Distributed Systems

The second half of *Concurrent and Distributed Systems*

https://www.cl.cam.ac.uk/teaching/current/ConcDisSys

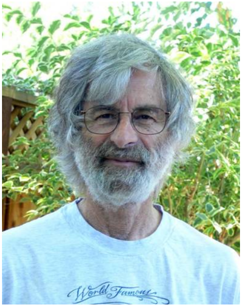
Dr. Martin Kleppmann (mk428@cam)

University of Cambridge

Computer Science Tripos, Part IB

A distributed system is. . .

- *“. . . a system in which the failure of a computer you didn’t even know existed can render your own computer unusable.”* — Leslie Lamport



A distributed system is...

- *"... a system in which the failure of a computer you didn't even know existed can render your own computer unusable."* — Leslie Lamport
- ... multiple computers communicating via a network...
- ... trying to achieve some task together
- Consists of "nodes" (computer, phone, car, robot, ...)

Recommended reading

- van Steen & Tanenbaum.
  "**Distributed Systems**"
  (any ed), free ebook available
- Cachin, Guerraoui & Rodrigues.
  "**Introduction to Reliable and Secure Distributed Programming**" (2nd ed), Springer 2011
- Kleppmann.
  "**Designing Data-Intensive Applications**",
  O'Reilly 2017
- Bacon & Harris.
  "**Operating Systems: Concurrent and Distributed Software Design**", Addison-Wesley 2003

Relationships with other courses

- **Concurrent Systems** – Part IB
  (every distributed system is also concurrent)
- **Operating Systems** – Part IA
  (inter-process communication, scheduling)
- **Databases** – Part IA
  (many modern databases are distributed)
- **Computer Networking** – Part IB Lent term
  (distributed systems involve network communication)
- **Further Java** – Part IB Michaelmas
  (distributed programming practical exercises)
- **Security** – Part IB Easter term
  (network protocols with encryption & authentication)
- **Cloud Computing** – Part II
  (distributed systems for processing large amounts of data)

Why make a system distributed?

Why make a system distributed?

- **It's inherently distributed:**
  e.g. sending a message from your mobile phone to your friend's phone

Why make a system distributed?

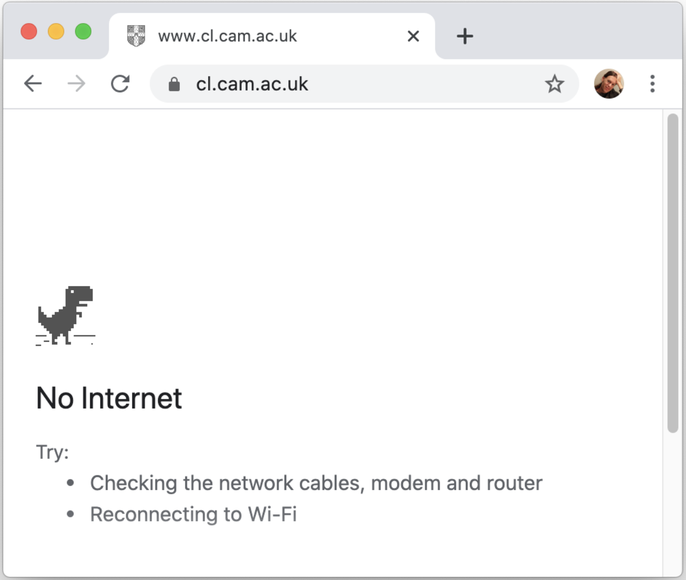
- **It's inherently distributed:**
  e.g. sending a message from your mobile phone to your friend's phone
- **For better reliability:**
  even if one node fails, the system as a whole keeps functioning

Why make a system distributed?

- **It's inherently distributed:**
  e.g. sending a message from your mobile phone to your friend's phone
- **For better reliability:**
  even if one node fails, the system as a whole keeps functioning
- **For better performance:**
  get data from a nearby node rather than one halfway round the world

Why make a system distributed?

- **It's inherently distributed:**
  e.g. sending a message from your mobile phone to your friend's phone
- **For better reliability:**
  even if one node fails, the system as a whole keeps functioning
- **For better performance:**
  get data from a nearby node rather than one halfway round the world
- **To solve bigger problems:**
  e.g. huge amounts of data, can't fit on one machine


www.cl.cam.ac.uk
cl.cam.ac.uk
No Internet
Try:
Checking the network cables, modem and router
Reconnecting to Wi-Fi

Why NOT make a system distributed?

The trouble with distributed systems:

- Communication may fail (and we might not even know it has failed).
- Processes may crash (and we might not know).
- All of this may happen nondeterministically.

Why NOT make a system distributed?

The trouble with distributed systems:

- Communication may fail (and we might not even know it has failed).
- Processes may crash (and we might not know).
- All of this may happen nondeterministically.

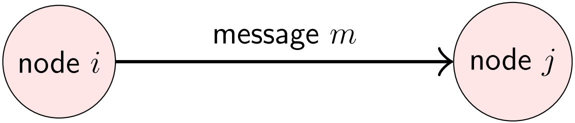
Fault tolerance: we want the system as a whole to continue working, even when some parts are faulty.

This is hard.

Writing a program to run on a single computer is comparatively easy?!

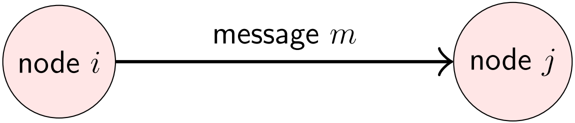
Distributed Systems and Computer Networking

We use a simple abstraction of communication:



Distributed Systems and Computer Networking

We use a simple abstraction of communication:



Reality is much more complex:

- **Various network operators:**
  eduroam, home DSL, cellular data, coffee shop wifi, submarine cable, satellite. . .
- **Physical communication:**
  electric current, radio waves, laser, hard drives in a van. . .

Hard drives in a van?!

https://docs.aws.amazon.com/snowball/latest/ug/using-device.html

High latency, high bandwidth!

Latency and bandwidth

Latency: time until message arrives

- In the same building/datacenter: ≈ 1 ms
- One continent to another: ≈ 100 ms
- Hard drives in a van: ≈ 1 day

Latency and bandwidth

Latency: time until message arrives

- In the same building/datacenter: ≈ 1 ms
- One continent to another: ≈ 100 ms
- Hard drives in a van: ≈ 1 day

Bandwidth: data volume per unit time

- 3G cellular data: ≈ 1 Mbit/s
- Home broadband: ≈ 10 Mbit/s
- Hard drives in a van: 50 TB/box ≈ 1 Gbit/s

(Very rough numbers, vary hugely in practice!)

Concurrent and Distributed Sys ×

Not Secure | cst.cam.ac.uk/teaching/2021/ConcDisSys

UNIVERSITY OF CAMBRIDGE | Study at Cambridge | About the University | Research at Cambridge | Quick links | Search

Department of Computer Science and Technology

Log in with Raven

Home | The department | Initiatives | Research | Admissions | Current students | Job vacancies | Intranet

Concurrent and Distributed Systems

Principal lecturer: Dr David Greaves
Martin Kleppmann

Students: Part IB CST 50%, Part IB CST 75%

Course code: ConcDisSys

Prerequisite course: Object-Oriented Programming
Operating Systems

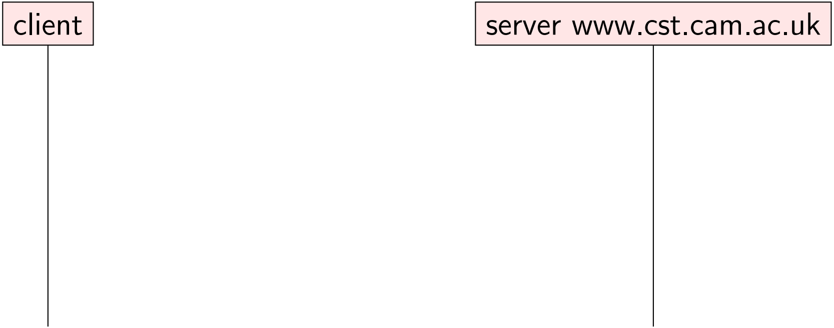
This course is a prerequisite for: Cloud Computing
Distributed Ledger Technologies: Foundations and Applications
Mobile and Sensor Systems

Related links

- Course materials
- Information for supervisors

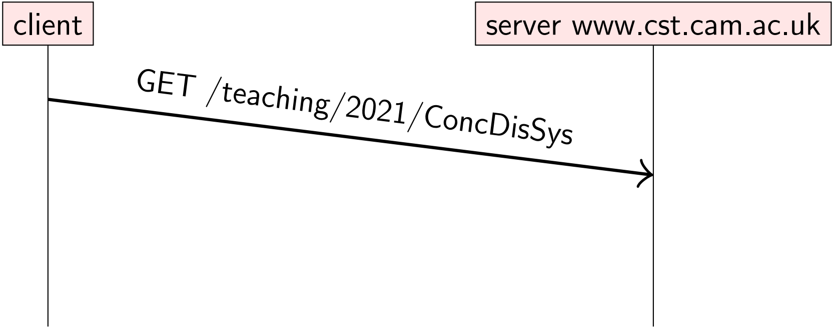
Client-server example: the web

Time flows from top to bottom.



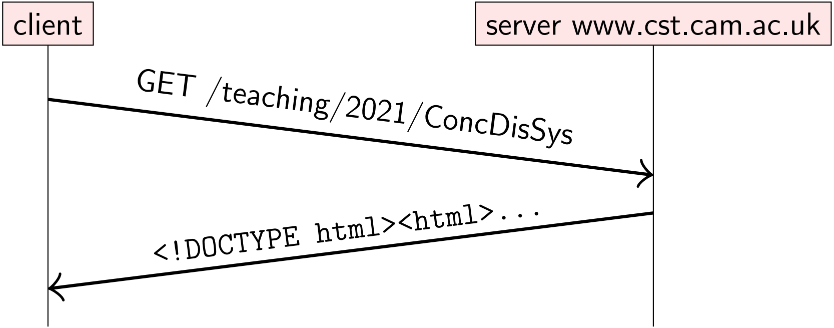
Client-server example: the web

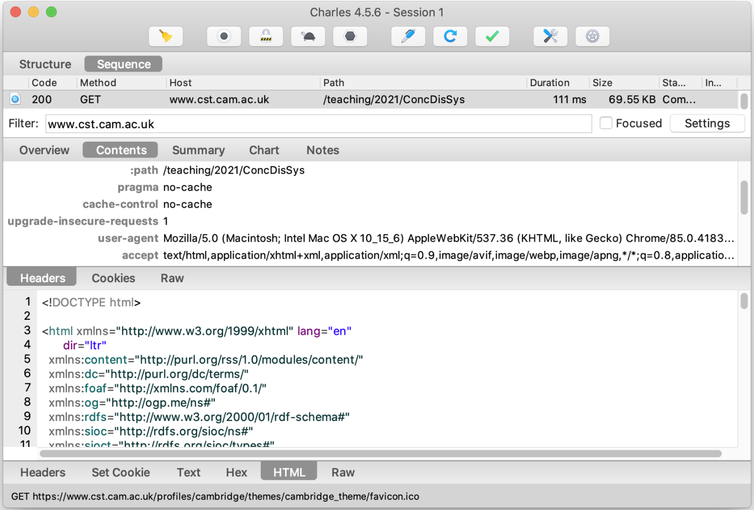
Time flows from top to bottom.

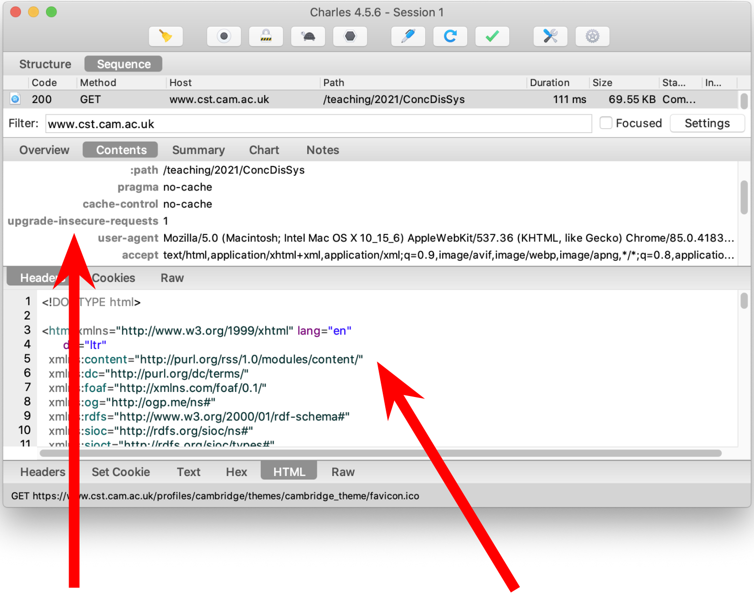


Client-server example: the web

Time flows from top to bottom.

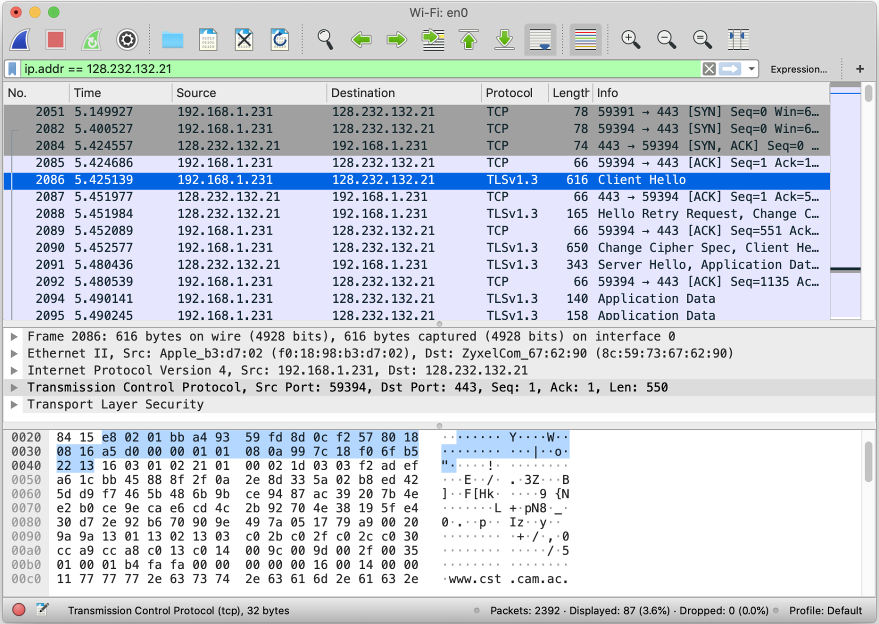



Charles 4.5.6 - Session 1
Structure
Sequence
Code
Method
Host
Path
Duration
Size
Sta...
In...
200
GET
www.cst.cam.ac.uk
/teaching/2021/ConcDisSys
111 ms
69.55 KB
Com...
Filter:
www.cst.cam.ac.uk
Focused
Settings
Overview
Contents
Summary
Chart
Notes
:path /teaching/2021/ConcDisSys
pragma no-cache
cache-control no-cache
upgrade-insecure-requests 1
user-agent Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183...
accept text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio...
Headers
Cookies
Raw
1 <!DOCTYPE html>
2
3 <html xmlns="http://www.w3.org/1999/xhtml" lang="en"
4 dir="ltr"
5 xmlns:content="http://purl.org/rss/1.0/modules/content/"
6 xmlns:dc="http://purl.org/dc/terms/"
7 xmlns:foaf="http://xmlns.com/foaf/0.1/"
8 xmlns:og="http://ogp.me/ns#"
9 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
10 xmlns:sioc="http://rdfs.org/sioc/ns#"
Headers
Set Cookie
Text
Hex
HTML
Raw
GET https://www.cst.cam.ac.uk/profiles/cambridge/themes/cambridge_theme/favicon.ico



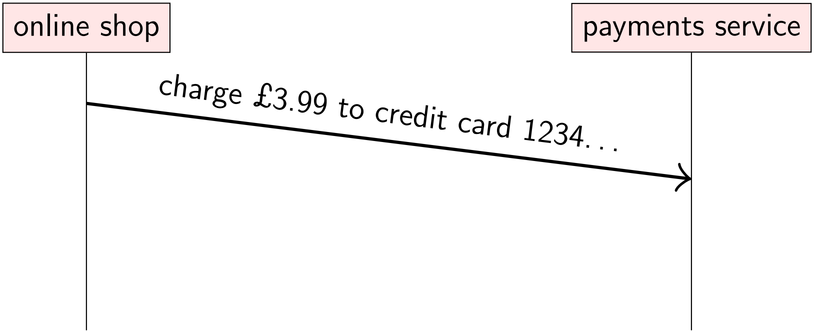
request message

response message

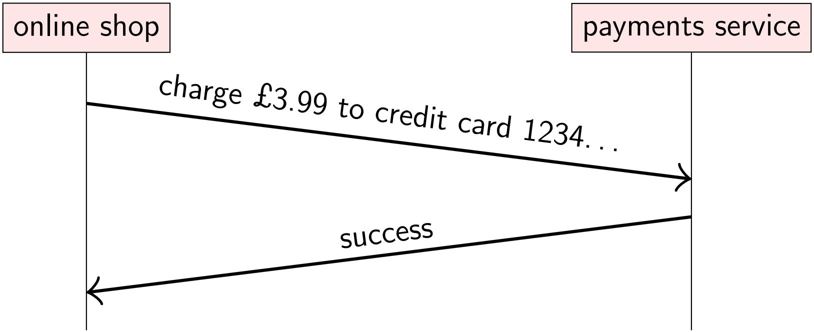

Wi-Fi: en0
ip.addr == 128.232.132.21
Expression…
No. Time Source Destination Protocol Length Info
2051 5.149927 192.168.1.231 128.232.132.21 TCP 78 59391 → 443 [SYN] Seq=0 Win=6…
2082 5.400527 192.168.1.231 128.232.132.21 TCP 78 59394 → 443 [SYN] Seq=0 Win=6…
2084 5.424557 128.232.132.21 192.168.1.231 TCP 74 443 → 59394 [SYN, ACK] Seq=0 …
2085 5.424686 192.168.1.231 128.232.132.21 TCP 66 59394 → 443 [ACK] Seq=1 Ack=1…
2086 5.425139 192.168.1.231 128.232.132.21 TLSv1.3 616 Client Hello
2087 5.451977 128.232.132.21 192.168.1.231 TCP 66 443 → 59394 [ACK] Seq=1 Ack=5…
2088 5.451984 128.232.132.21 192.168.1.231 TLSv1.3 165 Hello Retry Request, Change C…
2089 5.452089 192.168.1.231 128.232.132.21 TCP 66 59394 → 443 [ACK] Seq=551 Ack…
2090 5.452577 192.168.1.231 128.232.132.21 TLSv1.3 650 Change Cipher Spec, Client He…
2091 5.480436 128.232.132.21 192.168.1.231 TLSv1.3 343 Server Hello, Application Dat…
2092 5.480539 192.168.1.231 128.232.132.21 TCP 66 59394 → 443 [ACK] Seq=1135 Ac…
2094 5.490141 192.168.1.231 128.232.132.21 TLSv1.3 140 Application Data
2095 5.490245 192.168.1.231 128.232.132.21 TLSv1.3 158 Application Data
Frame 2086: 616 bytes on wire (4928 bits), 616 bytes captured (4928 bits) on interface 0
Ethernet II, Src: Apple_b3:d7:02 (f0:18:98:b3:d7:02), Dst: ZyxelCom_67:62:90 (8c:59:73:67:62:90)
Internet Protocol Version 4, Src: 192.168.1.231, Dst: 128.232.132.21
Transmission Control Protocol, Src Port: 59394, Dst Port: 443, Seq: 1, Ack: 1, Len: 550
Transport Layer Security
0020 84 15 e8 02 01 bb a4 93 59 fd 8d 0c f2 57 80 18
0030 08 16 a5 d0 00 00 01 01 08 0a 99 7c 18 f0 6f b5
0040 22 13 16 03 01 02 21 01 00 02 1d 03 03 f2 ad ef
0050 a6 1c bb 45 88 8f 2f 0a 2e 8d 33 5a 02 b8 ed 42
0060 5d d9 f7 46 5b 48 6b 9b ce 94 87 ac 39 20 7b 4e
0070 e2 b0 ce 9e ca e6 cd 4c 2b 92 70 4e 38 19 5f e4
0080 30 d7 2e 92 b6 70 90 9e 49 7a 05 17 79 a9 00 20
0090 9a 9a 13 01 13 02 13 03 c0 2b c0 2f c0 2c c0 30
00a0 cc a9 cc a8 c0 13 c0 14 00 9c 00 9d 00 2f 00 35
00b0 01 00 01 b4 fa fa 00 00 00 00 00 16 00 14 00 00
00c0 11 77 77 77 2e 63 73 74 2e 63 61 6d 2e 61 63 2e
·www.cst .cam.ac.
Transmission Control Protocol (tcp), 32 bytes
Packets: 2392 · Displayed: 87 (3.6%) · Dropped: 0 (0.0%)
Profile: Default

Client-server example: online payments

Client-server example: online payments



Client-server example: online payments



Remote Procedure Call (RPC) example

```java
// Online shop handling customer's card details
Card card = new Card();
card.setCardNumber("1234 5678 8765 4321");
card.setExpiryDate("10/2024");
card.setCVC("123");

Result result = paymentsService.processPayment(card,
    3.99, Currency.GBP);

if (result.isSuccess()) {
    fulfilOrder();
}
```

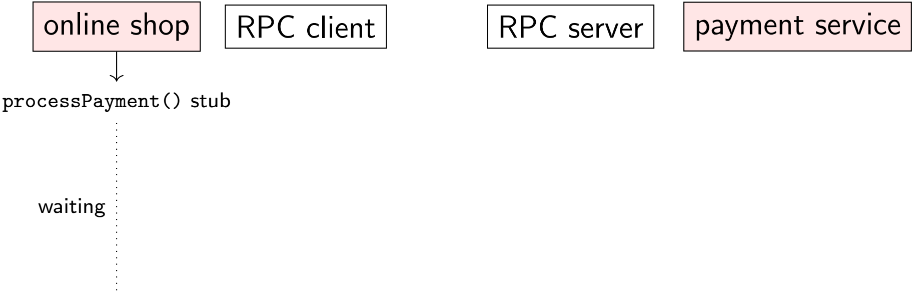
Remote Procedure Call (RPC) example

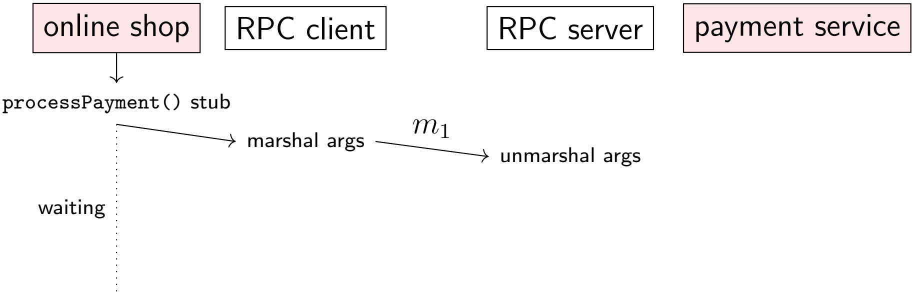
```java
// Online shop handling customer's card details
Card card = new Card();
card.setCardNumber("1234 5678 8765 4321");
card.setExpiryDate("10/2024");
card.setCVC("123");

Result result = paymentsService.processPayment(card,
    3.99, Currency.GBP);

if (result.isSuccess()) {
    fulfilOrder();
}
```

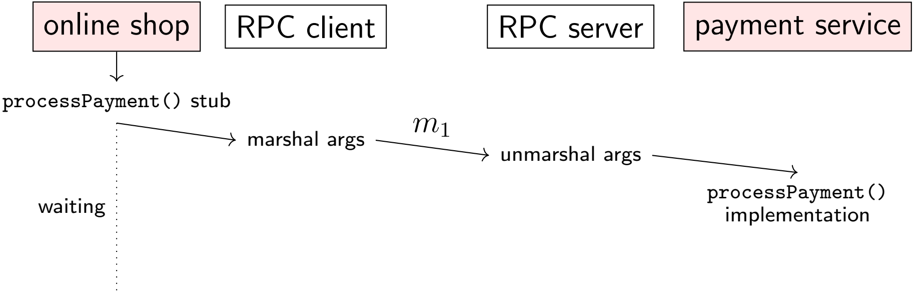
Implementation of this function is on another node!


online shop
RPC client
RPC server
payment service
processPayment() stub
waiting



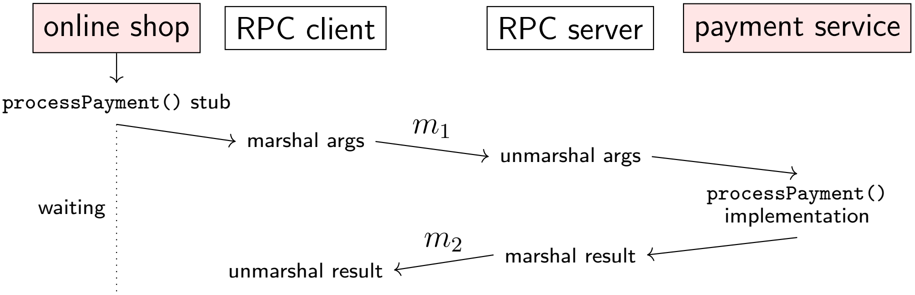
$m_1 =$


```json
{
  "request": "processPayment",
  "card": {
    "number": "1234567887654321",
    "expiryDate": "10/2024",
    "CVC": "123"
  },
  "amount": 3.99,
  "currency": "GBP"
}
```



$m_1 =$

```
{
  "request": "processPayment",
  "card": {
    "number": "1234567887654321",
    "expiryDate": "10/2024",
    "CVC": "123"
  },
  "amount": 3.99,
  "currency": "GBP"
}
```

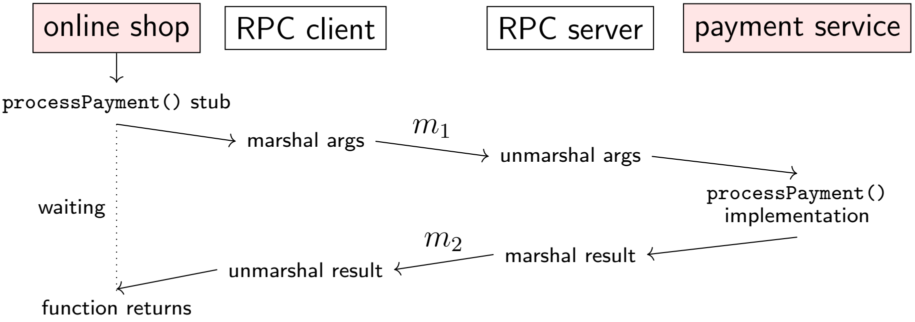


$m_1 =$

```
{
  "request": "processPayment",
  "card": {
    "number": "1234567887654321",
    "expiryDate": "10/2024",
    "CVC": "123"
  },
  "amount": 3.99,
  "currency": "GBP"
}
```

$m_2 =$

```
{
  "result": "success",
  "id": "XP61hHw2Rvo"
}
```



$m_1 =$

```
{
  "request": "processPayment",
  "card": {
    "number": "1234567887654321",
    "expiryDate": "10/2024",
    "CVC": "123"
  },
  "amount": 3.99,
  "currency": "GBP"
}
```

$m_2 =$

```
{
  "result": "success",
  "id": "XP61hHw2Rvo"
}
```

Remote Procedure Call (RPC)

Ideally, RPC makes a call to a remote function look the same as a local function call.

"Location transparency":
system hides where a resource is located.

Remote Procedure Call (RPC)

Ideally, RPC makes a call to a remote function look the same as a local function call.

"Location transparency":
system hides where a resource is located.

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?

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)
- Ajax in web browsers

RPC/REST in JavaScript

```
let args = {amount: 3.99, currency: 'GBP', /*...*/};
let request = {
  method:  'POST',
  body:    JSON.stringify(args),
  headers: {'Content-Type': 'application/json'}
};

fetch('https://example.com/payments', request)
  .then((response) => {
    if (response.ok) success(response.json());
    else failure(response.status); // server error
  })
  .catch((error) => {
    failure(error); // network error
  });
```

RPC in enterprise systems

"Service-oriented architecture" (SOA) / "microservices":

splitting a large software application into multiple services (on multiple nodes) that communicate via RPC.

RPC in enterprise systems

"Service-oriented architecture" (SOA) / "microservices":

splitting a large software application into multiple services (on multiple nodes) that communicate via RPC.

Different services implemented in different languages:

- interoperability: datatype conversions
- **Interface Definition Language** (IDL): language-independent API specification

gRPC IDL example

```
message PaymentRequest {
  message Card {
    required string cardNumber  = 1;
    optional int32  expiryMonth = 2;
    optional int32  expiryYear  = 3;
    optional int32  CVC         = 4;
  }
  enum Currency { GBP = 1; USD = 2; }

  required Card     card     = 1;
  required int64    amount   = 2;
  required Currency currency = 3;
}

message PaymentStatus {
  required bool   success      = 1;
  optional string errorMessage = 2;
}

service PaymentService {
  rpc ProcessPayment(PaymentRequest) returns (PaymentStatus) {}
}
```