# 프로세스와 스레드,
# 왜 나뉘게 되었을까?

발표자: 러키

# 프로그램과 프로세스

| 프로그램 | 프로세스 |
| --- | --- |
| 어떤 작업을 하기 위해 실행할 수 있는 파일 | 실행되어 작업 중인 컴퓨터 프로그램 |
| 저장 장치에 파일이 있지만,<br>메모리에는 올라가 있지 않은 정적인 상태 | 메모리에 적재되고 CPU 자원을 할당받아,<br>프로그램이 실행되고 있는 상태 |
| 개발자가 작성한 소스 코드 | 소스 코드를 실행한 것 |

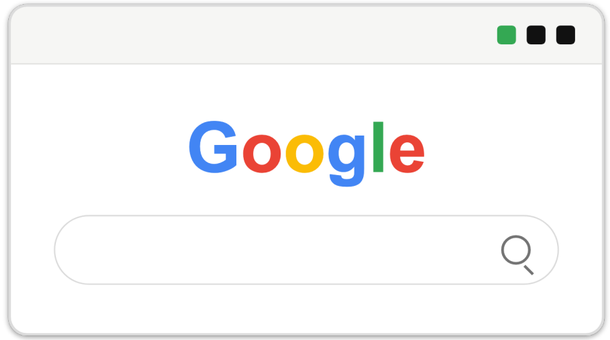
# 프로그램과 프로세스

프로그램

Chrome.exe

실행

프로세스




Chrome Process

실행되면 운영체제는 메모리에 올리고
CPU가 실행할 수 있는 단위가 된다.

# 프로세스의 메모리 영역

## 하나의 프로세스는 사용자 영역에 코드, 데이터, 힙, 스택 영역으로 나뉘어 저장

**Code**
기계어로 이뤄진 명령어

CPU가 실행할 읽기 전용 명령어

화면을 그리는 코드 | 탭 열고 닫는 코드

**Data**
전역/static 변수

프로그램이 실행되는 동안 어디서든 접근 가능한 변수

다크모드 여부 | 언어 설정

**Heap**
동적 할당 메모리

개발자(엔진)가 직접 할당 해제하는 데이터

탭 생성 -> 힙 공간 생성 | 탭 삭제 -> 힙 공간 반납

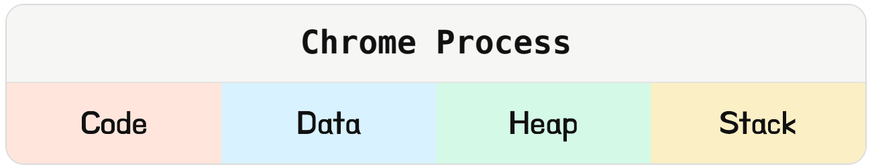
**Stack**
함수 호출, 지역 변수

현재 함수 호출 시 필요한 데이터를 일시적으로 저장

**3. 로딩 화면 표시**
2. 페이지 열기 시작
1. 탭 클릭함

# 하나의 프로세스가 하나의 일만 처리한다면?

## 하나의 작업이 끝나야 다음 작업을 할 수 있다.




**유튜브 재생**

지금 실행 중

→

**파일 다운로드**

재생이 끝나야 시작

→

**검색**

다운로드가 끝나야 가능

**프로세스가 하나뿐이라, 다운로드가 끝날 때까지 검색조차 할 수 없다.**

# 멀티 프로세스의 단점 1 - 메모리 낭비

**프로세스마다 메모리를 새로 만듦**

| 탭 1 프로세스 |
| --- |
| Code |
| Data |
| Heap |
| Stack |

| 탭 2 프로세스 |
| --- |
| Code |
| Data |
| Heap |
| Stack |

| 탭 3 프로세스 |
| --- |
| Code |
| Data |
| Heap |
| Stack |

탭 3개 = 같은 구조의 메모리 3벌

탭이 늘수록 메모리 사용량이 배로 늘어남.

# 멀티 프로세스의 단점 2 - 컨텍스트 전환 무거움

## PCB(Process Control Block)

프로세스의 그 순간 상태를 담아두는 자료구조

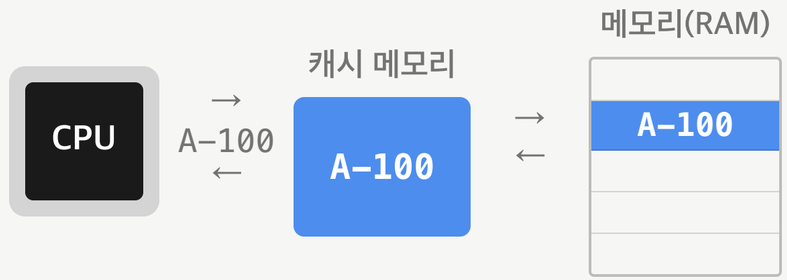
| PCB | Process A |
| --- | --- |
| PID | 1024 |
| State | Running / Ready / Waiting |
| PC · SP · 레지스터 | 다음 실행 위치, 레지스터 값 |
| 스케줄링 정보 | 우선순위, 대기열 상태 |
| 메모리 정보 | 페이지 테이블 위치 |

# 멀티 프로세스의 단점 2 - 컨텍스트 전환 무거움

**프로세스를 바꿀 때마다 큰 비용이 든다**

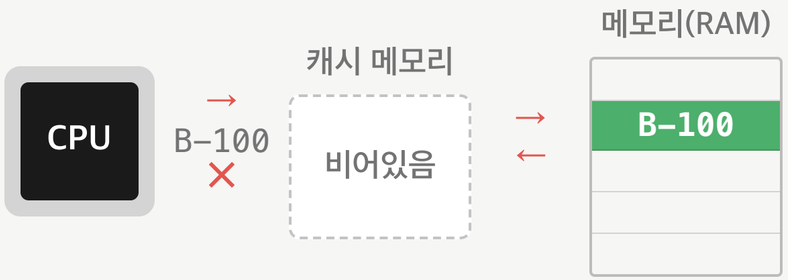
# 멀티 프로세스의 단점 2 - 컨텍스트 전환 무거움

**전환 전** Process A 실행 중 (캐시 적중)




필요한 값이 캐시에 이미 있어 바로 가져옴

**전환 후** Process B로 전환 직후 (캐시 미스)

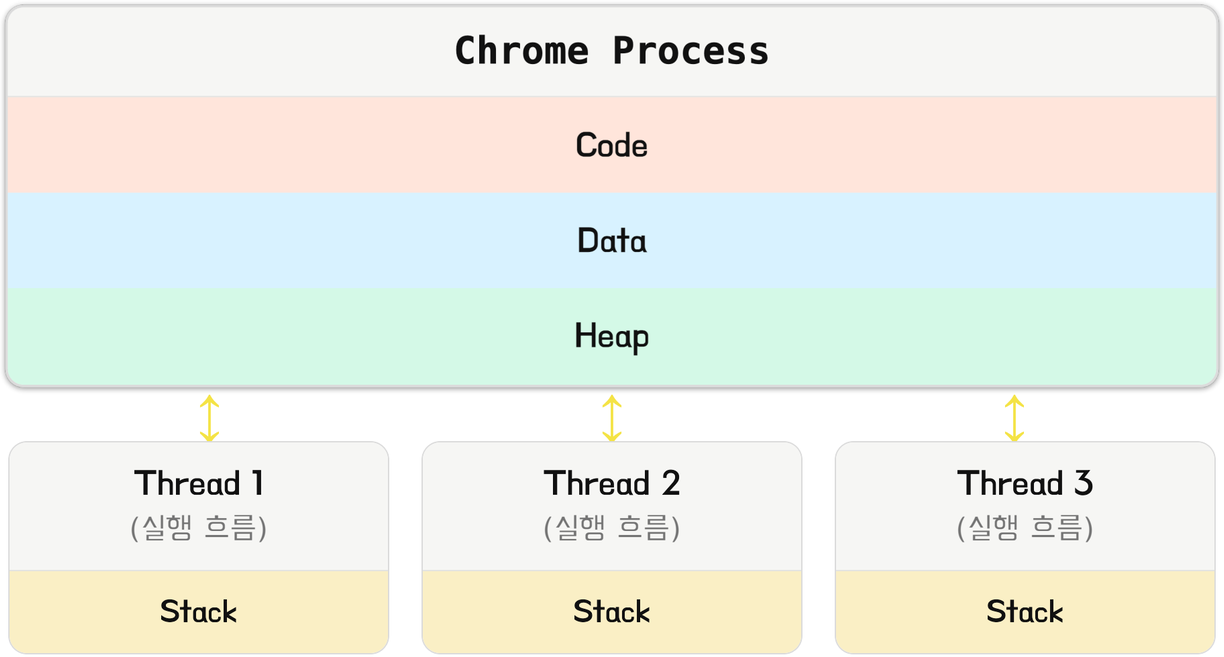



캐시에 없어 메모리까지 가서 가져와야 함 (느려짐)

전환 직후 캐시 미스 몰아서 발생 -> 캐시가 다시 채워질 때까지 CPU가 평소보다 느리게 동작

# Thread 등장 배경

## 하나의 프로세스 안에서 여러 일을 동시에 수행하려면?




스레드들은 Code, Data, Heap을 공유하고, Stack은 따로 가짐

# Thread란?

## 프로세스 안에서 코드를 실행하는 흐름의 최소 단위

예시: Chrome 프로세스 안에서 동작하는 여러 Thread

| JS 실행 | 렌더링 | 네트워크 | 입력 처리 |
| --- | --- | --- | --- |
| 자바스크립트 코드 실행 | 레이아웃 계산 화면 그리기 | 서버 응답 수신 대기 | 클릭 스크롤 이벤트 처리 |

**네 가지 일을 각각 새 프로세스로 만들지 않고, 하나의 프로세스 안 4개의 Thread가 나눠 맡는다.**

# Thread는 왜 Stack만 따로 사용할까?

함수 호출 시 사용하는 값(지역 변수, 매개변수 등)이 서로 달라야 하기 때문!

```
void search(String keyword) {
  String url = "https://google.com/search?q=" + keyword;
  int timeout = 5000;
}
```

**Thread 1**
`search("ChatGPT")`

| Stack |
| --- |
| keyword = "ChatGPT" |
| url = "...ChatGPT" |
| timeout = 5000 |

Stack을 공유하면?

서로 값이 엎어져서
정상 동작 불가

✕

**Thread 2**
`search("YouTube")`

| Stack |
| --- |
| keyword = "YouTube" |
| url = "...YouTube" |
| timeout = 5000 |

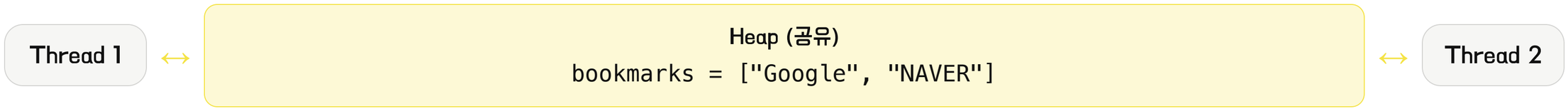
**Stack은 함수 호출의 '작업 공간'이므로 각 스레드마다 독립적이어야 한다.**

# Thread는 왜 Code, Data, Heap을 공유할까?

같은 데이터를 함께 사용해야 협업이 가능하기 때문!

```
List<String> bookmarks = new ArrayList<>();
```

| Thread 1 | Thread 2 |
| --- | --- |
| `bookmarks.add("Google");`<br>`bookmarks.add("NAVER");` | `System.out.println(bookmarks.size());` |




Heap을 공유하면 한 Thread가 만든 데이터를 다른 Thread도 바로 사용할 수 있다.

# 자원 공유로 인한 문제점

## Code, Data, Heap을 공유하기 때문에, 여러 Thread가 같은 자원을 동시에 바꾸면서 Race Condition 발생

재고가 1개 남은 상품을, 두 사용자가 동시에 주문하면?


A, B 모두 '**재고=1' 을 동시에 읽어**, 서로의 변경을 모른 채 각자 결제 진행.

예상 결과

**1건 성공**
**1건 품절**

실제 결과

**2건 성공**

재고는 1건뿐인데 2건이 판매되어 문제 발생 💥

# 동기화 (Synchronization)

## 한 번에 하나의 스레드만 공유 자원에 접근하도록 동기화 수행

Java에서는 어떻게 동기화할까?

```
public synchronized void order() {
    if (stock > 0) {
        stock -= 1;
    }
}
```

한 스레드가 **order()**를 실행하는 동안

다른 스레드는 메서드 앞에서 대기

**synchronized**

메서드·블록에 붙이면 JVM이 잠금/해제를 자동 처리

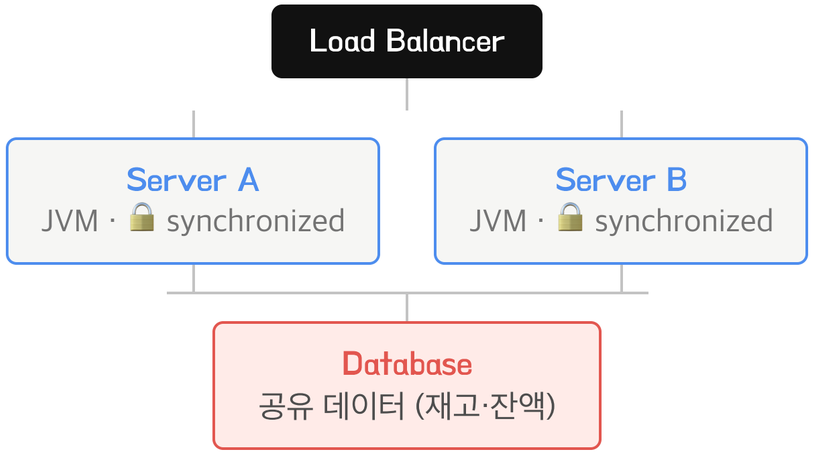
**ReentrantLock**

lock()·unlock()으로 직접 제어, tryLock·타임아웃 등 유연

# 동기화 (Synchronization)

## 주의 synchronized는 한 JVM 안에서만 통한다

실제 서비스는 서버가 여러 대. 서버마다 JVM이 다르면, 각자의 **synchronized**는 서로를 막지 못한다.




A의 락과 B의 락은 서로를 모른다. 둘 다 같은 DB의 재고를 건드리면 결국 충돌이 난다.

**그래서 DB 데이터를 보호할 땐 이렇게:**

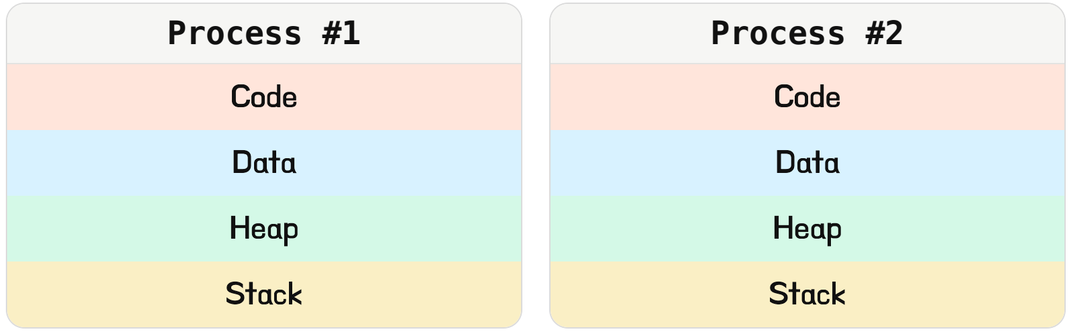
> **비관적 락** — `SELECT … FOR UPDATE`로 행을 잠금

> **낙관적 락** — version 컬럼으로 충돌을 감지

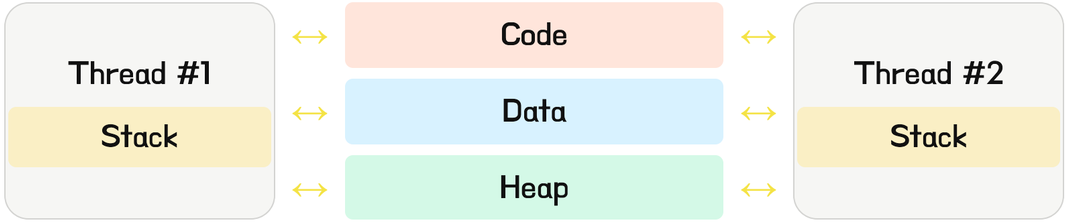
> **분산 락** — Redis 등으로 서버 밖에서 공통 잠금

# 멀티 프로세스 vs 멀티 스레드

## 멀티 프로세스



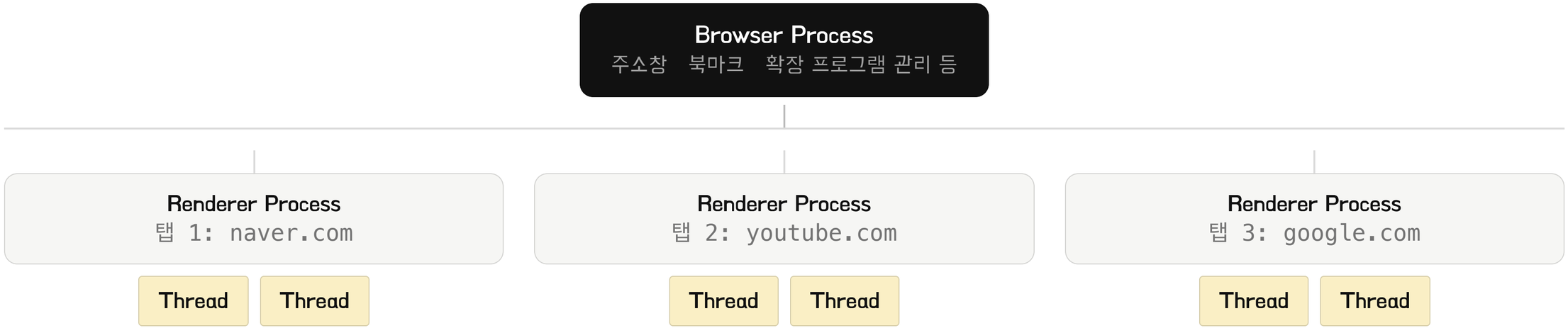

## 멀티 스레드




| 구분 | 멀티 프로세스 | 멀티 스레드 |
|---|---|---|
| **메모리** | 완전히 독립된 공간 | Code Data Heap 공유, Stack만 별도 |
| **생성 비용** | 무겁다 (메모리 전체 할당) | 가볍다 (Stack만 추가 할당) |
| **장애 영향** | 하나가 죽어도 다른 프로세스는 안전 | 하나가 죽으면 같은 프로세스 전체 영향 |
| **응답 속도** | 전환 통신 비용이 커 상대적으로 느림 | 가볍고 자원을 공유해 응답이 빠름 |
| **동기화** | 메모리가 독립적이라 대체로 불필요 | 자원 공유로 동시 접근 제어(Lock) 필수 |

# 실제 Chrome은 어떻게 동작할까?

## 멀티프로세스 + 멀티스레드 구조 함께 사용




✅ 각 탭은 독립적인 프로세스 -> 한 탭이 멈춰도 다른 탭은 정상

✅ 각 프로세스 내부에서는 여러 Thread가 작업을 나눠 처리

**안정성과 성능을 모두 확보!**

꼿