The Blindingly Simple Protocol Language

Amit K. Chopra

Lancaster University

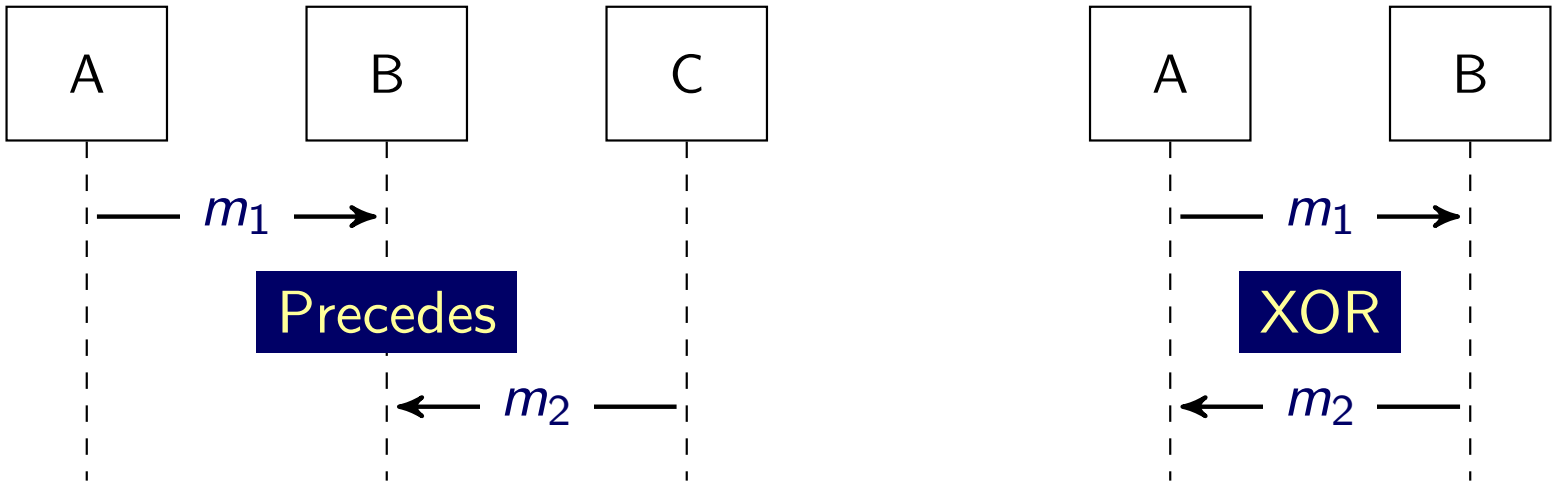
Information Protocols (Singh, 2011)

An information-based protocol language

- Declarative
  - Bag of protocols (message specification is atomic protocol)
- Specifies a decentralized information object
- Specifies information causality
  - The messages an agent can send depends upon what it knows
  - Local state approach
- Asynchronous communication
- Requires no ordering guarantees from infrastructure

Traditional Specifications: Procedural

Low-level, over-specified protocols, easily wrong



- Traditional approaches
  - Emphasize arbitrary ordering and occurrence constraints
  - Then work hard to deal with those constraints
- Our philosophy: The Zen of Distributed Computing
  - Necessary ordering constraints fall out from *causality*
  - Necessary occurrence constraints fall out from *integrity*
  - Unnecessary constraints: simply *ignore* such

Properties of Participants

- Autonomy
- Myopia
  - All choices must be local
  - Correctness must not rely on future interactions
- Heterogeneity: local $\neq$ internal
  - Local state (projection of global state, which is stored nowhere)
    - Public or observable
    - Typically, must be revealed for correctness
  - Internal state
    - Private
    - Must never be revealed: to avoid false coupling
- Shared nothing representation of local state
  - Enact via messaging

BSPL, the Blindingly Simple Protocol Language

Main ideas

- Only *two* syntactic notions
  - Declare a message schema: as an atomic protocol
  - Declare a composite protocol: as a bag of references to protocols
- Parameters are central
  - Provide a basis for expressing meaning in terms of bindings in protocol instances
  - Yield unambiguous specification of compositions through public parameters
  - Capture progression of a role's knowledge
  - Capture the completeness of a protocol enactment
  - Capture uniqueness of enactments through keys

Key Parameters in BSPL

Marked as ⌜key⌝

- All the key parameters *together* form the key
- Each protocol must define at least one key parameter
- Each message or protocol reference must have at least one key parameter in common with the protocol in whose declaration it occurs
- The key of a protocol provides a basis for the uniqueness of its enactments

Parameter Adornments in BSPL

Capture the essential causal structure of a protocol (for simplicity, assume all parameters are strings)

- ⌜in⌝: Information that must be provided to instantiate a protocol
  - Bindings must exist locally in order to proceed
  - Bindings must be produced through some other protocol
- ⌜out⌝: Information that is generated by the protocol instances
  - Bindings can be fed into other protocols through their ⌜in⌝ parameters, thereby accomplishing composition
  - A standalone protocol must adorn all its public parameters ⌜out⌝
- ⌜nil⌝: Information that is absent from the protocol instance
  - Bindings must not exist

The *Initiate* protocol

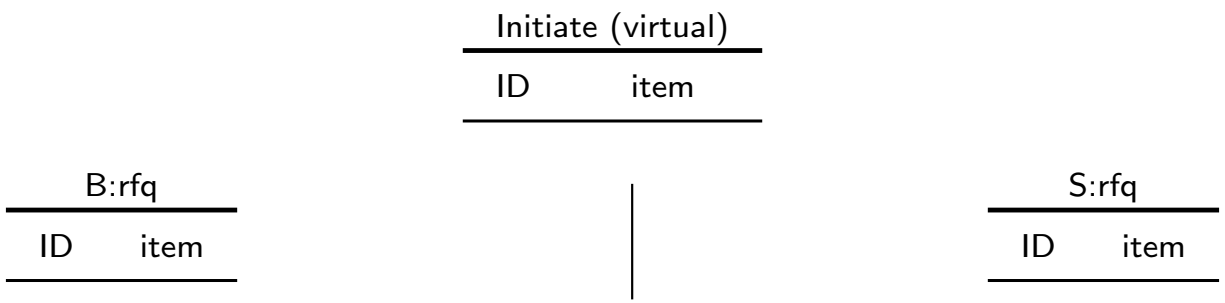
Global state (virtual): Aggregation of the local states

```
Initiate {
 role B, S
 parameter out ID key, out item

 B ↦ S: rfq[out ID key, out item]
}
```

The *Initiate* protocol

Global state (virtual): Aggregation of the local states

```
Initiate {
 role B, S
 parameter out ID key, out item

 B ↦ S: rfq[out ID key, out item]
}
```



The *Initiate* protocol

Global state (virtual): Aggregation of the local states

```
Initiate {
 role B, S
 parameter out ID key, out item

 B ↦ S: rfq[out ID key, out item]
}
```

Initiate (virtual)

ID	item
1	fig

B:rfq

ID	item
1	fig

S:rfq

ID	item

The *Initiate* protocol

Global state (virtual): Aggregation of the local states

```
Initiate {
 role B, S
 parameter out ID key, out item

 B ↦ S: rfq[out ID key, out item]
}
```

Initiate (virtual)

ID	item
1	fig
5	jam

B:rfq

ID	item
1	fig
5	jam

S:rfq

ID	item

The *Initiate* protocol

Global state (virtual): Aggregation of the local states

```
Initiate {
 role B, S
 parameter out ID key, out item

 B ↦ S: rfq[out ID key, out item]
}
```

Initiate (virtual)

ID	item
1	fig
5	jam

B:rfq

ID	item
1	fig
5	jam

S:rfq

ID	item
5	jam

The *Initiate* protocol

Global state (virtual): Aggregation of the local states

```
Initiate {
 role B, S
 parameter out ID key, out item

 B ↦ S: rfq[out ID key, out item]
}
```

Initiate (virtual)

ID	item
1	fig
5	jam

B:rfq

ID	item
1	fig
5	jam
×1	apple

S:rfq

ID	item
5	jam

The *Initiate* protocol

Global state (virtual): Aggregation of the local states

```
Initiate {
 role B, S
 parameter out ID key, out item

 B ↦ S: rfq[out ID key, out item]
}
```

Initiate (virtual)

ID	item
1	fig
5	jam
8	fig

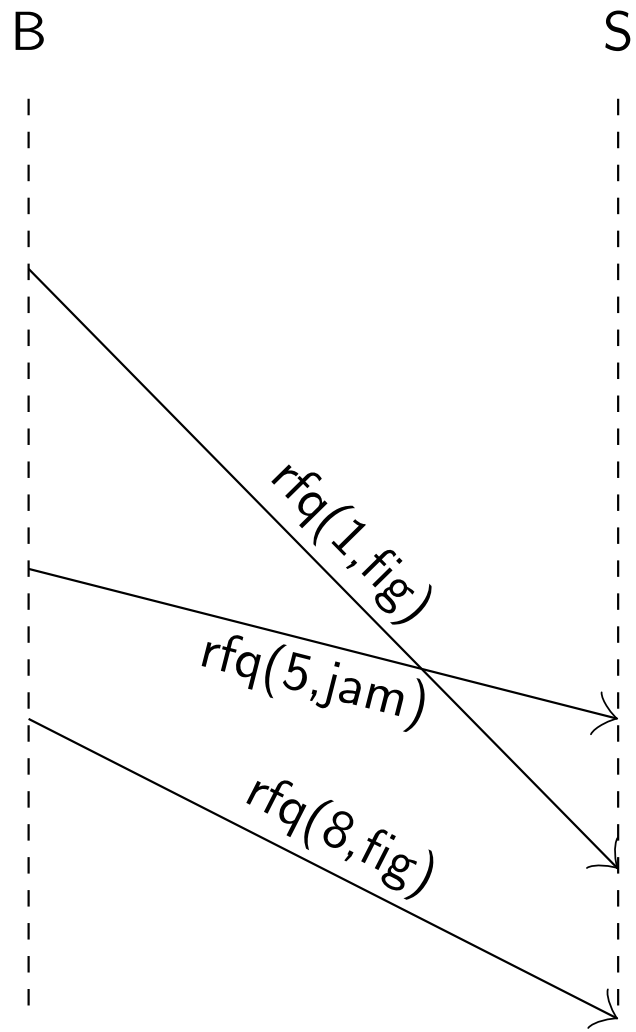
B:rfq

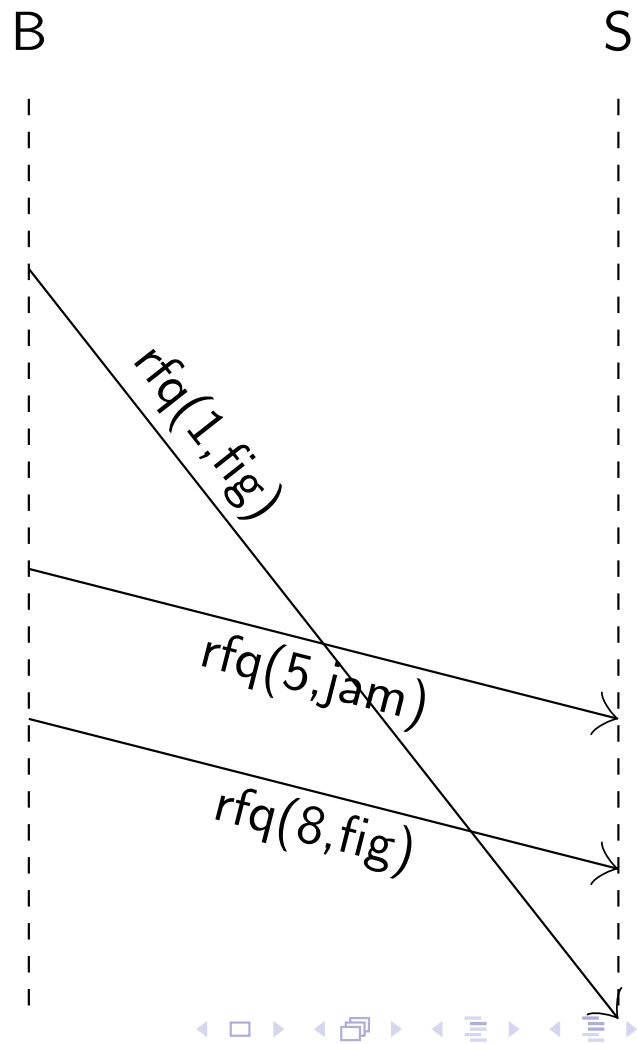
ID	item
1	fig
5	jam
8	fig

S:rfq

ID	item
5	jam

The Foregoing Interaction as possible UML Interaction Diagrams (Assuming all messages land)





The *Offer* Protocol

```
Offer {
 role B, S
 parameter out ID key, out item, out price

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]
}
```

The *Offer* Protocol

```
Offer {
 role B, S
 parameter out ID key, out item, out price

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]
}
```

Offer (virtual)

ID	item	price
1	fig	

B:rfq

ID	item
1	fig

B:quote

ID	item	price

S:rfq

ID	item
1	fig

S:quote

ID	item	price

The *Offer* Protocol

```
Offer {
 role B, S
 parameter out ID key, out item, out price

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]
}
```

Offer (virtual)

ID	item	price
1	fig	10

B:rfq

ID	item
1	fig

B:quote

ID	item	price

S:rfq

ID	item
1	fig

S:quote

ID	item	price
1	fig	10

The *Offer* Protocol

```
Offer {
 role B, S
 parameter out ID key, out item, out price

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]
}
```

Offer (virtual)

ID	item	price
1	fig	10

B:rfq

ID	item
1	fig

B:quote

ID	item	price
1	fig	10

S:rfq

ID	item
1	fig

S:quote

ID	item	price
1	fig	10

The *Offer* Protocol

```
Offer {
 role B, S
 parameter out ID key, out item, out price

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]
}
```

Offer (virtual)

ID	item	price
1	fig	10

B:rfq

ID	item
1	fig

B:quote

ID	item	price
1	fig	10

S:rfq

ID	item
1	fig

S:quote

ID	item	price
1	fig	10
×4	fig	10

The *Decide Offer* Protocol

Choice: *accept* and a *reject* with the same ID *cannot* both occur

```
Decide Offer {
 role B, S
 parameter out ID key, out item, out price, out decision

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]

 B ↦ S: accept[in ID, in item, in price, out decision]
 B ↦ S: reject[in ID, in item, in price, out decision]
}
```

The *Decide Offer* Protocol

Choice: *accept* and a *reject* with the same ID *cannot* both occur

```
Decide Offer {
 role B, S
 parameter out ID key, out item, out price, out decision

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]

 B ↦ S: accept[in ID, in item, in price, out decision]
 B ↦ S: reject[in ID, in item, in price, out decision]
}
```

Decide Offer (virtual)

ID	item	price	decision
1	fig	10	

B:rfq

ID	item
1	fig

B:quote

ID	item	price
1	fig	10

B:accept

ID	item	price	decision

B:reject

ID	item	price	decision

The *Decide Offer* Protocol

Choice: *accept* and a *reject* with the same ID *cannot* both occur

```
Decide Offer {
 role B, S
 parameter out ID key, out item, out price, out decision

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]

 B ↦ S: accept[in ID, in item, in price, out decision]
 B ↦ S: reject[in ID, in item, in price, out decision]
}
```

Decide Offer (virtual)

ID	item	price	decision
1	fig	10	nice

B:rfq

ID	item
1	fig

B:quote

ID	item	price
1	fig	10

B:accept

ID	item	price	decision
1	fig	10	nice

B:reject

ID	item	price	decision

The *Decide Offer* Protocol

Choice: *accept* and a *reject* with the same ID *cannot* both occur

```
Decide Offer {
 role B, S
 parameter out ID key, out item, out price, out decision

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]

 B ↦ S: accept[in ID, in item, in price, out decision]
 B ↦ S: reject[in ID, in item, in price, out decision]
}
```

Decide Offer (virtual)

ID	item	price	decision
1	fig	10	nice

B:rfq

ID	item
1	fig

B:quote

ID	item	price
1	fig	10

B:accept

ID	item	price	decision
1	fig	10	nice

B:reject

ID	item	price	decision
×1	fig	10	nice

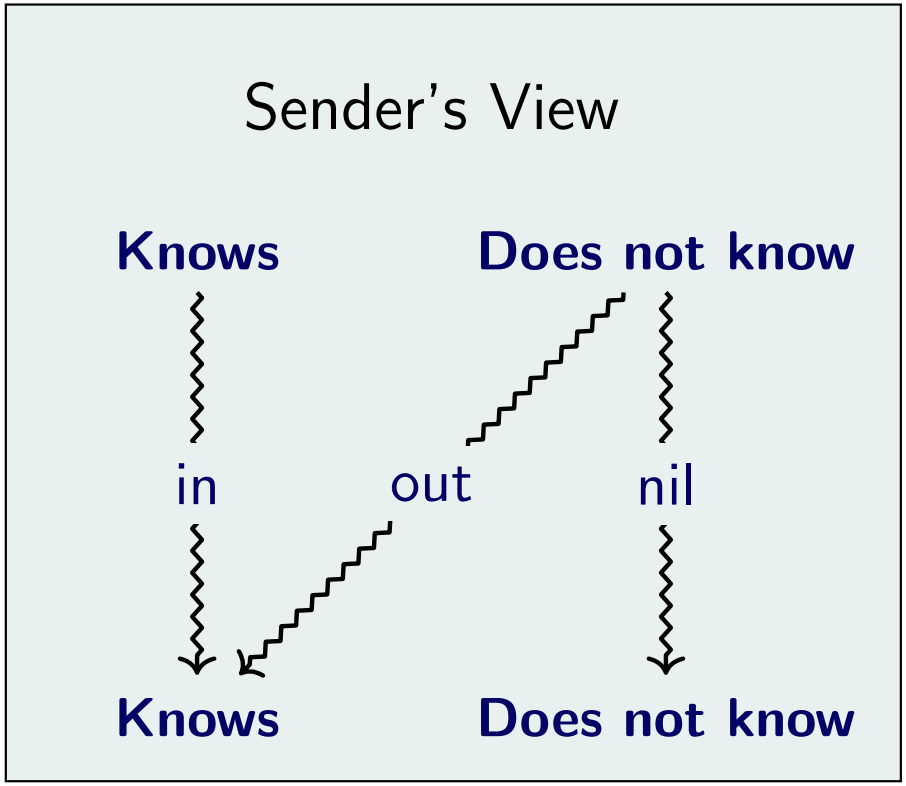
The *Purchase* Protocol

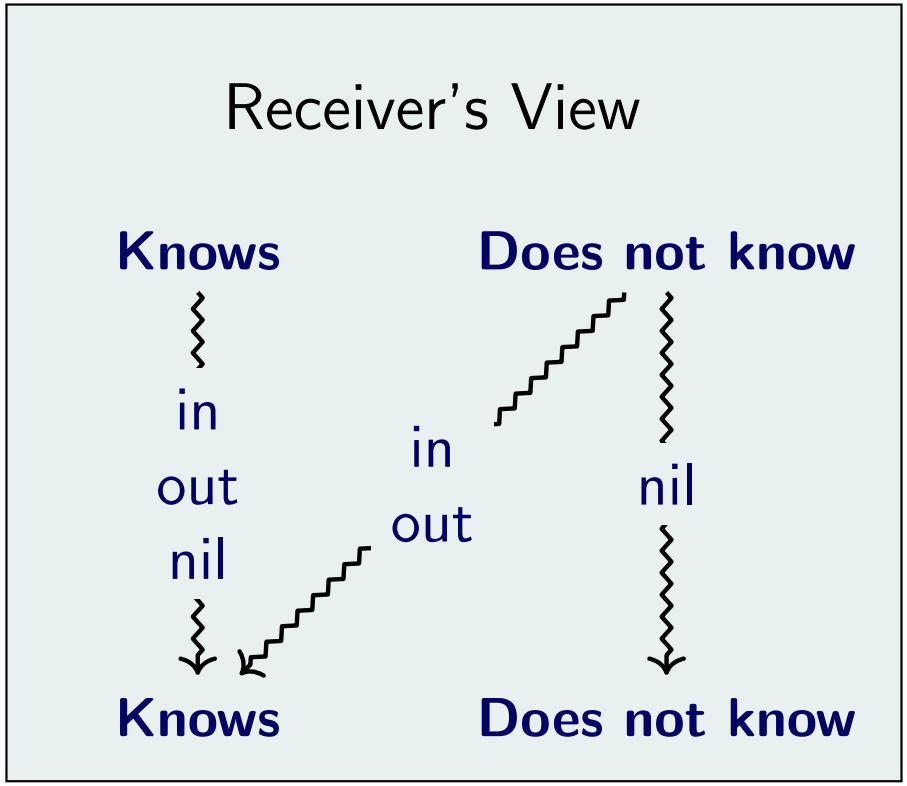
```
Purchase {
 role B, S, Shipper
 parameter out ID key, out item, out price, out outcome
 private address, resp

 B ↦ S: rfq[out ID, out item]
 S ↦ B: quote[in ID, in item, out price]
 B ↦ S: accept[in ID, in item, in price, out address, out resp]
 B ↦ S: reject[in ID, in item, in price, out outcome, out resp]

 S ↦ Shipper: ship[in ID, in item, in address]
 Shipper ↦ B: deliver[in ID, in item, in address, out outcome]
}
```

- *reject* conflicts with *accept* on resp (a *private* parameter)
- *reject* or *deliver* must occur for completion (to bind outcome)

Knowledge and Viability

When is a message viable? What effect does it have on a role's local knowledge?

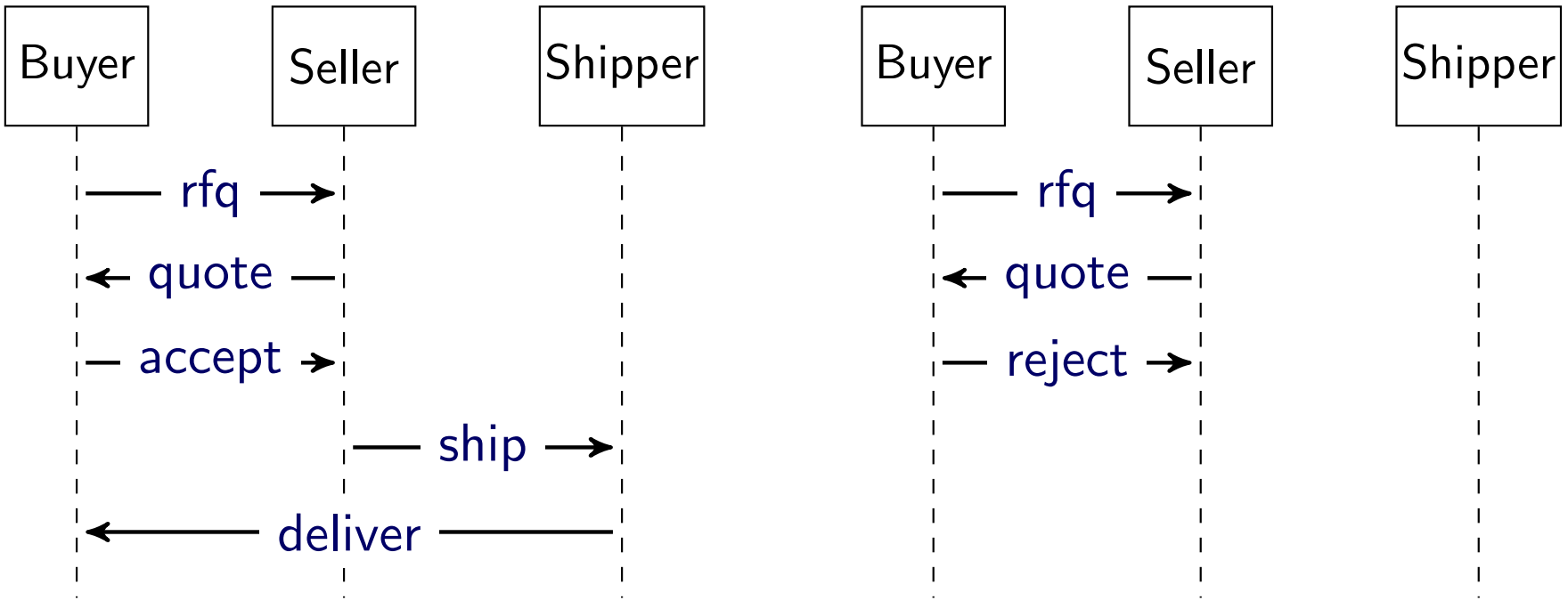




- Knowledge increases monotonically at each role
- An ⌜out⌝ parameter **creates** and transmits knowledge
- An ⌜in⌝ parameter transmits knowledge
- Repetitions through multiple paths are harmless and superfluous

Possible Enactments as Sets of Local Histories

Each participant's local history: set of messages sent and received



Polymorphism

Same message schema, but different adornments

```
Flexible-Offer {
 role B, S
 parameter in ID key, out item, out price, out qID

 B ↦ S: rfq[in ID, out item, nil price]
 B ↦ S: rfq[in ID, out item, out price]

 S ↦ B: quote[in ID, in item, out price, out qID]
 S ↦ B: quote[in ID, in item, in price, out qID]
}
```

- B has priority on generating price, but if it chooses not to (by sending *rfq* without price), then S can generate it

Remark on Control versus Information Flow

- Control flow
  - Natural within a single computational thread
  - Exemplified by conditional branching
  - Presumes master-slave relationship across threads
  - Impossible between mutually autonomous parties because neither controls the other
  - May sound appropriate, but only because of long habit
- Information flow
  - Natural across computational threads
  - Explicitly tied to causality

Information Centrism

Characterize each interaction purely in terms of information

- Explicit causality
  - Flow of information coincides with flow of causality
  - No hidden coordination
- Keys
  - Uniqueness
  - Basis for completion
- Integrity
  - Parameter has at most one value in any enactment
- Immutability
  - Durability
  - Robustness: insensitivity to
    - Reordering by infrastructure
    - Retransmission: one delivery is all it needs