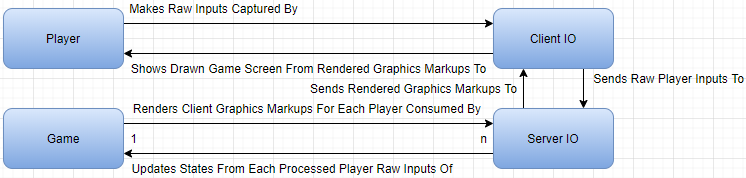
The following image briefly outlines the core structure of this whole idea, which is based on the idea of applying purely server-side rendering on games:



The following's the general flow of games using this architecture(all these happen per frame):

1. The players start running the game with the client IO

2. The players setup input configurations(keyboard mapping, mouse sensitivity, mouse acceleration, etc), graphics configurations(resolution, fps, gamma, etc), client configurations(player name, player skin, other preferences not impacting gameplay, etc), and anything that only the players can have information of

3. The players connect to servers

4. The players send all those configurations and settings to the servers(those details will be sent again if players changed them during the game within the same servers)

5. The players makes raw inputs as they play the game

6. The client IO captures those raw player inputs and sends them to the server IO

7. The server IO combines those raw player inputs and the player input configurations for each player to form commands that the game can understand

8. Those game commands generated by all players in the server will update the current game state set

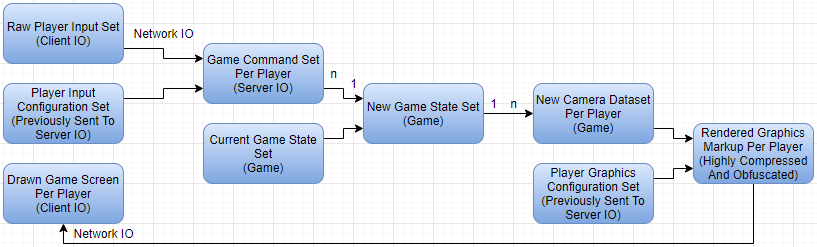
9. The game polls the updated current game state set to form the new camera data for each player

10. The game combines the camera data with the player graphics configurations to generate the rendered graphics markups which are highly compressed and obfuscated and have the least amount of game state information possible

11. The server IO captures the rendered graphics markups and send them to the client IO of each player

12. The client IO draws the rendered graphics markups on the game screen visible by each player

The aforementioned flow can also be represented this way:



The advantages of this architecture at least include the following:

1. The game requirements on the client side can be a lot lower than the traditional architecture, as now all the client side does is sending the captured raw player inputs to the server side, and draws the received rendered graphics markup on the game screen visible by each player

2. Cheating will become next to impossible, as all cheats are based on game information, and even the state of the art [machine vision](https://en.wikipedia.org/wiki/Machine_vision) still can't retrieve all the information needed for cheating within a frame(even if it just needs 0.5 seconds to do so, it's already too late in the case of professional FPS E-Sports, not to mention that the rendered graphics markup can change per frame, making machine vision even harder to work well there), and it'd be a epoch-making breakthrough on machine vision if the cheats can indeed generate the correct raw player inputs per frame(especially when the rendered graphics markups are highly obfuscated), which is definitely doing way more good than harm to the mankind, so games using this architecture can actually help pushing the machine vision researches.

3. Game piracy and plagiarisms will become a lot more costly and difficult, as the majority of the game contents and files never leave the servers, meaning that those servers will have to be hacked first before those pirates can crack those games, and hacking a server with the very top-notch security(perhaps monitored by network and server security experts as well) is a very serious business that not many will even have a chance

The disadvantages of this architecture at least include the following:

1. The game requirements on the server side will become ridiculous - perhaps a supercomputer, computer cluster, or a computer cloud will be needed for each server, and I just don't know how it'll even be feasible for MMO to use this architecture in the foreseeable future

2. The network traffic in this architecture will be absurdly high, because all players are sending raw input to the same server, which sends back the rendered graphics markup to each player(even though it's already highly compressed), all happening per frame, meaning that this can become serious connection issues with servers having low capacity and/or players with low connection speed/limited network data usage

3. The maintenance cost of the games on the business side will be a lot higher, because the servers need to be much, much more powerful than those running games not using this architecture

**Clearly, the advantages from this architecture will be unprecedented if the architecture itself can ever be realized, while its disadvantages are all hardware limitations that will become less and less significant, and will eventually becomes trivial.**

**So while this architecture won't be the reality in the foreseeable future(at least several years from now), I still believe that it'll be the distant future(probably in terms of decades).**

If this architecture becomes the practical mainstream, the following will be at least some of the implications:

1. The direct one time price of the games, and also the indirect one(the need to upgrade the client machine to play those games) will be noticeably lower, as the games are much less demanding on the client side(drawing an already rendered graphics markup is generally a much, much easier, simpler and smaller task than generating that markup itself, and the client side hosts almost no game objects so the memory required will also be a lot lower)

2. The periodic subscription fee will exist in more and more games, and those already having such fee will likely increase the fee, in order to compensate for the increasing game maintenance cost from upgraded servers(these maintenance cost increments will eventually be cancelled out by hardware improvements causing the same hardware to become cheaper and cheaper)

3. The focus of companies previously making high end client CPU, GPU, RAM and motherboard will gradually shift their business into making server counterparts, because the demands of high end hardware will be relatively smaller and smaller on the client side, but will be relatively larger and larger on the server side

In the case of highly competitive E-Sports, the server can even implement some kind of fuzzy logic, which is fine-tuned with a [deep learning](https://en.wikipedia.org/wiki/Deep_learning) AI, to help report suspicious raw player input sets with a rating on how suspicious it is, which can be further broken down to more detailed components on why they're that suspicious.

**This can only be done effectively and efficiently if the server has direct access to the raw player input set, which is one of the cornerstones of this very architecture.**

Combining this with traditional anti cheat measures, like having a server with the highest security level, an admin to monitor each player in the server(now with the aid of the AI reporting suspicious raw player input sets), another admin for each team/side to monitor player activities, a camera for each player, and thoroughly inspected player hardware, it'll not only make cheating next to impossible in major LAN events(also being cut off from external connections), but also so obviously infeasible and unrealistic that almost everyone will agree that cheating is indeed nearly impossible there, thus drastically increasing their confidence on the match fairness.

Of course, games can also use a hybrid model, and this especially applies to multiplayer games also having single player modes.

**If the games support single player, of course the client side needs to have everything(and the piracy/plagiarism issues will be back), it's just that most of them won't be used in multiplayer if this architecture's used.**

**If the games runs on the multiplayer, the hosting server can choose(before hosting the game) whether this architecture's used.**

**Alternatively, players can choose to play single player modes with a server for each player, and those servers are provided by the game company, causing players to be able to play otherwise extremely demanding games with a low-end machine(of course the players will need to apply for the periodic subscriptions to have access of this kind of single player modes).**

This hybrid model, if both technically and economically feasible, is perhaps the best model I can think of.