

# 게임 서버 포트폴리오

('Battle Snake' 프로젝트)

송 진규

# 목차

1. ['Battle Snake' 서버 구조](#)
2. [네트워크 모듈](#)
3. [컨텐츠 서버](#)
4. [라이브러리](#)

# 프로젝트 'Battle Snake'

## Battle Snake는?

탄창 / 메디킷 / 헬멧 등의 아이템을 이용해 유저를 처치하고 승리를 쟁취하는 MOTPS(Multiplayer Online Third-Person Shooter) 게임

## 작업 내역 요약

### 1. Stateful Server

- 채팅/매칭/마스터/배틀 서버 제작 (직접 제작한 네트워크 모듈 사용)
- DB에 데이터 쓰기 및 읽기
- HTTP 통신
- 락프리 자료구조 / 메모리풀 / 프로파일링 / 파서 등 라이브러리 제작

### 2. Web

- PHP를 이용해 DB에 데이터 쓰기 및 읽기 (Log 저장, 유저 정보 읽기 및 쓰기)
- Apache 사용

### 3. DB

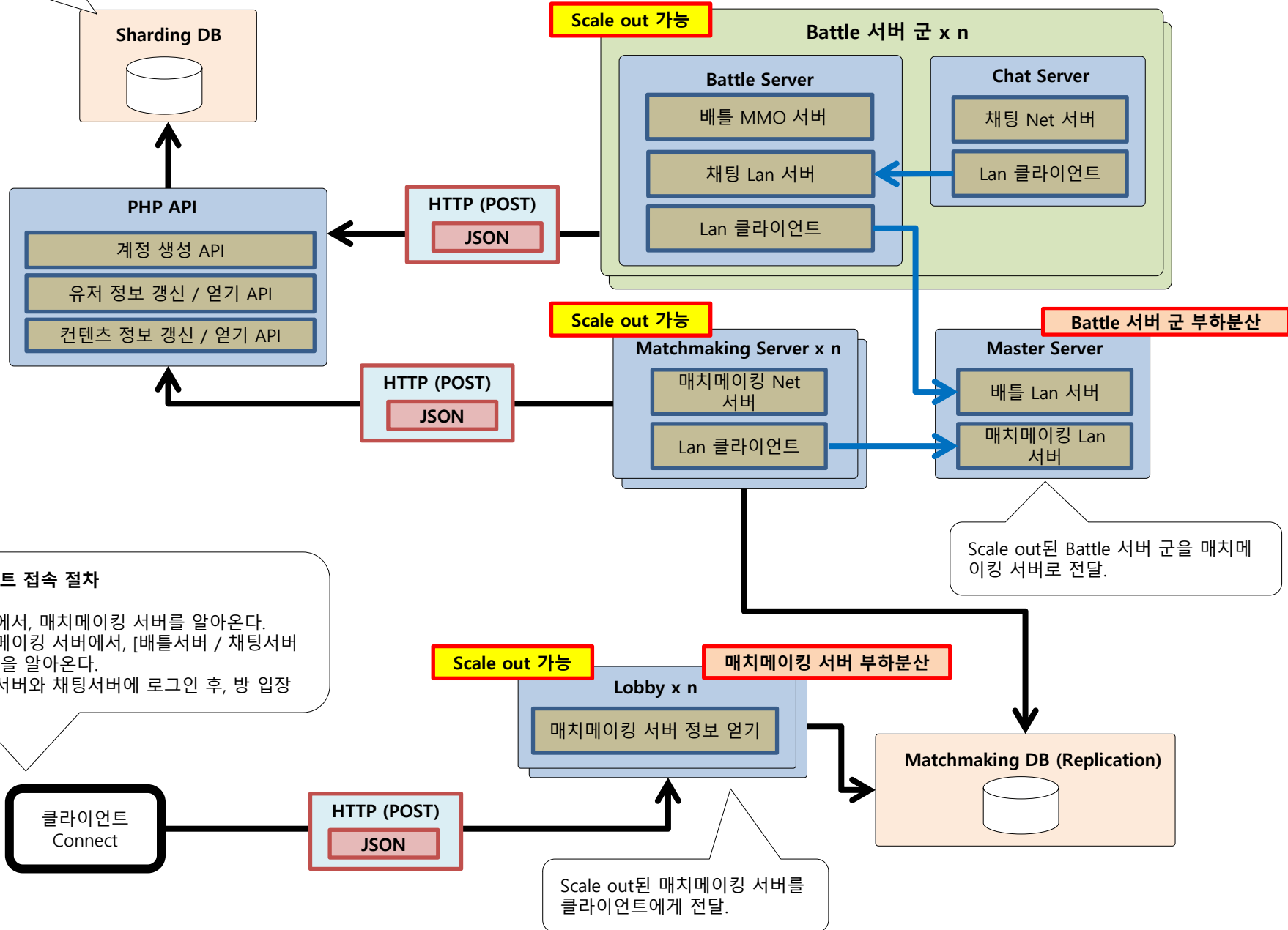
- MySQL 사용
- 유저 정보 관리
- Replication
- Sharding



# **'Battle Snake' 서버 구조**

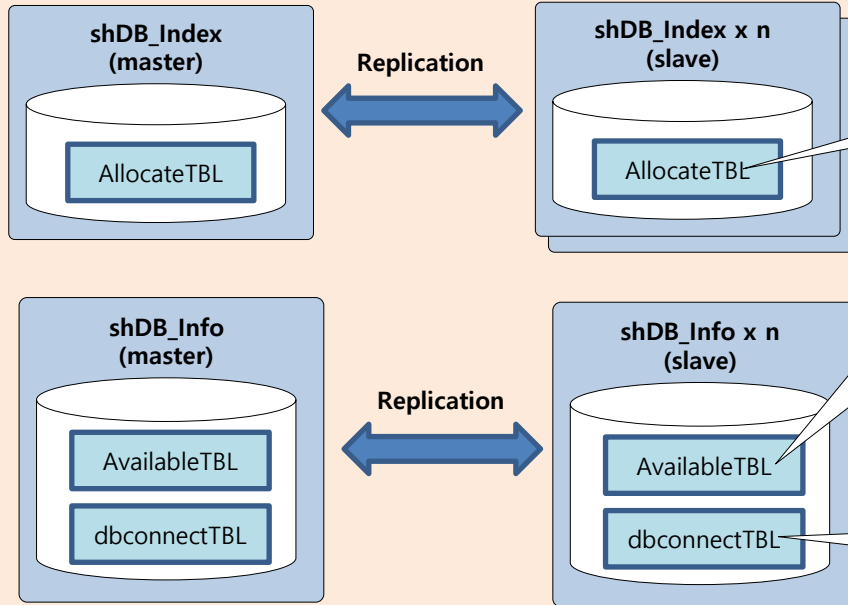
# 'Battle Snake' 서버 전체 구조

Sharding DB의 자세한 내용은 다음 페이지 참조



# Sharding DB 구조

## Sharding Info DB (Replication)



유저 식별자(AccountNo), Content DB Number  
(PK : AccountNo)

Content DB 별 저장 가능한 유저 수  
(PK : Content DB Number)

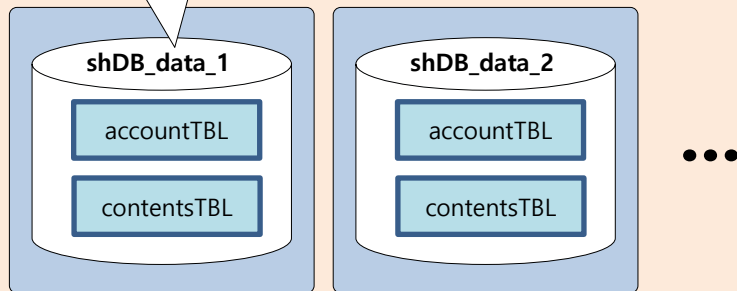
### Sharding 분산 지점

신규 계정 생성 시 AvailableTBL을  
기준으로, 저장될 Content DB 결정

Content DB Number, Content DB 접속 정보  
(PK : Content DB Number)

유저의 닉네임, 전적정보 등 콘텐츠 정보  
(PK : AccountNo)

## Content DB (Sharding)



## PHP API

계정 생성 API

유저 정보 갱신 / 얻기 API

컨텐츠 정보 갱신 / 얻기 API

Sharding Info DB를 거쳐, Content DB(Sharding)에 접속

1. 유저가 저장된 Content DB의 Number를 알아온다.  
(shDB\_Index에 접근)
2. Number를 이용해, IP와 Port 등의 접속정보를 알아온다.  
(shDB\_Info에 접근)
3. Content DB에 접속.

# 네트워크 모듈

# 개요

## 네트워크 모듈이란?

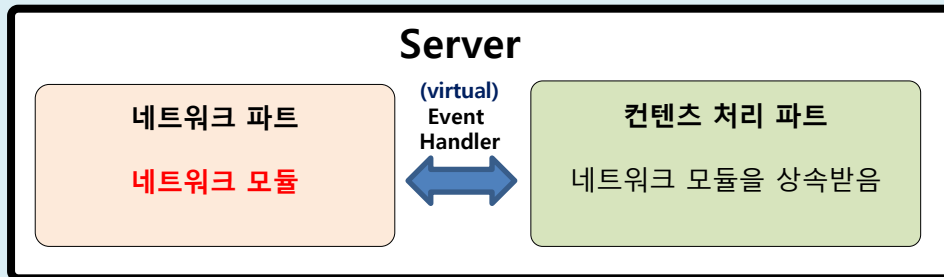
- Accept/Send/Recv/세션종료 등 네트워크 관련 작업을 처리하는 모듈
- IOCP(I/O Completion Port) 사용
- 각 서버는, 네트워크 모듈을 상속받아 제작

## 2종류의 네트워크 모듈 제작

- **Net 서버** : 기본적인 네트워크 모듈
- **MMO 서버** : 게임서버 제작에 특화된 네트워크 모듈

## 네트워크 모듈 사용 후 서버 구현부

- 콘텐츠 파트는 네트워크 모듈을 상속받아 제작
- 가상함수를 통한 이벤트 핸들러 제공





# Net 서버

## 장점

- LockFree 구조의 네트워크 모듈
- Worker 스레드가 직접 패킷 처리
- 범용적으로 사용하기 좋은 구조
- Unique값을 이용해 네트워크 파트 <-> 콘텐츠 파트간 통신

## 단점

보통 게임서버를 만들 때는 콘텐츠 처리부를 위한 단일스레드가 구성된다.

만약, 이 구조로 게임서버를 만들 경우, 패킷 내용을 다시 Update 스레드로 전달해야 하는 오버헤드 발생.

즉, 게임 서버에는 맞지 않는 구조

## 사용처

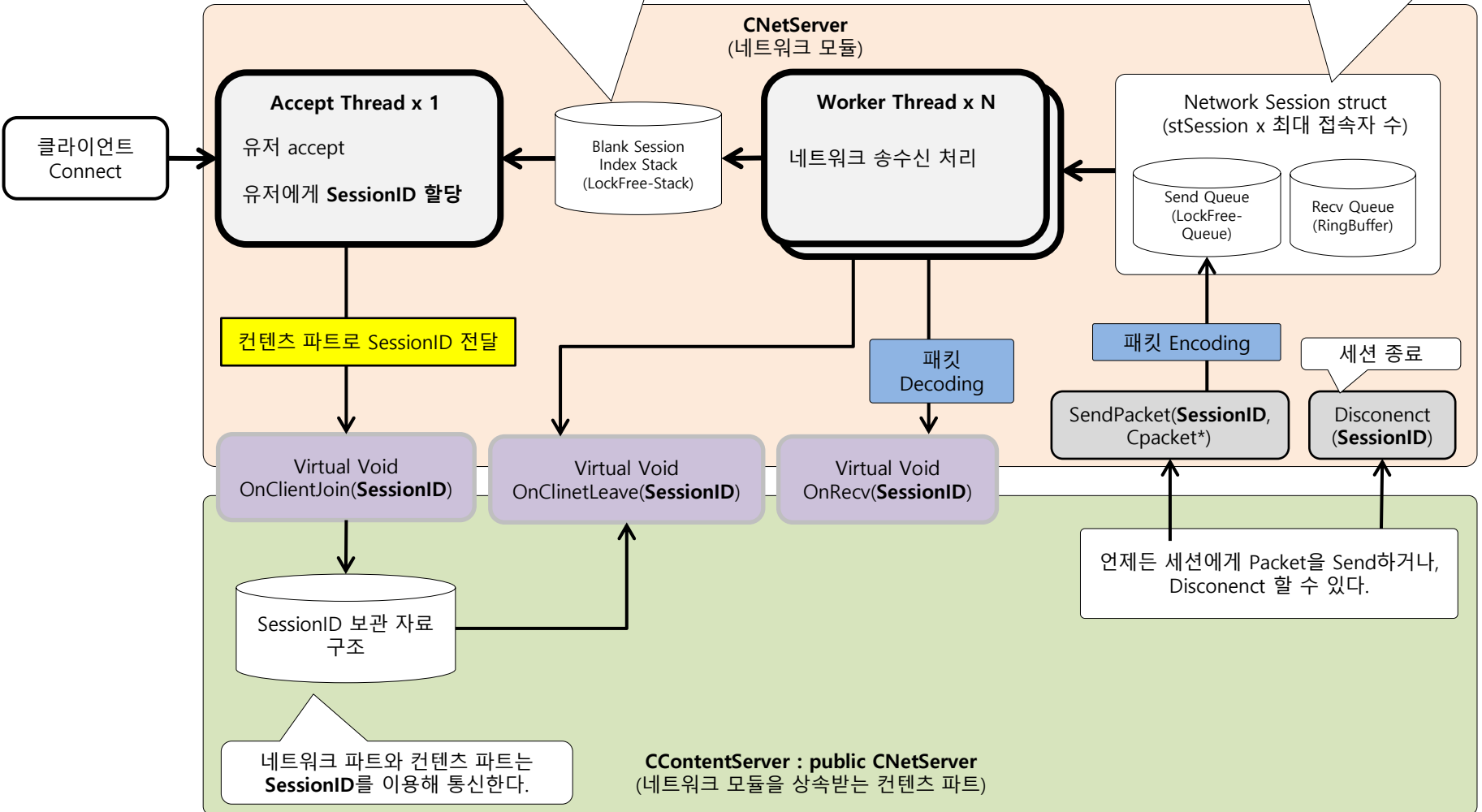
- 채팅 서버, 매치메이킹 서버, 마스터 서버

※ 채팅서버, 매치메이킹 서버, 마스터 서버는 게임 콘텐츠 처리부를 위한 단일 스레드를 구성할 필요가 없기 때문에, Net 서버 네트워크 모듈 사용

# Net 서버 구조

- 비어있는 Session의 Index 보관
- Accept 시, 이 stack에서 인덱스를 꺼내, 해당 인덱스의 배열에 세션 할당

- 정적 배열
- [Blank Session Index Stack] 에 빈 Session의 Index보관



# MMOServer

## 장점

- 네트워크의 Session과 콘텐츠의 Player가 1개로 구성. (상속 구조) → 네트워크 모듈과 콘텐츠 간의 통신 최소화
- Auth 스레드와 Game 스레드를 분리해 게임 개발에 적합한 구조.
- 스레드 별 모드 지정으로 스레드 동기화 제거
- 모듈 내부에 콘텐츠 처리 스레드와 패킷 처리 기능을 포함.

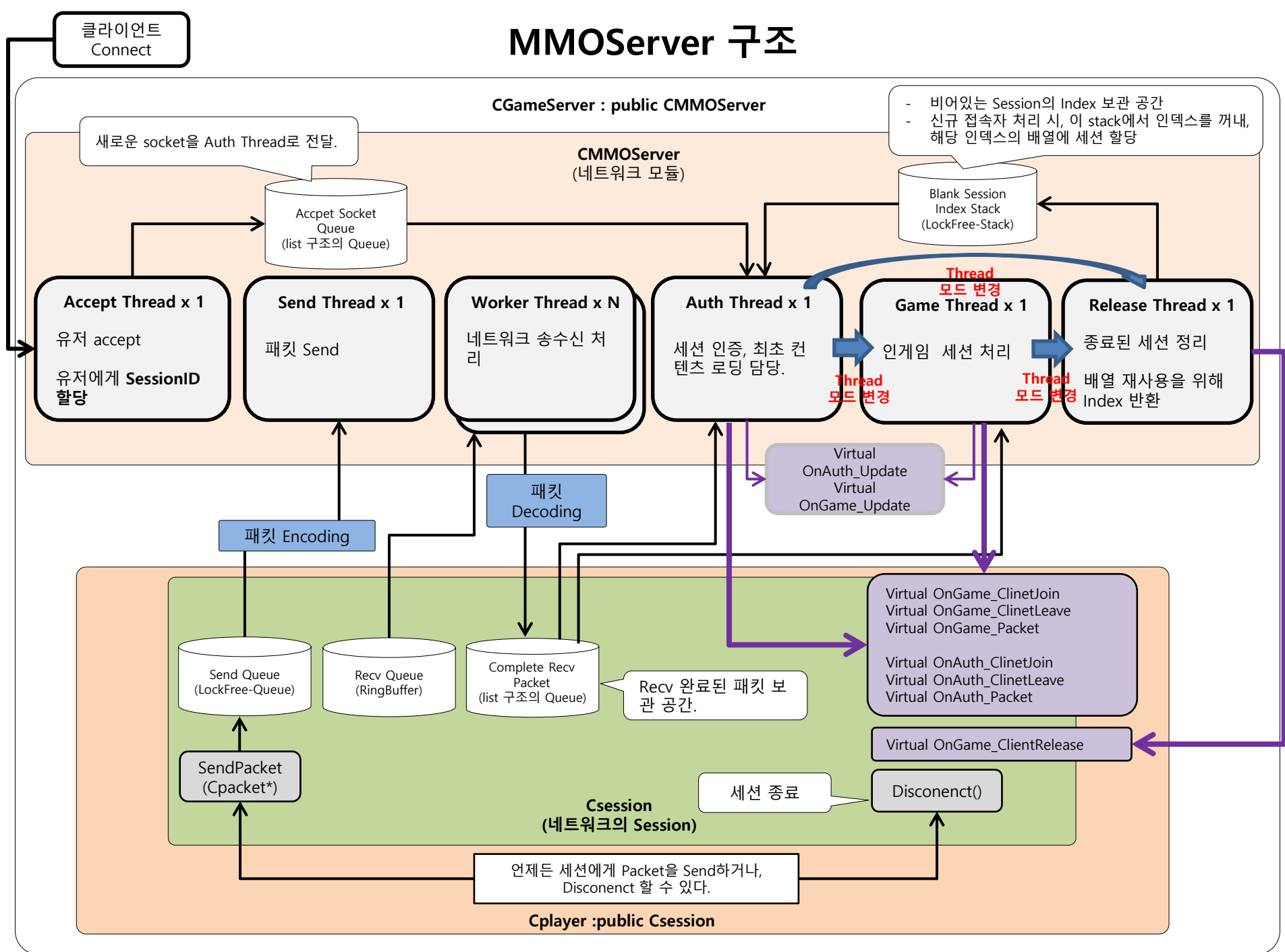
## 단점

- 모듈 내부의 스레드가 항상 루프를 돌기 때문에 CPU 사용율이 높다.
- Auth, Game 스레드에 지정된 프레임(루프 횟수)에 따라 패킷 처리 속도가 결정되며 최대 처리속도 한계가 있음
- 범용적으로 적용하기 어려운 구조

## 사용처

- 배틀 서버

# MMOServer 구조



# 컨텐츠 서버

# 채팅서버

## 개요

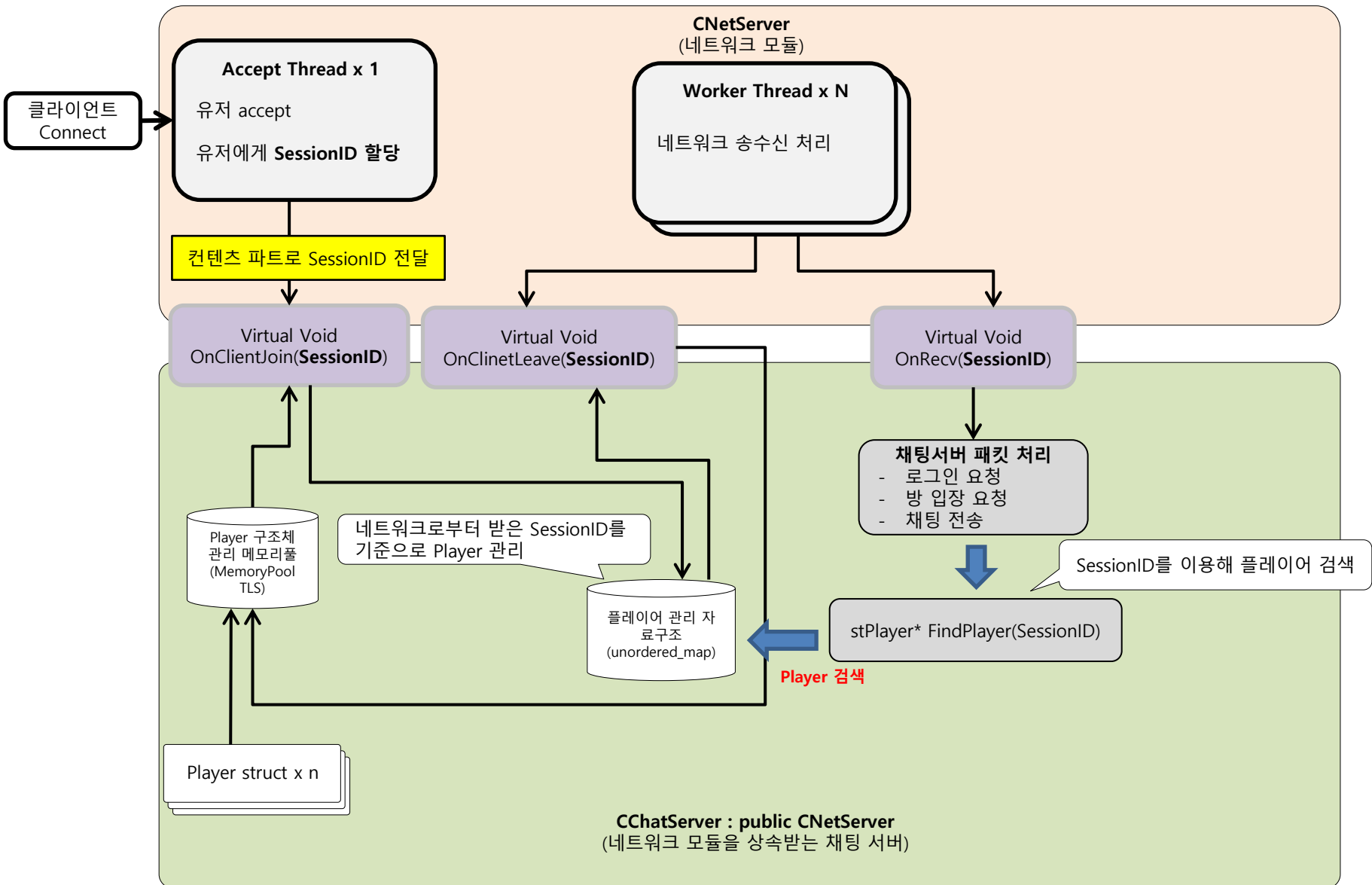
- Net 서버를 상속받는 채팅서버
- 2개의 채팅서버 제작

## 서로 다른 구조를 가진 2개의 채팅서버

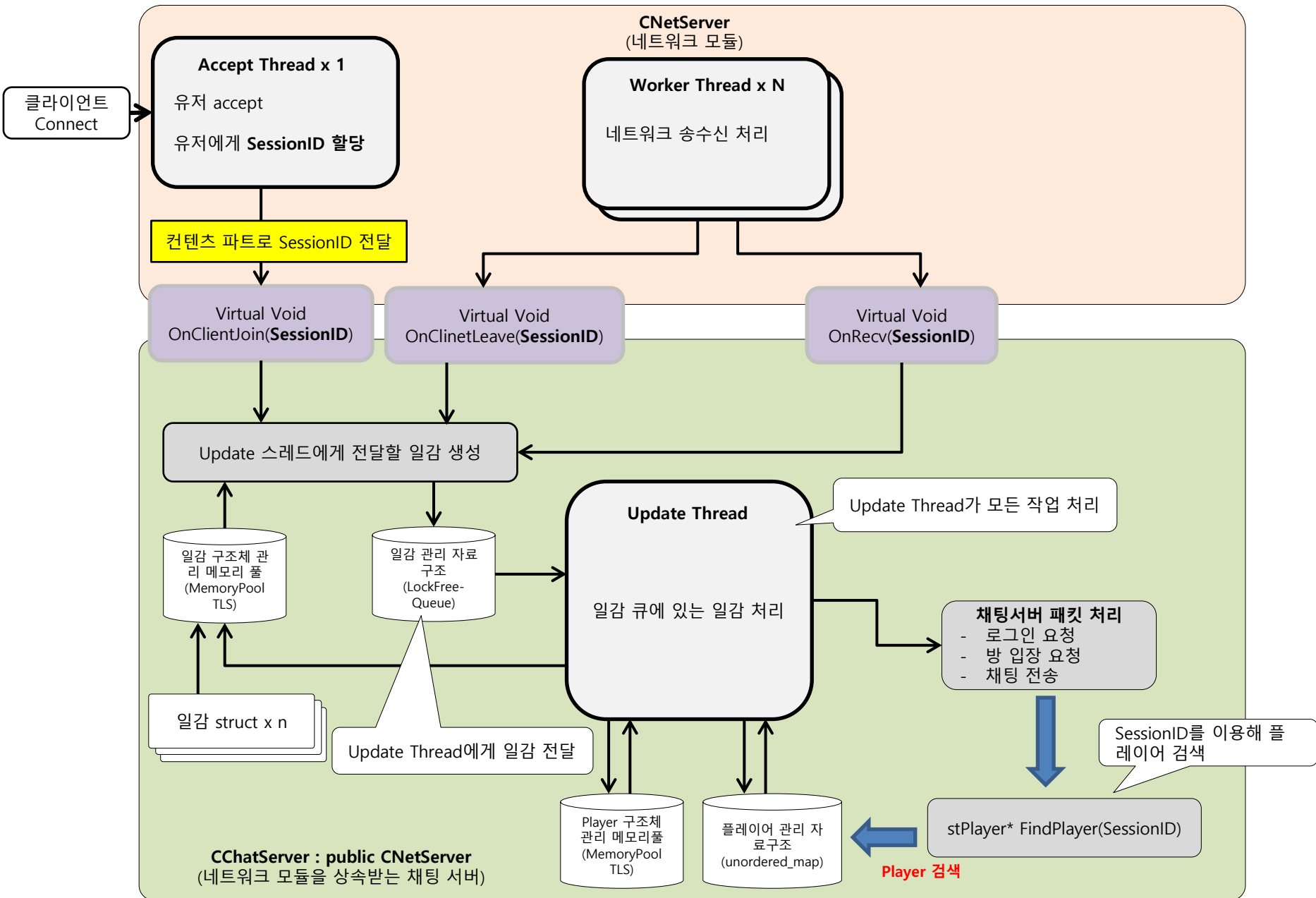
- 서로 다른 구조를 가진 채팅서버 2개 제작
- **Worker 스레드 구조** : Worker 스레드가 패킷 처리
- **단일 Update 스레드 구조** : Worker 스레드는 Update스레드에게 일감을 전달하고, 실제 패킷 처리는 단일 Update 스레드에서 처리

※ 최종 라이브는 **Worker 스레드 구조의 채팅서버**를 사용했으며, 단일 Update 스레드 구조는 연습 및 Worker 스레드 구조와 성능 비교 용도

# 채팅서버 - Worker 스레드 구조



# 채팅서버 - 단일 Update 스레드 구조





# 배틀서버

## 개요

- MMOServer를 상속받는 배틀 서버
- 접속한 유저 인증, 게임 패킷 처리 담당
- 비동기 HTTP 통신

## 비동기 HTTP 통신

### 목적

- Block 작업을 비동기로 처리함으로써, Auth/Game 스레드의 처리 속도 증가

### 1. HTTP Read Thread

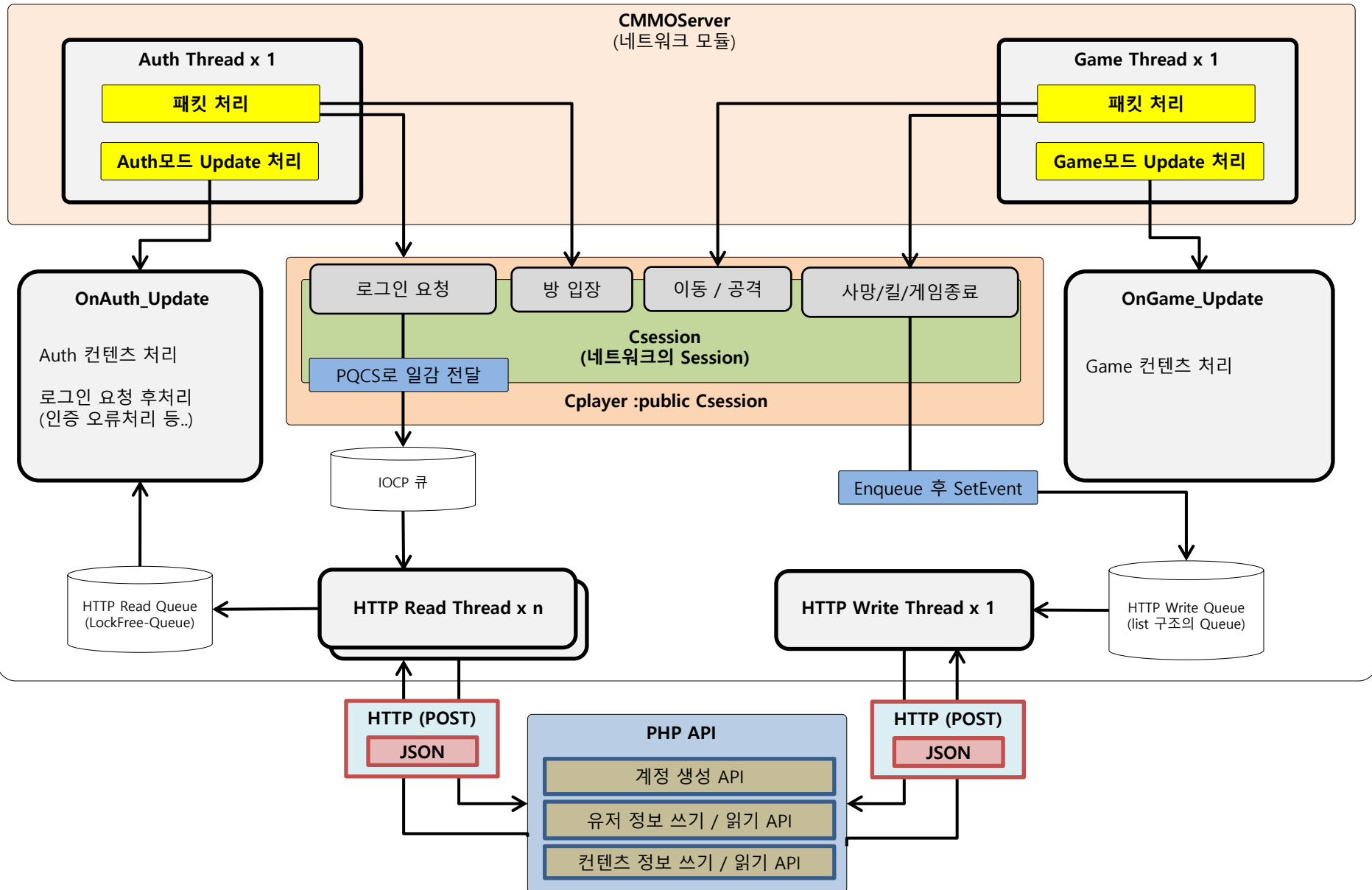
- PHP API 읽기용 스레드
- GQCS로 대기
- PQCS로 전달받은 일감 처리

### 2. HTTP Write Thread

- PHP API 쓰기용 스레드
- 이벤트로 대기
- Queue로 전달받은 일감 처리
- Write 동기화를 위해 스레드 1개로 제한

# 배틀서버 구조

CBattleServer : public CMMOServer



# 라이브러리

# Profiler, Parser, Logging

## Profiler

- 마이크로 세컨드 단위로 성능을 측정하는 프로파일러
- TLS(Thread Local Storage) 사용
- 결과를 File로 저장하거나, 콘솔 화면에 출력할 수 있다.

사용처 : 배틀 서버의 HTTP Read Thread , HTTP Write Thread

## Parser

- txt 파일을 파싱하는 라이브러리

사용처 : 각 서버 시작 시 Worker스레드 수, 최대 접속가능 유저수 등 서버의 기본정보를 읽어올 때 사용

## Logging

- 로그 내용을 txt파일로 저장하는 라이브러리
- 일반 로그, 16진수로그 총 2종류의 로그 남기기 가능
- 싱글톤 사용

사용처 : 서버가 Crash 나는 상황, 하트비트로 끊기는 유저에 대한 로그, 기타 등등...

# 메모리풀 TLS

## 개요

- Thread Local Storage를 이용한 메모리풀
- 템플릿 사용
- new / delete보다 빠른 속도 목표

## New/delete와 비교 테스트

- New/ delete / 메모리 풀 Alloc / 메모리 풀 Free를 각각 1억회 테스트  
(직접 제작한 프로파일링 라이브러리 사용)
- New 보다 **약 11배 빠른 Alloc**
- Delete 보다 **약 4.3배 빠른 Free**

MPool\_TLS\_Profiling\_4.txt - 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말

마이크로 세컨드 기준 평균

ThreadID	Name	Average	Min	Max	Call
12676	New_1	115450.608μs	72149.800μs	206439.200μs	99
12676	delete_1	82161.855μs	63528.900μs	149071.100μs	99
12676	TLS Alloc_1	10401.098μs	8791.100μs	20712.600μs	99
12676	TLS Free_1	18833.294μs	16878.500μs	34696.700μs	99
12676	New_2	92090.571μs	75408.700μs	162334.100μs	99
12676	delete_2	71245.266μs	64096.100μs	112125.800μs	99
12676	TLS Alloc_2	9109.922μs	8808.400μs	9620.900μs	99
12676	TLS Free_2	17931.126μs	16889.000μs	26214.000μs	99

## 사용처

- 동적 할당이 필요한 모든 경우 (Ex. 채팅서버의 플레이어 구조체)

# LockFree 자료구조

## 개요

- LockFree-Queue / LockFree-Stack 총 2개의 락프리 자료구조 제작
- 템플릿 사용
- **double CAS 기반**
- Unique Count 사용

## Critical Section과 비교 테스트

- 스레드 10개가 각각 10만개의 데이터, 총 100만개의 데이터를 Enqueue / Dequeue (직접 제작한 프로파일링 라이브러리 사용)
- LockFree-Queue의 Enqueue/ Dequeue가 **약 1.6배 빠름.**

Profiling\_LockFreeQueue\_5.txt - 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말

ThreadID	Name	Average	Min	Max	Call
672	Enq	149131.000µs	149131.000µs	149131.000µs	2
672	Deq	130771.000µs	130771.000µs	130771.000µs	2

마이크로 세컨드 기준 평균

LockFree-Queue 프로파일링

Profiling\_Critical\_5.txt - 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말

ThreadID	Name	Average	Min	Max	Call
12476	Enq	240281.800µs	240281.800µs	240281.800µs	2
12476	Deq	211991.700µs	211991.700µs	211991.700µs	2

Critical Section 프로파일링

## 사용처

- 데이터 관리, Thread 간 통신  
(Ex. 배틀서버에서 HTTP Read Thread에게 일감 전달)

# HTTP 라이브러리

## 개요

- IP / Domain을 이용한 HTTP 통신용 라이브러리
- 기본 웹통신 구조 (Connect -> Send -> Recv -> Disconnect)
- 비동기 HTTP 통신을 위한 도구

## 목적

웹 통신 자체는, Block으로 작동할 수 밖에 없다.

하지만, 웹 통신 담당 스레드를 따로 둬으로써 웹 통신 작업을 비동기처럼 처리할 수 있다. (**생산자 - 소비자 모델**)

해당 라이브러리는, 소비자 스레드가 웹 통신을 사용할 때 사용한다.

## 사용처

- 배틀 서버의 HTTP Read Thread , HTTP Write Thread

# HTTP 라이브러리 사용 예

