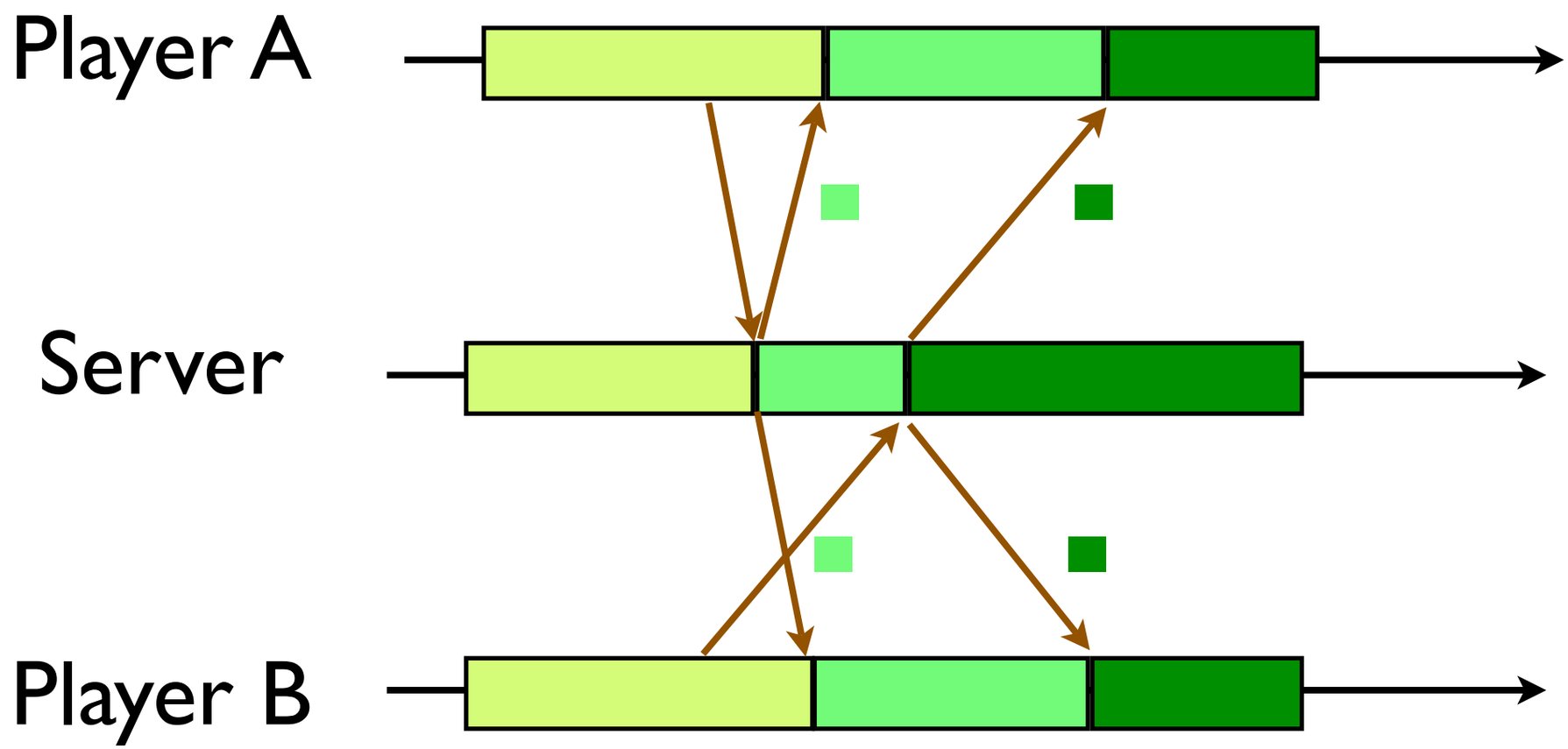
Centralized Server Architecture

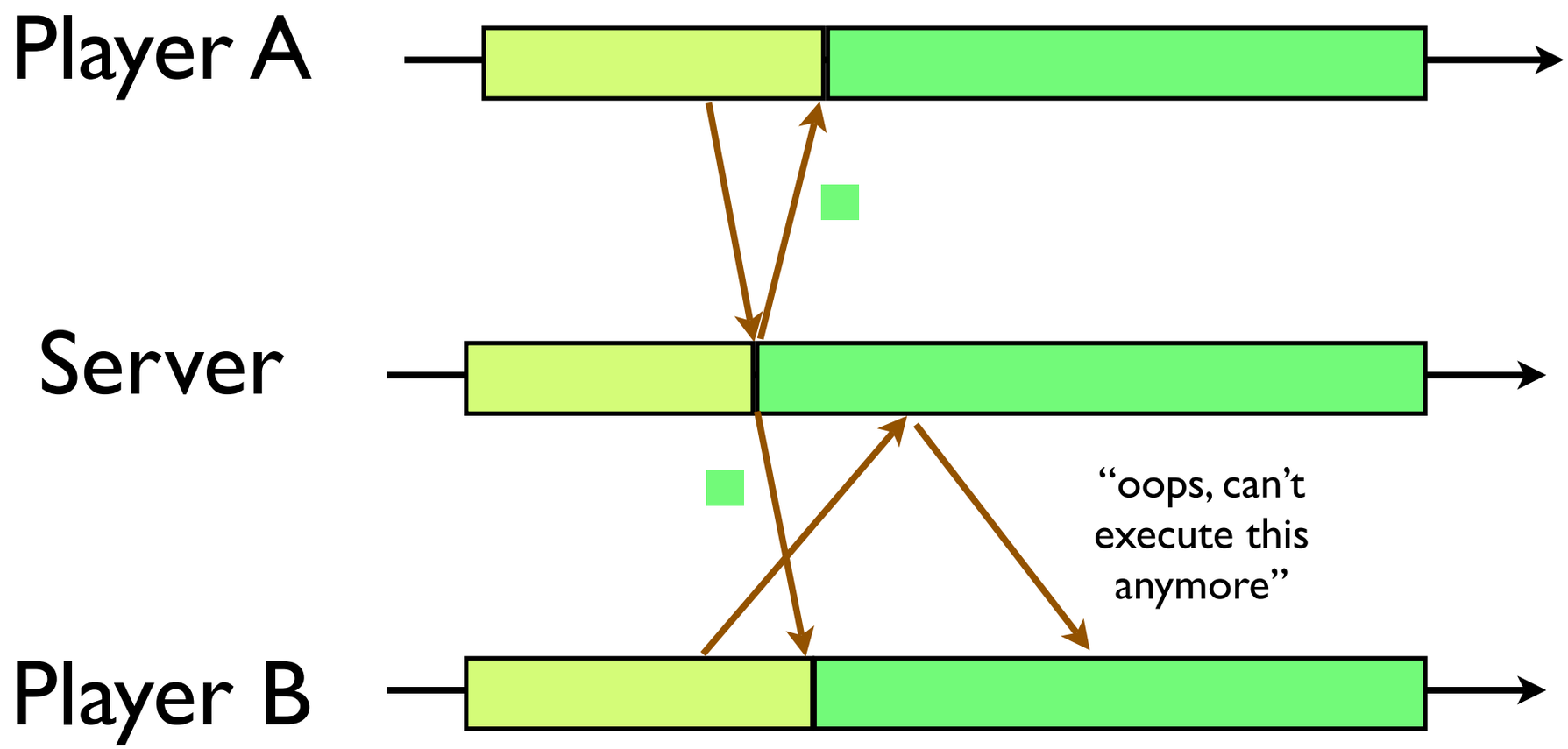
Synchronization Protocols

Permissible Client/ Server Architecture

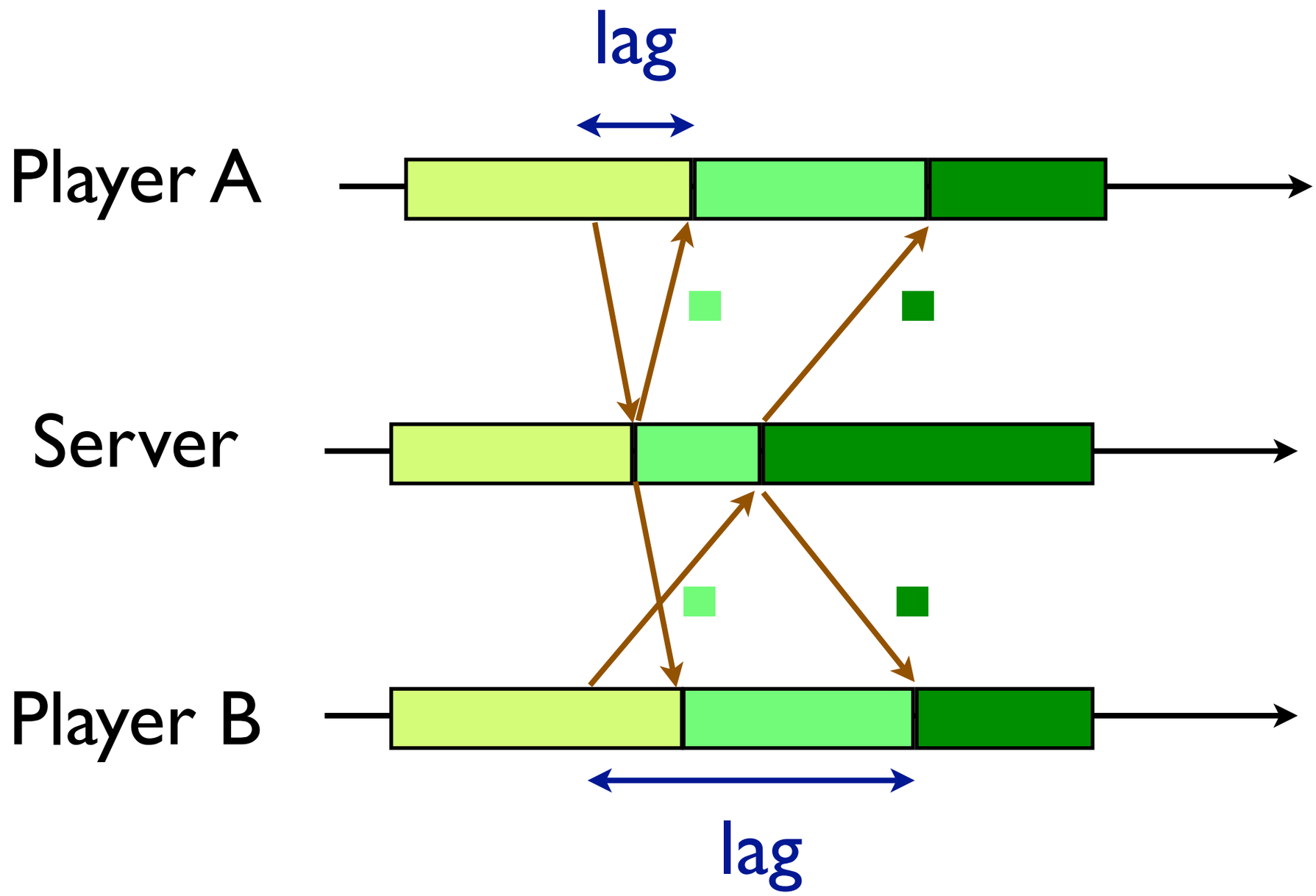
Client sends command to the server. Server computes new states and updates clients with new states.



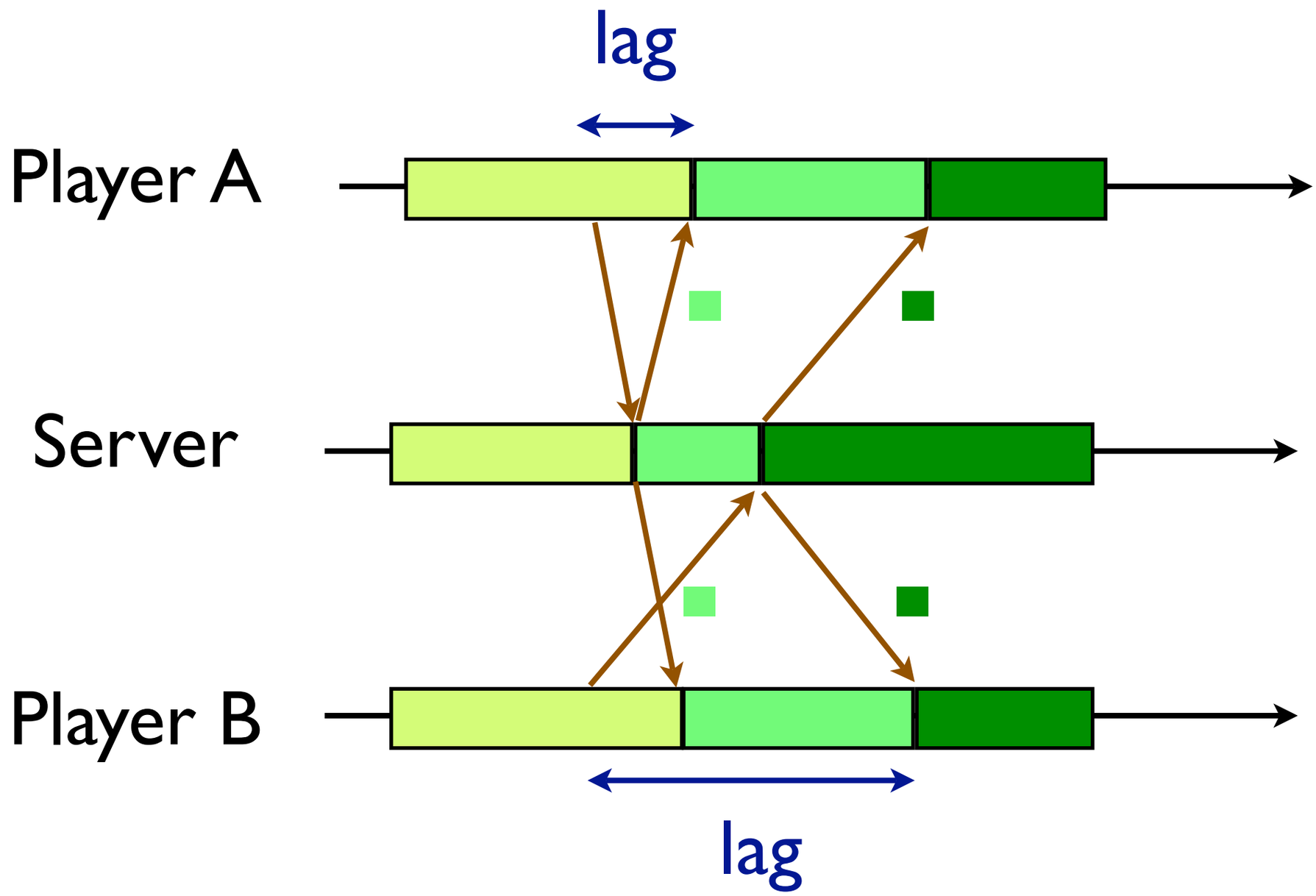
Server also serves the role of checking for consistency -- some operations might not be possible.



Problem: decrease responsiveness

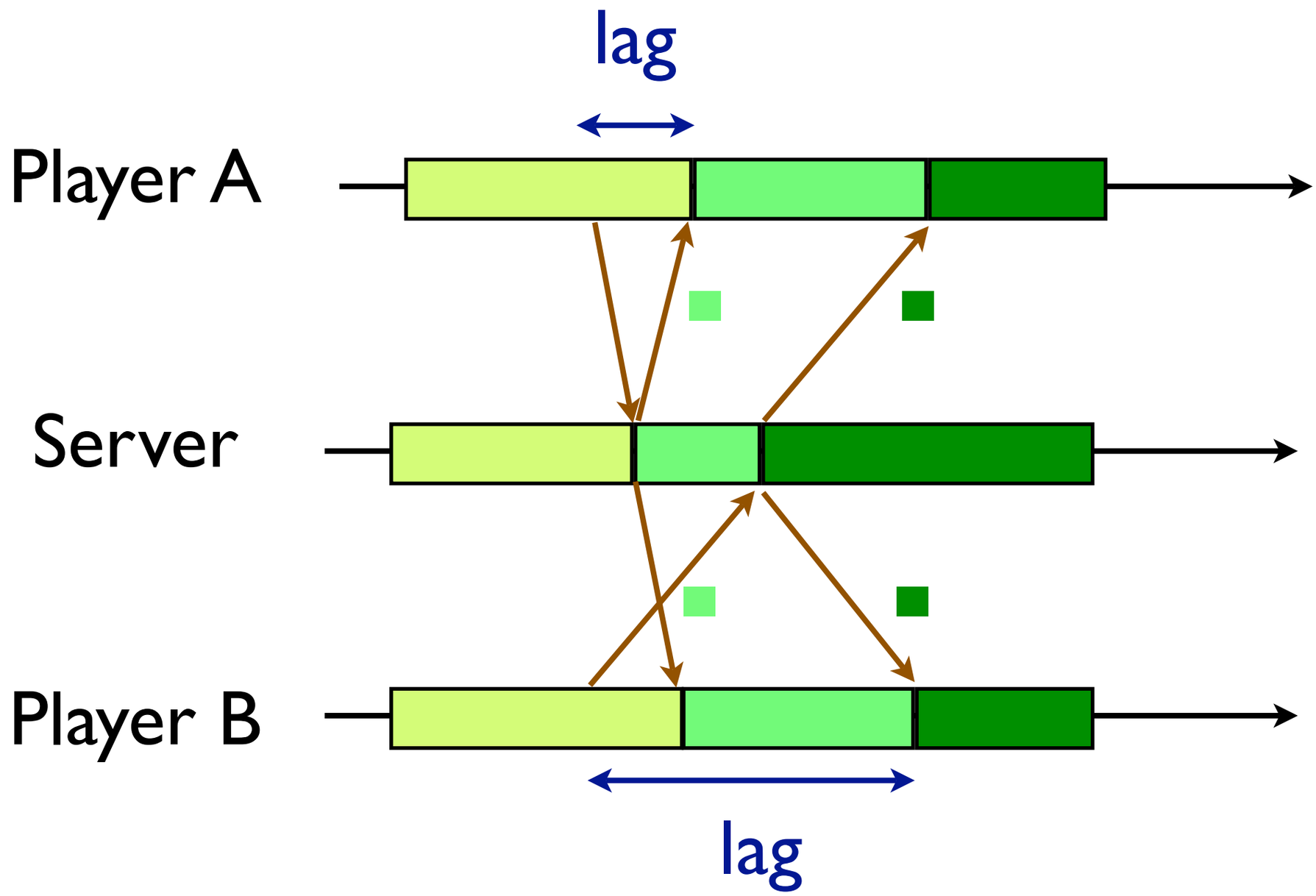


Problem: unfair to player with higher latency

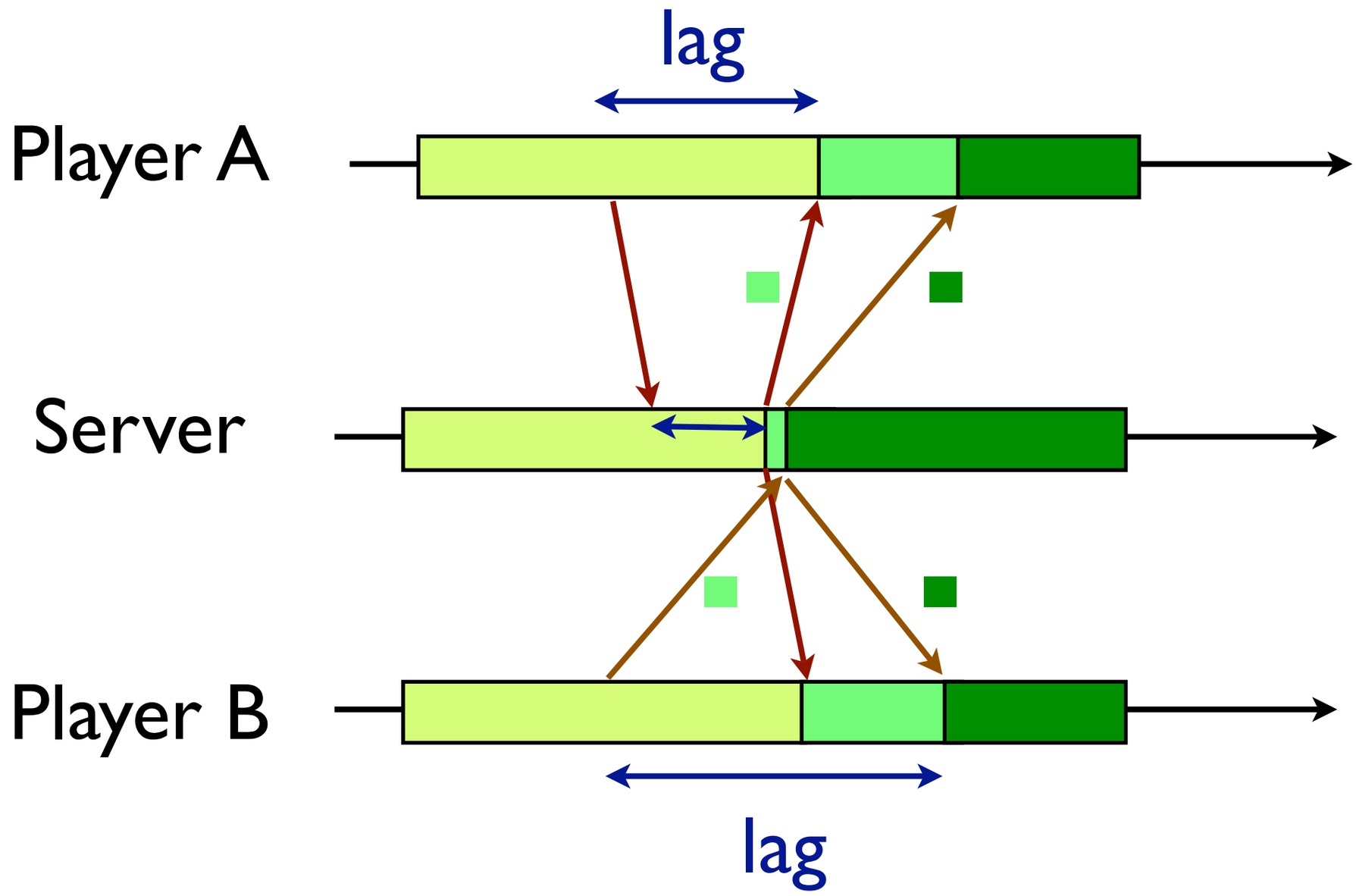


Improving Fairness

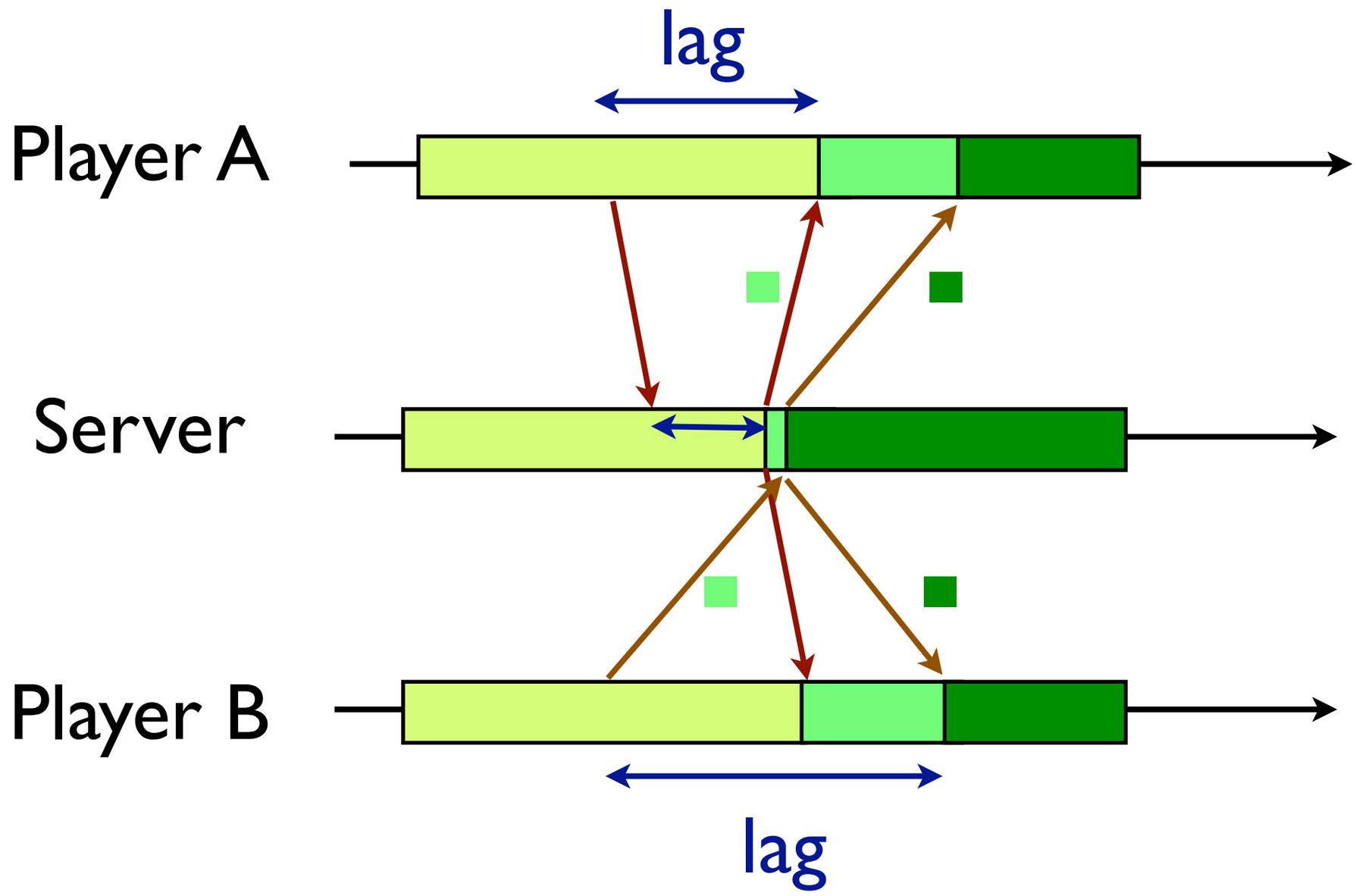
Problem: unfair to player with higher latency



Try: improve fairness by artificial delay at the server. (longer delay for "closer" player)

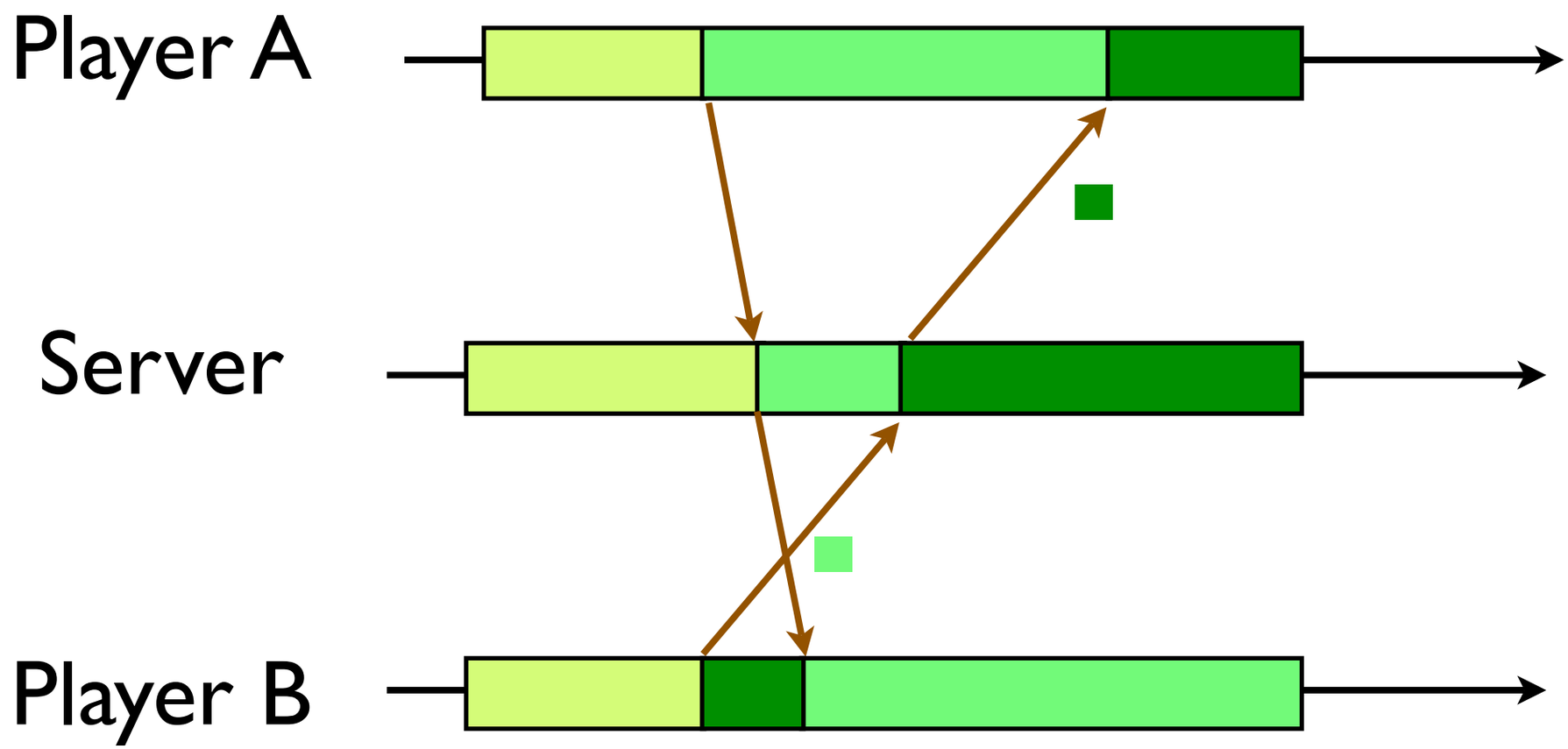


Problem: responsiveness is bounded by the slowest player

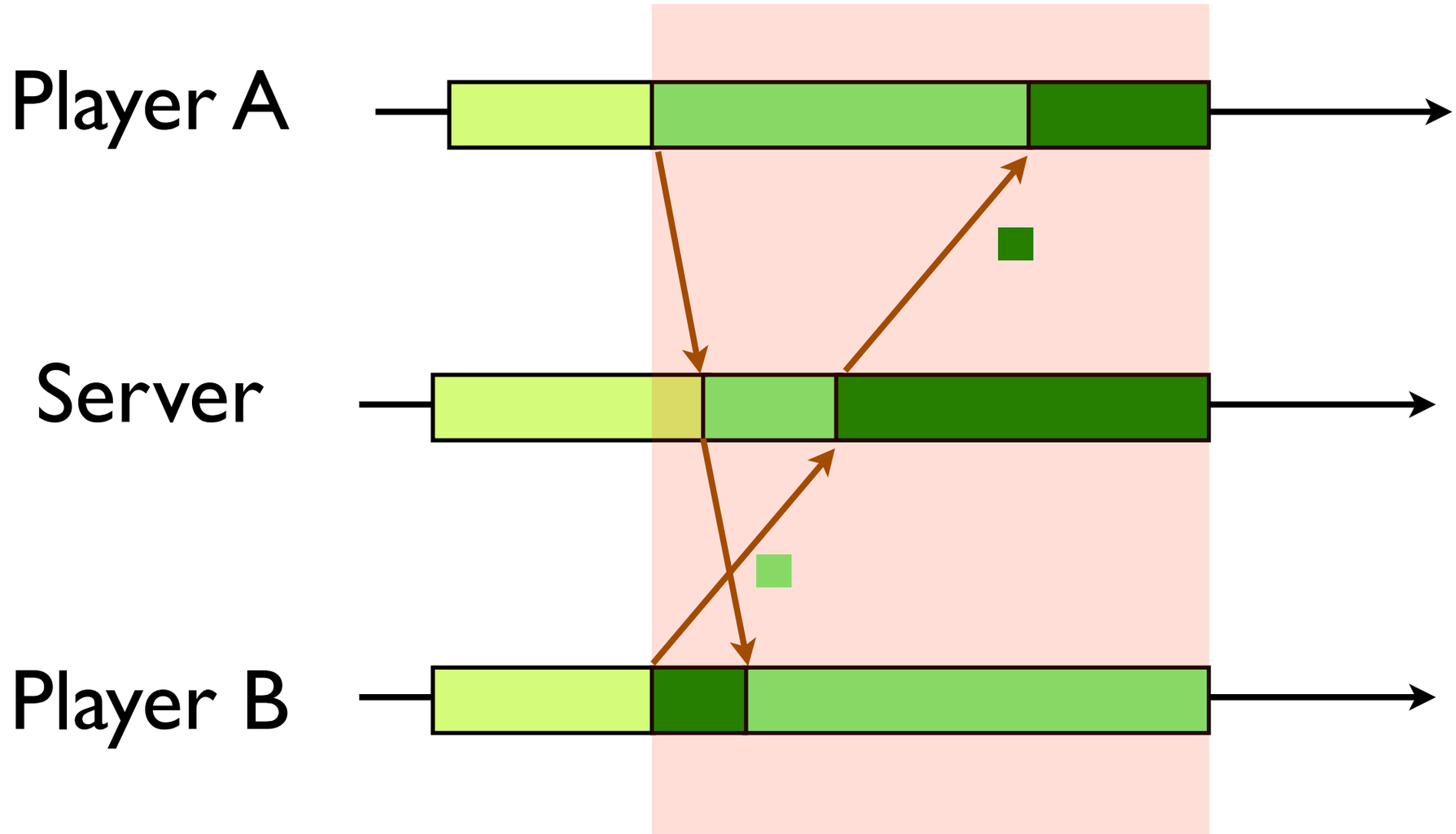


Improving Responsiveness

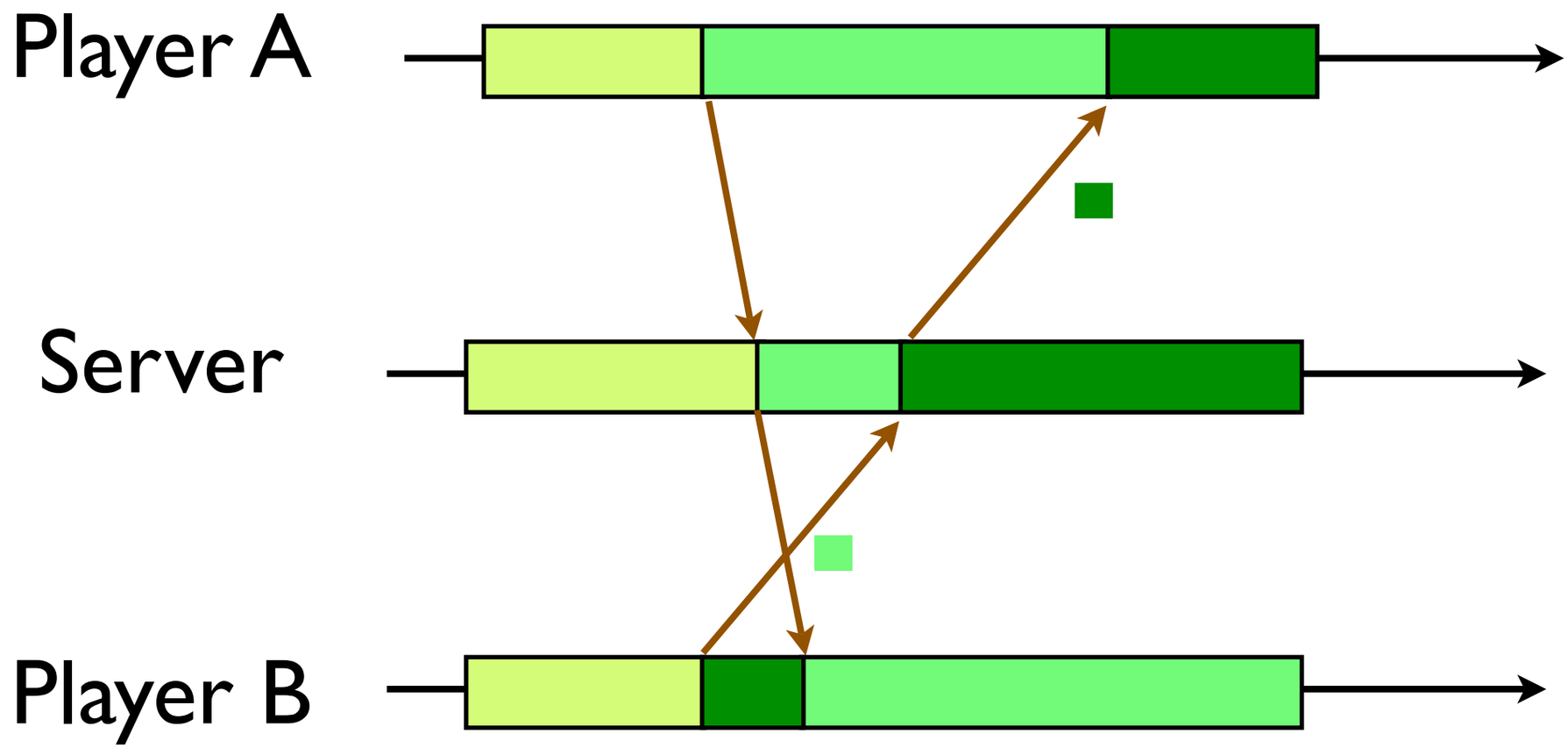
Try: Short circuiting -- execute action immediately locally. But inconsistency arises.



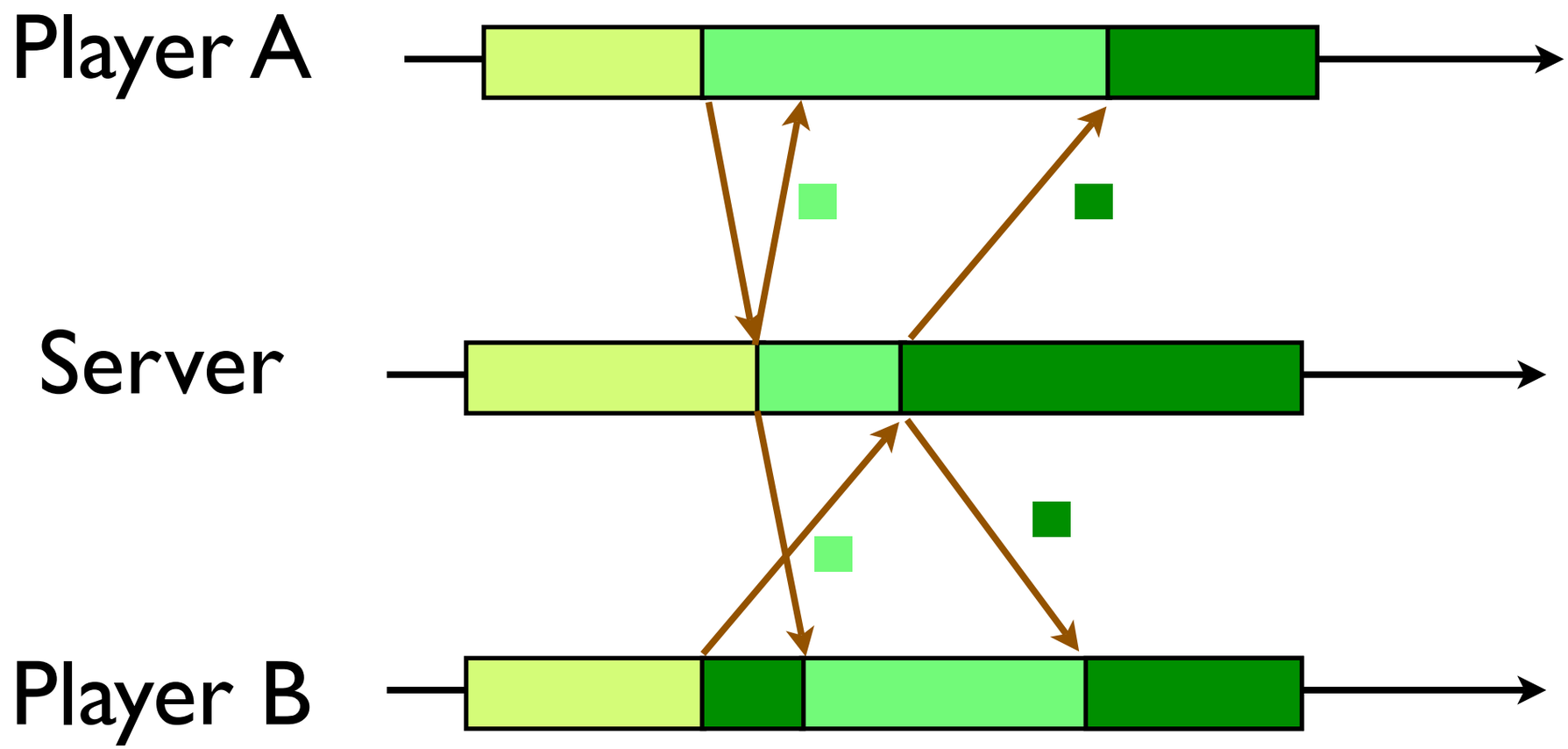
Try: Short circuiting -- execute action immediately locally. But inconsistency arises.



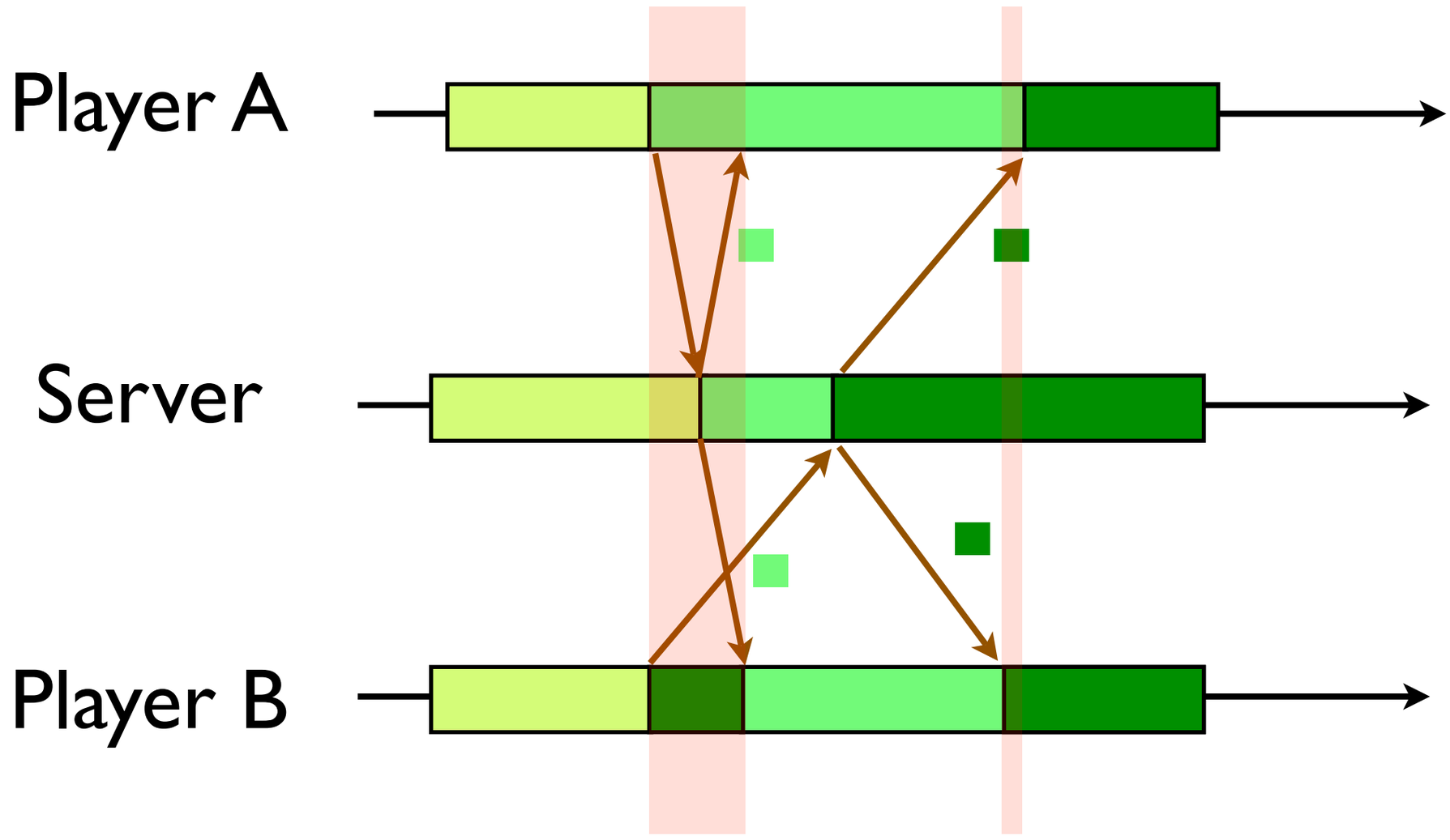
Recall: server is the authority and maintains the correct states.



We can fixed the inconsistency later using the states from the server.

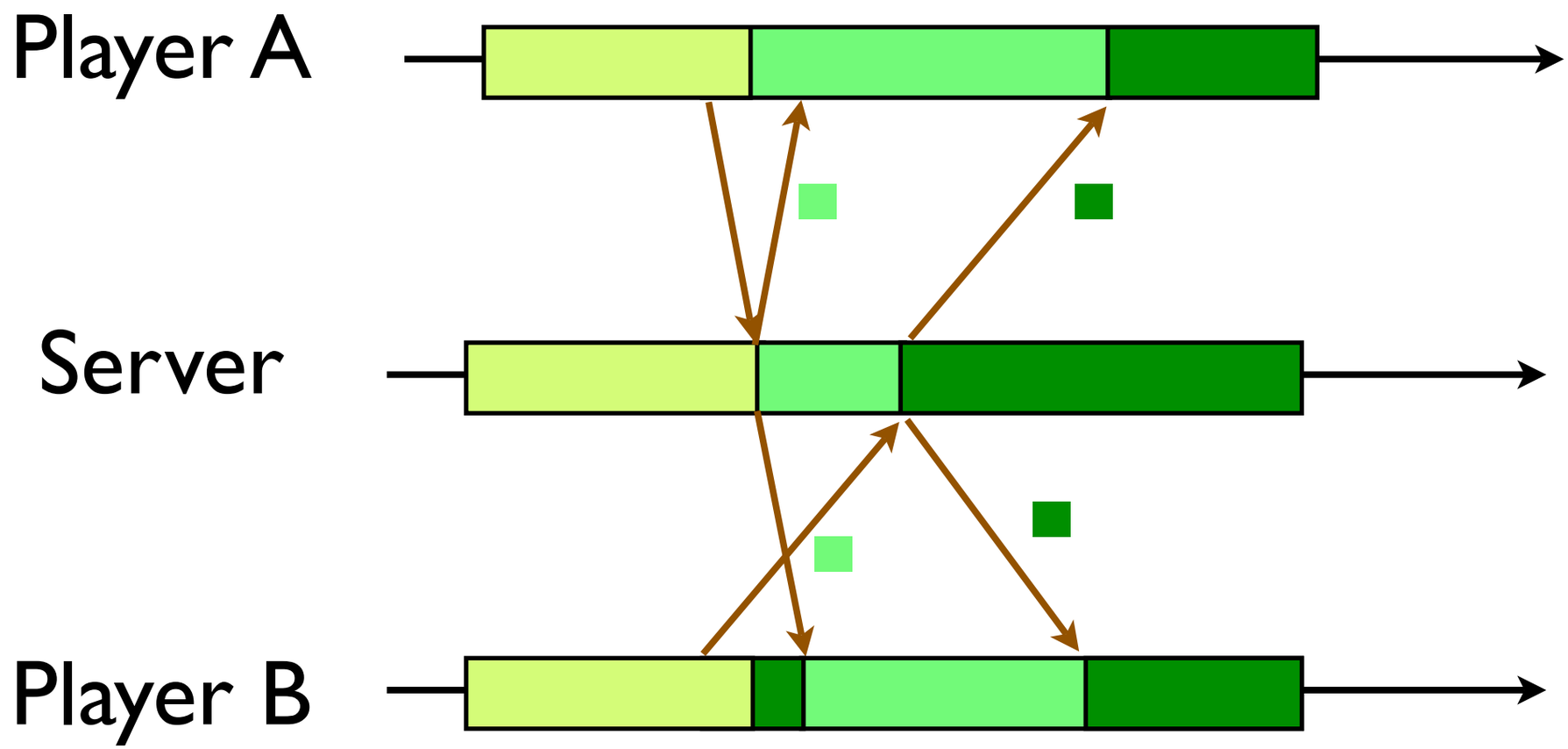


We can fixed the inconsistency later using the states from the server.

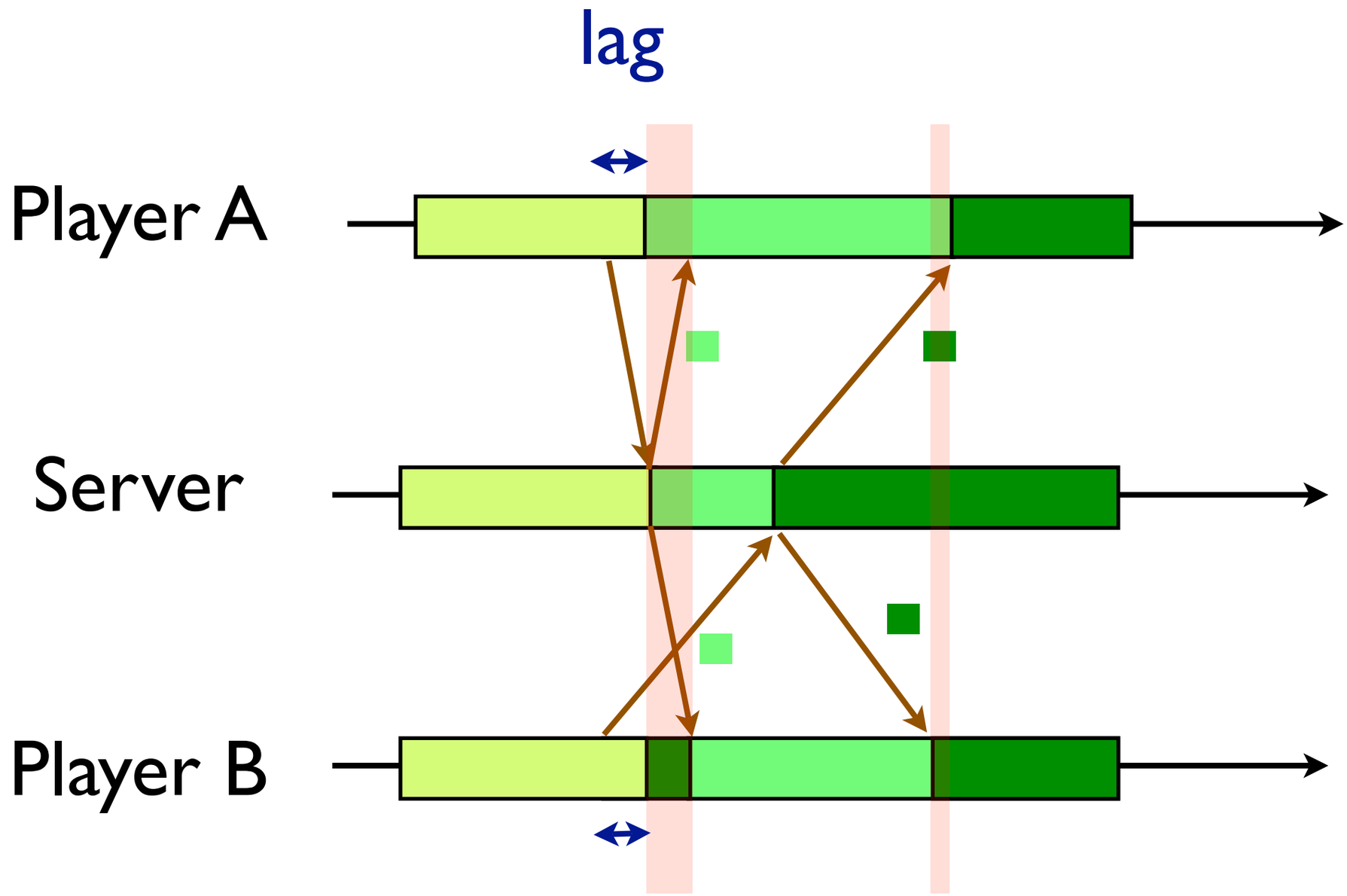


Inconsistent

Slight delay in response might be OK. **Idea:** introduce local lag -- wait for some time t before update states.



Effectively we are trading off responsiveness with consistency.



Inconsistent

Trade-off responsiveness with consistency

Do first, fix later (optimistic)

How responsive should the game be?

How consistent should the game be?

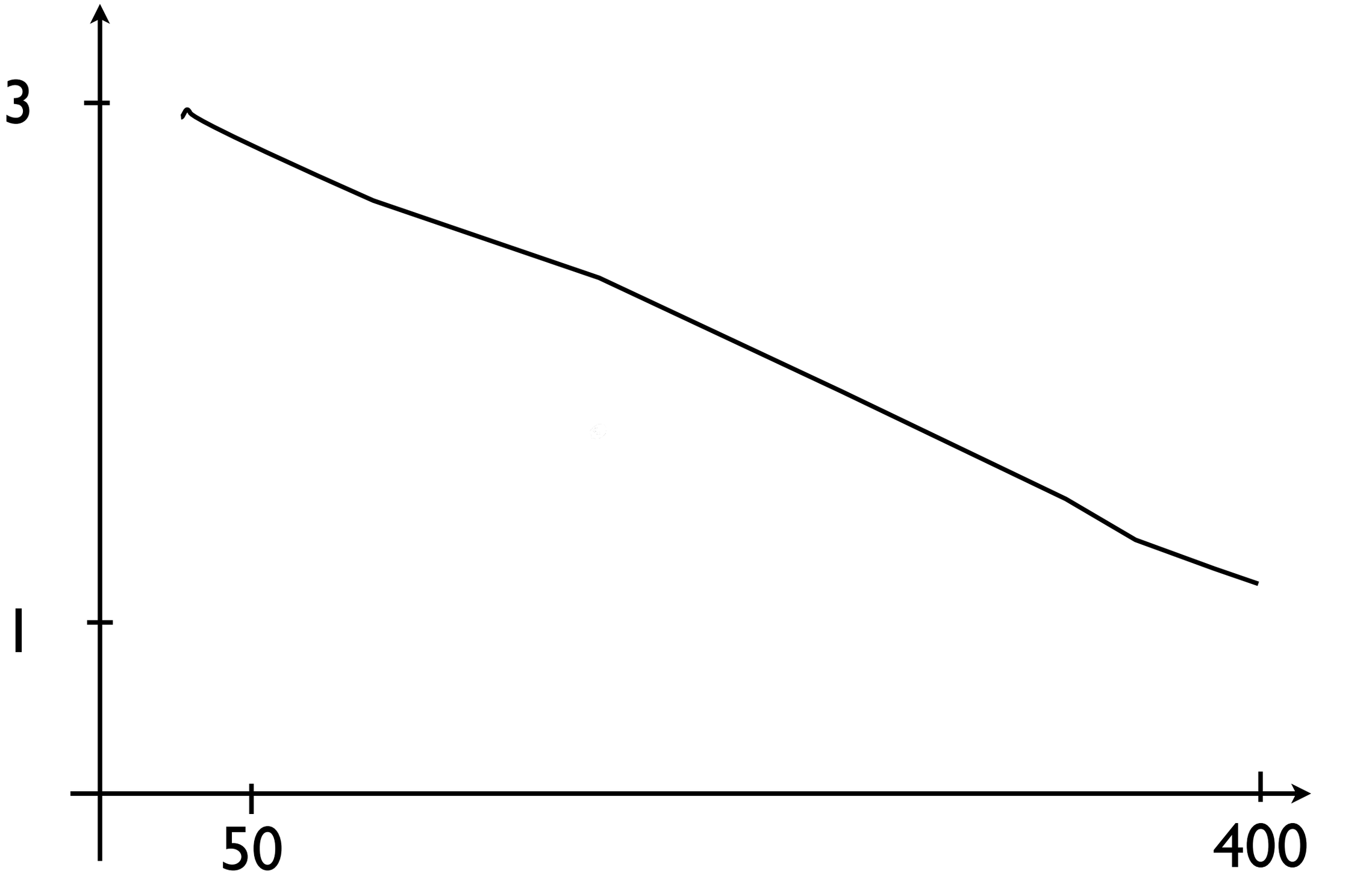
How to “fix later” ?

User Studies: Effects of Network on Games

Goal: How much network latency is tolerable?

Method: Analyze game servers log for Quake III Arena

Frags/minutes



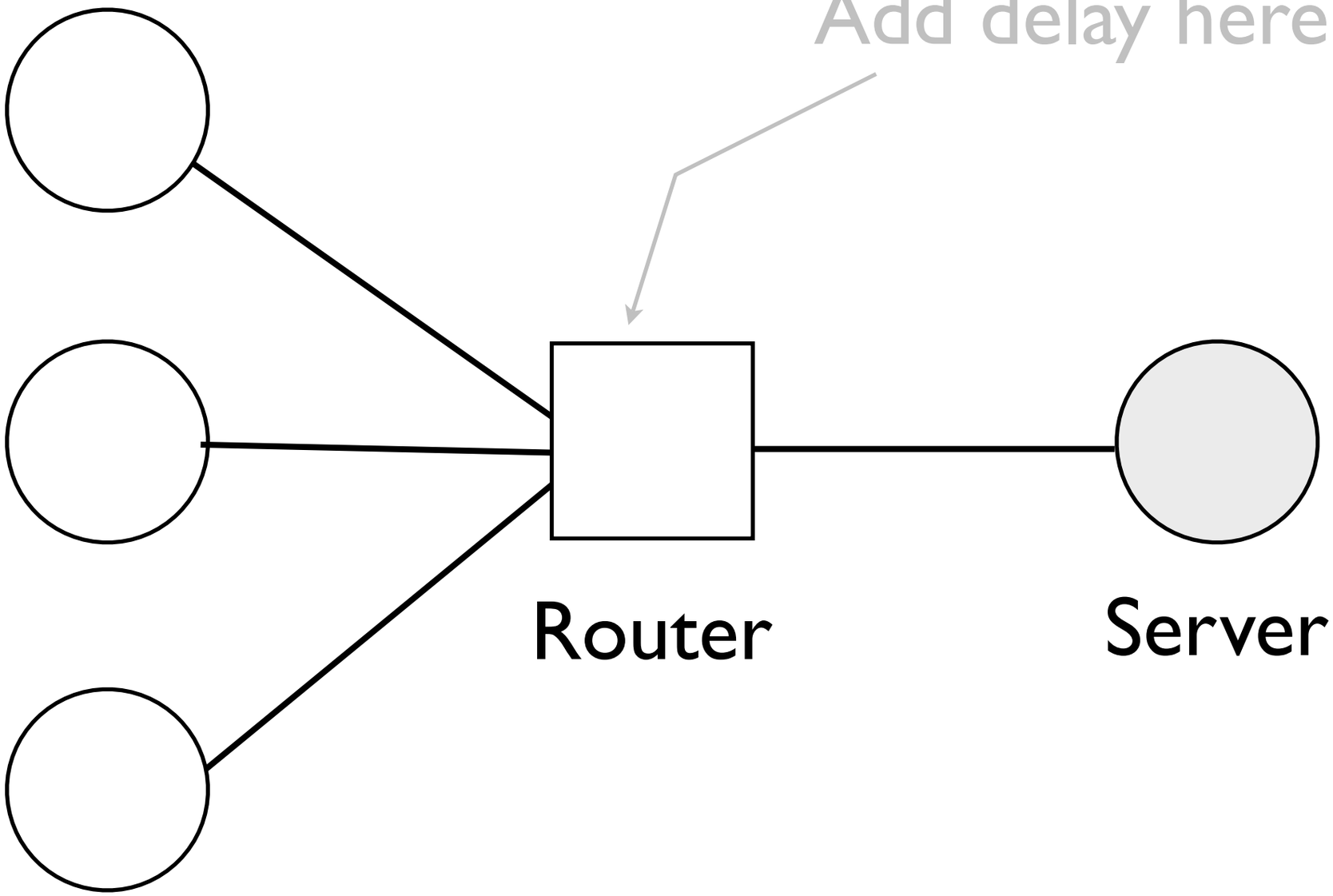
not the actual graph

Yes, latency does affect playability..

Question: what's the annoyance threshold?

Method: User studies using Unreal Tournament 2003

Clients



Game Activity:

move and shoot

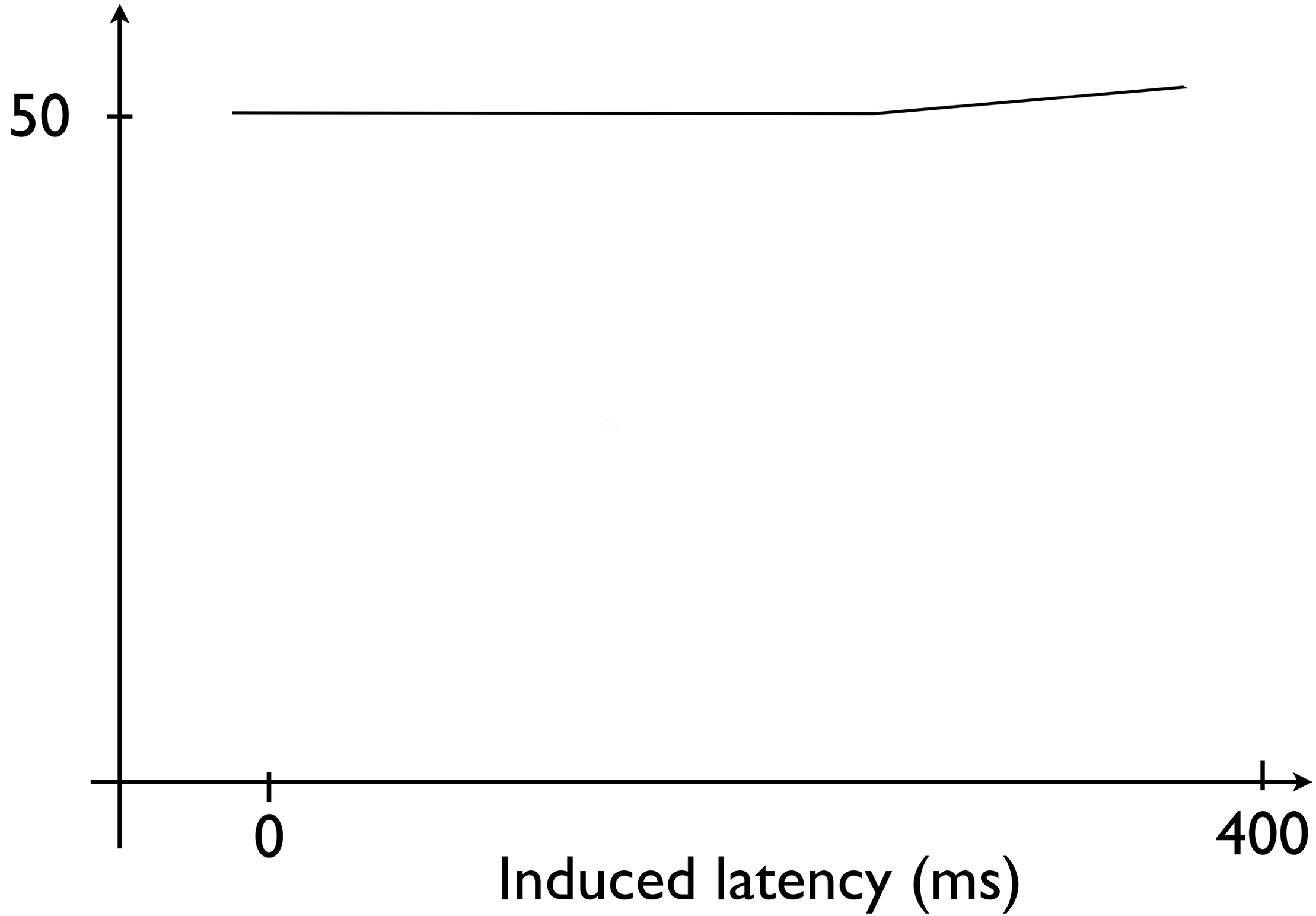
Movement Test:

Construct obstacle course


1
0
0
100
50
5

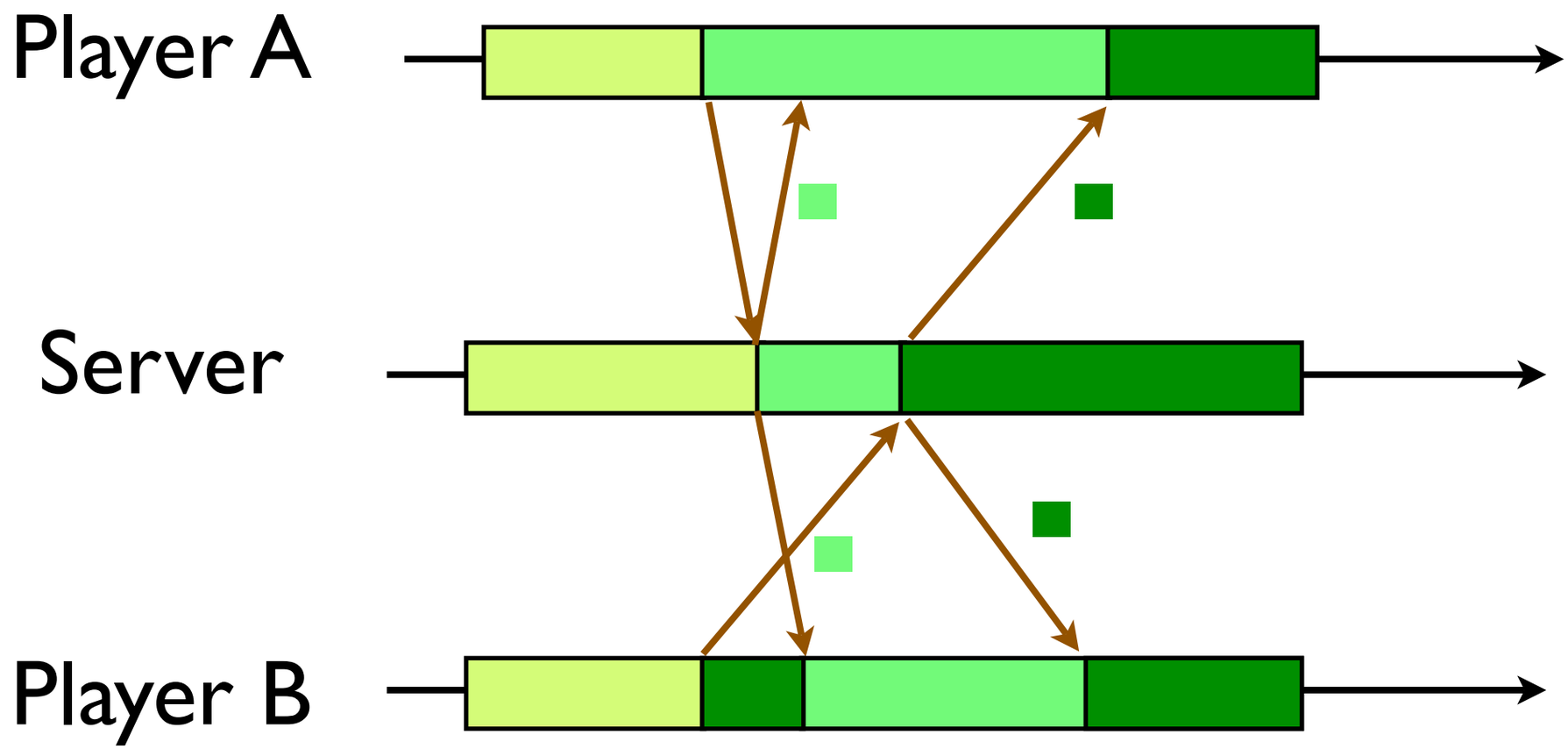
Over 200 users

Time to complete course (s)



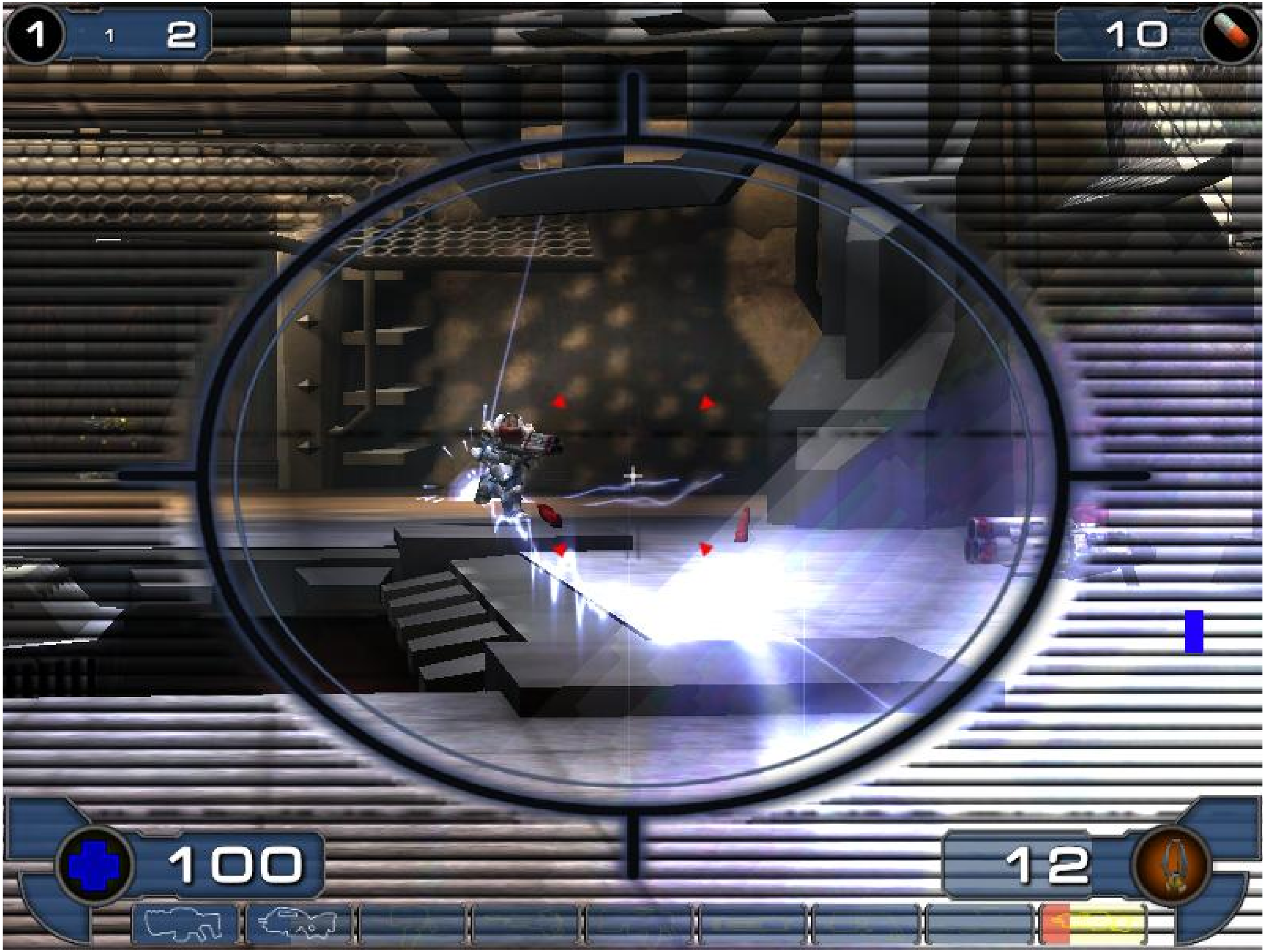
not the actual graph

Perhaps UT 2003 is using short circuiting for movement?

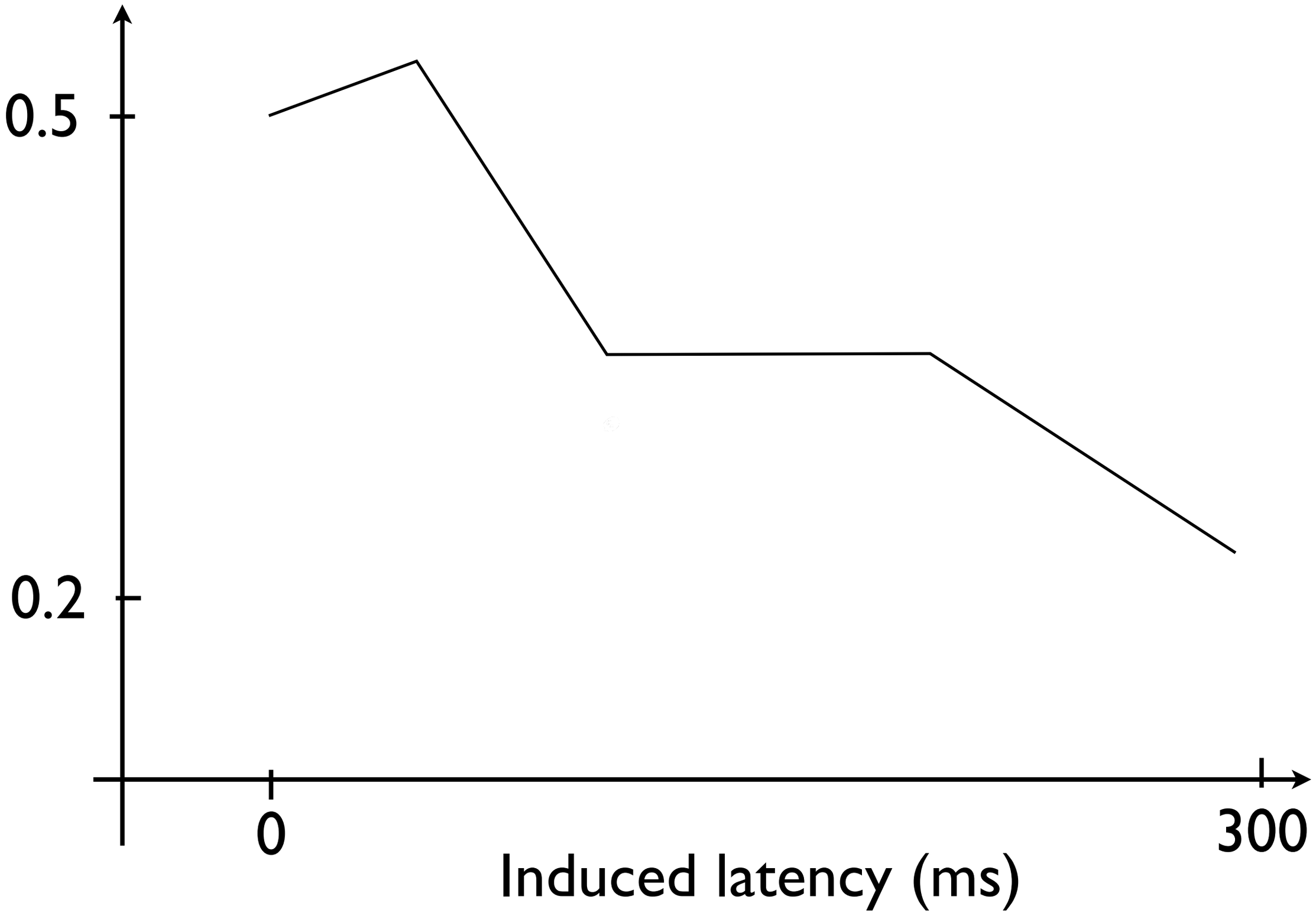


Shooting Test:

2 players shooting at each other using precision weapon


1
1
2
10
100
12

Hit Fraction



not the actual graph

“latency as low as **100** ms were noticeable and latencies around **200** ms were annoying”

Read the paper for complete results.

Other conclusion: loss rate up to 5% has no measurable effects.

How responsive should the game be?

How consistent should the game be?

How to “fix later” ?

Are we done?

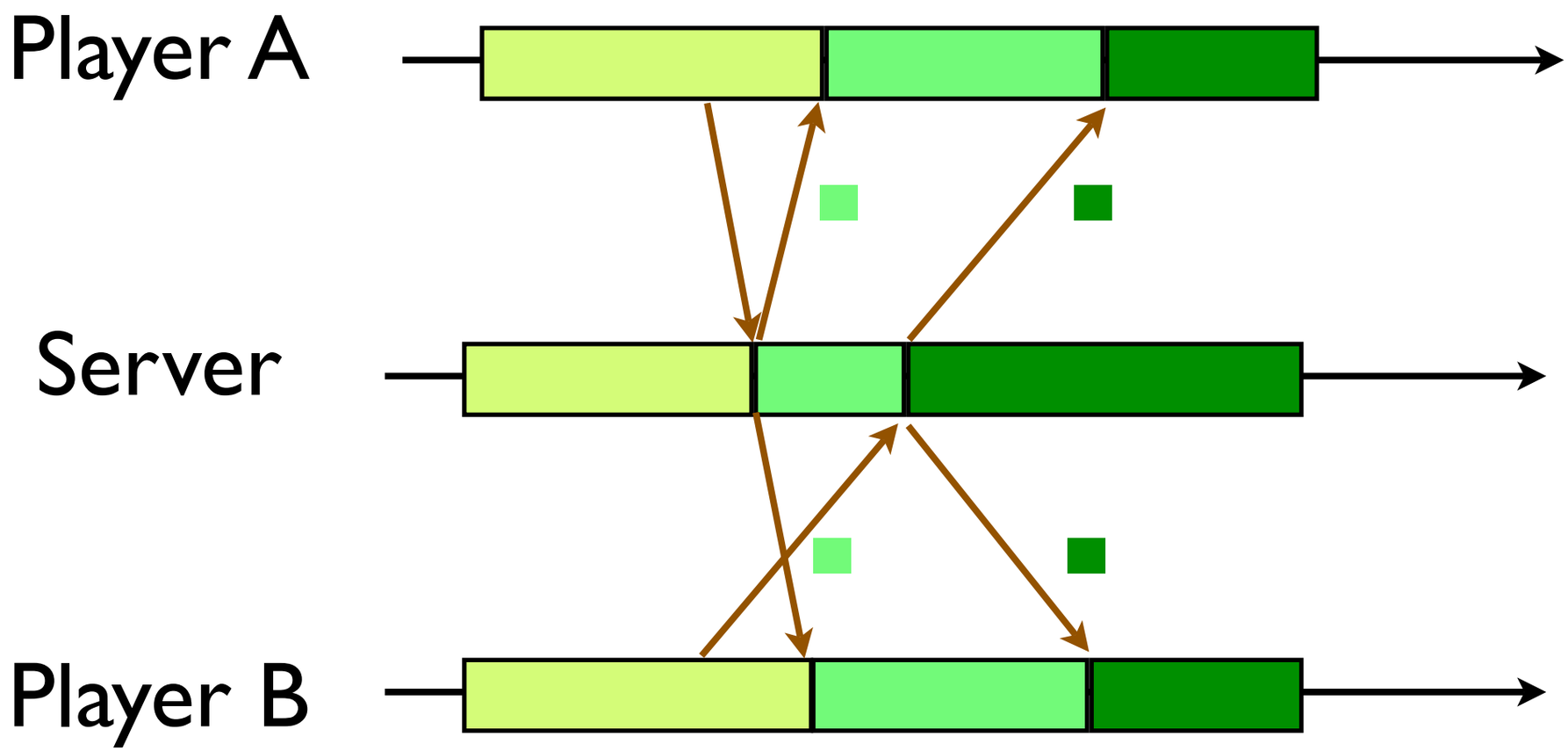
Method: User Studies using Warcraft III

Game Activity:

build, explore, fight!

Finding: Players with larger delays see exactly the same events as players with smaller delays, only at a later time.

Possible communication architecture?



Finding: Latency of up to 800 ms has negligible effect on the outcome of Warcraft III.

Finding: Latency of up to 500 ms can be compensate by the players

Finding: Latencies between 500 and 800 ms degrades game experience.

Finding: Players that micro-manage units in combat feel the latency more than players who don't.

Strategy is more important in RTS games, not reaction time.

Q: How responsive and consistent should the game be?

A: Depends on the characteristics of game.

Important:

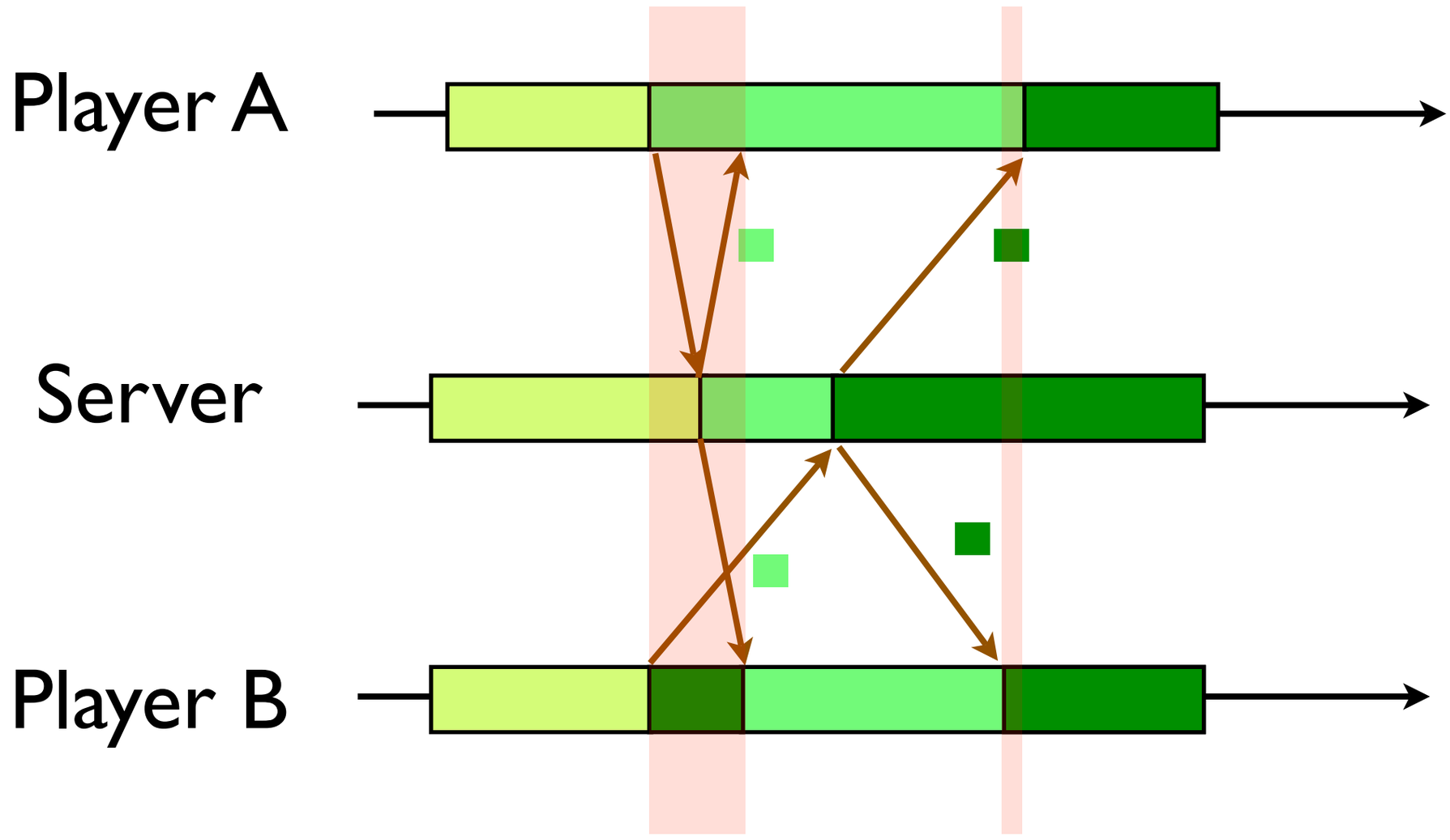
understand user requirements

How responsive should the game be?

How consistent should the game be?

How to “fix later” ?

We can fixed the inconsistency later using the states from the server.



Inconsistent

State: positions

Event: movements

Unreal Tournament's lock-step predictor/corrector algorithm for player's movement

Player

Server

Player

Player moves

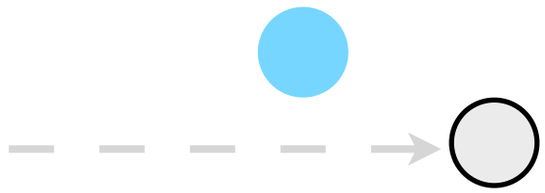
Server

Player

Player updates server

“I am moving east at 5m/s”

Server

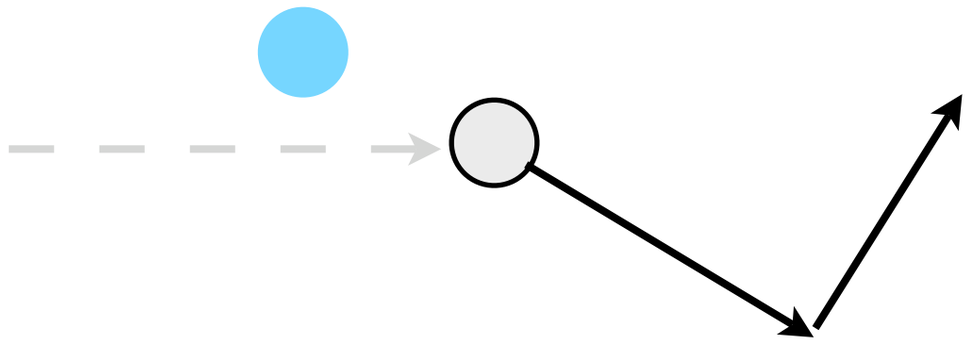


Player

RTT/2 later, server is notified

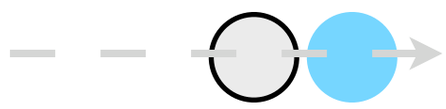
“Player A is moving east at 5m/s”

Server



Player

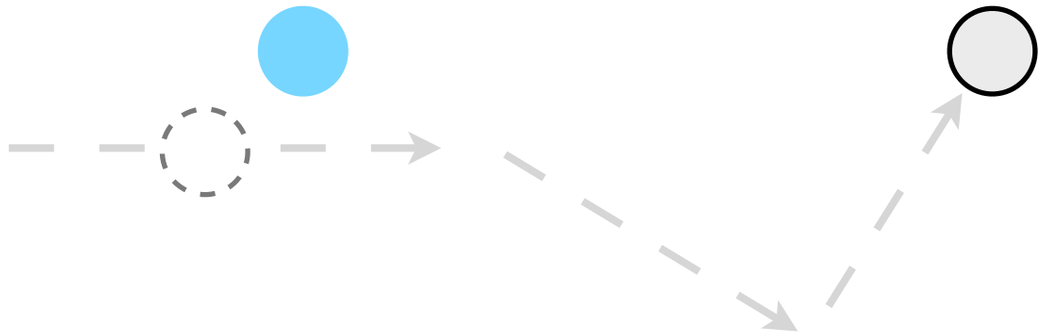
Player might moves again



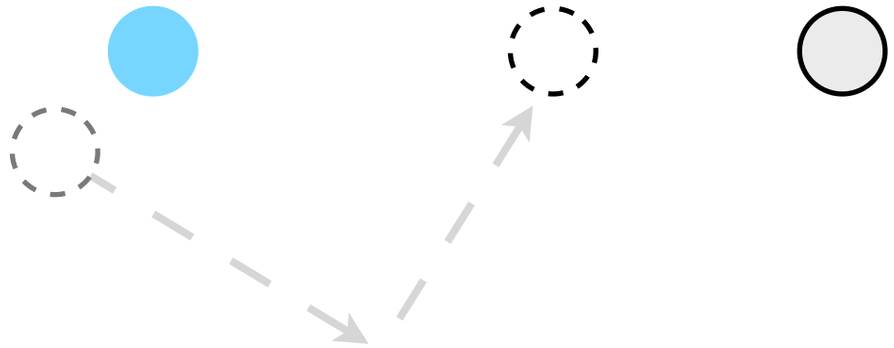
Server

Server simulates player and update player

“You are here at time t”



RTT/2 later, player learns its actual position sometime in the past.



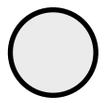
Player re-executes its moves to find its proper position now.

Convergence

If no convergence is used, player updates its position immediately -- in effect teleporting to the correct position, causing visual disruption.

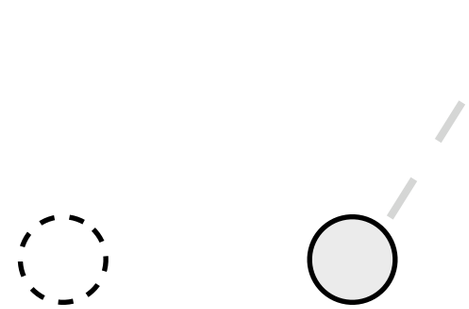
(zero order convergence)

If no convergence is used, player updates its position immediately -- in effect teleporting to the correct position, causing visual disruption.

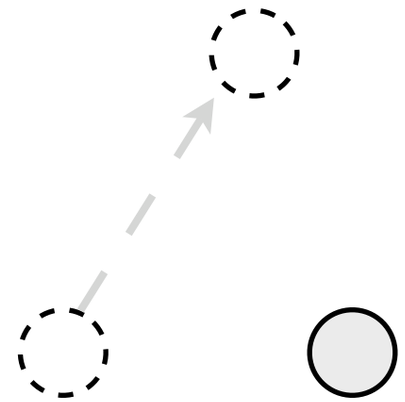


(zero order convergence)

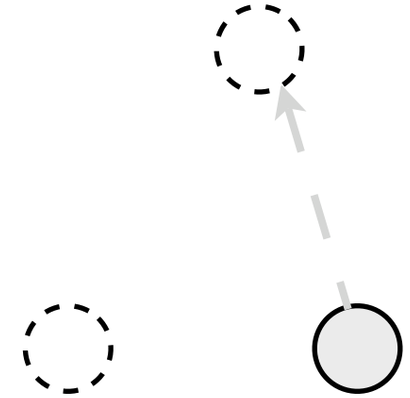
Convergence allows player to move
to the correct position smoothly. First
pick a convergence period *t*, and compute the correct
position after time *t*.



Convergence allows player to move to the correct position smoothly. First compute the correct position after time t.

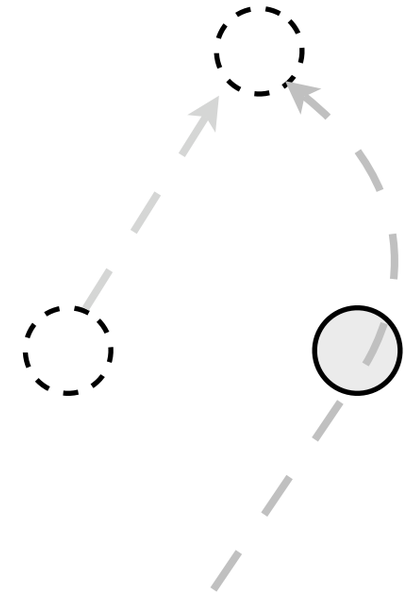


Move to that position in a straight line.

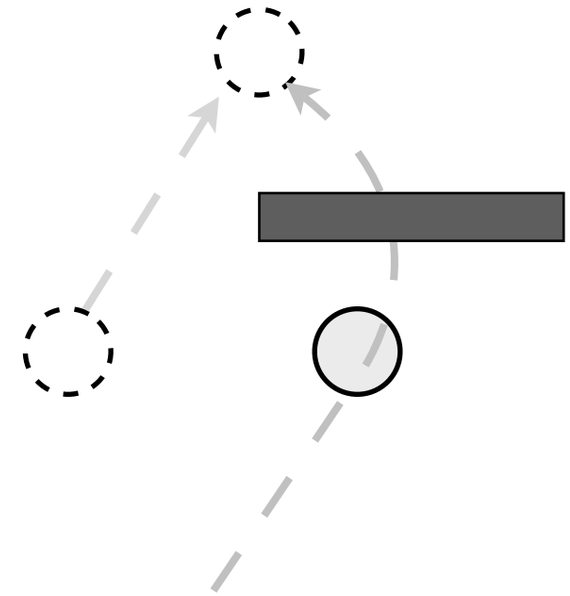


(linear convergence)

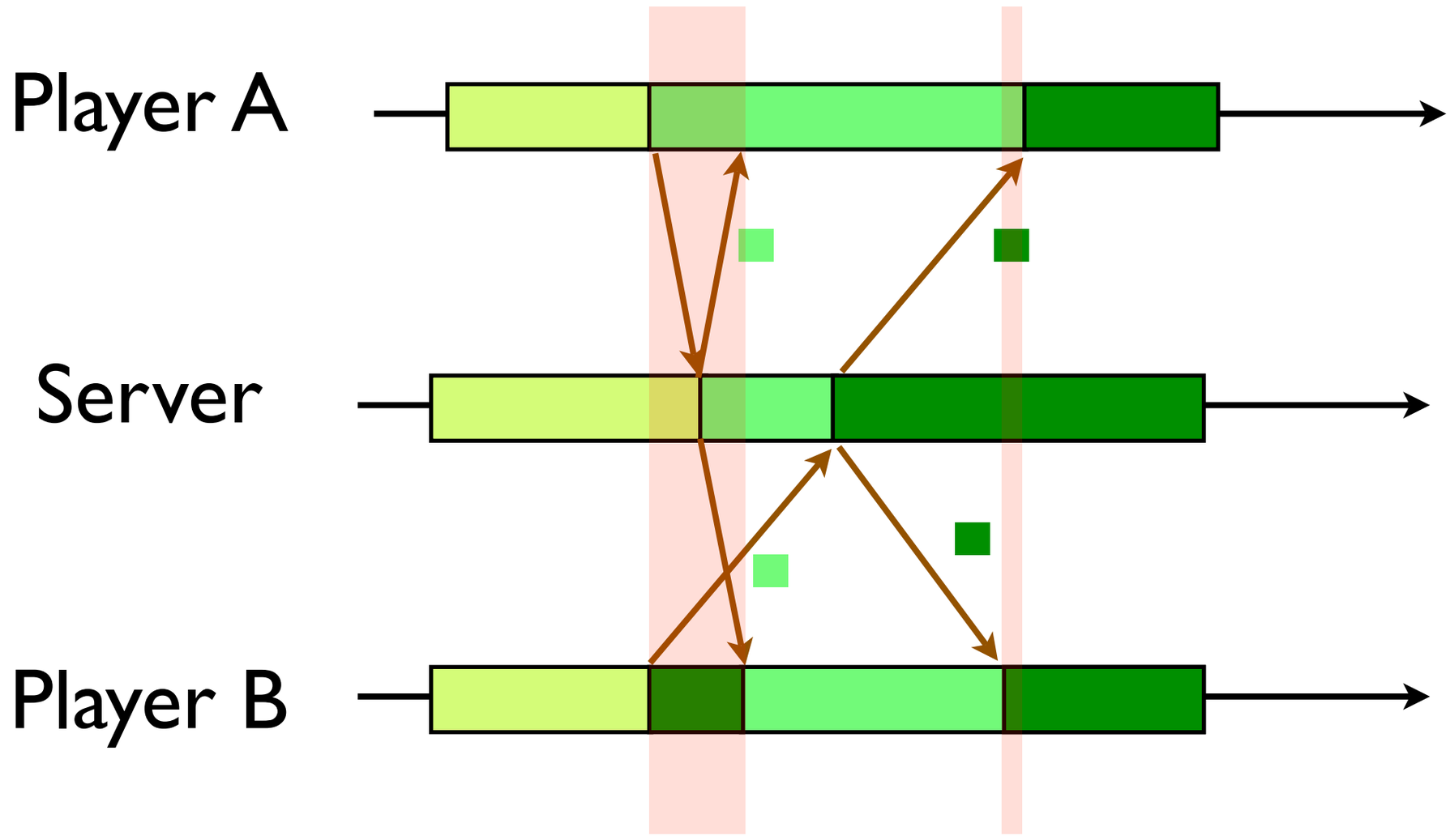
Curve fitting techniques can be used
for smoother curves.



Visual disruption can still occur with convergence.

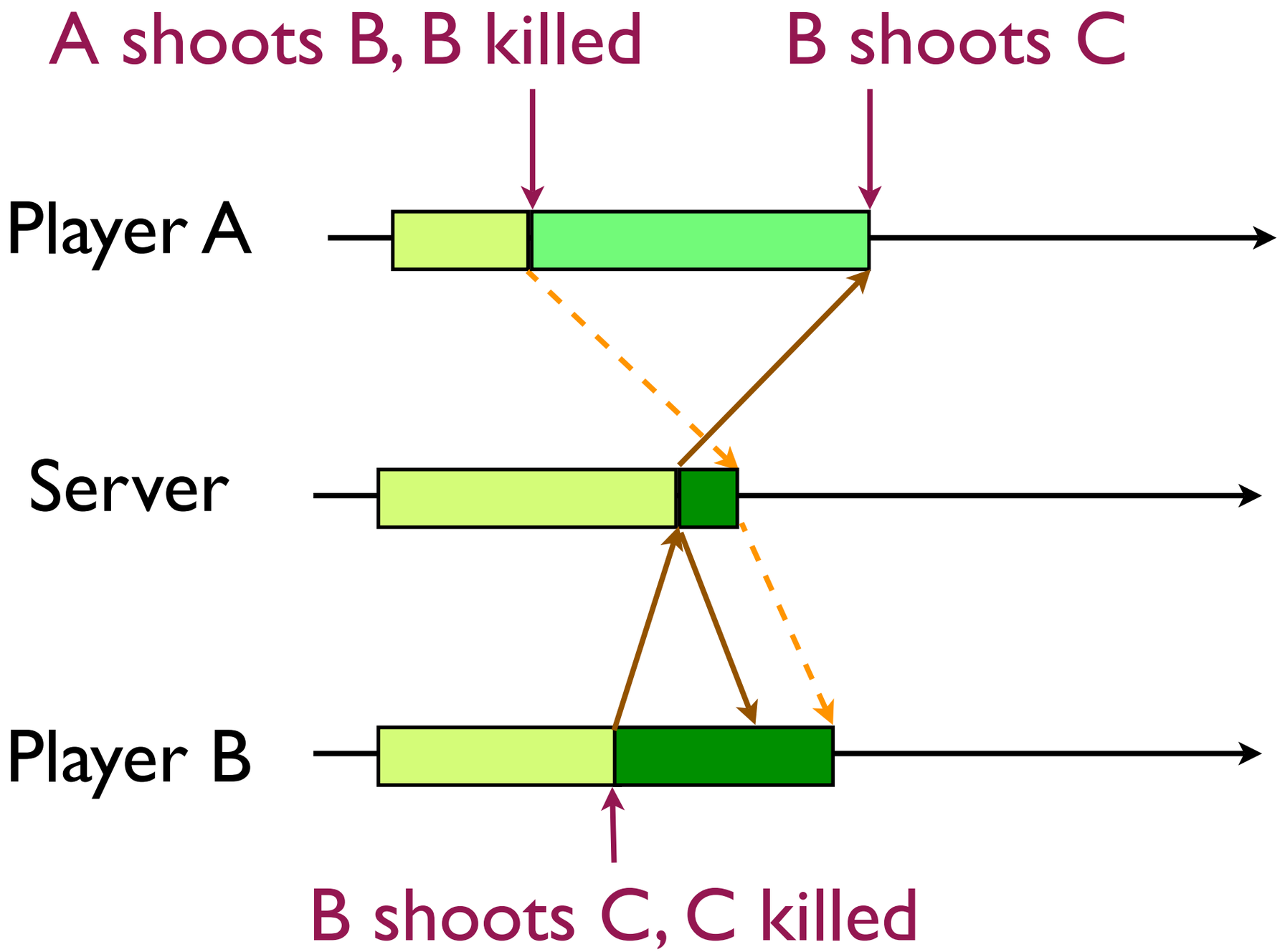


Recall: With short-circuit, we may need to fix inconsistency later using the server states.



Inconsistent

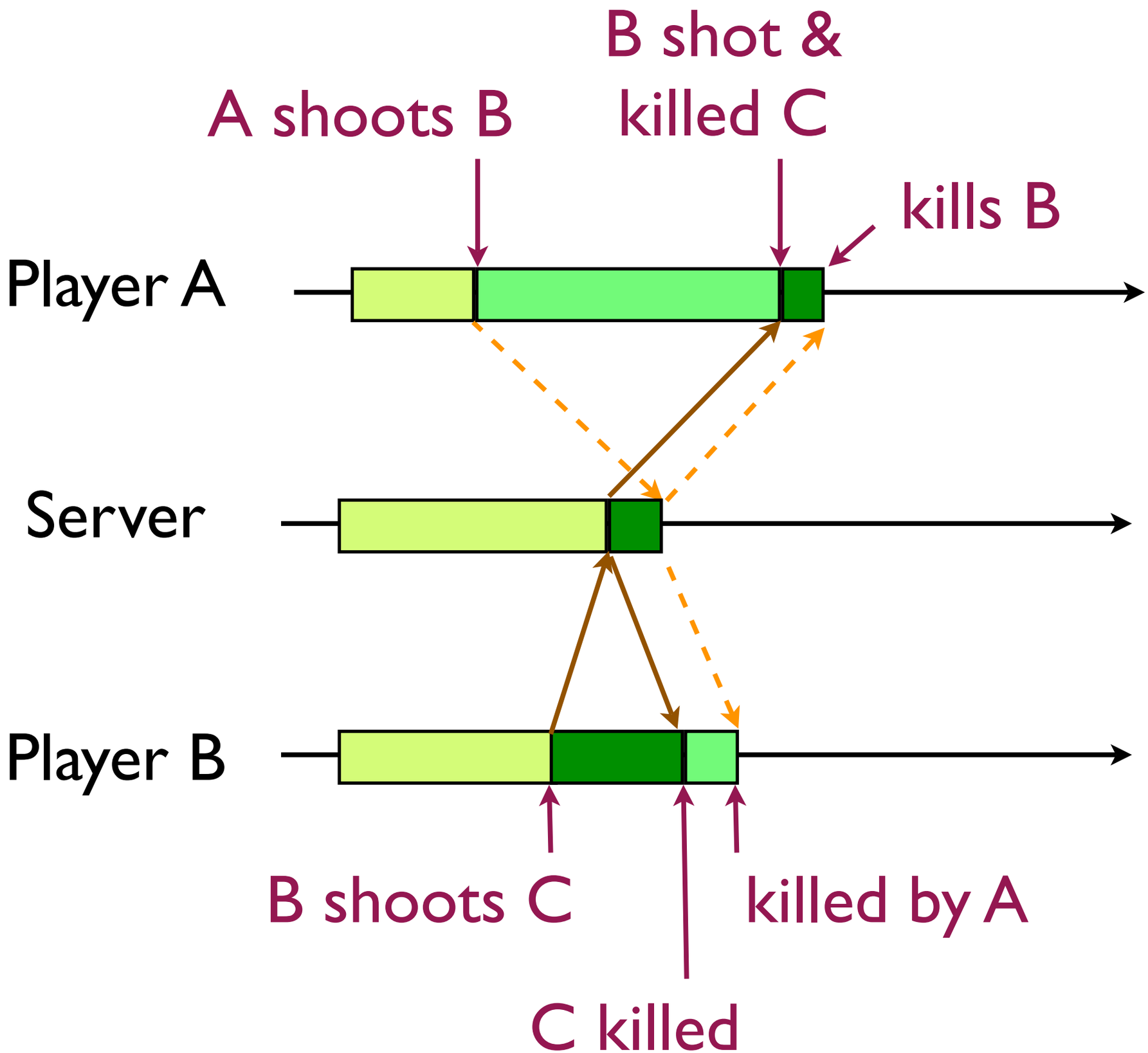
Can we fix all inconsistency?


A shoots B, B killed
B shoots C
Player A
Server
Player B
B shoots C, C killed

“A dead man that shoots”

Short-circuiting not suitable for all cases.

Besides, important events like “hit” should be decided by the server.

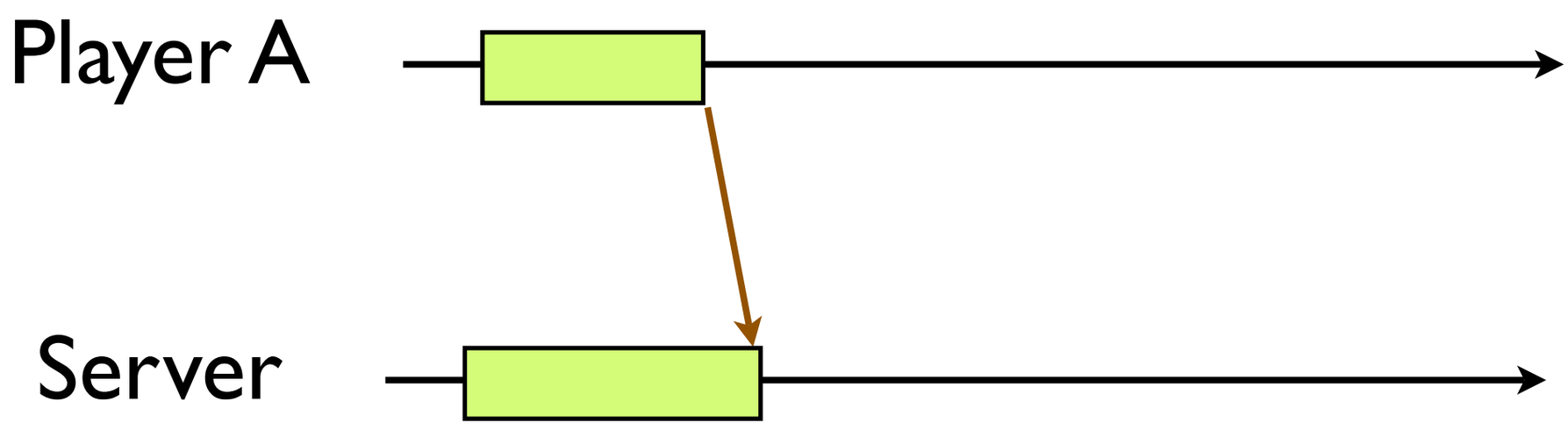

B shot &
killed C
A shoots B
kills B
Player A
Server
Player B
B shoots C
killed by A
C killed

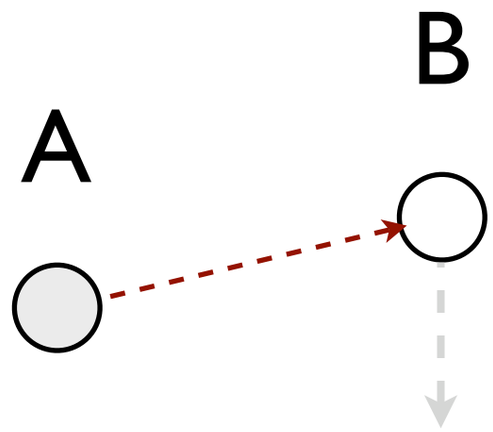
Games can use audio/visual tricks to hide the latency between shooting and hitting.

New Question: how can the server knows if A hits B?

Suppose player A aims and shoots at B. When A's message reaches the server, B already moved away.

Did A hit B?



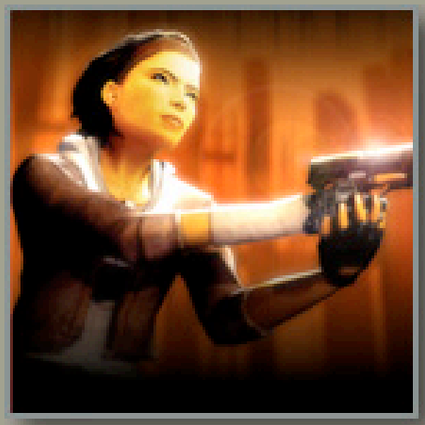


Player

Server

RTT/2 later, server is notified

Lag Compensation
or
Time Warp



Half-Life® 2: Episode One

Half-Life 2: Episode One The first in a trilogy of episodic games, Episode One reveals the aftermath of Half-Life 2 and launches a journey beyond City 17. Episode One does not require Half-Life 2 to play and also includes a first look at Episode Two.

GET HALF-LIFE 2: EPISODE ONE NOW!

Half-Life® 2

Half-Life 2 defines a new benchmark in gaming with startling realism and responsiveness. Powered by Source™ technology, Half-Life 2 features the most sophisticated in-game characters ever witnessed, advanced AI, stunning graphics and physical gameplay.

GET HALF-LIFE 2 NOW!

Counter-Strike™: Source™

Counter-Strike: Source blends Counter-Strike's award-winning teamplay action with the advanced technology of Source™ technology. Featuring state of the art graphics, all new sounds, and introducing physics, Counter-Strike: Source is a must-have for every action gamer.

GET COUNTER-STRIKE:SOURCE NOW!

Half-Life: Source

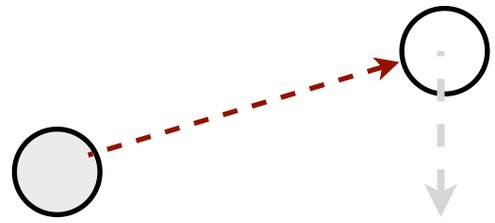
Winner of over 50 Game of the Year awards, Half-Life set new standards for action games when it was released in 1998. Half-Life: Source is a digitally remastered version of the critically acclaimed and best selling PC game, enhanced via Source technology to include physics simulation, enhanced effects, and more.

GET HALF-LIFE:SOURCE NOW!

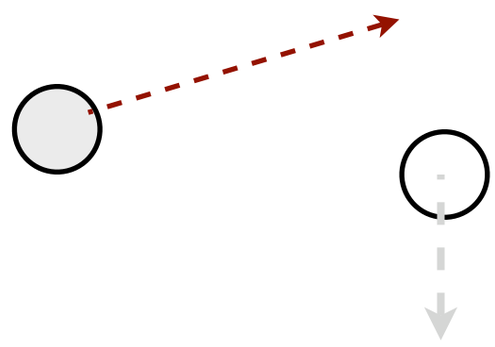
Server estimates the latency between itself and Player A.

Let the latency be t.

Server “rewind” to t
seconds ago.



Server
(now - t)



Server
(now)

Check if hit or miss.

Play forward to now.

http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

Responsive

Consistent

Cheat-Free

Fair

Scalable

Efficient

Robust

Simple

- Permissible client-server architecture is used in Unreal Tournament, and is described by [McCo03]. The article also mentioned the responsiveness issue and described how the client uses short-circuiting for movement command to improve responsiveness in Unreal Tournament.
- Local lag was introduced by [Diot99] in the form of bucket synchronization and in the context of peer-to-peer architecture (we will cover this later in class). The term "local lag" and the idea to adapt the lag was introduced by [Mauv04].
- Short circuiting with immediate feedback was mentioned by [Smed06], Section 9.1.1.
- Time delay is mentioned [Armi06], Section 6.3.1.
- See [Armi06], Section 7.1 for a summary of user studies and results.
- Papers on the Unreal Tournament and Warcraft III studies can be found on the web site http://web.cs.wpi.edu/~claypool/mqp/ut2003/ and http://web.cs.wpi.edu/~claypool/papers/war3/. Screenshot of Unreal Tournament taken from the same site.
- Unreal Tournament's networking component is described in http://unreal.epicgames.com/Network.htm.
- Convergence is described by [Smed06] in Section 9.3.2 in the context of dead reckoning.
- The "dead man that shoots" example was mentioned by [Mauv00] in the context of fully distributed games.
- Lag Compensation techniques used in Half Life in [Armi06] Section 6.3.2 and also in great details online at http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking.

References

[McCo03] A. McCoy, D. Delaney, and T. Ward, "*Game-State Fidelity Across Distributed Interactive Games*", Crossroads vol. 9, no. 4 (Jun. 2003), 4-9

[Diot99] C. Diot and L. Gautier, "*A distributed architecture for multiplayer interactive applications on the internet*," IEEE Networks magazine, vol. 13, no. 4, July/August 1999

[Mauv04] M. Mauve, J. Vogel, V. Hilt, and W. Eelsberg, "*Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications*," IEEE Transactions on Multimedia, vol. 6, no. 1, pp. 45-57, 2004.

[Smed06] J. Smed and H Hakonen, "*Algorithms and Networking for Computer Games*", Wiley, July 2006.

[Armi06] G. Armitage, M. Claypool and P. Branch, "*Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*," Wiley, June 2006.

[Mauv00] M. Mauve, "*How to Keep a Dead Man from Shooting*." In Proc of the 7th intl Workshop on interactive Distributed Multimedia Systems and Telecommunication Services, 2000.