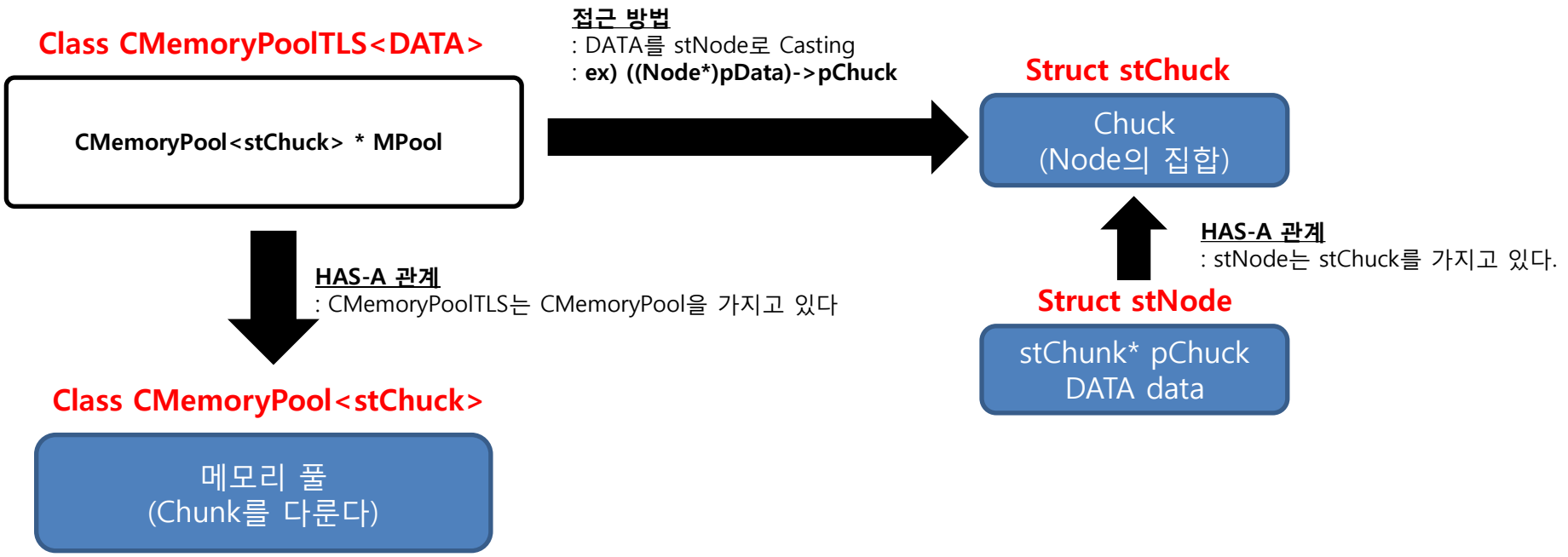
목표

1. New, delete보다 빠른 동적 할당, 동적 해제
2. Thread Safe 구조
3. 기존 메모리 풀보다 속도 향상

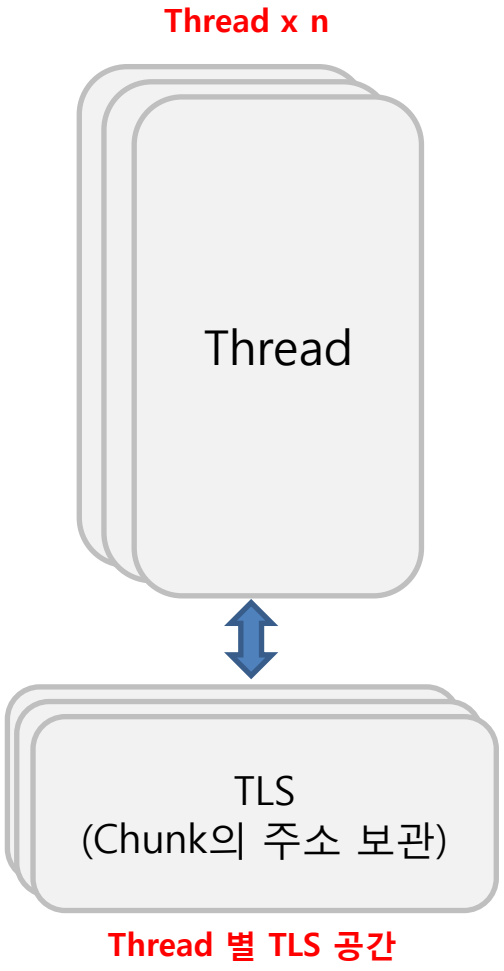
구현 스펙

1. Thread 1개(자기 자신)만 접근할 수 있는 공간 필요 **(TLS의 등장)**
2. TLS에 일정 수의 Node 보관 **(Chuck의 등장. ※ Chuck : Node의 집합)**
3. 각 Thread는 TLS의 Chuck에서 Node를 Alloc한다. Node가 모두 사용되면 다시 Chuck를 가져와, TLS에 보관한다. **(메모리 풀 등장. Chuck를 관리)**
4. TLS의 Chuck에 소속된 모든 Node가 사용됐다면, Chuck를 반환한다. **(모든 Node는 1회용)**

메모리 풀 TLS의 객체 관계도



메모리 풀 TLS 기본 구조



Class CMemoryPoolTLS<DATA>

Class CMemoryPool<stChuck>

메모리 풀
(Chunk를 다룬다)

Struct stChuck

Chuck
(Node의 집합)

Struct stNode

실제 데이터
보관 공간

DATA* Alloc()

TLS에 보관중인 Chunk에서 **Node의 data 주소 리턴**

Free(DATA* pData)

pData가 소속된 Chuck의 **모든 Node가 반환**되었다면,
메모리 풀에 Chuck 반환

구현 후 테스트

1. new, delete, 메모리 풀 Alloc, 메모리 풀 Free를 1억회 테스트 진행

결과

1. New보다 약 5배 빠른 Alloc 구현
2. Delete보다 약 2.5배 빠른 Free 구현

※ 첨부된 파일 Profiling_1 ~ Profiling_3 참조