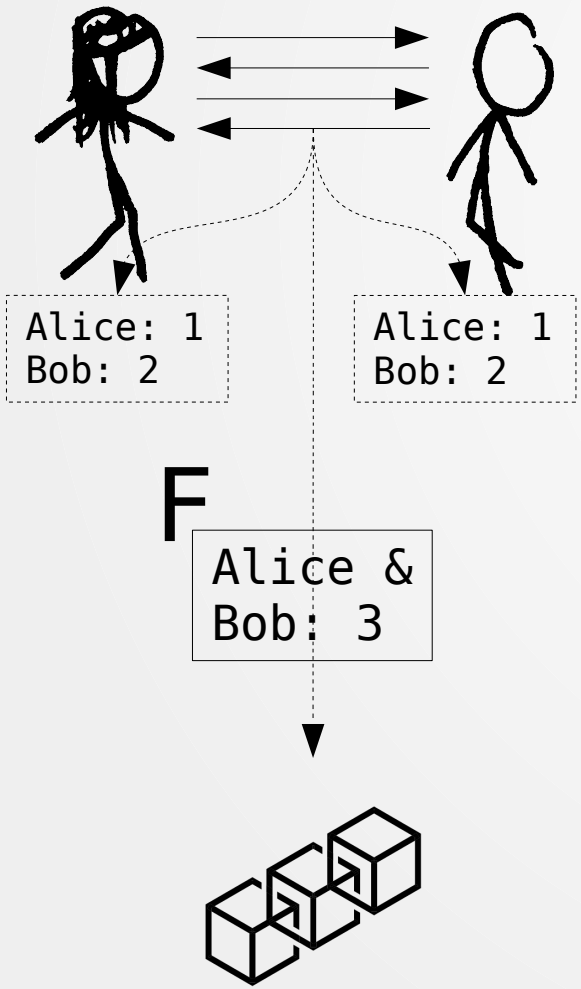
A Composable Security Treatment of the Lightning Network

Aggelos Kiayias
Orfeas Stefanos Thyfronitis Litos
University of Edinburgh
30/9/2020

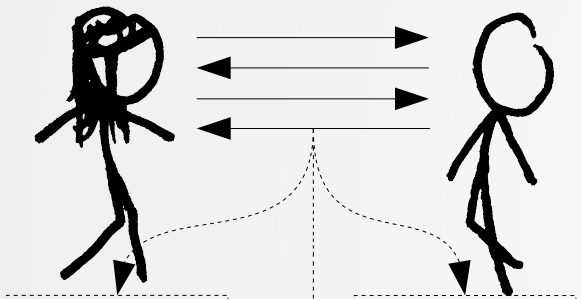
Lightning Channels

Open

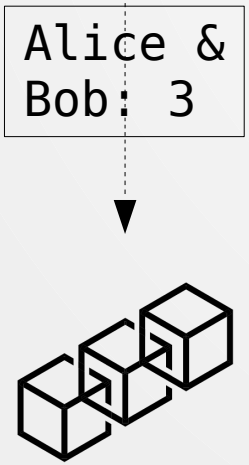


Lightning Channels

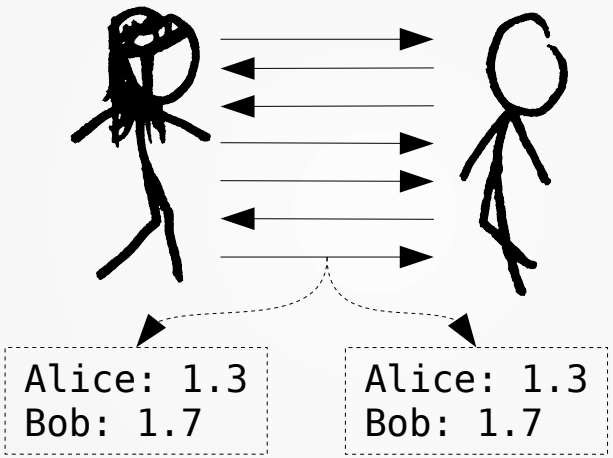
Open

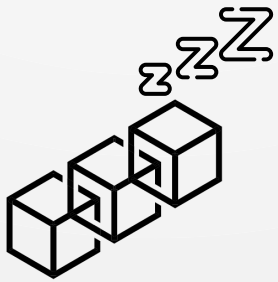


F



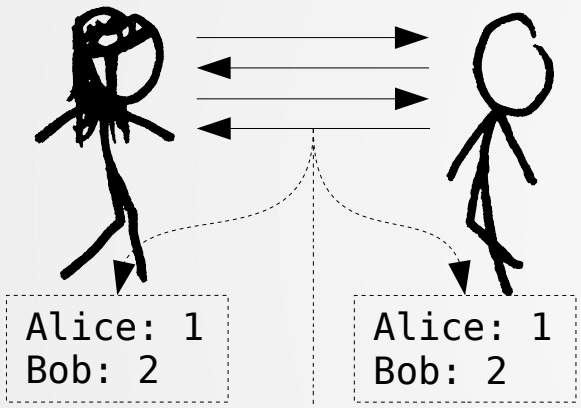
Pay



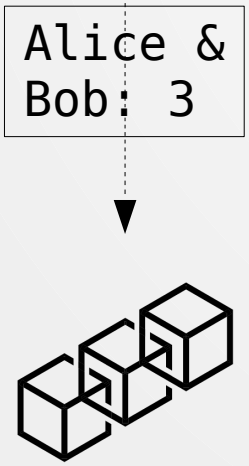


Lightning Channels

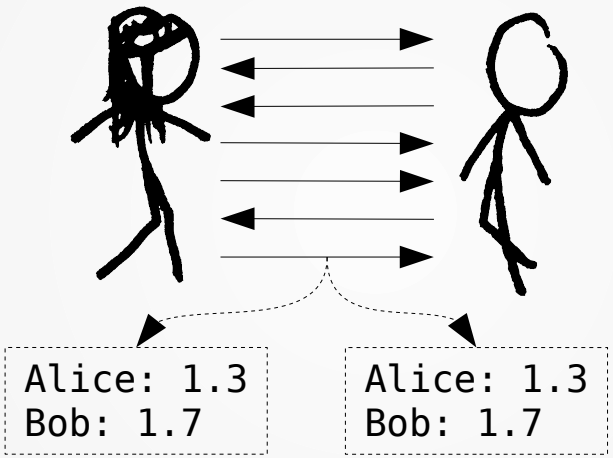
Open



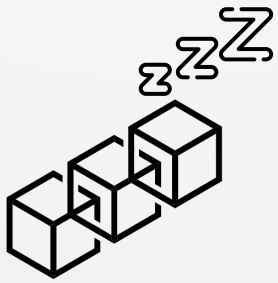
F



Pay

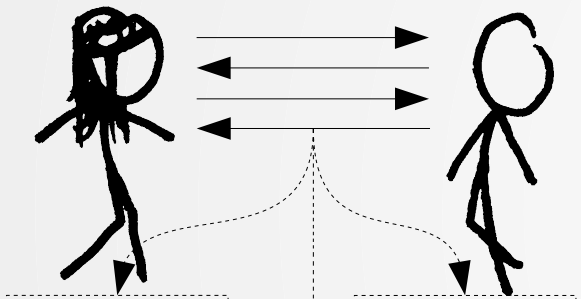


- Unlimited times
- No touching blockchain



Lightning Channels

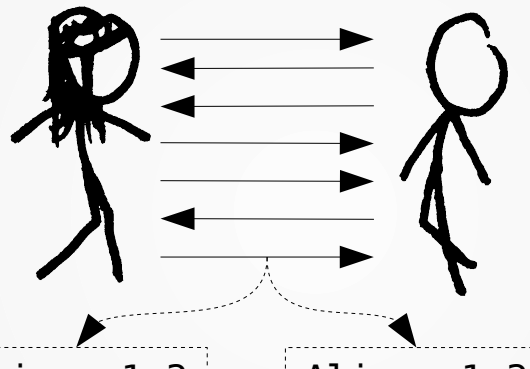
Open



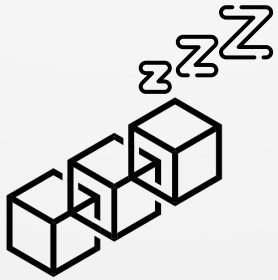
F

Alice &
Bob: 3

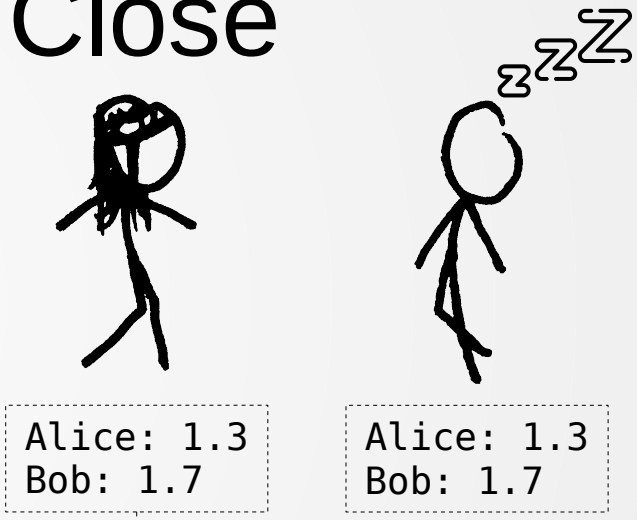
Pay



- Unlimited times
- No touching blockchain

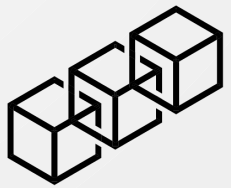


Close



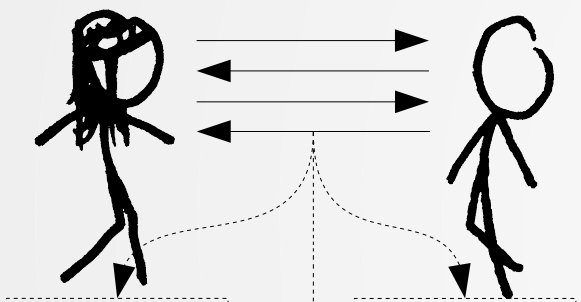
S

Alice: 1.3
Bob: 1.7



Lightning Channels

Open



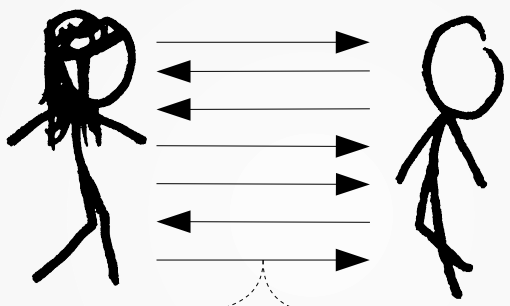
Alice: 1
Bob: 2

Alice: 1
Bob: 2

F

Alice &
Bob: 3

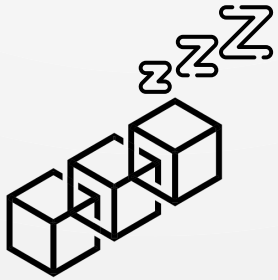
Pay



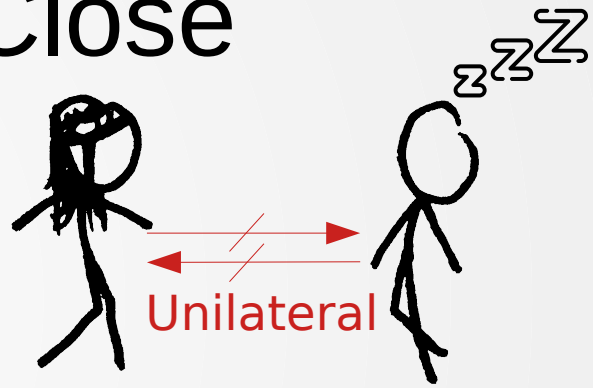
Alice: 1.3
Bob: 1.7

Alice: 1.3
Bob: 1.7

- Unlimited times
- No touching blockchain



Close

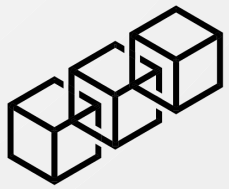


Alice: 1.3
Bob: 1.7

Alice: 1.3
Bob: 1.7

S

Alice: 1.3
Bob: 1.7

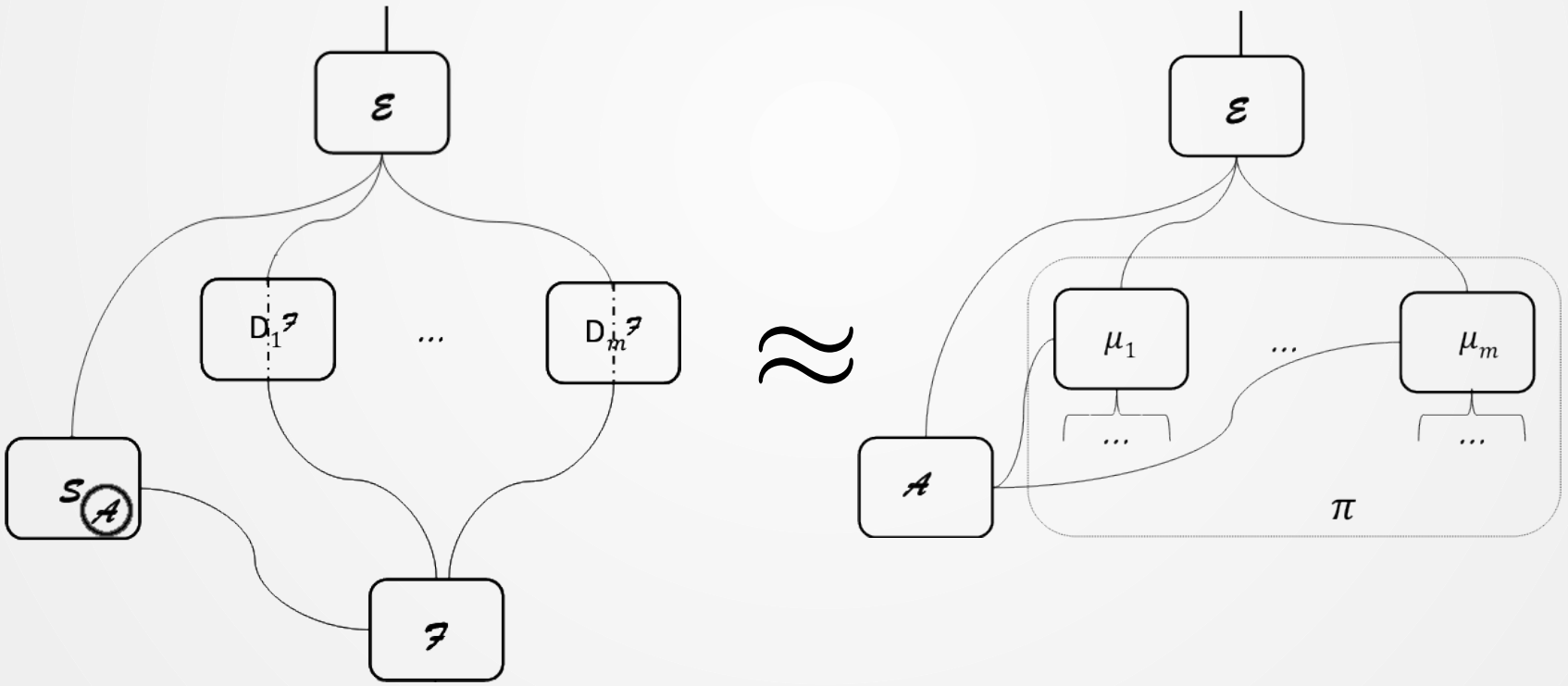


Multi-hop payments

From channels to network!

Simulation-based Security

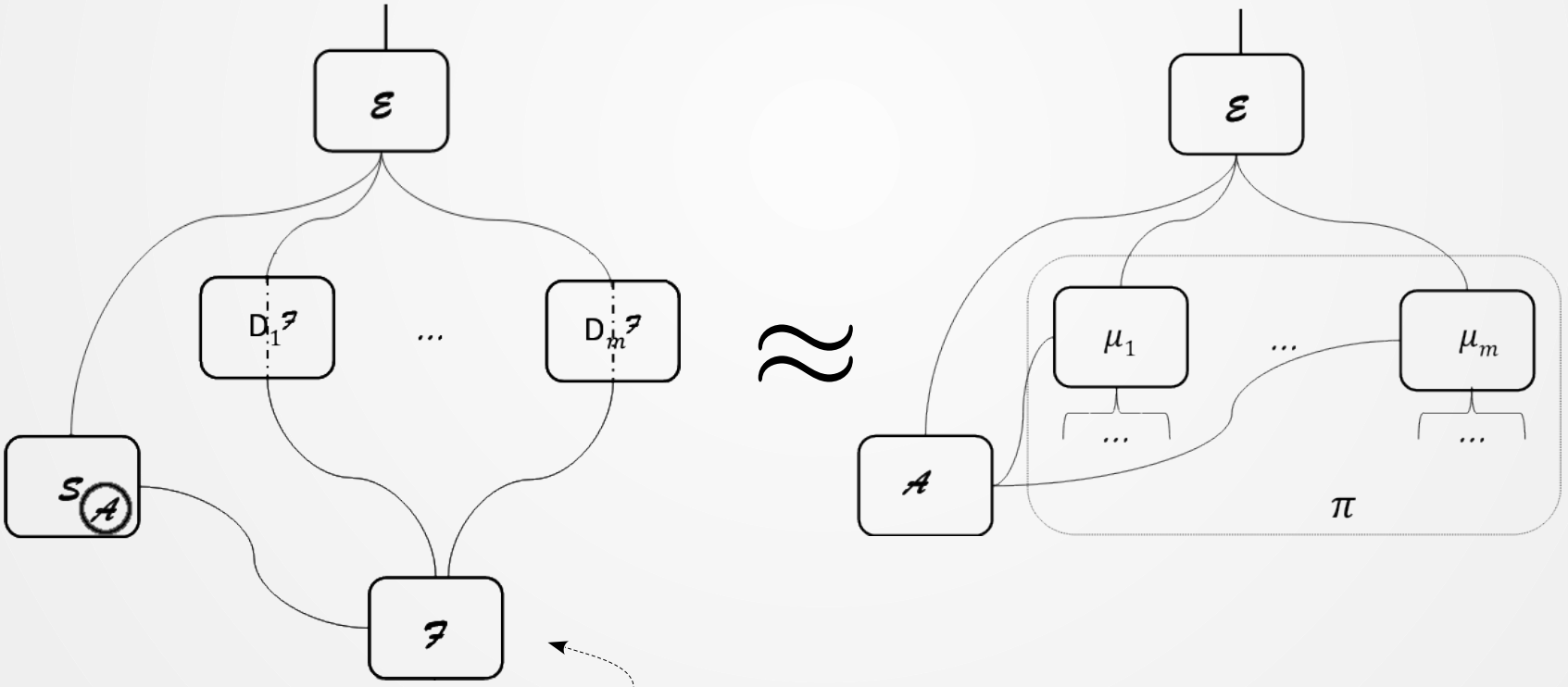
$$\forall \mathcal{E}, \mathcal{A}\ \exists \mathcal{S}:$$



Credits: “Universally Composable Security”, Ran Canetti
https://eprint.iacr.org/2000/067

Our paper

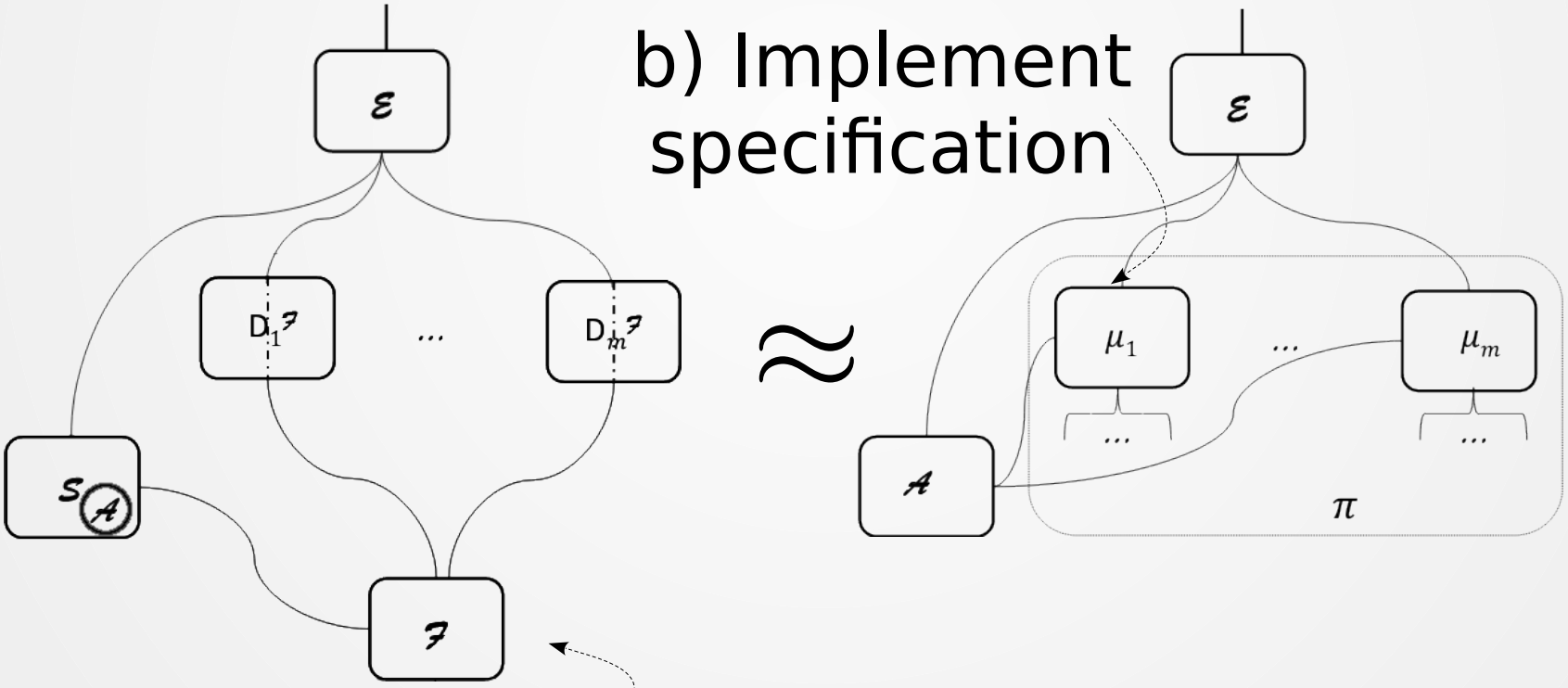
$$\forall \mathcal{E}, \mathcal{A}\ \exists \mathcal{S}:$$



a) Define functionality

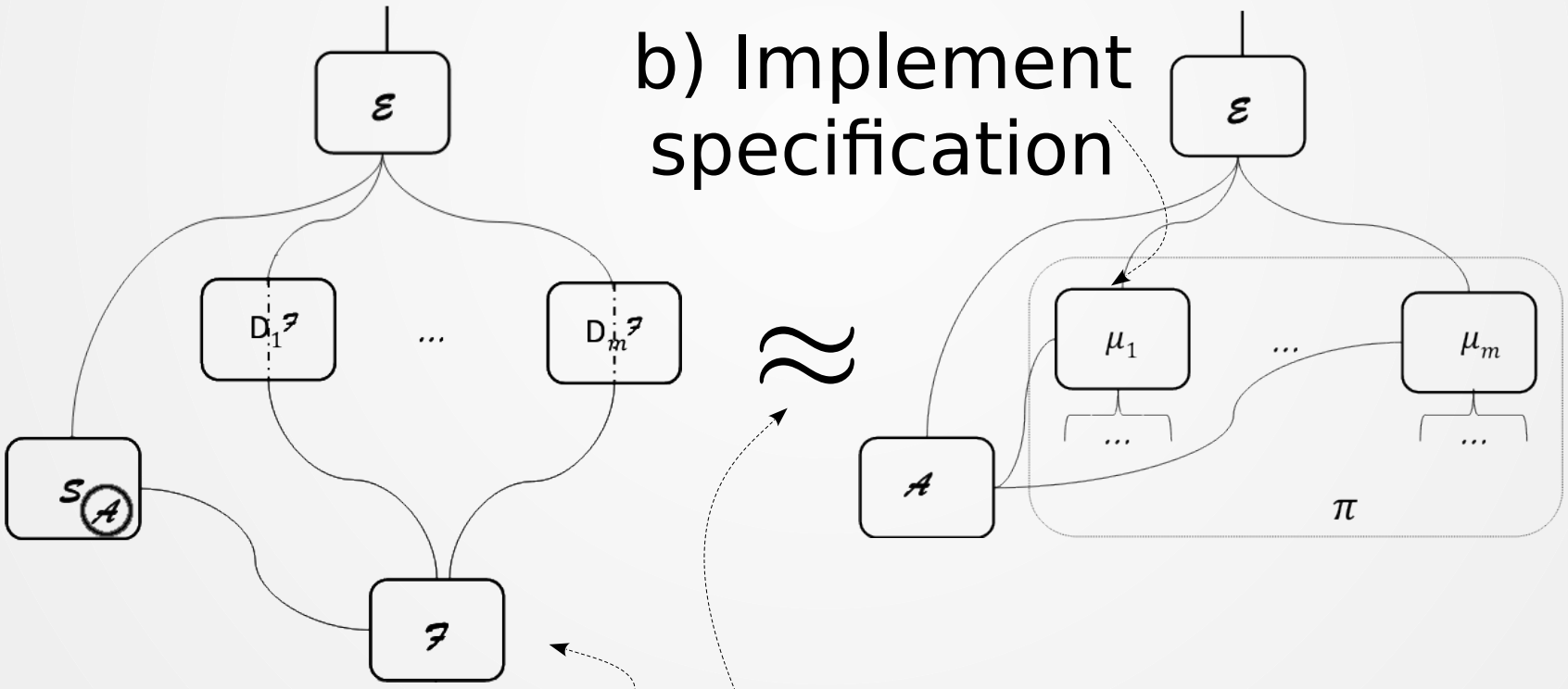
Our paper

$$\forall \mathcal{E}, \mathcal{A}\ \exists \mathcal{S}:$$



Our paper

$$\forall \mathcal{E}, \mathcal{A}\ \exists \mathcal{S}:$$



b) Implement specification

a) Define functionality

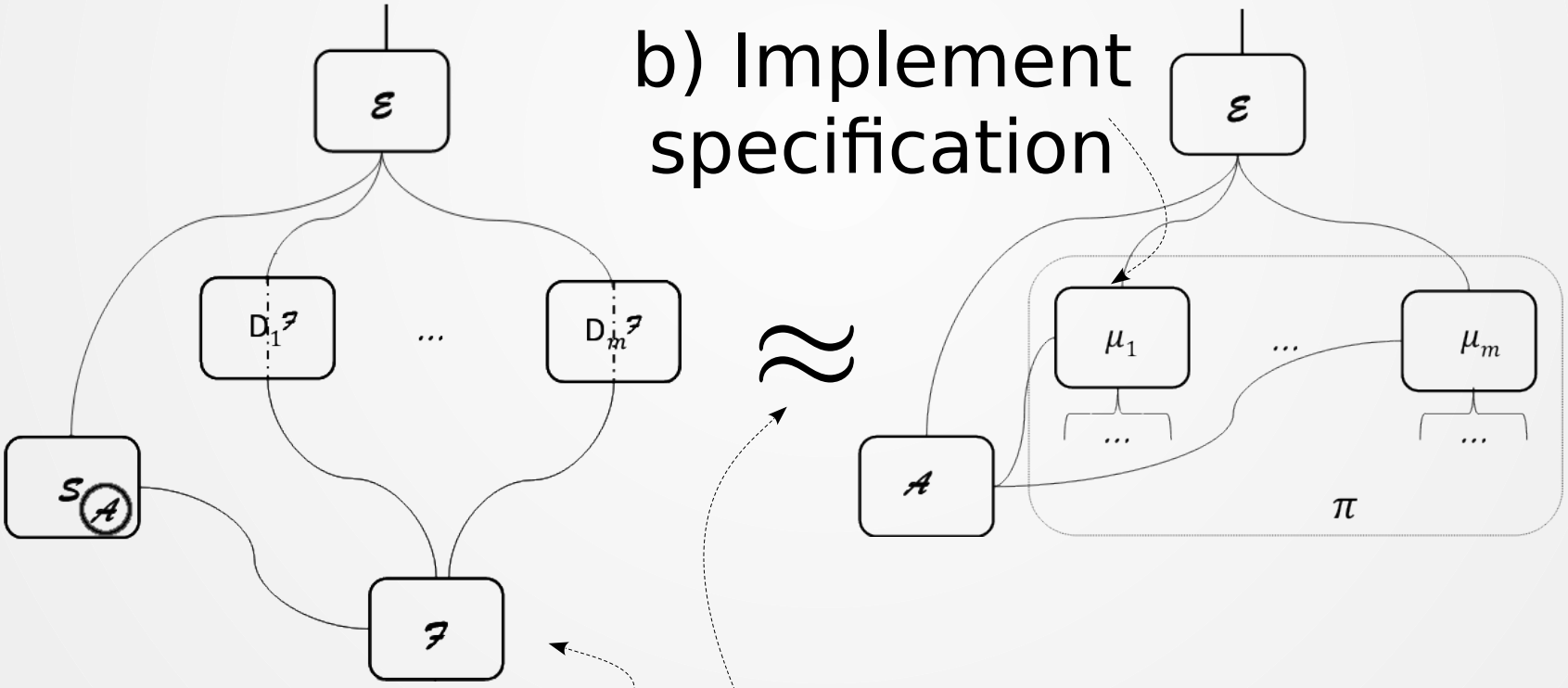
c) Prove indistinguishability

Our paper

$\forall \mathcal{E}, \mathcal{A}\ \exists \mathcal{S}$:

d) Prove naive ledger unrealizable

b) Implement specification



a) Define functionality

c) Prove indistinguishability

Functionality

Functionality $\mathcal{F}_{\text{PayNet}}$ – interface

- from $\mathcal{E}$:
  - (REGISTER, delay, relayDelay)
  - (TOPPEDUP)
  - (OPENCHANNEL, *Alice*, *Bob*, x, *tid*)
  - (CHECKFORNEW, *Alice*, *Bob*, *tid*)
  - (PAY, *Bob*, x, $\overrightarrow{\texttt{path}}$, `receipt`)
  - (CLOSECHANNEL, `receipt`, *pchid*)
  - (FORCECLOSECHANNEL, `receipt`, *pchid*)
  - (POLL)
  - (PUSHFULFILL, *pchid*)
  - (PUSHADD, *pchid*)
  - (COMMIT, *pchid*)
  - (FULFILLONCHAIN)
  - (GETNEWS)
- to $\mathcal{E}$:
  - (REGISTER, *Alice*, `delay`(*Alice*), `relayDelay`(*Alice*), pubKey)
  - (REGISTERED)
  - (NEWS, newChannels, closedChannels, updatesToReport)
- from $\mathcal{S}$:
  - (REGISTERDONE, *Alice*, pubKey)
  - (CORRUPTED, *Alice*)
  - (CHANNELANNOUNCED, *Alice*, $p_{Alice,F}$, $p_{Bob,F}$, *fchid*, *pchid*, *tid*)
  - (UPDATE, `receipt`, *Alice*)
  - (CLOSEDCHANNEL, `channel`, *Alice*)
  - (RESOLVEPAYS, *payid*, `charged`)
- to $\mathcal{S}$:
  - (REGISTER, *Alice*, delay, relayDelay)
  - (OPENCHANNEL, *Alice*, *Bob*, x, *fchid*, *tid*)
  - (CHANNELOPENED, *Alice*, *fchid*)
  - (PAY, *Alice*, *Bob*, x, $\overrightarrow{\texttt{path}}$, `receipt`, *payid*)
  - (CONTINUE)
  - (CLOSECHANNEL, *fchid*, *Alice*)
  - (FORCECLOSECHANNEL, *fchid*, *Alice*)
  - (POLL, Σ_{Alice}, *Alice*)
  - (PUSHFULFILL, *pchid*, *Alice*)
  - (PUSHADD, *pchid*, *Alice*)
  - (COMMIT, *pchid*, *Alice*)
  - (FULFILLONCHAIN, t, *Alice*)

Functionality

- Workhorse messages
  - (open_channel, Alice, Bob, x)
  - (pay, Bob, x, path, receipt)
  - ({,force}_close_channel, receipt, id)
- (poll) – sync and check for malicious closures
- (resolve_pays, charged) – HTLC resolutions
- check_closed(state)
- (get_news)

Thank you! Questions?

https://github.com/OrfeasLitos/PaymentChannels/

Image credits:

- https://www.flaticon.com/authors/freepik
- https://xkcd.com/