# 第六讲 网络应用

中国科学院计算技术研究所

网络技术研究中心

# 本讲提纲

- 网络应用
  - DNS (Domain Name System)/域名解析
  - Web应用
    - HTTP、性能和安全
  - 互联网视频
    - 系统设计、性能测量和优化

# 为什么需要域名解析？

- 用户倾向于使用可读的名字
  - e.g. retrieving the homepage of www.baidu.com
- 计算机更易处理数字地址
  - e.g. read index.html at address 61.135.169.125 on port 80
- DNS将两者关联映射起来
  - What’s the IP address of www.baidu.com -> 61.135.169.125
- DNS是互联网系统中最关键的服务之一
  - 功能、性能、安全

# 早期域名解析方案

## 本地文件存储DNS映射

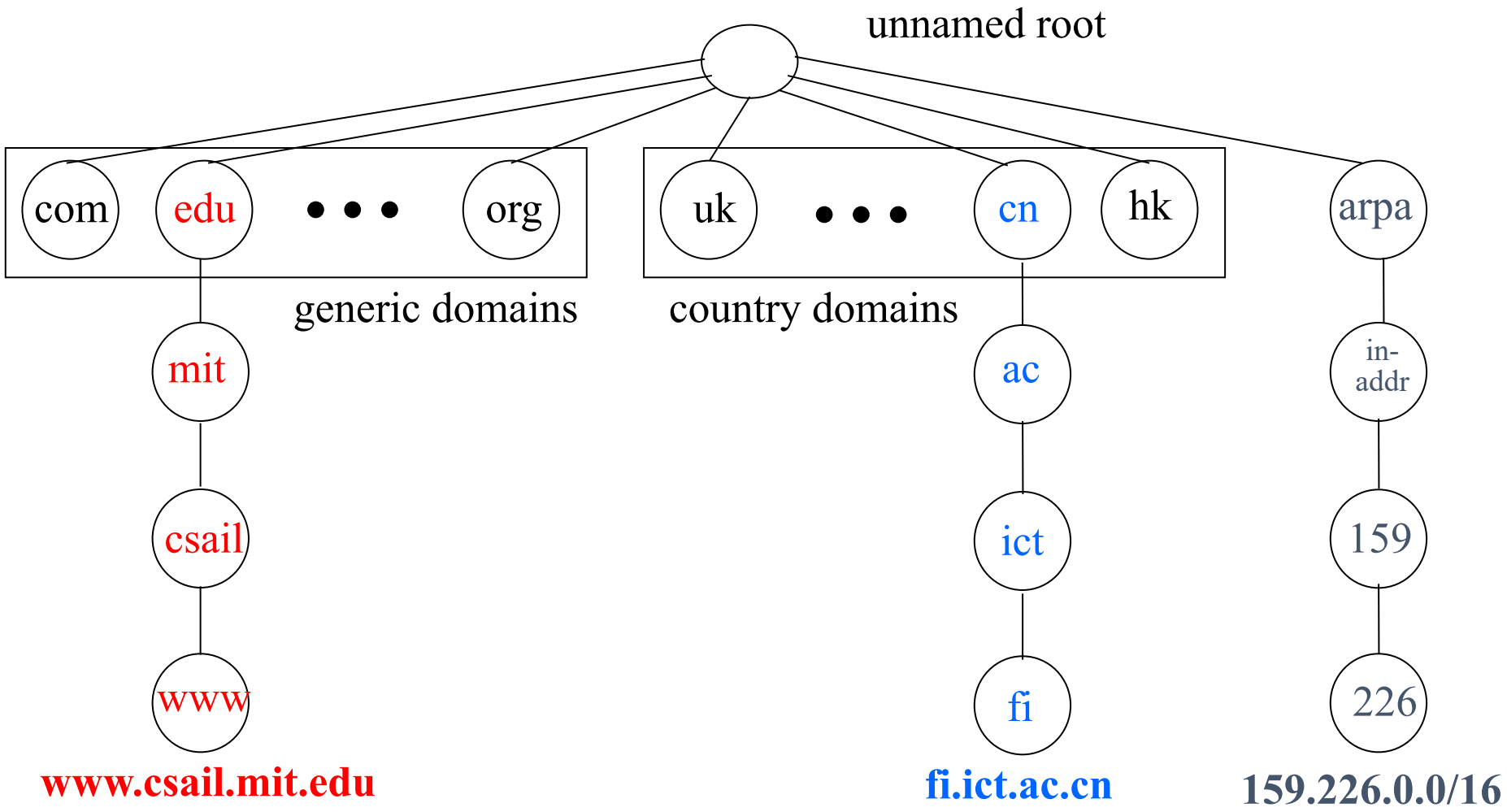
- 扁平化 (flat) 的命名空间
- /etc/hosts
- 由SRI-NIC维护正本
- 其他主机定期的从该主机更新副本

- 当主机数目增加时
  - SRI-NIC需要频繁更新正本
  - 越来越多的副本更新下载

# 当前域名解析方案：DNS

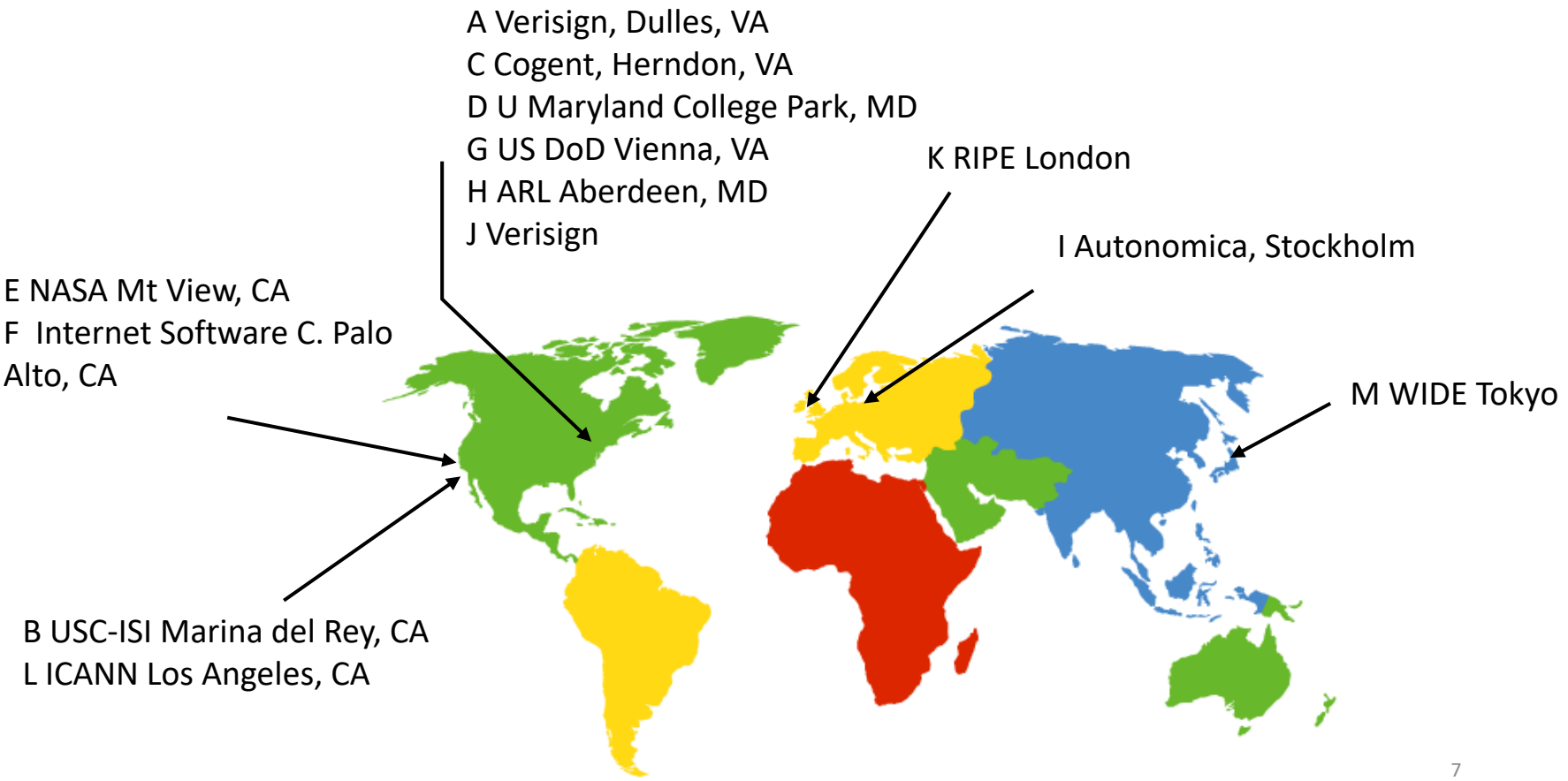
- 支持域名到地址的映射查询
  - 类比于网络层与数据链路层之间的ARP
- 层次化的命名空间
  - 域名：com, google.com, www.google.com
  - IP地址：10.0.0.0/8, 10.21.0.0/16, 10.21.2.0/24
- 分布式、层次化的域名空间存储和管理
  - 根服务器
  - 顶级域名服务器
  - 权威服务器

# 层次化的命名空间


unnamed root
com
edu
org
uk
cn
hk
arpa
generic domains
country domains
mit
ac
in-addr
csail
ict
159
www
fi
226
www.csail.mit.edu
fi.ict.ac.cn
159.226.0.0/16

# DNS根服务器

- 全球共有13个DNS根服务器，用A-M来标记



# 顶级/权威域名服务器

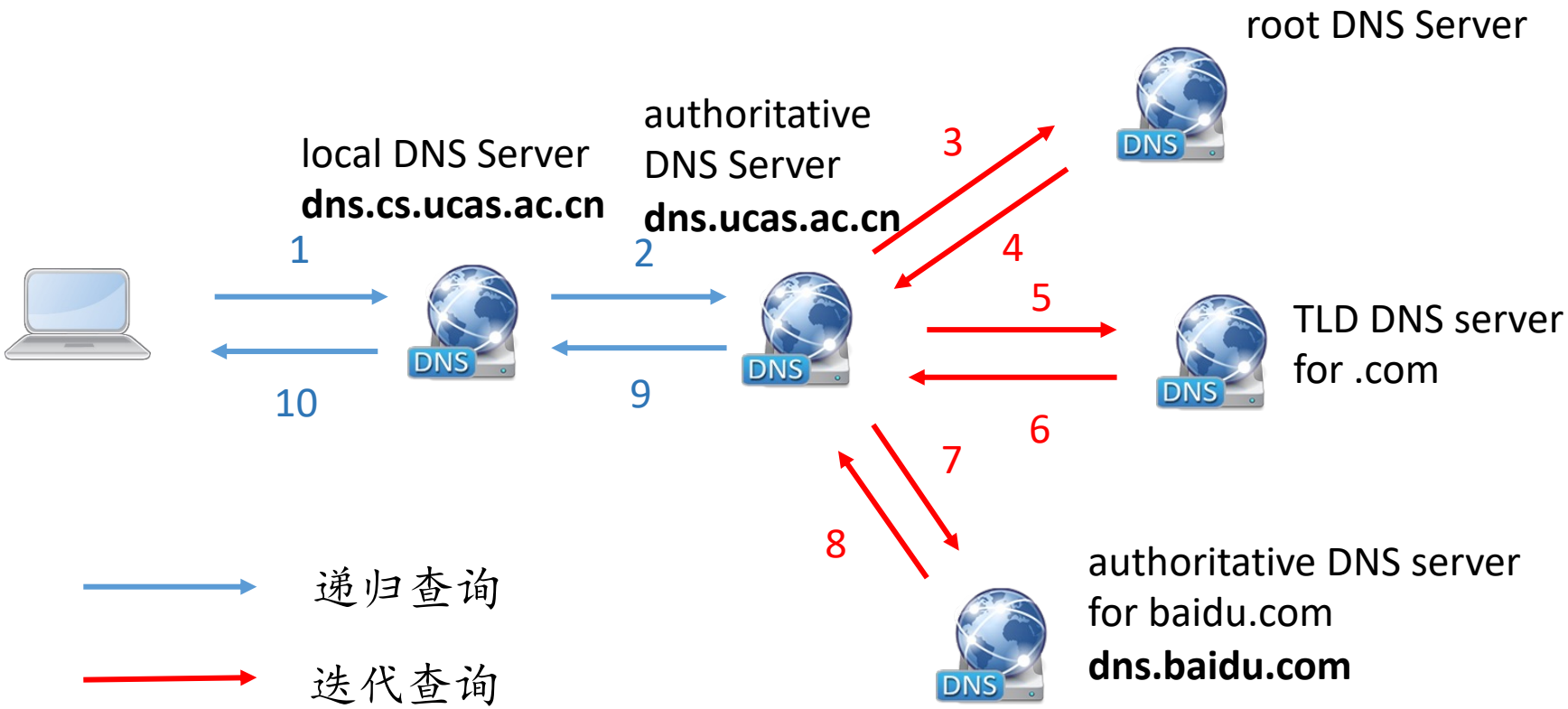
- 全球顶级域名(Global Top-level domain, gTLD)服务器
  - 一般性域名 (e.g. .com, .org, .info)
  - 国家地区域名 (e.g. .cn, .hk, .uk)
  - 一般由专业机构来维护管理 (e.g. VeriSign 管理 .com和.net域名)
- 权威(Authoritative)域名服务器
  - 提供一个组织内的域名与主机映射关系
    - 通常是该组织提供的服务
  - 一般由组织自己维护管理

# 如何使用DNS

- 本地DNS服务器
  - 通常离终端用户比较近
  - 在/etc/resolv.conf中配置，或由DHCP获取
- 客户端程序
  - 从URL中提取服务名字
  - 使用*getaddrinfo()* 来查询相应地址
- 服务器程序
  - 从socket中获取客户端IP地址
  - 可以使用*getnameinfo()*来查询客户端对应的域名

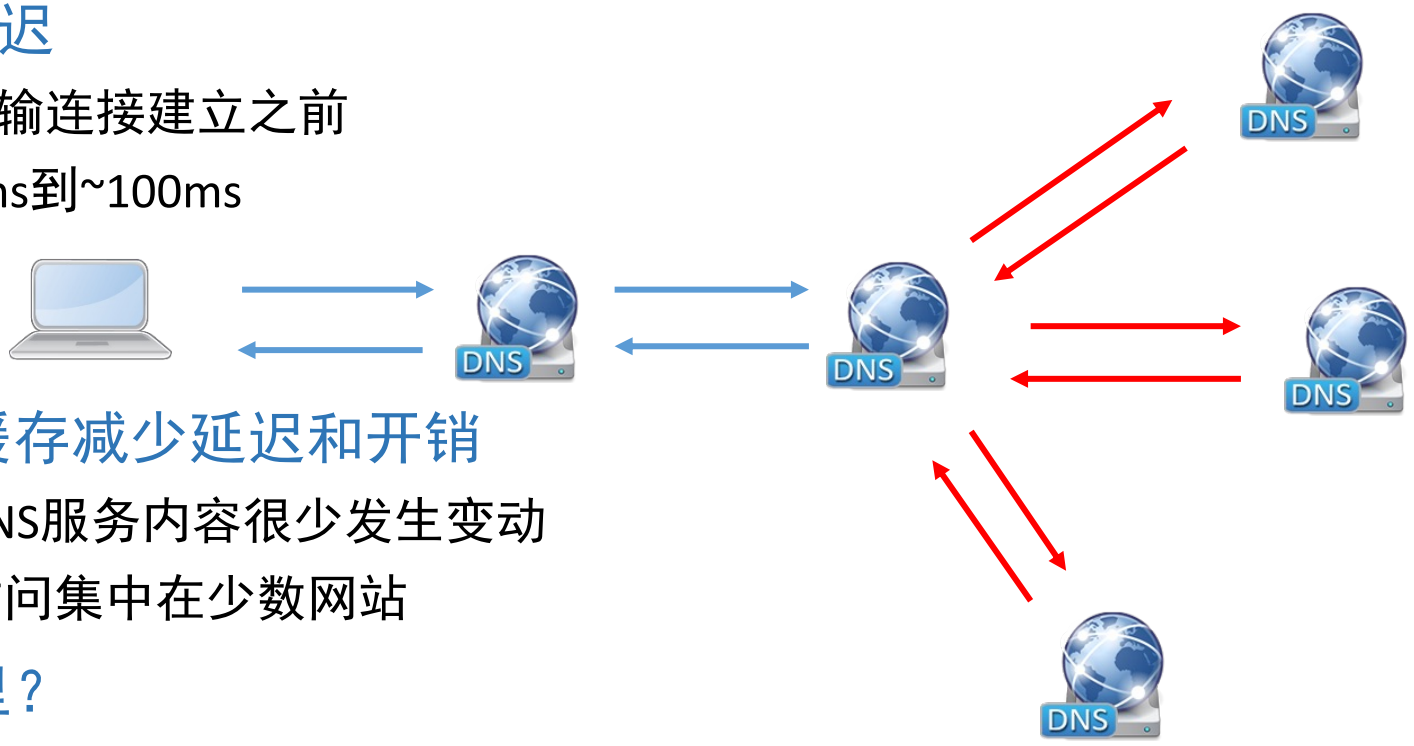
# DNS查询

- 主机pc1.cs.ucas.ac.cn查询www.baidu.com的IP地址



# DNS缓存

- DNS查询延迟
  - 发生在传输连接建立之前
  - 约为~10ms到~100ms




- 可以通过缓存减少延迟和开销
  1. 顶级DNS服务内容很少发生变动
  2. 服务访问集中在少数网站
- 缓存在哪里？
  - 本地DNS服务器
  - 浏览器

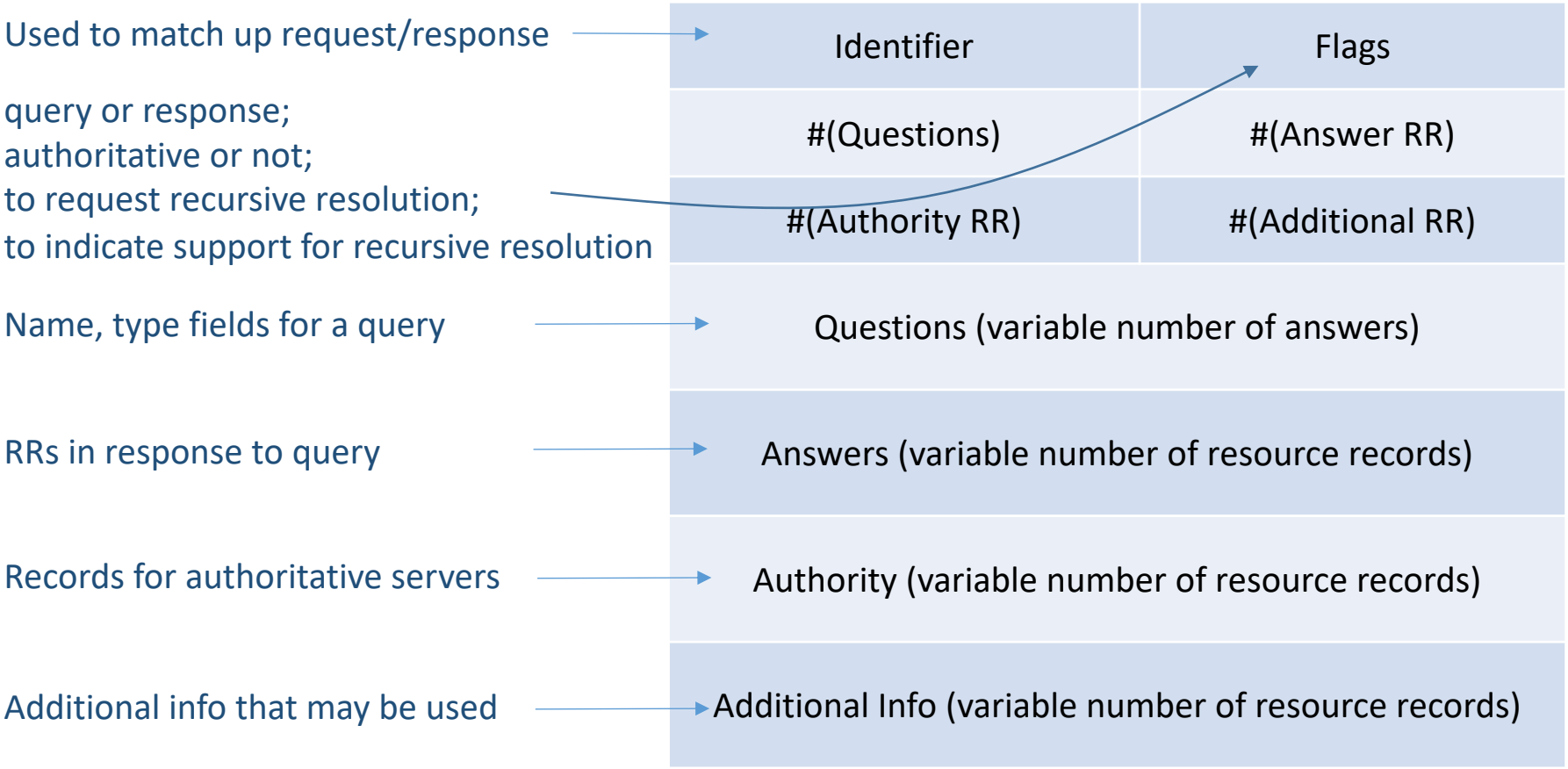
# DNS资源记录 (Resource Record)

RR format: **`(name, value, type, ttl)`**

- Type=A
  - **Name**: hostname
  - **Value**: IPv4 address
- Type=AAAA
  - **Name**: hostname
  - **Value**: IPv6 address
- Type=CNAME
  - **Name**: alias for the canonical name:

    www.baidu.com is really www.a.shifen.com
  - **Value**: canonical name

# DNS消息格式


Used to match up request/response
query or response;
authoritative or not;
to request recursive resolution;
to indicate support for recursive resolution
Name, type fields for a query
RRs in response to query
Records for authoritative servers
Additional info that may be used
Identifier
Flags
#(Questions)
#(Answer RR)
#(Authority RR)
#(Additional RR)
Questions (variable number of answers)
Answers (variable number of resource records)
Authority (variable number of resource records)
Additional Info (variable number of resource records)

# DNS服务可靠性

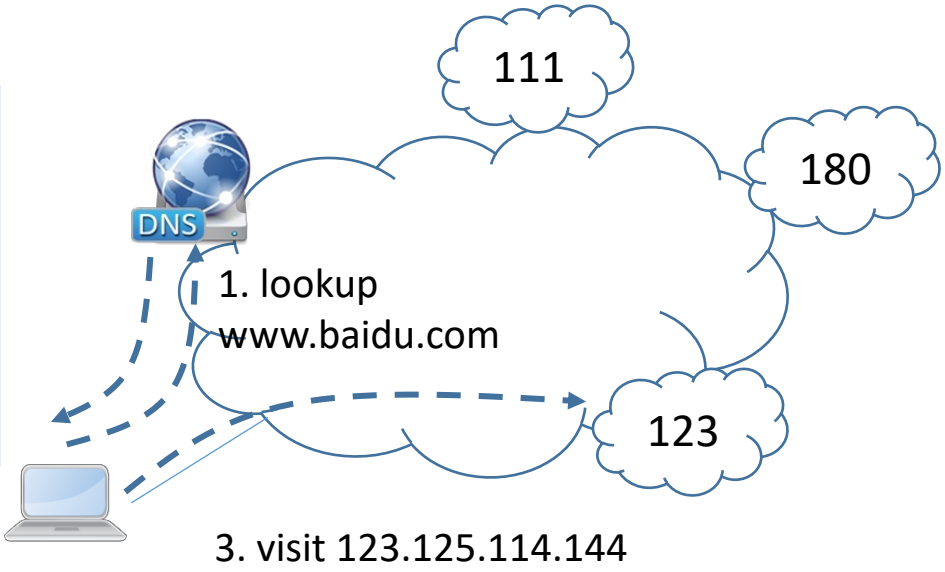
- 可以由多个DNS服务器提供服务
  - 只要有1个服务器工作，就能提供DNS服务
  - 可以在多个服务器之间做负载均衡
- 使用UDP进行服务查询
  - 非可靠传输
- 在超时之后可以选择其他DNS服务器
  - 同一服务器超时后进行指数退避
- 对所有查询，使用同一ID
  - 不关心由哪个服务器返回查询结果

# 使用DNS进行负载均衡

- Round Robin DNS
  - DNS服务器对同一域名解析请求，返回多个IP地址
    - 但是每次返回的顺序不同（Round Robin）
  - 客户端选择第一个IP地址作为目标服务器地址

2. return

```
Non-authoritative answer:
   Name:   baidu.com
   Address: 123.125.114.144
   Name:   baidu.com
   Address: 111.13.101.208
   Name:   baidu.com
   Address: 180.149.132.47
```

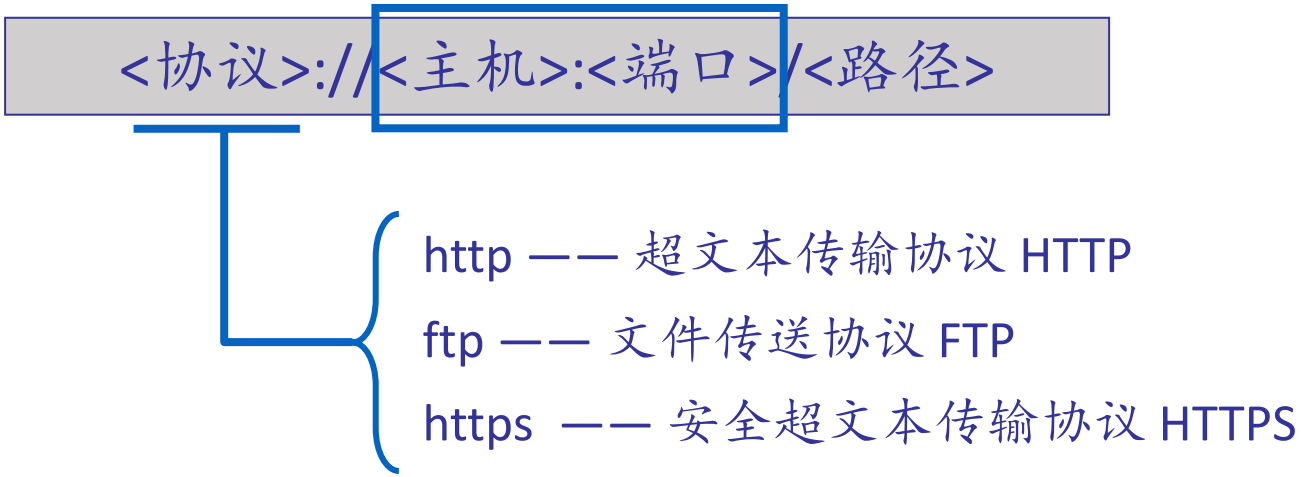


# Web

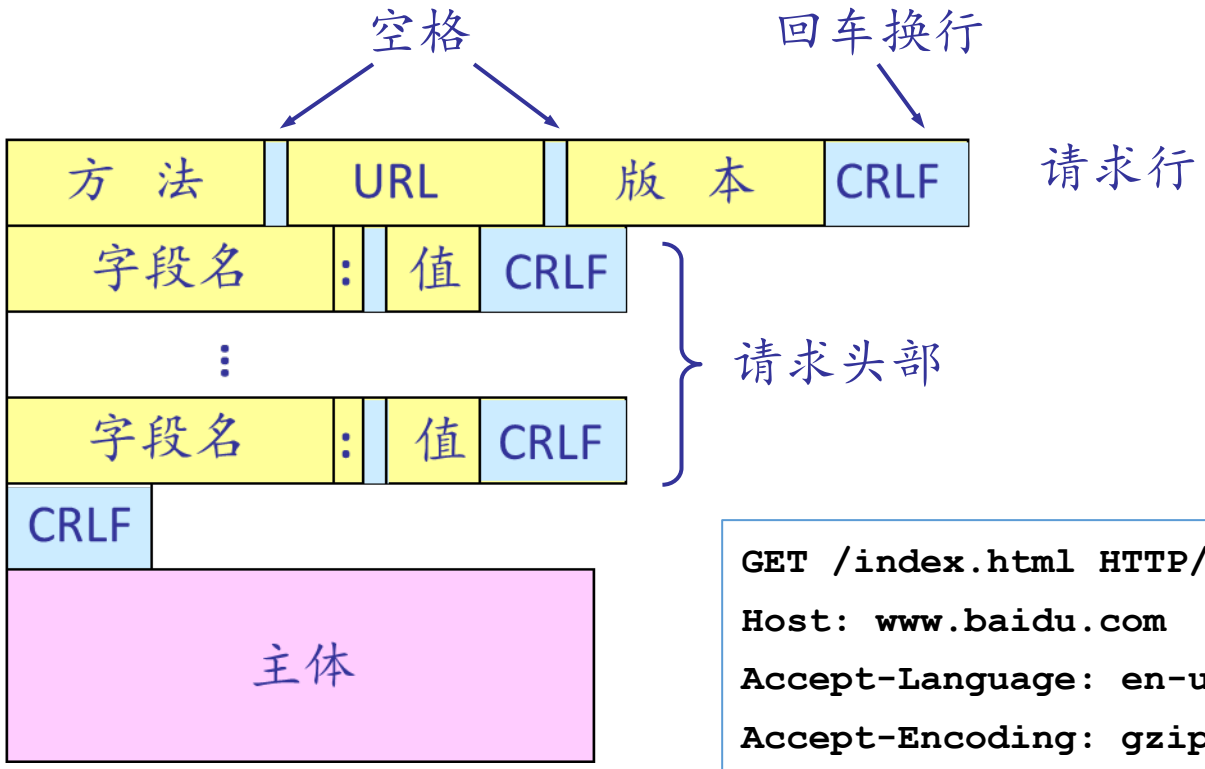
- World Wide Web/Web/WWW
  - 1989年由CERN的蒂姆·伯纳斯-李发明
  - 一个由许多互相链接的超文本组成的资源系统
  - 每个资源由一个全局唯一的“统一资源标识符”（URI）标识
    - http://www.baidu.com/
  - 使用超文本传输协议（Hypertext Transfer Protocol, HTTP）传输
    - GET /index.html HTTP/1.0

# 统一资源定位符 URL

- URL 是对资源的位置和访问方法描述
- 由以冒号隔开的两部分组成
  - URL 字符对大小写没有要求
  - e.g. https://www.baidu.com/index.html



# HTTP请求报文 (Request)




```
GET /index.html HTTP/1.1
Host: www.baidu.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5)
Connection: Keep-Alive
```

# HTTP请求行

## HTTP请求行

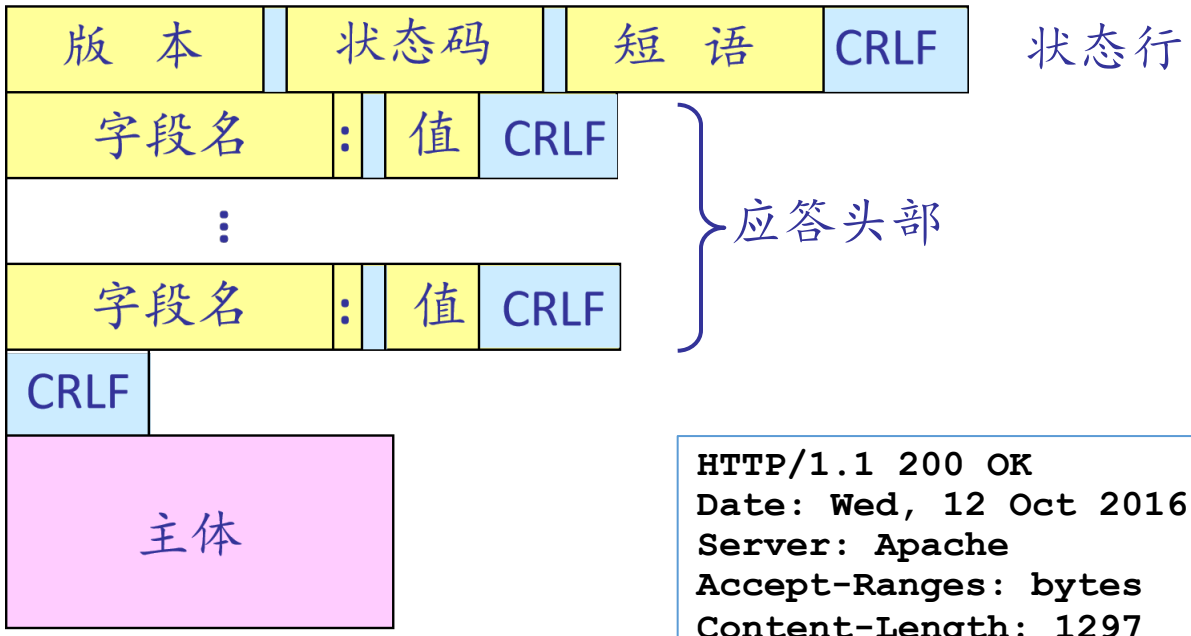
- 方法 (Method)
  - GET: 返回URI对应的内容
  - POST: 向服务器发送数据
  - DELETE、CONNECT、 OPTIONS、 HEAD、PUT
- URL (相对URL)
  - e.g. /index.html
  - 也可以写绝对URL
- HTTP版本
  - HTTP/0.9 HTTP/1.0 HTTP/1.1 HTTP/2.0

# HTTP请求头部

## HTTP请求头部

- 变长、可读的字符串
- 包括（不限于）：
  - 主机 (Host)
  - 认证 (Authorization)
  - 可接受文档类型、编码类型
  - 缓存 (Cache-Control)
  - 提交者 (Referer)
  - 用户代理 (User-Agent)
  - 连接管理 (Connection)

# HTTP应答报文 (Response)




```
HTTP/1.1 200 OK
Date: Wed, 12 Oct 2016 12:26:03 GMT
Server: Apache
Accept-Ranges: bytes
Content-Length: 1297
Connection: Keep-Alive
Content-Type: text/html

Body…
```

# HTTP状态码

| 状态码 | 定义 | 说明 | 示例 |
| --- | --- | --- | --- |
| 1XX | 信息 | 接收到请求，继续处理 | 100 Continue |
| 2XX | 成功 | 操作成功地收到，理解和接受 | 200 OK |
| 3XX | 重定向 | 为了完成请求，必须采取进一步措施 | 301 Moved Permanently |
| 4XX | 客户端错误 | 请求的语法有错误或不能完全被满足。 | 404 Not Found |
| 5XX | 服务端错误 | 服务器无法完成明显有效的请求。 | 500 Internal Server Error |

# 如何标识消息结束?

- 显式关闭连接
  - 由服务器来关闭
  - 每个TCP连接只处理一个Request，性能差
- 由Content-Length来标识
  - 在传输之前已经确定消息长度
- 对于没有消息内容的，使用两个CRLF结尾
  - 有些状态没有消息内容，e.g. 304
- 分块传输 (chunked)
  - 在发送相应头部之后，每传一个chunk之前，先用16进制标识其长度
  - 最后一个chunk写0

# HTTP分块传输

```
HTTP/1.1 200 OK <CRLF>
Transfer-Encoding: chunked <CRLF>
<CRLF>
10 <CRLF>
0123456789ABCDEF<CRLF>
1A <CRLF>
0123456789ABCDEF0123456789<CRLF>
0 <CRLF>
```

- HTTP分块传输对于动态生成内容非常有效
  - 由于服务器事先不知道生成内容的大小，如果使用Content-Length方法，则需将所有内容生成并缓存，才能计算长度并传输

# 如何追踪一个Web用户

HTTP Cookie

- Web站点使用 Cookie 来标记/追踪用户
- 由服务器发送给客户端，并由客户端保存一段时间
- 客户端接收到Cookie后，后面每次请求都将Cookie发送给服务器
- Cookie在HTTP头部中传输

- Cookie保存在浏览器中
  - 其他使用该浏览器的用户也会继续使用该Cookie
- 客户端可以从其他机器拷贝Cookie来继续访问服务器

# HTTP会话 (Session)

- 会话 (Session) 用于标识浏览器到站点的一系列请求/应答
  - 会话可以持续很长时间 (Web邮箱: 一周以上)
- 如果没有会话管理
  - 用户每次发送请求都需要进行再认证 (re-authenticate)
- 会话管理
  - 第一次请求时，对用户进行认证
  - 所有后续请求都和该用户绑定

# 会话令牌生成 (1)

## 不包含任何客户端状态

- 会话令牌是随机生成、且不可预测的字符串
  - 中间不包含任何用户数据
- 服务保存所有和该Token相关的信息
  - 用户ID、登录状态、登录时间等
- 导致服务的额外性能开销
  - 当主机提供多个Web服务时，需要查询多个数据库来获取用户状态

# 会话令牌生成 (2)

## 包含客户端状态

- 会话令牌生成方式如下：
  - SessID = [ userID， expire-time， user-data ]
  - SessToken = Enc-then-MAC (K，SessID)
    - K是该站点所有服务器共享的密码
- 服务器仍需要维护一些客户端状态
  - 例如，登出状态
- User-data中通常包含客户端IP地址
  - 缓解Cookie Theft攻击

# 存储会话令牌

- 浏览器Cookie

```
Set-Cookie: SessionToken=y3s2de
```

- 嵌入URL中

```
http://jd.com/checkout?SessionToken=y3s2de
```

- 放在HTML隐藏表单中

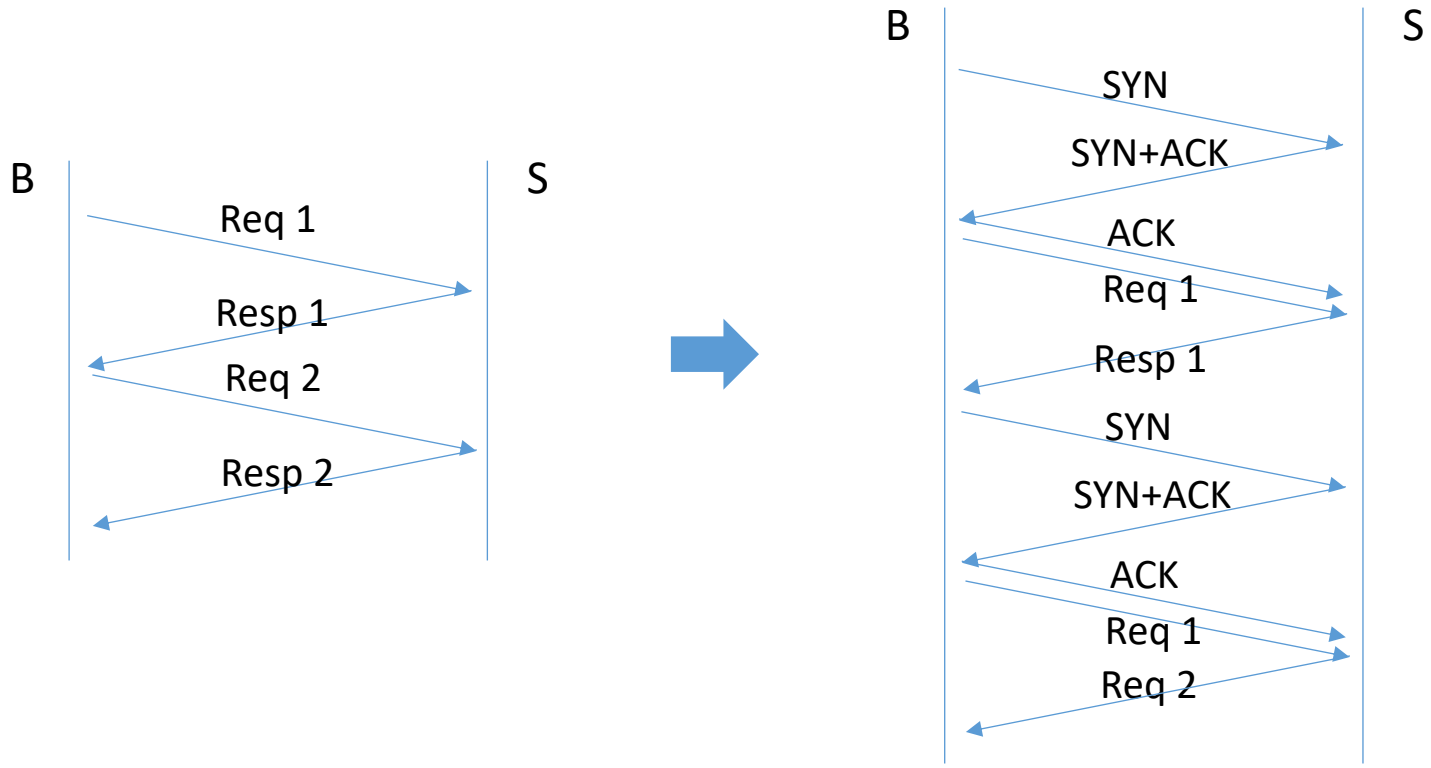
```
<input type="hidden"  name="sessionid"
value="y3s2de">
```

# 存储会话令牌：问题

- 浏览器Cookie
  - 浏览器每次请求都会附加Cookie
  - 跨站请求伪造攻击 (Cross-site request forgery)
- 嵌入URL中
  - HTTP Referer字段可能泄露会话令牌
- 放在HTML隐藏表单中
  - 只支持短时间的会话

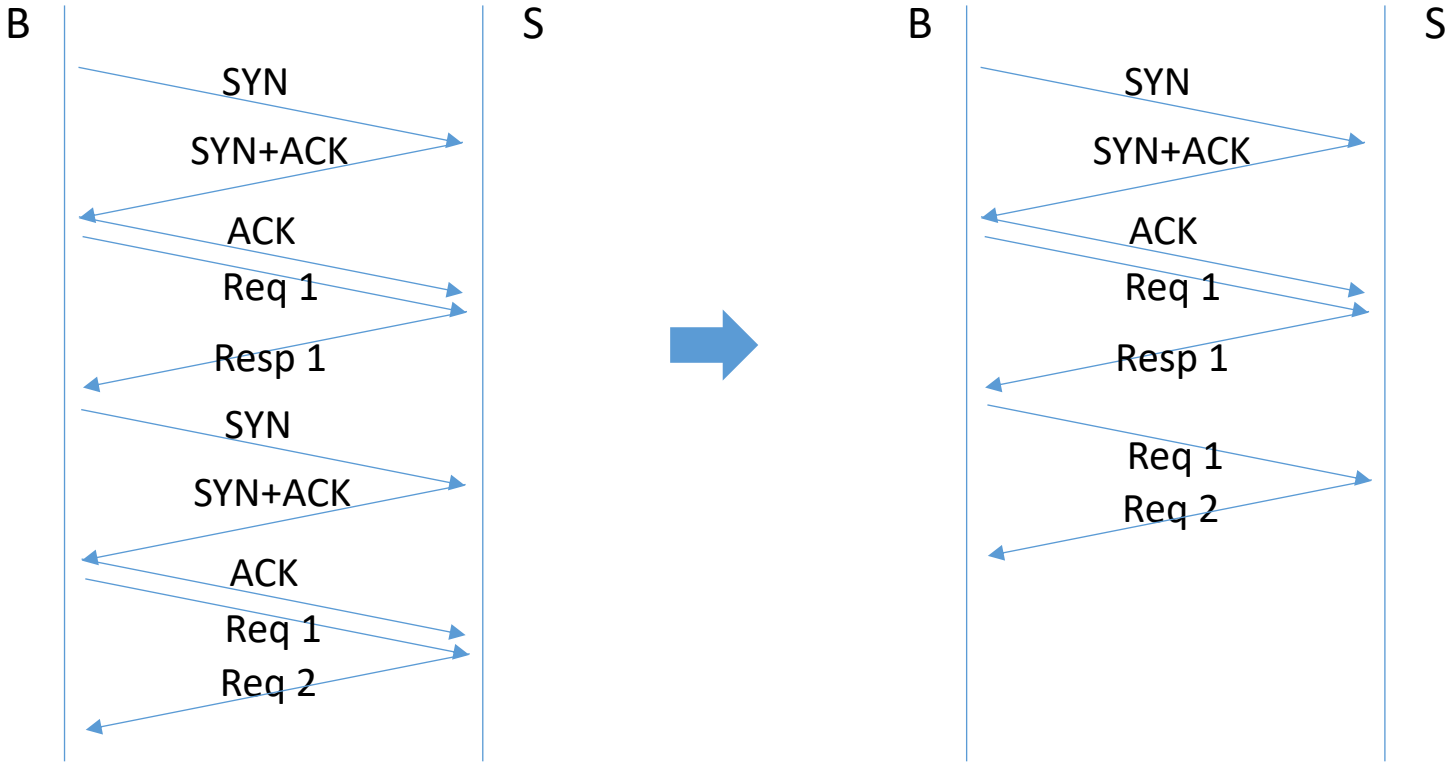
# 提升HTTP性能

- 早期HTTP协议(HTTP/0.9)为每个请求建立一个新的连接



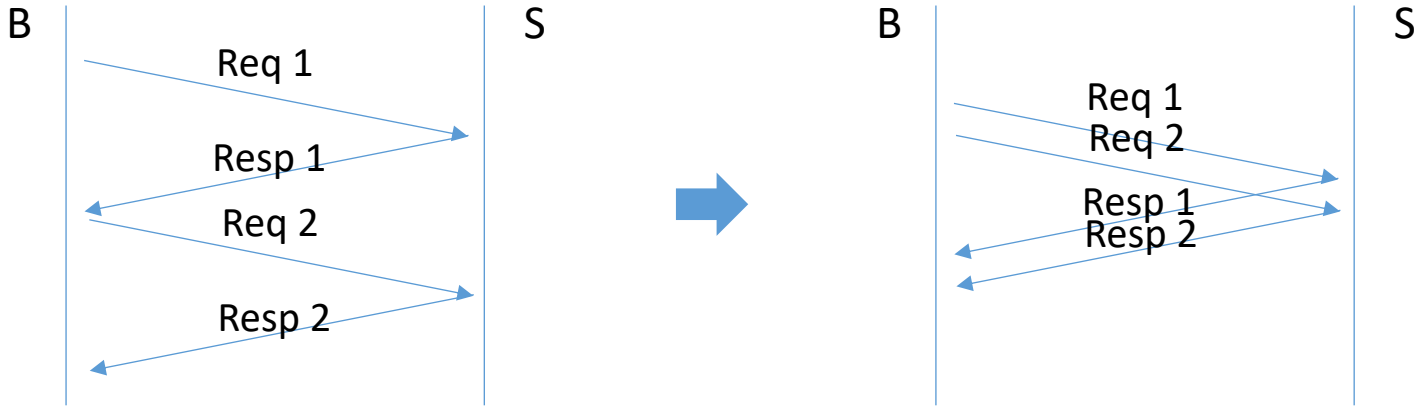
# HTTP持久连接

- HTTP持久连接(Keep-Alive)可使多个请求复用已有TCP连接
  - 节约了建立额外TCP连接的时间



# HTTP管道 (pipelining)

- HTTP持久连接(Keep-Alive)
  - 类似于停等机制 (Stop-and-Wait)，每单位时间只能处理一个请求
- HTTP管道利用窗口的思想，提升并行性，改进传输性能
  - 尽早发送请求，不用每次被应答阻塞，消除额外的往返延迟



# HTTP多连接

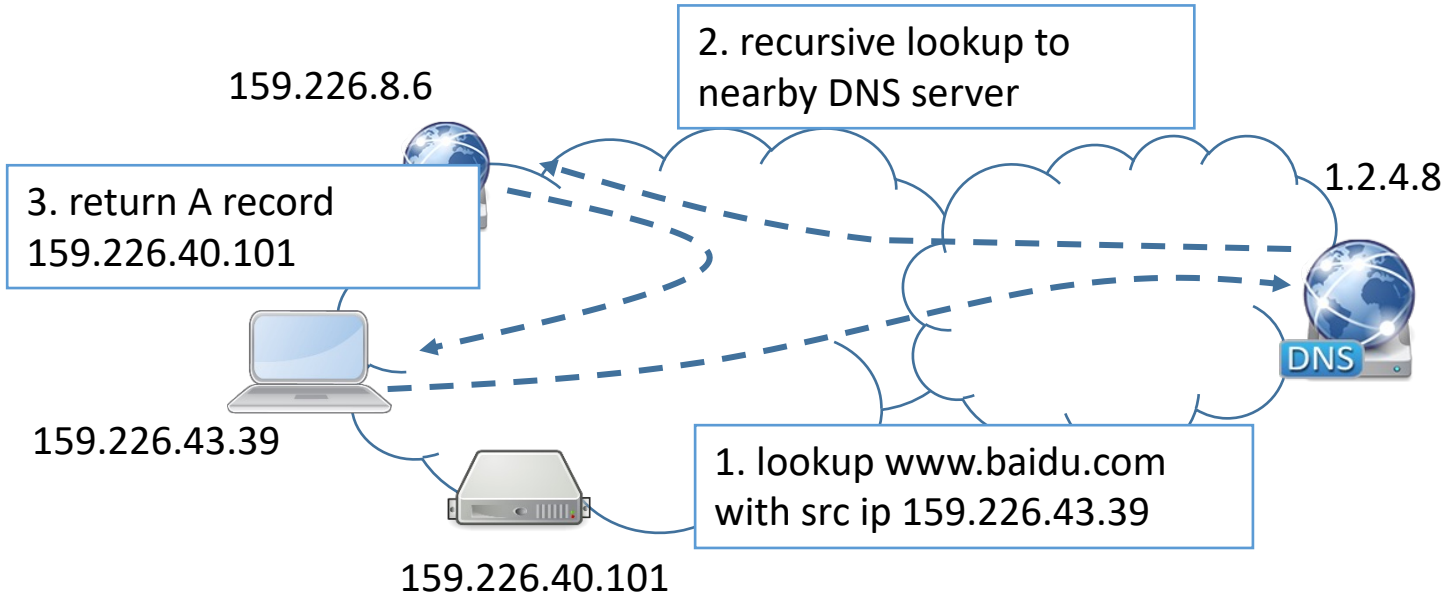
- 浏览器允许并行打开多个TCP连接
  - HTTP/1.x不支持pipelining数据交错到达
  - 最多允许6个并发连接
- 优点：
  - 相应报文可以交错到达
  - 相比于单连接，相当于TCP CWND变为原来的6倍
  - 绕过了TCP初始窗口小的问题
- 缺点：
  - 多个连接的维护消耗传输两端的更多资源、实现复杂性更高
  - 多个TCP流之间存在竞争，可能会造成网络拥塞

# 域名分区 (partitioning)

- 现代Web应用的复杂使得每个页面中包含很多资源
  - 平均每个Web页面包含90+个资源
  - 如果这些资源都来自同一主机，绝大部分的资源请求需要排队
- Web服务商可以将页面的资源分散到多个子域名
  - 优点：域名分区越多，并发性能越强
  - 缺点：
    - 每个新的主机名都需要一次DNS查询
    - Web服务商需手动分离，并部署到不同服务器

# 通过DNS选择就近服务器

- DNS通过就近DNS服务器选择
  - DNS支持递归查询
  - DNS服务器知道请求节点所在网络

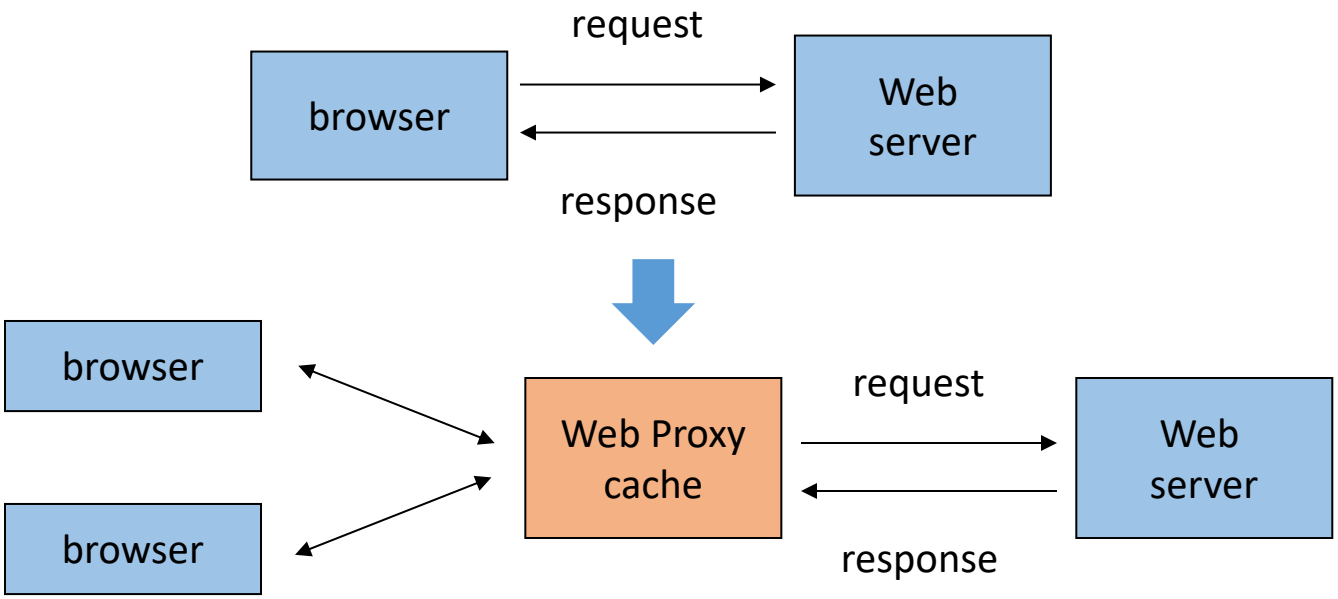


# Web缓存

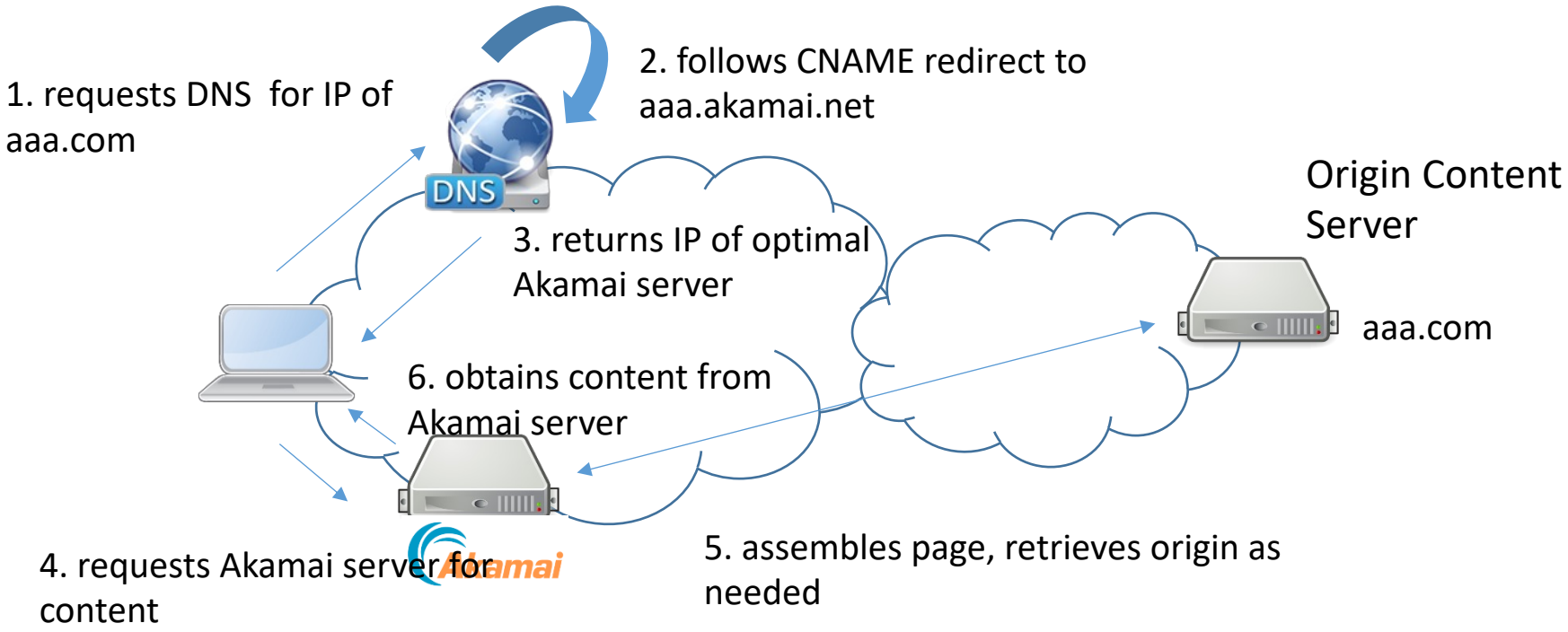
- 互联网访问服从Zipf分布
  - 对少数资源的请求占据了绝大部分的流量

$$f(k;s,N)=\frac{1/k^s}{\sum_{n=1}^{N}\frac{1}{n^s}}$$

- Web缓存不仅可减少网络流量，同时提升传输性能



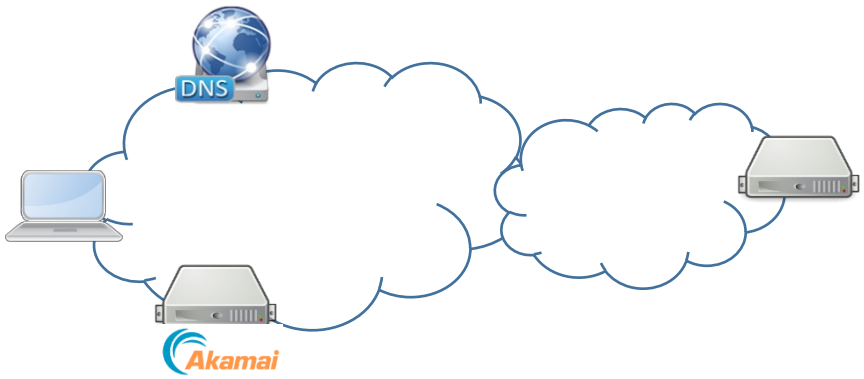
# 内容分发网络




**Content Delivery Network**

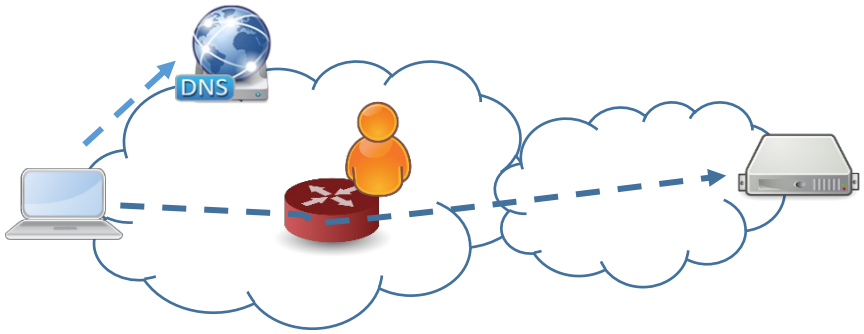
# 内容分发网络优点

- 改进用户体验(Quality of Experience)
  - 减少延迟
  - 减少网络丢包
- 减轻网络拥塞
- 减轻服务器负载
- 增加服务可扩展性
- 增强服务稳定性
- 降低运营成本



# HTTP安全

- 攻击者模型
  - 控制了网络基础设施：路由器、DNS服务器
- 例如，公共WiFi、甚至ISP



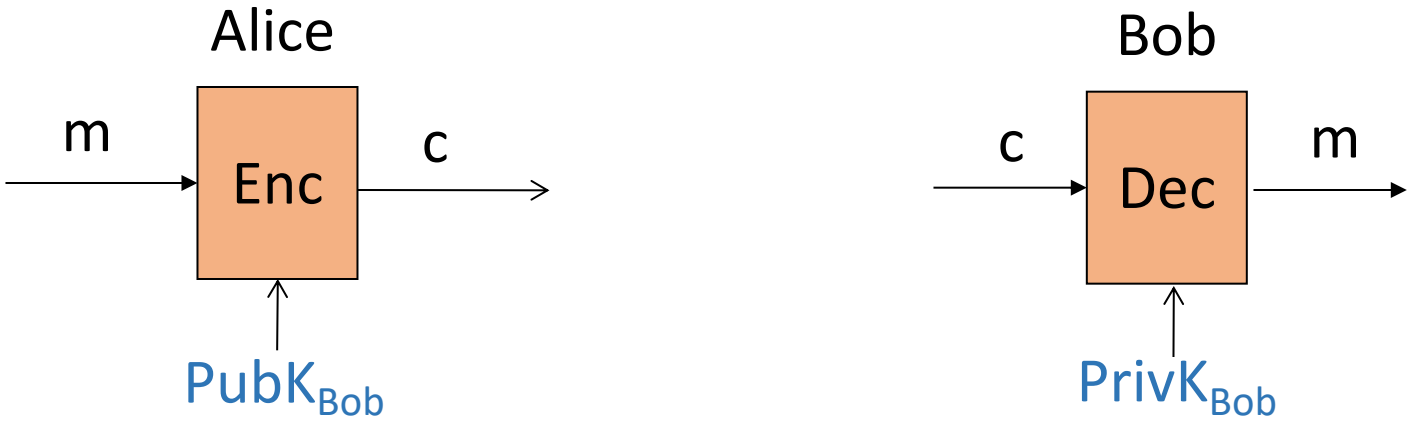

- 被动攻击
  - 监听网络流量
- 主动攻击
  - 监听、注入、拦截、篡改

# HTTPS

- HTTPS = HTTP + TLS
- 优点：
  - 很大程度上解决了互联网安全、隐私问题
- 缺点：
  - 对Web服务器造成一定性能负担
    - 加密、解密过程
  - 破坏了互联网缓存机制
    - ISP不能缓存HTTPS流量
    - 增加了Web服务器流量负担

# 非对称加密

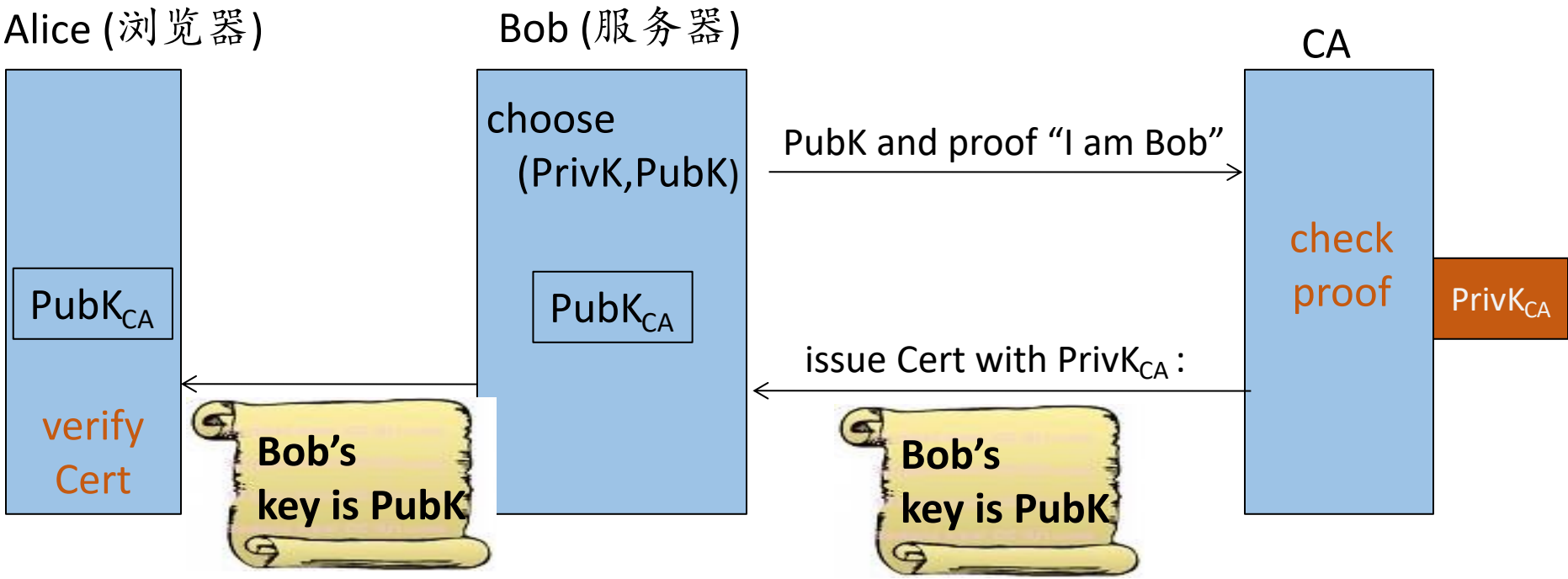
- 公钥加密体系



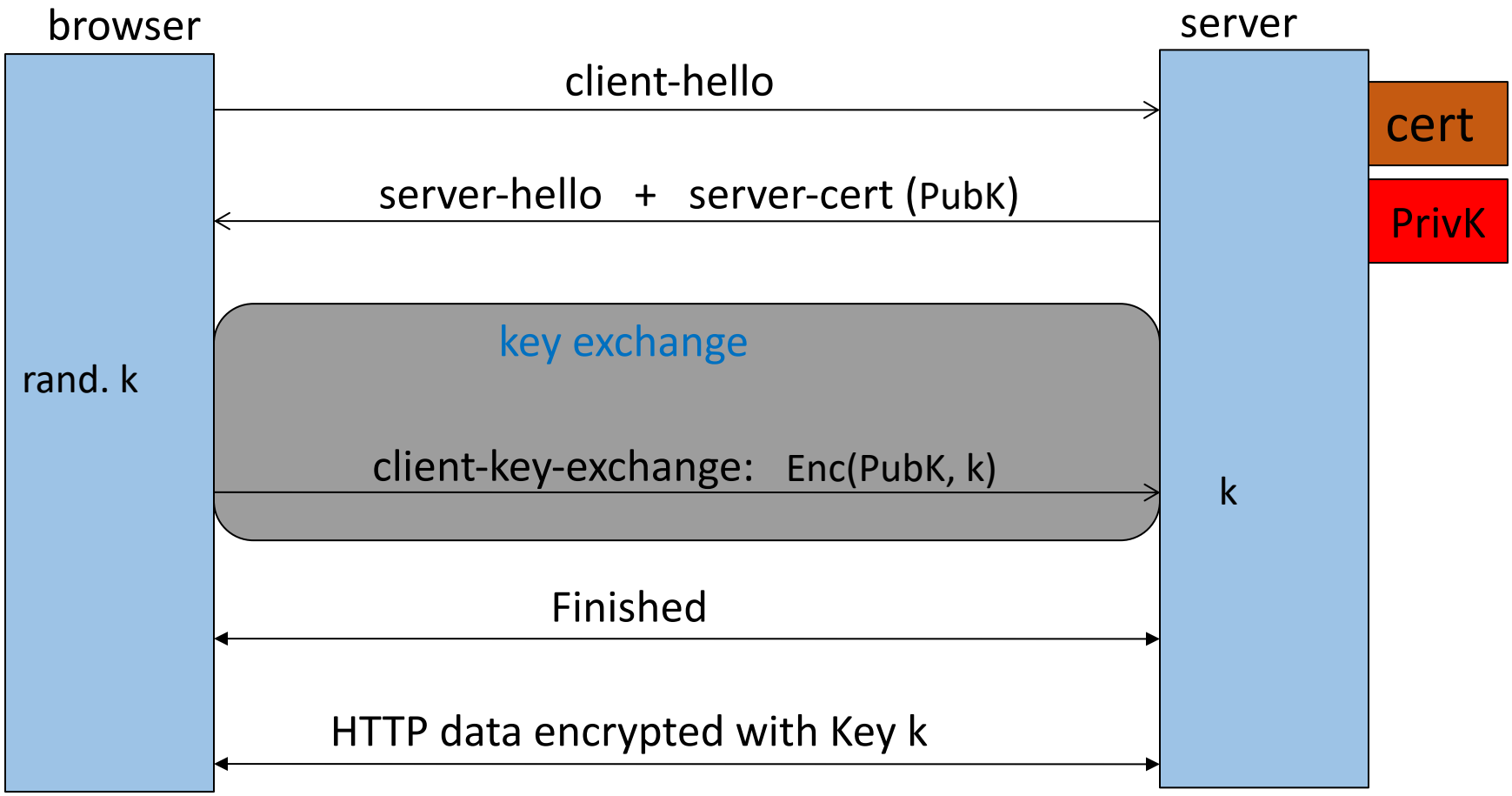

- Bob生成公钥对 ($PrivK_{Bob}$, $PubK_{Bob}$)
- Alice使用$PubK_{Bob}$对消息进行加密
  - 该加密消息只有Bob能解密

# 公钥认证 (证书体系)

- Alice如何获取Bob的公钥$PubK_{Bob}$
  - 如何防止第三者伪造Bob的公钥？



# SSL/TLS概览


browser
server
client-hello
cert
server-hello + server-cert (PubK)
PrivK
key exchange
rand. k
client-key-exchange: Enc(PubK, k)
k
Finished
HTTP data encrypted with Key k

# 生成自签名证书

```
# generate private key
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 \
-out cnlab.prikey

# generate self-signed certificate
openssl req -new -key cnlab.prikey -out cnlab.csr
openssl x509 -req -days 36500 -in cnlab.csr -signkey cnlab.prikey \
-out cnlab.cert
```

# 查看证书

openssl x509 -in cnlab.cert -text

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            16:08:e8:5f:f2:ad:b1:3e:9a:b1:77:b4:6d:74:6d:d2:91:3c:cb:c2
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = CN, ST = BJ, O = CNLab, CN = www.cnlab.cn
        Validity
            Not Before: Mar 29 02:41:57 2022 GMT
            Not After : Mar  5 02:41:57 2122 GMT
        Subject: C = CN, ST = BJ, O = CNLab, CN = www.cnlab.cn
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:a2:11:ea:d3:11:ba:a7:38:dd:9e:48:b7:25:03:
                    a1:ff:d5:e8:16:ec:d3:dd:a8:b8:fb:56:93:7d:16:
<< omitted >>
                    13:91:11:42:40:f1:e4:f8:49:7f:d9:14:04:58:48:
                    72:4b
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         2e:20:d6:1d:c8:9f:aa:ce:a2:da:ea:65:ac:cb:13:3b:5b:b1:
         5e:2a:fe:d4:73:c3:d8:d3:3d:55:bd:70:b9:bc:0b:0a:ae:d5:
<< omitted >>
         3b:d1:b3:7d:c3:9d:ce:de:d3:df:43:7c:f1:e4:f4:b7:90:11:
         48:d9:f1:a3
```

# 使用公钥/私钥进行加/解密数据

```
# extract pubkey from certificate

openssl x509 -pubkey -noout -in cnlab.cert > cnlab.pubkey

# or from prikey

openssl rsa -in cnlab.prikey -pubout > new-cnlab.pubkey


# encrypt secret.txt (data less than 200 bytes)

openssl rsautl -encrypt -inkey cnlab.pubkey -pubin -in secret.txt -out secret.enc

# decrypt

openssl rsautl -decrypt -inkey cnlab.prikey -in secret.enc -out new_secret.txt

# diff

md5sum secret.txt new_secret.txt
```

# Socket使用SSL/TLS通信的例子

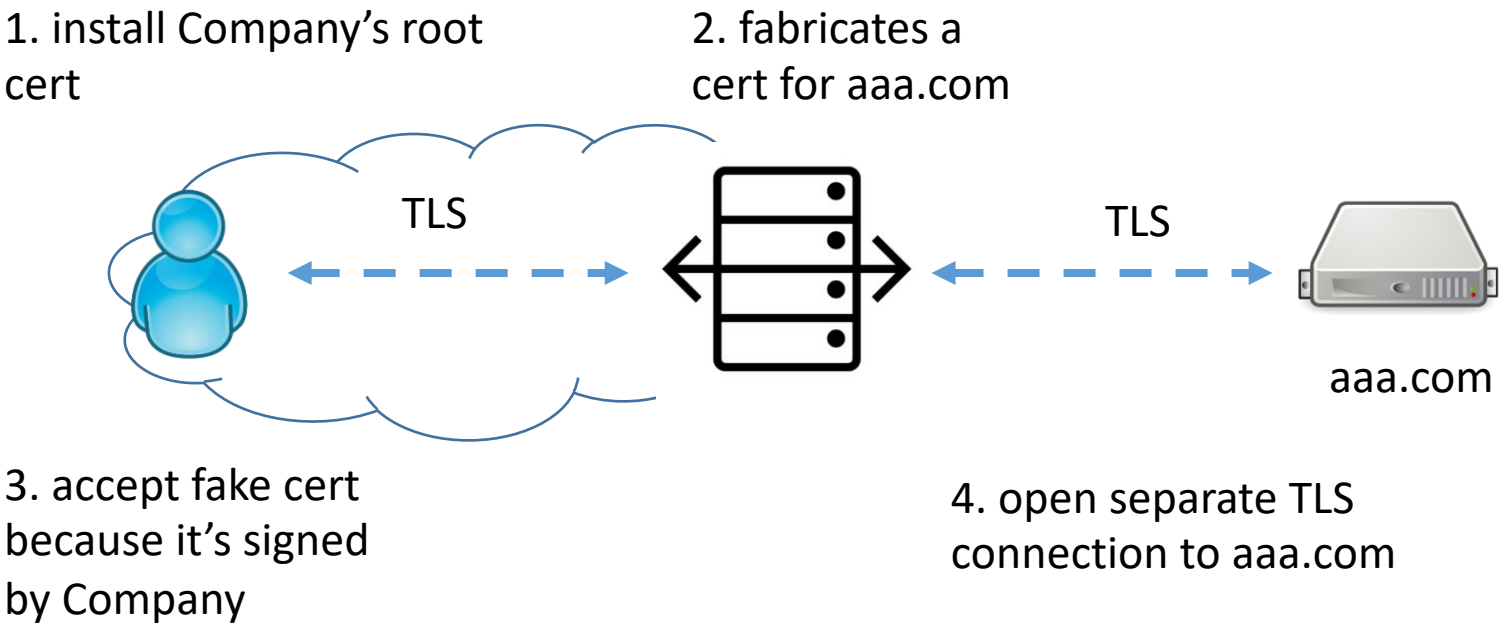
```
// init SSL Library
OpenSSL_add_all_algorithms();
SSL_load_error_strings();

const SSL_METHOD *method = TLS_server_method();
SSL_CTX *ctx = SSL_CTX_new(method);

// load certificate and private key
SSL_CTX_use_certificate_file(ctx, "./keys/cnlab.cert", SSL_FILETYPE_PEM);
SSL_CTX_use_PrivateKey_file(ctx, "./keys/cnlab.prikey",
SSL_FILETYPE_PEM);
```

# Socket使用SSL/TLS通信的例子（续）

```
int sock = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;
addr.sin_port = htons(443);

bind(sock, (struct sockaddr*)&addr, sizeof(addr));
listen(sock, 10);

int csock = accept(sock, (struct sockaddr*)&caddr, &len);

SSL *ssl = SSL_new(ctx);
```

# Middlebox + s (Secure)

- 以Web proxy为代表的Middlebox不能参与到HTTPS会话中



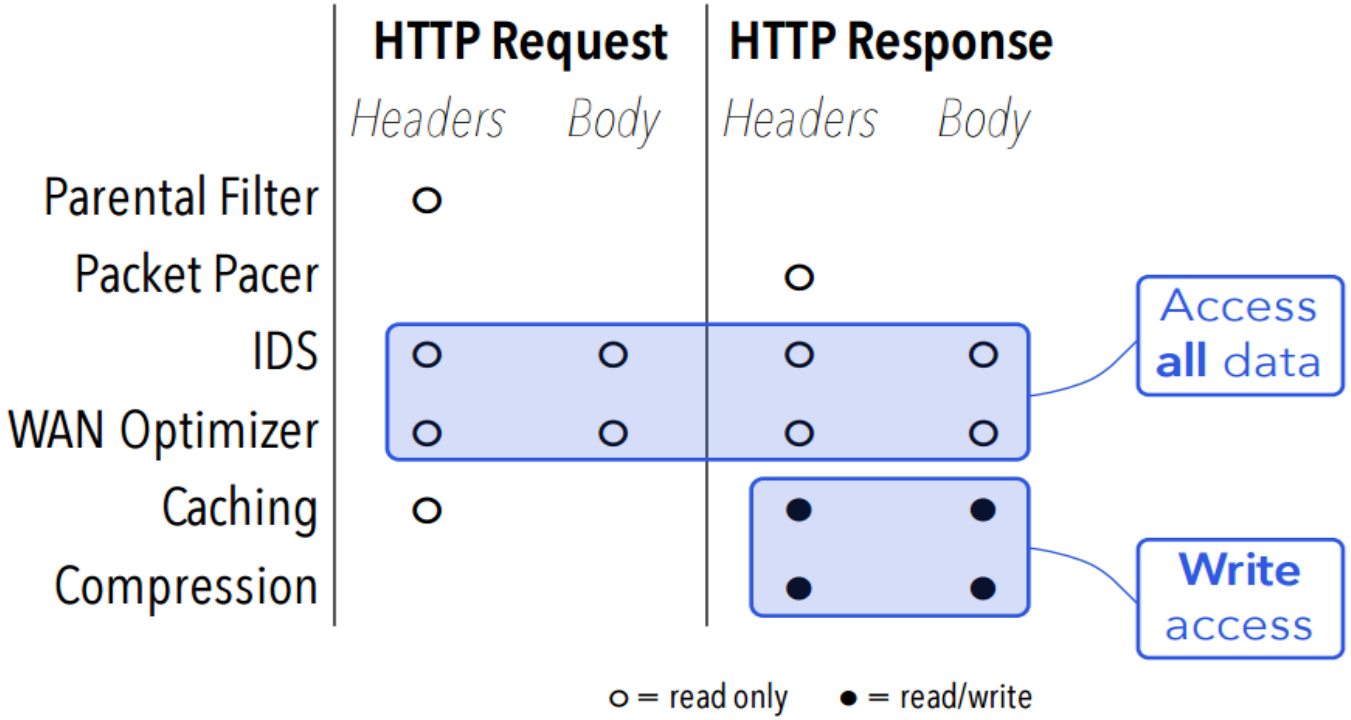

一个不成功的方案

# Middlebox与SSL/TLS

- SSL/TLS为传输两端进行认证和加密
  1. 实体认证 (Entity Authentication)
  2. 数据保密 (Data Secrecy)
  3. 数据完整性 (Data Integrity)
- Middlebox通过伪造证书支持TLS
  - 客户端数据在Middlebox处不再安全
  - Middlebox对数据访问具有完全的读/写权限

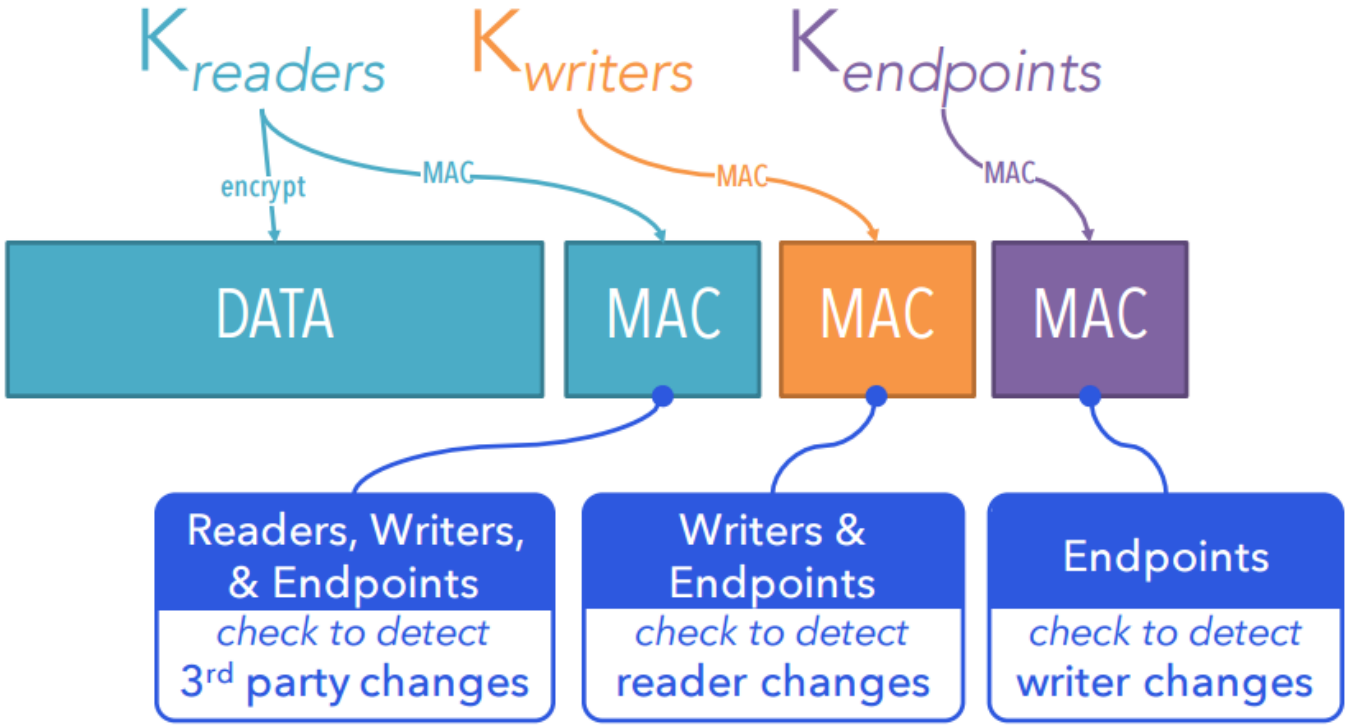
# Middlebox与数据访问

- 不是每种Middlebox都需要对所有数据具有完全读写权限

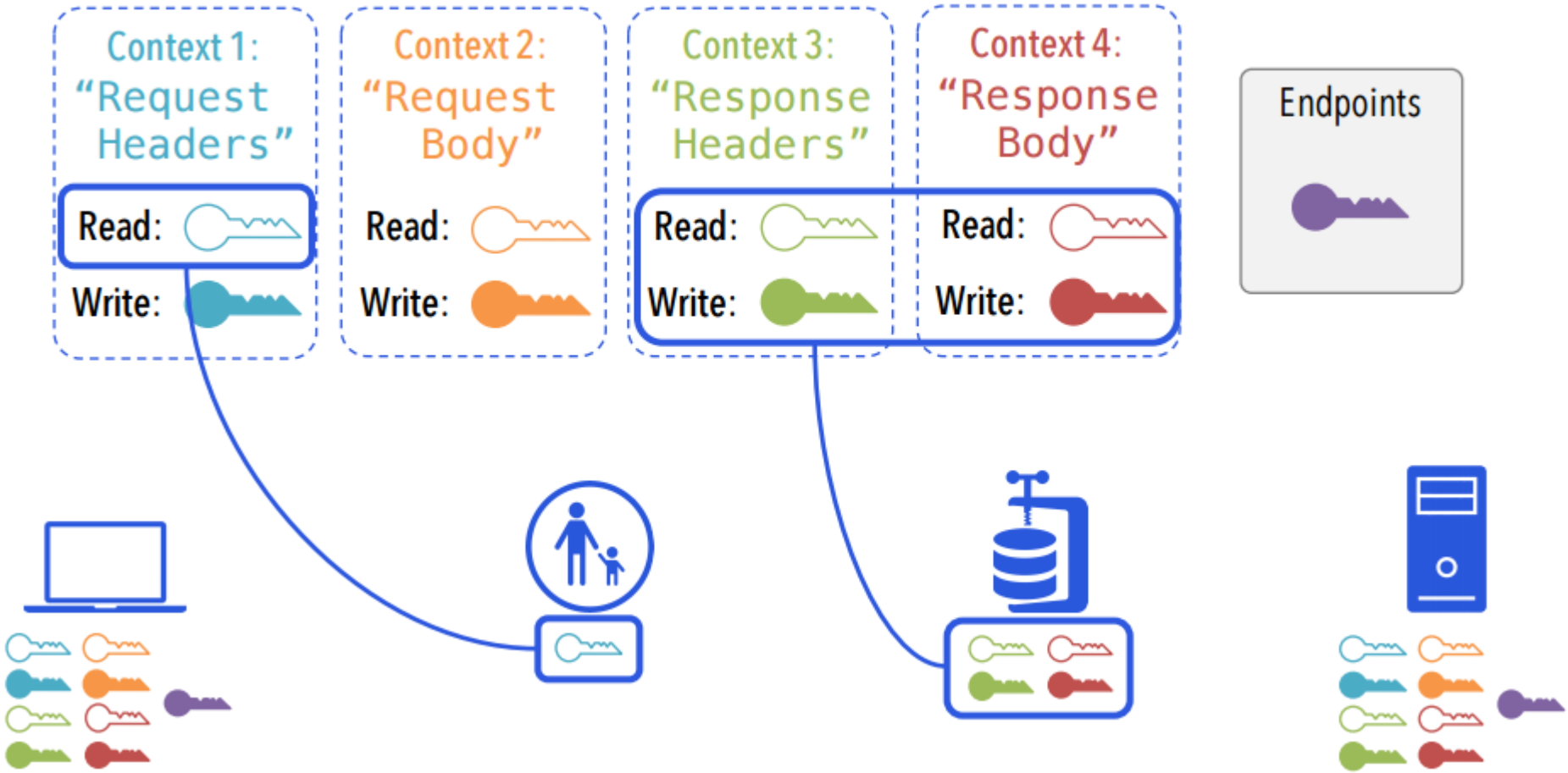


# 数据加密权限的分离

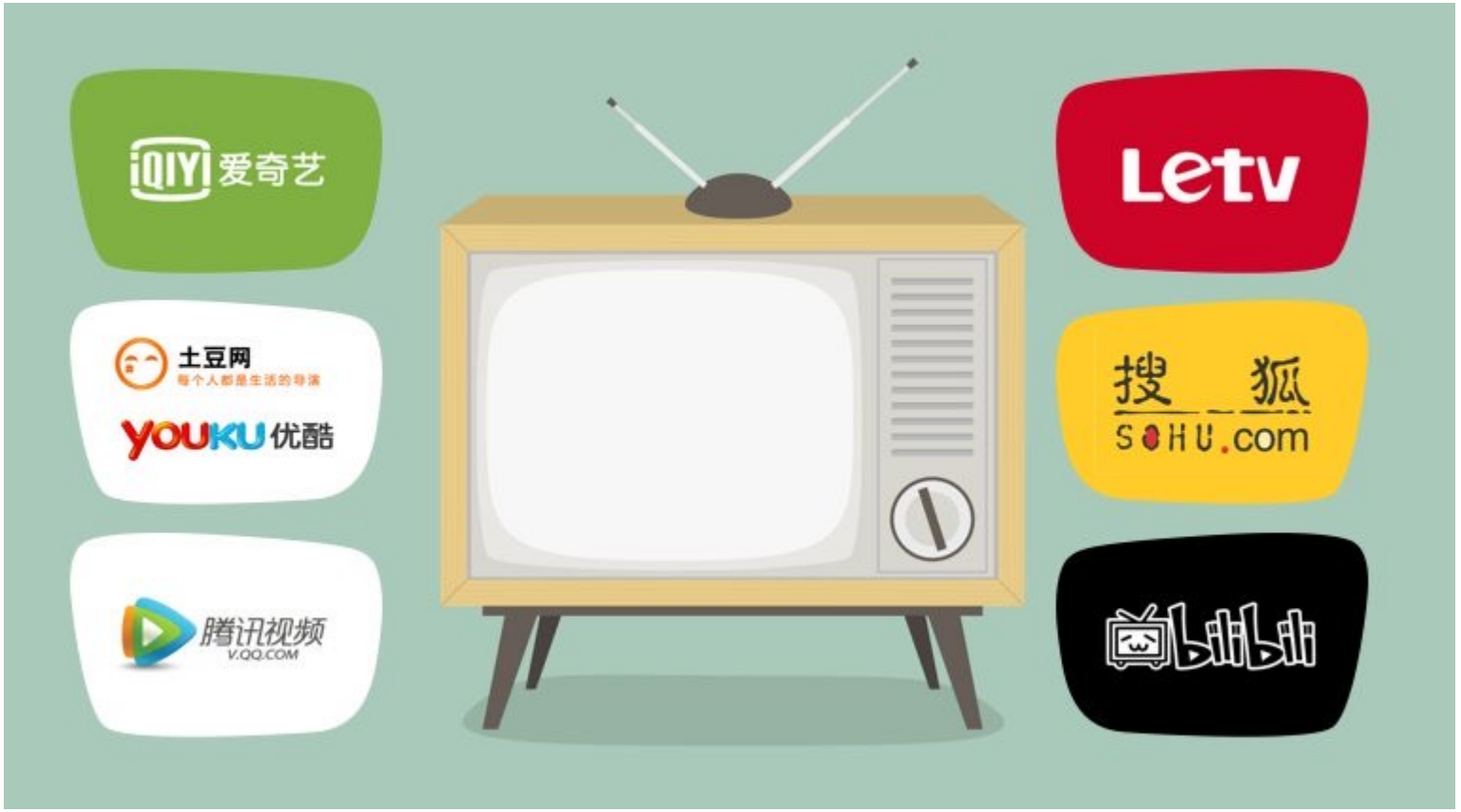
- 使用3个Key来分离只读权限和读写权限



# Middlebox支持TLS示例



# 互联网视频（Internet Video）



# 互联网视频 设计方案一

- 基于TCP的视频传输
  - 浏览器向服务器请求视频的元数据(Meta data)，启动播放器
  - 播放器与视频服务器建立TCP连接，请求视频文件并播放
- 优点：
  - 实现简单；元数据与视频文件分离，支持CDN，支持不同码率
- 缺点：
  - TCP连接尽力而为的传输可能浪费网络带宽
  - 数据传输与视频播放的播放/跳过/停止控制机制不匹配
  - 视频播放质量不能自适应网络带宽

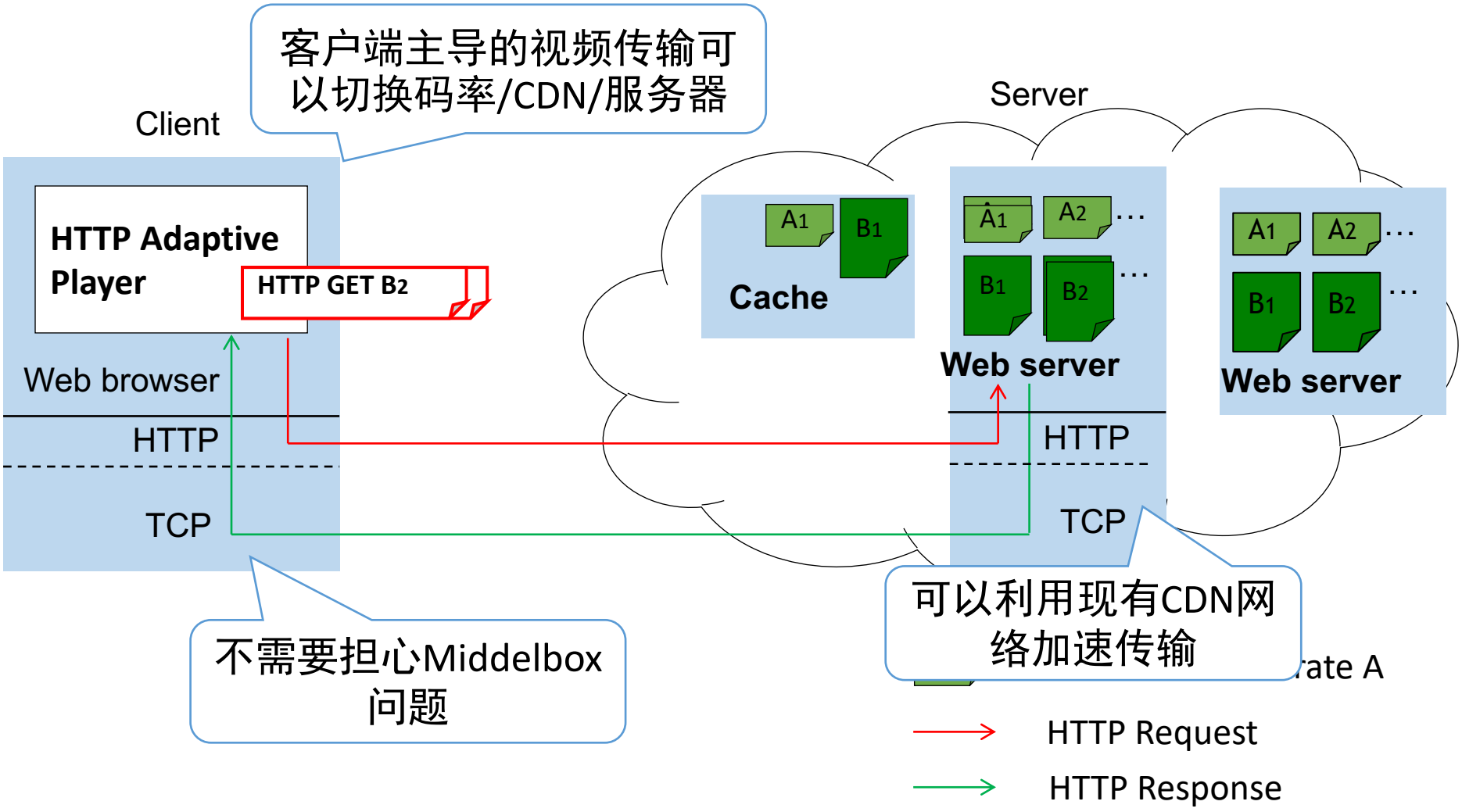
# 互联网视频 设计方案二

- 带状态的视频流传输
  - 将视频的数据传输和播放控制进行分离
  - 视频服务器维护每个视频播放的状态
  - 播放控制：开始、暂停、前进，…
  - 数据传输：基于TCP或UDP
- 优点：
  - 可以精确控制，可支持自适应码率
- 缺点：
  - 视频服务器维护会话状态需要额外的开销
  - 网络中广泛存在的Middlebox可能会阻断视频流传输

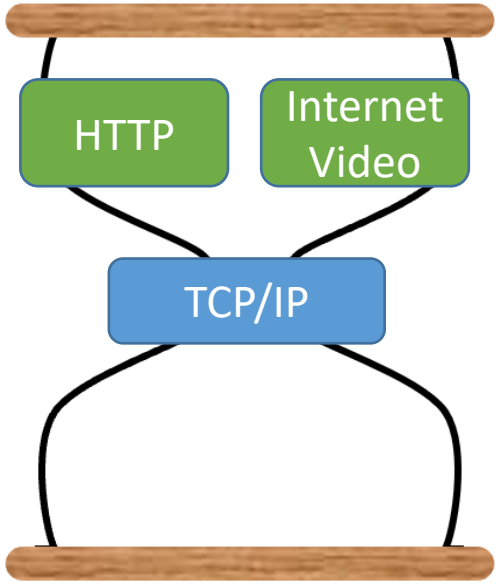
# 基于HTTP的视频流传输

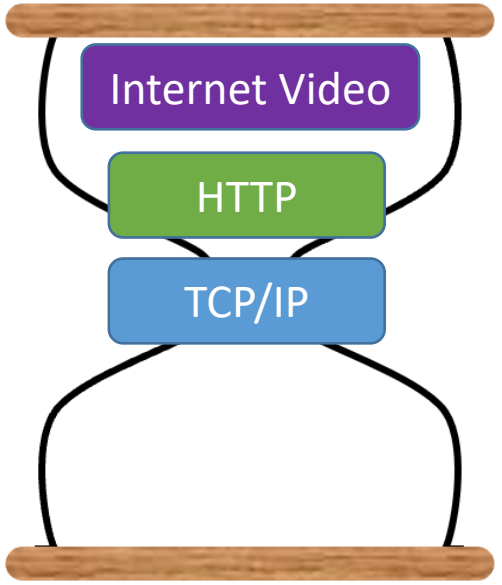
- Observation: 与其让互联网适配视频传输，不如让视频传输适配互联网
- HTTP流传输
  - 视频文件分割成多个块（Chunk），每个块由独立帧开始
  - 每个视频块有不同码率的版本，以适应不同的网络带宽
  - 客户端在同一个会话中可以请求不同码率的视频块
- 优点：
  - 完全标准HTTP协议，实现简单
  - 会话状态和控制逻辑由客户端维护，减轻服务器负担

# HTTP流传输示意


客户端主导的视频传输可以切换码率/CDN/服务器
Client
Server
HTTP Adaptive Player
HTTP GET B2
Web browser
HTTP
TCP
Cache
A1
B1
A2
B2
Web server
Web server
HTTP
TCP
不需要担心Middelbox问题
可以利用现有CDN网络加速传输
rate A
HTTP Request
HTTP Response

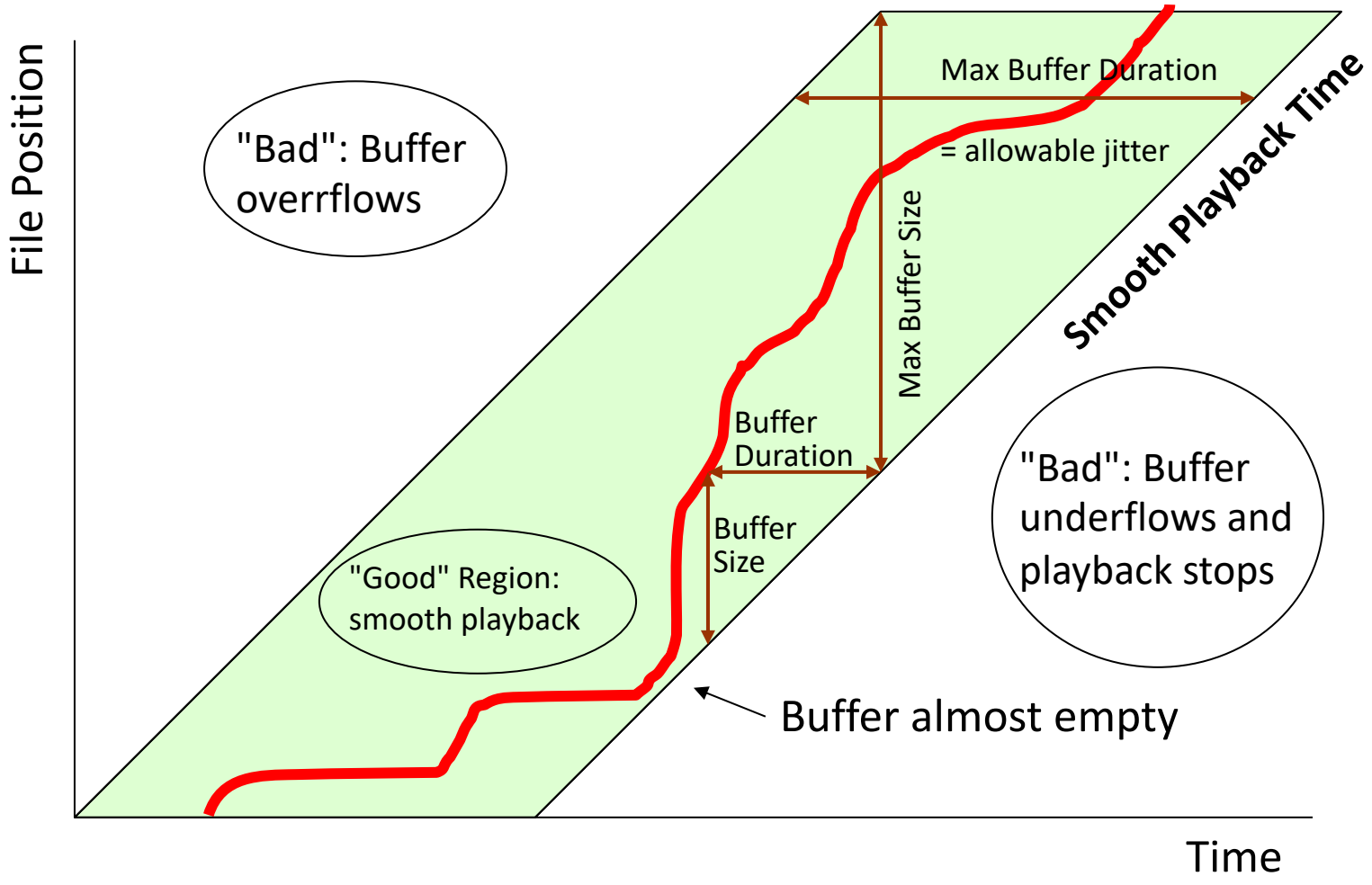
# 互联网体系结构与互联网视频



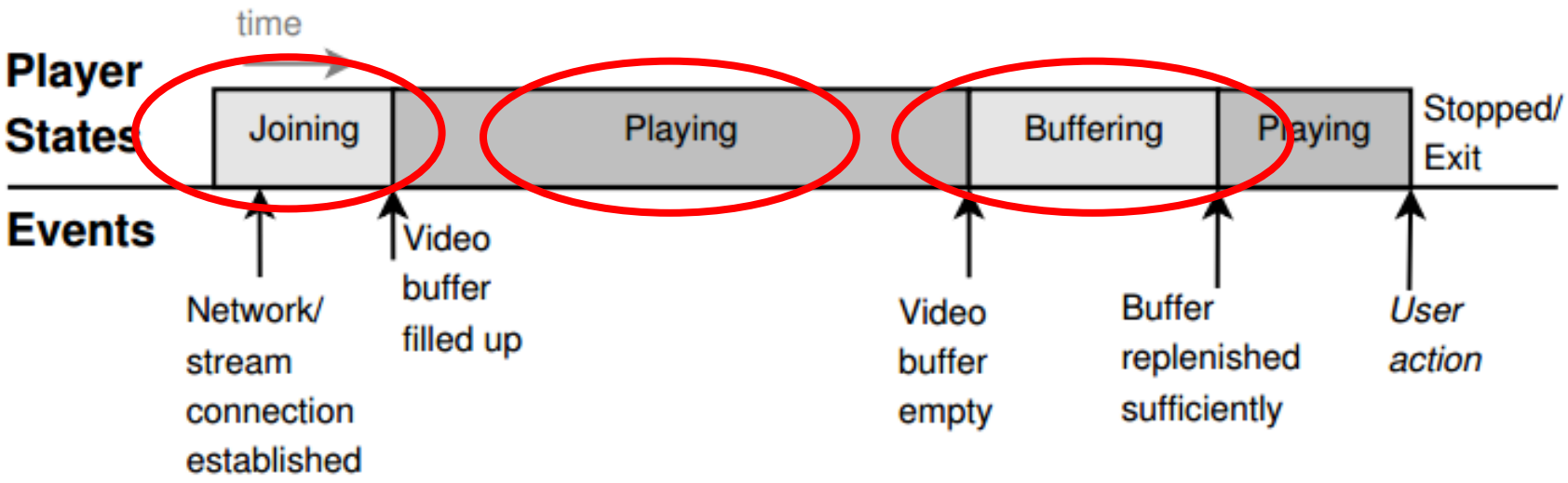




- 互联网视频从90年代的基于TCP/IP到现在的基于Web
  - 现有互联网已经针对HTTP做了很多适配工作，基于Web的互联网视频可以充分利用这些特性：DNS、CDN、Middlebox、…
  - HTTP将来可能成为互联网体系结构的细腰

# 视频播放与缓冲区大小


File Position
"Bad": Buffer overrflows
Max Buffer Duration
= allowable jitter
Smooth Playback Time
Max Buffer Size
Buffer Duration
Buffer Size
"Good" Region: smooth playback
"Bad": Buffer underflows and playback stops
Buffer almost empty
Time

# 互联网视频性能指标




***Join Failures (JF)***

***Join Time (JT)***

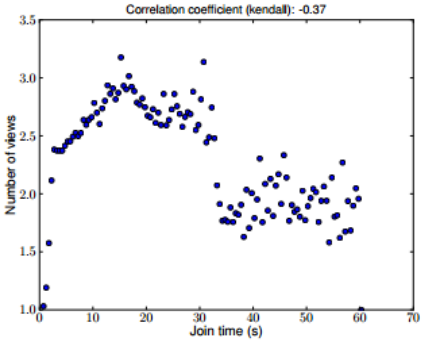
***Avg Bitrate (AB)***

***Buffering Ratio (BR)***

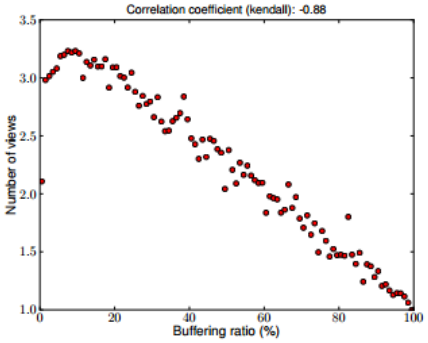
***Rate Of Buffering (RB)***

# 视频性能对用户参与度的影响

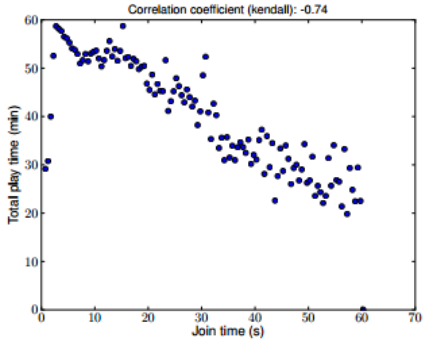
- 分析方法
  - Kendall相关性分析
  - 信息增益(Information Gain)
  - QED(Quasi-Experiment Design)



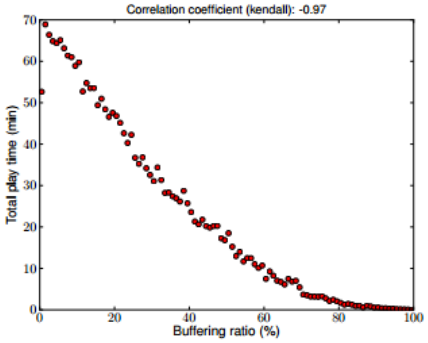

(a) Join time, # views




(b) Buffering ratio, # views




(c) Join time, Play time




(d) Buffering ratio, Play time

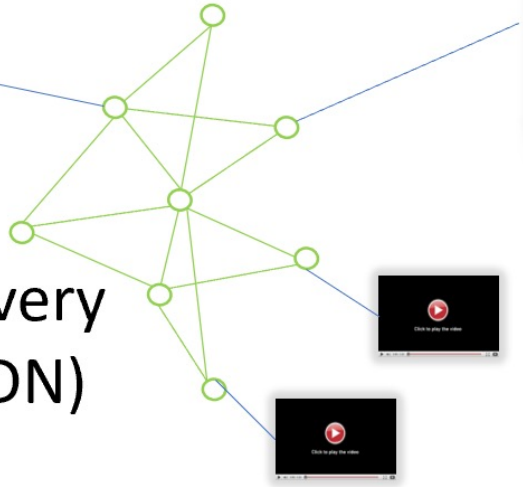
[ Dobrian 2011 SIGCOMM ]

# 提升互联网视频性能

Video Servers

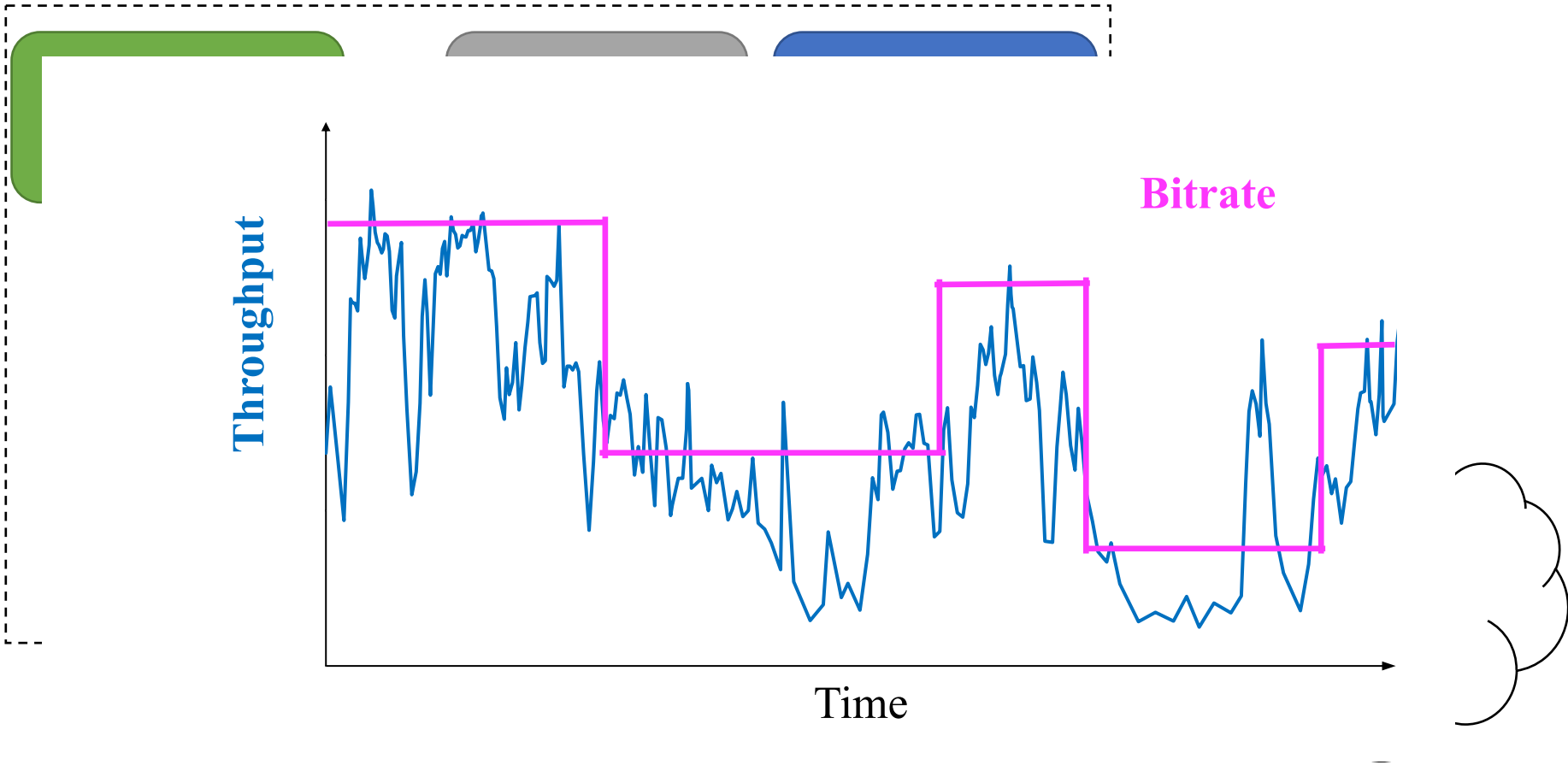
ISP & Home Net

Video Player

Content Delivery
Networks (CDN)

# 自适应码率（Adaptive BitRate）模型



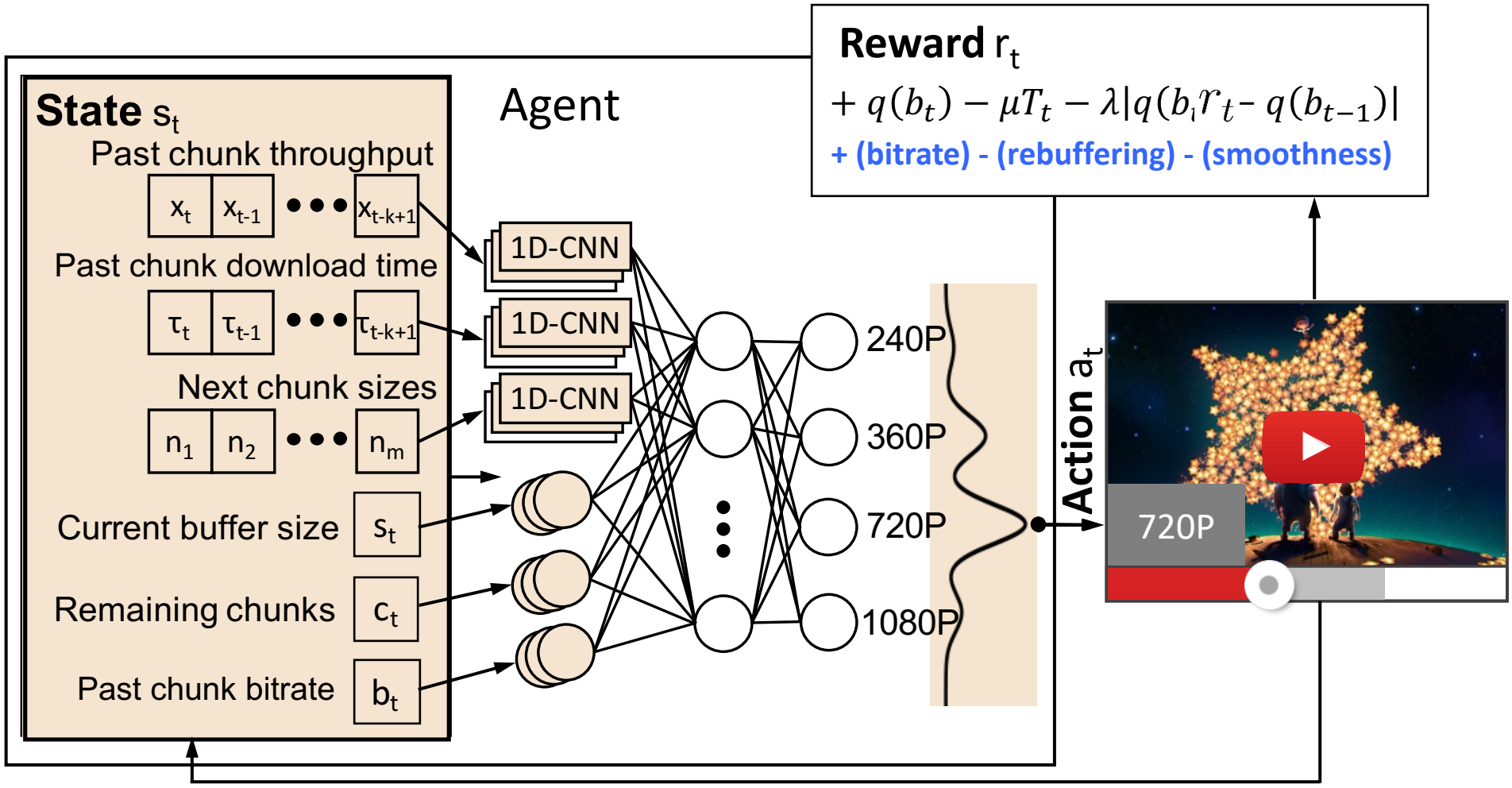

客户端接收视频数据块，决策调度策略，形成闭环反馈控制

# 自适应码率问题

- 自适应码率问题本质上是吞吐率预测问题

- 该问题的难点在于
  - 网络可用带宽变动非常快
  - QoE之间指标相互冲突
    - 高码率、低缓冲时间、码率切换尽可能少
  - 基于对吞吐率预测的码率选择对后续选择有影响

# 基于深度强化学习的自适应码率




[ Mao SIGCOMM 2017 ]

# 课后阅读

- 《计算机网络 – 系统方法》
  - 第8.4.3 、9.1.2、9.1.3、9.3.1节
- Web应用性能优化
  - Ankit Singla et al. The internet at the speed of light. ACM HotNets 2014
  - WANG Xiao et al. Speeding up web page loads with Shandian. USENIX NSDI 2016
- 互联网视频传输性能分析与优化
  - F. Dobrian et al. Understanding the Impact of Video Quality on User Engagement. ACM SIGCOMM 2011
  - H. Mao et al. Neural Adaptive Video Streaming with Pensieve Hongzi Mao. ACM SIGCOMM 2017


谢谢！