

# 온라인 게임 서버

배경지식

정내훈

2018년도 2학기

한국산업기술대학교 게임공학부

# 내용

---

- 게임 서버 종류
- 게임서버를 위한 하드웨어와 소프트웨어

# 게임 서버의 분류

## ● 로비 서버 (for MOG)

- 예) 스타크래프트의 배틀넷
- 실제 게임을 하기 전에 같이 플레이 할 상대를 찾는 곳 (match making)
- 채팅 서버의 확장
- 실제 게임은 P2P로 이루어 짐
  - 최근에는 P2P가 아니라 Client/Server로 실제 게임을 구현하는 추세
- 길드, 클랜, 아이템 구매 지원

## ● MMOG 서버

- MUD의 서버가 효시
- 게임 컨텐츠가 다 서버에 들어 있음
- 서버의 부하가 크다
- 이번 학기 강의의 주 관심사

# 게임 서버의 설계

- 게임 서버의 설계 목표 (안정성, 성능)
  - 쾌적한 게임 환경
    - 랙 없는 서버, 안정적인 서버
  - 동접
    - **게임성** : 경제시스템, 커뮤니티
    - 서비스 비용 절약
  - 핫스팟
    - 게임 콘텐츠 제약 완화 : 이벤트, 공성전, 혈맹전
- 성능향상을 위한 해결책
  - 효율적인 프로그래밍
  - 부하 분산 (여러 대의 컴퓨터)

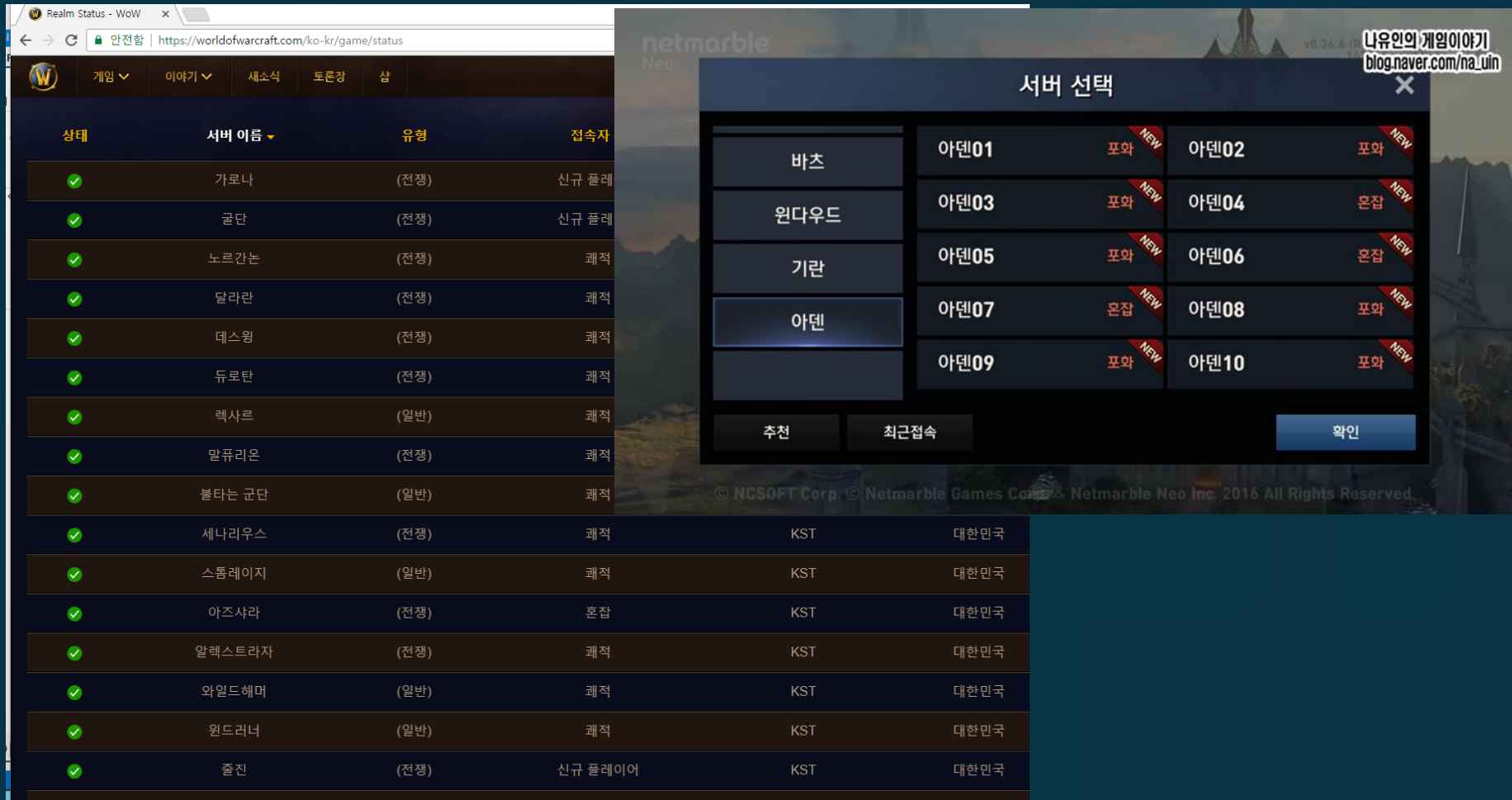
# 게임 서버의 설계

---

- 부하 분산
  - 서버 분리 (샤딩, Sharding)
  - 기능별 분산
  - 공간 분할(월드 분할)

# 부하 분산

## ● 서버 분리 (샤딩, Sharding)



The screenshot displays the World of Warcraft Realms Status page and a server selection window. The Realms Status page shows a list of servers with their status, names, types, and player counts. The server selection window shows a grid of servers with their names, status, and player counts.

상태	서버 이름	유형	전속자
✓	가로나	(전쟁)	신규 플레이어
✓	굴단	(전쟁)	신규 플레이어
✓	노르간논	(전쟁)	쾌적
✓	달라란	(전쟁)	쾌적
✓	데스윙	(전쟁)	쾌적
✓	듀로탄	(전쟁)	쾌적
✓	렉사르	(일반)	쾌적
✓	말퓨리온	(전쟁)	쾌적
✓	볼타는 군단	(일반)	쾌적
✓	세나리우스	(전쟁)	쾌적
✓	스톰레이지	(일반)	쾌적
✓	아즈사라	(전쟁)	혼잡
✓	알렉스트라자	(전쟁)	쾌적
✓	와일드해머	(일반)	쾌적
✓	윈드러너	(일반)	쾌적
✓	줄진	(전쟁)	신규 플레이어

서버 선택

바탕	아덴01	아덴02
원도우드	아덴03	아덴04
기란	아덴05	아덴06
아덴	아덴07	아덴08
	아덴09	아덴10

추천 최근접속 확인

© NCSOFT Corp. © Netmarble Games Corp. & Netmarble Neo Inc. 2016 All Rights Reserved.

# 부하 분산

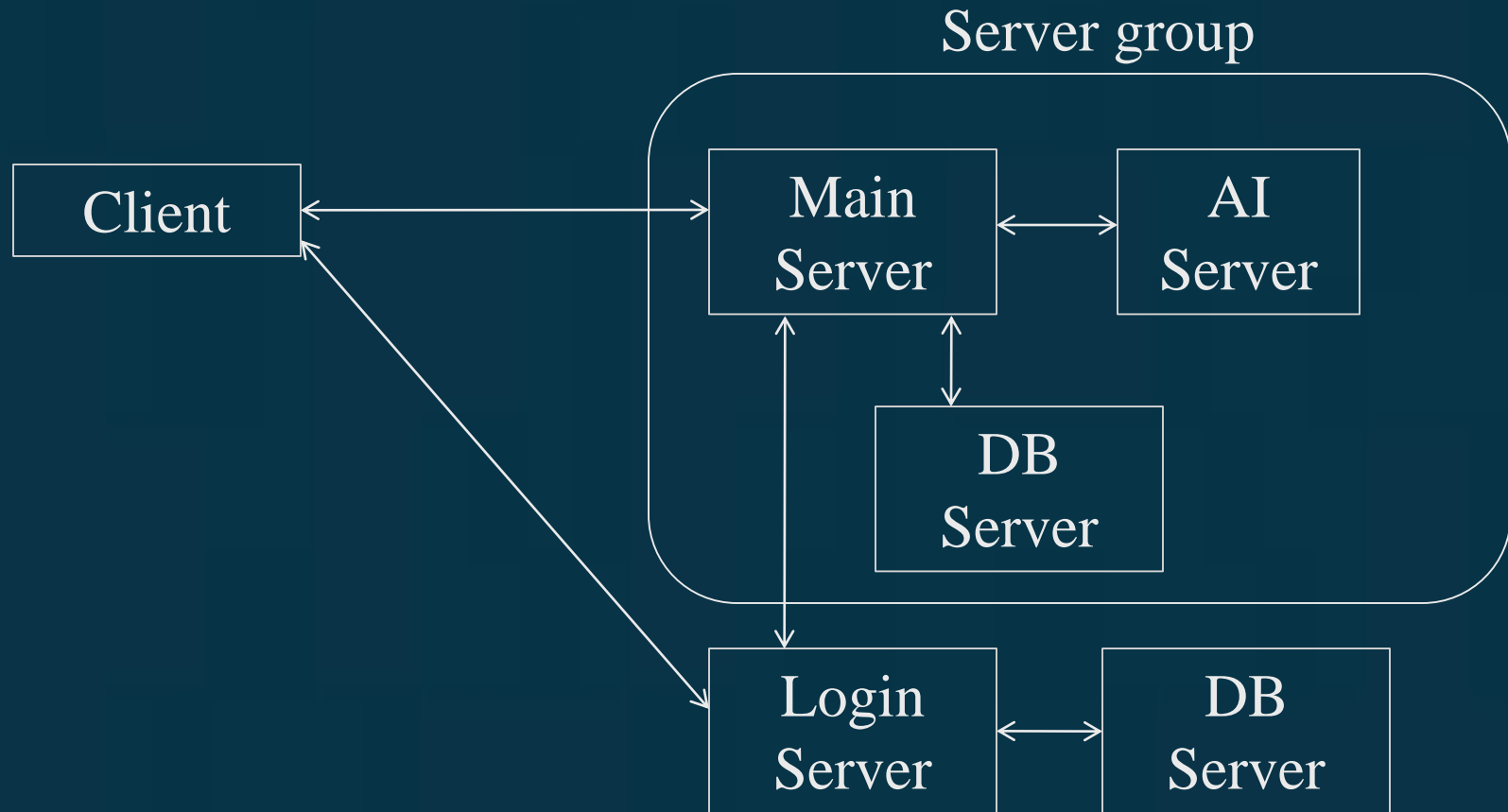
- 서버 분리 (샤딩, Sharding)
  - 월드 복사 & 분리
  - 다른 월드로 이동 불가
    - 불편 (친구 만나기 불가능...)
    - 돈내면 월드 이전 서비스...
    - DB가 분리되어 있음.
  - 통합하는 추세
    - 공동 경매장, 공동 인던 입장, 서버간 결투, 공동 Zone...

# 부하 분산

- 기능별 분산 구현
  - 사용자 분리 없이 확장
  - 부하가 크기 때문에 여러 종류의 서버의 조합으로 구성(기능별 분리)
    - Main Server
      - 실제 사용자가 접속해서 게임을 하는 서버
    - Login Server (Auth Server)
      - 사용자를 확인한 후 메인 서버에 접속시켜 주는 서버
    - AI Server (NPC Server)
      - NPC들의 AI를 담당하는 서버
    - DB Server, Query Server (Cache Server)
      - 게임 데이터를 관리하는 서버



# 부하 분산



# 부하 분산

- 기능별 분산 구현의 한계
  - Main Server
    - 부하가 집중된다.
    - 더 이상 기능적으로 나눌 수 없다. (통신 부담 증가)
      - 무시하고 계속 나누면... 넥슨의 마영전 온라인 사태가 벌어짐.
- Main Server의 부하 감소를 위한 분할
  - 공간 분할 : Zone 서버
    - 전체 맵을 지역별로 나누어서 다른 서버가 관리
    - Portal방식과 seamless의 두가지 방식이 있다.
    - 예) Portal : Wow의 대륙 분할, Seamless : BigWorld
  - 접속 분할 : Duplicate Server
    - 동일한 공간의 Clone을 만들어서 각각 다른 서버가 관리
    - 예) Mu(수동), Guild War(자동)
    - 샤딩과는 다르다!

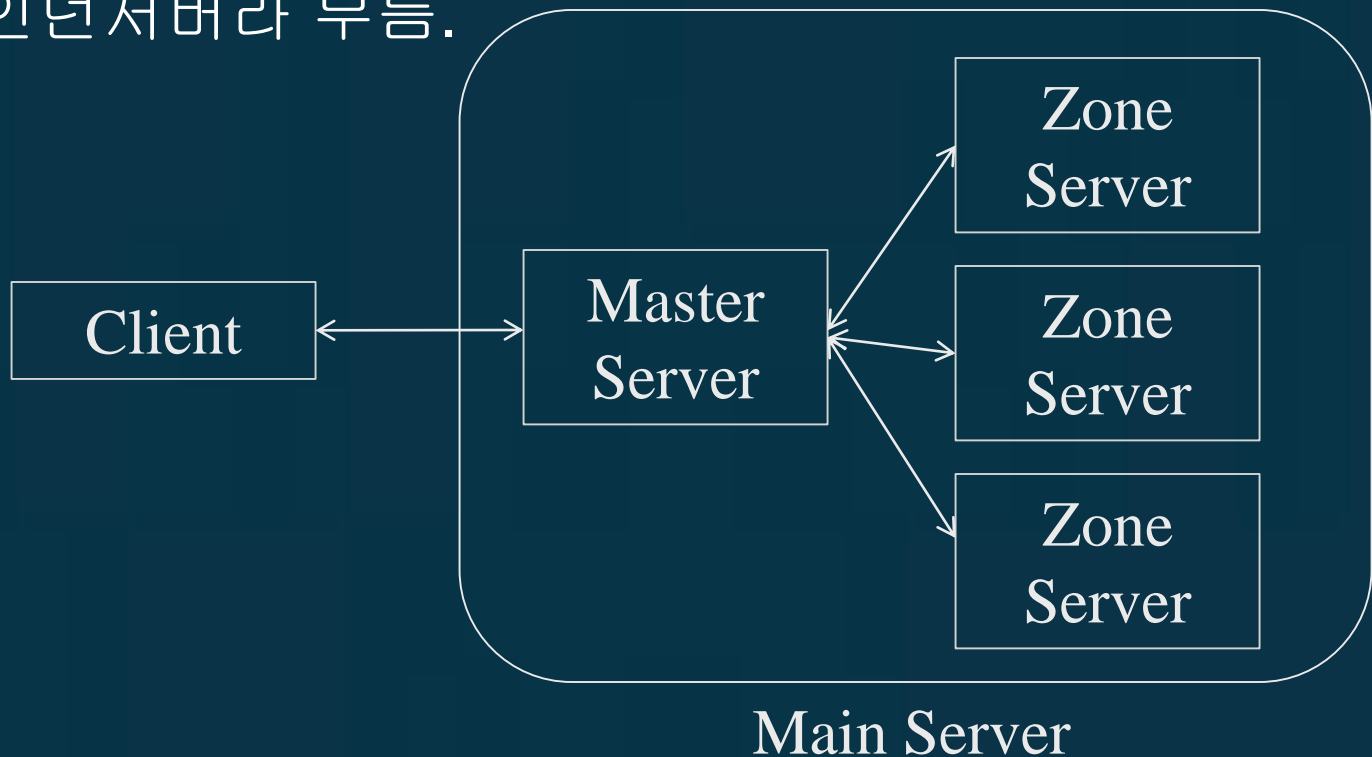
# 부하 분산

- 공간 분할 Zone 서버의 특징
  - Seamless
    - 구현하기 어렵다
    - 경계선 근처에 서있는 물체를 2개 이상의 서버가 동시에 관찰할 수 있어야 한다.
  - Potal
    - 구현하기 쉽다.
    - 게임성이 떨어진다.
      - Loading
      - 플레이어의 단절
    - 인스턴스 던전도 여기에 속한다.
- 정적 분할과 동적 분할

# 부하 분산

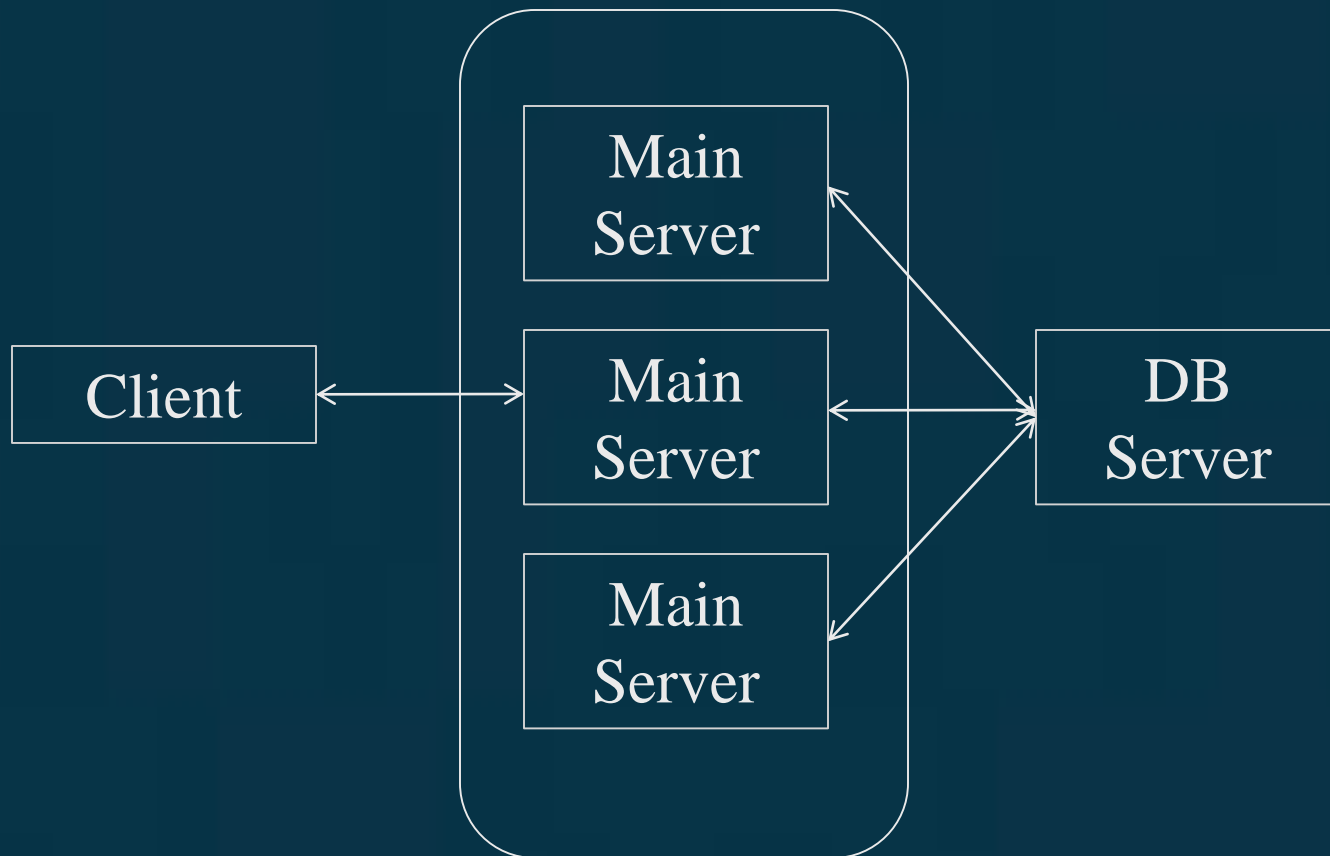
- Zone Server

- master 서버 (FrontEnd 서버): 연결 및 이동 담당
- zone 서버 : 게임 콘텐츠 담당, дина믹하게 생성될 경우 인스턴스 서버라 부름.



# 게임 서버의 설계

- Duplicate Server



# 내용

---

- 게임 서버 종류
- 게임서버를 위한 하드웨어와 소프트웨어

# 게임서버 하드웨어

- 하드웨어 지식이 필요한 이유
  - 서버에 걸리는 부하가 많기 때문에 하드웨어의 성능을 최대한 끌어내는 프로그래밍이 필요하다.
  - 하드웨어의 성능을 최대한 끌어내려면 하드웨어가 어떻게 성능에 영향을 미치는지 알아야 한다.

# 게임서버 하드웨어

---

- 중요 하드웨어
  - CPU
  - Memory
  - Network



# 게임서버 하드웨어

## ● CPU

- X86 계열이 대세 이다. (~~인텔의 Itanium~~, IBM의 PowerPC 계열도 존재) ARM은 사용불가.
- AMD와 Intel의 2가지 계열이 있다.
  - 서버용 CPU가 따로 존재한다.
  - 예) Intel Xeon, AMD EPYC
- 64bit와 Multi-core
- 속도와 Cache 크기, 메모리 Bus대역폭이 중요하다.

# 게임서버 하드웨어

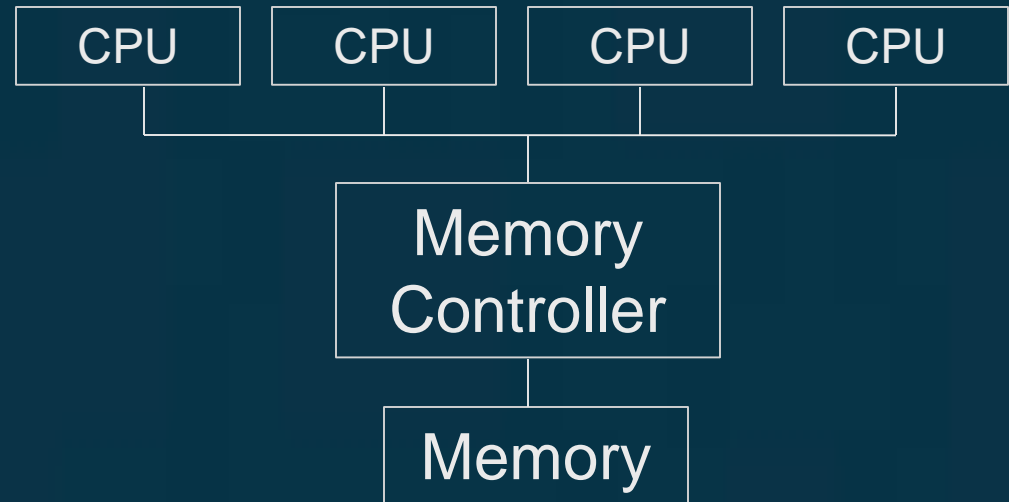
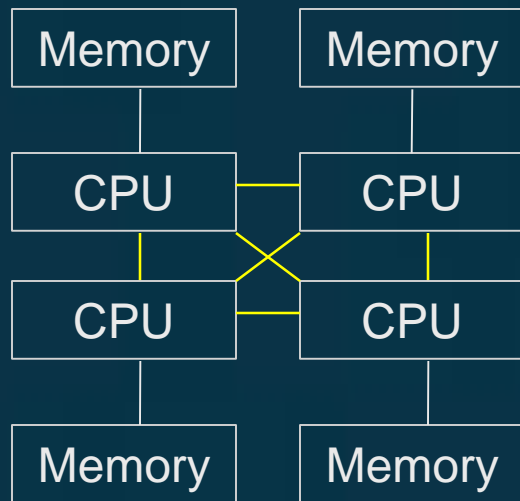
## ● CPU

- 서버용 CPU와 일반 Desktop용 CPU의 차이
  - 서버용 CPU는 Multi Processor를 지원한다.
    - Cache 동기화 지원
  - 서버용 CPU는 서로간의 데이터 전송 전용 통로를 제공한다.
    - AMD : HyperTransport (2001~) – 51GB/s, Infinity Fabric(2016~) – 30 ~ 512GB/s
    - 인텔 : QPI(Quick Path Interconnect) (2009~) – 26GB/s, Intel UltraPath Interconnect (2017~) – 6TB/s

# 게임서버 하드웨어

- CPU

NOW



OLD

CPU의 개수가 많아질 수록  
과거방식에서는 성능에 한계

# 게임서버 하드웨어(CPU)

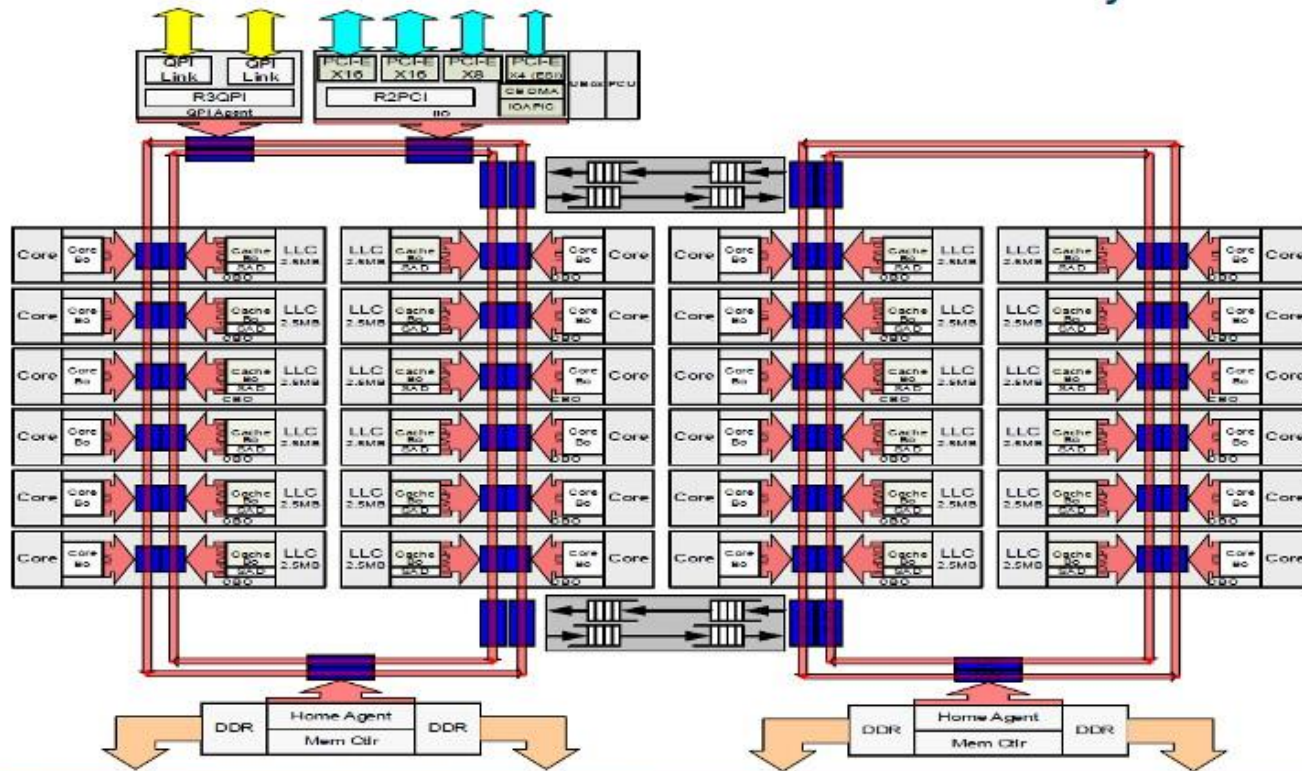
- 64비트 이슈
  - 64 비트를 사용한다.
    - OS와 Compiler도 64비트 버전을 사용해야 한다.
  - 기존의 32비트 CPU로는 최대 4GB의 메모리 밖에 쓸 수 없었기 때문에 서버 용량에 제한이 많았었다.
  - 64 비트의 경우 16ExaByte의 메모리가 가능하다.
  - 프로그래밍 시 Int type과 Pointer type의 크기가 달라지는 것을 주의 해야 한다.

# 게임서버 하드웨어(CPU)

- Multi-processor
  - SMP(Symmetric Multi Processing)
  - 빠른 네트워크 응답속도와 처리 속도 개선을 위해 Multi-processor을 사용
- Multi-core
  - 발열에 막힌 CPU의 성능향상 제한을 극복하기 위한 궁여지책
  - 기존의 4 ~ 8개의 CPU를 활용하던 프로그램 방식에서 8 ~ 64개의 Core를 활용하도록 변경필요. 앞으로도 계속 core 개수가 늘어날 예정
- Multi-Processor와 Multi-Core와의 차이점
  - SW적으로는 차이가 없음
  - HW적으로는 메모리 접근시 성능 차이가 존재
    - CPU의 개수 만큼 메모리 대역폭 증가
    - NUMA(Non Uniform Memory Access) 문제

# 게임서버 하드웨어(CPU)

## Intel® Xeon® Processor E5 v4 Product Family HCC



서버 CPU

# 게임서버 하드웨어(CPU)

## Intel® Xeon® Processor E5 v4 Product Family HCC



Intel

INTEL XEON 22 CORE PROCESSOR E5-2699V4  
2.2GHZ 55MB SMART CACHE 9.6 GT/S QPI TDP 145W

★★★★☆ 21 customer reviews | 19 answered questions

Price: \$1,750.00 & FREE Shipping

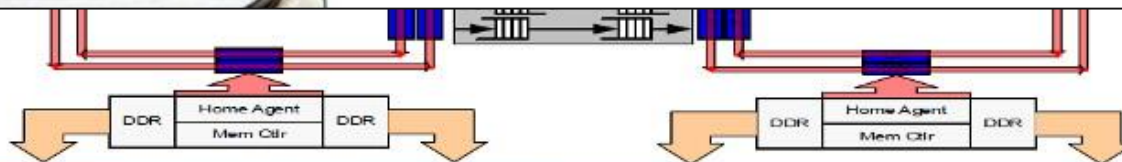
In Stock.

Get it as soon as Friday, March 10 when you choose Two-Day Shipping at checkout.  
Ships from and sold by IB Tech.

- XEON PROCESSOR E5-2699V4

[Compare with similar items](#)

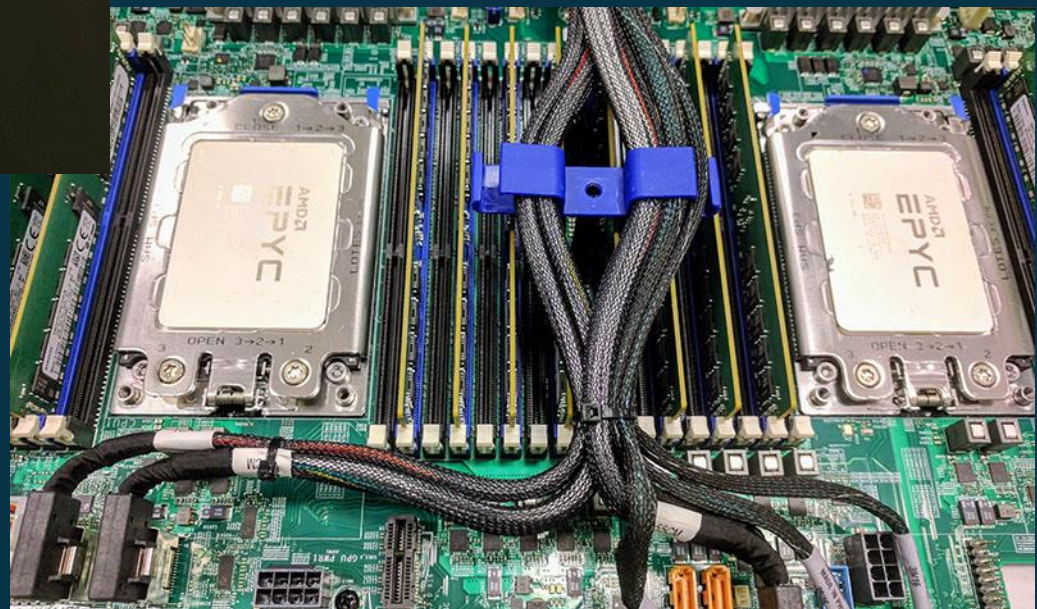
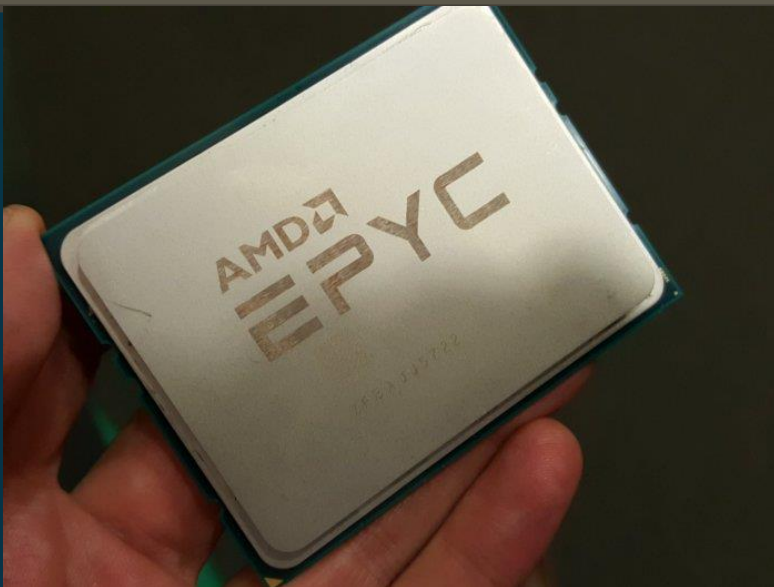
[Used & new \(23\) from \\$1,750.00 & FREE shipping.](#)



서버 CPU



# 게임서버 하드웨어(CPU)





# 게임서버 하드웨어(CPU)

- CPU 발전의 Trend

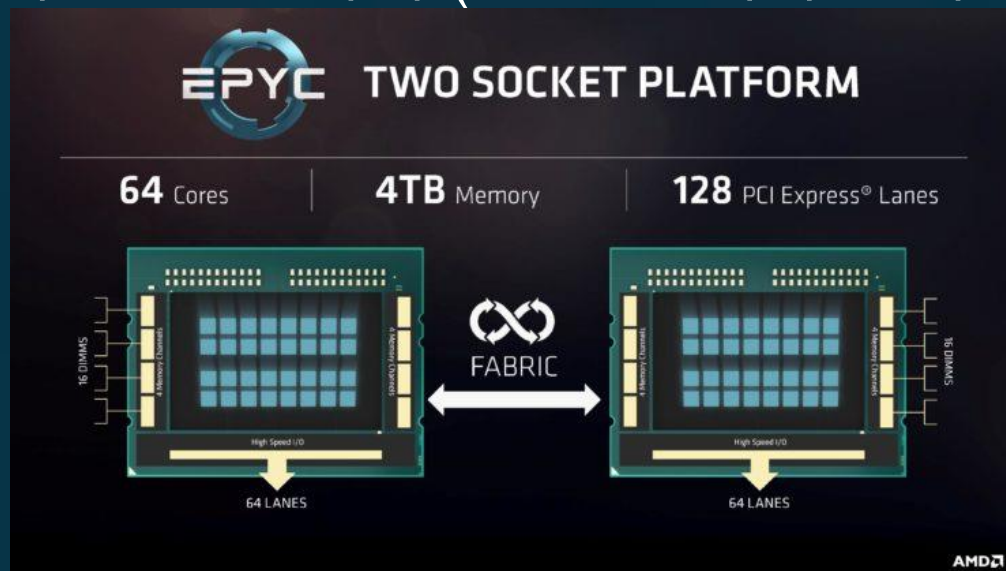
- 클럭속도 증가
  - 반도체 생산 공정 개선
  - 한계에 부딪침 : 발열
- Clock당 수행되는 명령어의 개수 증가
  - 아키텍처의 개선
  - Pipeline, SuperScalar, SuperPipeline, Out-of-order
  - 캐시용량 증가.
  - 한계에 부딪침 : 한계효용의 법칙

# 게임서버 하드웨어(CPU)

- CPU 발전의 Trend

- Core 개수 증가

- 현재의 방식, 프로그램 작성 방식이 바뀌어야 함
    - 발열로 인한 클럭속도 향상의 한계가 주 원인
    - 현재 32-Core까지 (앞으로 계속 늘어날 예정)



# 게임서버 하드웨어(CPU)

- 파이프라인의 발전

- “어떤 명령어를 수행하는데 몇 cycle이 걸리는가?”라는 질문은 의미가 없음
- 파이프라인이 무효화 되지 않는 한 프로그램의 실행 속도는 메모리 Read에 종속됨
  - 명령어가 더 많아도 메모리 Read가 적으면 더 빠름

- SIMD 명령어의 발전

- 하나의 명령어로 여러 개의 실수를 동시에 연산
- MMX(MultiMedia eXtension) -> SSE(Streaming SIMD Extension) -> AVX (Advanced Vector Extension)

# 게임서버 하드웨어(CPU)

- Pipeline의 고도화에 따른 주의
  - 파이프 라인을 리셋 시키면 손해가 너무 크다
    - Pentium 4의 경우 31단계
  - 리셋의 원인
    - 시스템 콜
    - 분기 예측 오류
    - Interrupt, Exception
  - 대책
    - 시스템 콜을 될 수 있으면 하지 말 것
    - If, switch등을 자제한다

# 게임서버 하드웨어(CPU)

## ● CACHE

- 프로그램 실행속도에 가장 큰 영향을 미치는 요소
- Cache가 큰 CPU일 수록 속도가 빠르다

## ● Tip

- 같이 쓰이게 되는 데이터는 묶어 놓는다.
- 루프 안에서 사용하는 데이터는 캐쉬에 다 올라올 수 있도록 한다.
- Int대신에 half나 char을 사용한다.

**Cache Behavior Affects Performance**

Cost of Data Access increases with Distance from CPU

Programming Tips:

- Maximize work done on cached data
- Work with Hardware Prefetch (arrays vs linked lists)

Cost of accessing data

Where Data Is Resident	Time to fetch data
Register	1 cycle
L1 Cache	4 cycles
L2 Cache	10 cycles
L3 Cache	40-75 cycles
Memory	60-100 ns

[http://software.intel.com/sites/products/collateral/hpc/vtune/performance\\_analysis\\_guide.pdf](http://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf)

# 게임서버 하드웨어(CPU)

## ● Multi Processor Programming

- 잘하면 N배 성능향상, 못하면 성능하락
- Lock을 줄여라
  - Lock을 실행하는 순간 모든 CPU의 cache사용이 중지됨
  - Semaphore, Condition 변수는 시스템 Call
- Cache Thrashing에 주의 하라
  - Cache는 line단위로 움직임

# 게임서버 하드웨어(CPU)

- 최근 CPU 구조를 알고 싶으면
  - <http://udteam.tistory.com/57>

# 게임서버 하드웨어

## ● Memory

- 서버가 요구하는 용량을 제공하면 된다.
- 일반적인 Desktop용 메모리가 아니라 Error수정기능이 있는 특수 메모리를 사용한다.
- 대역폭이 크지만 반응 속도는 느리다.



삼성에서 새롭게 출시한 PC2 5300 Fully Buffered DIMM 1GB 는 서버 및 워크스테이션에 장착하여 다중 채널로 동작하는 FBDIMM 240핀의 제품으로 667MHz의 높은 동작 속도로 고성능은 물론 안정성에서도 뛰어납니다. 최대 3.2Gb/s 의 링크 대역폭을 제공하고, 채널의 에러를 관리하는 ECC 기능으로 데이터의 무결성을 보장해 줍니다.



# 게임서버 하드웨어

## ● Network

- 10M -> 100M -> 1G -> 10G 까지 발전
- 서버간의 연결을 위해 한 서버에 여러 개의 네트워크 카드를 꽂아서 사용하기도 한다.
  - 확장 버스의 대역폭을 고려해야 한다.
  - PCI (32bit, 33MHz) : 133MB/s
  - PCI-Express 2.0 (1 lane) : 500MB/s
- 속도가 더 필요하면 Infiniband사용

Effective unidirectional theoretical throughput (actual data rate, not signaling rate)						
	SDR	DDR	QDR	FDR-10	FDR	EDR
1X	2 Gbit/s	4 Gbit/s	8 Gbit/s	10 Gbit/s	13.64 Gbit/s	25 Gbit/s
4X	8 Gbit/s	16 Gbit/s	32 Gbit/s	40 Gbit/s	54.54 Gbit/s	100 Gbit/s
12X	24 Gbit/s	48 Gbit/s	96 Gbit/s	120 Gbit/s	163.64 Gbit/s	300 Gbit/s

# 게임서버 운영체제

- 서버용 운영체제의 종류
  - Unix계열
    - 리눅스, FreeBSD, Solaris, OSx
    - 가격이 저렴하다.
    - 유지 보수 관리가 어렵다. (잘 아는 사람이 필요하다)
  - Windows계열
    - Windows 2008, Windows 2012, Windows 2016
    - 비싸다.
    - 유지 보수 관리가 비교적 쉽다. (배우기 쉽다. 신경쓸 것이 적다.)

# 실제 예제

- 하드웨어가 프로그래밍 성능에 미치는 영향  
실습
  - 시스템 Call
  - Cache
  - Pipelining

# 실제 예제

## ● 시스템 Call

```
#include <iostream>
#include <chrono>
#include <Windows.h>

using namespace std;
using namespace chrono;

int main()
{
    volatile int tmp = 0;
    auto start = high_resolution_clock::now();
    for (int j=0; j<10000000; ++j) {
        tmp += j;
        Sleep(0);
    }
    auto duration = high_resolution_clock::now() - start;
    cout << "Time " << duration_cast<milliseconds>(duration).count();
    cout << " msec\n";
    cout << "RESULT " << tmp << endl;
}
```

# 실제 예제

## ● Cache Miss

```
for (int i=0; i<21; ++i) {  
    int size = 1024 << i;  
    char *a = (char *) malloc(size);  
    unsigned int index = 0;  
    volatile unsigned int tmp=0;  
    auto start = high_resolution_clock::now();  
    for (int j=0; j<1000000000; ++j) {  
        tmp += a[index % size];  
        index += CACHE_LINE_SIZE * 11;  
    }  
    auto dur = high_resolution_clock::now() - start;  
}
```

Cache\_Line\_Size??? \* 11 ????

# 실제 예제

## ● Cache Miss

CPU-Z

CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About

Processor

Name: Intel Core i7 6700

Code Name: Skylake Max TDP: 65.0 W

Package: Socket 1151 LGA

Technology: 14 nm Core Voltage: 0.996 V

Specification: Intel® Core™ i7-6700 CPU @ 3.40GHz

Family: 6 Model: E Stepping: 3

Ext. Family: 6 Ext. Model: 5E Revision: R0

Instructions: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3, TSX

Clocks (Core #0)

Core Speed: 3798.14 MHz

Multiplier: x 38.0 (8 - 40)

Bus Speed: 99.95 MHz

Rated FSB:

Cache

Cache	Size	Associativity
L1 Data	4 x 32 KBytes	8-way
L1 Inst.	4 x 32 KBytes	8-way
Level 2	4 x 256 KBytes	4-way
Level 3	8 MBytes	16-way

Selection: Processor #1 Cores: 4 Threads: 8

CPU-Z Ver. 1.78.3.x64 Tools Validate Close

CPU-Z

CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About

L1 D-Cache

Size: 32 KBytes x 4

Descriptor: 8-way set associative, 64-byte line size

L1 I-Cache

Size: 32 KBytes x 4

Descriptor: 8-way set associative, 64-byte line size

L2 Cache

Size: 256 KBytes x 4

Descriptor: 4-way set associative, 64-byte line size

L3 Cache

Size: 8 MBytes

Descriptor: 16-way set associative, 64-byte line size

Size: Descriptor: Speed:

CPU-Z Ver. 1.78.3.x64 Tools Validate Close

# 실제 예제

- Cache Miss

```
C:\WINDOWS\system32\cmd.exe
SIZE : 1K, Time : 263msecs
SIZE : 2K, Time : 253msecs
SIZE : 4K, Time : 253msecs
SIZE : 8K, Time : 259msecs
SIZE : 16K, Time : 257msecs
SIZE : 32K, Time : 258msecs
SIZE : 64K, Time : 273msecs
SIZE : 128K, Time : 265msecs
SIZE : 256K, Time : 278msecs
SIZE : 512K, Time : 269msecs
SIZE : 1024K, Time : 313msecs
SIZE : 2048K, Time : 343msecs
SIZE : 4096K, Time : 476msecs
SIZE : 8192K, Time : 1026msecs
SIZE : 16384K, Time : 1137msecs
SIZE : 32768K, Time : 1021msecs
SIZE : 65536K, Time : 1222msecs
SIZE : 131072K, Time : 1208msecs
SIZE : 262144K, Time : 1153msecs
SIZE : 524288K, Time : 1196msecs
SIZE : 1048576K, Time : 1316msecs
계속하려면 아무 키나 누르십시오...
```

# 실제 예제

- Pipeline stall

- branch miss

- 조건부 분기와 분기 예측 실패

```
#define abs(x) (((x)>0)?(x):- (x))
```

VS

```
static long abs(long x) {  
    long y;  
    y = x>>31;    /* Not portable */  
    return (x^y)-y;  
}
```



# 실제 예제

## ● Pipeline stall

```
int main()
{
    int sum;

    for (int i = 0; i < T_SIZE; ++i) rand_arr[i] = rand() - 16384;

    sum = 0;
    auto start_t = high_resolution_clock::now();

    for (int i = 0; i < T_SIZE; ++i) sum += abs(rand_arr[i]);

    auto du = high_resolution_clock::now() - start_t;
    cout << "[abs] Time " << duration_cast<milliseconds>(du).count() << " ms\n" ;
    cout << "Result : " << sum << endl;

    sum = 0;
    start_t = high_resolution_clock::now();

    for (int i = 0; i < T_SIZE; ++i) sum += abs2(rand_arr[i]);

    du = high_resolution_clock::now() - start_t;
    cout << "[abs2] Time " << duration_cast<milliseconds>(du).count() << " ms\n";
    cout << "Result : " << sum << endl;
}
```

VS

# 실제 예제

- Pipeline stall
  - branch miss
    - 조건부 분기와 분기 예측 실패
    - 만일 루프에서 같은 값만을 사용한다면?

# 정리

- 게임서버의 최적화는 HW를 고려해야 한다.
  - 컴퓨터 구조에서 배운 것을 실제로 고려해야 한다.
  - System Call 하지 않기
  - Cache 잘 사용하기
    - 가능하면 메모리 적게 사용하기
  - Multi-Thread Programming (다음달에..)