

Package Transparency for WebAssembly Registries

Kyle Brown, SingleStore

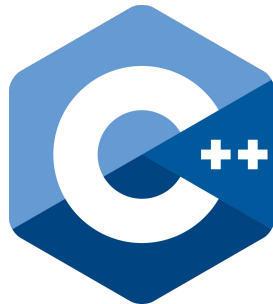
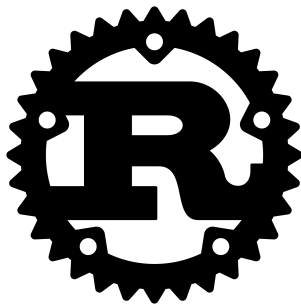
Overview

- Introduce **WebAssembly (Wasm)**
- What is a **Package Registry**?
- Applying **Certificate Transparency** to **Package Registries**
- **Package Transparency & warg**
- **Package Transparency vs. Various Attacks**

WebAssembly (Wasm)

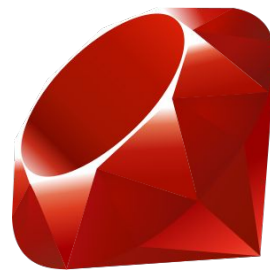
WebAssembly (Wasm) is...

A platform-agnostic “**compile target**” or...
something you can compile programs to.



Well-supported

Work is in progress for many more languages



Swift



Kotlin



History



Wasm was created as a web technology



It became a W3C Standard in 2019

supported by major browsers since 2017



Wasm **isn't** *just* for
the web

Wasm has really valuable properties

- **Portability**
- **Speed**
 - Low startup latency
 - Near-native performance
- **Security**
 - Capability safety
 - Sandboxing & memory isolation

People using **Wasm** outside the browser



Database Extensibility



Distributed Apps



Edge Computing



Serverless





BYTECODE
ALLIANCE

Wasm applications
and libraries will be
composed, shared,
and **deployed**

Wasm needs a native
package registry

We need a registry as secure as Wasm itself



You wouldn't seal a vault...



with a Cheeto

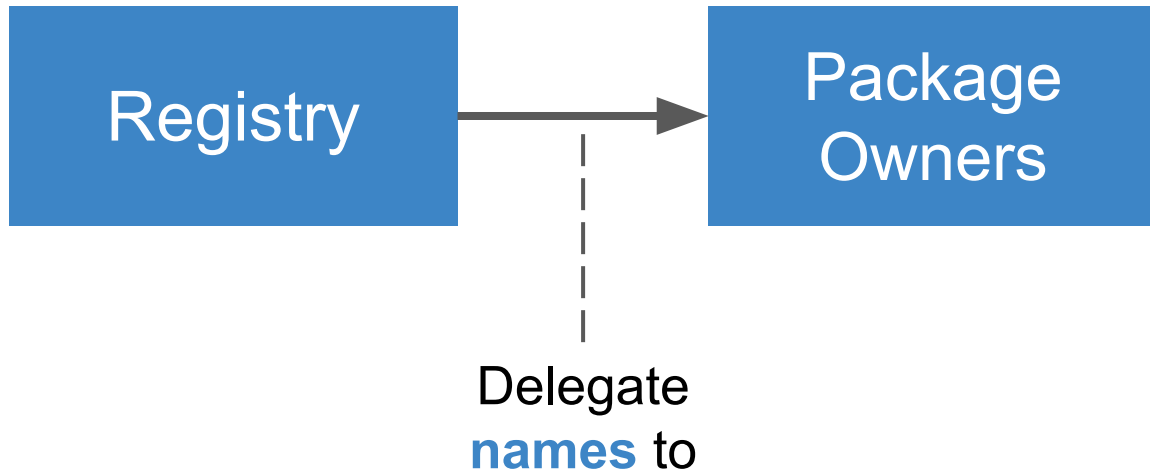
What is a Package Registry?

Package Registries' Role

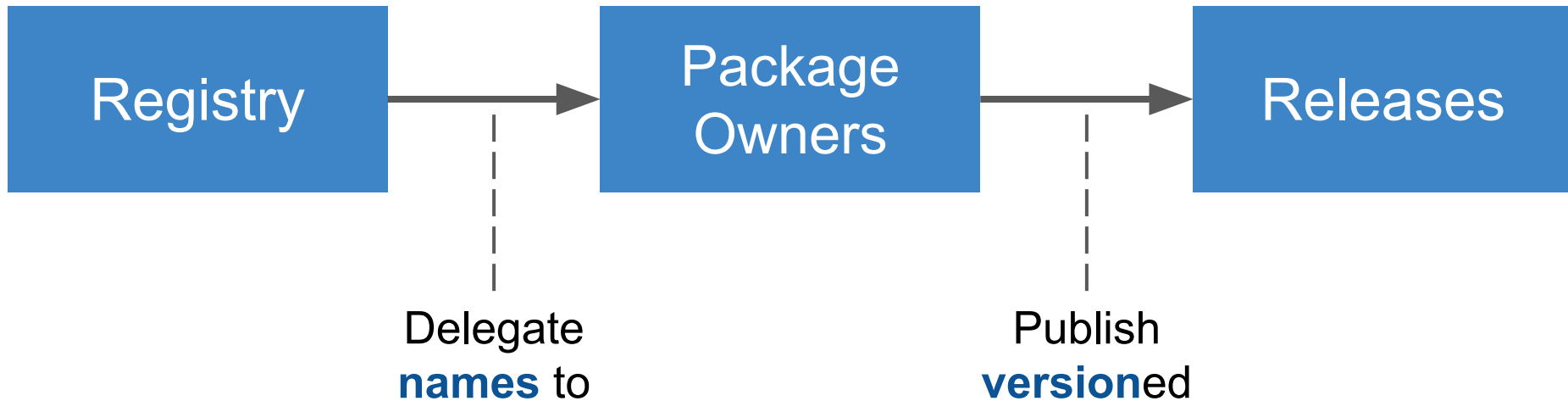


Registry

Package Registries' Role



Package Registries' Role



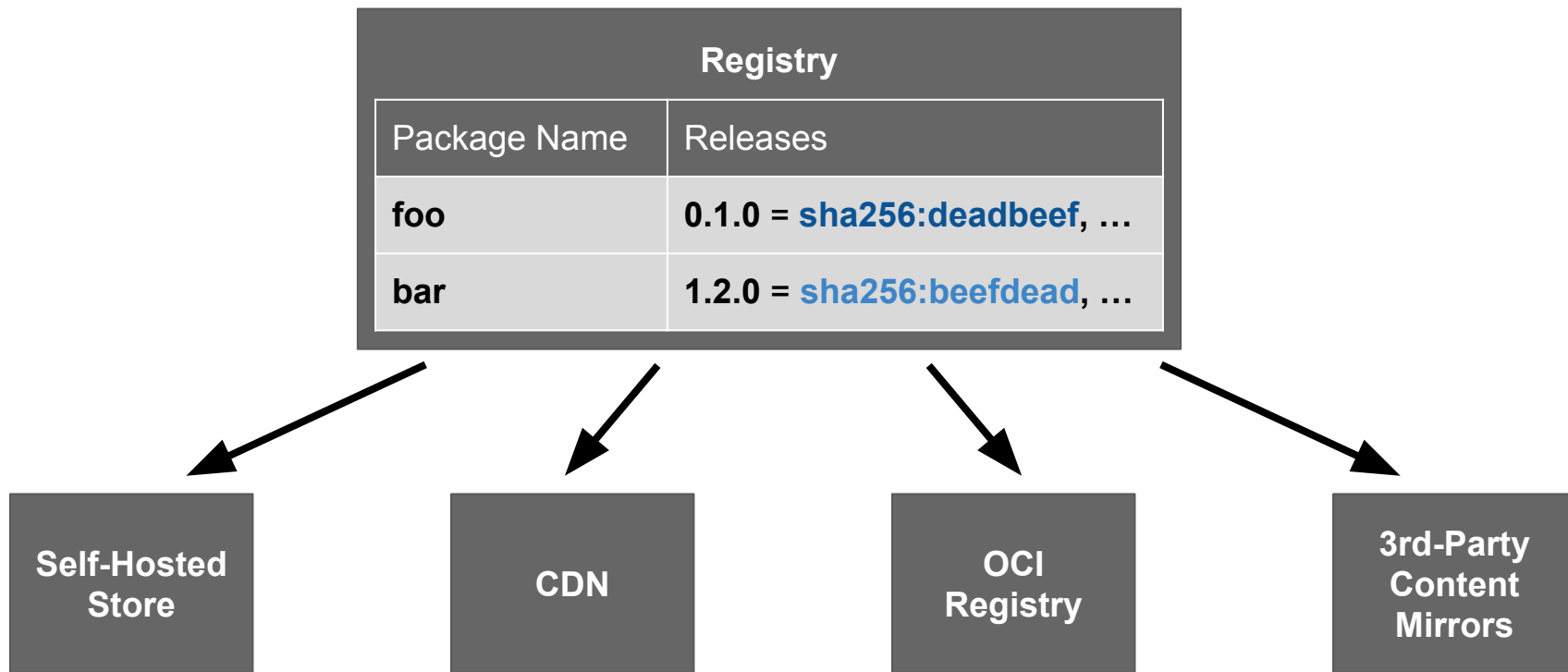
A Registry is an Index

(name, version) → package digest

Registries must control metadata

Registry	
Package Name	Releases
foo	0.1.0 = sha256:deadbeef, ...
bar	1.2.0 = sha256:beefdead, ...

Registries may delegate content hosting



How can we apply
Certificate Transparency to
Package Registries?

People are able to
detect when CAs
misissue certificates

Clients *should* be able to
detect when registries
accept **invalid**
package updates

Package Transparency

“Package transparency is

Lann Martin, Fermyon

“Package transparency is publishing
cryptographically-verifiable commitments

Lann Martin, Fermyon

“Package transparency is publishing
cryptographically-verifiable commitments
to the state of a package repository

Lann Martin, Fermyon

“Package transparency is publishing cryptographically-verifiable commitments to the state of a package repository to allow auditing of the actions of package authors and the registry itself over time.”

Lann Martin, Fermyon

Components of Package Transparency

1. Publicly Available Package Registry State
2. Cryptographically-Verifiable Commitments
3. Auditing Package Authors and the Registry

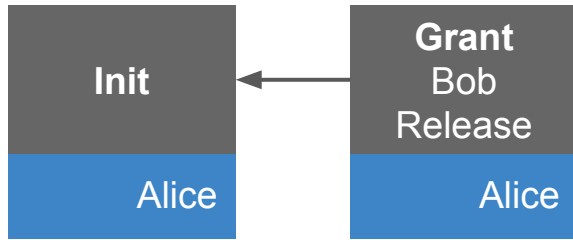
warg is a protocol for
Package Transparency

Publicly Available
Registry State

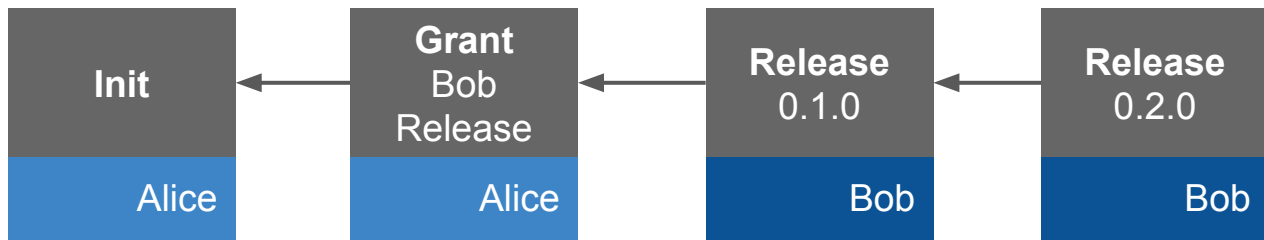
Append-only Package Logs



Append-only Package Logs



Append-only Package Logs



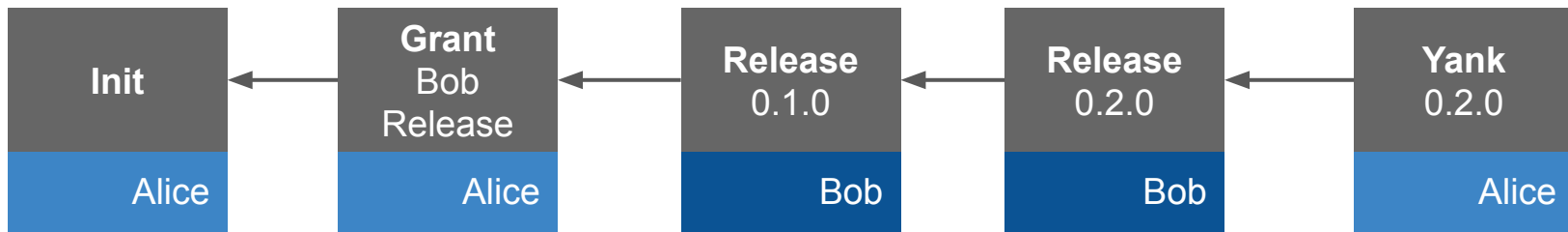
Releases

- 0.1.0

Releases

- 0.1.0
- 0.2.0

Append-only Package Logs



Releases

- 0.1.0

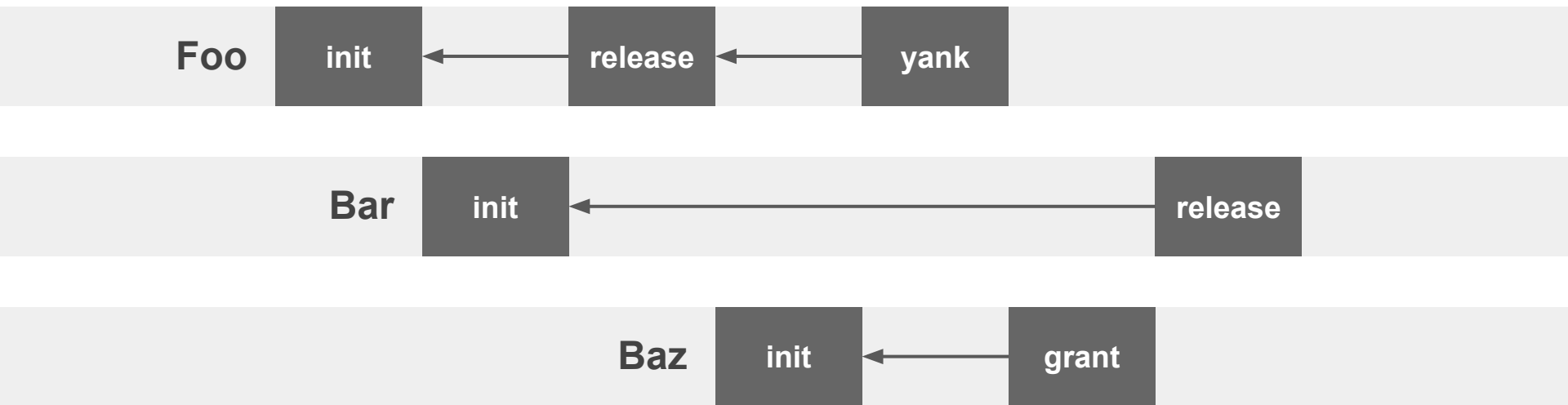
Releases

- 0.1.0
- 0.2.0

Releases

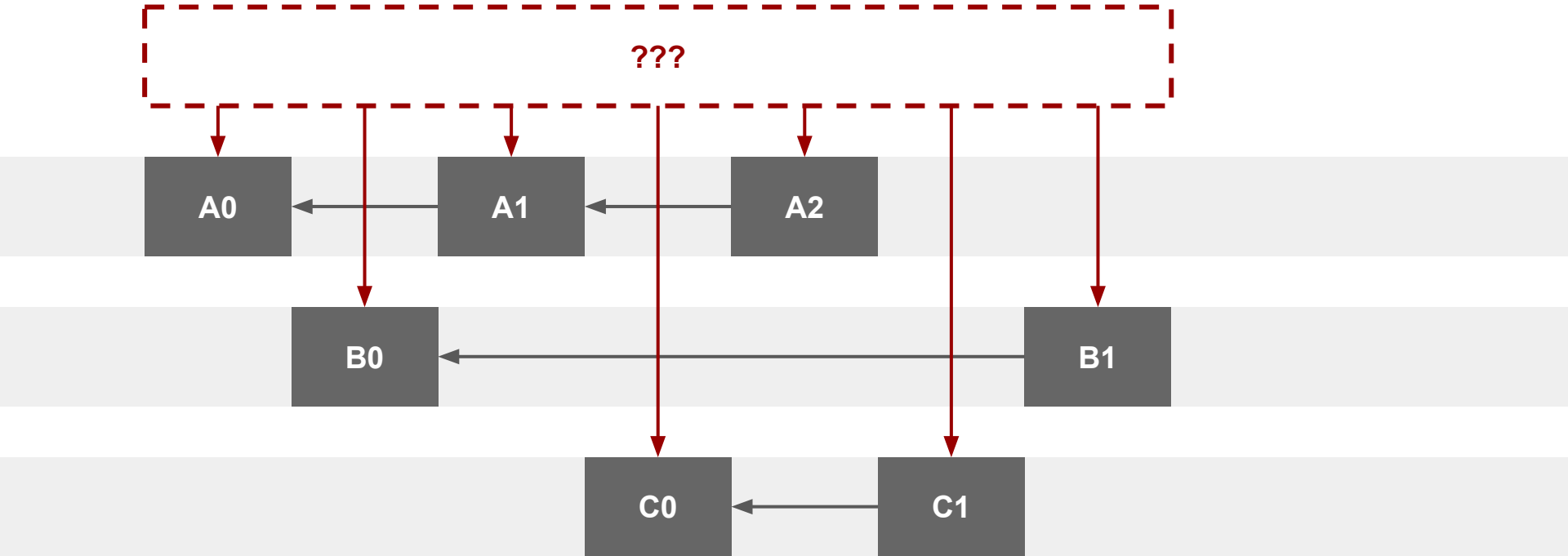
- 0.1.0
- ~~0.2.0~~

Package Logs



Cryptographically-Verifiable Commitments

What does the registry claim has happened?



Verifiable Log

(Merkle Tree)

What does a Verifiable Log do?

- A **Verifiable Log** is a total ordering of records
- Each **Verifiable Log** is described by a unique hash
- You can cryptographically check if a record is in the log

Verifiable Log (Merkle Tree)

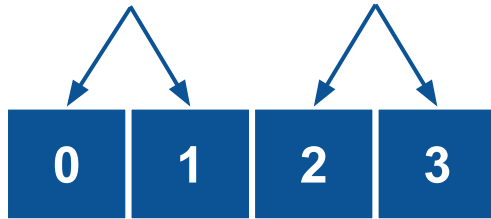


Verifiable Log (Merkle Tree)



Leaf Hash
`hash (0b00 || leaf)`

Verifiable Log (Merkle Tree)



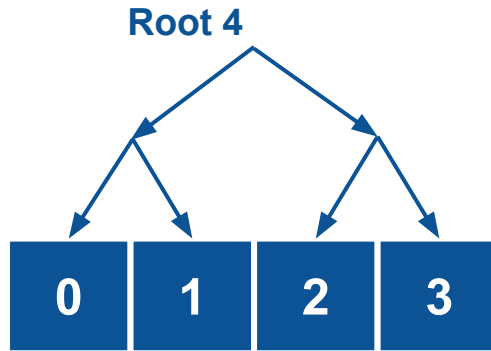
Branch Hash

`hash (0b01 || left || right)`

Leaf Hash

`hash (0b00 || leaf)`

Verifiable Log (Merkle Tree)



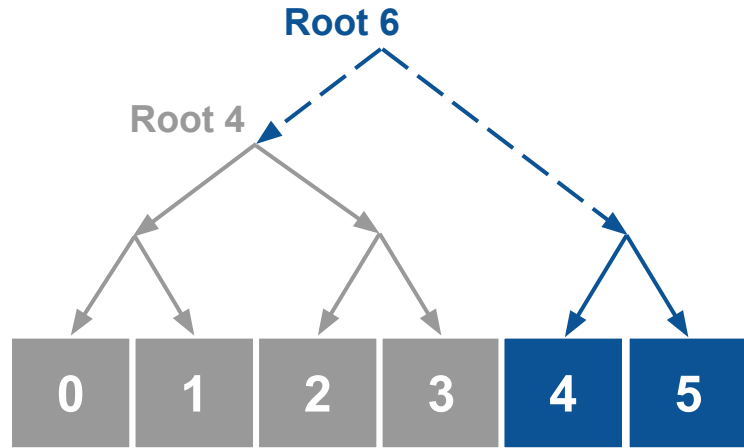
Branch Hash

`hash (0b01 || left || right)`

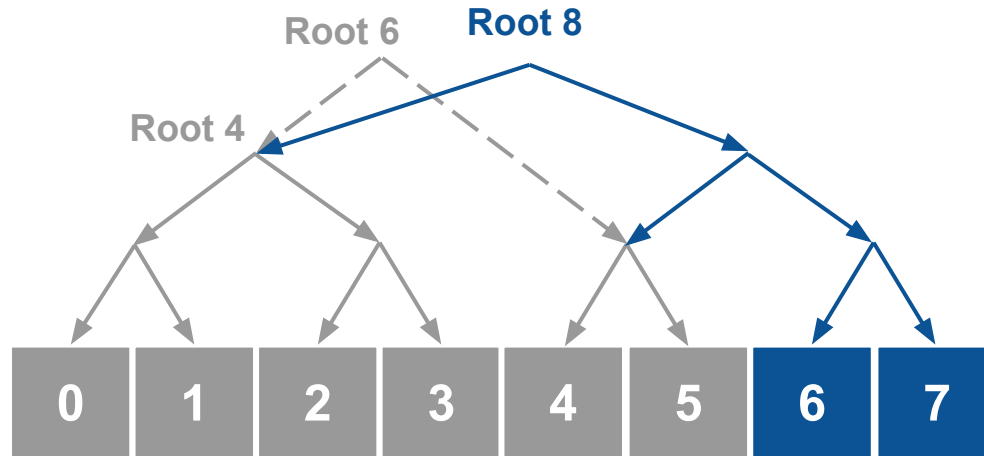
Leaf Hash

`hash (0b00 || leaf)`

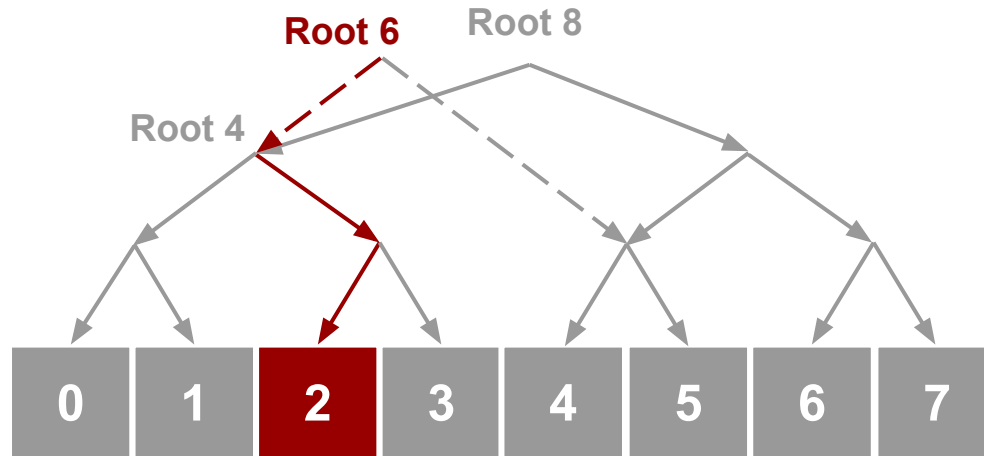
Verifiable Log (Merkle Tree)



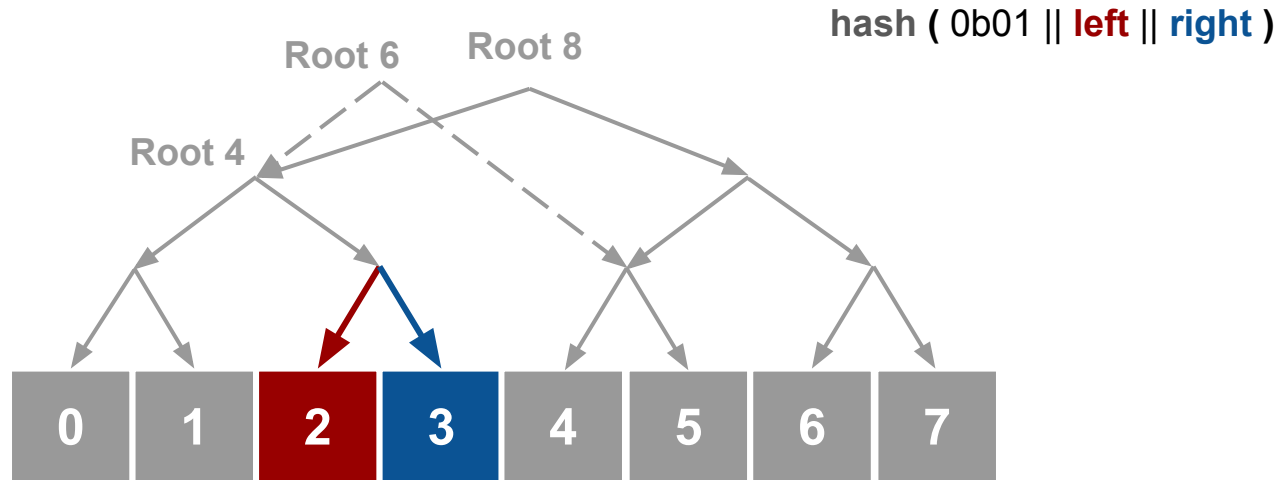
Verifiable Log (Merkle Tree)



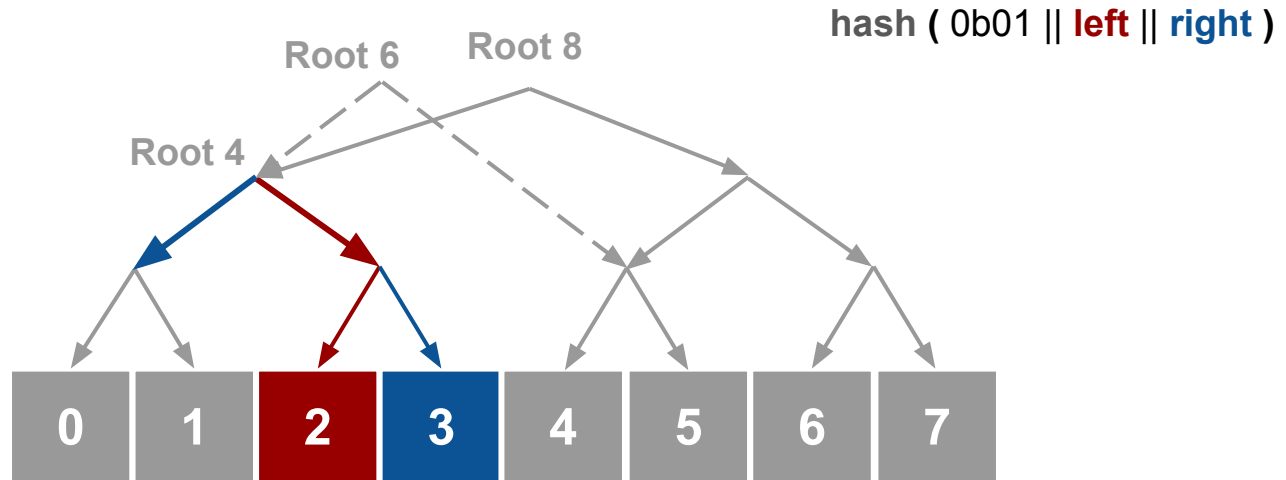
Verifiable Log (Merkle Tree) - Inclusion Proofs



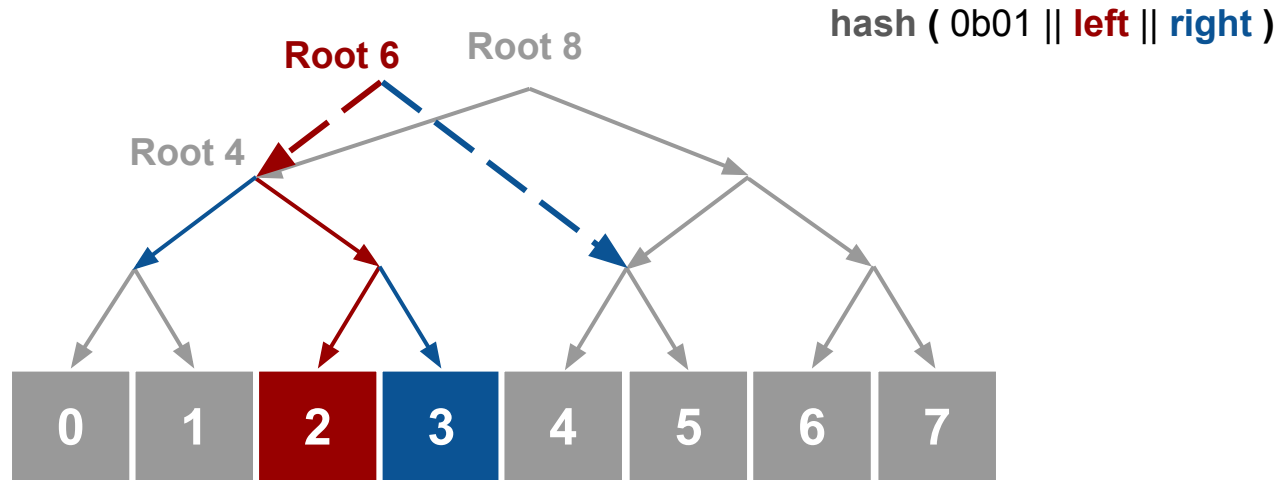
Verifiable Log (Merkle Tree) - Inclusion Proofs



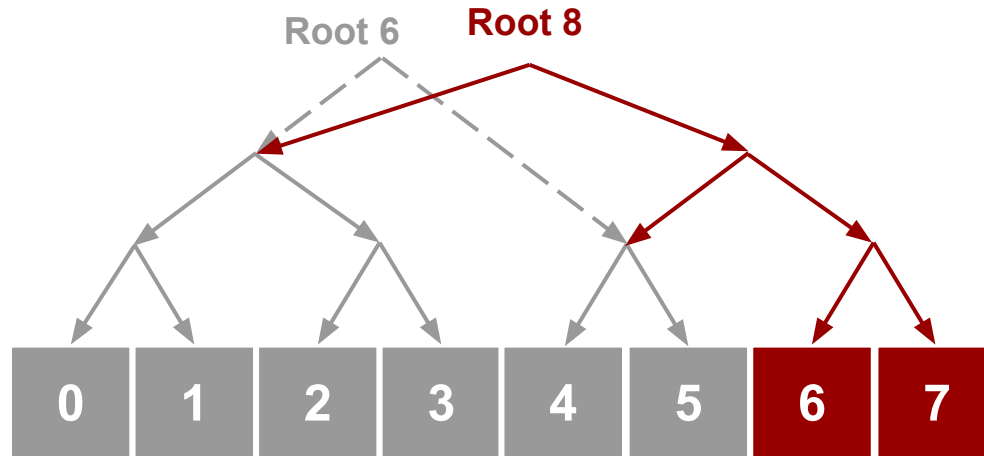
Verifiable Log (Merkle Tree) - Inclusion Proofs



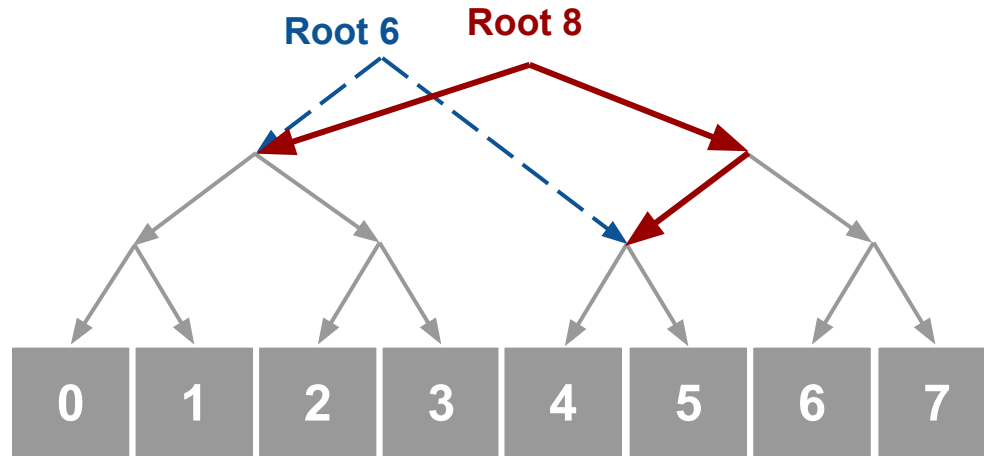
Verifiable Log (Merkle Tree) - Inclusion Proofs



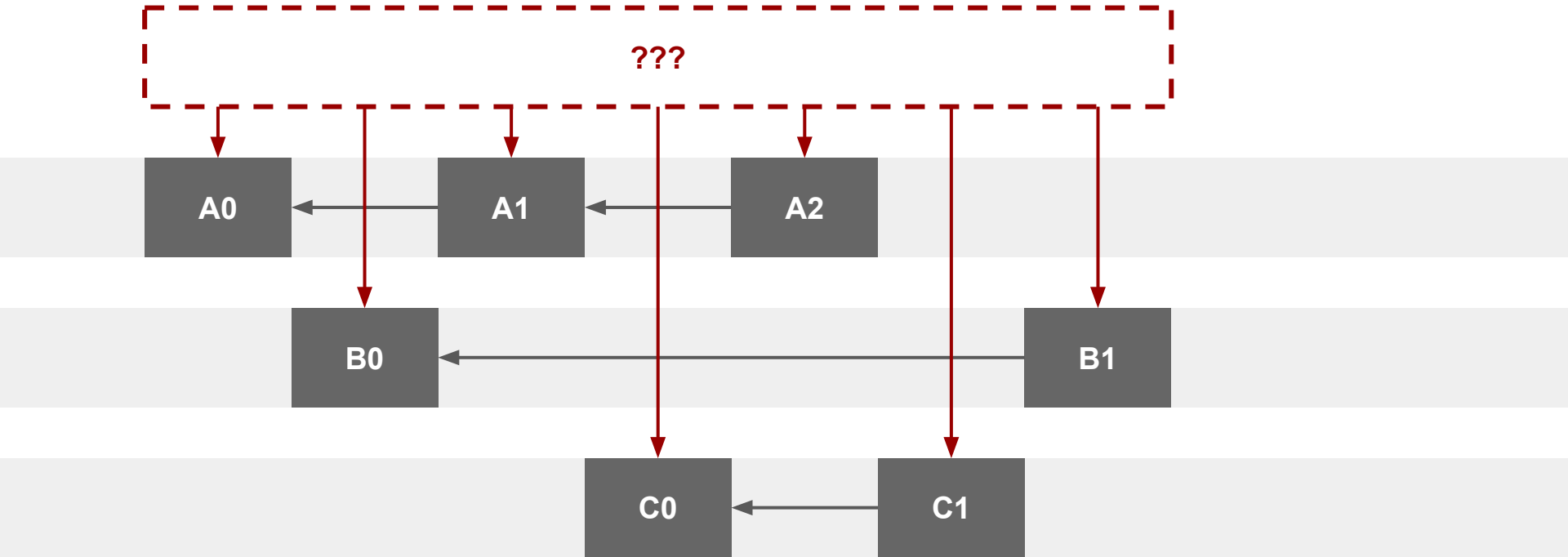
Verifiable Log (Merkle Tree) - Consistency Proofs



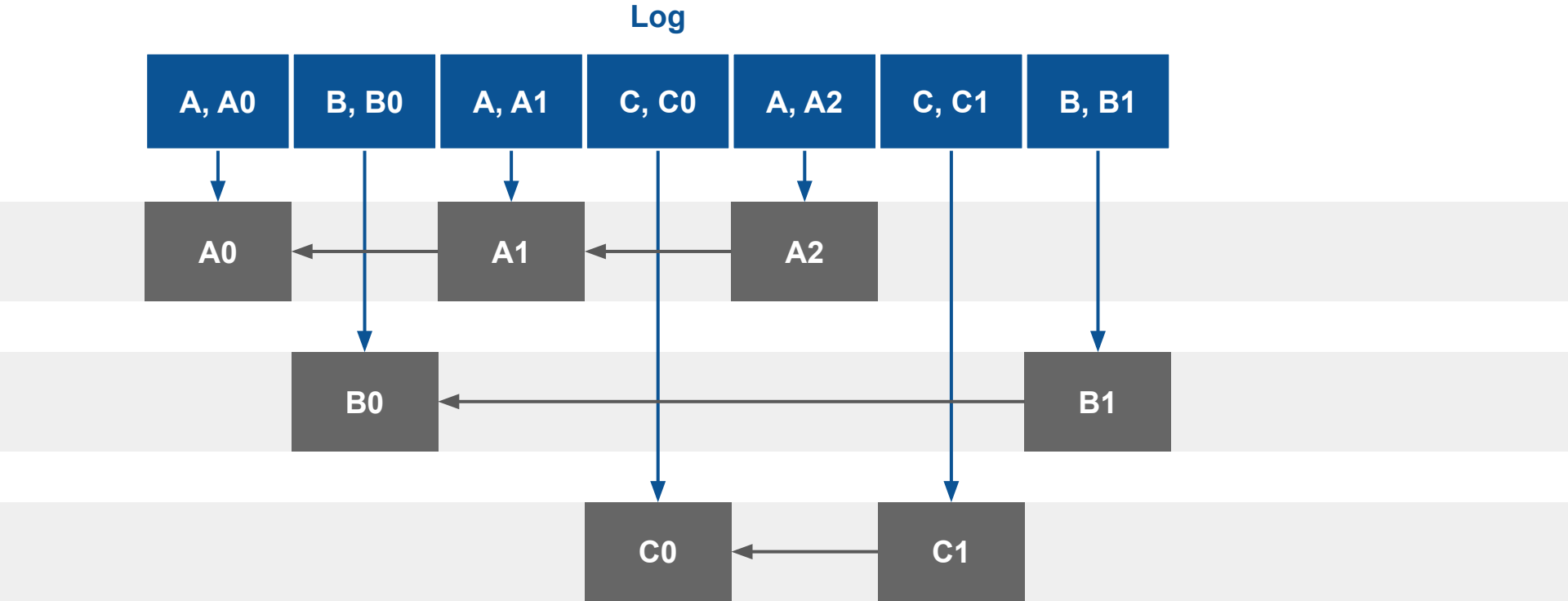
Verifiable Log (Merkle Tree) - Consistency Proofs



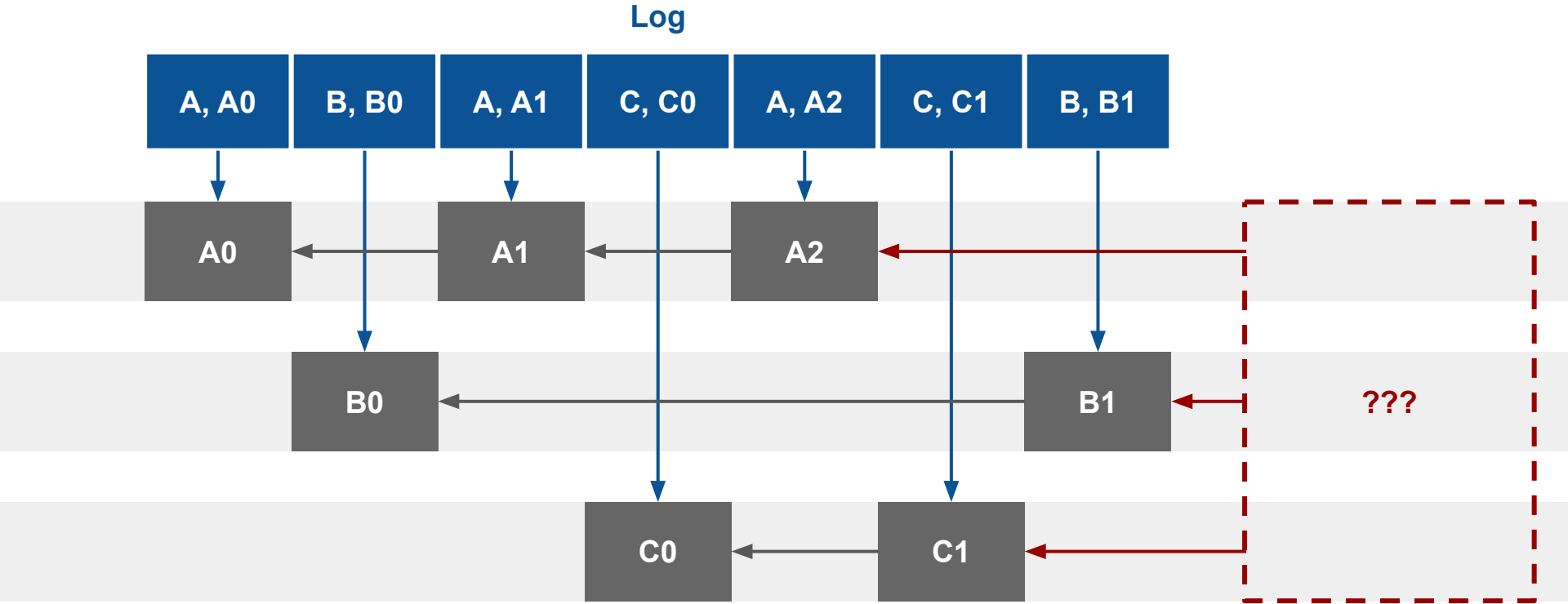
So, what does the registry claim has happened?



The Package Records in the Verifiable Log



How do clients know what the latest record is?



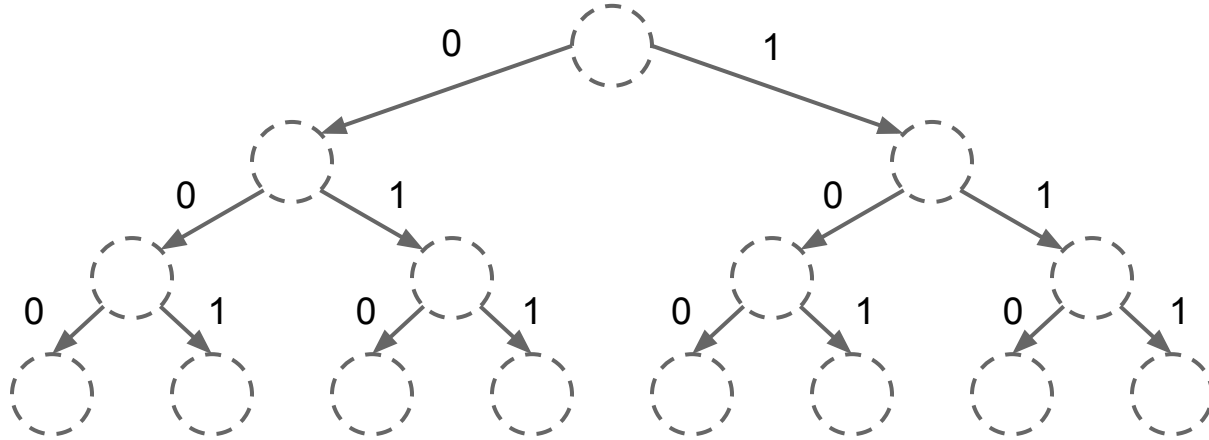
Verifiable Map

(Sparse Merkle Tree)

What does a Verifiable Map do?

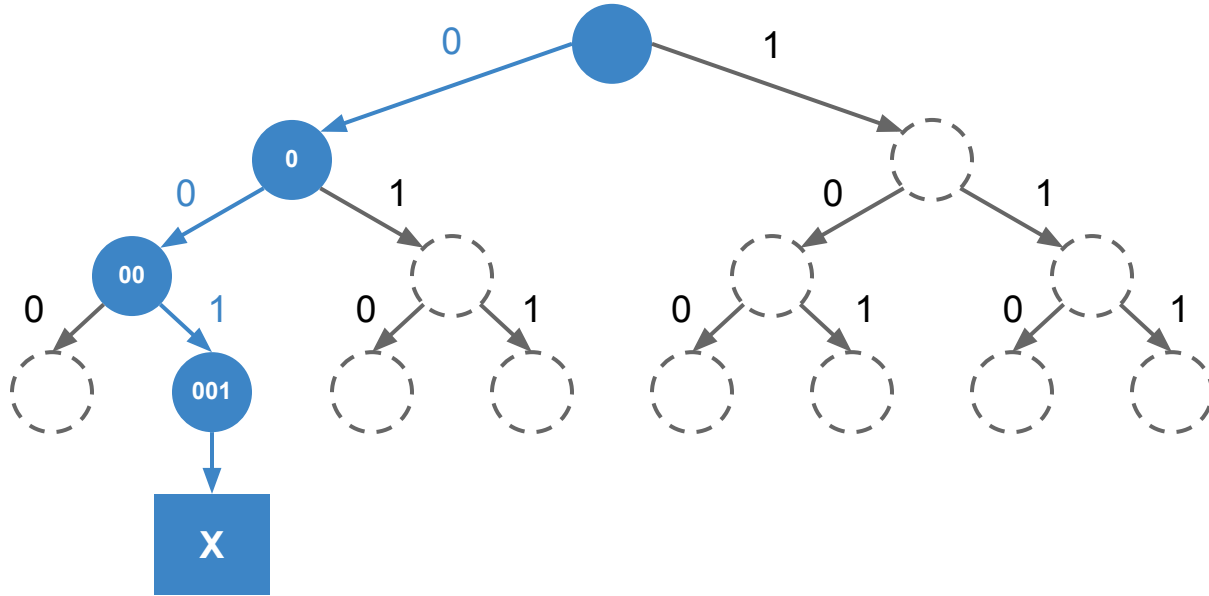
- A **Verifiable Map** is a key-value mapping
- Each **Verifiable Map** is described by a unique hash
- You can cryptographically check if a value is associated with a key

Verifiable Map (Sparse Merkle Tree)



Verifiable Map (Sparse Merkle Tree)

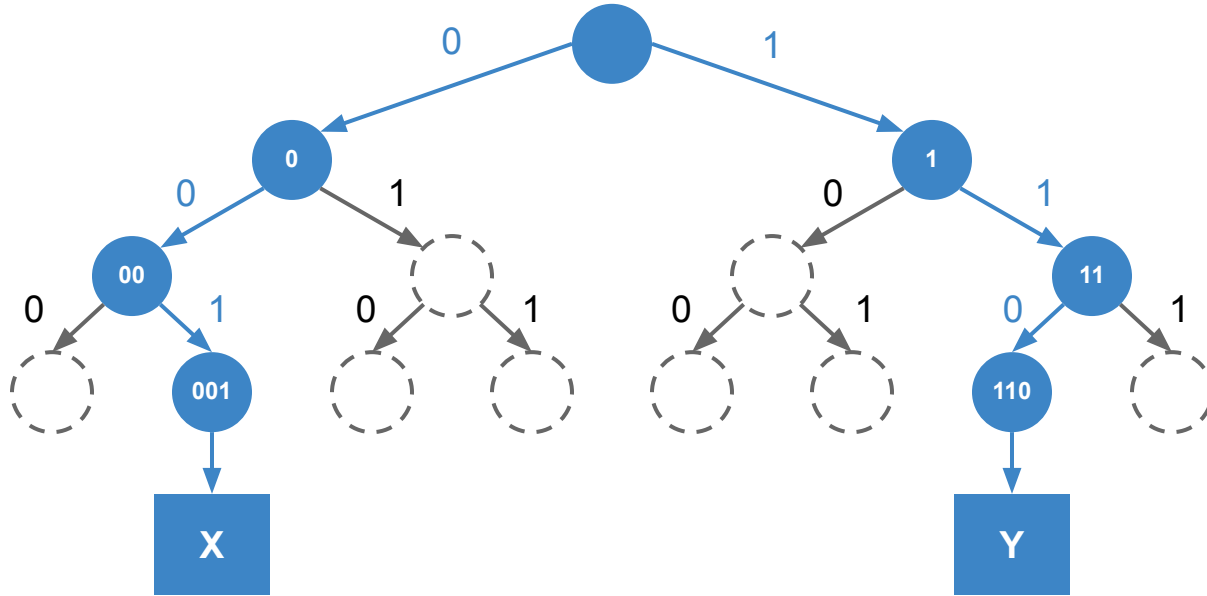
001 → X



Verifiable Map (Sparse Merkle Tree)

001 → X

110 → Y

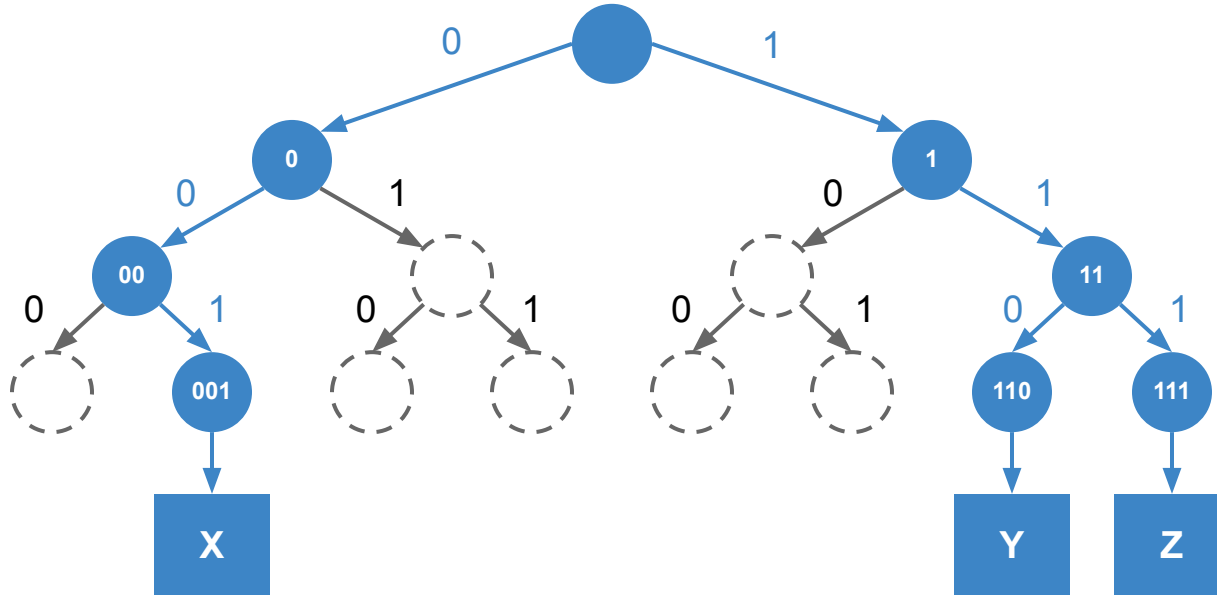


Verifiable Map (Sparse Merkle Tree)

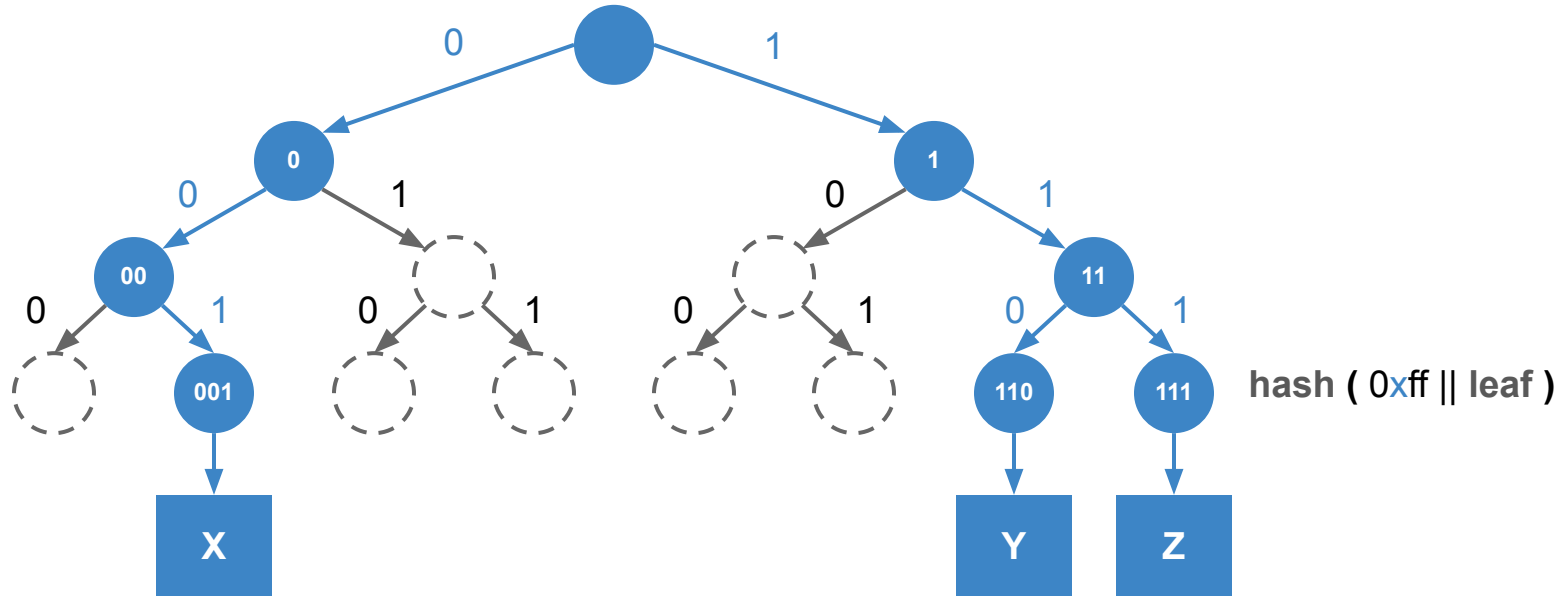
001 → X

110 → Y

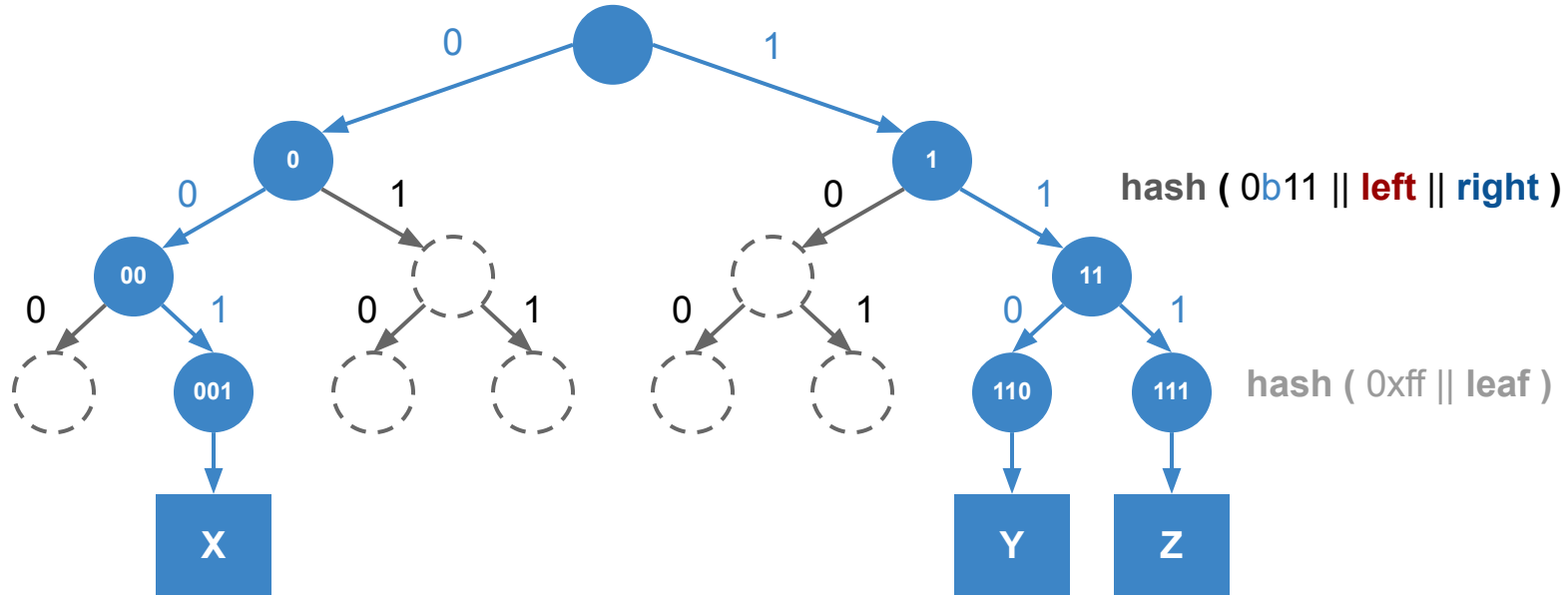
111 → Z



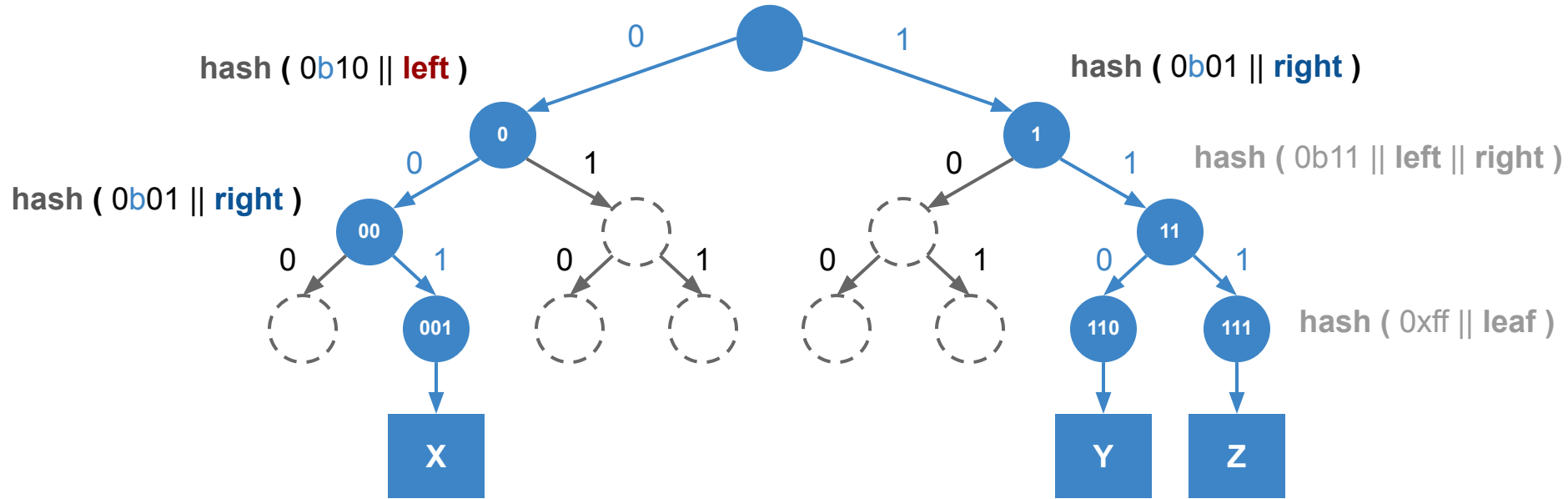
Verifiable Map (Sparse Merkle Tree)



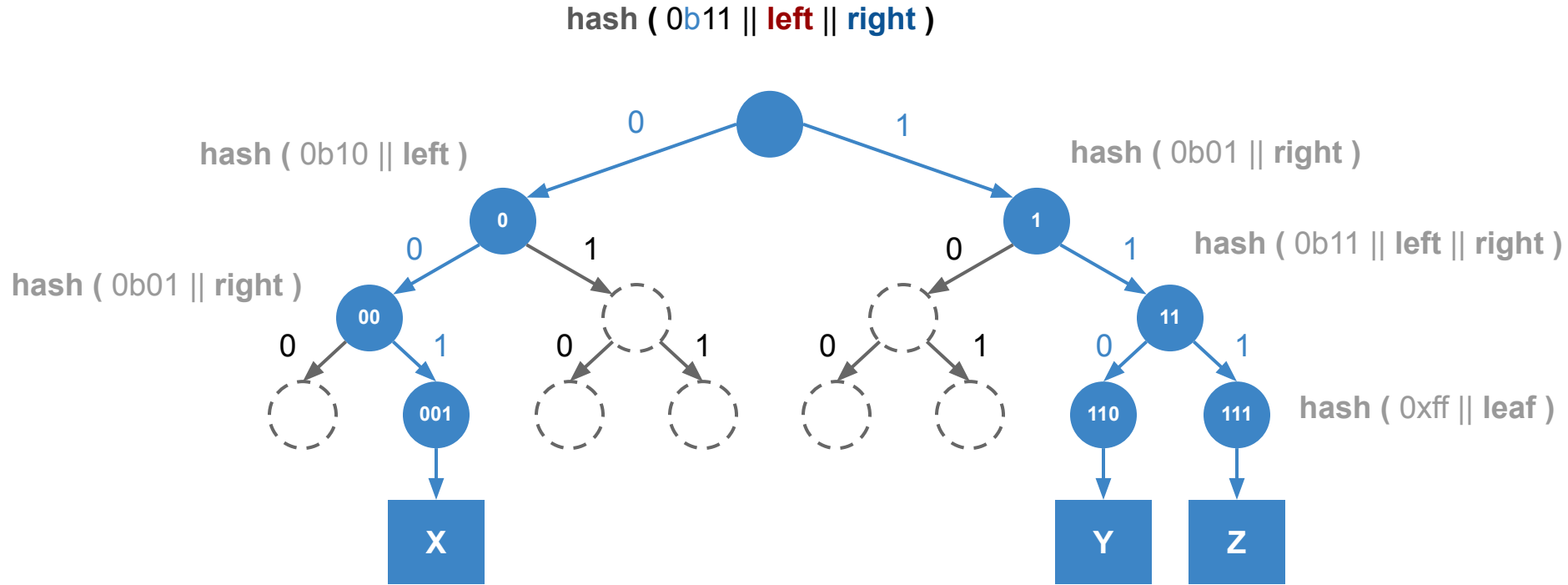
Verifiable Map (Sparse Merkle Tree)



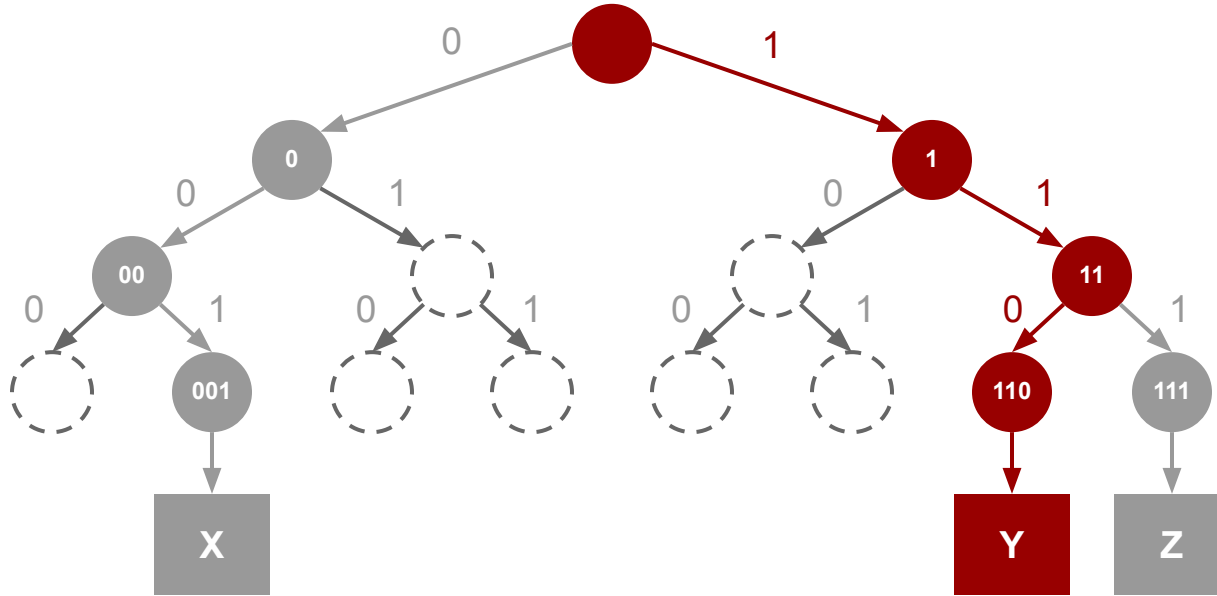
Verifiable Map (Sparse Merkle Tree)



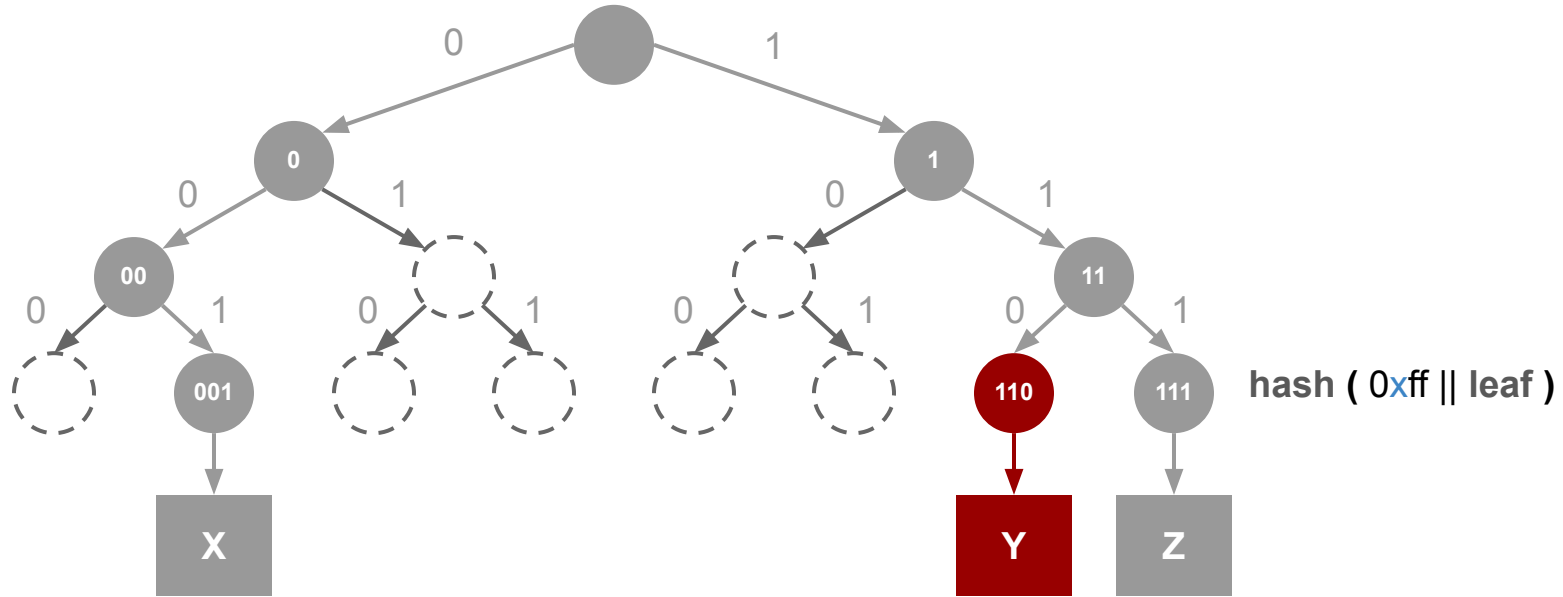
Verifiable Map (Sparse Merkle Tree)



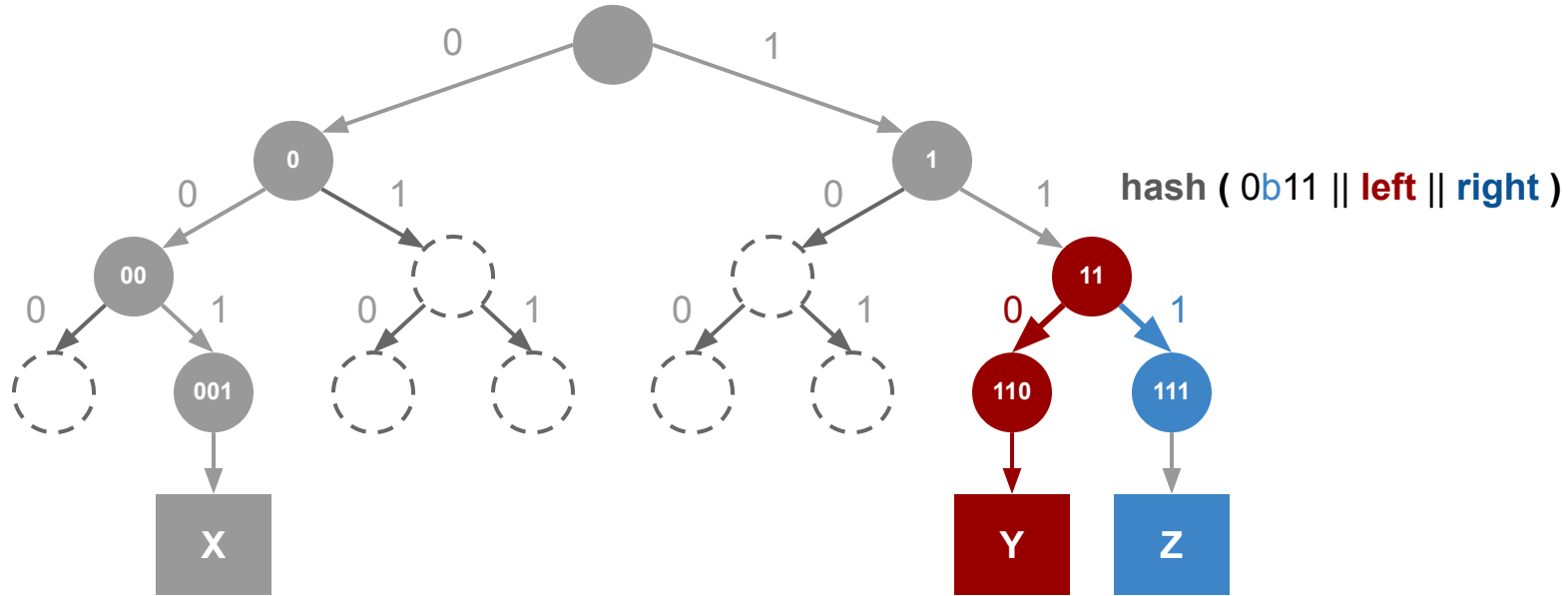
Verifiable Map (Sparse Merkle Tree) - Inclusion Proof



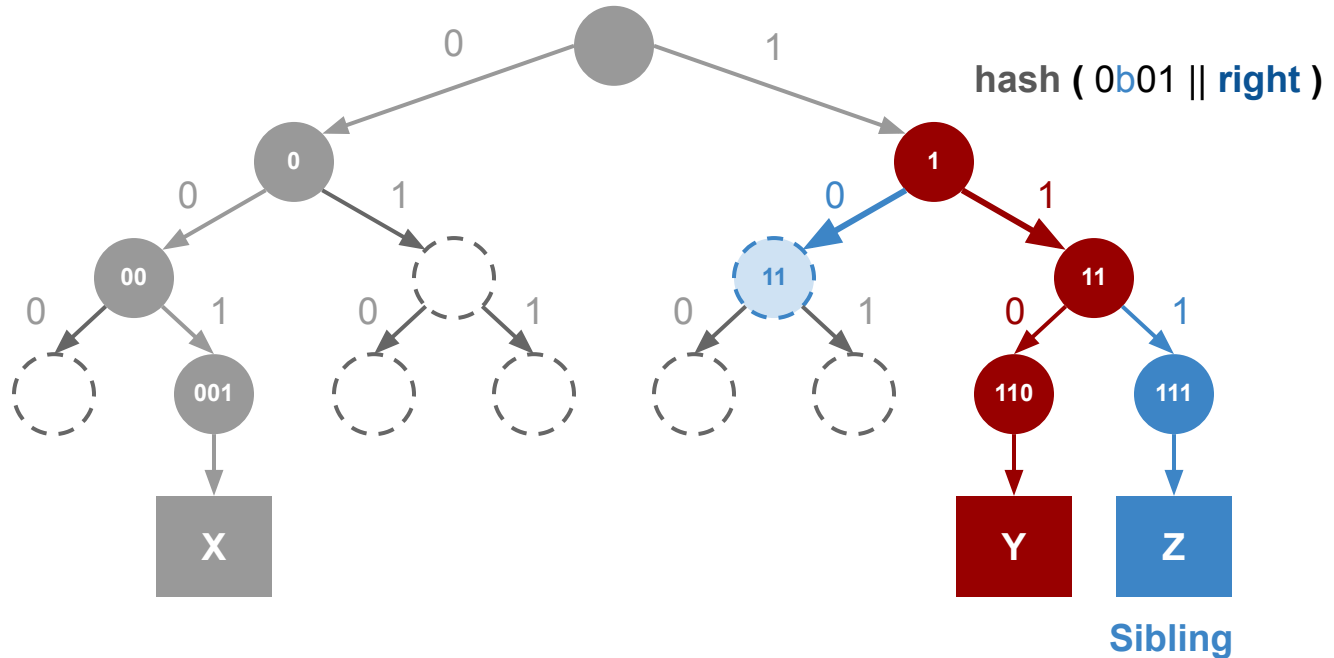
Verifiable Map (Sparse Merkle Tree) - Inclusion Proof



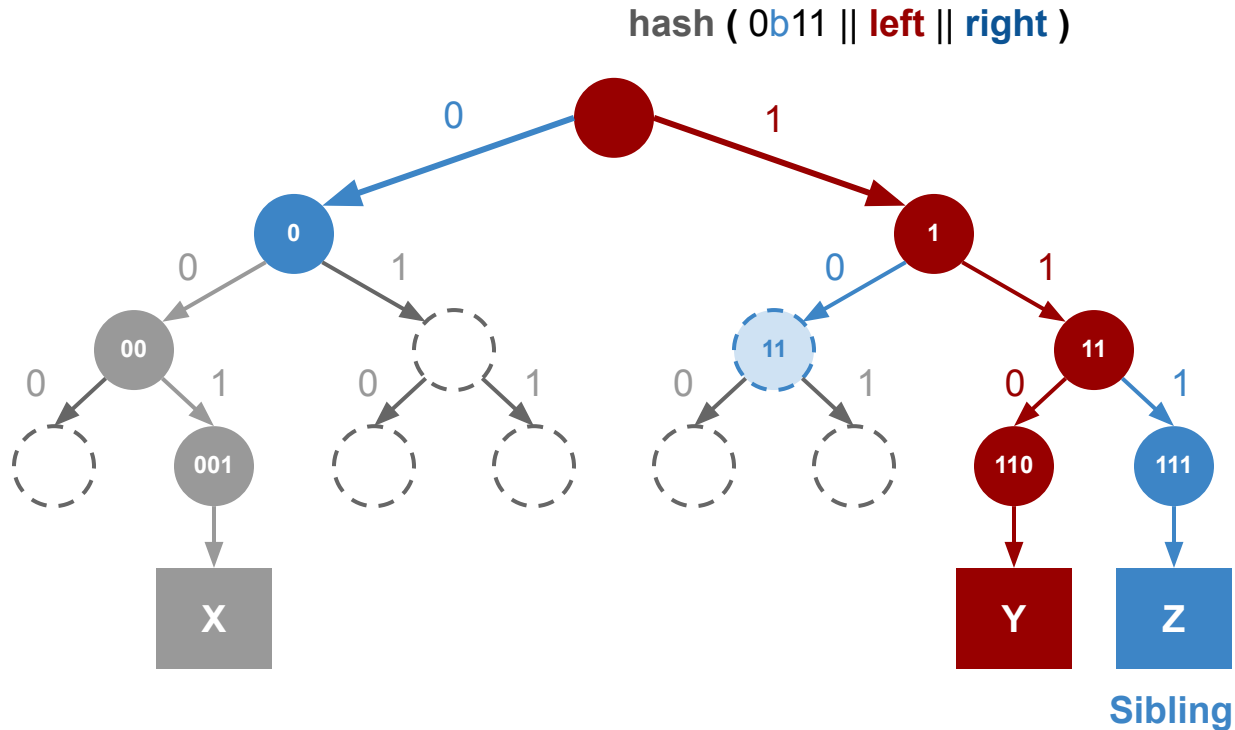
Verifiable Map (Sparse Merkle Tree) - Inclusion Proof



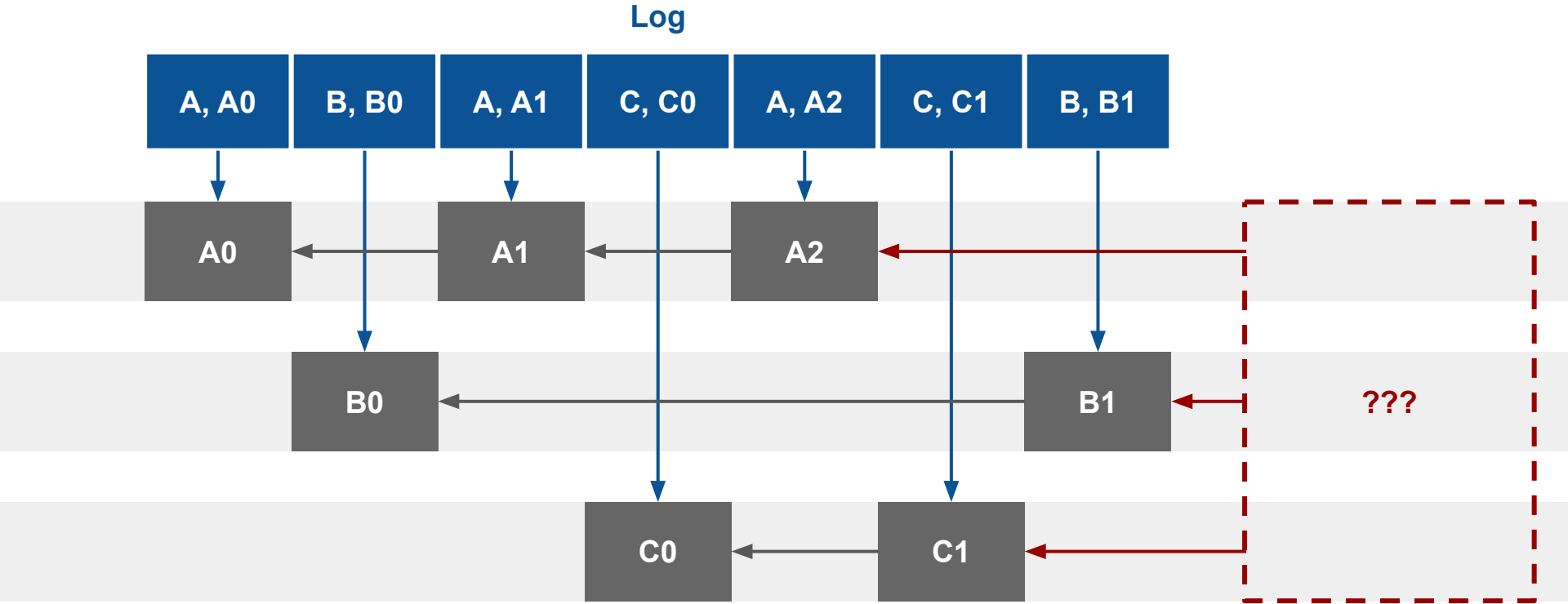
Verifiable Map (Sparse Merkle Tree) - Inclusion Proof



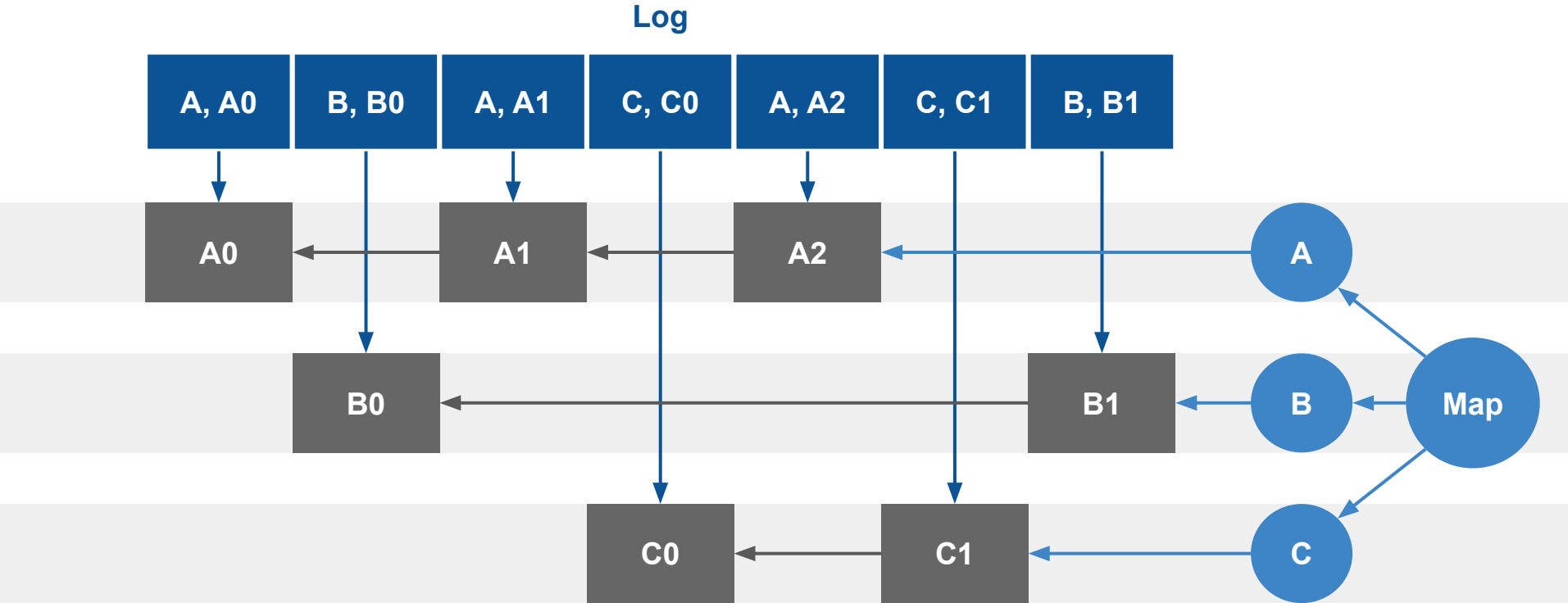
Verifiable Map (Sparse Merkle Tree) - Inclusion Proof



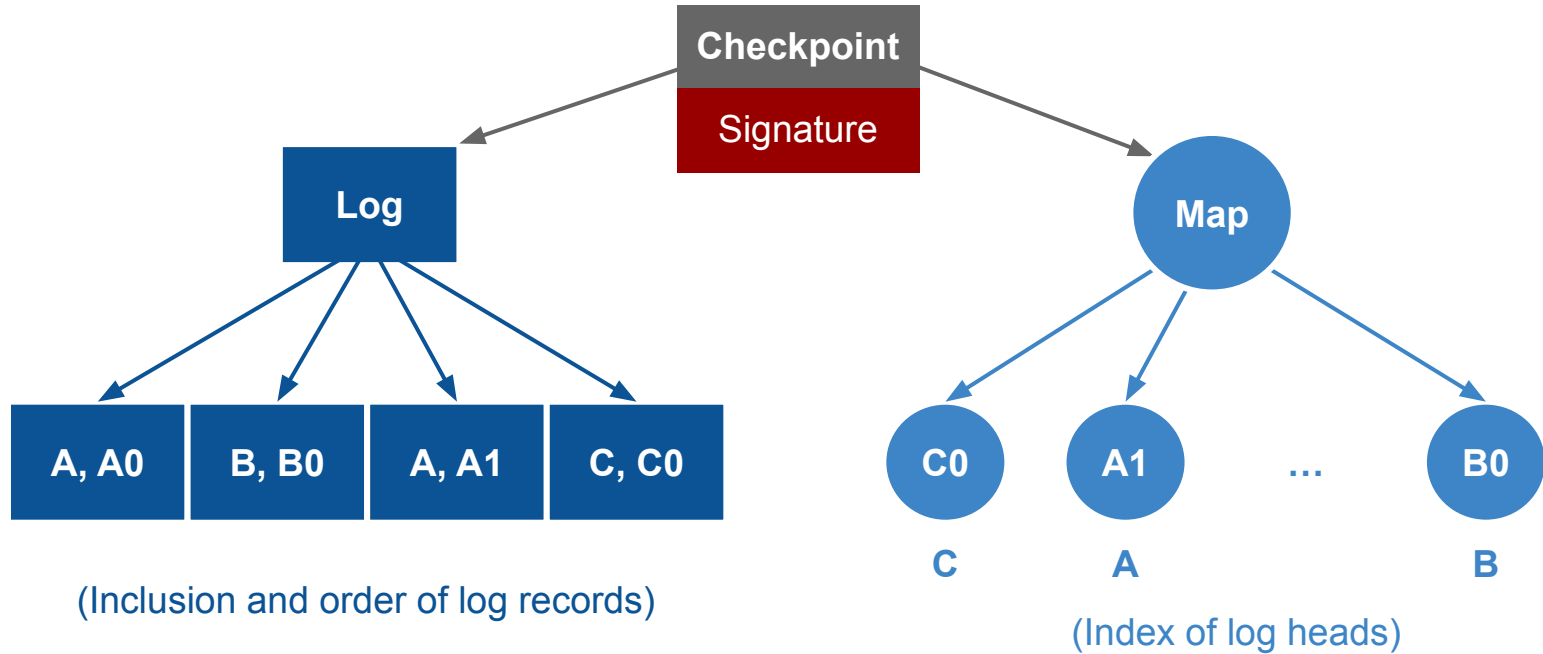
So, how do clients know what the latest record is?



Use a verifiable map of package log heads



Overall Checkpoint / Commitment



Auditing & Verification

Clients

They are

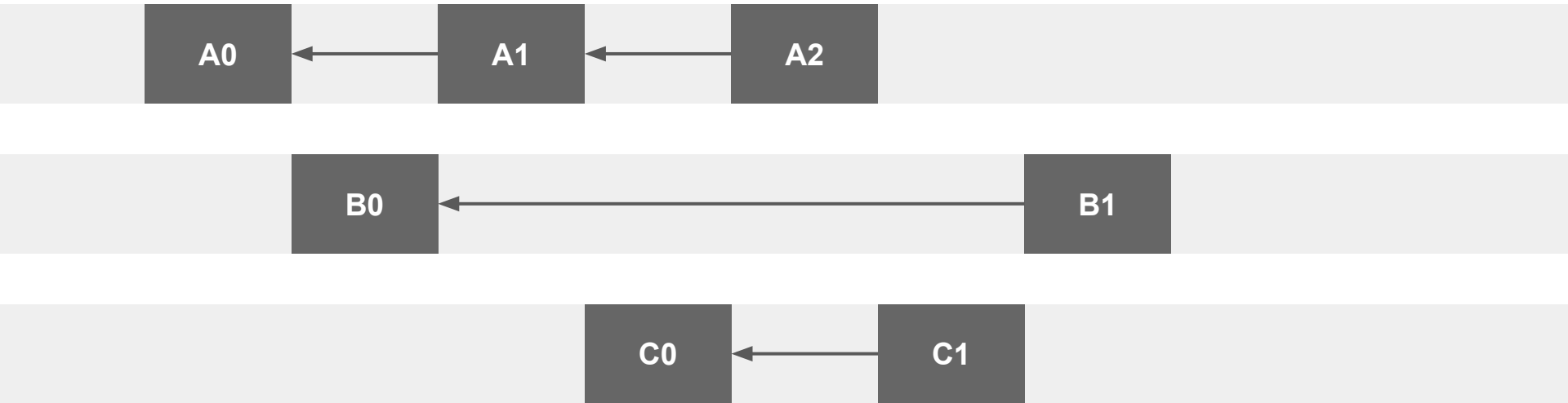
- resource constrained
- only interested in some of the state

So, clients verify

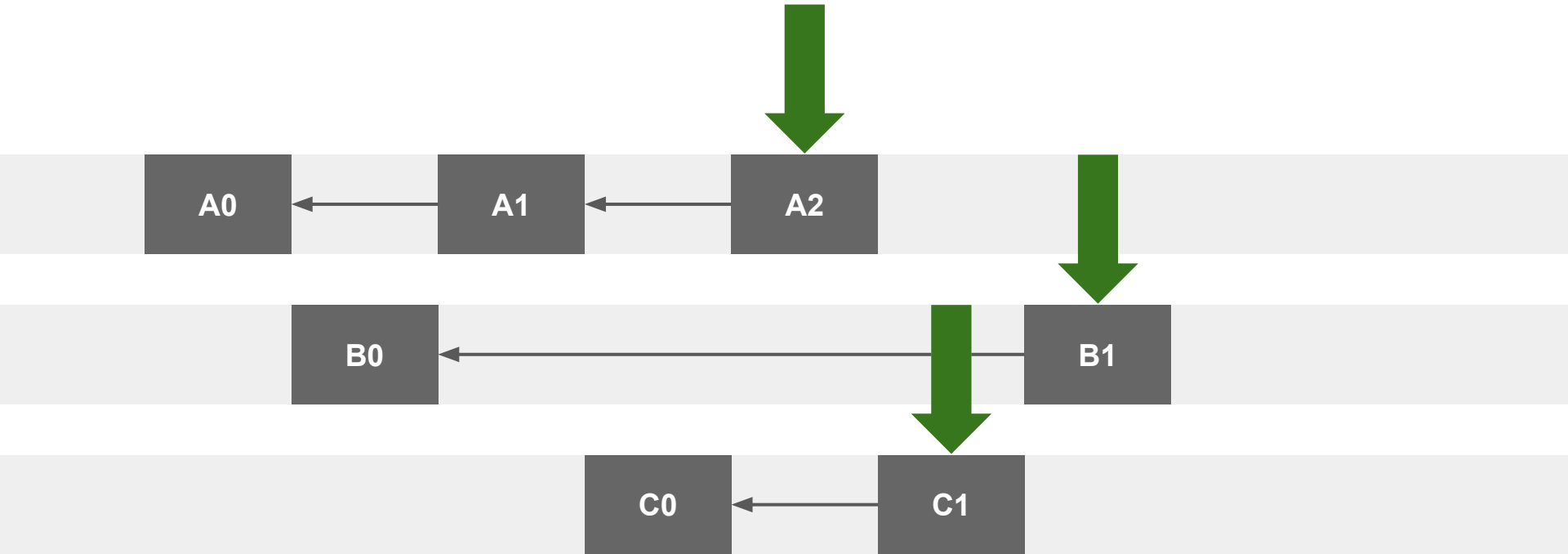
- relevant package state
- that the registry committed to that state
- the commitment is valid / correct

Clients know nothing at the start

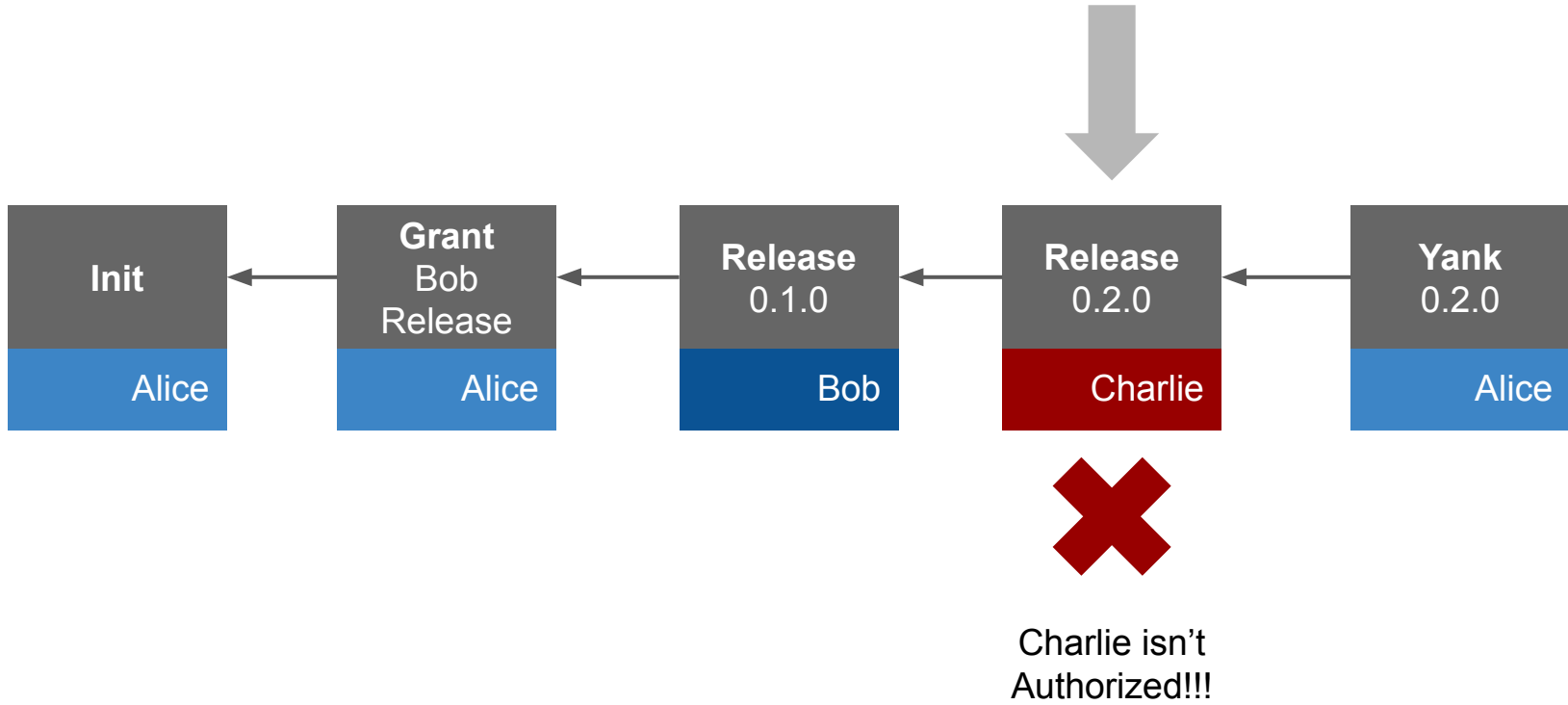
Clients **download** the Package Logs



Clients **verify** Package Logs

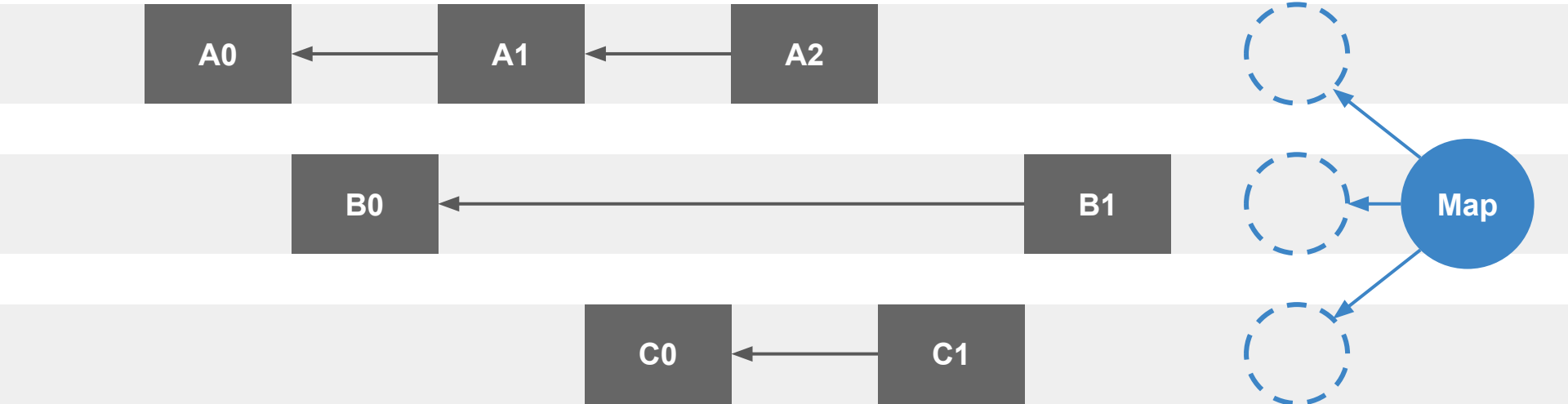


Clients **verify** Package Logs

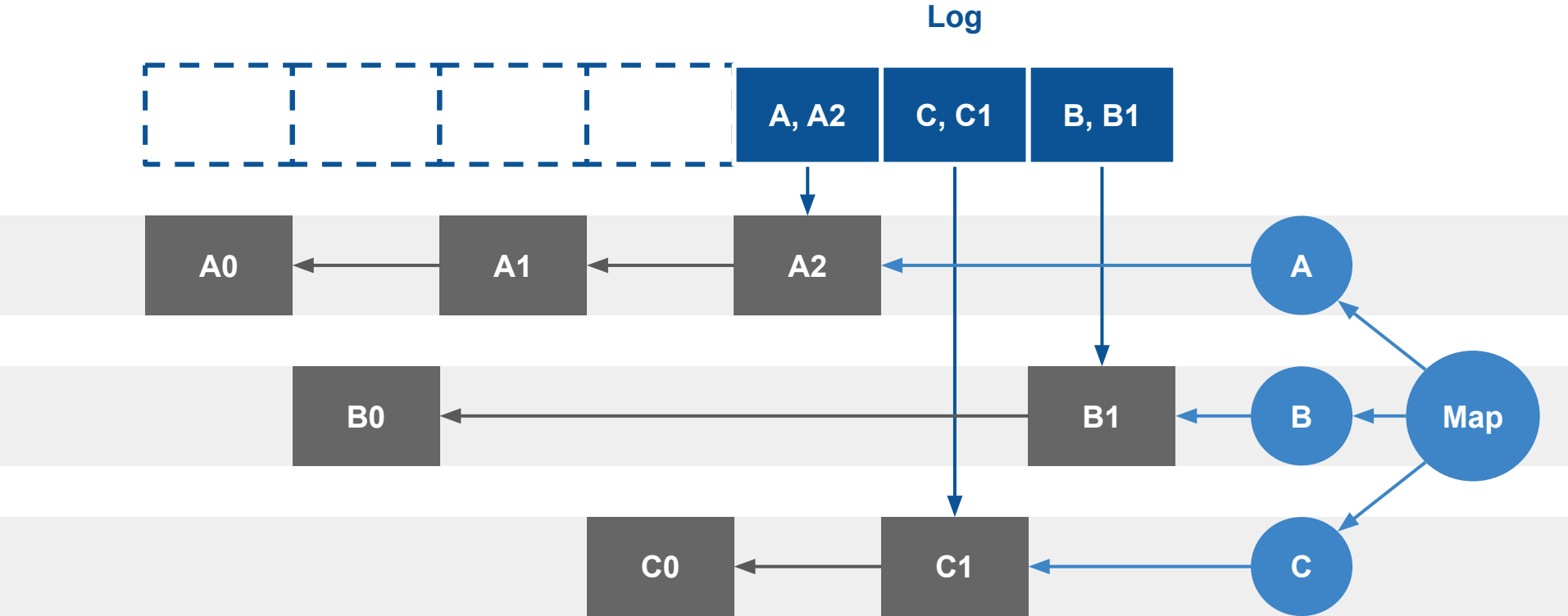


Clients **get** checkpoints for the map and log

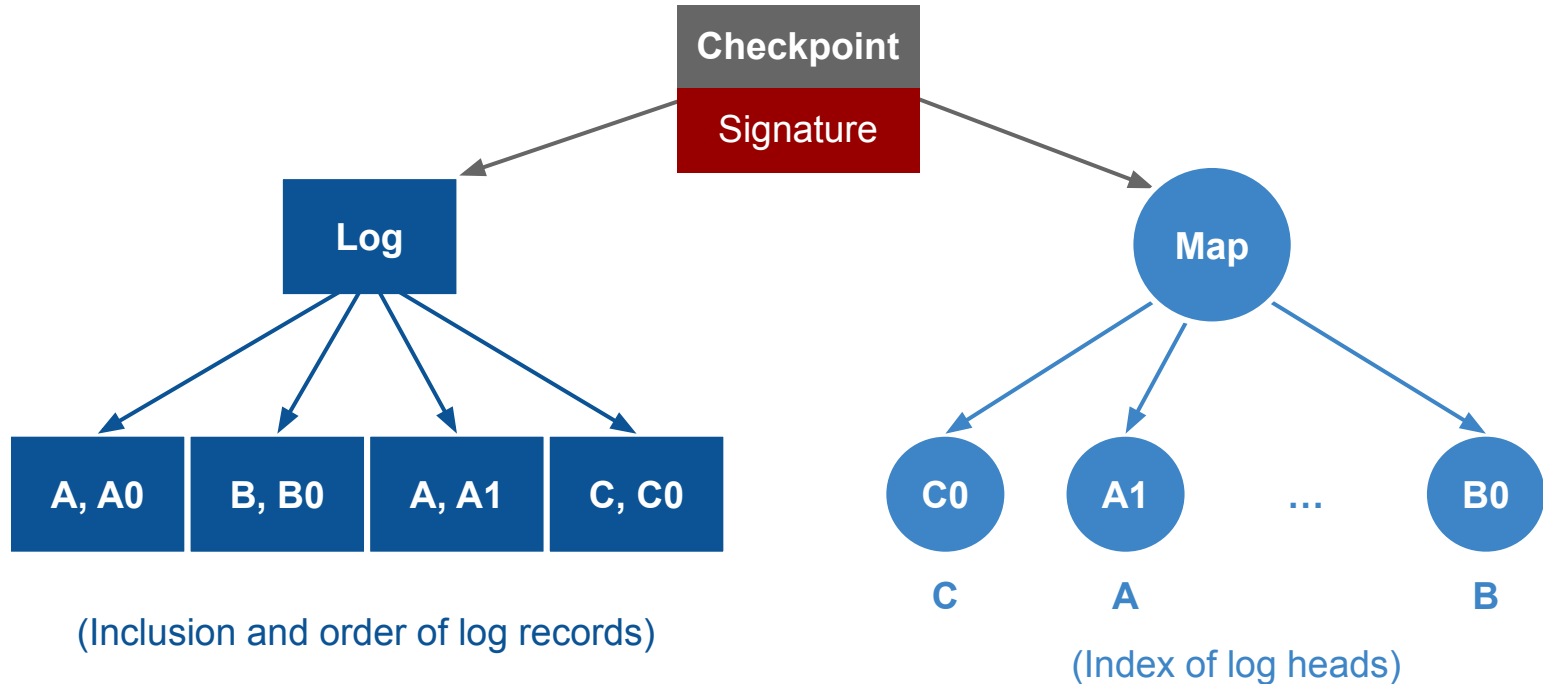
Log



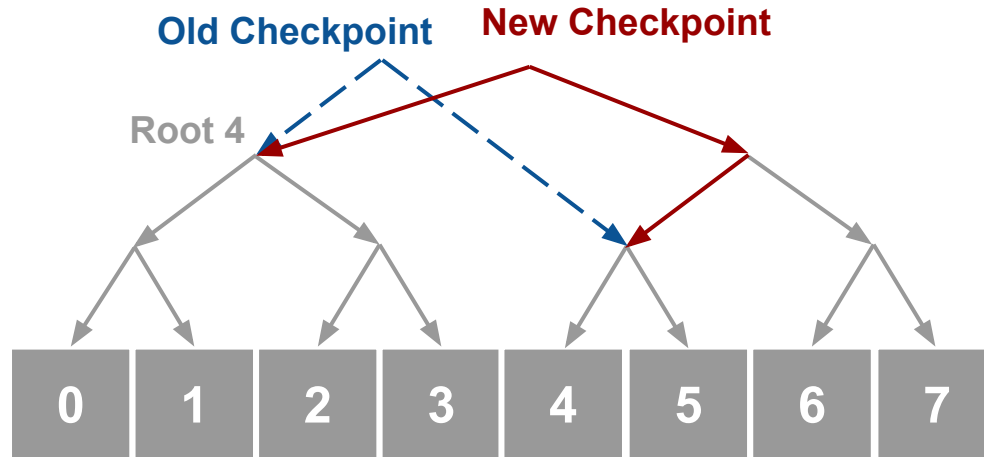
Clients **get** & **verify** proofs for map and log inclusion



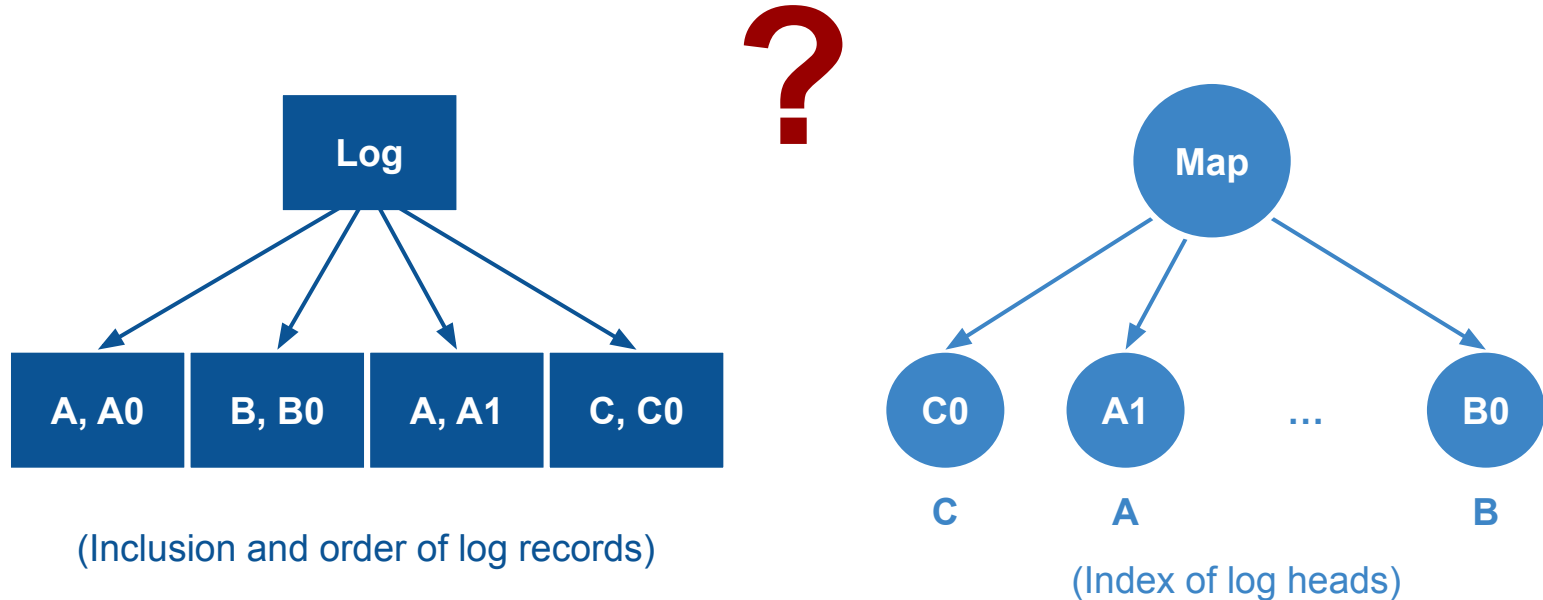
Clients **validate** Checkpoint Signature



Client Validation - Log Checkpoint Consistency



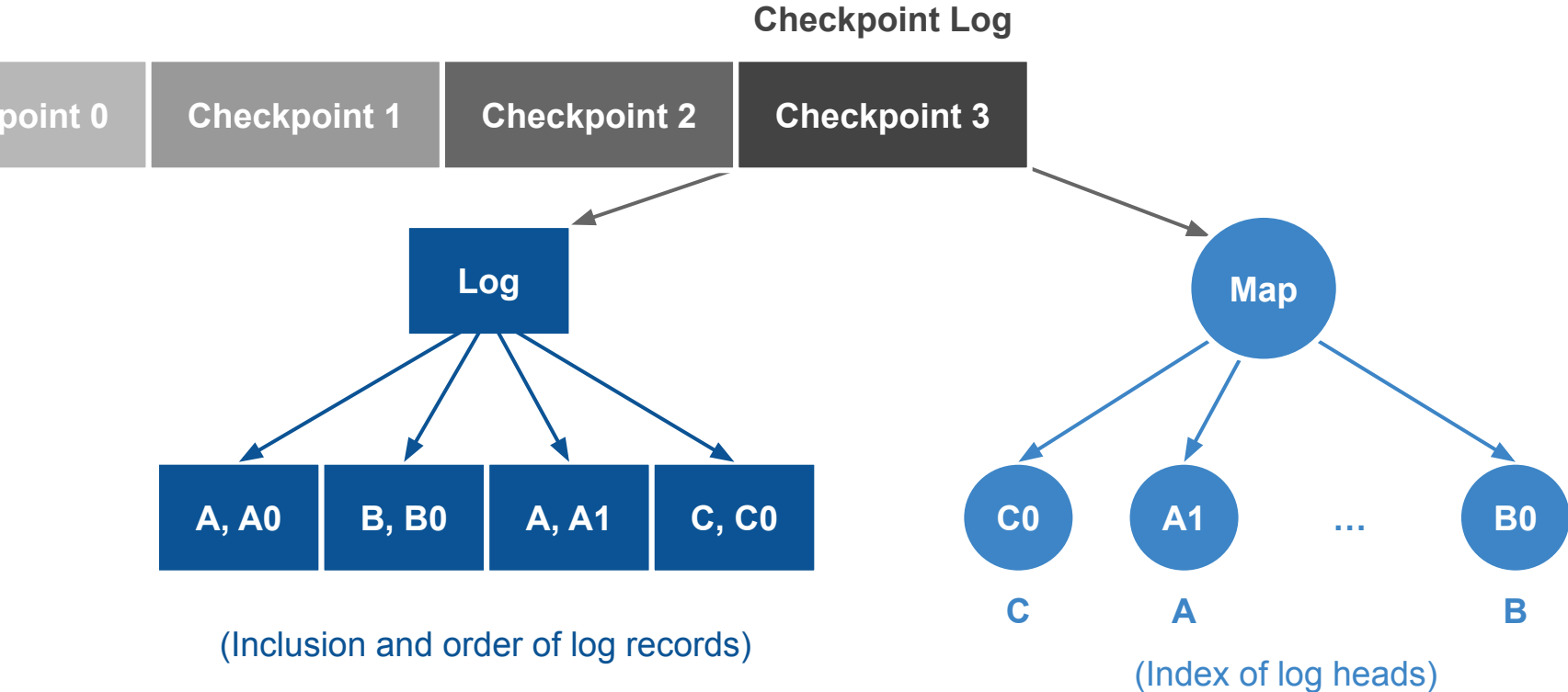
Client can't validate consistency of the **Log** & **Map**



Who are clients
gonna call?

Monitors

Monitors Audit Log/Map Consistency Over Time



Components of Package Transparency (Revisited)

1. Publicly Available Package Registry State

- A collection of package logs

2. Cryptographically-Verifiable Commitments

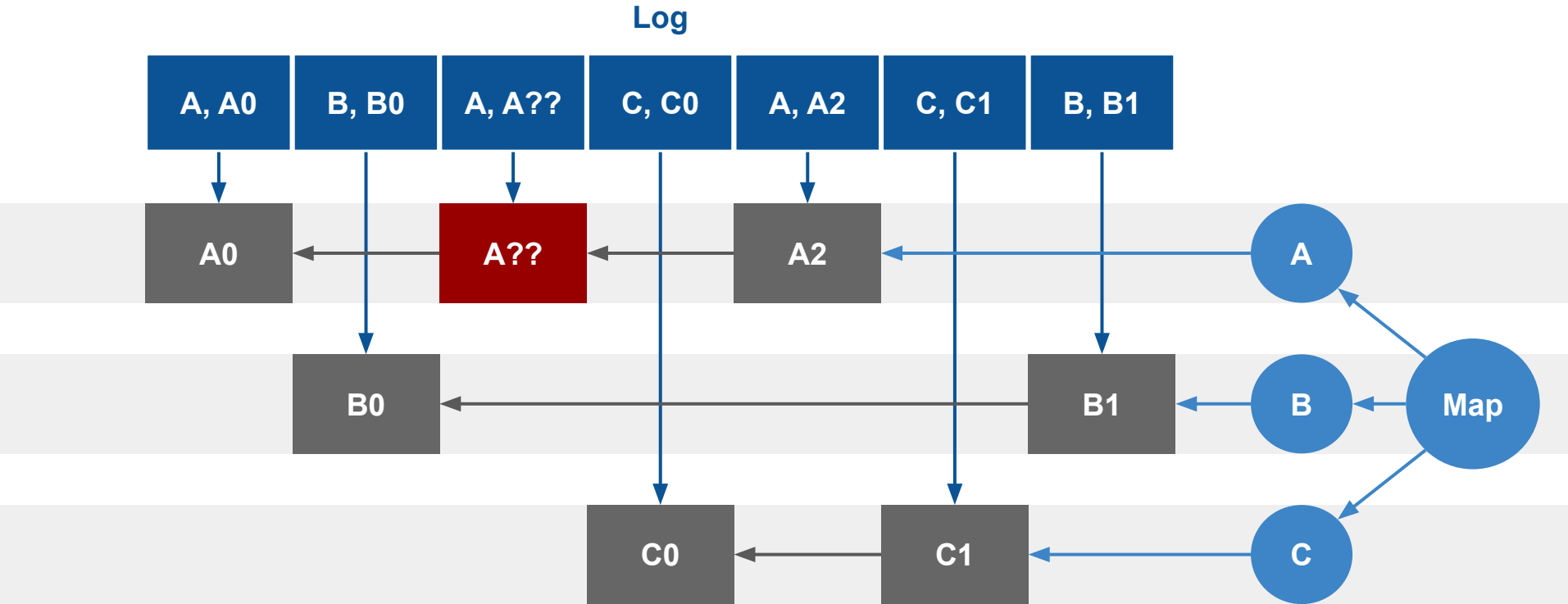
- Signed log + map checkpoints

3. Auditing Package Authors and the Registry

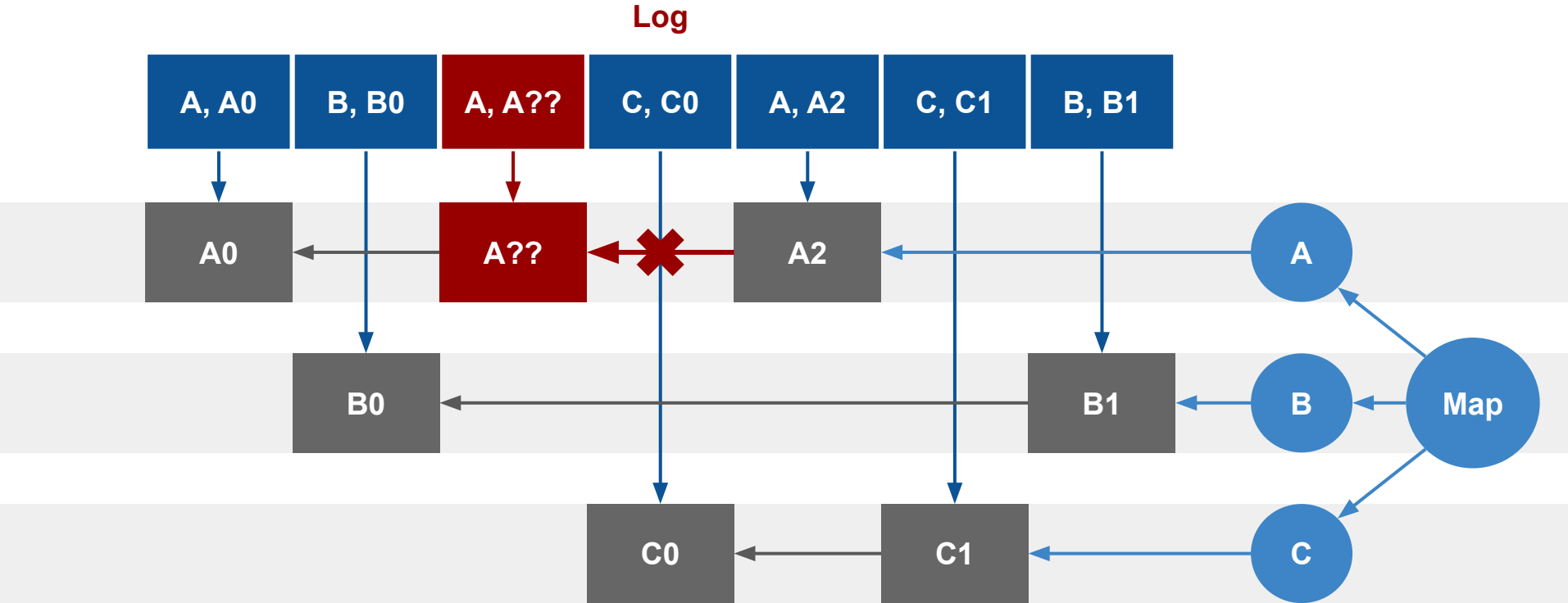
- Client and monitor validation

Package Transparency vs. Various Attacks

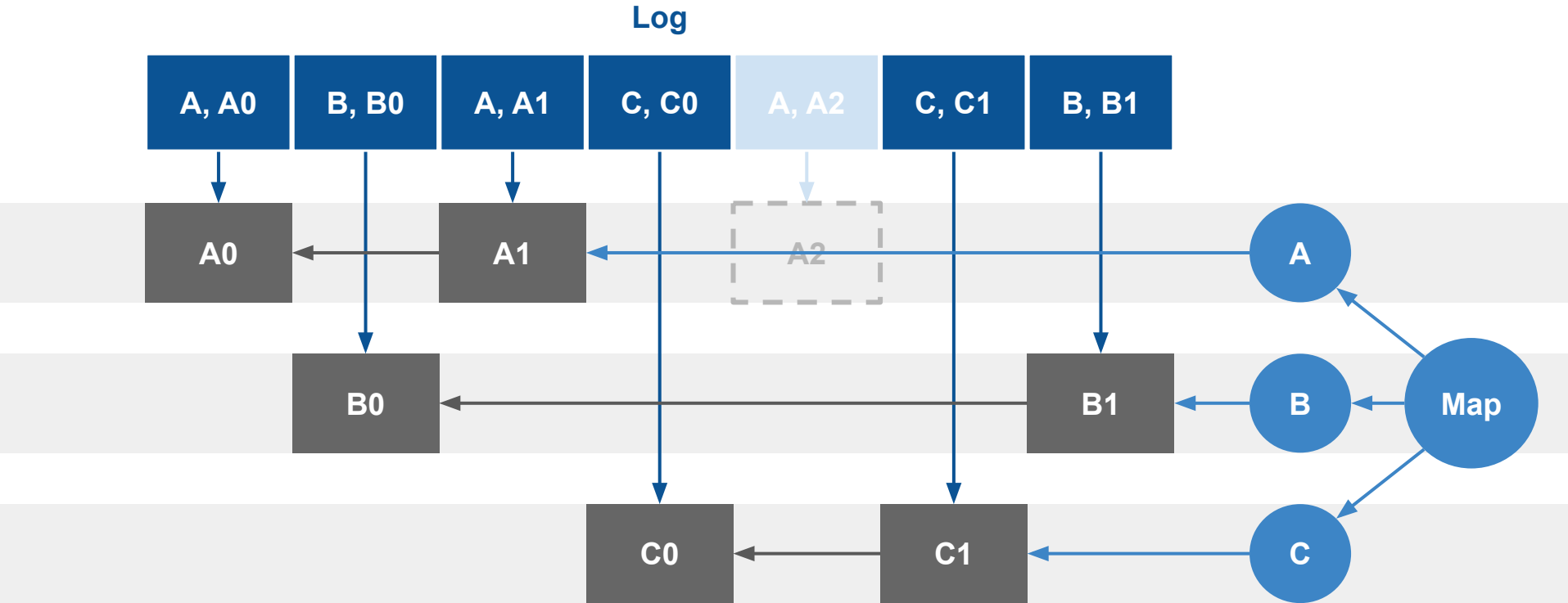
What if an attacker changes a record?



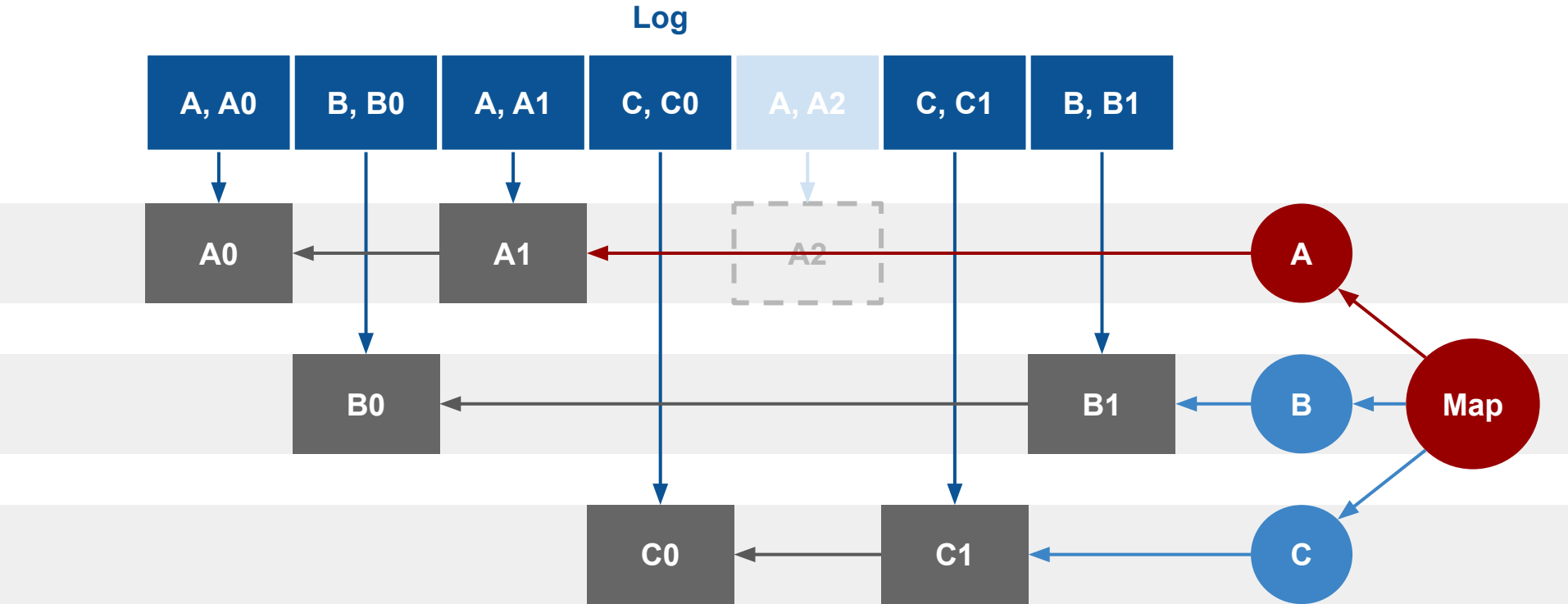
Internal hash linking will fail! (and log inclusion too)



What if an attacker hides new record(s)?



The map checkpoint will no longer match!



Summary

- **WebAssembly (Wasm)** is a promising way to make *portable and secure* software
- **Package Registries** are *indexes of content*, not necessarily content stores or providers
- **Package Transparency** builds on the the ideas of *Certificate Transparency* to offer registries valuable properties
- **Package Transparency** provides *defenses* against a variety of *attacks*

Special Thanks to

- Folks who reviewed / advised this presentation,
 - [Lann Martin](#), Fermyon
 - [Luke Wagner](#), Fastly
- Other contributors to the **warg** project,
 - [Bailey Hayes](#), Cosmonic
 - [Peter Huene](#), Fastly
- and finally **SingleStore**
 - for supporting **warg** by enabling me to contribute to this project