Empowered by Innovation NEC

# Webサーチエンジンと Big Data処理技術

日本電気株式会社（NEC）
情報・ナレッジ研究所
福島　俊一

# 講義のアウトライン

**第1部　大規模Webデータの検索技術**
**Webサーチエンジンはどんな仕組みで動いているの？**

**第2部　Big Data処理技術①　大規模な蓄積データ処理技術**
**大量に溜まったデータを効率良くさばくコツは？**

**第3部　Big Data処理技術②　大規模な実世界センシングデータ処理技術**
**大量に流れ込んでくるデータを溜めずにさばくコツは？**

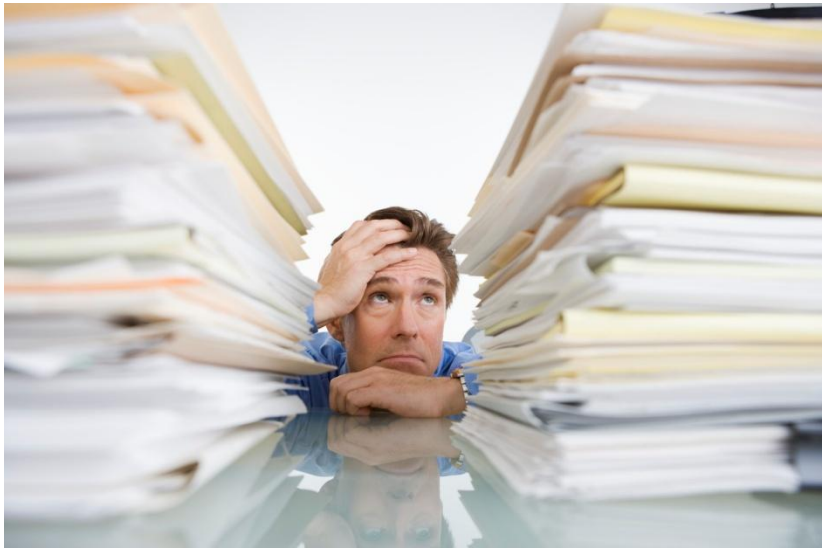
**第4部　Big Data処理技術③　大規模な実世界映像データ解析技術**
**カメラに見守られるのはSF世界だけの話なの？**

第1部　大規模Webデータの検索技術

# Webサーチエンジンはどんな仕組みで動いているの？

# 大量の資料からどうやって探しますか？

あちこちに山積みされた資料



M市場レポートは
どこだったかな？

Q事件の記事は
どこだったかな？

Z技術の資料は
どこだったかな？

・・・・・

# 大量の資料から探しやすくするには

①集めて

索引

| | |
|---|---|
| M市場 | a-2, a-3, k-2, … |
| Q事件 | b-6, m-4, m-7, … |
| Z技術 | a-2, d-5, e-2, … |
| ・・・・・ | |


M市場レポートはどこだったかな？

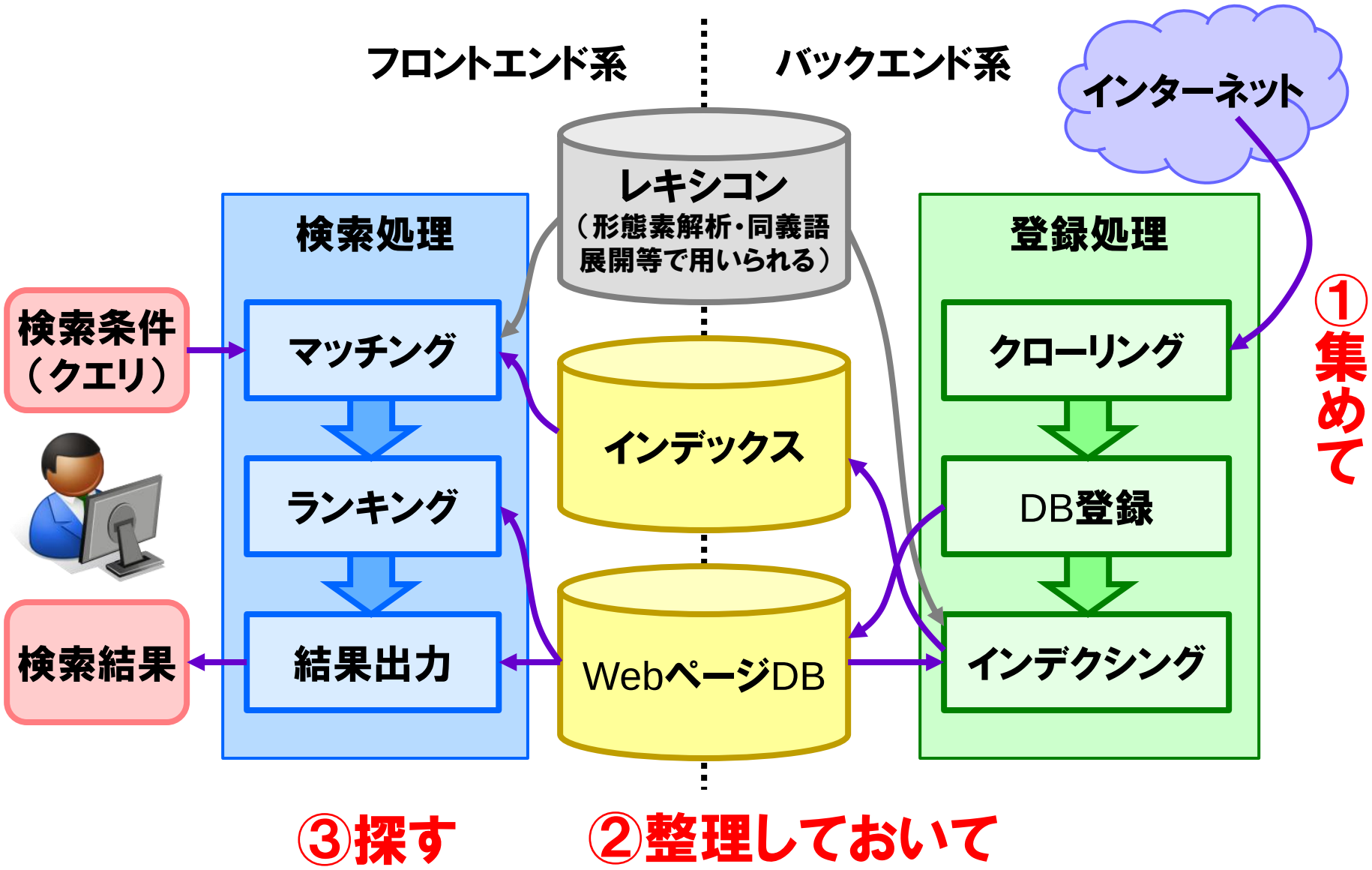
Q事件の記事はどこだったかな？

Z技術の資料はどこだったかな？

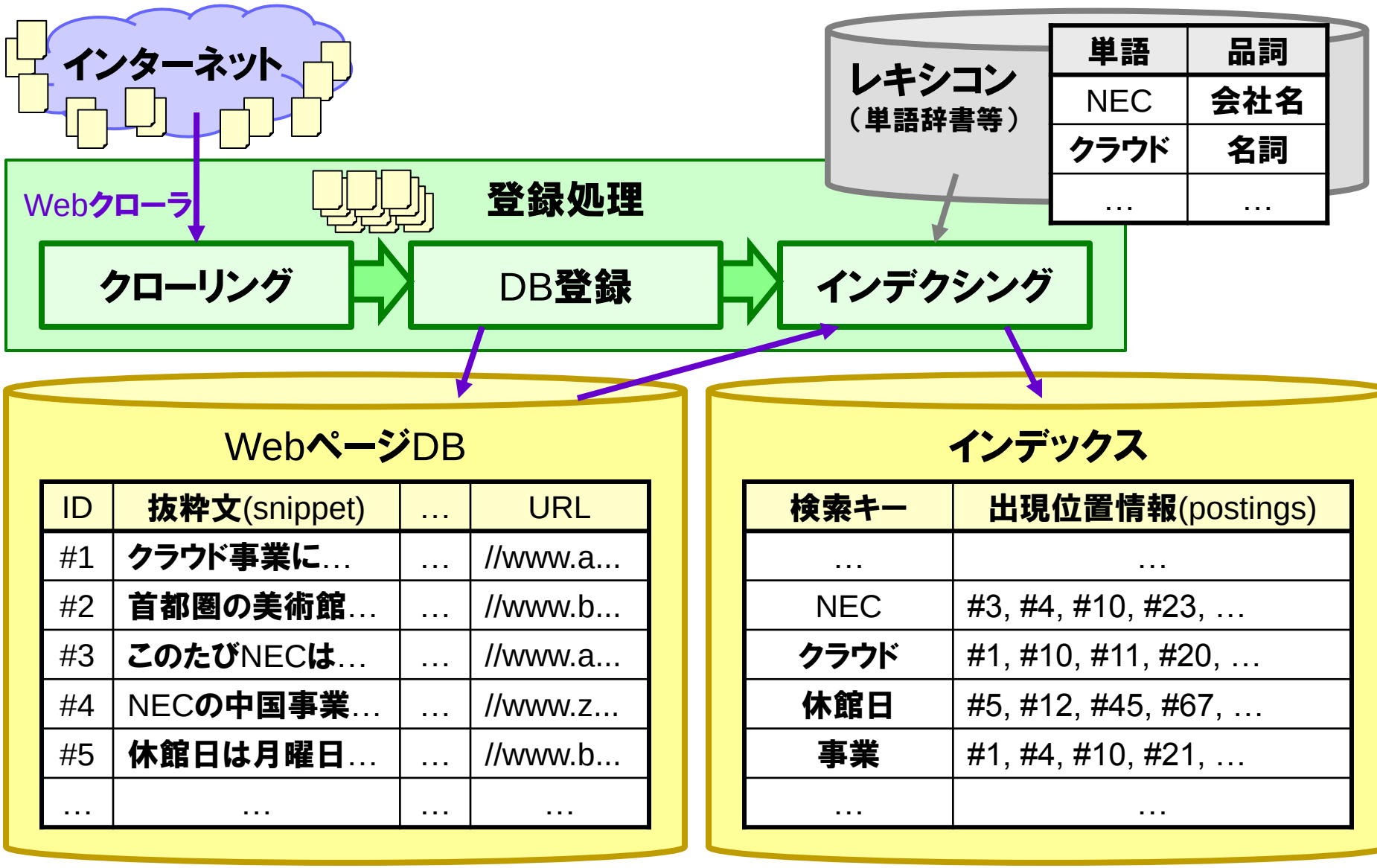
②整理しておいて

③探す ・・・・・

# Webページを探す ～ Webサーチエンジンの全体構成

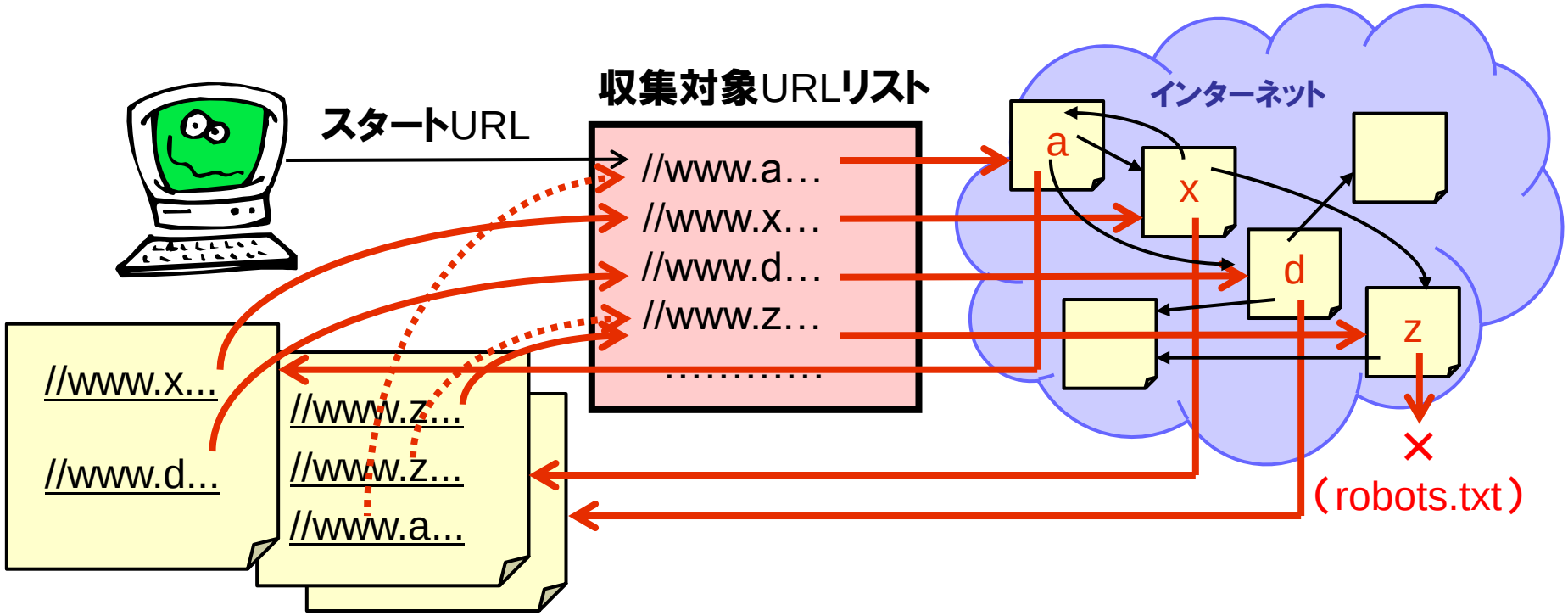


# バックエンド系の動作概要



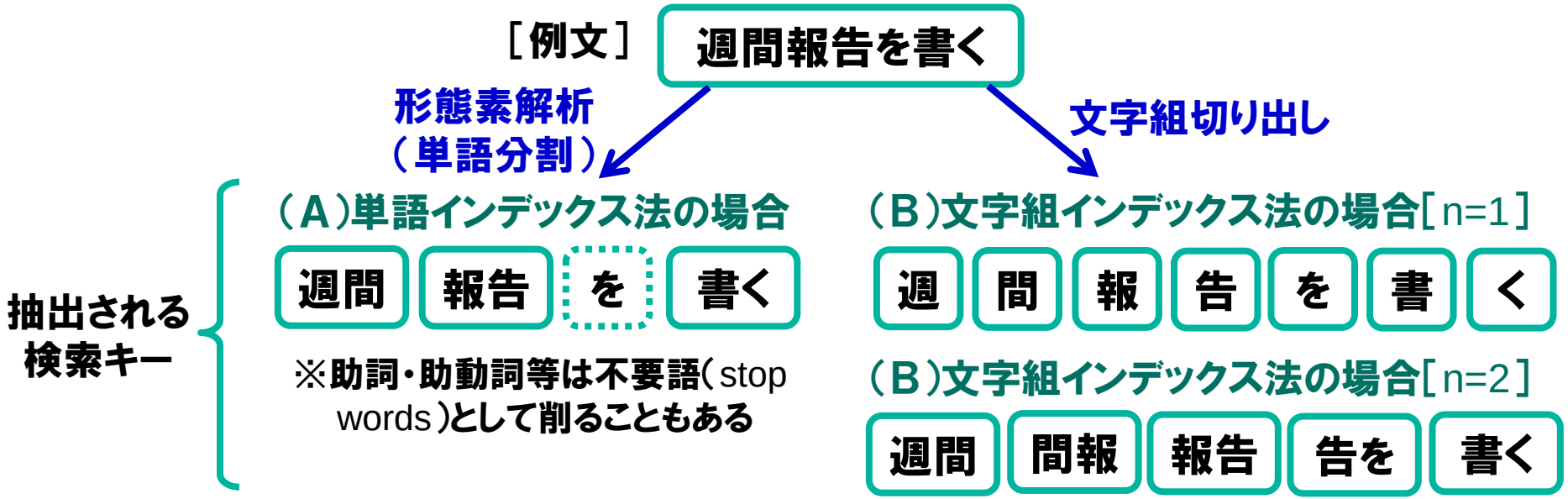
# ［バックエンド系］クローリング

- **Webクローラ（Webロボット）が、HTTPプロトコルを用いて自動的にハイパーリンクをたどりながらWebページ（HTML文書）を収集**
  - **ただし、収集先Webサイトに迷惑をかけないように、**robots.txt（**収集制限をサイト側の意向として記述可能**）**や収集間隔等のマナーは守る**

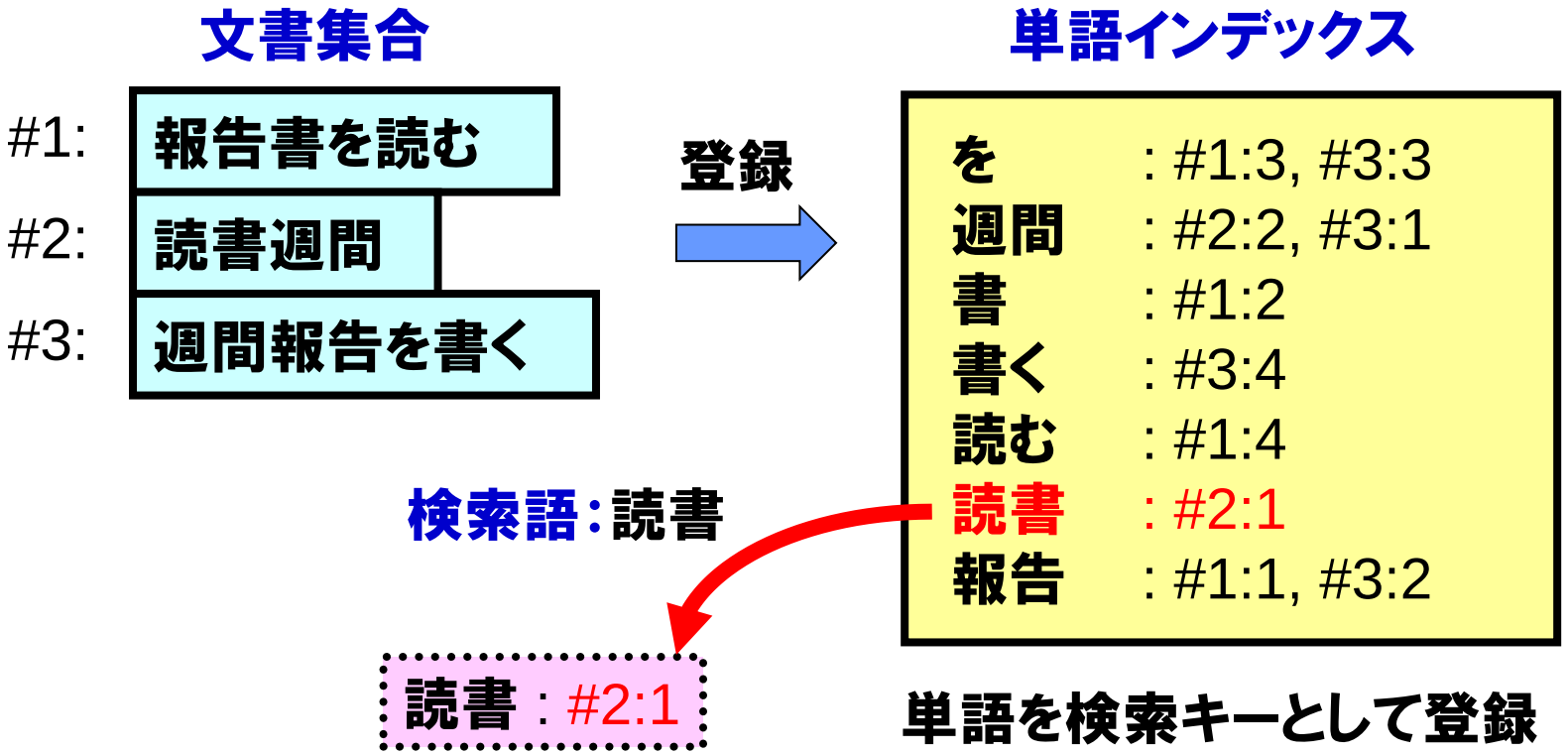


# ［バックエンド系］インデクシング

- 各Webページのフルテキストから、検索キーとその出現位置情報群（postings）のセットを生成して、フルテキストインデックスに登録
  - 検索条件が与えられた時にフルテキストを先頭からスキャンするgrep型のアルゴリズム（BM法・AC法等）では、大規模テキストに対して現実的でない
  - 大規模テキストの高速検索には転置型が必須
- 検索キーとして単語を用いる単語インデックス法、文字組（n-gram）を用いる文字組インデックス法がある



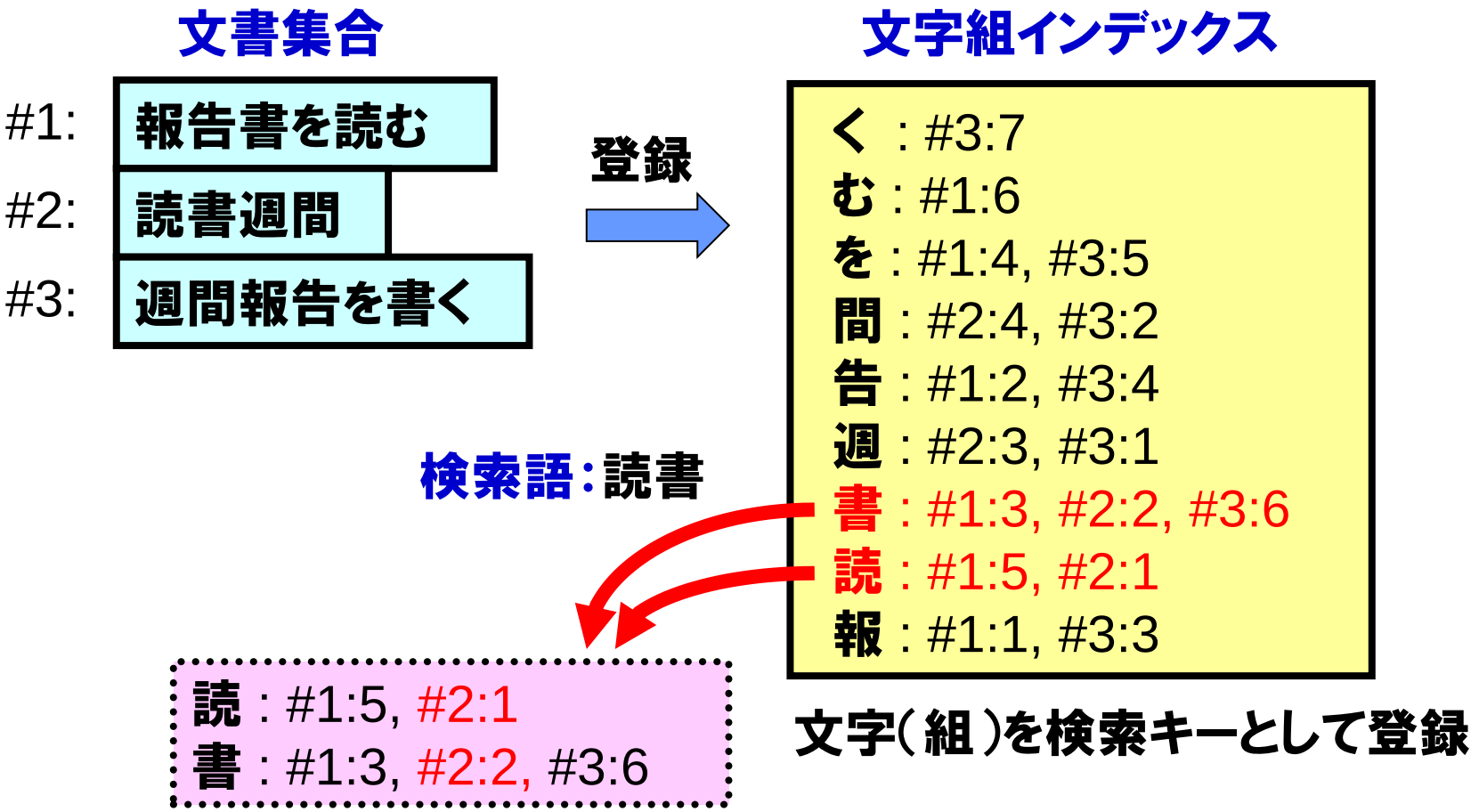
# （A）単語インデックス法




**出現位置情報群（postings）は**
**●文書ID（#d）のみ記録する方法と**
**●文書ID＋文書内位置（オフセット）の組（#d:p）を記録する方法の2通りがある**

#d:p　**文書#dのp単語目**

# （B）文字組インデックス法




文字（組）を検索キーとして登録

※文書内位置（オフセットp）の情報を持たない
場合は#1もヒットしてしまう（ノイズ発生）

#d:p　文書#dのp文字目

# 単語インデックス法 vs. 文字組インデックス法

## 検索精度（振る舞いの違い）

- **文字組インデックス法は単語の部分一致によるゴミが発生**
  【例】「京都」が「東京都」にヒット　⇒理由はわかりやすい
- **単語インデックス法は形態素解析の誤りや曖昧性による洩れ・ゴミが発生**
  【例】「きょ/どる」（未知語の誤解析）、「数/学科」or「数学/科」（曖昧性）
  ⇒対処方法として複数通りの解析結果をインデックス化する手がある※

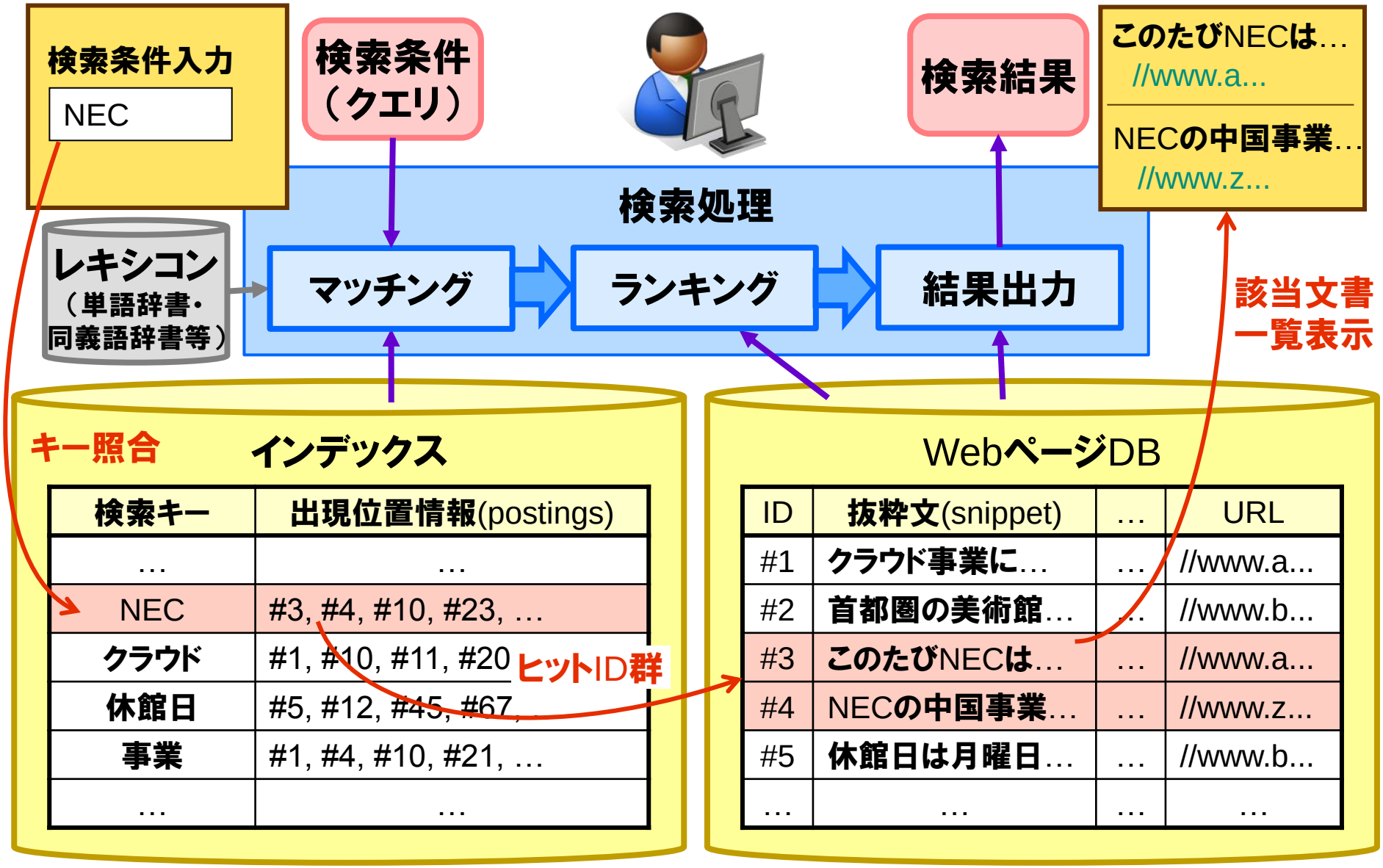
## 検索速度とインデックスサイズ

- 傾向として、postingsの件数が少ない方が検索が高速
  - 文字組インデックス法はnを大きくすると高速になる（インデックスサイズは激増）
  - 単語インデックス法は上記※で速度が若干低下（インデックスサイズも増加）
- **通常、単語インデックス法の方がインデックスサイズが小さく、検索も高速**

## インデックス法の選択

- 特許検索等での洩れを嫌い、**日本では文字組インデックス法が好まれた**
- 検索速度・インデックスサイズの優位性、検索精度問題への対処や多言語拡張の容易性等から、**国際的には単語インデックス法が主流になっている**
- オフセットは持たない方が高速・小容量だが、持った方が精緻な処理が可能

# フロントエンド系の動作概要



# ［フロントエンド系］ランキング

## 検索語との適合度

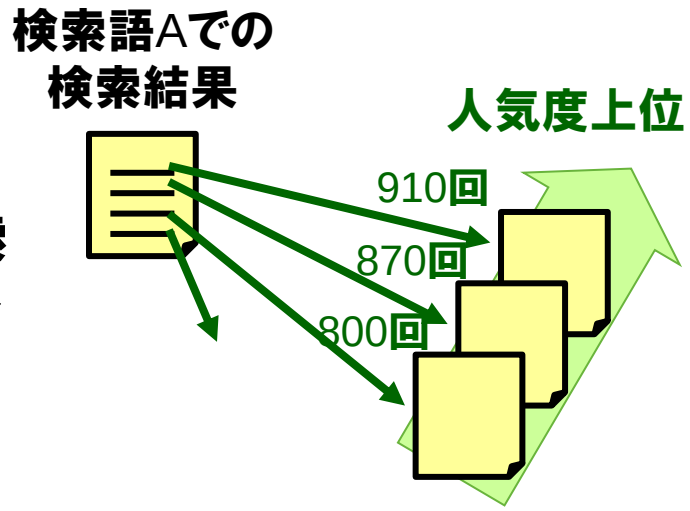
- 「そのページの主題に検索語が適合しているページほど重要」
- 情報検索における伝統的なランキング法（ベクトル空間モデル等）、HTMLタグを考慮したスコア（タイトルや見出し部分に検索語が出現すると加点等）

## ページ更新日時（新鮮度）

- 「新しいページほど重要」
- Webページの更新日時が新しい順にソート

## 参照履歴（人気度）

- 「多数の利用者が参照したページほど重要」
- 検索語とジャンプ先URLを記録しておき、検索語ごとにジャンプ回数の多い順にURLをソート

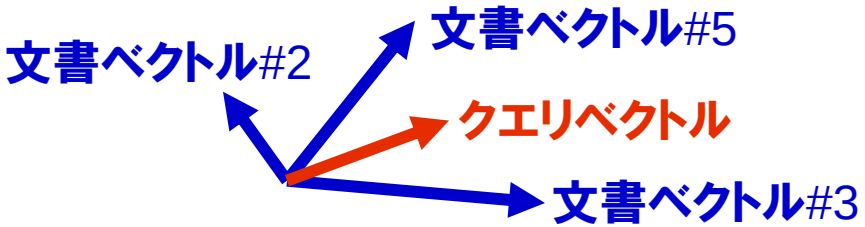



## リンク構造（引用度）

- 「多数引用されるページは重要」および「重要なページに引用されるページも重要」
- Google PageRankが代表的

# 【参考】ベクトル空間モデル

- **単語の種類を座標軸とした空間でクエリと文書をベクトル表現し、両者の類似度でスコア付ける方法**
  Bag-of-words
- **ベクトルの要素は各文書における各単語の重要性（tf・idf）**
  - Term Frequency $tf_{jk}$：**1文書に多数出現する単語は重要**
    **文書 $j$ における単語 $k$ の出現頻度 ÷ 文書 $j$ の長さ**
  - Inverse Document Frequency $idf_k$：**どの文書にも出現する単語は重要でない**
    **文書総数 ÷ 単語 $k$ の出現する文書数**




**クエリベクトル：**

(1.00, 0, 0, 1.00, 0)

**文書ベクトル：**

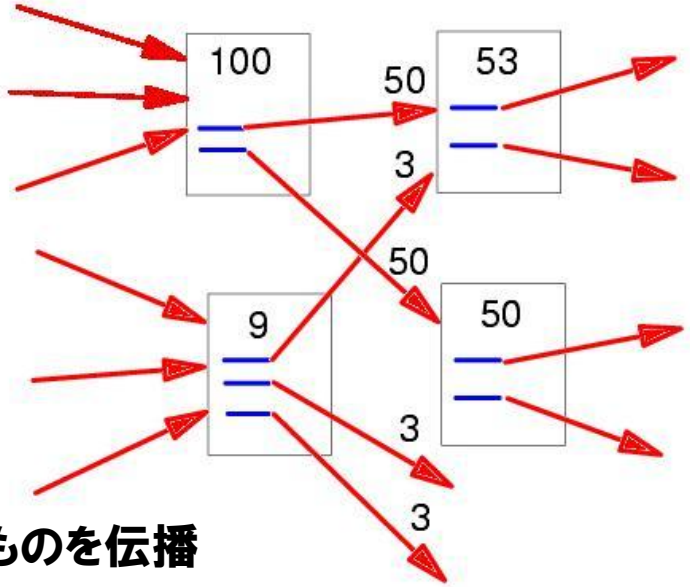
| | | |
|---|---|---|
| #1: | (0.17, 0.03, 0.38, 0.00, 0.00) | 0.17 + 0 + 0 + 0.00 + 0 = 0.17 |
| #2: | (0.00, 0.17, 0.67, 0.08, 0.00) | 0.00 + 0 + 0 + 0.08 + 0 = 0.08 |
| #3: | (0.67, 0.00, 0.00, 0.50, 0.00) | 0.67 + 0 + 0 + 0.50 + 0 = 1.17 |
| #4: | (0.00, 0.35, 0.00, 0.28, 0.56) | 0.00 + 0 + 0 + 0.28 + 0 = 0.28 |
| #5: | (0.17, 0.13, 0.00, 0.13, 0.00) | 0.17 + 0 + 0 + 0.13 + 0 = 0.30 |

**(イメージ, テキスト, 圧縮, 検索, 日本語)**

**ベクトル間の類似度計算**

**左例では類似度にベクトルの内積を用いたがコサイン値を用いることの方が多い**

# リンク構造を考慮したスコア計算

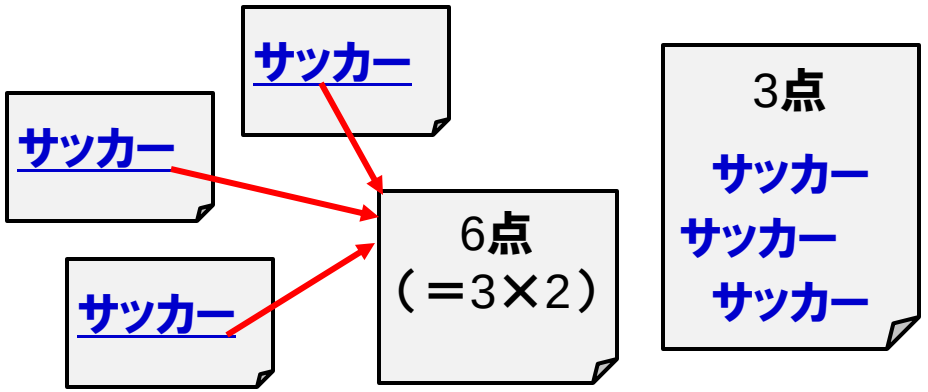



## 引用度スコア

- 「多数引用されるページは重要」および「重要なページに引用されるページも重要」
- ページのスコア値をリンク先に伝播させて再帰計算　⇒ Google PageRank
  - ページのスコア値を出ていくリンク数で割ったものを伝播
  - 入ってくるリンクのスコア値を合計

## リンク情報は、引用度スコアだけでなく、検索語との適合度スコアの計算でも有用　⇒アンカー文字列(リンク元文字列)をスコア計算で優先

- ページ内で検索語がヒットした場合よりも、アンカー文字列でヒットした場合に高得点を与える
- アンカー文字列は、ターゲットページの適切な要約であることが多いため



# Webの大規模化に伴う主要技術課題

## ① 大規模Webの効率的クローリング

- 大量Webページの収集に長期間かかるならば、検索できる情報は古くなる
- 同じサイトを短間隔でアクセスすると、サイトに大きな負荷がかかり大迷惑

## ② 大規模Webの高速インデクシング

- 大量Webページに対するインデックスの作成・更新に長時間かかるならば、収集してもインデックスへの反映が遅れる（反映できずに溢れる）

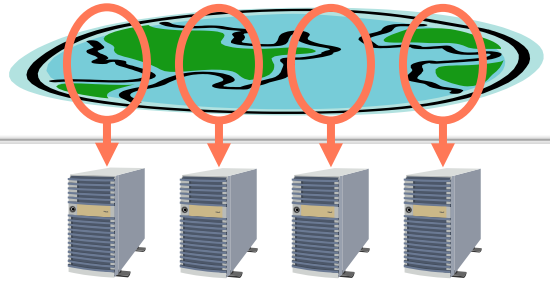
## ③ 高速参照可能な大規模Webインデックス構造

- インデックスの検索キー部・Postings部ともサイズが非常に大きくなり、オンメモリだけで処理することは不可能、検索速度はディスクI/O量に大きく依存

## ④ 大規模Webインデックスの高速並列検索

- インデックスに登録されるデータ量が増えるにつれて検索レスポンスは低下
- 利用者数が増えれば、同時に検索をかけたために待たされるケースが増加（スループットの低下）

# ①大規模Webの効率的クローリング



## 分散並列クローリング

- 複数台のマシンで並行してクローリング、かつ、1台のマシン上で複数のクローリングプロセスを実行、それらの間で収集範囲を分担

## 優先的クローリング

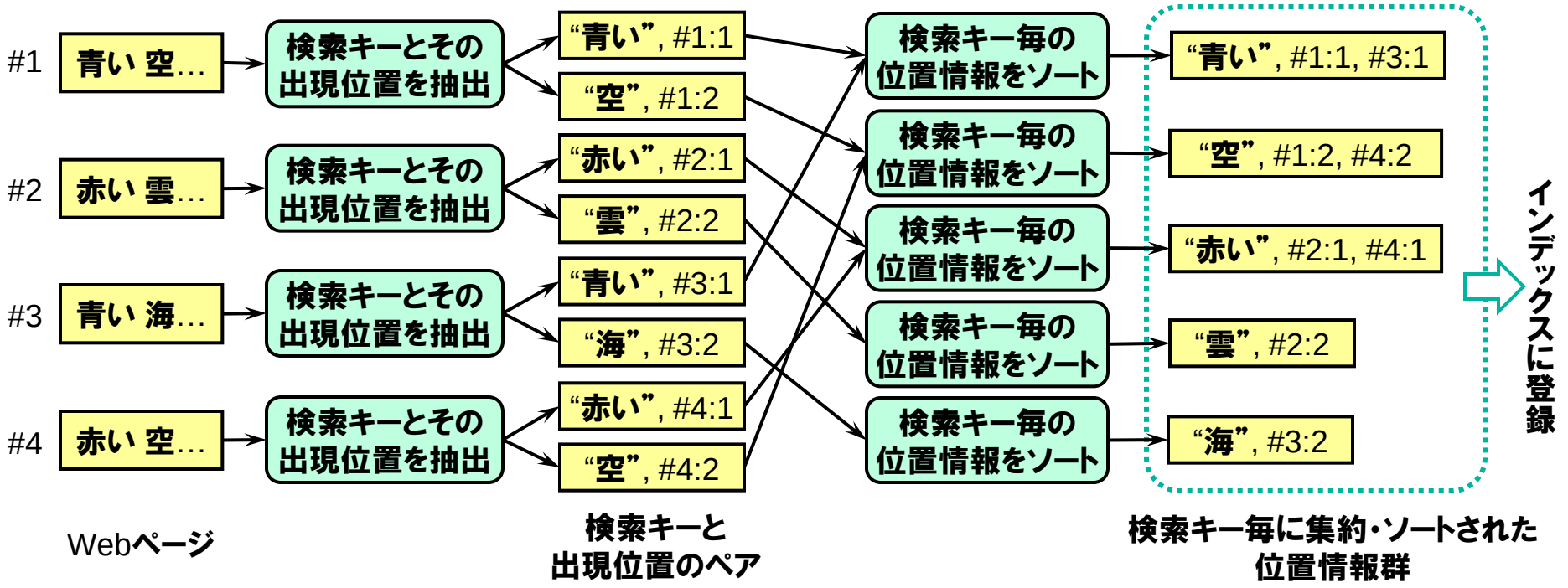
- 下記ファクタ等に基づいて、収集対象URLリスト中の優先順位を計算することで、収集価値の高いページを優先的にクローリング
  - 被リンク数あるいはPageRank（リンク関係から対象ページの重要性を判定）
  - URL（Webサイト内の階層の浅いページやインデックス的ページを判定）
  - アンカーテキスト（アンカーテキストから対象ページ内容を推測して関連性判定）
  - 更新頻度（更新頻度の低いところを頻繁に見に行くのは効率が悪い）

## 選択的クローリング

- 予め決めたWebサイトの範囲内のみクローリング
  - 例えば、特定の掲示板、更新頻度の高い新聞記事サイト等のみ頻繁に収集
- 特定のトピックやページタイプの範囲内のみクローリング
  - 例えば、求人情報のみ収集

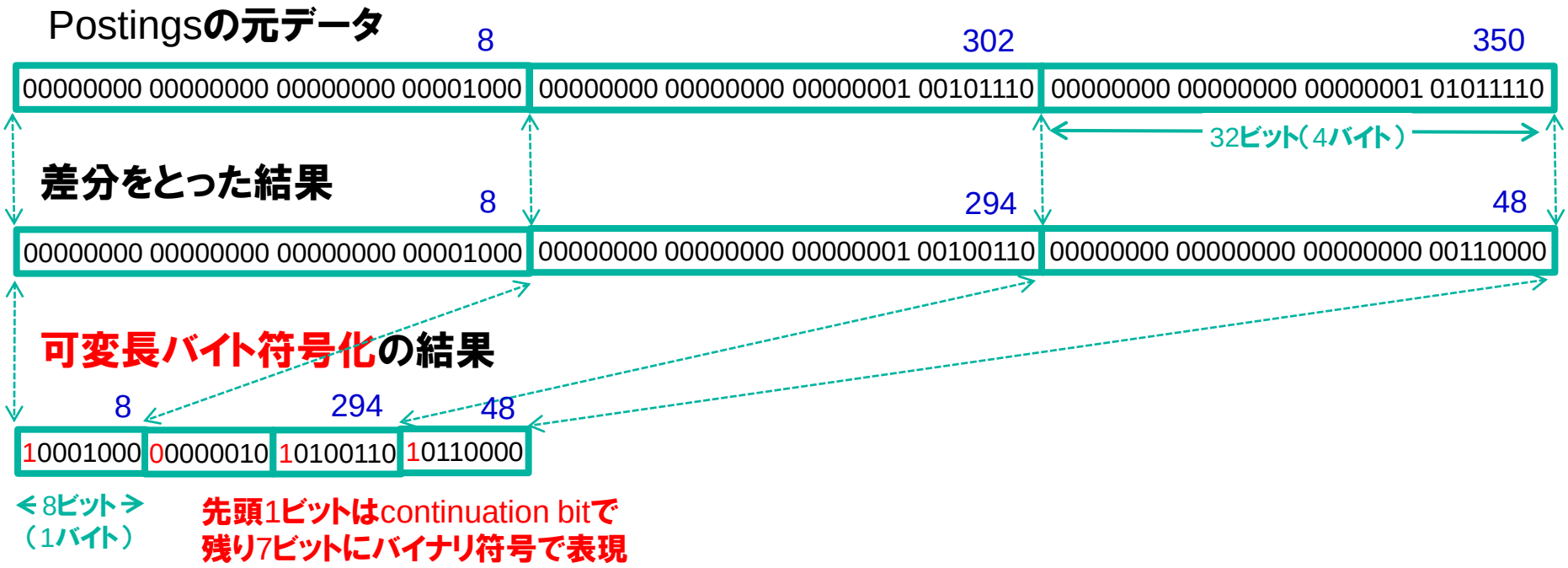
# ②大規模Webの高速インデクシング

- 大量Webページからインデックスに登録する情報を抽出・ソートするプロセスを分散並列化することで、インデクシングを高速化
  - 各Webページから検索キーとその出現位置を抽出するステップを分散並列実行（Webページ毎の処理は独立なので並列実行可能）
  - 異なるWebページから抽出された同じ値の検索キーをもつデータセットを1箇所に集約し、検索キー毎の位置情報群としてソートして、インデックスに登録



# ③高速参照可能な大規模Webインデックス構造

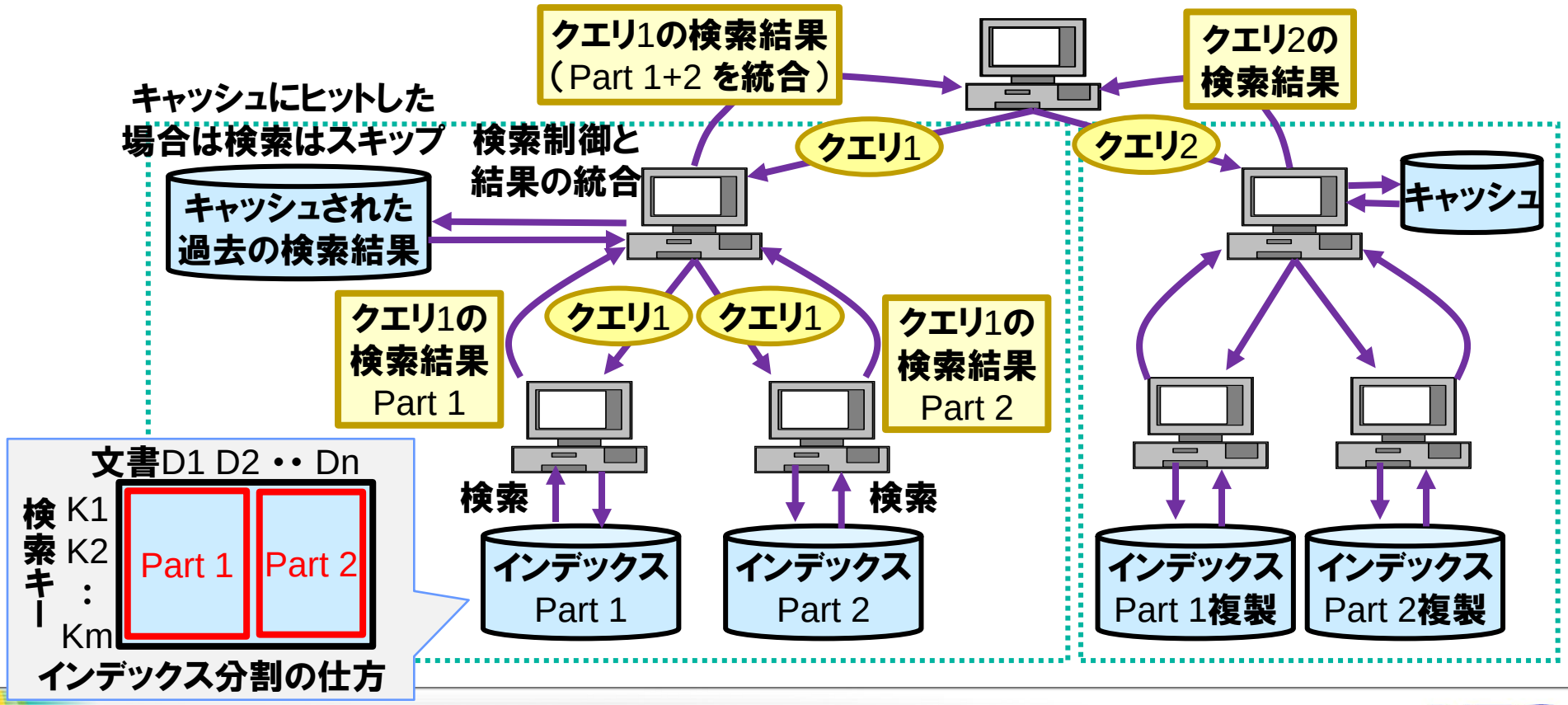
- **検索キー部：** 木構造データ探索アルゴリズム（例えば「B+木」等）を用いることで、ディスクI/O回数を削減
- **Postings部：** データ圧縮手法を適用することで、サイズを削減
  - Postings部は先頭からシーケンシャルに読み出されるだけだが、そもそもサイズが大きいため、データ圧縮がディスクI/O軽減に大きな効果がある
  - データ圧縮アルゴリズムは、圧縮率の高さに加えて、復号処理の軽さが重要



# ④大規模Webインデックスの高速並列検索

## データ量と利用者数(アクセス頻度)に対するスケーラビリティを実現

- インデックス分割によってレスポンス向上
  - 検索キー分割よりもPostings分割の方が負荷がバランスしてベター
- インデックス複製によってスループット向上



# 第1部のまとめ

## 大規模Webデータの検索技術

**Webサーチエンジンはどんな仕組みで動いているの？**

→ **①集めて　②整理しておいて　③捜す**

| | |
|---|---|
| ①集めて | Webページをクローリングして WebページDBに登録 |
| ②整理しておいて | インデックスを作成（インデクシング） |
| ③探す | クエリとインデックスをマッチングして結果をランキング |

### Web大規模時の課題への対応

- 大規模Webの効率的クローリング（分散並列クローリング等）
- 大規模Webの高速インデクシング（分散並列インデクシング）
- 高速参照可能な大規模Webインデックス構造（データ圧縮等）
- 大規模Webインデックスの高速並列検索（分散並列検索）

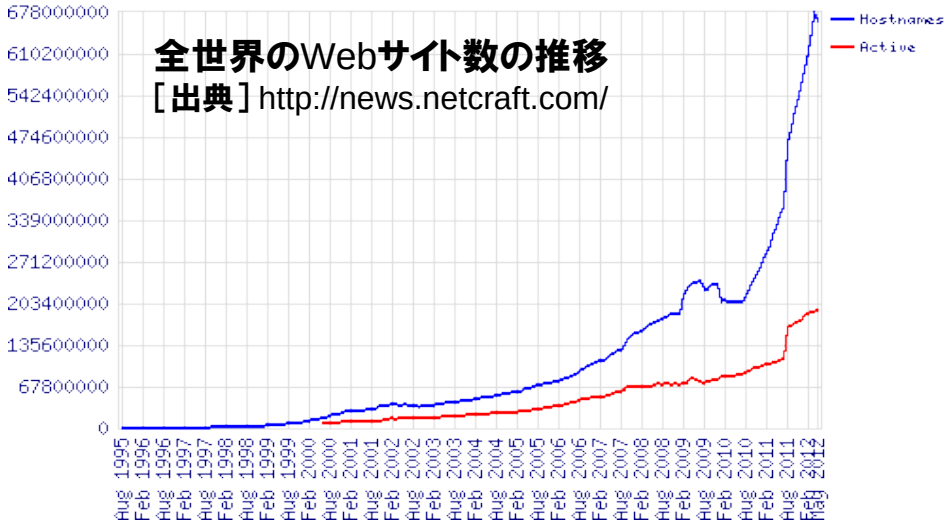
第2部　Big Data処理技術①　大規模な蓄積データ処理技術

# 大量に溜まったデータを効率良くさばくコツは？

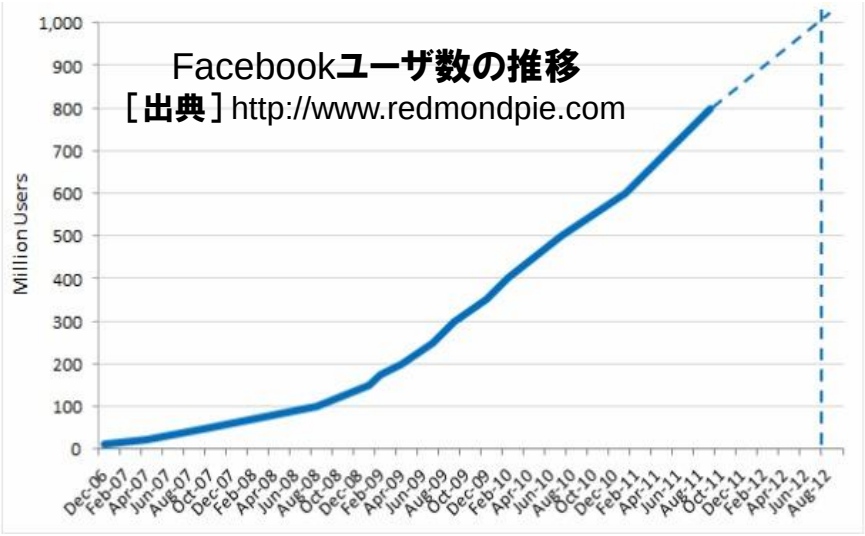
# 情報爆発 ～ 様々なデータが急激に増大

- 全世界のWebサイト数は6.6億以上、Webページ総数は兆オーダー
- Facebookのユーザ数は2012年8月に10億人を超える見込み
- 人手で作られるコンテンツだけでなく、カメラやセンサーから取り込まれるストリームデータも急増
- クラウドコンピューティングの普及によるデータ集中化も加速

**大量データ（Big Data）処理のニーズ増大・事業機会拡大**




全世界のWebサイト数の推移
［出典］http://news.netcraft.com/

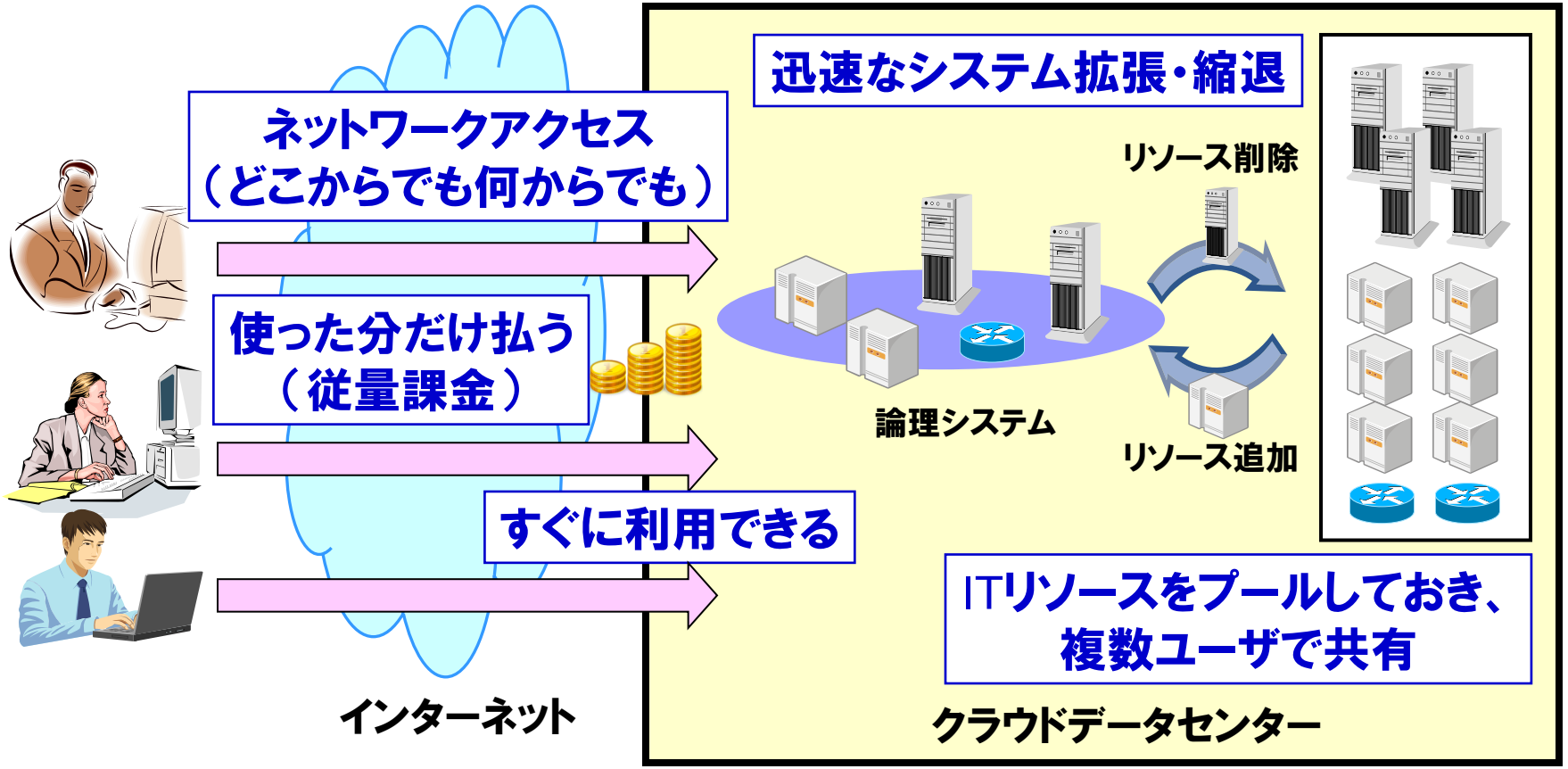



Facebookユーザ数の推移
［出典］http://www.redmondpie.com

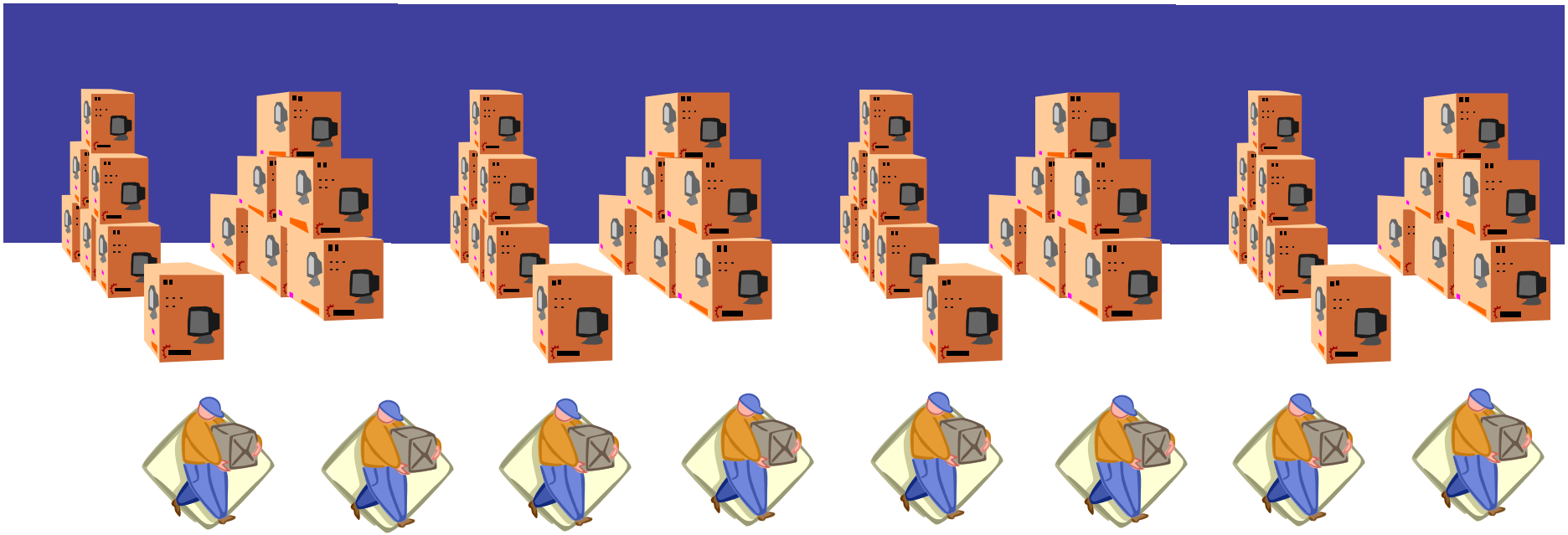
Googleの調査によると2008年7月にユニークURL数が1兆を突破
［出典］http://googleblog.blogspot.jp

# クラウドコンピューティング（Cloud Computing）

インターネット側（向こう側）にあるサーバ、ストレージ、データ、アプリケーションなどのITリソースを、こちら側からサービスとして利用するコンピューティングモデル



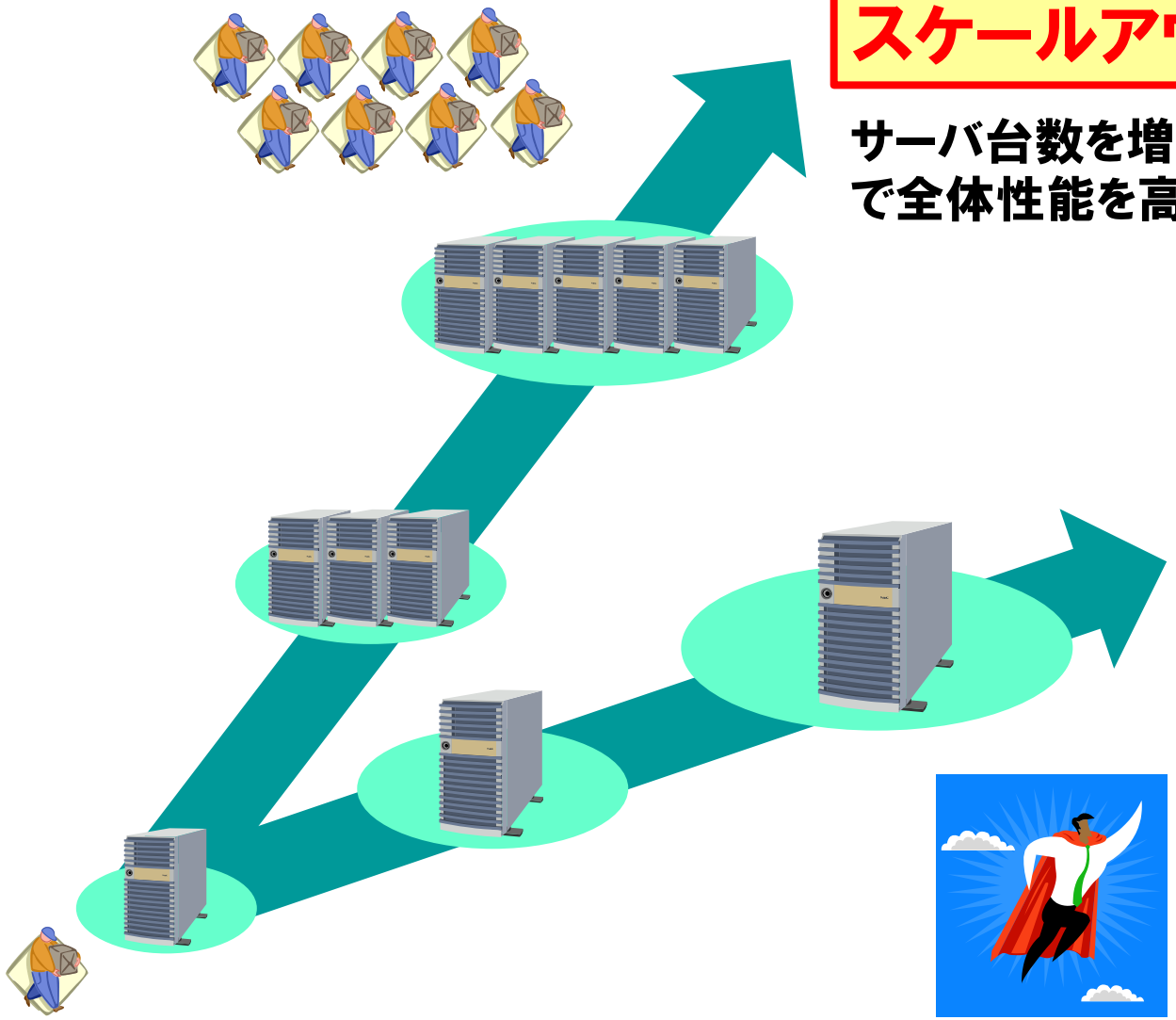
# 大量の荷物を急いで片付けるには？



一人では大変

多人数で分担すればすぐ片付けられる！

# サーバ処理能力を向上させる2つのアプローチ



**スケールアウト**

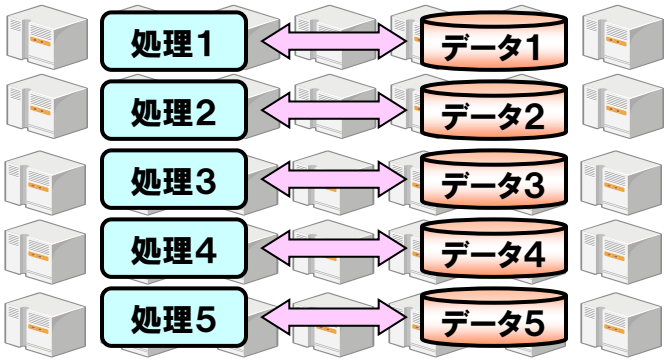
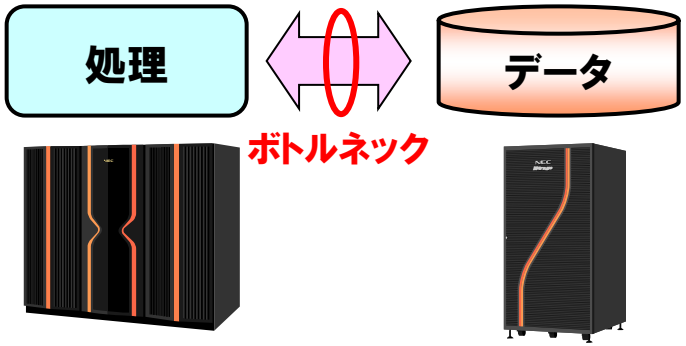
サーバ台数を増やすことで全体性能を高める

**スケールアップ**

サーバ単体の性能を向上させる

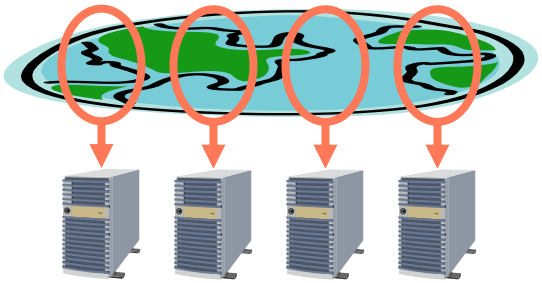
# スケールアウトとスケールアップの比較

- 増大し続けるデータ(Big Data)に対してはスケーラビリティ確保が必要
- そのためには、データを分割して分散並列処理することで、スケールアウトさせるアプローチが有効

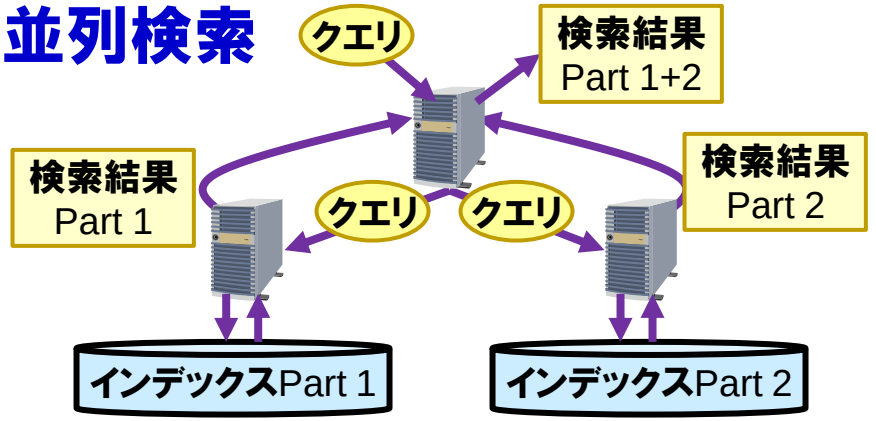
| スケールアウト | スケールアップ |
|---|---|
| ● 分散並列型、HWは普及機を使用できるのでスモールスタートから比較的安価に拡張可能、SWライセンスが高くなりがち<br>● 独立性が高いデータを分割管理するケースで効果的(DBはNoSQL系が適する)<br>● 保守や障害発生時にサービスを停止しなくともよい | ● 集中型、HWは高価なハイエンド機への投資に向かう、一方でSWライセンスは抑えられる<br>● データ間整合等の厳密な管理がしやすい(DBにRDBMSがよく用いられる)、データ増大時にI/Oがボトルネックになる<br>● 保守や障害発生時にサービス停止 |
|  |  |

# Webサーチエンジンの大規模化対応もスケールアウト

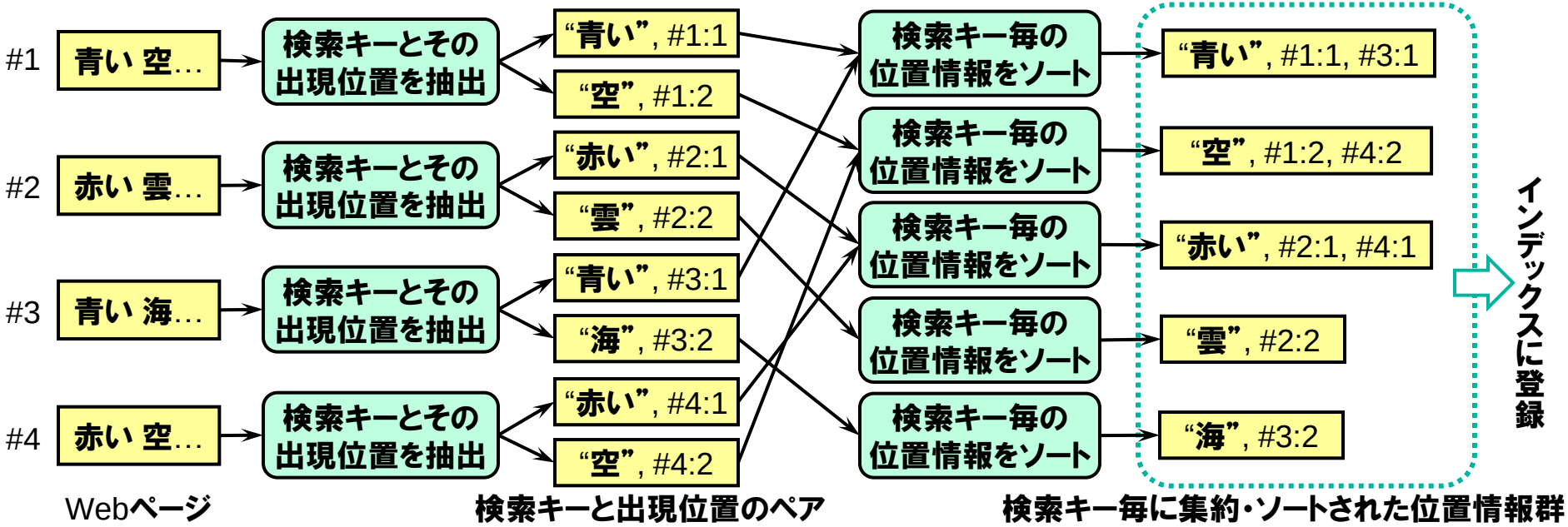
## 分散並列クローリング



## 分散並列検索

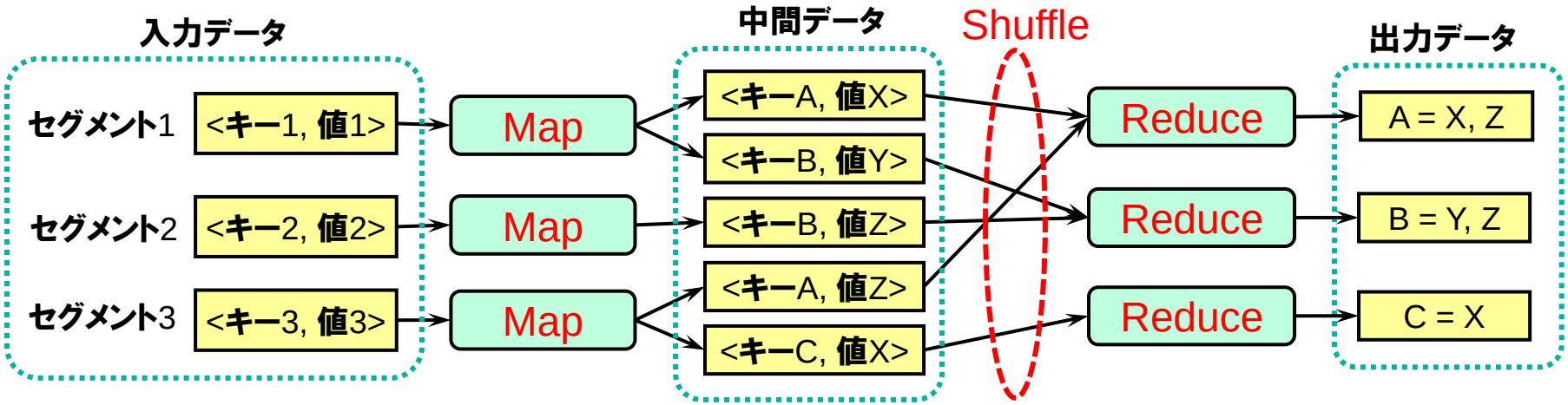



## 分散並列インデクシング



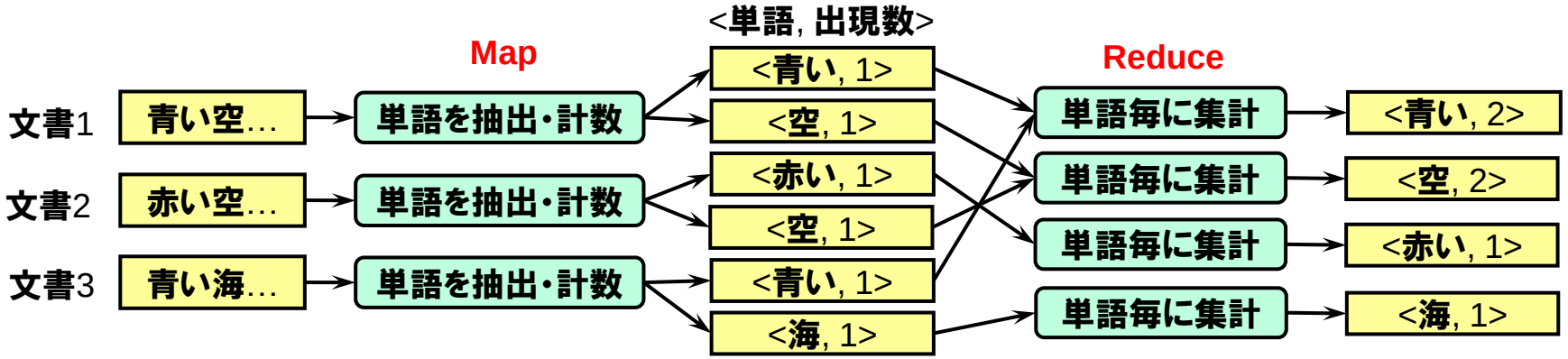
# 分散並列処理フレームワーク MapReduce（1/3）

- Webサーチエンジンに用いられていた分散並列処理を、より広い用途に利用可能にしたフレームワーク
  - Map：入力データの各セグメントに対して、新しいキーと値を生成する
  - Reduce：同じキーを持つMap結果を集めて、各キーに対する値を統合する

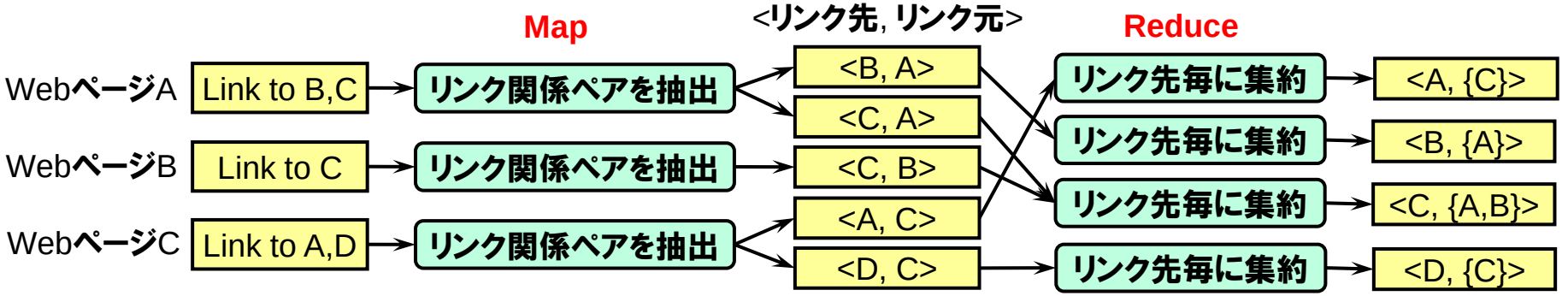


# 分散並列処理フレームワーク MapReduce（2/3）

## MapReduceの応用例① 文書集合に対する単語統計を計算



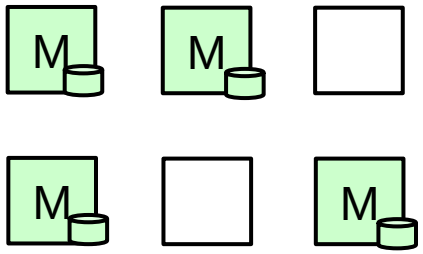

## MapReduceの応用例② Webページの順リンク情報から逆リンクリストを作成



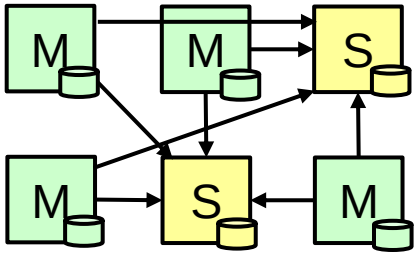
# 分散並列処理フレームワーク MapReduce（3/3）

- **サーバ群に対するMap処理やReduce処理の割り当ては、自動的に行われ、MapReduceを用いるプログラマは意識する必要がない**
  - **プログラマはMap関数とReduce関数を記述し、入力ファイルを指定**
  - **空いているサーバに順に処理が割り当てられ、タスクが遂行される**

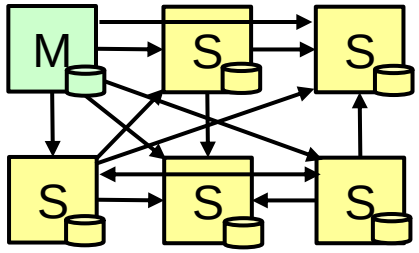
①Map処理が始まる



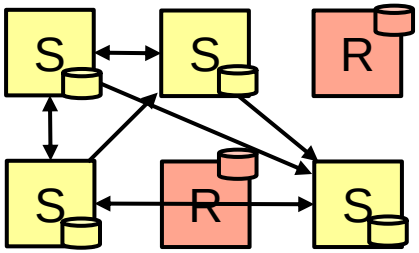

②Shuffleが始まる



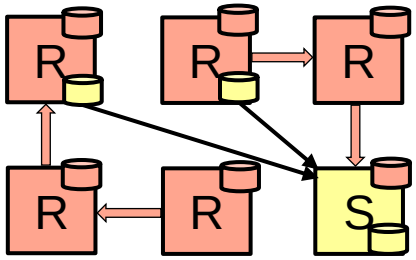

③Shuffleが続く



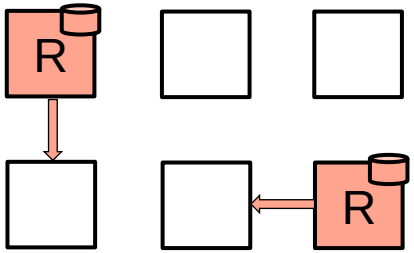

④Reduce処理が始まる




⑤Reduceによる出力




⑥MapReduceの完了




M=Map, S=Shuffle, R=Reduce

# 第2部のまとめ

## Big Data処理技術①　大規模な蓄積データ処理技術

**大量に溜まったデータを効率良くさばくコツは？**

**データを分割して分散並列処理することでスケールアウトさせる**

- Web検索エンジンのために開発されてきた大規模データ処理技術が、様々なビッグデータアプリケーションに活用されるようになった
- Googleの論文に基づき、その大規模データ処理技術をJavaでクローン実装したオープンソースソフトウェアHadoopがよく使われている

| 機能 | Google | Apache Hadoop |
|---|---|---|
| 抽出-集約型の分散並列処理フレームワーク | MapReduce | Hadoop MapReduce |
| 分散Key-Valueストア | BigTable | hBase |
| 分散ファイルシステム | GFS (Google File System) | HDFS (Hadoop Distributed File System) |

第3部　Big Data処理技術②　大規模な実世界センシングデータ処理技術

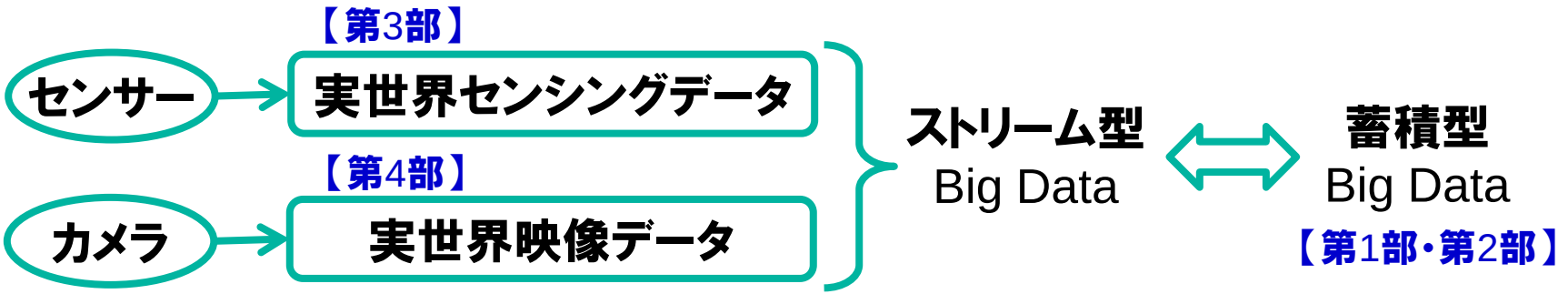
# 大量に流れ込んでくるデータを溜めずにさばくコツは？

# Big Data処理技術の方向性

## 大規模化への対応だけでなく、3Vの方向へ進化

- **Volume（量）**：データ源となる端末・デバイス・サービスが増大、情報量が爆発的に増加
- **Velocity（速度）**：情報が流通するスピード、意思決定に求められるスピードが速まり、データ処理のリアルタイム性が求められる
- **Variety（多様性）**：構造情報（DBレコード等）から非構造情報（Web・SNS・画像・映像・センサデータ等）までデータの多様性が拡大

## センサーやカメラから集まる実世界データが増大し、蓄積型Big Dataだけでなく、ストリーム型Big Dataのリアルタイム処理が必要

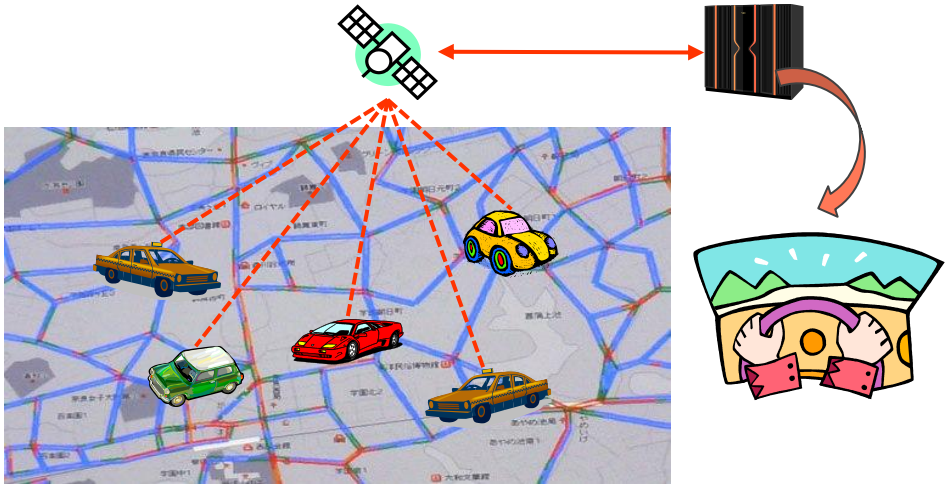


# 実世界センシングデータを活用するアプリケーション例

## 状況適応情報サービス

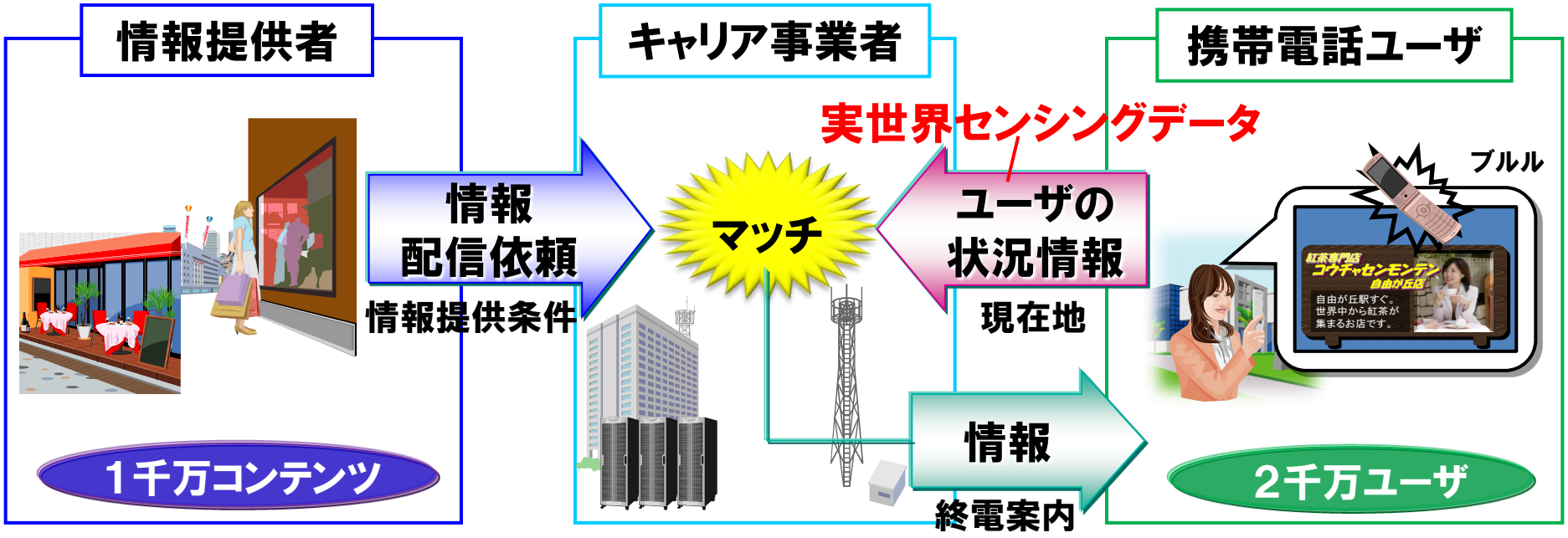
- モバイル機器から得られる実世界センシングデータ（位置・操作等）から
  ユーザの状況・嗜好を推定し、その状況にマッチする情報を提供


近くのお薦めスポットやレストランを紹介、これから立ち寄ると予想される場所の天気情報（傘の必要性等）、交通情報（電車の遅延等）、お得情報（店舗のクーポン等）を先回り配信



GPSにより自動車（タクシー等）の位置情報を収集・分析することで、現時点の渋滞状況を把握、さらに今後の渋滞傾向も予測、ドライバーに渋滞情報や事故発生時の回避ルート等をアドバイス

# 状況適応情報サービスの大規模化時の課題




## 大規模化時の問題

例1：終電案内が終電後に通知される
（推薦が遅い）

例2：急なタイムセール情報は通知できない
（前日に配信依頼が必要）

例3：推薦されたレストランは混雑していて
入れない（扱う実データが限定的）

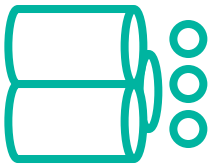
## 技術課題

① **スケーラビリティ**
数千万ユーザ×数百万ルールの大規模化にも対応

② **リアルタイム**
タイムラグなし、数秒程度の遅れ

# 実世界センシングデータの規模感と求められる処理性能

| サービス | 必要イベント数 | | ユーザから求められるレスポンスタイム |
|---|---|---|---|
| | イベント数・概要 | 算出ロジック | |
|  携帯電話位置情報  | 830,000イベント/秒<br>イベント数は83万イベントと大容量になる | ・5,000万人(携帯電話人口の半分)÷60秒(一分に一回送信) =83万イベント/秒 | 1秒程度<br>位置情報を用いて利用するサービスではハイレスポンスが求められる |
| プローブカー  | 1,330,000イベント/秒<br>イベント数は100万イベントと大容量になる | ・自動車数8,000万台(自動車検査登録協会2800)÷60秒(一分に一回情報を収集)=133万イベント/秒 | 10秒程度<br>渋滞情報の提供などで必要となるレスポンスタイムは10秒程度であると推定 |
|  家電・エコモニタリング  | 130,000イベント/秒<br>制御タイミング次第でシステムに吸い上げる頻度が決まるため、用途が明確になれば、大容量になる可能性がある | ・ハイエンドな一戸建て数40万戸(セコム加入世帯数と同程度と判断)×家電数(20個/1戸と仮定)×収集頻度(1分に1回)=約13万イベント/秒(電力モニタリングでは1msec単位で計測しているが、現状ネットワークに流すことが出来ず測定機内に蓄積している) | 1秒程度<br>電気設備の制御を行うのであれば、ハイレスポンスが求められる |
| 農業  | 200,000イベント/秒<br>農作地の状況データ(温湿度、風量、風向、日射量等)を広範囲で吸い上げるため大容量になると予想される | ・農家300万戸の40%がフィールドサーバを導入。センサ数平均約10個、1分に1回情報を収集すると仮定すると20万イベント/秒 | 1分程度<br>より高度なIT農業を行うのであれば、レスポンスタイムは1分程度である必要があると推定 |

実世界センシングデータ
＝イベントの時系列データ 

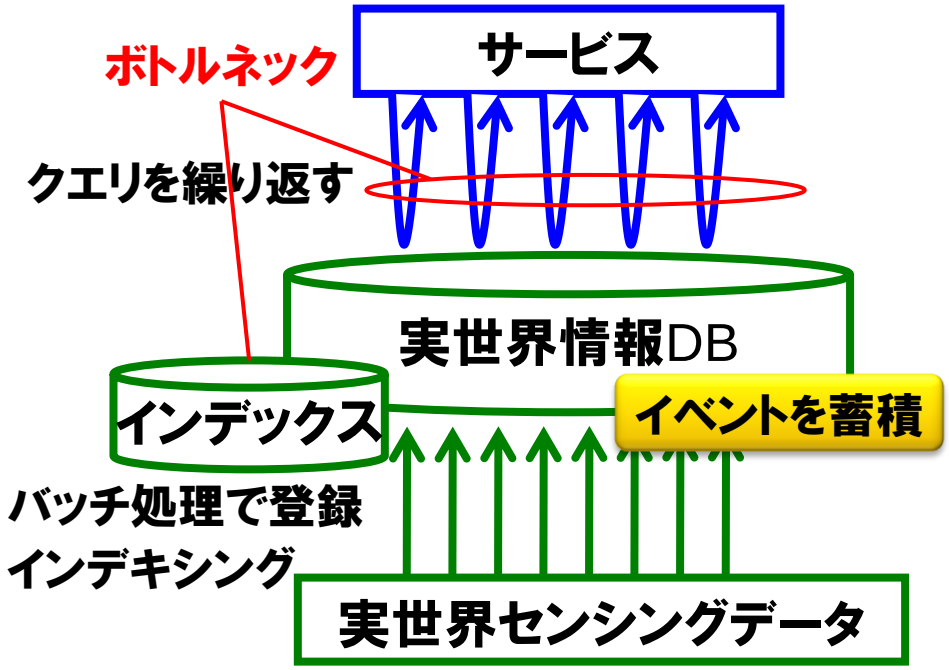
100万イベント/秒 以上の
リアルタイム処理性能が求められる

# リアルタイム処理へのアプローチ

- 従来の蓄積型Big Dataの処理方式ではリアルタイム性の確保が困難
- Pub/Sub型のオンザフライ処理によってリアルタイム性を実現

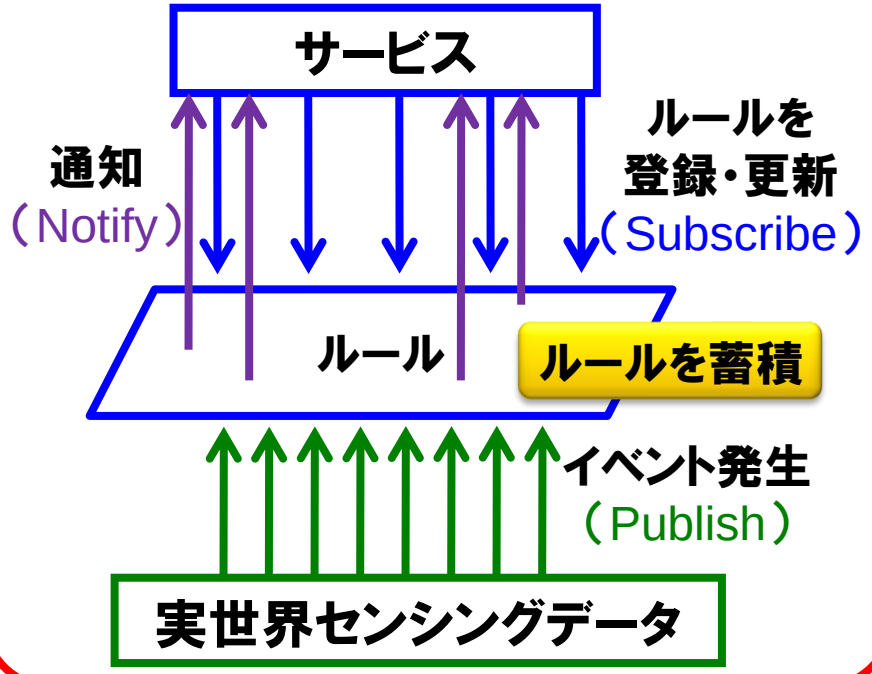
## 従来の蓄積型処理方式

- 実世界センシングデータ(イベント)を蓄積
- これにクエリをポーリングするアーキテクチャ




## Pub/Sub型のオンザフライ処理方式

- ルールをオンメモリで蓄積
- これに実世界センシングデータ(イベント)を流し込むPub/Sub型アーキテクチャ



# イベントとルール

## イベント

- **属性・属性値の集合**
  - Event（**属性**1 = **値**1, **属性**2 = **値**2, …）
  - ［**例**］ Event（**地域**ID = A, **時刻** = t, **測定値** = 10）

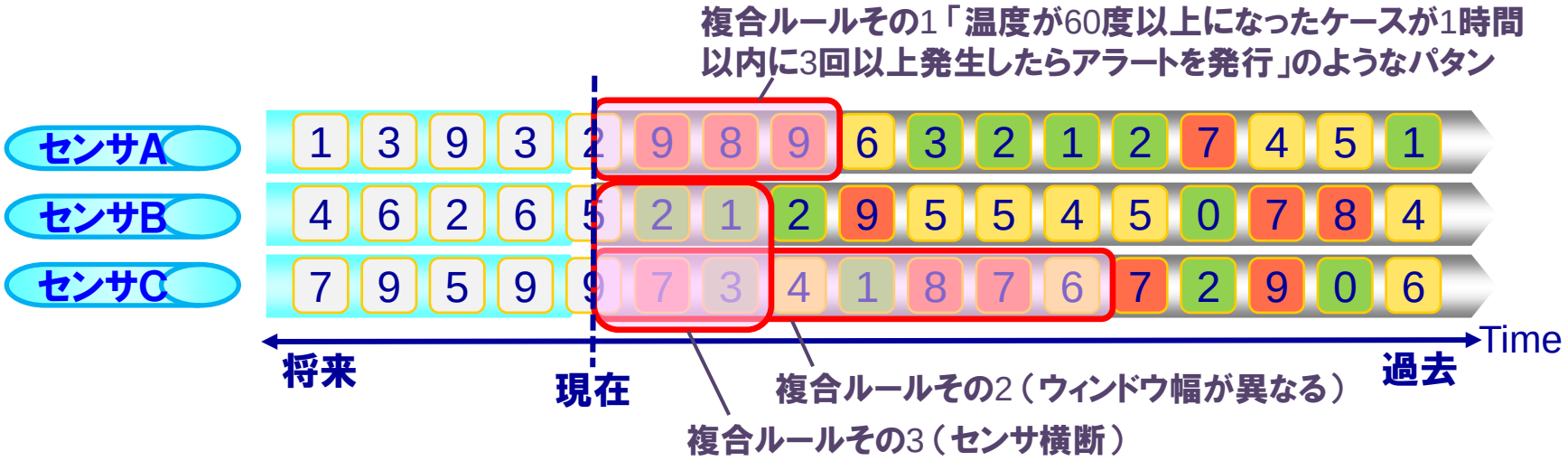
## ルール

- **単純ルール**（Simple Rule）
  **例えば「温度が60度以上になったらアラートを発行」のようなルール**
  - Rule（Event-A, Action）
- **複合ルール**（Complex Rule）
  **例えば「温度が60度以上になったケースが1時間以内に3回以上発生したらアラートを発行」のようなルール**
  - Rule（Event-A **|** Event-B **|** ... **|** Event-N, Action）←**どれか起きる**
  - Rule（Event-A **&** Event-B **&** ... **&** Event-N **within t**, Action）←**どれも起きる**
  - Rule（Event-A → Event-B → ... → Event-N **within t**, Action）←**順に起きる**

# 複合イベント処理（CEP）

## 複合イベント処理（Complex Event Processing）とは

- イベントの時系列データ中から、特定のパタン（複合ルールで指定）をリアルタイムに検出




## スケールアウト（分散並列処理）のための主な技術課題

- 複合ルールのマッチングでは、どこまでマッチしたかというステートを管理する必要があり、ルールの複雑さに応じてマッチング処理の負荷が異なる
- 複合ルール間で、それらを構成する単純ルールに重複があり、単純ルール・複合ルール間の依存関係を考慮した分散並列処理の効率化が必要

# スケールアウト可能なCEP方式［NEC］（1/4）

## CEPの処理をステートレス処理とステートフル処理に分解

- ステートレス処理を担当するイベントディスパッチャ（複数台）を前段、ステートフル処理を担当するイベントプロセッサ（複数台）を後段とした2段構成をとる

## 依存関係を考慮して最適配置したルールを用いて分散並列処理を実行

- 同じイベントに関するルールは同じイベントプロセッサに配置し、かつ、イベントプロセッサの負荷が均等になるように、ルール配置を最適化

［例］

Rule 1（A & **B**, Action-X）

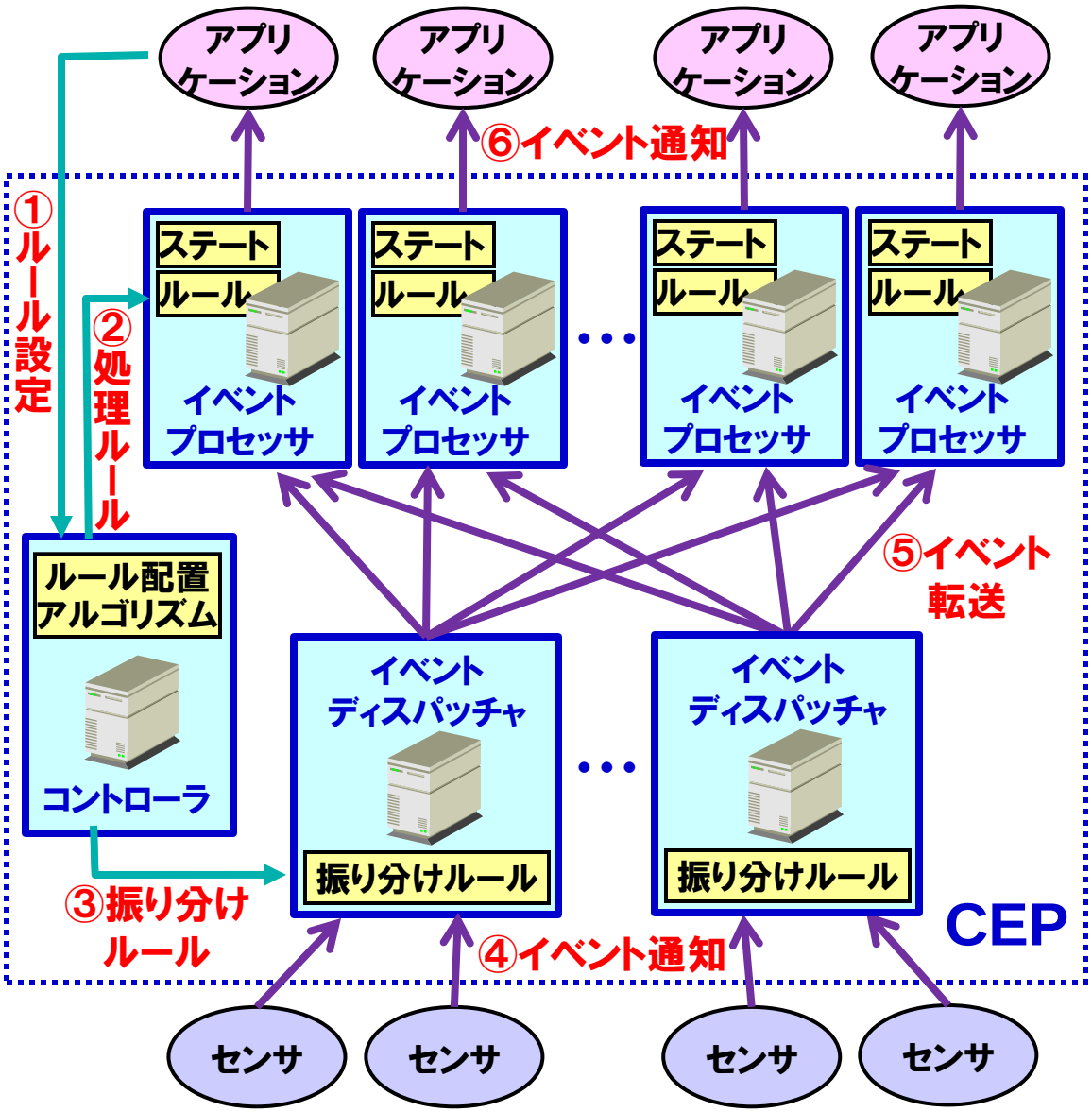
Rule 2（**B** → C, Action-Y）

Rule 3（D → E, Action-Z）

→ イベントBが共通なので同じイベントプロセッサに配置

**前頁にあげた課題を解決し、スケールアウト可能なCEPを実現**

# スケールアウト可能なCEP方式［NEC］(2/4)




## コントローラ

- アプリケーションからルールの登録を受け付け(①)、ステートレス部とステートフル部に分解
- ルールの最適配置を計算し、その結果に従って、イベントプロセッサにルールを配置(②)、イベントディスパッチャの振り分けルールを設定(③)

## イベントディスパッチャ

- イベント源(センサ)からイベントを受け付け(④)、コントローラからの指示通りにイベントプロセッサへ振り分ける(⑤)
- ステート管理なし

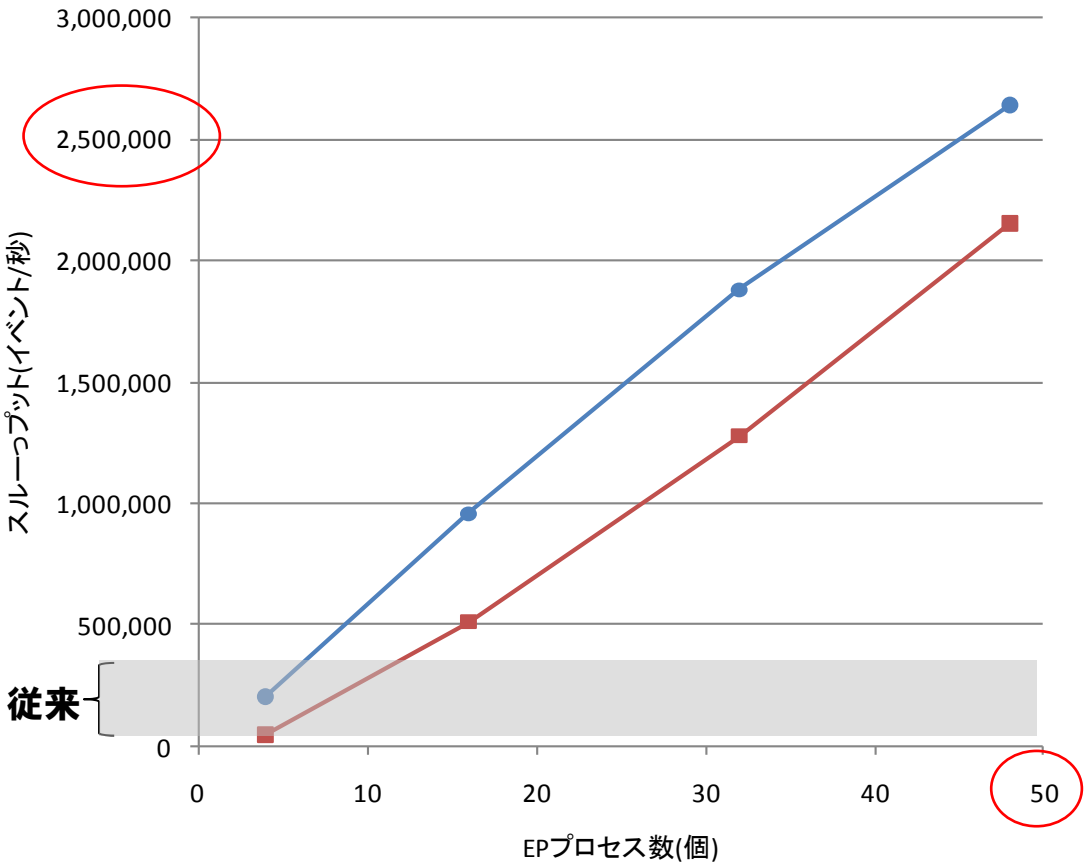
## イベントプロセッサ

- コントローラからセットされたルールと、イベントディスパッチャから転送されてくるイベントを、ステート管理をしながらマッチング
- マッチング結果をアプリケーションに通知(⑥)

# スケールアウト可能なCEP方式［NEC］（3/4）

## スケールアウトによる高いスケーラビリティを実現

- 50プロセスで250万イベント/秒




NEC Express5800/A1160 (4CPUサーバ) x 4台)
CPU: Xeon X7460 (6CPUコア内蔵、クロック2.66GHz)
メモリ: 128GB
NEC Express5800/120Bb-d6 (2CPUサーバ) x 12台
CPU: Xeon X5355 (4CPUコア内蔵、クロック2.66GHz)
メモリ: 16GB
OS: Windows Server 2008R2 Enterprise Edition (64bit)

スループット(マッチ率0.1)
スループット(マッチ率0.5)

# スケールアウト可能なCEP方式［NEC］(4/4)

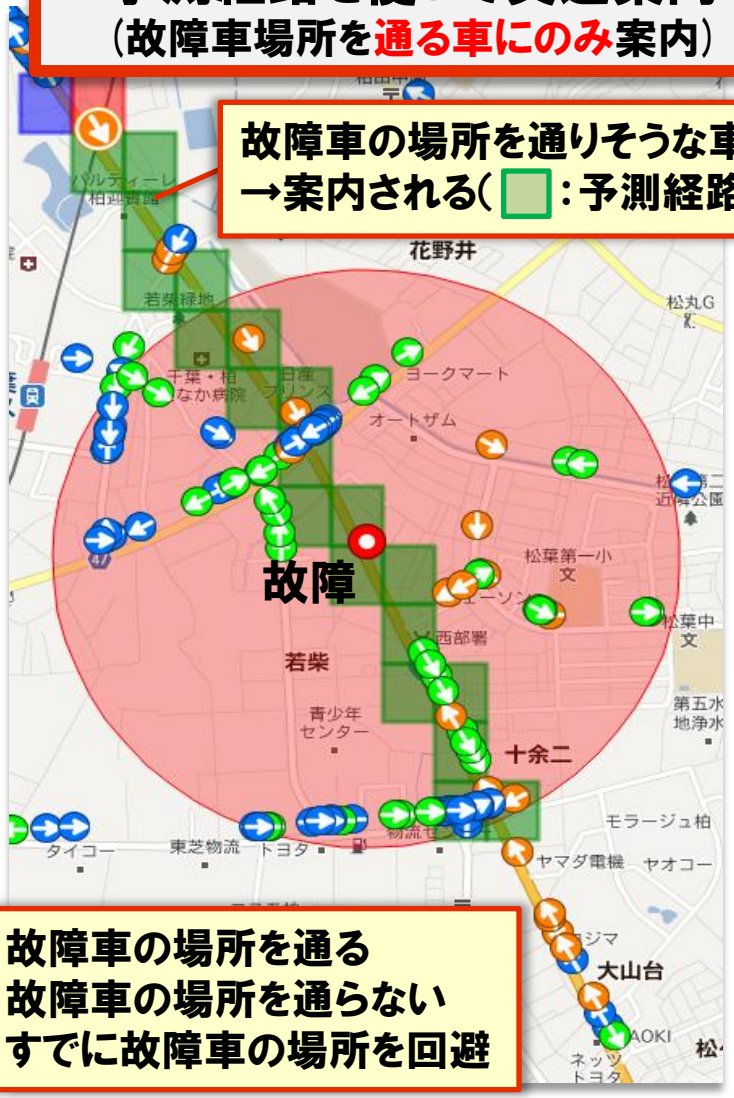
**Before**

**位置情報のみで交通案内**
(故障車から**一定範囲内**の車に一律に案内)


**複雑な処理を高速に**

**After**

**予測経路を使って交通案内**
(故障車場所を**通る車にのみ**案内)



# 第3部のまとめ

Big Data処理技術②　大規模な実世界センシングデータ処理技術

**大量に流れ込んでくるデータを溜めずにさばくコツは？**

| 従来の蓄積型処理方式 | Pub/Sub型のオンザフライ処理方式 |
|---|---|
| ● イベントを蓄積<br>● これにクエリをポーリングするアーキテクチャ | ● ルールをオンメモリで蓄積<br>● これにイベントを流し込むPub/Sub型アーキテクチャ |

＋

● 加えて、ルール配置を工夫することで、CEP（複合イベント処理）の分散並列化を可能にし、スケールアウトを実現

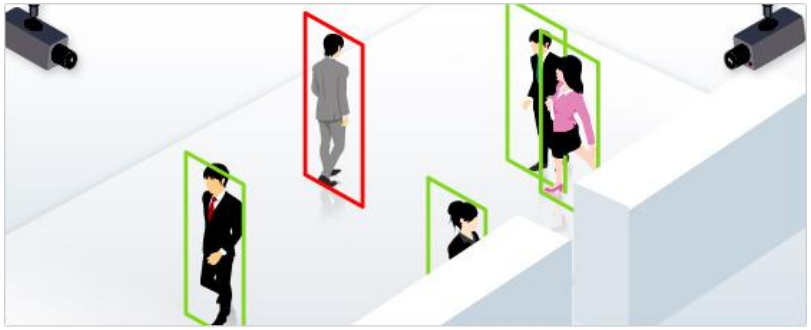
第4部　Big Data処理技術③　大規模な実世界映像データ解析技術

# カメラに見守られるのはSF世界だけの話なの？

# 実世界映像データを活用するアプリケーション例

## カメラによる人物追跡

- 監視カメラ等から得られる実世界映像データを解析して、人物を検出し、それが「誰であるか」、「何をしているか」を判別



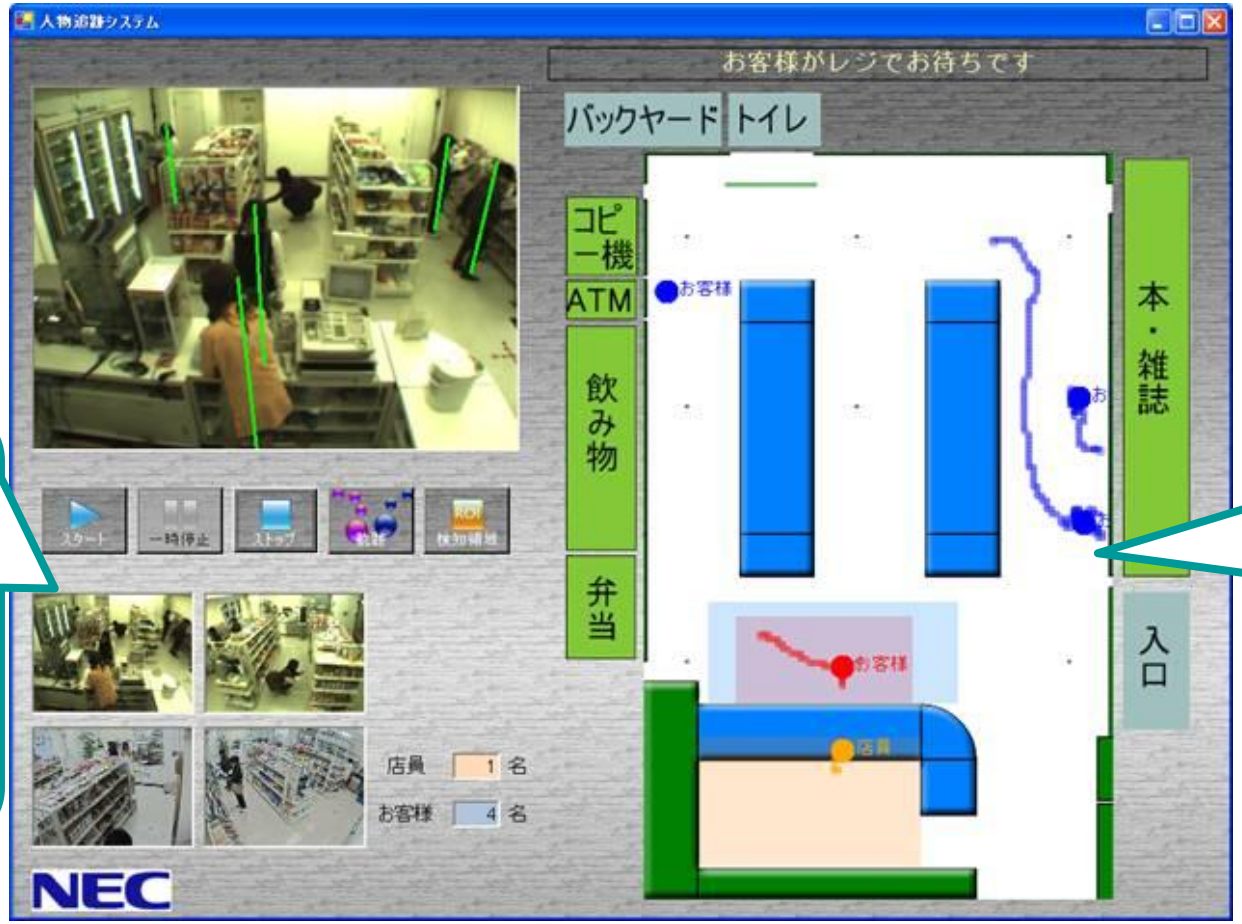
建物内や特定エリア等をカメラで監視し、不審者の侵入や不審行動を検知したらアラートを発信、その人物を追跡



店舗内をカメラで監視し、来店者の性別・年齢層や動線を抽出することで、来店者の傾向を把握し、今後の集客・売上向上に結び付ける

# マルチカメラ映像解析による人物追跡（1/3）

## 異なる角度から複数カメラで撮影した映像を対応付けて解析し、人の動きを高精度・リアルタイムに追跡

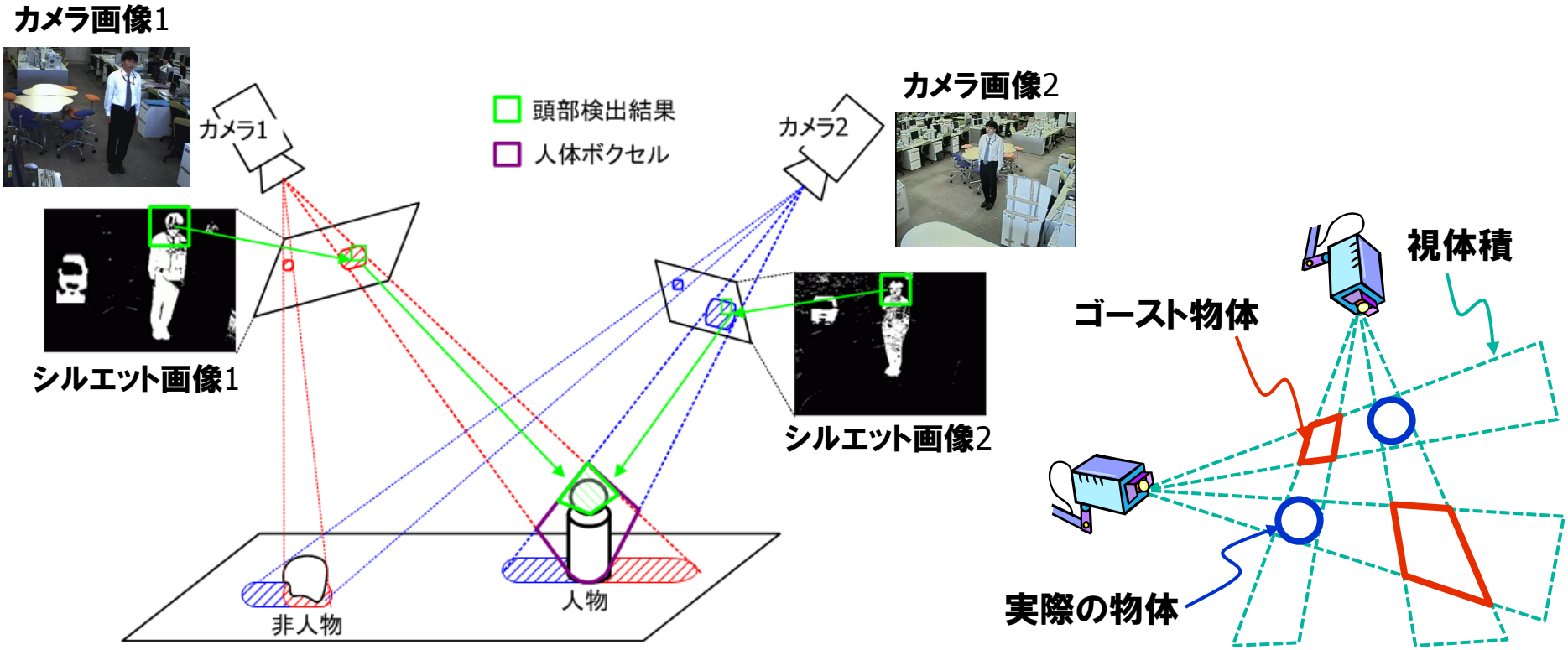



複数カメラで店内を撮影、各カメラの映像から人物部分を検出・対応付け、距離も計算

人物の位置・移動を追跡（マップ上に表示）

# マルチカメラ映像解析による人物追跡（2/3）

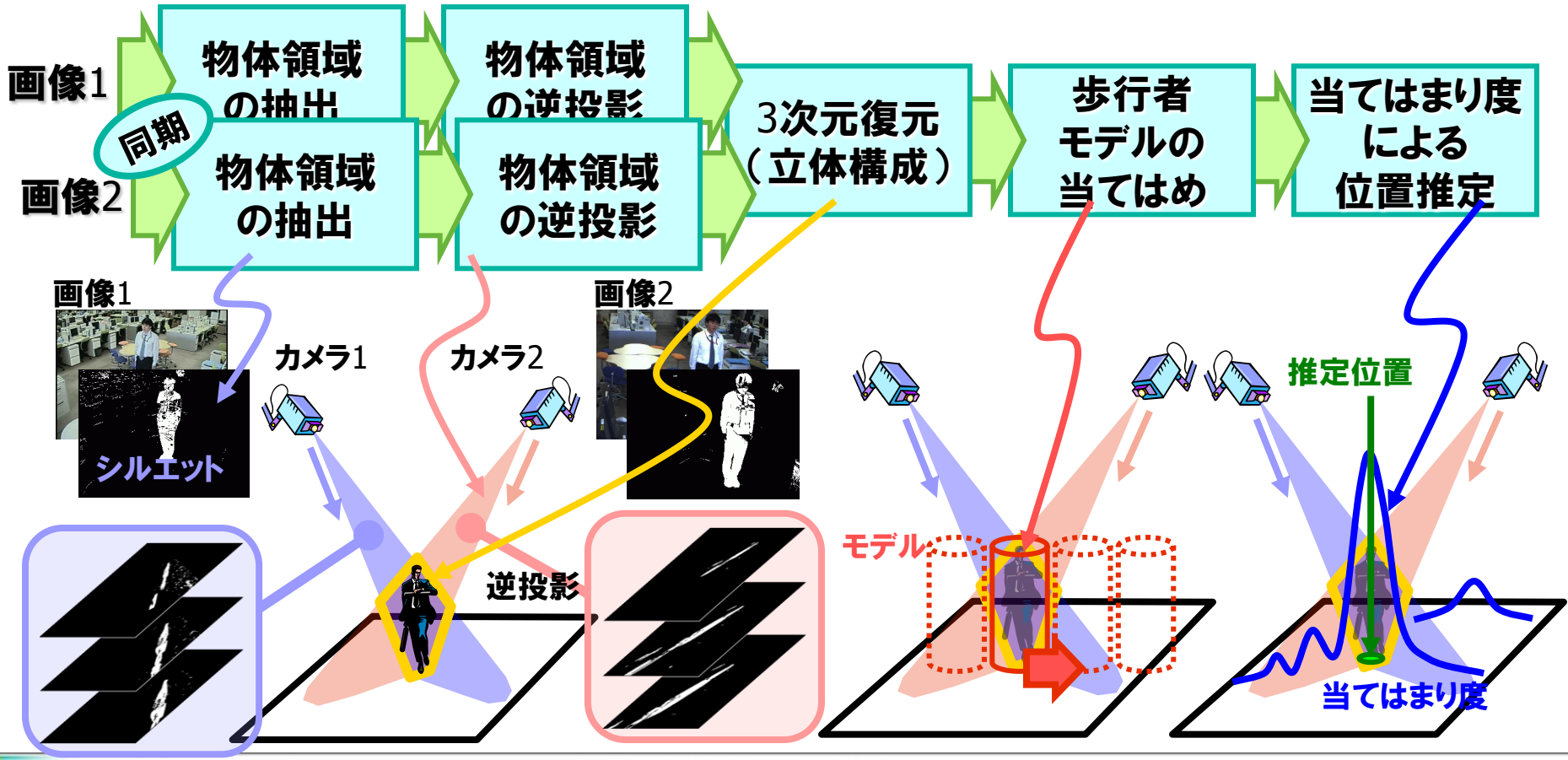
- 各カメラ映像から背景差分処理によって物体領域（シルエット）を抽出
- **視体積交差法**を利用して3次元空間における物体の位置・形状を推定
  - シルエットを3次元空間に逆投影し、それらの視体積の交差部分を求めることで物体の3次元形状を推定（復元）



# マルチカメラ映像解析による人物追跡（3/3）

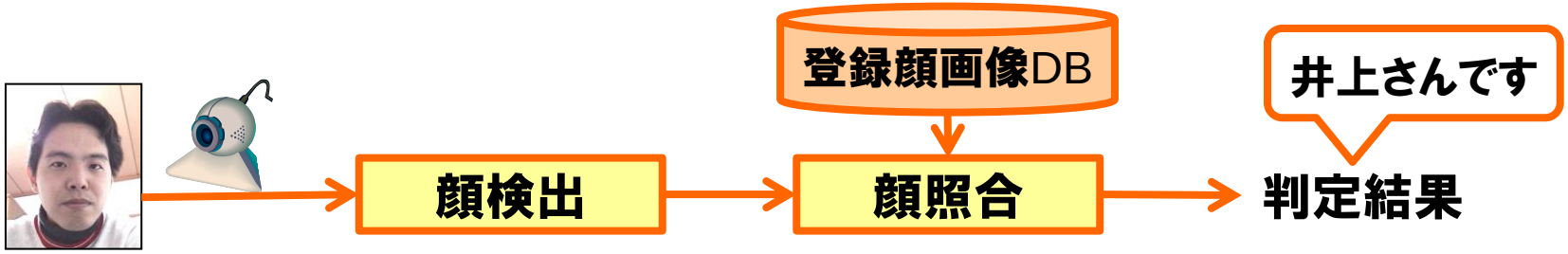
## 3次元形状を復元後、歩行者モデルの当てはめによって位置推定

- 人の存在位置の近似や時系列の追跡によって、追跡速度(リアルタイム性）や人物追跡精度(隠れや重なりの分離等)を改善



# 顔認証（1/3）

**顔認証：顔を写した画像・映像にもとに、それが誰であるかを判断する（事前にDBに登録してある人の中の誰であるかを見つける）**




## NECの顔認証製品 NeoFace

- http://www.nec.co.jp/soft/neoface/index.html

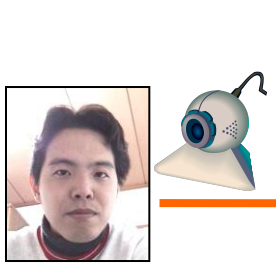
**入国管理・入館管理・入園管理等のゲートシステム（顔パスシステム）**

**不審者追跡・迷子検索等の顔検索**

**パソコンや携帯の使用者ロック機能等**

# 顔認証（2/3） 技術課題と解決方式

**技術的困難さ：** 照明条件が異なる、顔の向きが異なる、表情が異なる、歳をとる、眼鏡の有無や髪形が異なる等、環境や対象物に変動があっても同一人物と判定できる（その一方で別人を同一人物と間違わない）



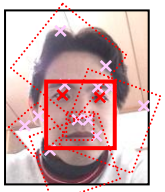


登録顔画像DB

井上さんです

顔検出 → 顔照合 → 判定結果

## 多重照合顔検出法



探索

顔判別

①目らしい領域の抽出： リングフィルタによって孤立点検出、中心が黒で周囲が白い領域を目候補とする

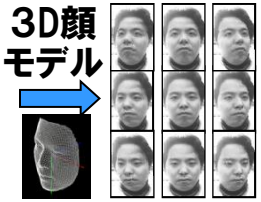
②顔・非顔判定： 目候補の任意のペアを両目候補とし、それを含む領域に対して顔/非顔を判定する

NEC独自の学習アルゴリズムGLVQ（一般化学習ベクトル量子化）を用いた大量の顔/非顔画像の学習によって、高精度な顔/非顔の判定を実現

## 摂動空間法

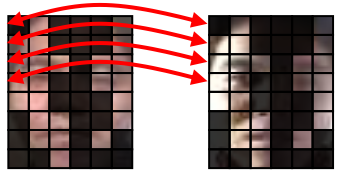
登録画像

3D顔モデル



登録画像（正面顔）から様々な姿勢・照明条件を変動させた顔画像を生成することで、顔の向きや照明等の大局的な変動を吸収

## 適応的領域混合マッチング法



登録画像　照合画像

類似度の高い部分領域に着目することで、眼鏡の有無や表情変化等の局所的な変動を吸収

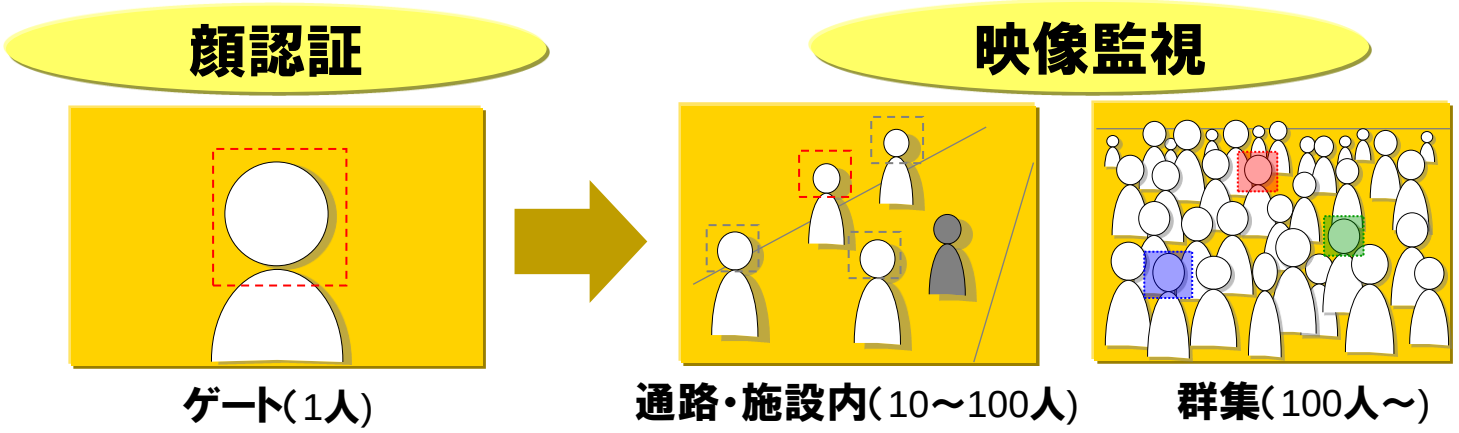
# 顔認証（3/3） ベンチマーク結果

## MBE (Multiple Biometrics Evaluation) 2010

- **米国NIST（国立標準技術研究所）が4年毎に開催する世界最高権威のベンチマークテスト、顔画像認証技術の実用性能評価**
- **世界中の有力ベンダーが参加、NECがダントツ1位の性能達成**

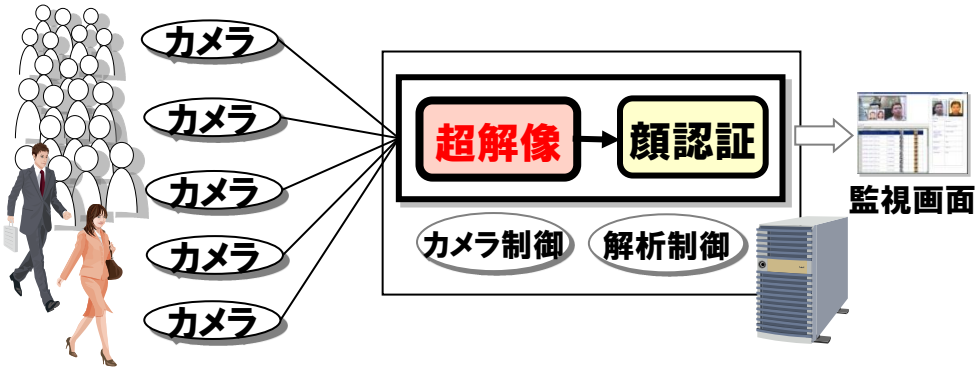
| | NEC（ダントツ1位） | | 2位企業 | 各性能が重視される応用例 |
|---|---|---|---|---|
| **認証精度**（エラー率） | 0.3%以下 | 1/10 | 2～3% | **出入国システム、利用者ログイン**（高精度が要求される用途） |
| **経年変化**（8年でのエラー率） | 3% | 1/4 | 12% | **パスポート認証、ブラックリスト照合**（10年前の写真との照合） |
| **多人種対応**（黒人・白人・アジア・アメリカンインディアン） | 平均4%<br>全人種でトップ | 1/2 | 平均8% | **出入国システム、国際空港内監視**（多様な人種が通過する場所での照合） |
| **顔向き**（25度でのエラー率） | 7% | 1/3 | 20% | **街頭・公共施設監視システム**（監視カメラによる照合） |
| **処理時間**（160万名） | 0.3秒 | 1/6 | 1.7秒 | **国民ID、ブラックリスト照合**（リアルタイム性が要求される用途） |

# 超解像（1/5）大規模映像監視の技術課題




## 大規模映像監視

- 街頭などに設置された多数の監視カメラに映った群集中の個人を認証
- 顔画像が低解像度になるため、顔認識が難しくなることが大きな課題
  ⇒顔認証の前に超解像処理を実行

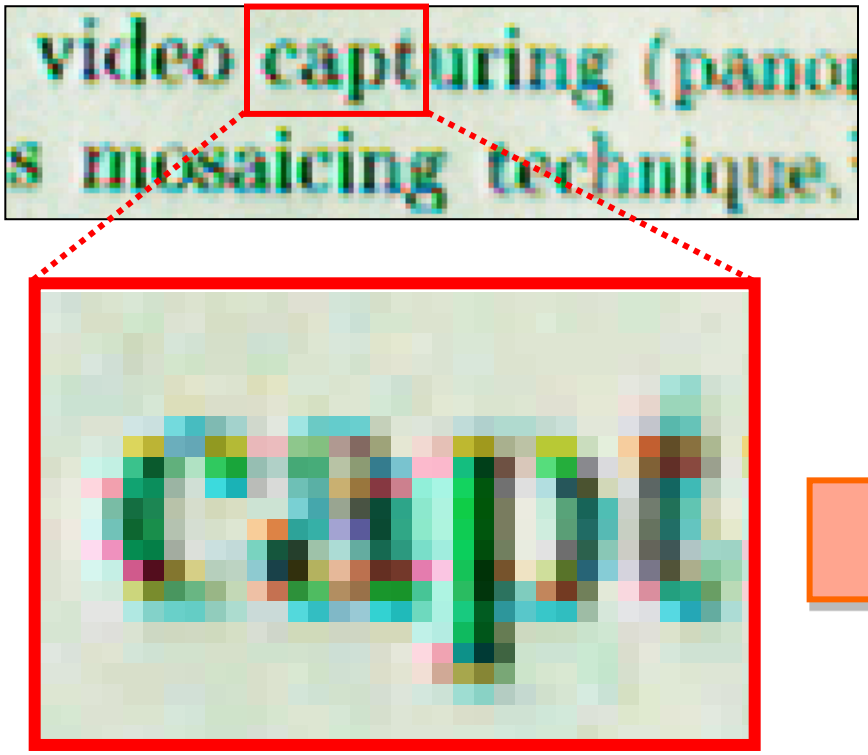


# 超解像（2/5）

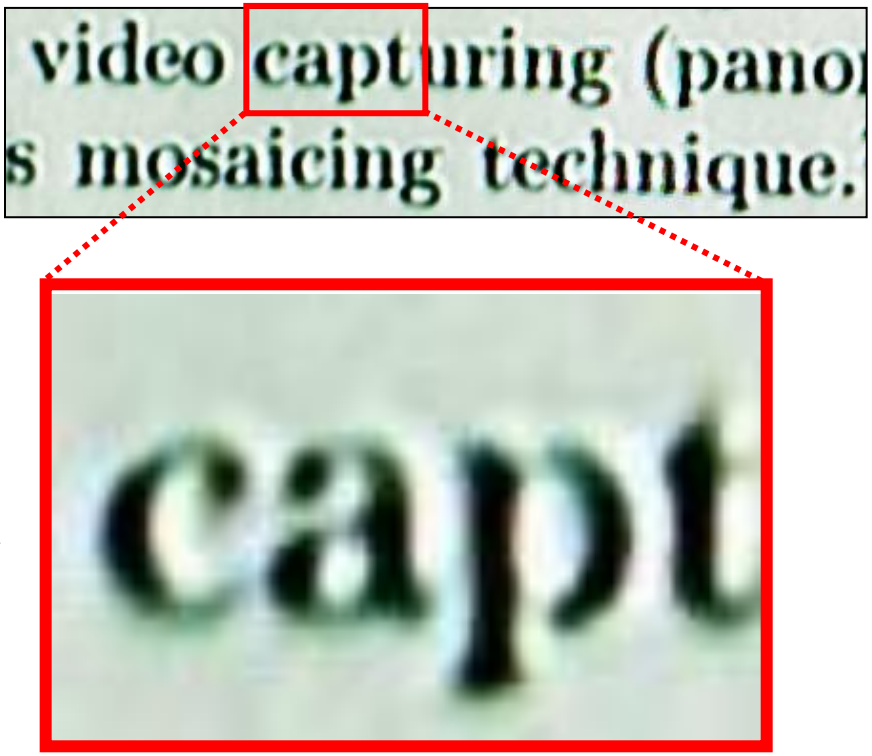
**超解像とは：　カメラ本来の分解能を越えた高解像画像を生成**

- カメラはそのままで、画像処理により解像度や画質を向上させられる
- 過去に撮影した画像を後処理で高解像度化することも可能

元画像




超解像画像



# 超解像（3/5） 方式の比較

## 複数枚超解像方式

- 複数枚の画像（動画もしくは連写）から高解像画像を復元

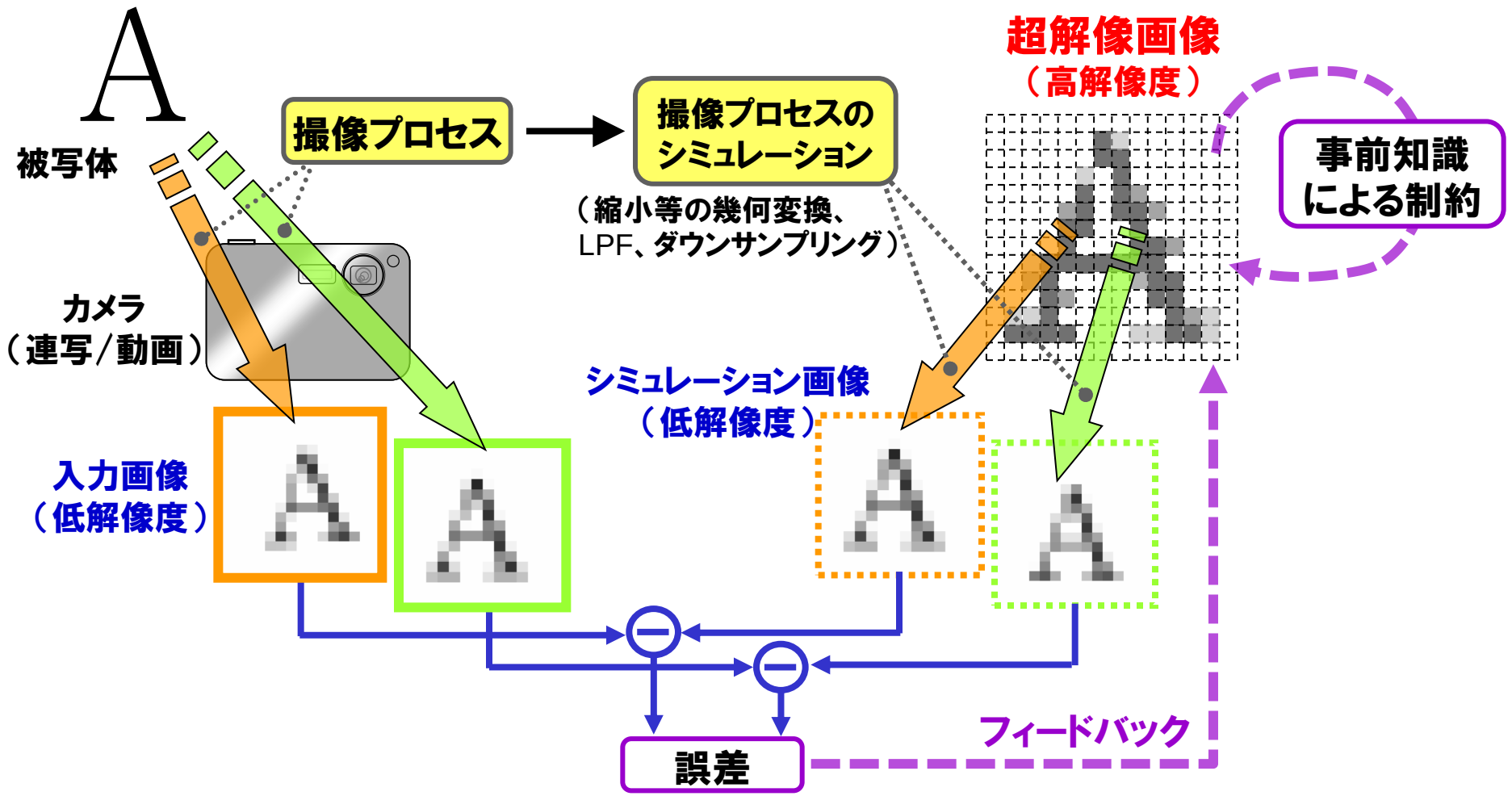
## 学習型超解像方式

- あらかじめ学習した知識を使って高解像画像を復元

| 方式 | 入力画像枚数 | 対象 | 倍率（画素数） |
|---|---|---|---|
| 複数枚超解像方式 | 複数枚（カメラまたは被写体の動きが必要） | 限定なし | ～3倍（～9倍） |
| 学習型超解像方式 | 1枚 | 特定の被写体（顔、ナンバープレート等） | 4倍～（16倍～） |

# 超解像（4/5） 複数枚超解像方式

- 撮像プロセスをシミュレートした低解像画像と入力画像との誤差を最小にする高解像度画像を推定



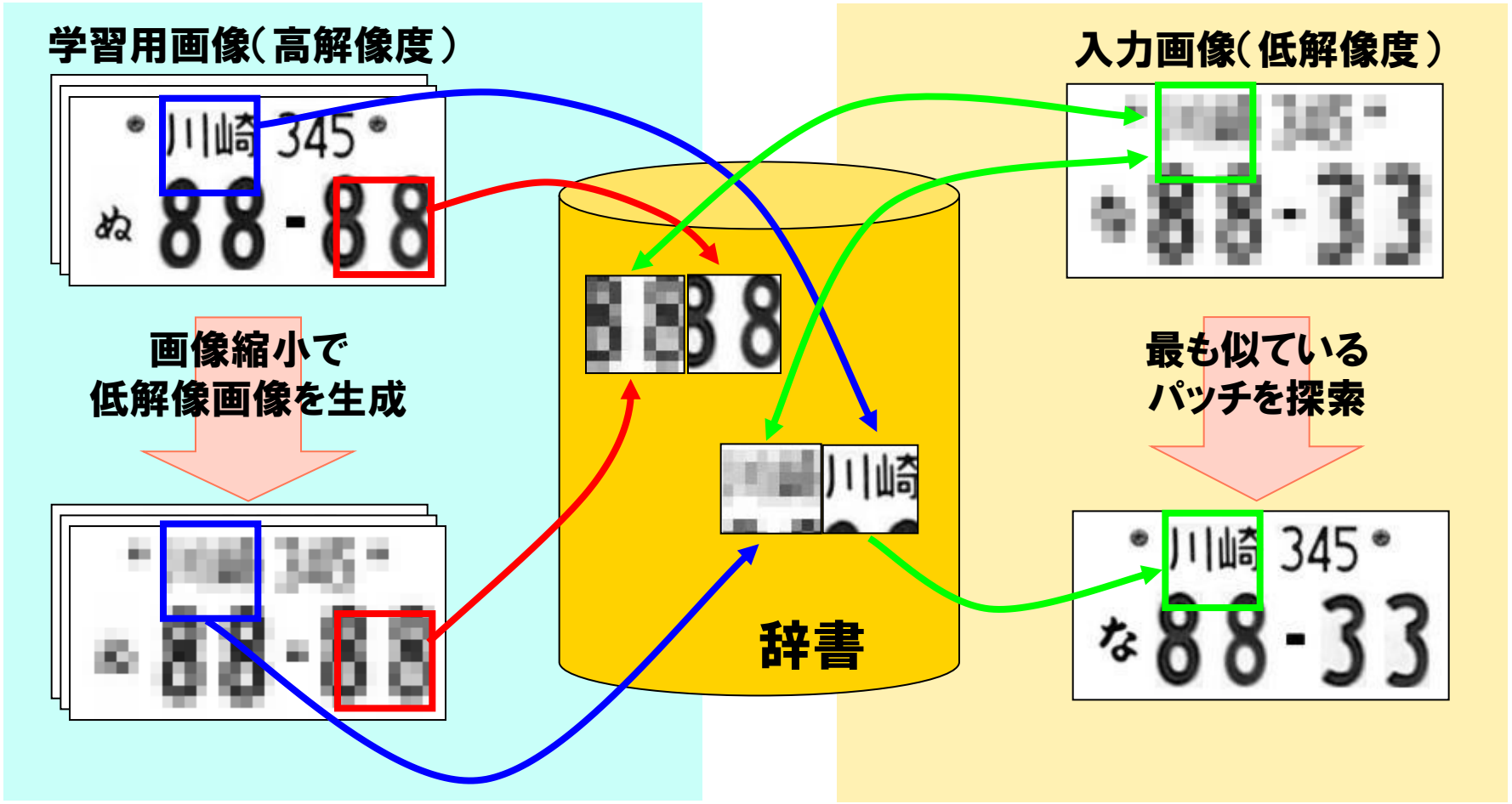
# 超解像（5/5） 学習型超解像方式

## 学習フェーズ

学習用画像中の小領域（パッチ）の
高解像・低解像画像ペアを多数辞書登録

## 復元（超解像）フェーズ

入力画像に対して最適なパッチを辞書から探索し、
合成処理によって高解像度画像を復元



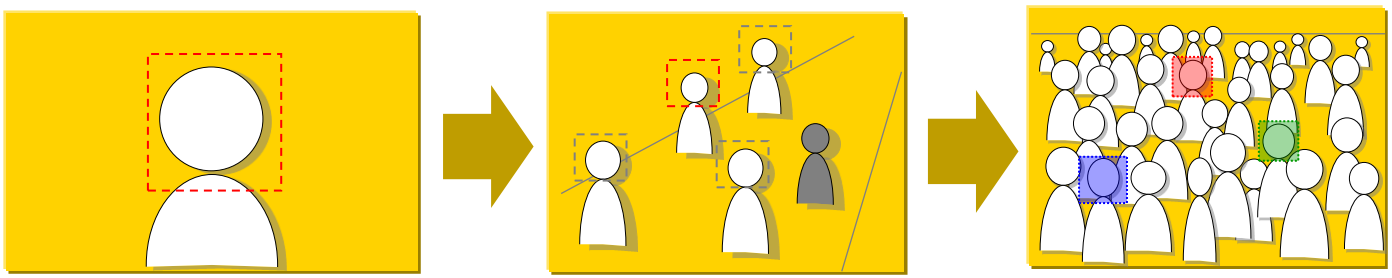
# 第4部のまとめ

## Big Data処理技術③　大規模な実世界映像データ解析技術

**カメラに見守られるのはSF世界だけの話なの？**

↳ **カメラによる人物追跡が一定の条件下で現実のものになりつつある**

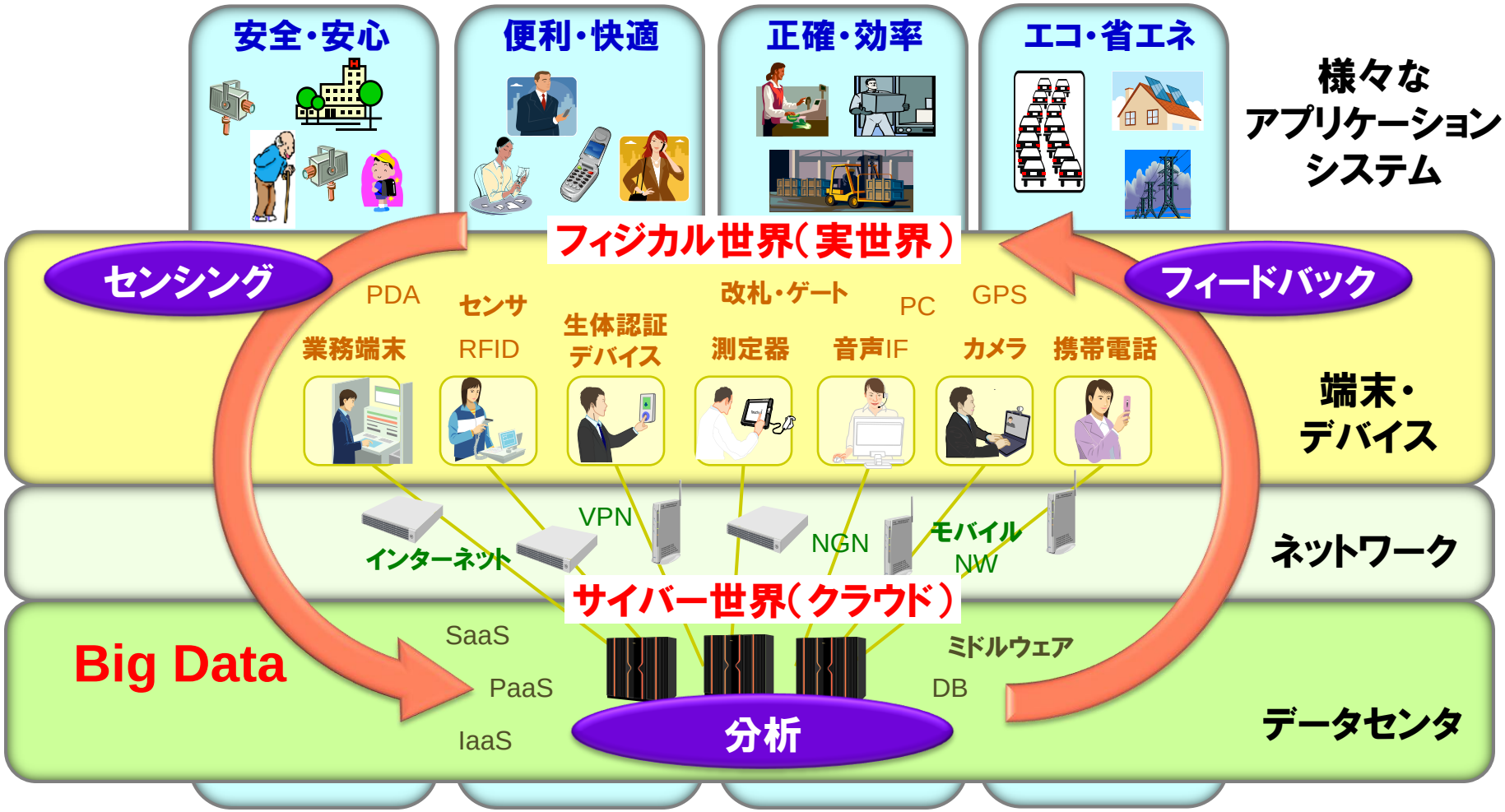
- 複数のカメラ映像を対応付けて解析することで人物の位置を特定・追跡
- 個人を特定する顔認証は、眼鏡や髭の有無、顔の表情や向きの違い、経年変化等があっても、同一人物だと判定
- カメラ映像の解像度が低くても、対象が顔やナンバープレート等に特定できるならば、解像度を高めること（超解像）が可能



ゲート（1人）　通路・施設内（10～100人）　群集（100人～）

# サイバーフィジカルシステム（Cyber-Physical Systems）

**実世界データもクラウドに取り込み、集まったデータ（Big Data）を解釈・分析することで、実世界の活動をより良くするフィードバックを生み出す**



# 【参考資料】より深い理解のための教科書・解説資料

- 徳永健伸「情報検索と言語処理」(言語と計算5) 東京大学出版会 (1999)
- Christopher D. Manning, Prabhakar Raghavan, Hinrich Schuetze: “Introduction to Information Retrieval”, Cambridge University Press (2008)
- William B. Frakes, Ricardo Baeza-Yates: “Information Retrieval: Data Structures and Algorithms”, Prentice Hall (1992)
- 伊藤直也・田中慎司 「Web開発者のための大規模サービス技術入門」 技術評論社(2010)
- 西田圭介「Googleを支える技術　巨大システムの内側の世界」 技術評論社(2008)
- 「情報処理」(情報処理学会誌) 第51巻 第12号 (2010年12月) 特集:画像認識技術の実用化への取り組み

# 【参考資料】NECの関連技術に関するもう少し詳しい説明

- 「NEC技報」第64巻 第4号（2011年11月）Network of Things特集号
  - http://www.nec.co.jp/techrep/ja/journal/g11/n04/g1104mo.html
  - 特に「M2Mサービスプラットフォームにおける大規模リアルタイム処理技術」
- 「NEC技報」第64巻 第3号（2011年9月）映像ソリューション特集号
  - http://www.nec.co.jp/techrep/ja/journal/g11/n03/g1103mo.html
  - 特に「NECの映像技術への取り組み」、「人の行動を「見える化」する動線解析技術と活用例」
- 「NEC技報」第63巻 第3号（2010年9月）パブリックセーフティを支える要素技術・ソリューション特集号
  - http://www.nec.co.jp/techrep/ja/journal/g10/n03/g1003mo.html
  - 特に「人物行動を把握する画像解析技術と適用例」、「顔認証技術とその応用」

Empowered by Innovation