5-3 如何实现时间切片

时间切片

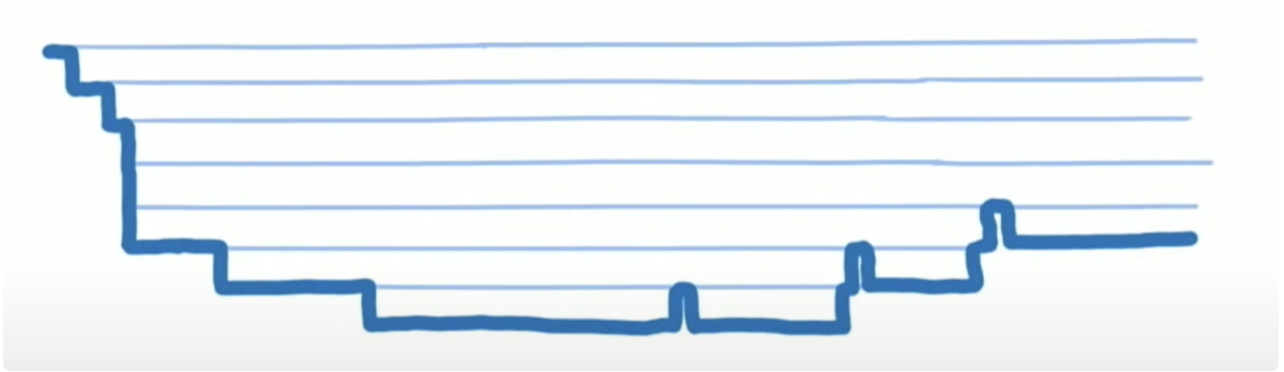
时间切片就是个时间段，比如 5ms。

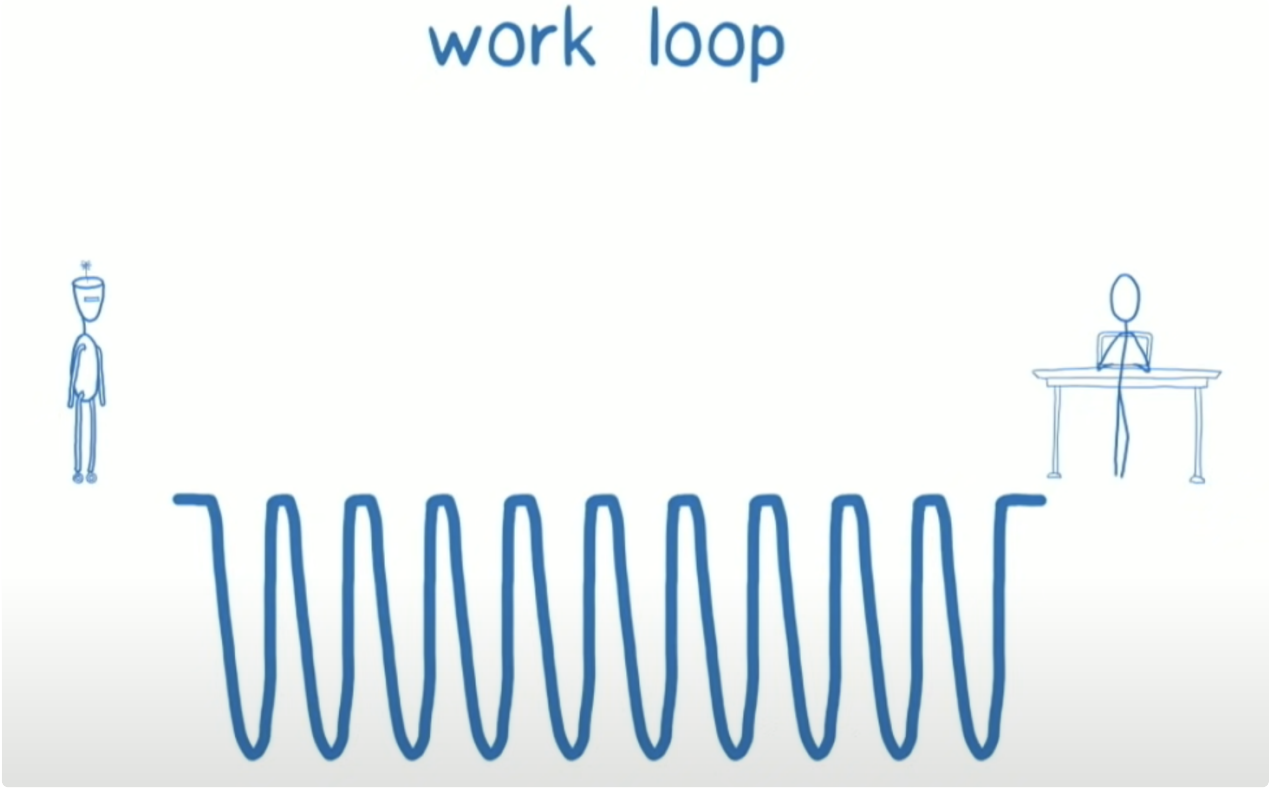
在单线程机制下，如果一个任务执行时间花费过久，容易堵塞后面的任务。早期的操作系统和 React 均遇到过这种问题。

比如 React 页面渲染时候，某一个任务堵塞后面了后面高优先级任务，如与**用户交互任务**、**布局任务**等，这个时候用户就会看到**卡顿现象**。为了解决这种问题，React 参照操作系统，引入了**时间切片**（time slice)机制，在某个时间段内周期性执行任务，即周期性把控制权还给浏览器。

时间切片解决的问题就是，高优先级任务迟迟得不到处理。

这就是 React 从 Stack Reconciler 到 Fiber Reconciler 的迭代，对应图分别如下：





callback、task、work

callback 是任务的初始值，task 是 scheduler 封装之后的 task，work 是指一个时间切片内的工作单元。

TypeScript

```
export type Task = {
  id: number;
  callback: Callback | null;
  priorityLevel: PriorityLevel;
  startTime: number;
  expirationTime: number;
  sortIndex: number;
};
```

何时交还控制权给主线程

TypeScript

```
// 记录时间切片的起始值，时间戳
let startTime = -1;

// 时间切片，这是个时间段
let frameInterval = 5;

function shouldYieldToHost() {
  const timeElapsed = getCurrentTime() - startTime;

  if (timeElapsed < frameInterval) {
    return false;
  }

  return true;
}
```

workLoop

循环执行 work 的函数如下：

TypeScript

```
// 是否有 work 在执行
let isPerformingWork = false;

function workLoop(initialTime: number) {
  // todo
  let currentTime = initialTime;
  // advanceTimers(currentTime);
  currentTask = peek(taskQueue);
  while (currentTask !== null) {
    // 执行task
    // 是否到了该让出控制权的时候了
    if (currentTask.expirationTime > currentTime && shouldYieldToHost(
      break;
    }
    // 执行任务
    const callback = currentTask.callback;
    if (typeof callback === "function") {
      currentTask.callback = null;
```

```
      currentPriorityLevel = currentTask.priorityLevel;
      const didUserCallbackTimeout = currentTask.expirationTime <= cur
      const continuationCallback = callback(didUserCallbackTimeout);
      currentTime = getCurrentTime();
      if (typeof continuationCallback === "function") {
        currentTask.callback = continuationCallback;
        return true;
      } else {
        if (currentTask === peek(taskQueue)) {
          pop(taskQueue);
        }
      }
    } else {
      pop(taskQueue);
    }
    currentTask = peek(taskQueue);
  }

  if (currentTask !== null) {
    return true;
  } else {
    return false;
  }
}
```