Probabilistic Broadcast

Course leader: Professor Seif Haridi

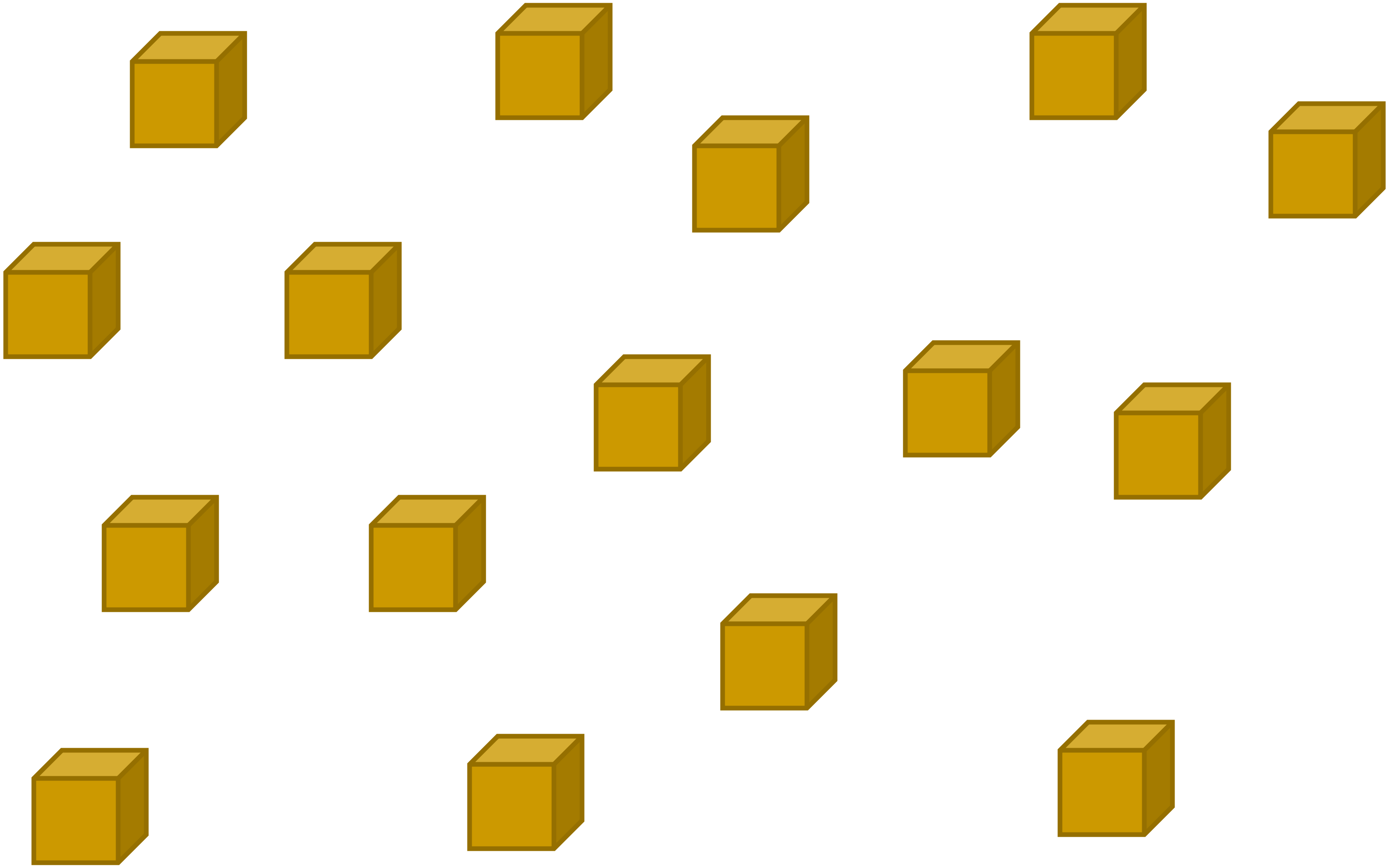
Assistants: **Cosmin Arad**, Tallat Shafaat

{haridi, icarad, tallat}@kth.se

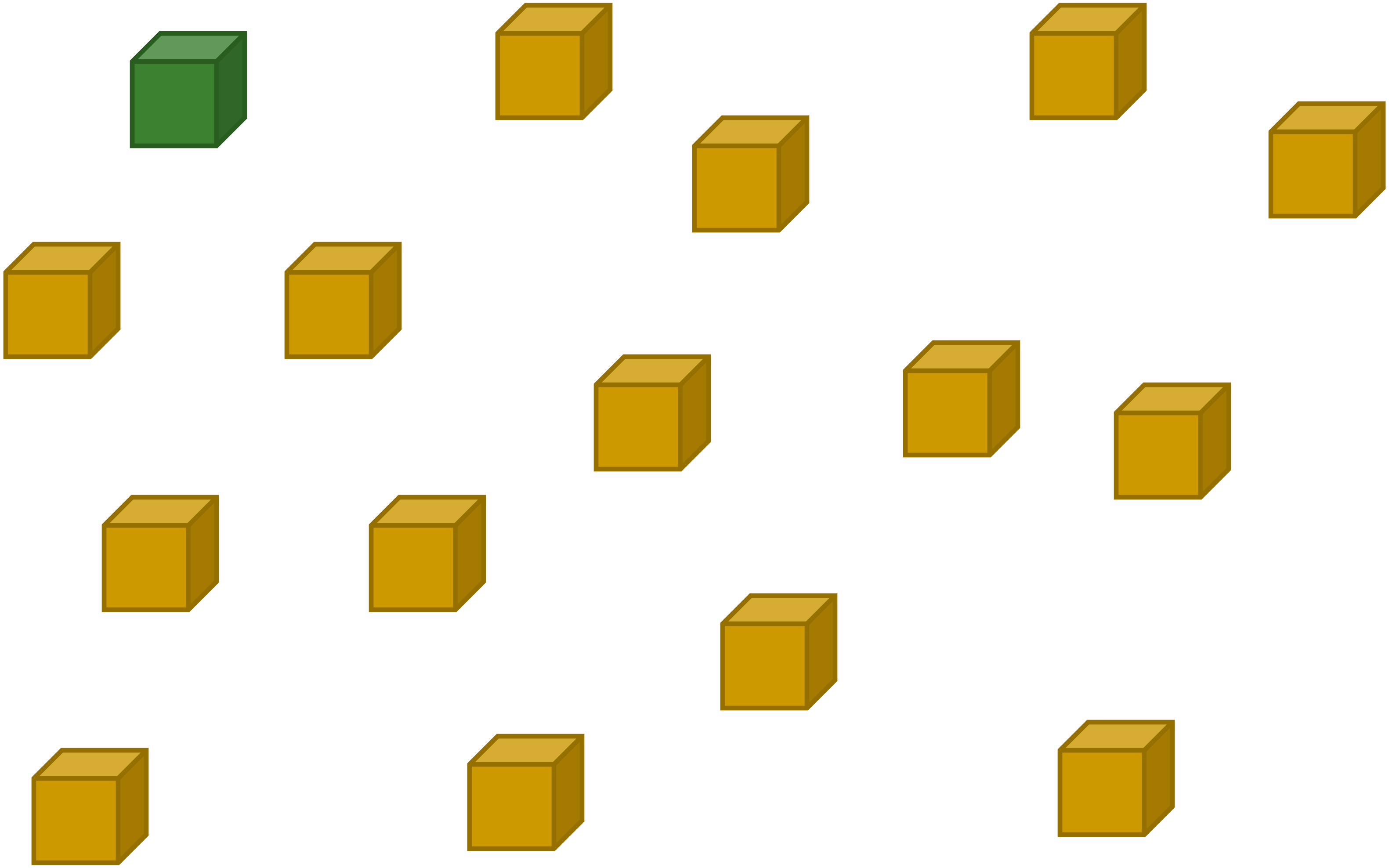
Motivation

- As systems grow larger and larger
  - Deterministic reliability is more and more costly
- Solution
  - Trade deterministic reliability guarantees for scalability
  - Provide probabilistic reliability guarantees
- Popular implementation technique
  - Epidemic dissemination (aka gossip)

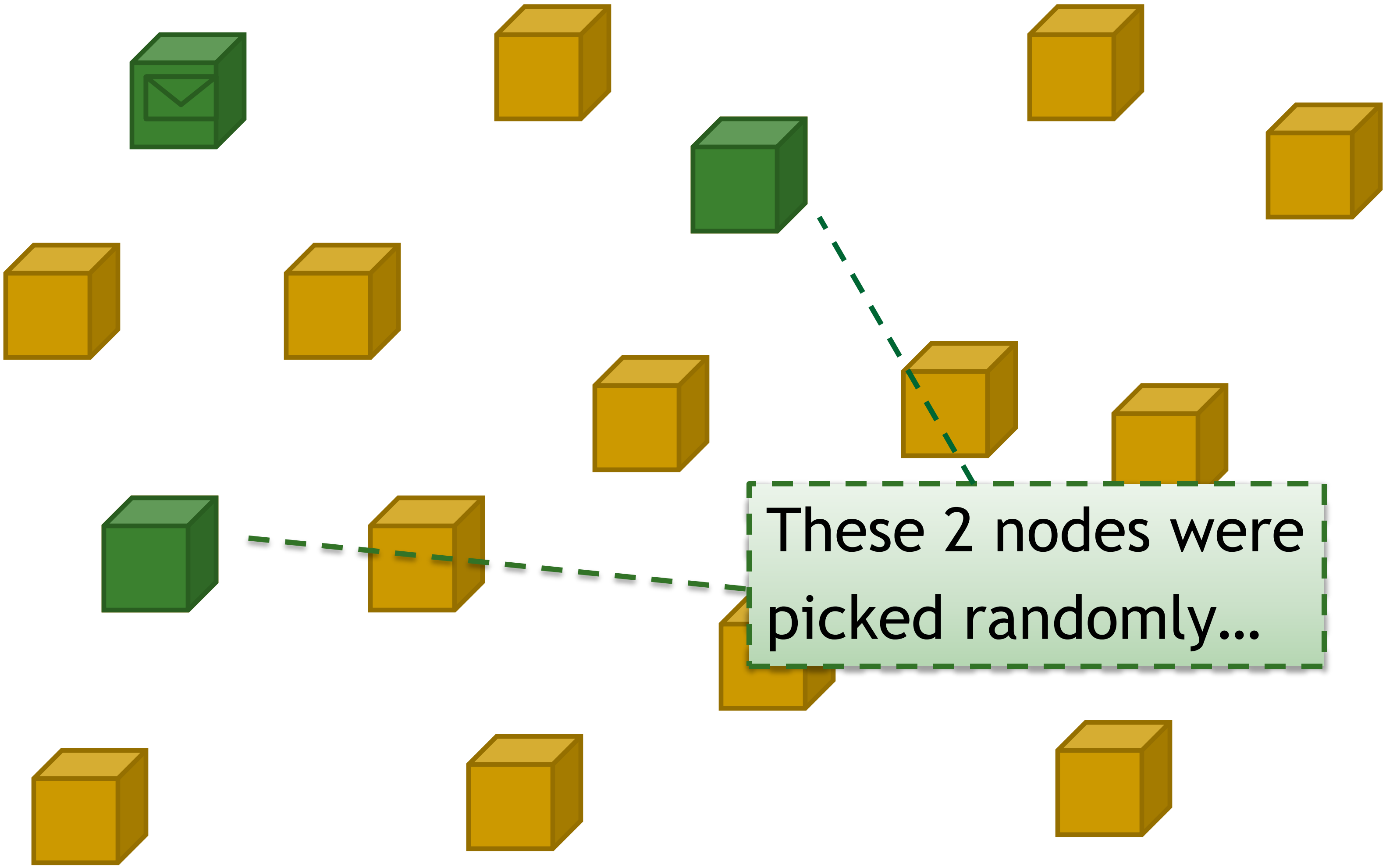
Epidemic dissemination



Epidemic dissemination, round 0

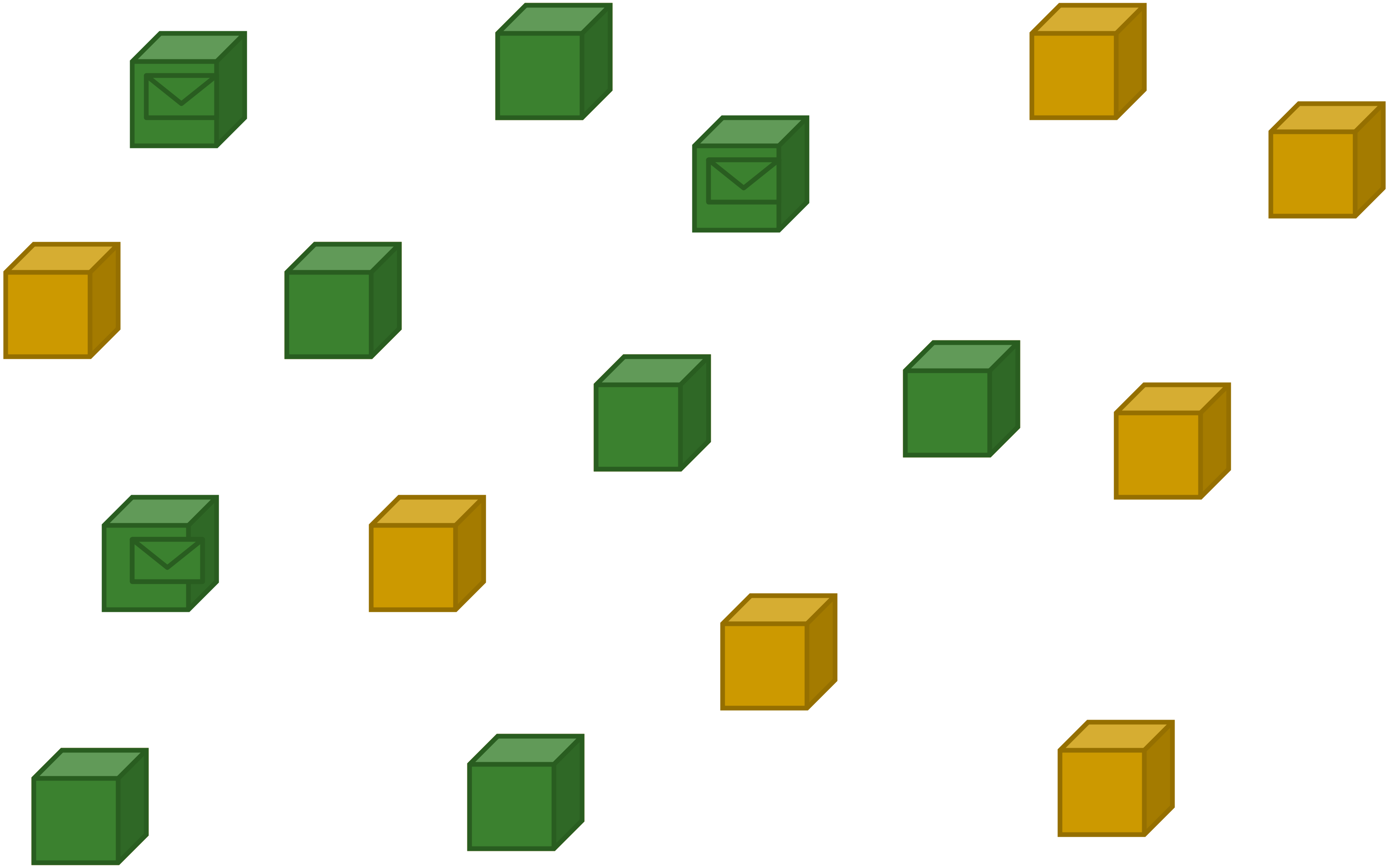


Epidemic dissemination, round 1

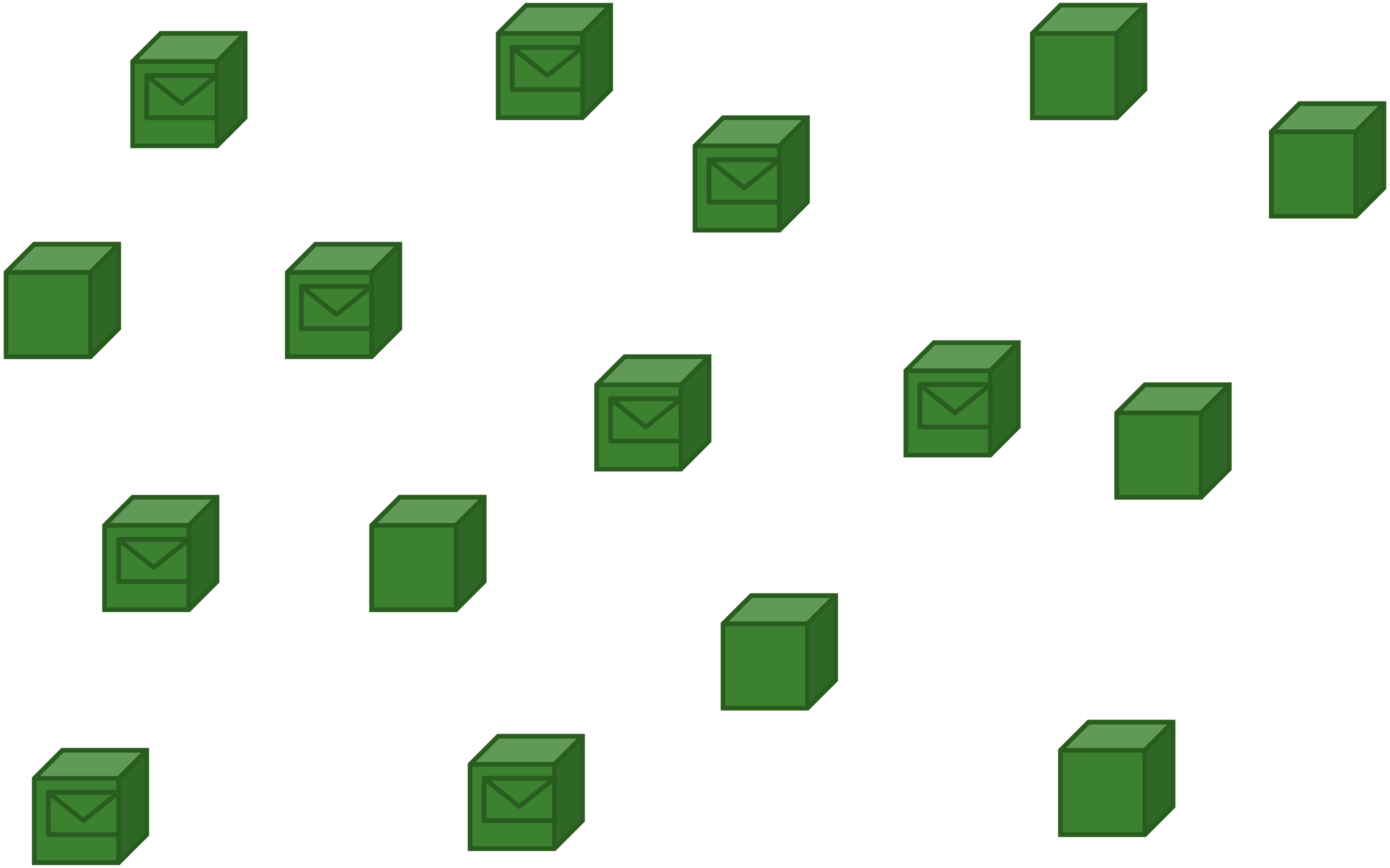


These 2 nodes were picked randomly…

Epidemic dissemination, round 2



Epidemic dissemination, round 3



Epidemic dissemination Pro&Con

- Advantages
  - Fast dissemination (low number of rounds)
  - Cheap (constant # of messages per round)
  - Nodes are equally loaded (no bottlenecks)
  - Simple (compared with building a tree)

- Disadvantaged
  - Redundancy
  - Not guaranteed to reach all nodes

Epidemic dissemination params

- Fanout
  - # of nodes contacted by one node in each round
  - 2 in our previous example
- Max rounds
  - Maximum number of rounds for which a message is going to be retransmitted

Probabilistic broadcast (pb)

- ***Events***
  - Request: ⟨pbBroadcast | m⟩
  - Indication: ⟨pbDeliver | src, m⟩

- ***Properties: PB1, PB2, PB3***
  - ***PB1: Probabilistic Validity***
  - ***PB2: No duplication***
  - ***PB3: No creation***

Probabilistic broadcast (pb)

- ***Intuitively***: messages are delivered with a certain probability

- ***Properties***
  - ***Probabilistic Validity:*** There is a given non-zero probability such that for any two correct processes *pi* and *pj*, every message broadcast by *pi* is eventually delivered by *pj* with this probability
  - ***No duplication:*** No message delivered more than once
  - ***No creation:*** No message delivered unless broadcast

Algorithms

- Eager probabilistic broadcast
- Lazy probabilistic broadcast

Eager probabilistic broadcast

- Main idea:
  - Similar to our previous example
  - Upon *pbBroadcast*, node picks *fanout* targets
    - Distinct and different from itself
  - Sends the message to targets
    - Message carries a time-to-live (TTL)
      - Initially TTL=*maxrounds*
    - May use fair-loss links
  - Receivers deliver filtering duplicates
  - They forward the message with a $TTL_{new}=TTL_{msg}-1$
    - To *fanout* distinct and different random nodes…

Lazy probabilistic broadcast

- Main idea:
  - Use a cheap unreliable broadcast to disseminate a message
  - Some nodes randomly save the message
  - Use gossiping to recover any lost messages
- How to know if a message is lost?
  - Sender tags messages with sequence numbers
  - Gap in sequence numbers of received messages indicates loss of messages