What Is A Work Queue
and
Why Should We Know About It?

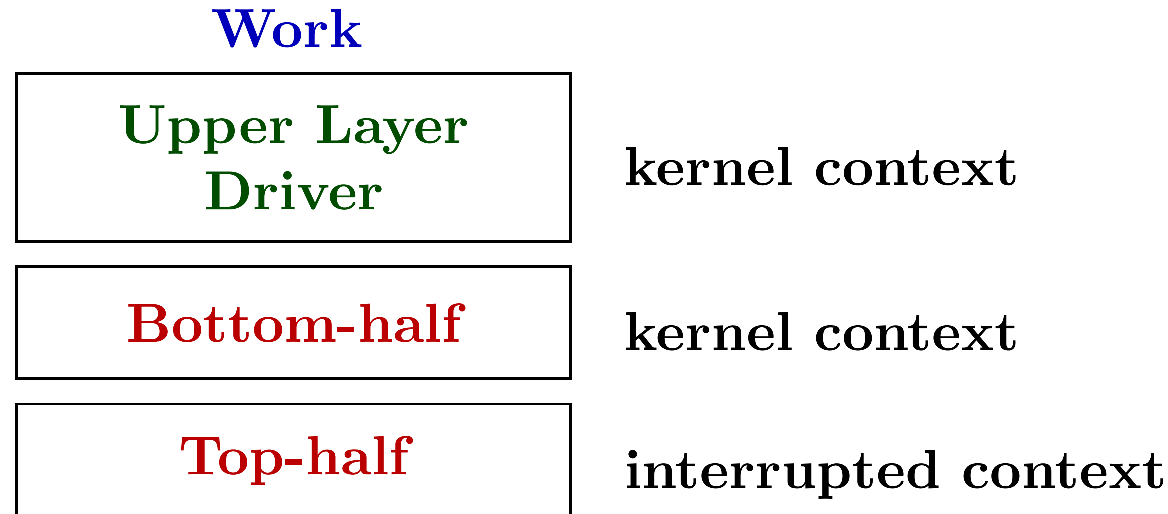
John Franco
Electrical Engineering and Computing Systems
University of Cincinnati

Process Control: managing deferral

May need to defer processing after an interrupt

Typically, some work needs to be done in the context of the interrupt

and some work needs to be passed to other OS elements for added processing



In interrupted context latency may be high because some interrupts may be disabled then

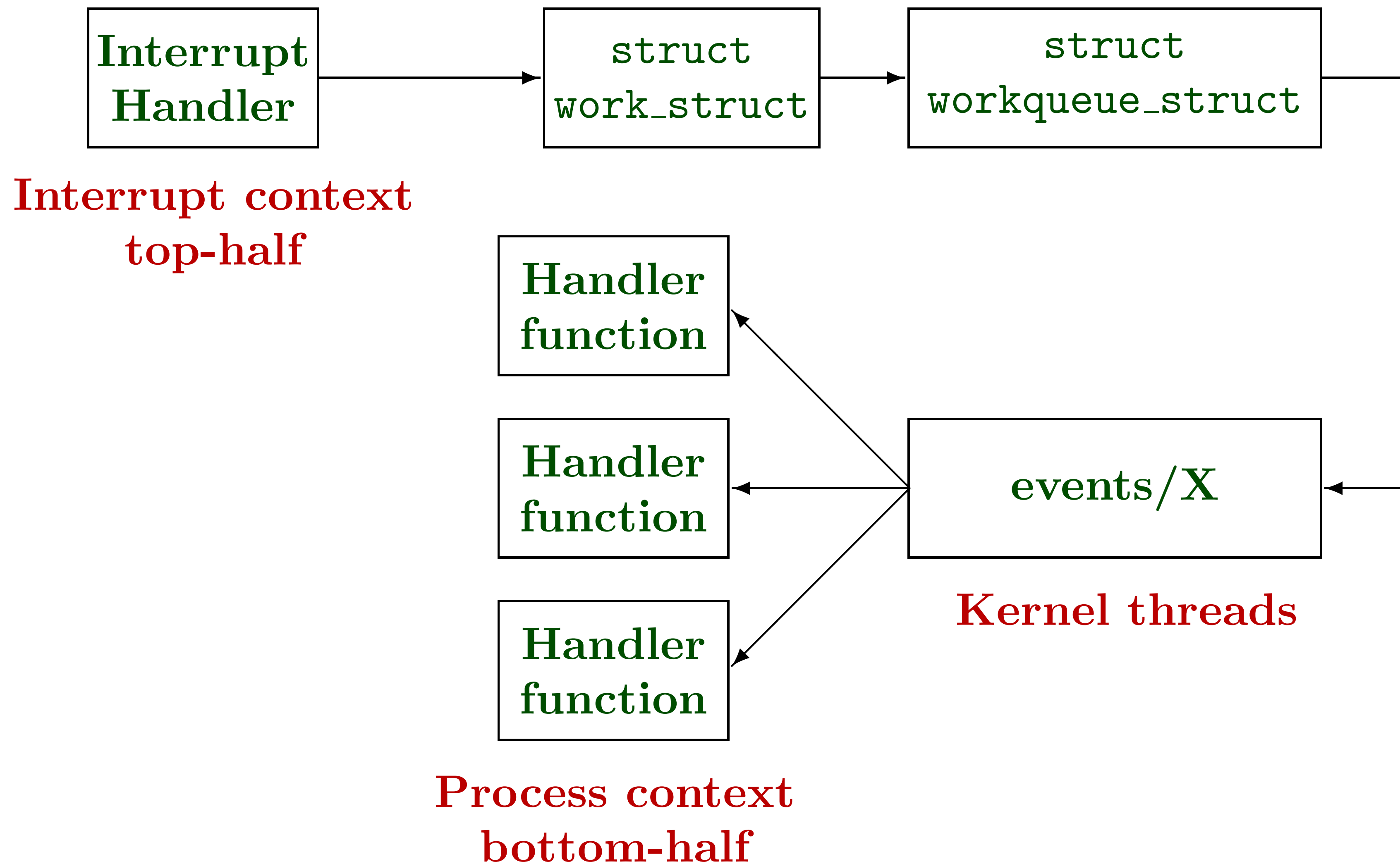
So minimizing work in the interrupt context is desired

This entails pushing some work into the kernel context
where interrupts are all enabled - thus performance not affected

Process Control: work queue

- a generic work deferral mechanism in which the handler function for the work queue can sleep for a specified period
- has a rich API for work deferral
- managed by kernel worker threads named events/X
- provide a generic method to defer functionality to bottom halves

Process Control: work queue



- an object of type `struct work_struct` identifies the deferred work and handler function to use
- work is placed into an object of type `struct workqueue_struct`
- events/X kernel threads extract work from the work queue and activate the specified bottom-half handler

Process Control: work

```
struct work_struct {
   atomic_long_t data;
   struct list_head entry;
   work_func_t func;
};

struct delayed_work {
   struct work_struct work;
   struct timer_list timer;
};
```

Process Control: work queue

```
struct workqueue_struct {
    unsigned int flags;                        /* W: WQ_* flags */
    union {
        struct cpu_workqueue_struct __percpu  *pcpu;
        struct cpu_workqueue_struct           *single;
        unsigned long      v;
    } cpu_wq;                                  /* I: cwq's */
    struct list_head      list;
    struct mutex          flush_mutex;         /* protects wq flushing */
    atomic_t              nr_cwqs_to_flush;    /* flush in progress */
    struct wq_flusher     *first_flusher;      /* F: first flusher */
    struct list_head      flusher_queue;       /* F: flush waiters */
    struct list_head      flusher_overflow;    /* F: flush overflow list */
    mayday_mask_t         mayday_mask;         /* request rescue */
    struct worker         *rescuer;            /* I: rescue worker */
    int                   nr_drainers;         /* W: drain in progress */
    int                   saved_max_active;    /* W: saved cwq max_active */
    char                  name[];              /* I: workqueue name */
};
```

Process Control: work queue

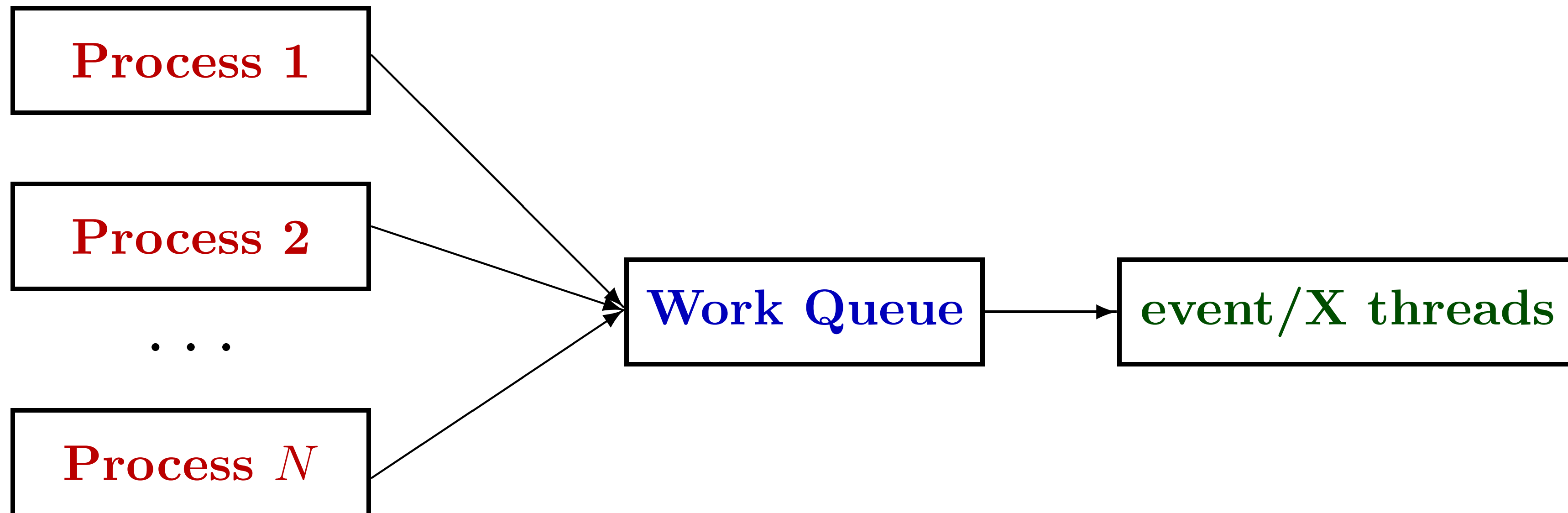
- A work queue is created with
  `struct workqueue_struct *create_workqueue(char *name);`
  the "work" to be done gets sent to the workqueue created here

- A work queue is destroyed with
  `void destroy_workqueue(struct workqueue_struct *wq);`

- A work queue may be initialized with the following macro:
  `INIT_DELAYED_WORK(struct delayed_work*,func(struct work_struct*))`
  where the first argument points to space allocated by `kmalloc` as so:
  `kmalloc(sizeof(struct delayed_work),GFP_KERNEL)`
  and the second argument is a function that does the work

- A work item is enqueued using the following:
  ```
  queue_delayed_work(struct workqueue_struct *,
                     struct delayed_work *, unsigned long);
  ```
  where the first argument is the object returned by `create_workqueue`
  the second argument is the object used as first argument to `INIT_WORK`
  and the third argument is the delay in jiffies.

Process Control: work queue



- To flush the work queue and block until complete do this:
  `flush_workqueue(struct workqueue_struct *wq);`
- To cancel work if it has not reached the handler do this:
  `cancel_delayed_work(struct delayed_work *wk);`
- To check whether work is in the queue do this:
  `delayed_work_pending(struct delayed_work *wk);`