Resource-Aware Session Types for Digital Contracts

Ankush Das
Carnegie Mellon University

February 27, 2019

Purdue University

Digital Contracts

- Programs to digitally facilitate the execution of a transaction between distrusting parties
- Transactions are carried out by miners and stored on a global distributed ledger, or blockchain
- User pays for the execution cost of transaction

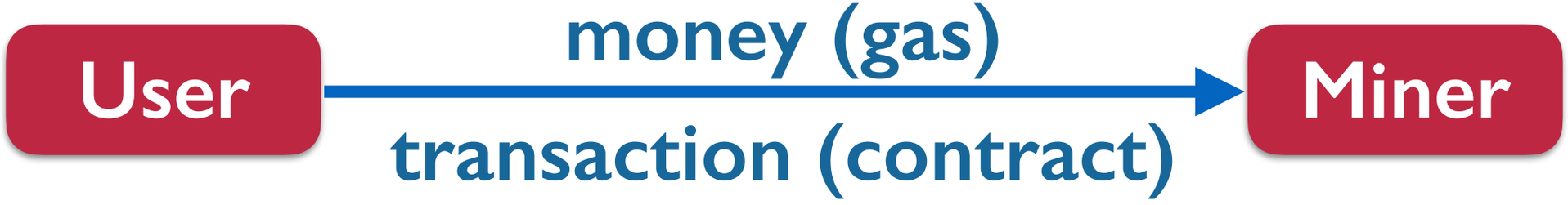
Digital Contracts

- Programs to digitally facilitate the execution of a transaction between distrusting parties
- Transactions are carried out by miners and stored on a global distributed ledger, or blockchain
- User pays for the execution cost of transaction

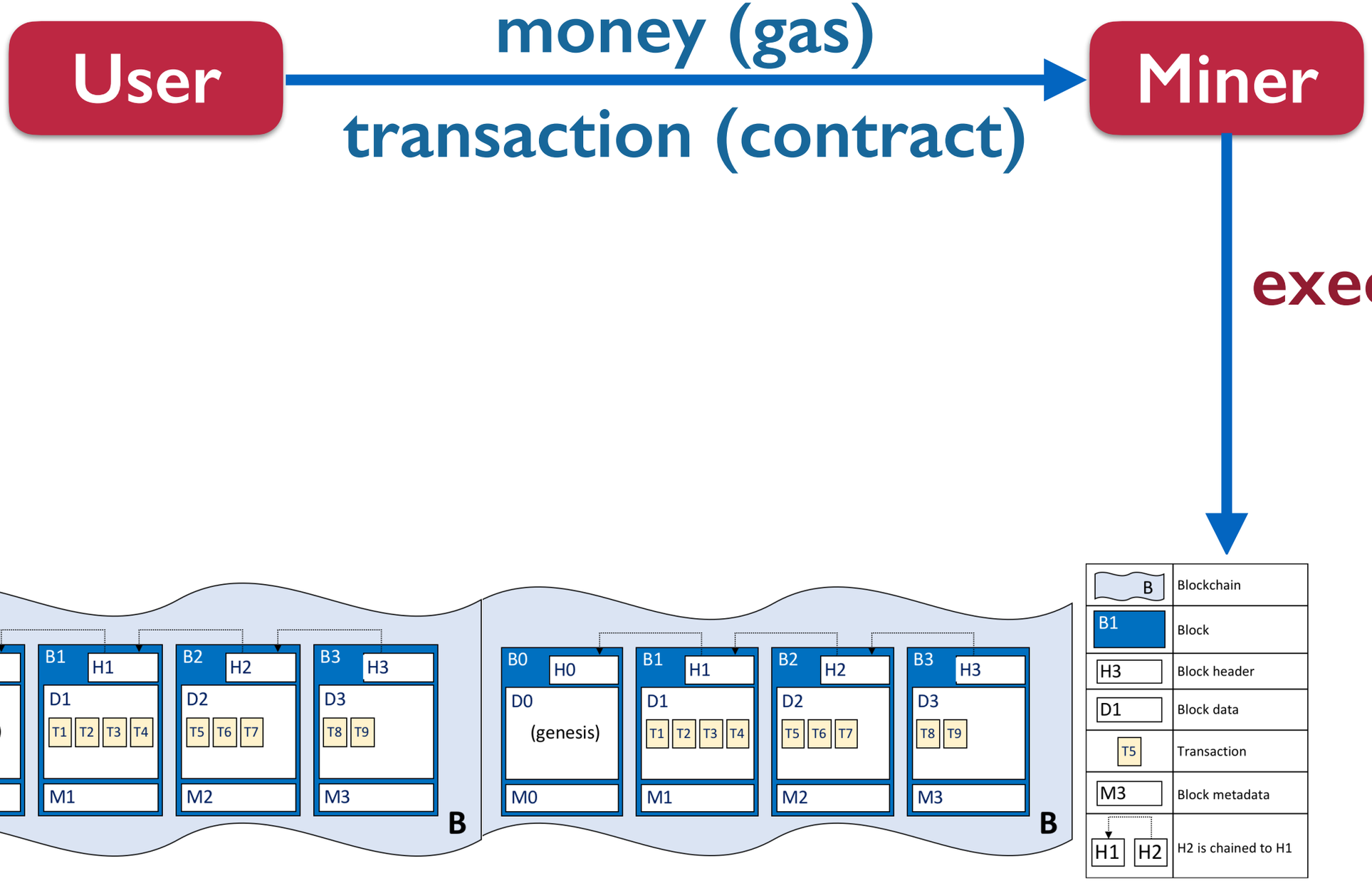
set standard assigns cost to each operation

Execution Model

Execution Model



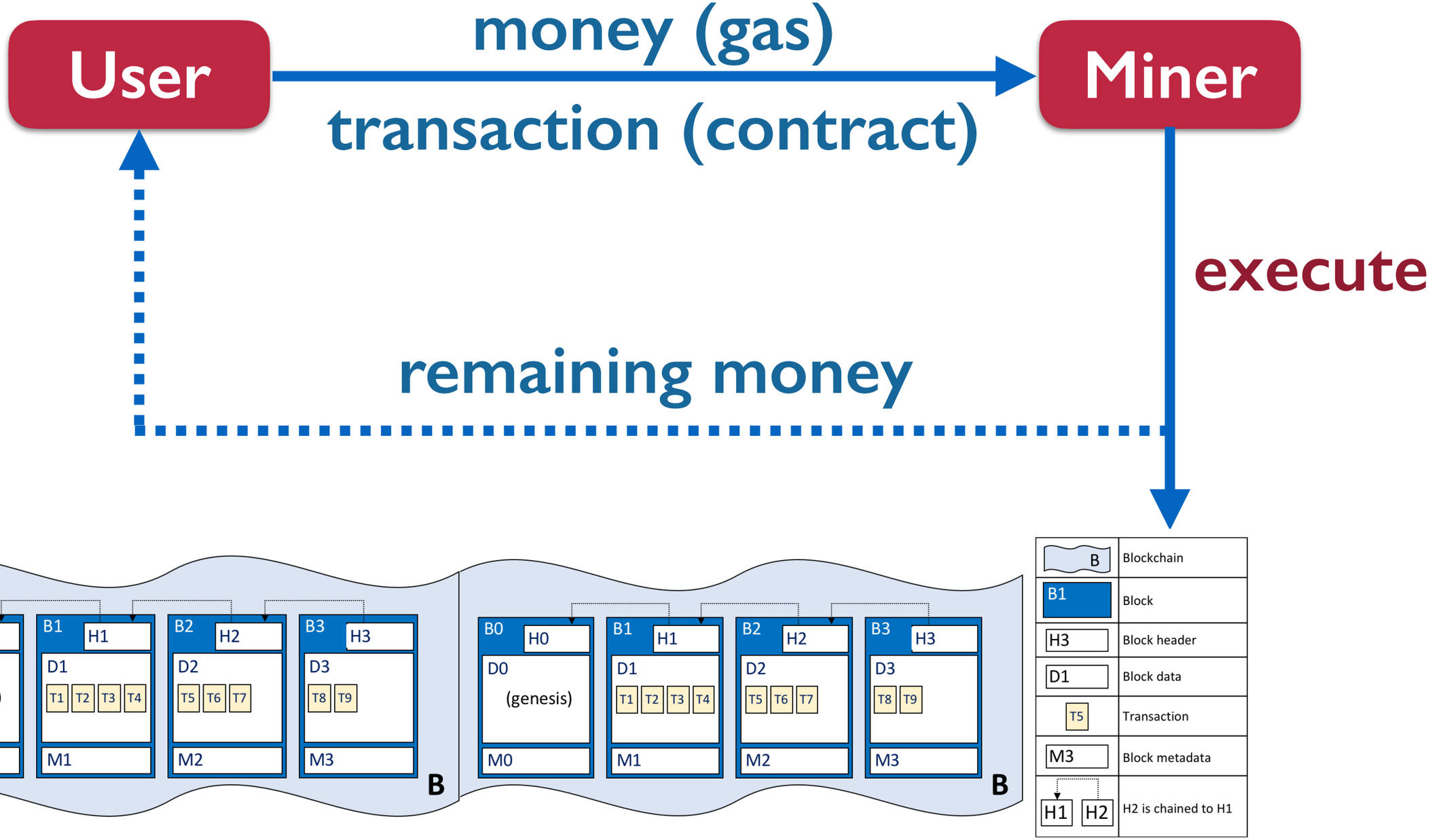
Execution Model



global ledger (blockchain)

new block

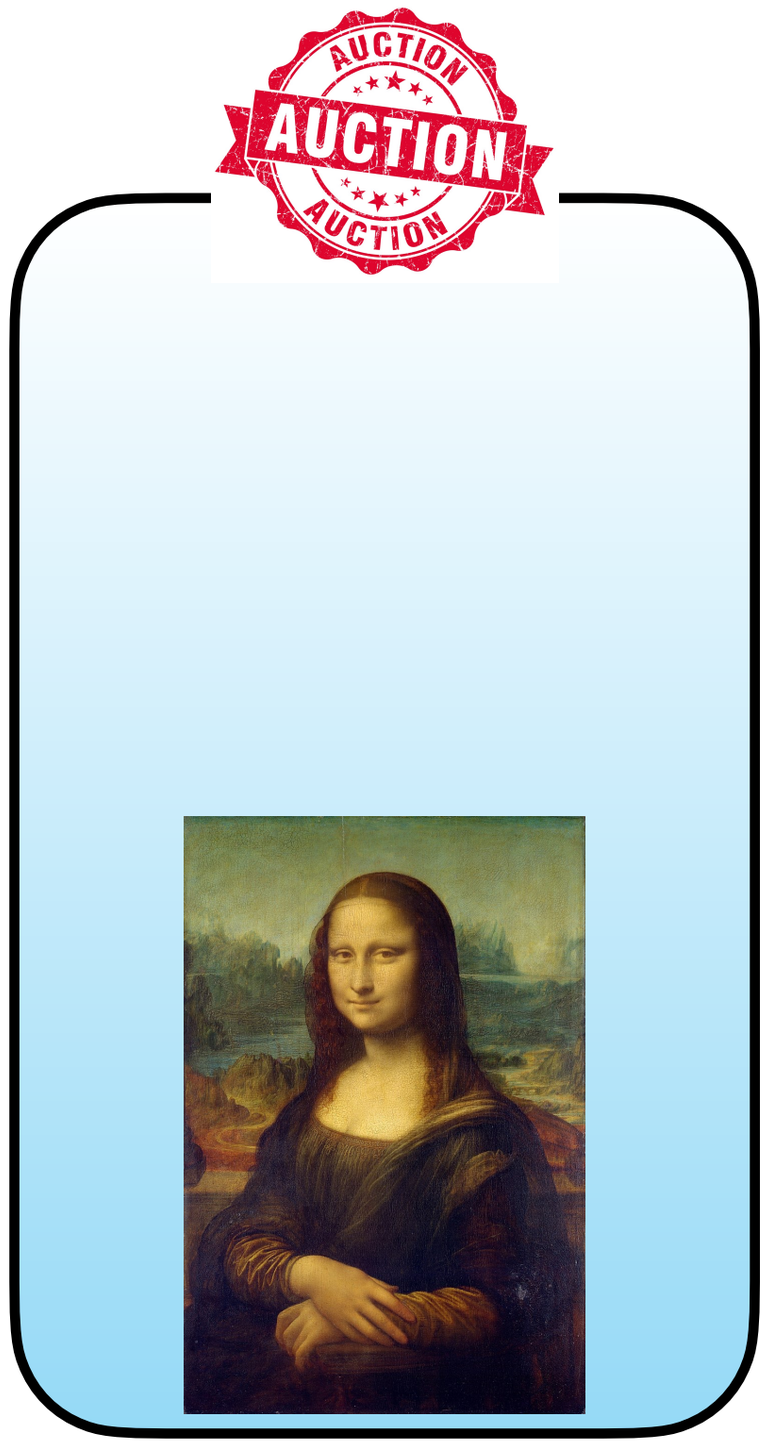
Execution Model



global ledger (blockchain)

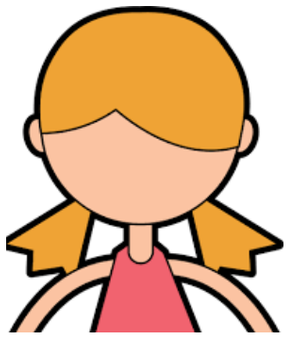
new block

Auction Contract



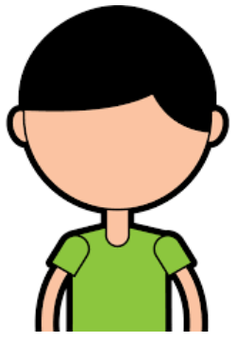
status: running

Auction Contract



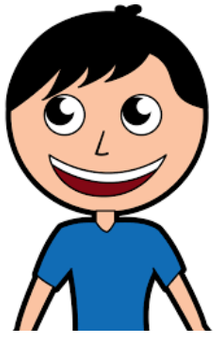
Bid 1

Bidder 1



Bid 2

Bidder 2

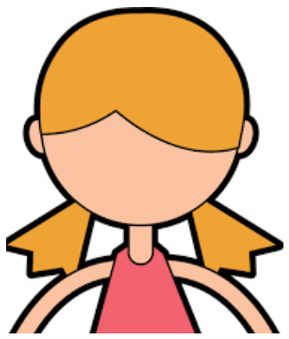


Bid 3

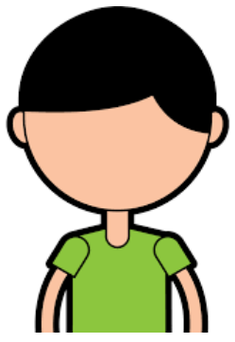
Bidder 3

status: running

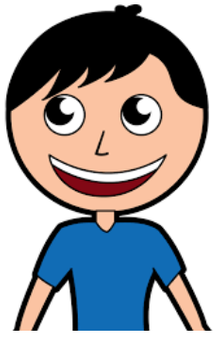
Auction Contract



Bidder 1



Bidder 2



Bidder 3

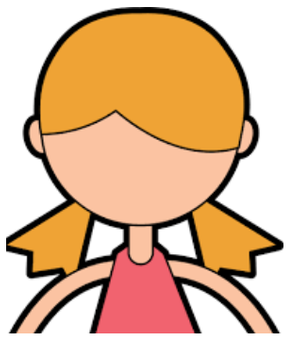
Bid 1

Bid 2

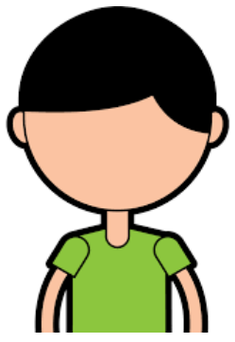
Bid 3

status: running

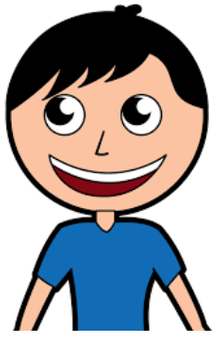
Auction Contract



Bidder 1



Bidder 2



Bidder 3

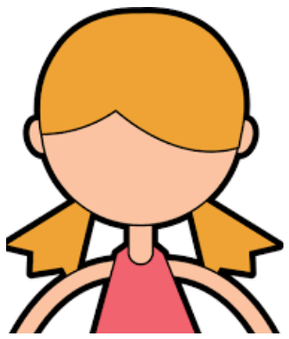
Bid 1

Bid 2

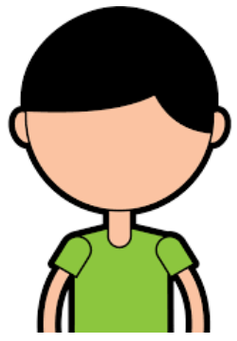
Bid 3

status: ended

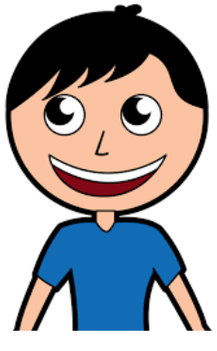
Auction Contract



Bidder 1



Bidder 2



Bidder 3

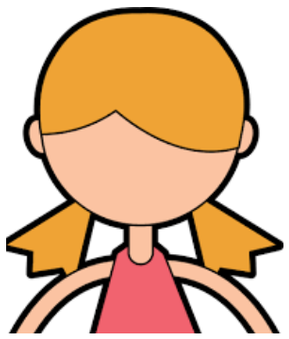
Bid 1

Bid 2

Bid 3

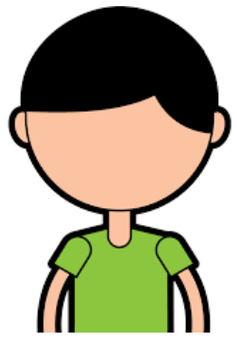
status: ended

Auction Contract



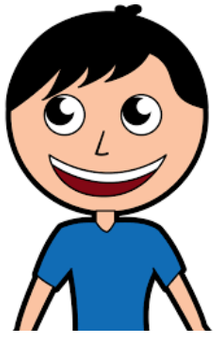
Bidder 1

Bid 1



Bidder 2

Bid 2



Bidder 3

Bid 3

status: ended

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

Auction in Solidity


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

Hint: think of the functions as server-client interactions

Auction in Solidity


```
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?

Hint: think of the functions as server-client interactions

Auction in Solidity


```
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?

Hint: think of the functions as server-client interactions

add require (status == ended);

Auction in Solidity


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

Protocol is not statically enforced!

What happens if collect is called when auction is running?

Hint: think of the functions as server-client interactions

add require (status == ended);

Auction in Solidity


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

Auction in Solidity


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

Hint: think of the functions as server-client interactions

Auction in Solidity


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

What happens if collect is called twice?

Hint: think of the functions as server-client interactions

Auction in Solidity


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

What happens if collect is called twice?

Hint: think of the functions as server-client interactions

set pendingReturns[msg.sender] = 0

Auction in Solidity


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

Linearity is not enforced!

What happens if collect is called twice?

Hint: think of the functions as server-client interactions

set pendingReturns[msg.sender] = 0

Auction in Solidity


```
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    pendingReturns[msg.sender] = 0;
    return true;
}
```

Auction in Solidity


```
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    pendingReturns[msg.sender] = 0;
    return true;
}
```

Auction in Solidity


```
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    pendingReturns[msg.sender] = 0;
    return true;
}
```

'send' transfers control to user who can call collect

Auction in Solidity


```
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    pendingReturns[msg.sender] = 0;
    return true;
}
```

'send' transfers control to user who can call collect

'send' should be the last instruction

Auction in Solidity


```
function collect() public returns (bool) {
  require (msg.sender != highestBidder);
  require (status == ended);
  uint amount = pendingReturns[msg.sender];
  msg.sender.send(amount);
  pendingReturns[msg.sender] = 0;
  return true;
}
```

Re-entrancy Attack

'send' transfers control to user who can call collect

'send' should be the last instruction

Reentrancy Attacks in News

The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

Michael del Castillo

Jun 17, 2016 at 14:00 UTC • Updated Jun 18, 2016 at 14:46 UTC

NEWS

ETHEREUM

ChainSecurity: Ethereum's Constantinople upgrade "enables new Reentrancy Attack"

JANUARY 15, 2019, 3:12PM EDT

Clever Ethereum honeypot lets coins come in but won't let them back out

John Biggs @johnbiggs / 1 year ago

Comment

Auction in Solidity


```
function collect() public returns (bool) {
  require (msg.sender != highestBidder);
  require (status == ended);
  uint amount = pendingReturns[msg.sender];
  pendingReturns[msg.sender] = 0;
  msg.sender.send(amount);
  return true;
}
```

Auction in Solidity


```
function collect() public returns (bool) {
  require (msg.sender != highestBidder);
  require (status == ended);
  uint amount = pendingReturns[msg.sender];
  pendingReturns[msg.sender] = 0;
  msg.sender.send(amount);
  return true;
}
```

Resource consumption?
User needs to pay appropriate gas

Auction in Solidity


```
function collect() public returns (bool) {
  require (msg.sender != highestBidder);
  require (status == ended);
  uint amount = pendingReturns[msg.sender];
  pendingReturns[msg.sender] = 0;
  msg.sender.send(amount);
  return true;
}
```

Resource consumption?
User needs to pay appropriate gas

APPENDIX G. FEE SCHEDULE

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.

EVM cost model

Auction in Solidity


```
function collect() public returns (bool) {
  require (msg.sender != highestBidder);
  require (status == ended);
  uint amount = pendingReturns[msg.sender];
  pendingReturns[msg.sender] = 0;
  msg.sender.send(amount);
  return true;
}
```

Automatic Resource Analysis

Resource consumption?

User needs to pay appropriate gas

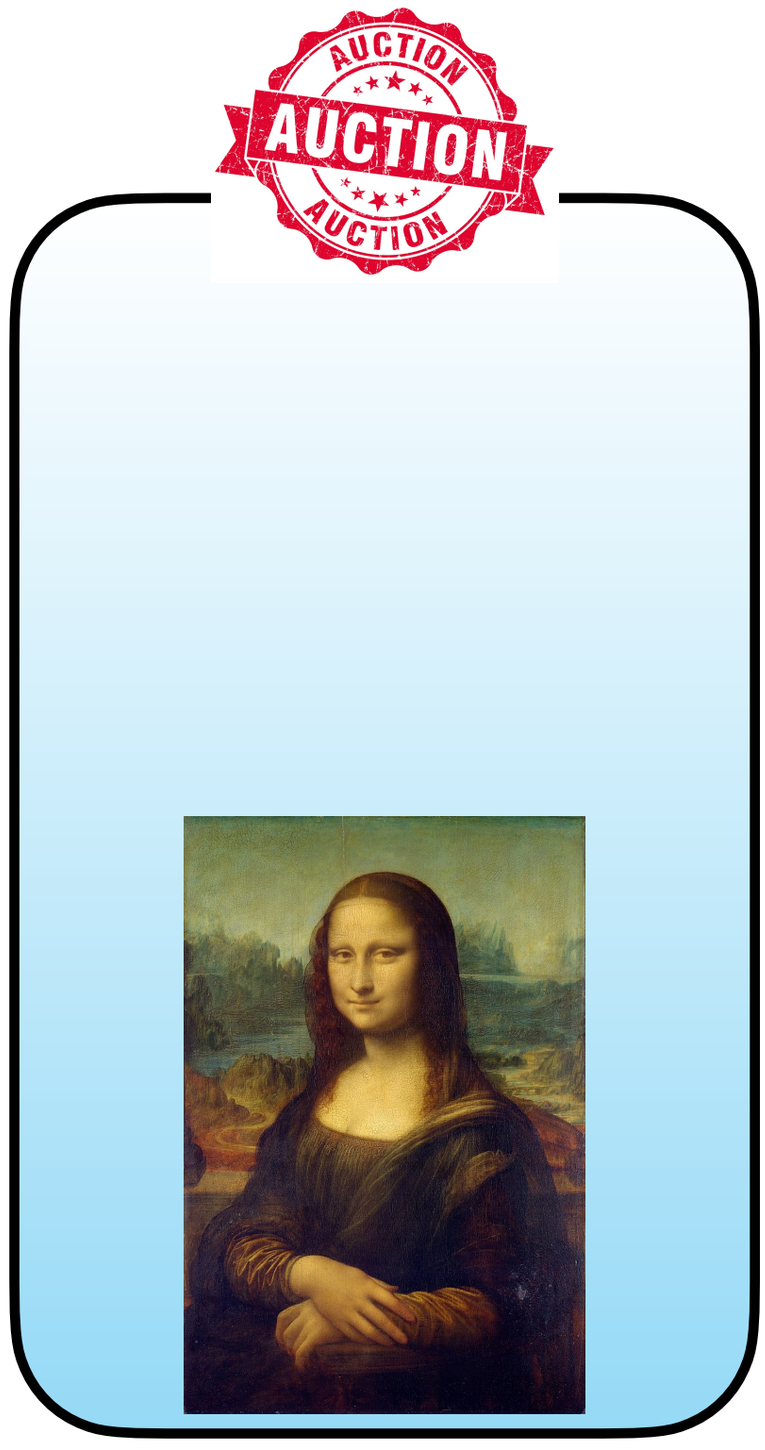
Appendix G. Fee Schedule

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.

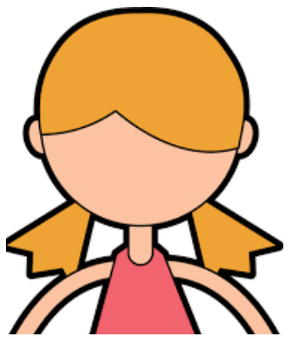
EVM cost model

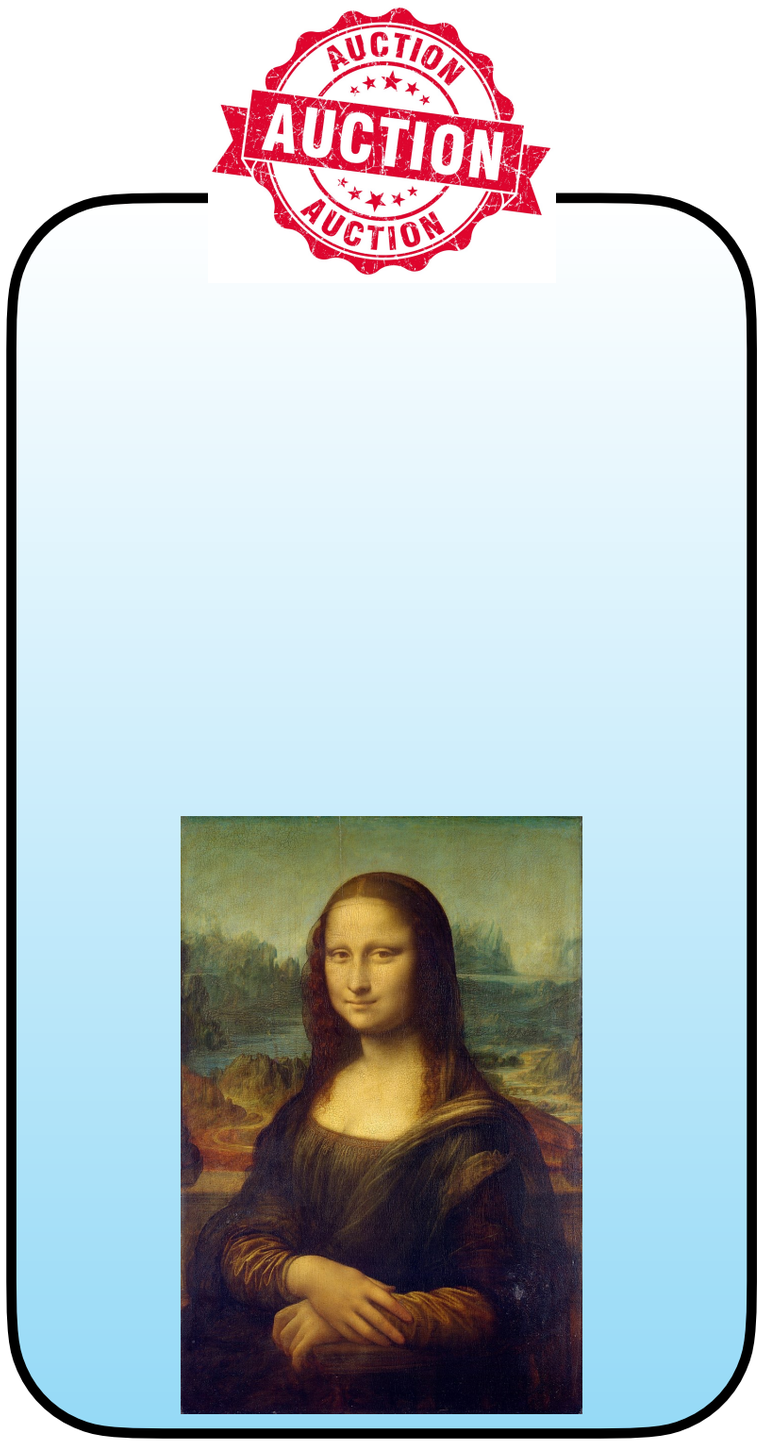
Auction Protocol



Auction Protocol

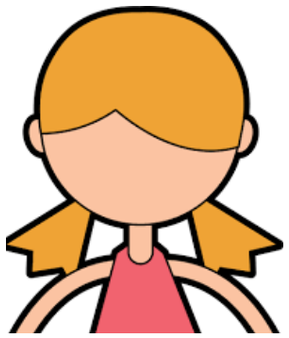
bidding phase



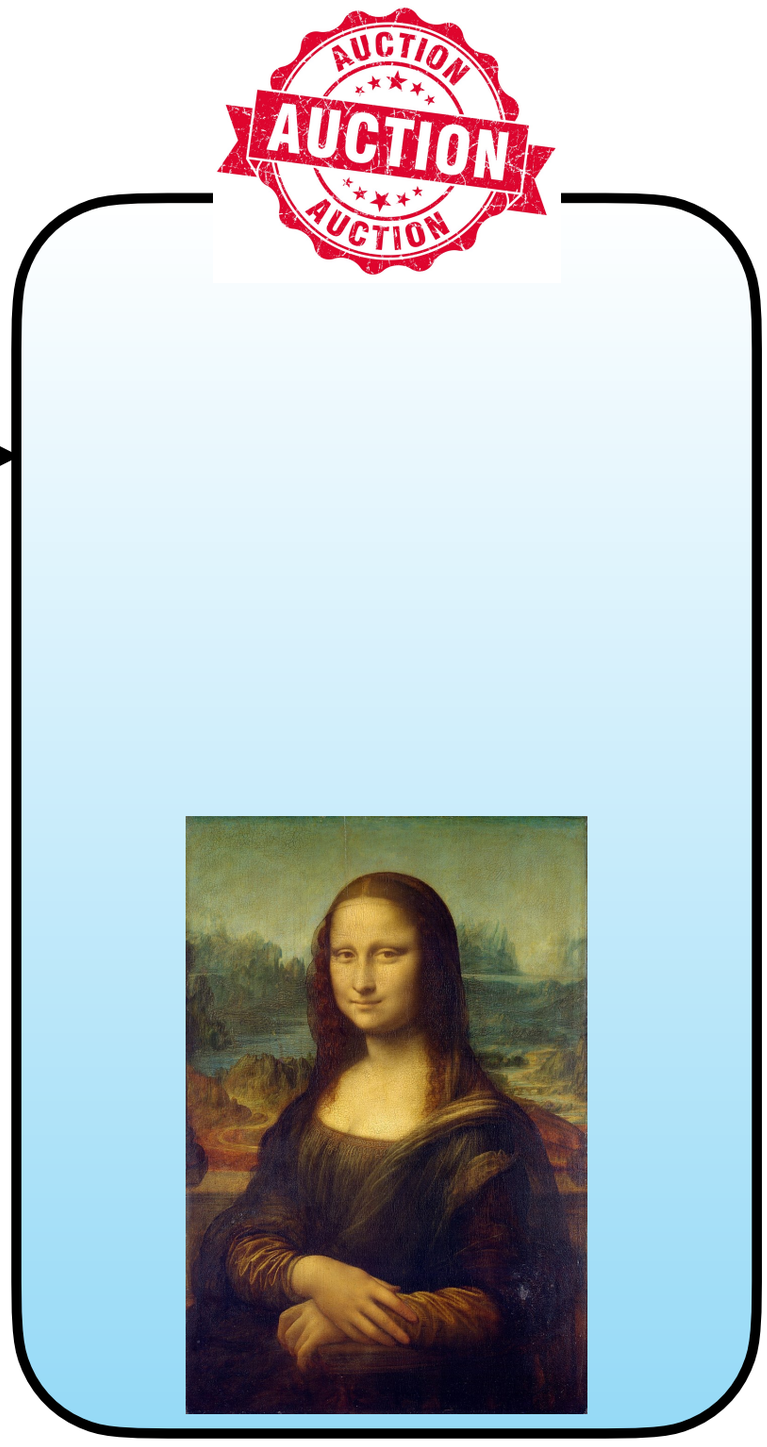


Auction Protocol

bidding phase

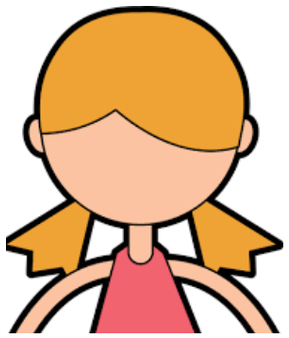


id, money



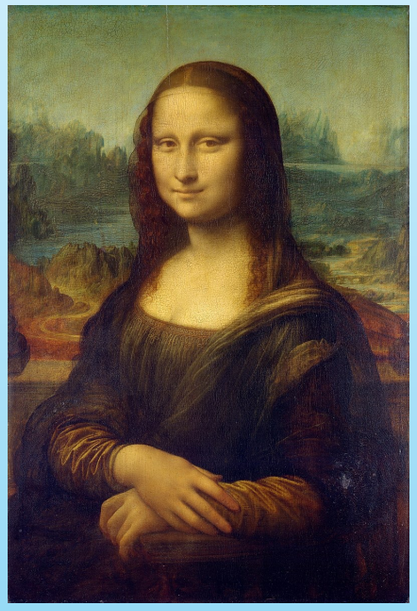
Auction Protocol

bidding phase



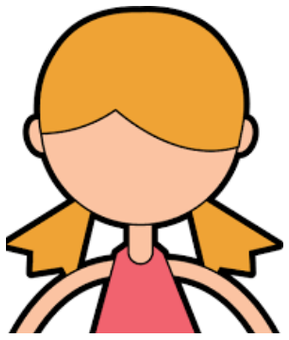
id, money

recurse



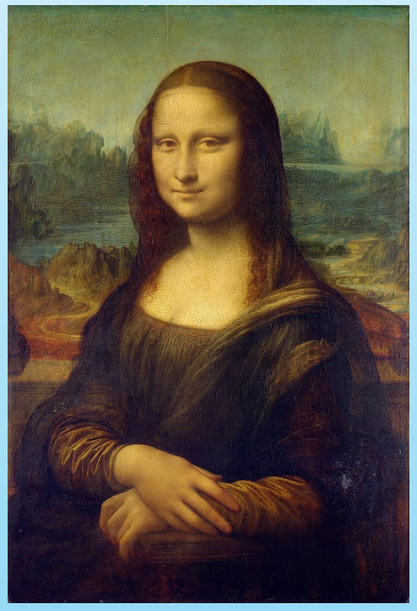
Auction Protocol

bidding phase

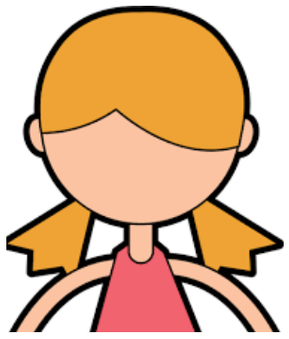


id, money

recurse

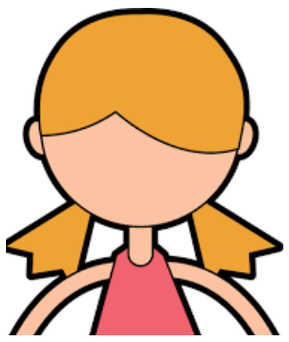


ended phase



Auction Protocol

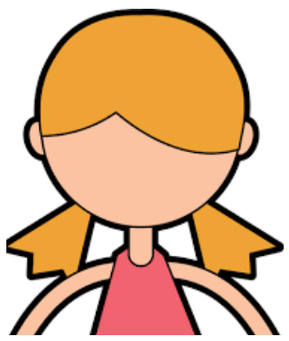
bidding phase



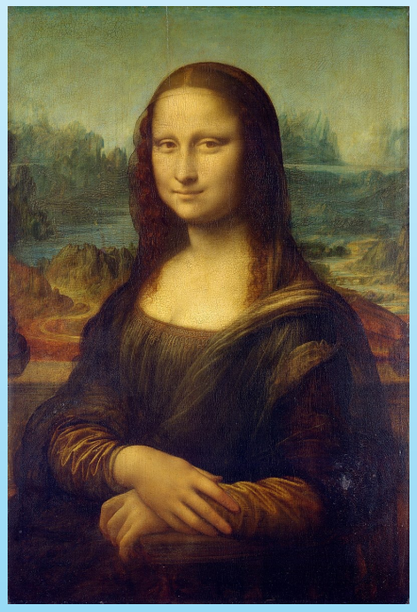
id, money

recurse

ended phase

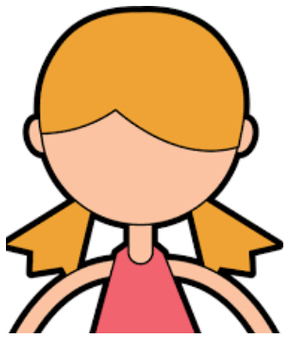


id



Auction Protocol

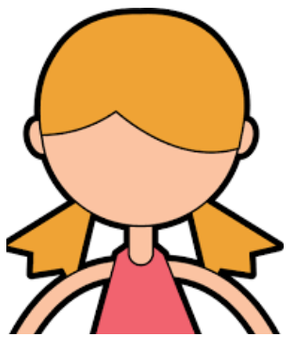
bidding phase



id, money

recurse

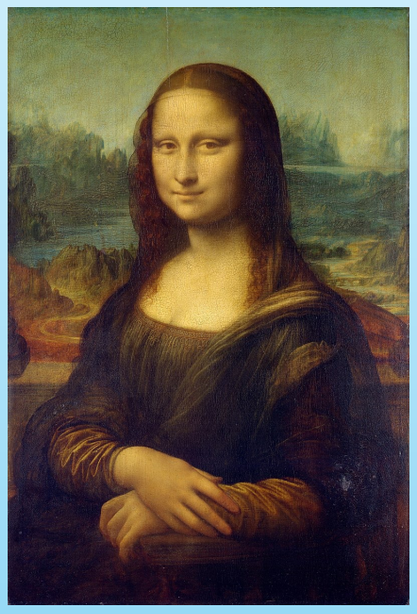
ended phase



id

recurse

monalisa / money



Auction as a Session Type

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

Auction as a Session Type

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

Auction is the Provider | Bidder is the Client

Auction as a Session Type

sends status of auction

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

Auction is the Provider | Bidder is the Client

Auction as a Session Type

sends status of auction

offers choice of bidding

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

Auction is the Provider | Bidder is the Client

Auction as a Session Type

sends status of auction

offers choice of bidding

receive id and money

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

Auction is the Provider | Bidder is the Client

Auction as a Session Type

sends status of auction

offers choice of bidding

receive id and money

recurse

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

Auction is the Provider | Bidder is the Client

Auction as a Session Type

sends status of auction

offers choice of bidding

receive id and money

recurse

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

offers choice to collect

Auction is the Provider | Bidder is the Client

Auction as a Session Type

sends status of auction

offers choice of bidding

receive id and money

recurse

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

offers choice to collect

sends result of bidding

Auction is the Provider | Bidder is the Client

Auction as a Session Type

sends status of auction

offers choice of bidding

receive id and money

recurse

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

offers choice to collect

sends result of bidding

send Mona Lisa

Auction is the Provider | Bidder is the Client

Auction as a Session Type

sends status of auction

offers choice of bidding

receive id and money

recurse

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

offers choice to collect

sends result of bidding

send Mona Lisa

send back bid

Auction is the Provider | Bidder is the Client

Talk Outline

Talk Outline

Session Types

Talk Outline

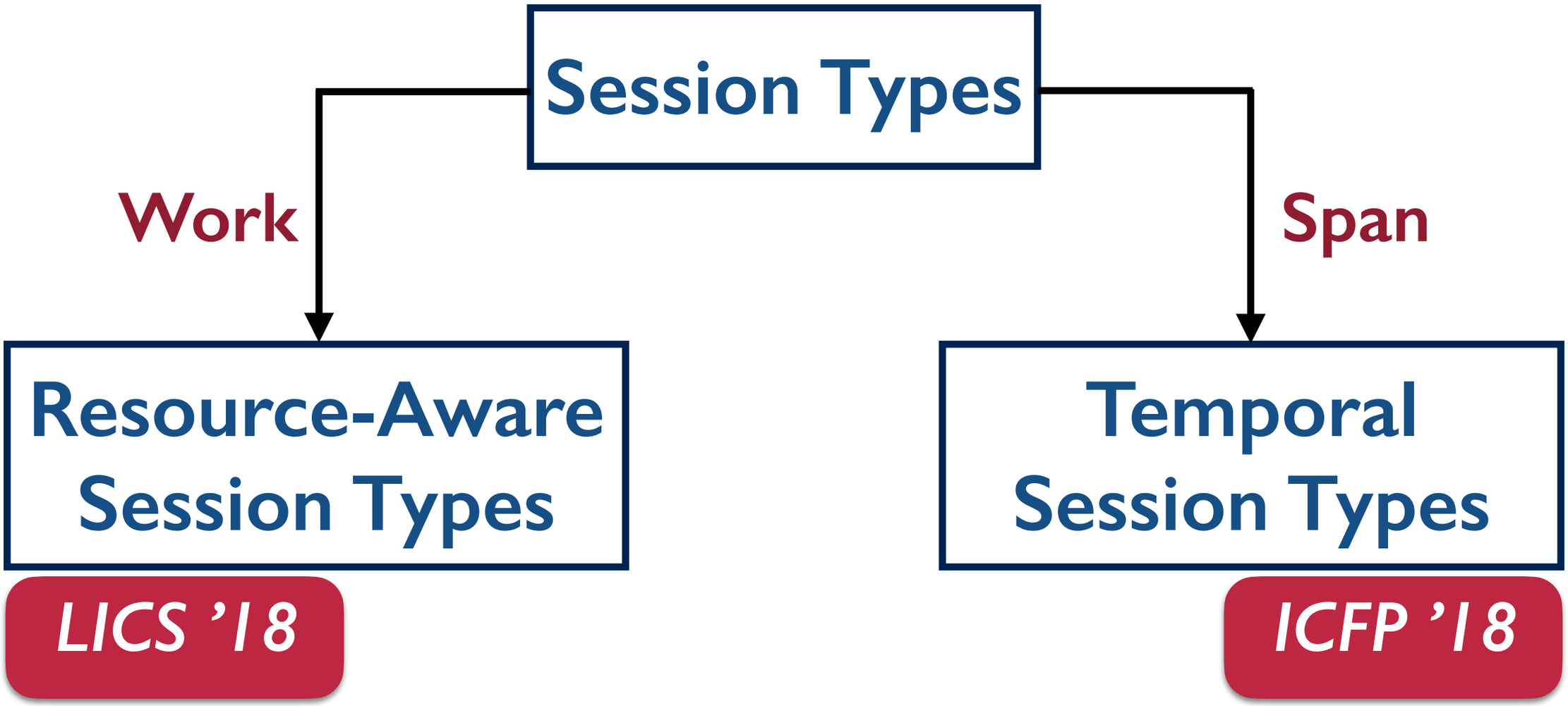
Session Types

Work

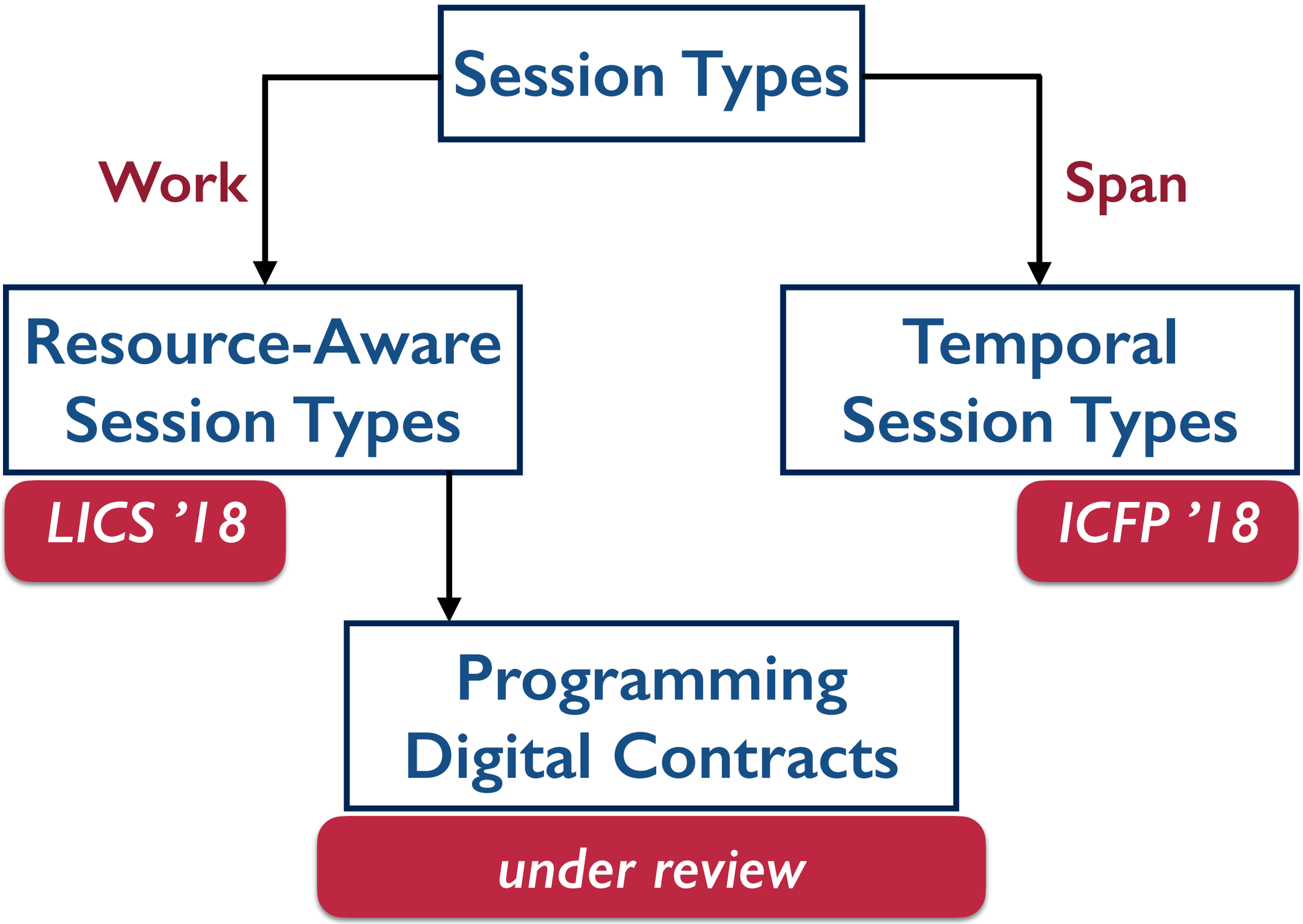
Resource-Aware Session Types

LICS '18

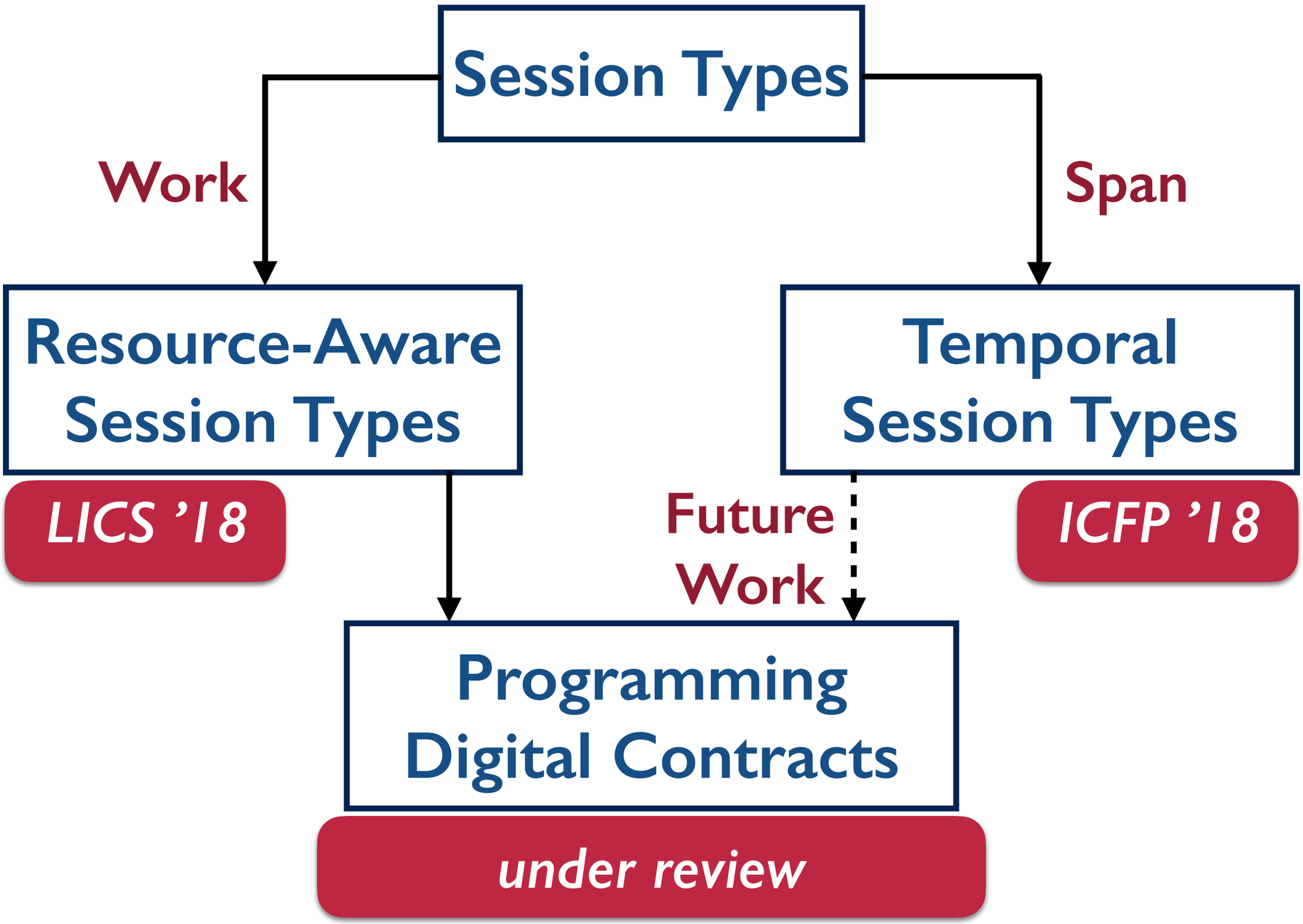
Talk Outline



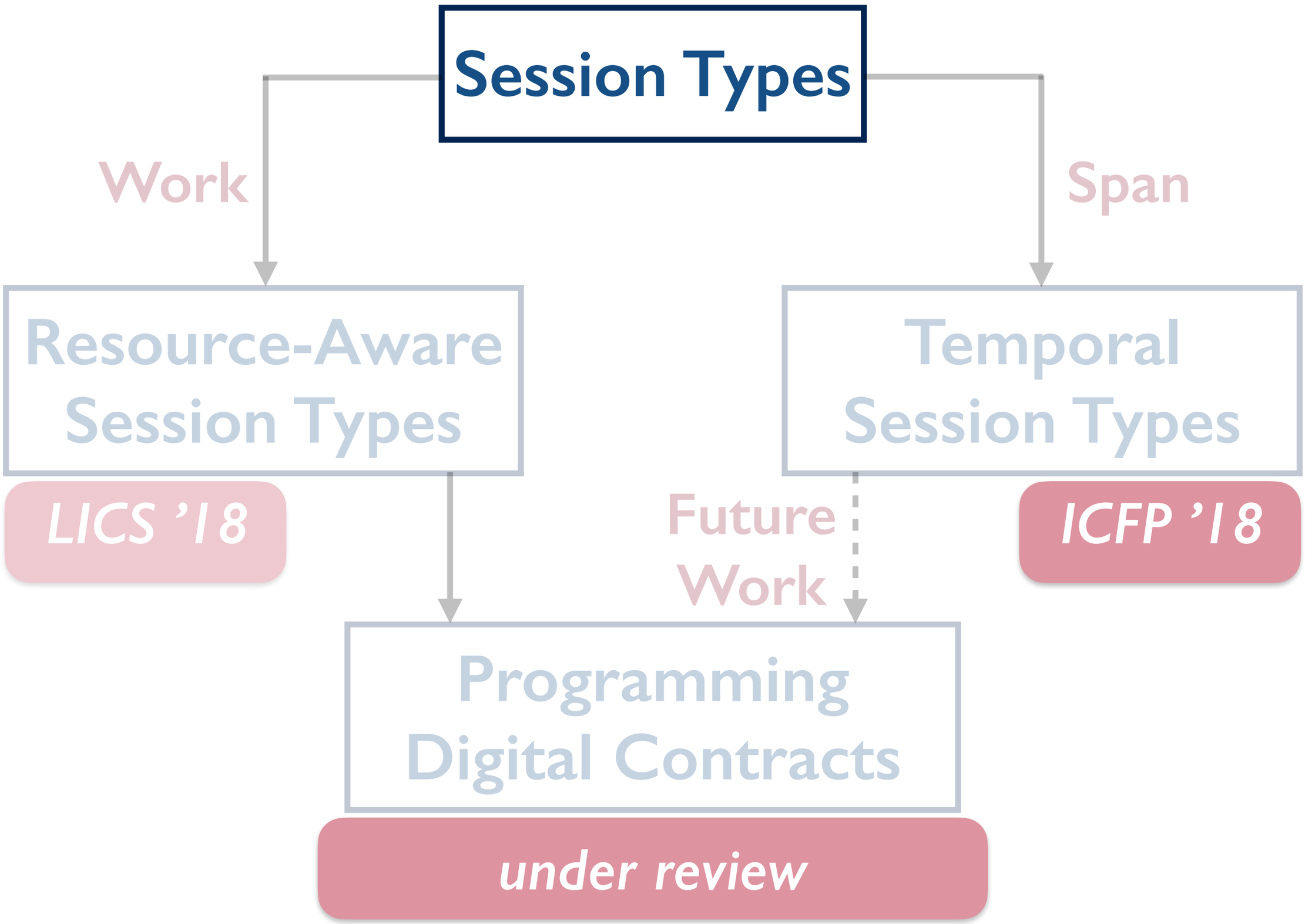
Talk Outline



Talk Outline



Talk Outline



What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Communication protocol enforced by session types

What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Communication protocol enforced by session types

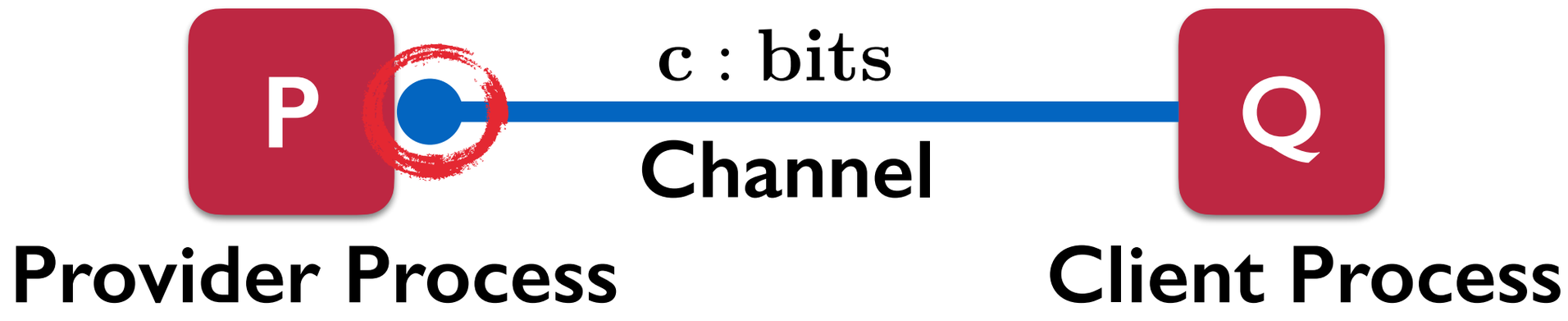
What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Communication protocol enforced by session types

What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Communication protocol enforced by session types

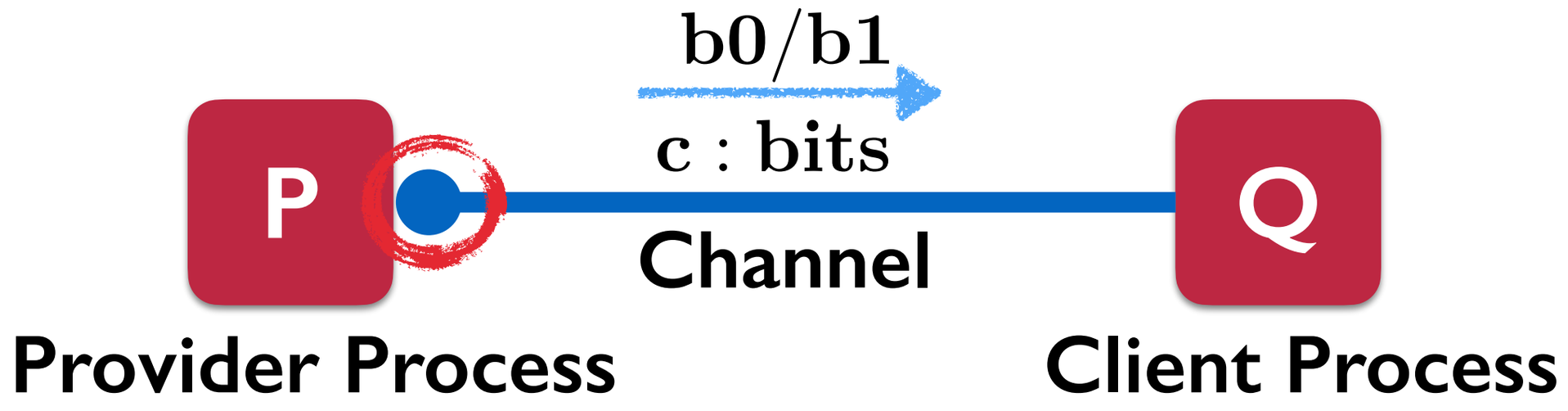
$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}\}$$



What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Communication protocol enforced by session types

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}\}$$



Example: Queues

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},$$
$$\mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

Example: Queues

offers choice of ins/del

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

Example: Queues

offers choice of ins/del

recv element of type A

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

Example: Queues

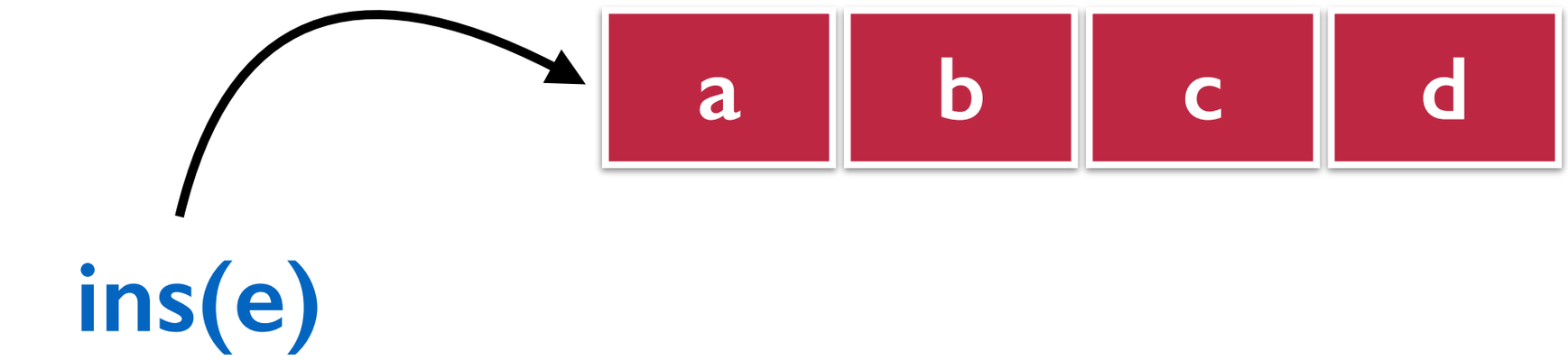
offers choice of ins/del

recv element of type A

behave as queue again

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

Example: Queues



offers choice of ins/del

recv element of type A

behave as queue again

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

Example: Queues

offers choice of ins/del

recv element of type A

behave as queue again

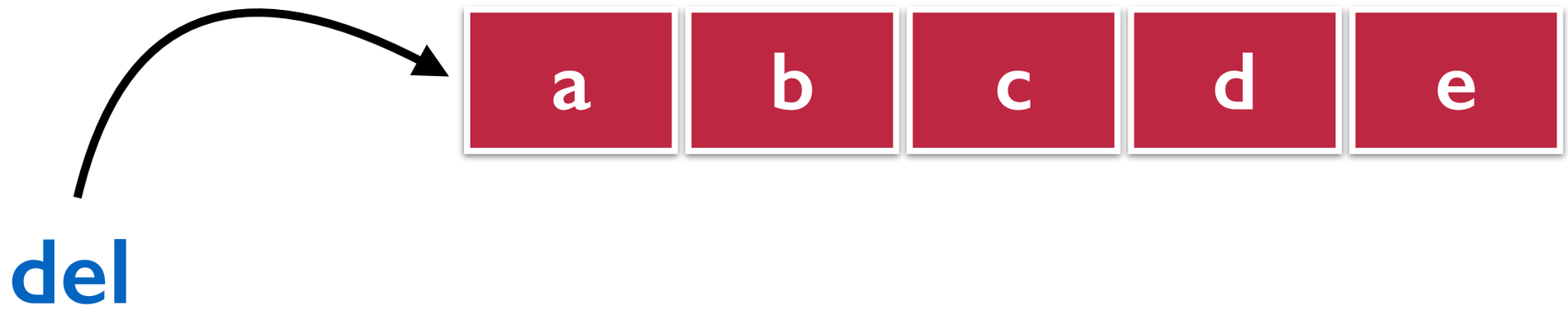
$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

Example: Queues

offers choice of ins/del

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

Example: Queues

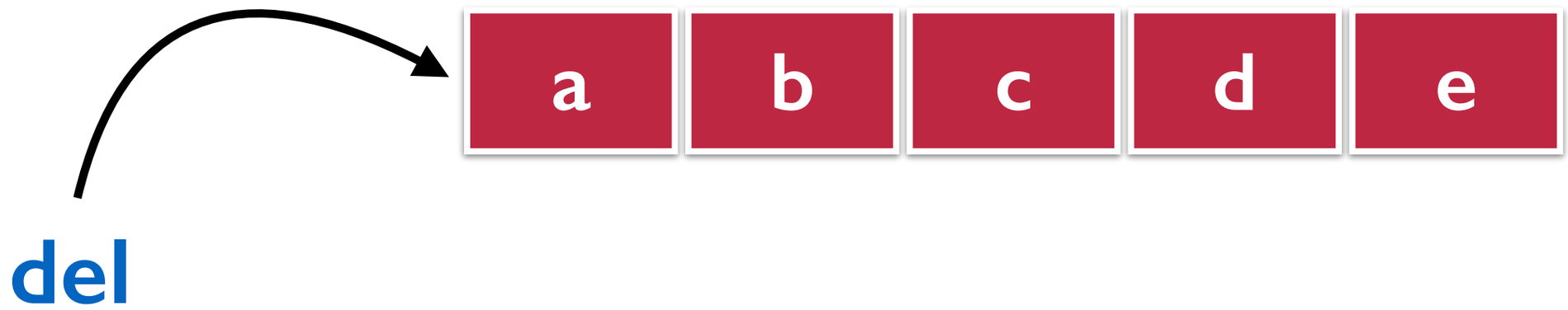


offers choice of ins/del

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

send none if queue is empty

Example: Queues



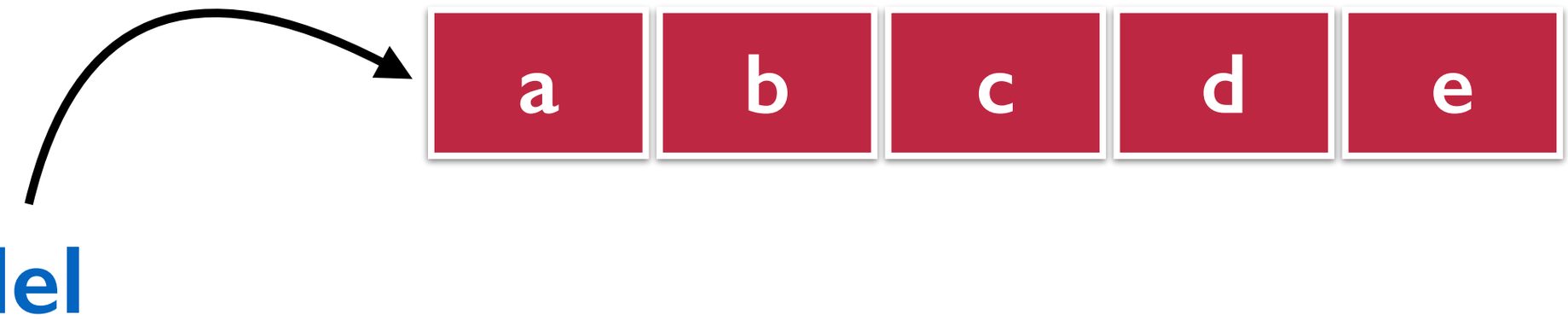
offers choice of ins/del

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

terminate

send none if queue is empty

Example: Queues

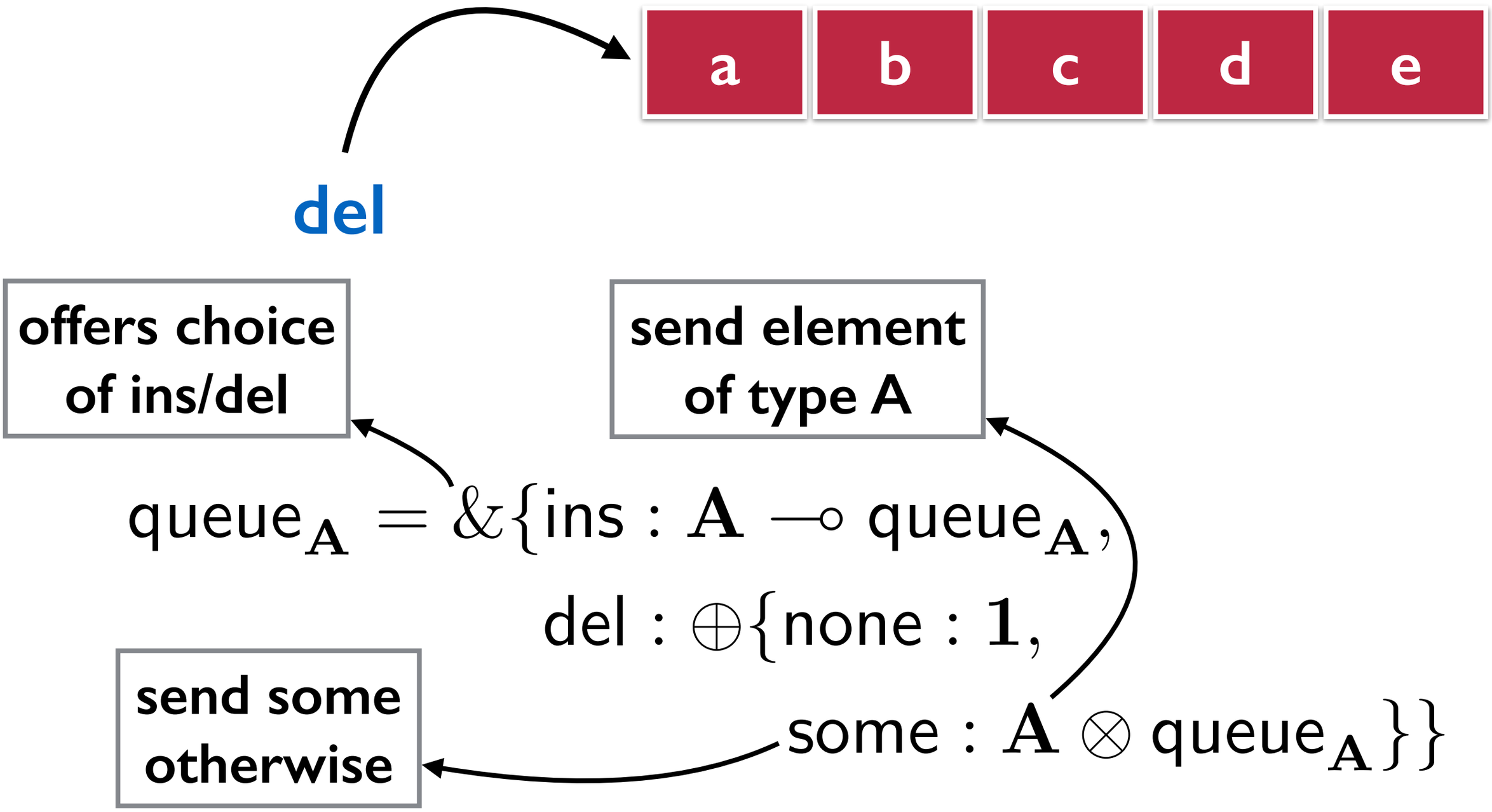


offers choice of ins/del

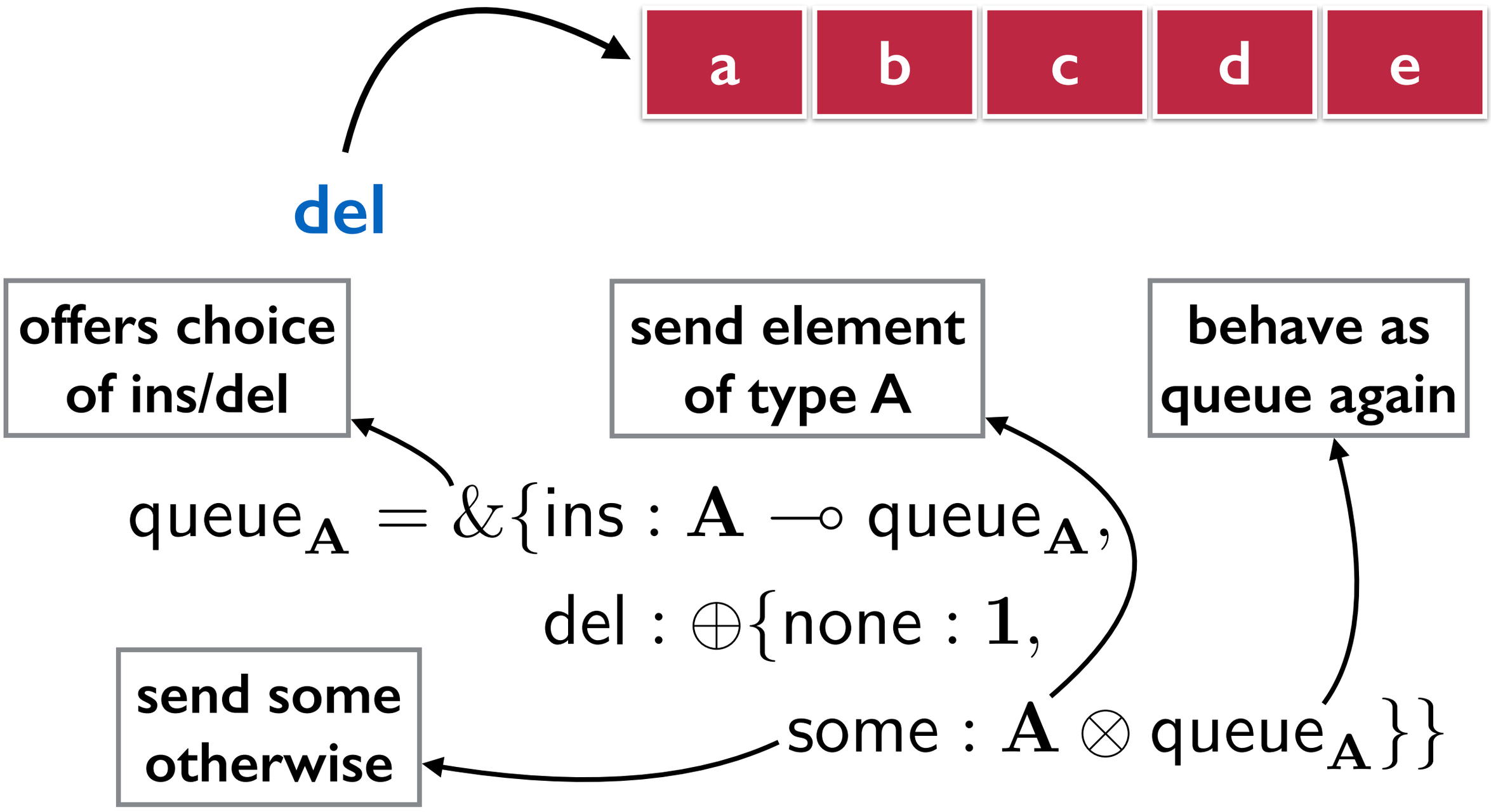
send some otherwise

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

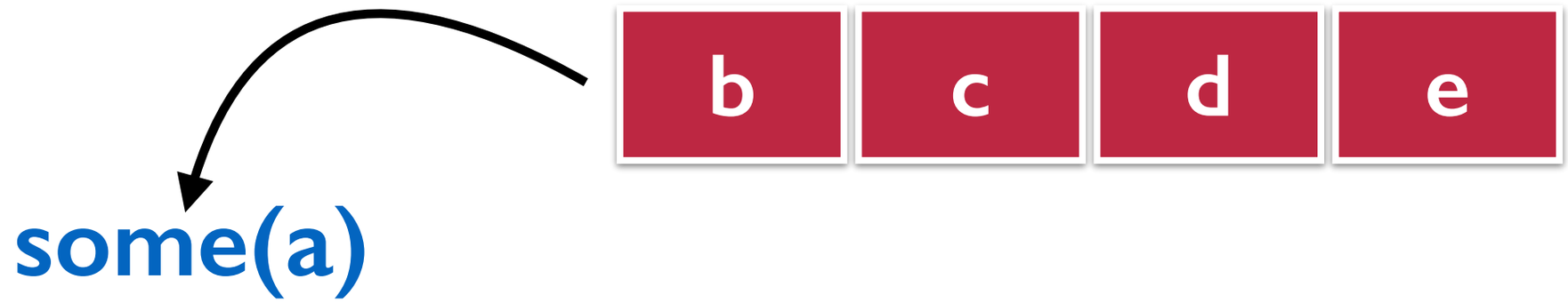
Example: Queues



Example: Queues



Example: Queues



offers choice of ins/del

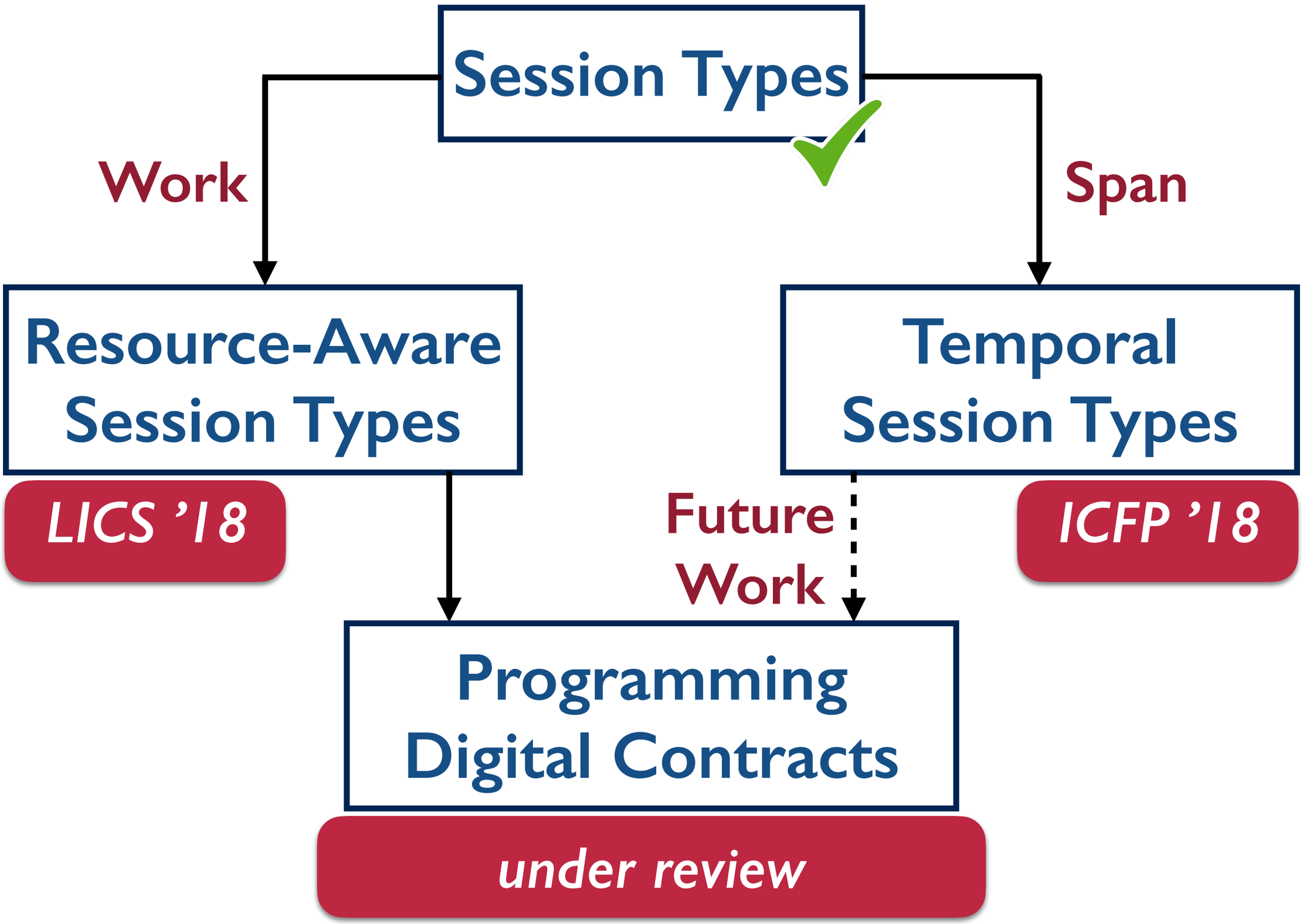
send element of type A

behave as queue again

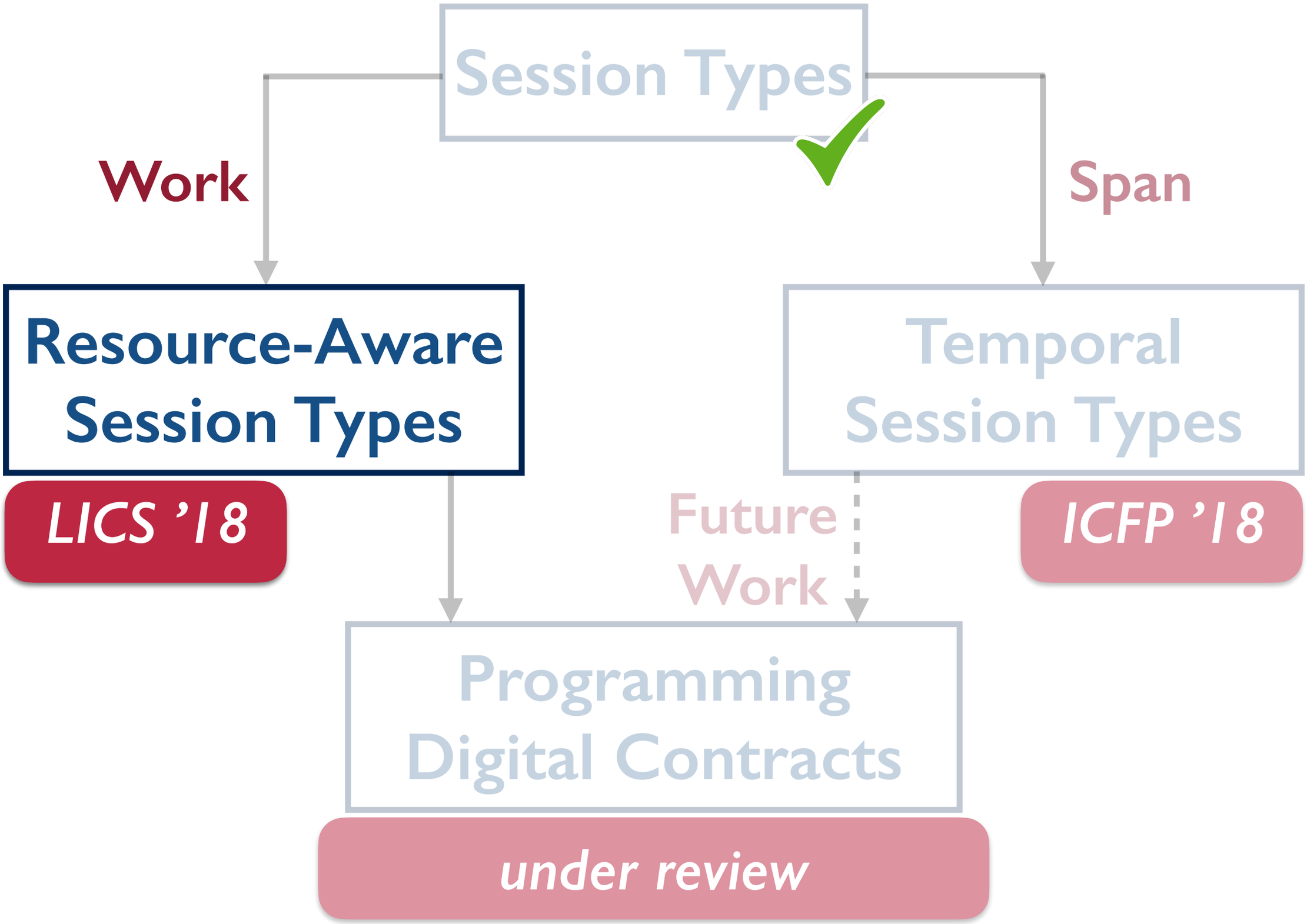
$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},$$
$$\text{del} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

send some otherwise

Talk Outline



Talk Outline



Resource Analysis

Concurrent Programs

Resource Analysis

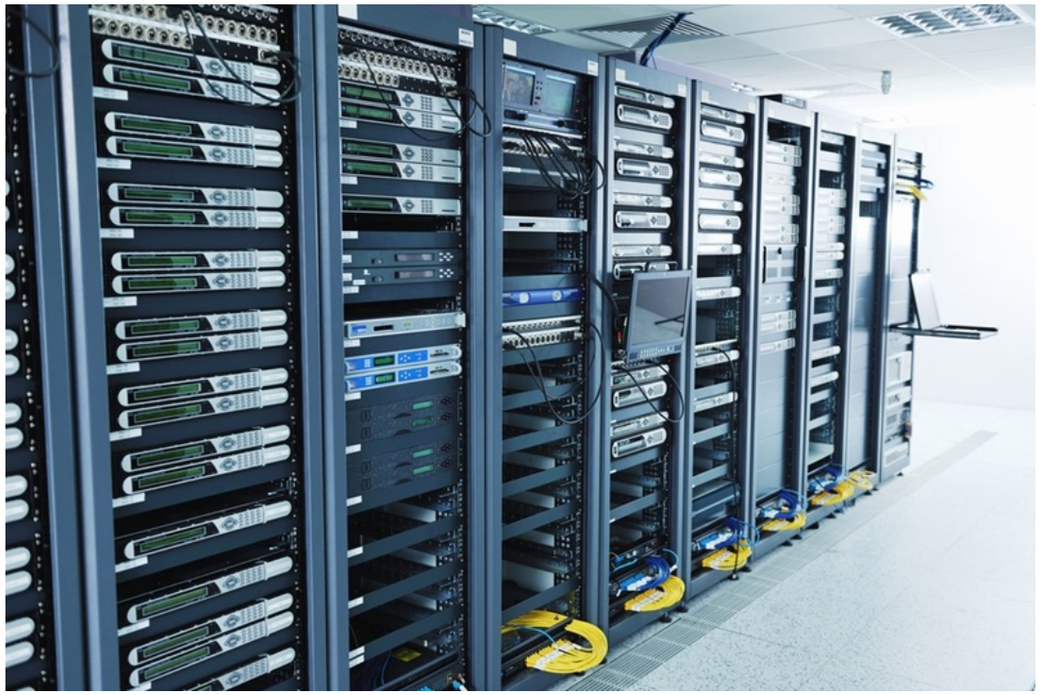
Concurrent Programs

Work
Sequential Complexity

Execution time
on one processor

Resource Analysis

Concurrent Programs



Work
Sequential Complexity

Execution time on one processor

Span
Parallel Complexity

Execution time on arbitrarily many processors

Resource Analysis

Concurrent Programs

Work
Sequential Complexity

Execution time
on one processor

Span
Parallel Complexity

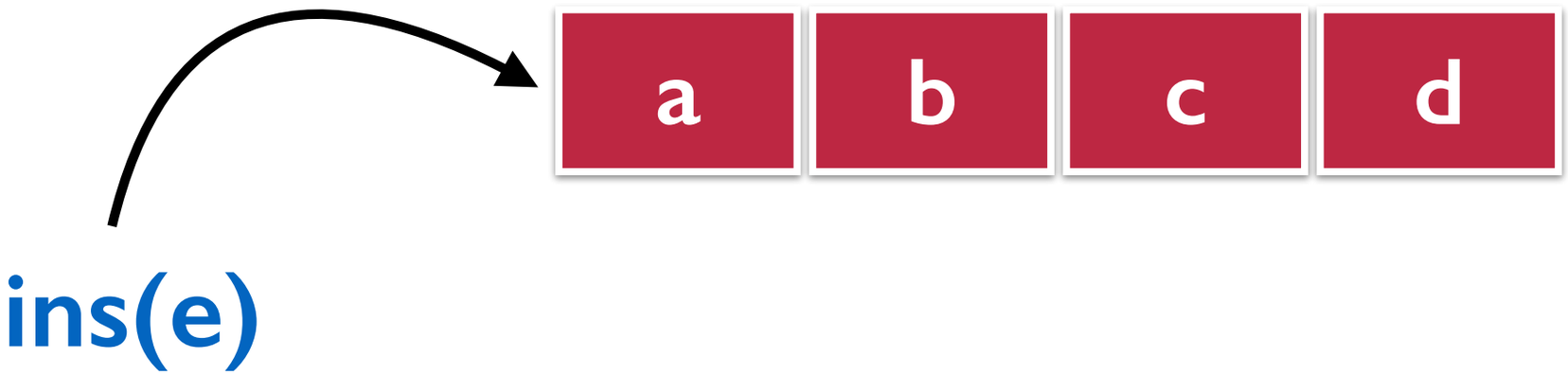
Execution time on
arbitrarily many processors

Work done by Queue

Count the total number of messages!

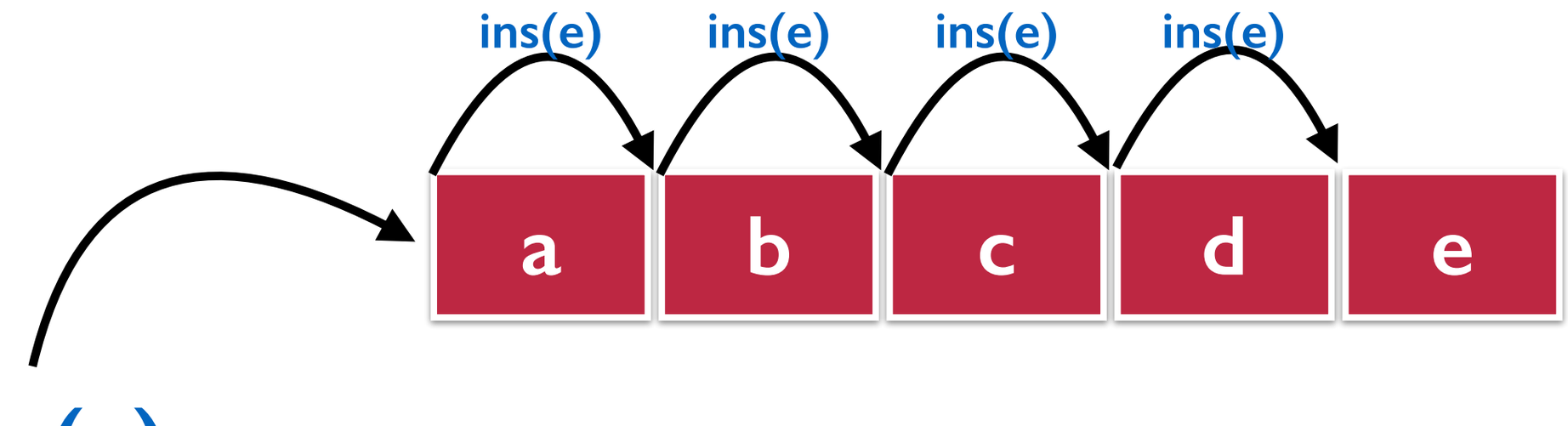
Work done by Queue

Count the total number of messages!



Work done by Queue

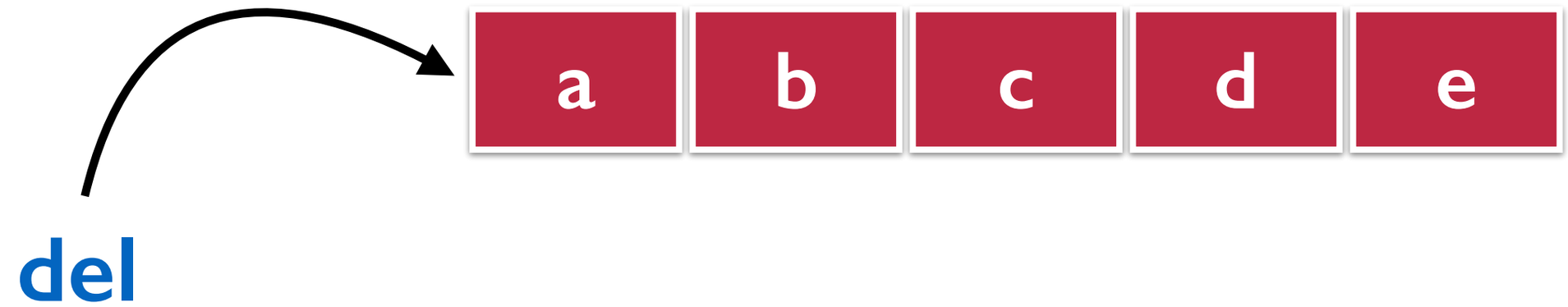
Count the total number of messages!



w_i = Work done to process insertion
= 2n (n is the size of queue)
= ‘ins’ and ‘e’ travel to end of queue

Work done by Queue

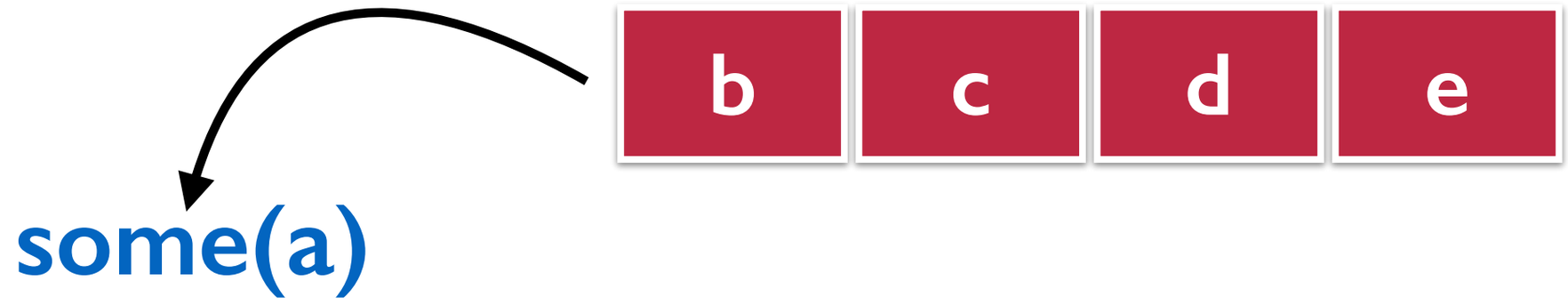
Count the total number of messages!



w_i = Work done to process insertion
= 2n (n is the size of queue)
= ‘ins’ and ‘e’ travel to end of queue

Work done by Queue

Count the total number of messages!



w_i = Work done to process insertion
= 2n (n is the size of queue)
= ‘ins’ and ‘e’ travel to end of queue

w_d = Work done to process deletion
= 2 (sends back ‘some’ and ‘a’)

Potential Method

- Processes store potential

- Potential is exchanged via messages

- Potential is consumed to perform ‘work’

Potential Method

- Processes store potential

only at type level
not needed at runtime

- Potential is exchanged via messages

- Potential is consumed to perform ‘work’

Potential Method

- Processes store potential

only at type level
not needed at runtime

- Potential is exchanged via messages

User defined cost model
This talk: number of messages

- Potential is consumed to perform ‘work’

Queue Type

$$\mathbf{queue_A}[\mathbf{n}] = \&\{\mathsf{ins} : \triangleleft^{\mathbf{2n}}(\mathbf{A} \multimap \mathbf{queue_A}[\mathbf{n+1}]),$$
$$\mathsf{del} : \triangleleft^{\mathbf{2}} \oplus \{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathbf{queue_A}[\mathbf{n-1}]\}\}$$

Queue Type

$$\mathbf{queue_A[n]} = \&\{\mathsf{ins} : \triangleleft^{\mathbf{2n}}(\mathbf{A} \multimap \mathbf{queue_A[n+1]}),$$
$$\mathsf{del} : \triangleleft^{\mathbf{2}} \oplus \{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathbf{queue_A[n-1]}\}\}$$

Index Refinement (Size of Queue)

Queue Type

$$\mathbf{queue_A[n]} = \&\{\mathsf{ins} : \triangleleft^{\mathbf{2n}}(\mathbf{A} \multimap \mathbf{queue_A[n+1]}),$$
$$\mathsf{del} : \triangleleft^{\mathbf{2}} \oplus \{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathbf{queue_A[n-1]}\}\}$$

Index Refinement (Size of Queue)

Potential Annotation

Queue Type

$$\mathbf{queue_A}[\mathbf{n}] = \&\{\mathsf{ins} : \triangleleft^{\mathbf{2n}}(\mathbf{A} \multimap \mathbf{queue_A}[\mathbf{n}+\mathbf{1}]),$$
$$\mathsf{del} : \triangleleft^{\mathbf{2}} \oplus \{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathbf{queue_A}[\mathbf{n}-\mathbf{1}]\}\}$$

Index Refinement (Size of Queue)

Potential Annotation

- receive 2n units of potential after ‘ins’
- receive 2 units of potential after ‘del’
- potential is consumed to exchange messages

Stacks vs Queues

$$\mathbf{stack_A[n]} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathbf{stack_A[n+1]},$$
$$\mathsf{del} : \triangleleft^{\mathbf{2}} \oplus \{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathbf{stack_A[n-1]}\}\}$$

$$\mathbf{queue_A[n]} = \&\{\mathsf{ins} : \triangleleft^{\mathbf{2n}}(\mathbf{A} \multimap \mathbf{queue_A[n+1]}),$$
$$\mathsf{del} : \triangleleft^{\mathbf{2}} \oplus \{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathbf{queue_A[n-1]}\}\}$$

Which one's more efficient?

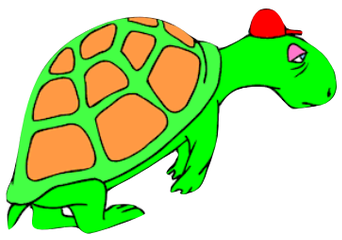
Stacks vs Queues

$$\mathbf{stack_A}[\mathbf{n}] = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathbf{stack_A}[\mathbf{n+1}],$$

$$\mathsf{del} : \triangleleft^{\mathbf{2}} \oplus \{\mathsf{none} : \mathbf{1},$$

$$\mathsf{some} : \mathbf{A} \otimes \mathbf{stack_A}[\mathbf{n-1}]\}\}$$

$$\mathbf{queue_A}[\mathbf{n}] = \&\{\mathsf{ins} : \triangleleft^{\mathbf{2n}}(\mathbf{A} \multimap \mathbf{queue_A}[\mathbf{n+1}]),$$

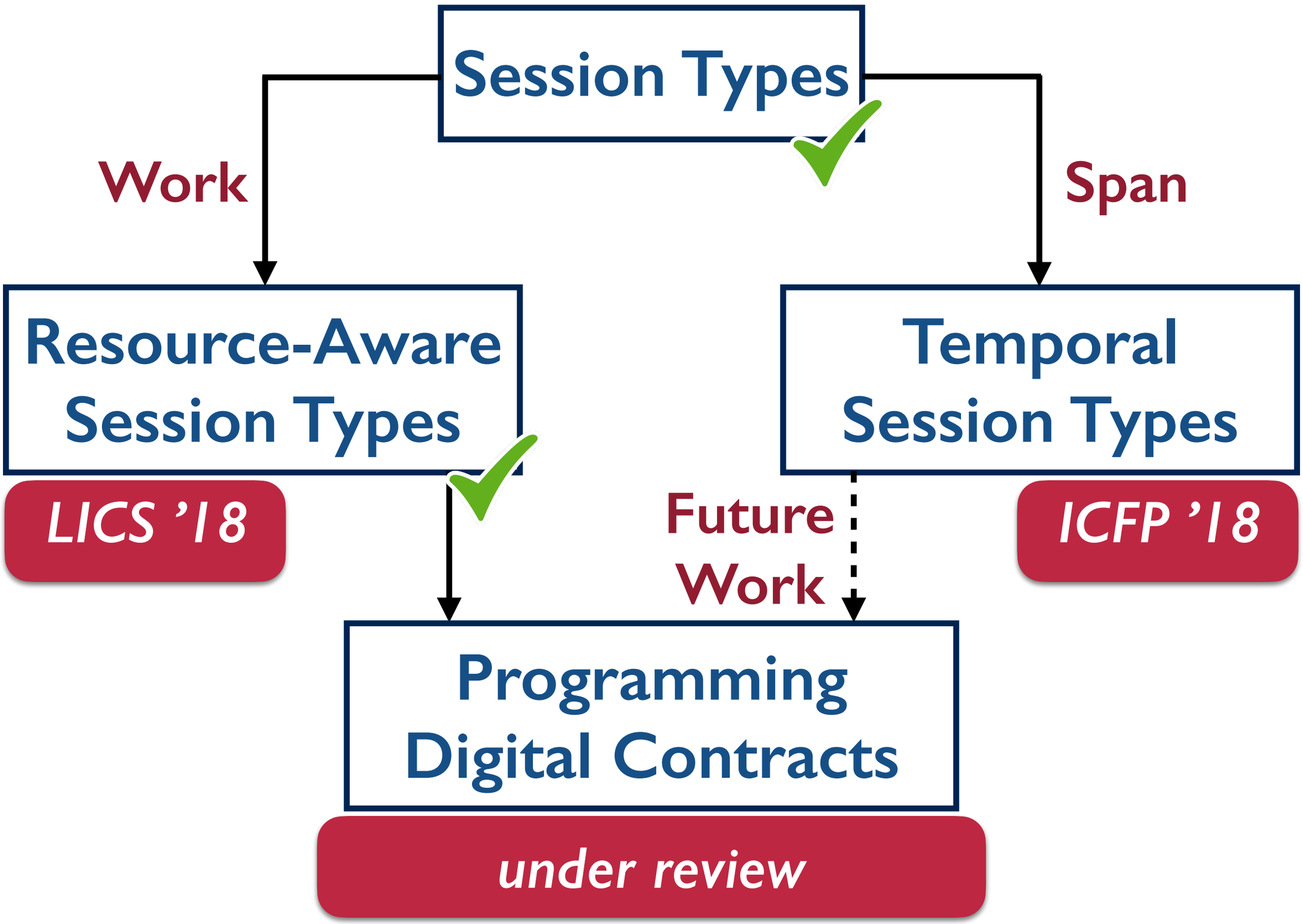


$$\mathsf{del} : \triangleleft^{\mathbf{2}} \oplus \{\mathsf{none} : \mathbf{1},$$

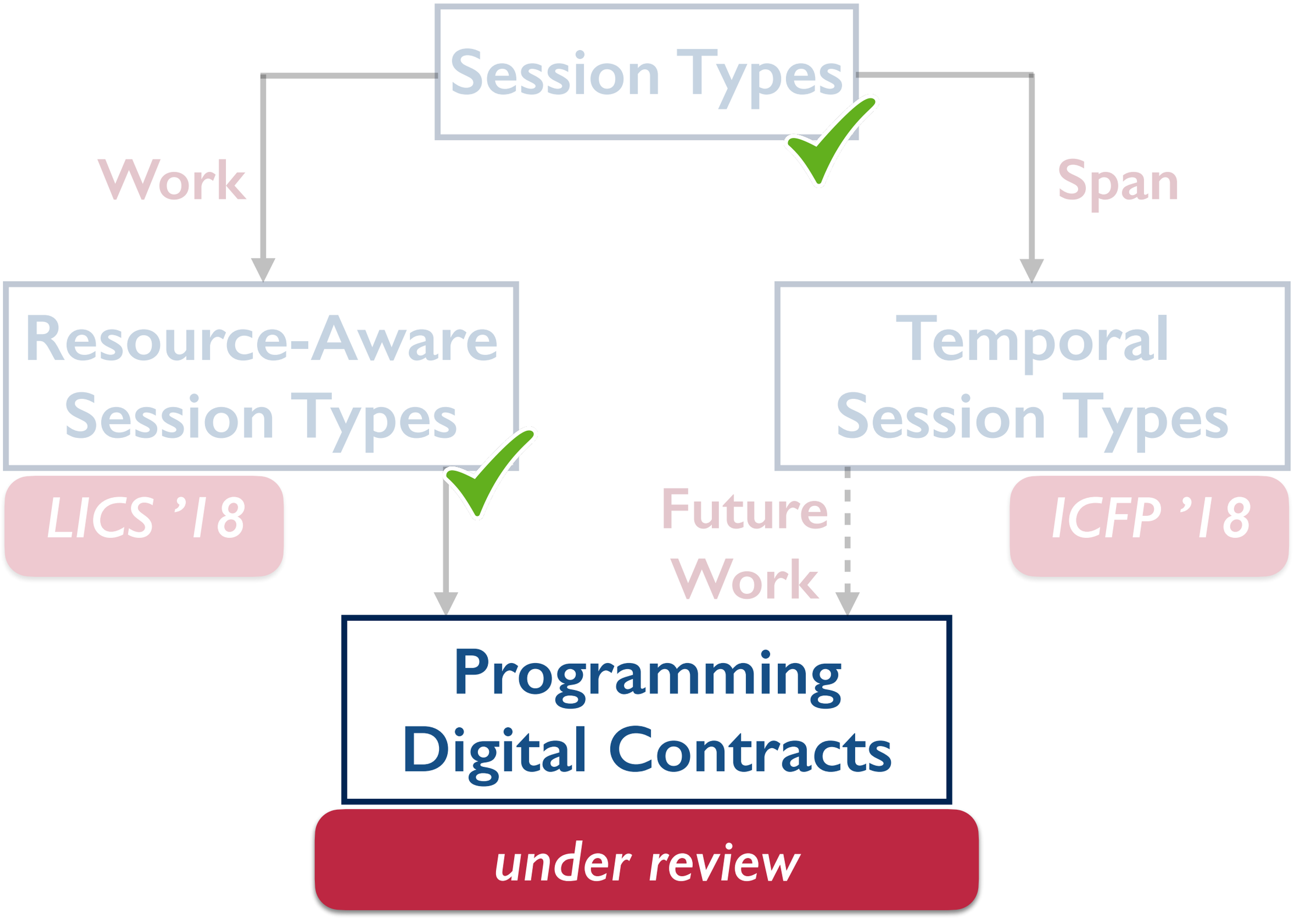
$$\mathsf{some} : \mathbf{A} \otimes \mathbf{queue_A}[\mathbf{n-1}]\}\}$$

Which one's more efficient?

Talk Outline



Talk Outline



Limitations of Session Types

Limitations of Session Types

Two Key Challenges

Limitations of Session Types

Two Key Challenges

Channels are linear, no sharing!

- **Auction can have only one bidder!**
- **To incorporate multiple bidders, channels need to be shared**

Limitations of Session Types

Two Key Challenges

Channels are linear, no sharing!

- Auction can have only one bidder!
- To incorporate multiple bidders, channels need to be shared

No functional layer, no state!

- Auction cannot store list of players, mapping of players to bids, etc.
- Needs integration with a functional language

Limitations of Session Types

Two Key Challenges

Channels are linear, no sharing!

- Auction can have only one bidder!
- To incorporate multiple bidders, channels need to be shared

No functional layer, no state!

- Auction cannot store list of players, mapping of players to bids, etc.
- Needs integration with a functional language

Explored in prior work, but never combined!

Shared Channels

Balzer and Pfenning, ICFP 2017

- Types stratified into linear and shared layers
- Modal operators connecting the layers

Shared Channels

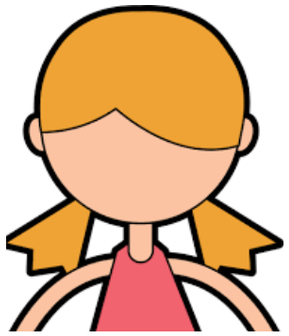
Balzer and Pfenning, ICFP 2017

- Types stratified into linear and shared layers
- Modal operators connecting the layers

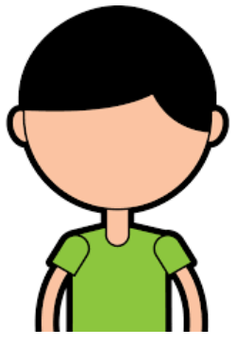
$\uparrow^{\mathsf{S}}_{\mathsf{L}} \mathbf{A_L} \Rightarrow$ **Shifts a linear type to shared**

$\downarrow^{\mathsf{S}}_{\mathsf{L}} \mathbf{A_S} \Rightarrow$ **Shifts a shared type to linear**

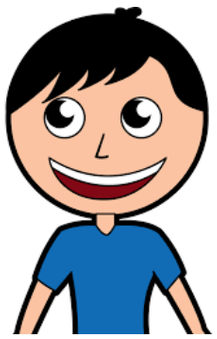
Shared Auction



Bidder 1

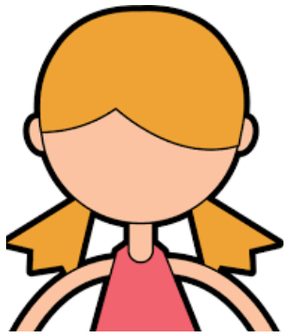


Bidder 2



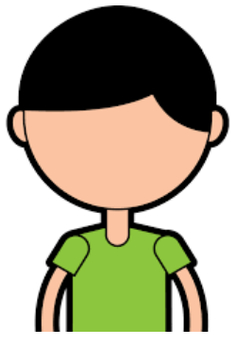
Bidder 3

Shared Auction

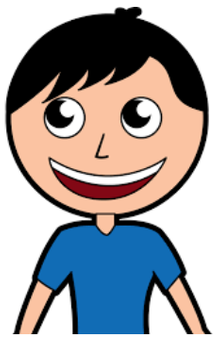


acquire!

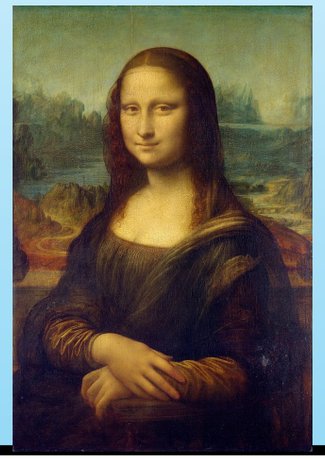
Bidder 1



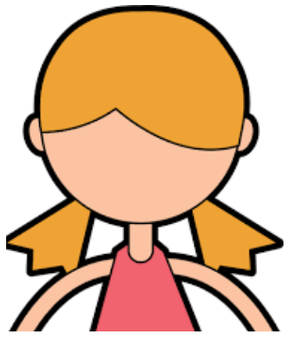
Bidder 2



Bidder 3

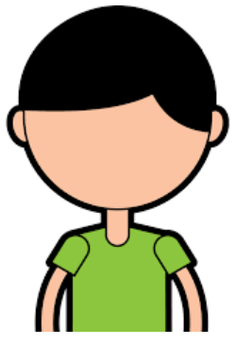


Shared Auction

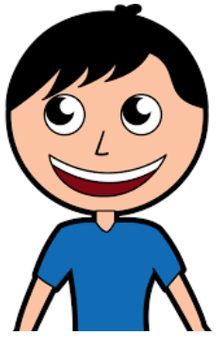


acquire!

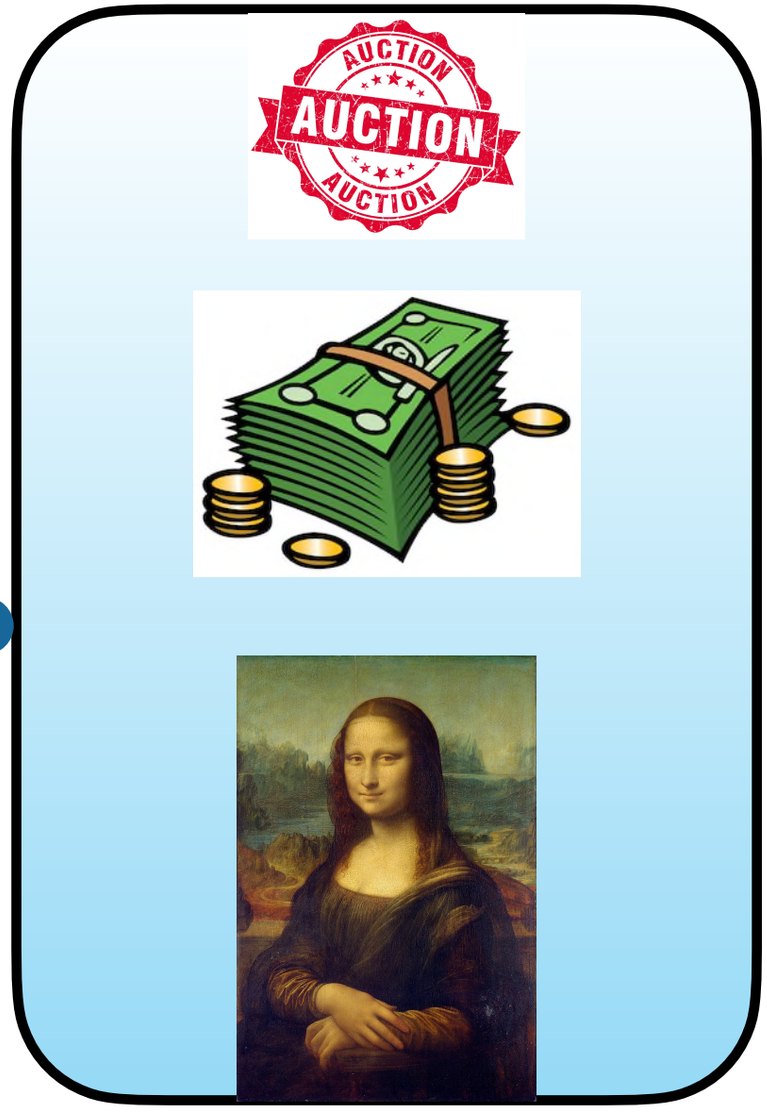
Bidder 1



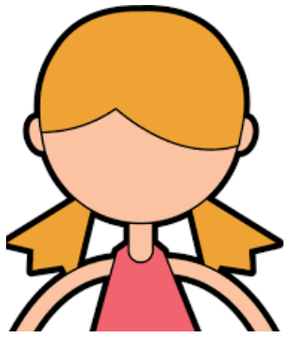
Bidder 2



Bidder 3

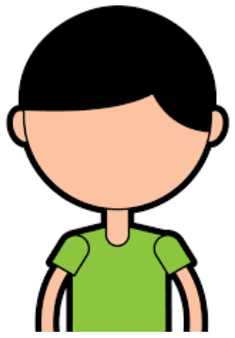


Shared Auction

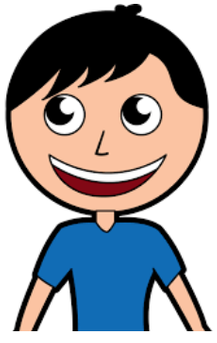


release!

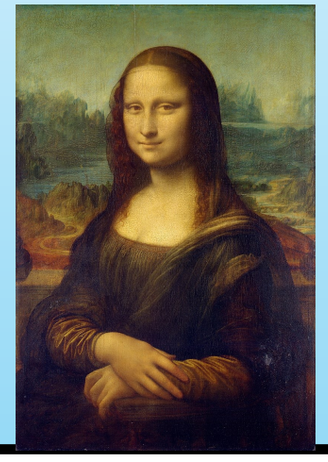
Bidder 1



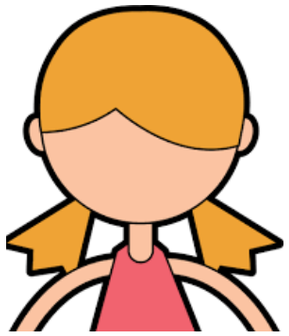
Bidder 2



Bidder 3

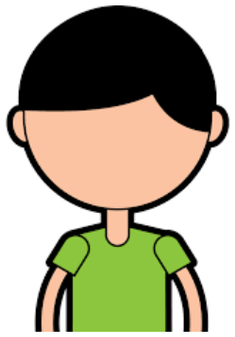


Shared Auction

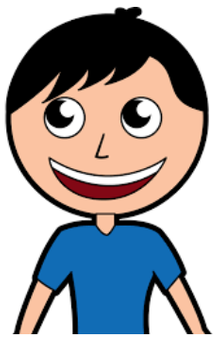


release!

Bidder 1

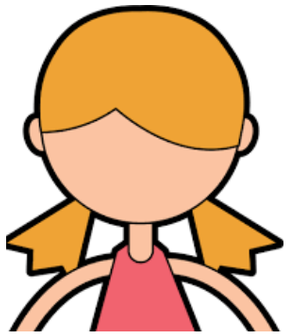


Bidder 2



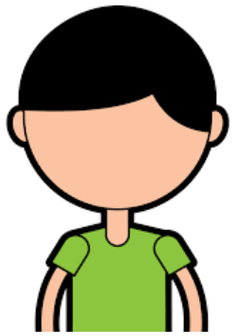
Bidder 3

Shared Auction

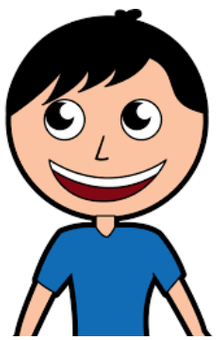


release!

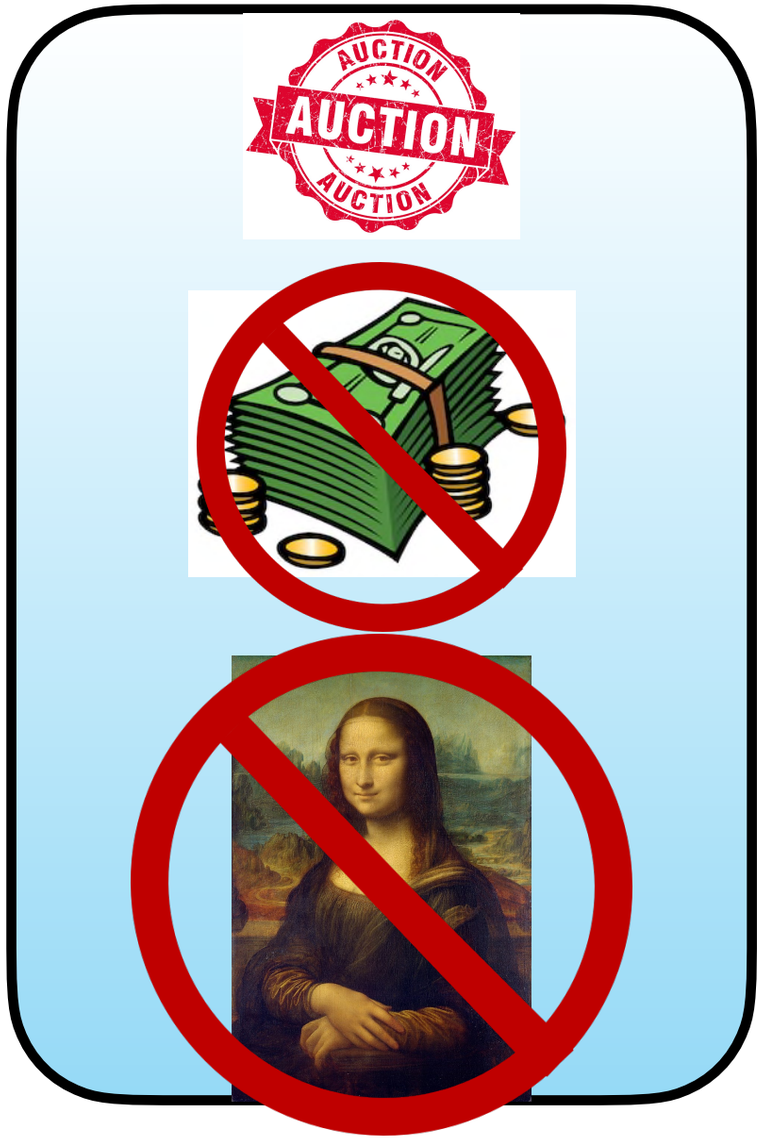
Bidder 1



Bidder 2



Bidder 3



Limitation (Independence Principle): shared process cannot store any linear assets (no money in the auction contract)

Shared Auction

release!

Bidder 1

Bidder 2

Bidder 3

Prevents Re-entrancy Attacks

Limitation (Independence Principle): shared process cannot store any linear assets (no money in the auction contract)

Functional Layer

Toninho et. al., ESOP 2013

- Integrate session types in a functional programming language via a linear contextual monad
- Functional data structures isolated in a separate context in the typing judgment
- In my case: integration with Resource-Aware ML (Hoffmann, Das and Weng, POPL '17)

Shared Auction Type

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{*} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{*} \downarrow^{S}_{L} \text{auction}\},$$
$$\textbf{ended} : \&\{\textbf{collect} : \text{id} \to \oplus\{\textbf{won} : \text{monalisa} \otimes \downarrow^{S}_{L} \text{auction},$$
$$\textbf{lost} : \text{money} \otimes \triangleright^{*} \downarrow^{S}_{L} \text{auction}\}\}\}$$

*Type checker fills in * annotations automatically*

Shared Auction Type

$$\begin{aligned}\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{&\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},\\ &\textbf{ended} : \&\{\textbf{collect} : \text{id} \to \oplus\{\textbf{won} : \text{monalisa} \otimes \downarrow^{S}_{L} \text{auction},\\ &\qquad\qquad \textbf{lost} : \text{money} \otimes \triangleright^{2} \downarrow^{S}_{L} \text{auction}\}\}\}\end{aligned}$$

*Type checker fills in * annotations automatically*

Shared Auction Type

shared: contract is acquired before use

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$
$$\textbf{ended} : \&\{\textbf{collect} : \text{id} \to \oplus\{\textbf{won} : \text{monalisa} \otimes \downarrow^{S}_{L} \text{auction},$$
$$\textbf{lost} : \text{money} \otimes \triangleright^{2} \downarrow^{S}_{L} \text{auction}\}\}\}$$

*Type checker fills in * annotations automatically*

Shared Auction Type

shared: contract is acquired before use

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$
$$\textbf{ended} : \&\{\textbf{collect} : \text{id} \to \oplus\{\textbf{won} : \text{monalisa} \otimes \downarrow^{S}_{L} \text{auction},$$
$$\textbf{lost} : \text{money} \otimes \triangleright^{2} \downarrow^{S}_{L} \text{auction}\}\}\}$$

receive 22 units of potential

*Type checker fills in * annotations automatically*

Shared Auction Type

shared: contract is acquired before use

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$
$$\textbf{ended} : \&\{\textbf{collect} : \text{id} \to \oplus\{\textbf{won} : \text{monalisa} \otimes \downarrow^{S}_{L} \text{auction},$$
$$\textbf{lost} : \text{money} \otimes \triangleright^{2} \downarrow^{S}_{L} \text{auction}\}\}\}$$

receive 22 units of potential

send back 7 units of potential

*Type checker fills in * annotations automatically*

Shared Auction Type

shared: contract is acquired before use

shared: contract is released after use

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$
$$\textbf{ended} : \&\{\textbf{collect} : \text{id} \to \oplus\{\textbf{won} : \text{monalisa} \otimes \downarrow^{S}_{L} \text{auction},$$
$$\textbf{lost} : \text{money} \otimes \triangleright^{2} \downarrow^{S}_{L} \text{auction}\}\}\}$$

receive 22 units of potential

send back 7 units of potential

*Type checker fills in * annotations automatically*

Running Auction

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$

$(b : \text{bids})\ ;\ (M : \text{money}), (ml : \text{monalisa}) \vdash \textit{run} :: (sa : \text{auction})$

$sa \leftarrow \textit{run}\ b \leftarrow M\ l =$

$la \leftarrow \text{accept}\ sa\ ;$

$la.\text{running}\ ;$

$\text{case}\ la$

$(\text{bid} \Rightarrow r \leftarrow \text{recv}\ la\ ;$

$m \leftarrow \text{recv}\ la\ ;$

$sa \leftarrow \text{detach}\ la\ ;$

$m.\text{value}\ ;$

$v \leftarrow \text{recv}\ m\ ;$

$b' = \textit{addbid}\ b\ (r, v)\ ;$

$M' \leftarrow add \leftarrow M\ m\ ;$

$sa \leftarrow \textit{run}\ b' \leftarrow M'\ ml)$

Running Auction

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$

```
(b : bids) ; (M : money), (ml : monalisa) ⊢ run :: (sa : auction)
  sa ← run b ← M l =
    la ← accept sa ;
    la.running ;
    case la
      (bid ⇒ r ← recv la ;
             m ← recv la ;
             sa ← detach la ;
             m.value ;
             v ← recv m ;
             b' = addbid b (r, v) ;
             M' ← add ← M m ;
             sa ← run b' ← M' ml)
```

accept ‘acquire’ ($\uparrow^{S}_{L}$)

Running Auction

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$

$$
\begin{array}{l}
(b : \text{bids}) \ ; \ (M : \text{money}), (ml : \text{monalisa}) \vdash \mathit{run} :: (sa : \text{auction}) \\
\quad sa \leftarrow \mathit{run}\ b \leftarrow M\ l = \\
\qquad la \leftarrow \text{accept}\ sa \ ; \\
\qquad la.\text{running} \ ; \\
\qquad \text{case}\ la \\
\qquad\quad (\text{bid} \Rightarrow r \leftarrow \text{recv}\ la \ ; \\
\qquad\qquad\quad m \leftarrow \text{recv}\ la \ ; \\
\qquad\qquad\quad sa \leftarrow \text{detach}\ la \ ; \\
\qquad\qquad\quad m.\text{value} \ ; \\
\qquad\qquad\quad v \leftarrow \text{recv}\ m \ ; \\
\qquad\qquad\quad b' = \mathit{addbid}\ b\ (r, v) \ ; \\
\qquad\qquad\quad M' \leftarrow \mathit{add} \leftarrow M\ m \ ; \\
\qquad\qquad\quad sa \leftarrow \mathit{run}\ b' \leftarrow M'\ ml)
\end{array}
$$

accept ‘acquire’ ($\uparrow^{S}_{L}$)

send status ‘running’

Running Auction

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$

$$
\begin{array}{l}
(b : \text{bids}) \; ; \; (M : \text{money}), (ml : \text{monalisa}) \vdash run :: (sa : \text{auction}) \\
\quad sa \leftarrow run \; b \leftarrow M \; l = \\
\qquad la \leftarrow \text{accept } sa \; ; \\
\qquad la.\text{running} \; ; \\
\qquad \text{case } la \\
\qquad\quad (\text{bid} \Rightarrow r \leftarrow \text{recv } la \; ; \\
\qquad\qquad\quad m \leftarrow \text{recv } la \; ; \\
\qquad\qquad\quad sa \leftarrow \text{detach } la \; ; \\
\qquad\qquad\quad m.\text{value} \; ; \\
\qquad\qquad\quad v \leftarrow \text{recv } m \; ; \\
\qquad\qquad\quad b' = addbid \; b \; (r, v) \; ; \\
\qquad\qquad\quad M' \leftarrow add \leftarrow M \; m \; ; \\
\qquad\qquad\quad sa \leftarrow run \; b' \leftarrow M' \; ml)
\end{array}
$$

accept ‘acquire’ ($\uparrow^{S}_{L}$)

send status ‘running’

recv ‘id’ and ‘money’

Running Auction

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$

```
(b : bids) ; (M : money), (ml : monalisa) ⊢ run :: (sa : auction)
  sa ← run b ← M l =
    la ← accept sa ;
    la.running ;
    case la
      (bid ⇒ r ← recv la ;
              m ← recv la ;
              sa ← detach la ;
              m.value ;
              v ← recv m ;
              b' = addbid b (r, v) ;
              M' ← add ← M m ;
              sa ← run b' ← M' ml)
```

accept ‘acquire’ ($\uparrow^{S}_{L}$)

send status ‘running’

recv ‘id’ and ‘money’

detach from client ($\downarrow^{S}_{L}$)

Running Auction

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$

$(b : \text{bids})\ ;\ (M : \text{money}), (ml : \text{monalisa}) \vdash run :: (sa : \text{auction})$

$sa \leftarrow run\ b \leftarrow M\ l =$

$la \leftarrow \text{accept}\ sa\ ;$

$la.\text{running}\ ;$

$\text{case}\ la$

$(\text{bid} \Rightarrow r \leftarrow \text{recv}\ la\ ;$

$m \leftarrow \text{recv}\ la\ ;$

$sa \leftarrow \text{detach}\ la\ ;$

$m.\text{value}\ ;$

$v \leftarrow \text{recv}\ m\ ;$

$b' = addbid\ b\ (r, v)\ ;$

$M' \leftarrow add \leftarrow M\ m\ ;$

$sa \leftarrow run\ b' \leftarrow M'\ ml)$

accept ‘acquire’ ($\uparrow^{S}_{L}$)

send status ‘running’

recv ‘id’ and ‘money’

detach from client ($\downarrow^{S}_{L}$)

add bid and money

Running Auction

$$\text{auction} = \uparrow^{S}_{L} \triangleleft^{22} \oplus \{\textbf{running} : \&\{\textbf{bid} : \text{id} \to \text{money} \multimap \triangleright^{7} \downarrow^{S}_{L} \text{auction}\},$$

```
(b : bids) ; (M : money), (ml : monalisa) ⊢ run :: (sa : auction)
  sa ← run b ← M l =
    la ← accept sa ;
    la.running ;
    case la
      (bid ⇒ r ← recv la ;
              m ← recv la ;
              sa ← detach la ;
              m.value ;
              v ← recv m ;
              b' = addbid b (r, v) ;
              M' ← add ← M m ;
              sa ← run b' ← M' ml)
```

accept ‘acquire’ ($\uparrow^{S}_{L}$)

send status ‘running’

recv ‘id’ and ‘money’

detach from client ($\downarrow^{S}_{L}$)

add bid and money

no work constructs!

Programming Contracts

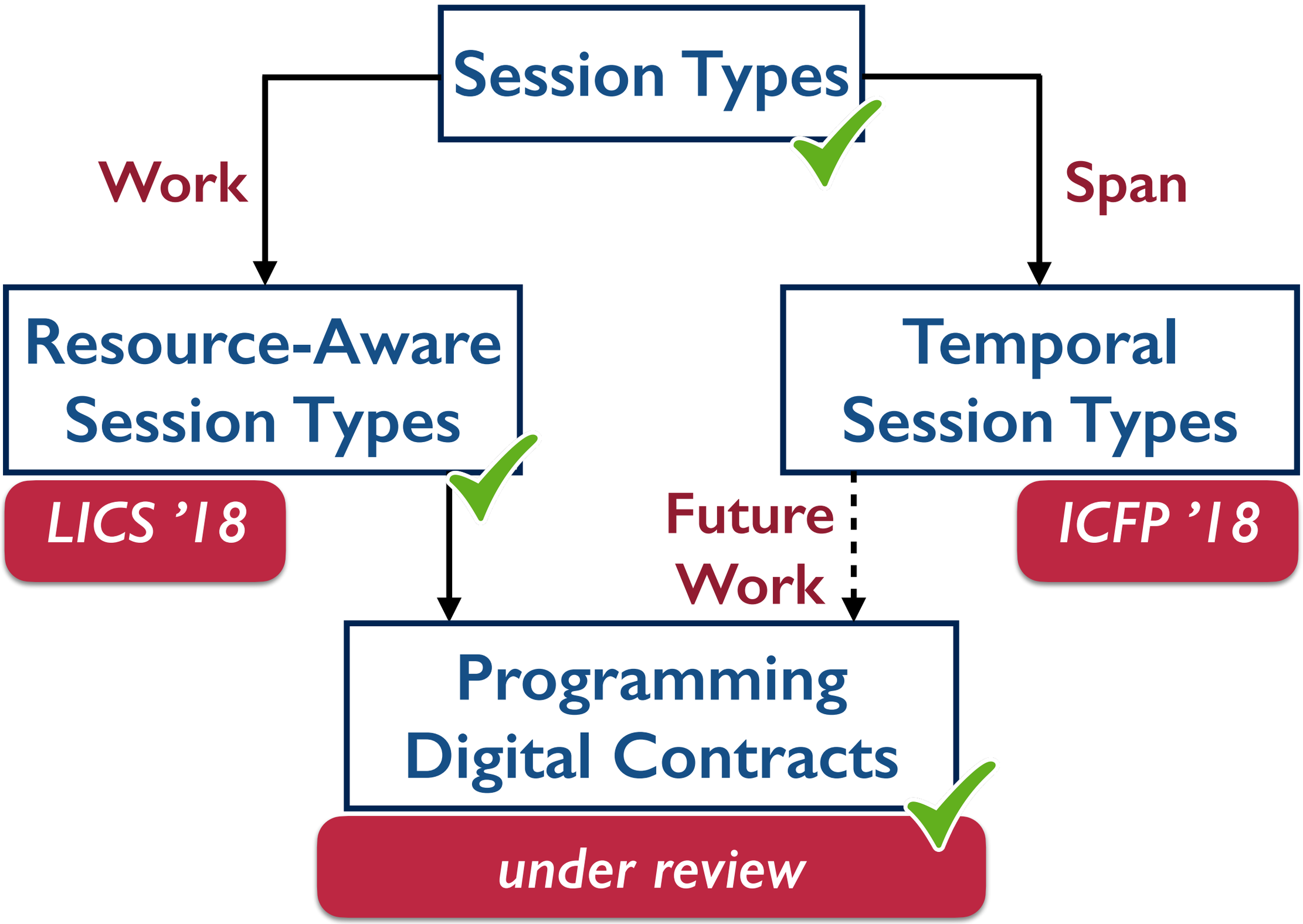
Existing Languages (e.g. Solidity)

- Protocol not explicit in code, enforced programmatically
- Resource (aka gas) usage not analyzed
- Linearity of assets (money) not enforced
- Prone to re-entrancy

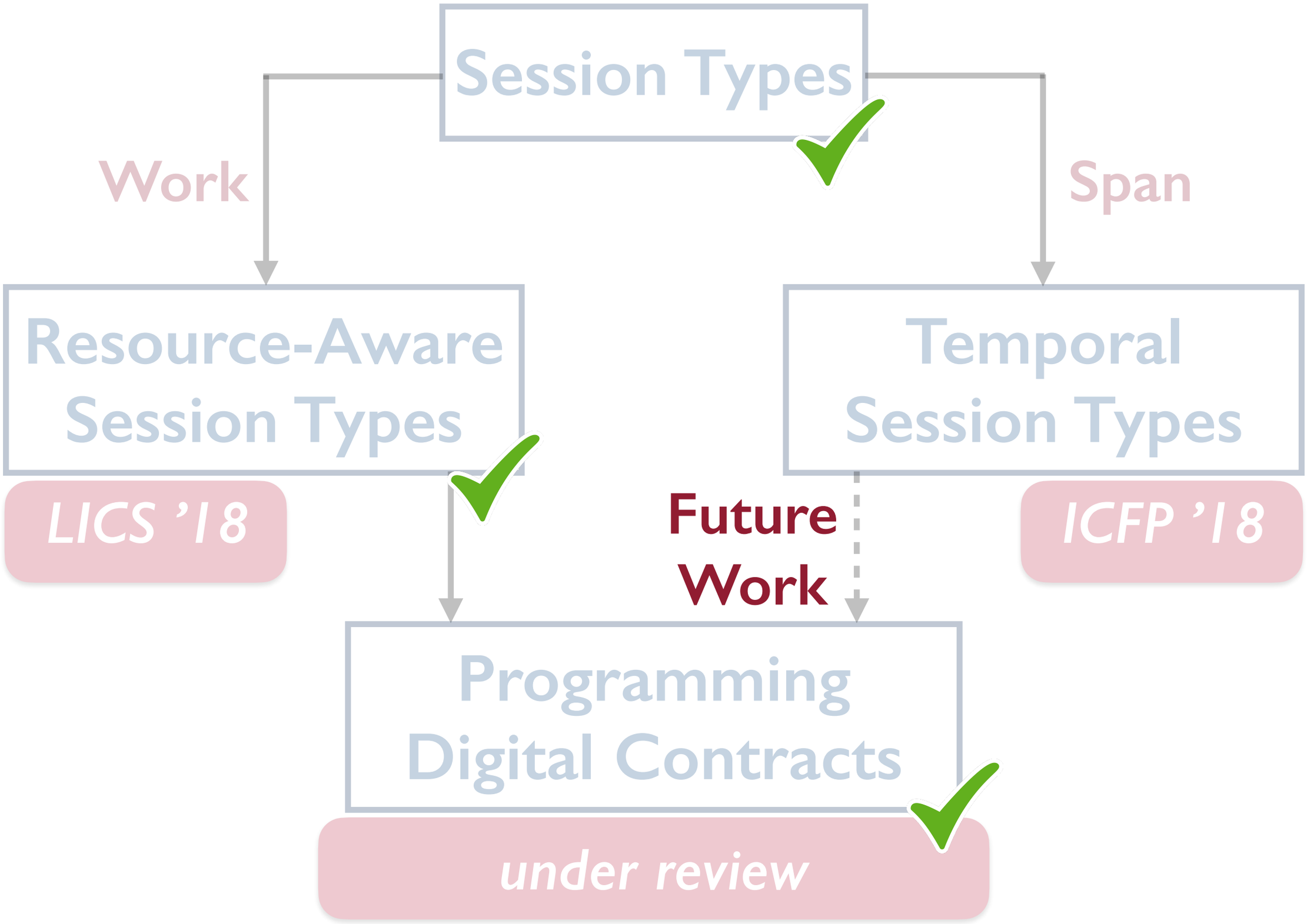
Proposed Language (Nomos)

- Session types express protocol, enforced by type checking
- Resource-aware types express gas usage
- Linear type system tracks assets
- No re-entrancy attack

Talk Outline



Talk Outline



Future Directions

- *Tracking time* in Nomos for time-specific contracts
- Evaluation of *efficiency* and *scalability* of Nomos
- *Runtime monitoring* to ensure Nomos contracts can interact with ill-typed and untyped clients
- Deadlock detection of session-typed programs
- Integrating *refinement types* to prove stronger invariants (e.g. money bid is equal to money returned) (under review)

Conclusion

- **Resource-Aware Session Types:** track sequential complexity using potential method
- **Temporal Session Types:** track parallel complexity using temporal operators

- Resource-aware session types are great for implementing digital contracts
- Types express contract protocol, track resource usage, enforce linearity of assets, prevent re-entrancy

Typing Judgment

$$\Psi \; ; \; \Gamma \; ; \; \Delta \; \vdash^{q} P :: (x : A)$$

Typing Judgment

$$\Psi \; ; \; \Gamma \; ; \; \Delta \vdash^{q} P :: (x : A)$$

Functional Context

- All structural rules
- Copying semantics
- copied during exchange

Typing Judgment

$$\Psi \; ; \; \Gamma \; ; \; \Delta \stackrel{q}{\vdash} P :: (x : A)$$

Functional Context

- All structural rules
- Copying semantics
- copied during exchange

Shared Context

- All structural rules
- Shared Semantics
- no copying of channels

Typing Judgment

$$\Psi \; ; \; \Gamma \; ; \; \Delta \stackrel{q}{\vdash} P :: (x : A)$$

Functional Context

- All structural rules
- Copying semantics
- copied during exchange

Shared Context

- All structural rules
- Shared Semantics
- no copying of channels

Linear Context

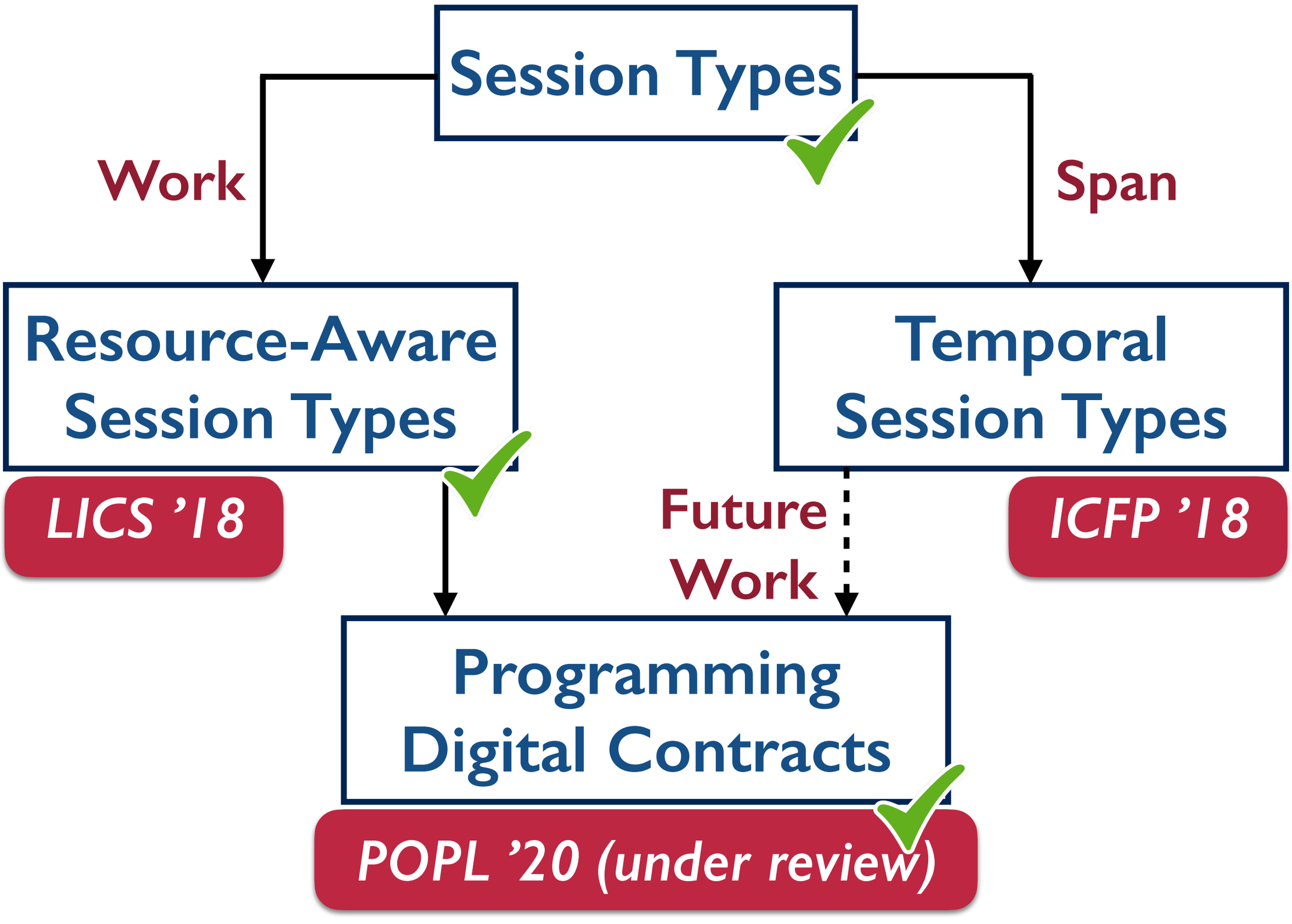
- Only exhibits exchange (no weakening or contraction)
- can't discard or duplicate

Relaxing Independence

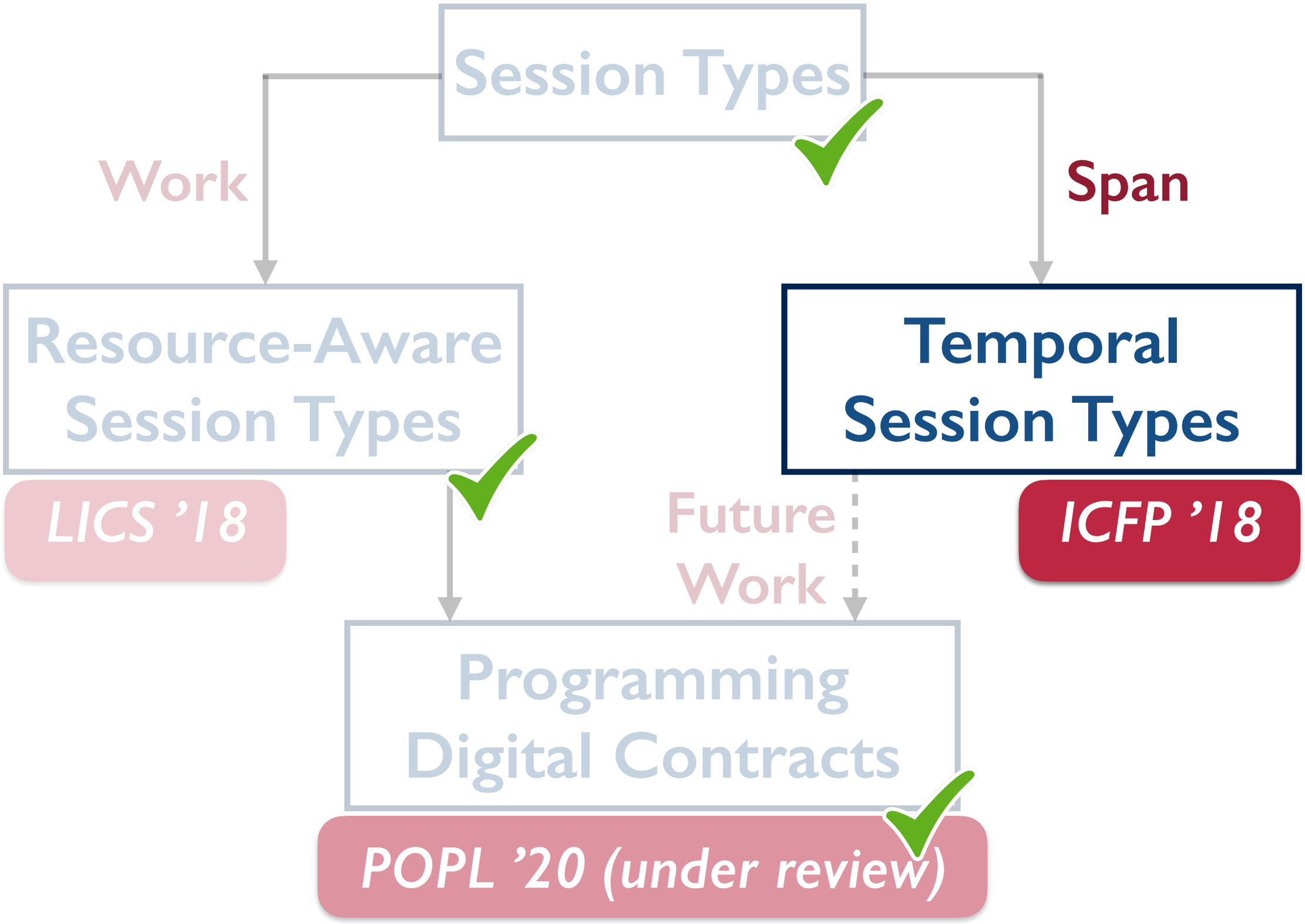
- Distinguish linear processes according to their roles
  - Assets : can only refer to other linear assets ⇒ assign mode R (e.g. money, Mona Lisa)
  - Contracts : can refer to other contracts or linear assets ⇒ assign mode L (e.g. auction)
  - Transactions : can refer to assets, contracts and transactions ⇒ assign mode T (e.g. bidder)

$$R < L < T$$

Talk Outline



Talk Outline



How is time defined?

- Time is defined using a cost model
- *Cost model* assigns a time cost to each operation

How is time defined?

- Time is defined using a cost model
- *Cost model* assigns a time cost to each operation

$\mathcal{R}$ cost model	$\mathcal{RS}$ cost model
Unit delay after each receive	Unit delay after each receive and send

How is time defined?

- Time is defined using a cost model
- *Cost model* assigns a time cost to each operation

$\mathcal{R}$ cost model	$\mathcal{RS}$ cost model
Unit delay after each receive	Unit delay after each receive and send

- Expressed by inserting appropriate delays in the source code, only the delays cost time
- Programmer specifies cost model, compiler automatically inserts delays for type checking

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

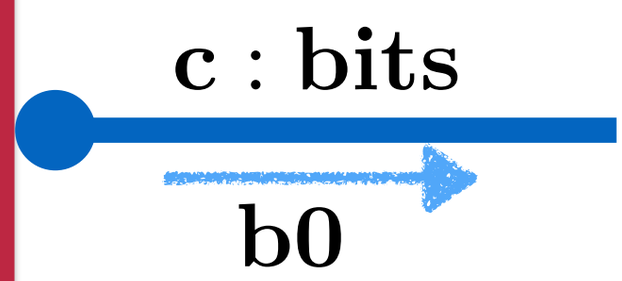
```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
 close c
```

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
  close c
```



Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \underline{\mathbf{bits}}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
close c
```

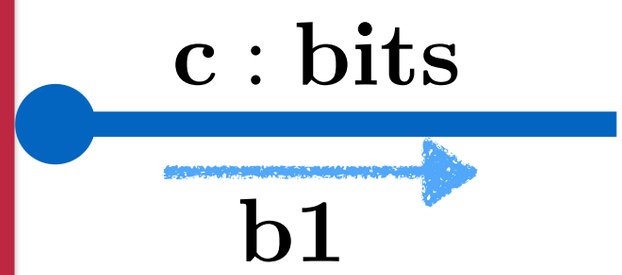
b0

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
  close c
```



b0

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \underline{\mathbf{bits}}, \$: \mathbf{1}\}$$

$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
close c
```

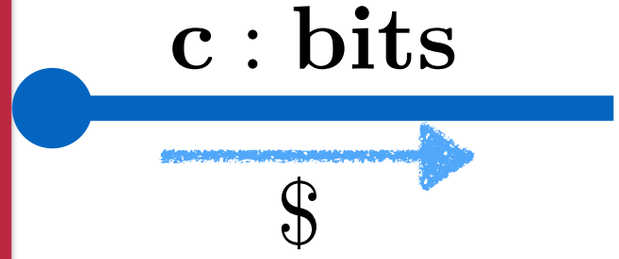
b1 b0

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
close c
```



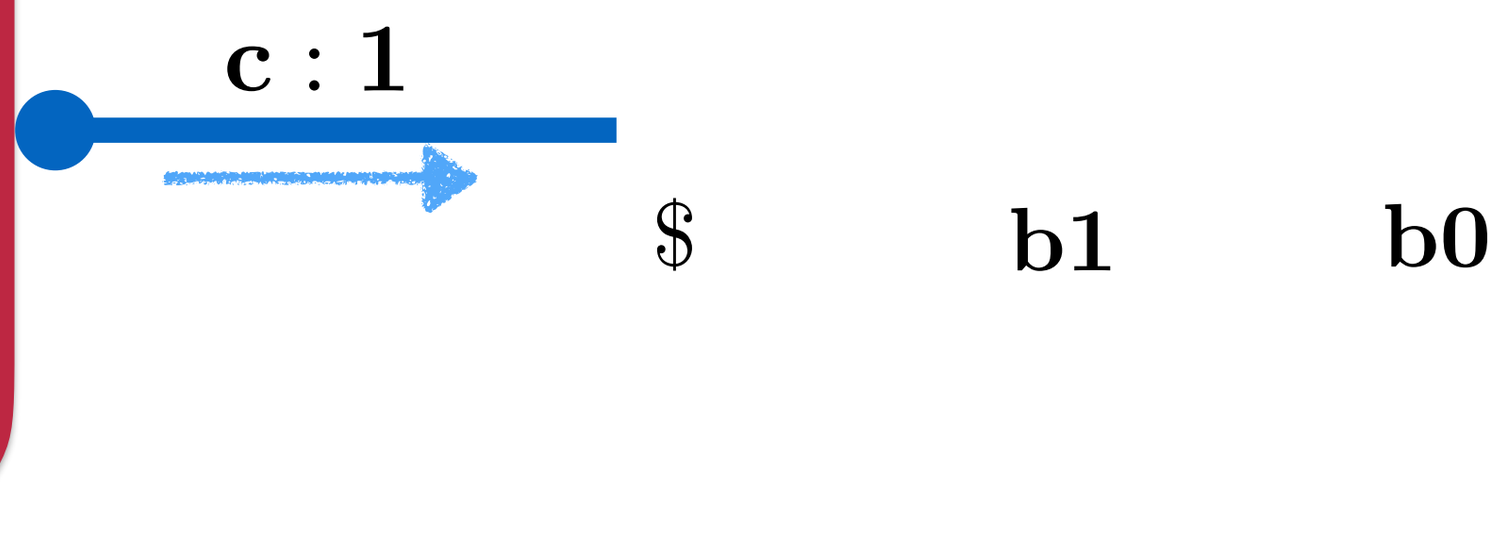
b1 b0

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \underline{\mathbf{1}}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
close c
```



Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
  close c
```

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

Timing Information?

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
  close c
```

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

Timing Information?

Sending a message causes unit delay

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
  close c
```

Example: Bit Streams

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}, \$: \mathbf{1}\}$$

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

Timing Information?

Sending a message causes unit delay

```
c ← two =
  c.b0 ;
  c.b1 ;
  c.$  ;
  close c
```

$\mathbf{close}$ $\quad \$ \quad \mathbf{b1} \quad \mathbf{b0}$

$\mathbf{t = 3} \quad \mathbf{t = 2} \quad \mathbf{t = 1} \quad \mathbf{t = 0}$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

$\mathbf{c} : \mathbf{bits}$

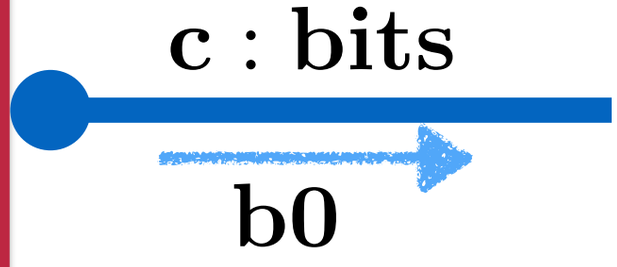
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```



$$\mathbf{t} = \mathbf{0}$$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

$\mathbf{c} : \bigcirc\mathbf{bits}$

$\mathbf{b0}$

$\mathbf{t} = \mathbf{0}$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

$\mathbf{c} : \bigcirc\mathbf{bits}$

$\mathbf{b0}$

$\mathbf{t} = \mathbf{0}$

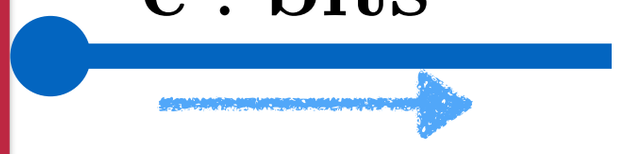
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```



$\mathbf{b0}$

$\mathbf{t = 1} \quad \mathbf{t = 0}$

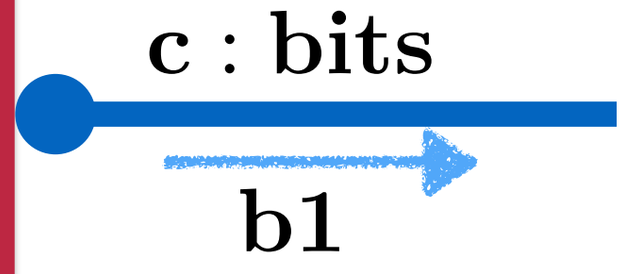
Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```



Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \underline{\bigcirc\mathbf{bits}}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

$\mathbf{c} : \bigcirc\mathbf{bits}$

$\mathbf{b1}$ $\mathbf{b0}$

$\mathbf{t = 1}$ $\mathbf{t = 0}$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

$\mathbf{c} : \bigcirc\mathbf{bits}$

$\mathbf{b1}$ $\mathbf{b0}$

$\mathbf{t = 1}$ $\mathbf{t = 0}$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

$\mathbf{b1}$ $\mathbf{b0}$

$\mathbf{t = 2}$ $\mathbf{t = 1}$ $\mathbf{t = 0}$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

$\mathbf{b1}$ $\mathbf{b0}$

$\mathbf{t = 2}$ $\mathbf{t = 1}$ $\mathbf{t = 0}$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \underline{\bigcirc\mathbf{1}}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

$\$$ $\mathbf{b1}$ $\mathbf{b0}$

$\mathbf{t = 2}$ $\mathbf{t = 1}$ $\mathbf{t = 0}$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

$\$$ $\mathbf{b1}$ $\mathbf{b0}$

$\mathbf{t = 3}$ $\mathbf{t = 2}$ $\mathbf{t = 1}$ $\mathbf{t = 0}$

Enforcing Time in the Type

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \bigcirc\mathbf{bits}, \mathbf{b1} : \bigcirc\mathbf{bits}, \$: \bigcirc\mathbf{1}\}$$

Next Operator - expresses unit delay

$$\cdot \vdash \mathbf{two} :: (\mathbf{c} : \mathbf{bits})$$

```
c ← two =
c.b0 ; delay ;
c.b1 ; delay ;
c.$  ; delay ;
close c
```

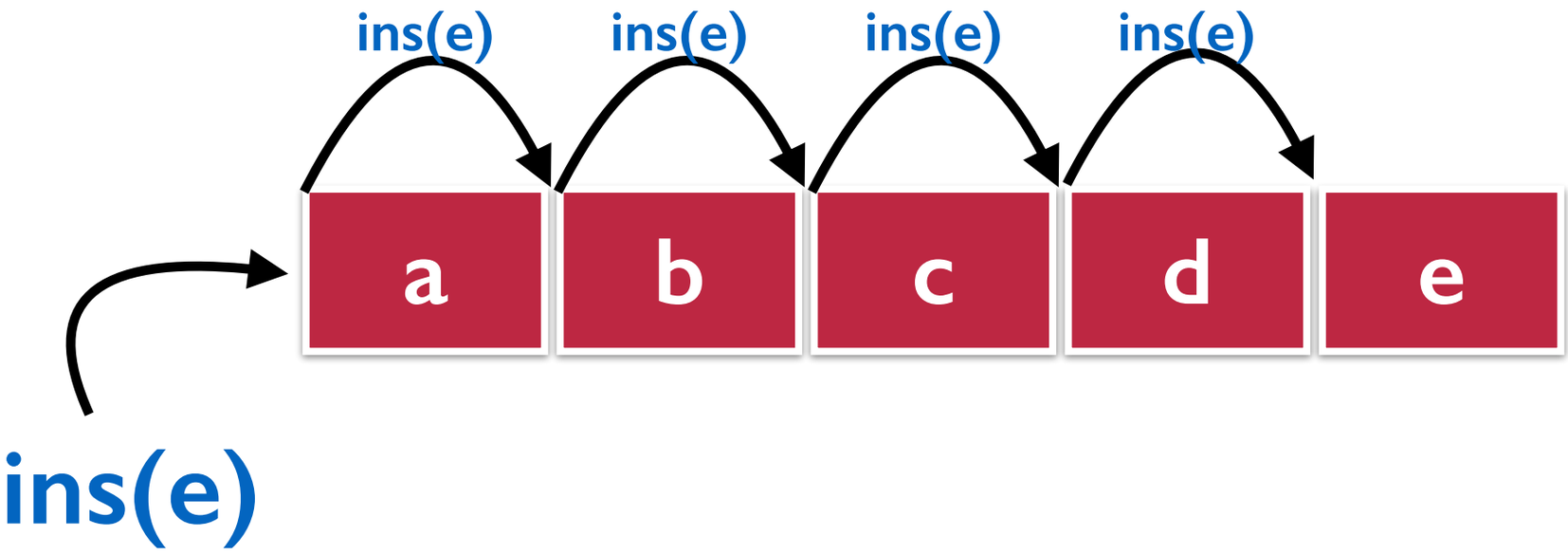
$\$$ $\mathbf{b1}$ $\mathbf{b0}$

$\mathbf{t = 3}$ $\mathbf{t = 2}$ $\mathbf{t = 1}$ $\mathbf{t = 0}$

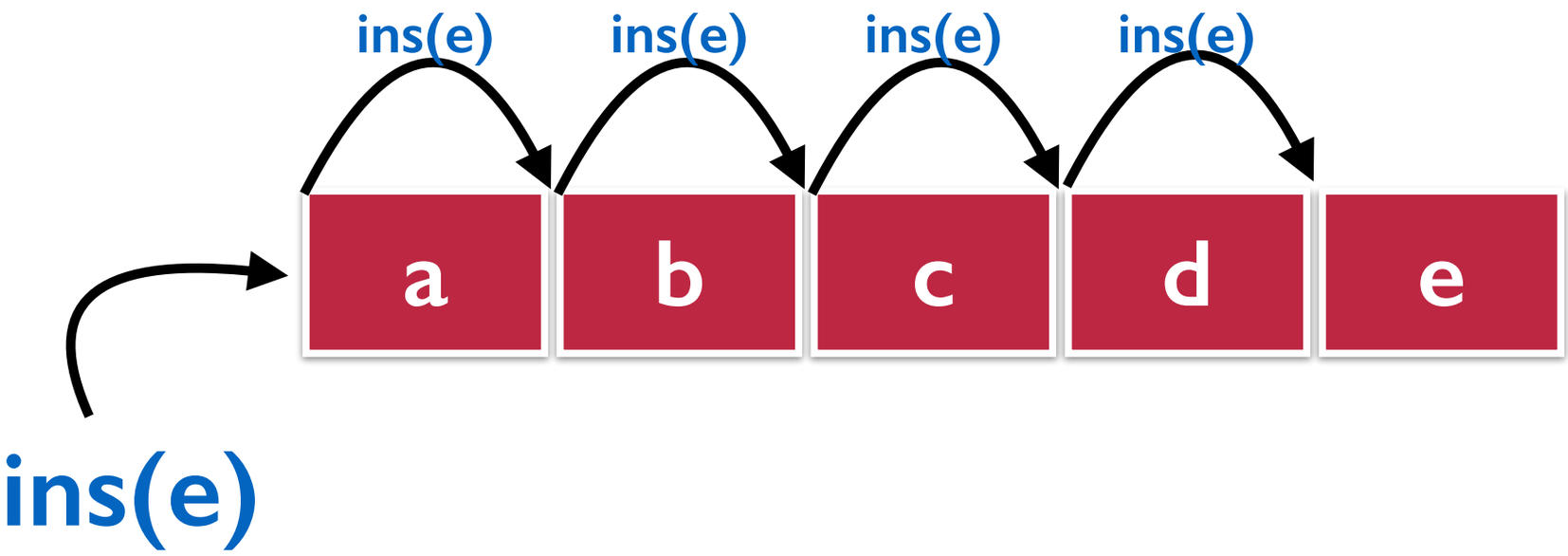
Can we type the queue?

Can we type the queue?

Can we type the queue?

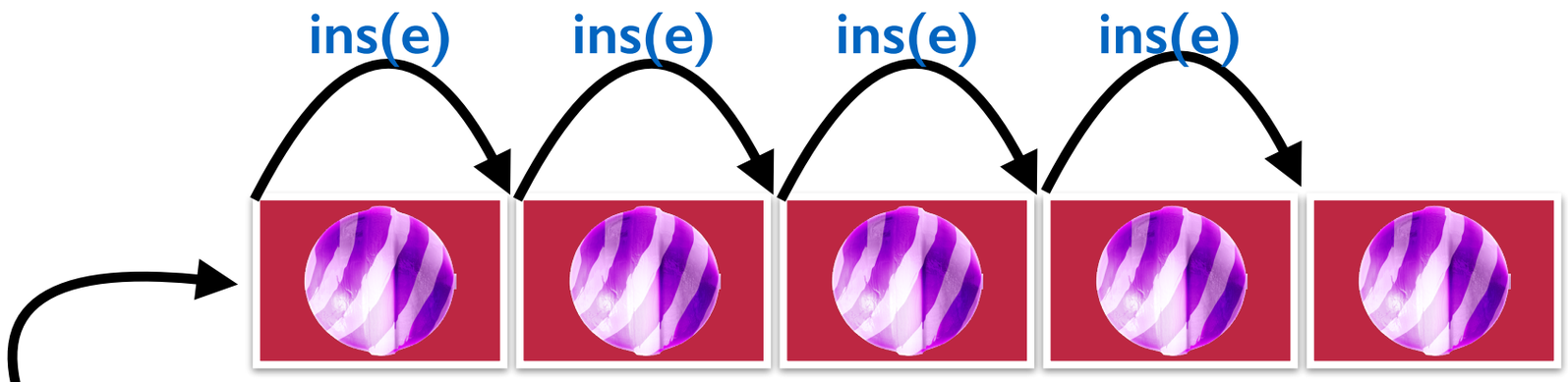


Can we type the queue?



- Next operator only expresses constant insertion rate
- But rate of insertion at the tail depends on the size of the queue — longer the queue, slower the rate
- To maintain a constant rate at the tail, new elements must be inserted at a faster rate than the previous one

Can we type the queue?



- Next operator only expresses constant insertion rate
- But rate of insertion at the tail depends on the size of the queue — longer the queue, slower the rate
- To maintain a constant rate at the tail, new elements must be inserted at a faster rate than the previous one

Providing Flexibility

- **The Box Operator (□)**
  - Provider Action: always be ready to receive token
  - Client Action: eventually send the token
    - Provider doesn't know when the token will come, only the client does
    - Different from ○ operator where both provider and client knew the timing of message exchange
- **The Diamond Operator (◇)**
  - Dual of the Box operator (provider and client flip)

Response Time of Queues

$$\mathbf{queue_A} = \Box\,\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A})\},$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

Response Time of Queues

$$\mathbf{queue_A} = \Box\,\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A})\},$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

Can always accept ins/del messages

Response Time of Queues

$$\mathbf{queue_A} = \Box\,\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A})\},$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

Can always accept ins/del messages

Response time for insertion: 3

Response Time of Queues

$$\mathbf{queue_A} = \Box\,\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A})\},$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

Can always accept ins/del messages

Response time for insertion: 3

Response time for deletion: 1

Response Time of Queues

$$\mathbf{queue_A} = \Box\,\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A})\},$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

Can always accept ins/del messages

Response time for insertion: 3

Response time for deletion: 1

Precision

WE ARE HERE!

Flexibility

Stacks vs Queues

$\mathcal{RS}$ cost model

$$\mathbf{stack_A} = \Box\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc\mathbf{stack_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{stack_A})\}\}$$

$$\mathbf{queue_A} = \Box\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

Stacks vs Queues

$\mathcal{RS}$ cost model

$$\mathbf{stack_A} = \Box\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc\mathbf{stack_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{stack_A})\}\}$$

$$\mathbf{queue_A} = \Box\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$

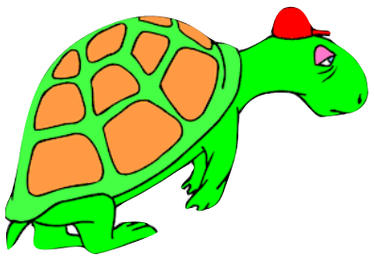
Which one's more efficient?

Stacks vs Queues

$\mathcal{RS}$ cost model

$$\mathbf{stack_A} = \Box\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc\mathbf{stack_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{stack_A})\}\}$$

$$\mathbf{queue_A} = \Box\&\{\mathbf{ins} : \bigcirc(\Box\mathbf{A} \multimap \bigcirc^{\mathbf{3}}\mathbf{queue_A}),$$
$$\mathbf{del} : \bigcirc \oplus \{\mathbf{none} : \bigcirc\mathbf{1},$$
$$\mathbf{some} : \bigcirc(\Box\mathbf{A} \otimes \bigcirc\mathbf{queue_A})\}\}$$



Which one's more efficient?

Contributions

Type system to analyze timing of message exchanges of session-typed programs

- types define the *timing* of message exchanges
- provides *precision* and *flexibility*
- proved *sound* w.r.t. *cost semantics* tracking time
- *conservative* extension to typical session type system
- applies to all *standard* session types examples
- can be *parameterized* to count resource of interest