Applying Generative Communication to Symbolic Computation in Common Lisp

Jónathan Heras, Vico Pascual and Julio Rubio

Departamento de Matemáticas y Computación
Universidad de La Rioja
Spain

2nd European Lisp Symposium 2009
May 29, 2009

Table of Contents

1. Introduction
2. Architecture of the system
3. Communication among modules
4. Clients of our system
5. Conclusions and Further Work

Table of Contents

1 Introduction

2 Architecture of the system

3 Communication among modules

4 Clients of our system

5 Conclusions and Further Work

Introduction

Kenzo is a Common Lisp system devoted to Symbolic Computation in Algebraic Topology

Introduction

Kenzo is a Common Lisp system devoted to Symbolic Computation in Algebraic Topology

Accessibility and usability are two weak points in Kenzo

Introduction

Kenzo is a Common Lisp system devoted to Symbolic Computation in Algebraic Topology
Accessibility and usability are two weak points in Kenzo
An *intermediary layer* was designed to interact with the Kenzo system

Introduction

Kenzo is a Common Lisp system devoted to Symbolic Computation in Algebraic Topology
Accessibility and usability are two weak points in Kenzo
An *intermediary layer* was designed to interact with the Kenzo system
Based on Microkernel pattern

Introduction

Kenzo is a Common Lisp system devoted to Symbolic Computation in Algebraic Topology
Accessibility and usability are two weak points in Kenzo
An *intermediary layer* was designed to interact with the Kenzo system
Based on Microkernel pattern

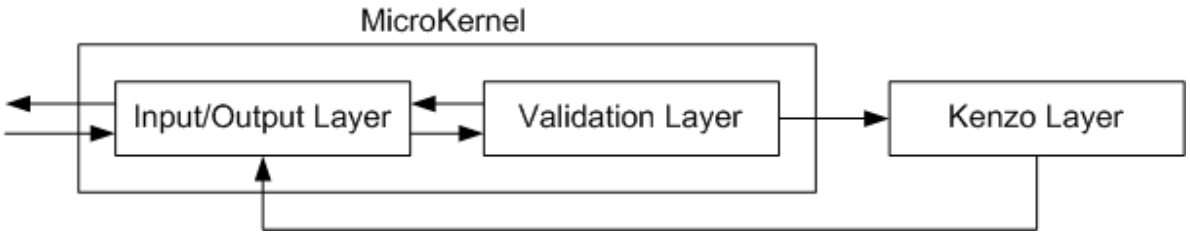


Figure: Simplified diagram of the previous architecture.

Introduction

Kenzo is a Common Lisp system devoted to Symbolic Computation in Algebraic Topology
Accessibility and usability are two weak points in Kenzo
An *intermediary layer* was designed to interact with the Kenzo system
Based on Microkernel pattern

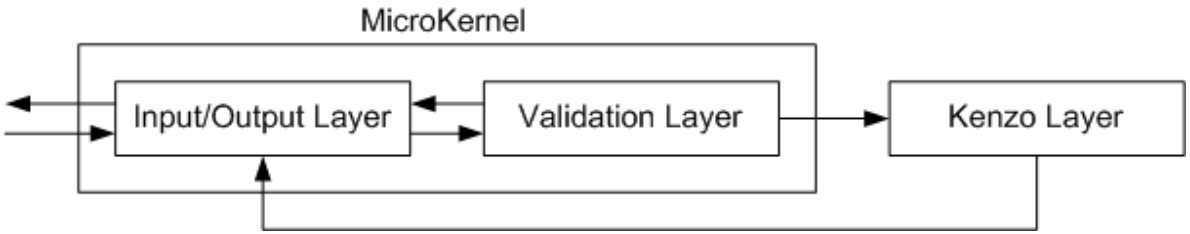


Figure: Simplified diagram of the previous architecture.

Drawbacks:

Introduction

Kenzo is a Common Lisp system devoted to Symbolic Computation in Algebraic Topology
Accessibility and usability are two weak points in Kenzo
An *intermediary layer* was designed to interact with the Kenzo system
Based on Microkernel pattern

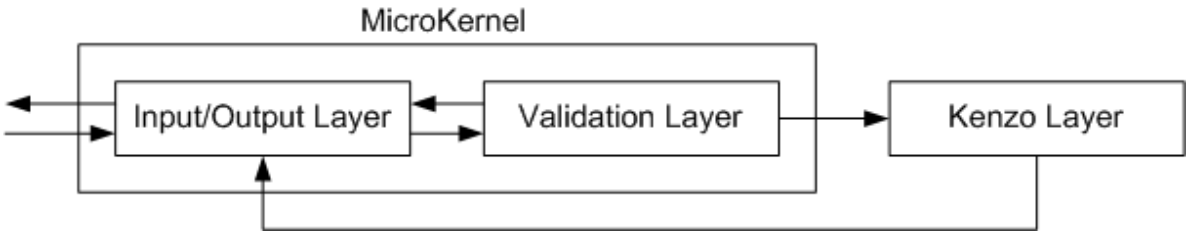


Figure: Simplified diagram of the previous architecture.

Drawbacks:

- Message passing style
- Recalculations

Table of Contents

1 Introduction

2 Architecture of the system
- Linda Model and Shared Repository
- An adapted implementation of the Linda Model
- An example of complete computation

3 Communication among modules

4 Clients of our system

5 Conclusions and Further Work

Linda Model and Shared Repository

Linda Model:

Linda Model and Shared Repository

Linda Model:

- Based on Generative Communication

Linda Model and Shared Repository

Linda Model:

- Based on Generative Communication
- Asynchronous communication is performed by means of the insertion and extraction of data over a shared memory

Linda Model and Shared Repository

Linda Model:

- Based on Generative Communication
- Asynchronous communication is performed by means of the insertion and extraction of data over a shared memory
- Three ingredients:

Linda Model and Shared Repository

Linda Model:

- Based on Generative Communication
- Asynchronous communication is performed by means of the insertion and extraction of data over a shared memory
- Three ingredients:
  - *tuple space*: the shared memory repository

Linda Model and Shared Repository

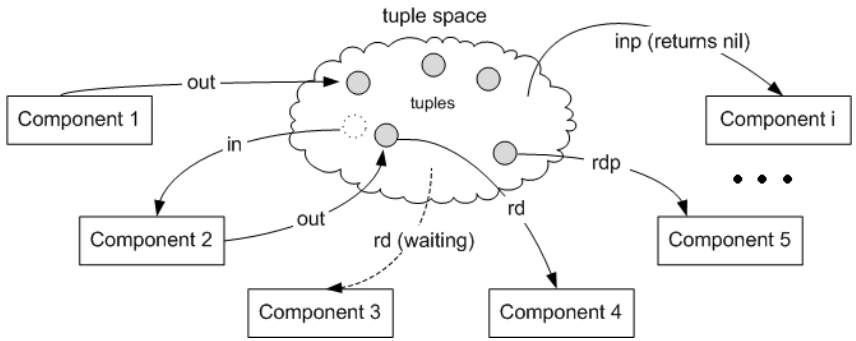
Linda Model:

- Based on Generative Communication
- Asynchronous communication is performed by means of the insertion and extraction of data over a shared memory
- Three ingredients:
  - *tuple space*: the shared memory repository
  - *tuples*: the data in the Linda Model, tuples can contain *actual* and *formal* items, e.g, `(3 ? 4.3)`

Linda Model and Shared Repository

Linda Model:

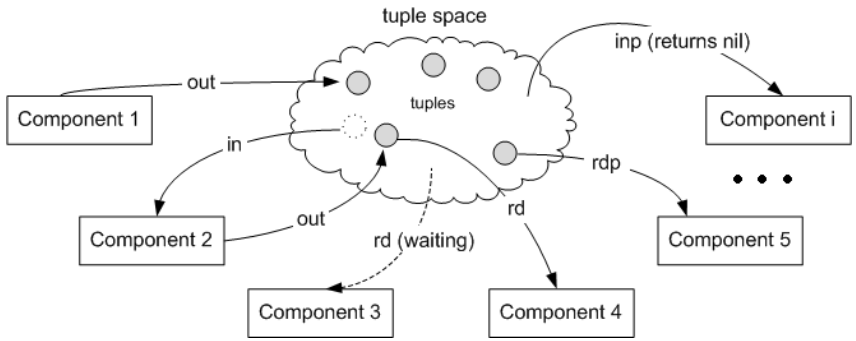
- Based on Generative Communication
- Asynchronous communication is performed by means of the insertion and extraction of data over a shared memory
- Three ingredients:
  - *tuple space*: the shared memory repository
  - *tuples*: the data in the Linda Model, tuples can contain *actual* and *formal* items, e.g, `(3 ? 4.3)`
  - *operations*:



Linda Model and Shared Repository

Linda Model:

- Based on Generative Communication
- Asynchronous communication is performed by means of the insertion and extraction of data over a shared memory
- Three ingredients:
  - *tuple space*: the shared memory repository
  - *tuples*: the data in the Linda Model, tuples can contain *actual* and *formal* items, e.g, `(3 ? 4.3)`
  - *operations*:

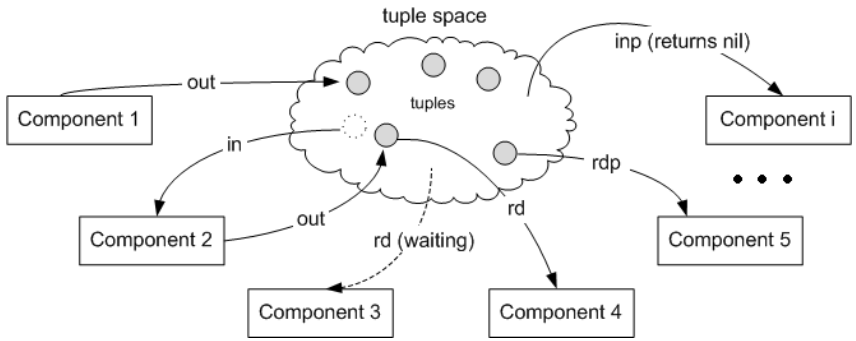


Shared repository pattern:

Linda Model and Shared Repository

Linda Model:

- Based on Generative Communication
- Asynchronous communication is performed by means of the insertion and extraction of data over a shared memory
- Three ingredients:
  - *tuple space*: the shared memory repository
  - *tuples*: the data in the Linda Model, tuples can contain *actual* and *formal* items, e.g, `(3 ? 4.3)`
  - *operations*:



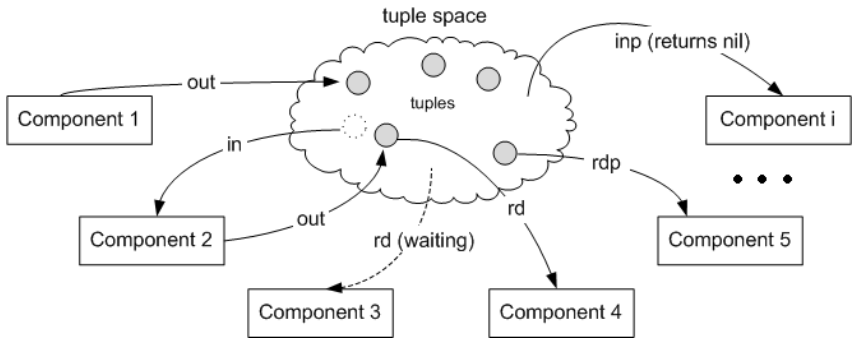
Shared repository pattern:

- Add the nuance that a component has no knowledge of:

Linda Model and Shared Repository

Linda Model:

- Based on Generative Communication
- Asynchronous communication is performed by means of the insertion and extraction of data over a shared memory
- Three ingredients:
  - *tuple space*: the shared memory repository
  - *tuples*: the data in the Linda Model, tuples can contain *actual* and *formal* items, e.g, `(3 ? 4.3)`
  - *operations*:



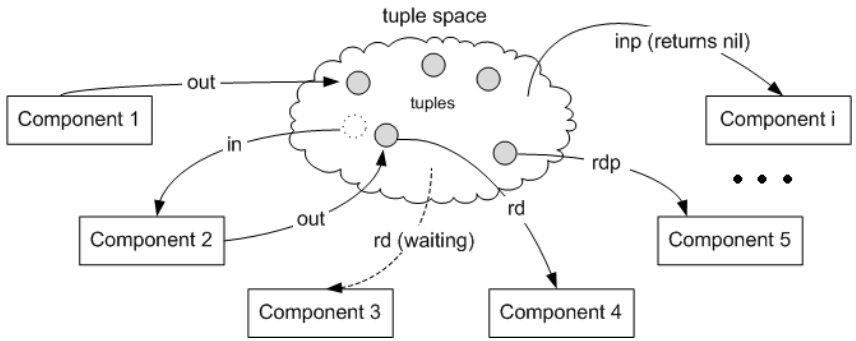
Shared repository pattern:

- Add the nuance that a component has no knowledge of:
  - what components have produced the data it uses,

Linda Model and Shared Repository

Linda Model:

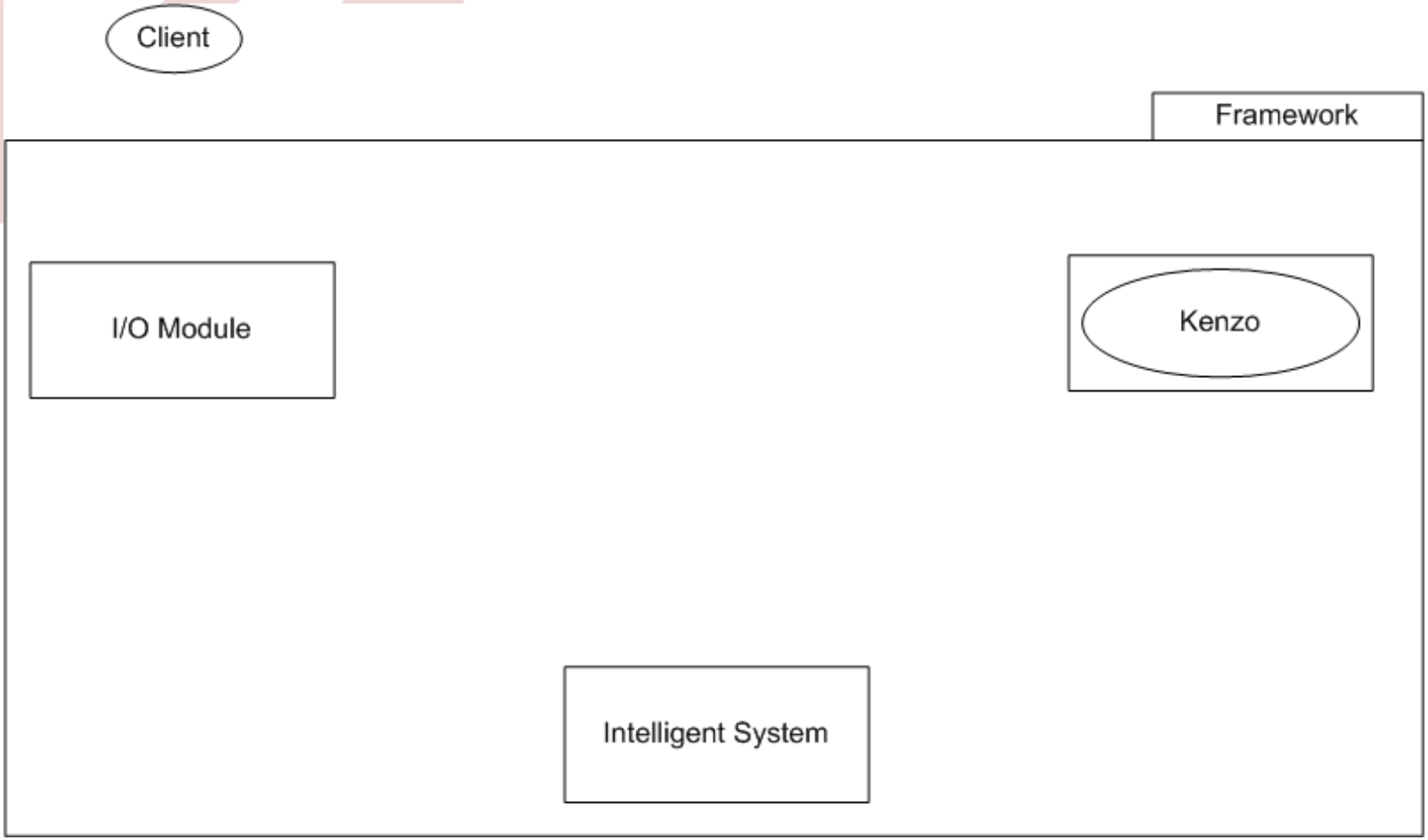
- Based on Generative Communication
- Asynchronous communication is performed by means of the insertion and extraction of data over a shared memory
- Three ingredients:
  - *tuple space*: the shared memory repository
  - *tuples*: the data in the Linda Model, tuples can contain *actual* and *formal* items, e.g, `(3 ? 4.3)`
  - *operations*:



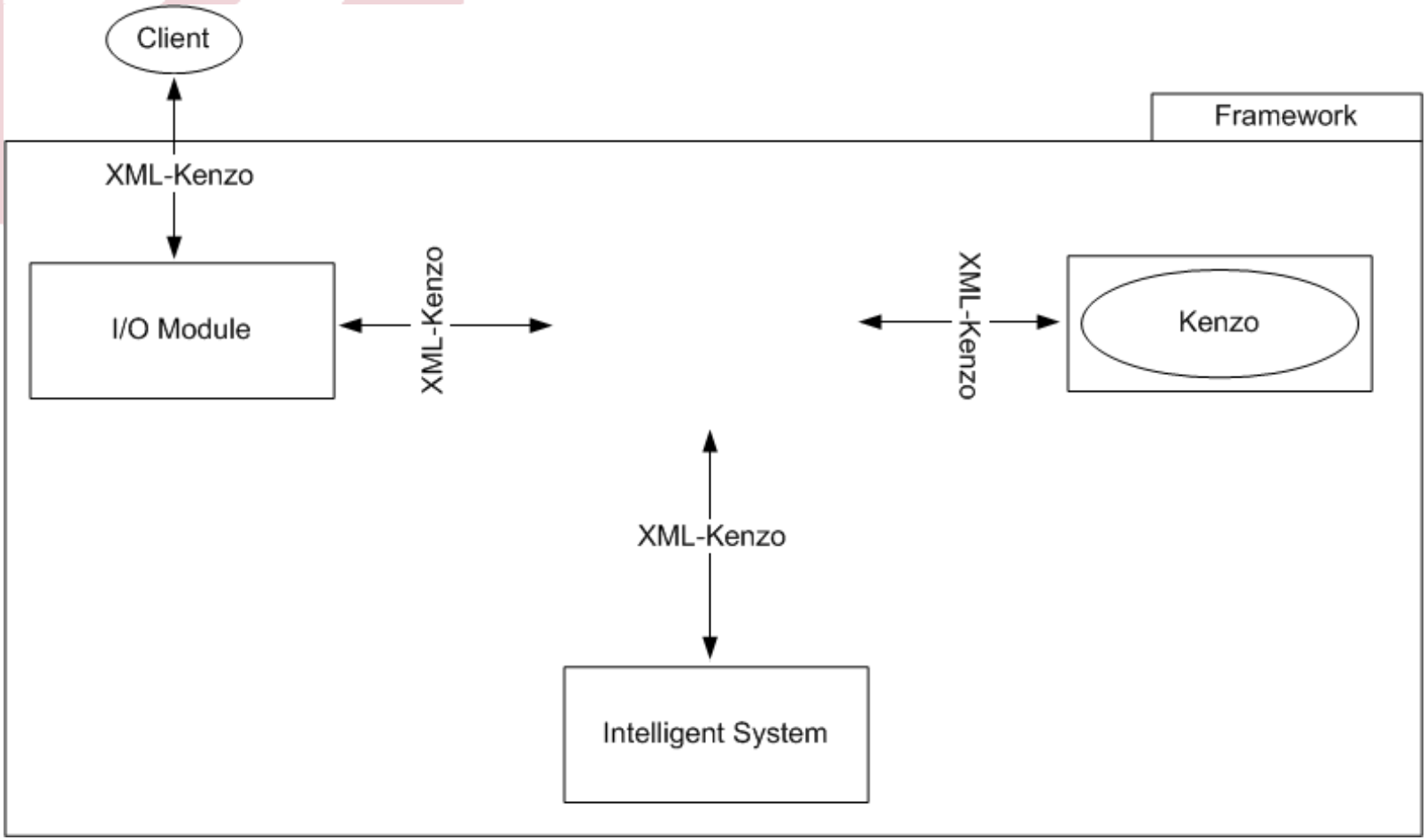
Shared repository pattern:

- Add the nuance that a component has no knowledge of:
  - what components have produced the data it uses,
  - what components will use its outputs

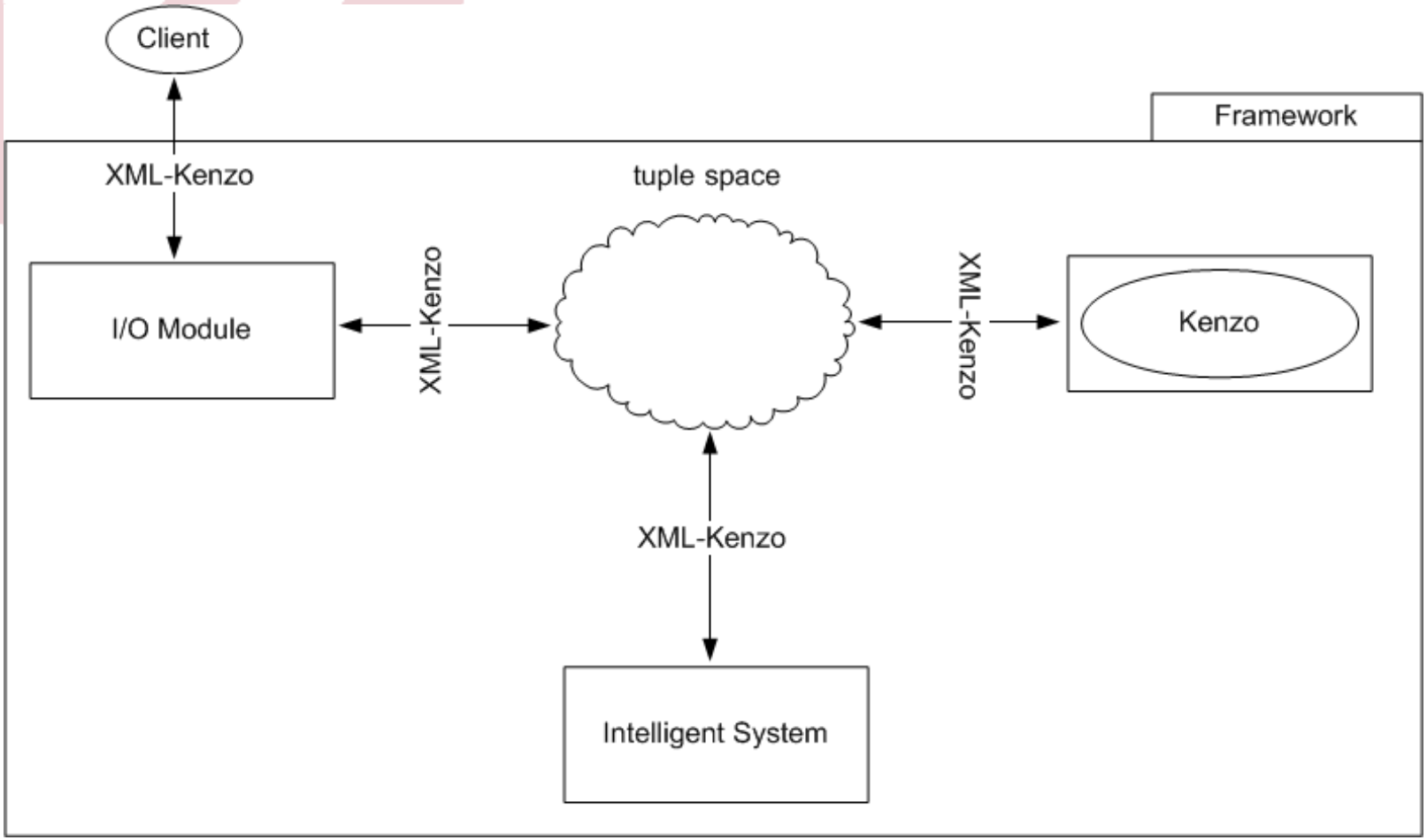
Architecture



Architecture



Architecture



- Recalculations solved
- No longer message passing style

About our Linda Model implementation

Desirable properties of any implementation of the Linda Model

About our Linda Model implementation

Desirable properties of any implementation of the Linda Model

- Shared

About our Linda Model implementation

Desirable properties of any implementation of the Linda Model

- Shared
- Persistent

About our Linda Model implementation

Desirable properties of any implementation of the Linda Model

- Shared
- Persistent
- ACID compliant

About our Linda Model implementation

Desirable properties of any implementation of the Linda Model

- Shared
- Persistent
- ACID compliant

Two possibilities

About our Linda Model implementation

Desirable properties of any implementation of the Linda Model

- Shared
- Persistent
- ACID compliant

Two possibilities

- Implement the Linda Model from scratch.

About our Linda Model implementation

Desirable properties of any implementation of the Linda Model

- Shared
- Persistent
- ACID compliant

Two possibilities

- Implement the Linda Model from scratch.
- Use an XML enabled database

About our Linda Model implementation

Desirable properties of any implementation of the Linda Model

- Shared
- Persistent
- ACID compliant

Two possibilities

- Implement the Linda Model from scratch.
- Use an XML enabled database
  - An XML enabled database is built on top of relational or object oriented databases.

About our Linda Model implementation

Desirable properties of any implementation of the Linda Model

- Shared
- Persistent
- ACID compliant

Two possibilities

- Implement the Linda Model from scratch.
- Use an XML enabled database
  - An XML enabled database is built on top of relational or object oriented databases.
  - Include two kind of processes:
    XML data $\longrightarrow$ database elements
    database elements $\longrightarrow$ XML data

Our Linda Model implementation

Implementation of a Linda Model

In our implementation

Our Linda Model implementation

Implementation of a Linda Model

tuple space

In our implementation

AllegroCache

Our Linda Model implementation

Implementation of a Linda Model	In our implementation
tuple space	AllegroCache
tuples	persistent objects of AllegroCache

Our Linda Model implementation

Implementation of a Linda Model	In our implementation
tuple space	AllegroCache
tuples	persistent objects of AllegroCache
operations	based on AllegroCache functions

Our Linda Model implementation

Implementation of a Linda Model	In our implementation
tuple space	AllegroCache
tuples	persistent objects of AllegroCache
operations tuples:	based on AllegroCache functions

Our Linda Model implementation

Implementation of a Linda Model	In our implementation
tuple space	AllegroCache
tuples	persistent objects of AllegroCache
operations	based on AllegroCache functions

tuples:

- tuples rely on XML-Kenzo but tuple space is based on AllegroCache, so it is necessary to convert from XML to objects and viceversa

Our Linda Model implementation

Implementation of a Linda Model	In our implementation
tuple space	AllegroCache
tuples	persistent objects of AllegroCache
operations	based on AllegroCache functions

tuples:

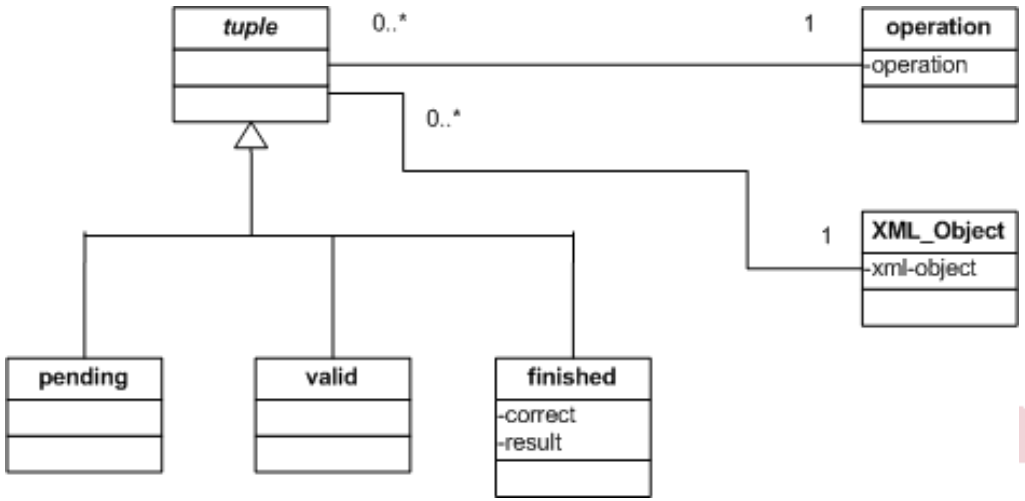
- tuples rely on XML-Kenzo but tuple space is based on AllegroCache, so it is necessary to convert from XML to objects and viceversa
- A tuple is implemented as an object belonging to any of the subclasses of the `tuple` class

Our Linda Model implementation

Implementation of a Linda Model	In our implementation
tuple space	AllegroCache
tuples	persistent objects of AllegroCache
operations	based on AllegroCache functions

tuples:

- tuples rely on XML-Kenzo but tuple space is based on AllegroCache, so it is necessary to convert from XML to objects and viceversa
- A tuple is implemented as an object belonging to any of the subclasses of the `tuple` class

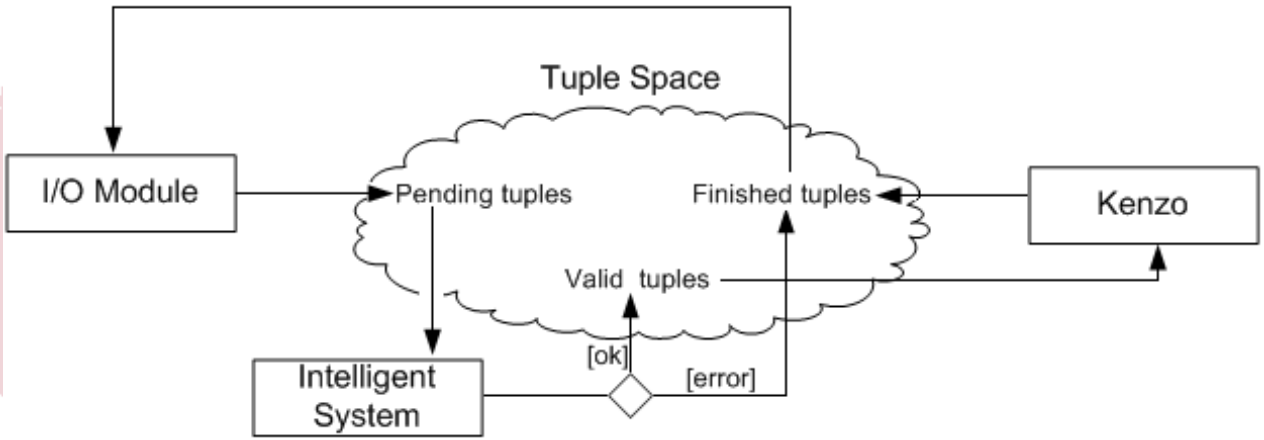


Implementation of the operations

`writetuple:`	`tuple` $\longrightarrow$ *Boolean*
`taketuple:`	`tuple` $\longrightarrow$ `tuple`
`readtuple:`	`tuple` $\longrightarrow$ `tuple-persistent`
`taketuplep:`	`tuple` $\longrightarrow$ `tuple` $\vee$ `nil`
`readtuplep:`	`tuple` $\longrightarrow$ `tuple-persistent` $\vee$ `nil`

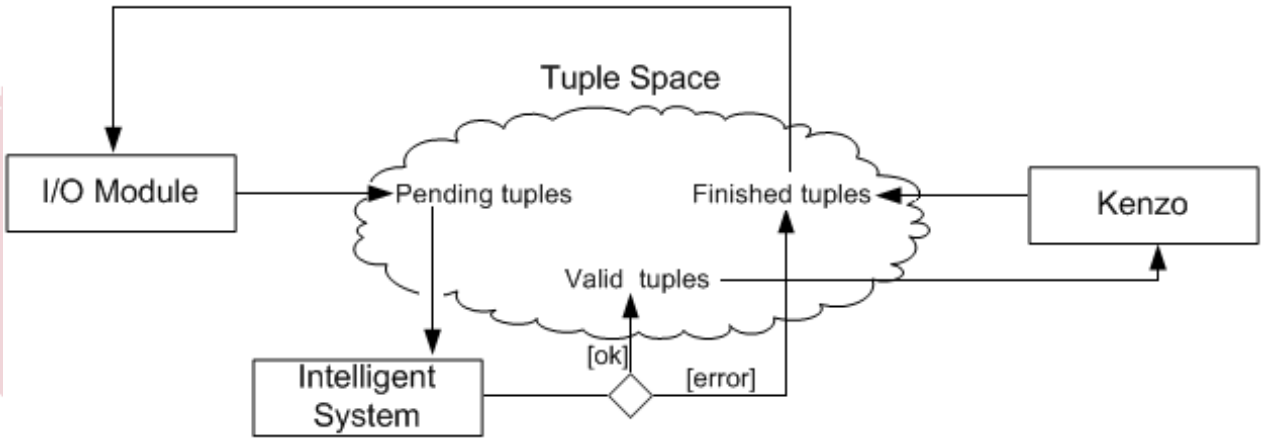
Implemented using the `select`, `insert` and `delete-instance` of AllegroCache

Relations between the tuple space and the modules



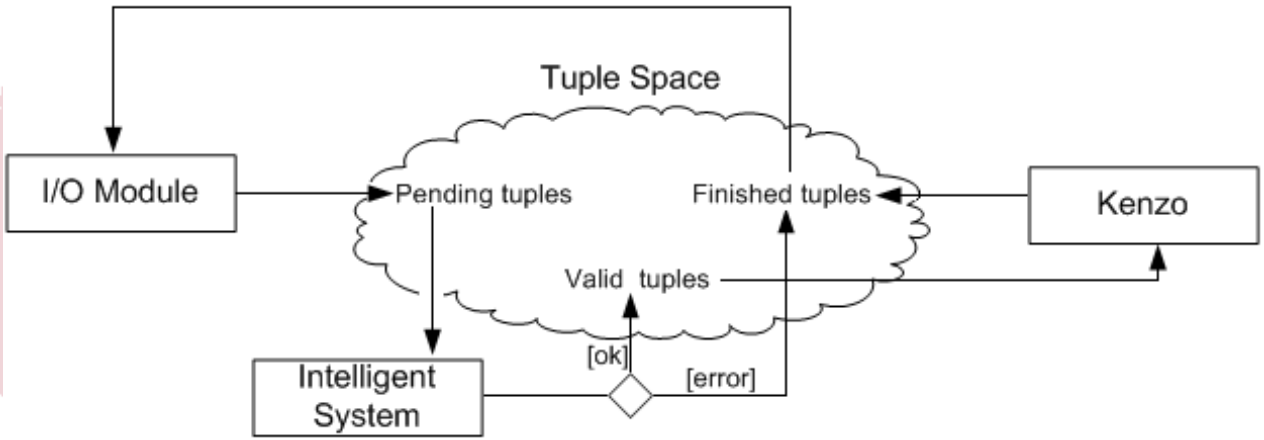
- I/O module
  - writes `pending` tuples
  - reads `finished` tuples

Relations between the tuple space and the modules



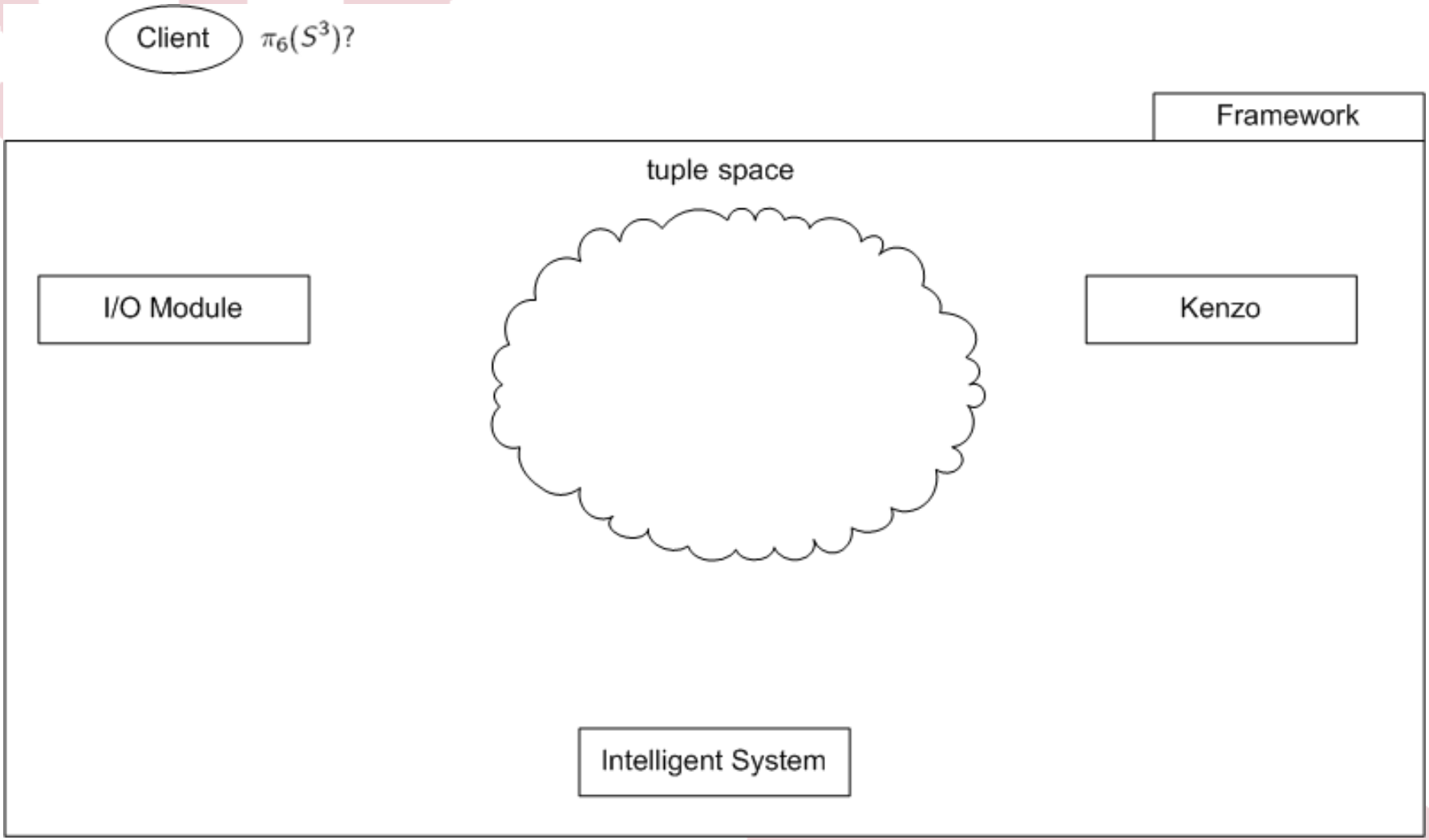
- I/O module
  - writes `pending` tuples
  - reads `finished` tuples
- Intelligent system
  - writes `valid` tuples
  - writes `finished` non valid tuples
  - takes `pending` tuples

Relations between the tuple space and the modules

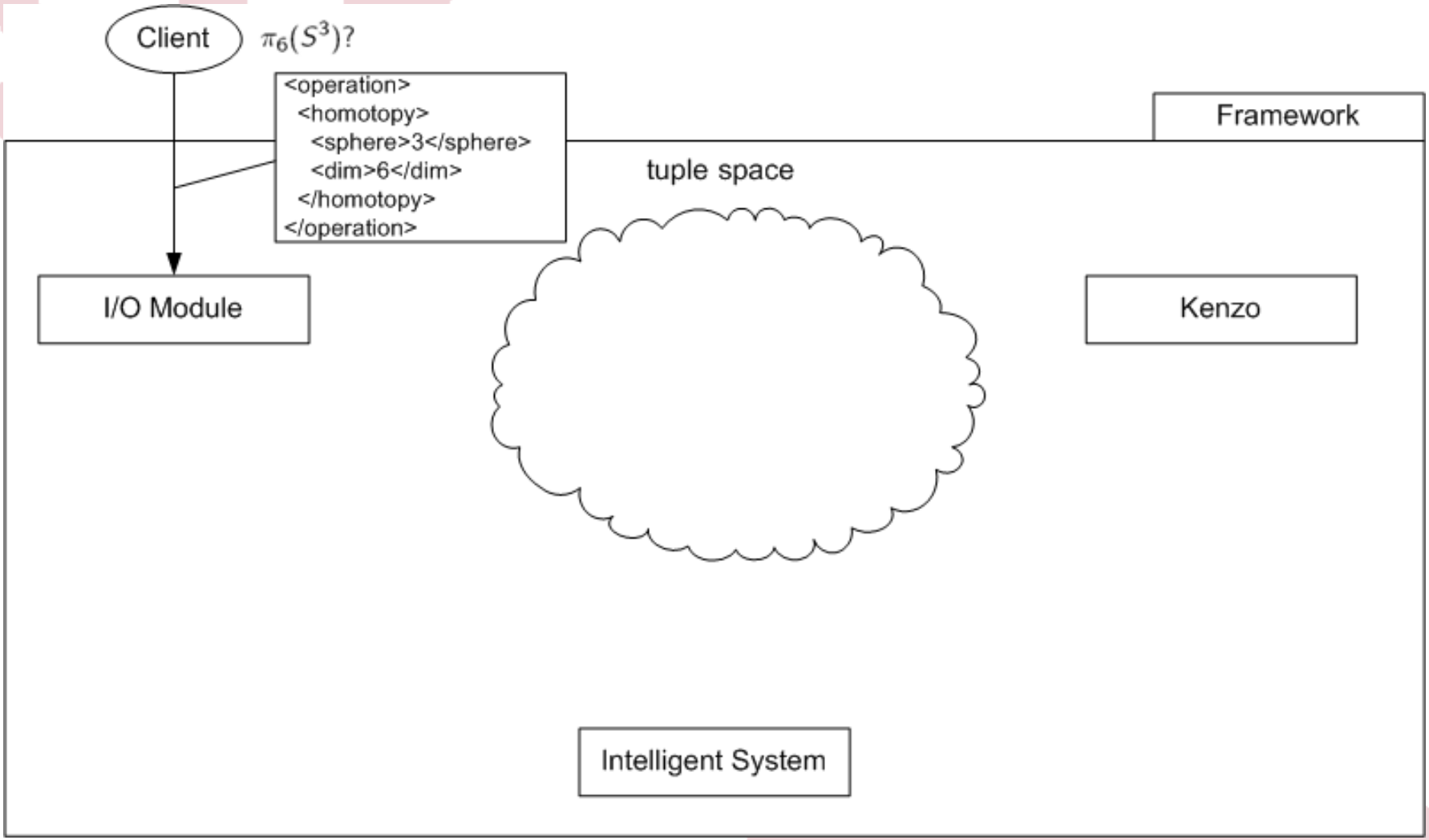


- I/O module
  - writes `pending` tuples
  - reads `finished` tuples
- Intelligent system
  - writes `valid` tuples
  - writes `finished` non valid tuples
  - takes `pending` tuples
- Kenzo module
  - writes `finished` tuples
  - takes `valid` tuples

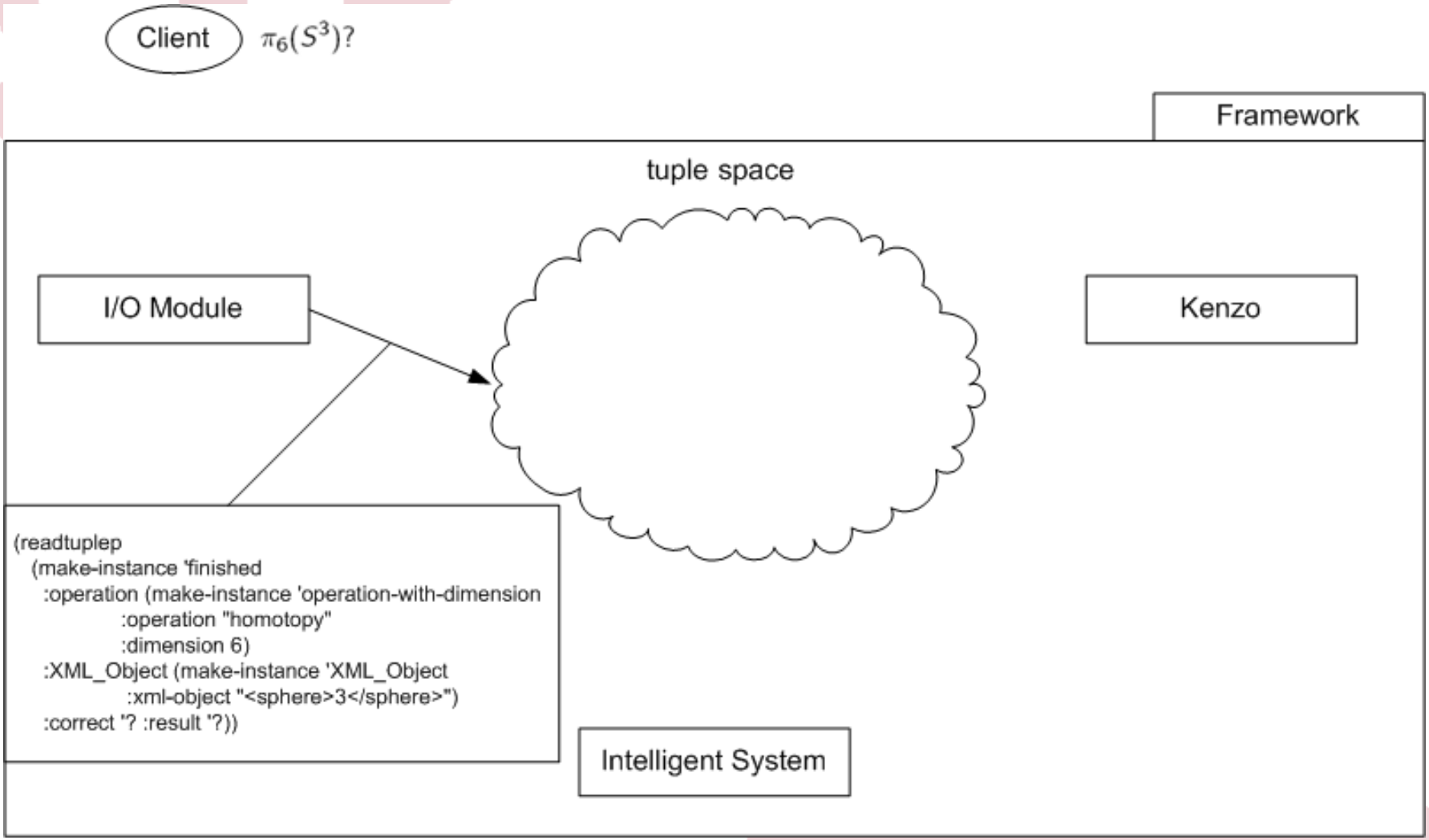
Computing $\pi_6(S^3)$



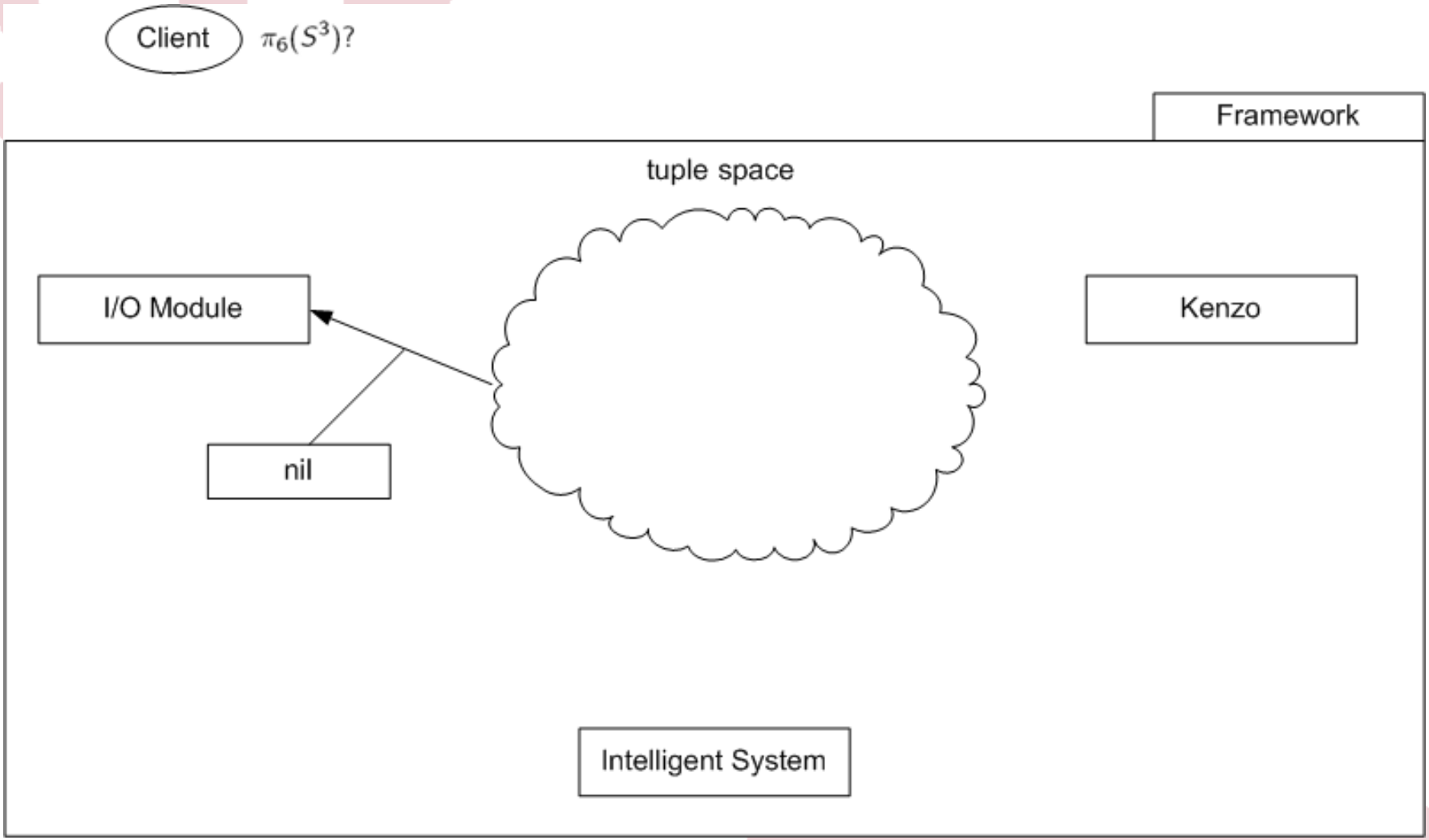
Computing $\pi_6(S^3)$



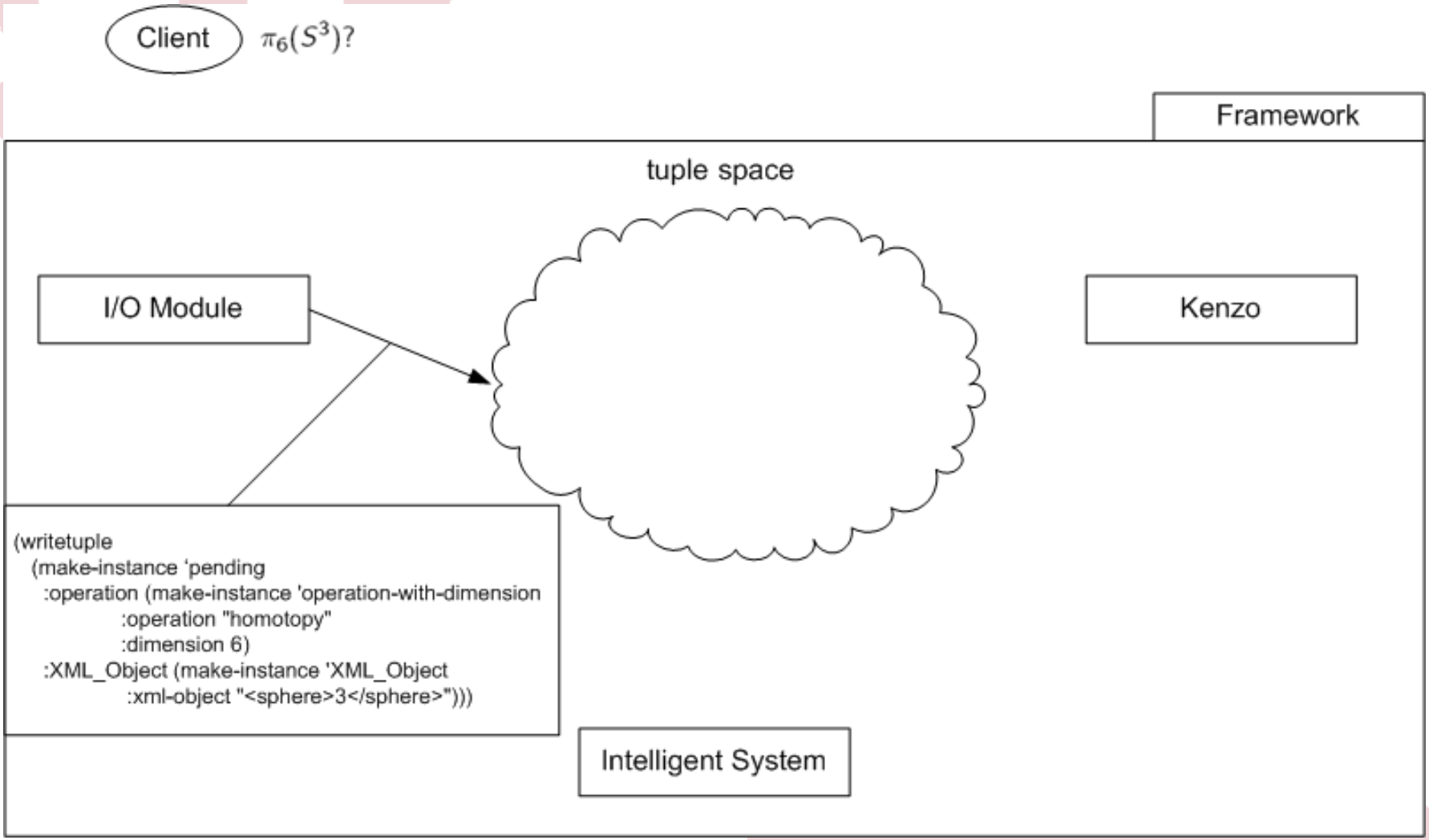
Computing $\pi_6(S^3)$



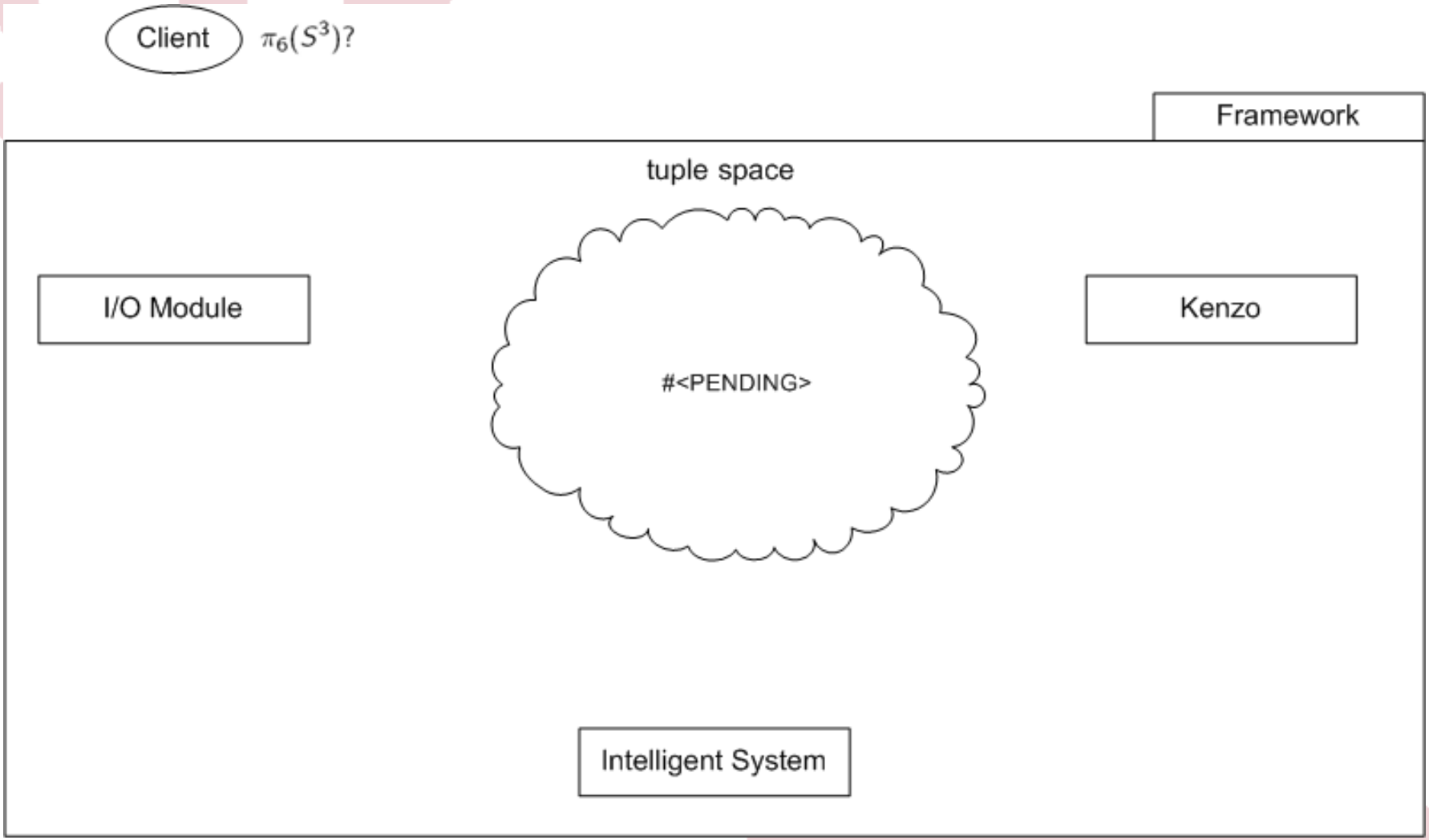
Computing $\pi_6(S^3)$



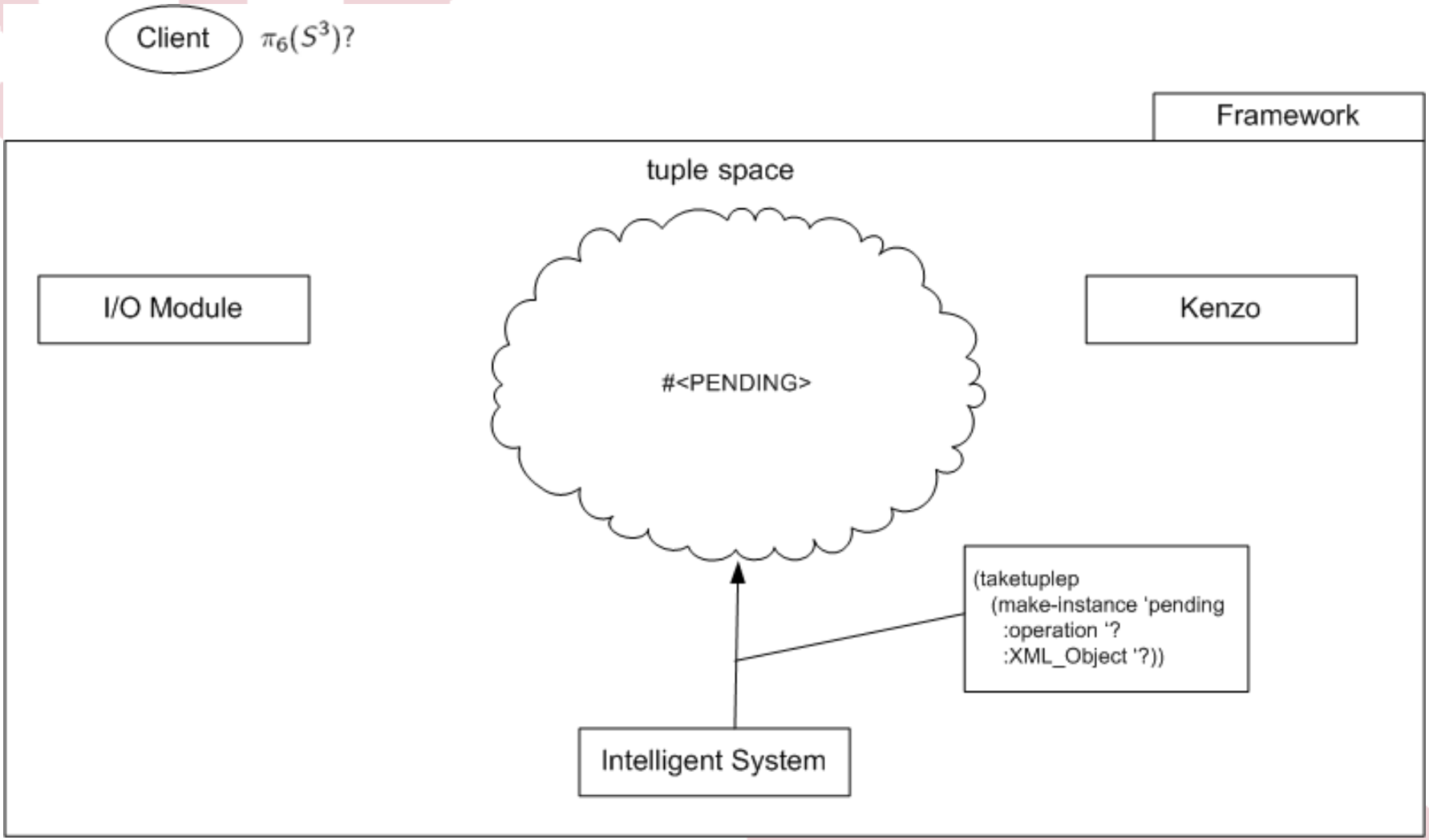
Computing $\pi_6(S^3)$



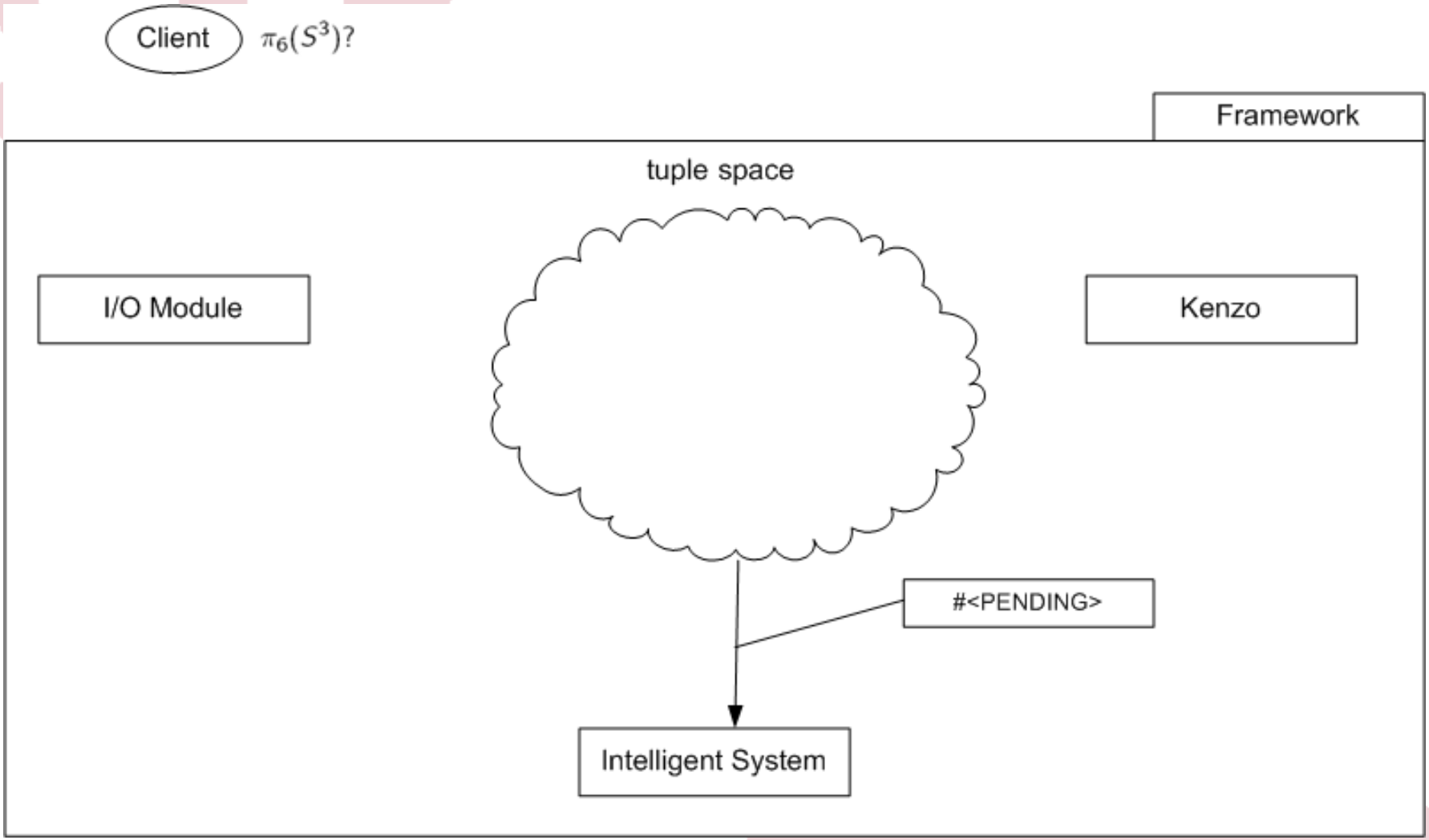
Computing $\pi_6(S^3)$



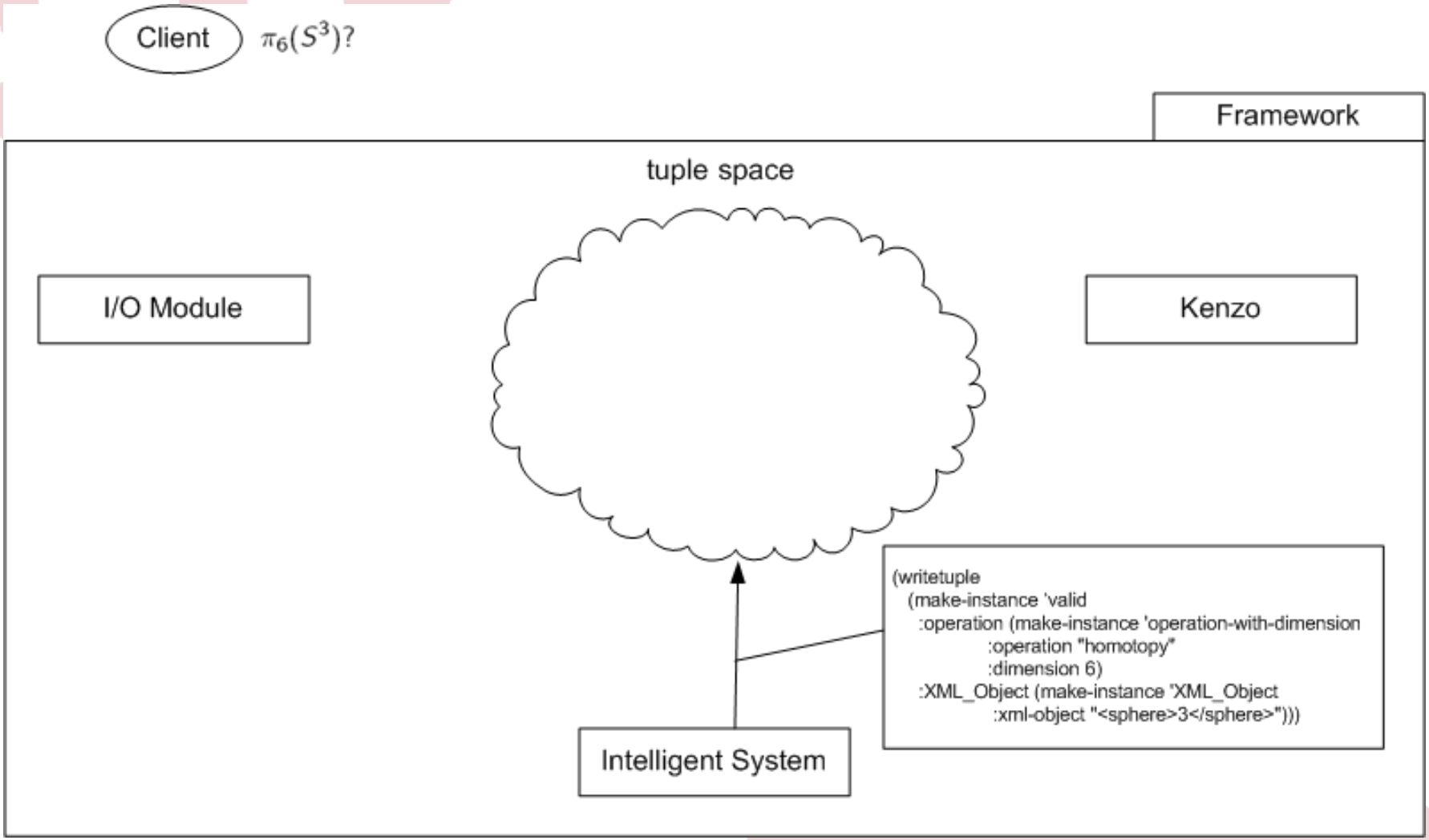
Computing $\pi_6(S^3)$



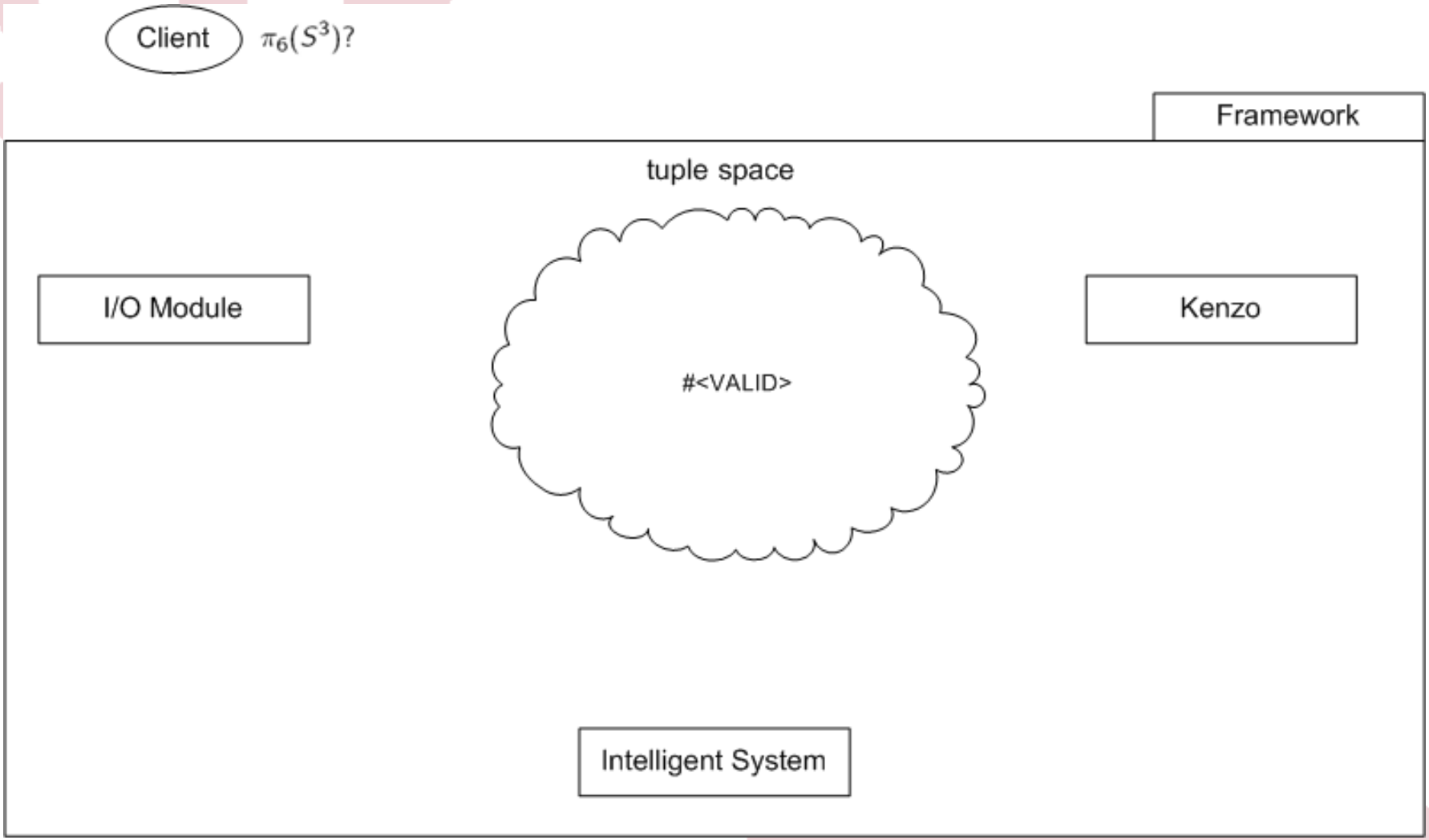
Computing $\pi_6(S^3)$



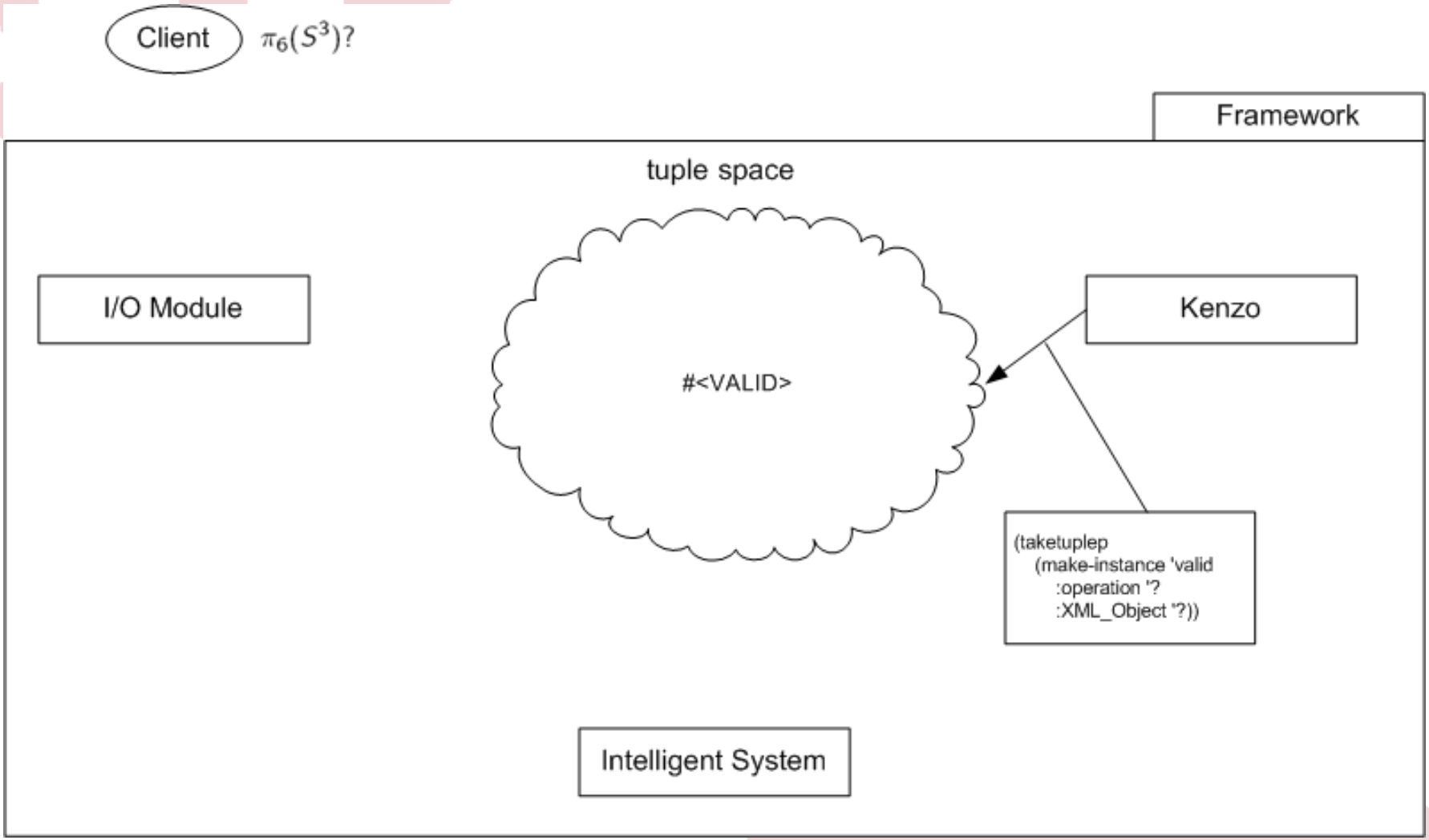
Computing $\pi_6(S^3)$



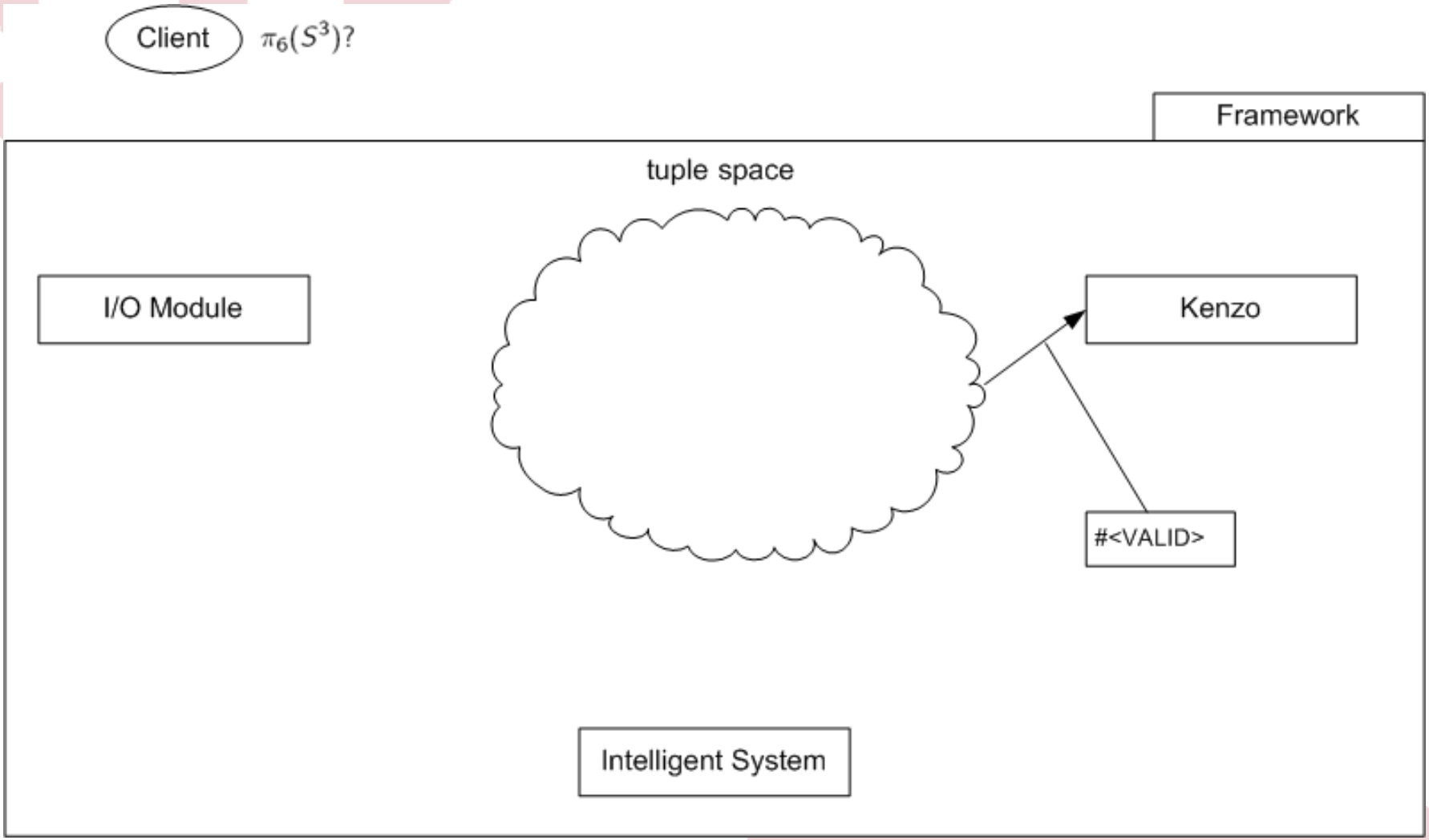
Computing $\pi_6(S^3)$



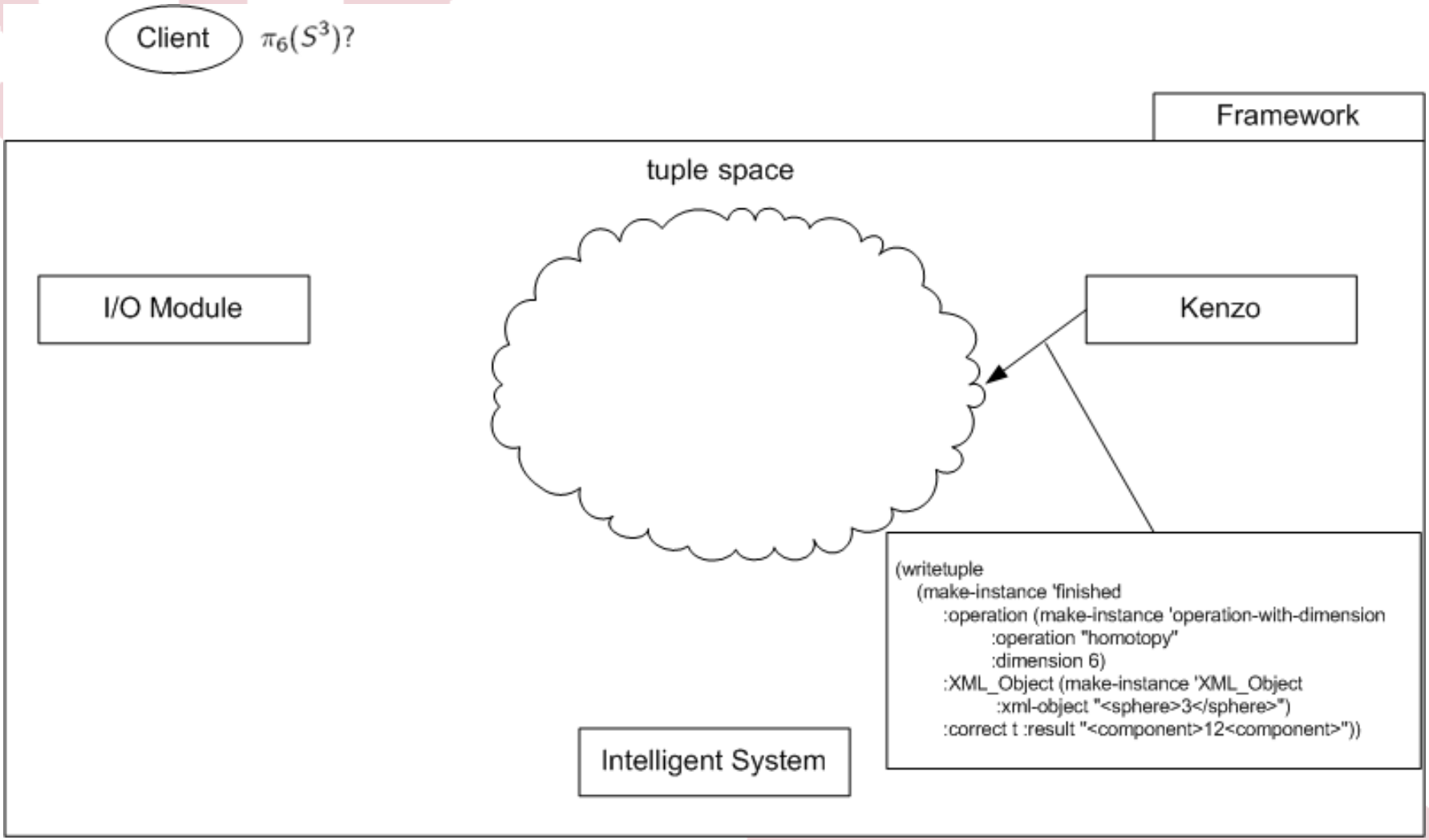
Computing $\pi_6(S^3)$



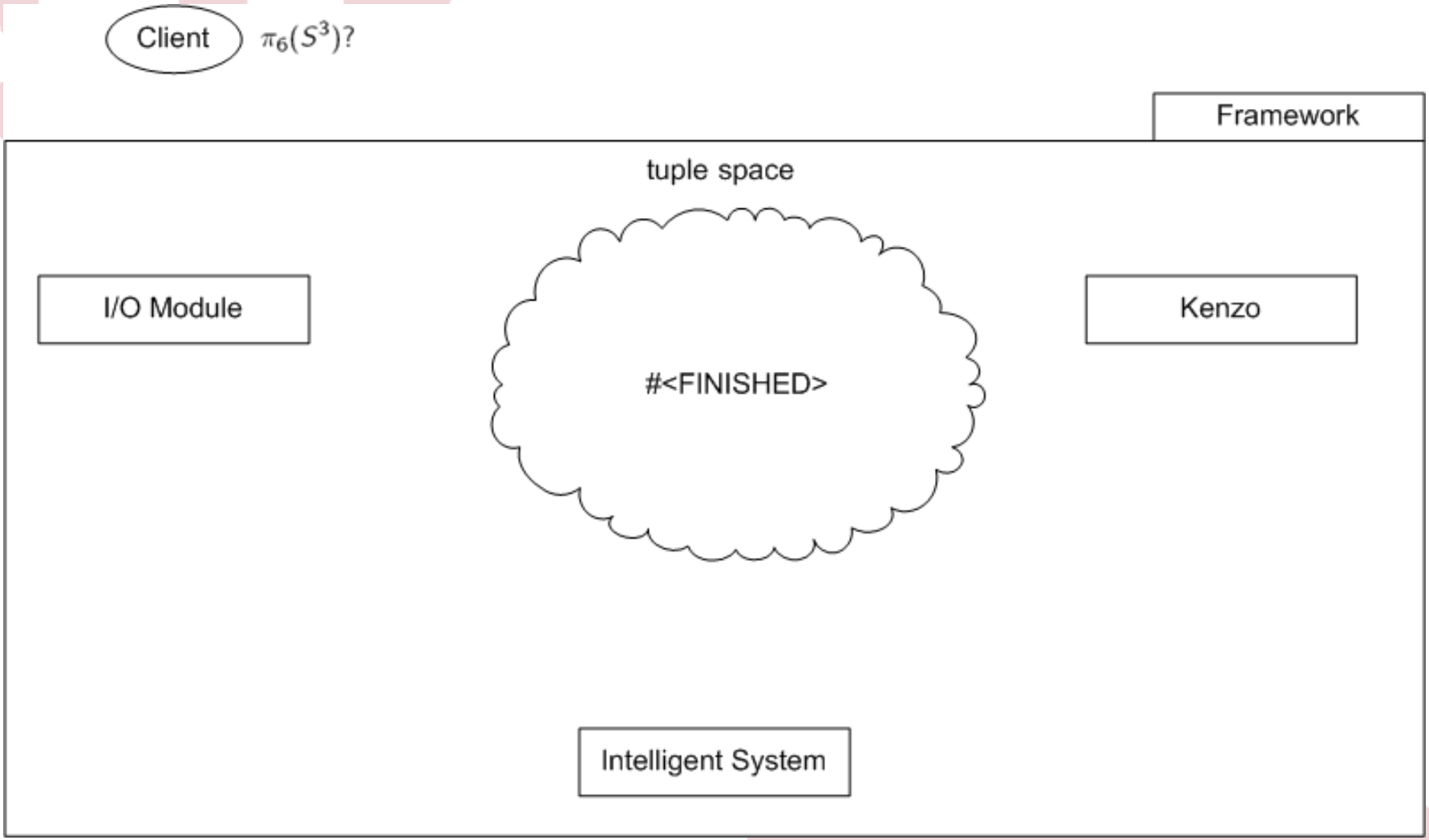
Computing $\pi_6(S^3)$



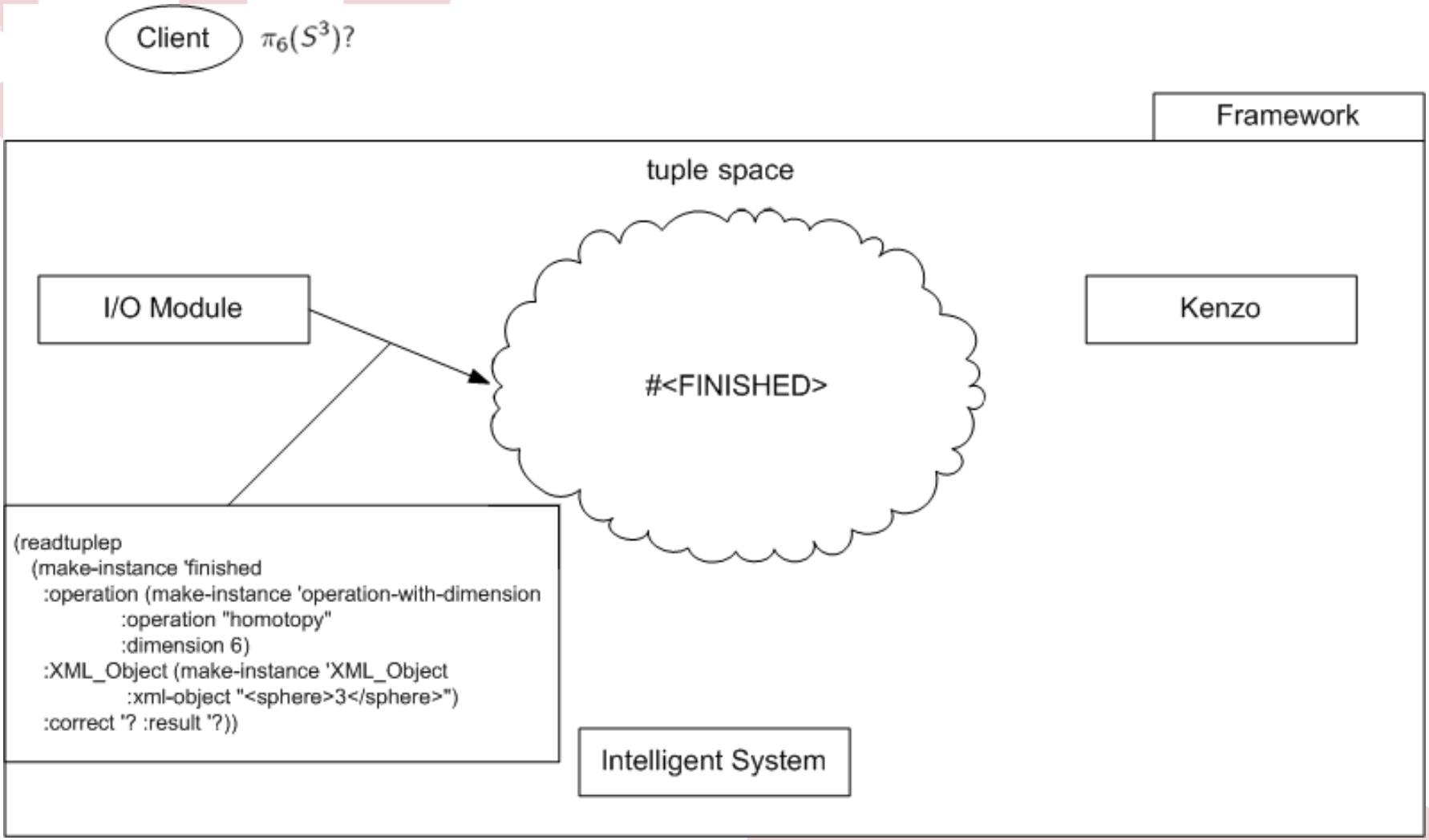
Computing $\pi_6(S^3)$



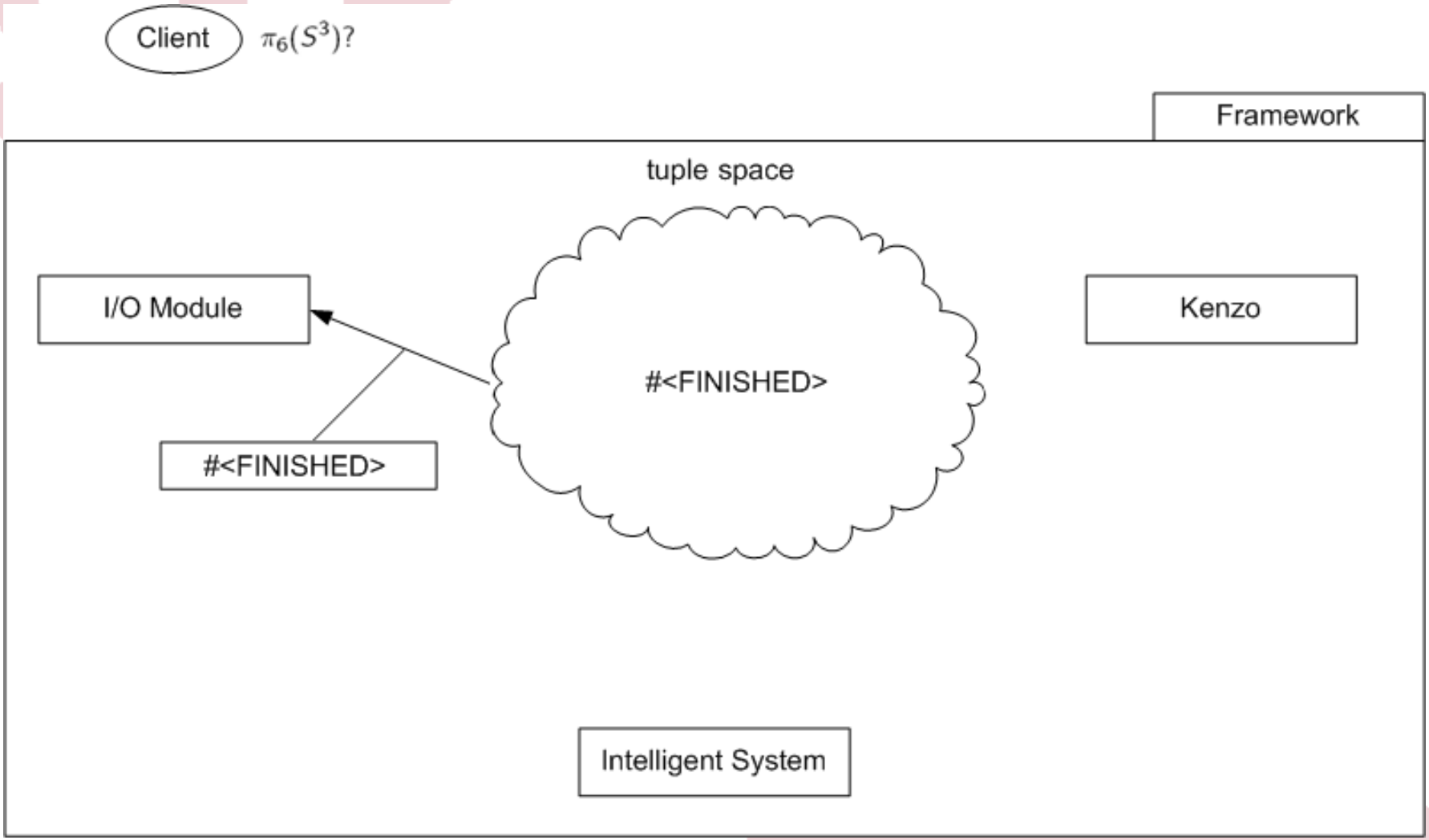
Computing $\pi_6(S^3)$



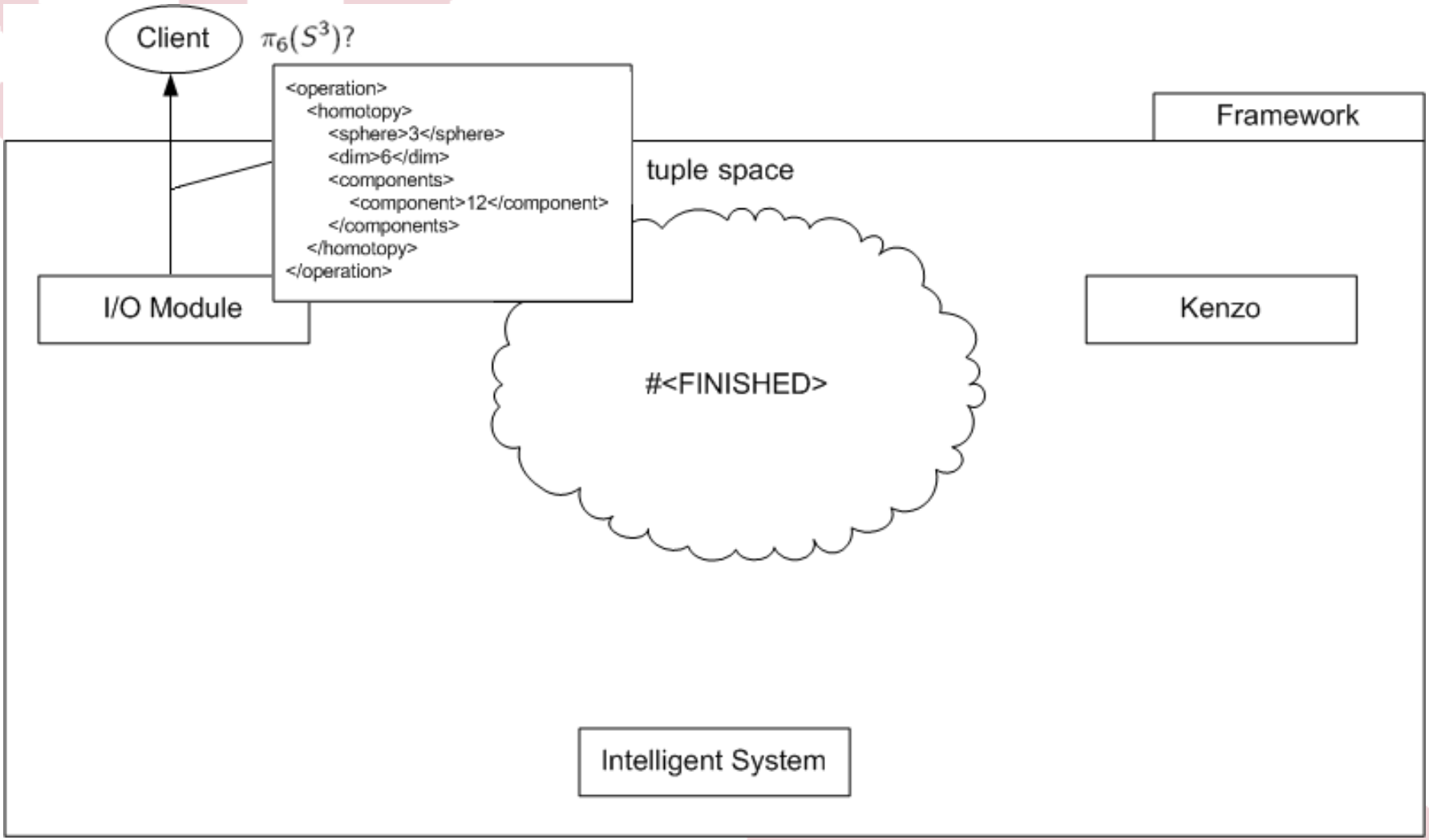
Computing $\pi_6(S^3)$



Computing $\pi_6(S^3)$



Computing $\pi_6(S^3)$



Computing $\pi_6(S^3)$

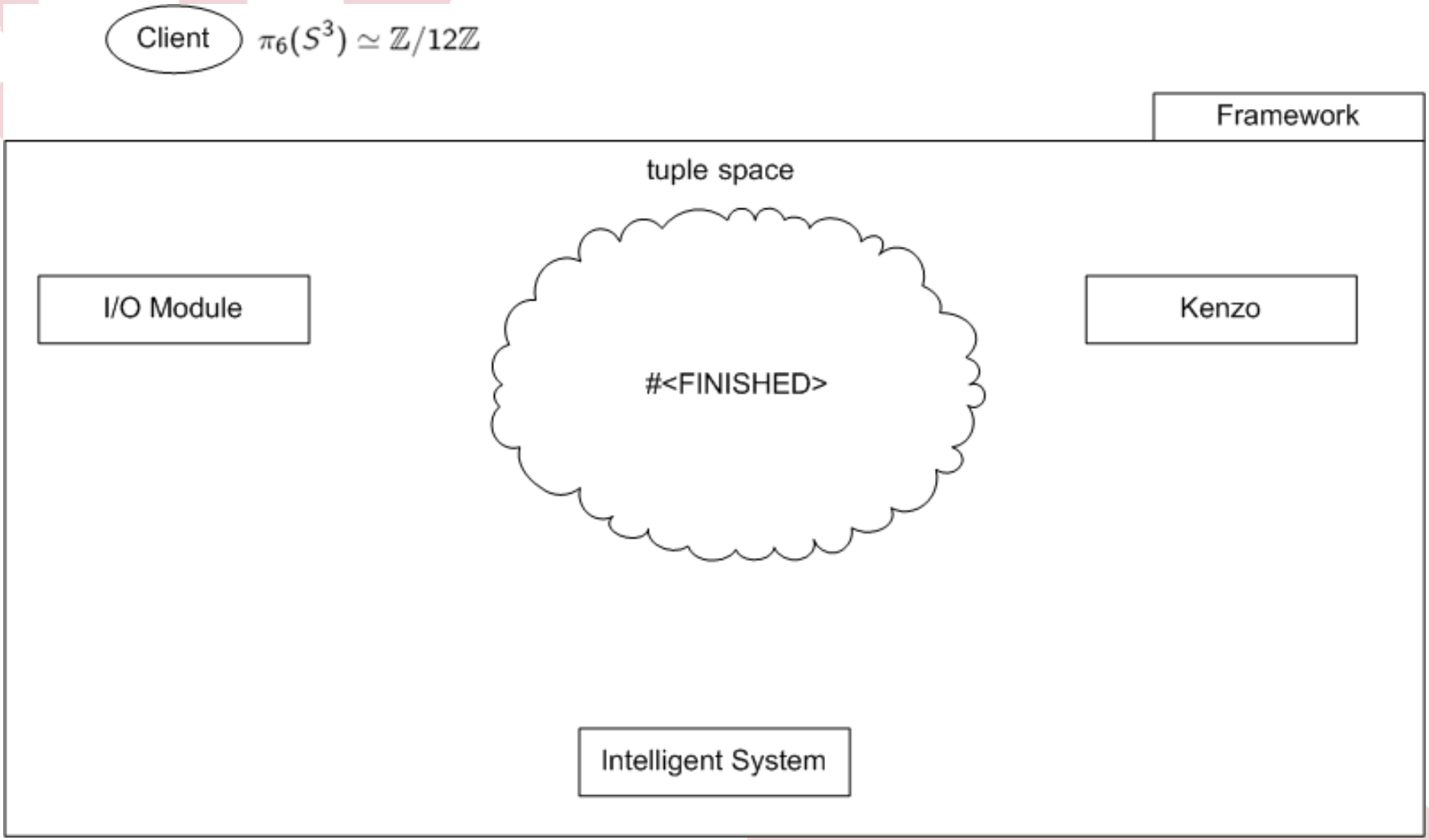


Table of Contents

1 Introduction

2 Architecture of the system

3 Communication among modules
- Communication
- Concurrency issues

4 Clients of our system

5 Conclusions and Further Work

Communication among modules

- By means of a reactive mechanism

Communication among modules

- By means of a reactive mechanism
- *publish-subscribe* machinery

Communication among modules

- By means of a reactive mechanism
- *publish-subscribe* machinery
  - subjects (can be modified)

Communication among modules

- By means of a reactive mechanism
- *publish-subscribe* machinery
  - subjects (can be modified)
  - observers (subscribed to any possible subjects modification)

Communication among modules

- By means of a reactive mechanism
- *publish-subscribe* machinery
  - subjects (can be modified)
  - observers (subscribed to any possible subjects modification)
- In our implementation:

Communication among modules

- By means of a reactive mechanism
- *publish-subscribe* machinery
  - subjects (can be modified)
  - observers (subscribed to any possible subjects modification)
- In our implementation:
  - tuple space $\longrightarrow$ subject
  - each module $\longrightarrow$ an observer

Communication among modules

- By means of a reactive mechanism
- *publish-subscribe* machinery
  - subjects (can be modified)
  - observers (subscribed to any possible subjects modification)
- In our implementation:
  - tuple space $\longrightarrow$ subject
  - each module $\longrightarrow$ an observer
  - tuple space $\longrightarrow$ has got associated an observer

Communication among modules

- By means of a reactive mechanism
- *publish-subscribe* machinery
  - subjects (can be modified)
  - observers (subscribed to any possible subjects modification)
- In our implementation:
  - tuple space $\longrightarrow$ subject
  - each module $\longrightarrow$ an observer
  - tuple space $\longrightarrow$ has got associated an observer
- Subscriptions are stored in an AllegroCache database

```
(defclass subscription ()
  ((host :initarg :host :accessor host )
   (port :initarg :port :accessor port )
   (type-tuples :initarg :type :accessor type :index any))
  (:metaclass persistent-class))
```

Communication among modules

Module observer creates a passive socket

```
(defun observer ()
  (loop
    (let ((sock
           (make-socket :connect :passive :local-port 8002)))
      (wait-for-input-available sock)
      (close sock) (validate))))
```

Communication among modules

Module observer creates a passive socket

```
(defun observer ()
  (loop
    (let ((sock
           (make-socket :connect :passive :local-port 8002)))
      (wait-for-input-available sock)
      (close sock) (validate))))
```

Tuple space observer

- Consults the database of subscriptions
- Sends a notification to the subscribers

```
(defun notify (type-tuple)
  (open-network-database "localhost" 8010)
  (let ((subscrites
         (retrieve-from-index 'subscription
                              'type-tuple type-tuple :all t)))
    (dolist (temp subscrites)
      (let ((socket (make-socket :remote-host (host temp)
                                 :remote-port (port temp))))
        (close socket))))
  (close-database))
```

Concurrency issues

Several requests could be in the system at the same time

Concurrency issues

Several requests could be in the system at the same time
Linda Model design avoids some of the concurrency problems

Concurrency issues

Several requests could be in the system at the same time
Linda Model design avoids some of the concurrency problems
Two situations can appear:

Concurrency issues

Several requests could be in the system at the same time
Linda Model design avoids some of the concurrency problems
Two situations can appear:

- Different modules working at the same time

Concurrency issues

Several requests could be in the system at the same time

Linda Model design avoids some of the concurrency problems

Two situations can appear:

- Different modules working at the same time
  - Absence of deadlocks
  - Mutual exclusion

Concurrency issues

Several requests could be in the system at the same time

Linda Model design avoids some of the concurrency problems

Two situations can appear:

- Different modules working at the same time
  - Absence of deadlocks
  - Mutual exclusion
- Different processes working in a same module

Concurrency issues

Several requests could be in the system at the same time

Linda Model design avoids some of the concurrency problems

Two situations can appear:

- Different modules working at the same time
  - Absence of deadlocks
  - Mutual exclusion
- Different processes working in a same module
  - No deadlocks
  - No race condition problems
  - No starvation

Table of Contents

1 Introduction

2 Architecture of the system

3 Communication among modules

4 Clients of our system

5 Conclusions and Further Work

A GUI and two web services

- A GUI
- Two web services

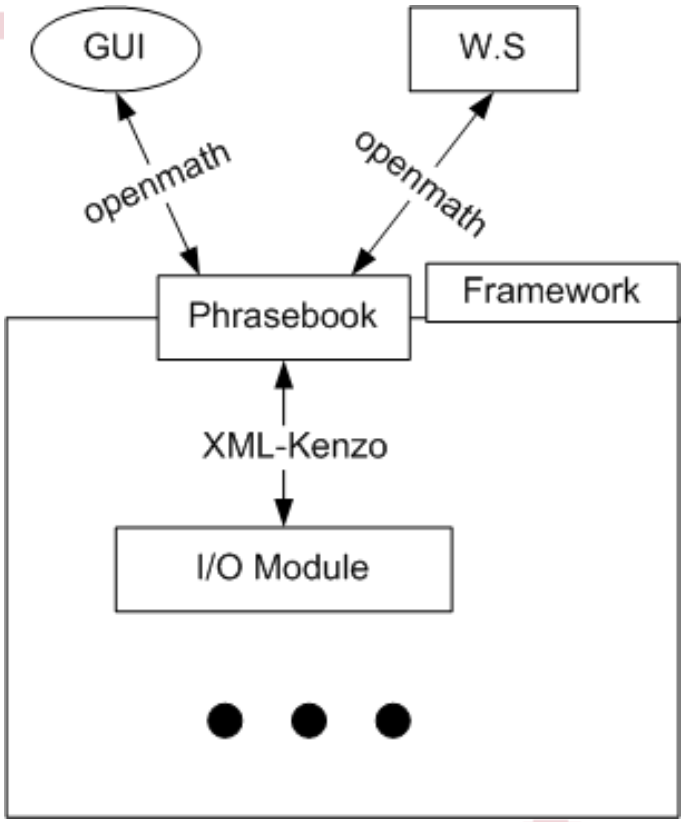


Table of Contents

1 Introduction

2 Architecture of the system

3 Communication among modules

4 Clients of our system

5 Conclusions and Further Work

Conclusions and Further Work

Conclusions

- We have reported on a system to interact with the Kenzo Computer Algebra system
- We have obtained a system which can process queries *concurrently* and where *persistent* results are available thanks to an adapted implementation in Common Lisp of the Linda Model

Conclusions and Further Work

Conclusions

- We have reported on a system to interact with the Kenzo Computer Algebra system
- We have obtained a system which can process queries *concurrently* and where *persistent* results are available thanks to an adapted implementation in Common Lisp of the Linda Model

Future Work

- Liberate our system from the proprietary features of Allegro Common Lisp
- Decide when a computation is easy enough to solve it locally or whether it must be sent to a central server
- Devise good heuristics to decide what is the meaning of the predicate "to be an easy computation"
- Integration with other systems

Applying Generative Communication to Symbolic Computation in Common Lisp

Jónathan Heras, Vico Pascual and Julio Rubio

Departamento de Matemáticas y Computación
Universidad de La Rioja
Spain

2nd European Lisp Symposium 2009
May 29, 2009