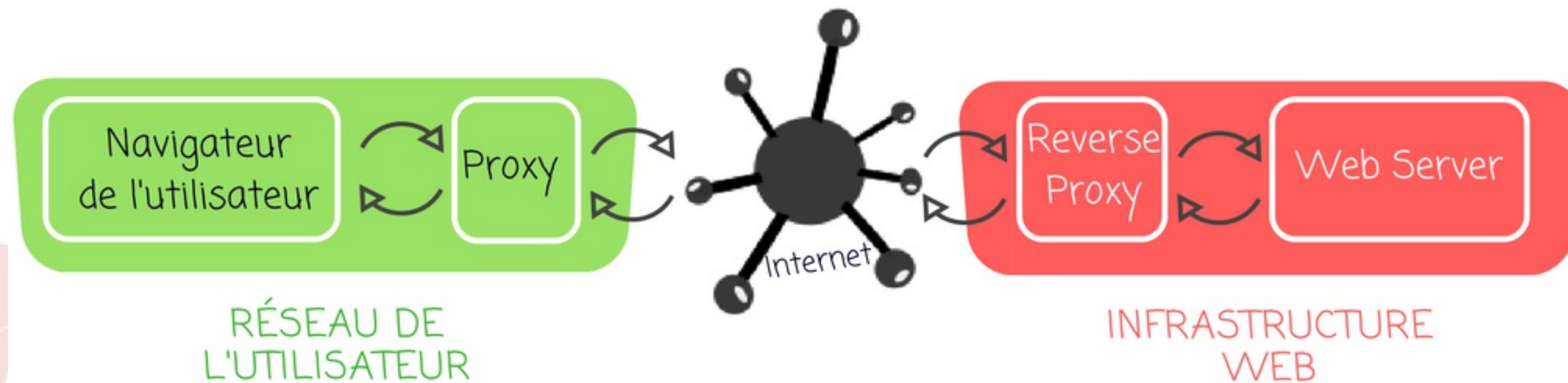




# CONTRÔLE DU CACHE EN HTTP

# CHAINE DE CONNEXION

Pour bien comprendre ce qui suit :



# LES DIFFÉRENTS ENTÊTES

- Expires
- Cache-Control
- Age
- Vary
- ETag (identifiant unique d'objet)
- Pragma (obsolète, date de HTTP/1.0)

# Expires

Indique une date d'expiration absolue.

Exemple :

```
Expires: Wed, 21 Jun 2017 00:42:00 GMT
```



# Age

Entête ajouté par les serveurs de cache.  
Indique l'ancienneté de l'objet, en seconde.

Si il vient d'être récupéré, Age: 0

# Cache-Control

Définit la politique de mise en cache pour le client.

Directives possibles, dans le cas d'une réponse :

- no-cache : ne peut pas être mis en cache
- private : contenu spécifique à l'utilisateur, peut être gardé dans un cache privé uniquement (par ex. le navigateur)
- public : peut être mis dans n'importe quel cache

# Cache-Control

- max-age : durée de validité de l'objet, relativement à la valeur de Age. Surcharge la valeur de Expires
- s-maxage : idem que max-age mais à destination des serveurs de cache uniquement. Surcharge max-age
- must-revalidate : le client doit d'abord s'assurer que le contenu n'est pas modifié coté serveur. Si identique, 304
- proxy-revalidate : idem que must-revalidate mais pour les serveurs de cache uniquement.

# Cache-Control

## Exemples :

- Pas de mise en cache :

```
Cache-Control: no-cache
```

- Page spécifique à un utilisateur, qui peut changer :

```
Cache-Control: private, must-revalidate
```

- Mise en cache des contenus statiques :

```
Cache-Control: public, s-maxage=604800, max-age=86400
```



# Vary

À l'intention des serveurs de cache intermédiaires.  
Permet de stocker plusieurs « variantes » d'une même page, ayant des entêtes différents.

Principaux cas d'usages :

```
Vary: Accept-Encoding  
Vary: User-Agent
```

# PAGES DYNAMIQUES ET COOKIES

- Attention au cache leak : pas de Cache-Control: public avec un Set-Cookie.
- Généralement les proxies ne cacheront pas les réponses à des requêtes contenant un Cookie.

# PAGES DYNAMIQUES ET COOKIES

Problème des pages dynamiques uniques par visiteurs.

Pistes :

- limiter autant que possible les cookies inutiles (à réserver aux utilisateurs connectés) ;
- les cookies utilisés uniquement en Javascript peuvent être sécuritairement supprimés au niveau du reverse-proxy ;
- idem sur les fichiers statiques ;
- utiliser ESI

# LANGAGE ESI

*Edge Side Include*

À destination des serveurs de cache.

Permet de séparer une page HTML en fragment.  
Chaque fragment peut être ou non mis en cache.

Exemple :

```
<html>
  <body>
    [...]
    <p>Votre panier : <esi:include src="/panier.php" />
    [...]
  </body>
</html>
```



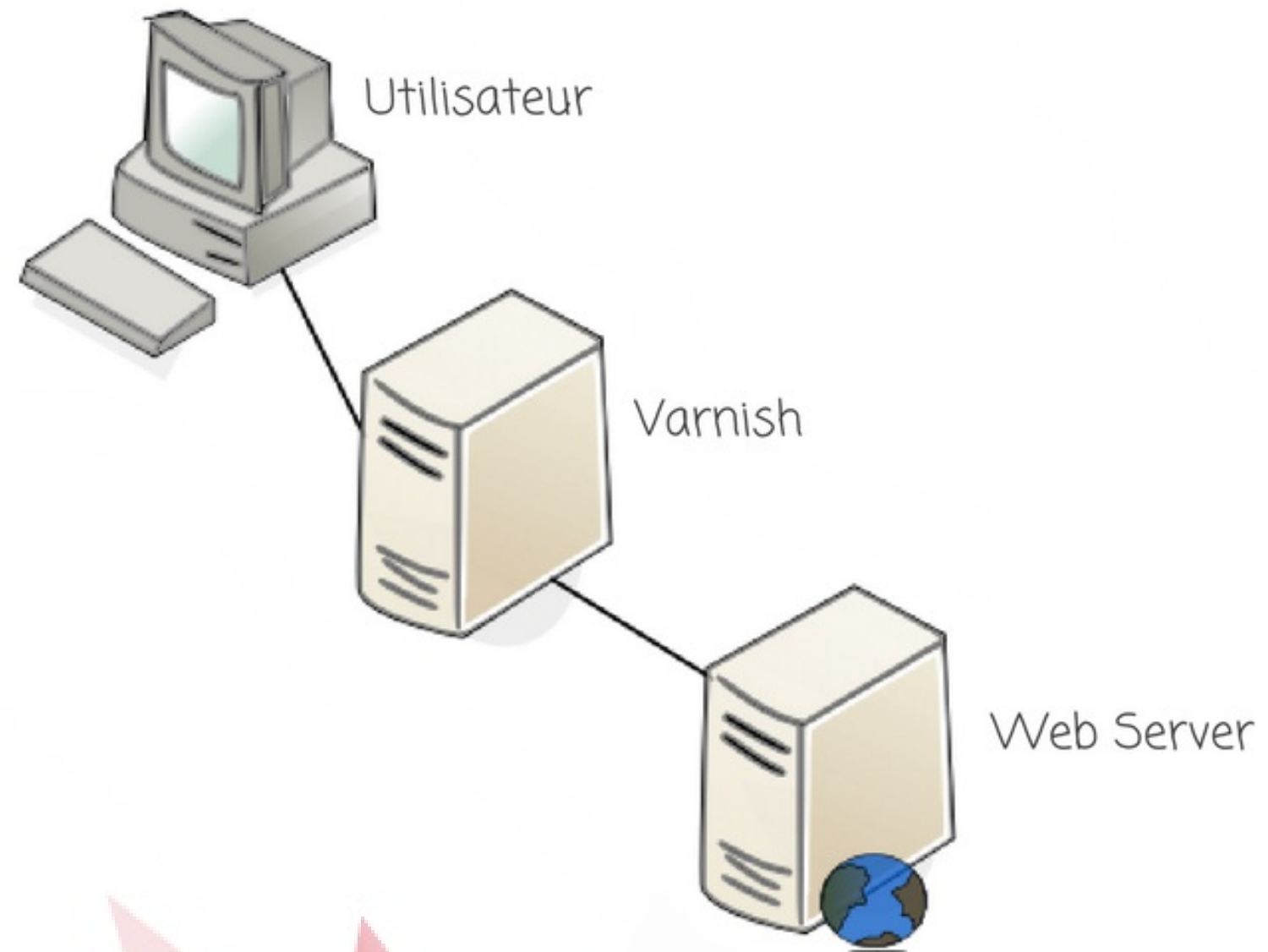
# **LE CACHE D'UN POINT DE VUE DE SYSADMIN**



# LOGICIELS DE CACHE LIBRES

- Apache2
- Nginx
- Squid
- Varnish

# INFRASTRUCTURE



# INSTALLATION DE VARNISH SOUS DEBIAN

- Tout simplement `# apt install varnish`
- Mais... attention à `/etc/default/varnish` en Debian Jessie avec `systemd`



# CONFIGURATION

- Langage de configuration propre à Varnish : VCL
- Deux routines principales : `vcl_recv` et `vcl_backend_response`
- Différents mots clés : `set` et `unset`, `(be)req` et `(be)resp`
- Différentes actions : `hash`, `pass`, `pipe`, `purge`...

# CERTAINES CONDITIONS POSSIBLES

```
# Condition sur l'entête HTTP Host:
if (req.http.host ~ "^regex$")
# Présence d'un cookie
if (req.http.cookie)
# Condition sur l'URL demandée
if (req.url ~ "^/regex$")
# Si le backend est accessible
if (req.backend.healthy)
# Présence de l'entête Accept-Encoding
if (req.http.Accept-Encoding)
# Condition sur la requête faite
if (req.method != "GET" && req.method != "HEAD")
```



# LES ACTIONS BASIQUES

- return (hash);
- return (pass);
- return (pipe);
- return (purge);

# CERTAINES ACTIONS POSSIBLES PLUS POUSSÉES

```
# Supprimer les cookies dans la requête  
unset req.http.cookie;  
# Supprimer un certain nombre d'entêtes HTTP  
unset req.http.X-Forwarded-For;  
unset req.http.Accept-Encoding;  
# Positionner un certain nombre d'entêtes HTTP pour la requête  
set req.http.X-Forwarded-For = client.ip;  
set req.http.Accept-Encoding = "gzip";  
set req.http.Accept-Encoding = "deflate";
```

# EXEMPLE

```
# No cache for WordPress' backoffice  
if (req.url ~ "^/wp-(login|admin)" ) {  
    return (pass);  
}
```

# CONFIGURATION SIMPLE

```
sub vcl_recv {
    if (req.http.host ~ "(www\.example\.com|example\.com)") {
        set req.backend_hint = default;
    }
    if (req.url ~ "^/images") {
        unset req.http.cookie;
    }
}
sub vcl_backend_response {
    if (bereq.url ~ "\.(png|gif|jpg)$") {
        unset beresp.http.set-cookie;
        set beresp.ttl = 3600s;
    }
}
```



# OUTILS D'ADMINISTRATION

- varnishtop - parse les logs pour donner des informations en temps réel
- varnishadm - pour administrer une instance varnish
- varnishlog - pour afficher les logs

# PURGE DU CACHE

- `varnishadm 'ban req.url ~ .'`
- `curl -X PURGE http://www.example.org/foo.txt #`  
cela nécessite une ACL !



# POUR EN SAVOIR PLUS...

- Wiki Evolix : [wiki.evolix.org/HowtoVarnish](http://wiki.evolix.org/HowtoVarnish)
- Rôles Ansible Evolix :  
[forge.evolix.org/projects/ansible-roles](http://forge.evolix.org/projects/ansible-roles)
- Twitter : [@EvolixCanada](https://twitter.com/EvolixCanada)
- Blog : [blog/evolix.ca](http://blog/evolix.ca)
- Mail : [hello@evolix.ca](mailto:hello@evolix.ca)