Effortless Eventual Consistency

Gossip, CRDTs, and Weave Mesh

Outline

- Theory
- Practice
- Extension

Outline

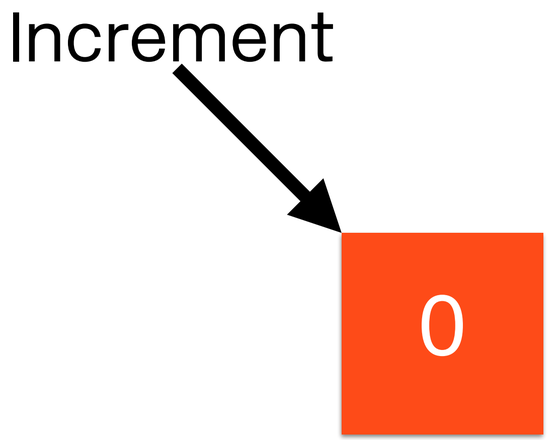
- Theory
  - Gossip
  - CRDT
- Practice
  - Weave Mesh
  - Weave Net
- Extension
  - User data types
  - Strong consistency

Motivating Example

Increment-only counter

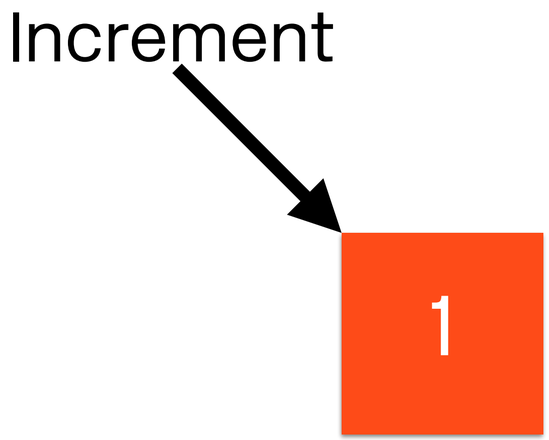
Motivating Example

Increment-only counter



Motivating Example

Increment-only counter

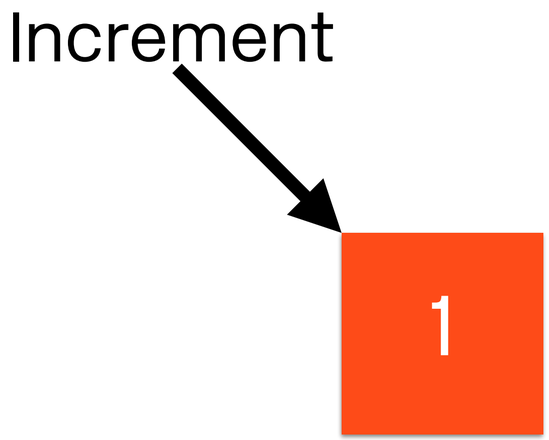


Motivating Example

Increment-only counter

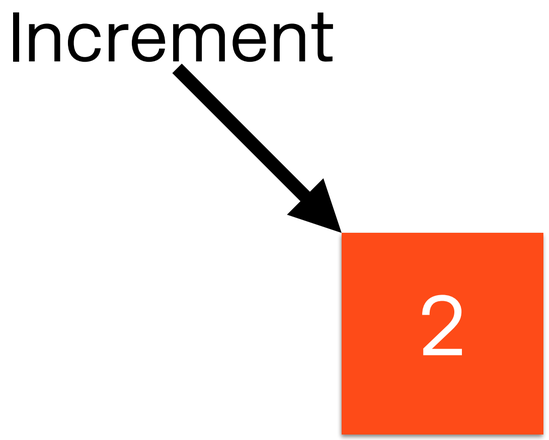
Motivating Example

Increment-only counter



Motivating Example

Increment-only counter

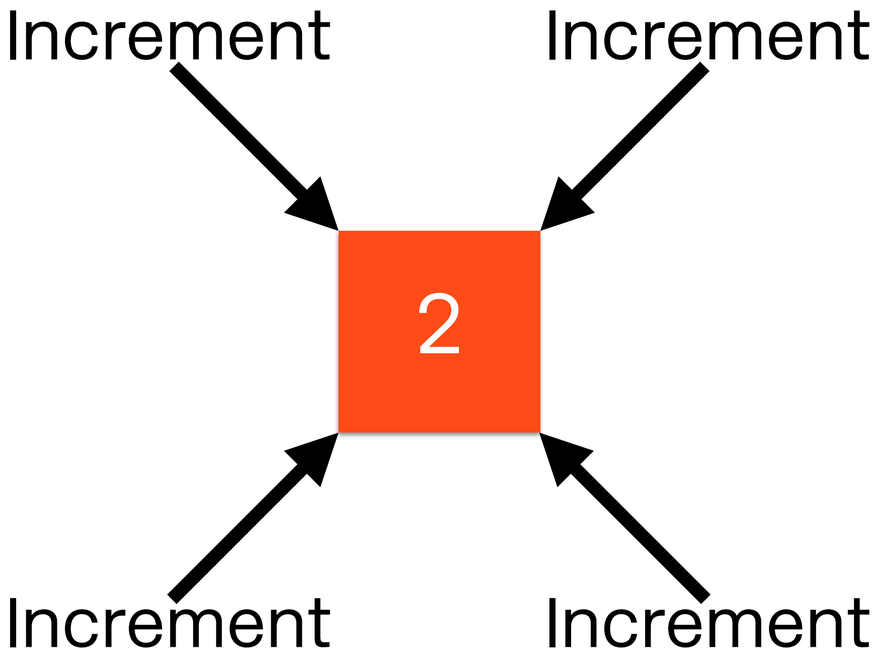


Motivating Example

Increment-only counter

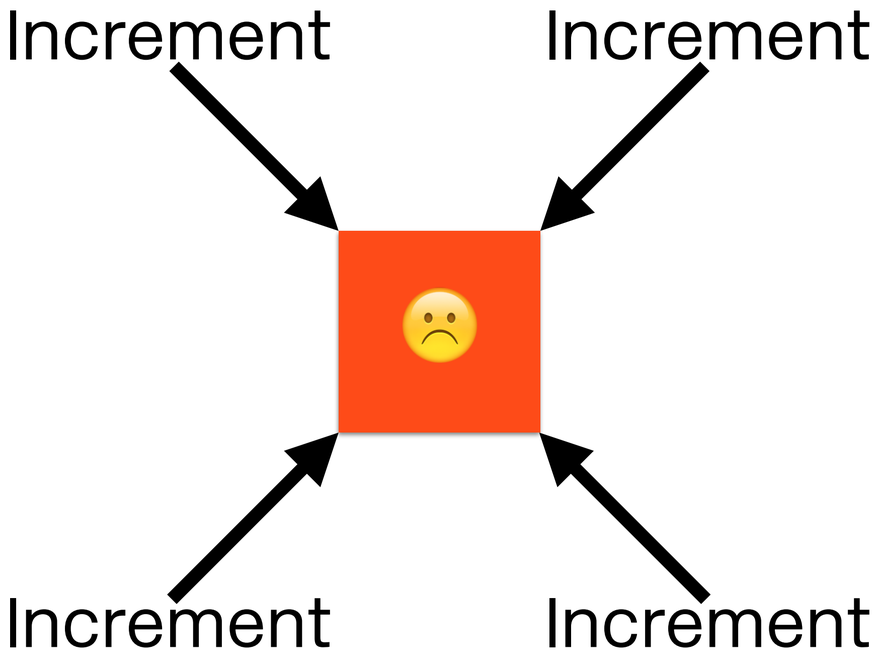
Motivating Example

Increment-only counter



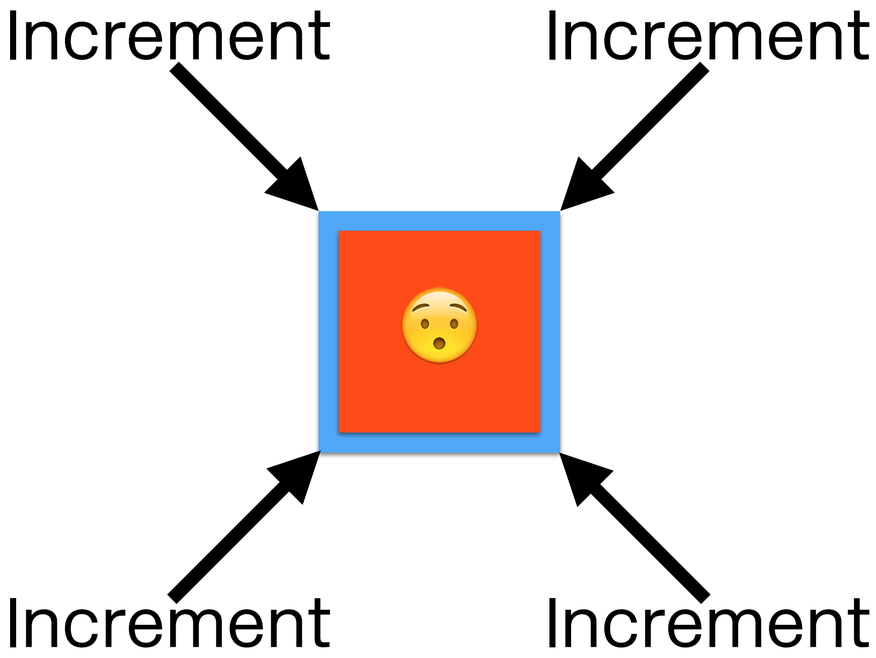
Motivating Example

Increment-only counter



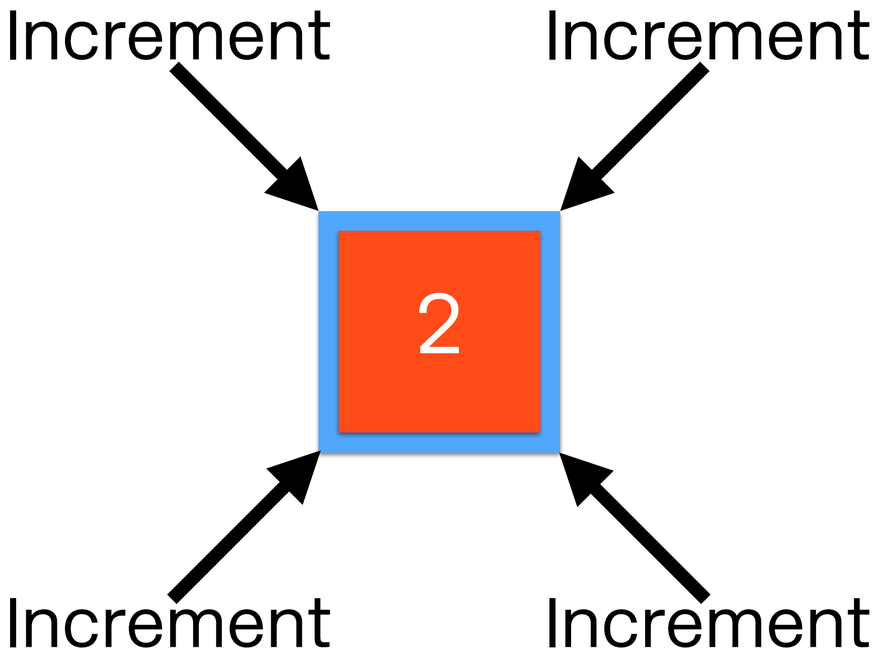
Motivating Example

Increment-only counter



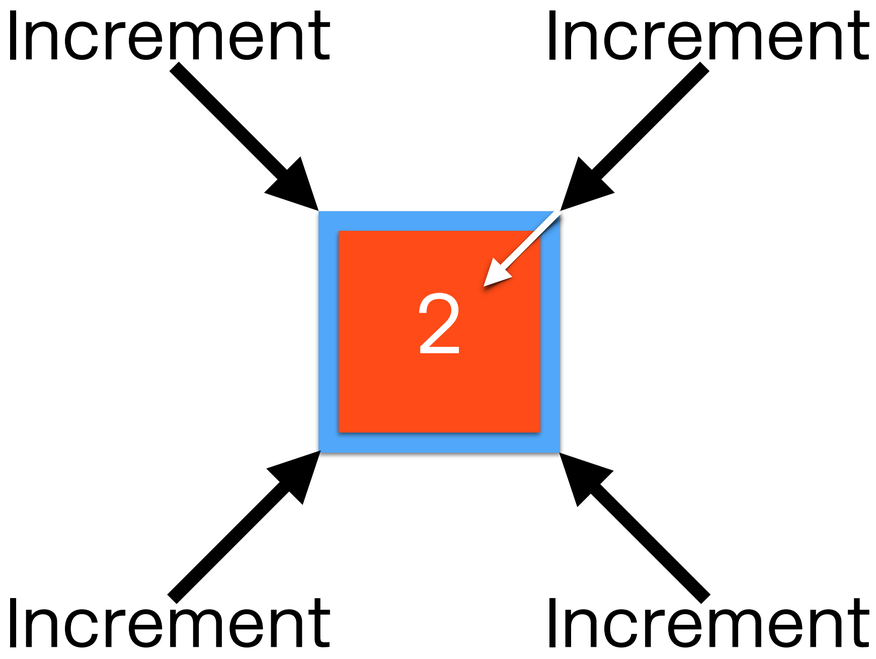
Motivating Example

Increment-only counter



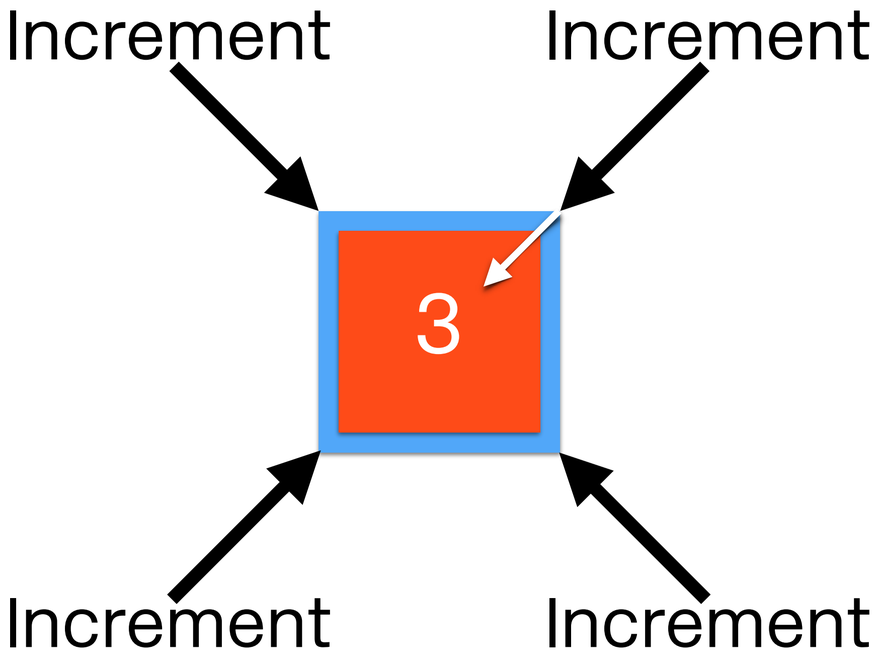
Motivating Example

Increment-only counter



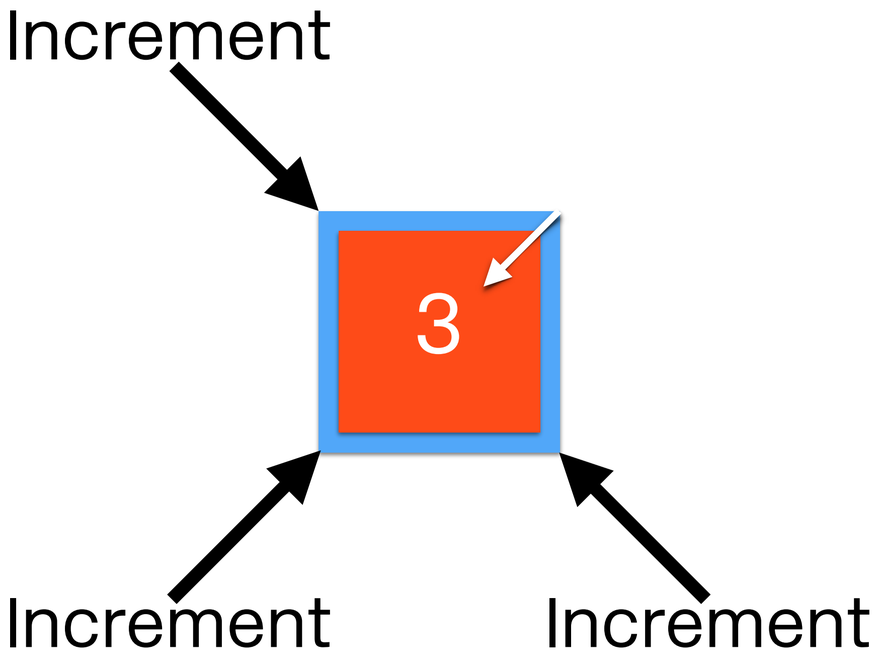
Motivating Example

Increment-only counter



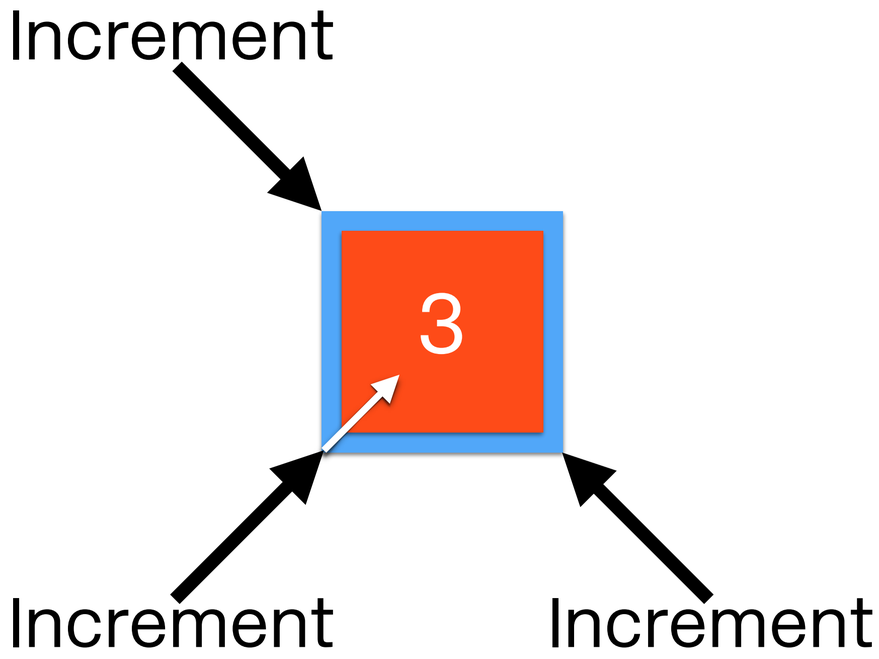
Motivating Example

Increment-only counter



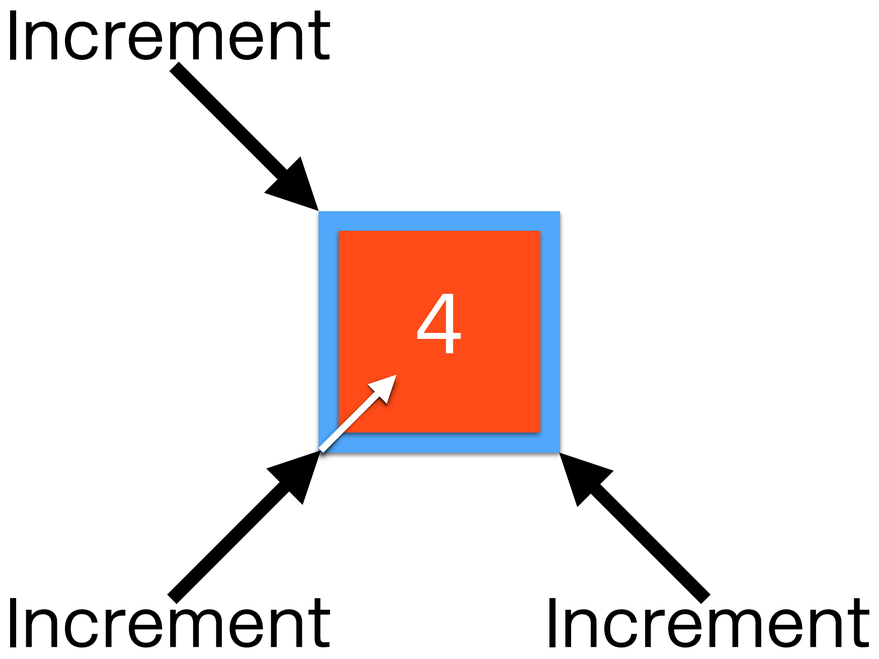
Motivating Example

Increment-only counter



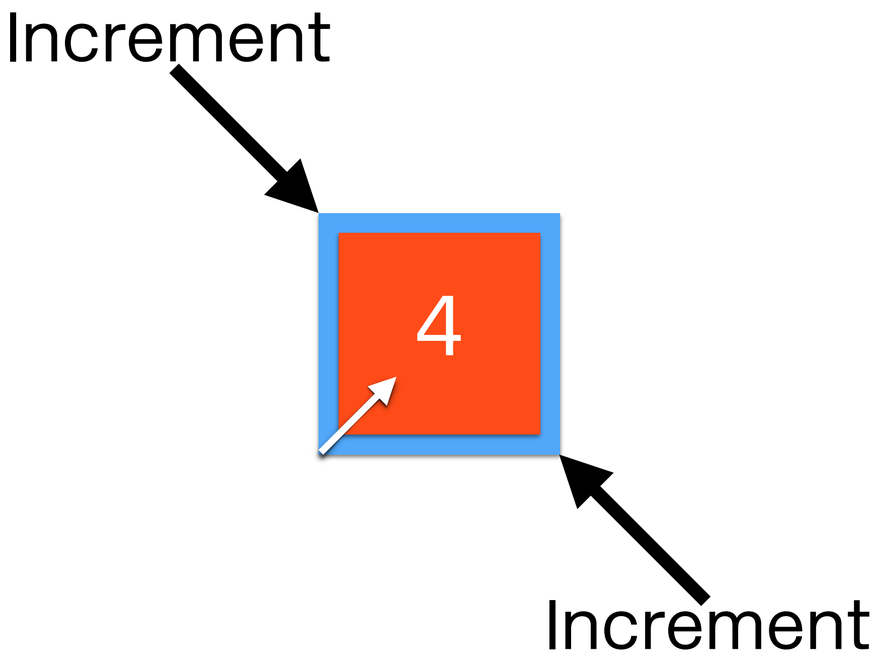
Motivating Example

Increment-only counter



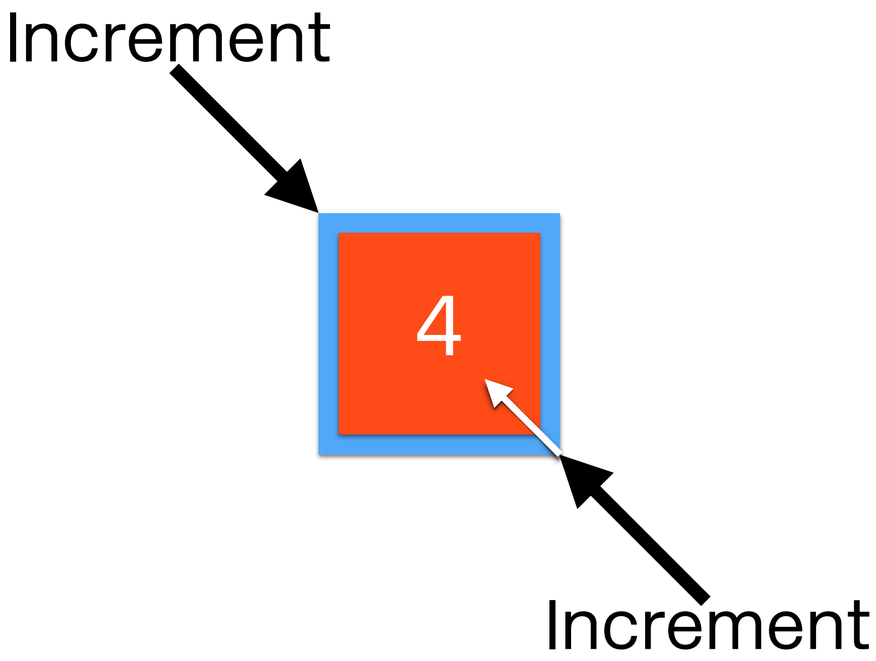
Motivating Example

Increment-only counter



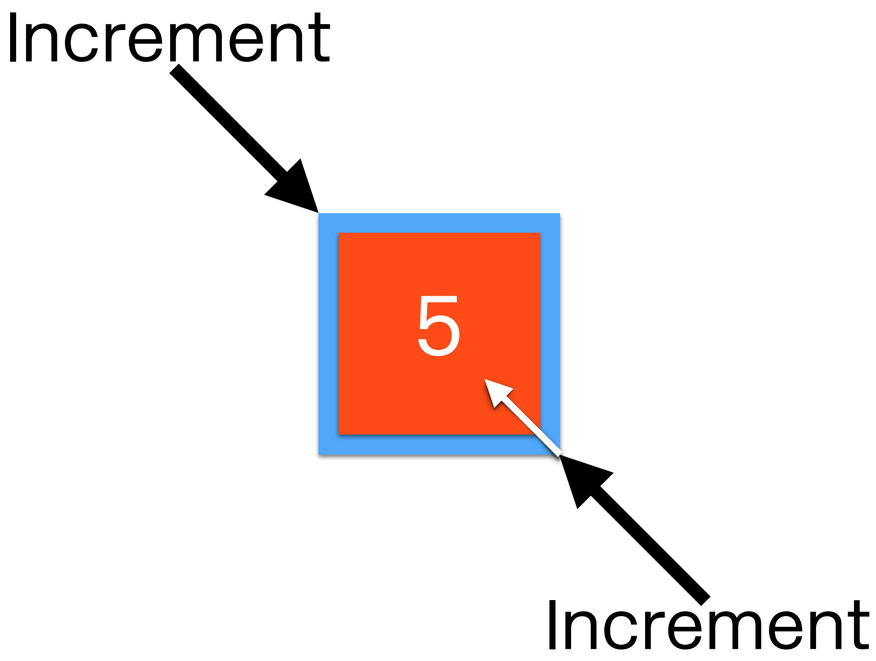
Motivating Example

Increment-only counter



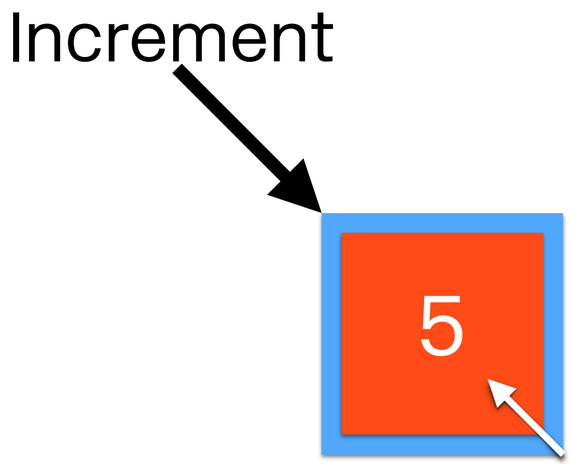
Motivating Example

Increment-only counter



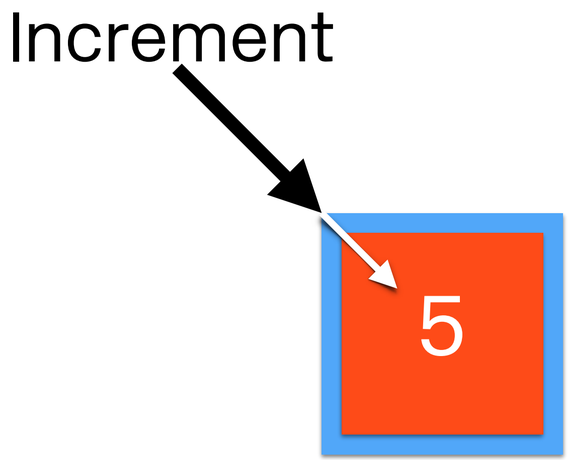
Motivating Example

Increment-only counter



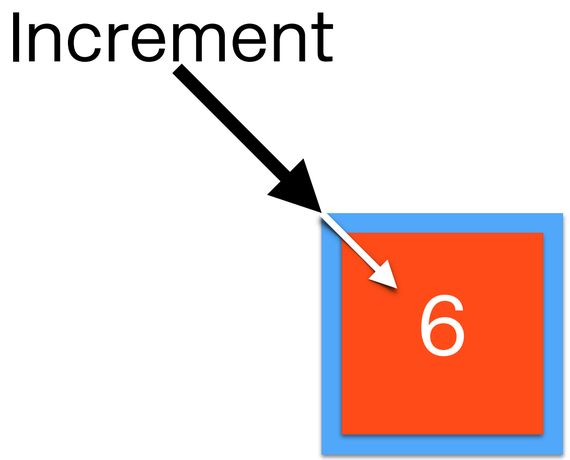
Motivating Example

Increment-only counter



Motivating Example

Increment-only counter

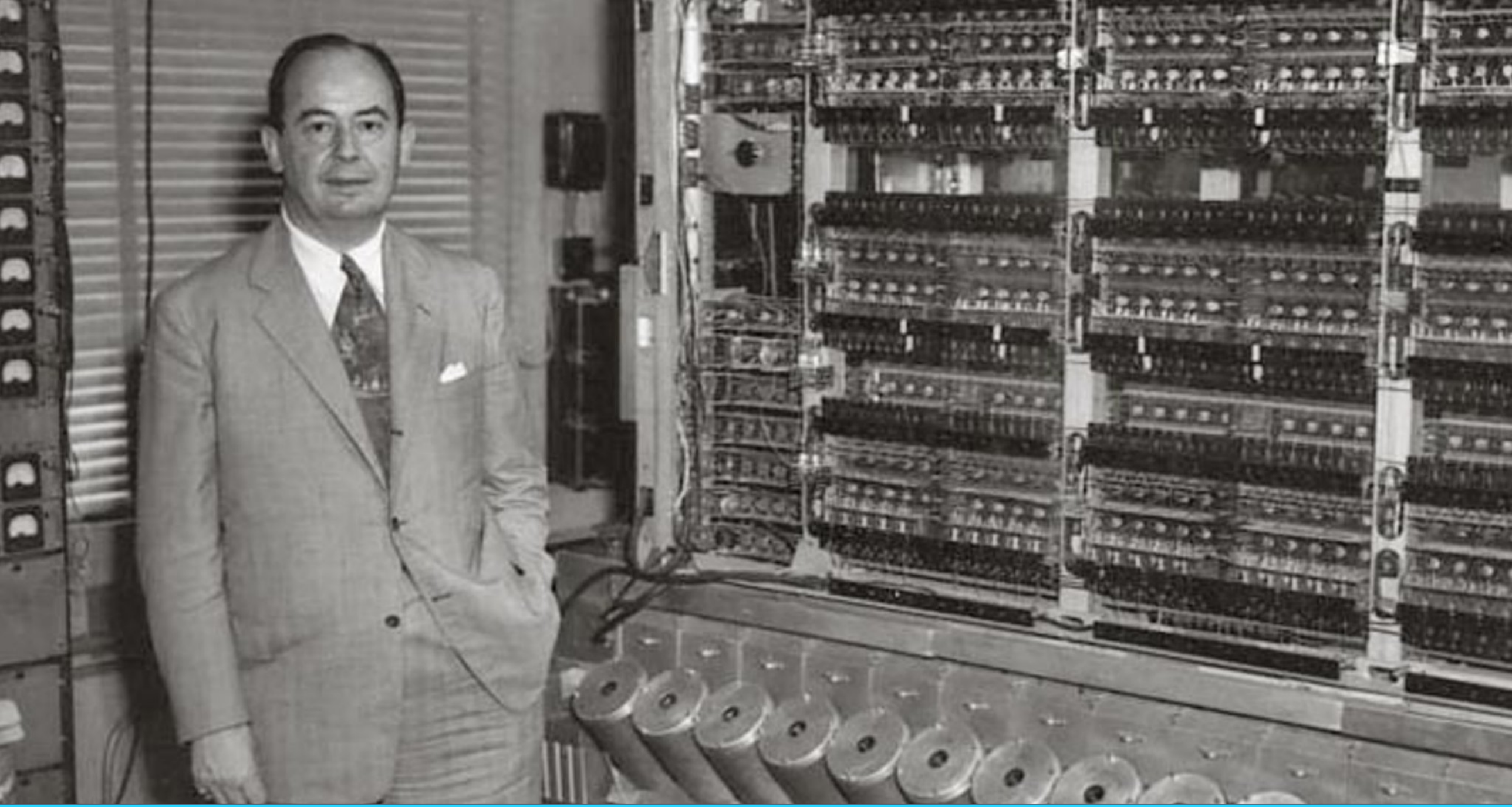


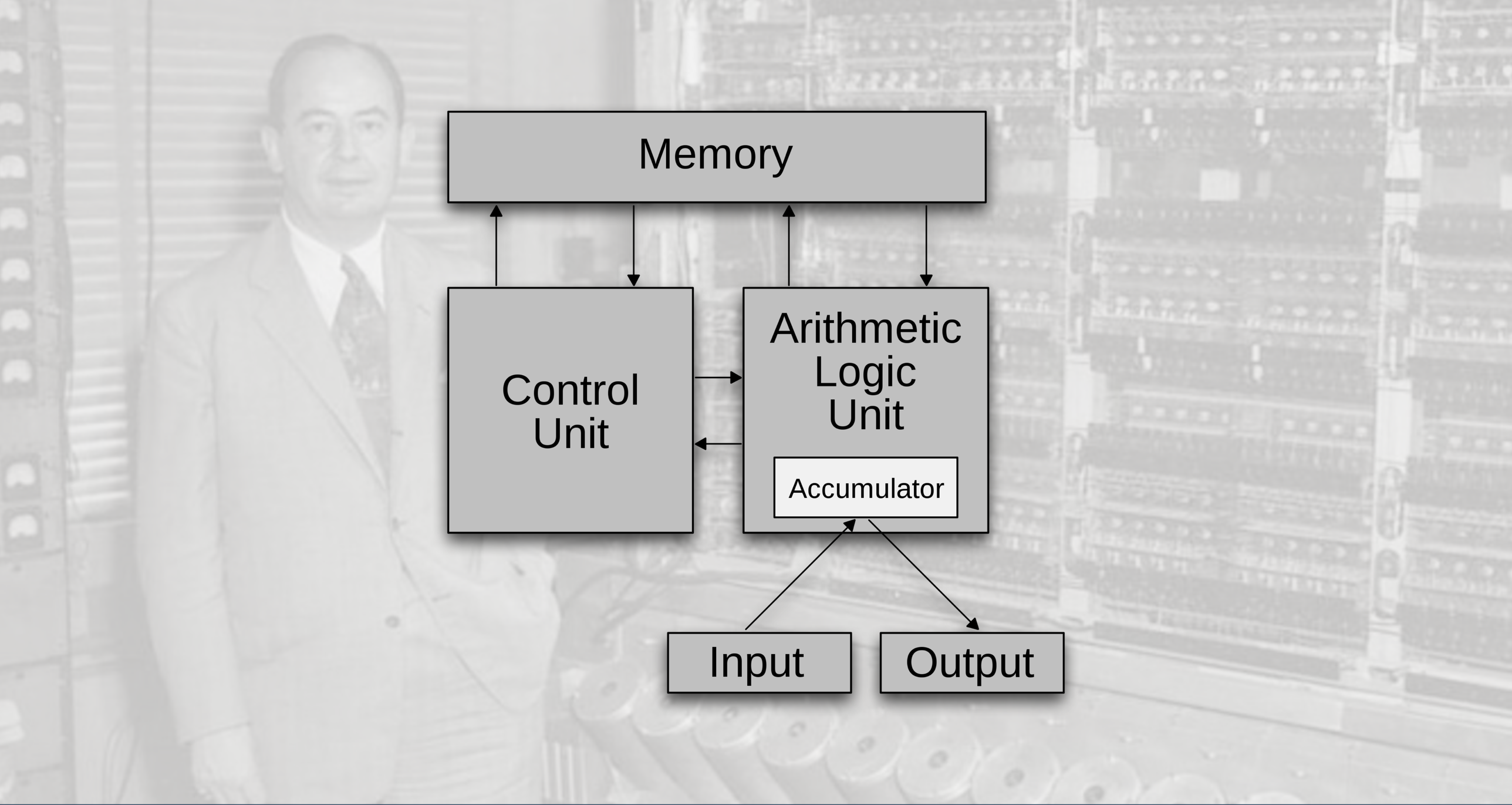
Motivating Example

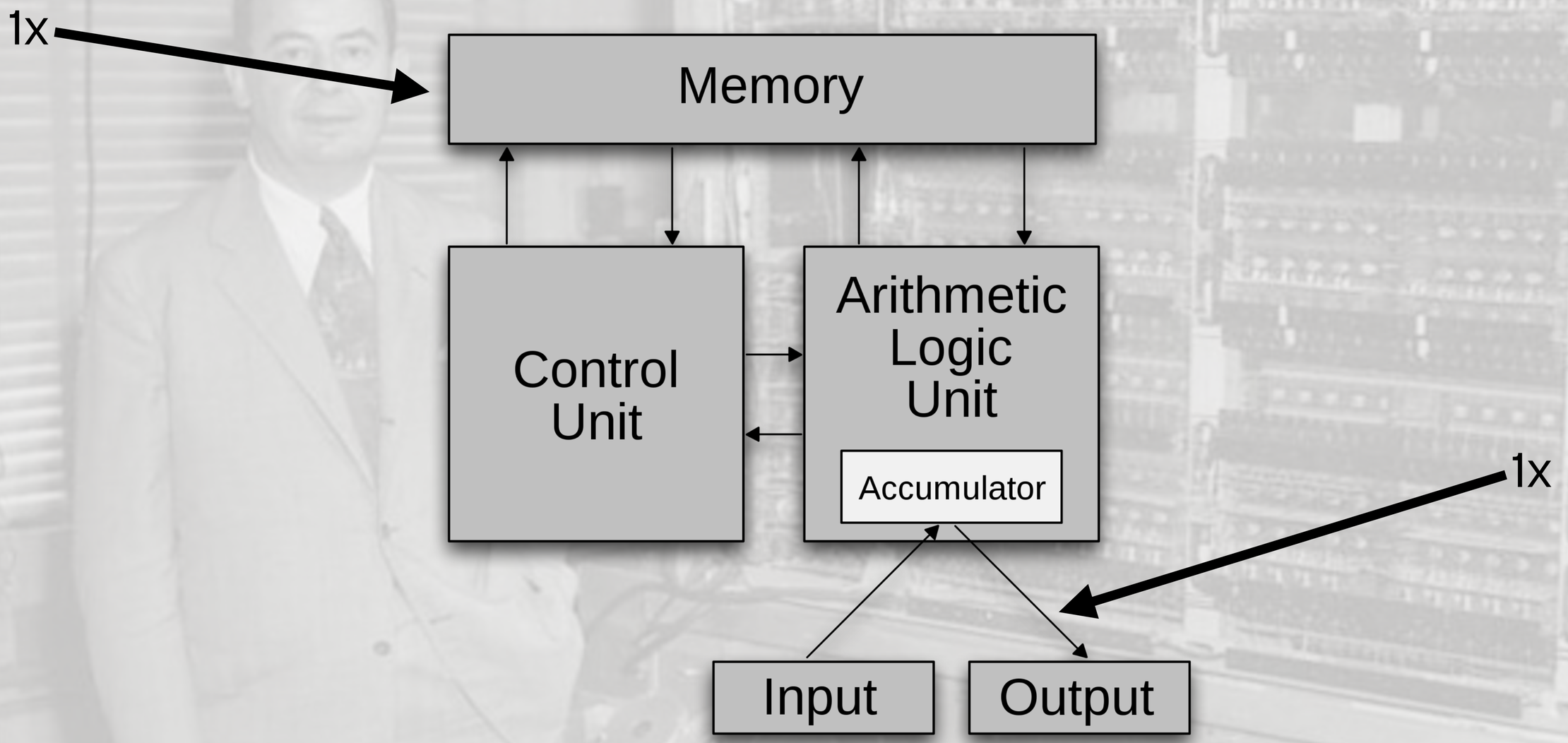
Increment-only counter

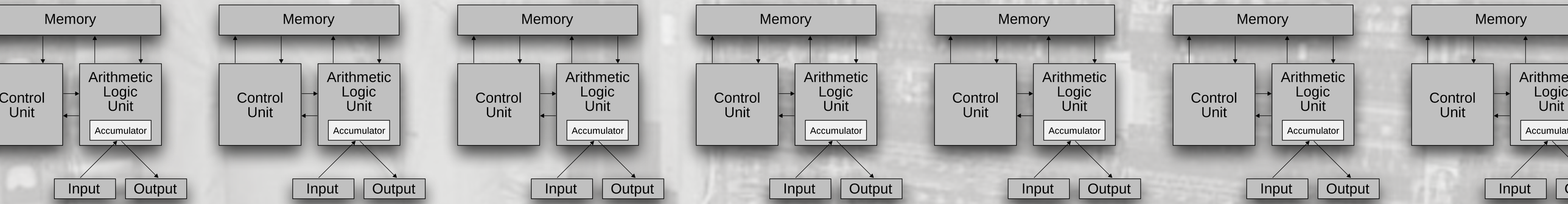
Motivating Example

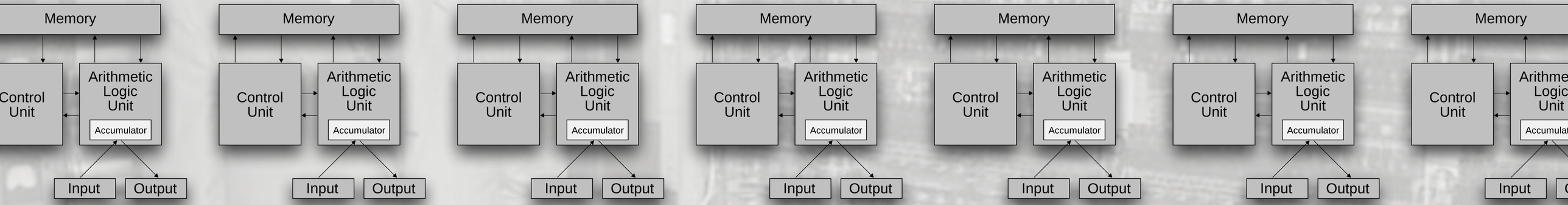
Increment-only counter



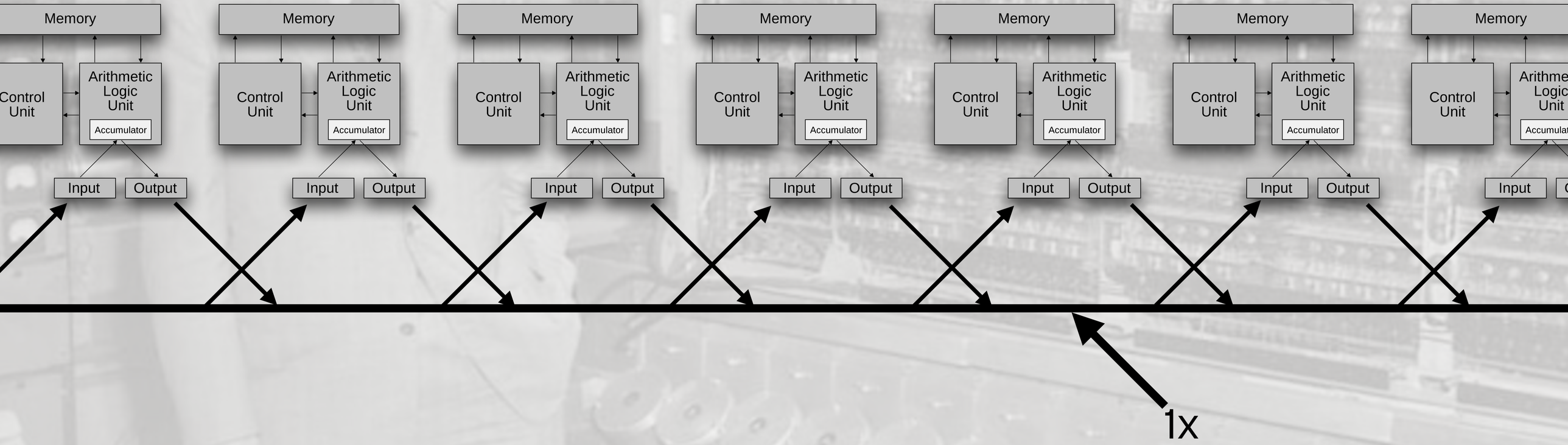

Memory
Control Unit
Arithmetic Logic Unit
Accumulator
Input
Output

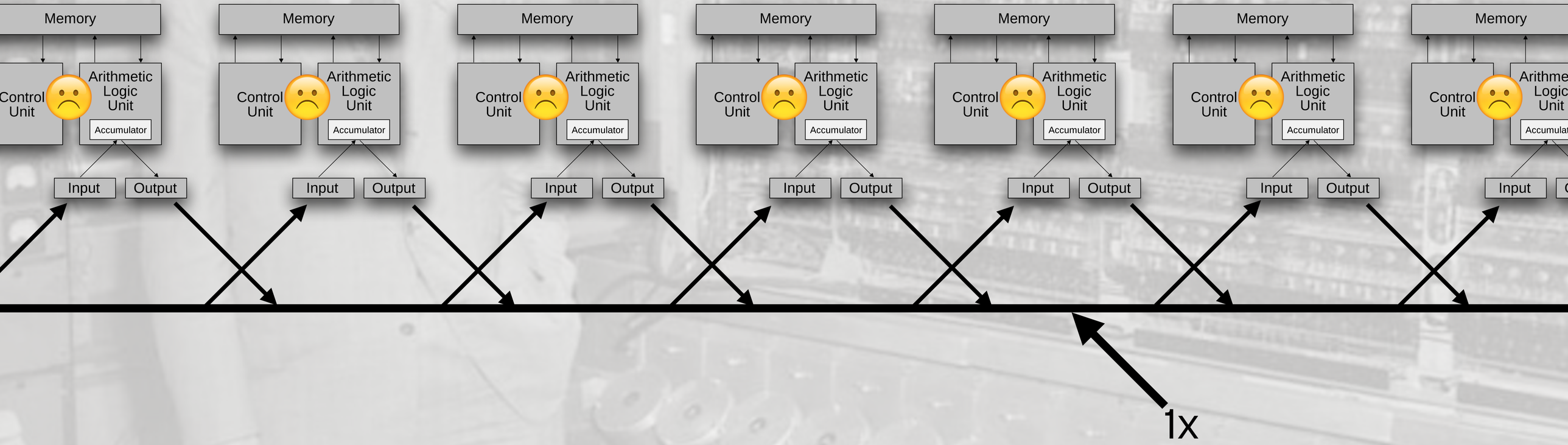

1x
Memory
Control Unit
Arithmetic Logic Unit
Accumulator
1x
Input
Output


Memory
Control Unit
Arithmetic Logic Unit
Accumulator
Input
Output


Memory
Control Unit
Arithmetic Logic Unit
Accumulator
Input
Output

1x


Memory
Control Unit
Arithmetic Logic Unit
Accumulator
Input
Output
1x


Memory
Control Unit
Arithmetic Logic Unit
Accumulator
Input
Output
1x

Theory:
Gossip

Gossip solves…

- Information dissemination in irregular networks
  - Unreliable links
  - Dynamic, changing topologies

Gossip requires…

- Periodic and pairwise communication
- Frequency of interaction ≪ individual message latency
- Redundancy in delivered information

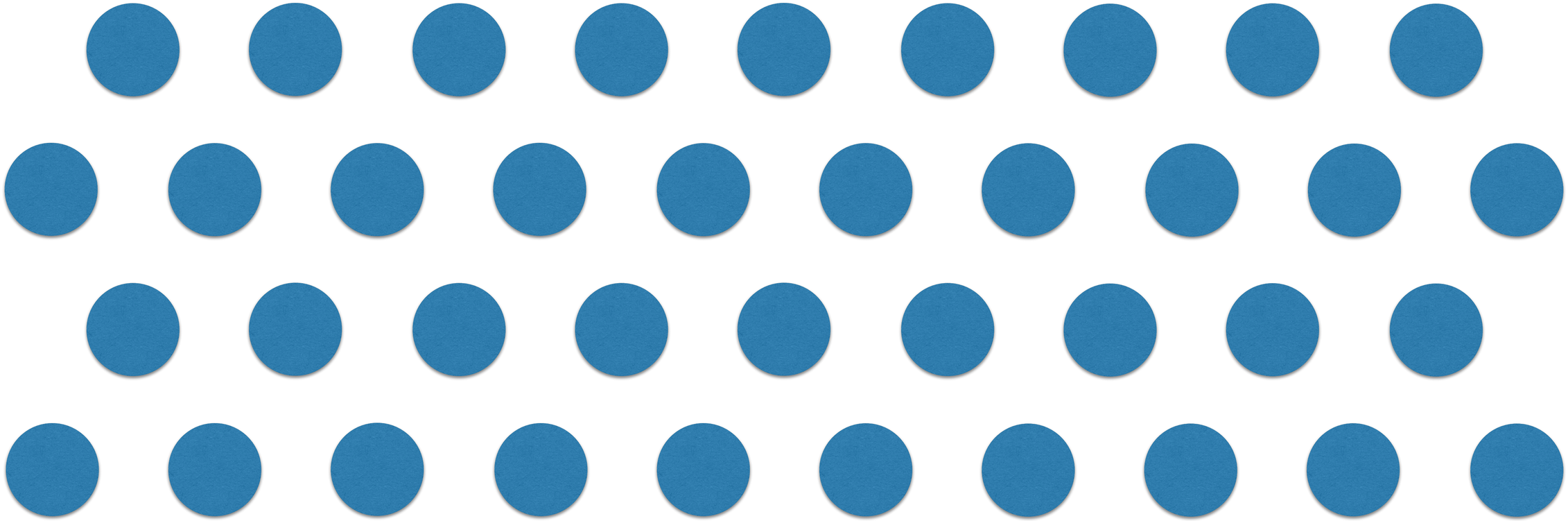
Gossip requires…

- Periodic and pairwise communication
- Frequency of interaction ≪ individual message latency
- Redundancy in delivered information

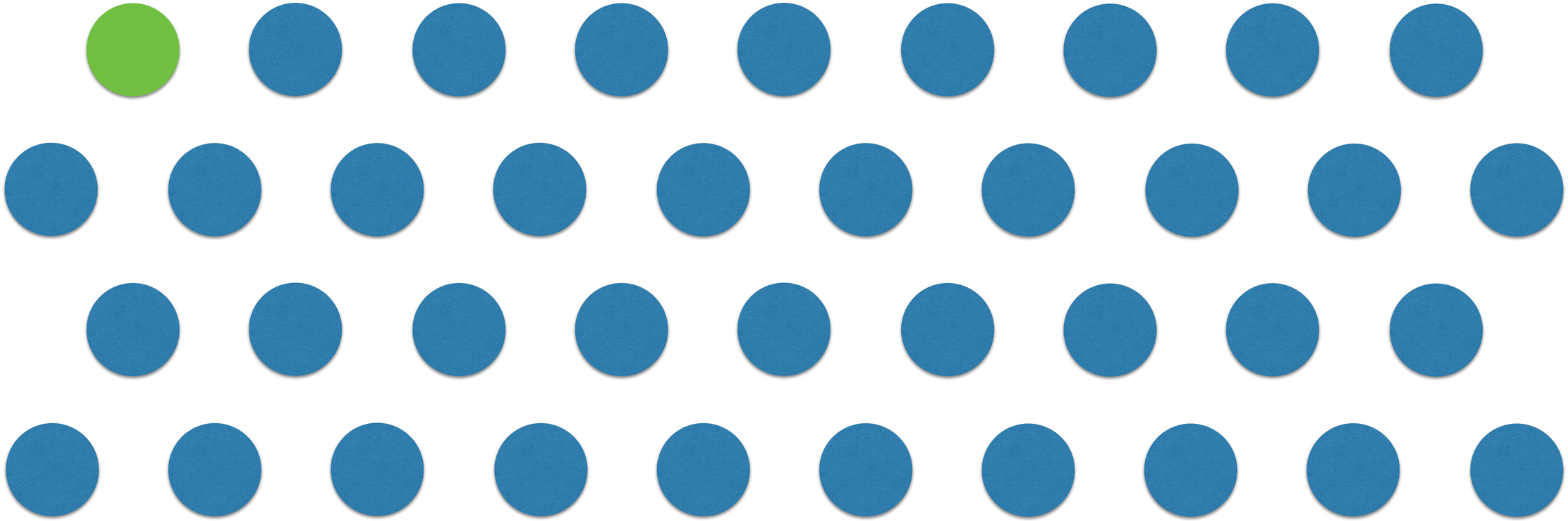
Gossip modes

- Push — if we know X, every round, tell a peer about X
- Pull — every round, ask a peer about all possible X
- Push/pull — push for some cycles, then pull for some cycles

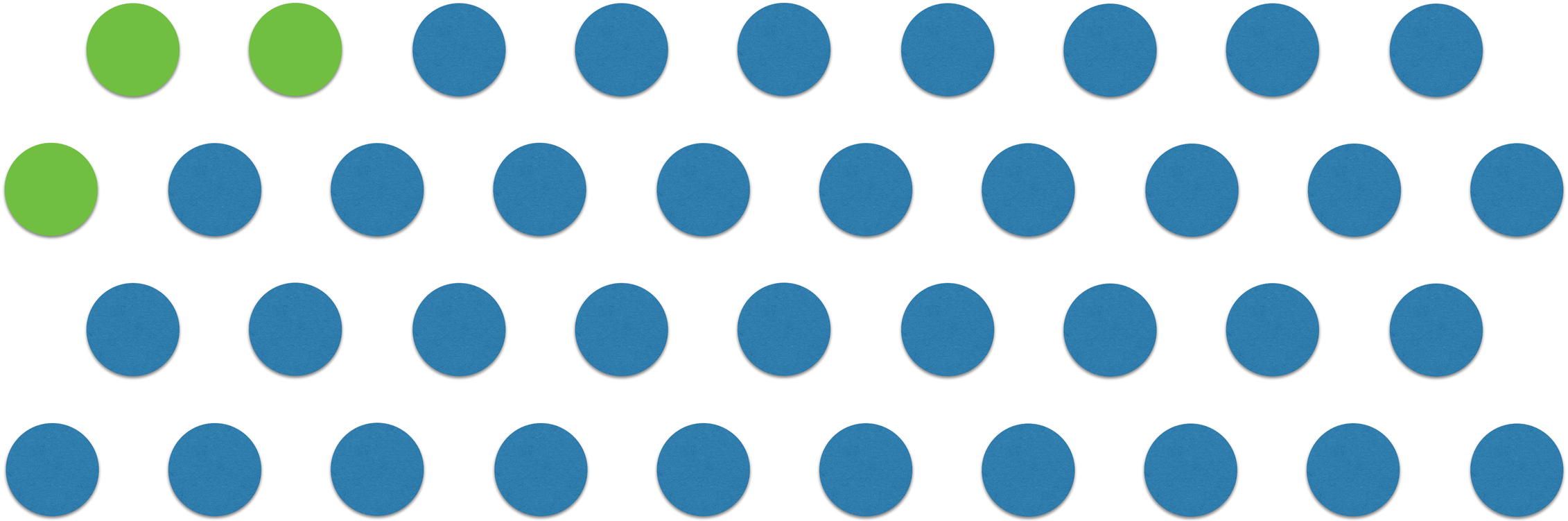
Rumor



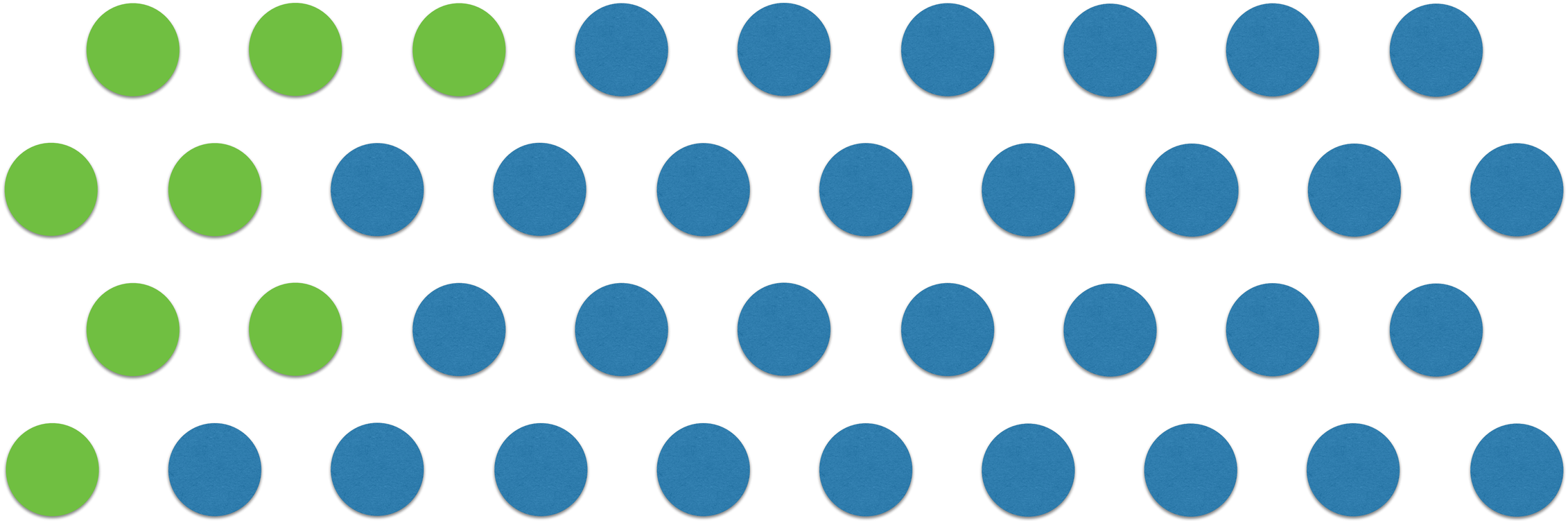
Rumor



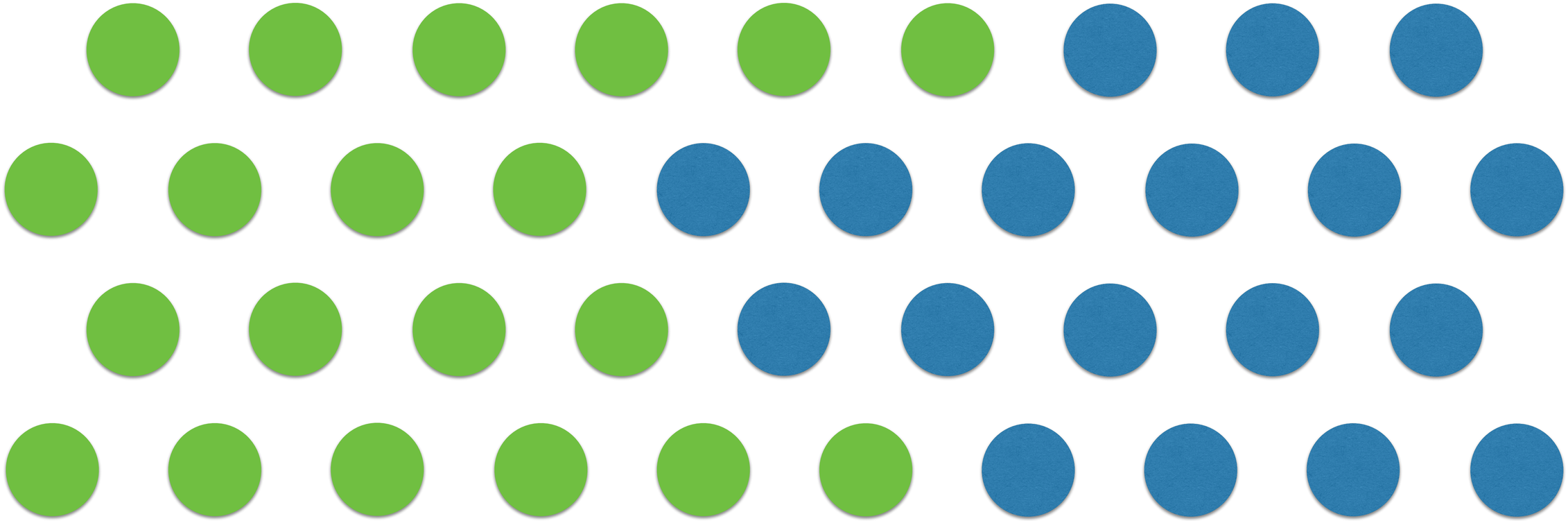
Rumor



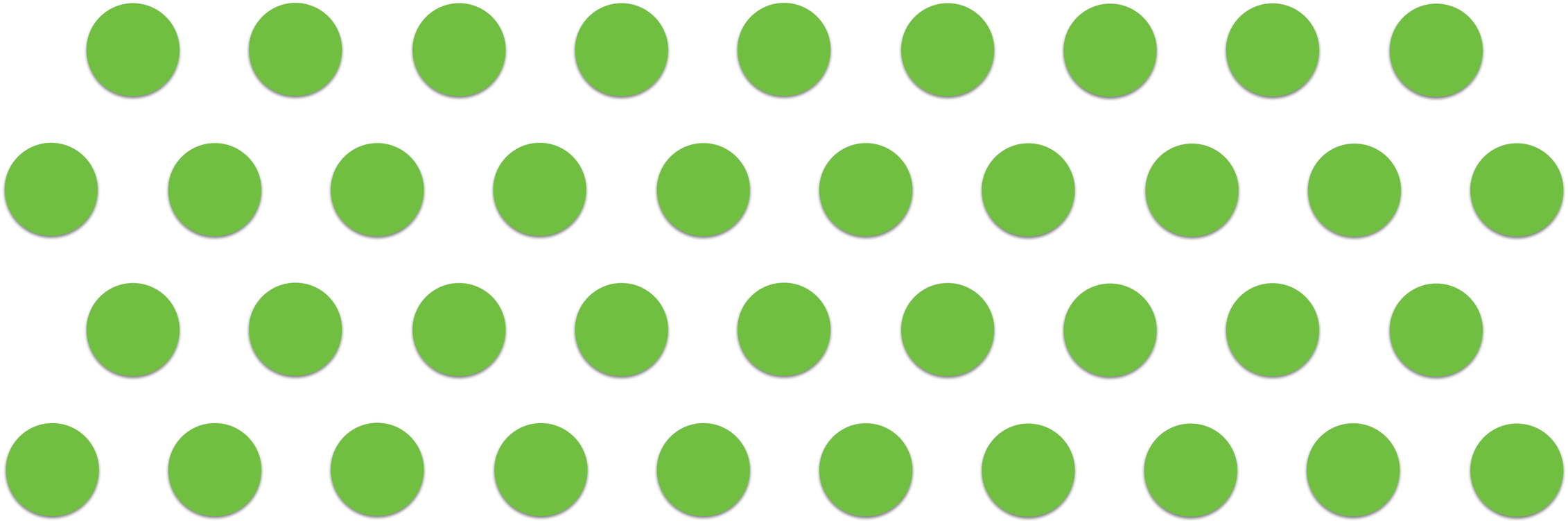
Rumor



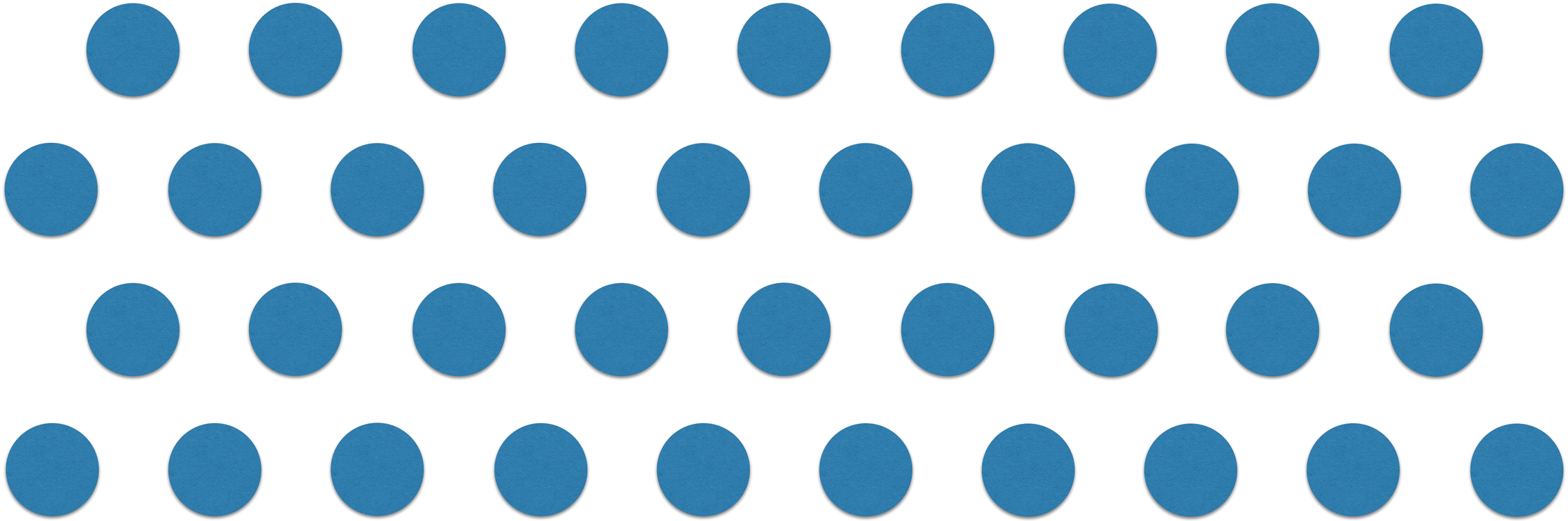
Rumor



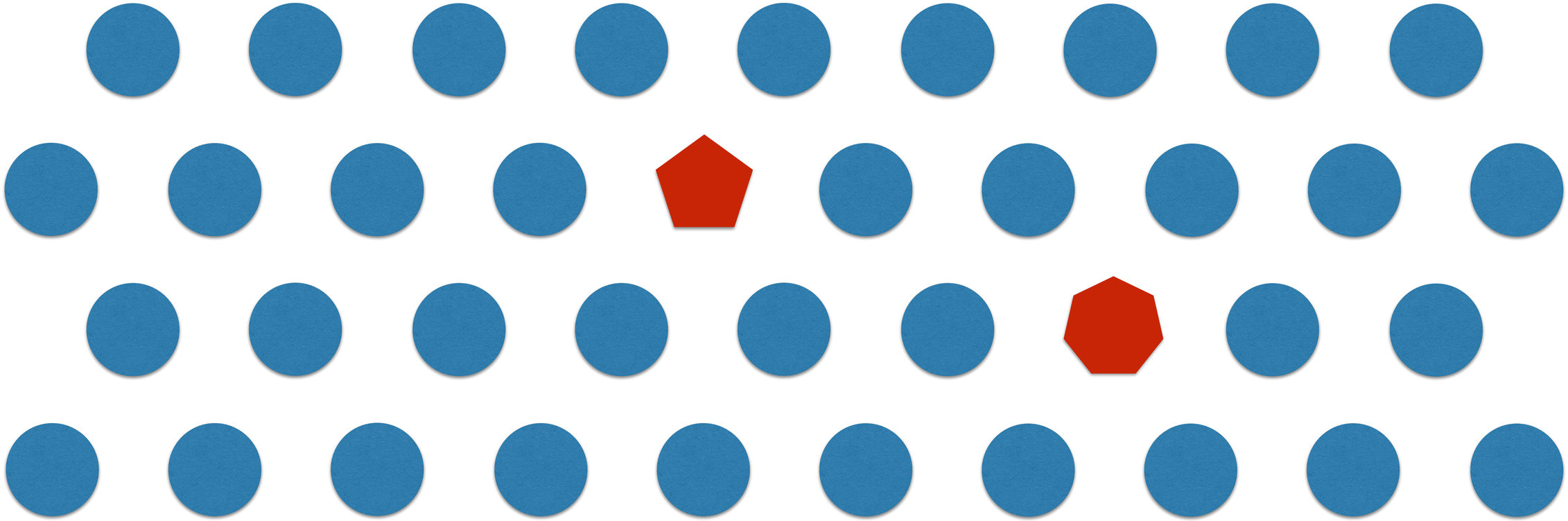
Rumor



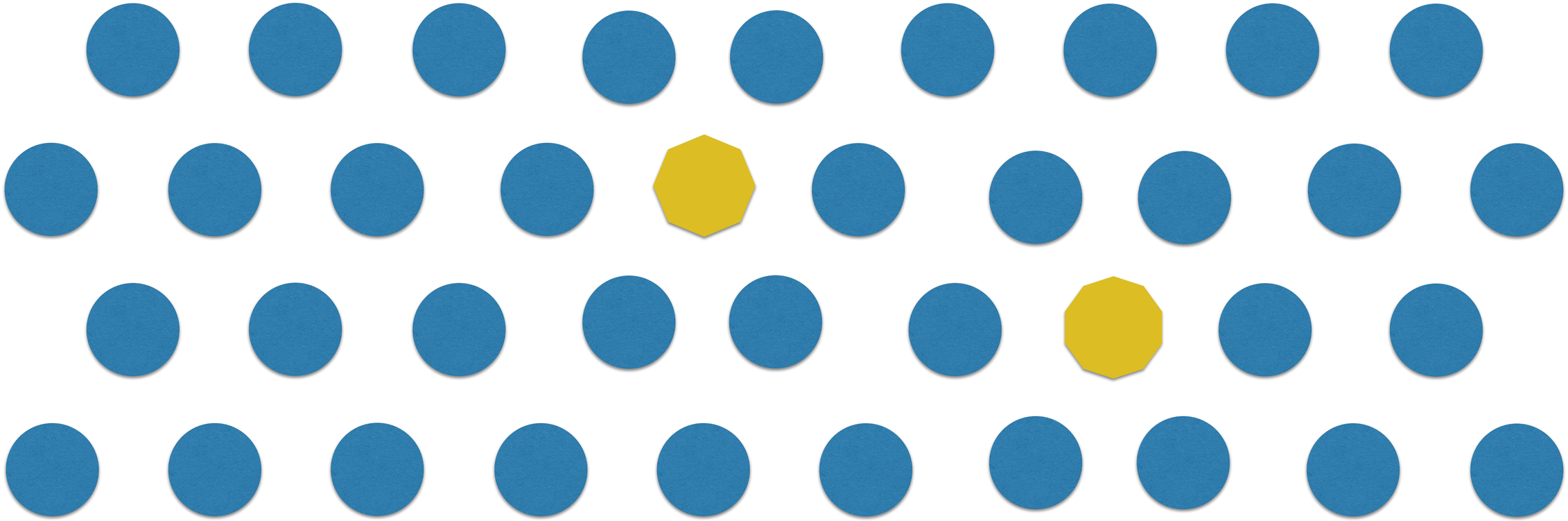
Anti-entropy



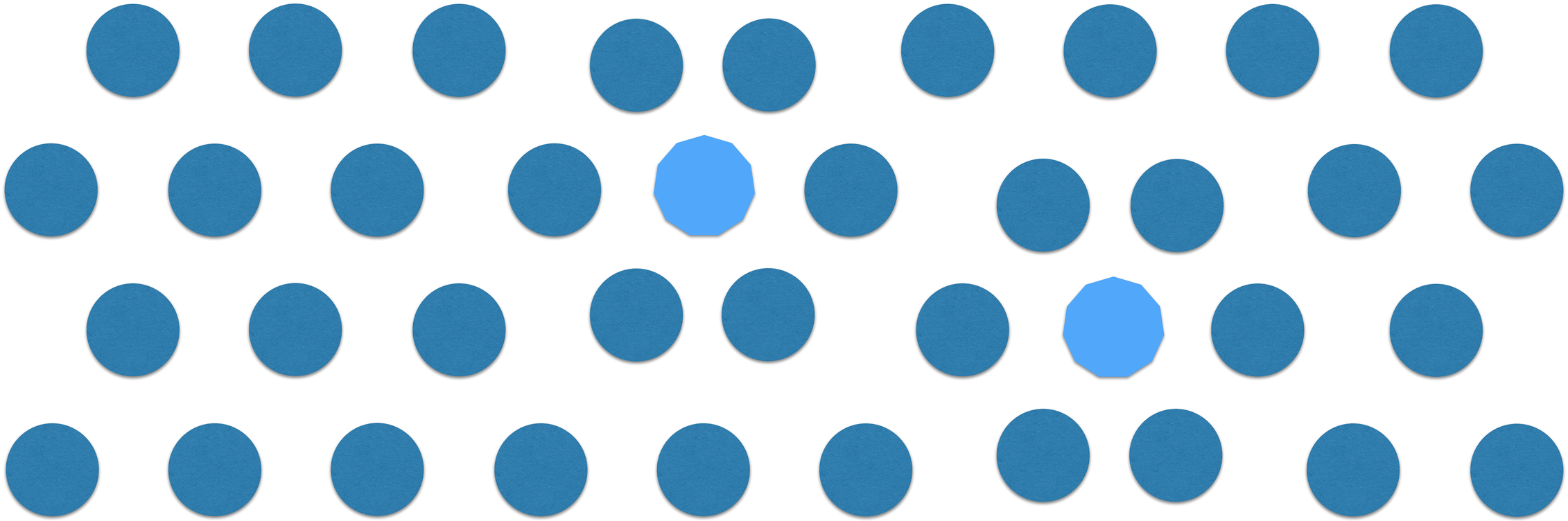
Anti-entropy



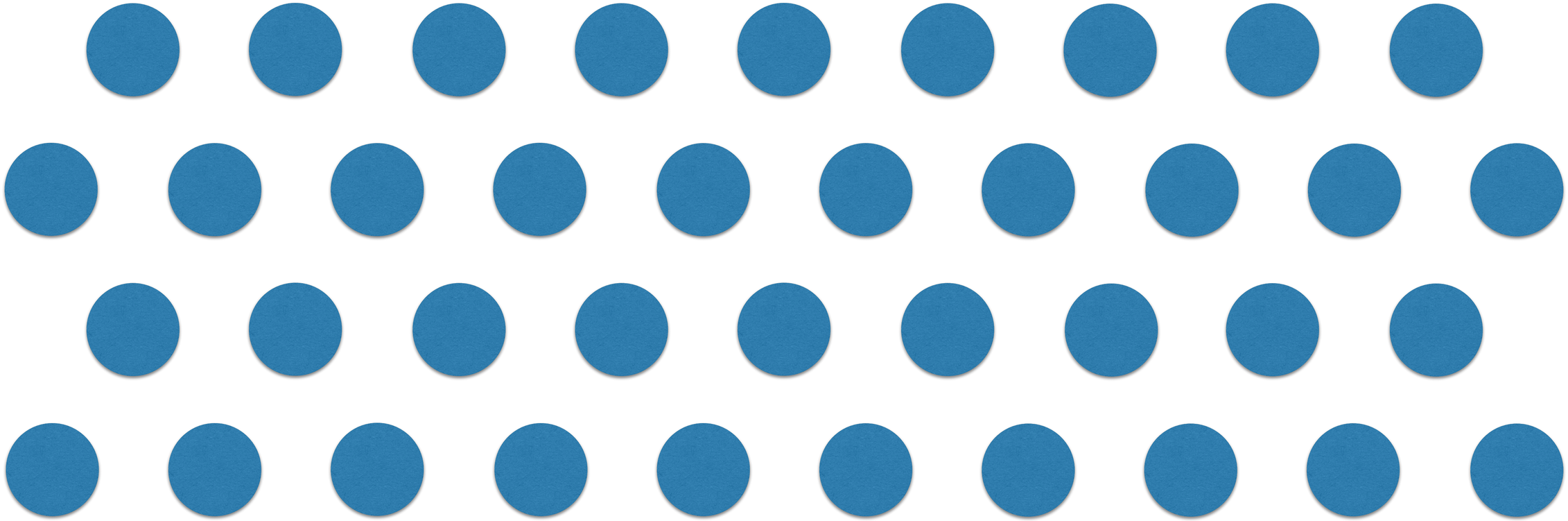
Anti-entropy



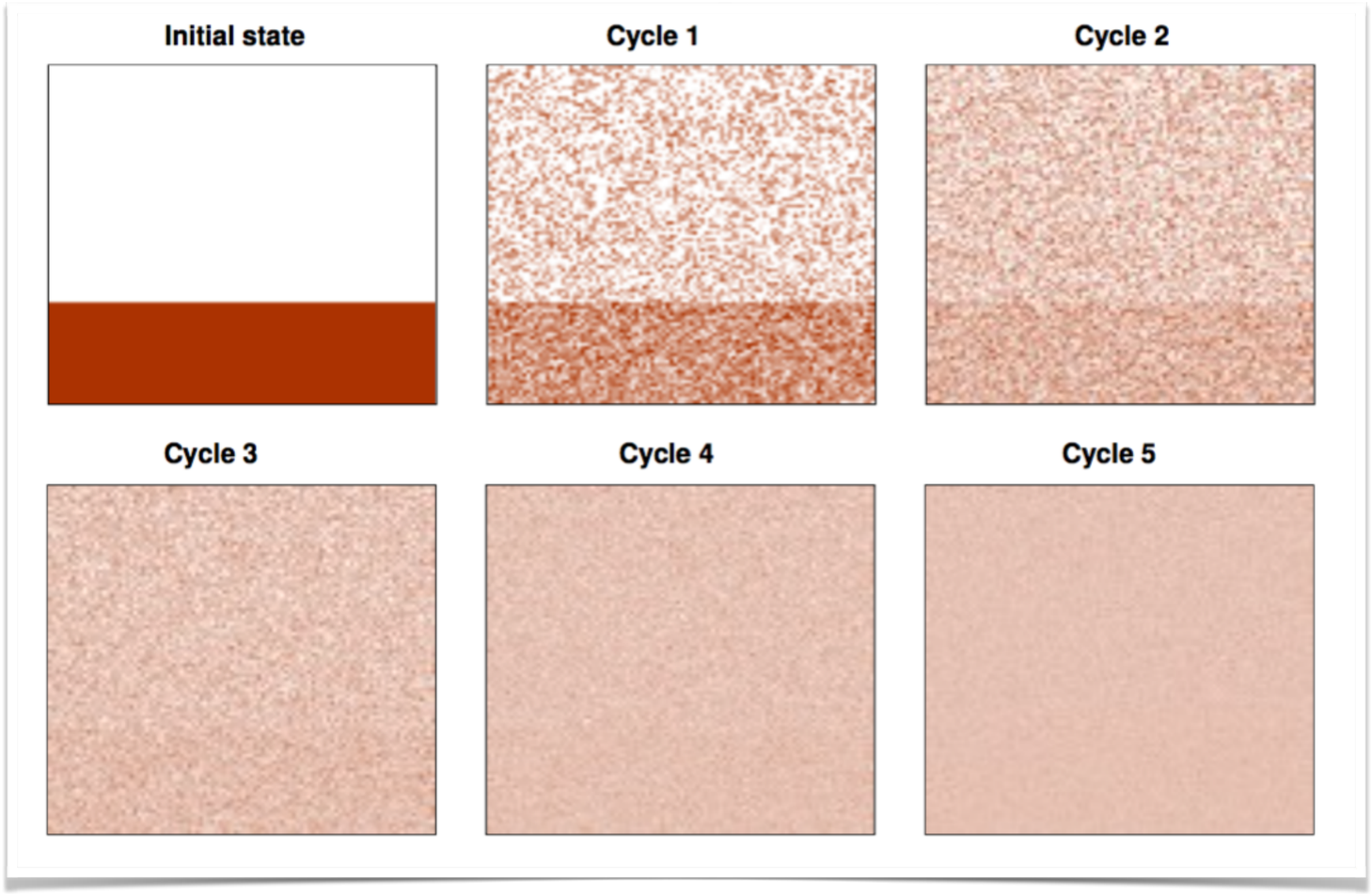
Anti-entropy



Anti-entropy gossip

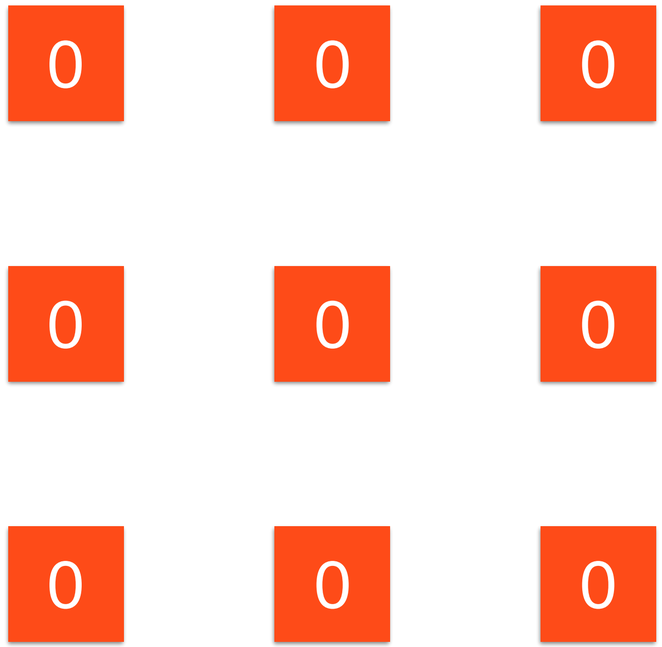


Aggregate



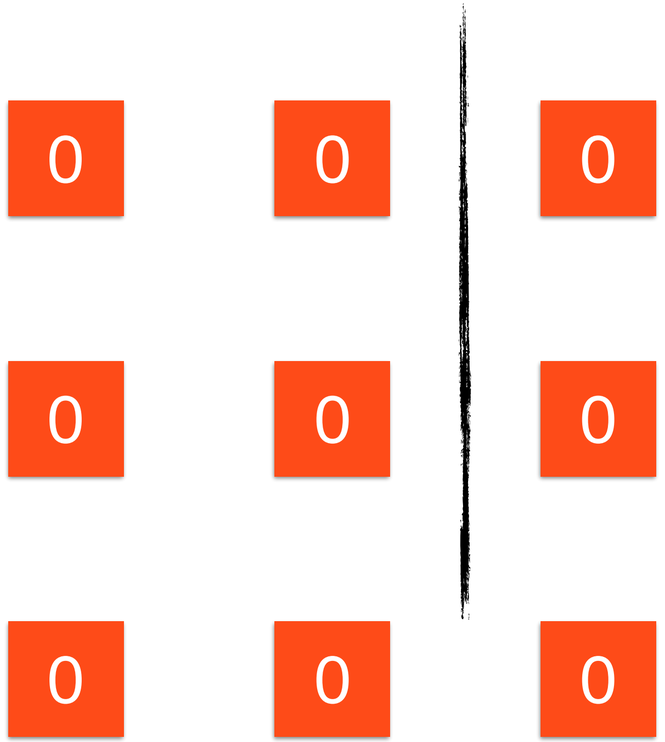
Motivating Example

Increment-only counter



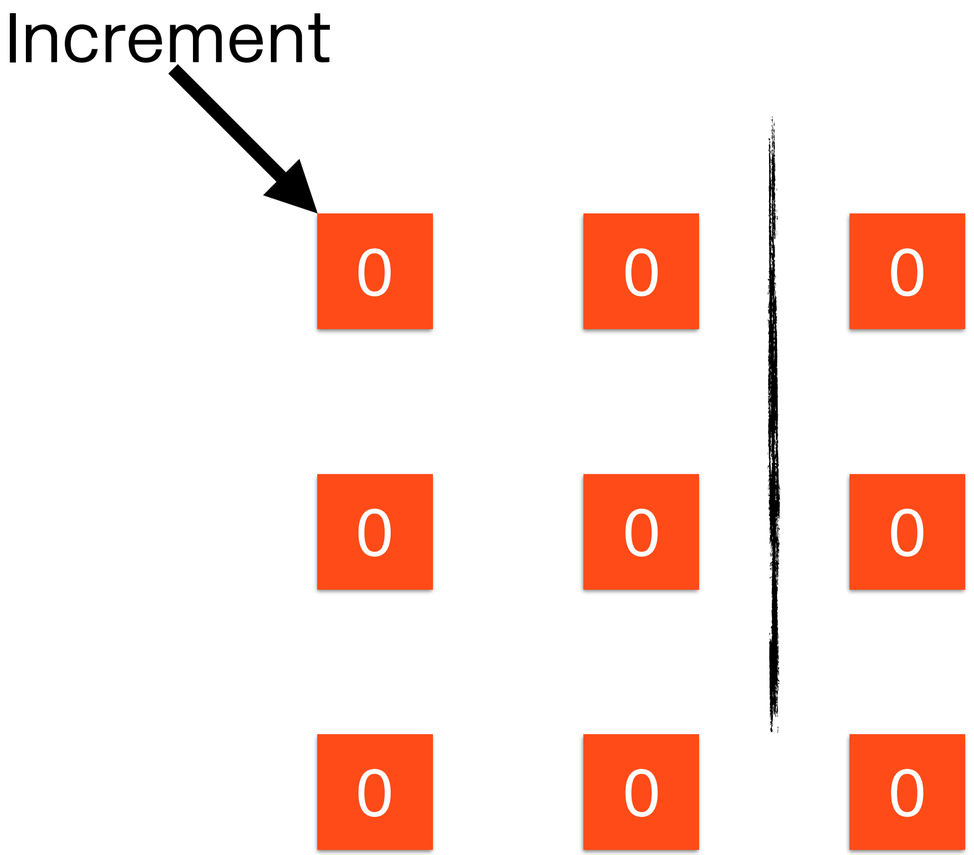
Motivating Example

Increment-only counter



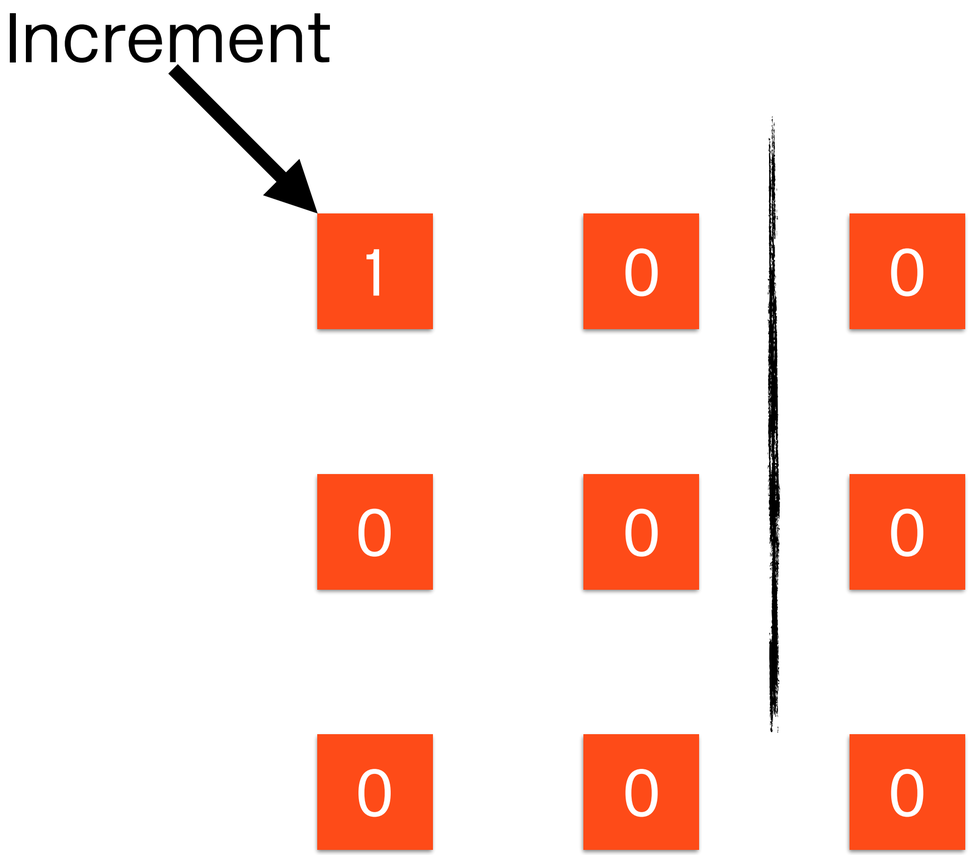
Motivating Example

Increment-only counter



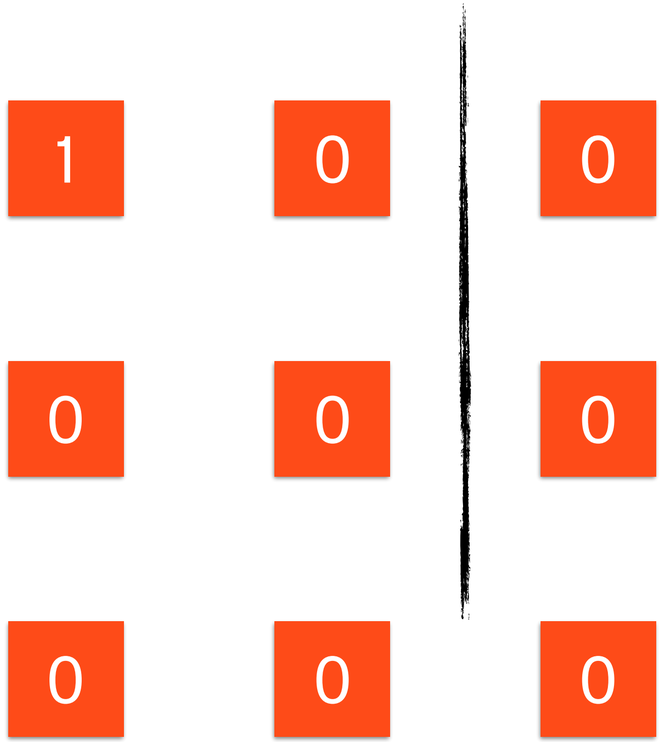
Motivating Example

Increment-only counter



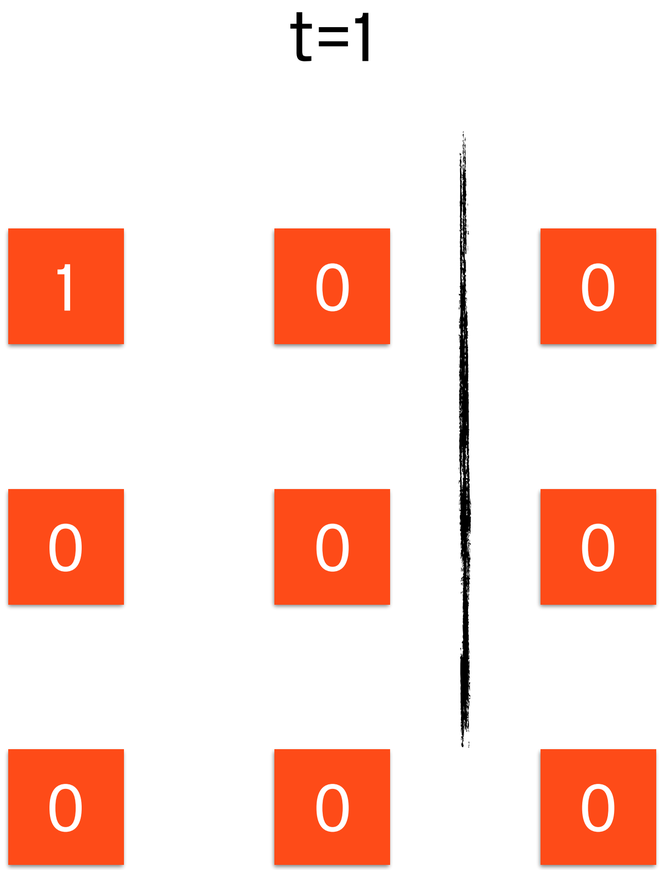
Motivating Example

Increment-only counter



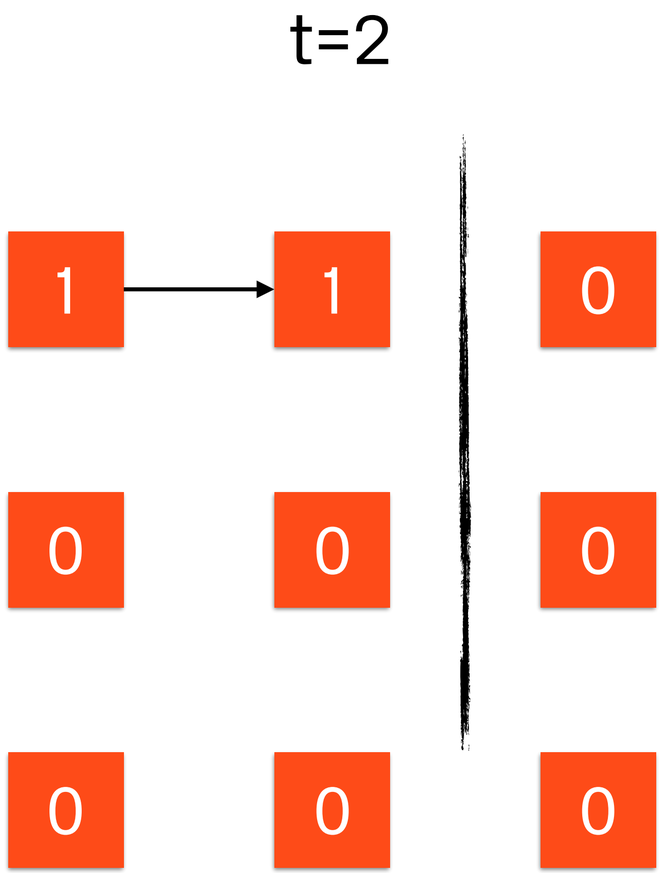
Motivating Example

Increment-only counter



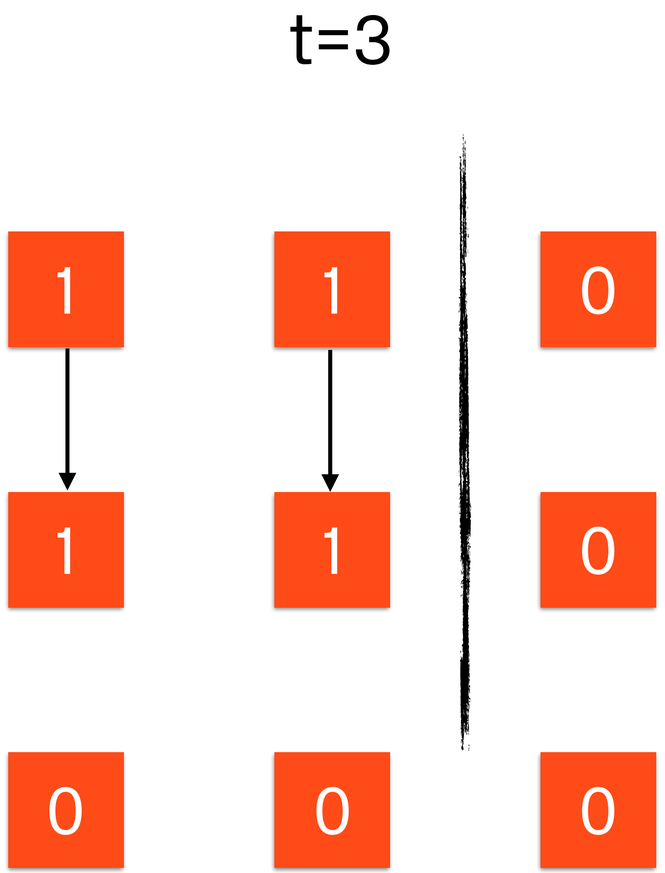
Motivating Example

Increment-only counter



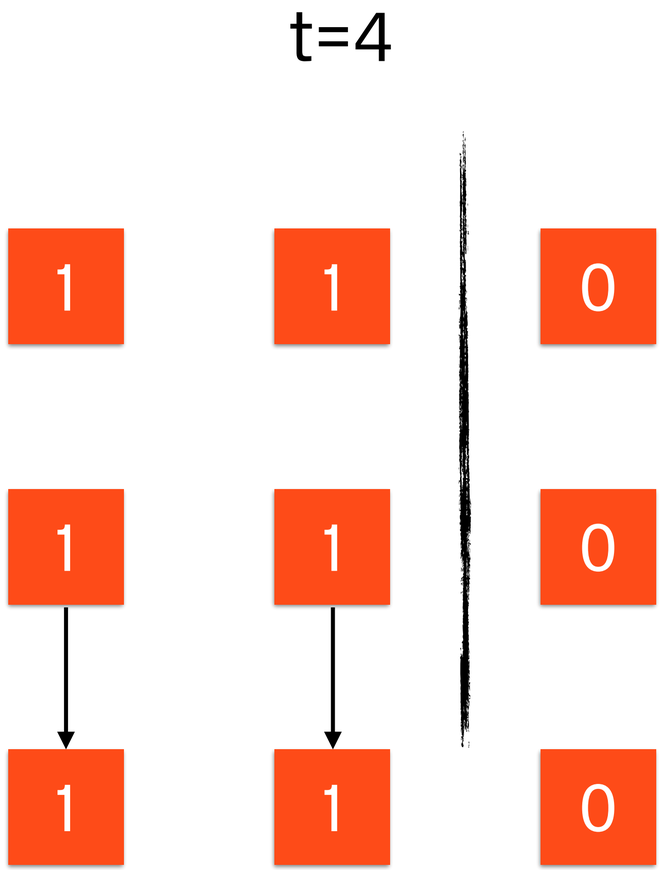
Motivating Example

Increment-only counter



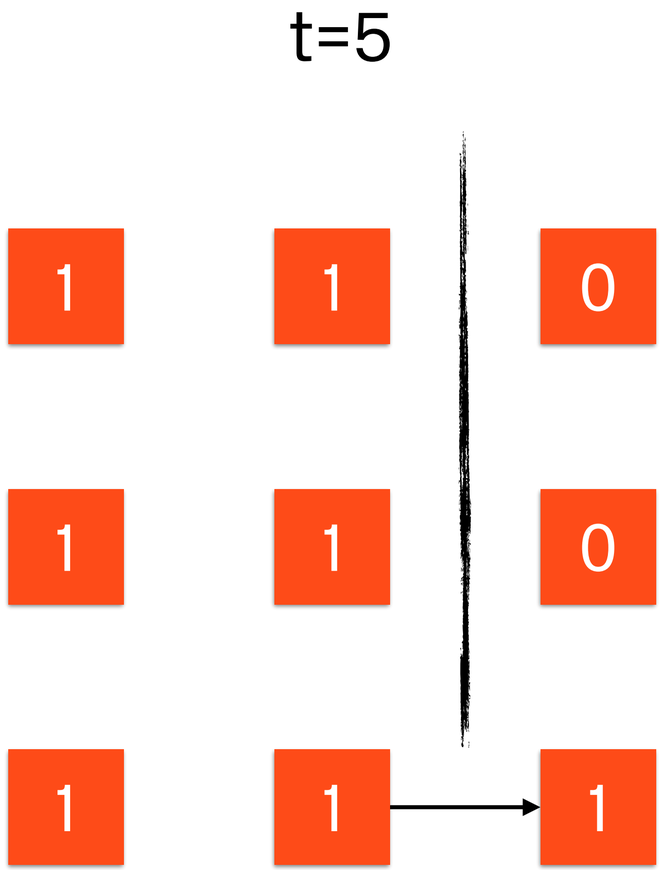
Motivating Example

Increment-only counter



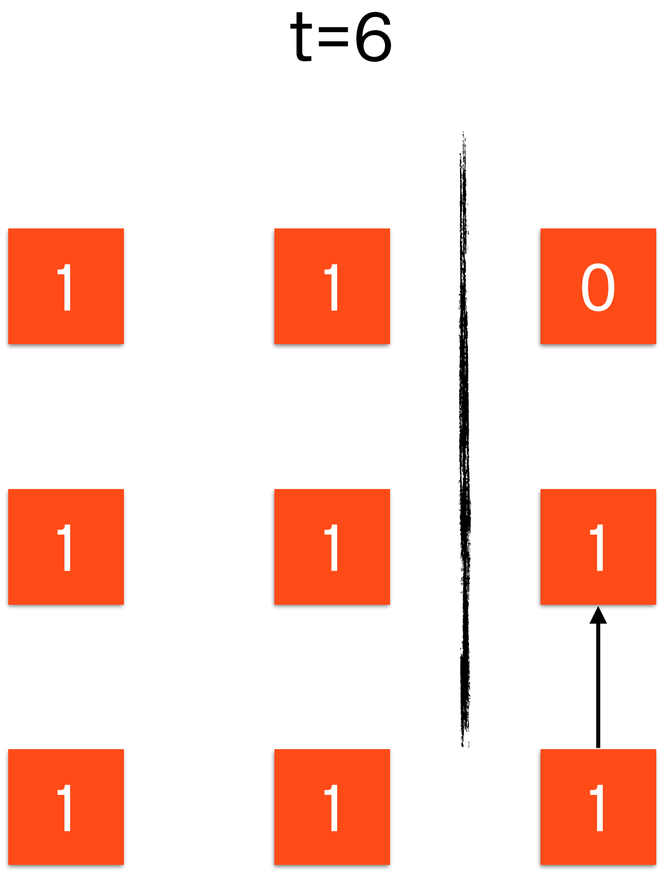
Motivating Example

Increment-only counter



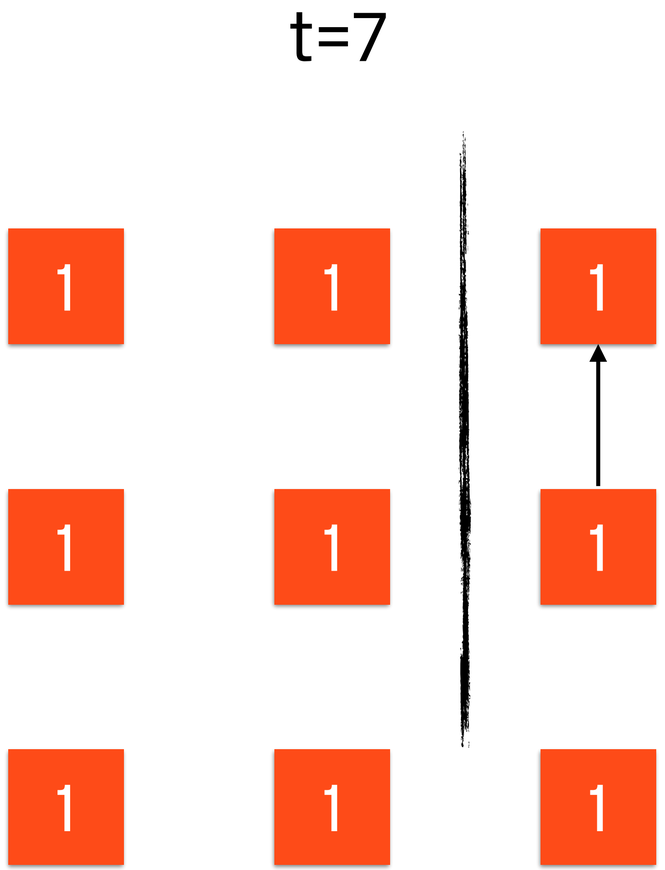
Motivating Example

Increment-only counter



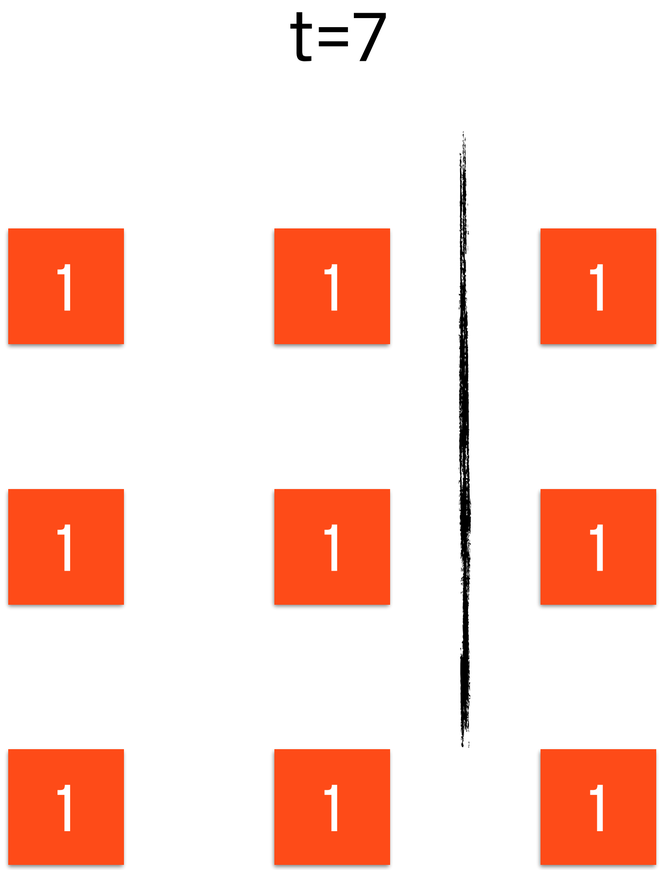
Motivating Example

Increment-only counter



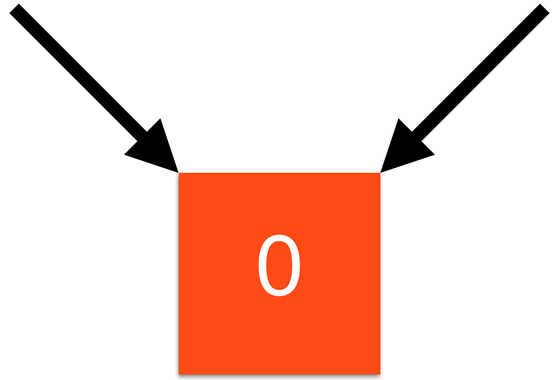
Motivating Example

Increment-only counter



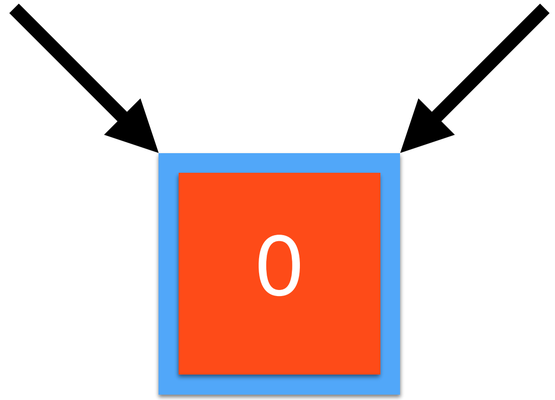
Motivating Example

Increment-only counter



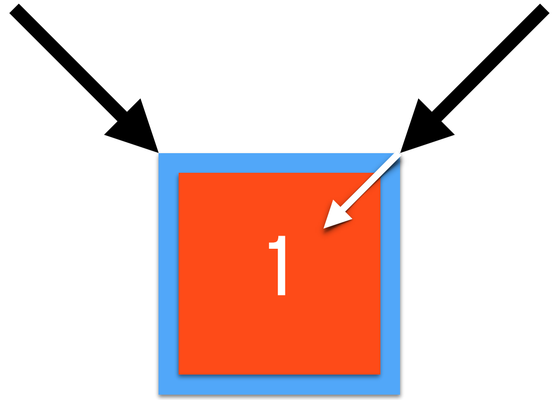
Motivating Example

Increment-only counter



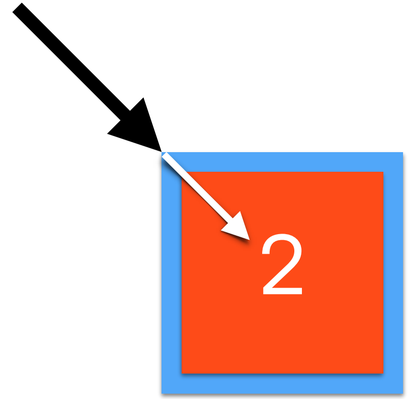
Motivating Example

Increment-only counter



Motivating Example

Increment-only counter

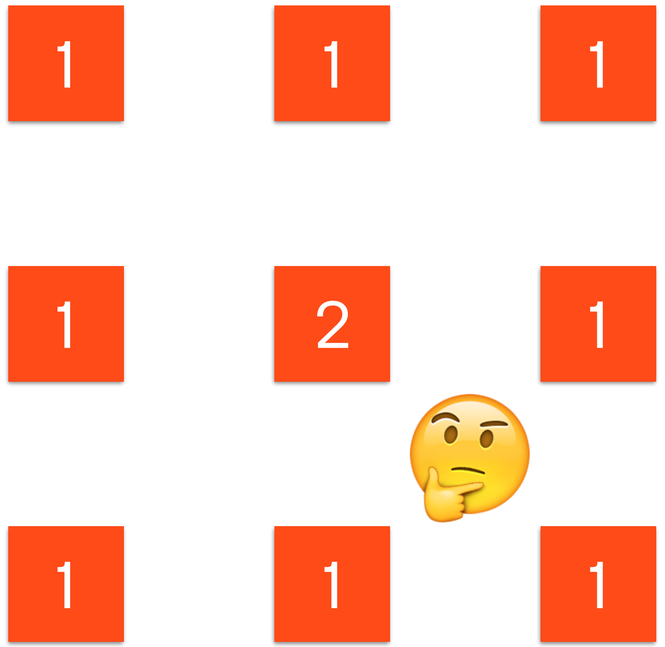


Motivating Example

Increment-only counter

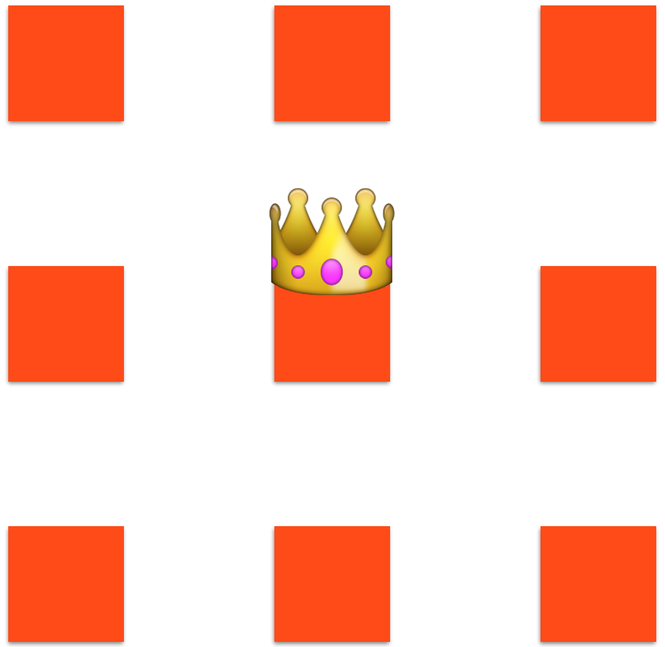
Motivating Example

Increment-only counter



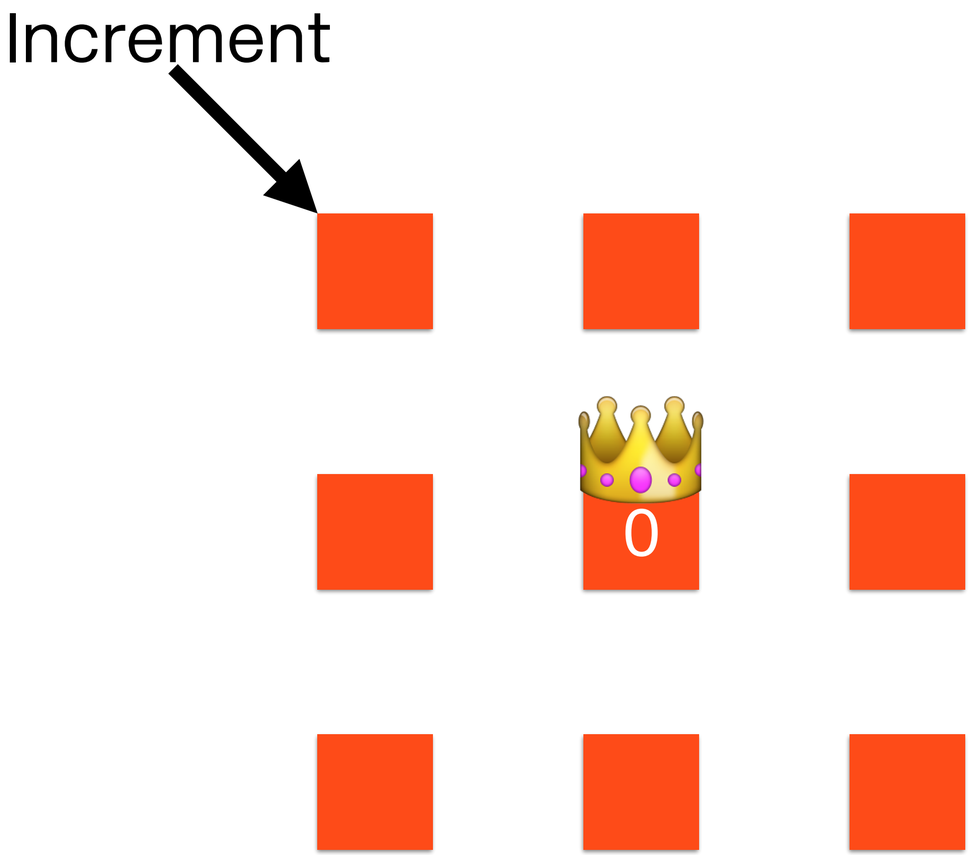
Motivating Example

Increment-only counter



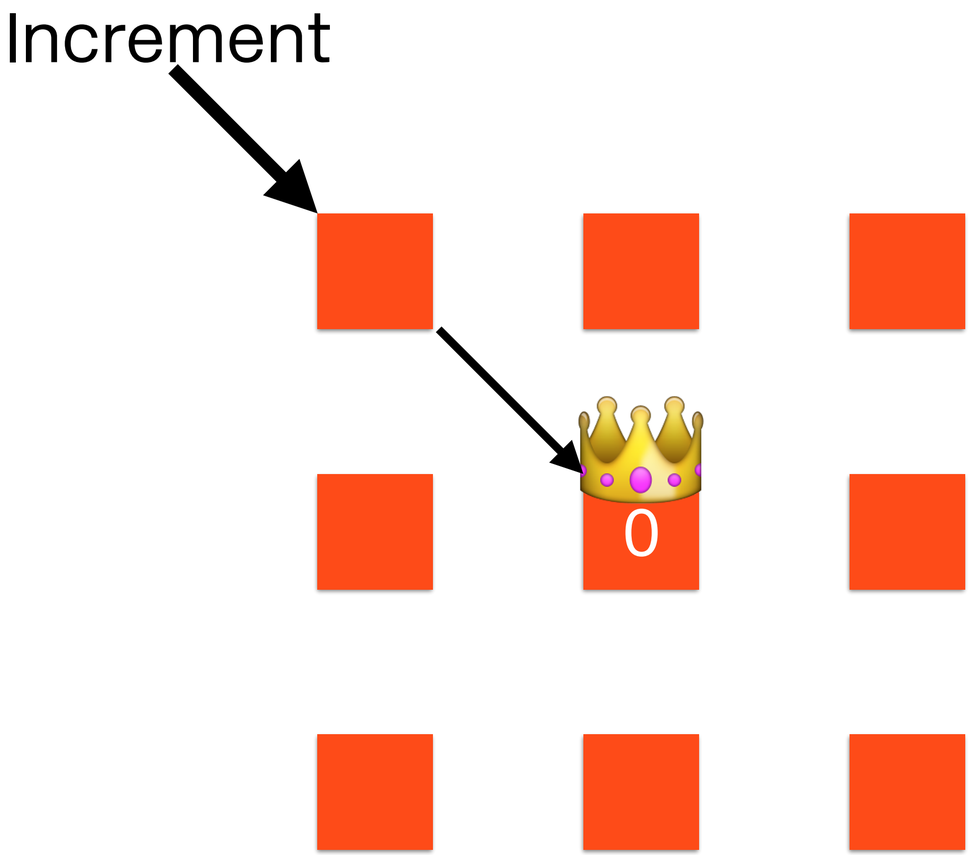
Motivating Example

Increment-only counter



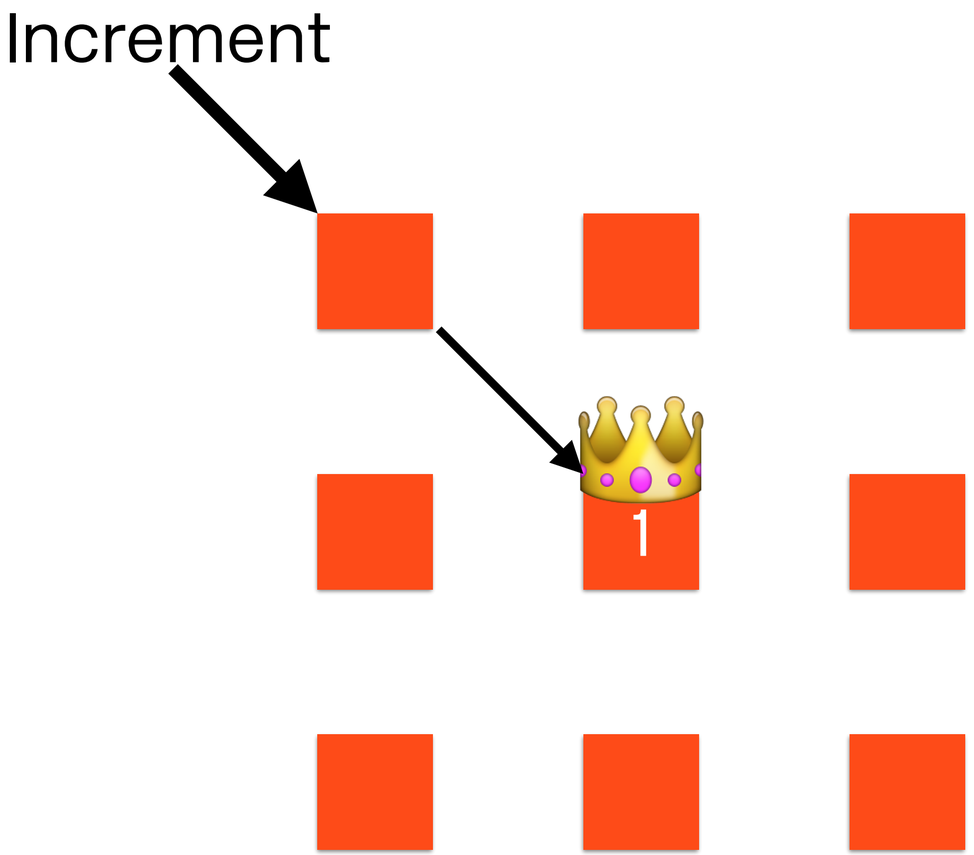
Motivating Example

Increment-only counter



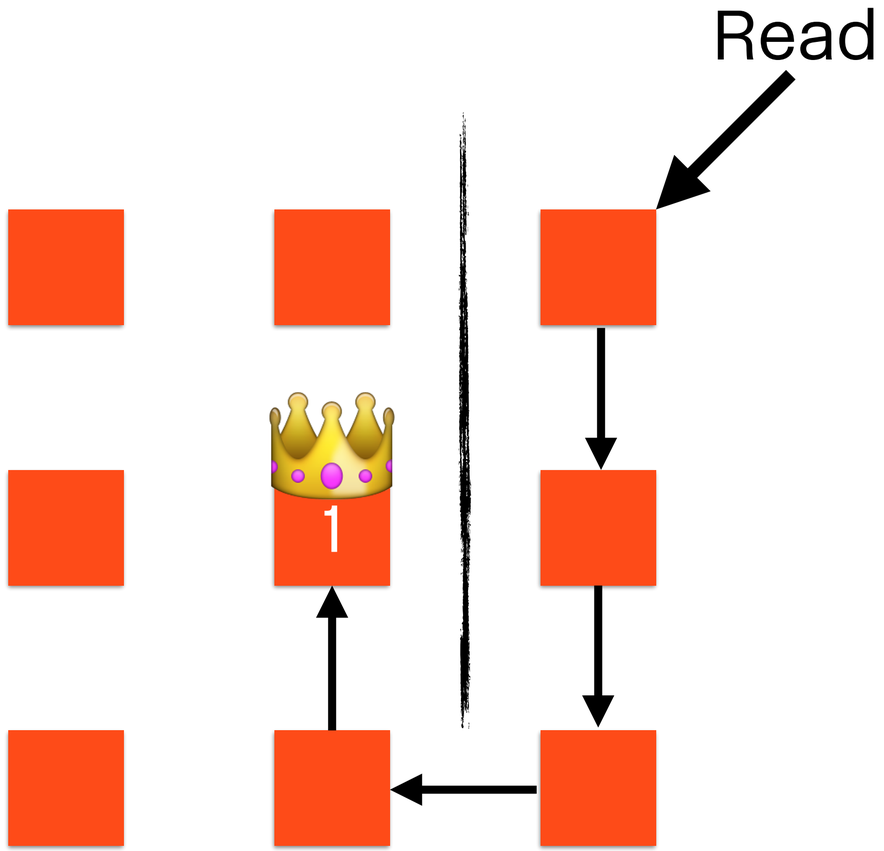
Motivating Example

Increment-only counter



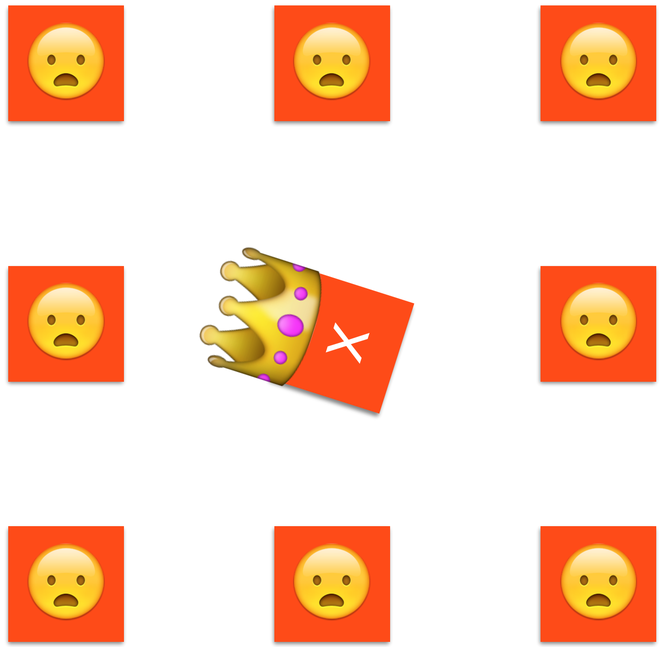
Motivating Example

Increment-only counter



Motivating Example

Increment-only counter



Theory:
CRDT

CRDTs solve…

- Shared state in a distributed system
  - Provide availability (A)
  - Provide partition tolerance (P)

BETWEEN THE DEVIL AND THE DEEP BLUE SEA

Distributed transactions

poor performance, operational problems, ...

OR

Eventual consistency

more like perpetual inconsistency, amirite?

BETWEEN THE DEVIL AND THE DEEP BLUE SEA

Distributed transactions

poor performance, operational problems, ...

OR

Eventual consistency

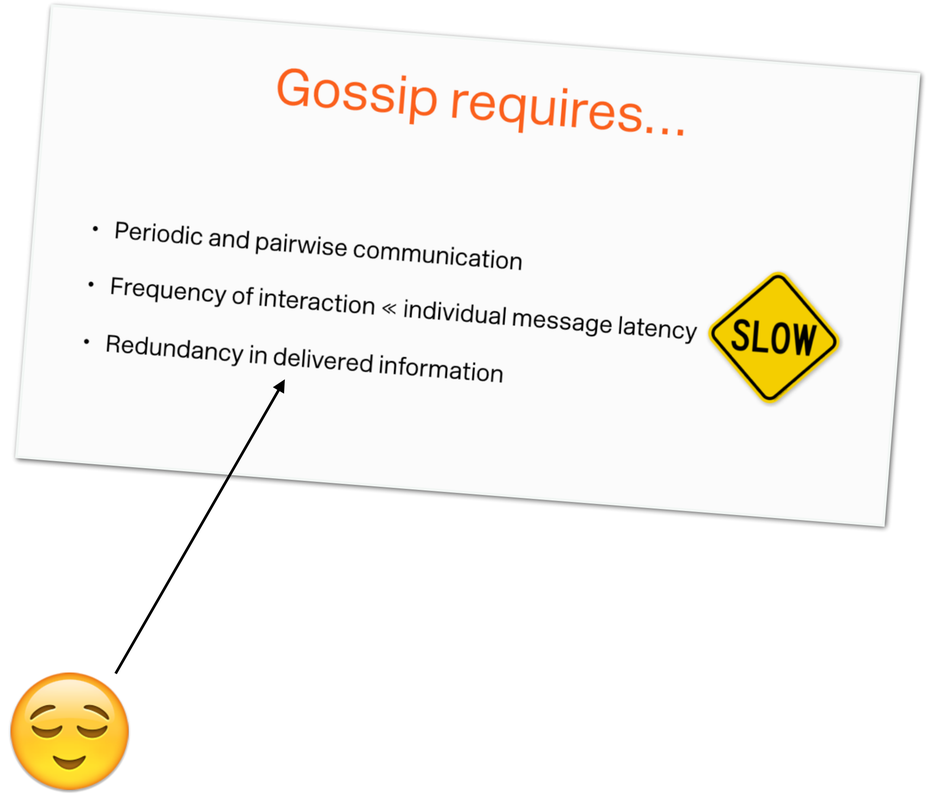
more like perpetual inconsistency, amirite?

CRDTs require...

- Operations that are
  - Associative — A • (B • C) = (A • B) • C
  - Commutative — A • B = B • A
  - Idempotent* — A • A = A

CRDTs require...

- Operations that are
  - Associative — A • (B • C) = (A • B) • C
  - Commutative — A • B = B • A
  - Idempotent* — A • A = A



CRDT

- Conflict-free
- Replicated
- Data
- Type

CmRDT

- Commutative Replicated Data Type
- Operation-based: transmit the op itself — "edge trigger"
- Assumes reliable, exactly-once delivery (ha!)

Motivating Example

Increment-only counter

CmRDT

Motivating Example

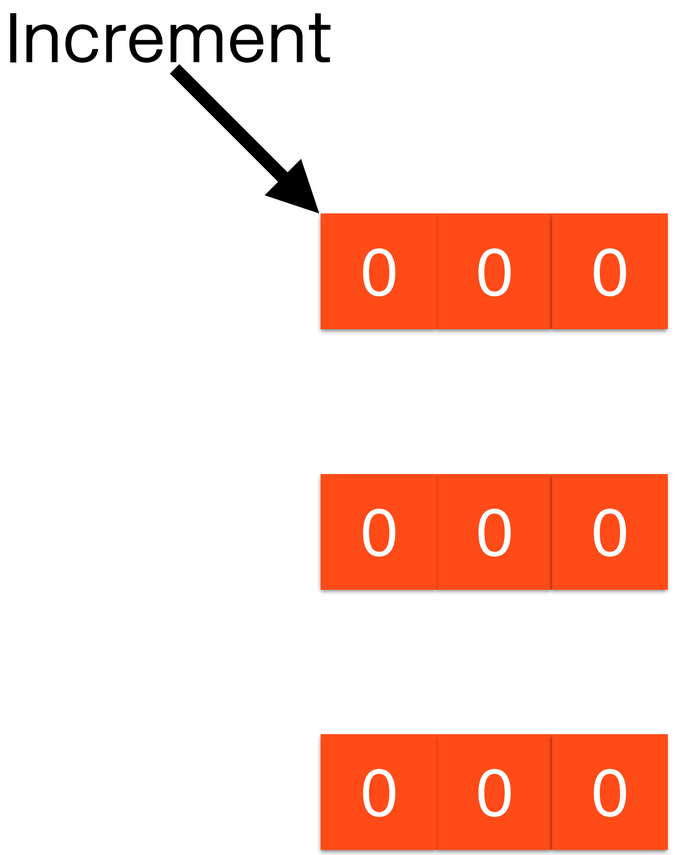
Increment-only counter

CmRDT

Motivating Example

Increment-only counter

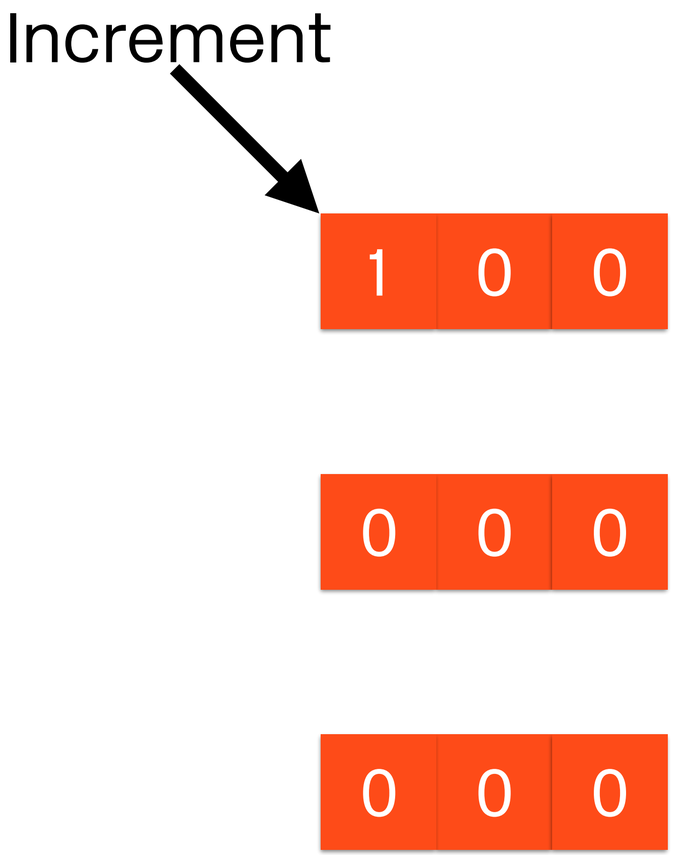
CmRDT



Motivating Example

Increment-only counter

CmRDT



Motivating Example

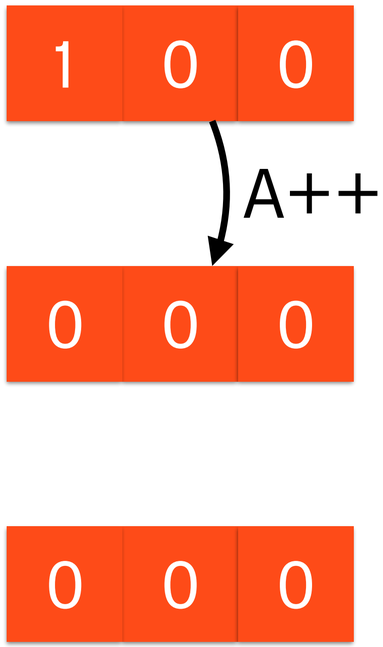
Increment-only counter

CmRDT

Motivating Example

Increment-only counter

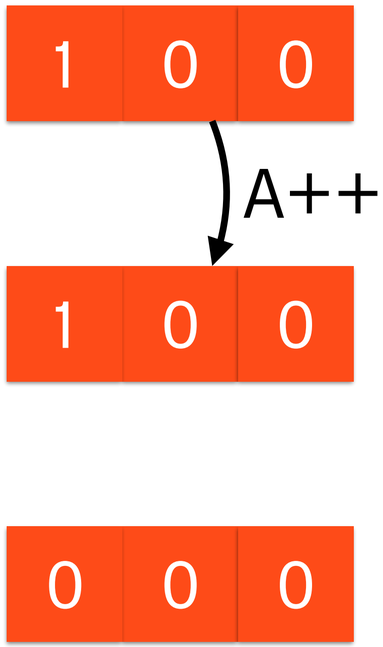
CmRDT



Motivating Example

Increment-only counter

CmRDT



Motivating Example

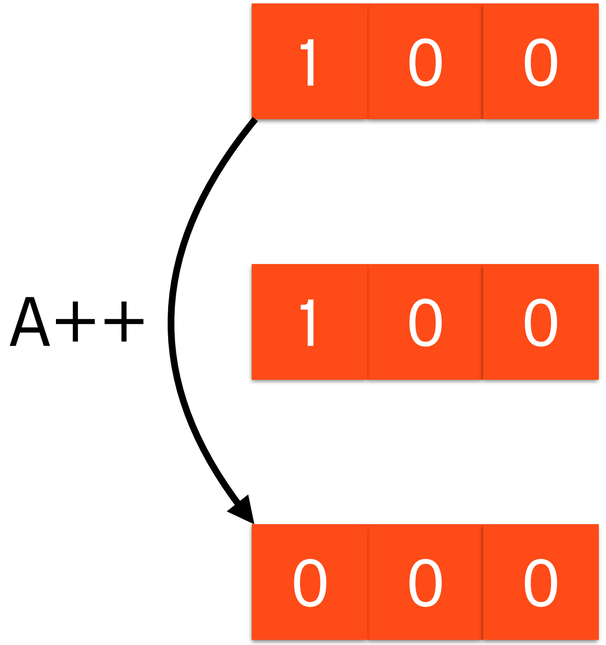
Increment-only counter

CmRDT

Motivating Example

Increment-only counter

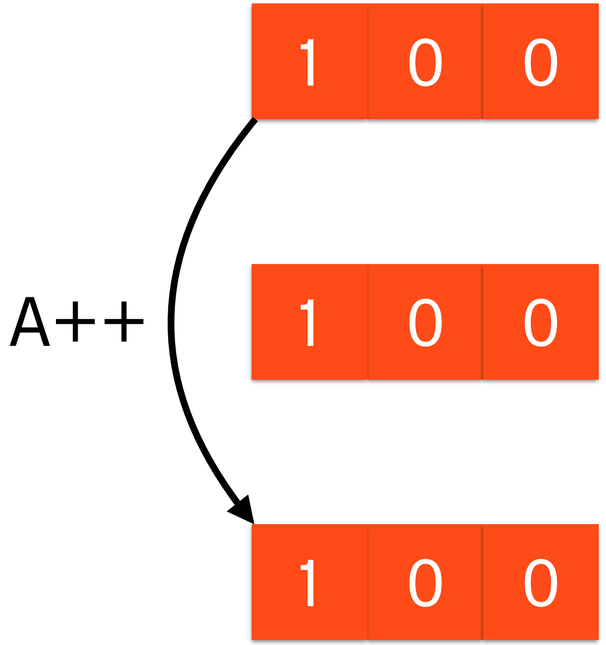
CmRDT



Motivating Example

Increment-only counter

CmRDT



Motivating Example

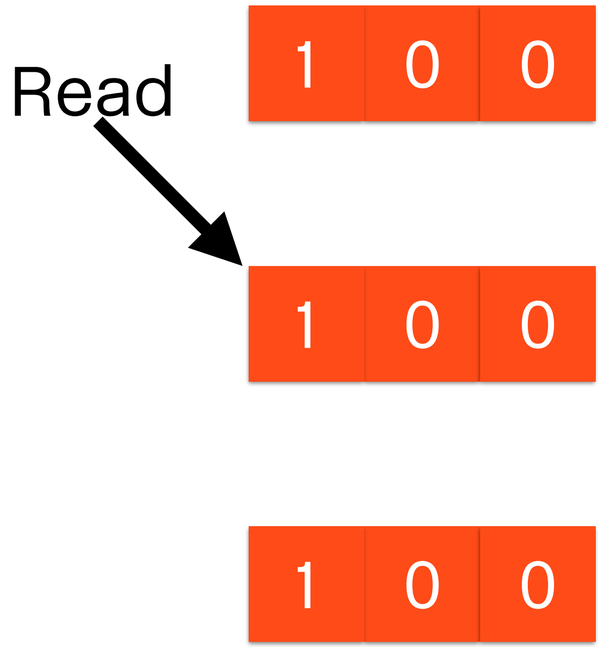
Increment-only counter

CmRDT

Motivating Example

Increment-only counter

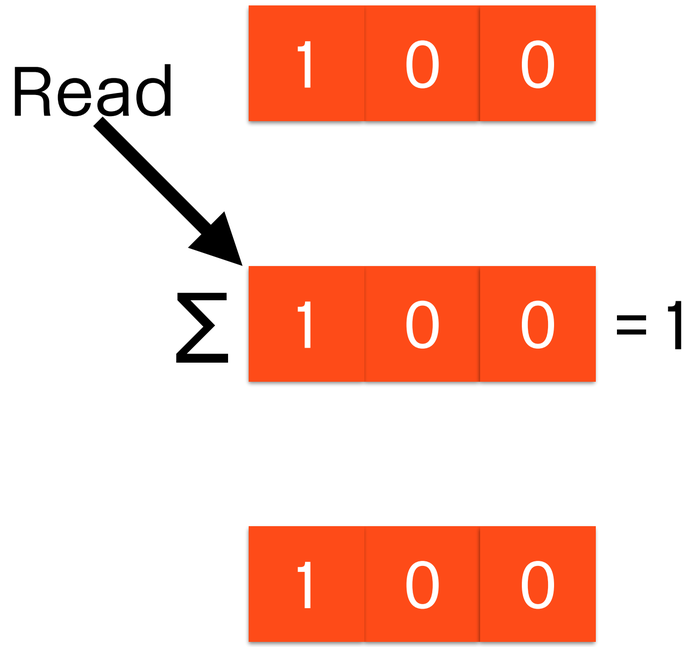
CmRDT



Motivating Example

Increment-only counter

CmRDT



CvRDT

- Convergent Replicated Data Type
- State-based: transmit the complete state — "level trigger"
- Achieve idempotency in fragile networks
- All CmRDT can be expressed as CvRDT (Shapiro et. al.)

Motivating Example

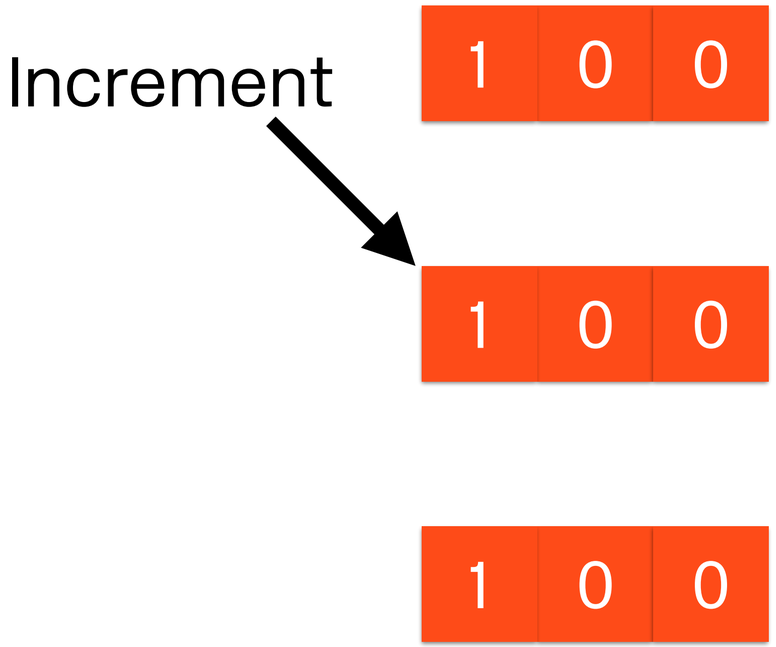
Increment-only counter

CmRDT

Motivating Example

Increment-only counter

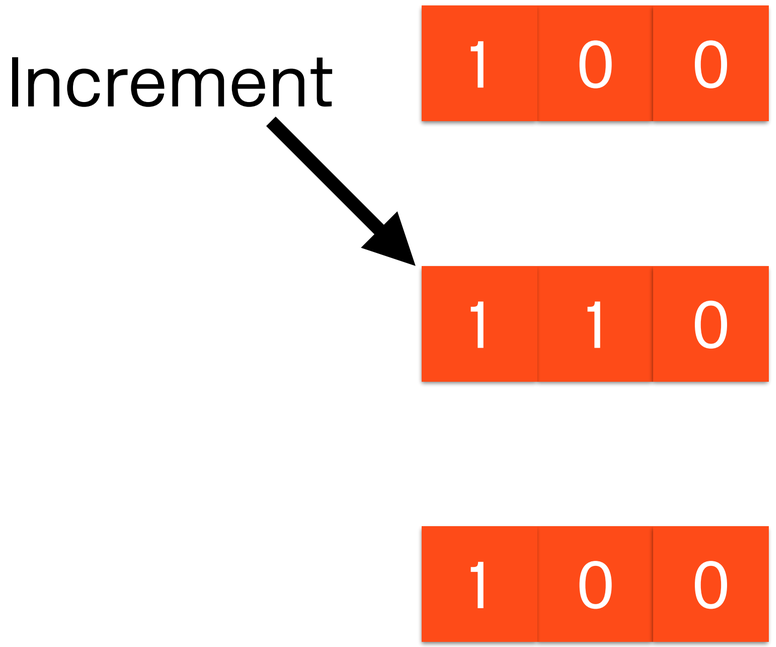
CmRDT



Motivating Example

Increment-only counter

CmRDT



Motivating Example

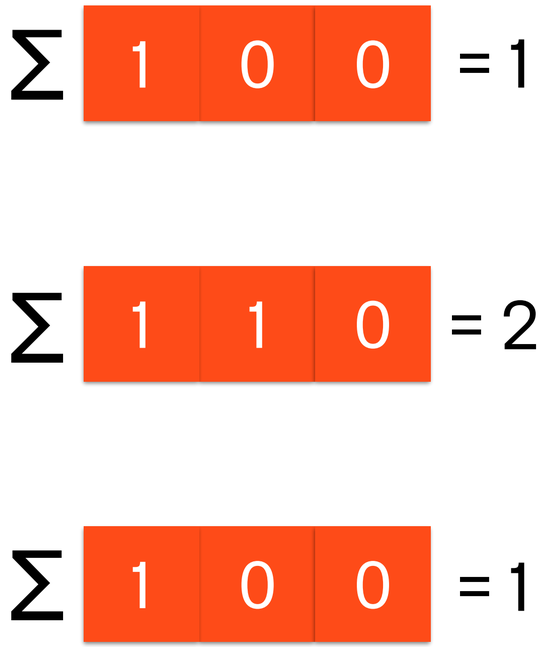
Increment-only counter

CmRDT

Motivating Example

Increment-only counter

CmRDT



Motivating Example

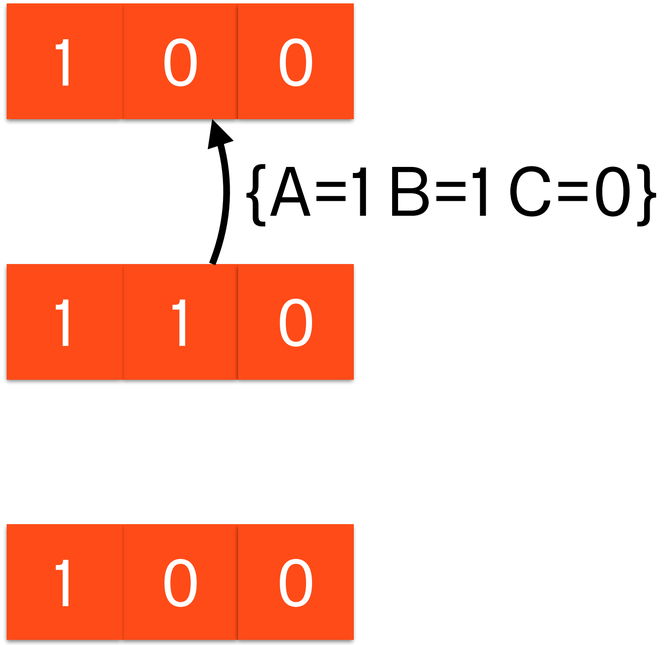
Increment-only counter

CmRDT

Motivating Example

Increment-only counter

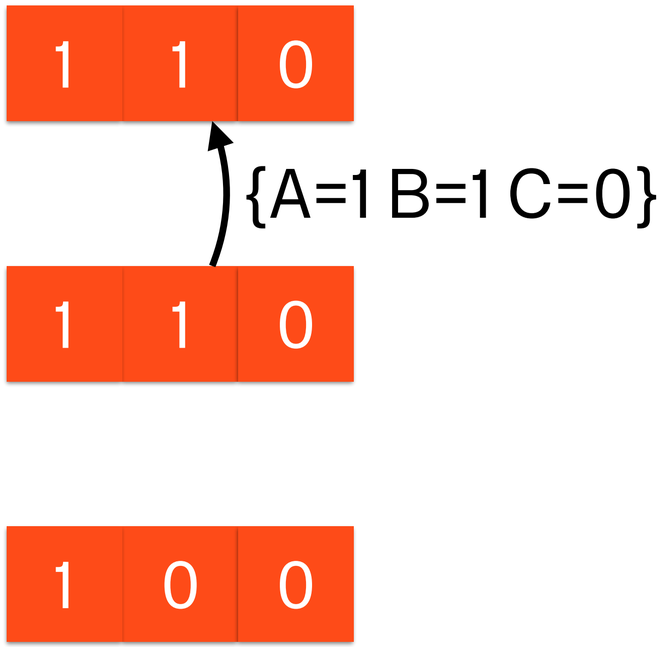
CmRDT



Motivating Example

Increment-only counter

CmRDT



Motivating Example

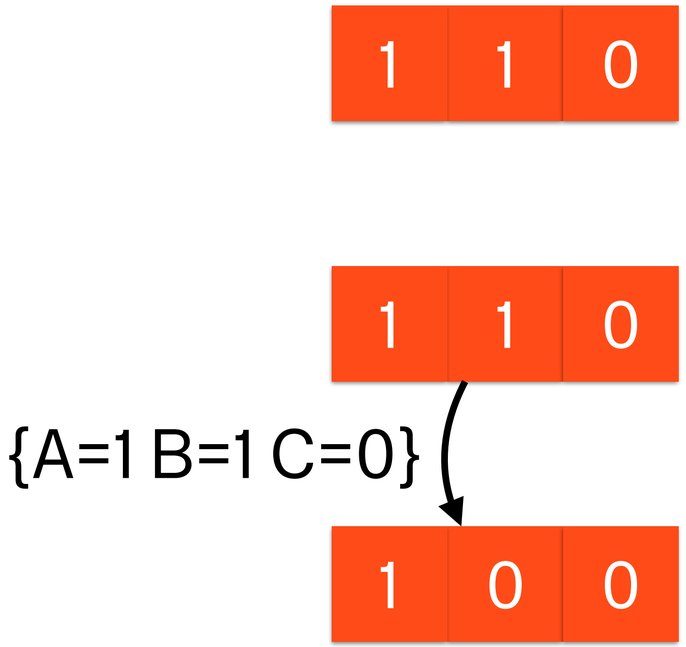
Increment-only counter

CmRDT

Motivating Example

Increment-only counter

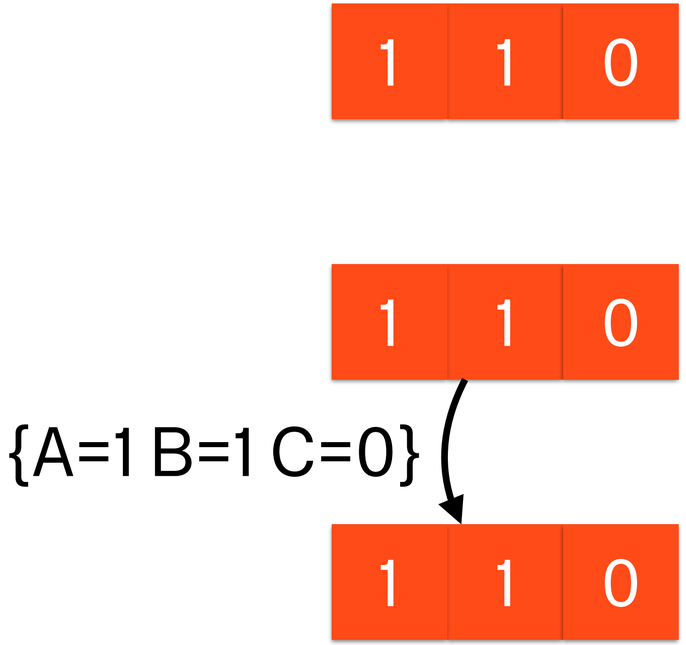
CmRDT



Motivating Example

Increment-only counter

CmRDT



Motivating Example

Increment-only counter

CmRDT

δ-CRDT

- Op-based CRDT: small/fast messages, needs impossible network
- State-based CRDT: idempotent, but large message size
- Delta CRDT: small messages + idempotency?

δ-CRDT

- Delta-mutator m^{δ} is an update as a function — also a CRDT
- Given prior state X, $m^{\delta}(X)$ encodes next state
- Multiple m^{δ} can be merged to a set M^{δ} and transmit
- Causal consistency only possible with constraints & anti-entropy!

Motivating Example

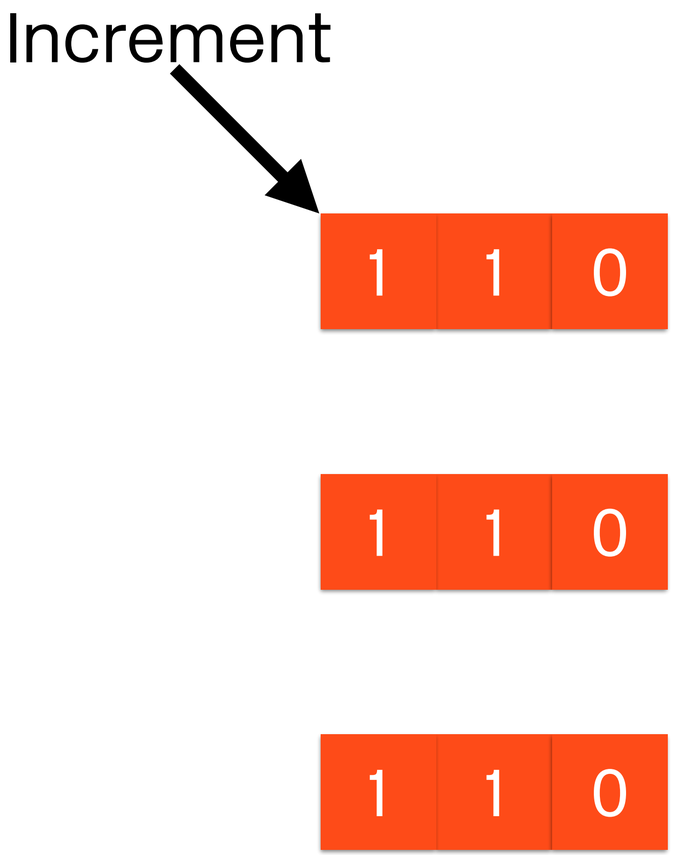
Increment-only counter

δ-CRDT

Motivating Example

Increment-only counter

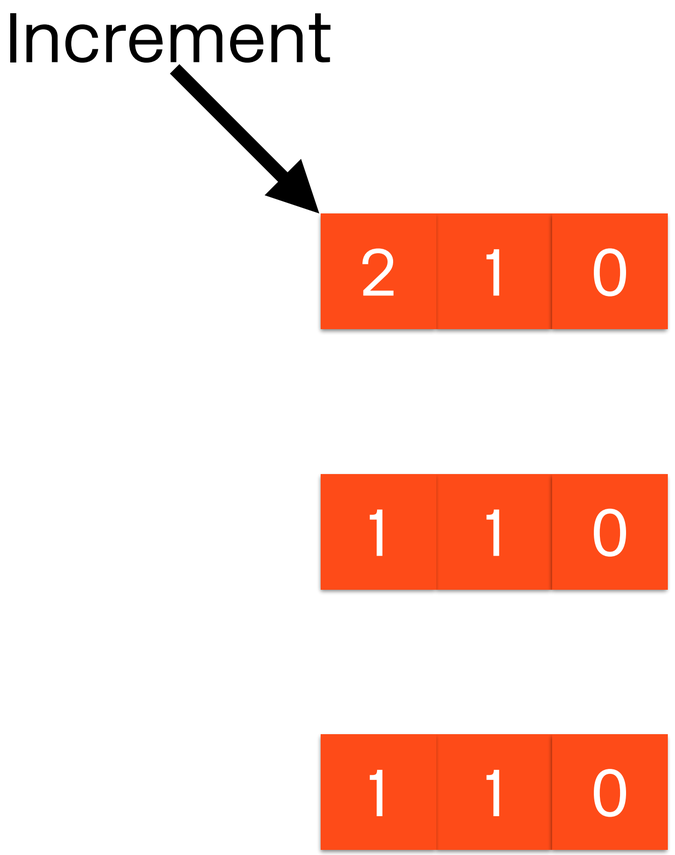
δ-CRDT



Motivating Example

Increment-only counter

δ-CRDT



Motivating Example

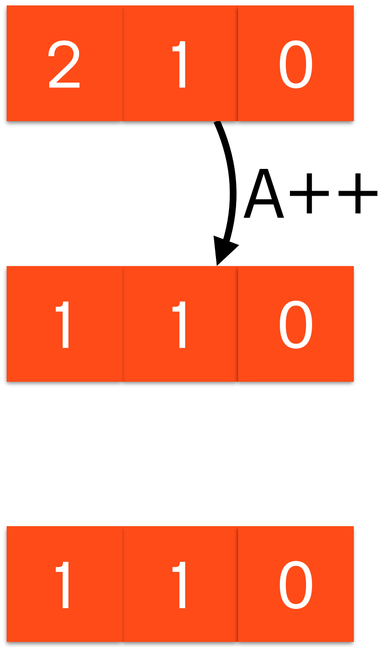
Increment-only counter

δ-CRDT

Motivating Example

Increment-only counter

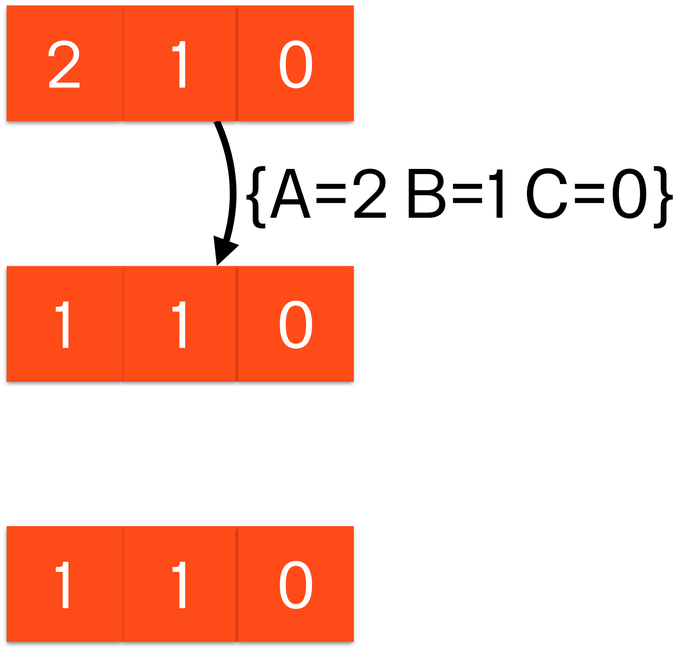
δ-CRDT



Motivating Example

Increment-only counter

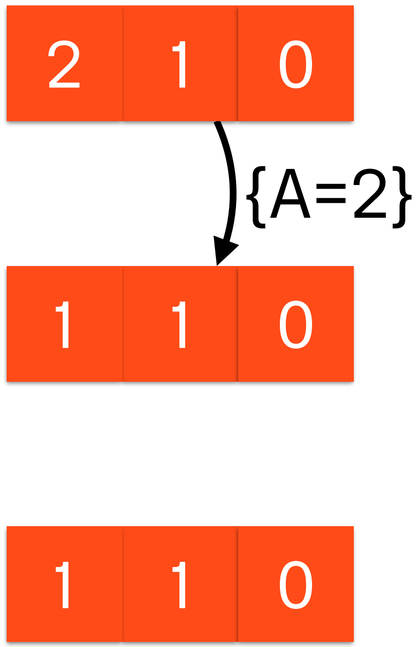
δ-CRDT



Motivating Example

Increment-only counter

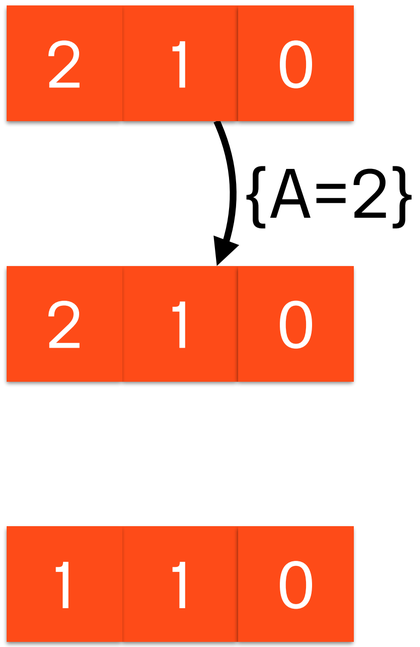
δ-CRDT



Motivating Example

Increment-only counter

δ-CRDT



Motivating Example

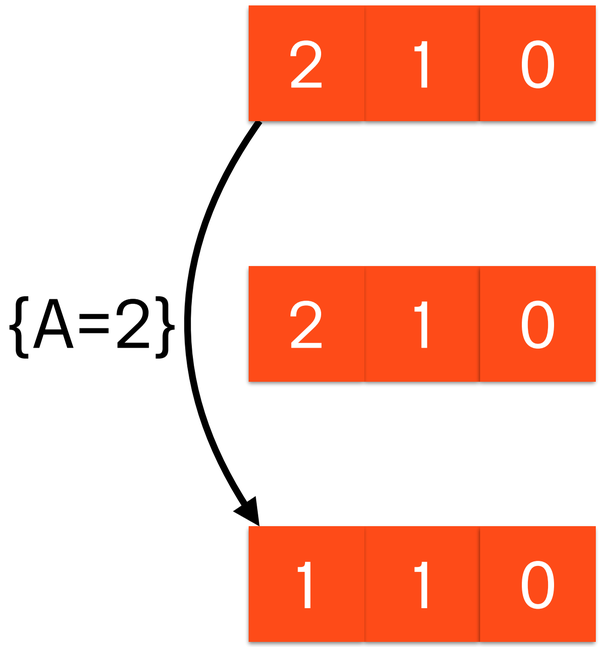
Increment-only counter

δ-CRDT

Motivating Example

Increment-only counter

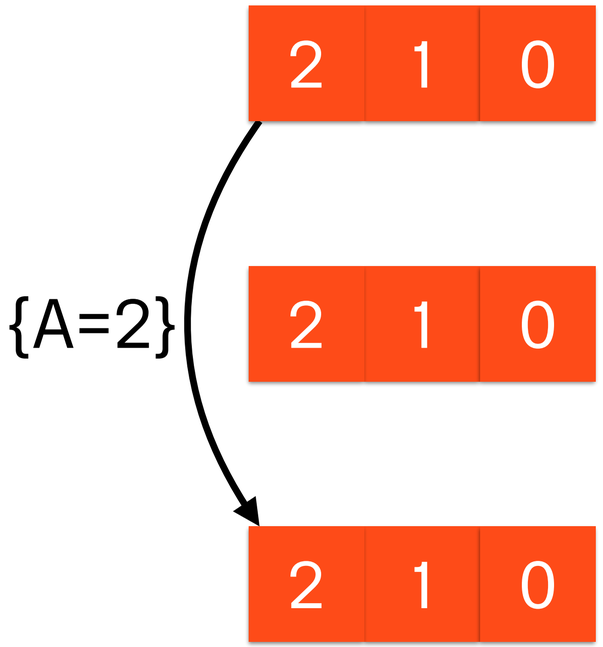
δ-CRDT



Motivating Example

Increment-only counter

δ-CRDT



Motivating Example

Increment-only counter

δ-CRDT

Theory → Practice

Communication Theory
+
Information Theory
=
System Theory

Gossip
+
CRDT
=
weavemesh

Practice:
weavemesh

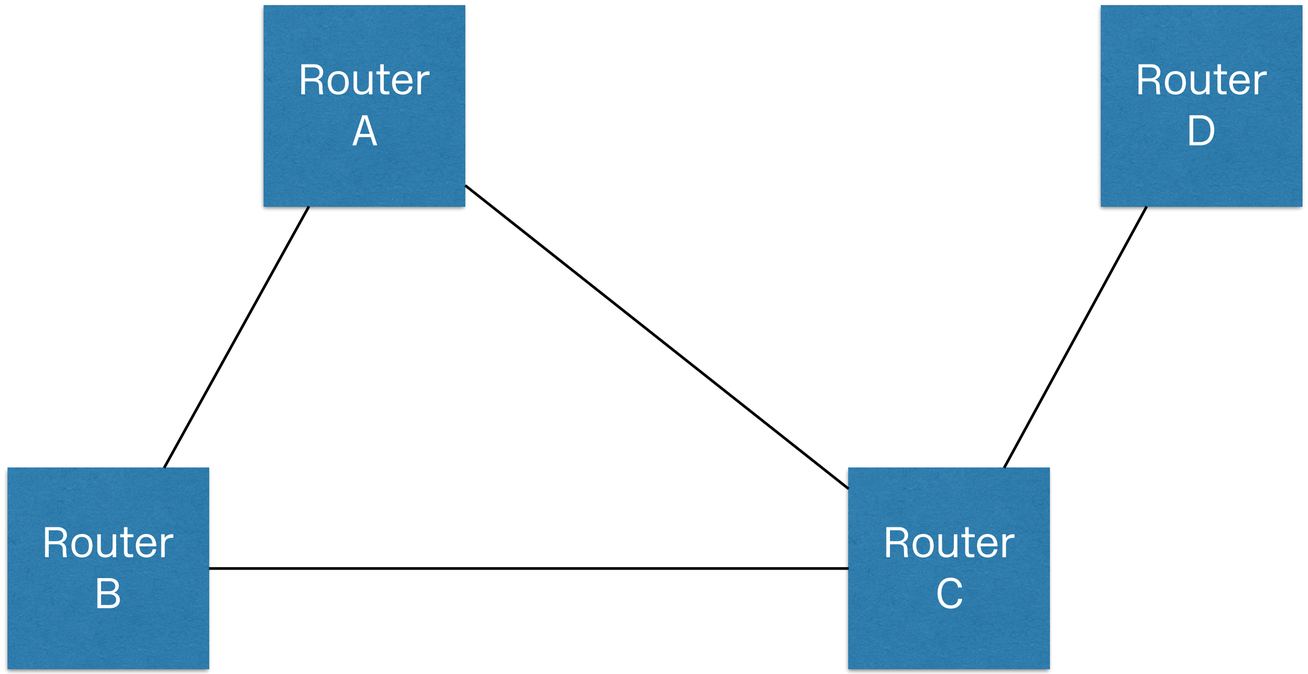
Communication abstraction

```
type GossipSender interface {
    GossipUnicast(dst Peer, m Msg) (err error)
    GossipBroadcast(m Msg)
}

type GossipReceiver interface {
    OnGossipUnicast(src Peer, m Msg) (err error)
    OnGossipBroadcast(src Peer, m Msg) (received Msgs, err error)
    OnGossip(m Msg) (delta Msgs, err error)
    DumpState() (complete Msgs)
}
```

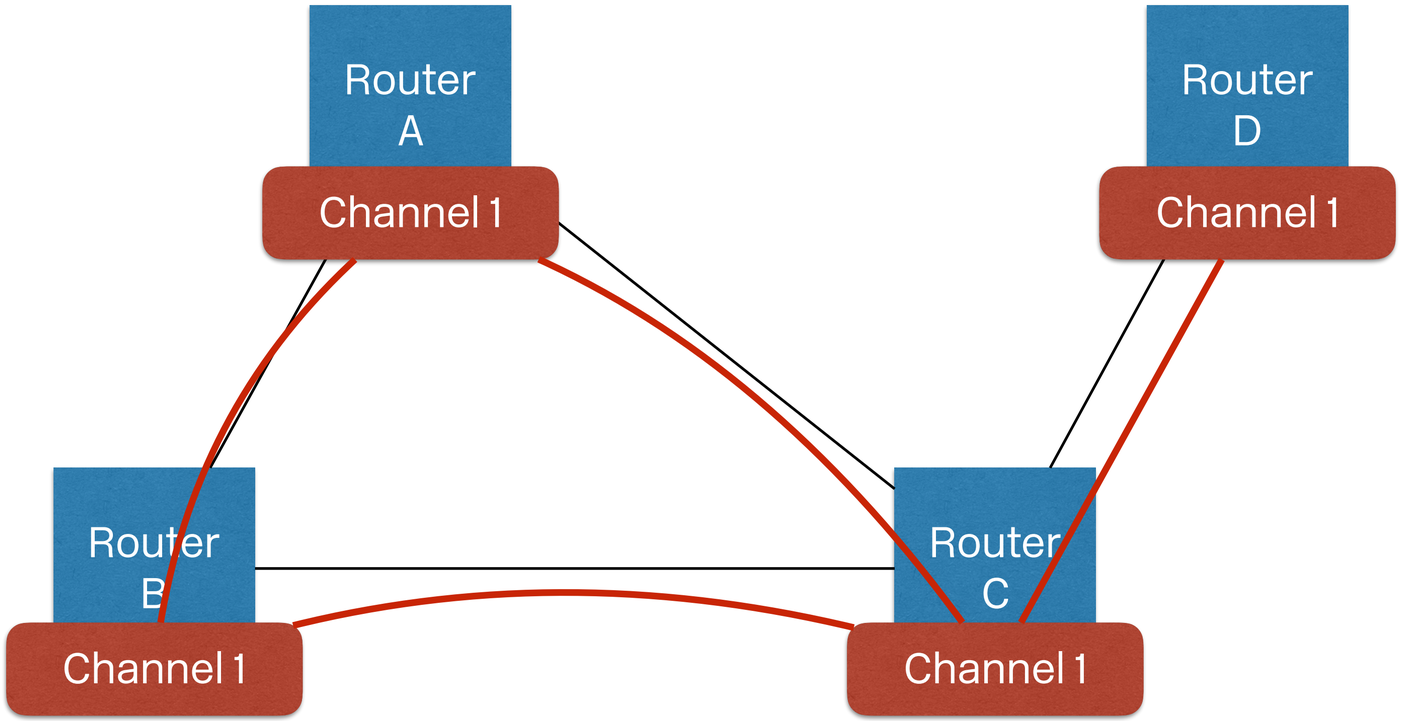
Data abstraction

```
type GossipData interface {
    Encode() Msgs
    Merge(other GossipData) (result GossipData)
}
```

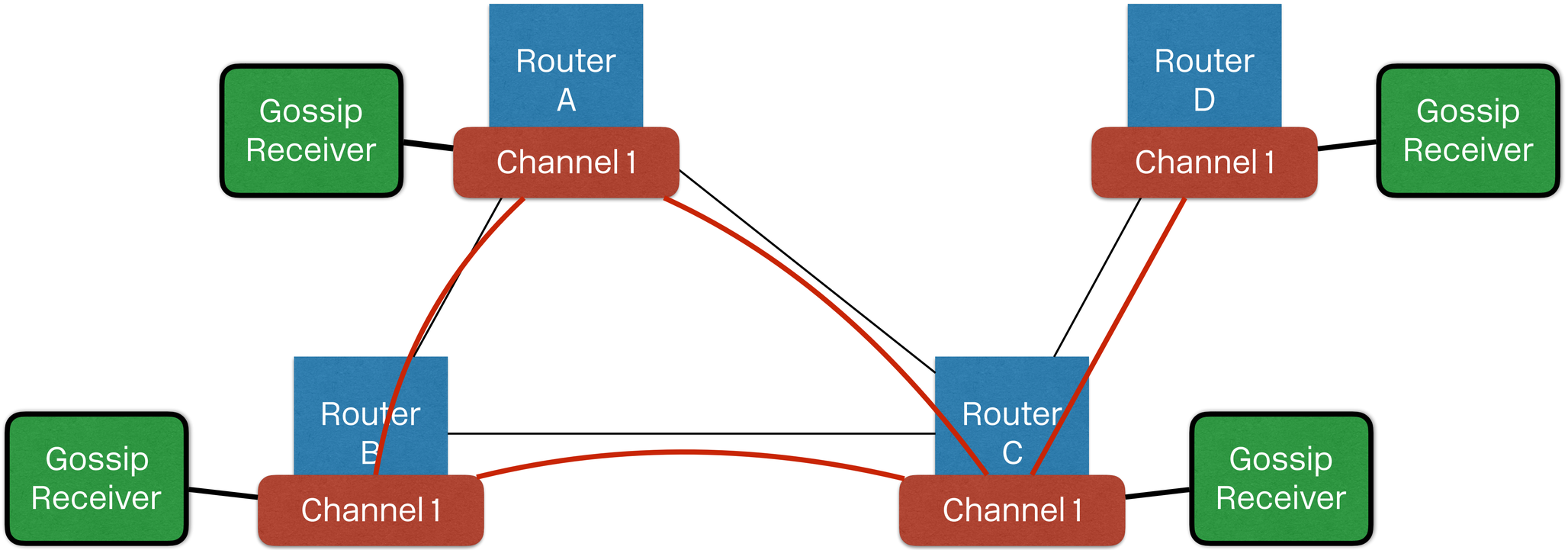
Communication component



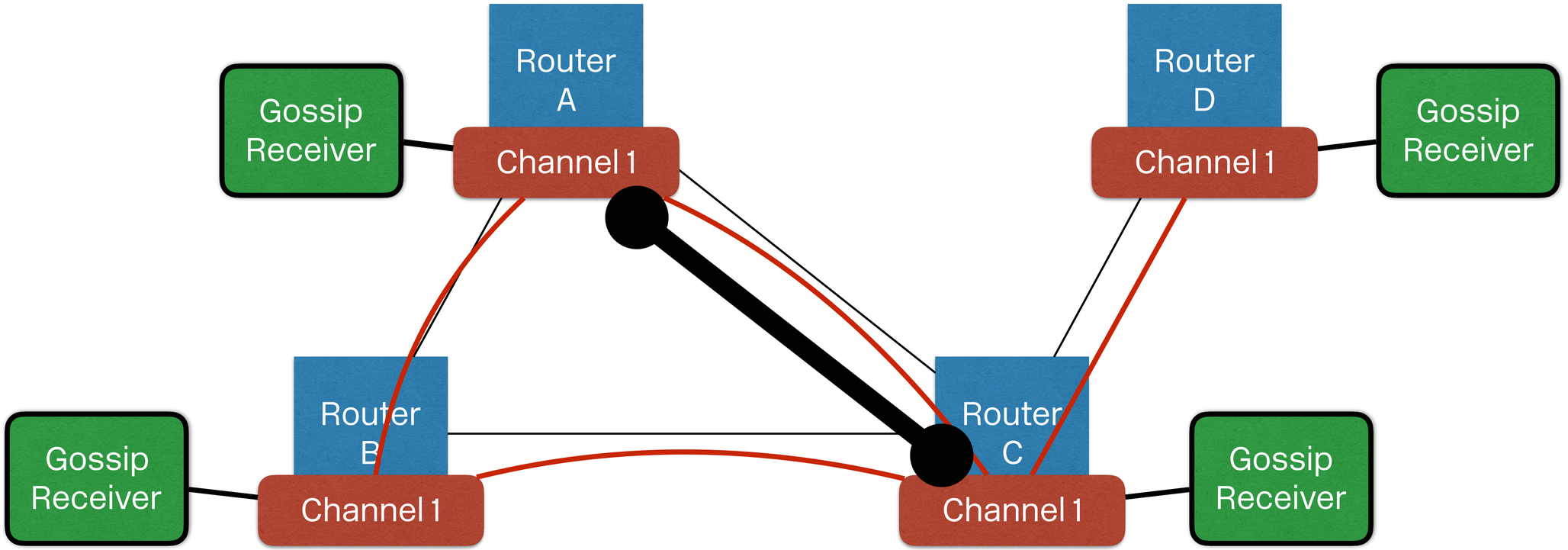
Communication component



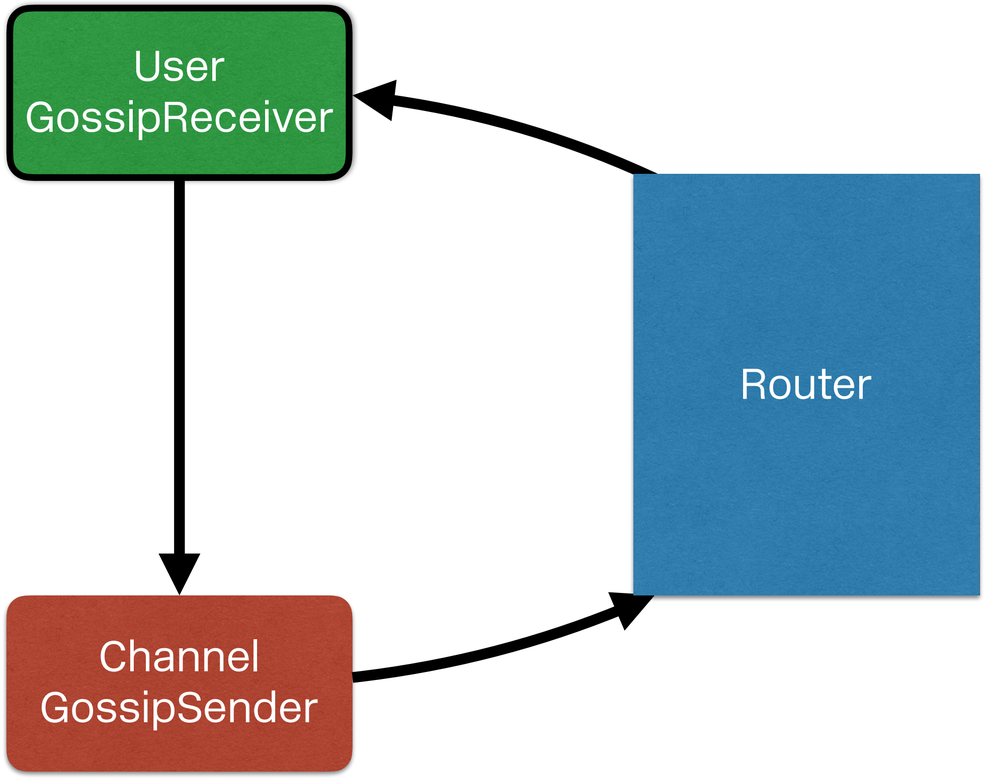
Communication component



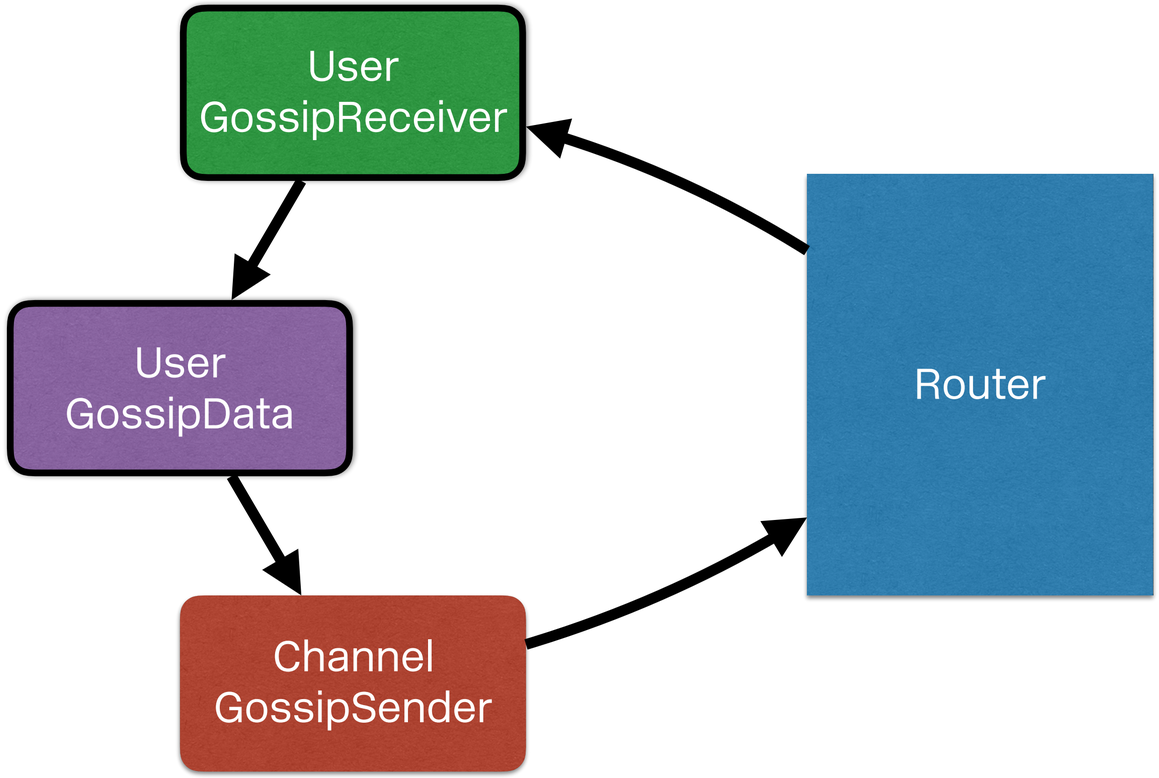
Communication component



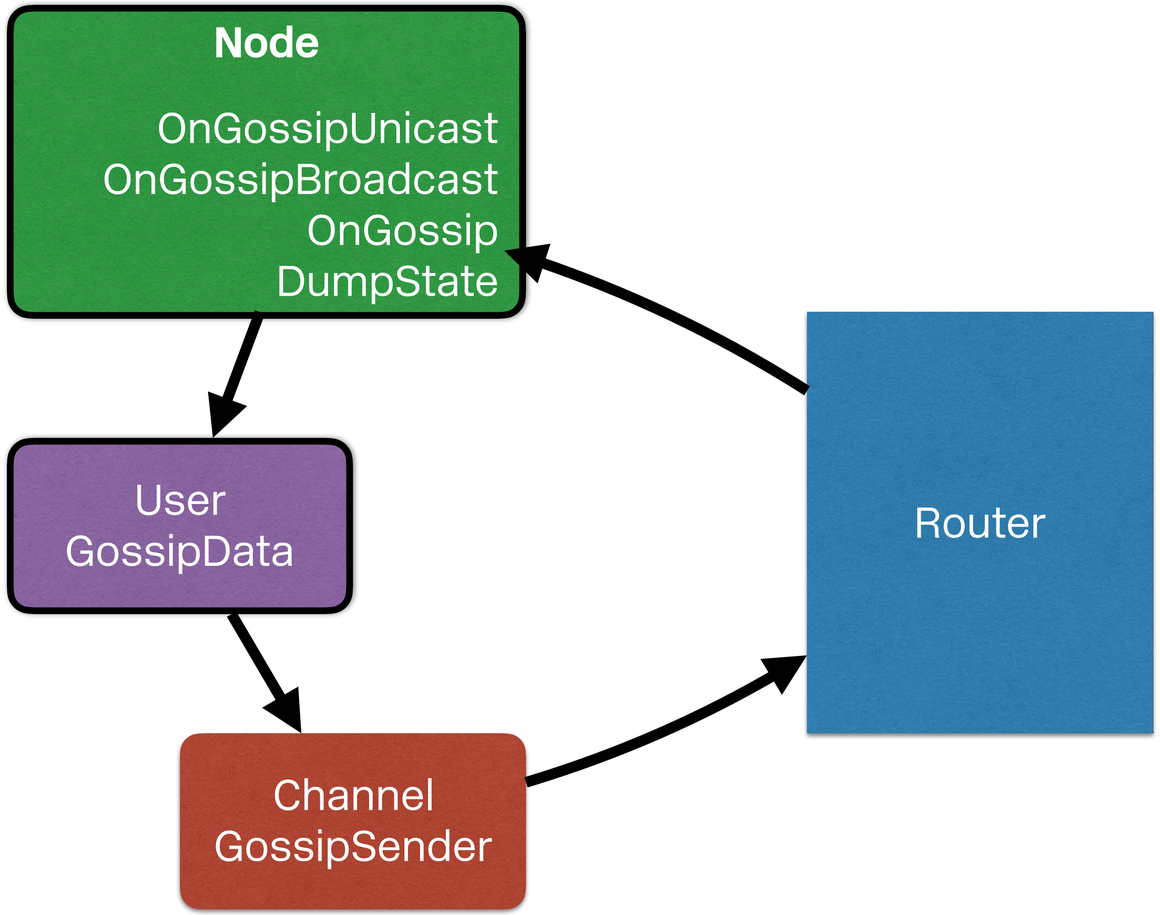
Communication component



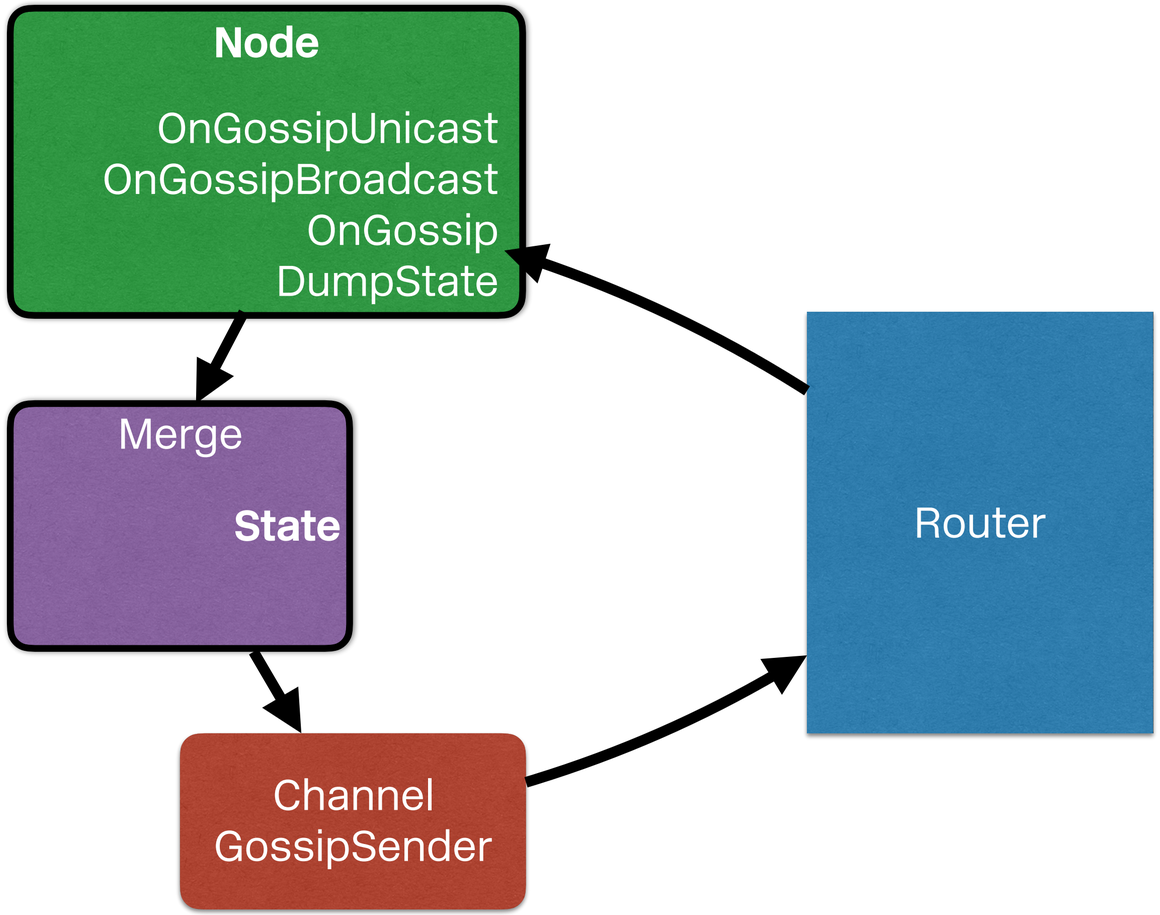
Communication component



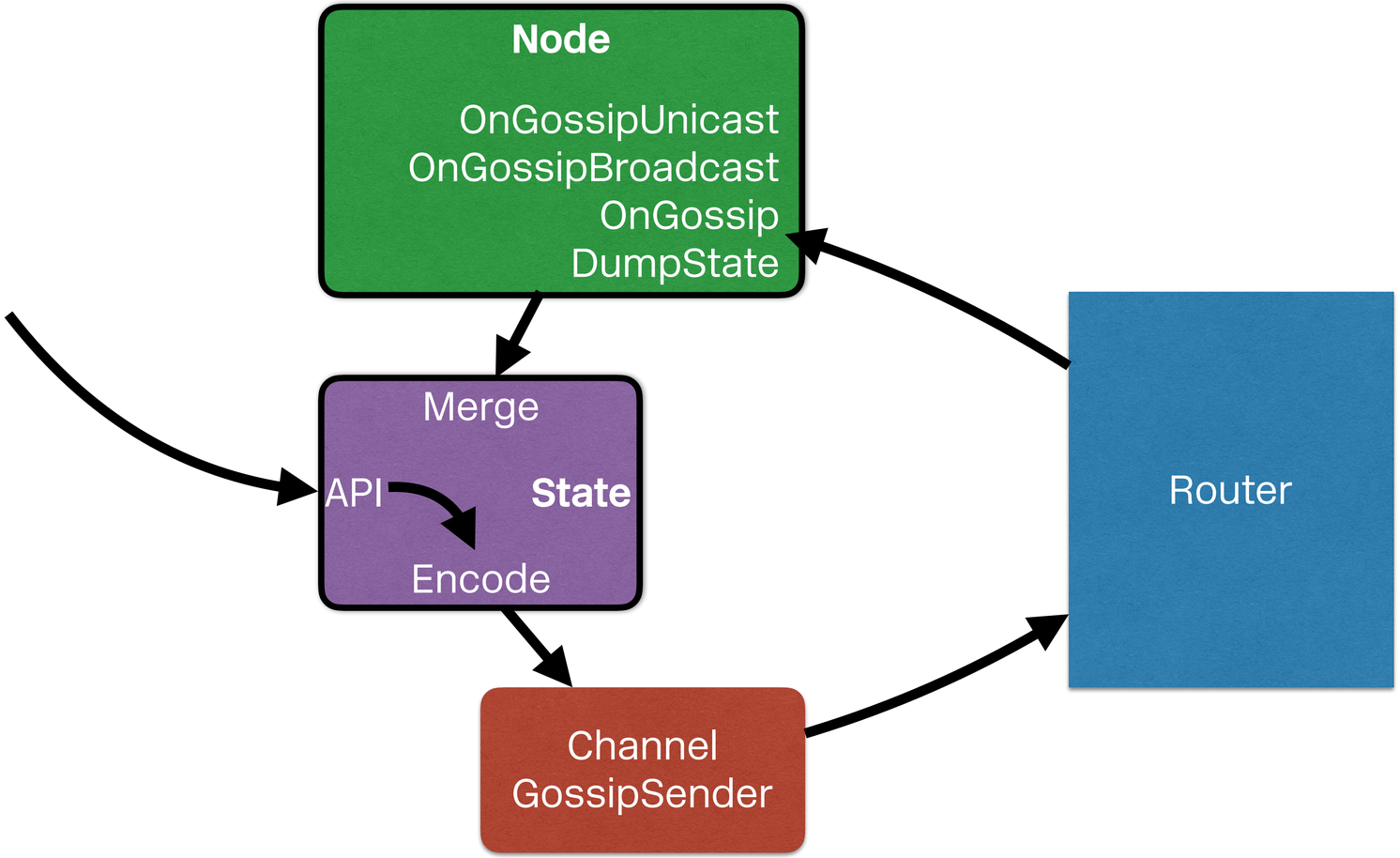
Communication component



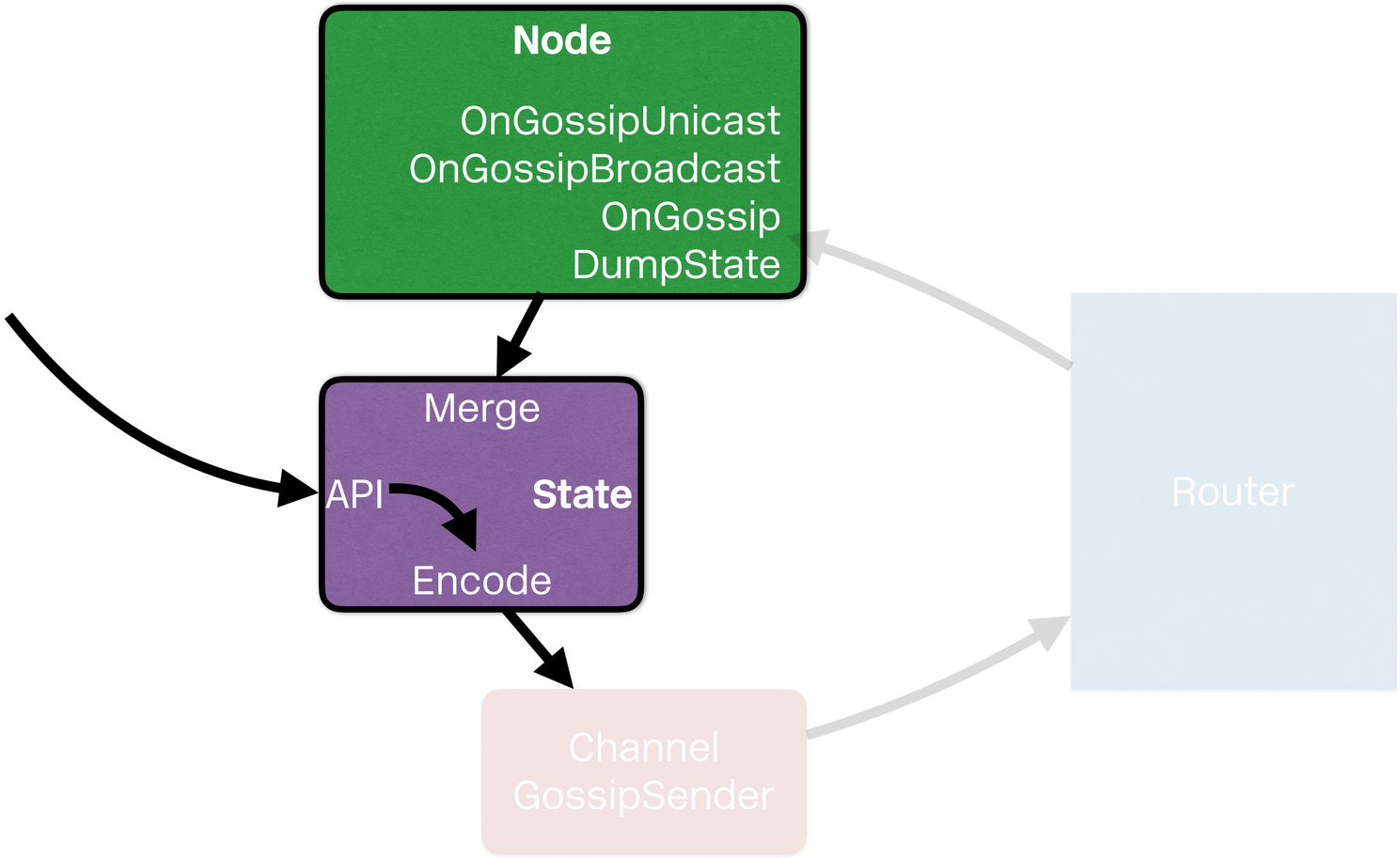
Communication component



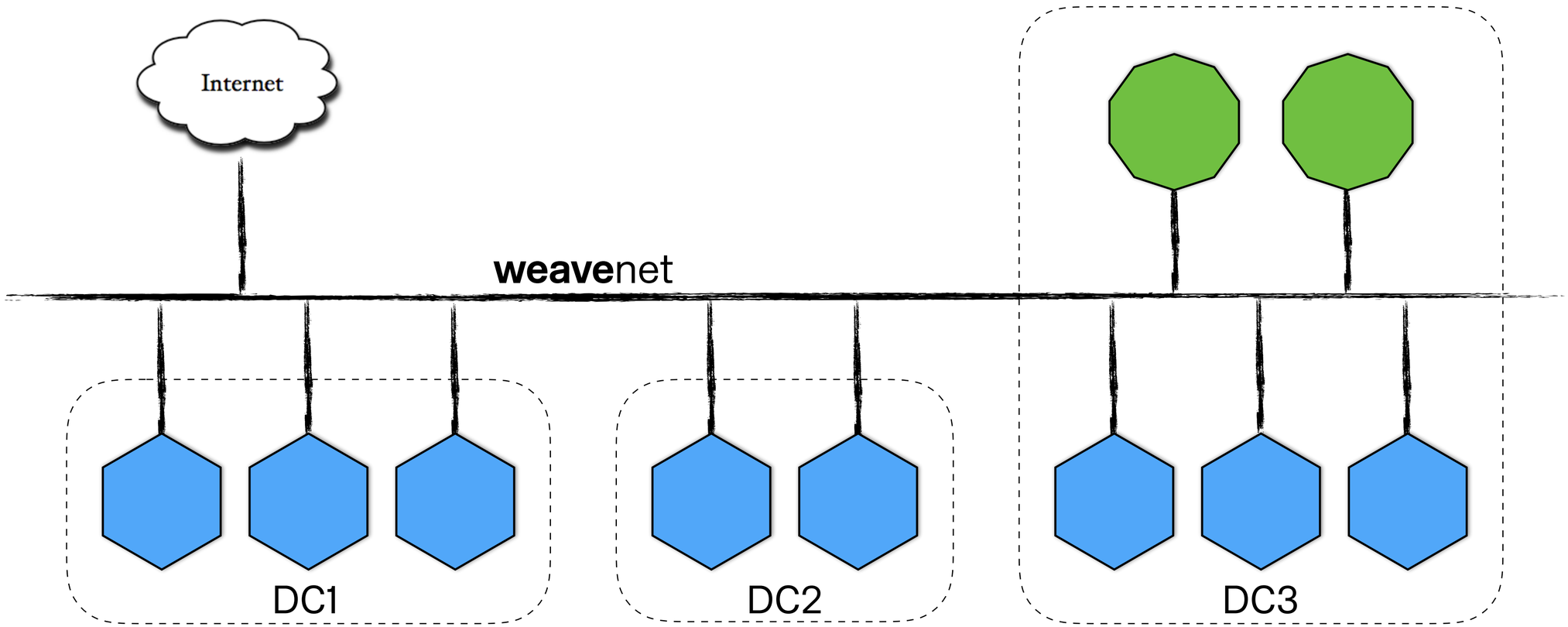
Communication component



Communication component



Practice:
weavenet


Internet
weavenet
DC1
DC2
DC3

weavenet

- Weave Net depends on Weave Mesh for…
  - Peer discovery and routing
  - IP address management
  - Service discovery and load balancing
- All of which is scalable, available, and partition-tolerant

weavedns

- Starting a named container anywhere

```
hostA$ docker run --name=foo ...
```

- Makes that name resolvable everywhere

```
containerX@hostB$ ping foo
containerY@hostC$ ping foo
```

Demo

weavedns

- Scalable
- Available
- Partition tolerant
- Eventually consistent
- Fast
- Thanks to Weave Mesh

Practice → Extension

Extension:
User data types

```
type GossipData interface {
    Encode() Msgs
    Merge(other GossipData) (result GossipData)
}
```

```
type GossipData interface {
    Encode() Msgs
    Merge(other GossipData) (result GossipData)
}

type myData struct {
    m map[Node]int
}
```

```
type GossipData interface {
    Encode() Msgs
    Merge(other GossipData) (result GossipData)
}

type myData struct {
    m map[Node]int
}

func (d myData) Encode() Msgs {
    return Msgs{json.Encode(d.m)} // single msg, complete state
}
```

```
func (d myData) Merge(other GossipData) (result GossipData) {
    theirs = other.(myData)

    for node, val := range theirs.m {
        if val > d.m[node] {
            d.m[node] = val
        }
    }

    return d
}
```

```
type GossipReceiver interface {
    OnGossipUnicast(src Peer, m Msg) (err error)
    OnGossipBroadcast(src Peer, m Msg) (received Msgs, err error)
    OnGossip(m Msg) (delta Msgs, err error)
    DumpState() (complete Msgs)
}
```

```
type GossipReceiver interface {
    OnGossipUnicast(src Peer, m Msg) (err error)
    OnGossipBroadcast(src Peer, m Msg) (received Msgs, err error)
    OnGossip(m Msg) (delta Msgs, err error)
    DumpState() (complete Msgs)
}

func OnGossipUnicast(src Peer, m Msg) (err error) {
    update := json.UnmarshalFrom(m)
    state.Merge(update)
    return nil
}
```

```
func OnGossipBroadcast(src Peer, m Msg) (received Msgs, err error) {
    update := json.UnmarshalFrom(m)
    received := state.MergeReceived(update)
    return received, nil
}
```

```
func OnGossipBroadcast(src Peer, m Msg) (received Msgs, err error) {
    update := json.UnmarshalFrom(m)
    received := state.MergeReceived(update)
    return received, nil
}

func OnGossip(m Msg) (delta Msgs, err error) {
    update := json.UnmarshalFrom(m)
    delta := state.MergeDelta(update)
    return delta, err
}
```

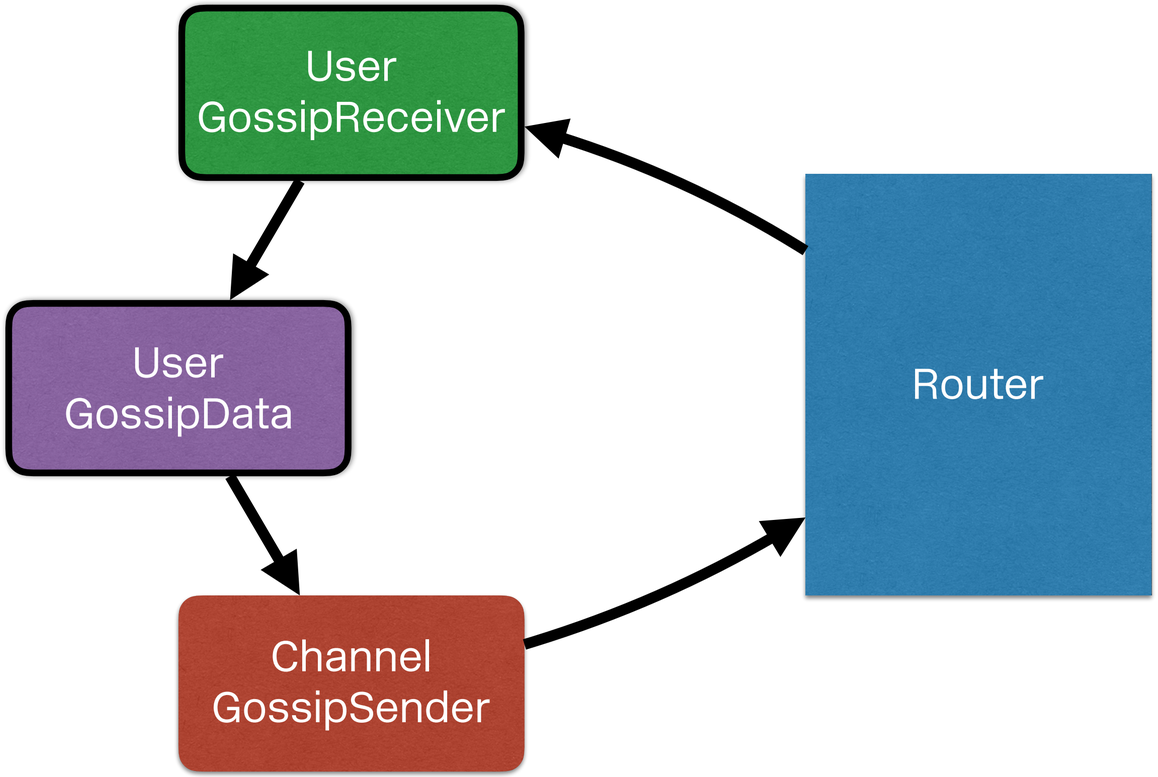
```
func OnGossipBroadcast(src Peer, m Msg) (received Msgs, err error) {
    update := json.UnmarshalFrom(m)
    received := state.MergeReceived(update)
    return received, nil
}

func OnGossip(m Msg) (delta Msgs, err error) {
    update := json.UnmarshalFrom(m)
    delta := state.MergeDelta(update)
    return delta, err
}

func DumpState() (complete Msgs) {
    return json.Marshal(state)
}
```

```
type GossipSender interface {
    GossipUnicast(dst Peer, m Msg) (err error)
    GossipBroadcast(m Msg)
}
```

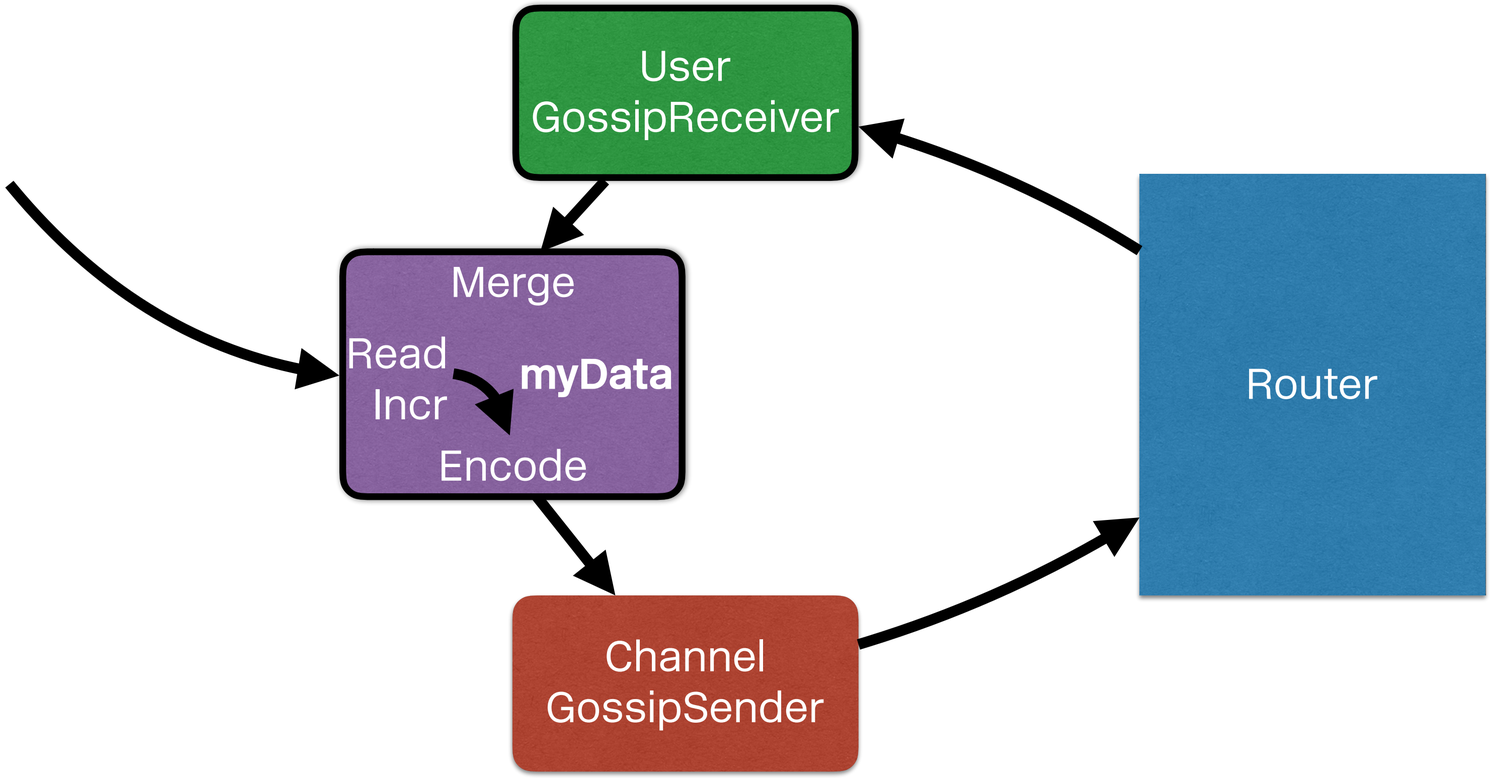
```
type GossipSender interface {
    GossipUnicast(dst Peer, m Msg) (err error)
    GossipBroadcast(m Msg)
}

func (d myData) Increment() {
    d.m[self]++                // make local state change
    update := d.Encode()       // create state-based CRDT update
    GossipBroadcast(update) // eventual consistency
}
```

```
type GossipSender interface {
    GossipUnicast(dst Peer, m Msg) (err error)
    GossipBroadcast(m Msg)
}

func (d myData) Increment() {
    d.m[self]++                // make local state change
    update := d.Encode()       // create state-based CRDT update
    GossipBroadcast(update) // eventual consistency
}

func (d myData) Read() (result int) {
    for _, val := range d.m {
        result += val
    }
    return result
}
```

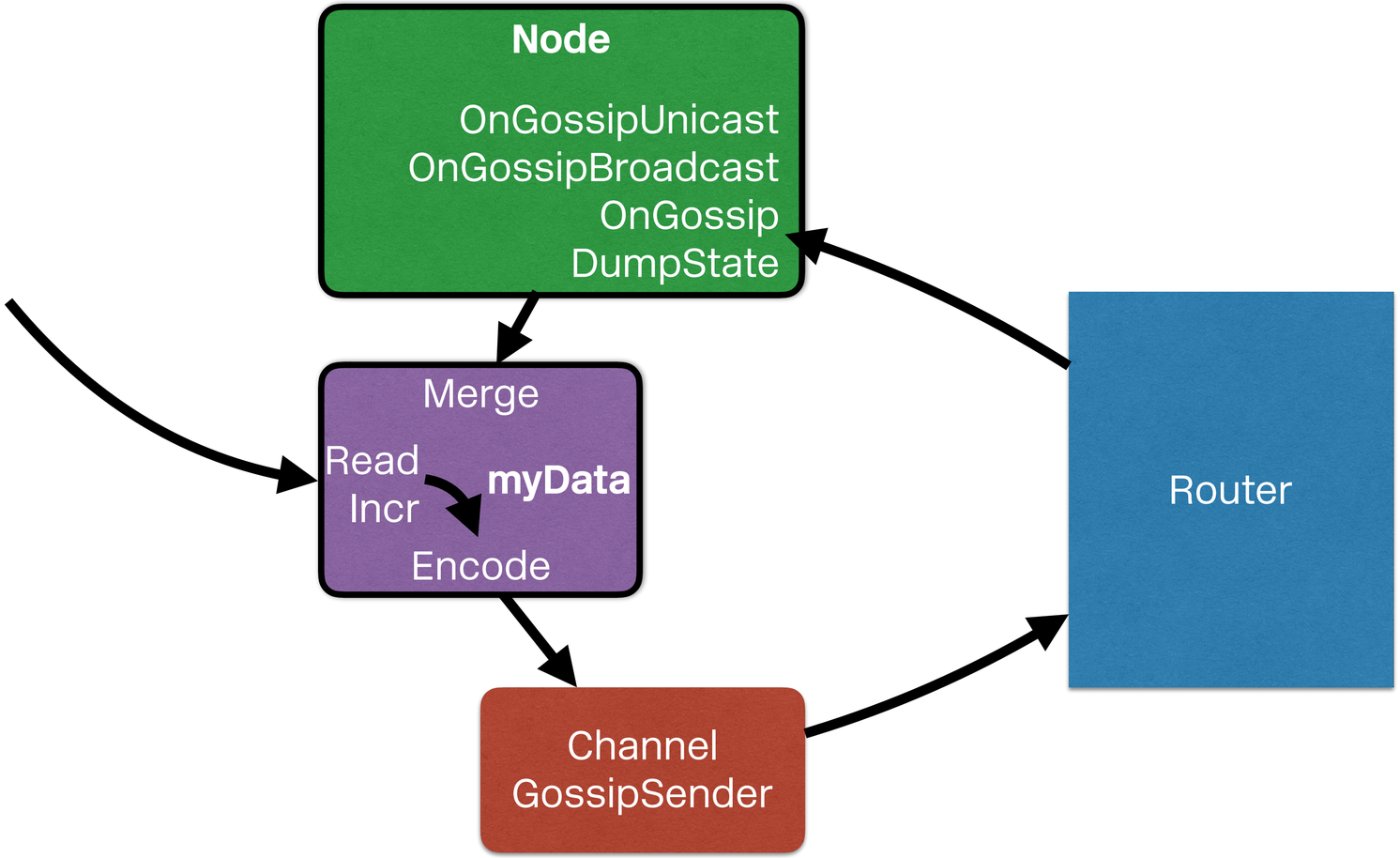
Component diagram



Component diagram



Component diagram



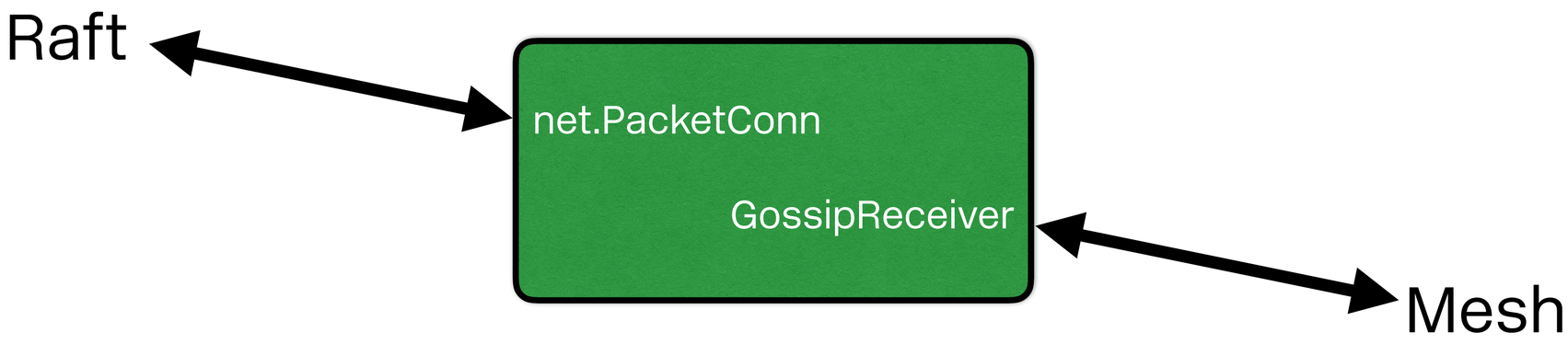
Extension:
Strong consensus

Intermediary:
UDP — net.PacketConn

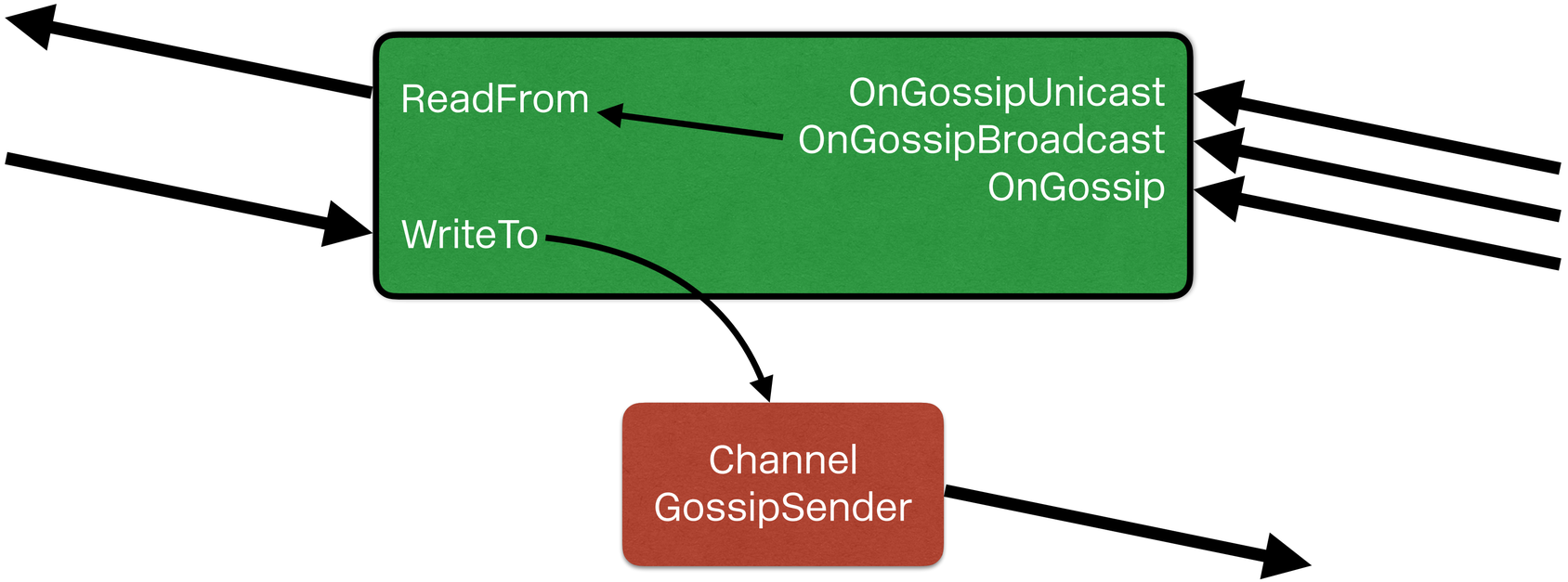
```
type PacketConn interface {
    ReadFrom(b []byte) (n int, addr Addr, err error)
    WriteTo(b []byte, addr Addr) (n int, err error)
    Close() error
    LocalAddr() Addr
    SetDeadline(t time.Time) error
    SetReadDeadline(t time.Time) error
    SetWriteDeadline(t time.Time) error
}
```

```
type PacketConn interface {
    ReadFrom(b []byte) (n int, addr Addr, err error)
    WriteTo(b []byte, addr Addr) (n int, err error)
    Close() error
    LocalAddr() Addr
    SetDeadline(t time.Time) error
    SetReadDeadline(t time.Time) error
    SetWriteDeadline(t time.Time) error
}
```

Component diagram



Component diagram



```
// Called from network driver to recv a packet.
func (p *Peer) ReadFrom(p []byte) (..., remote Addr, ...) {
    pkt := <-p.recv           // from mesh network
    copy(p, pkt)              // copy data into p
    return ..., pkt.Src, ... // pkt.Src addr is remote
}
```

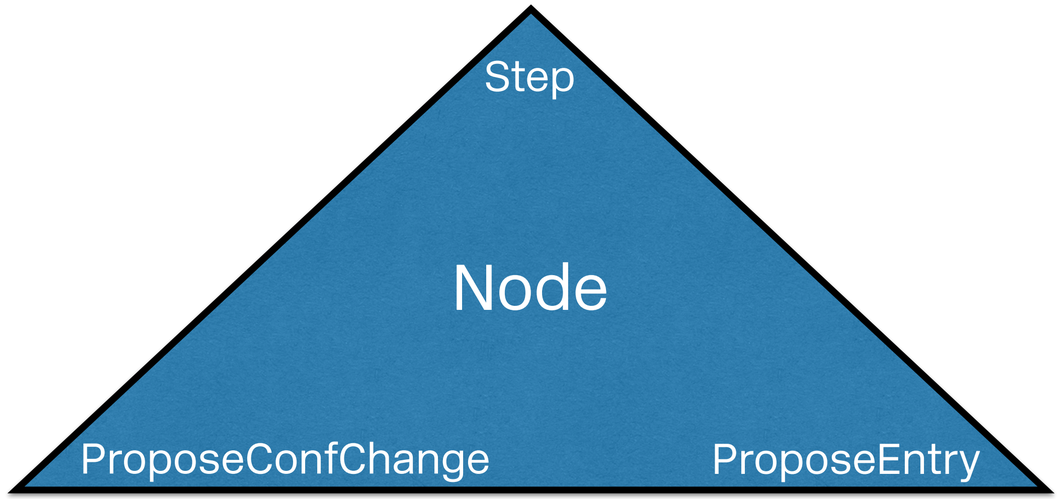
```
// Called from network driver to recv a packet.
func (p *Peer) ReadFrom(p []byte) (..., remote Addr, ...) {
    pkt := <-p.recv           // from mesh network
    copy(p, pkt)              // copy data into p
    return ..., pkt.Src, ... // pkt.Src addr is remote
}

// Called from network driver to send a packet.
func (p *Peer) WriteTo(b []byte, dst Addr) ... {
    pkt := Pkt{Src: p.Self, Dst: dst, Buf: b}
    p.sender.GossipUnicast(dst, pkt.Encode())
}
```

```
func (p *Peer) OnGossipBroadcast(src Peer, m Msg) ... {
    p.recv <- makePkt(m)
}

func (p *Peer) OnGossipUnicast(src Peer, m Msg) ... {
    p.recv <- makePkt(m)
}

func (p *Peer) OnGossip(m Msg) ... {
    p.recv <- makePkt(m)
}

func (p *Peer) DumpState() ... {
    return nil // stateless
}
```

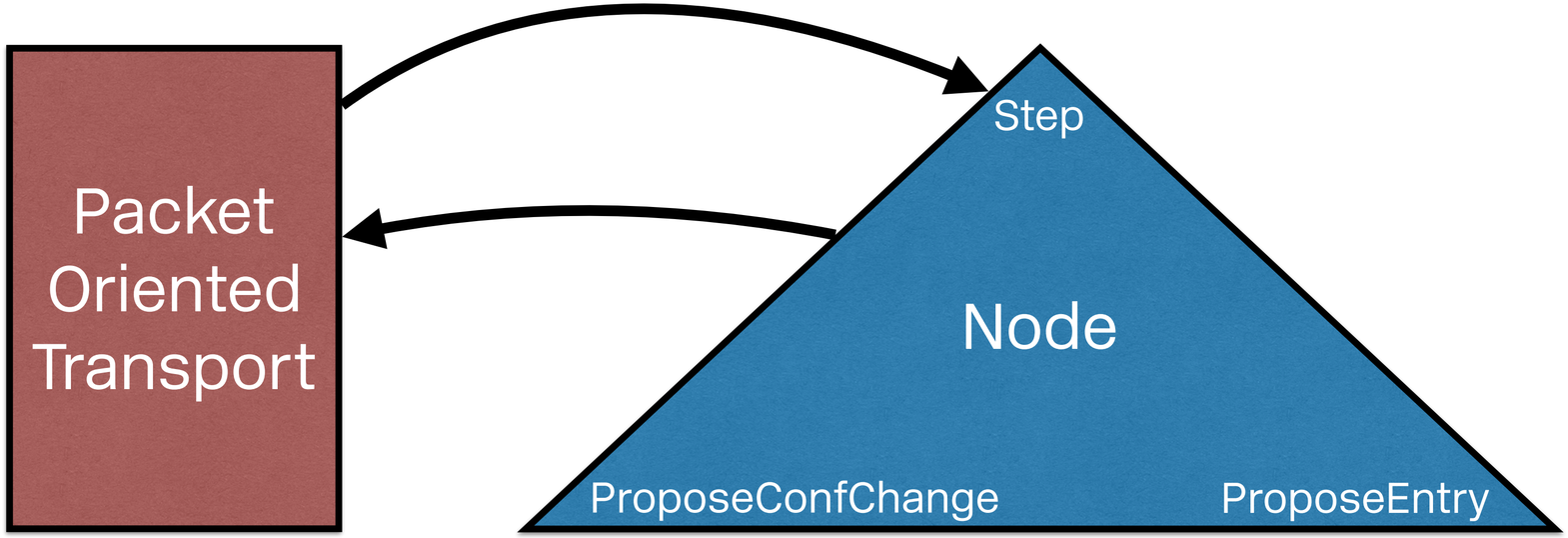
Weak Consistency
+
Adapter
=
Strong Consistency

Mesh
+
MeshConn
=
Raft

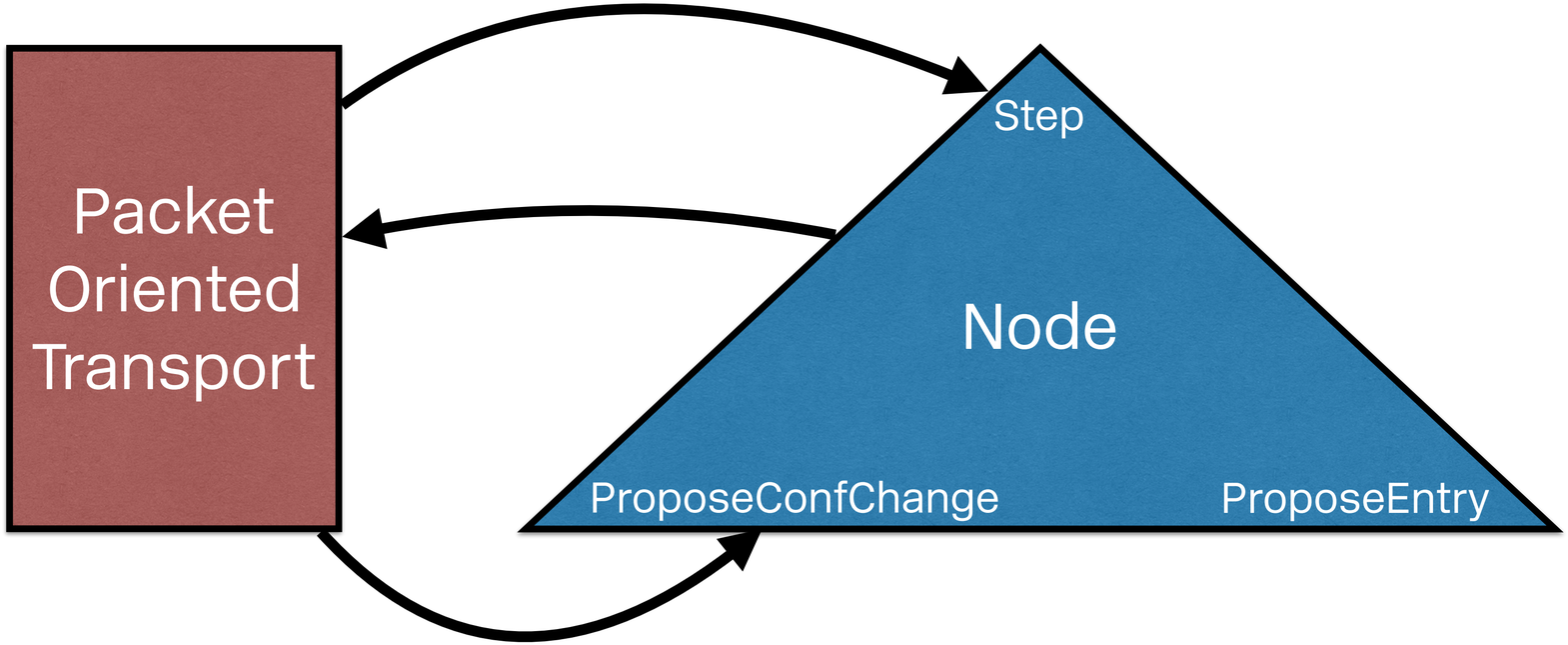
etcd-flavored Raft



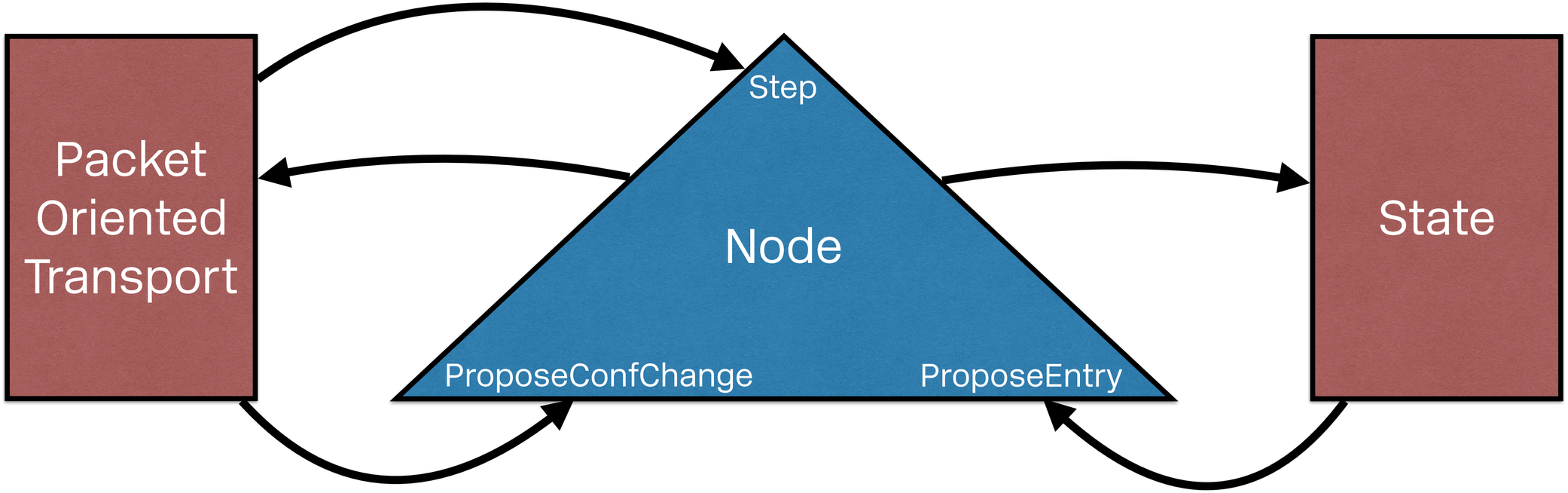
etcd-flavored Raft



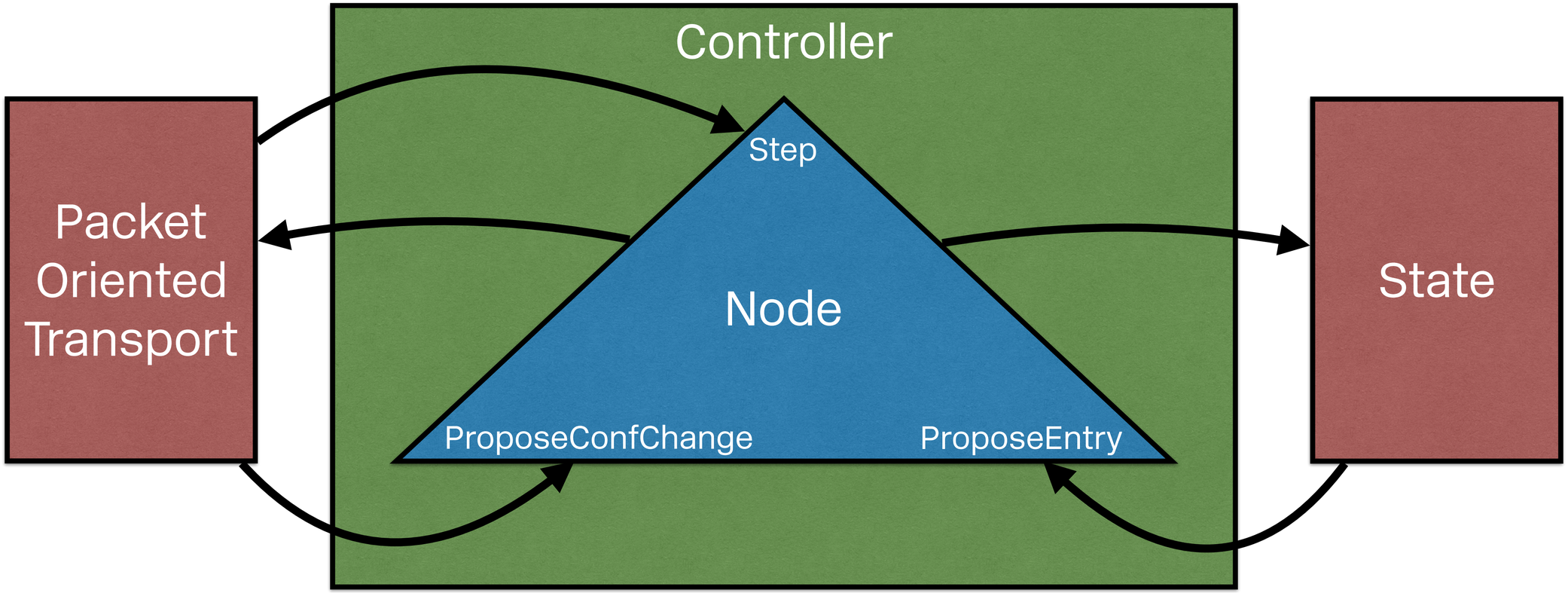
etcd-flavored Raft



etcd-flavored Raft



etcd-flavored Raft

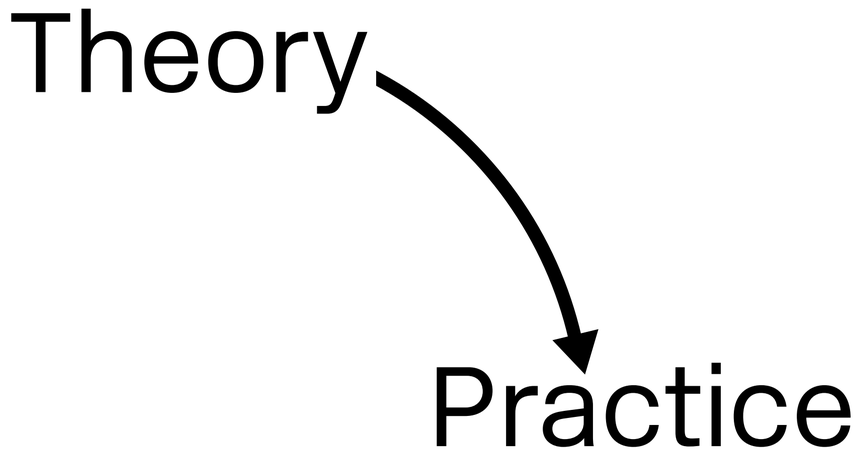


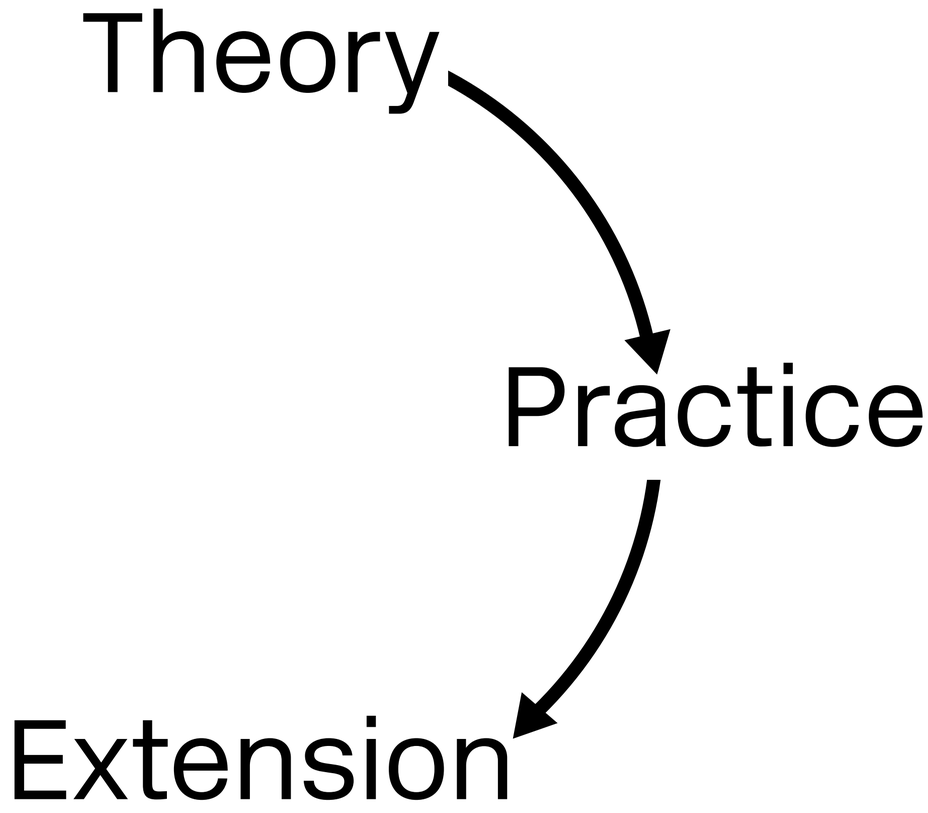
etcd-flavored Raft

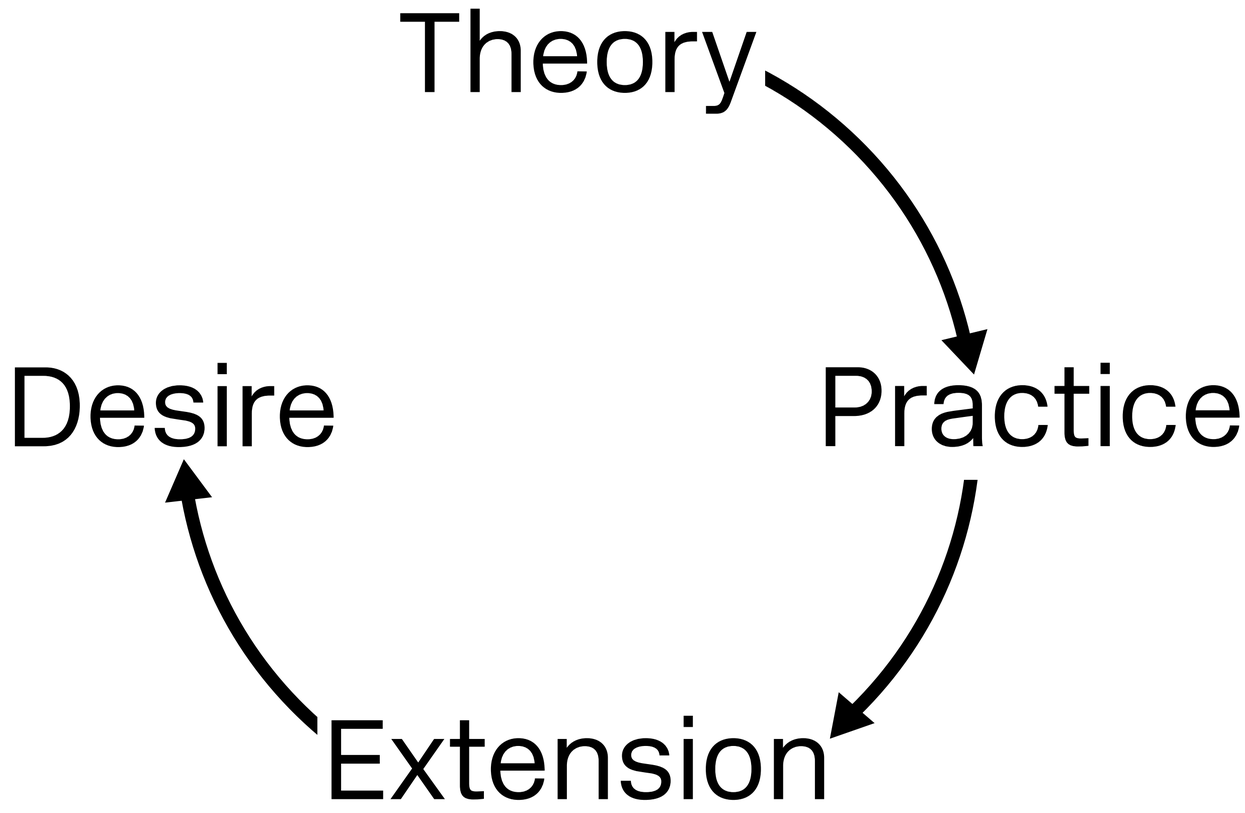
- Weakly-consistent substrate — Mesh, gossip
- Adapter — MeshConn, UDP
- Strongly-consistent semantics — etcd, Raft

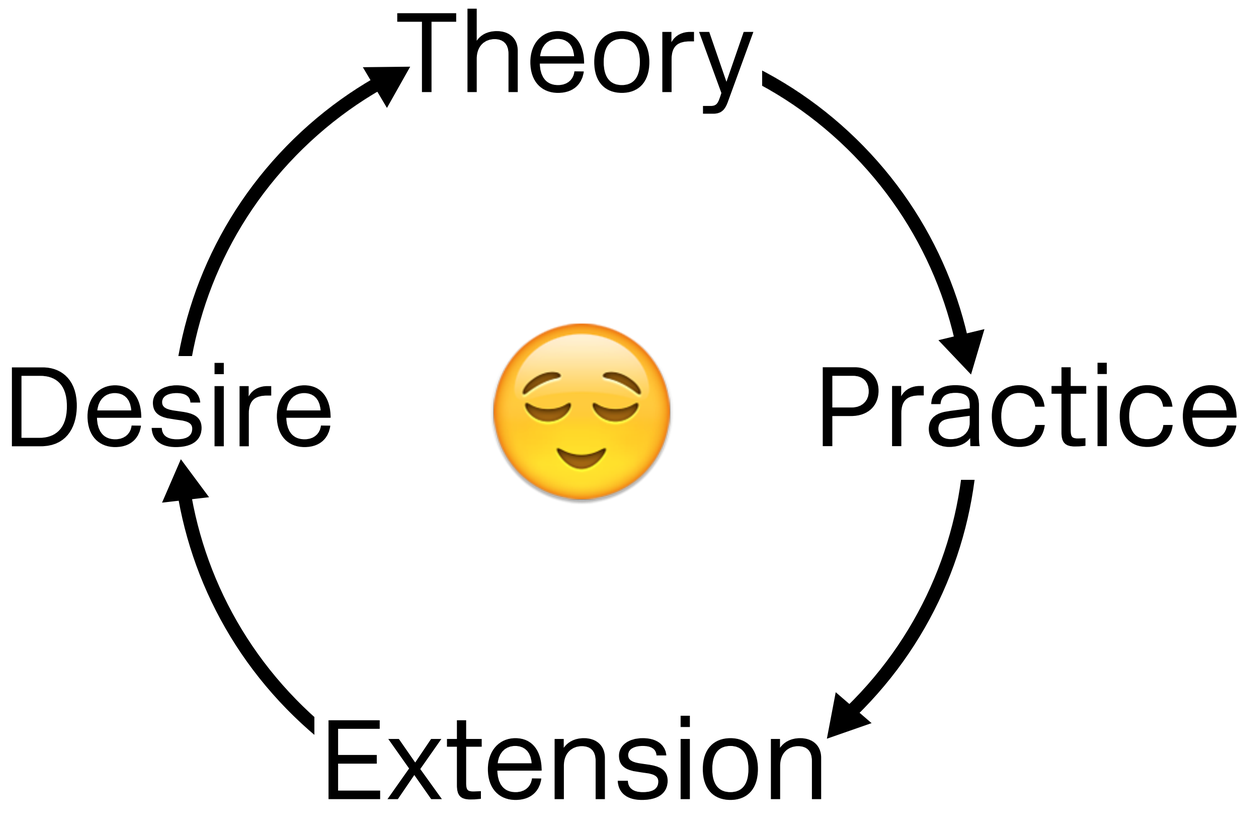
Extension → Desire?

Theory


Theory
Practice


Theory
Practice
Extension


Theory
Practice
Extension
Desire


Theory
Practice
Extension
Desire

Thank you!

Matthias Radestock, CTO
Peter Bourgon, Typist

https://weave.works

weavescope

scope.weave.works