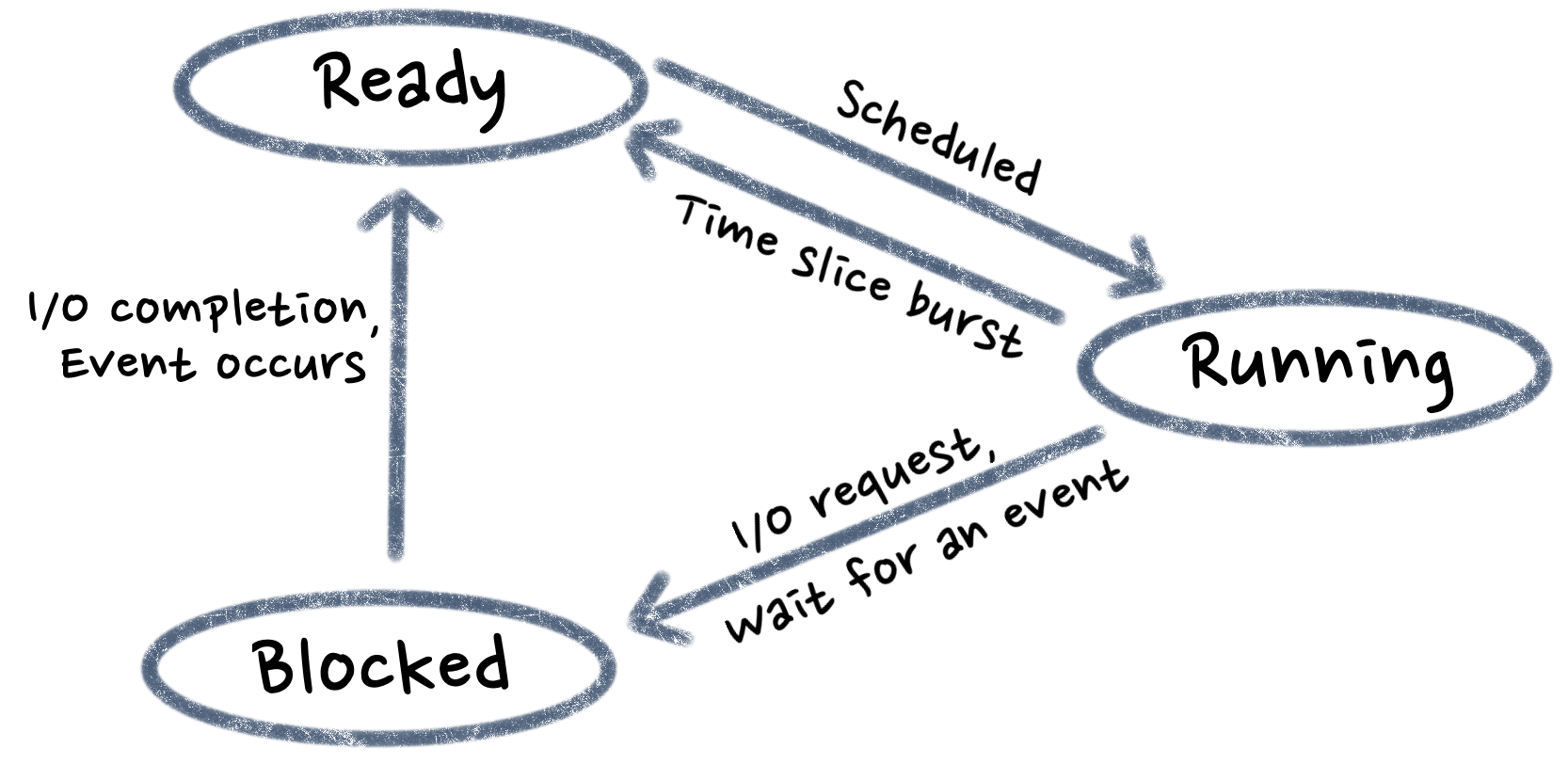
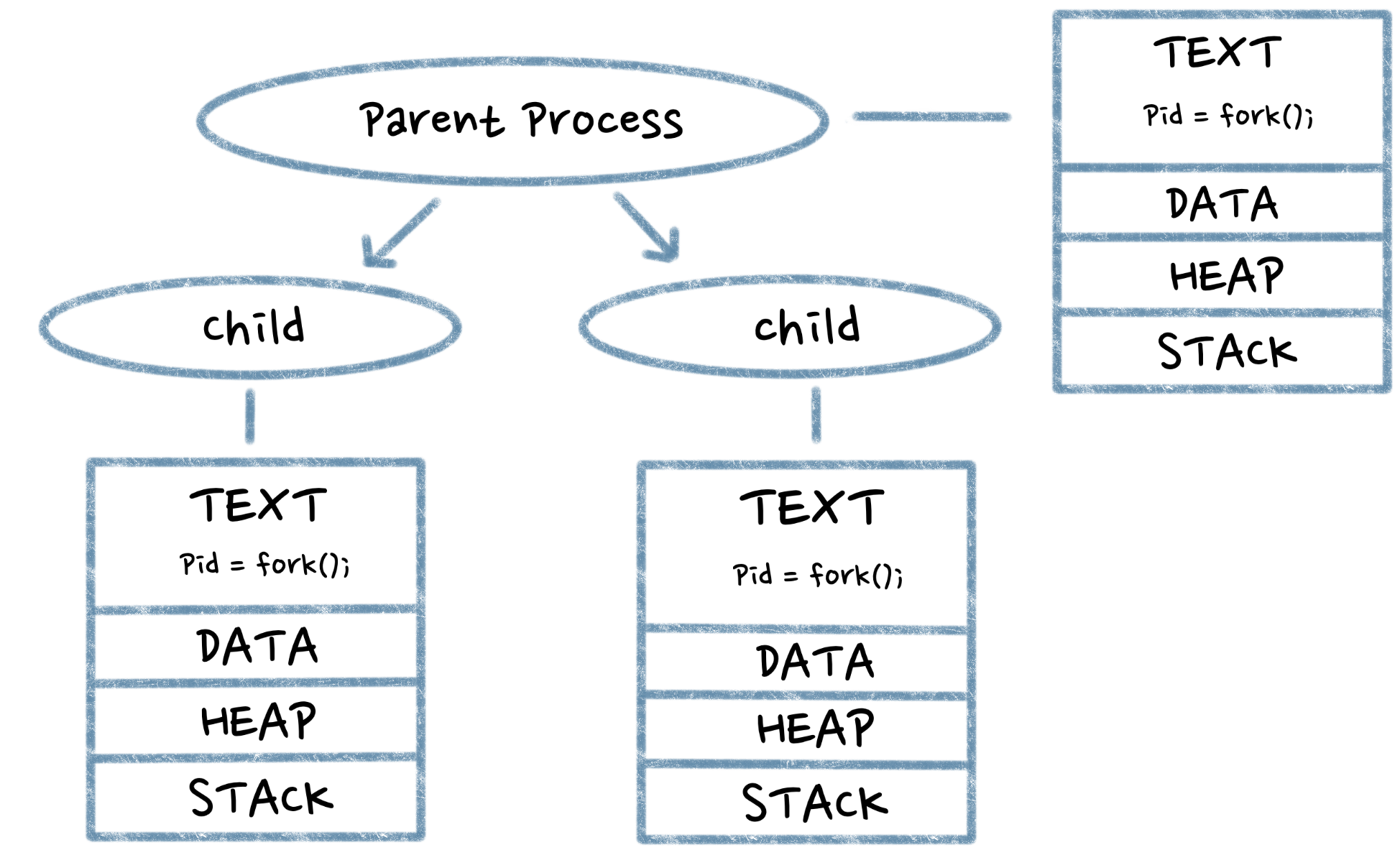
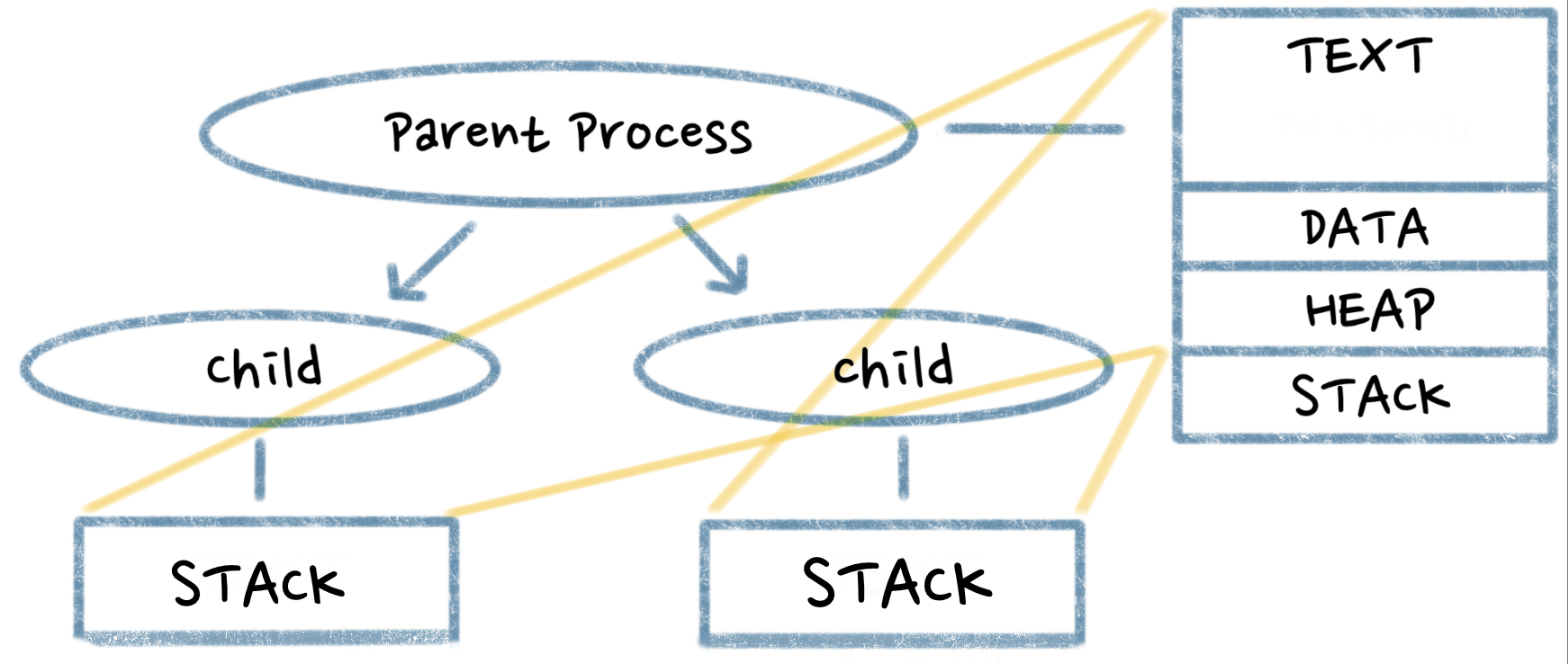
* **프로그램** 이란 ?
  + 어떤 작업을 위해 실행할 수 있는 파일
  + 일반적으로 Disk에 존재하는 상태를 뜻한다.
* **프로세스** 란?
  + Disk에서 메모리로 올라와 실행되고 있는 프로그램의 인스턴스
  + 
  + 운영체제로부터 시스템 자원을 할당받는 작업의 단위.
    - CPU 시간
    - 주소 공간
    - 독립된 메모리 영역
      * Code, Data, Stack, Heap 으로 이루어져있다.
      * 기본적으로 1개의 메인쓰레드를 갖는다.
      * 각 프로세스는 별도의 주소 공간에서 실행되며,   
        프로세스는 다른 프로세스의 변수나 자료구조에 접근할 수 없다.
      * 프로세스가 다른 프로세스에 접근하기 위해서는 통신작업이 필요하다.
* **멀티 프로세스** 란?
  + 하나의 응용프로그램을 여러 개의 프로세스로 구성하여 각 프로세스가 하나의 작업을 처리하도록 하는 것.
  + 프로세스 내부에서 ‘fork()’를 통하여 자식 프로세스를 만들어 낼 수 있다.
  + 이 때, 부모 프로세스의 메모리영역을 통째로 복사하여, 가지고 있는다.
  + 
* **스레드** 란?
  + 독립적이고 병행적인 실행 흐름을 갖는 프로그램 실행 단위
  + Linux환경에서의 스레드는 일반적으로  
    프로세스 안에 존재하며, 프로세스의 자원을 사용한다.
  + 스레드는 부모프로세스가 존재하는 동안 실행되며, 각 스레드는 독립적이다.
  + 프로세스 내부의 스레드는 text 와 data, heap 영역을 공유하며, 별도의 실행 스트림을 가지므로, 별도의 stack을 가진다.
* **멀티스레드** 란?
  + 하나의 응용프로그램을 여러 개의 스레드로 구성하고, 각 스레드에서 하나의 작업을 처리하도록 하는 것.



* **멀티프로세스 vs 멀티스레드**

**→ 멀티프로세스**

* + **[장점]**
    - 여러 개의 자식 프로세스 중 하나에 문제가 발생하면, 메모리 공간이 나뉘어져 있으므로, 해당 자식만 죽는 것으로 다른 영향이 커질 위험이 없다.
  + **[단점]**
    - CPU에서 여러 프로세스를 돌아가면서 작업을 처리하는 Context Switching 작업에 있어서 오버헤드가 발생한다.  
      → 각 프로세스가 모든 메모리영역 (text, data, heap, stack)을 가지고 있기 때문에, CS가 발생할 때마다, 캐쉬데이터를 삭제하고 다시 불러오는 작업에서 오버헤드가 발생할 수 있다.
    - 서로의 메모리영역을 공유할 수 없기 때문에, 같은 data 영역에 대해서 작업하는 경우, 프로세스간 통신을 사용하여야만 공유가 가능하다.

**→ 멀티스레드**

* + **[장점]**
    - stack을 제외한 다른 메모리 영역을 공유하기 때문에, 통신의 부담이 적다.
    - 스레드 사이의 작업량이 작아 CS가 빠르고, 부담이 적다.
    - 스레드를 생성할 때, stack영역 하나만을 할당하면 되므로, 자원을 효율적으로 관리할 수 있다.
  + **[단점]**
    - 공유하는 메모리영역이 많기 때문에, 설계단계가 중요하다.
    - 멀티 스레드일 경우, 자원 공유의 문제가 발생한다.
    - 하나의 스레드에 문제가 생길 경우, 전체 프로세스가 영향을 받는다.
* 멀티 프로세스 < 멀티 스레드 ?

1. 자원의 효율성 증대

* 멀티 프로세스로 실행되는 작업을 멀티 스레드로 실행할 경우, 프로세스를 생성하여 자원을 할당하는 시스템 콜이 줄어들어 자원을 효율적으로 관리할 수 있다.
* 스레드는 프로세스 내의 메모리를 공유하기 때문에 독립적인 프로세스와 달리 스레드 간 데이터를 주고 받는 것이 간단해지고 시스템 자원 소모가 줄어들게 된다.

1. 처리 비용 감소 및 응답 시간 단축

* 프로세스 간의 통신(IPC)보다 스레드 간의 통신의 비용이 적으므로 작업들 간의 통신의 부담이 줄어든다.
* 스레드는 Stack 영역을 제외한 모든 메모리를 공유하기 때문
* 프로세스 간의 전환 속도보다 스레드 간의 전환 속도가 빠르다.
* Context Switching시 스레드는 Stack 영역만 처리하기 때문

1. 스레드의 경우, 공유되는 메모리 영역이 있으므로, context switching 시에,  
   그 부분을 주의해서 다루어주어야 한다.  
   → 공유자원에 대한 문제만 해결한다면, 멀티스레드가 비용과 시간적인 면에서 훨씬 효율적이다.