ISCAS NIST

# 循序渐进，学习开发一个 RISC-V 上的操作系统

## 第 9 章 上下文切换和协作式多任务

汪辰

2021/4

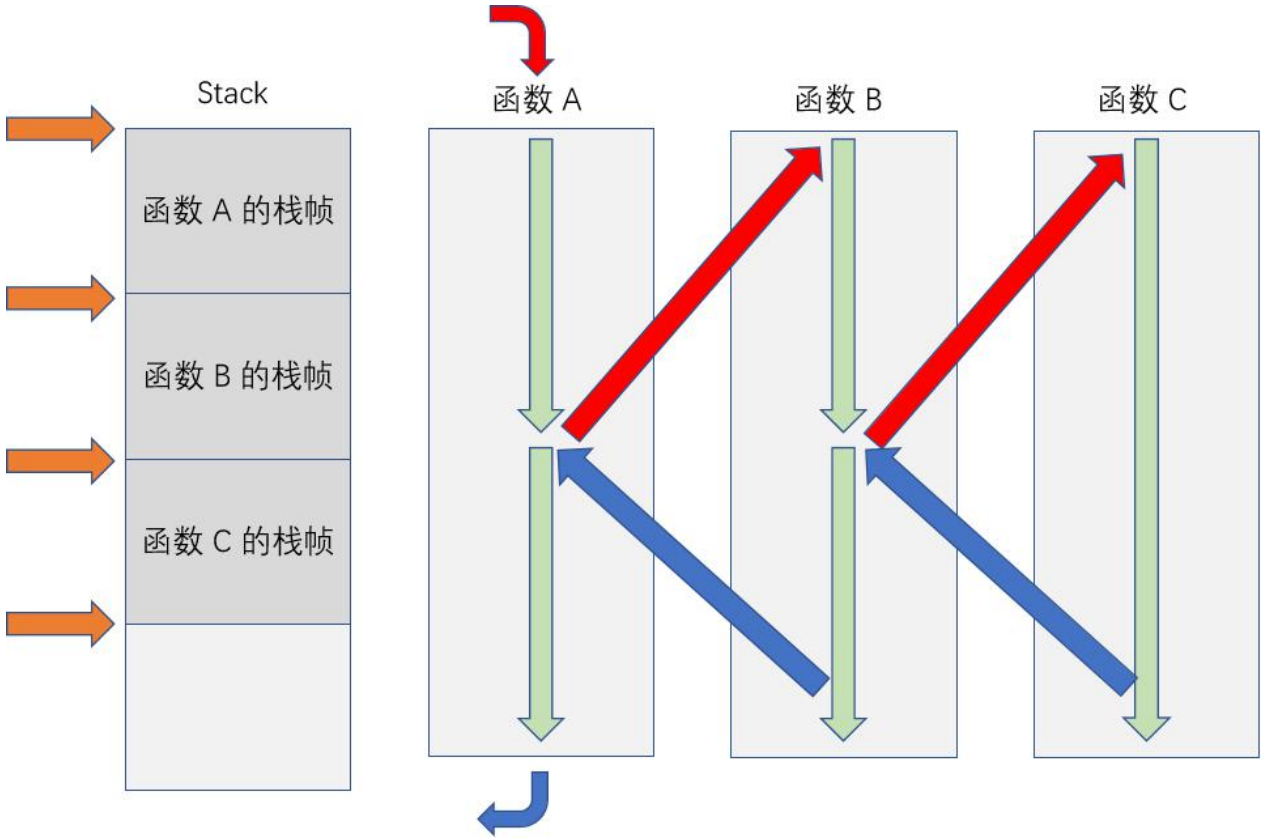
# 本章内容

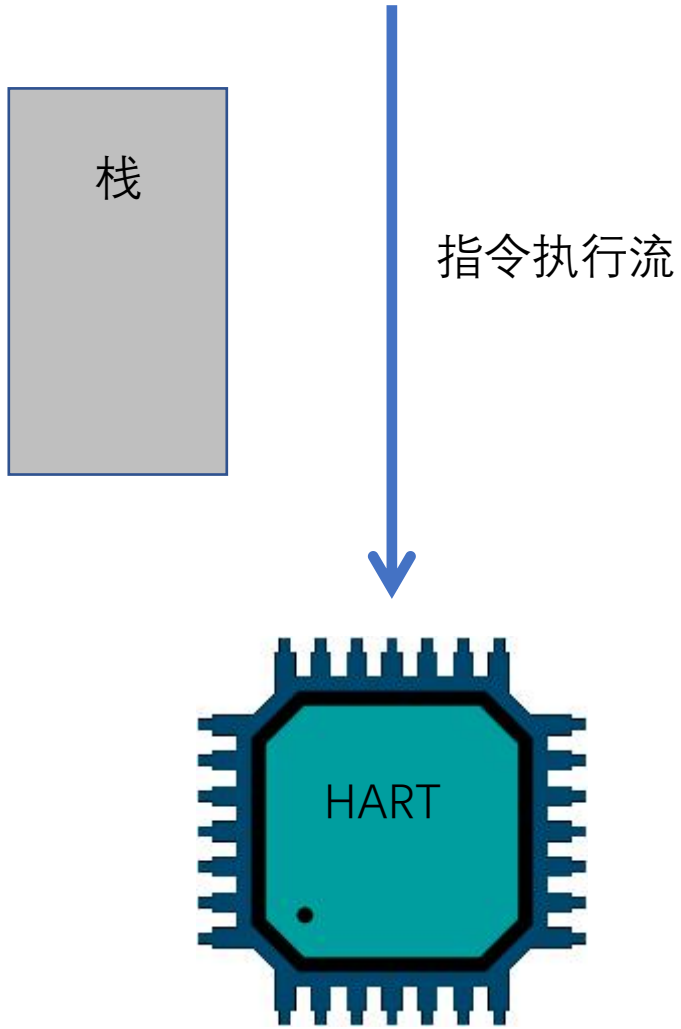
- 多任务与上下文
- 协作式多任务的设计与实现

# 本章内容

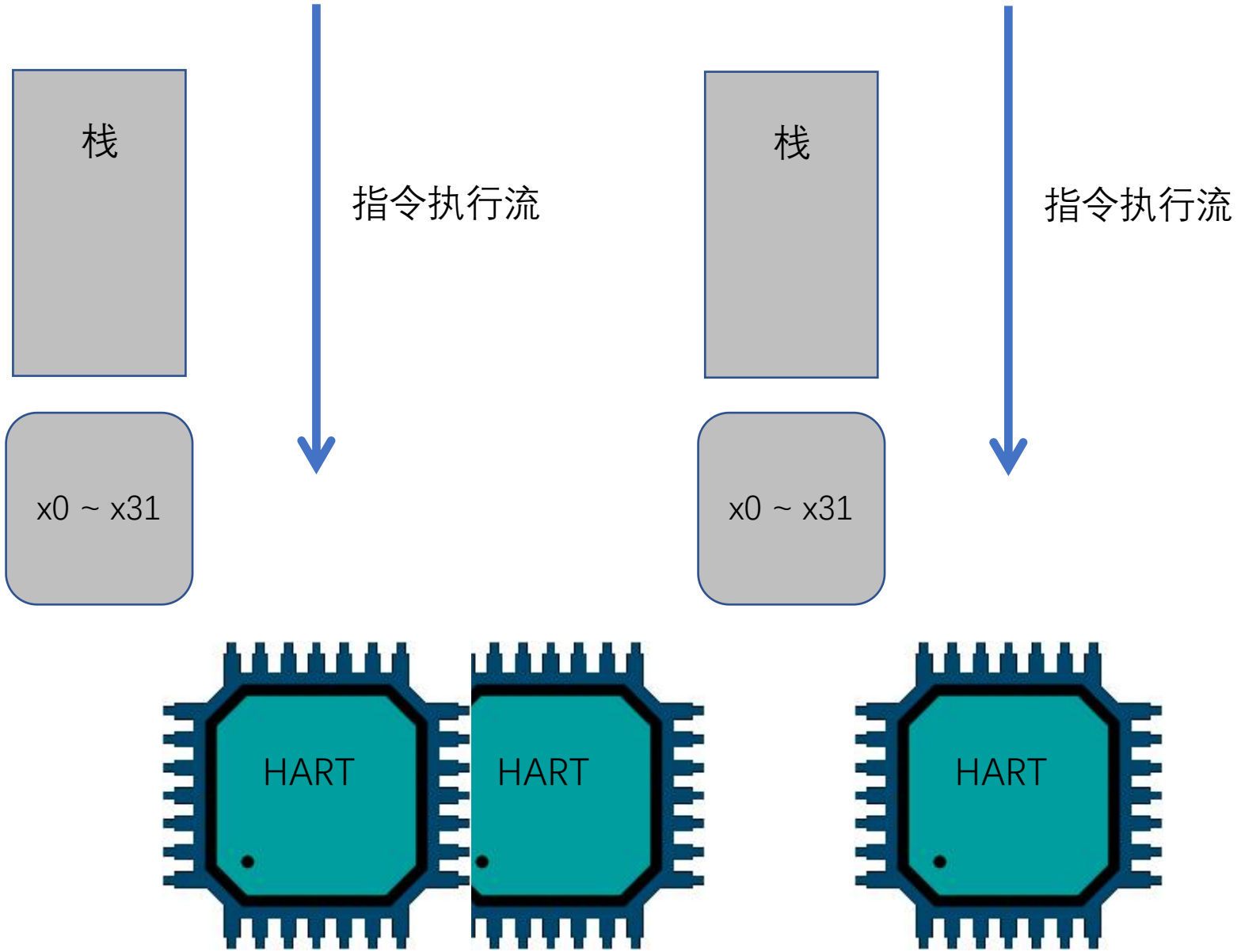
- **多任务与上下文**
  - 任务的概念
  - 多任务的概念
  - 任务上下文的概念
- **协作式多任务的设计与实现**

# 任务（task）

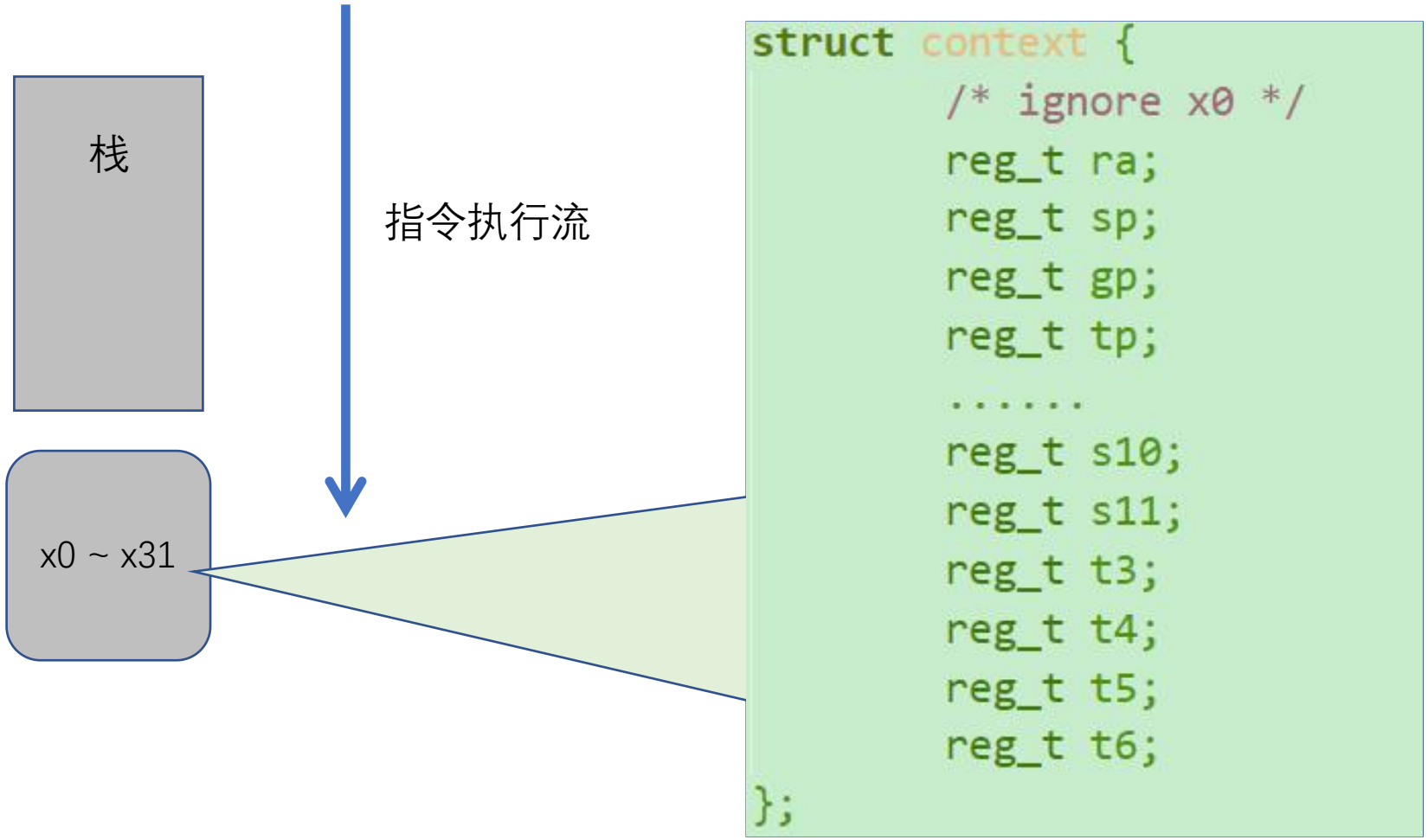


# 多任务 （Multitask）



# 任务上下文（Context）



# 本章内容

- **多任务与上下文**

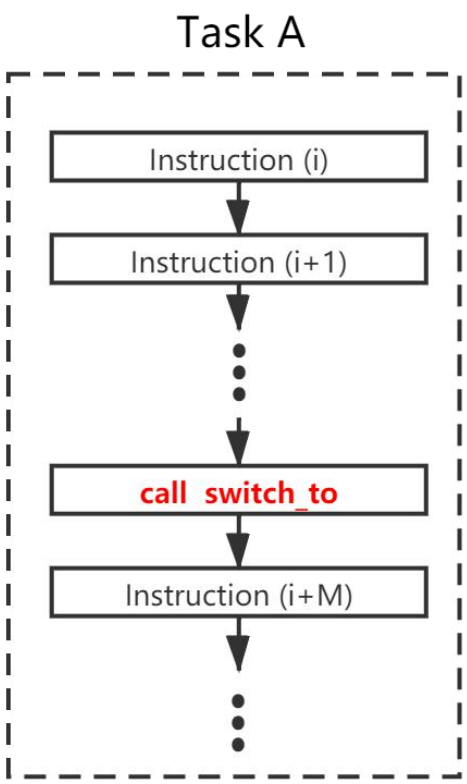
- **协作式多任务的设计与实现**
  - 协作式多任务和抢占式多任务
  - 协作式多任务的设计思路
  - 协作式多任务的关键实现

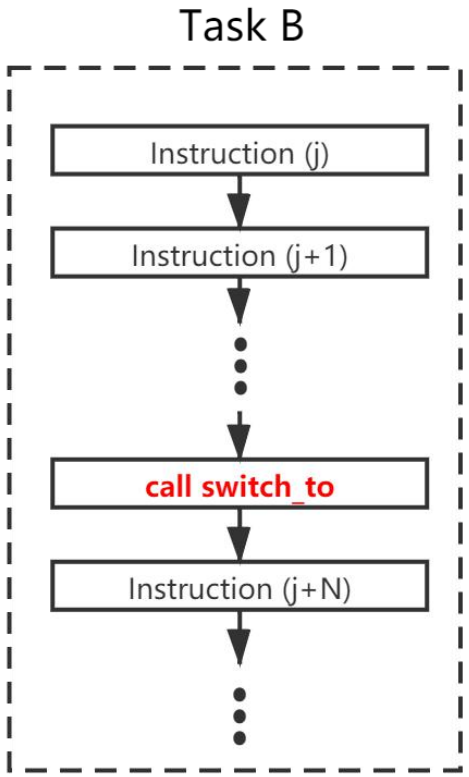
# 多任务系统的分类

- **协作式多任务 (Cooperative Multitasking)**：协作式环境下，下一个任务被调度的前提是当前任务主动放弃处理器。

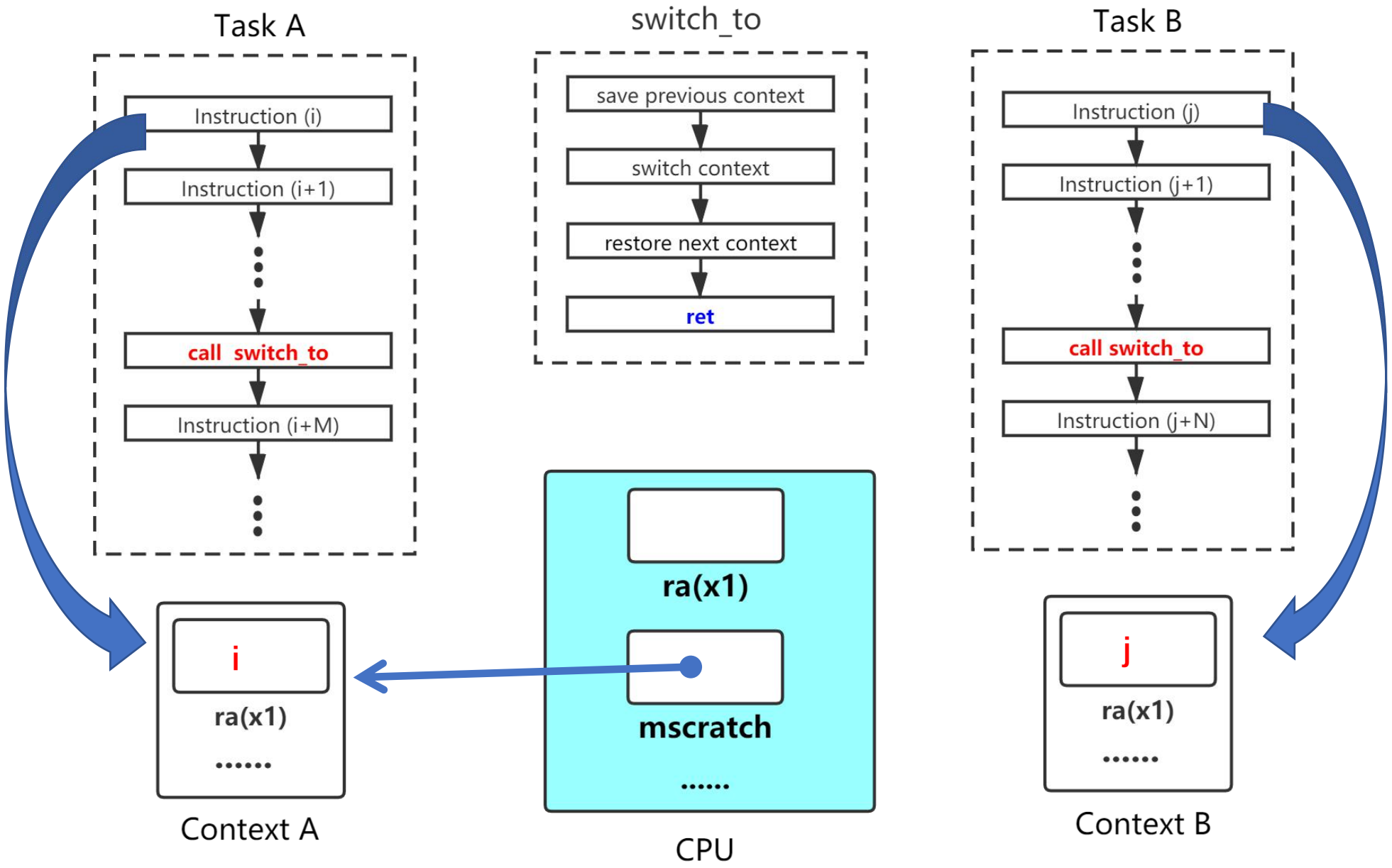
- **抢占式多任务 (Preemptive Multitasking)**：抢占式环境下，操作系统完全决定任务调度方案，操作系统可以剥夺当前任务对处理器的使用，将处理器提供给其它任务。

# 协作式多任务

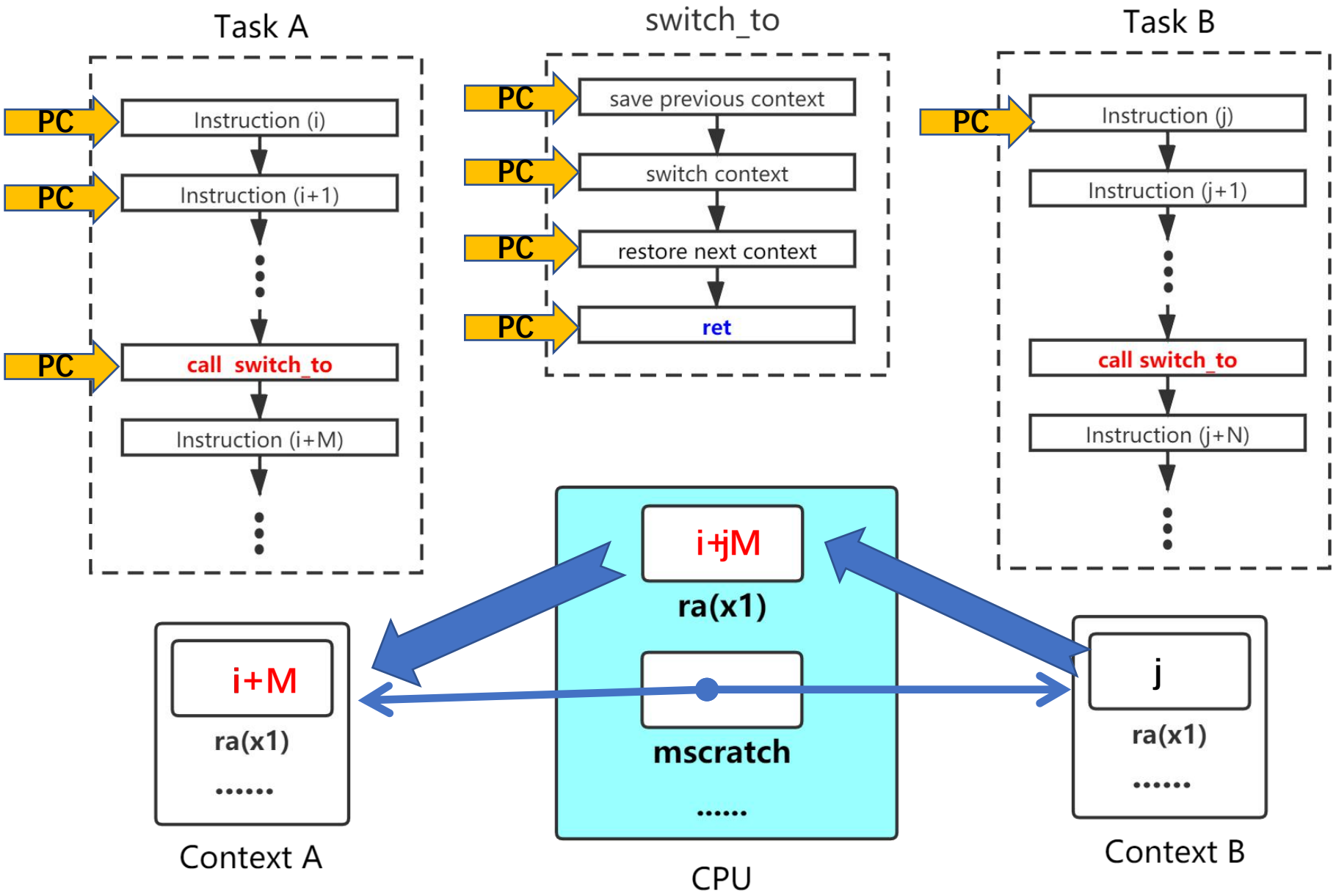

Task A
Instruction (i)
Instruction (i+1)
call switch_to
Instruction (i+M)


Task B
Instruction (j)
Instruction (j+1)
call switch_to
Instruction (j+N)

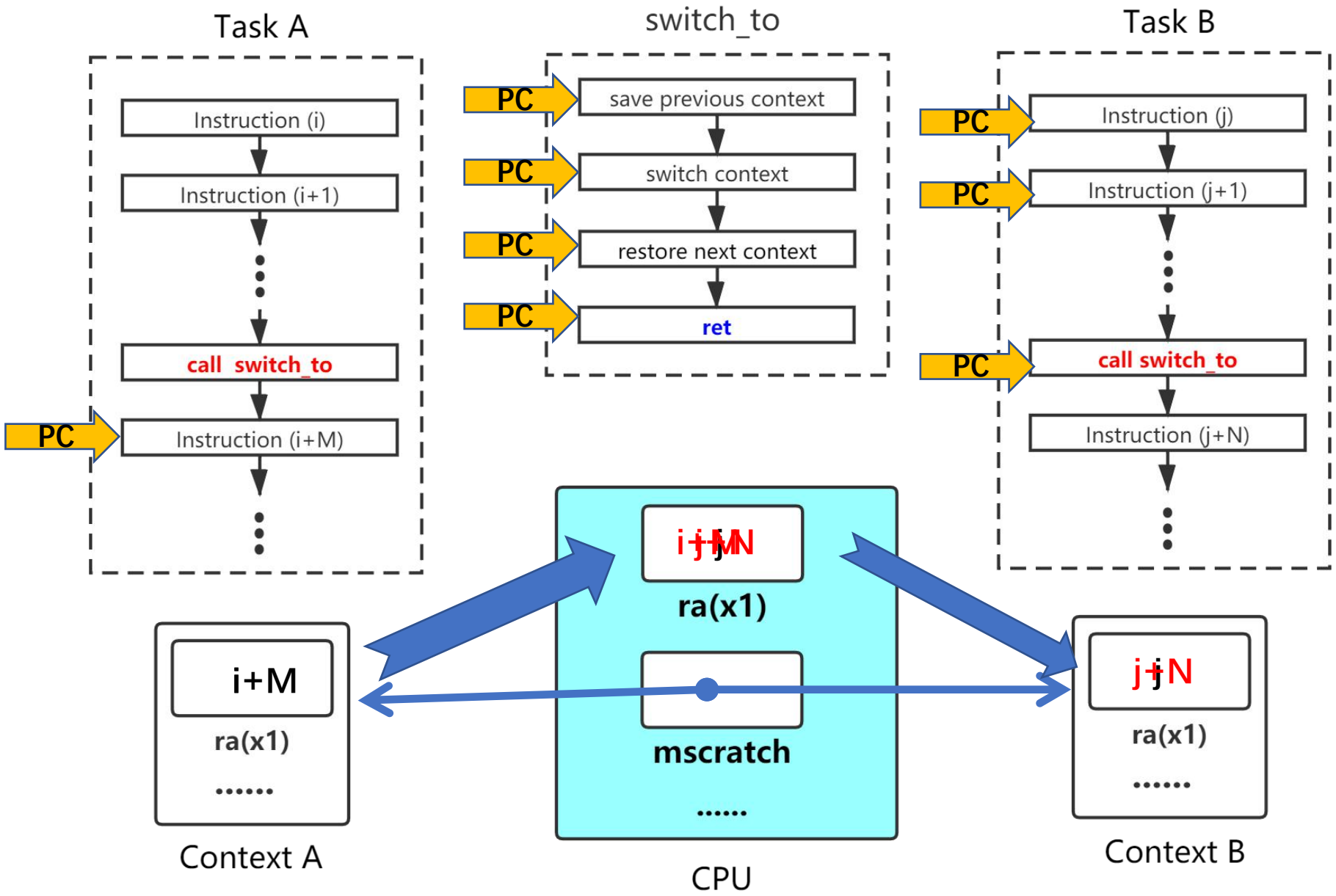
# 协作式多任务 - 初始化


Task A
Instruction (i)
Instruction (i+1)
call switch_to
Instruction (i+M)
switch_to
save previous context
switch context
restore next context
ret
Task B
Instruction (j)
Instruction (j+1)
call switch_to
Instruction (j+N)
ra(x1)
mscratch
.....
CPU
i
ra(x1)
.....
Context A
j
ra(x1)
.....
Context B

# 协作式多任务 - 运行


Task A
switch_to
Task B
PC
Instruction (i)
Instruction (i+1)
call switch_to
Instruction (i+M)
save previous context
switch context
restore next context
ret
Instruction (j)
Instruction (j+1)
call switch_to
Instruction (j+N)
i+jM
ra(x1)
mscratch
.....
i+M
ra(x1)
.....
Context A
CPU
j
ra(x1)
.....
Context B

# 协作式多任务


Task A
Instruction (i)
Instruction (i+1)
call switch_to
Instruction (i+M)
PC
switch_to
save previous context
switch context
restore next context
ret
Task B
Instruction (j)
Instruction (j+1)
call switch_to
Instruction (j+N)
i+M
ra(x1)
.....
Context A
ra(x1)
mscratch
.....
CPU
j+N
ra(x1)
.....
Context B

# 关键函数（switch_to）

```
# void switch_to(struct context *next);
# a0: pointer to the context of the next task
.globl switch_to
.align 4
switch_to:
        csrrw   t6, mscratch, t6        # swap t6 and mscratch
        beqz    t6, 1f                  # Notice: previous task may be NULL
        reg_save t6                     # save context of prev task

        # Save the actual t6 register, which we swapped into
        # mscratch
        mv      t5, t6          # t5 points to the context of current task
        csrr    t6, mscratch    # read t6 back from mscratch
        sw      t6, 120(t5)     # save t6 with t5 as base

1:
        # switch mscratch to point to the context of the next task
        csrw    mscratch, a0

        # Restore all GP registers
        # Use t6 to point to the context of the new task
        mv      t6, a0
        reg_restore t6

        # Do actual context switching.
        ret
```

# 创建和初始化第 1 号任务

code/os/03-contextswitch

```
struct context {
        /* ignore x0 */
        reg_t ra;
        reg_t sp;
        reg_t gp;
        reg_t tp;
        ......
        reg_t s10;
        reg_t s11;
        reg_t t3;
        reg_t t4;
        reg_t t5;
        reg_t t6;
};
```

```
#define STACK_SIZE 1024
uint8_t task_stack[STACK_SIZE];
struct context ctx_task;
```

```
void user_task0(void)
{
        uart_puts("Task 0: Created!\n");
        while (1) {
                uart_puts("Task 0: Running...\n");
                task_delay(1000);
        }
}
```

```
void sched_init()
{
        w_mscratch(0);

        ctx_task.sp = (reg_t) &task_stack[STACK_SIZE - 1];
        ctx_task.ra = (reg_t) user_task0;
}
```

# 踏出 context switch 的第一步，切换到第一个用户任务

code/os/03-contextswitch

```
#define STACK_SIZE 1024
uint8_t task_stack[STACK_SIZE];
struct context ctx_task;
```

```
void user_task0(void)
{
        uart_puts("Task 0: Created!\n");
        while (1) {
                uart_puts("Task 0: Running...\n");
                task_delay(1000);
        }
}
```

```
void schedule()
{
        struct context *next = &ctx_task;
        switch_to(next);
}
```

```
void start_kernel(void)
{
        ......
        sched_init();

        schedule();

        uart_puts("Would not go here!\n");
        while (1) {}; // stop here!
}
```

# 协作式多任务 - 调度

code/os/04-multitask

```
#define MAX_TASKS 10
#define STACK_SIZE 1024
uint8_t task_stack[MAX_TASKS][STACK_SIZE];
struct context ctx_tasks[MAX_TASKS];
```

```
/*
 * _top is used to mark the max available position of ctx_tasks
 * _current is used to point to the context of current task
 */
static int _top = 0;
static int _current = -1;
```

```
void schedule()
{
        ......
        _current = (_current + 1) % _top;
        struct context *next = &(ctx_tasks[_current]);
        switch_to(next);
}
```

```
void task_yield()
{
        schedule();
}
```

# 协作式多任务 - 初始化和任务创建

code/os/04-multitask

```
void sched_init()
{
        w_mscratch(0);
}
        ctx_task.sp = (reg_t) &task_stack[STACK_SIZE - 1];
int task_create(void (*start_routin)(void))
{
        ......
        ctx_tasks[_top].sp = (reg_t) &task_stack[_top][STACK_SIZE - 1];
        ctx_tasks[_top].ra = (reg_t) start_routin;
        _top++;
        ......
}
```

```
void user_task0(void)
{
        ......
        task_yield();
        ......
}
void user_task1(void)
{
        ......
        task_yield();
        ......
}
void os_main(void)
{
        task_create(user_task0);
        task_create(user_task1);
}
```

# 协作式多任务 - 任务运行

code/os/04-multitask

```
void user_task0(void)
{
        ......
        task_yield();
        ......
}
void user_task1(void)
{
        ......
        task_yield();
        ......
}
void os_main(void)
{
        task_create(user_task0);
        task_create(user_task1);
}
```

```
void schedule()
{
        ......
        _current = (_current + 1) % _top;
        struct context *next = &(ctx_tasks[_current]);
        switch_to(next);
}
```

```
void start_kernel(void)
{
        ......
        sched_init();
        os_main();
        schedule();
        uart_puts("Would not go here!\n");
        while (1) {}; // stop here!
}
```

## 练习 9-1

## 练习 9-2

# 谢 谢

欢迎交流合作