



Historic Integrity in Distributed Systems

Petros Maniatis
Computer Science Department
Stanford University



What Is History?

- A temporal ordering of system events
 - E.g., store a file on disk, sign a document
- They may happen on different *components* of the distributed system
 - E.g., humans in a social network, peers in p2p
- The ordering of events on different components may be important
 - Did I store my file before you sent that email?
- The ordering and the semantics of the events are sensitive and tamper-prone

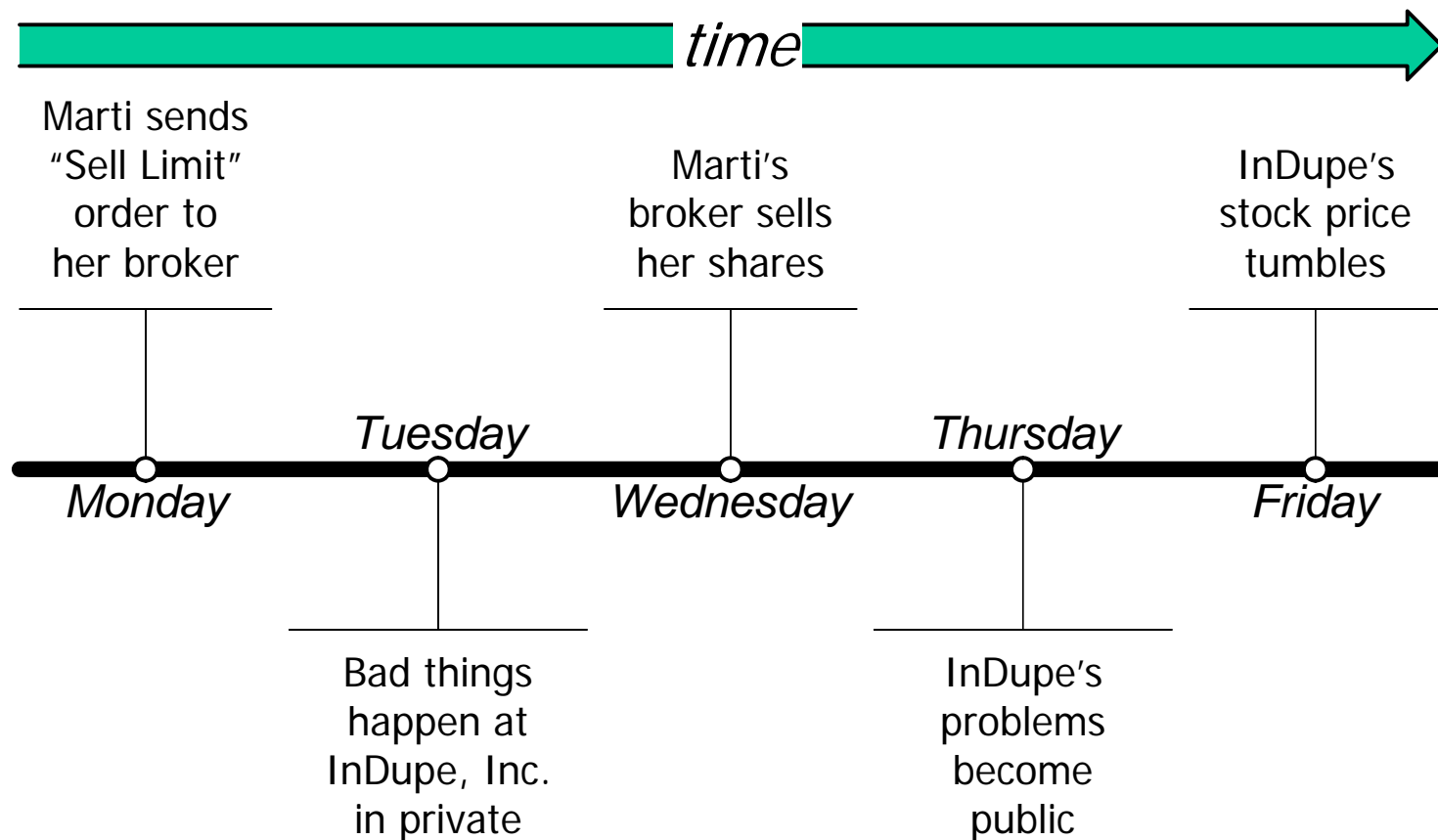


Motivating Example

- Marti is an investor
- She owns shares of InDupe, Inc.
- On Wednesday, Marti's broker sells her shares
- On Thursday, bad news about InDupe surfaces
- On Friday, the stock price of InDupe tumbles
- Later, the SEC accuses Marti of insider trading
- Marti claims her shares were sold according to an earlier, standing sell order

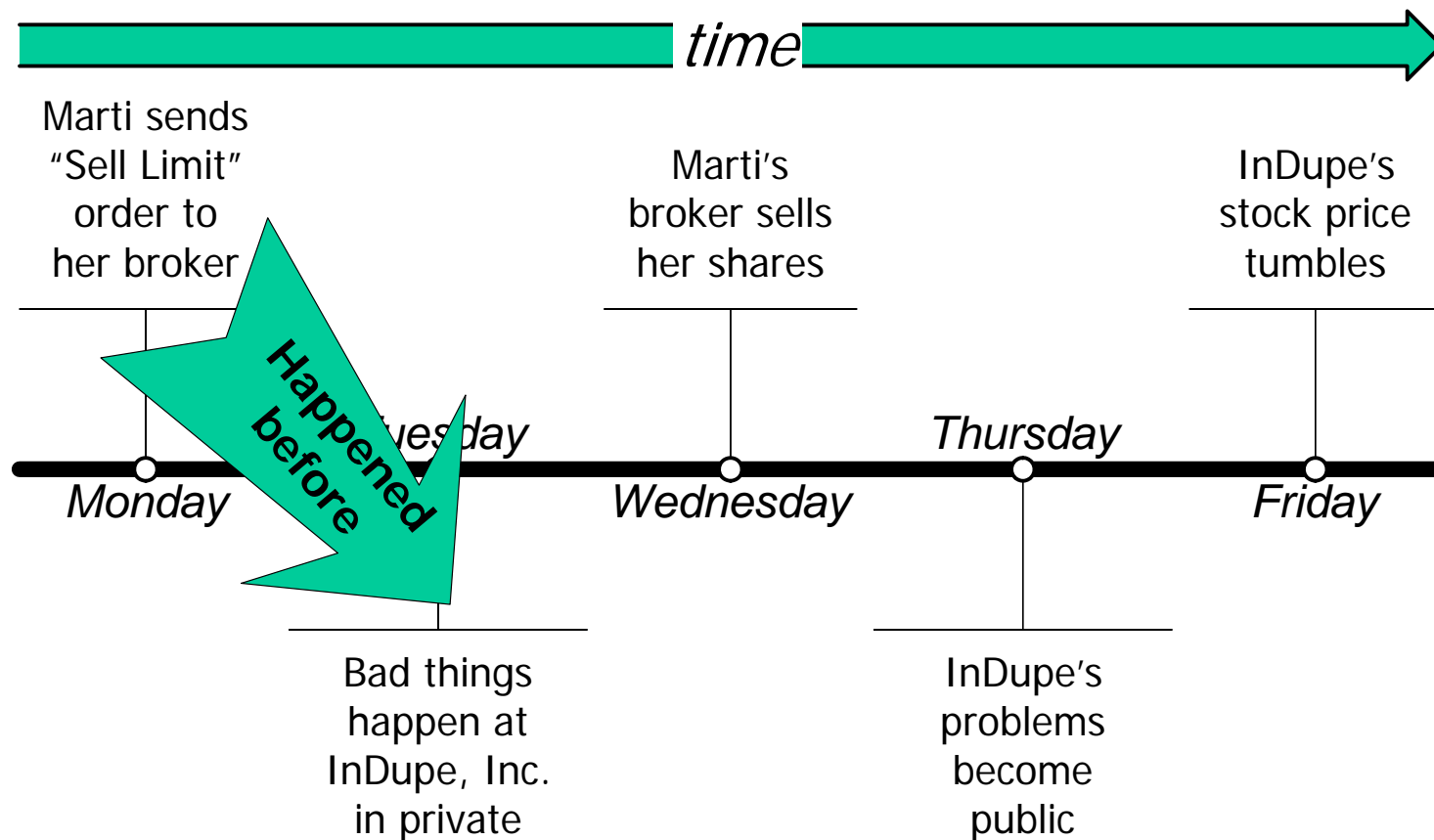


The Events In Pictures



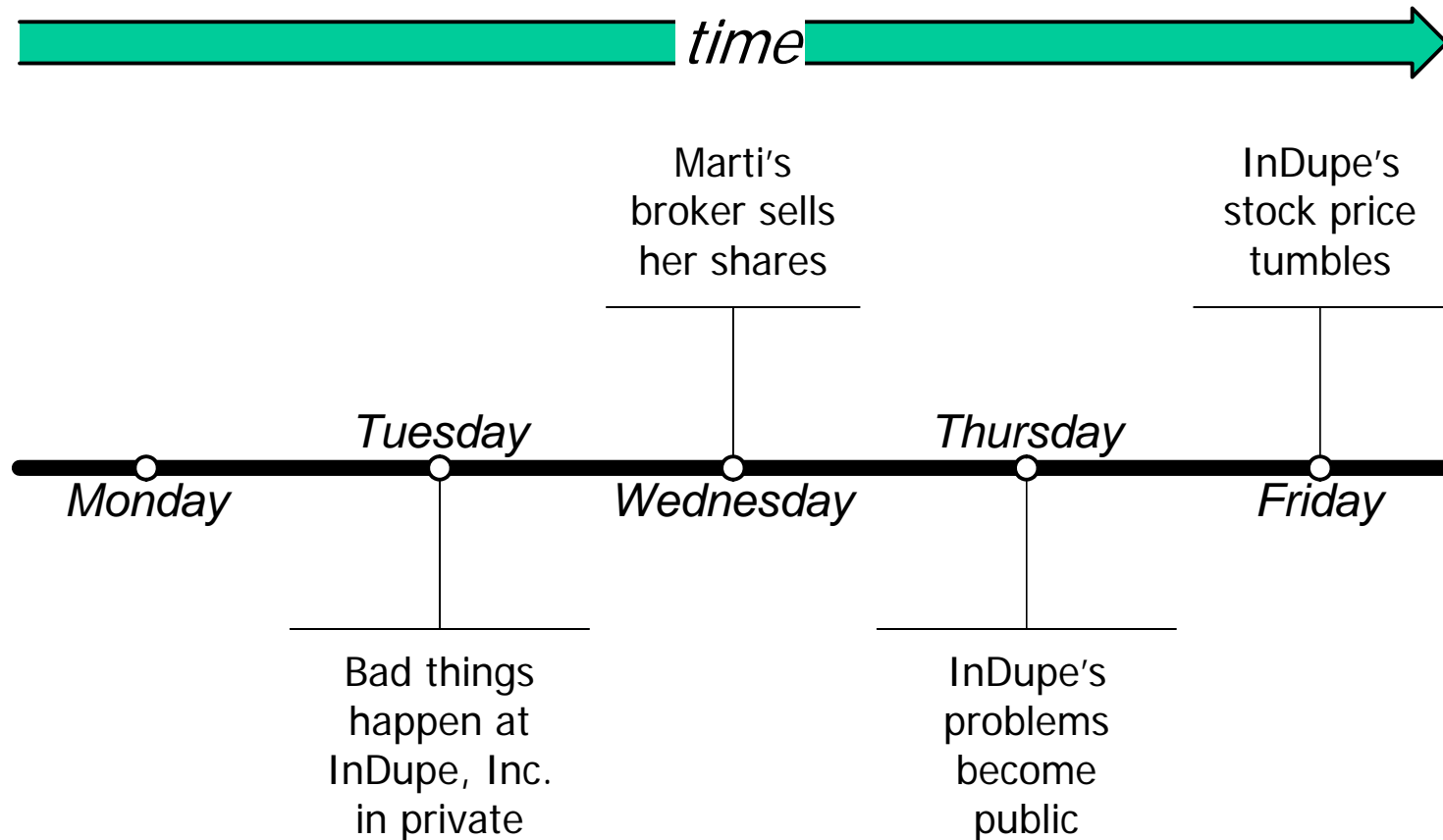


Marti's Dream



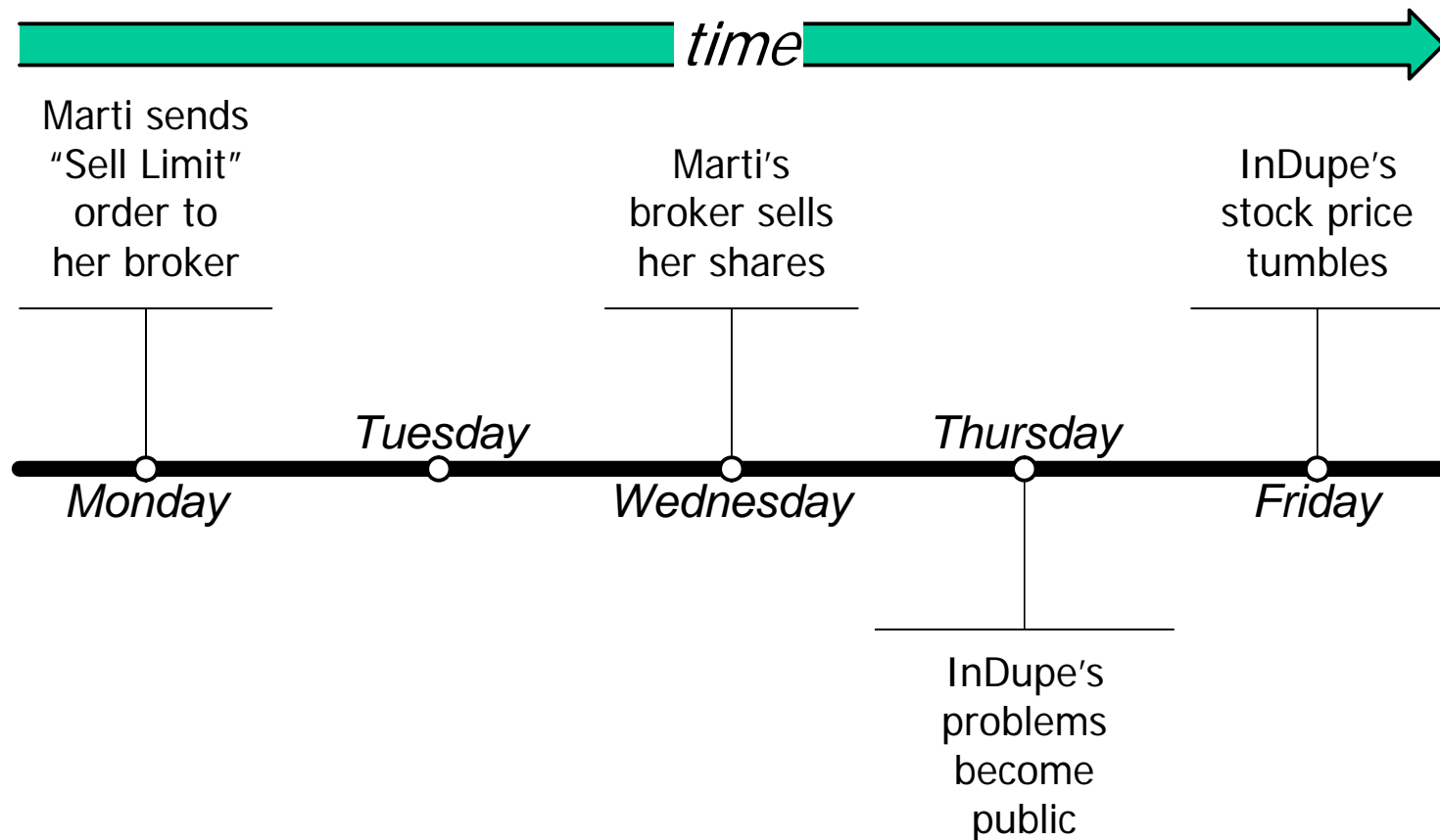


The Sordid Truth?





What If InDupe Goes Under?





Goals

- Maintain the history of a distributed system
 - So that it encompasses seamlessly individual components' histories
 - When components may be untrustworthy
 - Long after individual "historians" leave
- **Basic assumption: no one trusts anyone else**

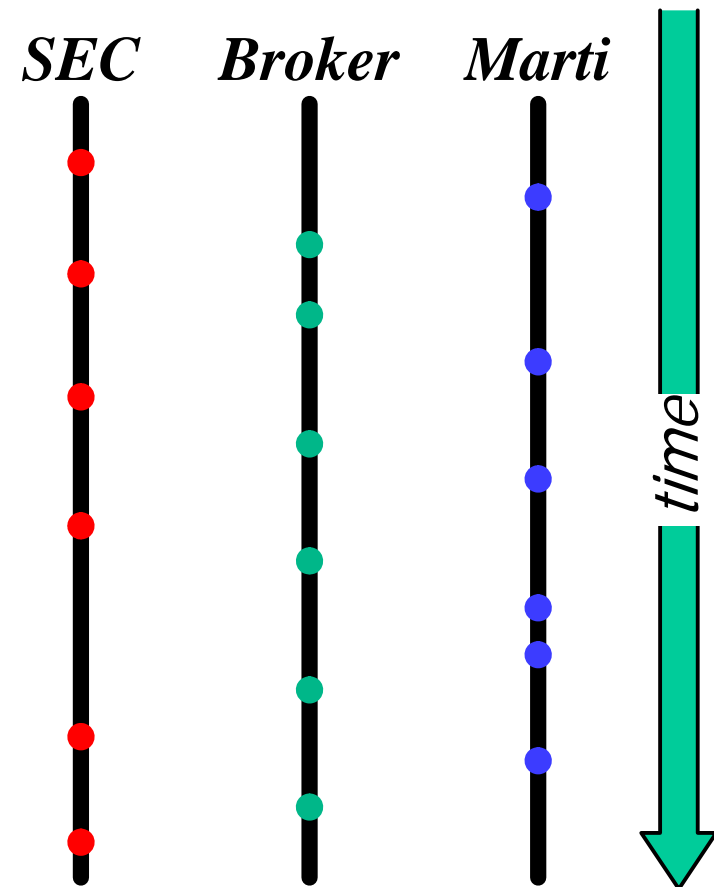
Collective





Goals

- Maintain the history of a distributed system
 - So that it encompasses seamlessly individual components' histories
 - When components may be untrustworthy
 - Long after individual "historians" leave
- **Basic assumption: no one trusts anyone else**





Our Approach: Timeweave

- Every component
 - Maintains its own local history
 - Maintains a local view of global history
 - Safeguards the integrity of portions of global history it knows about
 - Only trusts itself or provable information
- Requirements
 - Efficient, scalable, survivable, aggressively decentralized



Thesis Contributions

- Architecture for maintaining a secure collective history of a distributed system
 - Implementation (Timeweave)
- Novel authenticated data structures (persistent undeniable attestations)
 - Append-only authenticated skip lists
 - Persistent authenticated search trees
- Applications
 - Distributed and survivable secure time stamping
 - Archival storage of digitally signed documents



Overview

- Timeweave design
 - Secure timeline: Single component history
 - Timeline entanglement: Collective history
- Data structures
 - Persistent authenticated search trees
- Evaluation
 - Performance
- Applications
 - Distributed, Survivable Time Stamping



Secure Timelines

The history of a single component



Secure Timeline

- A hash chain of local event commitments
 - Event: state change, message received, message sent, etc.
 - Commitment: if an event is purely internal, I don't *have* to put it in my history
- The one-way hash function defines the "arrow of time" among events



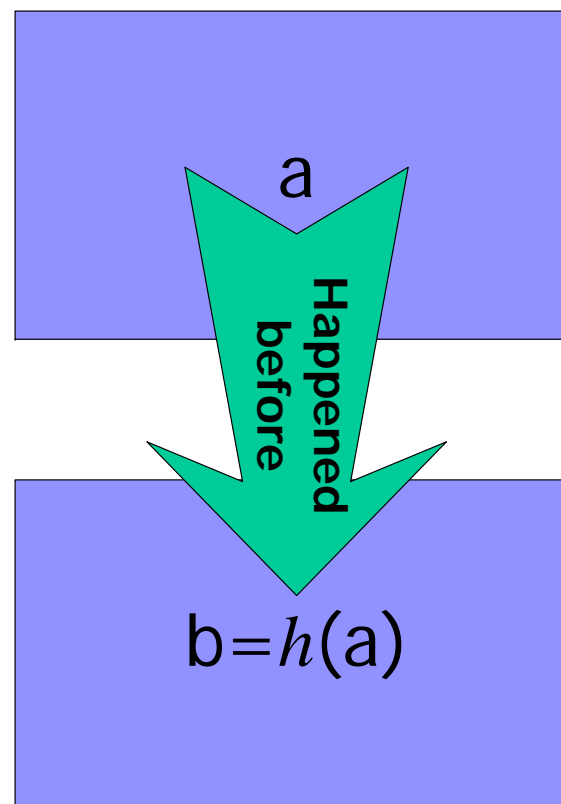
One-Way Functions and The Arrow of Time

- A function h is one-way if
 - Given \mathbf{a} , I can compute $h(\mathbf{a})$ easily
 - Given $h(\mathbf{a})$, it is intractable to compute \mathbf{a}
- SHA-1 is considered one-way
- If $h(\mathbf{a}) = \mathbf{b}$, then I can safely assume that \mathbf{a} was known before \mathbf{b}
 - Otherwise, someone must have found the right \mathbf{a} , either intentionally or by luck



One-Way Functions

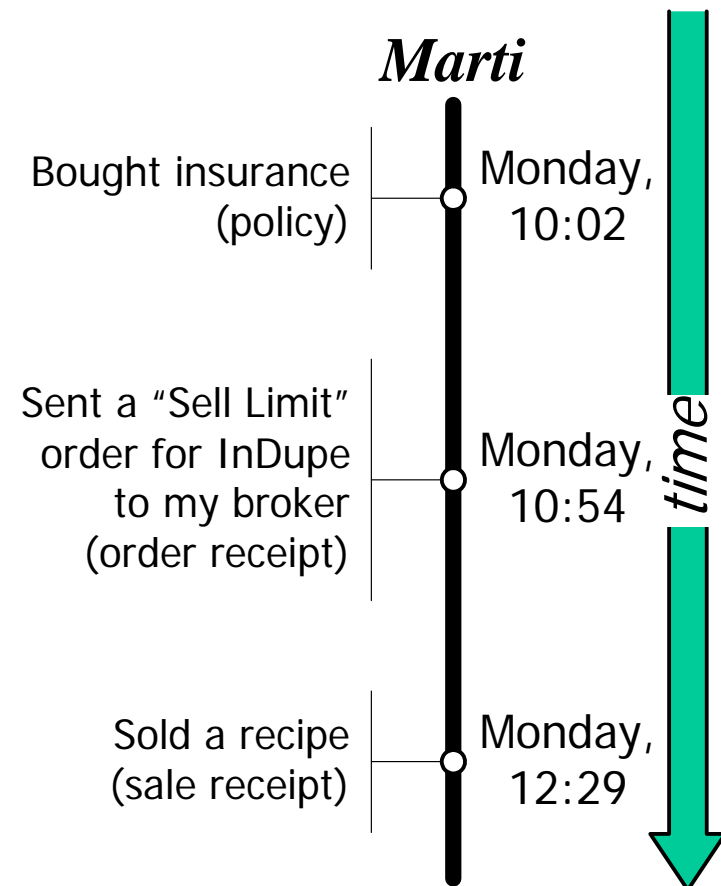
- Cryptographic hash functions can be used to “time” digital records
- I can reason about the ordering in which two values were computed
- The ordering stands regardless of how the values are written on a digital medium (disk)





Timeline Example

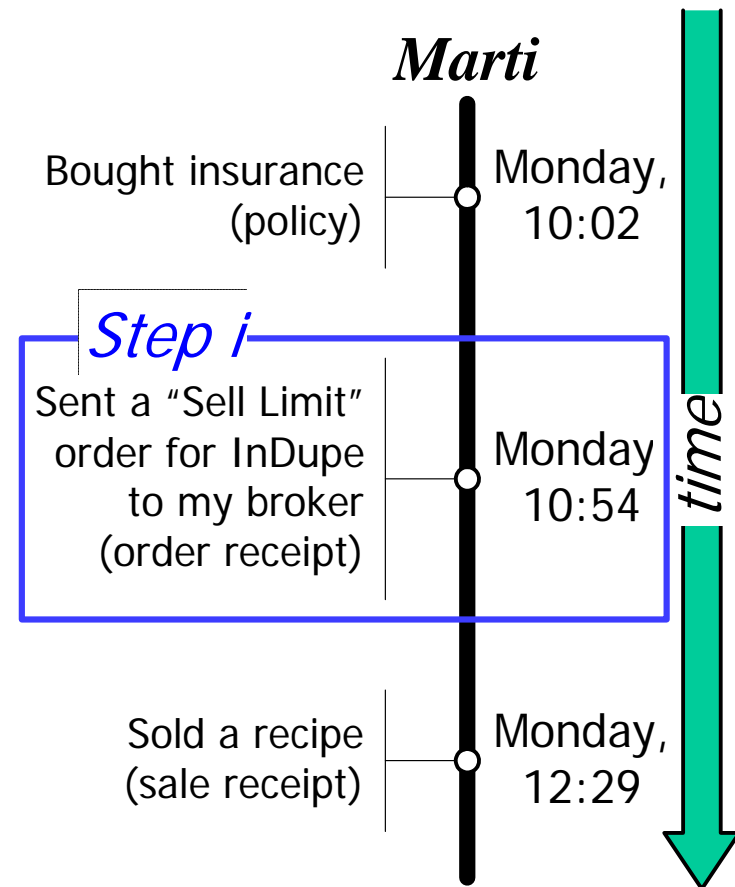
- Marti performs a bunch of "events" between 10am and 1pm
- Those she considers significant she places in her local history
- With every event she advances her history by one time step





Zoom In On Time Step i

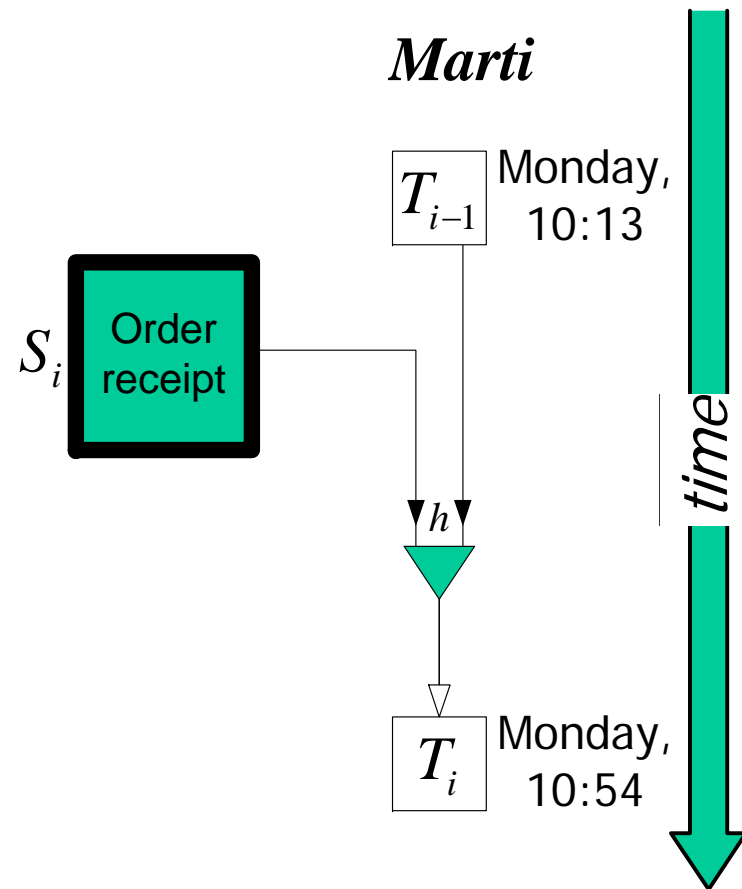
- Marti performs a bunch of "events" between 10am and 1pm
- Those she considers significant she places in her local history
- With every event she advances her history by one time step





Zoom In On Time Step i

- A time step consists of
 - Logical Time (i): counter of time steps
 - Committed event (S_i): order receipt
 - Authenticator (T_i):
$$T_i = h(i \parallel T_{i-1} \parallel S_i)$$





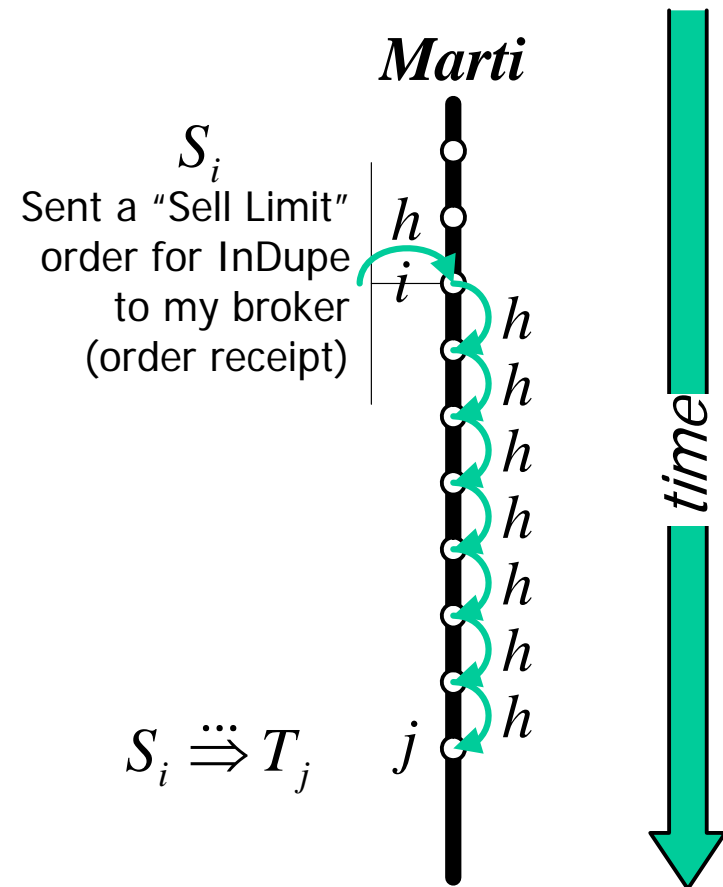
The Joy Of Timelines

- A timeline step authenticator is a one-way digest of an entire timeline
- Given the current authenticator, no previously committed event can
 - Change contents
 - Change logical times
 - Be inserted or deleted
- By committing to an authenticator, a component commits to its history thus far



Proof of Precedence

- Given the latest timeline authenticator, I can prove the commitment of an old event
- “Walk” the hash applications from the event to the authenticator
- All the intermediate information in the hash chain makes up the proof





Timeline Entanglement

Interconnecting Individual
Histories



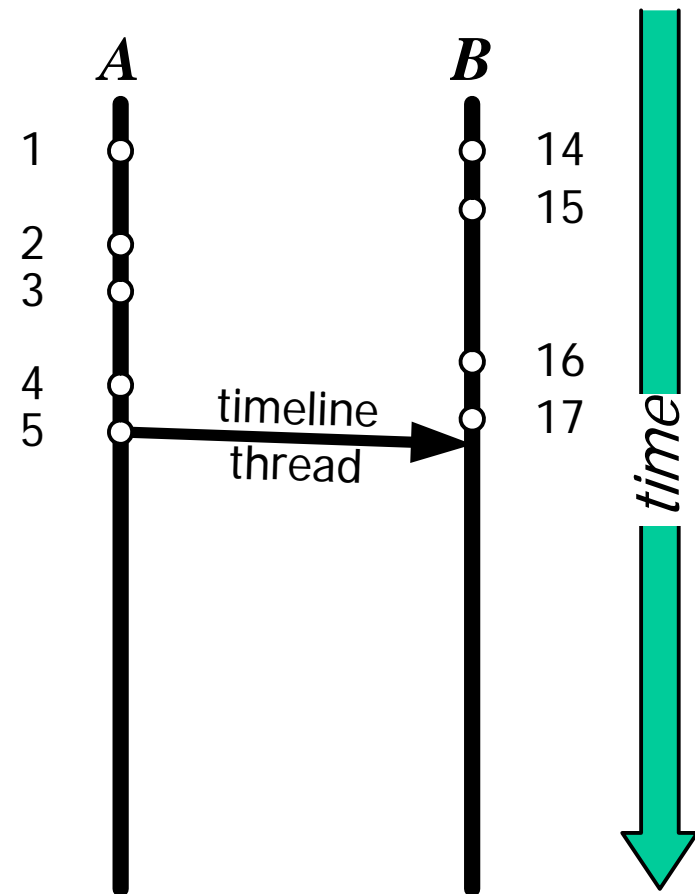
Timeline Entanglement

- Multiple interconnected hash chains
 - A component publishes samples (authenticators) of its timeline for other components to witness
 - Witness components commit received timeline samples in their own timelines
- Benefits
 - ✦ Witnessed timelines are tamper-evident
 - ✦ Entanglement enables temporal comparisons among different components' events



Timeline Entanglement

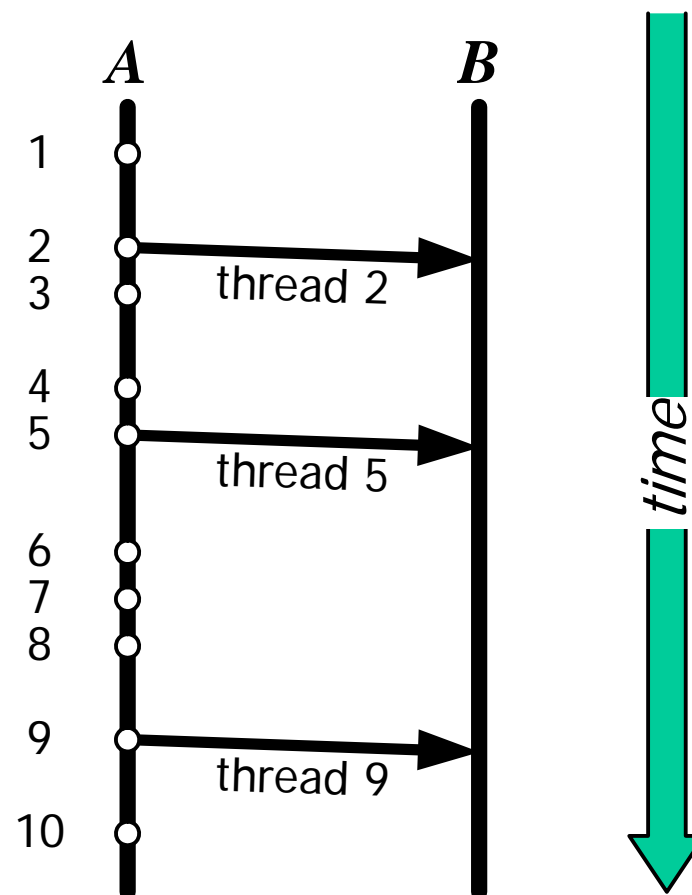
- Component *A* *entangles* its timeline with *B*
- It sends *B* a *timeline thread* :
 - Name
 - Logical time
 - Timeline authenticator
 - Signature on the above
 - Validation info
- *B* checks the validity of the thread and then records it in its timeline





Timeline Sampling

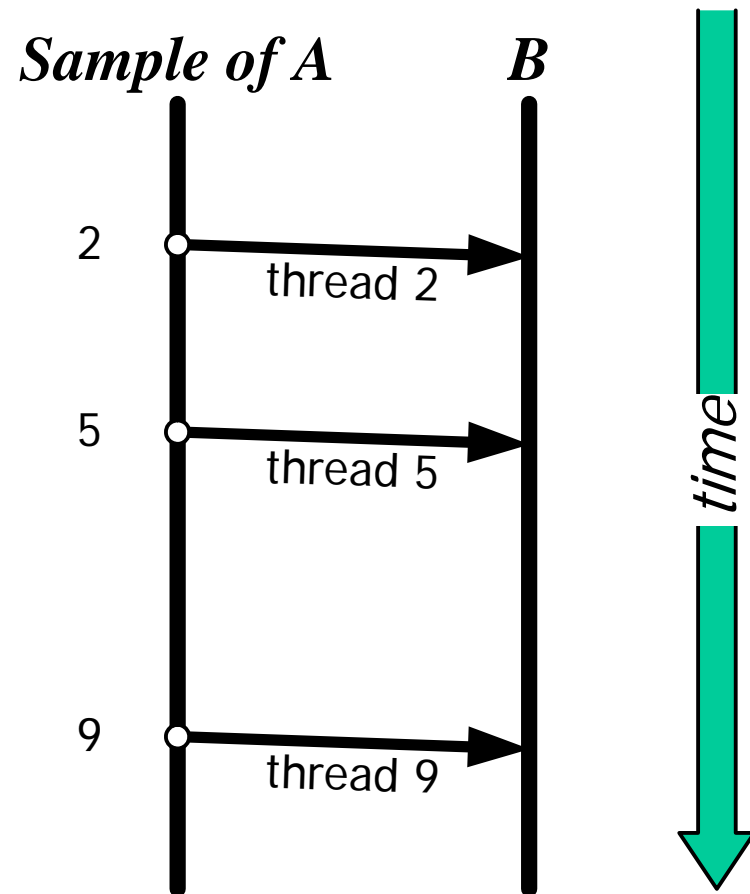
- Every entanglement conveys a *sample* of the sender's timeline





Timeline Sampling

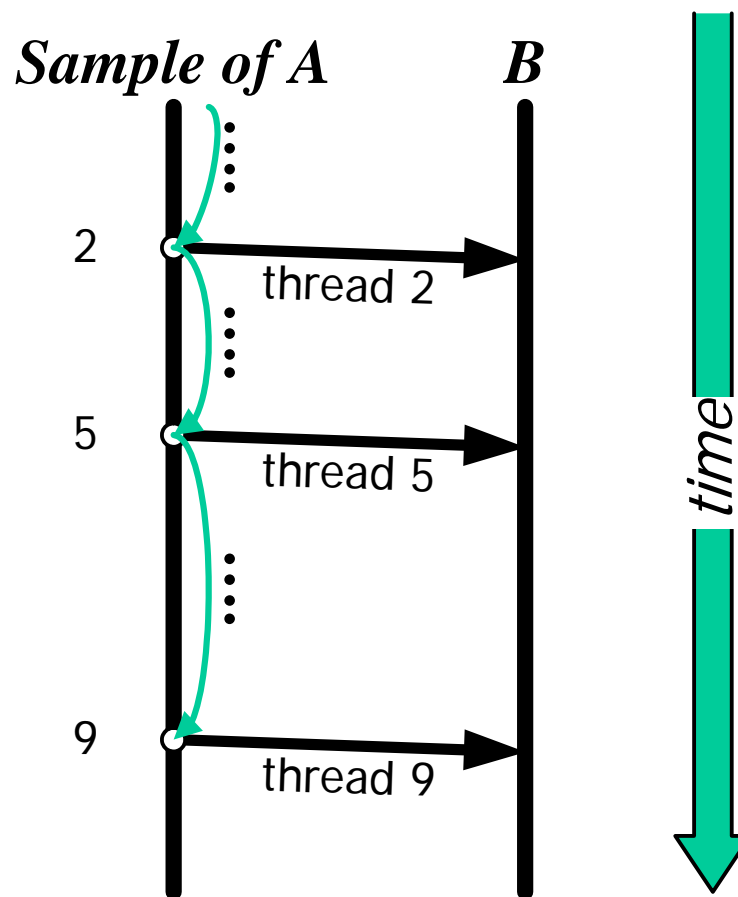
- Every entanglement conveys a *sample* of the sender's timeline
- Receiver's view of the sender's timeline has coarser granularity





Timeline Sampling

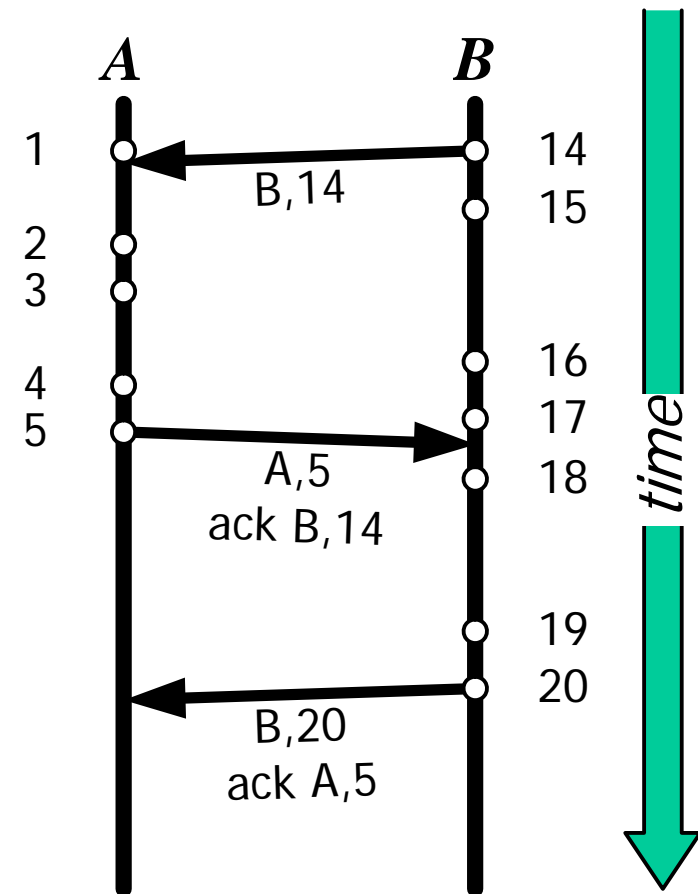
- Every entanglement conveys a *sample* of the sender's timeline
- Receiver's view of the sender's timeline has coarser granularity
- Validation info: sample-to-sample precedence proof





Entanglement Receipts

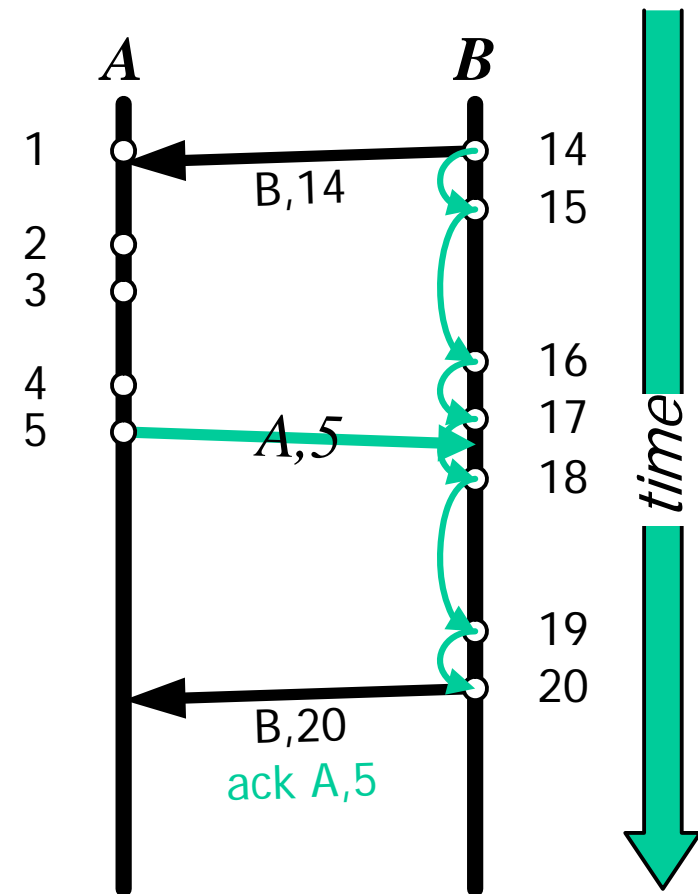
- Timeline threads are acknowledged with a *receipt*





Entanglement Receipts

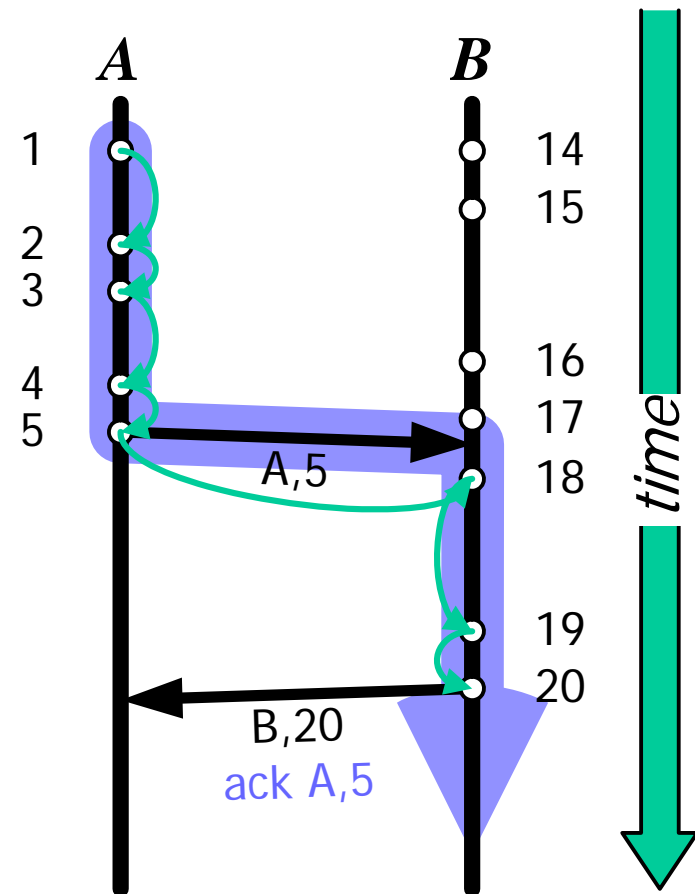
- Timeline threads are acknowledged with a *receipt*
- A receipt proves the commitment of a thread
 - It was archived in the latest archive
 - It will be found when looked up (undeniably)





Cross-timeline Precedence

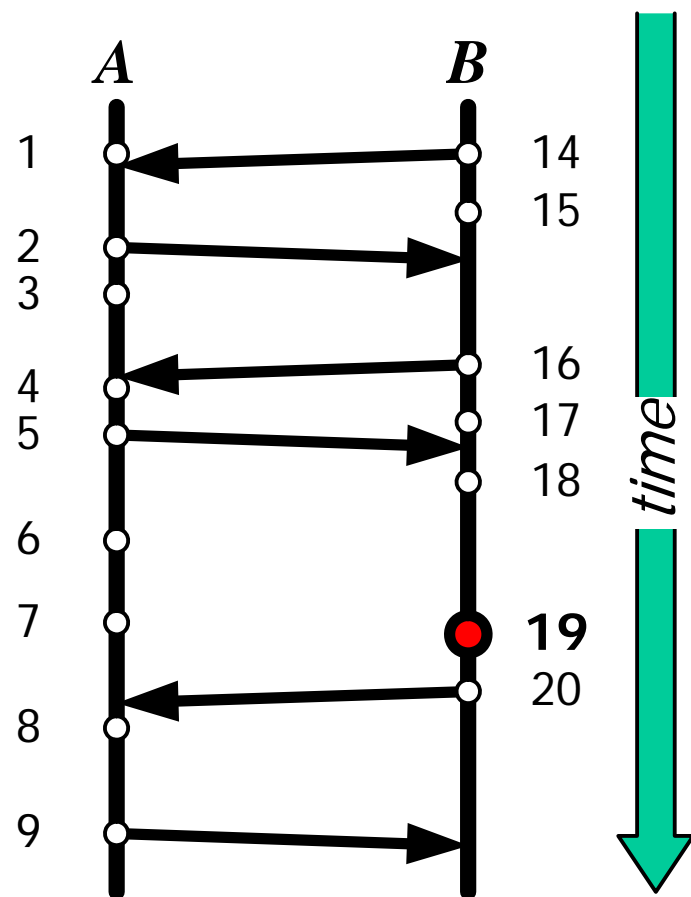
- Inclusion of a foreign thread in my timeline builds a precedence
 - From the past of the foreign timeline
 - To my future
- Sender receives proof of this with the subsequent receipt acknowledging the thread, which it stores





Temporal Mapping

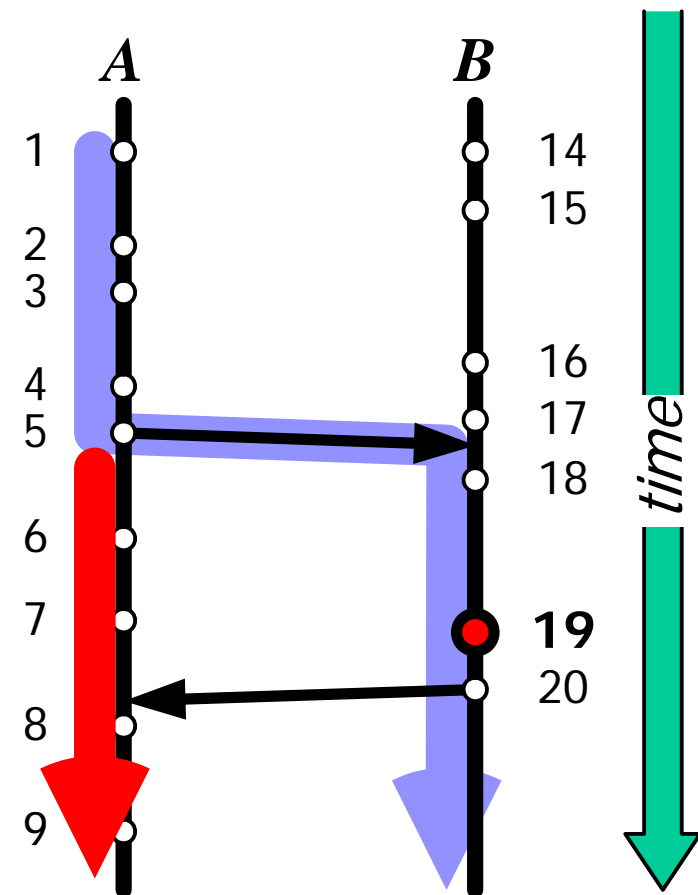
- A wants to know when event $\langle B, 19 \rangle$ happened





Temporal Mapping

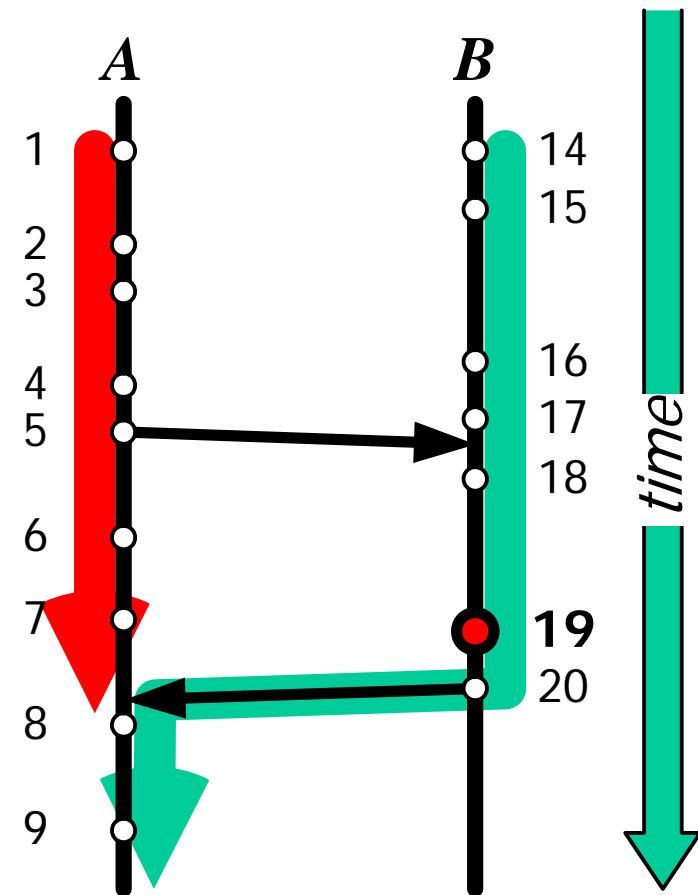
- A wants to know when event $\langle B, 19 \rangle$ happened
- Event $B, 19$ follows all of A's events up to and including 5





Temporal Mapping

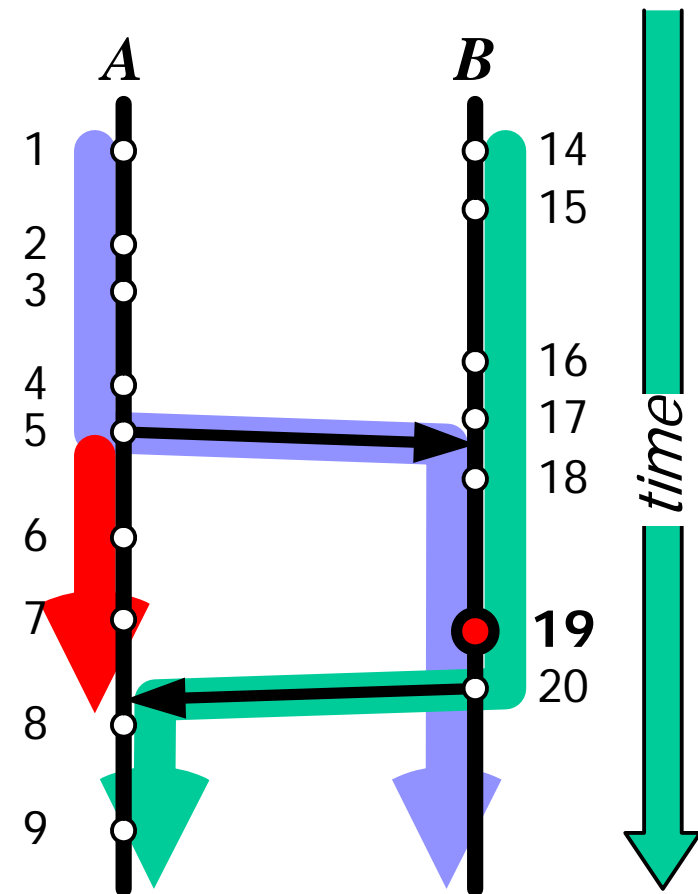
- Event B, 19 follows all of A's events up to and including 5
- Event B, 19 also precedes all of A's starting with 8





Temporal Mapping

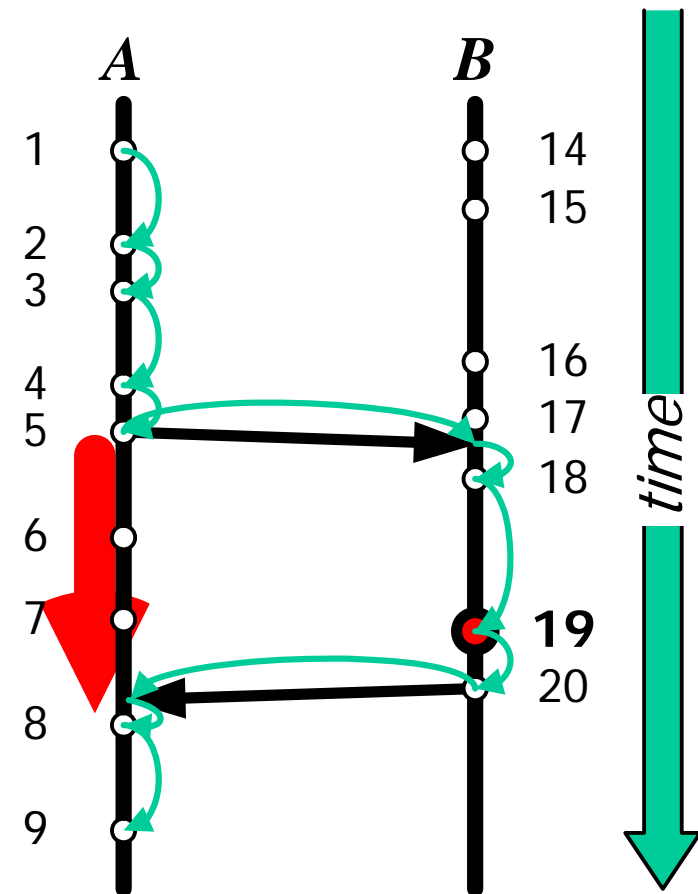
- Event B,19 follows all of A's events up to and including 5
- Event B,19 also precedes all of A's starting with 8
- Combining the two, we conclude that B,19 happened between A's 5 and 8





Temporal Mapping

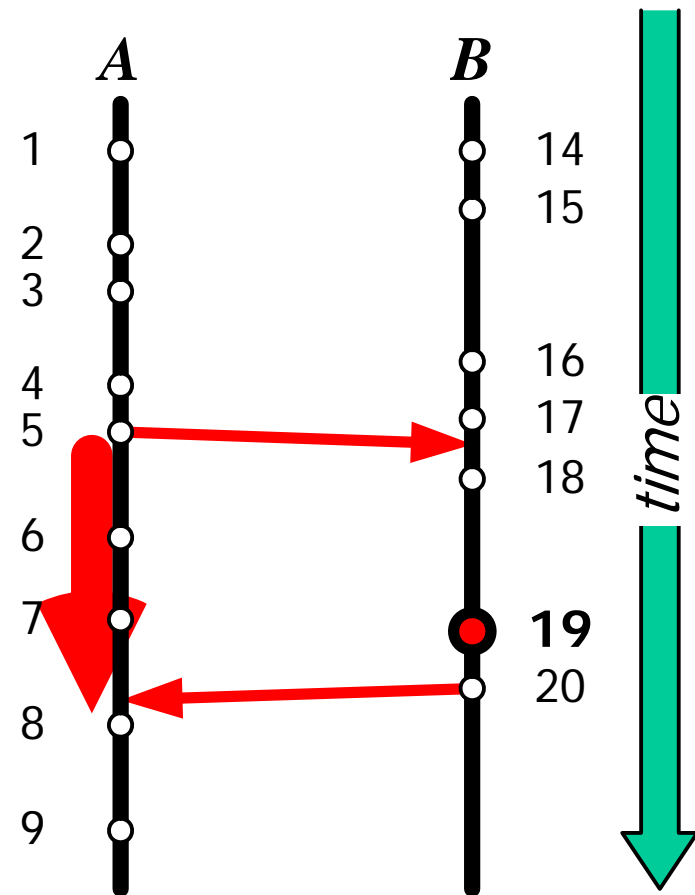
- Event B,19 follows all of A's events up to and including 5
- Event B,19 also precedes all of A's starting with 8
- Combining the two, we conclude that B,19 happened between A's 5 and 8





Loss of Temporal Resolution

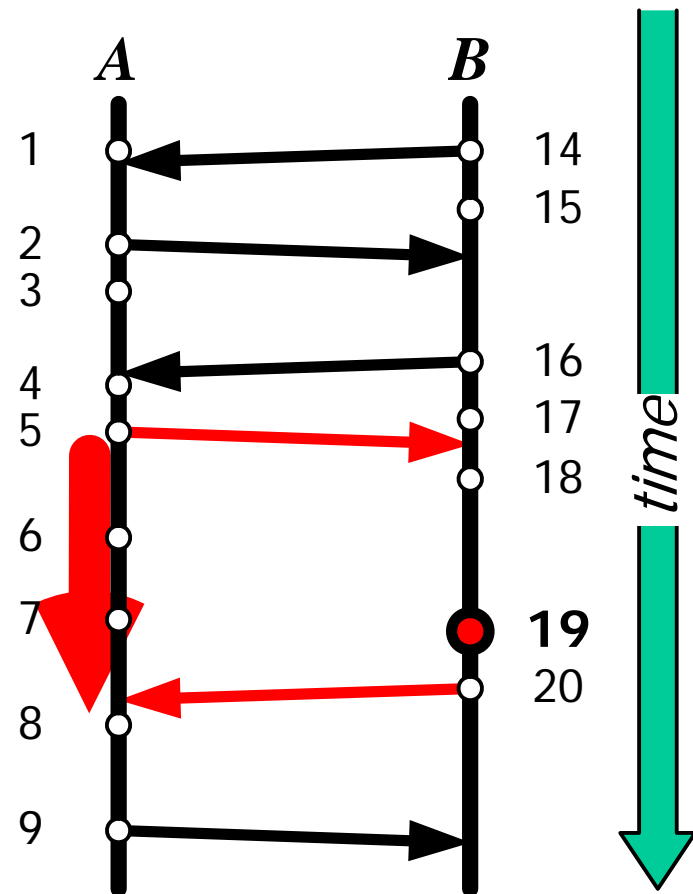
- Mapping causes time to expand
- Expansion means loss of *temporal resolution*
- Expansion is dependent on how frequently entanglement happens





Mapping Usefulness

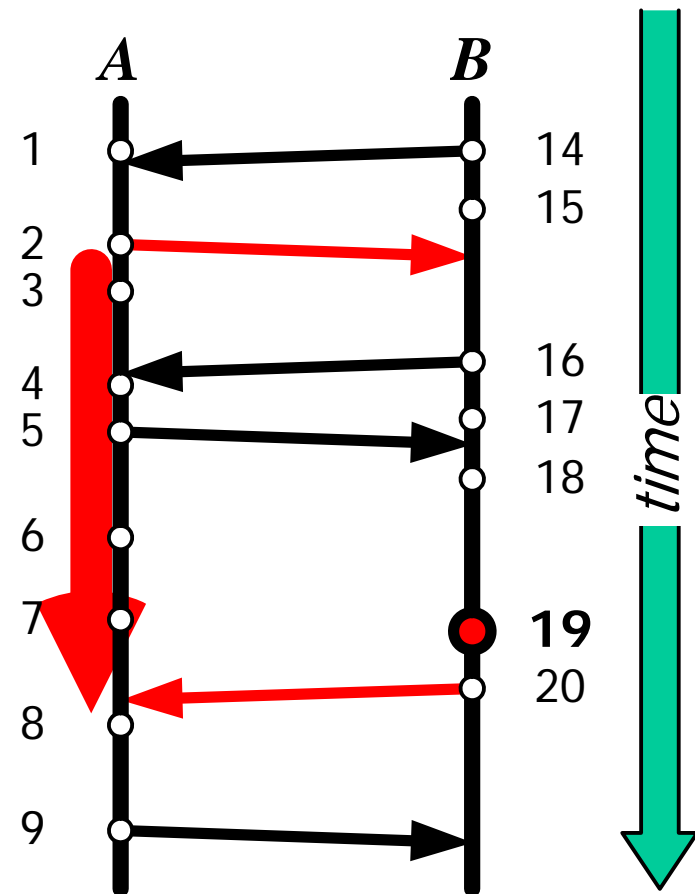
- If B produces the mapping, it should pick the tightest one possible





Mapping Usefulness

- If B produces the mapping, it should pick the tightest one possible
- As opposed to an arbitrary superset
- The data structure we use to store threads helps keep mappers honest





Entanglement Architecture

Putting it all together



Setting

- Entangled Service Set
 - A group of nodes
 - All nodes entangle (regularly) with all others
- Entanglement Interval
 - Period between successive entanglements between two nodes
 - It defines the loss of temporal resolution during entanglement



Architecture

- Every node maintains persistently
 - Its own secure timeline
 - An authenticated archive of the timeline threads it receives from other nodes
 - An archive of entanglement receipts
- Disposable (cached) information
 - Tracking information about other peers
 - last incoming thread (signed)
 - last outgoing thread



Functionality

- History
 - Commit events
- Entanglement message exchange
 - Send/receive threads & receipts
- Lookups
 - Map local time to remote timeline
 - Map remote time to local timeline
 - Extract proof of precedence for local event



Data Structures

Persistent Undeniable Attesters



Persistent Undeniable Attesters

- **Attester:** Given a data collection, and a digest of that collection, I can prove an element belongs to the collection succinctly
 - e.g., hash trees [Merkle]
- **Undeniable attester:** An attester that cannot prove conflicting answers [Buldas]
- **Persistent undeniable attester (PUA):** An attester that cannot prove conflicting answers even after its contents change



PUAs In Timeweave

- Secure Timelines
 - Once I prove an event belongs in my timeline, I can never prove it does not, to anyone
 - Once the verifier sees a satisfactory proof, he can discard it!
 - *Append-only authenticated skip lists*
- Thread Archives
 - Once I place a thread into my thread archive, I can never convince anyone the thread is not there
 - *Persistent authenticated search trees*



Persistent Authenticated Search Trees

Building archives of timeline
threads



Thread Archive Goals

- Archive threads as they arrive
 - **<Sender, Logical time, Authenticator>**
- Commit changes to the thread archive
- Prove that the archive contains a thread (“membership” questions)
 - for receipts
 - for mapping
- Prove that an answer is the closest possible (“predecessor” questions)
 - for mapping

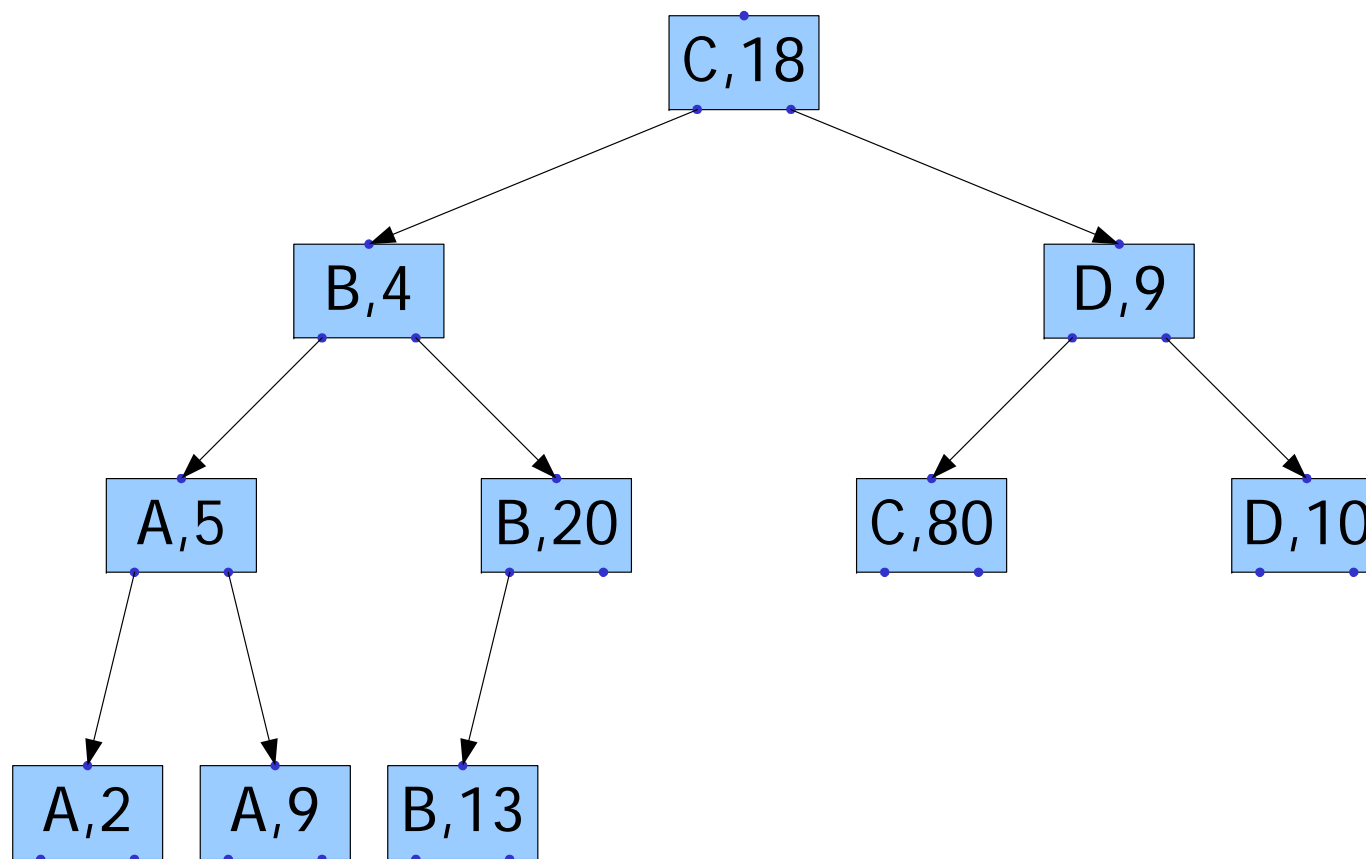


Archiving Timeline Threads

- Basis
 - Authenticated search trees [Buldas2002]
- ✦ Persistence
 - Maintain old versions of the data structure
 - Commit version digests in a skip list
- ✦ Storage design
 - Share data for space efficiency (copy on modify)
 - Retrofit B-Trees for access efficiency
 - Embed balanced binary trees for proof efficiency

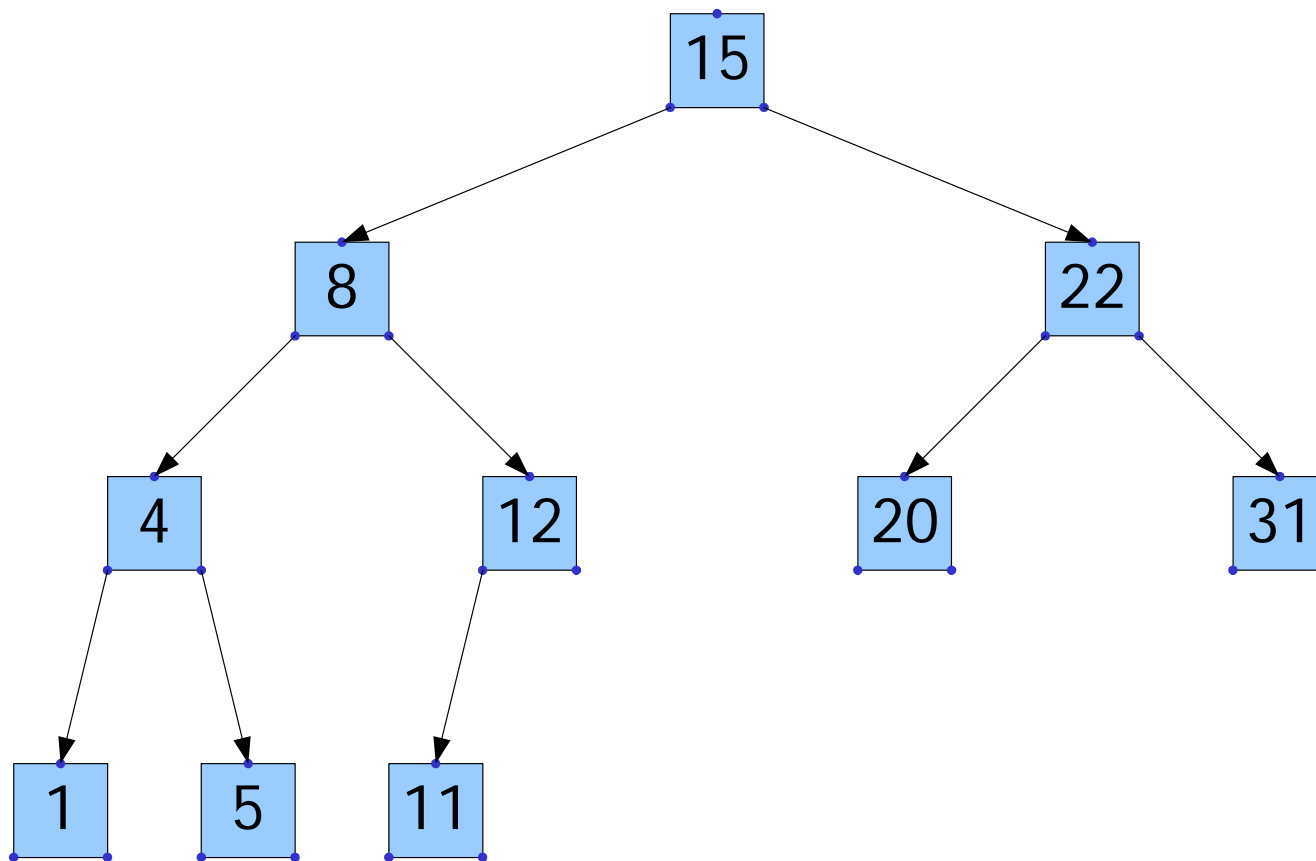


Binary Search Trees



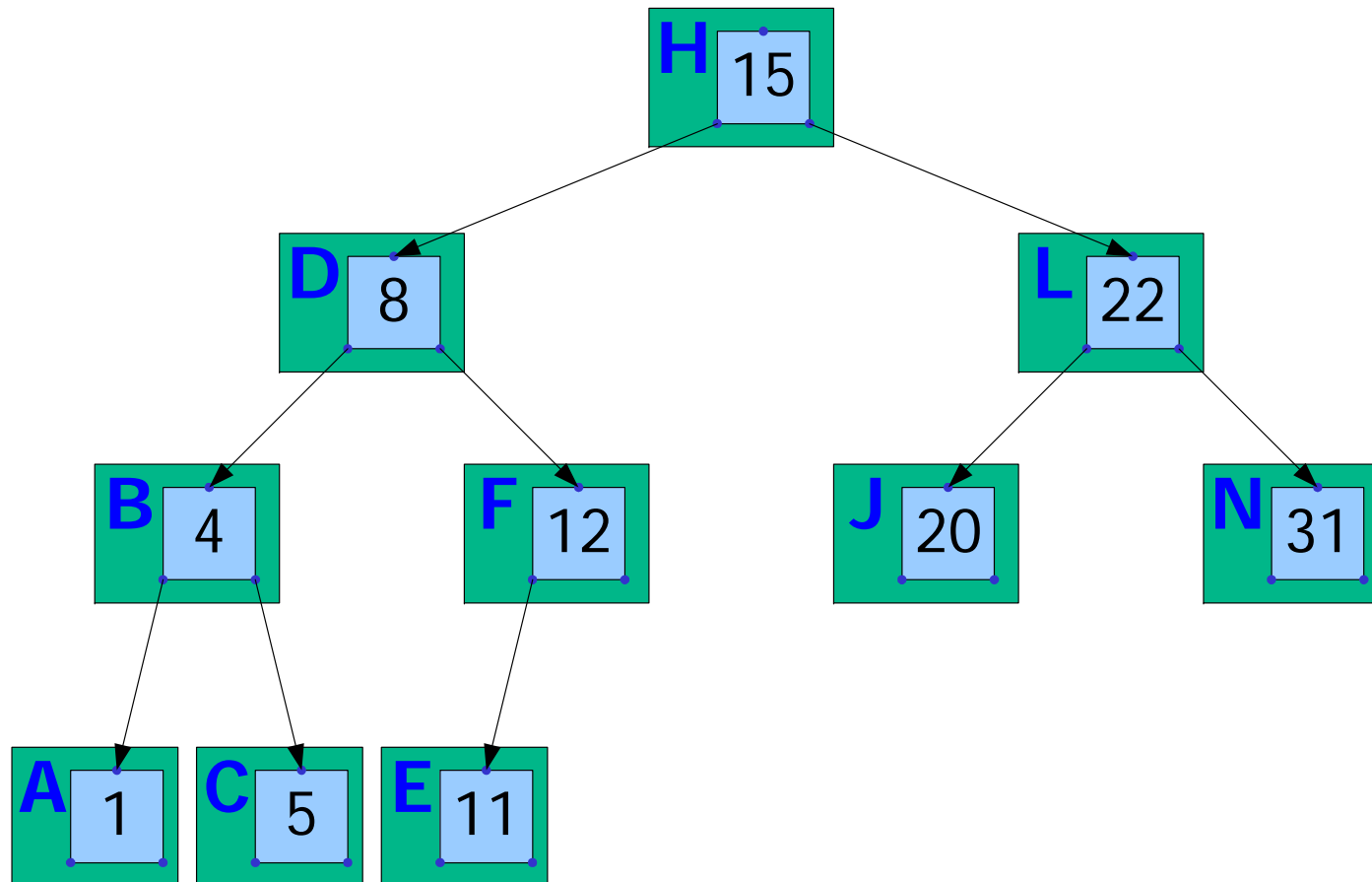


Binary Search Trees



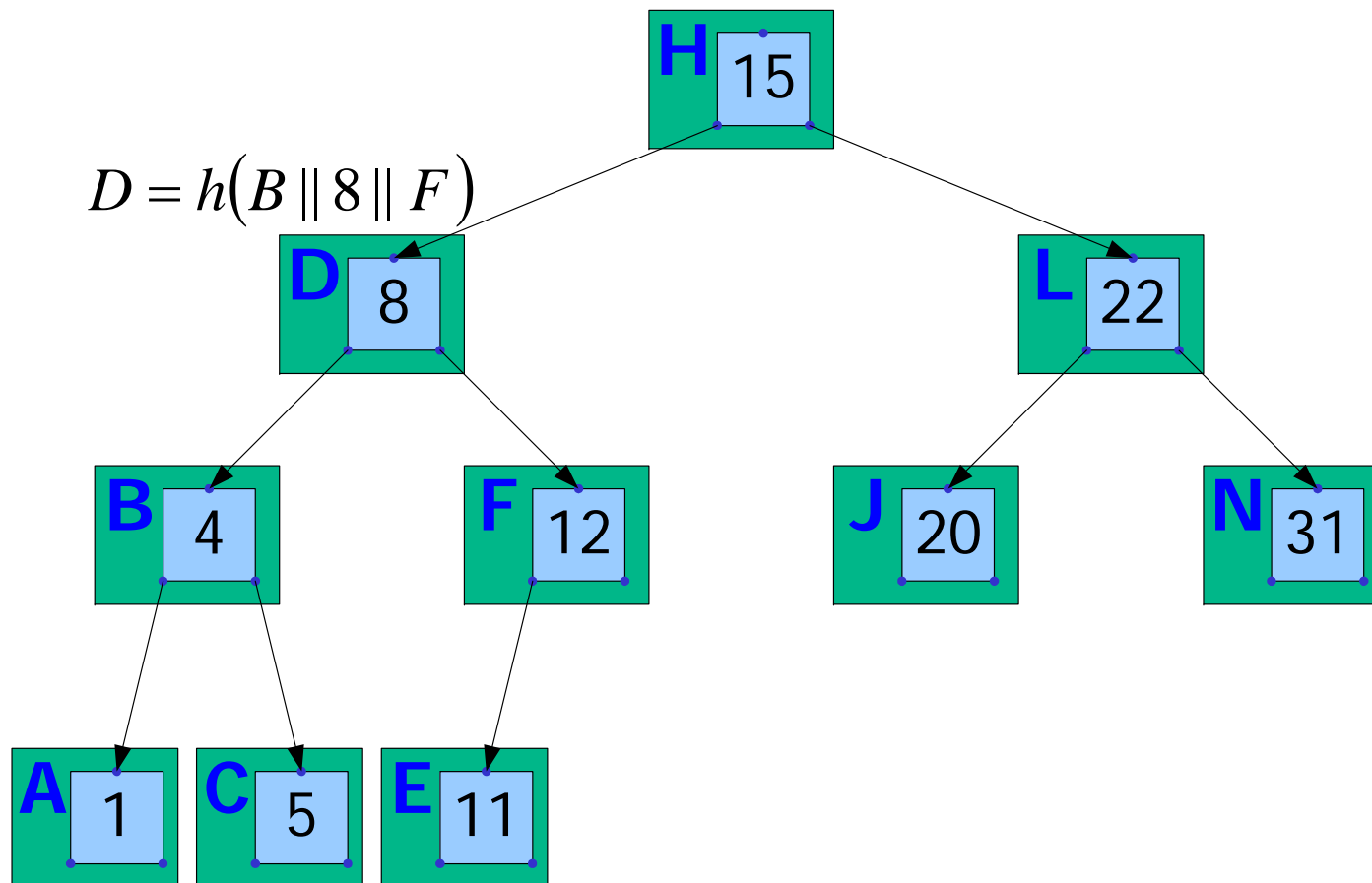


Authenticated Search Trees



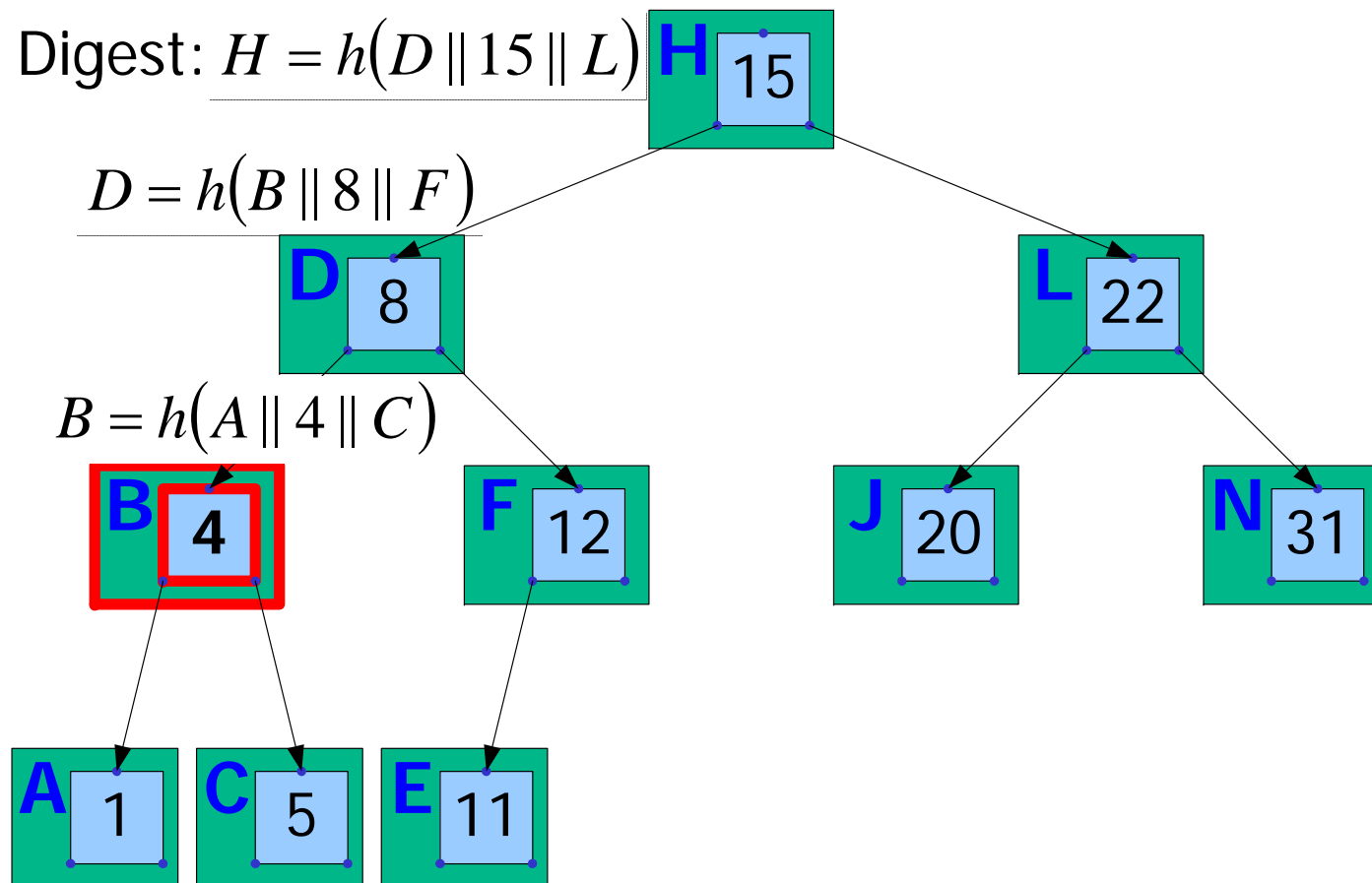


Authenticated Search Trees



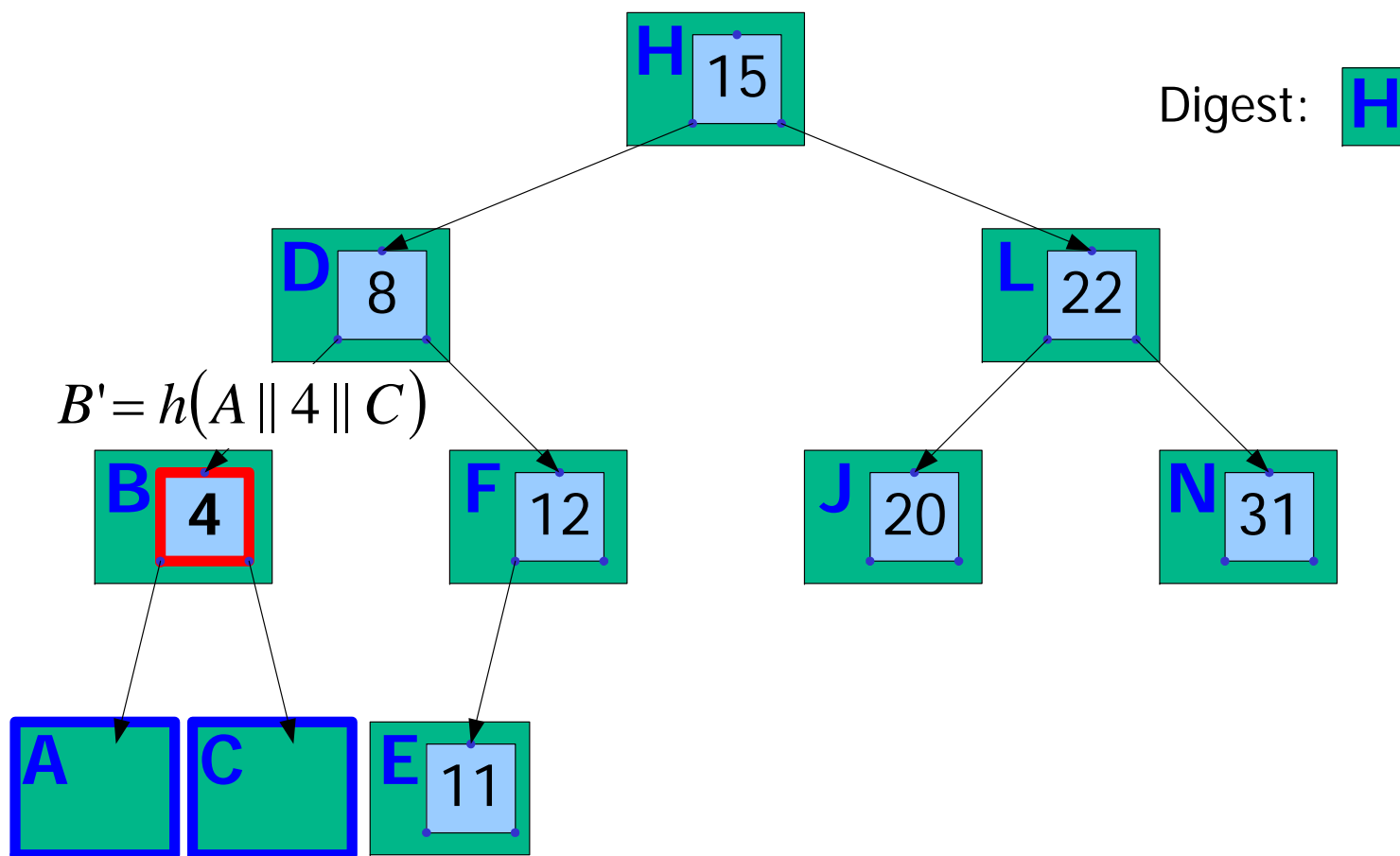


Tree Digest



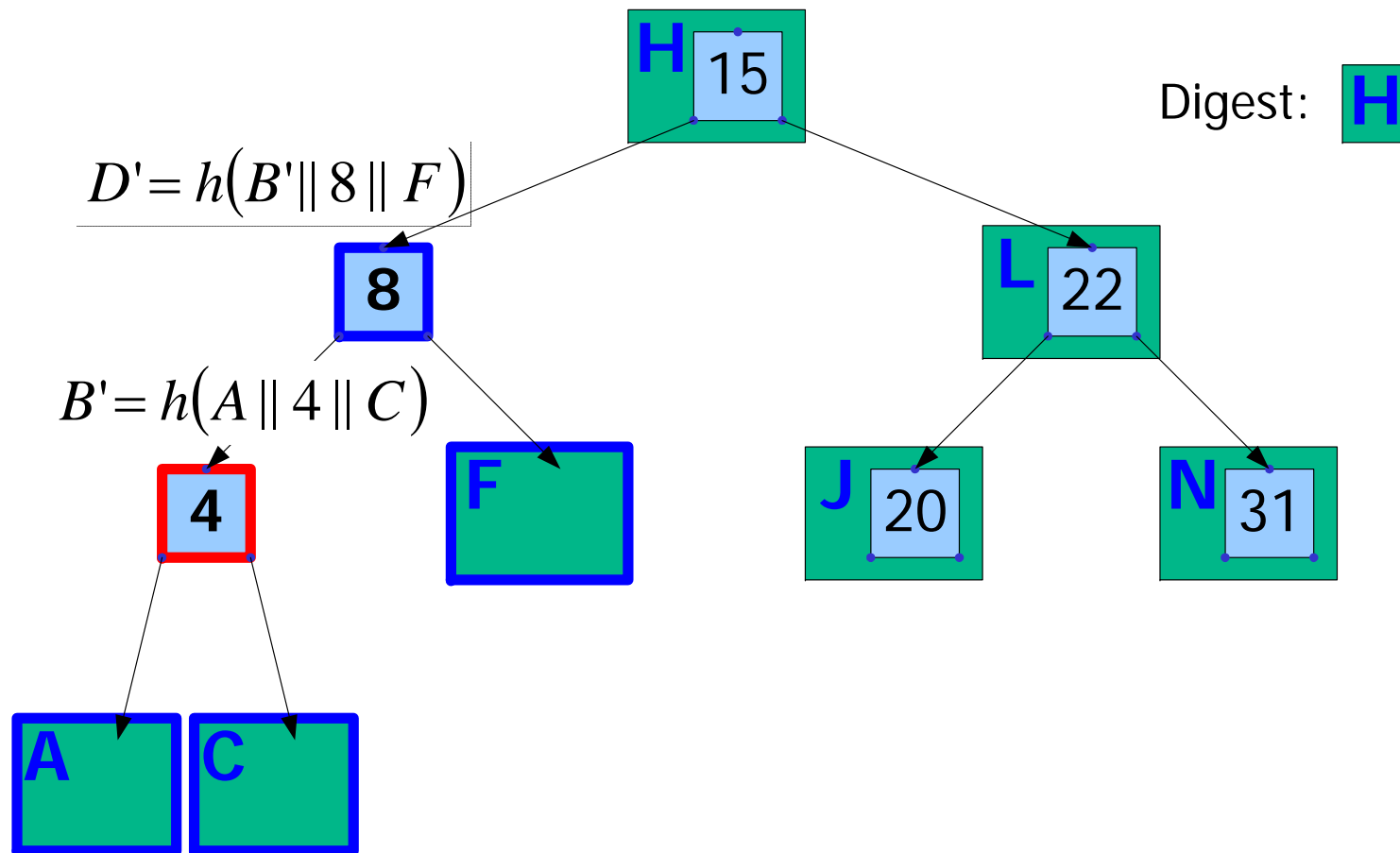


Existence Proof Construction



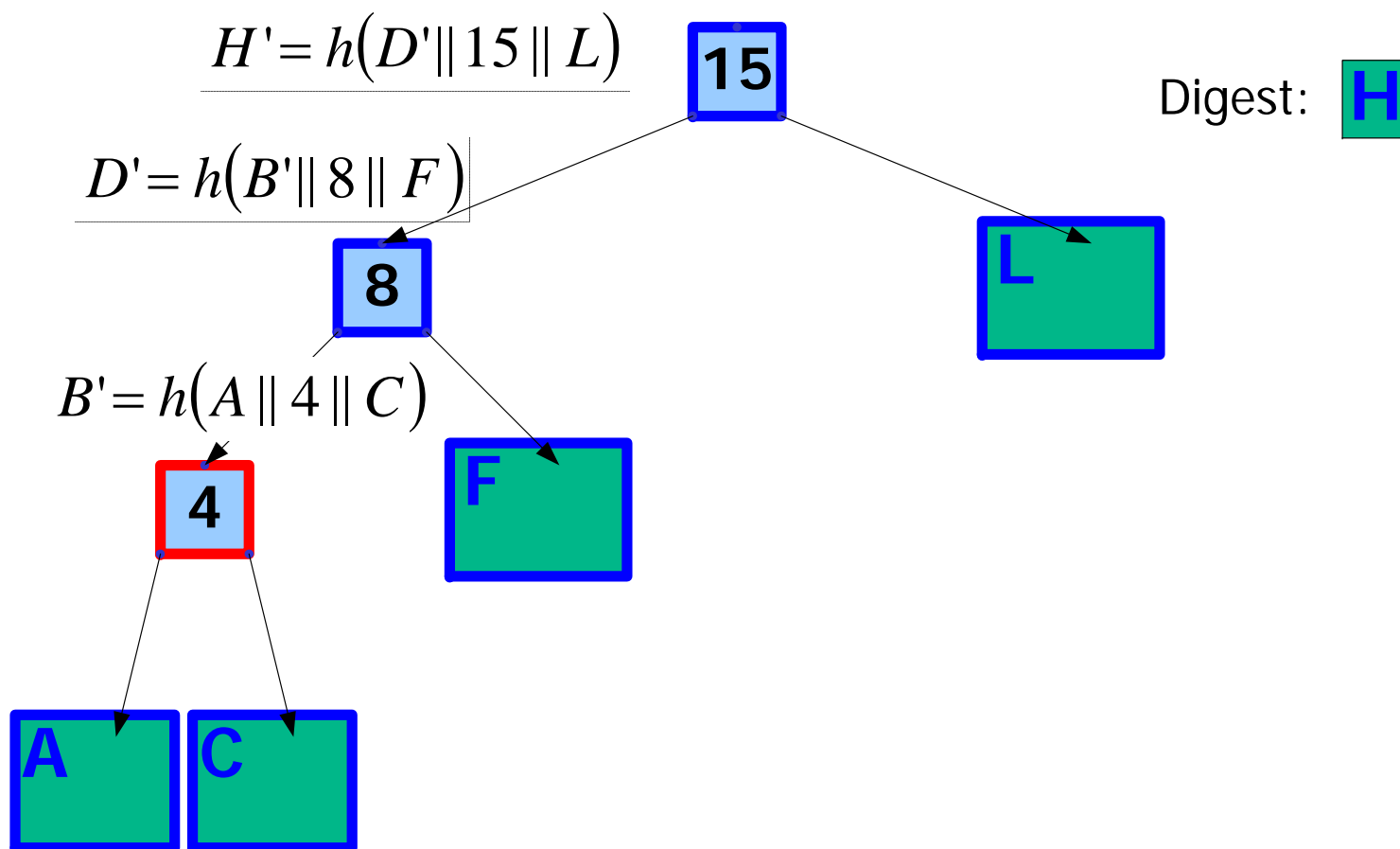


Existence Proof Construction



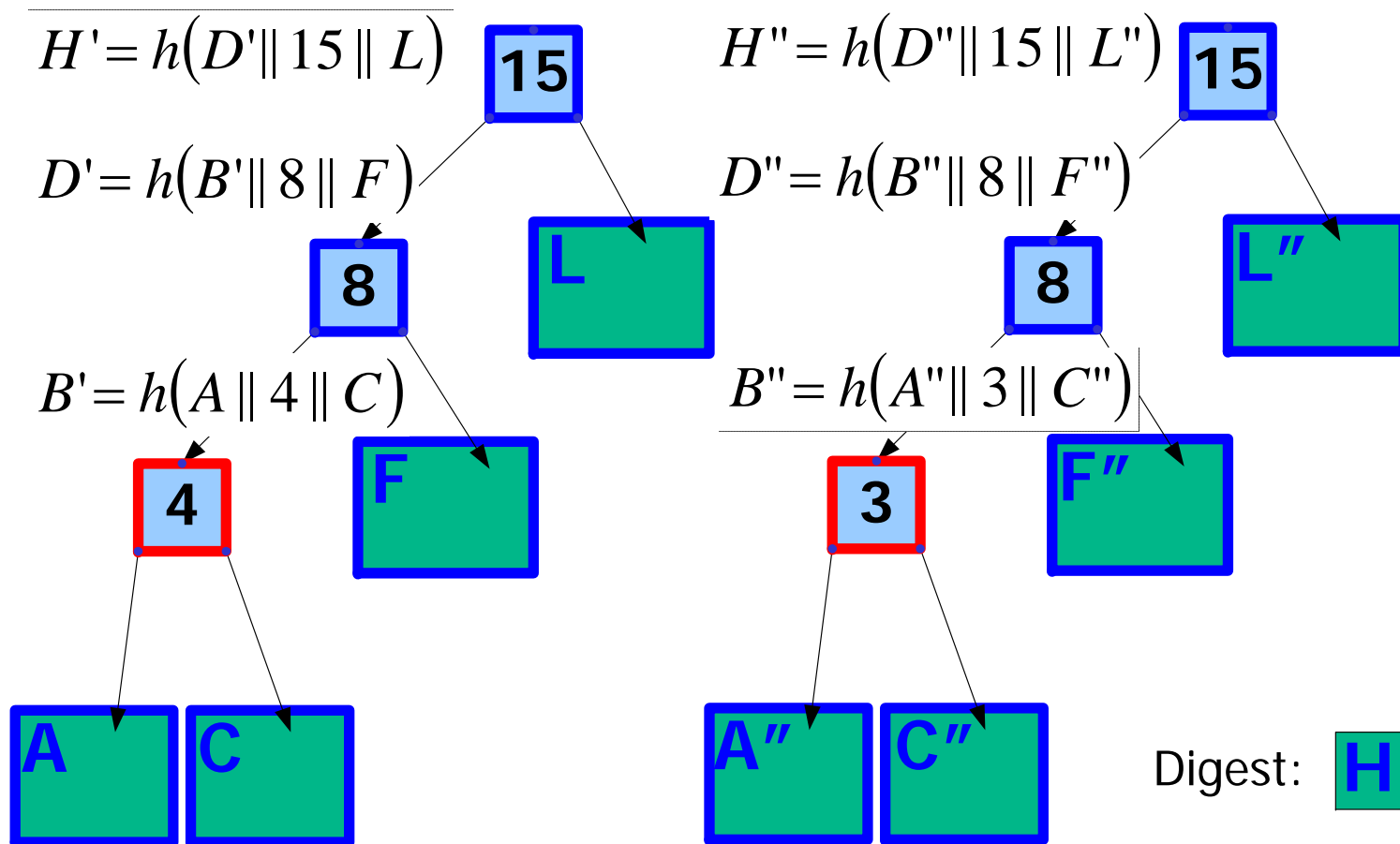


Existence Proof Construction





Cheating Is Infeasible



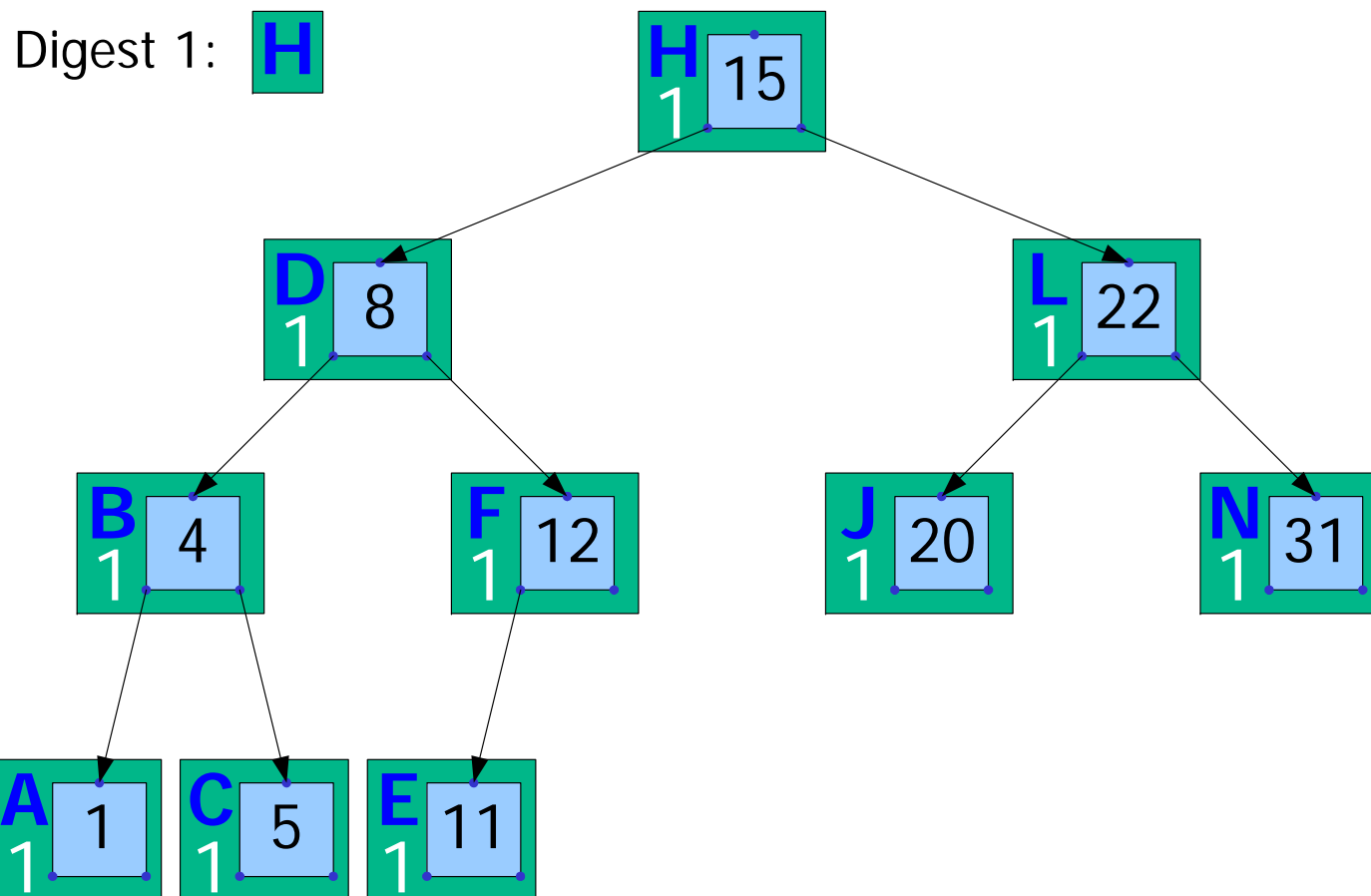


Authenticating Evolution

- Not only necessary to get a digest of an entire data collection
- Must also get a digest characterizing "deltas" from one version to another
- As before, the maintainer must be unable to cheat
 - By reordering deltas
 - By "forgetting" deltas
 - By modifying deltas

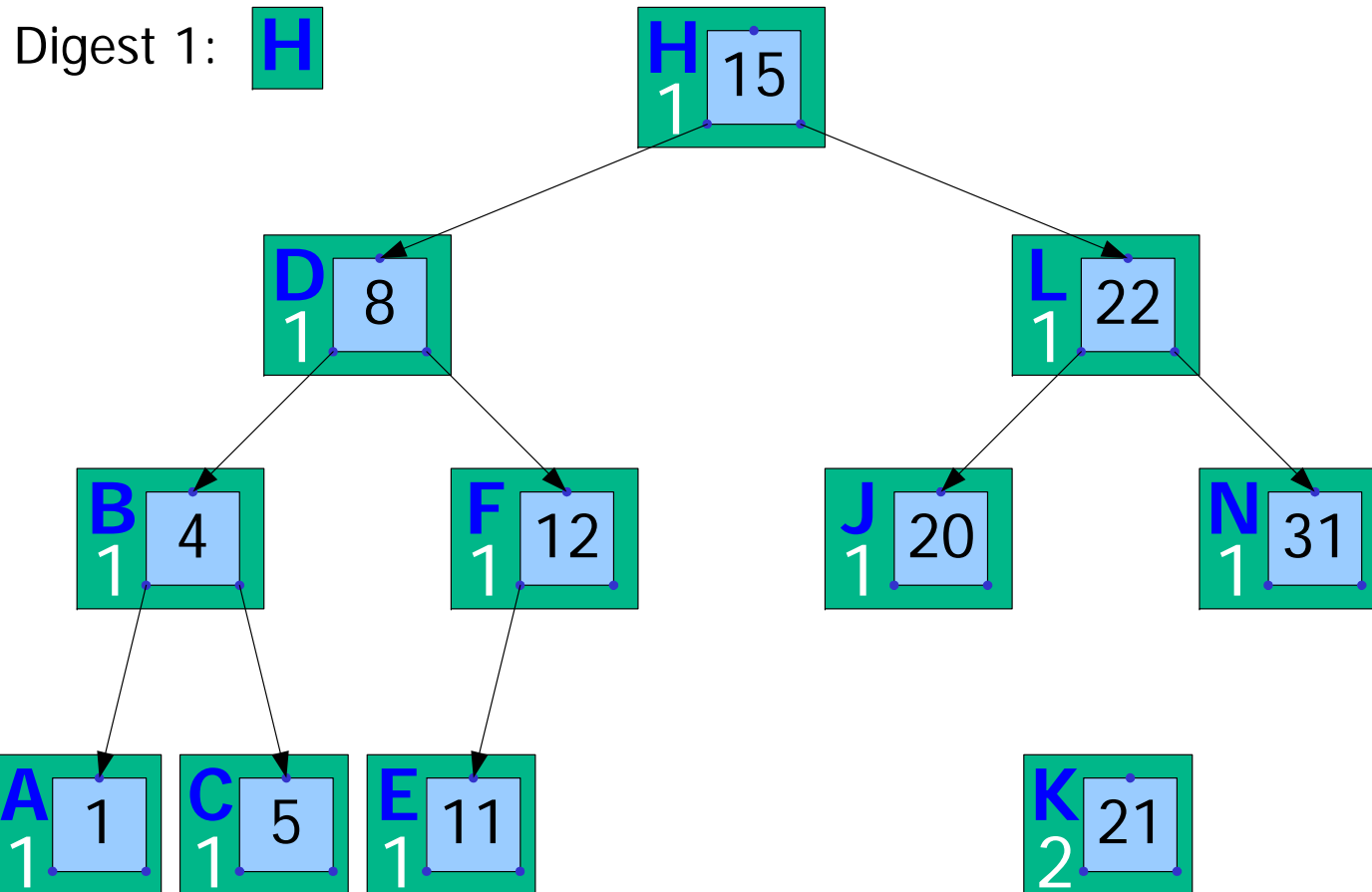


Persistent Authenticated Search Trees



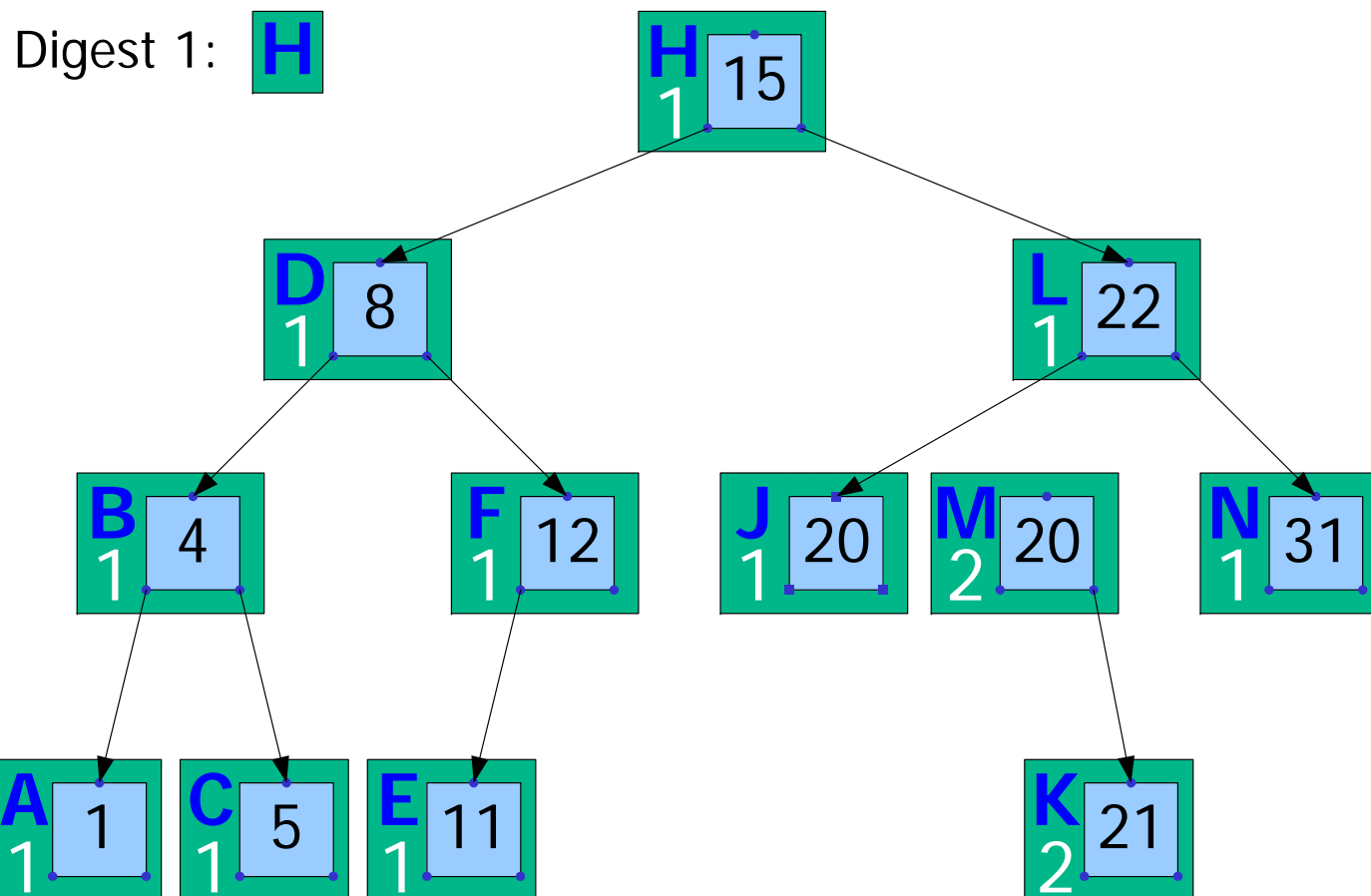


Persistent Authenticated Search Trees



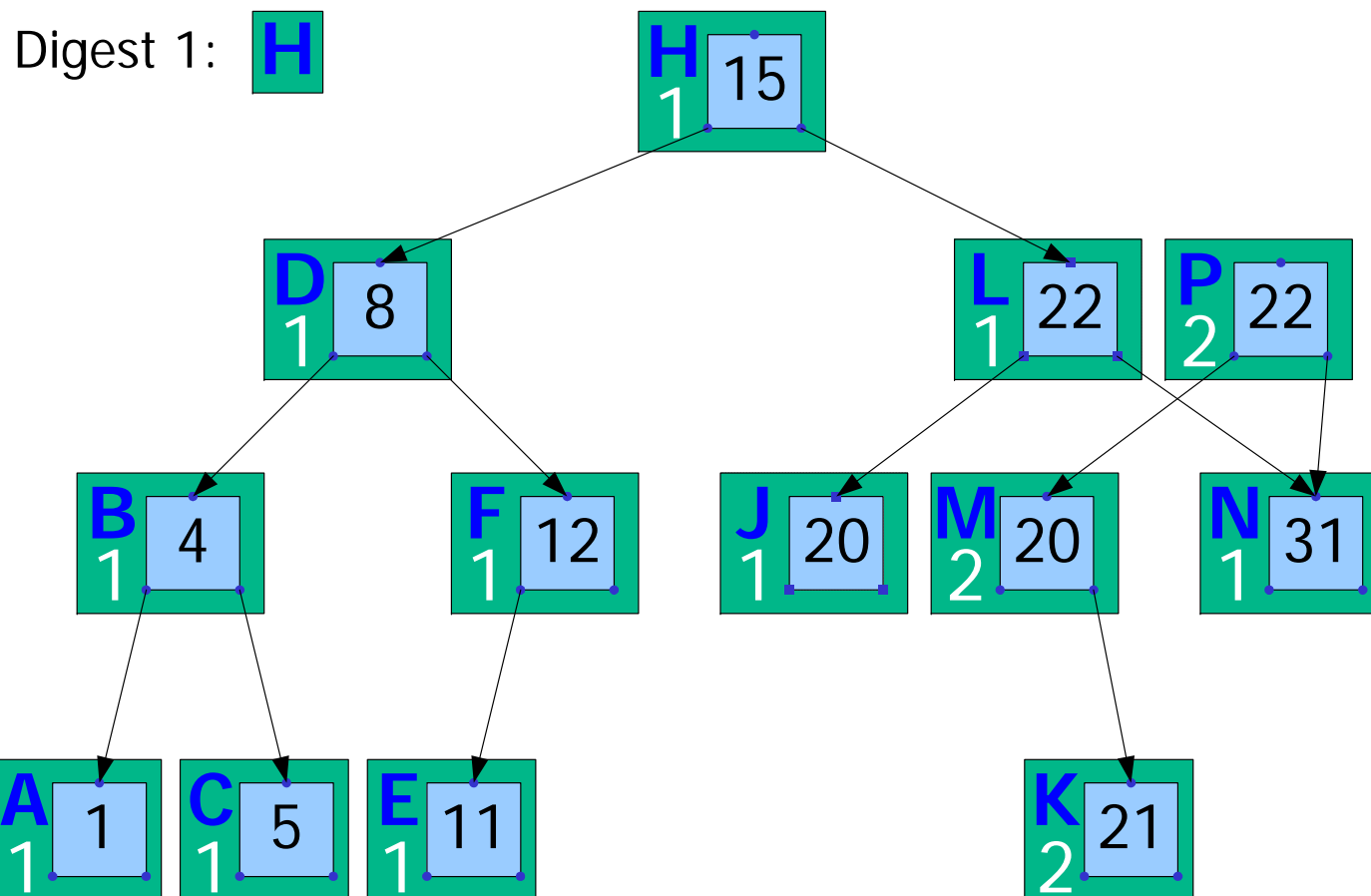


Persistent Authenticated Search Trees



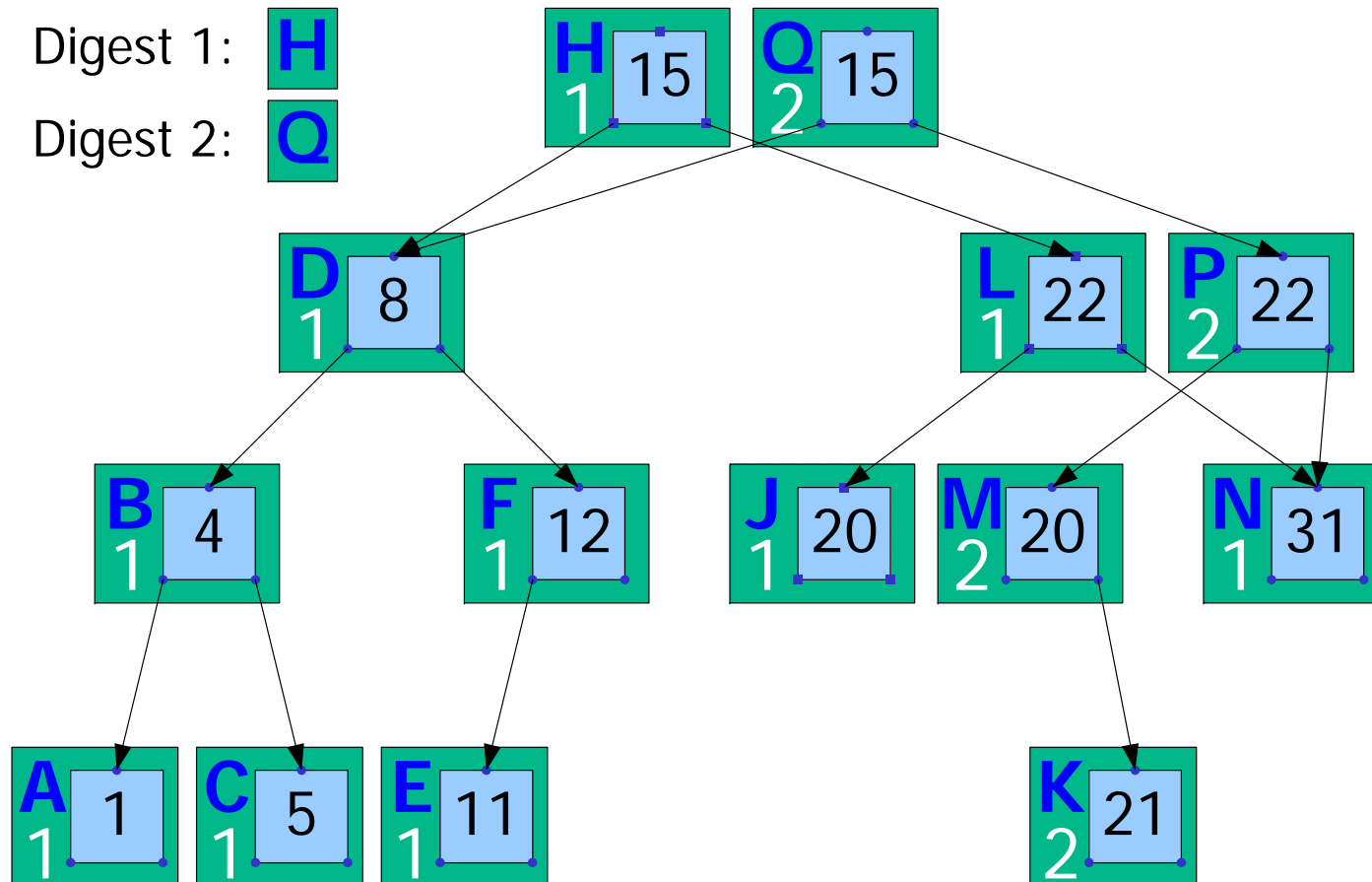


Persistent Authenticated Search Trees



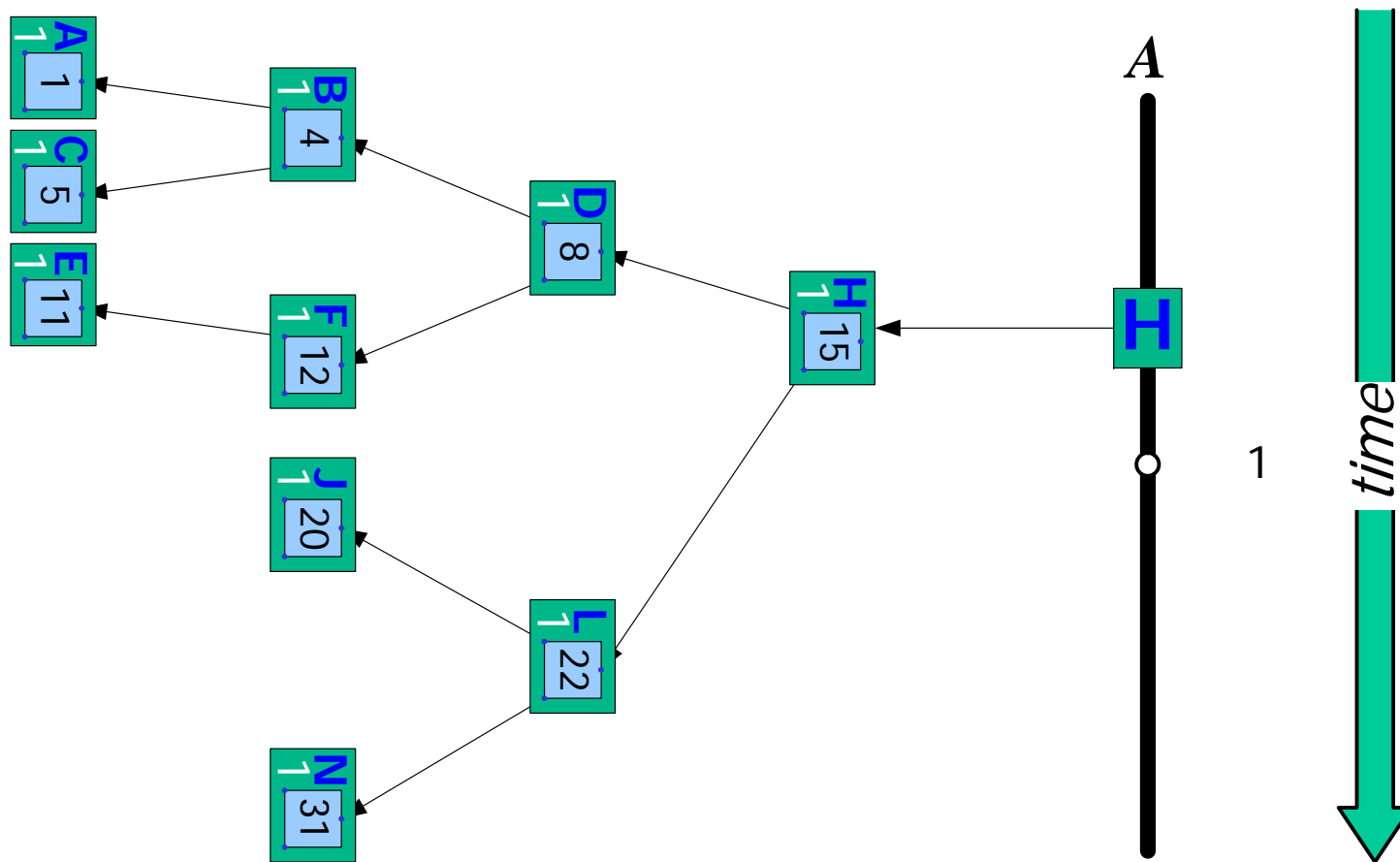


Persistent Authenticated Search Trees



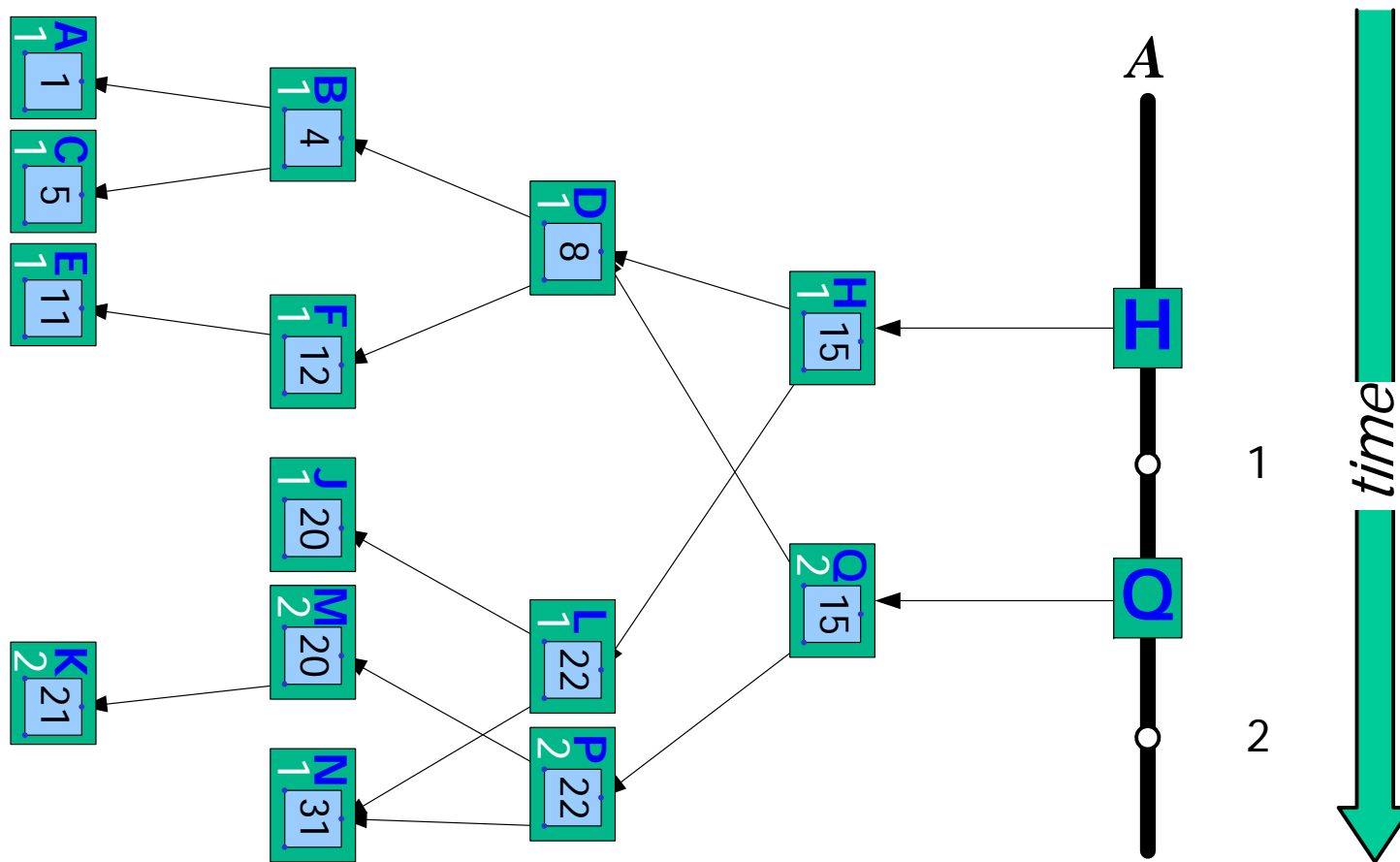


Committing Snapshots



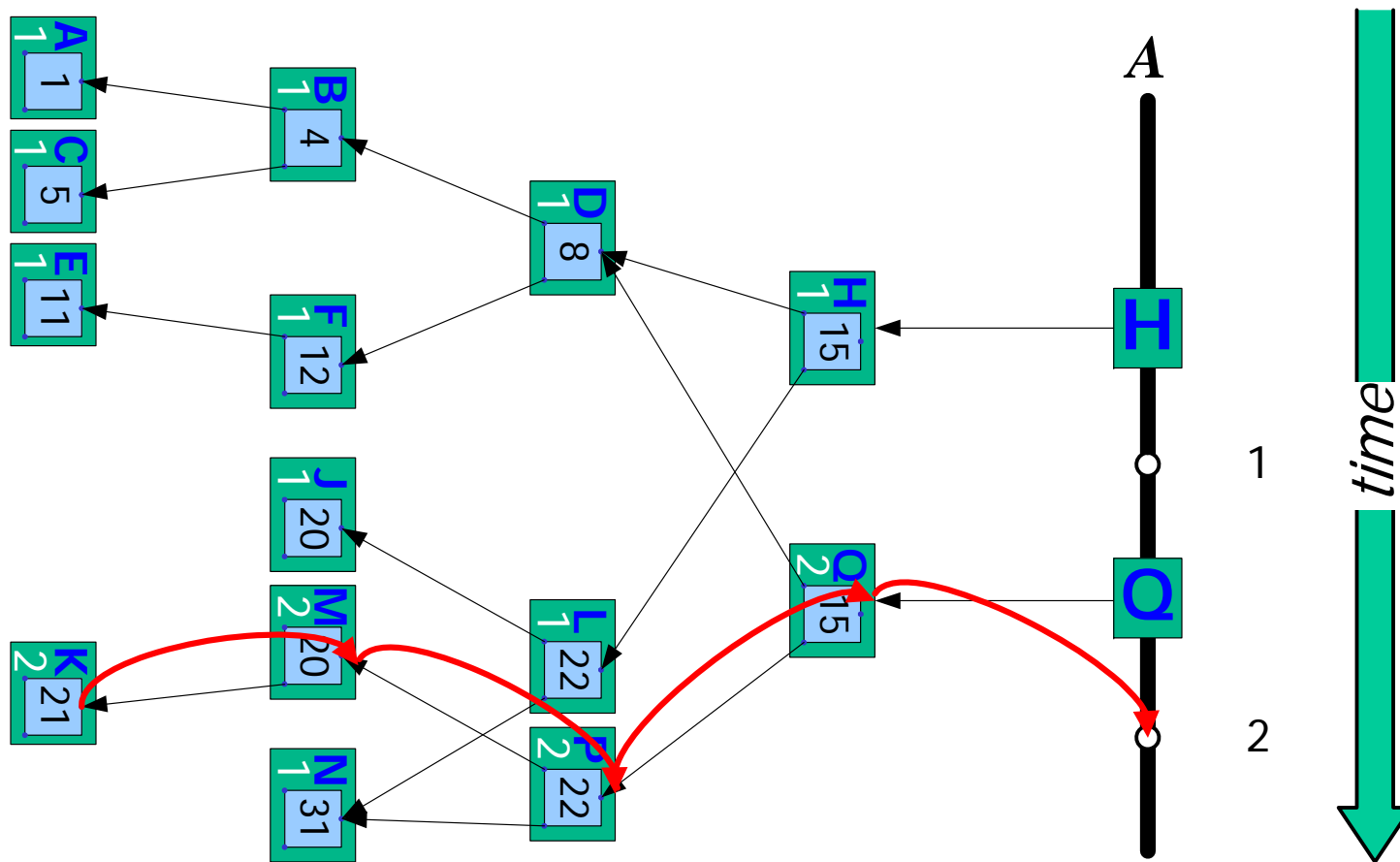


Committing Snapshots



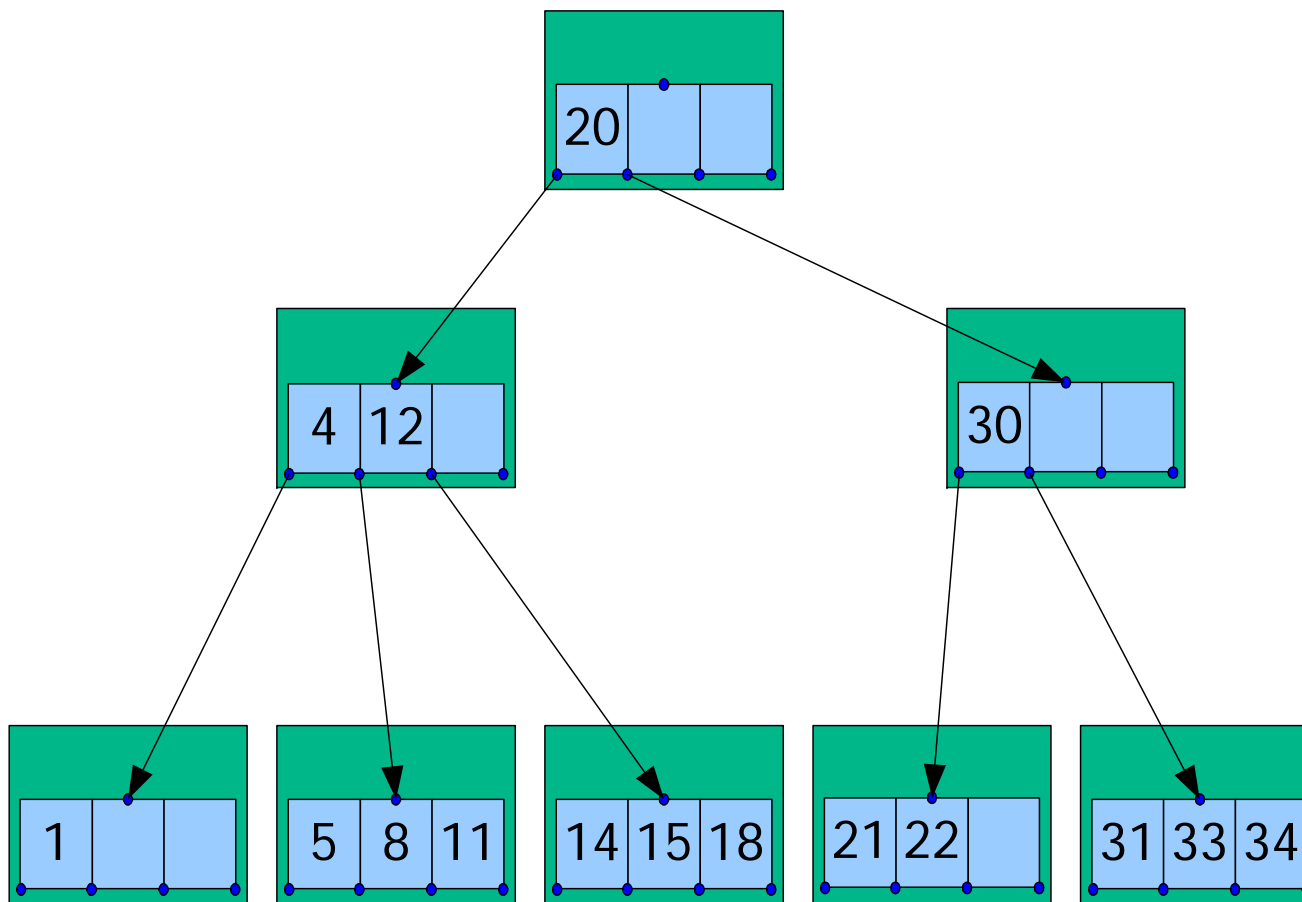


Receipt Generation



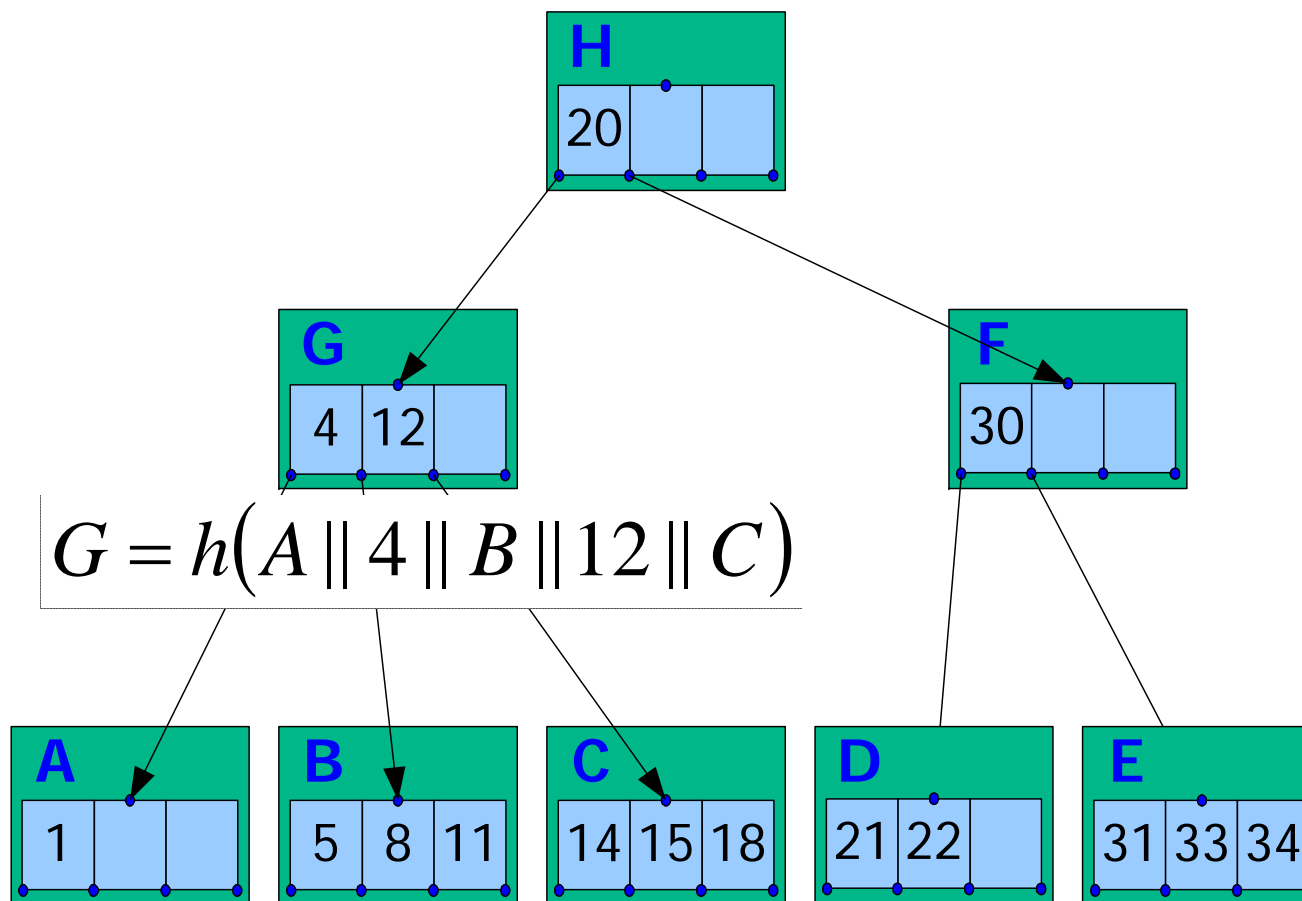


Storing Trees: B-Trees



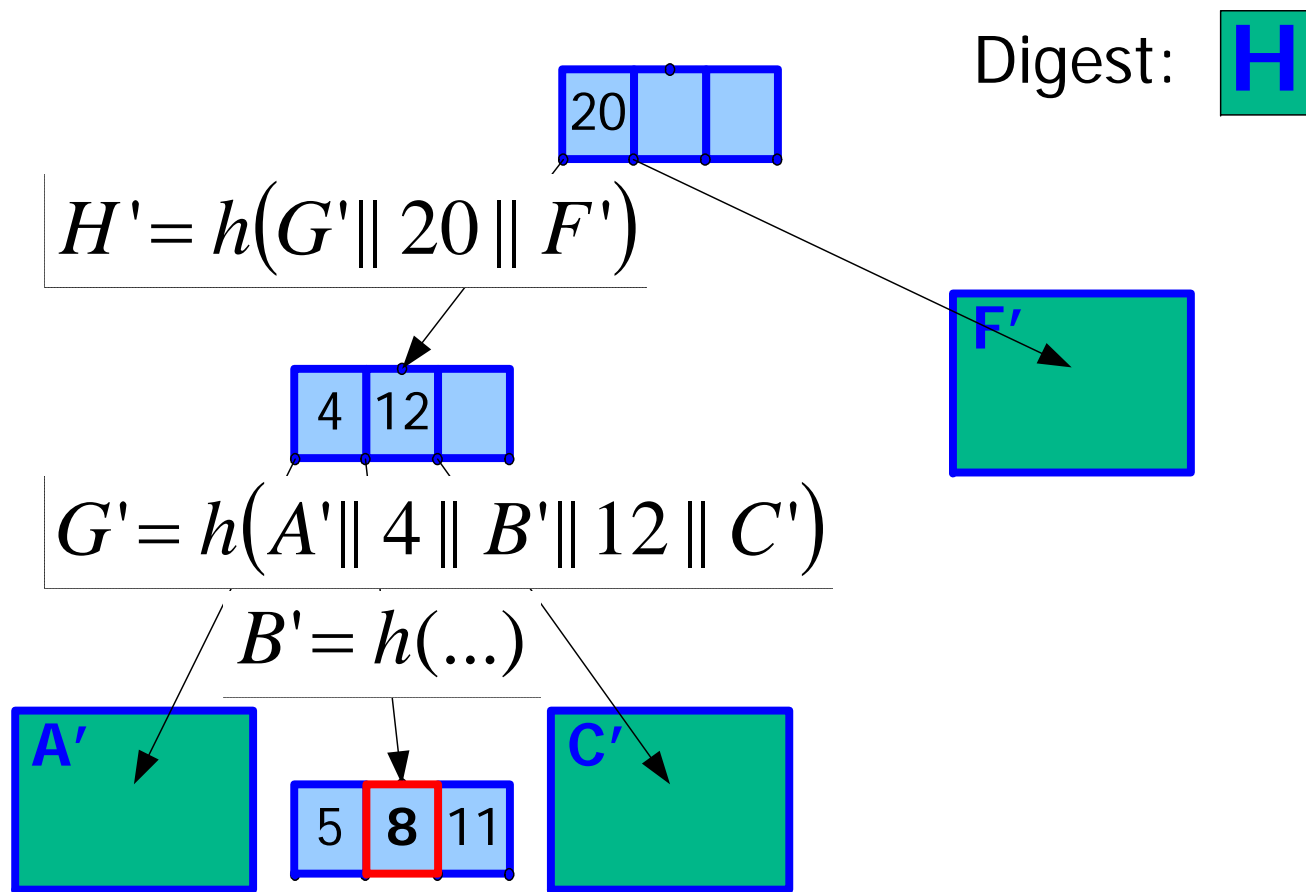


Hash B-Trees





Existence Proof Complications



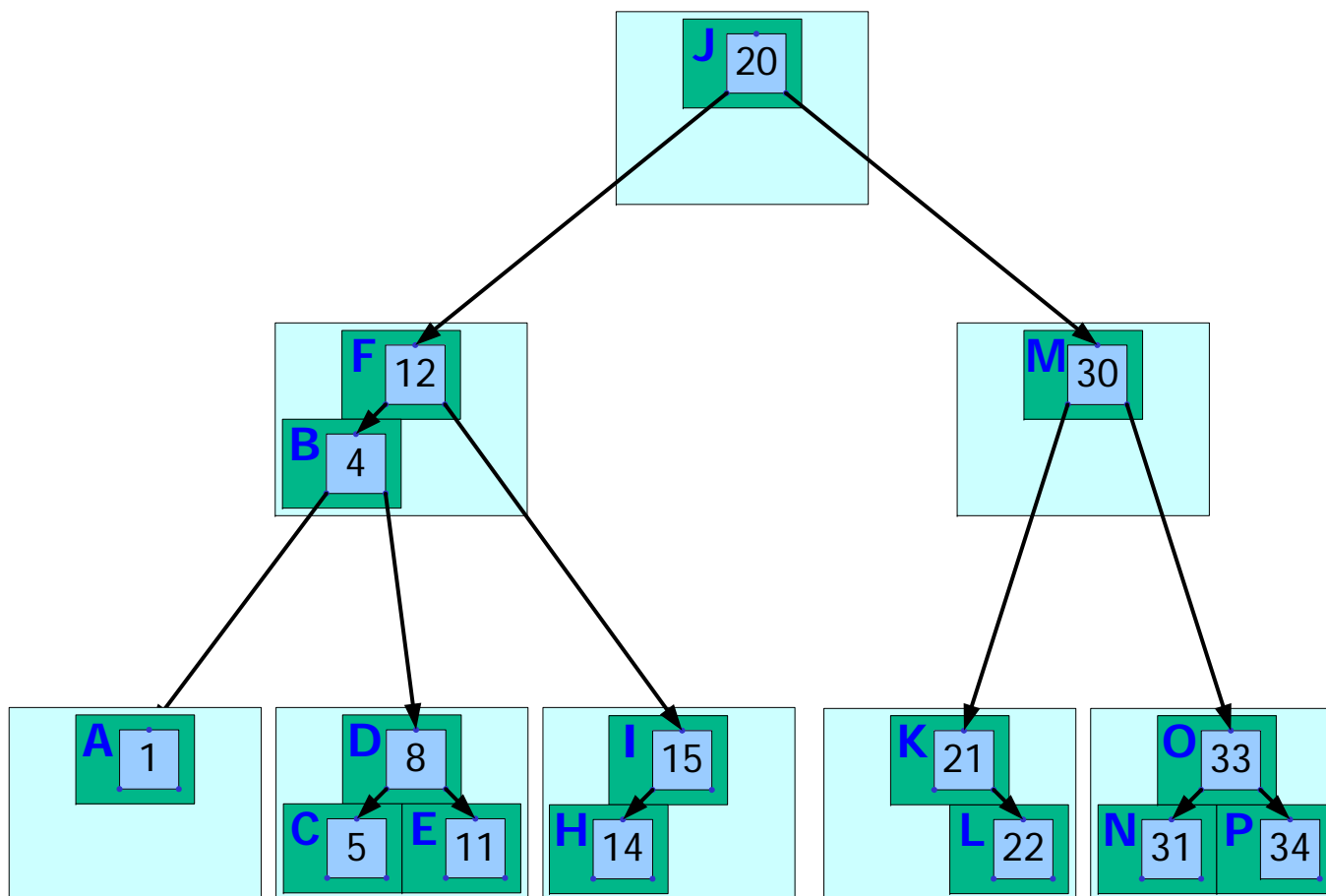


RBB-Trees

- A B-tree with embedded binary balanced trees
- B-tree
 - From the point of view of storage, i.e., clustering of data values in disk blocks
- Balanced binary tree (red-black tree)
 - From the point of view of data value structure, i.e., branching factor

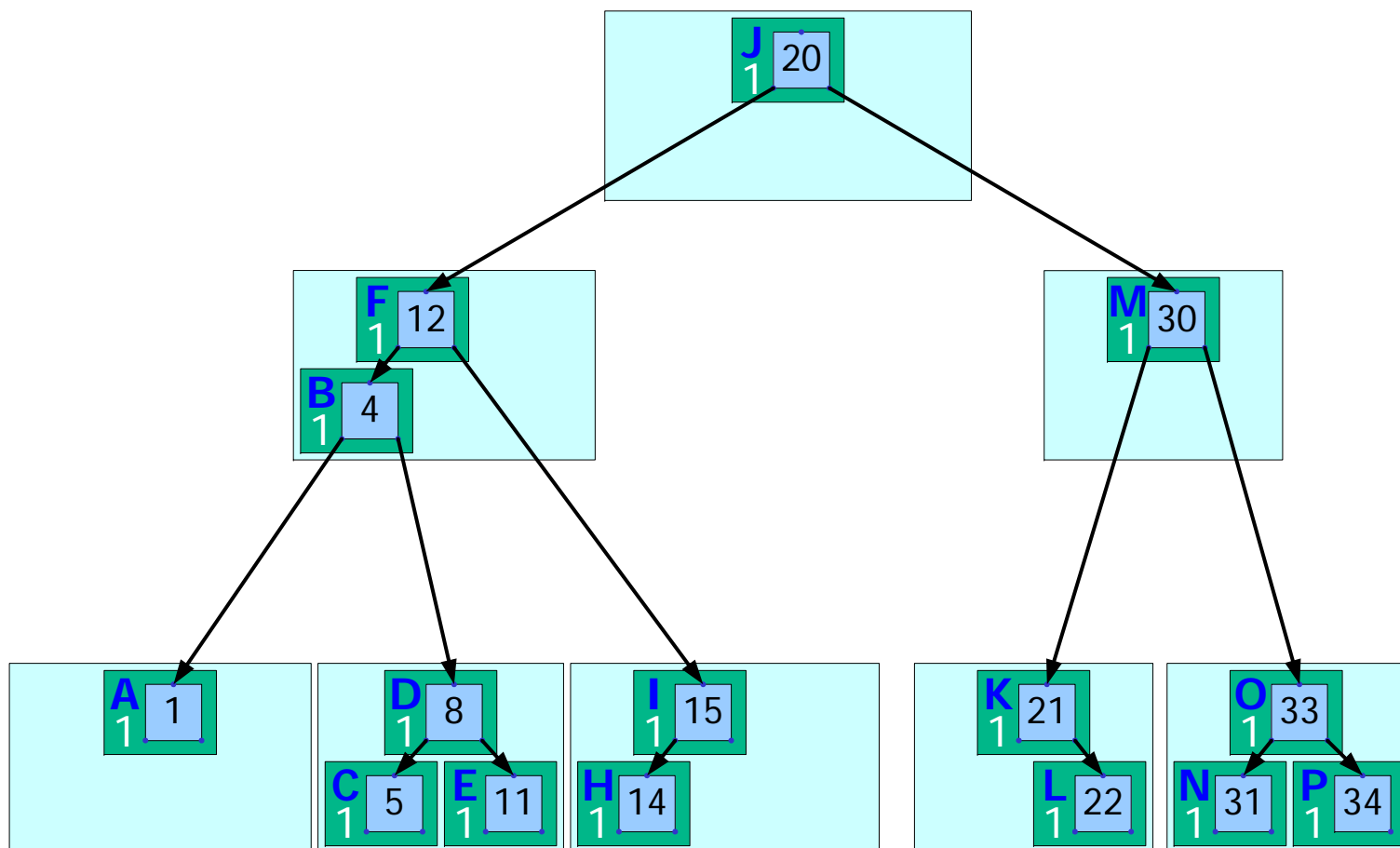


RBB-Trees



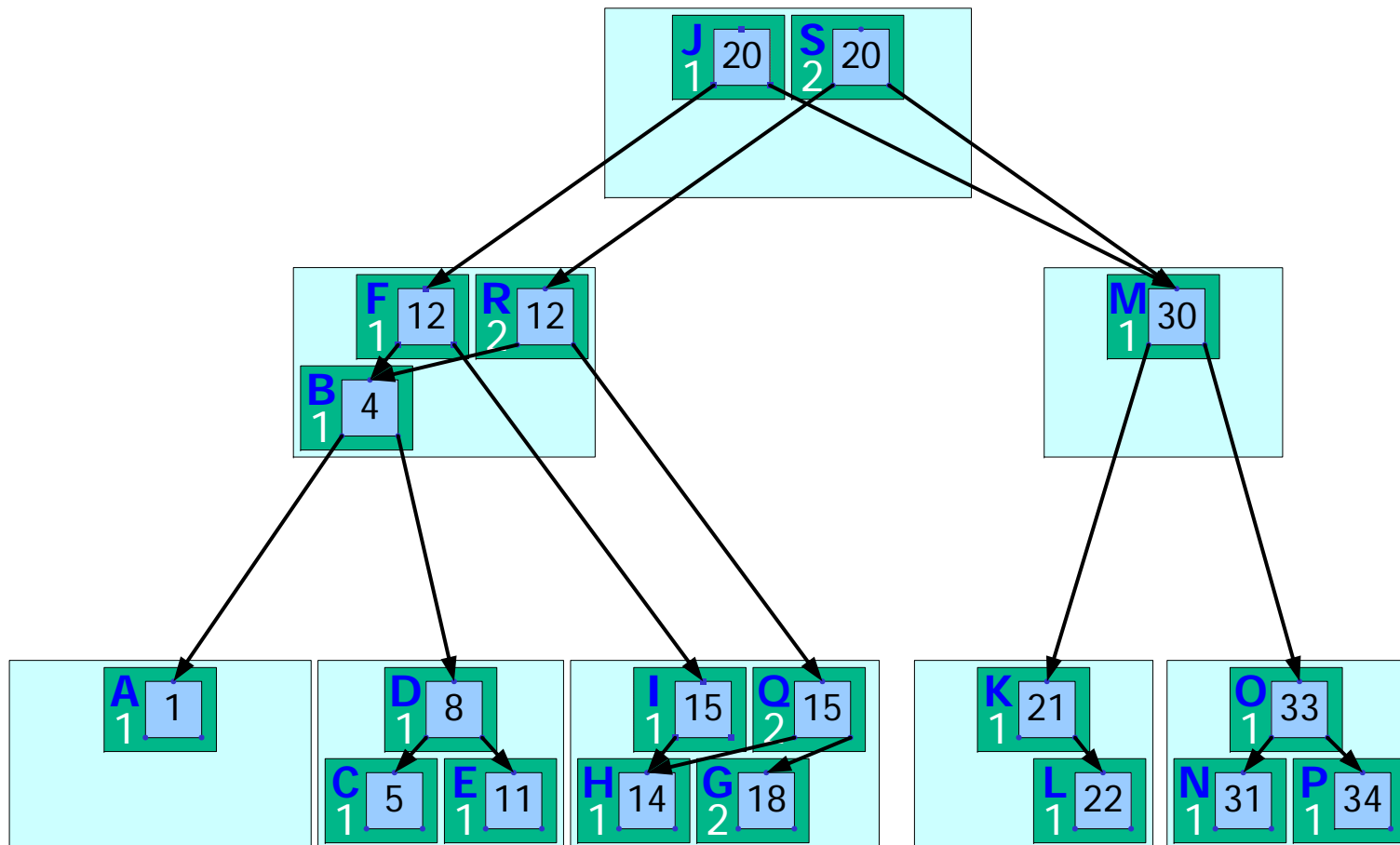


Persistent RBB-Trees



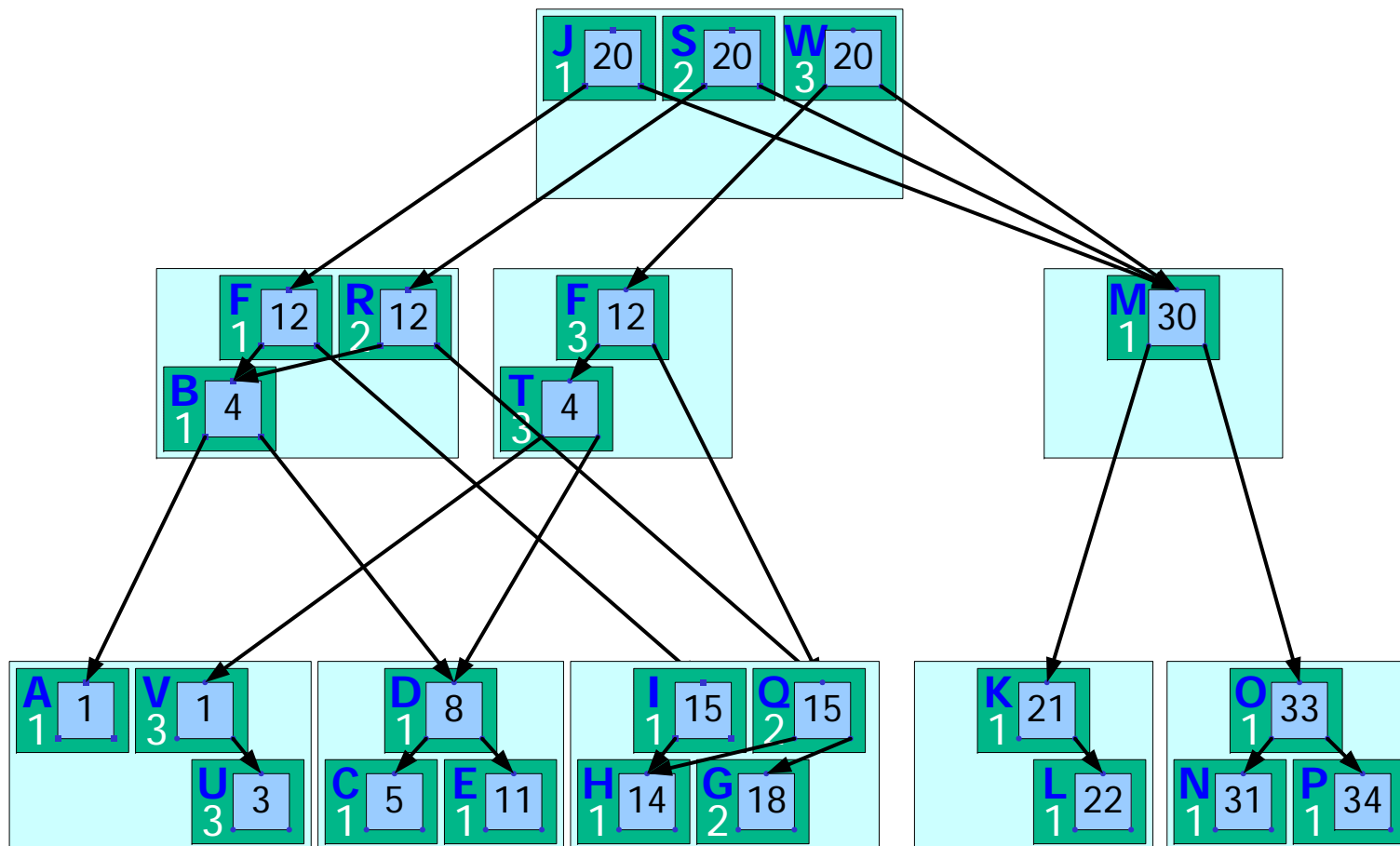


Persistent RBB-Trees





Persistent RBB-Trees





Properties of RBB-Trees

- Each tree version is an authenticated search tree
 - Proven to be an undeniable attester [Buldas et al]
- Version authenticators are placed in a PUA
 - A hash chain or our more efficient append-only skip lists
- Resulting structure is a PUA



Timeweave Performance

Can we afford to run Timeweave?



Performance Summary

- Setting:
 - 1200 hosts
 - Time steps last 1 sec
 - Entanglement interval is 10 minutes
- Steady-state overhead
 - ~100ms on last year's PC (dual PIII at 1GHz)
 - Grows linearly with thread processing rate
 - i.e. same for 2400 hosts, 20-minute entanglement
 - ~420GB of storage per year
- Java 1.3 & BerkeleyDB implementation



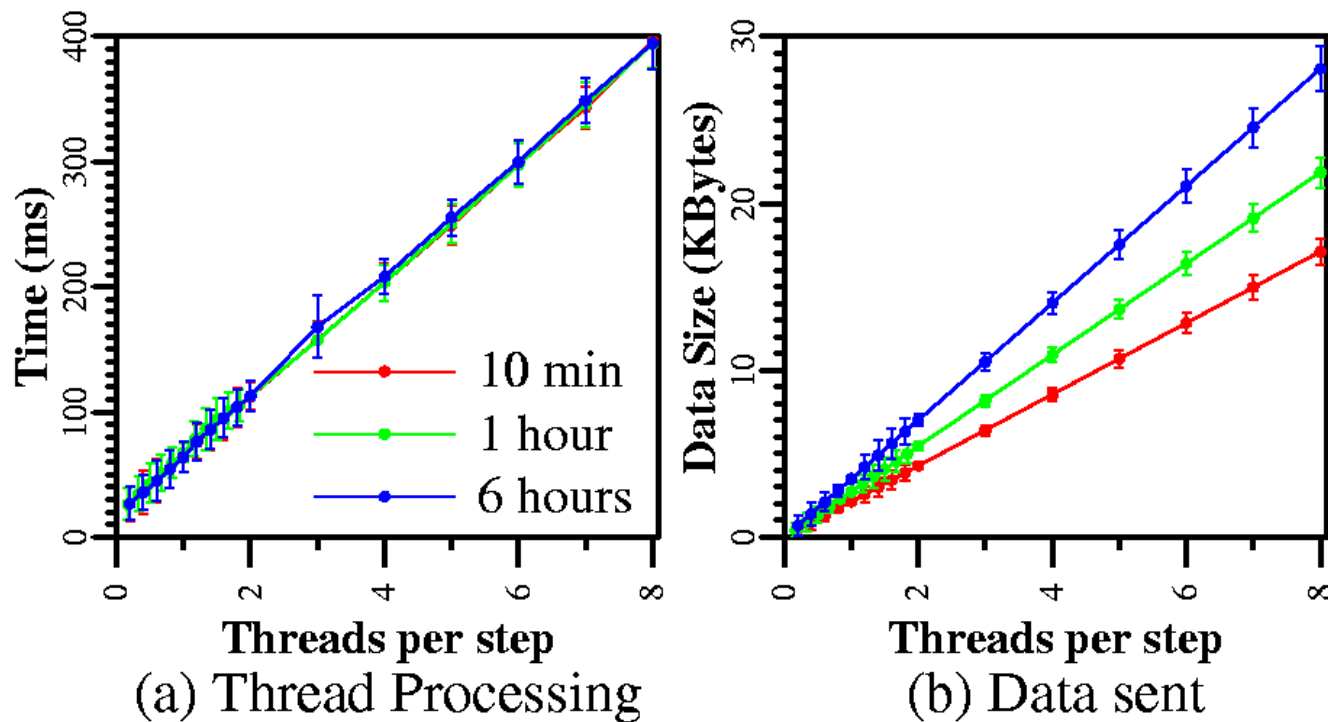
Performance

- How large a problem can we solve before the solution becomes unmanageable?
- Free variables
 - Size of entangled set (group size)
 - Entanglements per time step per node
- Metrics
 - Maintenance overhead
 - CPU & Disk I/O used by thread production, verification, location and storage
 - Communication overhead
 - Sending data rate



Entanglement Load

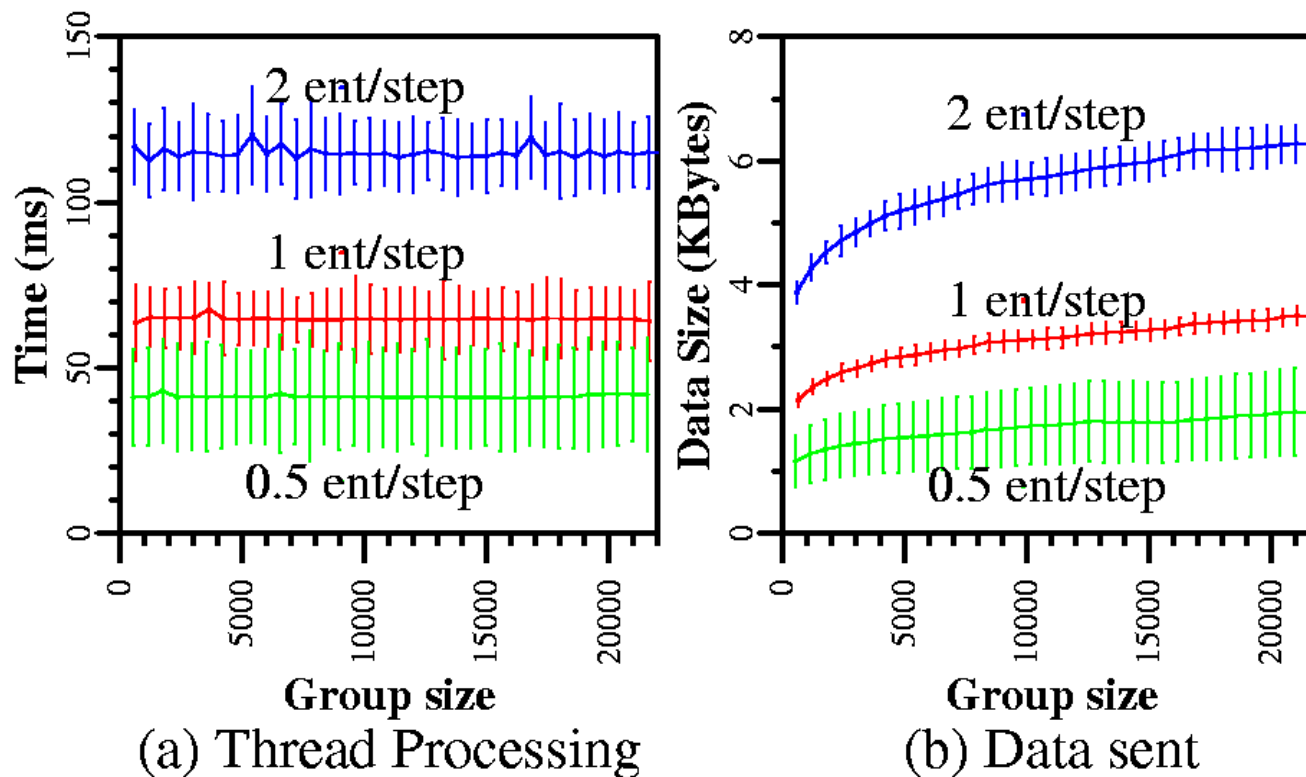
- 10 min, 1 h and 6 h entanglement intervals





Service Set Size

- 0.5, 1 and 2 entanglements per second





Distributed Survivable Time Stamping

An application of Timeweave



Secure Time Stamping

- Notarization for digital documents
- A trusted *Time Stamping Service* (TSS) maintains a history of document digests submitted to it
- A verifier can request proof of when a document was submitted, or a precedence proof from one document to another



Woes Of Centralization

- Single-point-of-failure problems
 - If the TSS goes out of business, I can no longer use my time stamps issued there
 - If the TSS becomes corrupt, no service I've received in the past can be trusted
- Interoperability
 - Not everyone trusts the same "trusted" service
 - If I want to give a convincing time stamp to someone who doesn't trust my service, I'm stuck



The Ticker Time Stamper

- A traditional TSS run on top of Timeweave
- Time stamps are amplified with a Timeweave proof of precedence
- Many TSSes belong in the same Timeweave entanglement group
- Benefits
 - Time stamps from different TSSes are comparable
 - Time stamps from one TSS may be mapped to other TSSes for survivability and fault tolerance



Conclusion

What's old, what's new, what's
relevant, what's next?



Related Work We Use

- Logical Clocks [Lamport]
- Cryptographic hashing (SHA-1) [FIPS-180-1]
- Randomized skip lists [Pugh]
- B-trees, red-black trees [Bayer & McCreight]
- Authenticated Search Trees [Buldac et al]



Related Work

- Secure time stamping
 - Haber & Stornetta, Benaloh & de Mare, Buldas et al, Preneel et al, Ansper et al
- Authenticated data structures
 - Merkle, Naor & Nissim, Buldas et al, Goodrich & Tamassia, Anagnostopoulos
- Secure “histories”
 - Schneier & Kelsey, Shapiro et al, Spreitzer



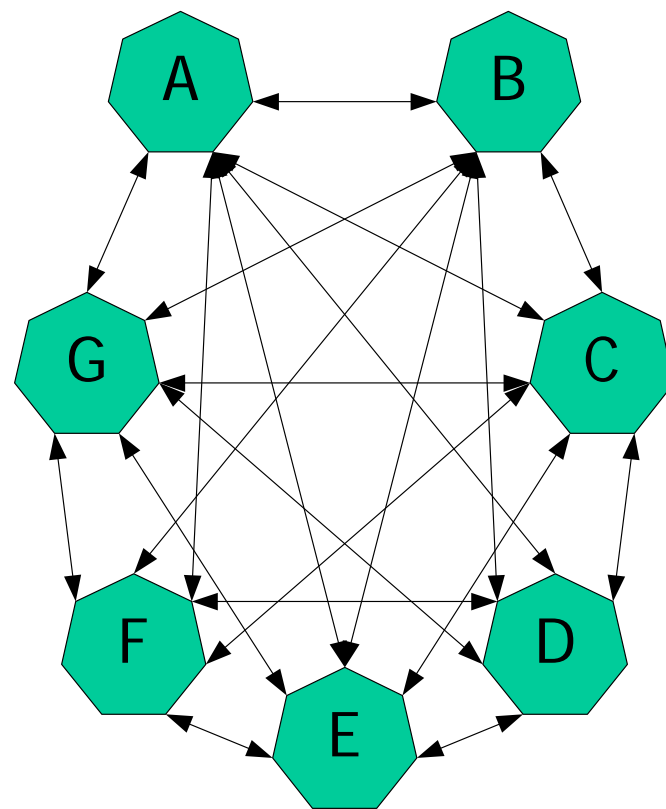
Future Work

- Scalability: How do I scale to millions of nodes?
 - Transitive entanglement
- Completeness: How do I capture *all* temporal precedences among committed events?
 - Limit duplicity through iterative information gathering
- More applications...
 - Historic integrity in reputation systems (in a DLib)
 - Accountable historic file systems (ConscienceFS)



Future Work – Scalability

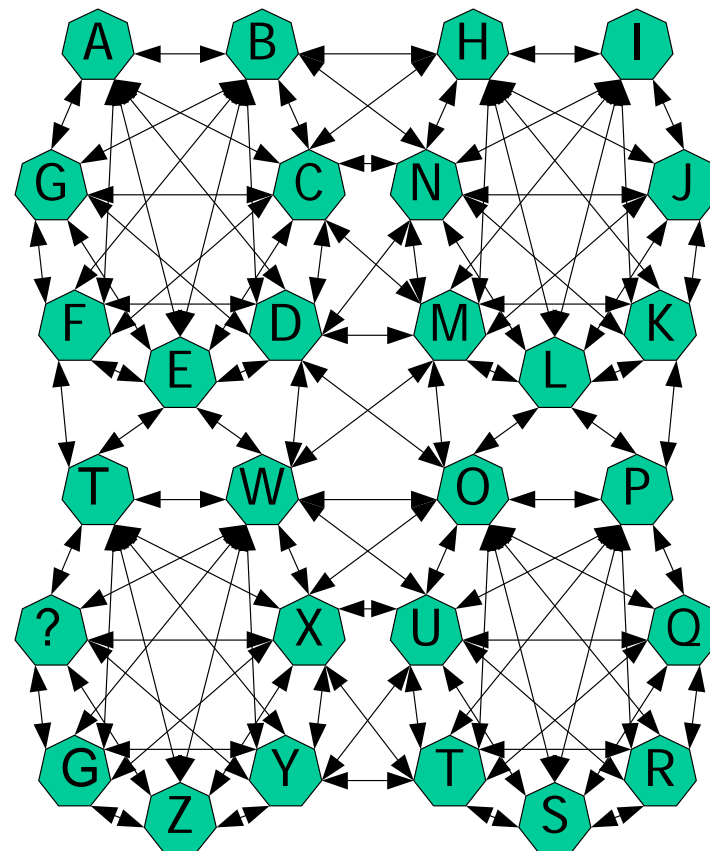
- Timeweave entangles all peers with all other peers
- This becomes expensive with very large group sizes





Future Work – Scalability

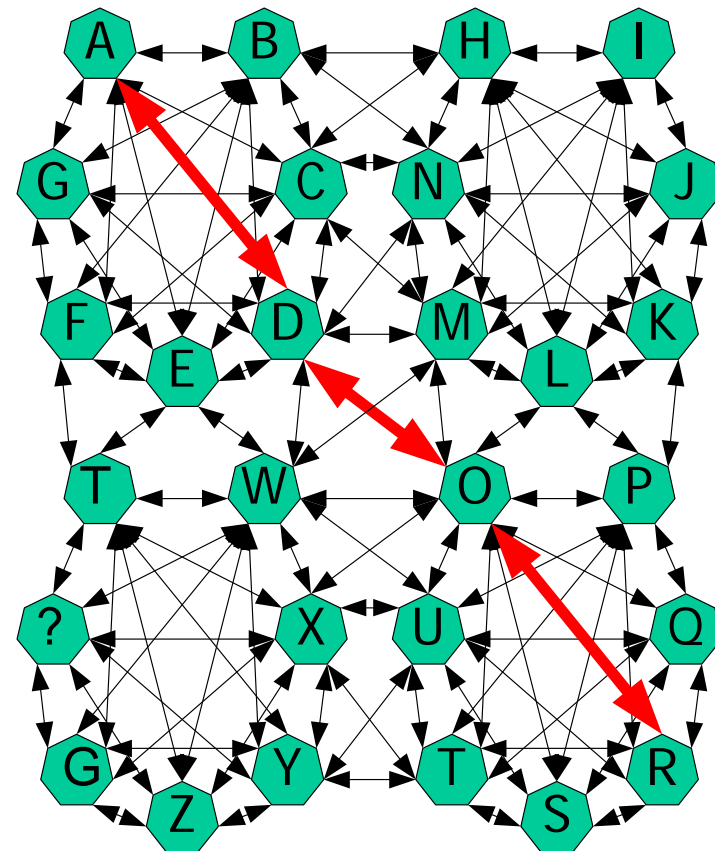
- Use, instead, less dense entanglement graphs





Future Work – Scalability

- Use, instead, less dense entanglement graphs
- Transitive entanglement is needed
- Now A must trust D to do its job
- Metric is temporal resolution loss





Thesis Contributions

- Architecture for maintaining a secure collective history of a distributed system
 - Implementation (Timeweave) [Sec 02]
- Persistent undeniable attestations
 - Append-only authenticated skip lists
 - Persistent authenticated search trees
- Applications
 - Distributed and survivable secure time stamping
 - Archival storage of digitally signed documents [FAST 02]



End of Talk

Audience Appreciation



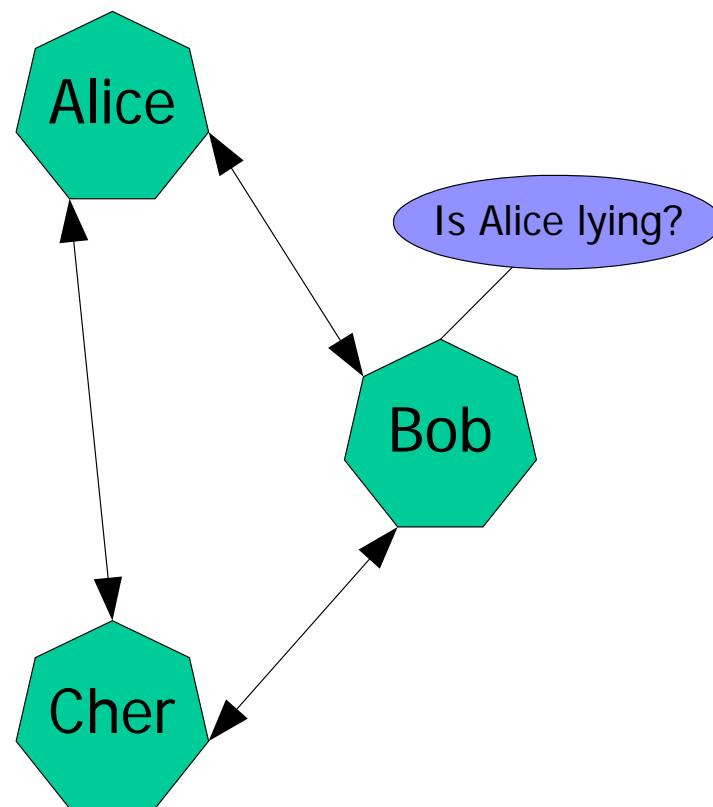
Timeweave Implementation

Data Structures:
Append-Only Authenticated
Skip Lists



Limitations – Transaction Size

- Timeweave currently works for 2-way transactions only
- Bob must know that Cher sees a consistent view of Alice with its own view
- Cher tells Bob "Alice told me A"



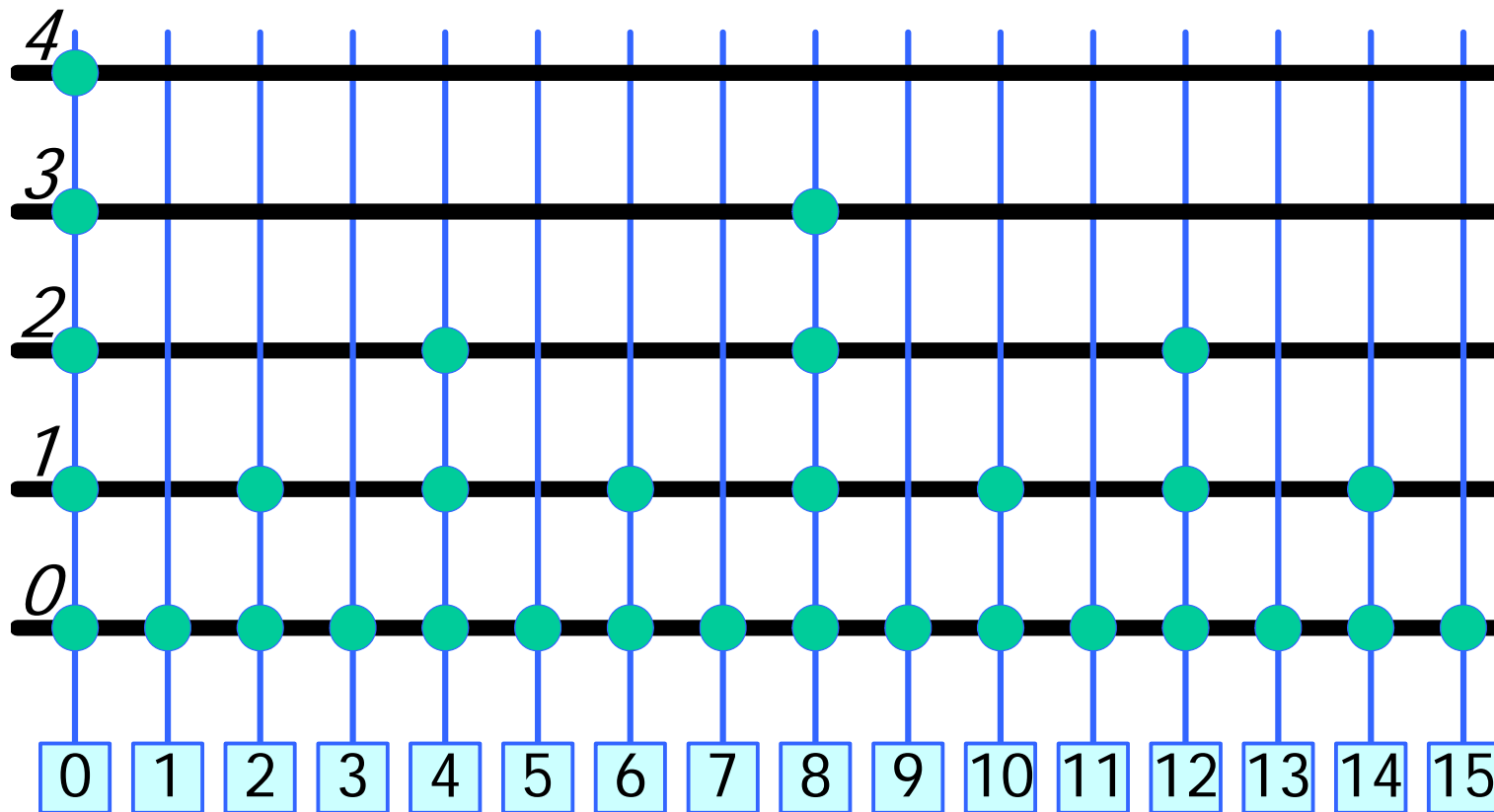


Append-Only Auth Skip Lists

- Multiple hash chains at different levels of detail
 - Level-0 contains all elements
 - Level-1 contains half the elements
 - Level-2 contains a quarter of the elements
 - Etc.
- i -th element participates in levels 0 through l
 - l is the highest power of 2 that divides i

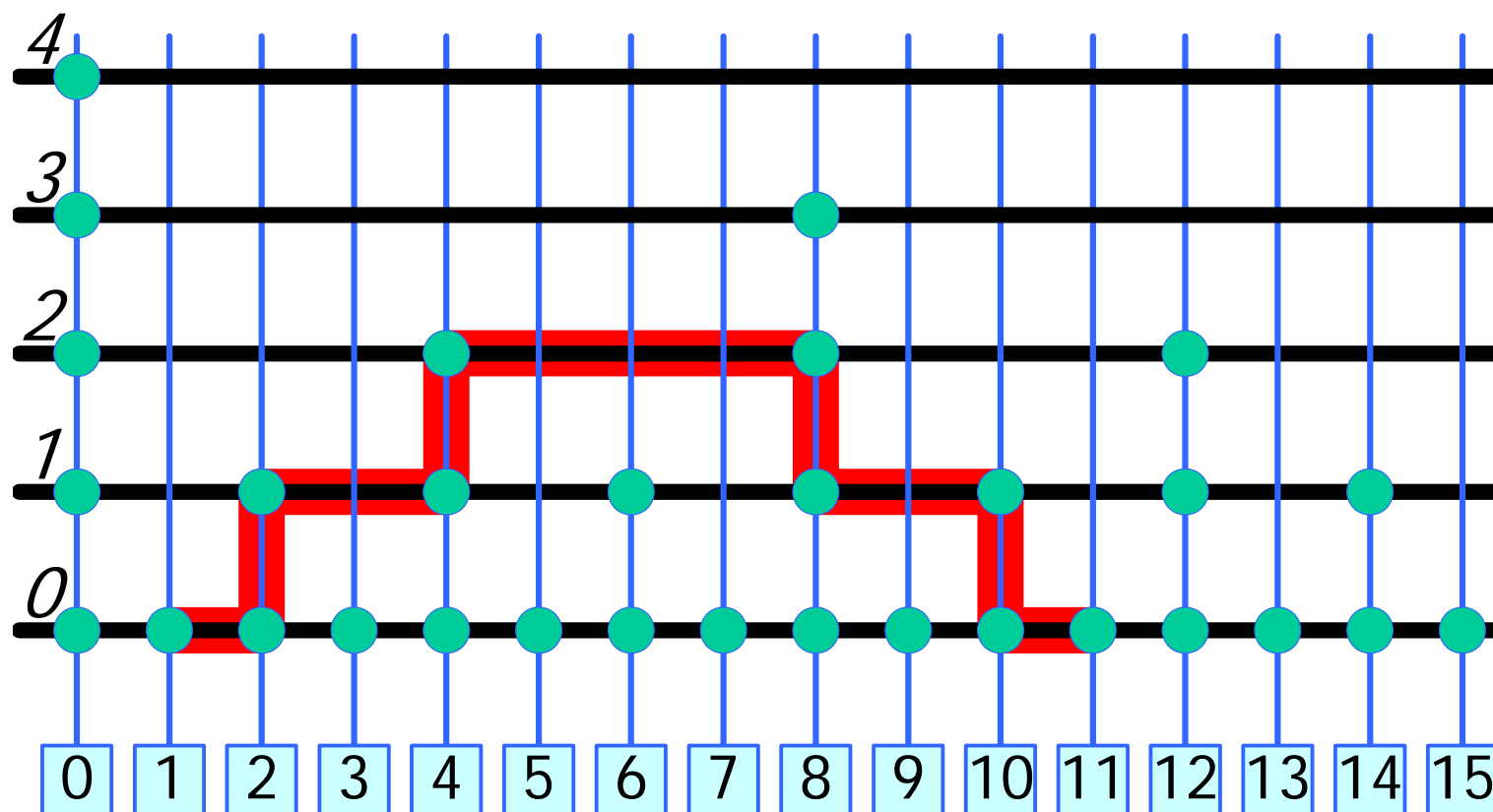


Append-Only Auth Skip Lists





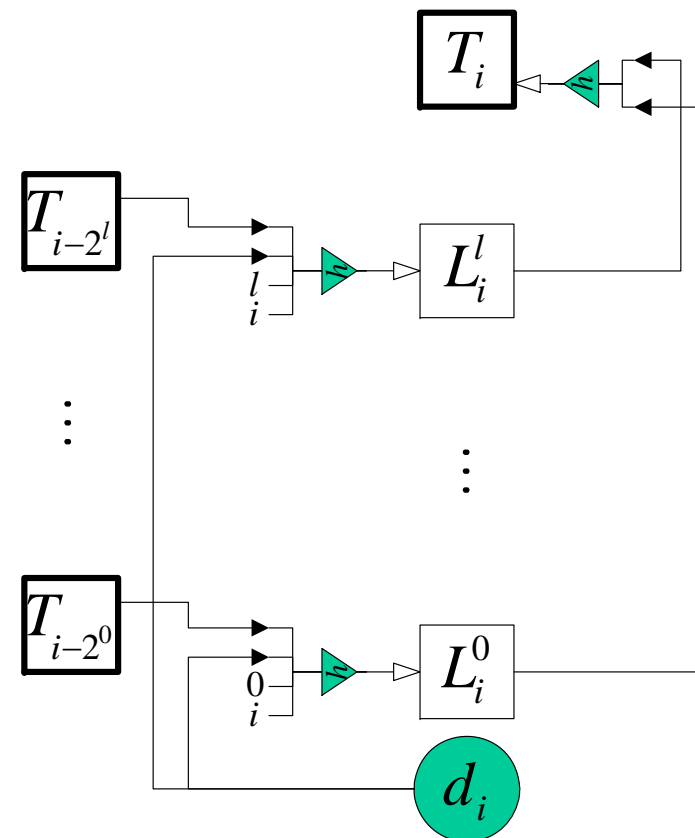
Skip List Traversal





Append-Only Auth Skip Lists

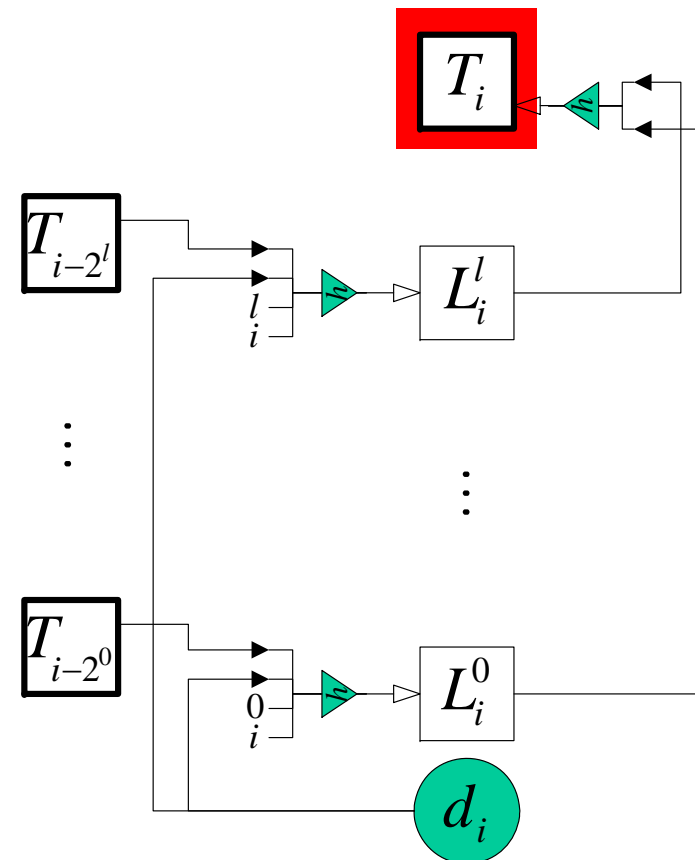
- At every level, an element has a *partial authenticator*
- Each partial authenticator is derived from the element and its predecessor at that level
- All partial authenticators are combined into the element's authenticator





Single-Hop Proof

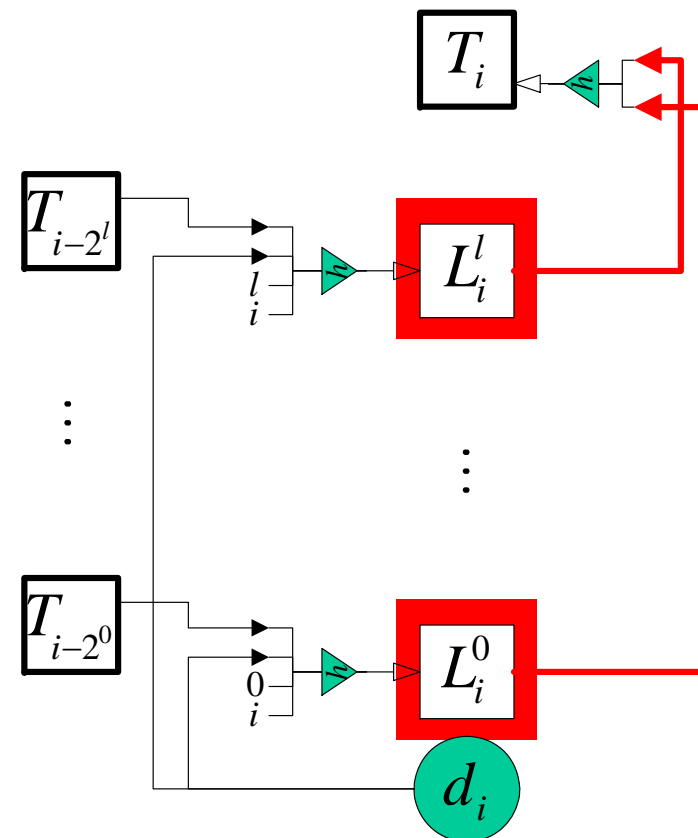
- Given an authenticator, prove that a particular datum is stored in the associated position





Single-Hop Proof

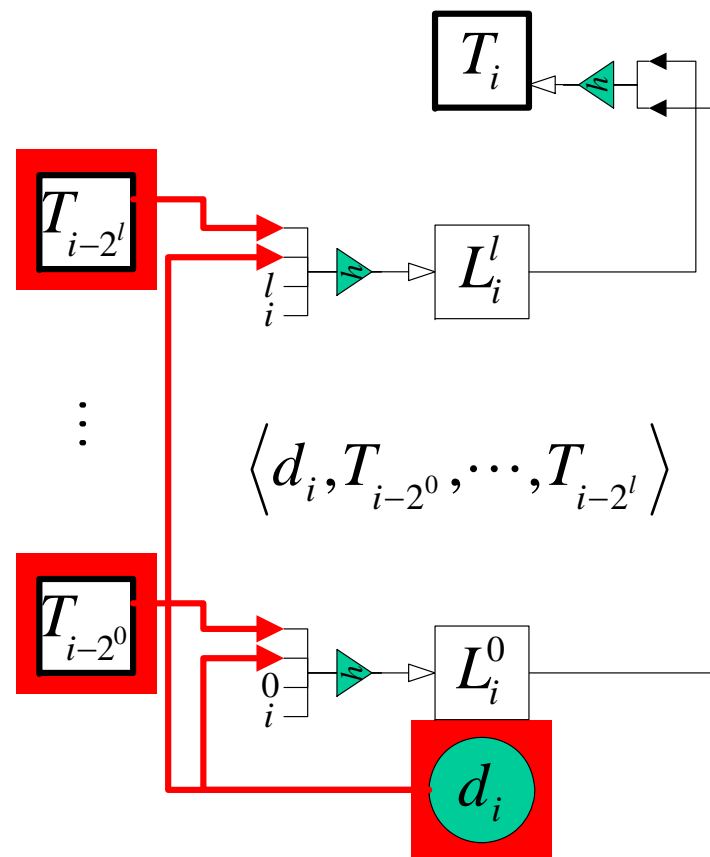
- Given an authenticator, prove that a particular datum is stored in the associated position





Single-Hop Proof

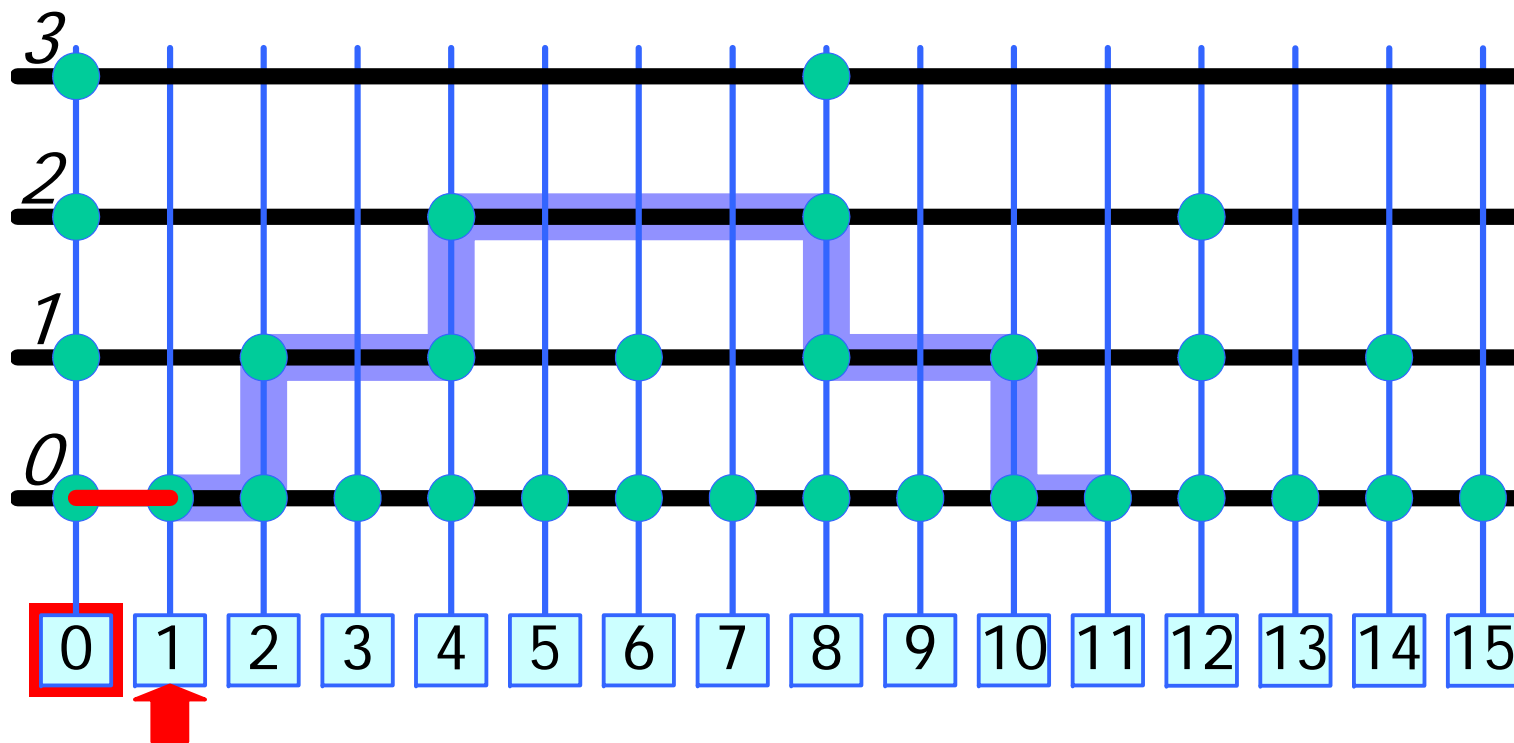
- Given an authenticator, prove that a particular datum is stored in the associated position
- I need all predecessors auths and the datum





Skip List Precedence Proofs

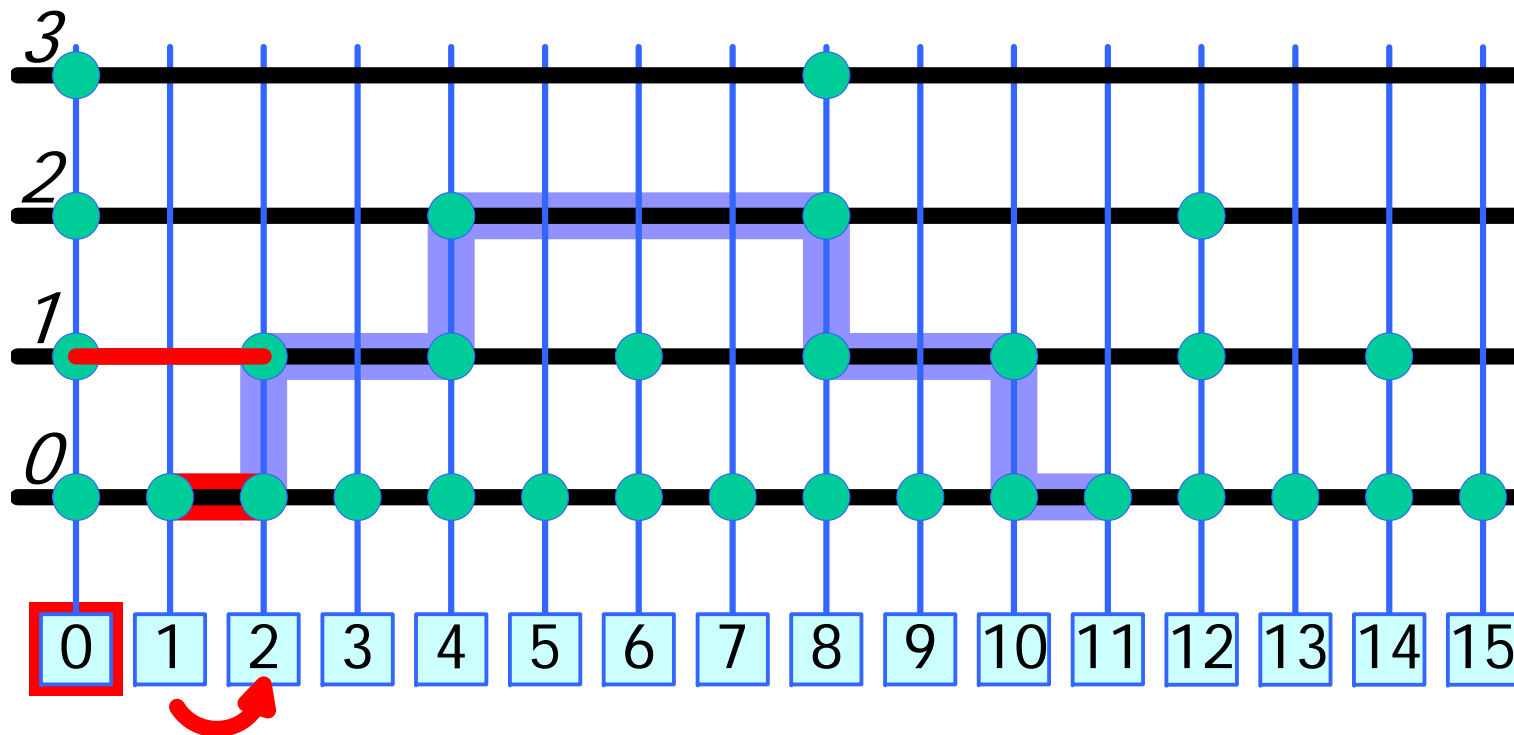
$$P_1^{11} = \langle d_1, T_0 \rangle$$





Skip List Precedence Proofs

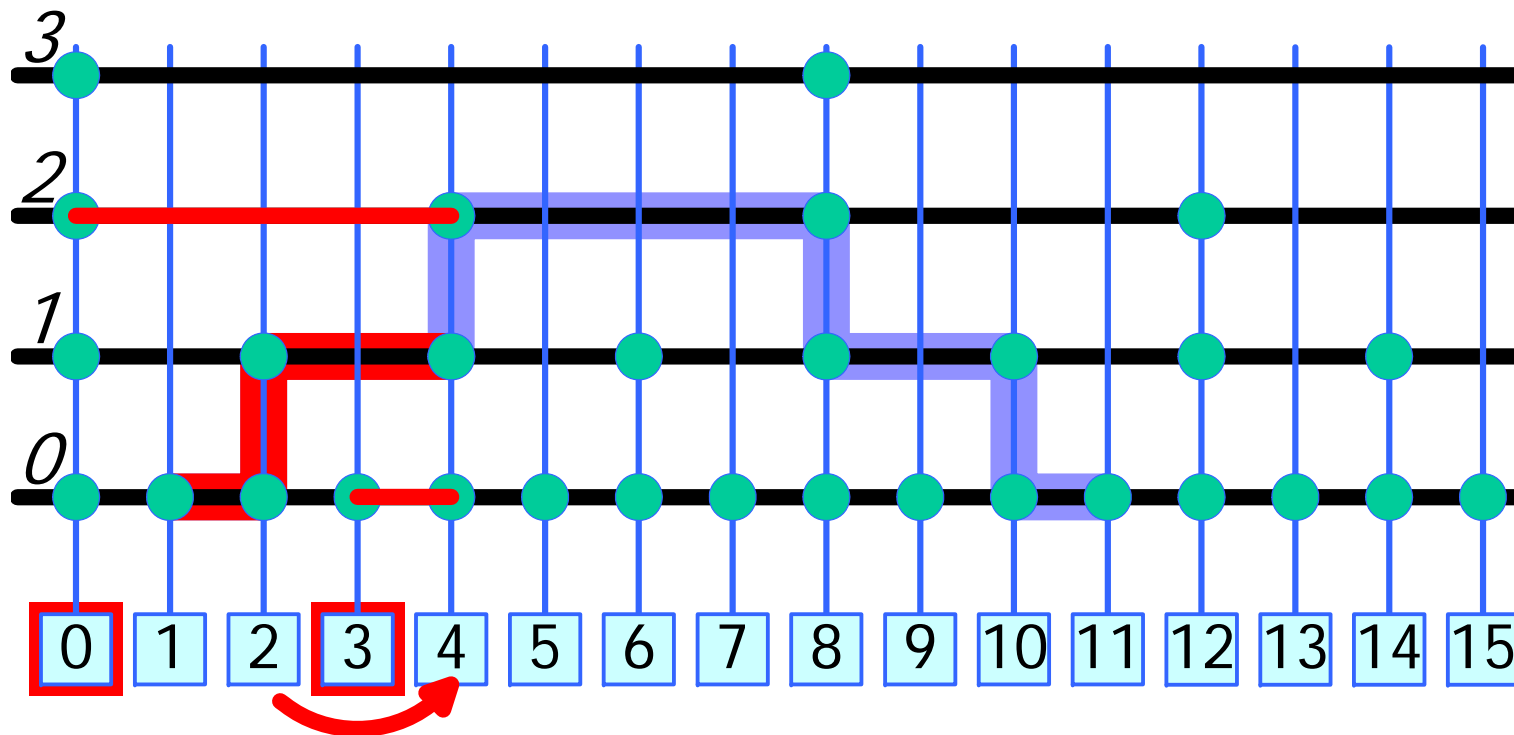
$$P_1^{11} = \langle d_1, T_0; d_2, T_0 \rangle$$





Skip List Precedence Proofs

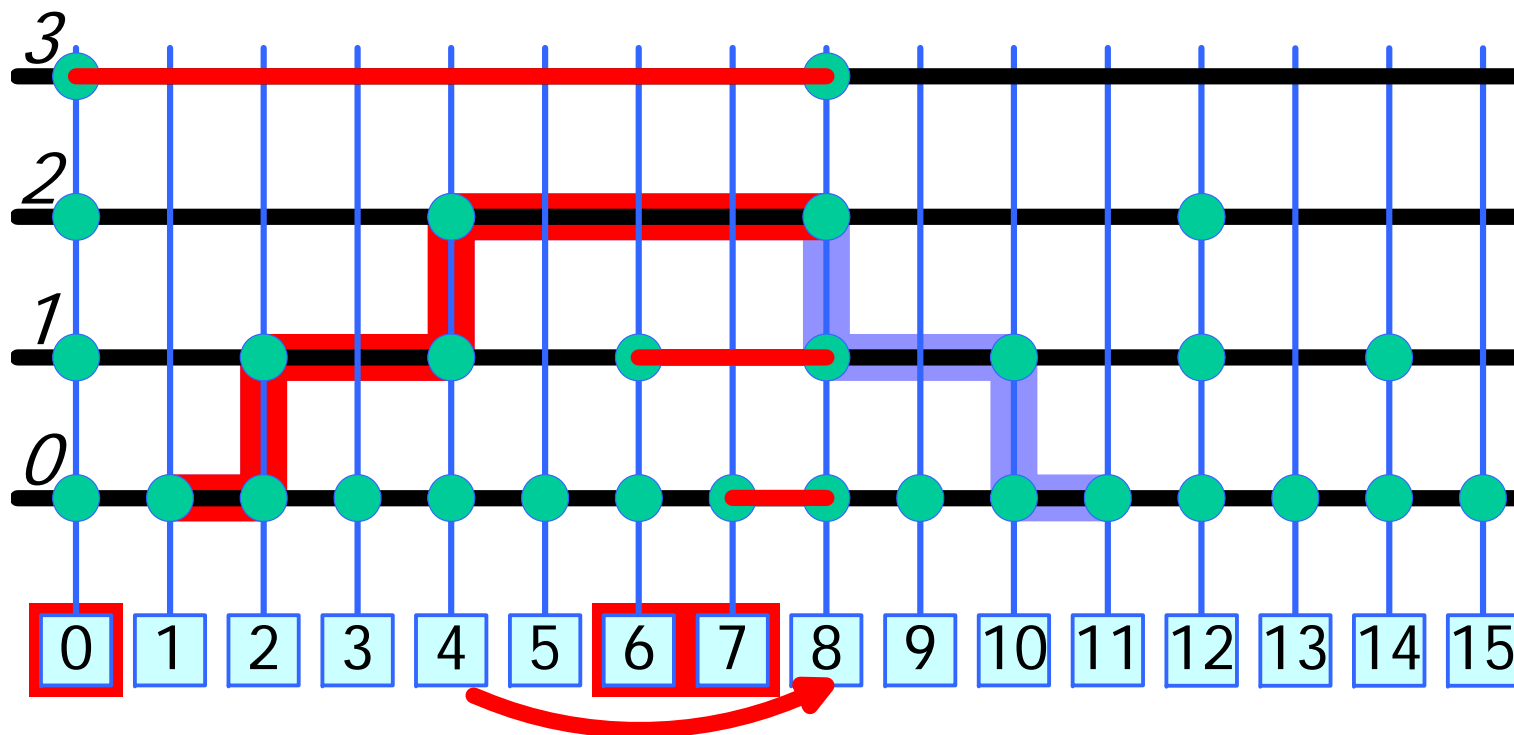
$$P_1^{11} = \langle d_1, T_0; d_2, T_0; d_4, T_0, T_3 \rangle$$





Skip List Precedence Proofs

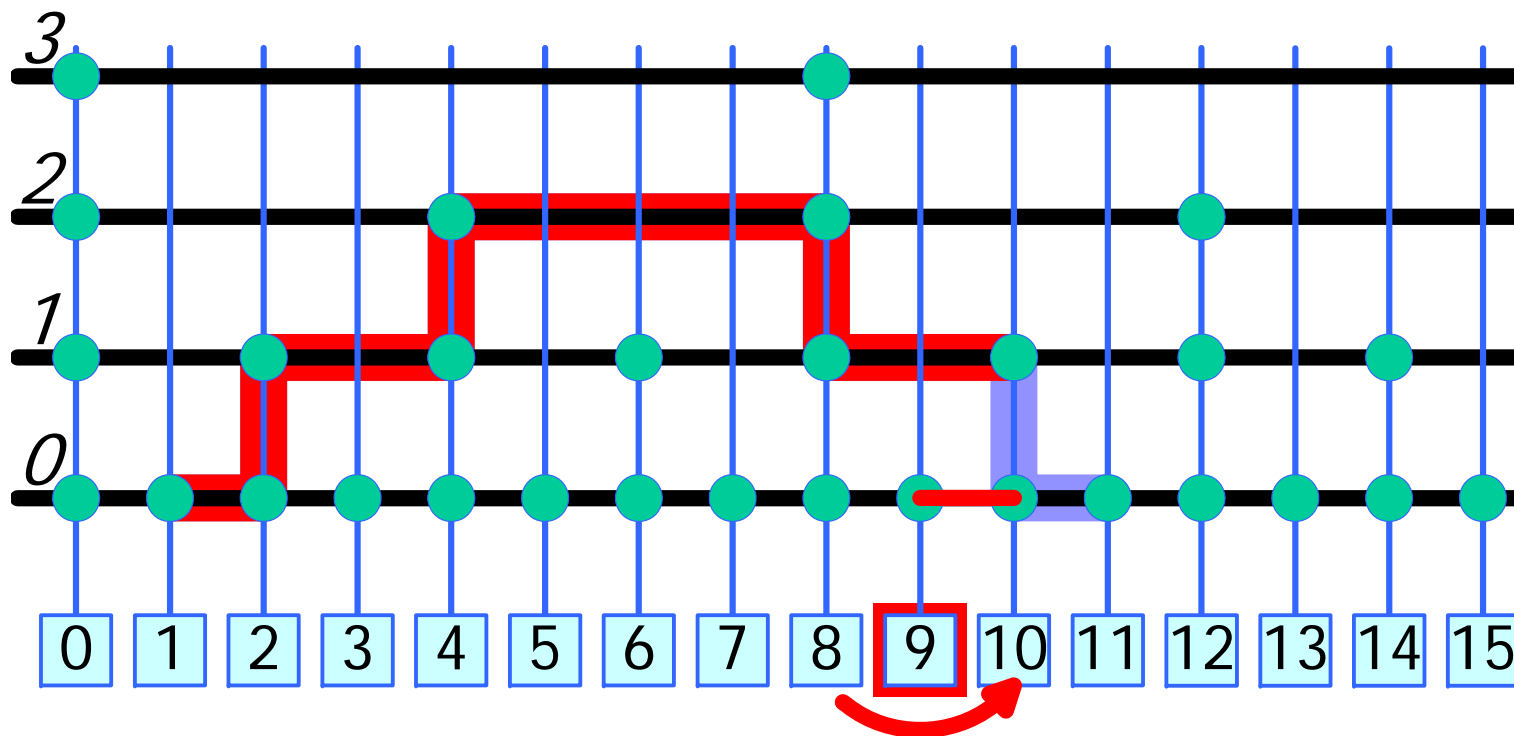
$$P_1^{11} = \langle d_1, T_0; d_2, T_0; d_4, T_0, T_3; d_8, T_0, T_6, T_7 \rangle$$





Skip List Precedence Proofs

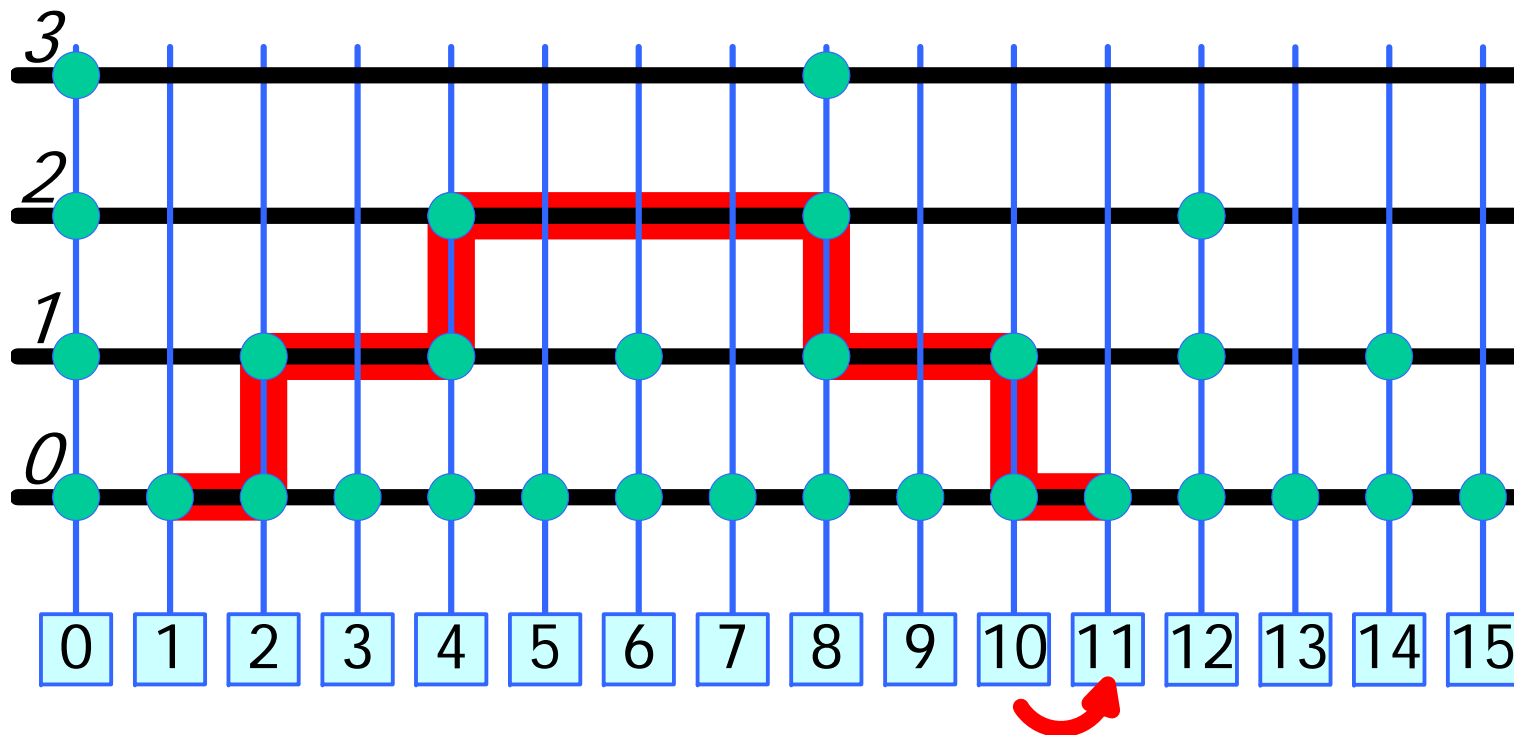
$$P_1^{11} = \langle d_1, T_0; d_2, T_0; d_4, T_0, T_3; d_8, T_0, T_6, T_7; d_{10}, T_9 \rangle$$





Skip List Precedence Proofs

$$P_1^{11} = \langle d_1, T_0; d_2, T_0; d_4, T_0, T_3; d_8, T_0, T_6, T_7; d_{10}, T_9; d_{11} \rangle$$





Skip List Machinery

- Digest is $D = \langle i, T, P \rangle$
 - latest known element index: i
 - authenticator of latest known element T
 - precedence proof from the well-known element 0 (*basis proof*) P
- Digest can be *extended* to include later elements of the skip list
 - Using a precedence proof



Digest Extensions

- Given
 - The known digest D (for time i)
 - The new digest D' (for time $j > i$)
 - A precedence proof P (from i to j)
- Recalculate the new digest D'' from D and P
- If $D'' = D'$, accept the extension



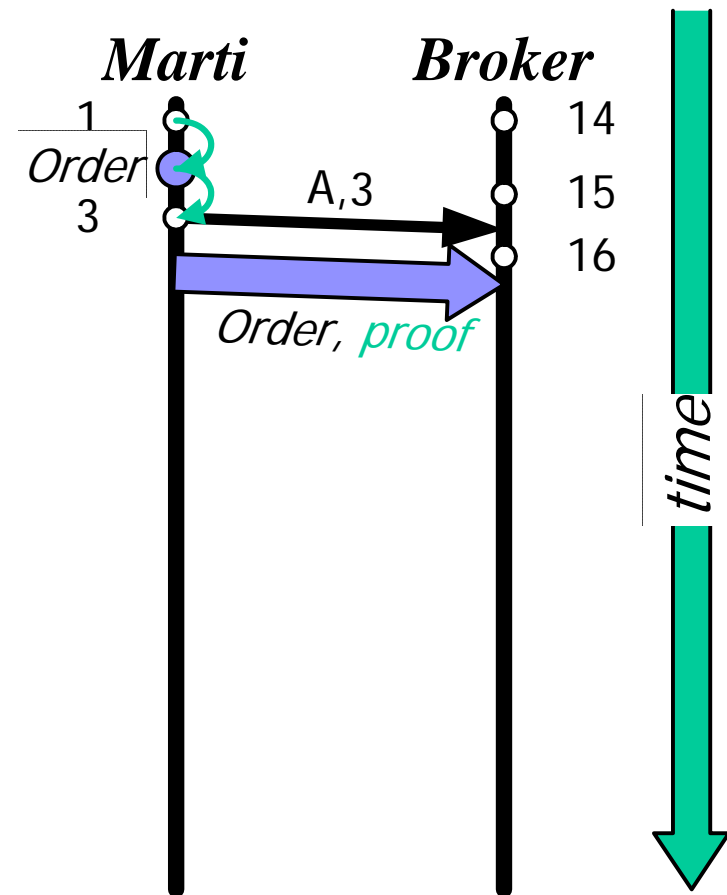
Skip List Properties

- It is infeasible to produce two conflicting proofs for element i that convince holders of two different digests D, D' , if D' is an extension of D
- If an adversary can accomplish this, then the adversary can also produce a second pre-image for the hash function used, which is "infeasible"



Timeweave Usage

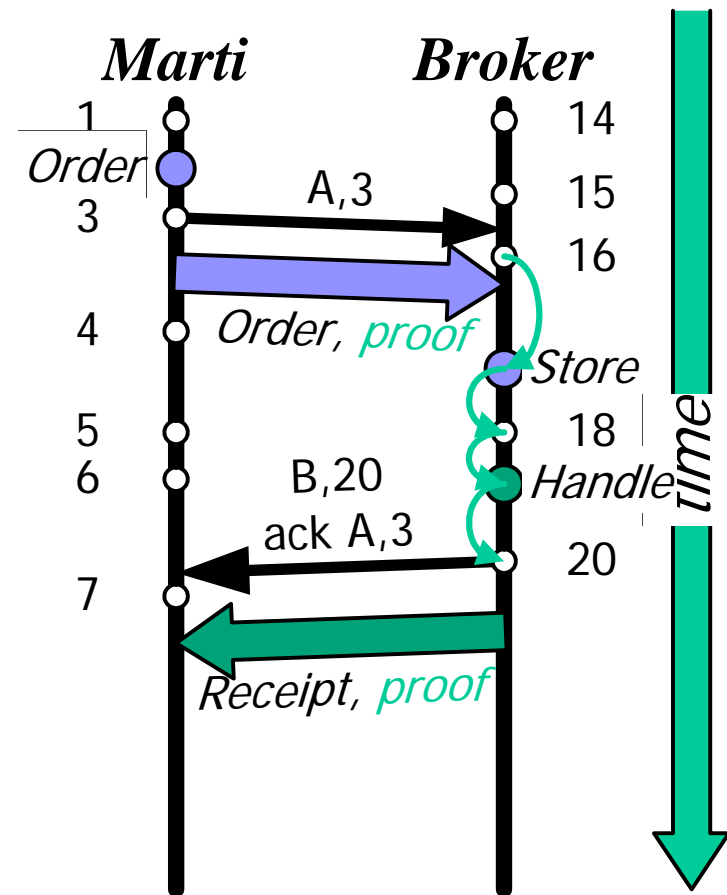
- Marti sends a sell order to her broker
- Order contains a proof of precedence showing it has been committed





Timeweave Usage

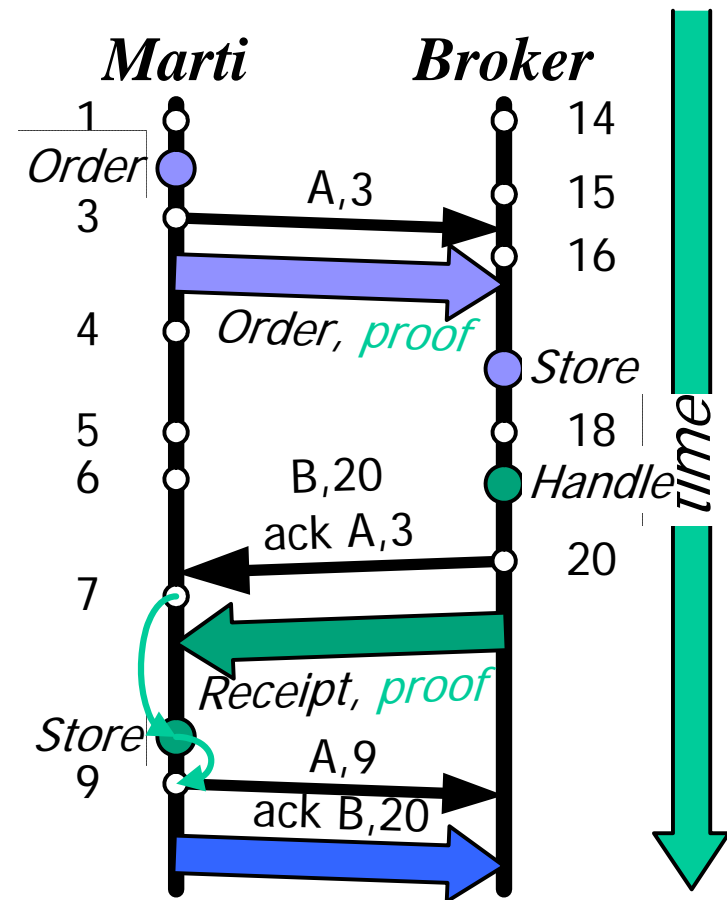
- Broker stores it for later fulfillment
- Broker returns an order receipt
- Marti stores the order receipt locally





Timeweave Usage

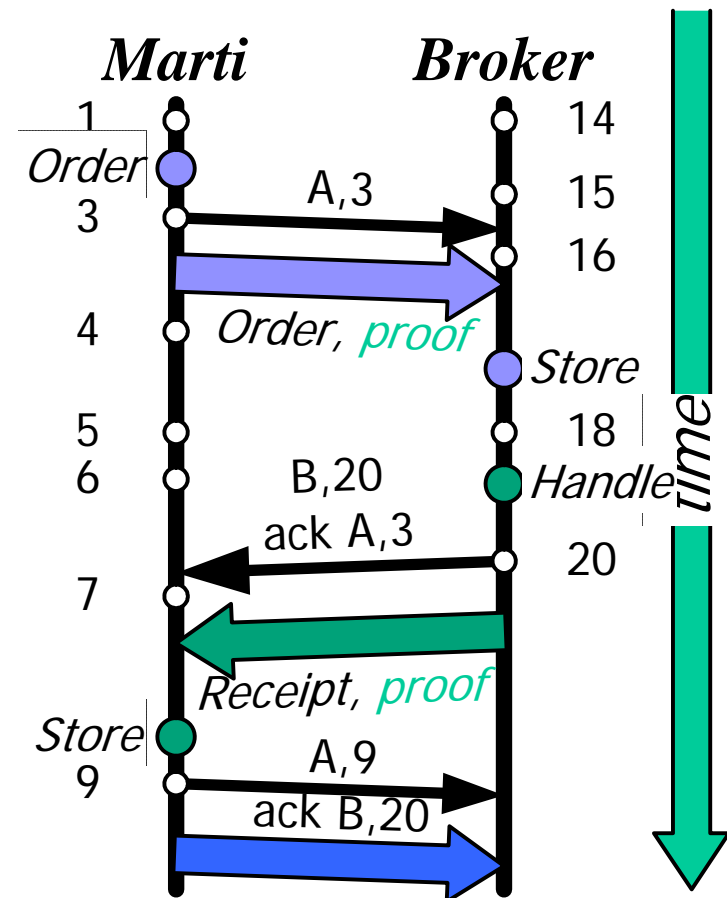
- Marti returns the commitment of the order receipt
- Broker stores it as completion of the exchange





Timeweave Usage

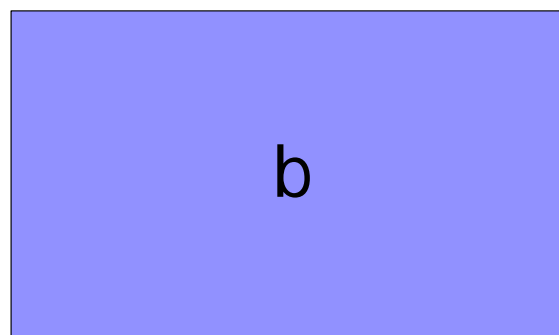
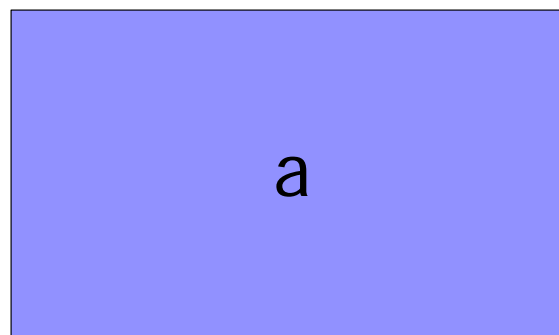
- Broker can show when
 - he received the order
 - he accepted it
 - he notified Marti
- Marti can show when
 - she sent the order
 - she received confirmation





One-Way Functions

- Consider two disk blocks
- One holds the value a
- The other holds the value b
- There is no way I can figure out which value was chosen first, a or b





One-Way Functions

- If $b=h(a)$, however, a **must** have been picked before b
- Otherwise, someone managed to compute the inverse of a , which is **infeasible**

a

$b=h(a)$