Network Game Programming

Understanding networking and making good choices

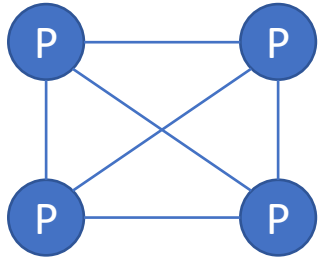
Game Types

- Client-server
- Peer to peer
- Things to consider:
  - Number of connections
  - Bandwidth of clients/server
  - Players with slow connections
  - Cheating

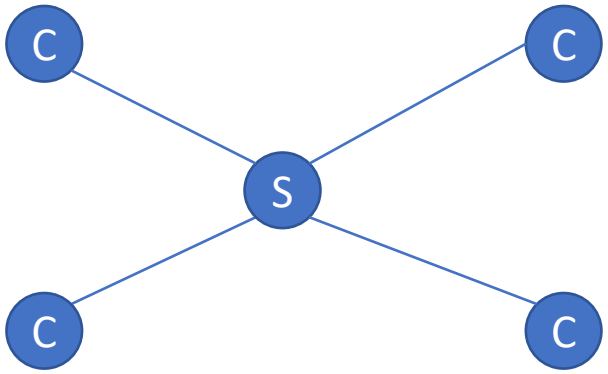
Network Architectures

- No networking
- Peer 2 peer
- Client-server
- Server-network

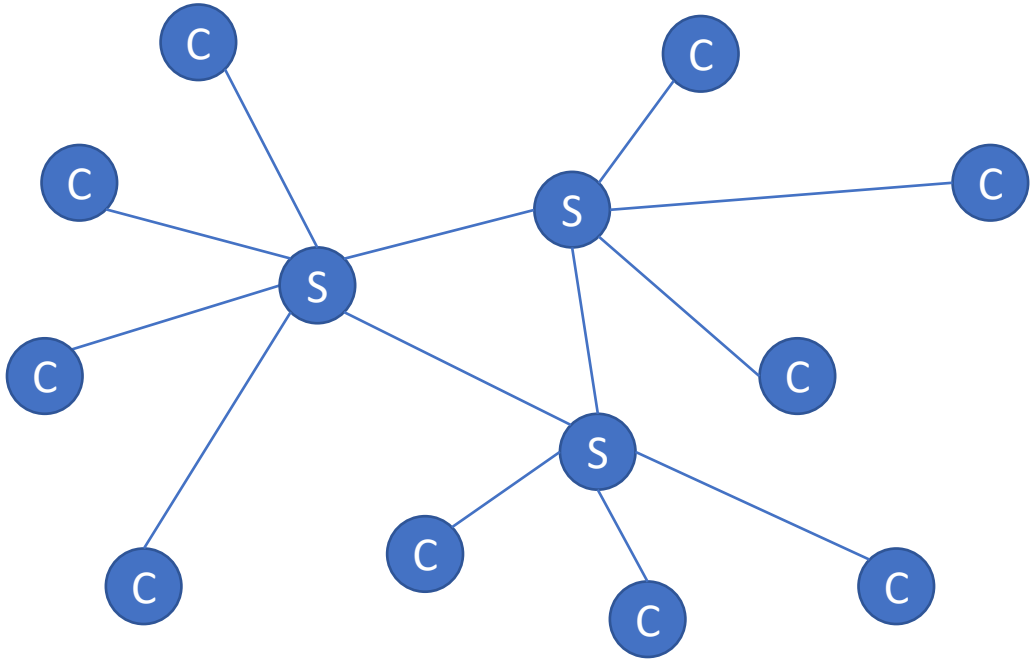
No networking



Peer 2 peer



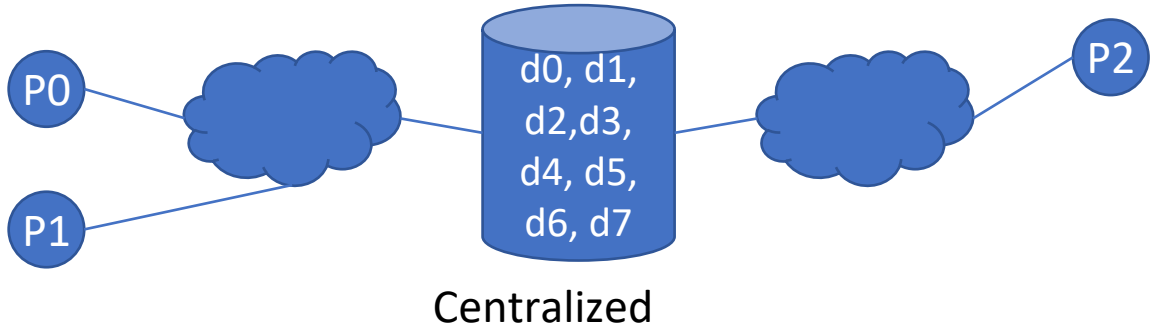
Client-server



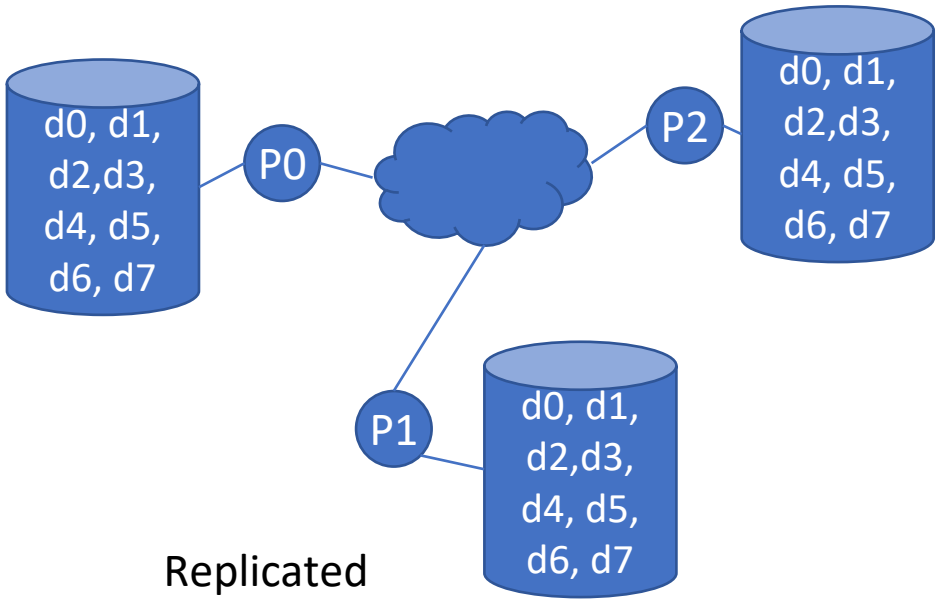
Server-network

Data Distribution

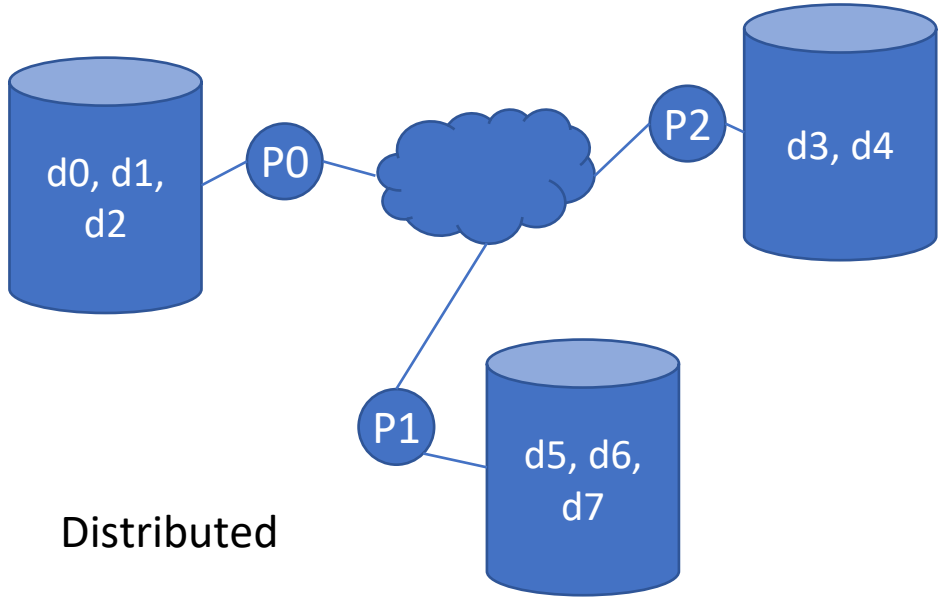
- Centralized
- Replicated
- Distributed



Centralized



Replicated



Distributed

Behavior Types

- Deterministic Behavior
- Non-Deterministic Behavior

Deterministic Behavior

- The behavior of an object at any given time is known
- Deterministic behavior might:
  - Follow the laws of physics
  - Have a fixed set of behaviors
  - Use a pseudo-random sequence to determine actions
- Seed all clients random-number generators with the same value to eliminate the need for updates

Non-Deterministic Behavior

- Impossible to know what is going to happen
  - User input or decisions
  - AI decisions

Network Lagging

- Network lagging can’t be avoided
- Compensation for network lagging is needed
- Every event already happened
- Need to either:
  - Predict what is about to happen (and be right)
  - React to what has already happened before it has actually happened i.e. Sleep(-1)
  - When you are wrong, compensate somehow

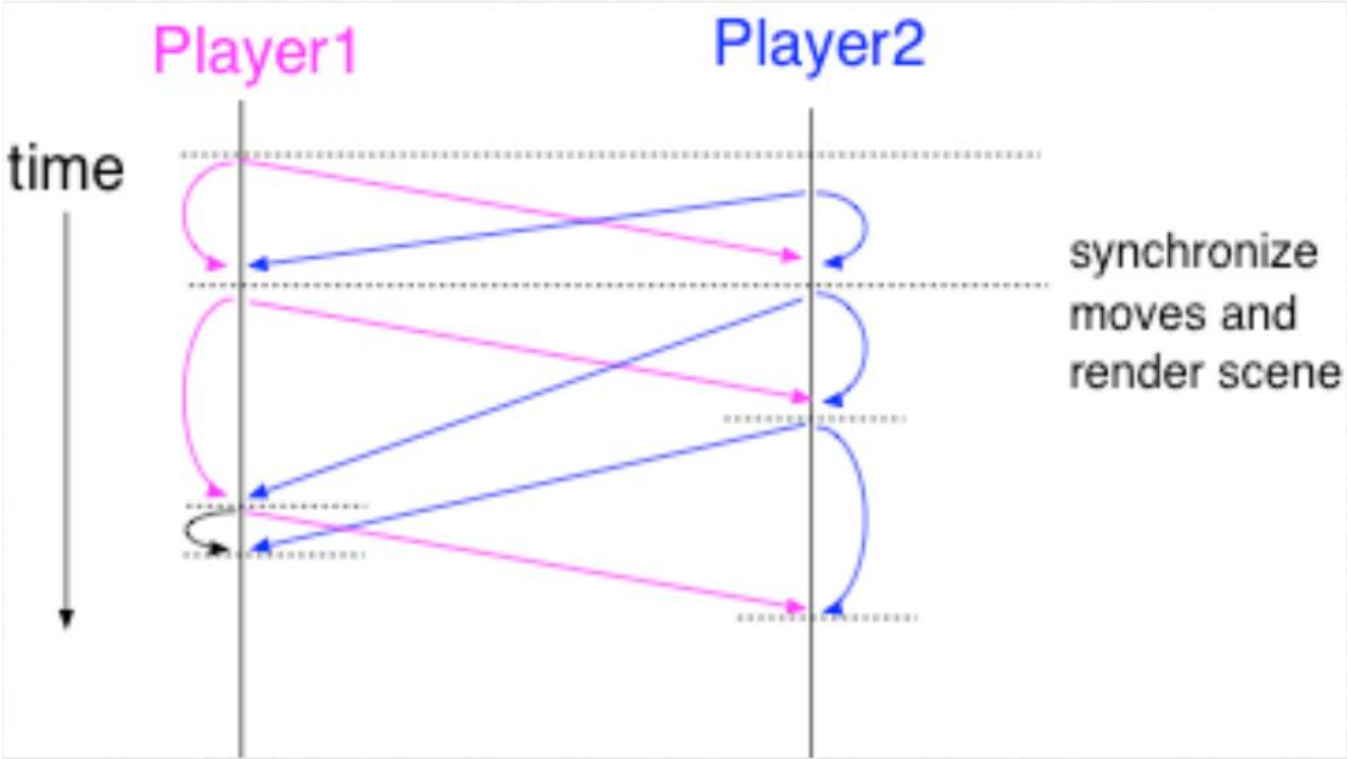
Sleep(-1) causes the current thread to suspend indefinitely, effectively blocking the thread until it's explicitly resumed or the program terminates.

Lockstep

- Send the intent to do something before you do it
- Clients operate in “turns”
  - A turn length can be dynamic
  - Could calculate based on the slowest client
- A message executes on the turn it was meant for
- A message must be sent and received prior to the turn it was meant for

Lockstep Protocol: Example

- Each player receives all other players' moves before rendering next frame
  - Long latency
  - Variable latencies
- Game speed decided by the slowest player



The easy way

- Don’t predict at all
- Get a new position from the server or other clients
- Teleport there
- If you have any lag, you’ll teleport randomly and game may not be very playable

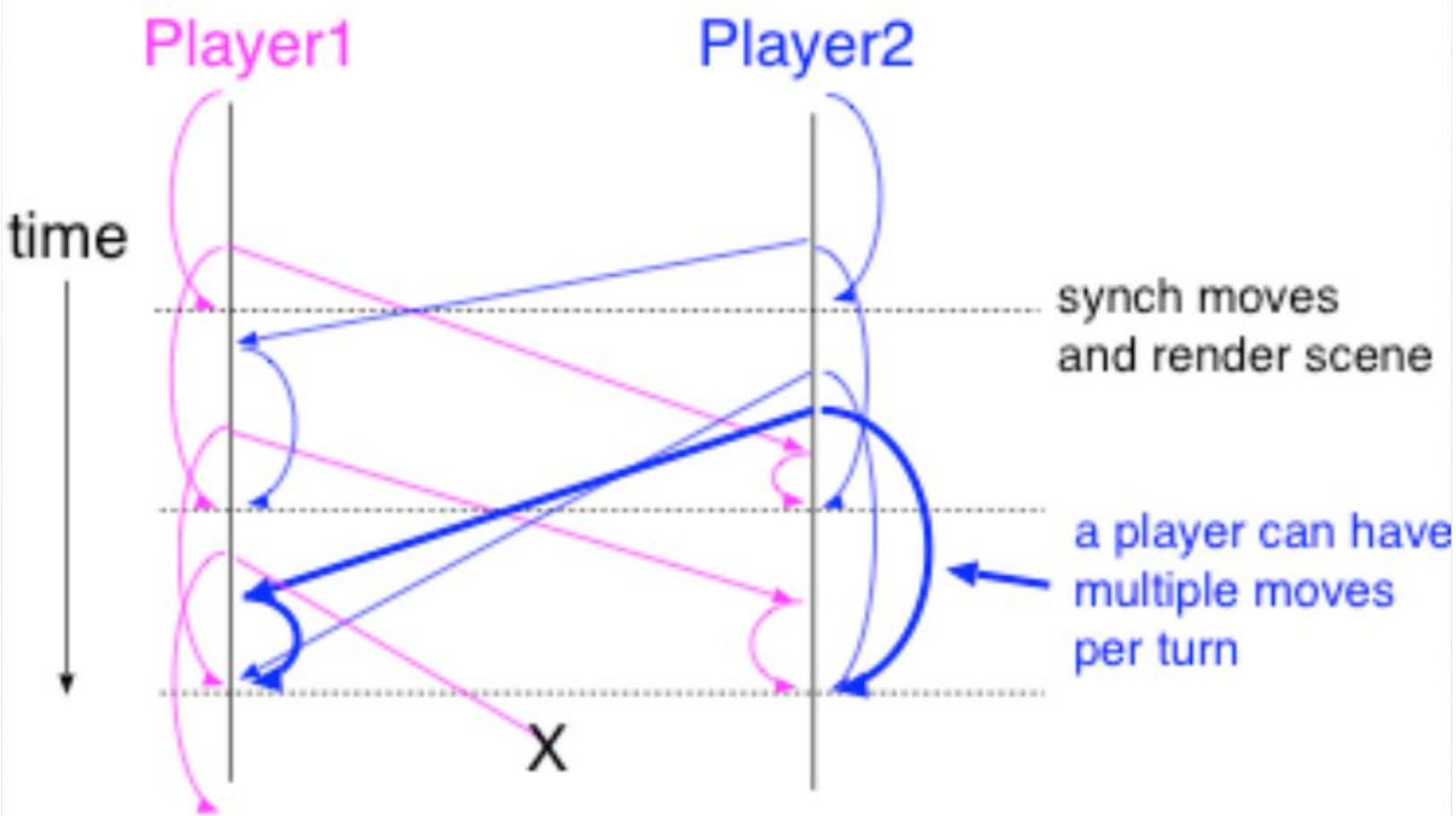
Delay executing commands

- When a command is given, don’t execute it right away
- Send it immediately.
- All hosts know how long the delay is between command receipt and execution
- By the time you need to execute the command, all hosts will have the message
- Hosts know how old a message is
- Sometimes called “moving in lock-step”
- Not very useful for real-time action games

Bucket Sync

- Algo
  - Buffer both local and remote moves
  - Play them in the future
  - Each bucket is a turn (e.g. 200ms)
  - Bucket size ~ measured RTT
- Problem
  - Game speed(bucket size) decided by slowest player
  - What if a move is lost/late?

Bucket Sync - example

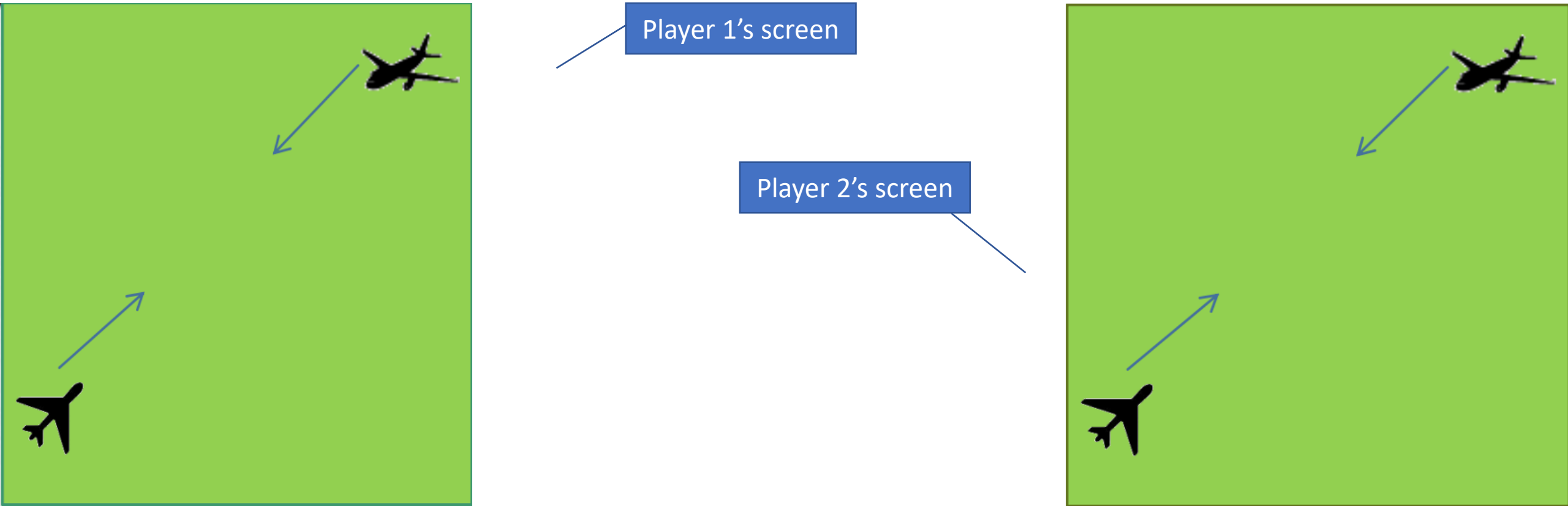


Prediction- Dead Reckoning

- Movement in one direction is likely to stay the same over time
- When movement changes, we probably won't be too far off
- Compensate for incorrect position smoothly
- Velocity may be a good way to predict
- Acceleration may be better but errors may make things worse
- Can use movement history to approximate current velocity and acceleration

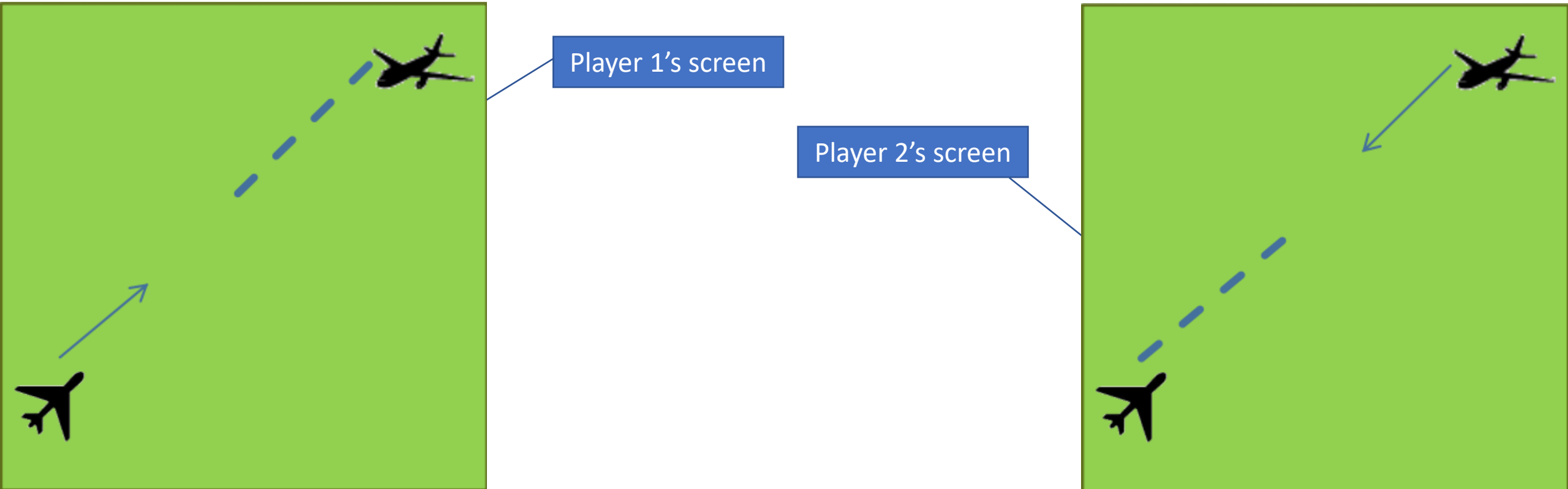
Basic Idea

- Sync at beginning
  - Position/Velocity/Acceleration



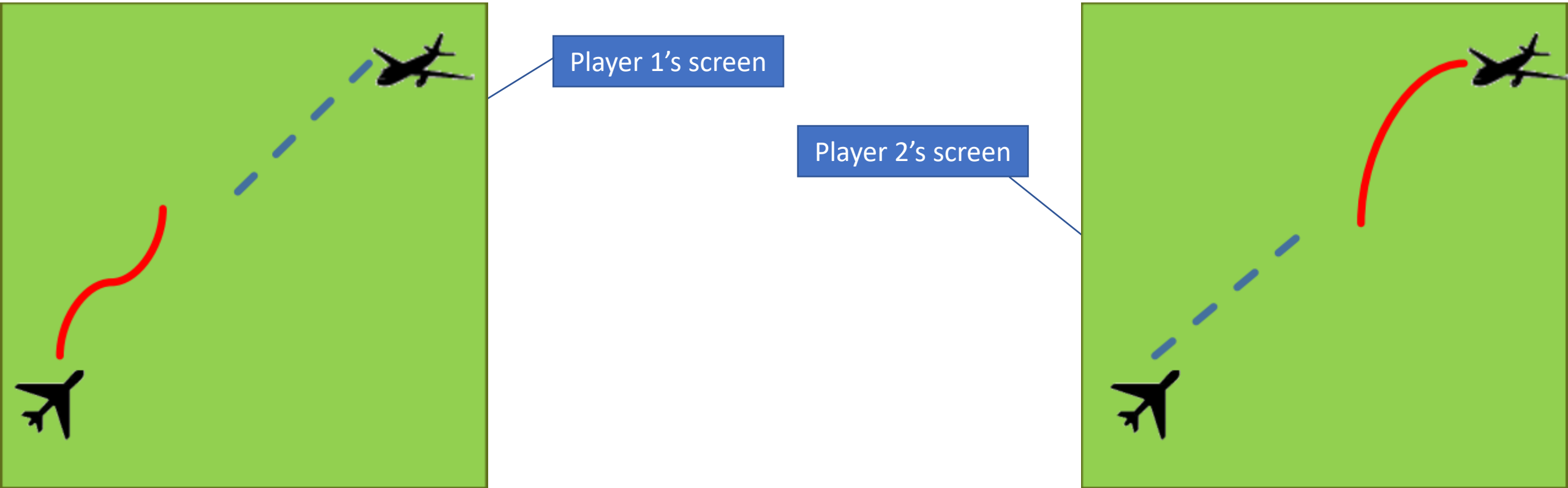
Prediction - I

- Need to predict for other players
- Use the same prediction algo
- Assume that nothing is going wrong



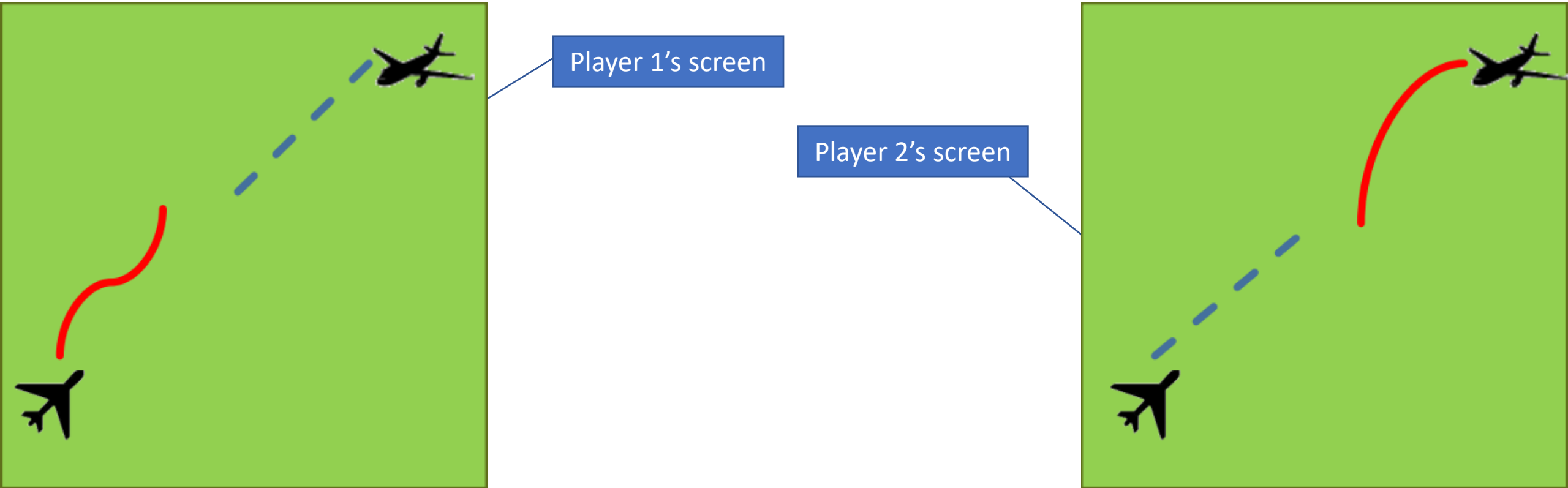
Reality Check

- Players don’t move according to your predictions!
- Question: When should I update?



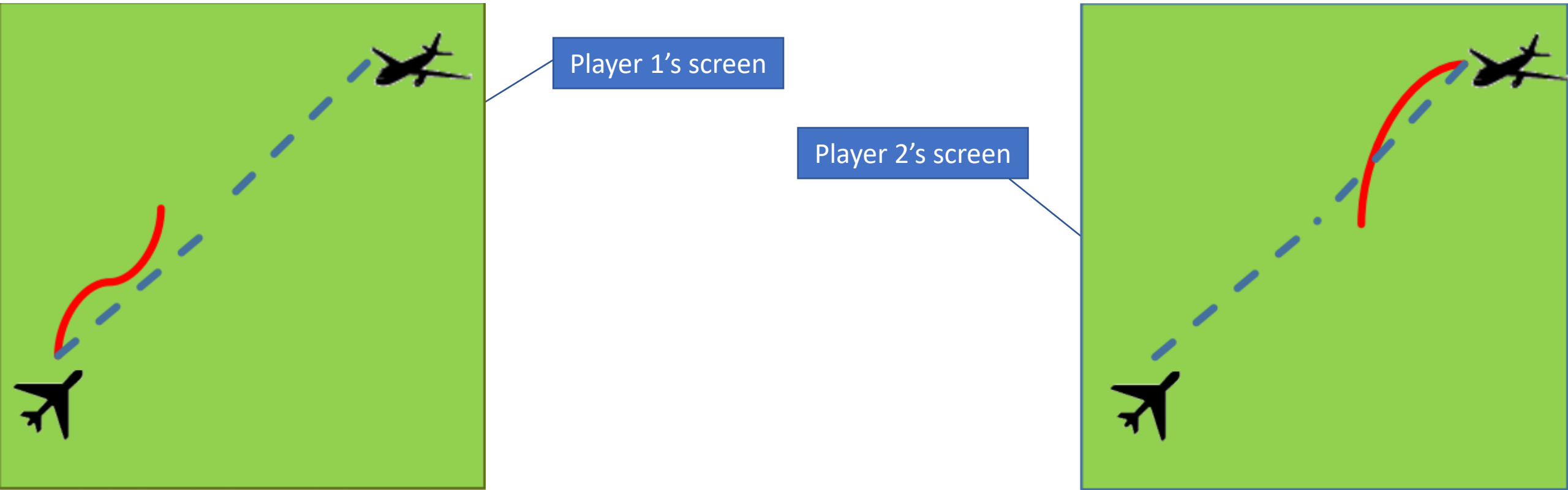
Reality Check

- Question: When should I update my position?



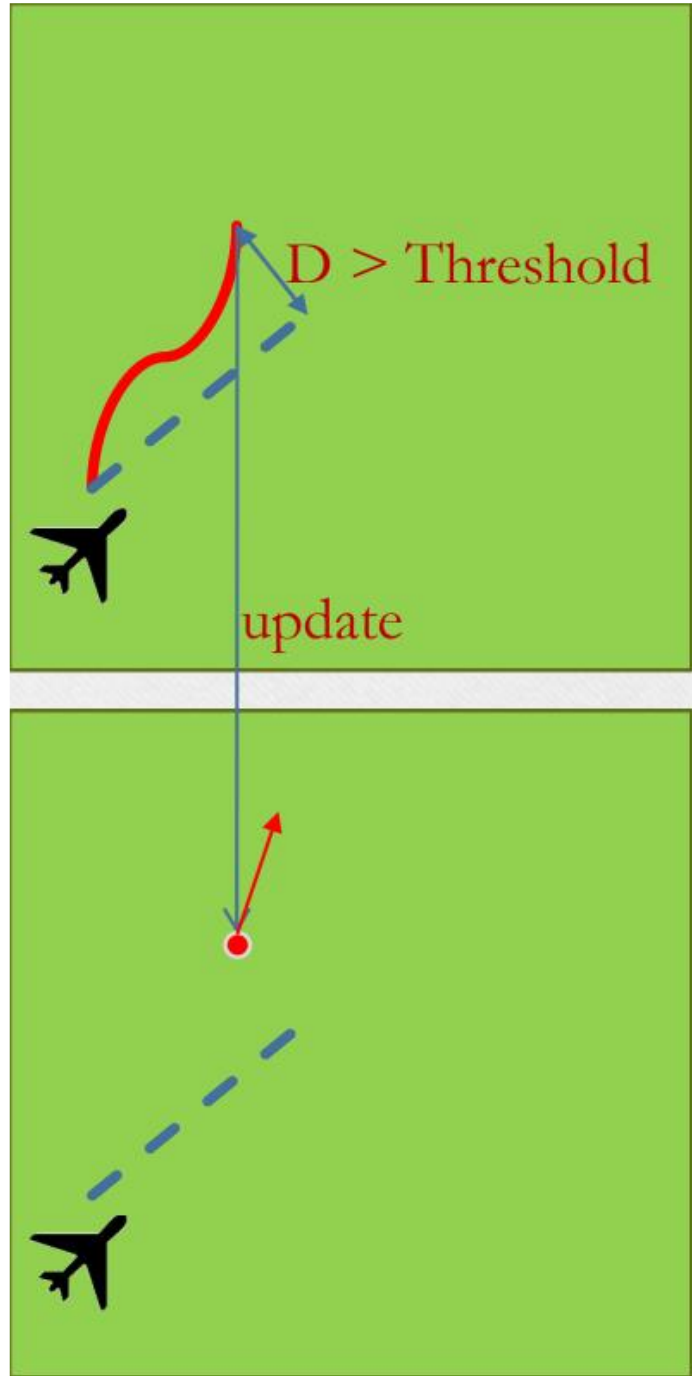
Predict - II

- Predict for yourselves too!
  - i.e., keep a copy of how other players will predict your movements



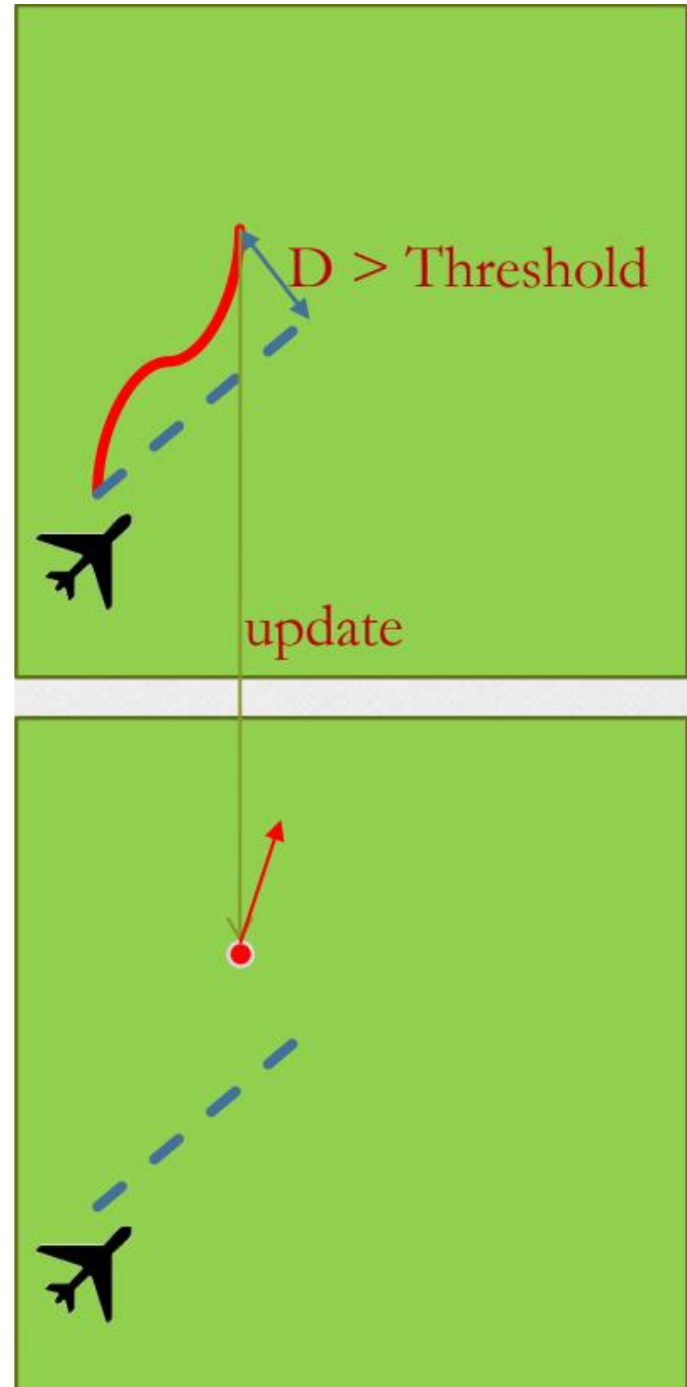
Update - II

- When to update?
  - Threshold …
  - Delay important!
- What's in an update?
  - Position
  - Velocity
  - Acceleration



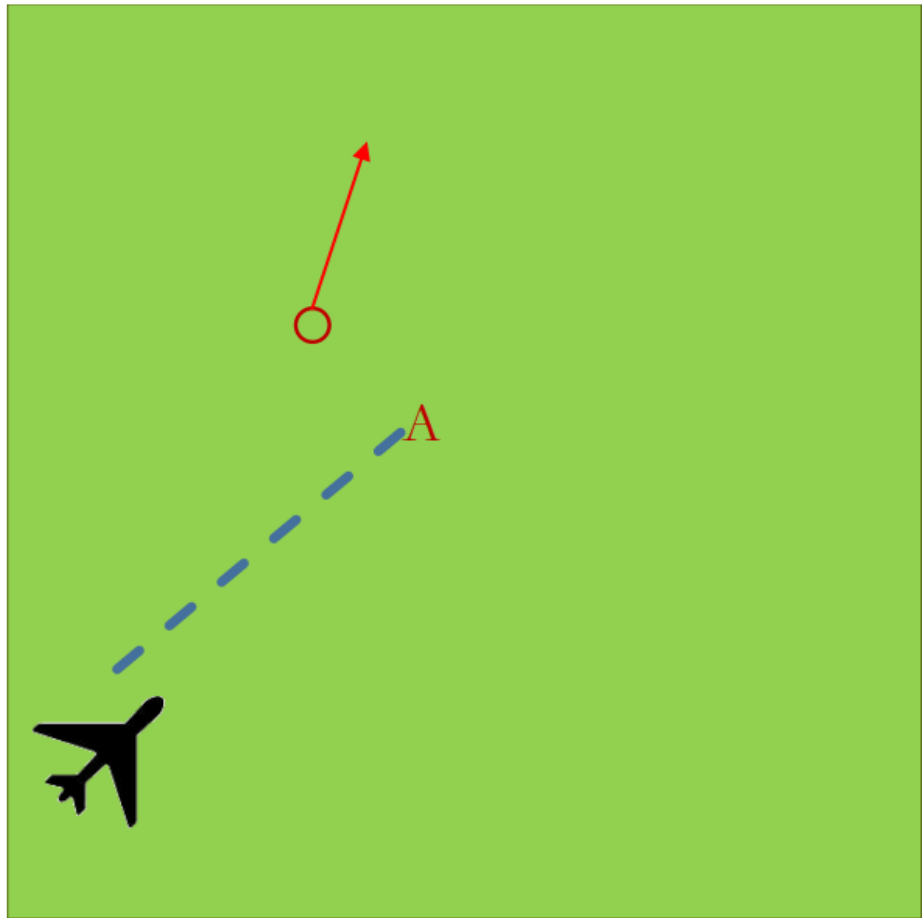
What to do when updates received?

- No updates
  - Pretend no problem



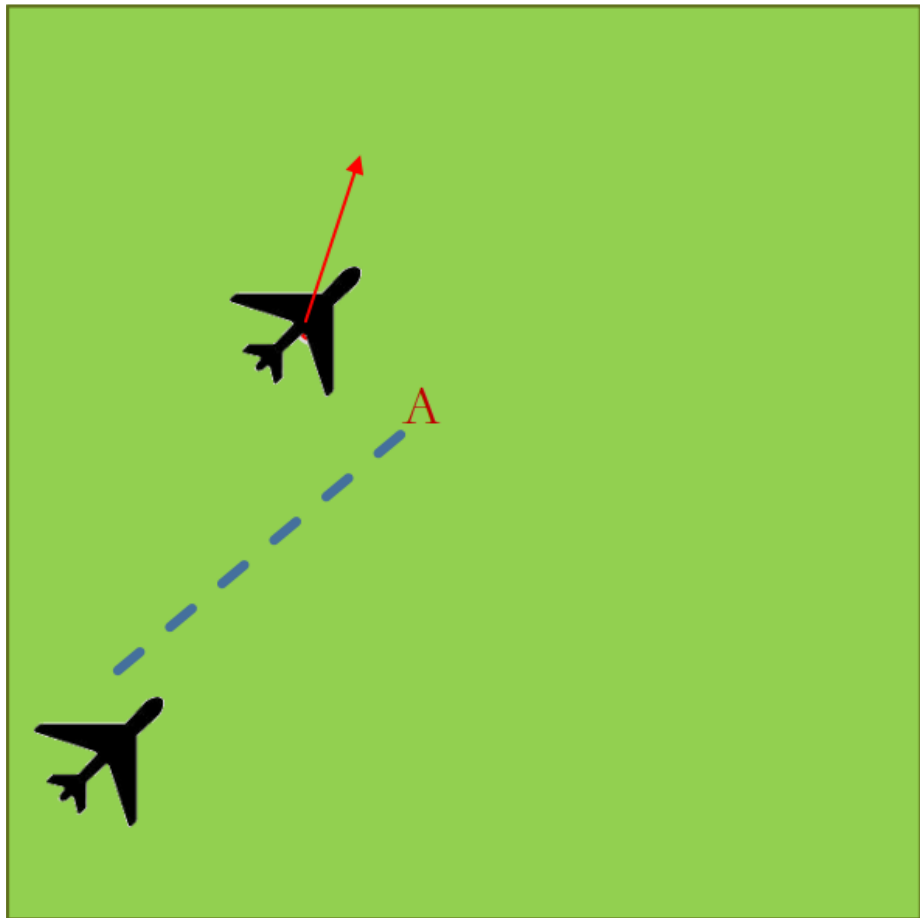
Convergence

- No updates
  - Pretend no problem
- Update comes..
  - Need to think about how to get from point A to…?
- A is the old predicted position



Option 1: Snap Convergence

- Snap to correct position
- Problem:
  - Very jerky behavior
- May be ok..
  - When movement is small
  - Movement is far away from player's perspective
- Cannot avoid totally

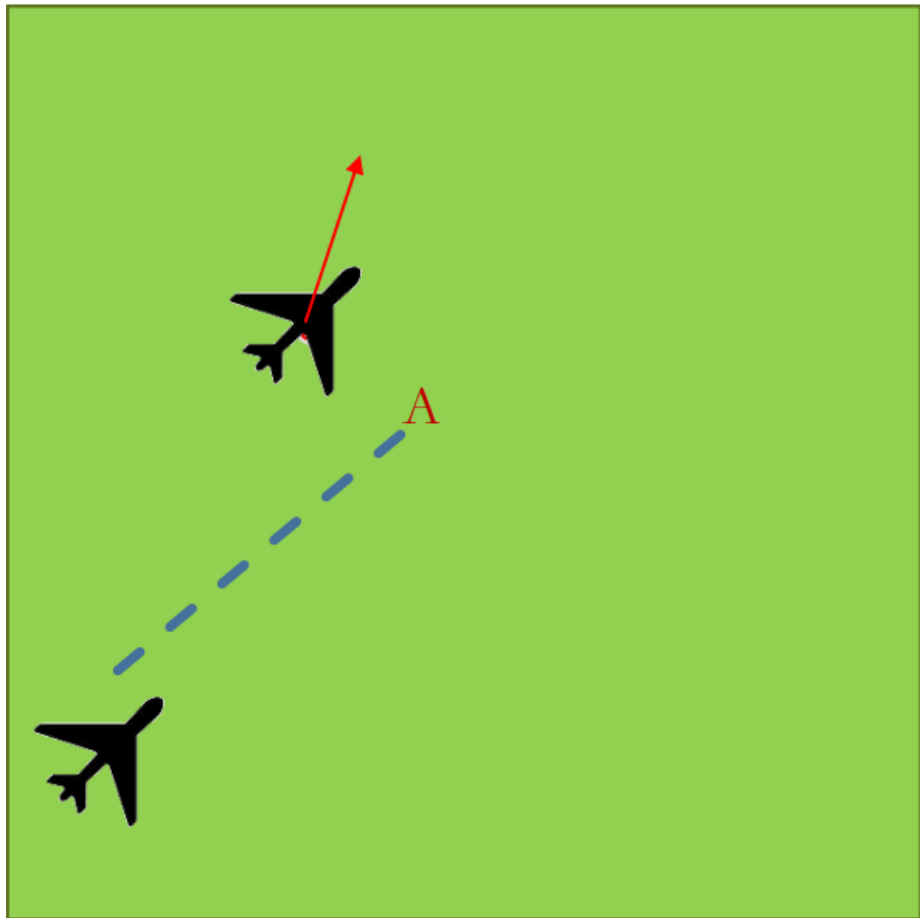


Option 2: Linear Convergence

- Predict “real” position
- Convergence time
  - When you want to meet.
- Use Point B and A to work out new velocity

$$\text{New Position} = \text{Old Postion} + v_{new}t$$

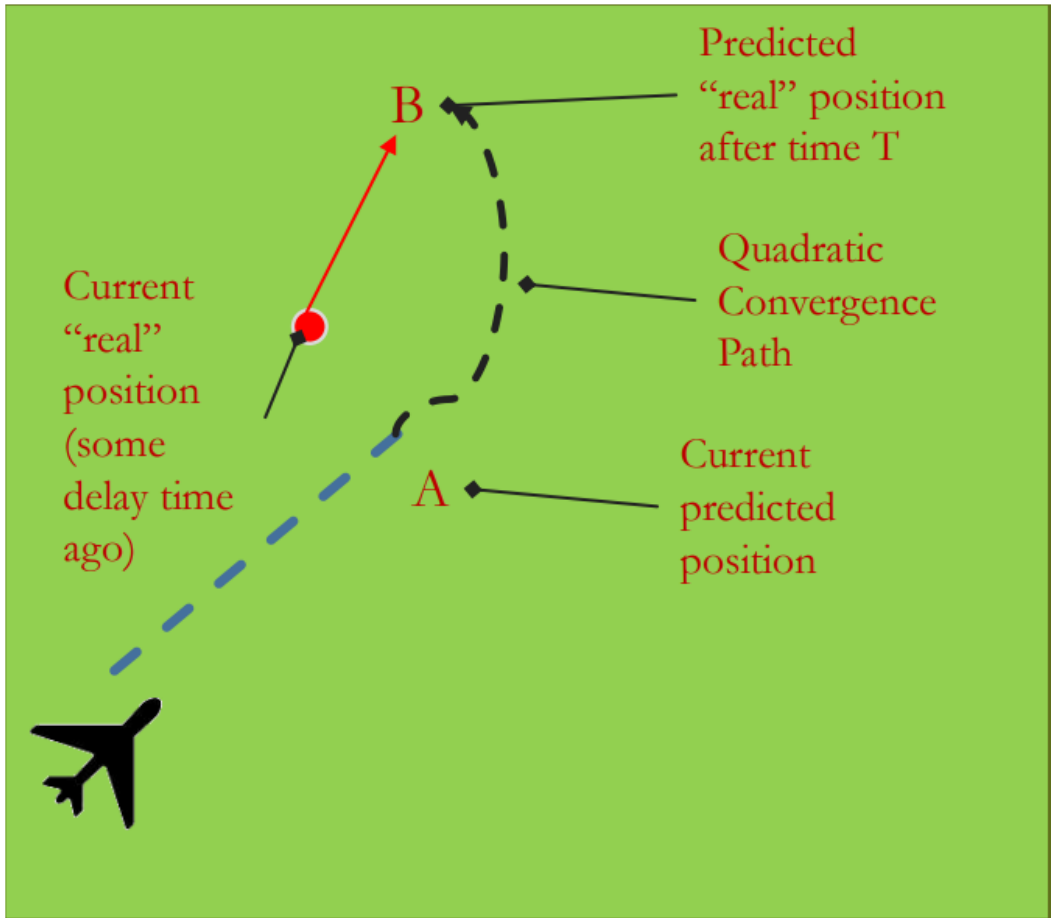
$$B = A + V_{new}t$$



Option 3: Quadratic Convergence

- Include acceleration
- Solve quadratic equation for
  - Speed v
  - Acceleration a
- Prob: final velocity

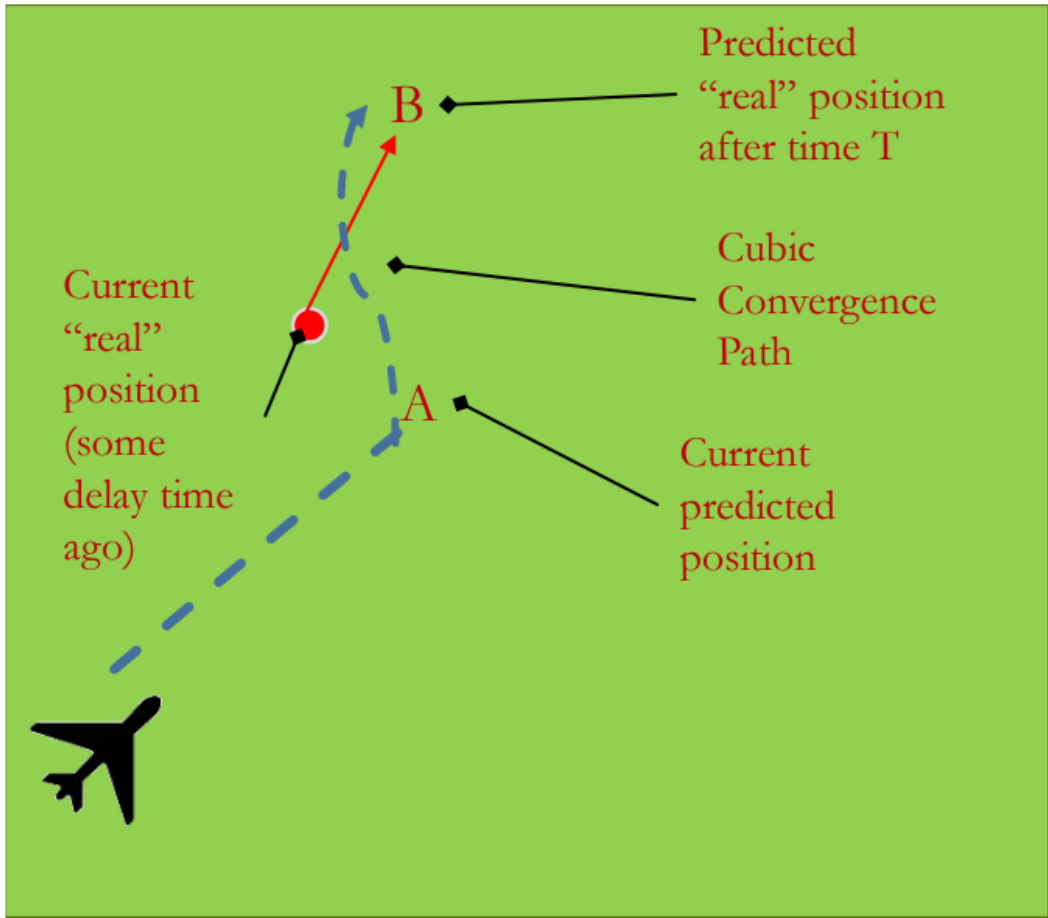
$$\text{New Postion} = \text{Old Position} + vt + 1/2(at^2)$$



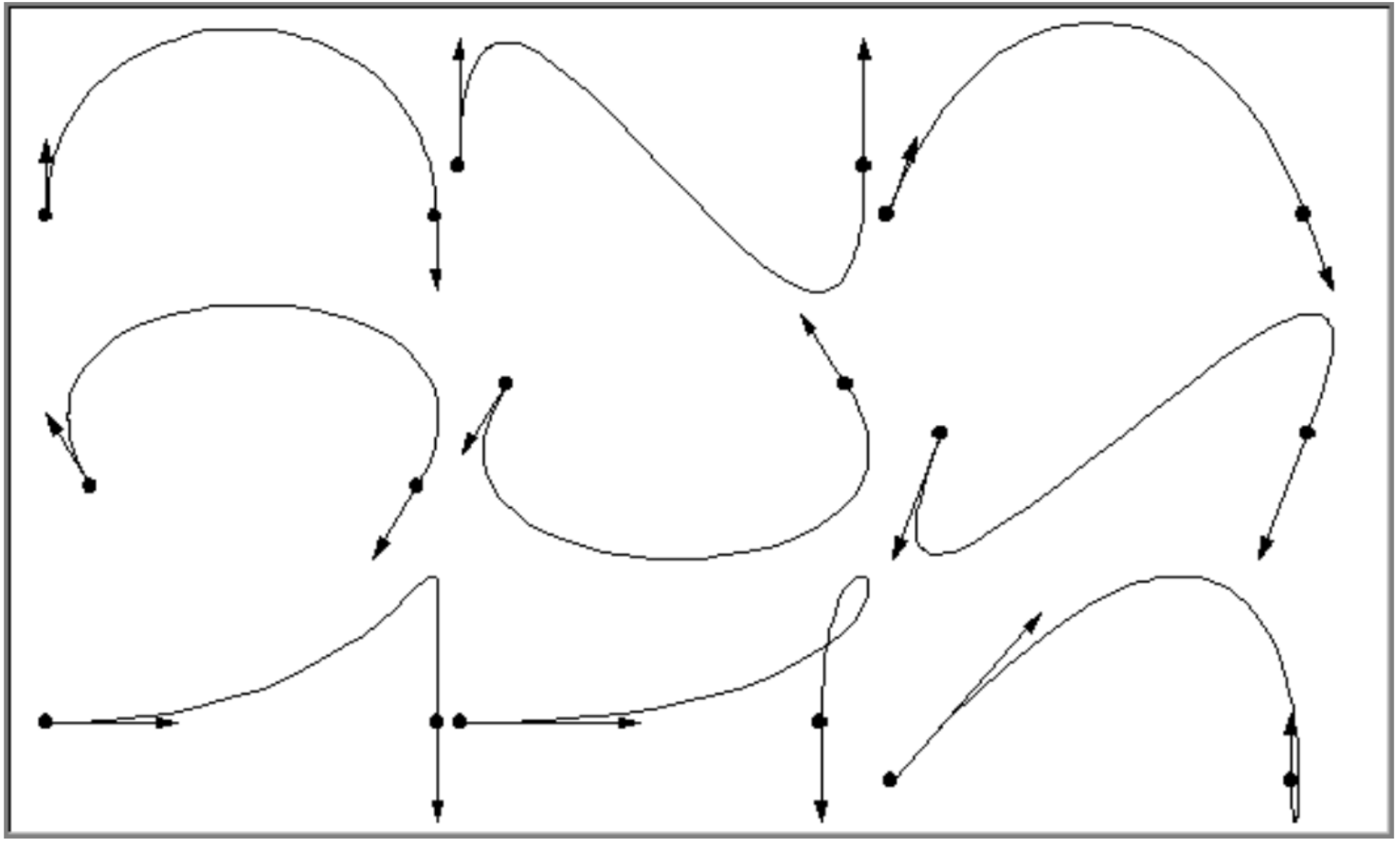
Option 4: Cubic Convergence

- Include acceleration
- Smoothest

$$\text{New Postion} = At^3 + Bt^2 + Ct + D$$

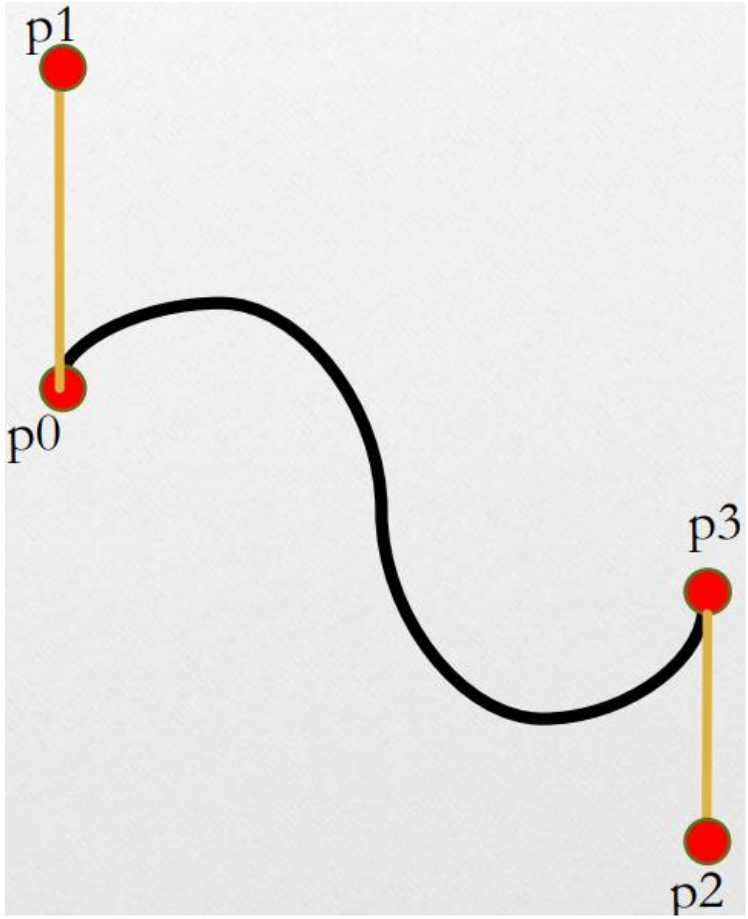


Hermite Splines



Hermite Splines – 4 coordinates

- A, B, C, D determines the shape of the curve
- t range between 0 and 1
- reach p3 when t = 1
- |p1-p0| and |p3-p2| are important



Applying Hermite Splines to Dead Reckoning

- What must be determined?
  - Magnitude of tangential vectors
  - Use original position for p0 and p1
  - Position p2 and p3

Compensating for when you're off

- Teleport there! All games do this to some extent.
- Sometimes you have no choice
- Interpolate between your actual location and the "correct" location to compensate smoothly
- Use splines- often the best way to choose the smoothest path

Splines (part 1)

- You need four points:
  - Coordinate 1- starting position
  - Coordinate 2- position after 1 second given current velocity (coordinate 1 + starting velocity)
  - Coordinate 3- position after 1 second using reversed ending velocity (coordinate 4 – end velocity)
  - Coordinate 4- end position (usually new packet that just arrived)

Coordinate Equations

$$\text{Coordinate 1} = \text{Starting position}$$

$$\text{Coordinate 2} = \text{Position after 1 second using the starting velocity} = Coordinate1 + StartVelocity \times 1$$

$$\text{Coordinate 3} = \text{Position before 1 second using reversed ending velocity} = Coordinate4 - EndVelocity \times 1$$

$$\text{Coordinate 4} = \text{Ending position}$$

Splines (part 2)

- $x = At^3 + Bt^2 + Ct + D$
- $y = Et^3 + Ft^2 + Gt + H$
- t = time (0 to 1)
- $A = x_3 - 3x_2 + 3x_1 - x_0$
- $B = 3x_2 - 6x_1 + 3x_0$
- $C = 3x_1 - 3x_0$
- $D = x_0$
- $E = y_3 - 3y_2 + 3y_1 - y_0$
- $F = 3y_2 - 6y_1 + 3y_0$
- $G = 3y_1 - 3y_0$
- $H = y_0$

Splines- (part “what was all that?”)

- First, get the current position and velocity. Calculate the future position using the current velocity
- Decide how many steps you’re going to move on this spline before calculating a new one (t)
- Calculate the final position after t intervals based on the packet you just got
- Use final position and subtract velocity * t
- Could also use acceleration with velocity to improve smoothness

YOU MUST KNOW DELAY!

- You must at all times have a good approximation of how late each message is.
- Each client may have a different delay
- If the server knows the delay of all the clients, it can compensate
- This may change over time

What to send?

- Keep data as small as possible
- Cram as much data into packets as possible
- Stay below your MSL (Maximum Segment Lifetime)!!!
  - ➢ You need to know what type of network the game should target for this
- Don’t send the same message twice.
  - ➢ If nothing has changed, there’s no need to mention it
- Compression is possible but the algorithm is often costly

Messages could have

- ID of object the message represents.
- The action on that object. May not necessarily represent movement e.g. game start, player death, object creation, etc.
- Position- you need this in some form
- Velocity- you can send this for more accuracy or you can approximate from past positions
- Acceleration- can send or approximate as with velocity

Ways to make messages smaller

- Use fewer bytes/bits!
  - Use enums for all actions
  - Keep object IDs small
  - Send deltas only (need to sync periodically though)
- Aggregate messages for same objects
- Best way to make a message smaller: don’t send it!
  - Only send messages that a client cares about

Area of interest filtering

- I don’t care about the player behind the building
- I can’t see the player about to sneak up on me anyway
- I can’t see things across the map
- Looking through a rifle scope gives a very small field of vision
- If I can’t see it, I don’t care about it
- Can implement as a range, a cone of perception, or both

Local perception filters

- Things close up
  - Must see as close to real-time as possible
  - Must see in enough detail
- Things far away- I may not notice them so much
  - Can use less detail
  - Can use less information to update
  - Can even be out of date (but should still be real-time)
  - Bullet time- I can speed things up as they get closer

Perception Filters

- A method used to hide communication delays in networked virtual environments (source: Sharkey et al.)
- Essentially: speed up and slow down speed of passive entities when they are near local and remote players

Active Entities

- Active entity (i.e. player)
  - Take actions on its own
  - Generate updates
  - Human participants, computer controlled-entities
  - Cannot be predicted typically
  - Rendered using state updates adjusted for the latency

Interaction between players

- Interaction = comm. between the players
  - Local players: immediate
  - Remote players: subject to the network latency
    - ✓ Time frame = current time – communication delay
- Interaction = players exchanging passive entities
  - Passive entities are predictable (can be rendered in the past or in the future)
- A passive entity can change its time frame dynamically
  - The nearer to a local player, the closer it is rendered to the current time
  - The nearer to a remote player, the closer it is rendered to its time frame

Passive Entities

- Passive entity
  - Reacts to events from the environment, does not generate its own actions
  - Inanimate objects (e.g. rocks, balls, books)
  - Active entities interact with passive entities
  - Rendered according to the latency of its nearest active entity
  - Reacts instantaneously to the actions of a nearby active entity