Objects as Session-Typed Processes

Stephanie Balzer and Frank Pfenning
Computer Science Department, Carnegie Mellon University

AGERE! 2015

The essence of object-orientation

The essence of object-orientation

39 concepts (“quarks”) of object-orientation [Armstrong06]:

object, encapsulation, message passing, information hiding, dynamic dispatch, reuse, modularization, inheritance, etc.

The essence of object-orientation

39 concepts (“quarks”) of object-orientation [Armstrong06]:

object, encapsulation, message passing, information hiding, dynamic dispatch, reuse, modularization, inheritance, etc.

Object-orientation in its inception:

- Objects encapsulate state (Simula)
- Objects interact by message exchange (Smalltalk, Actor model)

The essence of object-orientation

39 concepts (“quarks”) of object-orientation [Armstrong06]:

object, encapsulation, message passing, information hiding, dynamic dispatch, reuse, modularization, inheritance, etc.

Object-orientation in its inception:

- Objects encapsulate state (Simula)
- Objects interact by message exchange (Smalltalk, Actor model)

Object-orientation to Alan Kay [public email]:

“OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things. It can be done in Smalltalk and in LISP. There are possibly other systems in which this is possible, but I’m not aware of them.”

The essence of object-orientation

- Objects encapsulate state
- Computation by message exchange
- Messages are exchanged simultaneously
- State transitions due to message exchange

The essence of object-orientation

- Objects encapsulate state
- Computation by message exchange
- Messages are exchanged simultaneously
- State transitions due to message exchange

Suggested model of computation:

- inherently concurrent
- allows expression of valid sequences of state transitions

This paper in a nutshell

A fresh look at object-oriented programming:

- Objects as processes
- Objects interact by sending messages along channels, where objects are identified with offering channel
- Channels (and offering object) are typed by linear session types, making client own the offering object

We introduce the programming language CLOO and show that:

- typical oo patterns (dynamic dispatch, subtyping) arise naturally
- new forms of expression (type-directed reuse, internal choice) emerge

Important concern:

- Support program reasoning, whilst maintaining object-oriented style

Contributions

Concurrent message-passing programming model with:

- static protocol assurance
- absence of data races and deadlocks

Object-oriented programming model with:

- typical oo concepts (encapsulation, dynamic dispatch, subtyping)
- new forms of expressions (type-directed reuse, internal choice)

Prototype compiler:

- Supports most of presented features

Progress and preservation proof (in meantime):

- for core subset of language

Outline

Background: linear session types

Basic correspondence between CLOO - object-oriented concepts:

- Objects as processes
- Dynamic dispatch
- Structural subtyping

New forms of expression:

- Type-directed delegation
- Internal Choice

Conclusions

Outline

Background: linear session types

Basic correspondence between CLOO - object-oriented concepts:

- Objects as processes
- Dynamic dispatch
- Structural subtyping

New forms of expression:

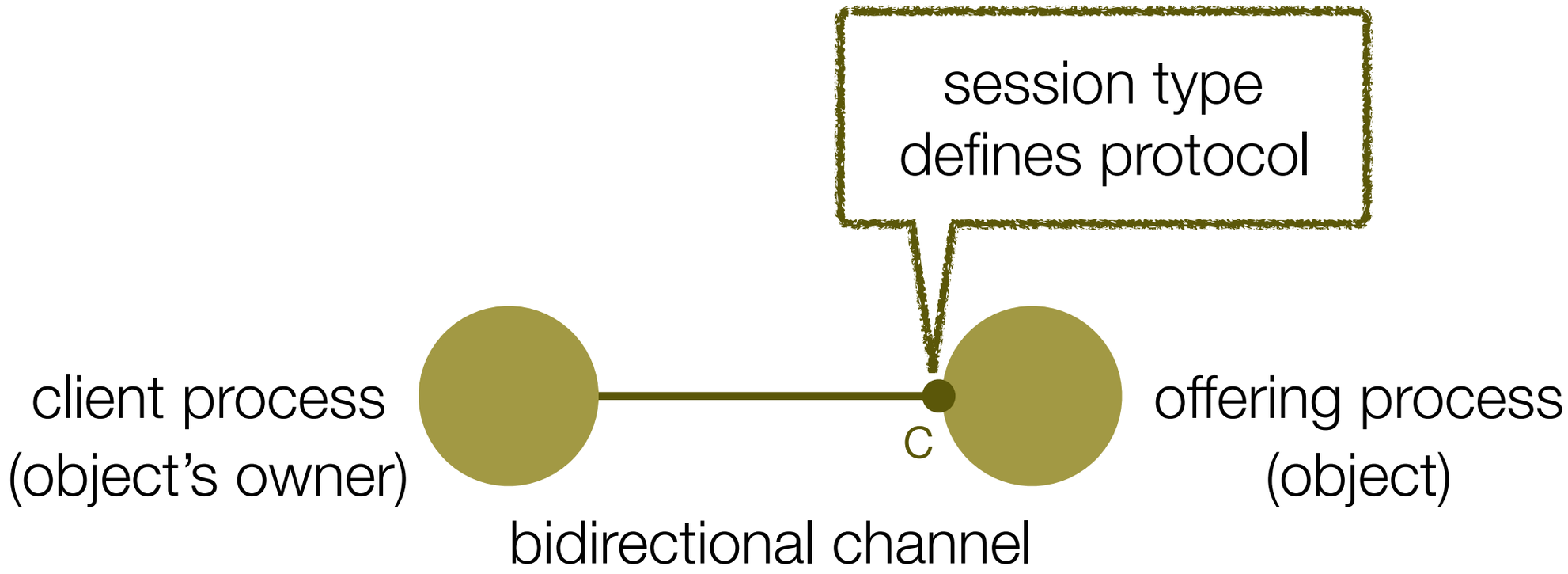
- Type-directed delegation
- Internal Choice

Conclusions

Linear session-based communication

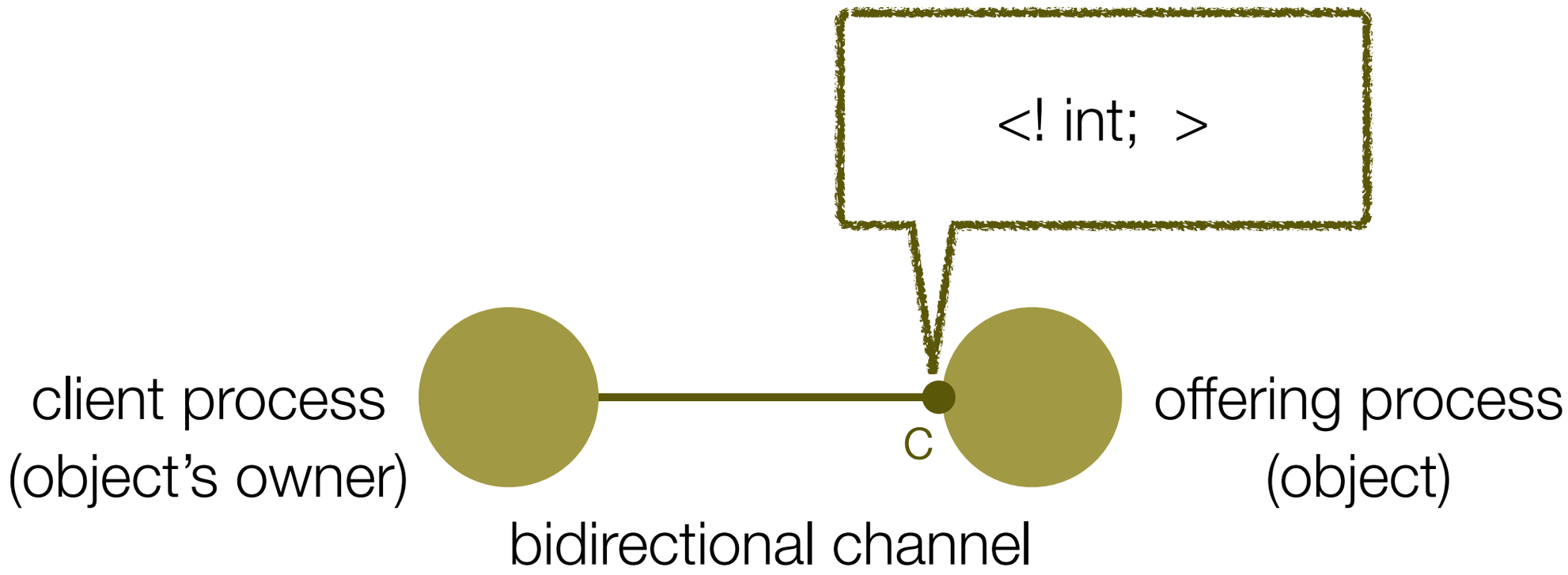
[Based on intuitionistic linear sequent calculus]

Linear session-based communication



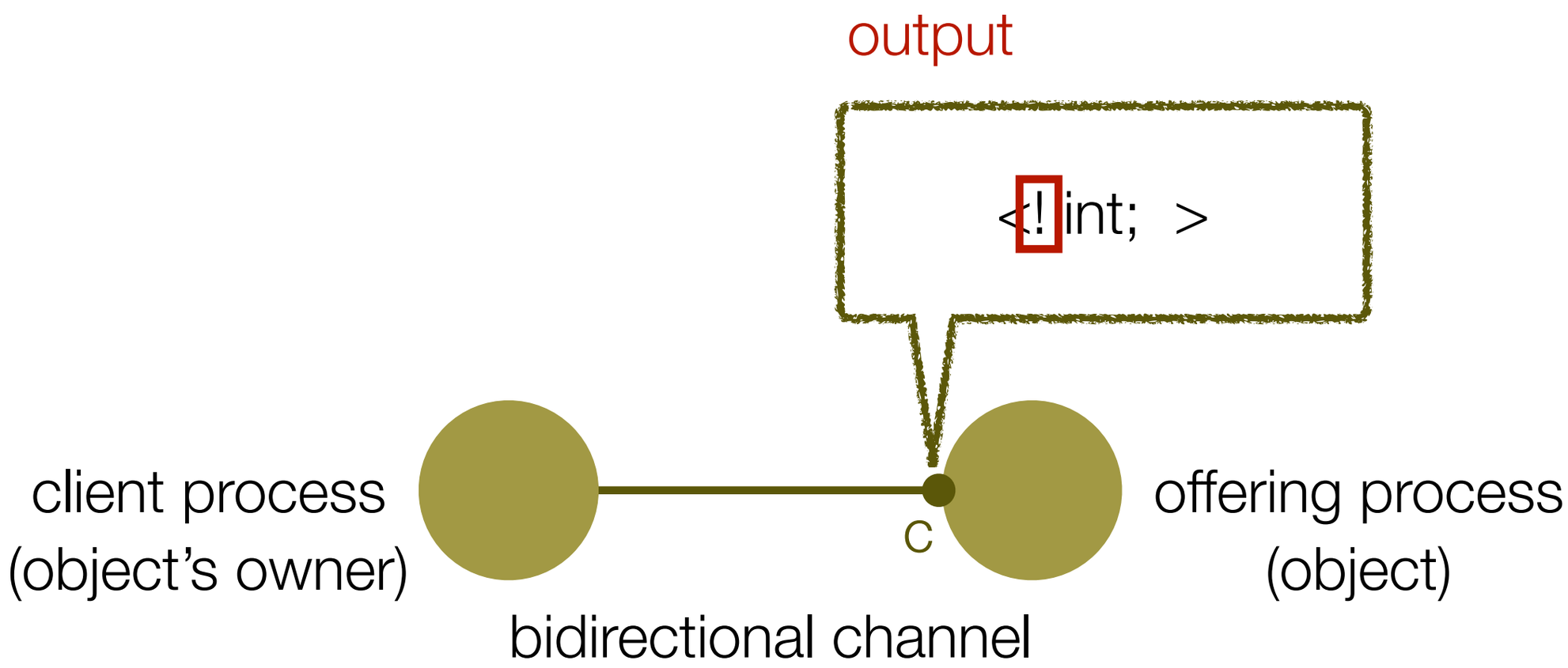
[Based on intuitionistic linear sequent calculus]

Linear session-based communication



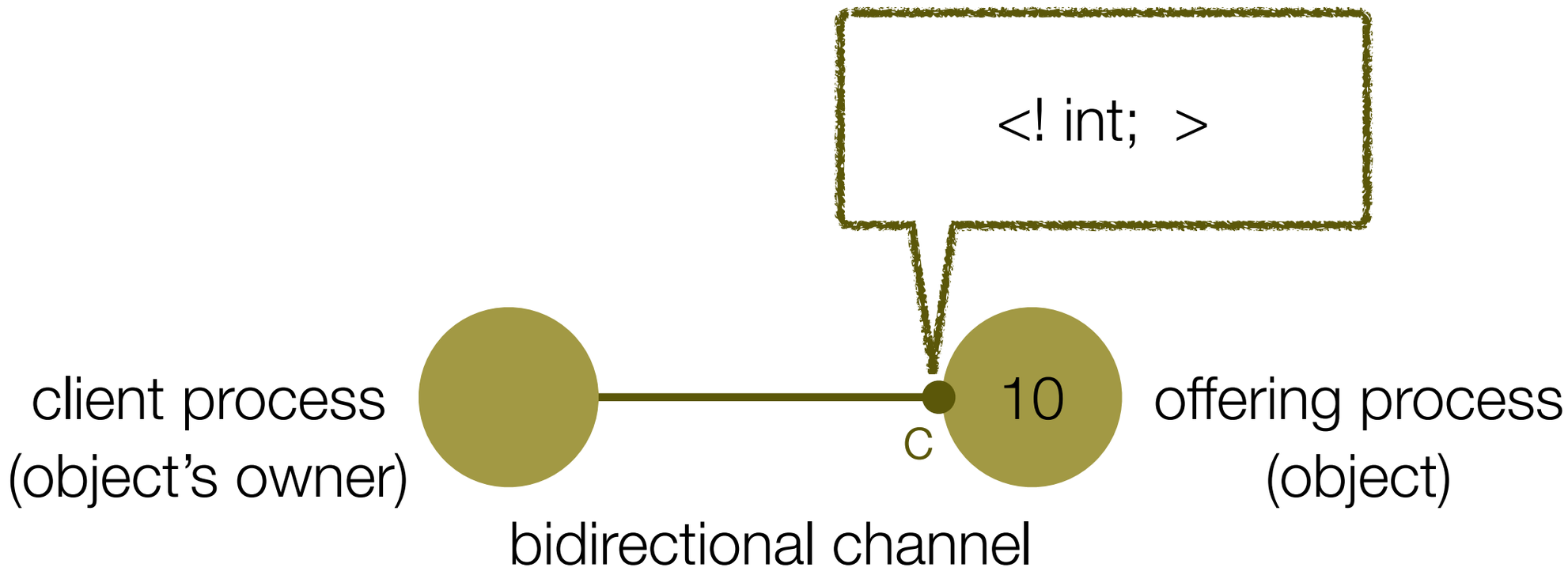
[Based on intuitionistic linear sequent calculus]

Linear session-based communication



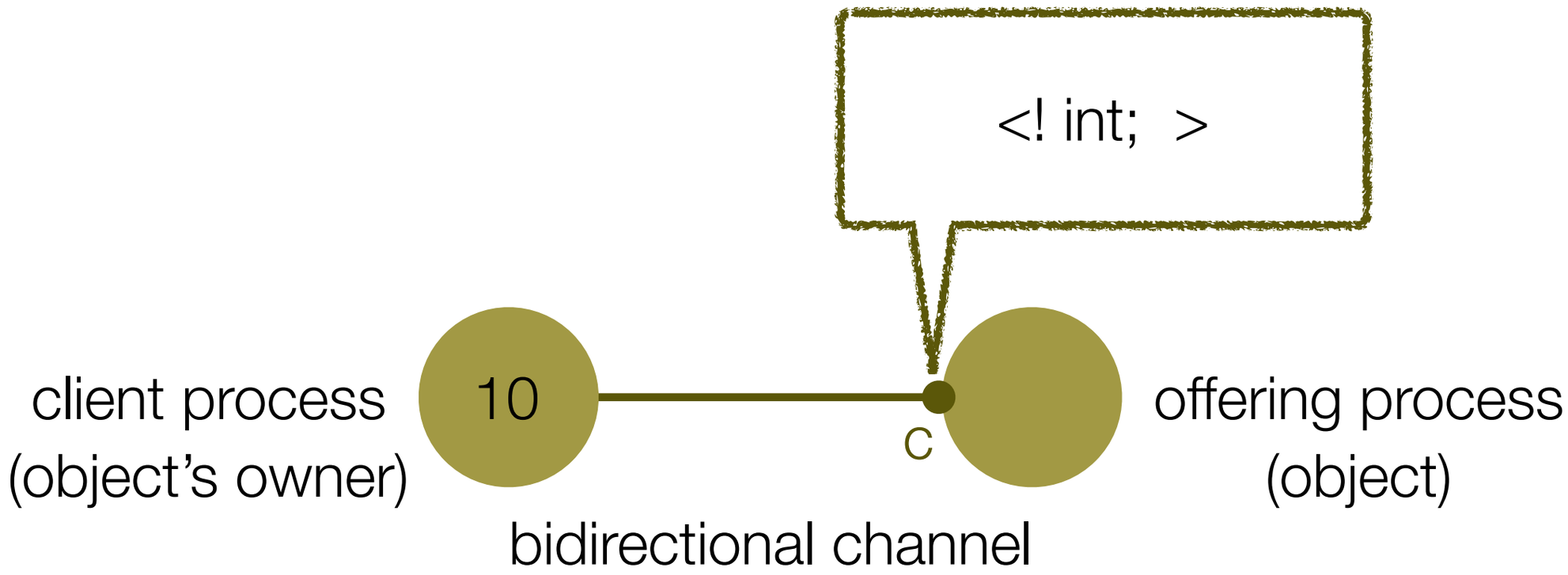
[Based on intuitionistic linear sequent calculus]

Linear session-based communication



[Based on intuitionistic linear sequent calculus]

Linear session-based communication



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

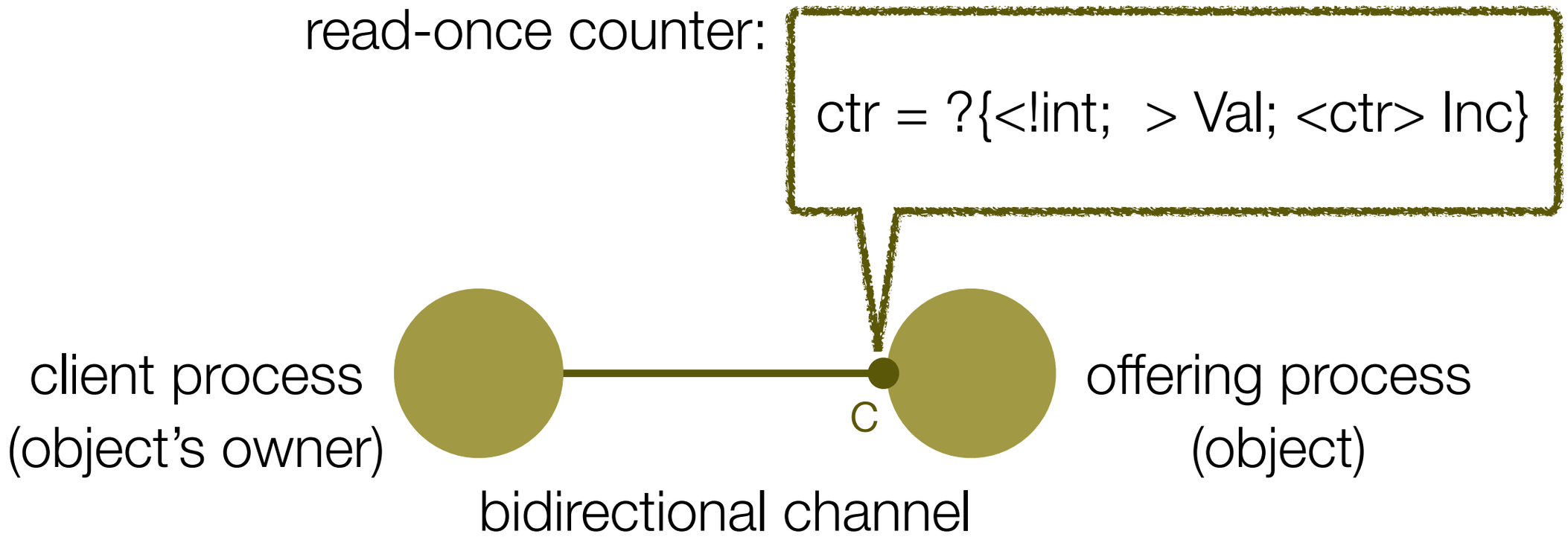
client process
(object's owner)

[Based on intuitionistic linear sequent calculus]

Linear session-based communication

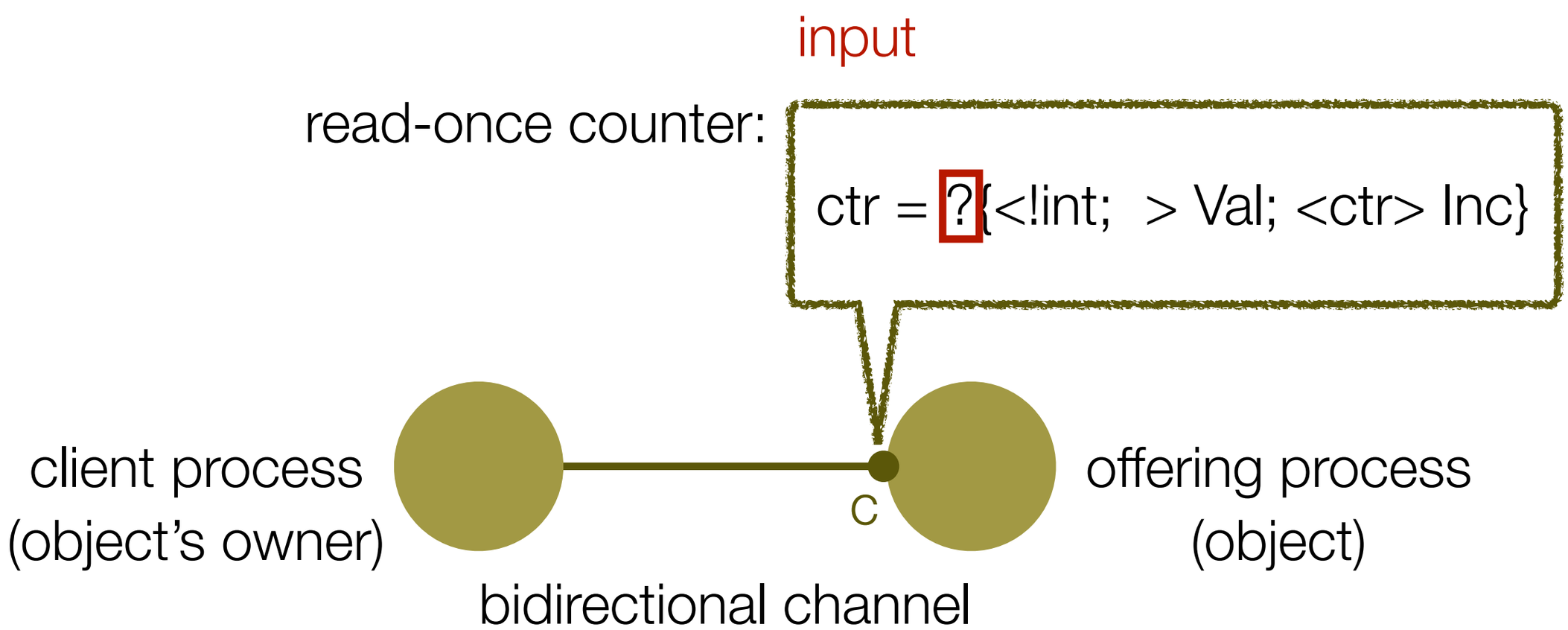
[Based on intuitionistic linear sequent calculus]

Linear session-based communication



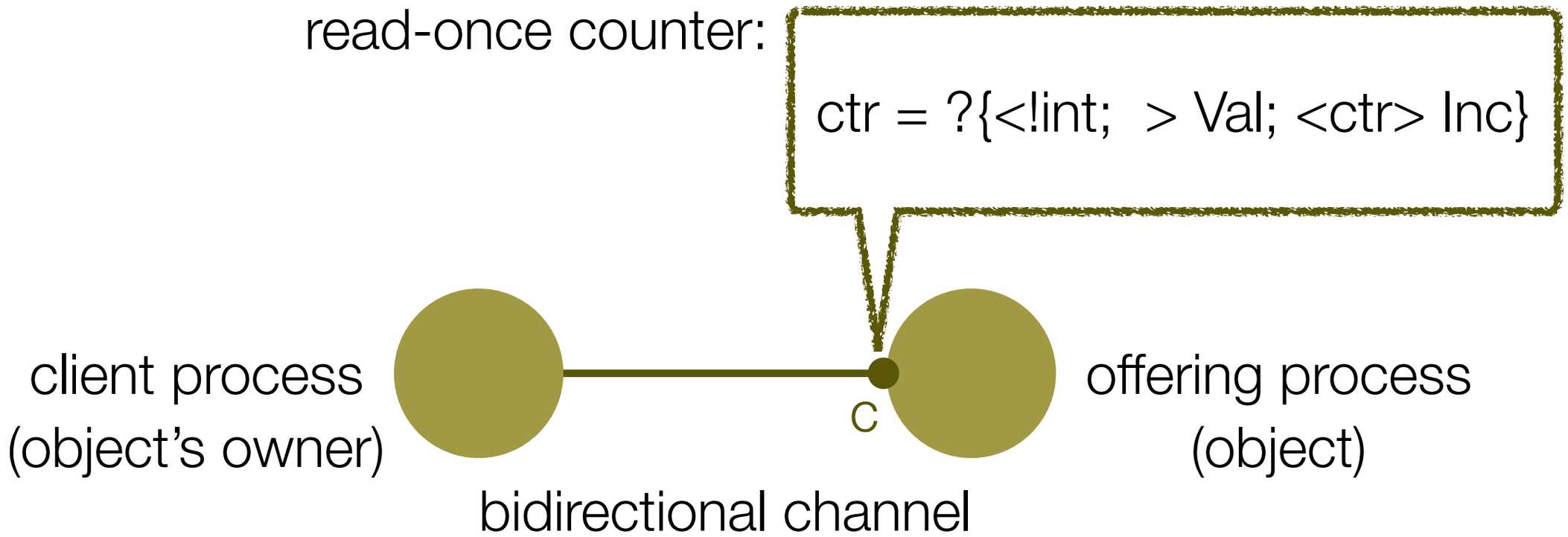
[Based on intuitionistic linear sequent calculus]

Linear session-based communication



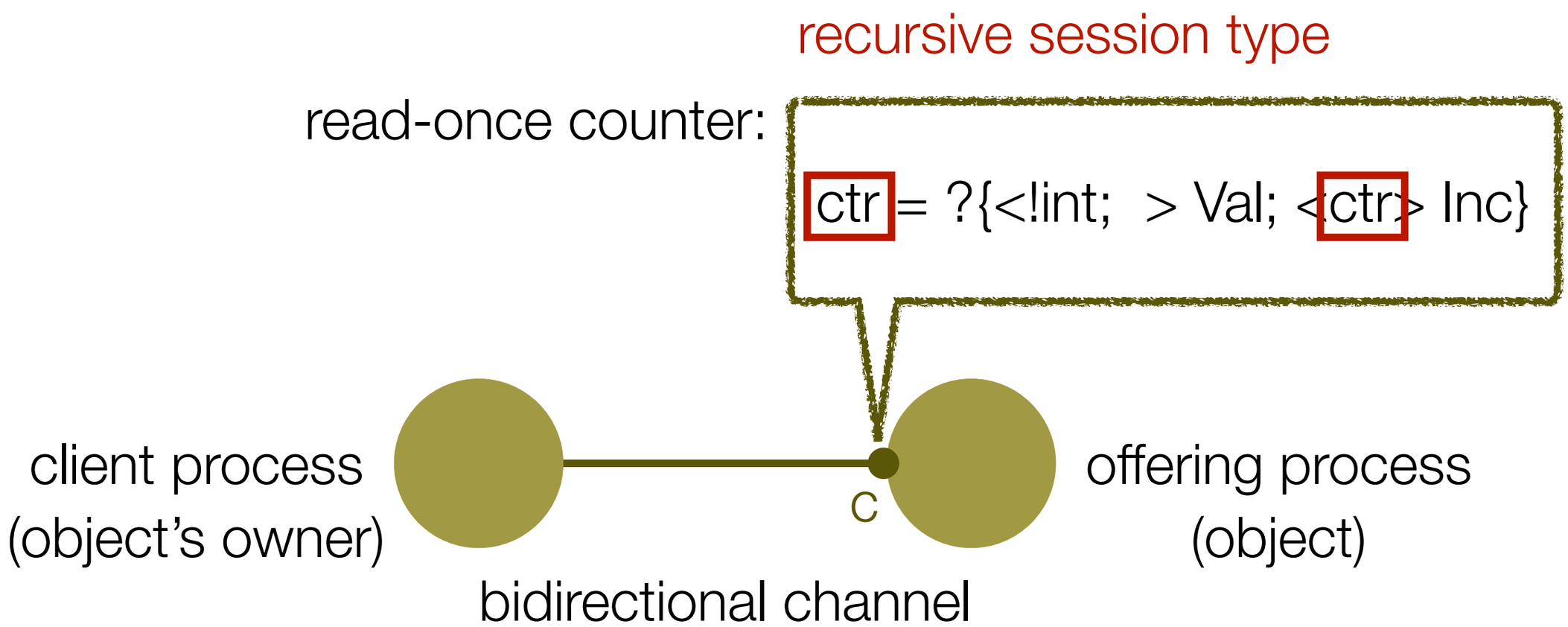
[Based on intuitionistic linear sequent calculus]

Linear session-based communication



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

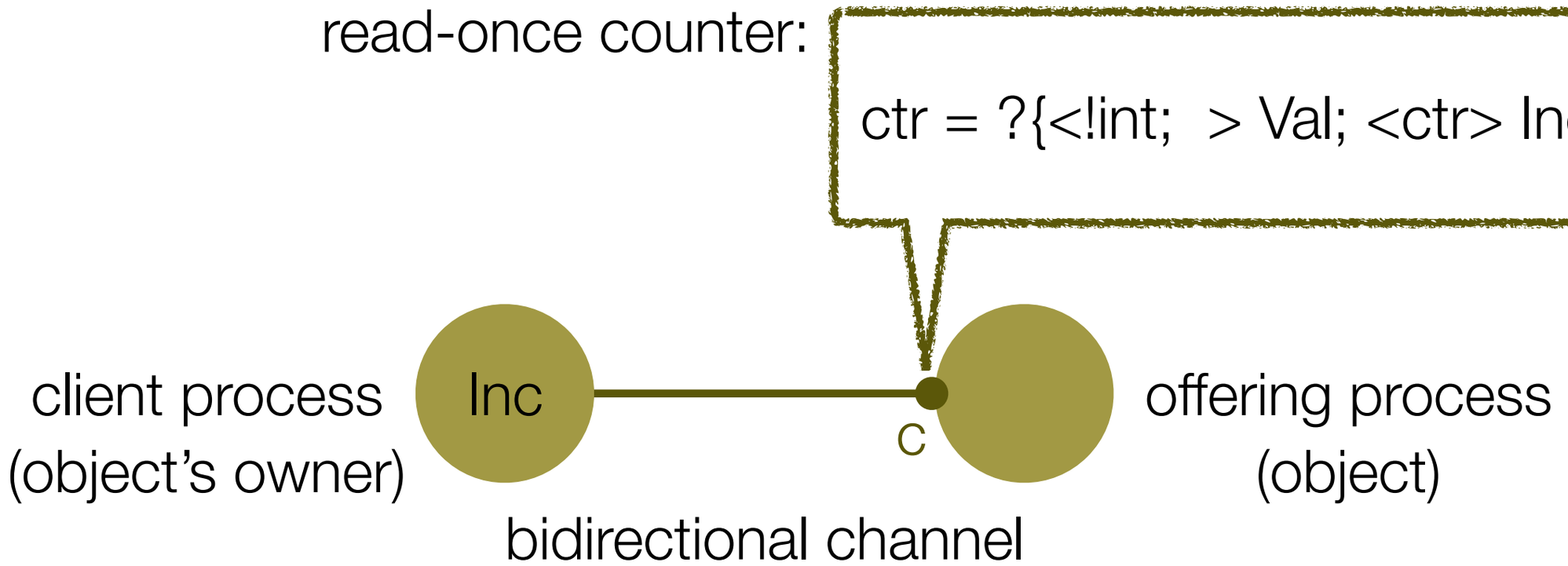


[Based on intuitionistic linear sequent calculus]

Linear session-based communication

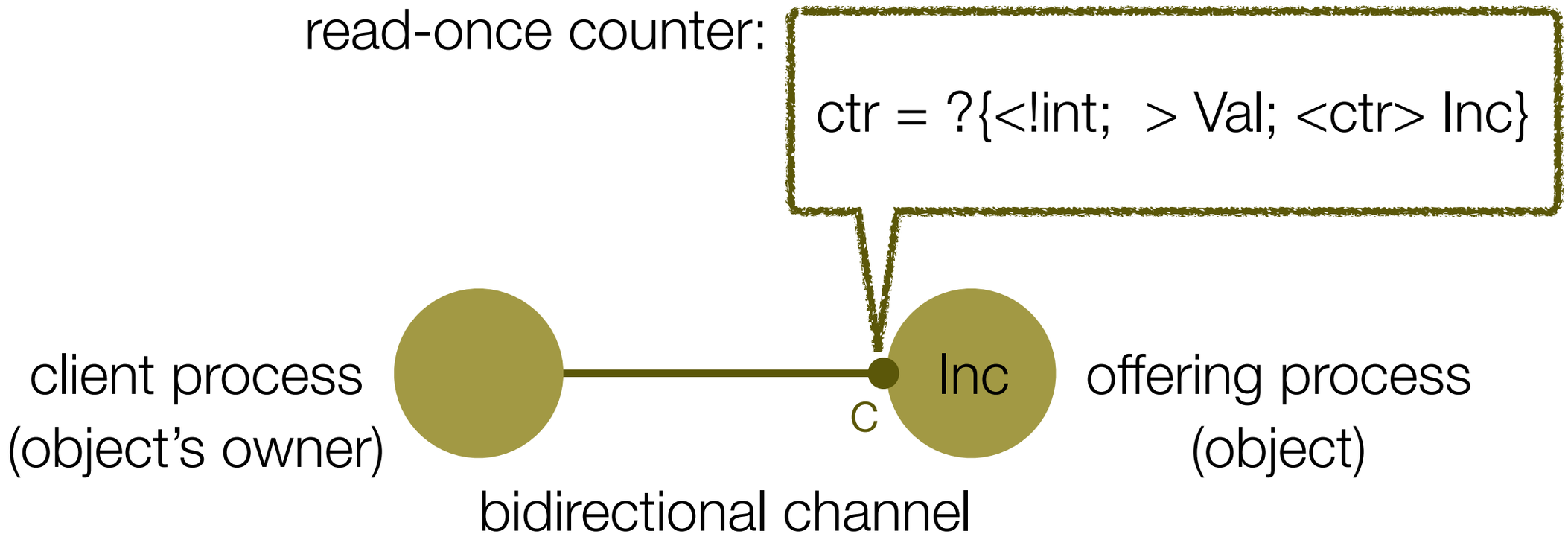
read-once counter:

ctr = ?{<!int; > Val; <ctr> Inc}



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

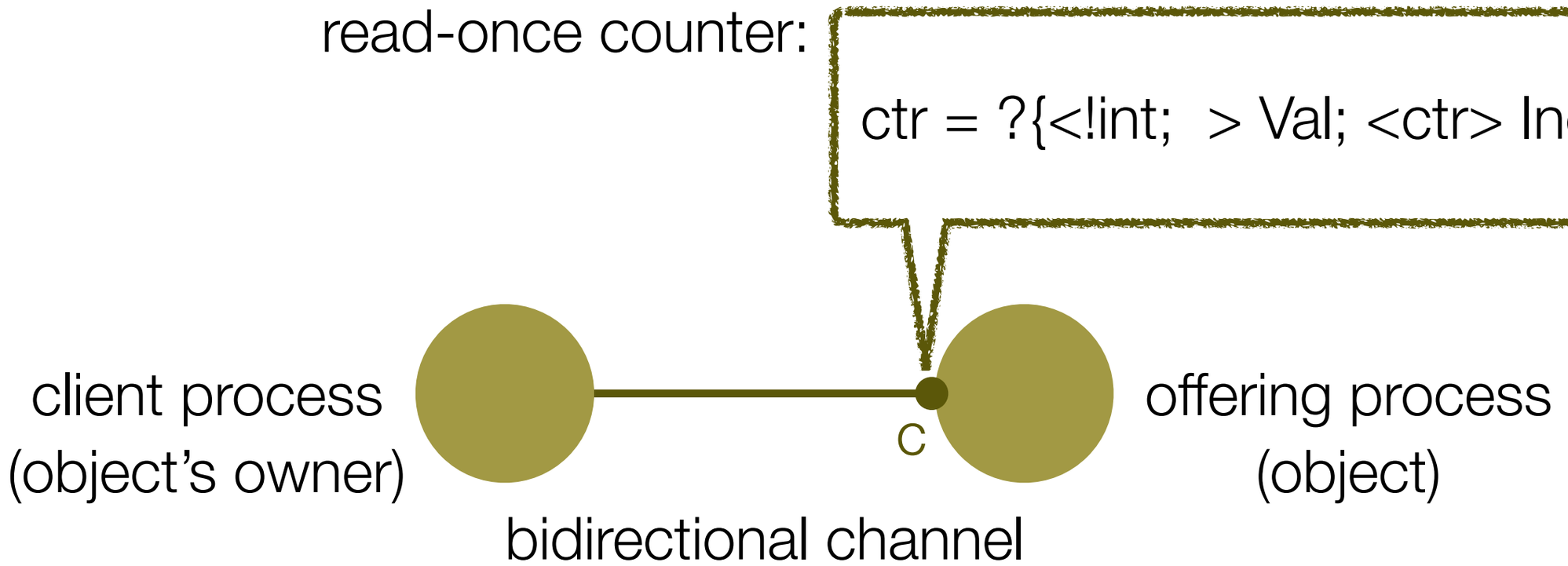


[Based on intuitionistic linear sequent calculus]

Linear session-based communication

read-once counter:

ctr = ?{<!int; > Val; <ctr> Inc}



[Based on intuitionistic linear sequent calculus]

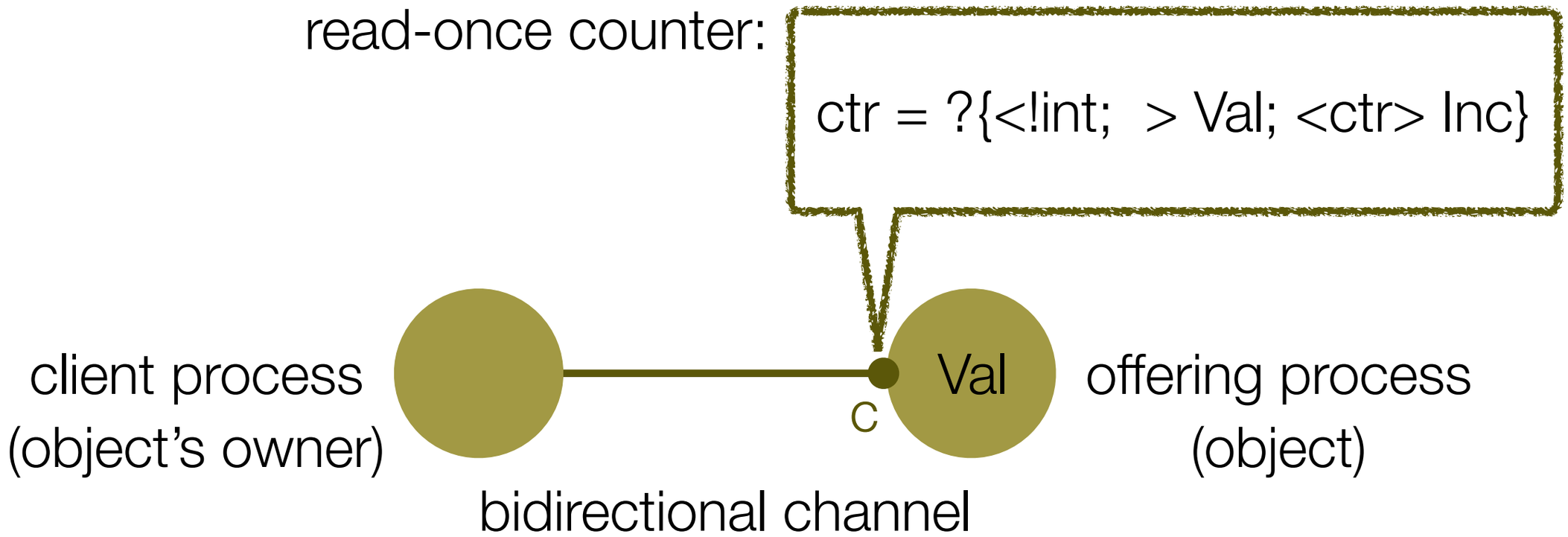
Linear session-based communication

read-once counter:

ctr = ?{<!int; > Val; <ctr> Inc}

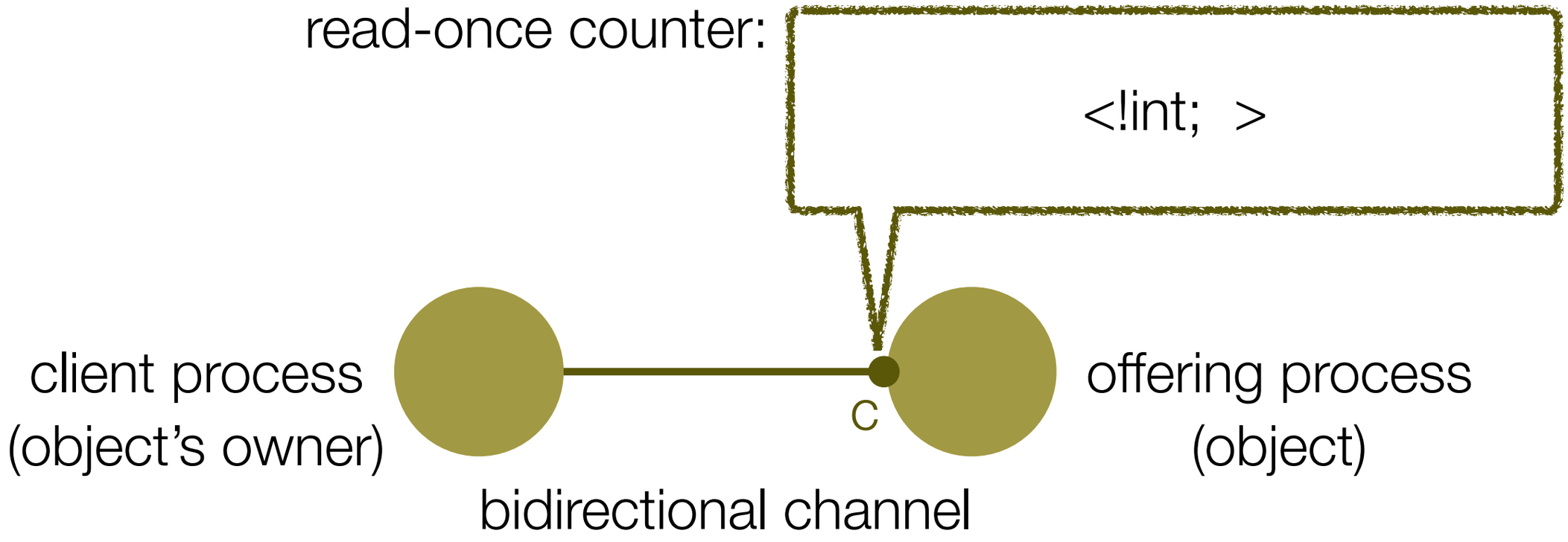
[Based on intuitionistic linear sequent calculus]

Linear session-based communication



[Based on intuitionistic linear sequent calculus]

Linear session-based communication



[Based on intuitionistic linear sequent calculus]

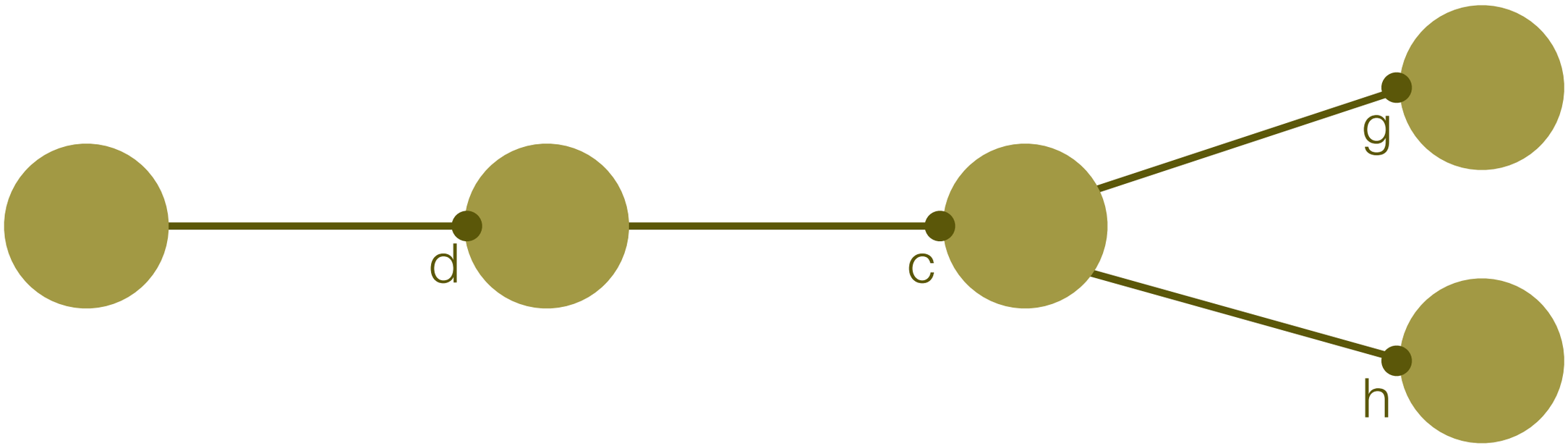
Linear session-based communication

[Based on intuitionistic linear sequent calculus]

Linear session-based communication

[Based on intuitionistic linear sequent calculus]

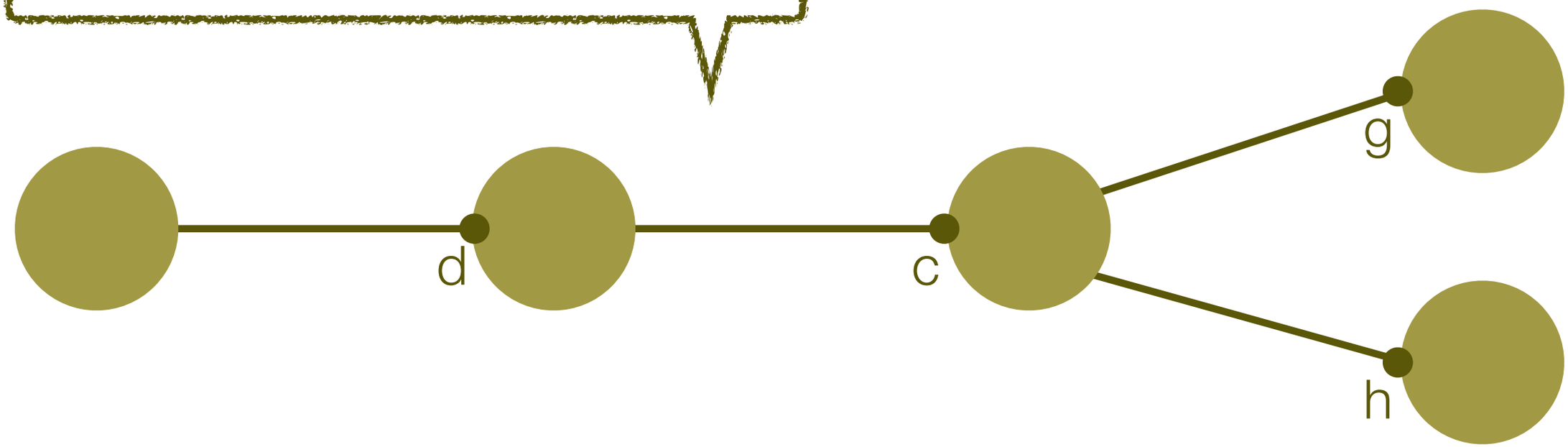
Linear session-based communication



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

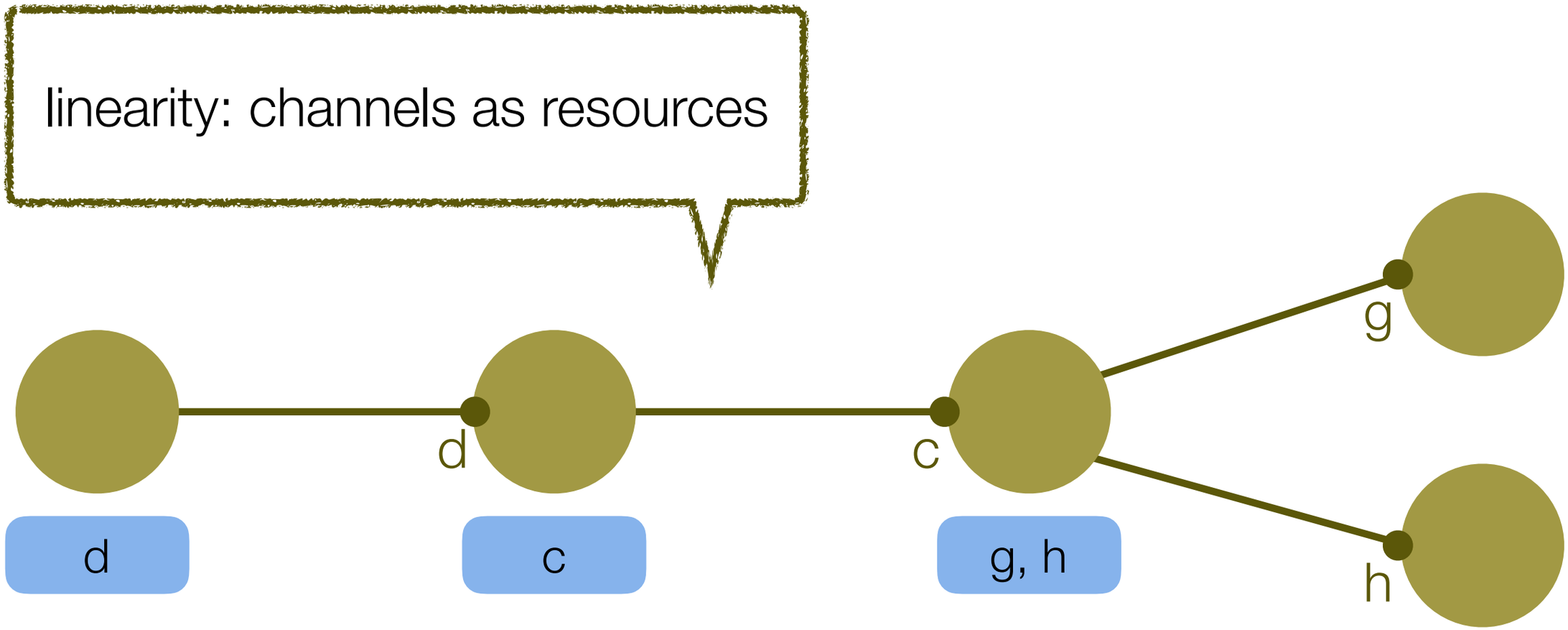
linearity: channels as resources



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

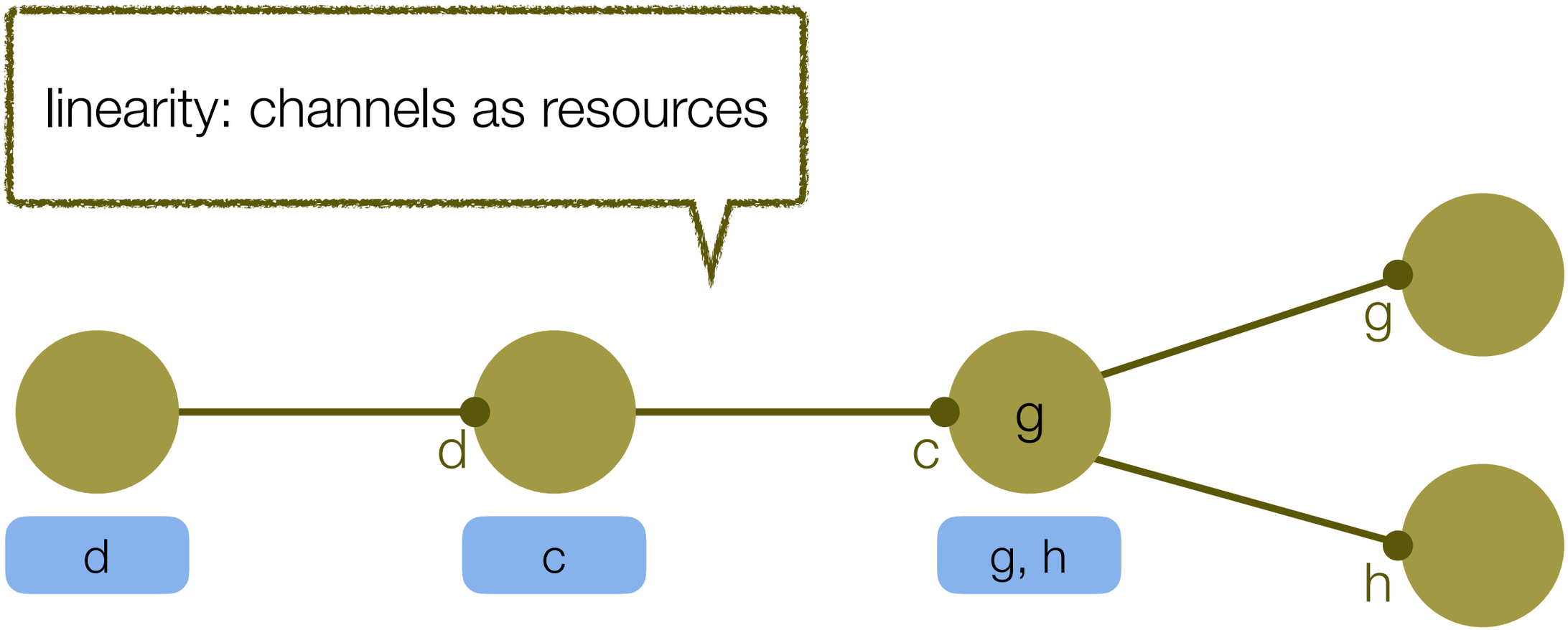
linearity: channels as resources



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

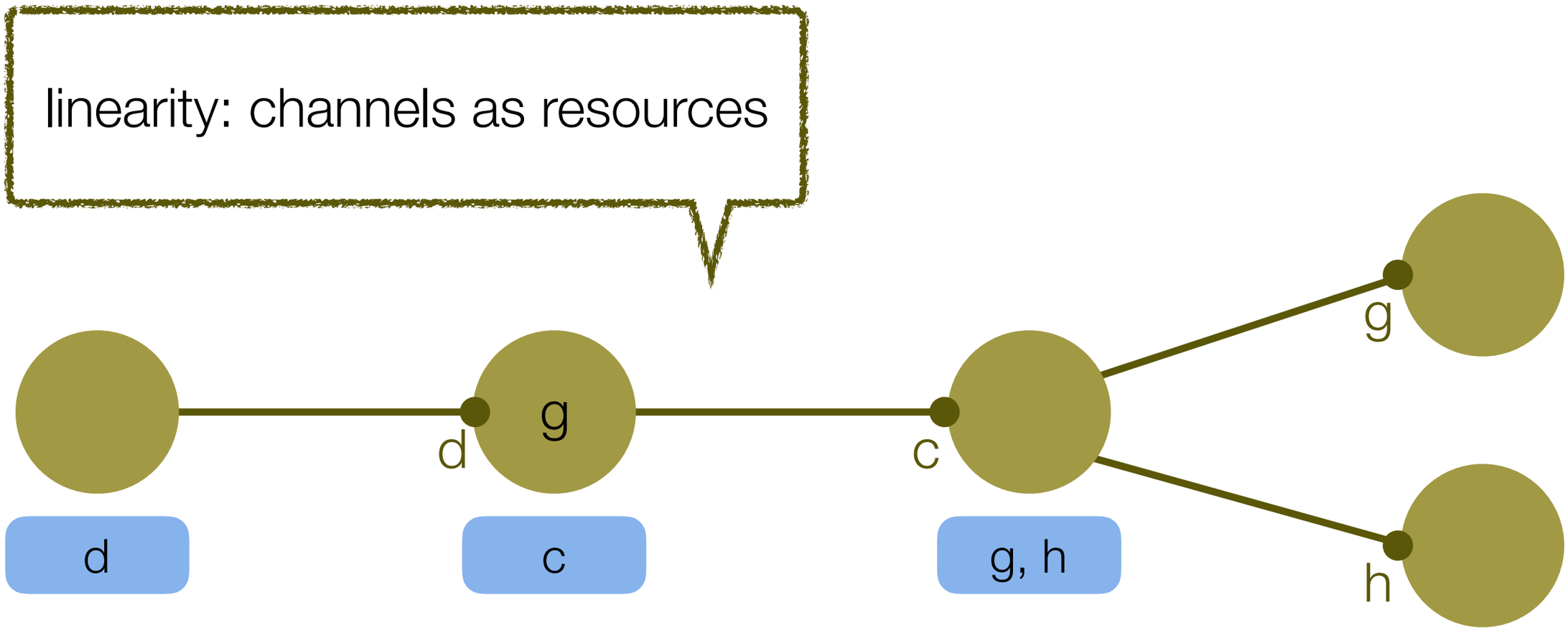
linearity: channels as resources



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

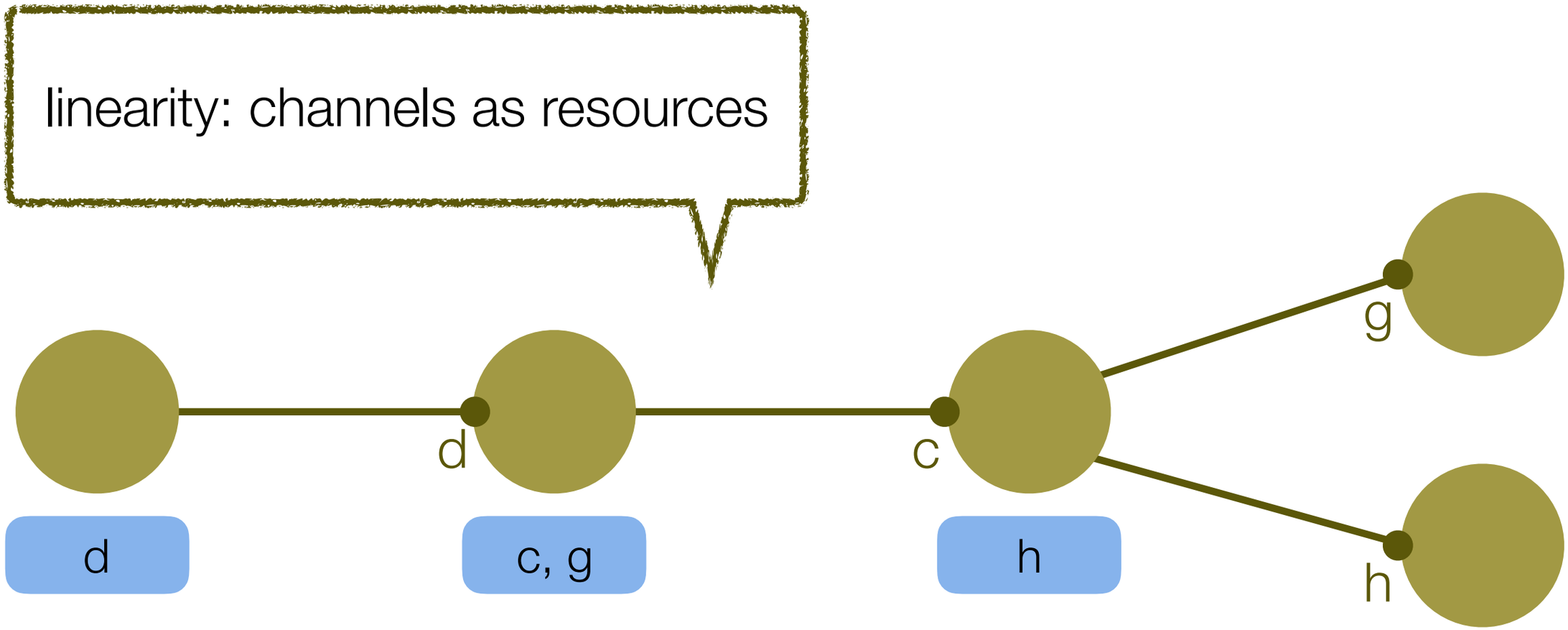
linearity: channels as resources



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

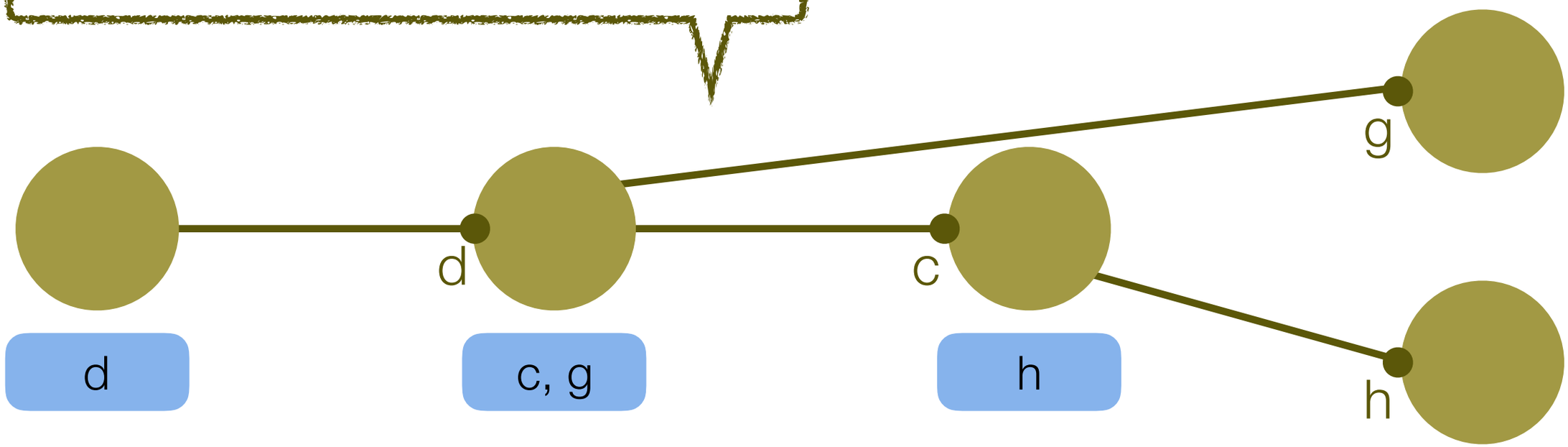
linearity: channels as resources



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

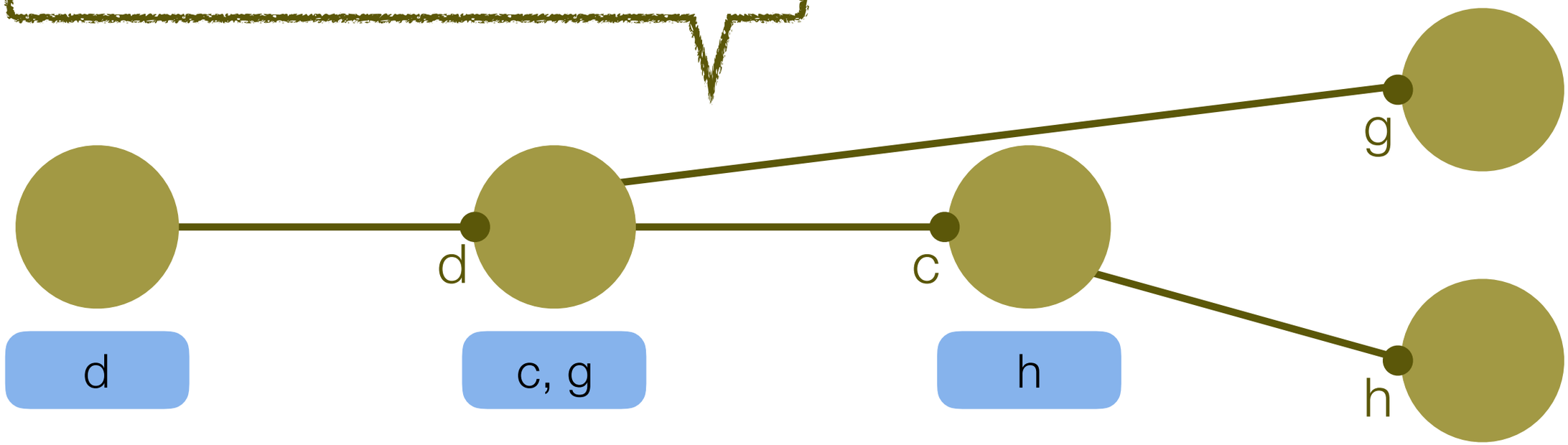
linearity: channels as resources



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

linearity: channels as resources

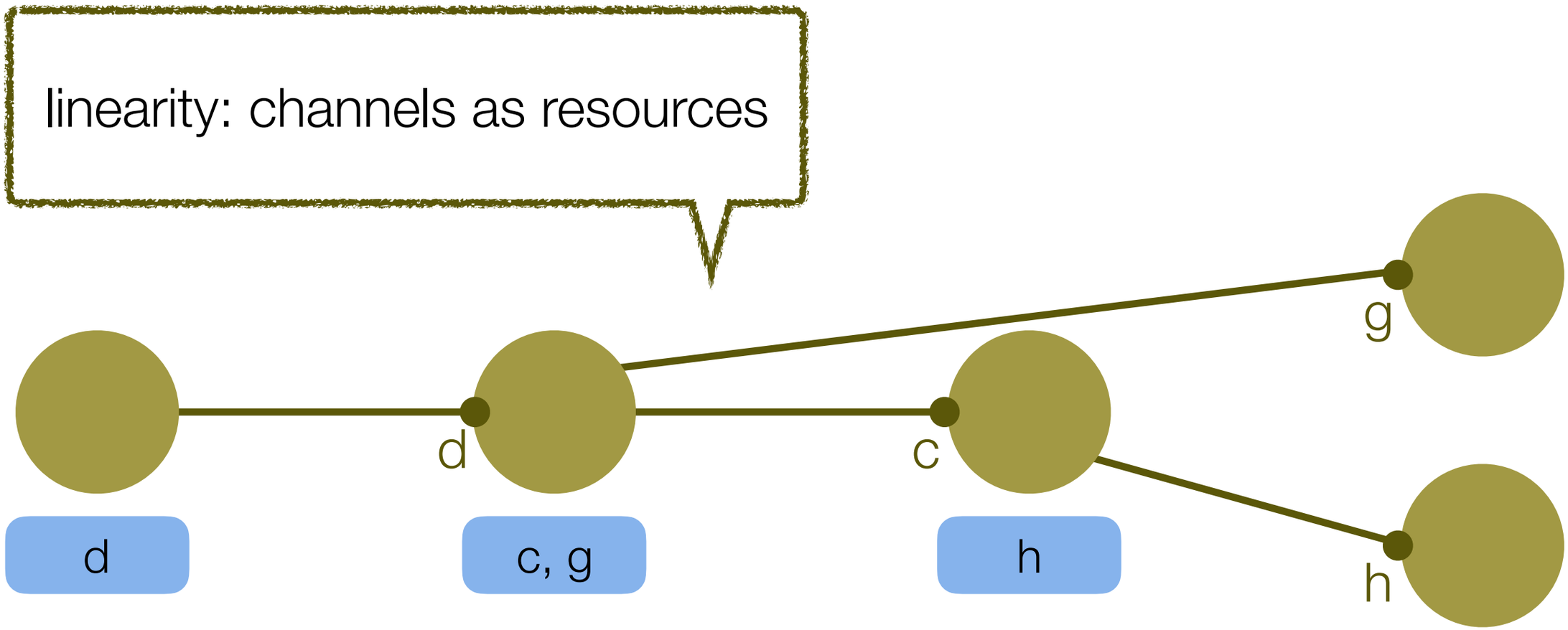


→ Processes form (dynamically changing) tree

[Based on intuitionistic linear sequent calculus]

Linear session-based communication

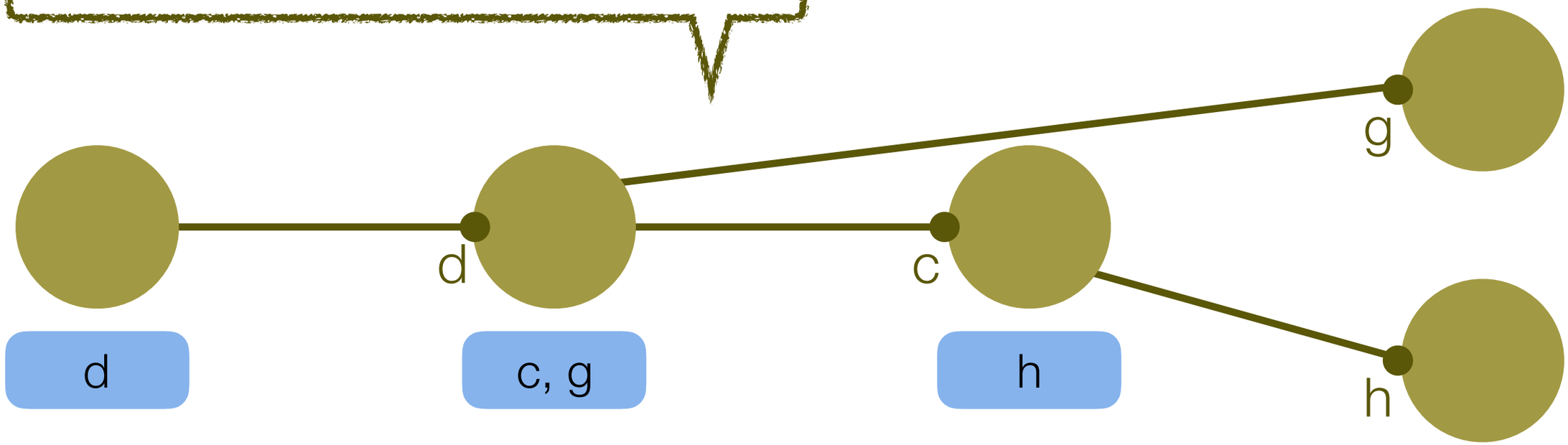
linearity: channels as resources



[Based on intuitionistic linear sequent calculus]

Linear session-based communication

linearity: channels as resources



Linearity guarantees session fidelity and freedom from data races and deadlock

[Based on intuitionistic linear sequent calculus]

Outline

Background: linear session types

Basic correspondence between CLOO - object-oriented concepts:

- Objects as processes
- Dynamic dispatch
- Structural subtyping

New forms of expression:

- Type-directed delegation
- Internal Choice

Conclusions

Session types in CLOO

```
typedef <?choice ctr> ctr; // external choice
choice ctr {
 <ctr>    Inc;             // increment value, continue
 <!int; > Val;             // send value, terminate
};
```

Session types in CLOO

```
typedef <?choice ctr> ctr;  // external choice
choice ctr {
 <ctr>     Inc;             // increment value, continue
 <!int; > Val;              // send value, terminate
};
```

choice

Process implementations in CLOO

session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

Process implementations in CLOO

session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

Process implementations in CLOO

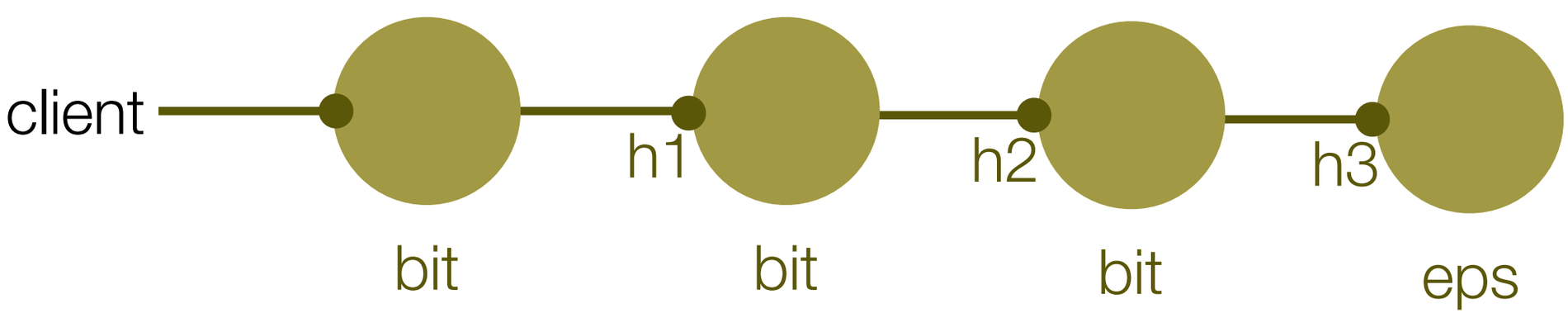
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

session type:

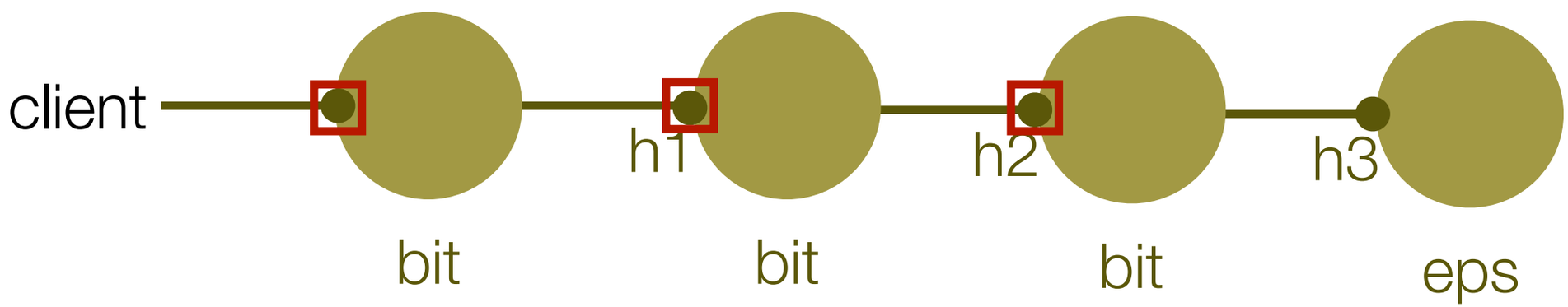
```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

offering channel



Process implementations in CLOO

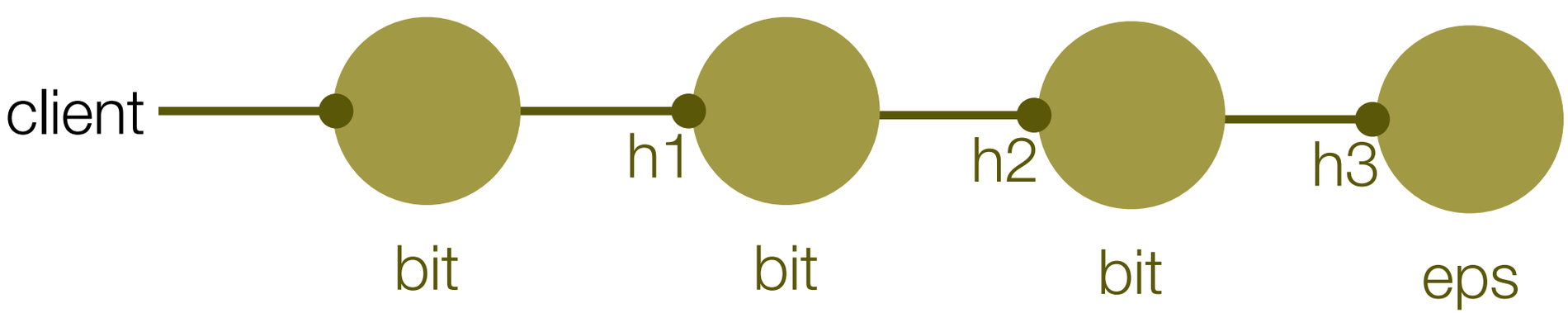
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

session type:

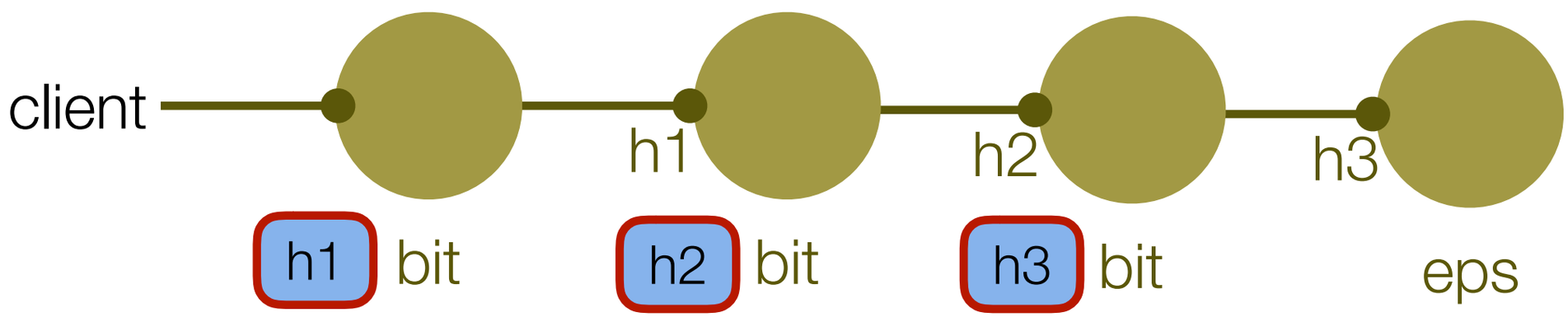
```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

channel resources



Process implementations in CLOO

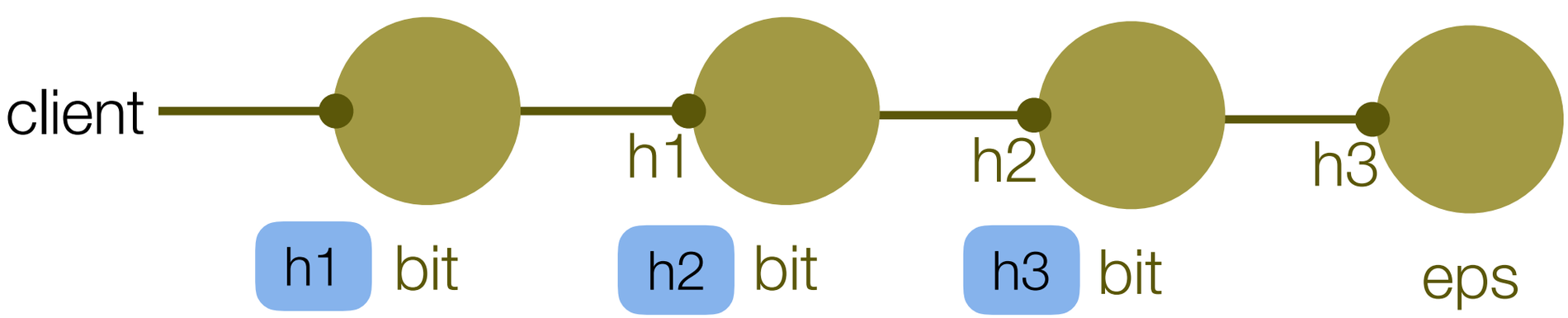
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

session type:

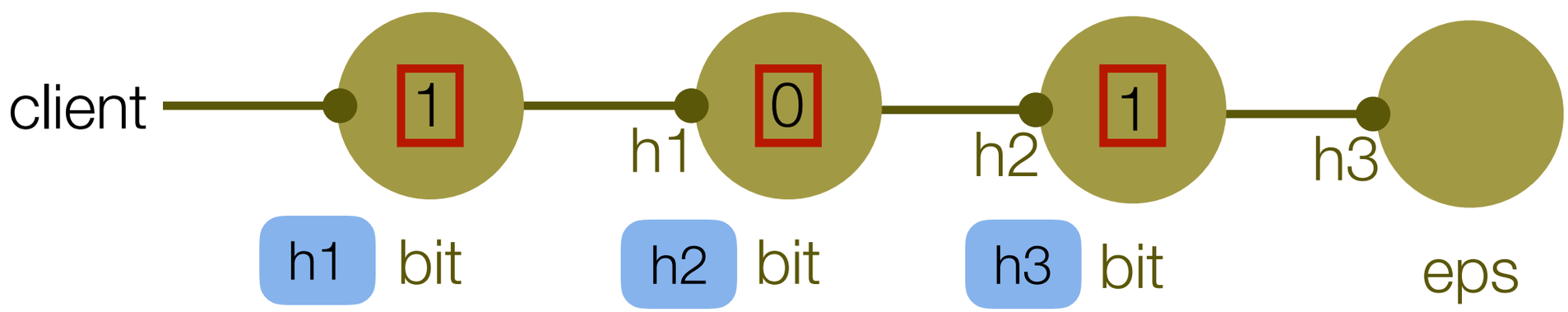
```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

local state



Process implementations in CLOO

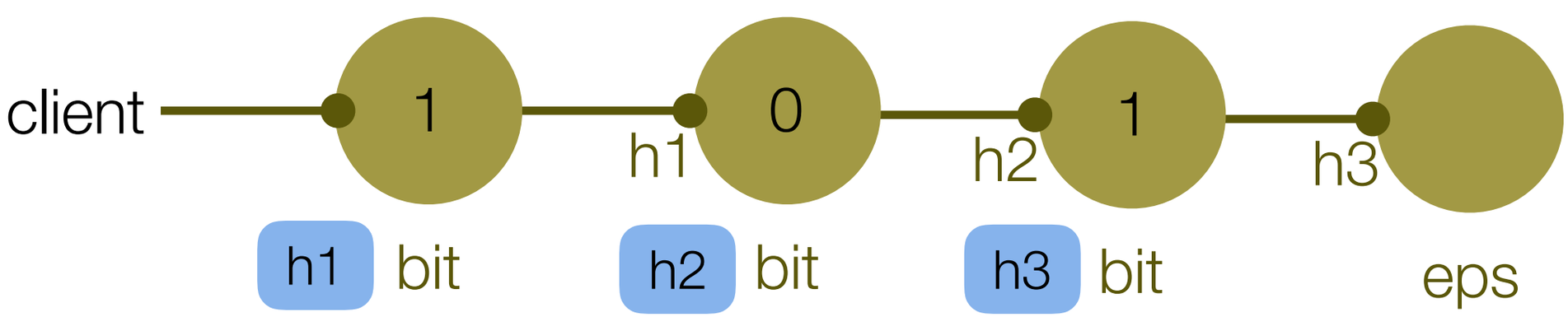
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

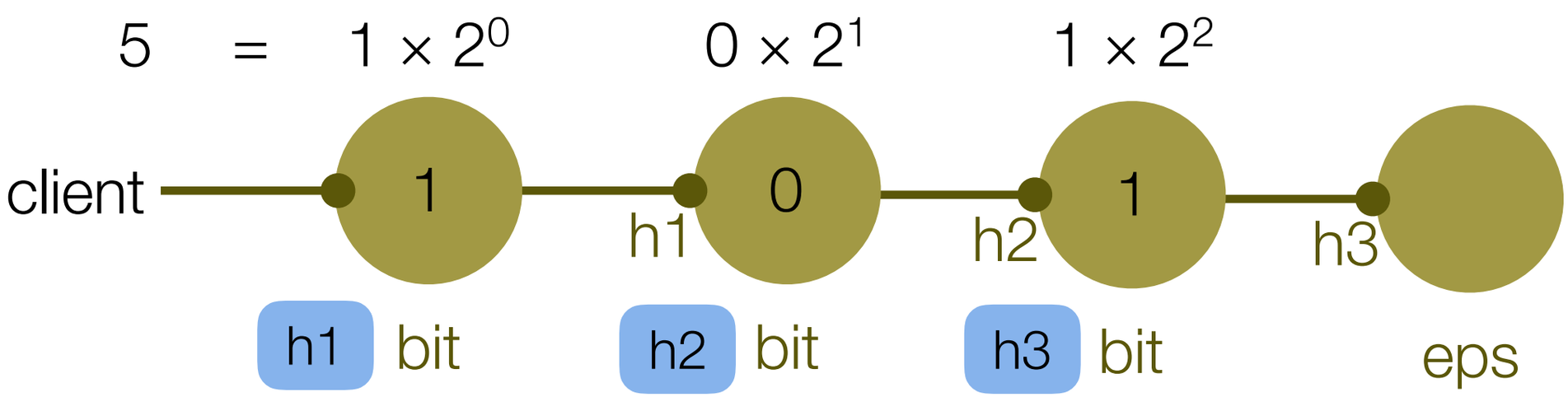
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

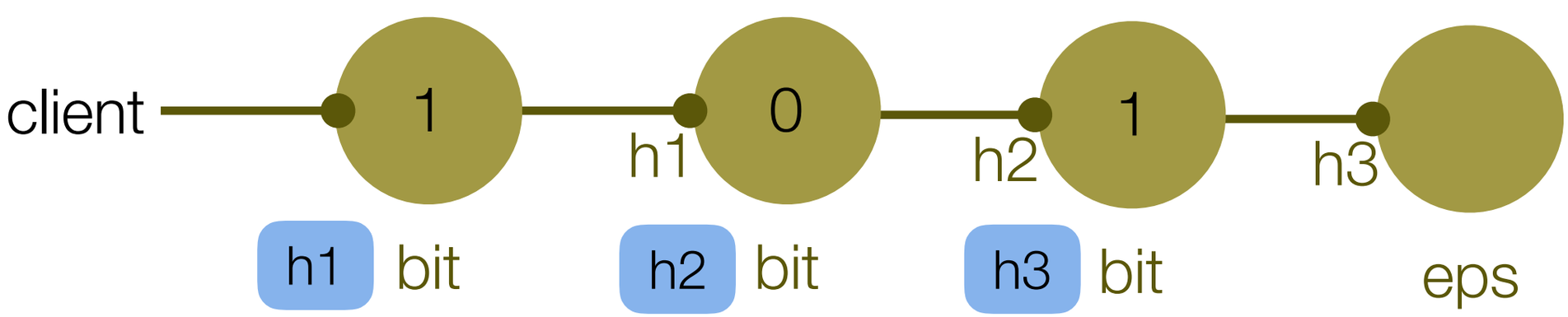
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

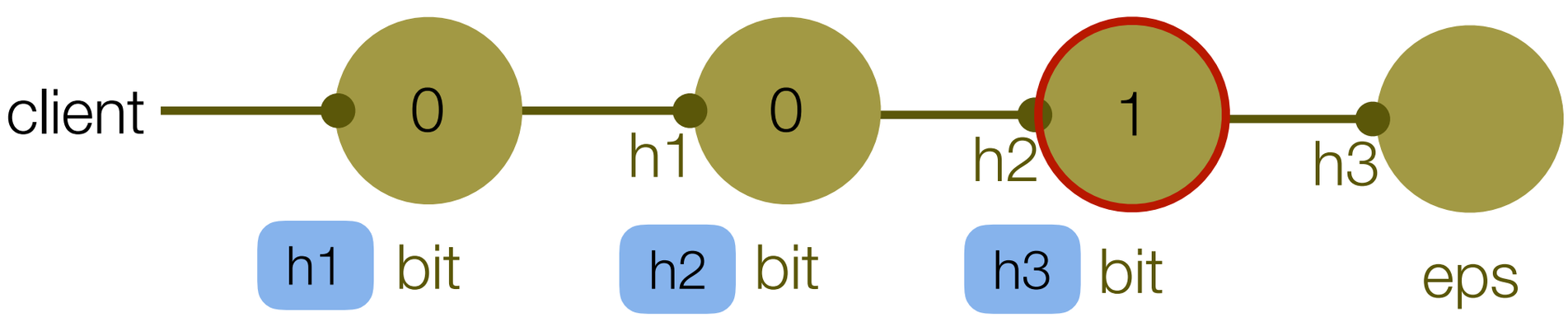
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



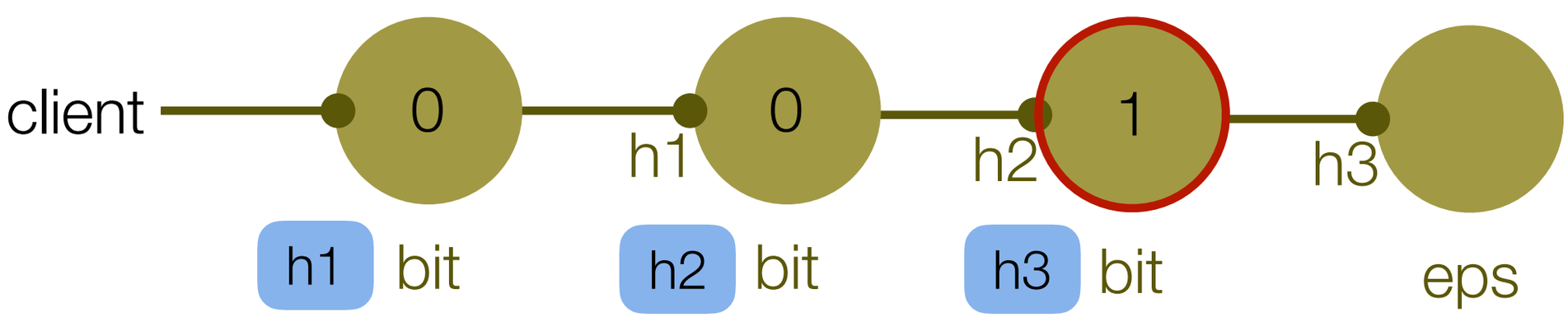
Process implementations in CLOO

session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

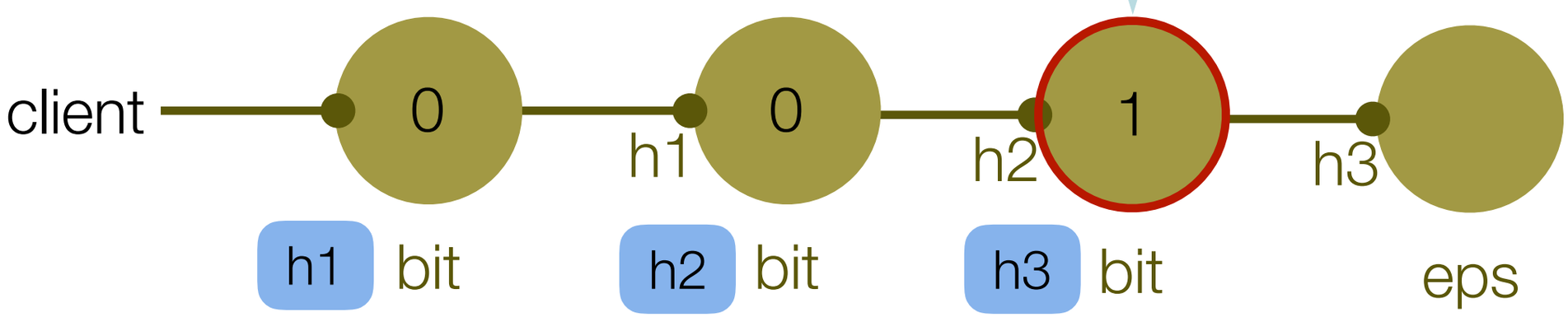
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
if (b == false) {
  b = true;
} else {
  $h.Inc;
  b = false;
}
```



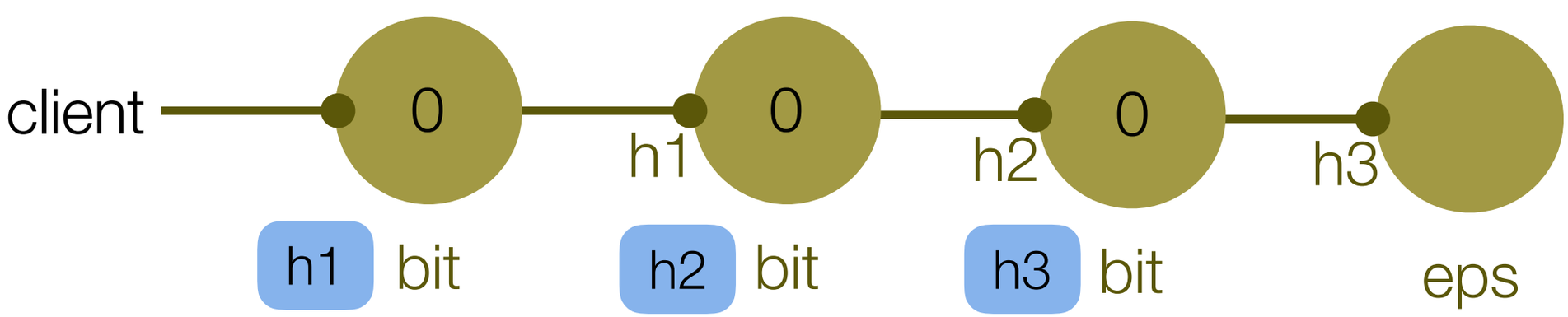
Process implementations in CLOO

session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

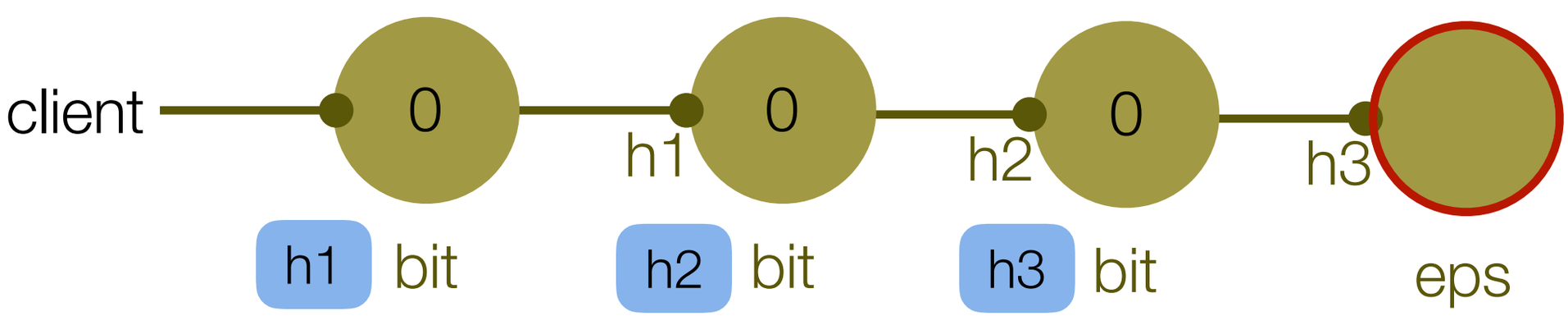
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l bit(bool b, ctr $h) {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



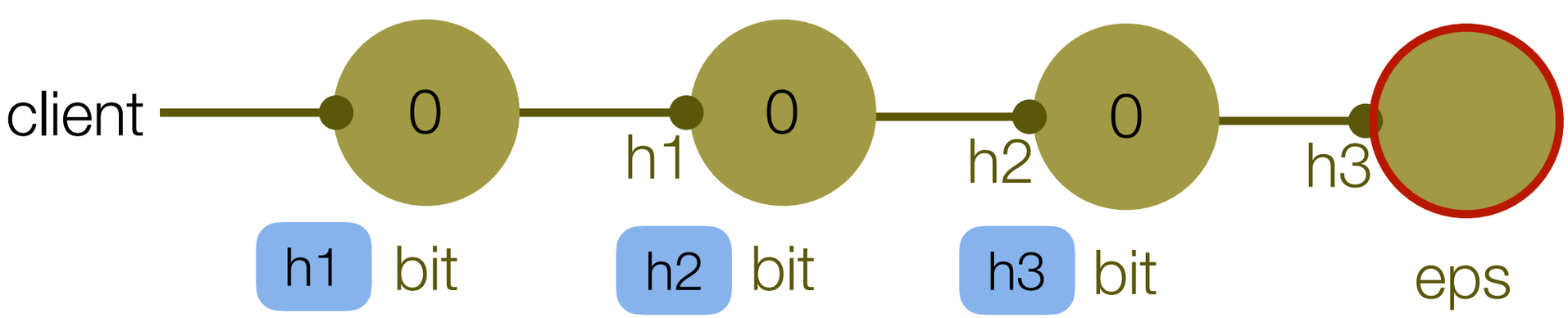
Process implementations in CLOO

session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

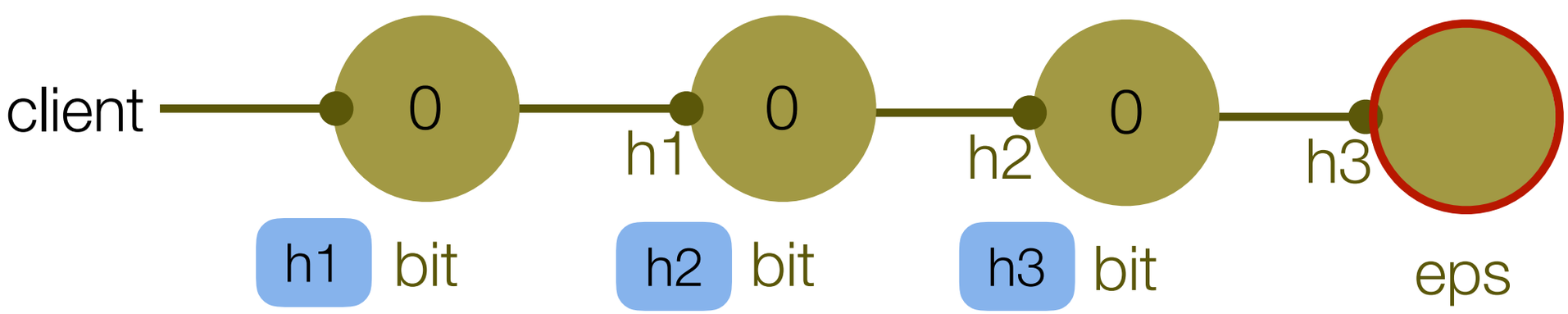
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $z = eps();
$l = bit(true, $z);
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

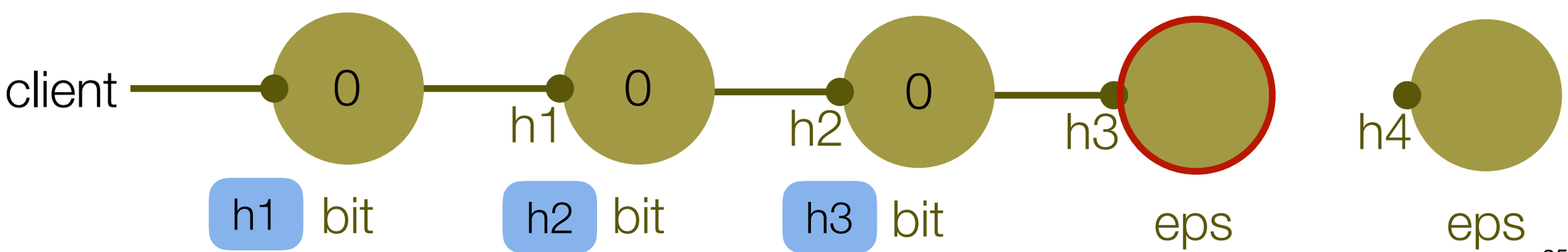
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $z = eps();
$l = bit(true, $z);
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

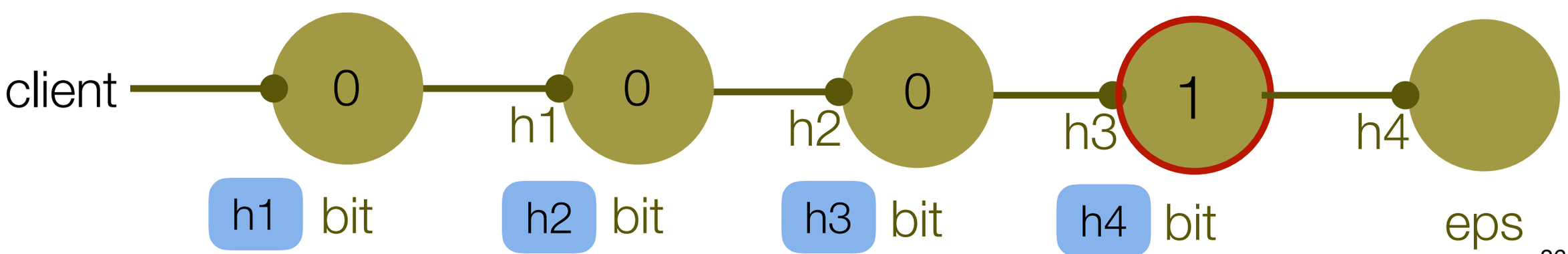
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $z = eps();
$l = bit(true, $z);
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Process implementations in CLOO

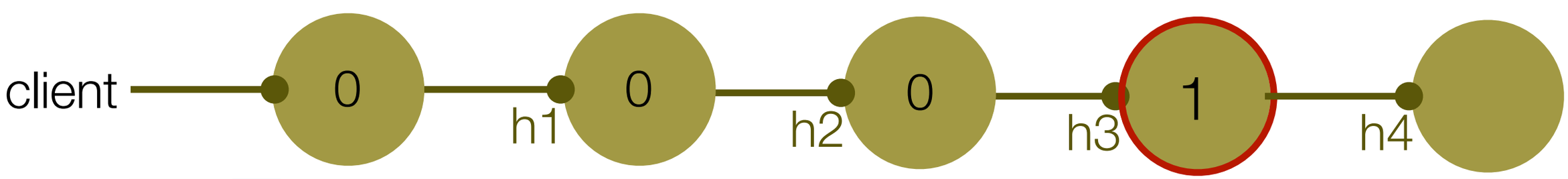
session type:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

process implementation as bit string:

```
ctr $z = eps();
$l = bit(true, $z);
```

```
ctr $l eps() {
  loop {
    switch ($l) {
      case Inc: … ;
      case Val: … ;
}}}
```



Dynamic dispatch of labels

Basic correspondence

- Objects as processes

A process has state and identity. Process state: local + protocol.

- Processes are encapsulated

Process local state can only be read or written by process.

Protocol state can only be changed by message exchange.

- Method invocations as receives of labels from external choice

Processes also permit sends of labels from internal choice.

- References as channels

Channels are bidirectional and linear. Ownership transfer possible.

Structural subtyping

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

Structural subtyping

supertype:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

subtype:

```
typedef <?choice ctr_inc2> ctr_inc2;
choice ctr_inc2 {
  <ctr_inc2>       Inc;
  <ctr_inc2>       Inc2;
  <!int; >         Val;
};
```

Structural subtyping

supertype:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

subtype:

```
typedef <?choice ctr_inc2> ctr_inc2;
choice ctr_inc2 {
  <ctr_inc2>       Inc;
  <ctr_inc2>       Inc2;
  <!int; >         Val;
};
```

ctr_inc2 is a subtype of ctr, it accepts at least same choices

Structural subtyping

supertype:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

subtype:

```
typedef <?choice ctr_inc2> ctr_inc2;
choice ctr_inc2 {
  <ctr_inc2>      Inc;
  <ctr_inc2>      Inc2;
  <!int; >        Val;
};
```

- ctr_inc2 is a subtype of ctr, it accepts at least same choices
- subtyping arises between external and internal choices

Outline

Background: linear session types

Basic correspondence between CLOO - object-oriented concepts:

- Objects as processes
- Dynamic dispatch
- Structural subtyping

New forms of expression:

- Type-directed delegation
- Internal Choice

Conclusions

Type-directed delegation

supertype:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

subtype:

```
typedef <?choice ctr_inc2> ctr_inc2;
choice ctr_inc2 {
  <ctr_inc2>      Inc;
  <ctr_inc2>      Inc2;
  <!int; >        Val;
};
```

Type-directed delegation

supertype:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

subtype:

```
typedef <?choice ctr_inc2> ctr_inc2;
choice ctr_inc2 {
  <ctr_inc2>      Inc;
  <ctr_inc2>      Inc2;
  <!int; >        Val;
};
```

process:

```
ctr_inc2 $c counter_inc2(ctr $d) {
  loop {
    switch ($c) {
      case Inc2: $d.Inc; $d.Inc; break;
      default:
        $c <=> $d; // type-directed delegation to ‘d'
}}}
```

Type-directed delegation

supertype:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

subtype:

```
typedef <?choice ctr_inc2> ctr_inc2;
choice ctr_inc2 {
  <ctr_inc2>      Inc;
  <ctr_inc2>      Inc2;
  <!int; >        Val;
};
```

process:

```
ctr_inc2 $c counter_inc2(ctr $d) {
  loop {
    switch ($c) {
      case Inc2: $d.Inc; $d.Inc; break;
      default:
        $c <=> $d; // type-directed delegation to ‘d'
}}}
```

Type-directed delegation

supertype:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

subtype:

```
typedef <?choice ctr_inc2> ctr_inc2;
choice ctr_inc2 {
  <ctr_inc2>      Inc;
  <ctr_inc2>      Inc2;
  <!int; >        Val;
};
```

inferable from session type declaration and subtyping relationship

```
nc2(ctr $d) {

    case Inc2: $d.Inc; $d.Inc; break;
    default:
      $c <=> $d; // type-directed delegation to ‘d'
}}}
```

Internal choice

bit string as external choice:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

bit string as internal choice:

```
typedef <!choice bits> bits;
choice bits {<> Eps; <!bool; bits> Bit;};
```

Internal choice

bit string as external choice:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

bit string as internal choice:

```
typedef <!choice bits> bits;
choice bits {<> Eps; <!bool; bits> Bit;};
```

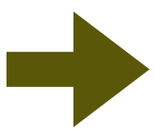
Internal choice

bit string as external choice:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

bit string as internal choice:

```
typedef <!choice bits> bits;
choice bits {<> Eps; <!bool; bits> Bit;};
```



Bit string is represented in terms of Bit and Eps messages, rather than bit and eps processes

External vs internal choice

bit string as external choice:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

```
ctr $l bit(bool b, ctr $h) {
 …
}
```

```
ctr $l eps() {
  …
}
```

bit string as internal choice:

```
typedef <!choice bits> bits;
choice bits {<> Eps; <!bool; bits> Bit;};
```

```
bits $succ inc(bits $ctr) {…}
```

```
<!int;> $val val(bits $ctr) {…}
```

```
bits $zero zero() {…}
```

External vs internal choice

bit string as external choice:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

```
ctr $l bit(bool b, ctr $h) {
  …
}
```

```
ctr $l eps() {
  …
}
```

bit string as internal choice:

```
typedef <!choice bits> bits;
choice bits {<> Eps; <!bool; bits> Bit;};
```

```
bits $succ inc(bits $ctr) {…}
```

```
<!int;> $val val(bits $ctr) {…}
```

```
bits $zero zero() {…}
```

External vs internal choice

bit string as external choice:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

```
ctr $l bit(bool b, ctr $h) {
 …
}
```

```
ctr $l eps() {
  …
}
```

bit string as internal choice:

```
typedef <!choice bits> bits;
choice bits {<> Eps; <!bool; bits> Bit;};
```

External vs internal choice

bit string as external choice:

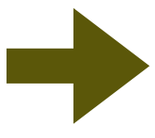
```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

```
ctr $l bit(bool b, ctr $h) {
 …
}
```

```
ctr $l eps() {
  …
}
```

bit string as internal choice:

```
typedef <!choice bits> bits;
choice bits {<> Eps; <!bool; bits> Bit;};
```



Lead to different program modularization

External vs internal choice

bit string as external choice:

```
typedef <?choice ctr> ctr;
choice ctr {<ctr> Inc; <!int; > Val;};
```

```
ctr $l bit(bool b, ctr $h) {
 …
}
```

```
ctr $l eps() {
  …
}
```

bit string as internal choice:

```
typedef <!choice bits> bits;
choice bits {<> Eps; <!bool; bits> Bit;};
```

→ Lead to different program modularization

→ External choice facilitates addition of new variants, internal choice addition of new operations

Conclusions

A fresh look at object-oriented programming

- accommodates existing and new object-oriented features
- inherently concurrent and protocol-aware

Ongoing work:

- compiler support of subtyping and type-directed delegation
- extend formalization

Future work:

- polymorphism for generic data structures
- affine and shared channels
- shared channels combined with traditional locking primitives