Status 103: Early Hints

# Módulo 1: Introducción

⏮agenda

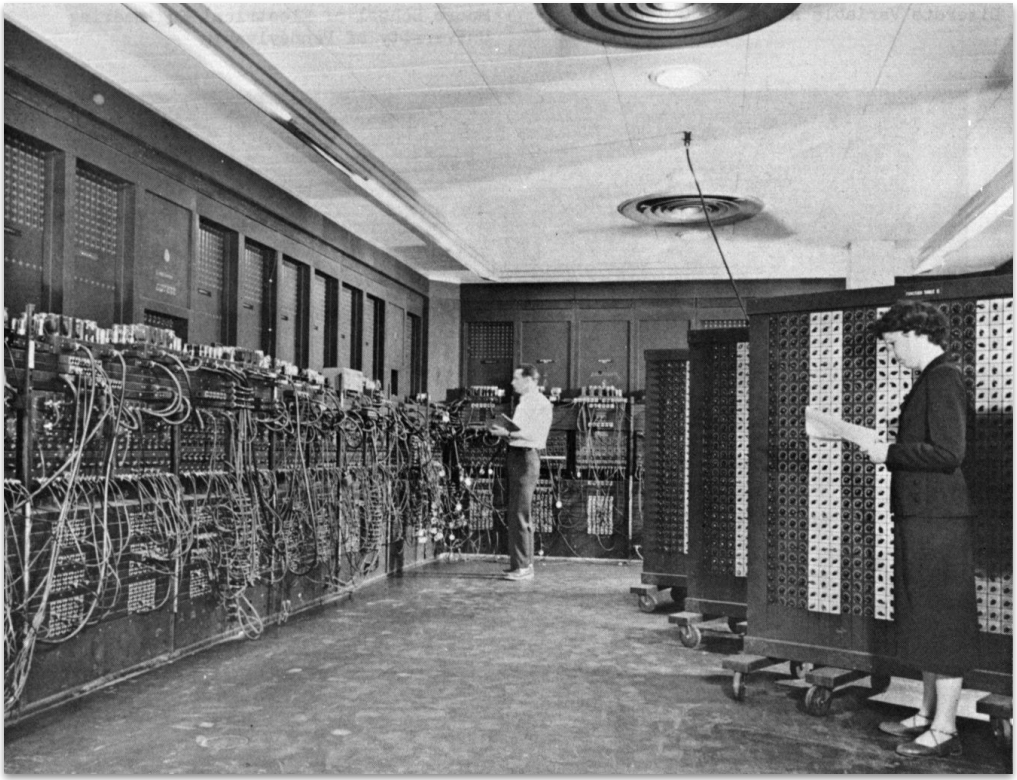
# En el comienzo...

Programas monolíticos:

- Cálculo de trayectorias
- La  bomba H

Carga manual.

Recursos subutilizados.



# En el comienzo...

¿Qué tal si automatizamos?

Surgen los primeros *sistemas operativos* (SOs)

1950s: SOs para la carga de programas

1960s: SOs más modernos

- Bibliotecas con rutinas para compilar, etc.
- Ej: OS/360 para los mainframes IBM (foto)

# Llamando al sistema: las syscalls

Estas rutinas básicas de los SOs estaban disponibles para cualquier programa.

Por ejemplo, un programa en C puede usar una llamada de sistema (*syscall)* para imprimir en pantalla.

```
int main(void) {
  register int    syscall no  asm("rax") = 1;
  register int    arg1        asm("rdi") = 1;
  register char*  arg2        asm("rsi") = "hello, world!\n";
  register int    arg3        asm("rdx") = 14;
  asm("syscall");
  return 0;
}
```

# Llamando a otras bibliotecas y *frameworks*

Luego surgieron bibliotecas/frameworks para complementar estas limitadas funcionalidades del SO

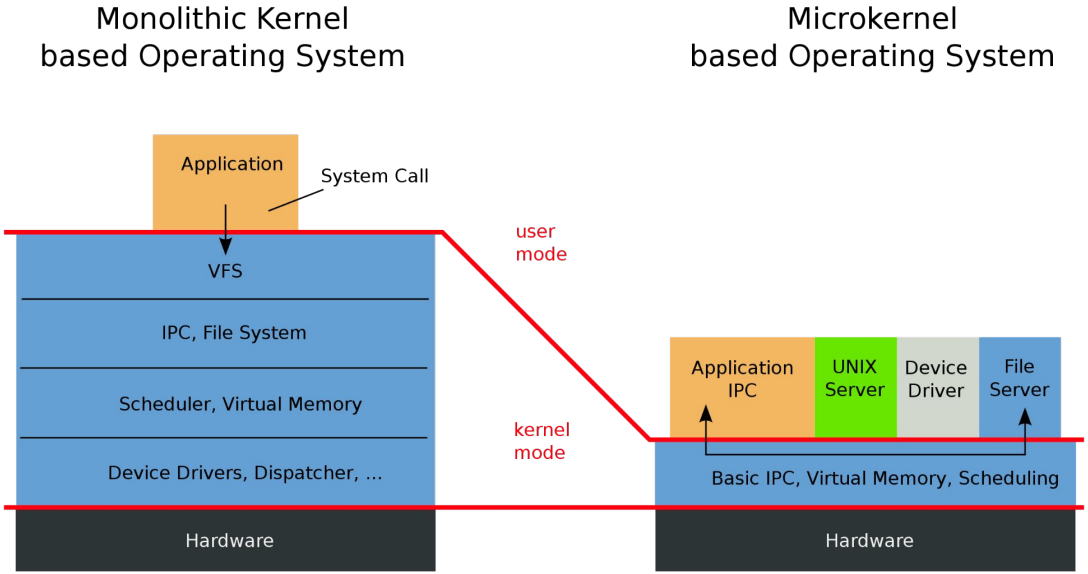
```
double[] values = new double[] {65, 51 , 16, 11 , 6519, 191 ,0 , 98, 19854, 1, 32};
DescriptiveStatistics descriptiveStatistics = new DescriptiveStatistics();
for (double v : values) {
    descriptiveStatistics.addValue(v);
}

double mean = descriptiveStatistics.getMean();
double median = descriptiveStatistics.getPercentile(50);
double standardDeviation = descriptiveStatistics.getStandardDeviation();
```

# Al ataque del kernel monolítico

1970s: microkernels

- Mover funcionalidad clave del SO a user space
- Clave: la comunicación interprocesos (IPC)



# Interprocess Communication (IPC)

Habilidad para que 2 procesos en una misma computadora se puedan comunicar entre sí

| | |
|---|---|
| **Archivos** | Muchos procesos acceden al mismo archivo a la vez |
| **Sockets** | Una comunicación punto a punto entre dos procesos |
| **Mensajes** | Cada proceso tiene su buzón y puede recibir mensajes de otros procesos |
| **Memoria compartida** | Muchos procesos acceden a las mismas posiciones de memoria a la vez |

# Interprocess Communication (IPC) - DEMO


https://github.com/gdecaso/curso-apis/tree/main/ipc

# Interprocess Communication (IPC)

Para observar de la demo:

- Empieza a surgir el concepto de **protocolo**.
  - Si en el ejemplo de Sockets arrancamos primero el LadoB, se rompe
  - Si leemos primero una fecha y luego un entero, se rompe

# ¿Y qué pasa si lo que necesito no está en la misma compu?

En IPC siempre comunicamos 2 procesos en la **misma** computadora.

A veces eso no alcanza. Ejemplos:

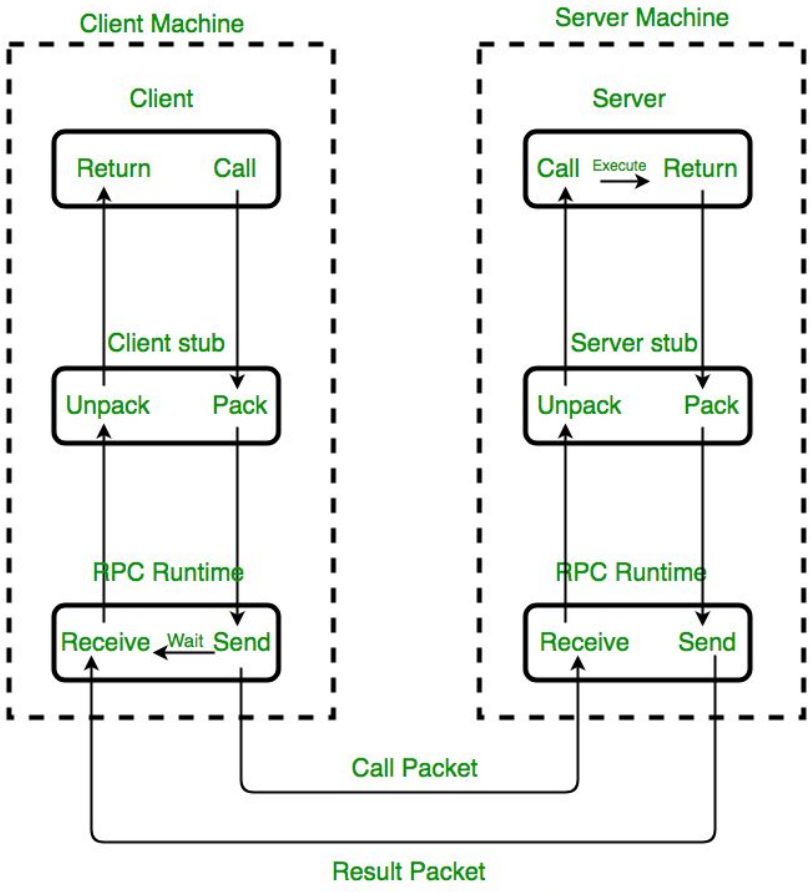
- Un cajero automático consultando el saldo de cuenta a la casa matriz
- Dos nodos en una red de descargas tipo peer-to-peer intercambian datos
- Mi navegador quiere obtener los titulares de las noticias de un portal
- ...

Hoy en día cada vez más cosas requieren de comunicarse por fuera de nuestra compu

Surgen las primeras tecnologías hacia la década de 1980

# Remote Procedure Call (RPC)

Esta técnica nos permite invocar un procedimiento remoto como si se tratara de una llamada local.



# gRPC:


Creado en 2016 por Google. Open Source.

Basado en Protocol Buffers (protobuf), un sistema de Google para codificar/decodificar mensajes

Adoptado por varias empresas: Uber, Netflix, Spotify, DropBox...

# Remote Procedure Call (RPC) - DEMO

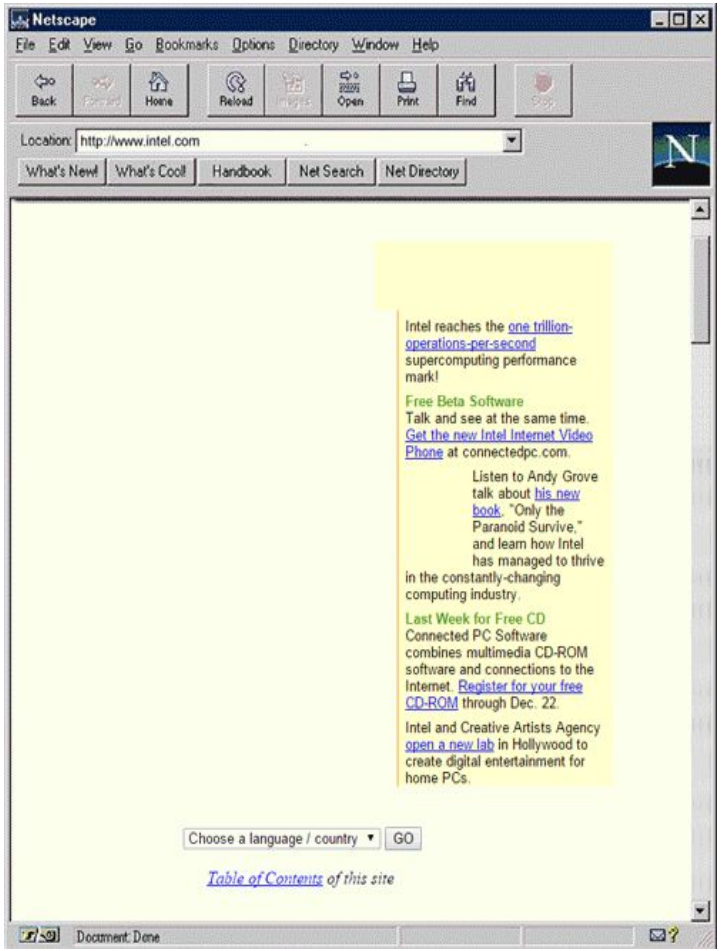
https://github.com/gdecaso/curso-apis/tree/main/grpc

# Syscalls, bibliotecas, IPC y RPC – limitaciones

- Syscalls
  - Sólo dentro de la misma computadora
  - Un conjunto muy limitado de funcionalidades
- Bibliotecas/frameworks
  - Dentro de un mismo proceso
  - Funcionalidades de todo tipo
- IPC
  - Limitado a 2 procesos en la misma computadora
  - Requiere de un protocolo pre-acordado
- RPC
  - Permite conectar distintas computadoras
  - Requiere de un protocolo pre-acordado

# IPC y RPC, limitaciones

Hasta los años 80s estas tecnologías eran suficientes... pero pasó algo en la década del 90



# La necesidad de establecer contratos

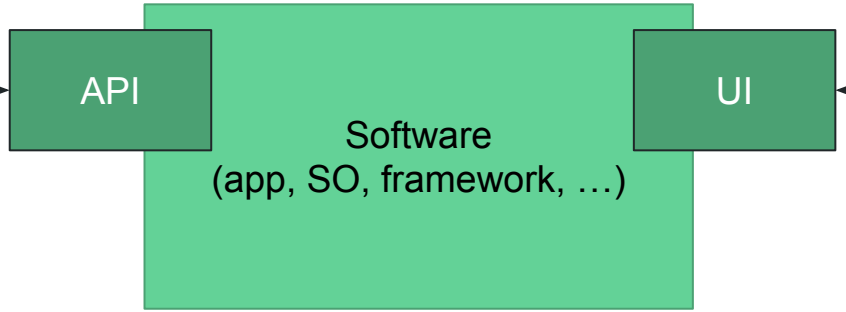
Ya no se trataba de coordinar un par de computadoras punta a punta.

Ahora contamos con miles, cientos de miles y hasta millones de computadoras interconectadas.

# Application Programming Interface (APIs)

Una API es una “ventanilla” que una pieza de software le ofrece a otras para que se comuniquen con ella.

Otro software



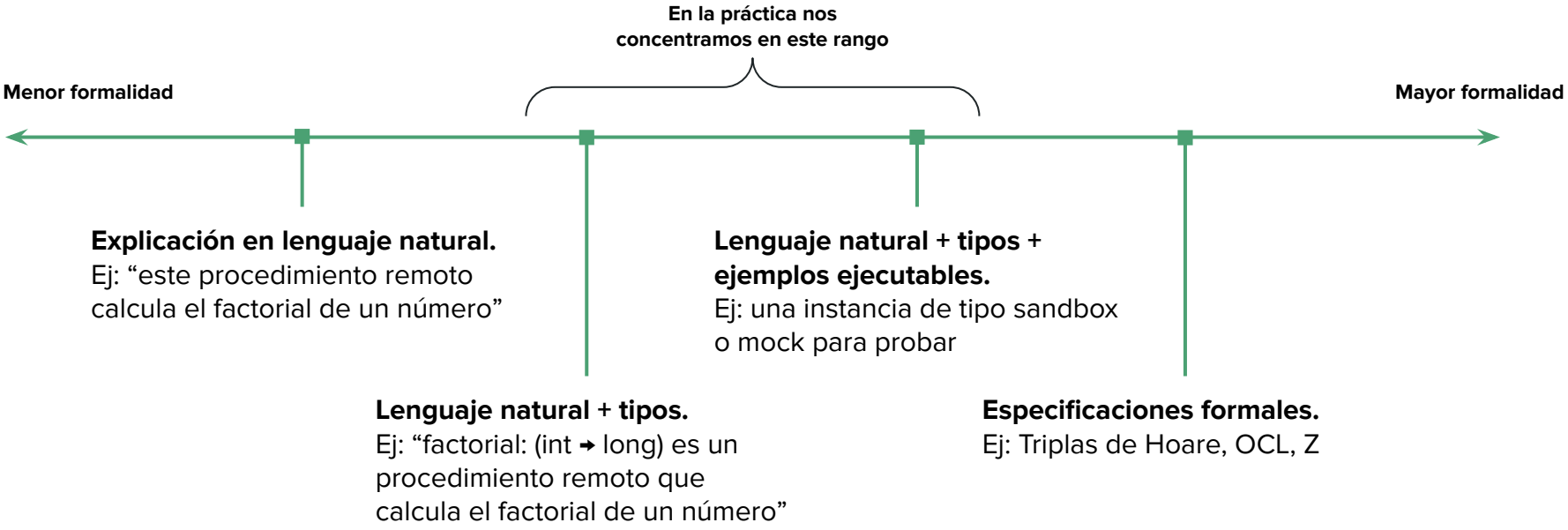

Usuari@s

# APIs: niveles de abstracción

- APIs a nivel sistema:
  - Estas APIs las utiliza un programa para comunicarse con el sistema operativo. Incluso hay APIs para que los dispositivos de hardware se comuniquen entre sí.
- APIs a nivel plataforma:
  - Estas APIs las utiliza un programa para comunicarse con la plataforma en la que está corriendo.
  - Ej: API de Java
- APIs a nivel aplicación:
  - Estas APIs se utilizan para comunicar aplicaciones entre sí.
  - Ej: Una aplicación de navegación se comunica con la API de Google Maps para mostrar imágenes de Street View.

# La necesidad de establecer contratos

Queremos conectar de manera predecible una pieza de software con otra.



# Contratos de APIs a nivel syscalls

Linux

**NAME** top

_exit, _Exit - terminate the calling process

**SYNOPSIS** top

```
#include <unistd.h>

noreturn void _exit(int status);

#include <stdlib.h>

noreturn void _Exit(int status);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

_Exit():
_ISOC99_SOURCE || _POSIX_C_SOURCE >= 200112L

**DESCRIPTION** top

**_exit**() terminates the calling process "immediately". Any open file descriptors belonging to the process are closed. Any children of the process are inherited by init(1) (or by the nearest "subreaper" process as defined through the use of the prctl(2) **PR_SET_CHILD_SUBREAPER** operation). The process's parent is sent a **SIGCHLD** signal.

The value *status & 0xFF* is returned to the parent process as the process's exit status, and can be collected by the parent using one of the wait(2) family of calls.

The function **_Exit**() is equivalent to **_exit**().

**RETURN VALUE** top

These functions do not return.

macOS

**NAME**

fork -- create a new process

**SYNOPSIS**

```
#include <unistd.h>

pid_t
fork(void);
```

**DESCRIPTION**

Fork() causes creation of a new process. The new process (child process) is an exact copy of the calling process (parent process) except for the following:

- The child process has a unique process ID.
- The child process has a different parent process ID (i.e., the process ID of the parent process).
- The child process has its own copy of the parent's descriptors. These descriptors reference the same underlying objects, so that, for instance, file pointers in file objects are shared between the child and the parent, so that an lseek(2) on a descriptor in the child process can affect a subsequent read or write by the parent. This descriptor copying is also used by the shell to establish standard input and output for newly created processes as well as to set up pipes.
- The child processes resource utilizations are set to 0; see setrlimit(2).

**RETURN VALUES**

Upon successful completion, fork() returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, no child process is created, and the global variable errno is set to indicate the error.

# Contratos de APIs a nivel bibliotecas/frameworks

**cos**

```
public static double cos(double a)
```

Returns the trigonometric cosine of an angle. Special cases:

- If the argument is NaN or an infinity, then the result is NaN.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

**Parameters:**

`a - an angle, in radians.`

**Returns:**

`the cosine of the argument.`

Java

**write**

```
public reactor.core.publisher.Mono<Void> write(Publisher<? extends Resource> inputStream,
                                               ResolvableType elementType,
                                               @Nullable
                                               MediaType mediaType,
                                               ReactiveHttpOutputMessage message,
                                               Map<String,Object> hints)
```

**Description copied from interface: HttpMessageWriter**

Write an given stream of object to the output message.

**Specified by:**

`write` in interface `HttpMessageWriter<Resource>`

**Parameters:**

`inputStream` - the objects to write

`elementType` - the type of objects in the stream which must have been previously checked via `HttpMessageWriter.canWrite(ResolvableType, MediaType)`

`mediaType` - the content type for the write (possibly null to indicate that the default content type of the writer must be used)

`message` - the message to write to

`hints` - additional information about how to encode and write

**Returns:**

`indicates completion or error`

Spring

`date.`**`weekday`**`()`

Return the day of the week as an integer, where Monday is 0 and Sunday is 6. For example, `date(2002, 12, 4).weekday() == 2`, a Wednesday. See also `isoweekday()`.

Python

# Contratos de APIs a nivel aplicación

Ciudad de Buenos Aires

GET /colectivos/vehiclePositionsSimple

Devuelve la posición de los vehículos monitoreados actualizada cada 30 segundos con el formato simplificado, agregando el route_short_name (línea de colectivo) a la respuesta.

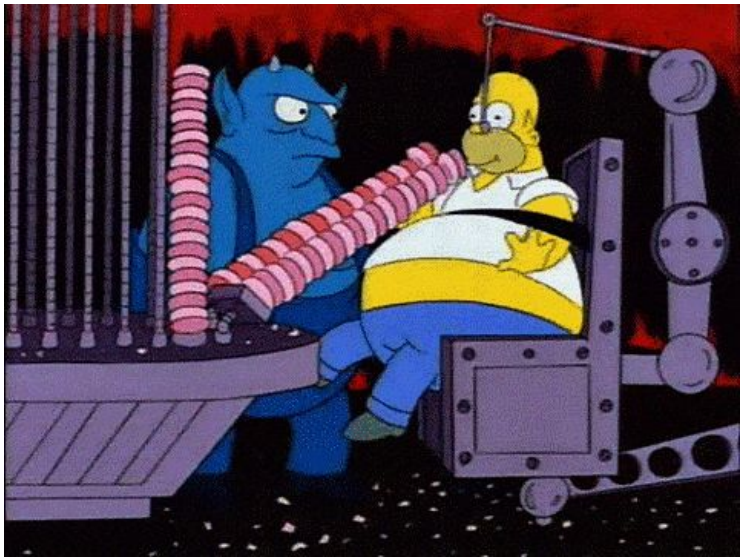
Parameters | Try it out

| Name | Description |
|---|---|
| route_id<br>string<br>(query) | route_id |
| agency_id<br>integer<br>(query) | agency_id |
| Trip<br>string<br>(query) | Trip |
| client_id * required<br>string<br>(query) | client_id |
| client_secret * required<br>string<br>(query) | client_secret |

Responses

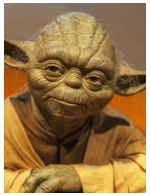
| Code | Description | Links |
|---|---|---|
| 200 | | No links |

Posiciones en tiempo real de todos los colectivos de la Ciudad de Buenos Aires



# Contratos de APIs a nivel aplicación

Autogenerada y redactada por Yoda



OpenShift

**HTTP method**

`POST`

**Description**

connect POST requests to proxy of Pod

*Table 36. HTTP responses*

| HTTP code | Reponse body |
|---|---|
| 200 - OK | `string` |
| 401 - Unauthorized | Empty |

# Contratos de APIs a nivel aplicación

No disponible salvo que seas un cliente pago


# La importancia de los contratos - el caso EU/Microsoft

1998:

- Sun inicia acciones legales contra Microsoft por posición dominante en mercado de SOs

2003:

- La Free Software Foundation Europe (FSFE) y Samba aportan evidencia:
  - "*Each (proprietary) protocol depends on the correct implementation of another (proprietary) protocol to work properly.* ***Full functionality*** *and interoperability* ***can only be achieved*** *when* ***all protocols*** *are known*"

2004:

- Primer fallo de la Comisión Europea: multa de 500 M de Euros

# La importancia de los contratos - el caso EU/Microsoft

2005:

- Microsoft inicia una serie de apelaciones
- La CE le impone una multa adicional de 3M de Euros por cada día que pasa

2007:

- Microsoft comienza con sus esfuerzos de documentar protocolos

# La economía de las APIs

Microsoft tuvo una mirada cerrada hacia los contratos.

- Los abrió porque no le quedaba otra.

¿Qué pasa si tomamos la mirada opuesta?

- Las APIs pueden ser una fuente para hacer crecer un negocio

# La economía de las APIs: Ejemplos

Biblioteca con **muchas** APIs: https://apislist.com/

Twilio:

- Telefonía como servicio mediante APIs.

AWS - Amazon:

- Exceso de capacidad de cómputo debido a los picos de carga (navidad, black friday)
- Surge el cómputo como servicio
- Luego seguirían almacenamiento, DNS, etc.
- Hoy AWS genera más ganancias que las ventas minoristas de Amazon

MercadoLibre:

- Solución de pagos para su plataforma de e-commerce
- Ya que estaban la abrieron a terceros