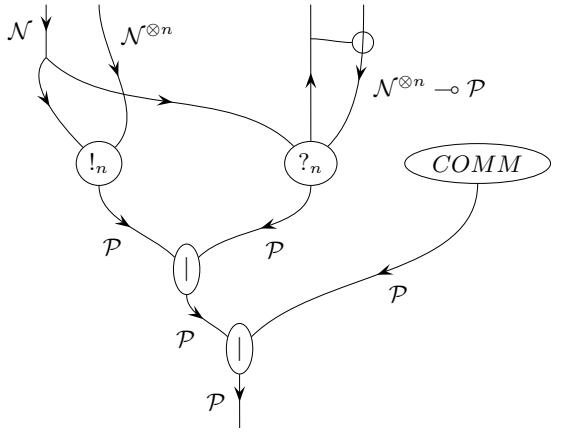
Casper Formalized Pt I

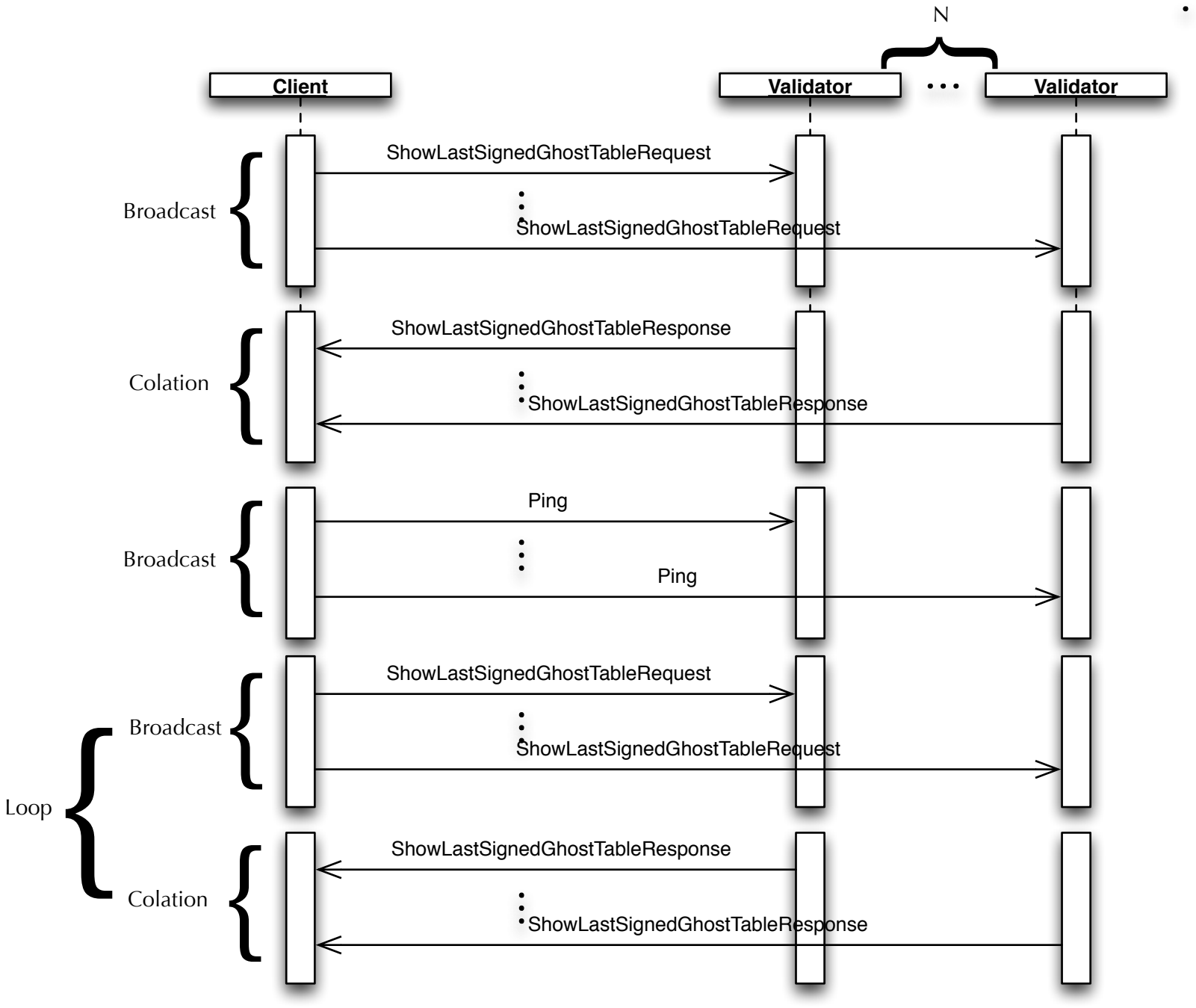
Applying π-calculus

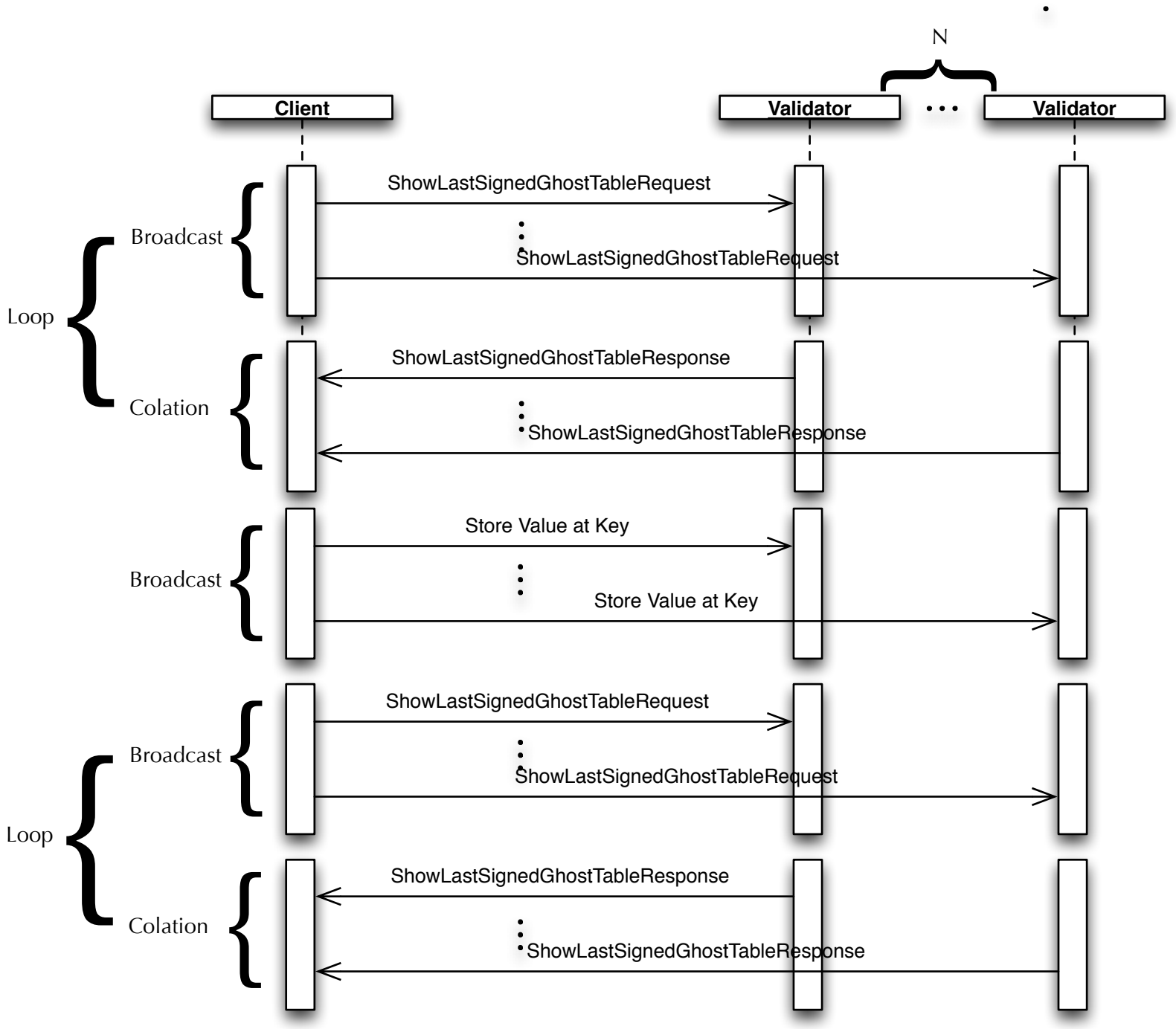
to modeling the Casper protocol



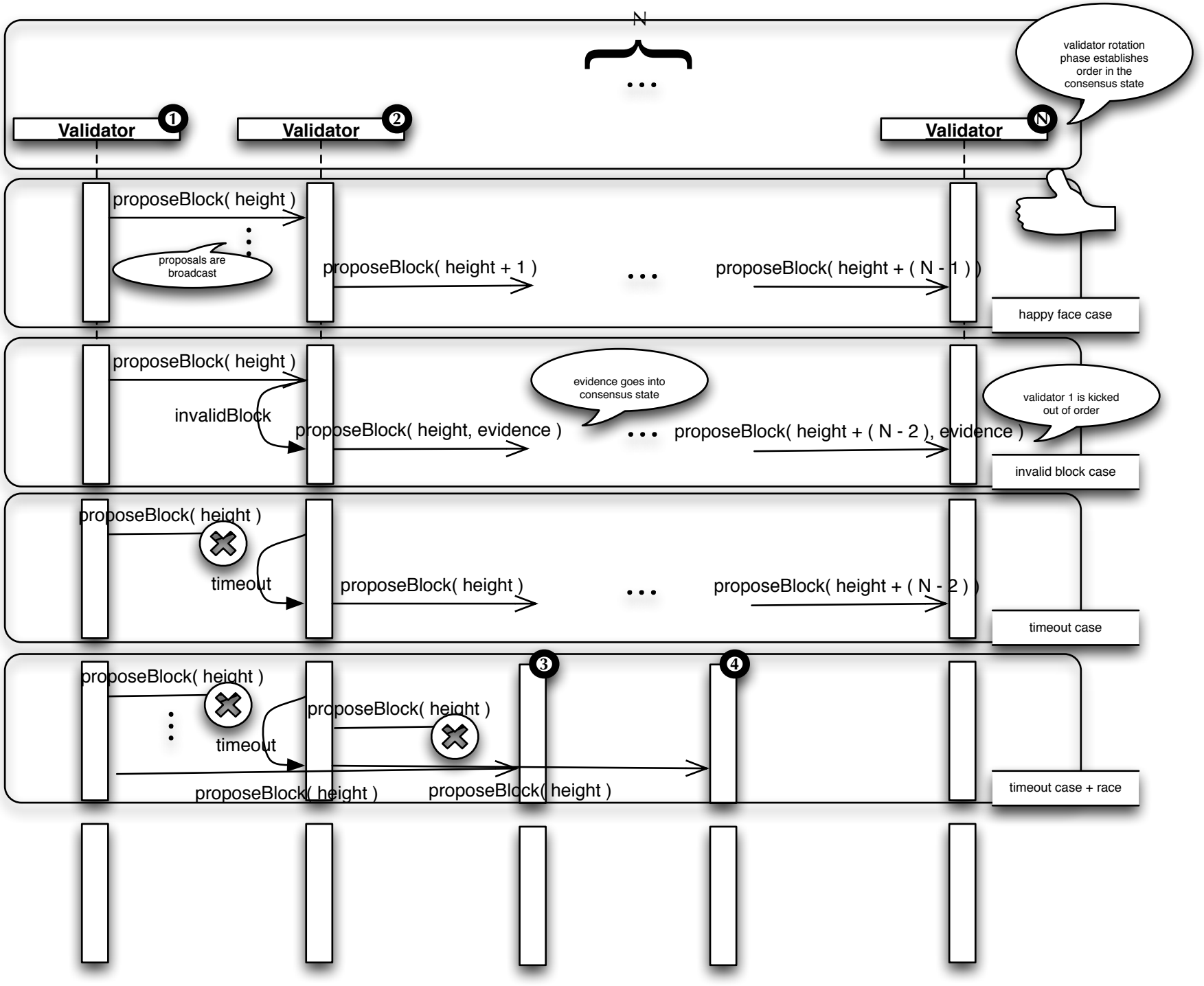
Vlad Zamfir, L.G. Meredith

Client <-> Validator interactions

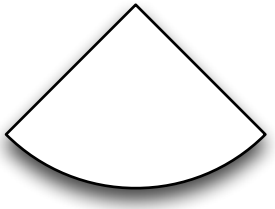

N
Client
Validator
Validator
ShowLastSignedGhostTableRequest
ShowLastSignedGhostTableRequest
Broadcast
ShowLastSignedGhostTableResponse
ShowLastSignedGhostTableResponse
Colation
Ping
Ping
Broadcast
ShowLastSignedGhostTableRequest
ShowLastSignedGhostTableRequest
Broadcast
Loop
ShowLastSignedGhostTableResponse
ShowLastSignedGhostTableResponse
Colation


N
Client
Validator
Validator
ShowLastSignedGhostTableRequest
ShowLastSignedGhostTableRequest
Broadcast
Loop
ShowLastSignedGhostTableResponse
ShowLastSignedGhostTableResponse
Colation
Store Value at Key
Store Value at Key
Broadcast
ShowLastSignedGhostTableRequest
ShowLastSignedGhostTableRequest
Broadcast
Loop
ShowLastSignedGhostTableResponse
ShowLastSignedGhostTableResponse
Colation

Consensus: Validator <-> Validator interactions


N
validator rotation phase establishes order in the consensus state
Validator 1
Validator 2
Validator N
proposeBlock(height)
proposals are broadcast
proposeBlock(height + 1)
proposeBlock(height + (N - 1))
happy face case
proposeBlock(height)
invalidBlock
evidence goes into consensus state
proposeBlock(height, evidence)
proposeBlock(height + (N - 2), evidence)
validator 1 is kicked out of order
invalid block case
proposeBlock(height)
timeout
proposeBlock(height)
proposeBlock(height + (N - 2))
timeout case
proposeBlock(height)
timeout
proposeBlock(height)
3
4
proposeBlock(height)
proposeBlock(height)
timeout case + race

How to turn interaction diagrams into π-calculus specs



programming model

allow Abed to check claim C

for(

?

e1 <- chan?(C2V)(allowCheck( )));

Verify Troy's claim C

?

e2 <- chan?(R2V)(verify( ))

) {

confirm Troy's claim C

!

chan!(V2R)(confirm( ))

}

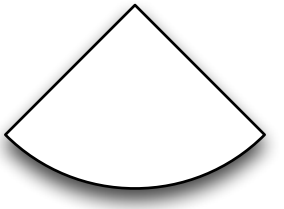
verifier behavior

Claimant

Verifier

Relying party

Just walk this line to calculate the specification of the verifier's behavior and derive code



programming model

allow Abed to check claim C

!

chan!(C2V)(allowCheck( ) , ...)

claim: C & Dean will verify

!

chan!(C2V)(claim( ), ...)

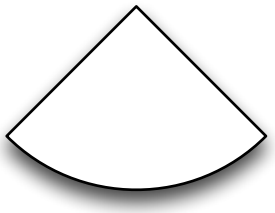
Claimant

Verifier

Relying party

claimant behavior

and this line to calculate the specification of the claimant's behavior and derive code

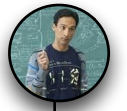


programming model

Claimant

Verifier

Relying party



```
for(
```

claim: C & Dean will verify

```
e1 <- chan?( C2R )( claim( ) ) )
) {
```

Verify Troy's claim C

```
chan!( C2R )( verify( ), ... )
for(
```

confirm Troy's claim C

```
e1 <- chan?( C2R )( trudat( ) ) )
) {
    ...
}
}
```

relying party behavior

and this line to calculate the specification of the relying party's behavior and derive code

How to turn π-calculus specs into scala code

P,Q ::= 0	`{ }`
a![v1, …, vn]	`[| a |](m) ![ [| v1 |](m), …, [| vn |](m) ]`
a?(x1, …, xn)P	`for( [ x1, …, xn ] <- [| a |](m) ){ [| P |](m)( x1, …, xn ) }`
P | Q	`spawn{ [| P |](m) };spawn{ [| Q |](m) }`
(new a)P	`{ val q = new Queue(); [| P |](m[ a <- q ]) }`
(def X(x1, …, xn) = P)[v1, …, vn]	`object X { def apply( x1, …, xn ) = { [| P |](m)( x1, …, xn ) } }`
X[v1, …, vn]	`X( [| v1 |](m), …, [| vn |](m) )`

`[| - |](-) : ( π-calculus, Map[Symbol,Queue] ) -> Scala`