CoCoA: Concurrent Continuous Group Key Agreement

Guillermo Pascual Pérez
ISTA

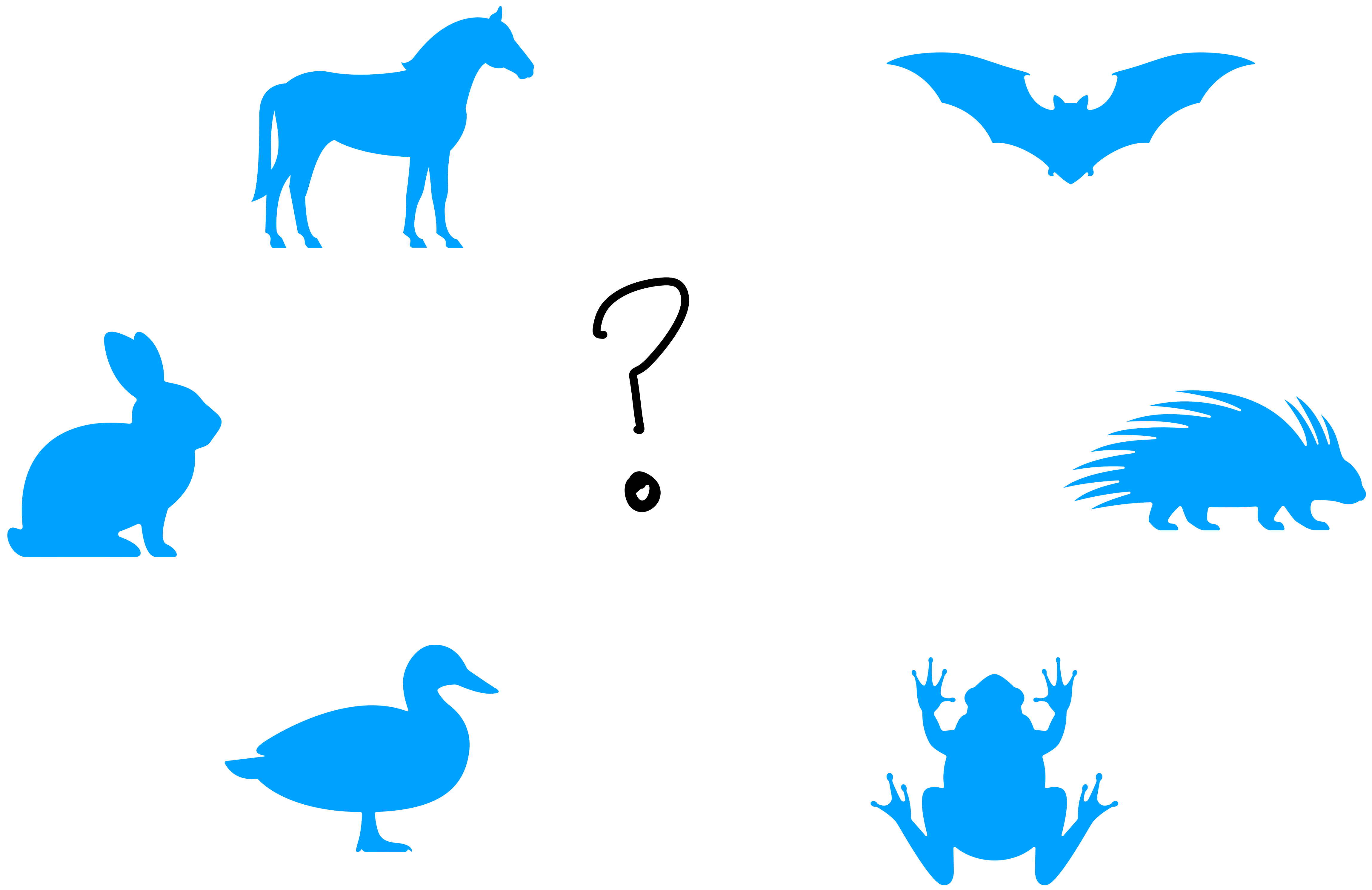
joint work with:

Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval,
AWS Wickr, ISTA, ISTA

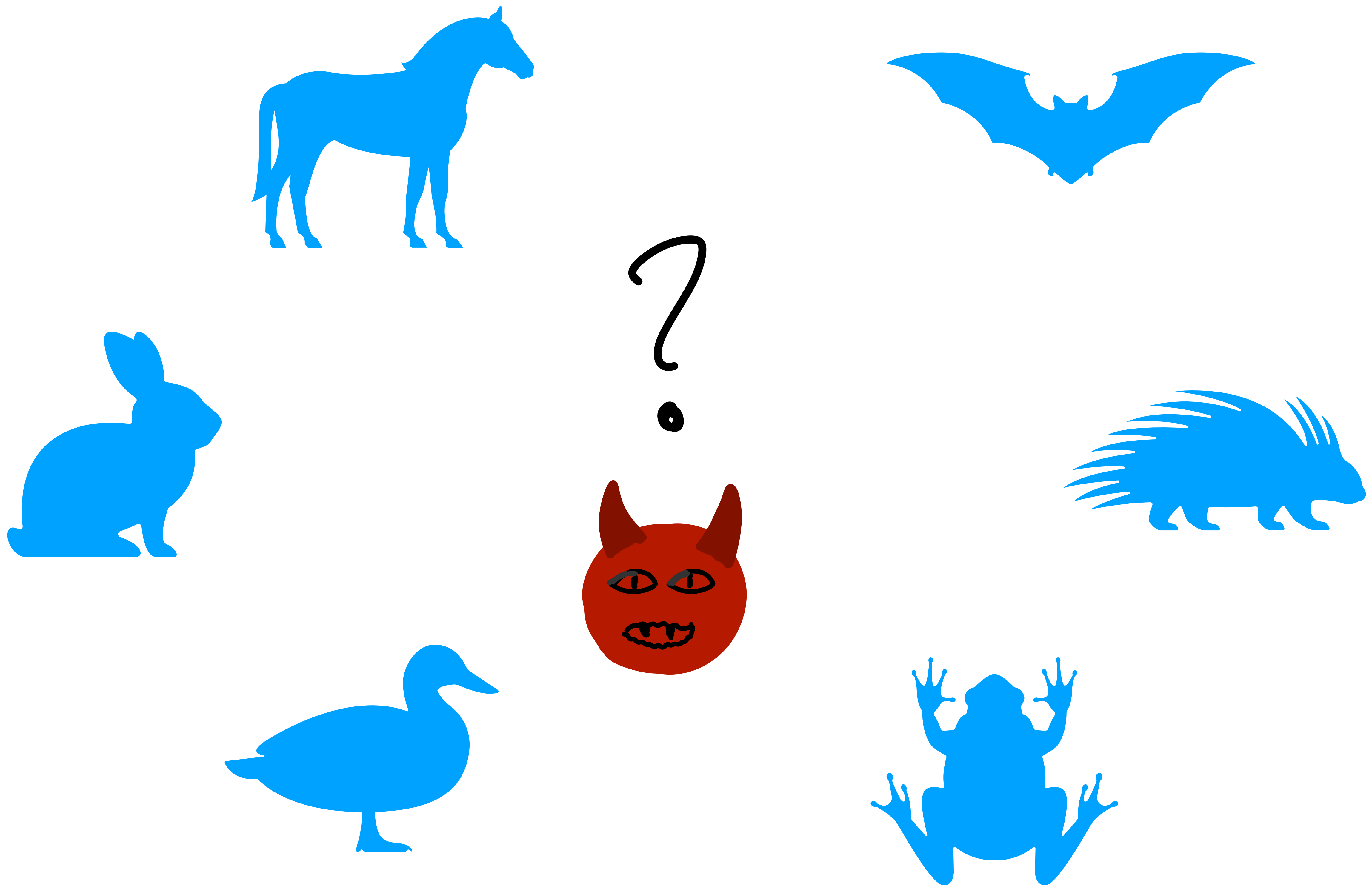
Karen Klein, Krzysztof Pietrzak and Michael Walter
ETH, ISTA, Zama

European Research Council
Established by the European Commission

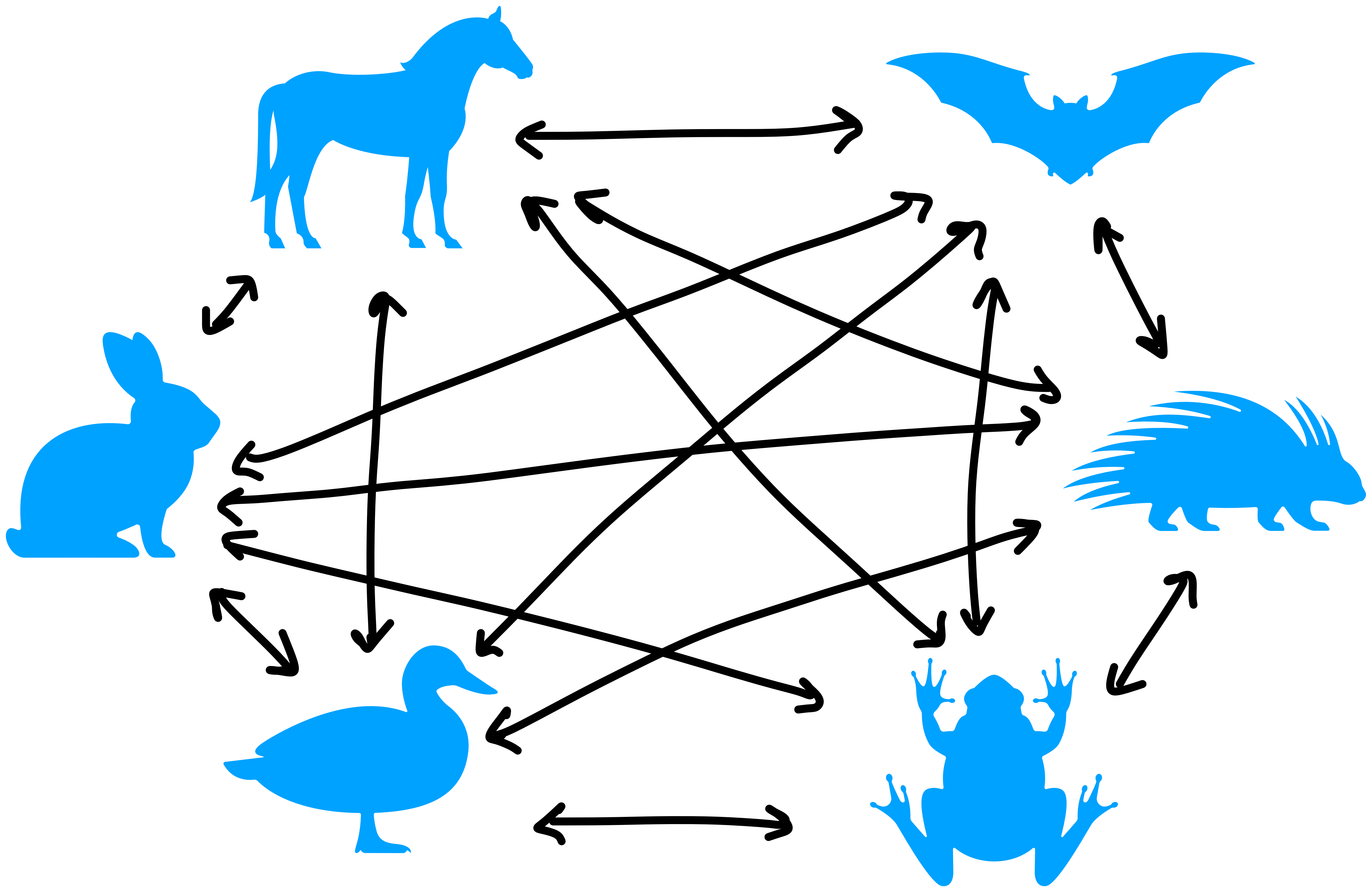
Motivation: Secure Group Messaging



Motivation: Secure Group Messaging

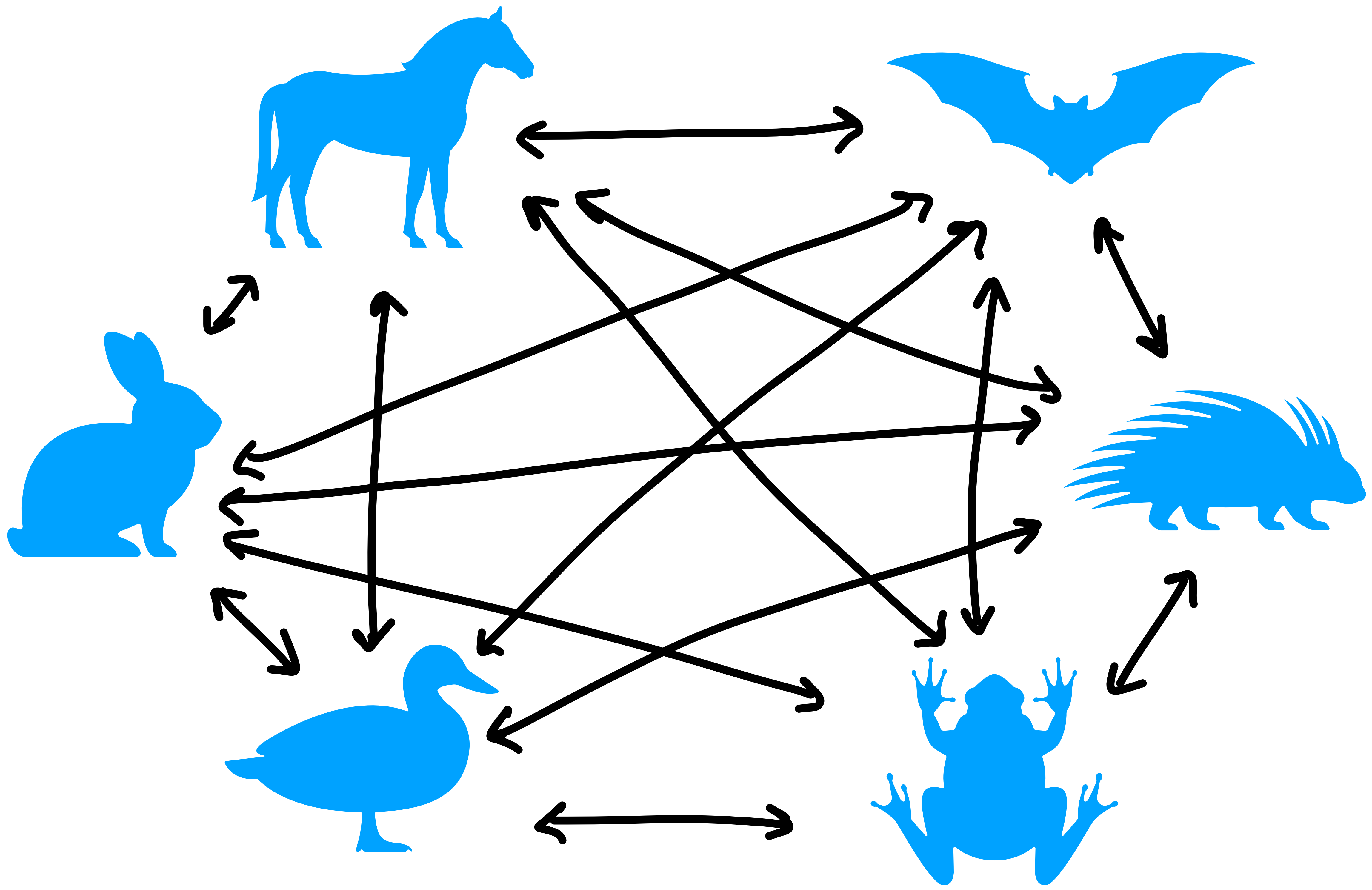


Motivation: Secure Group Messaging



Encrypted
Bidirectional
Channels

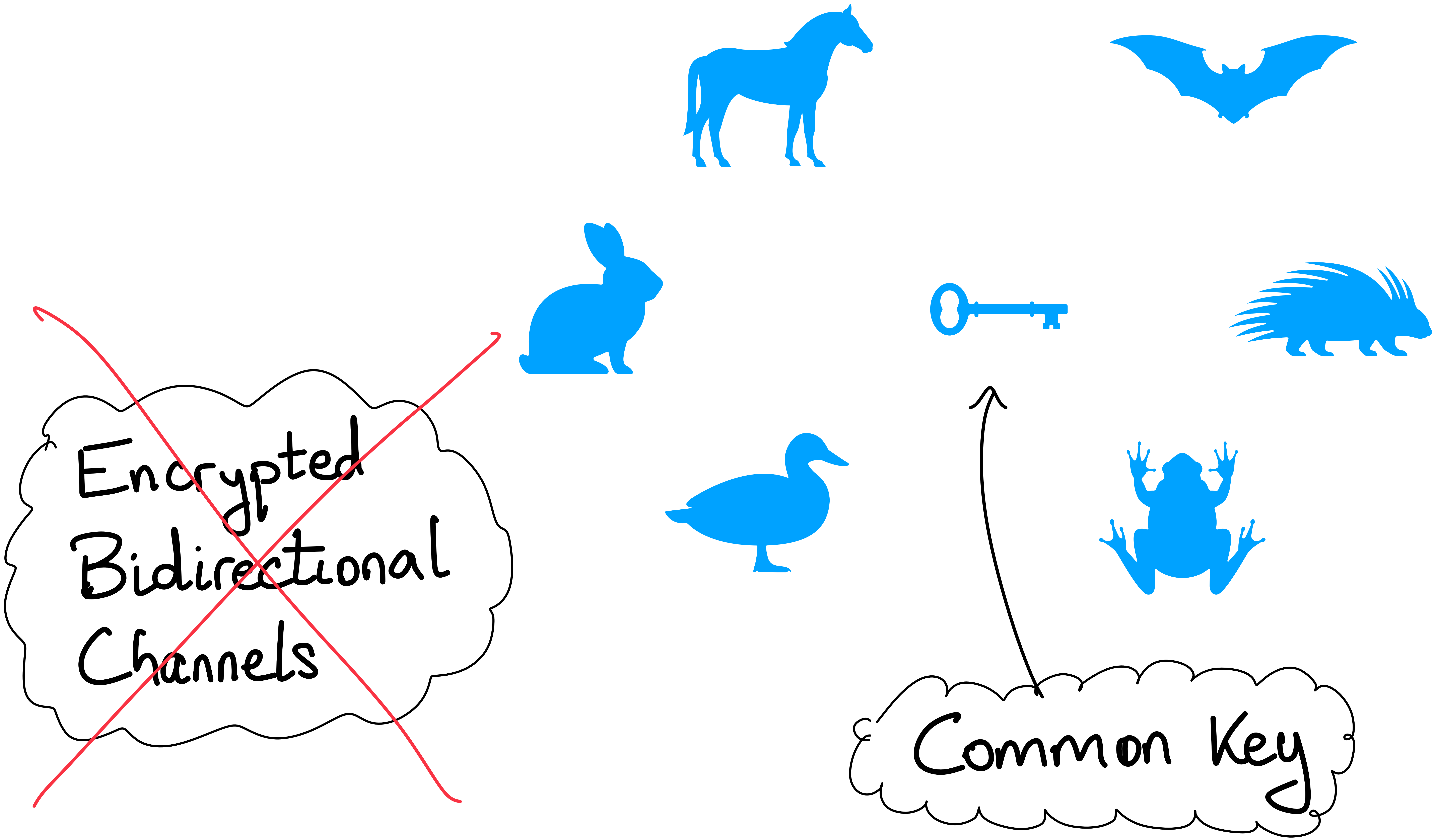
Motivation: Secure Group Messaging



Encrypted Bidirectional Channels

Linear cost communication → Not scalable!

Motivation: Secure Group Messaging



Continuous Group Key Agreement (CGKA) [Alwen et al. CRYPTO'20]

Allows n users to agree on common key

Continuous Group Key Agreement (CGKA) [Alwen et al. CRYPTO'20]

Allows n users to agree on common key

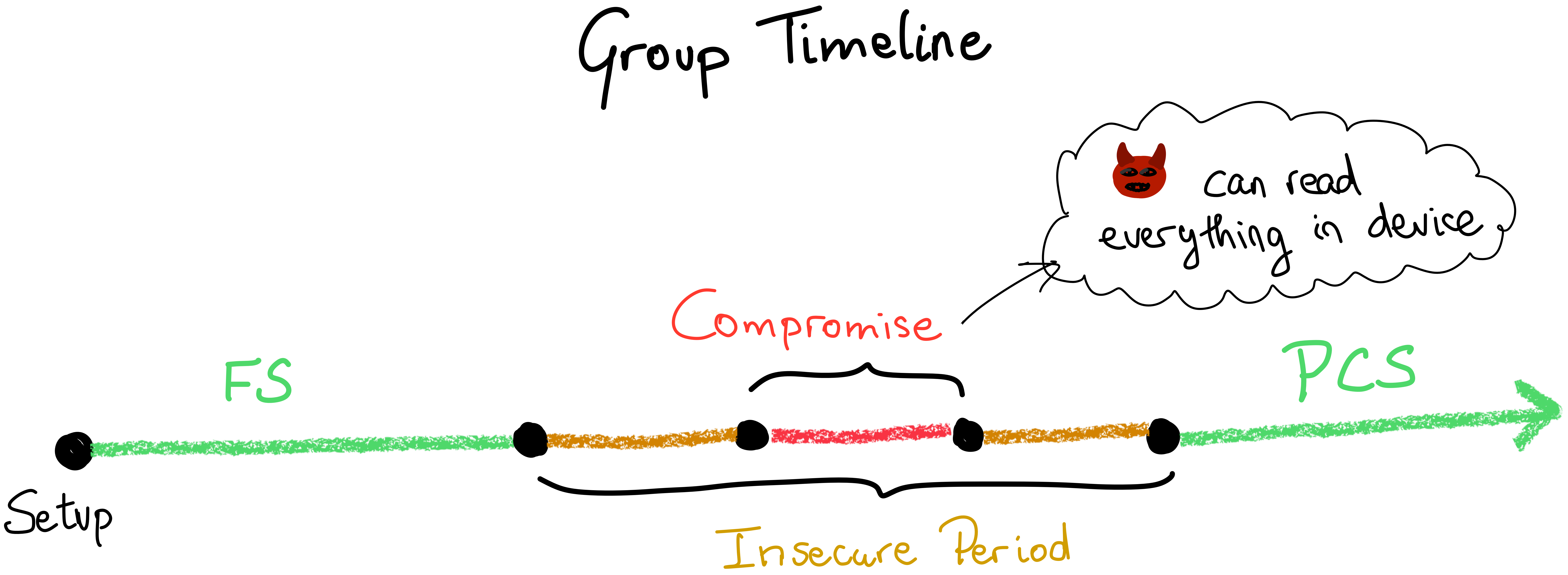
→ Dynamic Membership: Adds & Removes

→ Asynchronous: Untrusted server buffers & relays msgs.

→ Secure against compromise:

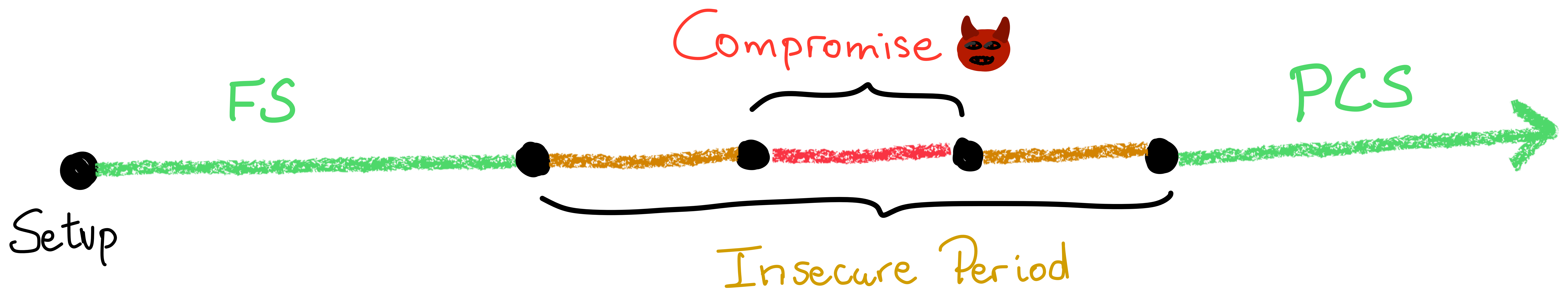
⇝ PCFS
- Forward Secrecy (FS)
- Post-Compromise Security (PCS)

Forward Secrecy and Post Compromise Security



Forward Secrecy and Post Compromise Security

Group Timeline



→ Efficient Key rotation: Log(n) communication p. rotation

Our Contribution

- → CoCoA, first CGKA allowing for concurrency without degrading efficiency
- → Overcome impossibility results on communication by relaxing PCS
- → Introduce notion of partial states

Outline

1. Introduction: TreeKEM
   - Concurrency in TreeKEM
2. CoCoA
   2.1 Concurrent updates in CoCoA
   2.2 Efficiency
   2.3 Partial States
   2.4 Security
3. Summary & Open Problems

Outline

1. Introduction: TreeKEM
   - Concurrency in TreeKEM
2. CoCoA
   - 2.1 Concurrent updates in CoCoA
   - 2.2 Efficiency
   - 2.3 Partial States
   - 2.4 Security
3. Summary & Open Problems

TreeKEM

→ CGKA underlying MLS

⇝ IETF standardization effort

→ Aim: scale to $n \leq 50$K

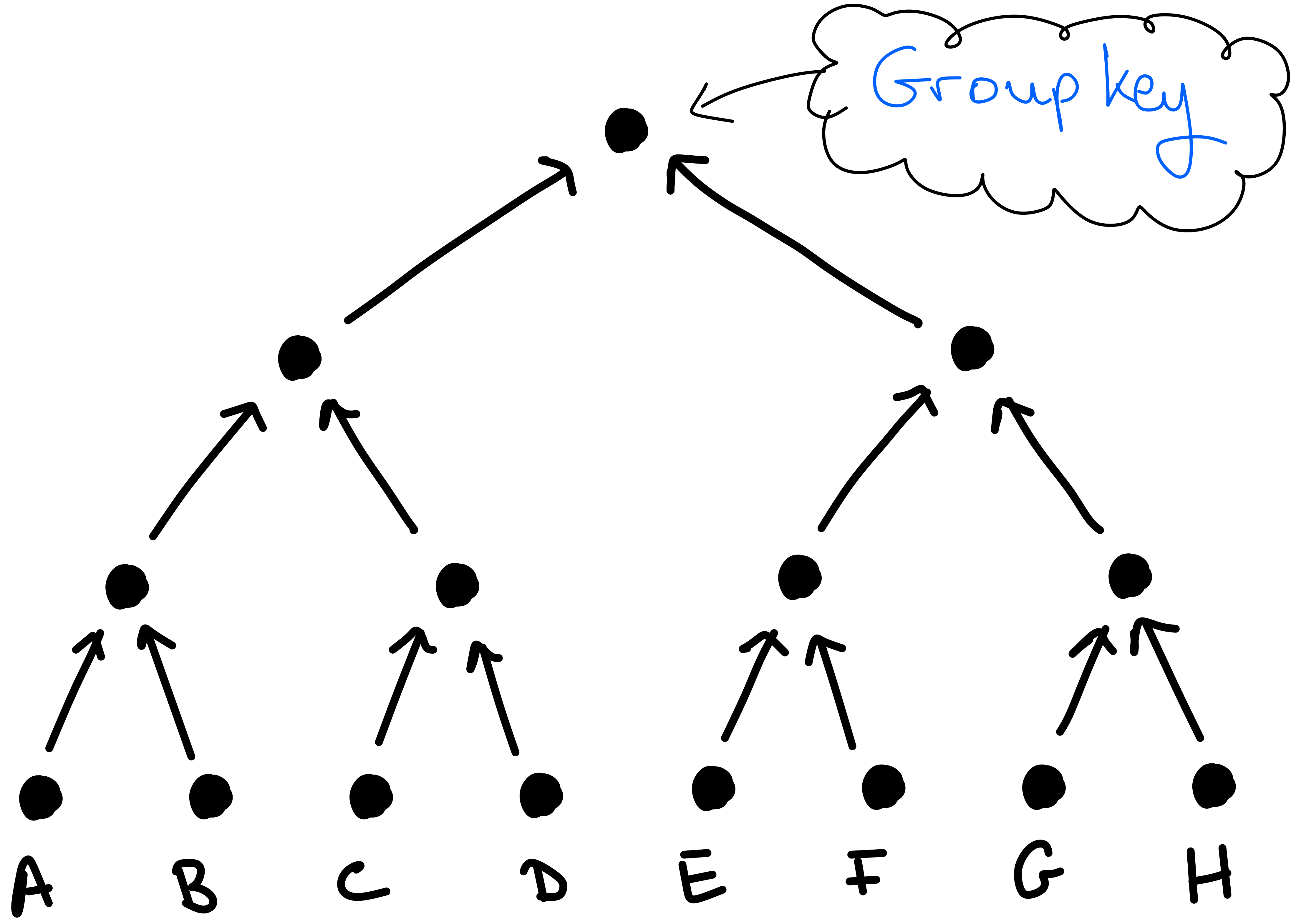
→ Proposed in 2018, replaced ART: Asynchronous Ratchetting Trees [Cohn-Gordon et al. CCS '18]

TreeKEM

→ CGKA underlying MLS
  ⇝ IETF standardization effort

→ Aim: scale to $n \leq 50K$

→ Proposed in 2018, replaced ART: Asynchronous Ratchetting Trees [Cohn-Gordon et al. CCS '18]



- PKE Key-pair associated to each node
- Users associated to leaves
- Arrows denote encryption or hash eval.

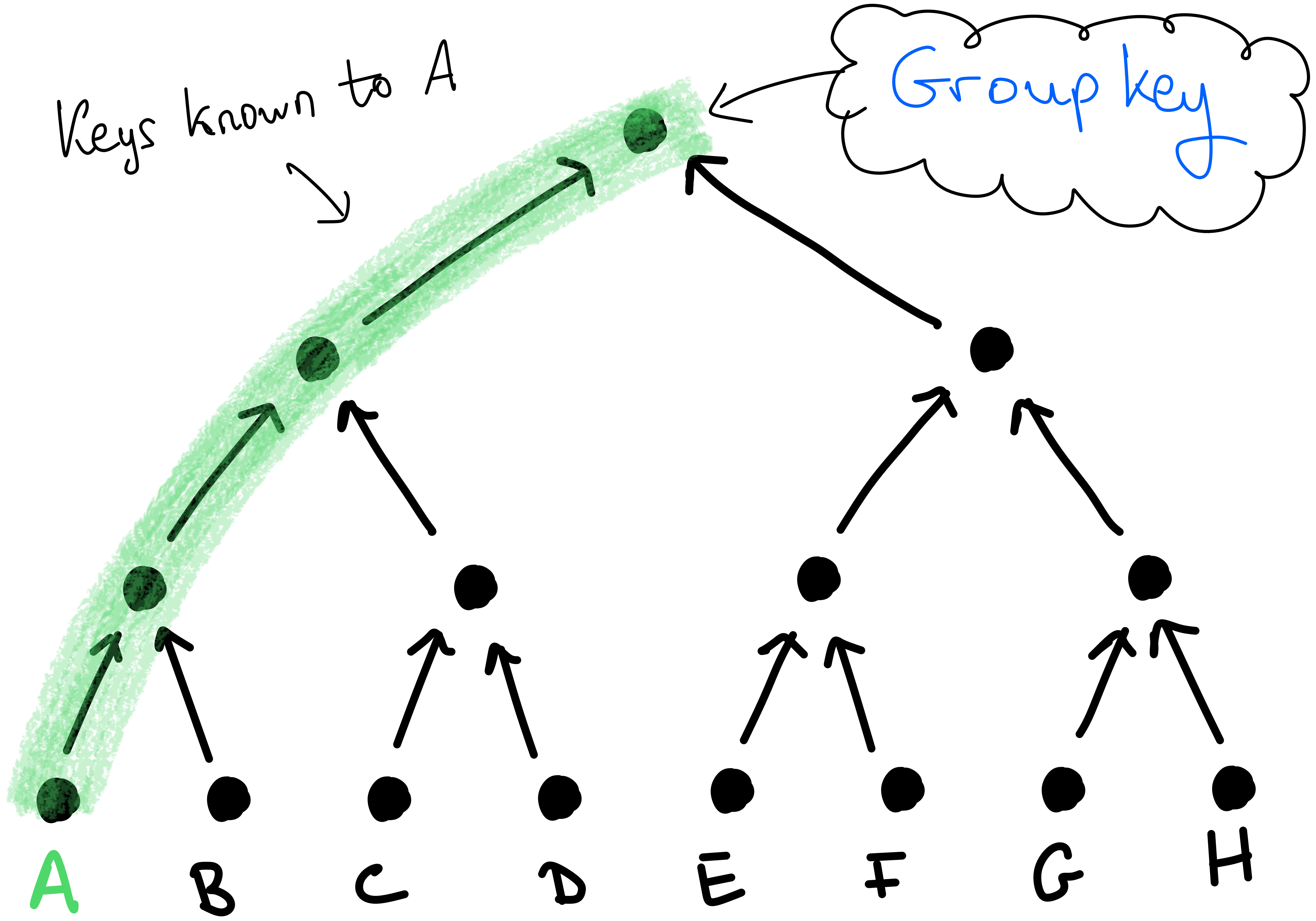
TreeKEM

→ CGKA underlying MLS

⇝ IETF standardization effort

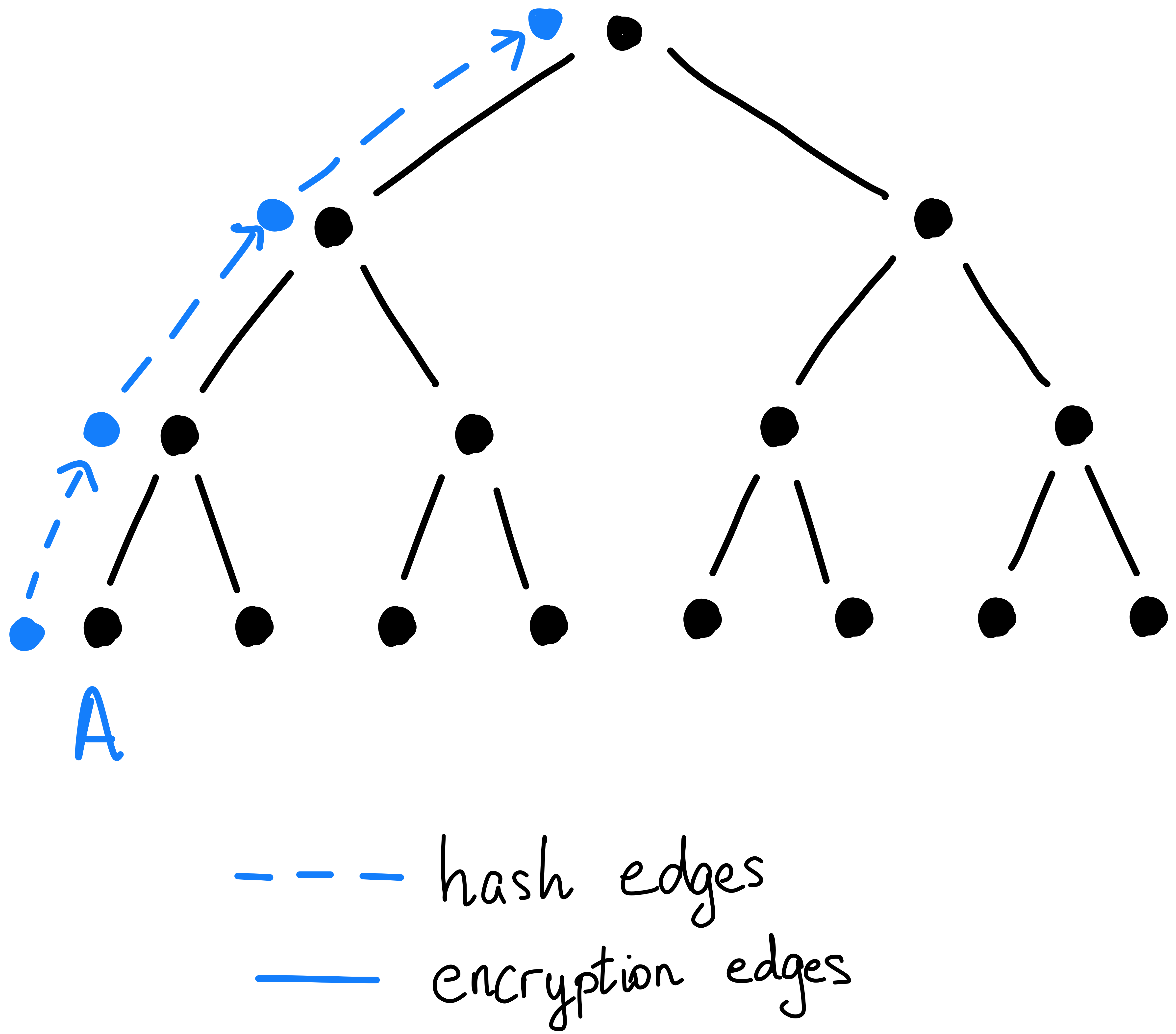
→ Aim: scale to $n \leq 50K$

→ Proposed in 2018, replaced ART: Asynchronous Ratchetting Trees [Cohn-Gordon et al. CCS '18]



- PKE Key-pair associated to each node
- Users associated to leaves
- Arrows denote knowledge correlation
- Users know keys on their path to root

TreeKEM: Update



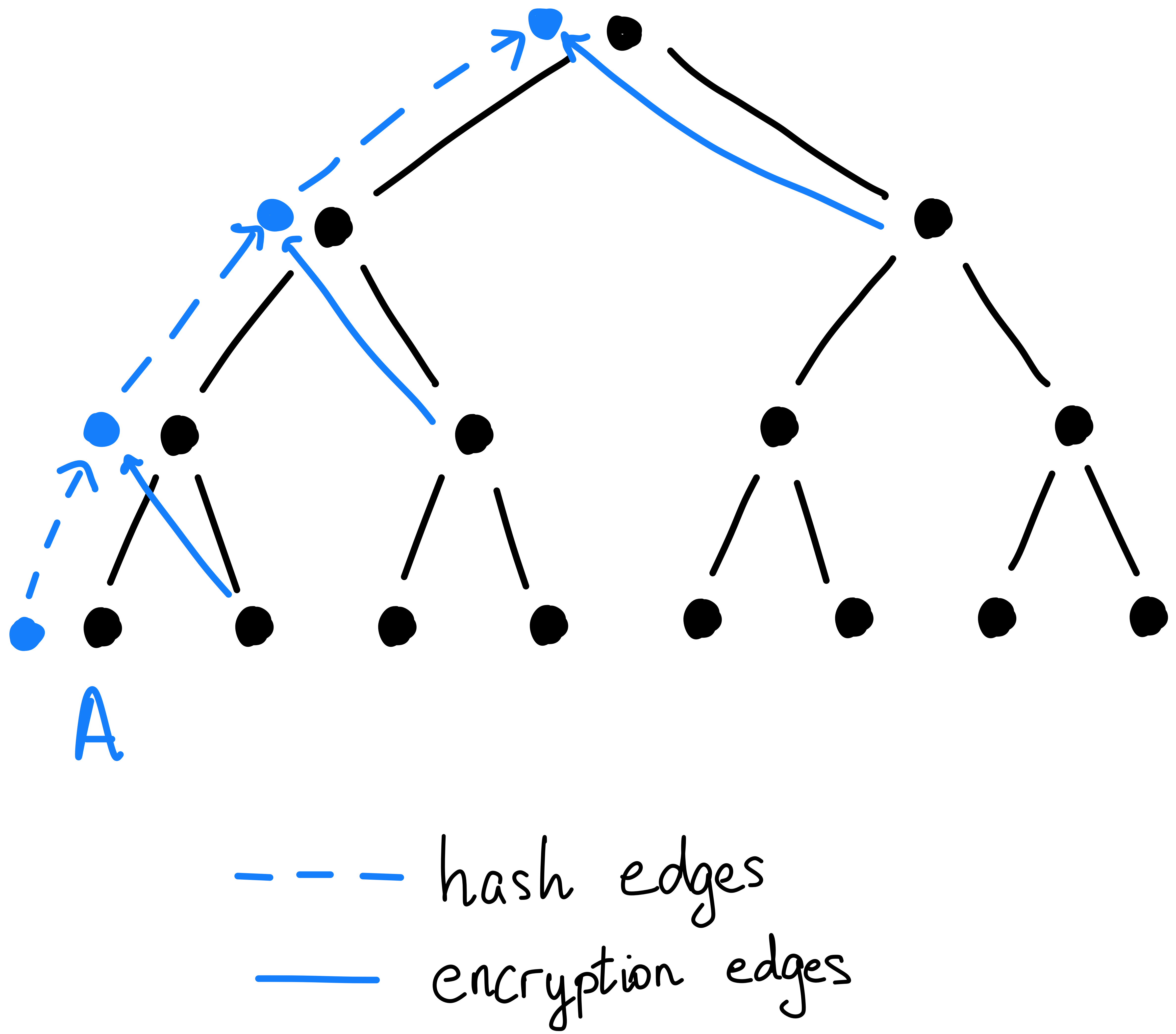
A updates:

i) samples seed $s_0 \overset{\$}{\leftarrow}$ for leaf

ii) derives seeds $s_i = H(s_{i-1})$

iii) derives keys $(sk_i, pk_i) \leftarrow \mathrm{PKE.Gen}(s_i)$

TreeKEM: Update



A updates:

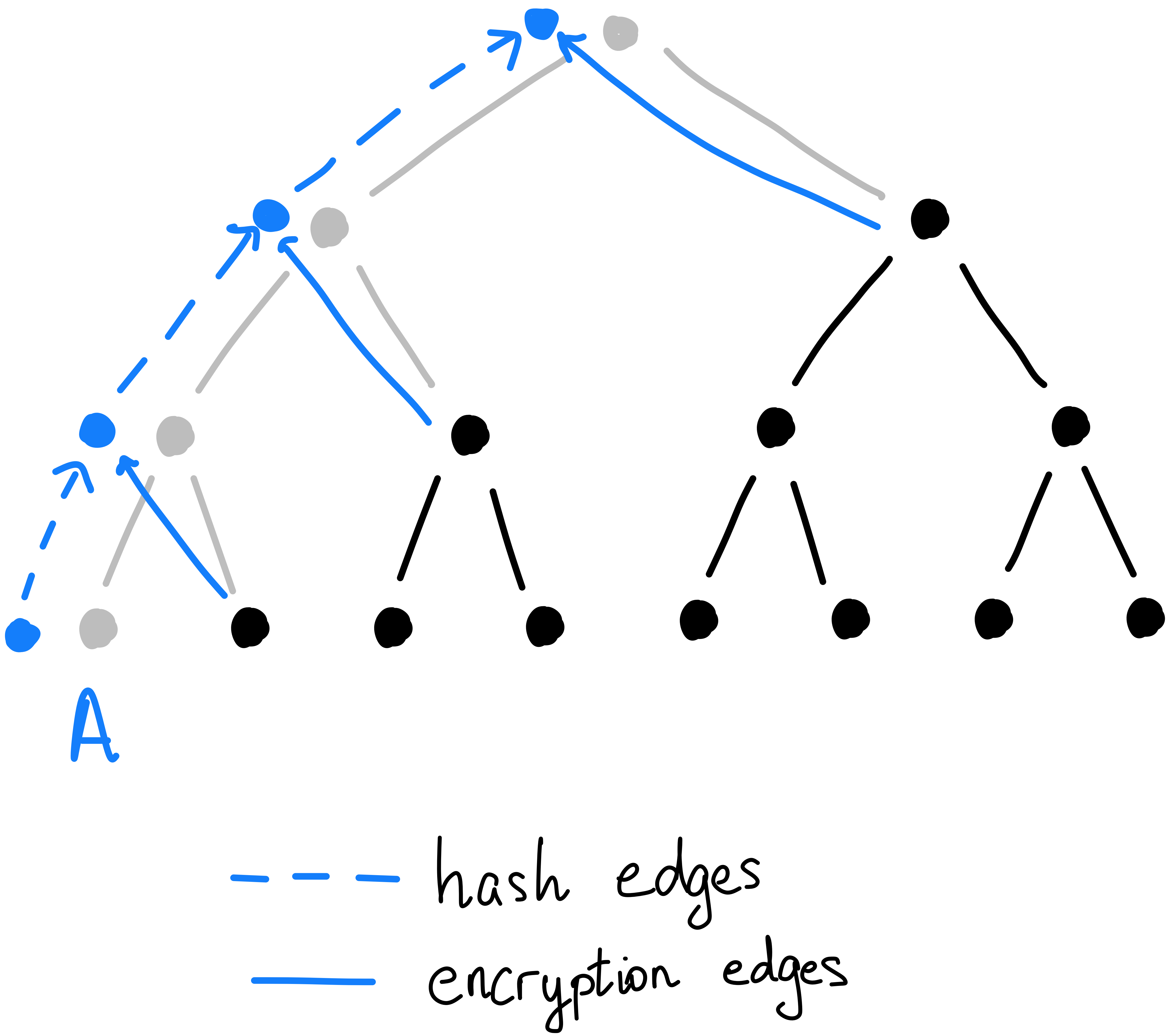
i) samples seed $s_0 \xleftarrow{\$}$ for leaf

ii) derives seeds $s_i = H(s_{i-1})$

iii) derives keys $(sk_i, pk_i) \leftarrow PKE.Gen(s_i)$

iv) Encrypts seeds to nodes in copath

TreeKEM: Update



A updates:

i) samples seed $s_0 \xleftarrow{\$}$ for leaf

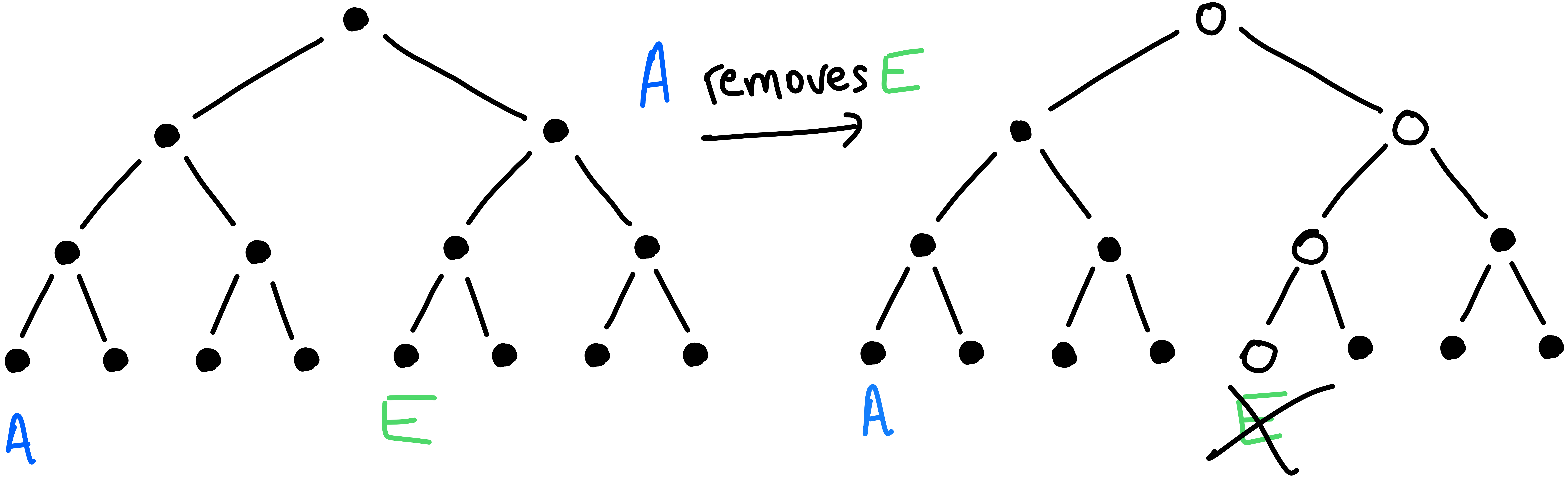
ii) derives seeds $s_i = H(s_{i-1})$

iii) derives keys $(sk_i, pk_i) \leftarrow PKE.Gen(s_i)$

iv) Encrypts seeds to nodes in copath

v) Old keys substituted by new

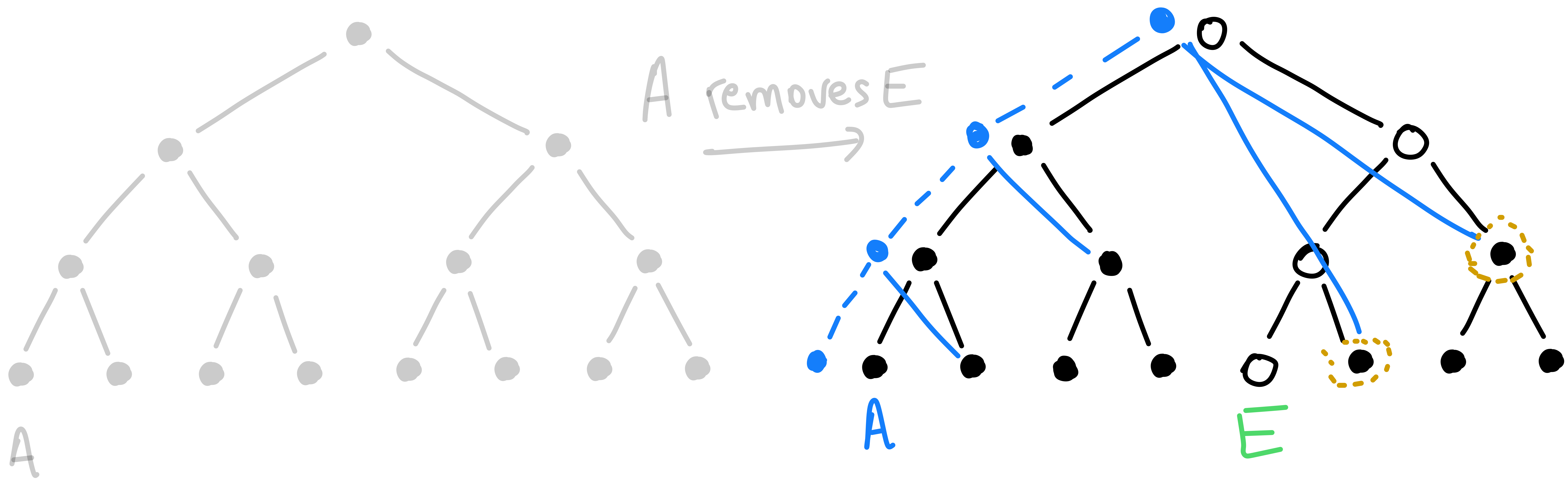
TreeKEM: Remove



→ A removes E by "blanking" nodes on E's path.

↳ Blank nodes (o) have no associated key

TreeKEM: Remove



→ A removes E by "blanking" nodes on E's path.

↳ Blank nodes (o) have no associated key

↳ Encryption to blank → Encryption to resolution

Outline

1. Introduction: TreeKEM
   - Concurrency in TreeKEM
2. CoCoA
   2.1 Concurrent updates in CoCoA
   2.2 Efficiency
   2.3 Partial States
   2.4 Security
3. Summary & Open Problems

Concurrency in TreeKEM

→ Plain TreeKEM (version ≤ 7):

- No concurrency; updates applied in order, user can only update if received all previous updates
  ↳ t updates need t comm. rounds

Concurrency in TreeKEM

→ **Plain TreeKEM** (version ≤ 7):

- No concurrency; updates applied in order, user can only update if received all previous updates
  ↳ t updates need t comm. rounds

→ **Propose and Commit (P&C) TreeKEM** (version ≥ 8):

- Allows concurrent updates.
- Users send update proposals
- Can commit several proposals: all executed simult.

P&C and Concurrency

- → Can achieve PCS in 2 rounds
  - ↳ Round 1: Corrupted users propose update
  - ↳ Round 2: Someone commits all proposals
- → Linear comm. complexity in # updating users
  - ↳ Optimal for 2-round healing (Bienstock et al. TCC '20)

P&C and Concurrency

→ Can achieve PCS in 2 rounds

- ↳ Round 1: Corrupted users propose update
- ↳ Round 2: Someone commits all proposals

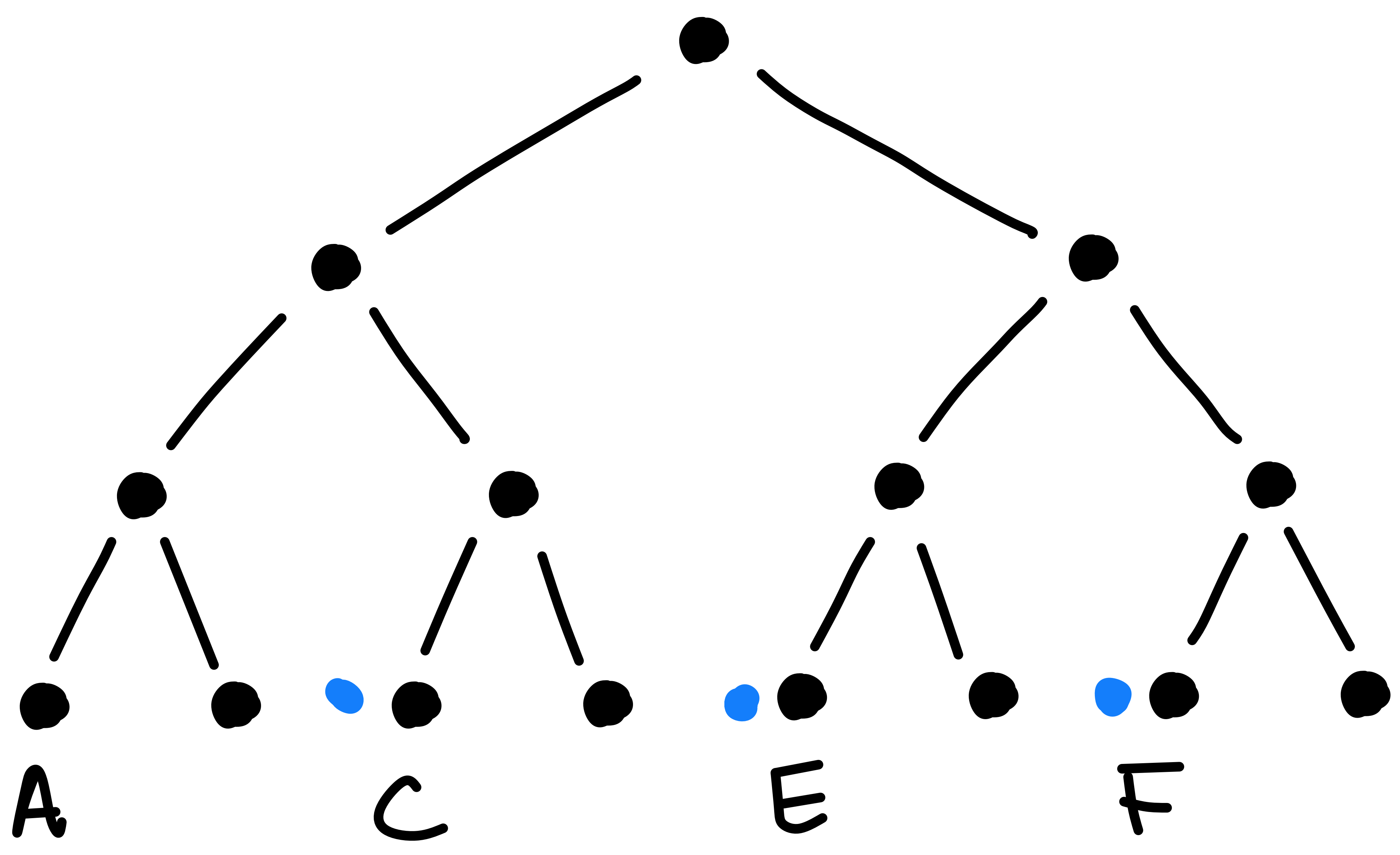
→ Linear comm. complexity in # updating users

- ↳ Optimal for 2-round healing (Bienstock et al. TCC '20)

→ Caveat: Update proposals ruin the tree structure

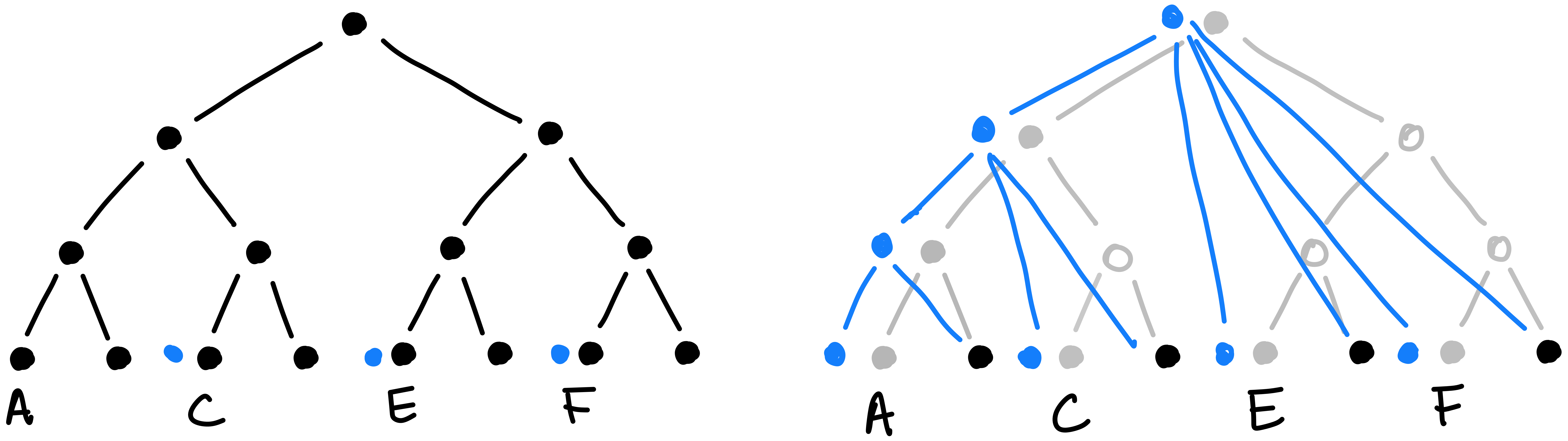
- ↳ Fast healing degrades performance

Example: Tree after many updates committed concurrently



(i) C, E, F propose updates

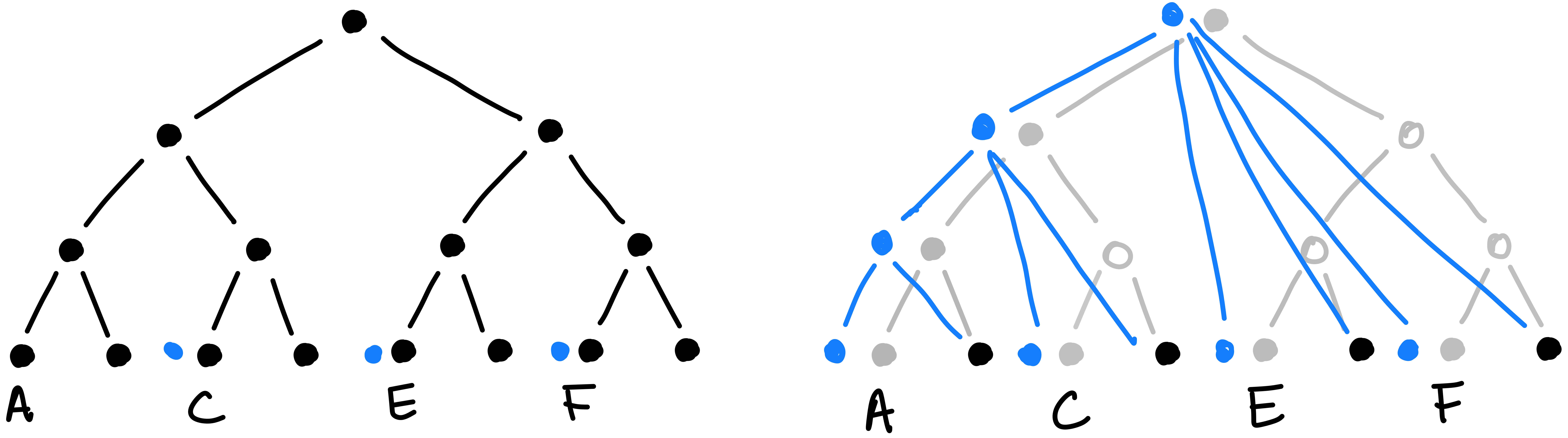
Example: Tree after many updates committed concurrently



(i) C, E, F propose updates

(ii) A commits the proposals

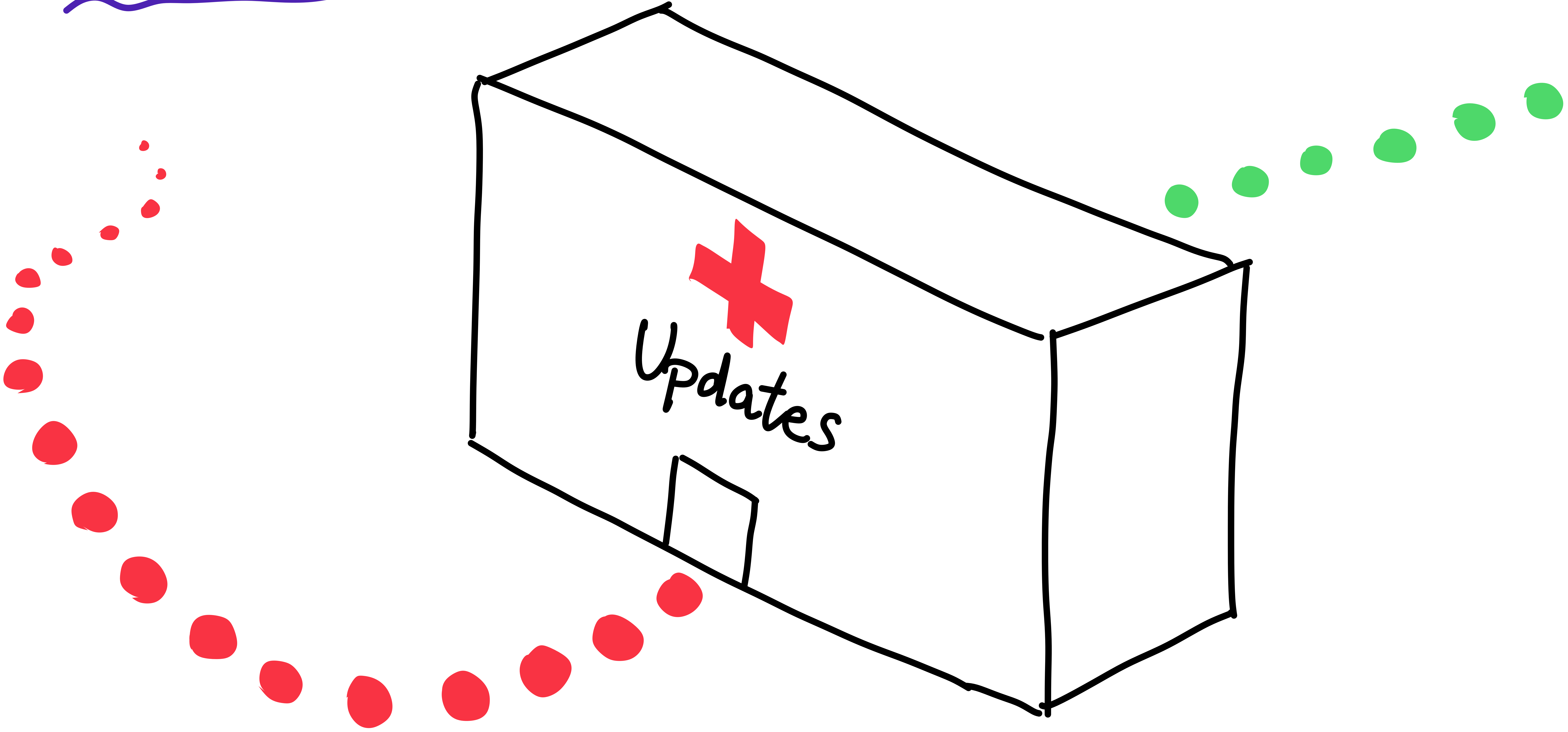
Example: Tree after many updates committed concurrently



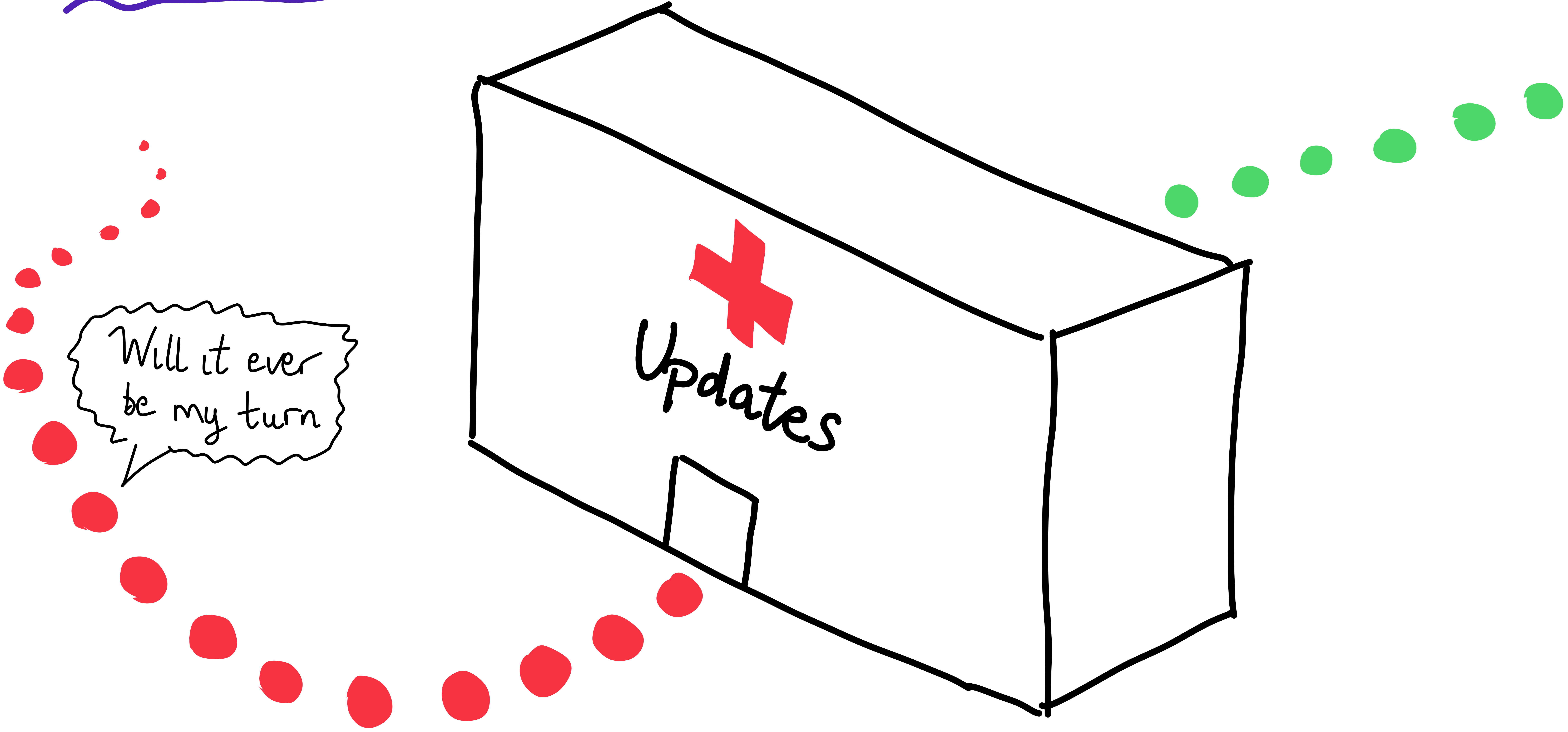
(i) C, E, F propose updates
(ii) A commits the proposals

→ Future updates as bad as trivial solution

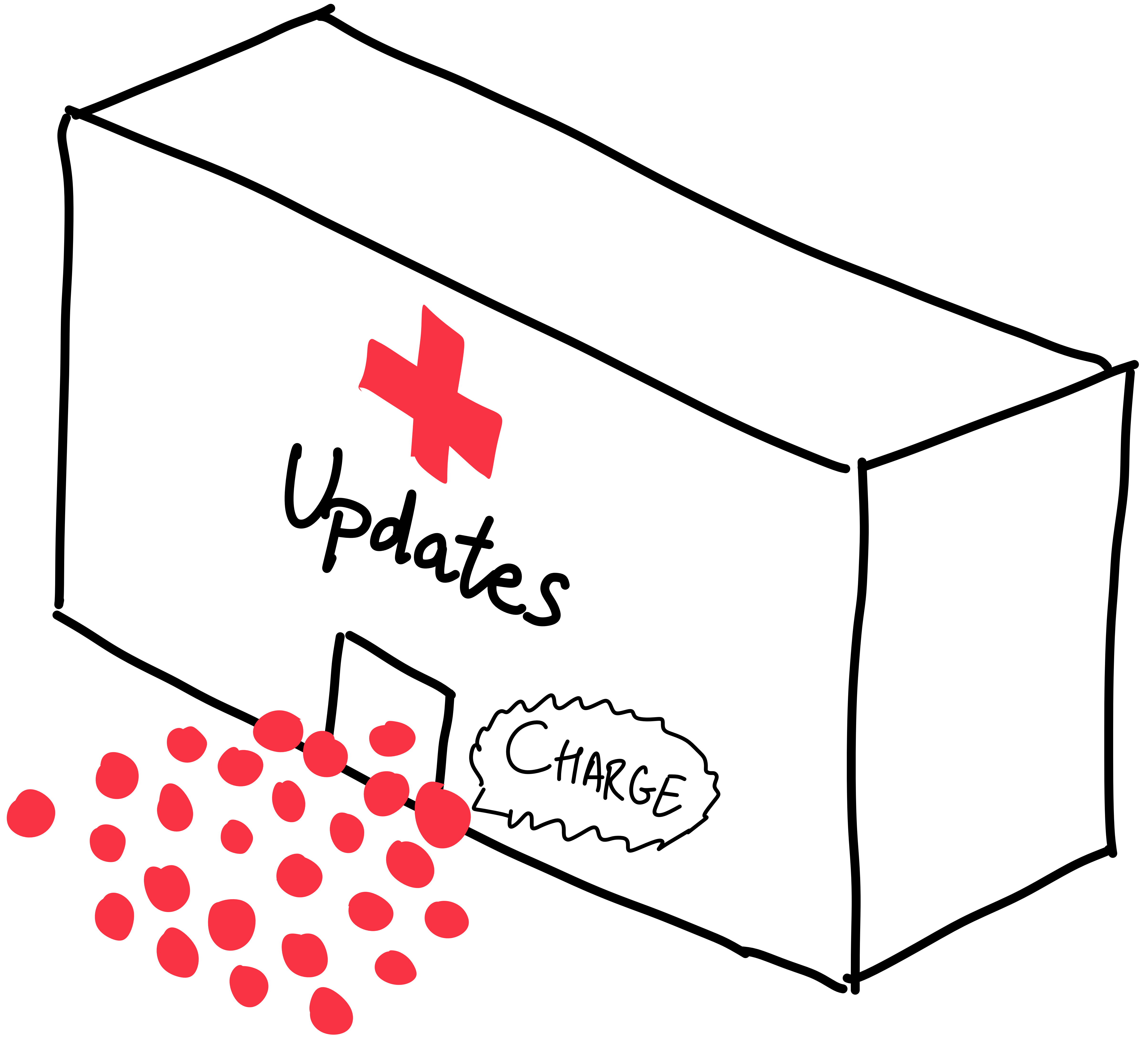
Plain TreeKEM



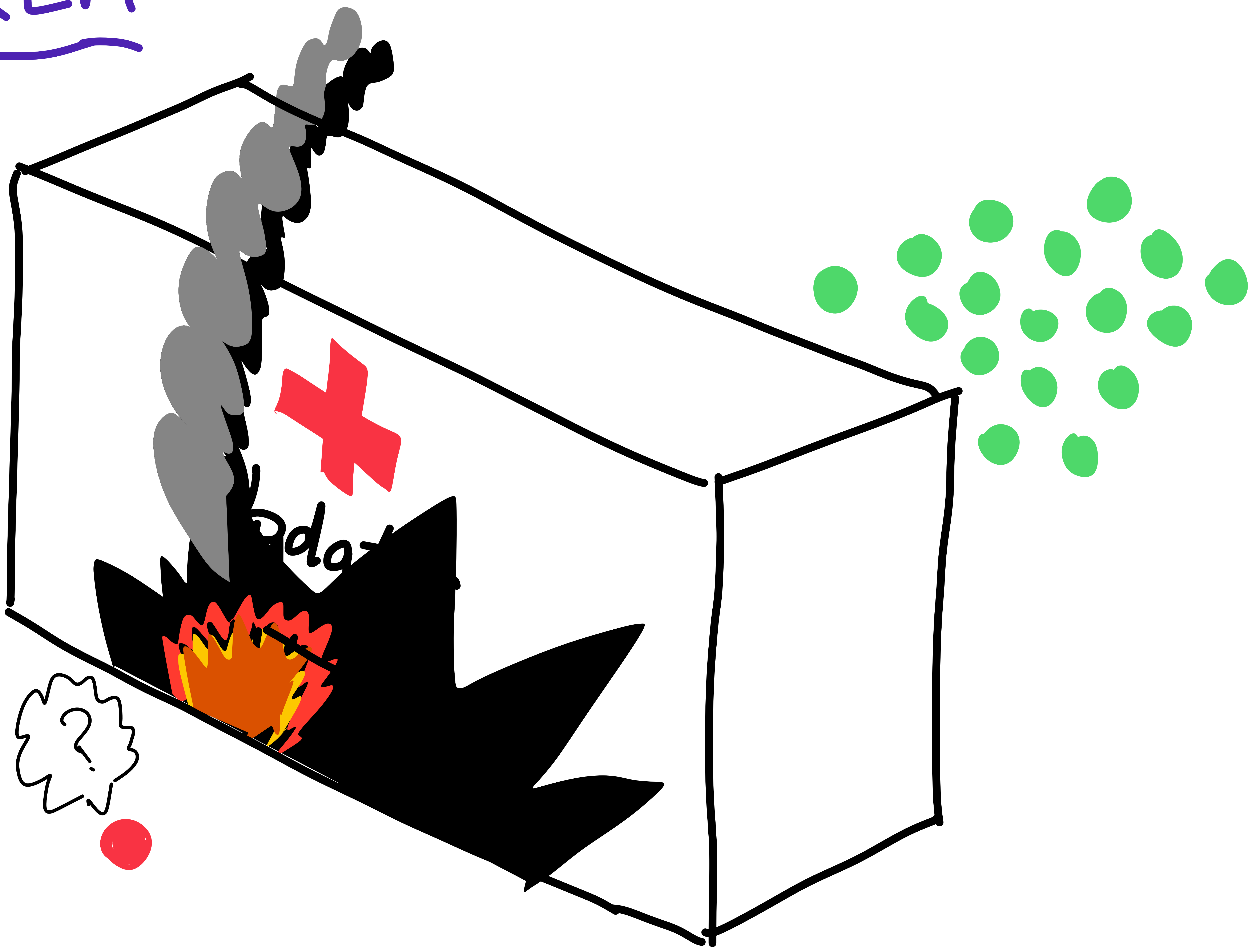
Plain TreeKEM



P&C TreeKEM



P&C TreeKEM

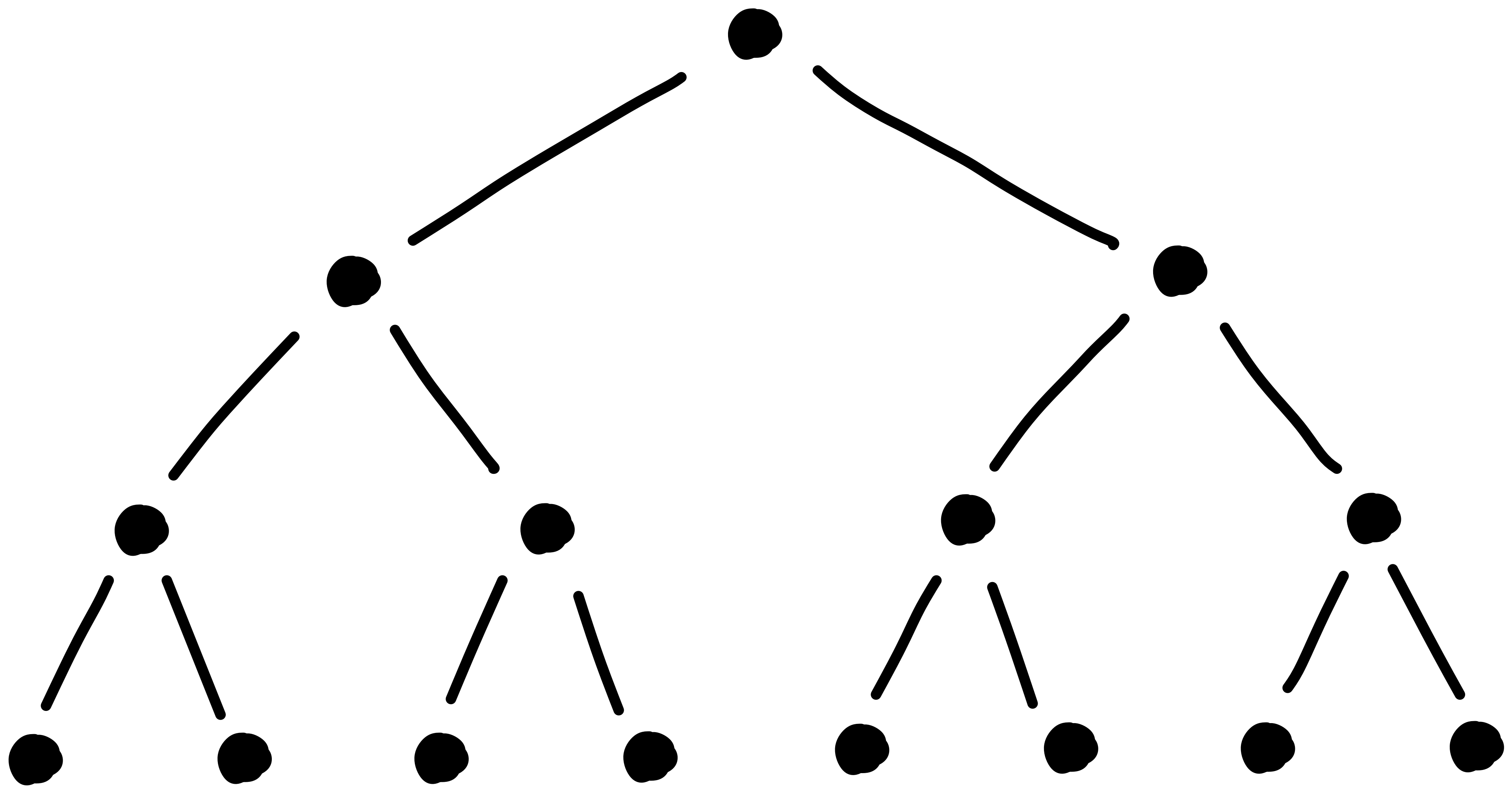


We can do better if we allow PCS to only require > 2 rounds !!

Outline

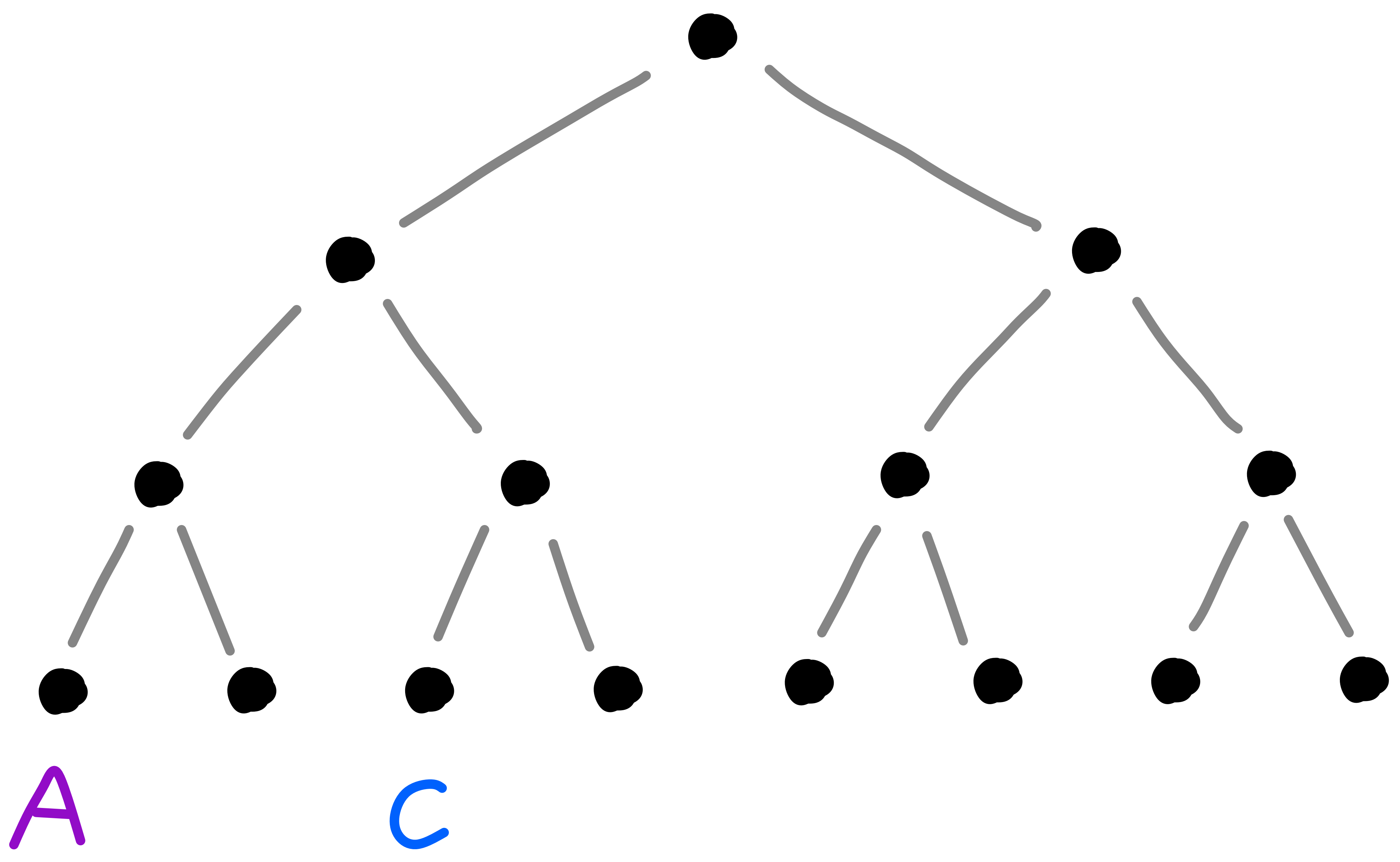
1. Introduction: TreeKEM
   - Concurrency in TreeKEM
2. CoCoA
   2.1 Concurrent updates in CoCoA
   2.2 Efficiency
   2.3 Partial States
   2.4 Security
3. Summary & Open Problems

Concurrent updates in CoCoA



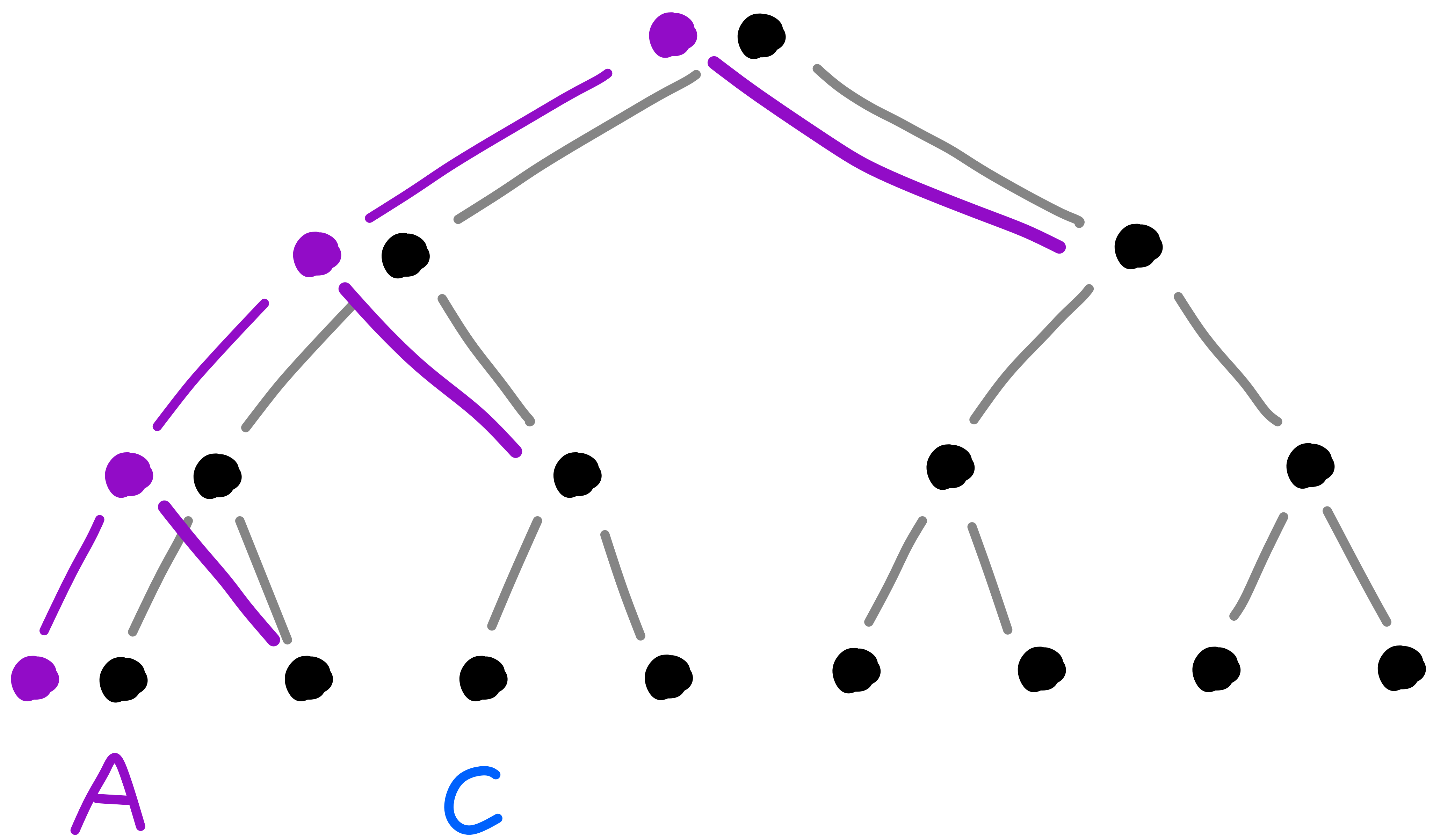
Concurrent updates in CoCoA

- Several parties update concurrently



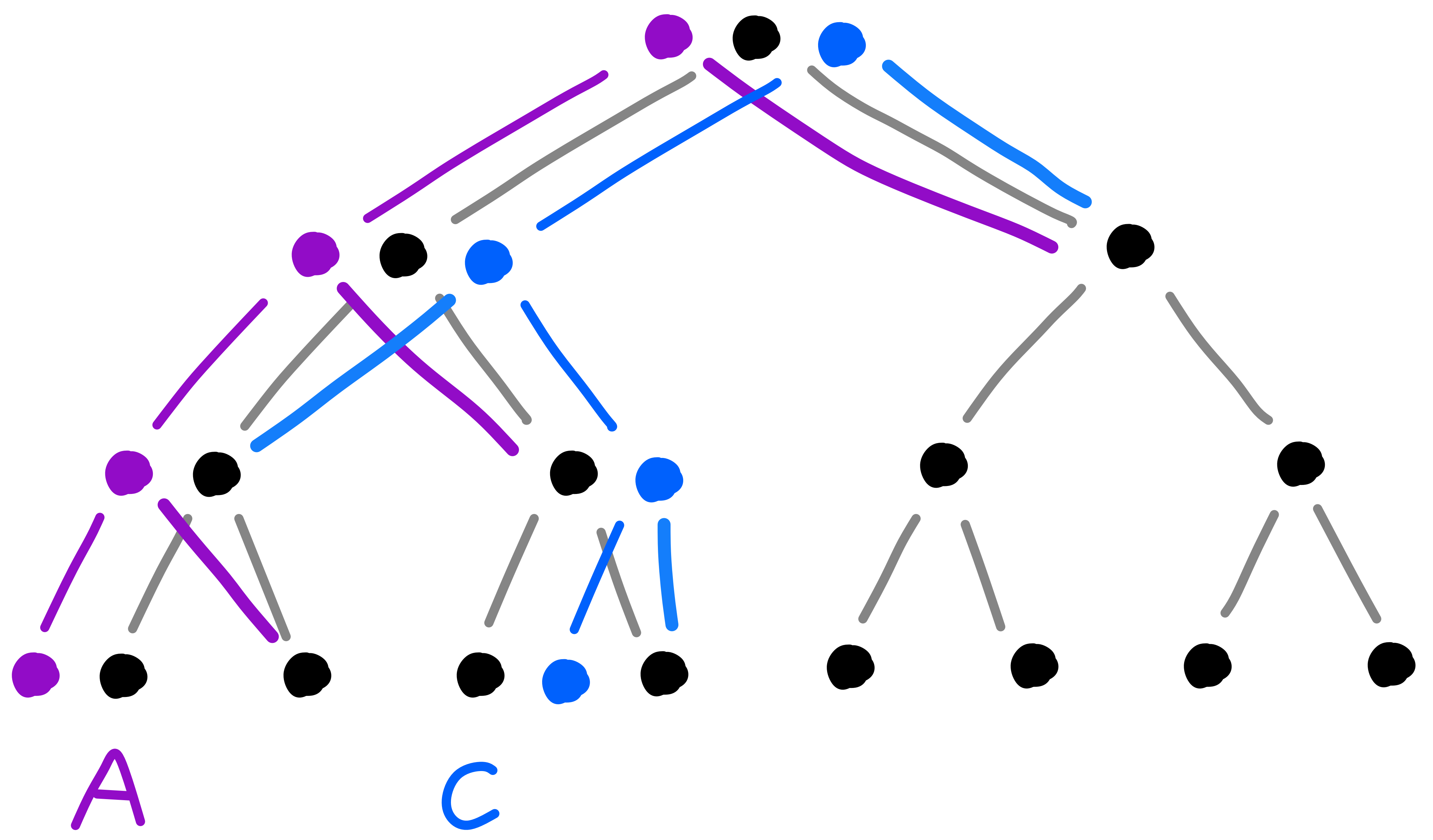
Concurrent updates in CoCoA

- Several parties update concurrently
  → Prepare updates as in TreeKEM

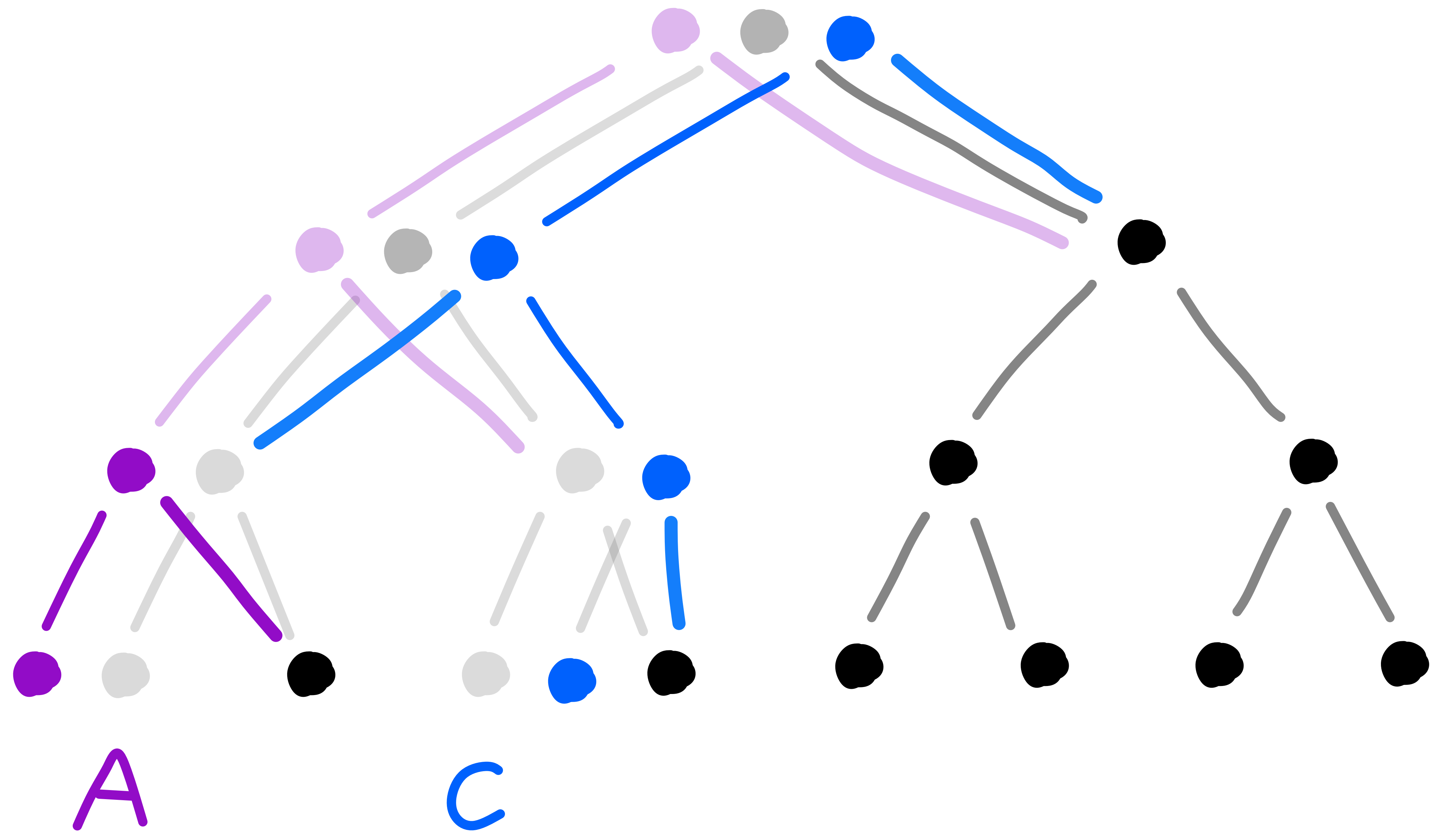


Concurrent updates in CoCoA

- Several parties update concurrently
  → Prepare updates as in TreeKEM

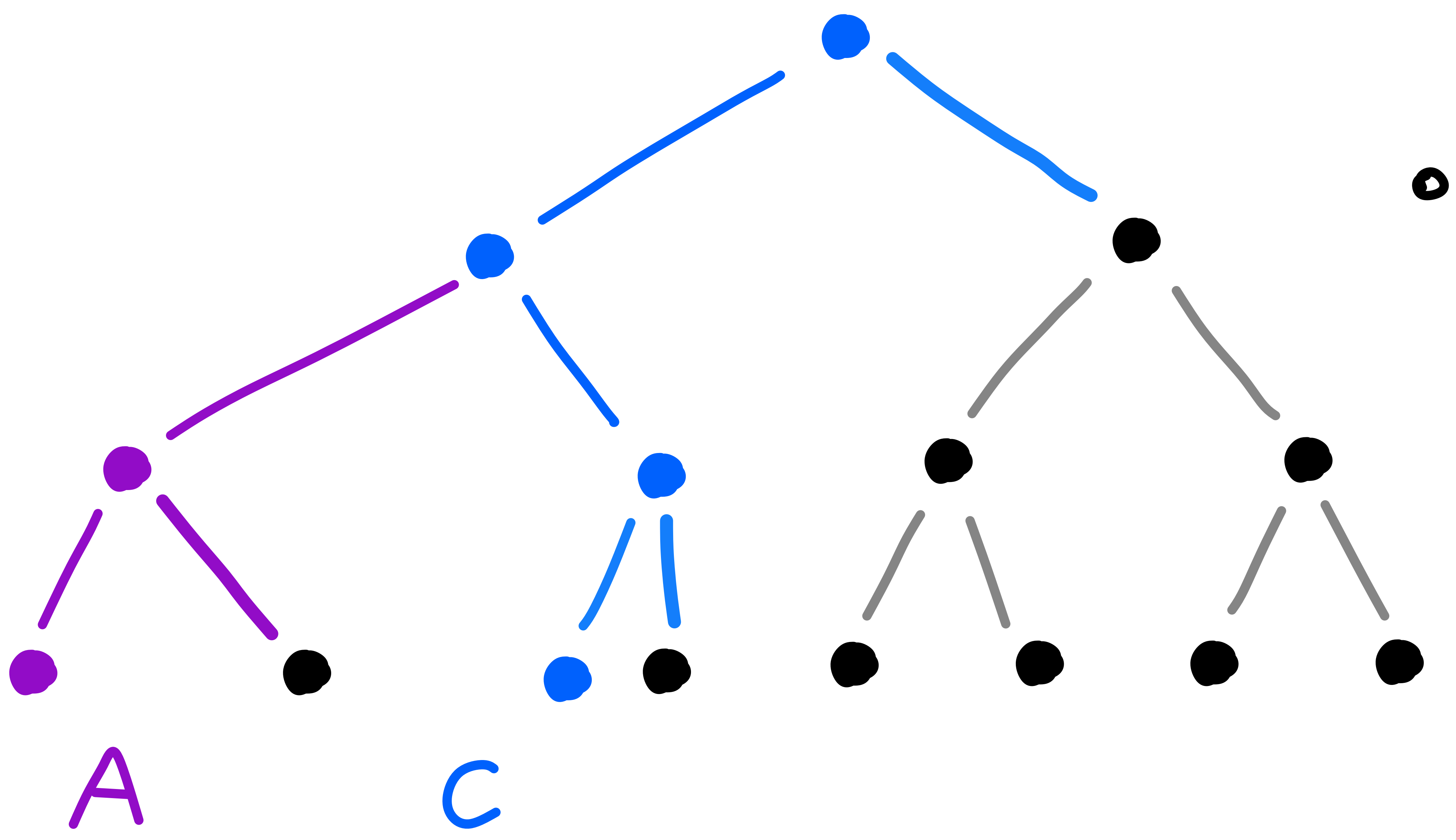


Concurrent updates in CoCoA



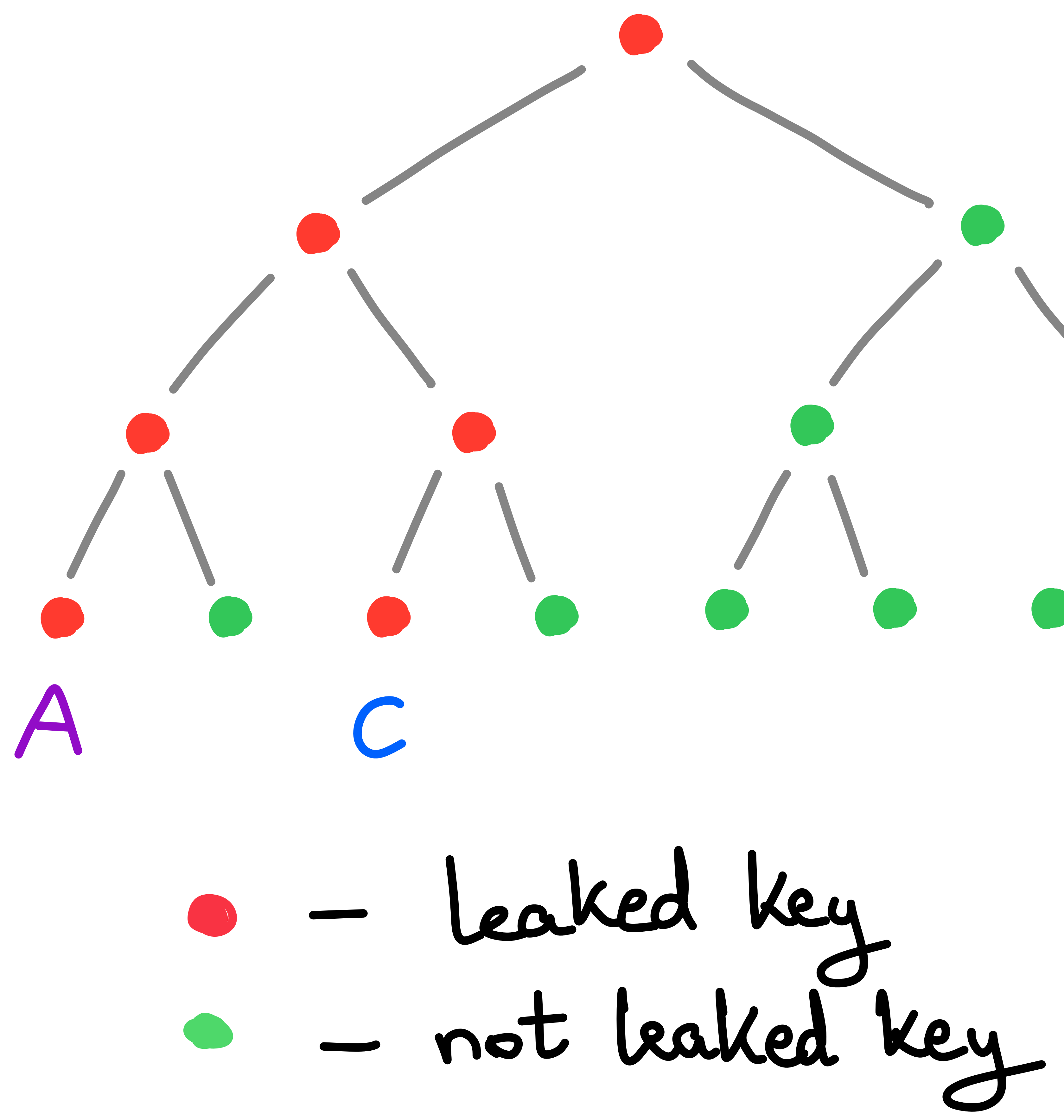
- Several parties update concurrently
  → Prepare updates as in TreeKEM
- Application of several updates
  (i) Node only affected by 1 update
    ↳ Apply as in TreeKEM
  (ii) Else:
    ↳ Determine "winner" by ordering
      e.g. "rightmost user wins"

Concurrent updates in CoCoA



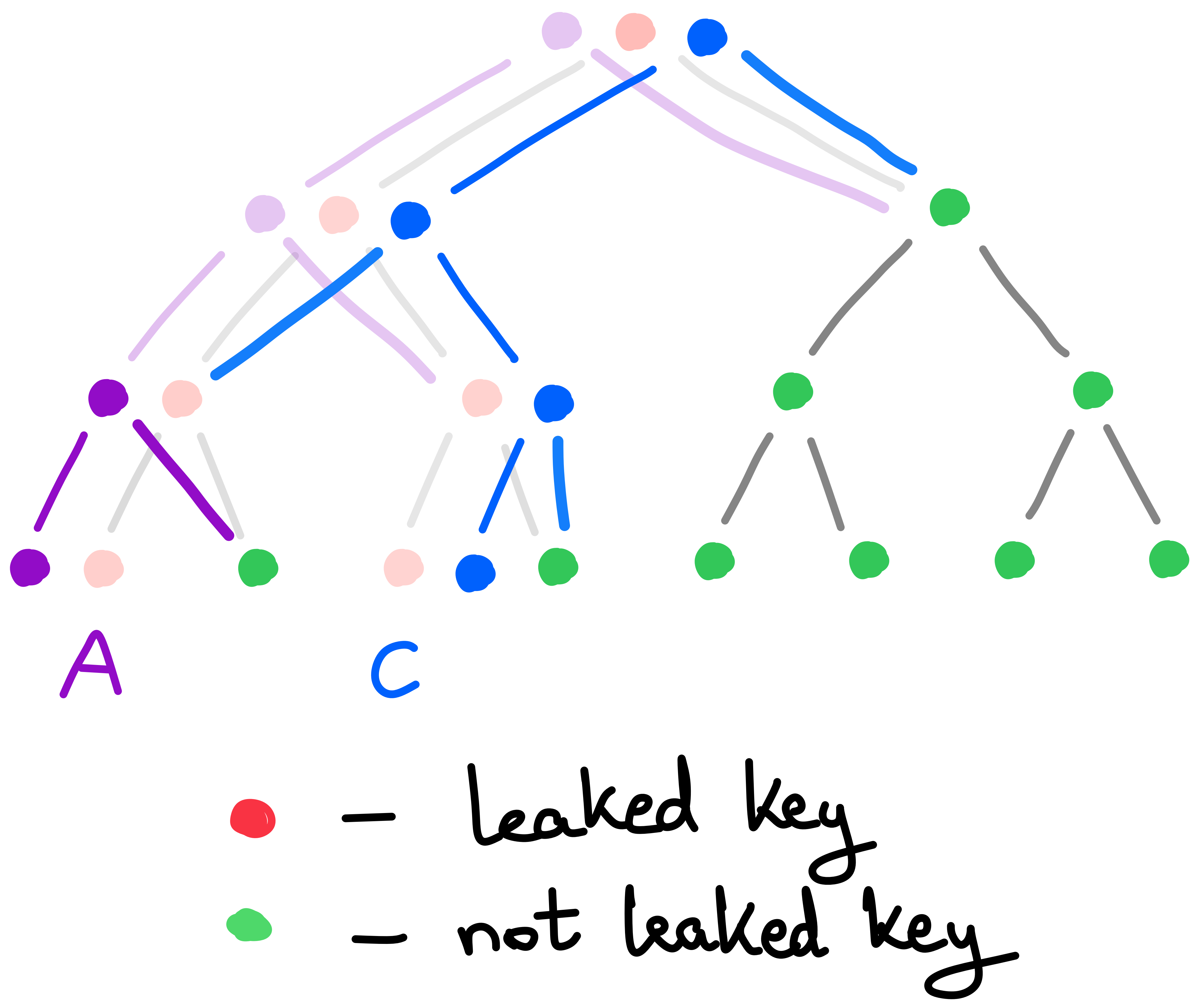
- Several parties update concurrently
  → Prepare updates as in TreeKEM
- Application of several updates
  (i) Node only affected by 1 update
  ↳ Apply as in TreeKEM
  (ii) Else:
  ↳ Determine "winner" by ordering
  e.g. "rightmost user wins"

Concurrent updates in CoCoA: PCS consequences



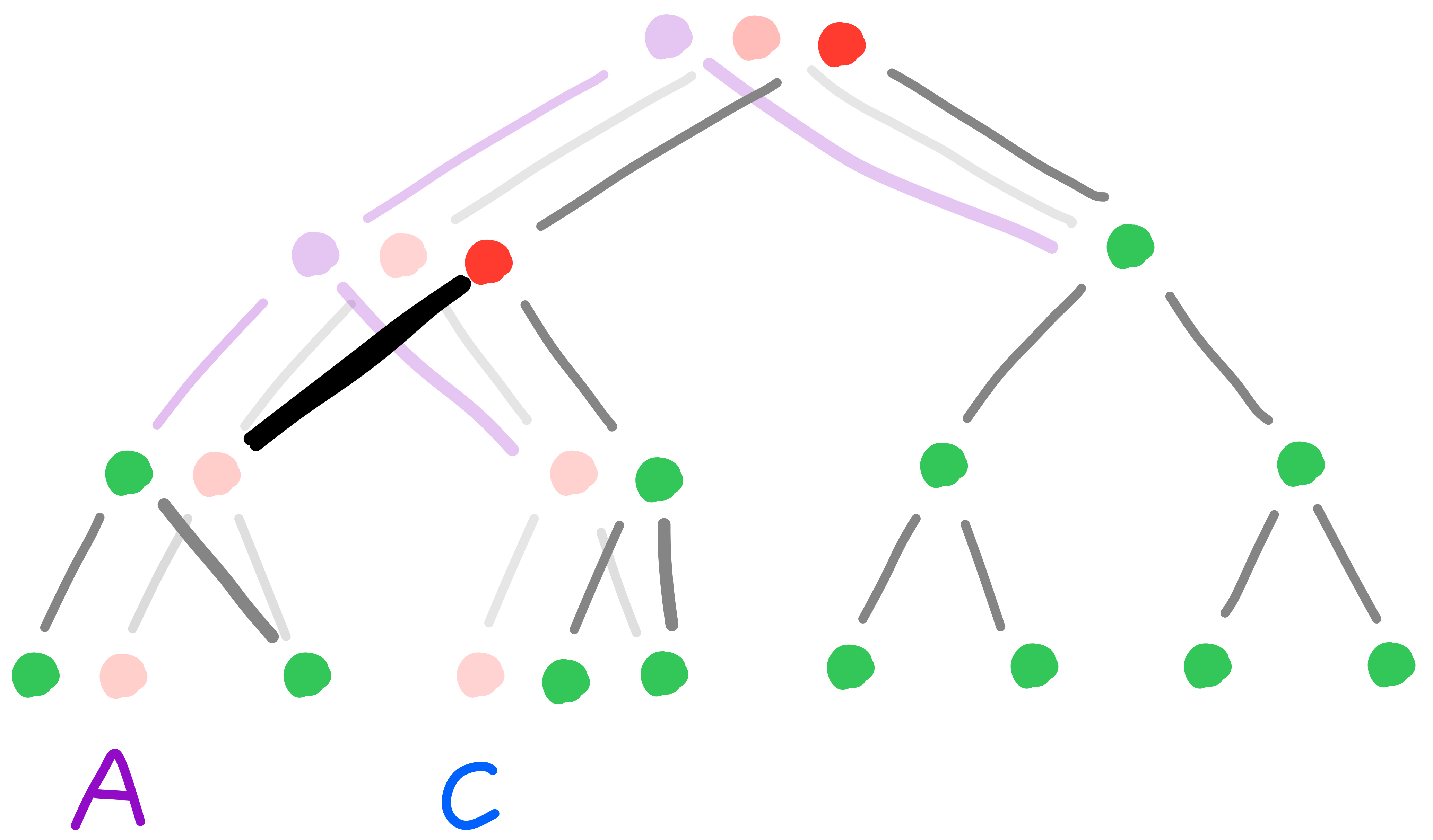
- A and C corrupted

Concurrent updates in CoCoA: PCS consequences



- A and C corrupted
- A and C update

Concurrent updates in CoCoA: PCS consequences



- leaked key (red)
- not leaked key (green)

- A and C corrupted
- A and C update
- Some keys encrypted to (red)
  - ↳ Still insecure
- Updating parties made progress

PCS of CoCoA

n – # users

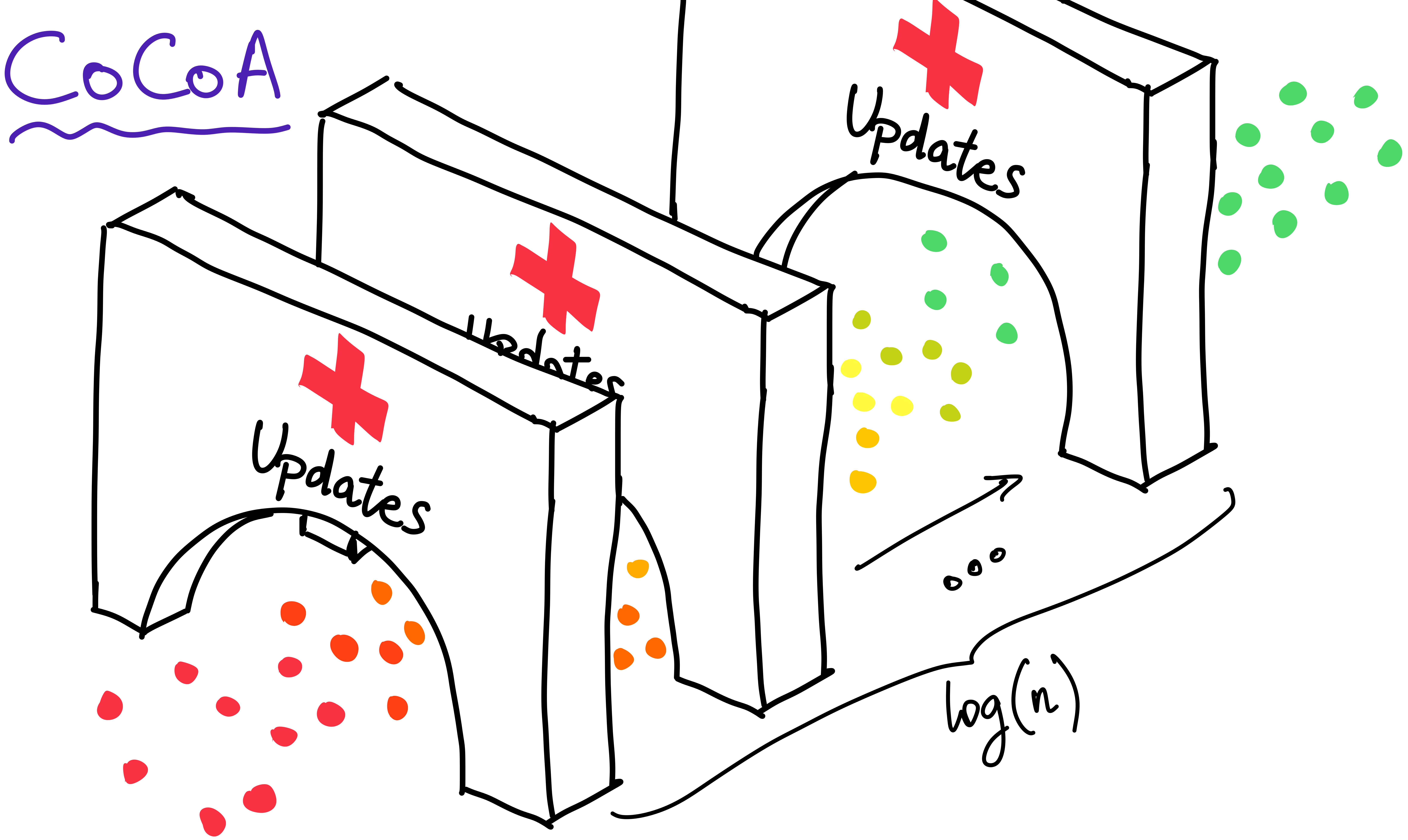
Theorem (informal):

Consider group key K in some round.

K is secure if for every user ID in the group, either:

(i) ID performed $\lceil \log(n) \rceil + 1$ updates since their last corruption

(ii) ID performed at least 1 update, where no other user updated concurrently, since their last corruption.


CoCoA
Updates
Updates
Updates
log(n)

Outline

1. Introduction: TreeKEM
   - Concurrency in TreeKEM
2. CoCoA
   - 2.1 Concurrent updates in CoCoA
   - 2.2 Efficiency
   - 2.3 Partial States
   - 2.4 Security
3. Summary & Open Problems

	Rounds to heal	Sender communication	Recipient Communication	Subsequent update cost (average / worst)
Plain TreeKEM (& variants)	n	$n \log(n)$	$n \log(n)$	$\log(n)$ / $\log(n)$
Propose & Commit	2	n	n	n / n
Bienstock et al. [TCC '21]	2	n	n	$\log(n)$ / n
Bidirectional Channels	2	n	n	n / n
CoCoA	$\lceil \log(n) \rceil + 1$	$\log^2(n)$	$\log^2(n)$	$\log(n)$ / $\log(n)$

Cost of healing per user, assuming:

- No knowledge of who corrupted + no coordination
- No further corruptions or Adds/Removes

	Rounds to heal	Sender communication	Recipient Communication	Subsequent update cost (average / worst)
Plain TreeKEM (& variants)	n	$n \log(n)$	$n \log(n)$	$\log(n)$ / $\log(n)$
Propose & Commit	2	n	n	n / n
Bienstock et al. [TCC '20]	2	n	n	$\log(n)$ / n
Bidirectional Channels	2	n	n	n / n
CoCoA	$\lceil \log(n) \rceil + 1$	$\log^2(n)$	$\log^2(n)$	$\log(n)$ / $\log(n)$

Cost of healing per user, assuming:

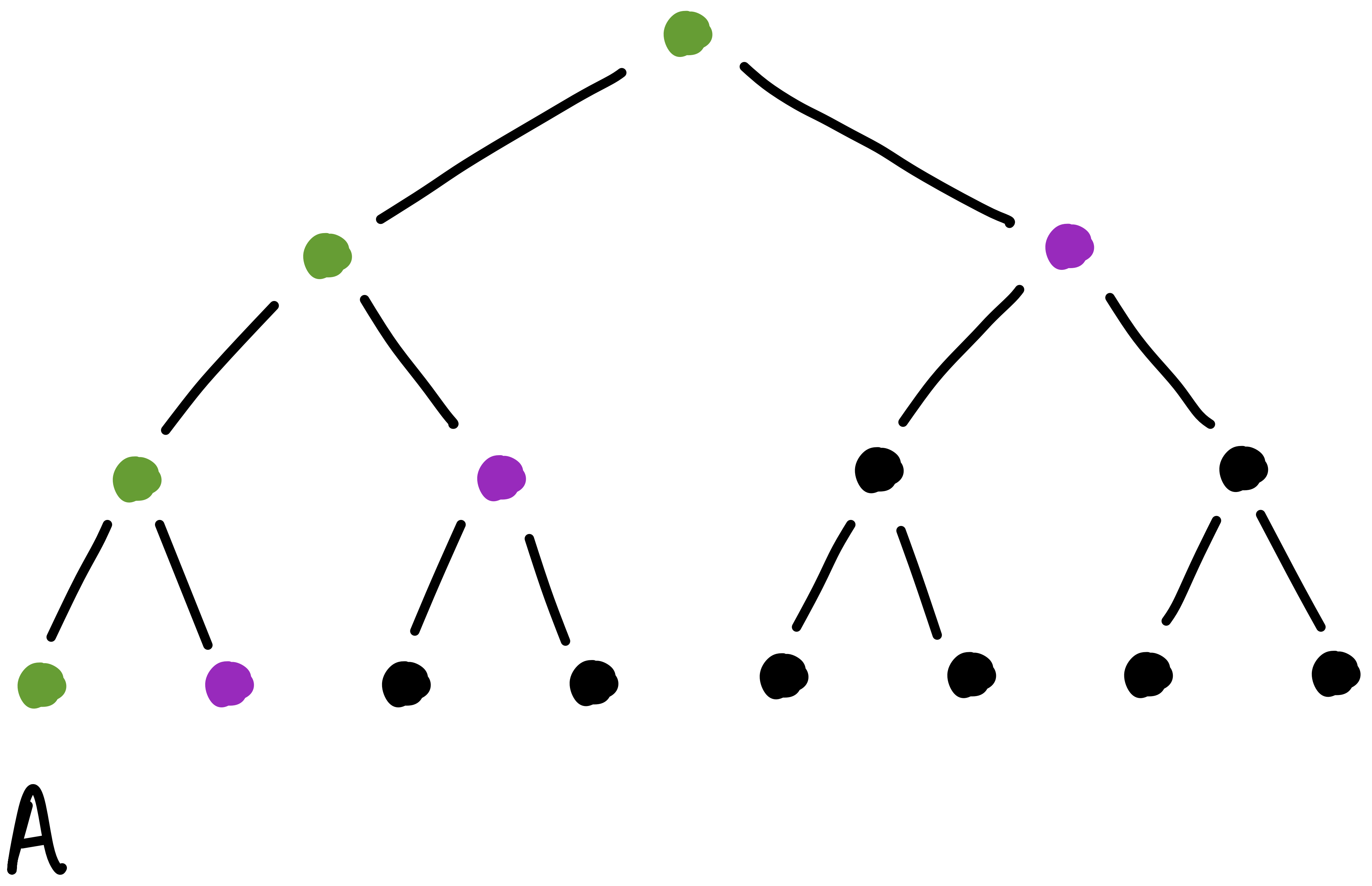
- No Knowledge of who corrupted + no coordination
- No further corruptions or Adds/Removes

Outline

1. Introduction: TreeKEM
   - Concurrency in TreeKEM
2. CoCoA
   2.1 Concurrent updates in CoCoA
   2.2 Efficiency
   2.3 Partial States
   2.4 Security
3. Summary & Open Problems

Partial States

→ Keeping track of whole tree
⇒ download linear in # updaters



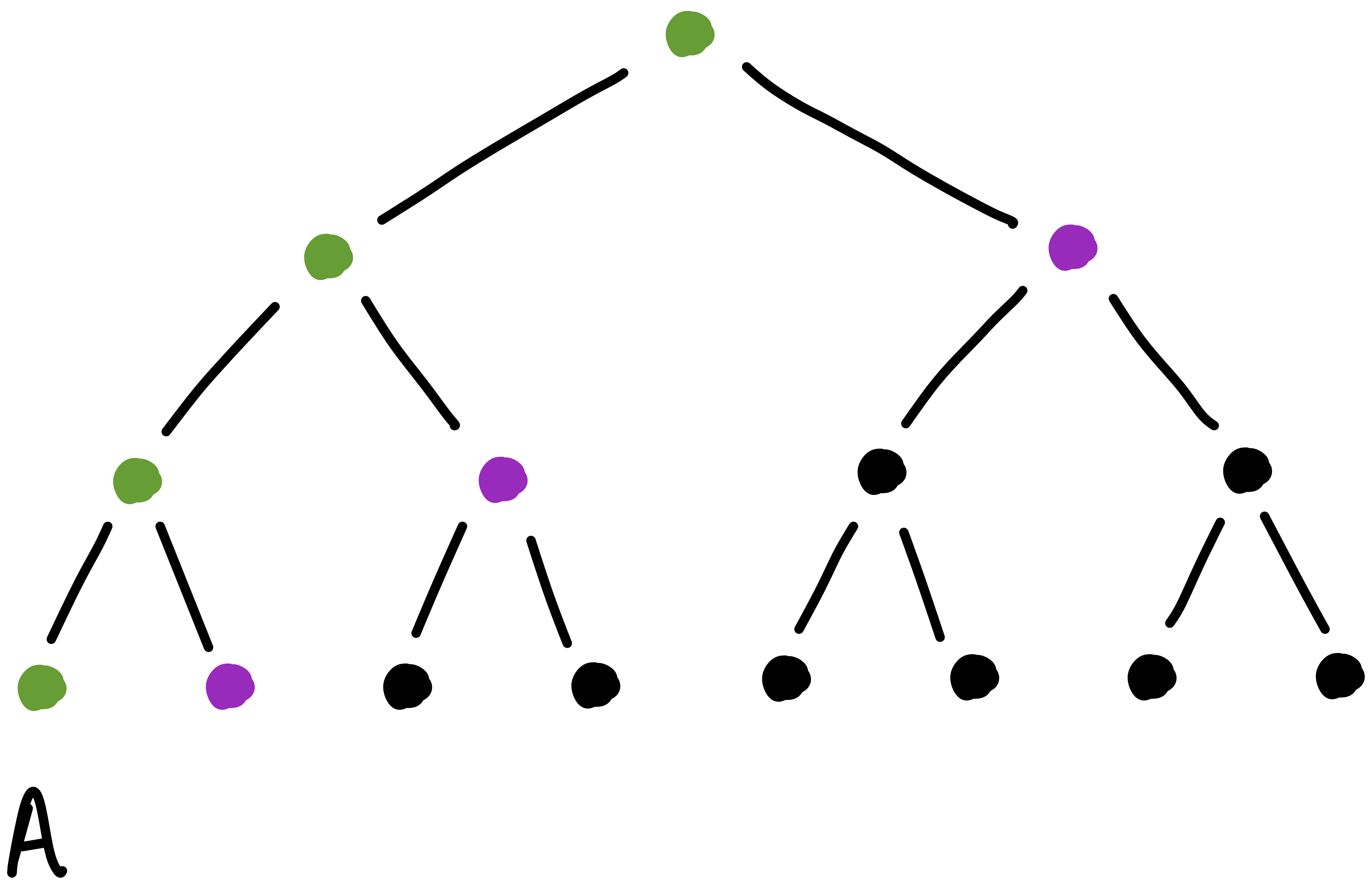
Nodes A knows secret Key of

Nodes A encrypts to

Partial States

→ Keeping track of whole tree
⟹ download linear in # updaters

→ A only needs Keys for •, • nodes



● Nodes A knows secret Key of

● Nodes A encrypts to

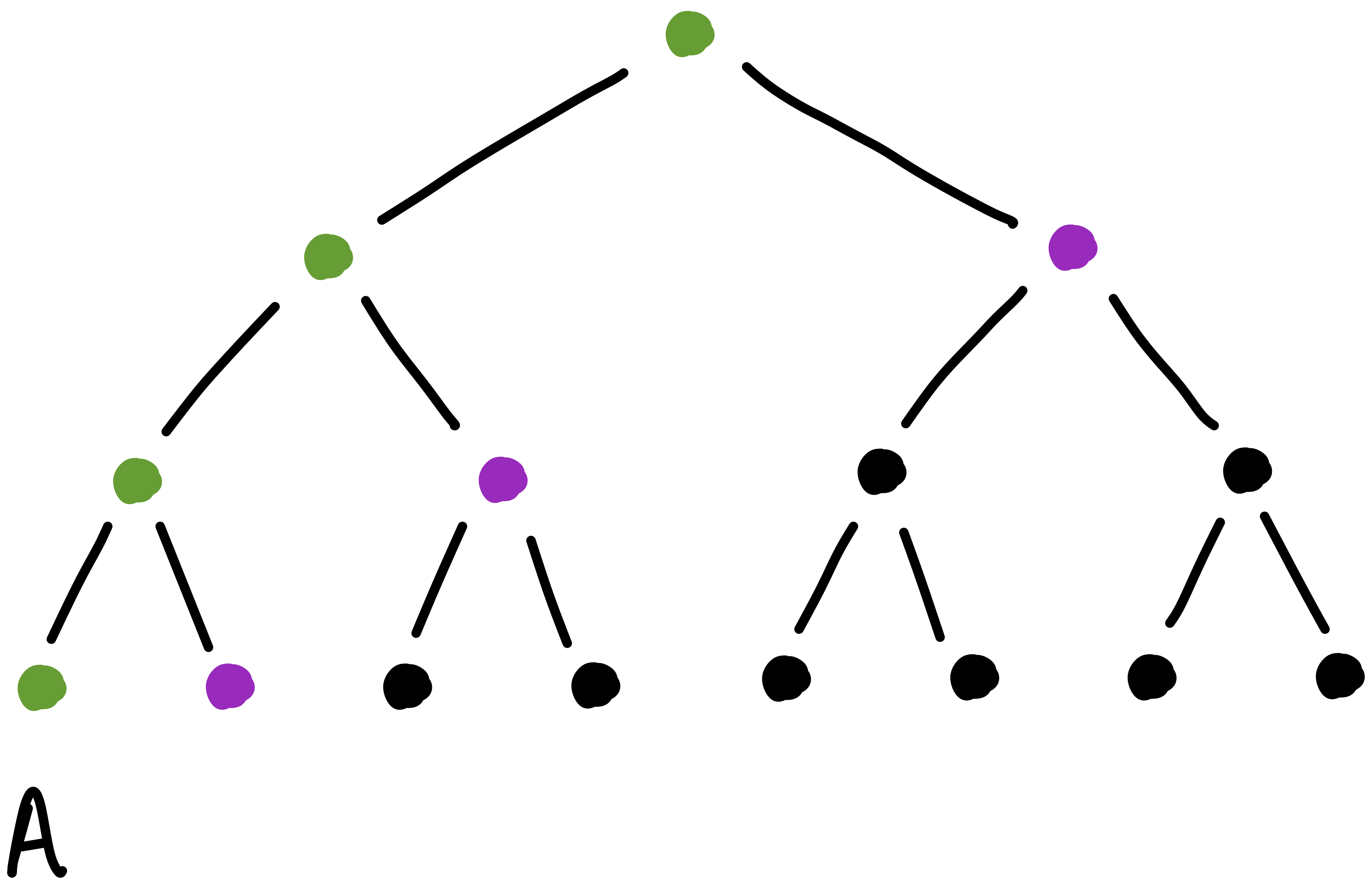
Partial States

→ Keeping track of whole tree
⟹ download linear in # updaters

→ A only needs Keys for •, • nodes

→ Server only relays packets for:

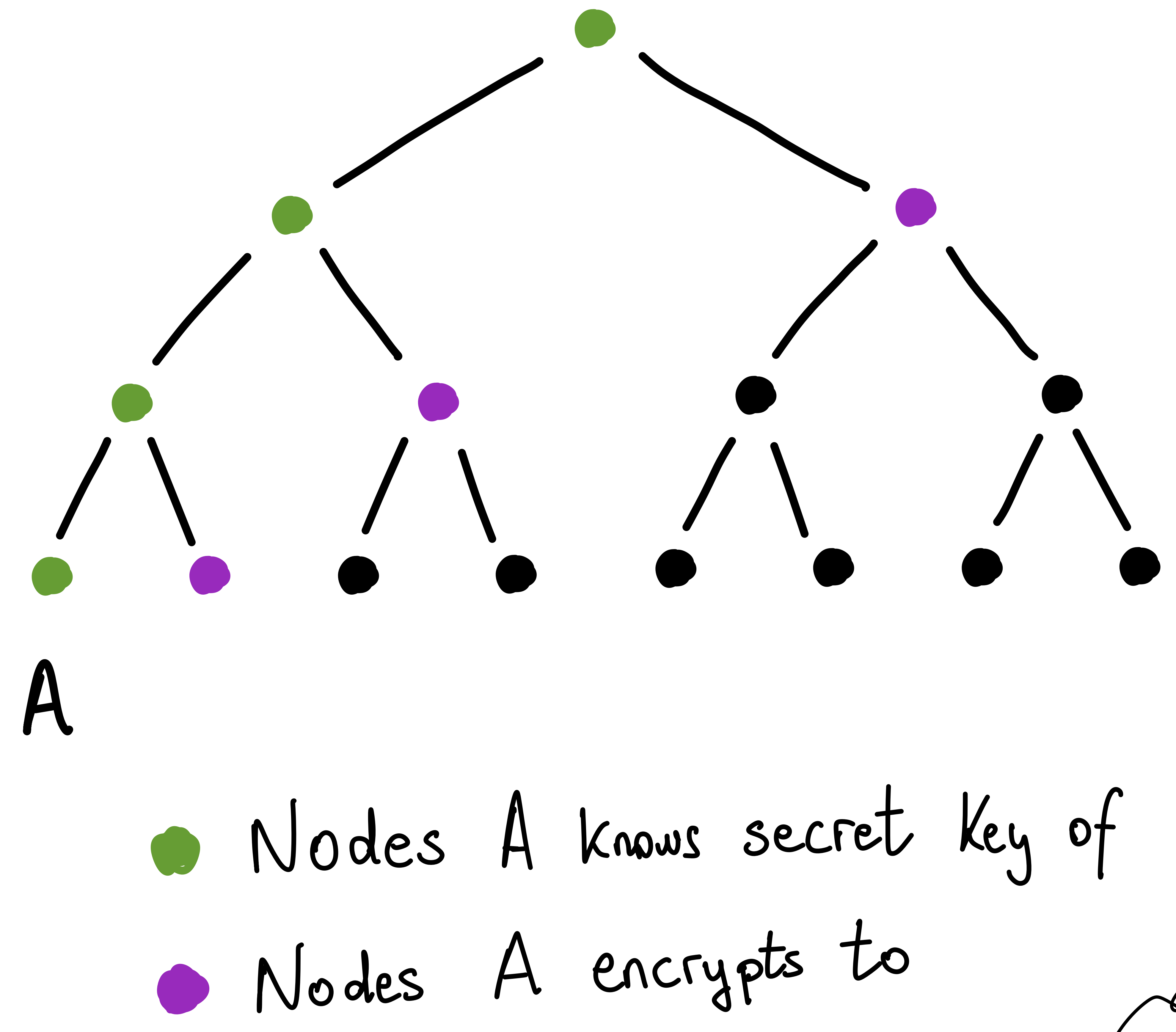
- Adds & Removes
- Update info for •, •



• Nodes A knows secret Key of

• Nodes A encrypts to

Partial States



→ Keeping track of whole tree
⟹ download linear in # updaters

→ A only needs keys for •, • nodes

→ Server only relays packets for:

- Adds & Removes
- Update info for •, •

→ Several new challenges!

Partial States: Challenge I - Consistency

- Consistency in TreeKEM ensured through:
  - ↳ Transcript hash & Tree Hash

Transcript hash → Hash of group history

Tree Hash ← Commitment to tree

Partial States: Challenge I – Consistency

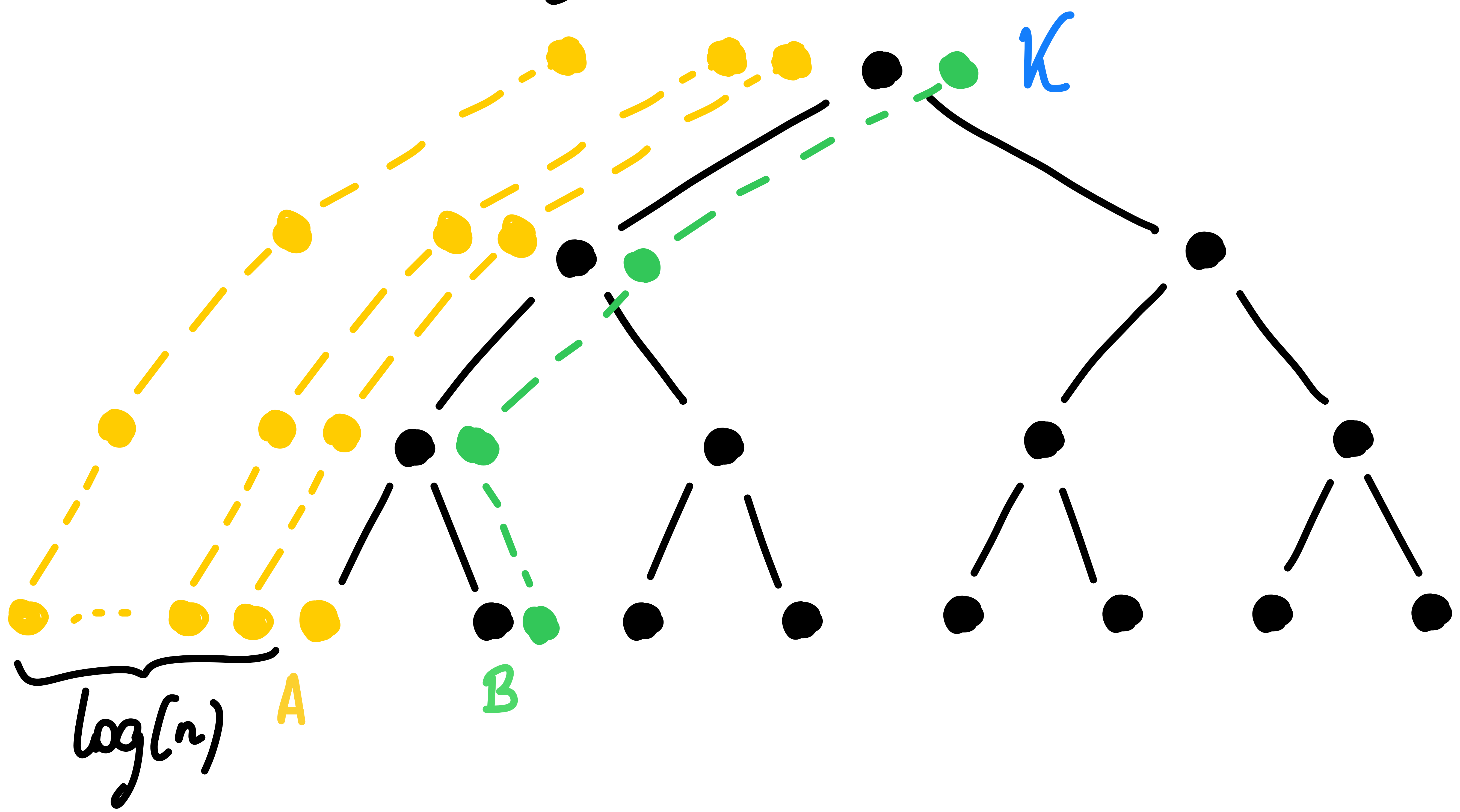
- Consistency in TreeKEM ensured through:
  - ↳ Transcript hash & Tree Hash
    - Transcript hash → Hash of group history
    - Tree Hash ← Commitment to tree
- Users in CoCoA not aware of all updates or keys

Partial States: Challenge I – Consistency

- Consistency in TreeKEM ensured through:
  ↳ Transcript hash & Tree Hash
  - Transcript hash → Hash of group history
  - Tree Hash ← Commitment to tree
- Users in CoCoA not aware of all updates or keys
- Server "completes" view of users through commitments to sub-trees.

Partial States: Challenge II – Defining Process

Example:
- B updates, sampling key K
- A updated log(n) times since last corruption
- Server ignored A's updates when talking to B
- Is K secure?



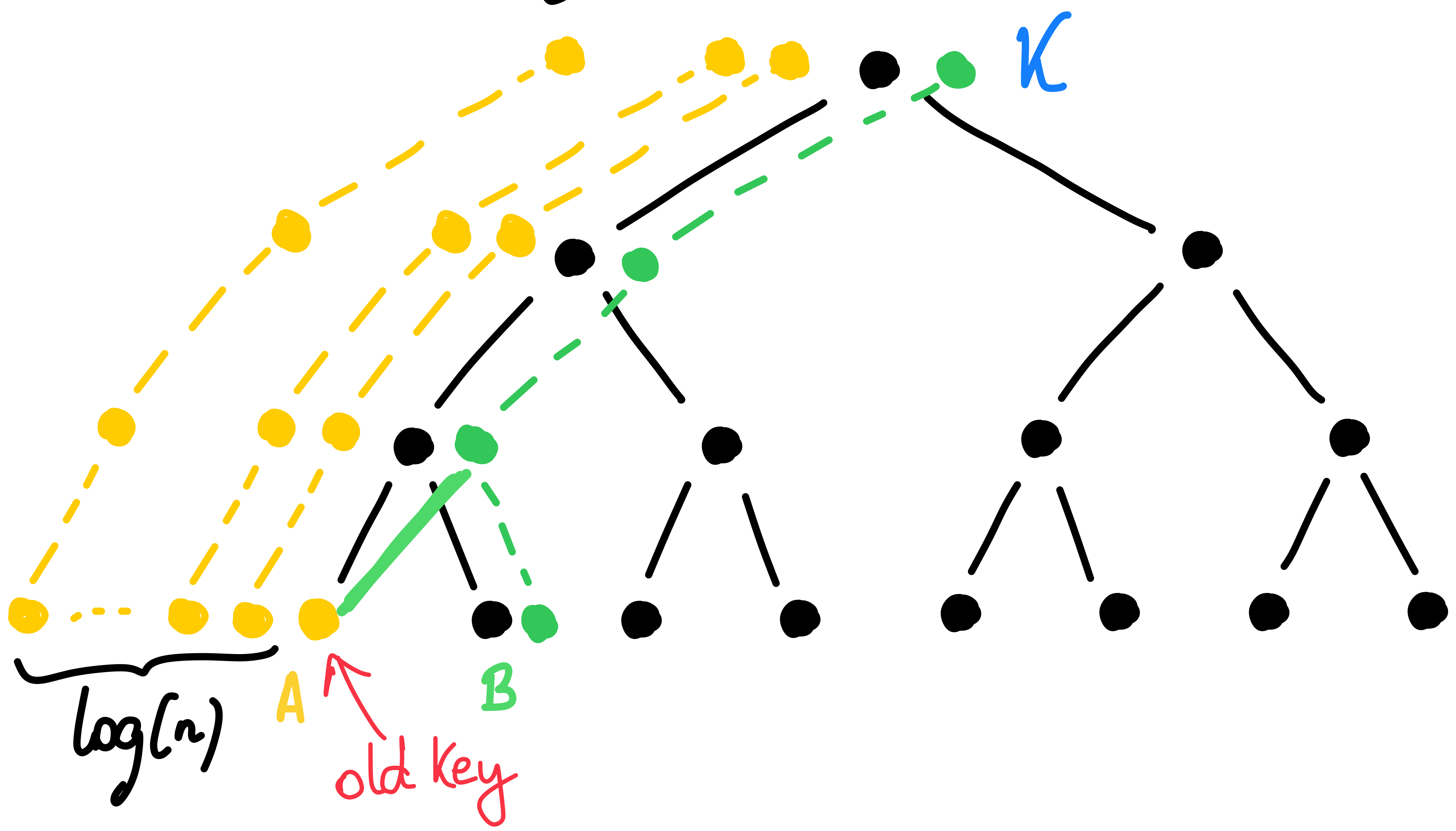
Partial States: Challenge II - Defining Process

Example: • B updates, sampling key K

• A updated log(n) times since last corruption

• Server ignored A's updates when talking to B

• Is K secure? No!

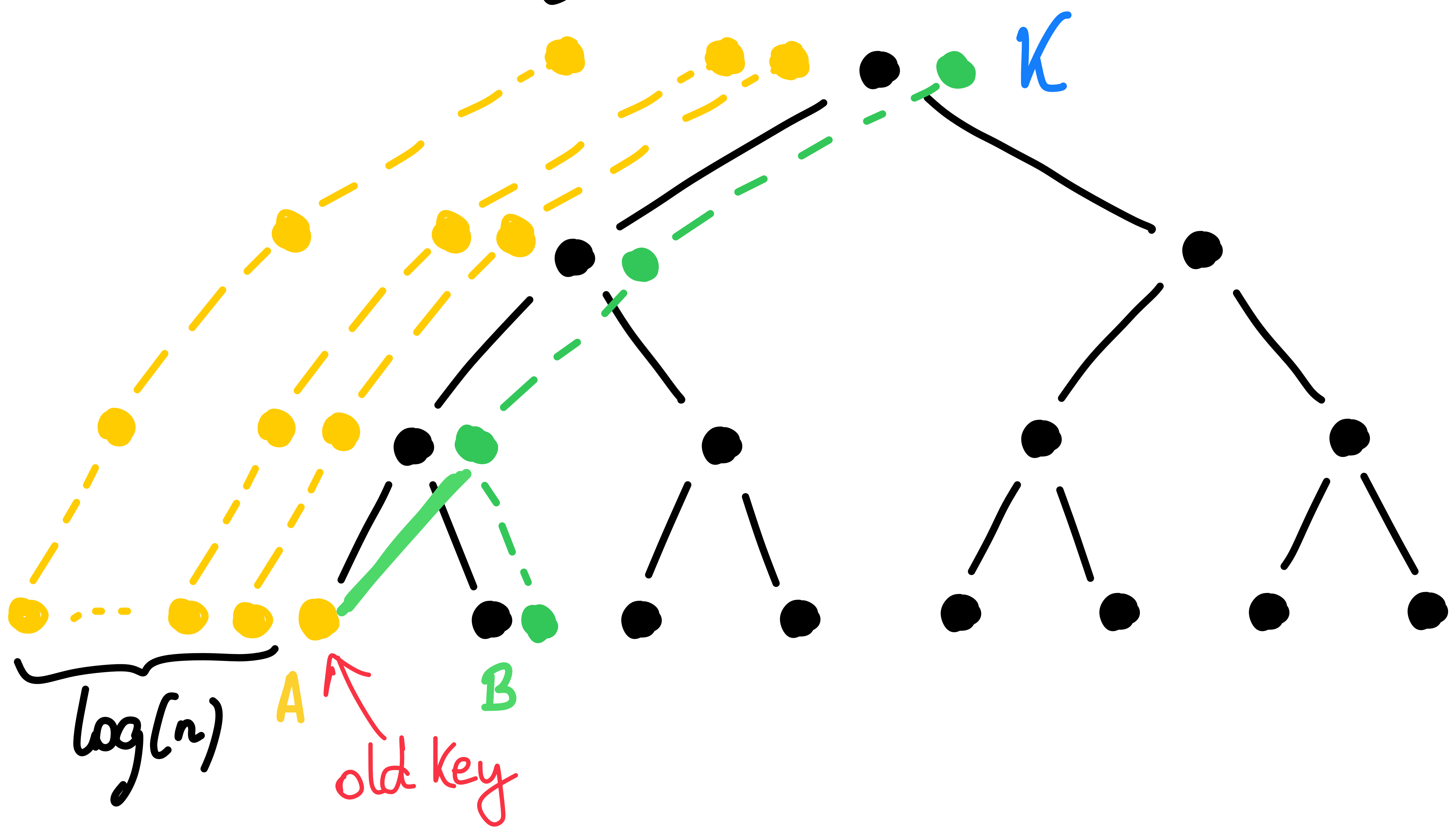


Partial States: Challenge II – Defining Process

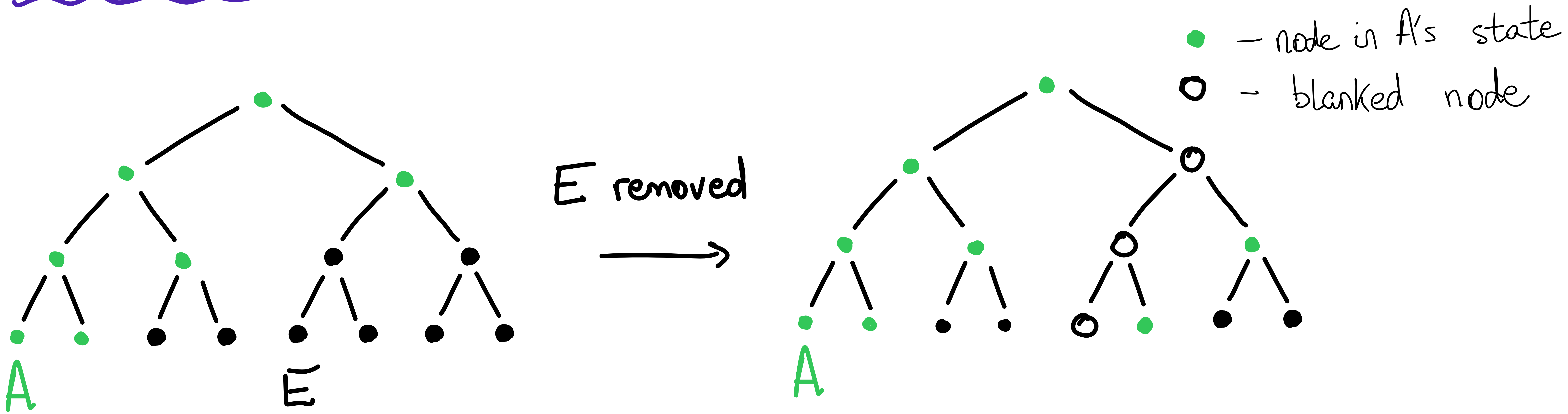
Example:
- B updates, sampling key K
- A updated log(n) times since last corruption
- Server ignored A's updates when talking to B
- Is K secure? No!

⟹ Need notion of B processing A's updates!

Non-trivial!



Partial States: Challenge III – Malicious Server



→ Server sends A public keys for new nodes in state
(no guarantee any party knowing them is online)

→ How does A guarantee keys are correct?

Outline

1. Introduction: TreeKEM
   - Concurrency in TreeKEM
2. CoCoA
   - 2.1 Concurrent updates in CoCoA
   - 2.2 Efficiency
   - 2.3 Partial States
   - 2.4 Security
3. Summary & Open Problems

Security

We prove CoCoA secure in the ROM:

→ Against adaptive, partially active adversaries

(partially active: control server, cannot impersonate)

→ Polynomial loss

→ Adapted proof of [Klein et al. S&P '21], for (Tainted) TreeKEM.

Summary

- CoCoA: CGKA can recover arbitrary corruption in log(n) rounds without degrading efficiency
  - ↳ Circumvent lowerbound by relaxing PCS
- CGKA with partial states possible
  - ↳ Lower recipient communication

Open Problems

- Better understand tradeoff between communication cost & rounds for PCS.
- Active security for CoCoA

Open Problems

- Better understand tradeoff between communication cost & rounds for PCS.
- Active security for CoCoA

Related work: "DeCAF: Decentralizable CGKA with Fast Healing"
ia.cr/2022/559

→ PCS in log(t) rounds, but no partial states
(t: # corruptions)