

# Resource-Aware Session Types for Digital Contracts

---

**Ankush Das**

**Stephanie Balzer**

**Jan Hoffmann**

**Frank Pfenning**

**Ishani Santurkar**

**Carnegie Mellon University**

**Session 2, CSF 2021**





## Digital Contracts



## Digital Contracts



### *Challenges:*

- 1. contract protocols are violated*
- 2. asset duplication/deletion*
- 3. automatic inference of execution cost*



## Digital Contracts

## Resource-Aware Session Types



### Challenges:

- 1. contract protocols are violated*
- 2. asset duplication/deletion*
- 3. automatic inference of execution cost*



## Digital Contracts



### Challenges:

- 1. contract protocols are violated*
- 2. asset duplication/deletion*
- 3. automatic inference of execution cost*



### Solutions:

- 1. session types enforce protocols*
- 2. linear types track assets*
- 3. resource-aware types infer execution cost*

## Resource-Aware Session Types



## Digital Contracts



### Challenges:

- 1. contract protocols are violated*
- 2. asset duplication/deletion*
- 3. automatic inference of execution cost*

## Resource-Aware Session Types



### Solutions:

- 1. session types enforce protocols*
- 2. linear types track assets*
- 3. resource-aware types infer execution cost*

# What are Digital Contracts?

---

*Programs implementing a digital (legal or financial) transaction*

# What are Digital Contracts?

*Programs implementing a digital (legal or financial) transaction*



## Online Transactions



# What are Digital Contracts?

*Programs implementing a digital (legal or financial) transaction*



**Online Transactions**



**Wealth Management Applications**

# What are Digital Contracts?

*Programs implementing a digital (legal or financial) transaction*



**Online Transactions**



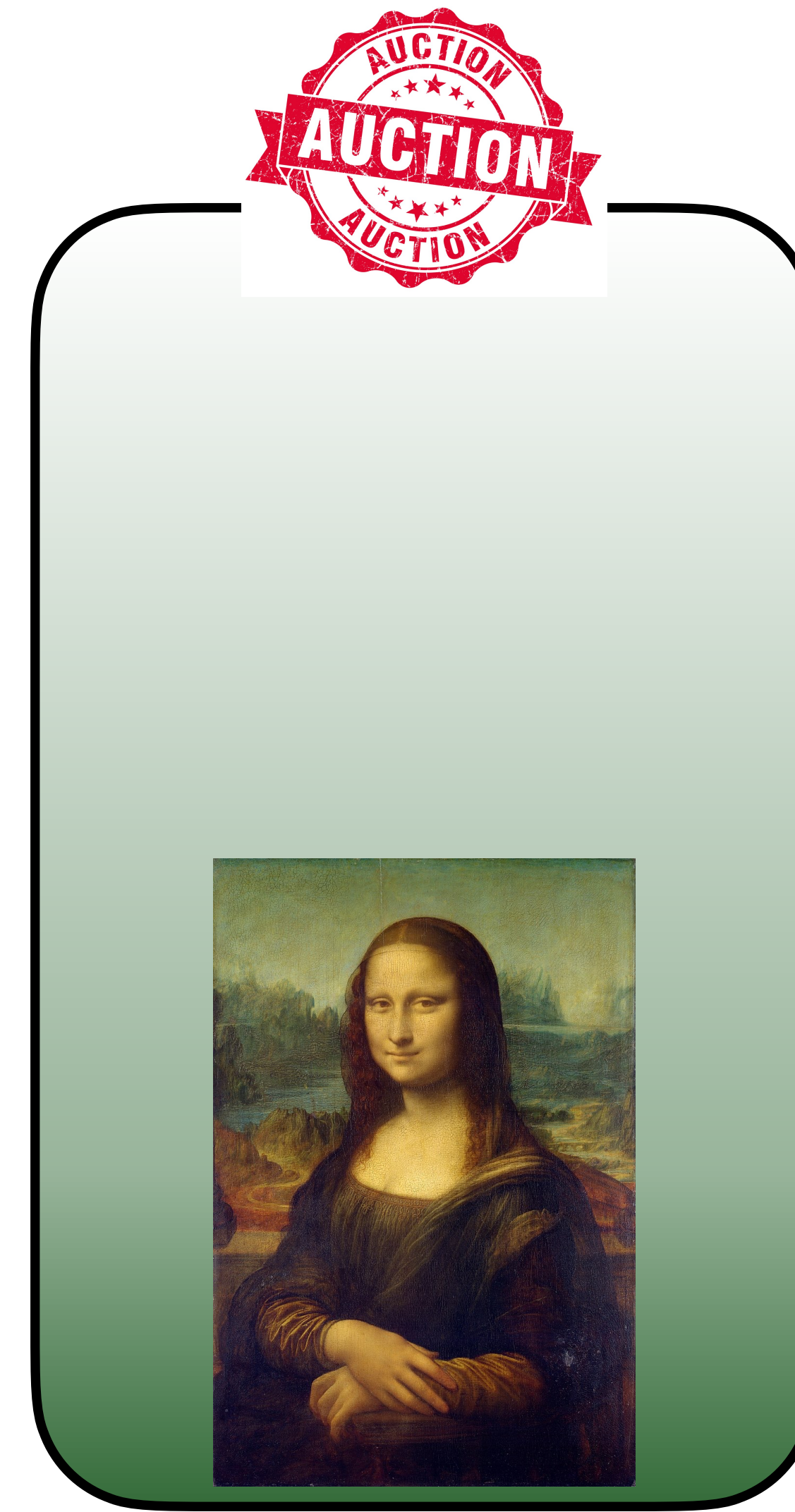
**Wealth Management Applications**

*recently popularized  
in the form of*



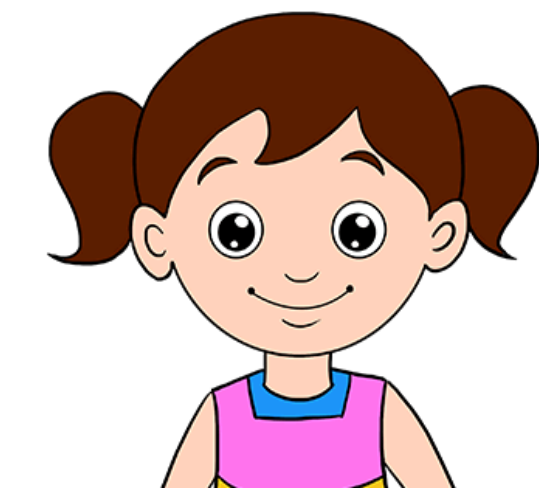
**Blockchains and Cryptocurrencies**

# Example: Auction Contract



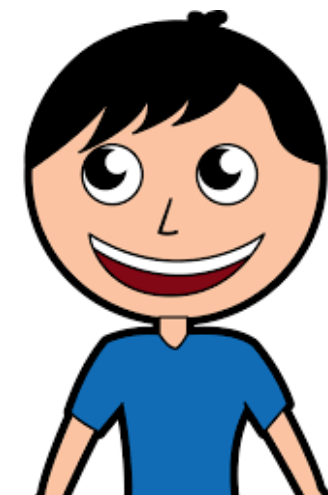
**status: running**

# Example: Auction Contract



Bidder 1

Bid 1



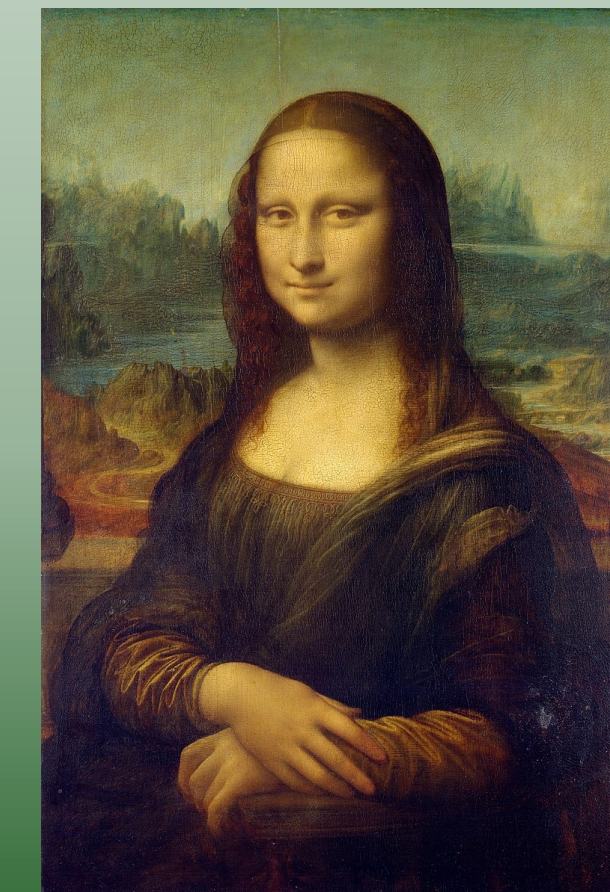
Bidder 2

Bid 2



Bidder 3

Bid 3

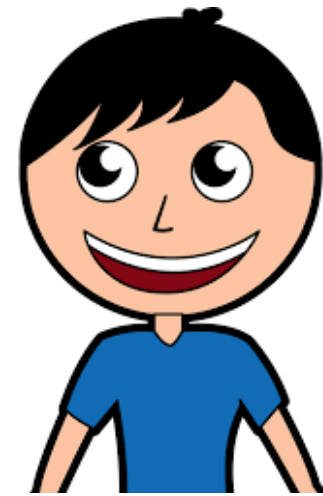


status: running

# Example: Auction Contract



**Bidder 1**



**Bidder 2**



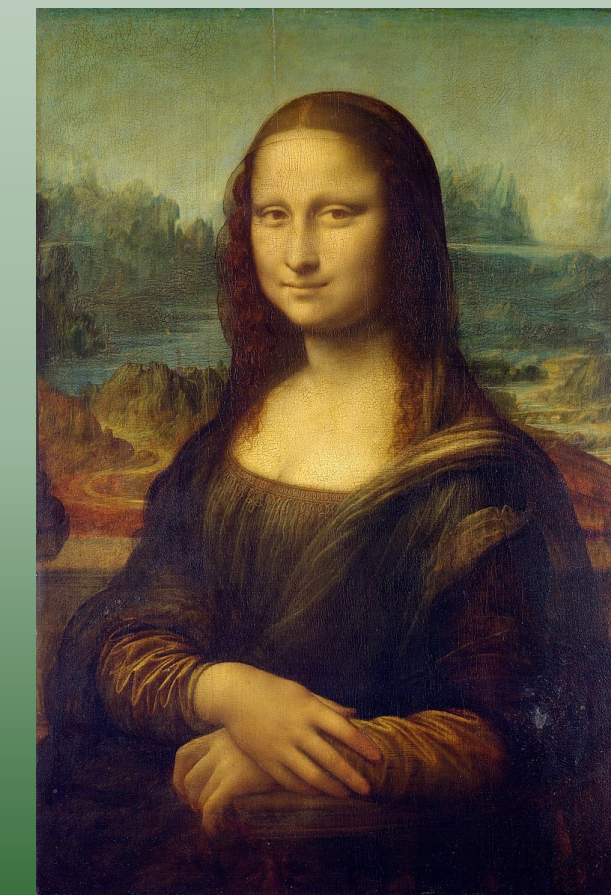
**Bidder 3**



**Bid 1**

**Bid 2**

**Bid 3**

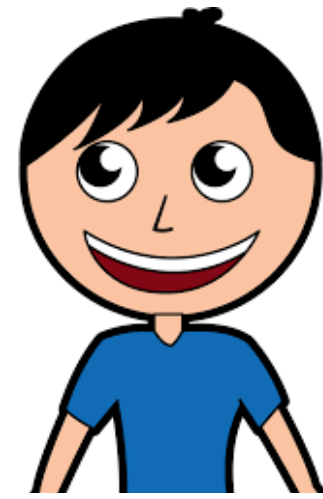


**status: running**

# Example: Auction Contract



**Bidder 1**



**Bidder 2**



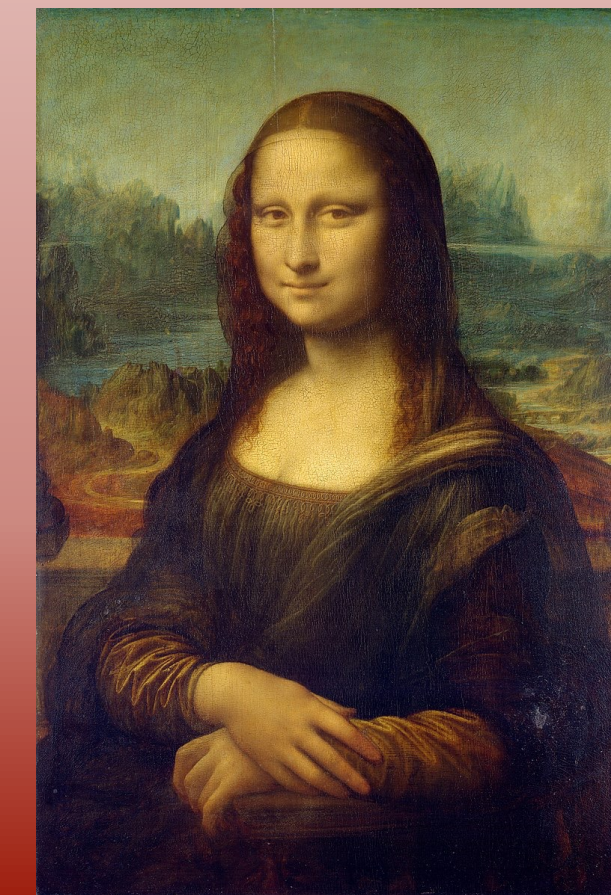
**Bidder 3**



**Bid 1**

**Bid 2**

**Bid 3**

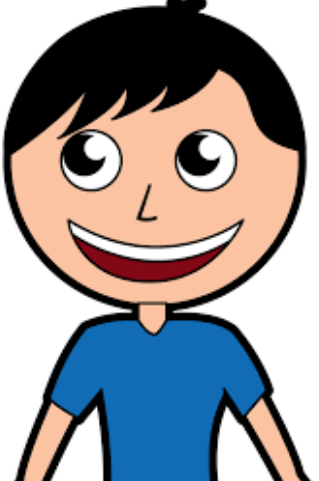


**status: ended**

# Example: Auction Contract



**Bidder 1**



**Bidder 2**



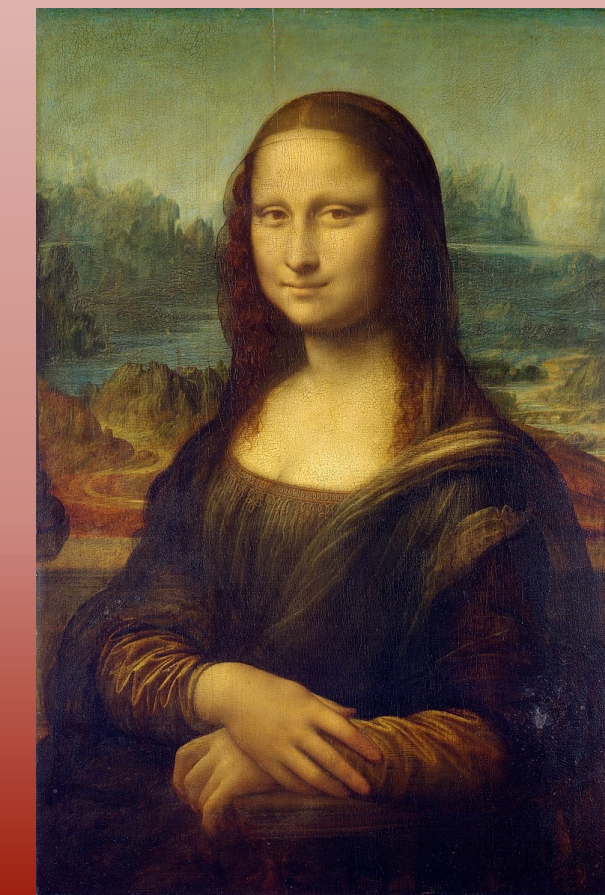
**Bidder 3**



**Bid 1**

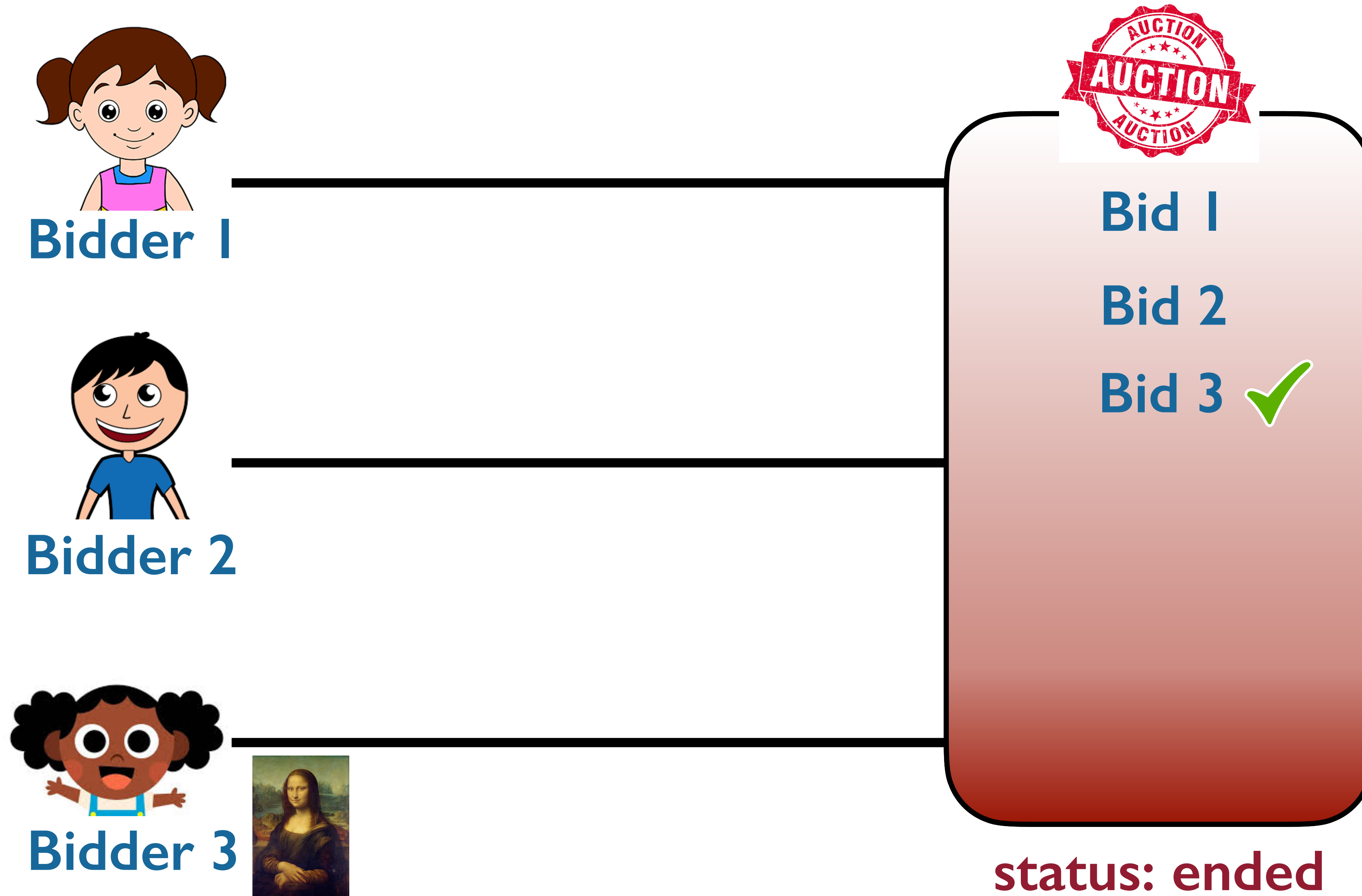
**Bid 2**

**Bid 3** ✓



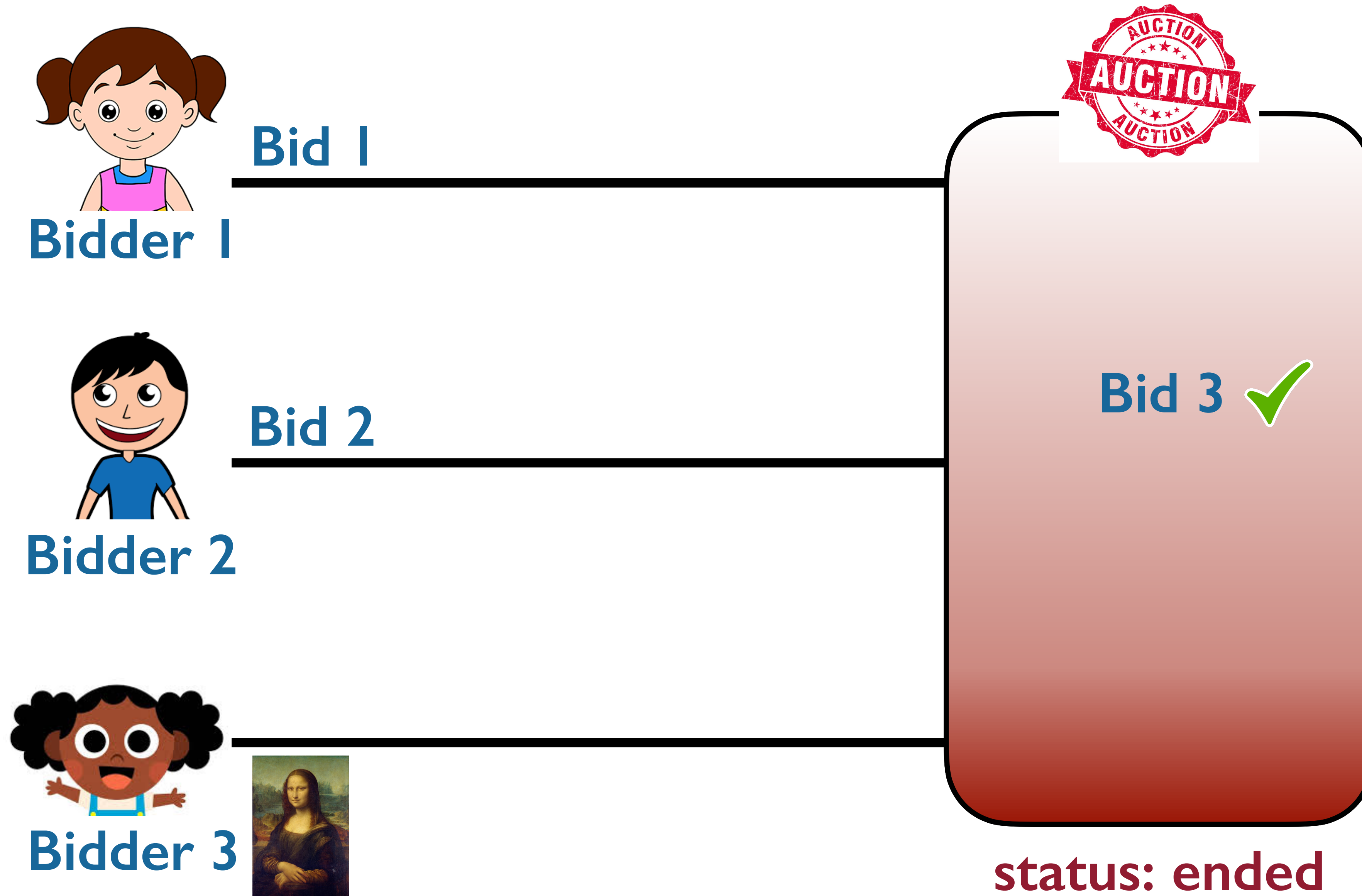
**status: ended**

# Example: Auction Contract





# Example: Auction Contract



# Auction in Solidity

```
function bid() public payable {
    bid = msg.value ;
    bidder = msg.sender ;
    pendingReturns[bidder] = bid ;
    if (bid > highestBid) {
        highestBidder = bidder ;
        highestBid = bid ;
    }
}

function collect() public returns (bool) {
    require (msg.sender != highestBidder) ;
    uint amount = pendingReturns[msg.sender] ;
    msg.sender.send(amount) ;
    return true ;
}
```

# Auction in Solidity

```
function bid() public payable {
    bid = msg.value ;
    bidder = msg.sender ;
    pendingReturns[bidder] = bid ;
    if (bid > highestBid) {
        highestBidder = bidder ;
        highestBid = bid ;
    }
}

function collect() public returns (bool) {
    require (msg.sender != highestBidder) ;
    uint amount = pendingReturns[msg.sender] ;
    msg.sender.send(amount) ;
    return true ;
}
```

*What happens if  
collect is called  
when auction is  
running?*

```
function bid() public payable {
    bid = msg.value ;
    bidder = msg.sender ;
    pendingReturns[bidder] = bid ;
    if (bid > highestBid) {
        highestBidder = bidder ;
        highestBid = bid ;
    }
}

function collect() public returns (bool) {
    require (msg.sender != highestBidder) ;
    uint amount = pendingReturns[msg.sender] ;
    msg.sender.send(amount) ;
    return true ;
}
```

*What happens if  
collect is called  
when auction is  
running?*

*add clause:  
require (status == ended);*

```
function bid() public payable {
    bid = msg.value ;
    bidder = msg.sender ;
    pendingReturns[bidder] = bid ;
    if (bid > highestBid) {
        highestBidder = bidder ;
        highestBid = bid ;
    }
}

function collect() public returns (bool) {
    require (msg.sender != highestBidder) ;
    uint amount = pendingReturns[msg.sender] ;
    msg.sender.send(amount) ;
    return true ;
}
```

# Auction in Solidity

*What happens if collect is called when auction is running?*

*add clause:  
require (status == ended) ;*

```
function bid() public payable {
    bid = msg.value ;
    bidder = msg.sender ;
    pendingReturns[bidder] = bid ;
    if (bid > highestBid) {
        highestBidder = bidder ;
        highestBid = bid ;
    }
}

function collect() public returns (bool) {
    require (msg.sender != highestBidder) ;
    uint amount = pendingReturns[msg.sender] ;
    msg.sender.send(amount) ;
    return true ;
}
```

*What happens if collect is called twice by the same user?*

# Auction in Solidity

*What happens if collect is called when auction is running?*

*add clause:  
require (status == ended) ;*

```
function bid() public payable {  
    bid = msg.value ;  
    bidder = msg.sender ;  
    pendingReturns[bidder] = bid ;  
    if (bid > highestBid) {  
        highestBidder = bidder ;  
        highestBid = bid ;  
    }  
}
```

*What happens if collect is called twice by the same user?*

*set:  
pendingReturns[msg.sender] = 0 ;*

```
function collect() public returns (bool) {  
    require (msg.sender != highestBidder) ;  
    uint amount = pendingReturns[msg.sender] ;  
    msg.sender.send(amount) ;  
    return true ;  
}
```

# Auction in Solidity

*What happens if collect is called when auction is running?*

*What happens if collect is called twice by the same user?*

```
function bid() public payable {
    bid = msg.value ;
    bidder = msg.sender ;
    pendingReturns[bidder] = bid ;
    if (bid > highestBid) {
        highestBidder = bidder ;
        highestBid = bid ;
    }
}
```

*add clause:  
require (status == ended) ;*

*set:  
pendingReturns[msg.sender] = 0 ;*

```
function collect() public returns (bool) {
    require (msg.sender != highestBidder) ;
    uint amount = pendingReturns[msg.sender] ;
    msg.sender.send(amount) ;
    return true ;
}
```

*How much fee should the user pay? No support for static and automatic cost prediction!*





## Digital Contracts



### Challenges:

- 1. contract protocols are violated*
- 2. asset duplication/deletion*
- 3. automatic inference of execution cost*



### Solutions:

- 1. session types enforce protocols*
- 2. linear types track assets*
- 3. resource-aware types infer execution cost*

## Resource-Aware Session Types



## Digital Contracts



### Challenges:

- 1. contract protocols are violated*
- 2. asset duplication/deletion*
- 3. automatic inference of execution cost*



### Solutions:

- 1. session types enforce protocols*
- 2. linear types track assets*
- 3. resource-aware types infer execution cost*

## Resource-Aware Session Types

# What are Session Types?

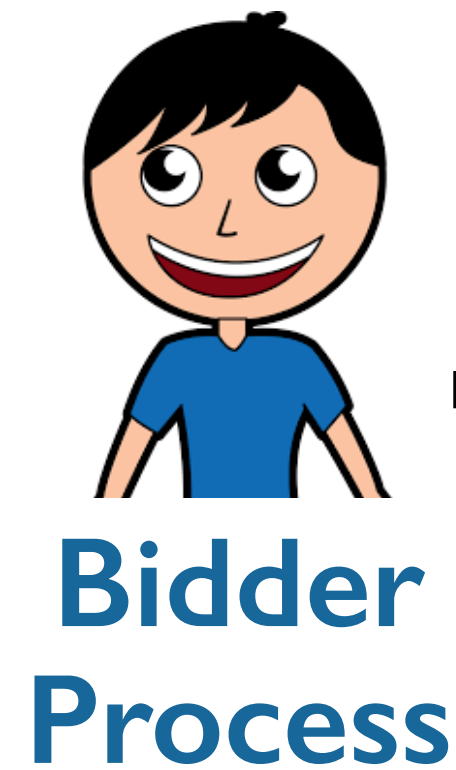
---

7

*Type system for implementing concurrent message-passing processes*

# What are Session Types?

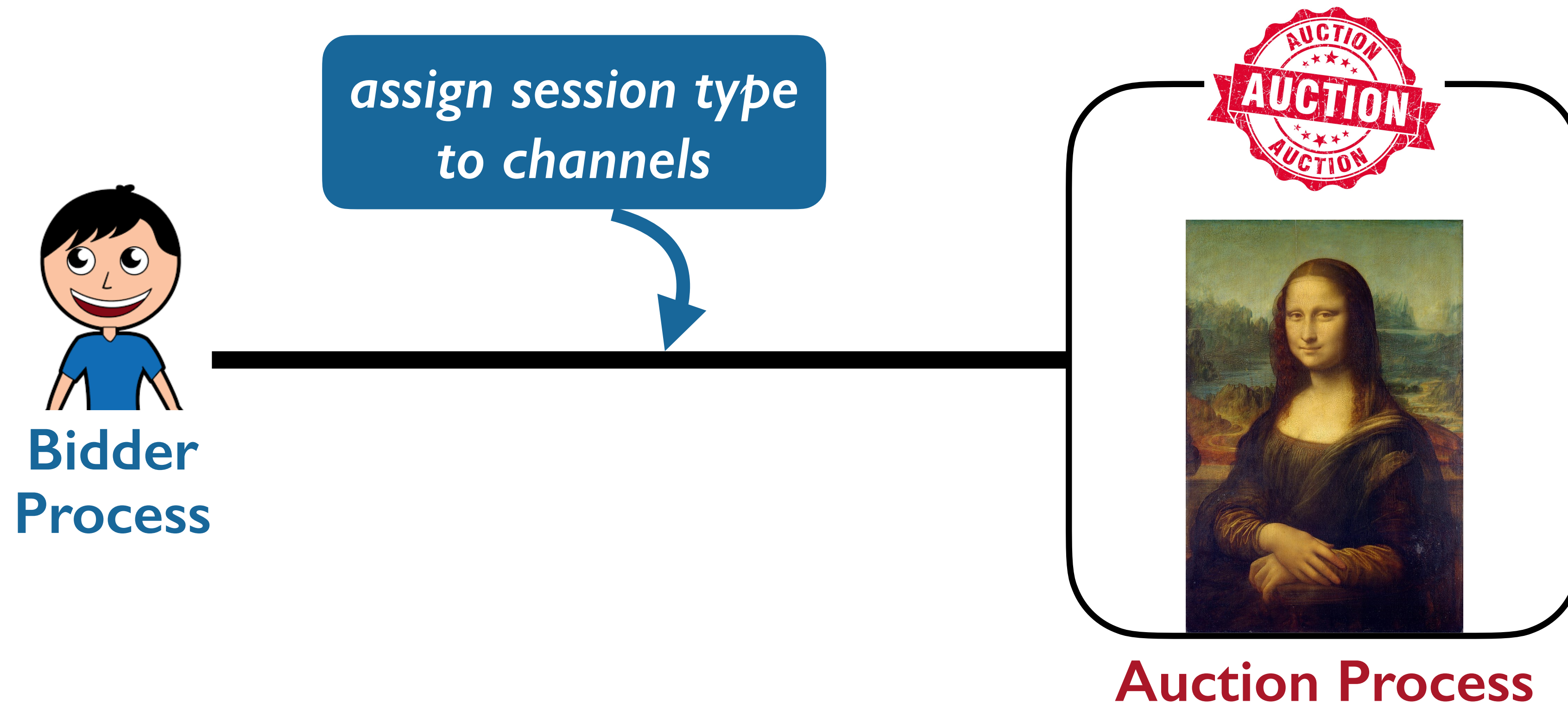
*Type system for implementing concurrent message-passing processes*



**Auction Process**

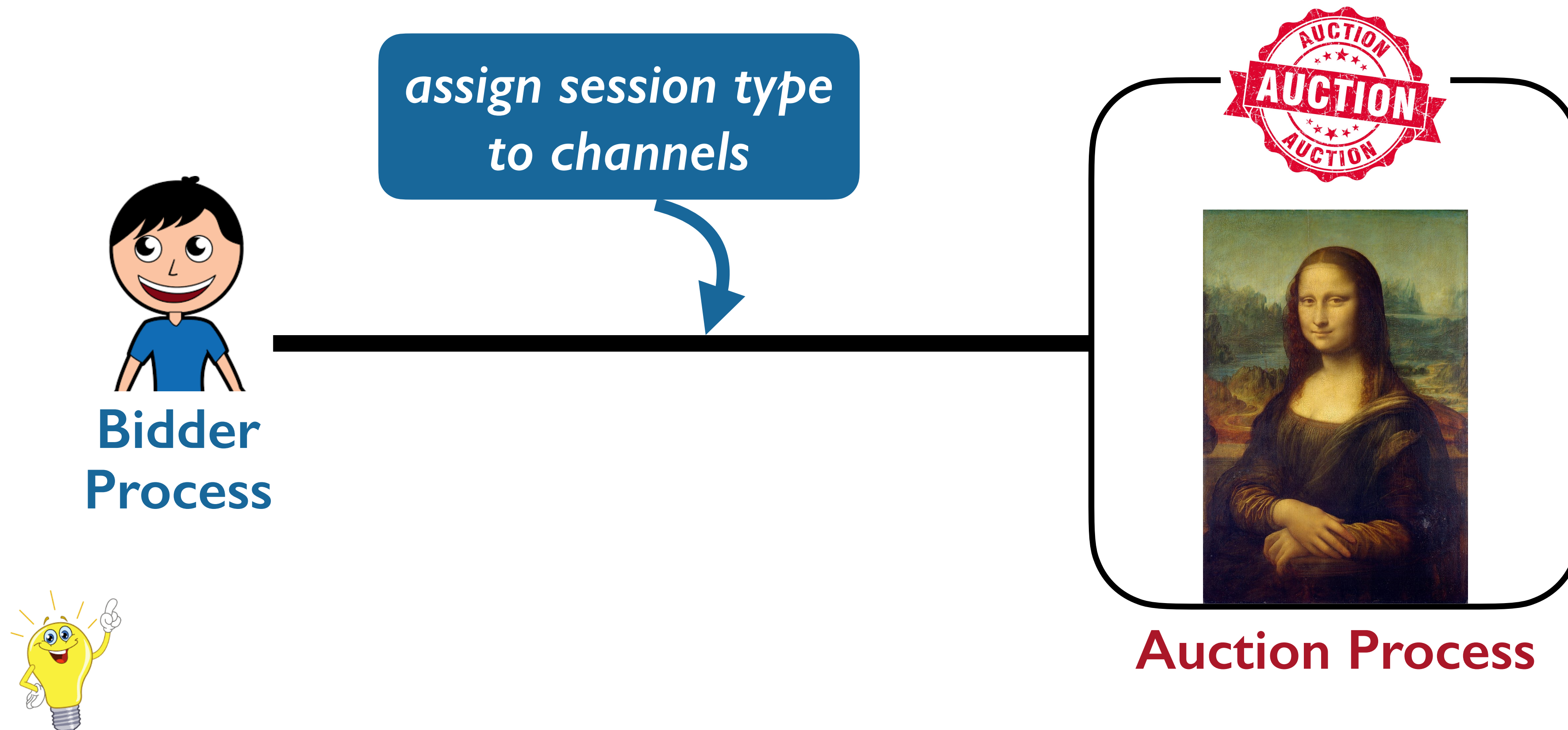
# What are Session Types?

*Type system for implementing concurrent message-passing processes*



# What are Session Types?

*Type system for implementing concurrent message-passing processes*



***Key Idea: Nomos uses session types to express contract protocols***

# Auction as a Session Type

---

# Auction as a Session Type

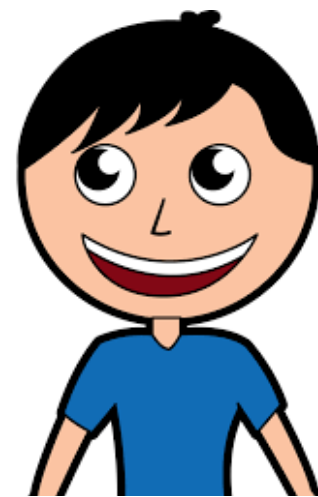
---

$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \mathbf{running} : \& \{ \mathbf{bid} : \text{money} \multimap \text{auction} \}, \\ & \mathbf{ended} : \& \{ \mathbf{collect} : \oplus \{ \mathbf{won} : \text{monalisa} \otimes \text{auction}, \\ & \mathbf{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

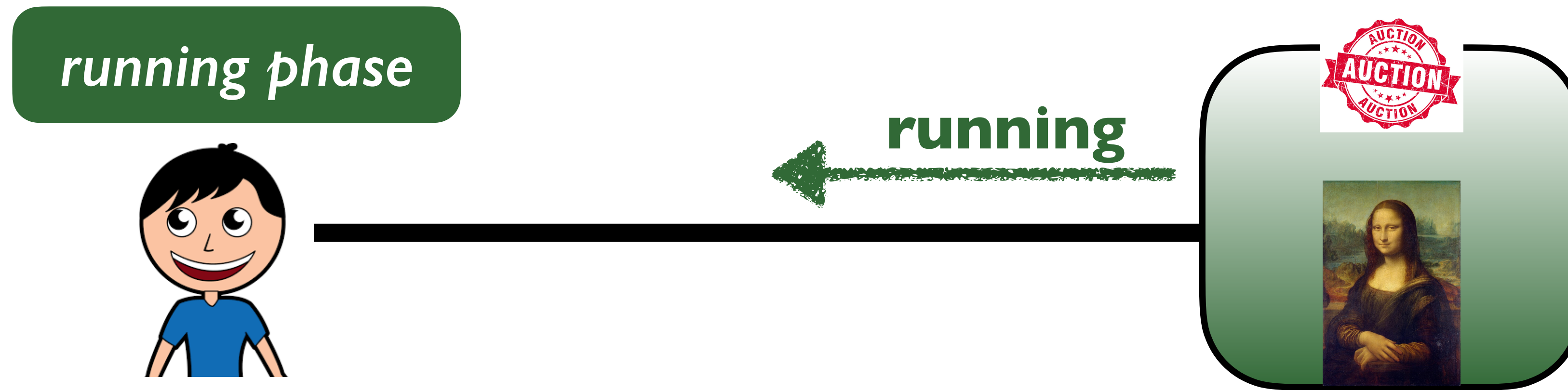


# Auction as a Session Type

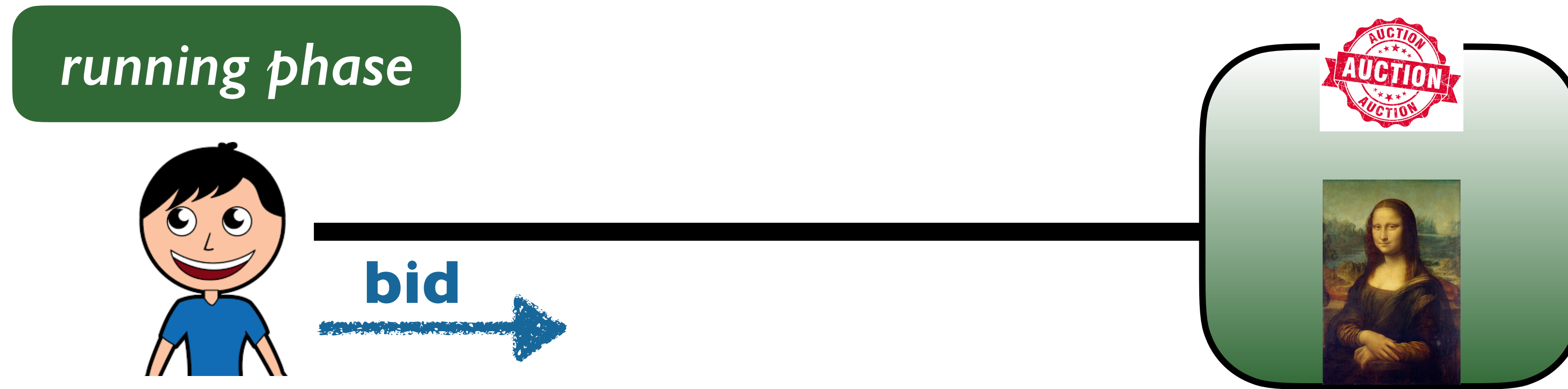
*running phase*


$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \&\{\text{bid} : \text{money} \multimap \text{auction}\}, \\ & \text{ended} : \&\{\text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

# Auction as a Session Type


$$\textit{type} \text{ auction} = \oplus \{ \text{running} : \&\{\text{bid} : \text{money} \multimap \text{auction}\}, \\ \text{ended} : \&\{\text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ \text{lost} : \text{money} \otimes \text{auction} \} \} \}$$

# Auction as a Session Type


$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \multimap \text{auction} \}, \\ & \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

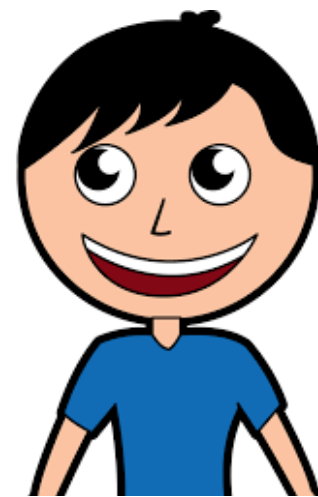
# Auction as a Session Type



$type\ auction = \oplus\{\text{running} : \&\{\text{bid} : \text{money} \multimap \text{auction}\},$   
 $\text{ended} : \&\{\text{collect} : \oplus\{\text{won} : \text{monalisa} \otimes \text{auction},$   
 $\text{lost} : \text{money} \otimes \text{auction}\}\}$

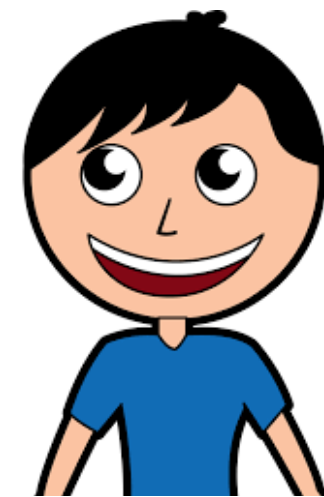
# Auction as a Session Type

*running phase*

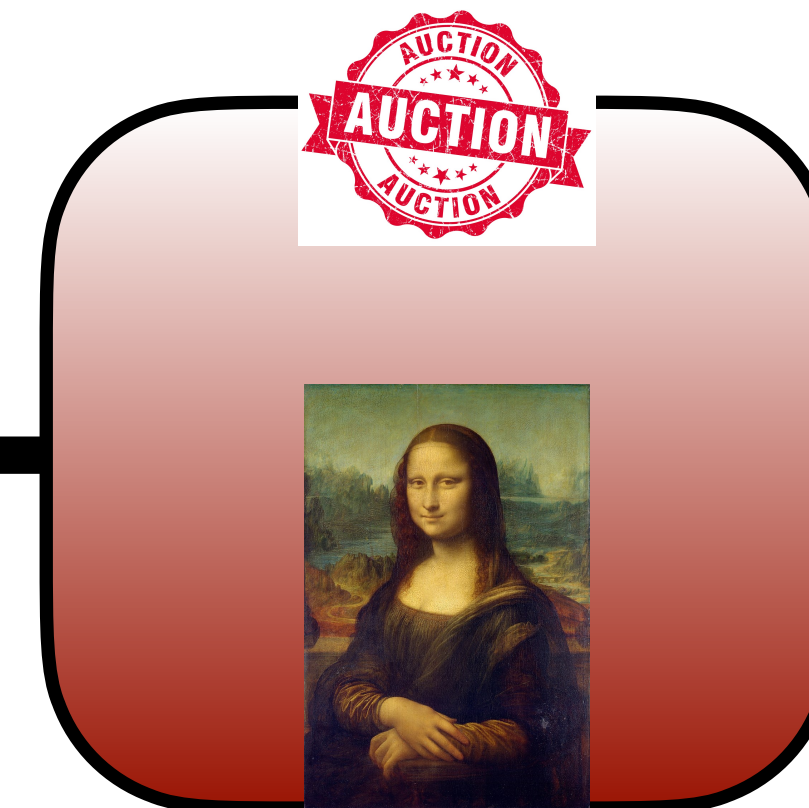
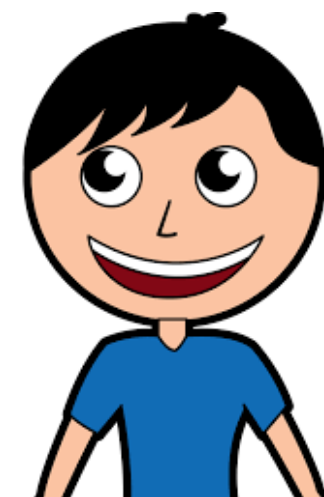

$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \&\{\text{bid} : \text{money} \multimap \text{auction}\}, \\ & \text{ended} : \&\{\text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

# Auction as a Session Type

*running phase*

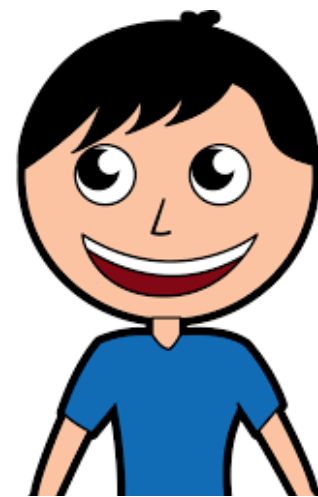


*ended phase*

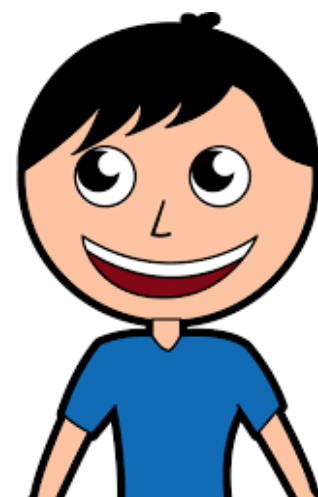

$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \multimap \text{auction} \}, \\ & \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

# Auction as a Session Type

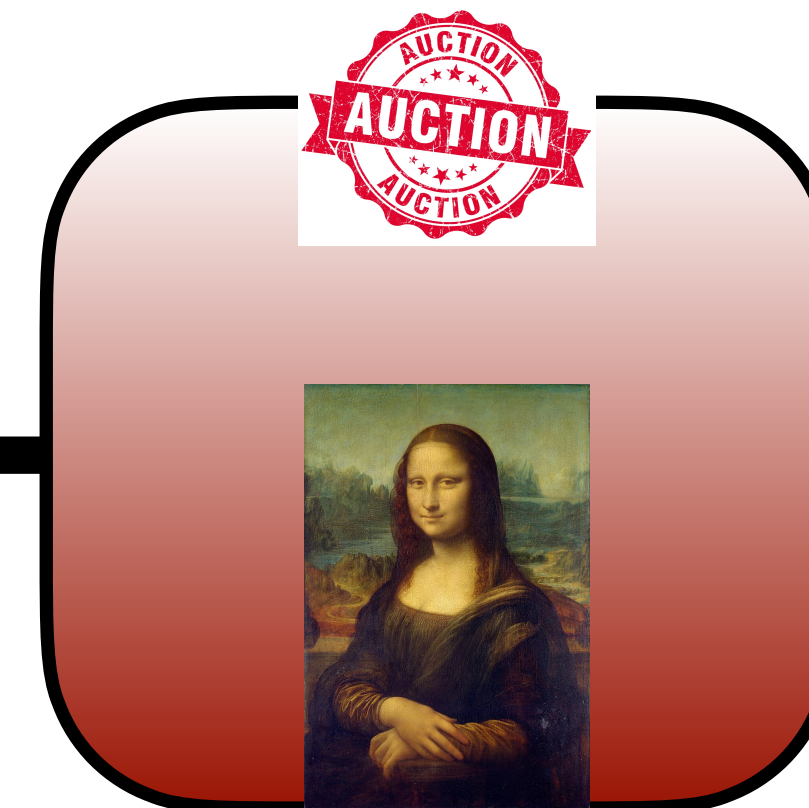
*running phase*



*ended phase*

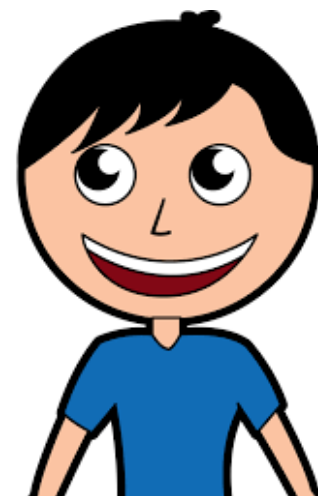


**ended**

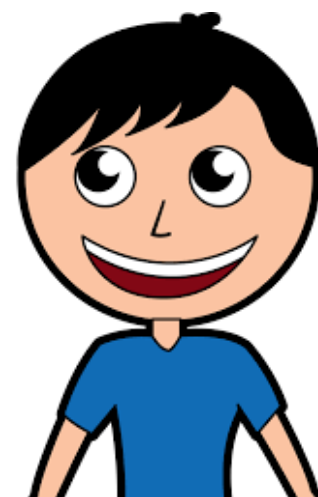

$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \multimap \text{auction} \}, \\ & \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

# Auction as a Session Type

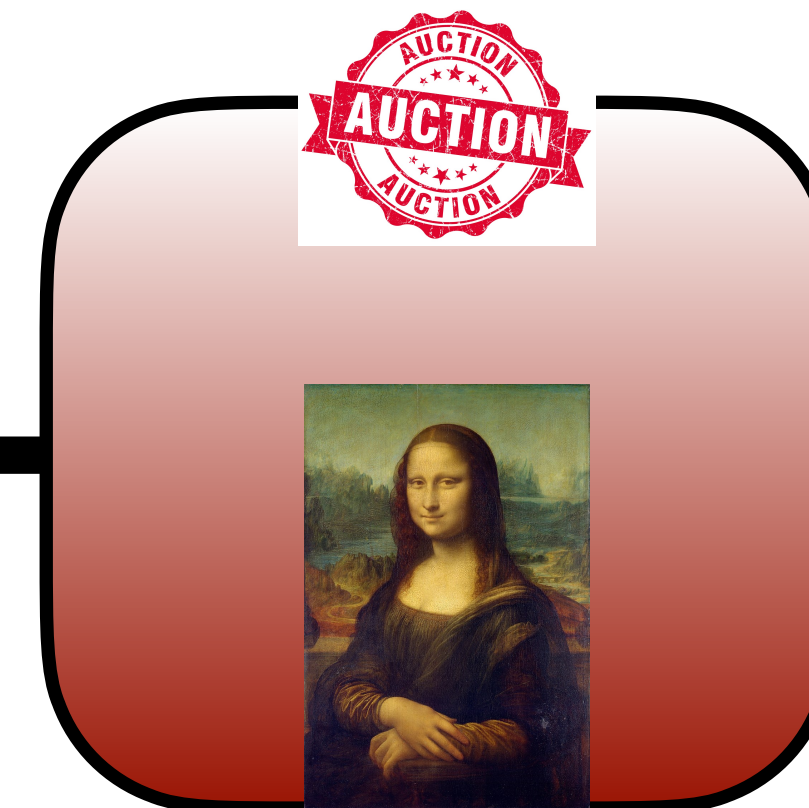
*running phase*



*ended phase*



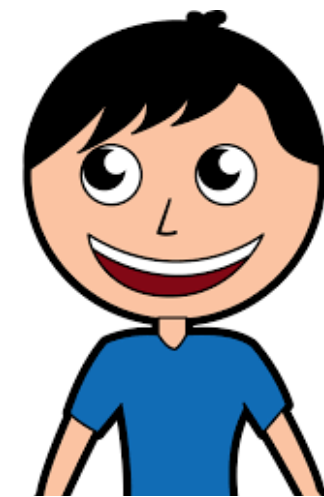
**collect**


$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \multimap \text{auction} \}, \\ & \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

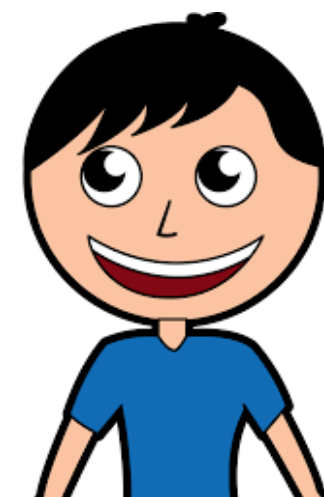


# Auction as a Session Type

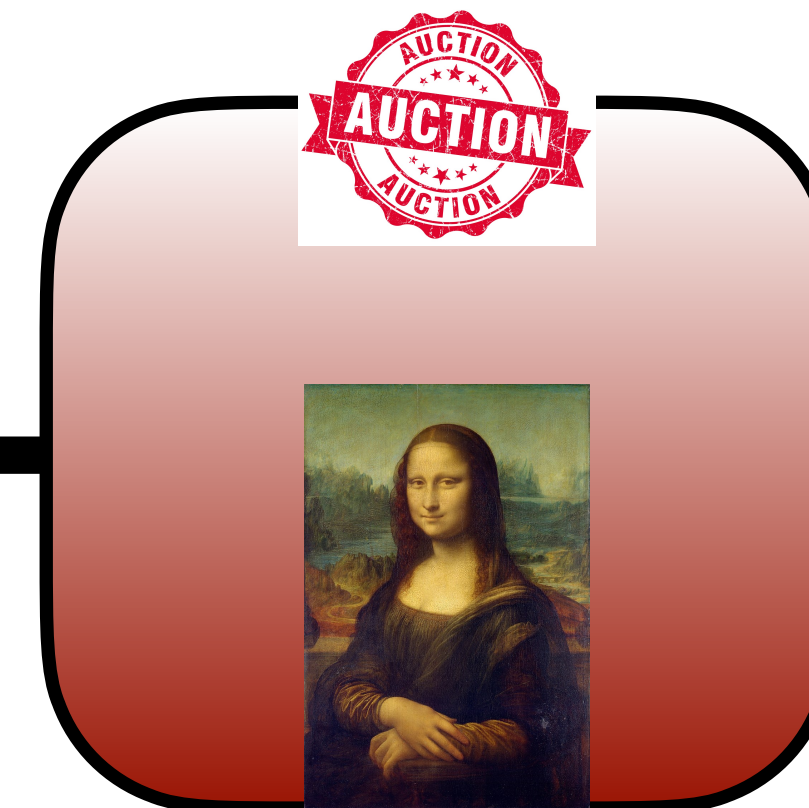
*running phase*



*ended phase*

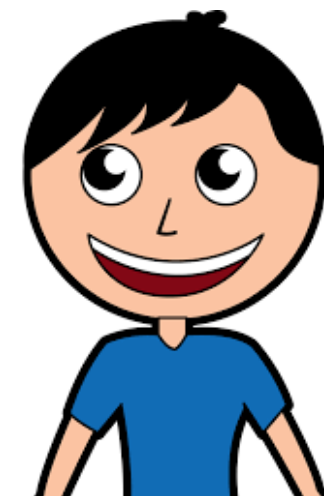


**won** ←

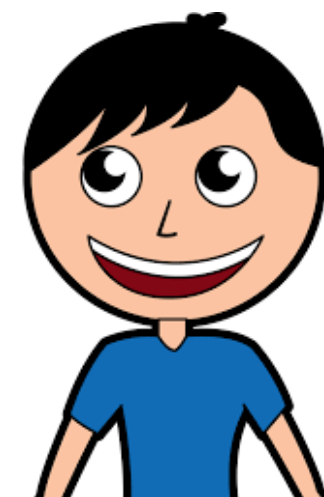

$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \multimap \text{auction} \}, \\ & \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

# Auction as a Session Type

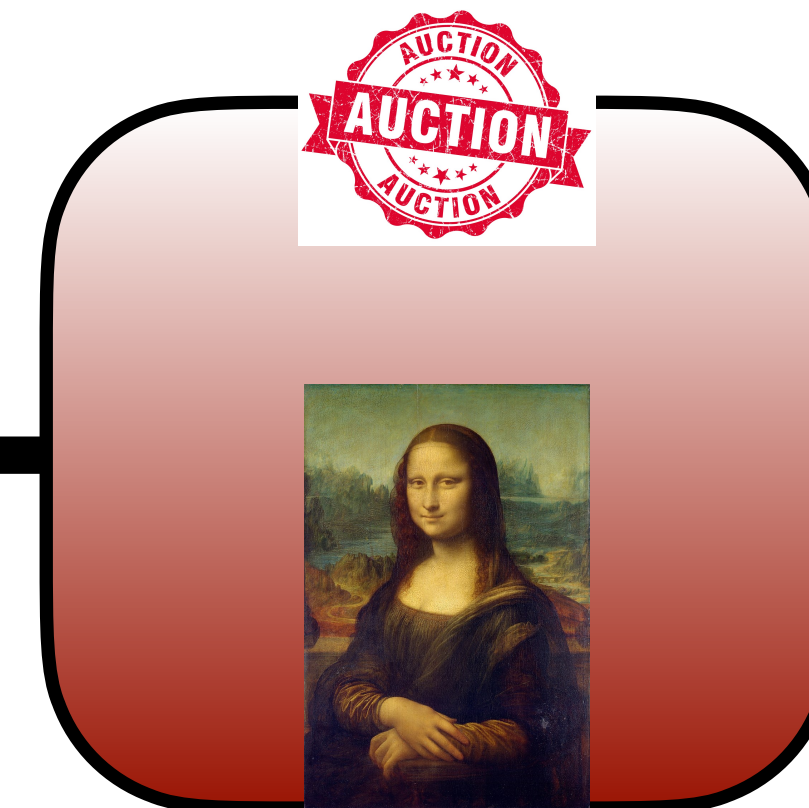
*running phase*



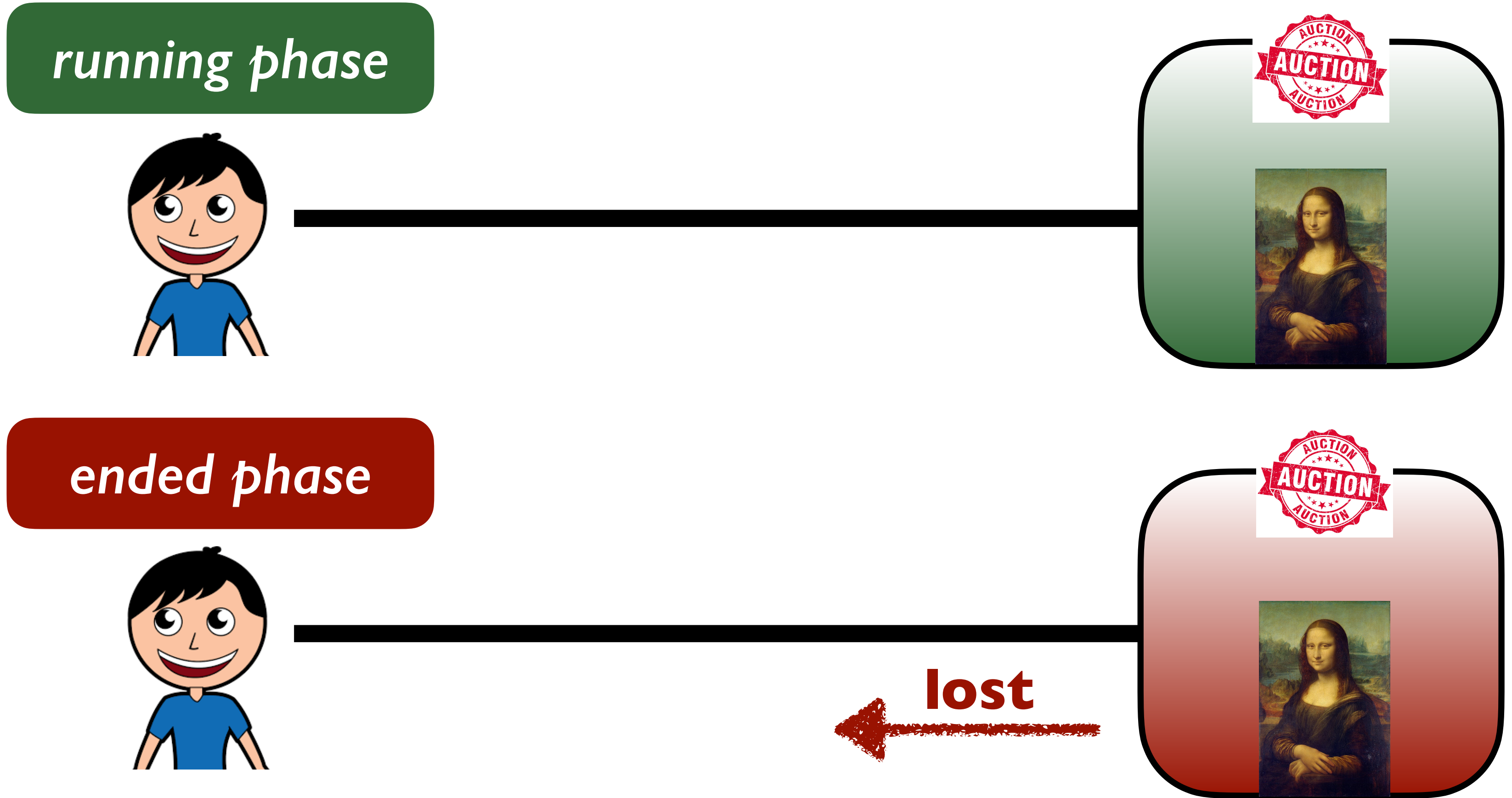
*ended phase*



← **monalisa**


$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \multimap \text{auction} \}, \\ & \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

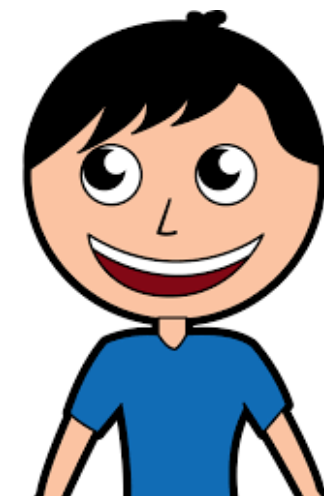
# Auction as a Session Type



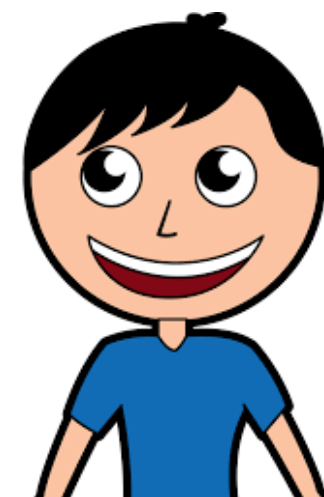
$type\ auction = \oplus\{\text{running} : \&\{\text{bid} : \text{money} \multimap \text{auction}\},$   
 $\text{ended} : \&\{\text{collect} : \oplus\{\text{won} : \text{monalisa} \otimes \text{auction},$   
 $\text{lost} : \text{money} \otimes \text{auction}\}\}$

# Auction as a Session Type

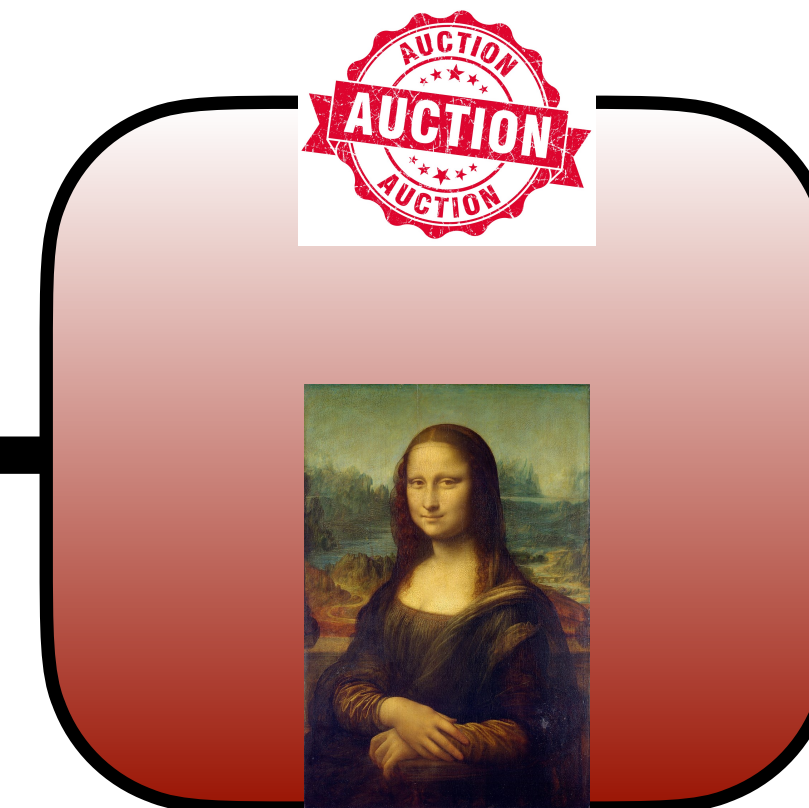
*running phase*



*ended phase*

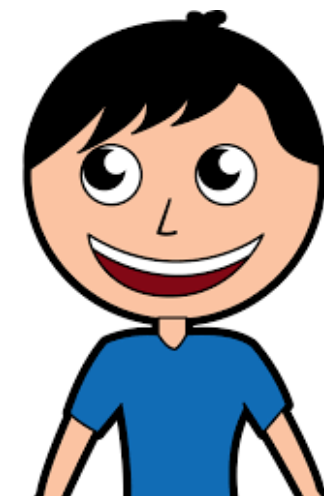


money ←

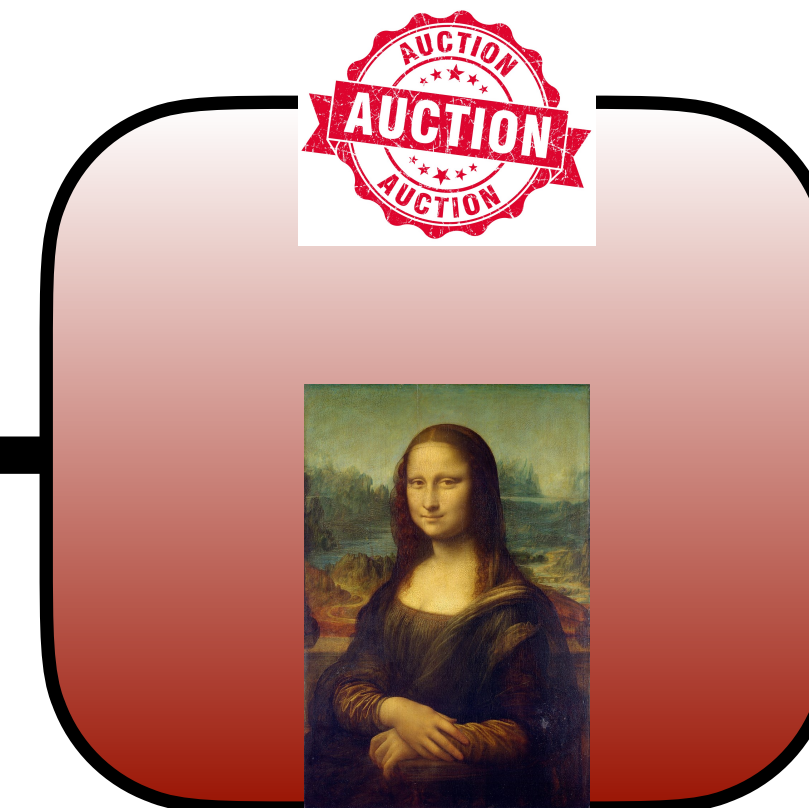
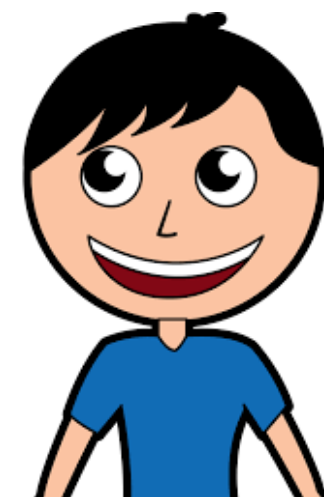

$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \multimap \text{auction} \}, \\ & \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

# Auction as a Session Type

*running phase*



*ended phase*


$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \multimap \text{auction} \}, \\ & \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\ & \text{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$





*Key Idea: Session types in Nomos are derived from linear logic (which is the logic of assets!)*



*Key Idea: Session types in Nomos are derived from linear logic (which is the logic of assets!)*

$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \mathbf{running} : \& \{ \mathbf{bid} : \text{money} \multimap \text{auction} \}, \\ & \mathbf{ended} : \& \{ \mathbf{collect} : \oplus \{ \mathbf{won} : \text{monalisa} \otimes \text{auction}, \\ & \mathbf{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$





*Key Idea: Session types in Nomos are derived from linear logic (which is the logic of assets!)*

$$\begin{aligned} \textit{type} \text{ auction} = & \oplus \{ \mathbf{running} : \& \{ \mathbf{bid} : \text{money} \multimap \text{auction} \}, \\ & \mathbf{ended} : \& \{ \mathbf{collect} : \oplus \{ \mathbf{won} : \text{monalisa} \otimes \text{auction}, \\ & \mathbf{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

*Nomos statically guarantees:*

- ▶ *money and monalisa are neither duplicated nor discarded, only transferred between processes*
- ▶ *no additional machinery needed!!*



*Key Idea: Session types in Nomos are derived from linear logic (which is the logic of assets!)*

$$\begin{aligned} \text{type auction} = & \oplus \{ \mathbf{running} : \& \{ \mathbf{bid} : \text{money} \multimap \text{auction} \}, \\ & \mathbf{ended} : \& \{ \mathbf{collect} : \oplus \{ \mathbf{won} : \text{monalisa} \otimes \text{auction}, \\ & \mathbf{lost} : \text{money} \otimes \text{auction} \} \} \} \end{aligned}$$

*Nomos statically guarantees:*

- ▶ *money and monalisa are neither duplicated nor discarded, only transferred between processes*
- ▶ *no additional machinery needed!!*

# Resource-Aware Auction

---

*cost of bidding = 2, cost of collecting = 3*

# Resource-Aware Auction

*cost of bidding = 2, cost of collecting = 3*



*Key Idea: Nomos enhances session types with potential (or fee) that pays for execution cost*

# Resource-Aware Auction

*cost of bidding = 2, cost of collecting = 3*



*Key Idea: Nomos enhances session types with potential (or fee) that pays for execution cost*

$$\begin{aligned} \text{type auction} = & \triangleleft^3 \oplus \{ \text{running} : \&\{ \text{bid} : \text{money} \multimap \triangleright^1 \text{auction} \}, \\ & \text{ended} : \&\{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \triangleright^0 \text{auction}, \\ & \text{lost} : \text{money} \otimes \triangleright^0 \text{auction} \} \} \} \end{aligned}$$

# Resource-Aware Auction

*cost of bidding = 2, cost of collecting = 3*



*Key Idea: Nomos enhances session types with potential (or fee) that pays for execution cost*

$type\ auction = \triangleleft^3 \oplus \{ \mathbf{running} : \&\{ \mathbf{bid} : money \multimap \triangleright^1 auction \},$   
 $\mathbf{ended} : \&\{ \mathbf{collect} : \oplus \{ \mathbf{won} : monalisa \otimes \triangleright^0 auction,$   
 $\mathbf{lost} : money \otimes \triangleright^0 auction \} \} \}$

*user pays 3 units as fees  
at start of execution*

# Resource-Aware Auction

*cost of bidding = 2, cost of collecting = 3*



*Key Idea: Nomos enhances session types with potential (or fee) that pays for execution cost*

$type\ auction = \triangleleft^3 \oplus \{ \mathbf{running} : \&\{ \mathbf{bid} : money \multimap \triangleright^1 auction \},$   
 $\mathbf{ended} : \&\{ \mathbf{collect} : \oplus \{ \mathbf{won} : monalisa \otimes \triangleright^0 auction,$   
 $\mathbf{lost} : money \otimes \triangleright^0 auction \} \} \}$

*user pays 3 units as fees  
at start of execution*

*return leftover fee back  
to user in the end*

# Resource-Aware Auction

*cost of bidding = 2, cost of collecting = 3*



*Key Idea: Nomos enhances session types with potential (or fee) that pays for execution cost*

$type\ auction = \triangleleft^3 \oplus \{ \mathbf{running} : \&\{ \mathbf{bid} : money \multimap \triangleright^1 auction \},$   
 $\mathbf{ended} : \&\{ \mathbf{collect} : \oplus \{ \mathbf{won} : monalisa \otimes \triangleright^0 auction,$   
 $\mathbf{lost} : money \otimes \triangleright^0 auction \} \} \}$

*user pays 3 units as fees at start of execution*

*return leftover fee back to user in the end*

*Potential annotations are automatically inferred using LP solver*



- ▶ *session types enforce contract protocols*
- ▶ *linear types track assets, prevent duplication/deletion*
- ▶ *resource-aware types infer execution cost*
- ▶ *re-entrancy attacks eliminated by construction*
- ▶ *type safety theorem guarantees all properties*
- ▶ *extensive evaluation on standard digital contracts*