CS425 /ECE428 – Distributed Systems – Spring 2020

Remote Procedure Calls & Distributed Objects

Material derived from slides by I. Gupta, M. Harandi, J. Hou, S. Mitra, K. Nahrstedt, N. Vaidya

Announcements

- MP2 extension until April 17
  - MP3 released Monday, will be reduced difficulty
- HW5 out today, due on Apr 21

- Can switch to credit/no credit by April 30
- Will support switch to 3-credit section

Communication b/w Processes

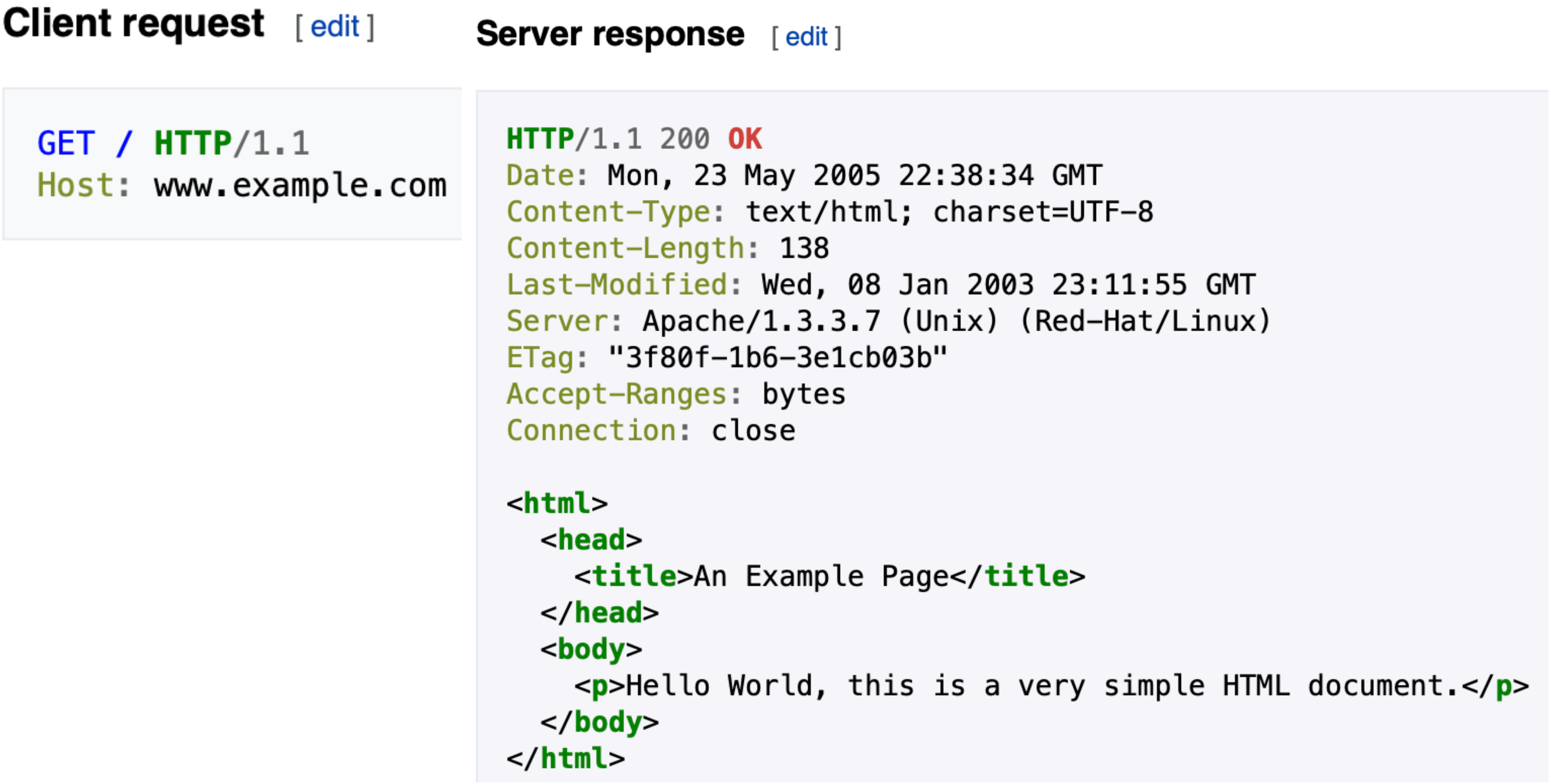
- Message-based distributed systems
  - E.g., Ping-Ack
  - E.g., Election/Coordinator
  - E.g., DHT Lookup/Insert
  - E.g., RequestVotes/AppendEntries
- What do these look like?

Process Communication

- Explicit Messages
  - Sender formats data, receiver parses it
- Remote Procedure Call (RPC)
  - Call procedure/function on remote process
  - Pass values as parameters / receive return values
- Remote Method Invocation (RMI) & Distributed Objects
  - Call *methods* on *remote objects*
  - Pass *remote references*

Messages—Text

HyperText Transfer Protocol



Client request [edit]

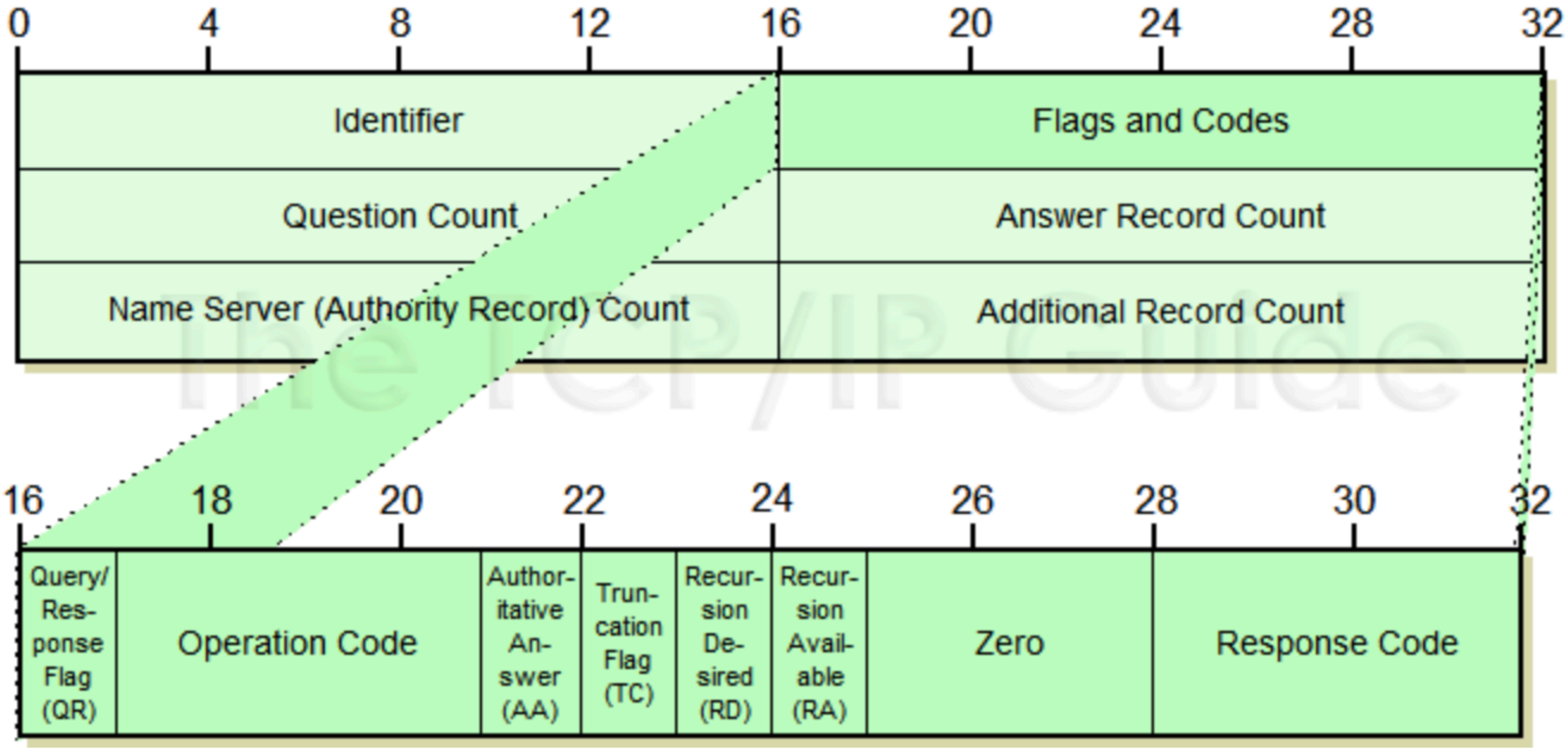
```
GET / HTTP/1.1
Host: www.example.com
```

Server response [edit]

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
  <head>
    <title>An Example Page</title>
  </head>
  <body>
    <p>Hello World, this is a very simple HTML document.</p>
  </body>
</html>
```

Messages—Binary

Domain Name System (DNS)



0–15	16–31
Identifier	Flags and Codes
Question Count	Answer Record Count
Name Server (Authority Record) Count	Additional Record Count

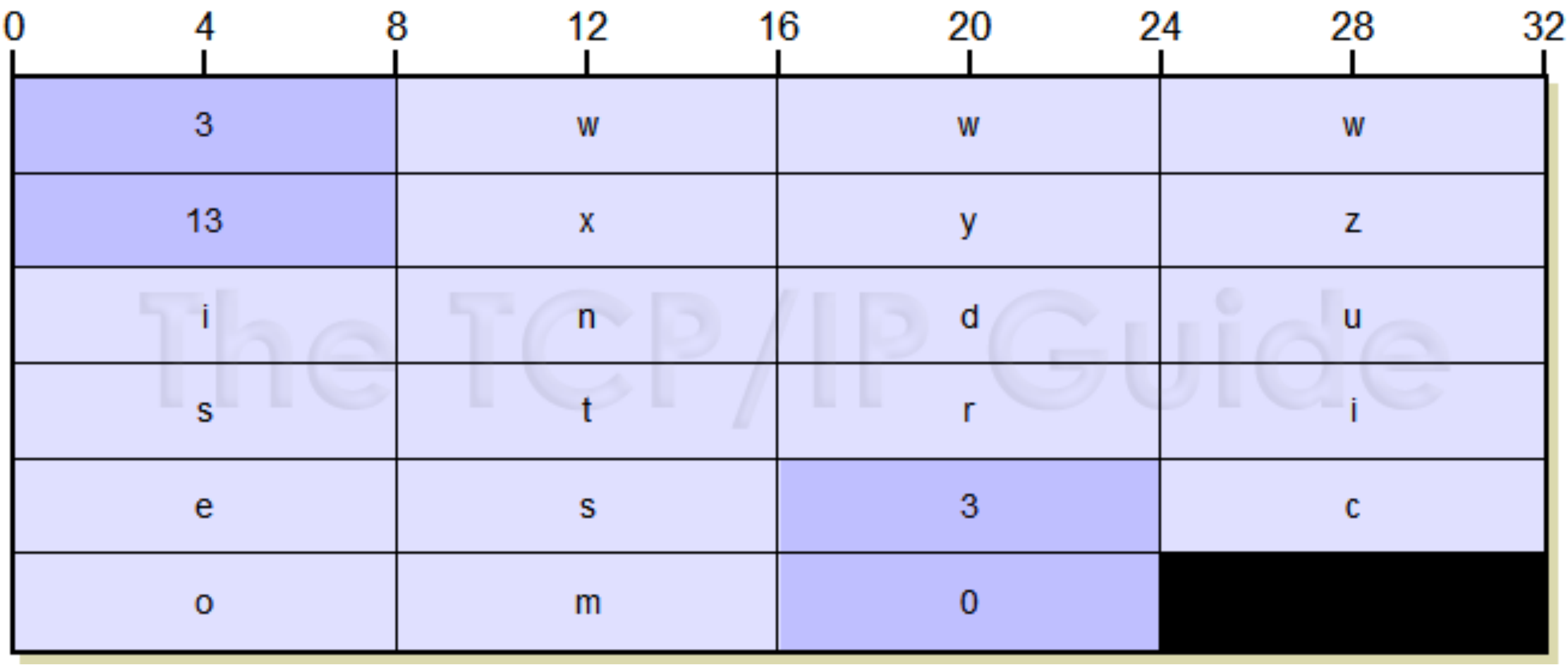
Query/ Response Flag (QR)	Operation Code	Authoritative Answer (AA)	Truncation Flag (TC)	Recursion Desired (RD)	Recursion Available (RA)	Zero	Response Code

Figure 248: DNS Message Header Format

Message Challenges

- Parsing
  - HTTP/1.1 message format (rfc7231): 100 pages, 32k words
  - Buggy/incompatible implementations
- Framing
  - TCP does **not** provide framing
  - HTTP message:
    - Header followed by CR LF CR LF
    - … optionally followed by body, depending on message type
    - … whose length is specified in the Content-Length header
    - … unless Transfer-Encoding: chunked
    - … unless Content-Range is used
    - …

Binary Message Framing



0 4 8 12 16 20 24 28 32

3	w	w	w
13	x	y	z
i	n	d	u
s	t	r	i
e	s	3	c
o	m	0	

Message Encoding Standards

- Google Protocol Buffers
- JSON
- Apache Thrift Binary Protocol
- ASN.1

Example: Google Protocol Buffers

```
message Test1 {
  required int32 a = 1;
}
```

```
08 96 01
```

```
message Test2 {
  required string b = 2;
}
```

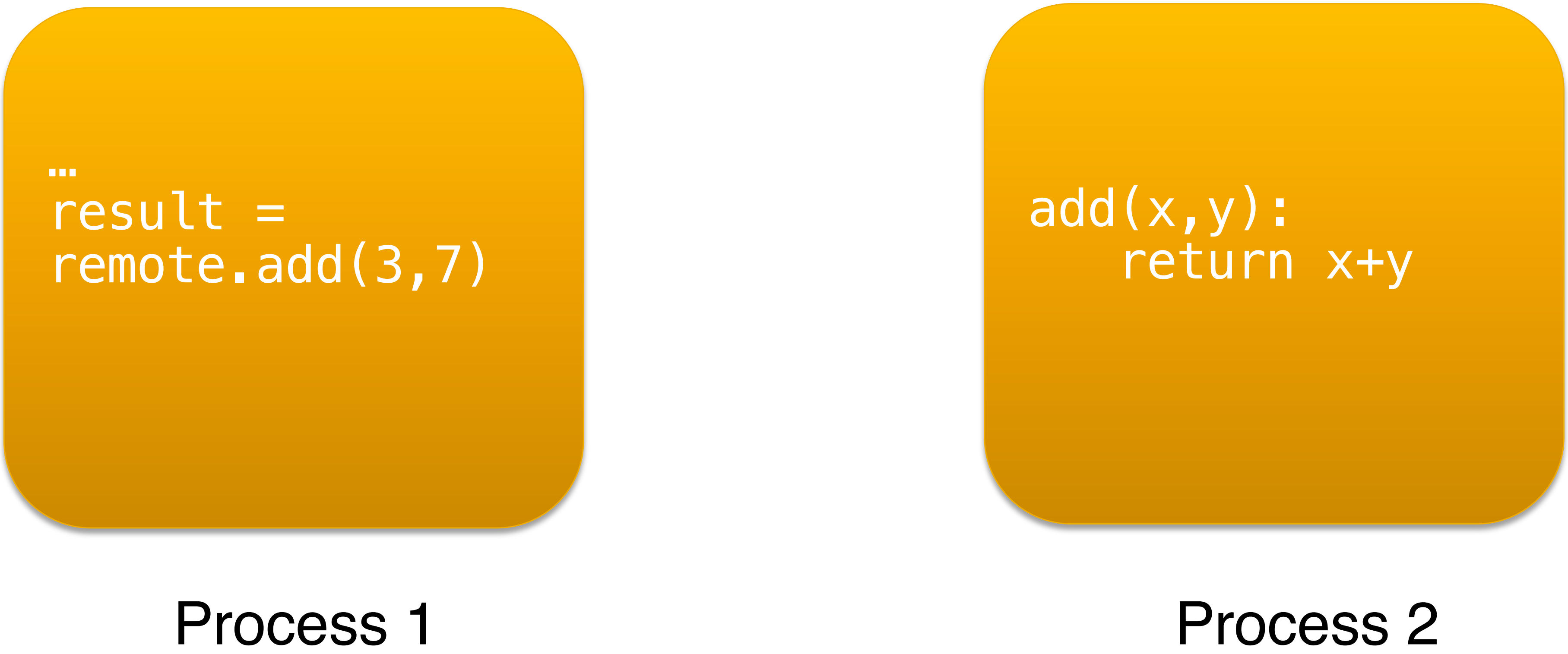
```
12 07 74 65 73 74 69 6e 67
       t  e  s  t  i  n  g
```

Protobuf code

```
syntax = "proto2";

package tutorial;

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default =
HOME];
  }

  repeated PhoneNumber phones = 4;
}

message AddressBook {
  repeated Person people = 1;
}
```

```
import addressbook_pb2

person = addressbook_pb2.Person()

person.id = 1234
person.name = "John Doe"
person.email = "jdoe@example.com"
phone = person.phones.add()
phone.number = "555-4321"
phone.type =
         addressbook_pb2.Person.HOME
```

Remote Procedure Calls



```
…
result =
remote.add(3,7)
```

```
add(x,y):
   return x+y
```

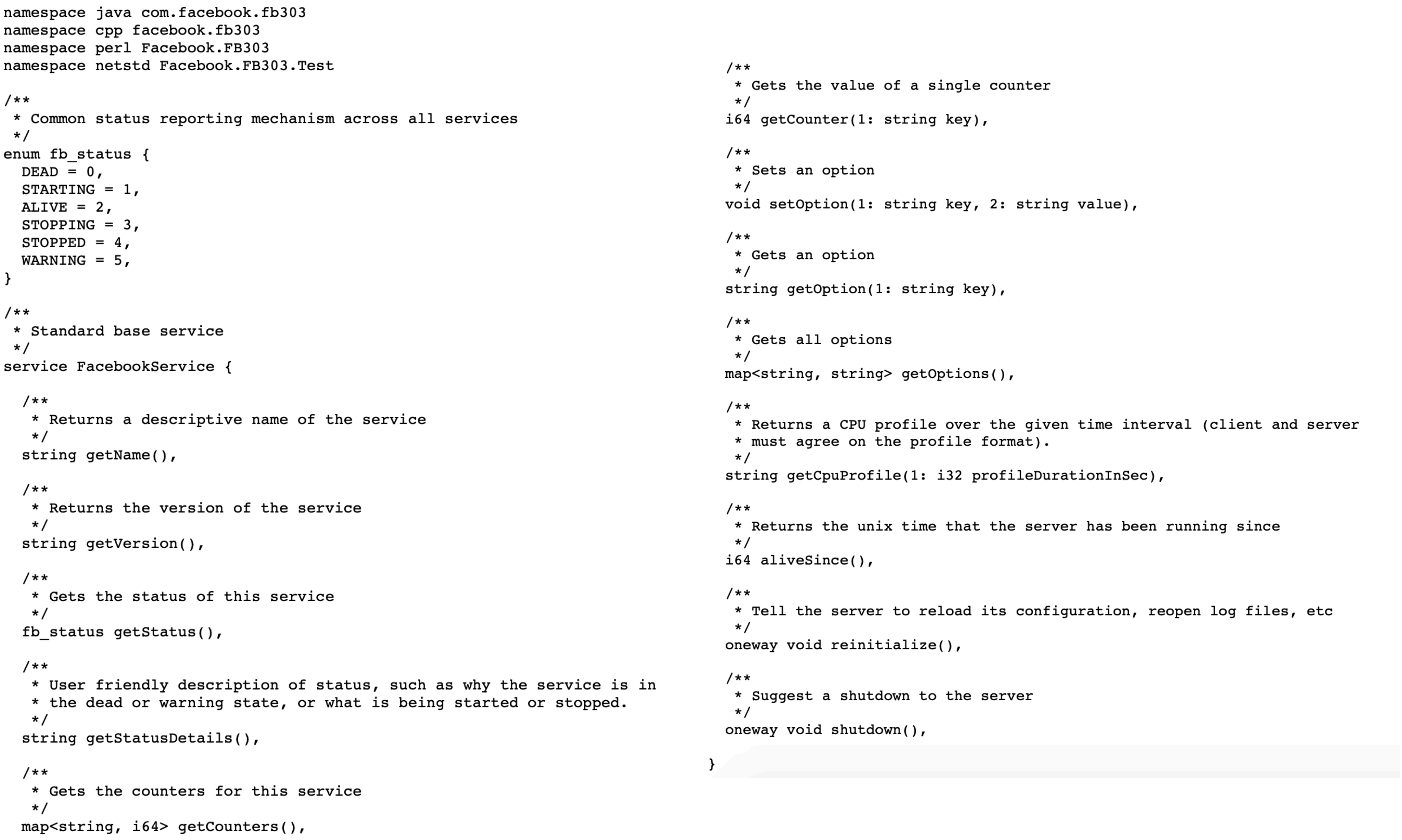
Process 1

Process 2

RPC issues

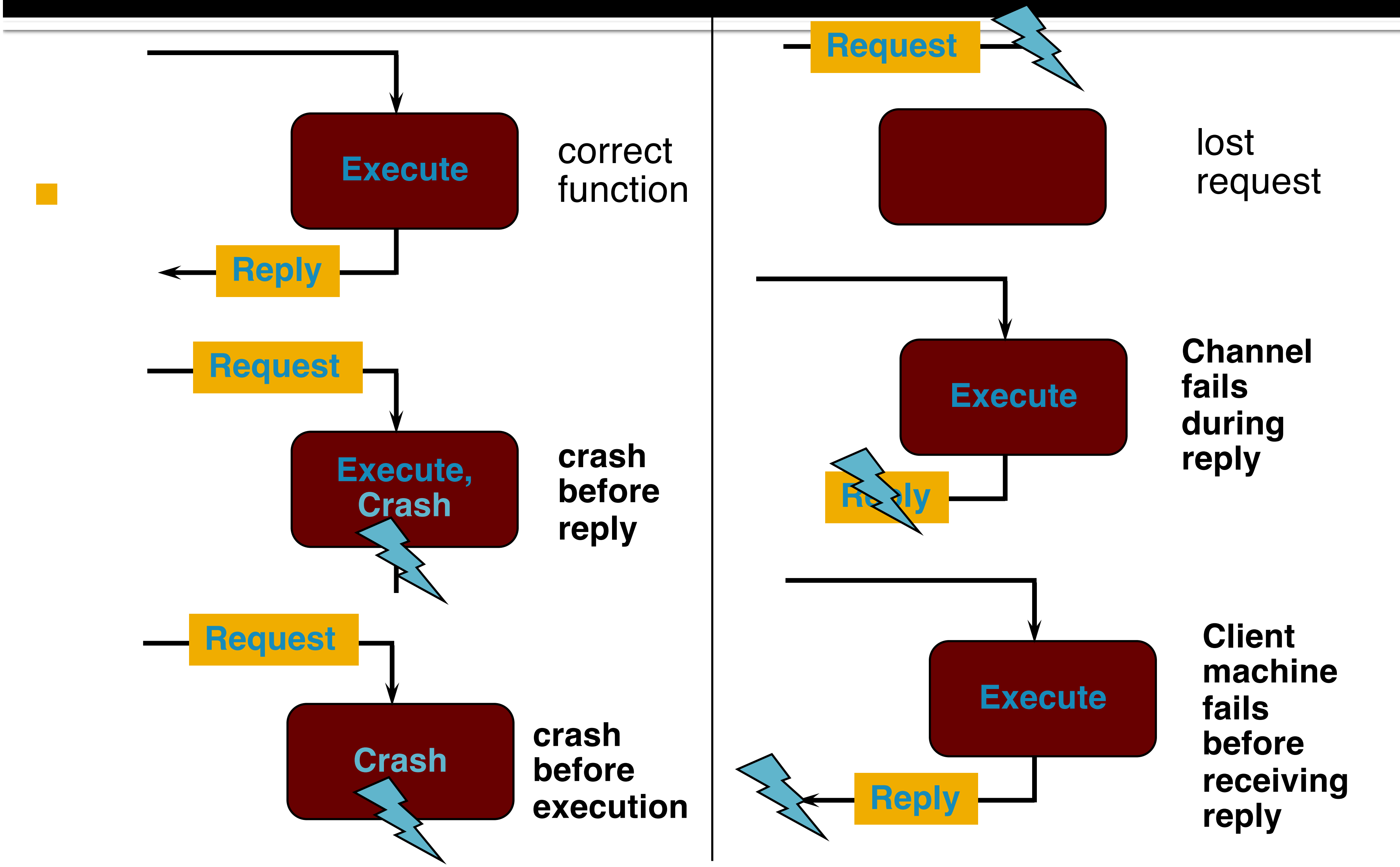
- Interface definition
  - Language-based
  - Polymorphic (E.g., Thrift)
- External data representation
  - Handle machine representation differences (e.g., byte order)
- Handle Failures

Thrift IDL



```
namespace java com.facebook.fb303
namespace cpp facebook.fb303
namespace perl Facebook.FB303
namespace netstd Facebook.FB303.Test

/**
 * Common status reporting mechanism across all services
 */
enum fb_status {
  DEAD = 0,
  STARTING = 1,
  ALIVE = 2,
  STOPPING = 3,
  STOPPED = 4,
  WARNING = 5,
}

/**
 * Standard base service
 */
service FacebookService {

  /**
   * Returns a descriptive name of the service
   */
  string getName(),

  /**
   * Returns the version of the service
   */
  string getVersion(),

  /**
   * Gets the status of this service
   */
  fb_status getStatus(),

  /**
   * User friendly description of status, such as why the service is in
   * the dead or warning state, or what is being started or stopped.
   */
  string getStatusDetails(),

  /**
   * Gets the counters for this service
   */
  map<string, i64> getCounters(),

  /**
   * Gets the value of a single counter
   */
  i64 getCounter(1: string key),

  /**
   * Sets an option
   */
  void setOption(1: string key, 2: string value),

  /**
   * Gets an option
   */
  string getOption(1: string key),

  /**
   * Gets all options
   */
  map<string, string> getOptions(),

  /**
   * Returns a CPU profile over the given time interval (client and server
   * must agree on the profile format).
   */
  string getCpuProfile(1: i32 profileDurationInSec),

  /**
   * Returns the unix time that the server has been running since
   */
  i64 aliveSince(),

  /**
   * Tell the server to reload its configuration, reopen log files, etc
   */
  oneway void reinitialize(),

  /**
   * Suggest a shutdown to the server
   */
  oneway void shutdown(),

}
```

Failure Modes of RPC



(and if request is received more than once?)

2020-04-01

Invocation Semantics

Whether or not to retransmit the request message until either a reply is received or the server is assumed to be failed

when retransmissions are used, whether to filter out duplicate requests at the server.

whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations

	Fault tolera measures			*Invocation semantics*
	Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
CORBA →	No	Not applicable	Not applicable	*Maybe*
Sun RPC →	Yes	No	(ok for *idempotent* operations) → Re-execute procedure	*At-least-once*
Java RMI, CORBA →	Yes	Yes	Retransmit old reply	*At-most-once*

Idempotent=same result if applied repeatedly, w/o side effects

Idempotent Operations

- Idempotent operations are those that can be repeated multiple times, without any side effects
- Examples (x is server-side variable)
  - x=1;
  - x=(argument) y;
- Non-examples
  - x=x+1;
  - x=x*2
- Idempotent operations can be used with at-least-once semantics

RMI / Distributed Objects

- Remote Method Invocation
  - Call a *method* on a remote object
- Incorporate remote object *references*
  - RPC generally uses call-by-value

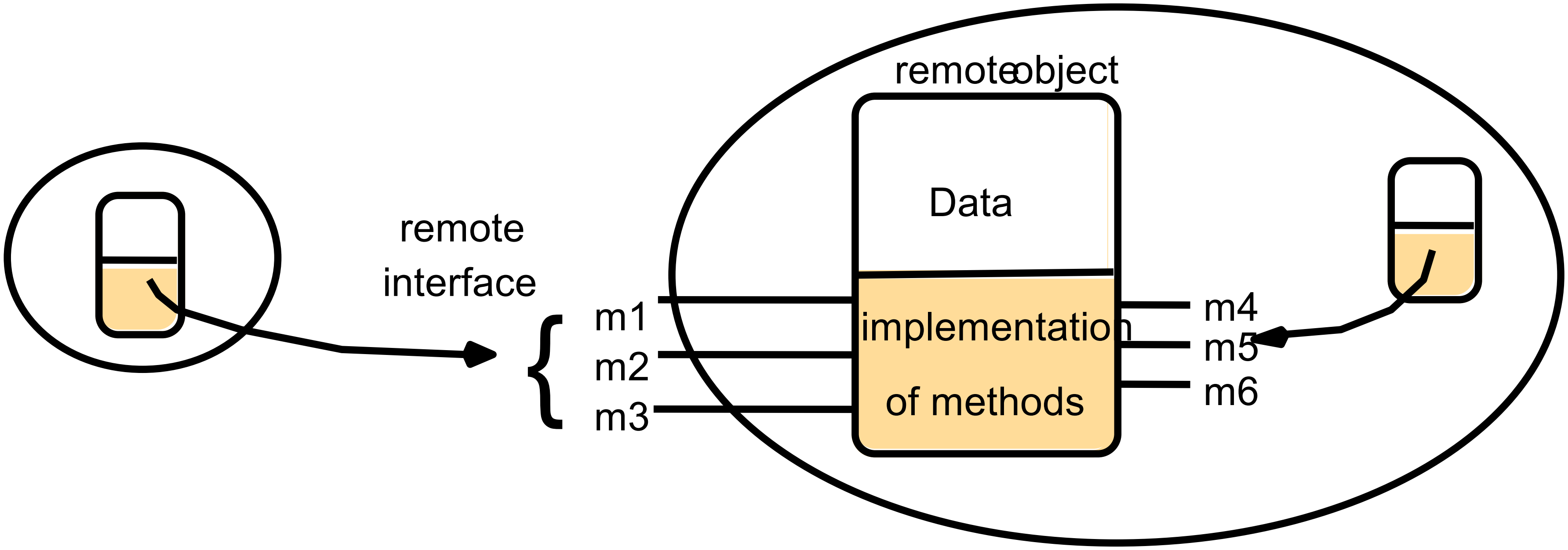
Local Objects

- Within one process’s address space
- Object
    - consists of a set of data and a set of methods.
    - E.g., C++/Java object
- Object reference
    - an identifier via which objects can be accessed.
    - i.e., a pointer (C++)
- Interface
    - Signatures of methods
        - Types of arguments, return values, exceptions
    - No implementation
    - E.g., hash table:
        - insert(key, value)
        - value = get(key)
        - remove(key)

Remote Objects

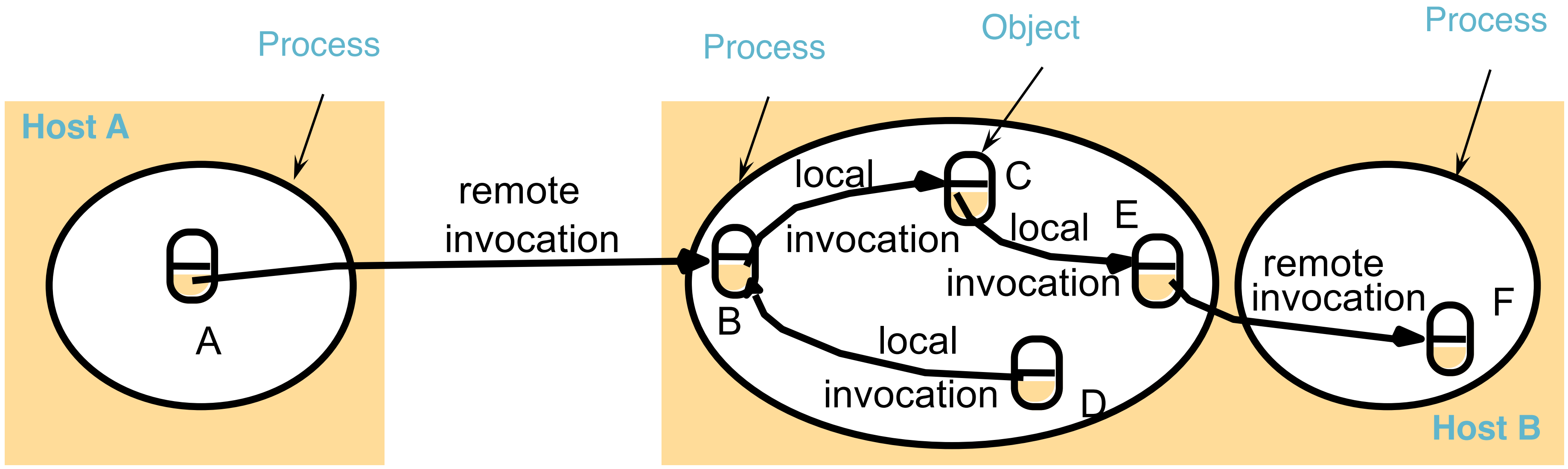
- May cross multiple process's address spaces
- Remote method invocation
  - method invocations between objects in different processes (processes may be on the same or different host).
  - *Remote Procedure Call (RPC): procedure call between functions on different processes in non-object-based system*
- Remote objects
  - objects that can receive remote invocations.
- Remote object reference
  - an identifier that can be used globally throughout a distributed system to refer to a particular unique remote object.
- Remote interface
  - Every remote object has a remote interface that specifies which of its methods can be invoked remotely. E.g., CORBA interface definition language (IDL).

A Remote Object and Its Remote Interface



Example Remote Object reference=(IP,port,objectnumber,signature,time)

[redacted]



Local invocation=between objects on same process.

Has *exactly once* semantics

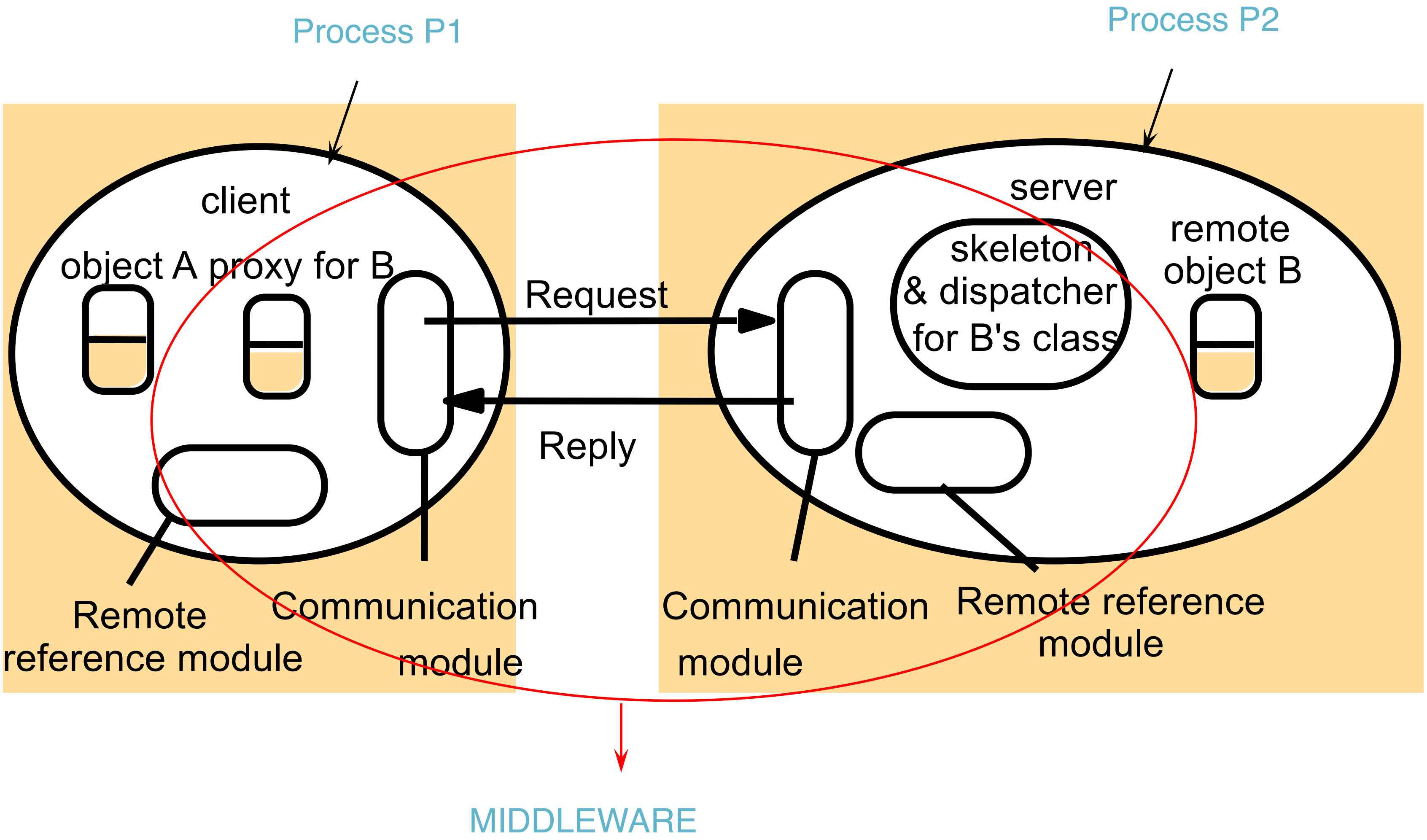
Remote invocation=between objects on different processes.

Ideally also want *exactly once* semantics for remote invocations

But difficult (why?)

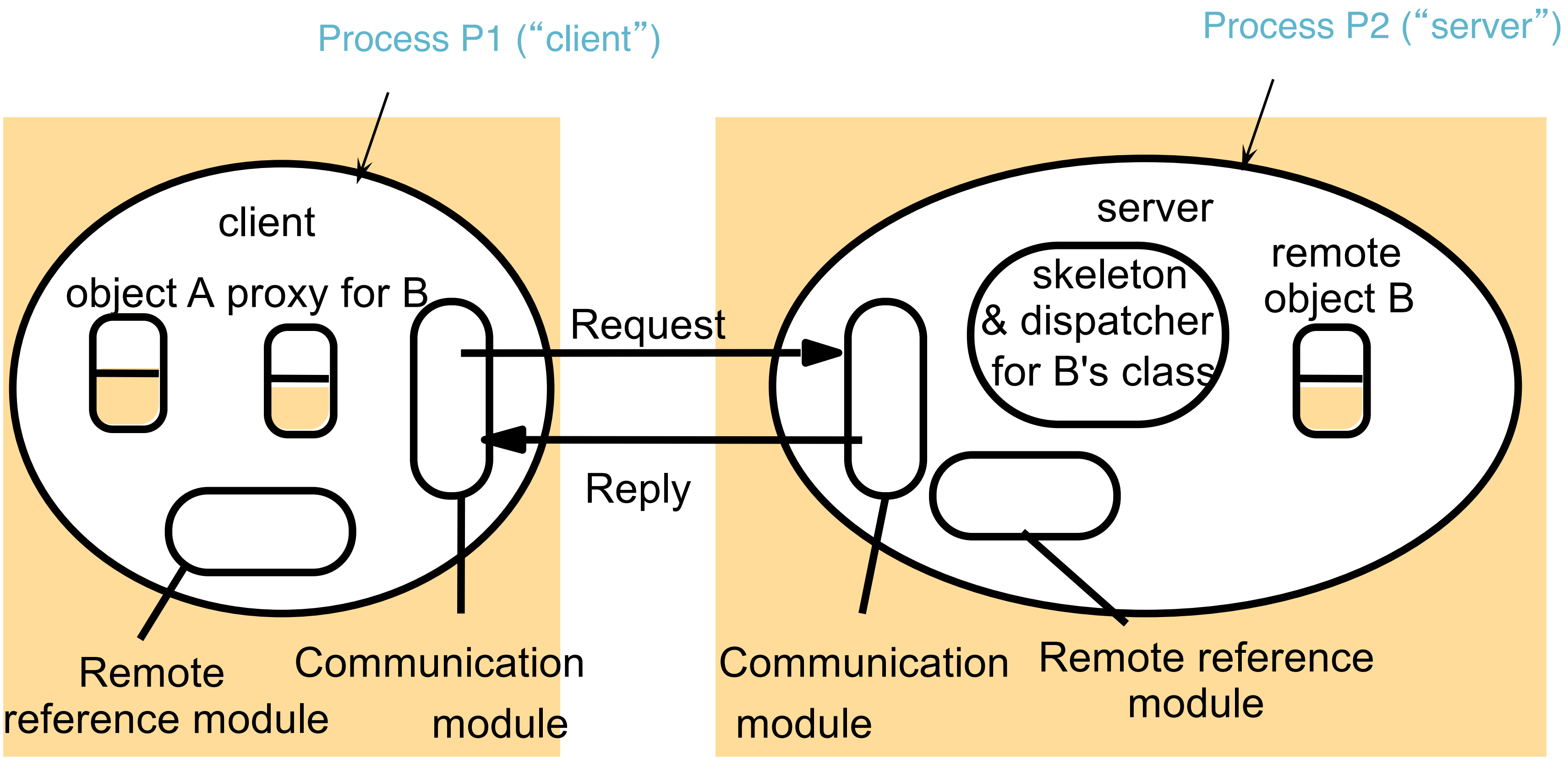
2020-04-01

Proxy and Skeleton in Remote Method Invocation



2020-04-01

Proxy and Skeleton in Remote Method Invocation



Proxy

- Provides *transparency* by behaving like a local object to the invoker
  - The proxy “implements” the methods in the interface of the remote object that it represents. But,…
- Instead of executing an invocation, the proxy forwards it to a remote object
  - Marshals a request message
    - Target object reference
    - Method ID
    - Argument values
  - Sends request message
  - Unmarshals reply and returns to invoker

Marshalling & Unmarshalling

- External data representation: an agreed, platform-independent, standard for the representation of data structures and primitive values.
  - CORBA Common Data Representation (CDR)
  - Sun's XDR
  - Google Protocol Buffers
- Marshalling: taking a collection of data items (platform dependent) and assembling them into the external data representation (platform independent).

- Unmarshalling: the process of disassembling data that is in external data representation form, into a locally interpretable form.

Example: JSON-RPC

REQUEST

```
{
        "jsonrpc": "2.0",
        "method": "subtract",
        "params": [42, 23],
        "id": 1
}
```

RESPONSE

```
{
        "jsonrpc": "2.0",
        "result": 19,
        "id": 1
}
```

Remote Reference Module

- Translates local and remote object references

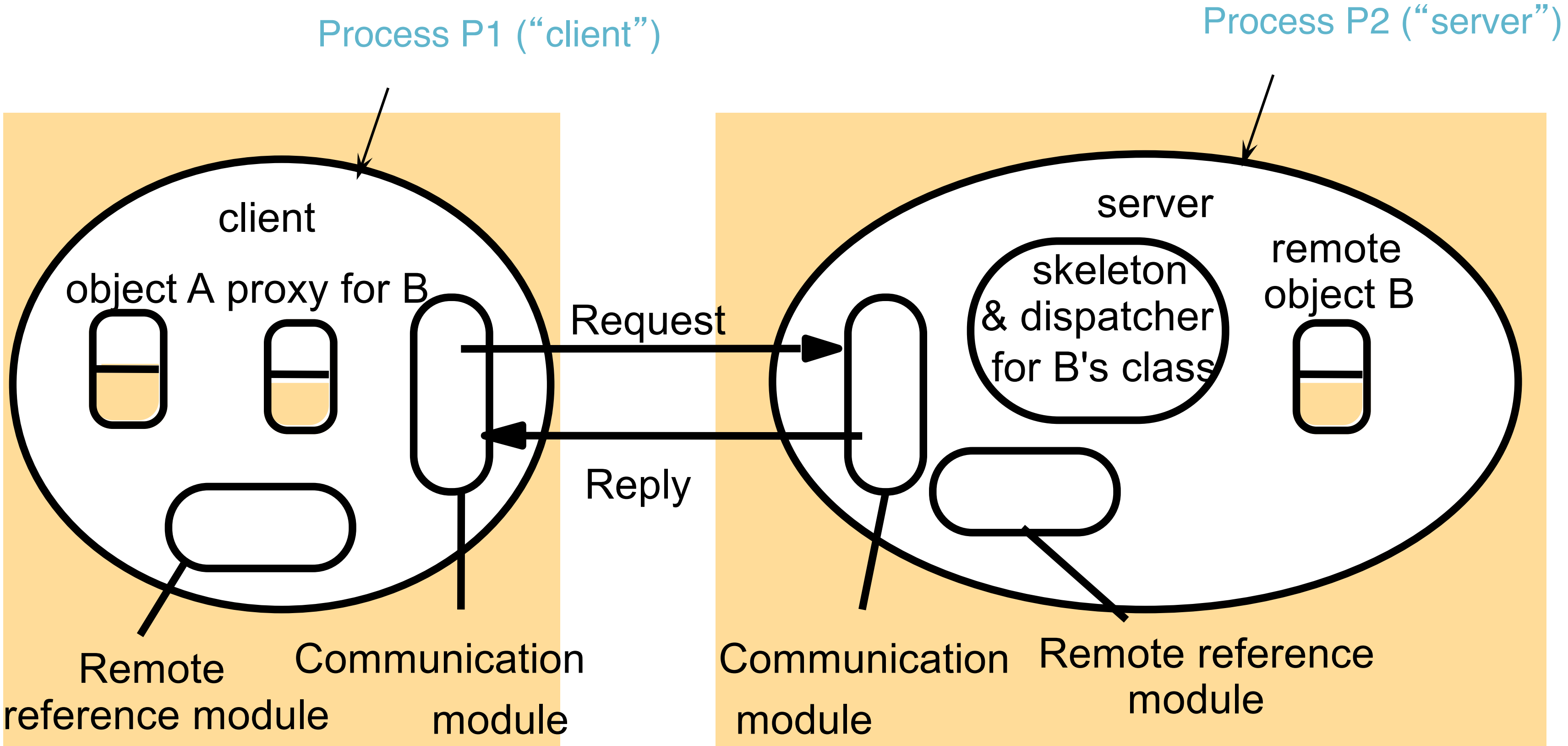
- Response:

```
{
    “postID” : 1234,
    “contents”: “What is on the
midterm”,
    “response”: {
            “objType”: “responseObject”,
            “objRef”: “12345”
    }
}
```

Remote Reference Module

- Remote object table
  - An entry for each remote object held by any process. E.g., B at P2.
  - An entry for each local proxy. E.g., proxy-B at P1.
- RRM looks up **remote object references** inside *request* and *reply* messages in table
  - If reference **not** in table, create a new **proxy** and add it to the table
  - Then (in either case), replace **reference** by **proxy** found in table

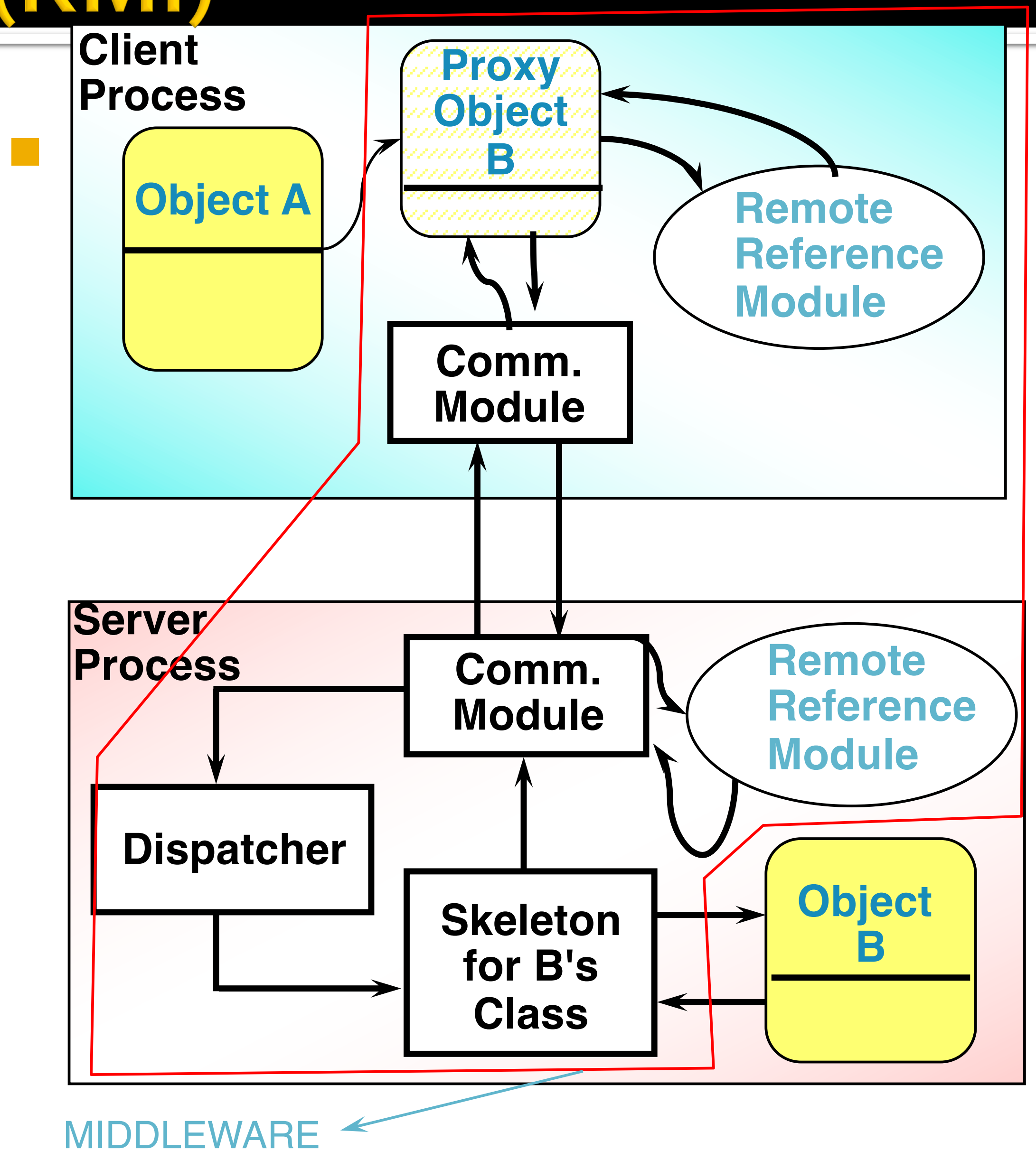
Proxy and Skeleton in Remote Method Invocation



What about Server Side? Dispatcher and Skeleton

- Each process has one **dispatcher**, and a **skeleton** for each local object (actually, class)
- The **dispatcher** receives all *request* messages from the communication module.
  - Uses the **method id** to select the appropriate method in the appropriate **skeleton**, passing on the *request* message.
- **Skeleton** "implements" the methods in the remote interface.
  - **Un-marshals** the arguments in the *request* message and invokes the corresponding method in the remote object (the actual object).
  - It waits for the invocation to complete and **marshals** the result, together with any exceptions, into a *reply* message.

Summary of Remote Method Invocation (RMI)



Proxy object is a hollow container of Method names.

Remote Reference Module translates between local and remote object references.

Dispatcher sends the request to Skeleton Object

Skeleton unmarshals parameters, sends it to the object, & marshals the results for return

Generation of Proxies, Dispatchers and Skeletons

- Programmer only writes object **implementations** and **interfaces**
  - E.g., CORBA: programmer specifies interface in CORBA IDL
  - E.g., Java RMI: programmer defines set of remote object methods as a Java interface
- Proxies, dispatchers, skeletons generated automatically from the specified **interfaces**
  - Compiler to generate code
  - Can be polymorphic (multiple languages)

Summary

- Local objects vs. Remote objects
- RPCs and RMIs
- RMI: invocation, proxies, skeletons, dispatchers