryptographic interactions with a third party

Intricate real-time mechanisms

✔ Good support from existing verification techniques

Takeaway message

Enforcing fairness often requires:

Complex cryptographic interactions with a third party

✔ Good support from existing verification techniques

Intricate real-time mechanisms

?

Specifying real-time protocols

Planned features

```
// case 1: B also locked its coin within time Delta: continue
@t : t < t_start + Delta : inLedger(BLock) :

  // pre-sign. of A's payment
  let gx = st(x) in
  let in_ptx_APay = { nSequence = 0 ; id = id_ALock ; index = 0 } in
  let out_ptx_APay = { CLTV = 0 ; CSV = 0 ; pub = pkB' } in
  let ptx_APay = {
    id = id_APay ; nLockTime = 0 ;
    ins = [| in_ptx_APay |] ;
    outs = [| out_ptx_APay |] ;
  } in
  new r3;
  let hsigA_ptx_APay = hpresign(ptx_APay,r3,skA',pkB',gx) in
```

Planned features

```
// case 1: B also locked its coin within time Delta: continue
@t : t < t_start + Delta : inLedger(BLock) :

  // pre-sign. of A's payment
  let gx = st(x) in
  let in_ptx_APay = { nSequence = 0 ; id = id_ALock ; index = 0 } in
  let out_ptx_APay = { CLTV = 0 ; CSV = 0 ; pub = pkB' } in
  let ptx_APay = {
    id = id_APay ; nLockTime = 0 ;
    ins = [| in_ptx_APay |] ;
    outs = [| out_ptx_APay |] ;
  } in
  new r3;
  let hsigA_ptx_APay = hpresign(ptx_APay,r3,skA',pkB',gx) in
```

record types

Planned features

```
// case 1: B also locked its coin within time Delta: continue
@t : t < t_start + Delta : inLedger(BLock) :

  // pre-sign. of A's payment
  let gx = st(x) in
  let in_ptx_APay = { nSequence = 0 ; id = id_ALock ; index = 0 } in
  let out_ptx_APay = { CLTV = 0 ; CSV = 0 ; pub = pkB' } in
  let ptx_APay = {
    id = id_APay ; nLockTime = 0 ;
    ins = [| in_ptx_APay |] ;
    outs = [| out_ptx_APay |] ;
  } in
  new r3;
  let hsigA_ptx_APay = hpresign(ptx_APay,r3,skA',pkB',gx) in
```

record types

fixed-size array?

reduction to the binary case?

Be careful of standard frameworks!

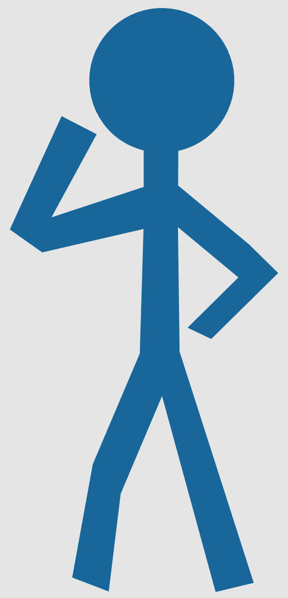
When modelling fairness, the ability to perform some action or not has *consequences*

Be careful of standard frameworks!

When modelling fairness, the ability to perform some action or not has *consequences*

Option 1:
Wait for a transaction until time t

Option 2:
Fallback solution

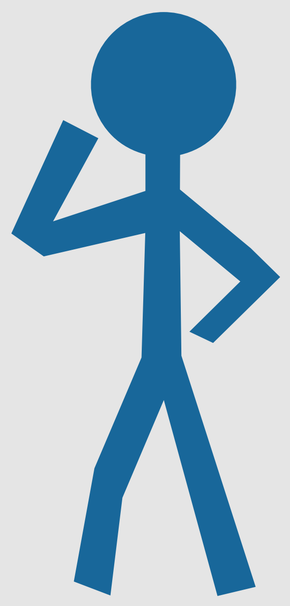


Be careful of standard frameworks!

When modelling fairness, the ability to perform some action or not has *consequences*

Option 1:
Wait for a transaction until time *t*

Option 2:
Fallback solution



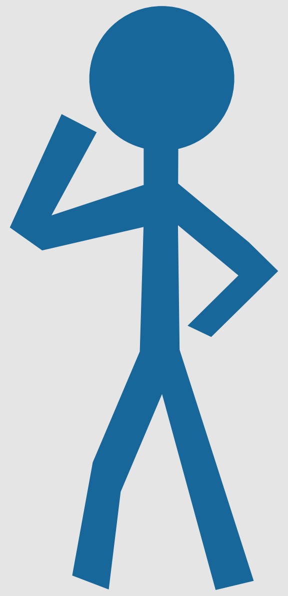
Progress: one option should be chosen, if possible

Be careful of standard frameworks!

When modelling fairness, the ability to perform some action or not has *consequences*

Option 1:
Wait for a transaction until time *t*

Option 2:
Fallback solution



Progress: one option should be chosen, if possible

Atomicity: certain operations should be done "*simultaneously*"

Be careful of standard frameworks!

When modelling fairness, the ability to perform some action or not has *consequences*

Option 1:
Wait for a transaction until time *t*

Option 2:
Fallback solution

Progress: one option should be chosen, if possible

Atomicity: certain operations should be done “*simultaneously*”

Process syntax

Parallel

P | Q

Replication

! P

Choice

P + Q

nil

0

tic

tic : P

output

out(u) : P

input

in(x) : P

event

Ev(ũ) : P

time stamp

@ t : P

time condition

when b : P

Process syntax

Structural operators

Parallel	Replication	Choice	nil
P | Q	! P	P + Q	0

tic	output	input	event
tic : P	**out**(u) : P	**in**(x) : P	**Ev**(ũ) : P

time stamp	time condition
@ t : P	**when** b : P

Process syntax

Structural operators

Parallel	Replication	Choice	nil
P | Q	! P	P + Q	0

Communications

tic	output	input	event
tic : P	**out**(u) : P	**in**(x) : P	**Ev**(ũ) : P

time stamp	time condition
@ *t* : P	**when** *b* : P

Process syntax

Structural operators

Parallel	Replication	Choice	nil
P | Q	! P	P + Q	0

Communications · *Security prop.*

tic	output	input	event
tic : P	**out**(u) : P	**in**(x) : P	**Ev**(ũ) : P

time stamp	time condition
@t : P	**when** b : P

Process syntax

Structural operators

Parallel	Replication	Choice	nil
P | Q	! P	P + Q	0

tic
tic : P

Communications

output	input
out(u) : P	**in**(x) : P

Security prop.

event
Ev(ũ) : P

time stamp	time condition
@t : P	**when** b : P

Process syntax

Structural operators

Parallel	Replication	Choice	nil
P | Q	! P	P + Q	0

Communications *Security prop.*

tic	output	input	event
tic : P	**out**(u) : P	**in**(x) : P	**Ev**(ũ) : P

Time management

time stamp	time condition
@ *t* : P	**when** *b* : P

Process syntax

Structural operators

Parallel	Replication	Choice	nil
P | Q	! P	P + Q	0

???

tic
tic : P

Communications

output	input
out(u) : P	**in**(x) : P

Security prop.

event
Ev(ũ) : P

Time management

time stamp	time condition
@ t : P	**when** b : P

Atomicity

$$\mathit{instr} \,;\, \mathrm{P} \;=\; \mathit{instr} : \mathbf{tic} : \mathrm{P}$$

Atomicity

$$instr \,;\, \mathrm{P} \;=\; instr : \mathbf{tic} : \mathrm{P}$$

$$\mathrm{in}(x) : \mathbf{Get}(x) : @t' : \mathbf{when}\ t' < t : \mathbf{tic} : \mathbf{Ans}(u) : \mathrm{out}(u) : \mathbf{tic} : \mathrm{P}$$

Atomicity

$$\mathit{instr}\ ;\ \mathrm{P} \;=\; \mathit{instr} : \textbf{tic} : \mathrm{P}$$

$$\underbrace{\mathrm{in}(x) : \textbf{Get}(x) : @t' : \textbf{when}\ t' < t}_{\text{instruction 1}} : \textbf{tic} : \underbrace{\textbf{Ans}(u) : \mathrm{out}(u)}_{\text{instruction 2}} : \textbf{tic} : \mathrm{P}$$

Planned features (again)

```
// case 1: B also locked its coin within time Delta: continue
@t : t < t_start + Delta : inLedger(BLock) :

  // pre-sign. of A's payment
  let gx = st(x) in
  let in_ptx_APay = { nSequence = 0 ; id = id_ALock ; index = 0 } in
  let out_ptx_APay = { CLTV = 0 ; CSV = 0 ; pub = pkB' } in
  let ptx_APay = {
    id = id_APay ; nLockTime = 0 ;
    ins = [| in_ptx_APay |] ;
    outs = [| out_ptx_APay |] ;
  } in
  new r3;
  let hsigA_ptx_APay = hpresign(ptx_APay,r3,skA',pkB',gx) in
```

record types

fixed-size array?

reduction to the binary case?

Planned features (again)


```
// case 1: B also locked its coin within time Delta: continue
@t : t < t_start + Delta : inLedger(BLock) :

  // pre-sign. of A's payment
  let gx = st(x) in
  let in_ptx_APay = { nSequence = 0 ; id = id_ALock ; index = 0 } in
  let out_ptx_APay = { CLTV = 0 ; CSV = 0 ; pub = pkB' } in
  let ptx_APay = {
    id = id_APay ; nLockTime = 0 ;
    ins = [| in_ptx_APay |] ;
    outs = [| out_ptx_APay |] ;
  } in
  new r3;
  let hsigA_ptx_APay = hpresign(ptx_APay,r3,skA',pkB',gx) in
```

record types

fixed-size array?

reduction to the binary case?

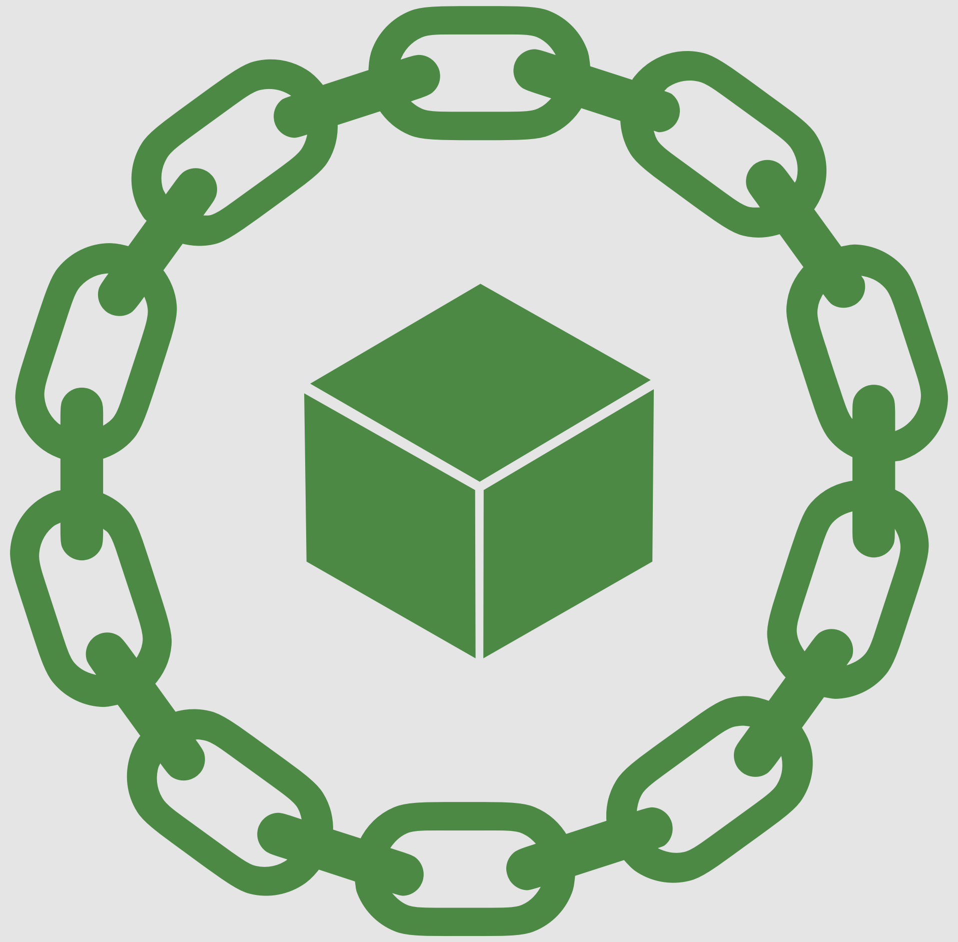
Specifying temporal properties

*Security properties: tidy CTL**

G. Barthe, U. Dal Lago, G. Malavolta, I. Rakotonirina, 2022.
Tidy: symbolic verification of timed cryptographic protocols

*Security properties: tidy CTL**

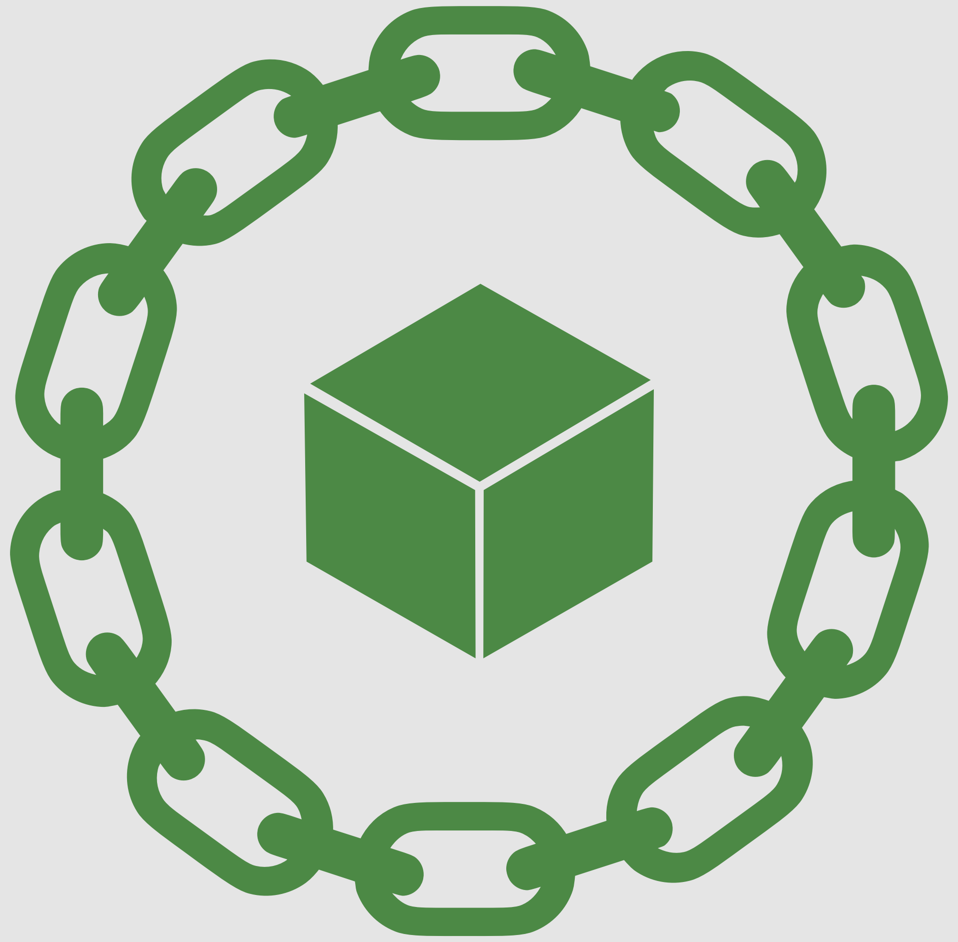
G. Barthe, U. Dal Lago, G. Malavolta, I. Rakotonirina, 2022.
Tidy: symbolic verification of timed cryptographic protocols



axiomatisation of a blockchain with time locks

Security properties: tidy CTL*

G. Barthe, U. Dal Lago, G. Malavolta, I. Rakotonirina, 2022.
Tidy: symbolic verification of timed cryptographic protocols



axiomatisation of a blockchain with time locks

Examples:

$$\mathbf{G}(\forall\, tx.\ \mathbf{Publish}(tx) \Rightarrow \mathbf{F}\ \mathit{doubleSpend}(tx) \Rightarrow \bot)$$

"at any point in the future"

"at some point in the future"

$$\forall\, tx.\ \mathbf{F}_{\mathrm{timeLock}(tx)}\ \mathbf{Publish}(tx) \Rightarrow \bot$$

"at some point in at most timeLock(tx) units of time"

Verifying real-time protocols?

(work in progress)

Verification chain

Calculus of concurrent processes

Proof / Attack

Verification chain

Calculus of concurrent processes

Sapic

Multiset rewrite rules

Proof / Attack

Verification chain

Calculus of concurrent processes

↓ ***Sapic***

Multiset rewrite rules

↓ ***Tamarin prover***

Proof / Attack

Verification chain

Calculus of concurrent processes

+ atomicity

+ real-time

Sapic

Multiset rewrite rules

Tamarin prover

Proof / Attack

Verification chain

Calculus of concurrent processes

+ atomicity

+ real-time

Sapic

approximate untimed model

Multiset rewrite rules

Tamarin prover

Proof / Attack

Verification chain

Calculus of concurrent processes

+ atomicity

+ real-time

Sapic

approximate untimed model

Multiset rewrite rules

Tamarin prover

Proof / Attack

Verification chain

+ atomicity

+ real-time

Calculus of concurrent processes

Sapic

approximate untimed model

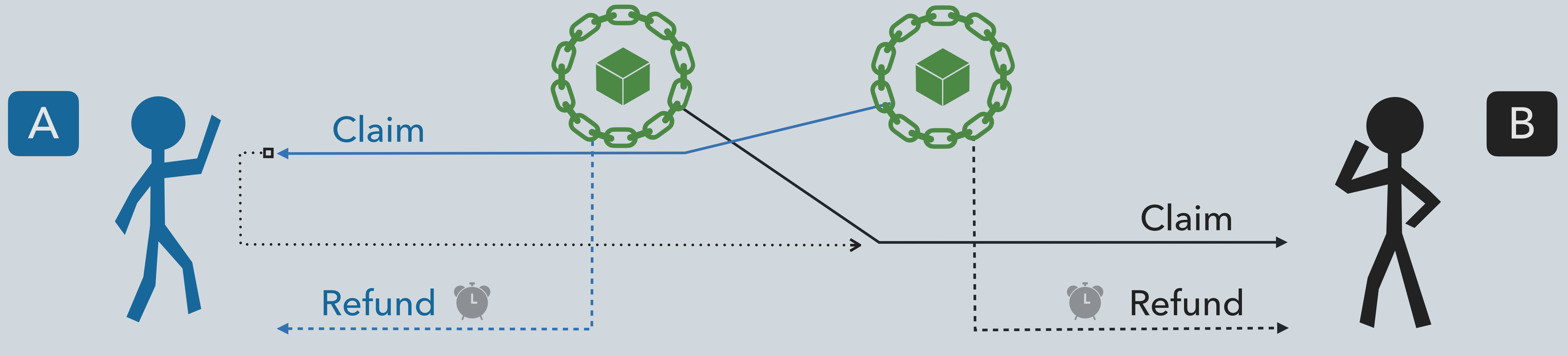
Multiset rewrite rules

Tamarin prover

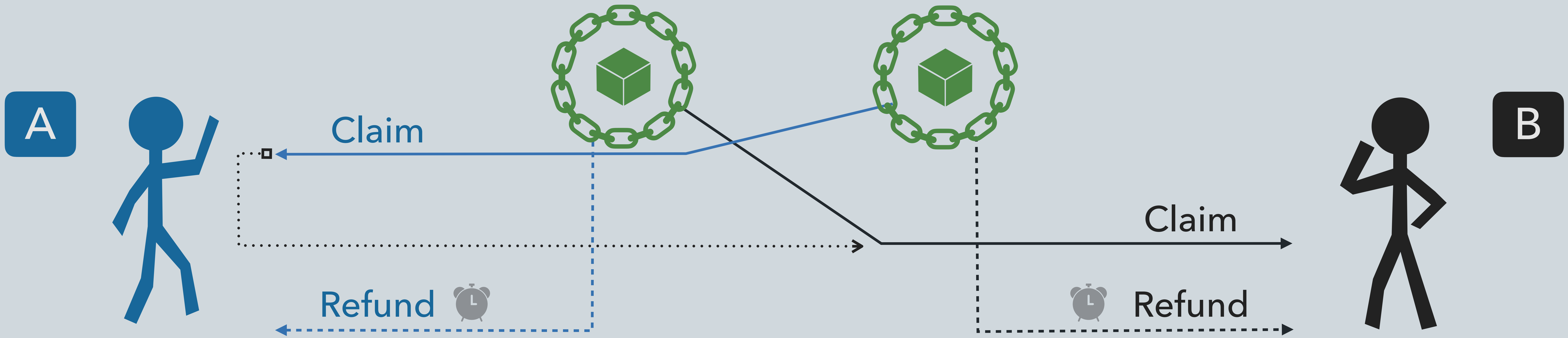
Proof / Attack

refinement if unsound attack

Typical unsound attacks

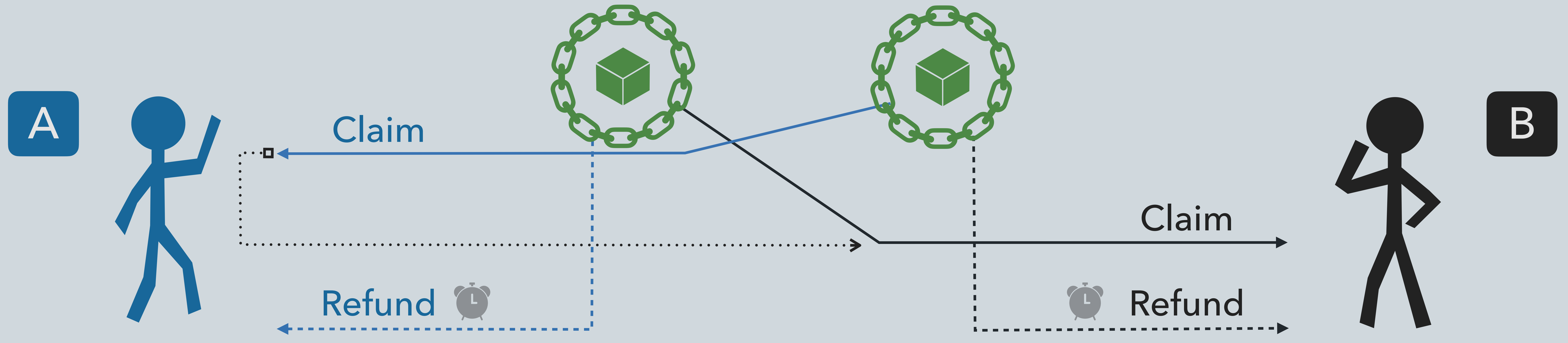


Typical unsound attacks



Submit (Claim$_A$); Submit(Refund$_A$); Publish (Claim$_A$); Publish (Refund$_A$)

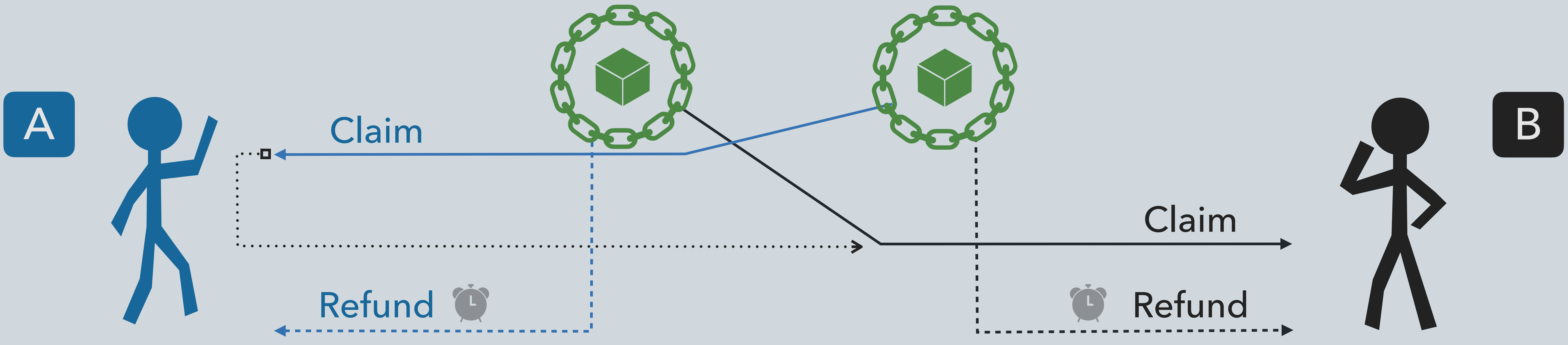
Typical unsound attacks



Submit(Claim$_A$); Submit(Refund$_A$); Publish(Claim$_A$); Publish(Refund$_A$)

⚠ *A's Refund timelock*

Typical unsound attacks

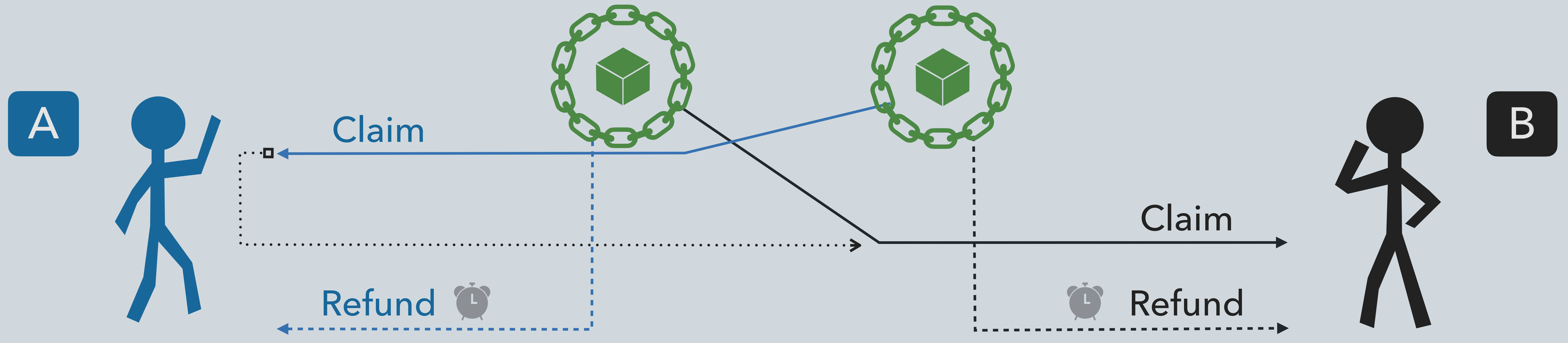


Submit (Claim$_A$); Submit(Refund$_A$); Publish (Claim$_A$); Publish (Refund$_A$)

⚠ *A's Refund timelock*

Submit (Claim$_A$); Publish (Claim$_A$); Submit (Refund$_A$); Publish (Refund$_A$)

Typical unsound attacks



Submit (Claim$_A$); Submit(Refund$_A$); Publish (Claim$_A$); Publish (Refund$_A$)

⚠ *A's Refund timelock*

Submit (Claim$_A$); Publish (Claim$_A$); Submit (Refund$_A$); Publish (Refund$_A$)

⚠ *B's fast reactiveness*

Conclusion

A calculus supporting real time and atomicity

Formalisation of a Blockchain with time locks

In progress: extending/adapt Sapic to fit the workflow