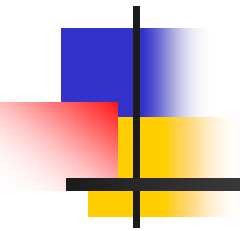


Process와 Thread

배 희호 교수
경복대학교
소프트웨어융합과

성능 문제

- Intel CPU의 발전

	80년대 초반	90년대 초반
반도체 미세 공정 능력	0.18㎛	65㎚
Transistor 집적 능력	약 9.5M개	약 291M
Clock(Hz)	500㎒	3.2㎓

- Gordon Moore의 법칙
  - 반도체의 회로 집적도는 2년마다 2배씩 증가

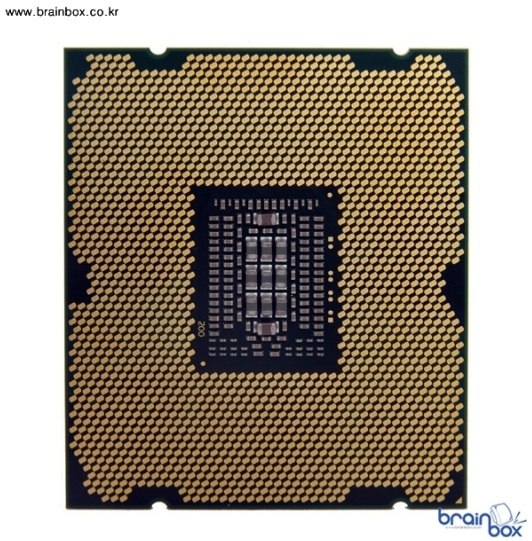
- 성능이 떨어지는 Code -> Hardware 교체로 개선

성능 문제

- Processor 성능의 둔화
  - AMD 4Ghz CPU
    - 매우 많은 전력 소모와 무자비한 발열
  - 전산 업계의 거품 현상
    - 비용 소모(전력 등)는 투자 및 자금 유치로 해결
    - 거품이 빠지면서 공황 상태
  - Green IT, Mobile 시장의 발전
    - 전력 절감만이 살길

성능 문제

- Multi Core CPU로의 진화
  - 공정 기술의 향상으로 CPU에 복수 개의 Core 이식
    - CMP(Chip Multi Processor), 내장 그래픽 등
  - 성능이 비슷하거나 낮은 여러 개의 Core
    - 병렬 처리, 비동기 처리 기법에 대한 연구 진행
    - 단일 Thread Program에서는 성능 저하 유발



i7

Program

- Software의 한 가지로써 어떤 문제를 해결하기 위하여 그 처리 방법과 순서를 기술하여 Computer에 주어지는 일련의 명령문 집합체
- 사용자의 명령에 반응하는 Software를 Program이라 함
- Computer에서 *.exe File을 'Execute File'이라고 하는데, Computer 위에서 동작하며 수행 가능한 여러 가지 명령어를 담아 놓은 상자라고 생각하면 됨
- 이런 Program은 저장소(HDD)에 Binary File(이진 파일)로 저장이 되어 있다가 사용자가 실행 시키면, 즉 Double Click하게 된다면 Memory(RAM)로 옮겨와 실행되는 것임

Multi-Programming

- 초기의 Computer는 하나의 Program을 처리한 후에 다음 Program을 처리해야 했는데, DOS를 사용해 본 사람이라면 쉽게 이해가 갈 것임
- 하나의 Program을 사용할 때 Processor의 처리 속도와 입/출력 속도 간의 차이가 너무 크기 때문에 입/출력 작업이 완료 될 때까지 Processor는 대기해야 함. 이후에 하나의 Program 처리를 마무리해야 다음 Program을 수행 가능함
- 이것은 Processor의 자원을 낭비하는 결과를 가져오게 됨
- Processor가 입/출력 작업의 응답을 대기할 동안 다른 Program(Process)를 수행시킬 수 있도록 하는 것을 Multi Programming이라고 함
- Multi Programming은 Processor의 자원 낭비를 최소화 하기 위해 낭비되는 시간을 다른 Program(Process)을 수행하게 하여, 하나의 Processor에서 여러 Program(Process)을 교대로 수행할 수 있게 하는 것

Processor

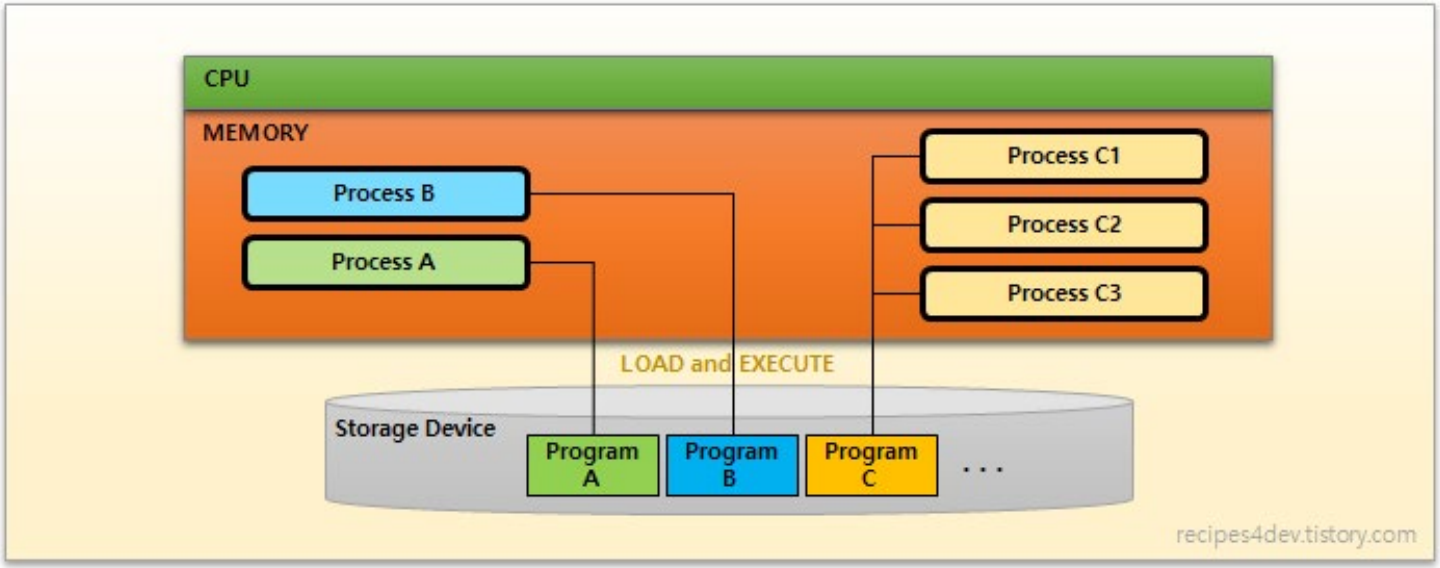
- Processor는 Program들이 실행할 수 있도록 해주는 Hardware
- Computer에 CPU가 Processor 임
  - 예) Intel Core i-7, AMD CPU 등
- MicroProcessor, Network Processor 등 분류를 한다면 여러 가지의 Processor가 있음

Process

- Process는 “실행중인 Program”
- Application Program의 실행 → 적어도 하나의 LINUX Process가 동작
  - 예) Android Phone에 설치되어 있는 날씨 App, News App, Facebook App 등의 Application 중에 실행되고 있는 것을 Process라고 생각하면 됨
- Android는 Multi Process를 지원하는 구조를 가지고 있음
- Process는 주소 공간이나 Register와 같은 자원을 독립적으로 보유하고 실행중인 Program을 의미
- Process는 항상 CPU를 사용하지 않을 뿐만 아니라 Process 수가 증가하면 System의 Overhead도 증가

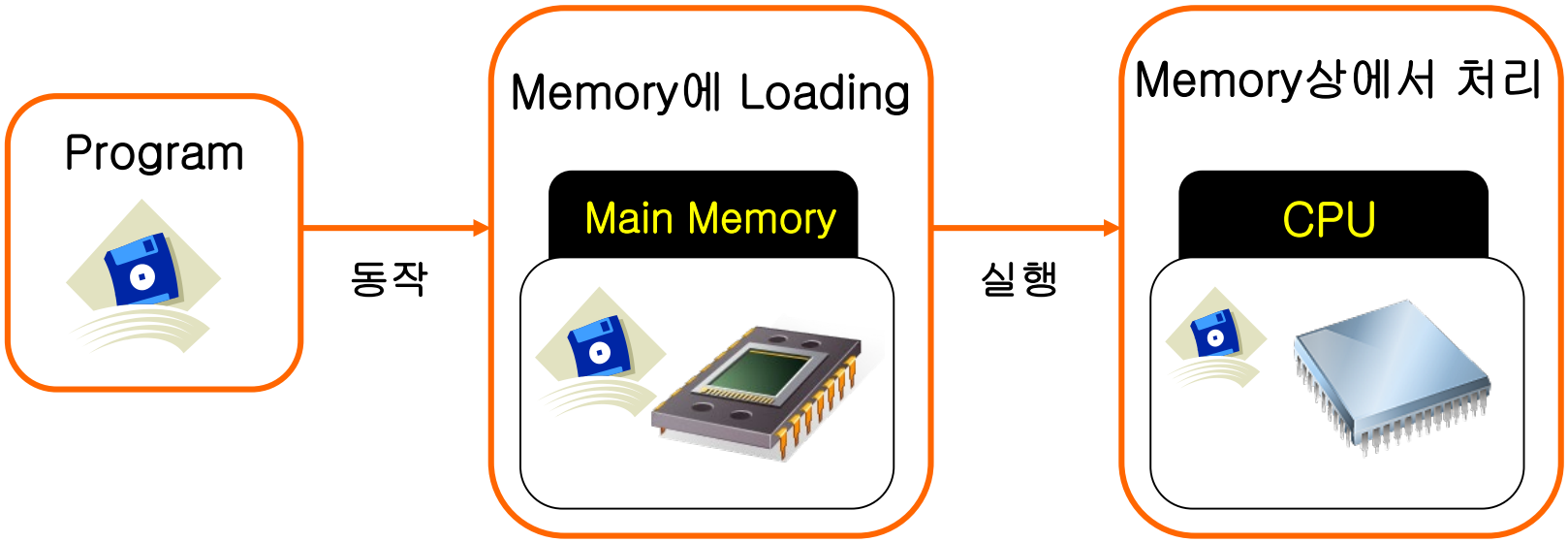
Process

- Program과 Process



Process

- Android와 관련한 Program에서 인스턴스(Instance)와 Process 개념



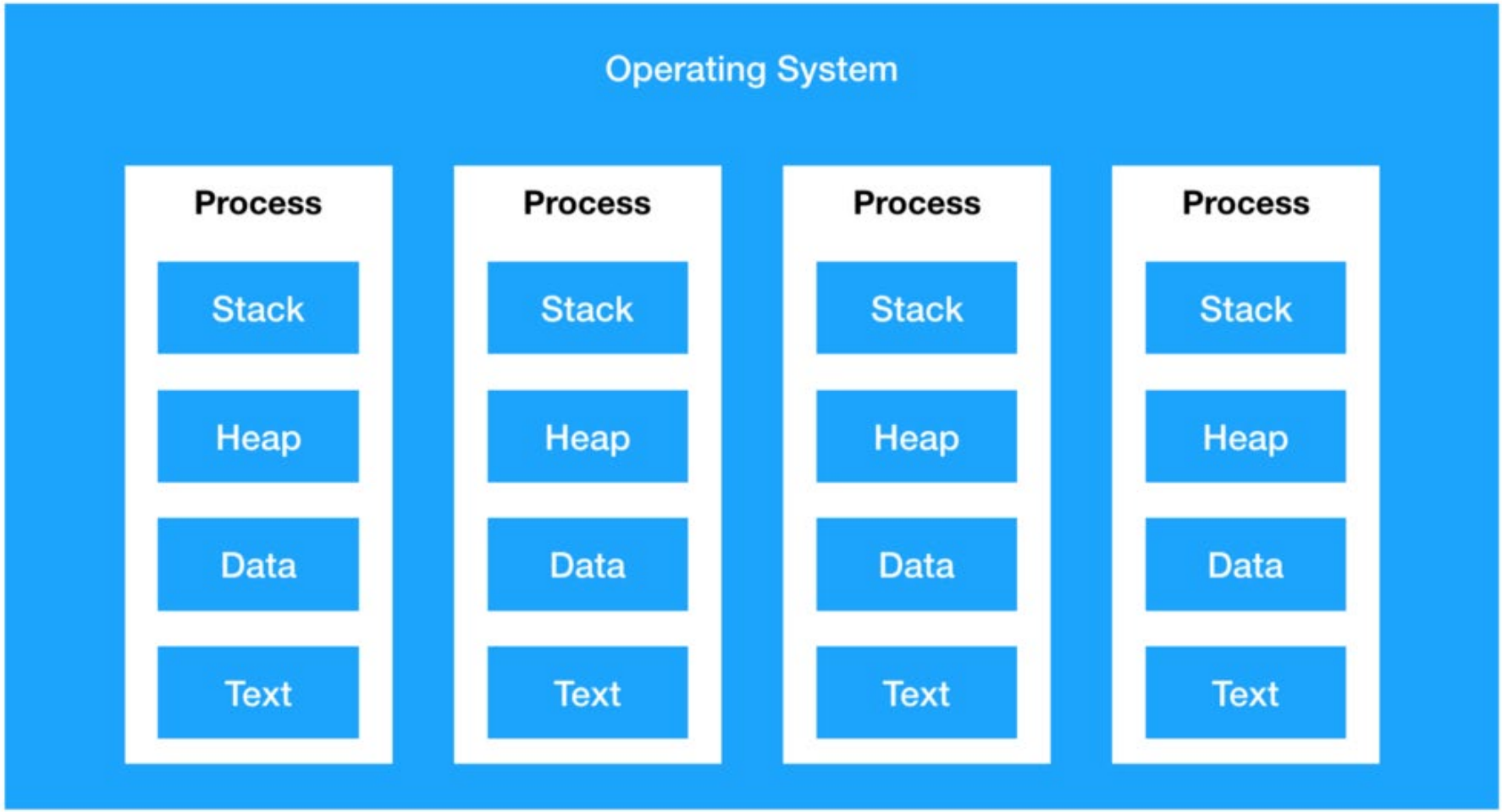
객체 생성
C : malloc()
C++ : new

Process

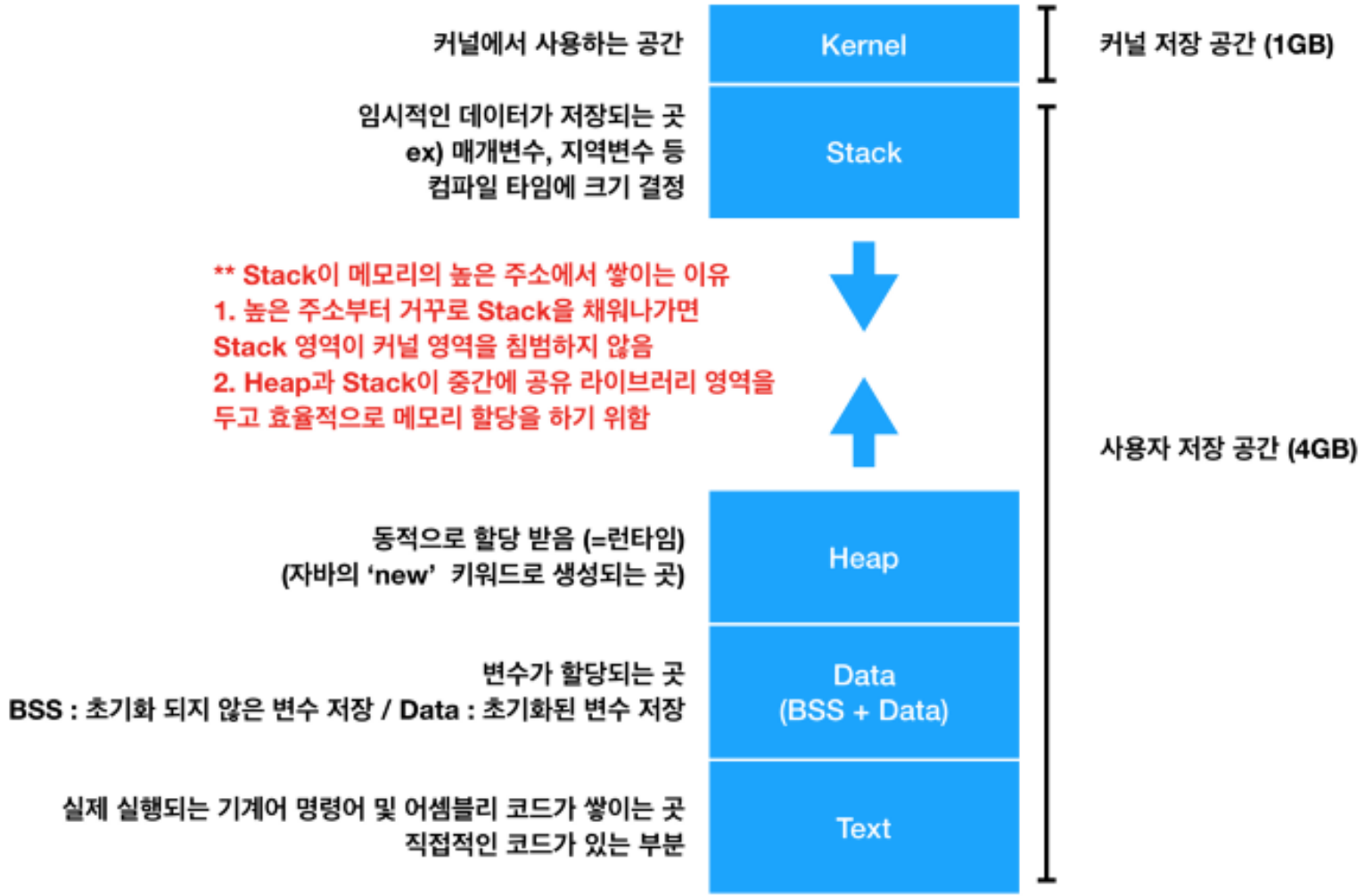
- 의미
  - Memory에 올라와 실행되고 있는 Program Instance
  - 운영 체제로부터 System 자원을 할당 받는 작업의 단위
  - 실행되는 Program

- System으로부터 할당 받는 자원
  - CPU 시간
  - 주소 공간
  - Stack, Heap, Data, Text의 구조로 되어 있는 독립된 Memory 영역

Process



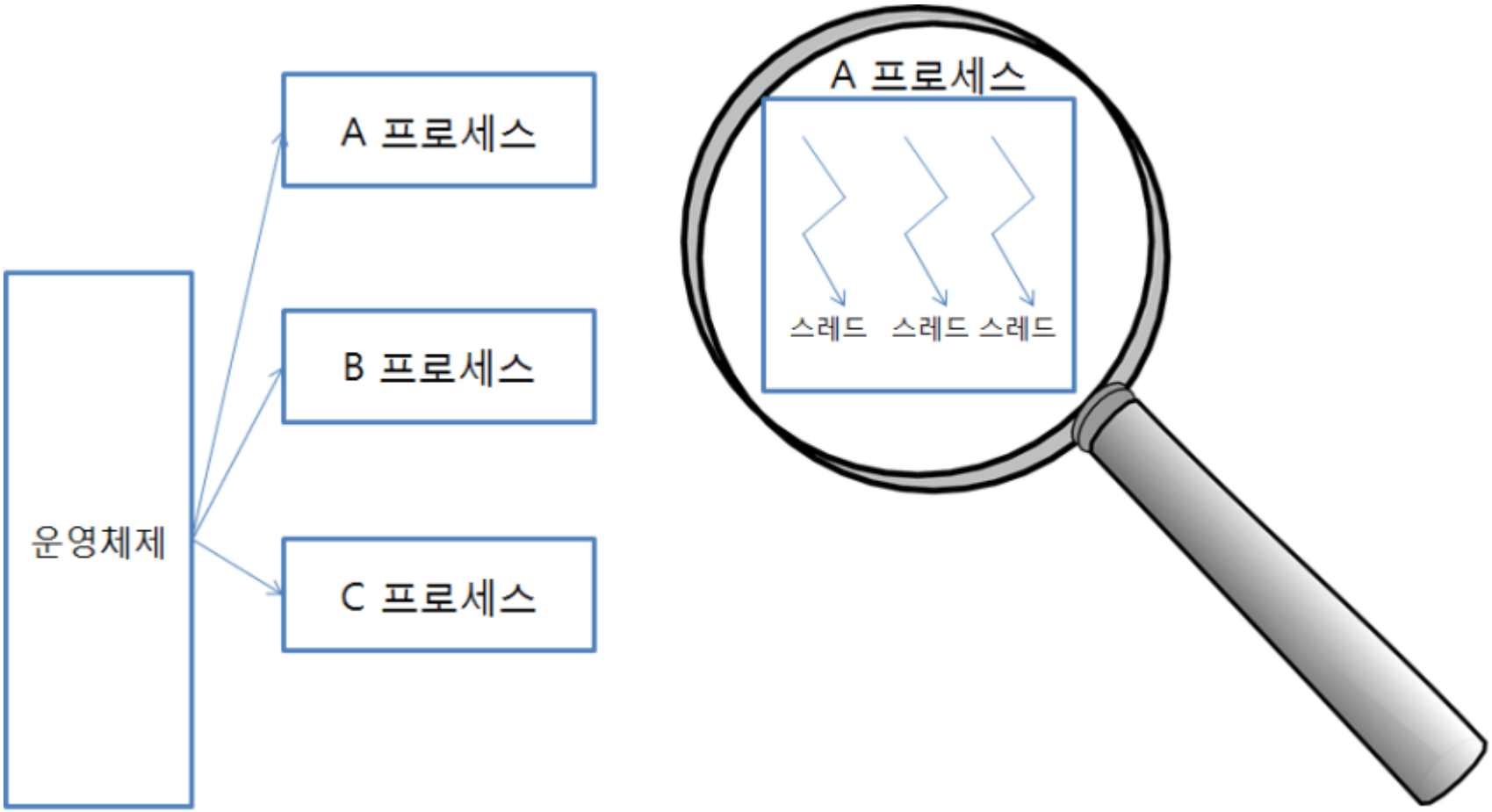
Process



Process

- 특징
  - Process는 각각 독립된 Memory 영역(Stack, Heap, Data, Text)을 할당 받음
  - Process당 최소 하나 이상의 Thread(Main Thread)를 갖음
  - 각각의 Process는 별도의 주소 공간에서 실행되며, 한 Process는 다른 Process의 변수나 자료 구조에 접근할 수 없음
  - 독립된 공간이기 때문에 한 Process가 다른 Process의 자원에 접근하려면 IPC(프로세스간 통신)를 사용해야 함

Process



Multi-Processing

- Multi Processing은 하나의 Processor가 아닌 하나 이상의 Processor가 서로 협력하여 일을 처리하는 것을 말함
- 처리해야 하는 작업이 간단한 경우에는 상관 없지만, 많은 작업을 빠른 시간에 처리하기 위해서는 하나의 Processor가 처리하는 것은 보다 많은 시간을 요구하게 됨
  - 여러 개의 Processor가 하나의 작업을 병렬 처리하는 것이 효율적 임
- 여러 개의 Processor가 하나의 Computer에 있을 수도 있고, 여러 대의 Computer에 있을 수도 있음
- Multi Processing의 개념을 Computer로 나누기 보다는 여러 개의 Processor, 즉 하나 이상의 Processor가 작업을 병렬 처리하는 것으로 정의하는 것이 좀 더 정확한 개념 임

Multi-Processing

- 장점
  - 여러 Process 중 하나가 죽어도 다른 Process에는 영향이 없음 (서로 독립된 공간이기 때문에)
- 단점
  - Context Switching Overhead
    - Context Switching 과정에서 Cache Memory 초기화 등 무거운 작업이 진행되고 많은 시간이 소모되는 등의 Overhead가 발생
  - Process는 각각의 독립된 Memory 영역을 할당 받았기 때문에 Process 사이에서 공유하는 Memory가 없어 매번 Cache에 있는 모든 Data를 Reset하고 다시 Cache 정보를 불러와야 함
  - 서로 별도의 공간을 가지기 때문에 Process간 통신이 어려움. 통신을 위해서 IPC같은 복잡한 기법을 사용해야 함

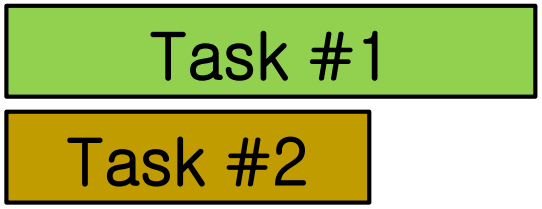
Multi-Tasking

- Task는 어떤 정해진 일을 수행하기 위한 명령어 집합이라고 볼 수 있음
- 하나의 Program은 Program 내에 정의에 따라 하나 또는 그 이상의 Process가 될 수 있음. 하지만 Program이 동작하기 위해서는 Program 실행으로 Memory에 Load된 Process 외에도 이미 수행되고 있는 운영 체제 상의 많은 Process와 상호 작용이 필요하게 됨
- 하나의 Task라는 개념은 Process의 개념보다 조금 확장된 개념이라고 생각하면 됨. 이러한 Task가 하나의 Processor 상에서 운영 체제의 Scheduling에 따라 조금씩 번갈아 가면서 수행되는 것이 Multi-Tasking
- Multi Programming과 다른 점은 Multi Programming이 낭비되는 자원을 최소화하기 위해 교대로 실행하였다면, Multi Tasking은 좀 더 확장하여 정해진 시간 동안 교대로 Task를 수행 함

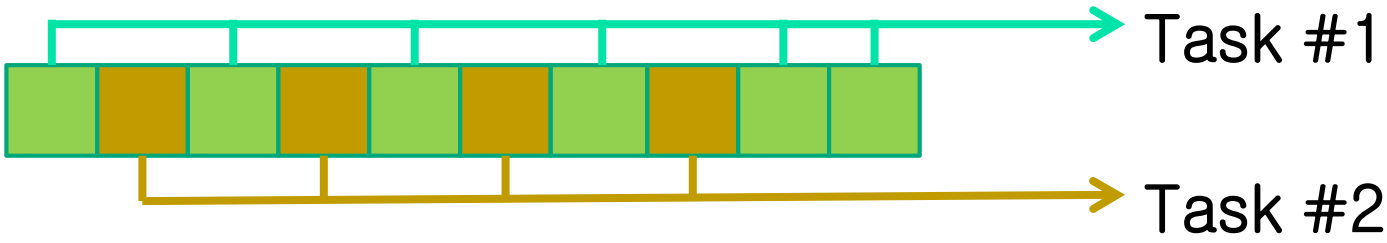
Multi-Tasking

- 단일 CPU에서의 처리 방법

- Multi Core CPU에서의 병렬 처리 방법



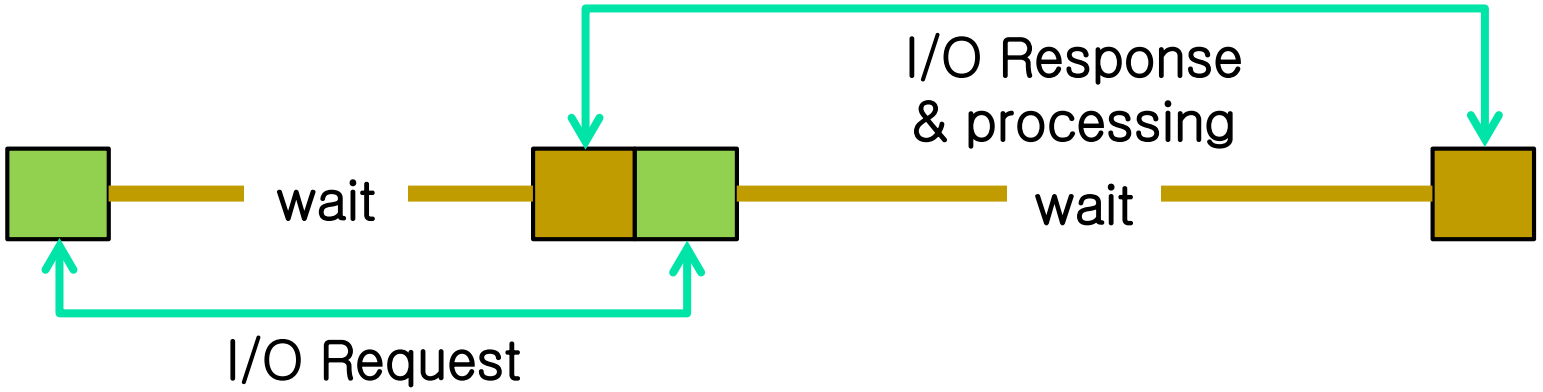
- 단일 CPU에서의 동시 처리 방법 (Round Robin 방법)



Multi-Tasking

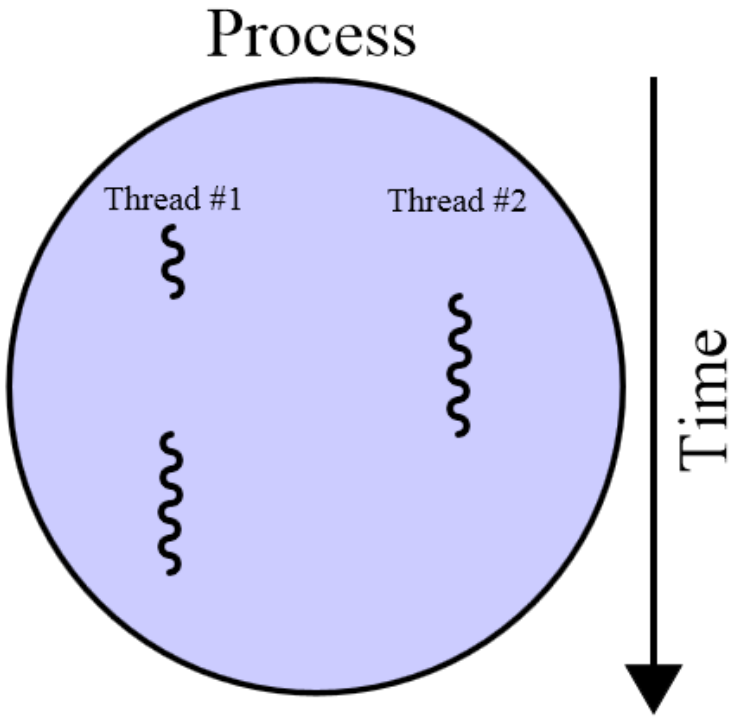
- Latency(지연 시간)
  - 요청 후 실제 응답이 오기까지 걸리는 대기 시간
  - CPU가 다른 장치에 비해 매우 빠르므로 생기는 시간

- Latency 중 CPU에서 다른 Thread의 작업을 처리
  - I/O를 비동기로 처리하므로 Latency Hiding 가능



Thread

- Thread는 하나의 Process 내에서 동시에 수행되는 작업 (Task) 단위를 말함
- 일반적으로 한 Program은 하나의 Thread를 가지고 있지만, Program 환경에 따라 2개 이상의 Thread를 동시에 실행할 수도 있음



Thread

- Process가 동시 작업을 하는 것처럼 Process 내에서 동시 작업을 하는 주체가 있는데 이것을 Thread라고 함
- 다른 용어로는 경량 프로세스(LightWeight-Process)라고 함
  - 아침을 준비할 때 밥솥의 취사 버튼 누르고 밥이 되는 동안 식탁을 차리는 것과 유사함
  - 국을 끓인다 던가, 계란말이를 한다던가, 보통 압력 밥솥의 취사 버튼을 누르고 아무 것도 하지 않다가 완료 후 식사를 준비하지는 않음
- 이처럼 무언가를 백그라운드(Background)로 돌려놓고 다른 여러가지 일을 하는 것이 Thread라 고 할 수 있음

Thread

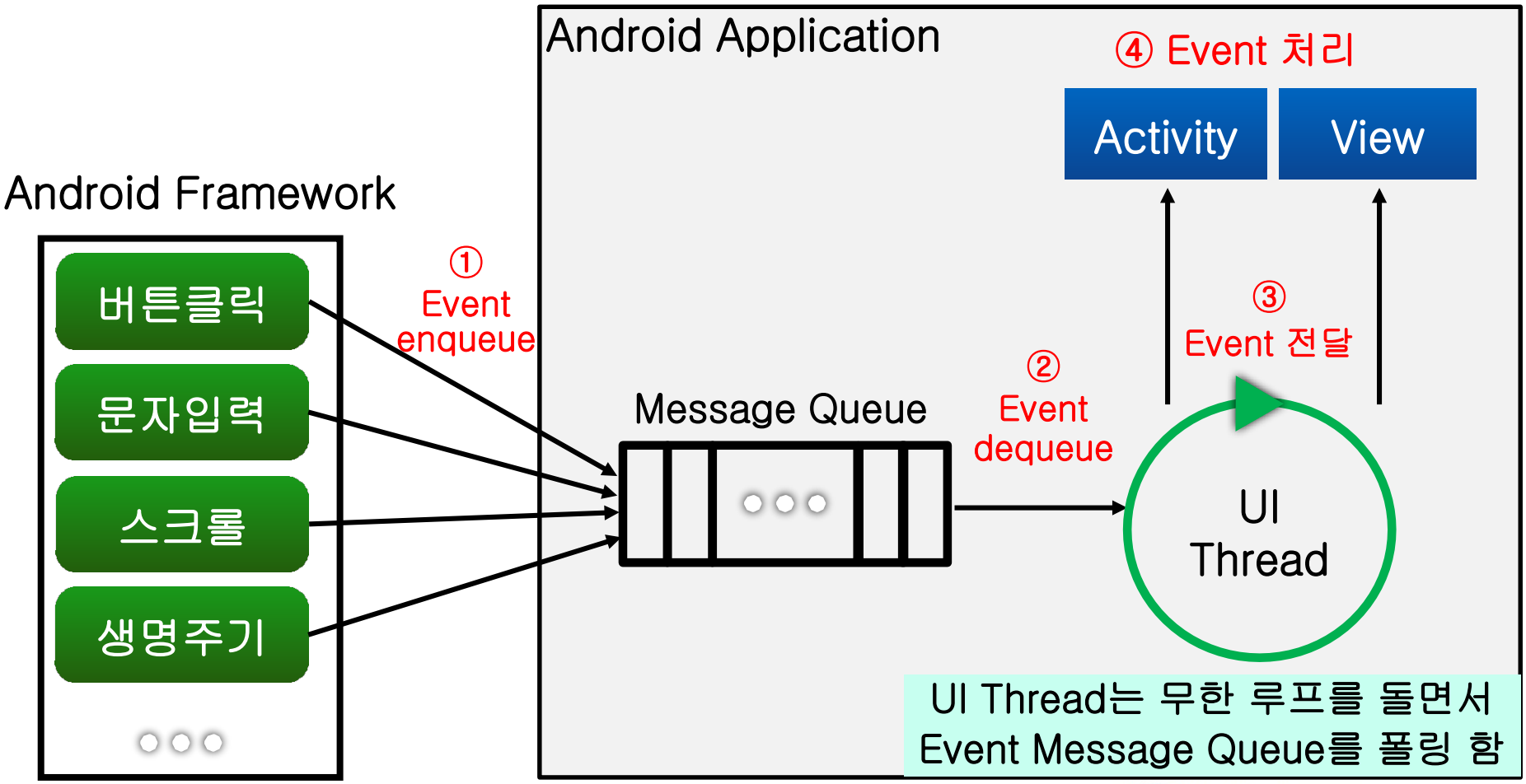
- CPU를 효과적으로 사용하고 다수의 Process에 의한 Overhead를 줄이기 위하여 Thread 사용
- Thread는 Process에서 독자적인 명령어 제어권을 가지며 Process에 포함된 자원을 공유하는 Program의 실행 단위를 의미
- Thread를 지원하지 않는 System은 Process를 CPU의 Scheduling 단위로 사용하지만 Thread를 지원하는 System은 Thread를 Scheduling 단위로 사용
- 대부분의 운영체제는 Thread를 지원

Thread

- 의미
  - Process의 특정한 수행 경로
  - Process가 할당 받은 자원을 이용한 실행 단위
- 특징
  - Thread는 Process내에서 각각 Stack만 할당 받고 Code, Data, Heap은 Process내의 다른 Thread들과 공유
  - Thread는 한 Process내에서 동작되는 실행의 흐름으로 Process내의 주소 공간이나 자원들을 같은 Process내의 Thread들과 공유
  - 같은 Process내의 Thread들은 Heap 공간을 공유하지만 Process는 다른 Process의 Memory에 직접 접근 불가능
  - 각각의 Thread는 별도의 Register와 Stack을 갖지만, Heap Memory는 서로 읽고 쓸 수 있음
  - 한 Thread가 공유된 Process 자원을 변경하면 다른 Thread들도 변경 결과를 즉시 볼 수 있음

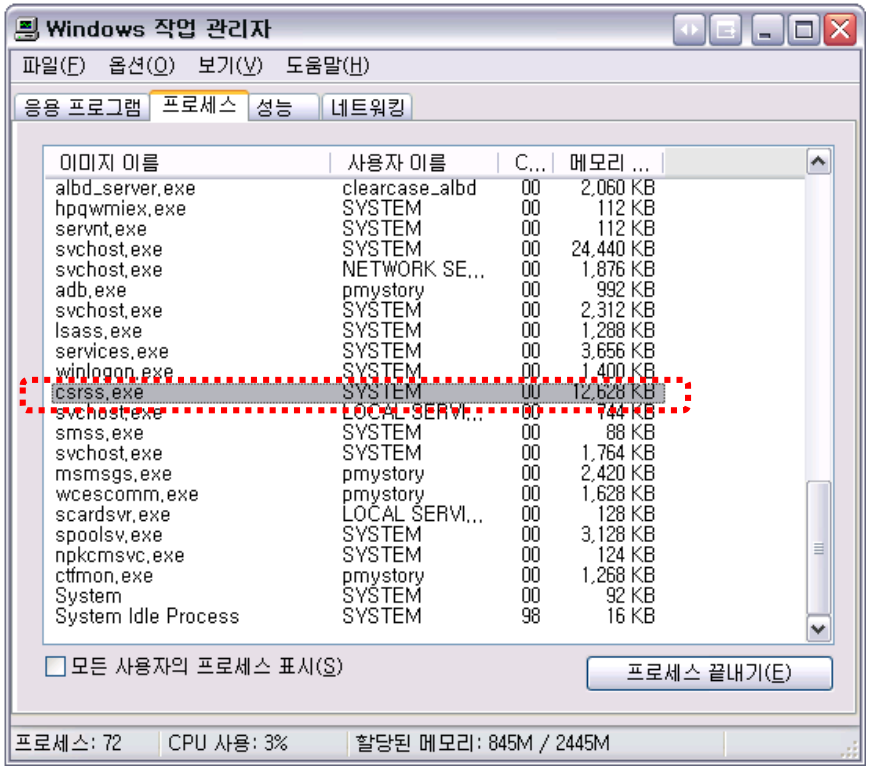
Thread

- UI Event 처리 – Single Thread Model



Process와 Thread

- Process는 반드시 하나 이상의 Thread가 있음
  - Process 혼자서는 아무런 일도 할 수 없음
- 예) Windows System에서는 csrss.exe Process가 순회하면서 Thread를 관리하고, Kernel에게 Thread의 실행을 알림



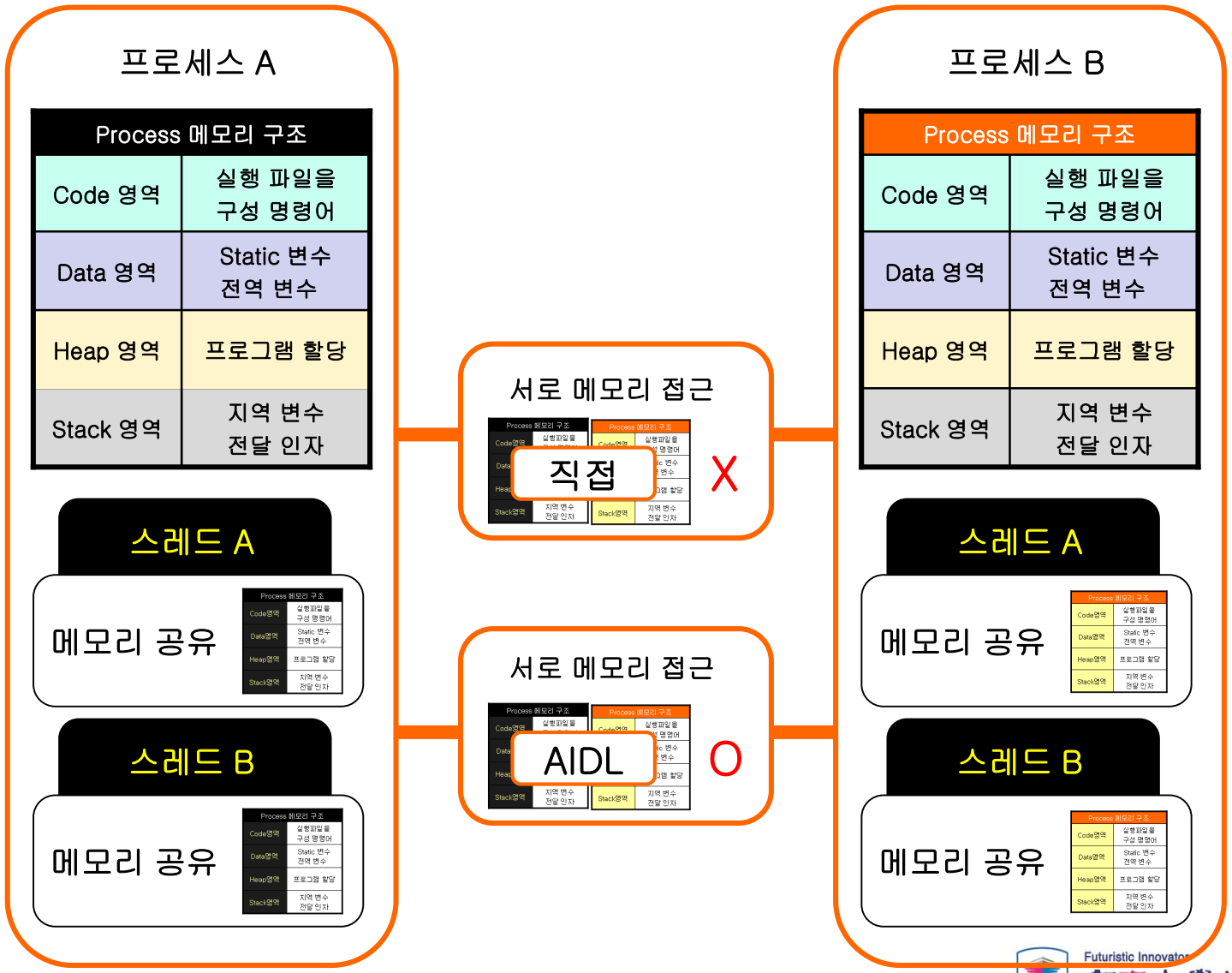
프로세스

Process와 Thread

- Android는 기본적으로 Multi Process와 Multi Thread를 지원

- Android에서 App 하나를 실행하면 Process 하나가 생성
- Process에는 App에서 사용될 각종 자원과 그 자원을 가지고 처리할 Thread가 있음
- 예) 한 회사를 Process로 비유했을 때 회사 내 각종 서류, 책상, 의자, Computer 등이 자원이고, 그러한 자원을 가지고 일하는 직원을 작업자라고 할 수 있음. 여기서 자원은 Memory고 작업자는 Thread에 해당함

Process와 Thread



AIDL(Android Interface Definition Language)

Process와 Thread

- 각 Process 간에는 자원을 공유할 수 없음
- Process는 다른 Process 간섭 없이 완벽하게 독립적으로 처리되며, 만일 특정 Process에 문제가 발생하더라도 다른 Process에 영향을 주지 않고 자신만 종료하게 됨
- Thread를 이용하면 동시 처리가 가능함. 단 하나의 Thread는 단 하나의 실행 흐름만 가지지만, 여러 개의 Thread를 생성하면 여러 실행 흐름을 가질 수 있기 때문임

Process와 Thread

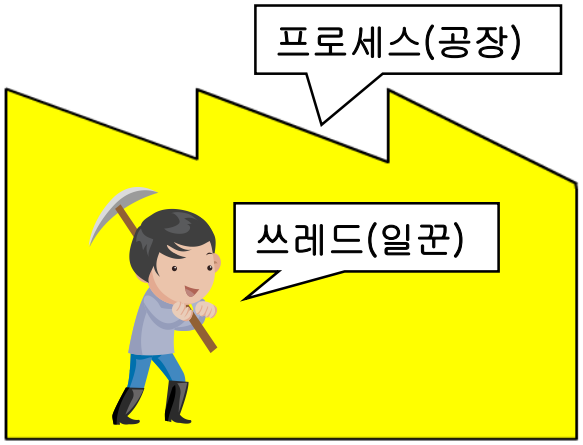
- Application이 시작되면 Android System은 새로운 LINUX Process를 생성
- 기본적으로 Application 안의 모든 Component들은 동일한 Process의 동일한 Thread로 실행
- 이 기본적인 Thread를 Main Thread (Activity 자체에서 제공하는 기본 작업)라고 부름
- 이 Main Thread외에 별도로 추가된 작업을 Worker Thread라 함

Process와 Thread

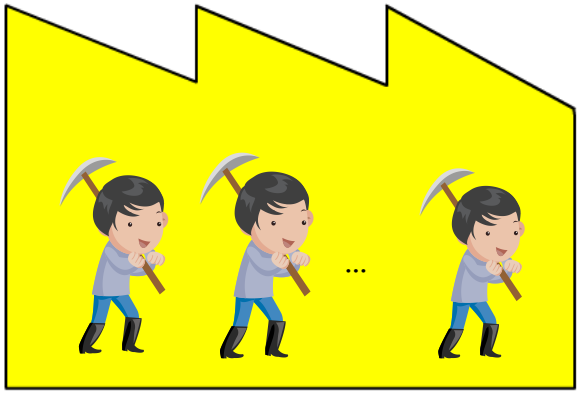
- Process
  - 실행 중인 Program, 자원(resources)과 Thread로 구성
- Thread
  - Process내에서 실제 작업을 수행
  - 모든 Process는 하나 이상의 Thread를 가지고 있음

Process : Thread = 공장 : 일꾼

▶ Single Thread Process
= 자원 + Thread



▶ Multi Thread Process
= 자원 + Thread + Thread + …

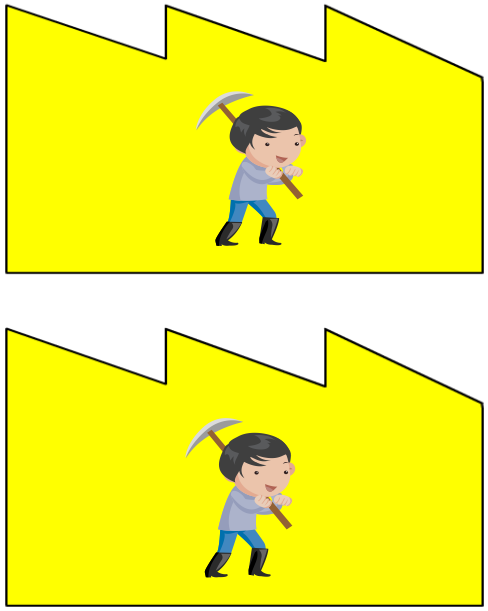


Process와 Thread

- MultiProcess vs. MultiThread

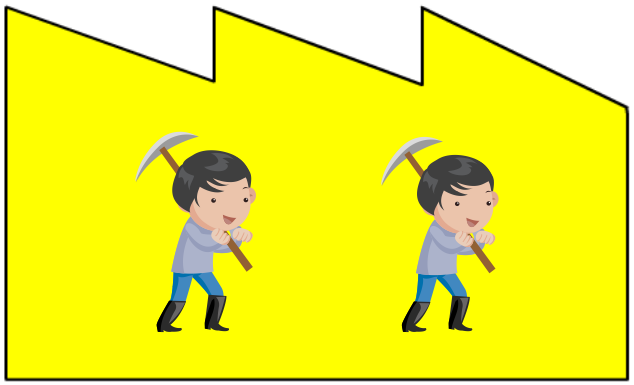
"하나의 새로운 Process를 생성하는 것보다, 하나의 새로운 Thread를 생성하는 것이 더 적은 비용이 든다."

2 Process 1 Thread



VS

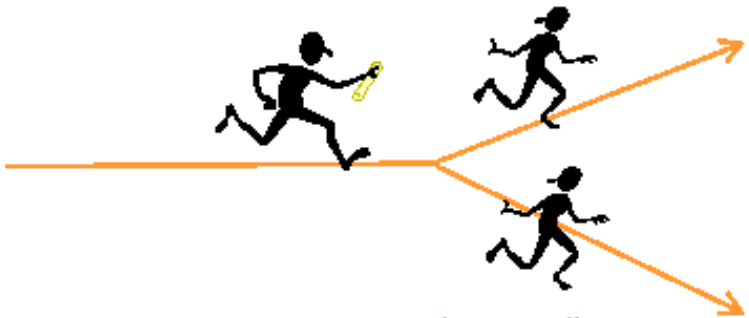
1 Process 2 Thread



Multi-Thread

- 하나의 Application이 동시에 여러 가지 작업을 하는 것
- 이들 작업은 Thread라고 불림
- Android는 JAVA의 Threading Model을 동일하게 지원함

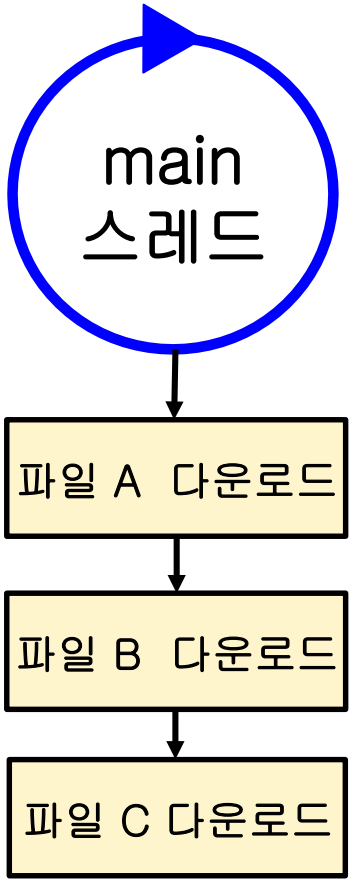
단일 스레드



다중 스레드

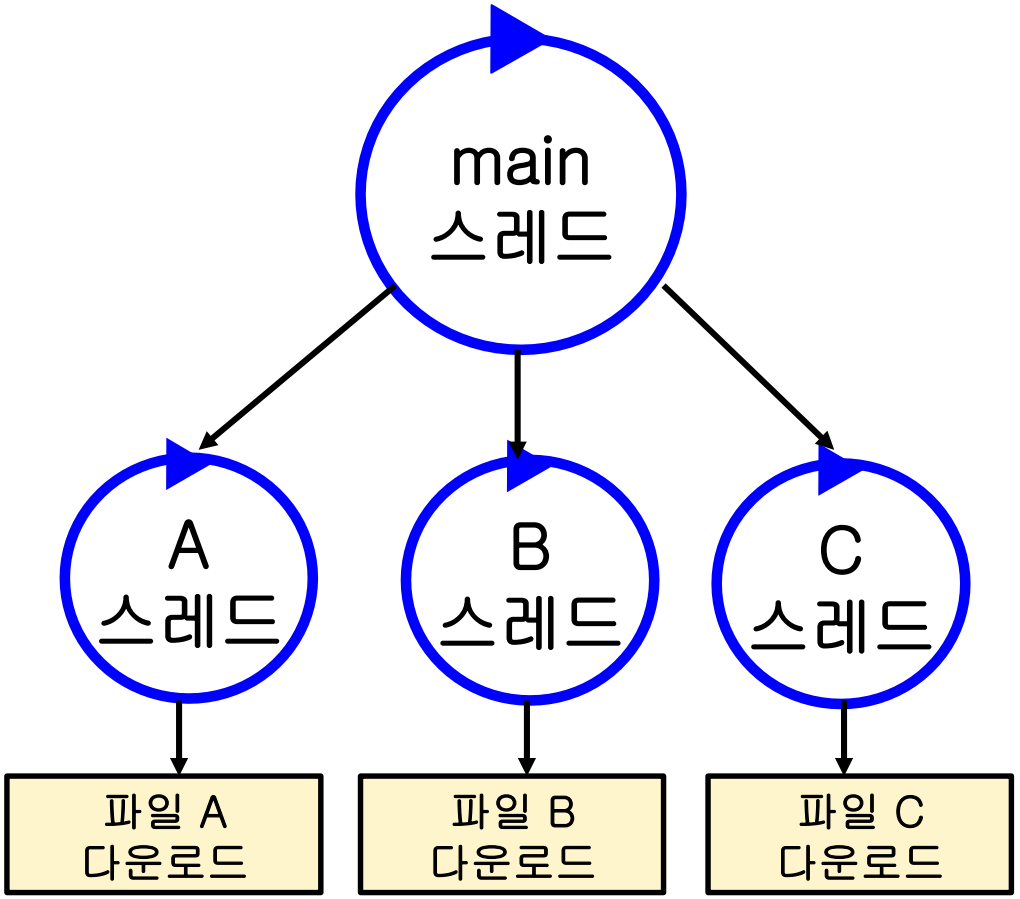
Multi-Thread

Single Thread



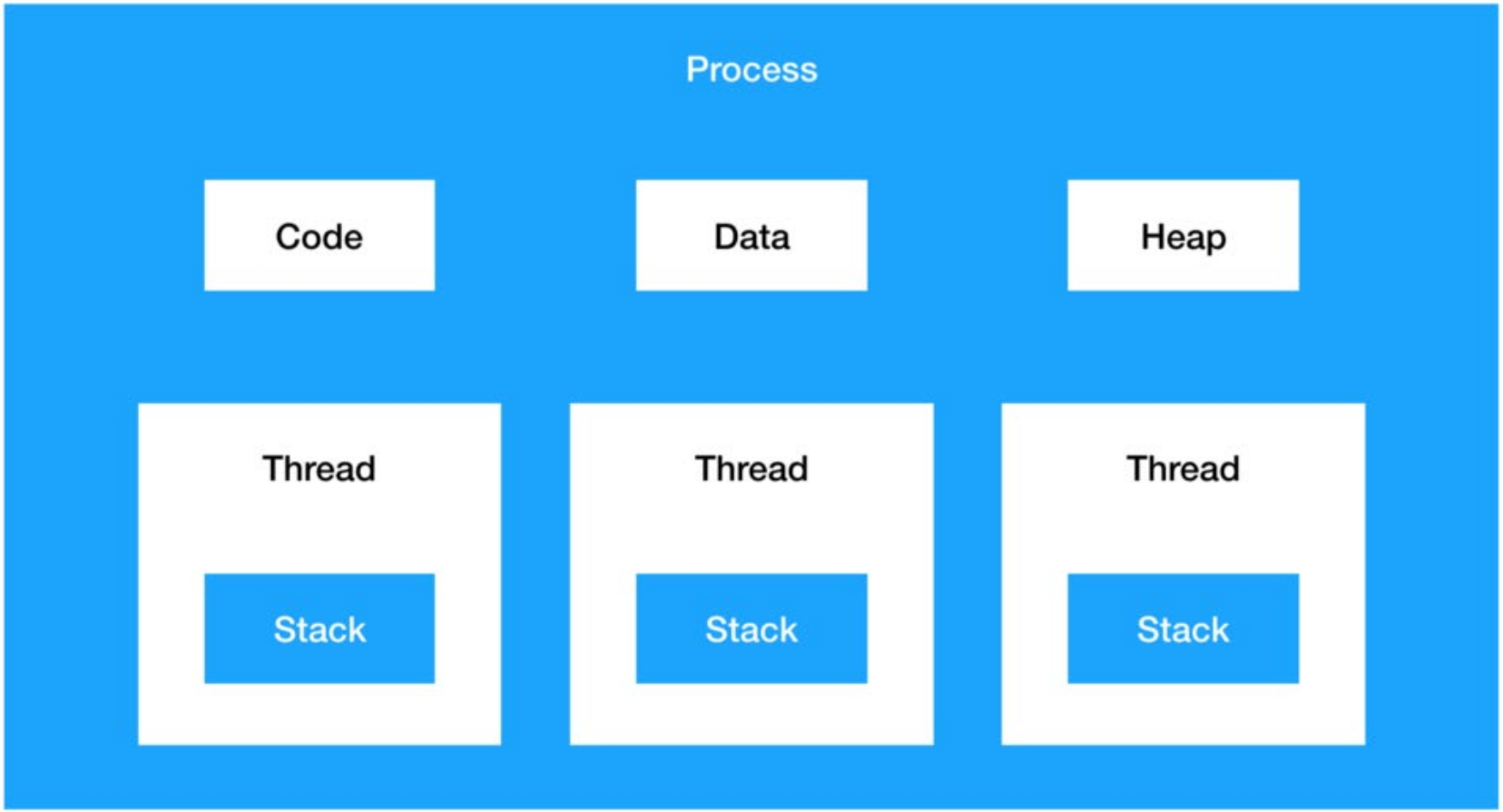
순차 실행(Sequential)

Multi Thread



병행 실행(Concurrent)

Multi-Threading



Multi-Threading

- Thread는 Process내에서 생성되는 하나의 실행 주체
- 한 Process 내에서 생성되는 것으로 여러 개가 동시에 생성이 가능
- 또한 생성된 여러 Thread는 하나의 공유 Memory를 가지게 됨
- 그렇기 때문에 서로간의 정보를 주고 받는데 최소한의 Overhead로써 제한이 많이 없는 편임
- 예) Networking을 지원하는 Program이 있다고 가정하면, 한 Program 내에 사용자가 접속할 때마다 사용자 각각을 처리할 수 있는 처리 Module이 생성되어야 하는데, 이것이 Thread라고 볼 수 있음

Multi-Threading

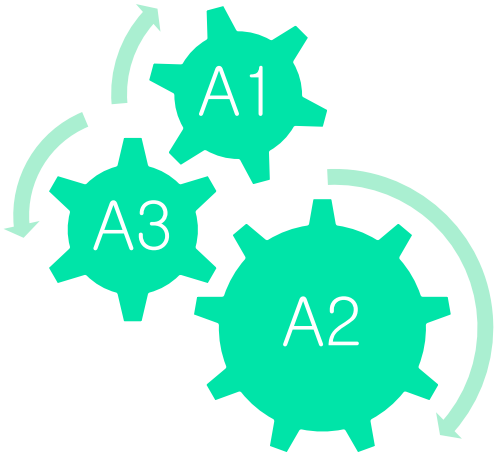
- 장점
  - 응답성
    - 다른 Thread의 실행 시간이 길거나 입출력을 위해 Block 되더라도 계속 실행을 허용
    - Application은 UI를 전담하는 Main Thread와 느린 Task는 Background Thread로 나누어 처리함
  - 자원 공유
    - Threads는 Process의 리소스를 공유하지만 독립적으로 실행할 수 있음
    - Process(코드 및 데이터 등)의 자원을 자동적으로 공유

Multi-Threading

- 장점
  - 경제성
    - Process의 생성과 Context Switching보다 빠르고 자원 사용이 적음
    - Threading은 Concurrent execution(동시 실행)이 유용함
  - 확장성
    - Multi Processor System에서 병렬성 증가
    - Multi-Threaded Program은 다중 CPU가 있는 Computer System에서 더 빠르게 작동함

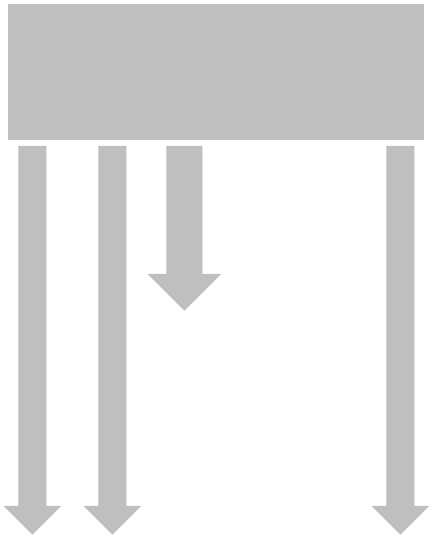
Multi-Threading

- 단점
  - 주의 깊게 설계해야 함
  - Debugging의 어려움
  - 다른 Process에서 Thread를 제어할 수 없음 (Process간 서로 다른 공간이기 때문에)
  - Multi Thread의 경우 자원 공유의 문제가 발생할 수 있음 (동기화 문제)
  - 하나의 Thread에 문제가 생기면 전체 Process가 영향을 받음
  - DeadLock을 감지, 회피, 해결해야 함



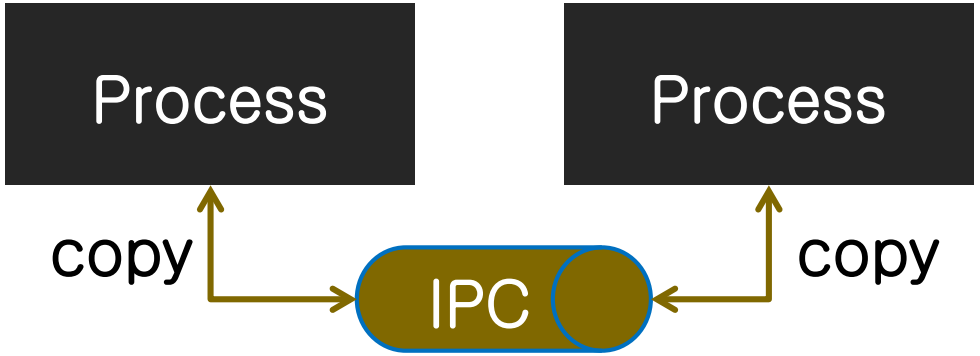
Multi-Threading

- Threads http://developer.android.com/reference/java/lang/Thread.html
  - Thread는 동시 실행 단위임
  - 각 Thread에는 자체 호출 Stack이 있음. 호출 Stack은 호출 된 메소드의 로컬 변수에 대한 메소드 호출, 매개 변수 전달 및 저장에 사용
  - 각 가상 머신 Instance에는 하나 이상의 기본 스레드(Main Thread)가 있음
  - 동일한 VM의 Thread는 이러한 개체와 연결된 공유 개체 및 모니터를 사용하여 상호 작용하고 동기화함

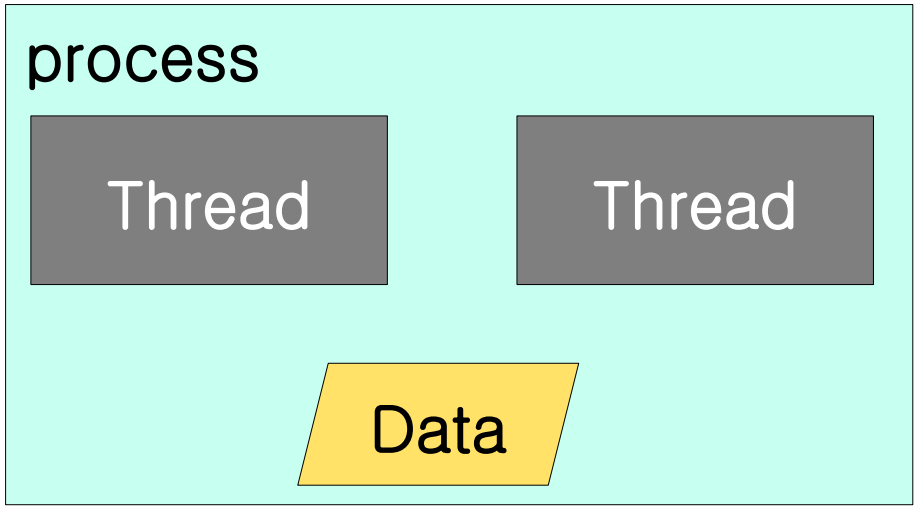


Multi Thread

- Multi Process



- Multi Thread



Multi Thread

- Program의 성능 향상을 위한 I/O 및 저장장치 접근 최소화를 위한 2가지 방법
  - Multi Thread
    - Process 내부 처리 구조를 Multi Process처럼 분업화
    - Thread 간 통신에 따른 비용 소모가 거의 없음
    - Thread 간 동기화 처리 비용 소모
  - Multi Process
    - 동기적 Programming Model
    - 보호되는 자원(Memory, 장치 등)
    - Process간 통신에 따른 비용 소모(IPC 등)

Multi Thread

- Multi Threading은 최후의 보루
  - 개발, 테스트, 디버깅이 복잡해짐
  - 설계나 개발 단계에서 예측하지 못한 상황에 빠질 가능성이 있음
  - “프로그래머가 관찰을 위해 개입하는 순간, 비결정적인 결과를 나타내어 버그가 사라지는 현상”

Multi Thread

- 고려 사항
  - I/O 측면에서 병목(bottleneck)이 제거된 상태인가?
  - Multi Thread 도입 전의 Code는 다른 최적화 방법을 적용해 보았는가?
  - Multi Thread 도입 전의 System에 가용할 수 있는 CPU나 관련 유휴 자원은 충분한가?
  - 효율을 극대화하려면 Thread의 숫자를 어떻게 결정해야 하는가?

Multi-Thread

- Multi Threading의 장점
  - Task 들의 병행 실행을 통한 최적화
  - CPU 활용의 극대화 (특히, 멀티코어)
  - UI 반응성을 높여줌
- Multi Threading은 성능 최적화의 훌륭한 수단이지만, 잘못 쓰면 오히려 독이 됨
- Multi Threading은 왜 어렵나?
  - 불규칙한 Thread 실행 순서 : 구현 & Debbguing이 어려움
  - 가장 큰 이유는 Race Condition !!

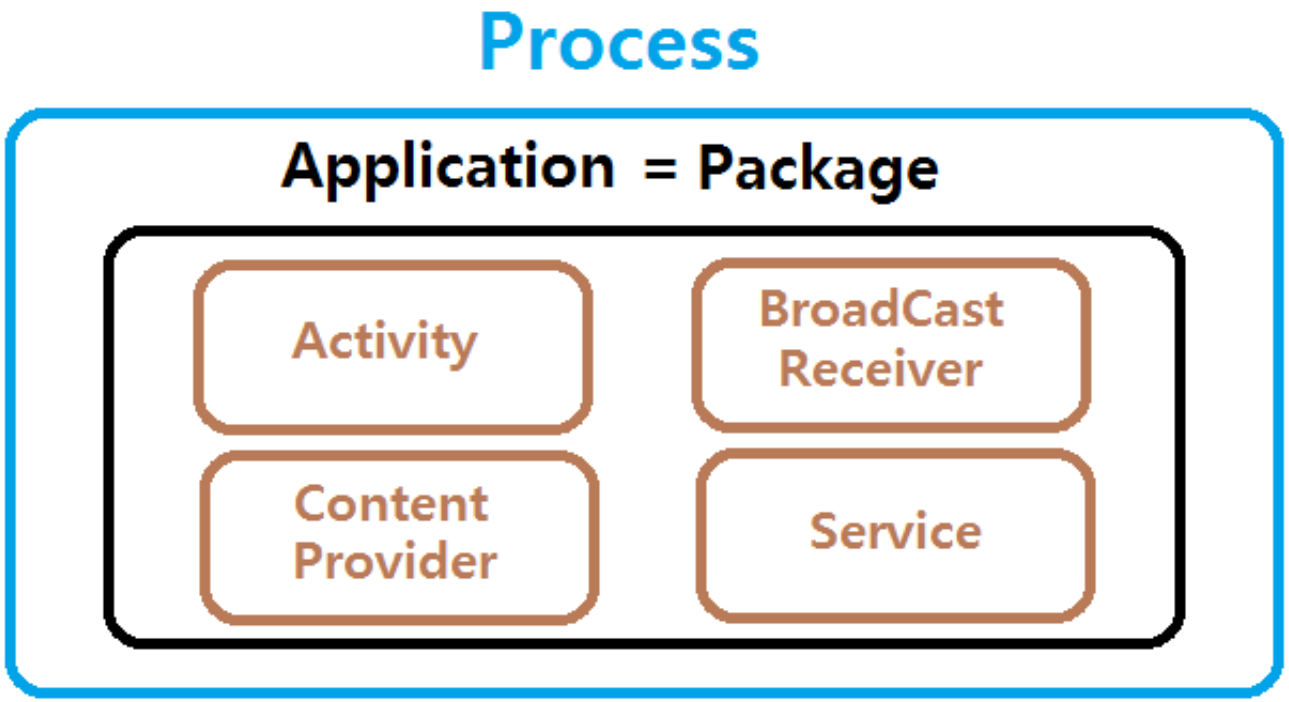
다양한 Thread Programming

- pthread(POSIX Thread)
  - 최신 UNIX Kernel은 대부분 지원
  - Windows native Thread에 비해 성능이 떨어짐

- OpenMP(Open Multi-Processing)
  - Thread 기법 몇 가지를 단순화
  - C/C++, Fortran 등 다양한 언어 지원
  - #pragma omp로 직접 함수를 Coding하지 않아도 됨
  - Preprocessor 수준에서 처리하므로 높은 이식성 확보
  - GCC/MSC 지원

Process

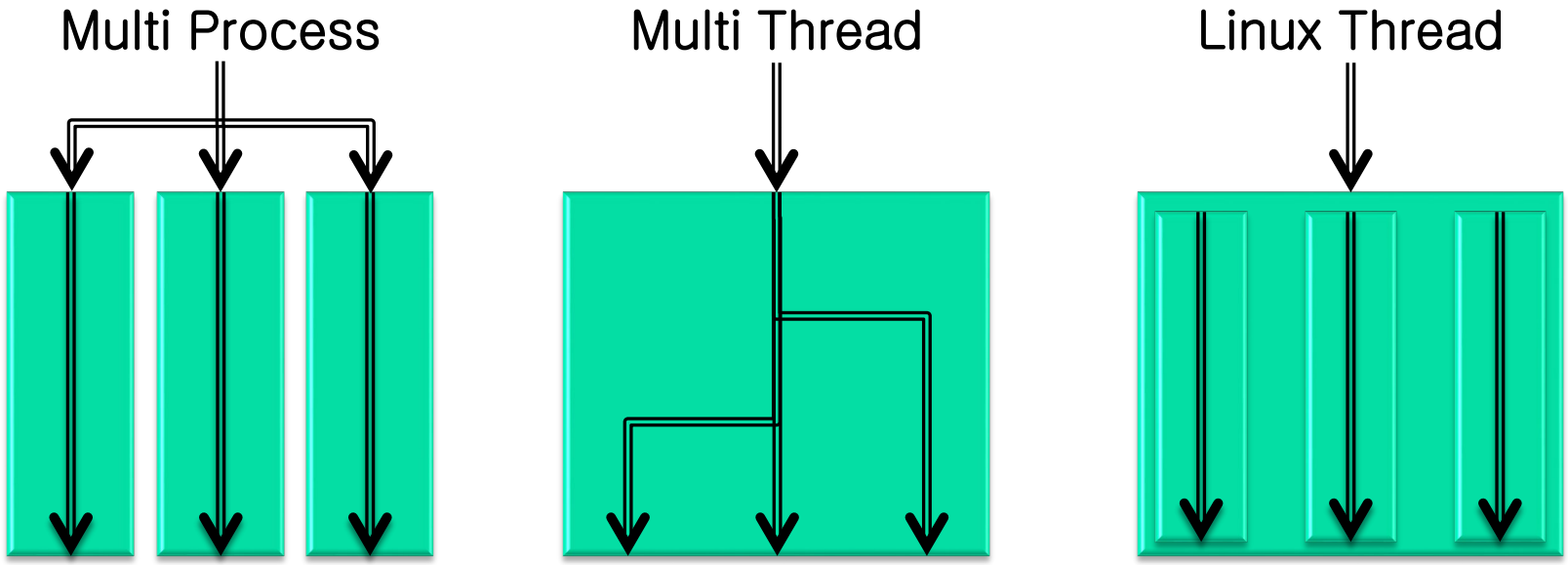
- 일반적으로 App 하나는 하나의 Process로 실행



Thread

- Thread Classification

EX. USER LEVEL



Task or Thread