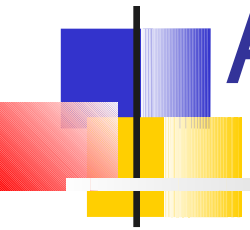


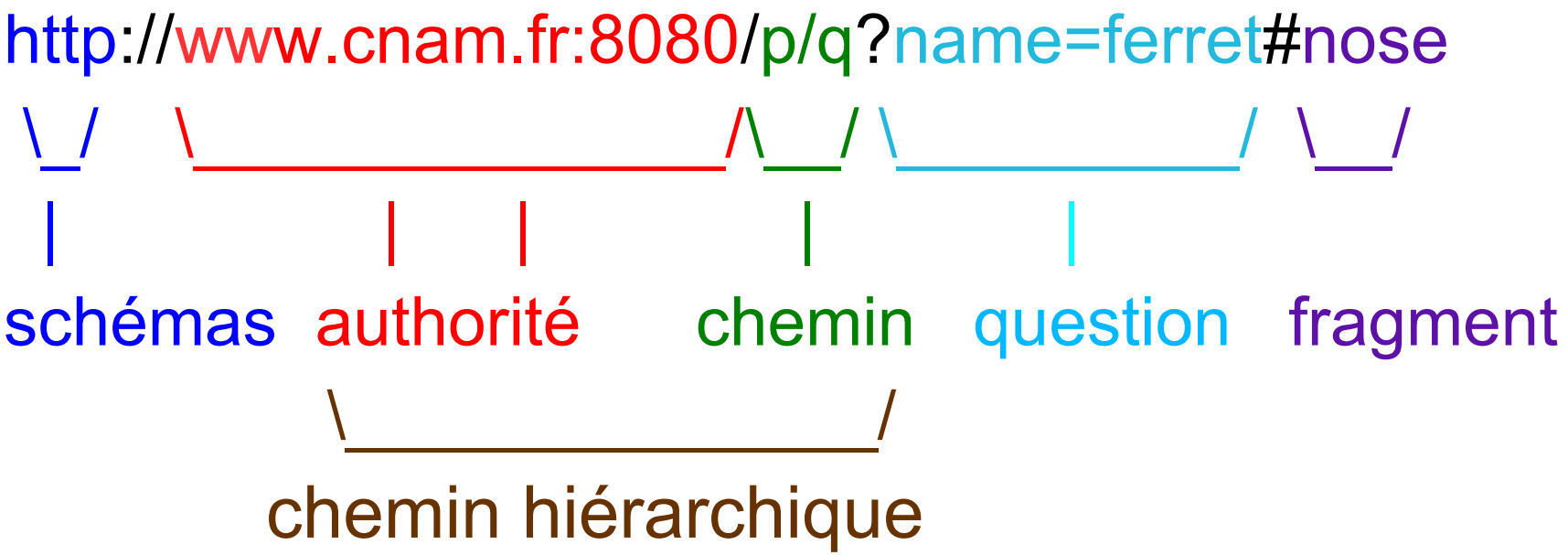
Addressage sur le Web

Uniform resource Identifier (URI)

Uniform Resource Identifier (URI), RFC 3986

- Objectif : identifier une ressource (donnée, service, ressource abstraire/physique, accessible ou non par l'Internet).
- Besoin:
  - Identification de la ressource
  - Localisation de la ressource (au niveau local ou dans le réseau)

URI – Format



- Syntaxe d'une URI (notation utilisée : Augmented Backus-Naur Form):

```
URI = schemas ":" cheminHierarchique
[ "?" question] [ "#" fragment ]
```

URI – Format

- **Schémas** : est associé à une application Internet permettant d'interagir avec la ressource. Par exemple : http
- `CheminHierarchique` : chemin d'accès à la ressource organisé de façon hierarchique

```
CheminHierarchique   =
"//" authorite  path-abempty
/ path-absolute
/ path-rootless
/ path-empty
```

URI - Format

- **Autorité** : éléments d'identification d'une ressource sur un hôte (distant)
  - Exemple : root@192.168.0.1:8080

```
Authorite = [userinfo "@"] host [":" port]
```

  - **Userinfo** : information permettant d'obtenir un accès à une ressource (typiquement un nom d'utilisateur)
  - **Host** : identifie l'hôte. En pratique, une adresse Ipv4/6 ou un nom (DNS)
  - **Port** : numéro de port
- Chemin : chemin d'accès à la ressource sur un hôte donné. Il est soit absolu, soit relatif, soit vide (et fait suite ou non à l'autorité)

URI-Format, compléments et accès interne à une ressource

- **Question**: paramètres complémentaires définissant une question

```
question = *( pchar / "/" / "?" )
```

  - Exemple: `http://www.google.fr/search?q=cnam`

- **Fragment** : identification indirecte d'une ressource secondaire (par exemple: portion d'un fichier HTML)

```
fragment = *( pchar / "/" / "?" )
```

Quelques exemples de chemins absolus et relatifs à http://a/b/c/d;p?

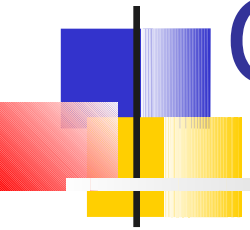
```
"g:h"           =  "g:h"
"g"             =  "http://a/b/c/g"
"./g"           =  "http://a/b/c/g"
"g/"            =  "http://a/b/c/g/"
"/g"            =  "http://a/g"
"//g"           =  "http://g"
"?y"            =  "http://a/b/c/d;p?y"
"g?y"           =  "http://a/b/c/g?y"
"#s"            =  "http://a/b/c/d;p?q#s"
"g#s"           =  "http://a/b/c/g#s"
"g?y#s"         =  "http://a/b/c/g?y#s"
";x"            =  "http://a/b/c/;x"
```

Conclusion - URL, URN, URI

- Vue classique : un identifiant est soit :
  - Une localisation (URL), soit,
  - Un nom (URN) indépendant de sa localisation, soit,
  - Un pointeur sur une métadonnée (URC)
- Vue contemporaine unifiée :
  - Un identifiant Web est une URI englobant plusieurs espaces de nom (`namespaces`)
  - Par exemple, une URL est un type d'URI particulier identifiant une ressource via une représentation de son mécanisme d'accès primaire (par ex. http)
    - http: urn: sont des shémas d'URIs
    - urn:isbn:n-nn-nnnnnn-n est un espace de nom d'une URI

Conclusion - URI

- URI : un adressage absolu ou relatif permettant une localisation
  - des ressources (par ex. document, service) sur l'Internet (ou localement)
  - À l'intérieur d'un document
- Une URI contient des informations d'adressage (identification et localisation) rendant difficile une migration des ressources (pas de suivie)
- Une URI n'a pas pour objet de décrire le contenu d'une ressource (méta-données)
- Spécifications : http://www.w3.org/Addressing

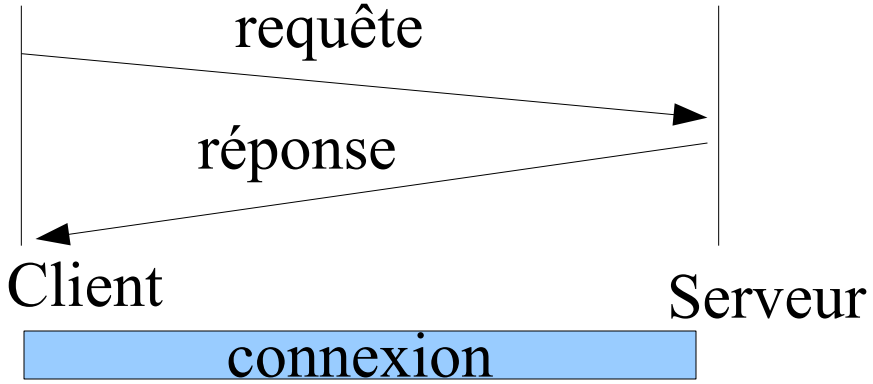


Communication sur le WEB

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP), RFC 2616

- Un protocole de niveau application, sans état, pour une interaction entre des systèmes multimédias distribuées et collaboratifs
- Un modèle de type client/serveur : le client HTTP émet une requête à laquelle répond le serveur



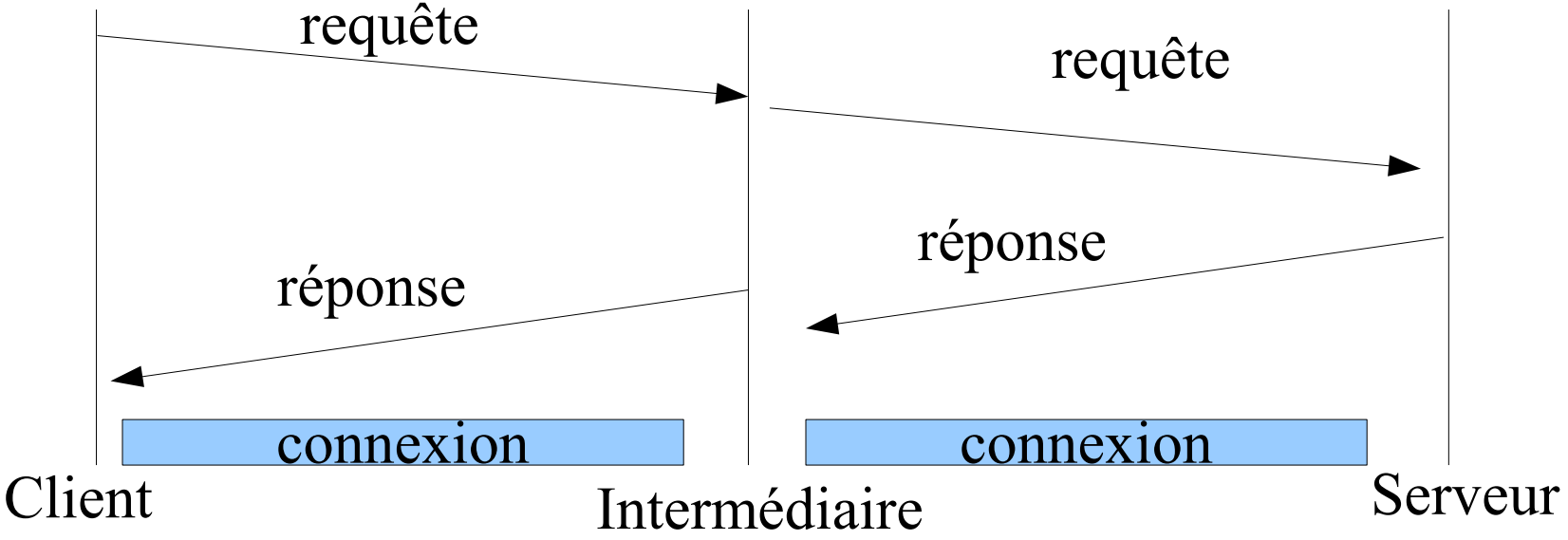
  - Une connexion est établie entre le client et le serveur de façon à acheminer la requête-réponse

Hypertext Transfer Protocol (HTTP)

- HTTP échange des messages en se basant sur un **protocole de transport ou de session fiable**
- La plupart des communications correspondent à des requêtes (GET) pour la représentation d'une ressource, ce qui nécessite une seule **connexion bidirectionelle**

HTTP - Intermédiaire

- Un intermédiaire (proxy, passerelle, tunnel) peut s'interposer entre le client et le serveur
- Client, serveur, intermédiaire peuvent disposer de cache



HTTP-Intermédiaire

- Une chaine de connexions est établie
- Les options de HTTP peuvent s'appliquer
  - avec le plus proche intermédiaire (à l’exception du tunnel)
  - Le destinataire final
  - Tous les intermédiaires

HTTP - Intermédiaire

- Un **proxy** modifie la requête avant de la faire suivre au serveur
- Une **passerelle** traduit si nécessaire la requête vers le protocole utilisé par le serveur
- Un **tunnel** est un relai (2 connections) ne modifiant par le message ; les tunnels sont utilisés lorsque la communication passe à travers un intermédiaire (un parre-feux par exemple)

HTTP – Connexion persistante

- Connexion non persistante : une connexion séparée est établie pour chaque URI demandée
  - Conséquence : surcharge du serveur, congestion de l'Internet
  - Versions antérieurs à HTTP1.1 : une connexion est par défaut non-persistante (et parfois non gérée)
- Connexion persistante : une seule connexion est établie pour plusieurs requêtes (et donc réponses)
  - **Pipeline** : les requêtes sont émises en parallèle, sans attendre de réponse. Les réponses suivent le même ordre que les requêtes. Option offerte par HTTP1.1
  - **Persistance** : si le serveur ferme la connexion, le client est préparé à réémettre

HTTP - Message

- Deux types de messages : requête , réponse

```
HTTP-message    = Request | Response
     ; HTTP/1.1 messages
```

- Un message est composé d'une première ligne, d'entêtes, d'une ligne vide, d'un corps

```
generic-message = start-line
*(message-header CRLF)
CRLF
[ message-body ]
```

HTTP-Message

- La première ligne spécifie s'il s'agit d'une requête ou d'une réponse

```
start-line = Request-Line | Status-Line
```

- Une requête contient la méthode, la cible (un destinataire typiquement identifié par une URI), la version du protocole

```
Request-Line = Method request-target
HTTP-Version CRLF
```

HTTP-Méthode

GET	Obtient des informations
HEAD	Est identique à un GET sauf que la réponse n' inclue pas de corps de message. Méthode utile pour obtenir des métas informations, **tester** des liens hypertextes
POST	Demande au serveur d'accepter l'entité inclue dans la requête en tant que ressource identifiée par l'URI. Est conçue pour annoter une ressource, poster un message (newsgroup), fournir un bloc de données (formulaire), étendre une base de données. La fonction à effectuer est déterminée par le serveur
PUT	L'entité contenue dans la requête doit être stockée (crée ou mise à jour) par le serveur
DELETE	Supprime la ressource identifié par l'URI
TRACE	Invoque une réponse de niveau applicatif à la requête. Sorte de ping auquel peut répondre le serveur, le proxy ou la passerelle. Est utile pour mener des diagnostiques
CONNECT	Est utilisée pour qu'un proxy puisse créer un tunnel (par exemple un tunnel de chiffrement de type SSL)
OPTIONS	Demande d'informations concernant les options d'une communication pour une requête-réponse concernant une URI. Cette méthode est disponible à partir d'HTTP1.1

HTTP - URI

- La cible est identifiée par une URI
  - Sous une forme absolue :

```
GET http://www.ietf.org/rfc/rfc2616.txt
HTTP/1.1
```

  - Sous une forme relative

```
 GET http://www.ietf.org/rfc/rfc2616.txt
HTTP/1.1 Host: www.w3.org
```

HTTP-Entête

- Une entête est exprimée sous la forme d'un nom suivie d'une valeur, tous deux en code ASCII
  - `message-header = field-name ":" [ field-value ]`
- Une entête est soit générale, soit spécifique à une requête/réponse ou à une entité contenue dans le message

HTTP- Entête générale

- Est relative au message (en opposition à son contenu)
- Est utilisée pour contrôler le traitement du message (par le client, le serveur, l'intermédiaire) ou fournir des informations supplémentaires
- N'est ni spécifique à une requête/réponse/entité du message

HTTP - Quelques entêtes générales

Connexion	Option de connexion non communiquée par un proxy	Connection: close
Date	Date de génération du message	Date: Tue, 15 Nov 1994 08:12:31 GMT
no-cache	Le cache doit valider avec le serveur	
no-store	La requête ou la réponse ne doit pas être stockée dans un cache	

HTTP – Quelques entêtes de requêtes

Accept-Charset	alphabet	Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
Accept-Encoding	encodage	Accept-Encoding: *
Accept-Language	langage	Accept-Language: en-gb;q=0.8, en;q=0.7
Accept-range	bornes	Accept-Ranges: bytes
Allow	Méthodes supportées par la source identifiée par l'URI	Allow: GET, HEAD, PUT
If-Modified-Since	Retourner l'entité si celle-ci a été modifiée depuis. Sinon, renvoyer le code 304	If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT

HTTP-Quelques entête réponse

Accept-ranges	Définie si le serveur accepte ou non des bornes	Accept-Range: none
Age	Age (exprimé en seconde) auquel la réponse a été généré	Age = 3000
Location	Redirection vers une localisation (autre que l'URI donnée)	Location = http://www.cnam.fr
Etag	Etiquette de l'entité incluse dans la réponse. Elle peut être utilisée par la suite par le client pour identifier l'entité	ETag: "nom"

HTTP- corps

- Le corps du message est encodé pour le transfert (l'encodage stipulé dans l'entête)

```
message-body = *OCTET
```

- Une ressource transportée dans le corps d'un message est appelée entité (entity)

HTTP – Format requête / réponse

- Requête

```
Request         = Request-Line
                        *( header-field CRLF )

                        CRLF
                        [ message-body ]
```

- Réponse

```
Response        = Status-Line
                        *( header-field CRLF )
                        CRLF
                        [ message-body ]
```

HTTP – Exemple requête / réponse

- Requête :

```
GET /path/file.html HTTP/1.0
From: someuser@jmarshall.com
User-Agent: HTTPTool/1.0
[blank line here]
```

- Réponse du serveur :

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html> <body>  ...</body> </html>
```

HTTP – Réponse

- La première ligne correspond un code notifiant le statut (trois entiers) et sa signification sous une forme textuelle

```
Status-Line = HTTP-Version
Status-Code Reason-Phrase CRLF
```

HTTP – Réponse, code de statut

Le premier chiffre du code définie la classe de réponse

1xx	Informationel – Requête reçue, processus en cours
2xx	Succès – L'action a été reçue, comprise et acceptée
3xx	Redirection – Une action supplémentaire doit être entamée pour achever la requête
4xx	Erreur client – la requête est sujette à un problème syntaxique ou ne peut être effectuée
5xx	Erreur Serveur – le serveur ne peut mener à terme une requête à première abord valide

HTTP – Réponse, quelques codes de status

100	Continue
101	Switching protocol
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No content
301	Moved permanently
307	Temporary redirect
400	Bad request
500	Internal server error

HTTP - Conclusion

- Un protocole simple de type requête-réponse majoritairement basé sur TCP et sans état
  - Cookies : un moyen simple de stocker des informations d'état sur le client pour des transactions longues durées. Le client
- Un véhicule pour n'importe quelle information
  - Schémas d'authentification basique (login utilisateur/password) ou par digest et d'encryption (Secure Socket Layer : SSL)
  - Un proxy est par essence un *man in the middle*
- Références vers les spécifications maintenues par le W3C : http://www.w3.org/Protocols/Specs.html