2-Phase Commit Among Strangers

Secure Distributed Escrow Exchange in 43 lines of JavaScript

Mark S. Miller (Google)
Tom Van Cutsem (VUB)
Tyler Close (Google)

Dr. SES
Distributed Resilient Secure EcmaScript

Secure: SES, object-capability subset of JS

- Unforgeable object references as capabilities

Distributed: Q, async distributed promises

- Communicating event loop concurrency
- Stretch reference graph between event loops & machines

Resilient: Ken, "Output Valid Rollback Recovery"

- Exists for Java, Scala, and C, not yet for JavaScript

Dr. SES
Distributed Resilient Secure EcmaScript

Secure: SES, object-capability subset of JS

- Unforgeable object references as capabilities

Distributed: Q, async distributed promises

- Communicating event loop concurrency
- Stretch reference graph between event loops & machines

Resilient: Ken, "Output Valid Rollback Recovery"

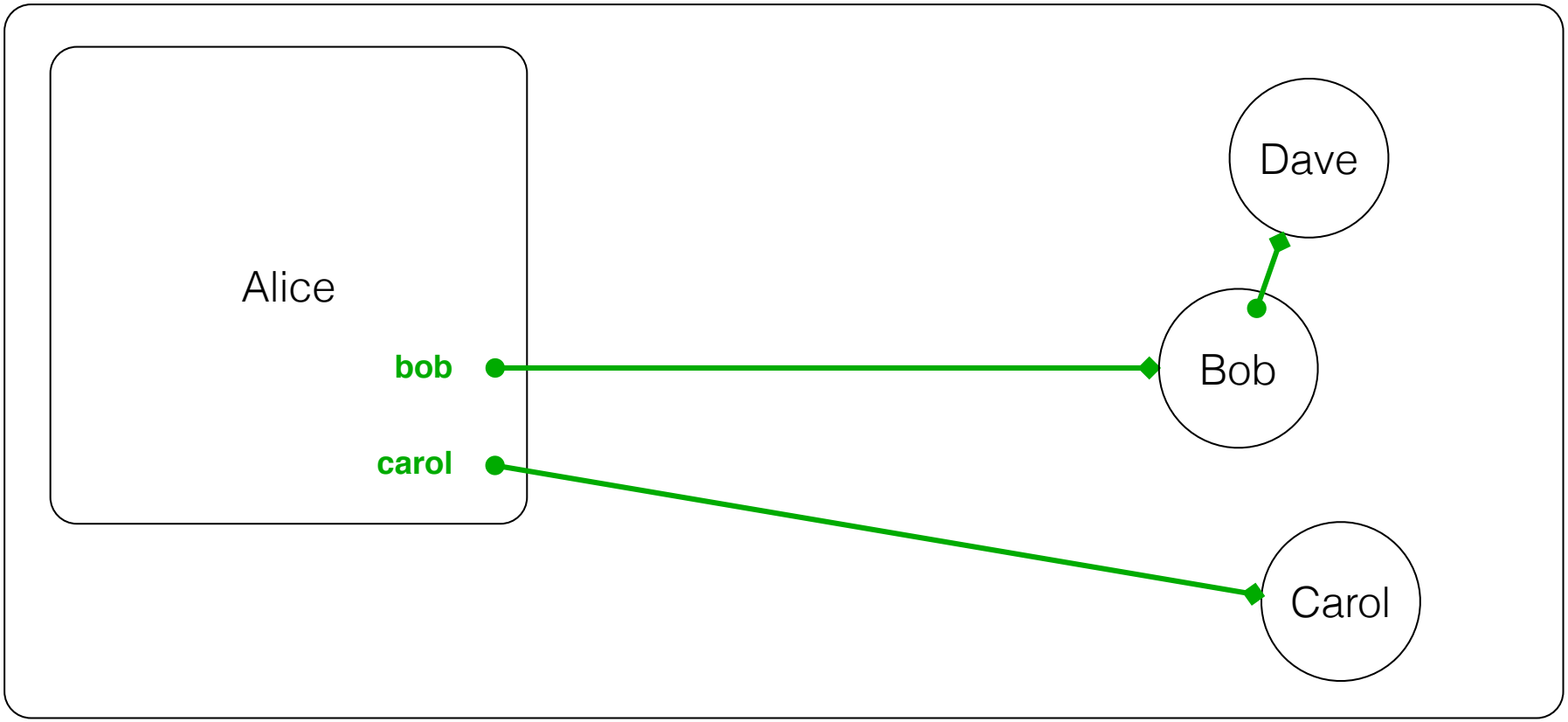
- Exists for Java, Scala, and C, not yet for JavaScript

Q primitives

var *daveP* = bobP ! foo(carol)	Eventual send
Q.defer()	Makes promise/resolver pair
Q(daveP).when(callback, errback)	The when-catch expression

Immediate call

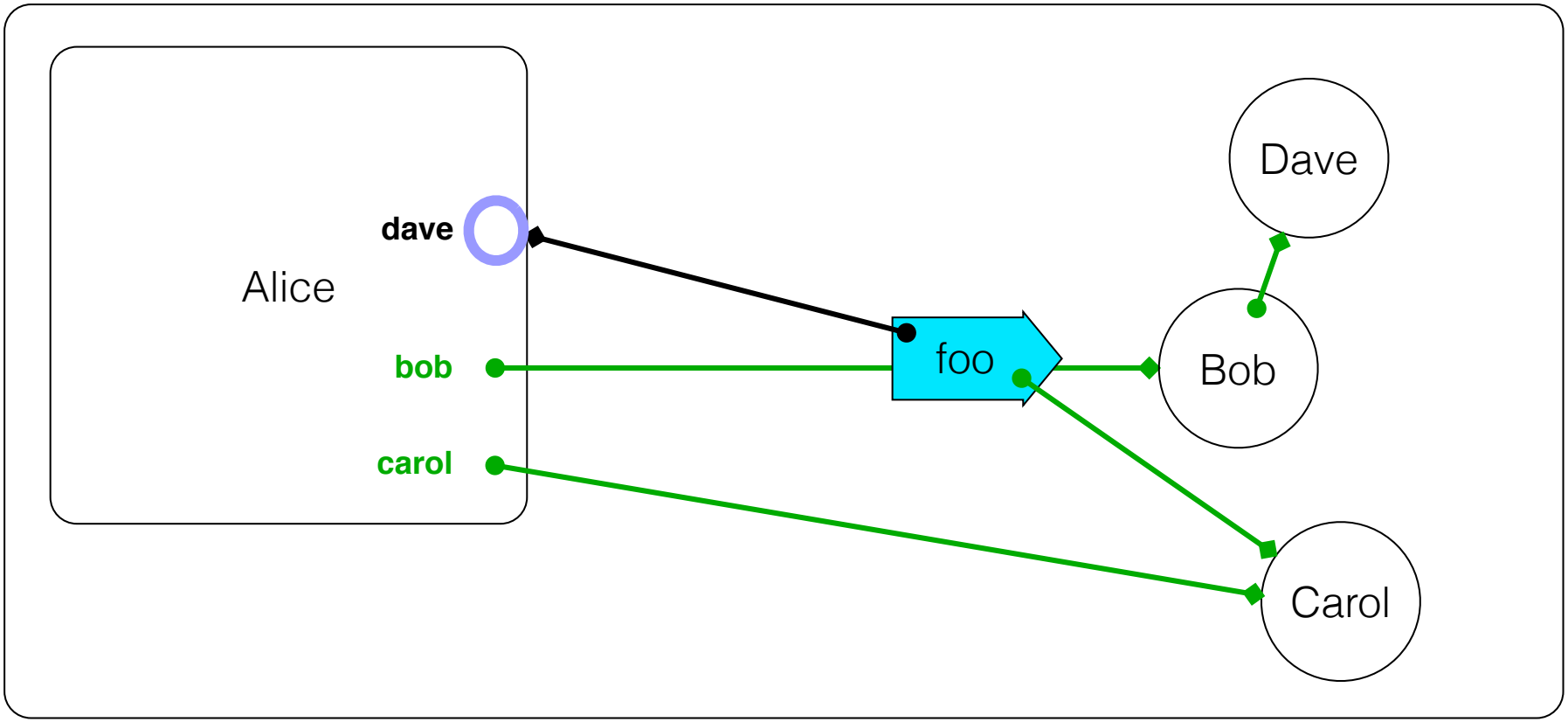
Everything local



Immediate call

var *dave* = bob.foo(carol);

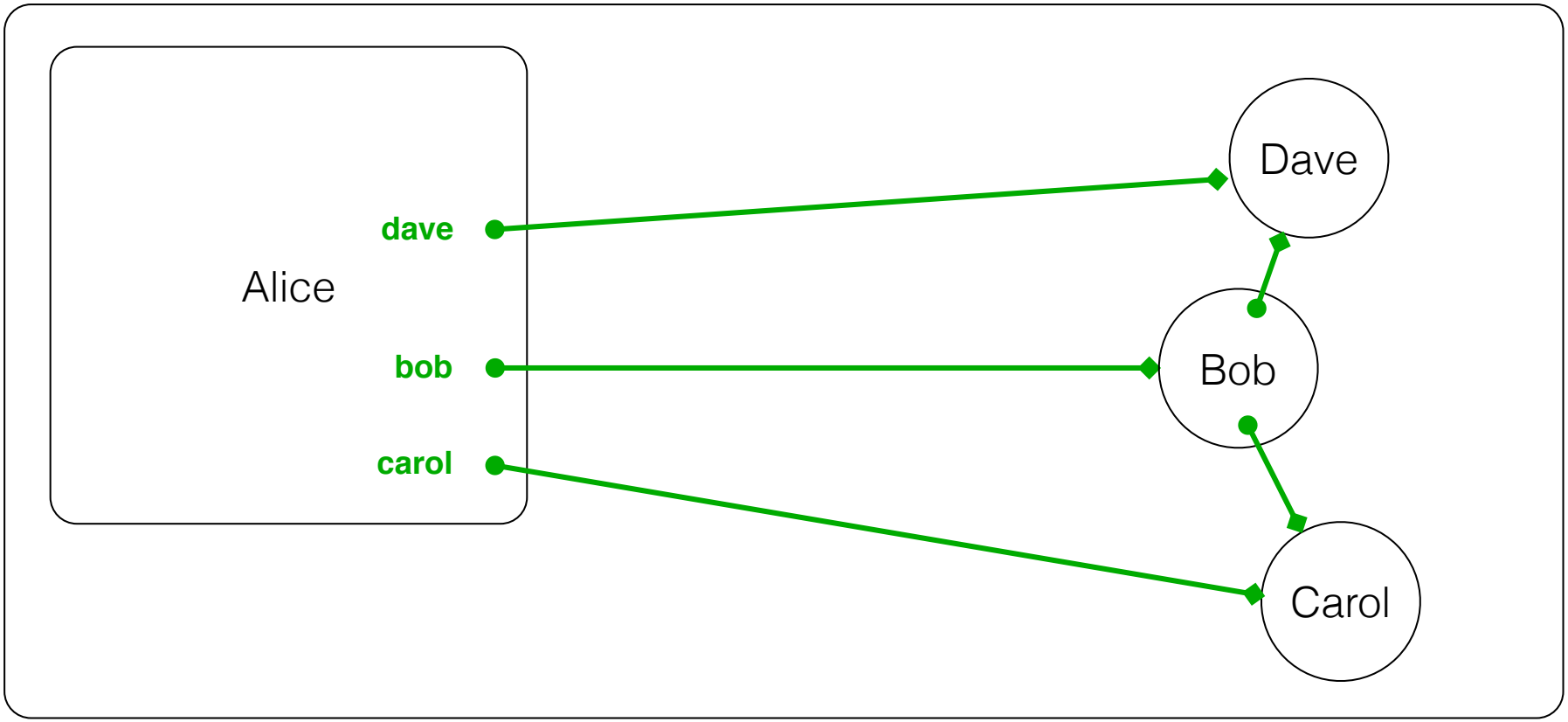
Local-only immediate call



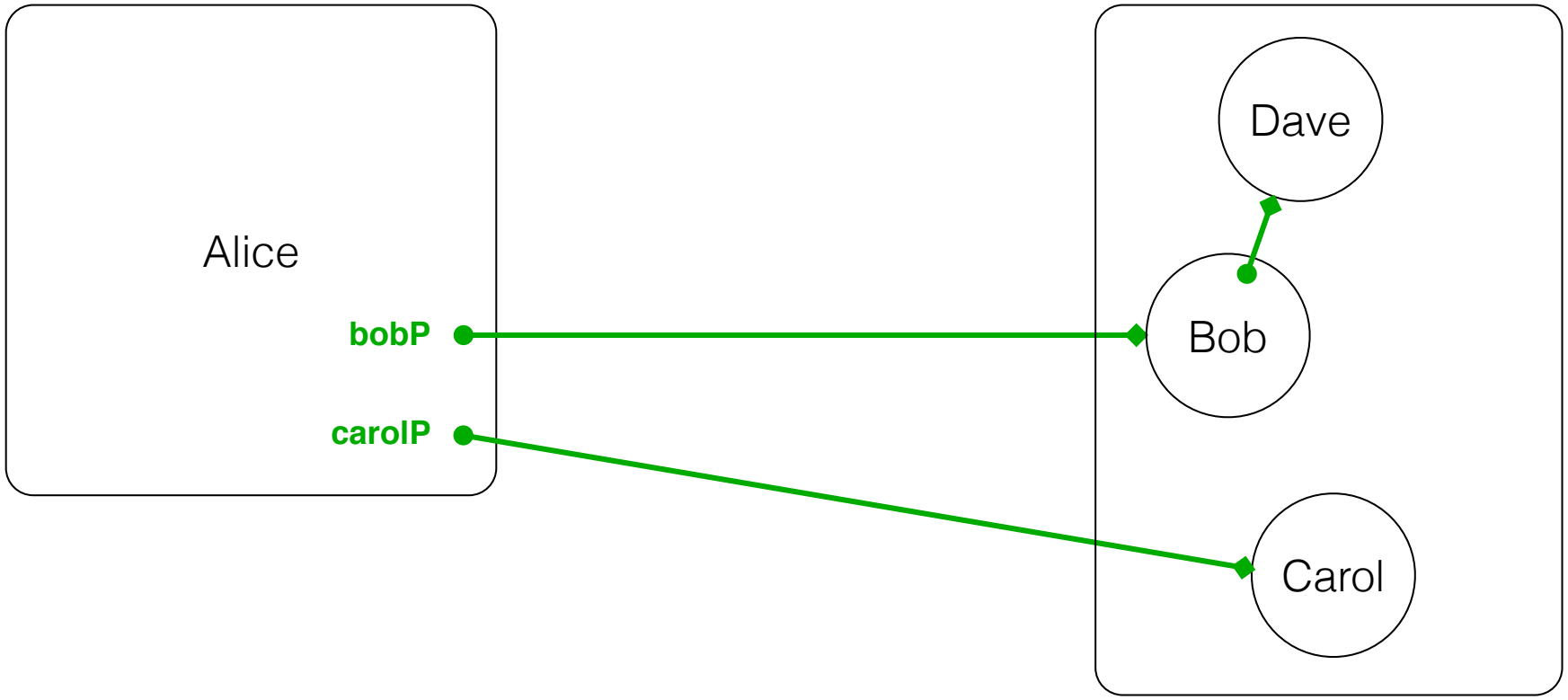
Immediate call

var *dave* = bob.foo(carol);

Success case

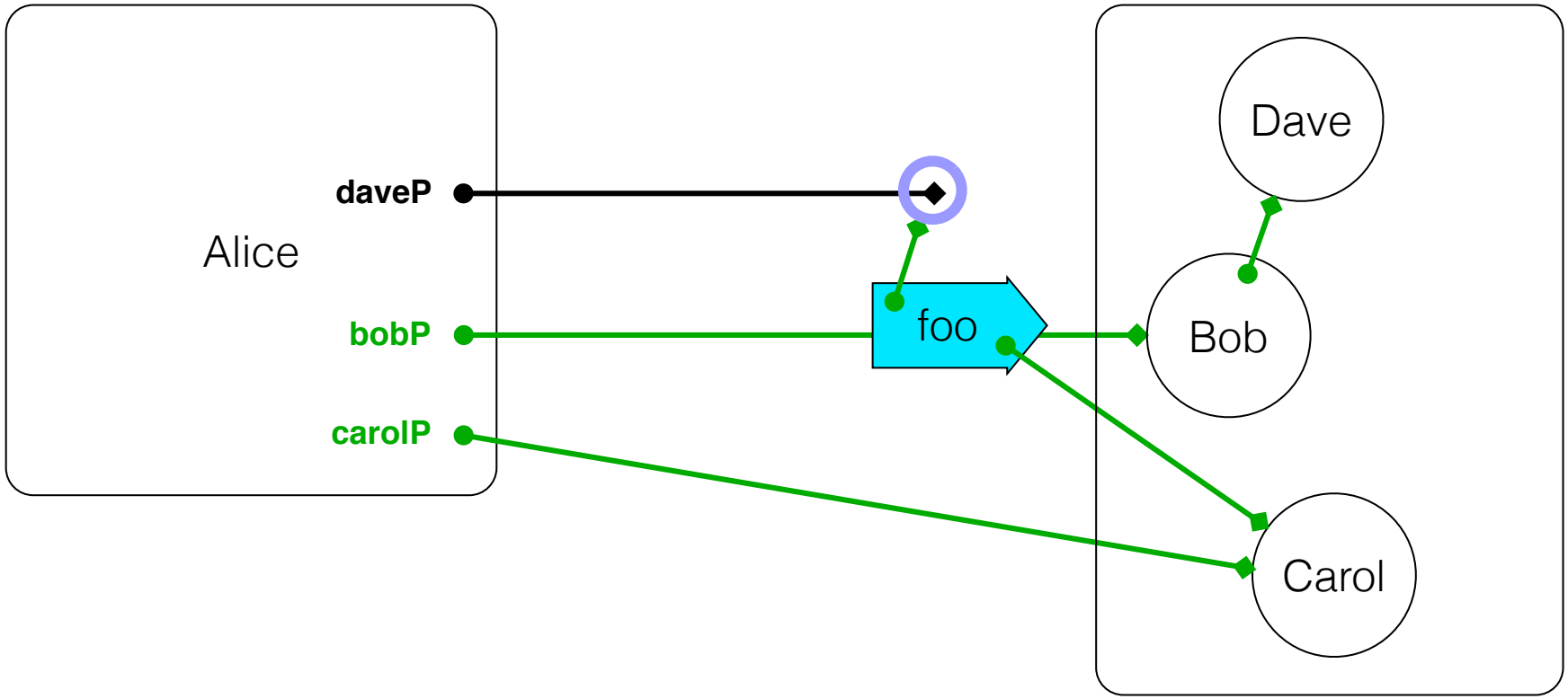


Eventual Send



Eventual Send

var *daveP* = bobP ! foo(carolP); Eventual send

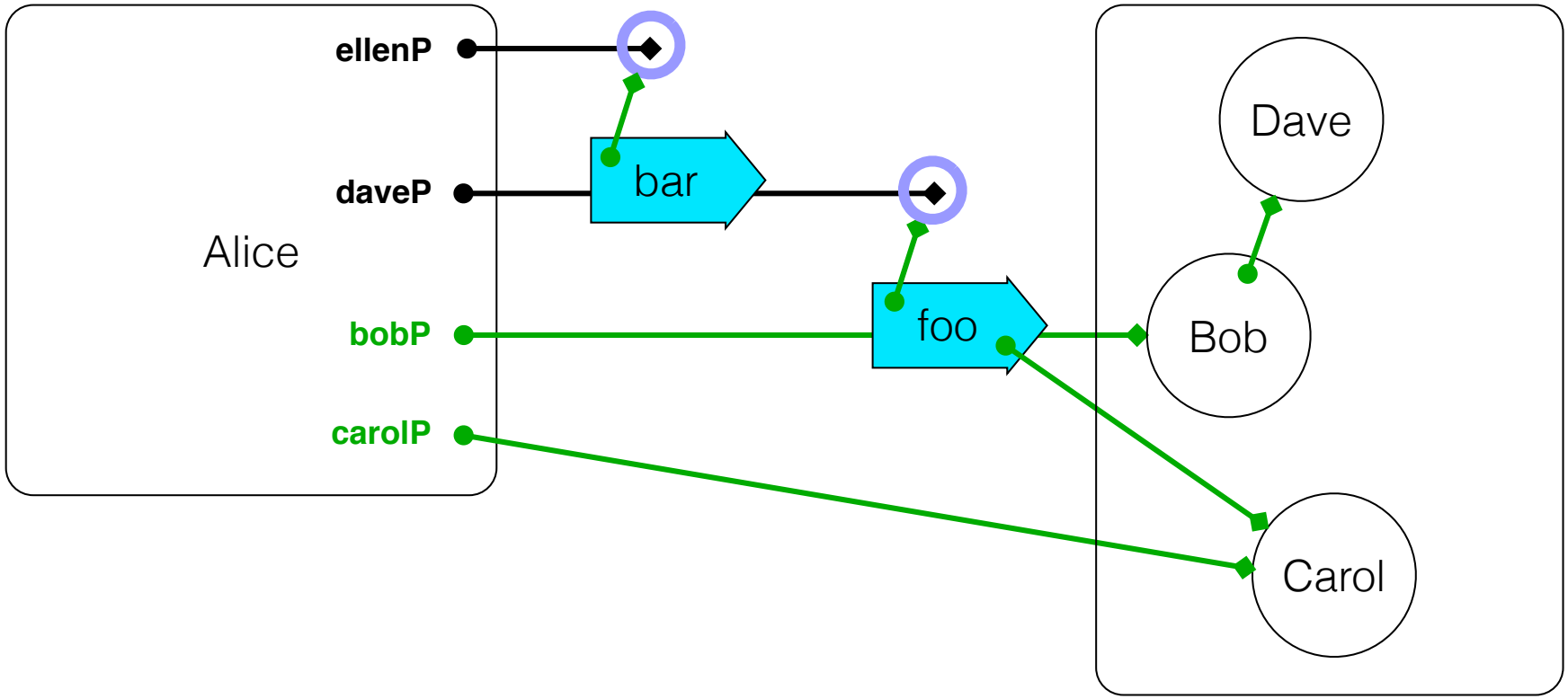


Eventual Send

var *daveP* = bobP ! foo(carolP);

var *ellenP* = daveP ! bar();

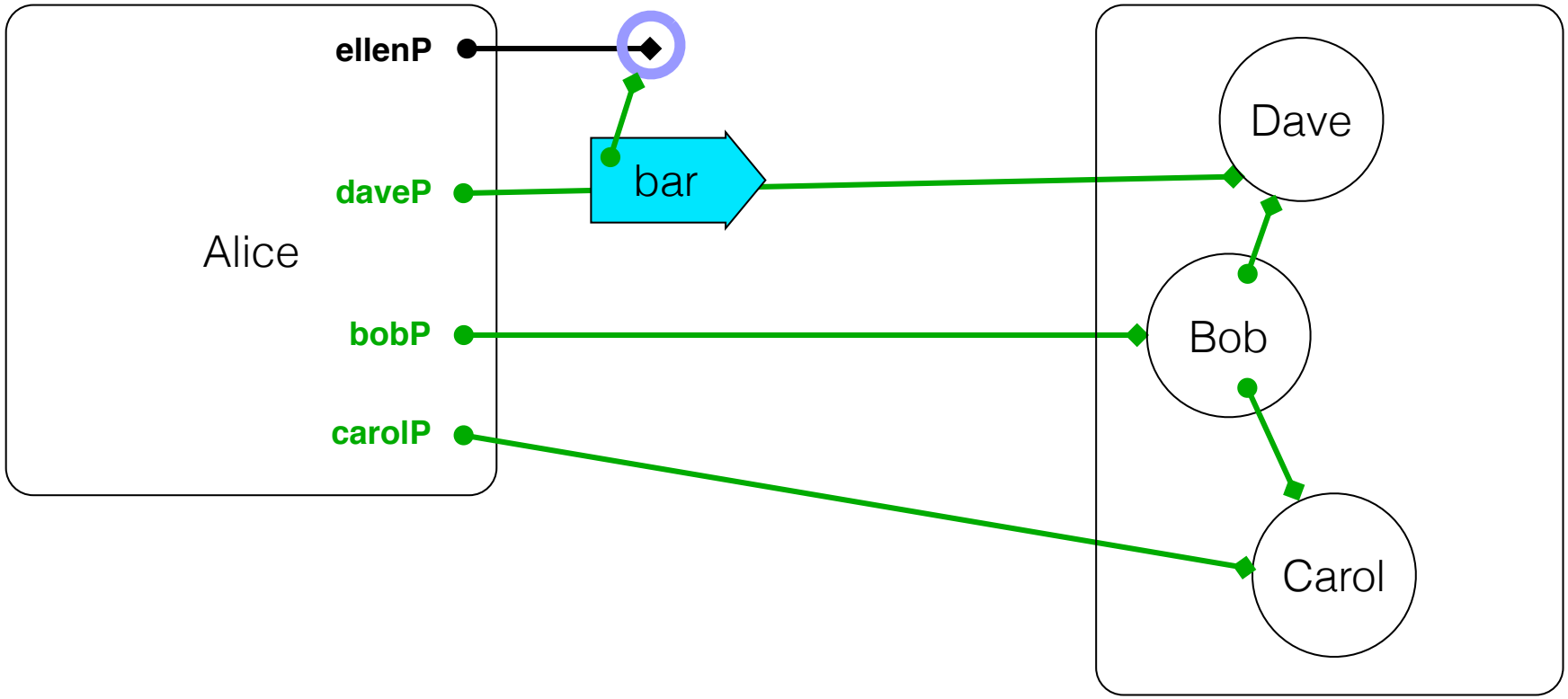
Promise pipelining



Eventual Send

```
var daveP = bobP ! foo(carolP);
var ellenP = daveP ! bar();
```

Success case

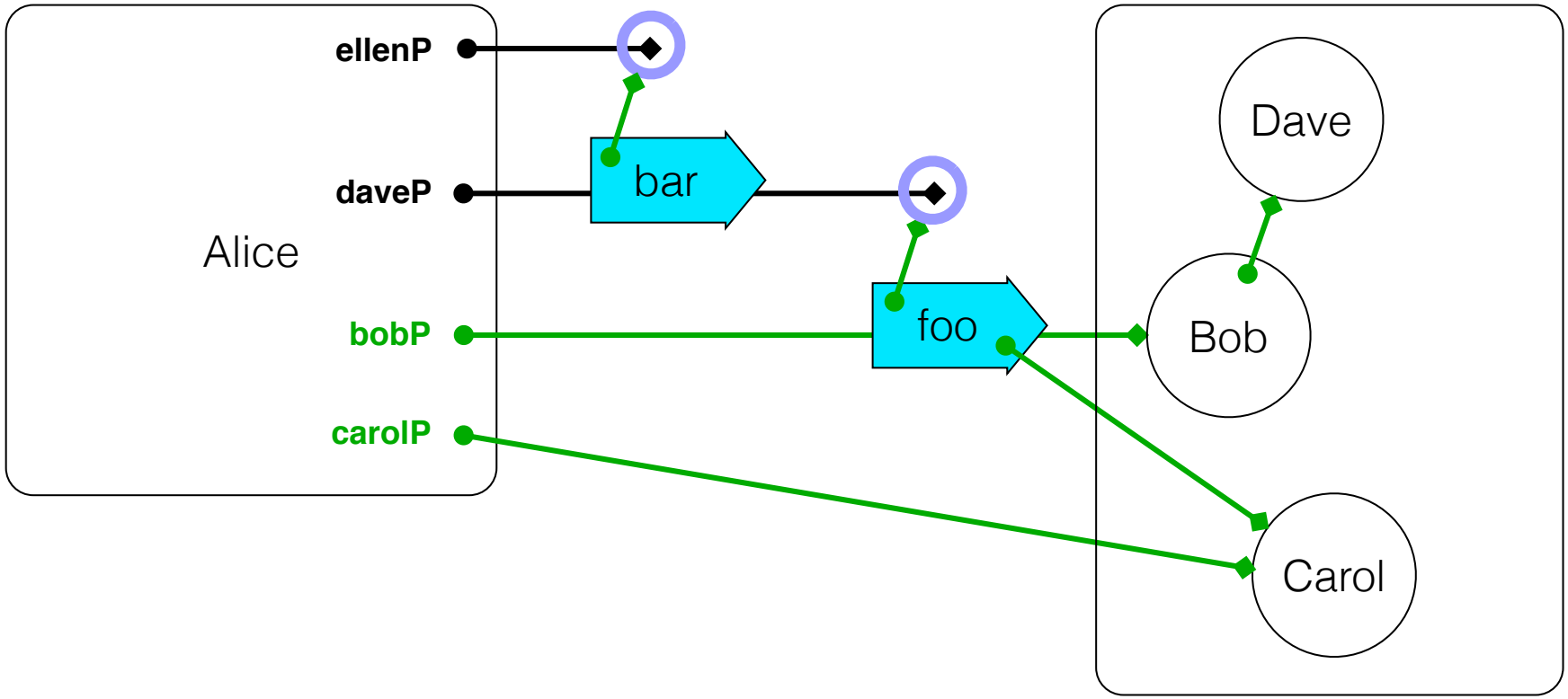


Eventual Send

var *daveP* = bobP ! foo(carolP);

var *ellenP* = daveP ! bar();

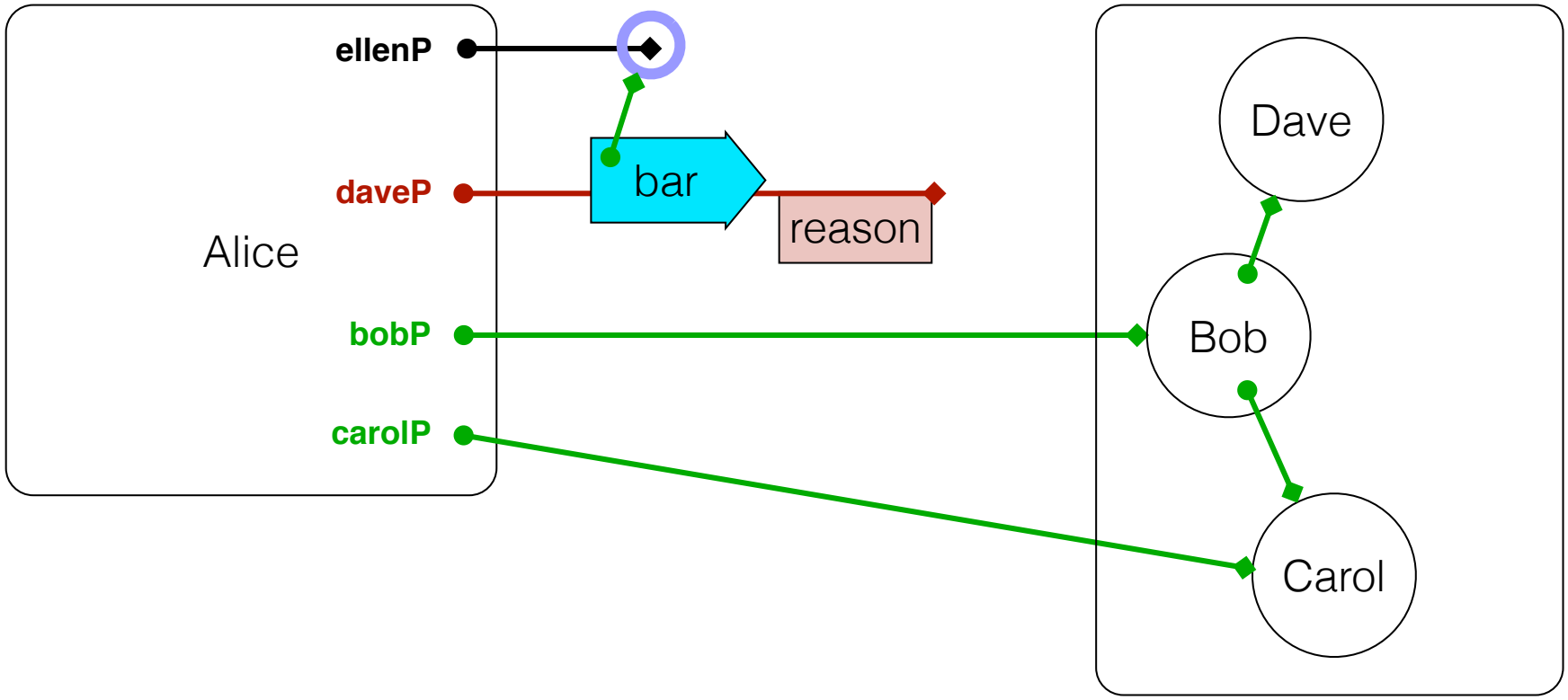
Failure case



Eventual Send

```
var daveP = bobP ! foo(carolP);
var ellenP = daveP ! bar();
```

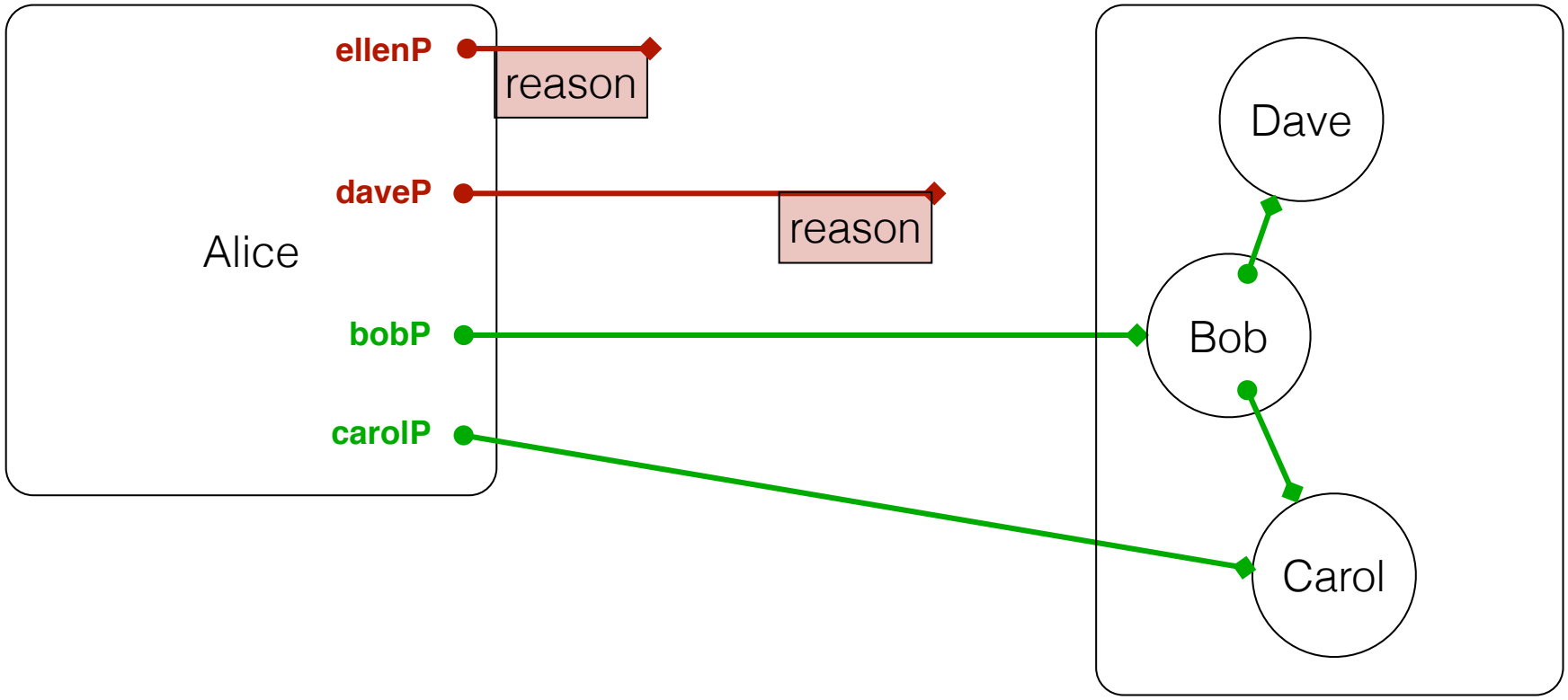
Failure case



Eventual Send

```
var daveP = bobP ! foo(carolP);
var ellenP = daveP ! bar();
```

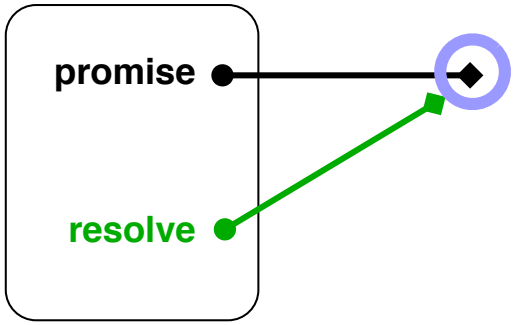
Broken promise contagion



Q.defer()

```
var deferred = Q.defer();
var promise = deferred.promise;
var resolve = deferred.resolve;
```

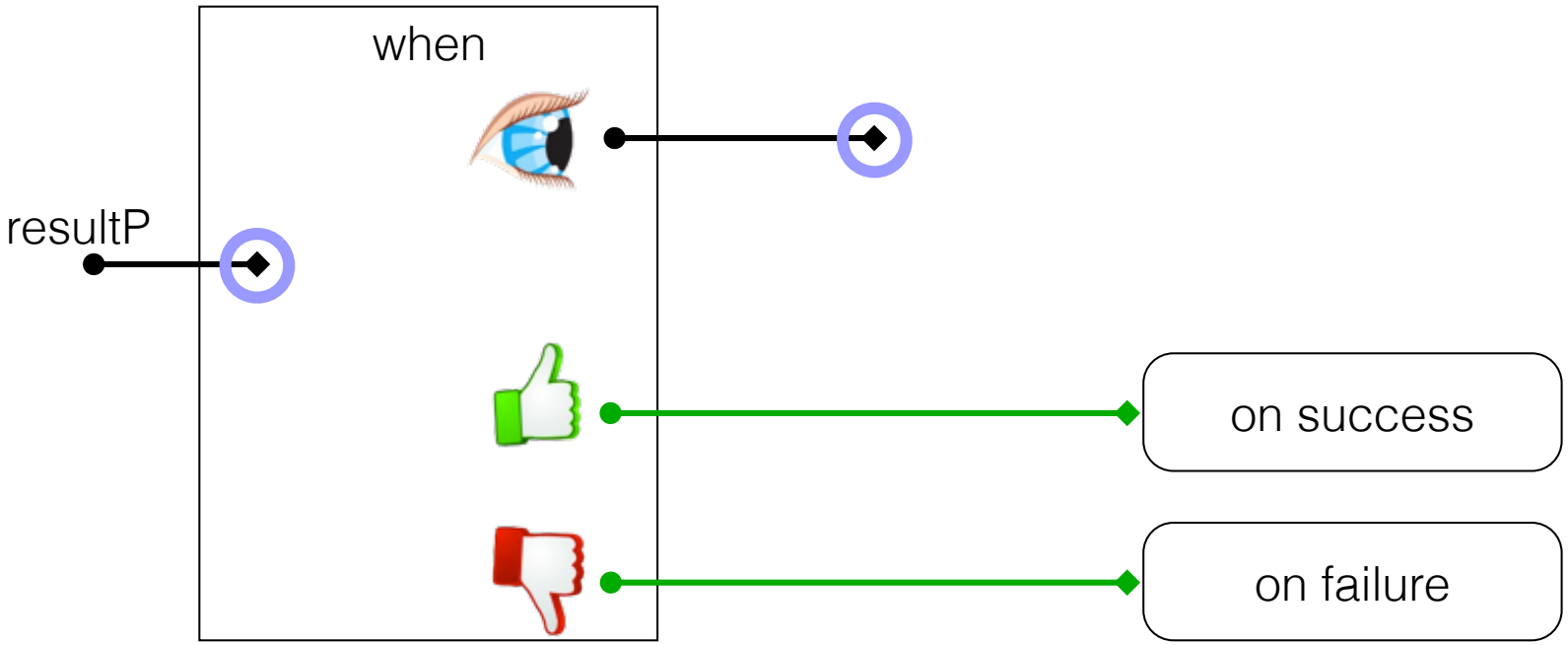
Makes promise/resolver pair



The when-catch expression

```
var resultP = Q(valP).when(
   function(val)  { ...val... },
   function(reason) { ...reason... });
```

Register for notification

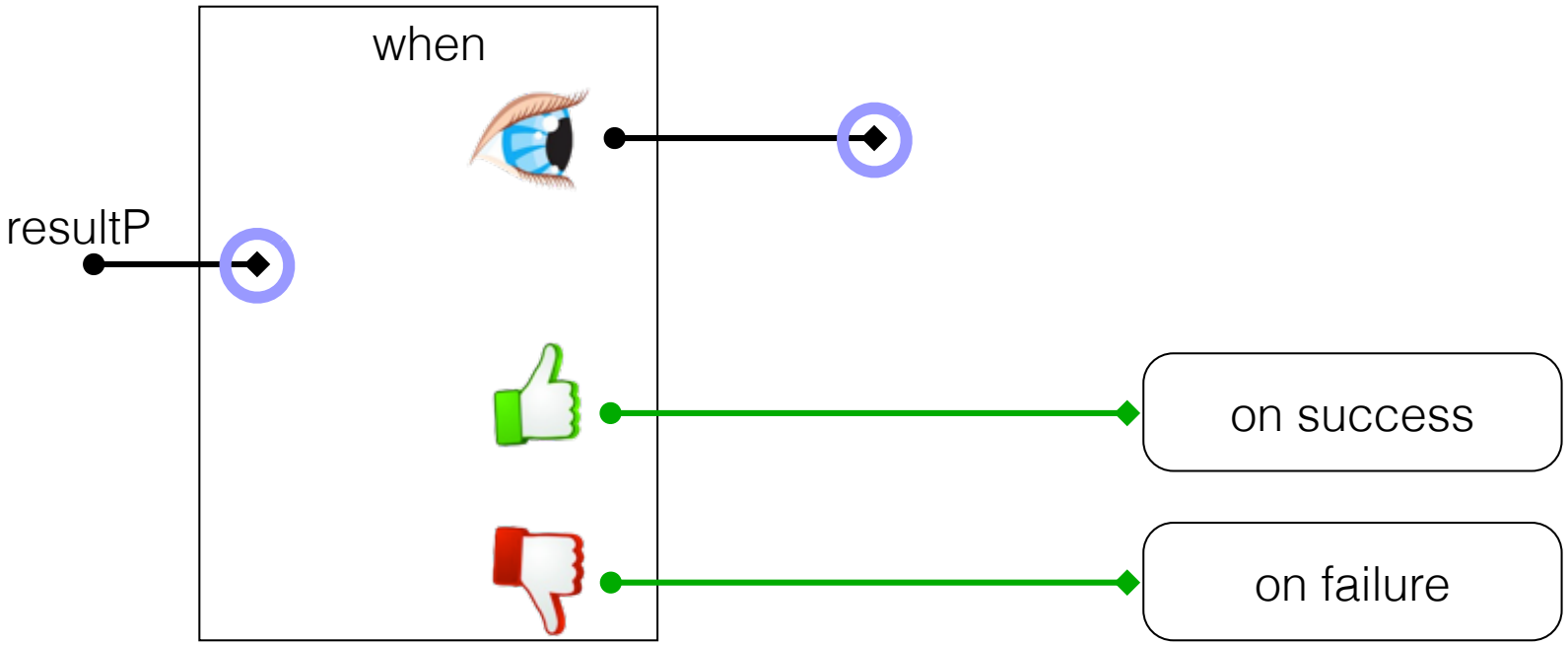


The when-catch expression

```
var resultP = Q(valP).when(function(val) {
    ...val...
}, function(reason) {
    ...reason...
});
```

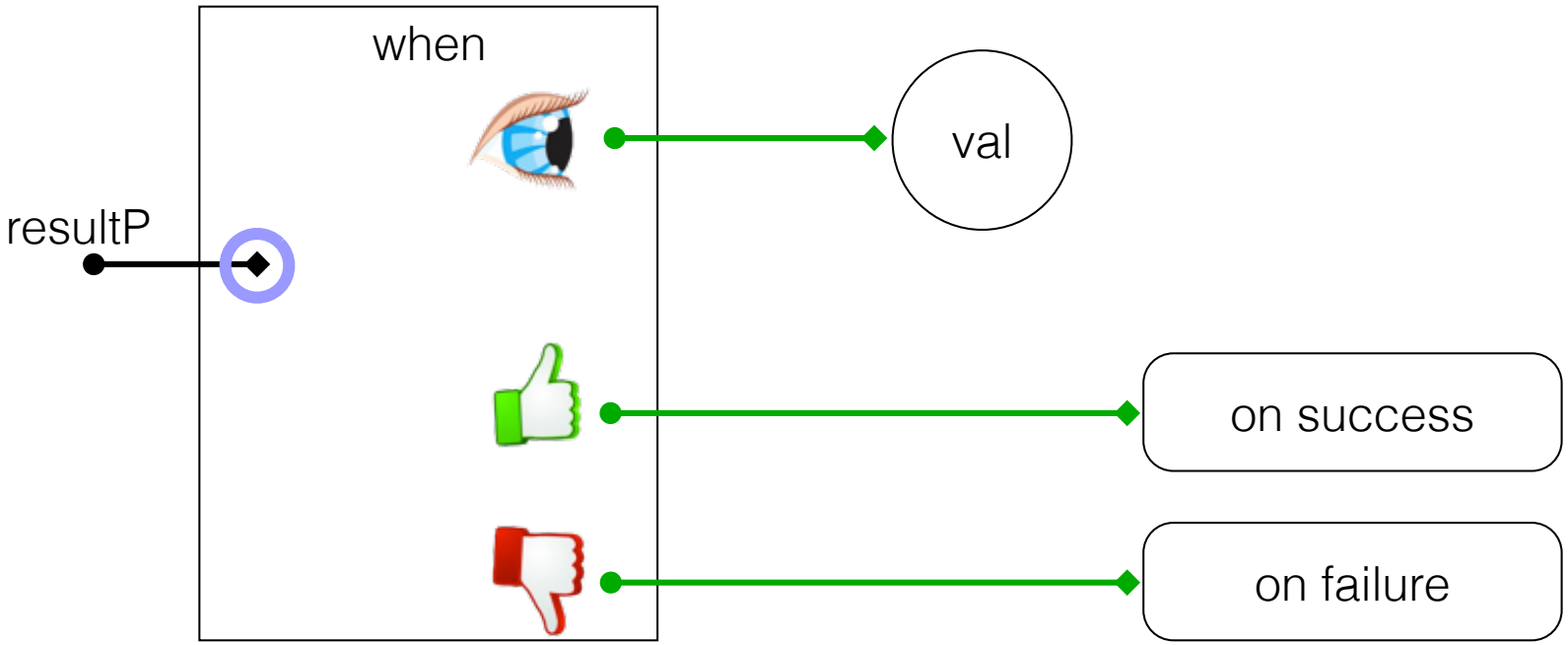
Register for notification

Try-catch-like layout



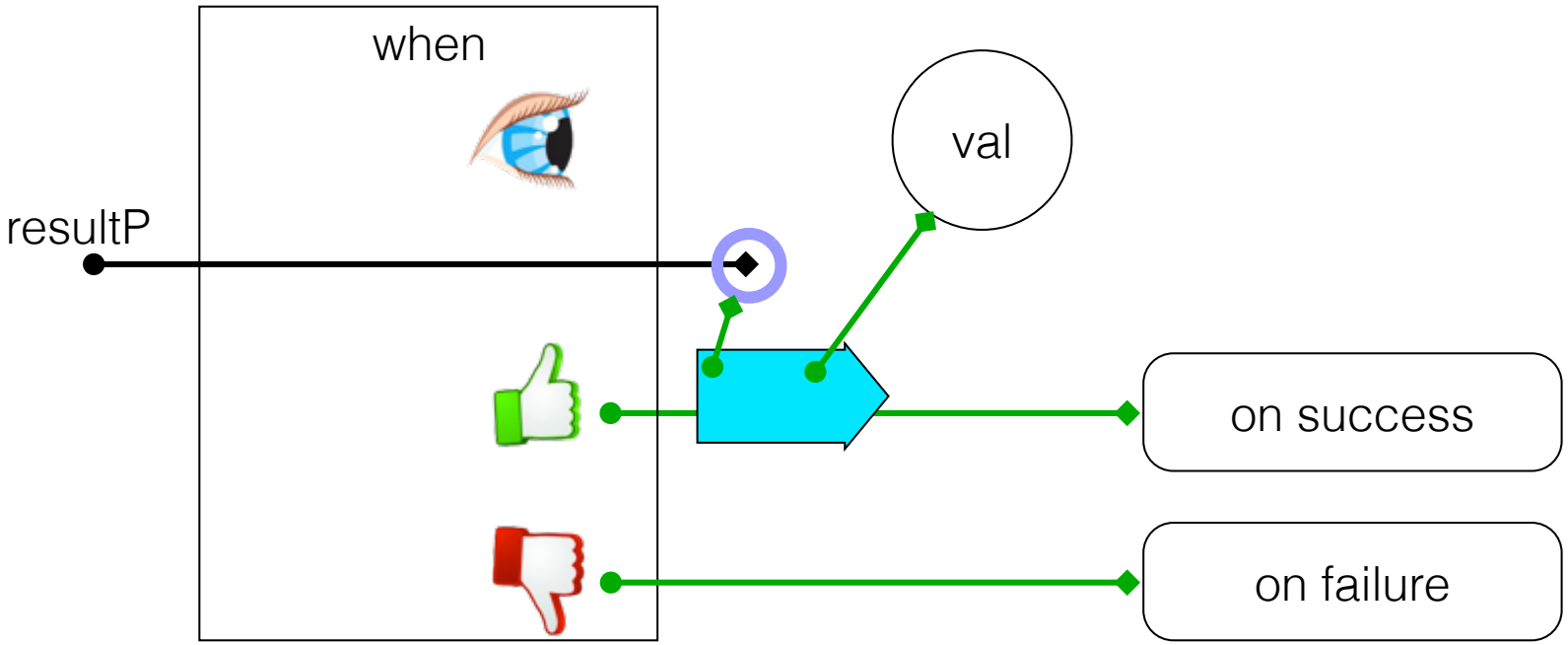
The when-catch expression

```
var resultP = Q(valP).when(function(val) {    Success case
    ...val...
}, function(reason) {
    ...reason...
});
```



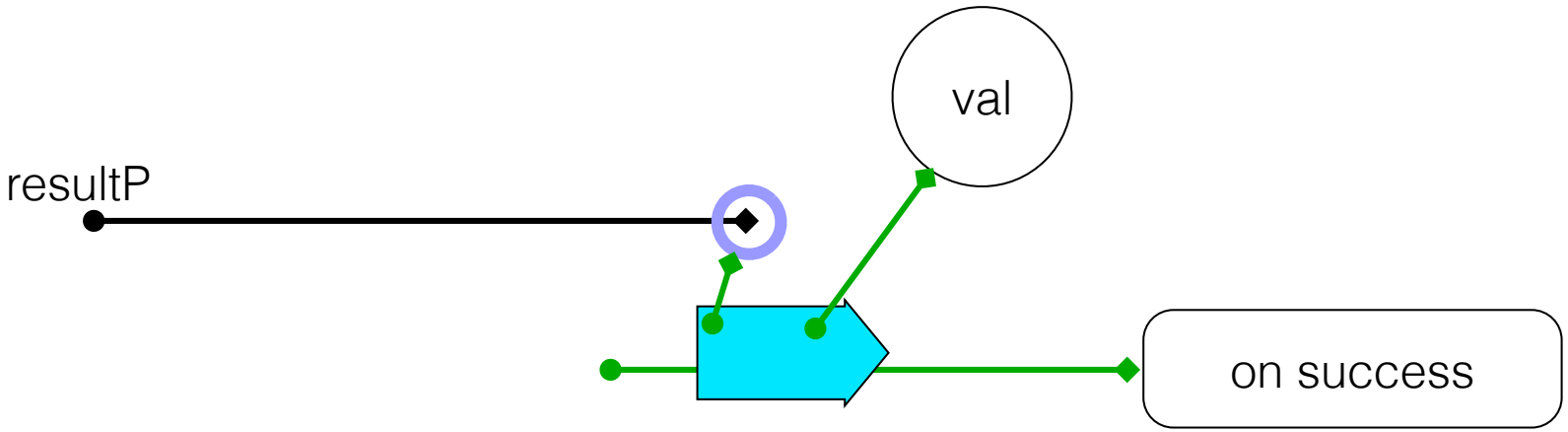
The when-catch expression

```
var resultP = Q(valP).when(function(val) {    Success case
   ...val...
}, function(reason) {
   ...reason...
});
```



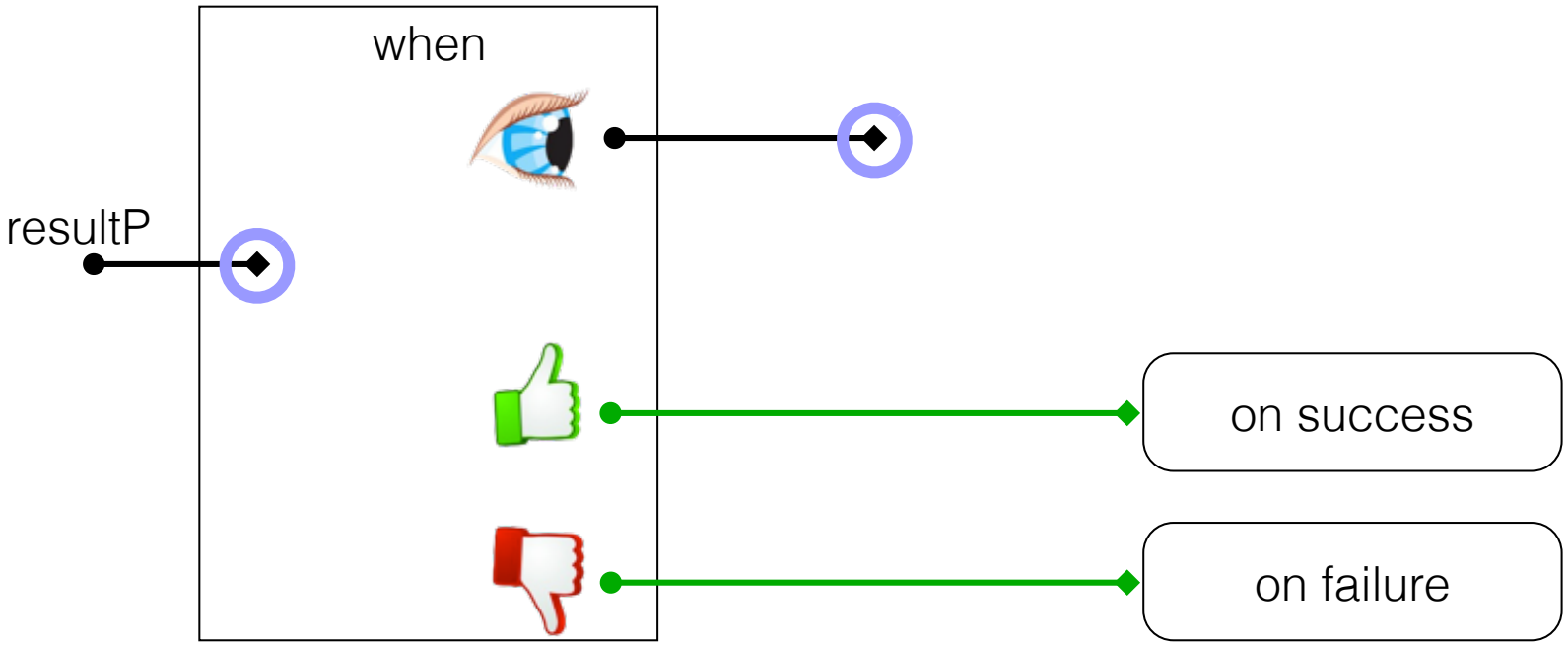
The when-catch expression

```
var resultP = Q(valP).when(function(val) {    Success case
   ...val...
}, function(reason) {
   ...reason...
});
```



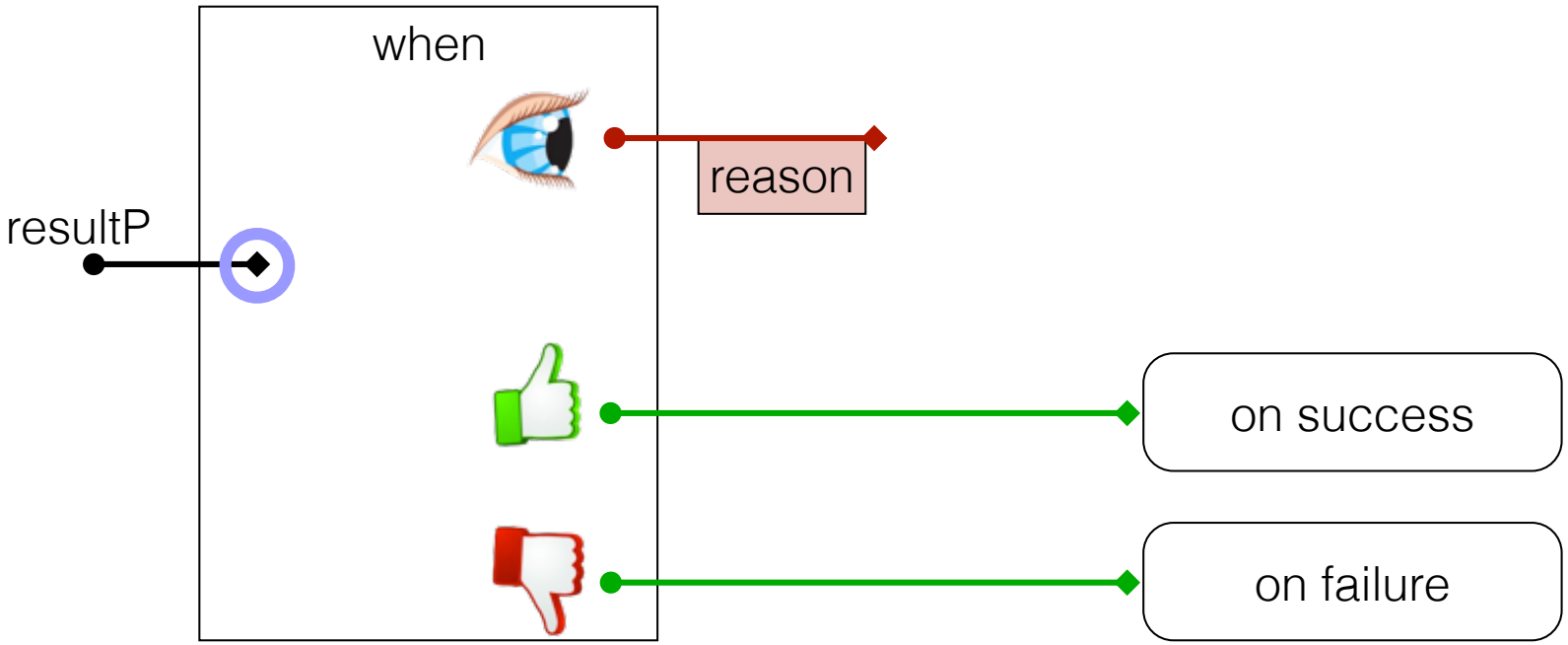
The when-catch expression

```
var resultP = Q(valP).when(function(val) {    Failure case
    ...val...
}, function(reason) {
    ...reason...
});
```



The when-catch expression

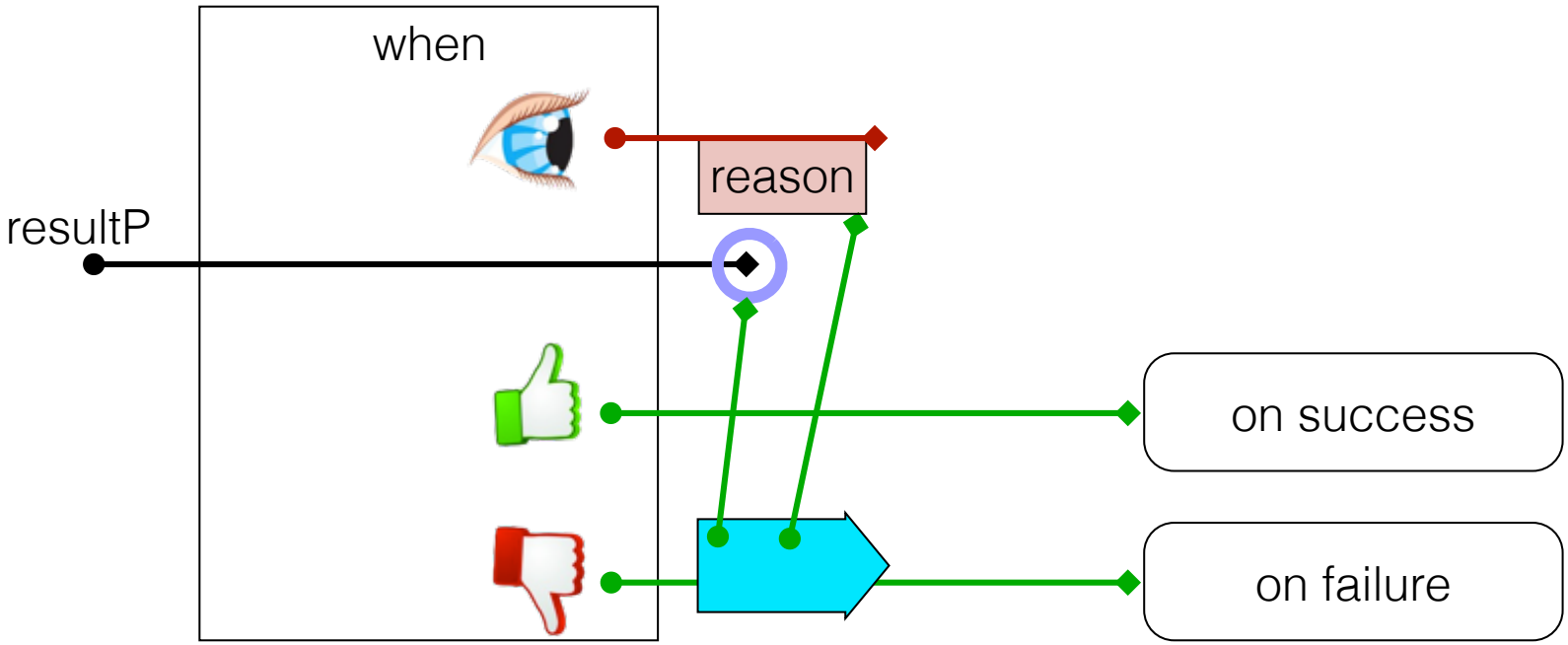
```
var resultP = Q(valP).when(function(val) {    Failure case
    ...val...
}, function(reason) {
    ...reason...
});
```



The when-catch expression

```
var resultP = Q(valP).when(function(val) {
    ...val...
}, function(reason) {
    ...reason...
});
```

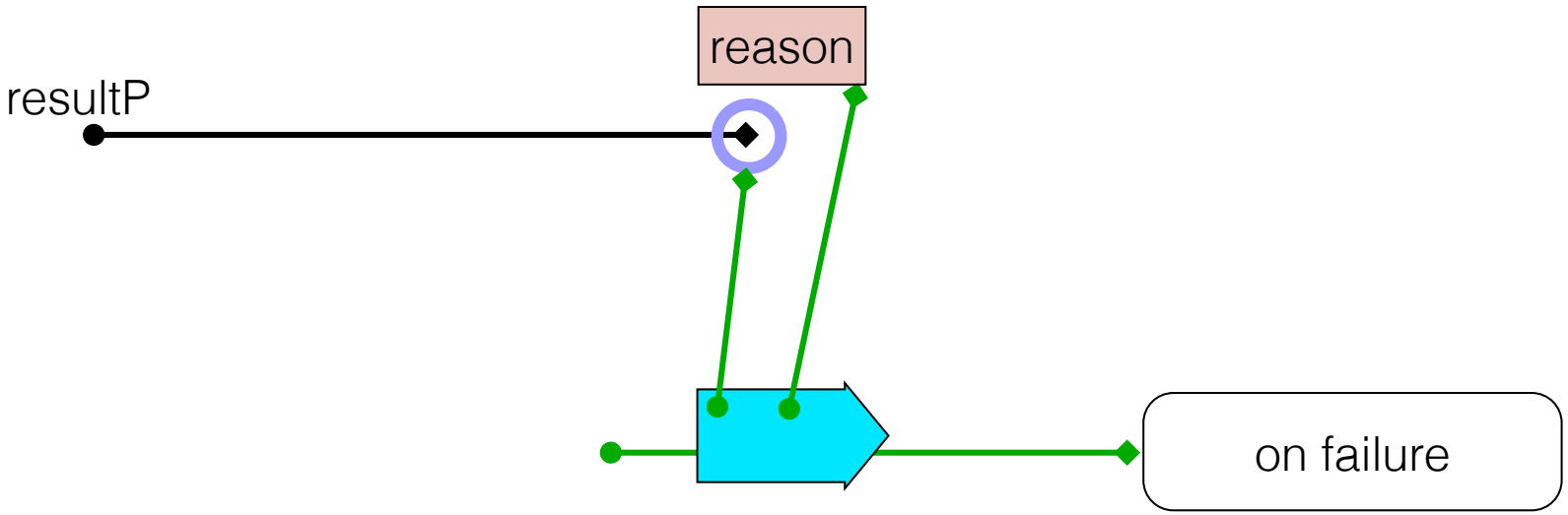
Failure case



The when-catch expression

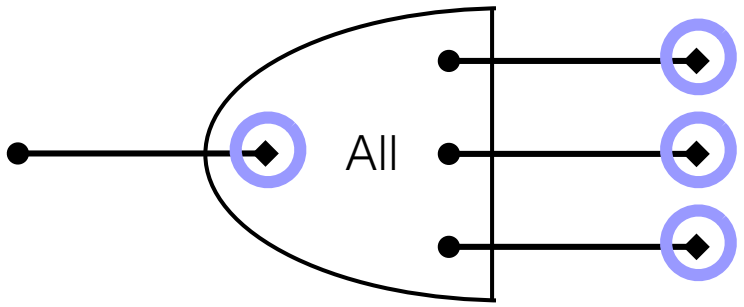
```
var resultP = Q(valP).when(function(val) {
   ...val...
}, function(reason) {
   ...reason...
});
```

Failure case



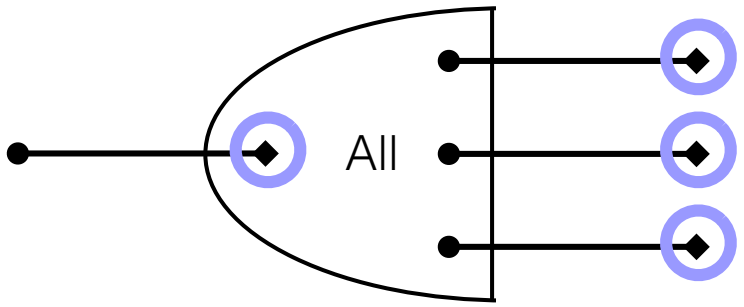
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



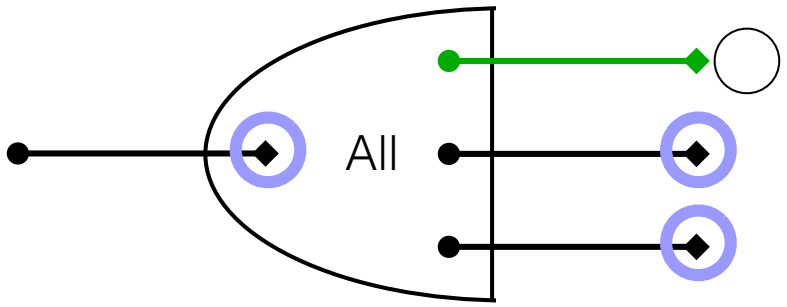
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



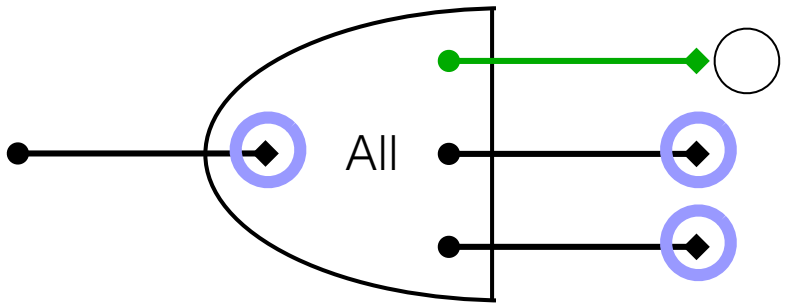
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



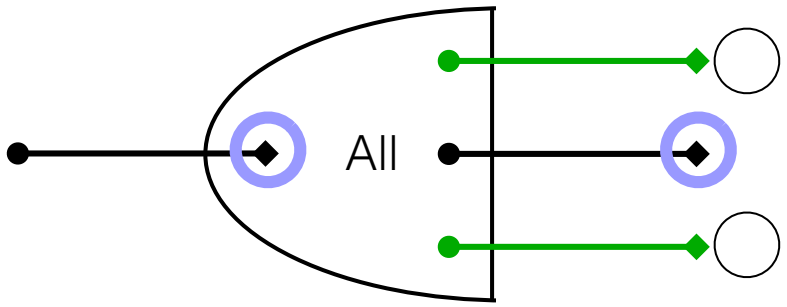
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



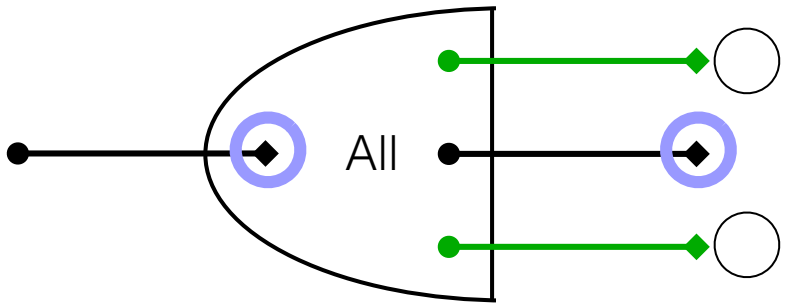
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



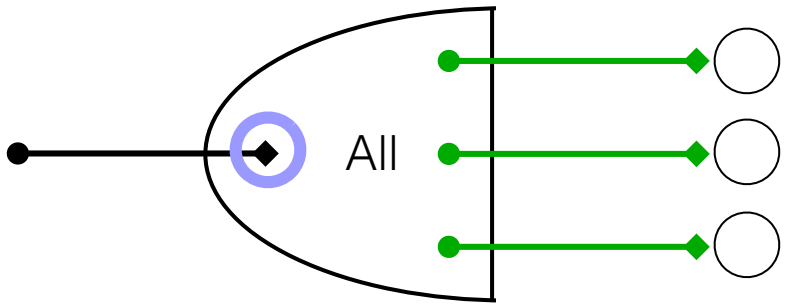
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



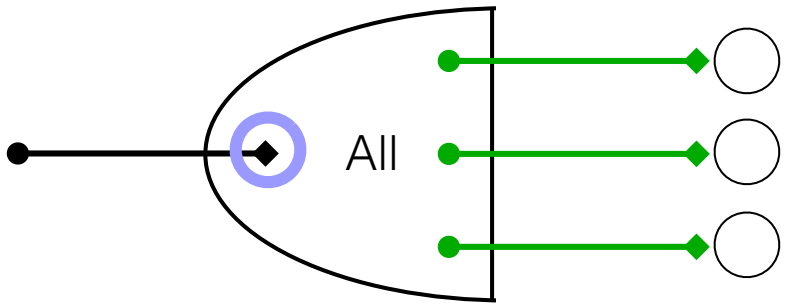
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



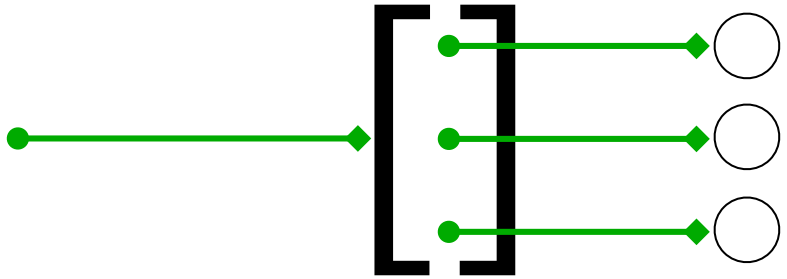
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



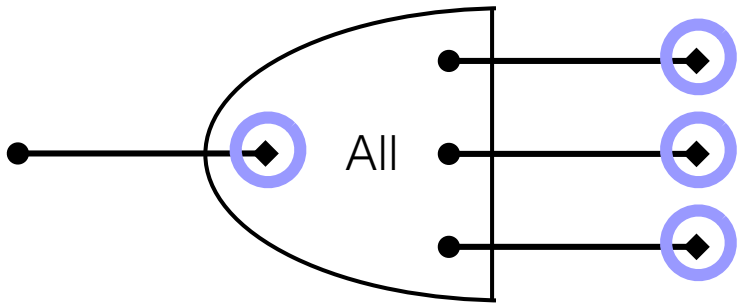
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



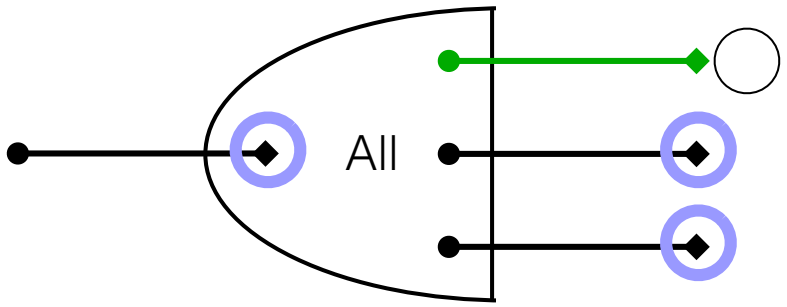
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



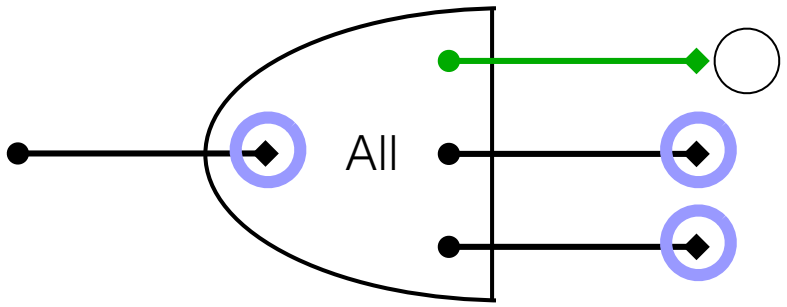
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



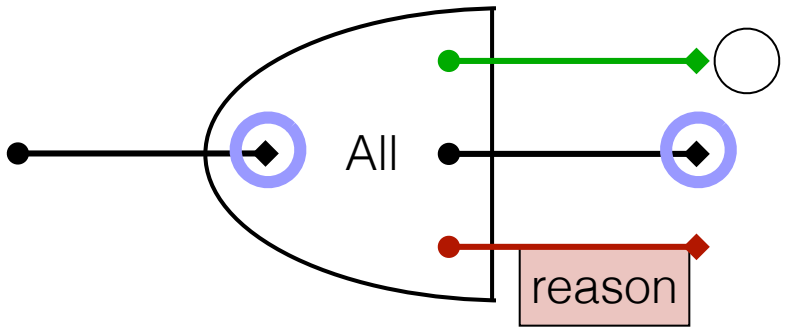
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



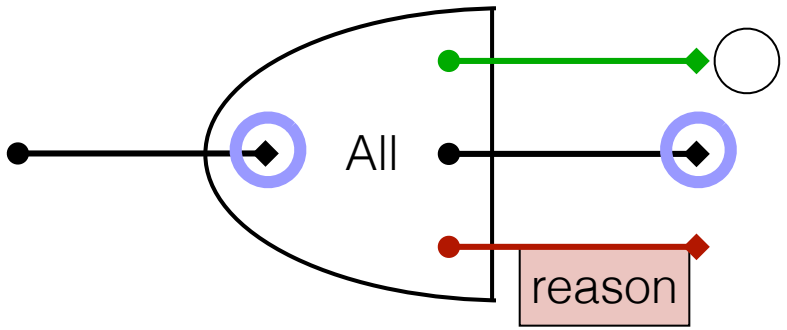
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



Q.all
All good? Any bad?

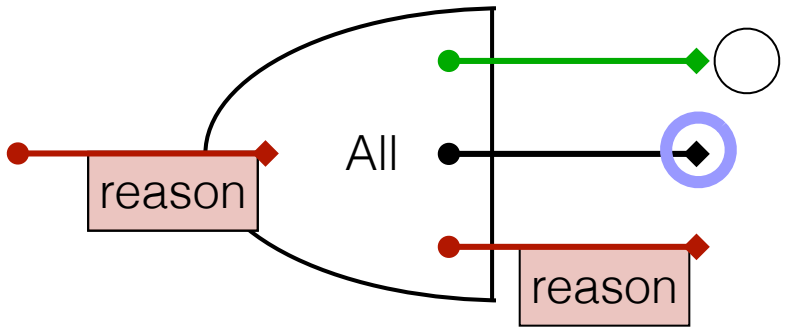
```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



Q.all

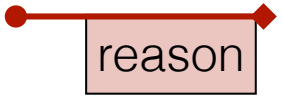
All good? Any bad?

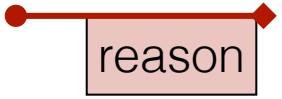
```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



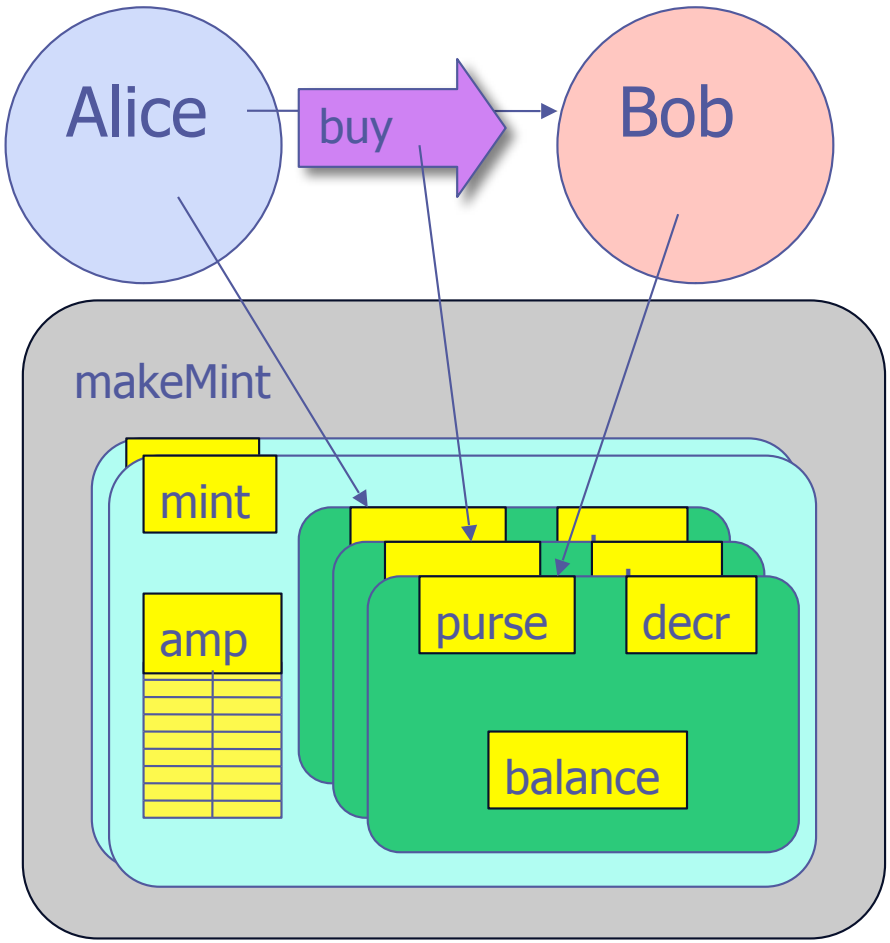
Q.all
All good? Any bad?

```
Q.all = function(answerPs) {
  var countDown = answerPs.length;
  if (countDown === 0) { return answerPs; }
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(_) {
      if (--countDown === 0) { result.resolve(answerPs); }
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



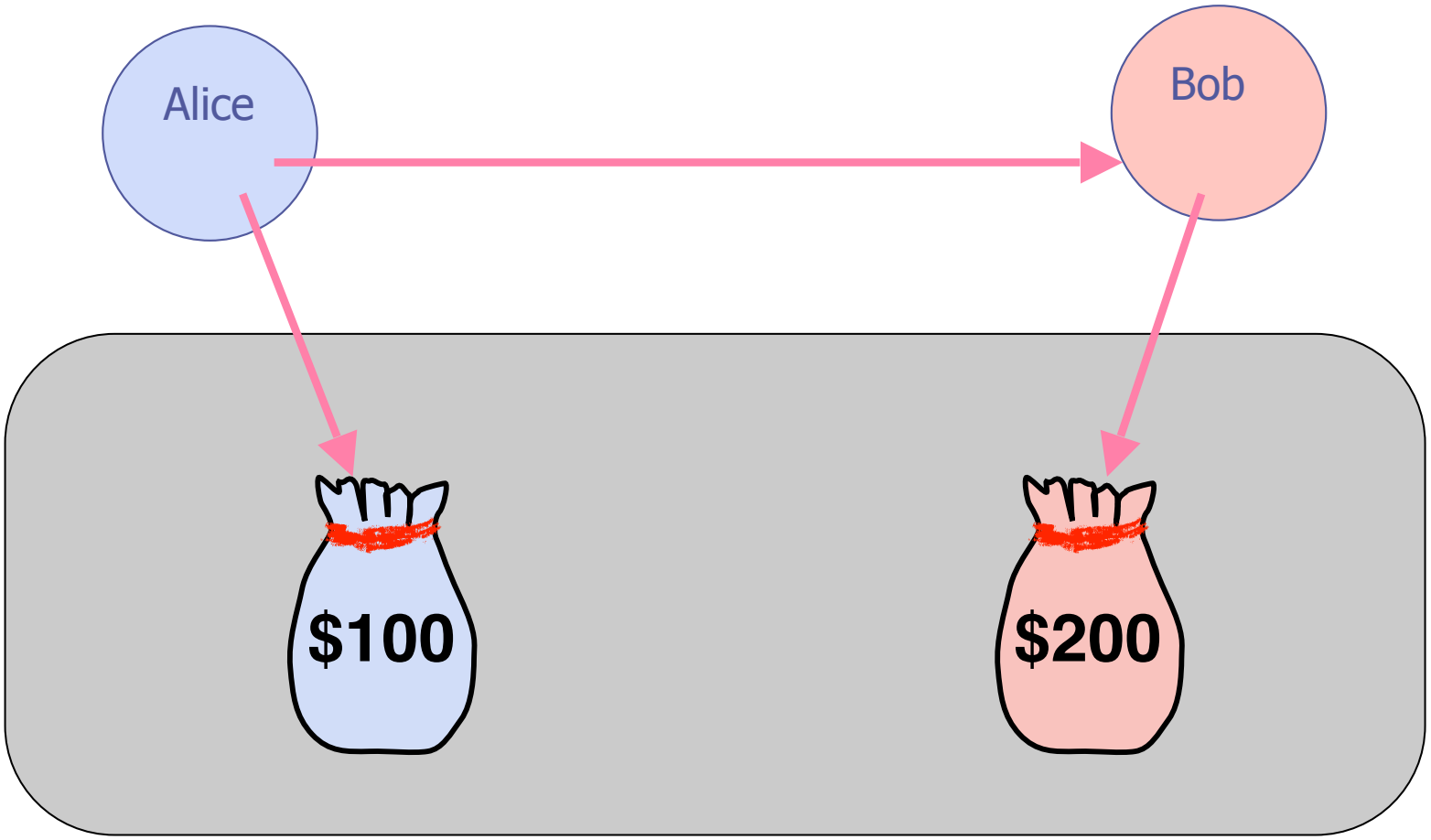


bank (17 lines)
Money as "factorial" of language security



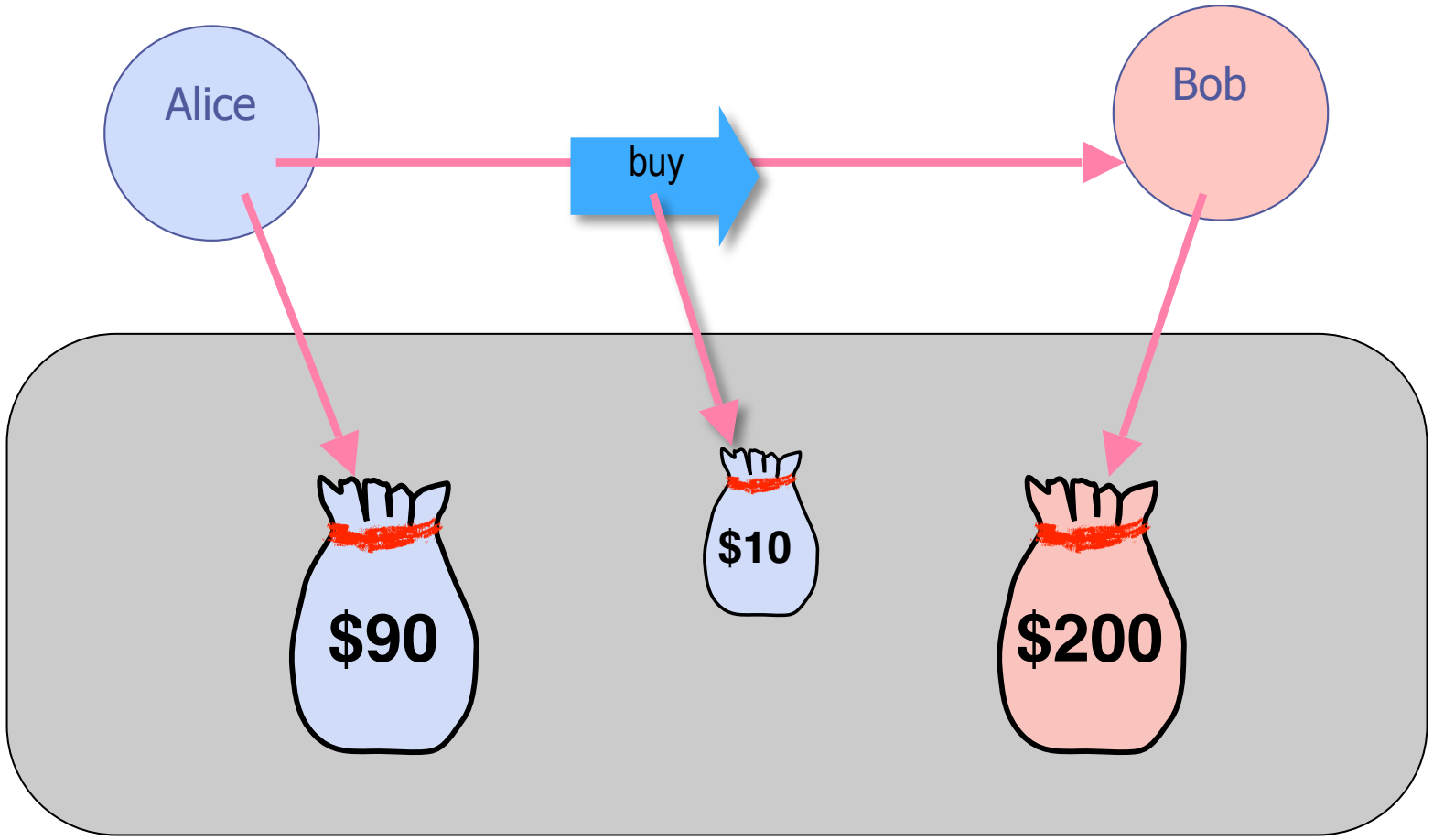

```
function bank() {
  var amp = WeakMap();
  return function mint(balance) {
    var purse = def({
      getBalance: function() { return balance; },
      makePurse: function() { return mint(0); },
      deposit: function(amount, src) {
        Nat(balance + amount);
        amp.get(src)(Nat(amount));
        balance += amount;
    } });
    function decr(amount) {
      balance = Nat(balance – amount);
    }
    amp.set(purse, decr);
    return purse;
}; }
```

Distributed Secure Currency



Distributed Secure Currency

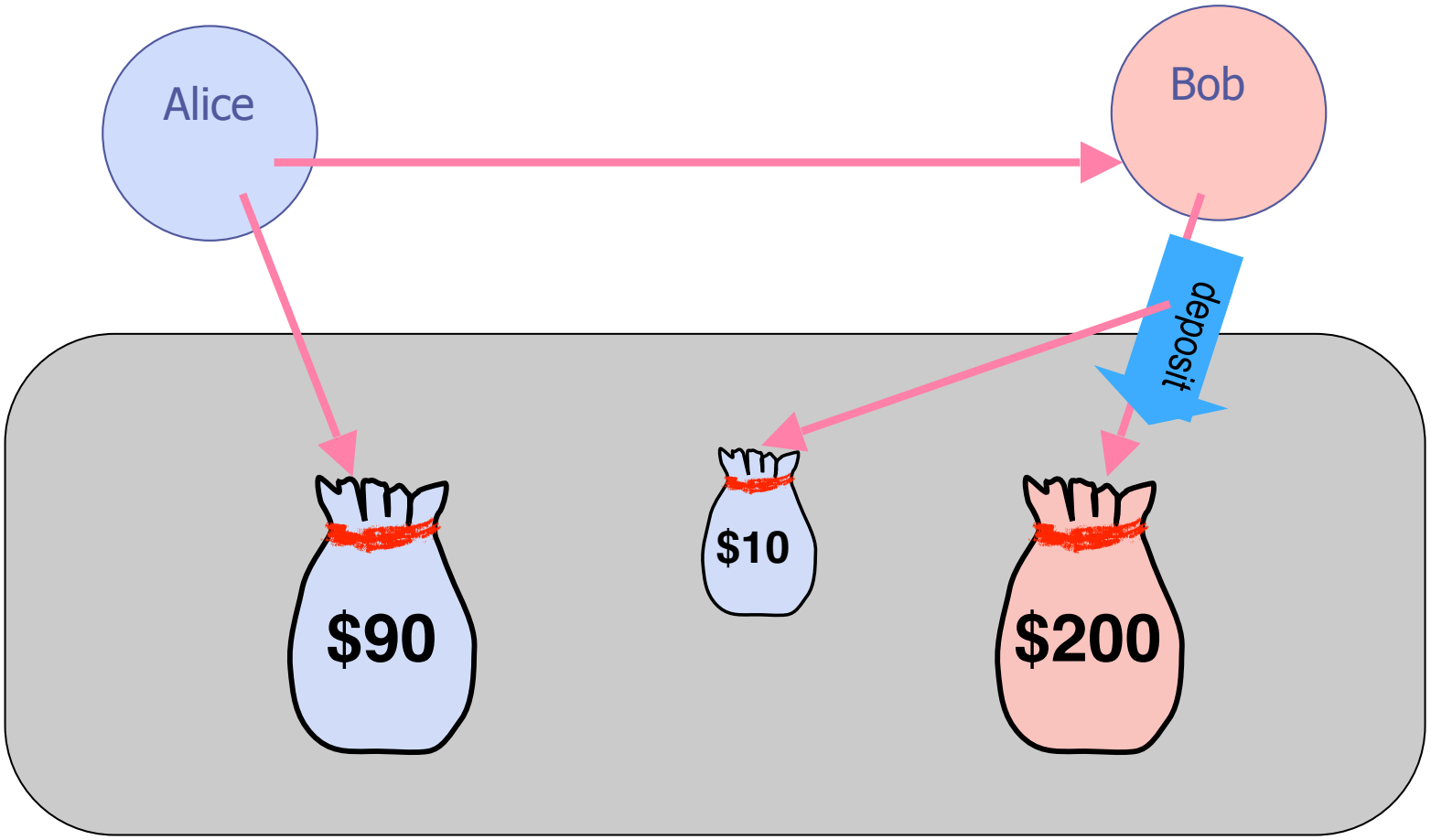
```
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
var goodP = bobP ! buy(desc, paymentP);
```



Distributed Secure Currency

```
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
var goodP = bobP ! buy(desc, paymentP);
```

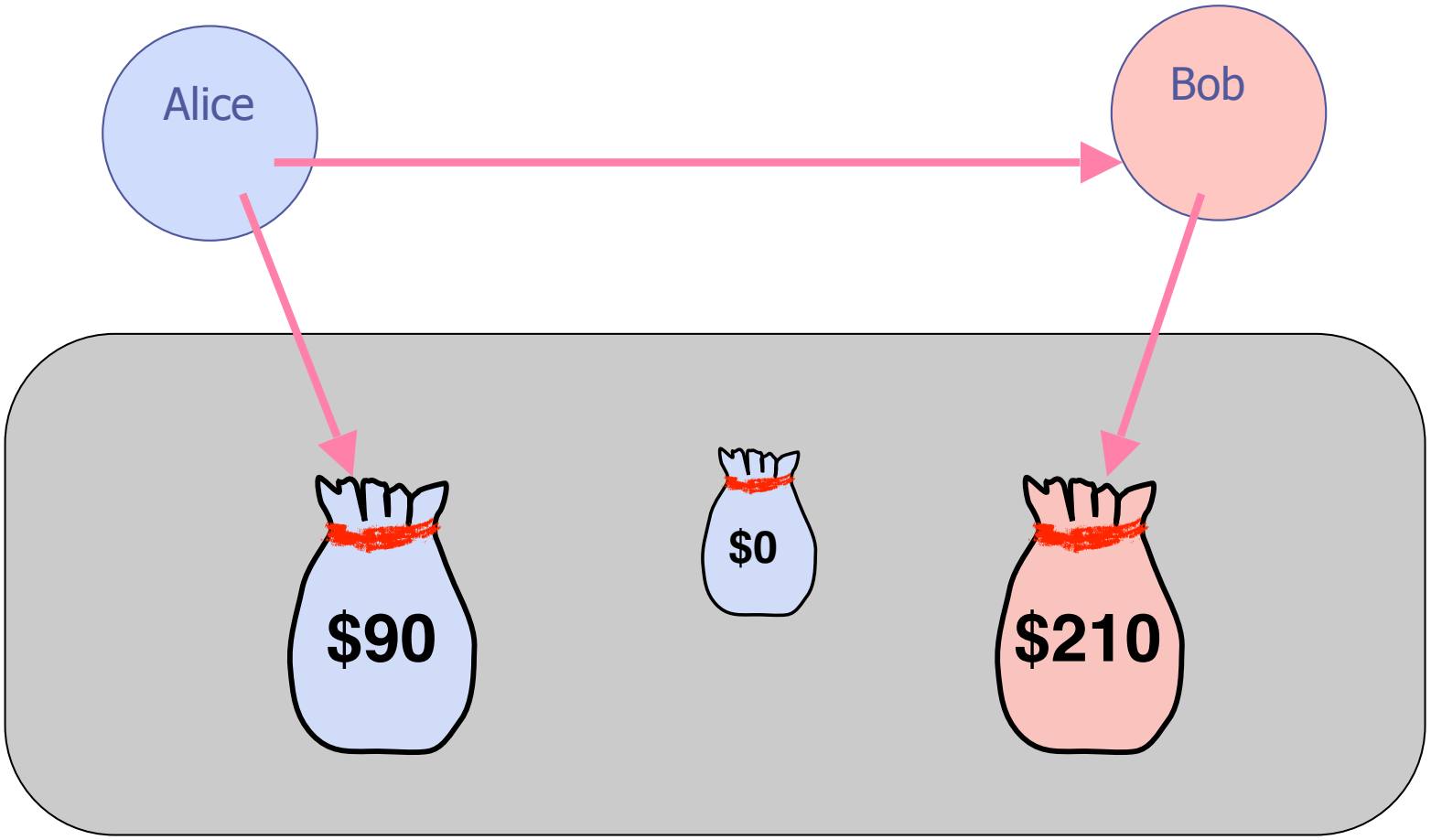
```
return Q(paymentP).when(function(p) {
  return Q(myPurse ! deposit(10, p)).when(function(_) {
    return good; }, ...
```



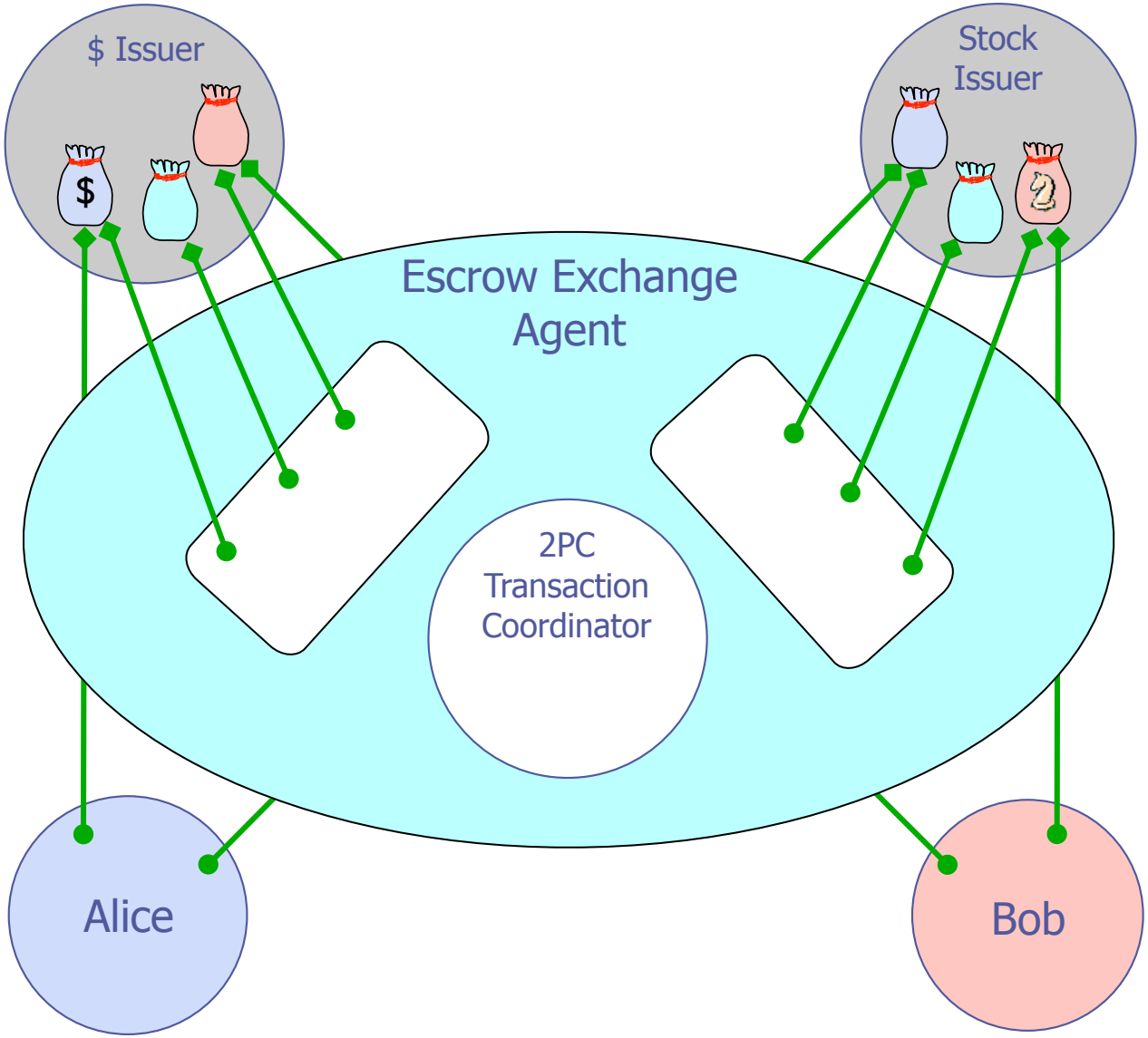
Distributed Secure Currency

```
var paymentP = myPurse ! makePurse();
paymentP ! deposit(10, myPurse);
var goodP = bobP ! buy(desc, paymentP);
```

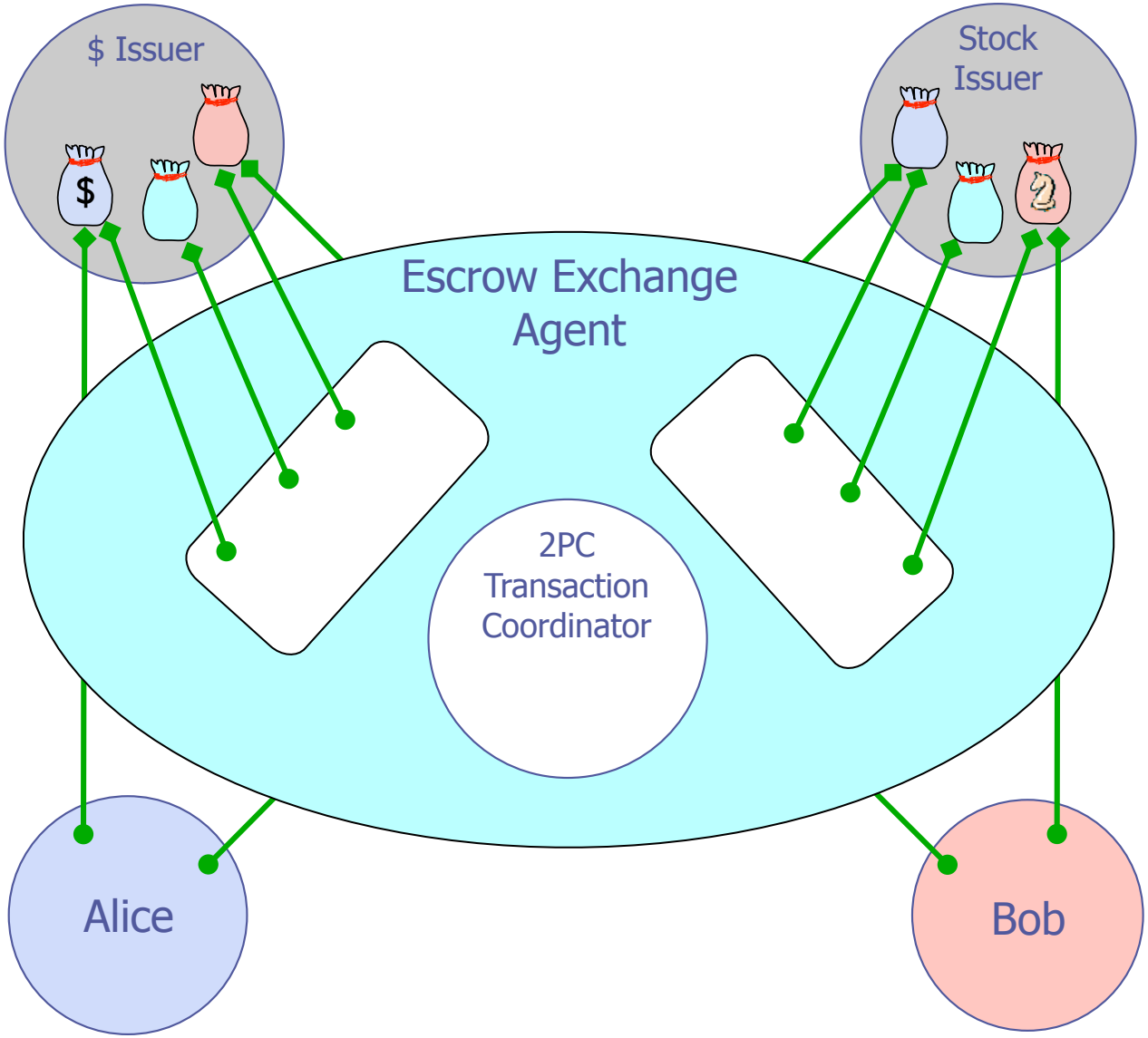
```
return Q(paymentP).when(function(p) {
  return Q(myPurse ! deposit(10, p)).when(function(_) {
    return good; }, ...
```



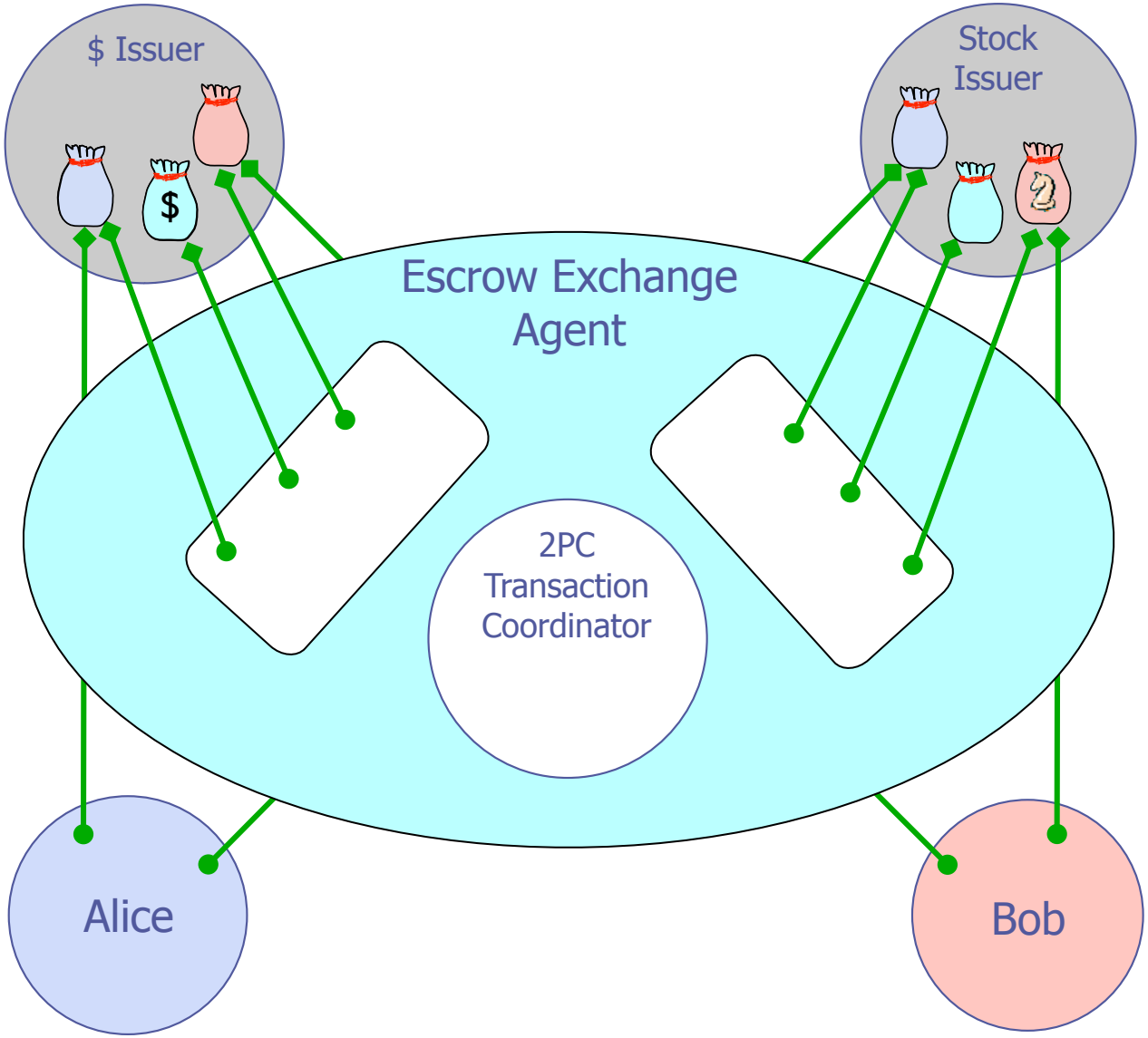
The Five Players



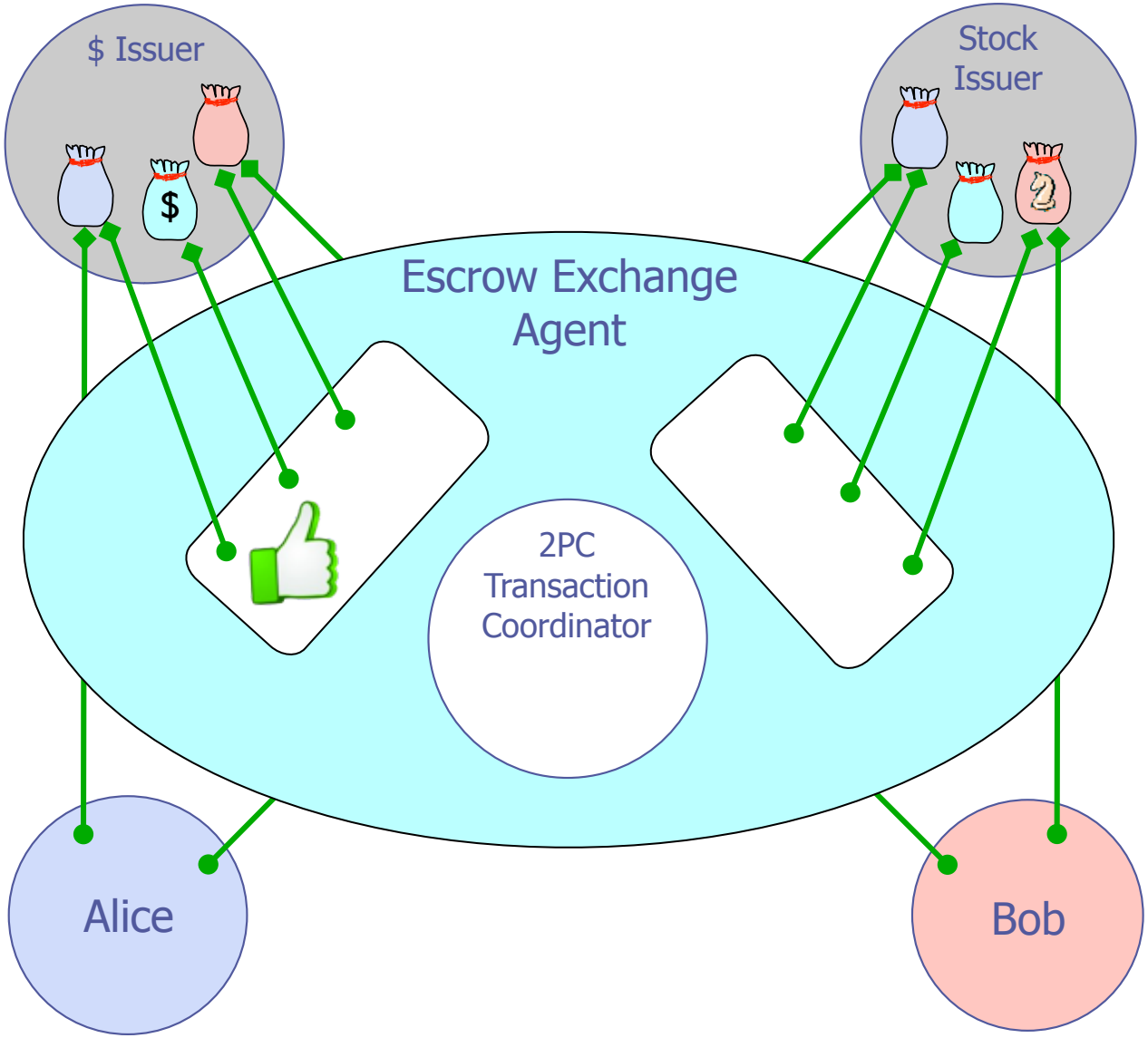
Successful Exchange -- phase 1



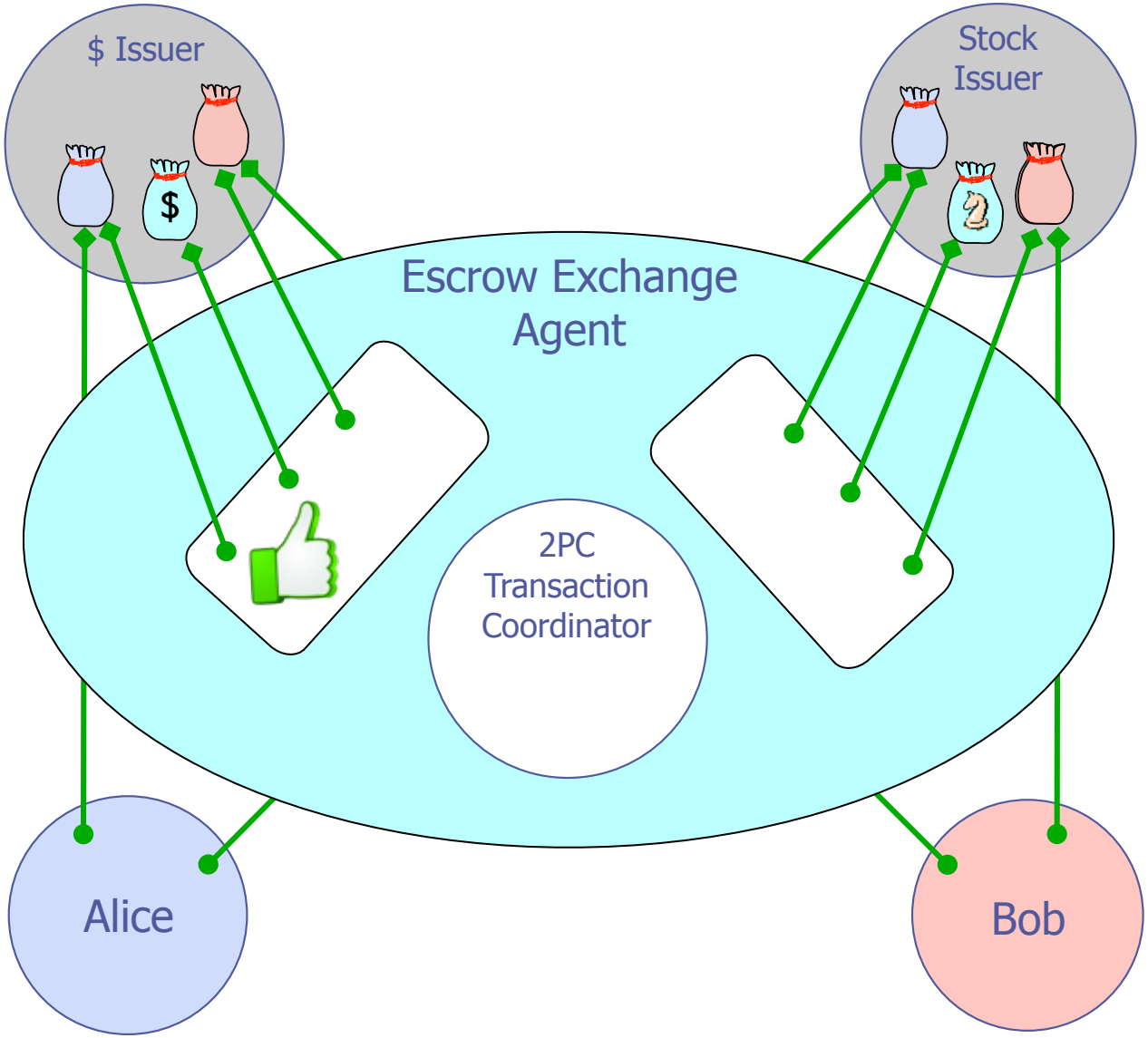
Successful Exchange -- phase 1



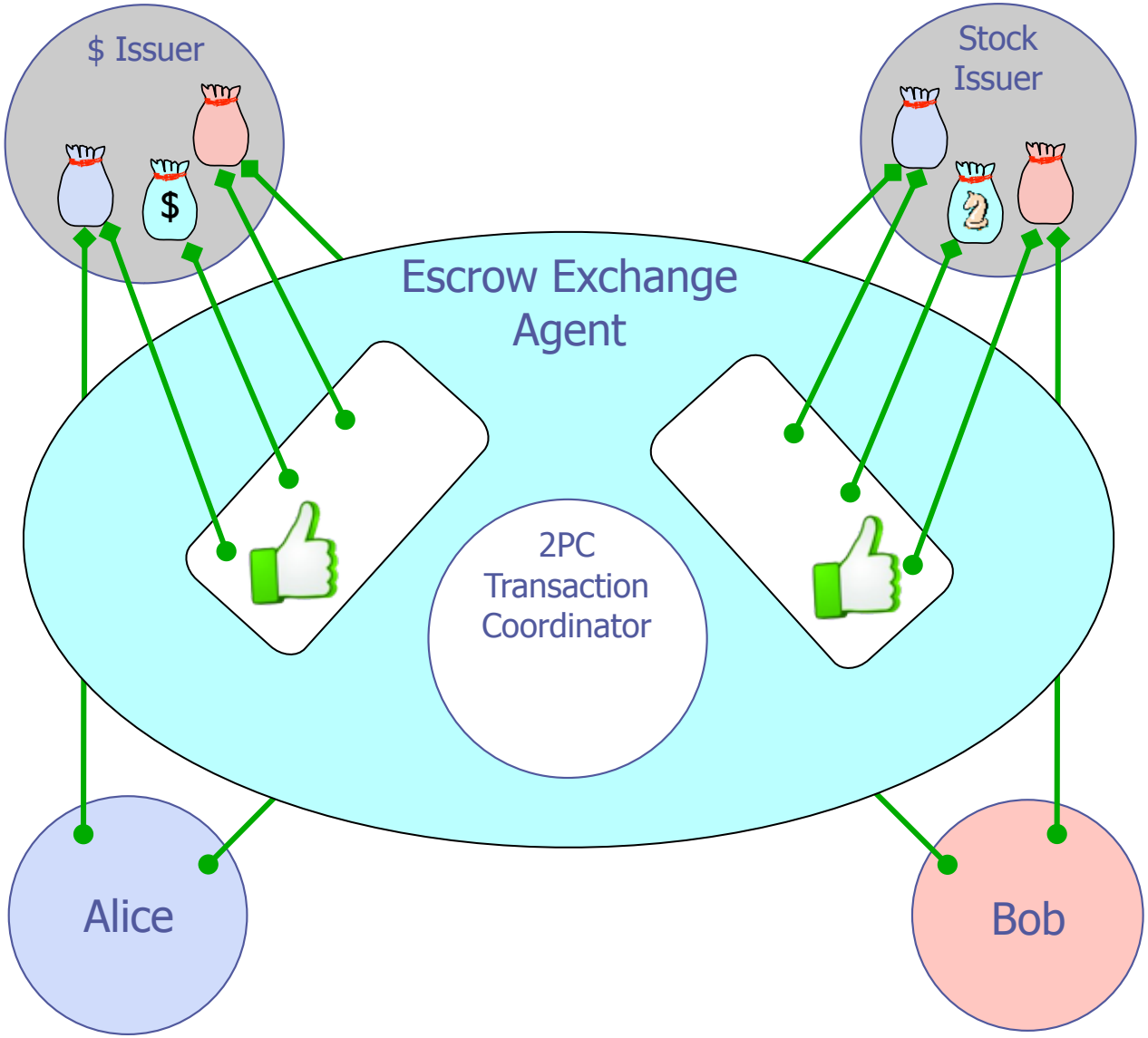
Successful Exchange -- phase 1



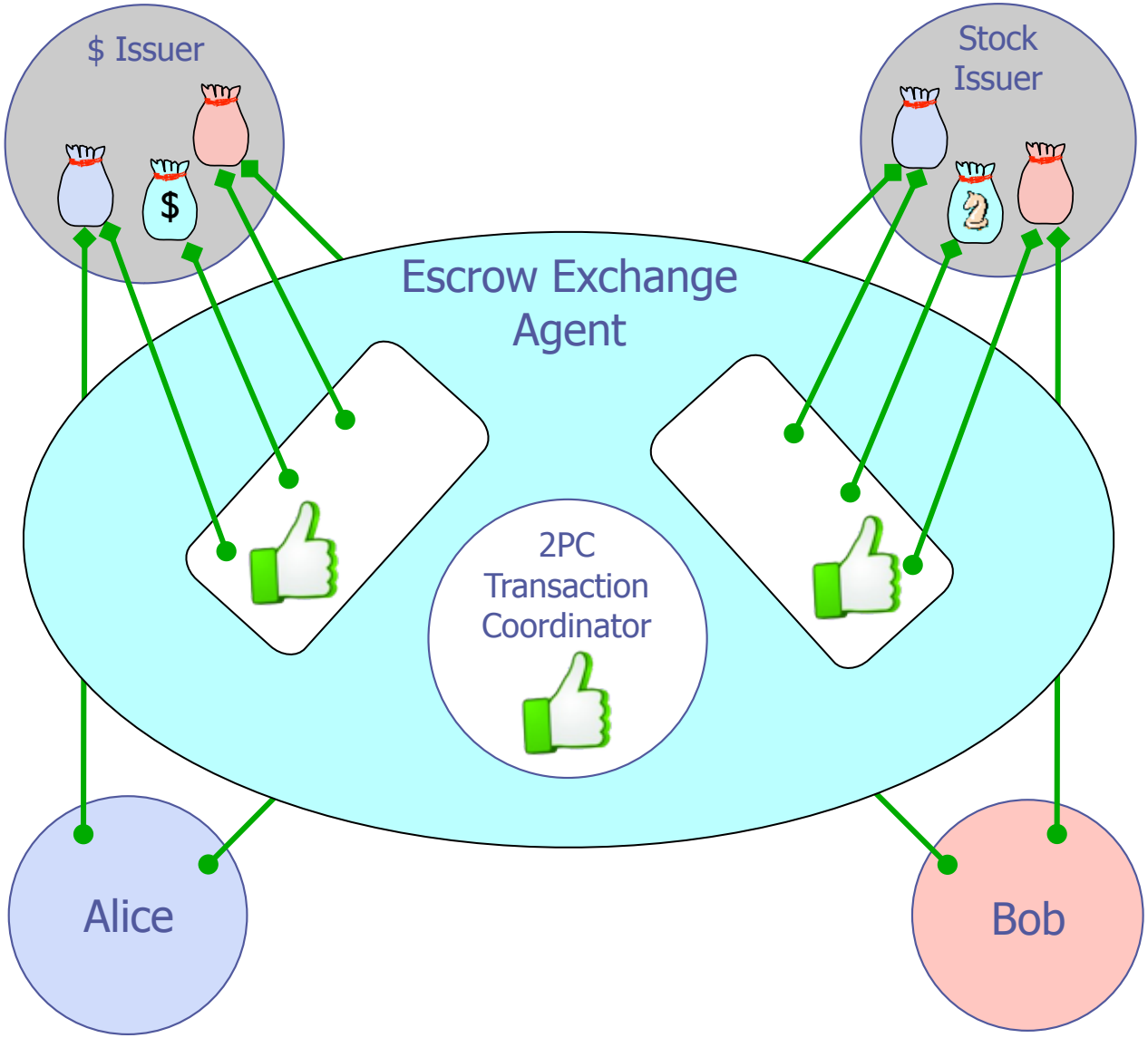
Successful Exchange -- phase 1



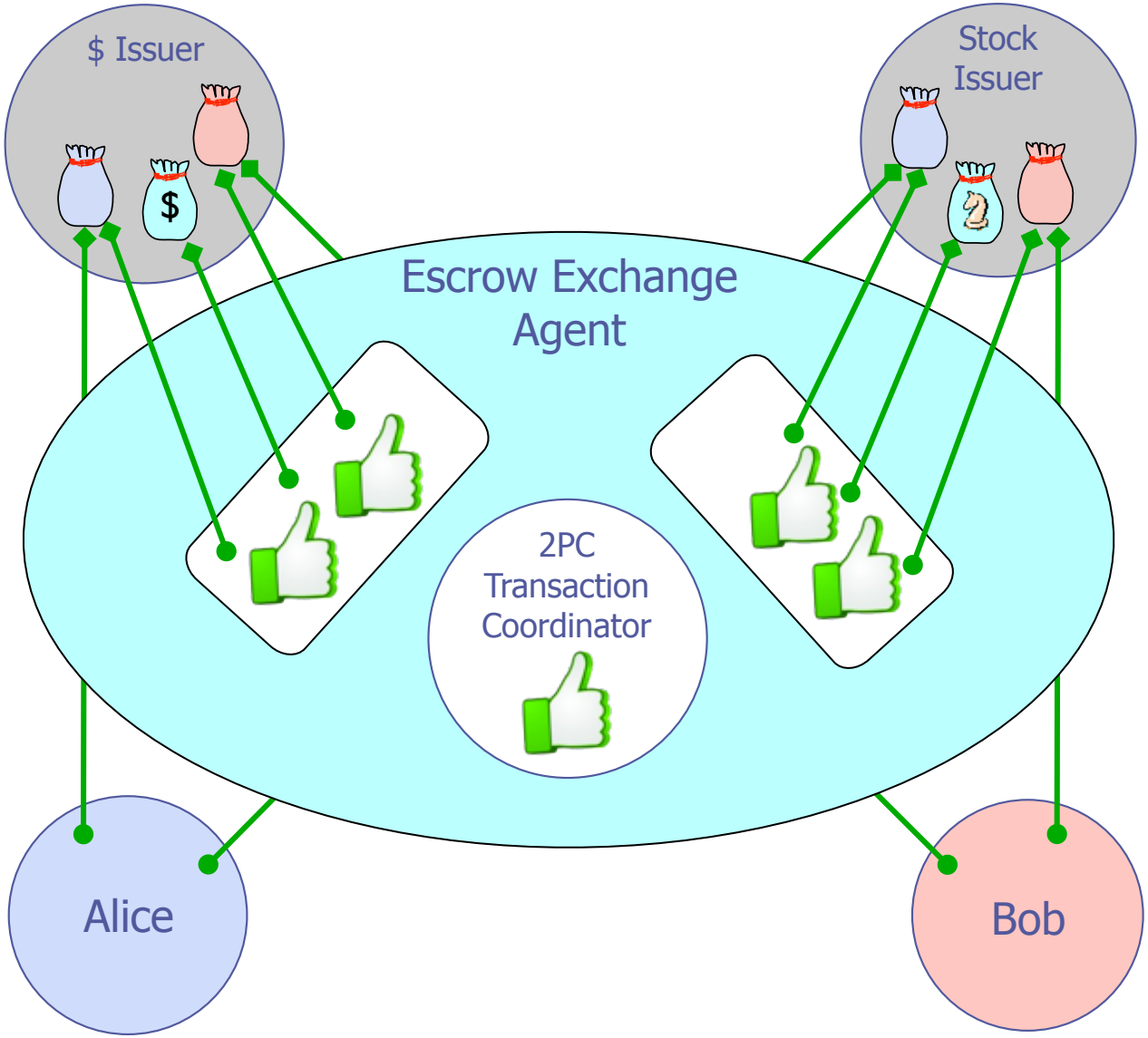
Successful Exchange -- phase 1



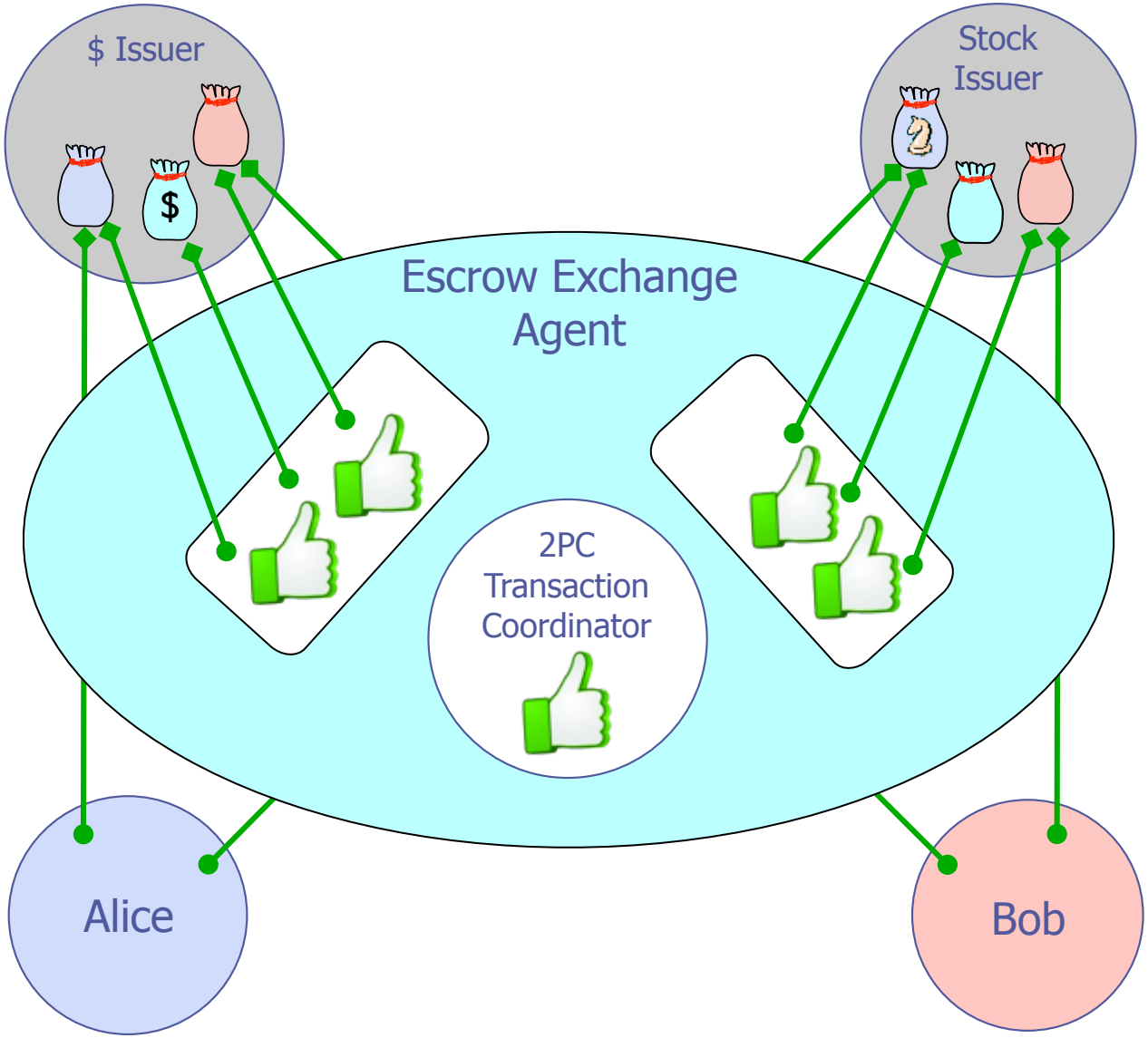
Successful Exchange -- commit



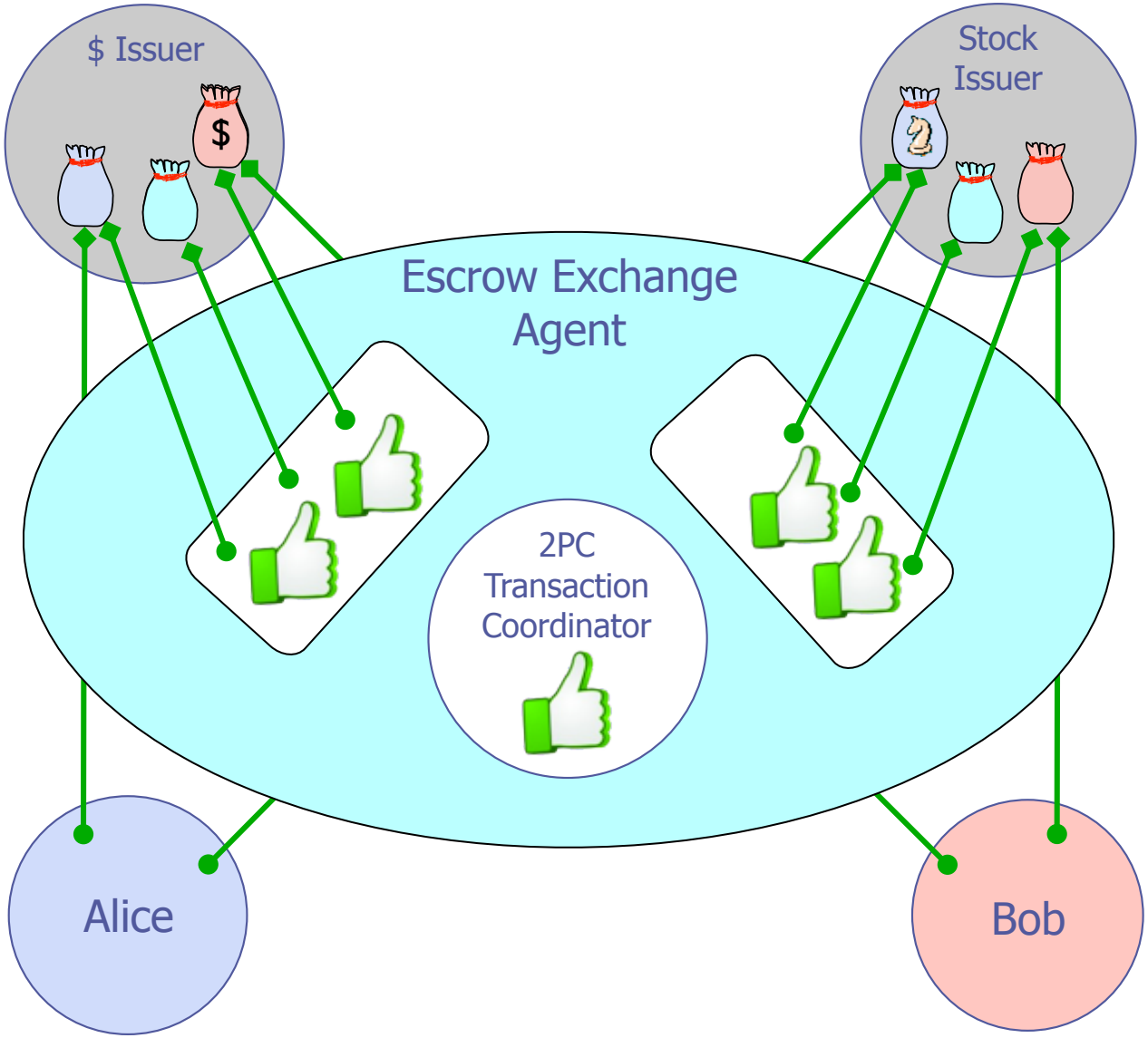
Successful Exchange -- phase 2



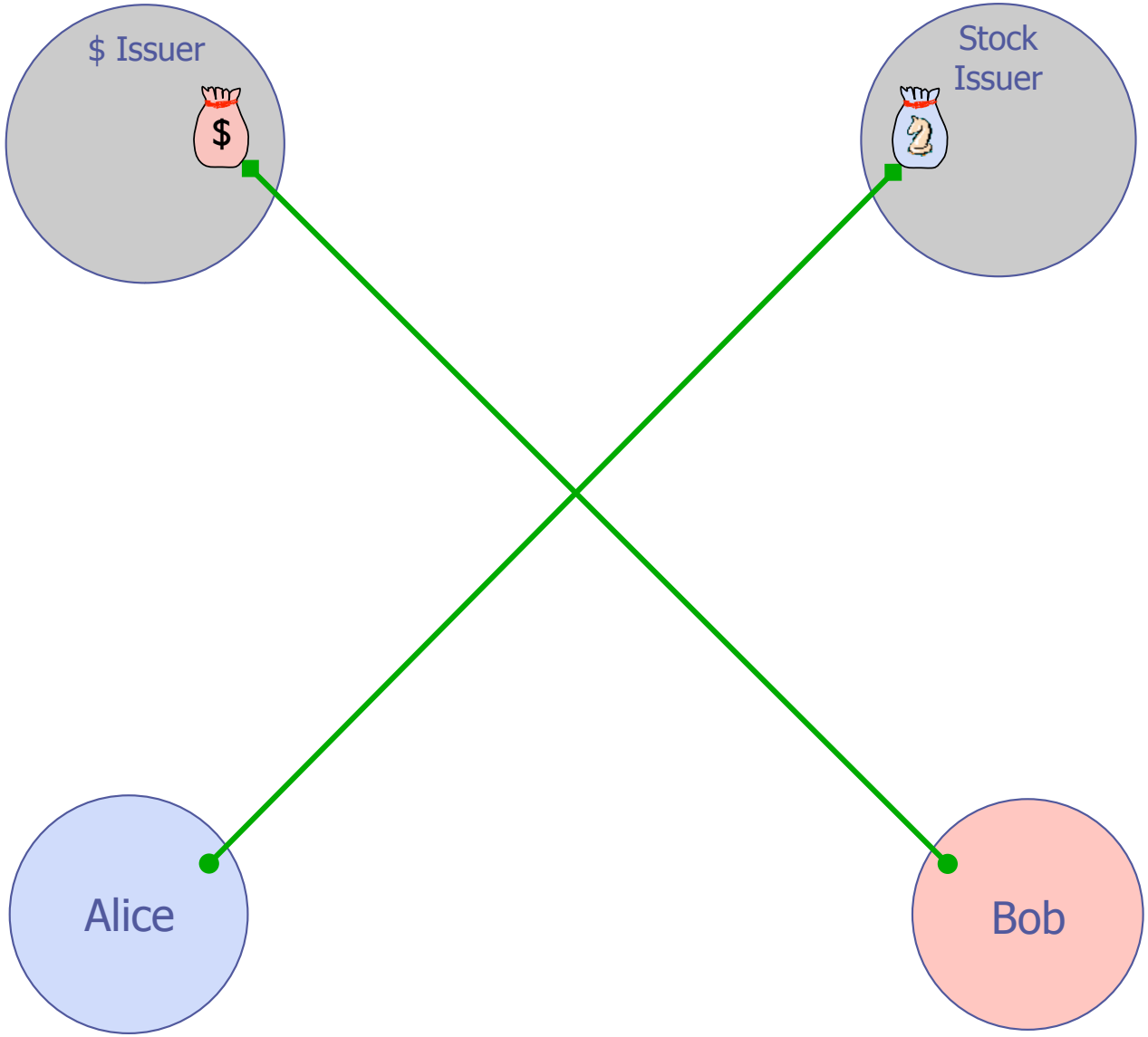
Successful Exchange -- phase 2



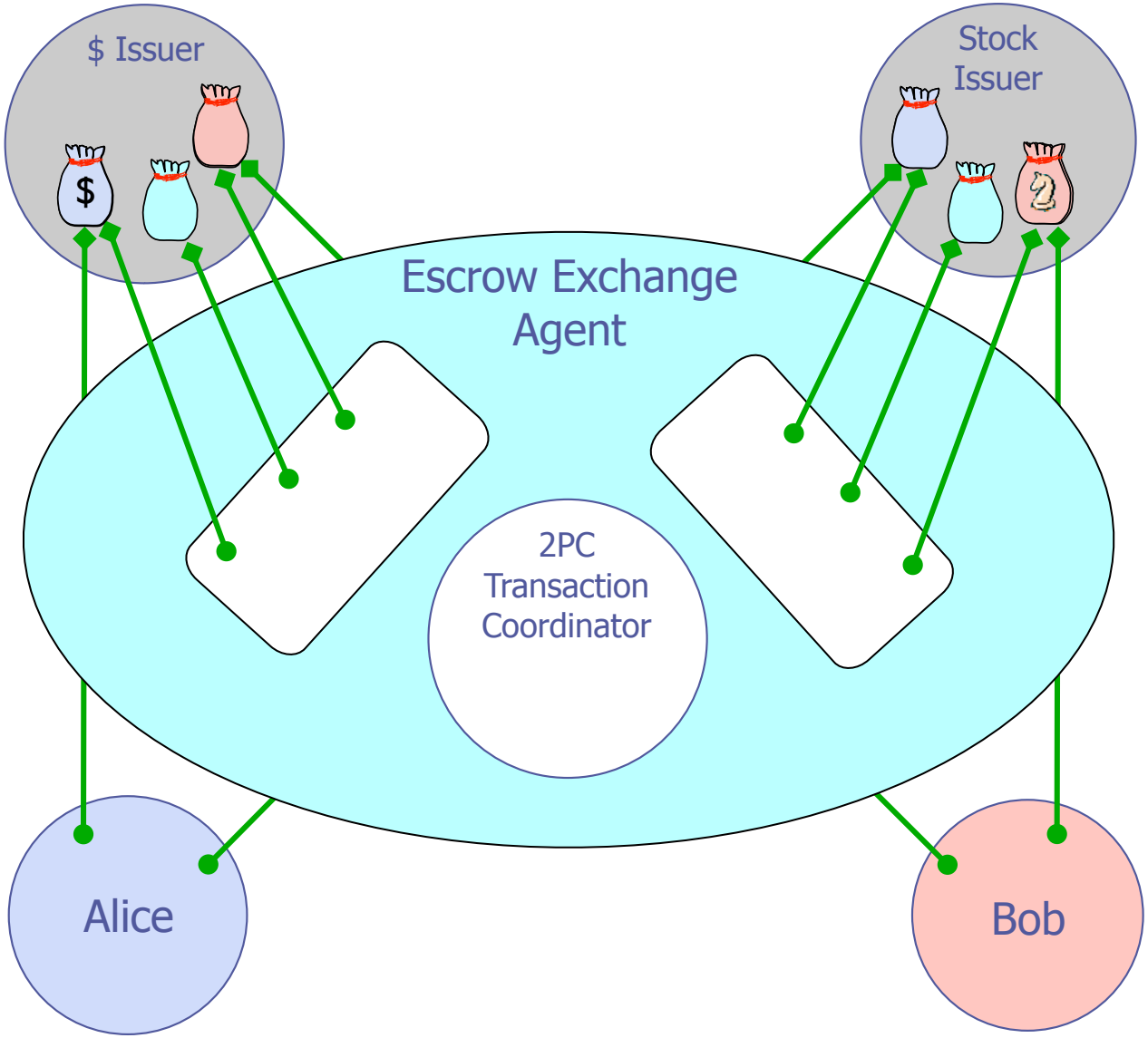
Successful Exchange -- phase 2



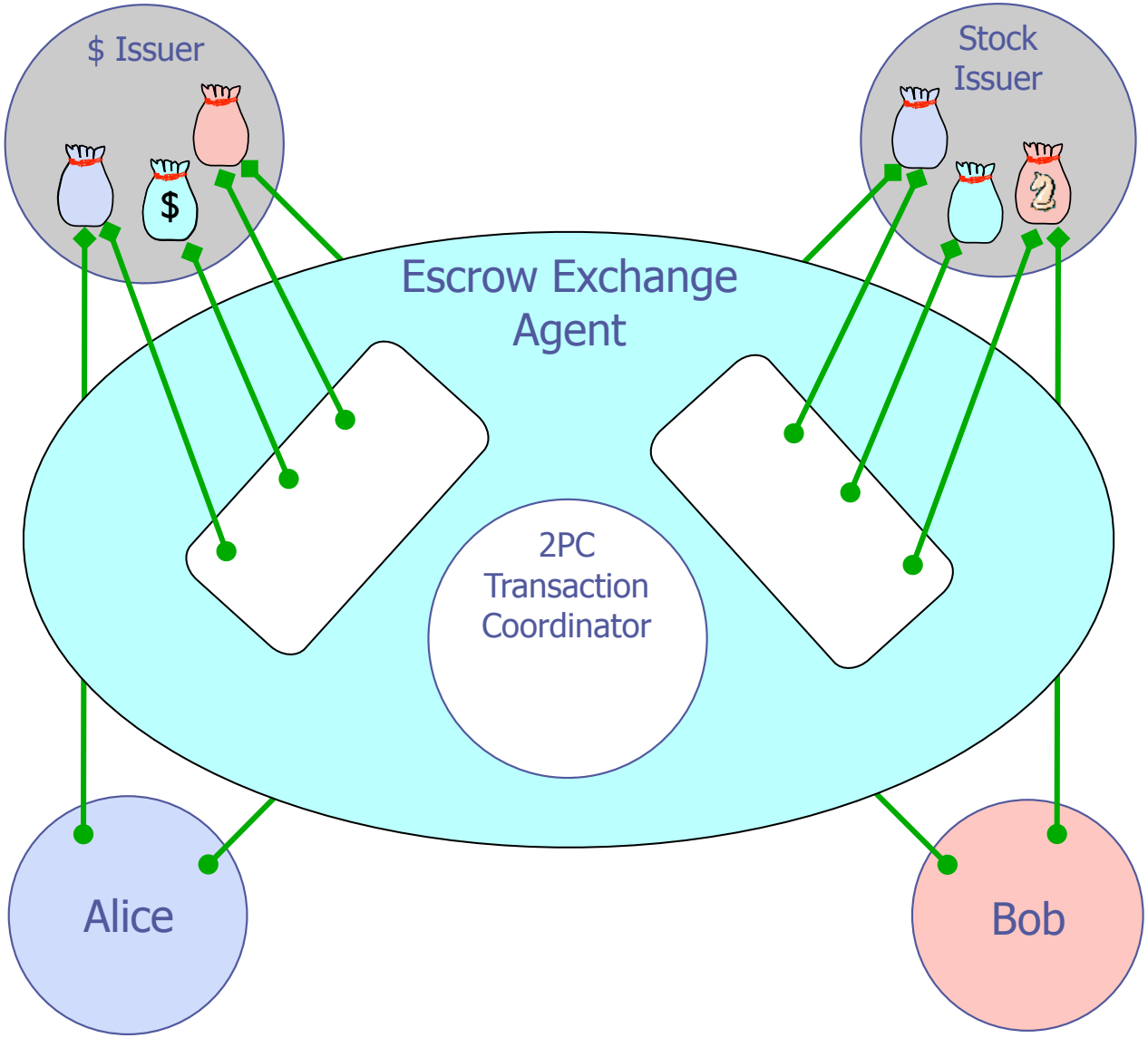
Successful Exchange -- phase 2



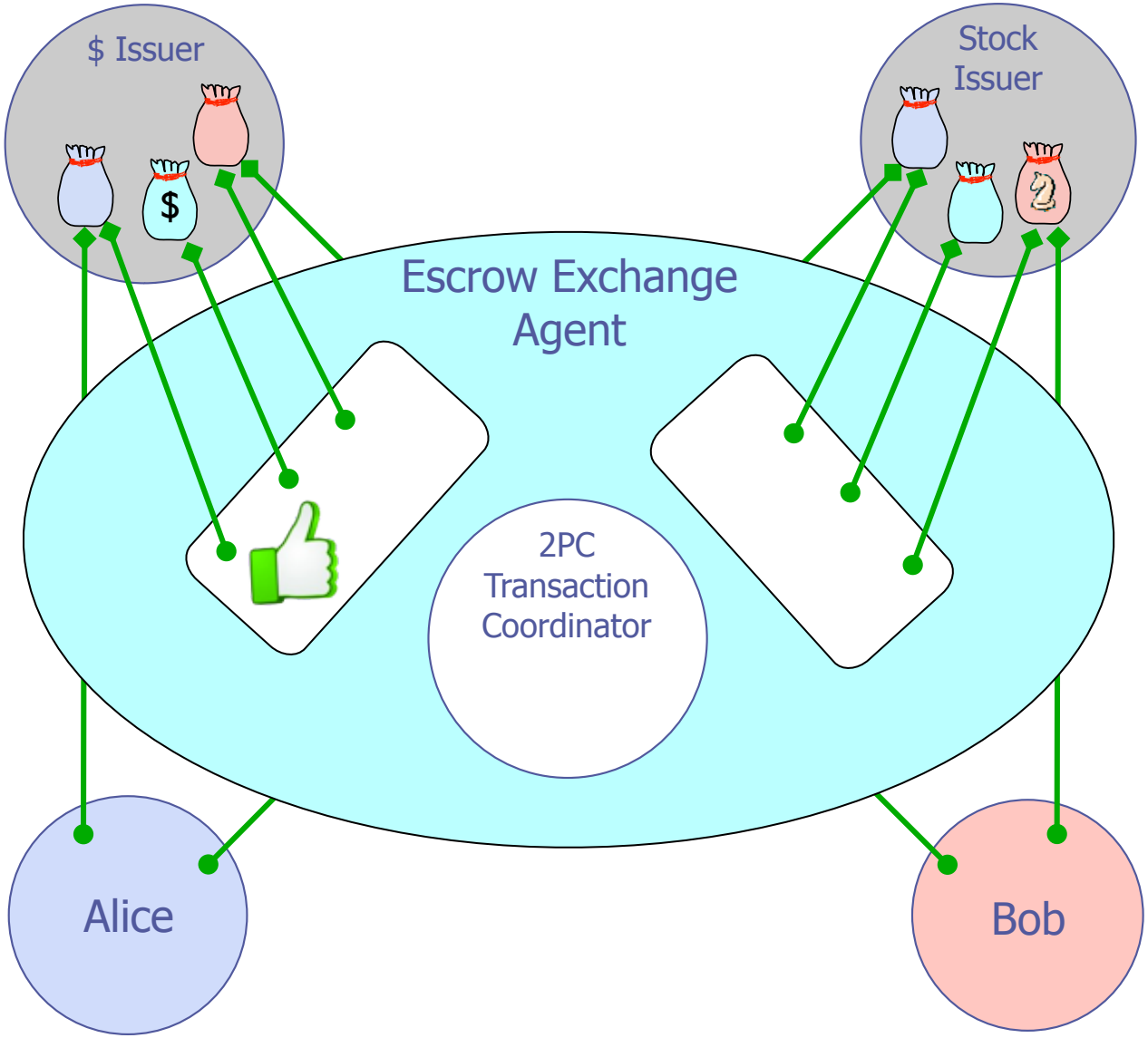
Failed Exchange -- phase 1



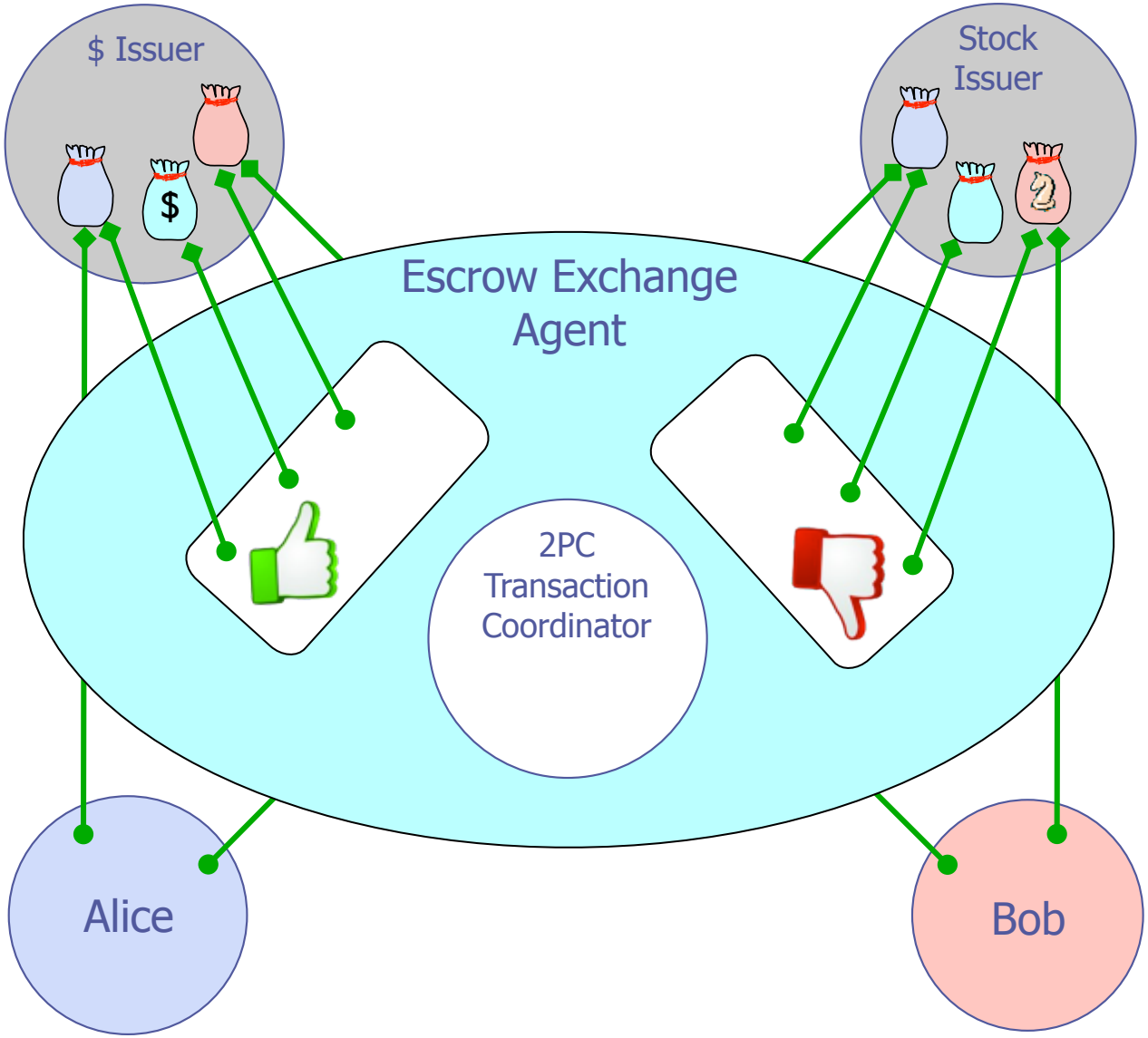
Failed Exchange -- phase 1



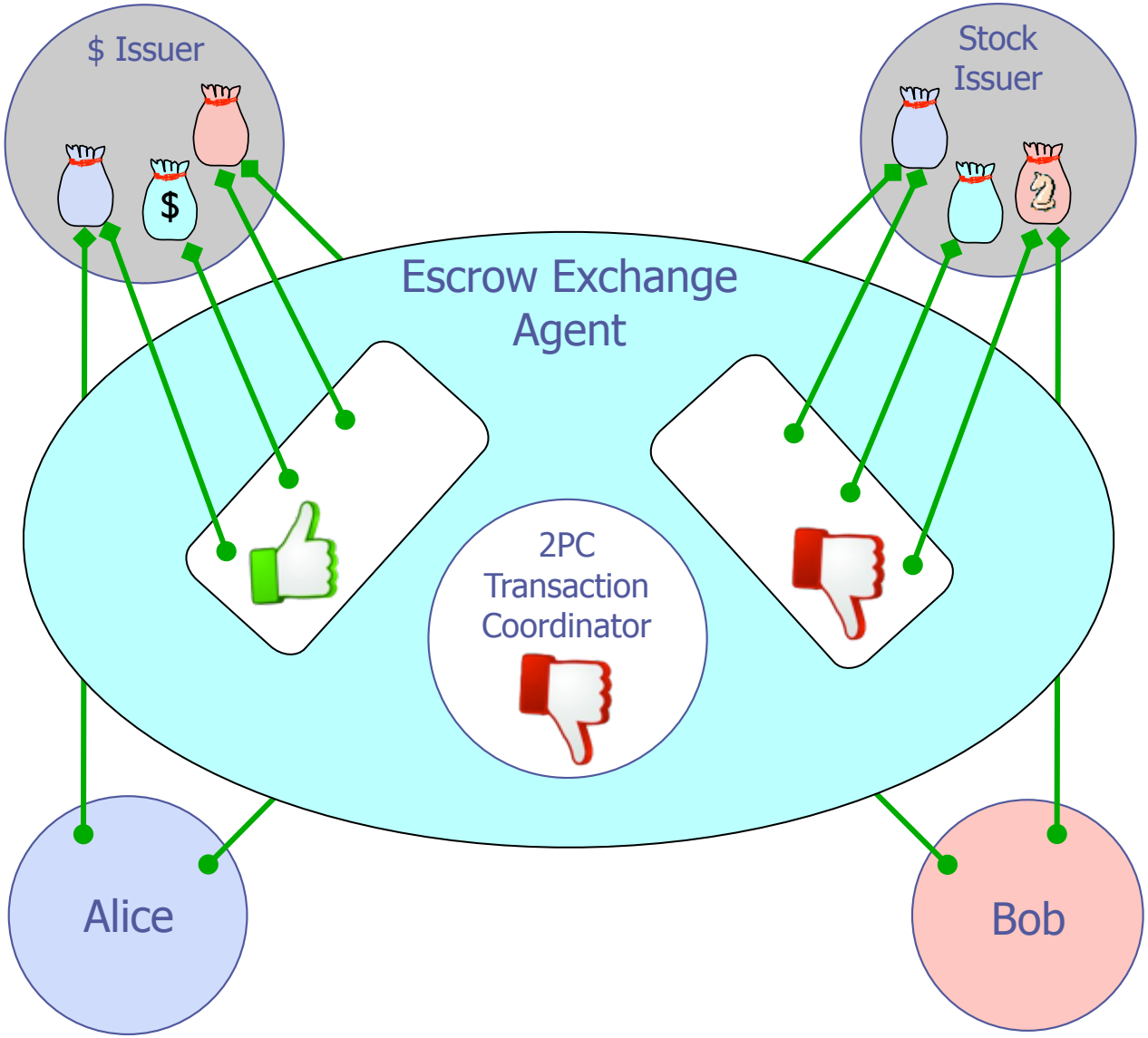
Failed Exchange -- phase 1



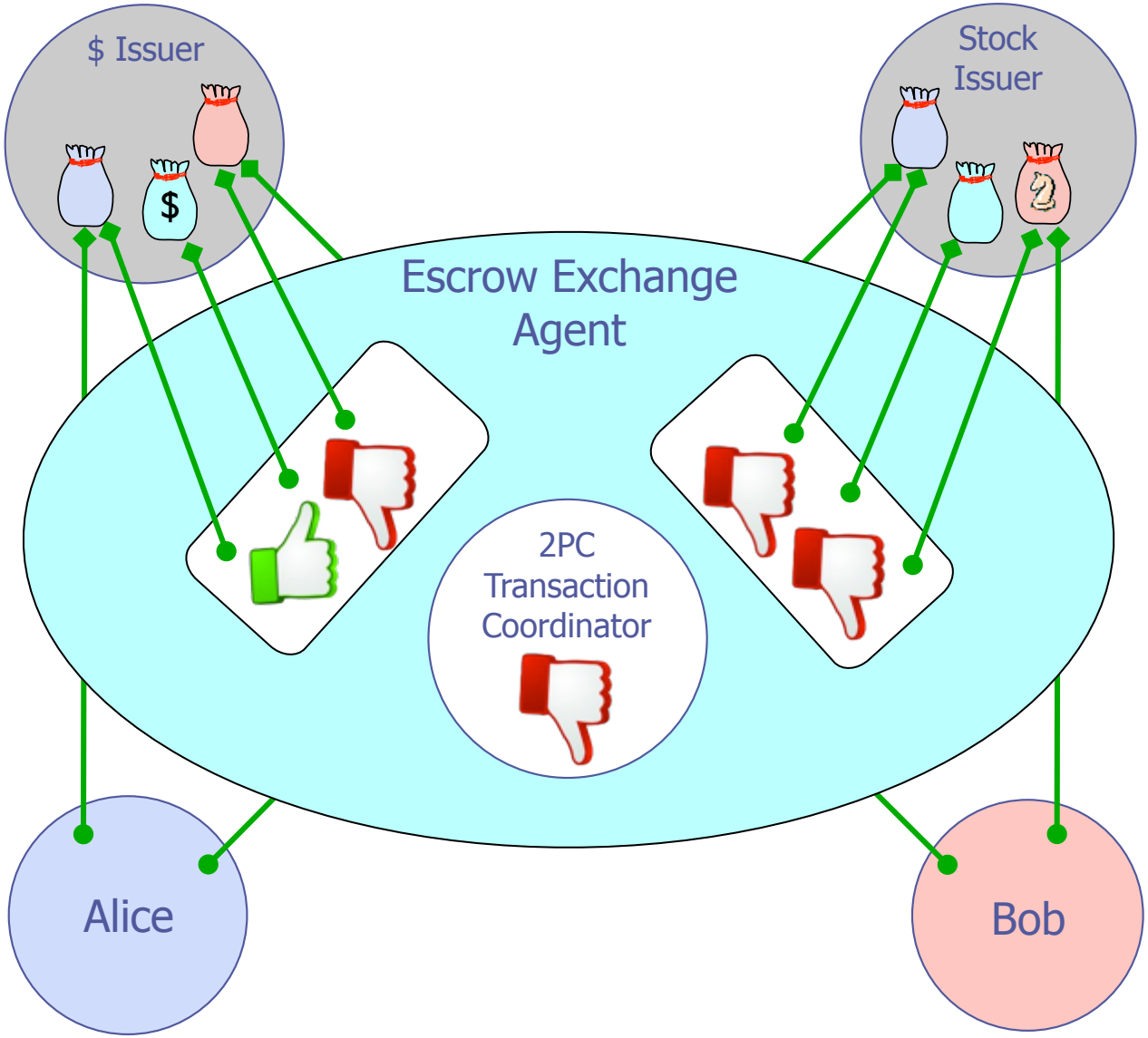
Failed Exchange -- phase 1



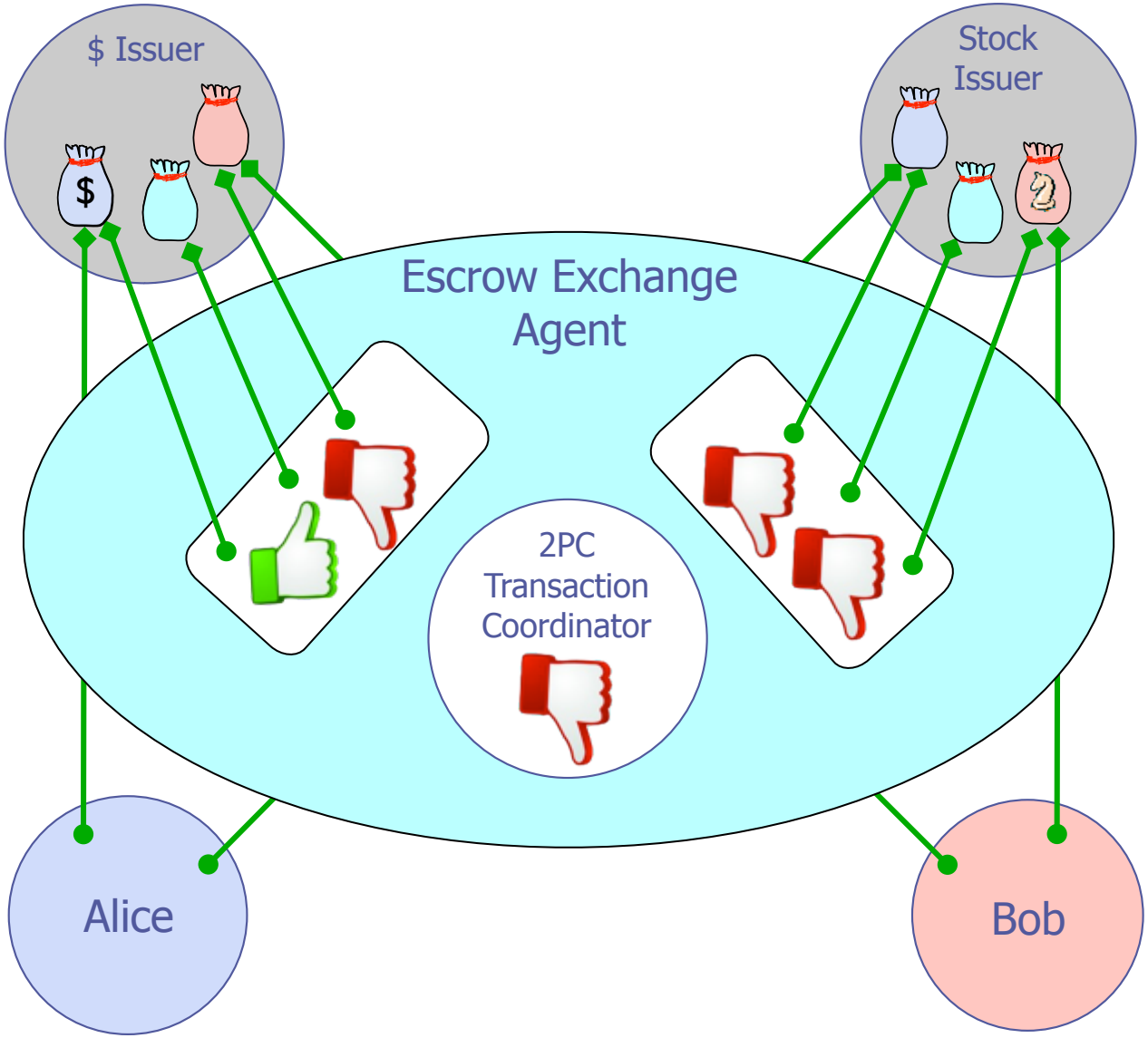
Failed Exchange -- abort



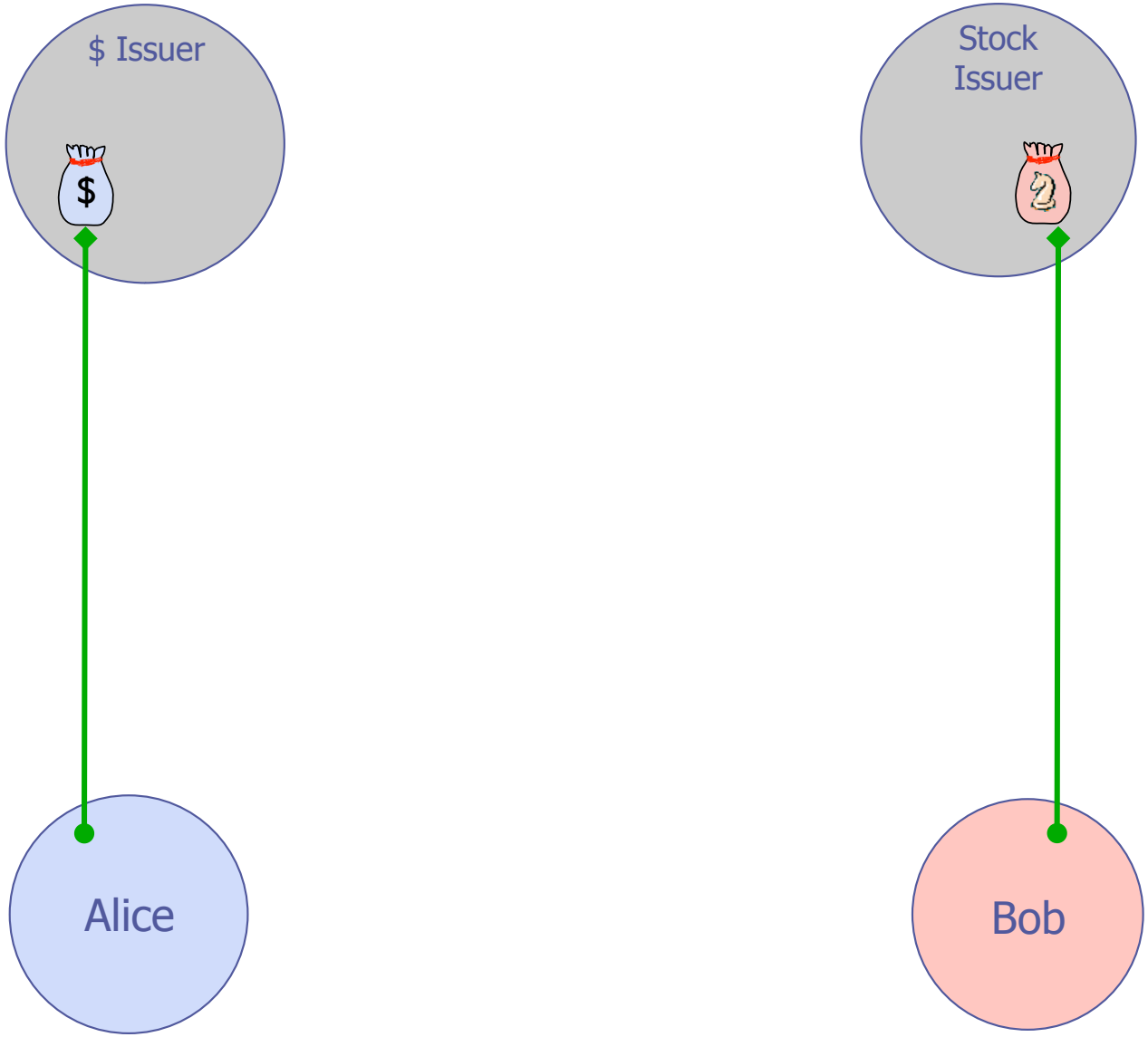
Failed Exchange -- phase 2



Failed Exchange -- phase 2



Failed Exchange -- phase 2



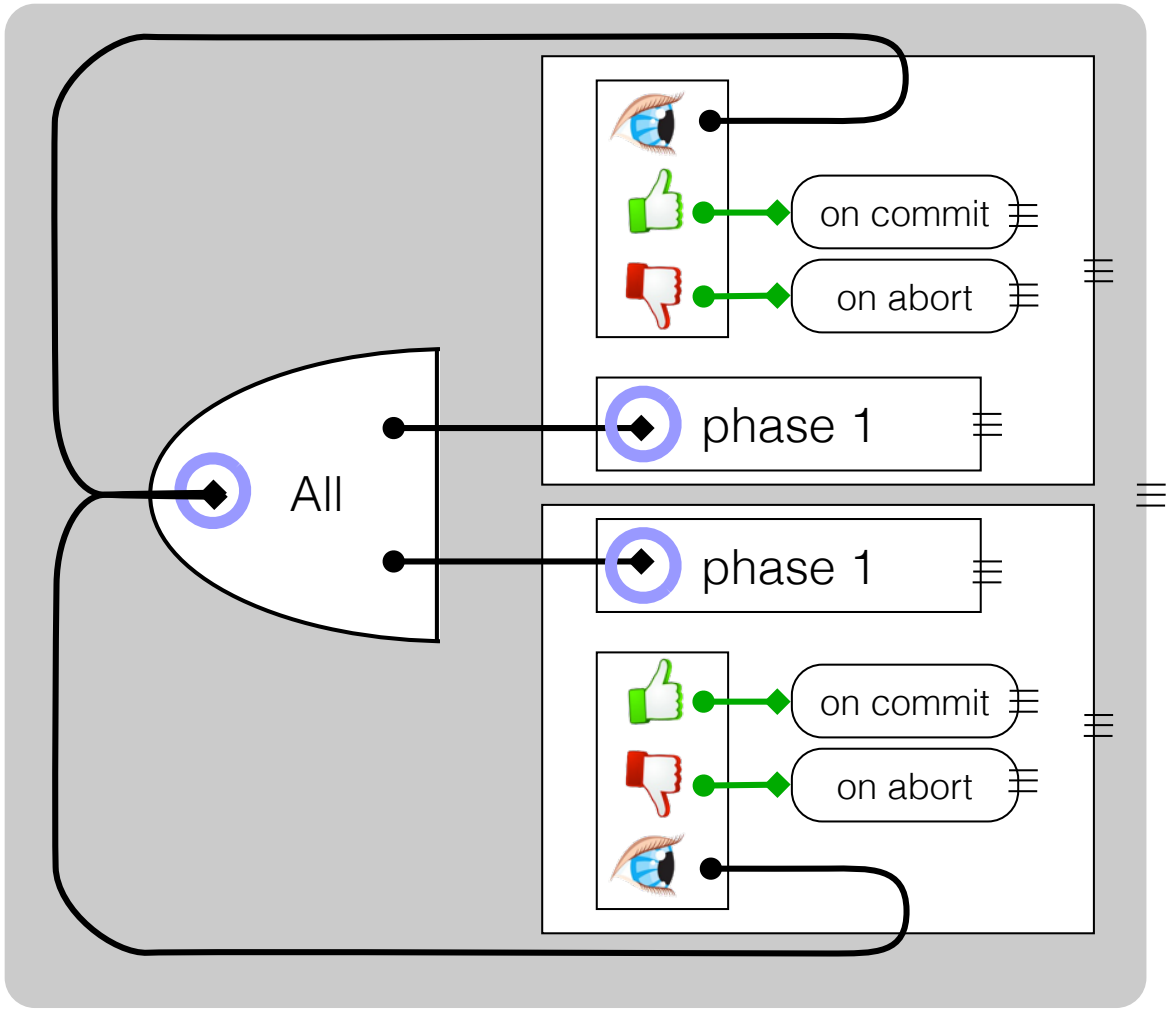
General 2PC Pattern -- transaction coordinator

```
function coordinate(...) {
   var decision = Q.defer();

   decision.resolve(Q.all([
      participantA(decision.promise, ...),
      participantB(decision.promise, ...)]));

   return decision.promise;
}
```

General 2PC Pattern -- participant

```
function participantA(decisionP, ...) {

  Q(decisionP).when(function(_) {        // setup phase 2
    ...;                                 // phase 2: on commit
  }, function(reason) {
    ...;                                 // phase 2: on abort
  });

  return ...;                  // phase 1: try to reserve, reveal whether ok
}
```

General 2PC Pattern



exchange (7 lines) // insecure

```
function exchange(argA, argB) {   // argA from Alice, argB from Bob
  var decision = Q.defer();

  decision.resolve(Q.all([
    transfer(decision.promise, argA.moneySrcP, argB.moneyDstP, argB.moneyNeeded),
    transfer(decision.promise, argB.stockSrcP,   argA.stockDstP,   argA.stockNeeded)]));

  return decision.promise;
}
```

transfer (9 lines) // insecure

```
function transfer(decisionP, srcPurseP, dstPurseP, amount) {
  var escrowPurseP = srcPurseP ! makePurse();

  Q(decisionP).when(function(_) {                    // setup phase 2
    dstPurseP ! deposit(amount, escrowPurseP);
  }, function(reason) {
    srcPurseP ! deposit(amount, escrowPurseP);
  });

  return escrowPurseP ! deposit(amount, srcPurseP);  // phase 1
}
```

transfer (9 lines) // insecure

```
function transfer(decisionP, srcPurseP, dstPurseP, amount) {
  var escrowPurseP = srcPurseP ! makePurse();       // security hole

  Q(decisionP).when(function(_) {                   // setup phase 2
    dstPurseP ! deposit(amount, escrowPurseP);
  }, function(reason) {
    srcPurseP ! deposit(amount, escrowPurseP);
  });

  return escrowPurseP ! deposit(amount, srcPurseP);  // phase 1
}
```

Q.join
Distributed EQ is Mutual acceptability

```
Q.join = function(xP, yP) {
   return Q.all([xP, yP]).when(function([x, y]) {
      if (x === y) {
         return x;
      } else {
         throw new Error("not the same");
}   }); };
```

makePurseMaker (3 lines)
All customers get the same makePurse

```
function makePurseMaker(purse) {
   return function makePurse() { return purse.makePurse(); };
}
```

makeTransfer (+1 lines to secure)
Make mutually acceptable participant

```
function makeTransfer(makeEscrowPurseP) {

  return function transfer(decisionP, srcPurseP, dstPurseP, amount) {
    var escrowPurseP = makeEscrowPurseP ! ();

    Q(decisionP).when(function(_) {                    // setup phase 2
      dstPurseP ! deposit(amount, escrowPurseP);
    }, function(reason) {
      srcPurseP ! deposit(amount, escrowPurseP);
    });


    return escrowPurseP ! deposit(amount, srcPurseP);  // phase 1
}; }
```

makeExchange (+4 lines to secure)
Secure Escrow Exchange Agent

```
function exchange(argA, argB) {  // argA from Alice, argB from Bob
  var transferMoney = makeTransfer(Q.join(argA.makeMoneyEscrowP,
                                          argB.makeMoneyEscrowP));
  var transferStock  = makeTransfer(Q.join(argB.makeStockEscrowP,
                                           argA.makeStockEscrowP));
  var d = Q.defer();
  d.resolve(Q.all([
    transferMoney(d.promise, argA.moneySrcP, argB.moneyDstP, argB.moneyNeeded),
    transferStock (d.promise, argB.stockSrcP,  argA.stockDstP,  argA.stockNeeded)]));
  return d.promise;
}
```

General 2PC Pattern -- transaction stuck

```
function coordinate(...) {
   var decision = Q.defer();

   decision.resolve(Q.all([
      participantA(decision.promise, ...),
      participantB(decision.promise, ...)]));

   return decision.promise;
}
```

General 2PC Pattern -- transaction stuck

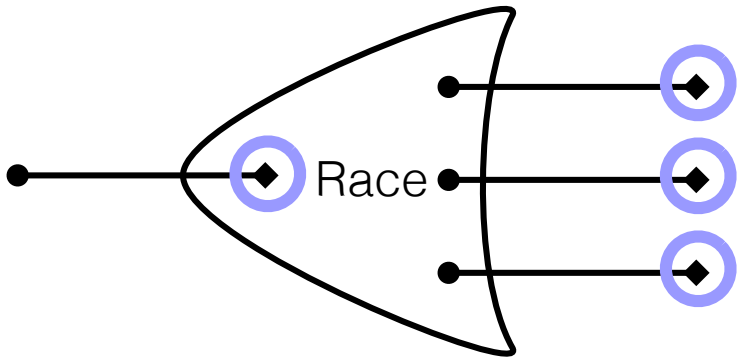
```
function coordinate(...) {
   var decision = Q.defer();

   decision.resolve(Q.all([
      participantA(decision.promise, ...),
      participantB(decision.promise, ...)]));

   return decision.promise;
}
```

General 2PC Pattern -- transaction stuck

```
function coordinate(...) {
  var decision = Q.defer();

  decision.resolve(Q.all([
    participantA(decision.promise, ...),
    participantB(decision.promise, ...)]));

  return decision.promise;
}
```

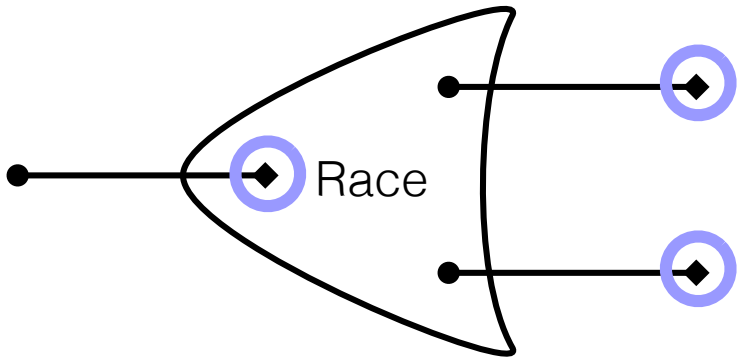
Q.race
Any done?

```
Q.race = function(answerPs) {
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(answer) {
      result.resolve(answer);
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```



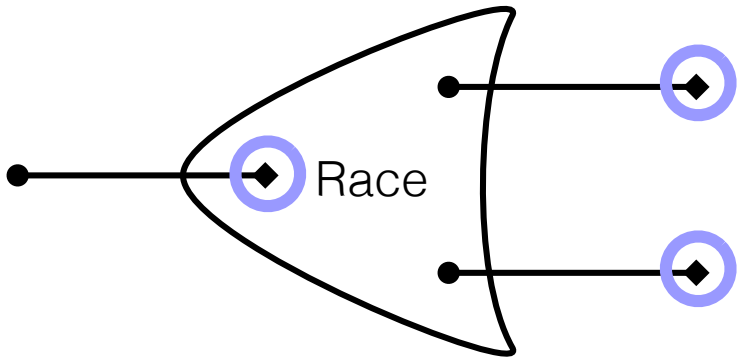
Q.race
Any done?

```
Q.race = function(answerPs) {
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(answer) {
      result.resolve(answer);
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};

var answerP = Q.race(bob ! foo(carol),
                     Q.delay(5000, Q.reject(new Error("timeout"))))]);
```



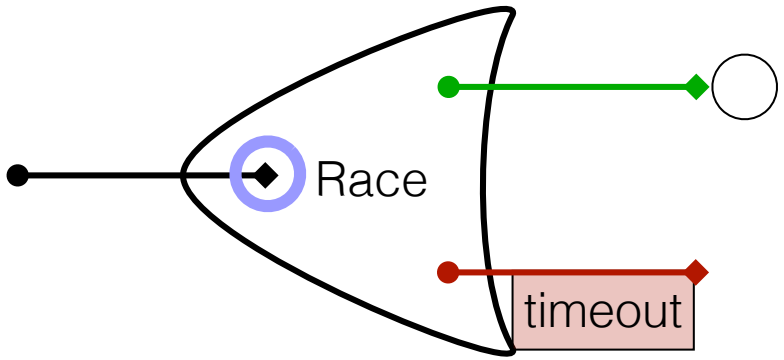
Q.race
Any done?

```
Q.race = function(answerPs) {
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(answer) {
      result.resolve(answer);
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};

var answerP = Q.race(bob ! foo(carol),
                     Q.delay(5000, Q.reject(new Error("timeout"))))]);
```



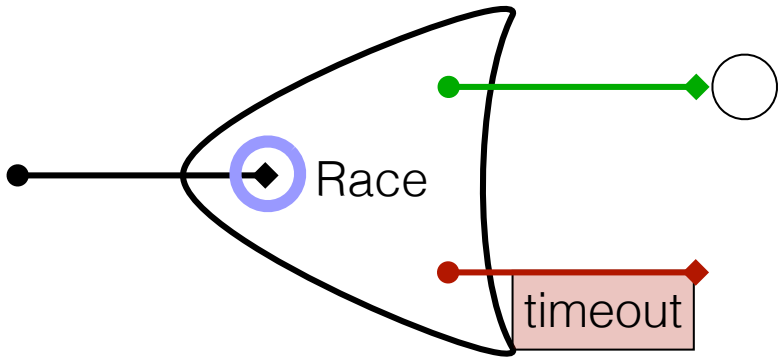
Q.race
Any done?

```
Q.race = function(answerPs) {
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(answer) {
      result.resolve(answer);
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};

var answerP = Q.race(bob ! foo(carol),
                     Q.delay(5000, Q.reject(new Error("timeout"))))]);
```



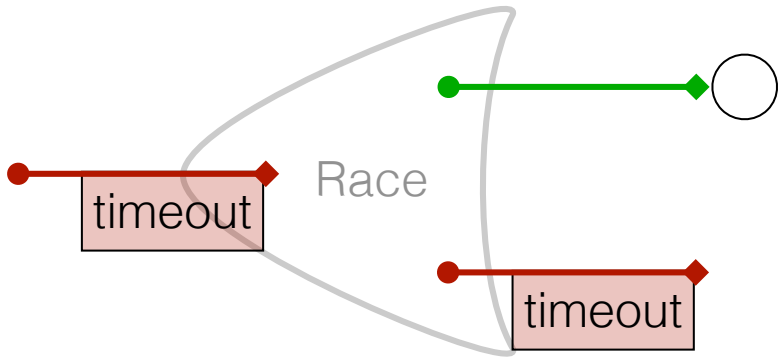
Q.race
Any done?

```
Q.race = function(answerPs) {
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(answer) {
      result.resolve(answer);
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};

var answerP = Q.race(bob ! foo(carol),
                     Q.delay(5000, Q.reject(new Error("timeout"))))]);
```



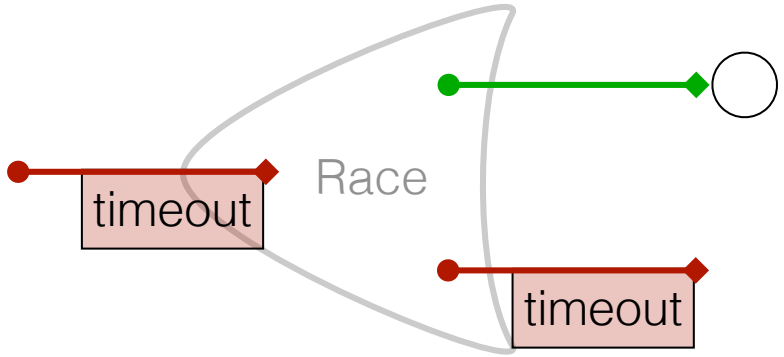
Q.race
Any done?

```
Q.race = function(answerPs) {
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(answer) {
      result.resolve(answer);
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};

var answerP = Q.race(bob ! foo(carol),
                     Q.delay(5000, Q.reject(new Error("timeout"))))]);
```



Q.race
Any done?

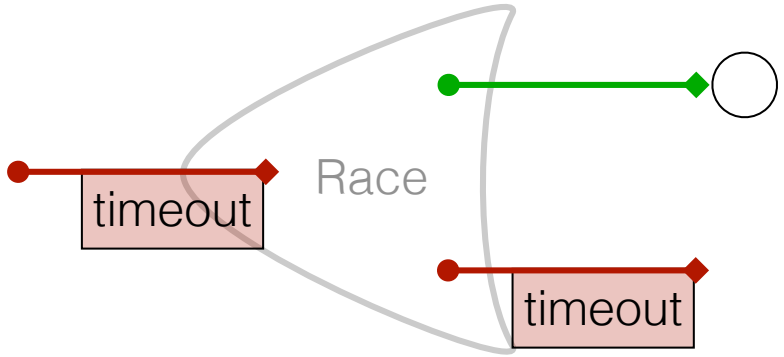
```
Q.race = function(answerPs) {
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(answer) {
      result.resolve(answer);
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```




```
var answerP = Q.race(bob ! foo(carol),
                     Q.delay(5000, Q.reject(new Error("timeout"))))]);
```

Q.race
Any done?

```
Q.race = function(answerPs) {
  var result = Q.defer();

  answerPs.forEach(function(answerP) {
    Q(answerP).when(function(answer) {
      result.resolve(answer);
    }, function(reason) {
      result.resolve(Q.reject(reason));
  }); });
  return result.promise;
};
```

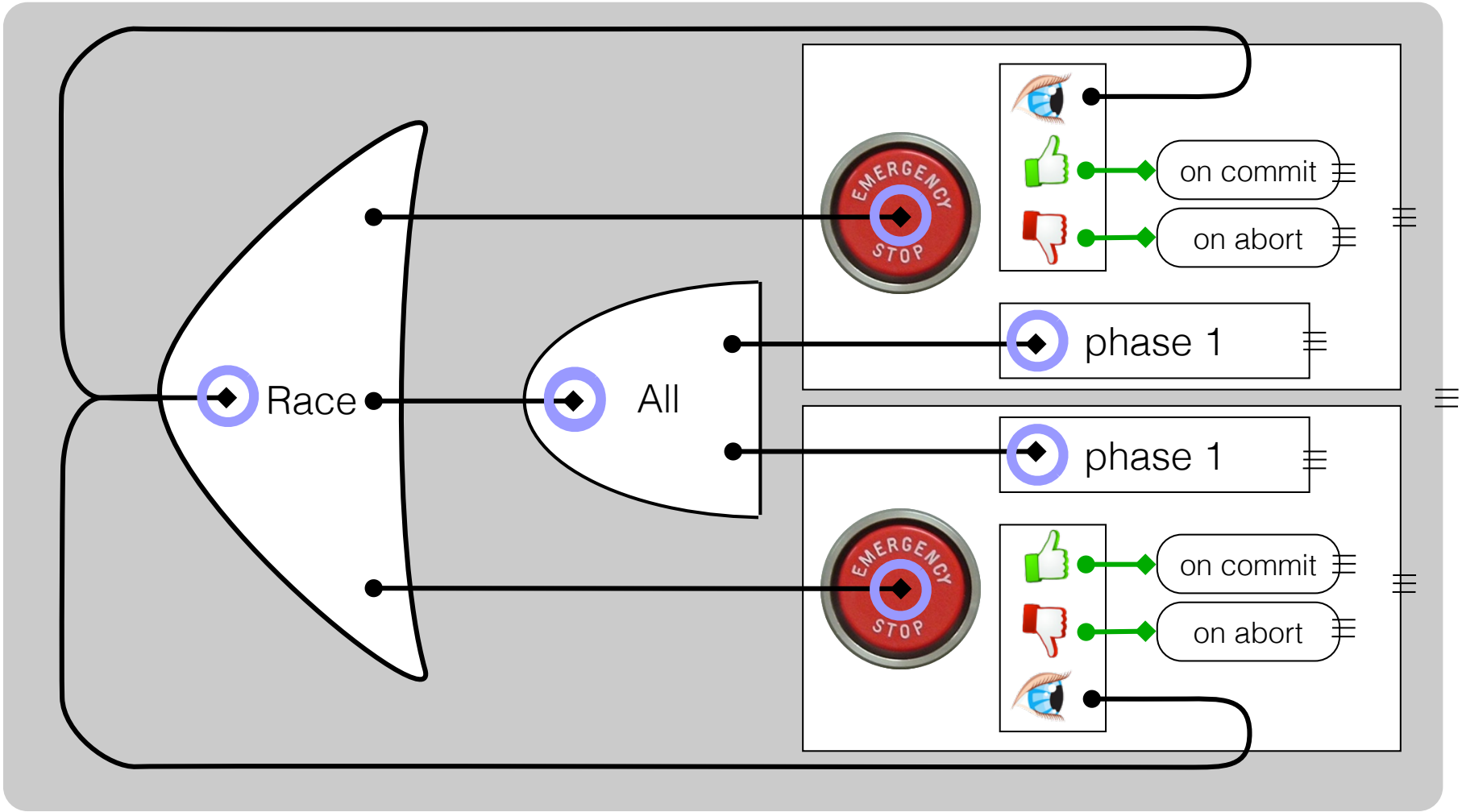



```
var answerP = Q.race(bob ! foo(carol),
                     Q.delay(5000, Q.reject(new Error("timeout"))))]);
```

General 2PC Pattern -- transaction unstuck

```
function coordinate(...) {
   var decision = Q.defer();

   decision.resolve(Q.race([Q.all([
         participantA(decision.promise, ...),
         participantB(decision.promise, ...)]),
      failOnly(cancelAP),
      failOnly(cancelBP)]));

   return decision.promise;
}
```

General Unstuck 2PC Pattern



makeExchange (+2 lines to unstick)
Secure Unstuck Escrow Exchange Agent

```
function failOnly(cancelP) { return Q(cancelP).when(function(cancel) { throw cancel; }); }

function exchange(argA, argB) {   // argA from Alice, argB from Bob
  var transferMoney = makeTransfer(Q.join(argA.makeMoneyEscrowP,
                                          argB.makeMoneyEscrowP));
  var transferStock  = makeTransfer(Q.join(argB.makeStockEscrowP,
                                           argA.makeStockEscrowP));
  var d = Q.defer();
  d.resolve(Q.race([Q.all([
      transferMoney(d.promise, argA.moneySrcP, argB.moneyDstP, argB.moneyNeeded),
      transferStock (d.promise, argB.stockSrcP,  argA.stockDstP,  argA.stockNeeded)]),
    failOnly(argA.cancelP), failOnly(argB.cancelP)]));
  return d.promise;
}
```

Conclusions

20 lines of general money/bank/stock code
16 lines of 2PC code + (**5**+**2**) **lines** to secure it

Brevity is evidence of good abstractions for

- security
- event loop concurrency
- distribution, decentralization
- partial failure

JavaScript becomes secure distributed language