システム構成とプロトコル

1. 階層化 は何のため？

- 各種の機能を誤り無く作り上げるのは大変
- モジュールを入れ替え可能にして アップグレードを可能にする。

→モジュール化

➔結局は、「選択肢」、「冗長性」 の確保

➔広い意味での、「(事業)継続性」

(*)BCP; Business Continuity Plan.

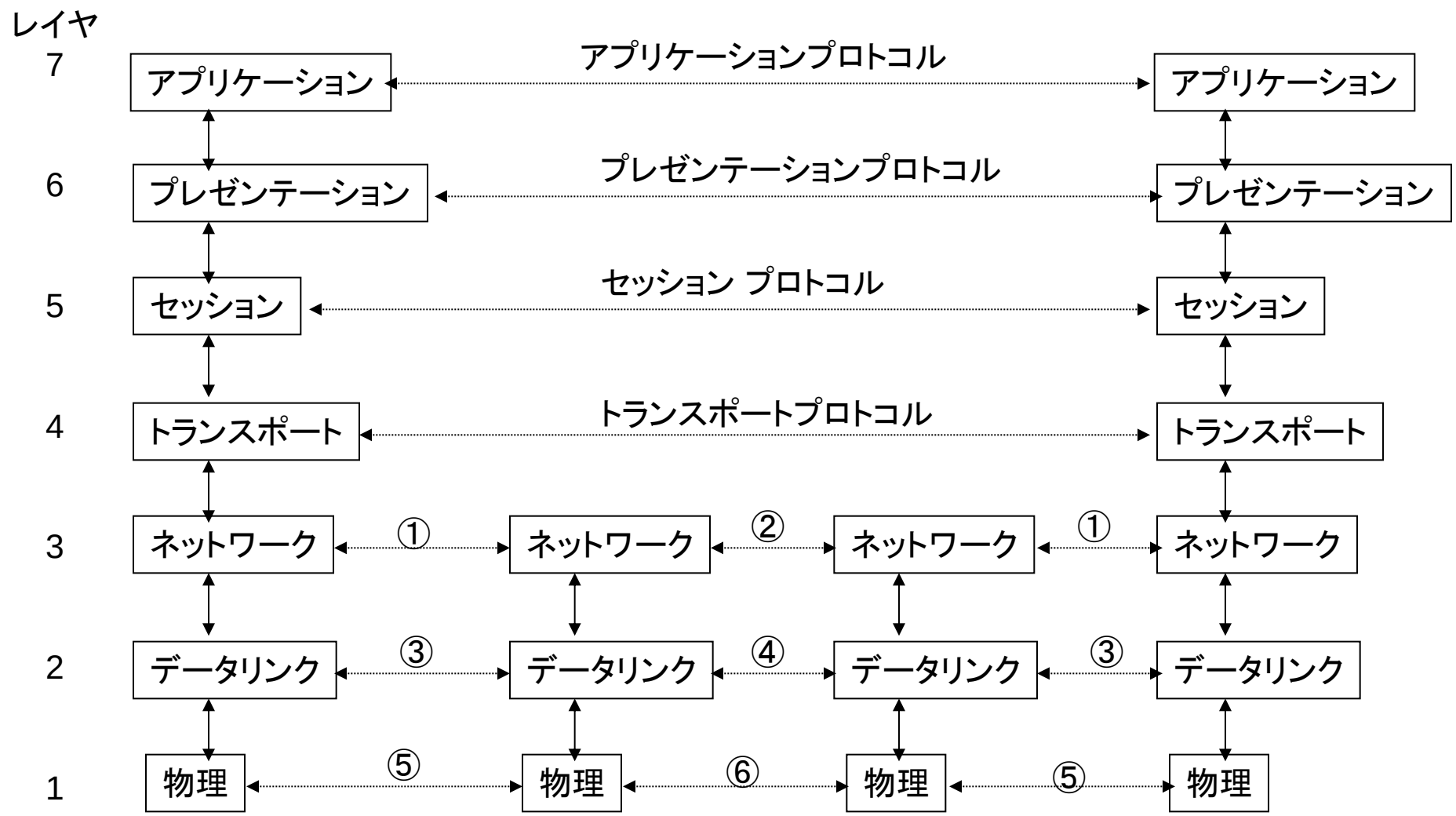


図1-11. OSI 参照7レイヤモデル

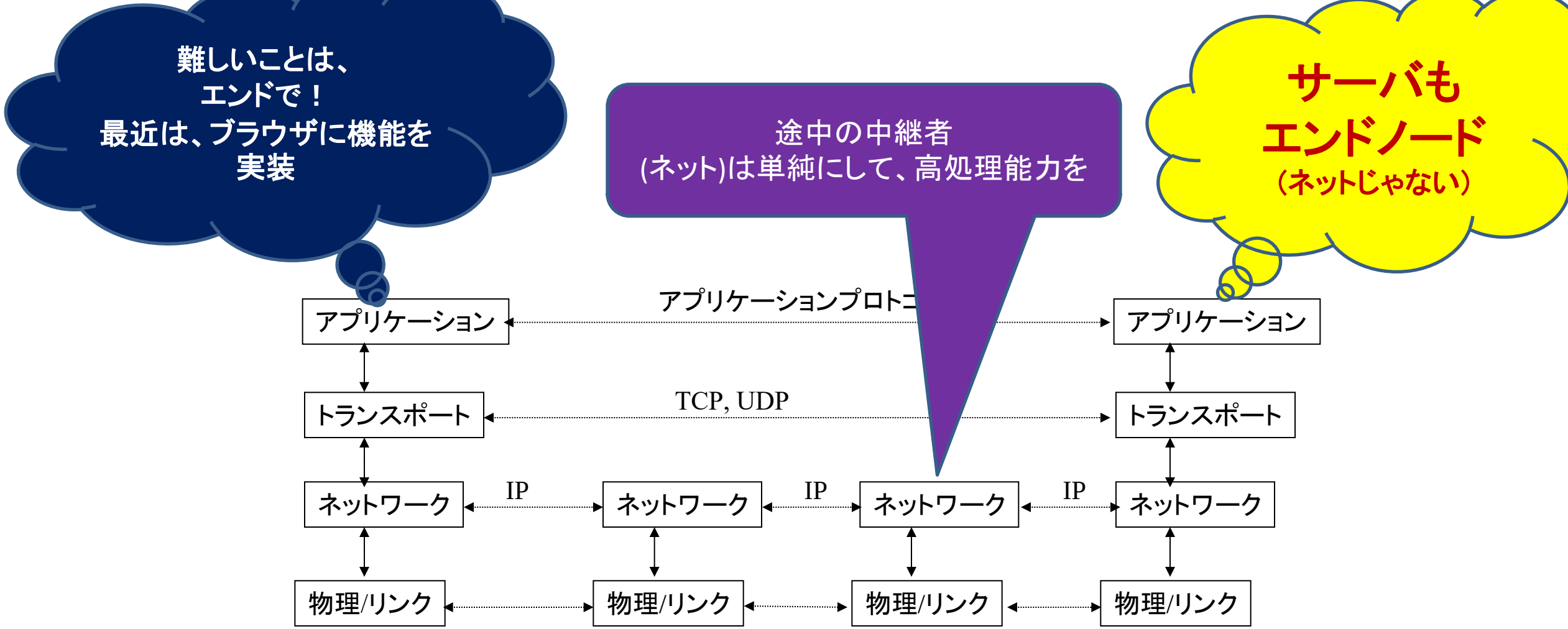
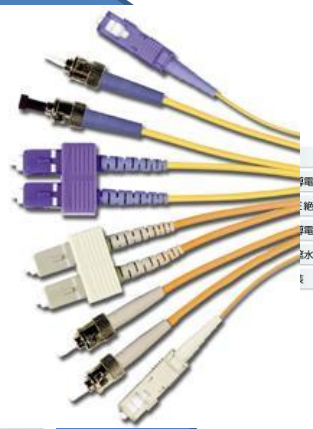
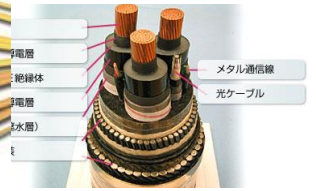


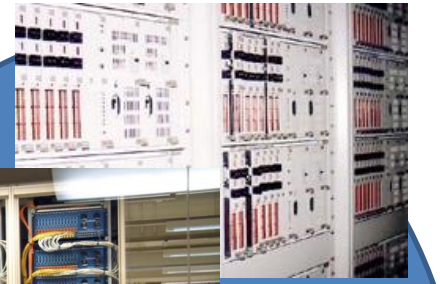
図1-12 TCP/IPの4レイヤモデル

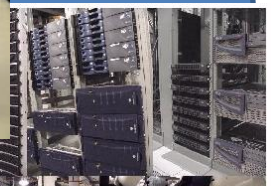
電話局



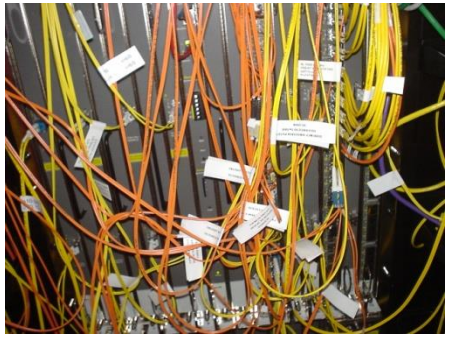


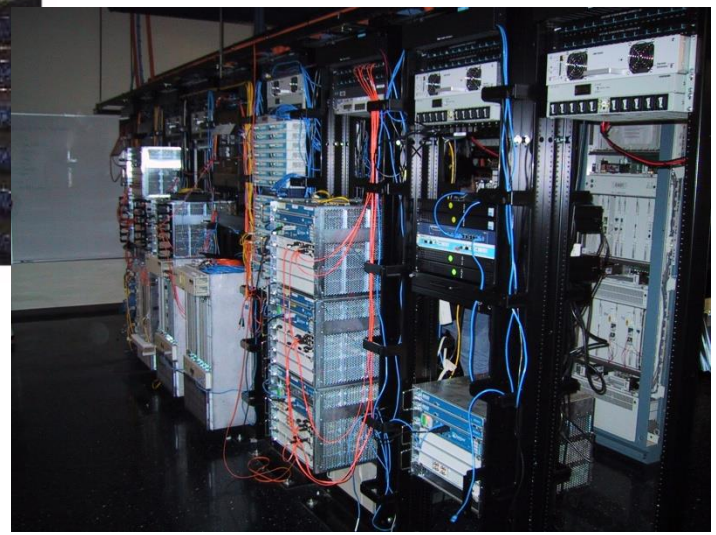
自宅

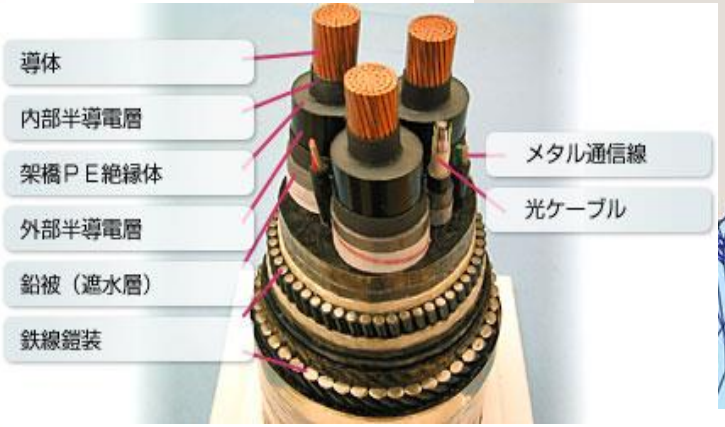




データセンタ



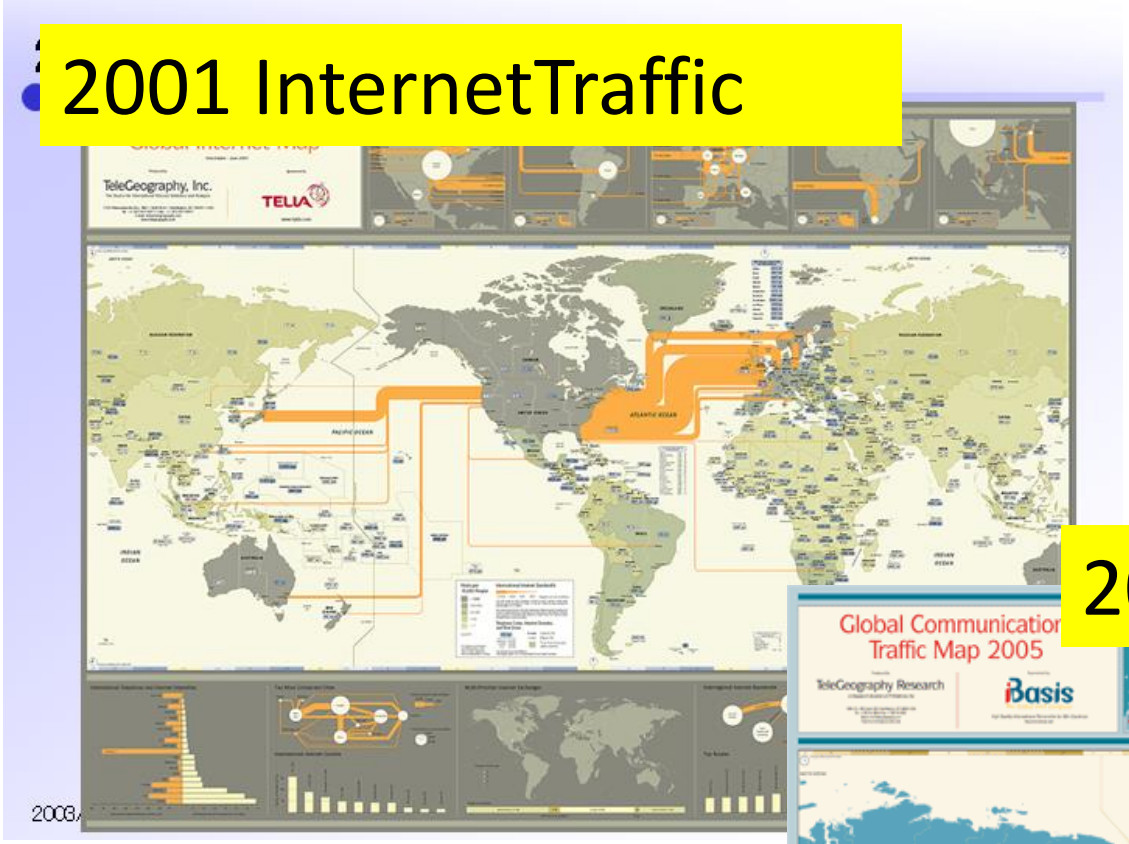



導体
内部半導電層
架橋ＰＥ絶縁体
外部半導電層
鉛被（遮水層）
鉄線鎧装
メタル通信線
光ケーブル

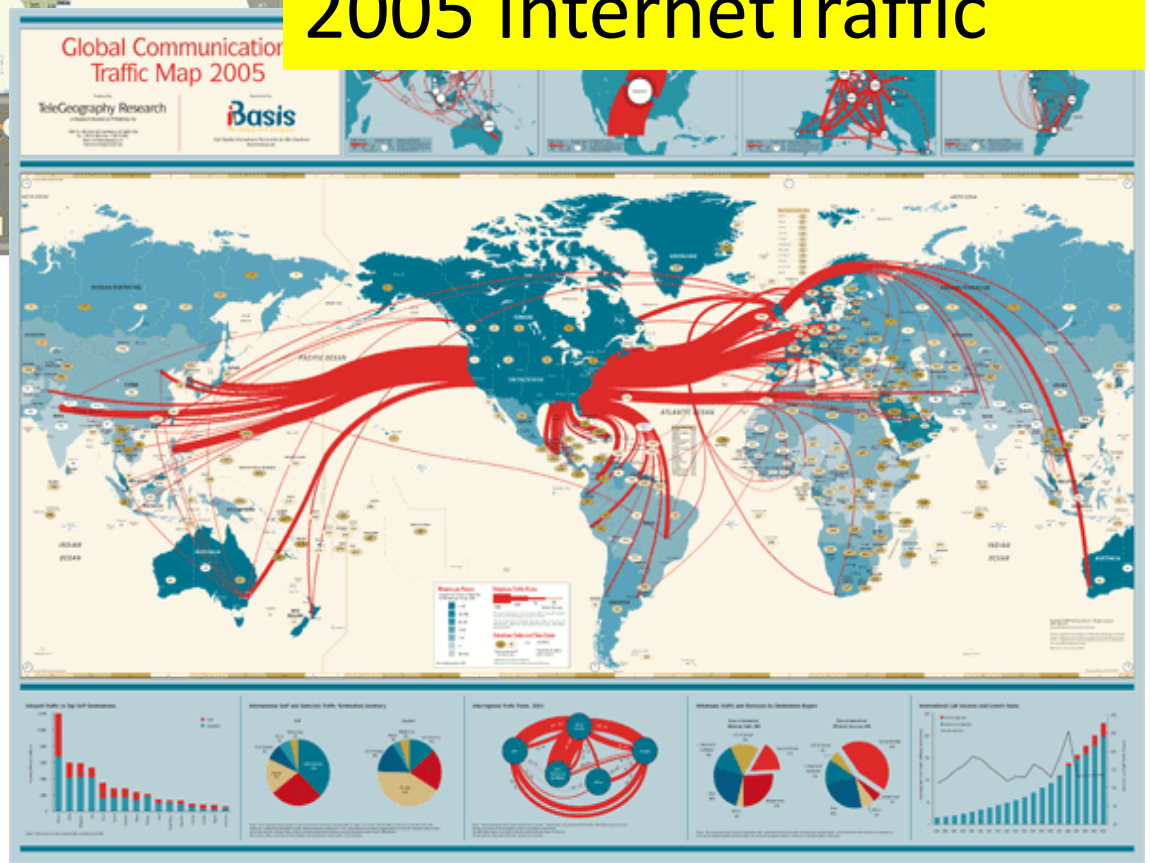

ASIA
Japan-U.S. CN
PC-1
TPC-4
TPC-5CN
China-U.S. CN
R-J-K
KJCN
CKC
APCN2
EAC
C2C
FLAG
H-J-K
TPC-3
APCN
A/C
GP
G-P-T


97/08/01 , 11:40:24 , 01277m
(cm)
(C) JAMSTEC

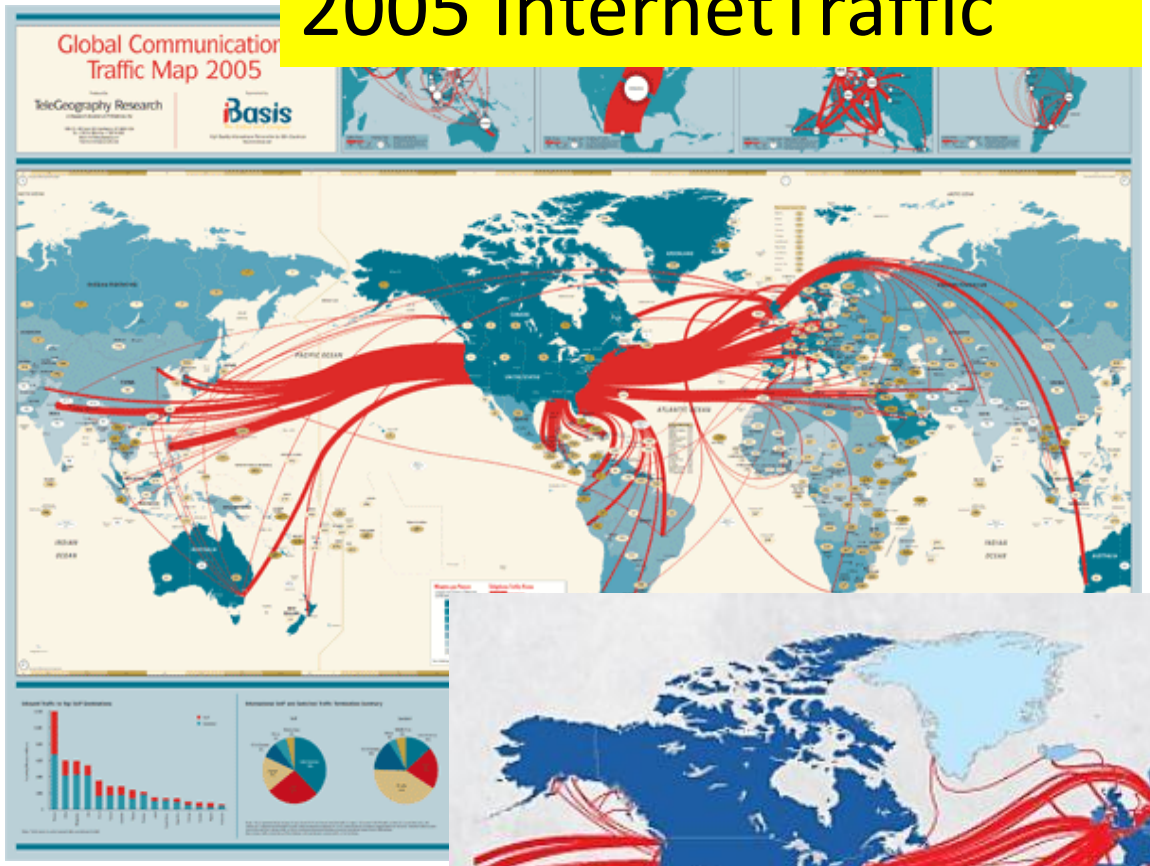
2001 InternetTraffic



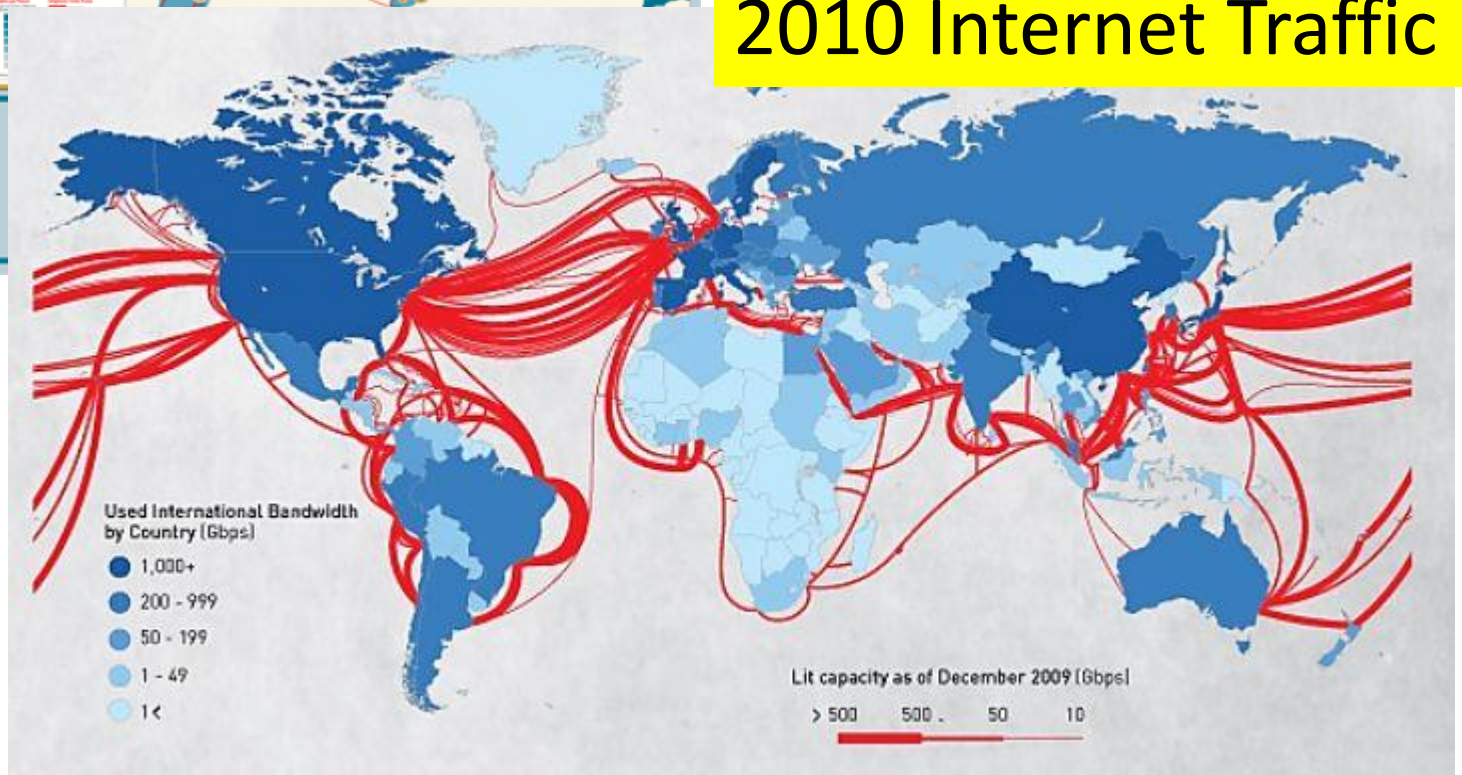
2005 InternetTraffic



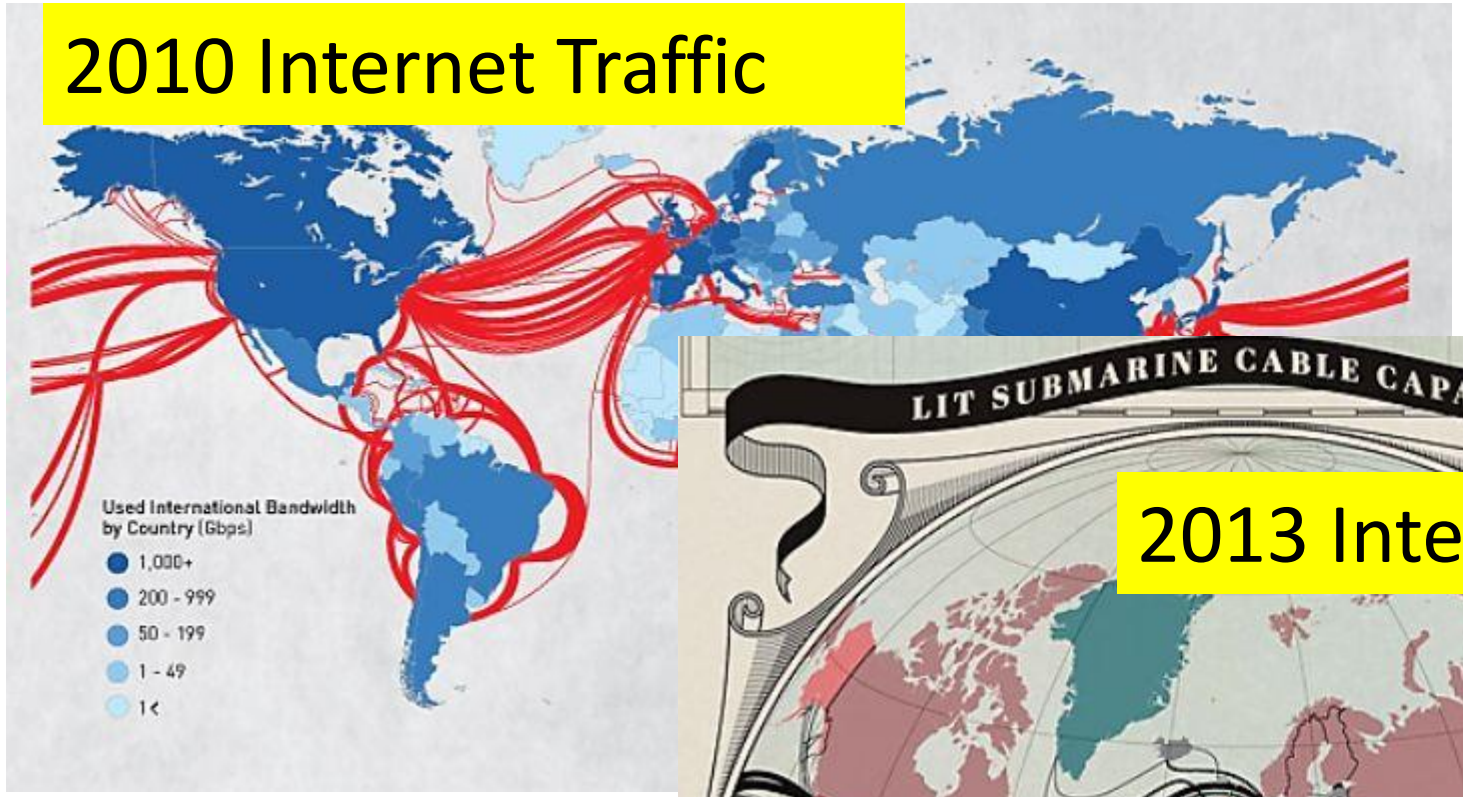
2005 InternetTraffic



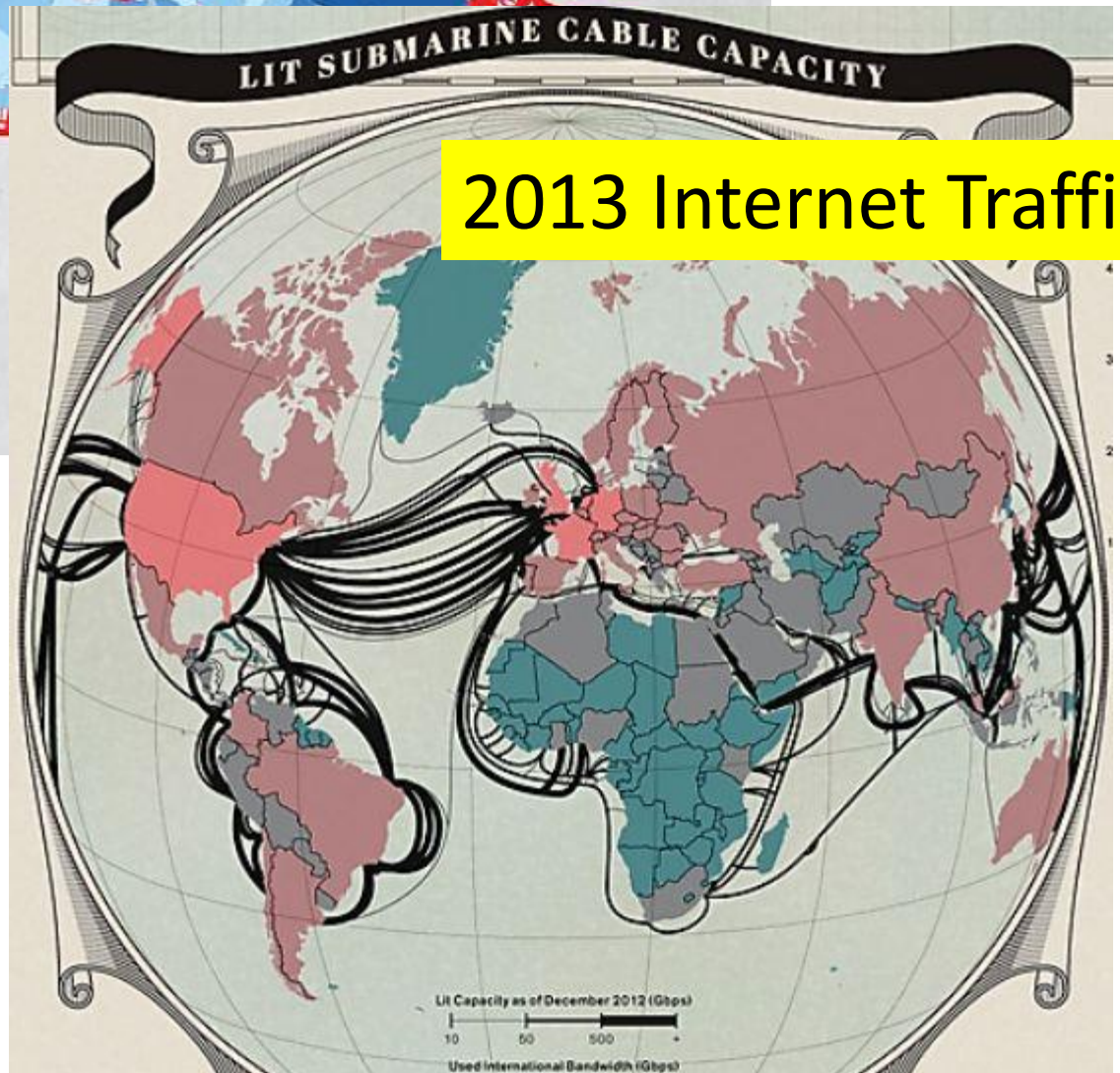
2010 Internet Traffic



2010 Internet Traffic

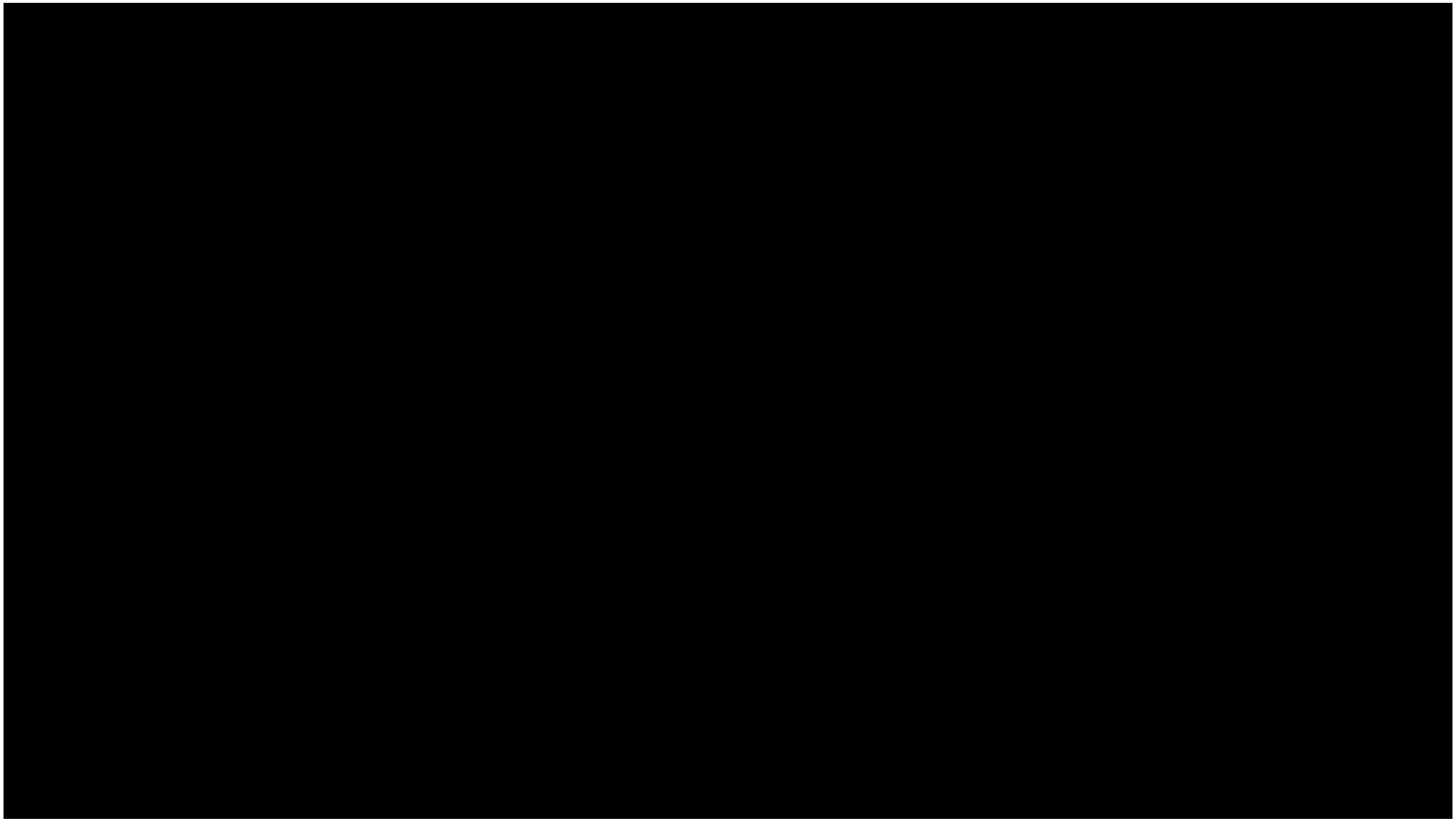


2013 Internet Traffic



[Submarine Cable Map]

https://www.visualcapitalist.com/wired-world-35-years-of-submarine-cables-in-one-map/



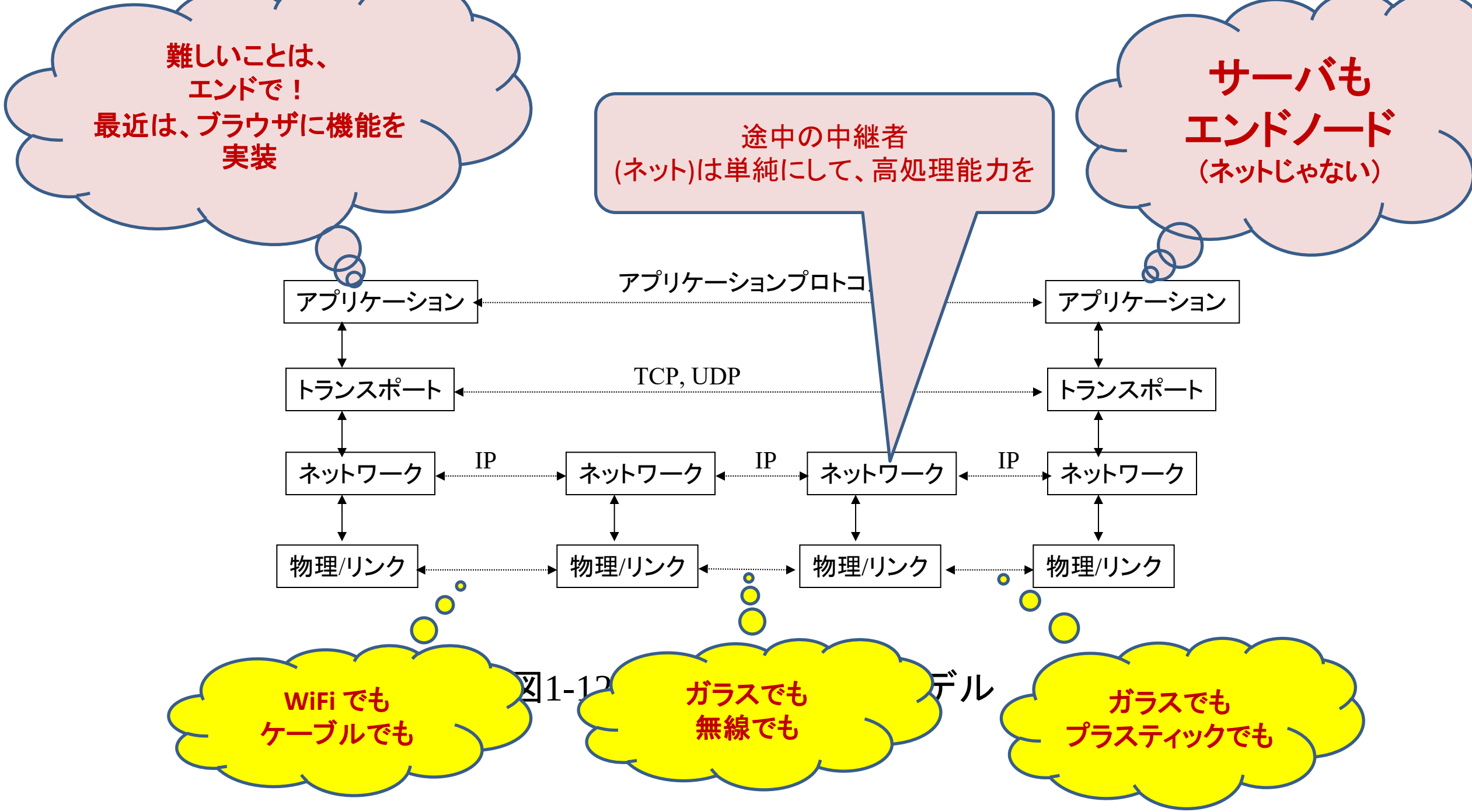
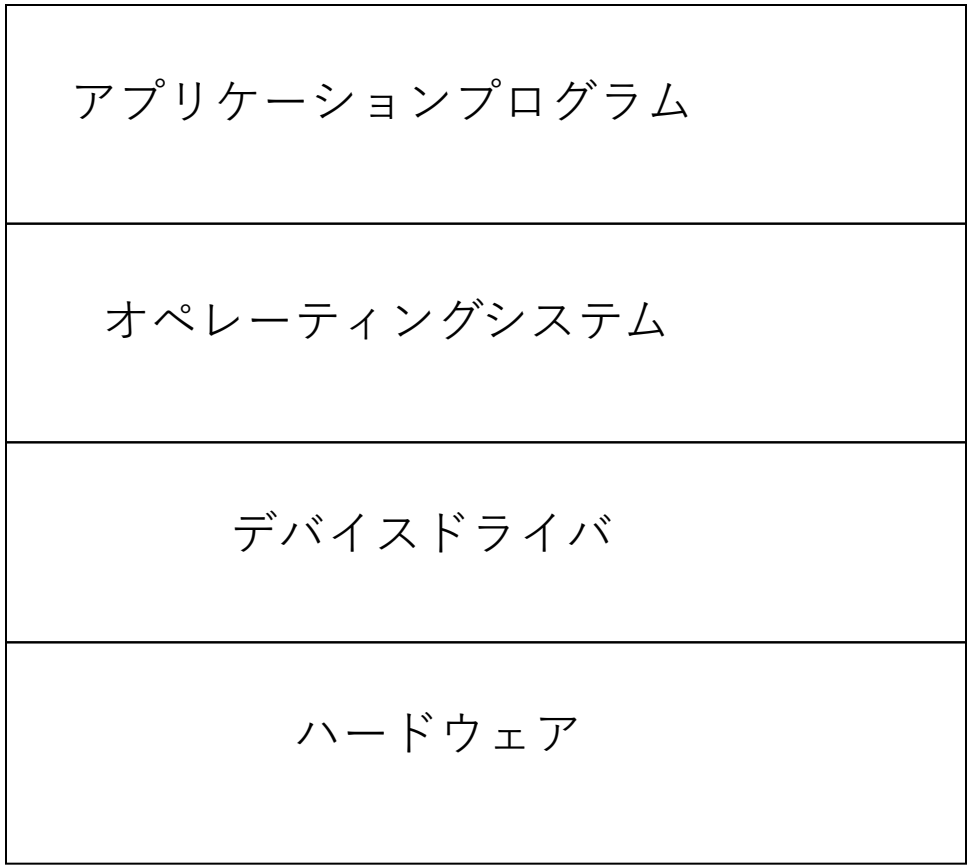

難しいことは、
エンドで！
最近は、ブラウザに機能を
実装
途中の中継者
(ネット)は単純にして、高処理能力を
サーバも
エンドノード
（ネットじゃない）
アプリケーション
アプリケーションプロトコル
アプリケーション
トランスポート
TCP, UDP
トランスポート
ネットワーク
IP
ネットワーク
IP
ネットワーク
IP
ネットワーク
物理/リンク
物理/リンク
物理/リンク
物理/リンク
WiFi でも
ケーブルでも
ガラスでも
無線でも
ガラスでも
プラスティックでも

図1-12 デル

cf. パソコンのソフトウェア



- それぞれのソフトウェアは直下のモジュールの機能だけを使用
- とくに、ハードウェアの差異はデバイスドライバで吸収し、オペレーティングシステム以上は機種非依存

階層化の嬉しい点

- オープン化
  - 取替え可能
    - 選択肢を与える
    - 競争原理の発生
    - 機器の継続的確保
  - 分担が可能
    - 一人で何でも行う必要がなくなる
    - 選択肢が増える(e.g., Third Partyからの提供)

2. プロトコル

- もともとは、外交に際するお作法/ルール
- 情報通信では、
  - データのやり取りを行う際の お約束事。
  - 会社の中 と 会社間 の2つがある。
    - 会社の中： モジュール間インターフェース
    - 会社間　： 通信プロトコル(e.g., IP, http)

階層プロトコル

- それぞれの層のソフトウェアは直下の層の提供する機能だけを使用
- 上位層は下位層非依存

階層プロトコル

- <*N*>層は、<*N*–1/*N*>インタフェースを通じて提供される<*N*–1>サービスを用いて<*N*>エンティティ間で<*N*>プロトコルに基づく通信を行うことにより、<*N*/*N*+1>インタフェースを通じてより付加価値の高い<*N*>サービスを<*N*+1>層に対して提供する

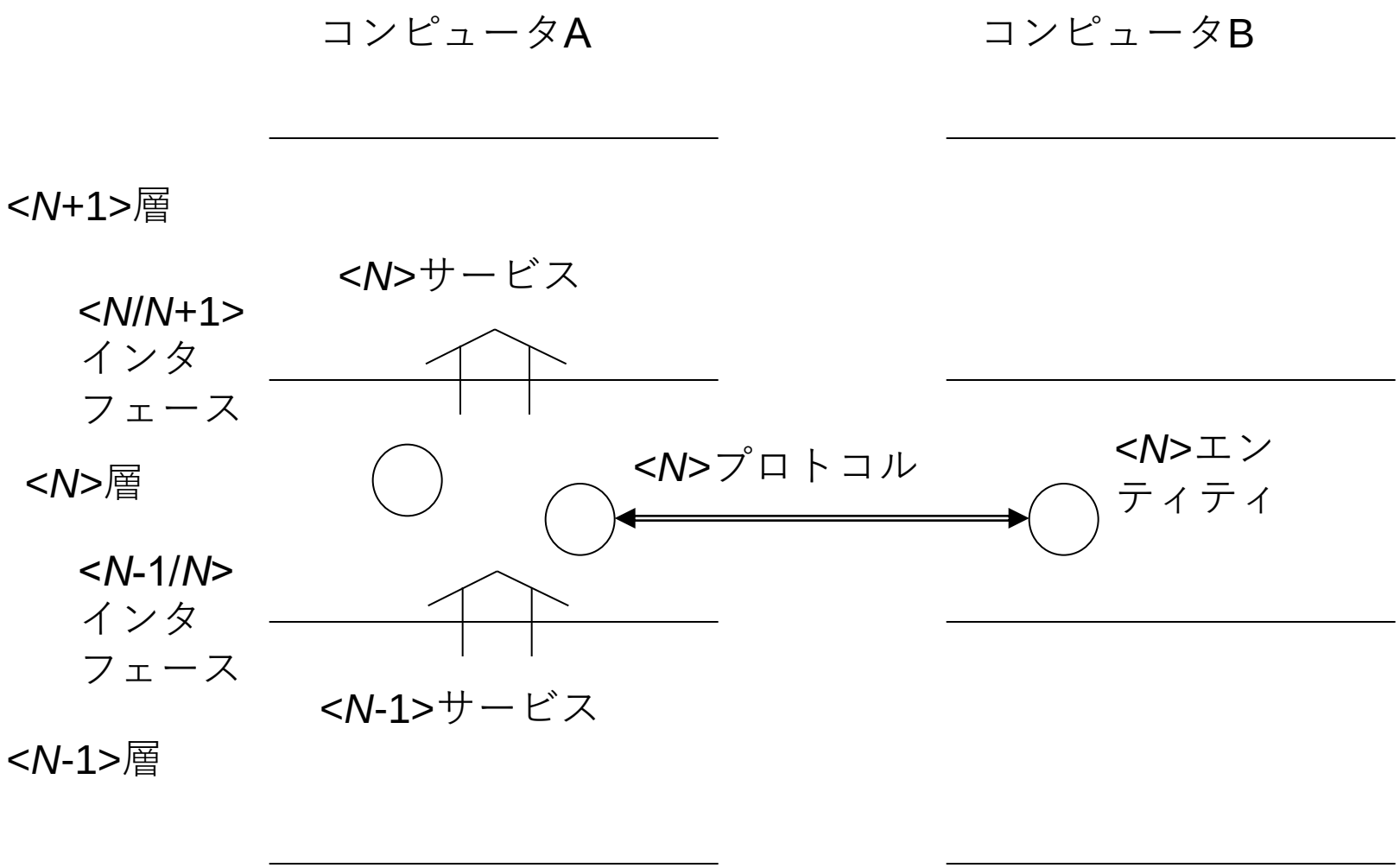
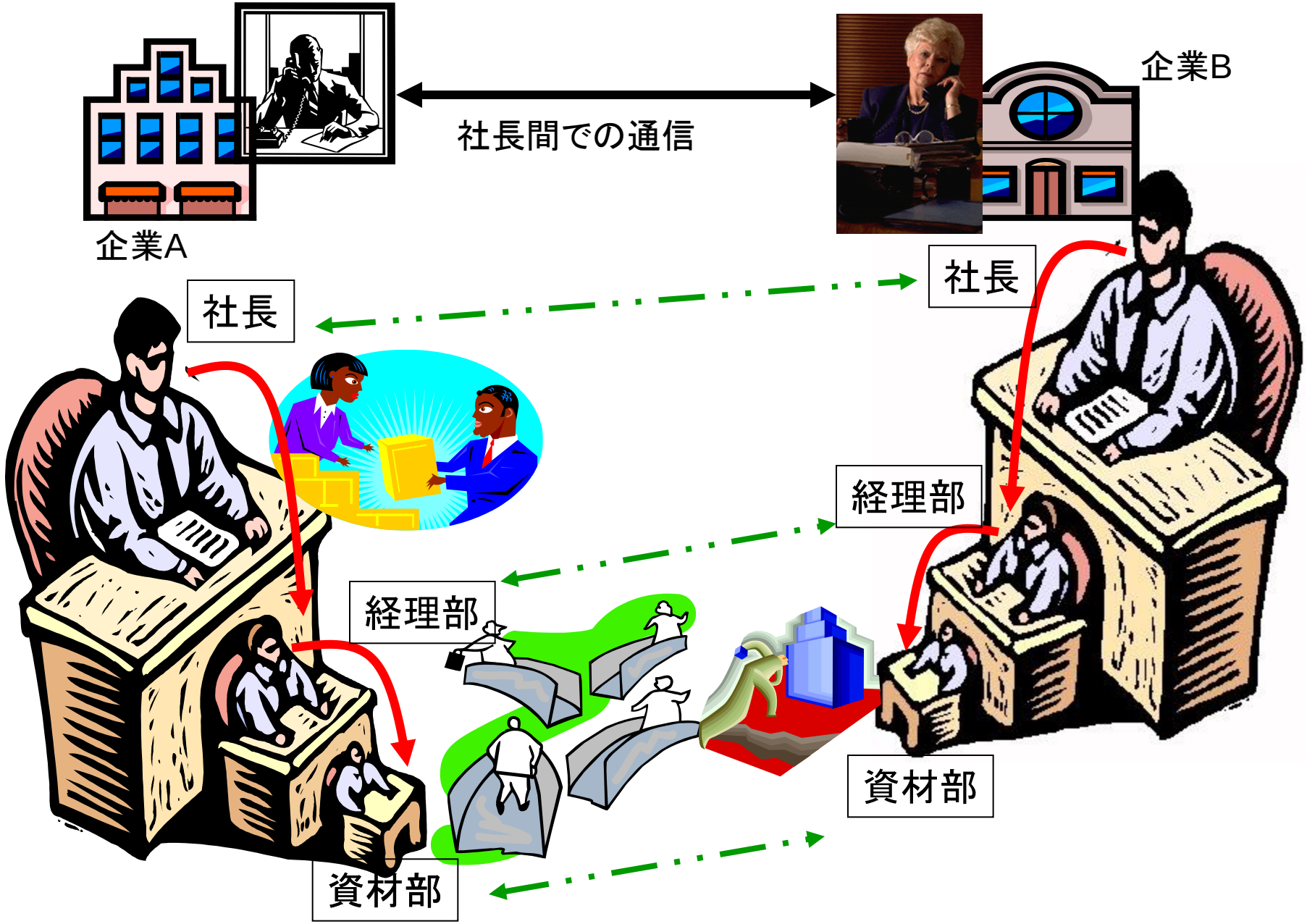

コンピュータA
コンピュータB
<N+1>層
<N>サービス
<N/N+1>
インタ
フェース
<N>層
<N>プロトコル
<N>エン
ティティ
<N-1/N>
インタ
フェース
<N-1>サービス
<N-1>層

図1-7. 企業間でのメッセージ通信を例にとったレイヤ構造の概念



システムアーキテクチャ

- エンド ツー エンド
  – トランスペアレントな基盤の上に、自由なサービスの展開
  (Q. Proxy/Cache は、エンド・ツー・エンドか?)

- エンド ツーエンドを もとに 2つのアーキテクチャ
  – ピア・ ツー・ ピア
    - 情報通信機器 が主役の アーキテクチャ
  – クライアント サーバ
    - サービスプロバイダが主役のアーキテクチャ

CS vs P2P

(client-server) vs (Peer-to-Peer)

- とちらも、”Transparent” な情報通信基盤
- “Server” は、「点」である必要はない。
  - “Server” のネットワーク化
  - “Proxy/Cache”もネットワーク化の一種???
- Client-Server
  - “Server” での機能/処理の共有
    - コスト削減 、高品質サービス、サービスの継続性
  - ISPもIT部門設備(企業/大学)も “Server” の一つ
- Peer-to-Peer
  - すべての機器が、サーバにもクライアントにもなる。

シグナリング(制御信号)

- アウトバンドシグナリング
  - ユーザデータ(Data-Plane) と 制御データ(Control-Plane) が分離している。
  - 例: 電話、MP3プレイヤ、放送
- インバンドシグナリング
  - ユーザデータ(Data-Plane) と 制御データ(Control-Plane) が縮退している。
  - 例: インターネット

状態管理 ポリシー

- ハードステート
  - コネクション型サービスにおいては、通信を開始する前に、シグナリング手順を実行し、通信する情報機器の間に、仮想的な通信回線(＝コネクション)を確立する。多くの場合(インターネットにおけるTCPは例外とみることができる)、データが転送される経路は、シグナリング時に決定され、通信路が解放されるまで、同一の経路が維持されるのが、一般的。システムの状態は、インスタンスが発生して、消滅するまで状態を変化させない。電話サービスにおいては、電話網内でコネクションが利用する経路上の情報機器(＝交換機)や通信回線に障害が発生した場合、基本的には、通信を継続することができない。

状態管理 ポリシー(続)

- ソフトステート
  - コネクションレス型サービスにおいては、シグナリング手順を持たず、ネットワークの状態に応じて、必要な時には、ディジタル小包の転送経路を動的に変化させる。システムの状態を、常に更新(Update)しながら運用するシステムを、ソフトステートのシステムと呼ぶ。典型例は、TCP/IP技術を用いたインターネット。IPパケットの転送経路は、動的に変化することを前提として、データ通信手法が設計されているために、ある通信路やルータに障害が発生しても、自動的に、IPパケットの転送サービスを実現可能な他の経路を探し出し、サービス提供を継続させる。

放送、電話、インターネットの技術比較

	放送	インターネット	電話
クライアント・サーバ or ピア・ツー・ピア	クライアント・サーバ	ピア・ツー・ピア	ピア・ツー・ピア
エンド・ツー。エンド or ゲートウェイ	ゲートウェイ	エンド・ツー・エンド	ゲートウェイ
オーバレイ or ピア	ピア	オーバレイ	ピア
保証型 or ベストエフォート型	保証型	ベストエフォート型	保証型 (+ベストエフォート)
シグナリング	なし	インバンド	アウトバンド
ハードステート or ソフトステート	なし	ソフトステート	ハードステート

通信 品質 (Communication Quality)

- QoS: Quality of Service
- CoS : Class of Service

『冗長性』、『安定性』の考え方 :
➔ 『頑丈』 vs 『ふにゃふにゃ』

- Guarantee (保証) vs Best-Effort (最大努力)
  - 運命的資源共有(Fate Share、Single Point of Failure)
  - State-Full と State-Less システム
  - 自律性

『ベスト・エフォート』で大丈夫!

1. 『保証型サービス』も実は『ベストエフォート』
   - 容量・能力を超えるとブロック

2. 『競争環境』が 品質の維持・向上の鍵
   - 『選択肢』の環境
   - 『入れ替え・取り換え可能』な環境
     (by オープンな モジュール構成)

3. 常時が、『ベストエフォート』なので、
   非常時にも『ベストエフォート』でサービス提供
   - 品質は低くても、サービスは継続

インターネットの仕組み

1. **ネットワーク の ネットワーク**
   a. 情けは人の為ならず。(他人の小包も配送)
   b. 参加者のすべてが 所有者・運用者

   ➔ あなたのものは私のもの、私のものはあなたのもの

2. **「デジタルの小包」を目的地まで 配送する。**
   a. どんな もの(コンテンツ)でも 運べる。
   b. どんな 伝送媒体でも 使える。
   c. 隣人に「デジタルの小包」を渡したら、その小包のことは忘れる。

ベスト・エフォート

物流システム の仕組み

1. ネットワークのネットワーク

(*) 『物流システム』で、「デジタルの小包」は、
「コンテナ」、「パレット」に対応する。
(*) 「人」も、「デジタルの小包」と 同じことができる。

➔ **あなたのものは私のもの、私のものはあなたのもの**

2. 「荷物」を、目的地まで 配送する。

a. どんな「荷物」でも、配送できる。

b. 「荷物」は、どんな 乗り物にも乗れる。

c. 「荷物」は、おろしたら、忘れて、
次の「荷物」のケア をする。

ベスト・エフォート

“物流”2つの大革命

19世紀以前 = 排他的個別網

本質は同じ
なんです!!

20世紀後半
(1) 物理的Sharing Economy

コンテナ
パレット
(1956年)

2020s =Cyber-First Sharing Economy

生産の大革命

20世紀終盤
(2) Cyber空間での
Sharing Economy

ディジタル小包
(=IP Packet)

Proxy、Overlay、Gatewayの違い

- Overlay： 論理的なネットワークを、あるプラットフォーム(e.g., IP ネットワーク) の上に仮想的に構築する。

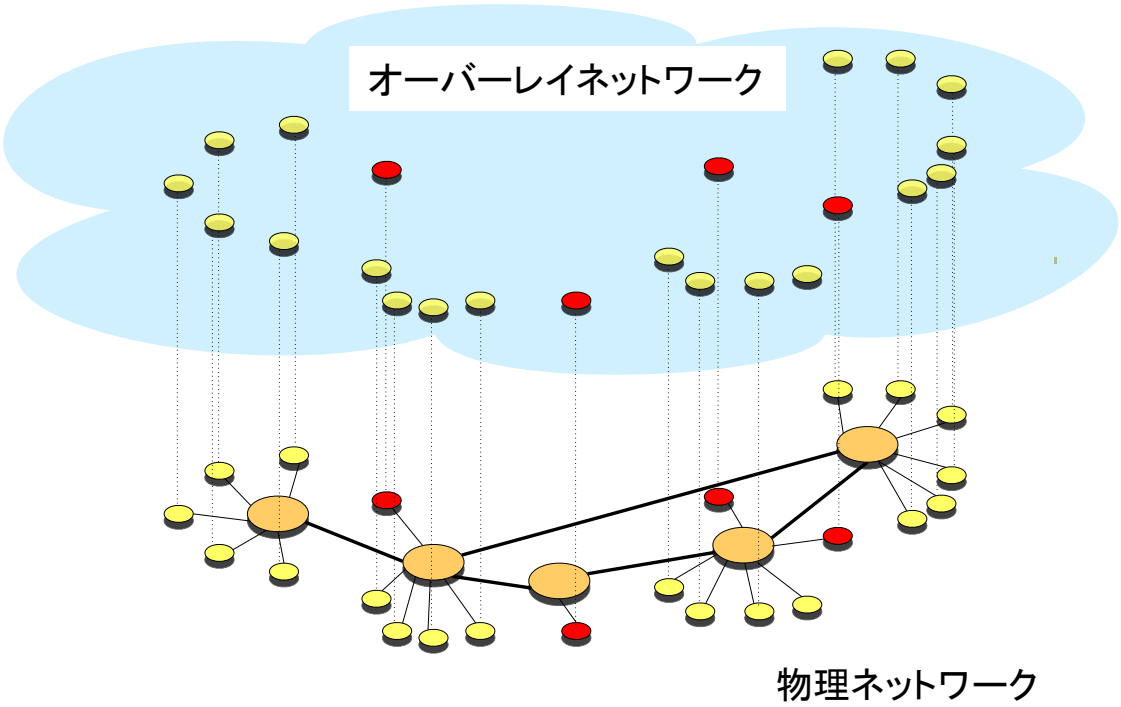
  ➔ システムの最適化が難しい。。。。。

- Proxy： “Transparent” な通信を、Intermediate Nodeで、なりすましをしたり、盗み見をしたりして、効率を向上させる。

  ➔ 効率向上は、データの移動パターンに大きく依存

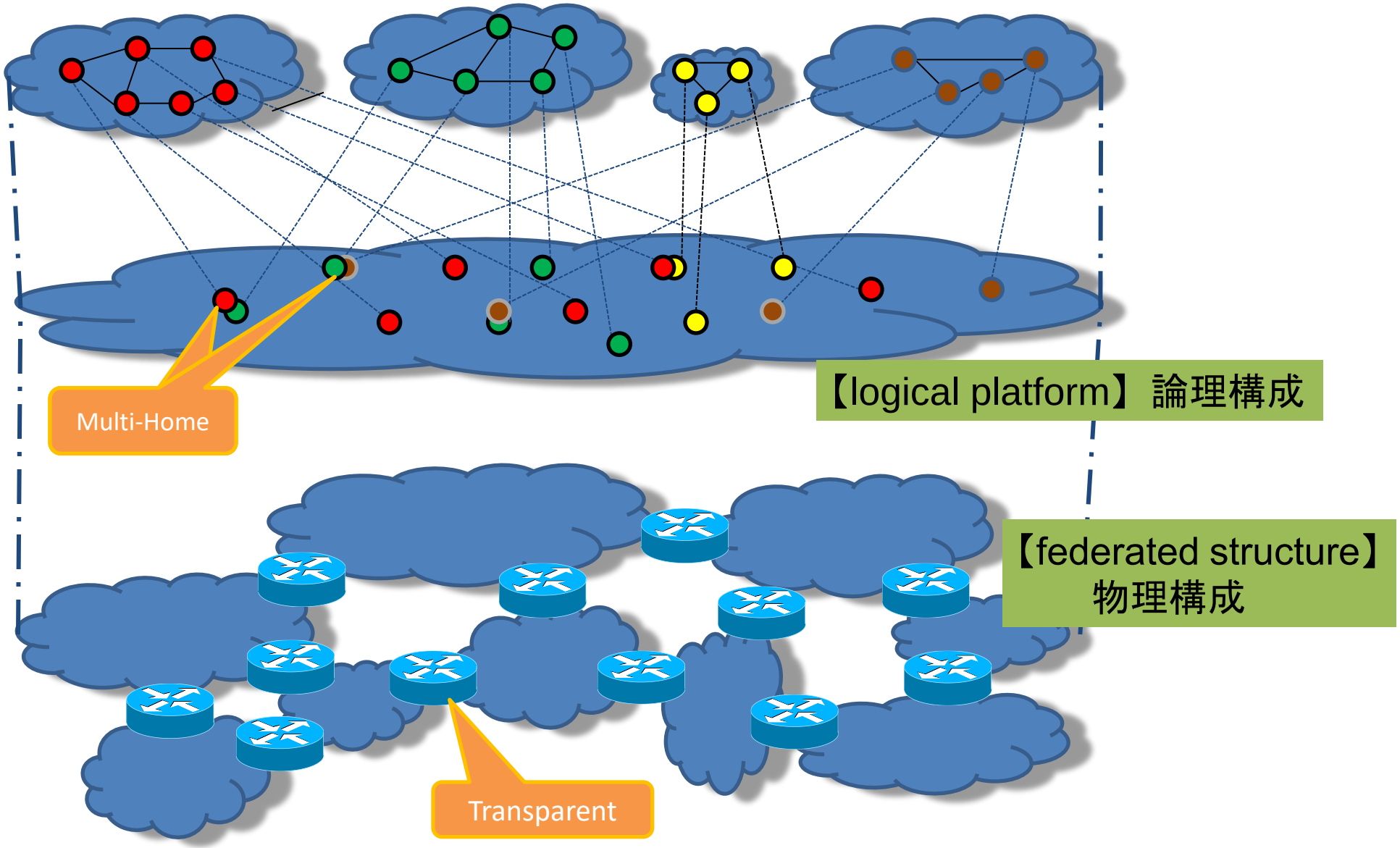
- Gateway： 違うプロトコル(=言語)を翻訳変換してあげる。

  ➔ もっとも、非効率。。。。大きなシステム同士を相互接続させる場合には、巨大な Stateful Machine を必要とする場合が多い。



オーバーレイネットワーキングの概念図

【Community over global infrastructure】



Proxy、Overlay、Gatewayの違い

- Overlay： 論理的なネットワークを、あるプラットフォーム(e.g., IPネットワーク) の上に仮想的に構築する。

  ➔ システムの最適化が難しい。。。。。

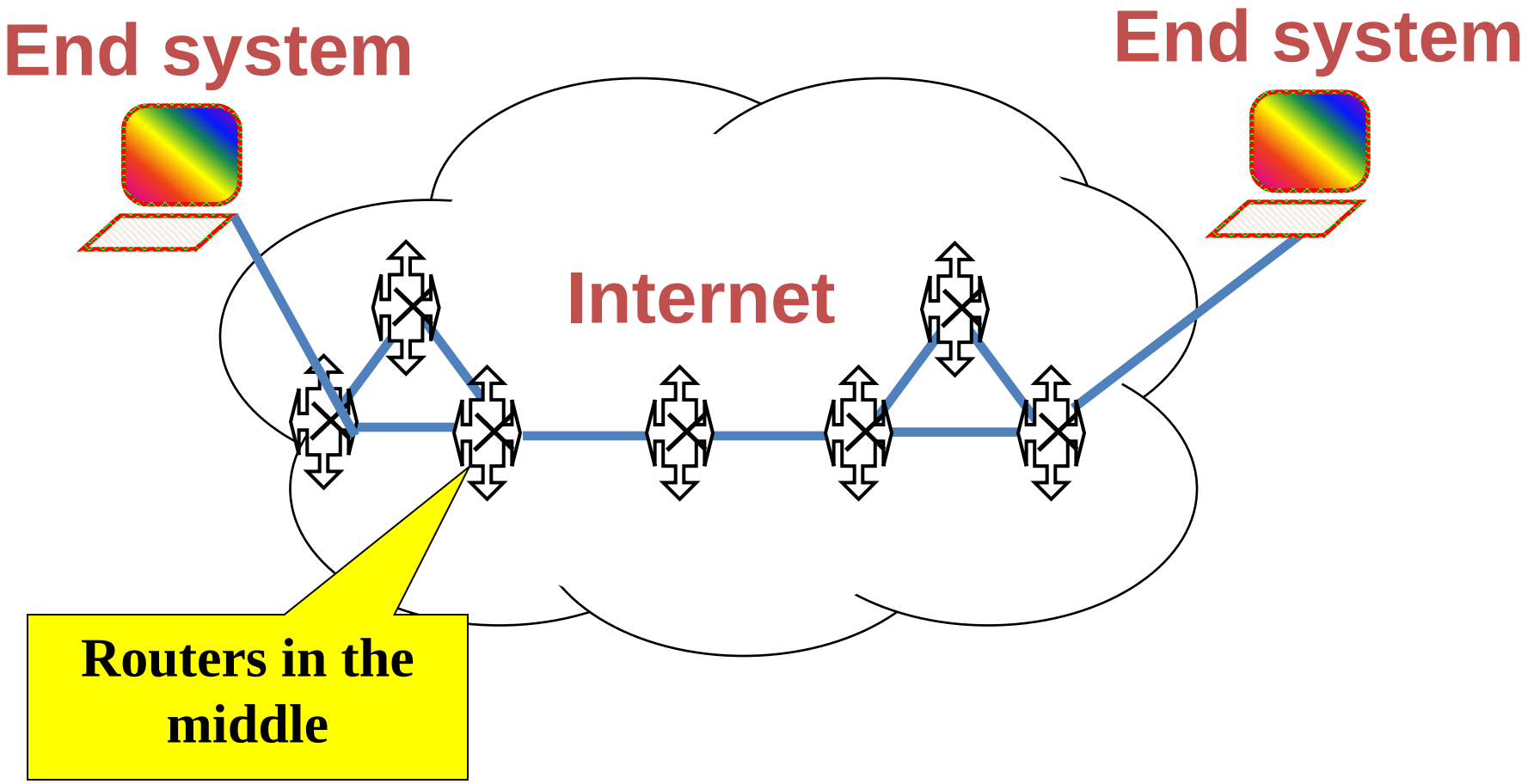
- Proxy： “Transparent” な通信を、Intermediate Node で、なりすましをしたり、盗み見をしたりして、効率を向上させる。

  ➔ 効率向上は、データの移動パターンに大きく依存

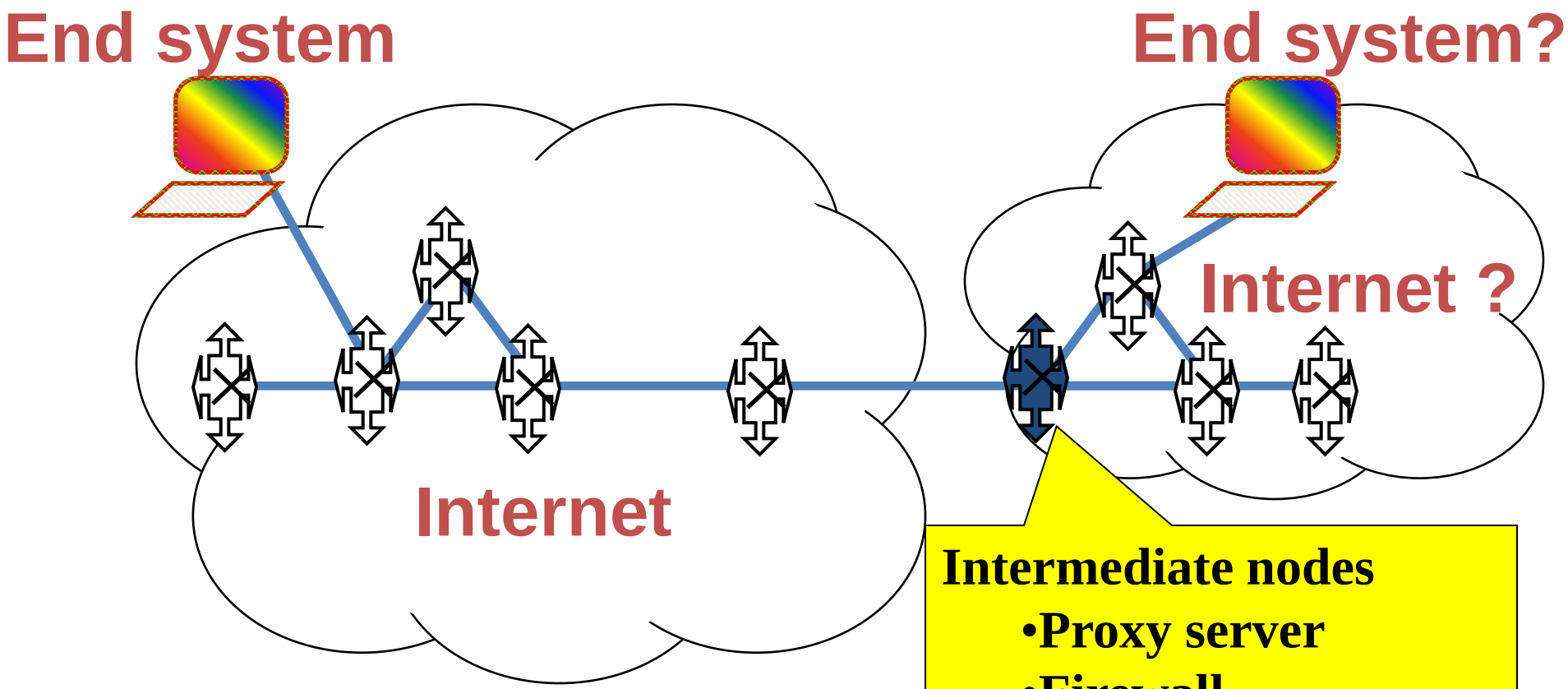
- Gateway： 違うプロトコル(=言語)を翻訳変換してあげる。

  ➔ もっとも、非効率。。。。大きなシステム同士を相互接続させる場合には、巨大な Stateful Machine を必要とする場合が多い。

“End-to-End Model”



ネットワークを分割したくなった?



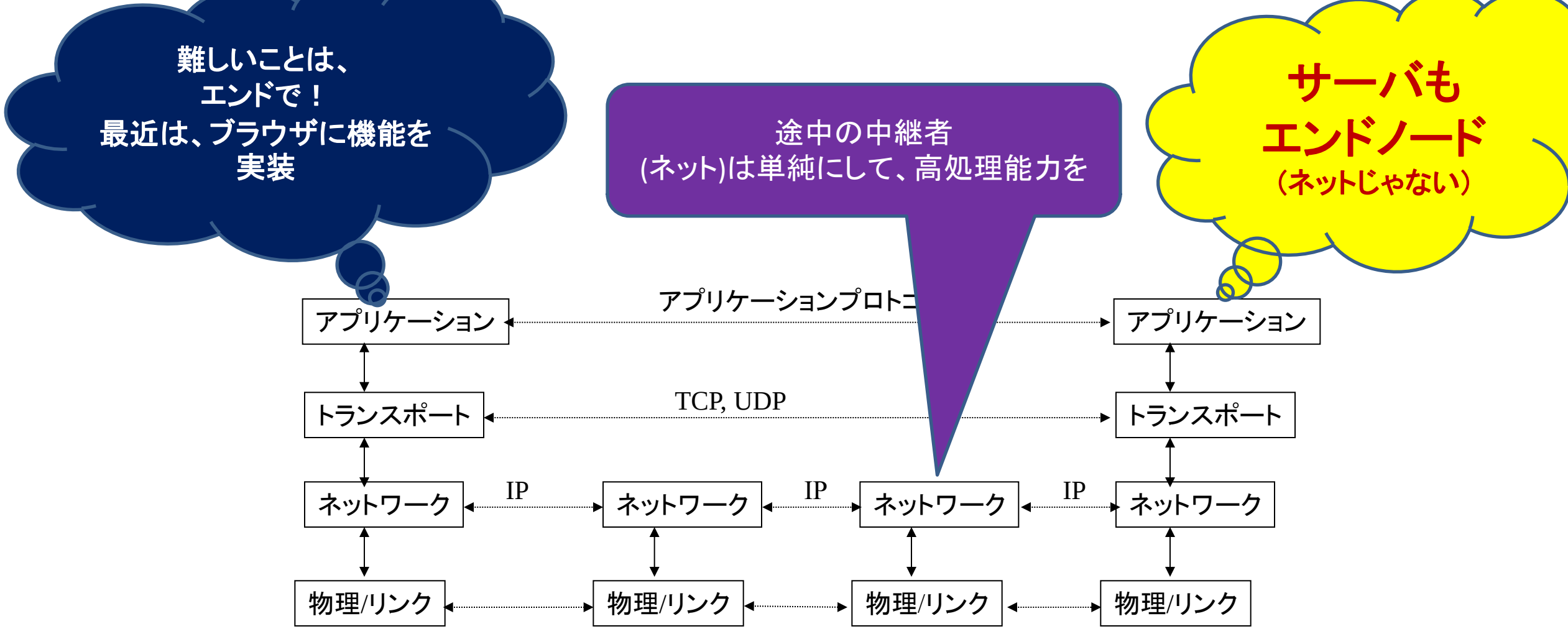
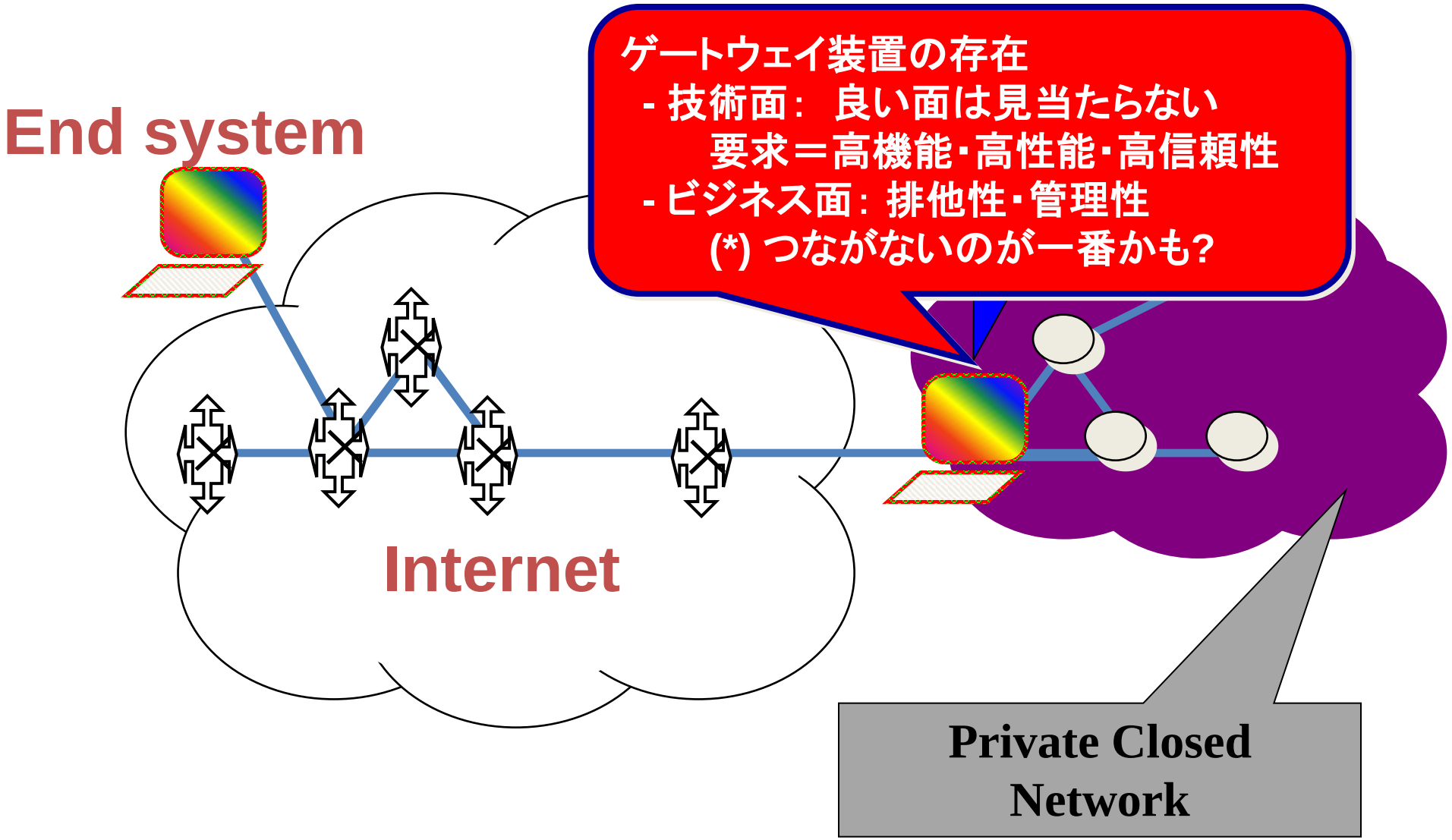


図1-12 TCP/IPの4レイヤモデル

大変な苦労を抱え込みます。

(*)でも“苦労”は、ビジネスチャンス?



Proxy、Overlay、Gatewayの違い

- Overlay： 論理的なネットワークを、あるプラットフォーム(e.g., IPネットワーク) の上に仮想的に構築する。

  ➔ システムの最適化が難しい。。。。。

- Proxy： “Transparent” な通信を、Intermediate Nodeで、なりすましをしたり、盗み見をしたりして、効率を向上させる。

  ➔ 効率向上は、データの移動パターンに大きく依存

- Gateway： 違うプロトコル(=言語)を翻訳変換してあげる。

  ➔ もっとも、非効率。。。。大きなシステム同士を相互接続させる場合には、巨大な Stateful Machine を必要とする場合が多い。

最近の 傾向

1. マルチプロセッサ型の計算機アーキテクチャの導入
   分散処理(機能分散、地理的分散)
2. キャッシュ技術の導入
   CDN、P2Pなど
3. 仮想化技術の導入
   Virtulization、Overlayネットワーク

処理負荷の分散手法

1. 水平分散
   - 複数のサーバに並列的に処理を実行させる。
   - 特定の機能ごとに、専門化したサーバを用意し、適切に Dispatch する。
2. 垂直分散
   - Proxy機能や Cache機能を提供し、クライアントに近い仮想ノード/分身ノードが、データ処理を分担する。
   - データの変更時の Consistency の維持が課題となる。

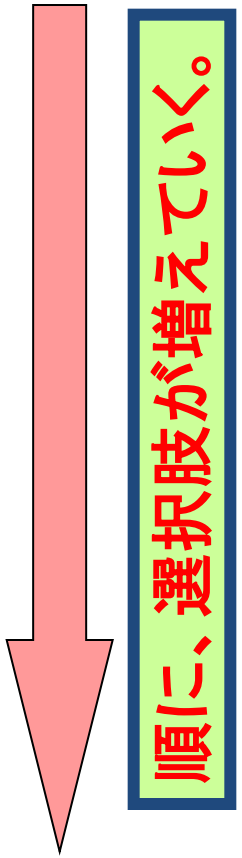
技術の上下関係。。。。

研究開発も同じ構造

- フレームワーク
  - どのような 方針でシステムを設計・動作さすか。
- アーキテクチャ
  - どのような 原理/技術 でシステムを構築するか。
- プロトコル、インターフェース
  - 具体的な、仕組み、決め事、アルゴリズム
- 実装
  - 具体的にどのように実現するか。

電話サービスを例にして。

研究開発も同じ構造



- フレームワーク
  - エンドーエンドに専用の通信パイプを提供する。
- アーキテクチャ
  - 空間分割 ➔ 時分割多重 ➔ パケット多重
- プロトコル
  - お願い(声) ➔ ダイヤル ➔ 小包の宛先札
- 実装
  - 手作業接続 ➔ 交換機 ➔ パケットスイッチ

OCP (Open Compute Project)に代表される、DevOps型の 設計・実装

汎用品・技術による カスタマイズ化
(e.g., マイクロ・ファブ化)
Driven by GAFA+BAT

USA: Google, Amazon, Facebook, Apple
China: Baidu, Alibaba, Tencent(WeChat)

Innovations in Data Center architecture

1. Open & Transparent , i.e., white xx

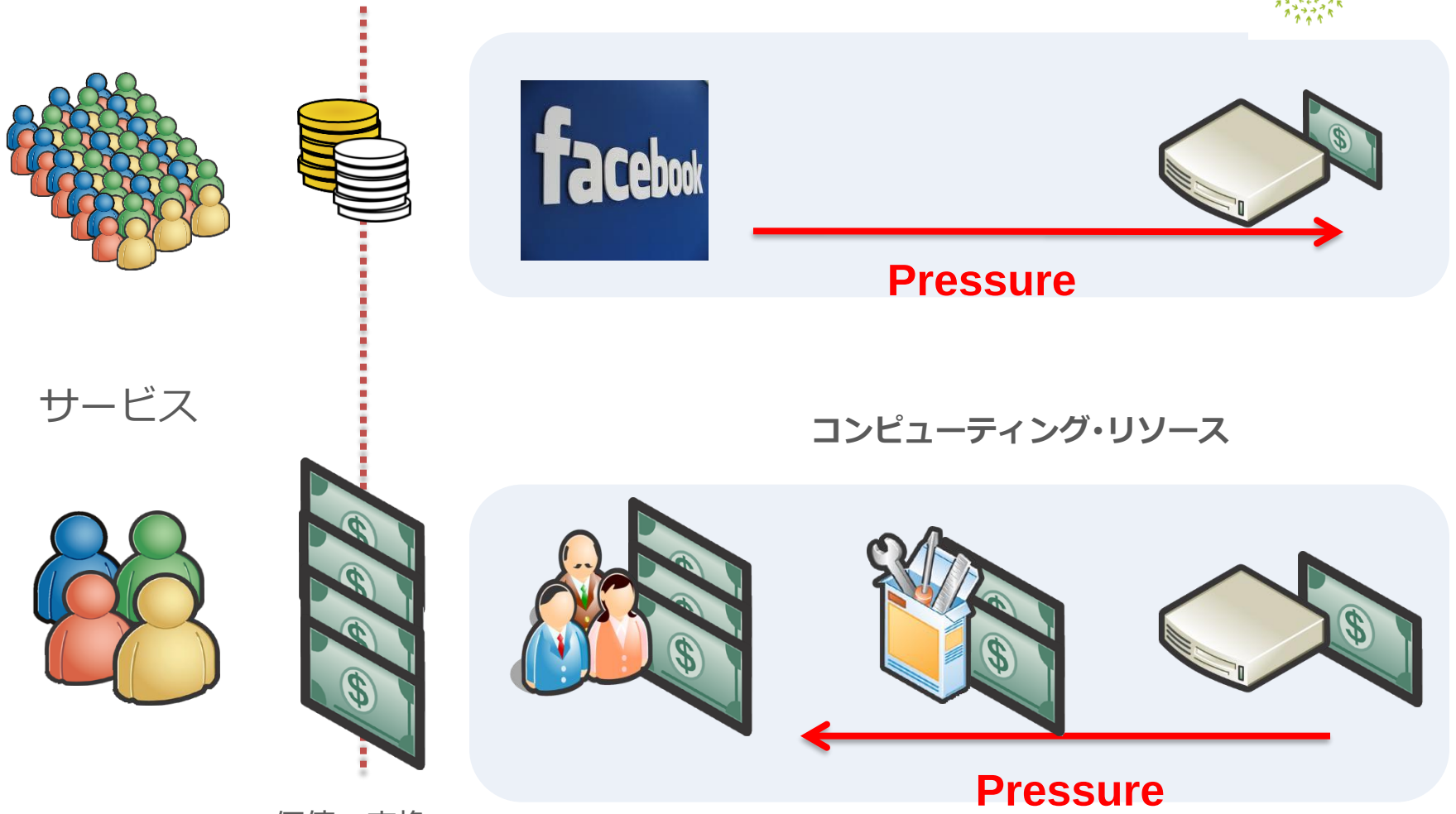
- ✓ HW
  - ➢ Chip, board, server, switch, router, Electric power, HVAC
- ✓ SW
  - ➢ Operating system, Middleware , Application

2. Data (Storage) Centric

- ✓ Big Data collection and analysis
  - ➢ Amount of data
  - ➢ Cross domain data integration
- ✓ **Processor/CPU centric ➔ Data/Storage centric**
  - ➢ **Migration overhead: data >> processing image**

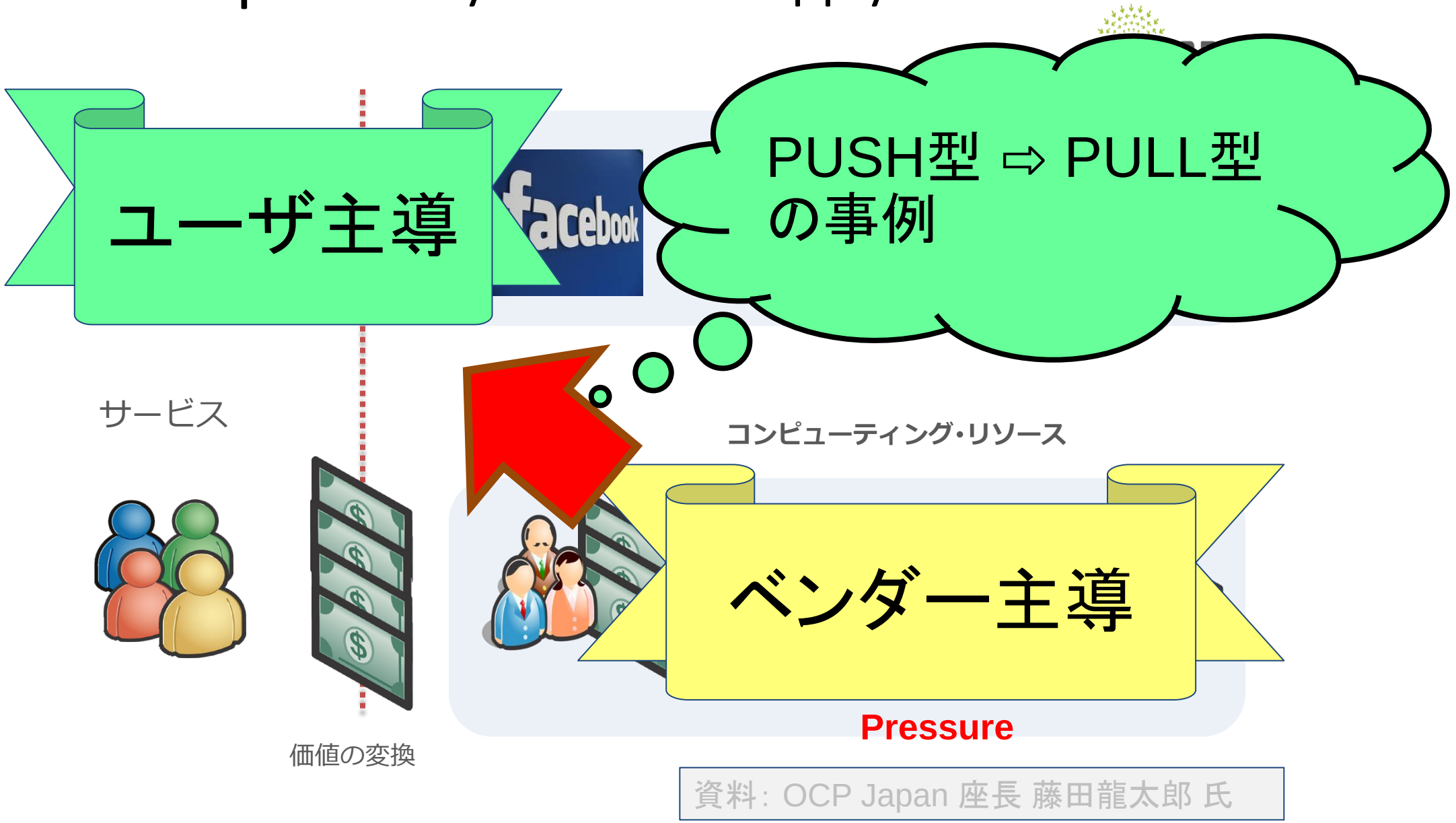
(*) Contribution of VM technology

DevOps: Eco-System New Supply-Chain



資料: OCP Japan 座長 藤田龍太郎 氏

DevOps: Eco-System New Supply-Chain



資料: OCP Japan 座長 藤田龍太郎 氏

PUSH型 から PULL型 へ

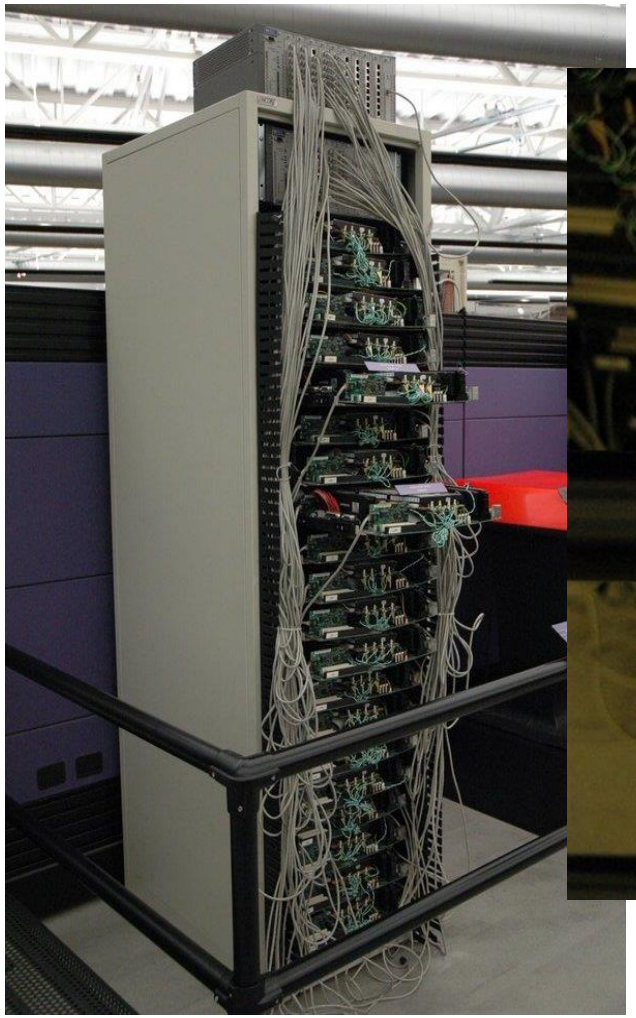
- **End-to-End 構造(i.e., stupid network)**

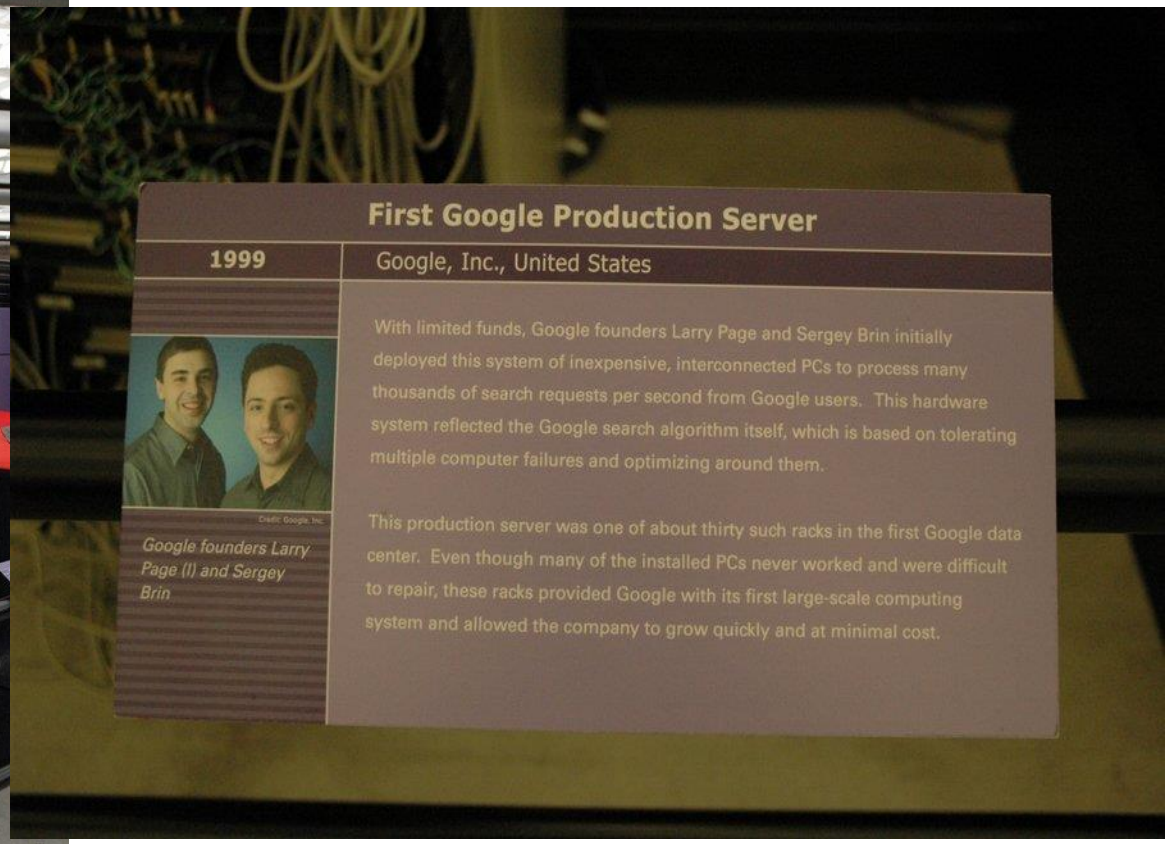
  これまでは、軽いEnd と 賢い&重い{あちら側の}End

- **ユーザー主導(vs ベンダー・プロバイダ主導)**
  - Industry 4.0 の {隠された/伝えられていない} 方向性

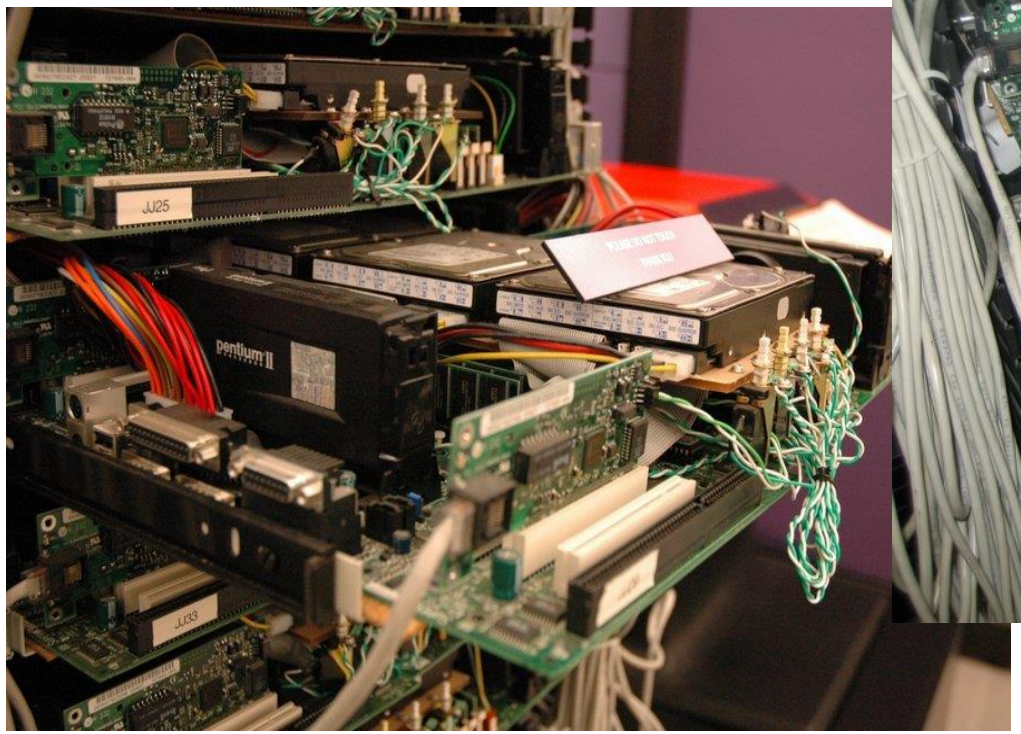
    Not Supply-Chain, but Demand-Chain
  - IoTデバイス は、Data-PUSH から **Data-PULL で、自分で大量のデータ処理を行う。**
  - 目的のIoT から dataをPULL して、目標のIoTデバイスに PUSH

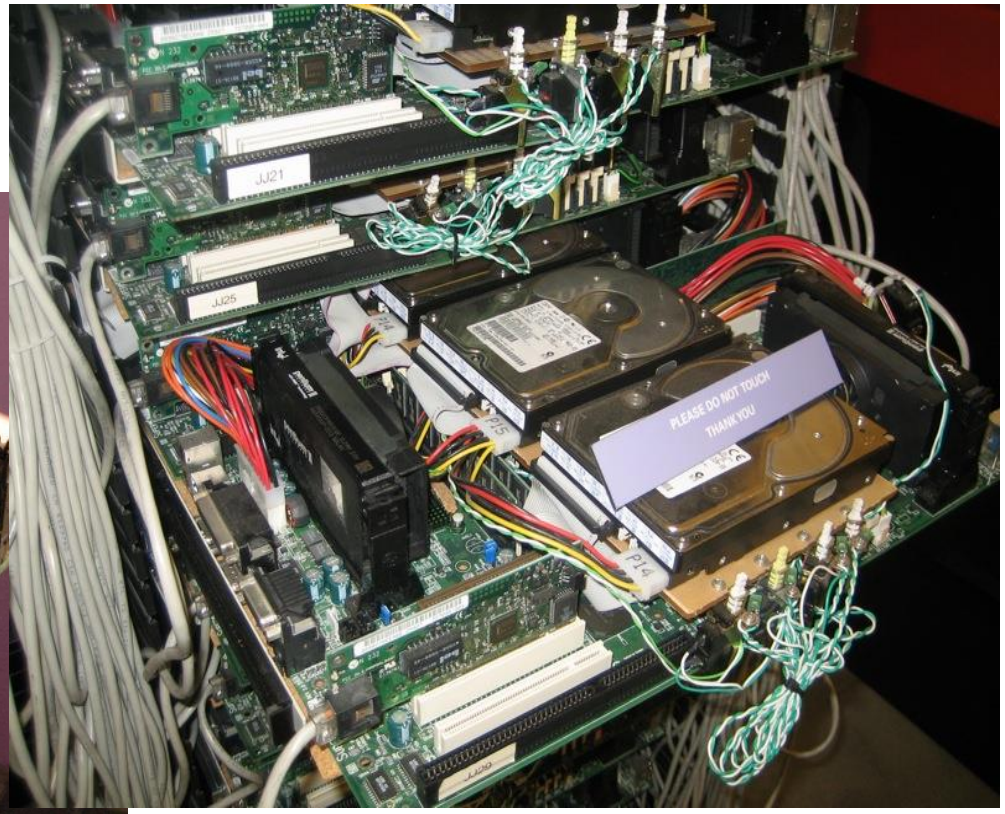
Google の 最初のサーバ (Computer Museum in California)

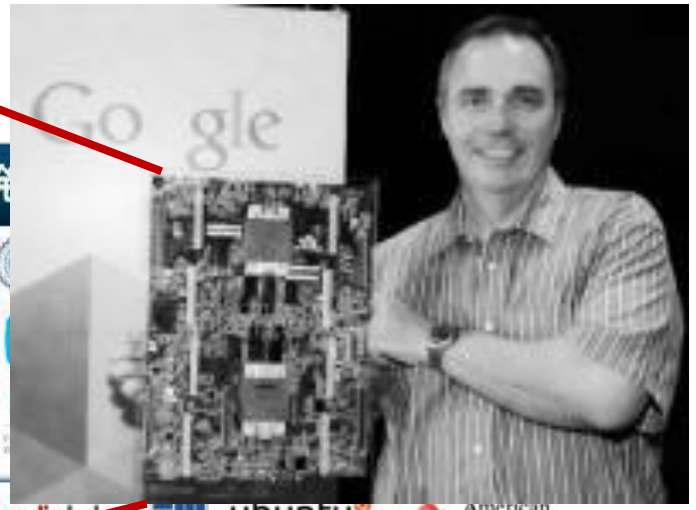




Google の 最初のサーバ
(Computer Museum in California)






Go gle


AVNET
Bull
CirrasScale
IBM
inspur 浪潮
OCF
ONE STOP SYSTEMS
STACK
ZTE中兴
ALTERA
HGST
numas
XILINX
Chip / SOC

Open COMPUTE SERVER

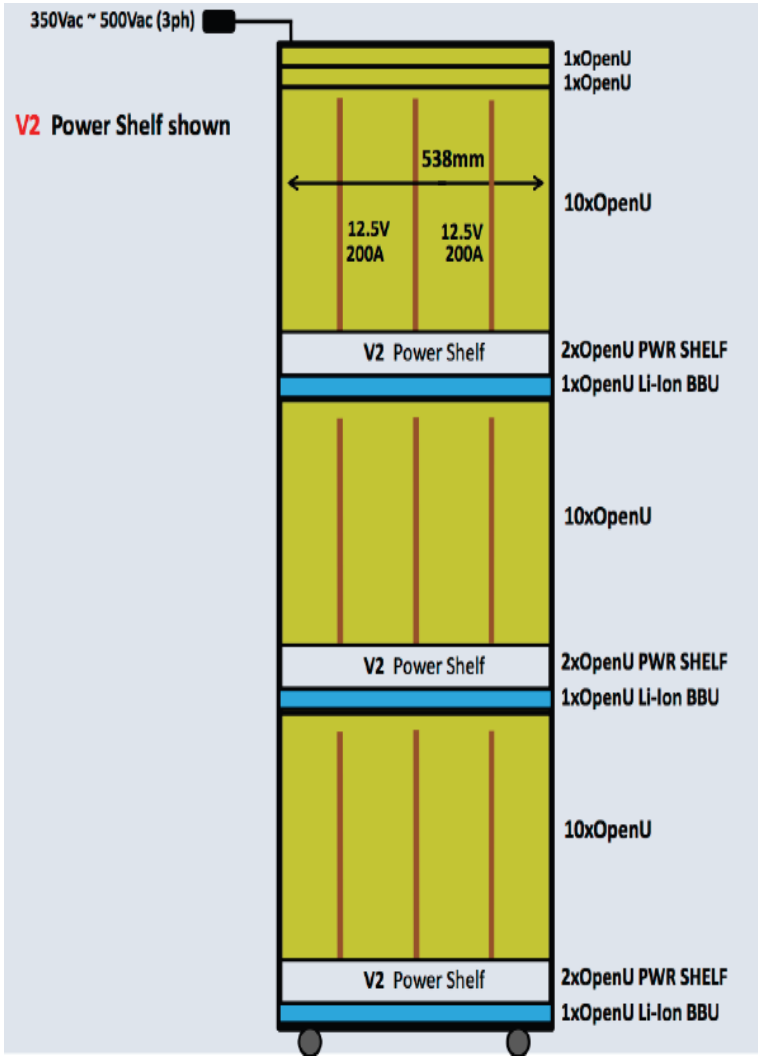
2U4ノード22インチ幅
Open Compute Project仕様
Quanta社製

2U3ノード 22インチ幅
Open Compute Project仕様
Quanta社製

12U24ノード 19インチ幅
Microsoft
Open Compute Project仕様
GigaByte社製

OpenRack外観

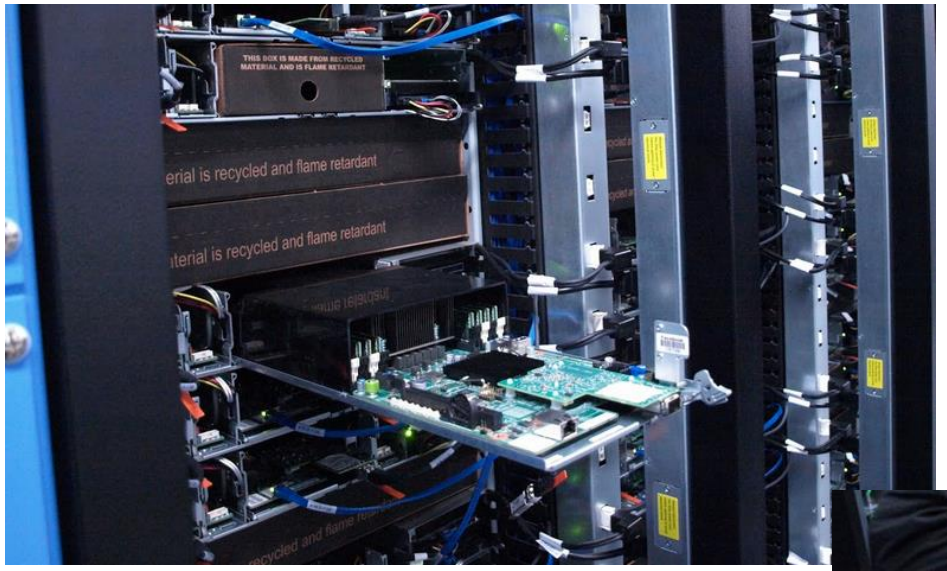
パワーゾーン単位に分割



Knox (Open Vault)

Cold Storage用サーバ

OCP のサーバーとラック



フタもなければ、
フロントパネルもない

ネジを１本も使わずに
キッティング

出典：http://wp.me/pwo1E-2Ku

資料：OCP Japan 座長 藤田龍太郎 氏

未来のインターネットに向けた開発戦略

シリコン、オプティクス、ソフトウェア、システムへの投資とイノベーション、
Converged SDN Transportへのアーキテクチャ変革により、インターネットの経済性を再定義する。

シリコン

25.6Tbps+ 高速・低消費電力

ビット単価・CAPEX/OPEXの削減

オプティクス

100G/400G 高速・長距離

装置点数・CAPEX/OPEXの削減

ソフトウェア

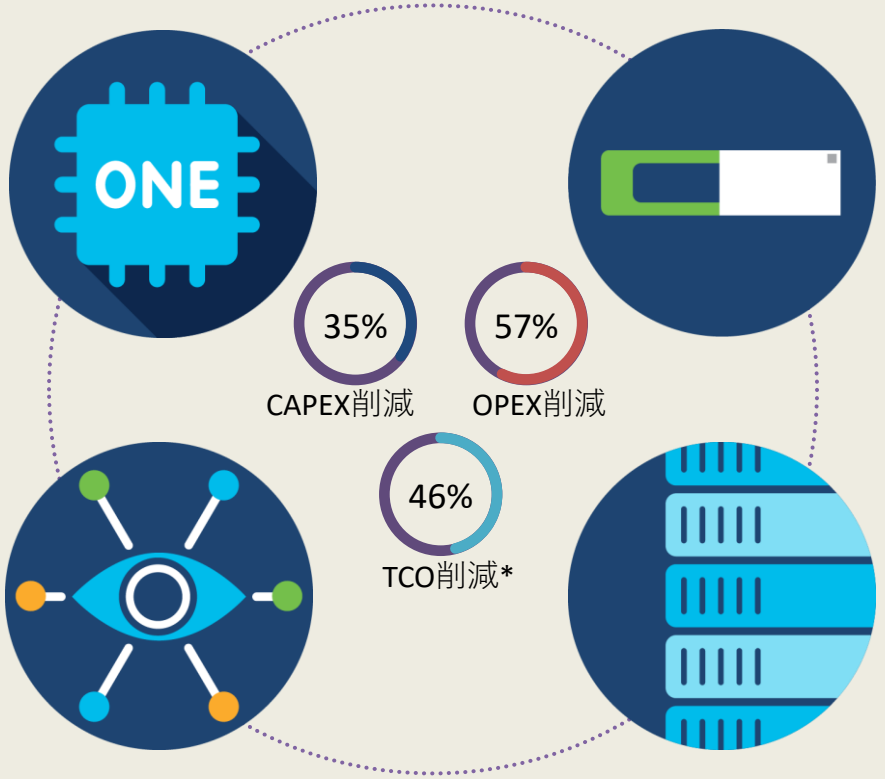
IOS-XR・オートメーション

運用コスト・OPEXの削減

システム

100G/400G ポートフォリオ

アクセス/エッジ/コア・CAPEX/OPEXの削減



Converged SDN Transport Architecture

IPと光伝送レイヤの統合・次世代トランスポートプロトコル・エンドツーエンドの自動化

ネットワークのシンプル化・TCOの削減

*ACG Research, 2021

イノベーションによる環境への取り組み

Cisco Silicon Oneのイノベーションにより、消費電力、スペース、サプライチェーンの圧倒的な効率化を実現

Cisco Silicon One
3.2Tbps–25.6Tbps

	従来のルータ 	Cisco 8201-32FH 	
ラックユニット	48	1	48x 削減
帯域	8Tbps	12.8Tbps	60% 向上
1 ラックあたりの帯域	0.166Tbps	12.8Tbps	77x 向上
消費電力	11,000W	288W	38x 削減
重量	907kg	14.09kg	64x 削減
データプレーンのシリコン	1,876個	1個	1,876x 削減

グローバル事例: Deutsche Telekom 様

設計からプロビジョニングまで、4ヶ月で完了して、環境目標達成に向けて大きく貢献

システム
パフォーマンス

260 Tbps

100Gbps
消費電力

92% 削減

システム
ラックスペース

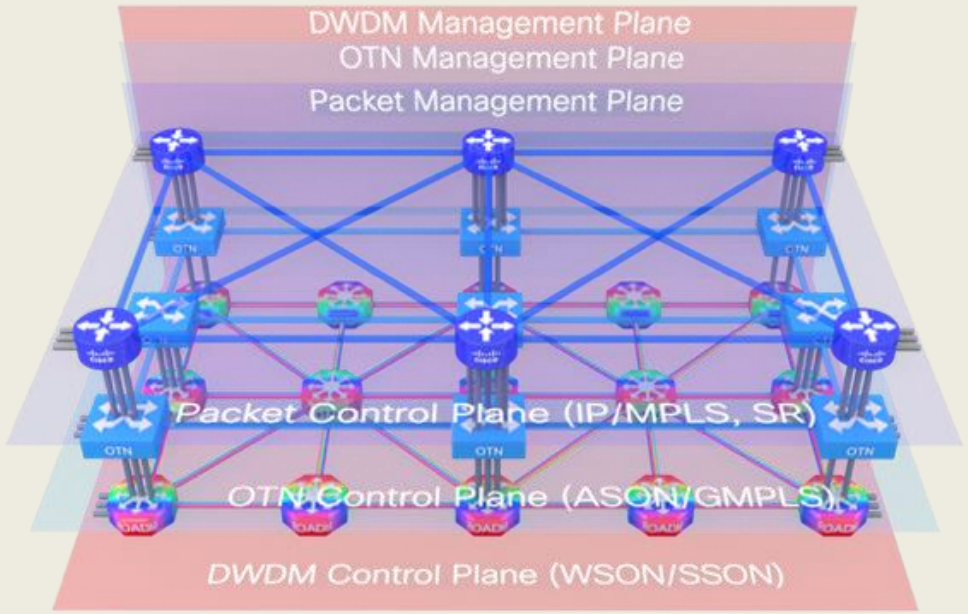
87.5% 削減

シスコとの協力により、最高の可用性とセキュリティを備えた
安全で安定したネットワークが提供されます

アーキテクチャ変革による環境への取り組み

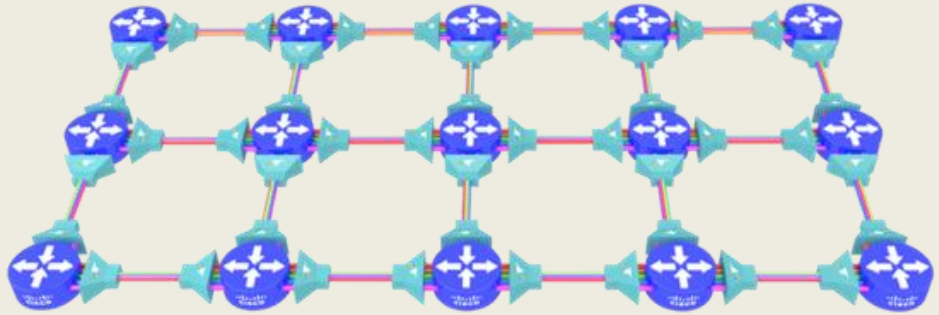
Routed Optical Networkingへのアーキテクチャ変革により、
消費電力、スペース、オペレーションの圧倒的な効率化を実現

現在：様々なレイヤー構成



将来：フラットでシンプル

IP+Optical | End-to-EndのIP化 | 自動化



35%
CapEx削減

57%
OpEx削減

46%
TCO削減

Python vs. Unicage by USP

work	Unicage by USP (Shell with C)	Python
Count number of raws in 20GB file	1	16.7
Sort 5GB file	1	48.89
Pick corresponding raws in 20GB file	1	More than 140 (not completed)
Transform to CSV format file from 10GB file	1	53.22

詳細資料：http://hiroshi1.hongo.wide.ad.jp/hiroshi/downloads/USP-Lab-Oct2024.pdf

Python vs. Unicage by USP

- 画像認識を行うニューラルネットワークをシェルコマンド（C言語で実装）とPythonライブラリで実装
- ニューラルネットワークの規模は隠れ層800-400の4層
- ニューラルネットワークの学習と画像認識テストを実施
- シェルコマンドの実装を1とした比較割合を以下の表

	シェルコマンド（C言語）	Pythonライブラリ
学習とテスト処理の合計時間	1/2	1
最大メモリ使用量	1/16	1
平均メモリ使用量	1/33	1

詳細資料：http://hiroshi1.hongo.wide.ad.jp/hiroshi/downloads/USP-Lab-Oct2024.pdf

NN比較検証: 数値検証(1)

- 画像認識を行うNNをシェルコマンド(C言語で実装)とPythonライブラリで実装
- NNの規模: 4096-800-400-10の4層
- 学習と画像認識テストを実施
- 逐次学習(バッチ処理なし)
- シェルコマンドの実装を1とした比較割合を以下の表

	シェルコマンド (C言語)	Pythonライブラリ (CPU)	Pythonライブラリ (GPU)
学習とテスト処理の合計時間	1	35.92	12.38
平均メモリ使用量 (RSS)	1	69.32	44.61
正答率	81.4%	82.5%	82.5%

NN比較検証: 数値検証(2)

- 画像認識を行うNNをシェルコマンド（C言語で実装）とPythonライブラリで実装
- NNの規模: 4096-800-400-10の4層
- 学習と画像認識テストを実施
- Pythonはバッチサイズ:150,エポック:300, シェルコマンドは未実装（逐次学習）
- シェルコマンドの実装を1とした比較割合を以下の表

	シェルコマンド(C言語):逐次学習	Pythonライブラリ(CPU)	Pythonライブラリ(GPU)
学習とテスト処理の合計時間	1	7.76	4.87
平均メモリ使用量	1	37.29	59.05
エネルギー使用量	1	56.99	89.50
正答率	81.4%	83.96%	70.73%

NN(大)比較検証: 数値検証(3)

- 画像認識を行うNNをシェルコマンド（C言語で実装）とPythonライブラリで実装
- NNの規模: 4096-8000-4000-10の4層
- 学習と画像認識テストを実施
- Pythonはバッチサイズ:150,エポック:300, シェルコマンドは未実装（逐次学習）
- シェルコマンドの実装を1とした比較割合を以下の表

	シェルコマンド（C言語）	Pythonライブラリ（CPU）	Pythonライブラリ（GPU）
学習とテスト処理の合計時間	1	17.70	0.12
平均メモリ使用量（RSS）	1	10.06	3.62
エネルギー使用量	1	23.53	0.28
正答率	80.02%	84.16%	73.86%