# Zircon 内核对象与系统调用: Interrupt 与 Port

~~主要就是翻译整理 API 文档~~

2019 年 11 月 22 日

# 进度

# 动机

无处不在的事件...

- 网络包来到
- 玩家按下 Ctrl-C
- IPC 通信
- Timer 到时间
- ...

事件?

# 动机

无处不在的事件...

- 网络包来到
- 玩家按下 Ctrl-C
- IPC 通信
- Timer 到时间
- ...

事件？系统某部分状态满足了某个条件

- 某部分 $\rightarrow$ 对应某个内核对象
- 状态改变 $\rightarrow$ 内核对象发射信号

# 信号处理

信号如何发生?

- 内核管理：ZX_FIFO_READABLE、ZX_THREAD_TERMINATED、ZX_TIMER_SIGNALED ...
- 某个应用手动产生：zx_object_signal

# 信号处理

如何等待/观察信号?

# 信号处理

如何等待/观察信号? `zx_object_wait_XXX`

- `zx_object_wait_one(handle, waitset, deadline, observedset)`: 阻塞等待**一个**内核对象
- `zx_object_wait_many(wait_items, num_items, deadline)`: 阻塞等待**至多八个**内核对象
- 更多内核对象?

# 信号处理

如何等待/观察信号? `zx_object_wait_XXX`

- `zx_object_wait_one(handle, waitset, deadline, observedset)`: 阻塞等待**一个**内核对象
- `zx_object_wait_many(wait_items, num_items, deadline)`: 阻塞等待**至多八个**内核对象
- 更多内核对象?
- `zx_object_wait_async + zx_port_wait`: 阻塞等待多个对象
- ...其实不只这些

# Port：概念

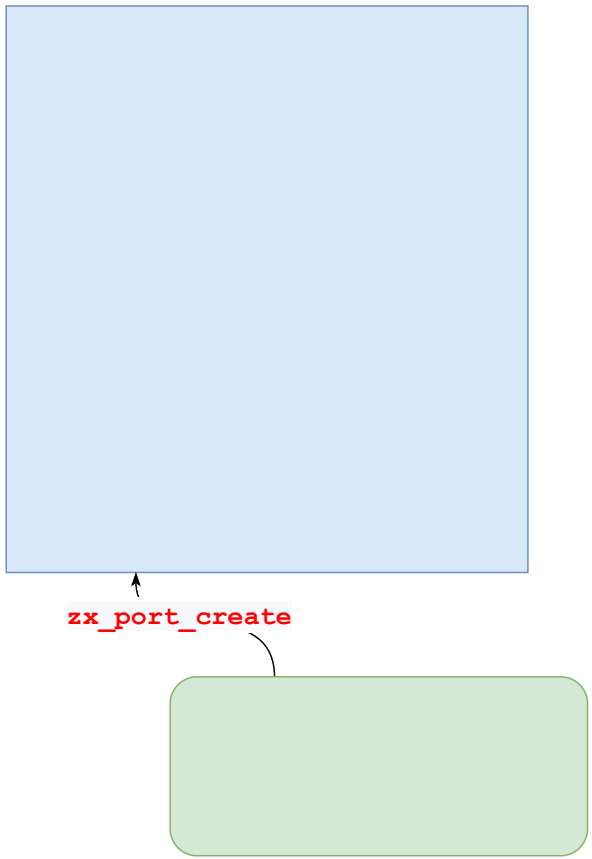
- Port 内部可以看成有一个 packet 的 queue
- 有很多源可以给 Port 入队
- Port 也可以被观察，即出队
- 队列元素是 32 字节的 packet，可以代表 signal, interrupt, 以及用户自定义的 packet

# Port 的一生

最初，世界上干干净净，一个 Port 也没有。

# Port 的一生

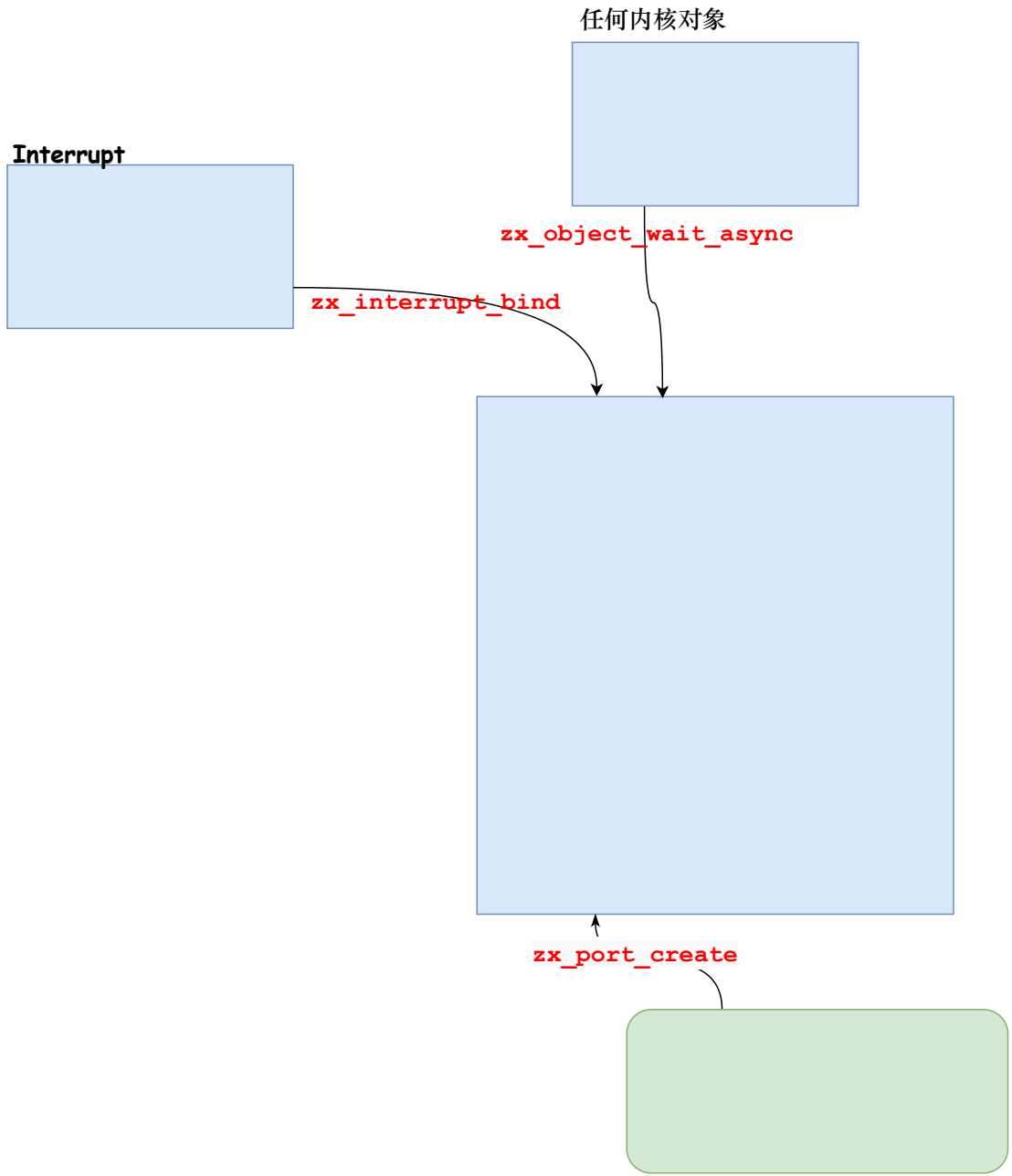
作为主人公的进程想要一个 Port，于是它调用了 `zx_port_create`


zx_port_create

# Port 的一生

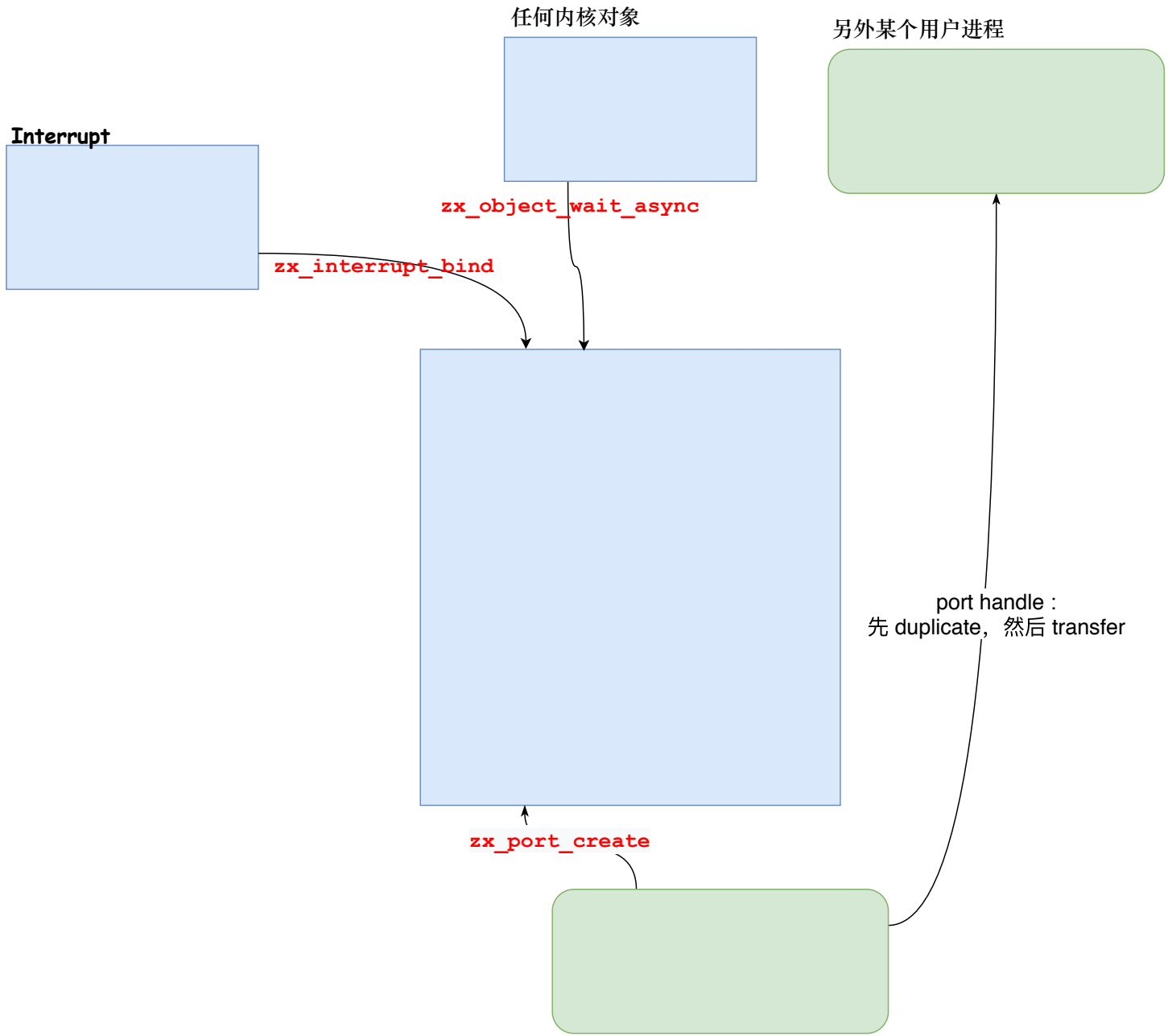
中断和信号可以绑定到 Port 上。

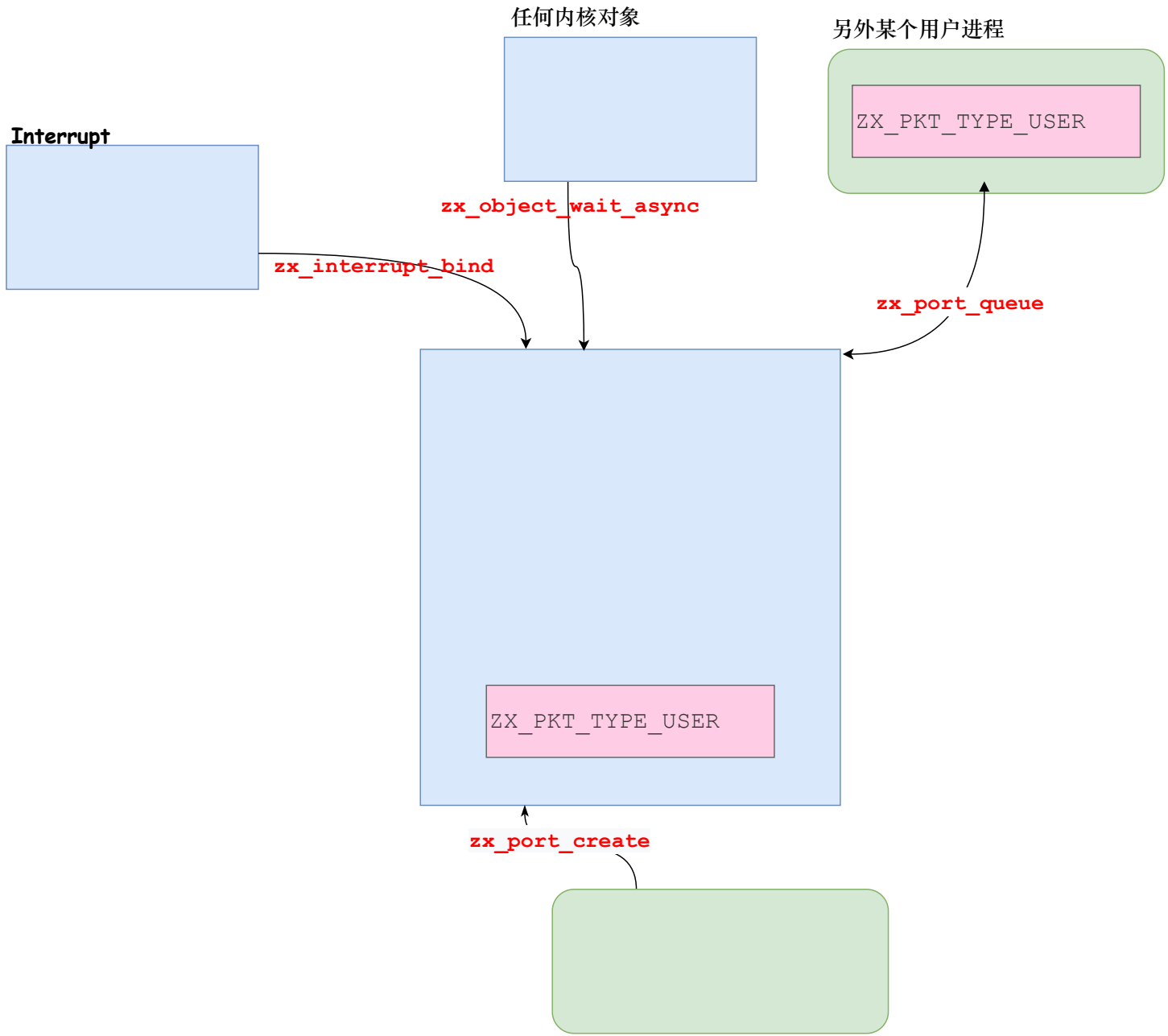
- 信号发生时，不再是 `object_wait`，而是直接入队一个 packet
- 一个内核对象只能绑定到一个 Port


任何内核对象
Interrupt
zx_object_wait_async
zx_interrupt_bind
zx_port_create

# Port 的一生

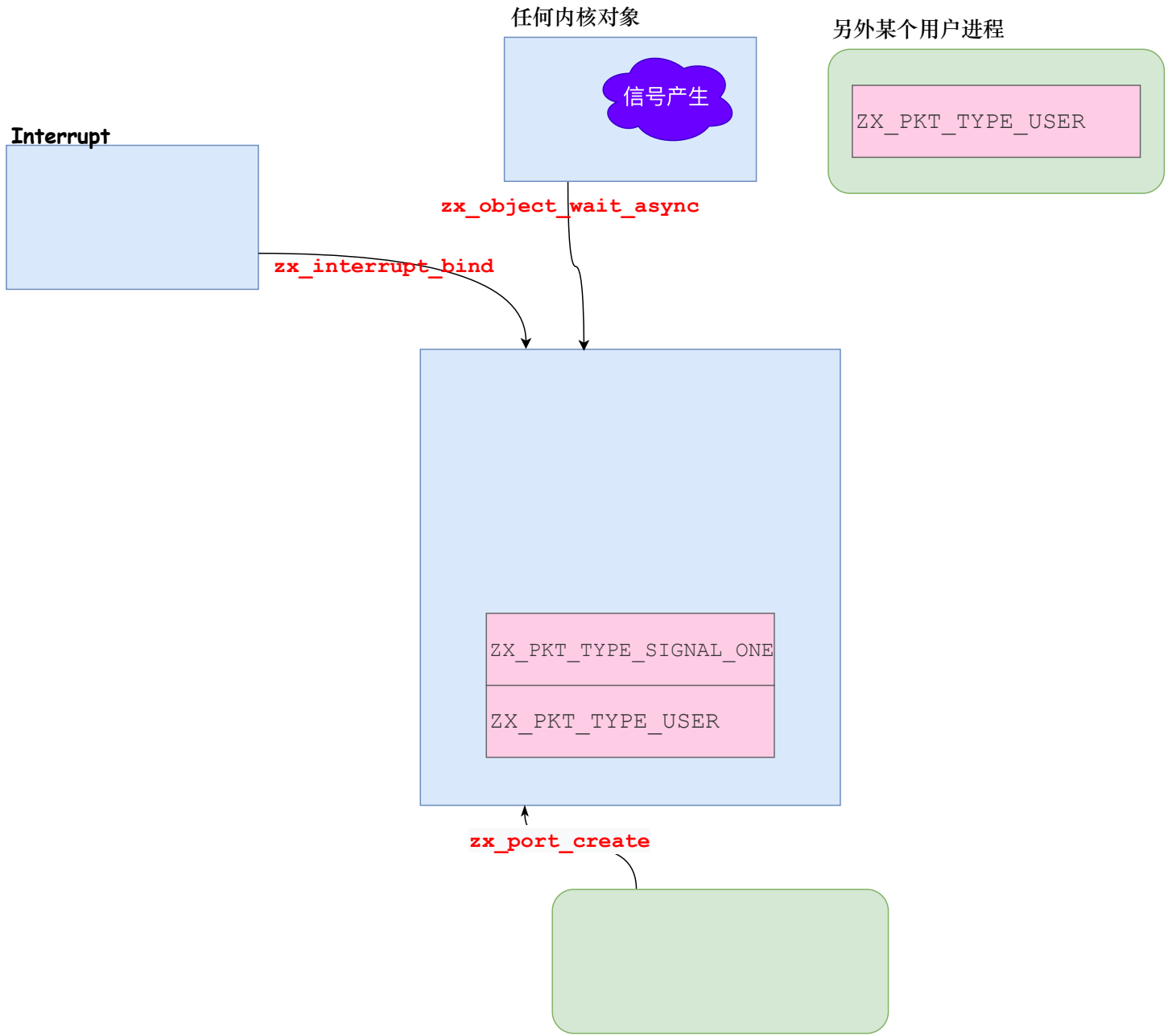
甚至 Port 可以完成用户进程间传输 packet
可以手动入队。


任何内核对象
另外某个用户进程
Interrupt
zx_object_wait_async
zx_interrupt_bind
port handle :
先 duplicate，然后 transfer
zx_port_create


任何内核对象
另外某个用户进程
ZX_PKT_TYPE_USER
Interrupt
zx_object_wait_async
zx_interrupt_bind
zx_port_queue
ZX_PKT_TYPE_USER
zx_port_create

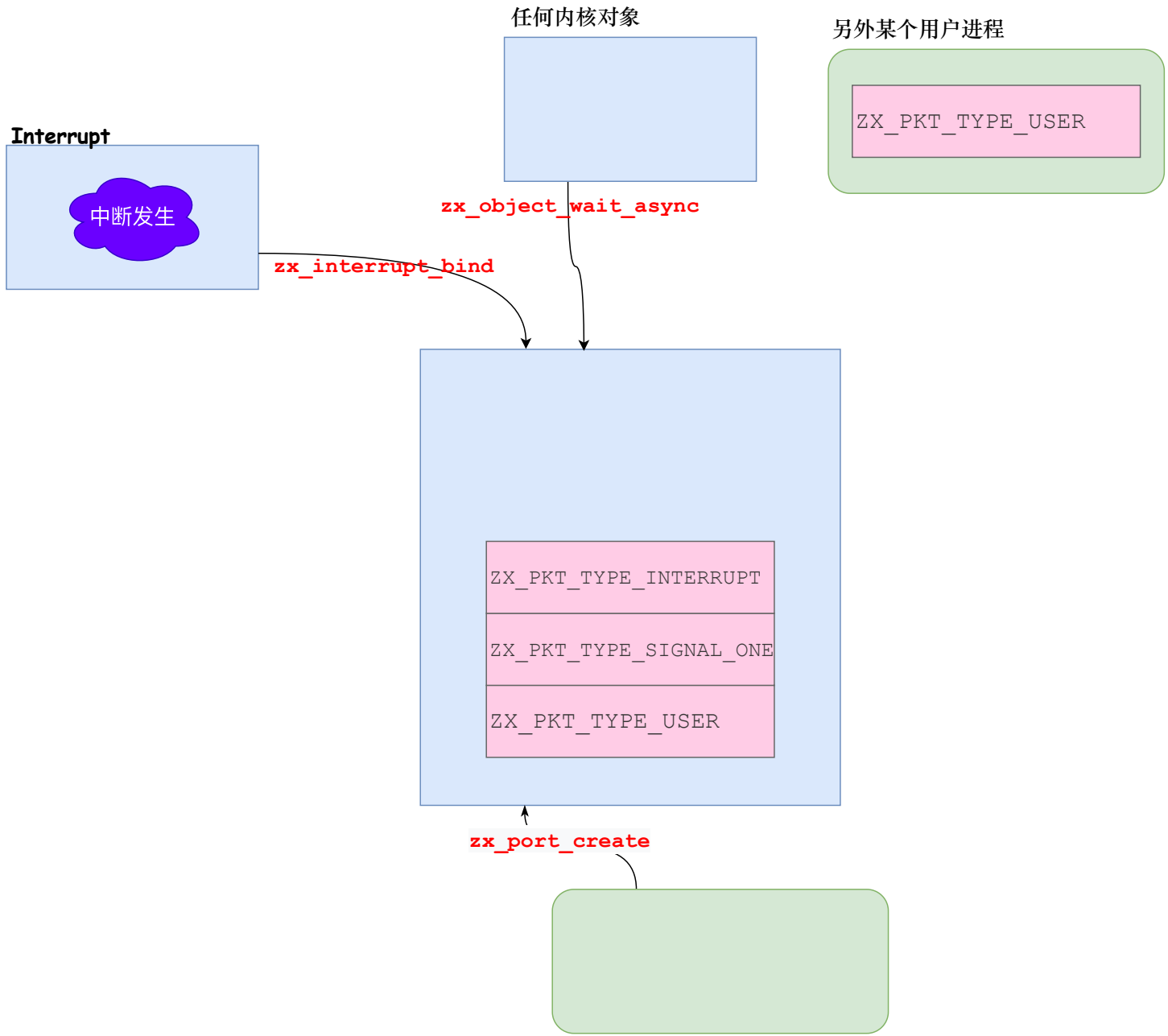
# Port 的一生

某个内核对象上，一个信号发生了。
这个内核对象绑定到了一个 Port。
于是又入队了一个 packet。


任何内核对象
信号产生
另外某个用户进程
ZX_PKT_TYPE_USER
Interrupt
zx_object_wait_async
zx_interrupt_bind
ZX_PKT_TYPE_SIGNAL_ONE
ZX_PKT_TYPE_USER
zx_port_create

# Port 的一生

某个中断发生了。
这个**中断对应的内核对象**绑定到了一个 Port。
于是又入队了一个 packet。

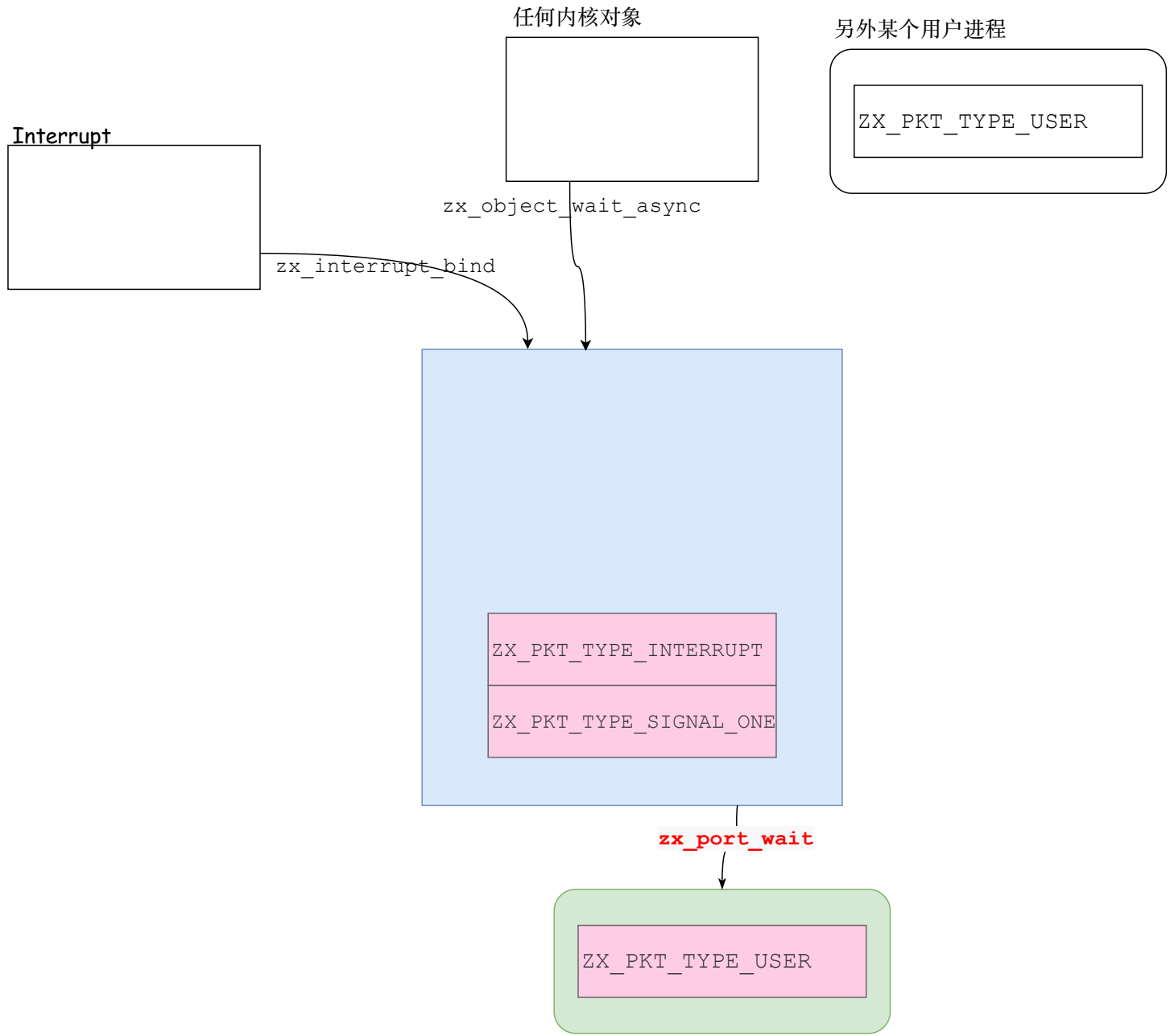

任何内核对象
另外某个用户进程
ZX_PKT_TYPE_USER
Interrupt
中断发生
zx_object_wait_async
zx_interrupt_bind
ZX_PKT_TYPE_INTERRUPT
ZX_PKT_TYPE_SIGNAL_ONE
ZX_PKT_TYPE_USER
zx_port_create

# Port 的一生

主人公进程终于想起来有个 Port 了，它决定检查 Port 里有没有 packet 到来。

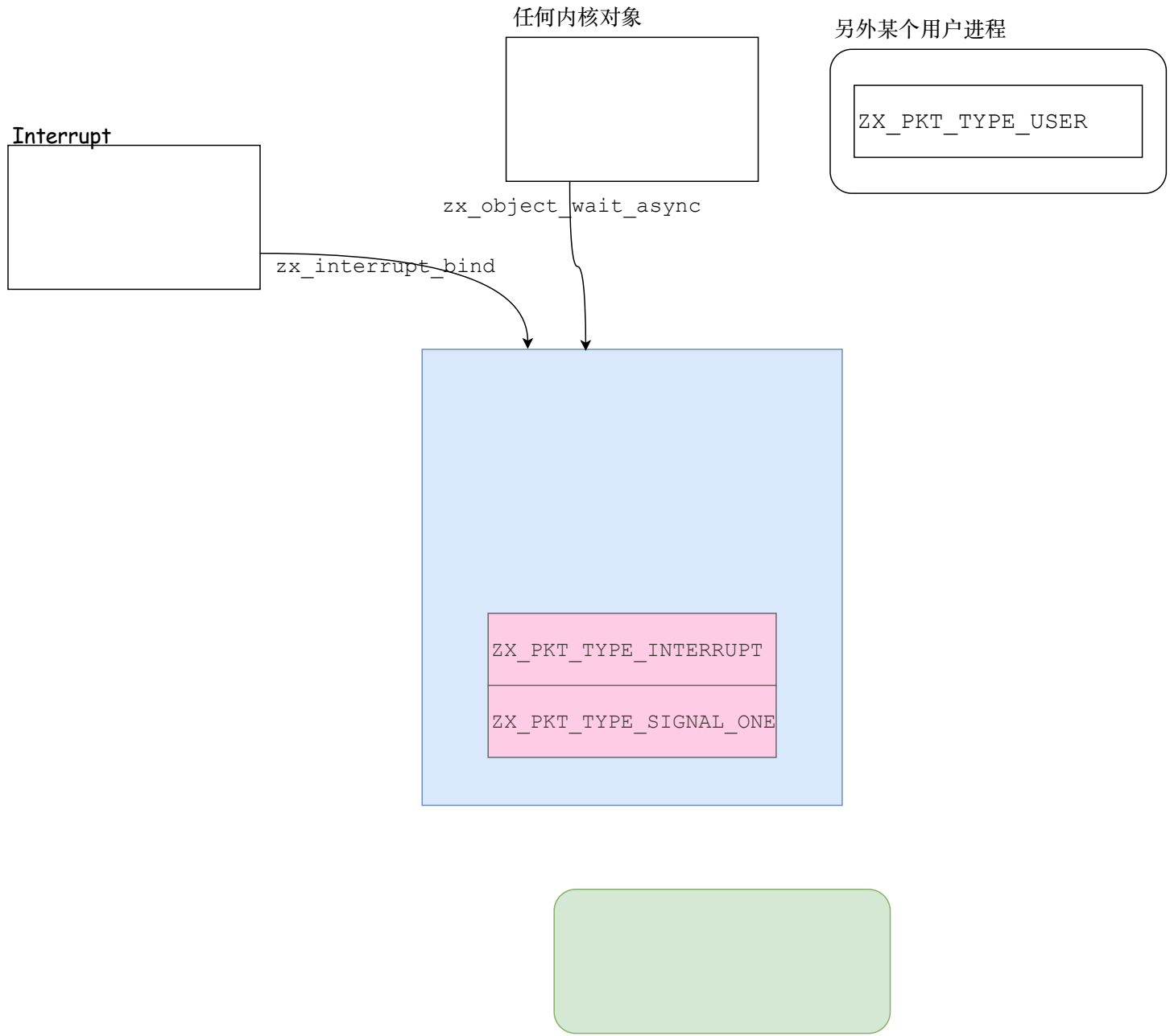
# Port 的一生

于是，他通过 `port_wait` 得到了**一个** packet.
（中断，信号，以及发送 packet 的进程都不再重要了）


任何内核对象
另外某个用户进程
ZX_PKT_TYPE_USER
Interrupt
zx_object_wait_async
zx_interrupt_bind
ZX_PKT_TYPE_INTERRUPT
ZX_PKT_TYPE_SIGNAL_ONE
zx_port_wait
ZX_PKT_TYPE_USER

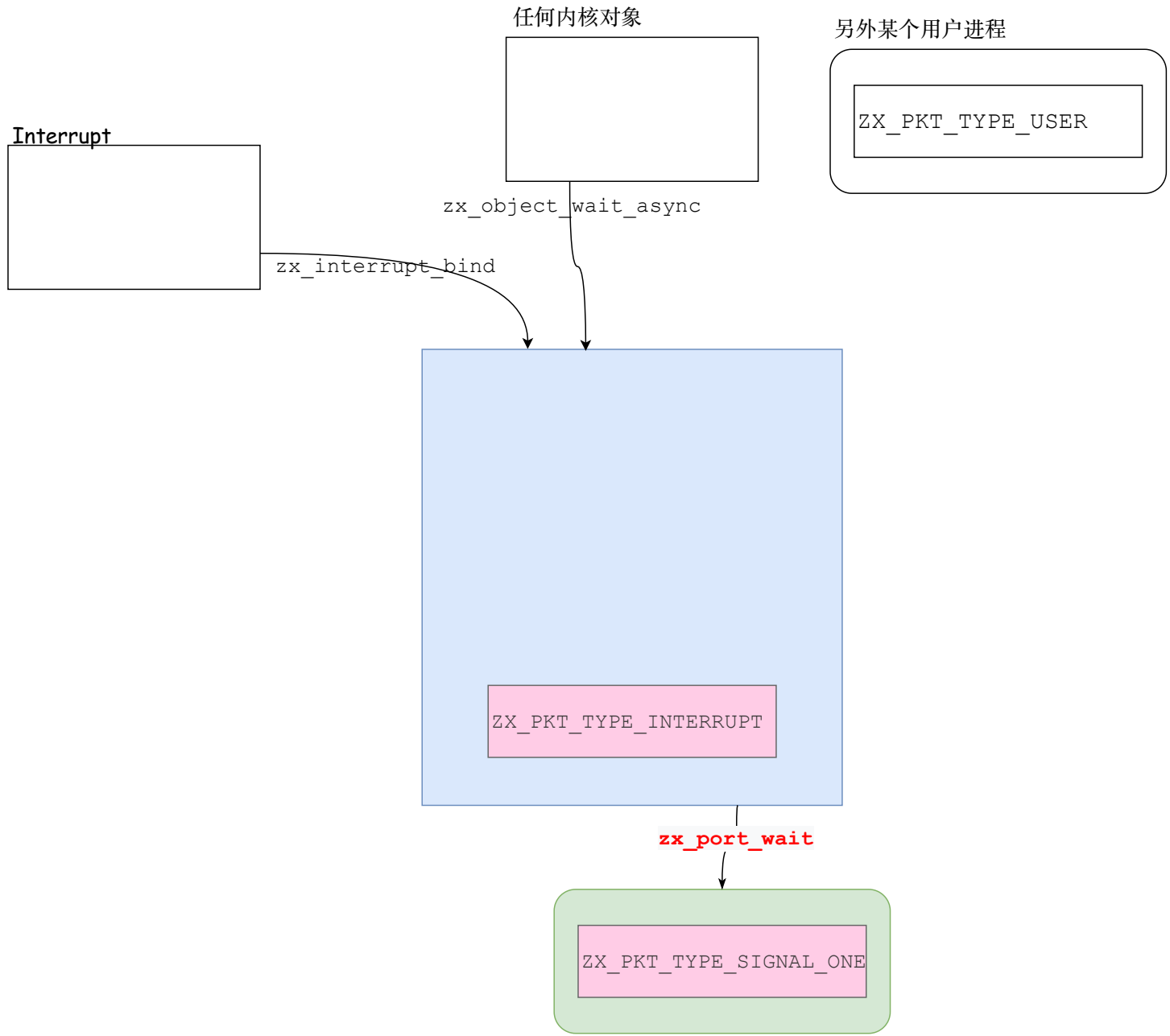
# Port 的一生

过了一会 ...


任何内核对象
另外某个用户进程
ZX_PKT_TYPE_USER
Interrupt
zx_object_wait_async
zx_interrupt_bind
ZX_PKT_TYPE_INTERRUPT
ZX_PKT_TYPE_SIGNAL_ONE

# Port 的一生

主人公进程又通过 `port_wait` 得到了**一个** packet.


任何内核对象
另外某个用户进程
ZX_PKT_TYPE_USER
Interrupt
zx_object_wait_async
zx_interrupt_bind
ZX_PKT_TYPE_INTERRUPT
zx_port_wait
ZX_PKT_TYPE_SIGNAL_ONE

# 相关系统调用：基础

**`port_create`**

`(options, handle*)`

- 创建一个 port
- port 能接受各种各样的东西，除了 signal 外还有 interrupt 等等
- **前面对于中断的处理不准确！**只是举个例子

**`port_queue`**

`(handle, packet)`

- 手动给 port 发送一个 packet

**`port_wait`**

`(handle, deadline, packet*)`

- 阻塞
- 取得 port 队首 packet
- 如果队首为空，最多阻塞 `deadline` 时间

# 相关系统调用：绑定中断和 signal 内核对象

**`object_wait_async`**

`(obj_handle, port_handle, key, signals, options)`

- 指示：如果 `obj` 发生 `signals` 中的某信号，那么给 port 入队一个中断 packet
- key 用于区分不同用途 packet

**`port_cancel`**

`(port_handle, obj_handle, key)`

- undo 上面的 `object_wait_async`

**`interrupt_bind`**

`(intr_handle, port_handle, key, options)`

- 类似 `object_wait_async`，如果中断发生那么入队一个中断 packet
- 中断是需要响应的：`port_wait` 取得 packet 后再手动响应

# 进度

# 基础概念

- 中断也是一种内核对象。中断也是一种资源，和 VMO、I/O Port 一样。
- 一般只有用户态的**驱动**需要关心
- 支持虚拟中断: 用户程序手动触发

# 中断和信号

中断和信号很类似...

- 都是一种事件
- 都可以被等待，或者放到 port 里面

但是

- 信号 deassert 对应中断的 ack。但是 deassert 是满足条件自动设置；而 ack 需要手动完成
- 中断一般只给驱动用，受 resource 管理的限制；而信号基本处处都有

# 相关系统调用：基础

**`interrupt_create`**

`(src_obj, u32, options, handle*)`

- 中断受 resource 限制，只能从已有的资源中创建中断
- 需要提供绑定中断号

**`interrupt_wait`**

`(handle, time*)`

- 阻塞地等待中断被触发
- 等待中断要么这种方式，要么 port 方式，二选一
- 返回中断被触发的时间

# 相关系统调用：port 方式

**`interrupt_bind`**

`(intr_handle, port_handle, key, options)`

- 把中断绑定到 port 上，这样发生中断时会给 port 入队一个 packet

**`interrupt_ack`**

`(handle)`

- 除非被响应 (ack)，否则中断不会再被触发
- 只有 port 方法需要 ack，wait 方式自动 ack

# 相关系统调用：其他

**`interrupt_destroy`**
`(intr_handle)`

- 安全地析构中断对象
- 从所有 port 解绑定，不能再被 wait

**`interrupt_trigger`**
`(handle, options, time)`

- 触发一个虚拟中断