Distributed Snapshots

CS 240: Computing Systems and Concurrency
Lecture 5

Marco Canini

Today

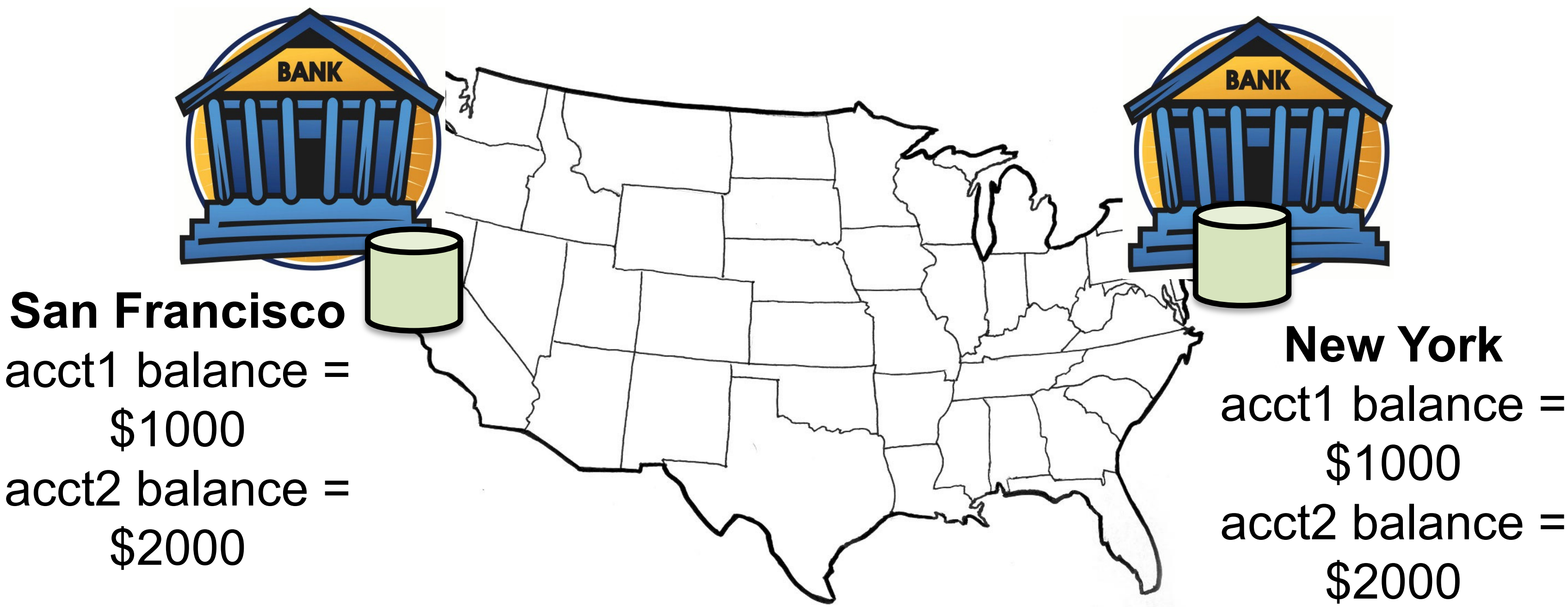
1. **Distributed Snapshots and Global State**

2. Chandy-Lamport algorithm

3. Reasoning about C-L: Consistent Cuts

Distributed Snapshots

- What is the state of a distributed system?



System model

- *N* **processes** in the system with no process failures
  - Each process has some **state** it keeps track of

- There are two first-in, first-out, unidirectional **channels** between every process pair P and Q
  - Call them **channel(P, Q)** and **channel(Q, P)**
  - All messages sent on channels arrive intact, unduplicated, in order
  - The channel has **state,** too: the set of messages inside

Aside: FIFO communication channel

"All messages sent on channels arrive intact, unduplicated, in order"

- Q: Arrive?
- Q: Intact?
- Q: Unduplicated?
- Q: In order?

- At-least-once retransmission
- Network layer checksums
- At-most-once deduplication
- Sender include sequence numbers, receiver only delivers in sequence order

- TCP provides all of these when processes don't fail

Global snapshot is global state

- Each distributed system has a number of processes running on a number of physical servers

- These processes communicate with each other via channels

- A ***global snapshot*** captures
  1. The **local states of each process** (*e.g.*, program variables), and

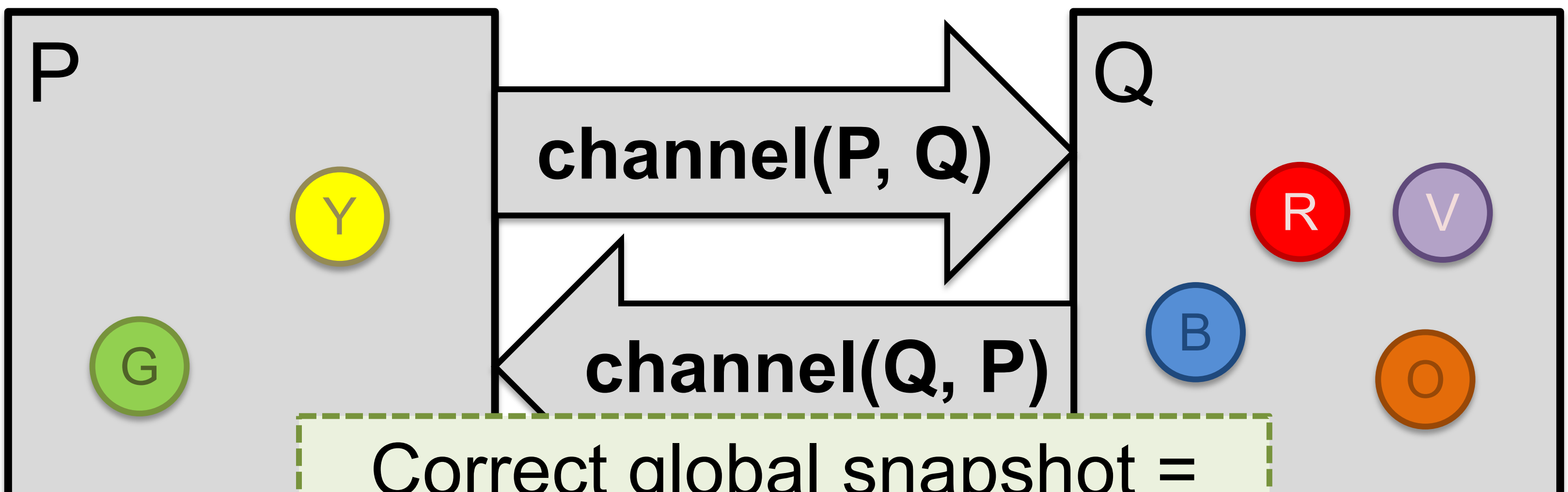
  2. The state of **each communication channel**

System model: Graphical example

- Let's represent process state as a set of colored ***tokens***

- Suppose there are two processes, **P** and **Q:**

Process P: **Process Q:**



Correct global snapshot =
Exactly one of each token

Why do we need snapshots?

- **Checkpointing:** Restart if the application fails
- **Collecting garbage:** Remove objects that don't have any references
- **Detecting deadlocks:** The snapshot can examine the current application state
  - **Process A** grabs **Lock 1**, **B** grabs **2**, **A** waits for **2**, **B** waits for **1... ... ...**
- **Other debugging:** A little easier to work with than printf…

Just synchronize local clocks?

- Each process **records state** at **some agreed-upon time**

- But **system clocks skew,** significantly with respect to CPU process' clock cycle
  - And we **wouldn't record messages** between processes

- Do we need synchronization?

- What did Lamport realize about ordering events?

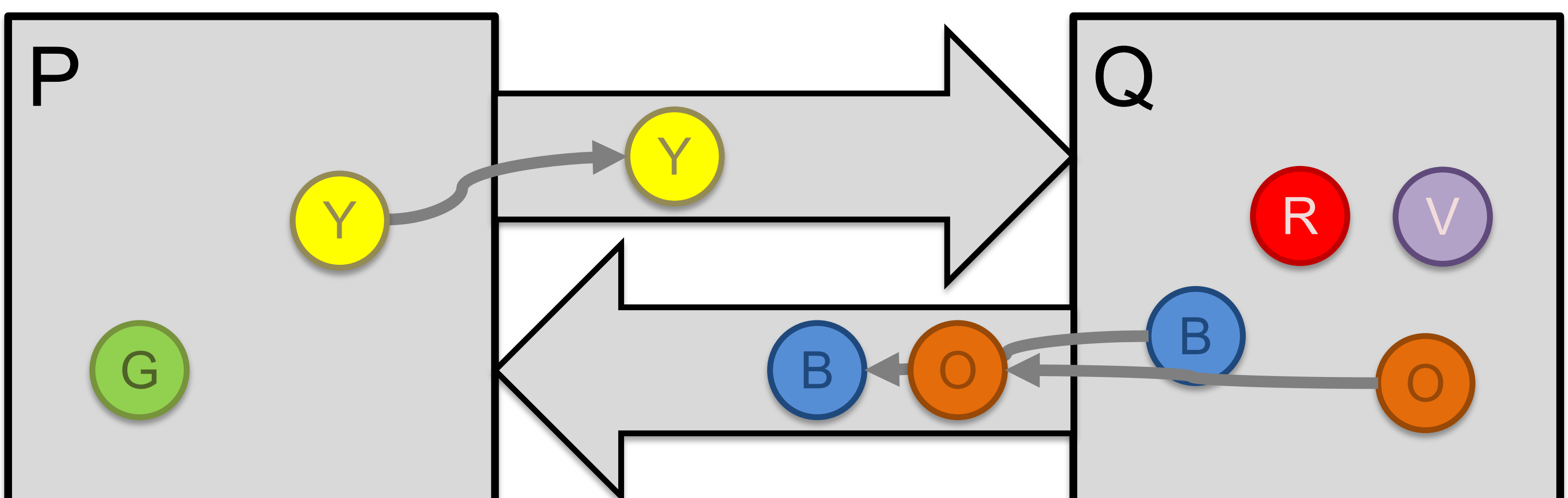
When is inconsistency possible?

- Suppose we take snapshots **only from a process perspective**
- Suppose snapshots happen **independently** at each process
- Let’s look at the implications...

Problem: Disappearing tokens

- P, Q put tokens into channels, **then** snapshot

This snapshot **misses** Y, B, and O tokens



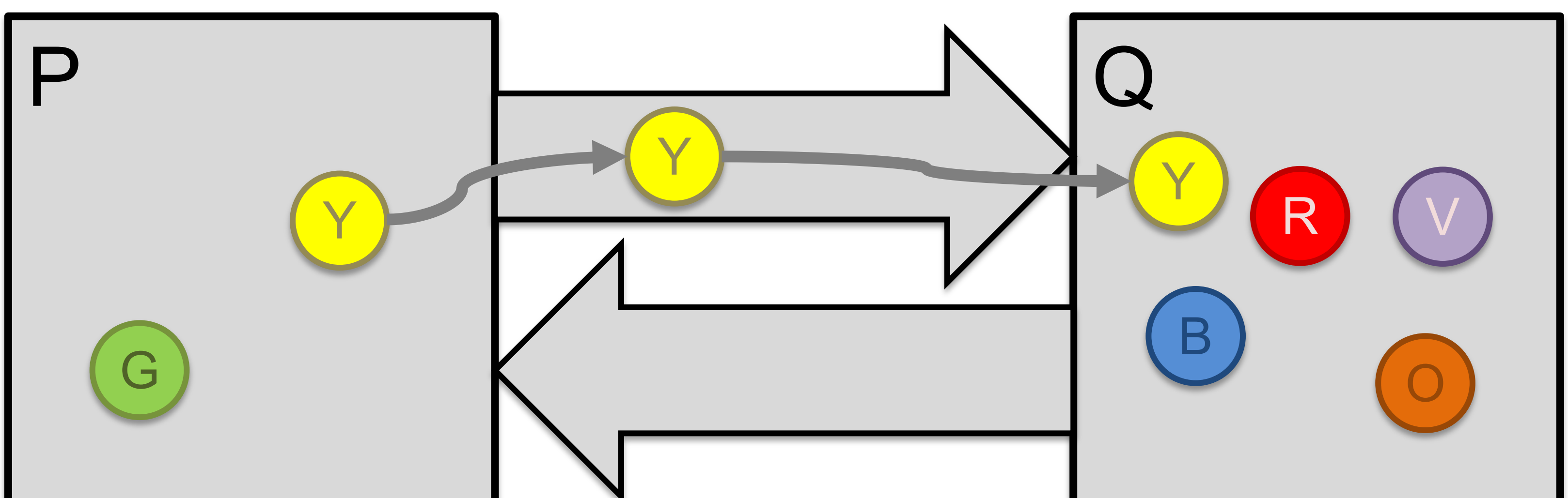
P = { G }

Q = { R, V }

Problem: Duplicated tokens

- P snapshots, **then** sends Y
- Q receives Y, **then** snapshots

This snapshot **duplicates** the Y token



P = { G, Y }

Q = { Y, R, V, B, O }

Idea: “Marker” messages

- What went wrong? We should have captured the state of the **channels** as well

- Let’s send a ***marker message*** ▲ to track this state
  - Not an application message, does not interfere with other application messages
  - Channels deliver marker and other messages FIFO

Today

1. Distributed Snapshots and Global State

2. **Chandy-Lamport algorithm**

3. Reasoning about C-L: Consistent Cuts

Chandy-Lamport algorithm: Overview

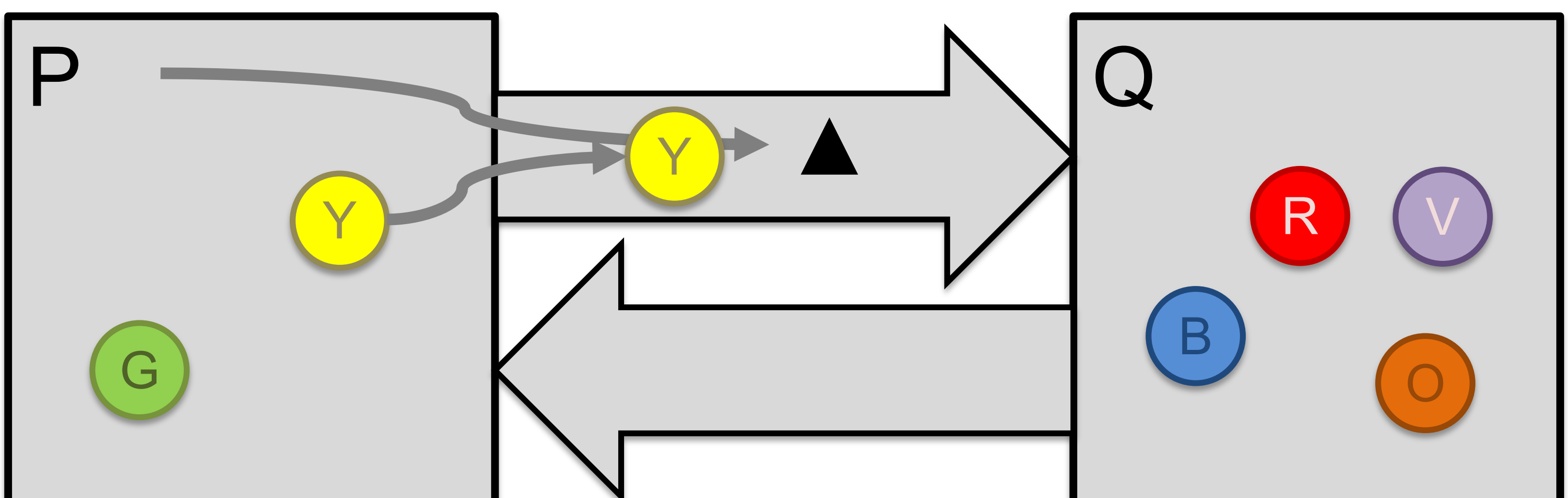
- We'll designate one node (say **P**) to **start** the snapshot
  - Without any steps in between, **P:**
    1. Records its local state ("snapshots")
    2. Sends a marker on each outbound channel

- Nodes remember **whether they have snapshotted**

- **On receiving a marker,** a **non-snapshotted** node performs steps (1) and (2) above

Chandy-Lamport: Sending process

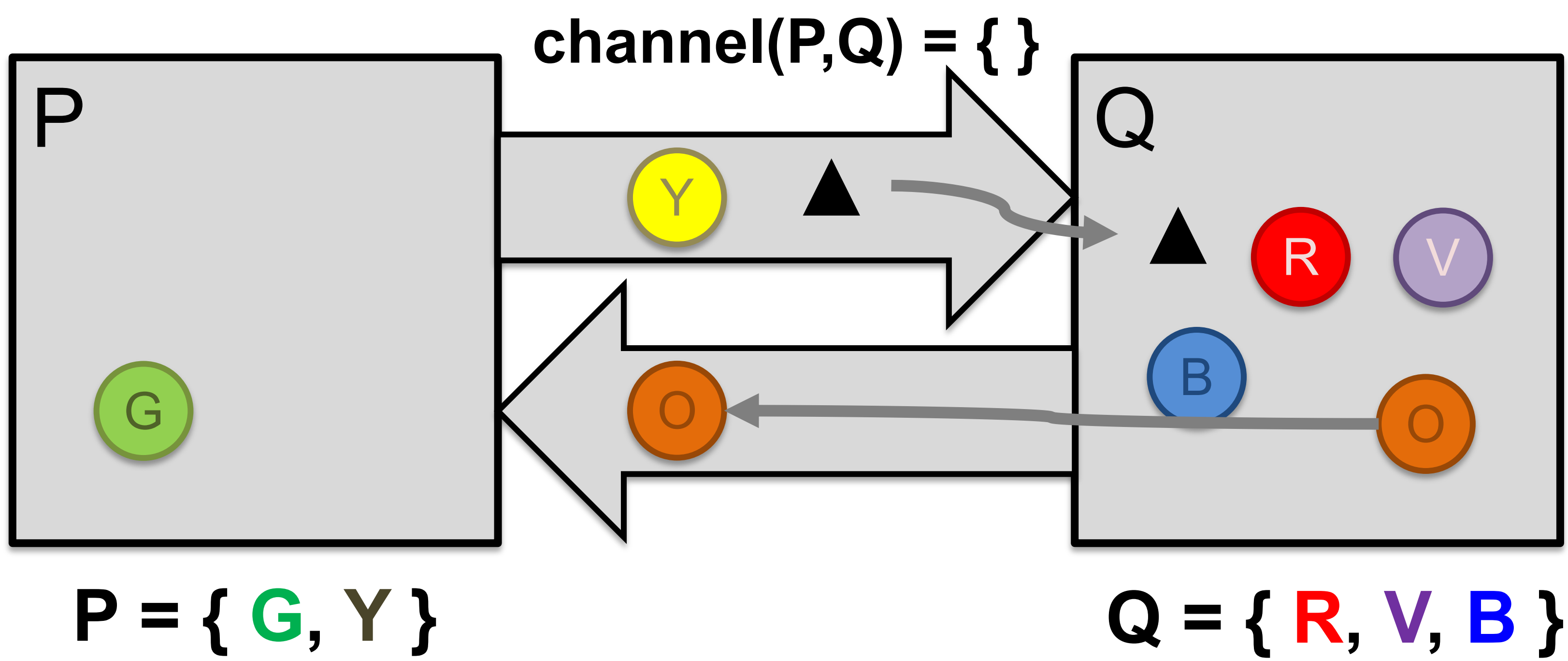
- P **snapshots and sends marker, then** sends Y
- **Send Rule:** Send marker on all outgoing channels
  - **Immediately after snapshot**
  - **Before** sending any **further messages**



snap: P = { G, Y }

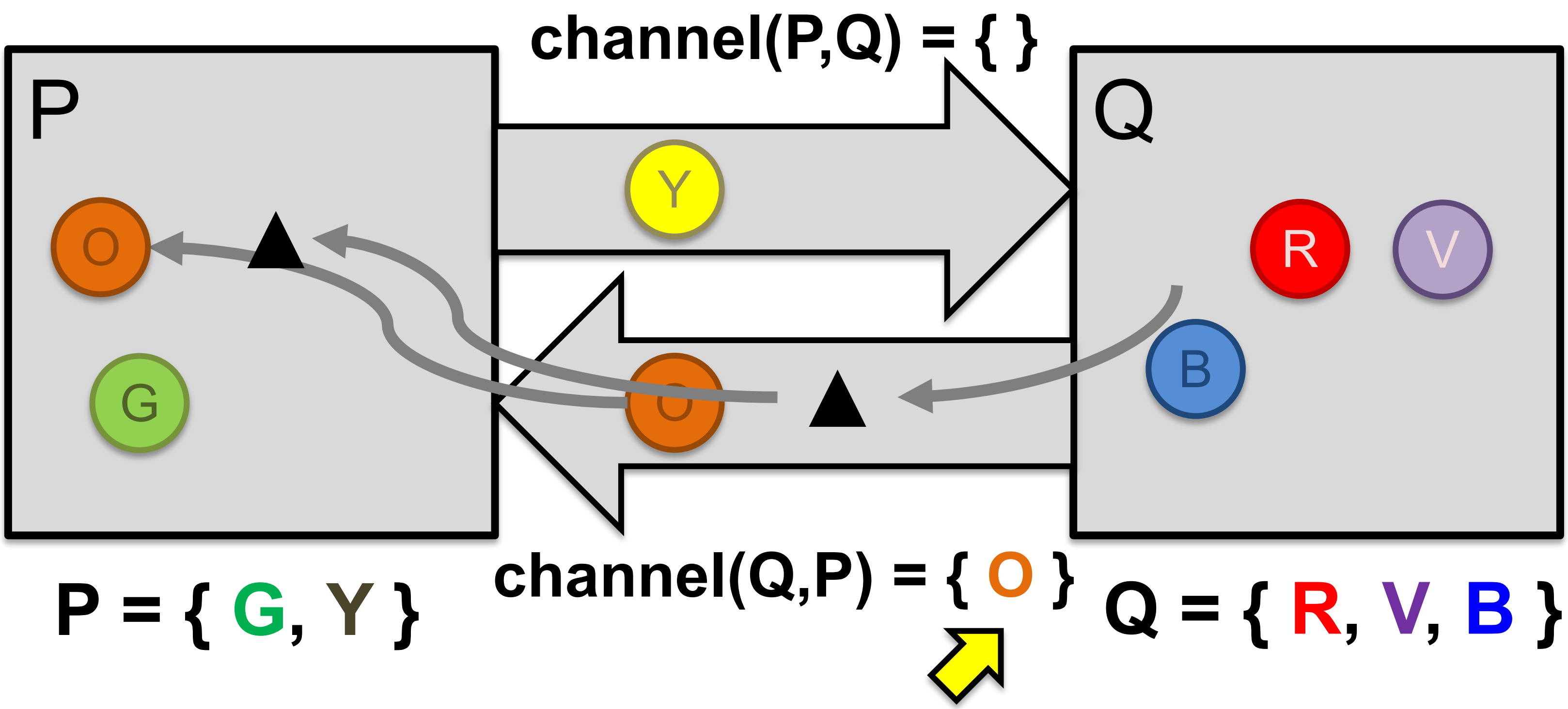
Chandy-Lamport: Receiving process (1/2)

- At the same time, Q sends orange token **O**
- Then, Q receives marker ▲
- **Receive Rule (if not yet snapshotted)**
  - On receiving marker on channel ***c*** record *c*'s state as **empty**



Chandy-Lamport: Receiving process (2/2)

- Q sends marker to P
- P receives orange token O, then marker ▲
- **Receive Rule (if already snapshotted):**
  - On receiving marker on ***c*** record *c*'s state: **all msgs from *c* since snapshot**



Terminating a snapshot

- **Distributed algorithm:** No single process decides when it terminates
- Eventually, all processes have received a marker (and recorded their own state)
- All processes have received a marker on all the N–1 incoming channels (and recorded their states)
- Later, a central server can **gather the local states** to build a global snapshot

C-L Global Snapshot Algorithm (1/2)

- **First: Initiator P*i* records its own state**

- **for *j=1 to N* except *i***
  - P*i* sends out a **Marker** message on outgoing channel $C_{i,j}$
  - (*N-1*) channels
- Starts recording the incoming messages on each of the incoming channels at P*i*: $C_{j,i}$ (for *j=1 to N* except *i*)

CL Global Snapshot Algorithm (2/2)

Whenever a process P*i* receives a Marker message on an incoming channel $C_{k,i}$

- **if** (this is the first Marker P*i* is seeing)
  - P*i* records its own state first
  - Marks the state of channel $C_{k,i}$ as "empty"
  - for *j=1 to N* except *i*
    - P*i* sends out a Marker message on outgoing channel $C_{i,j}$
  - Starts recording the incoming messages on each of the incoming channels at P*i*: $C_{j,i}$ (for *j=1 to N* except *i* and *k*)
- **else** /* already seen a Marker message */
  - Mark the state of channel $C_{k,i}$ as all the messages that have arrived on it since recording was turned on for $C_{k,i}$

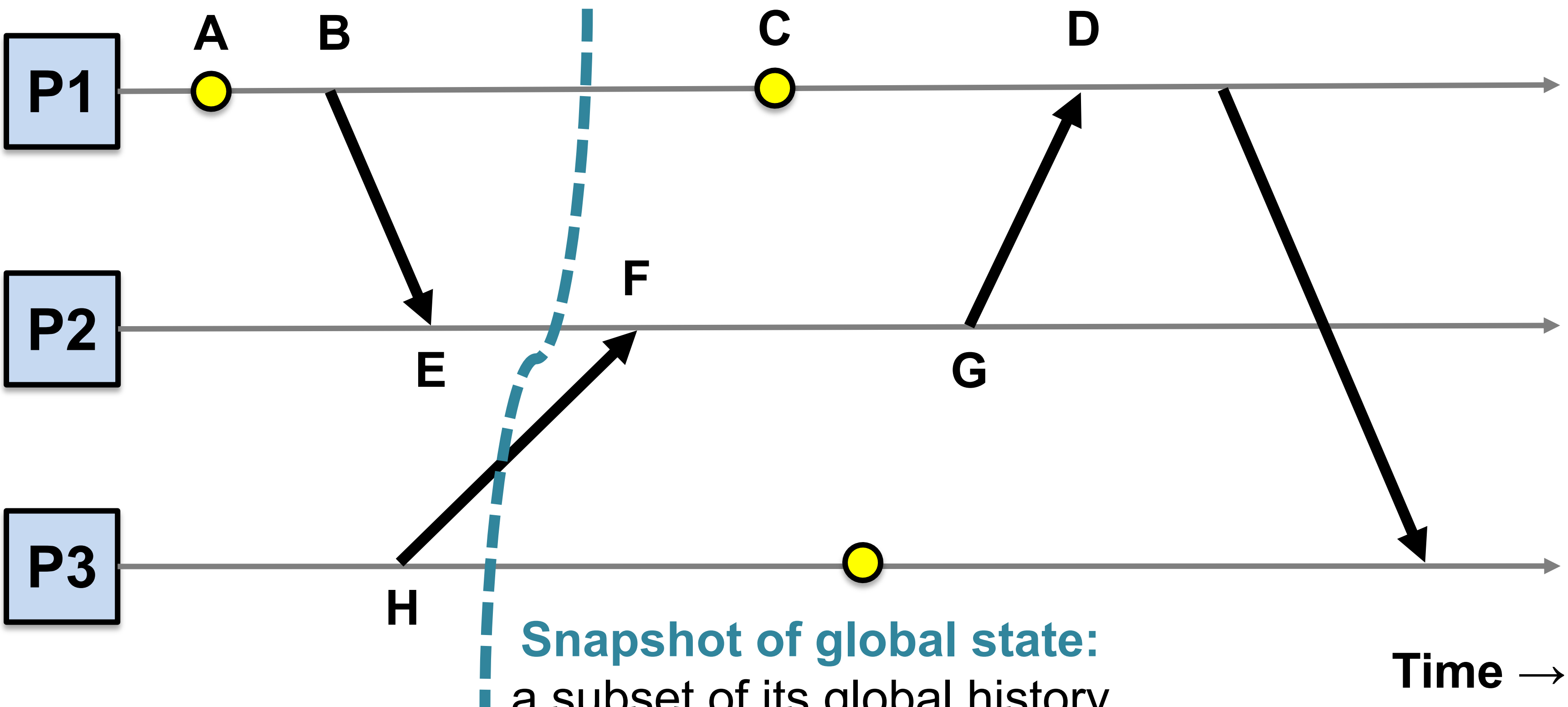
Today

1. Distributed Snapshots and Global State

2. Chandy-Lamport algorithm

3. **Reasoning about C-L: Consistent Cuts**

Global state as cut of system's execution



Cut = { The last **event** of each **process**, and **message** of each **channel** that is in the cut }

Global states and cuts

- ***Global state*** is a *n*-tuple of local states (one per process **and** channel)

- A **cut** is a subset of the global history that contains an initial prefix of each local state
  - Therefore every cut is a natural global state
  - Intuitively, a cut **partitions** the space time diagram along the time axis

- ***Cut*** = { The last **event** of each **process,** and **message** of each **channel** that is in the cut }

Consistent versus inconsistent cuts

- A **consistent cut** is a cut that **respects causality of events**

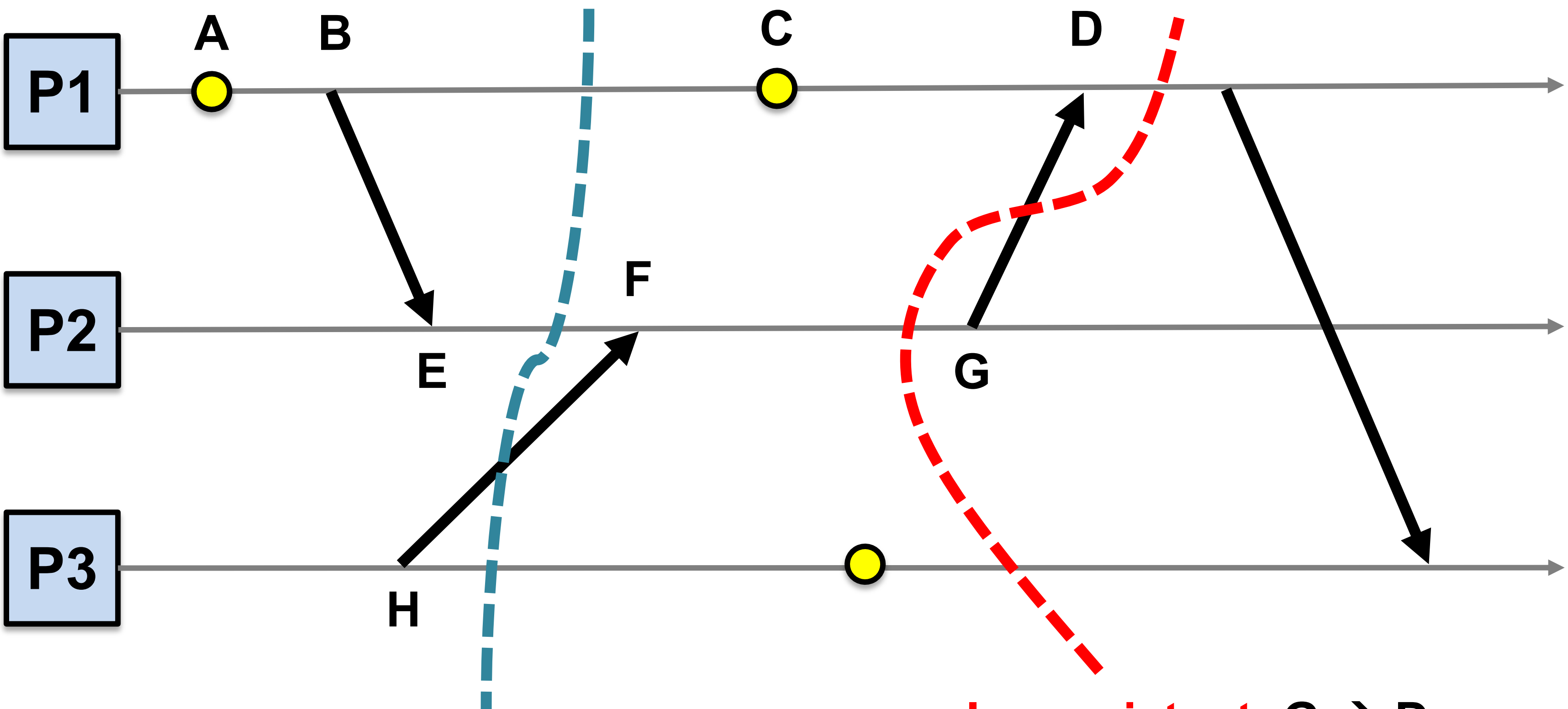
- A cut **C** is ***consistent*** when:

  – For each pair of events **x** and **y**, if:

    1. **y** is in the cut, and
    2. **x → y,**

  – then, event **x** is also **in the cut**

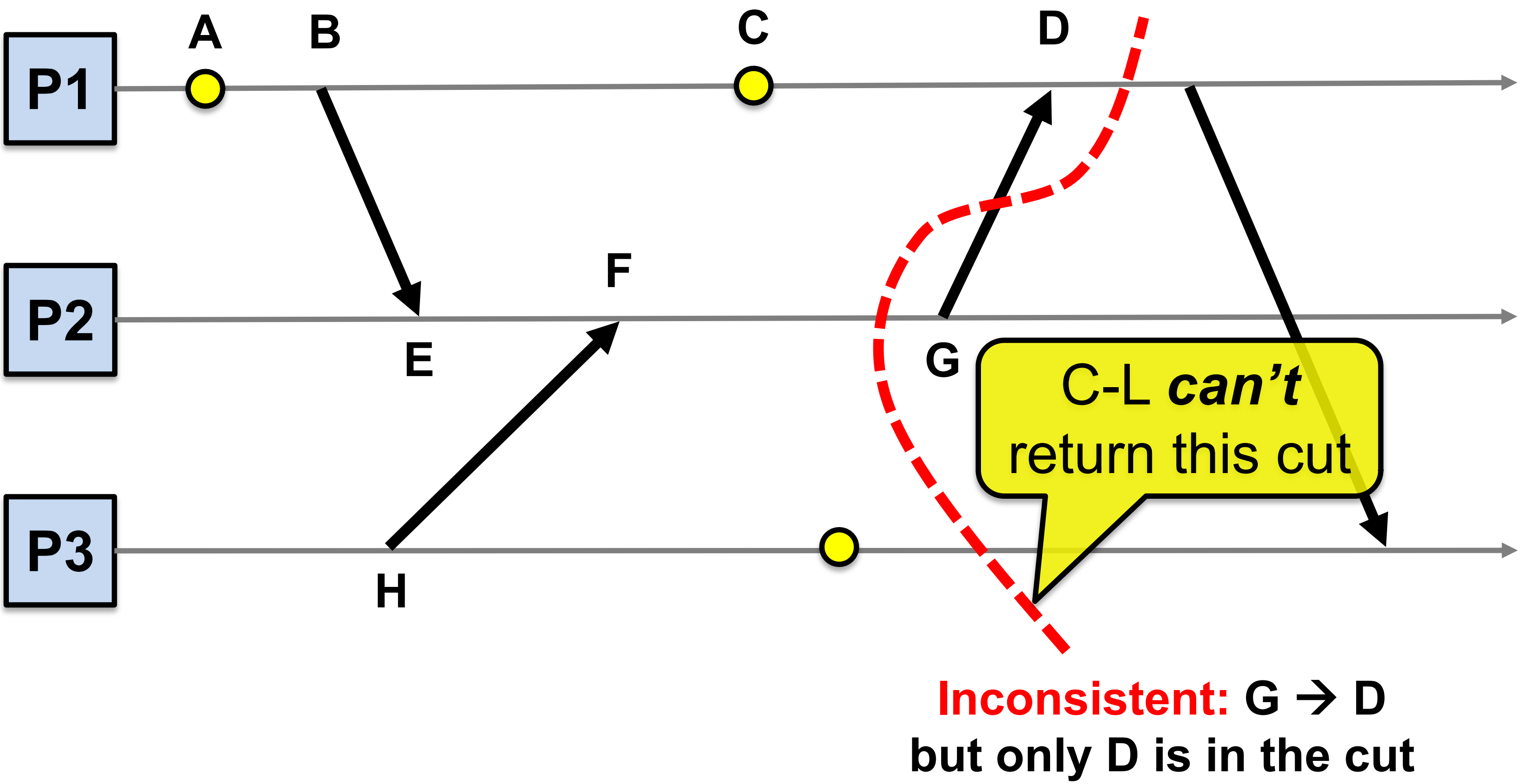
Consistent versus inconsistent cuts



Consistent: H → F
and H in the cut

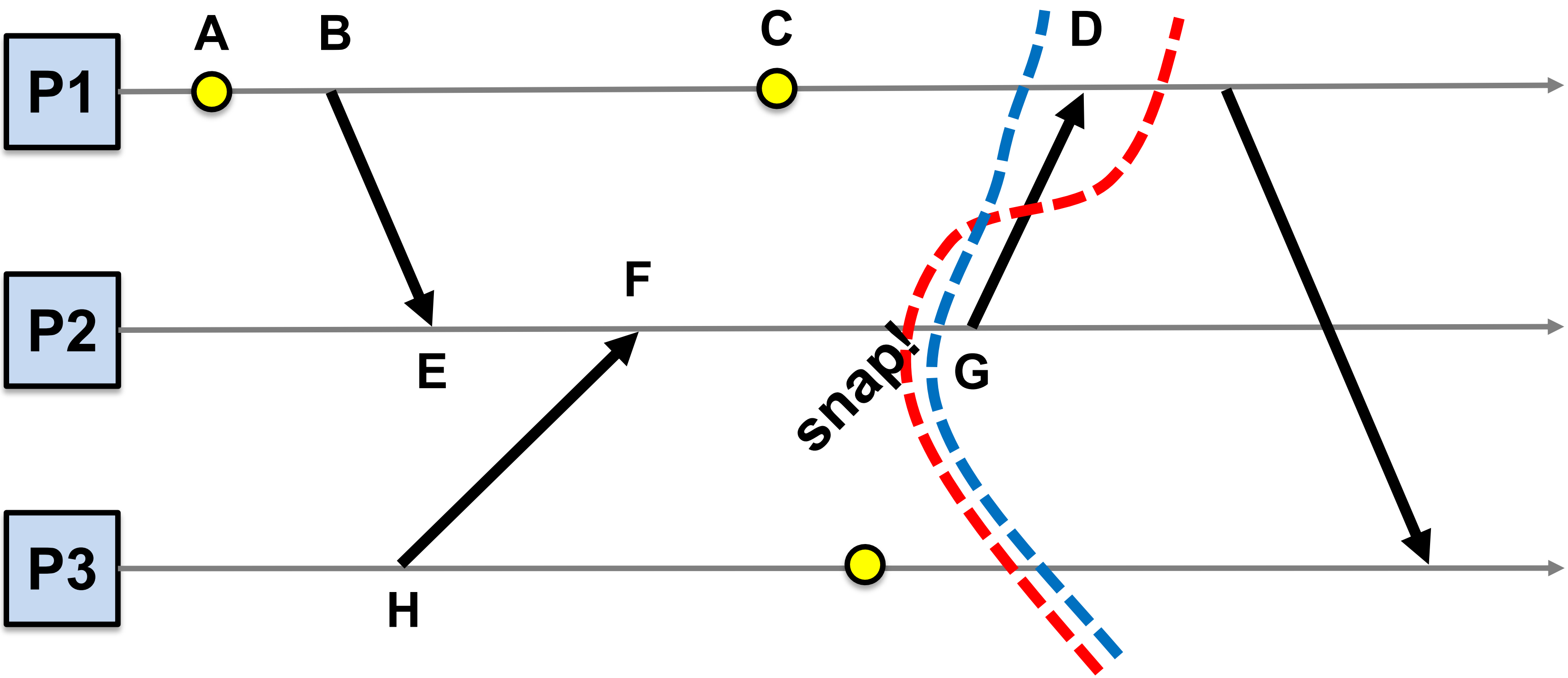
Inconsistent: G → D
but only D is in the cut

C-L returns a consistent cut



C-L ensures that if **D** is in the cut, then **G** is in the cut

C-L can't return this inconsistent cut



Take-away points

- Global State
  - A global snapshot captures
    - The local states of each process (e.g., program variables), and
    - The state of each communication channel

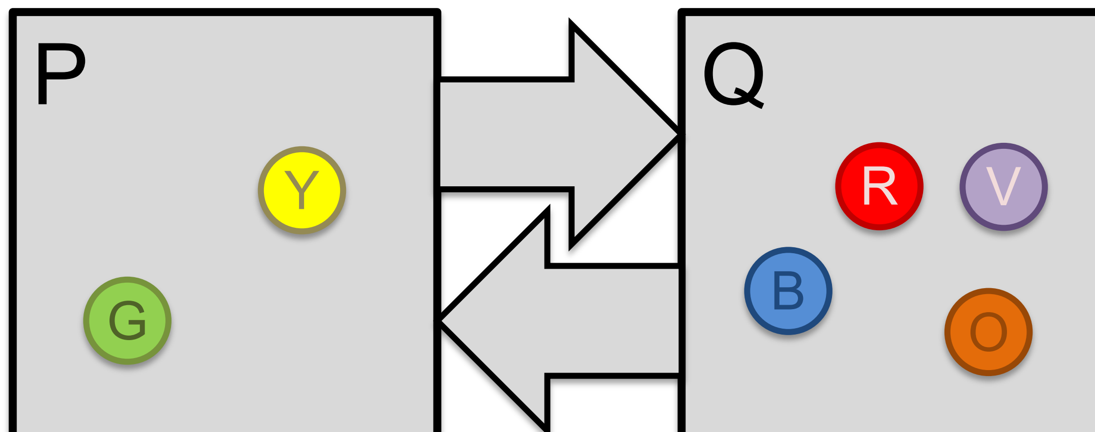
- Distributed Global Snapshots
  - FIFO Channels: we can realize them and build on guarantees
  - Chandy-Lamport algorithm: use marker messages to coordinate
  - Chandy-Lamport provides a consistent cut

Chandy-Lamport Puzzle #1

Is this snapshot possible? And if so, how?

P	= { G }
chan(P, Q)	= { Y }
Q	= { R, V }
chan(Q, P)	= { B, O }

Either P or Q starts CL from the current state



Chandy-Lamport Puzzle #2

Is this snapshot possible? And if so, how?

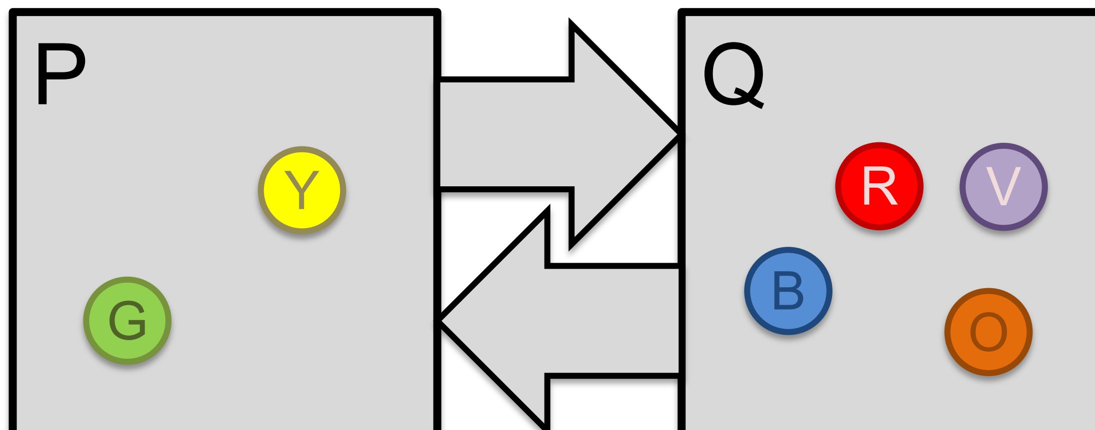
P = { G, Y, R, V, B, O }

chan(P, Q) = { }

Q = { }

chan(Q, P) = { }

Either P or Q starts CL from the current state



Chandy-Lamport Puzzle #3

Is this snapshot possible? And if so, how?

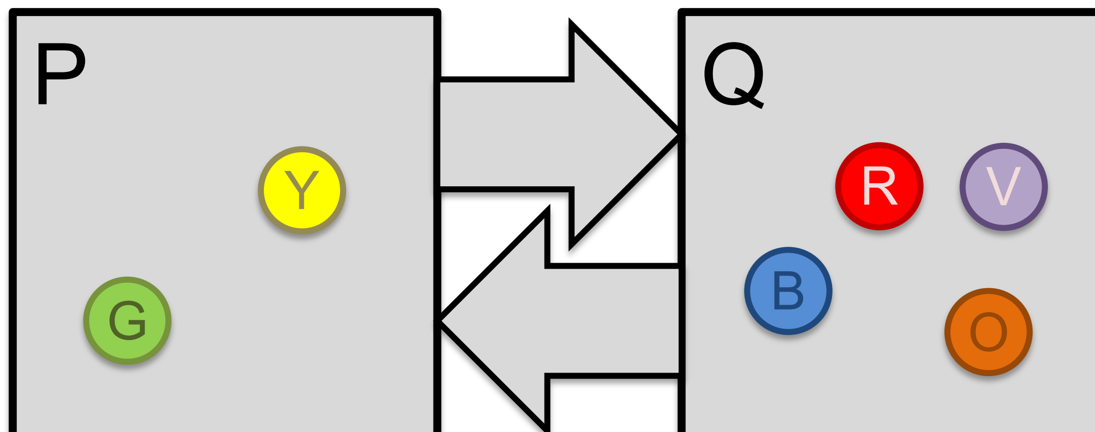
P = { }

chan(P, Q) = { }

Q = { }

chan(Q, P) = {G, Y, R, V, B, O }

Either P or Q starts CL from the current state

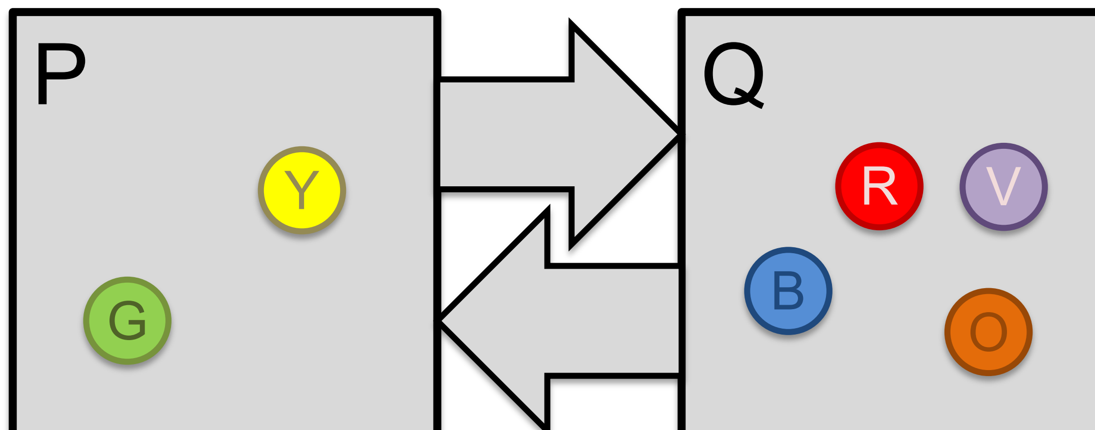


Chandy-Lamport Puzzle #4

Is this snapshot possible? And if so, how?

P = { G, Y }
chan(P, Q) = { R }
Q = { B, O }
chan(Q, P) = { V }

Either P or Q starts CL from the current state

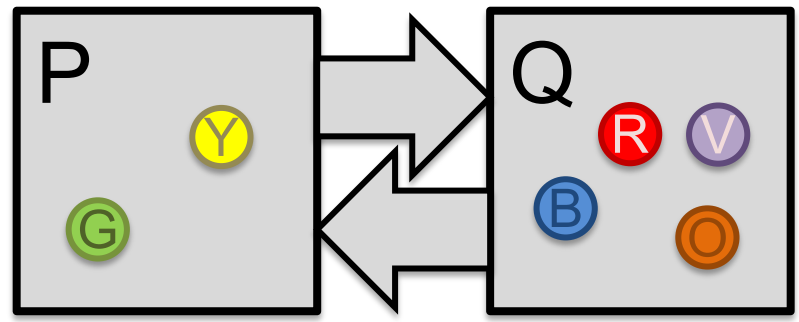


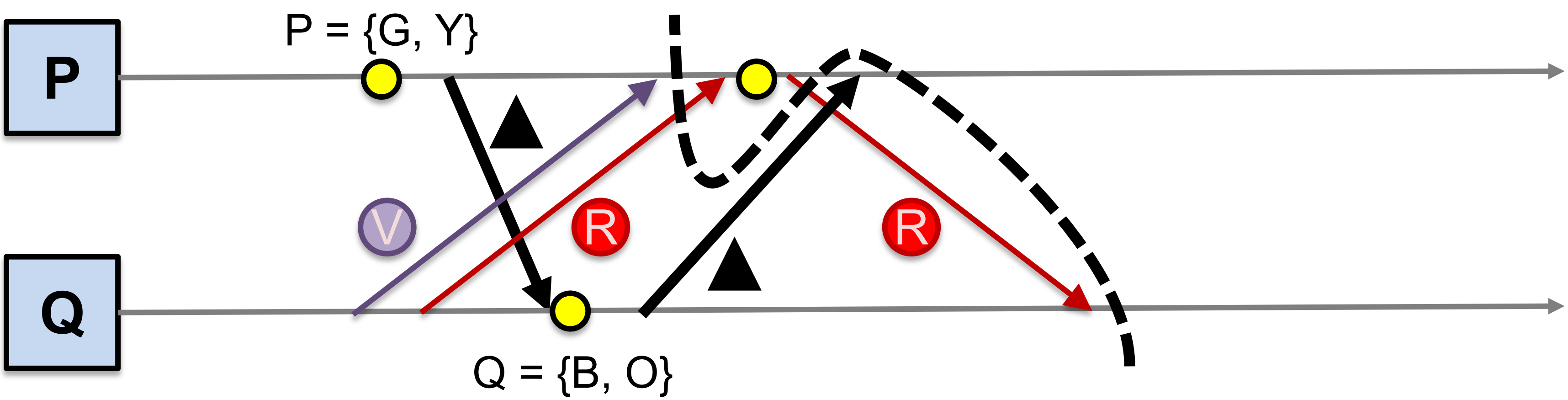
Puzzle #4: How are you thinking?

Is this snapshot possible? And if so, how?

P = { G, Y }
chan(P, Q) = { R }
Q = { B, O }
chan(Q, P) = { V }

Either P or Q starts CL from the current state

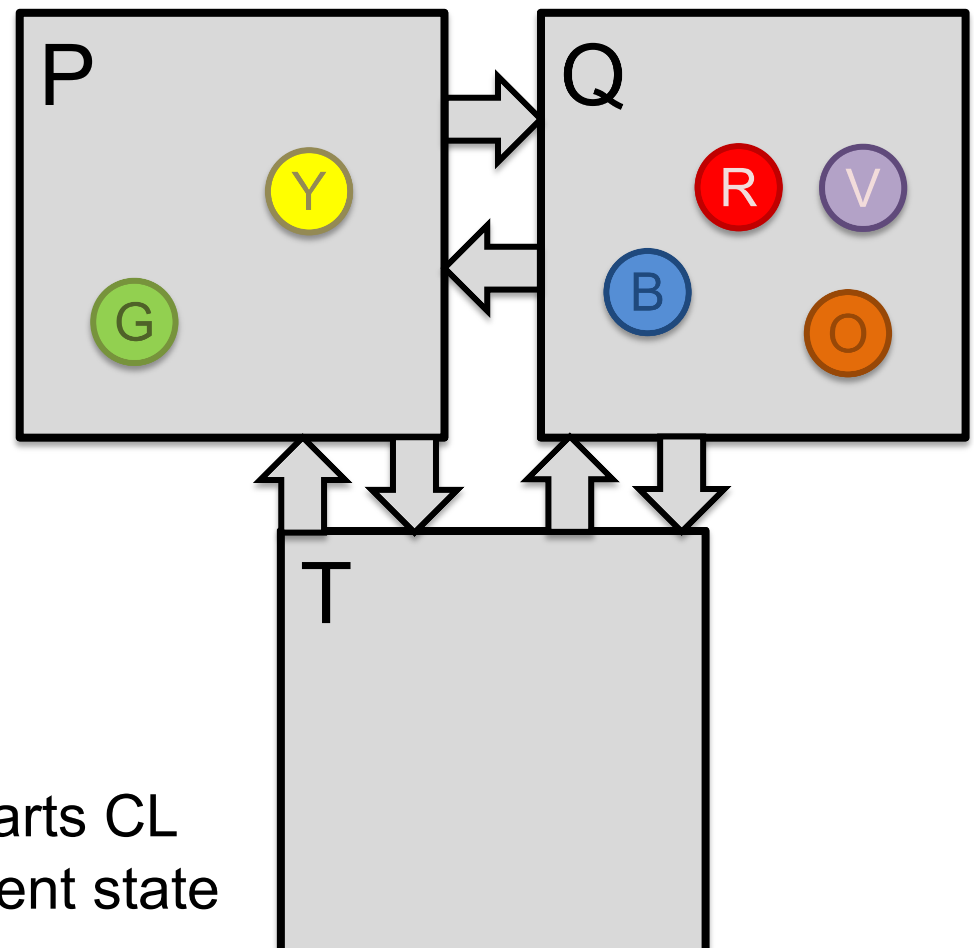




Chandy-Lamport Puzzle #5

Is this snapshot possible? And if so, how?

P = { G, Y }
chan(P, Q) = { }
chan(P, T) = { }
Q = { B, O }
chan(Q, P) = { V }
chan(Q, T) = { R }
T = { }
chan(T, P) = { }
chan(T, Q) = { }

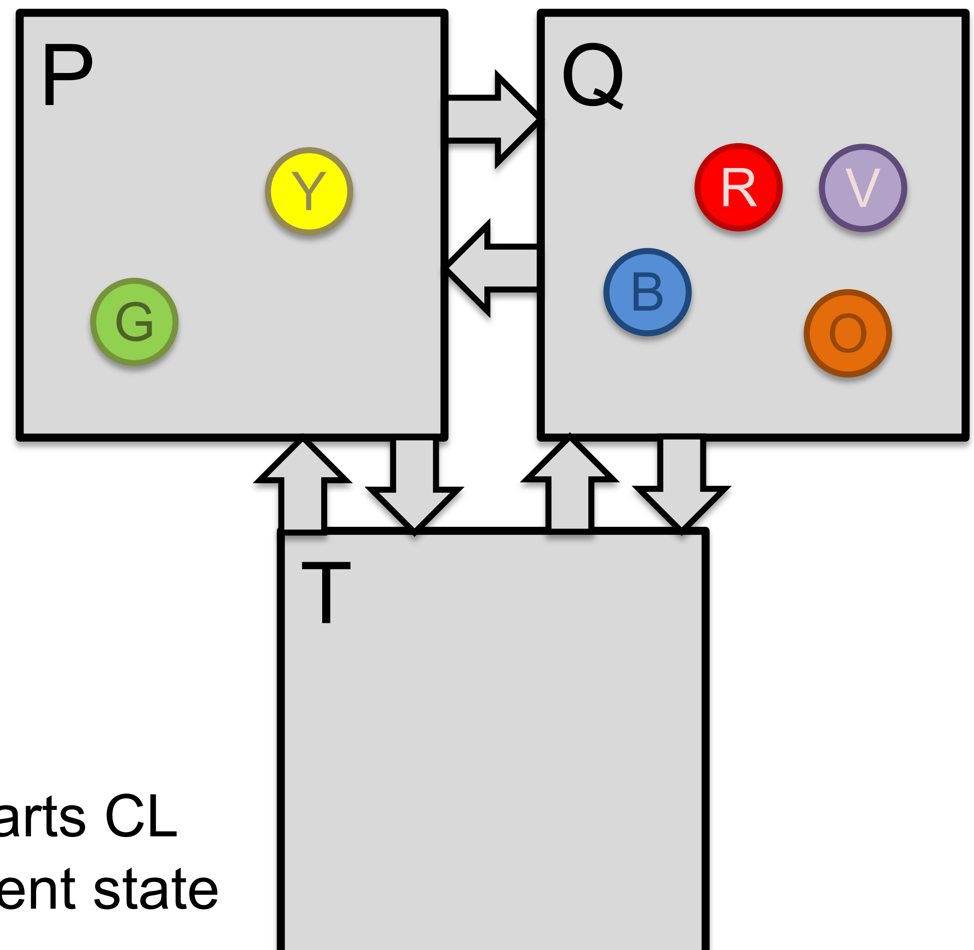


Assume P starts CL from the current state

Chandy-Lamport Puzzle #6

Is this snapshot possible? And if so, how?

P	= { G, Y }
chan(P, Q)	= { }
chan(P, T)	= { }
Q	= { B }
chan(Q, P)	= { V }
chan(Q, T)	= { R }
T	= { O }
chan(T, P)	= { }
chan(T, Q)	= { }



Assume P starts CL from the current state