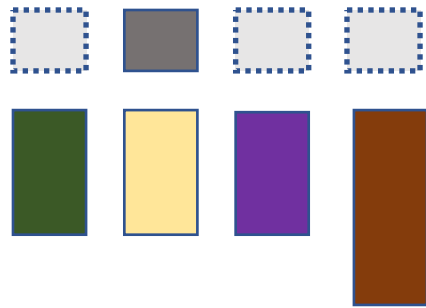


Process VS Thread

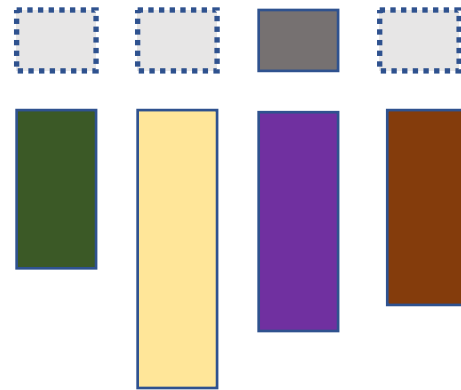
용어 정의

- Program: 작업을 위해 실행될 수 있는 파일
- Process: 실행 중인 프로그램으로 최소한 하나의 Thread를 갖는다.
- Thread: 프로세스의 실행 단위
- Concurrency (동시성, 병행성): Multi Tasking을 위해 Single Core에서 여러 개의 Thread가 돌아가며 실행되는 성질이다. (소프트웨어적인 용어)
 - 동시에 시간을 나누어 쓴다 (Time sharing 한다.)
- Parallelism(병렬성): Multi Core로 Multi Tasking하는 방식으로 동시에 작업이 처리된다.(하드웨어적인 용어)
- Context Switching : CPU 자원을 프로세스 -> 프로세스 / 스레드 -> 프로세스 / 스레드 -> 스레드로 전환하는 것

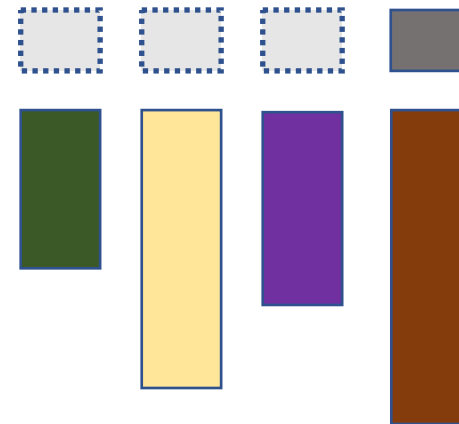
Concurrency(동시성)



작업1 작업2 작업3 작업4



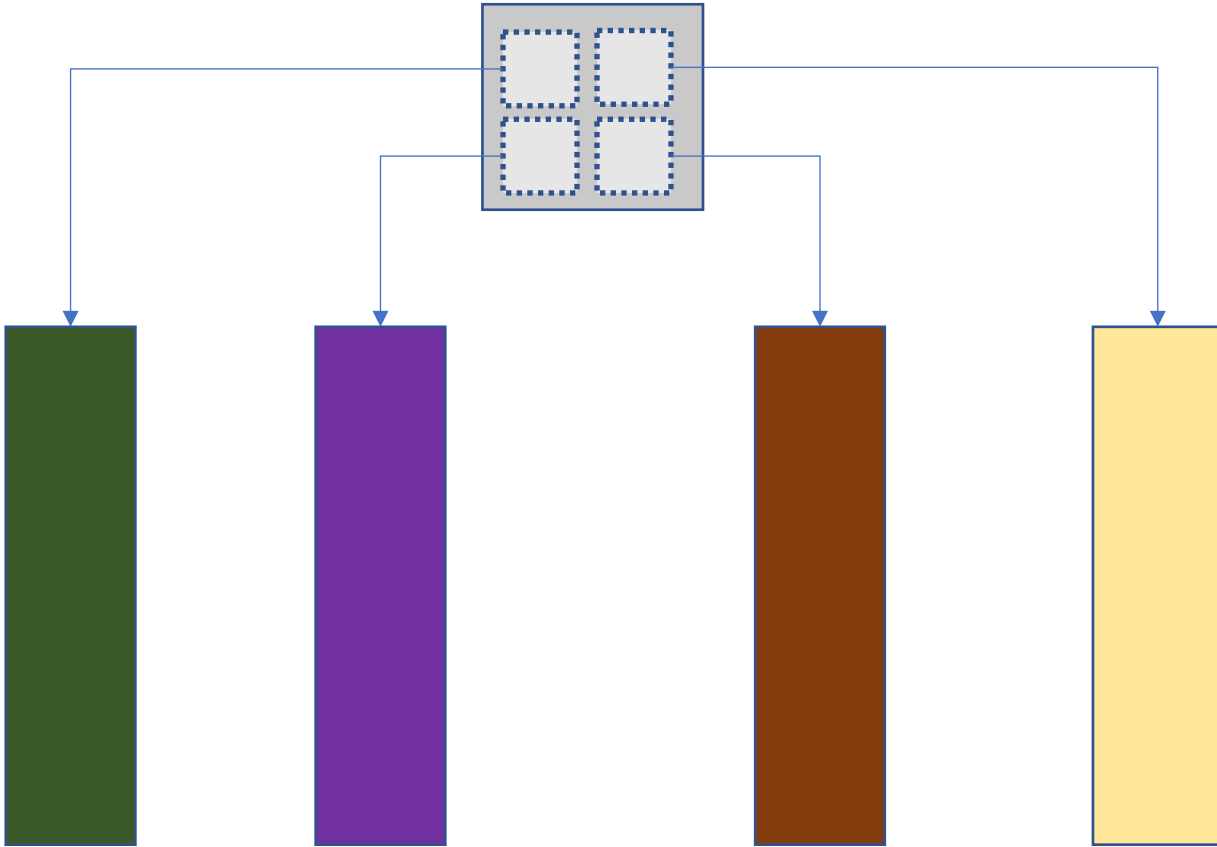
작업1 작업2 작업3 작업4



작업1 작업2 작업3 작업4

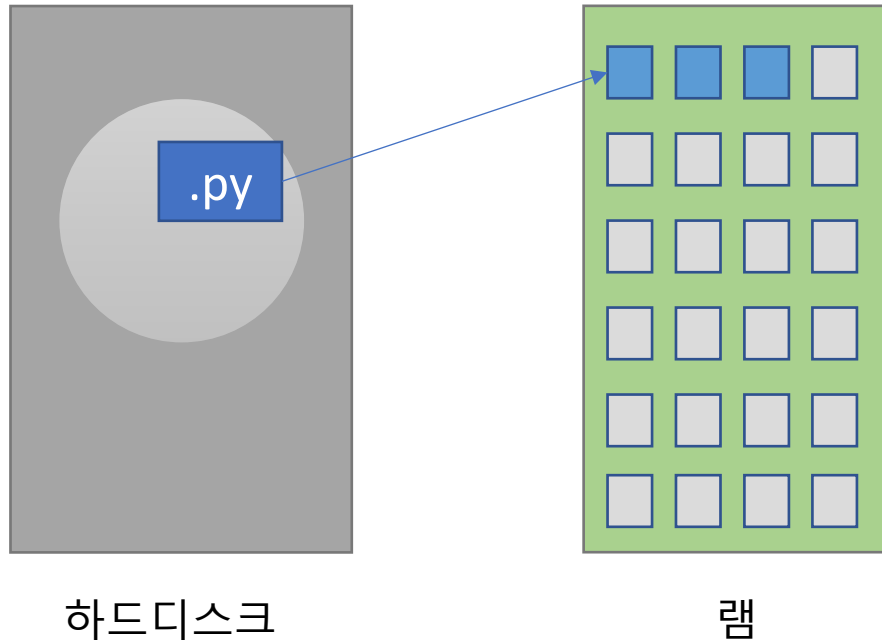
프로세서가 빠르게 돌아가며 마치 여러 개의 작업이 한 번에 이뤄지는 것처럼 보이게 한다.
이 때, 빠르게 돌아가는 기술을 **Context Switching**이라고 한다.

Parallelism (병렬성)



CPU의 여러 개의 코어를 사용해서 여러 개의 작업을 동시에 수행한다

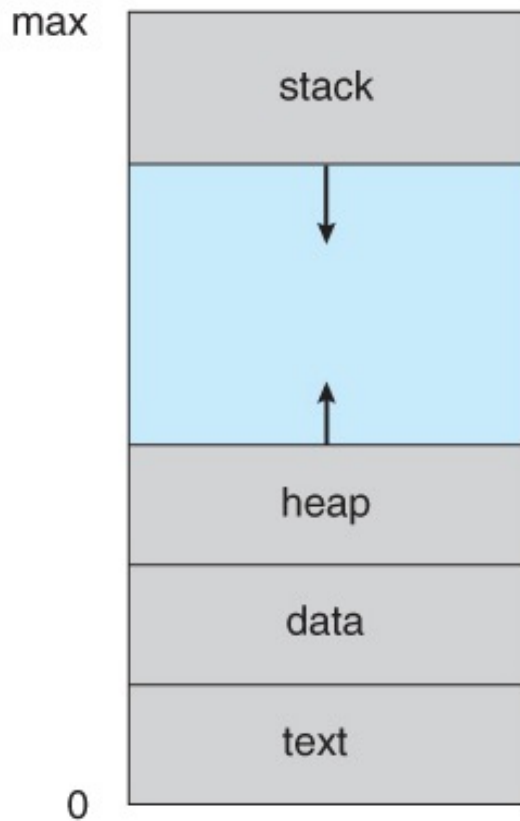
Process 실행 과정



1. 하드디스크에 있는 프로그램을 실행
2. 램에 메모리 할당
3. 할당된 메모리에 바이너리 코드 올라간다.

3번이 이뤄지는 순간부터 Process다.

Process가 할당된 메모리 구성



Stack

프로그램이 자동으로 사용하는 임시 메모리 영역으로 **지역변수, 매개변수, 리턴 값** 등이 잠시 사용되었다가 사라지는 데이터를 저장하는 영역입니다. 함수 호출 시 생성되고 함수가 끝나면 반환됩니다.

Heap (동적 데이터 영역)

필요에 의해 메모리를 동적 할당할 때 사용되는 영역

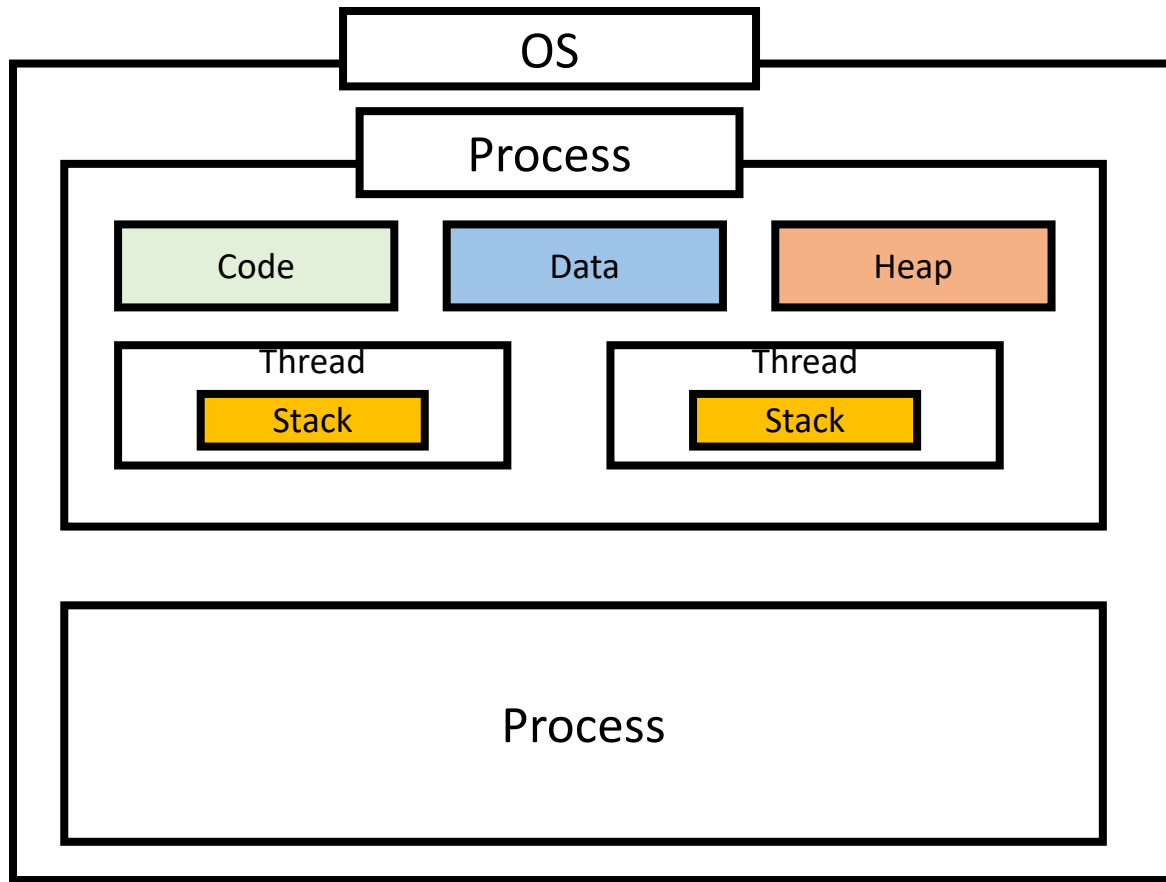
Data 영역

프로그램이 실행될 때 생성되고 프로그램이 종료되면 시스템에 반환되며 **전역변수, 정적변수, 배열, 구조체** 등이 저장됩니다.

Code(Text) 영역

코드 자체를 구성하는 메모리. 영역으로 Hex파일이나 Bin 파일 메모리

Process 구조



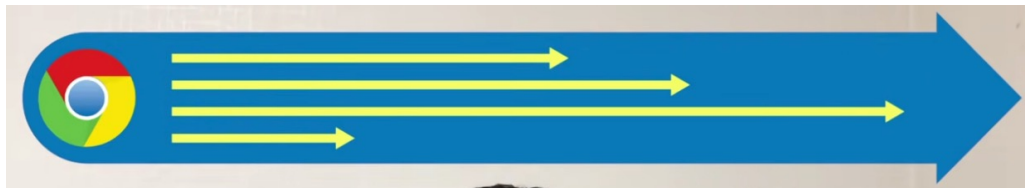
Process

- Process는 프로그램 실행 단위다.
- Process는 메모리 공간을 OS로부터 할당 받는다.
- 하나의 Process는 여러 개의 Thread를 가질 수 있다.
- Parallelism한 특성을 갖는다

Thread

- Thread는 Concurrency한 특성을 갖는다.
- Process 내 Code, Data, Heap 자원을 공유한다.
- 하나의 Thread는 독자적인 Stack을 사용한다.

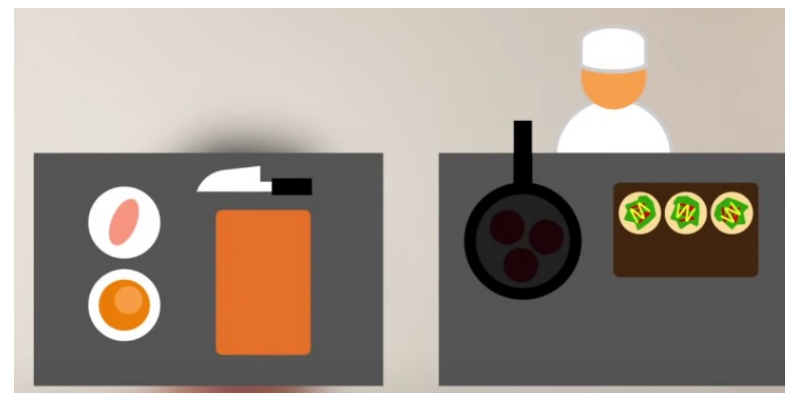
Thread가 필요한 이유



브라우저가 일을 할 때도

- 다른 페이지를 돌아다닐 수 있어야 하고
- 유튜브 동영상을 받을 수 있어야 하고
- 동영상을 재생할 수 있어야 한다.

하나의 프로세스 내에서도 여러 갈래의
작업들이 존재한다. 따라서 Thread가 필요하다



한 메뉴의 스레드들은 같은 조리대에서
작업된다.

공유 자원(Code Heap, Data)을 통해 **효율적으로**
작업할 수 있기 때문이다.

하지만 두 개 이상의 Thread가 동시에 공유되는
자원에 손 대는 **동기화 문제**가 발생할 수 있다.

- Lambda, Closure, Functional Programming
등으로 해결

멀티 프로세스 vs 멀티 스레드 장단점

멀티 프로세스 / 멀티 스레드

1. 멀티 프로세스

: 하나의 프로그램을 여러개의 프로세스로 구성하여 각 프로세스가 병렬적으로 작업을 수행하는 것

장점

- 메모리 침범 문제를 OS 차원에서 해결
- 여러 자식 프로세스 중 하나에 문제가 발생하여도 그 프로세스만 타격, 확산되지 X

단점

- 각 프로세스가 독립된 메모리 영역 (Code, Data, Heap, Stack)을 가지고 있기 때문에 작업량이 많아지면 오버헤드가 발생함 (context switching)
- 프로세스 간의 복잡한 통신 (IPC) 가 필요함

2. 멀티 스레드

: 하나의 응용 프로그램에서 여러 스레드를 구성해 각 스레드가 하나의 작업을 처리하는 것

장점

- 메모리 공간, 시스템 자원의 효율성 증가
- Data, Heap 영역을 이용해 데이터를 주고 받으므로 스레드간 통신이 간단함
- **context switching**시 비용이 적음 (교환해야 할게 적으니까!) -> 시스템 처리량 향상, 프로그램 응답 시간 단축됨

단점

- 서로 다른 스레드가 Stack을 제외한 메모리 공간을 공유하기 때문에 동기화 문제가 발생할 수 있음
- 하나의 스레드에 문제가 생기면 전체 프로세스가 영향을 받음
- 주의 깊은 설계가 필요하며 디버깅이 까다로움