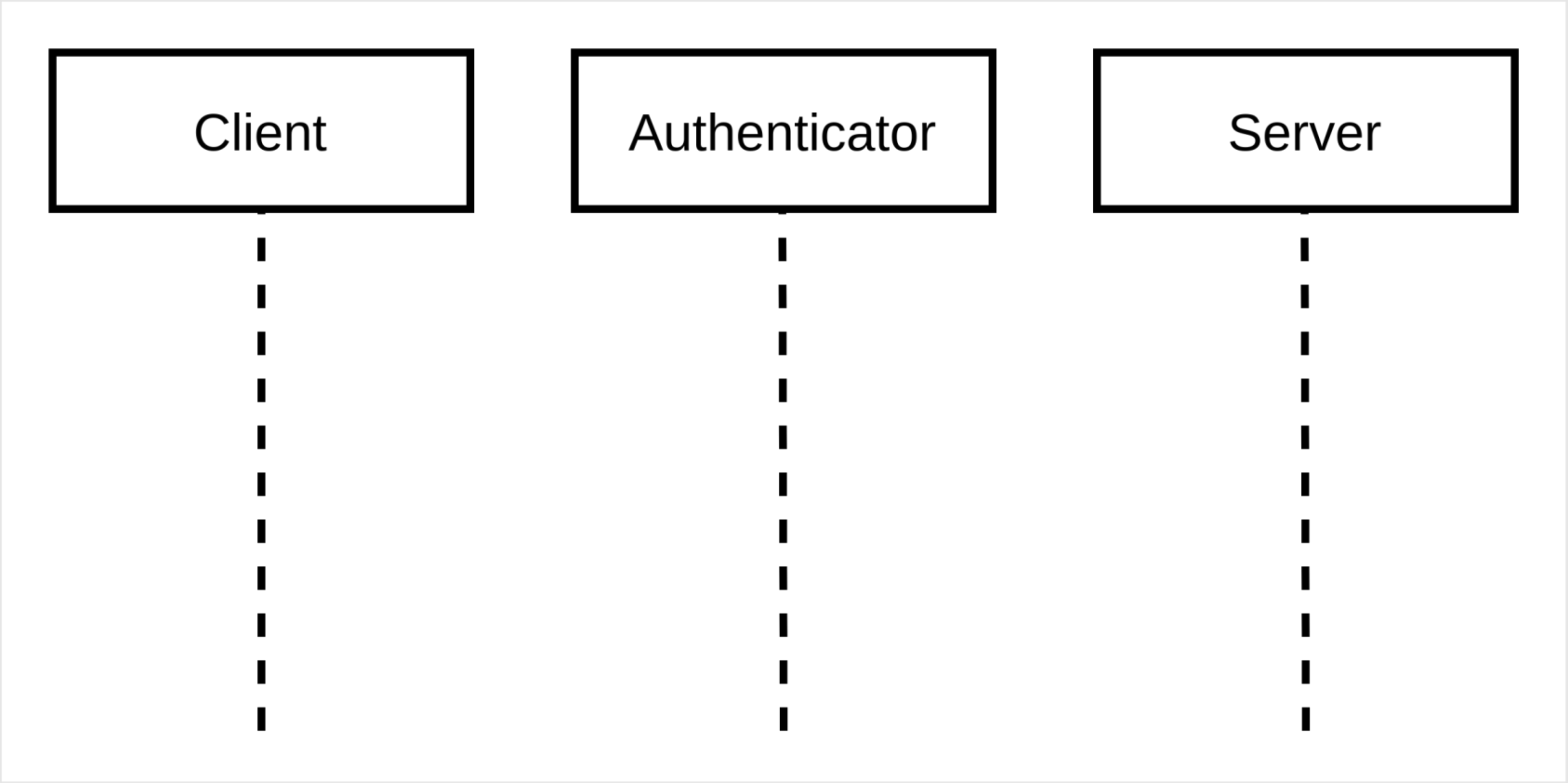
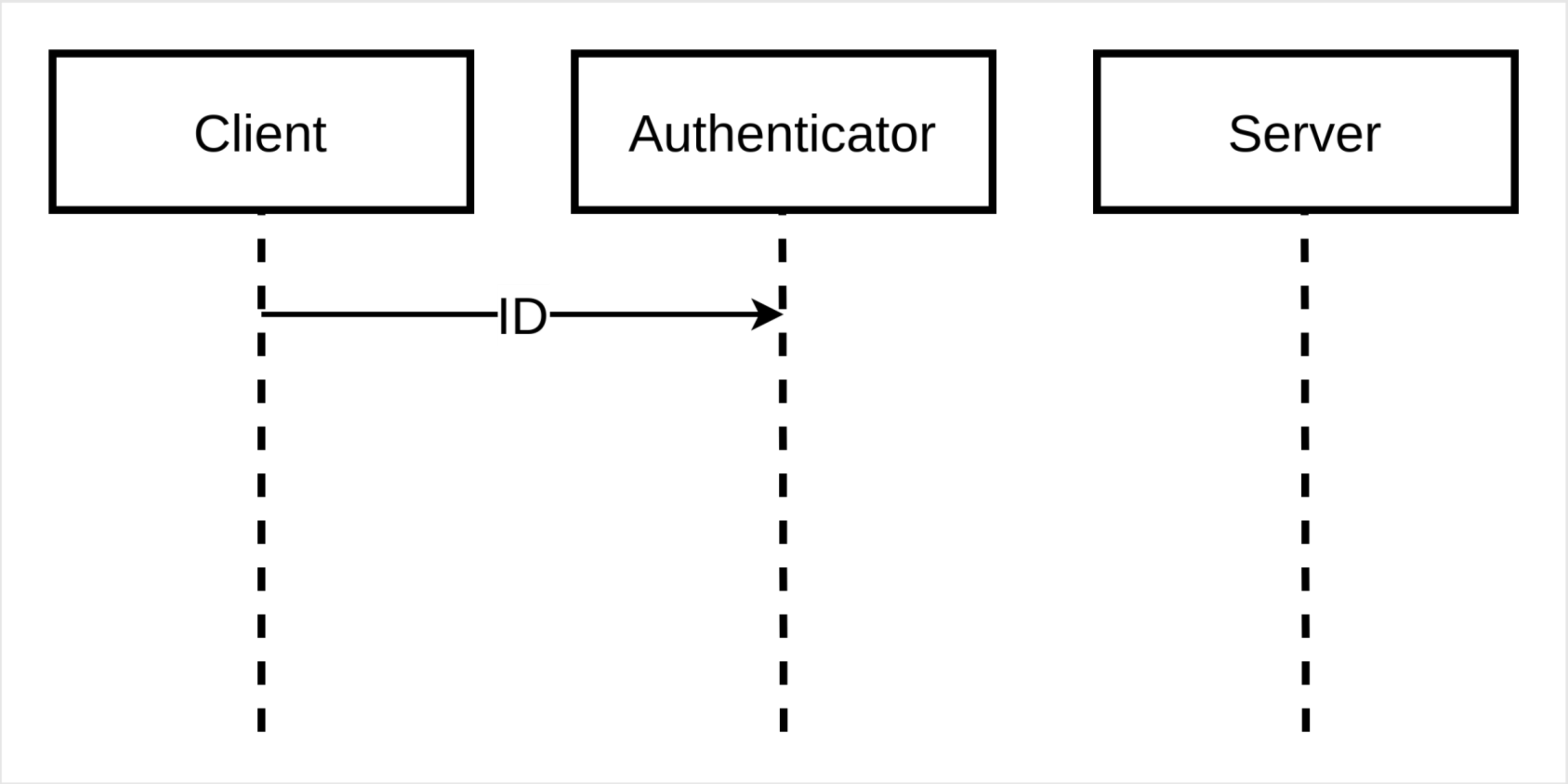
Stay Safe under Panic: Affine Rust Programming with Multiparty Session Types

Nicolas Lagaillardie, Rumyana Neykova and Nobuko Yoshida

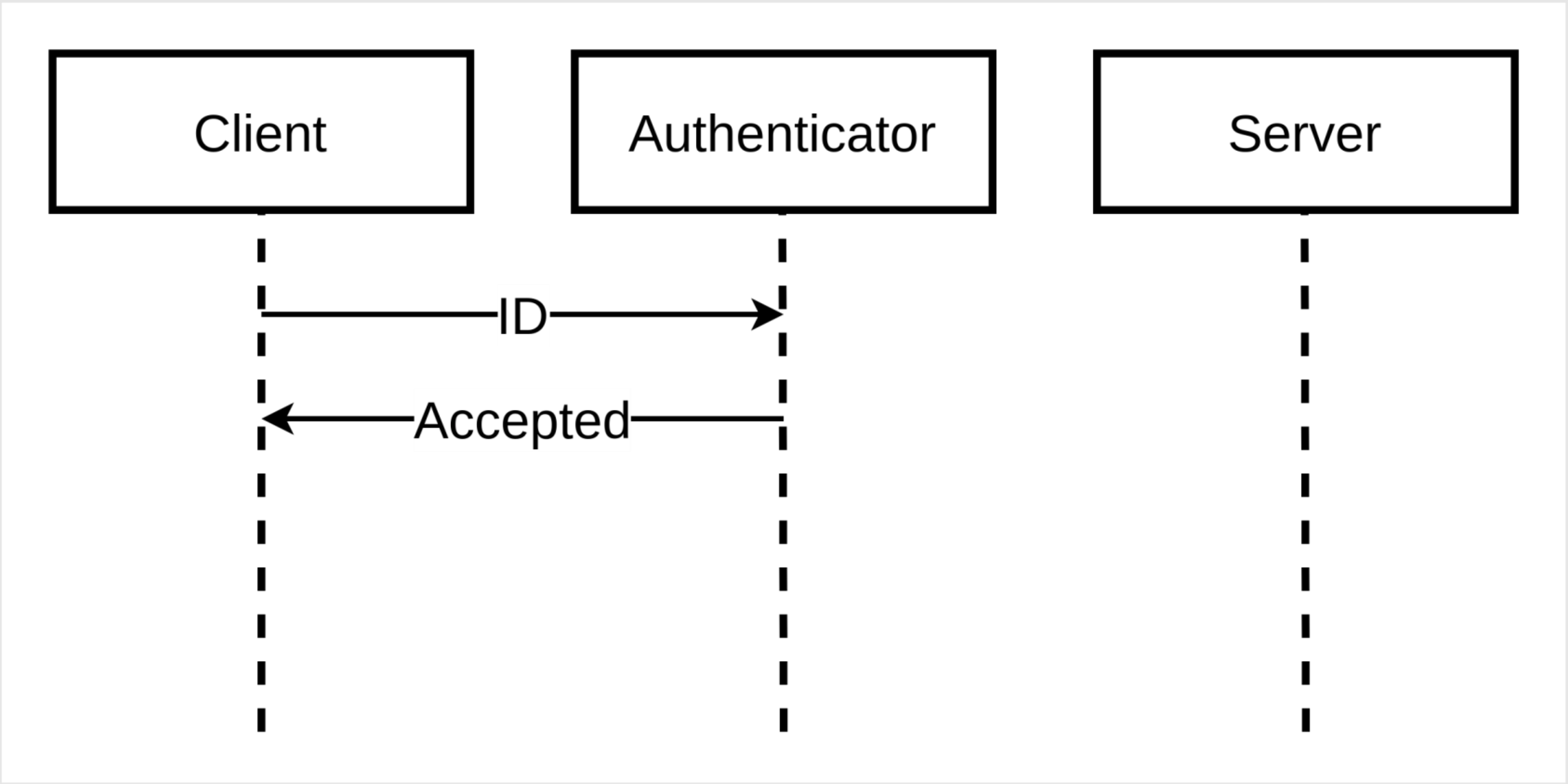
Stay Safe under Panic



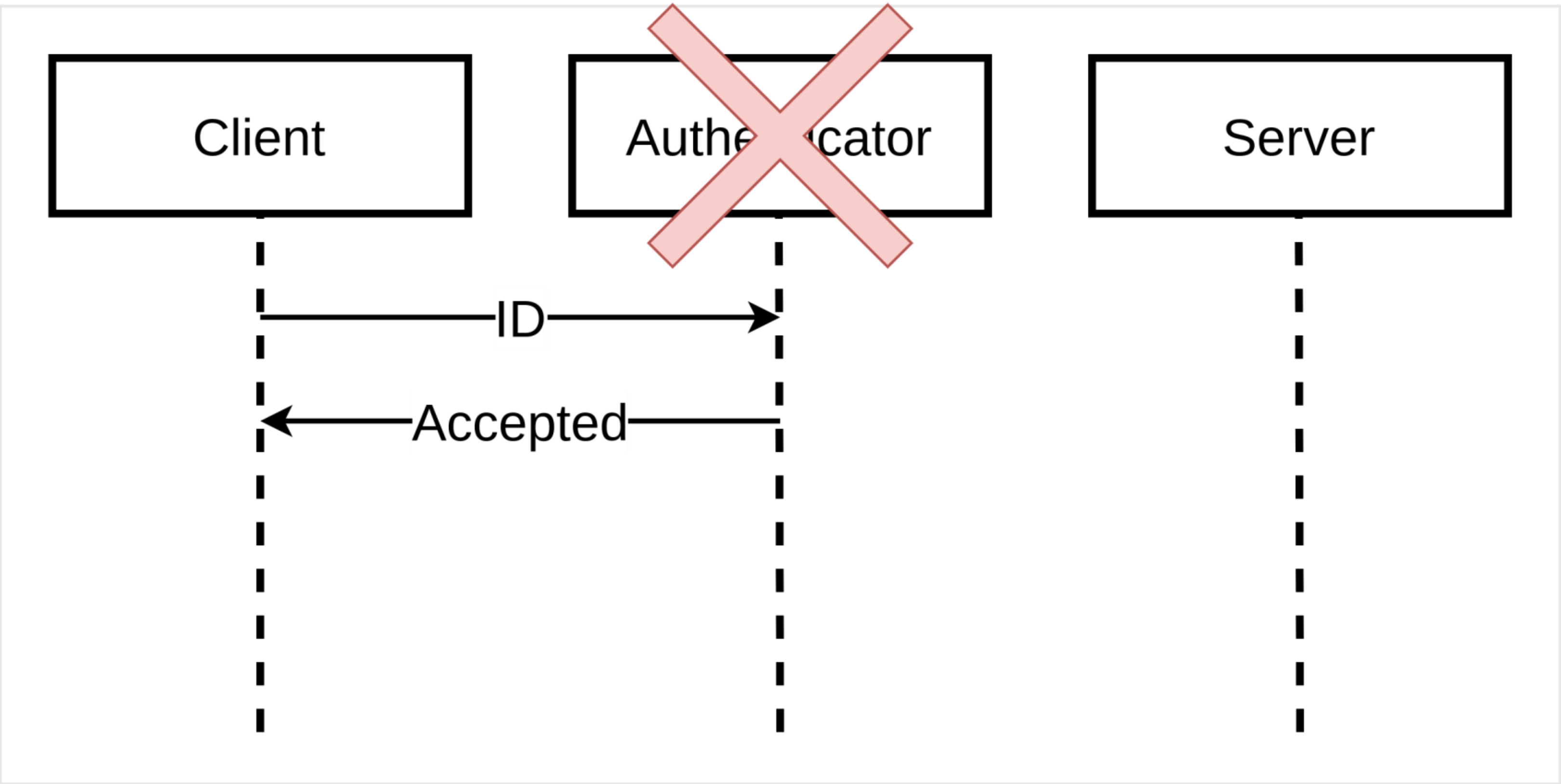
Stay Safe under Panic



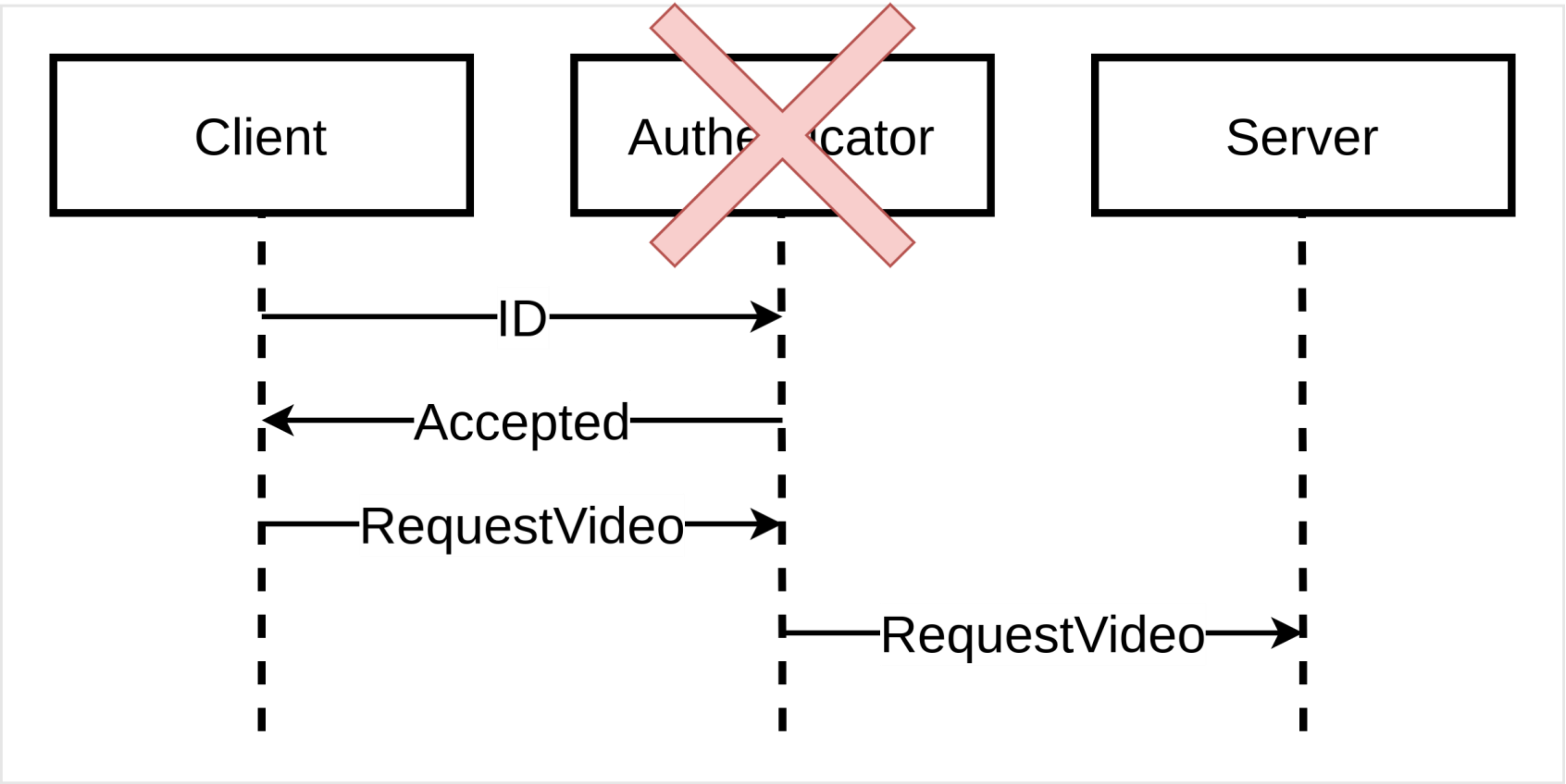
Stay Safe under Panic



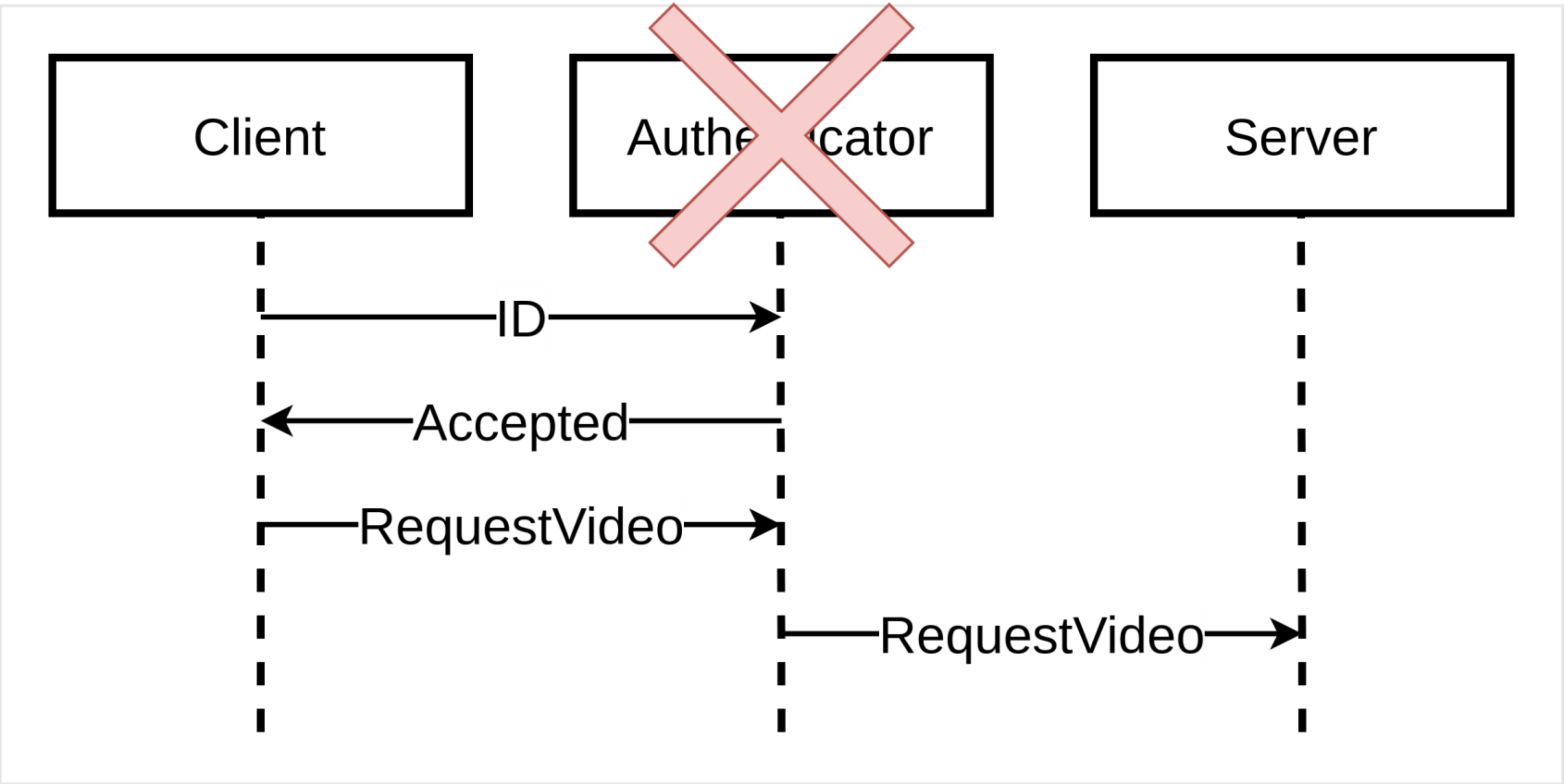
Stay Safe under Panic



Stay Safe under Panic



Stay Safe under Panic
→ **Client** and **Server** stuck forever?



Outline

Affine Multiparty Session Types (AMPST)

- Multiparty Session Types
- Affine Multiparty Session Types

Implementation in Rust: MultiCrusty

- Types and primitives
- Top-down approach

Summary and future work

Multiparty Session Types

- A framework to write and check communication protocols for at least 2 participants
  - Global protocol and local protocols

Multiparty Session Types

- A framework to write and check communication protocols for at least 2 participants
  - Global protocol and local protocols

- Three key properties:
  - Deadlock-freedom
  - Liveness
  - Safety

Session Types

Literature: MPST

Linear types

Session Types

Literature: MPST

Contribution: Affine MPST

Linear types → **Affine** types

Affine Multiparty Session Types

- Main idea: **cascading** the notification of the failure then **kill** the notified participants

Affine Multiparty Session Types

- Main idea: **cascading** the notification of the failure then **kill** the notified participants

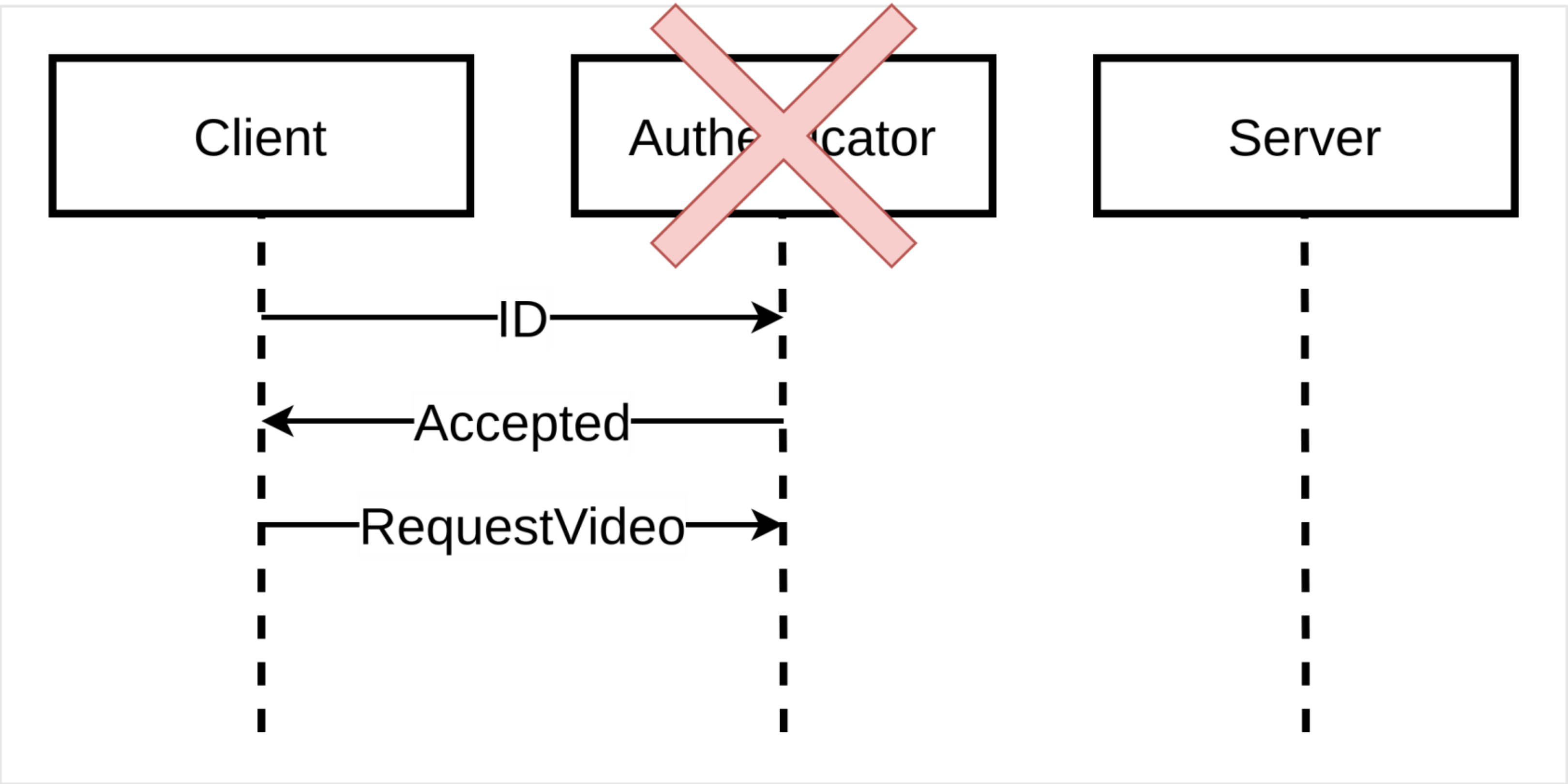
- Goal: **handling** failures at runtime while **preserving** *deadlock-freedom*, *liveness* and *safety*

Affine Multiparty Session Types

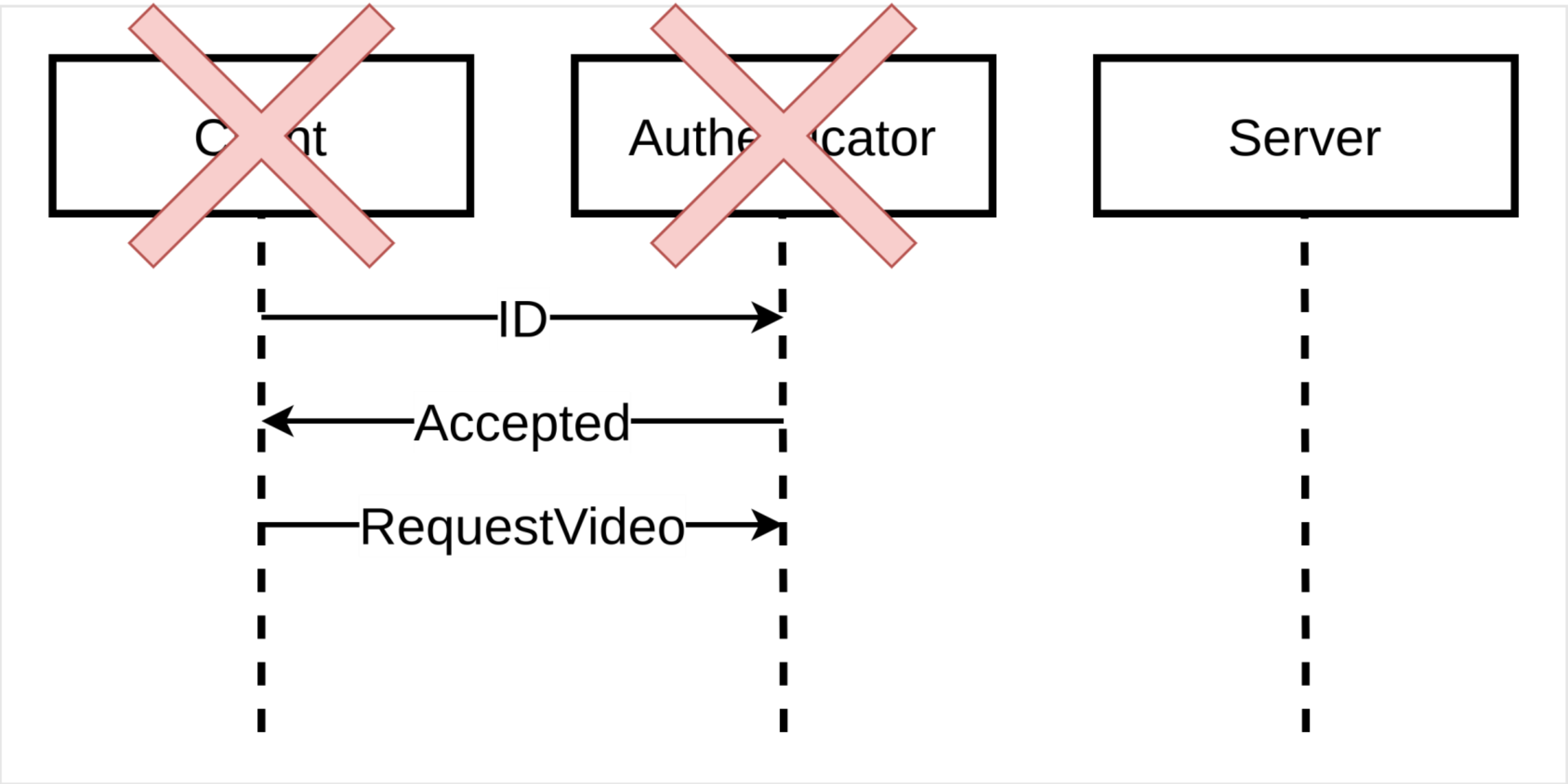
$\textbf{try}\ P\ \textbf{catch}\ \textbf{cancel}(c).Q$

$s\lightning$

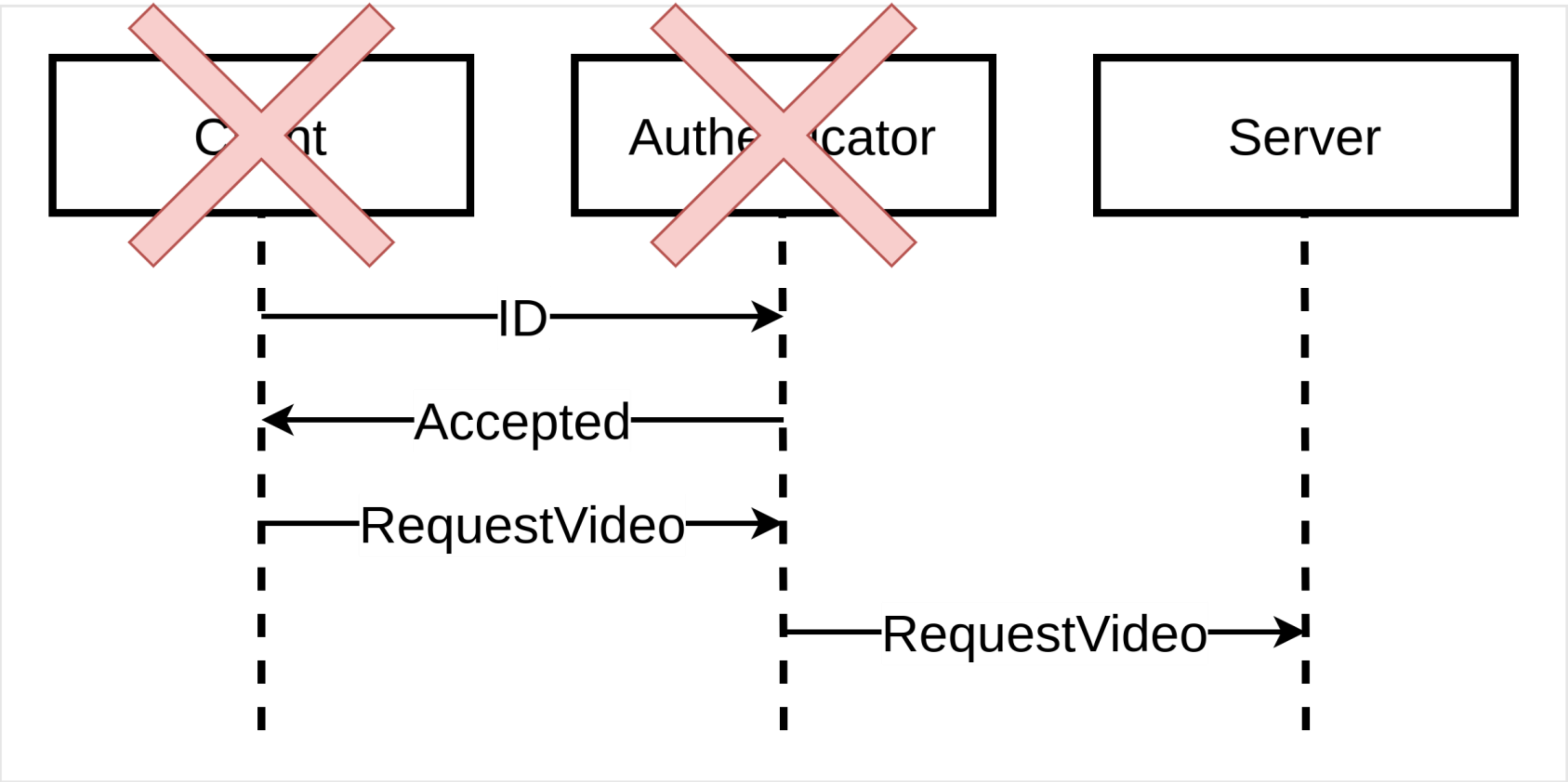
Crashes and failures in communication protocols



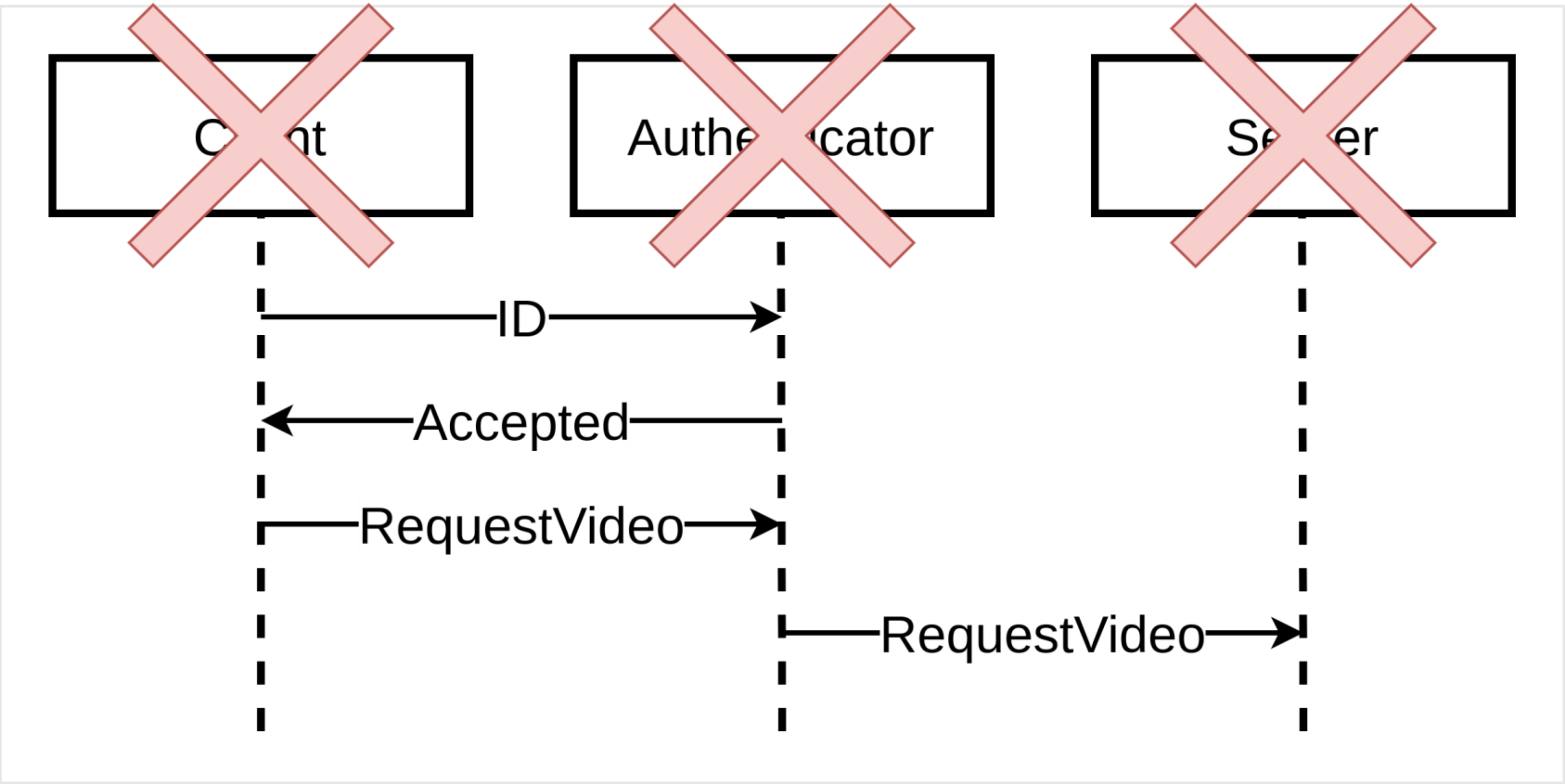
Crashes and failures in communication protocols



Crashes and failures in communication protocols



Crashes and failures in communication protocols



Automation of the process

Human

Computer

Manually writing and checking can be error-prone → Automatic checking

MultiCrusty:
a Rust implementation of AMPST

- Literature: binary types and primitives implemented in Kokke's library[1]
  - Send/Recv/End with send()/recv()/close()

1: https://doi.org/10.4204/EPTCS.304.4

MultiCrusty:
a Rust implementation of AMPST

- Literature: binary types and primitives implemented in Kokke's library[1]
  - Send/Recv/End with send()/recv()/close()

- Contributions (main ideas):
  - include those **binary types** in a structure
  - add a **stack** to provide the order of operations
  - add a **name** to distinguish each participant

1: https://doi.org/10.4204/EPTCS.304.4

Binary channels, stack and name

- **Binary channels**:
  - Transferring messages between threads
  - End to close a connection
  - Send<T, S> and Recv<T, S> where T is the type of the payload and S is the continuation

Binary channels, stack and name

- **Binary channels**:
  - Transferring messages between threads
  - End to close a connection
  - Send<T, S> and Recv<T, S> where T is the type of the payload and S is the continuation

- **Stack**: indicates which binary channel to use at each step

Binary channels, stack and name

- **Binary channels**:
  - Transferring messages between threads
  - End to close a connection
  - Send<T, S> and Recv<T, S> where T is the type of the payload and S is the continuation

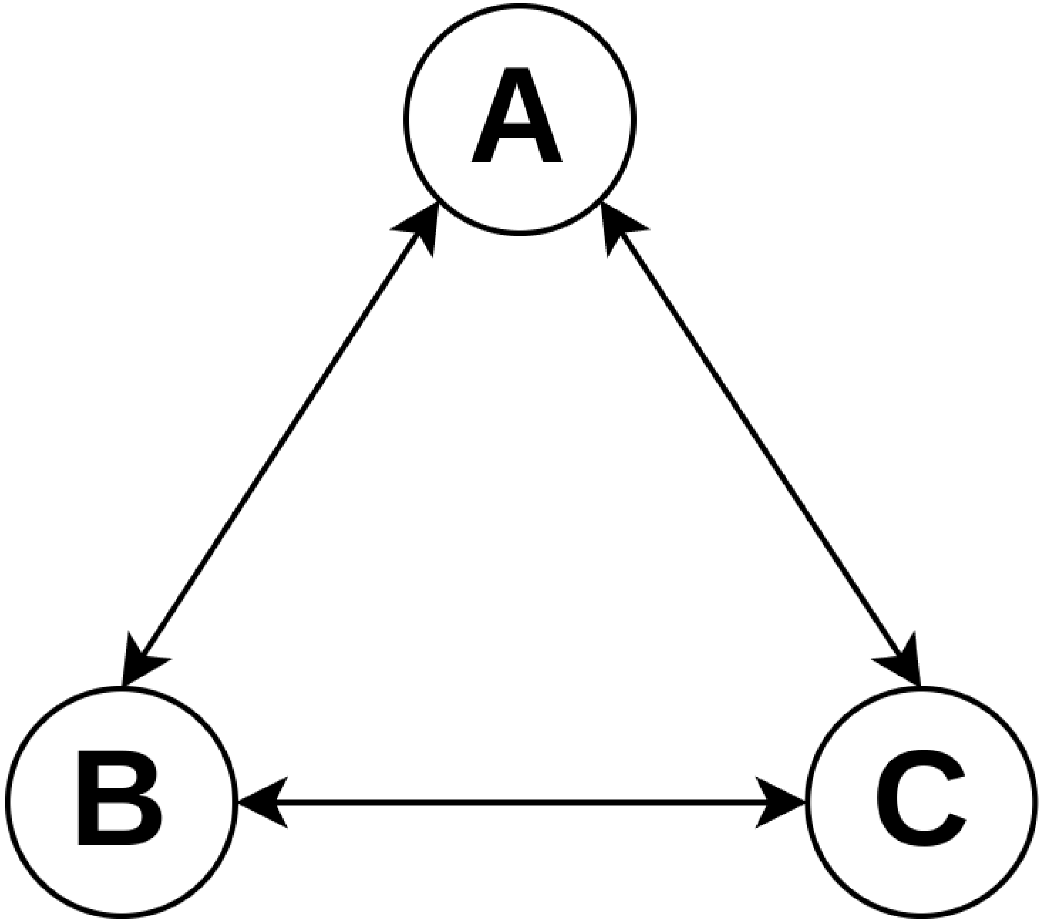
- **Stack**: indicates which binary channel to use at each step

- **Name**: indicates to which participant those previous elements belong

MeshedChannels

- Assuming a protocol with *n* participants

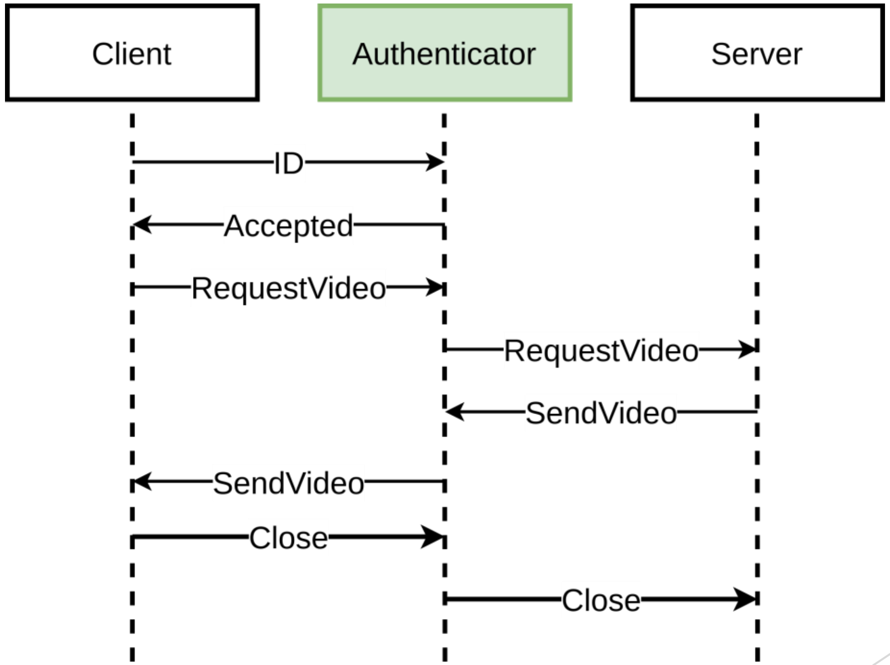
- Encapsulates *n-1* binary channels, one stack and one name to represent one participant at one step in a protocol



(Simplified) video streaming protocol

MeshedChannels<

…



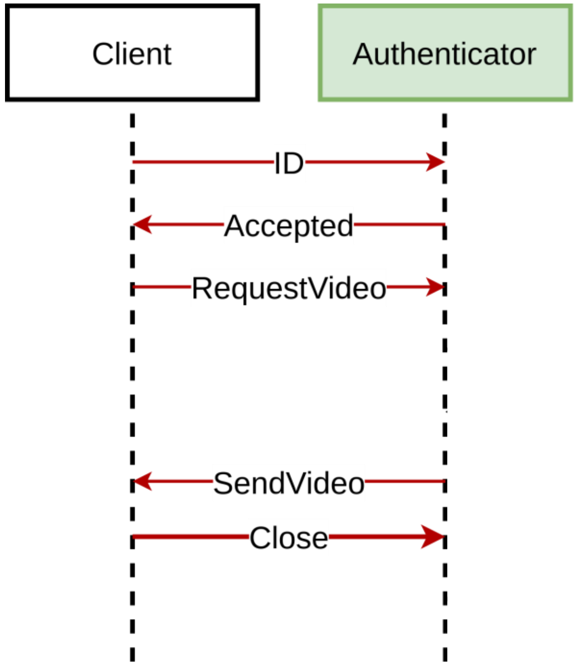
>

(Simplified) video streaming protocol

```
MeshedChannels<
 Recv<ID,
  Send<Accepted,
   Recv<RequestVideo,
    Send<SendVideo,
     Recv<Close, End>>>>,
 …


>
```

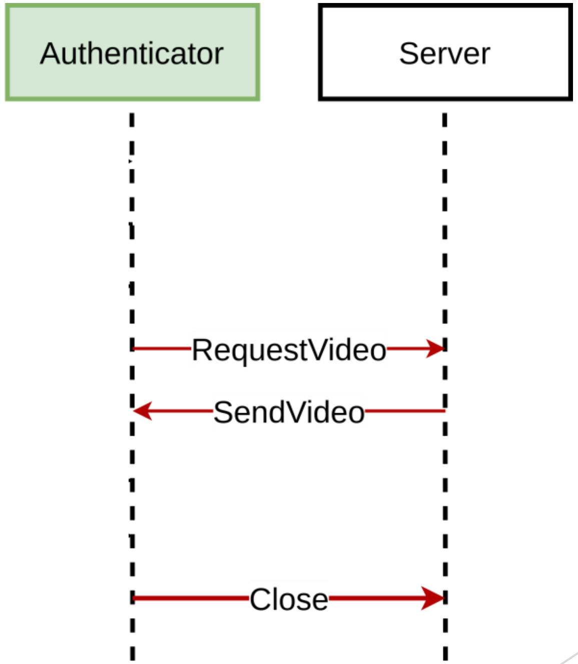
} Client/Authenticator



(Simplified) video streaming protocol

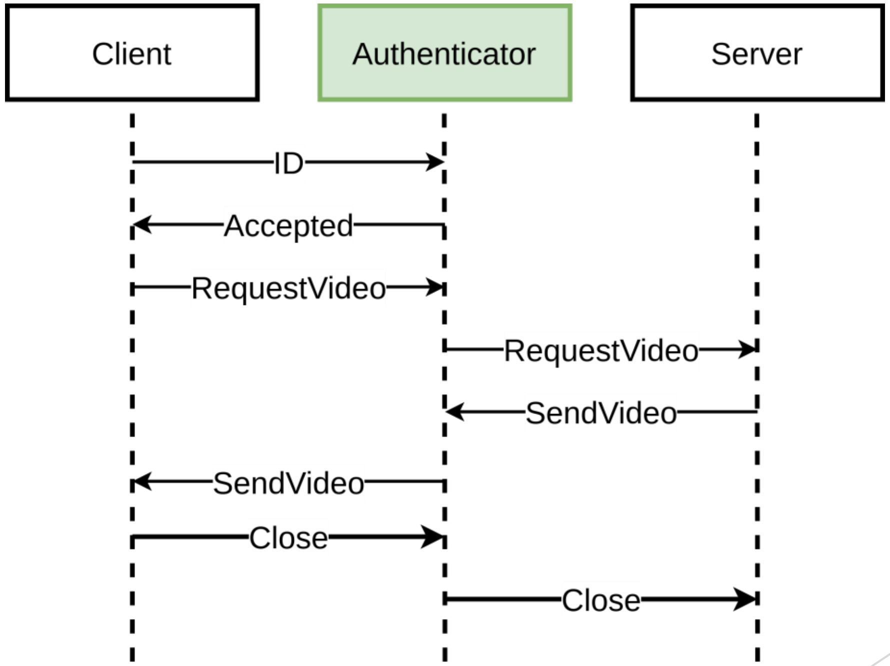
```
MeshedChannels<
 Recv<ID,
  Send<Accepted,
   Recv<RequestVideo,                 Client/Authenticator
    Send<SendVideo,
     Recv<Close, End>>>>>,
 Send<RequestVideo,
  Recv<SendVideo,                     Server/Authenticator
   Send<Close,End>>>,
 …


>
```



(Simplified) video streaming protocol

```
MeshedChannels<
 Recv<ID,
  Send<Accepted,
   Recv<RequestVideo,
    Send<SendVideo,
     Recv<Close, End>>>>,          } Client/Authenticator
 Send<RequestVideo,
  Recv<SendVideo,
   Send<Close,End>>>,              } Server/Authenticator
 Client<Client<Client<Server<Server<
  Client<Client<Server<Stop>>>>>>>>,   } Stack
 …
>
```



(Simplified) video streaming protocol

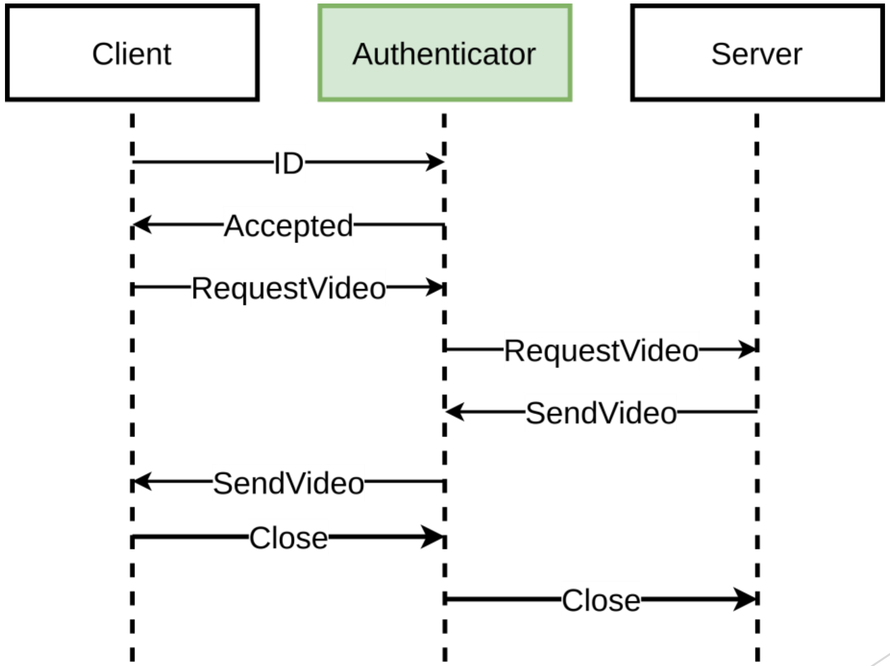
```
MeshedChannels<
 Recv<ID,
  Send<Accepted,
   Recv<RequestVideo,
    Send<SendVideo,
     Recv<Close, End>>>>,
 Send<RequestVideo,
  Recv<SendVideo,
   Send<Close,End>>>,
 Client<Client<Client<Server<Server<
  Client<Client<Server<Stop>>>>>>>>,
 Authenticator
>
```

Client/Authenticator

Server/Authenticator

Stack

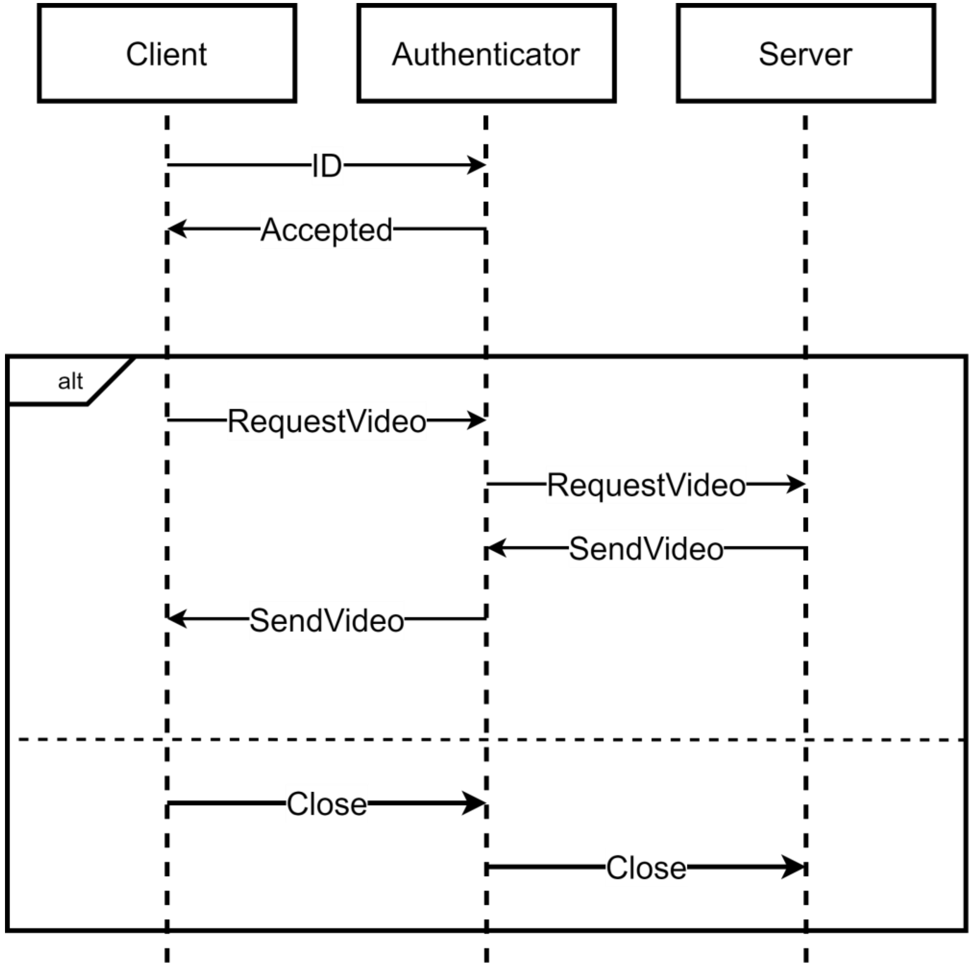
Name



Choice in MultiCrusty

enum ChoiceToAuth {
 Video(MeshedChannels<…>),
 Close(MeshedChannels<…>)
} …

Choice

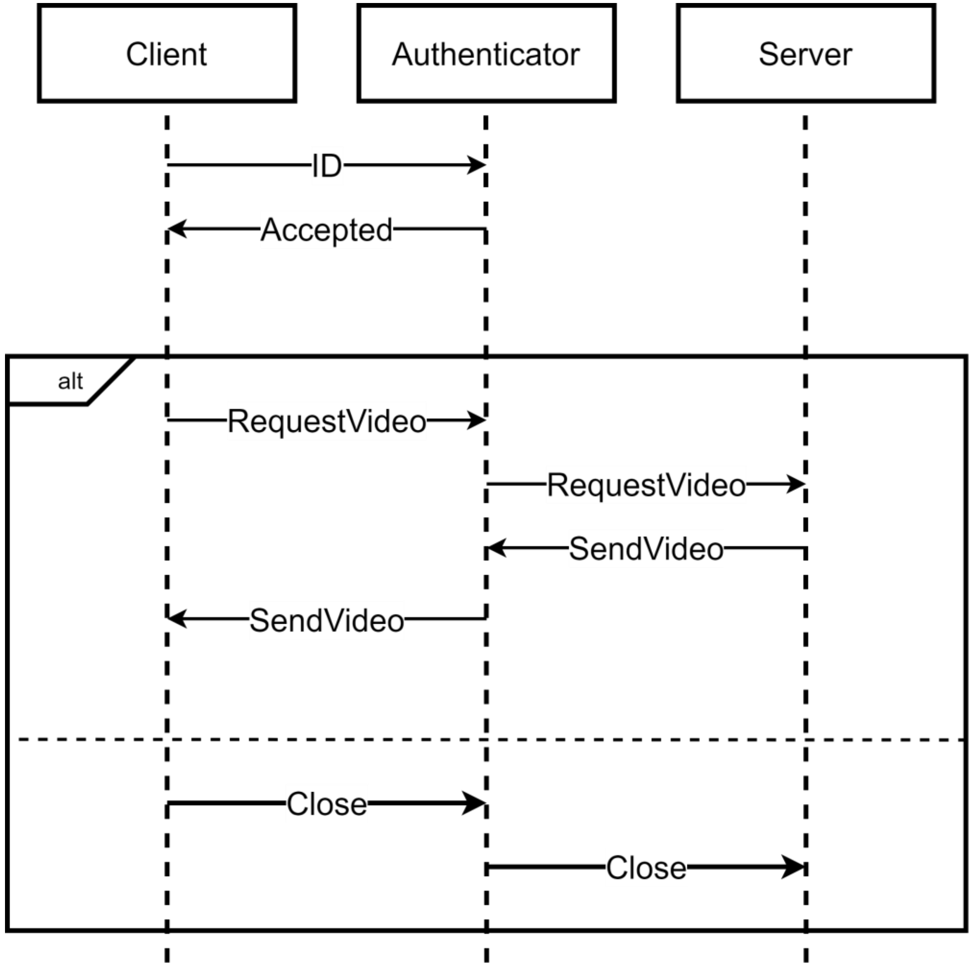


Choice in MultiCrusty

```
enum ChoiceToAuth {
 Video(MeshedChannels<…>),
 Close(MeshedChannels<…>)
}
MeshedChannels<
 Recv<ChoiceToAuth , End>, End ,
 …
> …
```

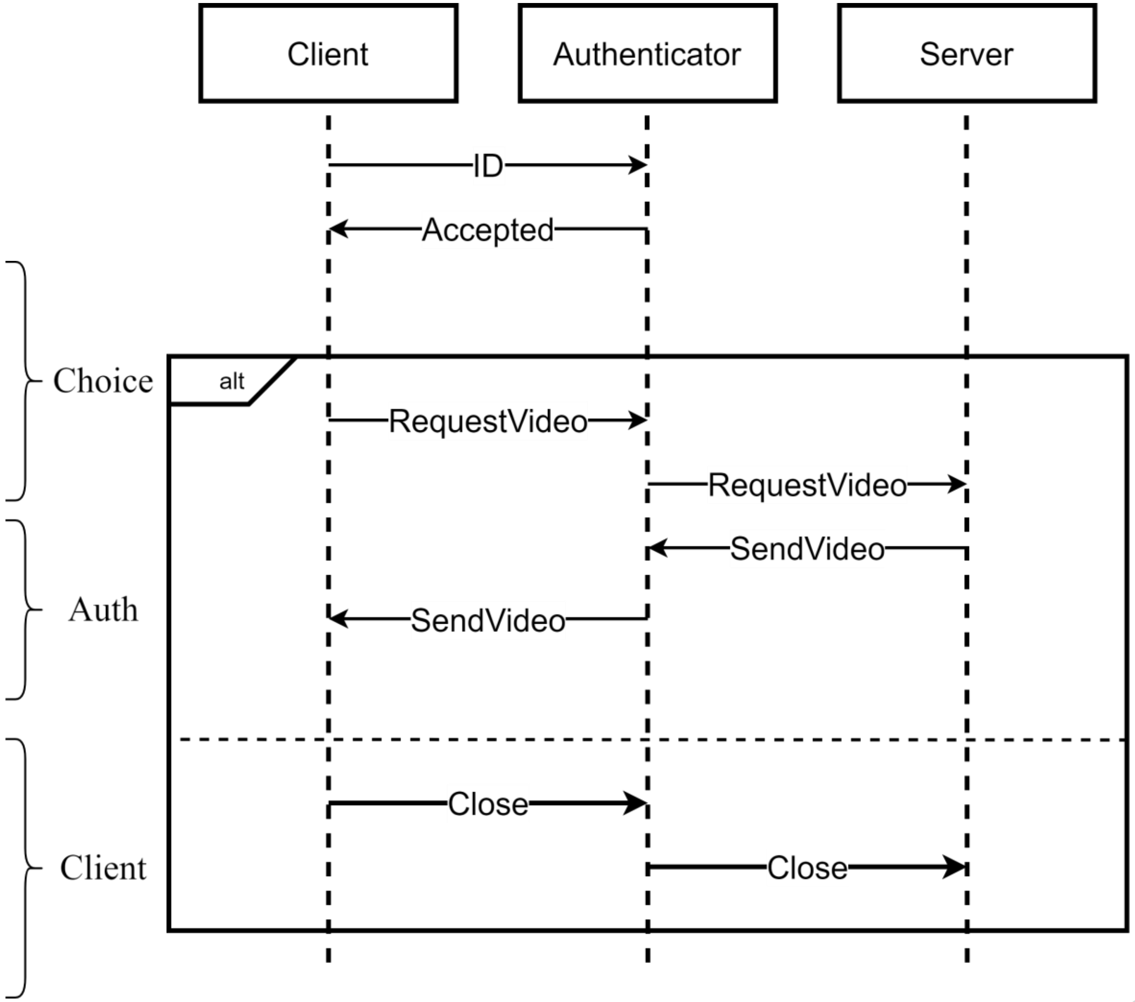
Choice

Auth



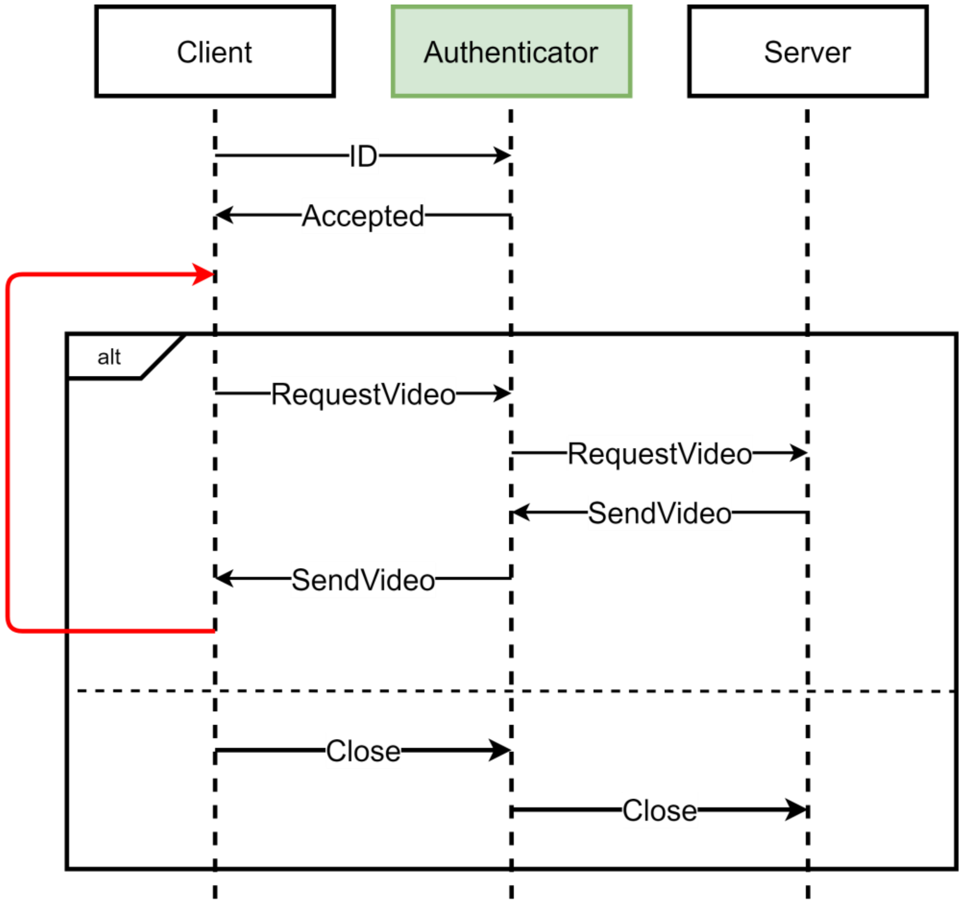
Choice in MultiCrusty

```
enum ChoiceToAuth {
 Video(MeshedChannels<…>),
 Close(MeshedChannels<…>)
}
MeshedChannels<
 Recv<ChoiceToAuth , End>, End ,
 …
>
MeshedChannels<
 Send<ChoiceToAuth , End> ,
 Send<ChoiceToServer , End> , …
>
```



Recursion in MultiCrusty

```
enum ChoiceToAuth {
 Video(MeshedChannels<
  Recv<ChoiceToAuth, End>,
  End, …
  >),
 Close(MeshedChannels<…>)
}
```



Affinity in Rust

```
fn foo( … ) -> Result<i32, Error> { … }
```

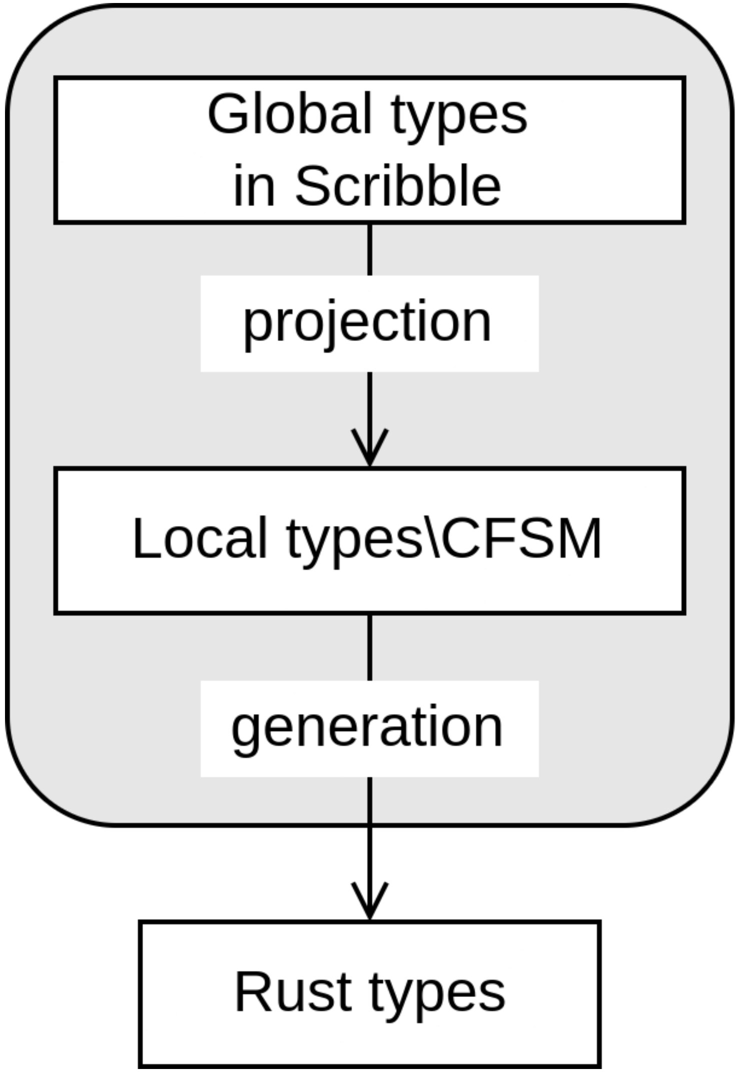
```
let bar = foo( … )?;
```

Primitives in MultiCrusty

Primitives	Description
let s = s.send(p)?;	Sends a payload **p** on channel **s**
let (p, s) = s.recv()?;	Receives a payload **p** on channel **s**
s.close()?;	Closes channel **s**
choose!(s, { enum_i :: variant_k , $\}_{i \in I}$)	Sends the chosen branch **k** to all other roles **i** in **I**
offer!(s, { { enum_i :: variant_k(e) => { … } , $\}_{k \in K}$ })	*Choice-participant* **i** expects to receive a branch **k**, among **K** branches, on channel **s**, then runs the block of code

send(p) implementation

```
/// Trait implementation to send a payload of type T to role Auth from Client
impl<S1: Session, S2: Session, R: Role, T: marker::Send>
  MeshedChannels<Send<T, S1>, S2, RoleAuth<R>, Client>
{
  pub fn send(self, payload: T) -> ReturnType<S1, S2, R> {
    …
  }
}
```

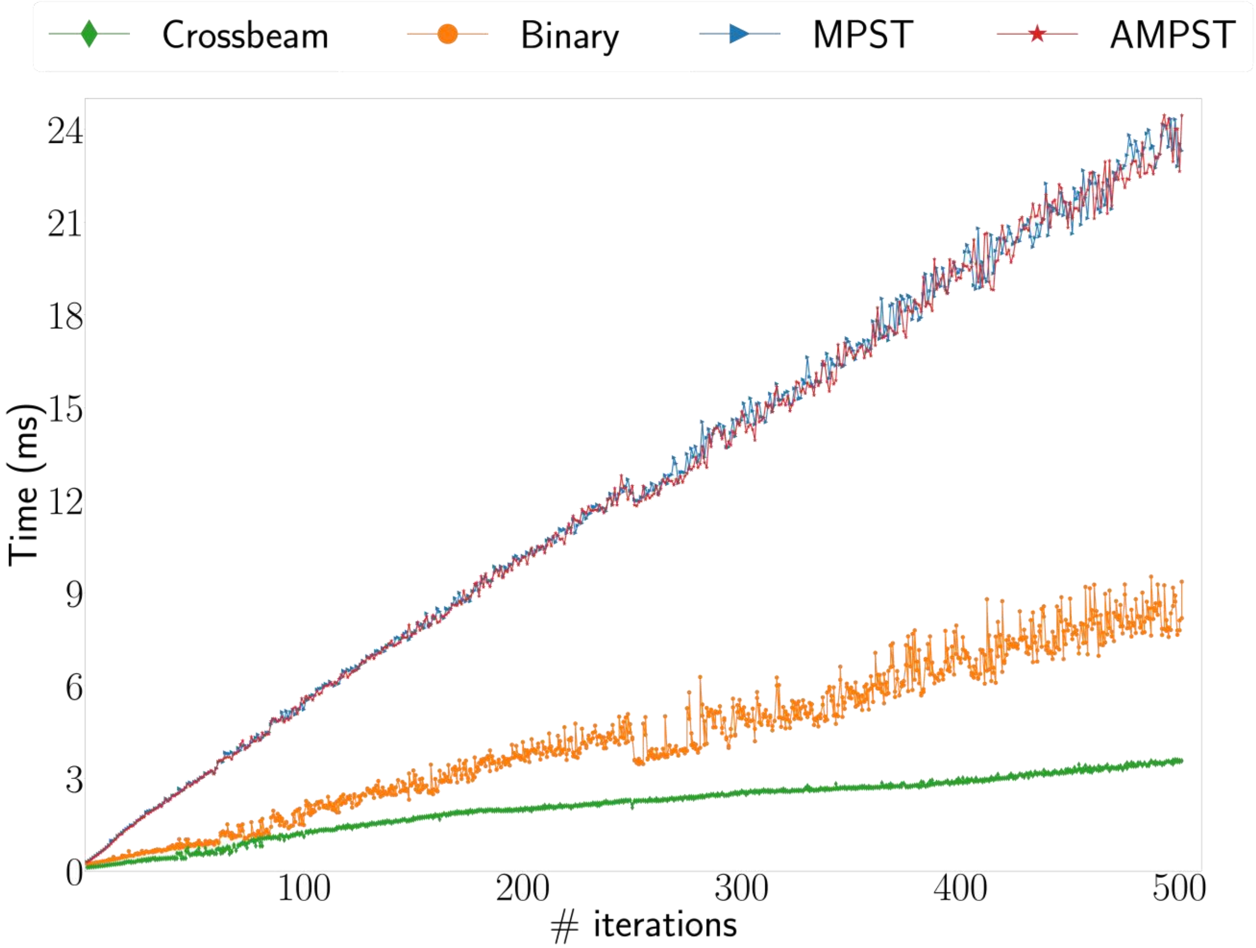


Top-down approach

Selected examples from the literature

	Compilation time (s)	Execution time (ms)	N° of lines
Video stream	37.4	11	143
Three buyers	37.1	0,568	180
Calculator	36.9	0,467	168
Travel agency	37.6	8	247
Simple voting	36.7	0,396	268
Fibonacci	36.7	9	164
oAuth2	37.5	12	276
SMTP	41.1	5	714

Benchmarks: ping-pong protocol



Summary

Theory: Affine Multiparty Session Types

- Extension of MPST to handle failures
- Introduction of **try-catch**, **cancel** and **s↯**

Implementation: MultiCrusty

- MeshedChannels
  - Binary channels, stack and name
- Can be used with Scribble
  - Top-down approach

Additional resources

- Artifact available, reusable and functional
- Arxiv full version: https://arxiv.org/abs/2204.13464
- Github repository: github.com/NicolasLagaillardie/mpst_rust_github
- Crates library: https://crates.io/crates/mpstthree

Future work

- Develop recovery strategies based on causal analysis
- Verify role-parametric session types in an affine setting
- Study polymorphic meshed channels with different delivery guarantees such as TCP and UDP

Ongoing work

- Creating the Affine Asynchronous Timed MPST framework
- Implementing the theory by extending MultiCrusty

Questions?

Appendix

Additional resources

Useful websites

- Known implementations of Session Types
  - http://www.simonjf.com/2016/05/28/session-type-implementations.html
- Nobuko Yoshida's group website
  1. http://mrg.doc.ic.ac.uk/

Comparison with other Rust implementations

<u>Ferrite</u>

- Lacks documentation and (unit) testing
- Not based on Rust logic
- Binary
- No formalism
- No top-down approach
- No cancellation termination

<u>Rumpsteak</u>

- Asynchronous
- Rely on types, not π-calculus
  - Partial proven Safety
  - Partial proven Deadlock-freedom
  - No proven Liveness
  - No cancellation termination

Forking

```
fn foo_1(s: Enpoint1) ->
    Result<(), Error> { … }
fn foo_2(s: Enpoint2) ->
    Result<(), Error> { … }
fn foo_3(s: Enpoint3) ->
    Result<(), Error> { … }

let (thread_1, thread_2, thread_3)
    = fork(foo_1, foo_2, foo_3);

thread_1.join().unwrap(); …
```

Forking

```
fn foo_1(s: Enpoint1) ->
        Result<(), Error> { … }
fn foo_2(s: Enpoint2) ->
        Result<(), Error> { … }
fn foo_3(s: Enpoint3) ->
        Result<(), Error> { … }

let (thread_1, thread_2, thread_3)
        = fork(foo_1, foo_2, foo_3);

thread_1.join().unwrap(); …
```

```
fn fork<…, F0, F1, F2> ( f0: F0, f1: F1, f2: F2 ) -> ( JoinHandle<()>, … ) where

F0: FnOnce(MeshedChannels<S0, S1, … >) -> Result<(), Error>,

F1: FnOnce(MeshedChannels<<S0 as
        Session>::Dual, S2, … >) -> Result<(),
        Error>,

F2: FnOnce(MeshedChannels<<S1 as
        Session>::Dual, <S2 as Session>::Dual,
        … > ) -> Result<(), Error>,

… { … }
```

Forking

```
fn foo_1(s: Enpoint1) ->
    Result<(), Error> { … }
fn foo_2(s: Enpoint2) ->
    Result<(), Error> { … }
fn foo_3(s: Enpoint3) ->
    Result<(), Error> { … }

let (thread_1, thread_2, thread_3)
    = fork(foo_1, foo_2, foo_3);

thread_1.join().unwrap(); …
```

```
fn fork<…, F0, F1, F2> ( f0: F0, f1: F1, f2: F2 ) -> ( JoinHandle<()>, … ) where

F0: FnOnce(MeshedChannels<S0, S1, … >) -> Result<(), Error>,

F1: FnOnce(MeshedChannels<<S0 as
    Session>::Dual, S2, … >) -> Result<(),
    Error>,

F2: FnOnce(MeshedChannels<<S1 as
    Session>::Dual, <S2 as Session>::Dual,
    … > ) -> Result<(), Error>,

… { … }
```

Session

```
/// Trait for binary session types. Provides duality.
/// marker::Sized -> Types with a constant size known at compile time.
/// marker::Send -> Types that can be transferred across thread boundaries.
trait Session: marker::Sized + marker::Send {
    /// The session type dual to `Self`.
    type Dual: Session<Dual = Self>;
… }
```

Send

```
impl<T: marker::Send, S: Session> Session
    for Send<T, S> {
```

Send

```
impl<T: marker::Send, S: Session> Session
    for Send<T, S> {

type Dual = Recv<T, S::Dual>;
```

Send

```
impl<T: marker::Send, S: Session> Session
    for Send<T, S> {
type Dual = Recv<T, S::Dual>;

fn new() -> (Self, Self::Dual) {




… }
```

Send

```
impl<T: marker::Send, S: Session> Session
    for Send<T, S> {
type Dual = Recv<T, S::Dual>;

fn new() -> (Self, Self::Dual) {
let (sender, receiver) = bounded::<(T,
    S::Dual)>(1);
( Send { channel: sender },
Recv { channel: receiver } )
}
… }
```

Send

```
impl<T: marker::Send, S: Session> Session
    for Send<T, S> {
type Dual = Recv<T, S::Dual>;

fn new() -> (Self, Self::Dual) {
let (sender, receiver) = bounded::<(T,
    S::Dual)>(1);
( Send { channel: sender },
Recv { channel: receiver } )
}
… }
```

```
fn send<T, S>(x: T, s: Send<T, S>) -> S
{
let (here, there) = S::new();
s.channel.send((x, there)).unwrap_or(());
here
}
```

π-calculus

▶ **Definition 3.1.** The **affine multiparty session π-calculus** (AMPST) is defined as follows:

$$
\begin{array}{rcll}
c, d & ::= & x \mid s[\mathtt{p}] \qquad \dagger ::= \emptyset \mid ? & \text{(variable, channel with role } \mathtt{p}\text{, error, flag)} \\
P, Q & ::= & \mathbf{0} \mid P \mid Q \mid (\nu s)\, P & \text{(inaction, composition, restriction)} \\
 & & ?\, c[\mathtt{q}] \oplus \mathtt{m}\langle d \rangle . P \mid ?\, c[\mathtt{q}] \sum_{i \in I} \mathtt{m}_i(x_i) . P_i & \text{(affine selection, branching } I \neq \emptyset\text{)} \\
 & & c[\mathtt{q}] \oplus \mathtt{m}\langle d \rangle . P \mid c[\mathtt{q}] \sum_{i \in I} \mathtt{m}_i(x_i) . P_i & \text{(selection, branching } I \neq \emptyset\text{)} \\
 & & \mathsf{def}\ D\ \mathsf{in}\ P \mid X\langle \widetilde{c} \rangle & \text{(process definition, process call)} \\
 & & \mathsf{try}\ P\ \mathsf{catch}\ Q \mid \mathsf{cancel}(c).P \mid s\lightning & \text{(catch, cancel, kill)} \\
D & ::= & X(\widetilde{x}) = P & \text{(declaration of process variable } X\text{)}
\end{array}
$$

Reduction rules

[R-Com] $\mathbb{E}_1[\dagger\, s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i] \mid \mathbb{E}_2[\dagger\, s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.Q] \;\rightarrow\; P_k\{^{s'[\mathtt{r}]}/_{x_k}\} \mid Q \quad \text{if } k\in I$

[C-?Sel] $?\, s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{m}\langle s'[\mathtt{r}]\rangle.P \rightarrow s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{m}\langle s'[\mathtt{r}]\rangle.P \mid s\lightning$

[T?Sel] $\mathbf{try}\ ?\, s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{m}\langle s'[\mathtt{r}]\rangle.P\ \mathbf{catch}\ Q \rightarrow Q \mid s\lightning$

[C-Sel] $s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{m}\langle s'[\mathtt{r}]\rangle.P \mid s\lightning \rightarrow P \mid s\lightning \mid s'\lightning$

[C-?Br] $?\, s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i \rightarrow s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i \mid s\lightning$

[T?Br] $\mathbf{try}\ ?\, s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i\ \mathbf{catch}\ Q \rightarrow Q \mid s\lightning$

[C-Br] $s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i \mid s\lightning \rightarrow (\nu s')\,(P_k\{^{s'[\mathtt{r}]}/_{x_k}\} \mid s'\lightning) \mid s\lightning \quad s'\notin \mathrm{fc}(P_k),\, k\in I$

[R-Can] $\mathbb{E}[\mathbf{cancel}(s[\mathtt{p}]).Q] \rightarrow s\lightning \mid Q$ [C-Cat] $\mathbf{try}\ P\ \mathbf{catch}\ Q \mid s\lightning \rightarrow Q \mid s\lightning \quad \exists \mathtt{r}.\ s[\mathtt{r}] = \mathrm{sbj}(P)$

[R-Def] $\mathbf{def}\ X(x_1,\dots,x_n) = P\ \mathbf{in}\ (X\langle s_1[\mathtt{p}_1],\dots,s_n[\mathtt{p}_n]\rangle \mid Q)$

$\rightarrow \mathbf{def}\ X(x_1,..,x_n) = P\ \mathbf{in}\ (P\{^{s_1[\mathtt{p}_1]}/_{x_1}\}\cdot\cdot\{^{s_n[\mathtt{p}_n]}/_{x_n}\} \mid Q)$

[R-Ctx] $P \rightarrow P'$ implies $\mathbb{C}[P] \rightarrow \mathbb{C}[P']$ [R-Struct] $P \equiv P' \rightarrow Q' \equiv Q$ implies $P \rightarrow Q$

Syntax of types

▶ **Definition 3.8** (Global types). The syntax of a **global type** G is:

$$G ::= \mathtt{p}{\to}\mathtt{q}\colon \{\mathtt{m_i}(S_i).G_i\}_{i\in I} \;\big|\; \mu\mathbf{t}.G \;\big|\; \mathbf{t} \;\big|\; \mathbf{end} \qquad \text{with } \mathtt{p}\neq\mathtt{q},\; I\neq\emptyset,\; \text{and } \forall i\in I : \mathrm{fv}(S_i)=\emptyset$$

The syntax of **local types** is:

$$S,T ::= \mathtt{p}\&_{i\in I}\mathtt{m}_i(S_i).S_i' \;\big|\; \mathtt{p}\oplus_{i\in I}\mathtt{m}_i(S_i).S_i' \;\big|\; \mathbf{end} \;\big|\; \mu\mathbf{t}.S \;\big|\; \mathbf{t} \quad \text{with } I\neq\emptyset, \text{ and } \mathtt{m}_i \text{ pairwise distinct.}$$

$$\frac{\Theta(X) = S_1, \ldots, S_n}{\Theta \vdash X{:}S_1, \ldots, S_n}\;[\text{T-}X] \qquad \frac{S \leqslant S'}{c{:}S \vdash c{:}S'}\;[\text{T-sub}] \qquad \frac{\forall i \in 1..n \quad c_i{:}S_i \vdash c_i{:}\mathbf{end}}{\text{end}(c_1{:}S_1, \ldots, c_n{:}S_n)}\;[\text{T-end}] \qquad \frac{\text{end}(\Gamma)}{\Theta \cdot \Gamma \vdash \mathbf{0}}\;[\text{T-}\mathbf{0}]$$

$$\frac{\Gamma_1 \vdash c{:}\mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S_i' \quad \forall i \in I \quad \Theta \cdot \Gamma, y_i{:}S_i, c{:}S_i' \vdash P_i}{\Theta \cdot \Gamma, \Gamma_1 \vdash \dagger\, c[\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(y_i).P_i}\;[\text{T-}\&] \qquad \frac{\Theta \cdot \Gamma_1 \vdash P_1 \qquad \Theta \cdot \Gamma_2 \vdash P_2}{\Theta \cdot \Gamma_1, \Gamma_2 \vdash P_1 \mid P_2}\;[\text{T-}|]$$

$$\frac{\Gamma_1 \vdash c{:}\mathtt{q}\oplus\mathtt{m}(S).S' \quad \Gamma_2 \vdash c'{:}S \quad \Theta \cdot \Gamma, c{:}S' \vdash P}{\Theta \cdot \Gamma, \Gamma_1, \Gamma_2 \vdash \dagger\, c[\mathtt{q}]\oplus\mathtt{m}\langle c'\rangle.P}\;[\text{T-}\oplus] \qquad \frac{\Theta \cdot \Gamma \vdash P \qquad \text{sbj}(P) = \{c\} \qquad \Theta \cdot \Gamma \vdash Q}{\Theta \cdot \Gamma \vdash \textbf{try}\ P\ \textbf{catch}\ Q}\;[\text{ T-try }]$$

$$\frac{\text{end}(\Gamma) \qquad 0 \leq n}{\Theta \cdot \Gamma, s[\mathtt{p}_1]{:}S_1, \ldots, s[\mathtt{p}_n]{:}S_n \vdash s\lightning}\;[\text{ T-kill }] \qquad \frac{\Theta \cdot \Gamma \vdash Q}{\Theta \cdot \Gamma, c{:}S \vdash \textbf{cancel}(c).Q}\;[\text{ T-cancel }]$$

$$\frac{\Theta, X{:}S_1, \ldots, S_n \cdot x_1{:}S_1, \ldots, x_n{:}S_n \vdash P \qquad \Theta, X{:}S_1, \ldots, S_n \cdot \Gamma \vdash Q}{\Theta \cdot \Gamma \vdash \textbf{def}\ X(x_1{:}S_1, \ldots, x_n{:}S_n) = P\ \textbf{in}\ Q}\;[\text{T-def}]$$

$$\frac{\Theta \vdash X{:}S_1, \ldots, S_n \quad \text{end}(\Gamma_0) \quad \forall i \in 1..n \quad \Gamma_i \vdash c_i{:}S_i}{\Theta \cdot \Gamma_0, \Gamma_1, \ldots, \Gamma_n \vdash X\langle c_1, \ldots, c_n\rangle}\;[\text{T-call}]$$

$$\frac{\Gamma' = \{s[\mathtt{p}]{:}S_{\mathtt{p}}\}_{\mathtt{p}\in I} \quad s \notin \Gamma \quad \text{safe}(\Gamma') \quad \Theta \cdot \Gamma, \Gamma' \vdash P}{\Theta \cdot \Gamma \vdash (\nu s{:}\Gamma')\,P}\;[\text{T-}\nu]$$

$$\frac{\Gamma' = \{s[\mathtt{p}]{:}G{\restriction}\mathtt{p}\}_{\mathtt{p}\in\text{roles}(G)}\ \text{or}\ \text{end}(\Gamma') \quad s \notin \Gamma \quad \Theta \cdot \Gamma, \Gamma' \vdash P}{\Theta \cdot \Gamma \vdash (\nu s{:}\Gamma')\,P}\;[\text{ T-init }]$$