Merkle Search Trees:
Efficient State-based CRDTs in Open Networks

Alex Auvolat, François Taïani

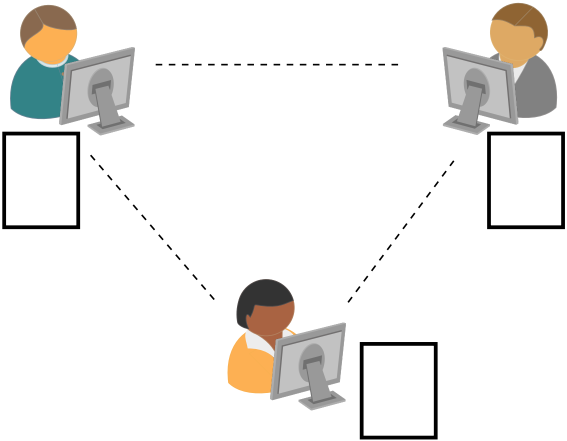
Team WIDE
Univ. Rennes, Inria, CNRS, IRISA
alex.auvolat@inria.fr

October 3, 2019
SRDS 2019, Lyon, France

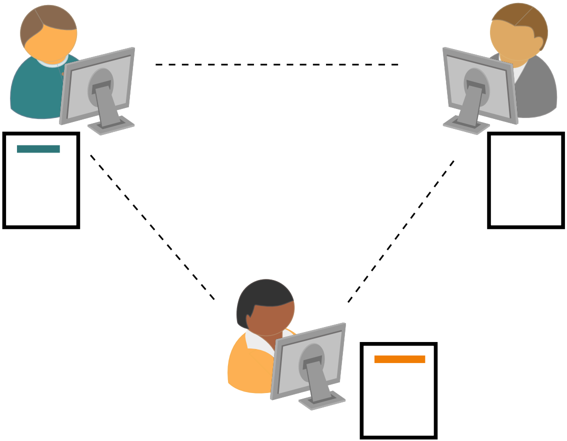
Context

- We consider the problem of **efficient state reconciliation**
- Applicable to CRDTs that implement **eventual consistency**

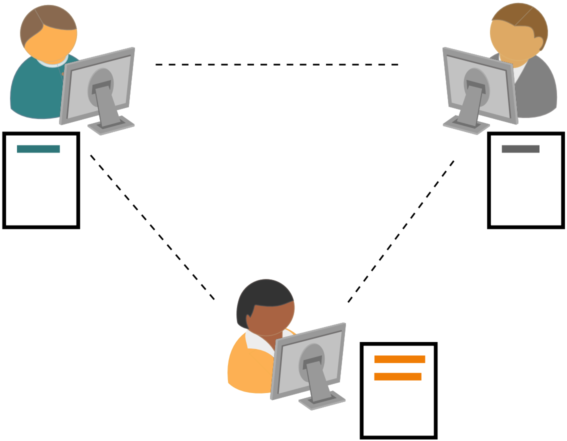
Example



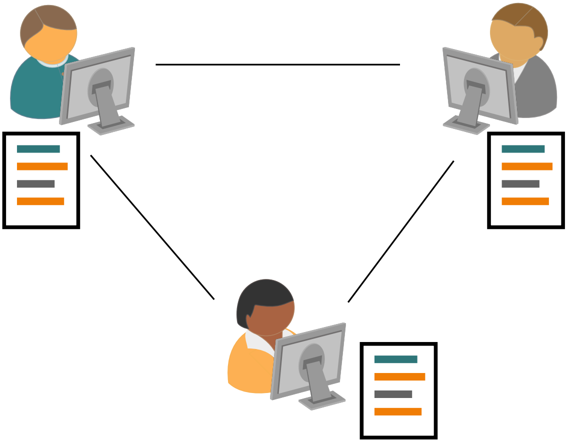
Example



Example



Example



Our Setting

- We study the simple case of storing and broadcasting events (a distributed **event store**)
- It is a simple CRDT: a **grow-only set**
- The principles of this paper can be applied to **other CRDTs**

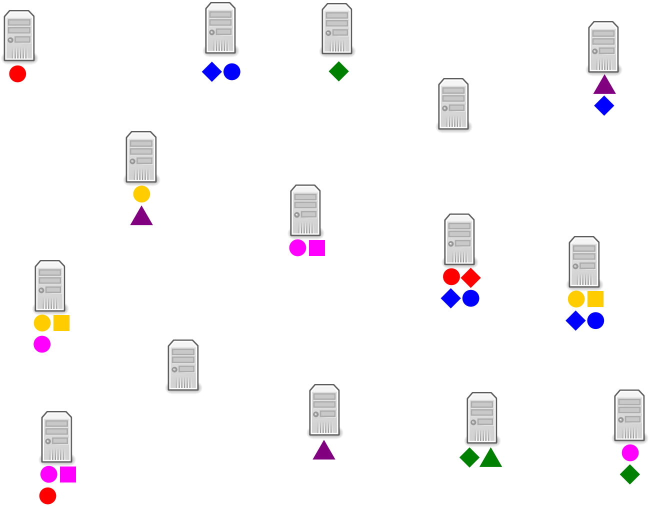
Context: Large Networks

- We consider a **large network with many nodes**
- Nodes may **join, leave and re-join** at any time
- Typically the case of **large-scale geo-replicated systems**

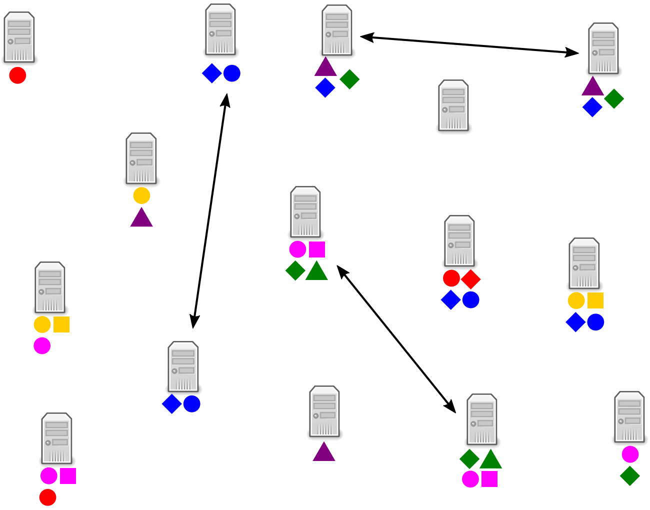
Gossip in Very Large Networks

- Nodes communicate with **random other nodes** using a **gossip protocol**

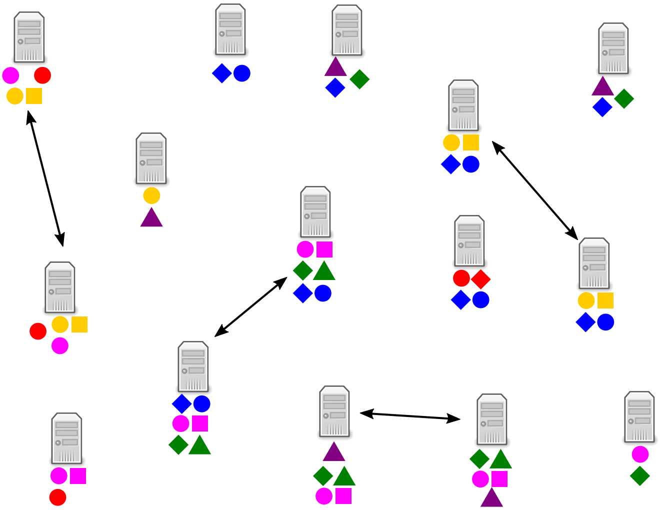
Gossip in Very Large Networks



Gossip in Very Large Networks



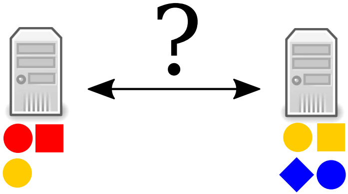
Gossip in Very Large Networks



Gossip in Very Large Networks

- Nodes communicate with **random other nodes** using a **gossip protocol**
- Nodes do not keep track of other nodes' states (too expensive)

The Problem: How to Minimize Cost of Exchange?



Nodes must be able to exchange missing events with zero a-priori knowledge on each other's state, with **minimal data exchange**.

This is known as **anti-entropy**.

SotA: Scuttlebutt Anti-entropy

Approach 1: **vector clocks**

- Known as **Scuttlebutt anti-entropy** [1]

  [1] Van Renesee et al., *Efficient Reconciliation and Flow Control for Anti-Entropy Protocols*, 2008

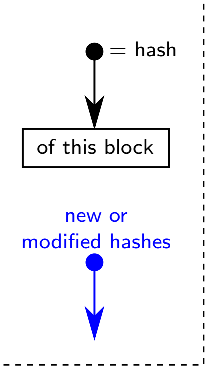
- The clock **grows linearly with the number of nodes**
  $\rightarrow$ inefficient in large networks
  $\rightarrow$ metadata for disconnected nodes cannot be discarded
  $\rightarrow$ fundamentally unsuited to networks with churn

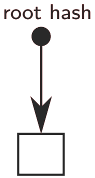
SotA: Merkle Tree Anti-Entropy

Approach 2: **Merkle trees**

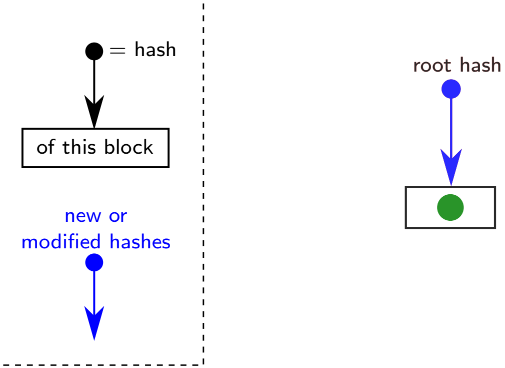
- Encode the set of events in a **Merkle tree**
- Enables **efficient comparison** between two remote sites: equal subtrees have equal hashes and can be skipped entirely
- Only **paths to changed nodes** need to be exchanged

What Is a Merkle Tree Anyways?

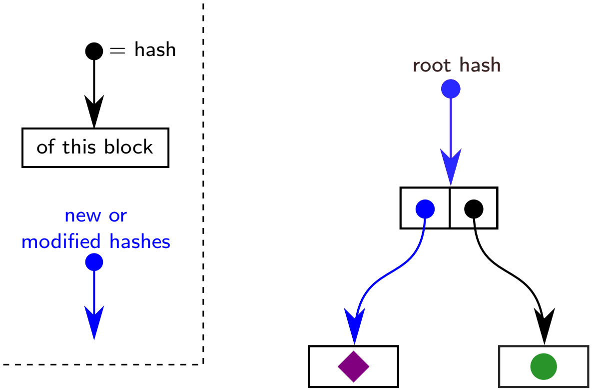




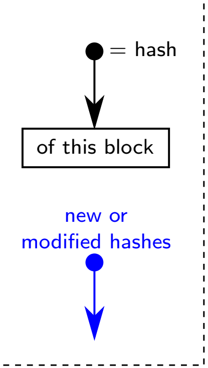
What Is a Merkle Tree Anyways?

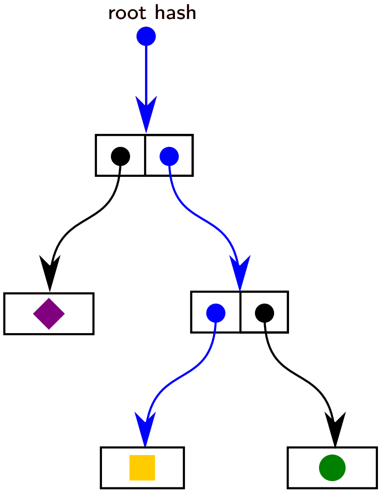


What Is a Merkle Tree Anyways?

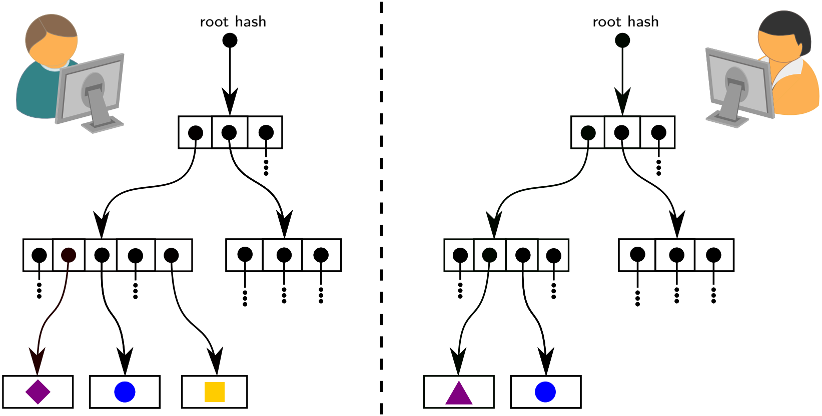


What Is a Merkle Tree Anyways?

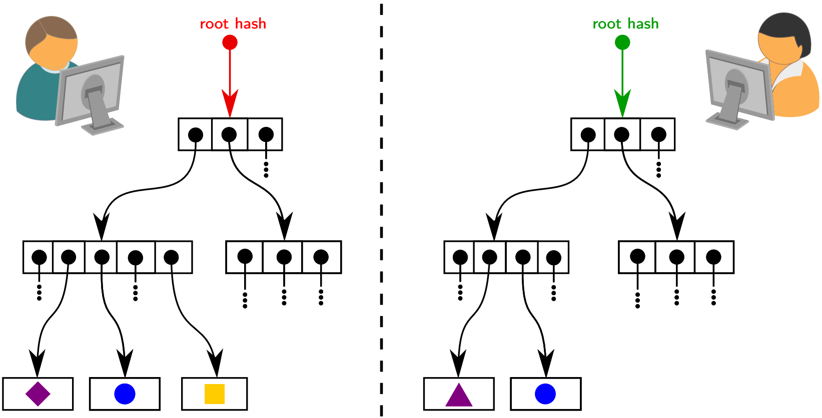




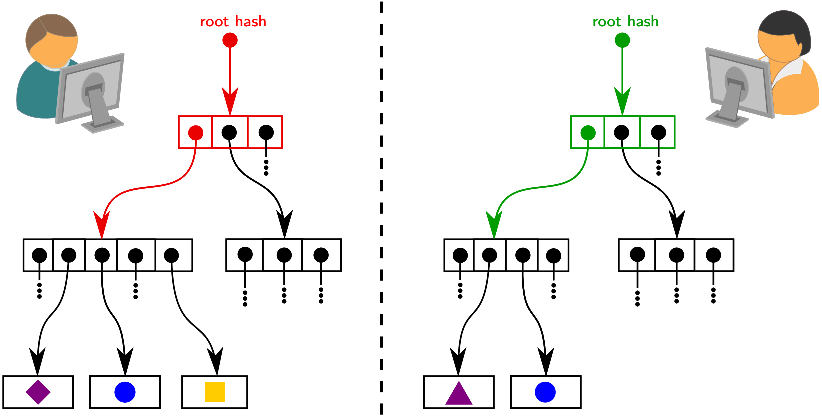
Merkle Tree Remote Comparison



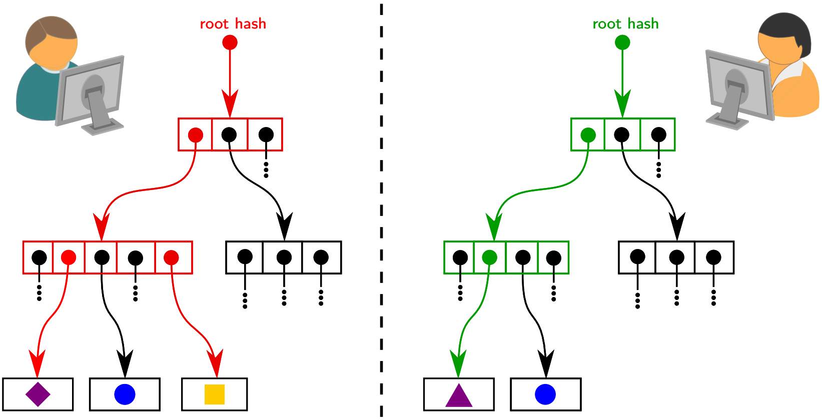
Merkle Tree Remote Comparison



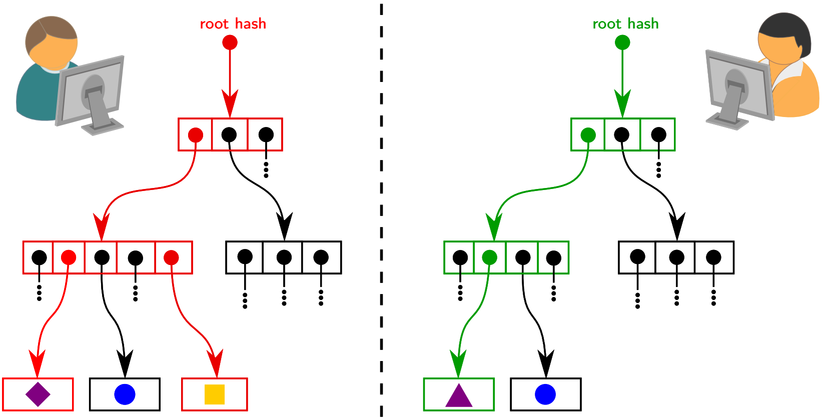
Merkle Tree Remote Comparison



Merkle Tree Remote Comparison



Merkle Tree Remote Comparison



Kinds of Merkle Trees: State of the Art

- **Standard Merkle trees:** balanced binary tree

  keys = integers from 0 to n
  $\rightarrow$ too restrictive for our use case

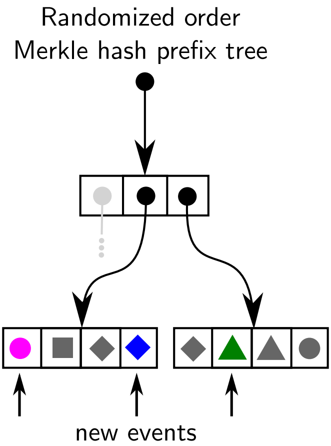
- **Merkle hash prefix trees:**
  Keys can be any values in any space with a hash function

  Algorithm: hash the keys and build a prefix tree
  $\rightarrow$ uses randomization to generate a balanced tree

Our Contribution: Merkle Search Trees

- **Issue:** the new events end up in *different branches*,
  lots of data needs to be exchanged for intermediate nodes

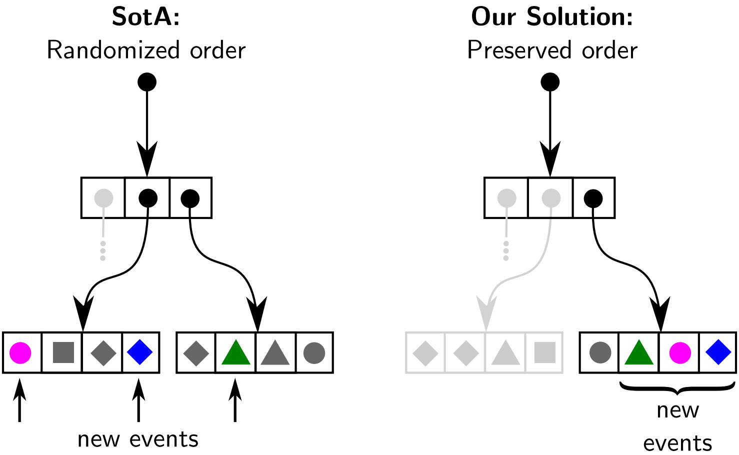
Our Contribution: Merkle Search Trees



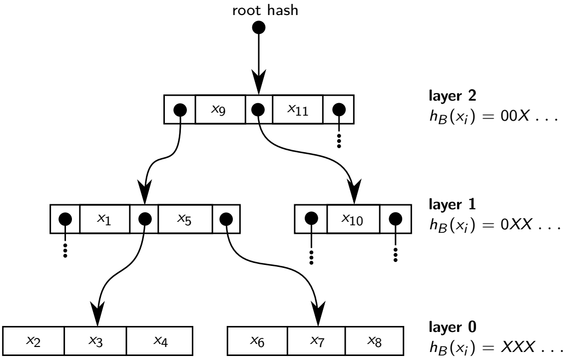
Our Contribution: Merkle Search Trees

- **Issue:** the new events end up in *different branches*, lots of data needs to be exchanged for intermediate nodes
- **Our contribution:** a tree structure that *preserves order*
- We order events by their creation timestamp
  $\rightarrow$ new events end up close together

Our Contribution: Merkle Search Trees



Our Contribution: Merkle Search Trees



$x_1 < x_2 < x_3 < x_4 < x_5 < x_6 < x_7 < x_8 < x_9 < x_{10} < x_{11}$
h_B = hash written in base B
X = any digit except 0

Use randomization to generate a **balanced tree structure**, but preserve the **order of items**

Theoretical Comparison: Merkle Trees vs. Scuttlebutt

Theoretical comparison

Anti-entropy algorithm	**Dissemination time**	**Traffic per anti-entropy round**
Scuttlebutt (vector clocks)	$2\lambda \log m$	$O(p + d)$
Merkle Search Trees	$2\lambda \log m \log_B n$	$O(d \log_B n)$

p	number of nodes, past and present
m	number of nodes currently connected
n	number of past events
d	number of new events in anti-entropy round
λ	network latency

Experimental Evaluation

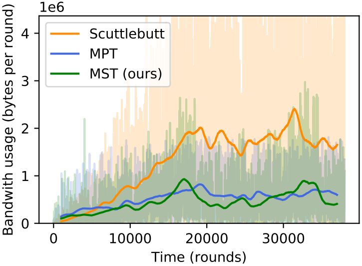
- We evaluate Merkle Search Trees against:
  - SB: Scuttlebutt (vector clocks)
  - MPT: Merkle hash prefix trees (do not preserve order)
- By adjusting the interval between gossip events,
  we can adapt for **faster delivery** or **lower bandwidth use**

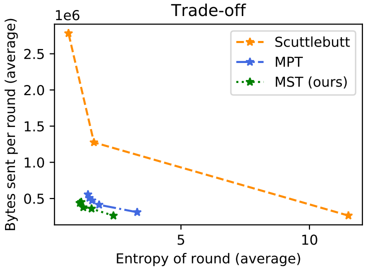
Experimental Evaluation

We test our experiment in a **simulation**:

- **1000 nodes** that generate events at random
- First experiment: 1 event every 10 simulation rounds (in the whole network)

Experimental Results: Low Event Rate





Method	Bandwidth use[a]	Entropy[b]	99% delivery delay
Scuttlebutt	1.3 Mo	1.61	64 rounds
MPT	0.51 Mo	1.44	56 rounds
MST (ours)	**0.44 Mo**	**1.06**	**44 rounds**
Gain vs. SB	-66%	-34%	-31%
Gain vs. MPT	-13%	-26%	-21%

[a] per round, on average
[b] at each round, on average

Experimental Results: High Event Rate

Second experiment: **10 times higher event rate**

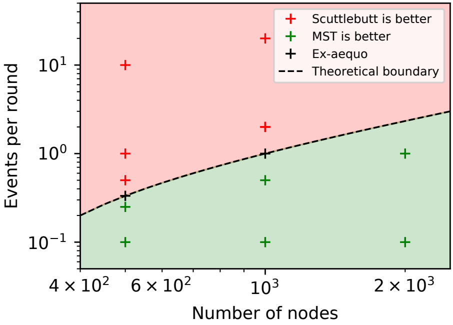
1000 nodes

Method	**Bandwidth use**	**Entropy**	**99% delivery delay**
Scuttlebutt	**2.1 Mo**	**15.4**	**50 rounds**
MST (ours)	2.2 Mo	17.5	74 rounds

2000 nodes

Scuttlebutt	7.6 Mo	**14.9**	**54 rounds**
MST (ours)	**4.2 Mo**	21.0	88 rounds

Experimental Results: Finding the Best Option



- Merkle Search Trees are competitive for low event rates
- Merkle Search Trees scale better when many nodes participate

Conclusion

- Merkle Search Trees are an efficient way of implementing CRDTs in open networks
- Merkle Search Trees could have many other applications (examples: distributed databases)

See our paper for more details:

Alex Auvolat and François Taïani, *Merkle Search Trees: Efficient State-based CRDTs in Open Networks*, SRDS 2019