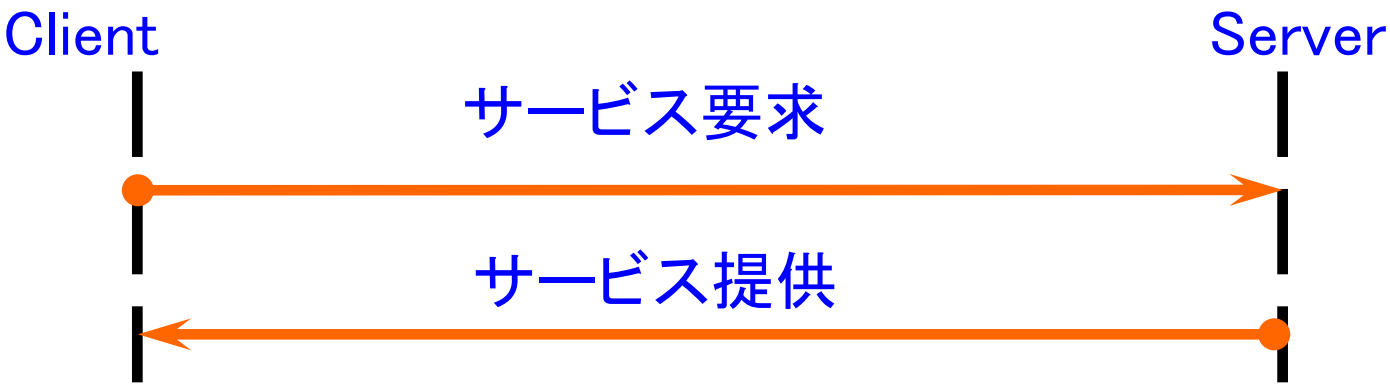
WEB技術

WWWの生い立ち

- Tim Berners-Lee (CERN), 1989年
  - 高エネルギー物理学資料のハイパーリンク化
- ハイパーテキスト技術
  - マークアップ言語: HTML
  - 転送プロトコル：HTTP
- Mac Andreessen (イリノイ大), 1993年
  - Mosaic ➔ インターネットの一般化のトリガ
- Netscape Navigator, Internet Explorer
- WWWコンソーシアム(W3C)

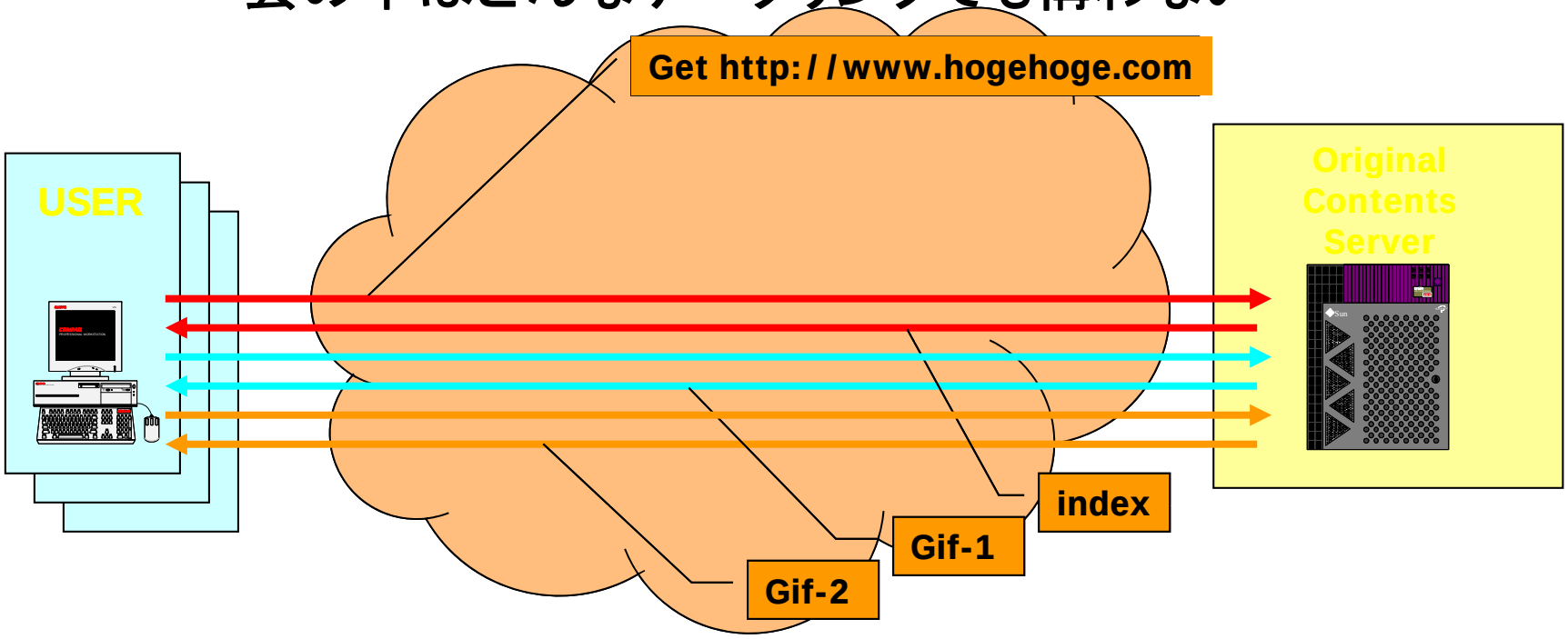
client-server と peer-peer(1)

- クライアント
  - 能動的にサービス提供を促す側
- サーバ
  - 受動的にサービス提供する側



通常のWEBコンテンツの流れ

“雲の中はどんなデータリンクでも構わない”



一度のhttpリクエストに際し、ファイル形状ごとにそれぞれTCPセッションの確立から始めなければならないため、アクセス数の増加に対しサーバの負荷が指数関数的に増加する

要は、大変たくさんの TCP コネクション

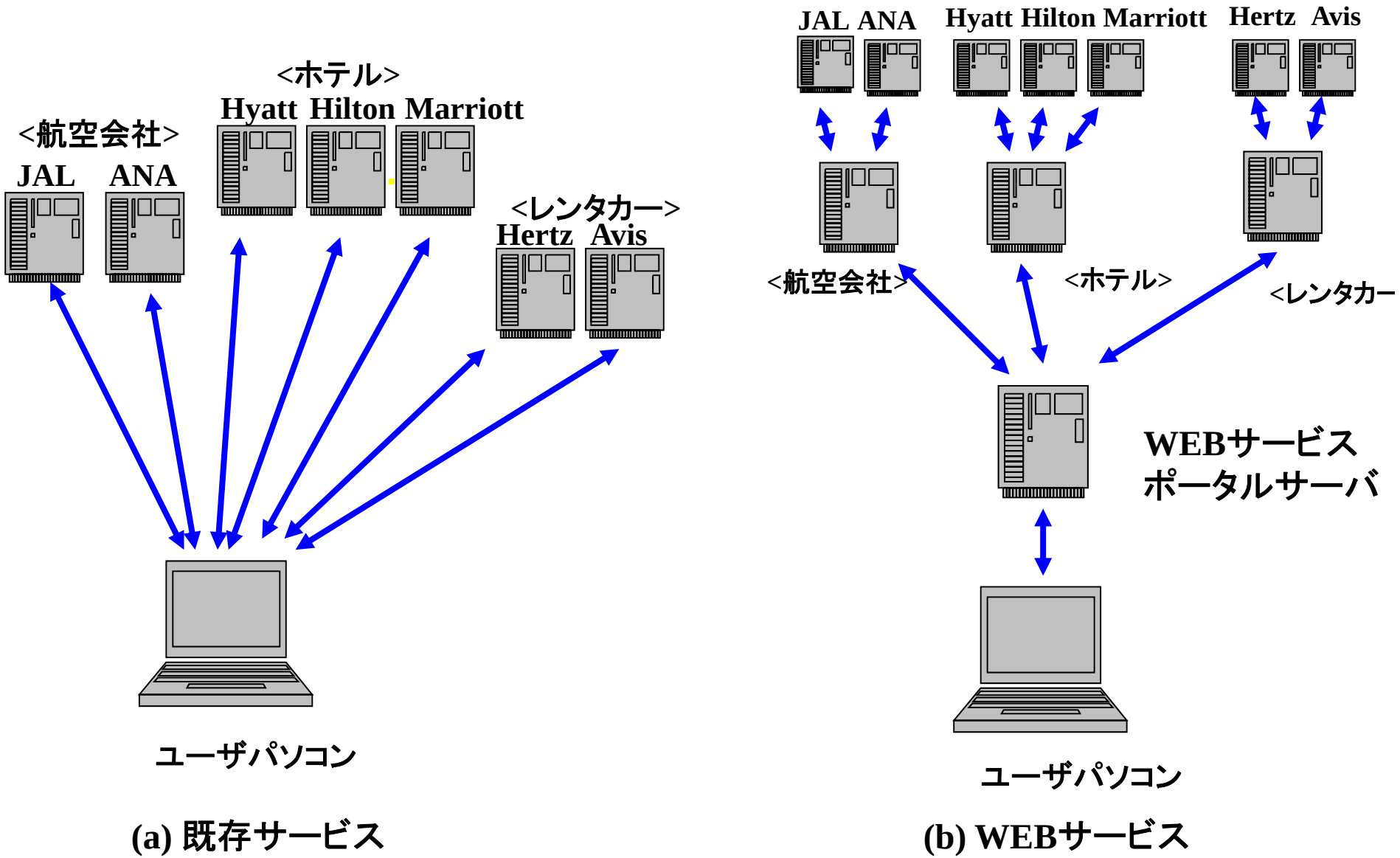


図8-2 WEBサービスの基本概念

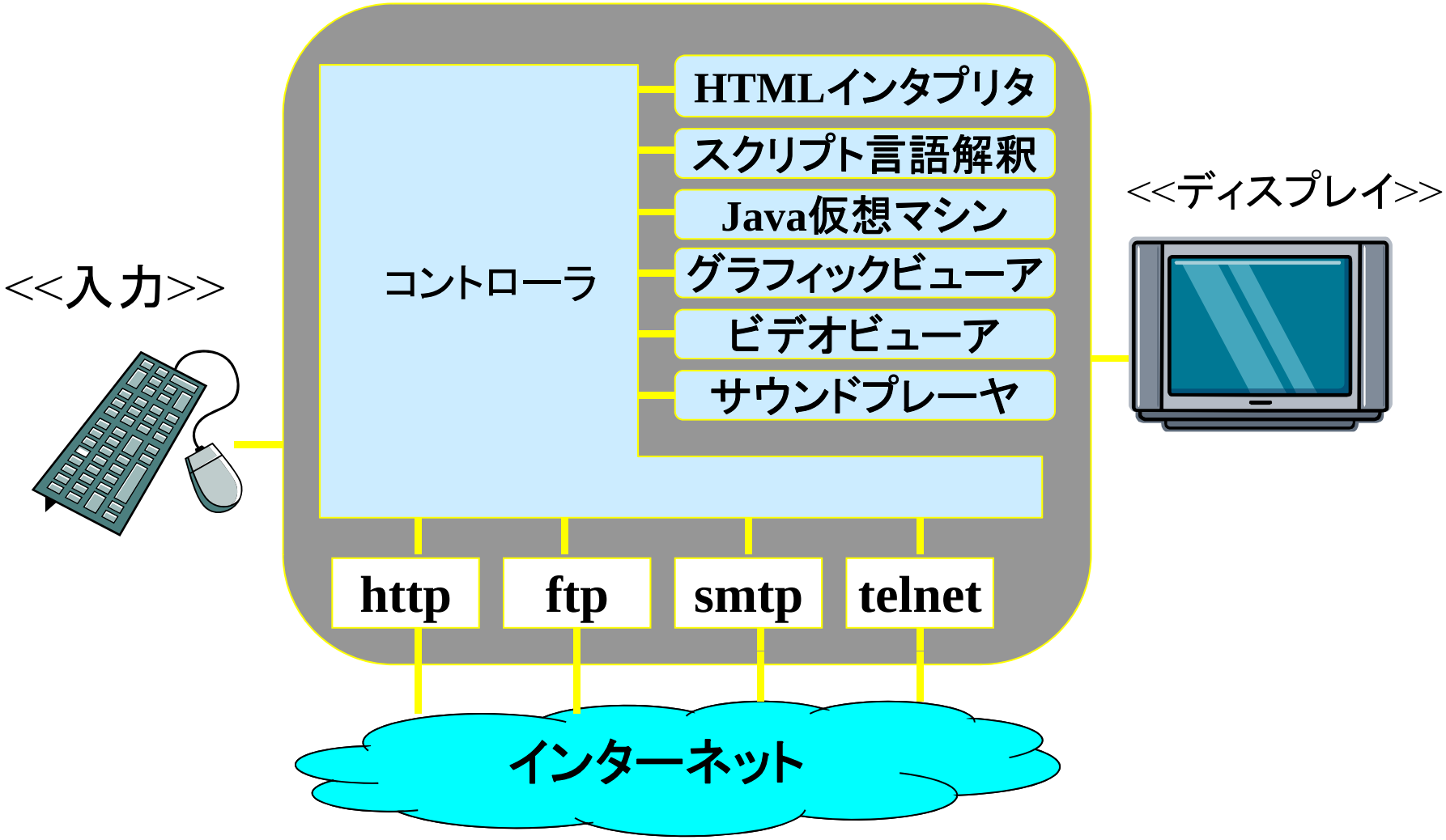
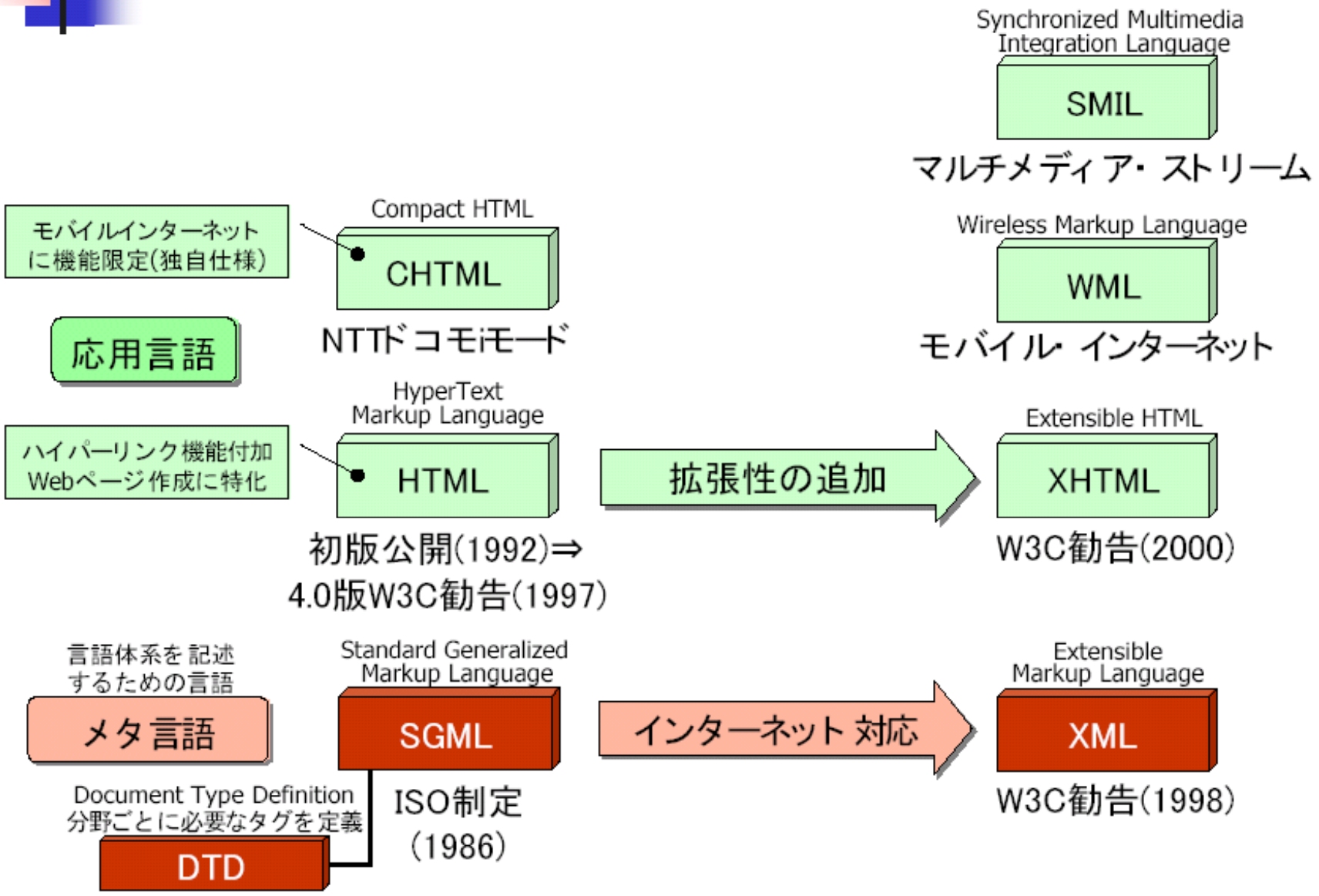


図8-4 WEBブラウザの基本構造

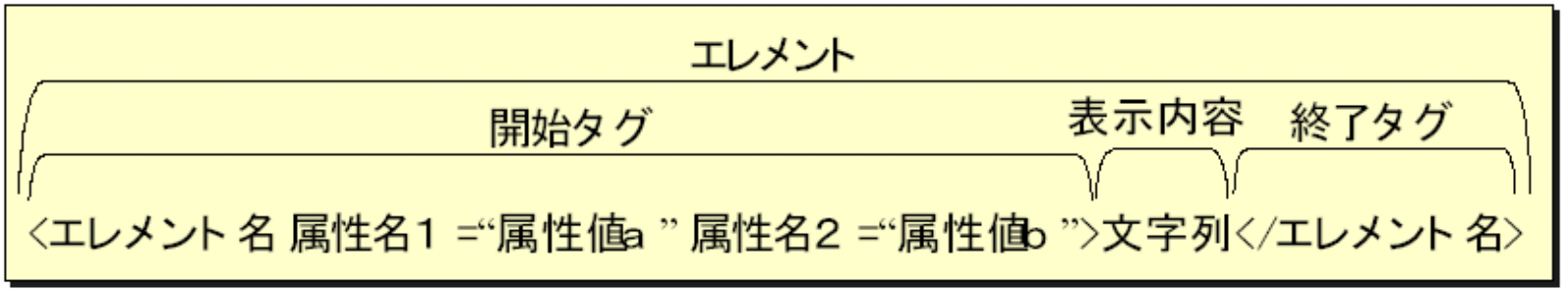
マークアップ言語

- 文書中に タグ を埋め込むことによって、文書の論理構造(表題、段落など)や表示属性(レイアウト、文字サイズ、色など)を規定する言語

マークアップ言語の発展



タグの表記規則



- 見出しの記述例
  <h1 align= ″center″ > HTML言語とWebページの作成 </h1>
- ハイパーリンク先の指定
   文字列

アンカ:リンクの基点

- HTML言語では終了タグのないものもある
  例: <img>,
←アプリ開発を複雑化

HTML言語の記述と表示例

ヘッダ部

```
1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2: <html>
3: <head>
4: <title>address</title>
5: </head>
```

ボディ部

```
6: <body>
7:    <h1 style="margin-left:100pt">アドレス帳</h1>
8:    <table border="3">
9:       <tr style="background-color: yellow"><th>氏名</th><th>住所</th><th>郵便番号</th><th>電話</th></tr>
10:         <tr><th>東京太郎</th><th>東京都新宿区</th><th>160-1234</th><th>03-1234-5678</th></tr>
11:         <tr><th>大阪花子</th><th>大阪府大阪市</th><th>535-4321</th><th>06-8765-4321</th></tr>
12:         <tr><th>青森鈴子</th><th>青森県青森市</th><th>038-7654</th><th>017-765-4321</th></tr>
13:      </table>
14:      <a href="index.html"><img src="msb1.gif"></a>
15:   </body>
16:   </html>
```

XML言語の特徴

- タグの自由な定義(日本語、データの意味)
- 簡素で厳密な言語仕様、終了タグの必須化
- 内容情報(XML文書)と表示属性(スタイル)の分離
- 業界での共通タグの定義(DTD)
- 人間には理解し易く、コンピュータには扱い易い文書。

XML文書とXMLスタイルシート

```
1: <?xml version="1.0" encoding="Shift-JIS"?>
2: <?xml-stylesheet type="text/xsl" href="address.xsl"?>
3: <アドレス帳>
4:   <個人情報><氏名>東京太郎</氏名><住所>東京都新宿区</住所><郵便番号>160-1234</郵便番号><電話>03-1234-5678</電話></個人情報>
5:   <個人情報><氏名>大阪花子</氏名><住所>大阪府大阪市</住所><郵便番号>535-4321</郵便番号><電話>06-8765-4321</電話></個人情報>
6:   <個人情報><氏名>青森鈴子</氏名><住所>青森県青森市</住所><電話>017-765-4321</電話><郵便番号>038-7654</郵便番号></個人情報>
7: </アドレス帳>
```

(a) XML文書(内容情報)

```
11:<?xml version="1.0" encoding="Shift-JIS"?>
12:    <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl" xml:lang="ja">
13:    <xsl:template match="/">
14         <html lang="ja">
15:            <head><title>address</title></head>
16:            <body>
17:               <h2 style="margin-left: 100pt">アドレス帳</h2>
18:               <table border="3">
19:                  <tr style="background-color: yellow"><th>氏名</th><th>住所</th><th>郵便番号</th><th>電話</th></tr>
20:                  <xsl:for-each select="アドレス帳/個人情報">
21:                     <tr><td><xsl:value-of select="氏名" /></td><td><xsl:value-of select="住所" />
22:                     </td><td><xsl:value-of select="郵便番号" /></td> <td><xsl:value-of select="電話" /></td></tr>
23:                  </xsl:for-each>
24:               </table>
25:            </body>
26:         </html>
27:    </xsl:template>
28:    </xsl:stylesheet>
```

(b) XSLスタイルシート

eビジネスとXML

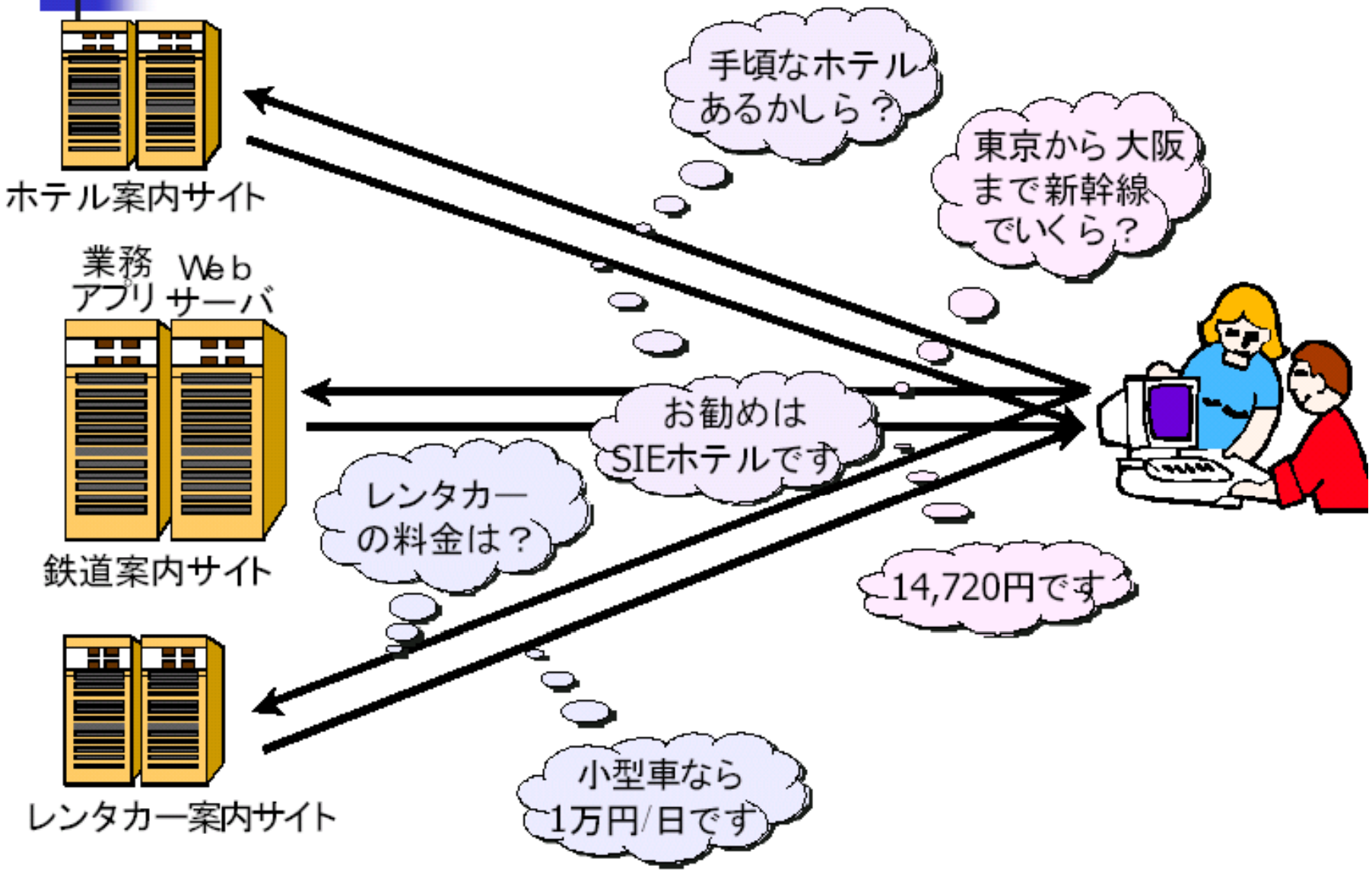
- IBMが蘇った根本的ビジネスモデル
- これまで、ばらばらに 個別データ処理していたものを、一つの窓口(これをポータルという) から処理可能に統合化する。バックエンドには、たくさんの個別システムが存在する。 データのパイプライン処理、並列処理、統合化処理を行う。
- 必要なもの ; データと手続きの統合化
  - XML言語という共通メタ言語の利用
- デジタル＝情報の水平展開
  - まさに、旅行代理店の仕事。。。。。
  - ってことで、じきに、旅行代理店産業は消える運命。。。

例7：旅行(＝eビジネスの典型)

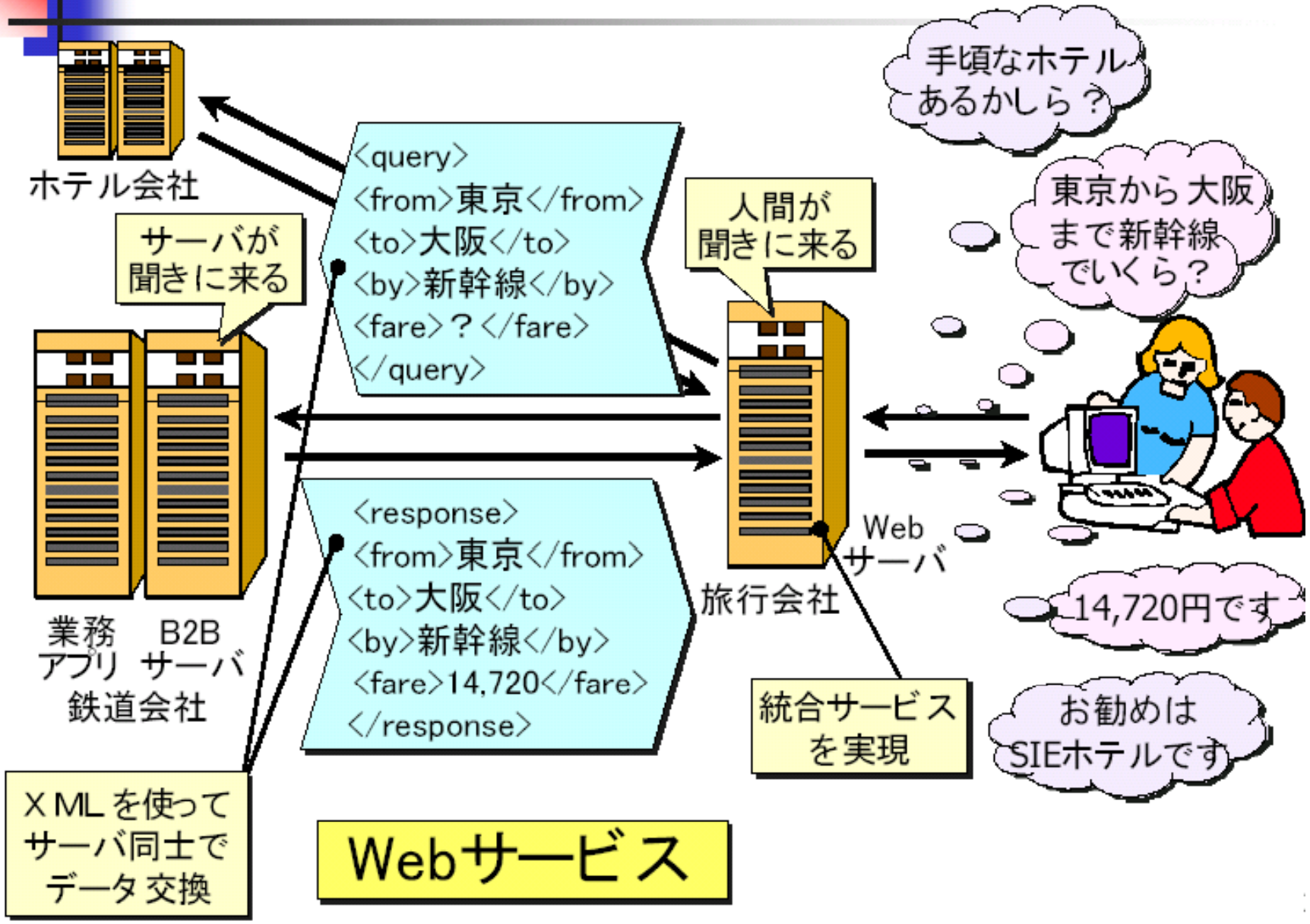
- 移動手段
  - 航空券：既に フルデジタル
  - レンタカー：ほぼ、デジタル化完了
  - 電車：最も遅れている業界
  - 宿泊：Webですべてが完了

  (*) ここまでは、既に ポータルが存在。
- 食事：Web検索が普通
- 嗜好：Web検索で対応可能(含 予約)
- お金：キャッシュレス化(デジタル化)

従来のWebサイト へのアクセス



Webサービスの概念


手頃なホテル
あるかしら？
ホテル会社
<query>
<from>東京</from>
<to>大阪</to>
<by>新幹線</by>
<fare>？</fare>
</query>
人間が
聞きに来る
東京から大阪
まで新幹線
でいくら？
サーバが
聞きに来る
<response>
<from>東京</from>
<to>大阪</to>
<by>新幹線</by>
<fare>14,720</fare>
</response>
Web
サーバ
旅行会社
業務
アプリ
B2B
サーバ
鉄道会社
14,720円です
統合サービス
を実現
お勧めは
SIEホテルです
XMLを使って
サーバ同士で
データ交換
Webサービス

例7：旅行(＝eビジネスの典型)

- 移動手段
  - 航空券：既に フルデジタル
  - レンタカー：ほぼ、デジタル化完了
  - 電車：最も遅れている業界
  - 宿泊：Webですべてが完了

  (*) ここまでは、既に ポータルが存在。
- 食事：Web検索が普通
- 嗜好：Web検索で対応可能(含 予約)
- お金：キャッシュレス化(デジタル化)

大規模（WEB）サーバ

ＷＥＢシステムの歴史と
技術チャレンジ

WWWサーバ：基本中の基本

- レスポンスが遅いとどうなる？
  - クリックを繰り返す (10秒くらが限度)

    ➔Positive Feedback

    たくさんのTCPコネクションがForkされる。。。

    計算機資源がcatastrophy的に使用される。。。。

    要は、DDOSと同じ状況

  - 2度とアクセスしない……

  - ビジネスチャンスの喪失

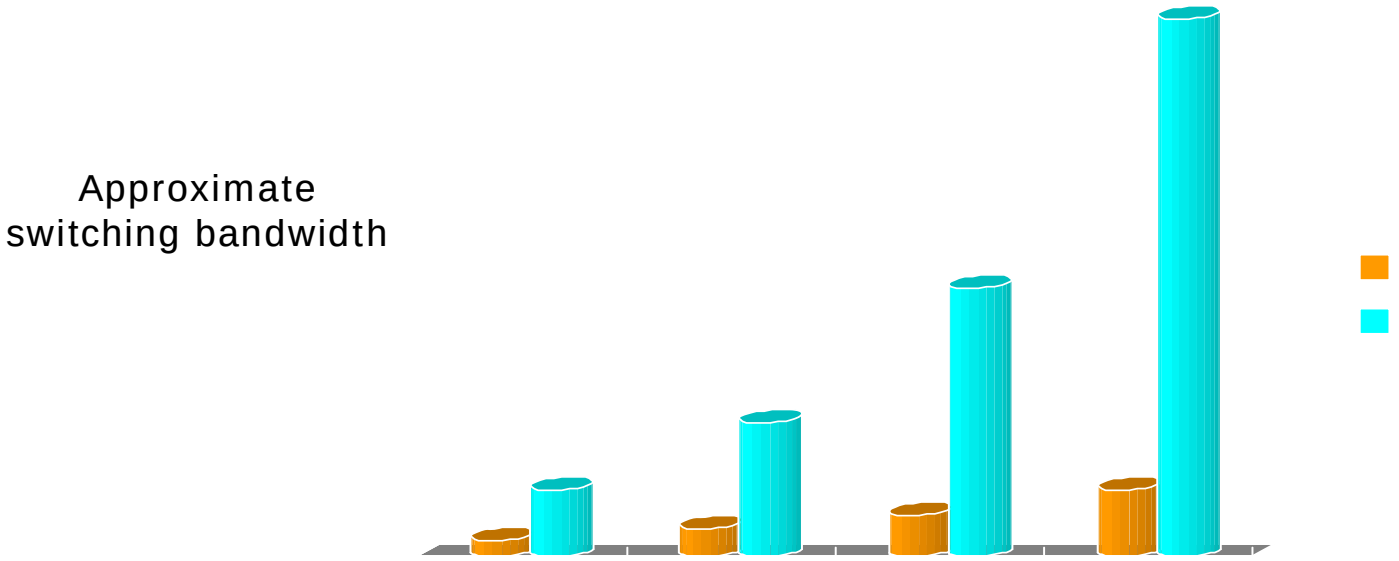
Optical Networking at Double Moore's Law

- Moore's Law says that computer speed=2x every 18 months, and the cost = 50%
- John Roth, president and chief executive officer, says that Nortel Networks is moving at twice the speed of Moore's Law, doubling the capacity of its fiber-optic systems and halving the cost *every nine months*.
- Networks: 3 years=16x capacity, 6% cost
  - Computers: 3 years=4x speed, 25% cost
- Networks: 6 years=256x capacity, >1/2% cost
  - Computers: 6 years=16x speed, 6% cost

Source: HPCwire hpcwire@tgc.com>

Scaling problem as we know it

- ネットワーク帯域は１年で２倍に
  - AT&T Research [Coffman, Odlyzko 2000]
- コンピュータの性能は１８ヶ月で２倍に
  - Moore’s Law



WWWサーバ: 基本中の基本動作

- GET methodのみを処理する簡単なWebサーバ
  - コネクションを受け付ける
  - リクエストラインを入力し、URI(パス名)を取り出す
  - 適当なステータスラインとヘッダを出力する
  - URIで指定されたオブジェクトを出力する
  - 掃除をしてから最初に戻る
- レスポンスが遅いとどうなる？
  - クリックを繰り返す (10秒くらが限度)
  - 2度とアクセスしない

実用サーバのプロセス構造

- 基本的な構造
  - シングルプロセス
  - コネクションごとにプロセスをforkする
  - あらかじめ複数のプロセスをforkしておく
  - 上記+足りなくなったら新しいプロセスをforkする
- お得な拡張
  - マルチスレッドプロセス
  - 入出力マルチプレキシング・非同期入出力
  - 出力ヘルパー (e.g.: Squid in http-accelerator mode)

実用サーバの機能

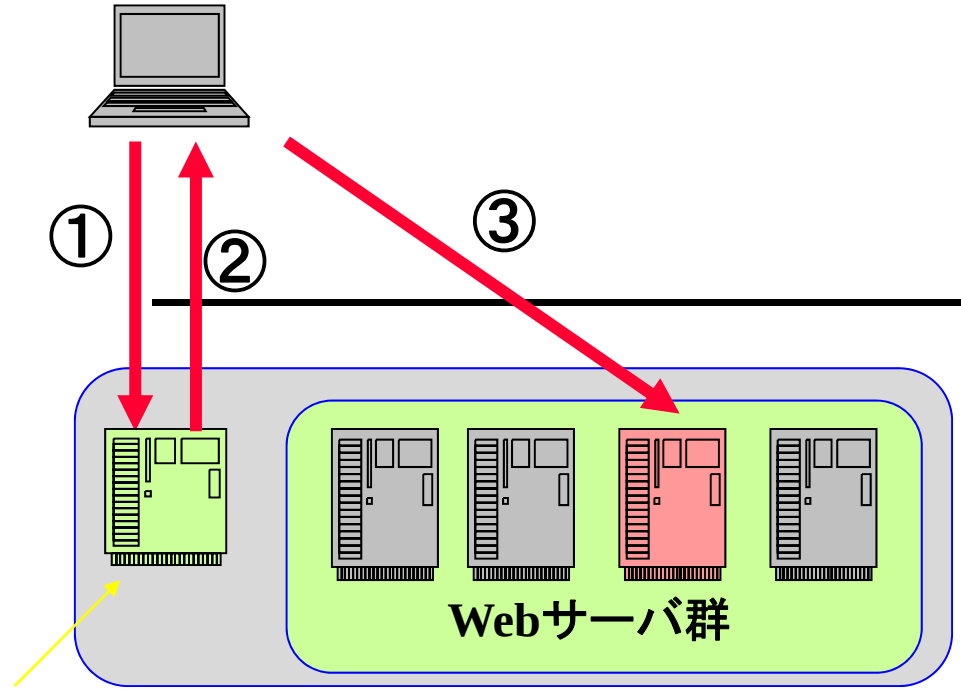
- URLリダイレクション (Remote Redirection)
  - 他のサーバへ振る
- URI 書き換え (Aliasing)
  - 自身の中で振る
- アクセス制御
  - ACL (ホスト/ネットワーク×URI)
  - User Authentication
- Virtual Hosting (Virtual Server)

大規模サーバファームの構築

- Webサイトに押し寄せる大量のトラフィック
  - DNSによる負荷分散
  - TCPコネクションディスパッチ
  - レイヤ7スイッチ
- 信頼性の向上
- サービスの継続性
  - ドキュメントアップロード方法
- 対攻撃性の確保
  - DoS(Deny of Service)への対策

基本的な手法

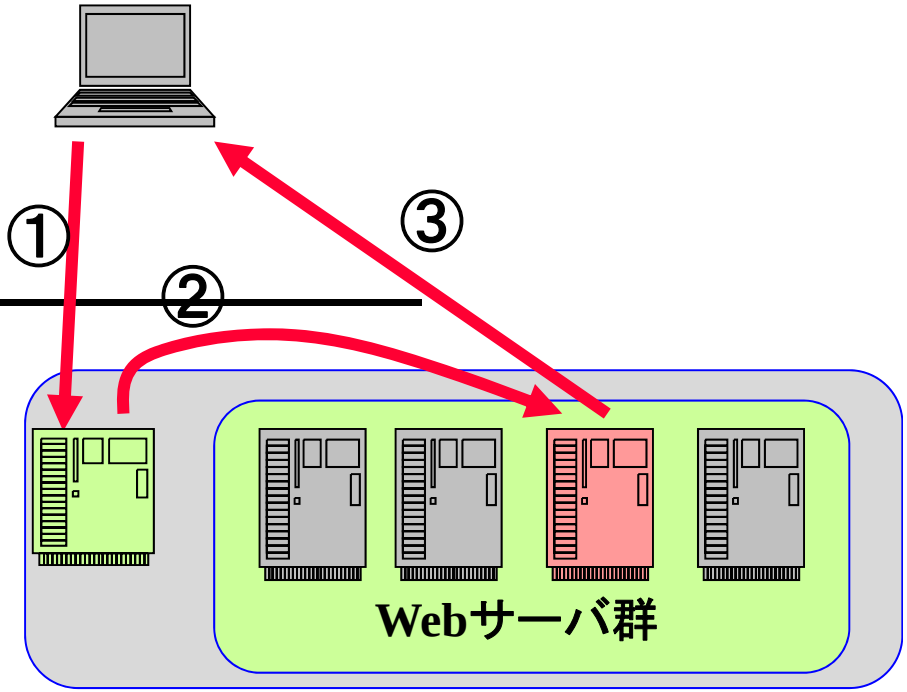
- 水平分散 (Redirection)
- 垂直分散 (キャッシュ)



リダイレクション(Redirection)サーバ

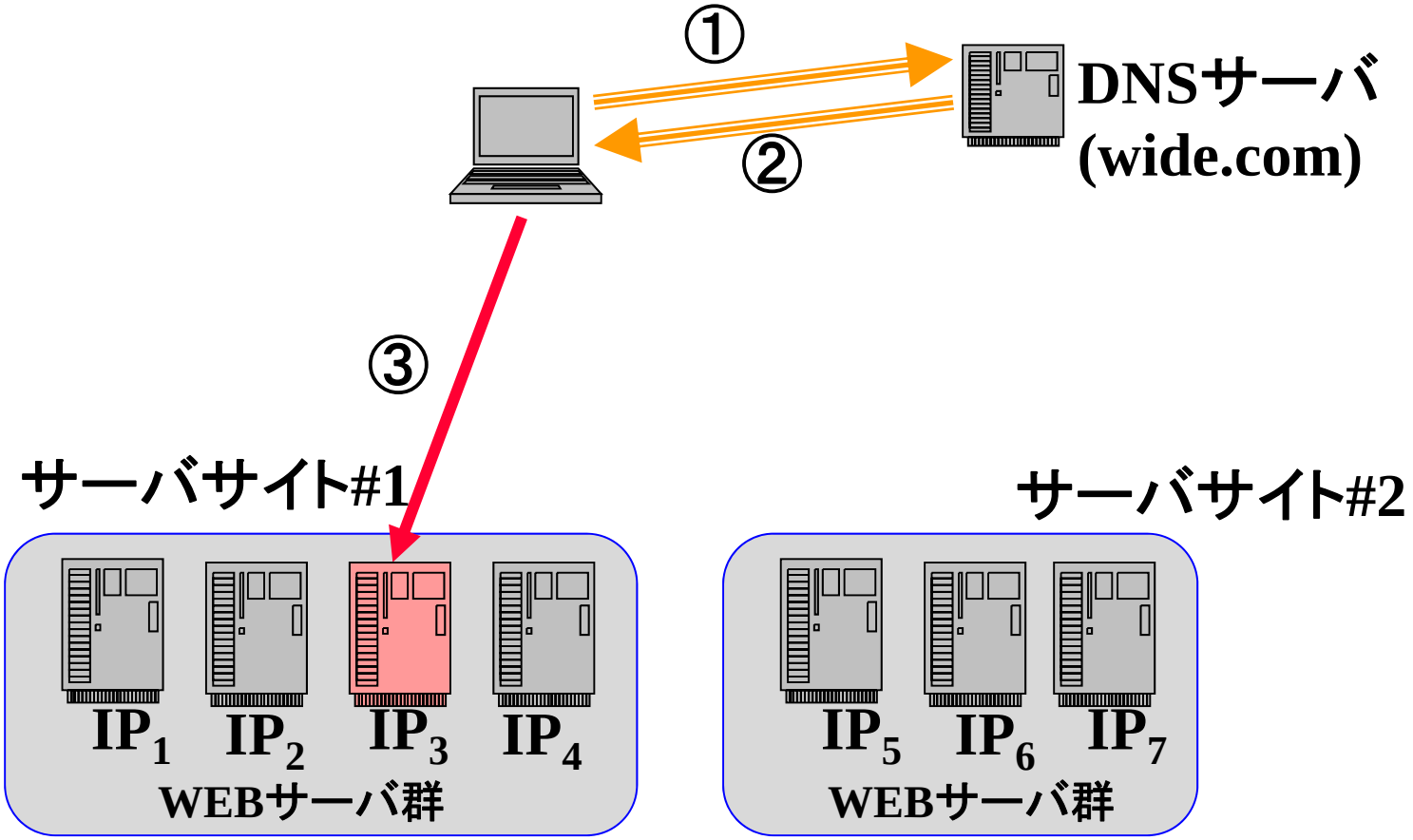
① アクセス要求
② URL Redirect (リダイレクト)命令
③ Redirected (リダイレクト)アクセス要求

図8-7 URLリダイレクション



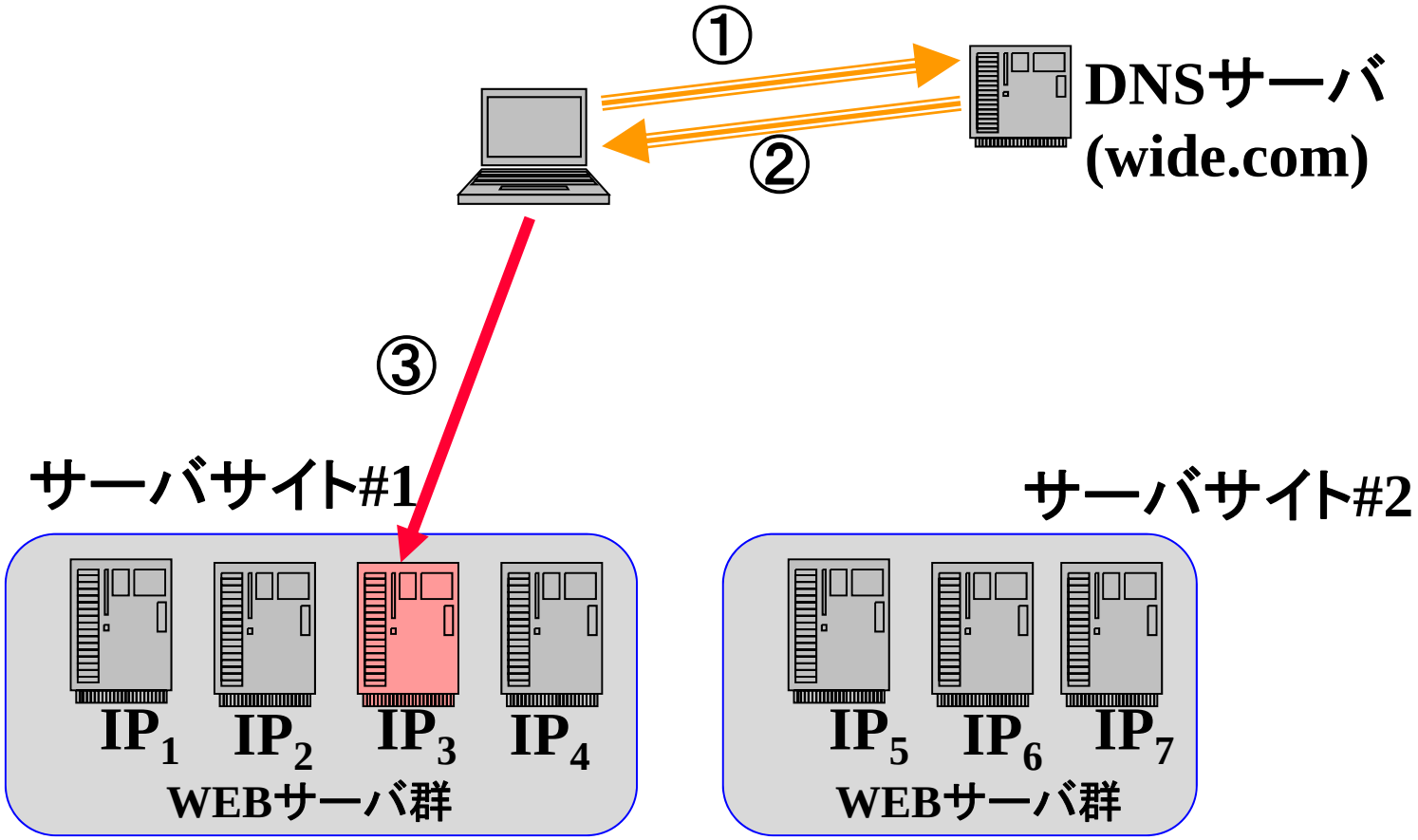
① アクセス要求
② Aliasing(書き換え)命令
③ Aliased(書き換え) URLアクセス応答

図8-8 URL書き換え(Aliasing)



① DNS query (www.wide.com)
② DNS reply メッセージの内容 ;
$\{IP_1, IP_2, IP_3, IP_4, IP_5, IP_6, IP_7\}$
③ IP_3 をwww.wide.com のサーバとして選択

図8-9 DNSによる負荷分散



① DNS query (www.wide.com)
② DNS reply メッセージの内容 ;
　　{IP_1, IP_2, IP_3, IP_4, IP_5, IP_6, IP_7}
③ IP_3 をwww.wide.com のサーバとして選択

図8-9 DNSによる負荷分散

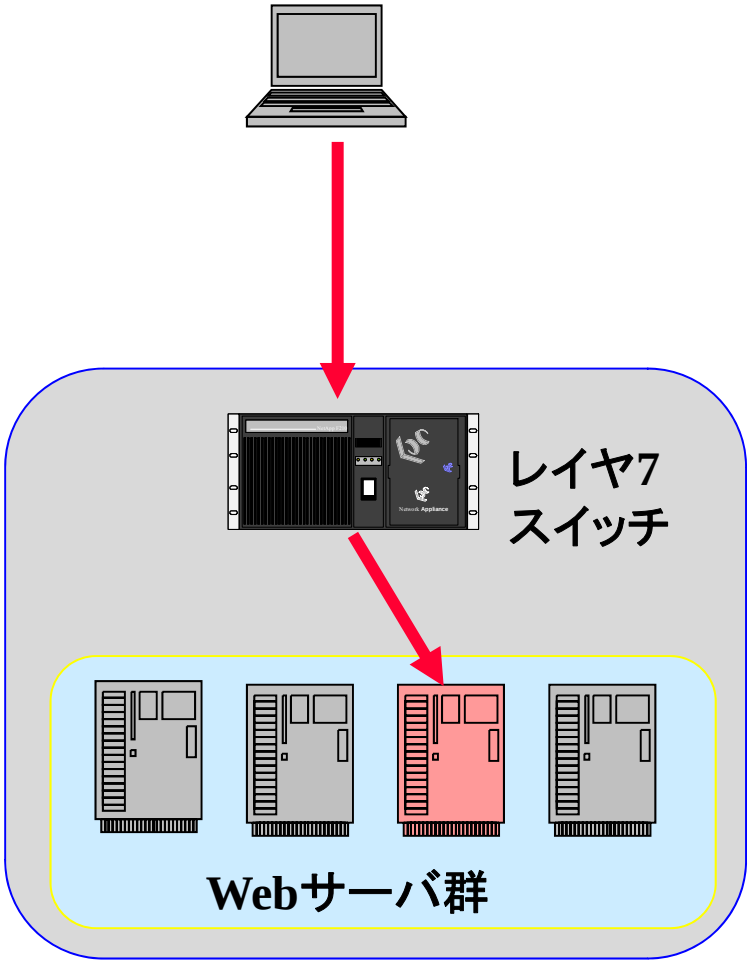


図8-10 レイヤ7スイッチによる負荷分散

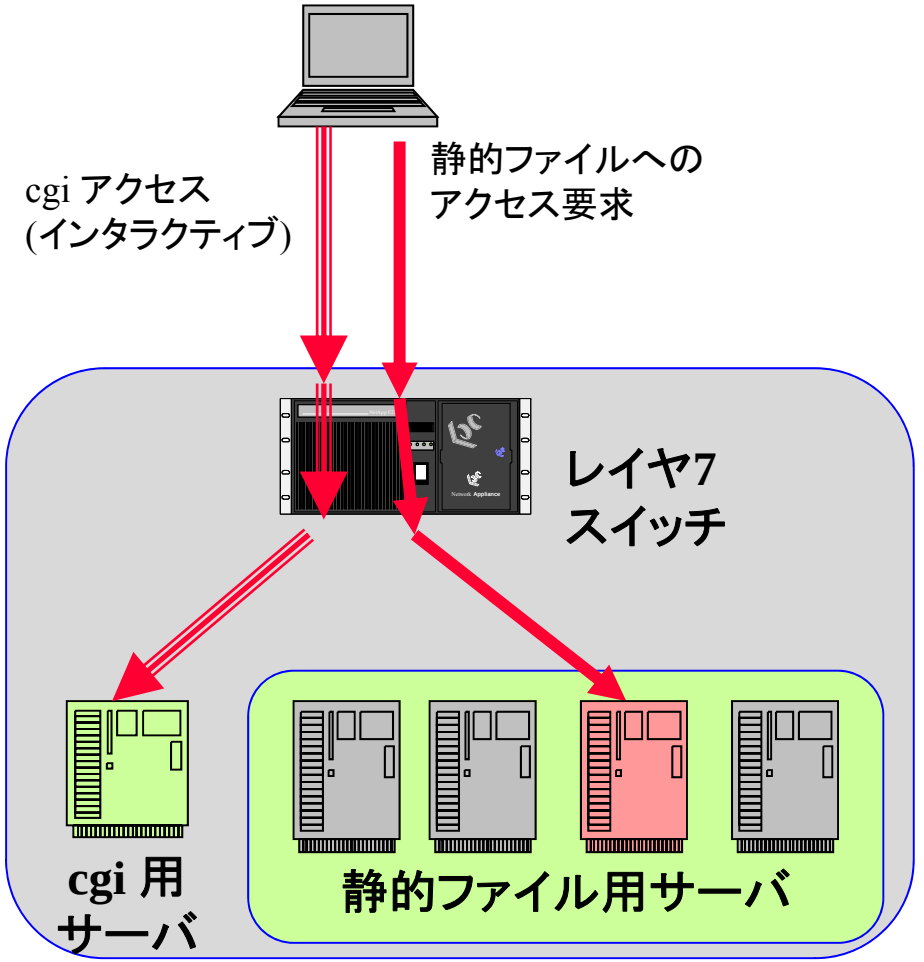


図8-11処理機能によるアクセスの誘導

DNS負荷分散の問題点

- システムダウンが外部に見えてしまう
- システム構成を短時間で変更することが不可能

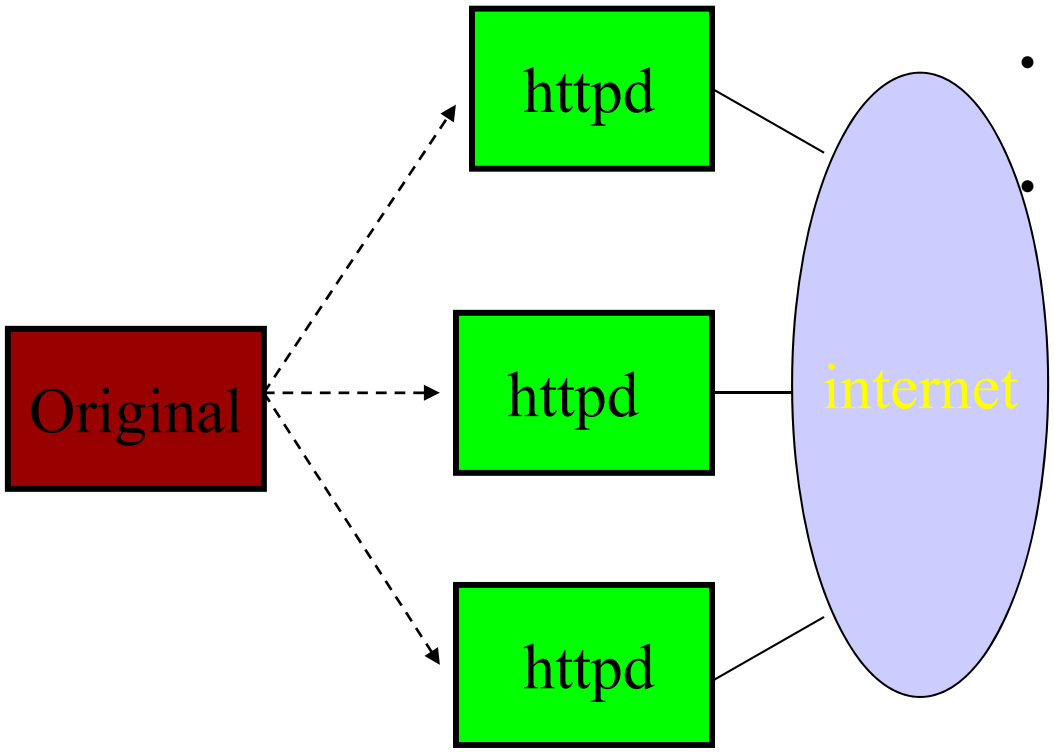
TCPコネクションディスパッチング (3)

- 可能性
  - ローカルな負荷分散
    - 同じ処理能力を持つシステムは、大規模な単一システムより小規模な複数のシステムで構成したほうが安価
    - スケーラビリティ
  - TCPのコネクション能力の上限を超えられる可能性
  - 他のテクノロジとの組み合わせによる可能性
    - サーバリダイレクション+ コネクションディスパッチング

地理分散：大規模サイトのもう一つの形態

- サービスの分散配置
  - 地理的に均一なサービスを提供する
- クライアントから見て最も適切なサーバを選択する
  - ネットワーク距離（メトリック）
  - 往復時間
  - アクセスポリシー
- 問題
  - サーバの位置決め（ロケータ）
  - サーバの選択
  - 自動化：インターネット的に…

大規模サイト構築へのアプローチ



- DNS RoundRobin
- ミラーサーバ
- インターネット上に複数のサーバを配置
- コンテンツの同期問題

トレンド

- Webサービスにおいて常に小さなネットワークレーテンシを確保するためには、「近く」のサーバへアクセスすればよい
  - ユーザにサーバを選択させる方法は、あまりにも透過性に欠け、あまりにもインターネット的でない
  - HTTPリダイレクションの利用
    - アドレスからドメイン名への逆引きと位置の推測
    - ネットワーク (IP)層が持っている情報の利用
    - 実測値の計測と利用
- 透過的な負荷分散
- サーバのサービス能力の差を隠蔽

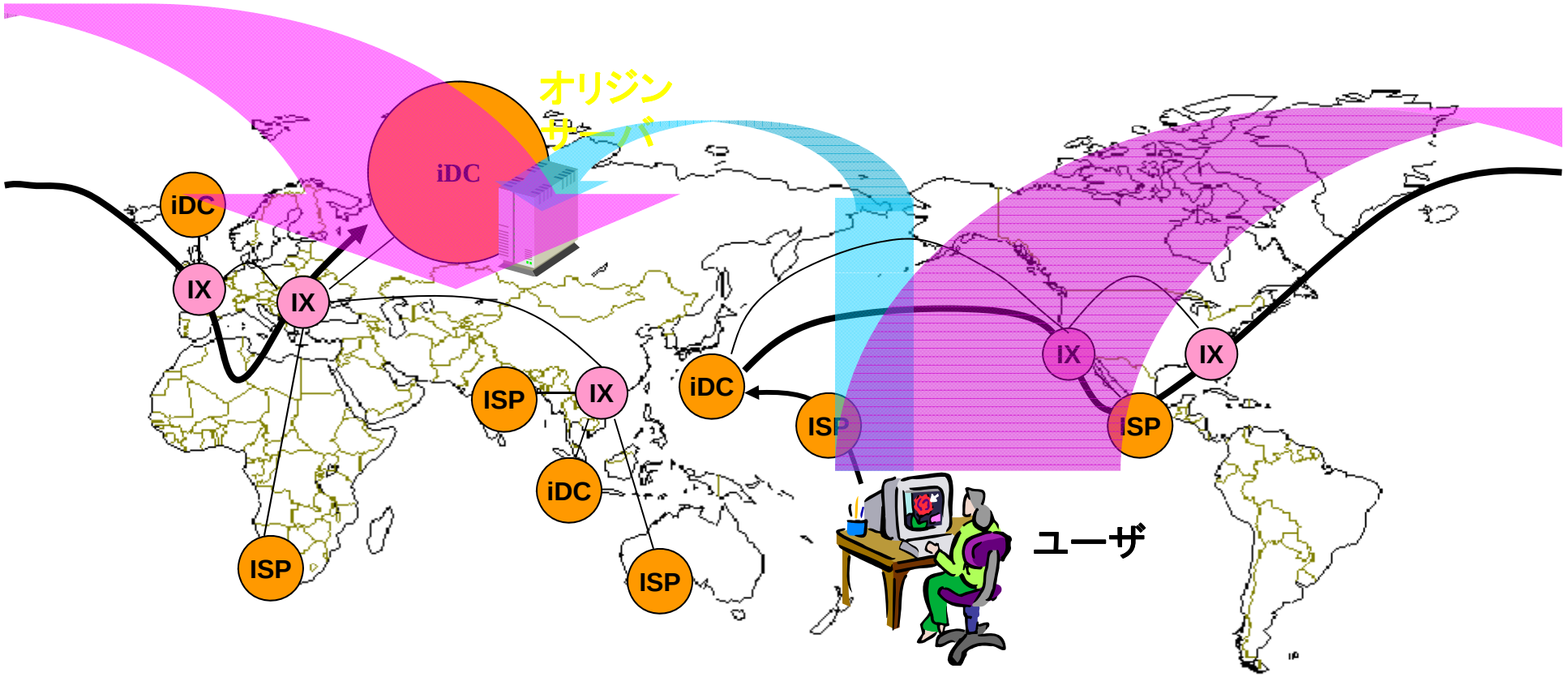
ＣＤＮの概要

- Contents Delivery Network –
- Contents Distribution Network -

CDS/CDNとは？

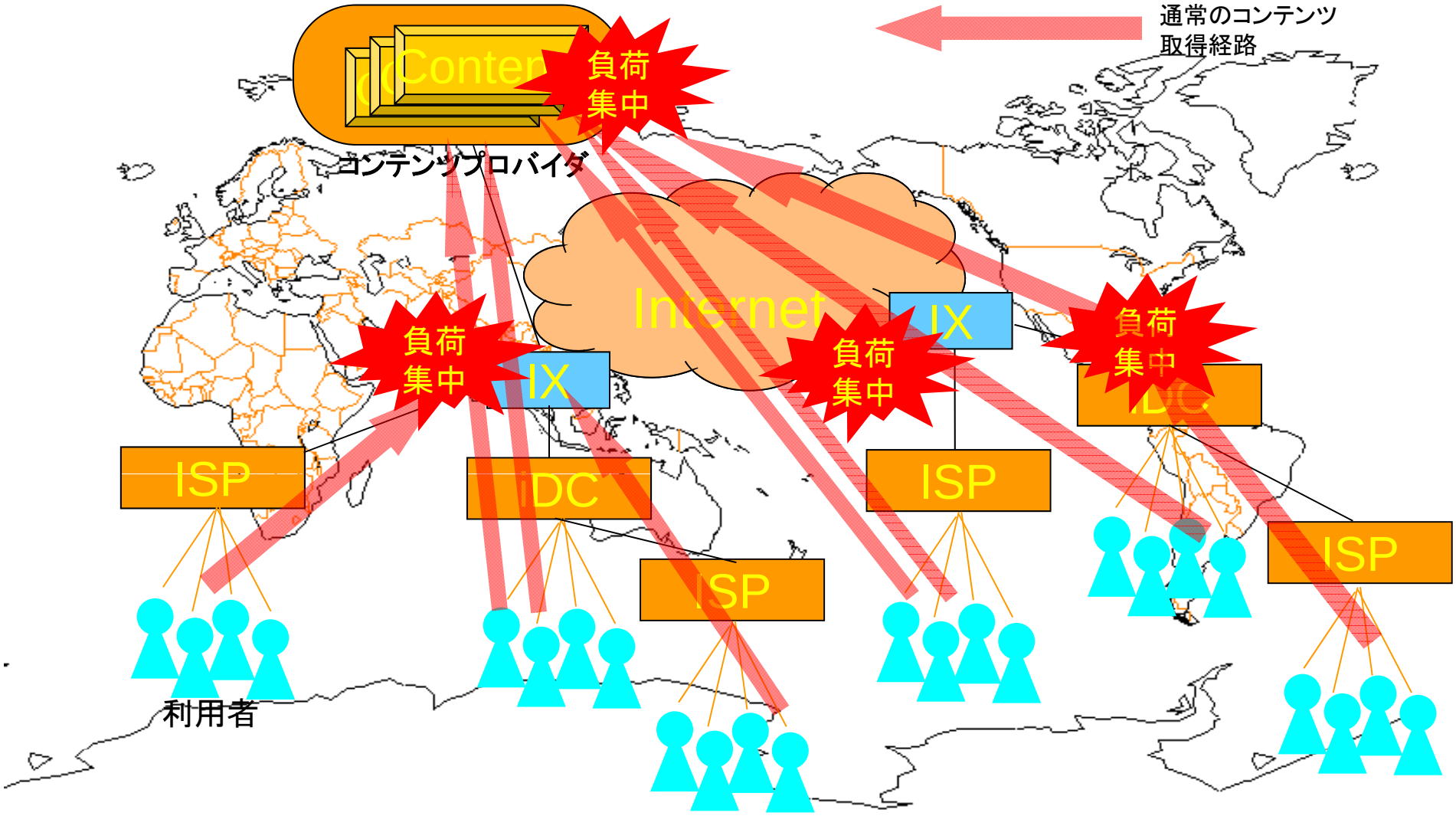
- CDS
  - Content(s) Delivery Service/System
  - Content(s) Distribution Service/System
  - Web/Streamingなどのrich contentsに対して
    キャッシュやミラーを積極的に用いた負荷分散システム(サービス)
    - リバースキャッシュ技術
    - キャッシュ管理技術
    - リクエストナビゲーション技術
- CDN
  - Content(s) Distribution Network
    - CDSの基盤ネットワーク
  - CDS間のコンテンツピアリングを行う単位
    - 例：　A社のαというコンテンツをB社のCDSへ提供する

Internetの構造上の問題の回避



- 遅延の問題
- ISPの複雑な経路制御ポリシー

Webサービスの構造上の問題（負荷の集中）



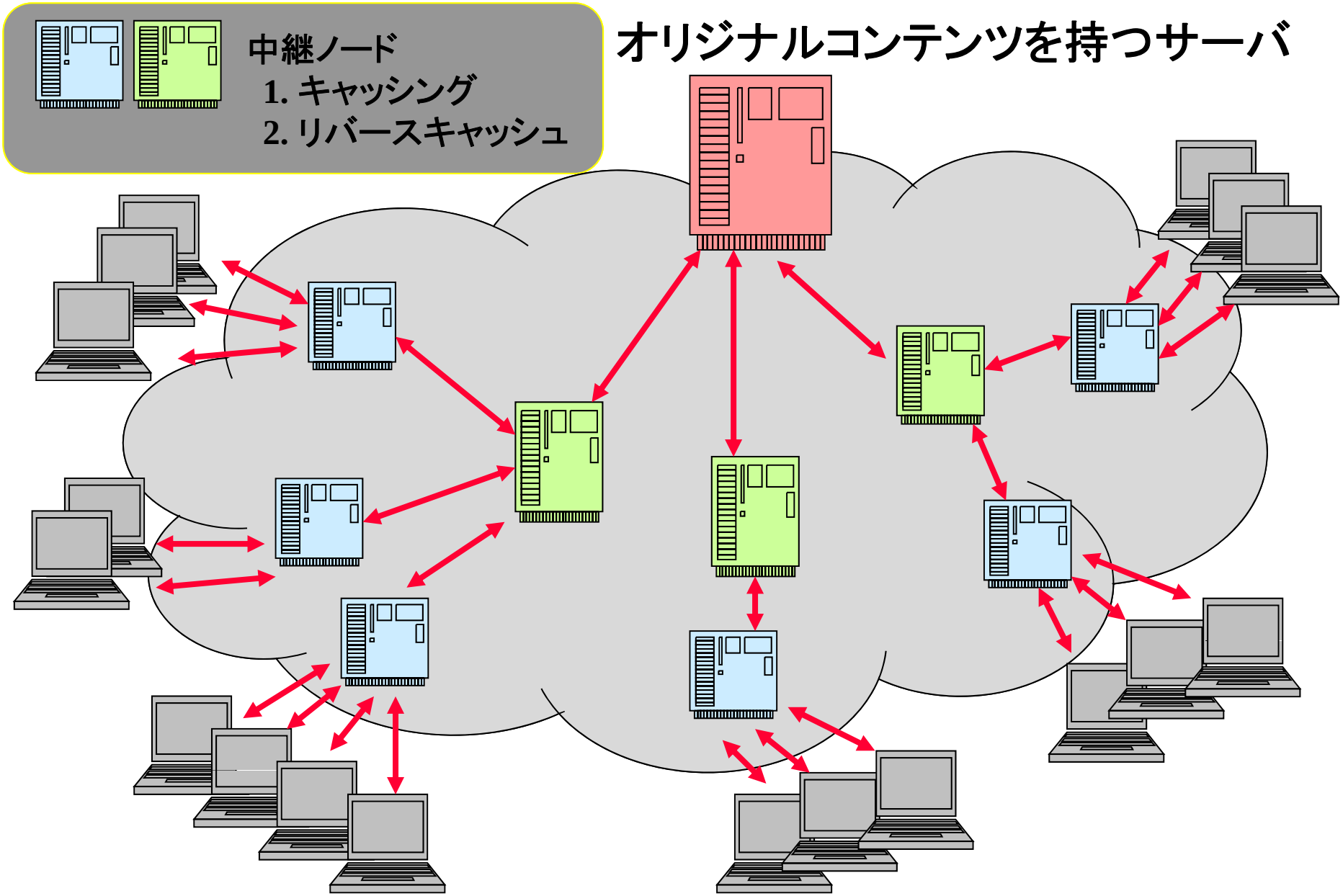
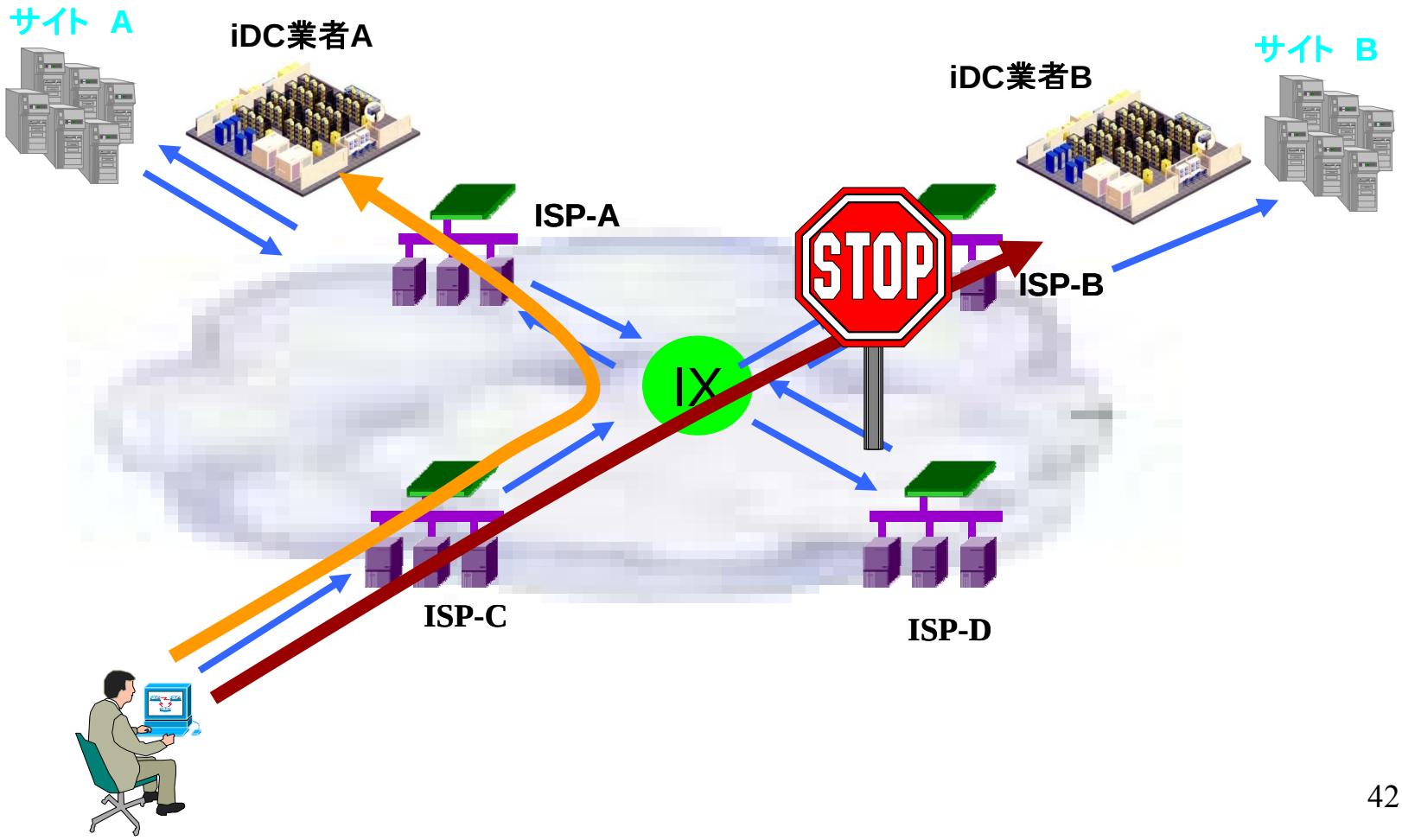
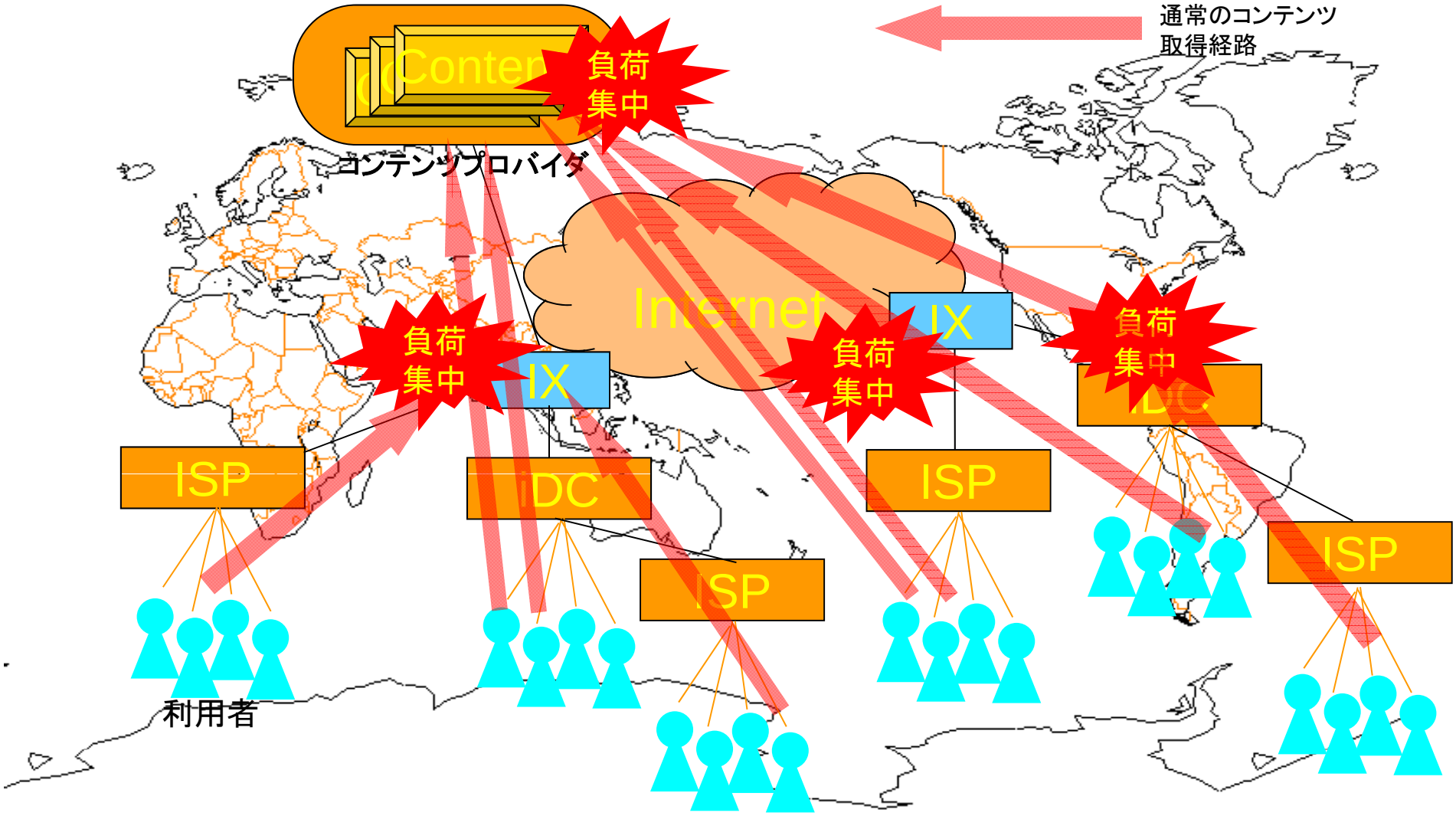


図8-12 CDNの全体構造概念図

iDC、ISP間のフィルタリングに左右されない



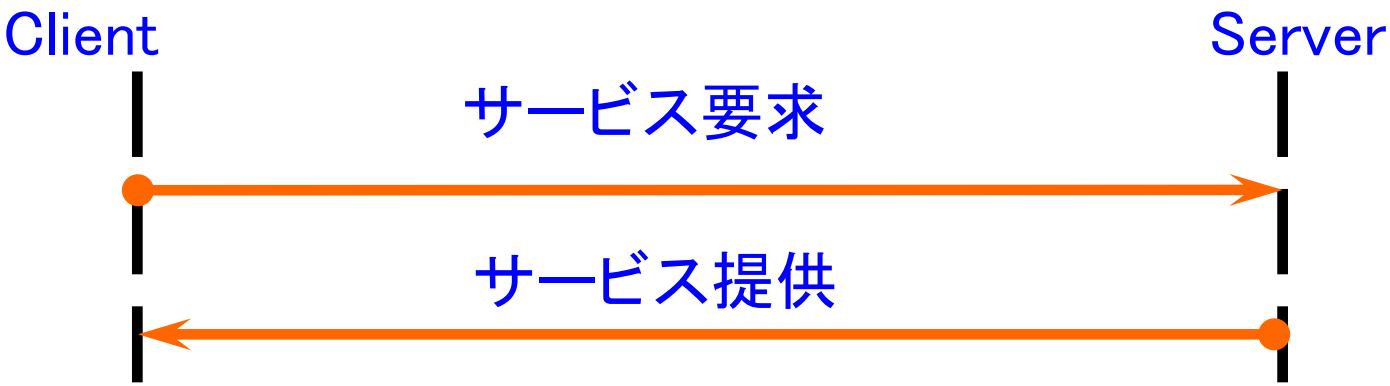
Webサービスの構造上の問題（負荷の集中）



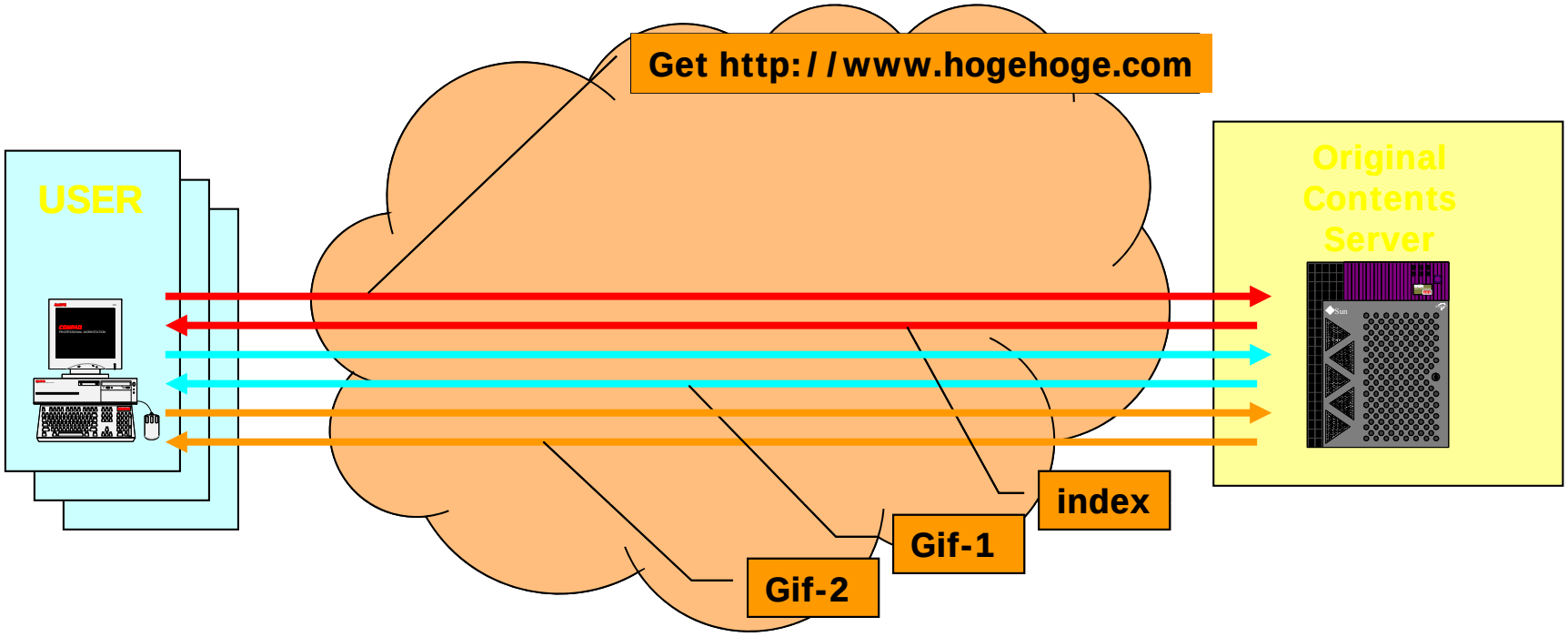
CDN/CDS基本動作

client-server と peer-peer(1)

- クライアント
  - 能動的にサービス提供を促す側
- サーバ
  - 受動的にサービス提供する側

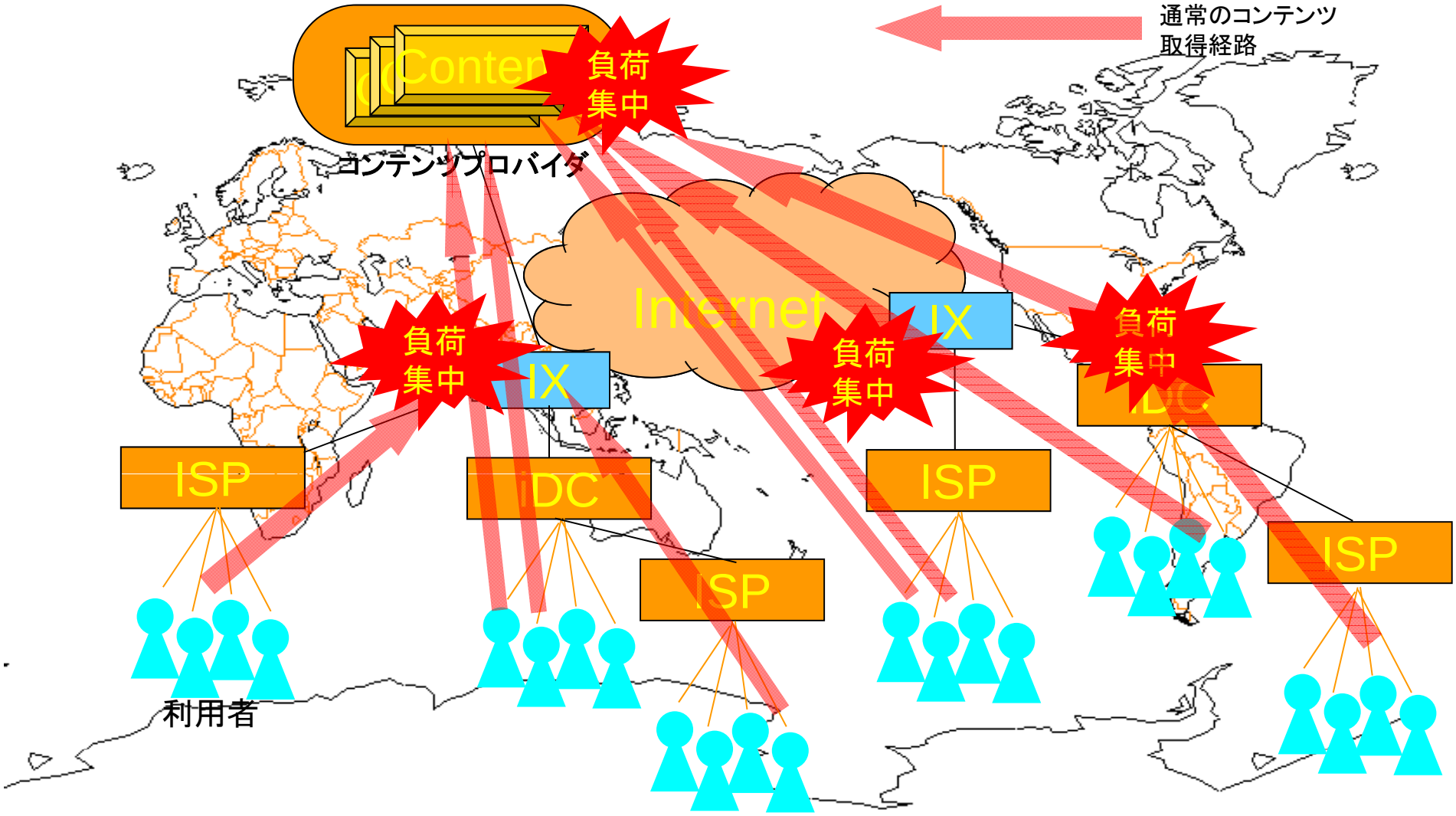


通常のWEBコンテンツの流れ



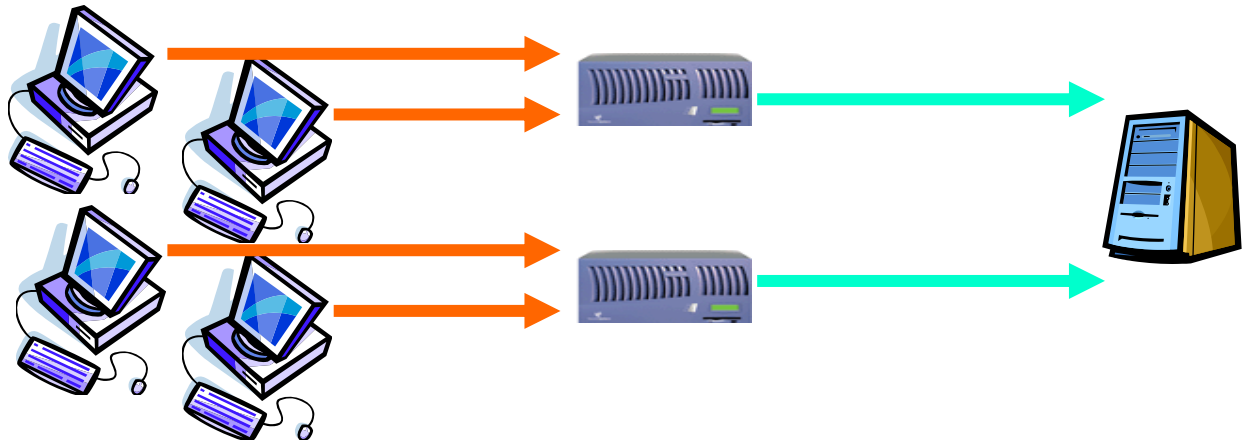
一度のhttpリクエストに際し、ファイル形状ごとにそれぞれTCPセッションの確立から始めなければならないため、アクセス数の増加に対しサーバの負荷が指数関数的に増加する

Webサービスの構造上の問題（負荷の集中）

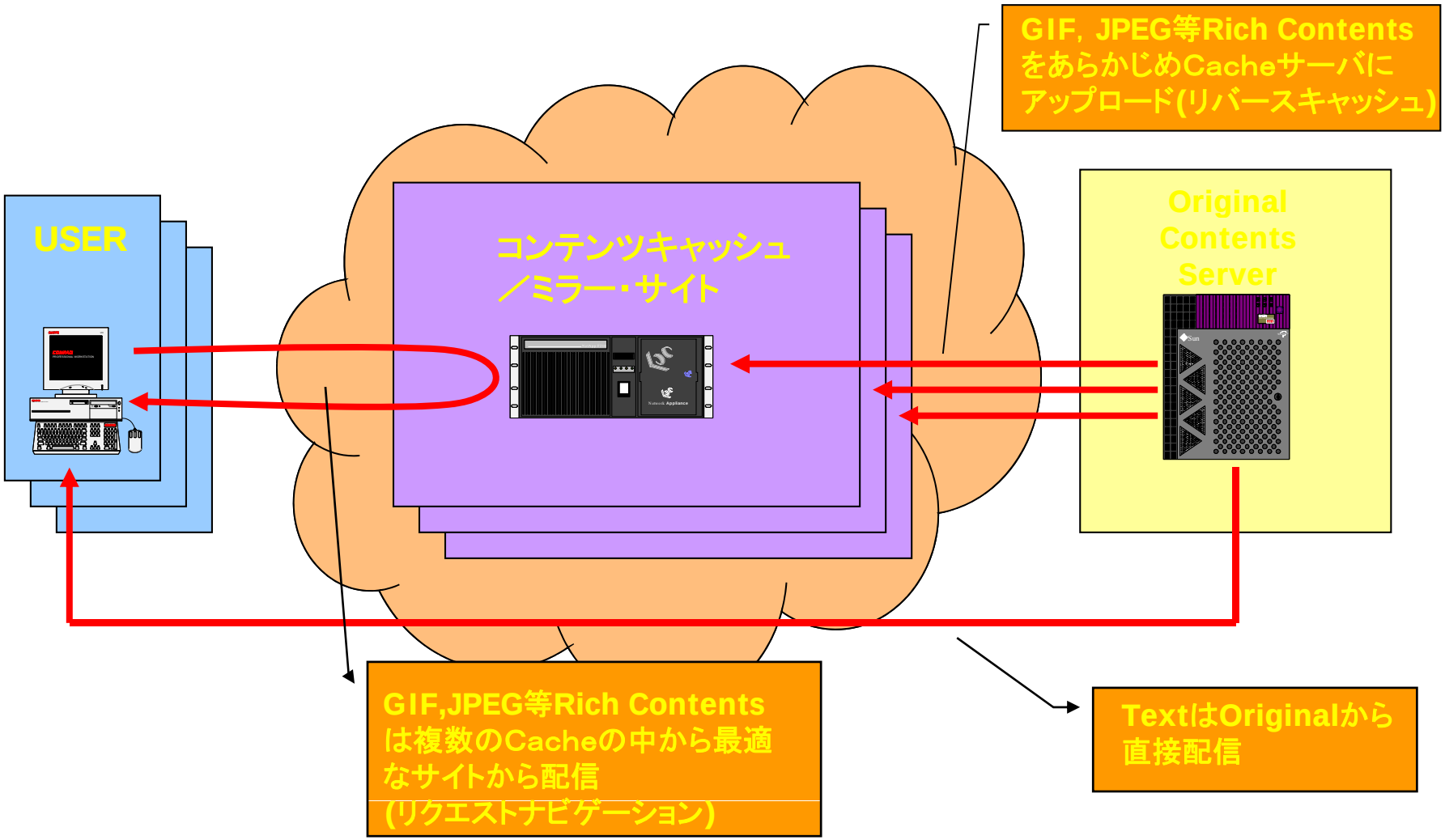


CDN as scaling mechanism

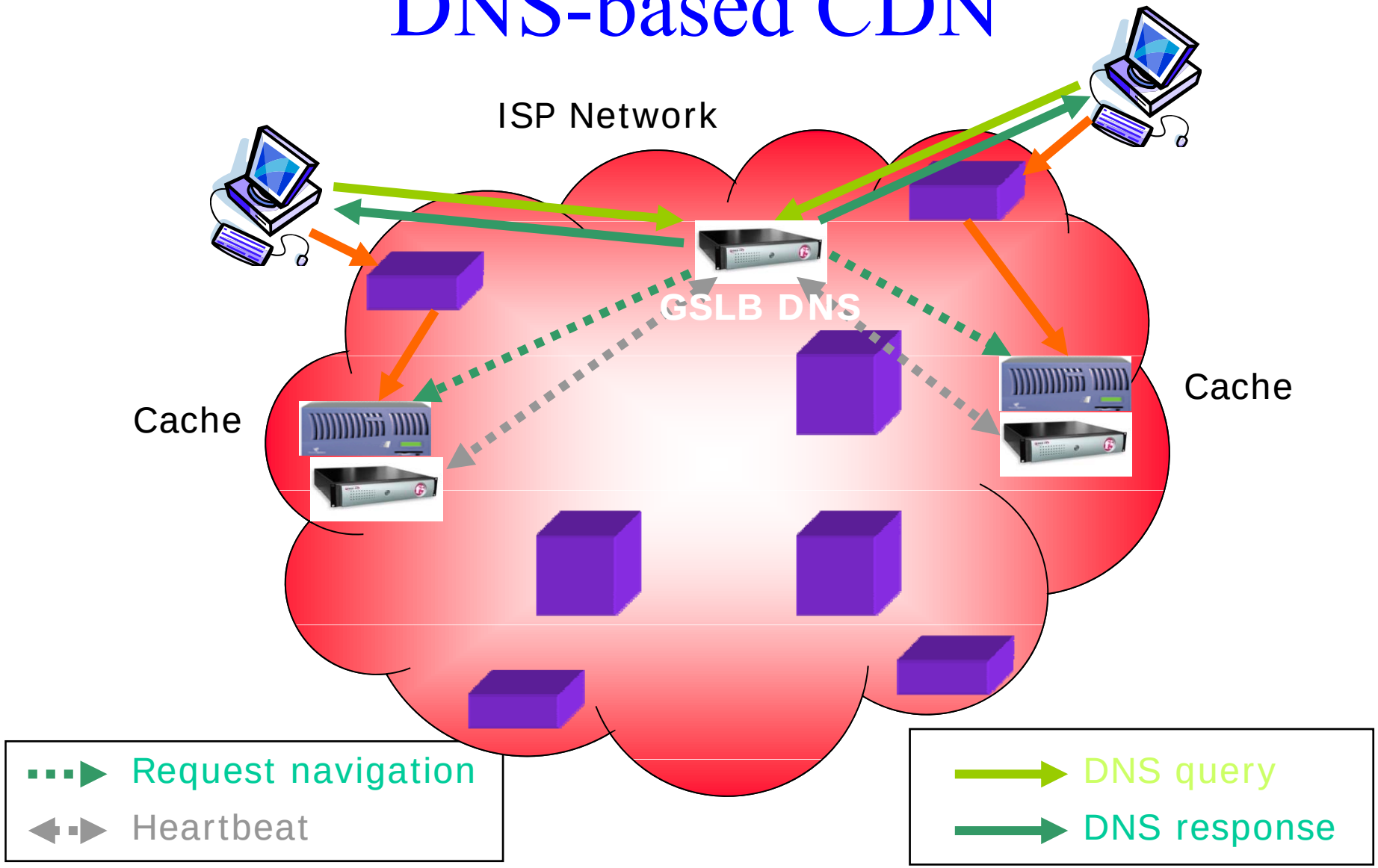
- Mooreの法則とCoffmanの観測のギャップを埋める
  - Reverse proxy
  - Mirroring
- また、end-to-end delayを改善
  - End-to-edge へ



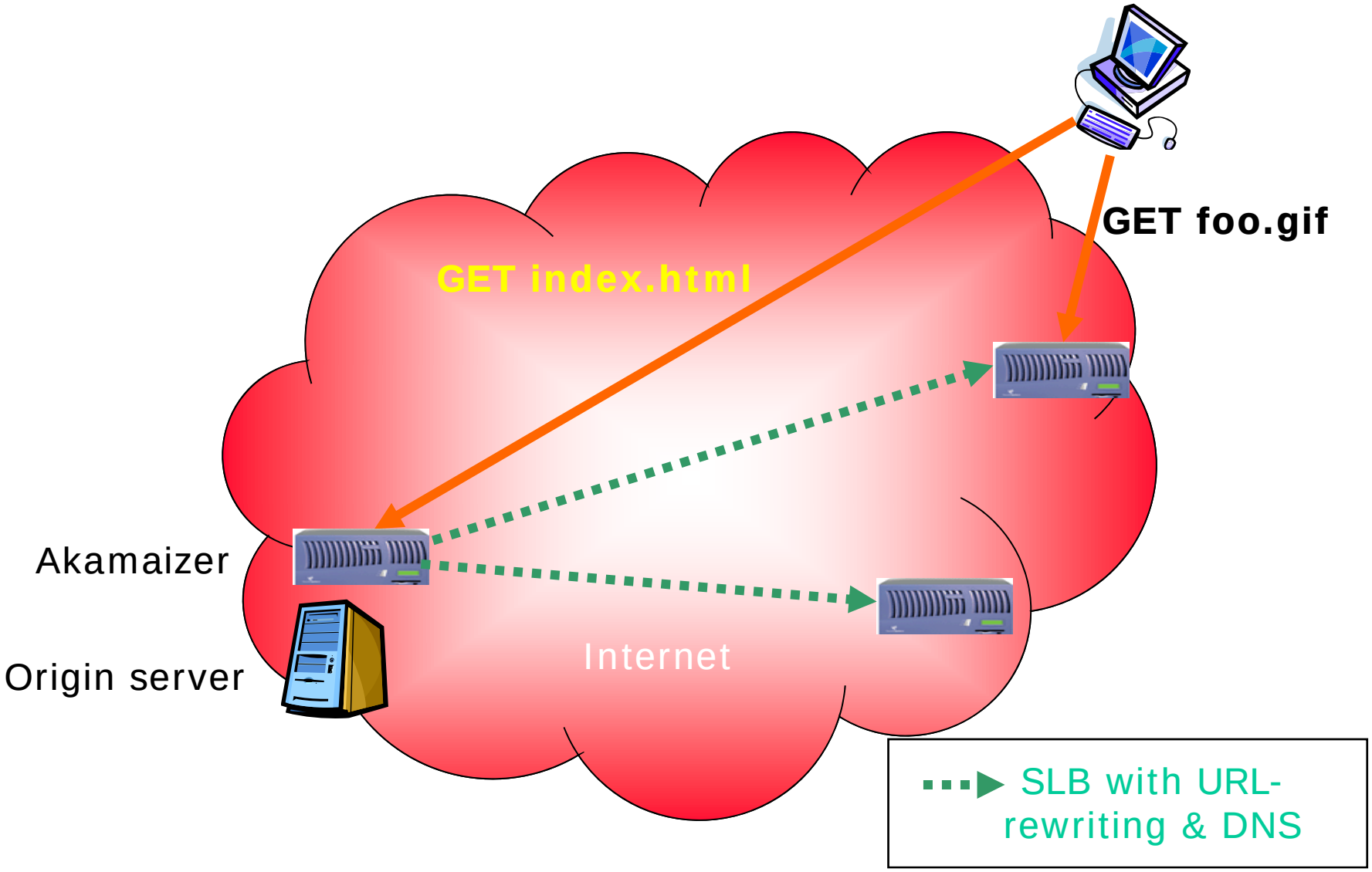
CDS(キャッシュ同期技術)



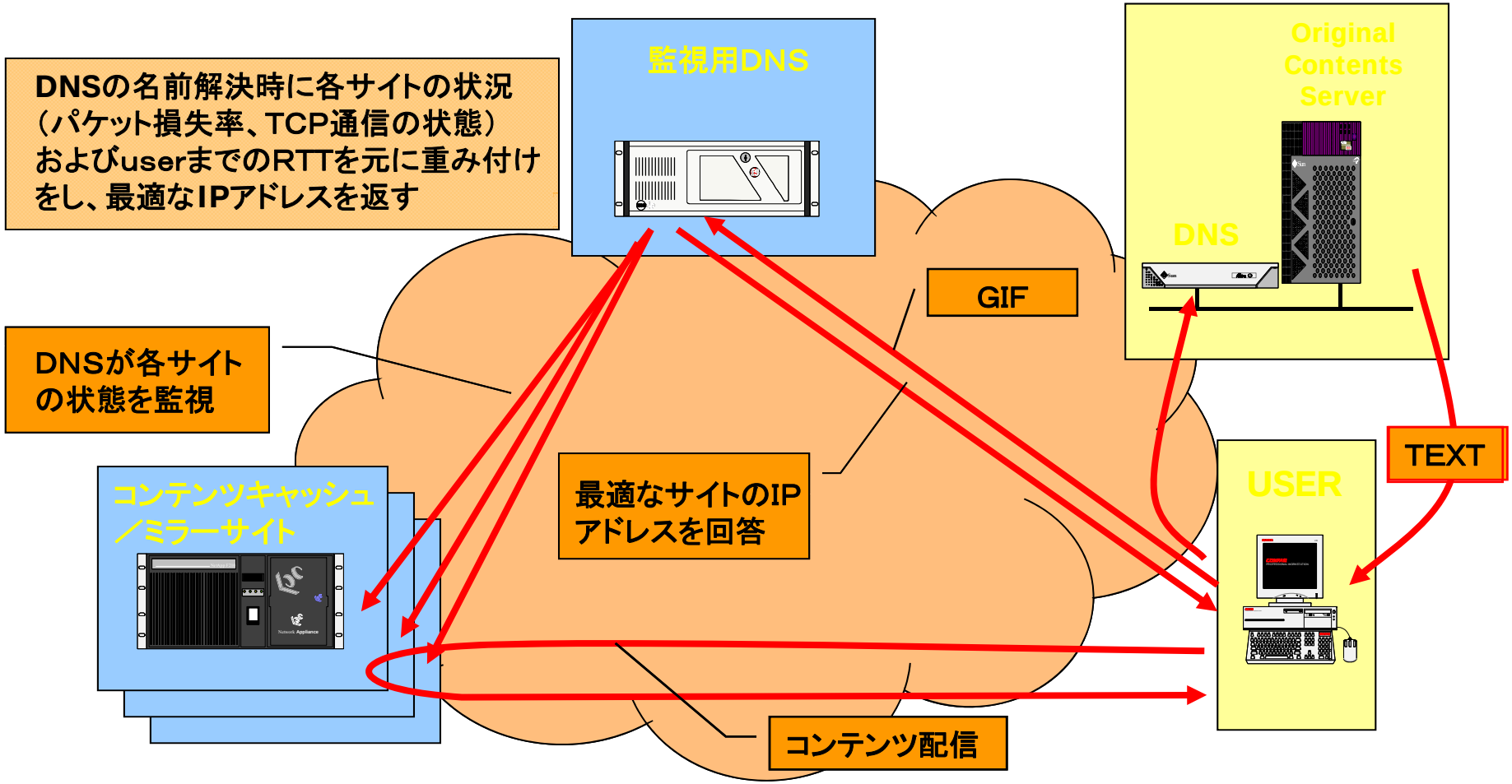
DNS-based CDN



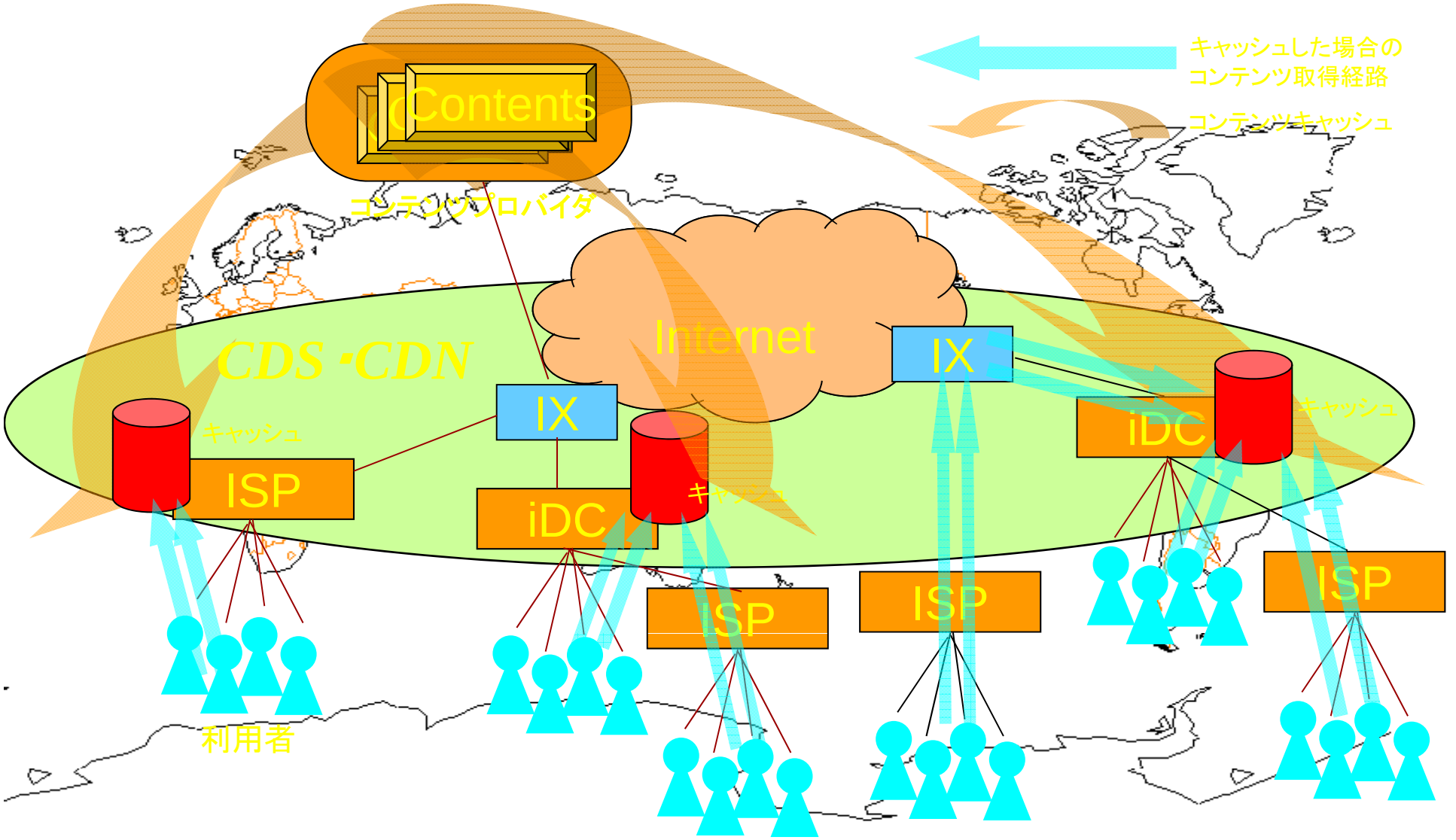
Reverse proxy + URL rewriting



配信元決定方法
（リクエストナビゲーション）



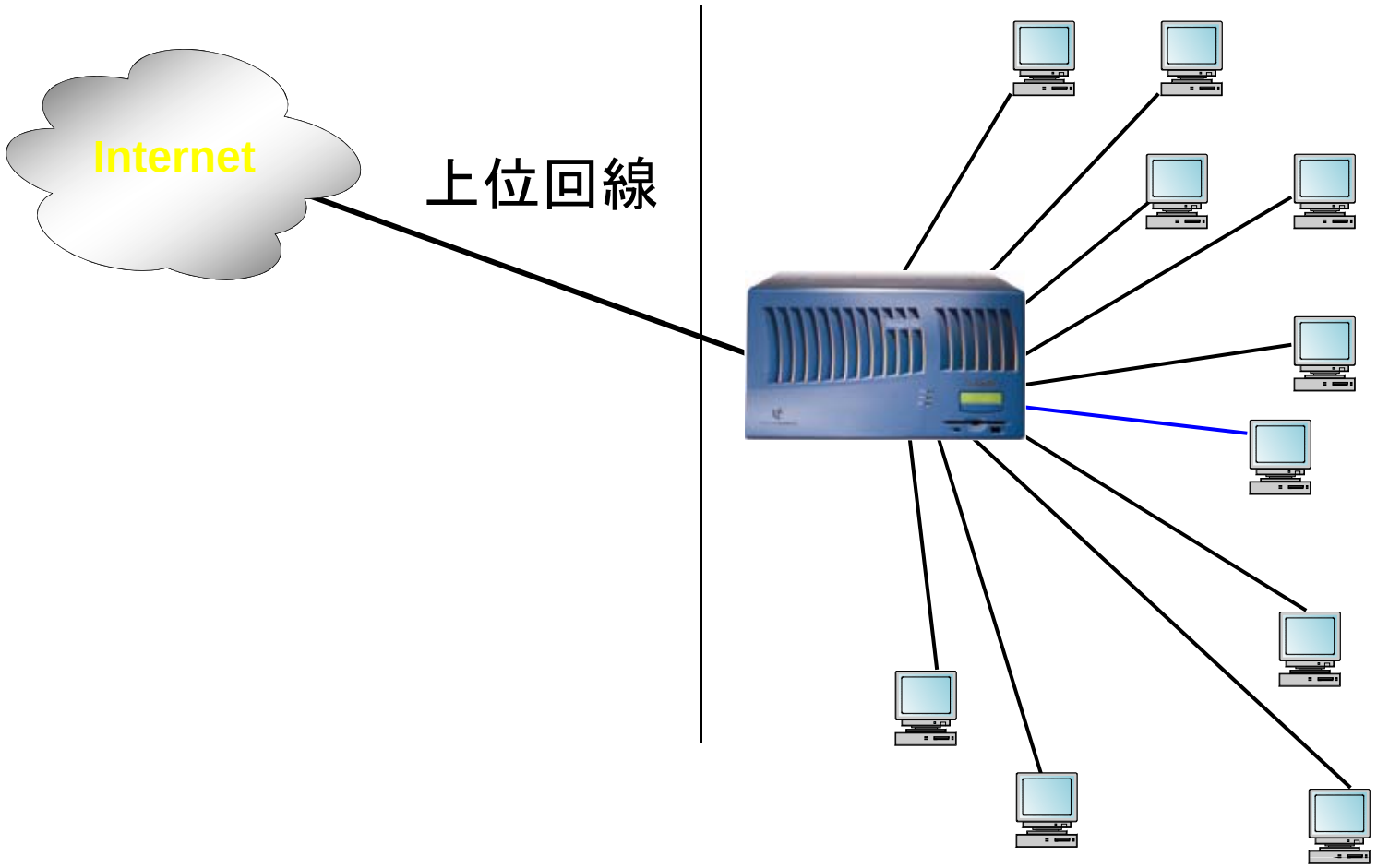
CDS・CDNによる負荷分散と
エッジからのコンテンツ配信イメージ図



キャッシュ

- データアクセスの偏りを利用し、再びアクセスが予想されるデータを「ユーザー」が速くアクセスできる場所に置いておく事
  - Temporal Locality
  - Spacial Locality
- ミスは透過的に処理される事が前提
- Cacheの例
  - Memory Cache
  - Disk (Buffer) Cache
  - HTTP Cache
  - DNS Cache

HTTPフォワードCache



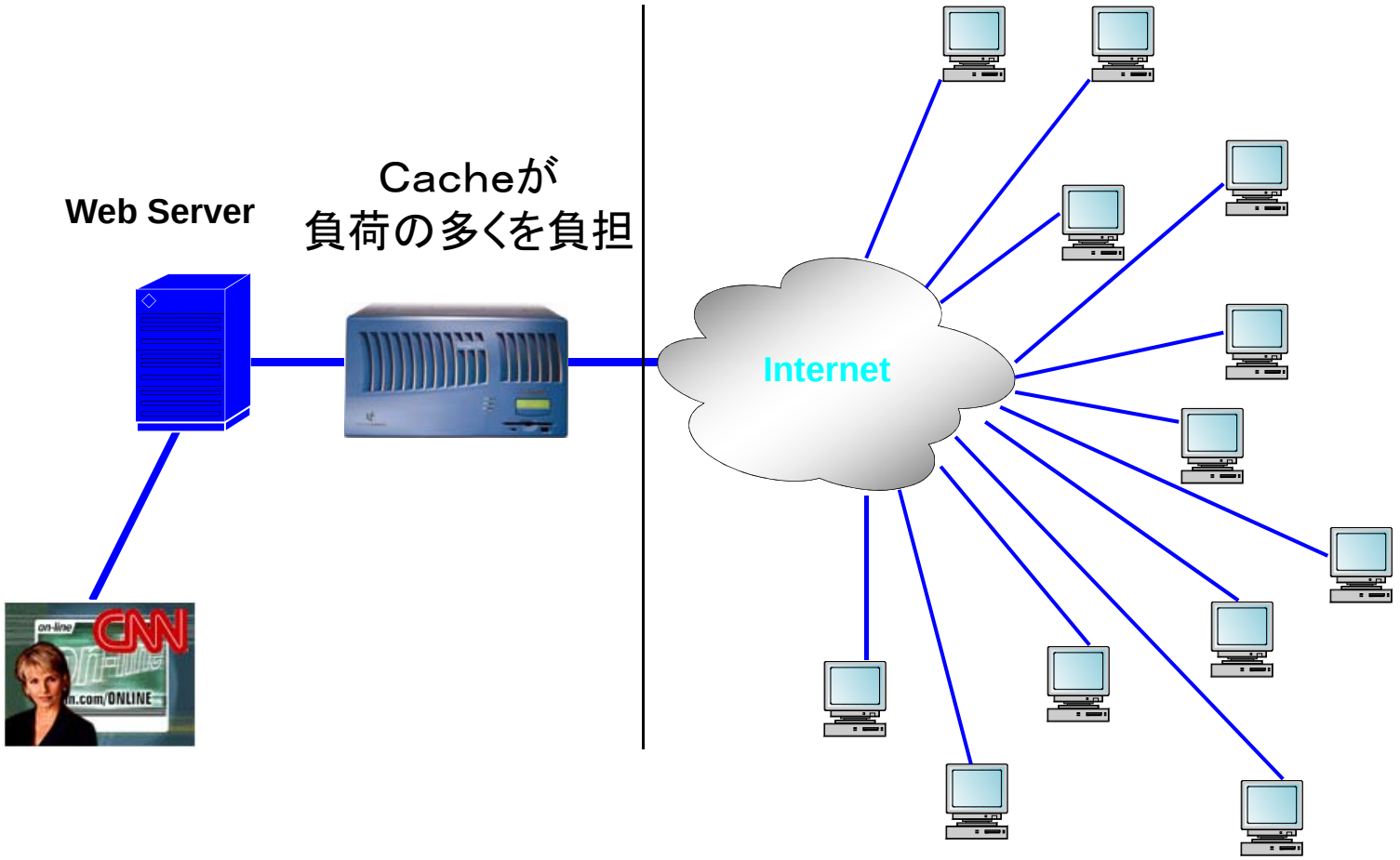
HTTPフォワードCache

- 必要に応じてウェブオブジェクトをローカルに保管（Cacheする）
- ２回目のアクセスからはCacheしたオブジェクトを転送（一度ダウンロードしたオブジェクトをリサイクル）
- インターネット経由でのウェブアクセスを減らす
  - レスポンスタイムの向上
  - 上位回線の帯域幅を節約
- アクセスコントロールと併用される事も多い

HTTPフォワードCache

- 利点
  - サーバー負荷を含む、データ転送経路上全ての帯域、負荷削減が可能
- 問題点
  - コンテンツホルダー側の協力が前提となっていない
    - コンテンツの鮮度保障が難易
    - Cache効率が高いオブジェクトの識別が困難
    - コンテンツクリエーター側の「文化」の違い
      - Cache経由のアクセスを禁止するCGI
      - アクセス数が減る事を嫌うクリエーター
- ネットワーク全体としてはプラスになるが、ビジネスとして成り立ちにくい

HTTPアクセレレーター



HTTPアクセレレーター

- 静的オブジェクト（画像等）をウェブサーバーに代わって配信
  - サーバー負荷を著しく軽減
  - サーバーはオーダー処理や動的オブジェクトの配信に集中する
- １台のCacheで複数台のサーバーを置換
  - ラックスペース（recurring cost)の節約
- セキュリティーを強化、ブレークイン及びDoSアタックに対する耐性を向上

HTTPアクセレレーター

- 問題
  - データーセンター内の問題しか解決できない
    - End-to-Endの配信保障ができない
    - 広帯域メディアの一般化により、ボトルネックがIDCからネットワークの末端に移動しつつある
- 極めて限られたスコープの問題しか解き得ないソリューション
  - CDNはHTTPアクセレレーターの延長上にあるソリューション
    - コンテンツホルダーの協力を前提としたエッジ配信の実現

基盤技術としてのキャッシュとOS

パフォーマンス

- CDNにより基幹回線への負荷が軽減
- CDNによりオリジナルサーバの負荷が軽減
- 問題は末端のCacheがどれだけのユーザーをサポートができるか
- Cacheが小型、高性能でないとビジネスとして成り立たせる事が困難
  - ➔ アプライアンス型のサーバの登場

CacheのOS及びパッケージング

- Cache用OSのタイプ及びそれぞれの特徴
  - 管理性
  - 拡張性
  - セキュリティー
  - パフォーマンス

CacheのOS、パッケージングの種類

- アプリケーション型
  - 汎用OS（UNIX，Linux等）上で走るアプリケーション、ソフトウェアを入手してマシンにインストール
- パッケージ型
  - 汎用OS上で走るアプリケーションではあるが、アプリケーションはプリインストール、OSはプリコンフィギュアーされている
- アプライアンス型
  - Cacheに特化された専用OSを使用
  - 専用OSのみを提供するモデルとハードウェア込みで提供するモデルがある

評価基準

- 管理性
  - 設定、管理は容易か？
- 拡張性
  - 基本のCacheにサービスを付加する事は容易か？
- セキュリティ
  - ブレークイン耐性
  - ダメージスコープ
- パフォーマンス

各タイプの特徴

	管理性	拡張性	セキュリティ	パフォーマンス
アプリケーション型	×	○	×	×
パッケージ型	○	△	△	×
アプライアンス型	○	△	○	○

管理性について

- アプリケーション型　－　悪い
  - 汎用OSをセキュリティーの穴が無い様に設定
  - 汎用OSをCache用にチューン
  - Cacheアプリケーションをインストール、設定
  - アプリケーションとOSのパッチが別々
- パッケージ型、アプライアンス型　－　良い
  - Cache稼動に必要な項目を設定
  - １つのパッチでアプリケーションとOSを処理可能

拡張性について

- アプリケーション型　－　良い
  - Cacheの付加機能を汎用OS上で走るプログラムとしてインプリメント（Cacheアプリケーションの付加機能API を使用）
  - しかしスケーラビリティーの点で不利
- パッケージ型　－　？
- アプライアンス型　－　普通
  - Cache上での付加機能プログラム実行は困難
  - iCapによりリモートサーバーを使い付加機能を実装する事が可能になった

セキュリティーについて

- アプリケーション型　ー　悪い
  - 汎用OSの設定でミスをしやすい
  - 汎用OSのセキュリティーバグは良く知られている
  - ブレークイン後、悪さをするツールがいっぱい！
- パッケージ型　ー　普通
  - 汎用OSのセキュリティーバグは良く知られている
  - ブレークイン後、悪さをするプログラムをダウンロードできるかも
- アプライアンス型　ー　良い
  - 専用OSなので、セキュリティーバグが少ない・知られていない
  - ブレークインしても大した悪さができない！

セキュリティー&パフォーマンス

- アプリケーション型　－　悪い
  - 汎用OSの設定でミスをしやすい
  - 汎用OSのセキュリティーバグは良く知られている
  - ブレークイン後、悪さをするツールがいっぱい！
- パッケージ型　－　普通
  - 汎用OSのセキュリティーバグは良く知られている
  - ブレークイン後、悪さをするプログラムをダウンロードできるかも
- アプライアンス型　－　良い
  - 専用OSなので、セキュリティーバグが少ない・知られていない
  - ブレークインしても大した悪さができない！

サマリー

- アプリケーション型は拡張性に優れるが、管理性、セキュリティ、パフォーマンスに劣る
  - サーバールーム内での使用には耐えるかもしれない
    - オペレーター、ファイヤーウォール保護、ラックスペース、電源
  - 「使いまわしが効く」というのは単なる逃げ？
- パッケージ型はどっちつかず、安いのでSOHOインストレーション向き？
- アプライアンス型が主流にならざるを得ない？
  - Cf. ルーター、ファイルサーバー

OSの役割

- 仮想マシンの提供
  - ハードウェアアクセスの複雑さを隠し、仮想化し、簡略化されたプログラミングインターフェース(system calls)をアプリケーションプログラムに提供する
- リソースのコントロール
  - CPU時間、メモリー、ディスク、ネットワーク等のリソースを複数ユーザー、或いはプログラムに分配する
- 上記を満足させた上で出来る限りのパフォーマンスを確保する

汎用OSと専用OSのタスクの違い

- 汎用OS
  - 複数ユーザー、或いは複数のプログラムが安全にシステムを共用できなければならない
  - コンパイラー、データベースから単純なあらゆるアプリケーションをまんべんなくサポートしなければならない
- 専用OS
  - 一つ、或いは少数のアプリケーションを効率良くサポートすれば良い

これを設計前提に言い換えると。

- 汎用OS
  - システム上で走るプログラムはシステムの敵
    - Buggy Code
    - Trojan horse
    - Virus
  - プログラムはお互い敵対関係にある
  - 「公平」なリソースの分配方法はOSが決める
  - ワークロードの予期が不可能
    - ワークロードに特化したインターフェースは作れない
    - ワークロードに特化したパフォーマンスチューニングはできない

これを設計前提に言い換えると。

- 専用OS
  - システム上で走るプログラムはシステムの一部であり、友好関係にある
  - ワークロードの予測ができる

専用OSの利点

- 専用OSを書くほうがずっと簡単
  - 簡単だからこそ、より優れた「解」を導き出す事ができる
- ソリューション・スペースが広い
  - アプリケーションのみならず、OS、ハードウェアまで再構築、再設計の対象にできる
  - 特にパフォーマンスの分野では、汎用OSでは考えられないような高度なチューニングが可能

機能別対比
仮想マシンの提供

- 汎用OS
  - Open/close/read/write等の一般性があり、安全に提供できるインターフェースしか提供できない
  - 安全性、セキュリティー上の問題から、殆どのハードウェアを仮想化せざるを得ない
- 専用OS
  - アプリケーションのニーズに特化したインターフェースが設置できる
  - ハードウェアは必要に応じて仮想化し、する必要が無いものは仮想化しなくともかまわない

機能別比較
リソースへのコントロール

- 汎用OS
  - システム内は「敵」だらけという前提、プログラムはそれぞれの檻の中で実行し、メモリーを触るにも、CPUを使うにもOSのチェックが入る
  - 「公平」はOSが定義し、必ずしも現実と一致するものではない
- 専用OS
  - システム内のプログラムは全て友好関係にあり、紳士協定及び譲り合いによる共有が可能、つまりOSによるオーバーヘッドを節約する事ができる

機能別比較
チューニング

- 汎用OS
  - OSはメモリーのアクセスパターンやファイルのアクセスパターン等のアプリケーションの特性について予期する事ができない、よってLRU等の最大公約数的最適化ポリシーしか実現できない
  - アプリケーションは最適化のための情報をOSより遥かに多く持っているが、限られたインターフェースのため、それをOSに伝える事ができない
- 専用OS
  - 通常アプリケーションが持つ最適化情報をOSが利用する事ができる

汎用OS、パフォーマンスへの障壁

- セキュリティーモデルによる制限
  - メモリープロテクション
  - CPUスケジューリング
- 予期不可能なワークロードへの対処
  - 仮想メモリー、Buffer Cacheマネジメント等の最適化ができない
- デザインパラメターと用法のミスマッチ
  - 数百コネクション向けにデザインされたネットワークスタック
  - 数百プロセスを前提にデザインされたプロセス構造

専用OSで可能になるチューニング

- データコピーの大幅削減
- データの重要度を反映したメモリーマネジメント
- アクセスパターンを理解したうえでのディスク先読みアルゴリズム、ディスクレイアウト
- 用法にマッチしたOSデザインパラメター
  - 数万コネクションサポート可能なネットワークスタック
  - 数万スレッドサポート可能なプロセス構造
- Solution Spaceの広さを利用した高度な最適化

汎用OSの問題例
過度のメモリーコピー

- CPUの高速化はめまぐるしいものがあるが、メモリーはさほど高速化していない
- 最新のCPUの実行時間のおよそ30%はメモリーアクセスによるストール
- よって最新ハードウェアを効率良く動かすにはメモリーリファレンスをできるだけ減らす事が課題である。

サマリー

- 基本設計段階から汎用OSはペナルティーを課されている
  - OS上で走るプログラムを敵視しなければならない
    - デバイスの過度な仮想化の必要性
    - 極限られたインターフェースの提供
  - ワークロードが特定できない
    - 設計パラメターの不一致
    - 各種ストラトジーのチューンの限界
- 専用OSはデザインの自由度が大きい

ＷＥＢからマルチメディア流通

- Streaming Media -

ストリーミングとは

- ダウンロードは行なわない
- 音声や映像を受信しながら再生を行う
- 画像・音声品質は接続している回線に依存
- サーバ・クライアント間で制御を行い効率的な伝送が可能

動画再生の方法

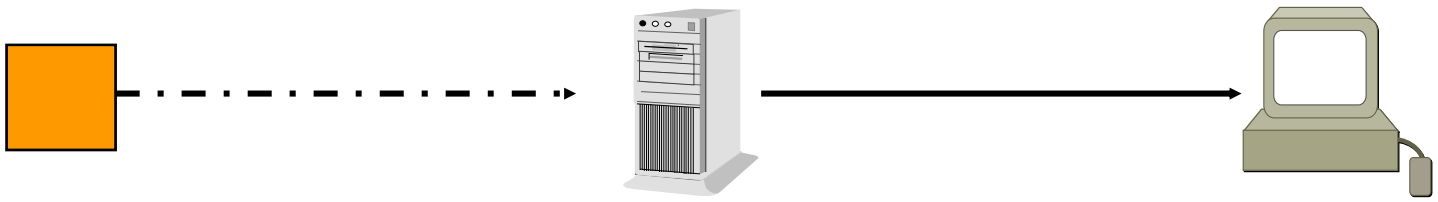
- ダウンロード再生
- 疑似ストリーミング
- ストリーミング

ダウンロードによる動画再生

- メディアファイルをファイル転送によってクライアント側にコピーし終わった後再生
- 再生開始までの時間はファイルサイズと回線容量に依存する
- クライアントにファイルが残るため加工される可能性がある

疑似ストリーミング
- パイプライン -

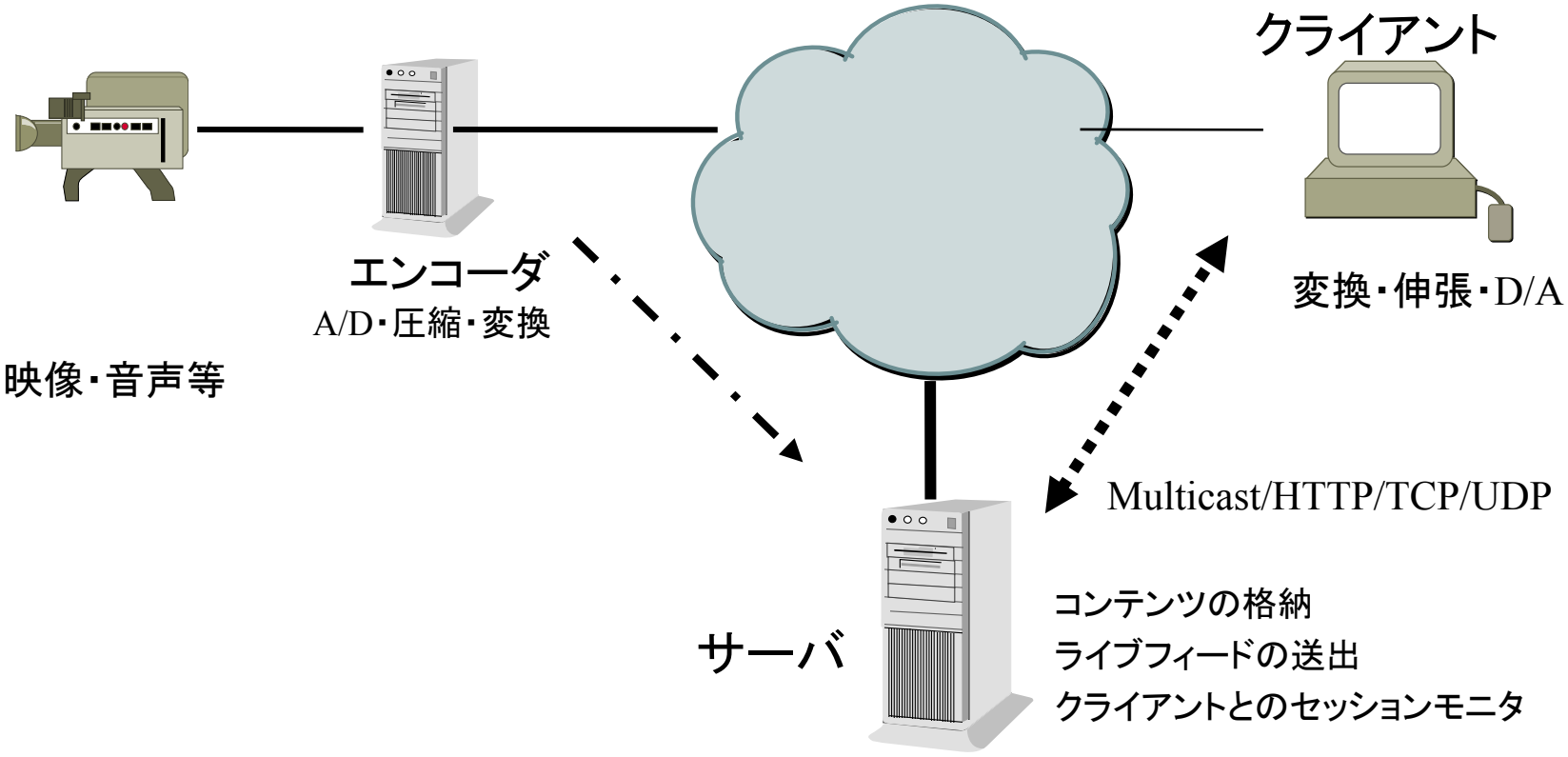
- 従来の方法でファイル転送を行いつつ、転送終了を待たずに再生
- フロー制御ができない為、安定した再生が困難
- ライブに対応できない



ストリーミング

- 連続的にデータ転送を行い再生する
- サーバ・クライアント間で制御を行い効率的な伝送が可能
- バッファリング等の技術を応用

ストリーミングの流れ


クライアント
エンコーダ
A/D・圧縮・変換
変換・伸張・D/A
映像・音声等
Multicast/HTTP/TCP/UDP
サーバ
コンテンツの格納
ライブフィードの送出
クライアントとのセッションモニタ

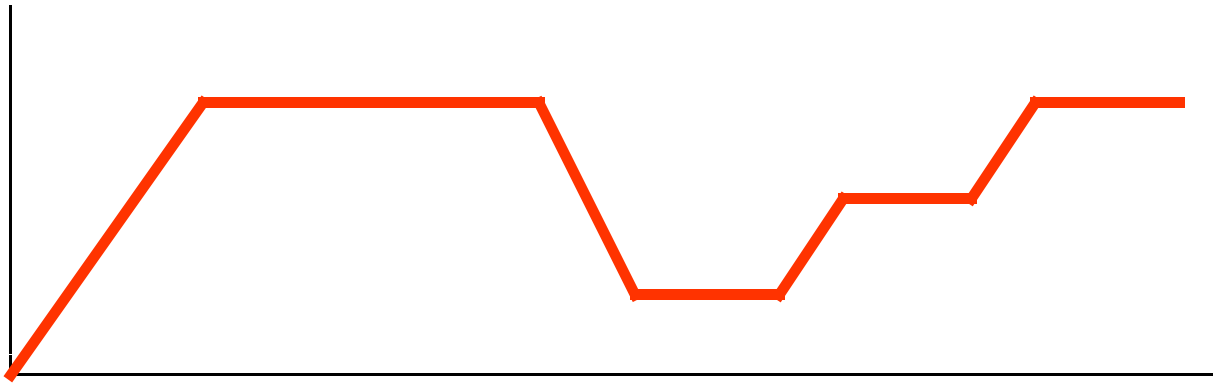
Streamingに用いられる技術

- マルチエンコード
  - 一台のエンコーダで複数のファイルを作成
    →複数の帯域向けにデータを作成する必要がない
  - SureStream / Intelligent Streaming

    ---注意点----
  - CPU/Memoryの力が必要
  - ファイルサイズが大きくなる

Streamingに用いられる技術

- バリアブルビットレート
  - クライアントの受信状況に応じて、サーバがダイナミックに送信データのビットレートを変化させる
  - ヘルスチェックのための TCP コネクションをサーバとクライアント間にて保持



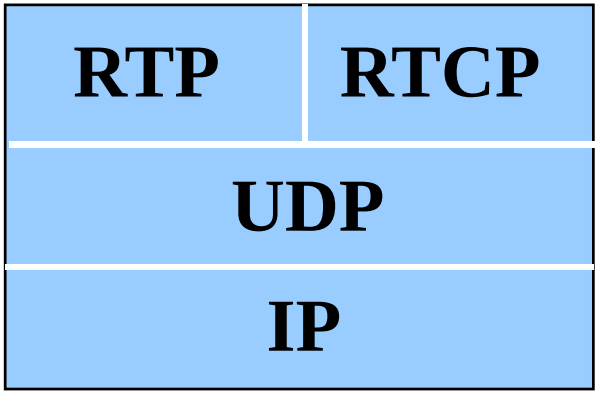
バッファリング

- 不安定・低速な伝送回線への対応
- 受信したデータをそのまますぐに再生せず、キューイングを行う
- データ落ちなどに対応する
- 入力が出力を上回った時もバッファに蓄積
- バッファリングのための時間も遅延となる

Streamingに用いられるProtocol

IETF MMUSIC-WG にて制定

- Real-time Transport Protocol, RFC1889
- RealTime Streaming Protocol, RFC2326
  - port 554/tcp,udp

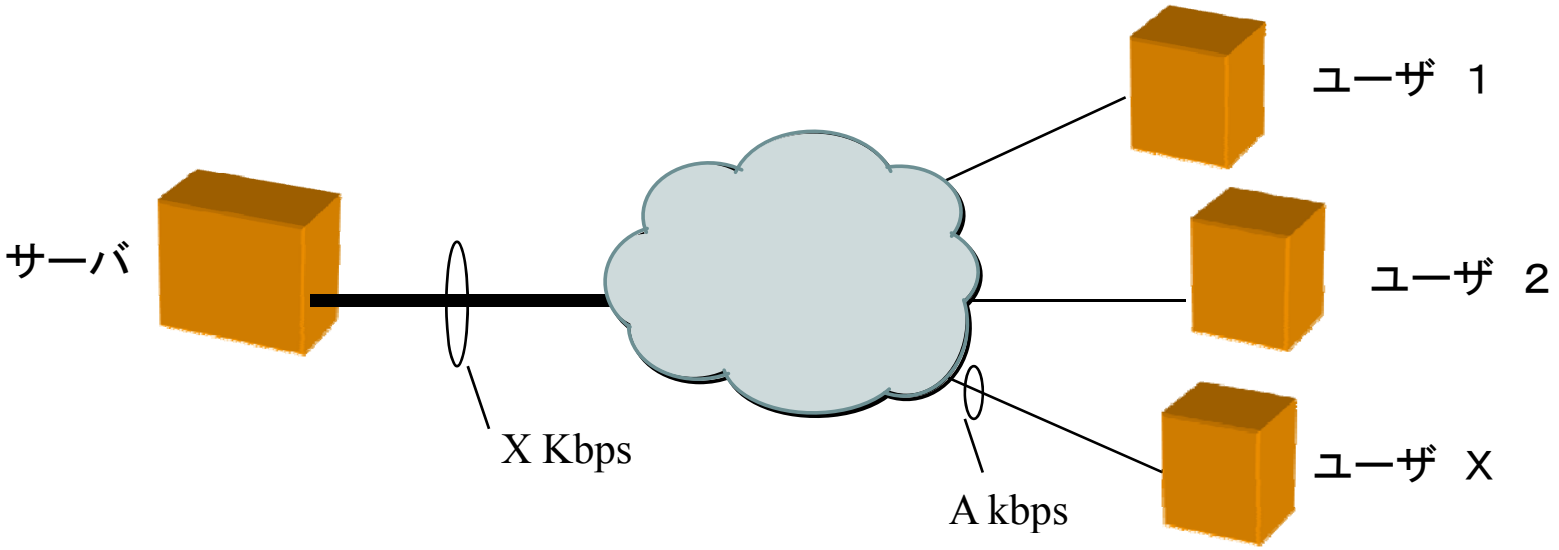


ストリーミングアプリケーション

- RealNetworks RealSyatem
  - 歴史が長く、シェアも一番
  - サーバのプラットホームが多種類
- Microsoft Windows Media Technologies
  - ほとんどのコンポーネントがフリー
  - シェアの伸び率がNo1
- Apple Computer Quicktime
  - ハリウッド系映画の予告編などのコンテンツが多い
  -

負荷分散の必要性

- サーバ･クライアント間



要求される帯域(X kbps)＝　A kbps x user数

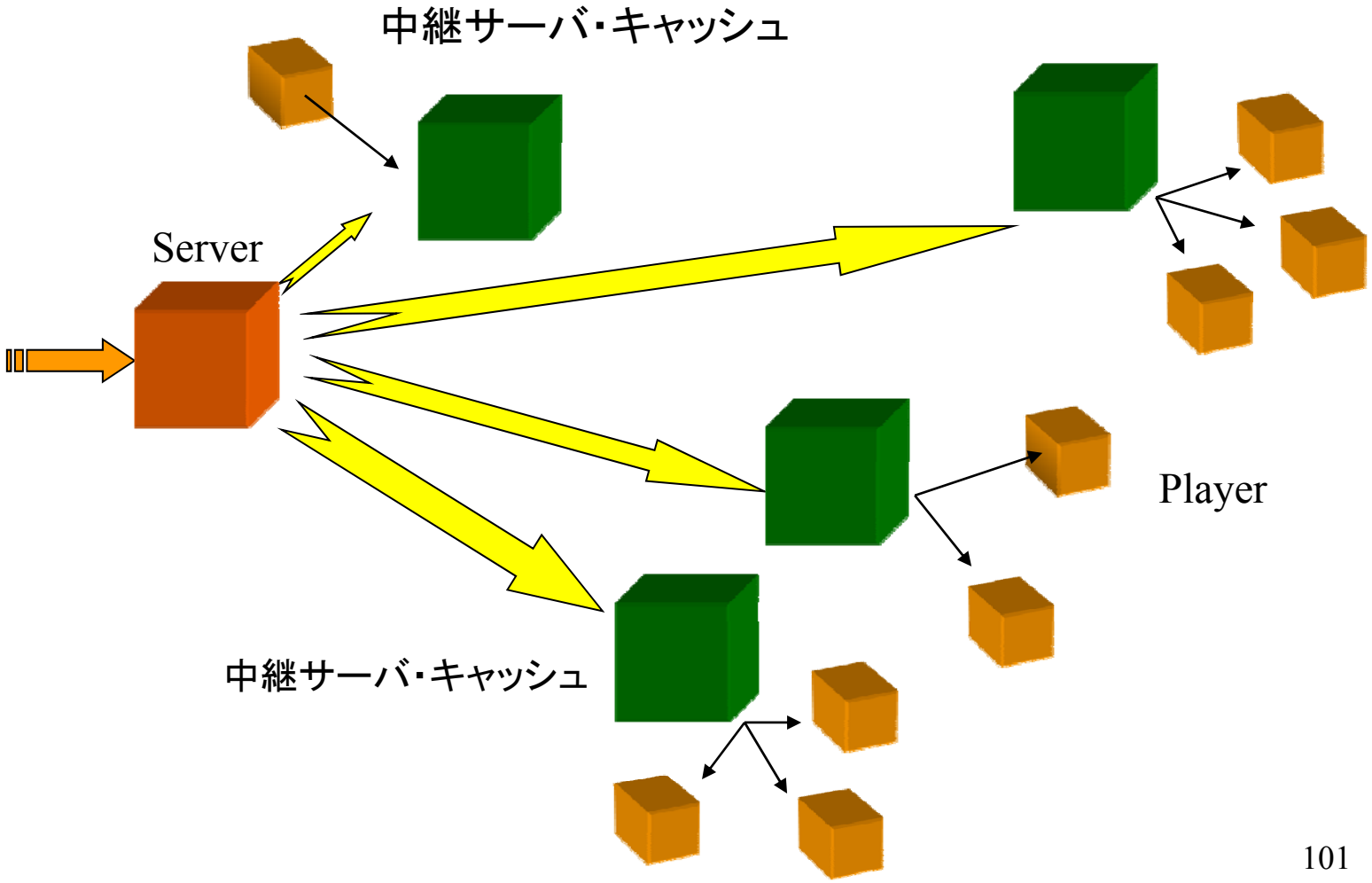
負荷分散の必要性

- 回線で処理できる容量は決まってくる
- 1.5Mbpsの回線で20kbpsのストリームを75同時アクセス
- 45Mbpsの回線で45kbpsのストリームを1000同時アクセス

Webサーバへのアクセス

- 中継時の高負荷問題
  - そもそも中継へのポインタを持つWebサーバにアクセスできない
  - すでに1995年には知られていた現象
  - Streamingの一般化に伴い、大きな問題としてクローズアップされてきている

中継装置の設置



IPストリーミングの今後

- 広域負荷分散の重要性が高まる
  - 「よいコンテンツをよいクオリティで配信できるのが、よいサービスプロバイダ」
- ブロードバンドのブレイク
  - 300kbpsのコンテンツクオリティのインパクト

WEB技術