データベースシステム
第13回

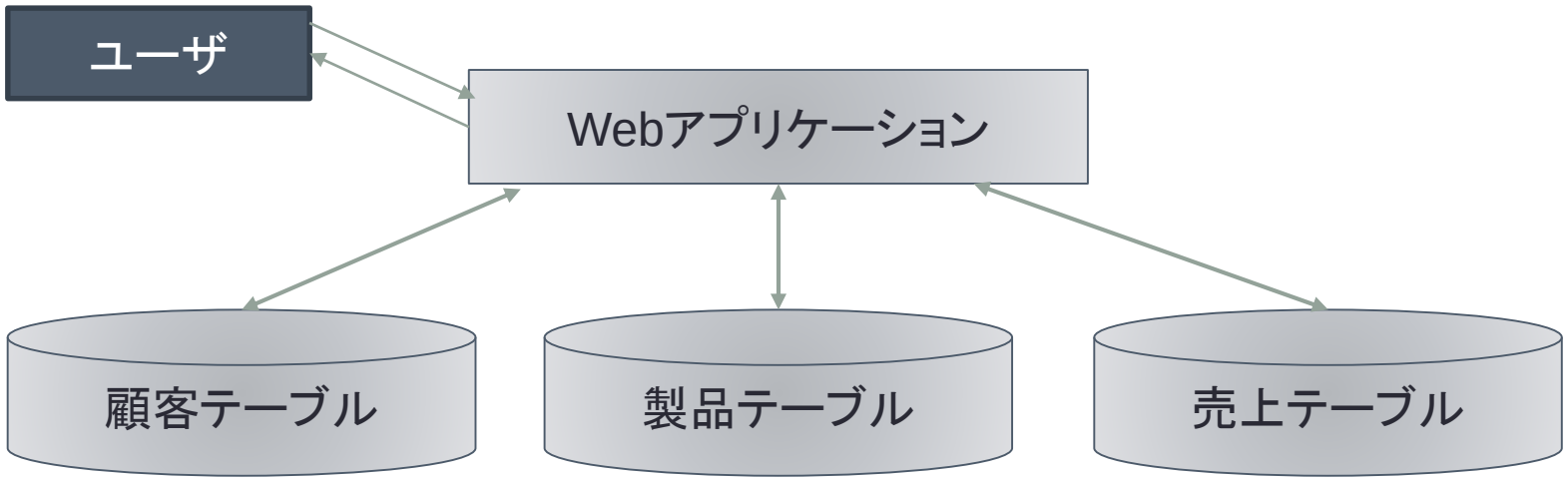
理工学部情報科学科
松澤　智史

本日の内容

- 分散データベース
- 様々なデータベース

分散データベース

- 複数のDBMSをネットワーク経由で結合する
- 利用者からは1つのDBMSのように見える
- 物理的に別の計算機で稼働させることが多いが，
  同一計算機上に複数稼働する場合もある



DBMSを分散させる理由

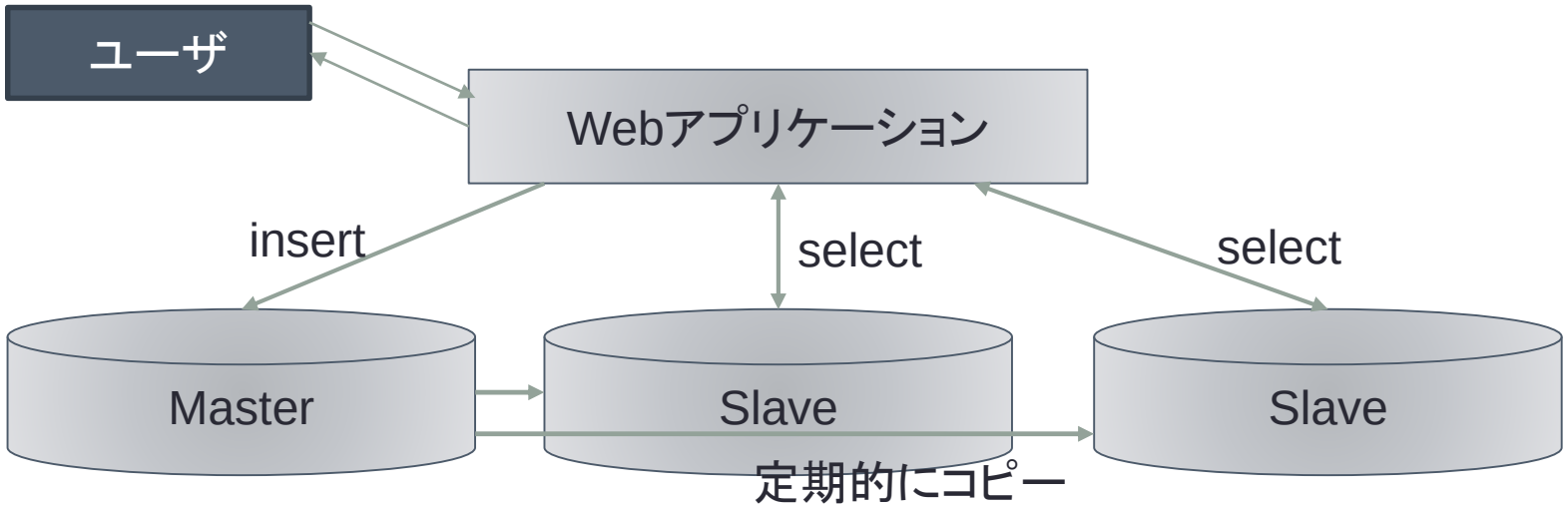
- 水平スケールアップによる性能アップが期待できる
  - ※水平スケール（計算機数増加）
  - ※垂直スケール（計算機単体の性能増加）
  - DBMSに限らず垂直スケールに比べて水平スケールはある閾値を超えると格段にコストパフォーマンスが良くなる
- 対故障性の増加
  - 設計によっては一つのDBMSに障害が起きた場合でもサービスを継続することができる

分散データベースシステムの透過性

利用者に分散データベースシステムを意識させずにデータを
取り扱う状態にすることで透過性には以下のものがある

- アクセス透過性
  - 利用者はネットワークに接続されているデータに対して同一方法でアクセスできる
- 位置透過性
  - データベースが物理的にどこに配置されていても利用者に意識させない
- 移動透過性
  - データの格納先が変更されても利用者に意識させない
- 分割透過性
  - 一つのデータが複数のサイトに分割して格納されていても意識させない
- 重複透過性
  - 一つのデータが複数のサイトに重複して格納されていても意識させない
- 規模透過性
  - OSやアプリケーションの構成に影響を与えることなくシステム規模を変更できる
- 並行透過性
  - 同時並行でデータベースの処理を行える

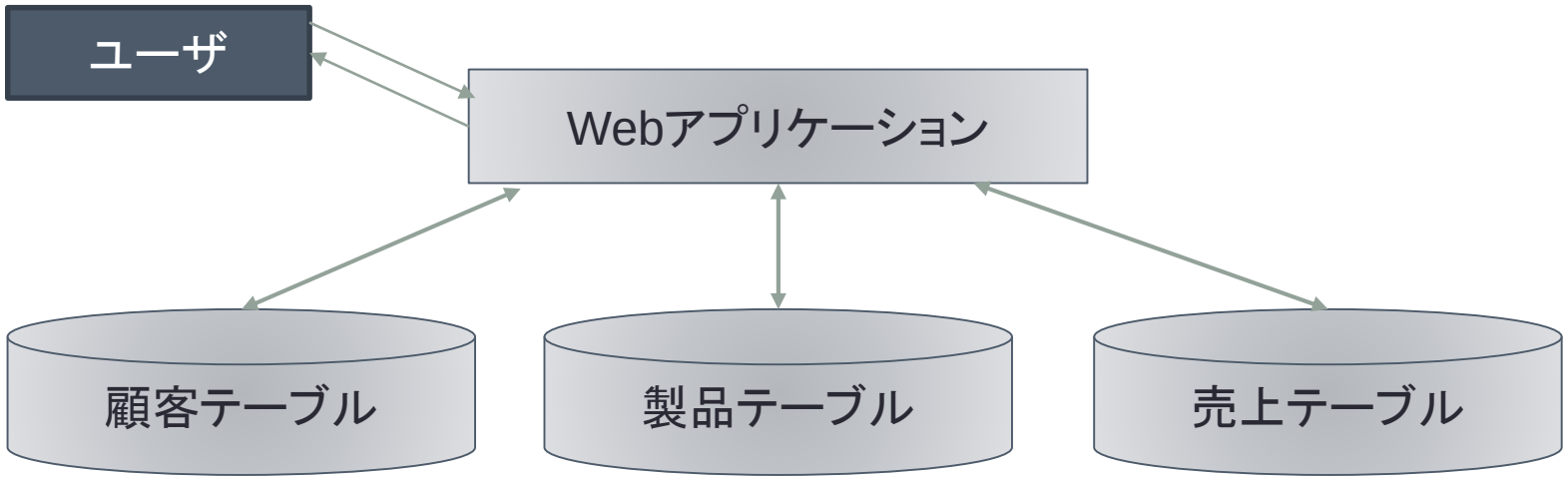
分散データベースの問題



- 3命令を同時に実行できることにより性能アップ

- MasterにInsertしたデータは即時にSlaveから読み込めない
- ACIDのC(Consistency)を満たすことができない

分散データベースの問題

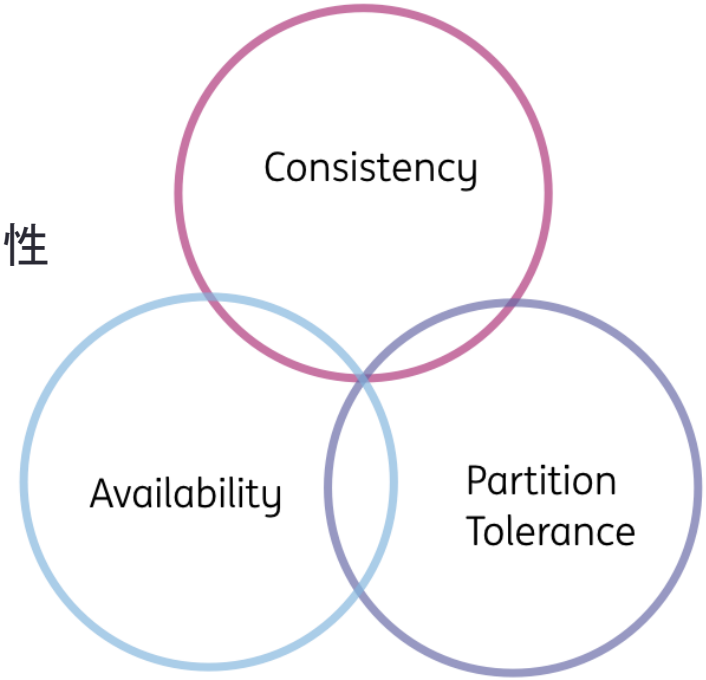


- ACIDを満たすことができる
- アクセスに関しては均等にすることができない

CAP定理

- 2002年に実証済みの定理
- 分散システムは，一貫性，可用性，分断耐性を同時に成立させることはできないという定理

Consistency:一貫性
Availability:可用性
Partition Tolerance:分断耐性



証明はhttps://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/

CAP定理

- Consistency
  - 一貫性、いつも最新のデータが読める
- Availability
  - 可用性 必ずデータにアクセスできる= 単一障害点がない
- Partition-tolerance
  - ネットワークの分断への耐性　複数サーバにデータの複製がある

- CとAを保証
  - ネットワーク分断がおきるとCかAを捨てなければならない
- CとPを保証
  - 単一障害点のある分散処理システムになる
- AとPを保証
  - 緩やかにデータ同期する
  - DNS

BASE

- 分散システムは基本的にPartition-toleranceを捨てられないため，CとPを保証かAとPの保証を選ぶことになる
- Consistencyを真面目に保証しようとするとスケールできない
- 分散システムにおいては別の特性を適用

- ACIDはDBにおけるトランザクションの特性を表す
- BASEはシステム全体の特性を表す

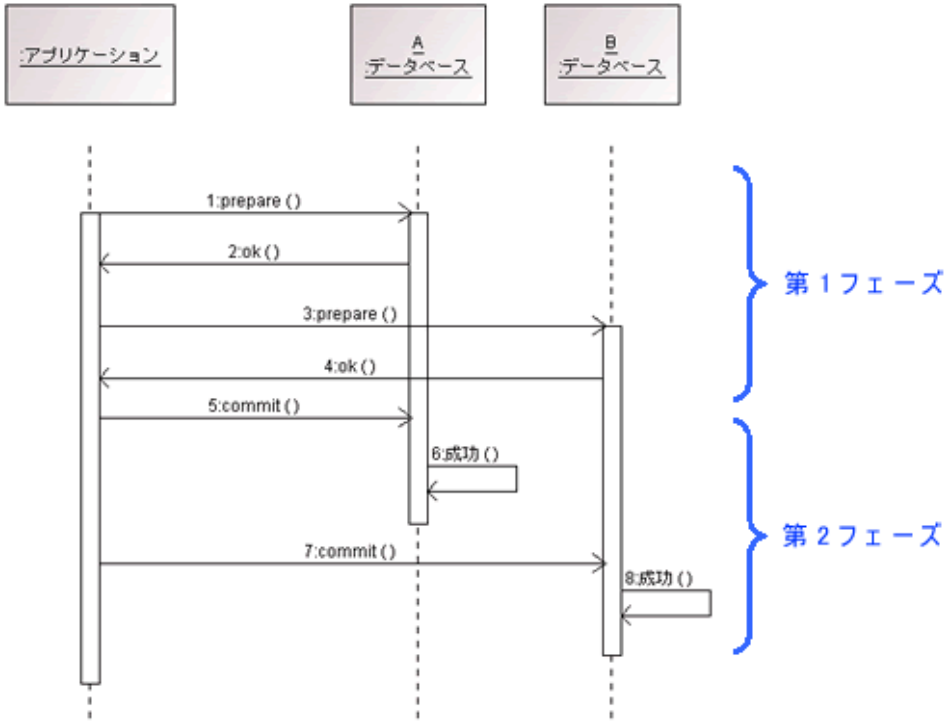
BASE

- BA
  - Basically Available
  - いつでも提供可能
- S
  - Soft-State
  - あるノードの状態は自律的ではなく外部からの情報により変化する
- E
  - Eventually Consistent
  - 最終的に整合性とれてればよい（結果整合性）

世の中の大半のサービスはBASE特性を満たす

2相コミット

- 遠隔地にあるデータベースシステム（サイト）の更新と確定を2相（2フェーズ）に分けて実行するためのプロトコル



- 手順1,3と5,7は同時に行われる

2相コミット

- 片方へのネットワークが不調の場合は，ロールバックを行う
- コミット可否問合せの返答からコミット指示までの間にネットワークが不調になるケースはタイムアウトによるロールバック等を行わないと中途半端な状態になる
- 可否問合せの返答からコミット指示までの時間は非常に短い時間なので，実用には十分耐えると考えてよい

レプリケーション

- 元のデータベース（オリジナル）のデータと同じ内容を持つデータベースを用意し，データのバックアップをする手法

- 非同期レプリケーション
  - 一定時間の間隔を置いて定期的にコピーを行う手法

- 同期レプリケーション
  - オリジナルのデータが更新・追加・削除される都度，コピーを行う手法

様々なデータベース

- NoSQL
- データウェアハウス
- オブジェクト指向データベース
- XMLDB

NoSQL

- Not Only SQL
- 広義にはRDBMS以外のデータベースを指す総称
- RDBMSでは扱いきれないケースに対応した
  様々なNoSQLがある
- データの格納や取得が高度に最適化されているものが多い
- 機能性を最小限にしているものもある

まとめ

NoSQLはRDBMSのような汎用性は欠くが,
制約された条件下ではRDBMSより高いパフォーマンスを持つ

NoSQLの種類

NoSQLは共通したモデルはない

- 代表的なモデル
  - KVS型データベース
  - ドキュメント型データベース
  - カラムファミリー型データベース
  - グラフ型データベース

KVS型データベース

- Key-Value Store型
- データをキーと値という単位で格納
- 検索はキーに対して行われる
- シンプルな構造なので高速
- キー以外の検索ができないなど機能的にRDBMSに劣る

ドキュメント型データベース

- ドキュメントと呼ばれる単位でデータを格納する
- ドキュメントの構造はあらかじめ決めておく必要がある
- とりあえず気軽にデータを入れておくという場合には便利
- 多くの場合，ドキュメントはJSON形式で格納されている

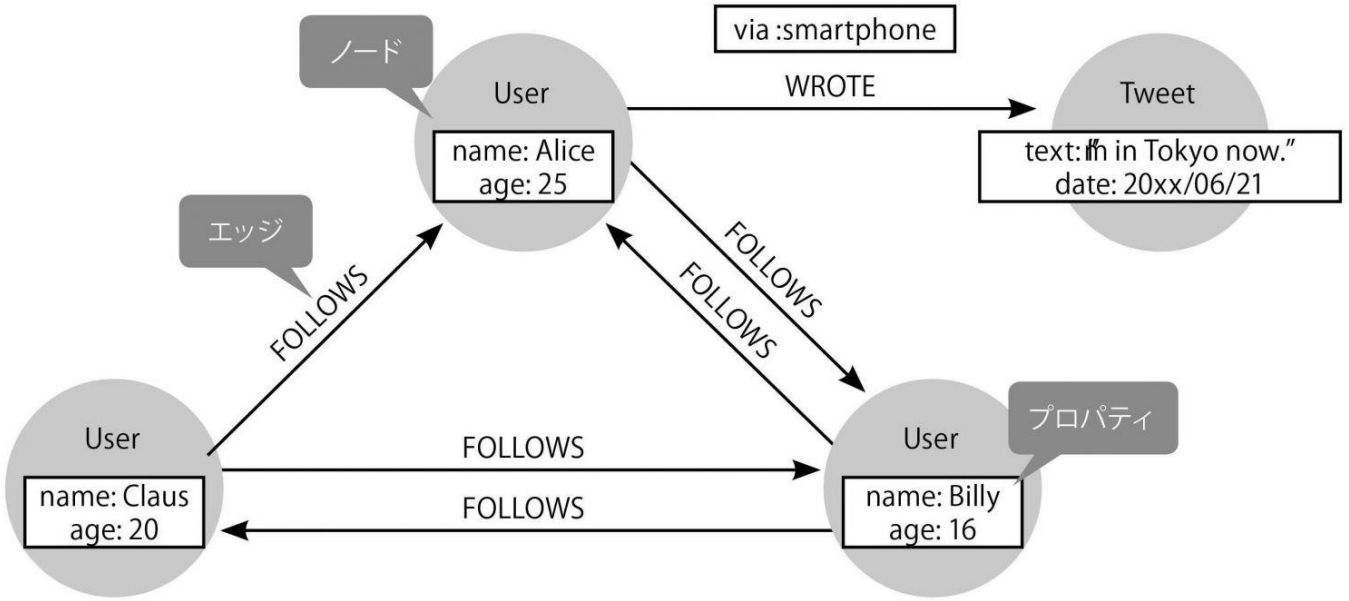
カラムファミリー型データベース

- テーブルをベースとした構造
- 複数の列をカラムファミリーという単位で管理される
- 列が非常に多くなるデータや
  多くの列から空になるデータに対して効率が良い
- 分散データベースとも相性が良い

	カラムファミリー1			カラムファミリー2		
	列1	列2	列3	列4	列5	列6
行1						
行2						

グラフ型データベース

- 各レコードが他のレコードへのリンクを持つようなデータを格納するデータベース
- ノード，エッジ，プロパティの3要素で構成される



NoSQLのデメリット

- トランザクションはサポートしていない
  - トランザクションの実装は技術的に複雑で，
    多くのNoSQLではほとんどサポートされていない
- データの性質がつかみにくい
  - テーブルのような明確な構造がない
- 主キー以外の検索に向かない
  - 主キー以外での検索は著しくパフォーマンスが落ちる

Neo4j

- グラフ型データベースの一つ

- Javaで実装され，オープンソースで公開・開発がされている

- Windowsなら

https://neo4j.com/download-center/

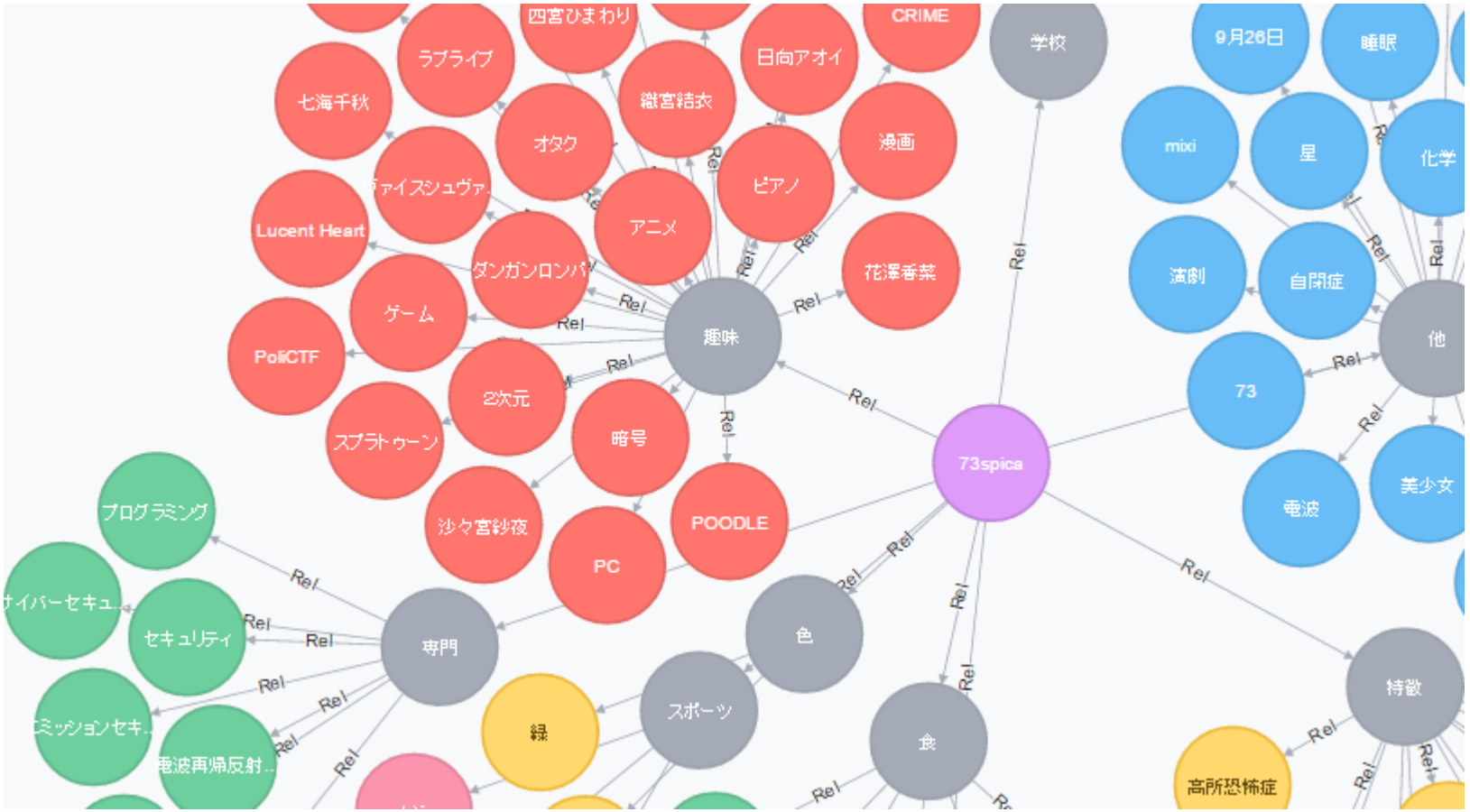
- Macなら

$ brew install neo4j

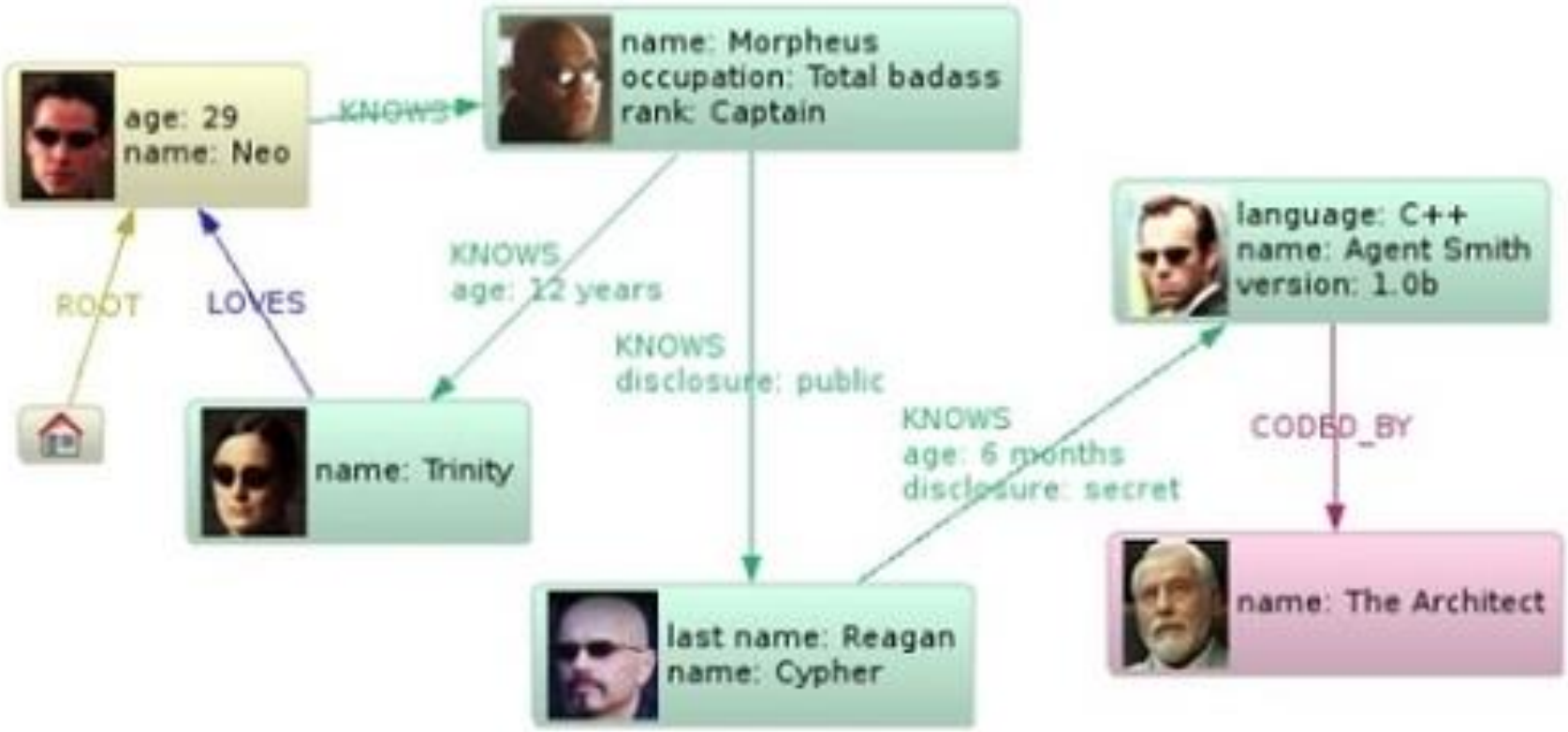
$ neo4j start

Neo4j

- ソーシャルグラフのようなデータベースを簡単に構築可能



Neo4j


age: 29
name: Neo
KNOWS
name: Morpheus
occupation: Total badass
rank: Captain
ROOT
LOVES
KNOWS
age: 12 years
KNOWS
disclosure: public
name: Trinity
language: C++
name: Agent Smith
version: 1.0b
KNOWS
age: 6 months
disclosure: secret
CODED_BY
last name: Reagan
name: Cypher
name: The Architect

まとめ

- 分散データベース
  - 透過性
  - CAP定理
  - BASE
  - 2相コミット

- NoSQL
  - KVS型
  - ドキュメント型
  - カラムファミリー型
  - グラフ型

質問あればどうぞ