Remote Procedure Call (RPC)

Web Applications and Services

Spring term

Naercio Magaia

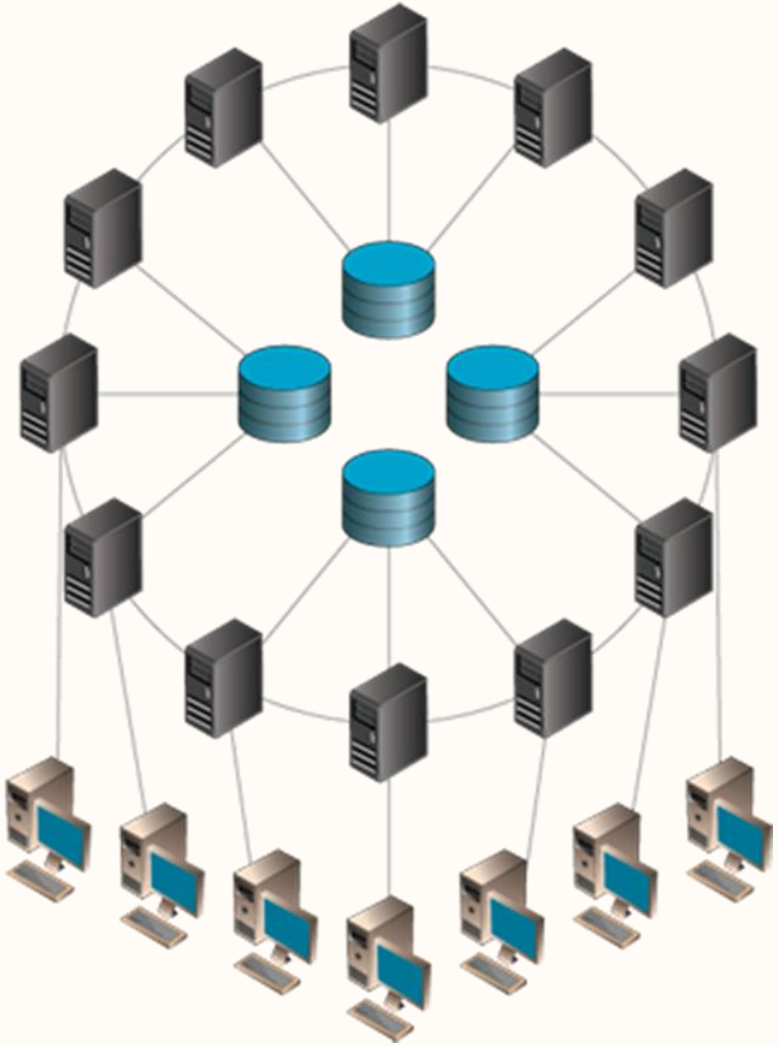
UNIVERSITY OF SUSSEX

Contents

UNIVERSITY OF SUSSEX

Distributed systems

- It is a group of computers working together as to appear as a single computer to the end-user
- These machines have a shared state, operate concurrently
- They communicate via a network to achieve a task
- Networked computers communicate and coordinate their actions only via messages passing



UNIVERSITY OF SUSSEX

Why distributed systems?

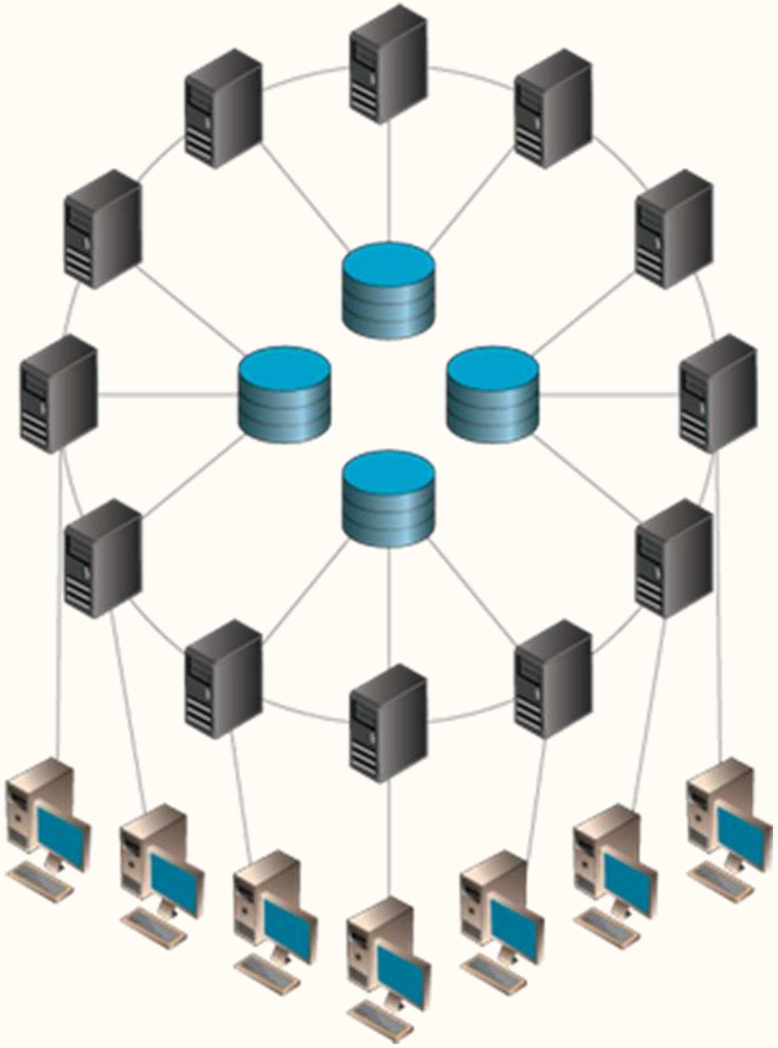
- The desire to share resources
- For scalability, performance and high availability of applications
- Some examples include
  - web search
  - distributed information
  - network file systems
  - real-time process control
  - parallel computation

Benefits of distributed systems

- Computation speedup, better performance
  - Parallel computation on multiple servers
  - Solve bigger problems
- Redundancy, reliability, fault-tolerant
  - Several machines can provide the same services, so if one is unavailable, the work does not stop.
- Many applications are inherently distributed

Challenges

- The network may fail
- Processes may crash
- Writing a program to run on a single computer is comparatively easy?
- How can we coordinate the processes of these machines?



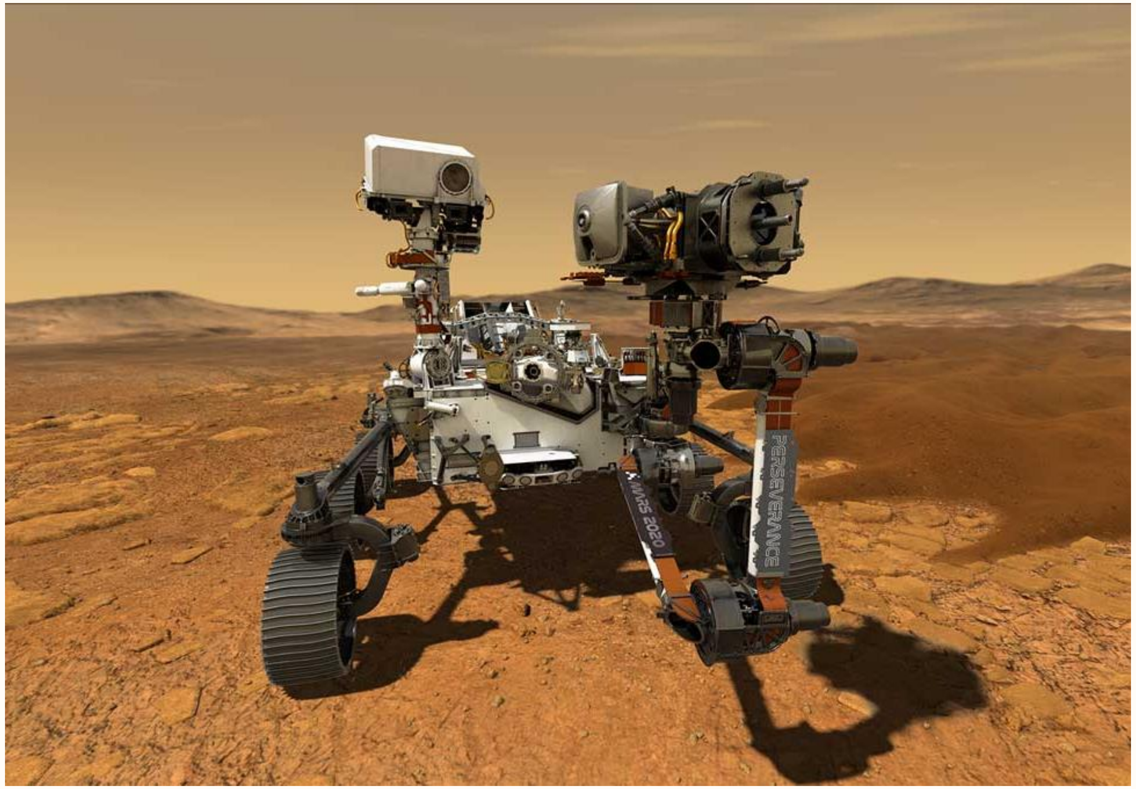
UNIVERSITY OF SUSSEX

Programming

- The goal of making the programming of distributed systems look similar, if not identical, to conventional programming

Examples

The black hole image



Perseverance on Mars

UNIVERSITY
OF SUSSEX

Local Procedure Call

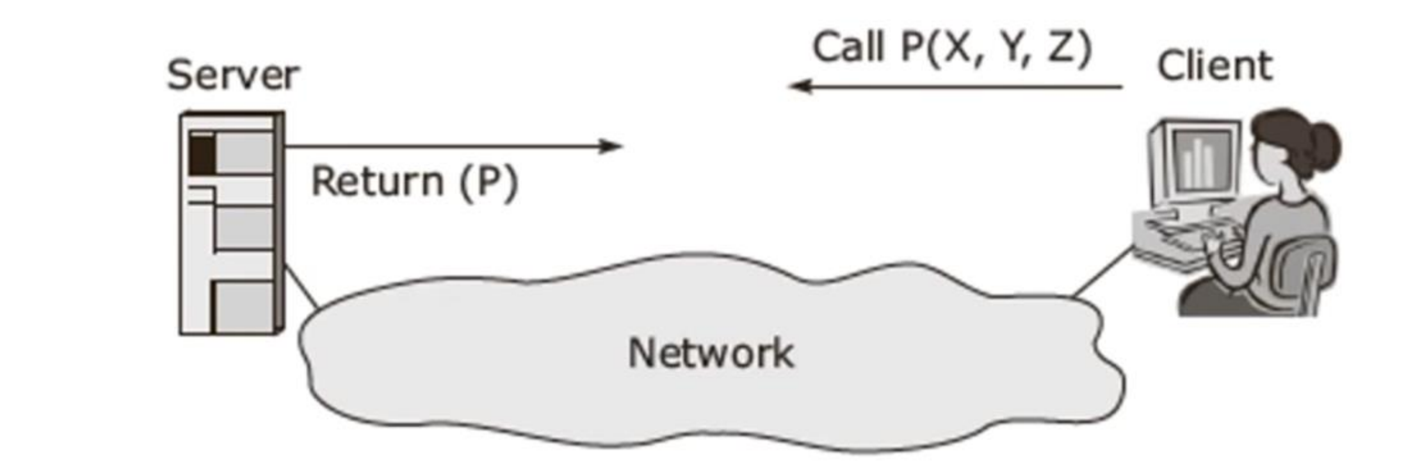
- Everything in the same address space
- Calls cannot fail (i.e., they can but programmers have full control)

```
ret = foo(64, "string", &myStruct);
```

- Push arguments, local variables and return address in the stack
- Call `foo`
- Pop everything out of the stack and put `ret` to a register
- Continue with the next call
- What about parameter passing?

Remote Procedure Call (RPC)

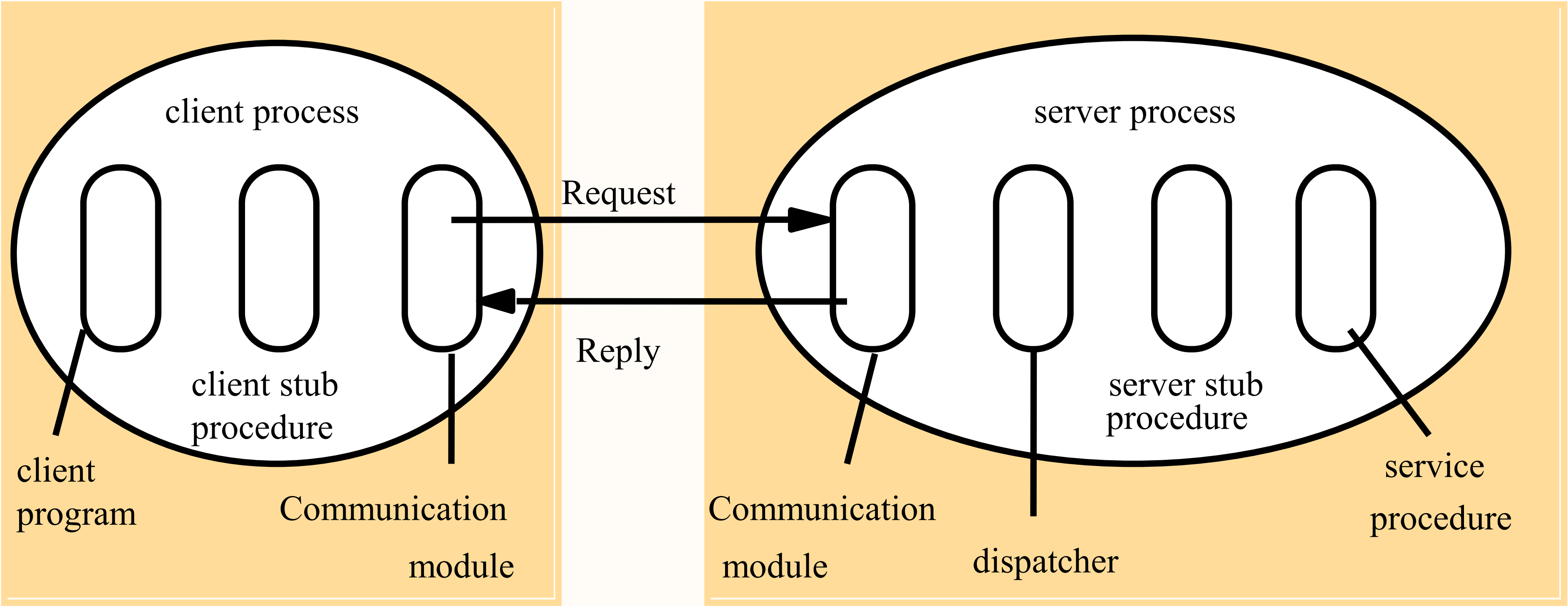
- Makes a call to a remote function look the same as a local function call



- In practice,
  - What if the service crashes during the function call?
  - What if a message is lost?
  - What if a message is delayed?
  - If something goes wrong, is it safe to retry?

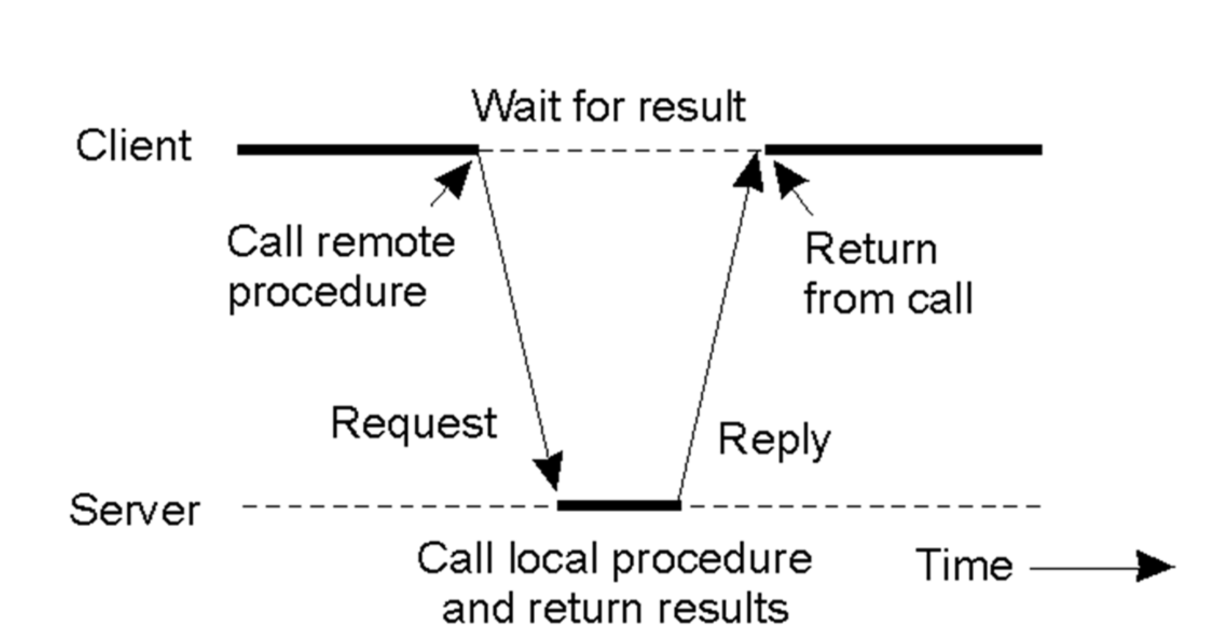
UNIVERSITY OF SUSSEX

RPC at a Glance



RPC

- A process on machine A can call a procedure on machine B
- The process on A is suspended and execution continues on B
- When B returns, the return value is passed to A, and it continues its execution.

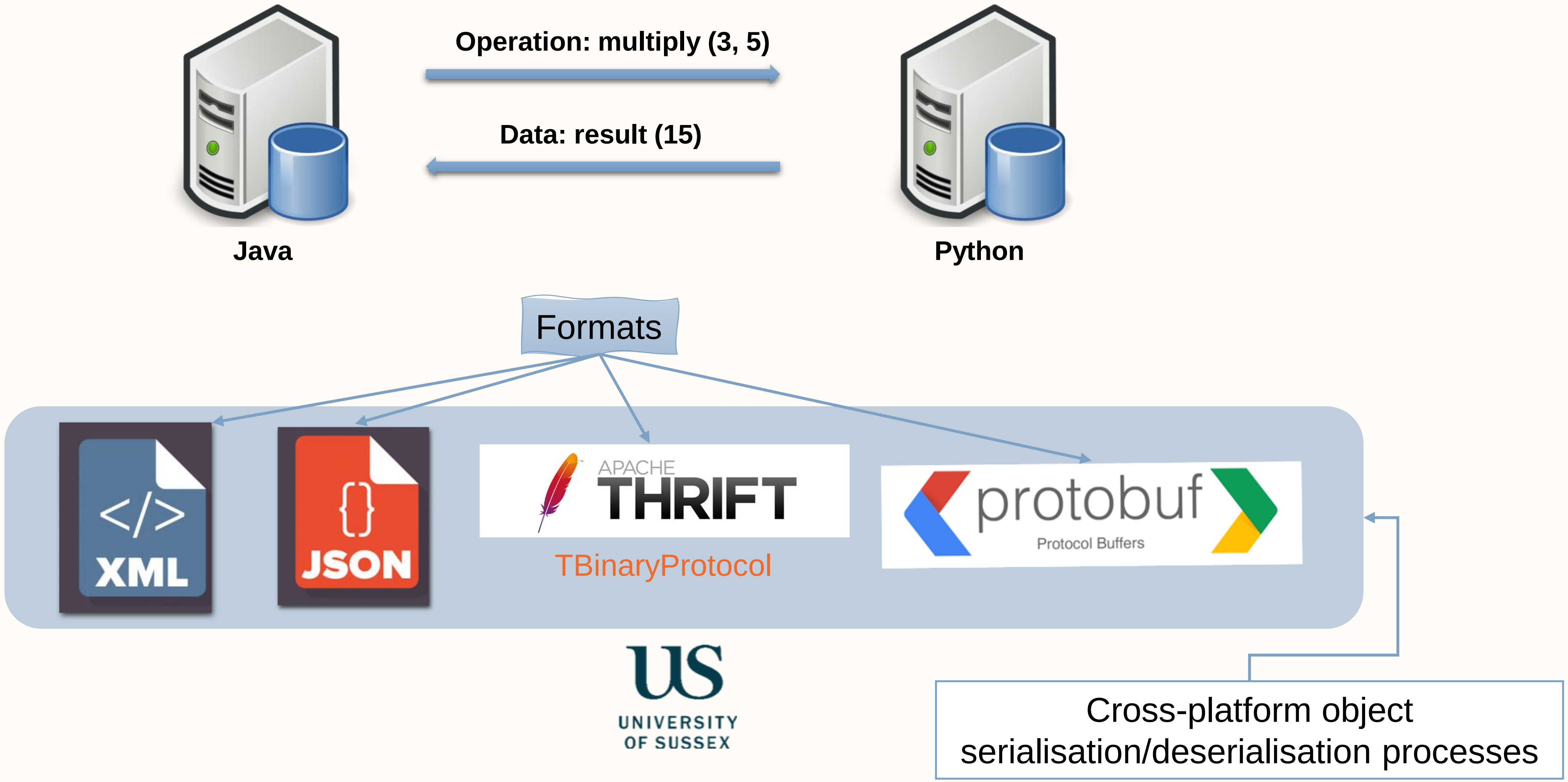


UNIVERSITY OF SUSSEX

RPC history

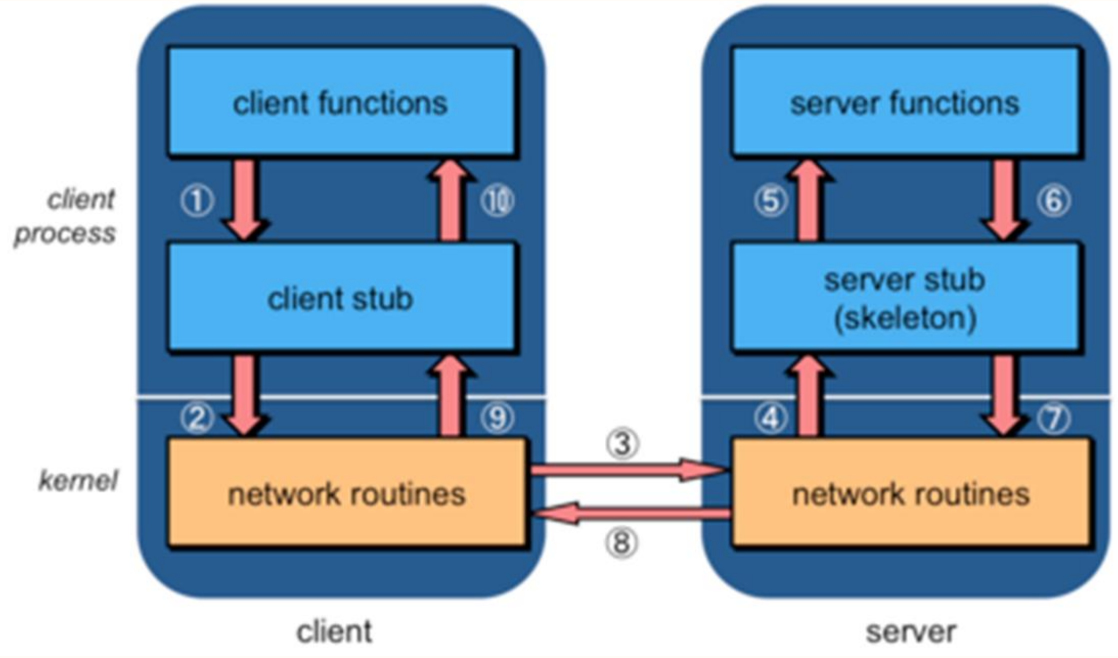
- SunRPC/ONC RPC (1980s, basis for NFS)
- CORBA: object-oriented middleware, hot in the 1990s
- Microsoft’s DCOM and Java RMI (similar to CORBA)
- SOAP/XML-RPC: RPC using XML and HTTP (1998)
- Thrift (Facebook, 2007)
- gRPC (Google, 2015)
- REST (often with JSON)
- …

RPC


Operation: multiply (3, 5)
Data: result (15)
Java
Python
Formats
XML
JSON
APACHE THRIFT
TBinaryProtocol
protobuf
Protocol Buffers
Cross-platform object serialisation/deserialisation processes

RPC : Client side

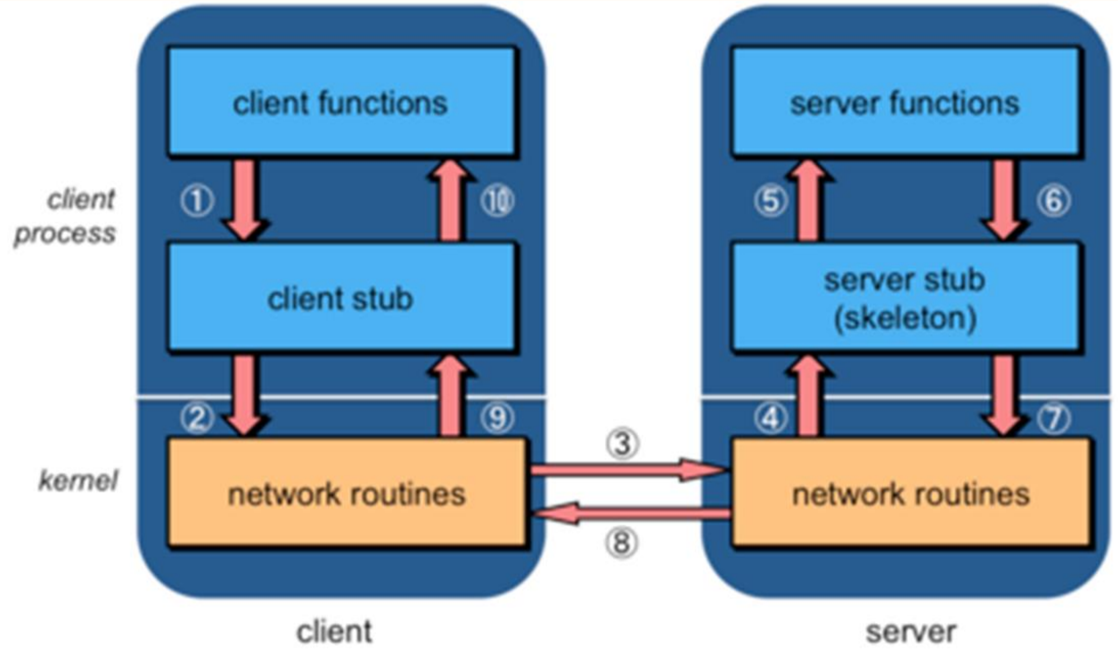
- Emulate the behaviour of local calls (transparently)
- Stub function
  - the same signature as if it was called locally
  - marshals parameters
  - sends request message
  - waits for a response from the server
  - un-marshals the response & returns the appropriate data



UNIVERSITY OF SUSSEX

RPC : Server side

- Dispatcher
  - Receives client requests
  - Identifies appropriate function to call
- Skeleton
  - Un-marshals parameters
  - Calls the local function
  - Marshals the response & sends it back to the dispatcher



UNIVERSITY OF SUSSEX

Parameter passing

- Passing by value (is simple)
  - Just copy the value into the network message
- Passing by reference (is hard)
  - It makes no sense to pass an address to a remote machine
  - A memory location in a process on one machine is meaningless to another process on another machine

Parameter passing

1. Copy items referenced to message buffer
2. Ship them over
3. Un-marshal data at server
4. Pass local pointer to server stub function
5. Send new values back

Representing data

- Local Calls
  - Primitive types have the same exact representation
- But, for Remote Calls
  - Different byte ordering
    - Big Endian – most significant byte is stored in the smallest address
    - Little Endian - least significant byte is stored in the smallest address
  - Different sizes of integers and other types
  - Different floating-point representations
  - Different Languages!!

Representing data

- We need standards!
  - SunRPC uses eXternal Data Representation (XDR)
  - Abstract Syntax Notation (ASN.1)
  - WSDL for Web Service
  - Google Protocol Buffers
  - JSON
- Implicit typing
  - only values are transmitted, not data types or parameter info
- Explicit typing
  - Type is transmitted with value

UNIVERSITY OF SUSSEX

Interface Definition Language

- Allow programmer to specify remote procedure interfaces (i.e., names, parameters, return values)
- Pre-compiler can use this to generate
  - client and server stubs
  - Marshalling code
  - Un-marshalling code
  - Network transport routines

RPC Semantics

- Local Calls
  - exactly Once!
- But RPC involves networking and a remote host
  - delays, server failures
- A remote procedure call may be called
  - never: server crashed, or server process died before executing server code
  - once: everything worked well, as expected
  - more than once: Why?

RPC Semantics

- RPC systems may offer either
  - *maybe* semantics
  - *at least once* semantics
  - *at most once* semantics
- How can these be implemented?
- What are the ramifications with respect to
  - idempotent functions?
  - non-idempotent functions?

Protocol Buffers (Protobuf)

- Protobuf is Google’s language-neutral, platform-neutral, extensible mechanism for serializing structured data for
  - communications protocols
  - data storage
  - specification of services
- Think XML, but smaller, faster, and simpler
- It currently support generated code in Java, Python, Objective-C, and C++
  - With the new proto3 language version, it is possible to also work with Kotlin, Dart, Go, Ruby, PHP, and C#

protobuf
Protocol Buffers

Protocol Buffers

- Why not XML?
  - Protobuf is simpler, i.e., 3 to 10 times smaller and 20 to 100 times faster
  - It generate data access classes that are easier to use programmatically (should we write source code and auto-produce description/IDL or vice versa?)

```
<person>
    <name>John Doe</name>
    <email>jdoes@example.com</email>
</person>
```

```
# Textual representation of a protocol buffer
# This is *not* the binary format used on thr wire
person {
    name: "John Doe"
    email: "jdoes@example.com"
}
```

- Many projects use protocol buffers, including the following: [gRPC](), [Google Could](), [Envoy Proxy]()

Protocol Buffers : RPC

- The protocol compiler will read .proto files and generate
  - an abstract interface called `SearchService` (must be implemented!)
  - a corresponding “stub” implementation
- The stub forwards all calls to an RpcChannel
- an abstract interface that must be defined for RPC system

```
message SearchRequest {
    required string query = 1;
    optional int32 page_number = 2;
    optional int32 result_per_page = 3;
}

message SearchResponse {
    ...
}
```

```
service SearchService {
    rpc Search(SearchRequest) returns (SearchResponse);
}
```

US
UNIVERSITY
OF SUSSEX

Apache Thrift

- A software framework for scalable cross-language services development
- Combines a software stack with a code generation engine
- Builds services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages
- Type system consists of pre-defined base types, user-defined structs, container types, exceptions and service definitions

UNIVERSITY OF SUSSEX

APACHE THRIFT

Thrift Types

- Base Types
  - bool: A boolean value (true or false)
  - byte: An 8-bit signed integer
  - i16: A 16-bit signed integer
  - i32: A 32-bit signed integer
  - i64: A 64-bit signed integer
  - double: A 64-bit floating point number
  - string: A text string encoded using UTF-8 encoding
- Special Types
  - binary: a sequence of unencoded bytes
- Containers
  - list: An ordered list of elements
  - set: An unordered set of unique elements
  - map<type1,type2>: A map of strictly unique keys to values
- Container elements may be of any valid Thrift Type.

Thrift Types

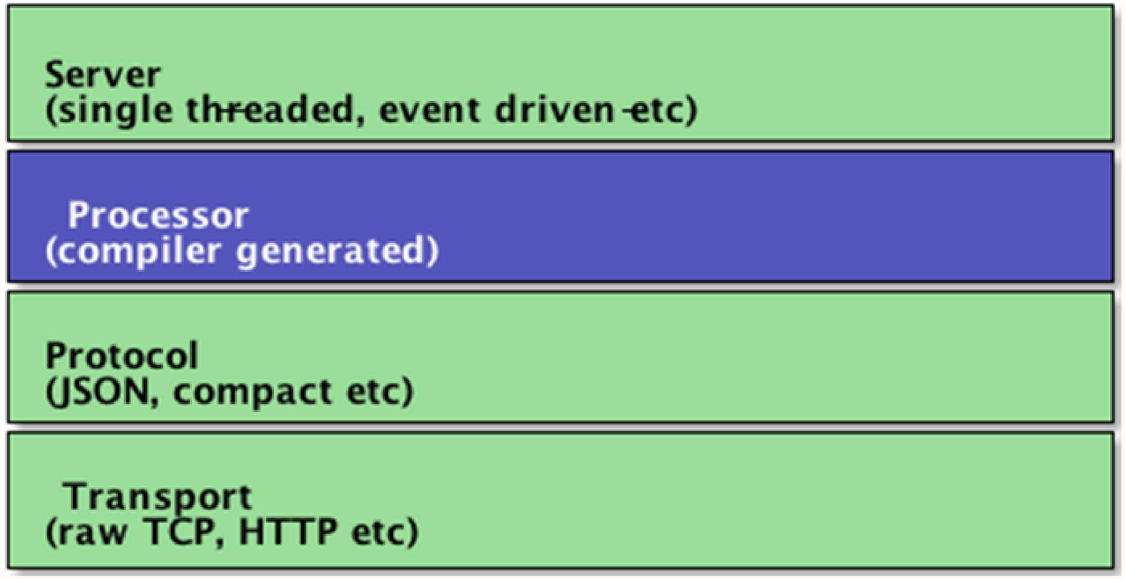
- Structs
  - define a common object – they are essentially equivalent to classes in object-oriented programming (OOP) languages, but without inheritance
  - A struct has a set of strongly typed fields, each with a unique name identifier
  - Fields may have various annotations (numeric field IDs, optional default values, etc.) that are described in the Thrift IDL
- Exceptions
  - are functionally equivalent to structs, except that they inherit from the native exception base class

Thrift Types

- Services
  - Are defined using Thrift types
  - Definition of a service is semantically equivalent to defining an interface (or a pure virtual abstract class) in OOP
  - The Thrift compiler generates fully functional client and server stubs that implement the interface
  - A service consists of a set of named functions, each with a list of parameters and a return type

Thrift network stack

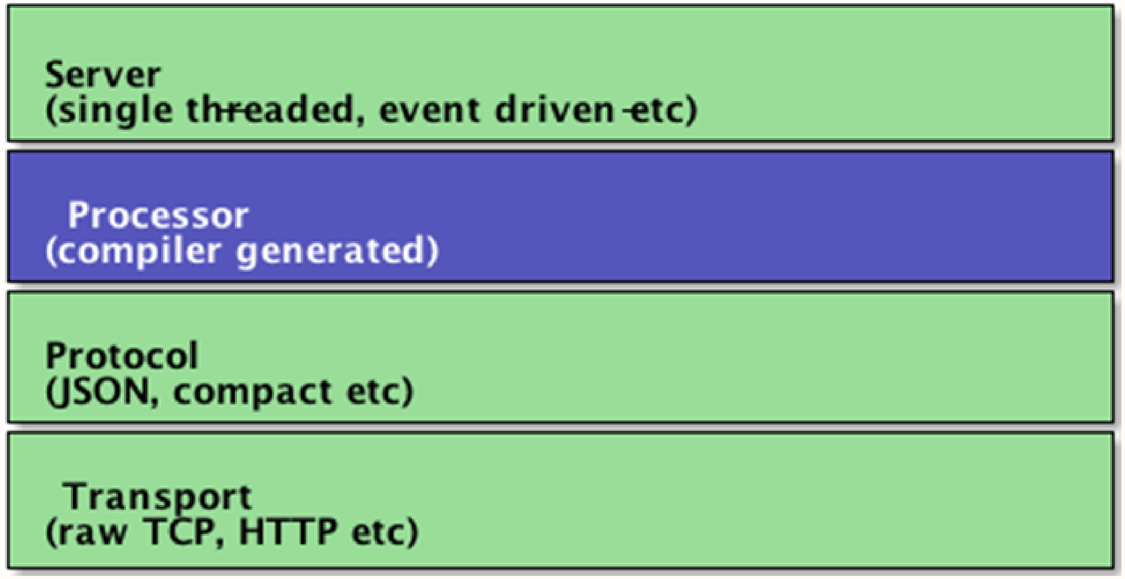
- Transport
  - Provides a simple abstraction for reading/writing from/to the network
  - Enables Thrift to decouple the underlying transport from the rest of the system (serialization/deserialization, for instance)
- Protocol
  - Defines a mechanism to map in-memory data structures to a wire-format
    - specifies how datatypes use the underlying Transport to encode/decode themselves
  - Some examples of protocols in this sense include JSON, XML, plain text, compact binary, etc.



UNIVERSITY OF SUSSEX

Network Stack

- Processor
  - Encapsulates the ability to read data from input streams and write to output streams
  - The input and output streams are represented by Protocol objects
- Server
  - Create a transport
  - Create input/output protocols for the transport
  - Create a processor based on the input/output protocols
  - Wait for incoming connections and hand them off to the processor



UNIVERSITY
OF SUSSEX

Thrift definition file

```
service Calculator extends shared.SharedService {

  /**
   * A method definition looks like C code. It has a return type, arguments,
   * and optionally a list of exceptions that it may throw. Note that argument
   * lists and exception lists are specified using the exact same syntax as
   * field lists in struct or exception definitions.
   */

   void ping(),

   i32 add(1:i32 num1, 2:i32 num2),

   i32 calculate(1:i32 logid, 2:Work w) throws (1:InvalidOperation ouch),

   /**
    * This method has a oneway modifier. That means the client only makes
    * a request and does not listen for any response at all. Oneway methods
    * must be void.
    */
   oneway void zip()

}
```

Method definitions can be terminated using comma or semi-colon

void is a valid return type for functions

Arguments can be primitive types or structs, likewise for return types

Next Lecture ...

- ✓ Introduction
- ✓ HTTP, Caching, and CDNs
- ✓ Views
- ✓ Templates
- ✓ Forms
- ✓ Models
- ✓ Security
- ✓ Transactions
- ✓ Remote Procedure Call
- ➢ **Web Services**
- RESTful Services
- Time
- Elections/Group Communication
- Zookeeper

UNIVERSITY OF SUSSEX