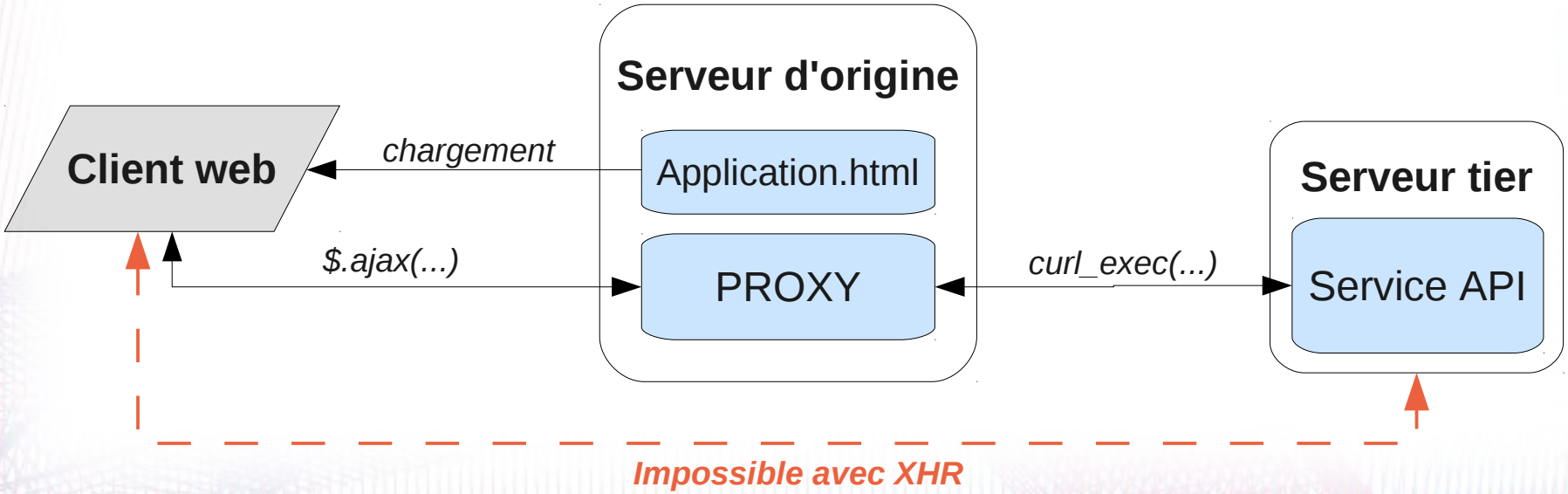
AJAX, dialogue « cross-domain »

- Same Origin Policy, un élément de sécurité des navigateurs web :
  - impossible de demander un contenu à un serveur autre que celui d'origine de l'application web.
  - un danger : XSS (cross-site scripting)
    - par ex. eval() d'un flux JSON (démo)
  - mais pas de pb pour un flux approuvé
- Charger du contenu depuis des sources tierces
  - quelques solutions de contournement : PROXY, Injection de script, JSONP, ...

➔**il faut une confiance certaine en la source.**

Approche PROXY

- Le serveur d'origine charge le contenu et le renvoi au client AJAX.
  - Puissant car en plus le serveur peut faire un prétraitement de son côté si besoin.
  - Mais double le traffic



Approche « injection de script »

- Basé sur une souplesse de la balise <script>
  - on peut charger un script depuis un serveur tier
  - en insérant « à la demande » une balise <script> dans le DOM, càd au moment de la requête
  - sa source est une URL sur le serveur tier, en GET
- Nécessite une étroite coopération avec le serveur tier
  - sur le nom de la fonction de rappel
    - le service impose le nom de la fonction
    - le client indique au serveur tier le nom

Approche JSONP avec jQuery

- C'est une application de l'injection de script
  - considérant un flux résultat formatté en JSON
- jQuery facilite son utilisation :
  - pas d'insertion de balise à gérer
  - un dataType en soit (jsonp)
  - configuration du nom de la fonction de rappel
  - Inconvénients :
    - Pas de fonction de rappel sur erreur (error)
    - Ne fonctionne pas en POST

Cross-Origin Resource Sharing

- CORS, implémentation native pour le dialogue AJAX dit « cross-domain » ou « cross-origin »
  - Nouvel entête de requête

    ```
    Origin: http://www.heig-vd.ch
    ```

  - Nouvel entête de réponse

    ```
    Access-Control-Allow-Origin: *
    ```

- Supporté par :
  - IE8+, FF3.5+, Safari 4+, Chrome
- Côté serveur :
  - niveau application, mod_headers d'Apache, ...