

Project Hydra

Phases 2 & 3

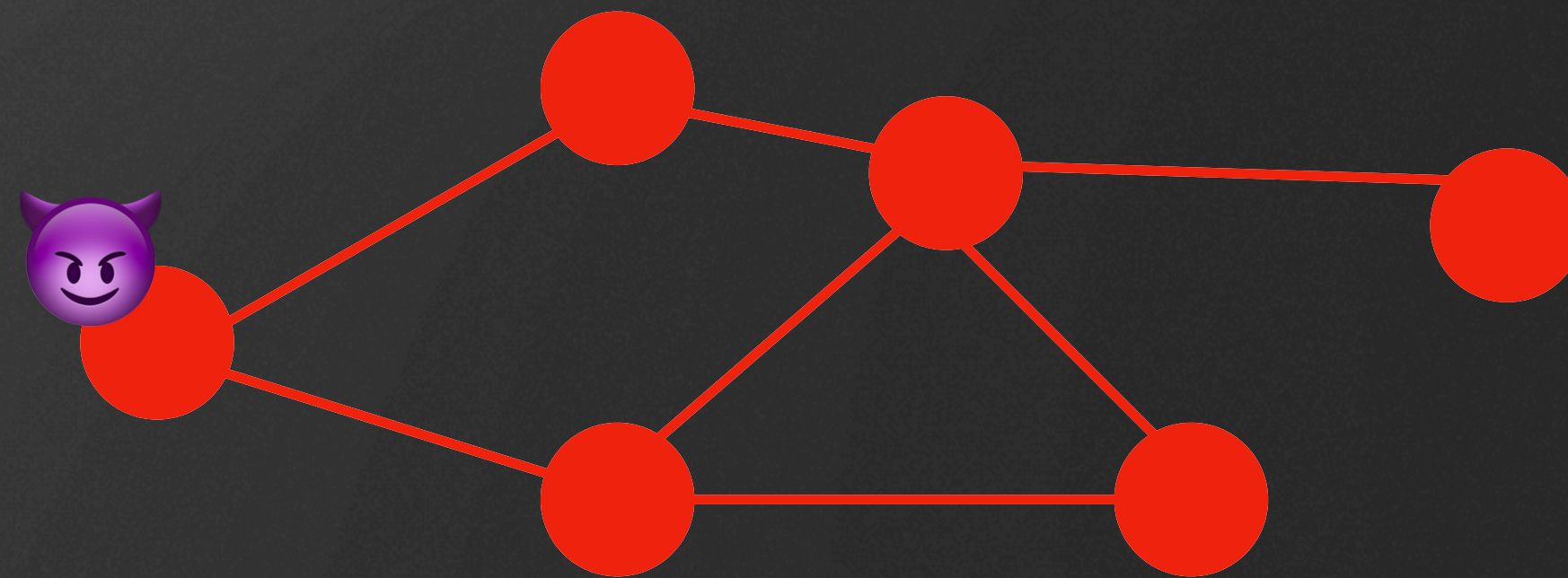
Kegan Dougal [matrix]

What is Project Hydra?

- An initiative to improve the security of the federation protocol
- Phase 1 finished last year:
 - MSC4297: State Resolution v2.1
 - MSC4291: Room IDs as hashes of the create event
 - MSC4289: Explicitly privileged room creators

Why should I care?

- A core goal of Matrix is to support a secure open federation: without gatekeepers restricting which servers can talk to which other servers.
- This means the protocol **MUST** deal with malicious and faulty servers over federation.



Phase 2

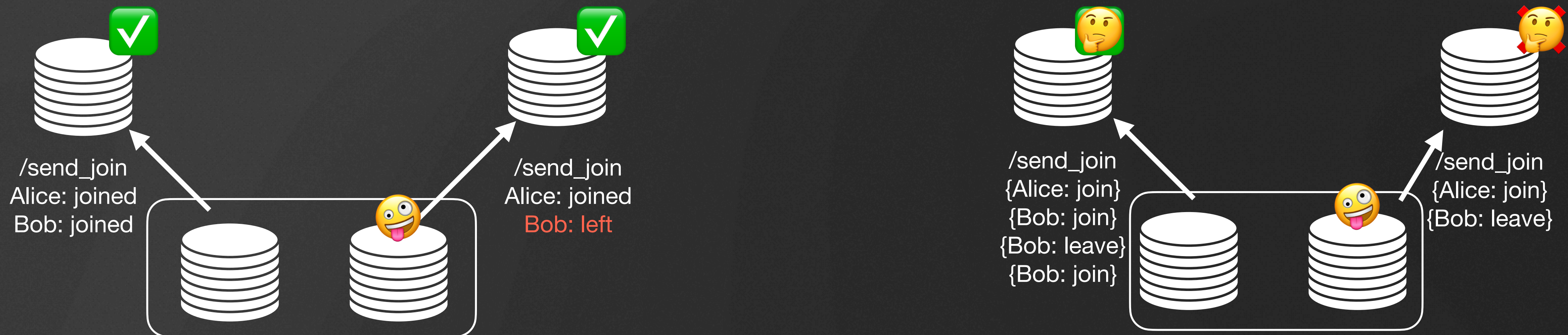
Aim: ensure all servers have consistent views of the room

- Problem: Member counts in rooms can diverge in very large rooms, for all sorts of reasons. The aim of Phase 2 is to fix this.
- Proposals:
 - MSC4428: Stable Identifiers for Room Members
 - MSC4430: Member Keys
 - MSC4242: State DAGs

Phase 2

How do the proposals ensure consistent views?

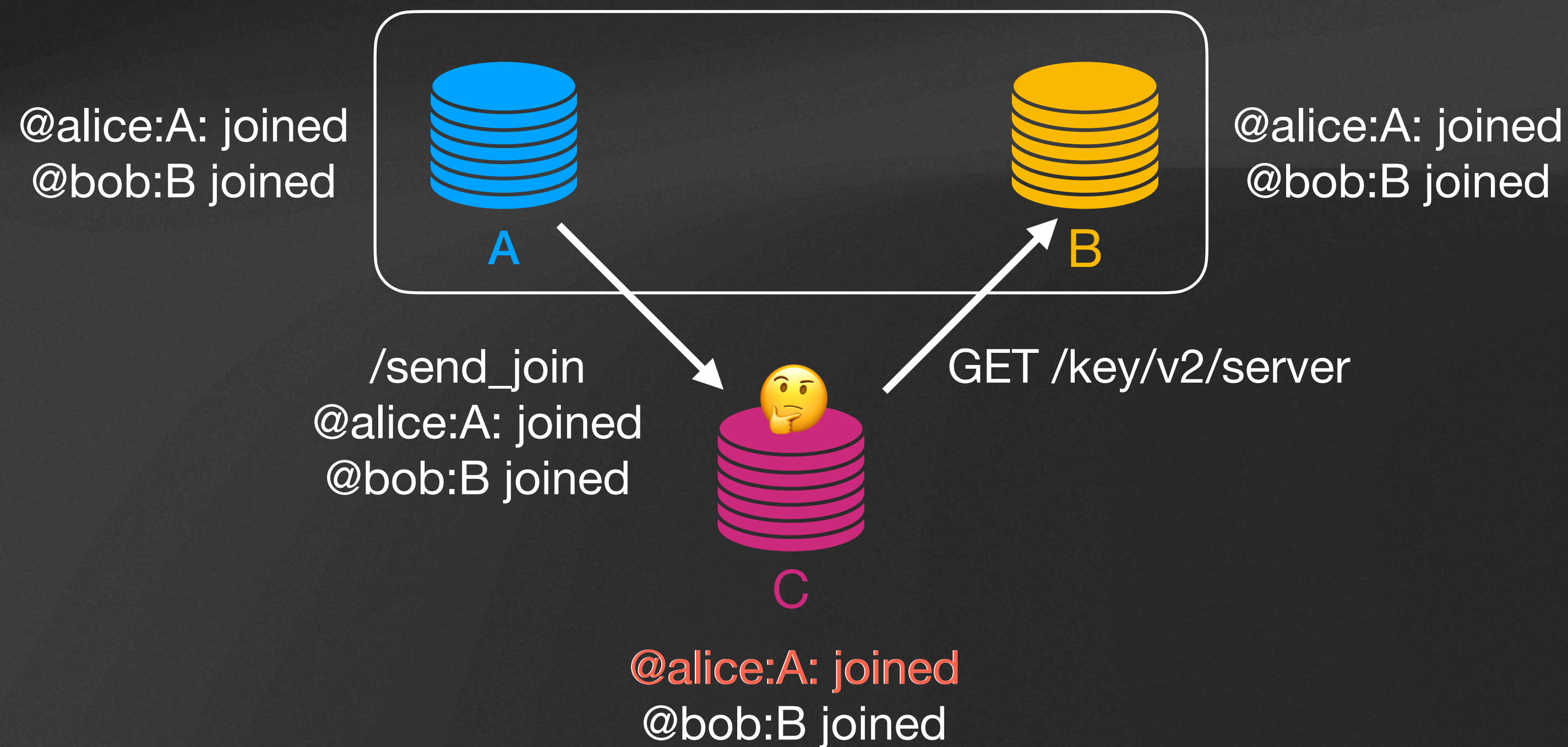
- Today, servers *partially* synchronise room state by asking for a **snapshot** of the room state at some earlier point in time. Buggy servers may return the wrong snapshot causing state to diverge, which is unverifiable.
- MSC4242: State DAGs is a **complete** synchronisation model, removing the need to ask other servers for snapshots entirely. The DAG verifies that the correct events were returned.



Phase 2

How do the proposals ensure consistent views?

- Today, events which fail signature checks are dropped. If servers cannot fetch server keys, they will drop events that other servers have not dropped.



Phase 2

How do the proposals ensure consistent views?

- MSC4430: Member Keys removes the need for server keys entirely for event signature checks. It adds a layer of indirection to determine the sender of an event.
- MSC4428: Stable Identifiers for Room Members provides a mechanism for clients to understand the layer of indirection and map from Member Keys to User IDs.
- This is a  **major breaking change**  to the protocol. Clients will need to opt-in to see these rooms.
- ..and introduces an insecure form of account portability.

Phase 3

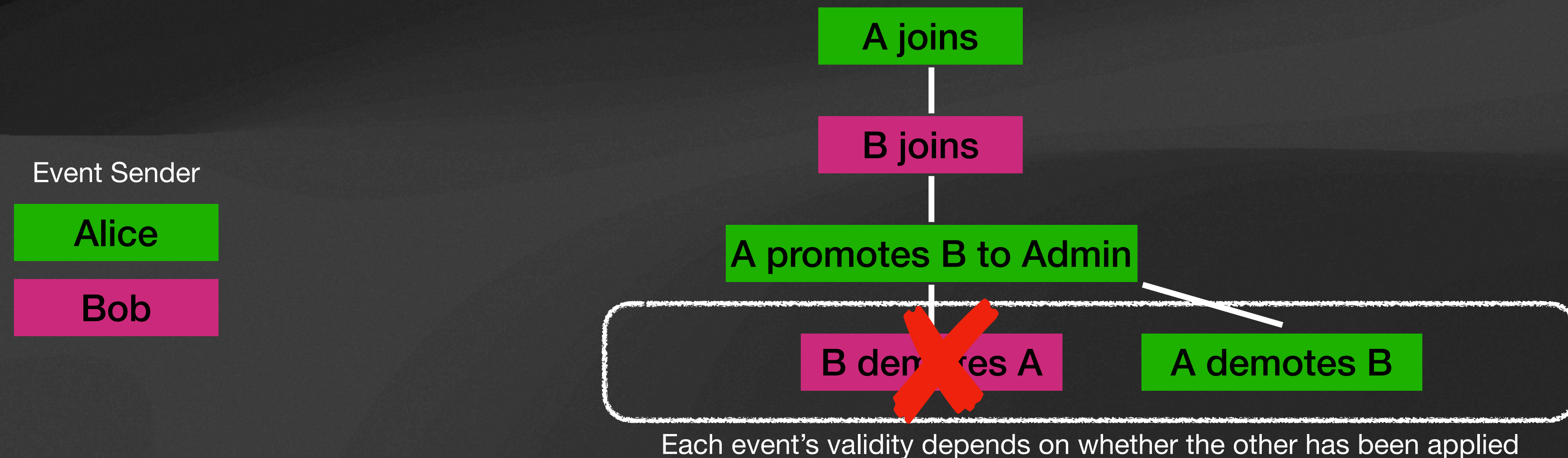
Aim: backdating protection

- Problem: The protocol must be able to rollback changes that later turn out to be unauthorised because the user making the change was concurrently removed... but this rollback must not be weaponised to allow for backdating.
- Best outlined in the “Duelling Admins” problem.
- The aim of Phase 3 is to fix this by introducing “finality”.

The Duelling Admins Problem

How do you agree who is authorised to do actions?

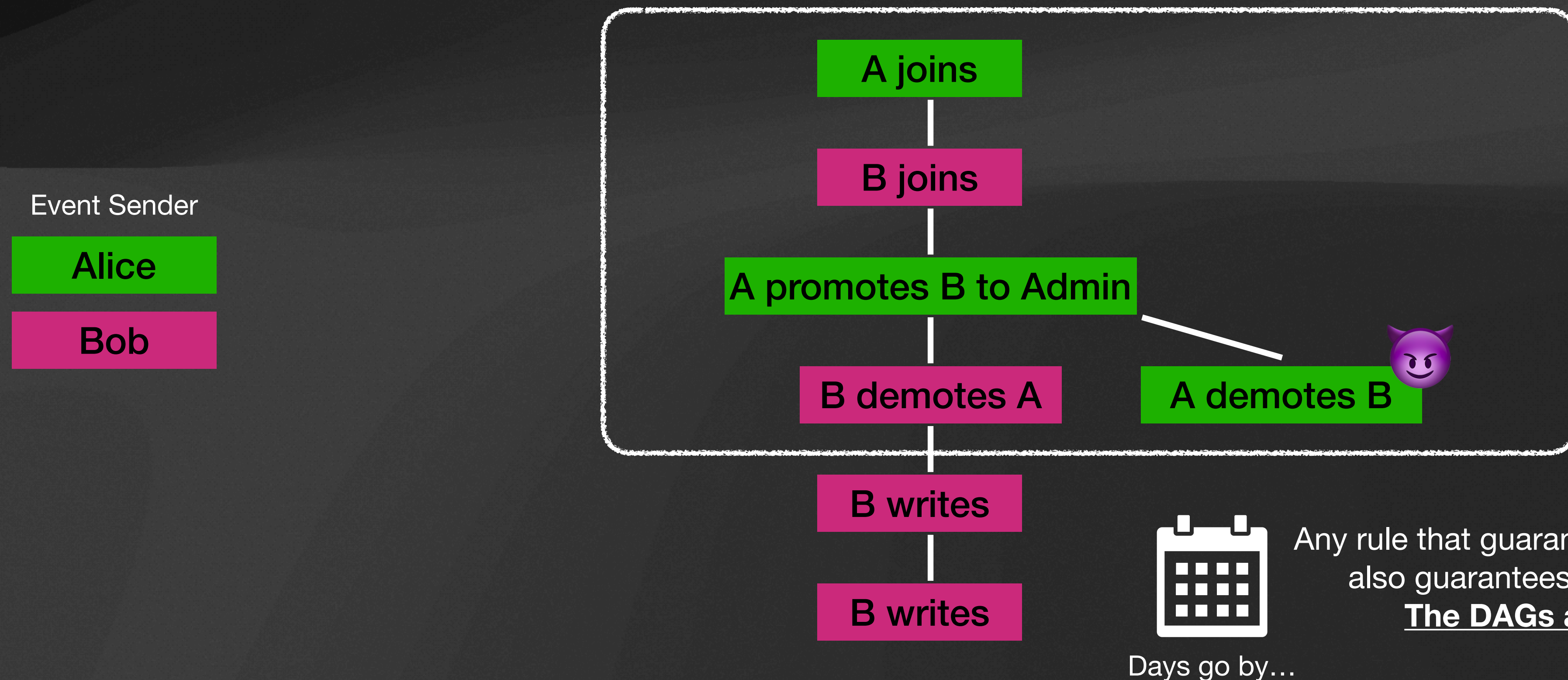
When authorisation is embedded in the DAG, concurrent demotion operations can conflict with each other.



Execution Order: A joins → B joins → A promotes B to Admin → ??? → ???

The Duelling Admins Problem

When authorisation is embedded in the DAG, concurrent non-monotonic operations (demotions) can conflict with each other.



Any rule that guarantees A wins in the legitimate case also guarantees A wins in the retaliatory case.
The DAGs are structurally identical.

Can't we just...

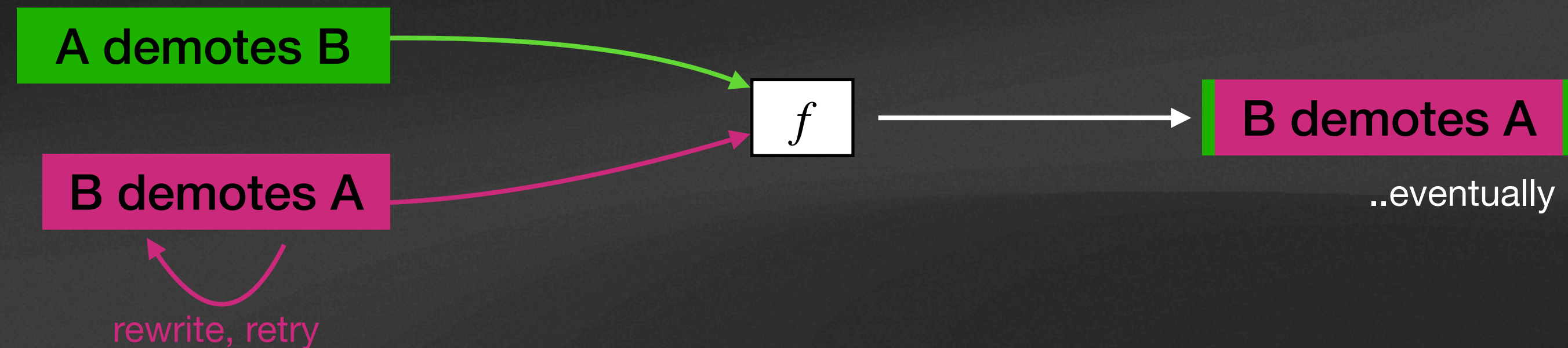
- You can't predict hashes! Use the event ID? => Hash-grind
- The most senior user wins! => The most senior user can always do this attack
- Why order events at all? Concurrent specifications! => Retaliation / Locked-out
- Consensus! => Users can backdate sockpuppet admins to have a majority
- Proof of Work/Stake! => The user with the bigger CPU / stake can do this attack
- 🤔
- Forbid demotions? => Unrealistic in practice. Employees leave companies. Devices get compromised, business contracts end, mistakes happen.
- Forbid same-level demotions? => Admins aren't equal. Some admin *promoted* the other to Admin, so they can backdate events to invalidate that promotion event.

Phase 3

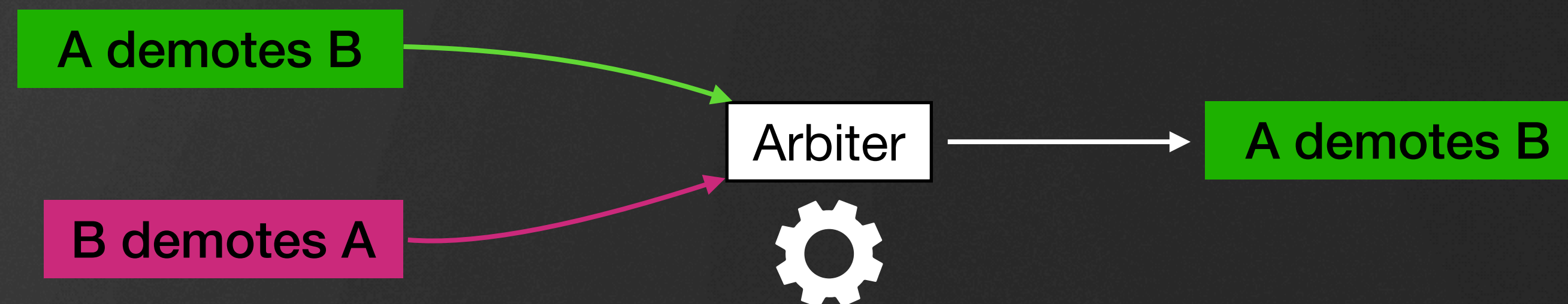
Duelling Admins: fundamental tension

- After a duel, **something** has to pick an outcome. You have two choices:

1) Let the events decide (deterministic, coordination-free, everyone computes the same answer)



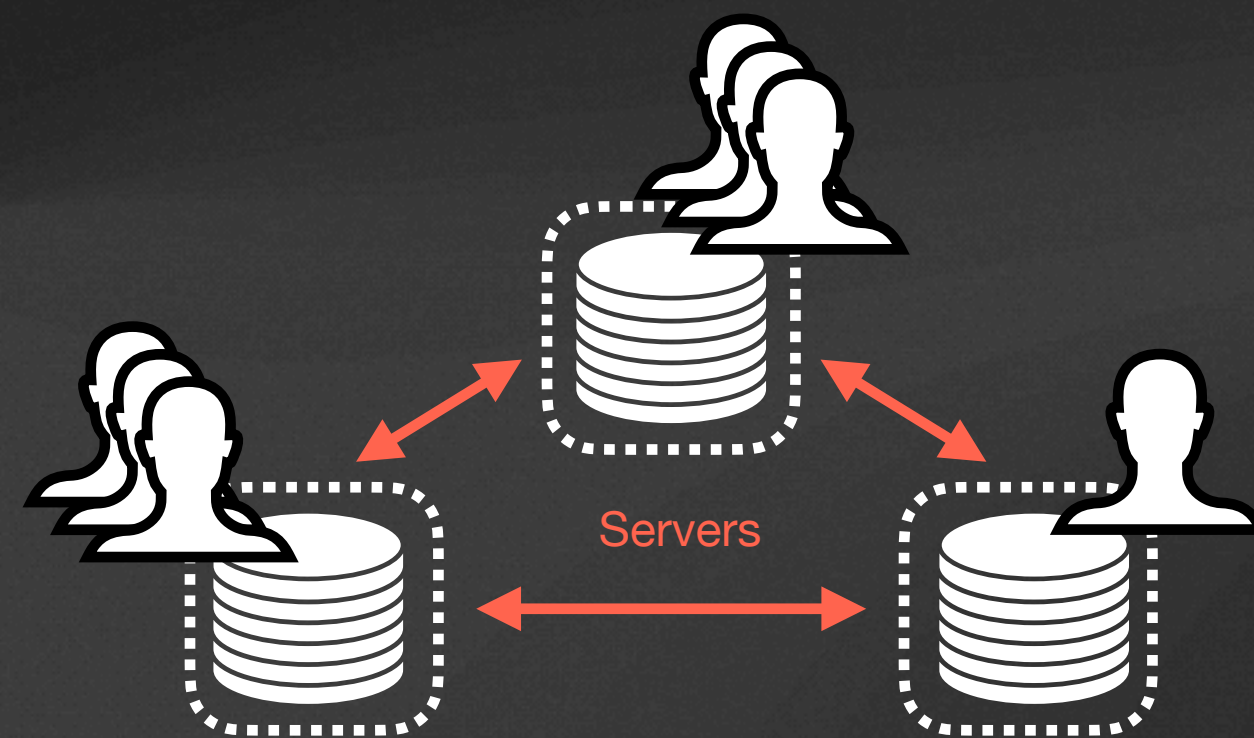
2) Let someone decide (rules out other valid futures, requires coordination)



Phase 3

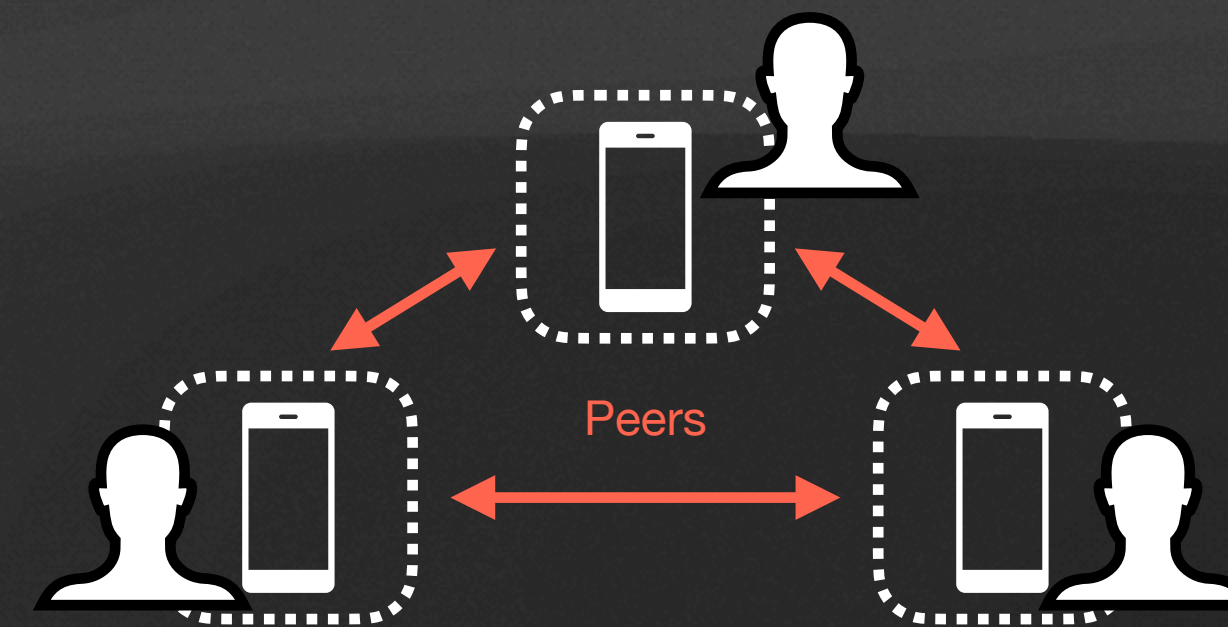
Aim: backdating protection

- This is a Hard problem.
- We're not the only ones working on this...

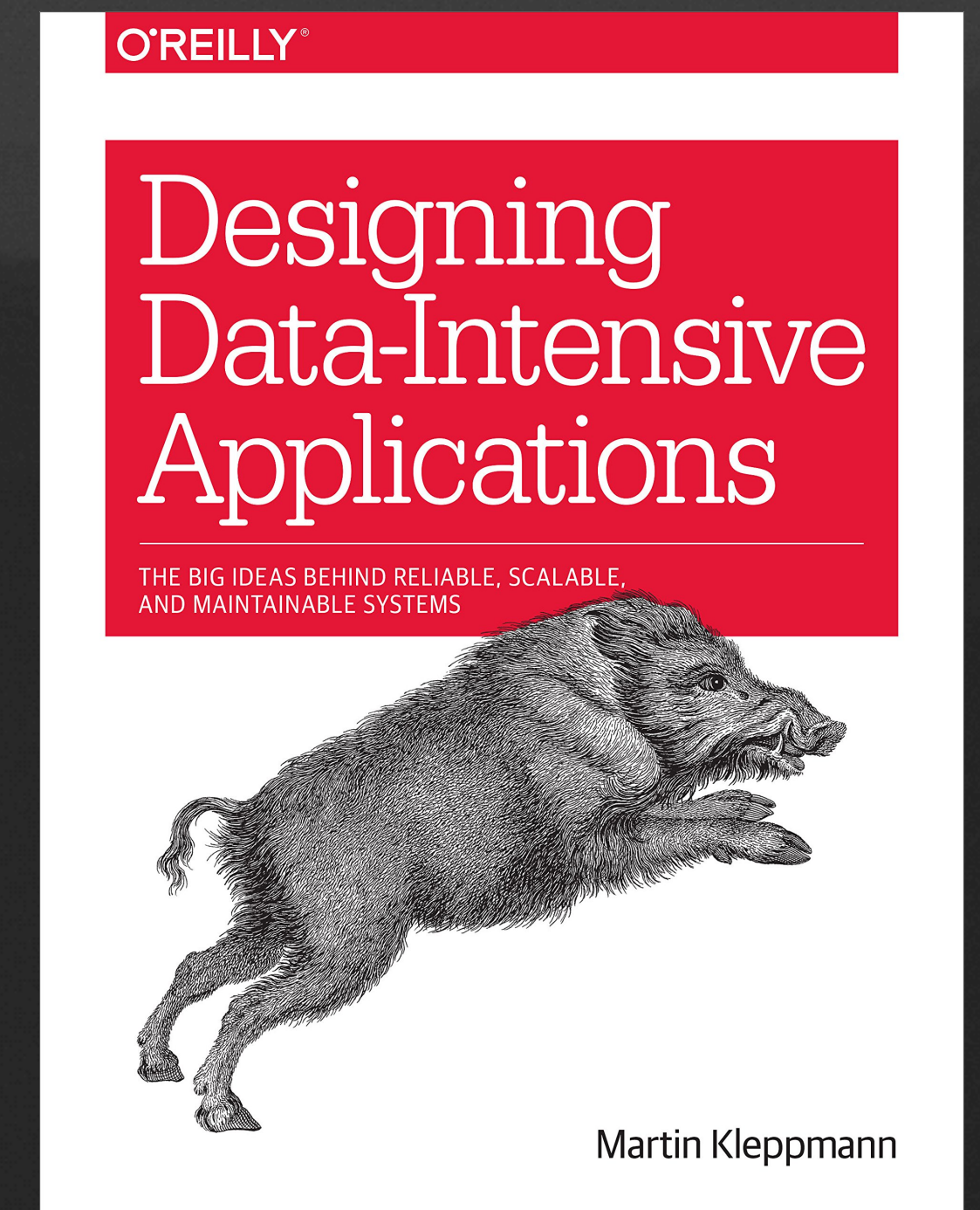


Federated

$\begin{bmatrix} m \end{bmatrix}$



Local-First



Phase 3

Aim: backdating protection

- Proposals:
 - ERA: Epoch-Resolved Arbitration
 - State Resolution v3

PaPoC 2026: 13th Workshop on Principles and Practice of Consistency for Distributed Data

ERA: Epoch-Resolved Arbitration for Duelling Admins in Group Management CRDTs

Kegan Dougal
kegan@element.io
Element Creations Ltd
London, United Kingdom

Abstract

Conflict-Free Replicated Data Types (CRDTs) are used in a range of fields for their coordination-free replication with strong eventual consistency. By prioritising availability over consistency and partition tolerance, group management CRDTs can be used in distributed systems to manage a group of users in a way that is consistent across all replicas.

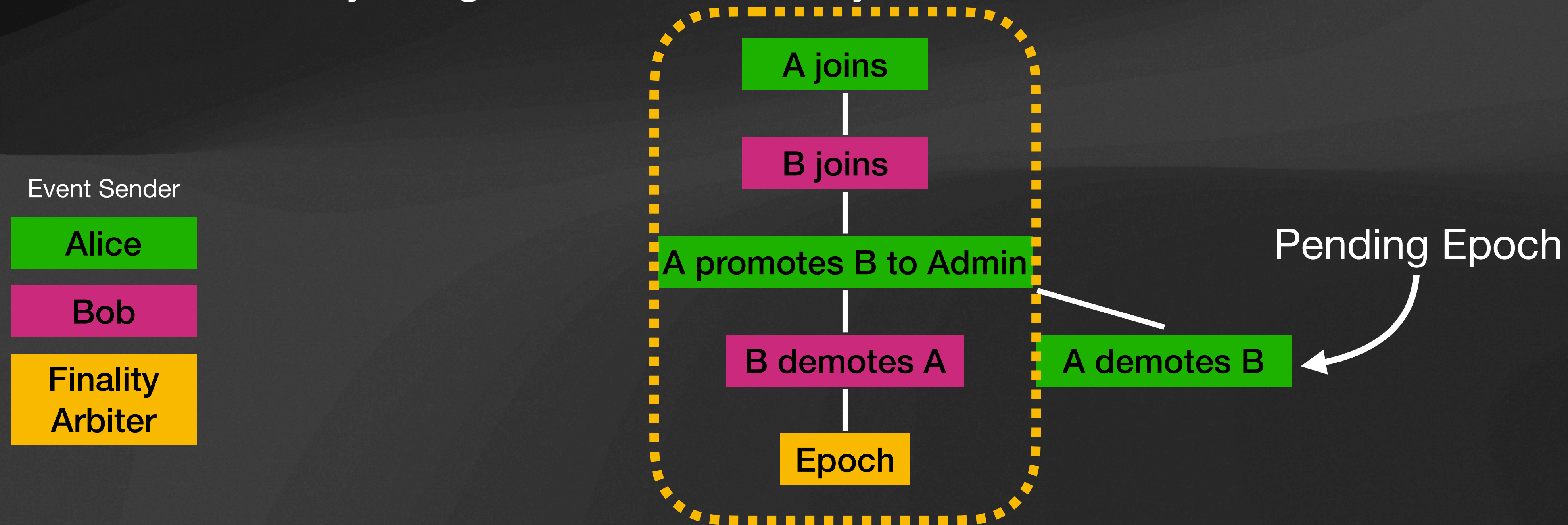
1 Introduction

Conflict-Free Replicated Data Types (CRDTs) [25] are in widespread use in a variety of fields including databases [1, 23], consumer messaging [2] and collaborative apps [26]. Many products rely on separate systems for authorising

Epoch-Resolved Arbitration (ERA)

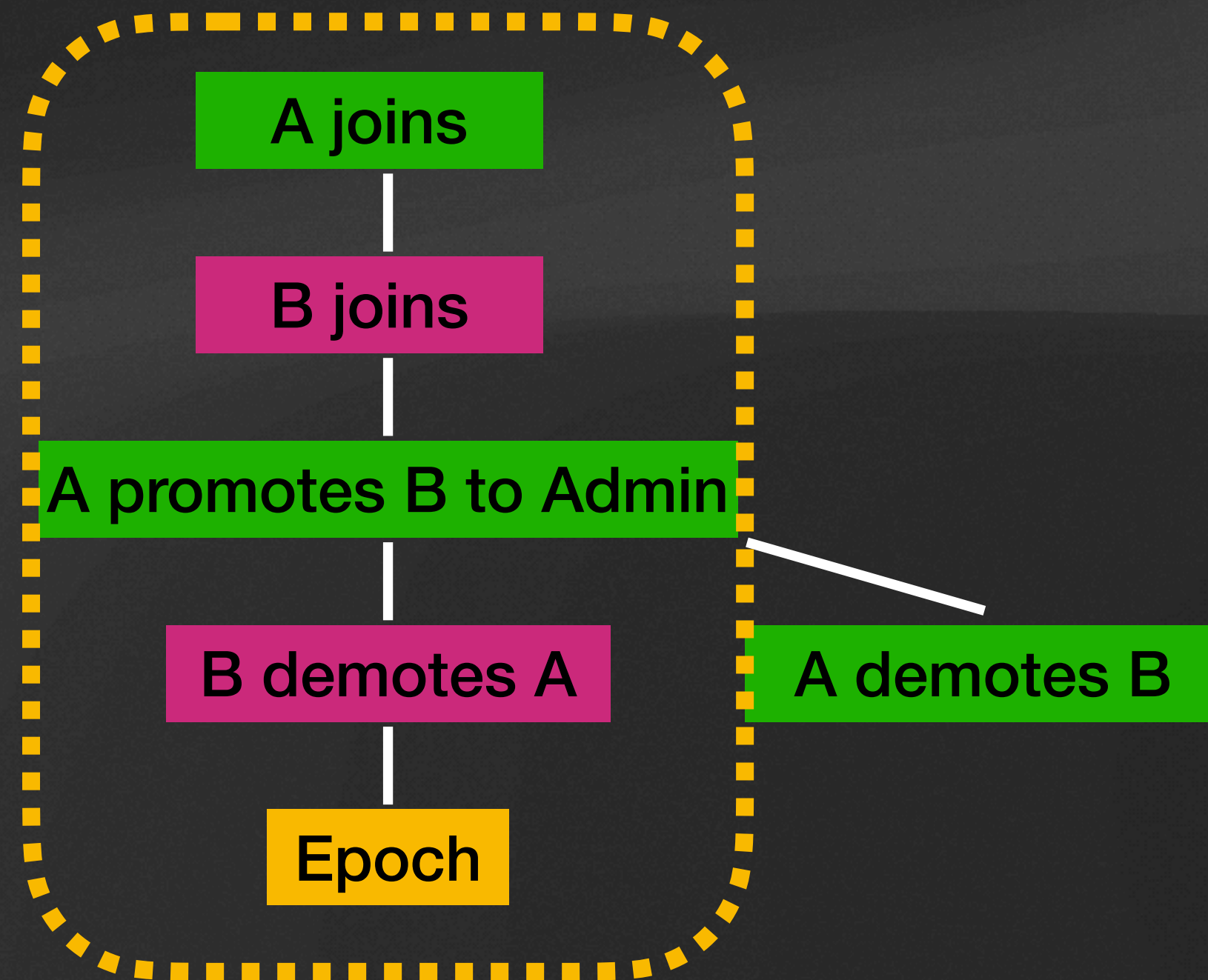
Events seen by system are ordered by the arbiter and are deposited in the epoch event log. After all closed epochs, regardless of causal predecessors.

The epoch event's hash links reference the arbiter's current heads / forward extremities: everything it has transitively seen.

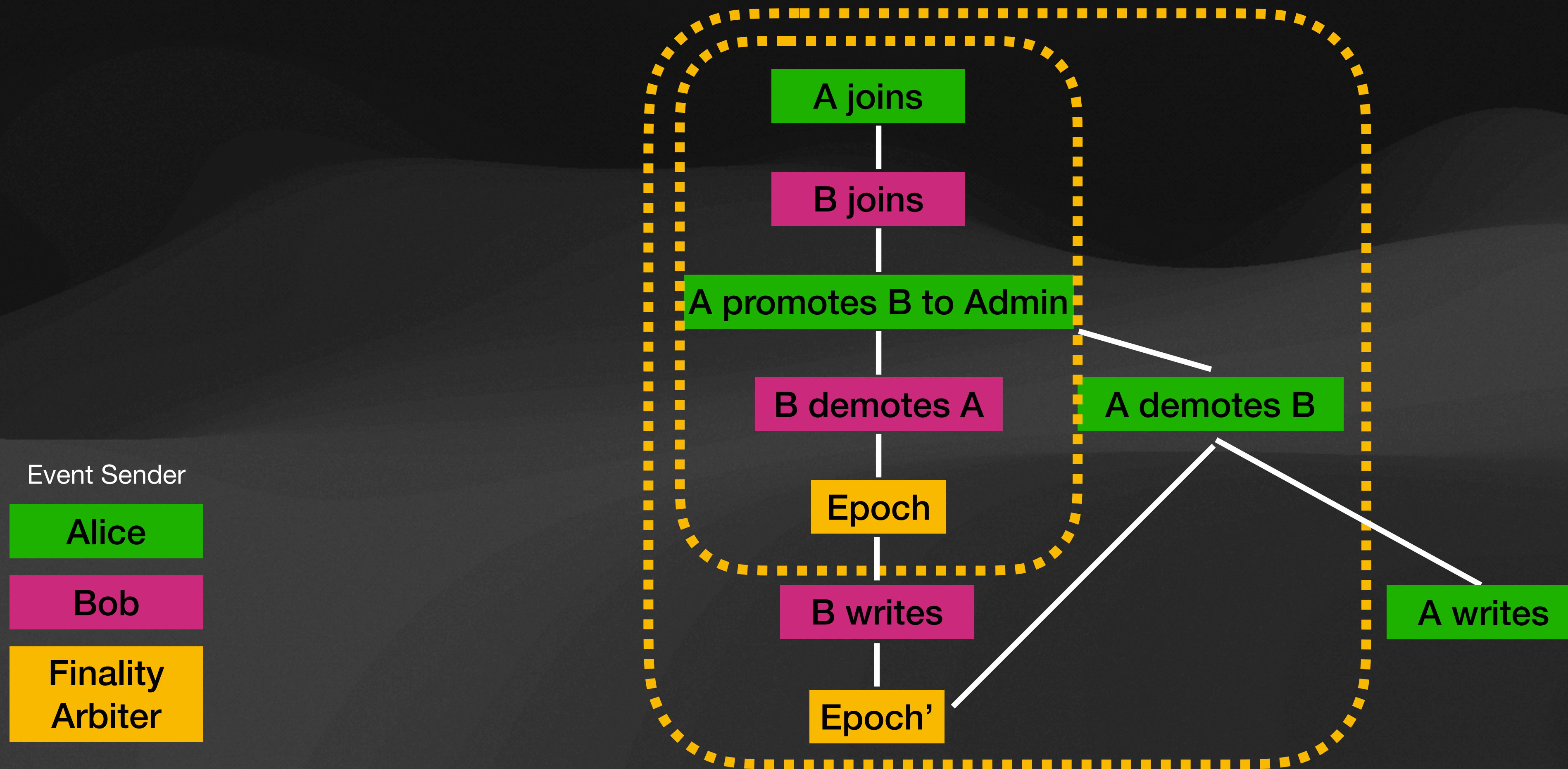


Execution Order: A joins → B joins → A promotes B to Admin → B demotes A → A demotes B

Epoch-Resolved Arbitration (ERA)

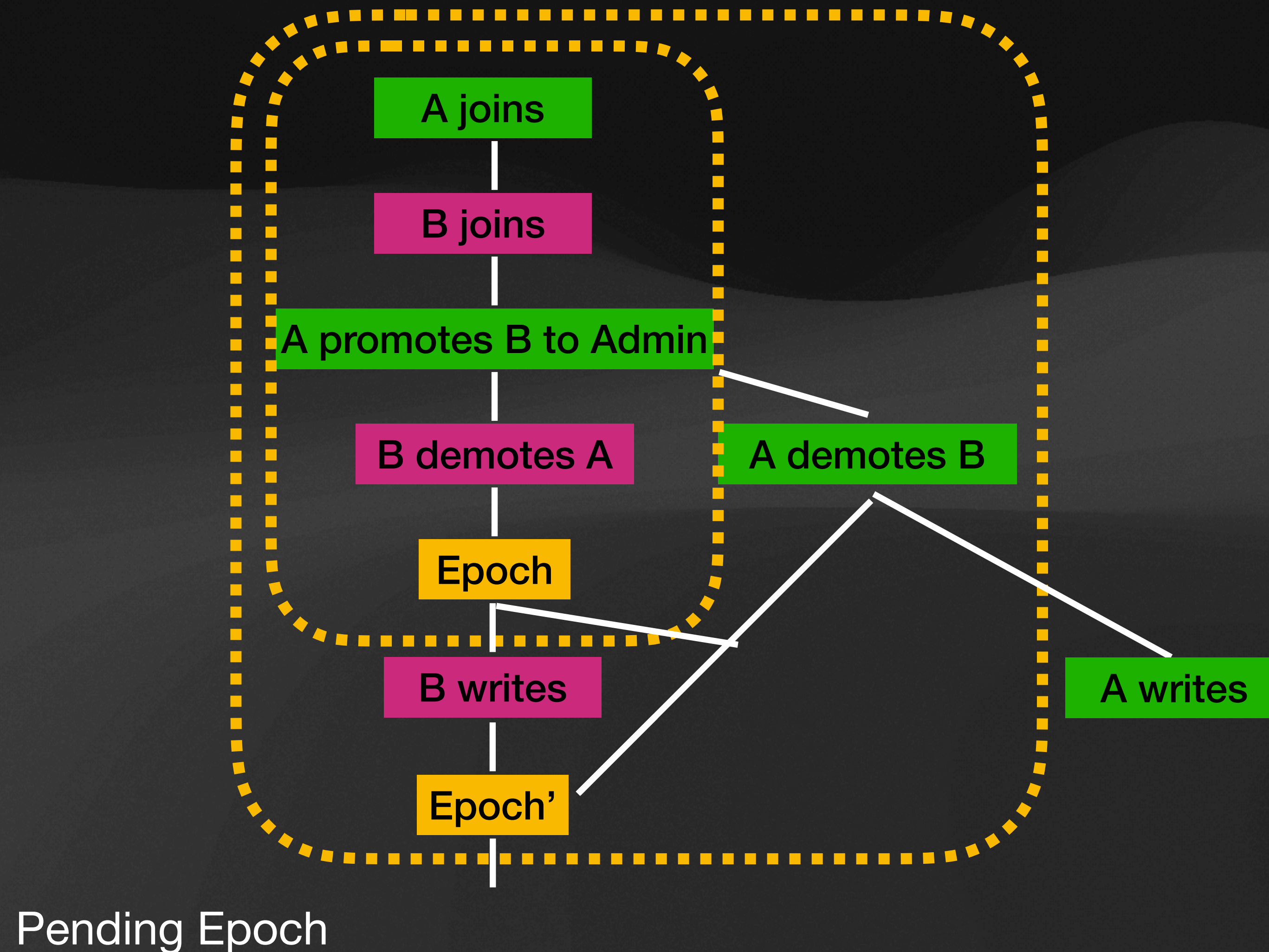


Epoch-Resolved Arbitration (ERA)



Epoch-Resolved Arbitration (ERA)

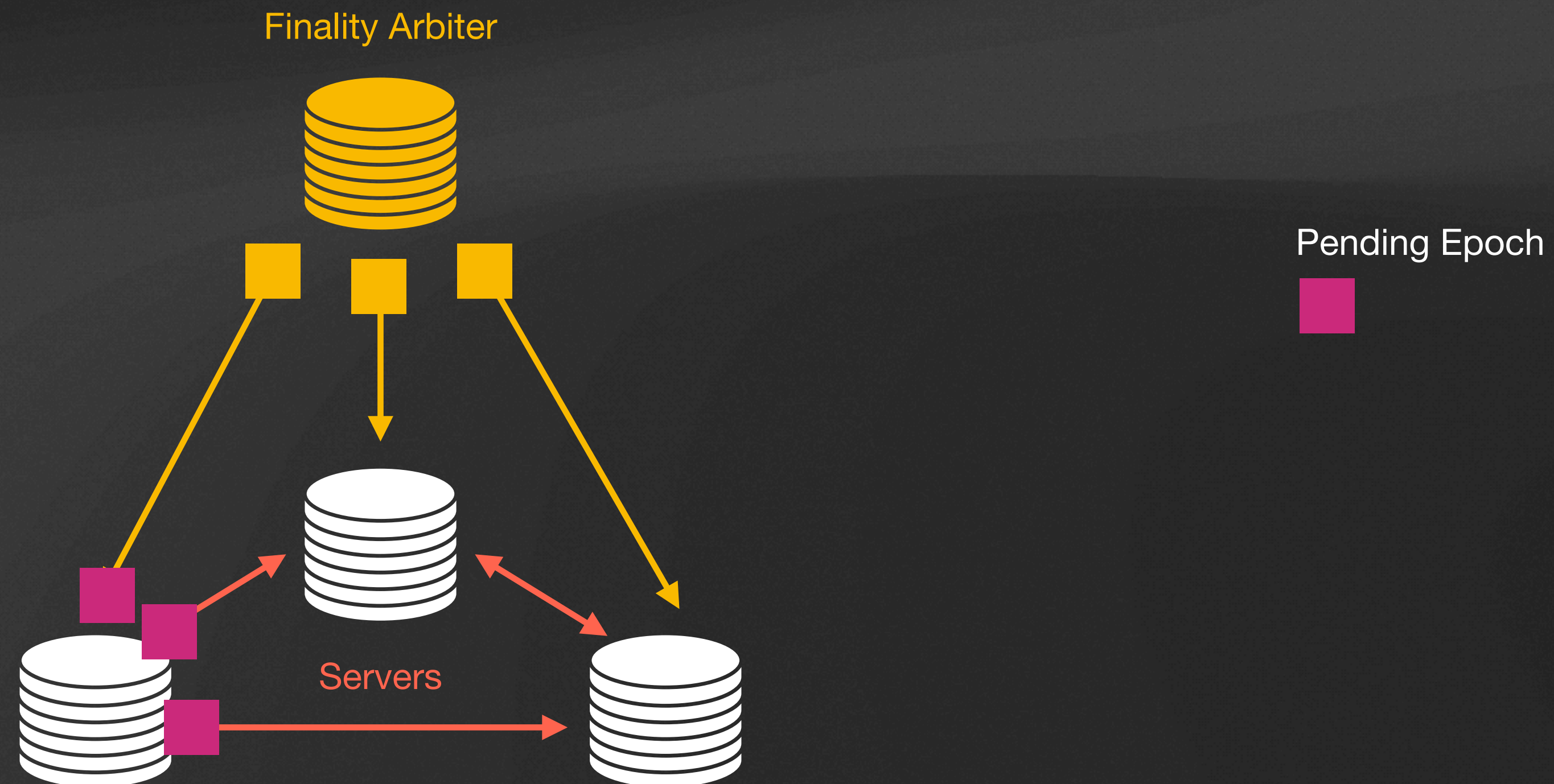
Execution order



Epoch-Resolved Arbitration (ERA)

What about availability?

The finality arbiter is a server which provides non-blocking, asynchronous sequencing.



Epoch-Resolved Arbitration (ERA)

What about availability?

If the arbiter disappears, the pending epoch grows. New events accumulate but the CRDT still functions with unstable sequencing (like Matrix today).



Epoch-Resolved Arbitration (ERA)

Isn't this centralisation?

- For availability: no. The room still functions* if the arbiter disappears.
*but any functionality we ADD to the protocol which relies on finality requires the arbiter to be available.
- For trust: partially. You trust it to create epochs honestly. It is accountable if it equivocates.
- Who is best placed to be the arbiter *in Matrix*?
 - The **creator** of the room (already trusted, simple, but if they disappear it's a problem)
 - A **group of creators** and/or **admins** (requires consensus; complex; vulnerable to Sybil attacks)

Epoch-Resolved Arbitration (ERA)

Consensus?

- Byzantine consensus has the $3f+1$ bound: **to tolerate f faulty servers, you need $3f+1$ total servers**. Does a group of creators/admins make sense given this? For 94.3% of rooms: not really.

For crash tolerance, we need at least 3 servers (so 1 can go down and still reach consensus)

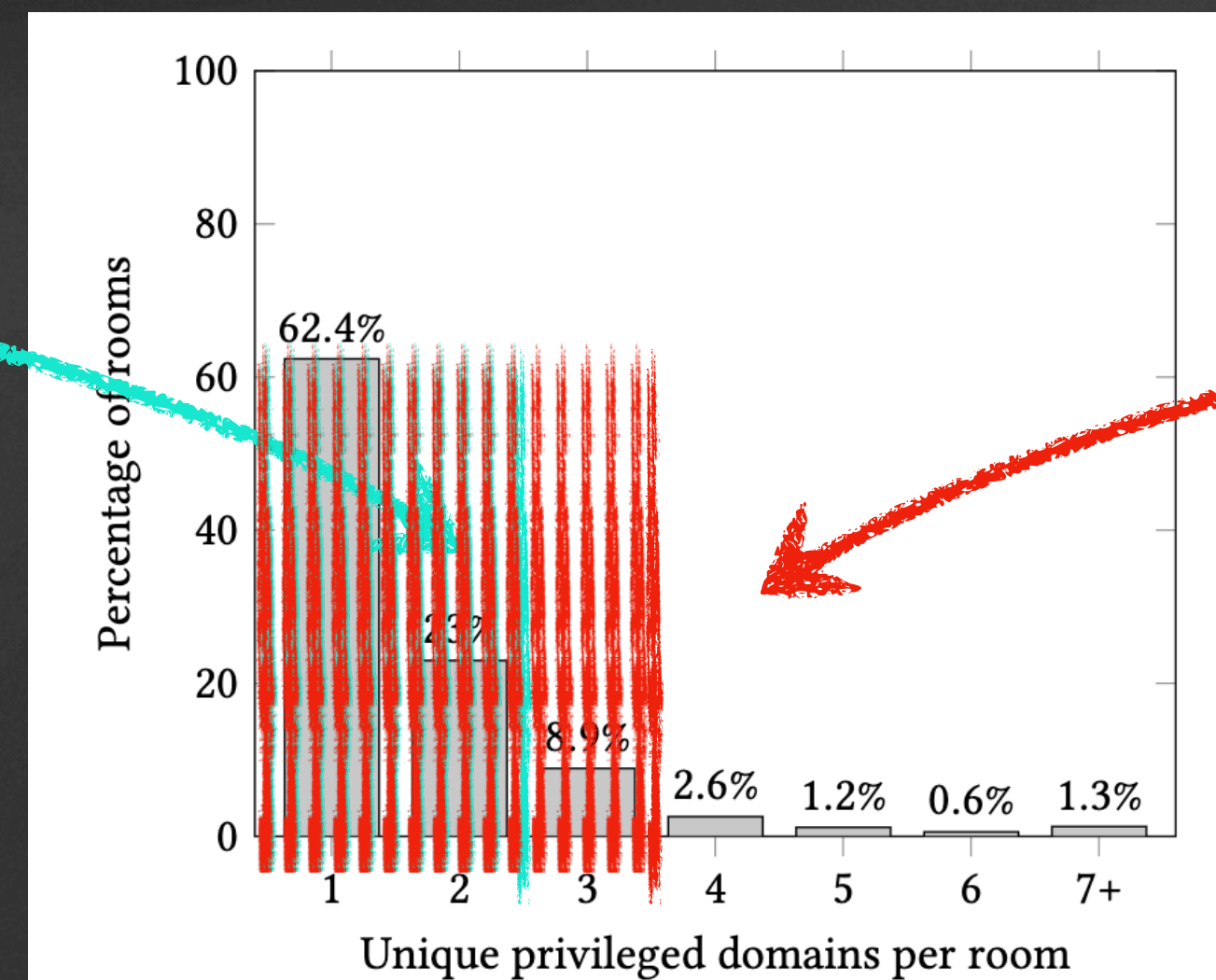


Figure 9. Domain distribution of privileged users e.g. Admins

To tolerate just 1 faulty server we need 4+ unique privileged domains, which we just don't have in the ecosystem today.

What finality unlocks

- **Backdating protection:** the point of Phase 3.
- **Ability to compact the DAG:** epochs define GC boundaries => less storage
- **A total order (depending on arbiter availability):** Opens up the possibility of layering strong consistency protocols on top of Matrix (e.g. vanilla MLS) at the cost of *requiring* the arbiter to be online. Operational requirements vary depending on who is using Matrix, so let's make it room configurable.

Conclusion

- **Phase 2 is coming.** This will have a significant impact on the ecosystem because it involves changing a fundamental data type in the protocol (user IDs)
- **Phase 3 is being researched.** The general problem is unsolvable, the tradeoffs are bleeding edge research.
- Special thanks:
 - Florian Jacob @ Karlsruher Institut für Technologie (KIT)
 - Martin Kleppmann and the Local-First community