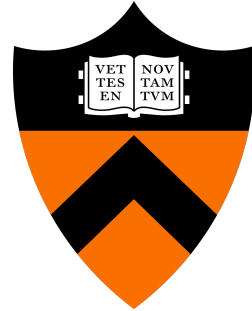


# Distributed Snapshots

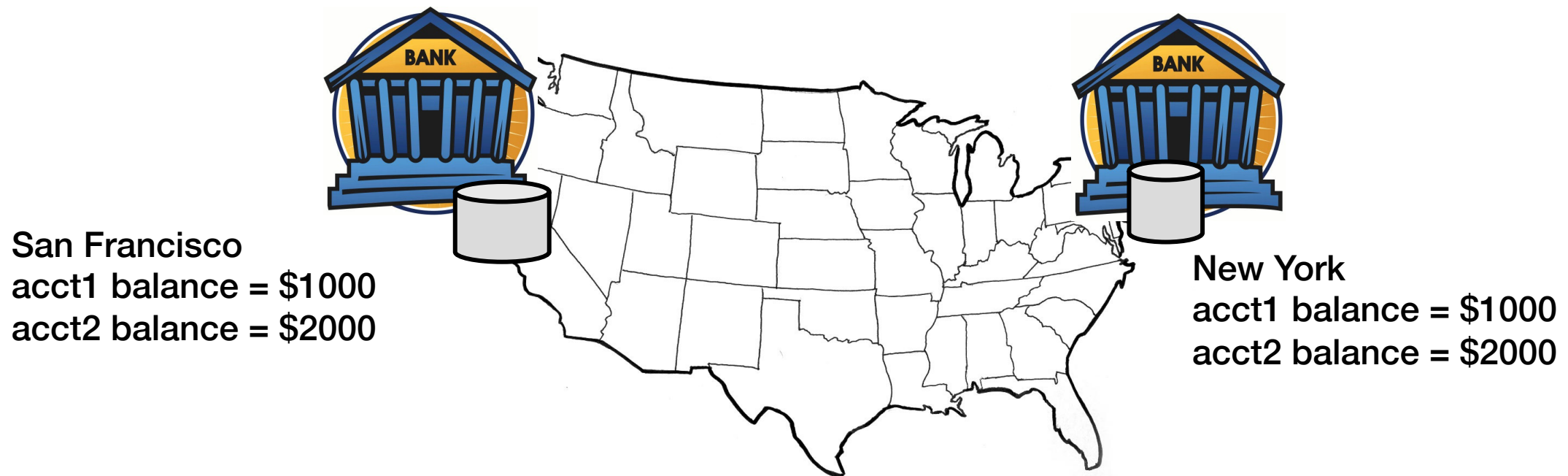


COS 418/518: Distributed Systems  
Lecture 7

Wyatt Lloyd, Mike Freedman

# 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)`
  
  - The channel has state, too: the set of messages inside
  
  - All messages sent on channels arrive intact, unduplicated, in order

# Aside: FIFO communication channel

- “All messages sent on channels arrive intact, unduplicated, in order”
- Q: Arrive?
- Q: Intact?
- Q: Unduplicated?
- Q: In order?
- TCP provides all of these when processes don't fail

# Global snapshot is global state

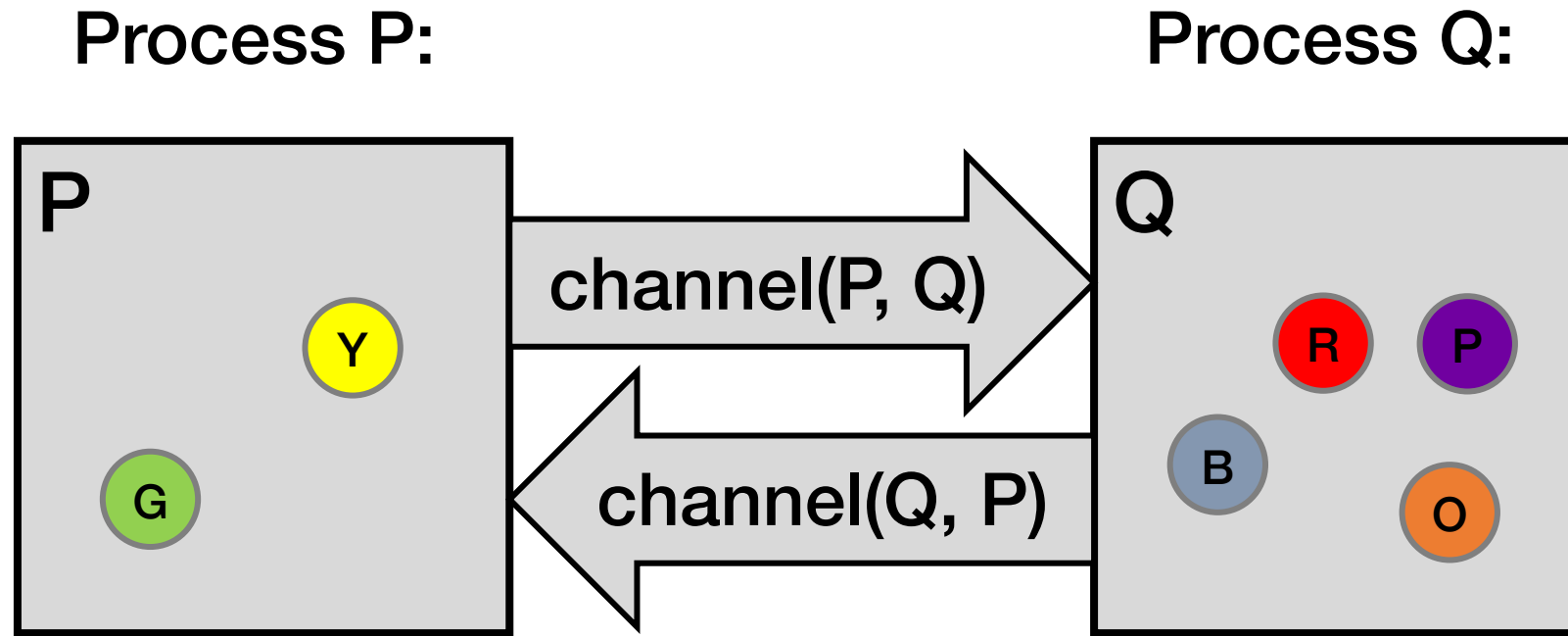
- Each distributed application has a number of processes running on a number of physical servers
- These processes communicate with each other via channels
- A **global snapshot** captures
  1. The local states of each process (e.g., program variables), and
  2. The state of each communication channel

# Why do we need snapshots?

- Checkpointing: Restart if the application fails
- Collecting garbage: Remove objects that aren't referenced
- 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... ..

# System model: Graphical example

- Let's represent process state as a set of colored tokens
- Suppose there are two processes, P and Q:



Correct global snapshot = Exactly one of each token

# When is inconsistency possible?

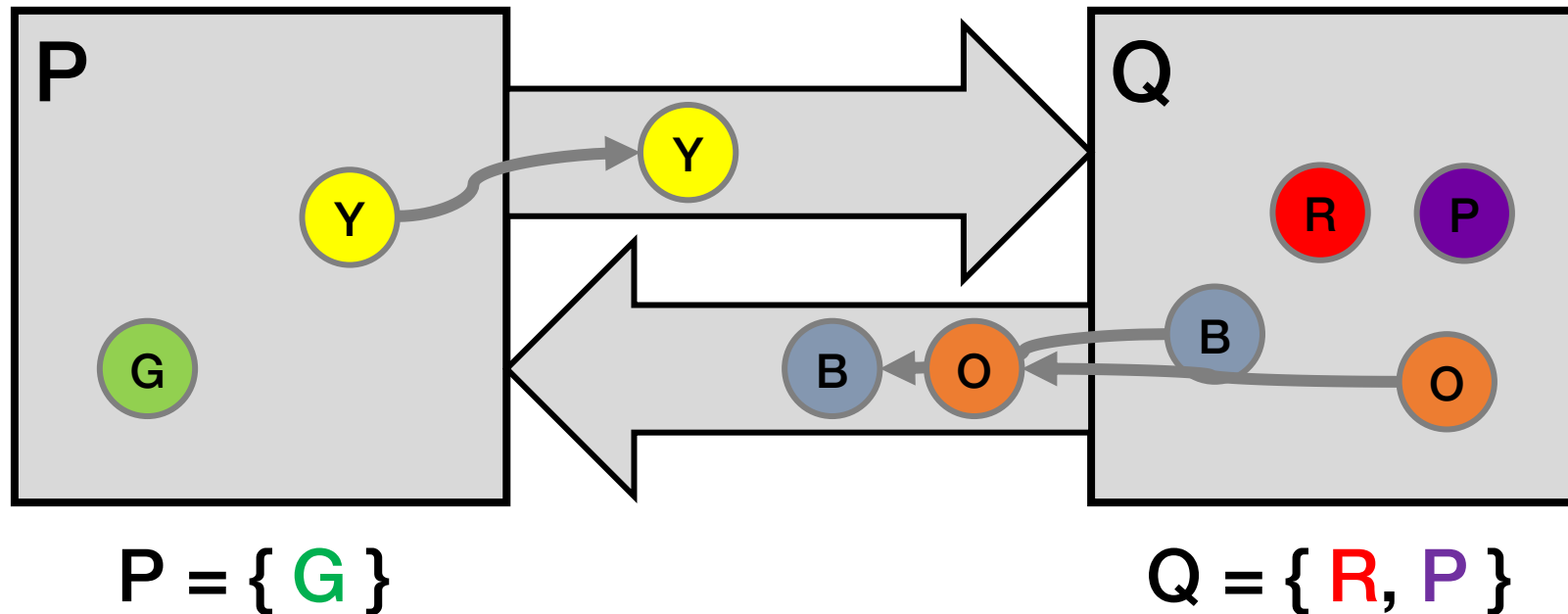
- Suppose we take snapshots only from a process perspective
- Suppose snapshots happen independently at each process
- Let's look at the implications...



# Problem: Disappearing tokens

- P, Q put tokens into channels, then snapshot

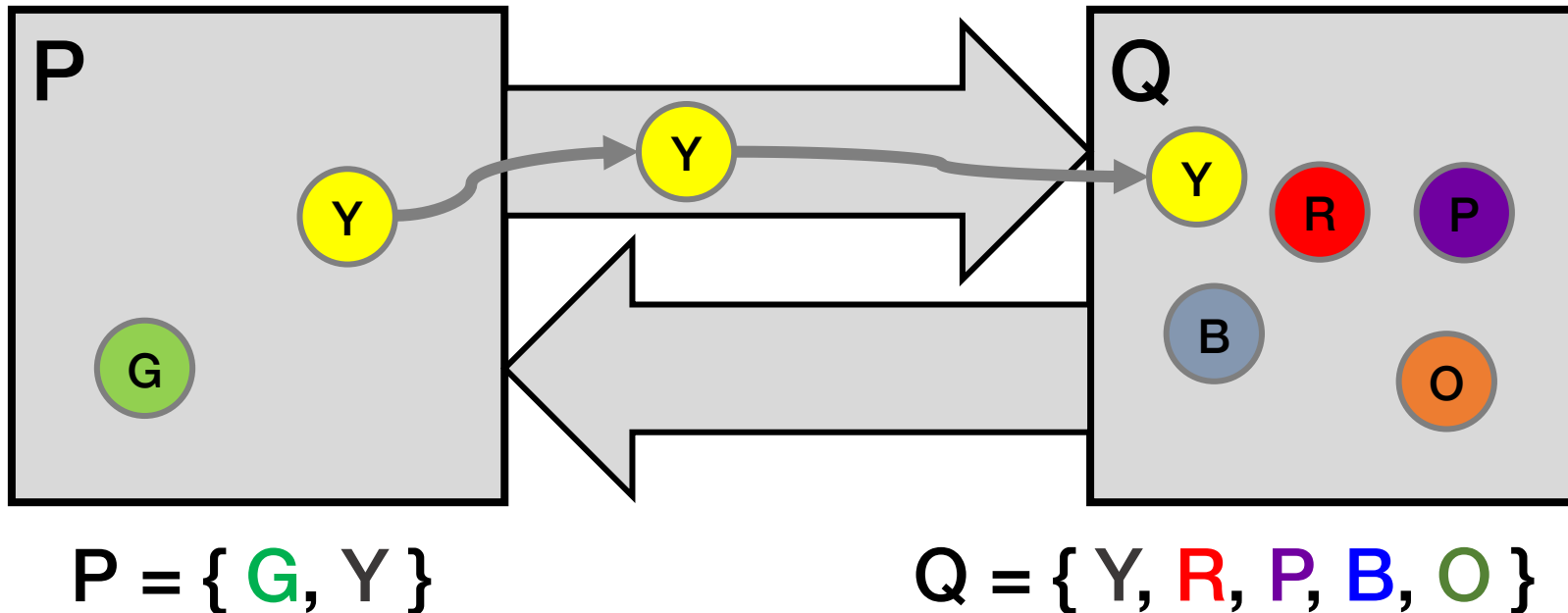
This snapshot **misses** Y, B, and O tokens



# Problem: Duplicated tokens

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

This snapshot **duplicates** the Y token



# Idea: “Marker” messages

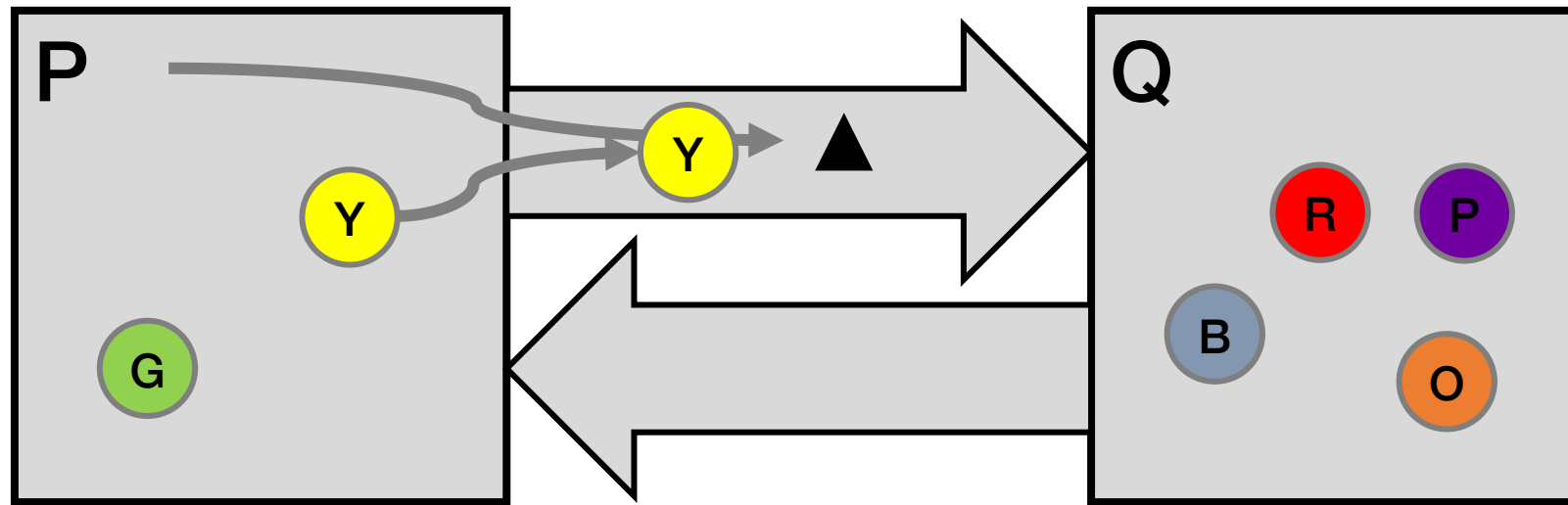
- What went wrong?
  - We should have captured the state of the **channels** as well
    - Avoids missing tokens
  - We should have coordinated snapshots at different processes
    - Avoids duplicate tokens
- Let’s send a **marker message** ▲
  - Used to capture channel state and coordinate different processes
  - Distinct from other messages
  - Channels deliver marker and other messages FIFO

# 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

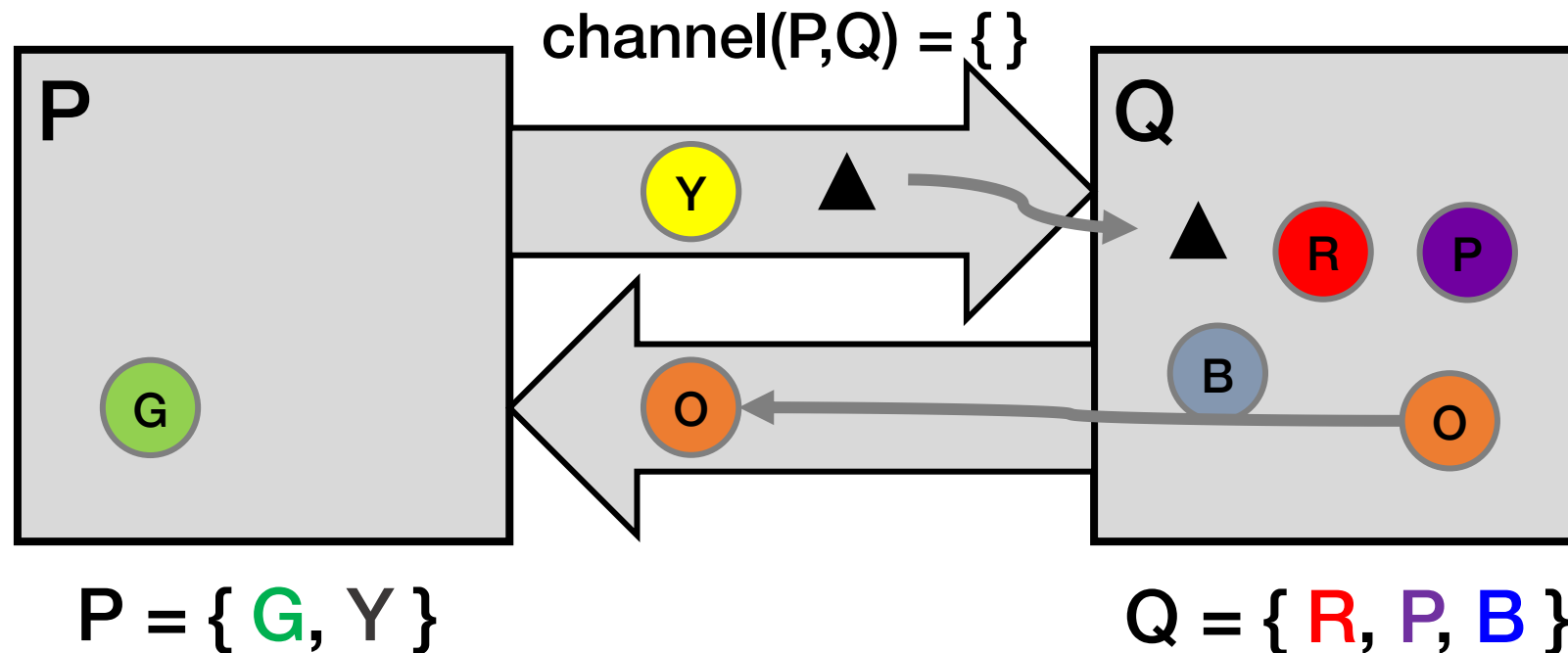
- P snapshots and sends marker, then sends Y
- **Send Rule:** Send marker on all outgoing channels
  - Immediately after snapshot
  - Before sending any further messages



snap: P = { G, Y }

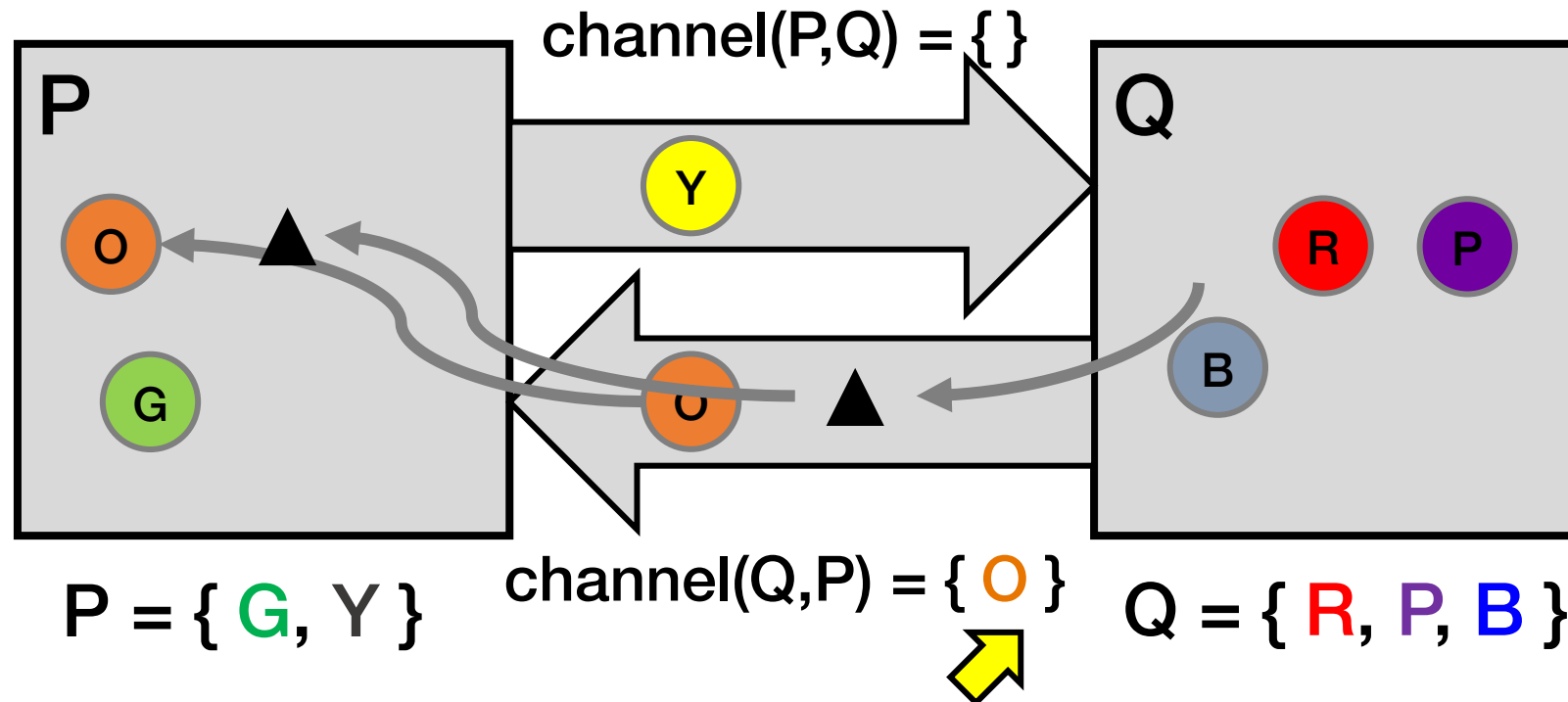
# Chandy-Lamport: Receiving process (1/2)

- 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  $\blacktriangle$
- **Receive Rule (if already snapshotted):**
  - On receiving marker on  $c$  record  $c$ 's state: all msgs from  $c$  since snapshot



# Chandy-Lamport Algorithm: Details

- 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
    3. Starts recording messages on each inbound channel
- Nodes remember whether they have snapshotted
- On receiving a marker on channel c
  - If ( $\neg$ snapshotted) then performs steps 1 – 3
  - 4. Stop recording messages on channel c



# Chandy-Lamport Puzzle #1

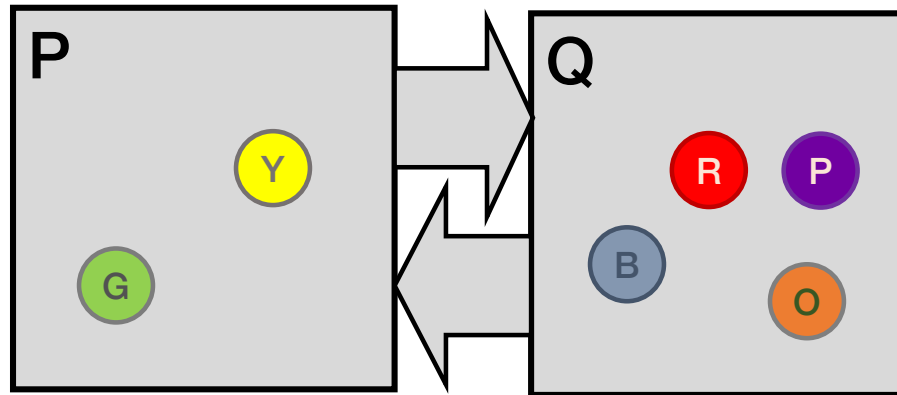
Is this snapshot possible? And if so, how?

P = { G },

chan(P, Q) = { Y },

Q = { R, P },

chan(Q, P) = { B, O },



# Chandy-Lamport Puzzle #2

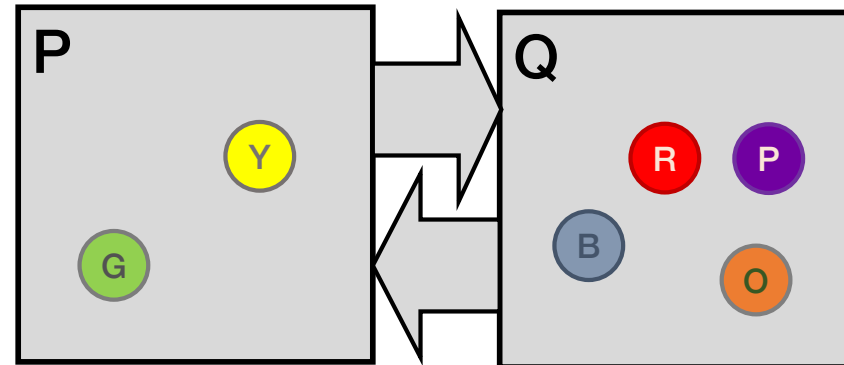
Is this snapshot possible? And if so, how?

$P = \{ G, Y, R, P, B, O \}$

$\text{chan}(P, Q) = \{ \}$

$Q = \{ \}$

$\text{chan}(Q, P) = \{ \}$



# Chandy-Lamport Puzzle #3

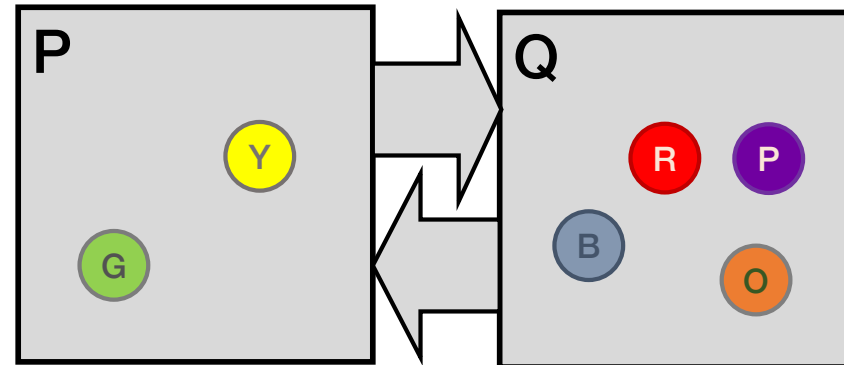
Is this snapshot possible? And if so, how?

P = { }

chan(P, Q) = { }

Q = { }

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



# Chandy-Lamport Puzzle #4

Is this snapshot possible? And if so, how?

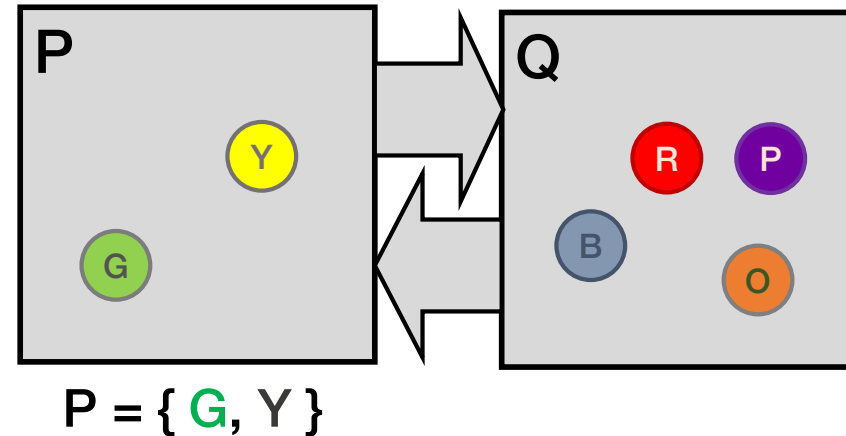
$P = \{ G, Y, \}$

$\text{chan}(P, Q) = \{ R \}$

$Q = \{ B, O \}$

$\text{chan}(Q, P) = \{ P \}$

- Is it possible (and how) if we add:
  - A process T and just a  $\text{chan}(T,P)$ ?
  - T,  $\text{chan}(T,P)$  and  $\text{chan}(T,Q)$ ?



# Chandy-Lamport Puzzle #5

Is this snapshot possible? And if so, how?

P = { G, Y, }

chan(P, Q) = { }

chan(P, T) = { }

Q = { B, O }

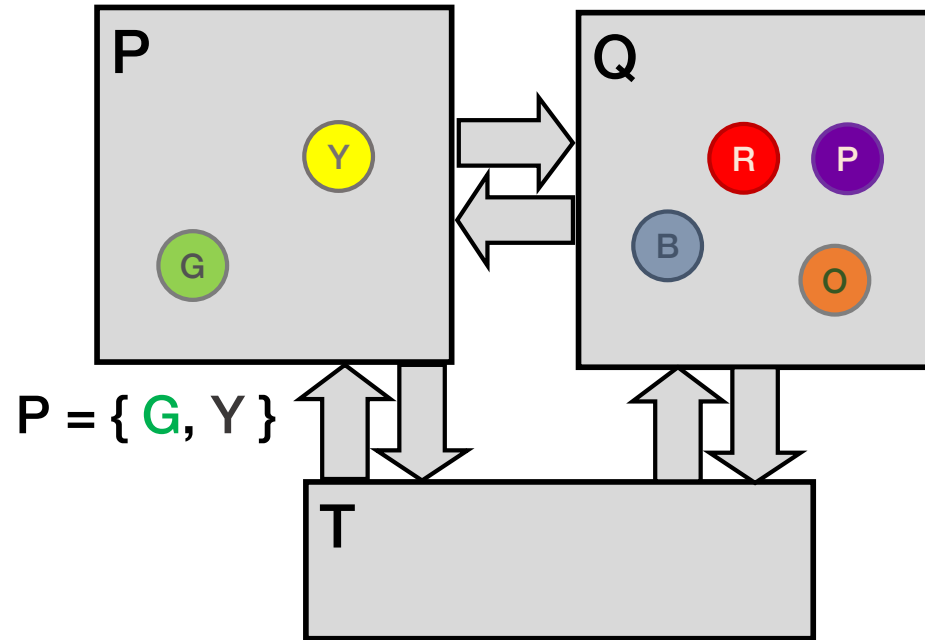
chan(Q, P) = { P }

chan(Q, T) = { R }

T = { }

chan(T, P) = { }

chan(T, Q) = { }



# Chandy-Lamport Puzzle #6

Is this snapshot possible? And if so, how?

P = { G, Y, }

chan(P, Q) = { }

chan(P, T) = { }

Q = { B }

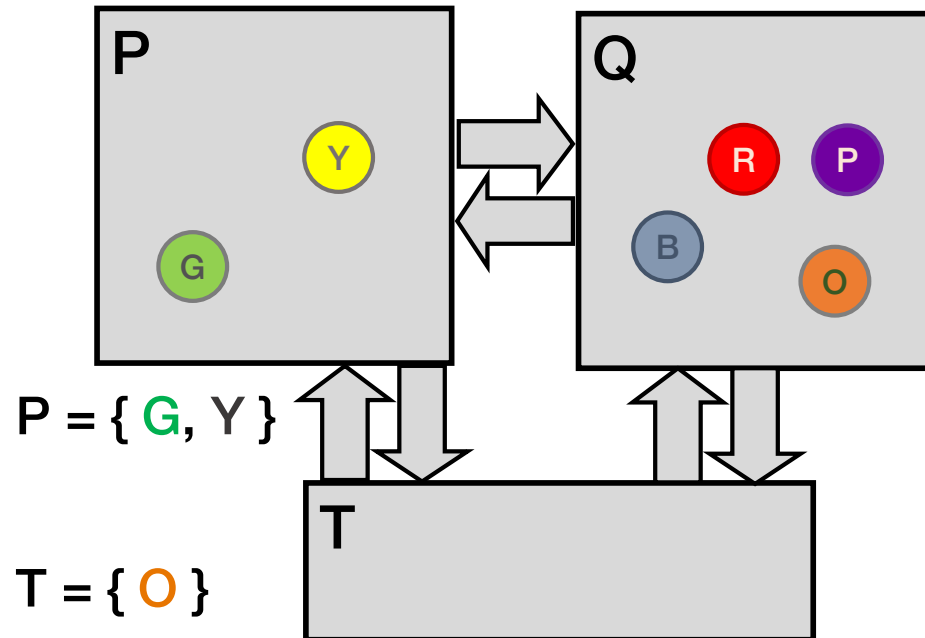
chan(Q, P) = { P }

chan(Q, T) = { R }

T = { O }

chan(T, P) = { }

chan(T, Q) = { }



# Terminating a Snapshot

- **Distributed algorithm: No one 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**

# Take-away points

- **Distributed Global Snapshots**
  - FIFO Channels: we can do that!
  - Chandy-Lamport algorithm: use marker messages to coordinate
- **Reasoning about concurrency**
  - You're doing it!
  - Use trickier and trickier puzzle methodology to understand how (and if) systems really work



