Distributed Systems

Spring Semester 2020

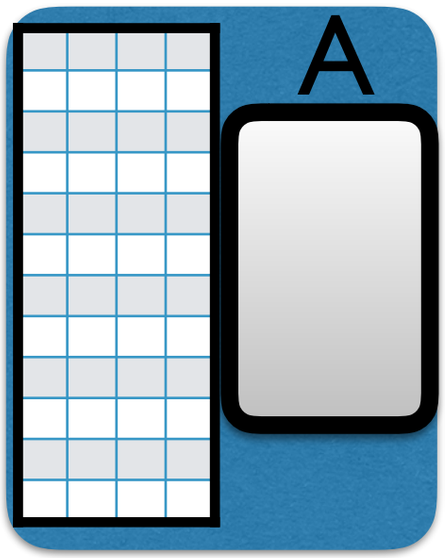
Lecture 16: Bayou (Eventual Consistency)

John Liagouris
liagos@bu.edu

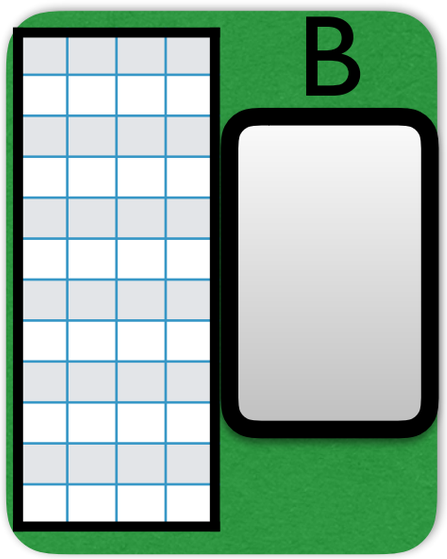
- Will updates be consistent with wall clock time?
- Will updates be consistent with causality?

Wall Clock

<10,A>: op

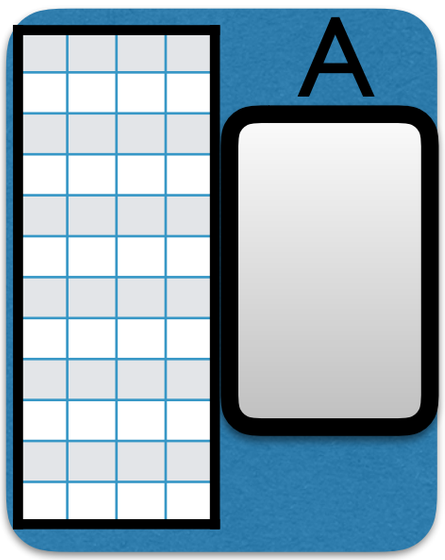


- Lets assume that A does it's op first and its clock reads 10



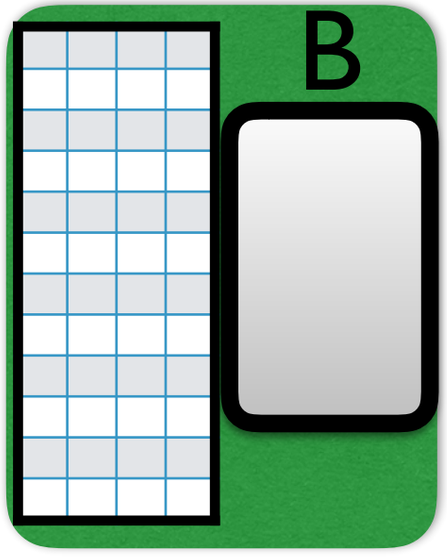
Wall Clock

<10,A>: op



- Lets assume that A does it's op first and its clock reads 10

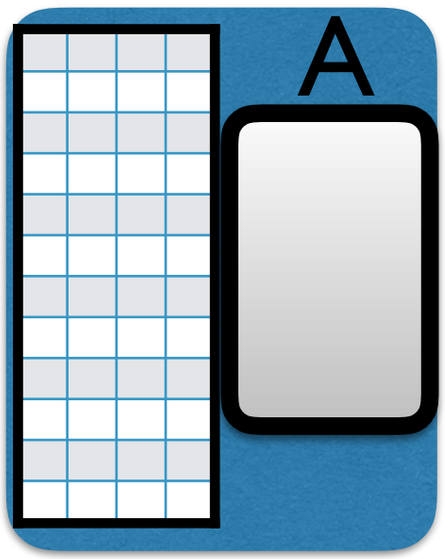
<9,B>:op



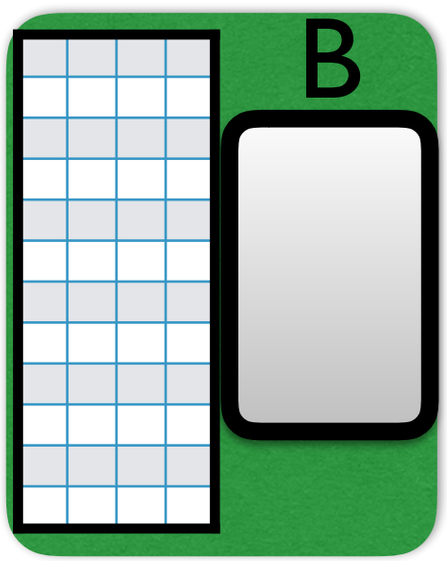
- Then B does its op but its clock is slow so and it op is at 9

Wall Clock

<10,A>: op



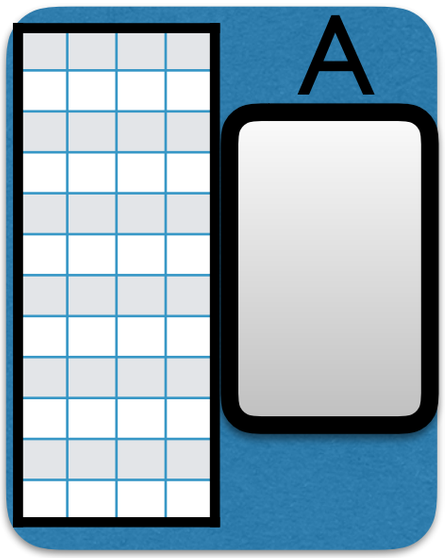
<9,B>:op



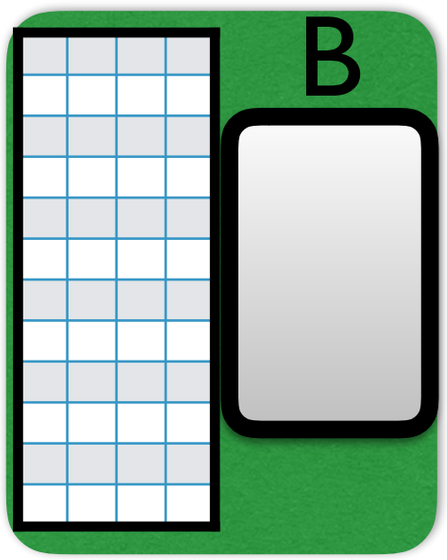
- Lets assume that A goes and op first and its clock reads 10
- Then B does its op but its clock is slow so and it op is at 9
- After sync both with see the order : <9,B> then <10,A>
- NOT EXTERNALLY CONSISTENT

Causally Consistency

<10,A>: op



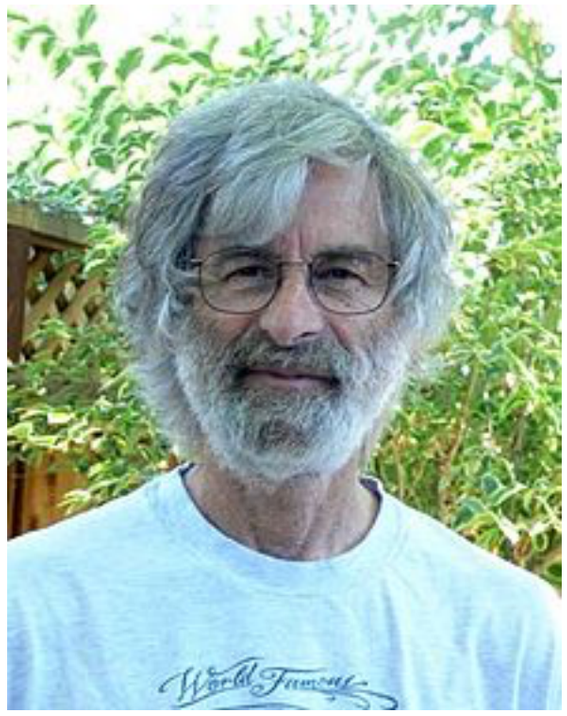
<9,B>:op



- A Adds a Meeting
- B sees A's meeting and deletes it
- But given clock B's clock being slow you can see delete before add :-(

LAMPORT LOGICAL CLOCKS

- This is a critical idea
- Let's look a little more closely in the context of using a logical clock in BAYOU for global time stamps that preserve causal consistency



Leslie Lamport

The Happened-before relationship

Problem

We first need to introduce a notion of ordering before we can order anything.

The happened-before relation

- If a and b are two events in the same process, and a comes before b, then $a \rightarrow b$.
- If a is the sending of a message, and b is the receipt of that message, then $a \rightarrow b$
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Note

This introduces a partial ordering of events in a system with concurrently operating processes.

The Happened-before relationship

Problem

We first need to introduce a notion of ordering before we can order anything.

The happened-before relation

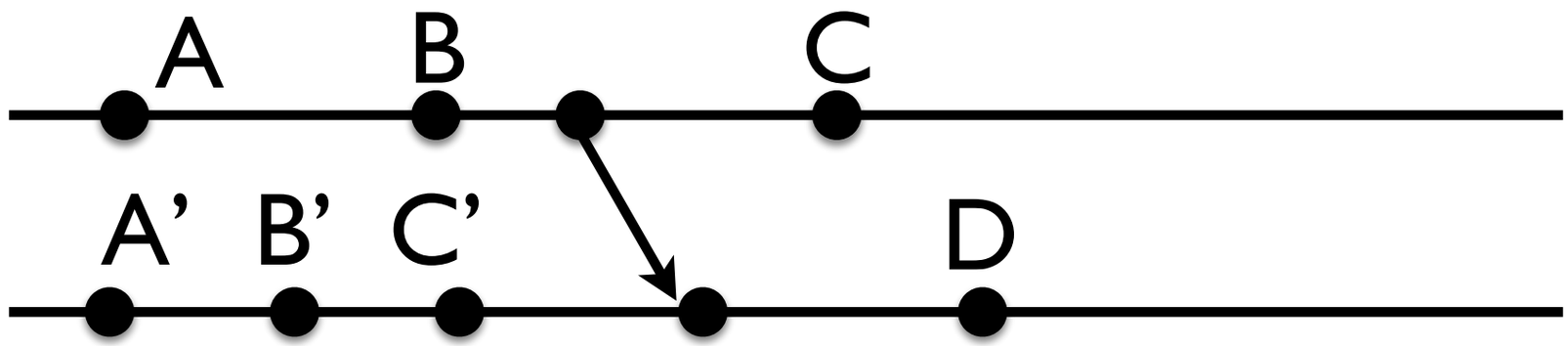
- If a and b are two events in the same process, and a comes before b, then $a \rightarrow b$.
- If a is the sending of a message, and b is the receipt of that message, then $a \rightarrow b$
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Note

This introduces a partial ordering of events in a system with concurrently operating processes.

The Happened-before relationship

- Why **partial** order?
- Thread 1:
- Thread 2:



- A < B < C and A < B < D
- A' < B' < C' < D
- A, B, C are **incomparable** with A', B', C' — We don't know their actual order

Logical clocks

Problem

How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?

Solution

Attach a timestamp $C(e)$ to each event e, satisfying the following properties:

- P1 If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$.
- P2 If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Problem

How to attach a timestamp to an event when there's no global clock $\Rightarrow$ maintain a consistent set of logical clocks, one per process.

Logical clocks

Problem

How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?

Solution

Attach a timestamp $C(e)$ to each event e, satisfying the following properties:

- P1 If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$.
- P2 If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Problem

How to attach a timestamp to an event when there's no global clock $\Rightarrow$ maintain a consistent set of logical clocks, one per process.

Logical clocks

Problem

How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?

Solution

Attach a timestamp $C(e)$ to each event e, satisfying the following properties:

- P1 If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$.
- P2 If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Problem

How to attach a timestamp to an event when there's no global clock $\Rightarrow$ maintain a consistent set of logical clocks, one per process.

Logical clocks

Solution

Each process P_i maintains a local counter C_i and adjusts this counter according to the following rules:

1: For any two successive events that take place within P_i, C_i is incremented by 1.

2: Each time a message m is sent by process P_i, the message receives a timestamp $ts(m) = C_i$.

3: Whenever a message m is received by a process P_j, P_j adjusts its local counter C_j to $\max\{C_j, ts(m)\}$; then executes step 1 before passing m to the application.

Notes

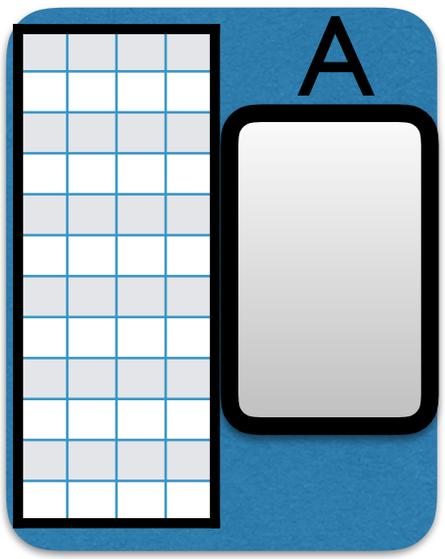
- Property P1 is satisfied by (1); Property P2 by (2) and (3).
- It can still occur that two events happen at the same time. Avoid this by breaking ties through process IDs.

Logical Clocks

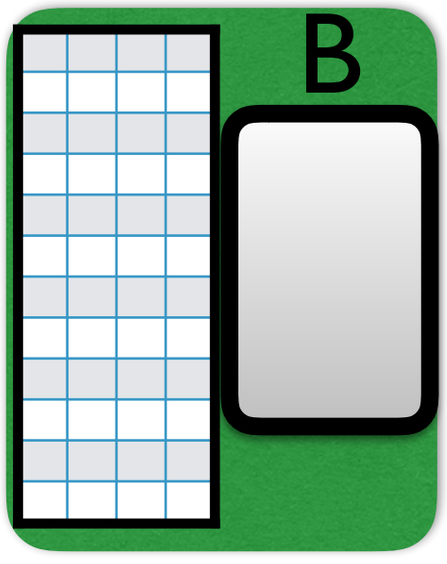
- Want to timestamp events s.t.
- if node observes E1, then generates E2, then TS(E2) > TS(E1)
- So all nodes will order E1, then E2
- Tmax = highest time-stamp seen from any node (including self)
- T = max(Tmax + 1, wall-clock time) -- to generate a timestamp
- Properties
  - E1 then E2 on same node => TS(E1) < TS(E2)
  - BUT TS(E1) < TS(E2) does not imply E1 came before E2

Causally Consistency

<10,A>: op



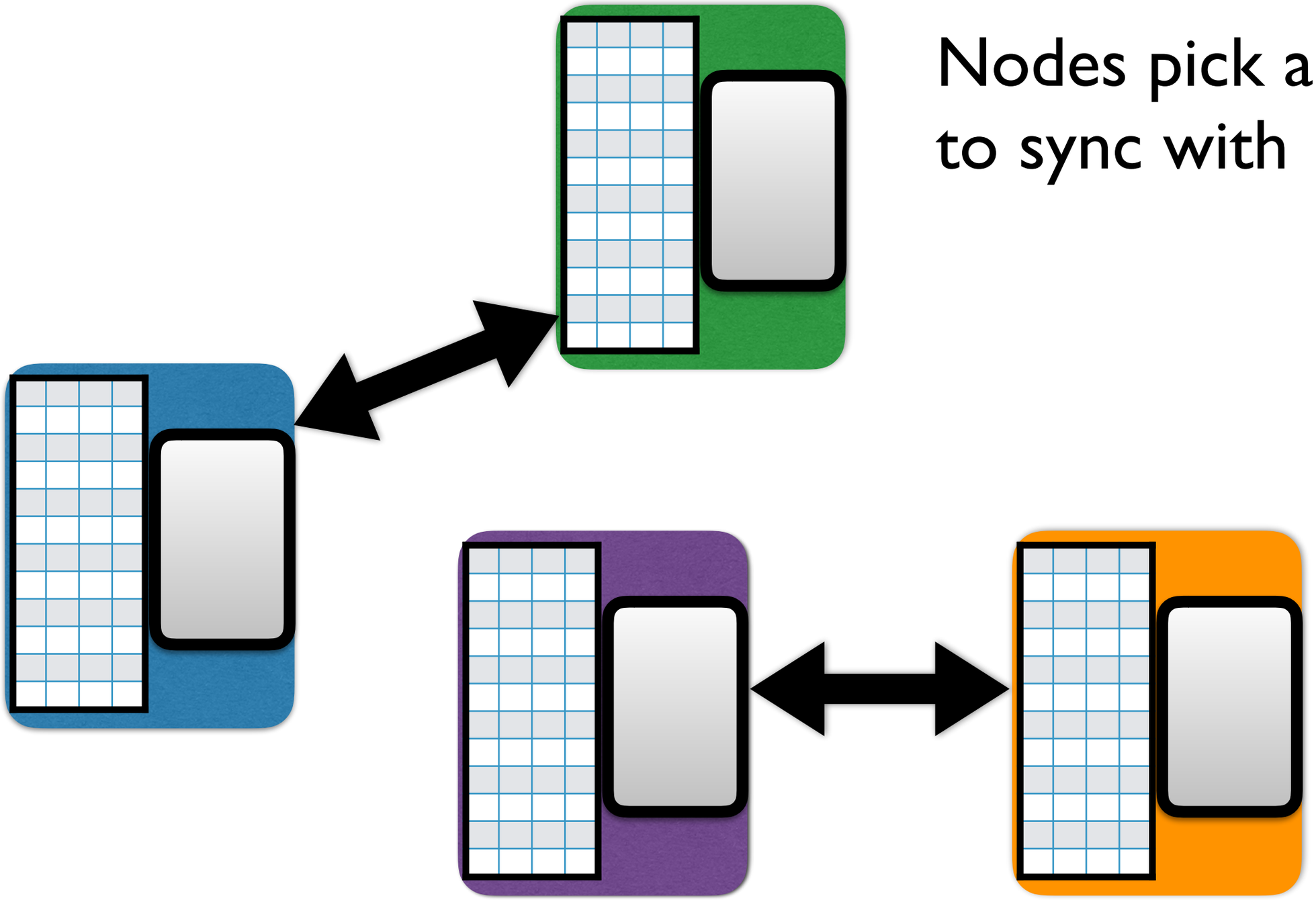
<9,B>:op



- A Adds a Meeting
- B sees A's meeting and Deletes it
- But when B sync to see A's meeting its clock will advance to max(Tmax=10 + 1, local wall clock) so its Delete will be ordered after A's Add

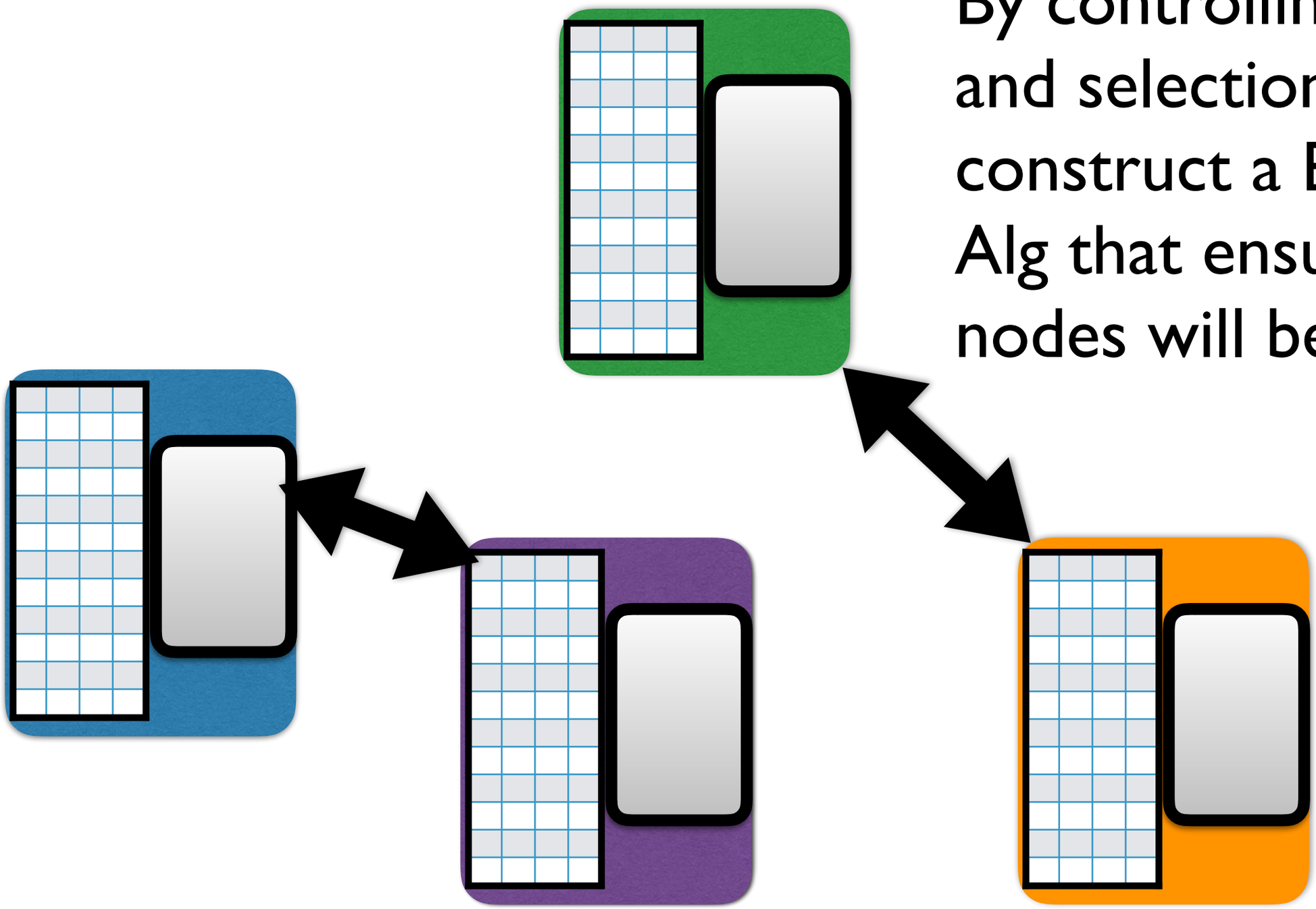
Logical Clocks Fix our Causality problem

Gossip & Epidemic Alg



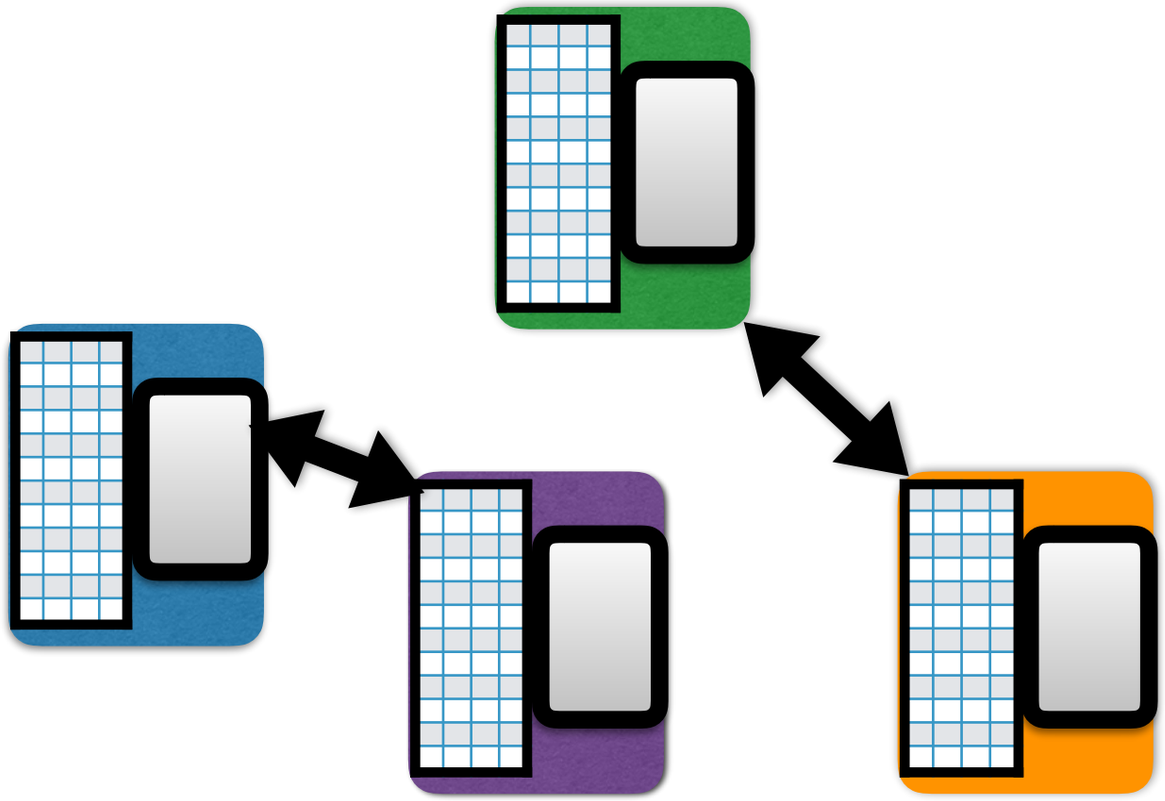
Nodes pick a partner to sync with

Gossip & Epidemic Alg



By controlling rate and selection one can construct a Epidemic Alg that ensure all nodes will be synced

Gossip & Epidemic Alg



- http://issg.cs.duke.edu/epidemic/epidemic.pdf
- https://www.usenix.org/legacy/publications/library/proceedings/nsdi04/tech/levisTrickle/

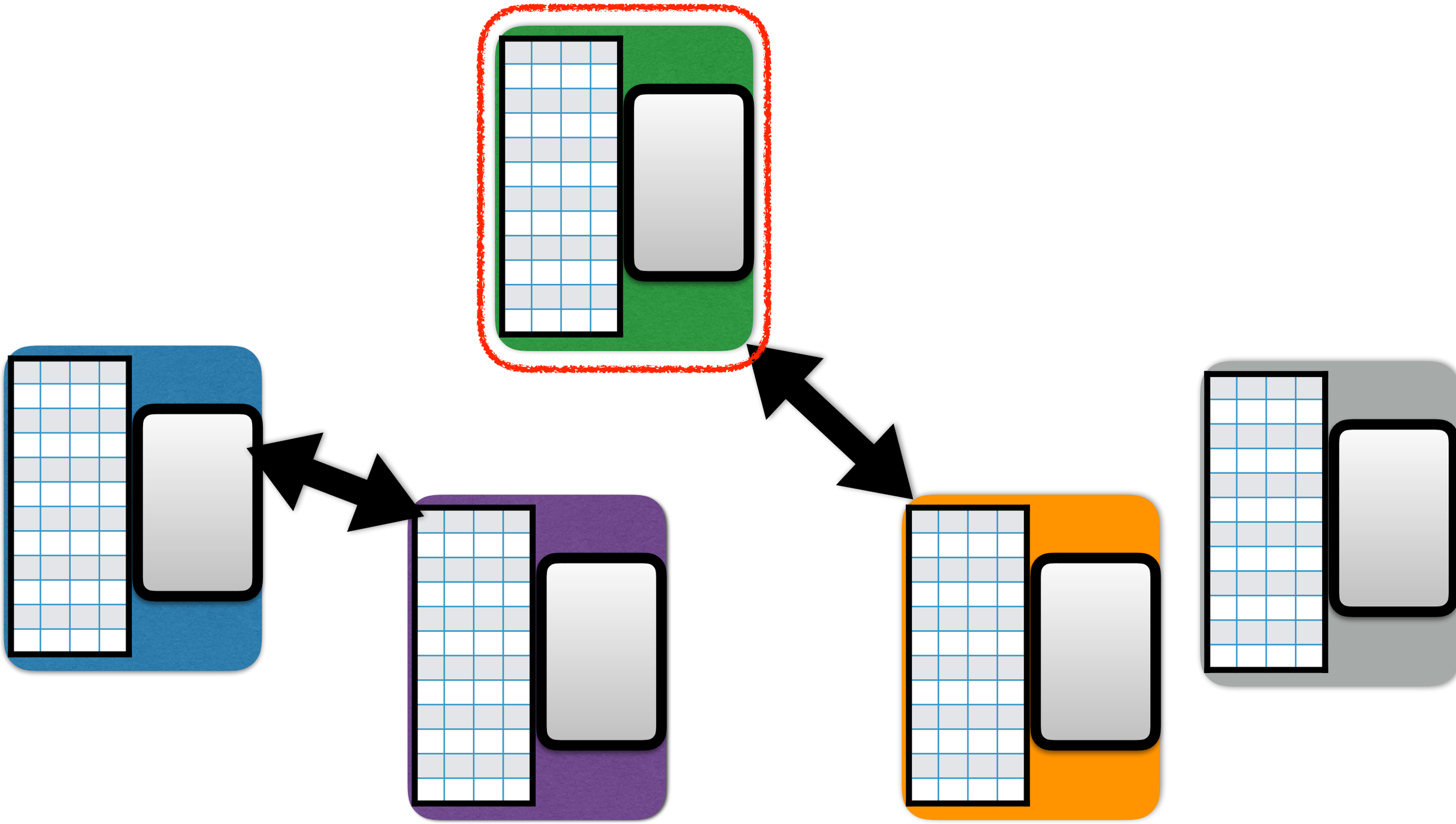
How can we be sure a meeting is happening?

How can we be sure a meeting is happening?

We need to make entries, up to some point in the log, permanent / stable

But if we can't be sure to hear from all nodes prior to meeting what do we do?

Primary



Primary

- The Primary is like any other node but
  - it assigns Commit Sequence Numbers to WriteID's <CSN,TS,NodeID>
  - tentative WriteIDs that don't have a CSN are always considered to be greater than committed WriteIDs — uncommitted updates come after committed updates
- CSN are synced
- CSN define total order

There is other aspects to BAYOU

- Checkpoints for redo
- CSN's allow discard of log
- How to add and delete servers

Eventual Consistency

- Is Eventual Consistency useful?
  - YES — iPhone sync, Dropbox, Dynamo, Riak, Cassandra,… — fast writes are nice
  - Conflicts are real
  - There is a lot of complexity to BAYOU not sure it is all worth it …
  - But some good ideas to remember: update functions, log as truth not DB, Logical clocks for causal consistency