

Definición de clases

```
ldapsearch -H ldaps://info.fi.upm.es -x -b cn=subschema -s base objectClasses
```

```
objectclass ( 2.5.6.0 NAME 'top' ABSTRACT MUST objectClass )
```

```
objectclass ( 2.5.6.6 NAME 'person' SUP top STRUCTURAL  
MUST ( sn $ cn )  
MAY ( userPassword $ telephoneNumber $ seeAlso $ description ) )
```

```
objectclass ( 2.5.6.7 NAME 'organizationalPerson' SUP person STRUCTURAL  
MAY ( title $ x121Address $ registeredAddress $ destinationIndicator $  
preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $  
telephoneNumber $ internationaliSDNNumber $  
facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $  
postalAddress $ physicalDeliveryOfficeName $ ou $ st $ l ) )
```

```
objectclass ( 1.3.6.1.4.1.1466.344 NAME 'dcObject'  
DESC 'RFC2247: domain component object'  
SUP top AUXILIARY MUST dc )
```

Definición de atributos

```
ldapsearch -H ldaps://info.fi.upm.es -x -b cn=subschema -s base attributetypes
```

```
attributetype ( 2.5.4.41 NAME 'name'
```

```
    EQUALITY caseIgnoreMatch
```

```
    SUBSTR caseIgnoreSubstringsMatch
```

```
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )
```

```
attributetype ( 2.5.4.3 NAME ( 'cn' 'commonName' ) SUP name )
```

```
attributetype ( 0.9.2342.19200300.100.1.25
```

```
    NAME ( 'dc' 'domainComponent' )
```

```
    DESC 'RFC1274/2247: domain component'
```

```
    EQUALITY caseIgnoreIA5Match
```

```
    SUBSTR caseIgnoreIA5SubstringsMatch
```

```
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE )
```

Creación de un nuevo esquema

- Sólo si es estrictamente necesario
 - Nunca cambiar comportamiento de objetos/atrib. estándar
- 2 alternativas para extender clase ya existente
 - Crear nueva clase estructural derivada de clase existente
 - Permite mejor control: se pueden definir reglas de contenido/estructura
 - Pero requiere eliminar y reinsertar todos los objetos existentes
 - Crear clase auxiliar derivada de *top* e incluirla en definición de objetos
 - Se puede añadir directamente usando *Modify*
- C. auxiliar también permite incluir atrib. en objetos de \neq clases
 - p.e. fecha de alta en organización, tanto personas como dispositivos

Extracto de esquema del LDAP de FI

```
attributetype ( 1.3.6.1.4.1.7547.1.19.10.4.2.4 NAME 'fiRelationShip' DESC 'Relacion del usuario con
la Escuela' EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX
1.3.6.1.4.1.1466.115.121.1.15 )
attributetype ( 1.3.6.1.4.1.7547.1.19.10.4.2.1 NAME 'fiGender' DESC 'Sexo de la persona (ISO
5218)' EQUALITY integerMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE )
attributeTypes: ( 1.3.6.1.4.1.7547.1.19.10.4.2.5 NAME 'fiTeaching' DESC 'Asignaturas impartidas por
el profesor' EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX
1.3.6.1.4.1.1466.115.121.1.15{20} )
objectclass ( 1.3.6.1.4.1.7547.4.3.1.2 NAME 'irisPerson' DESC 'Persons inside the IRIS community'
SUP top AUXILIARY MAY ( sn1 $ sn2 $ irisPersonalTitle $ irisPersonalUniqueID $
irisUserEntitlement $ irisUserPrivateAttribute $ irisUserStatus $ irisMailHost $
irisMailRoutingAddress $ irisMailbox $ irisMailMainAddress $ irisMailAlternateAddress $
irisUserPresenceID $ irisClassifCode ) )
objectclass ( 1.3.6.1.4.1.7547.1.19.10.4.1.1 NAME 'fiPerson' DESC 'Persona perteneciente a la
Facultad de Informatica (UPM)' SUP irisPerson AUXILIARY MUST ( uid $ mail )
MAY ( fiPwdChangedOperTime $ fiMailQuotaSize $ fiGender $fiRelationShip ) )
objectClasses: ( 1.3.6.1.4.1.7547.1.19.10.4.1.3 NAME 'fiEmployee' DESC 'Empleado de la Facultad
de Informatica (UPM)' SUP fiPerson AUXILIARY MAY fiTeaching)
```

Índice

- Introducción
- Servicio de nombres
 - Estudio de un ejemplo práctico: DNS
- Servicio de directorio
 - Estudio de un ejemplo práctico: LDAP
- Descubrimiento de servicios
 - Gestión de nombres en sistemas móviles/ubicuos
 - Auto-configuración
 - Servicios de descubrimiento de servicios

Gestión de nombres en SD móvil/ubicuo

- Computación ubicua → “invisible” → auto-configuración
- Sistemas dinámicos, “espontáneos” y volátiles:
 - Nodos entran y salen de un SD: de un “espacio inteligente” (EI)
 - Mi cámara digital y yo entramos/salimos en habitación de hotel
 - Un vehículo entra/sale de EI controlado por un semáforo inteligente
- Descubrimiento de servicios clave para computación ubicua
- Nodo entra en E:
 - Se autoconfigura y descubre, y es descubierto, por nodos restantes
 - Si proveedor de servicios, hace conocerlos a quiénes le interesen
 - Si consumidor de serv., descubre los de otros nodos que le interesen
 - *Plug & play* de servicios/dispositivos en SD
 - Notificación de aparición de nuevo servicio a nodos interesados

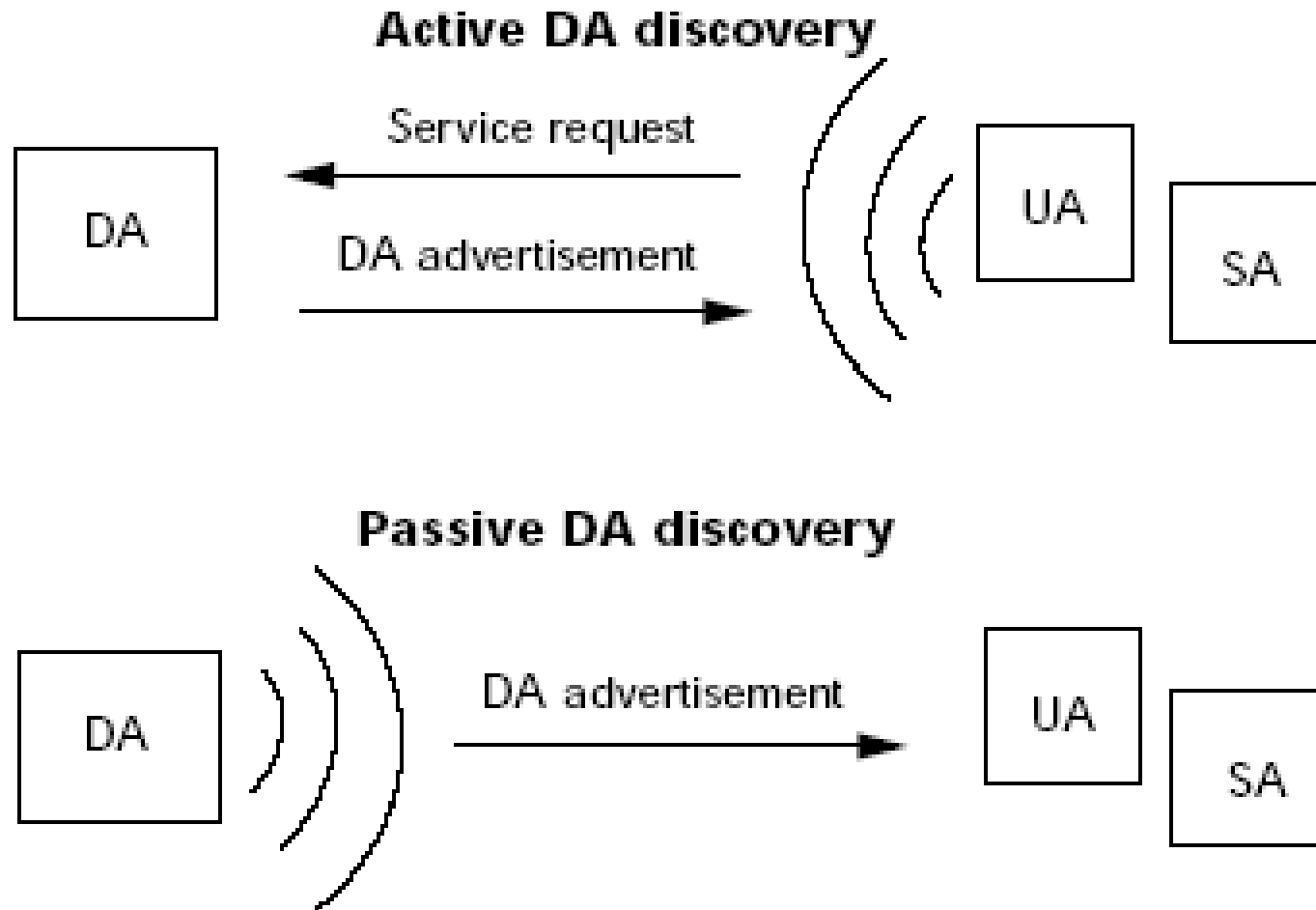
Auto-configuración

- Nodo necesita IP y, en ocasiones, también nombre DNS
- Obtención de dirección IP (e info asociada: máscara, *router*,...)
 - Uso de DHCP: Nodo *broadcast* petición de dirección IP
 - Servidor DHCP asigna dirección IP con *lease* asociado
 - Si DHCP no disponible (por volatilidad o limitación de recursos)
 - *Dynamic Configuration of IPv4 Link-Local Addresses* (RFC 3927)
 - Nodo elige su dir. IP y usa ARP para comprobar que no está en uso
 - Si conflicto, selecciona otra
- Obtención de nombre DNS (si requerido)
 - Uso de DNS con protocolo de actualización: *Dynamic-DNS*
 - Si DNS no disponible (por volatilidad o limit. recursos): *Multicast-DNS*
 - Consultas a dominio `.local.` usan *multicast* dir. fija
 - Nodo correspondiente responde (con *unicast* o *multicast*)

Servicios de descubrimiento de servicios

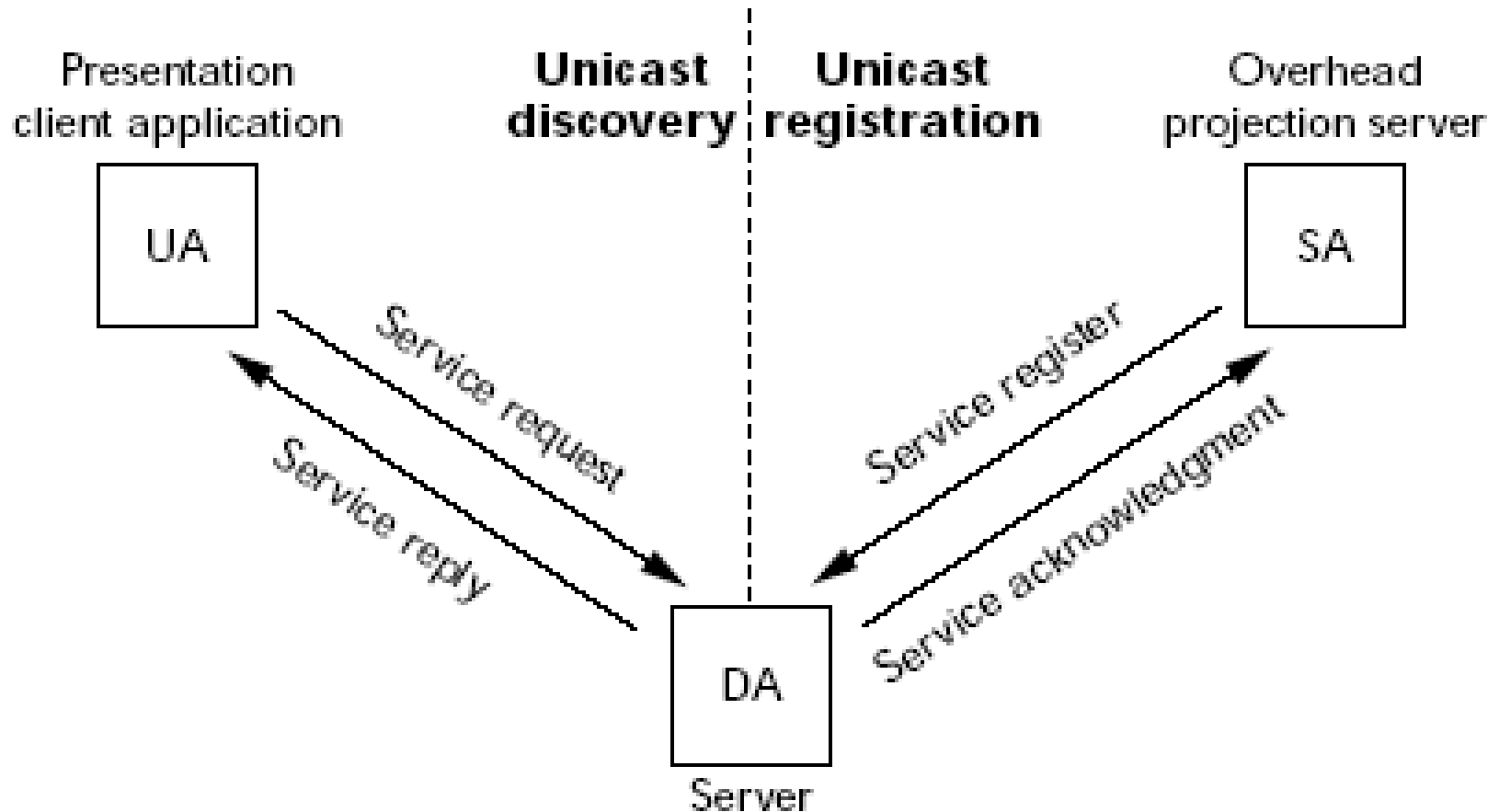
- Con DA (servidor de descubrimiento):
 - UA (cliente) y SA (servidor) deben localizar DAs
 - Localización pasiva: DA *multicast* a dirección fija
 - Localización activa: UA/SA *multicast* a dirección fija
 - SA registra servicio mediante *unicast* en DAs localizados
 - UA consulta mediante *unicast* a alguno de los DAs localizados
- Sin DA: *pull* versus *push*
 - *Push*: SA *multicast* anuncio de servicio; UAs guardan esa info.
 - *Pull*: UA *multicast* petición; SA la recibe y responde
- Esquema híbrido
 - Mientras no haya ningún DA: *push* o *pull*
 - Cuando aparece un DA, pasa a modo con DA
 - Si desaparece DA: vuelta al primer punto

Descubrimiento de DA en SLP



Service Location Protocol: Automatic Discovery of IP Network Services. Erik Guttman

Registro y búsqueda de servicio en SLP



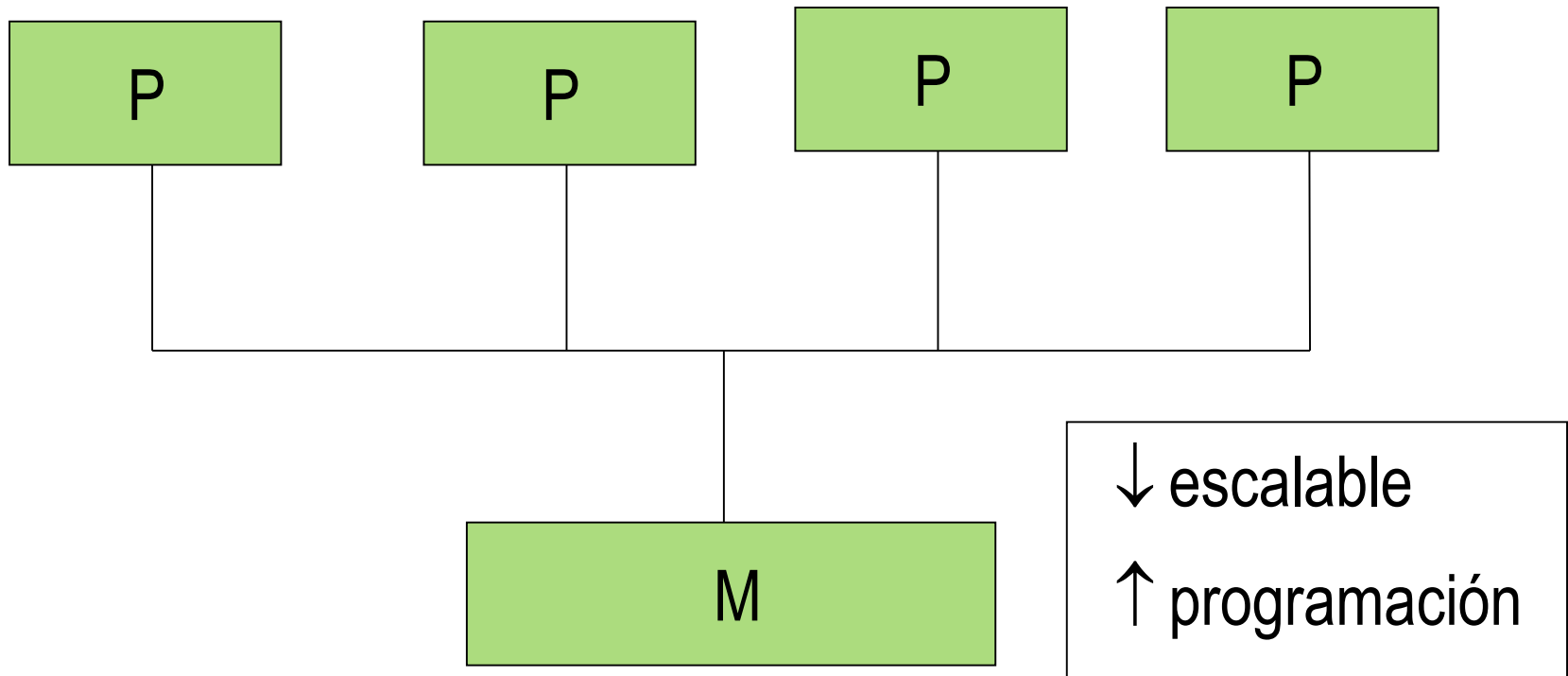
Service Location Protocol: Automatic Discovery of IP Network Services. Erik Guttman

Sistemas Distribuidos

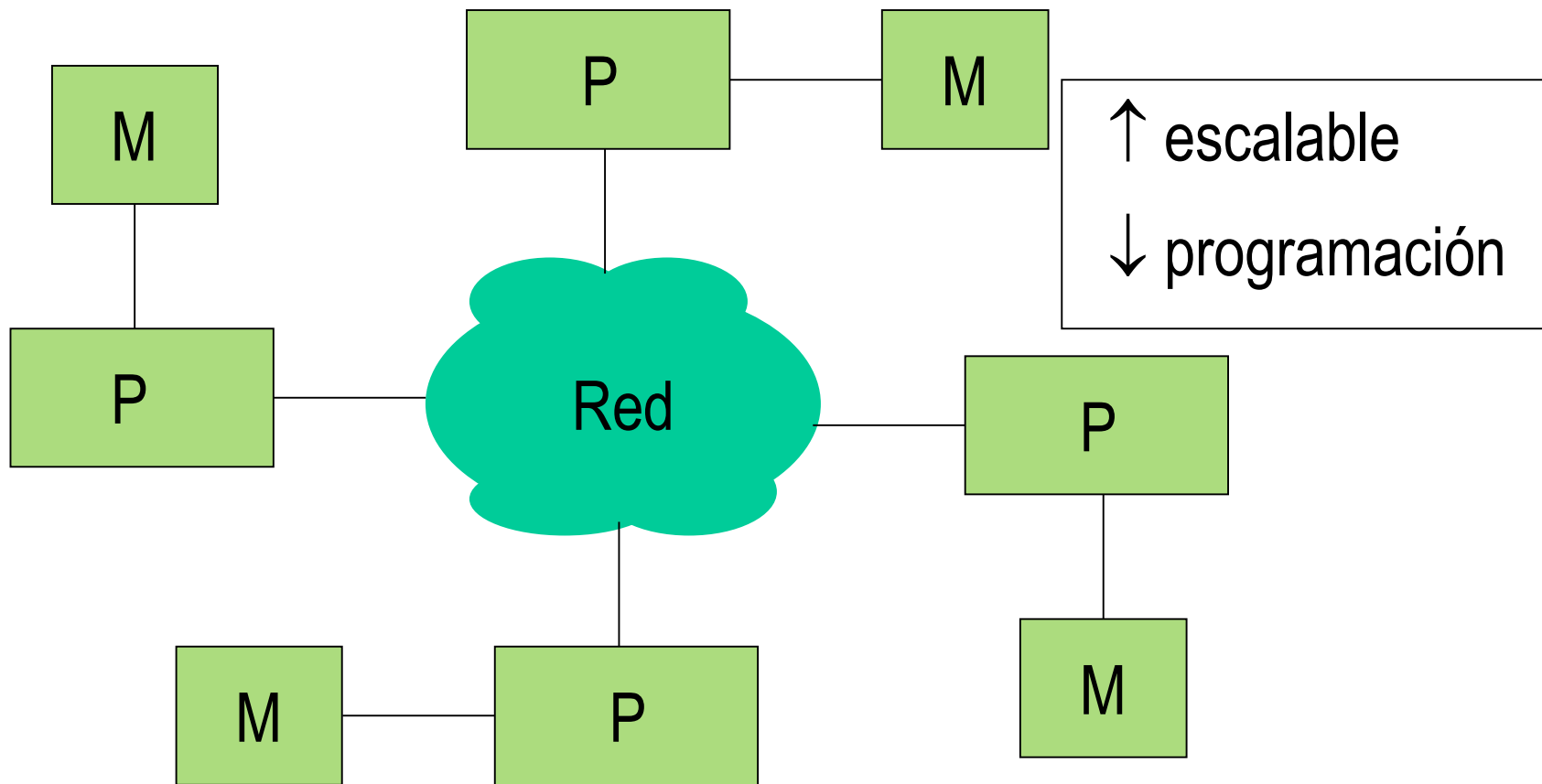
**Memoria
compartida
distribuida**

**(DSM, *Distributed
Shared Memory*)**

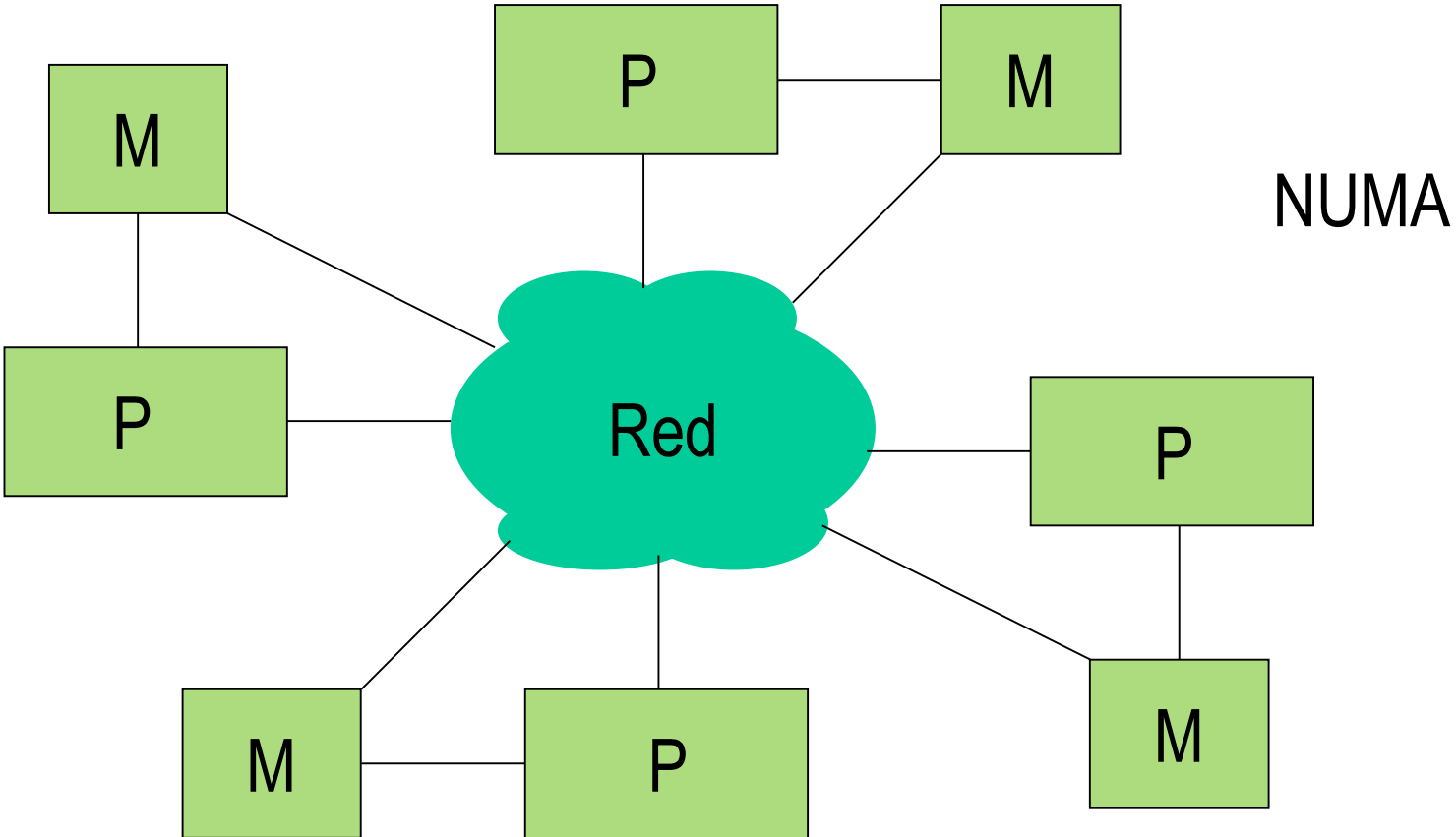
Memoria centralizada y compartida



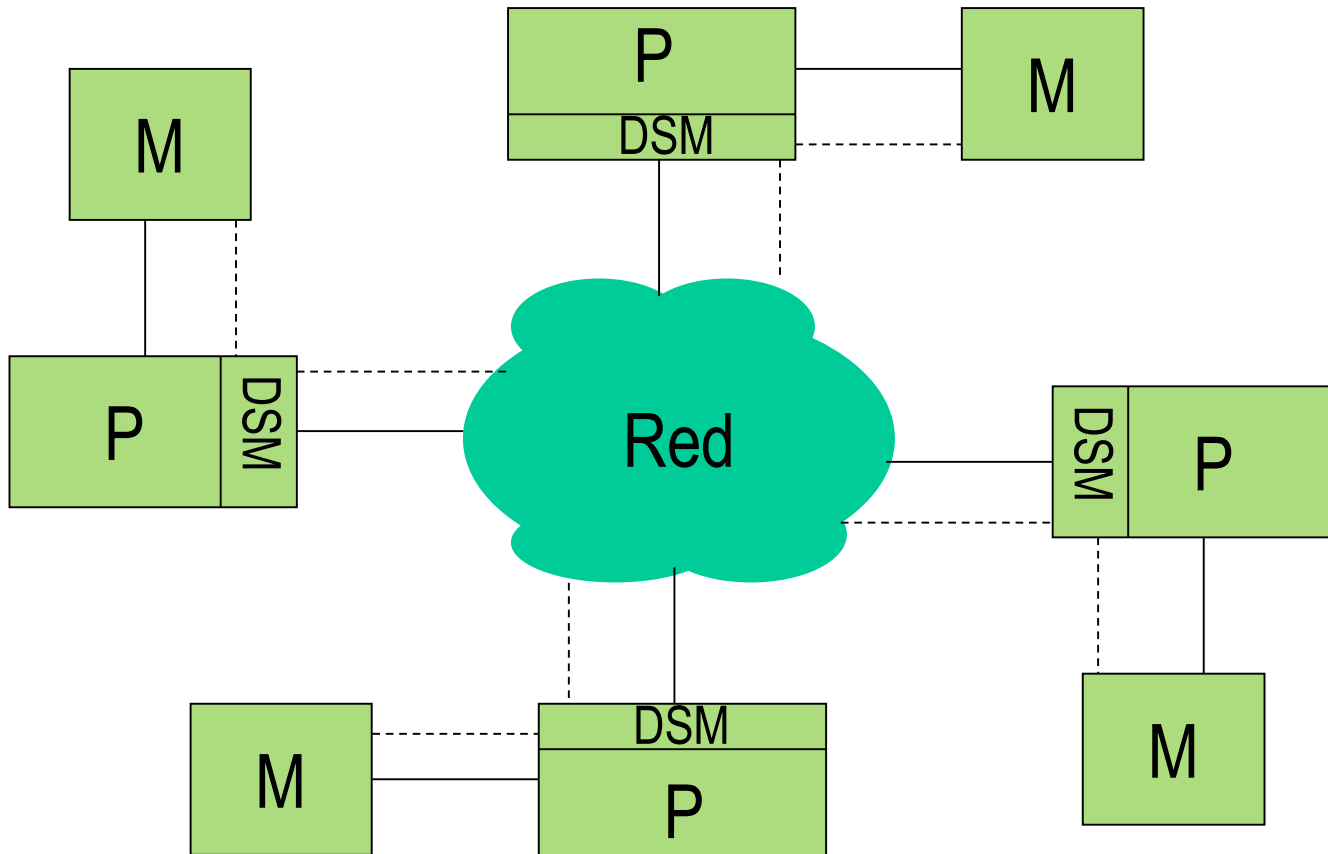
SD: Memoria distribuida y privada



DSM HW: Memoria distribuida y compartida



DSM SW: Memoria distribuida y compartida



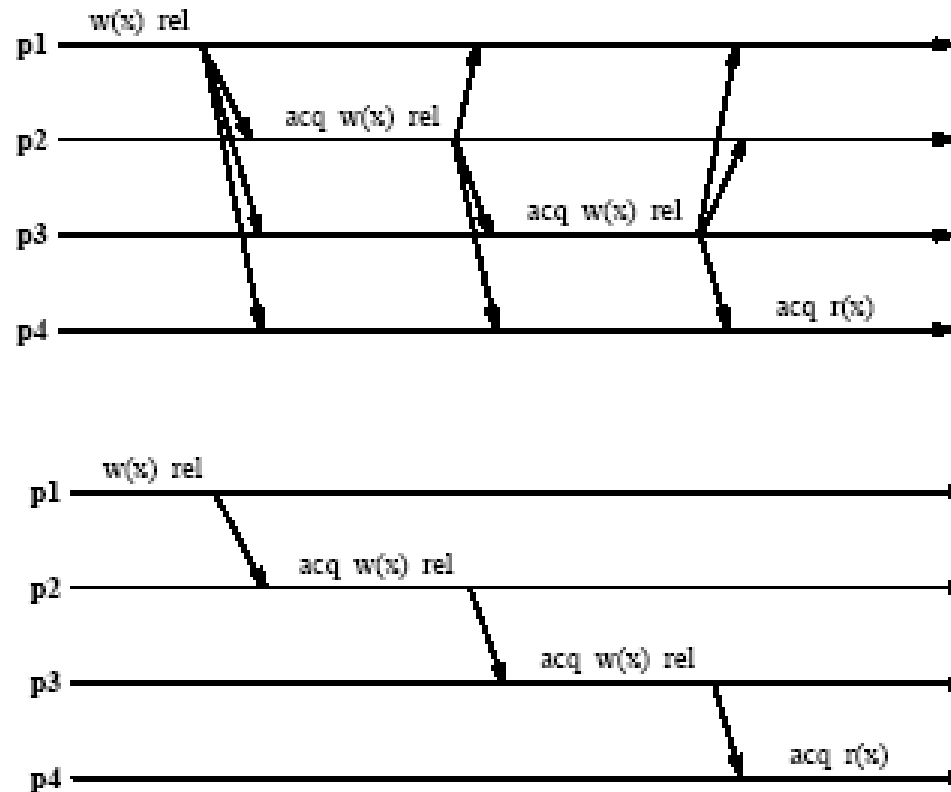
Introducción

- Multiprocesadores con mem. compartida vs. s. distribuidos:
 - HW más complejo y menos *escalable*; SW más sencillo y conocido
- Modelo de programación en sistemas con mem. compartida
 - Llamadas a procedimiento (o invocación de métodos)
 - Comunicación mediante datos compartidos
 - Sincronización mediante semáforos o mecanismos equivalentes
- Modelo de programación tradicional en s. distribuidos
 - Paso de mensajes para comunicación y sincronización
- Nuevo modelo de programación en SD: RPC (o RMI) + DSM
 - Ejecutar en SD aplicación paralela basada en m.comp.
- Dos implementaciones alternativas:
 - Basada en m.virtual (VM-DSM) o en *runtime* del lenguaje (RT-DSM)

Modo de operación

- Si cada escritura en memoria genera mensaje: DSM inviable
- ¿Cómo lograr implementación relativamente eficiente?
 - Programa bien construido → no condiciones de carrera
 - Accesos conflictivos a datos deben usar sección crítica (SC)
 - Sólo asegurar coherencia de los datos dentro de la SC
 - Articular la propagación de los cambios con entrar o salir de la SC
- 3 esquemas:
 - *Eager Release Consistency* (ERC)
 - Al salir de SC se propagan a todos cambios en variables compartidas
 - *Lazy Release Consistency* (LRC)
 - Al entrar en SC se contacta con último proceso que estuvo en SC
 - Se le solicitan todos los cambios sobre las variables compartidas
 - *Entry Consistency* (EC): similar a LRC pero
 - Toda variable compartida está asociada a un cerrojo
 - Al entrar SC de un cerrojo, se contacta con último proceso que lo usó
 - Se le solicitan sólo cambios sobre v. compartidas asociadas a ese cerrojo

ERC vs. LRC



Lazy Release Consistency for Distributed Shared Memory.

Tesis de Peter Keleher, 1995