MMOG Server-Side 충돌 및 이동처리 설계와 구현

Project D Online에서의 적용사례

유영천
tw: @dgtman
Blog: megayuchi.com

WSAD + 마우스 컨트롤

- FPS 혹은 TPS
- 지금은 무척 흔한 조작방식

FPS – Quake3

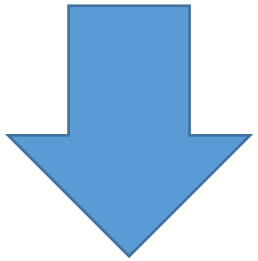
TPS – Project D Online

WSAD + 마우스 컨트롤 기본 충돌 처리

- 캐릭터 vs 지형지물
- 캐릭터 vs 캐릭터
- 캐릭터 vs 각종 오브젝트(로켓탄 등)

이동처리 ? 충돌처리?

- 왜 자꾸 이동처리랑 충돌처리란 단어를 혼용하나?



- 3D에서 이동처리를 제대로 하려면 충돌처리를 해야한다.

싱글게임에서의 충돌처리

- 이동 및 전투 판정을 위한 기본 충돌처리

  +

- 지형지물 파괴 처리
- 파티클 충돌처리
- 여기에 온갖 물리처리 추가
- 얼마나 더 정밀하게, 더 화려하게 할 지가 관건

WSAD + 마우스 컨트롤 Online Game

- 서버랑 동기를 맞춰야 하므로 간단한 모델을 사용한다.
- 그래도 위치와 타격 판정에 있어선 복잡한 처리를 피할 수 없다.
- 문제는 서버에서도 이걸 해야됨.
- 나(로컬 클라이언트)를 보고 있는 다른 클라이언트에서도 해야 됨.
- 자금,인력 되면 연출용으로 싱글게임에서의 충돌처리를 추가한다.

이동처리 기본전략

- 키를 누르면 해당 패킷 보냄
- 움직이기 시작
- 서버에서 주변 캐릭터에게 브로드캐스팅
- 서버에서 패킷 받아서 서버측 캐릭터를 움직임

2D일땐 쉽다.

- 높이 개념이 없으면 쉽다.
- 딱히 3D충돌처리 필요없다.
- 간단한 수식으로 비교적 정확하게 위치를 동기화할 수 있음.

3D일땐 어렵다

- 지형이 울퉁불퉁하면 시간 n경과 후 위치를 정확히 알기 어렵다.
- 지형과 다른 캐릭터에 충돌시켜 봐야 안다.
- 플레이어에 해당하는 매쉬를 지형(삼각형)에 속도(벡터)만큼 충돌시켜서 미끄러짐 벡터를 구한다.
- 이 벡터가 소진할때까지 루프

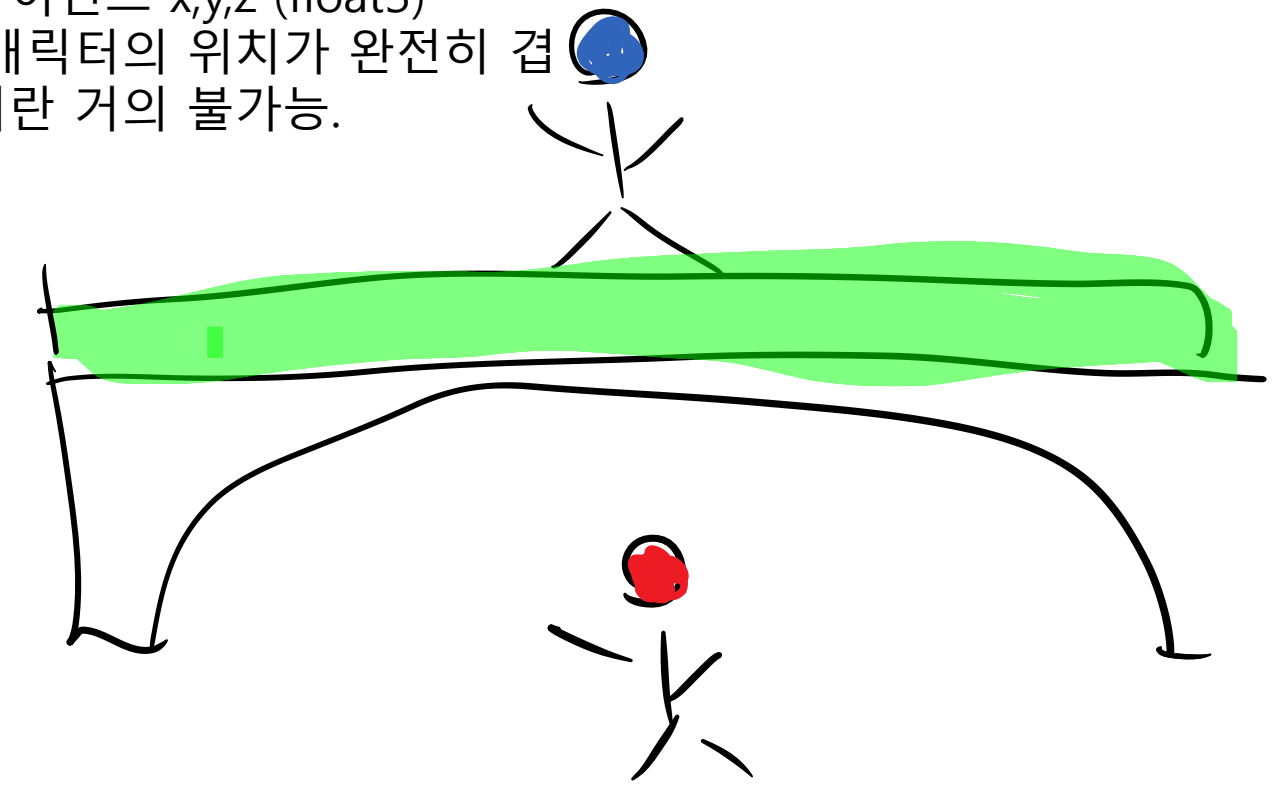
쉬운 방법

- 자기자신만 충돌처리
- 위치를 알리는 패킷을 주기적으로 보냄
- 서버에선 타일 등으로 검사
- 시간대비 이동거리로 검사
- 다른 플레이어들이 볼때 이동보간을 제대로 할 수 없어서 웃기게 보임

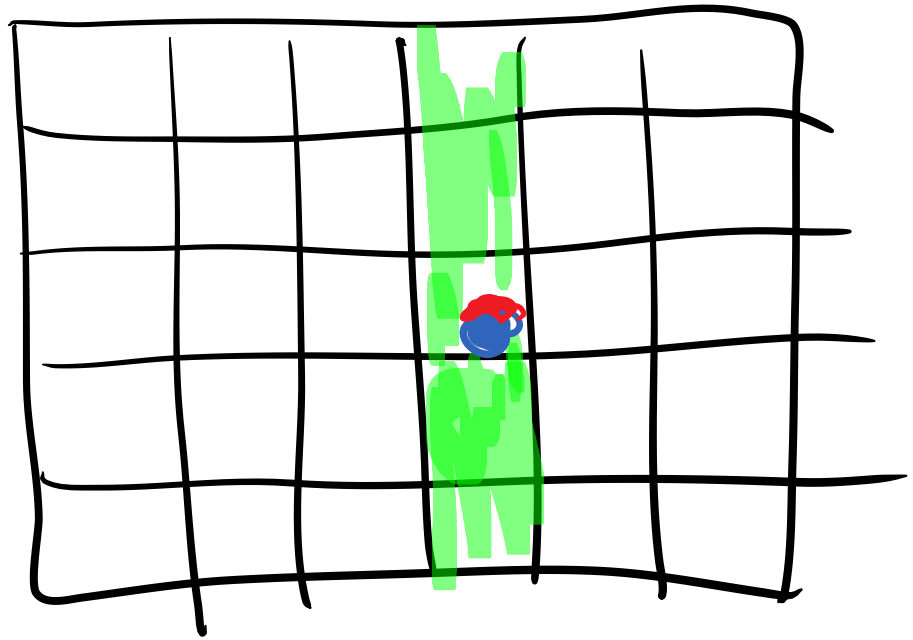
왜 다리 밑에는 캐릭터가 있으면 안되나?

MMORPG가 우후죽순처럼 쏟아지던 시절 기획자와 서버 프로그래머과 클라이언트 프로그래머가 죽도록 싸웠던 화두

클라이언트 x,y,z (float3)
두 캐릭터의 위치가 완전히 겹치기란 거의 불가능.



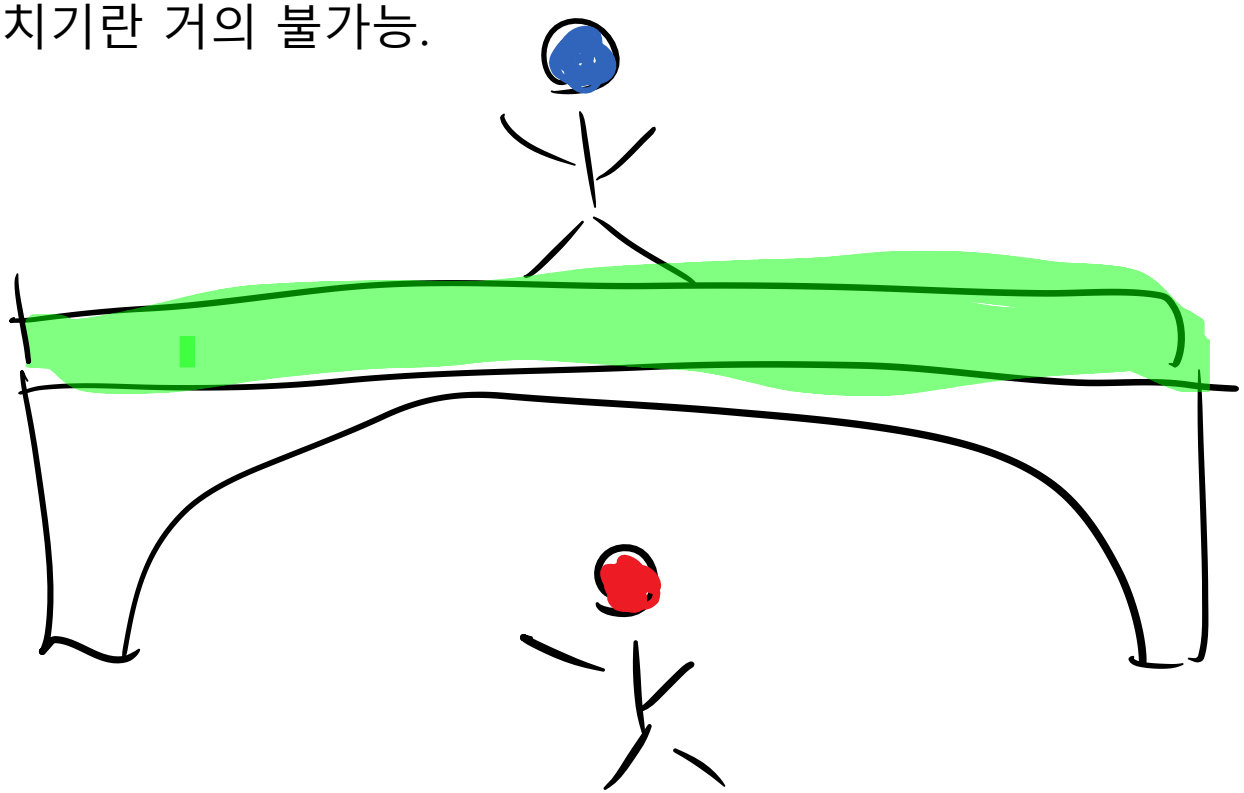
서버 x,z (int2) 다리를 만들어 놓으면 두 캐릭터의 위치가 겹치는 상황이 쉽게 발생한다,



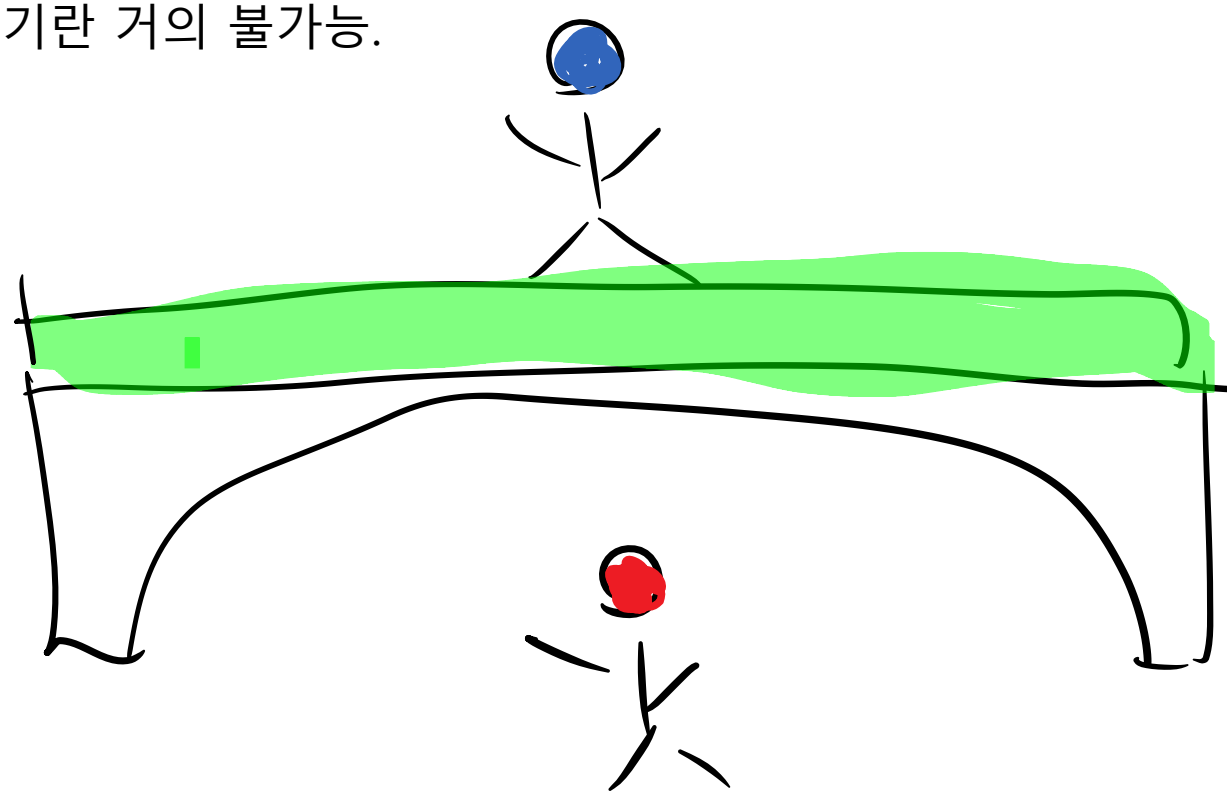
과거 많은 온라인 게임 서버들이 2D타일구조를 사용했고 타일 상에서 좌표가 겹치는 경우 이를 구분할 깔끔한 방법은 없었다.

이렇게 만들고 싶었다.

클라이언트 x,y,z (float3)
두 캐릭터의 위치가 완전히 겹치기란 거의 불가능.



서버 x,y,z (float3)
두 캐릭터의 위치가 완전히 겹치기란 거의 불가능.



쉬운 방법 쓰면 안되요?

- 정밀한 체크가 흥행성공을 보장하진 않는다.
- 오히려 기술적인 코스트 때문에 게임 내용에 제약을 줄 수도 있음.
- 해킹에는 취약함. MMORPG라면 문제가 될 수 있음.
- 그래도 돈 잘 번 게임도 많음.
- 논쟁할 생각 없음.

다만...
진짜 제대로 만들어보고 싶었다! 서버에서 완벽하게 처리해보고 싶었다고!!!

그래서 어떻게 만들었나...

클라이언트에서 하던 충돌처리 서버로 옮기면 되겠네. - 2004년 봄(딱 10년전)

Dual CPU(Pentium 3 600MHz x 2 머신으로 나름 시뮬레이션을 해봤다.
될것 같았다. 플레이어 4000명 정도는 처리할 수 있을것 같다.

다만 그때 소요 클럭을 잘못 계산했다. 실제보다 1/10로 측정했다. 그때 제대로 측정했다면 일찌감치 포기하고 10년간 이 삽질을 안했을지도.

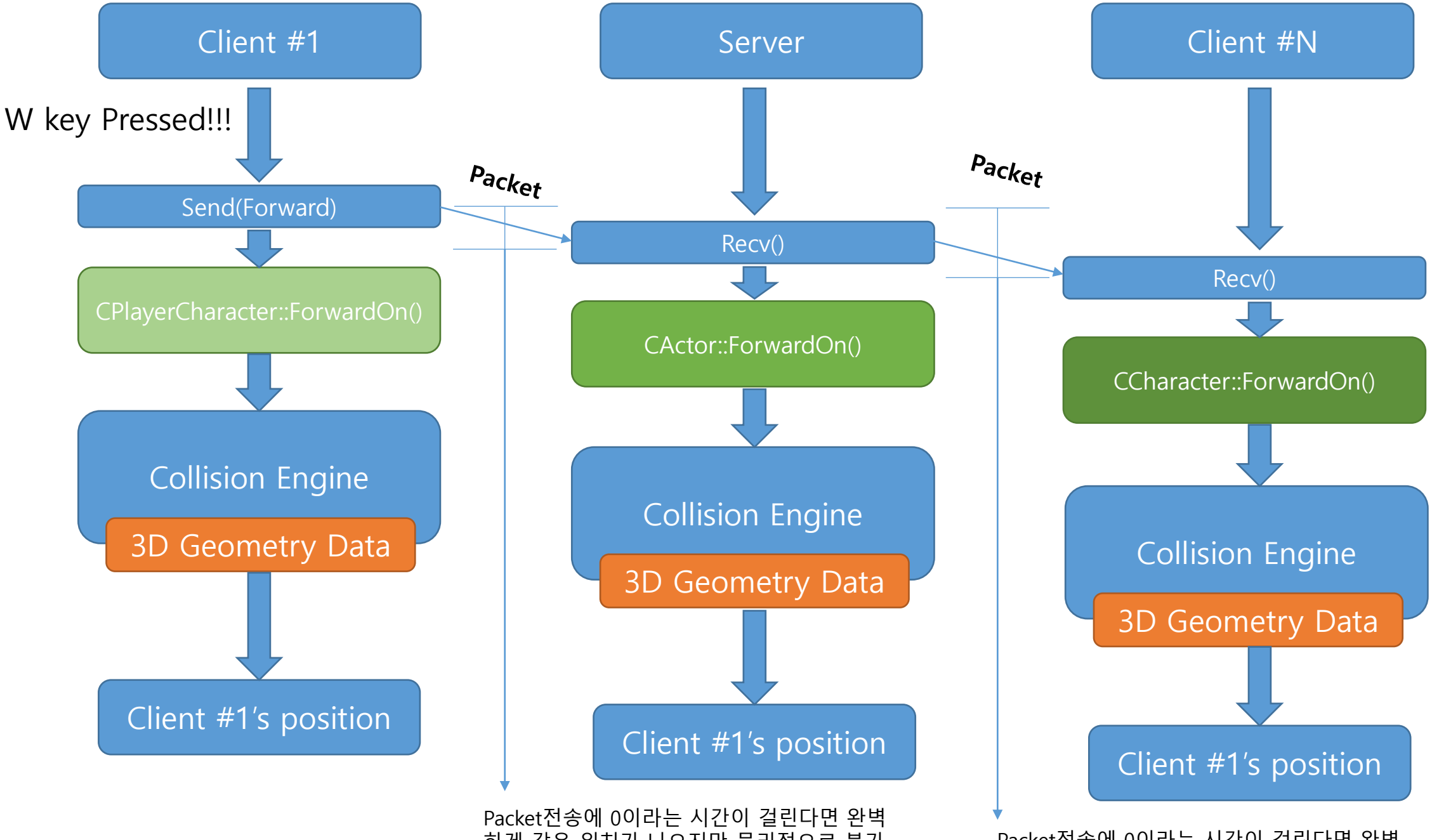
기본적인 이동처리 전략

- 이동시작
  - 클라이언트가 패킷보냄 (Forward)
- 서버에서 패킷 수신
  - 브로드캐스팅
  - 서버측 충돌엔진에서 이동시작
- 다른 클라이언트에서 패킷 수신
  - 클라이언트측 충돌엔진에서 이동시작

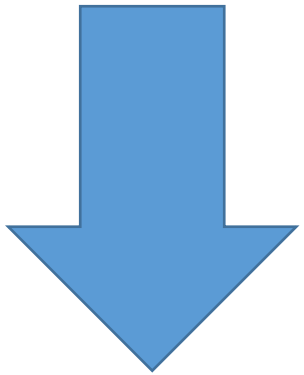
✓서버와 클라이언트 모두 게임 프레임은 30프레임 이상으로 돌아감

✓패킷 지연시간은 일단 무시함

✓일정범위의 오차를 허용함


Client #1
W key Pressed!!!
Send(Forward)
CPlayerCharacter::ForwardOn()
Collision Engine
3D Geometry Data
Client #1's position
Packet
Server
Recv()
CActor::ForwardOn()
Collision Engine
3D Geometry Data
Client #1's position
Packet
Client #N
Recv()
CCharacter::ForwardOn()
Collision Engine
3D Geometry Data
Client #1's position
Packet전송에 0이라는 시간이 걸린다면 완벽
하게 같은 위치가 나오지만 물리적으로 불가
Packet전송에 0이라는 시간이 걸린다면 완벽
하게 같은 위치가 나오지만 물리적으로 불가

컨텐츠 프로그래머가 이동처리와 충돌처리를 생각하려면 머리 아프잖아?



생각하기 편한 구조를 만들자.

게임 전체에서의 이동 및 충돌처리 전략

- 뭔가 마법이 있다.
- 그 마법으로 인해서 게임의 매 프레임마다 내 클라이언트에서 보고 있는 모든 캐릭터의 위치가 정확히 계산되어 있다.
- 그 마법으로 인해서 서버에서도 매 프레임마다 모든 캐릭터의 위치가 정확히 계산되어있다.
- 총을 들고 있을 경우 그 총의 ray에 걸치는 오브젝트 또는 지형에 대한 충돌점이 정확히 계산되어 있다.
- 게임 루프에서는 이 모든 것을 그냥 읽어서 사용하면 된다.

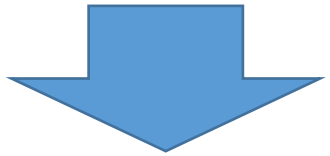
이것만 처리하면 된다.

- 임의의 타이밍에 아래 사항만 확실히 알수 있으면 된다.
  1. 누가(캐릭터) 어디에 있는가.
  2. 총을 들었을 경우 어느 방향을 향하고 있고 발사할 경우 맞는 캐릭터는 누구인가. 캐릭터가 맞지 않는다면 탄착점의 위치는 어디인가.

- 서버와 클라이언트 똑같다. 이 사항만 확실히 알 수 있으면 된다.

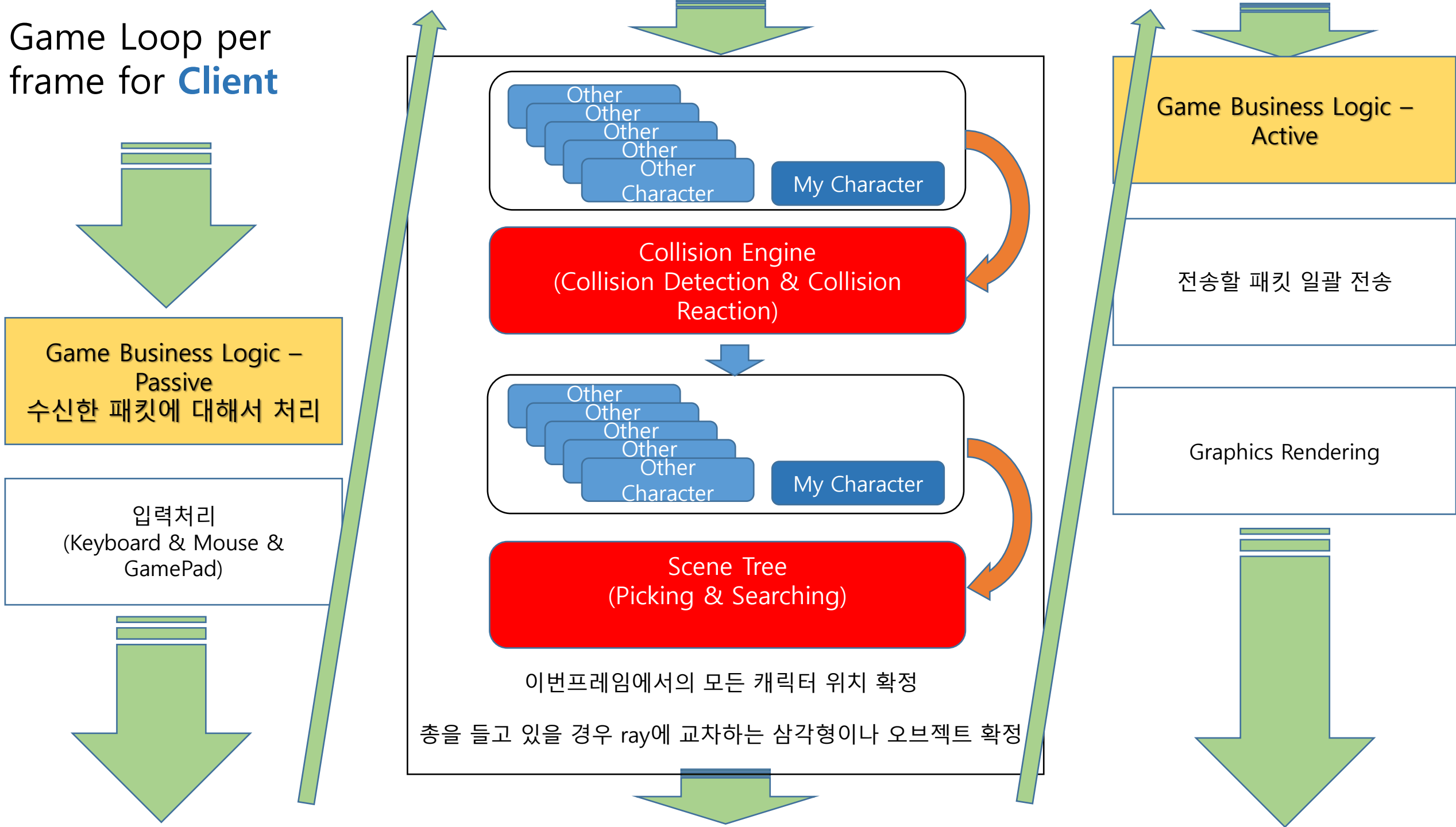
클라이언트측 구현(매 프레임마다)

- 내 캐릭터 위치 확정
- 내 클라이언트가 보고 있는 다른 캐릭터들의 위치 확정
- 내 캐릭터가 총을 들고 있다면 ray에 걸치는 삼각형 또는 오브젝트 확정
- 내가 보고 있는 다른 캐릭터들이 총을 들고 있다면 ray에 걸치는 삼각형 또는 오브젝트 확정



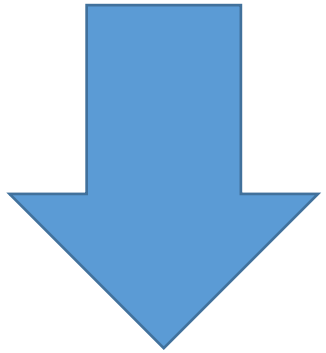
- 이제 이동처리 따위 신경쓰지 말고 게임 비지니스 로직 처리

Game Loop per frame for **Client**



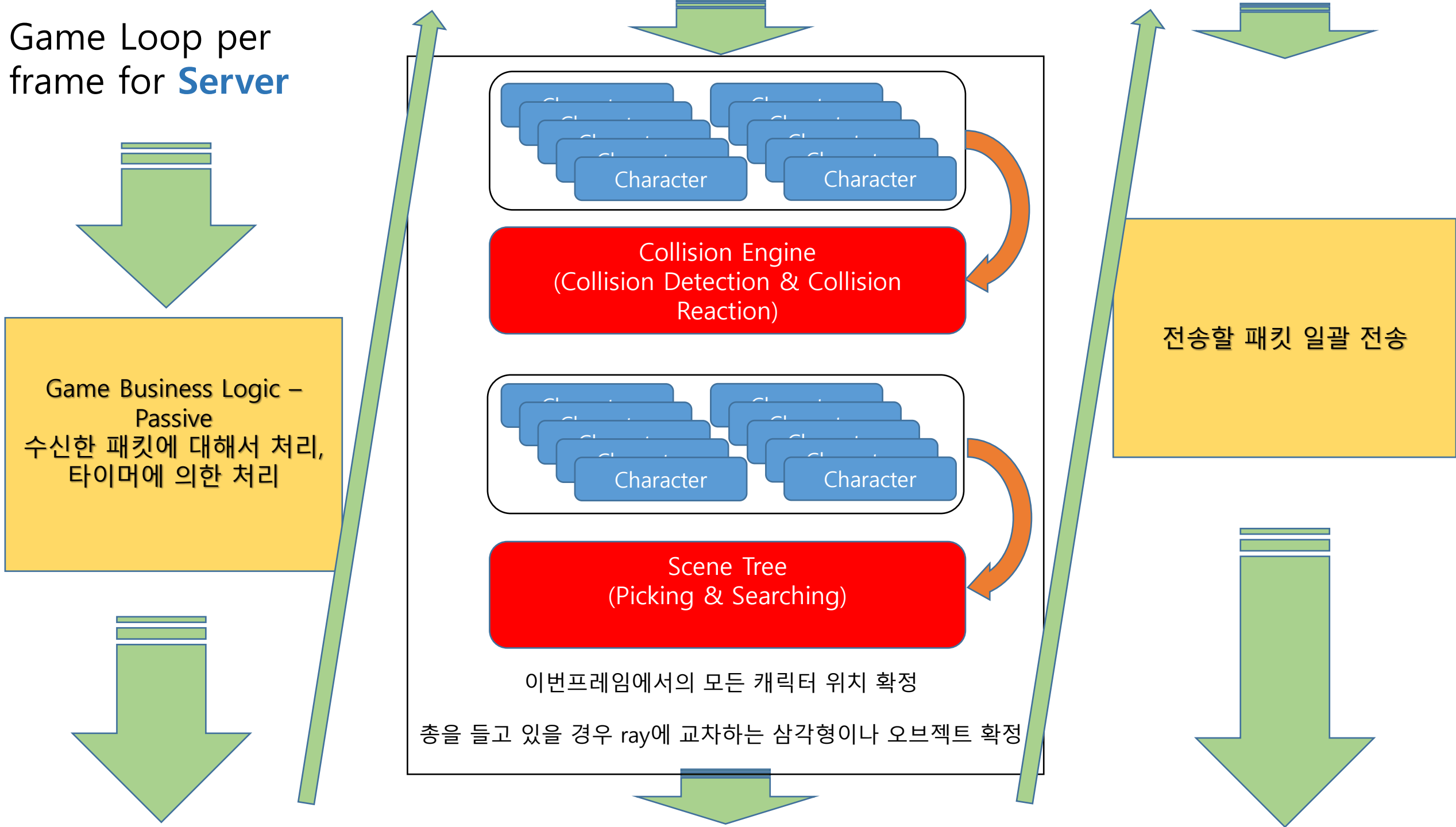
서버측 구현(매 프레임마다)

- 모든 캐릭터의 위치 확정(NPC포함)
- 총을 들고 있는 모든 캐리터에 대해서 ray에 걸치는 삼각형 또는 오브젝트 확정



- 이제 이동처리 따위 신경쓰지 말고 게임 비지니스 로직 처리

Game Loop per frame for **Server**



해야할 일은 비슷하네.

똑같은 코드를 작성해서 서버/클라이언트 양쪽 다 쓸 수 있을것 같다.

서버 클라이언트 겸용엔진을 만들자

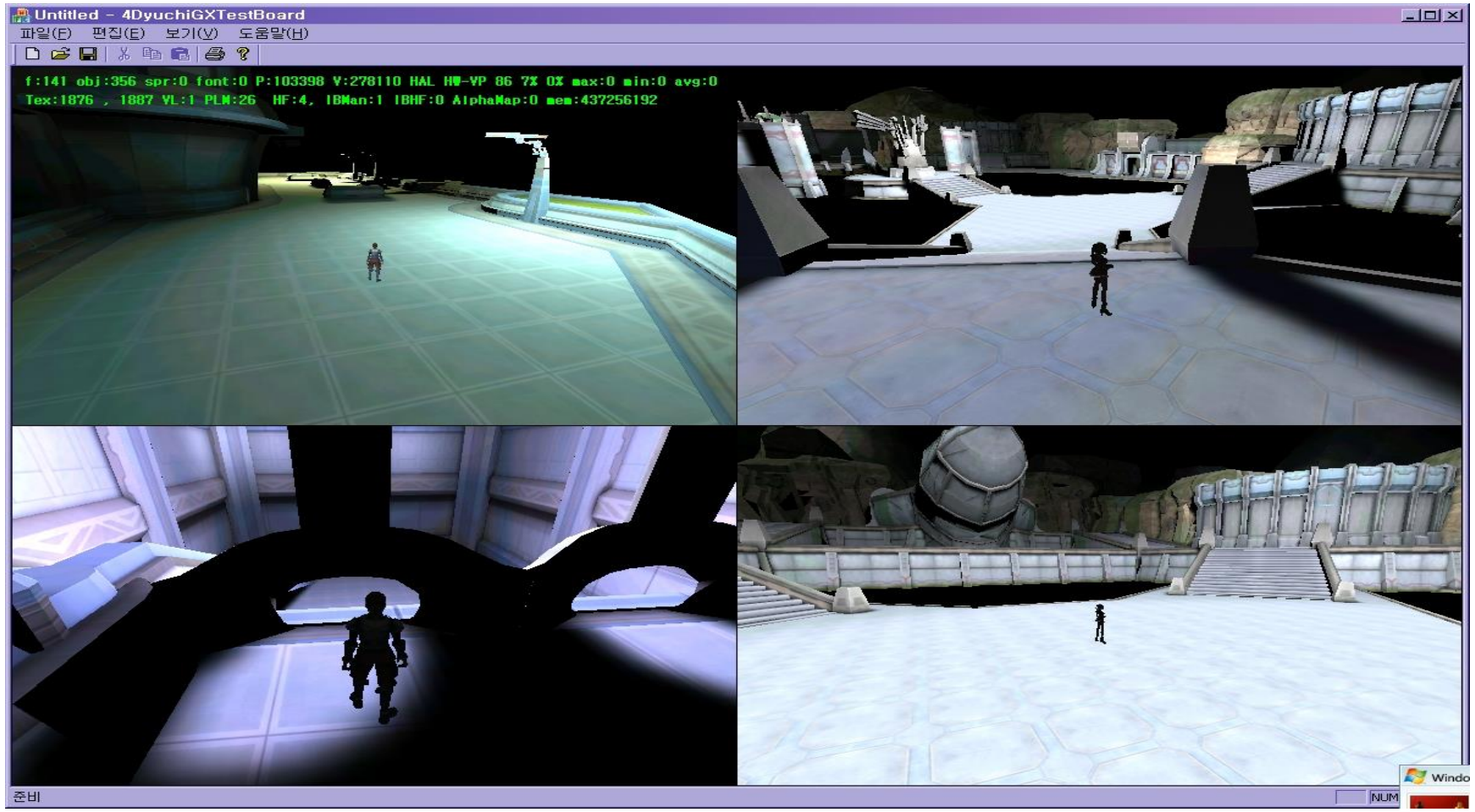
- 어차피 대량의 코드를 공유할테니 아예 바이너리 레벨에서 공유 가능한 엔진을 만들자.
- 그래픽 렌더링과 사운드 출력, 키보드 마우스 입력을 빼면 서버랑 클라이언트랑 다를게 없다.
- 아니 사실은 다른게 있다. -> 서버는 클라이언트보다 훨씬 많은 수의 오브젝트를 처리하고 동시에 여러 개의 맵을 로드하고 관리한다.
- 그리고 또? -> 그리고 없슈.

Cilent/Server 공용 엔진 요구사항

- 다수의 맵을 동시에 로드하고 관리할 수 있을 것
- 하나의 리소스로 여러개의 맵을 인스턴싱할 수 있을 것.
- 64bit 코드(64Bit 주소공간) 사용 가능할것.
- 그래픽API에 독립적일 것-렌더러만 분리할 수 있을 것.
- **복수의 캐릭터들에 대한 충돌처리 기능.(뒤에 자세히 다룸)**
- **복수의 레이체크 기능.(뒤에 자세히 다룸)**

다수의 맵을 동시에 로드

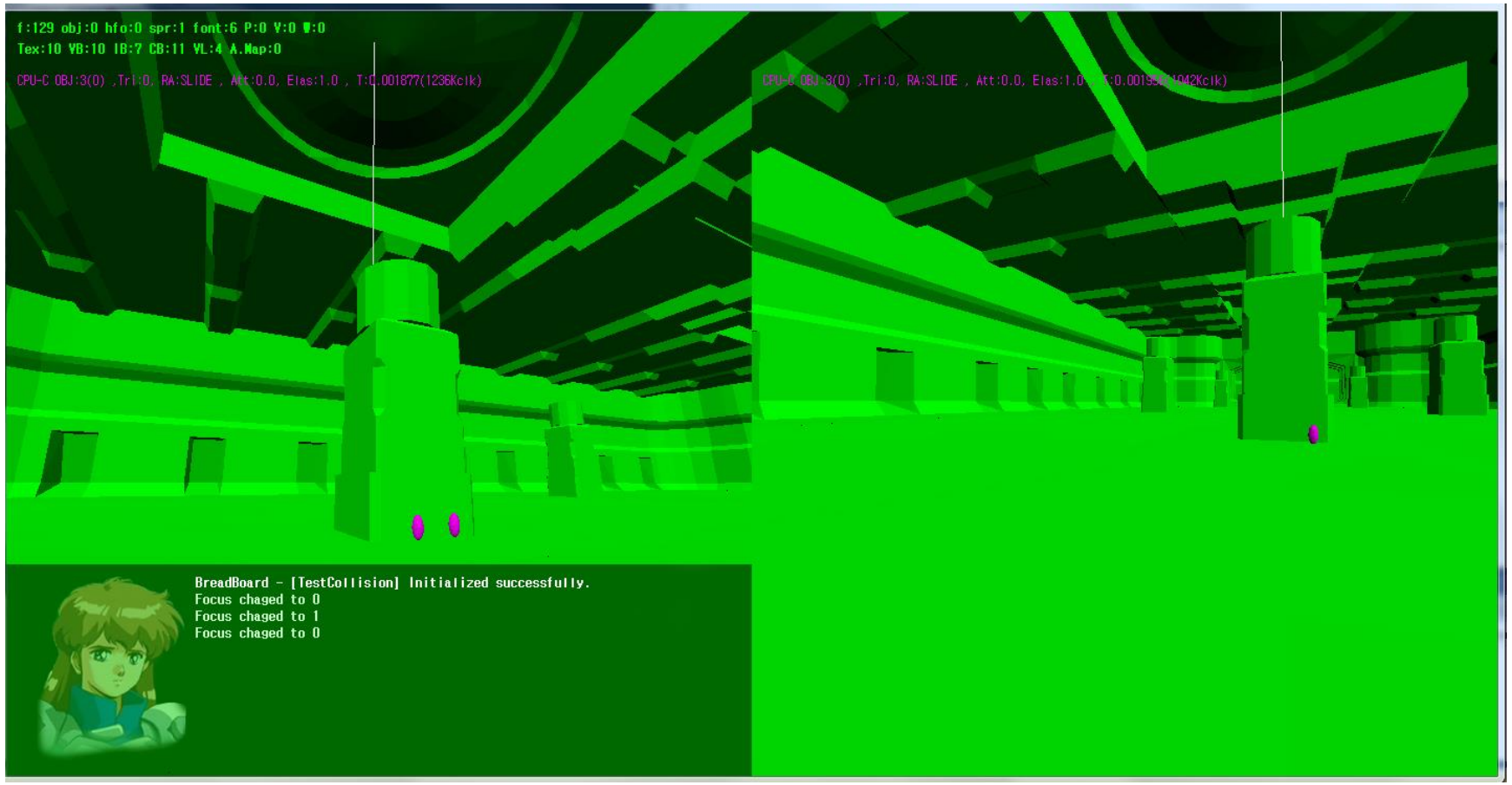
- 클라이언트는 한번에 1개의 맵을 로드한다.
- 서버는 모든 맵을 다 로드한다.



하나의 엔진에서 동시에 여러개의 맵을 로드한 경우

게임 맵 인스턴싱

- 하나의 리소스로 여러 개의 맵을 생성한다.
- 채널 사용을 위해선 필수.
- 그 밖에도 수백 수천개의 인스턴스 맵이 생성될 수 있다.
- 메모리를 절약하고 성능을 높이려면 반드시 필요하다.



- **하나의 리소스 여러개의 맵 인스턴스**
  고정적인 지형지물은 하나의 리소스를 참조해서 사용.
  움직이는 오브젝트들은 새로 생성해서 사용.

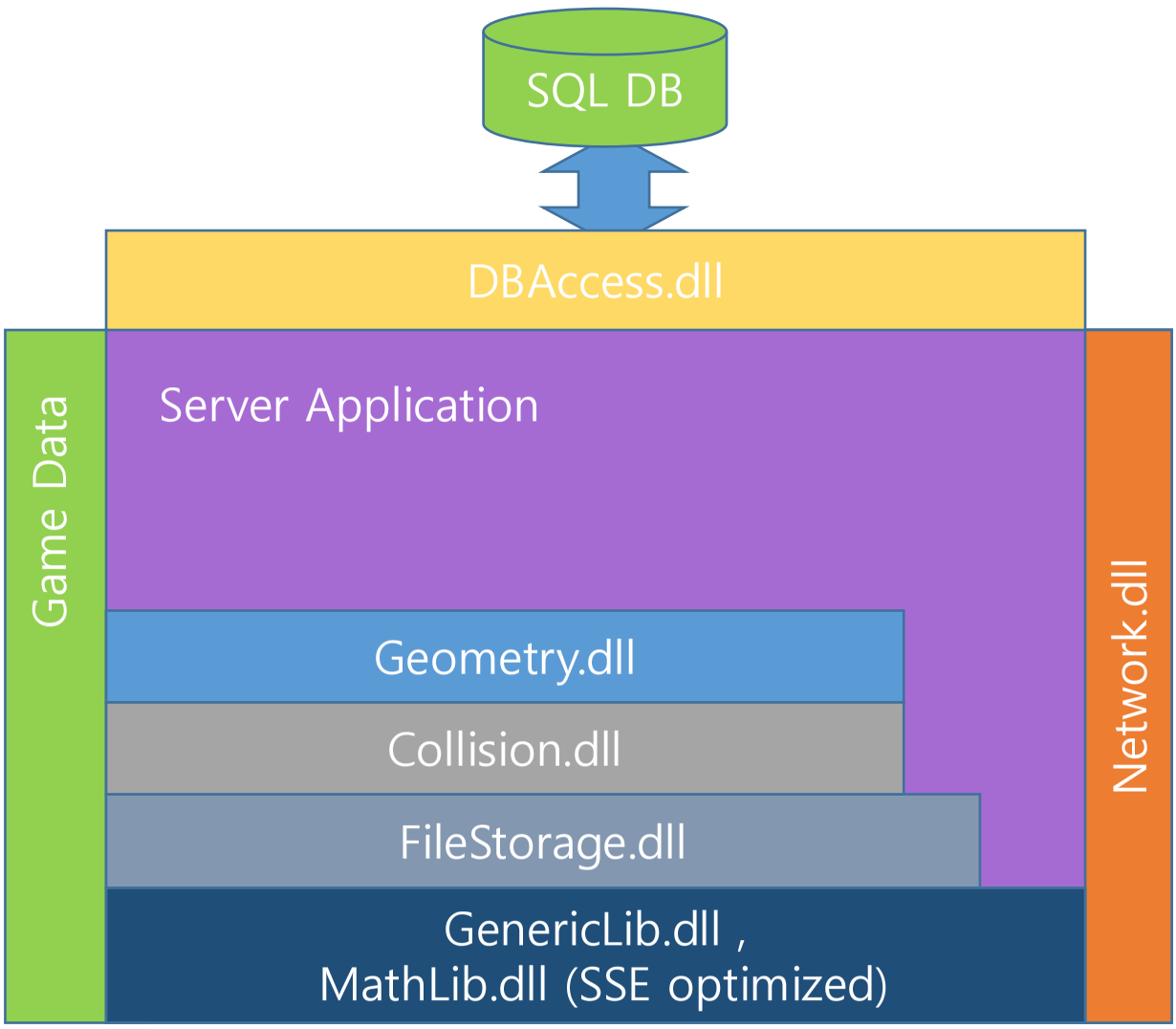
서버를 위해 꼭 짚고 넘어갈 사항

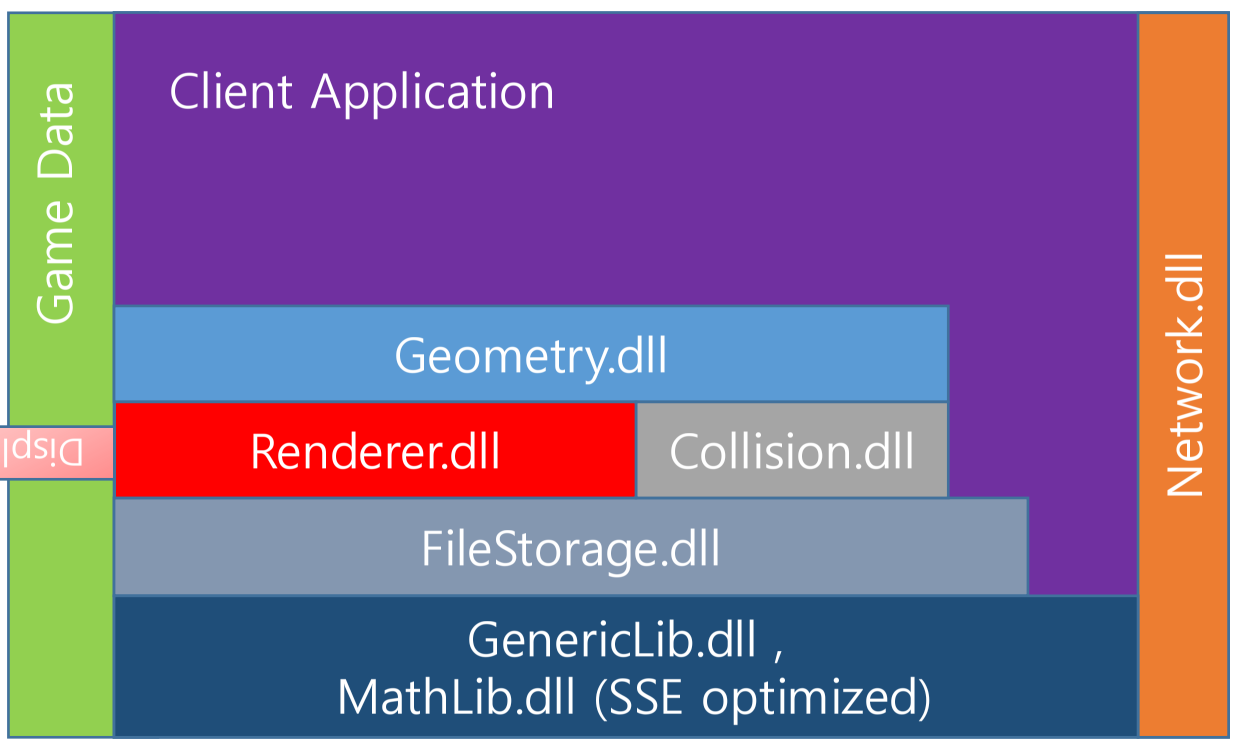
- 개별 기능들을 병렬로 multiple 처리할 수 있어야한다.
  - 클라이언트에선 1-1000개 정도 병렬 처리.
  - 서버에선 1-10000개 정도 병렬 처리
- Multiple로 처리한다고 메모리를 무작정 x N으로 먹으면 안된다.
- 그리고 GPU나 CPU자원을 최대한 활용할 수 있어야 한다.

하여간 만들어보았다.

모듈 구성

- MathLib.dll – 수학함수
- GenericLib.dll – 자료구조,검색,정렬,Memory Pool, Heap등.
- FileStorage.dll – pakcing된 파일과 일반파일을 동시 억세스 하기 위한 파일시스템.
- Renderer.dll – DirectX,OpenGL등 그래픽 API를 직접 호출하고 상위 레이어에 대해 독립적인 렌더링 기능을 제공한다.
- Geoemtry.dll – 오브젝트 관리,맵 관리, 트리거, 충돌처리 등등 직접 화면에 렌더링만 안하는 게임 엔진 본체
- Collision.dll – 충돌처리 및 오브젝트와 속도벡터를 넣었을때 최종 위치와 속도벡터를 산출해주는 이동처리 엔진. GPGPU지원을 위해 따로 분리했다.
- Network.dll – IOCP기반, 서버와 클라이언트 공용 네트워크 엔진.
- DBAccess.dll – OLEDB기반, SQL서버와 통신. 비동기/동기 모드 양쪽 모두 지원.


SQL DB
DBAccess.dll
Game Data
Server Application
Network.dll
Geometry.dll
Collision.dll
FileStorage.dll
GenericLib.dll ,
MathLib.dll (SSE optimized)


Game Data
Client Application
Network.dll
Geometry.dll
Display
Renderer.dll
Collision.dll
FileStorage.dll
GenericLib.dll ,
MathLib.dll (SSE optimized)

엔진의 API 인터페이스 예시

게임 맵생성-클라이언트/서버 공용

```
virtual ULONG __stdcall CreateSceneGraph(
                ISceneGraph** ppGraph, DWORD dwFlag
                ) = 0;
```

맵 로딩-클라이언트용

```
virtual DWORD __stdcall ReadFile(
                char* szFileName, ProgressCallBack pFunc
,               char* szCommonObjectPath,
                DWORD dwFlag) = 0;
```

맵 로딩-서버용

```
virtual DWORD__stdcall ReadFileWithoutMesh(
                char* szFileName, char* szCommonObjectPath,
                DWORD dwFlag) = 0;
```

캐릭터 생성-클라이언트/서버 공용

```
virtual ICharacterObject*__stdcall CreateCharacterObject(
        char* szFileName, GXSchedulePROC pProc,
        VECTOR3* pv3InitPos,
        void* pData,
        char** ppExtModFileNameList,
        DWORD dwExtModFileNum,
        DWORD dwFlag) = 0;
```

서버에서 사용시

```
dwFlag = GXOBJECT_CREATE_TYPE_NOT_USE_MODEL;
```

캐릭터 위치 설정 – 서버/클라이언트 공용

```
virtual void __stdcall SetPosition(
        VECTOR3* pv3Pos,
        BOOL bDoInterpolation) = 0;
```

충돌처리를 적용한 캐릭터 이동 – 서버/클라이언트 공용

Non-block mode요청

```
virtual DWORD__stdcall QueryCollisionTest(
        COLLISION_REACTION_TYPE reactionType,
        float Elasticity,
        VECTOR3* pVel) = 0;
```

Block mode 결과획득

```
virtual BOOL __stdcall CollisionCompleteTest(
        DWORD dwQueryIndex,
        COLLISION_TEST_RESULT* pOutColResult,
        BOOL bInterpolation) = 0;
```

Project D Online에서의 적용

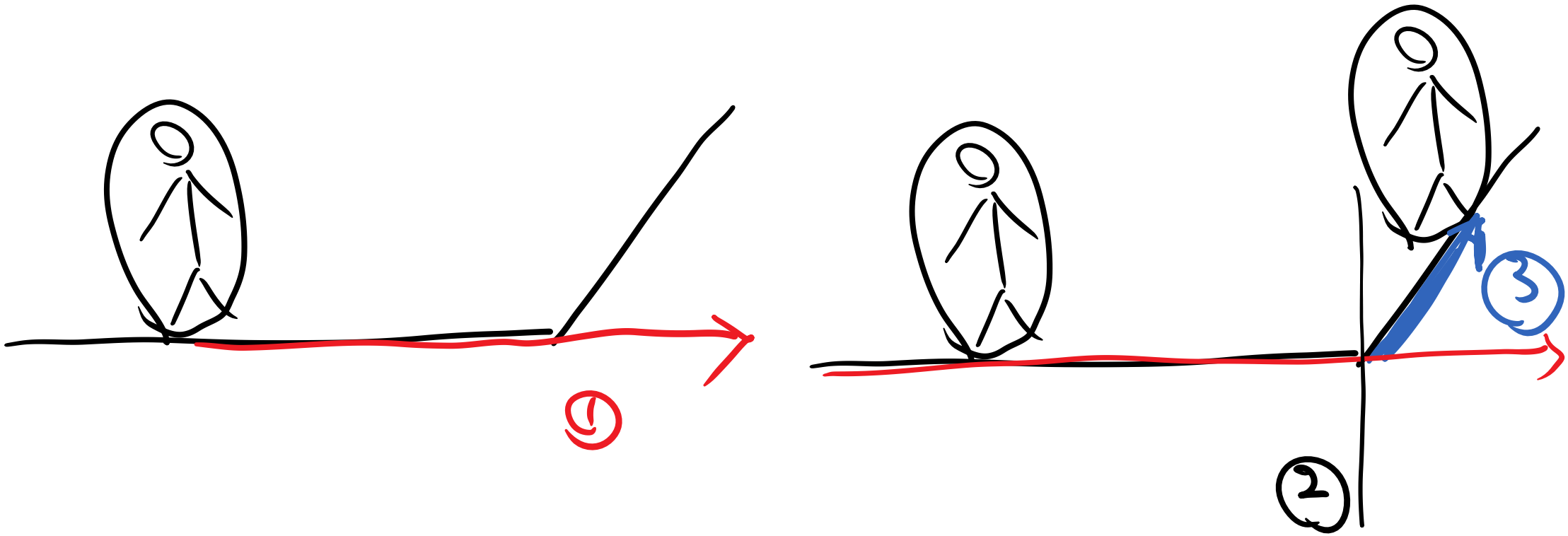
http://www.projectd-online.com
http://www.facebook.com/projectdonline

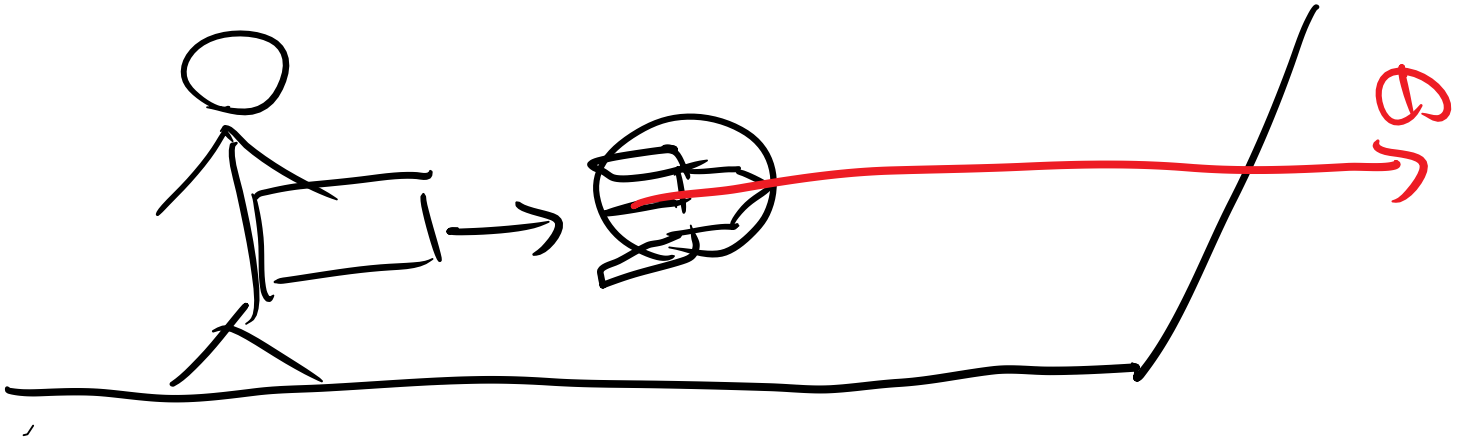
사연이 길지만 각설하고...
이 프로젝트의 큰 목적중 하나는 서버사이드 충돌처리 기술의 검증이었다.

Project D Online에서의 이동

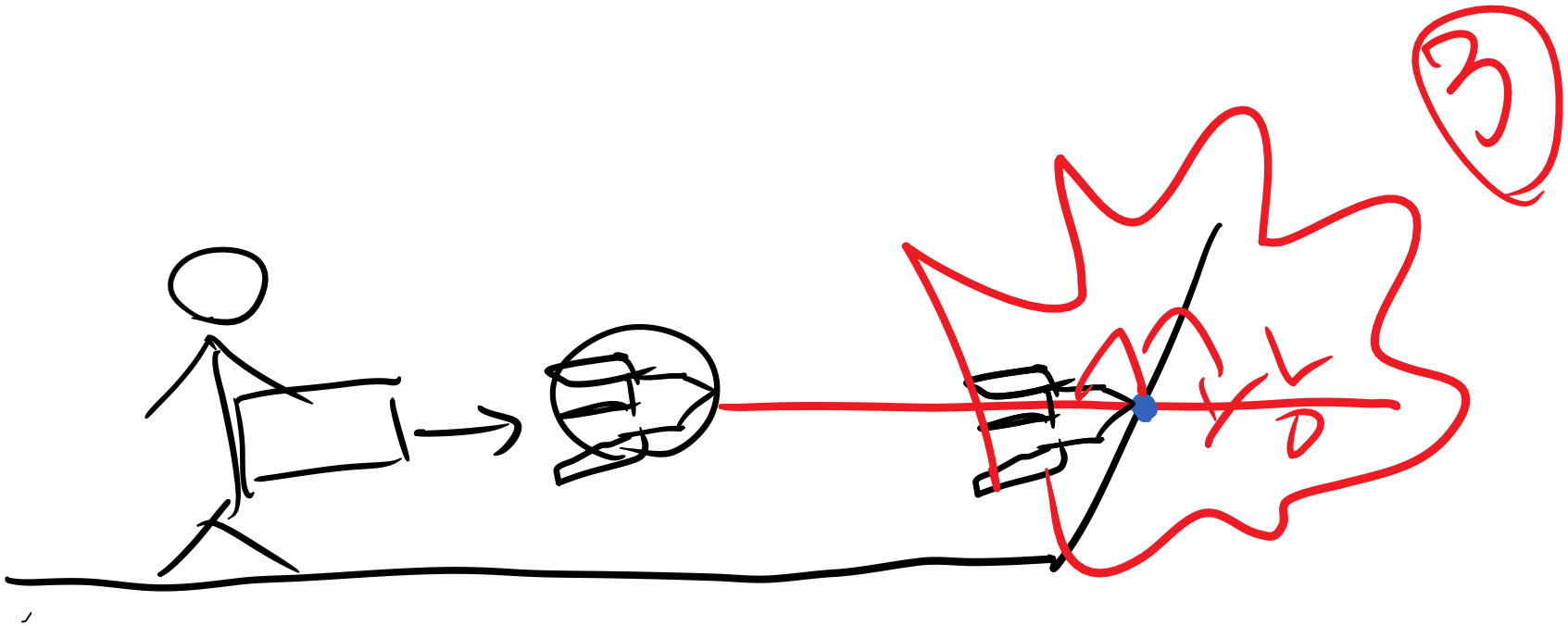
- 지형을 따라 뛰어다닌다.
- 일정 각도의 삼각형과 충돌하면 미끄러진다.
- 일정 각도의 타원체와 충돌하면 미끄러진다.
- 로켓탄은 캐릭터 한마리와 동등하게 처리한다.
- 로켓탄은 삼각형이나 타원체나 어디든 충돌하면 폭발한다.

1. 정지상태에서 속도벡터만큼 이동 지도
2. 충돌감지.
3. 비끄러짐 벡터를 구해서 새로운 속도 벡터로 이동 시도





1. 로켓탄 발사
2. 충돌. 그대로 정지
3. 폭발처리



Project D Online의 전투

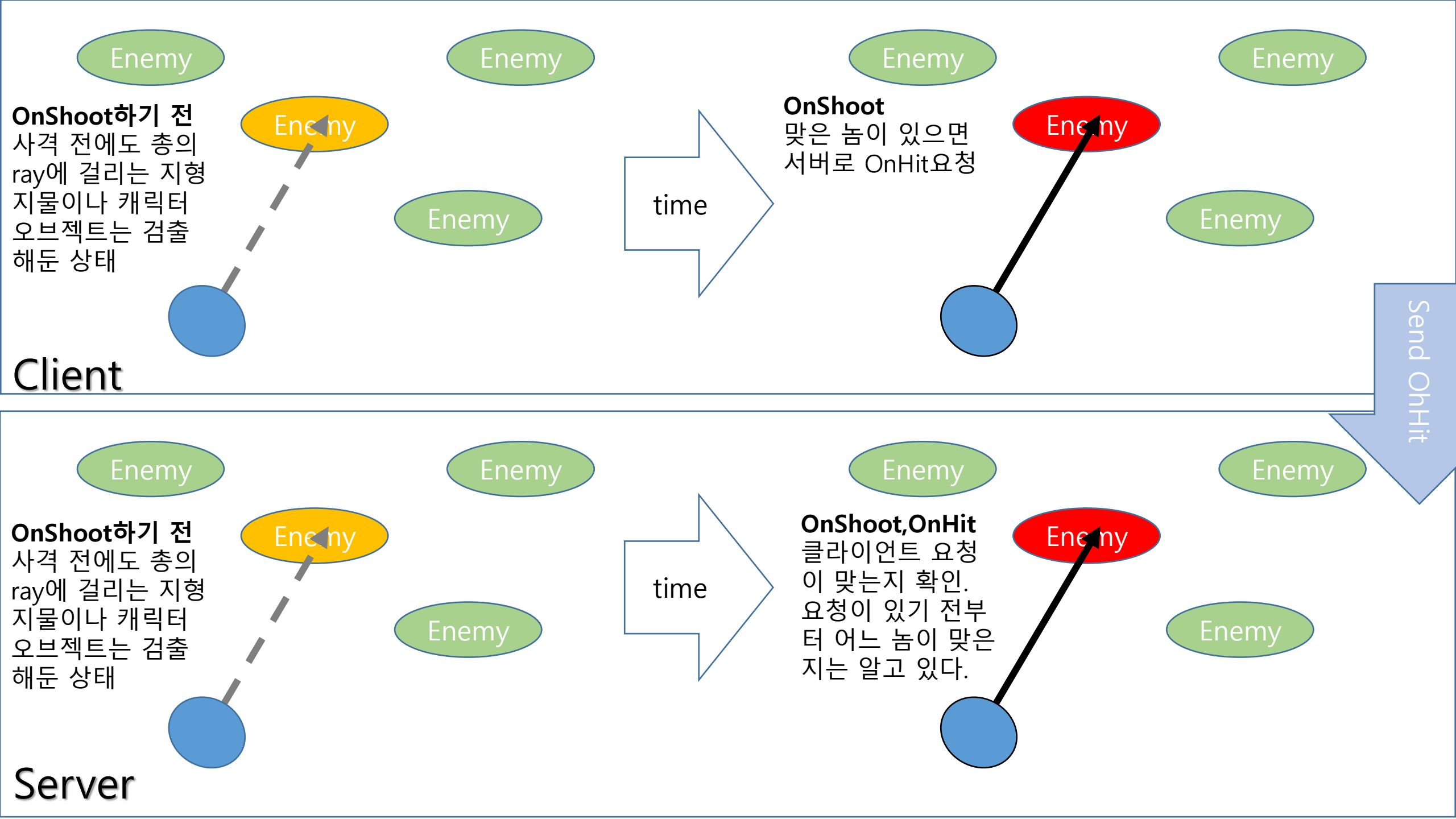
- 총으로 쏜다.
- 칼로 때린다.
- 로켓탄이 날아가서 충돌시 폭발한다.

SMG(퀘이크의 머신건)

1. 클라이언트에서 OnShoot -> 서버로 패킷 전송
2. 매 프레임마다 OnShootingProcessGun()호출
3. 총의 ray에 대해서 충돌하는 오브젝트는 이미 찾아서 저장해두고 있다.
4. 총의 ray에 충돌하는 캐릭터가 있으면 서버에 타격 판정을 요청한다.
5. 서버에서 패킷 수신서버에서 패킷을 수신
6. 서버 역시 30프레임으로 총의 ray에 걸치는 캐릭터들을 이미 찾아서 저장해두고 있다.
7. 플레이어의 요청에 따라 ray에 걸친 캐릭터가 있는지 확인.클라이언트의 요청이 사실로 판별되면 Hit처리.

라이플(퀘이크의 레일건)

1. 클라이언트에서 OnShoot -> 서버로 패킷 전송
2. OnShootingProcessGun()호출
3. 총의 ray에 대해서 충돌하는 오브젝트는 이미 찾아서 저장해두고 있다.
4. 총의 ray에 충돌하는 캐릭터가 있으면 서버에 타격 판정을 요청한다.
5. 서버에서 패킷 수신서버에서 패킷을 수신
6. 서버 역시 30프레임으로 총의 ray에 걸치는 캐릭터들을 이미 찾아서 저장해두고 있다.
7. 플레이어의 요청에 따라 ray에 걸친 캐릭터가 있는지 확인.클라이언트의 요청이 사실로 판별되면 Hit처리.

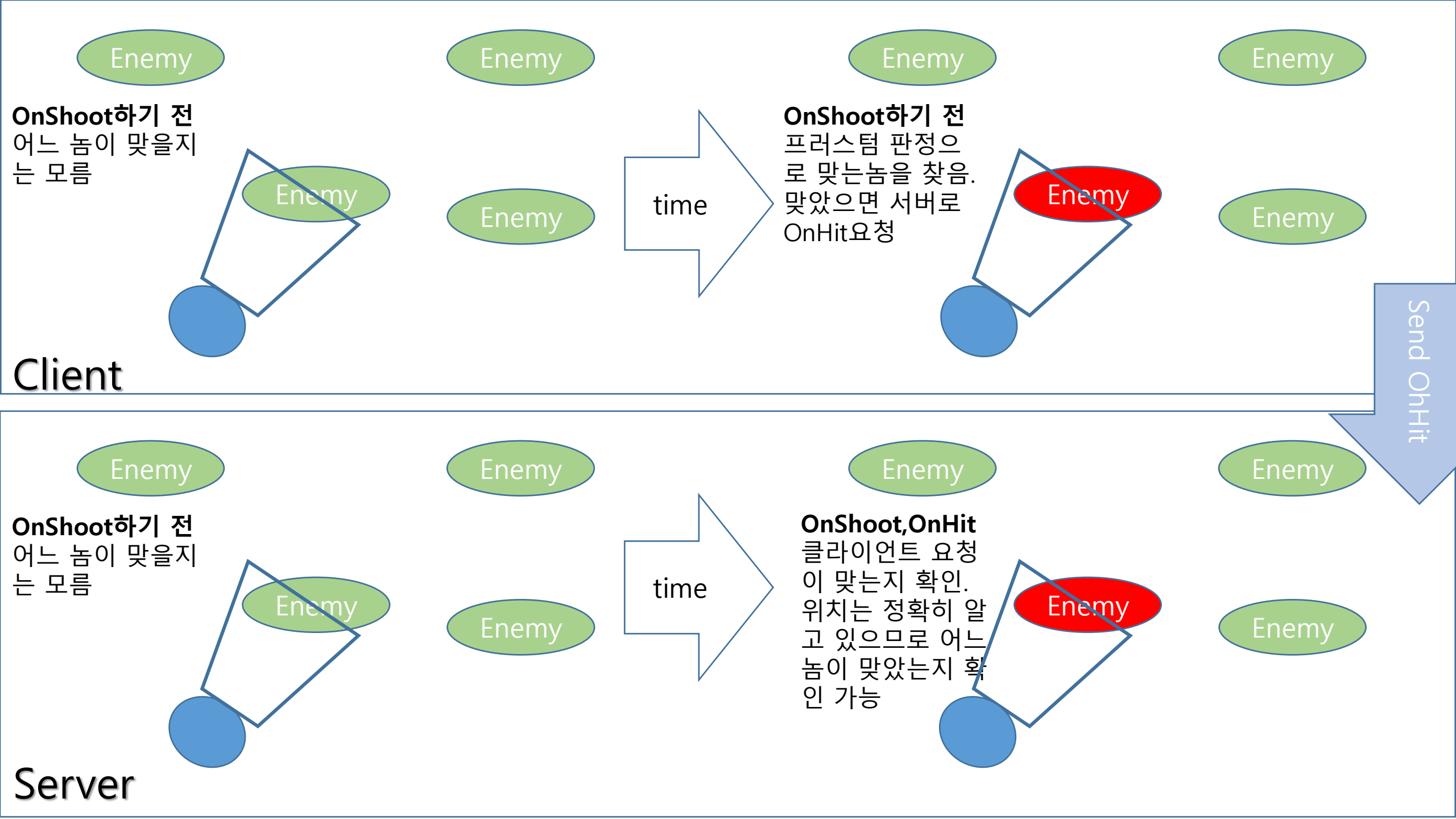

Enemy
Enemy
Enemy
Enemy
OnShoot하기 전
사격 전에도 총의 ray에 걸리는 지형 지물이나 캐릭터 오브젝트는 검출 해둔 상태
time
Enemy
Enemy
Enemy
Enemy
OnShoot
맞은 놈이 있으면 서버로 OnHit요청
Send OhHit
Client
Enemy
Enemy
Enemy
Enemy
OnShoot하기 전
사격 전에도 총의 ray에 걸리는 지형 지물이나 캐릭터 오브젝트는 검출 해둔 상태
time
Enemy
Enemy
Enemy
Enemy
OnShoot,OnHit
클라이언트 요청이 맞는지 확인. 요청이 있기 전부터 어느 놈이 맞은지는 알고 있다.
Server

Bastard Sword(연타 불가능한 칼)

1. 클라이언트에서 OnShoot -> 서버로 패킷 전송
2. 캐릭터 중심으로 사다리꼴 프러스텀을 만든다.
3. SceneTree에서 프러스텀에 걸치는 오브젝트들을 찾아낸다.
4. 프러스텀에 들어가면 Hit -> 서버로 타격판정 요청
5. 서버에서 패킷을 수신
6. 클라이언트와 마찬가지로 해당 캐릭터의 위치에서 프러스텀에 걸치는 오브젝트들을 찾아낸다
7. 프러스텀에 들어가는 오브젝트들에 대해서 hit처리

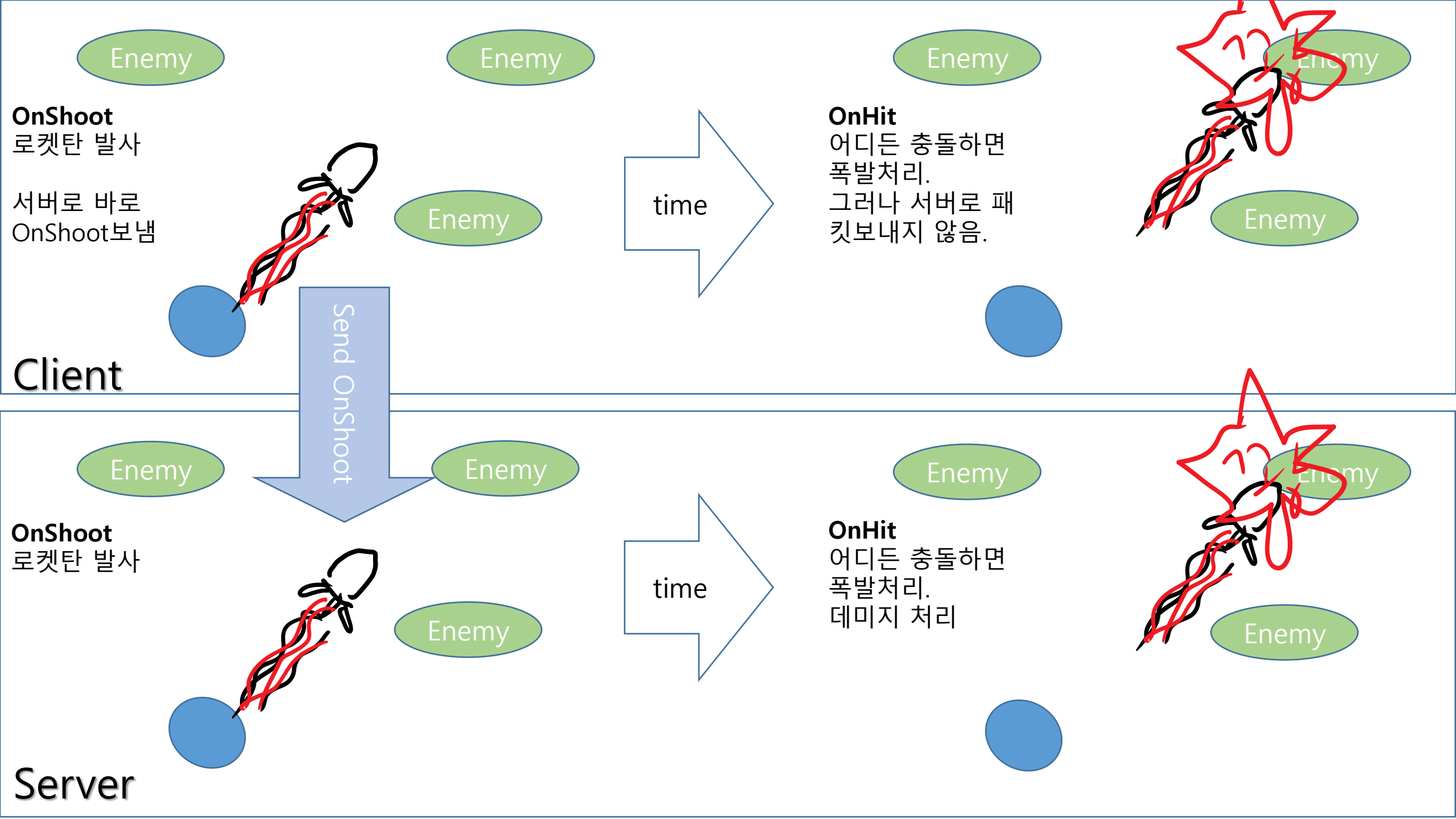
Saber(연타 가능한 칼)

1. OnShoot
2. 캐릭터 중심으로 사다리꼴 프러스텀을 만든다.
3. SceneTree에서 프러스텀에 걸치는 오브젝트들을 찾아낸다.
4. 프러스텀에 들어가면 Hit -> 서버로 타격판정 요청
5. 서버에서 패킷을 수신
6. 클라이언트와 마찬가지로 해당 캐릭터의 위치에서 프러스텀에 걸치는 오브젝트들을 찾아낸다
7. 프러스텀에 들어가는 오브젝트들에 대해서 hit처리


Enemy
Enemy
Enemy
Enemy
OnShoot하기 전
어느 놈이 맞을지
는 모름
Enemy
Enemy
time
OnShoot하기 전
프러스텀 판정으
로 맞는놈을 찾음.
맞았으면 서버로
OnHit요청
Enemy
Enemy
Enemy
Client
Send OhHit
Enemy
Enemy
Enemy
Enemy
OnShoot하기 전
어느 놈이 맞을지
는 모름
Enemy
Enemy
time
OnShoot,OnHit
클라이언트 요청
이 맞는지 확인.
위치는 정확히 알
고 있으므로 어느
놈이 맞았는지 확
인 가능
Enemy
Enemy
Enemy
Server

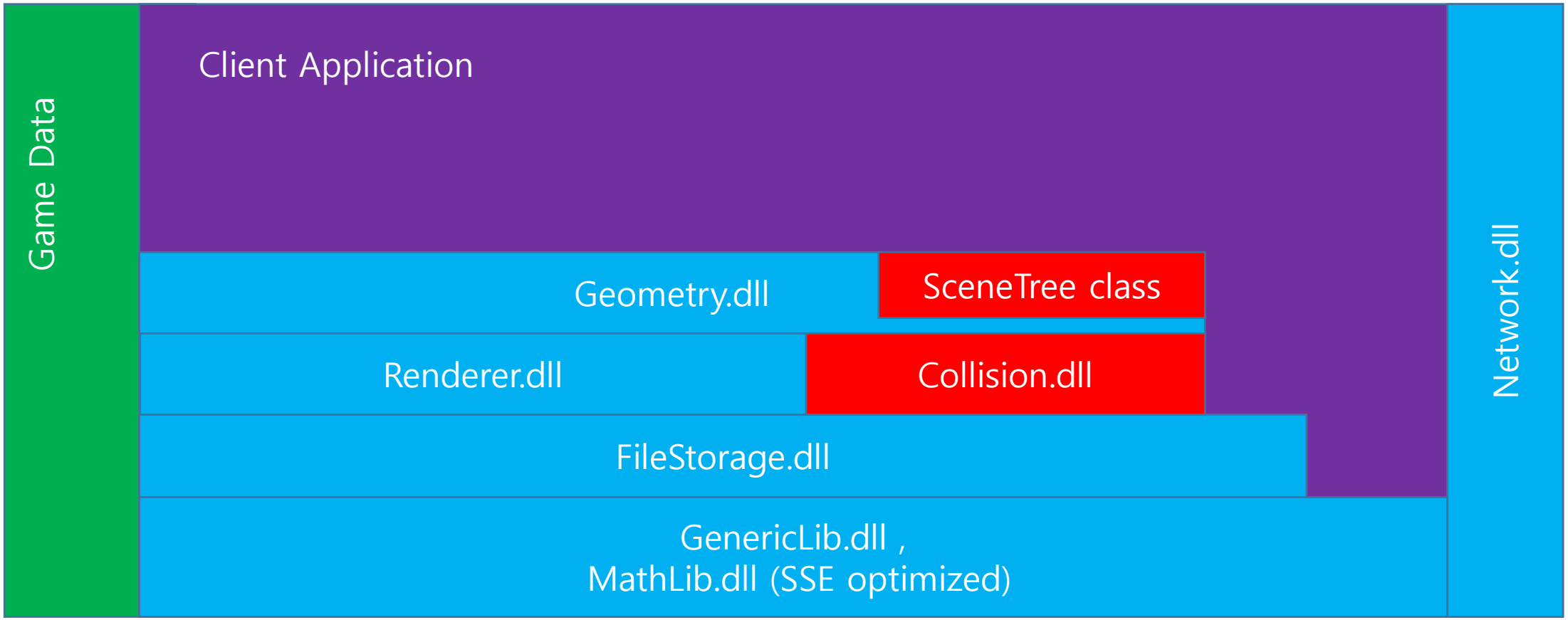
Rocket Launcher

1. 클라이언트에서 OnShoot호출 -> 서버로 패킷 전송
2. 중력상수가 0이고 속도가 꽤 빠른 캐릭터 오브젝트(로켓탄)를 만든다.
3. 만들자마자 충돌처리 엔진에 바로 등록된다.
4. 서버에서 패킷 수신
5. 서버에서도 중력상수가 0이고 속도가 꽤 빠른 캐릭터(로켓탄)을 만든다.
6. 클라이언트와 서버 모두 로켓에 대한 이동&충돌처리를 동시에 진행한다.
7. 로켓탄이 캐릭터 혹은 지형지물에 충돌하면 폭발한다.
8. 서버와 클라이언트 모두 동일하게 작동하지만 클라이언트는 폭발 시 아무것도 하지 않는다. 애니메이션 처리만 한다.
9. 서버의 경우 로켓탄이 폭발하면 폭발 위치에서 폭발 범위에 들어가는 캐릭터 오브젝트들을 찾는다.
10. 찾은 캐릭터들에 대해 거리별로 다이렉트 데미지와 스플래쉬 데미지를 적용한다.


Enemy
Enemy
Enemy
Enemy
OnShoot
로켓탄 발사
서버로 바로
OnShoot보냄
time
OnHit
어디든 충돌하면
폭발처리.
그러나 서버로 패
킷보내지 않음.
Send OnShoot
Client
Enemy
Enemy
Enemy
Enemy
OnShoot
로켓탄 발사
time
OnHit
어디든 충돌하면
폭발처리.
데미지 처리
Server

이동&충돌&타격 판정의 중요 모듈

1. Collision.dll (충돌처리 엔진)
2. SceneTree class (오브젝트 픽킹 및 검색 엔진)



Collision.dll (충돌처리 엔진)

- 3차원 그리드 자료구조 – KD Tree를 사용하지 않고 그리드구조를 사용한 이유는 GPGPU최적화 때문
- 멀티 스레드
- CUDA 동시 지원
- 삼각형에 충돌했는지 타원체에 충돌했는지 비트플래그 리턴
- 최종 속도벡터 리턴
- 로켓탄이 어딘가에 충돌했을때 타원체 충돌했다는 비트가 켜지면 로켓탄의 데미지 범위로 SceneTree로부터 캐릭터 오브젝트 탐색.

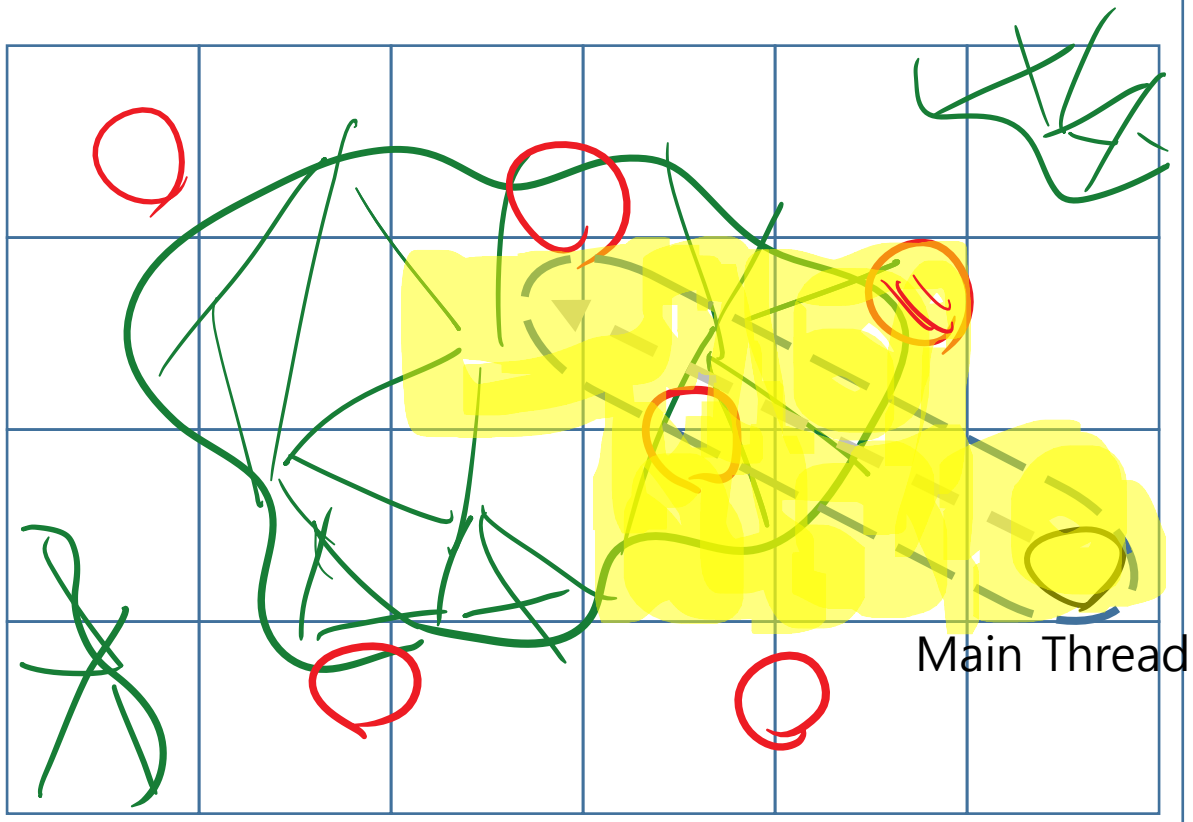
충돌처리 기본 컴포넌트

- 움직이는 타원체 vs 삼각형
- 움직이는 타원체 vs 타원체
- 움직이는 타원체 vs 움직이는 타원체

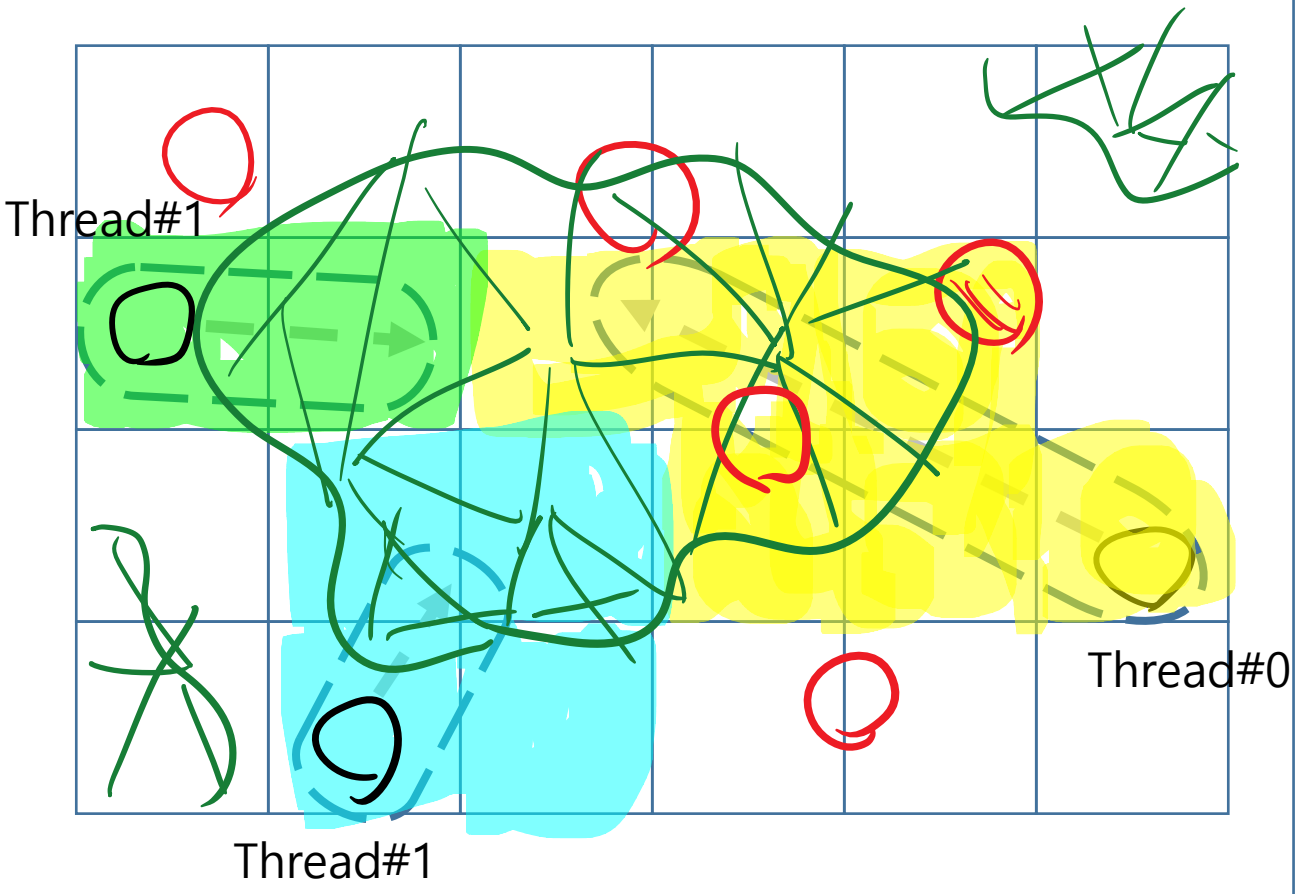
충돌처리 기본 리액션

- 충돌시 미끄러짐
- 충돌시 정지
- 충돌시 반사

Single Thread

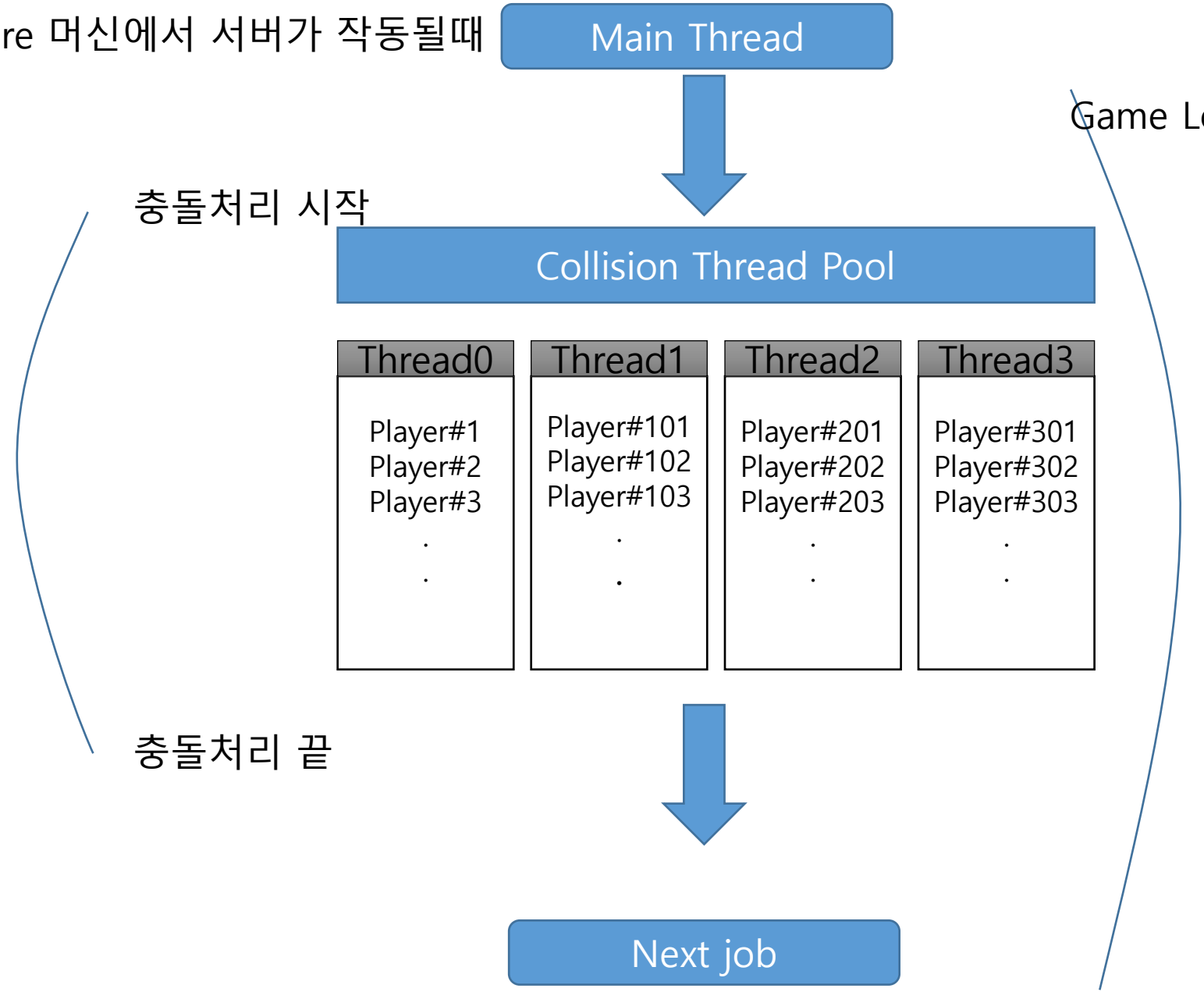


Multi Threads



Multi Core 지원

4Core 머신에서 서버가 작동될때

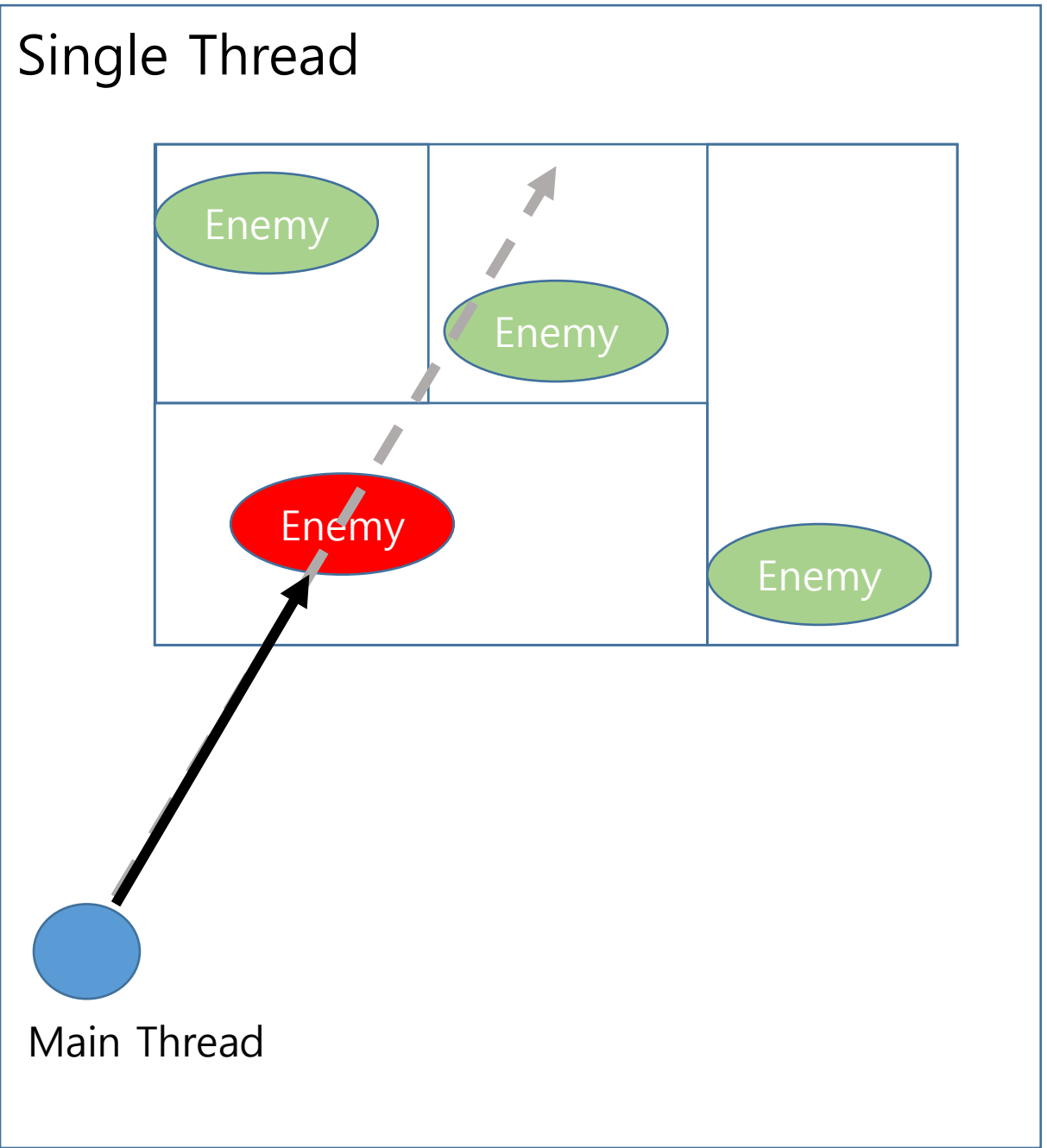


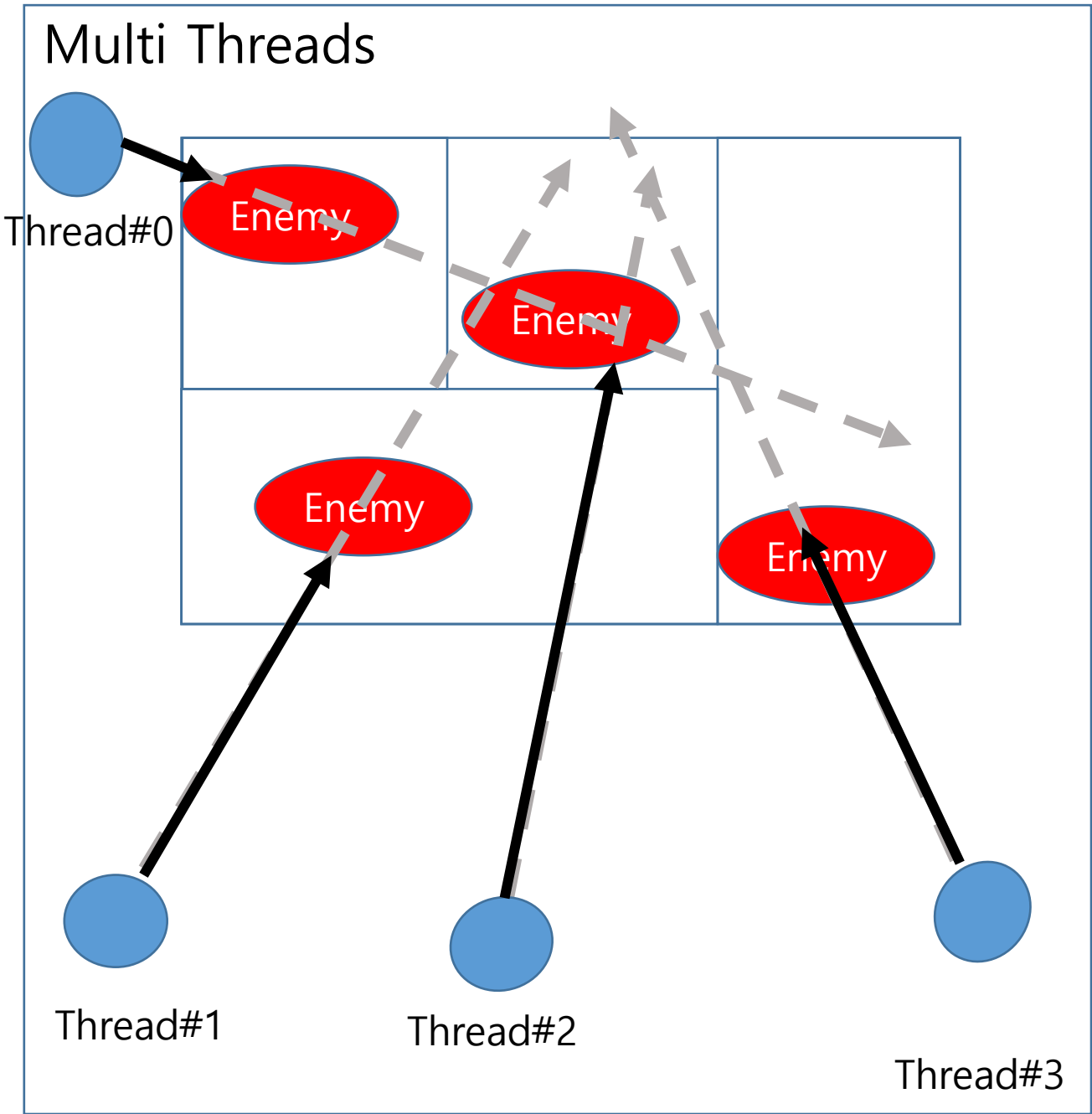
1.Core개수만큼 충돌처리 스레드를 만들어둔다(Thread Pool)
2.충돌처리 타이밍이 되면 스레드풀의 스레드들이 플레이어들의 현재 위치, 속도벡터,시간을 기준으로 위치 계산을 시작한다.
3. 메인스레드는 스레드풀의 모든 스레드가 계산을 완료할때까지 대기한다.
4.모든 충돌처리가 끝나면 메인스레드가 완료된 위치정보를 갱신한다.

SceneTree Class (픽킹 및 오브젝트 검색)

- 멀티 스레드
- KD Tree – KD Tree Traversal
- 구,타원체로 오브젝트 검색
- Ray로 삼각형 또는 오브젝트 검색
- Ex) 총들고 다닐때 미리 Ray에 걸치는 오브젝트들을 검색해둔다.
- Ex) 로켓탄이 폭발했을때 로켓탄 주변의 오브젝트들을 찾아낸다.

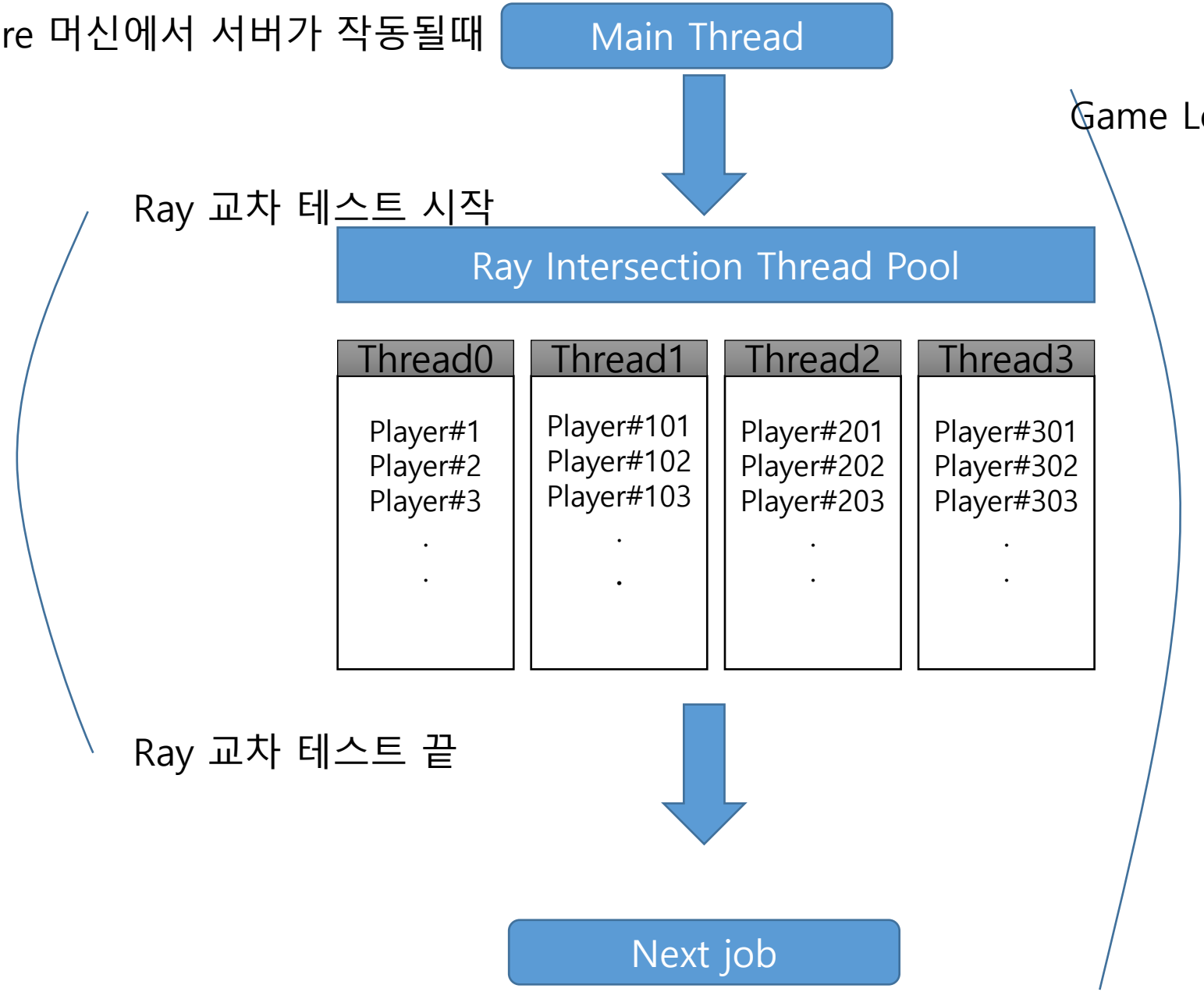
총 들고 다닐때


Single Thread
Enemy
Enemy
Enemy
Enemy
Main Thread


Multi Threads
Thread#0
Enemy
Enemy
Enemy
Enemy
Thread#1
Thread#2
Thread#3

Multi Core 지원

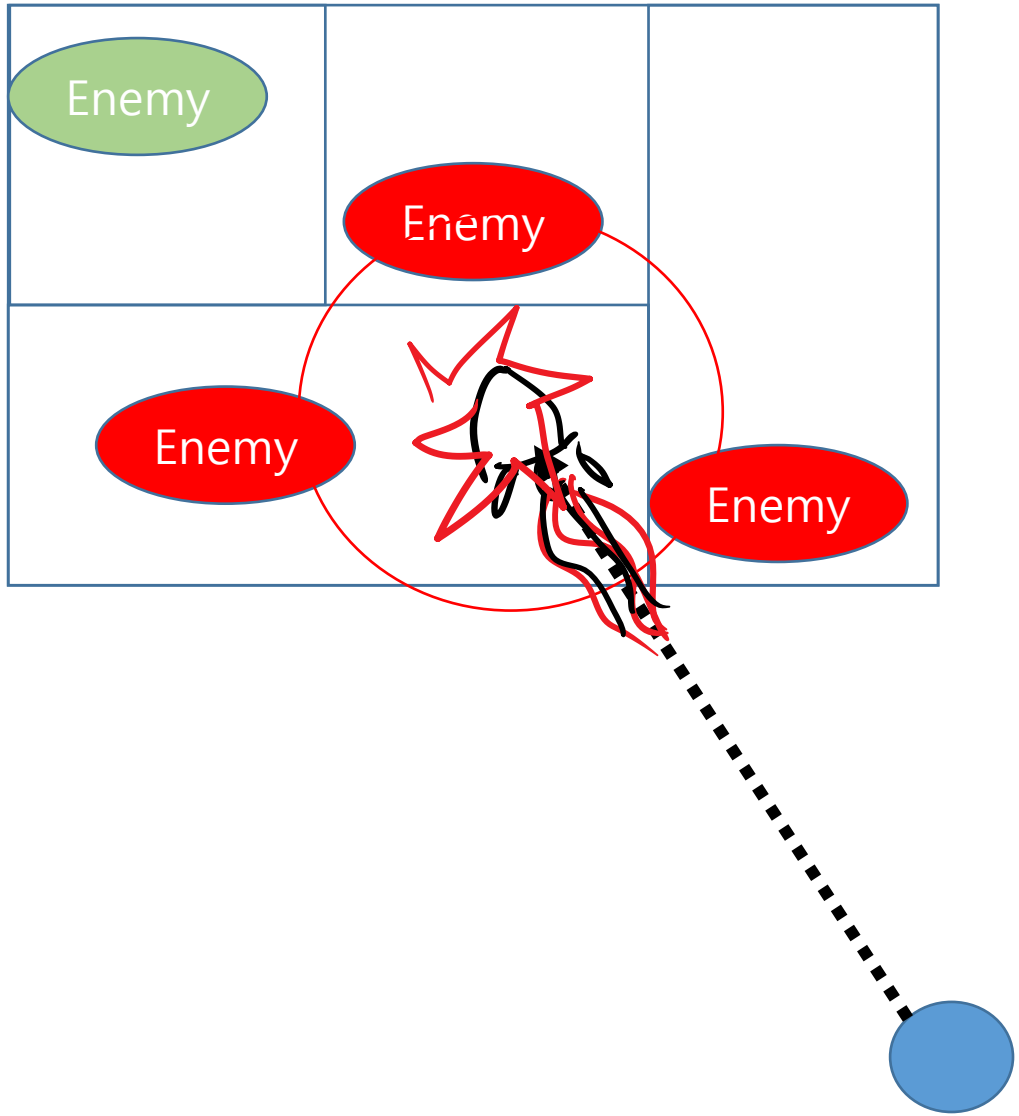
4Core 머신에서 서버가 작동될때



1.Core개수만큼 충돌처리 스레드를 만들어둔다(Thread Pool)
2.Ray 테스트 타이밍이 되면 스레드풀의 스레드들이 플레이어들이 들고 있는 총의 Ray에 걸치는 오브젝트와 지형지물에 대한 충돌점을 계산한다.
3. 메인스레드는 스레드풀의 모든 스레드가 계산을 완료할때까지 대기한다.
4.모든 충돌처리가 끝나면 메인스레드가 완료된 Ray Intersection 정보를 갱신한다.

로켓탄이 터졌을때

이벤트 드리븐이라 Single Thread로만 작동



초반에 예측하지 못했던 난관

- 패킷량 과부하
- 코딩작업량

패킷량 폭주 – 빈번한 마우스 이벤트

- 클라이언트에서 이동정보에 변화가 생기면
  - [현재 위치,움직일 방향(마우스회전),속력]을 서버로 보낸다
- 그런데 마우스를 마구 흔드는 등 이동상태를 자주 바꾸면 패킷 전송 회수가 많아지고 패킷 전송회수가 많아지면 패킷량이 폭주한다.
- 가장 큰 문제가 되는 경우는 마우스를 좌우로 흔드는 행위

NAgle알고리즘의 사용은 불가

- TCP에서 제공하는 NAgle알고리즘을 사용하면 간단히 해결 가능하다. 그러나 WSAD형태의 온라인 게임에선 NAgle 알고리즘을 사용할 수 없다. 이런 류의 게임은 응답성이 중요한데 NAgle 알고리즘을 사용하면 최대 500ms의 지연이 발생하기 때문이다. 즉 내가 총을 쏘면 최대 0.5초 후에 총을 발사했다는 패킷이 발송된다.

- W버튼을 눌렀다가 떼는 경우도 클라이언트에선 미세하게 전진하는 것으로 보이지만 서버에서는 OnForward,OffForward를 동시에 수신하므로 전혀 움직이지 않은 것이 된다.

해결책 – 패킷량 과부하

1. 클라이언트측 키보드와 마우스 샘플링 레이트 제한
2. 다이나믹 샘플링 레이트
3. 서버측 우선순위 패킷 큐

마우스 이벤트 줄이기 – 샘플링 레이트 제한

- 마우스가 x,y축으로 움직일때 샘플링레이트를 제한한다.
- 게임 내부 프레임에 맞춰서 초당 60번이나 30번 마우스의 위치를 얻어서 이전 위치와의 변량을 측정한다.
- 이렇게 하면 플레이어가 마우스를 아무리 흔들어도 초당 30번의 마우스 이벤트만 발생하는게 된다.

마우스 이벤트 줄이기
– 가변 마우스 샘플링 레이트 도입

- 주변에 다른 플레이어가 많으면 내가 보내는 패킷이 많은 플레이어들에게 브로드캐스팅 되어야하므로 서버에서 송출되는 패킷량이 늘어나게 된다.
- 일반적으로 주변에 다른 플레이어가 많아지면 게임 렌더링 성능이 떨어지기 마련이다. 따라서 이런 경우 마우스의 입력 샘플링 레이트를 낮춰도 유져는 잘 느끼지 못한다.
- 주변 플레이어의 수가 30명 이하, 60명 이하, 120명 이하인 경우에 맞춰서 마우스 샘플링 레이트도 30프레임, 15프레임,10프레임으로 조정한다.

서버측 브로드캐스팅 패킷량 폭주

- 한 섹터 안에 플레이어가 100명 있다면 그 중 한명의 플레이어가 움직여서 초당 1KBytes의 패킷을 발생시켰을 경우 서버는 1KBytes * 99 = 99KBytes의 패킷을 전송해야한다. 100명의 플레이어가 모두 동시에 움직였다면 1KBytes * 99 * 100 = 9900KB로 약 10MB의 패킷을 전송해야한다.
- 개별 클라이언트가 보내는 패킷의 양을 줄이는 것도 중요하지만 서버에서 브로드캐스팅하는 양을 줄이는 것도 중요하다.
- 패킷 전송 회수를 줄이는게 가장 확실하다. 일반적인 게임 패킷은 한 단위가 50Bytes를 넘지 않지만 TCP헤더 사이즈가 최대 60Bytes 달하기 때문에 패킷을 자주 전송하면 배보다 배꼽이 더 큰 상황이 발생한다.일정시간 동안 패킷을 모아서 한번에 보내는 TCP헤더로 차지하는 양을 크게 줄일 수 있다.

패킷이라고 다 같은 패킷이 아니다!

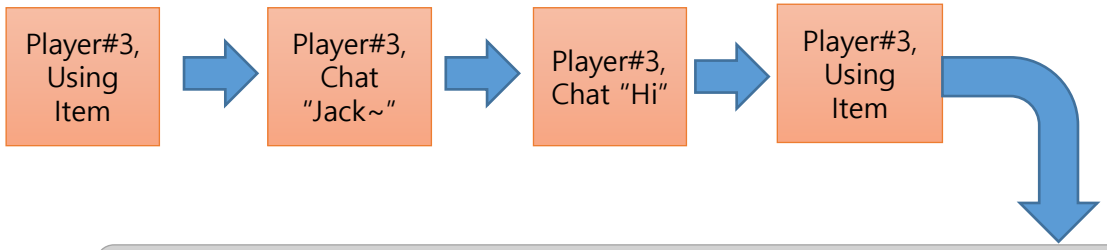
- 어떤 패킷은 최대 0.5초 정도의 지연이 있어도 별 문제가 되지 않는다. 채팅 메시지나 아이템을 장착하는 패킷이라든가 하는 것들이다. 뛰고 점프하고 방향을 바꾸고 공격을 하는 패킷들 외에는 대부분 어느 정도의 지연이 있어도 문제가 되지 않는다.

- 따라서 서버에서 이러한 패킷들을 감별해서 바로 보낼지 모았다가 한번에 보낼지를 결정한다면 패킷 전송 회수를 줄일수 있고 패킷량도 크게 줄어든다.

서버측 패킷 우선순위큐 - 전략

- 서버에서 클라이언트에 패킷을 전송할때 어떤 패킷도 곧바로 전송하지는 않는다.
- Queue를 만들어두고 일단 push한 후 나중에 Flush()함수를 호출해서 Queue의 내용을 전송하고 Queue를 비운다.
- Client로부터 패킷을 받아서 요청을 처리한 직후와 초당 30번 돌아가는 게임루프의 끝에서 Flus()함수를 호출한다.
- 이동,전투 관련 패킷은 긴급패킷으로 분류한다. 그 외에는 모두 일반 패킷이다.
- Flush()함수 호출 시점에서 긴급패킷이 Queue에 한개라도 있으면 Queue의 내용을 모두 전송한다.
- 긴급패킷이 없으면 최대 500ms패킷을 모았다가 전송한다.

Client#3

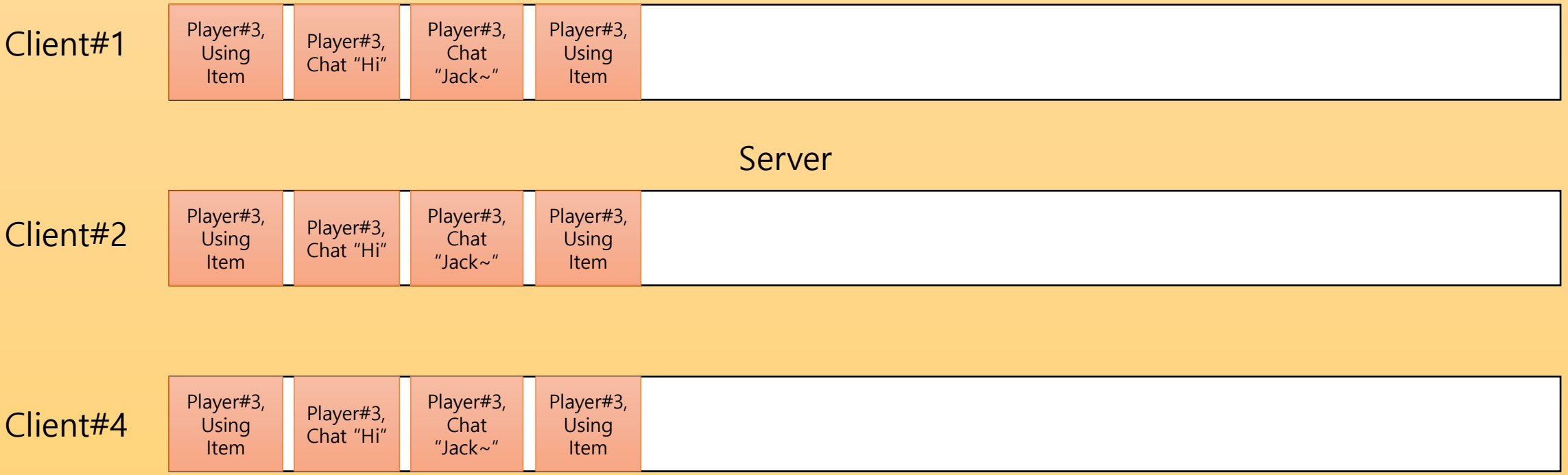


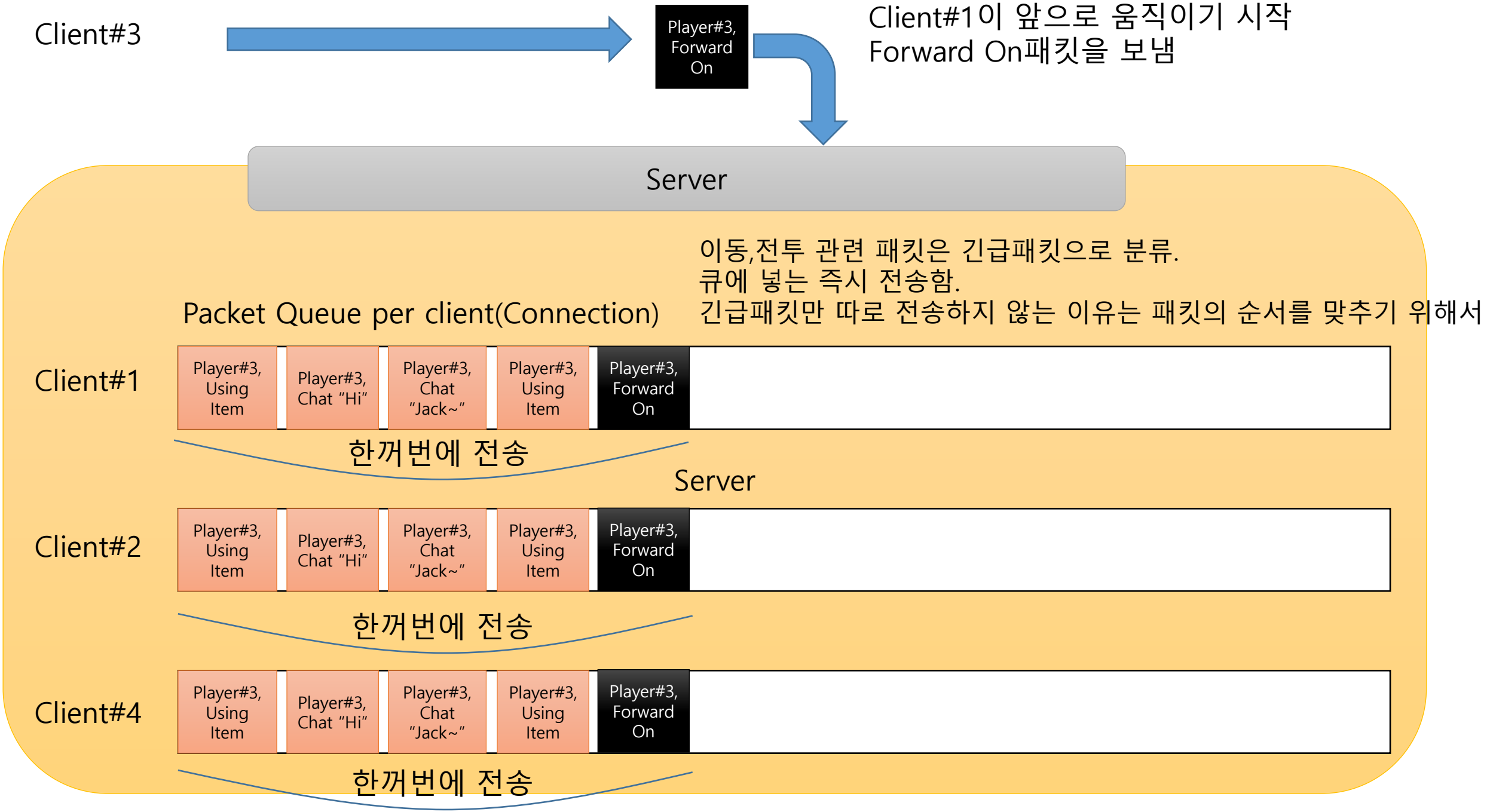
Client#1에서 연속으로 패킷 전송
아이템 사용 , 채팅:Hi, 채팅:Jack~, 아이템 사용

Server

서버에서는 각 클라이언트에 브로드캐스팅.
바로 전송하지 않고 큐에 먼저 쌓아둠.

Packet Queue per client(Connection)




Client#3
Player#3, Forward On
Client#1이 앞으로 움직이기 시작
Forward On패킷을 보냄
Server
이동,전투 관련 패킷은 긴급패킷으로 분류.
큐에 넣는 즉시 전송함.
긴급패킷만 따로 전송하지 않는 이유는 패킷의 순서를 맞추기 위해서
Packet Queue per client(Connection)
Client#1
Player#3, Using Item
Player#3, Chat "Hi"
Player#3, Chat "Jack~"
Player#3, Using Item
Player#3, Forward On
한꺼번에 전송
Server
Client#2
Player#3, Using Item
Player#3, Chat "Hi"
Player#3, Chat "Jack~"
Player#3, Using Item
Player#3, Forward On
한꺼번에 전송
Client#4
Player#3, Using Item
Player#3, Chat "Hi"
Player#3, Chat "Jack~"
Player#3, Using Item
Player#3, Forward On
한꺼번에 전송

난관#2 - 코딩해야할 양이 많다.

- 그냥 죽어라 하면 된다.
- 많이 짜고 많이 테스트 하고.
- 의외로 혼자 짜는게 낫다.
- 해보니 할만함.

결론

- 10년전에 이게 정말 되는지를 테스트하고 싶었다.
- 된다. 확실하게 말할 수 있는데 된다.
- 이젠 아무도 Stream I/O기반의 온라인 게임을 만들려고 하지 않기 때문에 별 쓸모는 없을듯.
- 테스트 해보고 싶으면 http://www.projectd-online.com 으로.
- Windows Azure RAM 3.5GB, 2Core VM으로 일본서버 운영중.