

# Naming in Distributed Systems

Unique identifiers **UIDs** e.g. 128 bits

- are never reused
- refer to the same thing at all times, or to nothing at all

UIDs should be location-independent! Can the named object be moved?

Pure and impure names ( Needham )

pure names

- the name itself yields no information, and commits the system to nothing
- it can only be used to compare with other similar names e.g. in table look-up

impure names

- the name yields information,
- and commits the system to maintaining the context in which it can be resolved

## Examples of impure names

[jmb25@cl.cam.ac.uk](mailto:jmb25@cl.cam.ac.uk)  
[jeanb@lcs.mit.edu](mailto:jeanb@lcs.mit.edu)

name of a person, registered in a DNS domain

name of a person, registered in another DNS domain

another or the same person?

[puccini.cl.cam.ac.uk](http://puccini.cl.cam.ac.uk)

name of a machine, registered in a DNS domain

*(disc-pack-ID, object-ID)* Bad idea from history of naming files and directories.  
Seemed efficient until the objects had to be moved

*(host-ID, object-ID)* OK, it's impure ... but how are pure names generated in a DS?

We must not have centralised name allocation.

*(host-ID, object-ID)* has been used (badly) in middleware,  
and has made the objects unmoveable.

It could be used to generate pure names,

*if we do not make use of the separate fields.* Typical example:

32-bit host-ID	96 bit object-ID
----------------	------------------

# Unique names

Both pure and impure names can be unique.

Uniqueness can be achieved by using:

for impure names:

- hierarchical names: scope of uniqueness is level in hierarchy  
( uniqueness is within the names in the directory in which the name is recorded )

for pure names

- a bit pattern: flat, system-wide uniqueness,

## Problems with pure names:

- where to look them up to find out information about them?
- how do you know that an object does not exist? How can a global search be avoided?
- how to engineer uniqueness reliably in a distributed system?  
centralised creation of names? As discussed above, (*host-ID, object-ID*)?

## Problems with impure names:

- how to restructure the namespace  
e.g. when objects move about or when companies restructure

## Examples of (pure/impure) names - unique identification

UK national insurance - allocated on employment

US Social Security - allocated on employment

Passport

Driving licence

Services: RAC, AA, AAA(US), AAA(Aus)

Credit cards

Bank accounts

Utilities' customer numbers: gas/electricity/water/phone

Charity members

Loyalty card members

For the above examples:

- Is the structure explicit or implicit?
- Is allocation centralised or distributed?
- What is the resolution context?

## More examples of names - unique identification?

UK health service (NHS) ID – allocated to every citizen at birth

but hospitals still use local patient numbers per treatment, with names recorded at the time

e.g. is J. Ken Moody the same person as John K. Moody?

can medication information be used interchangeably?

Professional societies: BCS, ACM, IEEE

*aside: A database key MUST be unique, e.g. Jean Bacon is not unique*

*Jean Bacon in Connecticut USA, and I were allocated the same IEEE membership number*

*– our records had been merged: Cambridge work, Connecticut home. I got renewal notice.*

*IEEE membership would not believe me until I found her and we emailed them together.*

e.g. from the Computer Lab, Jatinder Singh is held up at immigration because he shares the name and date of birth of another Jatinder Singh

Middleware 2009 keynote address:

Professor Hector Garcia-Molina, Stanford

Entity Resolution – Glue for Middleware

*available from the MW09 program, and the course materials page*



## Telephone company analogy – wired service

Geographically partitioned distributed naming database.

Electronic version is current, paper directories are an official cache

Frequency of update (some years ago):

Cambridge area 1,000,000 entries, 5,000 updates a week

Given a name e.g. ( Yudel Luke ), or ( Yudel Luke, 3 Acacia Drive ) which directory to use?

- don't know where to look up pure names

Lookup doesn't yield useable information:

Call# -> unobtainable, where # is from the official cache (paper directories)

we detect out-of-date values, call directory enquiries, cache unofficially until new directory

Call# -> unobtainable, for a number that we know and use often, or from personal address book

redial, report fault, check official cache, ask social network if X has moved phone

Can't find an entry in the official cache ( exact matching required )

e.g. Phillips - check spelling – Philips

e.g. try acronyms S.S. for Social Services

BT offer a web service [www.thephonebook.com](http://www.thephonebook.com) (name and address -> number )

only offers exact matching e.g. Philips not suggested for Phillips (do you mean?)

increasingly, search engine approaches are used to augment directory lookup

# Name spaces and naming domains

In general, provide clients with values of attributes of named objects

## Name space

- the collection of valid names recognised by a name service
  - a precise specification is required, giving the structure of names
- e.g. *ISBN:1-234567-89-1* namespace identifier, namespace-specific string  
*/a/b/c/d* file system pathname, variable length, hierarchical  
*puccini.cl.cam.ac.uk* DNS machine name – see case study below  
128 bit system-wide OS port name for Mach OS, system-wide UID

## Naming domain

a name space for which there exists a single overall administrative authority  
for assigning names within it  
this authority may delegate name assignment for nested sub-domains ( see DNS below )

# Name resolution - binding

## Name resolution or binding

obtaining a value for an attribute of the named object that allows the object to be used

Late binding is considered good practice  
Programs should contain names, not addresses



A machine may fail and the service moved to another machine.  
Your local agent may cache resolved names for subsequent use,  
and may expire values based on timestamps (TTL time-to-live)

Cached values are always used “at your own risk”.  
They should not be embedded in programs.

If cached values don't work, the lookup has to be repeated.  
Lookup may be iterative for large-scale systems – see later.



## Names, attributes and values stored in a name service

	object type	attribute list
example:	user	login-name, mailbox-hosts(s)
	computer	architecture, OS, network-address, owner
	service	network-address, version#, protocol
	group	list of names of members
	alias	canonical name
	directory?	list of hosts holding the directory (may be held in a separate structure rather than as a type of name, as here )

Directories are likely to be replicated for scalability, fault-tolerance, efficiency, availability

Directory names resolve to a list of hosts plus their addresses to avoid an extra lookup per host

Attribute-based (inverse) lookup may be offered –

A **YELLOW PAGES** style of service for object discovery e.g. **X.500, LDAP**

Too much information can be dangerous – useful for hackers

# Names, attributes and values - examples

*object type -> list of attribute names*

name service holds:

*object-type, object-name -> list of attribute values*

You can acquire a standard directory service e.g. LDAP and use it to store whatever your service/application needs

querying:

*object-type, object-name, attribute-name -> attribute value*

computer, puccini.cl.cam.ac.uk, location -> IP-address

user, some-user-name, public-key -> PK bit pattern

checking:

*object-type, object-name, attribute-name, attribute value -> yes/no*

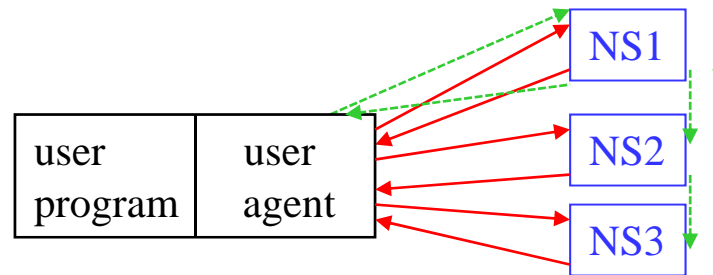
ACL, filename, some-user-name, write-access -> yes/no

Attribute-based (inverse) lookup:


*object-type, attribute-name, attribute value -> list of object-names*

computer, OS version#, OS version# value -> list of computers

## Iterative name resolution




user agent (UA) starts from the root address of the name service, or tries some well-known sub-tree root, e.g. the location of the *uk* directory may be used by agents in the UK

To resolve *cl.cam.ac.uk* the UA, as in 

looks up *ac*'s address in *uk*, then looks up *cl*'s address in *ac*

The UA will cache resolved names as hints for future use, see slide 6

Alternatively, any name server will take a name, resolve it and return the resolved value, as in 

The client may be able to choose, e.g. select "recursive" in DNS

Engineering optimisations:

- use of caching at UA and at directories

- try cached values first

# Name services - examples

**DNS** Internet **Domain Name Service**, see below

**Grapevine**: Xerox PARC early 1980's, *Birrell, Levin, Needham, Schroeder CACM 25(1) 1982*

two-level naming hierarchy e.g. *name@registry birrell@pa*

primarily for email, but also gave primitive authentication and access control

( check password as attribute of user, check ACL )

any Grapevine server would take any request from a GV user agent

**Clearinghouse**, Xerox PARC, Oppen and Dalal 1983

ISO standard based on an extension of Grapevine

three-level hierarchy

**GNS Global Name Service**, DEC SRC, *Lampson et al. 1986* - see below

full hierarchical naming

support for namespace restructuring

**X.500** and **LDAP**, see below

Name services in **Middleware**

CORBA: naming and (interface) trading services, Java JNDI, Web W3C: UDDI

Allow registration of names/interfaces of externally invocable components with interface references and attributes such as location

May offer separate services for: *name -> object-reference, object-reference -> location*

## Case Study: DNS - Internet Domain Name Service

Before 1987 the whole naming database was held centrally and copied to selected servers periodically  
The Internet had become too *large scale* and a *distributed, hierarchical scheme* was needed  
( *Paul V. Mockapetris, 1987* )



What does DNS name?

In practice, the objects are:

- computers
- servers such as mail hosts

The directory structure (resolution context) is captured as:

- domains

# DNS – Definition of names

## Definition of names

hierarchy of components (labels), highest level in hierarchy is last component, total max 255 chars

**label:** max 63 chars, case insensitive, must start with a letter, only letters, digits, hyphens allowed

final label of a fully qualified name can be:

**3-letter code:** type of hosting organisation

*edu, gov, mil* are still US-based, others, e.g. *com, net, org, int*, can be anywhere

**2-letter code:** country of origin defined by ISO e.g. *uk, fr, ie, de*, ...

*final 2-letter label doesn't always imply country of location of host, but where the host was registered*

e.g. [www.yahoo.co.uk](http://www.yahoo.co.uk) is in S germany

e.g. ISO have defined many small “country of origin” domains such as *to, cc, bv*, ...

**arpa:** for inverse lookup, e.g. *128.232.56.7.in-addr.arpa*

Examples of domain names:

- mit.edu*
- cl.cam.ac.uk*
- cs.tcd.ie*
- tu-darmstadt.de*

# DNS

Computers using DNS are grouped into **zones** e.g. *uk, cam*

Within a zone, management of nested sub-domains can be delegated

e.g. *cl* is managed locally by the domain manager who adds names to a local file

Each zone has a **primary name server** that holds the master list for the zone. Secondary name servers hold replicas for the zone.

**Queries** can relate to individual hosts or zones/domains, examples:

query		response
<b>A</b>	computer name	-> IPv4 address
<b>AAAA</b>	computer names	-> IPv6 address
<b>MX</b>	mail host for domain	-> list < host, preference, IP address > <i>includes mail hosts for detached computers</i>
<b>NS</b>	DNS servers for a domain	-> list < host, authority?Y/N, IP address >

## DNS - Unix examples

```
$ /usr/bin/nslookup
> set q = A
> cosi.cl.cam.ac.uk
Address: 128.232.8.106
```

```
> www.cl.cam.ac.uk
Address: 128.232.0.20
```

```
> set q = MX
> cl.cam.ac.uk
cl.cam.ac.uk mail exchanger = 5 mx.cam.ac.uk
> set q=A
> mx.cam.ac.uk
Address: 131.111.8.145
```

```
> set q = NS
> cl.cam.ac.uk
cl.cam.ac.uk nameserver = dns1.cl.cam.ac.uk
.....
cl.cam.ac.uk nameserver = dns0.eng.cam.ac.uk
> set q = A
> dns0.eng.cam.ac.uk
Address: 129.169.8.8
```



## DNS name servers – note the large scale

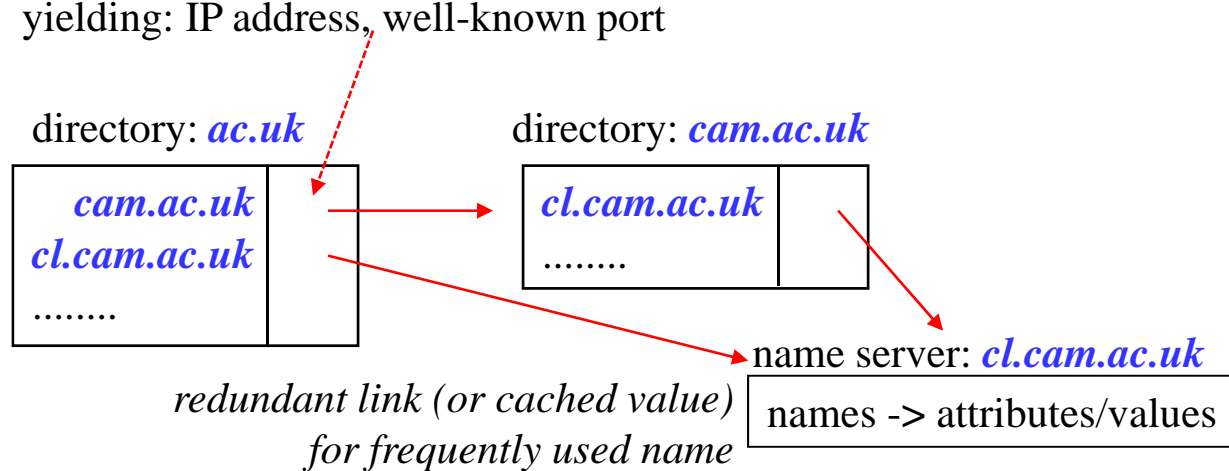
The domain database is partitioned into directories that form a distributed namespace

e.g. the directory *ac.uk* is held on the computer *nsl.cs.ucl.ac.uk*

We need a starting point for name resolution – e.g. the above for domains within the uk

DNS directory addresses can be looked up for a domain

yielding: IP address, well-known port



Directories are replicated for availability and good response (primary and secondaries per domain)

Authorised name server for domain is distinguished - weak consistency of secondaries with primary

Resolved queries are cached with a TTL (time to live)

(by user agents and directories) – works because *naming data tends to be stable*

Queries and responses may be batched into composite query messages

## DNS design assumptions and future issues

DNS and other name services were designed, in the days of desktops, on the assumption that objects are static, so that cached values continue to work, update rates are low, etc.

With huge data centres and high bandwidth available, is it time to recentralise to a few first-class servers? See Tim Deegan's thesis (examined by Paul Mockapetris) and:  
*T Deegan, J Crowcroft and A Warfield,*  
*“The MAIN name system, an exercise in centralized computing”*  
*ACM SIGCOMM 35(5), Oct 2005.*

New issues relate to [mobile devices](#) and [myriad small devices](#) including sensors

Mobile devices may attach anywhere worldwide  
device's MAC address is a UID, IP address?  
see comms. courses for details of protocols

## Name service design: Replication and Consistency

Directories are replicated for scalability, availability, reliability , ...

How should propagation of updates between replicas be managed?

*lookup (arguments)* < - > is the most recent value known, system wide,  
guaranteed to be returned?

If **system-wide (strong) consistency** were guaranteed this would imply:

- **delay on update**
- **delay on lookup**

It is essential to have fast access to naming data

– so we relax the consistency requirement

Is this justified?

## Name services – assumptions to justify weak consistency

Design assumptions were as below, but new issues have arisen

- naming data change rarely,
- changes propagate quickly,
- inconsistencies will be rare

**YES** – information on users and machines

**NO** – distribution lists ( see analysis of how Grapevine outgrew its specification )

**NEW** – mobile users, computers and small devices e.g. Internet-enabled phones

**NEW** – huge numbers of devices to be named – does the design rely on low update traffic?

- we detect obsolete naming data when it doesn't work

**YES** – users

**NO** – distribution lists

- If it works it doesn't matter that it's out of date
  - you might have made the request a little earlier
  - recall uncertainties over time in DS

## Consistency – vs - Availability

We have argued that **availability** must be chosen for name services, so **weak consistency**

When only weak consistency is supported:

*lookup (arguments)* - > returns either: *value, version# / timestamp*

or: *not known at time of last update*

### Examples:

Service on failed machine, restart at new IP address – update directory(s) – rare event

User changes company – coarse time grain

Companies merge – coarse time grain

Change of password – takes time to propagate – insecurity during propagation

Changes to ACLs and DLs – insecurity during propagation

Revocation of users' credentials – may have been used for authentication/authorisation  
at session start – can the effect be made instantaneous?

Hot lists e.g. stolen credit cards – must propagate fast – push rather than pull model

### Lessons:

Note design assumptions

Take care what data the name service is being used for

Does the service offer notification of change, on registration of interest, as in active databases?

# Long-term Consistency

## Requirement:

If updates stopped there would be consistency when all updates had propagated

Note that failure and restart behaviour must be specified for updates to propagate reliably

This requirement cannot be tested in a distributed system

- no guarantee that there will be periods of quiescence (no activity)

Updates are propagated by the message transport system

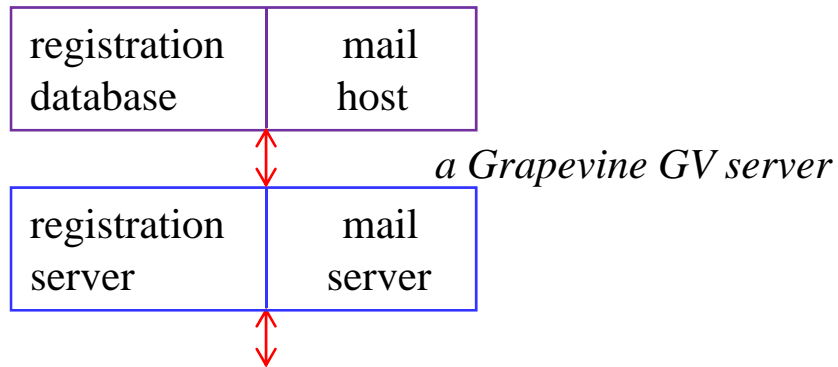
- **conflicting updates** might arrive **out of order**, from different sources
- need an arbitration policy based on timestamps
- but recall unreliability of source timestamps, so outcomes of an agreement protocol may not meet the *external* requirements.

Name services typically exchange whole directories periodically and compare them.

The directory is tagged with a new version# after this consistency check

e.g. GNS declares a new “epoch”

## Example: Grapevine – the first?



Note that small scale allows a simple design with rapid navigation

2D names *name@registry*

Every GV server contains the GV registry that contains, for all GV registries worldwide,  
*registry-name -> list of addresses* where registry is held

2 types of name within a registry

*group-name -> list of members* for distribution lists, also used for ACLs  
*individual-name -> attributes* such as password, mail-host-list, ...

Problems – soon outgrew its specification of #servers, #clients

huge distribution lists were not foreseen

client mail transport protocol used for system updates – could be held up

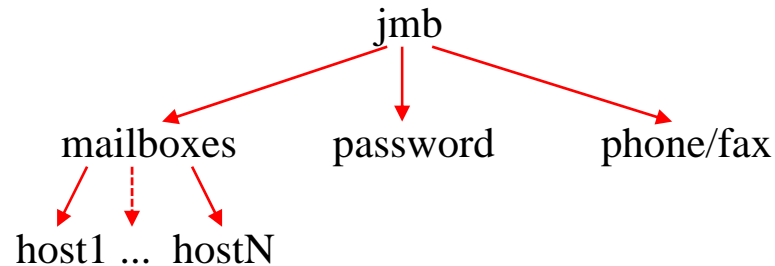
# Global Name service GNS ( DEC – 1986 )

*Lampson, Designing a Global Name service, Proc 5<sup>th</sup> ACM PODC, 1986*

Aims: long life – allowing many changes in the organisation of the namespace  
large scale – an arbitrary number of names and domains

Names

Define 2D names of the form < directory-name, value-name >  
where value-names may be a tree such as:



The GNS directory structure is hierarchical

**Every directory has a Directory Identifier ( DI ), a UID** *the novel design aspect of GNS*

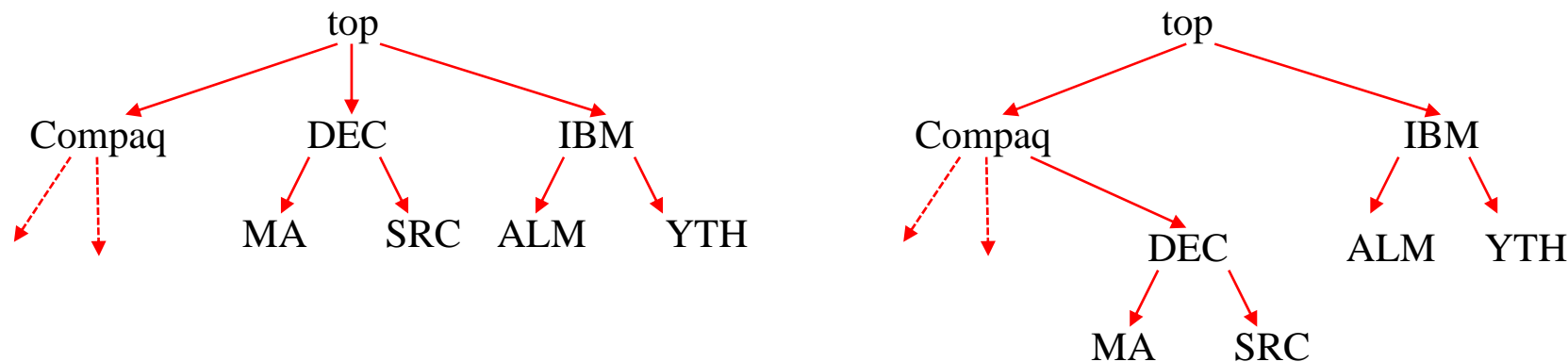
A full name is any name starting with a DI

- doesn't require a root directory
- doesn't rely on the availability of some root directory



## GNS namespace reconfiguration

If the directory hierarchy is reconfigured, a directory may still be found via its DI  
Names starting with that DI do not change if the reconfiguration is above that DI



*old names still work below the DEC directory*

Support is needed by the directory service to locate a directory from its DI

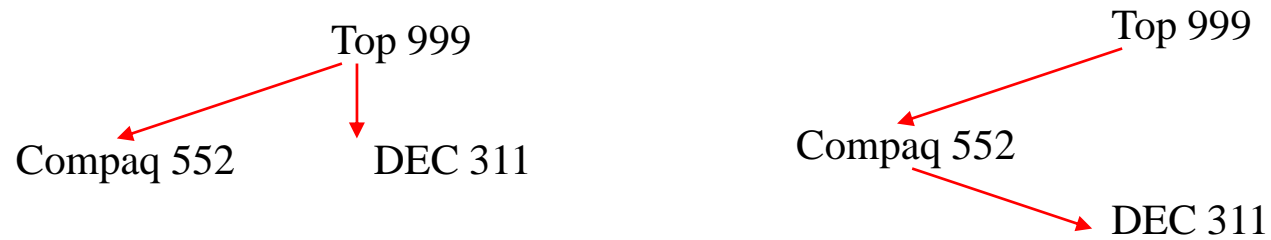
*a DI is a pure name – where do we look it up?*

Directories usually map directory pathnames to IP addresses

In addition, top level GNS directories store DIs with directory names, e.g.



## GNS namespace reconfiguration – directory updates



Names starting from DEC: *311/SRC, birrell* do not change. DEC is always 311

Names starting above DEC: *999/DEC/SRC, birrell* -> *999/Compaq/DEC/SRC, birrell*

Directory entries include – DIs with directory names  
 – pathnames from root

552 = 999/Compaq	IP	→
311 = 999/DEC	addresses	→
n = 999/DEC/SRC		→

552 = 999/Compaq	IP	→
311 = 999/Compaq/DEC	addresses	→
n = 999/Compaq/DEC/SRC		→

## X.500 Directory Service (White and yellow pages )

ISO and CCITT standard, above OSI protocol stack

More general than most name services where names must be known precisely  
and are resolved to locations

Components:

**DIT** directory information tree

**DSA** directory service agent

**DUA** directory user agent

**DAP** directory access protocol

*X.500 is resource-consuming*

*and difficult to use*

1993 major revision including replication, access control, schema management,

But X.500 was not accepted as a generic name service

X.509 certificates for authentication and attributes/authorisation have been successful

**LDAP Lightweight Directory Access Protocol**, *Howes, Kille, Yeong, Robins, 1993*

accepted by IETF – can download free and deploy – widely used

- access protocol built on TCP/IP
- heavy use of strings, instead of ASN.1 data-types
- simplification of server and client
- current status V3
- LDUP duplication and update protocol being developed

## Naming Postlude

Naming for the Internet, see [DNS](#)

Naming for companies, worldwide, motivated [Grapevine](#), see also [GNS](#)

Standard name services, [X.500](#), [LDAP](#)

Naming for the web – document names are based on internet naming

*[scheme://host-name:port/pathname at host](#)*

*[scheme](#)* = protocol: http, ftp, local file

*[host-name](#)* = web server's [DNS](#) address, default port 80

*[pathname](#)* is in web server's filing system of file containing data for web page

e.g. <http://www.cl.cam.ac.uk/research/>

Also, [W3C](#) have defined standards for web services (see Middleware)

with message content expressed in [XML](#)

[SOAP](#) – simple object access protocol

[WSDL](#) – web service description language

[UDDI](#) – universal description, discovery and integration

(directory service with web service descriptions in [WSDL](#))