Centralized Server Architecture
Short circuiting: players perform “local prediction” to predict their own state without waiting for replies from the server.
Opponent Prediction
Dead Reckoning
Extrapolation
Also used in marine navigation, ariel navigation, GPS etc.
A general technique that works between any two parties (players/server). But we will see example for a server and a player.
Server keeps track of the position of entities through updates from the players.

Naive method: update position only
Two issues:

Message overhead
Delay jitter
Delay jitter causes player’s movement to appear erratic.

Naive method: update position only
Improvement: update the position and velocity -- if an update arrives late, server can predict B’s position.
Improvement: if the velocity remain constant, server can predict every position at all time.
Server, however, needs to update position and velocity when velocity has changed.
\[ x[t_i] = x[t_{i-1}] + v \times (t_i - t_{i-1}) \]

- \( x[t] \): position of entity at time \( t \)
- \( v \): velocity of the entity
But velocity may change all the time (e.g. a car accelerating). To counter this, we send position, velocity, and acceleration as update.


\[ x[t] \quad \text{position of entity at time } t \]

\[ v \quad \text{velocity of the entity} \]

\[ a \quad \text{acceleration of the entity} \]

\[ x[t_i] = x[t_{i-1}] + v(t_i - t_{i-1}) + \frac{1}{2}a(t_i - t_{i-1})^2 \]
caution: any delay in updating the acceleration would result in large error in position.
History-based Prediction
\[ x[t] \] position of entity at time \( t \)

\( v \) velocity of the entity

\[ v = \frac{x[t_{i-1}] - x[t_{i-2}]}{t_{i-1} - t_{i-2}} \]

\[ x[t_i] = x[t_{i-1}] + v \times (t_i - t_{i-1}) \]
\[ x[t] \quad \text{position of entity at time } t \]

\[
x[t_i] = x[t_{i-1}] + \frac{t_i - t_{i-1}}{t_{i-1} - t_{i-2}}(x[t_{i-1}] - x[t_{i-2}])
\]
We will still need substantial number of updates if the direction changes frequently (e.g. in a FPS game).
idea: trade-off message overhead and accuracy. No need to update if error is small.
Server

Player A

time
where the entity is according to server
where the entity is according to A at the server
A’s version of the entity’s position is now too far away from the correct position. Server updates A with the new velocity and position.
A converges the entity to the correct position smoothly.
How to set threshold?
Depends on games. One can adapt the threshold according to requirement (e.g. distance to other players)
drawback: higher CPU cost -- since a player needs to simulate the opponent.
**drawback:** player with higher latency experience more error (but we can introduce server lag to equalize the error).
Generalized Dead Reckoning: Prediction Contract
e.g.: “return to base” : the path of the unit can be predicted if the same path finding algorithm is executed.
e.g.: “drive along this road”
Responsive
Consistent
Cheat-Free
Fair
Scalable
Efficient
Robust
Simple
Predictions, both local and opponent prediction, is discussed in [Armi06] Section 6.2. Dead reckoning is discussed in Section 9.3 of [Smed06].


Putting it all together..
convergence period = 1
latency between A and S = 1
latency between B and S = 2
DR threshold = 1
latency is known