- 동작 구조 설계 -

Boot Loader

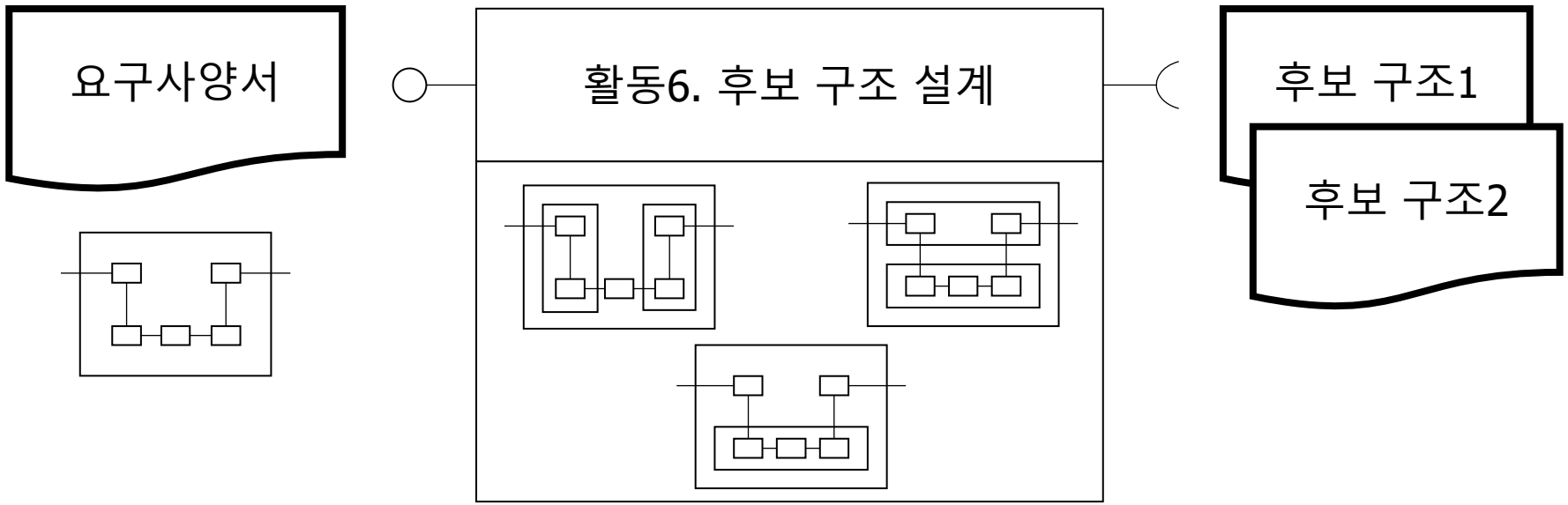
조 용 진

(drajin.cho@bosornd.com)

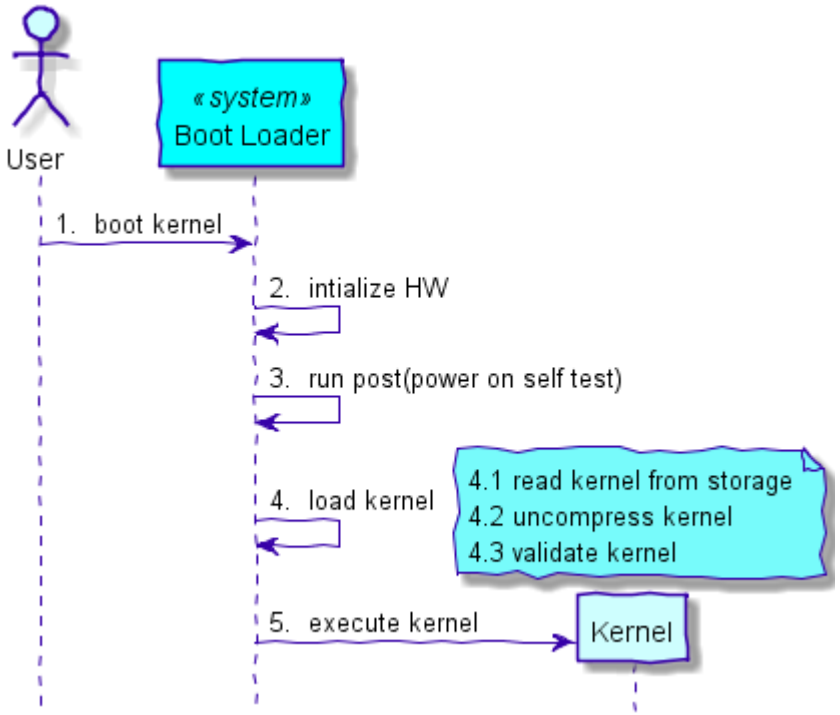
주식회사 보이는 소프트웨어 연구소
http://www.bosornd.com

활동6. 후보 구조 설계

목적	도메인 모델(개념적 구조)을 실현하기 위한 시스템 구조를 설계한다. 품질 요구사항을 개선하는 후보 구조를 설계한다.

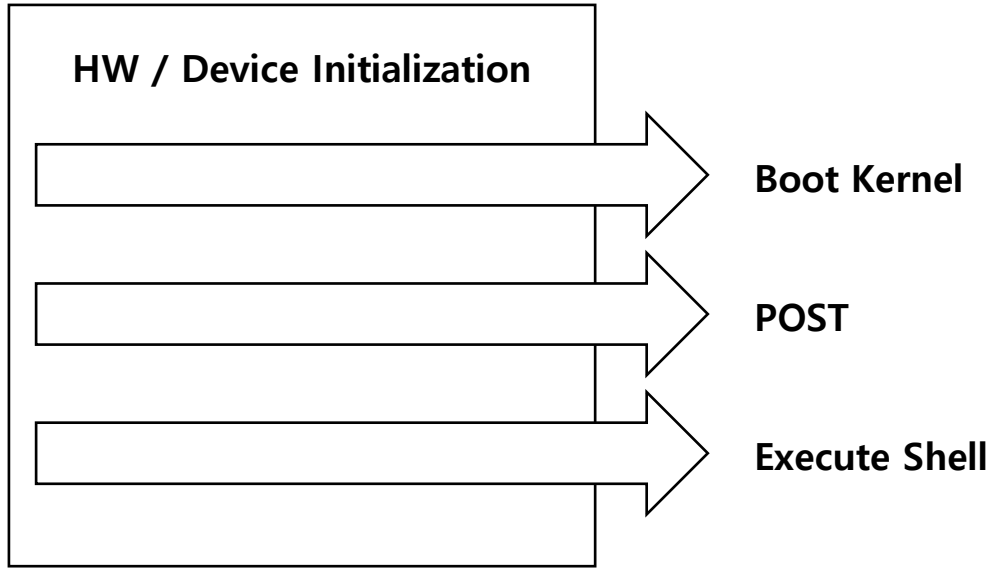


활동6. 후보 구조 설계 (실행)

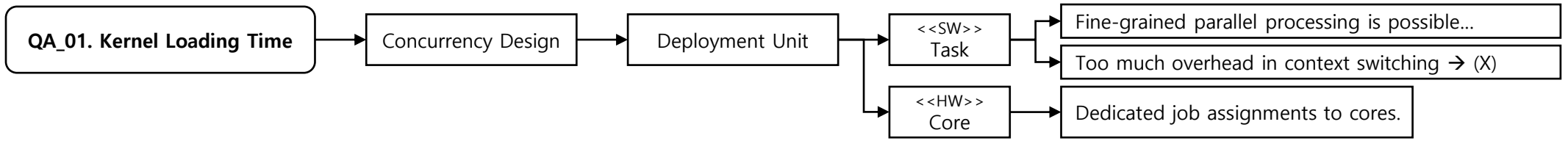


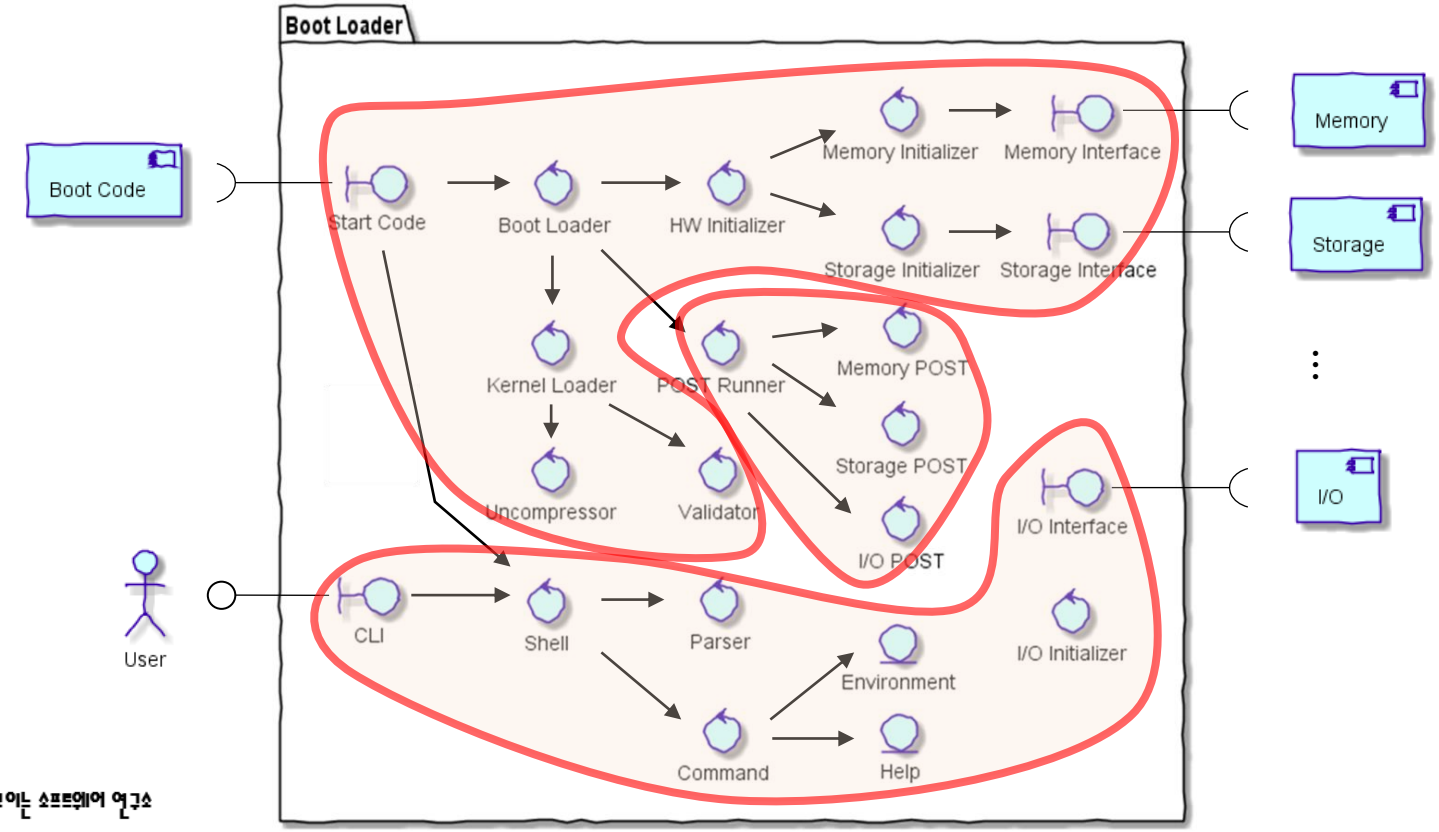
Architectural Concern

- Concurrency -

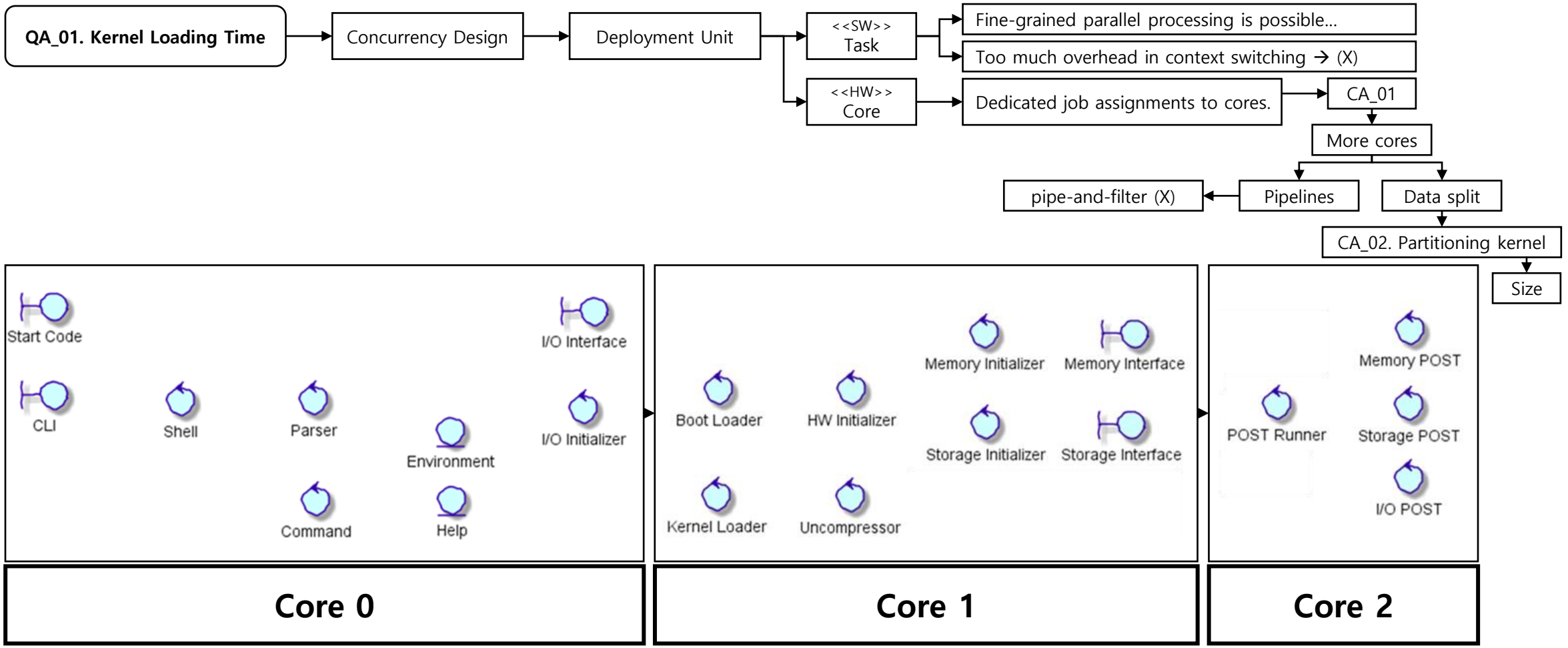


활동6. 후보 구조 설계 (실행)





활동6. 후보 구조 설계 (실행)

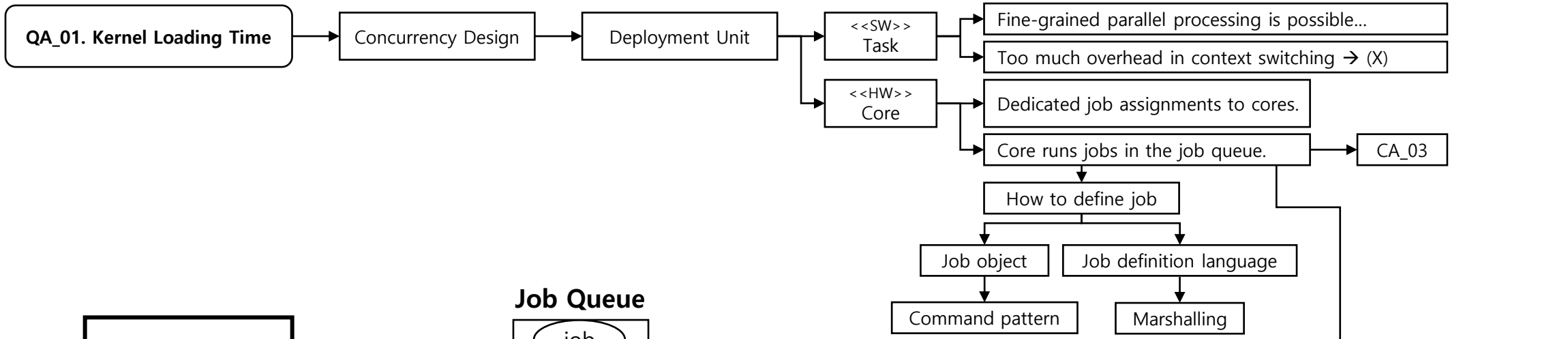


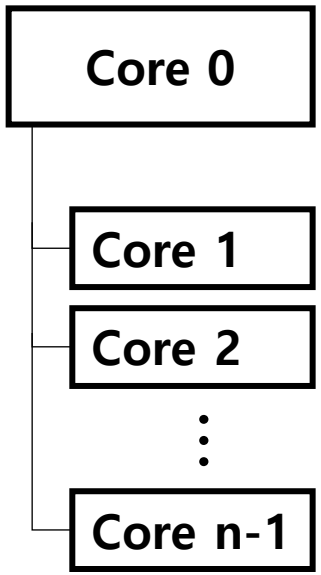
CA_01. Kernel Load, Shell, POST dedicated core

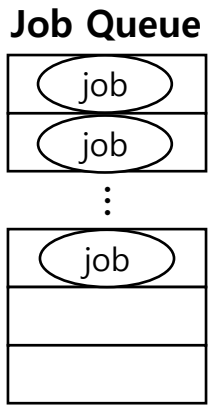
CA_02. Partitioning kernel data

활동6. 후보 구조 설계 (실행)

https://en.wikipedia.org/wiki/Batch_processing
https://en.wikipedia.org/wiki/Job_queue







CA_03. Redundant worker cores

CA_04. Job scheduling by queue ⟶ Alternatives?

CA_05. Job scheduler (server)

활동6. 후보 구조 설계 (실행)

```cpp
#include <windows.h>

#include <fstream>
#include <iostream>

const char* filename = "image.dat";

Int main() {
    std::ofstream outFile(filename, std::ios::binary | std::ios::trunc);

    for (int r = 0, count = 0; r < 400000; ++r) {
        int data[256];
        for (int i = 0; i < 256; ++i, ++count) data[i] = count;

        outFile.write((char*)data, 256 * sizeof(int));
    }

    outFile.flush();
    outFile.close();

    return 0;
}
```

```cpp
#include <windows.h>

#include <fstream>
#include <iostream>

const char* filename = "image.dat";

Int main() {
    ULONGLONG startTime = GetTickCount64();

    std::ifstream inFile(filename, std::ios::binary);

    int data[256];
    for (int r = 0, count = 0; r < 400000; ++r) {
        inFile.read((char*)data, 256 * sizeof(int));

        for (int i = 0; i < 256; ++i, ++count)
            if (data[i] != count) std::cerr << "error" << std::endl;
    }

    inFile.close();

    printf("elapsed time = %lldWn", GetTickCount64() - startTime);

    return 0;
}
```

활동6. 후보 구조 설계 (실행)

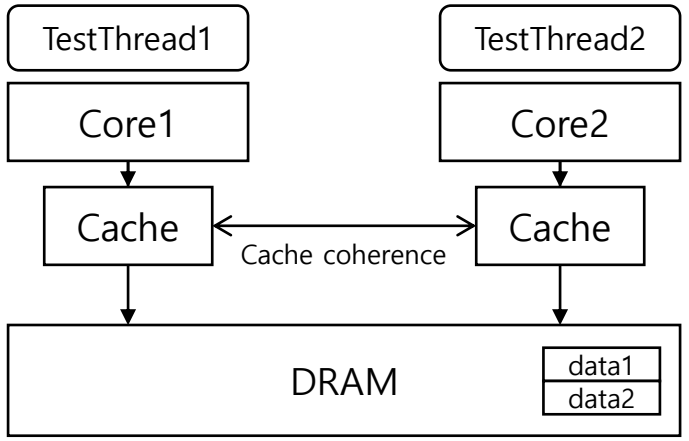
```cpp
#include <windows.h>

#include <fstream>
#include <iostream>

const char* filename = "image.dat";

struct THREAD_DATA {
    int start, end;
};

DWORD CALLBACK thread_func(void* param) {
    THREAD_DATA* td = (THREAD_DATA*)param;

    std::ifstream inFile(filename, std::ios::binary);
    inFile.seekg(td->start * 256 * sizeof(int));

    int data[256];
    const int length = td->end - td->start;
    for (int r = 0, count = td->start * 256; r < length; ++r) {
        inFile.read((char*)data, 256 * sizeof(int));

        for (int i = 0; i < 256; ++i, ++count)
        if (data[i] != count) std::cerr << "error" << std::endl;
    }

    inFile.close();

    return 0;
}
```

```cpp
int main() {
    ULONGLONG startTime = GetTickCount64();

    const int num_threads = 4;
    HANDLE thread[num_threads];

    int start = 0, delta = 400000 / num_threads;
    for (int i = 0; i < num_threads; ++i) {
        THREAD_DATA* td = new THREAD_DATA;
        td->start = start; start += delta;
        td->end = start;

        thread[i] = CreateThread(NULL, 0, thread_func, td, 0, 0);
    }

    WaitForMultipleObjects(num_threads, thread, TRUE, INFINITE);

    printf("elapsed time = %lld₩n", GetTickCount64() - startTime);

    return 0;
}
```

활동6. 후보 구조 설계 (실행)

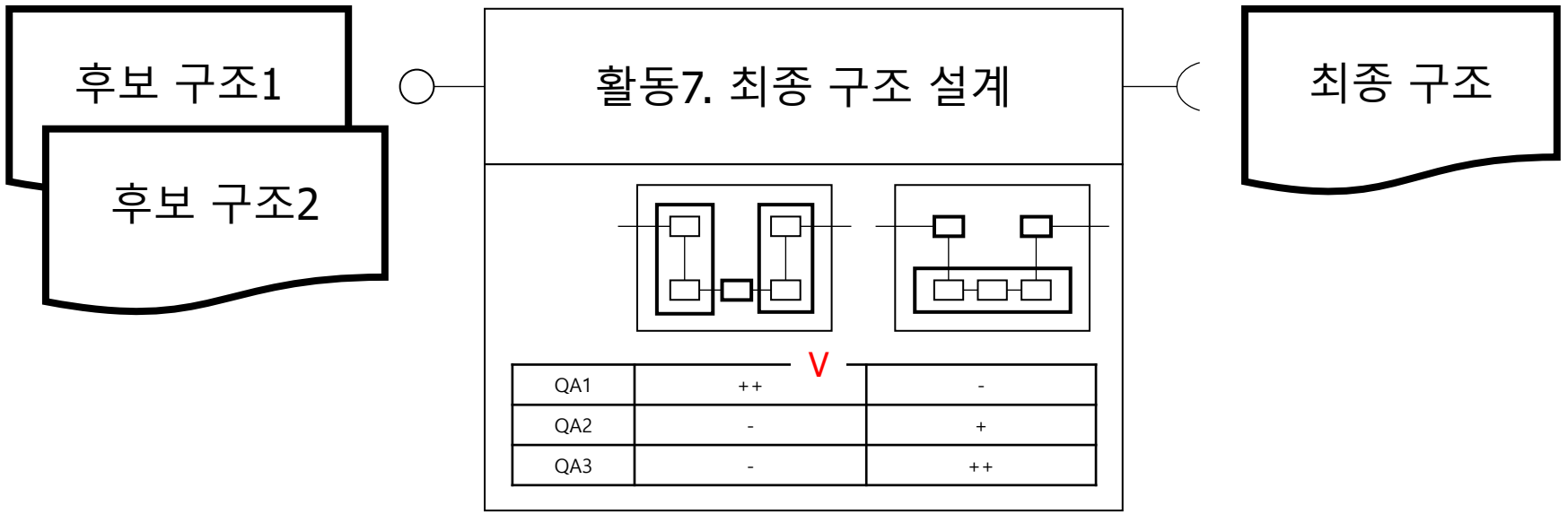
```
#include <stdio.h>
#include <windows.h>

int data1;
int data2;

const int LOOP_COUNT = 100000000;
#define COUNT(data)for (int i = 0; i < LOOP_COUNT; ++i) data = data + 1;

DWORD CALLBACK TestThread1(void*) {
    SetThreadAffinityMask(GetCurrentThread(), 1 << 1);// running on Core 1
    COUNT(data1);
    return data1;
}

DWORD CALLBACK TestThread2(void*) {
    SetThreadAffinityMask(GetCurrentThread(), 1 << 2);// running on Core 2
    COUNT(data2);
    return data2;
}

int main() {
    HANDLE thread[2];
    SetPriorityClass(GetCurrentProcess(), HIGH_PRIORITY_CLASS);

    startTime = GetTickCount64();

    thread[0] = CreateThread(NULL, 0, TestThread1, 0, 0, 0);
    thread[1] = CreateThread(NULL, 0, TestThread2, 0, 0, 0);

    WaitForMultipleObjects(2, thread, TRUE, INFINITE);

    printf("elapsed time with threads = %lldWn", GetTickCount64() - startTime);

    return 0;
}
```

```
#define CACHE_LINE      64      // 64 bytes
#define CACHE_ALIGN     __declspec(align(CACHE_LINE))

CACHE_ALIGN int data1;
CACHE_ALIGN int data2;
```



활동7. 최종 구조 설계

목적	과제의 요구사항에 가장 적합한 구조를 설계한다. 기능적/비기능적 요구사항을 만족하는 후보 구조 중에서, 품질 속성을 가장 잘 만족하는 후보 구조를 선정한다. 최종 구조의 단점을 보완한다.



활동7. 최종 구조 설계 (실행)

	CA_01. Kernel Load, Shell, POST dedicated core	CA_03. Redundant worker cores
		SELECTED
QA_01. Kernel Loading Time	(++) The optimized deployment for product HW is possible. But manual optimization is required.	(+) Optimization by concurrent execution. But there is a job scheduling overhead.
QA_02. Multi-core Changes	(-) Deployment should be re-designed.	(++) No adaptation is required.
QA_04. Storage Changes QA_05. HW Modifiability QA_07. Shell Modifiability QA_08. POST Modifiability	(-) Manual optimization is required.	(+) Automatically adjusted at runtime.

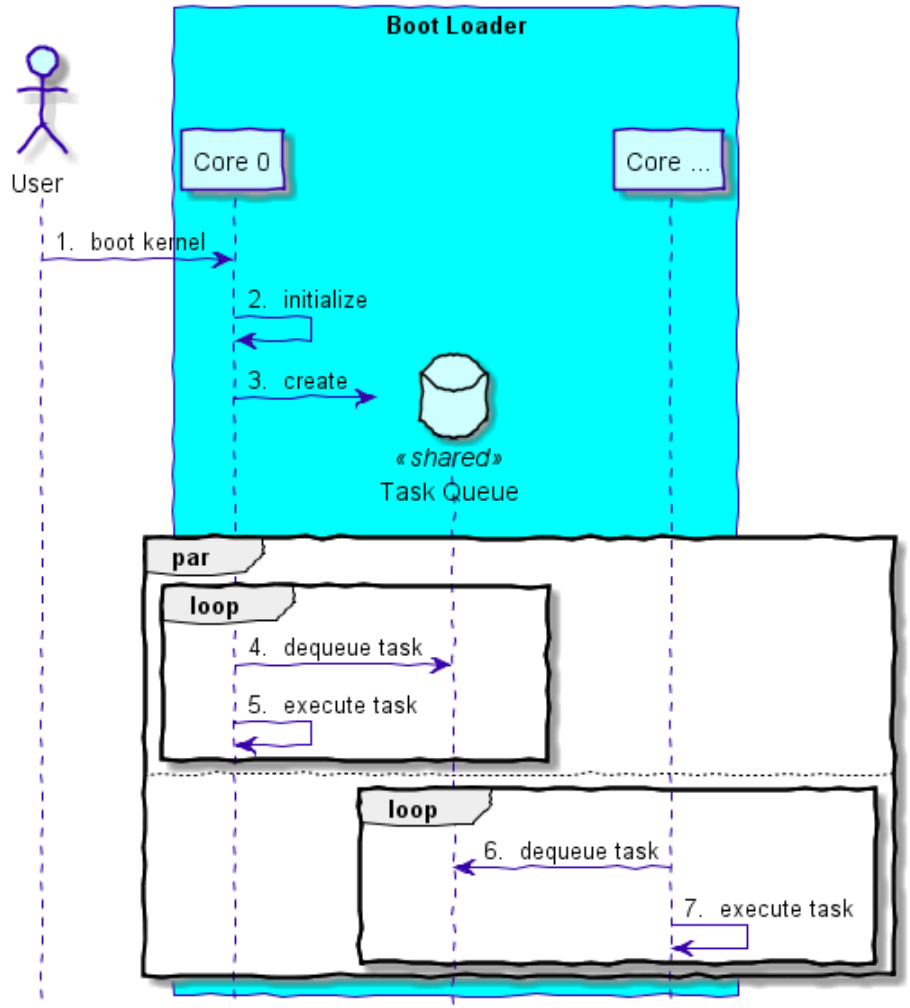
활동7. 최종 구조 설계 (실행)

Core 0 performs initialization and creates Task Queue.

Then, core 0 deques and executes a task sequentially.

Other Cores also dequeue and execute tasks concurrently.

Task Queue is shared by Cores.



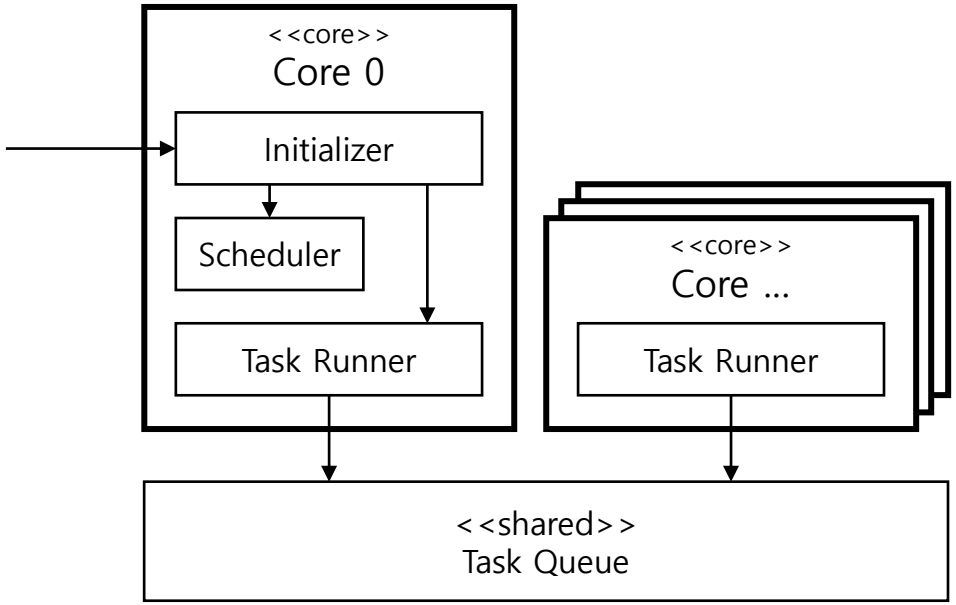
활동7. 최종 구조 설계 (실행)

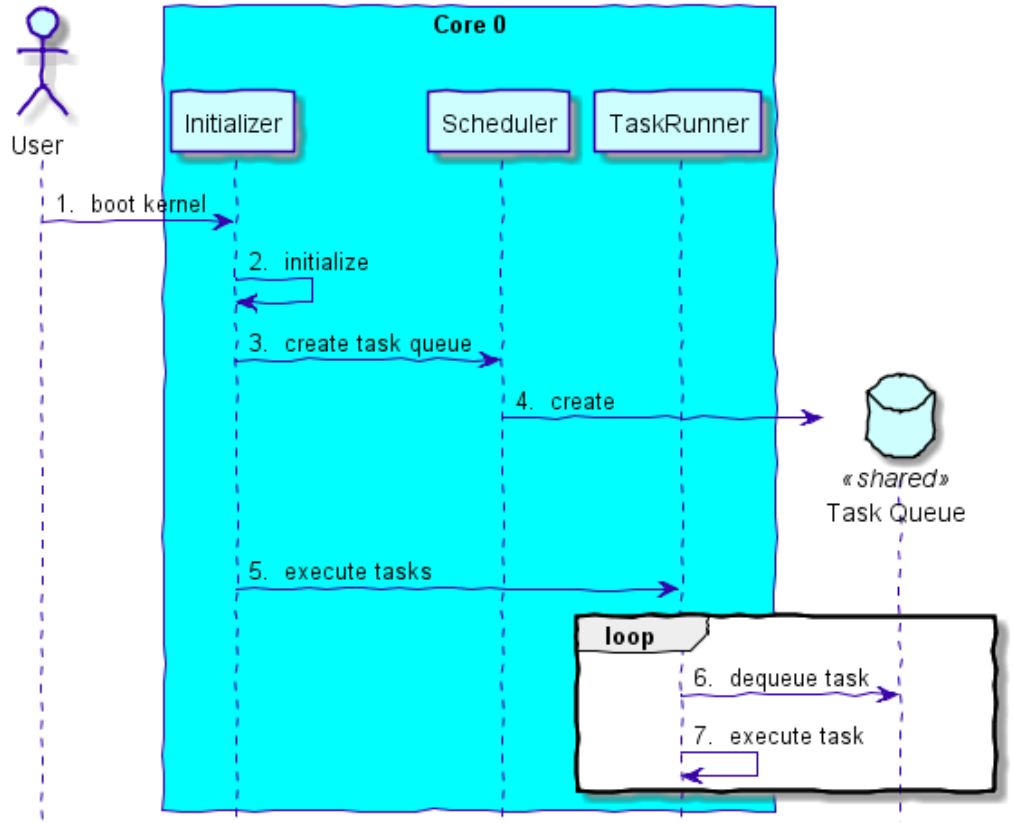
Core 0 performs initialization and creates Task Queue.

Then, core 0 deques and executes a task sequentially.

Other Cores also dequeue and execute tasks concurrently.

Task Queue is shared by Cores.





활동7. 최종 구조 설계 (실행)

Core 0 runs Initializer. Initializer initializes CPU and memory.

Initializer requests Scheduler to create Task Queue.

Initializer requests Task Runner to execute tasks sequentially.

Other cores run Task Runner to execute tasks concurrently.