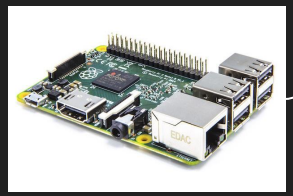
Tizen Main-Loop 이해

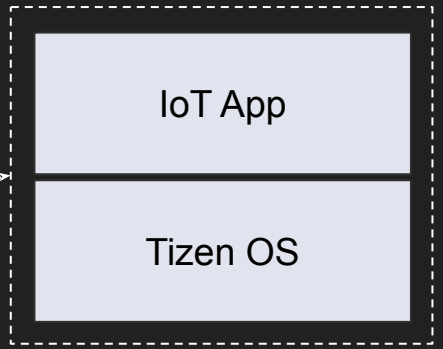
for IoT 앱 개발

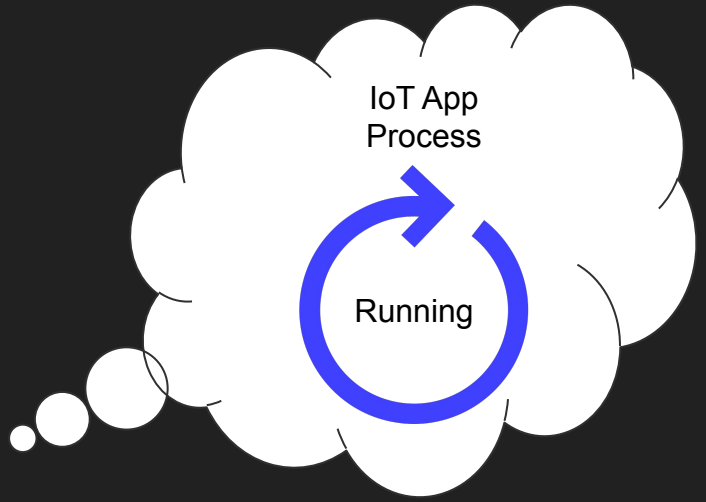
박춘언@서울하드웨어해커톤

2019.08.24

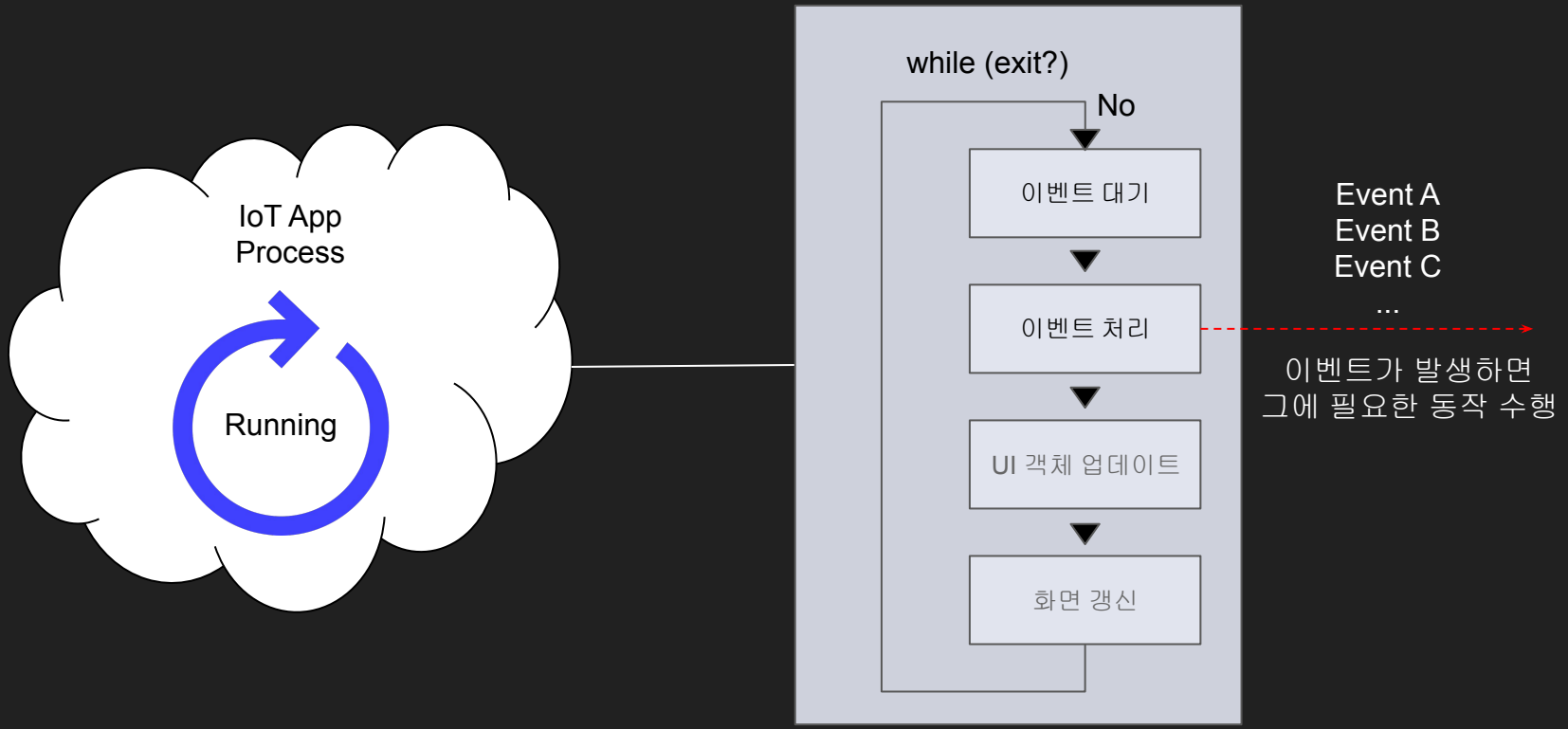
메인루프 Main-Loop,







메인루프 Main-Loop,

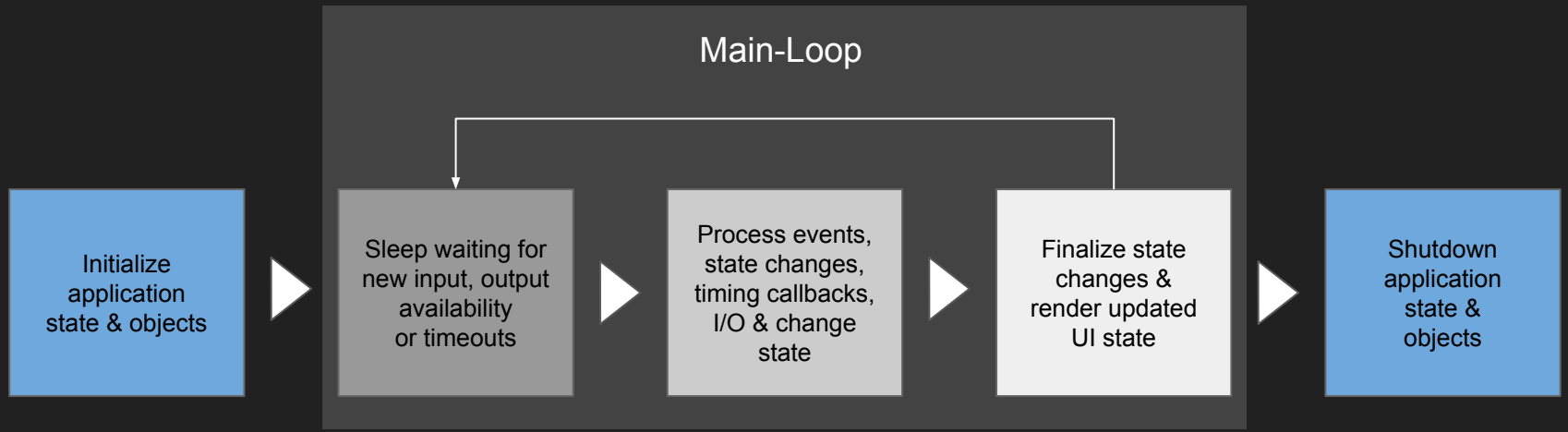


메인루프 Main-Loop,

앱 라이프사이클을 비롯 시스템과 유기적으로 동작할 수 있는 환경(Backbone) 구축

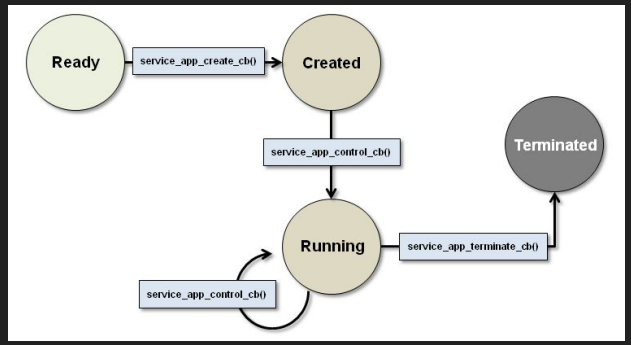
큰 맥락에서, [대기 : 이벤트 처리] 의 반복

이벤트 드리븐(Event-Driven) 방식으로 사용자(앱)가 원하는 동작 추가



Tizen App Life-Cycle 과 Main-Loop

```c
int main(int argc, char* argv[])
{
    appdata s ad = {0,};
    service_app_lifecycle_callback_s event_callback= {0,};

    event callback.create = service app create;
    event callback.terminate = service app terminate;
    event_callback.app_control = service_app_control;

    return service_app_main(argc, argv, &event_callback, &ad);

}


bool service_app_create(void *data)
{
    //TODO:

    return true;
}
```



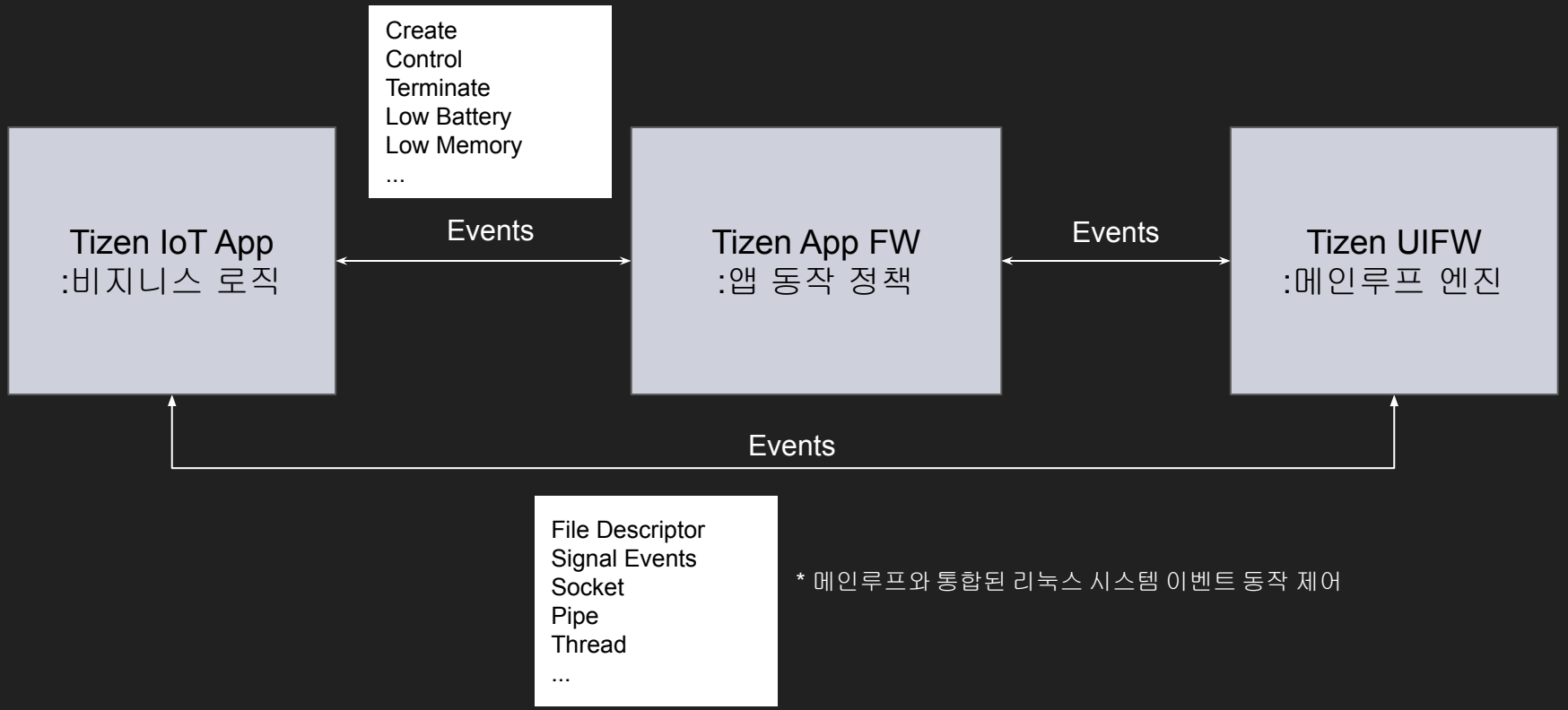
Tizen App Life-Cycle 과 Main-Loop

* 라이프사이클 및 시스템 이벤트 처리

Create
Control
Terminate
Low Battery
Low Memory
...

Tizen App Life-Cycle 과 Main-Loop

* 라이프사이클 및 Tizen 시스템 이벤트 처리



* 메인루프와 통합된 리눅스 시스템 이벤트 동작 제어

Ecore (EFL Core)

Tizen UIFW => EFL(Enlightenment Foundation Libraries)

EFL은 Canvas, Core, UI 기능 모듈로 구성

이 중 Core 모듈이 Main-Loop 기능 담당

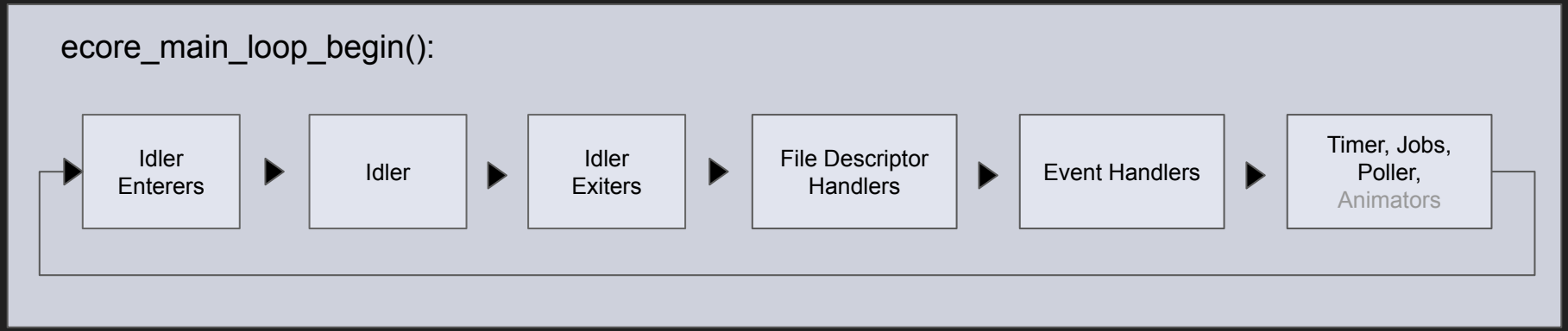
Tizen UIFW
:메인루프 엔진

=

Ecore

www.developer.tizen.org/dev-guide/tizen-iot-headed/5.0/group__Ecore__Main__Loop__Group.html

Ecore Main Loop



https://www.enlightenment.org/develop/efl/mainloop

Ecore Functions

Ecore_Idler: 대기 상태 또는 대기 상태 진입, 대기 종료 시점에 이벤트 호출

- 메인 루프가 수행할 다른 작업(이벤트)이 없을 경우에만 수행
- 대기 상태에 진입해야 하므로 매우 짧은 작업만 수행할 것**!**

Ecore_FD_Handler: 파일 디스크립터(File, Socket, Device, Pipe 등)를 통해 스트리밍 데이터를 모니터링, 읽고 쓰는 작업 수행

- Ecore_Con, Ecore_File, Ecore_Pipe ...

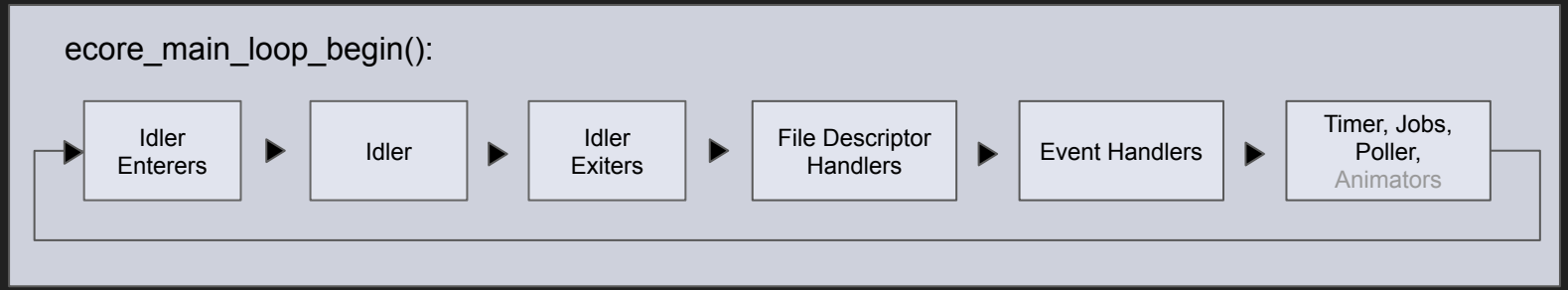
Ecore_Event: 정의된 시그널 이벤트 핸들링을 수행하거나 사용자 정의 이벤트 추가 / 호출 수행

Ecore_Timer: 시간 주기로 이벤트 발생 (ms)

Ecore_Poller: Tick(default = 0.125ms) 단위로 이벤트 발생. Tick 주기 시간 설정 가능

Ecore_Jobs: 지연(Lazy) 작업 처리 수행

- 특히 동일 루프 내에서 동일 작업을 요구하는 이벤트가 다수 발생 시, 이를 Job으로 전환 (중복 작업 회피)

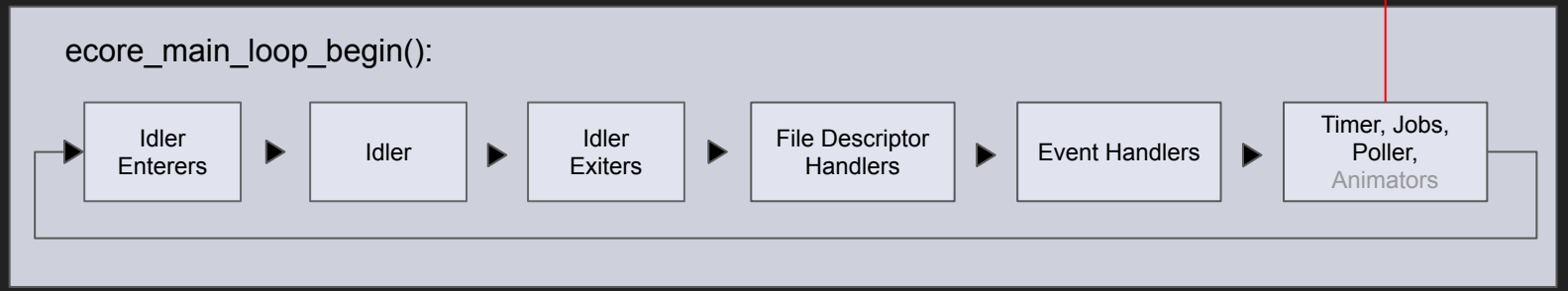


Ecore Timer

```
Ecore_Timer *timer = NULL;

/* 약 2초 주기로 어떤 작업을 시작해야 한다고 가정… */
void do()
{
   /* 타이머 추가. ecore_timer_del(timer)로 명시적 제거 가능. */
   timer = ecore_timer_add(2, timer_cb, user_data);
   ...
}

/* 약 2초 주기로 호출 */
Eina_Bool timer_cb(void *data)
{
   ...

   //이벤트 연장
   return ECORE_CALLBACK_RENEW;

   //이벤트 종료(취소)
   timer = NULL;                    //Dangling Handle 주의!
   return ECORE_CALLBACK_CANCEL;
}
```

Ecore Timer

```
Eina_Bool timer_cb(void *data)
{
   ...

   //이벤트 연장
   return ECORE_CALLBACK_RENEW;

   //이벤트 종료(취소)
   timer = NULL;
   return ECORE_CALLBACK_CANCEL;
}
```

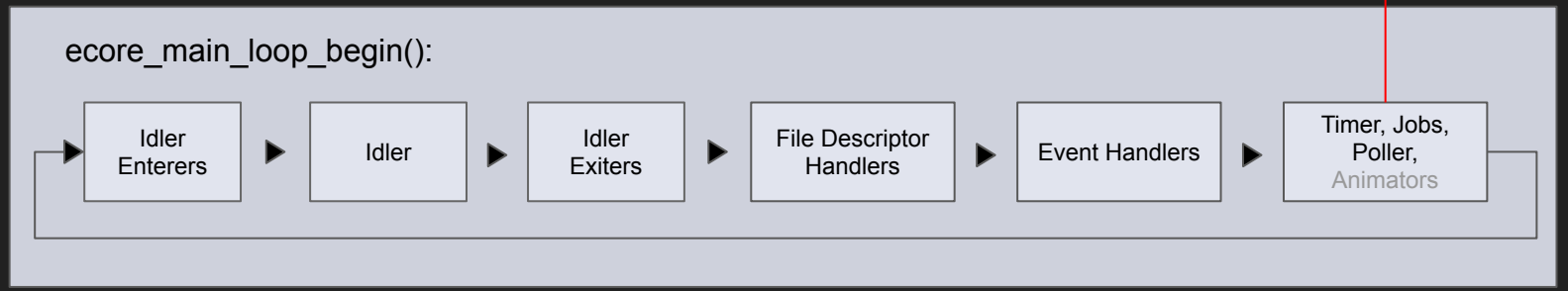


Ecore Job

```
Ecore_Job *job = NULL;

/* 어떤 작업을 추가 한다고 가정… */
void request()
{
   /* 타이머 추가. ecore_job_del(job)로 명시적 취소 가능. */
   if (!job) job = ecore_job_add(job_cb, user_data);
   ...
}

/* 메인루프 마지막에 작업 처리 */
void job_cb(void *data)
{
   ...
   job = NULL;           //Dangling Handle 주의!
}
```

Ecore Job

```
/* 메인루프 마지막에 작업 처리 */
void job_cb(void *data)
{
   ...
   job = NULL;
}
```



Too Heavy!

```
/* 약 2초 주기로 호출 */
Eina_Bool timer_cb(void *data)
{
   ...

   for (i = 0; i < 999999999; ++i)
   {
      ...
   }

   ...
}
```

```
/* 메인루프 마지막에 작업 처리 */
void job_cb(void *data)
{
   ...

   while (true)
   {
      ...
   }

   ...
}
```

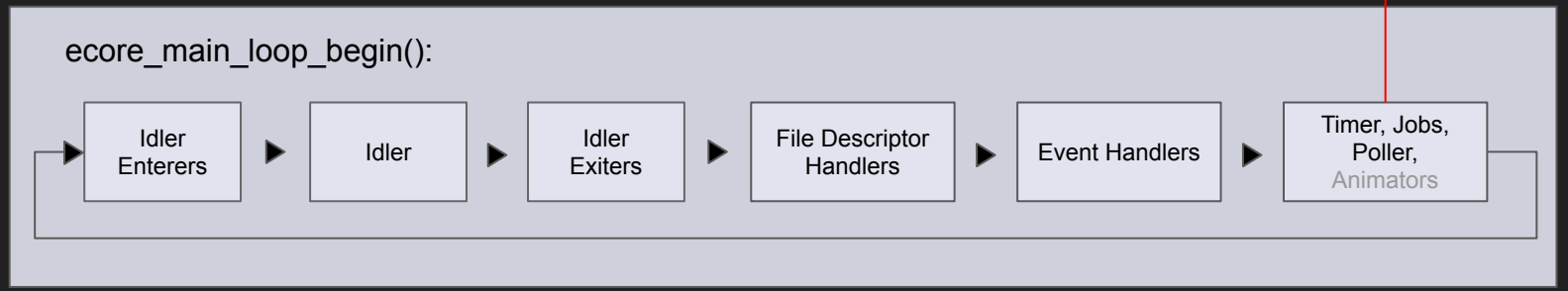
Too Heavy!

각 단계에서 발생한 이벤트 처리 작업에서

메인루프가 Block 되지 않도록 주의!

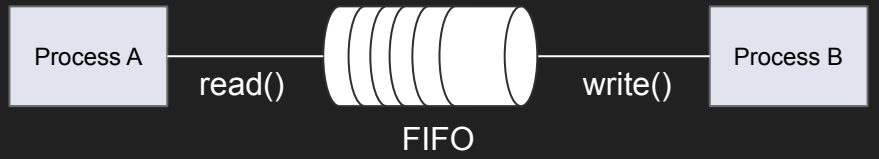
```c
/* 메인루프 마지막에 작업 처리 */
void job_cb(void *data)
{
   while (true)
   {
      /* You died.
         당신은 돌아올 수 없는
         블랙홀에 빠졌습니다. */
   }
}
```

복귀 불가

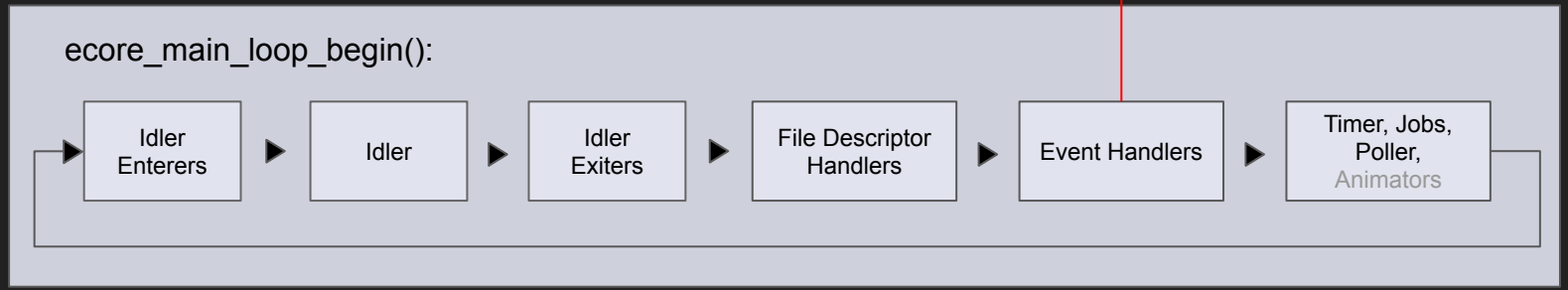


모니터링 예제...

```c
/* 이벤트 핸들링 */
Eina_Bool event_cb(void *data)
{

   /* Process 간 데이터를 주고받을 파이프 연결.
      만약, 전달받을 데이터가 비여있다면… */
   int fd = open("/tmp/xxxx", O_RDWR | ND_DELAY);

   /* 안돼! 당신은 블랙홀로 다시 소환됐습니다. */
   read(fd, buffer, buffer_size);
   ...

   return ECORE_CALLBACK_DONE;
}
```



Ecore Event 호출 후 대기



비동기 접근

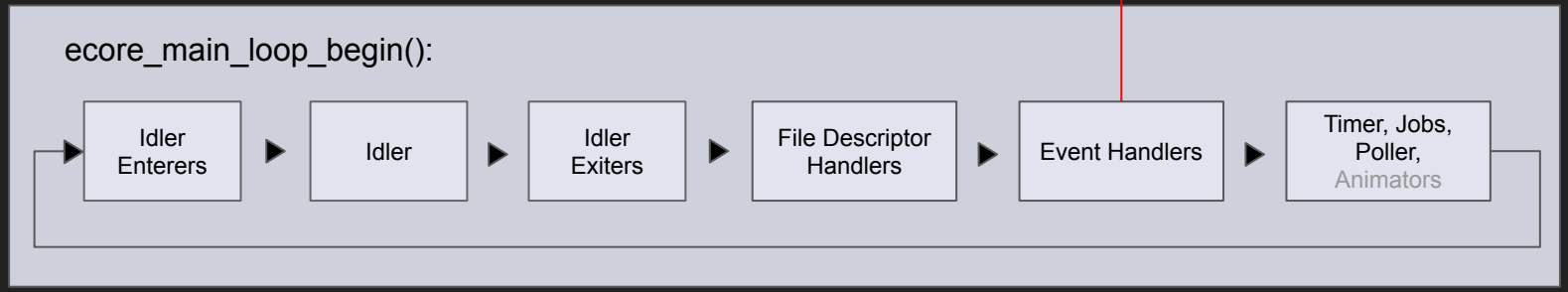
```c
/* 이벤트 핸들링 */
Eina_Bool event_cb(void *data)
{
   /* 쓰레드 생성
   pthead_t th = pthread_create(thread_worker, ...);

   ...
}
```

쓰레드 생성

```c
/* 쓰레드 함수 */
void* worker_thread(void *data)
{

   /* Process 간 데이터를 주고받을 파이프 구현.
      만약, 전달받을 데이터가 없다면… */
   int fd = open("/tmp/xxxx", O_RDWR | ND_DELAY);
   read(fd, buffer, count);
   ...

   /* 처리한 결과물을 메인 쓰레드로 전달하긴 해야하는데?
*/
}
```

Thread 2

Ecore Event 호출 후 정상 복귀



* Thread 1 (메인 쓰레드)

비동기 접근

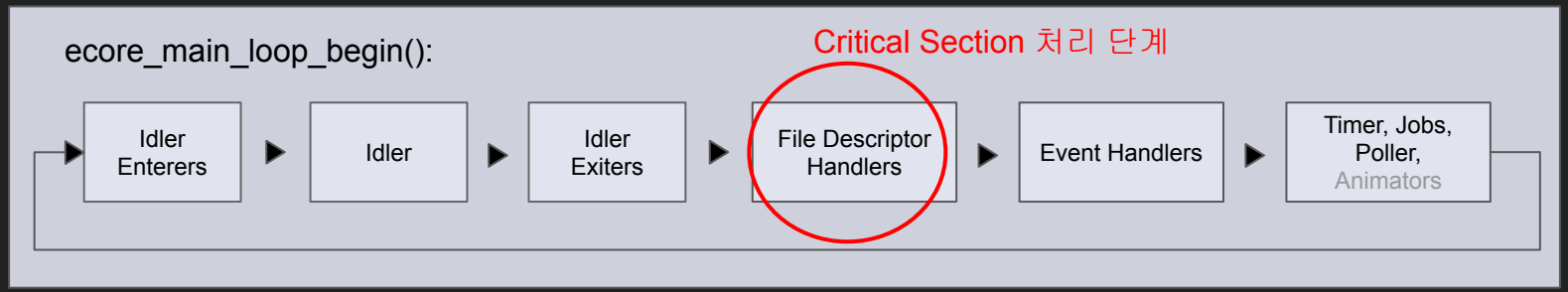
Ecore_Thread, ecore_pipe_write() 등 몇 API를 제외한 API는 Thread Safe 하지 않음.

사용자는 메인루프와 앱 스레드 간의 동기화/상호배제를 고려, 애플리케이션을 디자인해야 함.

Ecore에서 제공하는 편의 Critical Section 도구

- **ecore_thread_main_loop_begin(), ecore_thread_main_loop_end()**
- **ecore_main_loop_thread_safe_call_sync(), ecore_main_loop_thread_safe_call_async()**
- 동기화를 수행하는 동안 **Main Loop** 에 일시적(매우 잠깐) **Block**이 발생할 수 있음
- 최소의 호출로 과부하 방지할 것

Thread-Pool 기반, 보다 고수준의 쓰레드로 **Ecore_Thread** 기능 제공.



비동기 접근

```
/* 쓰레드 함수 */
void* worker_thread(void *data)
{
   /* Process간 데이터를 주고받을 파이프 구현.
      만약, 전달받을 데이터가 없다면… */
   int fd = open("/tmp/xxxx", O_RDWR | ND_DELAY);
   read(fd, buffer, count);
   ...


   /* Critical Section! */
   ecore_thread_main_loop_begin();

   //begin - end 구간에서 메인쓰레드와 공유자원 접근.

   ecore_thread_main_loop_end();
}
```

Ecore_FD_Handler

Socket, Pipe 및 기타 Stream 모니터링: read(), write(), select() 동작을 메인루프 Block 없이 수행

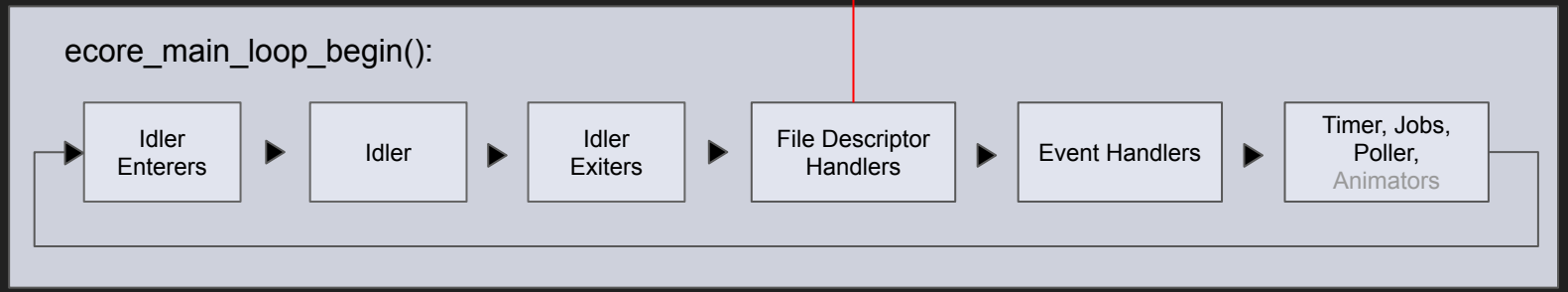
메인루프는 등록된 FD Handler의 파일디스크립터가 active 한 경우에만 등록한 콜백 함수를 호출해 줌

Ecore_IPC, Ecore_Con 등의 Network 전용 인터페이스도 존재하므로 참고

```
Ecore_FD_Handler * handler = NULL;

/* 이벤트 핸들링 */
Eina_Bool event_cb(void *data)
{
   /* Process 간 데이터를 주고받을 파이프 구현.
   int fd = open("/tmp/xxxx", O_RDWR | ND_DELAY);
   handler = ecore_main_fd_handler_add(fd, ECORE_FD_READ | ECORE_FD_ERROR,
                                       fd_handler_cb, user_data, NULL, NULL);
   ...
}

Eina_Bool fd_handler_cb(void *data, Ecore_FD_Handler *handler)
{
   if (ecore_main_fd_handler_active_get(handler, ECORE_FD_ERROR))
      //Error 처리 …
      return ECORE_CALLBACK_CANCEL;

   int fd = ecore_main_fd_handler_fd_get(handler);
   read(fd, buffer, count);

   ...
}
```

Ecore_FD_Handler

```
Eina_Bool  fd_handler_cb (void *data, Ecore_FD_Handler  *handler)
{
   if ecore_main_fd_handler_active_get(handler, ECORE_FD_ERROR))
      //Error 처리 …
      return ECORE_CALLBACK_CANCEL;

   int fd = ecore_main_fd_handler_fd_get(handler);
   read(fd, buffer, count);

   ...
}
```



hermetpark@gmail.com