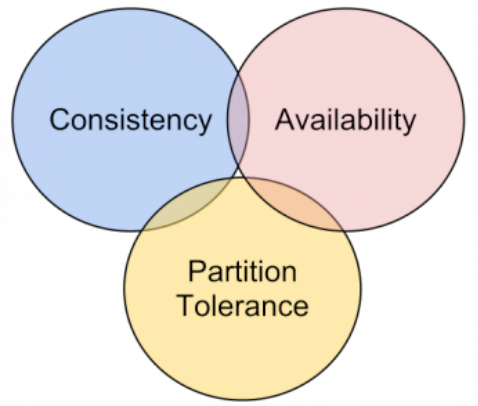


Conflict-free replicated data types: распределенные данные в деталях

Max Klymyshyn

Tech Lead at Takeoff Technologies

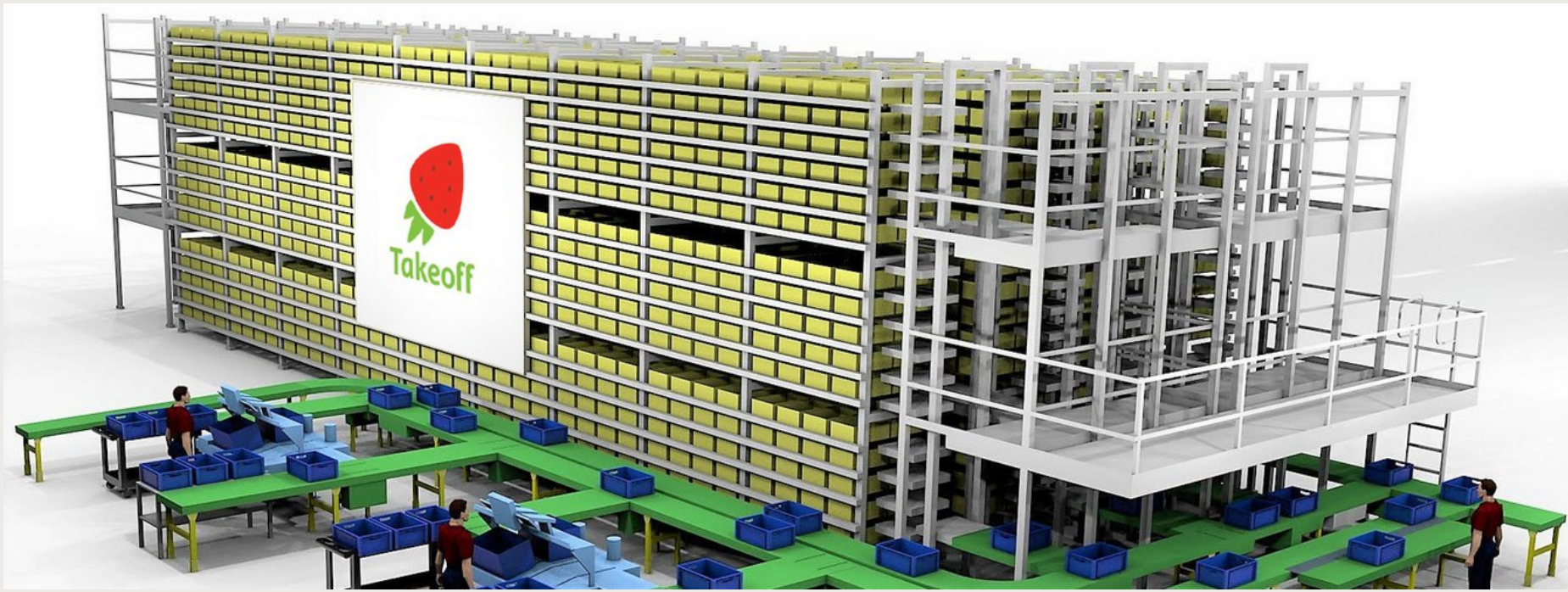
Работа

- Tech Lead, Takeoff Technologies, 2017–
- CTO @ ZAKAZ.UA and CartFresh, 2012
- Team Lead @ oDesk (now Upwork), 2010
- Project Coordinator @ 42 Coffee Cups, 2009

Сообщество

- соорганизатор PapersWeLove Kyiv
- соорганизатор PyCon Ukraine
- сооснователь KyivJS
- сооснователь LvivJS
- соорганизатор PiterPy
- соорганизатор Hotcode
- судья UA Web Challenge

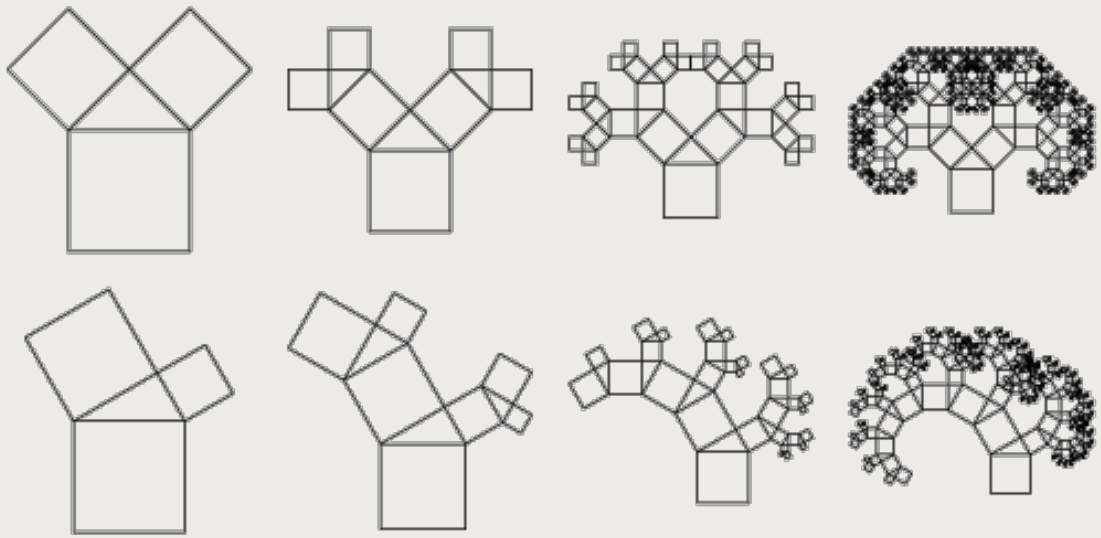
Takeoff Technologies



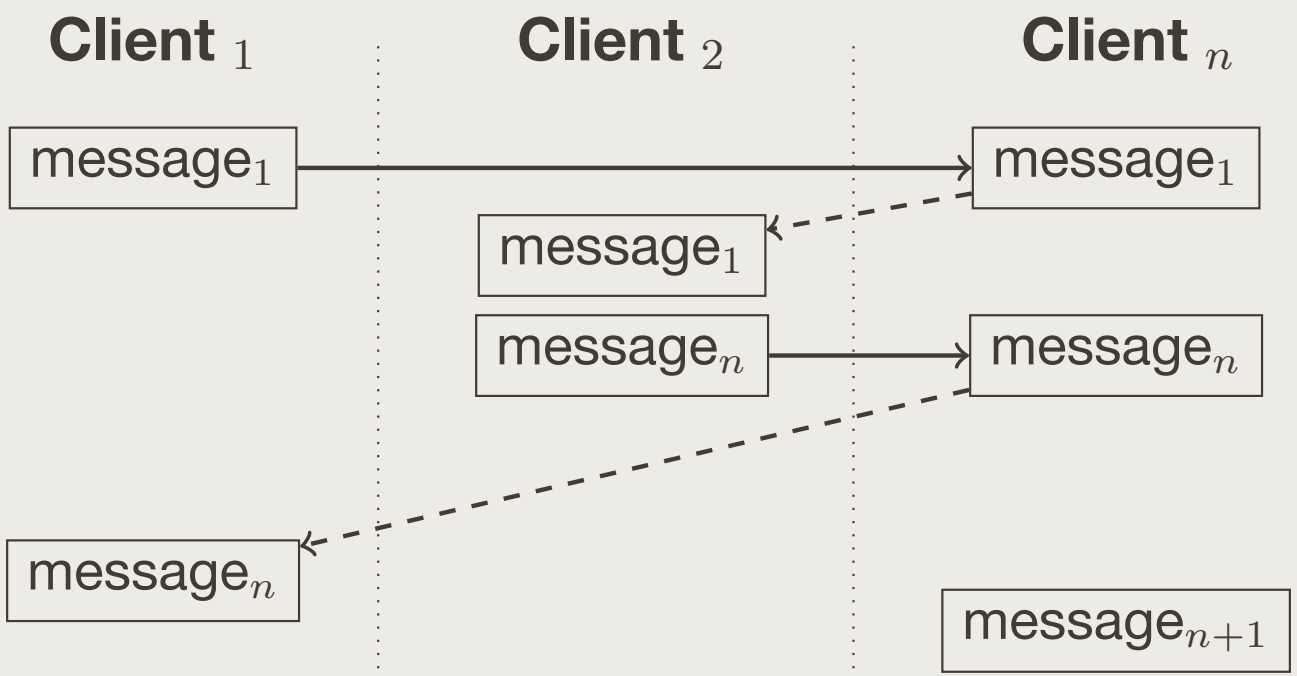
Распределенная система

The best material model of a cat is another, or preferably the same, cat.

Norbert Wiener



Схема



Server **is** just a client

Ограничения сети

- Задержки
- Потери TCP пакетов или сообщений
- Нарушение порядка доставки сообщений
- Дубликаты
- Head-of-Line blocking

Чего мы хотим добиться?

- Чтобы все участники получали сообщения
- Избежать потери данных (data losses)
- Чтобы было понятно как разрешать ограничения сети

Три проблемы распределенних систем

- **divergence** (расхождение): $\cup_{i=1}^{n} o_i^1 \neq \cup_{j=1}^{m} o_j^2$
- **causality-violations** (нарушение порядка): $o_1 \rightarrow o_3$
- **intention-violations** (нарушение намерений): $o_1 || o_2$

Модели целостности

CONSISTENCY MODELS

Модель консистентности

- обещание или **контракт** между разработчиком и системой, на которой будет запущен его код
- следование правилам системы будет гарантировать **определенные свойства** данных

Strong Consistency

- “strong consistency” все изменения применяются последовательно (т.е. **sequential** with **no operations overlap**) в глобальном порядке
- Узкое горлышко : маленькая пропускная способность и большие задержки (из-за консенсуса): RAFT/PAXOS/Zab
- можновы брать либо availability, либо partition-tolerance

Weak Consistency

- никаких гарантий после записи
- Read-Your-Writes hack

Eventual Consistency

- $\forall i, j : f \in c_i \Rightarrow f \in c_j$, Convergence and Termination properties
- RIAK, MONGO, Cassandra, Couch

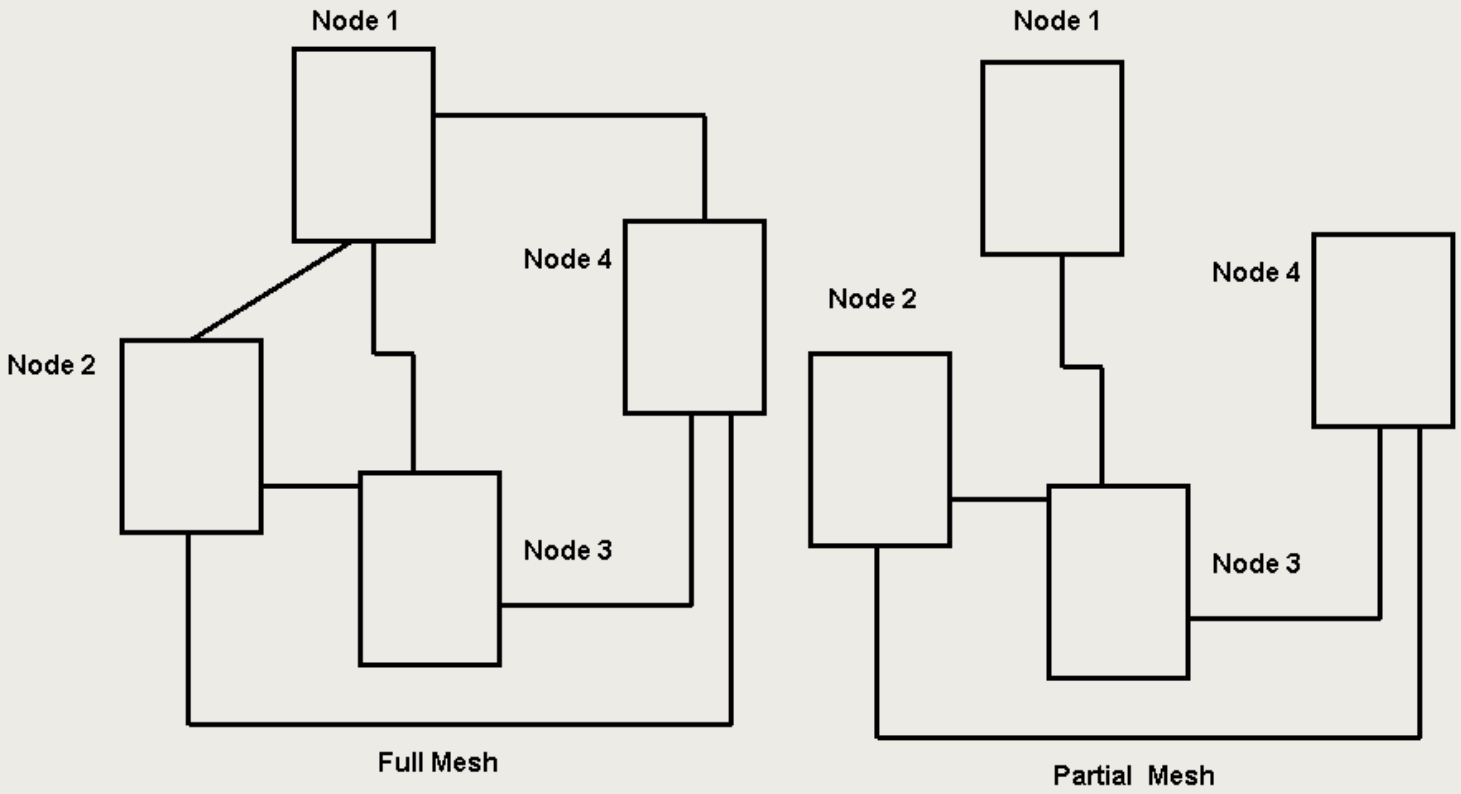
Strong Eventual Consistency

- ... То же что и Eventual Consistency
- and $\forall i, j : c_i = c_j \Rightarrow S_i \equiv S_j$: корректные реплики, получившие одинаковы апдейты будут иметь эквивалентное состояние
- CALM: консистентность как логическая монотонность

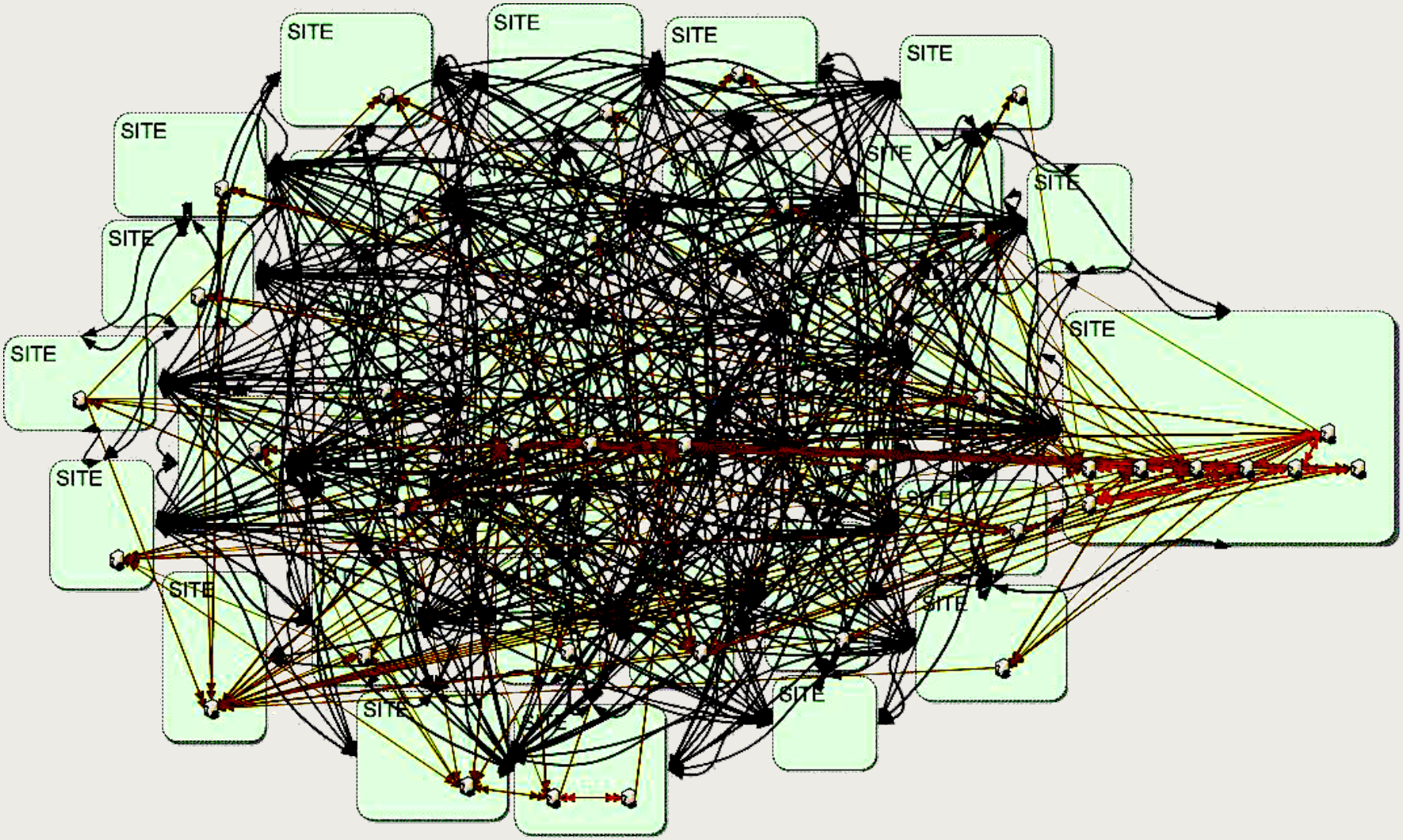
Топология репликции

ОБМЕН ТРАФИКОМ И ДОСТУПНОСТЬ

Fully connected (or mesh network) replication



Careful replication :)



Асинхронная репликация

- Клиенту возвращается ответ об успешной операции после успешной отправки в канал
- Следствие - нет гарантии консистентности со всеми нодами, очередь лога репликации может серьезно отставать (т.е. потенциально data-loss)

Кворум

- Клиенту возвращается ответ об успешной операции после записи на $V_w > V/2$ где V - общее количество нод (голосов), V_w - количество нод доступных для записи
- $V_r + V_w > V$ где V_r - количество голосов (нод) доступных для чтения

Подходы

- Локинг (Distributed Locking)
- Один активный участник
- Транзакции
- Tentative Транзакции
- Версионирование/тэггирование
- Выполнение в обратном порядке (reversible execution)

Conflict-Free Replicated Data Types

OR CRDT

CRDT

- Поддержка конкурентных операций на нескольких устройствах
- Офлайн-режим
- Long-running SAP (Facebook/Gmail/Soundcloud-типа)
- В целом проблема одно из решений по работе с асинхронными или конкрутентными сторонами, например плагины для редактора (Xi editor)

Решения доставки

- Event Stream (Pull)
- WebSTOMP/MQTT
- Aeron (Websocket), Akka etc.
- Pusher.com
- Google Cloud Pub/Sub
- Amazon SNS

Conflict-free replicated data types (CRDT) – типы данных без конфликтов

Свойства полурешентки ($\sqcup$ or LUB – Least Upper Bound, наименьшая верхняя грань):

- **коммутативность** – $\forall x, y : x\ y = y \sqcup x$
- **ассоциативность** – $x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$
- **идемпотентность** – $x \sqcup x = x$

Strong Eventual Consistency

$$C = [c_1, ..., c_n], \forall i, j : c_i = c_j \Rightarrow s_i \equiv s_j$$

ACID 2.0

- ACID 1.0: atomicity, consistency, isolation, durability
- ACID 2.0: associative, commutative, idempotent, distributed

Counter

$$1 + 1$$

+ not idempotent

$$1 + 1 \neq 1$$

$\cup$ – sets are good

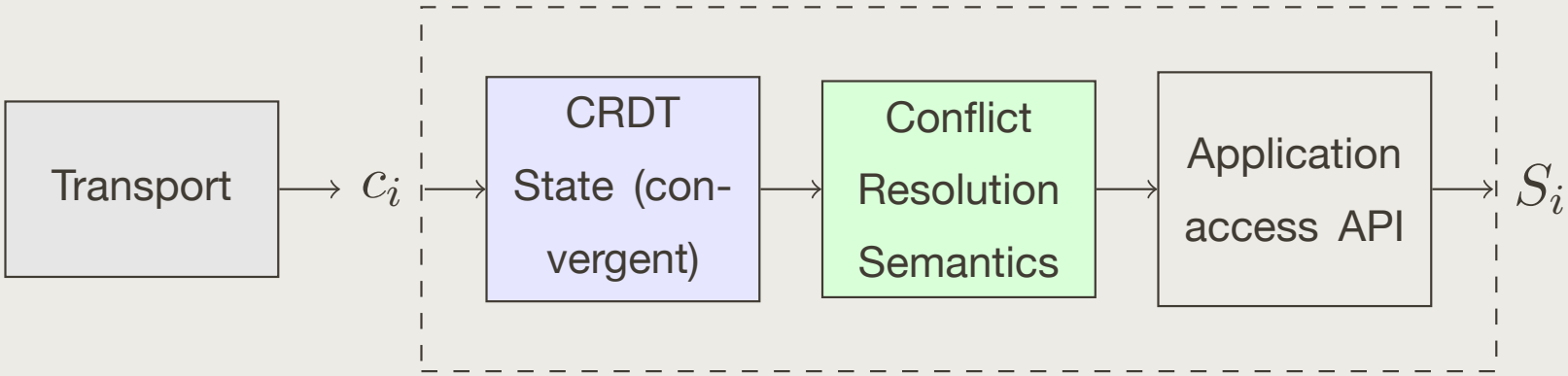
$$1 \cup 1 = 1$$

Прочие свойства

$$(1 \cup 2) \cup 3 = 1 \cup (2 \cup 3)$$

$$1 \cup 2 = 2 \cup 1$$

CRDT


Transport
c_i
CRDT State (convergent)
Conflict Resolution Semantics
Application access API
S_i

CRDT: пример свойств

```
messages = [
  { tag: 1, site: '1', payload: { key: 'val' } },
  { tag: 1, site: '1', payload: { key: 'val' } },
  { tag: 1, site: '2', payload: { key1: 'val1' } },
  { tag: 2, site: '1', payload: { key2: 'val2' } },
  { tag: 0, site: '2', payload: { key0: 'val0' } } ]
// idempotency,
messages.reduce(
  (v, c) =>
    v.filter(
      m => m.tag == c.tag && m.site == c.site
    ).length == 0 ? v.concat([c]) : v, []);

// partial order
messages.sort(/* ...tag, site... */)
```

CRDT: GCounter

```
export class GCounter {
  constructor(counters) { this.counters = counters; }
  increment(id) {
    return new GCounter(Object.assign({
      [id]: (this.counters[id] || 0) + 1}))}
  query() {
    return Object.values(this.counters)
                 .reduce((a, b) => a + b, 0); }
  merge(counter) {
    let sites = Object.keys(counter.counters)
                      .concat(Object.keys(this.counters));
    return new GCounter(sites.reduce(
      (merged, site) => Object.assign(
        merged, {[site]: Math.max(
          merged[site] || 0,
          counter.counters[site] || 0)}),
        this.counters));
  }
}
```

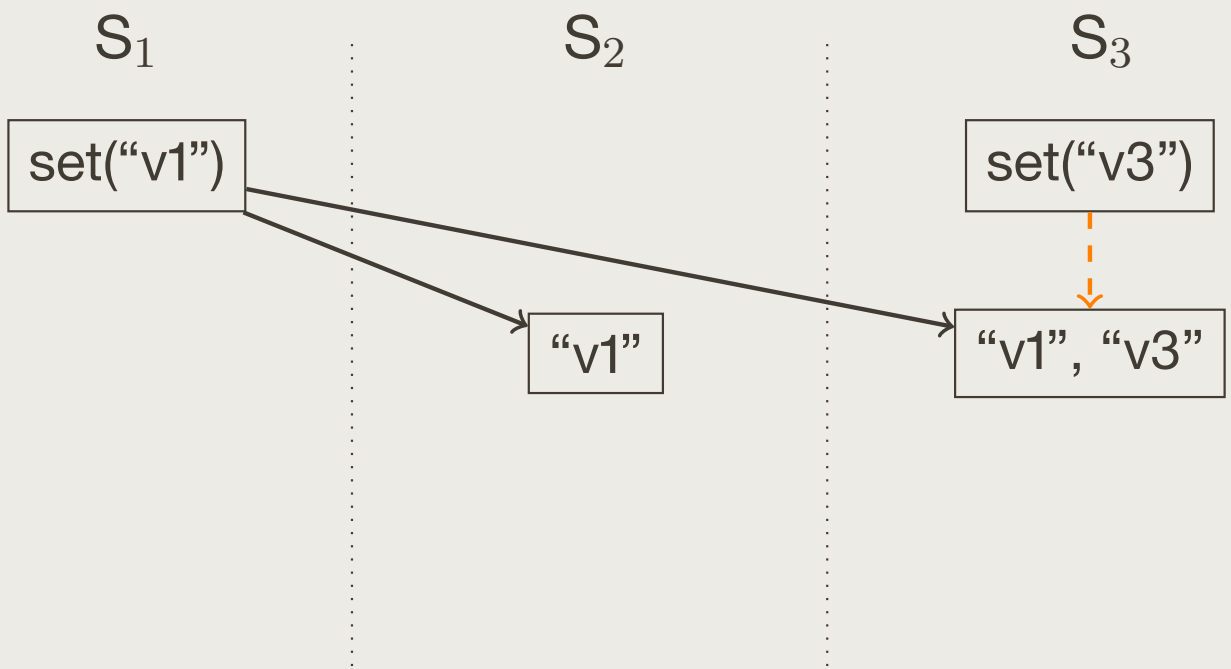
CRDT: GCounter payload

```
let counters = {
 1: 0
 2: 0
 ...
 N: 0}
```

CRDT: PNCounter

```
class PNCounter {
  constructor(p, n) { this.n = n; this.p = p}
  increment() { return new PNCounter(this.p.increment(), this.n); }
  decrement() { return new PNCounter(this.p, this.n.increment()); }
  query() { return this.p.query() - this.n.query(); }
  merge(pncounter) {
    return new PNCounter(
      this.p.merge(pncounter.p),
      this.n.merge(pncounter.n)); }
}
```

CRDT: MVRegister Figure

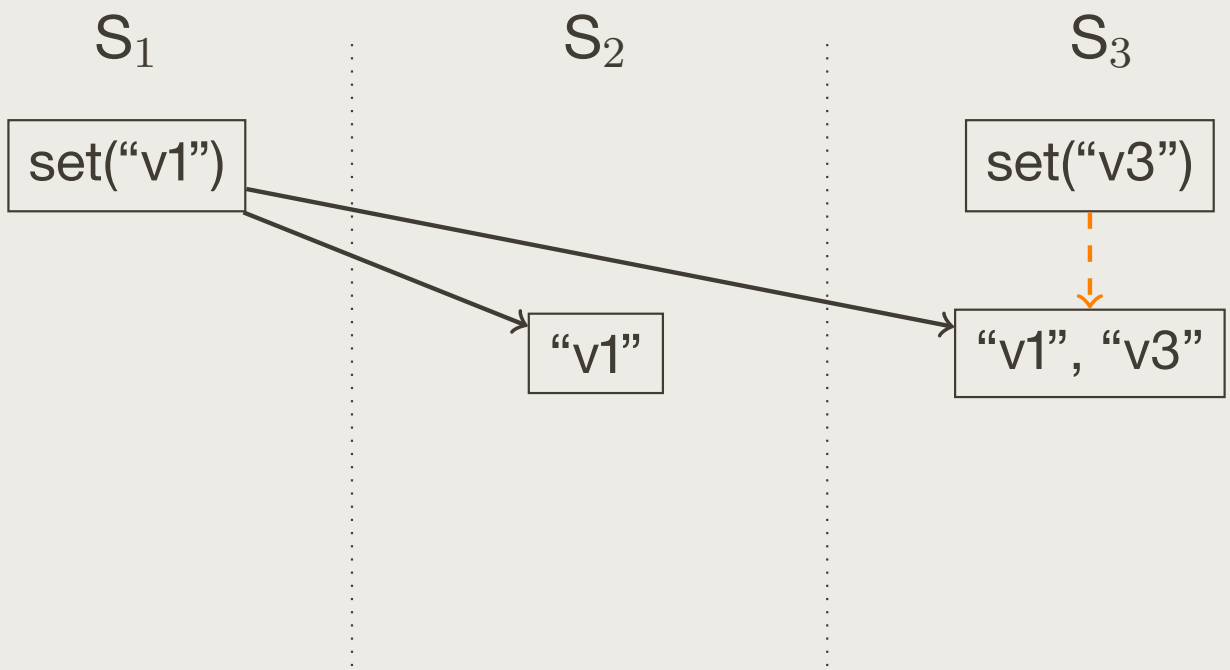


MVRegister Concurrent Operation

CRDT: MVRegister

```
export class MVRegister {
  constructor(id, register) { this.id = id; this.register = register;
  set(value) {
    return new MVRegister(
      this.id,
      Object.assign(this.register, {[this.id]: value}))}
  query() {return Object.values(this.register); }
  merge(register) {
    let state = this.register[this.id] === undefined ?
            {} : {[this.id]: this.register[this.id]};
      return new MVRegister(this.id,
              Object.assign(register.register, state))
  }
}
```

CRDT: MVRegister Figure



MVRegister Concurrent Operation

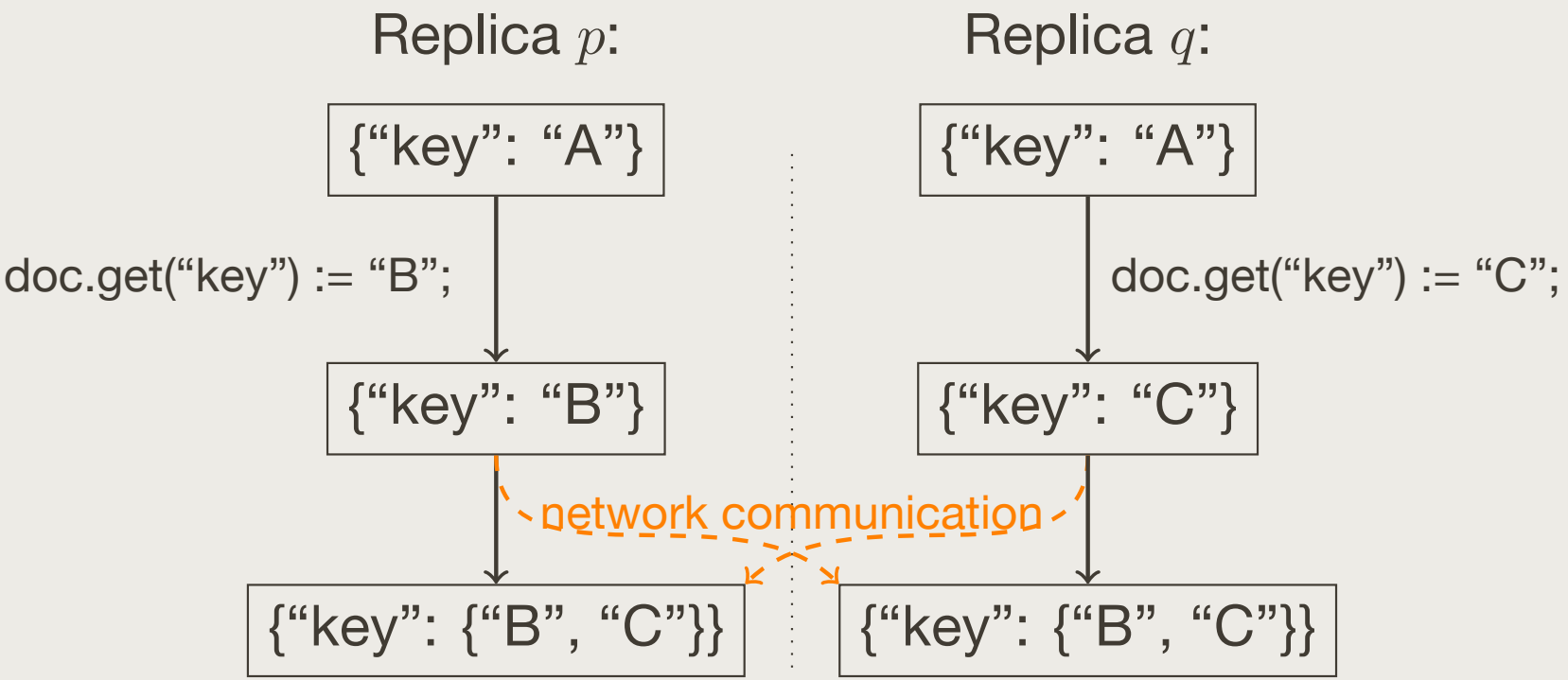
CRDT: MVRegister

```
const repr = (arr) => arr.length === 0 ? '[]' :
                                         '["' + arr.join('", "') + '"]';

function example_mvregister() {
  let log = (op, r1, r2) => console.log("[OP] " + op +
                                        ": r1=" + repr(r1.query()) +
                                        ", r2=" + repr(r2.query()));

  let r1 = new MVRegister(1, {});
  let r2 = new MVRegister(2, {});
  log("register1", r1, r2);
  r1 = r1.set("key1")
  log("r1.set(key1)", r1, r2);
  r2 = r2.merge(r1);
  log("r1 U r2", r1, r2);
  r1.set("v1");
  log("r1.set(v1)", r1, r2)
  r2.set("v2")
  log("[CONCURRENT] r2.set(v2)", r1, r2)
  r2 = r2.merge(r1);
  log("[MERGED]", r1, r2);
}
```

CRDT: MVRegister Output

```
[OP] register1: r1=[], r2=[]
[OP] r1.set(key1): r1=["key1"], r2=[]
[OP] r1 U r2: r1=["key1"], r2=["key1"]
[OP] r1.set(v1): r1=["v1"], r2=["key1"]
[OP] [CONCURRENT] r2.set(v2): r1=["v1"], r2=["key1", "v2"]
[OP] [MERGED]: r1=["v1", "v2"], r2=["v1", "v2"]
```

JSON CRDT



Конкуретные присвоения в регистре по ключу doc.get(“key”) репликами p и q.

CRDT: Типы

- Register: LWW или Multi-Value (как Dynamo или Couchdb)
- Counter только растущий
- G-Set – множество с возможностью исключительно добавления
- 2P-Set – множество, где один уникальный элемент можно удалять только один раз (G-Set + Tombstones set)
- LWW-Element-Set – LWW на базе vector clocks
- OR-Set – тэгированные элементы, тэги помещаются в множество Tombstones
- WOOT, LOGOOT, Treedoc, RGA, LSEQ для упорядоченных списков

Инструменты

- Roshi by Soundcloud
- Riak 2.0: Counters, Flags, Sets, Registers, Maps
- Redis Labs CRDT
- Y-js – framework for offline-first p2p shared editing on structured data
- Swarm (and forever-in-pre-alpha tool)
- replikativ.io – p2p distributed system framework
- GUN framework: p2p distributed framework

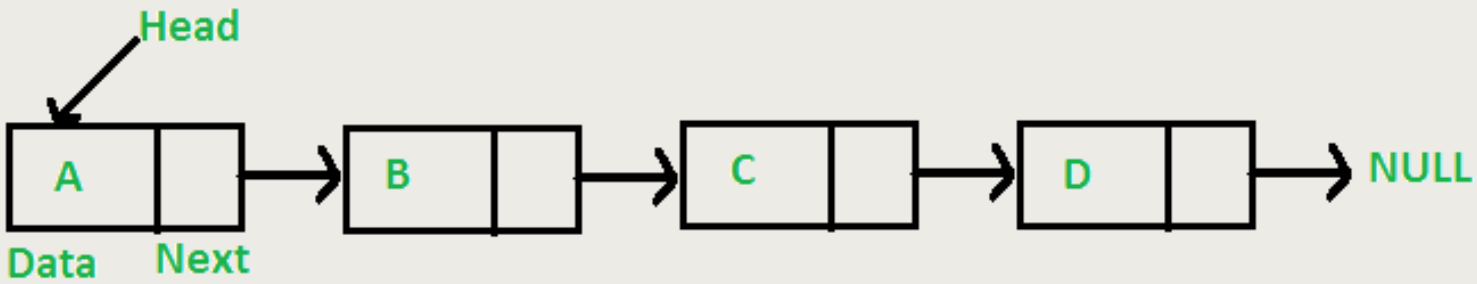
Кто использует CRDT?

- Facebook
- TomTom
- League of Legends
- SoundCloud
- Bet265
- RIAK Distributed Database

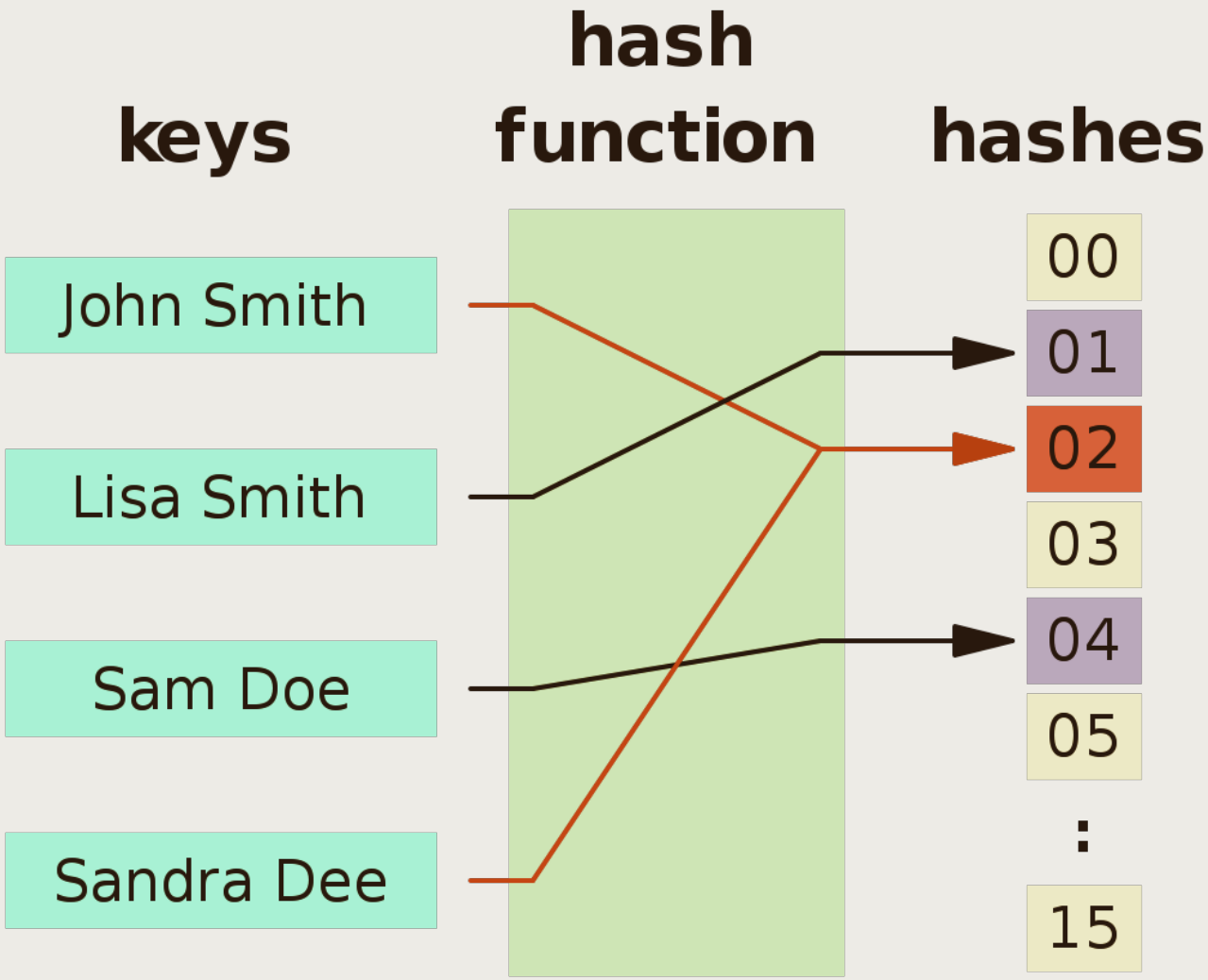
Blockchain

AND MERKLE TREES

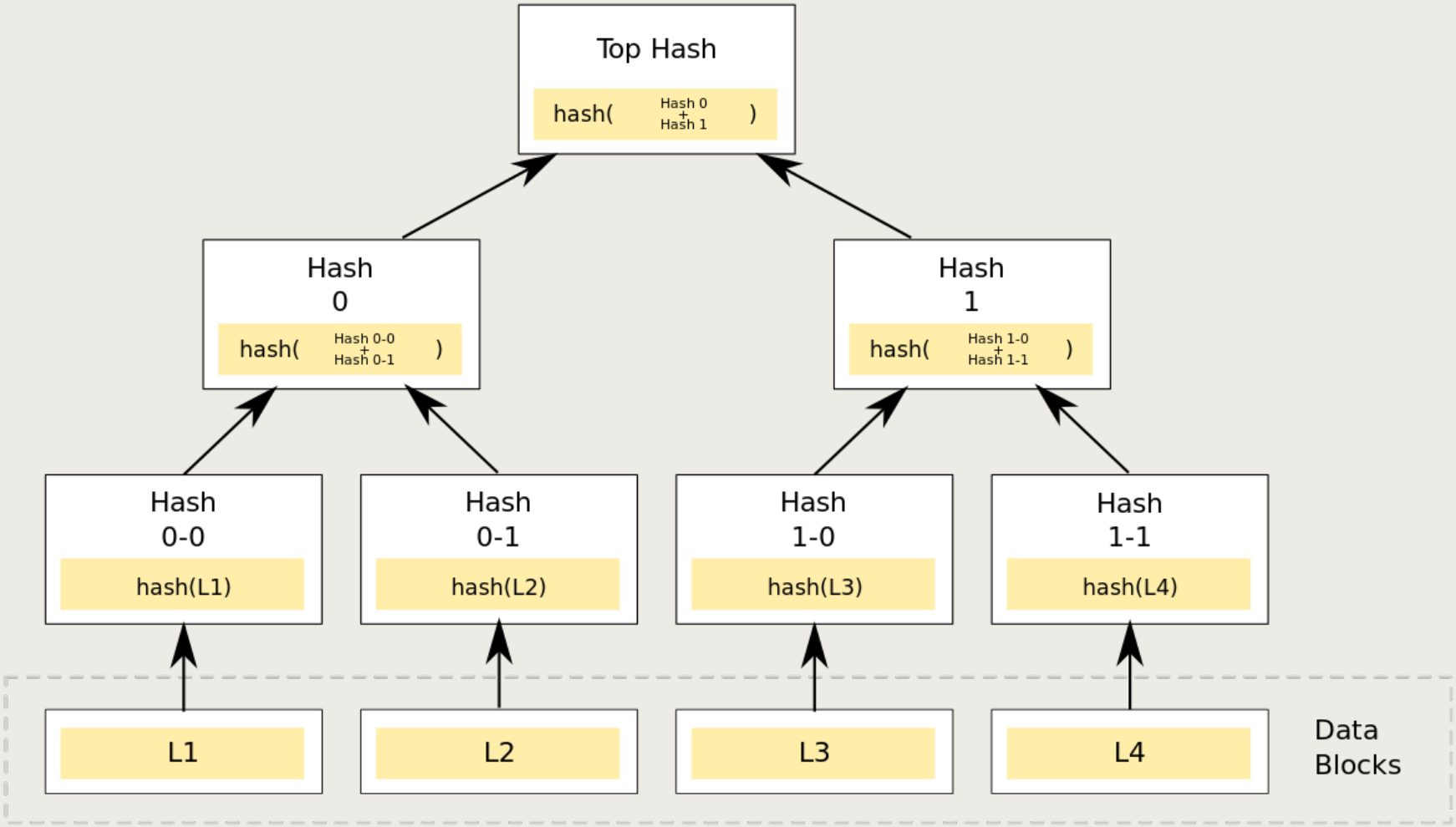
LinkedList



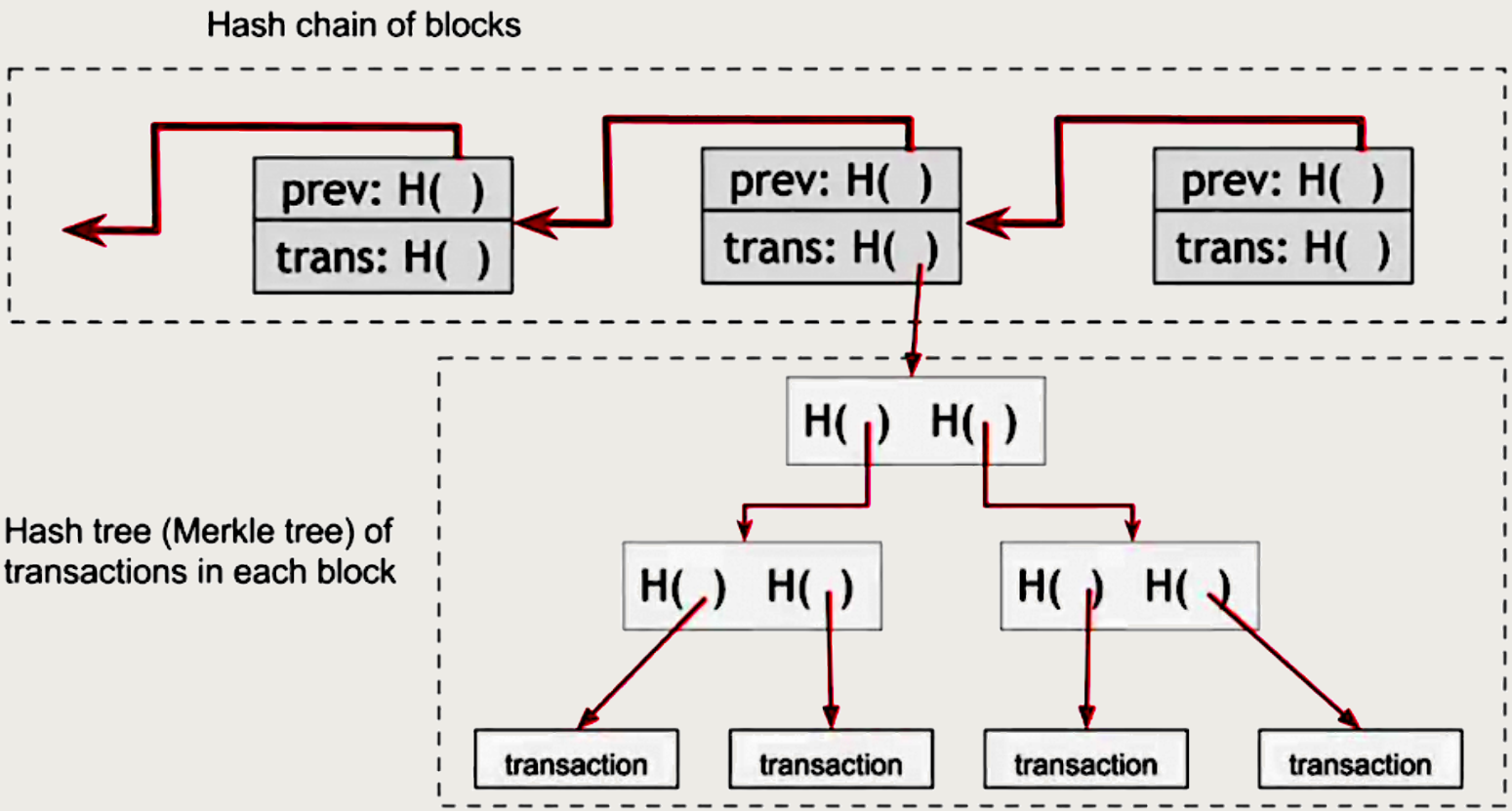
Hash function



MerkleTree


Top Hash
hash(Hash 0 + Hash 1)
Hash 0
hash(Hash 0-0 + Hash 0-1)
Hash 1
hash(Hash 1-0 + Hash 1-1)
Hash 0-0
hash(L1)
Hash 0-1
hash(L2)
Hash 1-0
hash(L3)
Hash 1-1
hash(L4)
L1
L2
L3
L4
Data Blocks

Blockchain


Hash chain of blocks
prev: H( )
trans: H( )
prev: H( )
trans: H( )
prev: H( )
trans: H( )
Hash tree (Merkle tree) of transactions in each block
H( ) H( )
H( ) H( )
H( ) H( )
transaction
transaction
transaction
transaction

Thanks

@maxmaxmaxmax