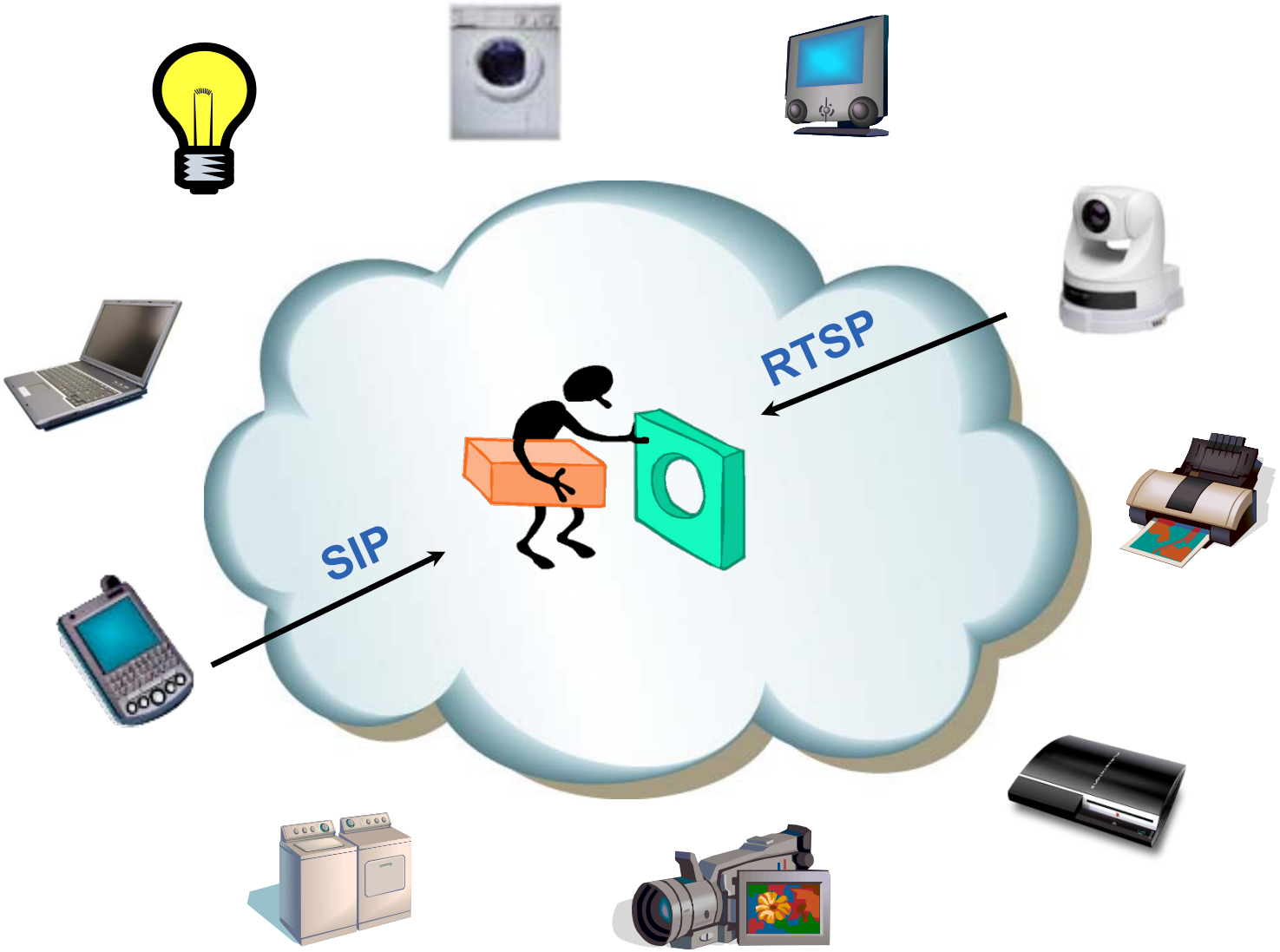
# Automatic Generation of Network protocol gateways

David Bromberg, Laurent Réveillère,
Julia Lawall and Gilles Muller

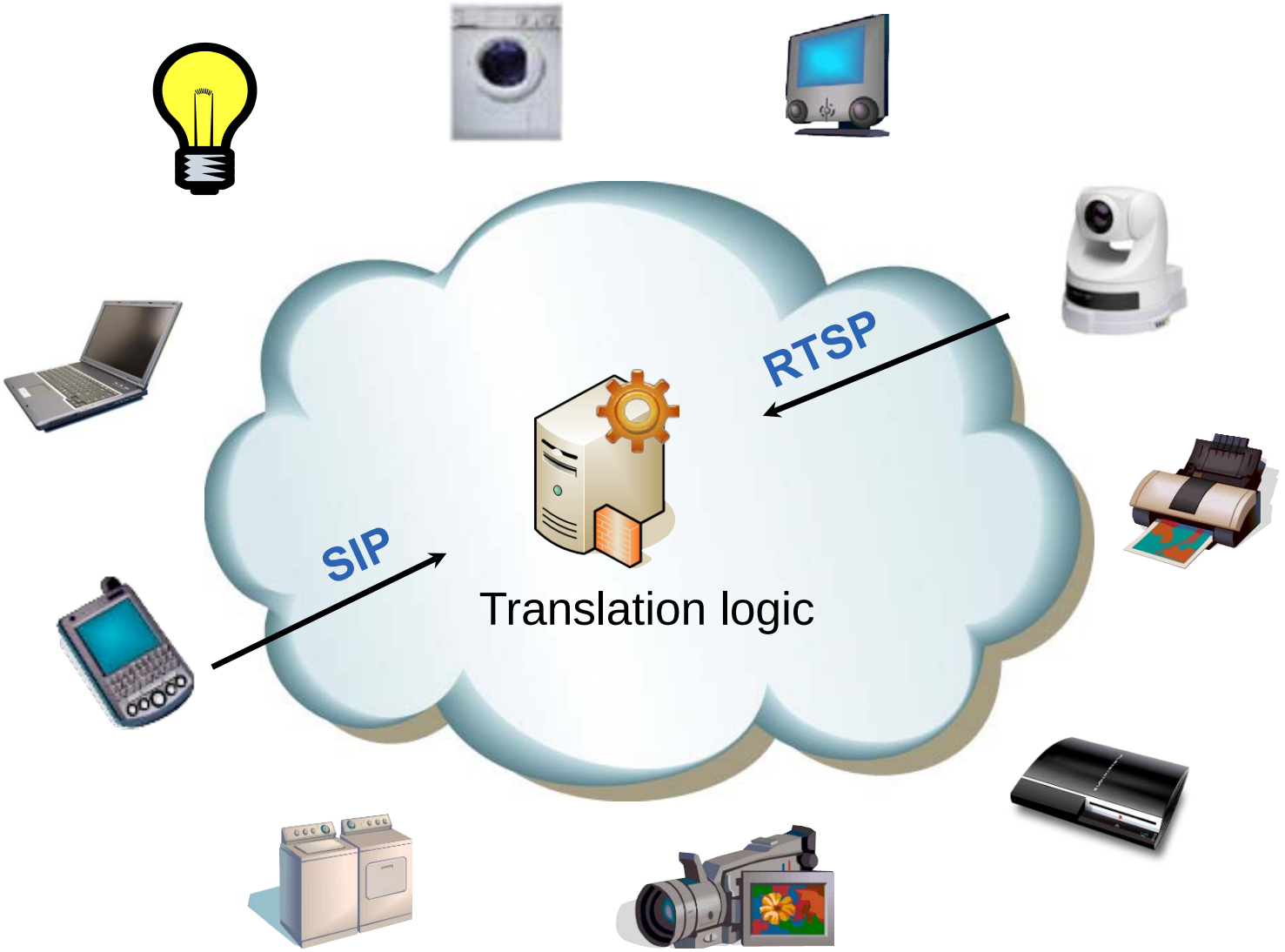
# Ubiquitous environment



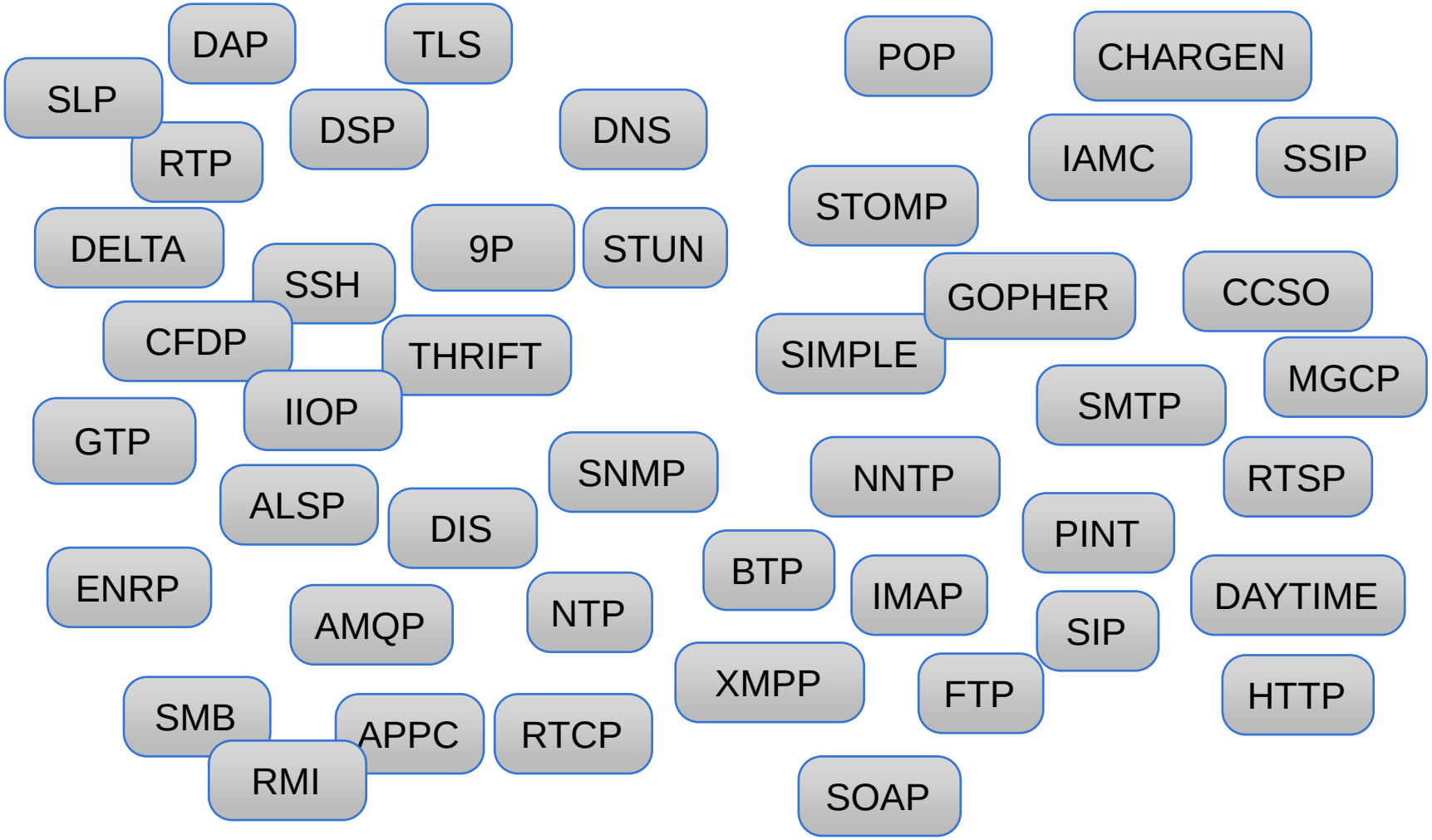
# Heterogeneous protocols



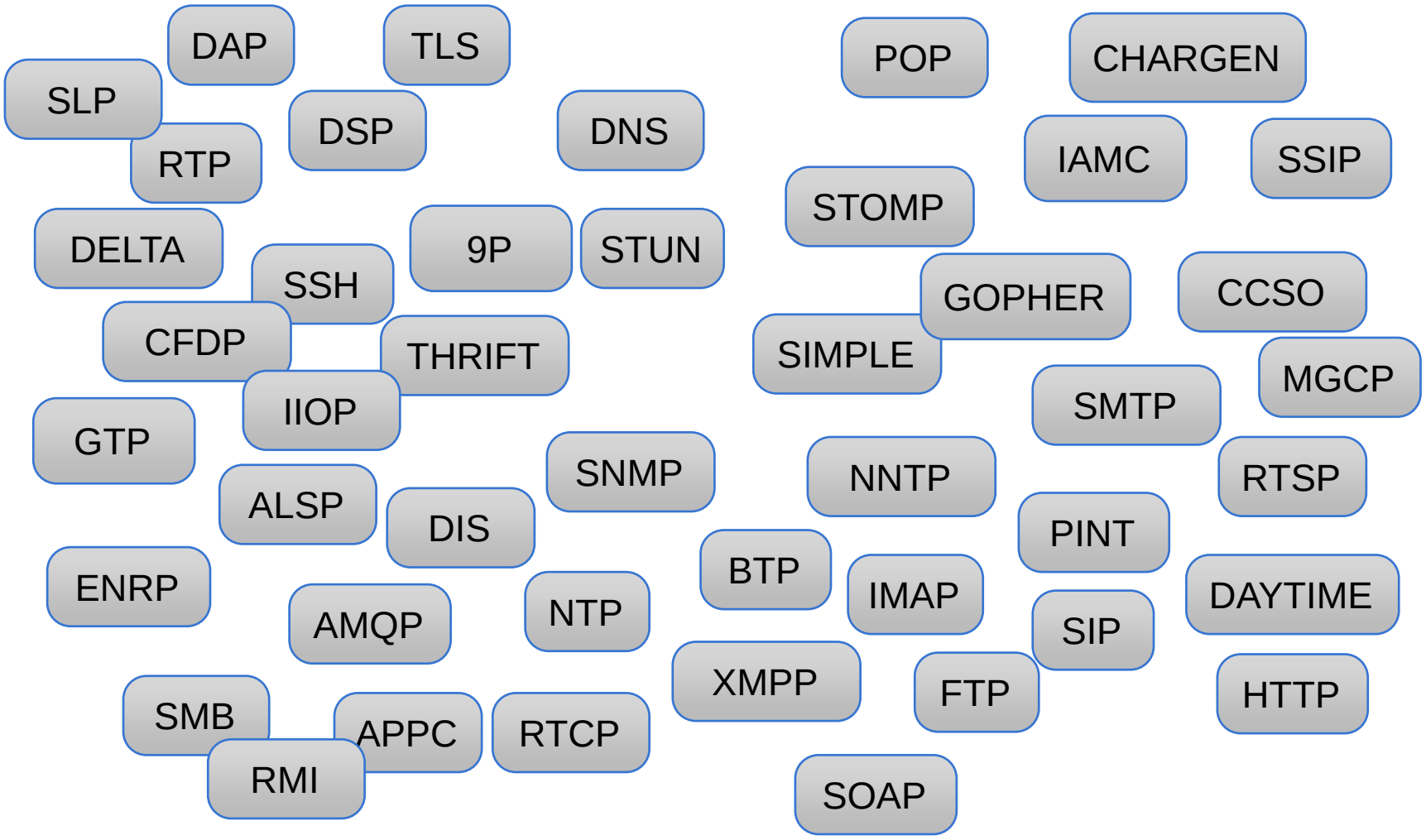
# Gateways for providing interoperability



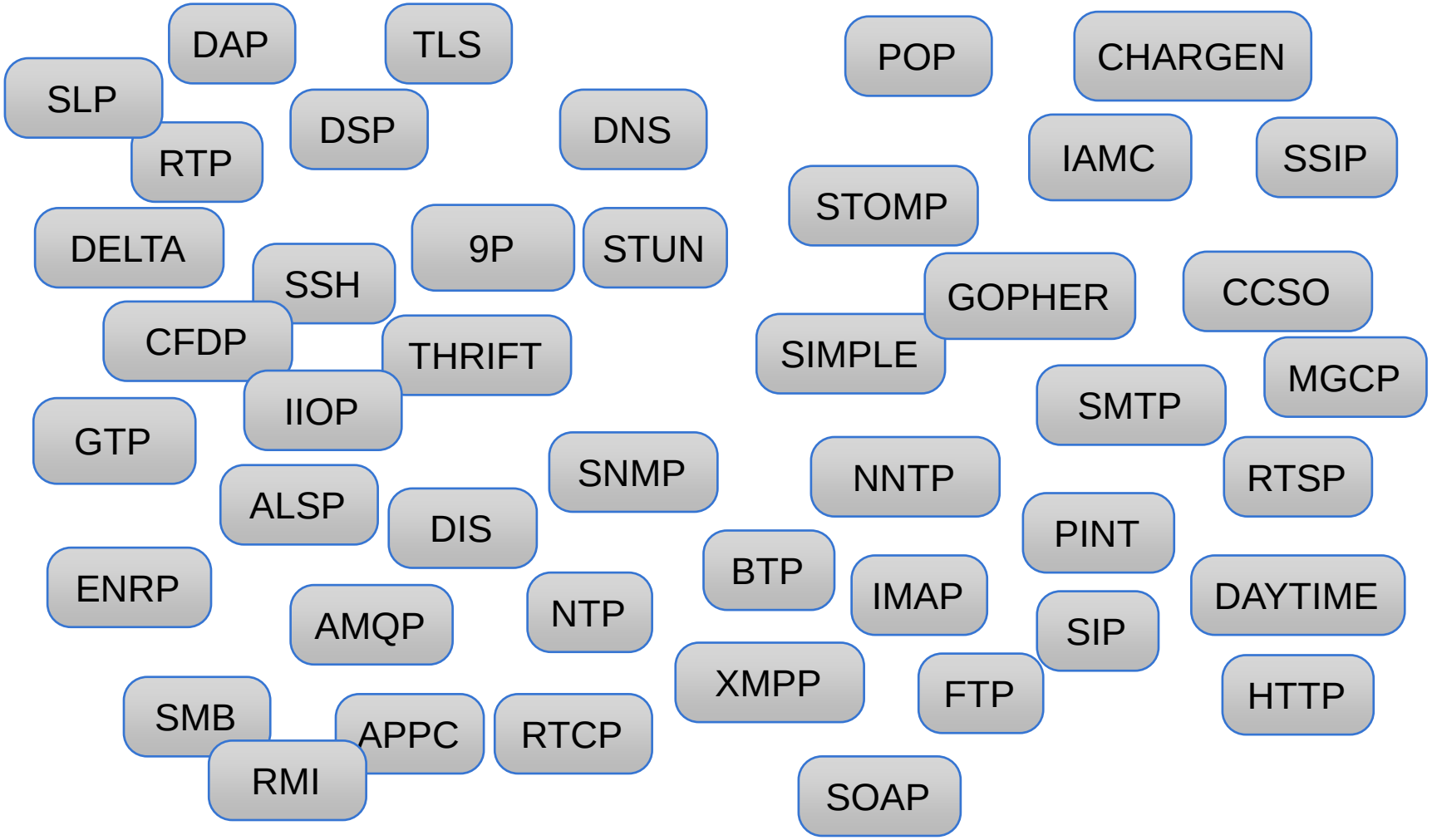
# Huge set of interaction protocols



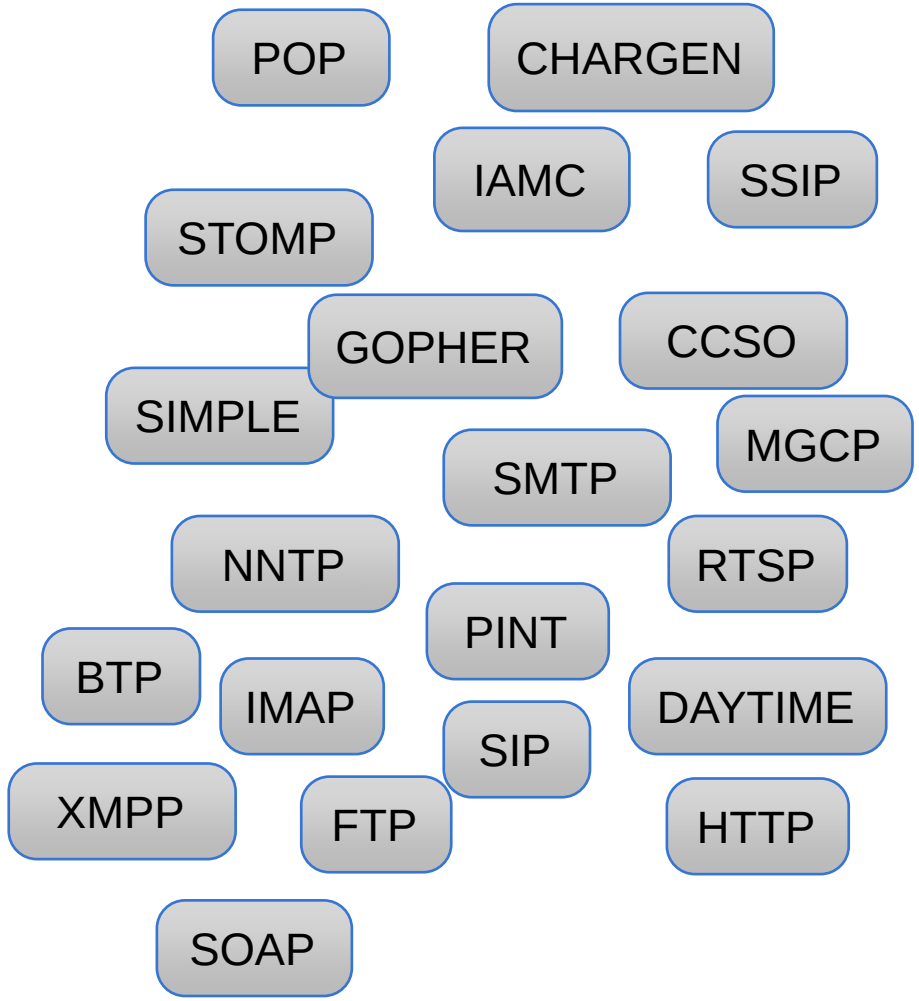
# Binary protocols



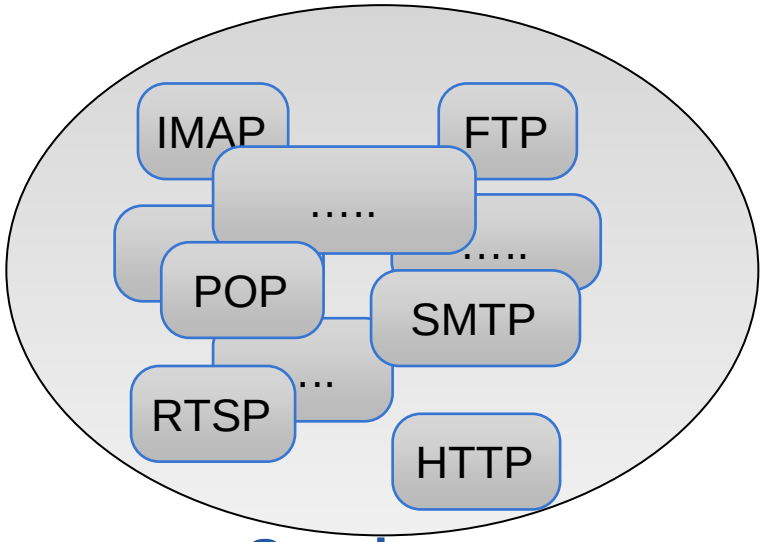
# Textual protocols



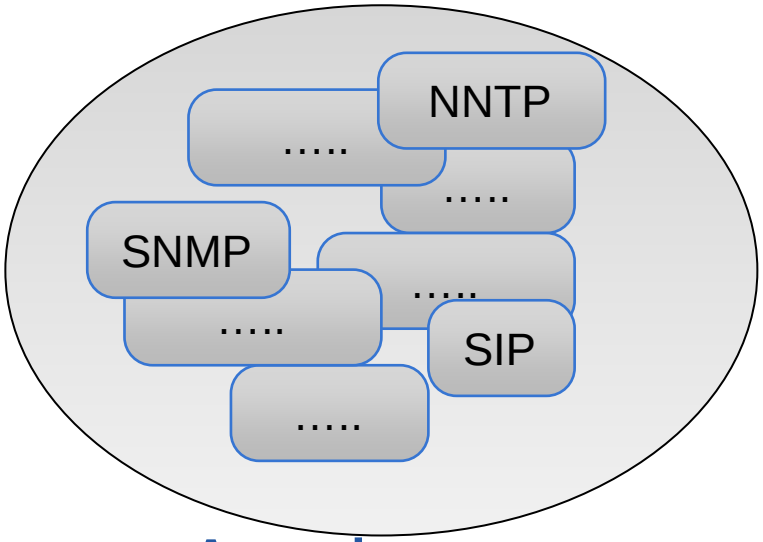
# XML protocols



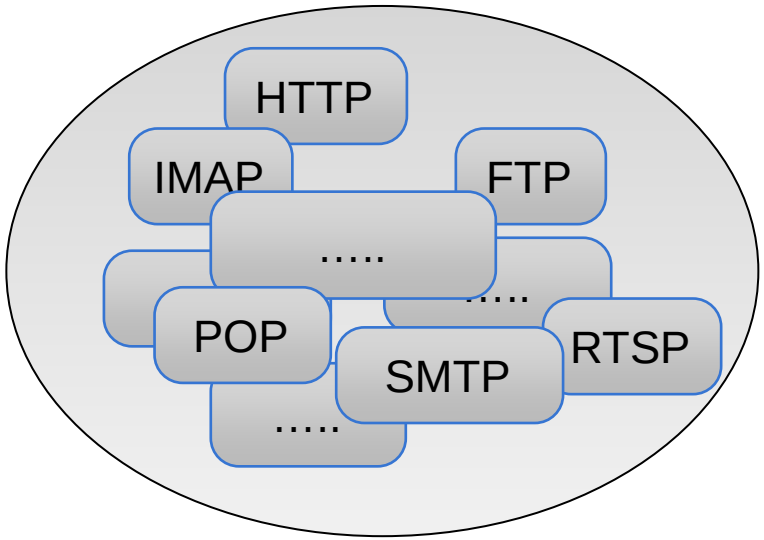
# Other protocol characteristics



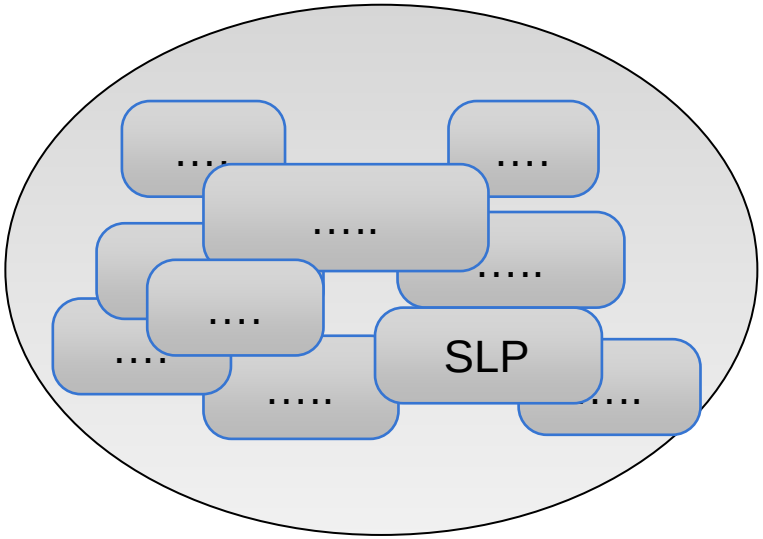

**Synchronous**




**Asynchronous**




**Unicast**



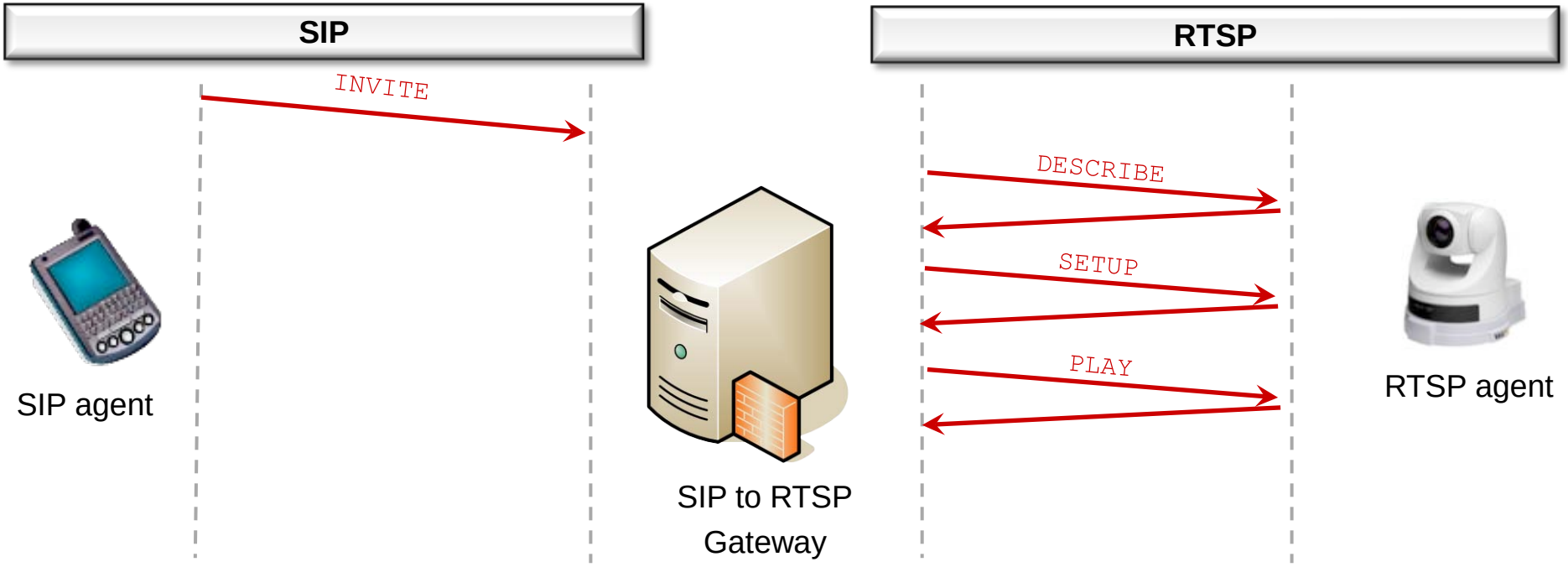

**Multicast**

# Developing a gateway

- Challenging task requiring
  - knowledge of the protocols involved
  - substantial understanding of low-level network programming
- A gateway must take into account
  - the different degrees of expressiveness of the different protocols
  - the range of communication methods used by the protocols
  - the need of the various protocols to maintain state across multiple messages

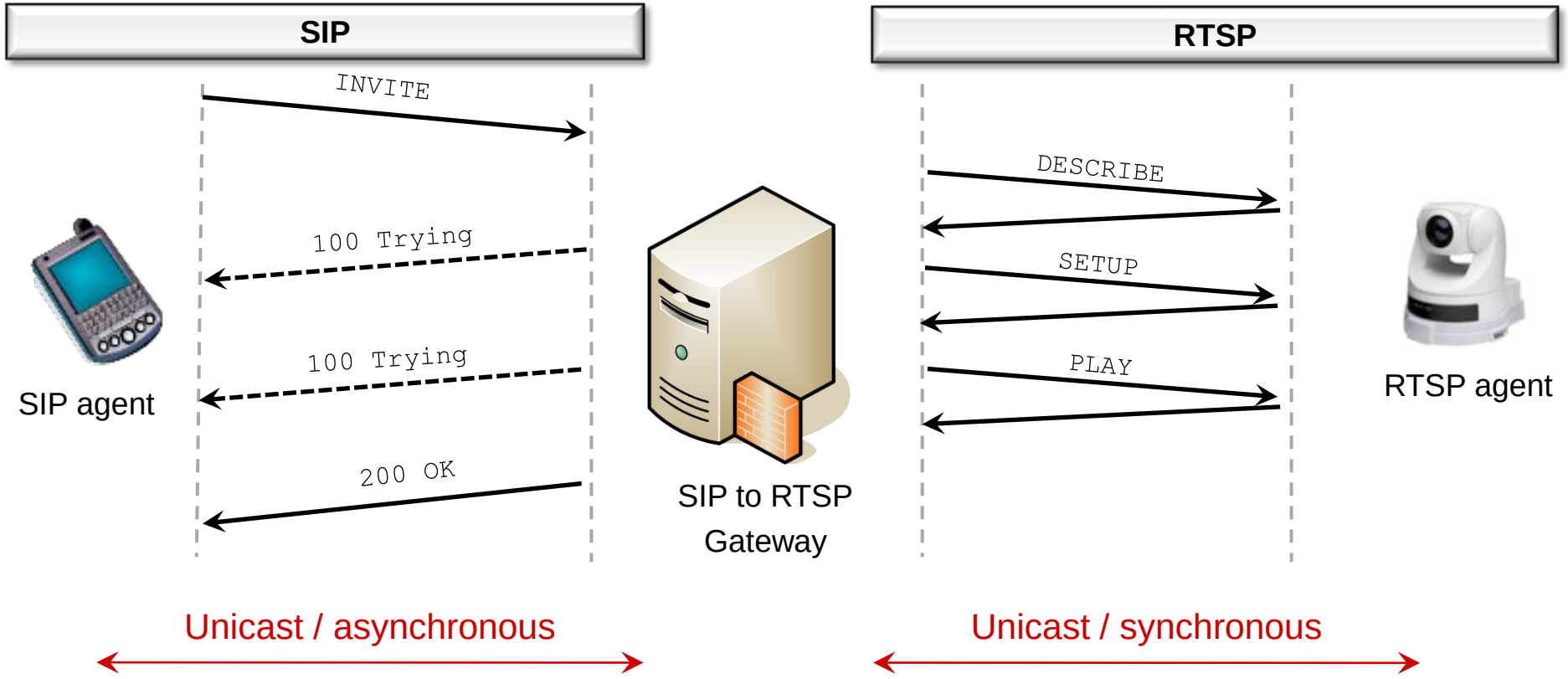
# Mismatched protocol expressiveness

- One single SIP request
  - multiple RTSP messages



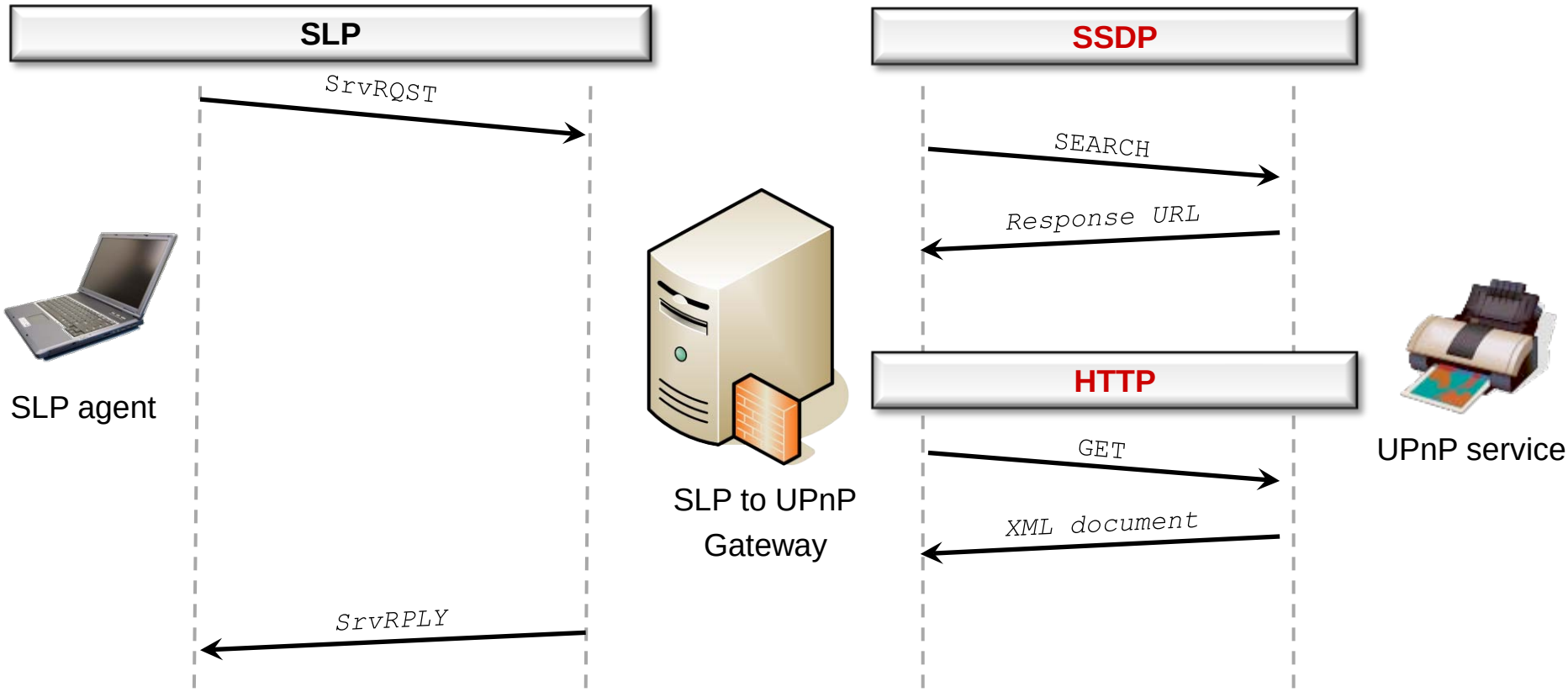
# Mismatched protocol expressiveness

- RTSP requests send synchronously using separate TCP connections
  - ➔SIP responses returned asynchronously over UDP



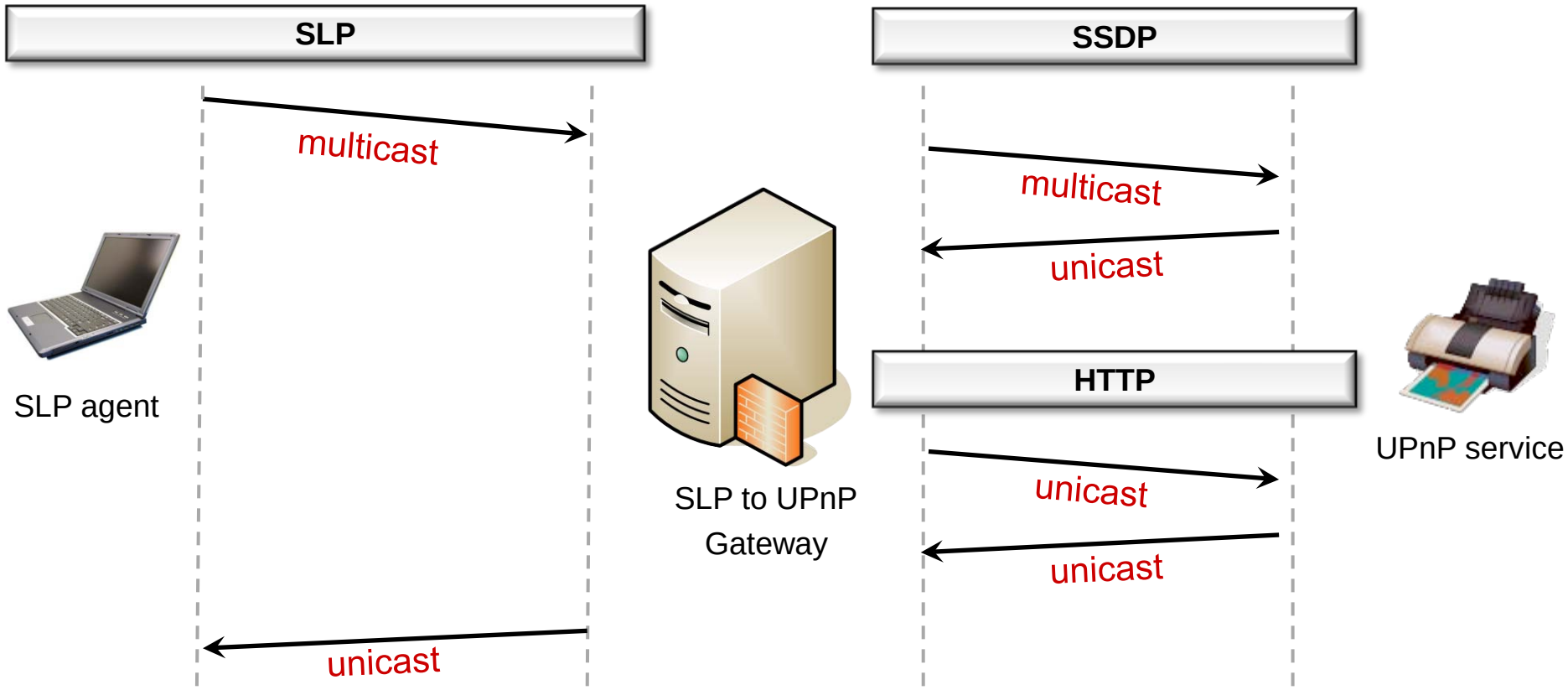
# Heterogeneous communication

- One protocol on the left
  - ➔several protocols on the right



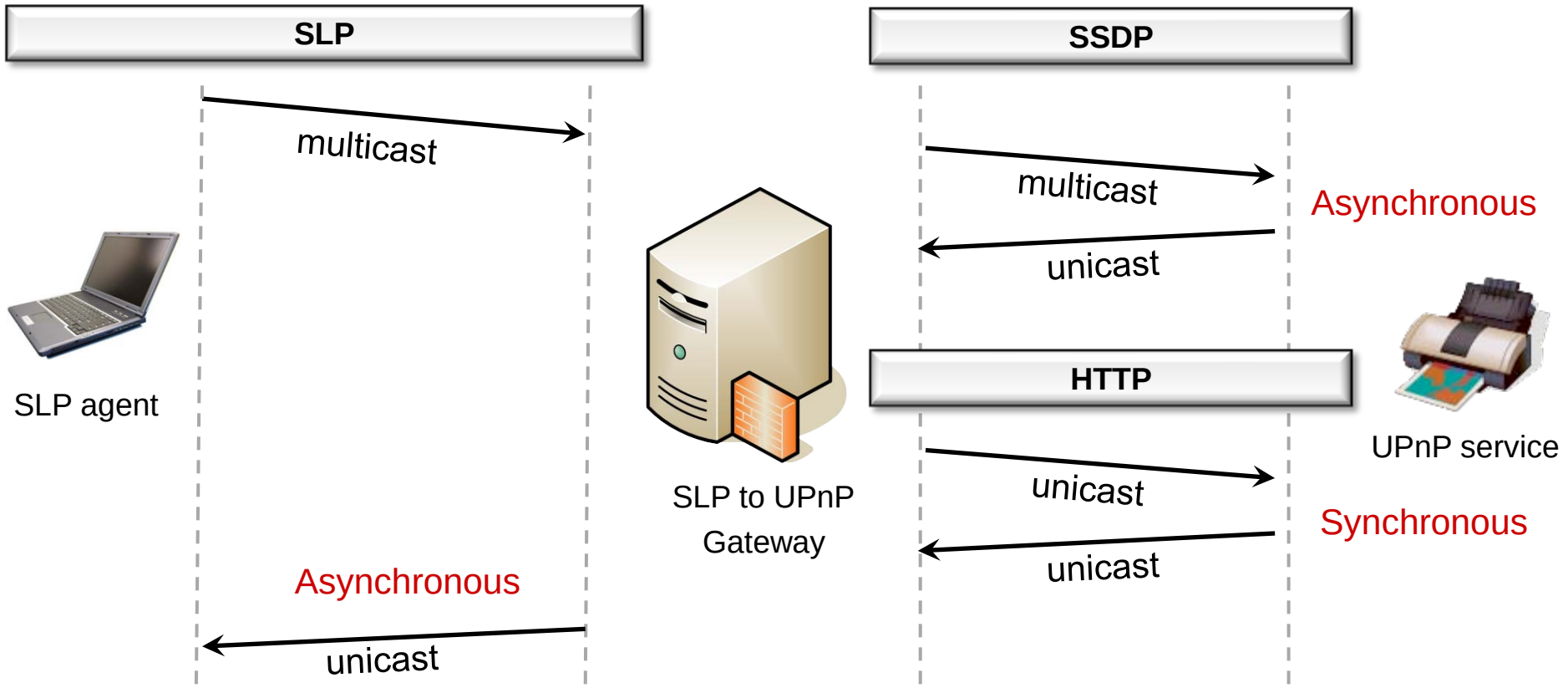
# Heterogeneous communication

- Different communication schemes
  - Unicast / Multicast



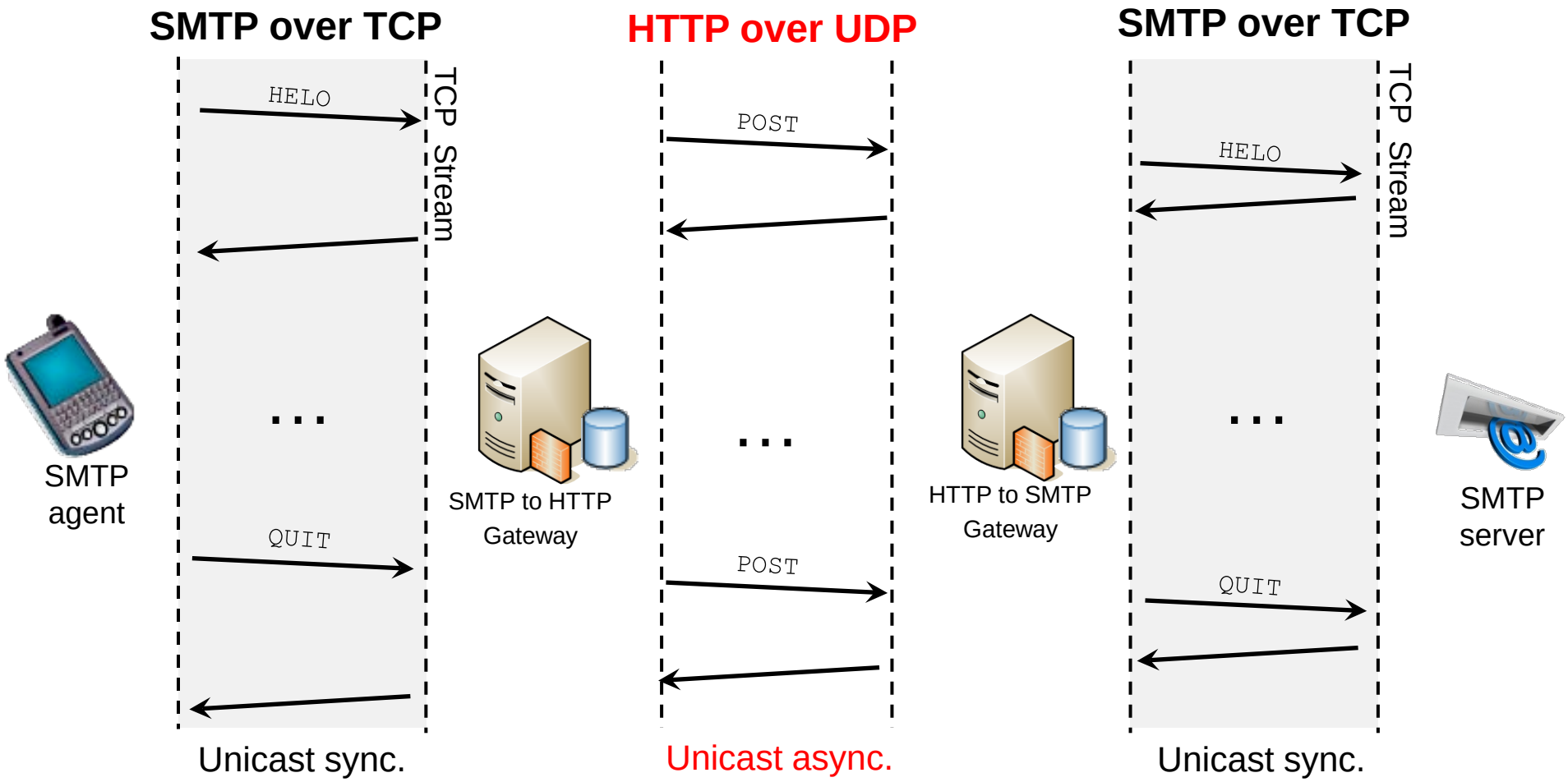
# Heterogeneous communication

- Different communication schemes
  - Unicast / Multicast
  - Synchronous / Asynchronous

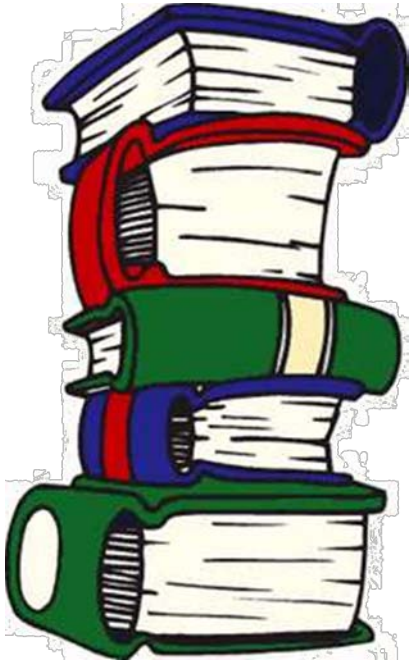


# Session management

- Multiple SMTP messages flow within the same TCP stream
  - ➔ gateways need to manage a state



# Open issue

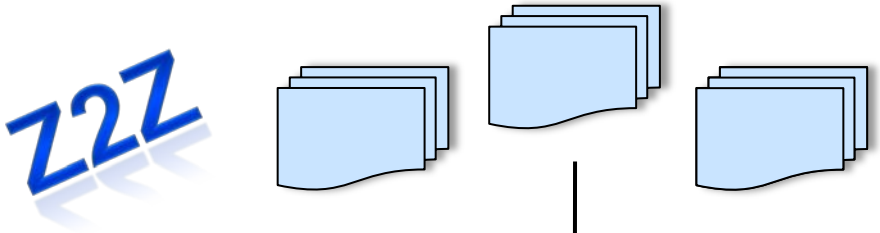


Protocol specification
& translation semantics




Code

# Our approach




High level specifications

Verification

Code generation

- Consistency checks
- Dataflow analyses

Translation engine

Runtime system

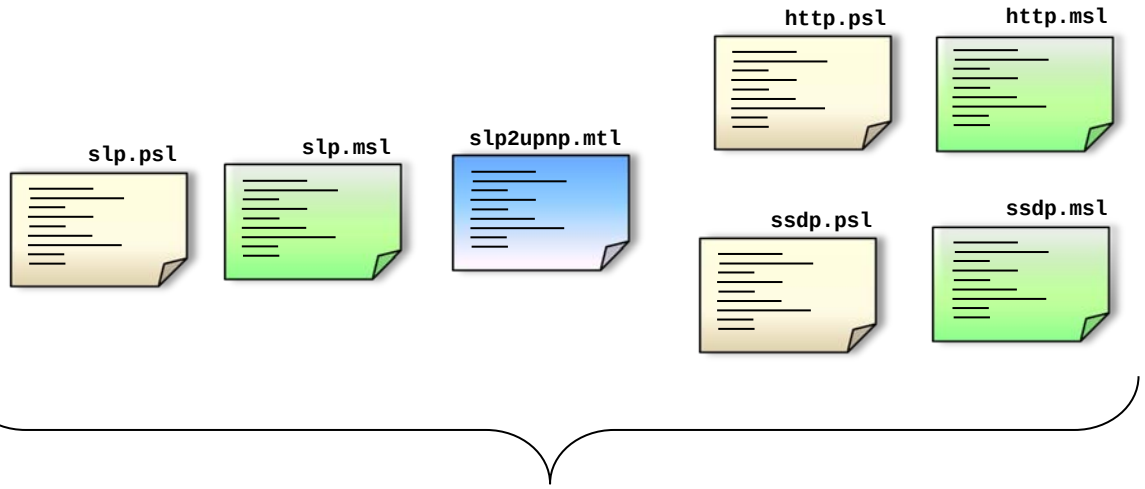
# z2z : an umbrella of DSLs

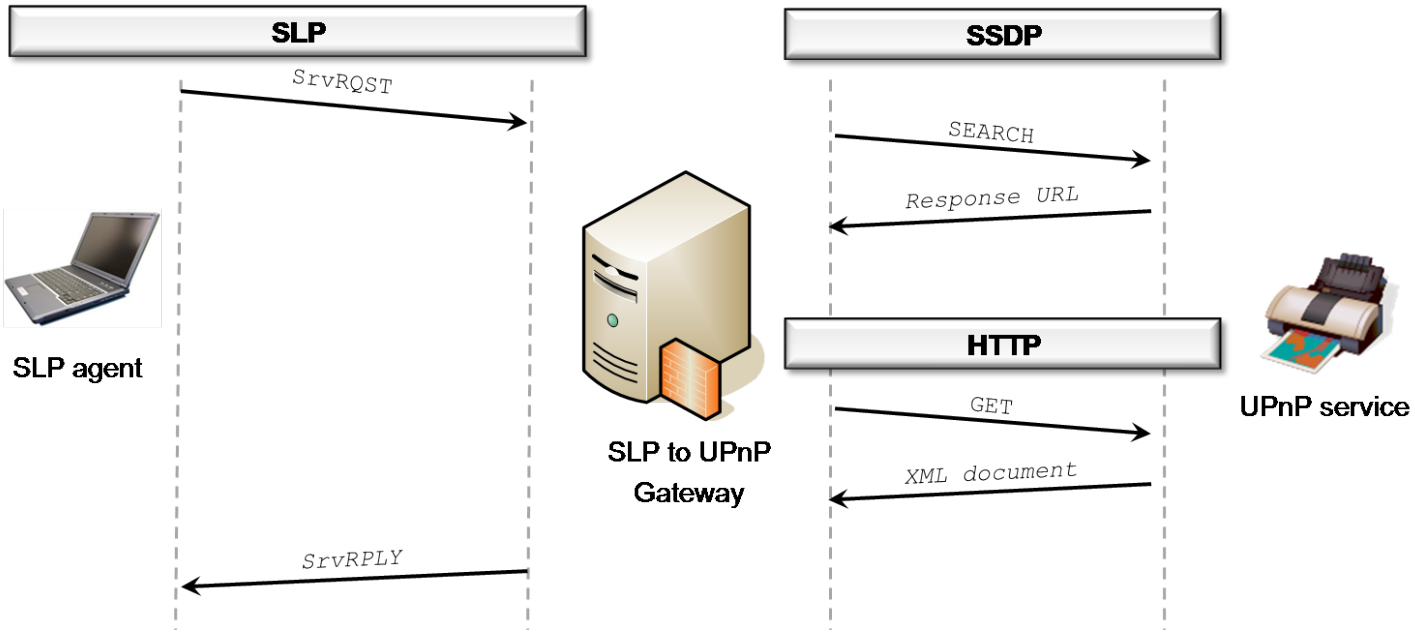



- Protocol specification language (PSL)
  - How to interact with the network to send or receive messages
- Message specification language (MSL)
  - Defines a *view* from an incoming message
  - Defines *templates* to forge new messages
- Message translation language (MTL)
  - Defines a set of handlers
    (one for each relevant request in the source protocol)
  - C-like notation augmented with domain-specific operators

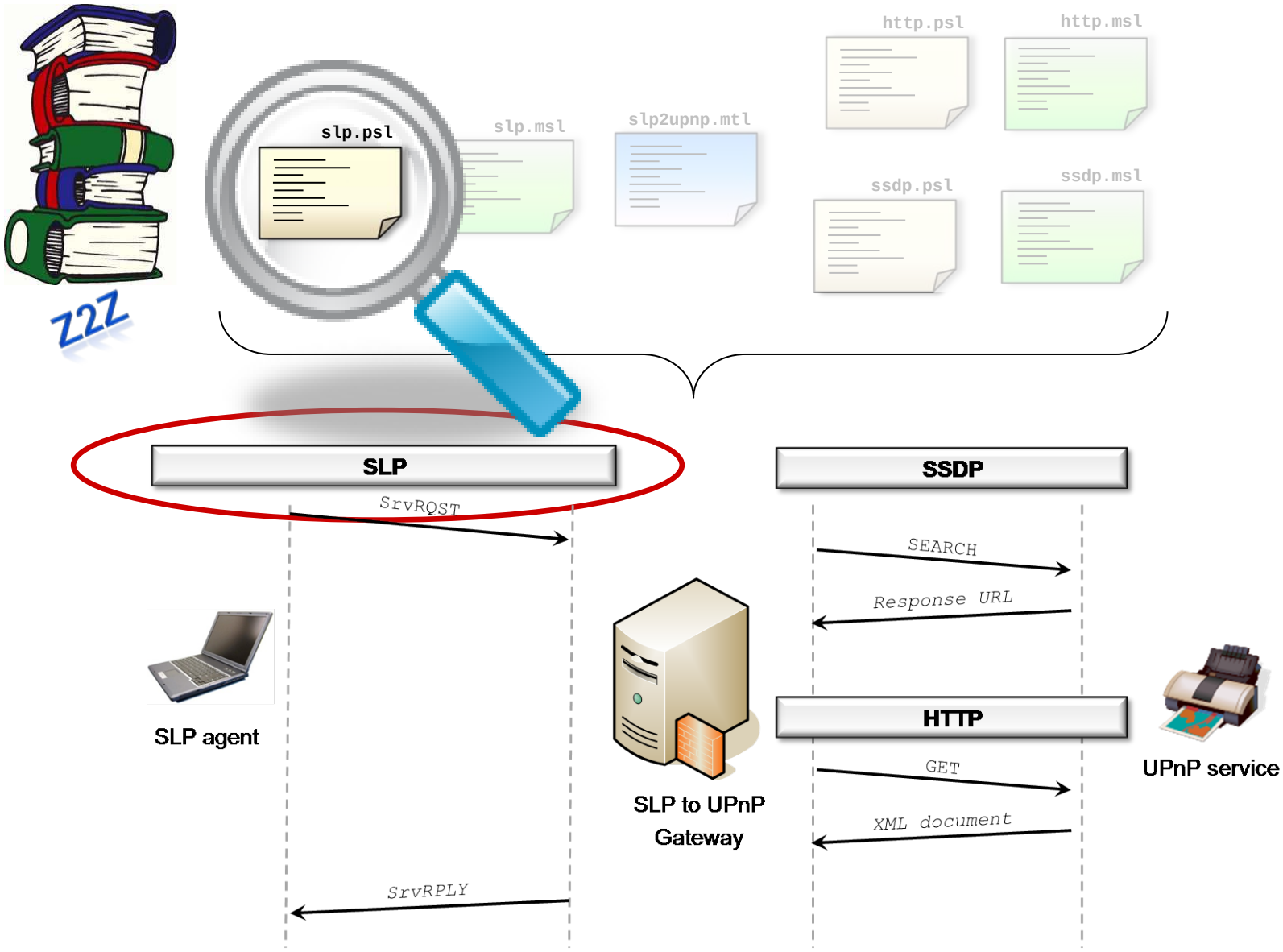
# Z2Z by example



# Protocol Specification (SLP)





# Protocol Specification (SLP)

slp.psl

```
protocol slp {
   attributes {
      transport = udp / 427;
      mode = async / multicast;
      group = 239.255 .255 .253;
   }
   request req {
      response SrvReq when req.function_id == 1;
      …
   }
   sendin                                    req) {
      // a
      re
      re
      …
   }
   fl
   …
}
```

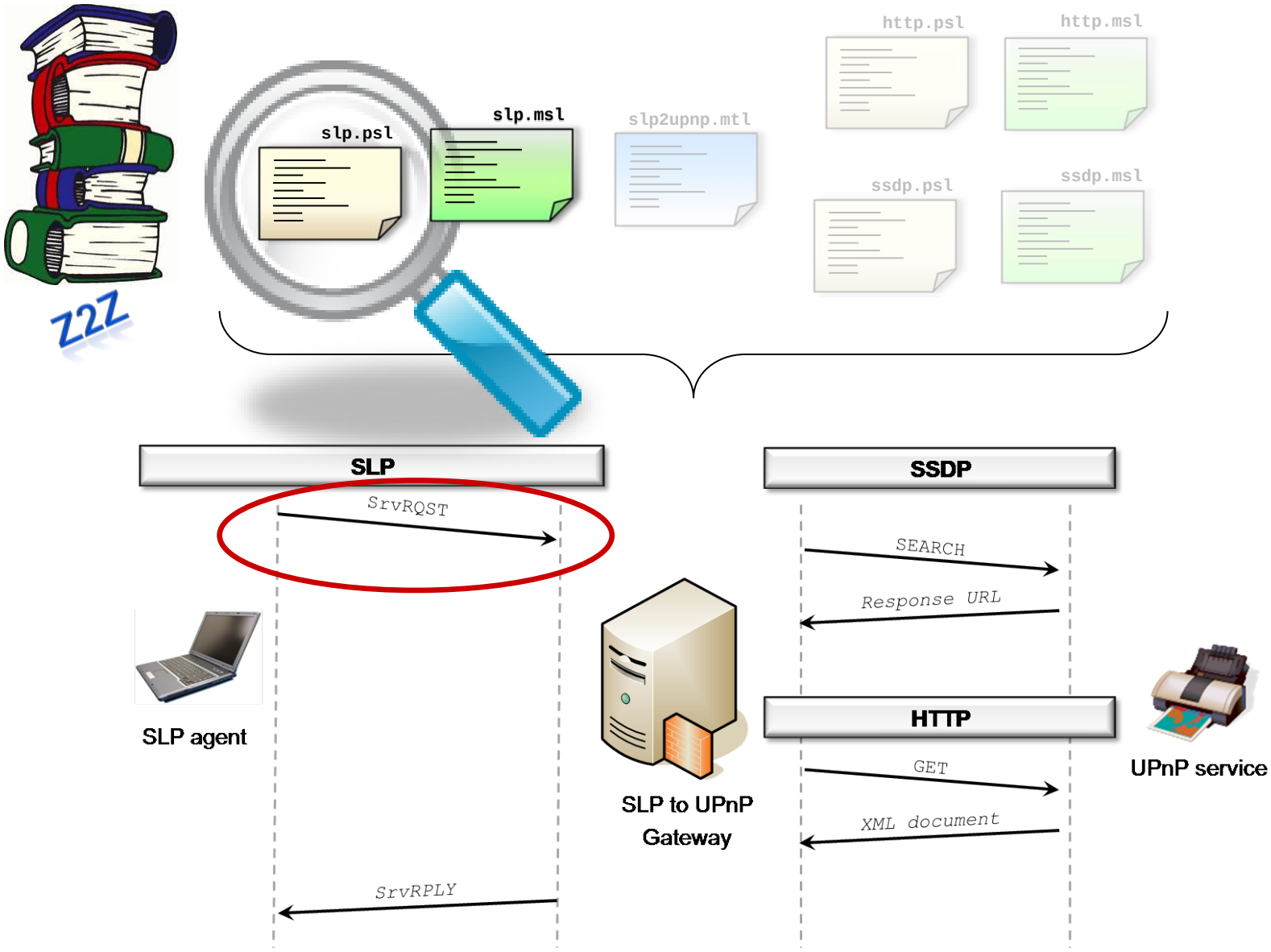
Information about interaction with the network

Requests that should be handled by the gateway

How to identify a transaction:
Response associated to a request

Protocol-specific information that all messages must contain

# Message Specification (SLP)





# Message Specification (SLP)

**slp.msl**

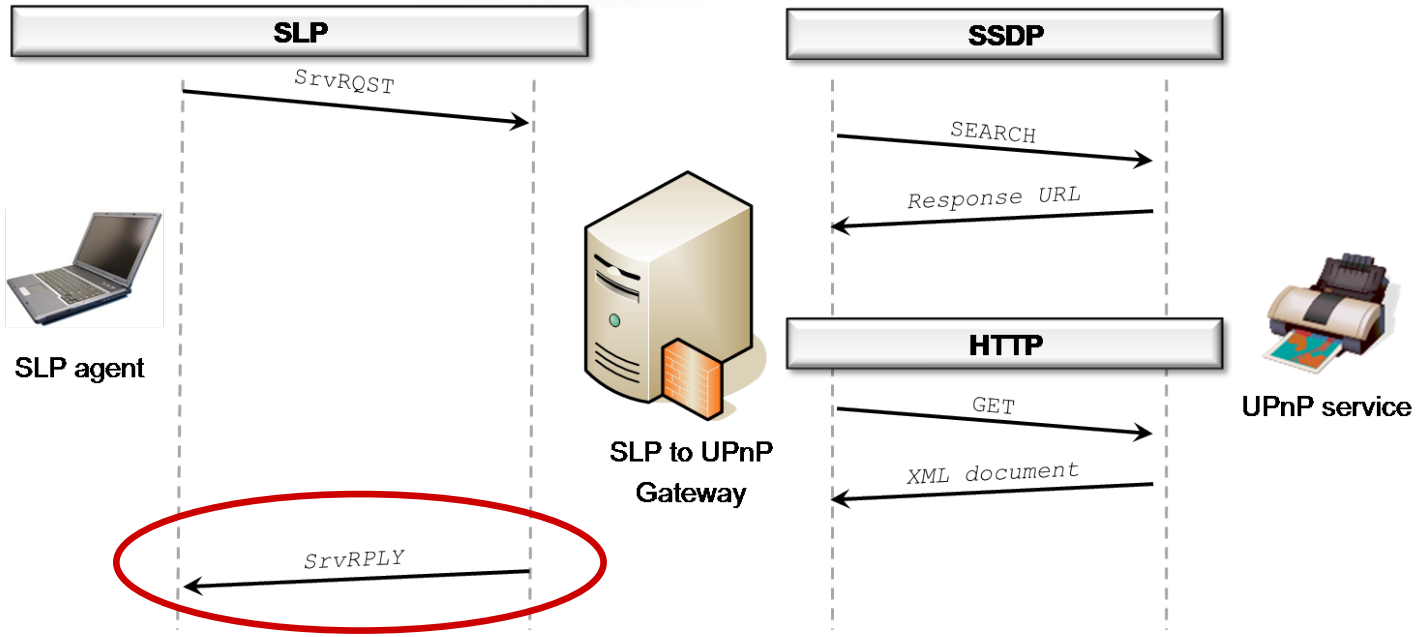
```
message slp {
   read {
      mandatory private int function_id;
      mandatory private int xid;
      optional  public int url_count;
      optional  public fragment list urls;
      optional  public fragment scope;
      optional  public fragment service_type;
      optional  public fragment predicate;
   }
}
```

Message View

- ✓ Interface of ad-hoc message parser
- ✓ Select information useful for the gateway

# Message Specification (SLP)





# Message Specification (SLP)

slp.msl

```
message slp {
   read { … }
   reponse binary template srvReply {
       magic ="foo";
       private int16 total_len;
       private int16 url_len;
--foo
02 02 00 <%tot
<%xid%> 00 02
<%lifetime%> <
--foo
    }
}
```

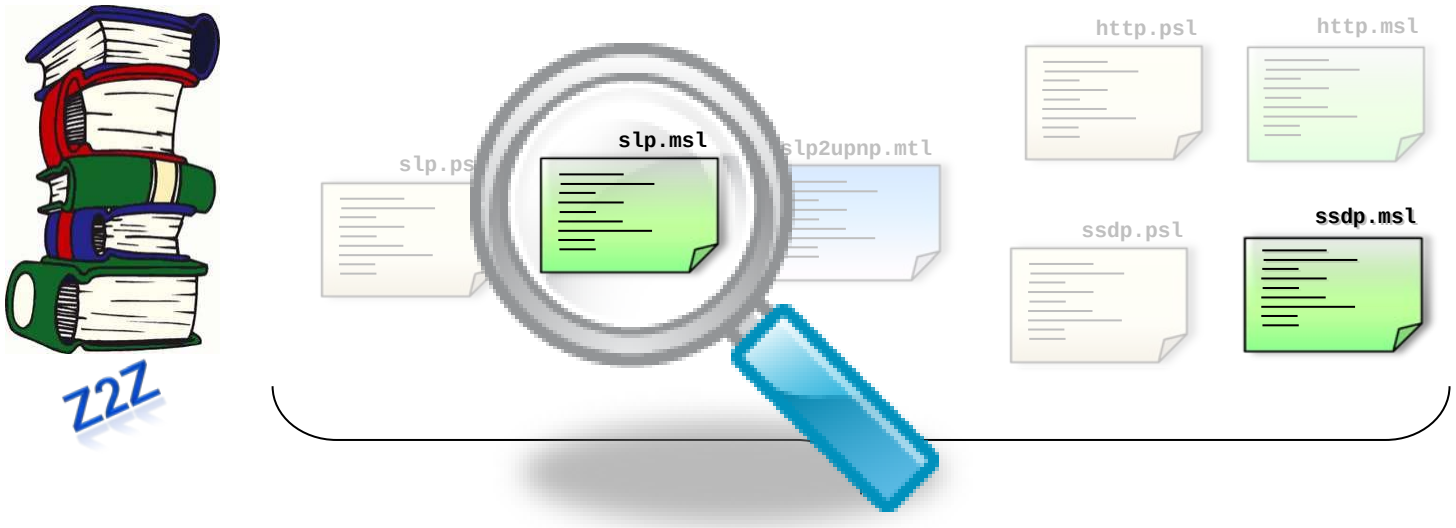
Binary Template

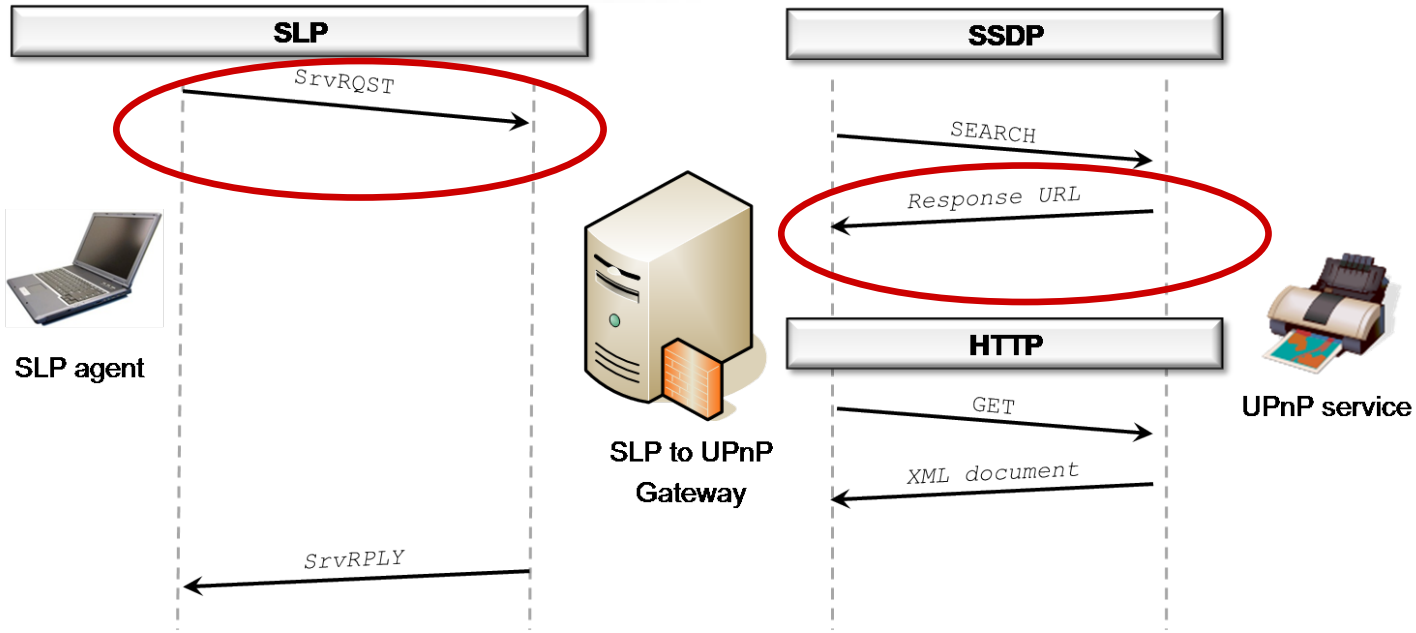
- ✓ Expressed as a sequence of hexadecimal values
- ✓ Holes fields with values fields

Response Template

- ✓ Similar to a message view
- ✓ All field are mandatory
- ✓ Private fields are filled in *sending* block (psl)
- ✓ Public fields must be filled in MTL

# Message Specification (SSDP)





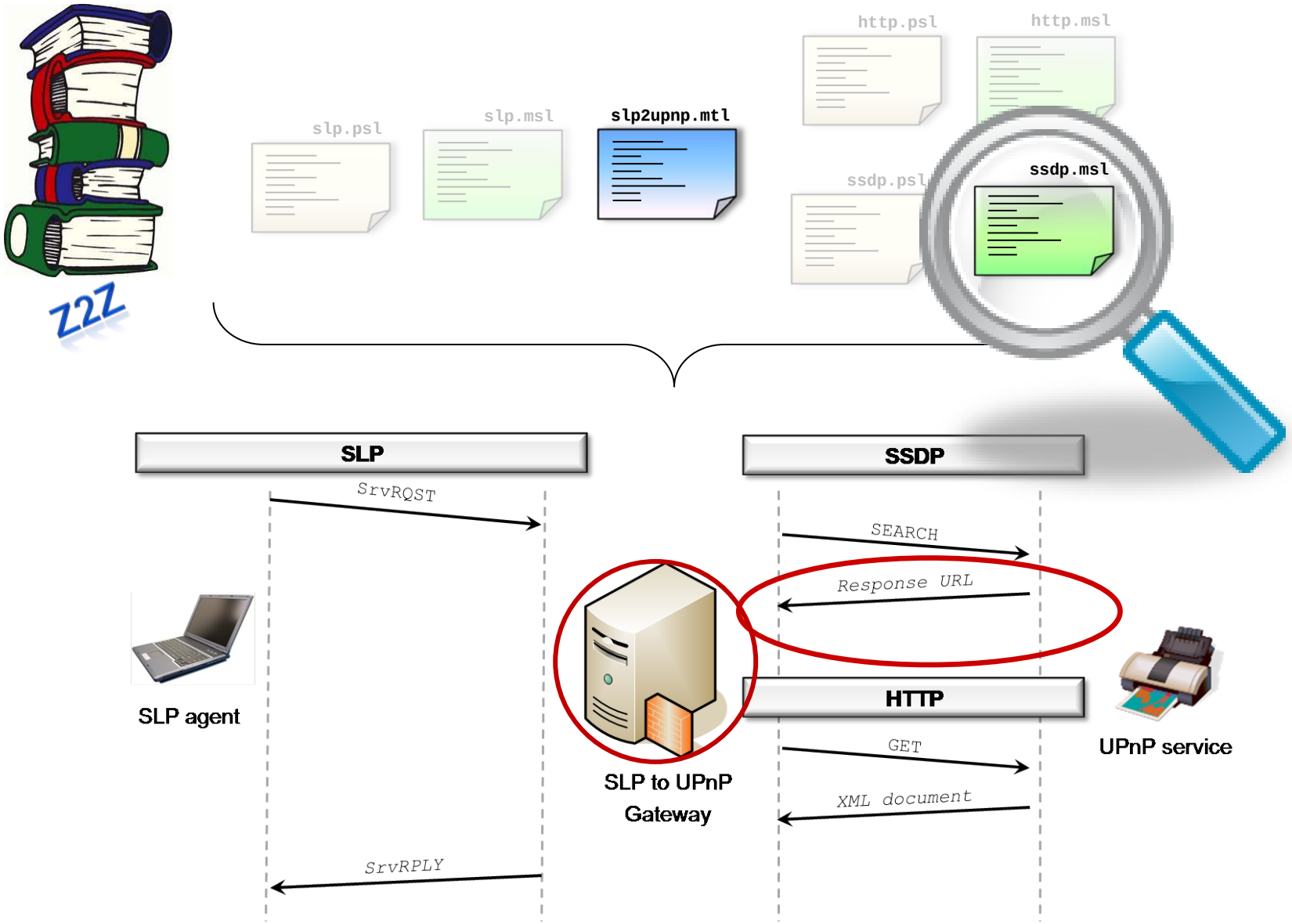
# Message Specification (SSDP)

**ssdp.msl**

```
message ssdp {
   read { … }
   request template response Search {
       magic   = "foo";
       newline = "\r\n";
       public int mx;
       public fragment st;
--foo
M-SEARCH * HTTP/1.1
ST:  <%st%>
MX:  <%mx%>
MAN: "ssdp:discover"
HOST: 239.255.255.250:1900

--foo
   }
}
```

# Message Translation (SLP2UPnP)





# Message Translation

- Defines a set of handlers
  - One for each relevant request in the source protocol
- Handlers written using a C-like notation augmented with domain-specific operators for
  - Manipulating messages
  - Forging messages
  - Sending requests (target protocol)
  - Returning responses (source protocol)
  - Session managment



# Message Translation

**Slp2upnp.mtl**

```
slp response SrvReq (slp request s) {

}
```

SrvReq Handler

- ✓ Parametrized by a view of the request
- ✓ Information in the view extracted using structure field access notation



# Message Translation

**Slp2upnp.mtl**

```
slp response SrvReq (slp request s) {
   ssdp response res_ssdp;
   ssdp message  search_ssdp;
   http response res_http;
   search_ssdp = Search(mx= 3, st="upnp:rootdevice"));


}
```

- ✓ New messages forged by invoking a template (Search)
- ✓ Public fields must be initialized before sending the message



# Message Translation

**Slp2upnp.mtl**

```
slp response SrvReq (slp request s) {
   ssdp response res_ssdp;
   ssdp message  search_ssdp;
   http response res_http;
   search_ssdp = Search(mx= 3, st="upnp:rootdevice"));
   res_ssdp = send(search_ssdp);

}
```

Sending a request (multicast send)

- ✓ Execution pauses until a response is received
- ✓ Semantic preserved regardless of the sync/async properties



# Message Translation

**Slp2upnp.mtl**

```
slp response SrvReq (slp request s) {
   ssdp response res_ssdp;
   ssdp message  search_ssdp;
   http response res_http;
   search_ssdp = Search(mx= 3, st="upnp:rootdevice"));
   res_ssdp = send(search_ssdp);

}
```

Receiving a ssdp response (unicast response)

- ✓ Execution resumed



# Message Translation

**Slp2upnp.mtl**

```
slp response SrvReq (slp request s) {
   ssdp response res_ssdp;
   ssdp mes
   http res
   search_s
   res_ssdp = send(                    _ssdp);
   res_http = send(Get(path = res_ssdp.location_path,
                       ip   = res_ssdp.location_ip,
                       port = res_ssdp.location_port));

}
```

Sending a request (unicast send)

- ✓ HTTP Get message generated and sent
- ✓ Response received synchronously (no continuation)



# Message Translation

**Slp2upnp.mtl**

```
slp response SrvReq (slp request s) {
   ssdp response res_ssdp;
   ssdp message  search_ssdp;
   http response res_http;
   search_ssdp = Search(mx= 3, st="upnp:rootdevice"));
   res_ssdp = send(search_ssdp);
   res_http = send(Get(path =res_ssdp.location_path,
                       ip   = res_ssdp.location_ip,
                       port = res_ssdp.location_port));
   return srvReply(url=res_http.URLBase);
}
```

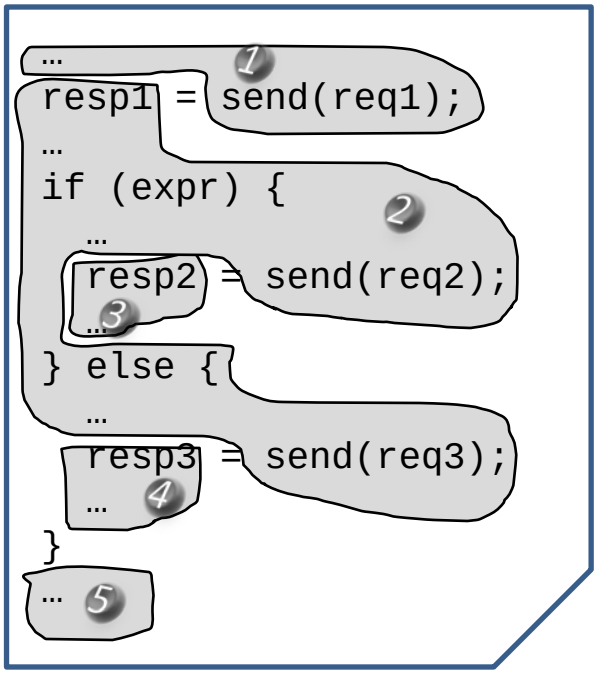
Sending a reply (unicast send)

- ✓ SrvReply message generated and sent

# Managment of asynchronous send

- Handler code
  - sequential function with C-like syntax and semantics
  - Send operation treats as function call returning a value
- Asynchronous target protocol
  - ➔ Send does not return a value
  - Save the current state
  - Save information about the rest of the handler (continuation)
  - Handler restarted at the current point when a response become availables

➔ Handler split in many continuations





# Implementation (current status)

- z2z runtime system
  - ➔7,500 lines of C code

- Domain-specific middleware for gateway construction
  - Offers high-level operations
    - Management of active connections (incoming/outgoing)
    - Dispatch of requests to appropriate handlers
    - Management of continuations for asynchronous communications
    - Message creation (efficient template management)
    - Management of sessions (save/restore of session states)
  - Parameterized by the information of each protocol
    - Communication scheme
    - Message information to identify transaction/session

# Implementation (current status)

- Z2z compiler
  - ➔11,000 lines of OCaml code
- Verifications
  - Consistency checks
    - » Inter-module dependencies
  - Dataflow analyses
    - » Check that values are well-defined when they are used (MT)
- Code generation
  - ➔ Generates C code
  - Send operation as split point
  - Variables and asynchronous sends
  - Dynamic memory management (reference counts)

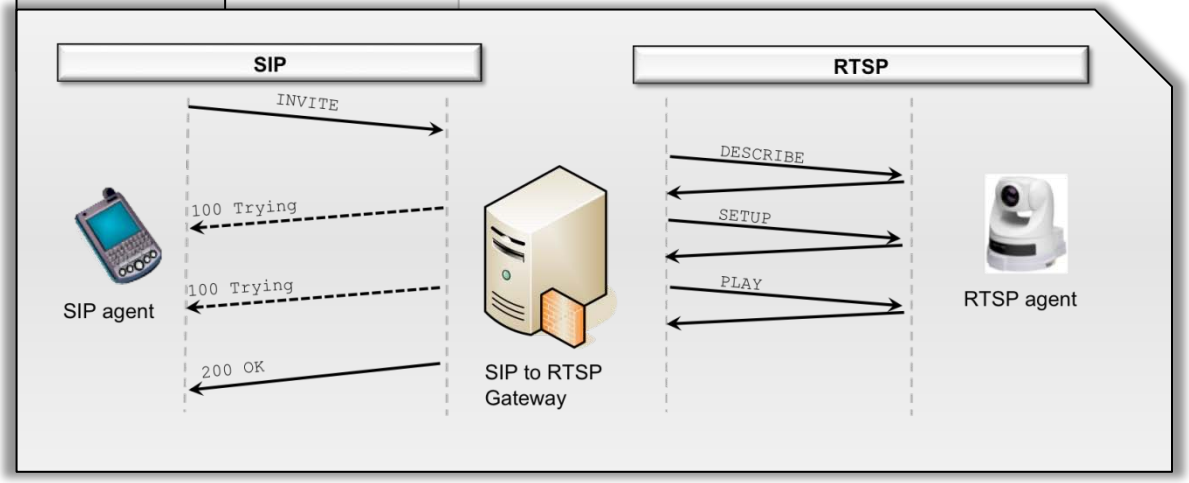
# Evaluation

- 2 case studies
  - SIP/RTSP gateway
  - SLP/UPnP gateway

- 3 kinds of evaluation
  - Size of specifications
  - Response time
  - Dynamic memory consumption at runtime

# Evaluation SIP/RTSP, Size of Specifications

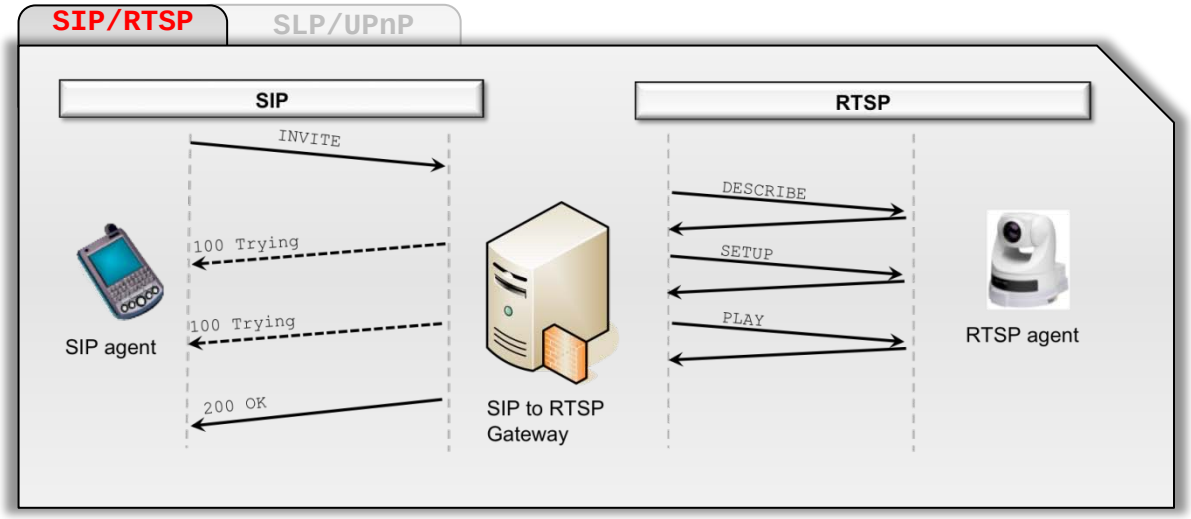
SIP/RTSP | SLP/UPnP




Line of code | Response time | Memory consumption

| | | Input specification (lines of z2z code) | | | Parser (lines of C code) | z2z gateway (size in (KB)) | | |
|---|---|---|---|---|---|---|---|---|
| | | PS | MS | MT | | Generated modules | Runtime systeme | Total |
| SIP/RTSP | SIP | 24 | 118 | 102 | 168 | 72 | 80 | **152** |
| | RTSP | 20 | 104 | | 210 | | | |
| | SDP | | 12 | | 83 | | | |

# Evaluation SIP/RTSP, Response Time




Line of code | **Response time** | Memory consumption

**Native Service Access**

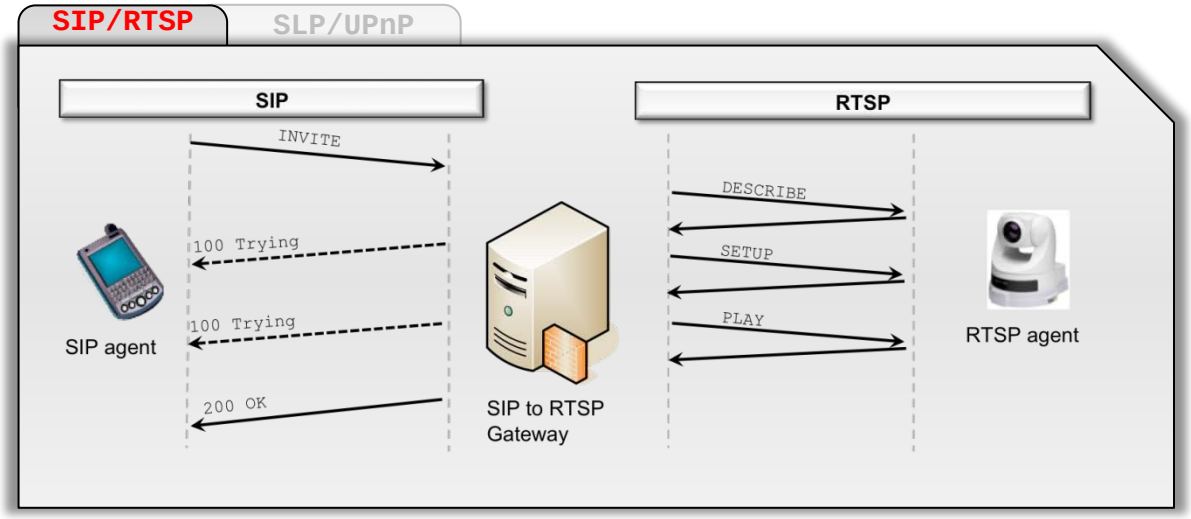
| | SIP <-> SIP | RTSP <-> RTSP |
|---|---|---|
| Time (ms) | 351 | 701 |

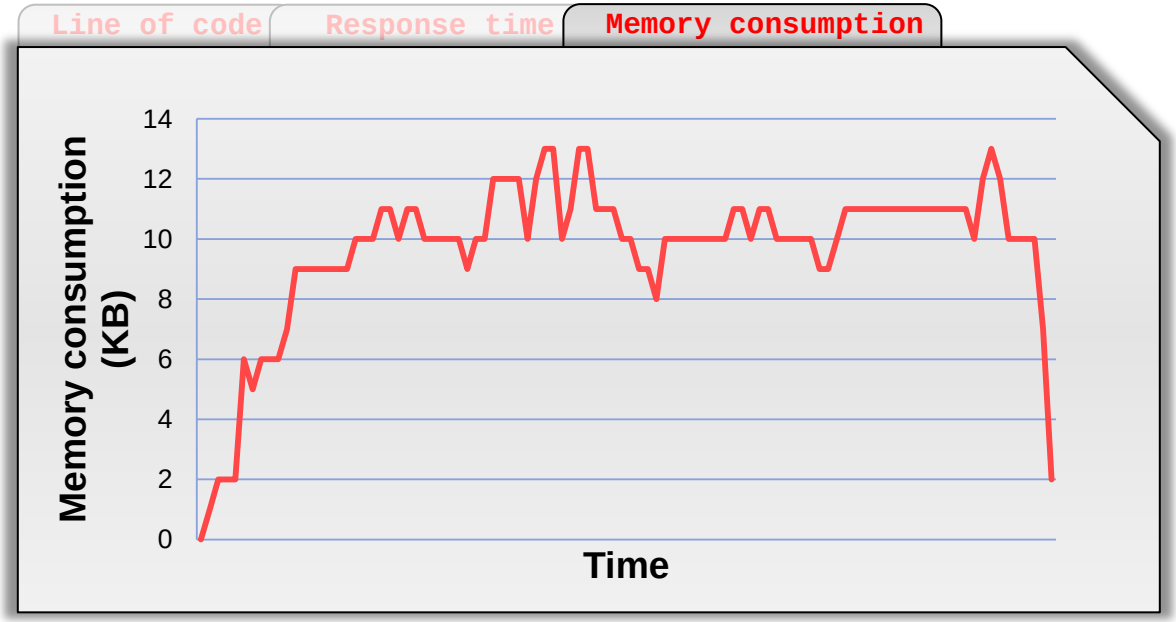
**Z2Z**

| SIP <-> RTSP |
|---|
| **986** |

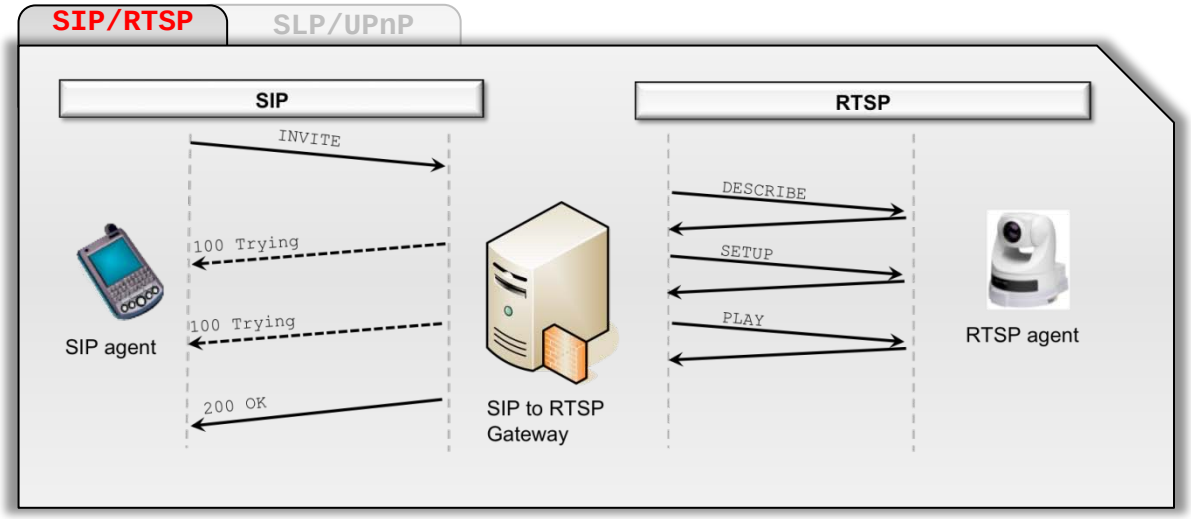
- No overhead
  - Z2Z response time less than the sum of the native response times

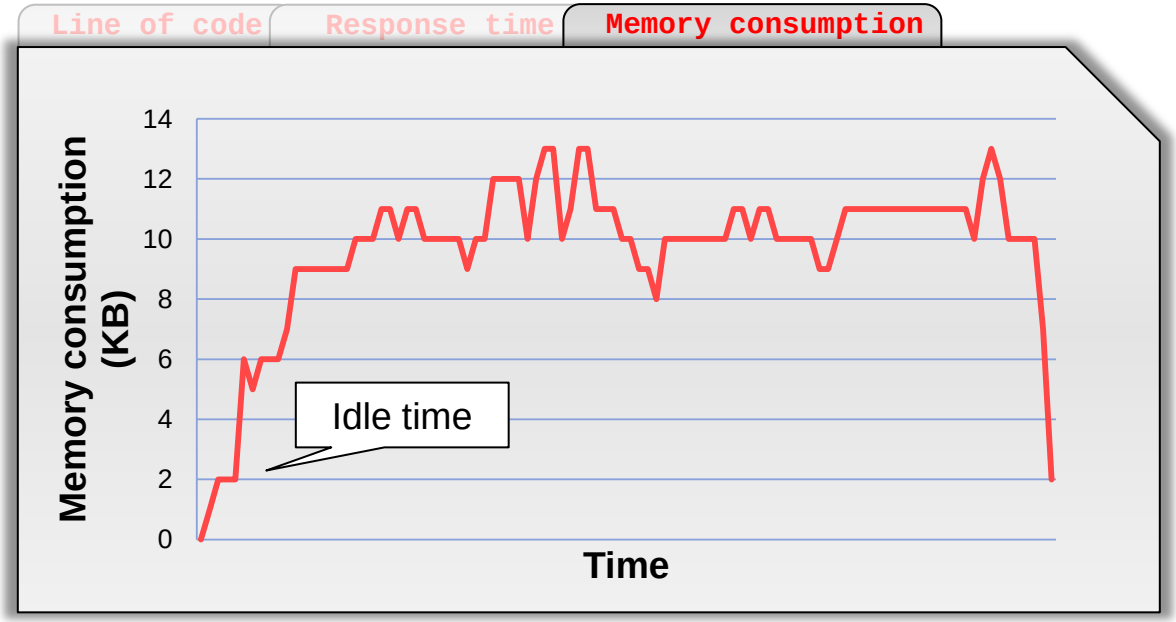
# Evaluation SIP/RTSP, Memory Consumption



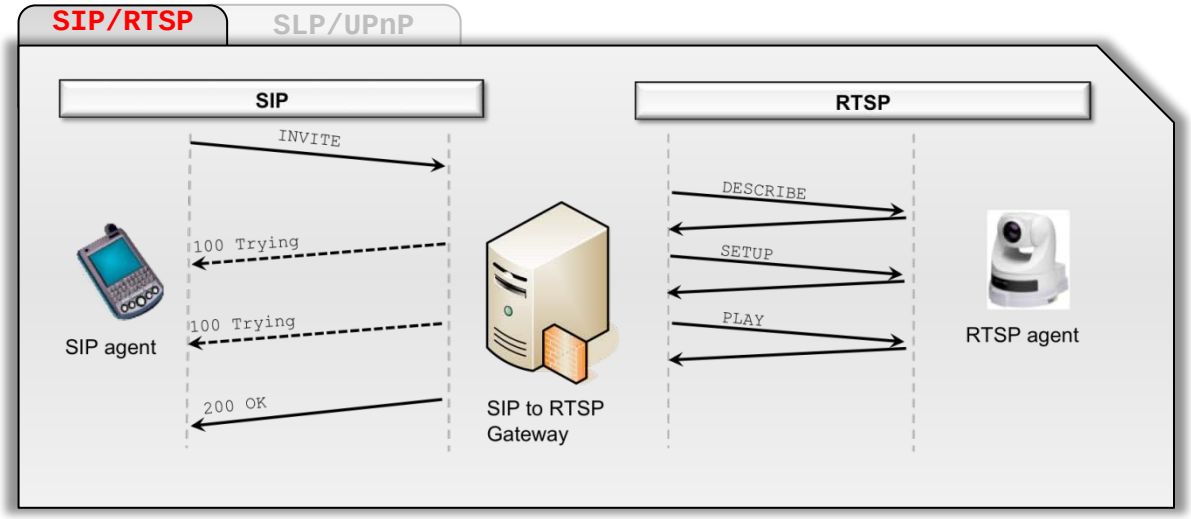



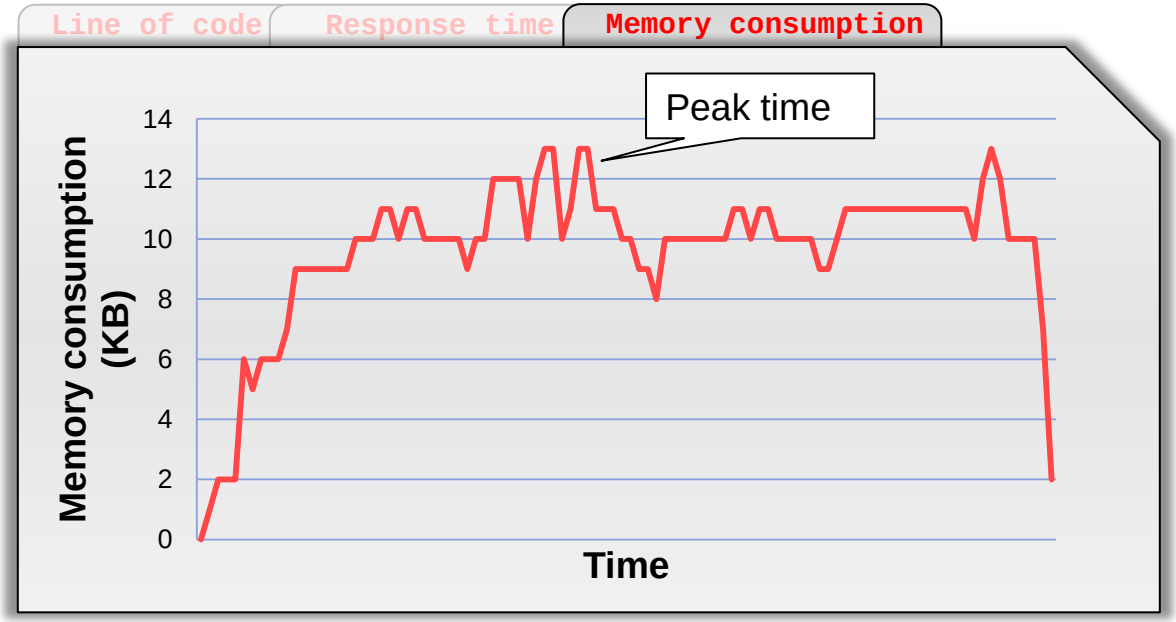
# Evaluation SIP/RTSP, Memory Consumption



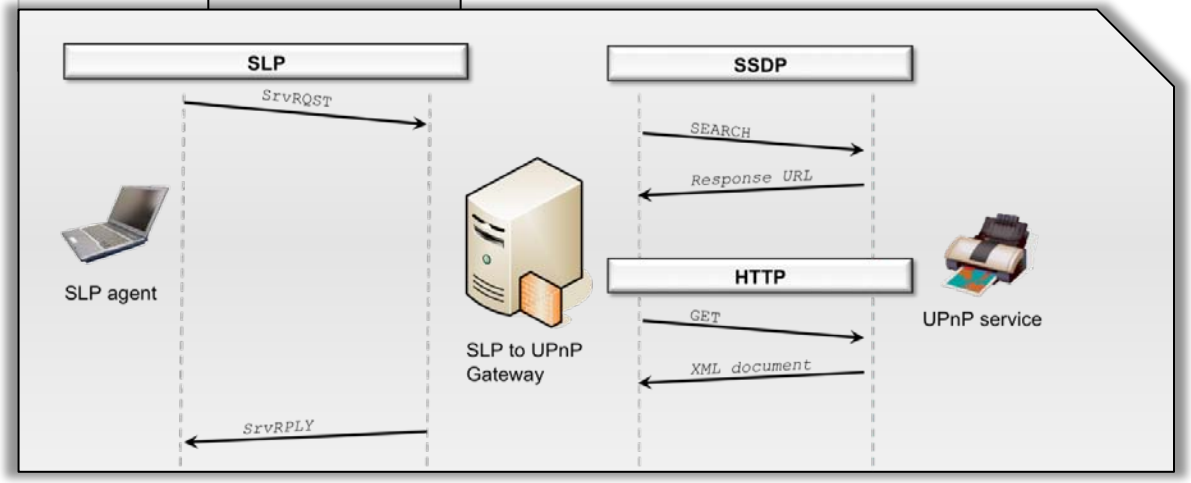
# Evaluation SIP/RTSP, Memory Consumption



# Evaluation SLP/UPnP, Size of Specifications

SIP/RTSP | **SLP/UPnP**

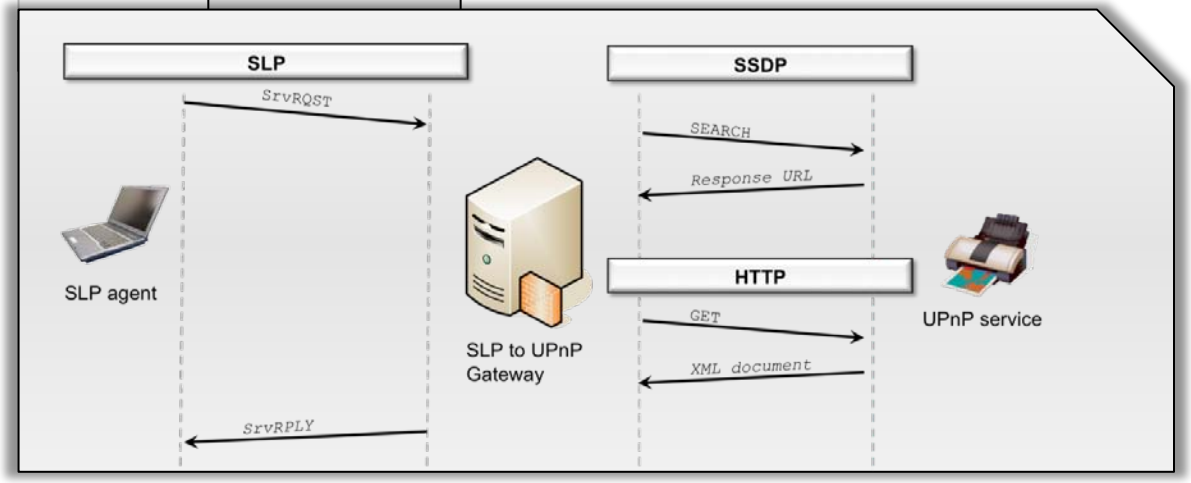



**Line of code** | Response time | Memory consumption

| | | Input specification (lines of z2z code) | | | Parser (lines of C code) | z2z gateway (size in (KB)) | | |
|---|---|---|---|---|---|---|---|---|
| | | PS | MS | MT | | Generated modules | Runtime systeme | Total |
| **SLP/UPnP** | SLP | 12 | 21 | 5 | 166 | 44 | 80 | 124 |
| | SSDP | 6 | 31 | | 223 | | | |
| | HTTP | 9 | 43 | | 178 | | | |

# Evaluation SLP/UPnP, Size of Specifications

SIP/RTSP | SLP/UPnP



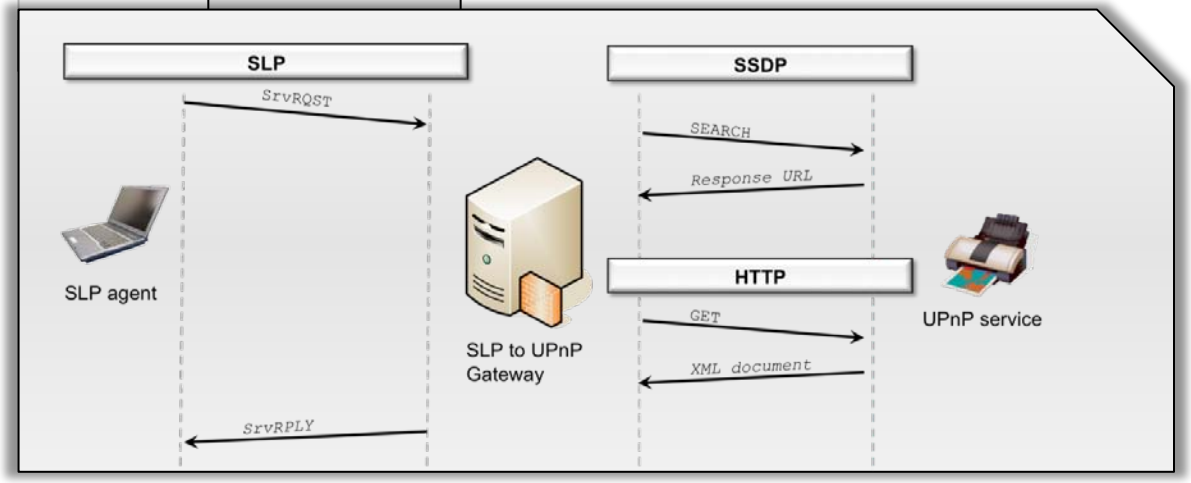

Line of code | Response time | Memory consumption

| | | Input specification (lines of z2z code) | | | Parser (lines of C code) | z2z gateway (size in (KB)) | | |
|---|---|---|---|---|---|---|---|---|
| | | PS | MS | MT | | Generated modules | Runtime systeme | Total |
| SLP/UPnP | SLP | 12 | 21 | 5 | 166 | 44 | 80 | 124 |
| | SSDP | 6 | 31 | | 223 | | | |
| | HTTP | 9 | 43 | | 178 | | | |

Complete z2z specification is around 100 lines of code

# Evaluation SLP/UPnP, Size of Specifications

SIP/RTSP | **SLP/UPnP**

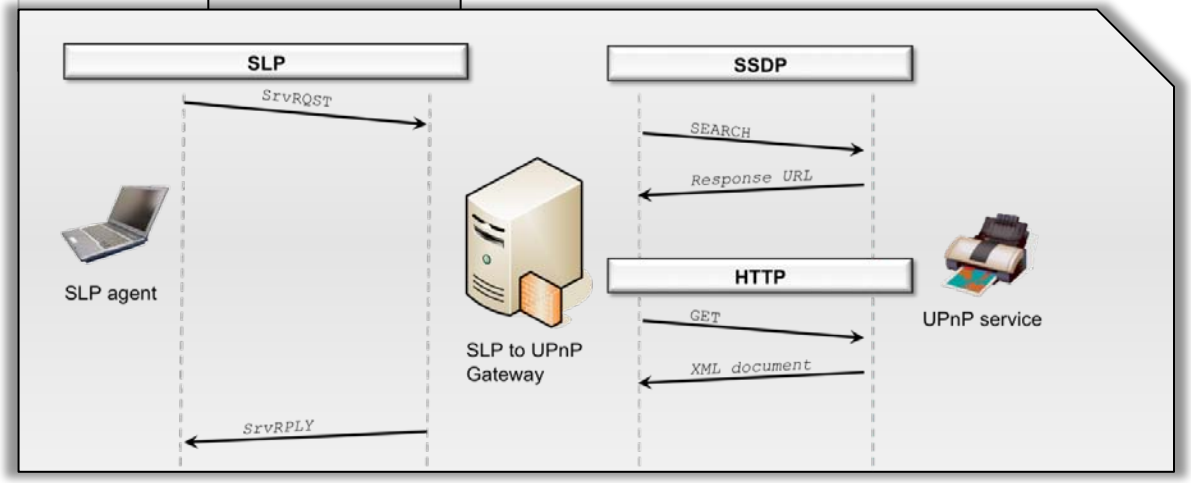



**Line of code** | Response time | Memory consumption

| | | Input specification (lines of z2z code) | | | Parser (lines of C code) | z2z gateway (size in (KB)) | | |
|---|---|---|---|---|---|---|---|---|
| | | PS | MS | MT | | Generated modules | Runtime systeme | Total |
| **SLP/UPnP** | SLP | 12 | 21 | 5 | 166 | 44 | 80 | 124 |
| | SSDP | 6 | 31 | | 223 | | | |
| | HTTP | 9 | 43 | | 178 | | | |

# Evaluation SLP/UPnP, Size of Specifications

SIP/RTSP | **SLP/UPnP**

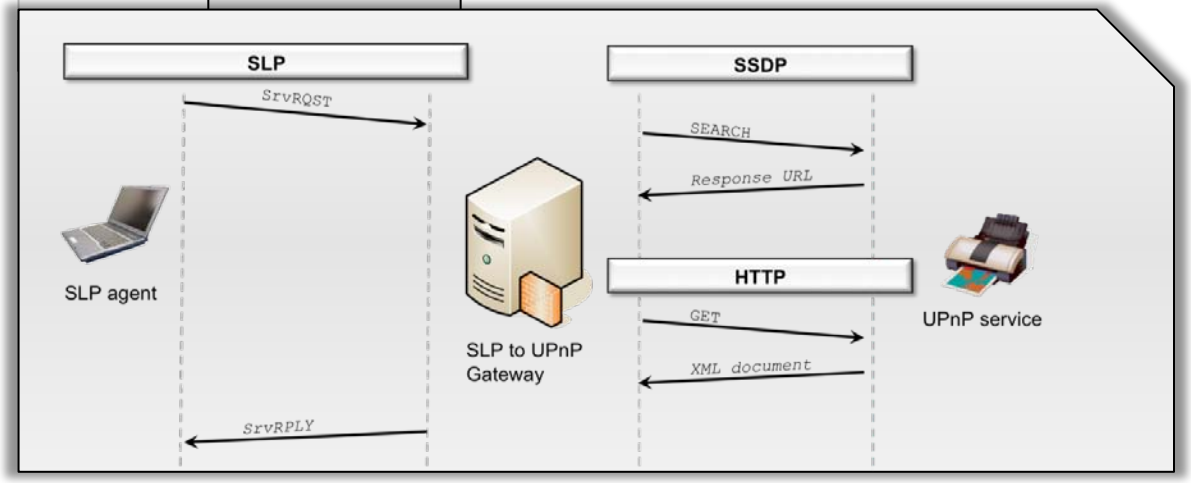



**Line of code** | Response time | Memory consumption

| | | Input specification (lines of z2z code) | | | Parser (lines of C code) | z2z gateway (size in (KB)) | | |
|---|---|---|---|---|---|---|---|---|
| | | PS | MS | MT | | Generated modules | Runtime systeme | Total |
| **SLP/UPnP** | SLP | 12 | 21 | 5 | 166 | 44 | 80 | 124 |
| | SSDP | 6 | 31 | | 223 | | | |
| | HTTP | 9 | 43 | | 178 | | | |

# Evaluation SLP/UPnP, Response Time

SIP/RTSP | **SLP/UPnP**




Line of code | **Response time** | Memory consumption

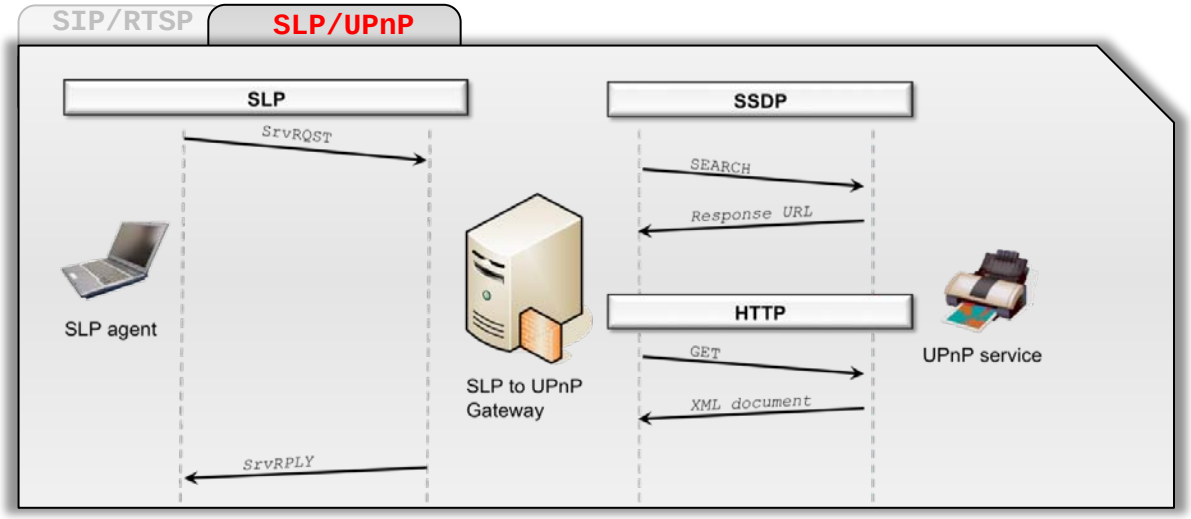
**Native Service Access**

| | SLP <-> SLP | UPnP <-> UPnP |
|---|---|---|
| Time | 2 ms | 58 ms |

**Z2Z**

| SLP <-> UPnP |
|---|
| 78 ms |

# Evaluation SLP/UPnP, Response Time




Line of code | **Response time** | Memory consumption

**Native Service Access**

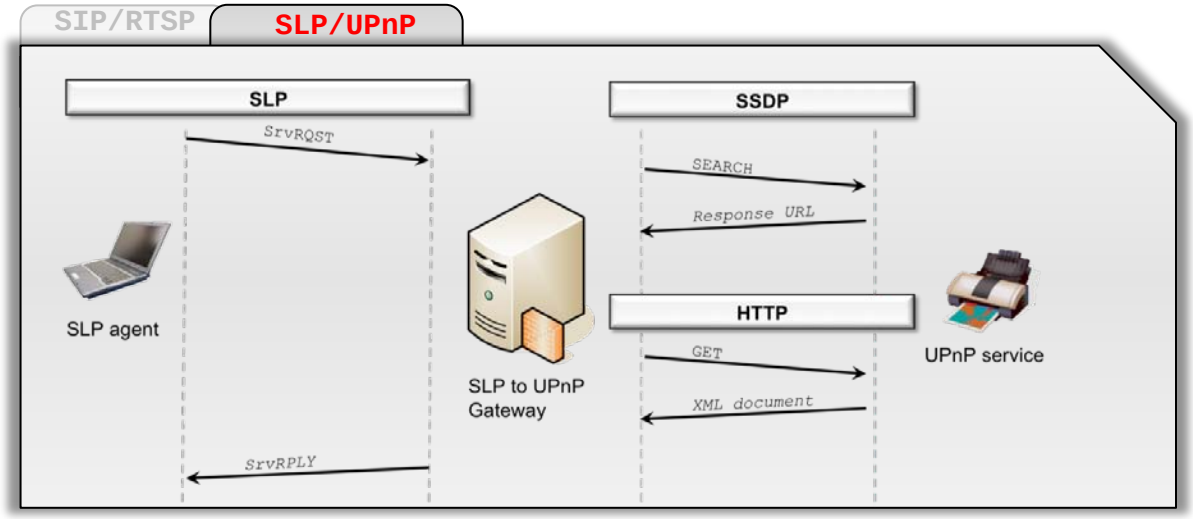
| | SLP <-> SLP | UPnP <-> UPnP |
|---|---|---|
| Time | 2 ms | 58 ms |

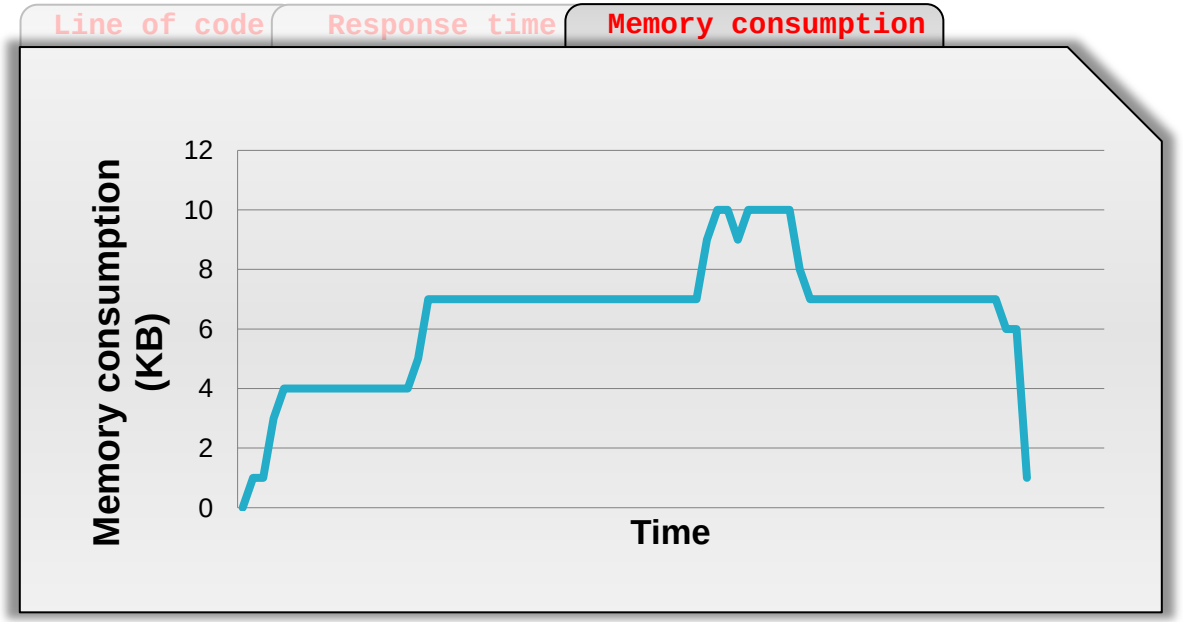
**Z2Z**

| SLP <-> UPnP |
|---|
| 78 ms |

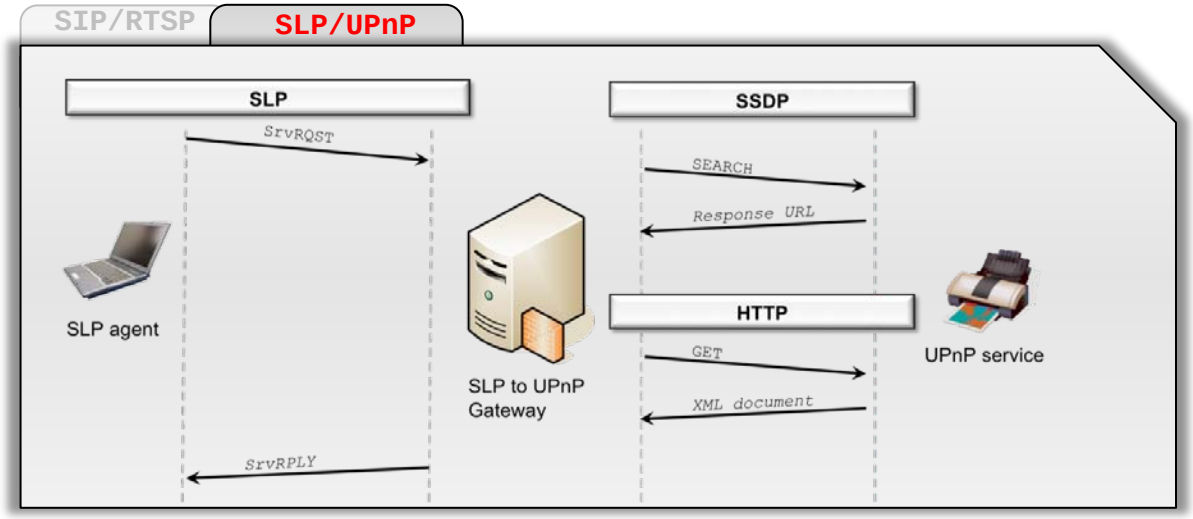
- Z2Z response time little bit higher
  - Polling to check incomming asynchronous responses
  - Not a one-to-one mapping

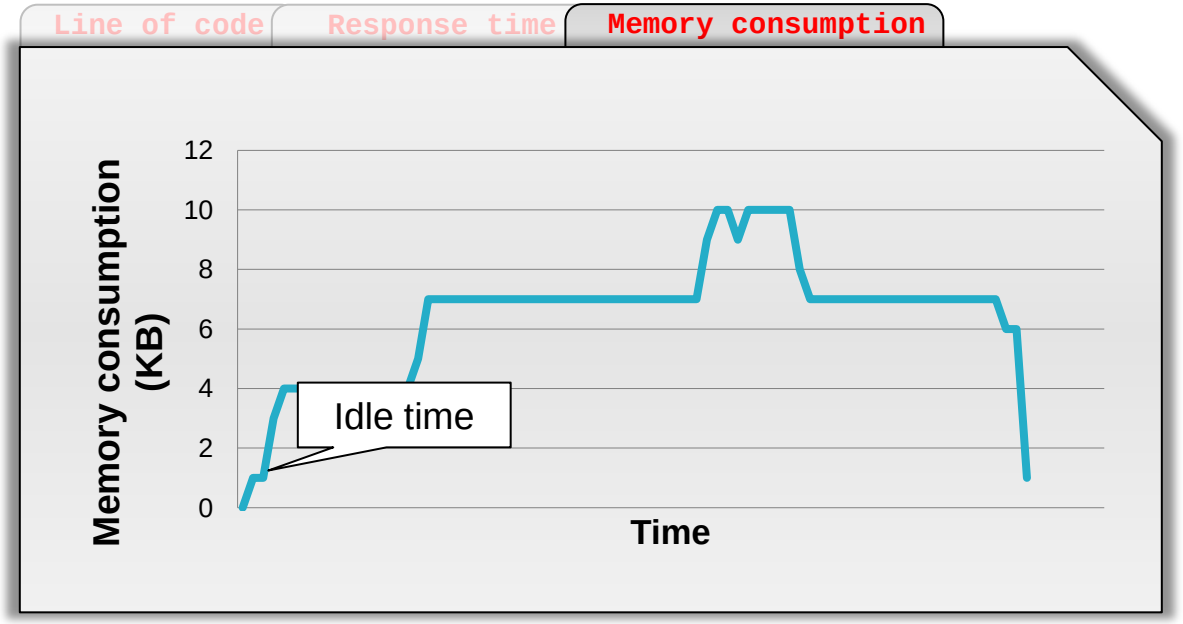
# Evaluation SLP/UPnP, Memory Consumption



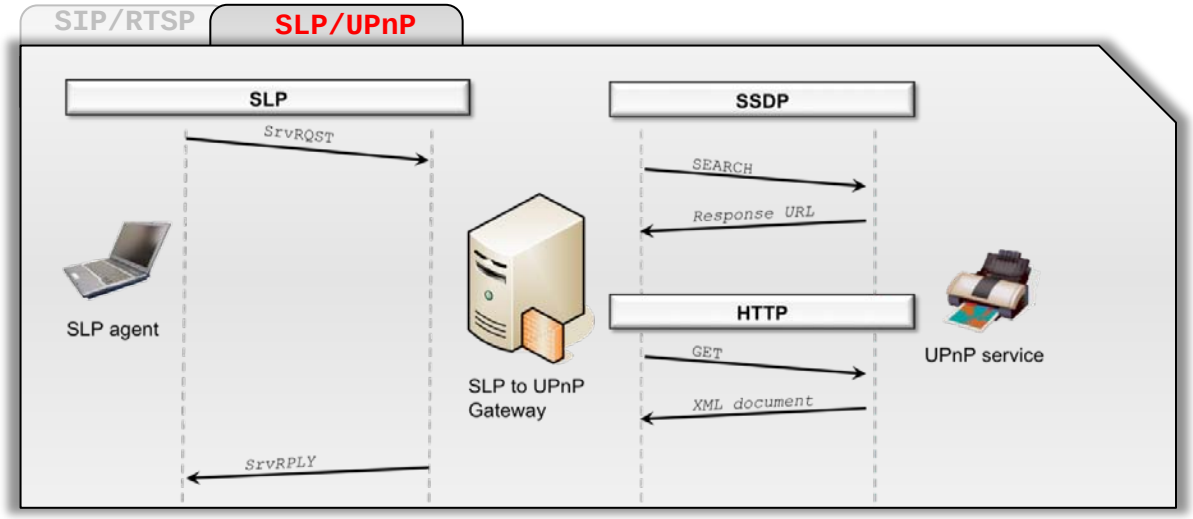



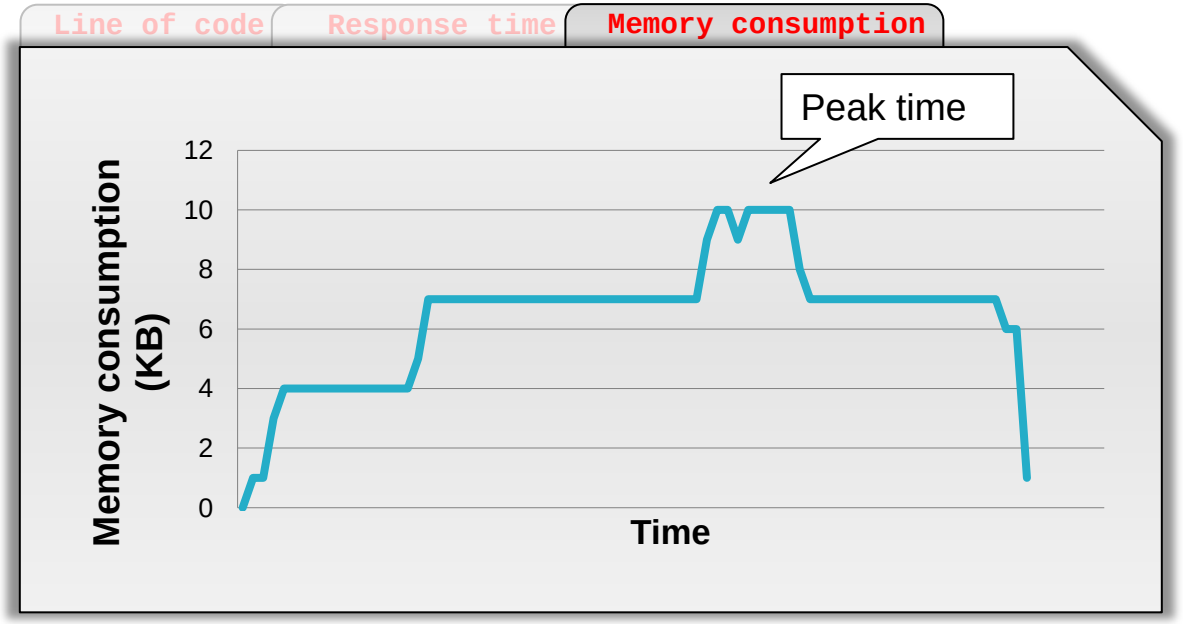
# Evaluation SLP/UPnP, Memory Consumption



# Evaluation SLP/UPnP, Memory Consumption



# Conclusion / Perspectives

- A generative language-based approach
  - Simplify gateway construction
  - Hide low-level networking code
  - Check correctness properties
  - Produce efficient code
- Extend z2z approach to take into account
  - Error recovery
    - Handle failures among the participants of the interaction
      - Timeout
      - Try and catch semantic