データベース
アーキテクチャーの動向
と使い分け

2015/4/21 QConTokyo

Bashoジャパン　上西康太

自己紹介

- @kuenishi
  - Github, Twitter, etc
- 分散システム歴7年
- Bashoジャパンの方から来ました
- Riak CSの開発
- スピリチュアルな話をします

提供

ACID

Atomicity
Consistency
Isolation
Durability

Atomicity

Consistency

Isolation

Durability

Durability

"The ACID property which guarantees that transactions that have committed will survive permanently."

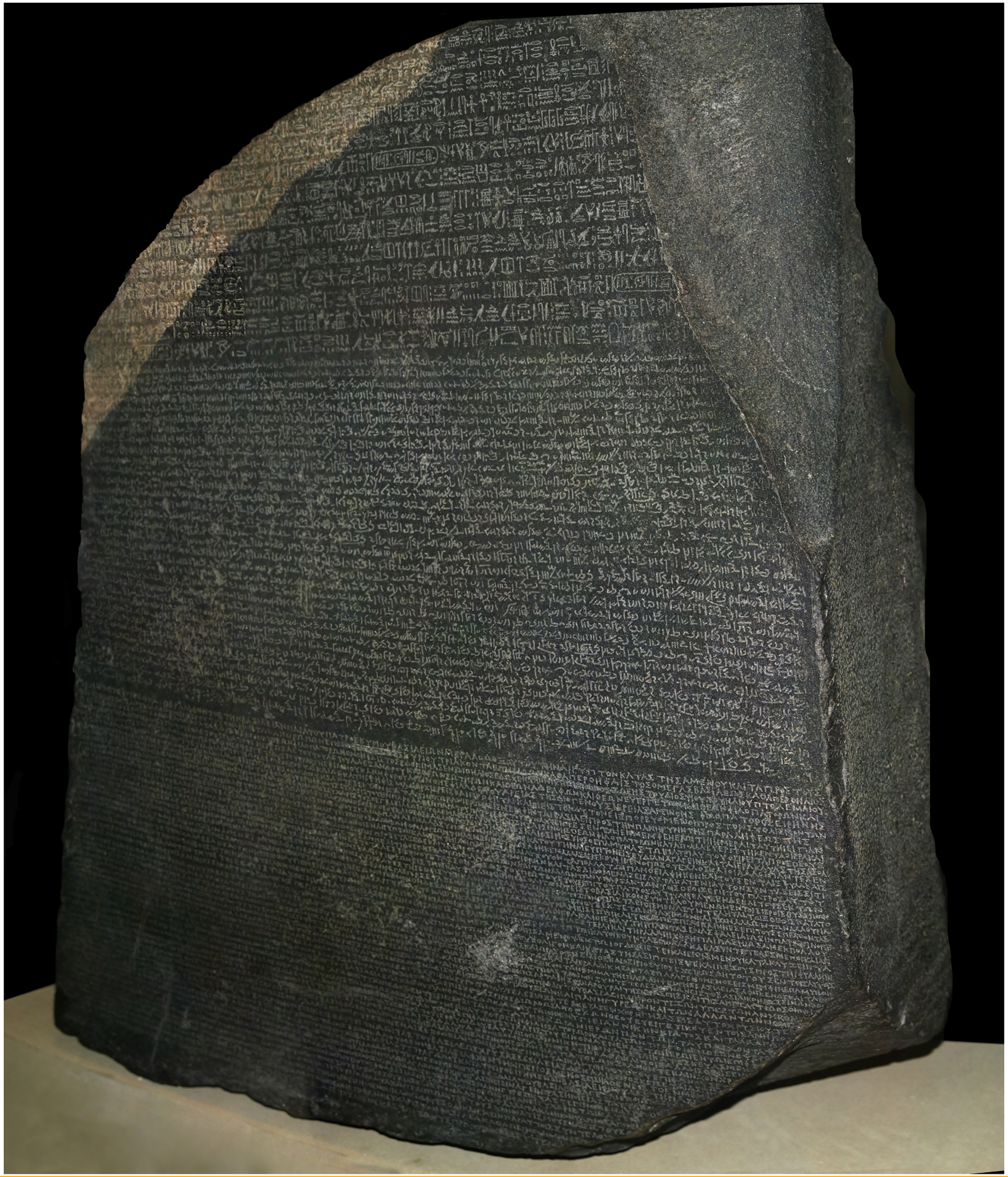
http://en.wikipedia.org/wiki/Durability_(database_systems)

Permanently?

永続化は

（思っていたより）

タイヘン



basho

https://www.flickr.com/photos/bathyporeia/9086009348/

諸行無常

सब्बे संखारा अफिच्चा

大般涅槃経　諸行無常偈

生滅の法は苦であるとされているが、生滅するから苦なのではない。生滅する存在であるにもかかわらず、それを常住なものであると観るから苦が生じるのである。

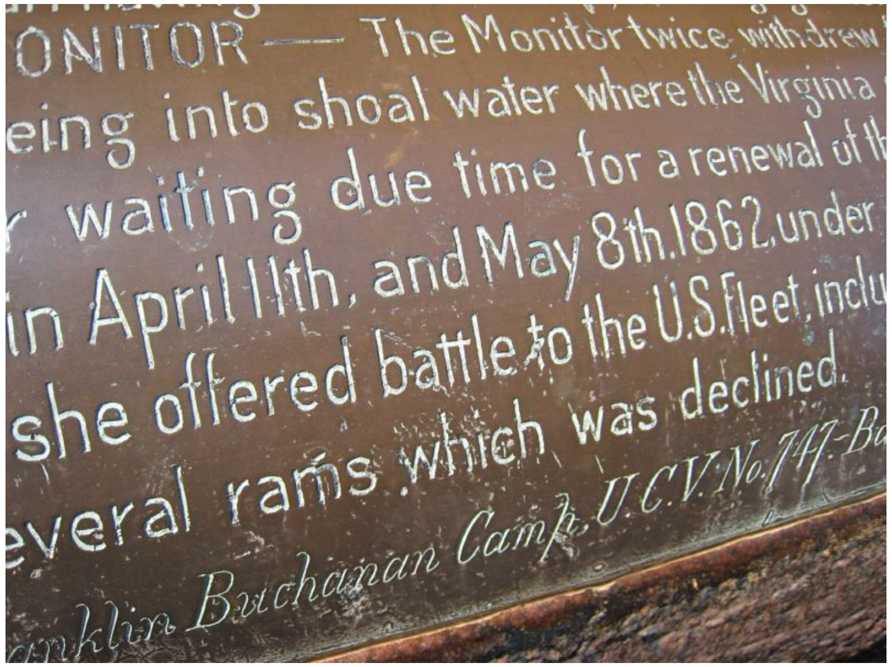
Wikipedia 「諸行無常」

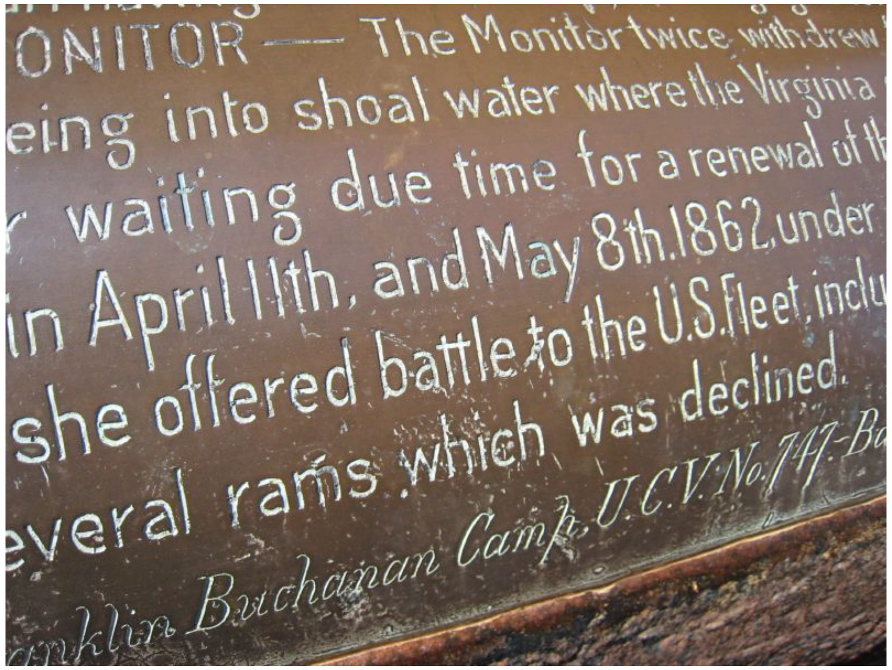
記録媒体は壊れる

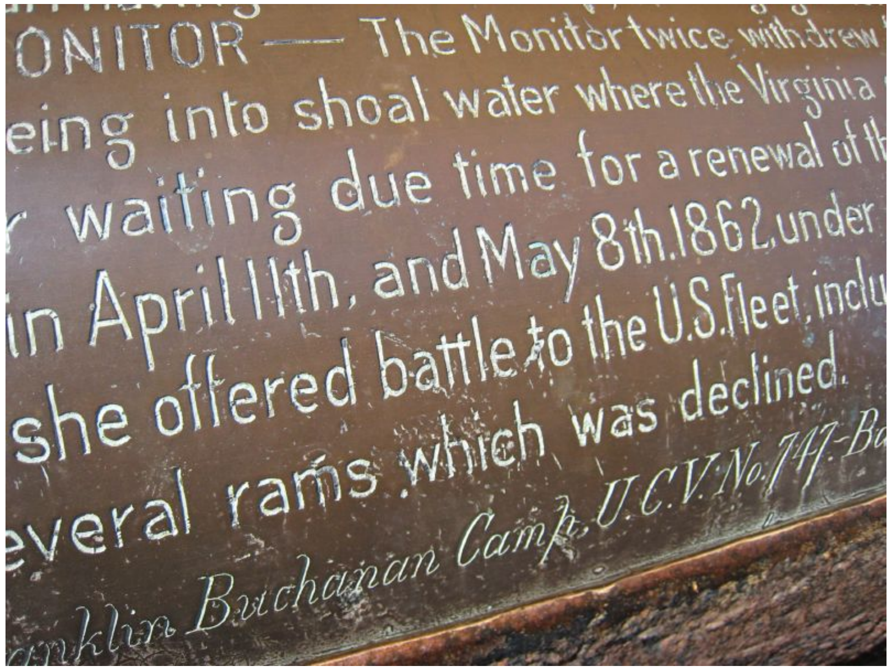
記録媒体は壊れる

なら

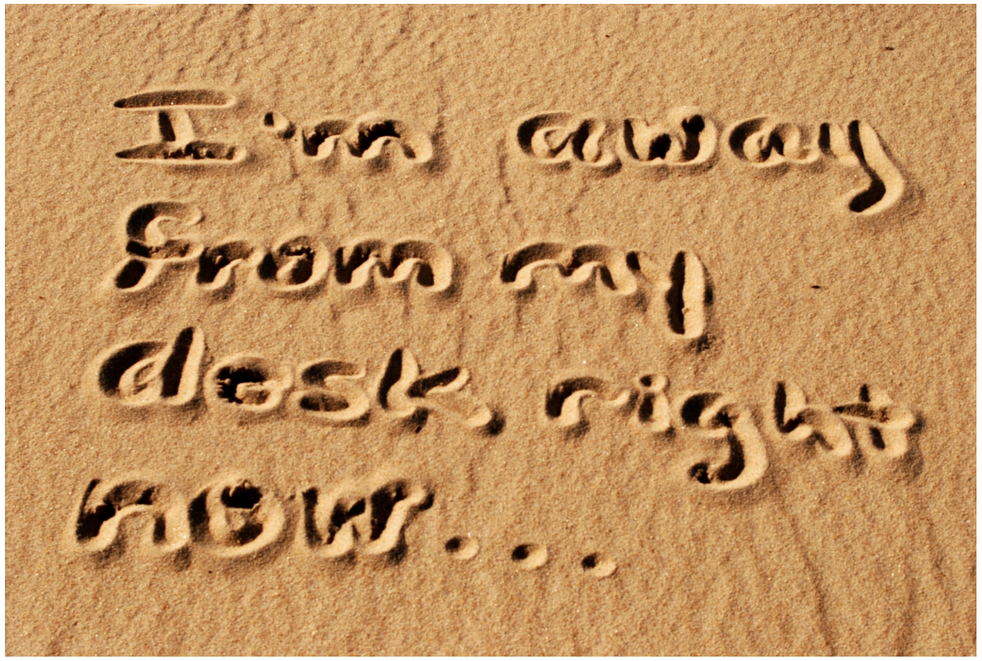
複製すればよい

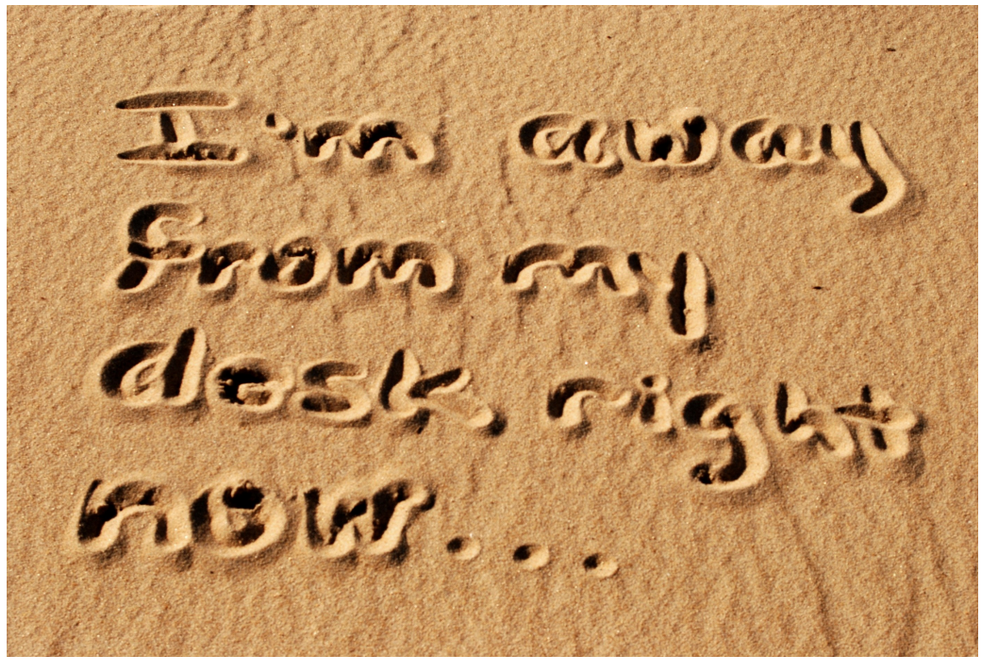

ONITOR — The Monitor twice withdrew
eing into shoal water where the Virginia
waiting due time for a renewal of th
in April 11th, and May 8th.1862, under
she offered battle to the U.S. fleet, inclu
several rams which was declined.
anklin Buchanan Camp, U.C.V. No. 747-Ba


ONITOR — The Monitor twice withdrew
eing into shoal water where the Virginia
waiting due time for a renewal of th
in April 11th, and May 8th.1862, under
she offered battle to the U.S. fleet, inclu
several rams which was declined.
anklin Buchanan Camp, U.C.V. No. 747-Ba


ONITOR — The Monitor twice withdrew
eing into shoal water where the Virginia
waiting due time for a renewal of th
in April 11th, and May 8th.1862, under
she offered battle to the U.S. fleet, inclu
several rams which was declined.
anklin Buchanan Camp, U.C.V. No. 747-Ba

理想

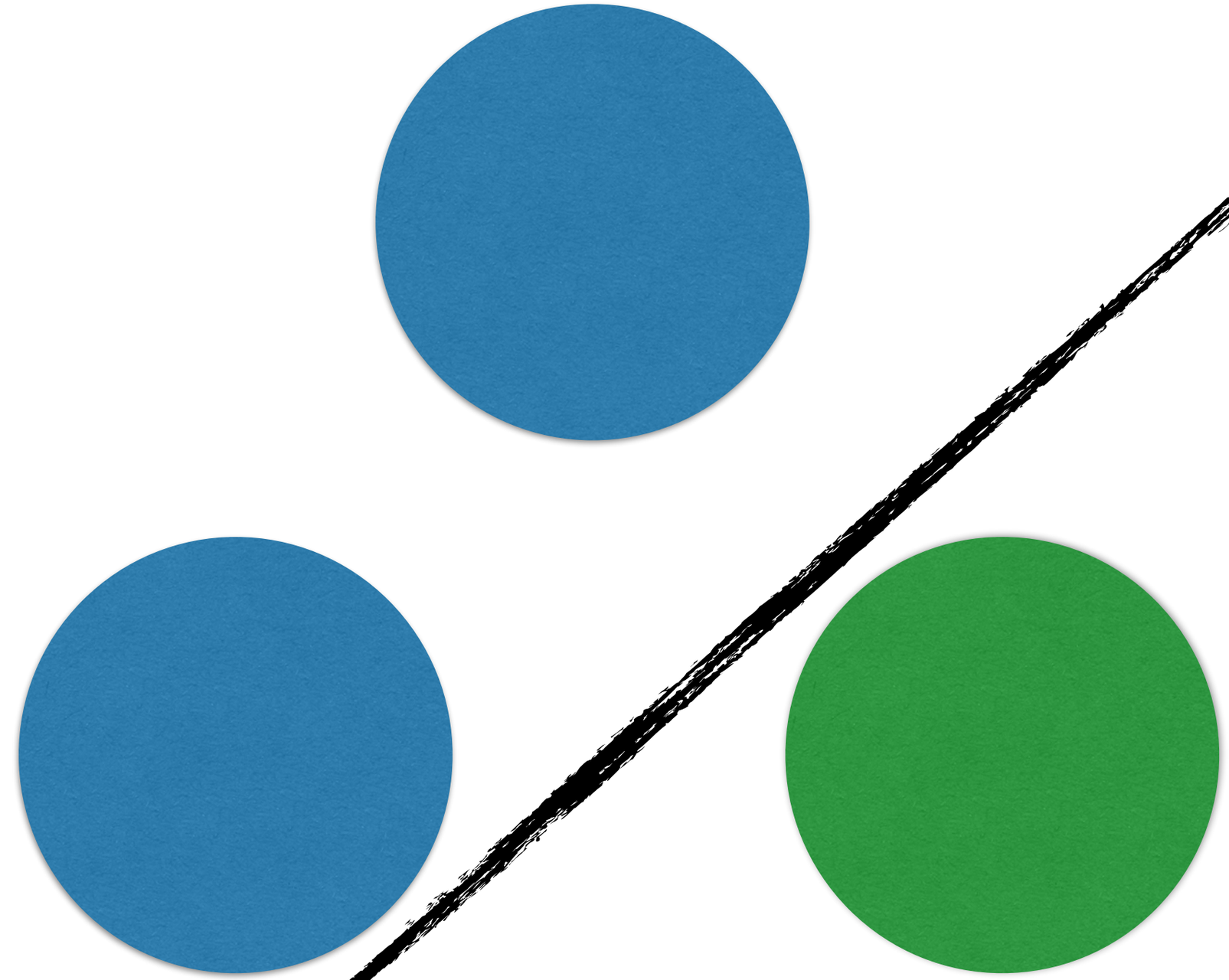




現実

https://www.flickr.com/photos/leeziet/3021612079/

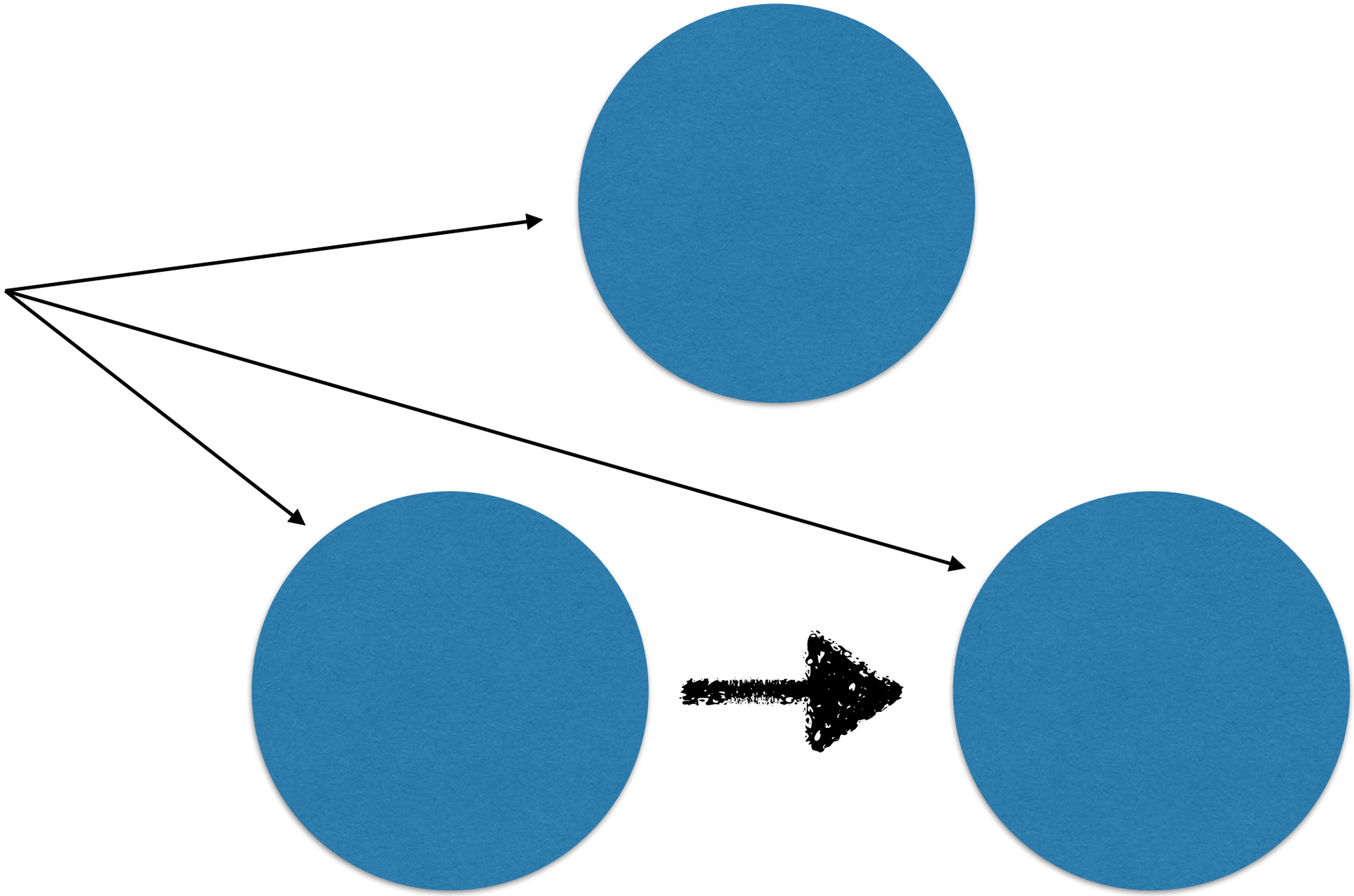
https://www.flickr.com/photos/pranavbhasin/6109327813/



CAP定理

レプリケーションは難しい

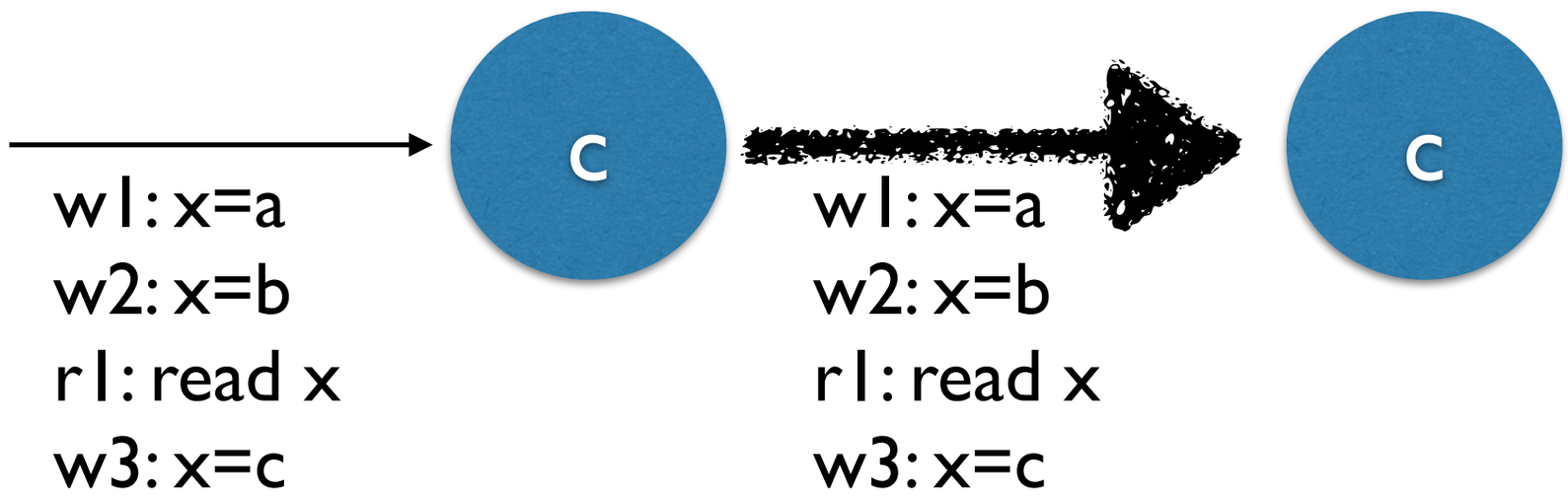
- CAP定理
  - 複数のノードが保持している、最初は同じオブジェクトに変更の列を送り続ける
    - メッセージが到達しないときに、全てのノードが同じ変更の列を保持することができない
- 難しさの根源は故障単位を分けたこと
- 別のものが同一であることを保証する



凡例

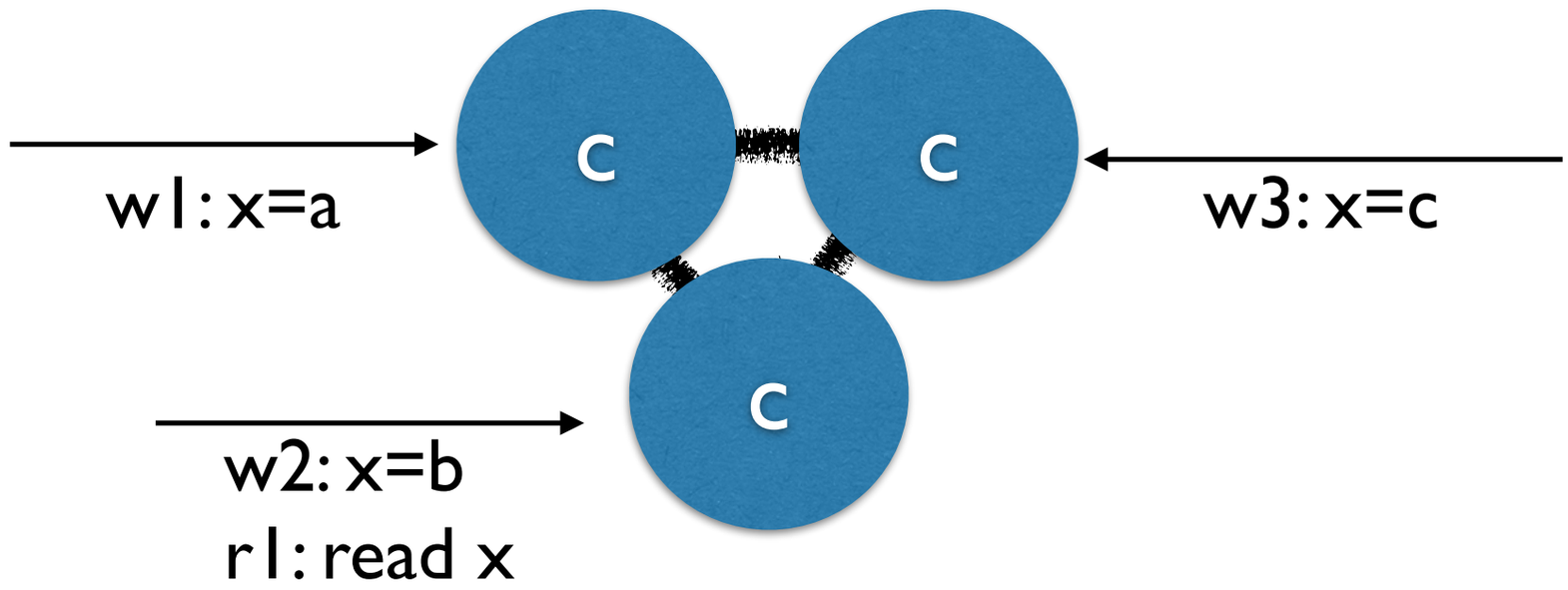
解法その０: Master-Slave

- 更新の命令列が唯一であることを保証する
- スレーブ、レプリカは、決定された更新の命令列を受け取ってローカルに反映するだけ
- マスター不在では何もできない



解法その1：コンセンサス

- 異なる実体が同じ状態を持つことがゴール
- レプリケーションはいわゆる分散合意問題
- これを解く通信方式を、コンセンサスプロトコルという



愚者は経験に学び、

賢者は歴史に学ぶ

1990年代 RDBMS普及期

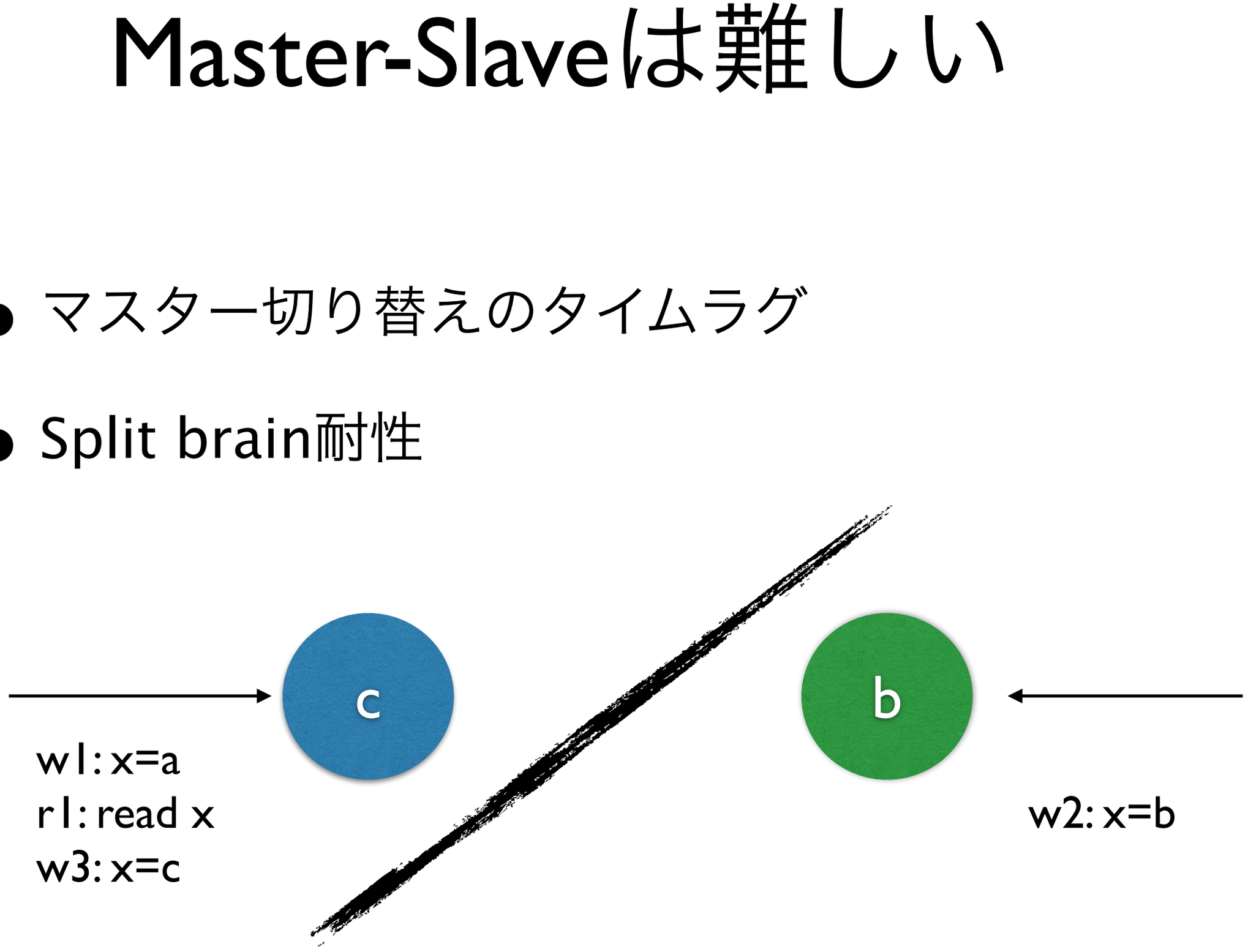
- OSが意識することはなく、カーネル以下で複製
- ディスクレベルで同期型。片方が故障したら全系故障
- ハードディスクとネットワーク帯域が希少
- RAID, SANでまとめて一元管理、運用
- トランザクション、クエリ処理の基礎技術の確立

2000年代Web時代 (1/2)

- アプリケーション、ミドルウェアのレイヤ(TCP/IP) でレプリケーションが一般的になる
- ネットワークレベルでの同期型。片系が故障しても動作継続
- Readをスケールアウトできるタイプのものもいくつか登場
- MasterからSlave (Replica)へ差分を流すタイプが主流
- MySQLのbinlog, GFS (BigTable), HDFS (HBase)

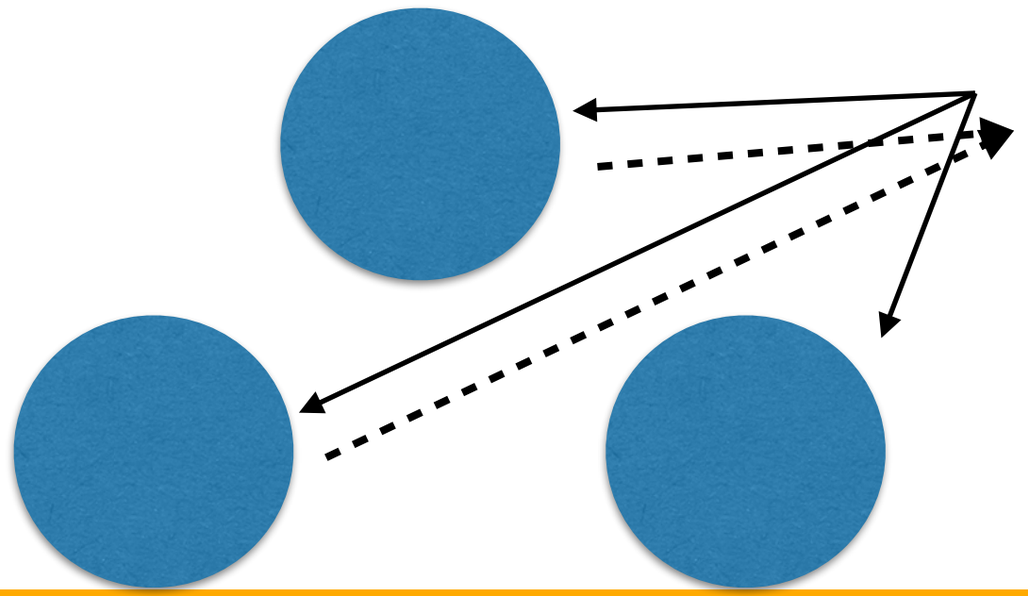
Master-Slaveは難しい

- マスター切り替えのタイムラグ
- Split brain耐性



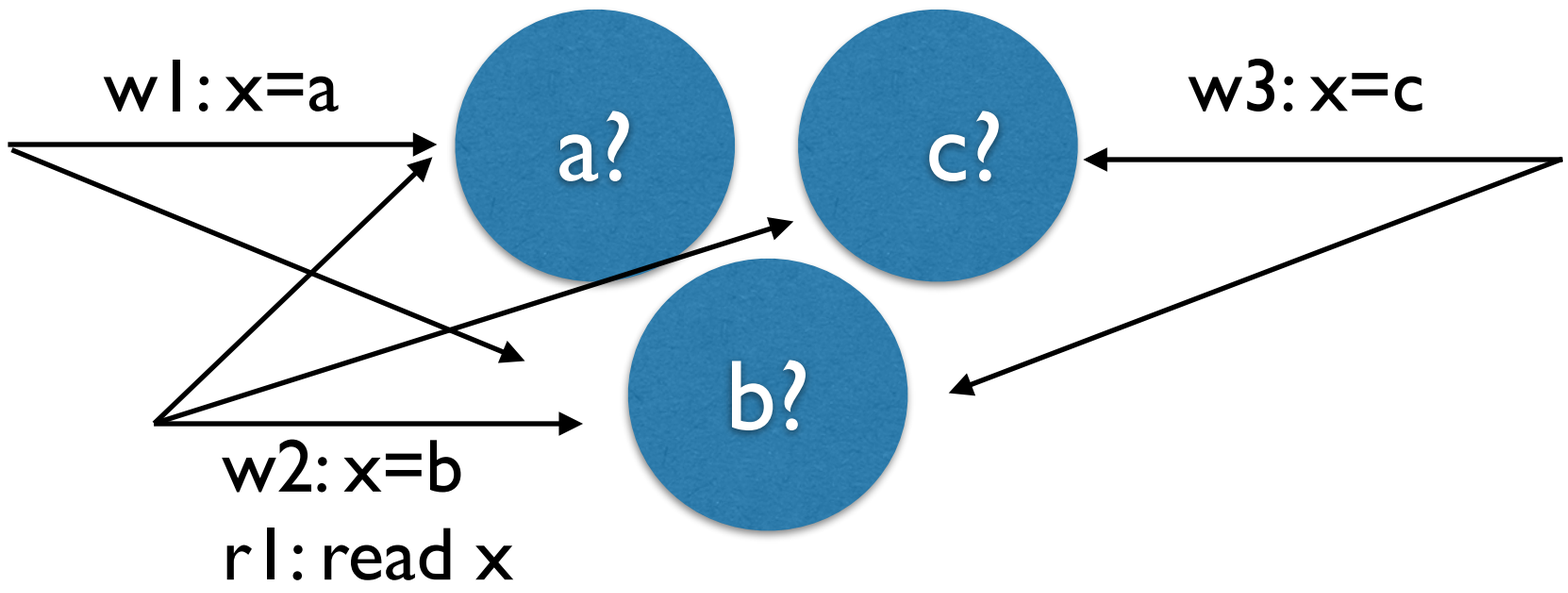
2000年代 Web時代 (2/2)

- Webシステムの複雑化、巨大化
- コンセンサス型のレプリケーションの実用化
- ネットワーク分断が起きてもなんとかなる
  - Dynamo, Chubby, ZooKeeper, SQL Server (2008?~)
  - Paxos (1989), Quorum (1979) など



Quorum: コンセンサスは難しい

- 上書きを許容するナイーブなプロトコル設計では簡単にデータが破壊される
- いつでも誰でも故障するし黙るし復活する...という現実世界では実用的ではない



Paxos: コンセンサスは難しい

- 2フェーズの合意プロトコル
  - Proposer (値を提案する人) を多数決で決定
  - Proposed Value (提案された値) を多数決で決定
    - 提案内容に順序番号を振って新旧管理する
- いつでも誰でも故障するし黙るし復活する...という制約下でも、無限に時間があって過半数が故障していなければ合意する
- 実装は難しいが、頑張ればなんとかなる

コンセンサス型
レプリケーションの分類

- CP型
  - 複製間の同一性を保障するタイプ
  - Paxos, Raftなどのアルゴリズムを採用
  - ネットワーク分断したときに多数側のネットワークにいるノードしか利用できない
- AP型
  - 複製が完全に一致していないことを許容する
  - Vector ClockやCRDTにより因果整合性を保障（もしくは単なるタイムスタンプ）
  - ネットワーク分断しても、すべてのノードで利用可能

レプリケーションからみたデータベースの分類

- Master-Slave型
  - 実装がシンプル、高速
- コンセンサスとMaster-Slaveのハイブリッド型
  - マスター選出にコンセンサスプロトコルを採用
  - レプリケーションそのものはMaster-Slave
- コンセンサス型
  - レプリケーションにもコンセンサスプロトコルを利用
  - マスター故障に伴うダウンタイムがない

レプリケーションからみたデータベースの分類

- Master-Slave型
  - MySQL, PostgreSQL
- コンセンサスとMaster-Slaveのハイブリッド型
  - MongoDB, HBase, Redis
- コンセンサス型
  - Riak, Cassandra (いずれもAP, CPモードあり)
  - CouchBase (CP型)

2010年代 クラウドの時代

- NewSQLといわれる分類の登場
  - FoundationDB, NuoDB
  - 既存のNoSQLがSQL(-likeなもの)を実装する場合
    - NewSQL の中には ACID を満たす(?)ものも
- 複数データセンターでのレプリケーションが必須に
  - ネットワーク分断やレイテンシがより重要な課題に
- MPPがOLAPのワークロードで実用化（分散クエリ処理）
  - BigQuery, Impala, PrestoDB

復習: データベースの要素技術

- クエリ処理の最適化
  - SQLを解析して、統計情報から最適なクエリプランを作成・実行する
    - そのためのデータ配置、インデックス戦略
- トランザクション処理の最適化
  - Anomalyを排除し整合性を保証しつつ、なるべく速くデータを更新していく

クラウド時代の
データベースの要素技術

- 分散環境でのクエリ処理の最適化
  - MPPで並列処理、故障時は投機実行
  - Nested Columnarでデータ配置を局所化
- 分散環境でのトランザクション処理の最適化
  - 分散しているのにAnomalyを排除？整合性を保証？
  - ノード間だけでなくDC間の整合性も課題

分散DBでACID

- 現実的な設計はひと通りしかない
  - コンセンサスによるMaster選出＋M/Sレプリケーション or CP型のレプリケーション
  - タイムスタンプの同期を保証する仕組み
  - 楽観的並行性制御
- MegaStore (2011), Spanner (2012)

分散DBでACID

- そのままトレードオフになる
  - ネットワーク分断時の可用性
    - タイムスタンプ管理ノード？→SPOF
    - TSOをOLTPのワークロードにそのまま応用したらアボートの嵐

Writeをスケールさせる

- Paxosなどは、コンセンサスメンバを固定しなければならない
- 楽観的レプリケーション (2005)
  - 強い整合性を満たさないが、特殊な状況下で別種の整合性を保証する仕組み（結果整合性など）
    - DNSなど
  - 演算子の順番が入れ替わっても整合するようなデータ管理の仕組み
    - Vector Clocks, CRDT, boom

CRDT

- 楽観的レプリケーションを簡単にするデータ型とレプリケーション技術のひとつ
  - Conflict-Free Replicated Data Types
  - w1(w2(x)) == w2(x1(x)) を満たすような
    データ型・データ構造と演算子の組み合わせ
  - ネットワーク分断時でも更新、読み出し可能

CRDT例: G-Counter

- merge
  - aが持っているデータ: {a: 1, b: 1, c: 2}
  - bが持っているデータ: {a: 0, b: 2, c: 0}
    - x => {a: 1, b:2, c:2} => 5
- update
  - aが {increment, 3} を受けとると{a: 4, b: 1, c: 2}
  - C < x という条件演算を処理できる

CRDT例: PN-Counter

- merge
  - {a: {1,-1}, b: {1,0}, c: {2,0}}
  - {a: {0,0}, b: {2, 0}, c: {0, -2}}
    - => {a: {1,-1}, b:{2,0}, c:{2,-2}} => 2
- update
  - aが {increment, 3} を受け付けると
  - {a: {4,-1}, b: {1,0}, c: {2,0}}
- c < x という条件演算を処理できない

CRDT例: OR-Sets

- merge
  - a:{“foo”:false, “bar”:true, “baz”:true}
  - + b:{“bar”:true, “baz”:false}}
    - => {“foo”:false, “bar”:true, “baz”:true}
    - => [“foo”]
- update
  - add: a:{} => +”foo” => a:{“foo”:false}
  - remove: a: {“foo”:false} => a: {“foo”:true}

CRDT

- ネットワーク分断時でも更新、読み出し可能
- Writeの ”並行処理” が可能になるデータ
- 値を計算する方法に一定の制約がある
- 効率的なCRDTの実装はまだ研究段階

予想: 2010年代後半

- 実装面では工夫の余地があり、ACIDを満たそうとする分散DBはまだまだ登場する
  - 分散を考慮した並行性制御
  - データセンターを跨ぐCP型レプリケーション、トランザクション
  - 運用ノウハウの普及
- NoSQLデータベースの採用はしばらく続くだろう（いくつかは淘汰されるだろう）
- 強い整合性と楽観的レプリケーションのハイブリッド型データベースが登場するだろう

ハイブリッド型データベース

- OLTP向けのデータベースが更新処理の可用性と性能を目的に楽観レプリケーションを導入し始めるだろう
- 業務処理のうち必ずしも全てが強い整合性を必要としているわけではない
- アプリケーション側で強い整合性と楽観的レプリケーションを使い分けることでパフォーマンスを出すことが期待できる
- インターフェースとしてはSQL, JDBCが残るのではないか

予想: 2020年代

- わかりません
- ハイブリッド型データベースの安定した実装が登場、普及
- SSDやNVMが普及しIOはボトルネックではなくなる
  - Shared Nothing型のスケールアウト型DBではなくなるのでは
- メモリバンド幅またはCPUがボトルネックになる
- 新しいハードウェアが登場すれば、またどうなるか分からない
- 2000年代の技術Xが再登場

まとめ

- 2000年ころから、データベースの2大技術要素に、徐々に分散システムの技術が要素技術として必須になっていった
- 2015年までに登場したデータベースのレプリケーション技術について簡単に解説
- 2015年後半には、CPとAPのレプリケーションを同じインターフェースで使い分け、ACIDを満たす分散データベースが登場するだろう
- 2020年の大まかな妄^H予想を

※Disclaimer: この資料の内容は上西の個人的な予想であり、何らかの未来を保証するものではありません

参考文献

- Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.
- James C. Corbett et al., 2012. Spanner: Google's Globally Distributed Database.
- Yasushi Saito and Marc Shapiro. 2005. Optimistic Replication.
- Peter Bailis and Kyle Kingsbury. 2014. The Network is Reliable.
- Peter Bailis et al. 2014. Coordination Avoidance in Database Systems.
- Mihai Letia et al. 2010. Consistency without Concurrency Control in Large, Dynamic Systems.