Sistemas Distribuidos I
(75.74)

Nombres, Grupos y Relojes

Nombres y Direccionamiento. Mensajes. Grupos. Tiempo y Relojes.

Docentes

- Pablo D. Roca
- Ezequiel Torres Feyuk
- Guido Albarello
- Ana Czarnitzki
- Cristian Raña

Agenda

- **Nombres y direccionamiento**
- Mensajes
- Grupos
- Tiempo y relojes - Relojes Físicos
- Tiempo y relojes - Relojes Lógicos

Nombres y Direccionamiento | Introducción

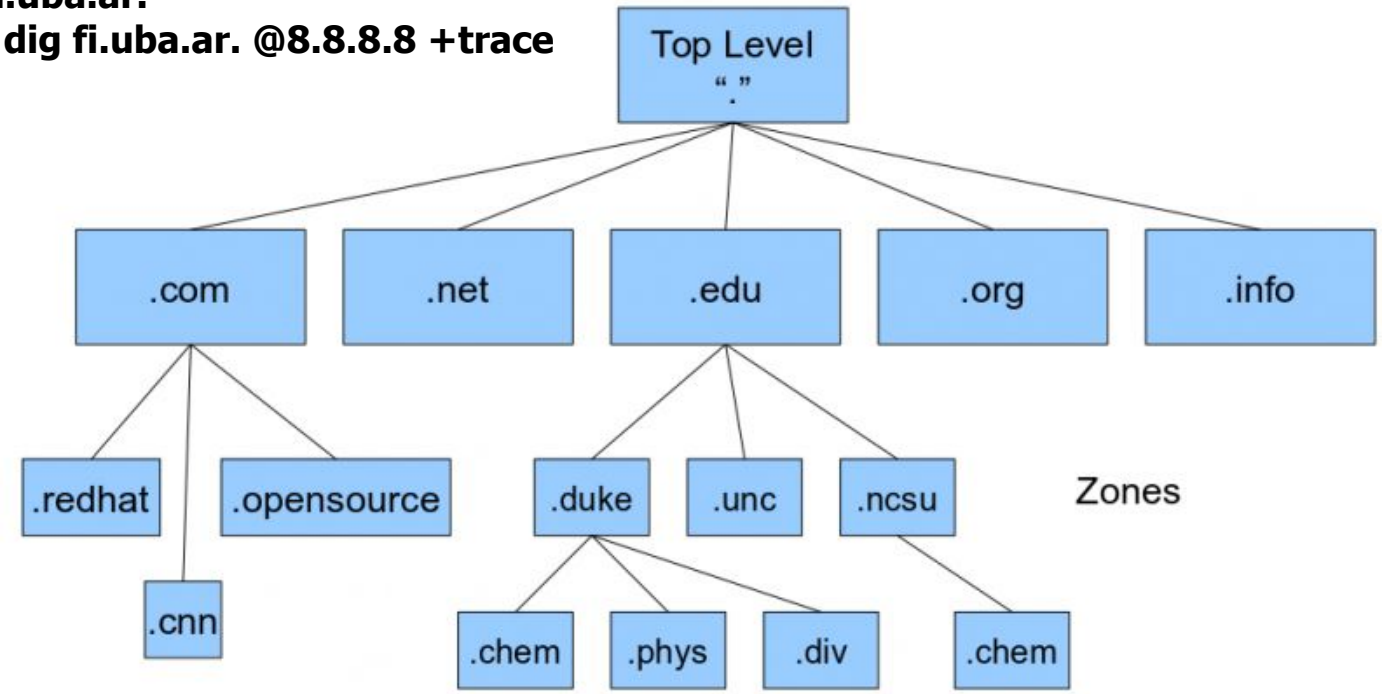
- **Nombres**
  - Permiten identificar *unívocamente a una entidad* dentro de un sistema
  - Deben describir a la entidad
  - Abstraen al recurso de las propiedades que atan al mismo con el sistema (lugar geográfico, direcciones de red)

- **Direccionamiento (Addressing)**
  - Mapeo entre un nombre y una dirección
  - Dirección de una entidad puede cambiar, **nombre no** (*)
  - Dirección puede ser reutilizada

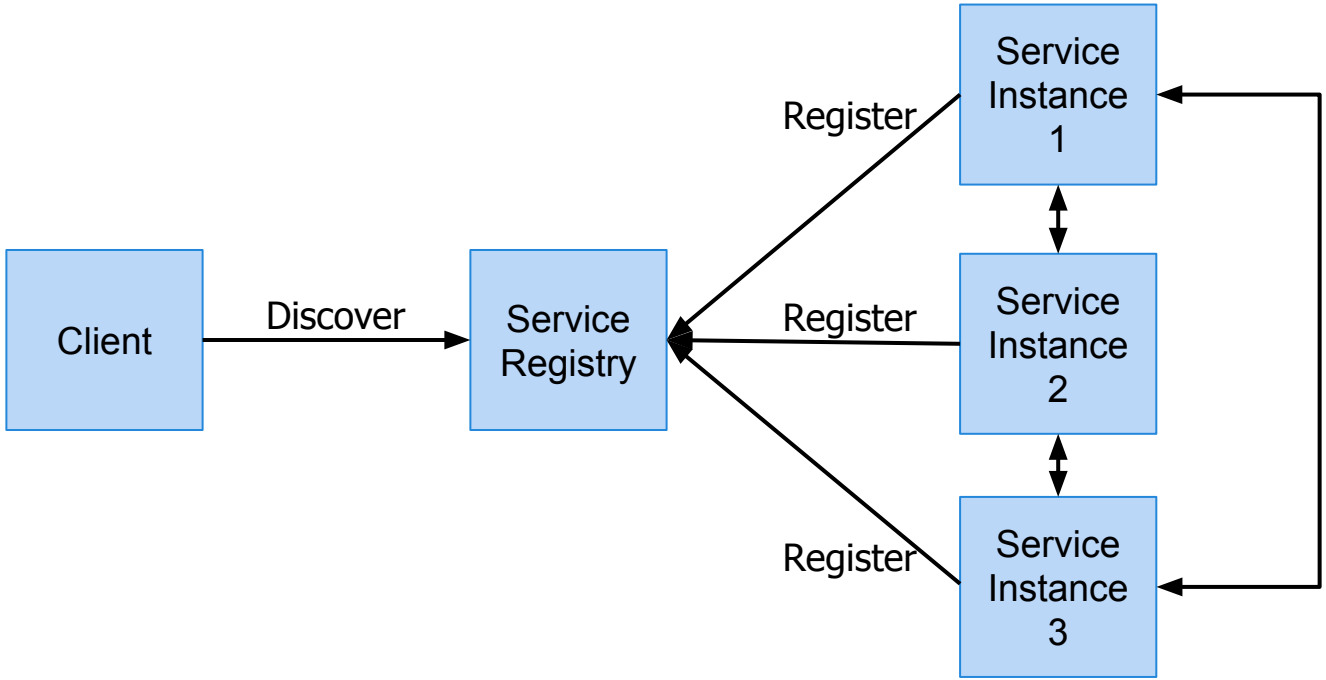
Nombres y Direccionamiento | Ejemplos

- **IP Address (name) -> Ethernet Address (address)**
  - IP address identifica a un nodo en un red (sea local o no)
  - Ethernet address identifica a NIC (network interface card) de un nodo en una red local
  - Resolución se realiza a través de protocolo ARP en IPv4 o ND (Neighbor Discovery) en IPv6
- **Domain Name (name) -> IP Address (address)**
  - Mapeo de un servicio/nodo/otra entidad a una dirección IP
  - Traducción a través de protocolo DNS
- **Service (name) -> Instances (address)**
  - Mapeo del nombre de un servicio a alguna instancia
  - Resolución a través de Service Discovery
  - Diferentes implementaciones existentes: Zookeeper, Istio, Linkerd

Ejemplo: fi.uba.ar.
Comando: dig fi.uba.ar. @8.8.8.8 +trace



Nombres y Direccionamiento | Ejemplo: Service Discovery



Agenda

- Nombres y direccionamiento
- **Mensajes**
- Grupos
- Tiempo y relojes - Relojes Físicos
- Tiempo y relojes - Relojes Lógicos

Mensajes | Formato de Paquetes | Binario

- **Alta performance**
  - Tamaño de mensajes eficientes
  - Compresión puede no ser necesaria
- **Serialización**
  - Autogeneración de código (e.g. Google Protobuf, Thrift, ASN.1)
  - No siempre existe soporte en todos los lenguajes
- **Interacción**
  - Cliente específico para cada aplicación
  - Decoder para interpretar los mensajes

Mensajes | Formato de Paquetes | Ejemplo (Protobuf)

person.proto

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

Generación de Código

```
protoc -I=$SRC_DIR --cpp_out=$DST_DIR
$SRC_DIR/person.proto
```

Serialización (Java)

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new
FileOutputStream(args[0]);
john.writeTo(output);
```

Deserialización (C++)

```
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

- **Baja performance**
  - *Throughput* bajo
  - Compresión agrega *overhead*
- **Serialización**
  - Formatos *human-readable* (JSON, XML)
  - Serialización básica (e.g. HTTP, SMTP)
- **Interacción**
  - Cliente único si se conoce el protocolo (e.g. cURL + REST API)
  - Fácil de *debuggear*

Mensajes | Formato de Paquetes | Ejemplo (CURL)

```
user@hostname:~$ curl -H "Content-Type: application/json" \
-X POST \
-d '{"username":"lalala","password":"supersecurepass"}' \
http://localhost:8080/api/login/
```

Request

```
POST /login HTTP/1.1
Host: localhost:8080
User-Agent: curl/7.58.0
Accept: */*
Content-Type: application/json
Content-Length: 50

{"username":"lalala","password":"supersecurepass"}
```

Mensajes | Longitud de Paquete

- **Bloques fijos**
  - Cada dato a enviar posee una longitud fija
  - Fácil de serializar
  - Subóptimo con tipos de longitud variable (e.g. *strings*)

- **Bloques dinámicos**
  - Separador para delimitar comienzo y terminación de un campo
  - Longitud del tipo para delimitar longitud del campo
  - Pequeño overhead al agregar *bytes* extra

- **Esquema mixto**
  - Parámetro fijos (e.g. *integers*) no llevan delimitadores / longitud
  - Parámetros variables llevan delimitadores / longitud

Mensajes | Longitud de Paquete (Ejemplo: TLV)

- Todos los parámetros siguen el formato Type - Length - Value
  - **Type**: Indica el tipo de dato / entidad. Tamaño fijo
  - **Length**: Longitud del value sin contar el tipo y el length. Tamaño fijo
  - **Value**: Dato a enviar. Tamaño variable. **Admite subtipos**

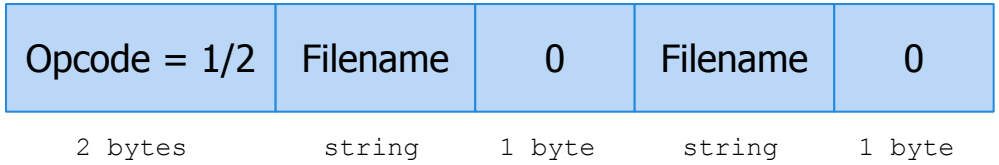
- Ejemplo: Encodear una dirección: Alsina 2B

```
Mini protocolo
Tipo 01: Integer (4 bytes)
Tipo 02: string (variable)
Tipo 03: Dirección (compuesto)
Tipo 03.01: Calle (string)
Tipo 03.02: Número (integer)
Tipo 03.03: Piso (string)
```

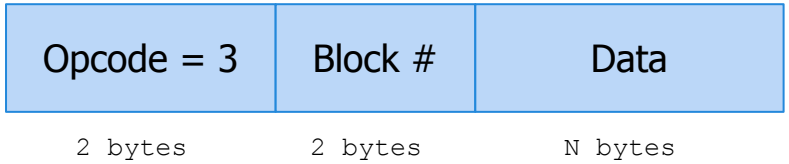
```
Serialización
03 11 01 06 41 6c 73 69 6e 61 02 04 00 00
00 02 03 01 42

Serialización (desglose)
03 11 (Dirección)
     01 06 41 6c 73 69 6e 61 (Calle)
     02 04 00 00 00 02 (Número)
     03 01 42 (Piso)
```

Mensajes | Longitud de Paquete (Ejemplo: TFTP)

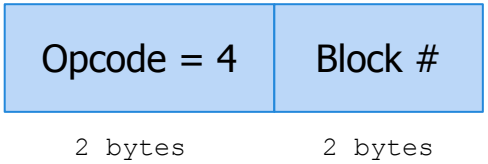
Read/Write Packet



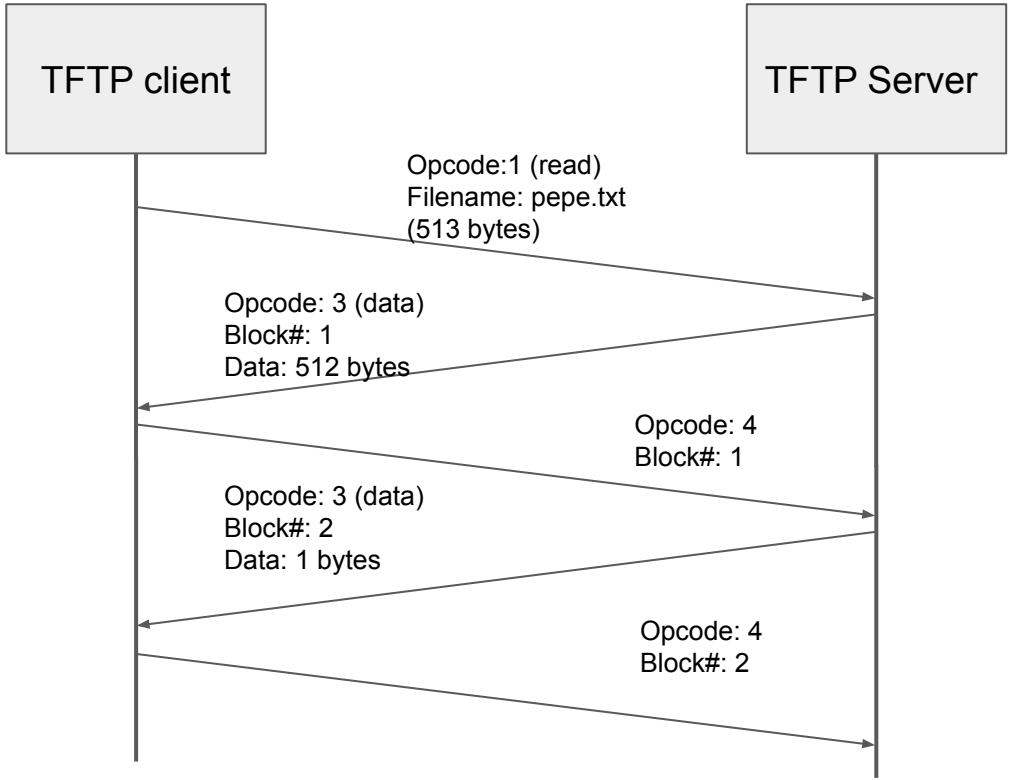
Data Packet



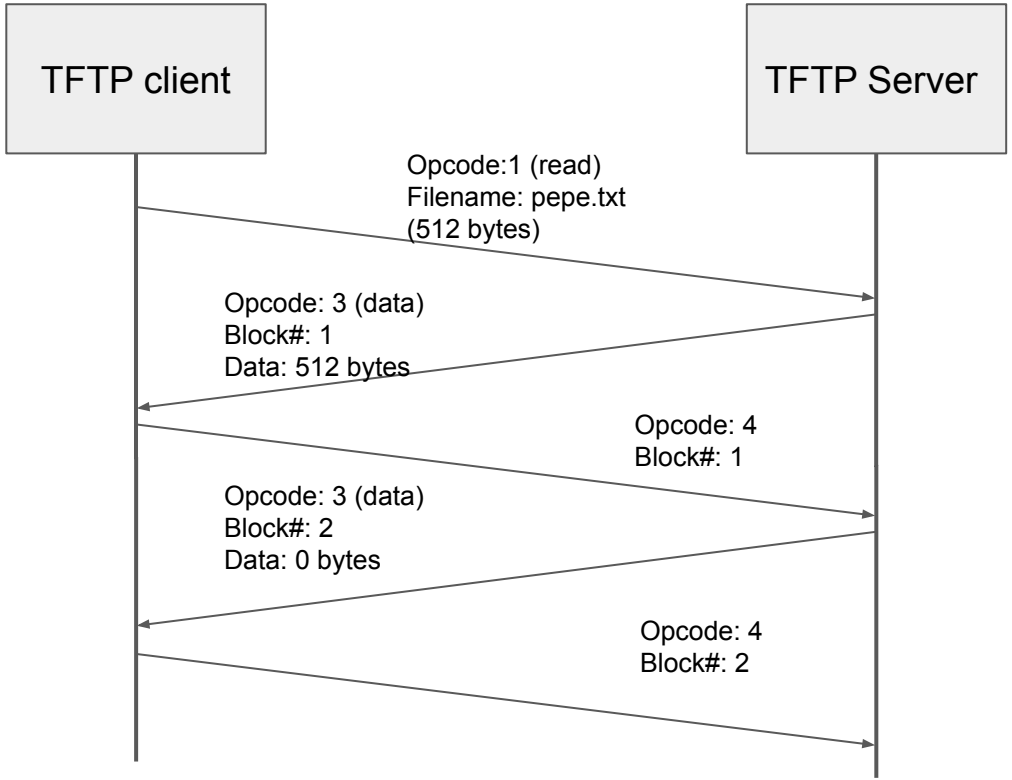
ACK Packet



Mensajes | Longitud de Paquete (Ejemplo: TFTP)

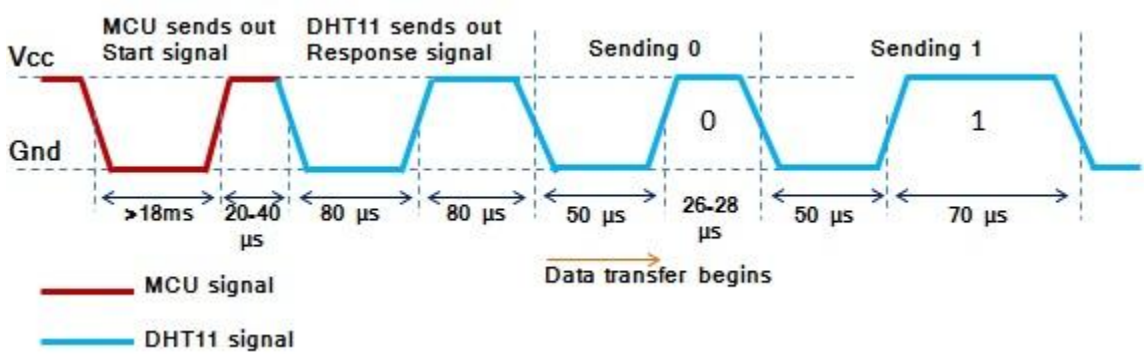


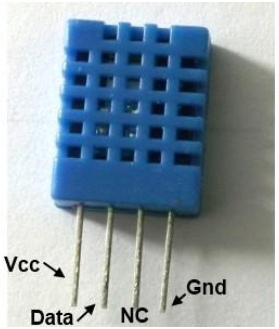
Mensajes | Longitud de Paquete (Ejemplo: TFTP)



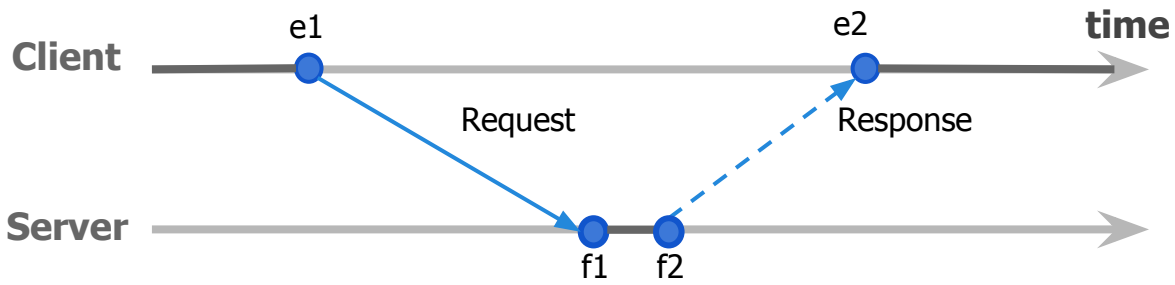
Mensajes | Sincronización de señales

- En ciertos protocolos es necesario saber dónde empieza una señal para entender la secuencia de paquetes.
- **Ejemplo bajo nivel: DHT11 (sensor de humedad y temperatura)**
  - DHT11 envía data sobre pin de comunicación
  - Comienzo con una señal de sincronismo
  - Data (40-bit) = Integer Byte of RH + Decimal Byte of RH + Integer Byte of Temp. + Decimal Byte of Temp. + Checksum Byte

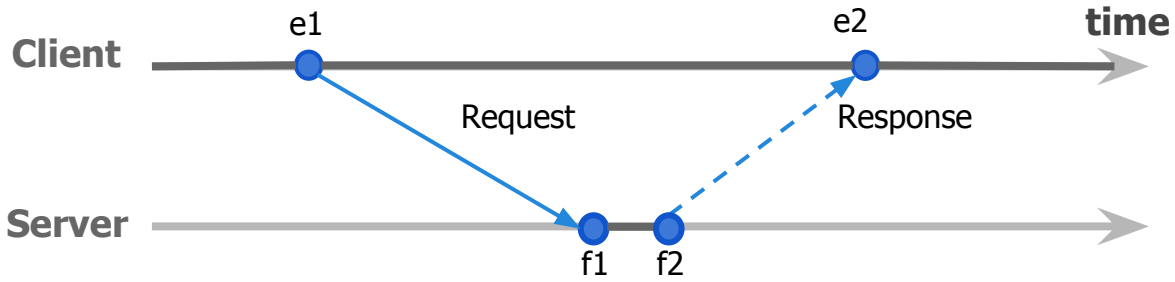




Sincrónico



Asincrónico

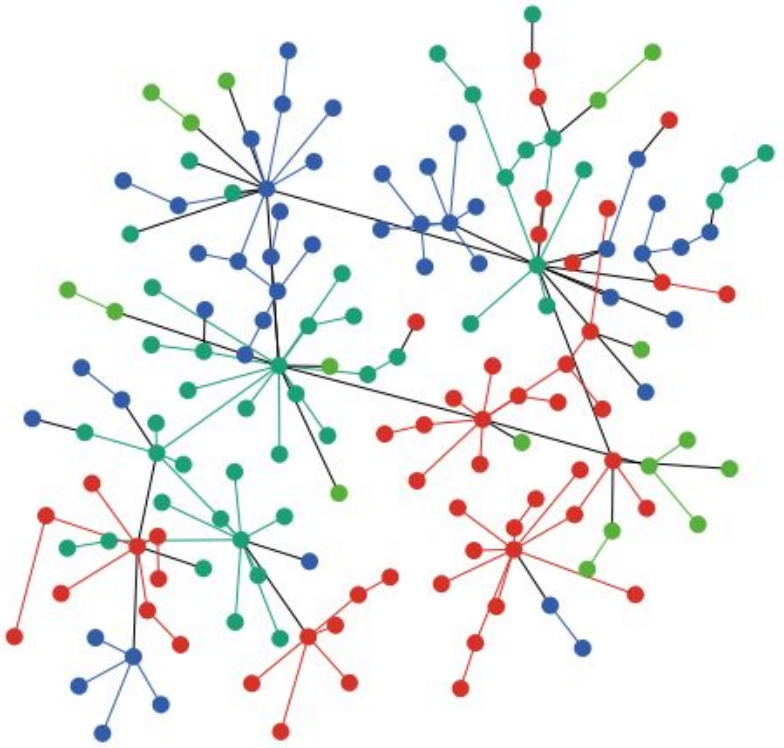


Agenda

- Nombres y direccionamiento
- Mensajes
- **Grupos**
- Tiempo y relojes - Relojes Físicos
- Tiempo y relojes - Relojes Lógicos

Grupos de Comunicación | Introducción

- Permiten ver a una colección de procesos como una abstracción
- Mensaje es enviado a todas o algunas entidades que componen el grupo
- Grupos son dinámicos
  - Deben poder crearse y destruirse en todo momento
- Procesos se deben poder suscribir y cancelar la suscripción a grupos (deben existir primitivas para que esto suceda)

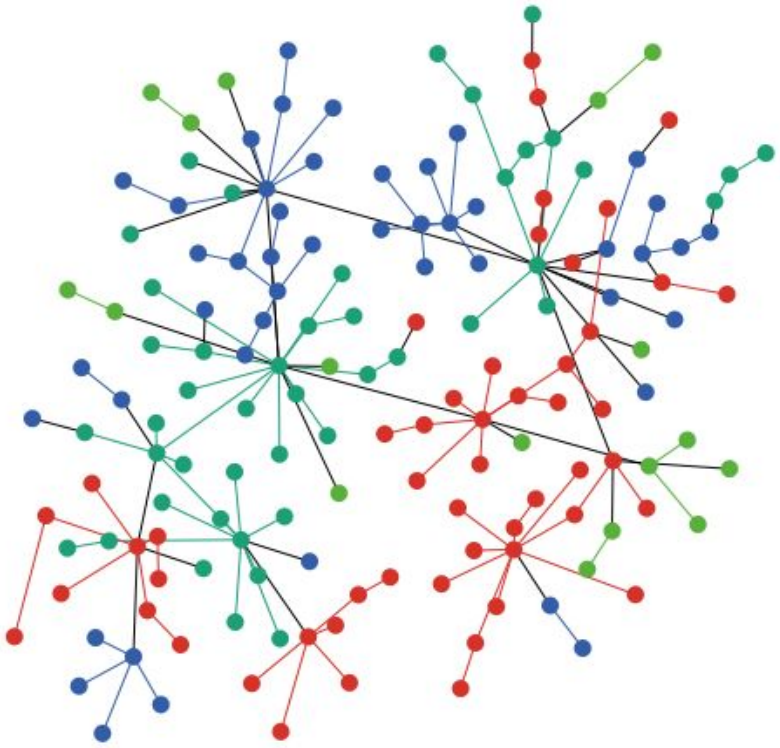


Uno a uno

- *Unicast*: Comunicación punto a punto
- *Anycast*: Uno sólo en el grupo recibe el mensaje
  - E.g.: Envío al nodo más cercano (ECMP). Que implica cercanía?

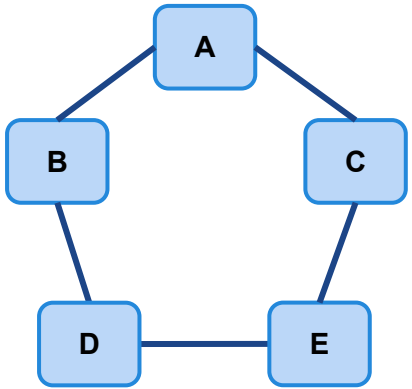
Uno a Muchos

- *Multicast*: Sólo aquellos que se encuentran en el grupo reciben el mensaje
- *Broadcast*: Todos reciben el mensaje

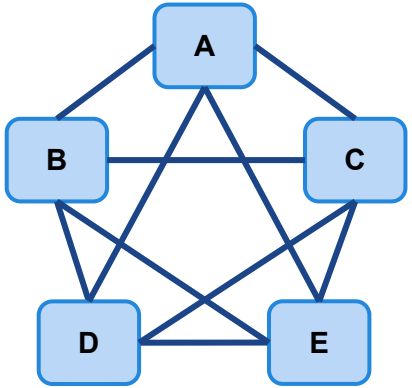


Grupos de Comunicación | Topología

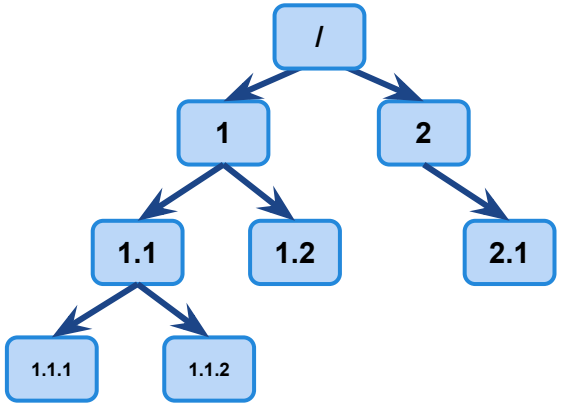
Anillos



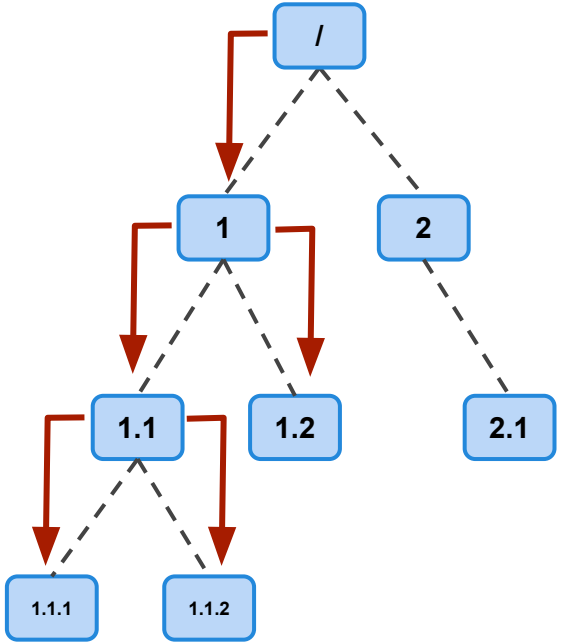
Punto a Punto



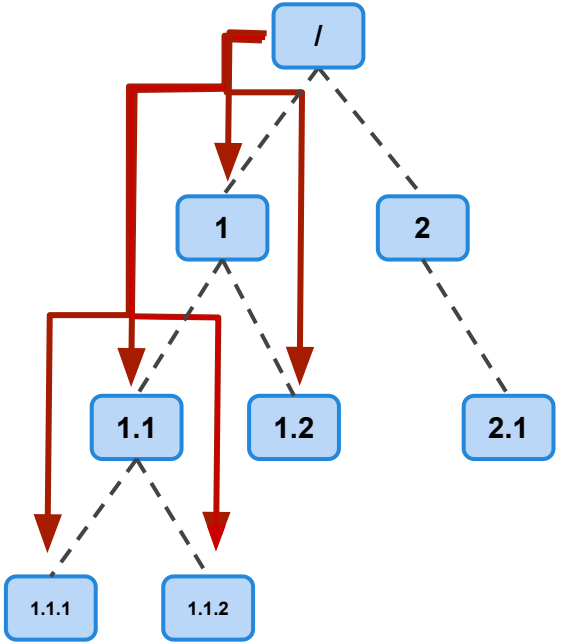
Grupos Jerárquicos



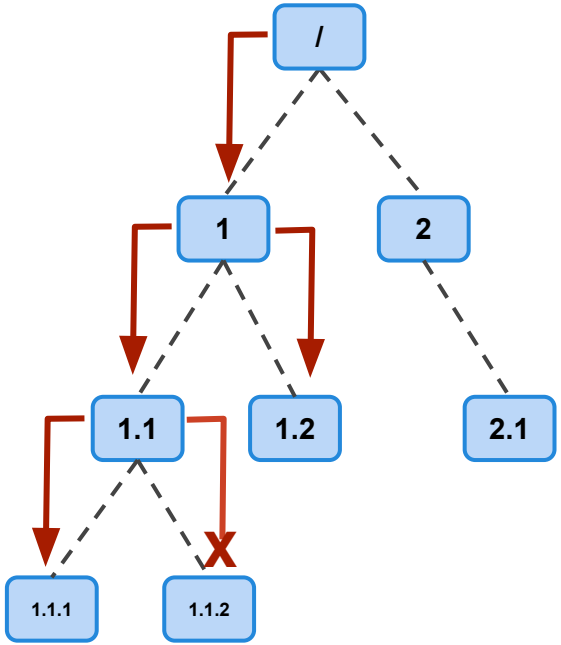
Difusión Descentralizada



Difusión Centralizada



- Los mensajes deben entregarse a todos o a ninguno de los miembros
- Necesidad de realizar ACK de mensajes
- Necesidad de demorar el *delivery* de los paquetes recibidos
- Reintentos frente a:
  - Caída de receptores
  - Caída del coordinador
  - No recepción de mensajes
  - No recepción de ACKs



Agenda

- Nombres y direccionamiento
- Mensajes
- Grupos
- **Tiempo y relojes - Relojes Físicos**

  Tiempo y relojes - Relojes Lógicos

Tiempo | Introducción

Magnitud para medir duración y separación de eventos.

Se lo puede definir mediante una variable monotónica creciente:

- En sistemas de computación será discreta
- No necesariamente vinculada con la hora de la vida real

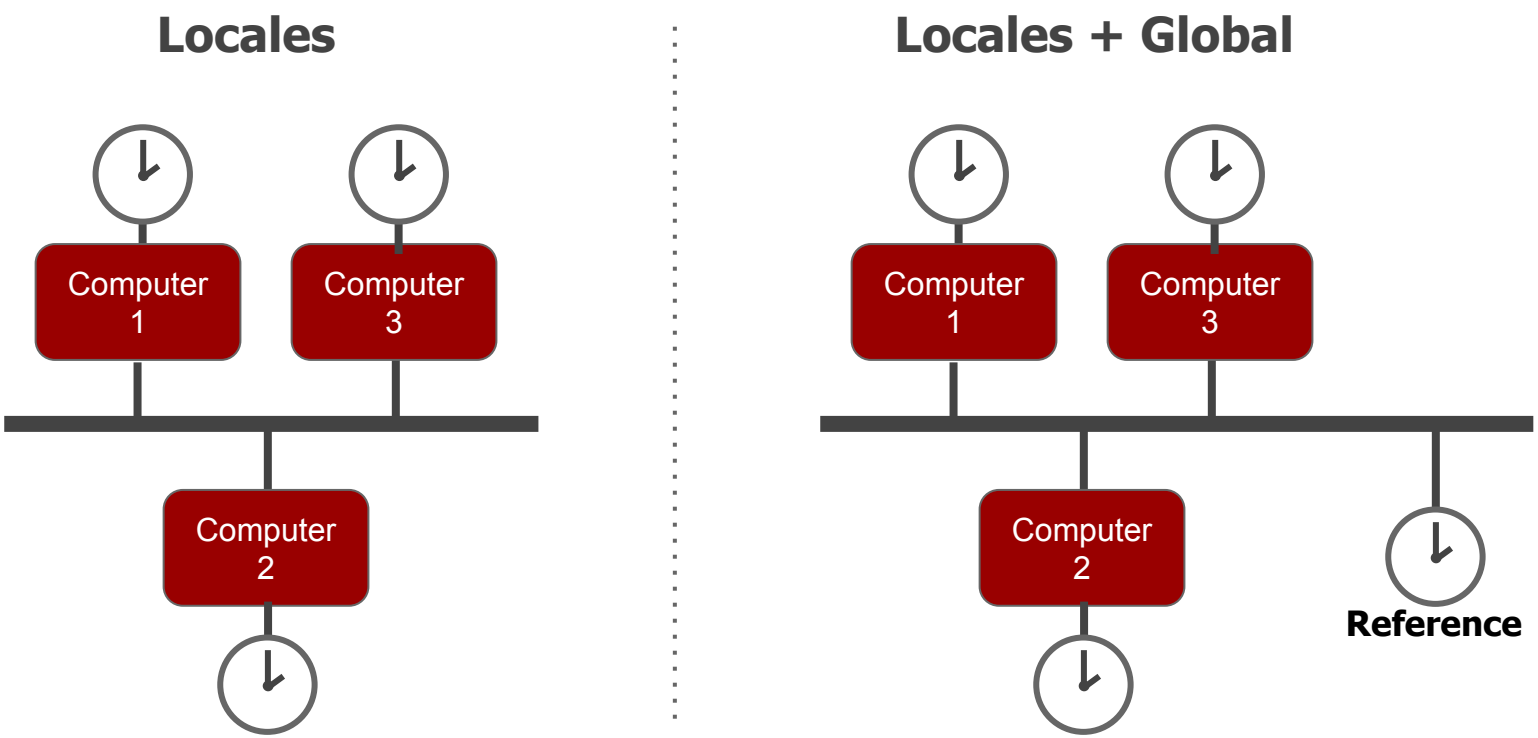
El tiempo siempre avanza, nunca retrocede.

Tiempo | Usos

La medición del tiempo permite:

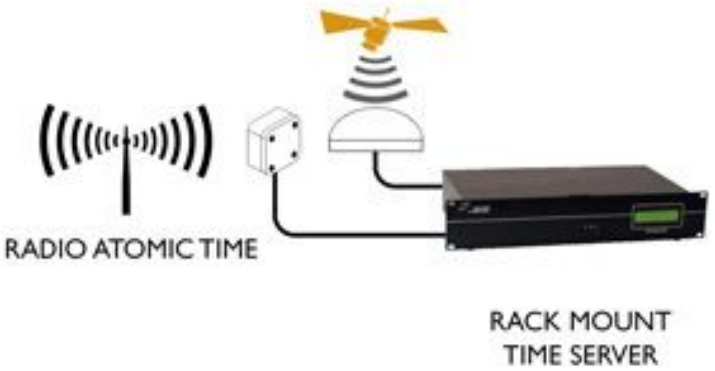
- Ordenar y sincronizar
- Marcar la ocurrencia de un suceso
  - **Timestamps**: puntos absoluto en la línea de tiempo
- Contabilizar duración entre sucesos
  - **Timespans**: intervalo en la línea de tiempo

Relojes Físicos | Locales vs Globales



Relojes Físicos

- Brindan la fecha y hora del día
- Pueden ser de cuarzo, atómicos, por GPS
- Los cambios de temperatura, presión y humedad descalibran relojes: *drift*



Source: galsys.co.uk

Relojes Físicos | Referencias Globales

GMT (Greenwich Mean Time)
- Basado en el tiempo de rot. Terrestre (t. astronómico)

UTC (Universal Time Coordinated)
- Basado en medición de relojes atómicos
- Dif. de +/-1 seg con GMT y recibe ajustes periódicos

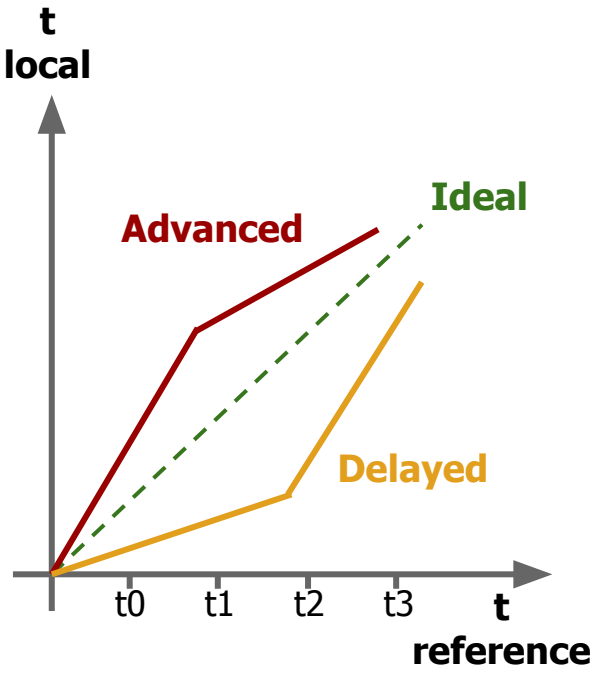
GPS time (Global Positioning System time)
- Basado en relojes atómicos en la Tierra
- No recibe ajustes pero permite ajustar satélites

TAI (Temps Atomique International)
- 200 relojes atómicos en 70 países
- No recibe ajustes astronómicos

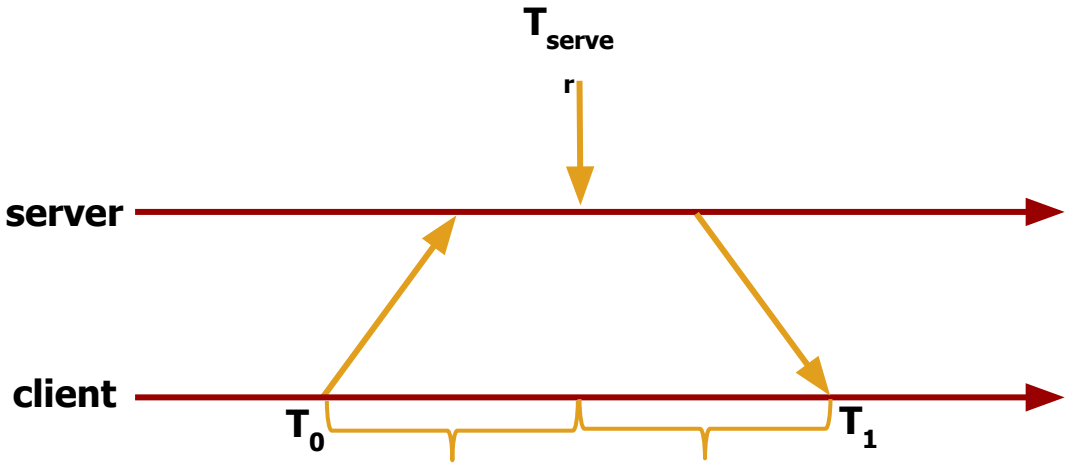
Source: wikipedia.org

Relojes Físicos | *Drift*

- Los relojes físicos no son confiables para su comparación por efecto del *drift*
- Hay que sincronizarlos periódicamente:
  - Medir el desvío respecto de un reloj de referencia (UTC, GPS, etc.)
  - Aplicar corrección o compensación lineal cambiando la frecuencia del reloj local
  - Nunca atrasar un reloj
- Hay que sincronizarlos al despertar la computadora



- Realiza una compensación del delay existente al obtener la medida de tiempo
  - T_0: Cliente envía request
  - T_1: Cliente recibe respuesta
  - Hipótesis: ***Delays en la red son constantes***



$$T_{new} = T_{server} + (T_1 - T_0) / 2$$

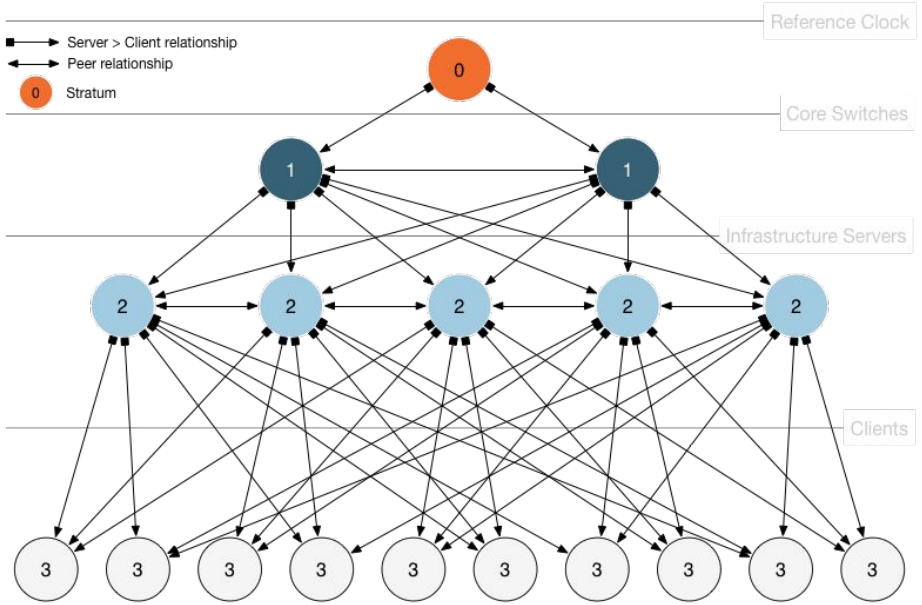
NTP | Objetivos

- **Sincronización**
  - Clientes (UTC) sincronizados aunque existan delays en la red
  - Análisis estadístico para filtrar data y obtener resultados de calidad
- **Alta disponibilidad**
  - Sobrevivir a caída a largas caídas de conectividad
  - Rutas redundantes
  - Servidores redundantes
- **Escalabilidad**
  - Gran número de clientes sincronizados de forma frecuente
  - Debe tener en cuenta efectos de drift

NTP | Estructura de Servidores

Basado en stratums (estratos)

- Estrato 0: Master clocks
- Estrato 1: Servidores conectados directamente a master clocks
- Estrato 2: Servidores sincronizados con servidores en estrato 1
- Estrato N: Servidores sincronizados con servidores en estrato N-1
- Servidores sincronizados entre sí con conexiones peer-to-peer
- Mensajes enviados de forma no *reliable* (UDP puerto 123)



NTP | Modos de Sincronización

- **Modo multicast/broadcast**
  - Usado en LANs de alta velocidad
  - Eficiente pero de baja precisión

- **Modo Cliente-Servidor (RPC)**
  - Grupos de aplicaciones se conectan formando un grupo
  - Aplicaciones entre sí no pueden sincronizarse

- **Modo Simétrico (Peer Mode)**
  - Peers sincronizados entre sí para proveer backup mutuo
  - Utilizado en estratos 1 y 2

Agenda

- Nombres y direccionamiento
- Mensajes
- Grupos
- Tiempo y relojes - Relojes Físicos
- **Tiempo y relojes - Relojes Lógicos**

Evento

- Suceso relativo al proceso Pi que modifica su estado

Estado

- Valores de todas las variables del proceso Pi en un momento dado

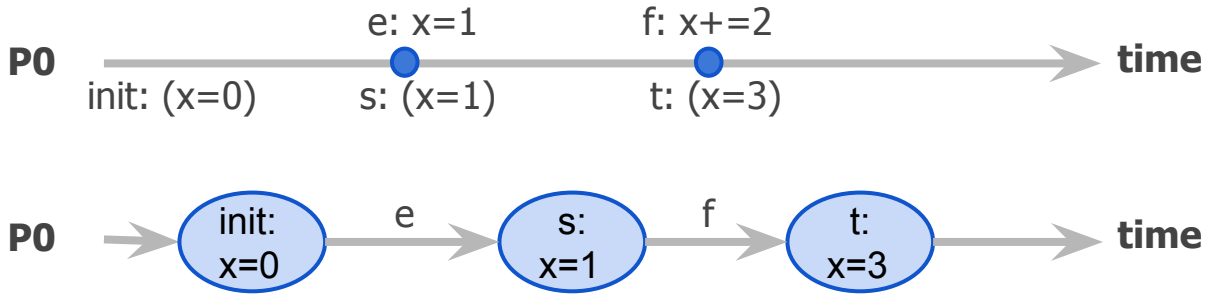
Relación 'ocurre antes' *(happened before)*:

- Relación de causalidad entre eventos o estados tales que:
  - **a** $\rightarrow$ **b,** si **a**,**b** pertenecen al mismo proceso Pi y **a** ocurre antes de **b**
  - **a** $\rightarrow$ **b,** si **a** es un evento de Pi, **b** es un evento de Pj, **a** es el envío del mensaje '**m**' a Pj y **b** es la recepción del mensaje '**m**' desde Pi
  - **a** $\rightarrow$ **c**, si **a** $\rightarrow$ **b** y **b** $\rightarrow$ **c** (transitividad)

Dado S, el conjunto de todos los estados locales posibles del sistema y $\rightarrow$ la relación temporal de implicancia 'ocurre antes' (o *happened before*), un reloj lógico es una función C monotónica creciente que mapea estados con un número Natural y garantiza:

$$\forall s, t \in S : s \rightarrow t \Rightarrow C(s) < C(t)$$

Ej.:

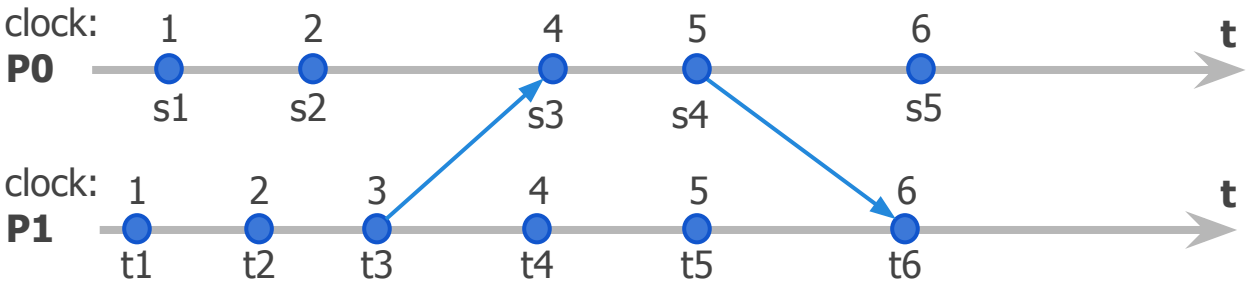


$s \rightarrow t$

$C(s) < C(t)$

```
Pi:
  var:
    c := 0
  send event (s, send, t):
    //include t.c in msg
    //send msg
    t.c := s.c + 1
```

```
receive event (s, receive(u), t):
  //receive msg from u
  t.c := max(s.c, u.c) + 1
internal event (s, internal, t):
  t.c := s.c + 1
```



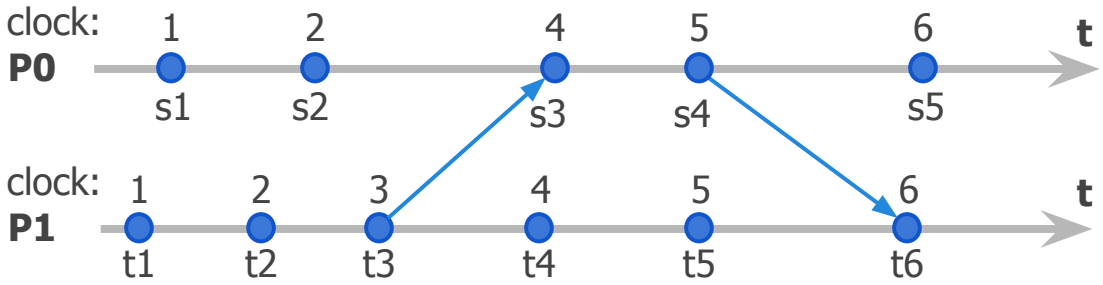
Los relojes de Lamport garantizan:

$$s \rightarrow t \Rightarrow C(s) < C(t)$$

pero:

$$C(s) < C(t) \not\Rightarrow s \rightarrow t$$

Ej.:



$$C(t1) < C(s4)$$
$$t1 \rightarrow s4$$

$$C(s1) < C(t4)$$
$$s1 \not\rightarrow t4$$

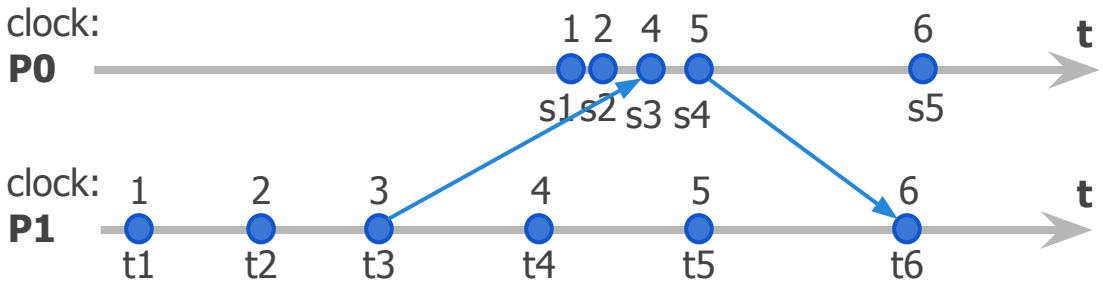
Los relojes de Lamport garantizan:

$$s \rightarrow t \Rightarrow C(s) < C(t)$$

pero:

$$C(s) < C(t) \not\Rightarrow s \rightarrow t$$

Ej.:



$$C(t1) < C(s4)$$
$$t1 \rightarrow s4$$

$$C(s1) < C(t4)$$
$$s1 \not\rightarrow t4$$

Un vector de relojes es el mapeo de todo estado del sistema compuesto por k procesos, con un vector de k números Naturales y garantiza:

$$\forall s, t \in S : s \rightarrow t \Leftrightarrow s.v < t.v$$

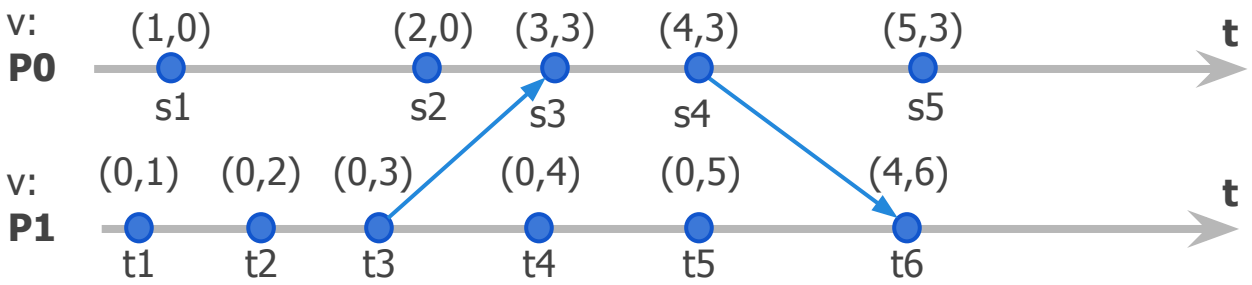
donde s.v y t.v son los vectores de k componentes para los estados s y t respectivamente y

$$s.v < t.v \Leftrightarrow \forall\ k : \ s.v[k] \ \leq \ t.v[k] \ \wedge$$
$$\exists\ j : \ s.v[j] \ < \ t.v[j]$$

Vector de Relojes | Implementación

```
Pi:
  var:
    v[i] := 1
    v[j] := 0, con i<>j
  send event (s, send, t):
    //include t.v in msg and send
    t.v := s.v
    t.v[i] := t.v[i] + 1
```

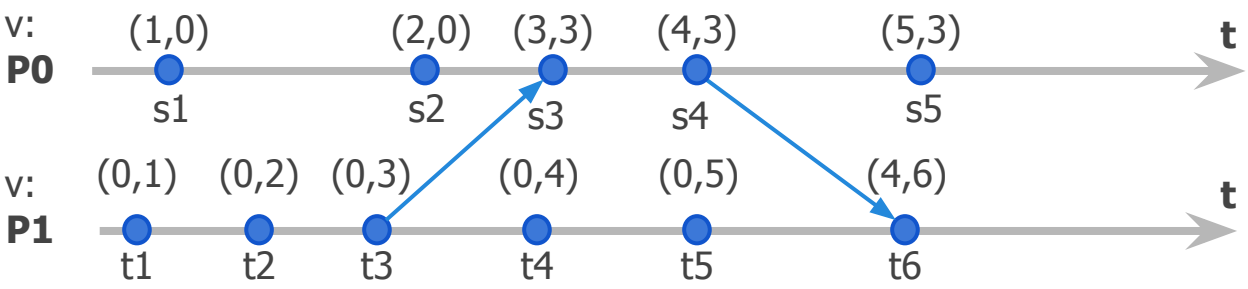
```
receive event (s, receive(u), t):
  //receive msg from u
  for j:= 1 to N:
    t.v[j] := max(s.v[j], u.v[j])
  t.v[i] := t.v[i] + 1
internal event (s, internal, t):
  t.v := s.v
  t.v[i] := t.v[i] + 1
```



Vector de Relojes | Implementación

```
Pi:
  var:
    v[i] := 1
    v[j] := 0, con i<>j
  send event (s, send, t):
    //include t.v in msg and send
    t.v := s.v
    t.v[i] := t.v[i] + 1
```

```
receive event (s, receive(u), t):
  //receive msg from u
  for j:= 1 to N:
    t.v[j] := max(s.v[j], u.v[j])
  t.v[i] := t.v[i] + 1
internal event (s, internal, t):
  t.v := s.v
  t.v[i] := t.v[i] + 1
```

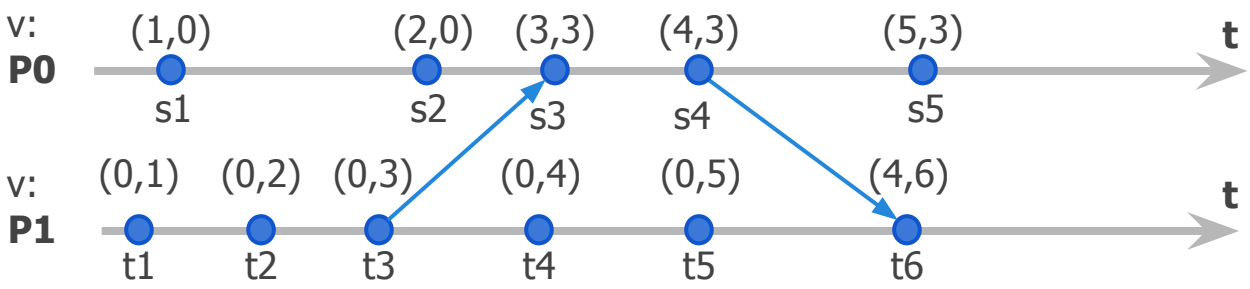


V(t1) < V(s4)
(0,1) < (4,3)
i=0 => 0 < 4
i=1 => 1 < 3
=> t1 -> s4

Vector de Relojes | Implementación

```
Pi:
  var:
    v[i] := 1
    v[j] := 0, con i<>j
  send event (s, send, t):
    //include t.v in msg and send
    t.v := s.v
    t.v[i] := t.v[i] + 1
```

```
receive event (s, receive(u), t):
  //receive msg from u
  for j:= 1 to N:
    t.v[j] := max(s.v[j], u.v[j])
  t.v[i] := t.v[i] + 1
internal event (s, internal, t):
  t.v := s.v
  t.v[i] := t.v[i] + 1
```



V(t4) < V(s1)
(0,4) < (1,0)
i=0 => 0 < 1
i=1 => 4 < 0
=> t1 != s4

Bibliografía

- Garg, V., Elements of Distributed Computing, 1st. Ed. Wiley IEEE Press, 2002.
  - Capítulo 2: Model of a computation
  - Capítulo 3: Logical Clocks
- P. Verissimo, L. Rodriguez: Distributed Systems for Systems Architects, Kluwer Academic Publishers, 2001.
  - Capítulo 2.1: Naming and addressing
  - Capítulo 2.4: Group Communication
  - Capítulo 2.5: Time and Clocks
- IPSES Scientific Electronics - Standard of Time Definition
  - https://www.ipses.com/eng/In-depth-analysis/Standard-of-time-definition