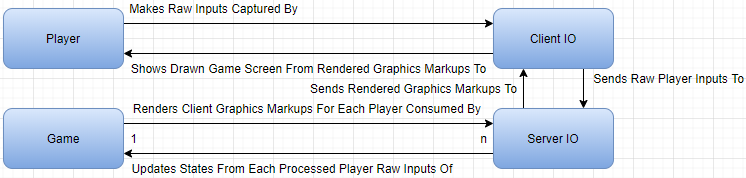
**Descriptions**

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:

**Note that the client side should have next to no game state or data, nor audio/visual assets, as they're supposed to never leave the server side.**

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(like keyboard presses, mouse clicks, etc) as they play the game

6. The client IO captures those raw player inputs and sends them to the server IO(but there's never any game data/state synchronization among them)

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

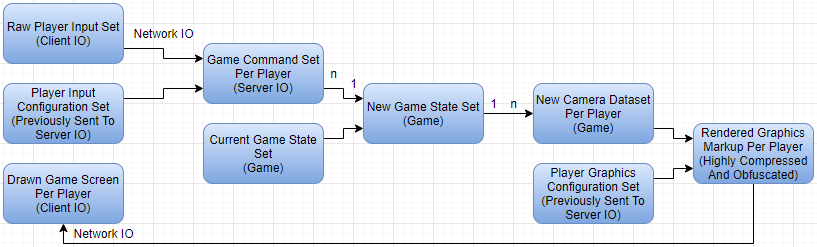
9. The game polls the updated current game state set to form the new camera data for each player

10. The game combines the camera data with the player graphics configurations to generate the rendered graphics markups(with all relevant audio/visual assets used entirely in this step) 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(and nothing else will ever be sent in this direction)

12. The client IO draws the fully rendered graphics markups(without needing nor knowing any audio/visual asset) on the game screen visible by each player

The aforementioned flow can also be represented this way:



**Differences From Cloud Gaming**

Do note that it's different from [cloud gaming](https://en.wikipedia.org/wiki/Cloud_gaming) in the case of multiplayer(although it's effectively the same in the case of single player), because cloud gaming doesn't demand the games to be specifically designed for that, while this architecture does, and the difference means that:

1. In cloud gaming, different players rent different remote machines, each hosting the traditional client side of the game, which communicates with the traditional server side of the game in the same real server that's distinct from those middlemen devices, meaning that **there will be at most 2 round trips per frame**(between the client and the remote machine, and between the remote machine and the real server), so if the remote machines isn't physically close to the real server, and the players aren't physically close to the remote machines, the latency can raise to an absurd level

2. This architecture forces games complying with it to be designed differently from the traditional counterparts right from the start, so **it can install the client version(having minimal contents) directly into the device for each player, which directly communicates with the server side of the game in the same server(which has almost everything), thus removing the need of a remote machine per player as the middleman**, and hence the problems created by it(latency and the setup/maintenance cost from those remote machines)

3. The full cycle of the communications in cloud gaming is the following:

    - The player machines send the raw input commands to the remote machines

    - The remote machines convert those commands into new game states of the client side of the game there

    - The client side of the game in those remote machines synchronize with the server side of the game in the real server

    - The remote machines draw new visuals on their screens and play new audios based on the latest game states on the client side of the game there

    - The remote machines send those audio and visual information to the player machines

    - The player machines redraw those new audios and visuals there

4. The full cycle of the communications of this architecture is the following:

    - The player machines send the raw input commands directly to the real server

    - The real server convert those commands into the new game states of the server side of the game there

    - The real server send new audio and visual information to the player machines based on the involved parts of the latest game states on the server side of the game there

    - The player machines draw those new audios and visuals there

3 + 4 means **the rendering actually happens 2 times in cloud gaming - 1 in the remote machines and 1 in the player machines, while the same happens just once in this architecture - just the player machines directly**, and the redundant rendering in cloud gaming can contribute quite a lot to the end latency experienced by players, so this is another advantage of this architecture over cloud gaming.

**In short, cloud gaming supports games not having cloud gaming in mind(and is thus backward compatible) but can suffer from insane latency and increased business costs(which will be transferred to players), while this architecture only supports games targeting it specifically(and is thus not backward compatible) but removes quite some pains from the remote machine in cloud gaming(this architecture also has some other advantages over cloud gaming, but they’ll be covered in the next section).**

On a side note: If some cloud gaming platforms don't let their players to join servers outside of them, while it'd remove the issue of having 3 entities instead of just 2 in the connection, it'd also be more restrictive than this architecture, because the latter only restricts all players to play the same game using it.

**Advantages**

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(although cloud gaming also has this advantage), as now all the client side does is sending the captured raw player inputs(keyboard presses, mouse clicks, etc) to the server side, and draws the received rendered graphics markup(without using any audio/visual assets in this step and the client side doesn't have any of them anyway) on the game screen visible by each player

2. Cheating will become next to impossible(cloud gaming may or may not have this advantage), 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(cloud gaming may or may not have this advantage), 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

4. Game data and state synchronization should no longer be an issue(while cloud gaming won't have this advantage), because the client side should've nearly no game data and state, meaning that there should be nothing to synchronize with, thus this setup not only removes tons of game data/state integrity troubles and network issues, but also deliberate or accidental exploits like [lag switching](https://makezine.com/2009/02/20/lag-switch-how-some-gamers-che/)(so servers no longer has to kick players with legitimately high latency because those players won't have any advantage anymore, due to the fact that such exploits would just cause the users to become inactive for a very short time per lag in the server, thus they'd be the only ones being under disadvantages)

**Disadvantages**

The disadvantages of this architecture at least include the following:

1. The game requirements and the maintenance cost 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 lead to serious connection issues with servers having low capacity and/or players with low connection speed/limited network data usage

3. The rendered graphics markup needs to be totally lossless in terms of visual qualities on one hand, otherwise it'd be a bane for games needing the state of the art graphics; It also needs to be highly compressed and obfuscated on the other, because the network traffic must be minimized and the markup needs to defend against cheats. These mean it'd be extremely hard to properly implement the rendered graphics markup, let alone without creating new problems

4. The inherent network latency due to the physical distance between the clients and the servers will be even more severe, because now the client has to communicate with the server per frame, meaning that the servers must be physically located nearby the players, and thus many servers across many different cities will be needed

**How Disadvantages Diminish Over Time**

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

For instance, let's say a player joins a server being 300km away from his/her device(which is a bit far away already) to play a game with a 1080p@120Hz setup using this architecture, and the full latency would have to meet the following requirements in order to have everything done within around 9ms, which is a bit more than the maximum time allowed in 120 FPS:

1. The client will take around 1ms to capture and start sending the raw input commands from the player
2. The minimum ping, which is limited by the speed of light, will be 2 \* 300km / 300,000km per second = around 2ms
3. The server will take around 1ms to receive and combine all raw input commands from all players
4. The server will take around 1ms to convert the current game state set with those raw input commands to form the new game state set
5. The server will take around 1ms to generate all rendered graphics markups(which are lossless, highly compressed and highly obfuscated) from the new camera state of all players
6. The server will take around 1ms to start sending those rendered graphics markups to all players
7. The client will take around 1ms to receive and decompress the rendered graphics markup of the corresponding player
8. The client will take around 1ms to render the decompressed rendered graphics markup as the end result being perceived by the player directly

Do note that hardware limitations, like mouse and keyboard polling rate, as well as monitor response time, are ignored, because they'll always be there regardless of how a multiplayer game is designed and played.

**Of course, the above numbers are just outright impossible within years, especially when there are dozens of players in the same server, but they should become something very real after a decade or 2, because by then the hardware we've should be much, much more powerful than those right now.**

Similarly, for a 1080p@120Hz setup, if the rendering is lossless but isn't compressed at all, it'd need (1920 \* 1080) pixels \* 32 bit \* 120 FPS + little bandwidth from raw inputting commands sent to the server = Around 1GB/s per player, which is of course insane to the extreme right now, and the numbers for 4K@240Hz and 8K@480Hz(assuming that it'll or is always a real thing) setups will be around 8GB/s and 64GB/s per player respectively, which are just incredibly ridiculous in the foreseeable future.

**However, as the rendering markups sent to the client should be highly compressed, the actual numbers shouldn't be this large, and even if the rendering isn't compressed at all, in the distinct future, when 6G, or even newer generations, become the new norm, these numbers, while will still be quite something, should become practical enough in everyday gaming, and not just for enthusiasts.**

Nevertheless, there might be an absolute limit on the screen resolution and/or FPS that can be supported by this architecture no matter how powerful the hardware is, so while I think this architecture will be the distinct future(like after a decade or 2), it probably won't be the only way multiplayer games being written and played, because the other models still have their values even by then.

**Future Implications**

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, especially without needing any audio nor visual assets, is generally a much, much easier, simpler and smaller task than generating that markup itself, and the client side hosts almost no game data nor state so the hard disk space and 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, hard disk, motherboard, etc 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

4. The demands of high end servers will be higher and higher, not just from game companies, but also for some players investing a lot into those games, because they'd have the incentive to build some such servers themselves, then either use them to host some games, or rent those servers to others who do

**Anti-Cheating**

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(consisted of keyboard presses, mouse clicks, etc) 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 in-game admin having server level access to monitor all players in the server(now with the aid of the AI reporting suspicious raw player input sets for each player), 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.

**Hybrid Models**

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(of course, only players with the full client side package can join servers using the traditional counterpart, and only players with the server side subscription can join servers using this architecture).**

**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).**

**On the business side, it means such games will have a client side package, with a one time price for everything in the client side, and a server side package, with a periodic subscription for being able to play multiplayer, and single player with a dedicated server provided, then the players can buy either one, or both, depending on their needs and wants.**

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