深入了解现代网络浏览器（第 2 部分）

Mariko Kosaka

在导航过程中会发生什么

该博客系列共包含 4 部分，专门介绍 Chrome 的内部工作原理，这是第 2 部分。 在上一篇博文中，我们介绍了 进程和线程负责处理浏览器的不同部分在这篇博文中，我们将深入探讨 每个进程和线程都会通信，以便显示网站。

我们来看一个简单的网络浏览用例：您在浏览器中键入网址，然后在浏览器中 从互联网提取数据并显示网页。在这篇博文中，我们将重点介绍 当用户请求访问网站时，浏览器就准备呈现网页（也称为导航）。

首先是浏览器进程

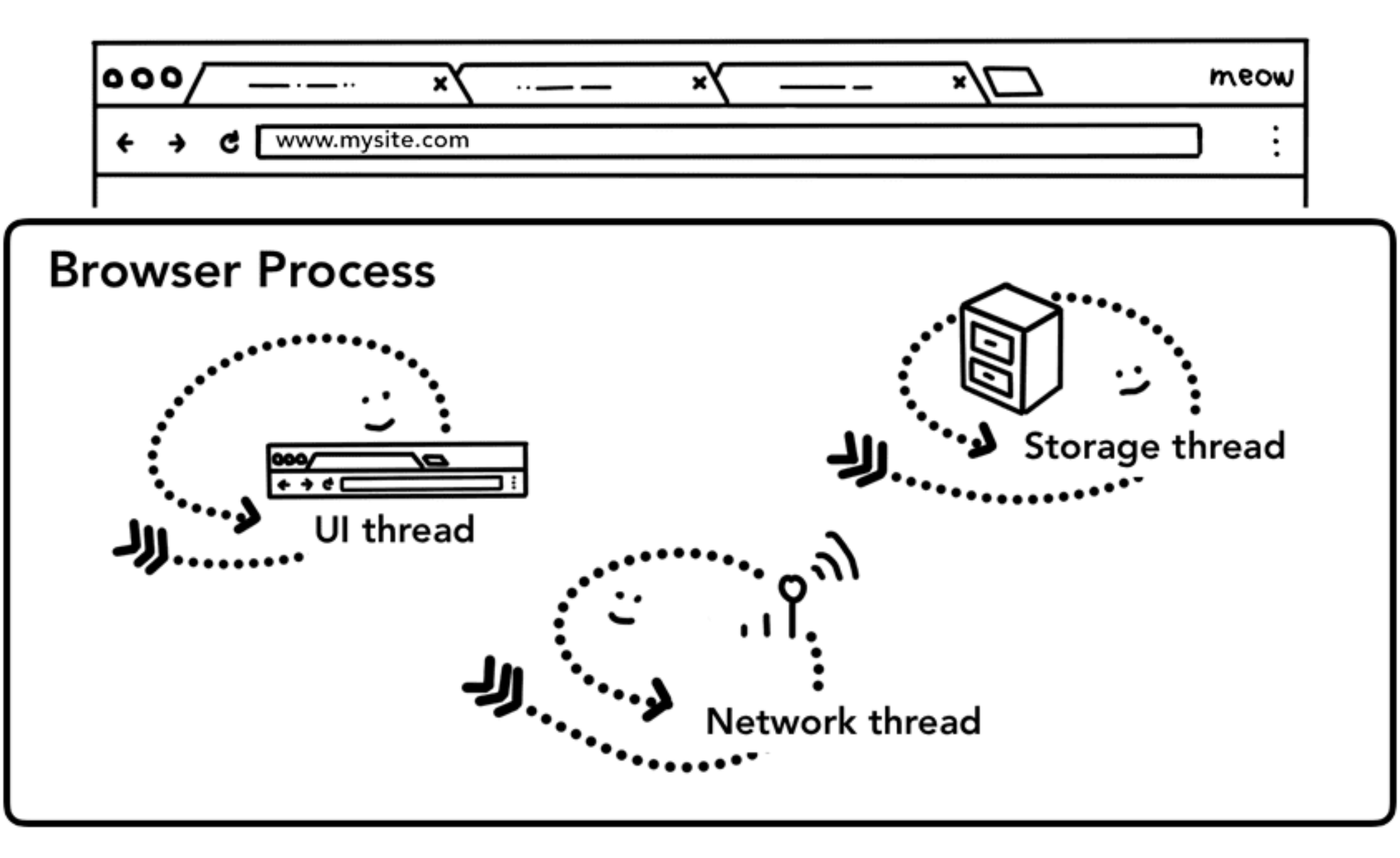

图 1：顶部的浏览器界面，包含界面、网络和存储空间的浏览器进程示意图 消息串在底部

我们在本课程中 第 1 部分：CPU、GPU、内存和多进程架构， 标签页以外的所有内容都由浏览器进程处理。 浏览器进程具有类似于界面线程的线程，用于绘制 浏览器（处理网络堆栈以从互联网接收数据的网络线程）， 控制文件访问等的存储线程。当您在地址中输入网址时 则表示您的输入由浏览器进程的界面线程处理。

简单的导航

第 1 步：处理输入

当用户开始在地址栏中输入内容时，界面线程首先会询问：“这是 搜索查询或网址？”。在 Chrome 中，地址栏也是一个搜索输入字段 需要解析并决定是将您转到搜索引擎还是您请求的网站。

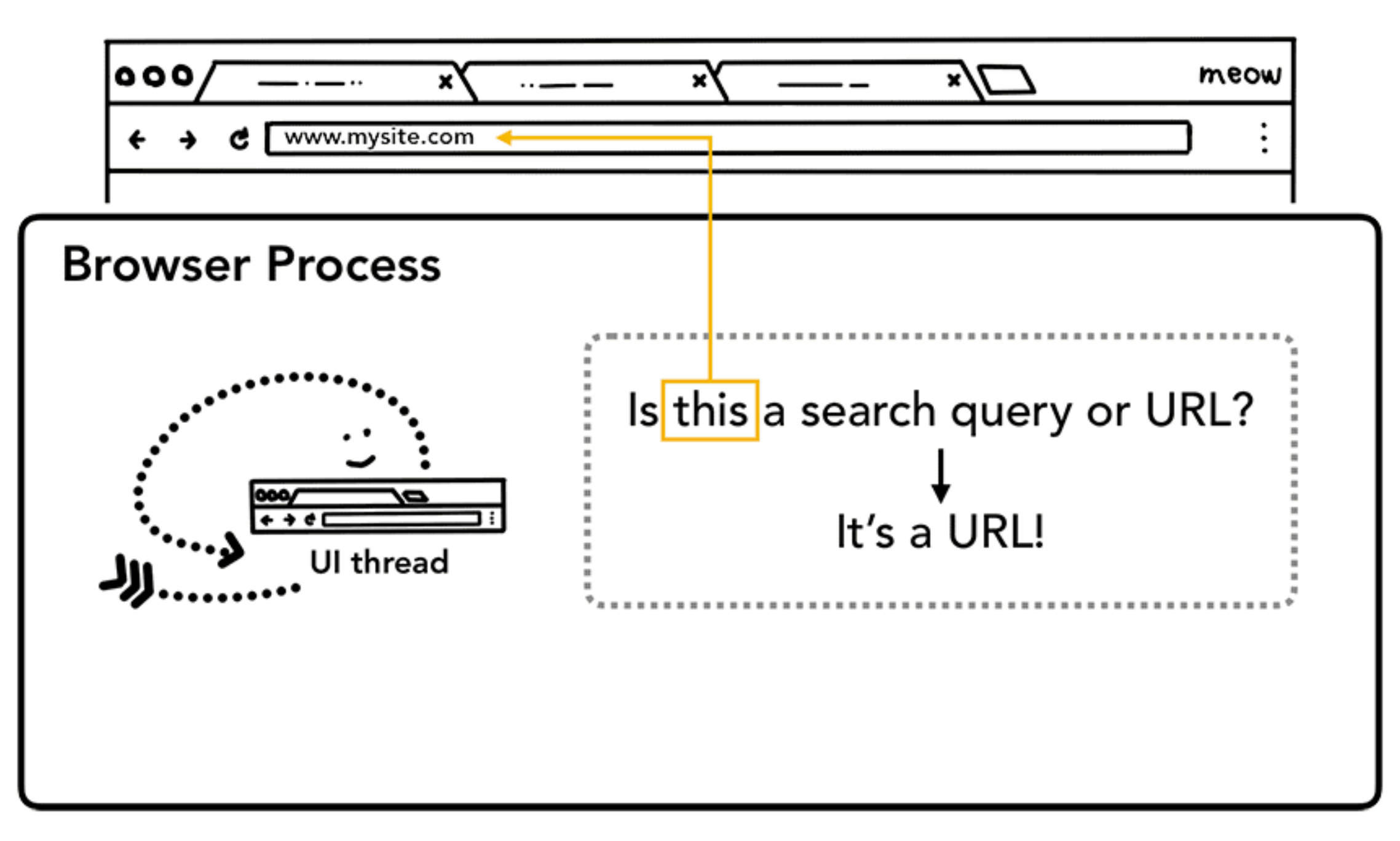

图 1：询问输入是搜索查询还是网址的界面线程

第 2 步：开始导航

当用户点击进入按钮时，界面线程会发起网络调用以获取网站内容。加载旋转图标 显示在标签页的一角，且网络线程会经过相应的协议，例如 为请求进行 DNS 查找并建立 TLS 连接。

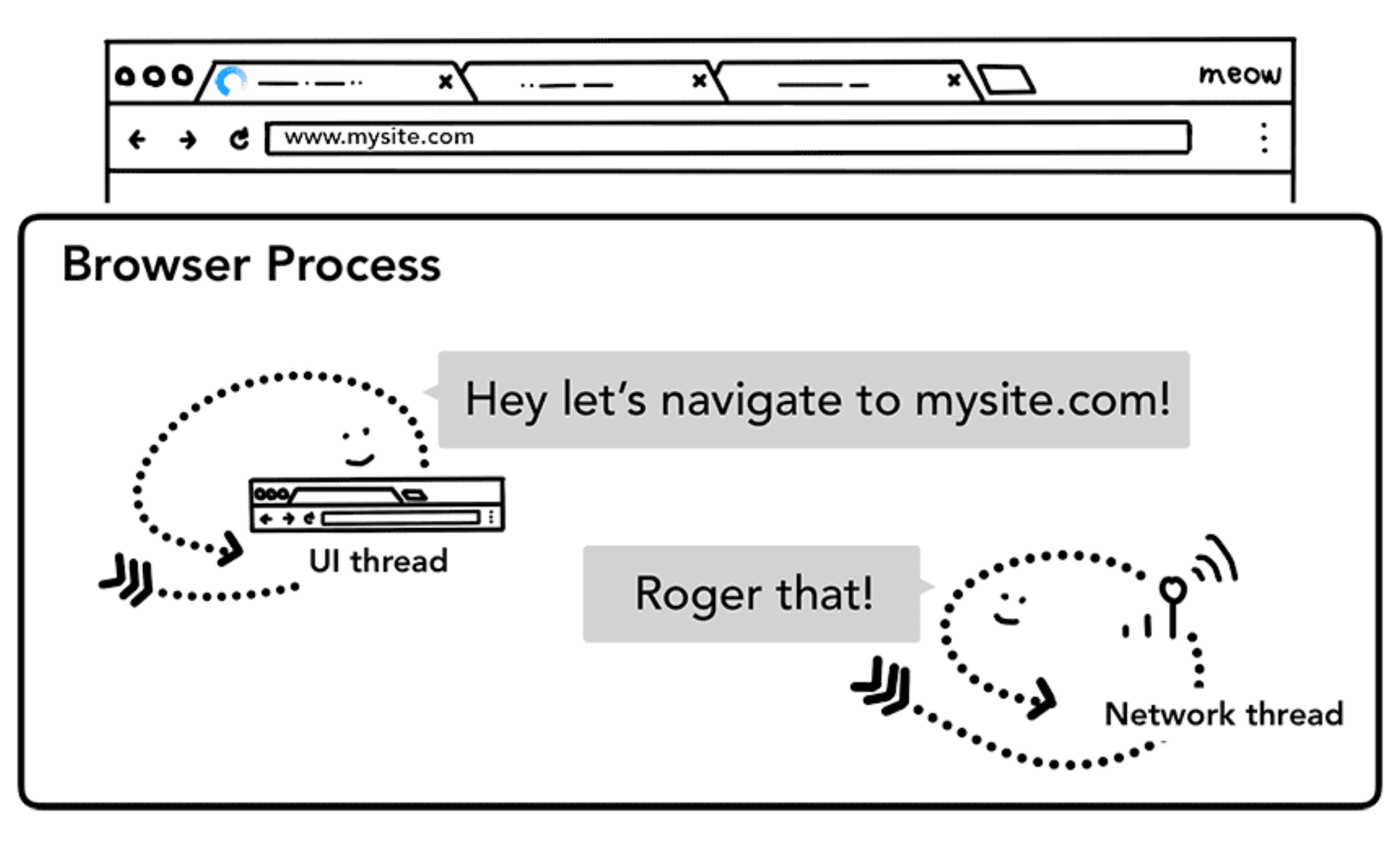

图 2：与网络线程进行通信以导航到 mysite.com 的界面线程

此时，网络线程可能会收到服务器重定向标头，如 HTTP 301。在这种情况下 网络线程与服务器请求重定向的界面线程进行通信。然后， 系统会发起另一个网址请求

第 3 步：阅读回答

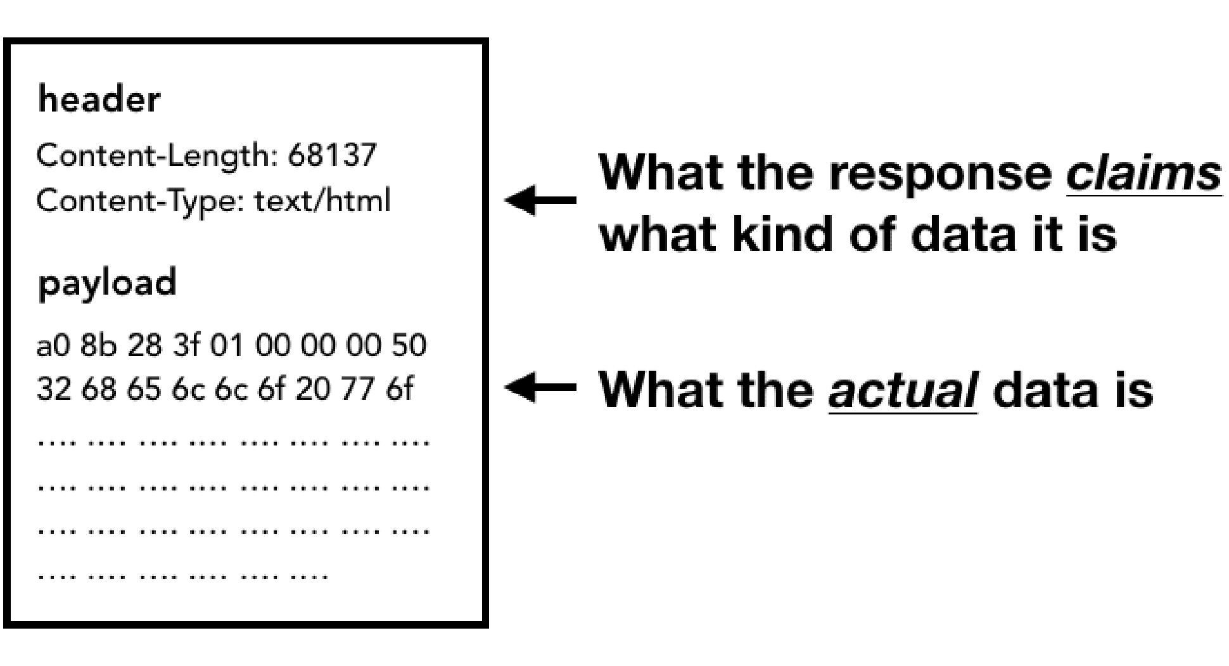

图 3：包含 Content-Type 和有效负载（实际数据）的响应标头

一旦响应正文（载荷）开始进入，网络线程会查看前几个字节 （如有必要）。响应的 Content-Type 标头应指明数据类型 但由于它可能有缺失或错误 MIME 类型嗅探 这里就完成了这是一项“棘手的业务”如源代码中所注释的那样。 您可以阅读该注释，了解不同的浏览器如何处理“内容类型/载荷”对。

如果响应是一个 HTML 文件，那么下一步是将数据传递给渲染程序 但如果是 zip 文件或其他文件，则表示它是一个下载请求， 他们需要将数据传递给内容下载管理器。

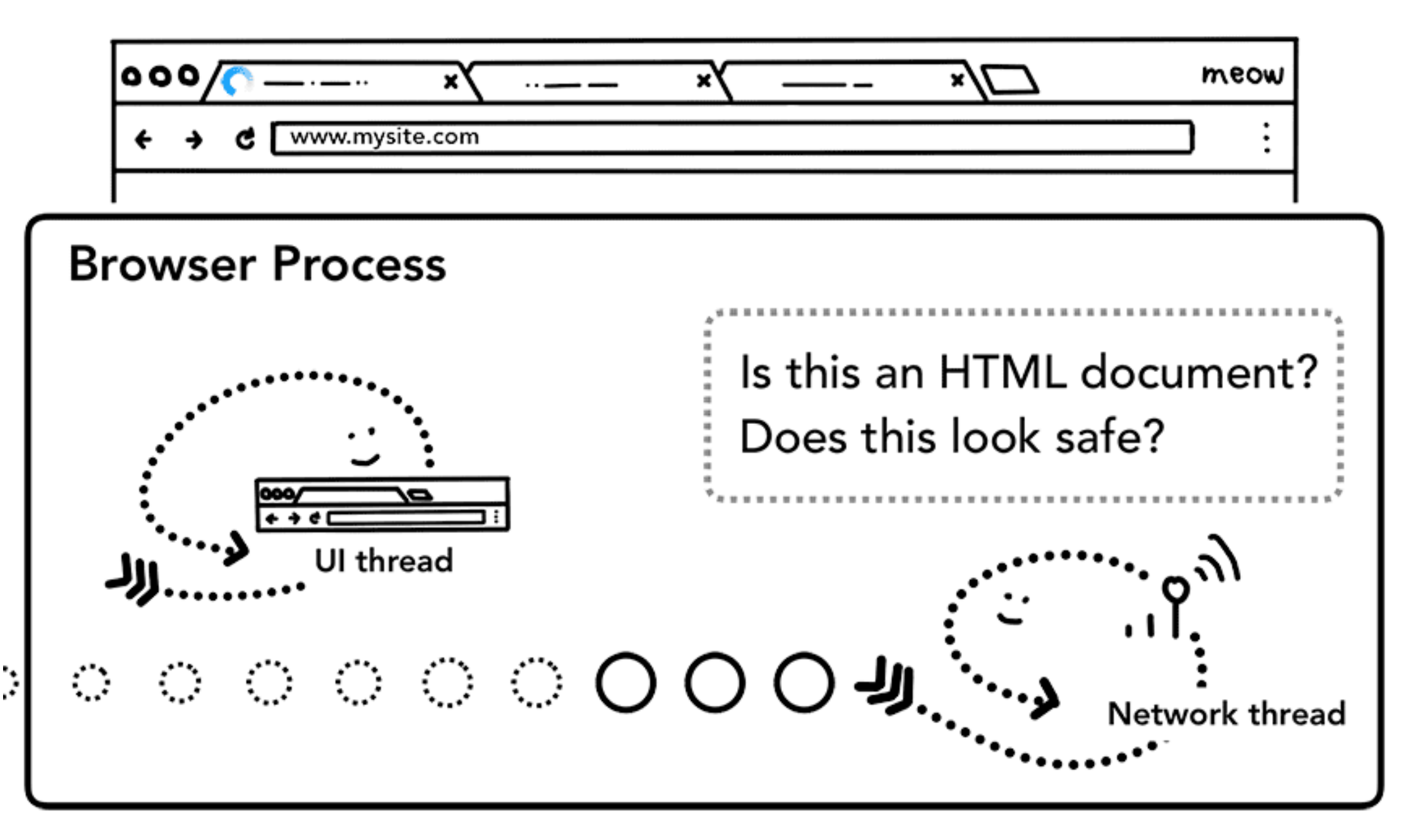

图 4：询问响应数据是否为来自安全网站的 HTML 的网络线程

这也是SafeBrowsing检查的地方。 如果域名和响应数据似乎与某个已知的恶意网站匹配，那么此网络线程 显示警告页面。此外， Ross Origin Read Blocking (CORB) 以便确保跨网站 数据不会进入渲染器进程。

第 4 步：查找渲染程序进程

完成所有检查且网络线程确信浏览器应导航到 请求的网站，网络线程会通知界面线程数据已就绪。然后，界面线程会 渲染程序进程继续渲染网页

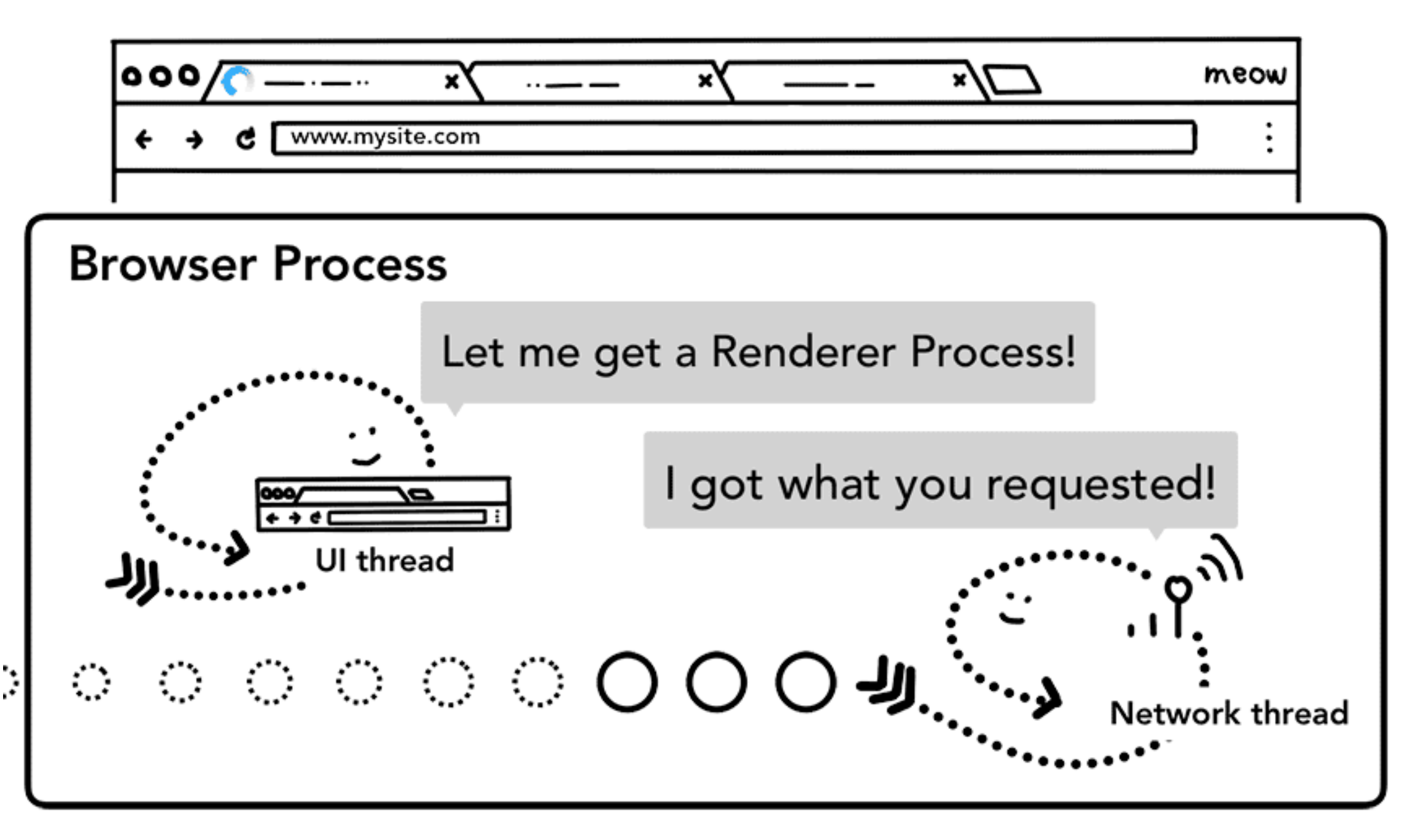

图 5：让界面线程查找渲染程序进程的网络线程

由于网络请求可能需要几百毫秒才能获取响应， 以加快此过程。当界面线程向 网络线程，那么它已经知道用户要导航到哪个网站。界面线程 尝试主动查找或启动与网络请求并行的渲染器进程。这样， 如果一切按预期进行，那么在网络线程收到的数据。如果导航在跨网站重定向中， 在这种情况下，可能需要执行不同的流程。

第 5 步：提交导航

现在，数据和渲染程序进程都已准备就绪，浏览器进程向 渲染程序进程提交导航。它还会传递数据流 进程可以持续接收 HTML 数据。当浏览器进程听到确认已提交 发生在渲染器进程中，导航已完成，且文档加载阶段 。

此时，地址栏会更新，安全指示器和网站设置界面会反映 新网页的网站信息。该标签页的会话历史记录会进行更新，以便往返 按钮可以分步浏览刚刚浏览的网站。便于恢复标签页/会话 当您关闭标签页或窗口时，会话记录会存储在磁盘中。

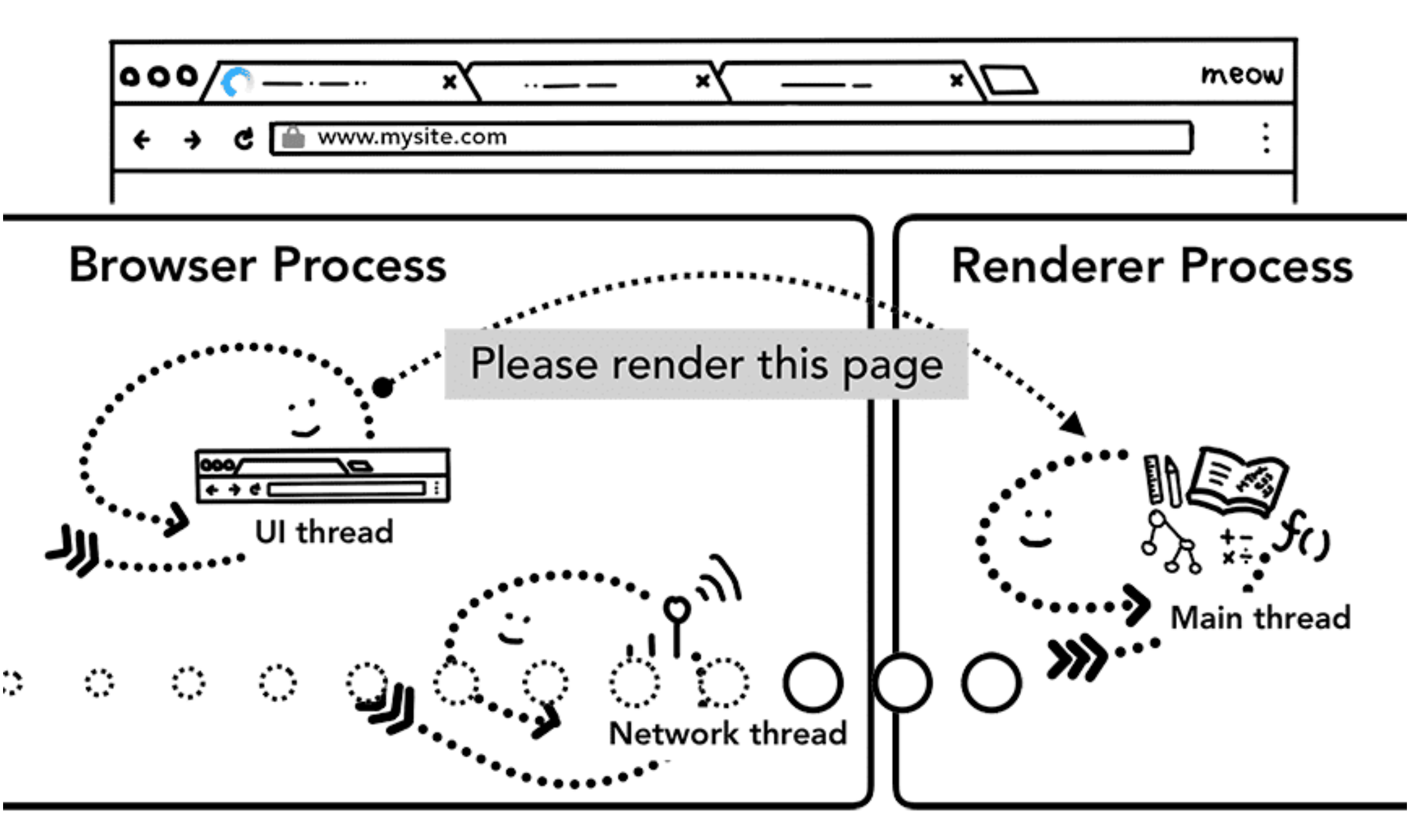

图 6：浏览器和渲染程序进程之间的 IPC，用于请求渲染网页

额外步骤：初始加载完成

提交导航后，渲染器进程会加载资源， 页面。我们将在下一篇帖子中详细介绍此阶段会发生什么。渲染程序 进程“完成”它会将 IPC 发送回浏览器进程（完成所有 `onload` 事件已针对页面中的所有帧触发并执行完毕）。此时， 界面线程会停止标签页上的“正在加载”旋转图标。

我说“finishes”，因为客户端 JavaScript 仍然可以加载 并在此后渲染新的视图。

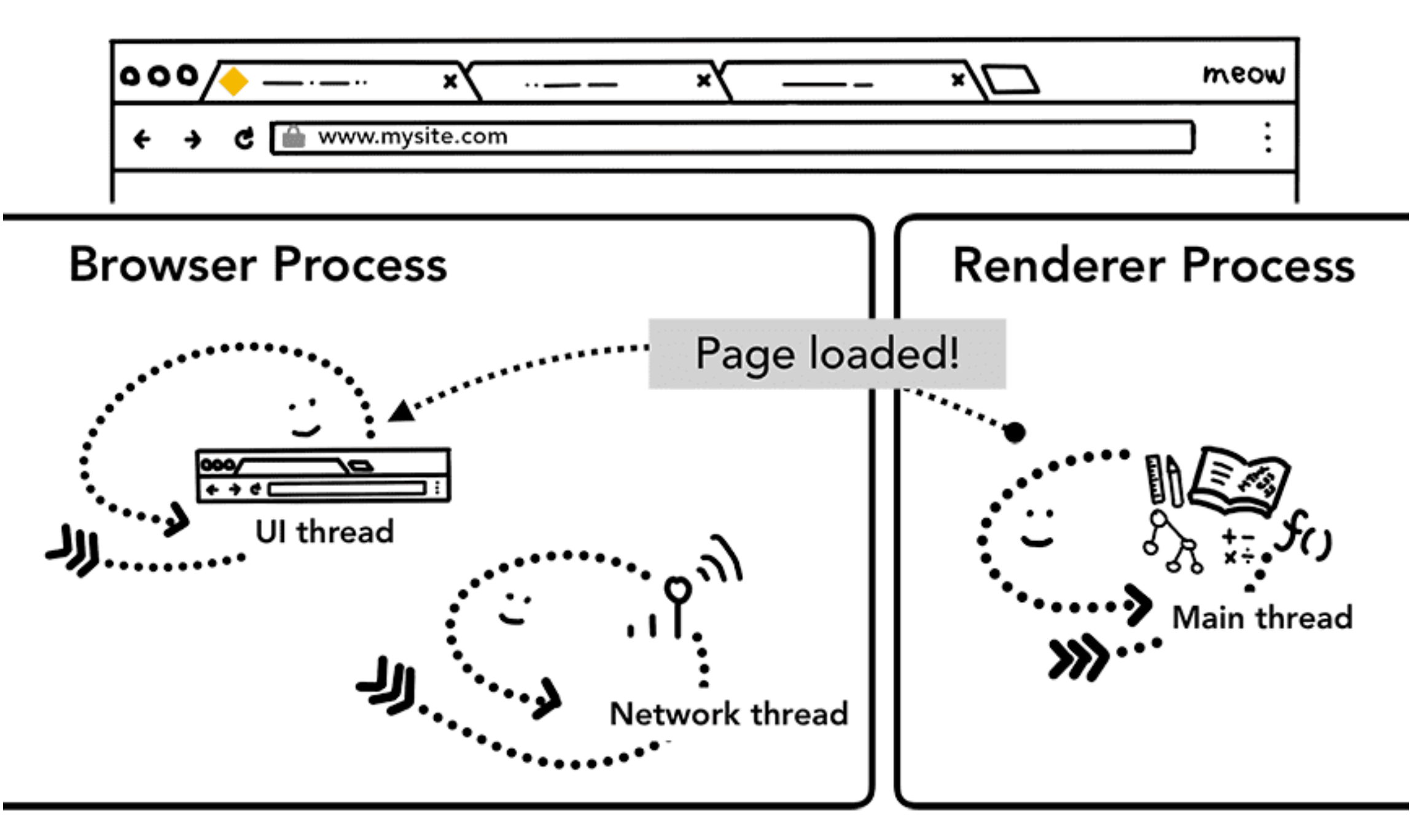

图 7：从渲染程序到浏览器进程的 IPC，用于通知页面“加载”

导航到其他网站

简单的导航已完成！但是，如果用户在地址栏中输入不同的网址，会出现什么情况 吗？浏览器进程要执行相同的步骤才能导航到不同的网站。 但在此之前，它需要确认当前呈现的网站是否关心 `beforeunload` 事件。

`beforeunload` 可以创建“要离开此网站吗？”在您尝试离开或关闭标签页时发送提醒。 标签页内的所有内容（包括 JavaScript 代码）均由渲染器进程处理，因此 收到新导航请求时，浏览器进程必须与当前的渲染程序进程进行检查。

> **注意**：请勿添加无条件 `beforeunload` 处理程序。这会增加延迟时间，因为 处理程序需要在导航开始之前执行。此事件处理脚本应 仅在需要时添加，例如在用户需要警告他们可能会丢失数据的情况下 。

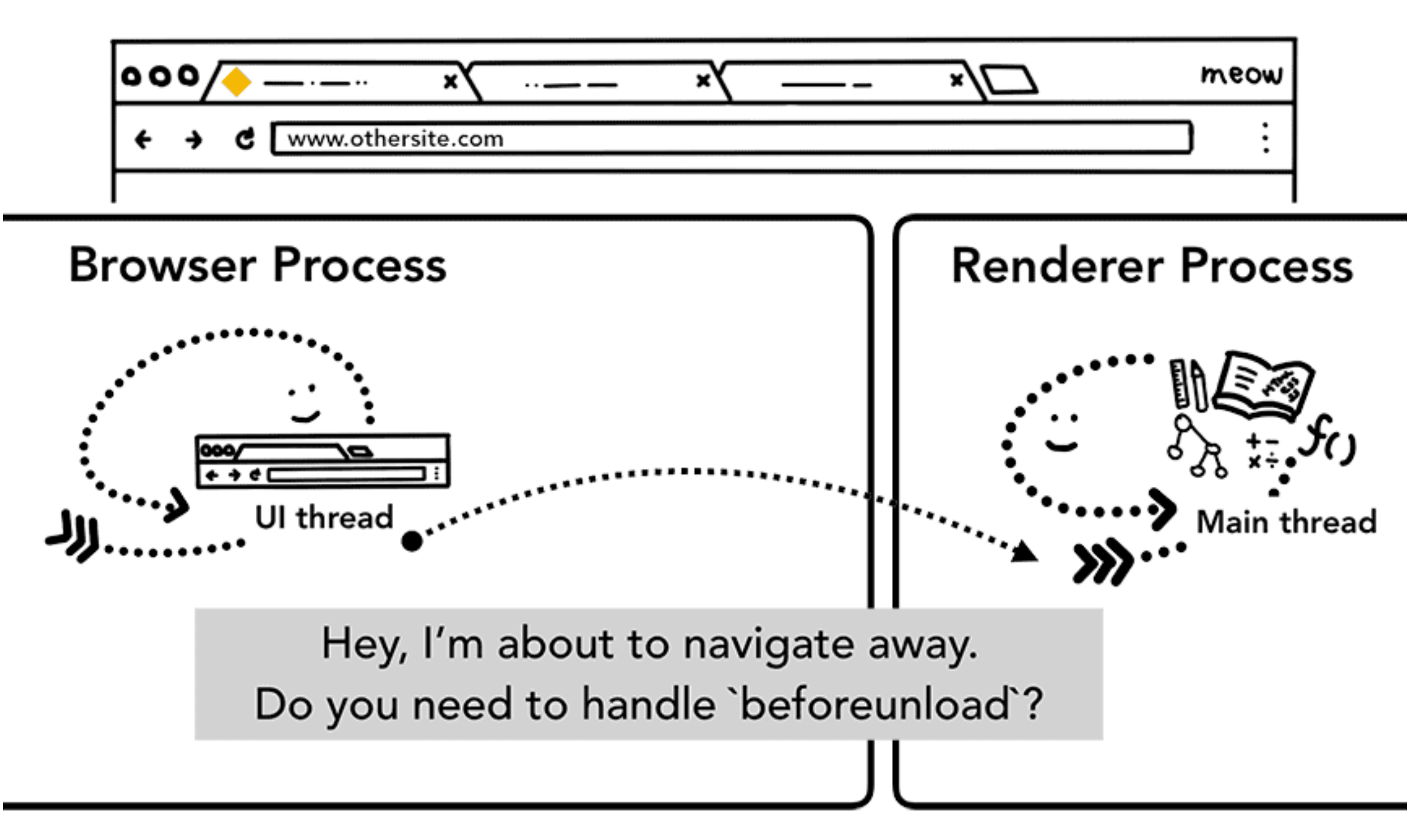

图 8：从浏览器进程到渲染程序进程的 IPC，指示它即将进行 导航到其他网站

如果导航是从渲染程序进程发起的（例如用户点击链接或 客户端 JavaScript 已运行 `window.location = "https://newsite.com"`）渲染器进程 首先检查 `beforeunload` 处理程序。然后，它会通过与浏览器进程相同的进程启动导航。唯一的区别在于，导航请求是从 渲染程序进程传递到浏览器进程

当新的导航指向与当前呈现的网站不同的网站时，单独的呈现 系统会调用进程来处理新的导航，同时将当前渲染进程保留在 处理 `unload` 等事件。如需了解详情，请参阅页面生命周期状态概览 以及如何借助 Page Lifecycle API。

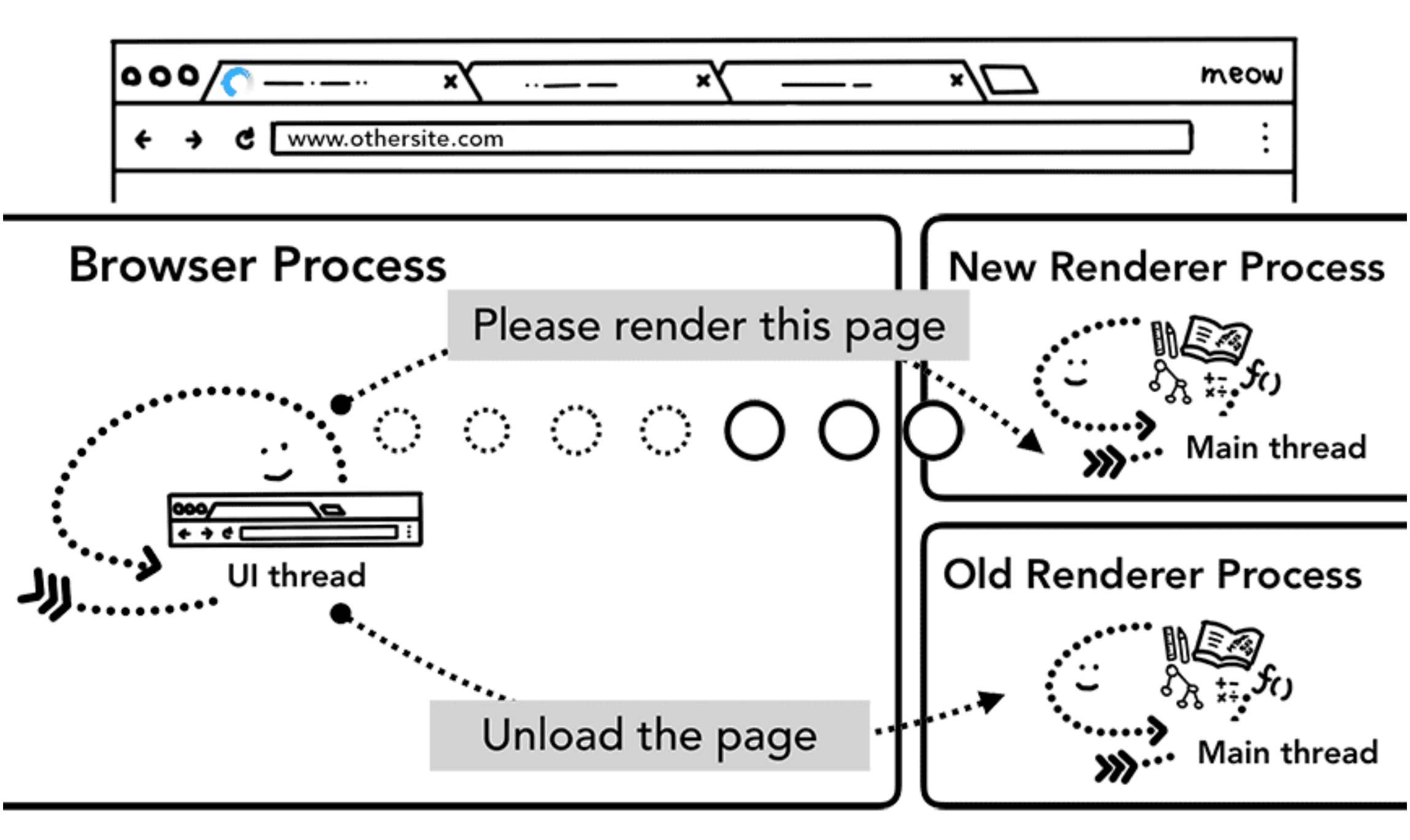

图 9：从浏览器进程到新渲染程序进程的 2 个 IPC，指示渲染网页 并指示旧的渲染程序进程卸载

对于 Service Worker

此导航流程最近的一项更改是引入了 Service Worker。Service Worker 是一种 网络代理让网站开发者能够更好地控制 本地缓存以及何时从网络获取新数据。如果 Service Worker 设置为加载页面 因此无需从网络请求数据。

需要注意的重要一点是，Service Worker 是在渲染程序中运行的 JavaScript 代码。 过程。但是，当导航请求传入时，浏览器进程如何知道网站已有 Service Worker?

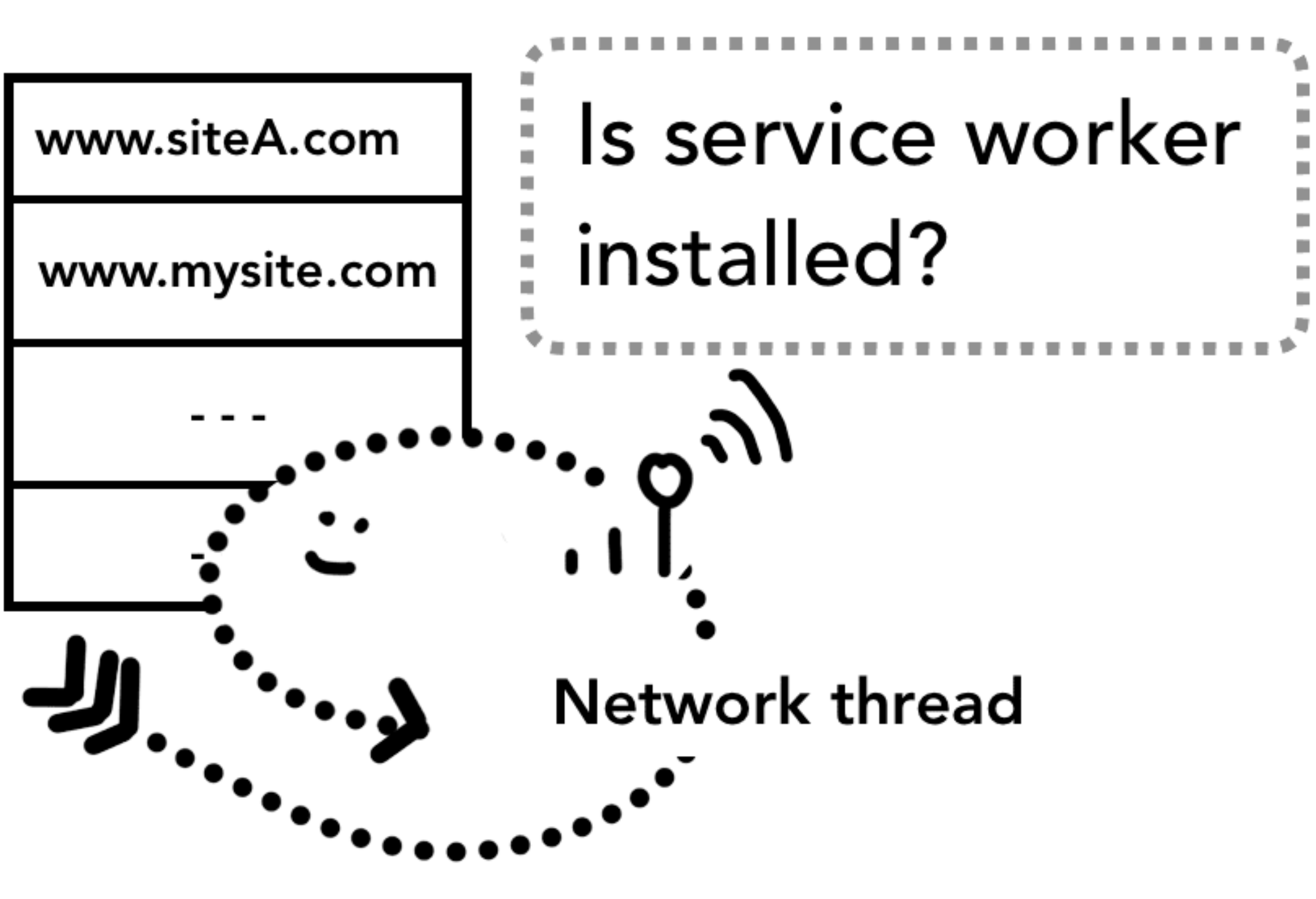

图 10：浏览器进程中的网络线程查找 Service Worker 作用域

注册 Service Worker 时，Service Worker 的作用域将作为引用保留 （如需详细了解范围，请参阅 Service Worker 生命周期一文）。 发生导航时，网络线程会根据已注册的 Service Worker 检查网域 作用域，如果针对该网址注册了 Service Worker，则界面线程会在 来执行 Service Worker 代码Service Worker 可从缓存加载数据， 需要从网络请求数据，或者可能会从网络请求新资源。

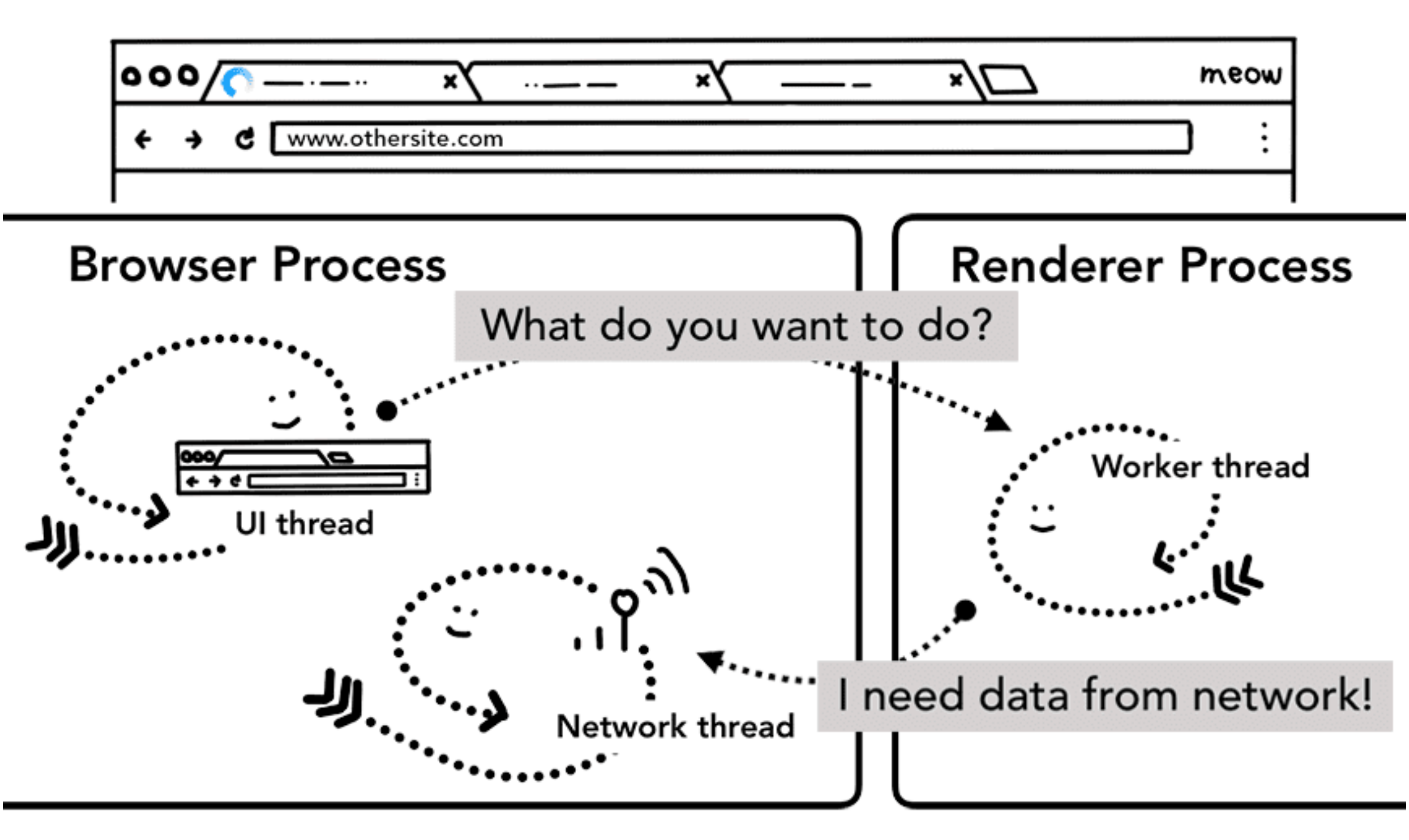

图 11：浏览器进程中的界面线程启动渲染程序进程来处理服务 工作器；然后，渲染器进程中的工作器线程从网络请求数据

导航预加载

您可以看到，浏览器进程与渲染程序进程之间的这种往返可能会导致延迟 如果 Service Worker 最终决定从网络请求数据。 Navigation Preload 是加快加载过程的 在 Service Worker 的启动过程中并行加载资源。 它会使用标头标记这些请求，以便服务器决定为哪些请求发送不同的内容 这类请求；例如，只更新数据，而不是整个文档。

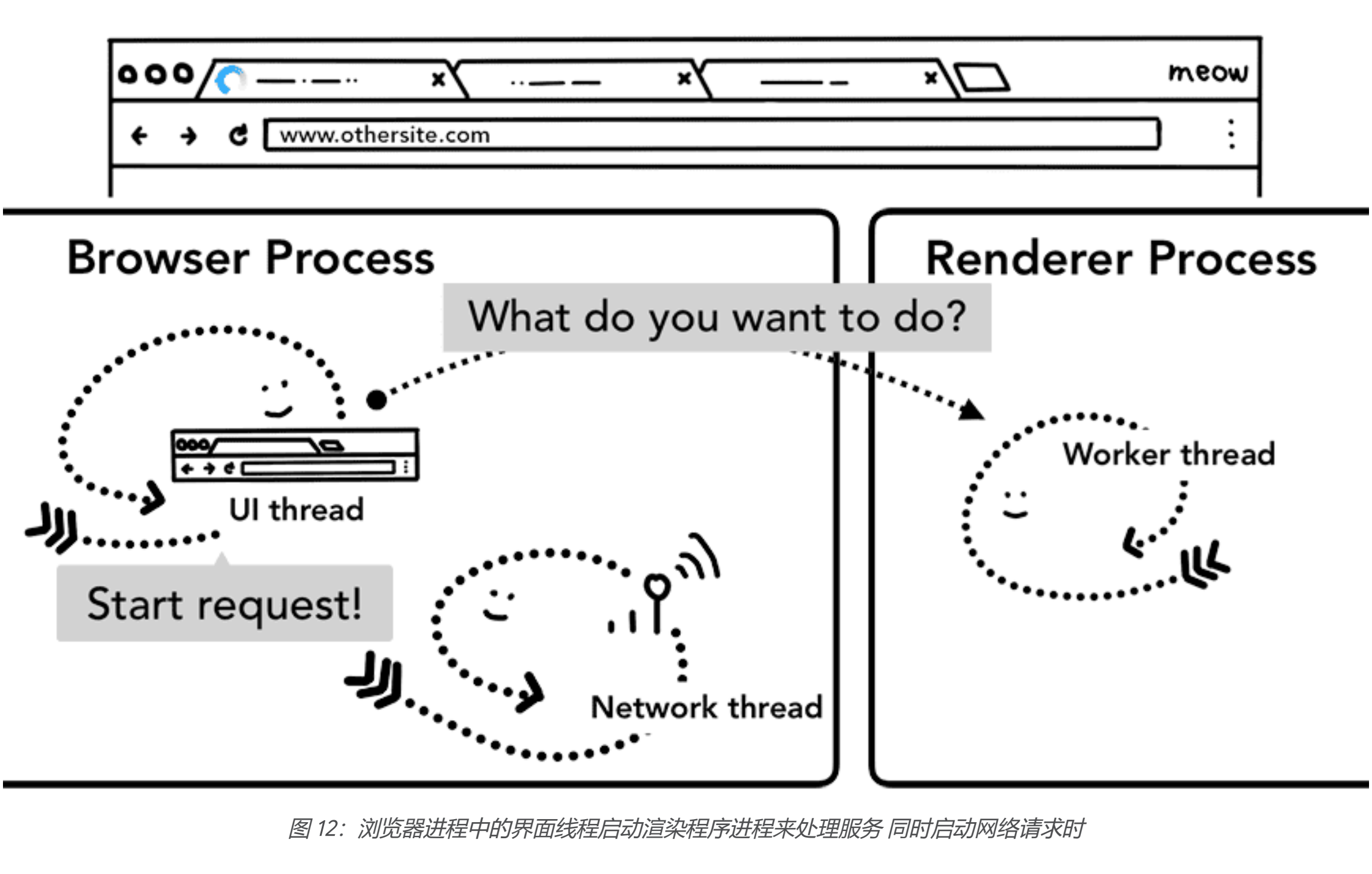

图 12：浏览器进程中的界面线程启动渲染程序进程来处理服务 同时启动网络请求时

小结

在这篇博文中，我们了解了导航期间会发生什么，以及您的 Web 应用代码是如何 响应标头和客户端 JavaScript 与浏览器交互。了解浏览器的步骤 以便从网络获取数据，从而更轻松地理解导航等 API 开发了预加载功能。在下一篇博文中，我们将深入介绍浏览器 HTML/CSS/JavaScript 来呈现网页。

您喜欢这个帖子吗？如果您对以后的帖子有任何疑问或建议， 通过下方的评论部分或 Twitter 上的 @kosamari 分享自己的想法。

下一篇：渲染程序进程的内部工作