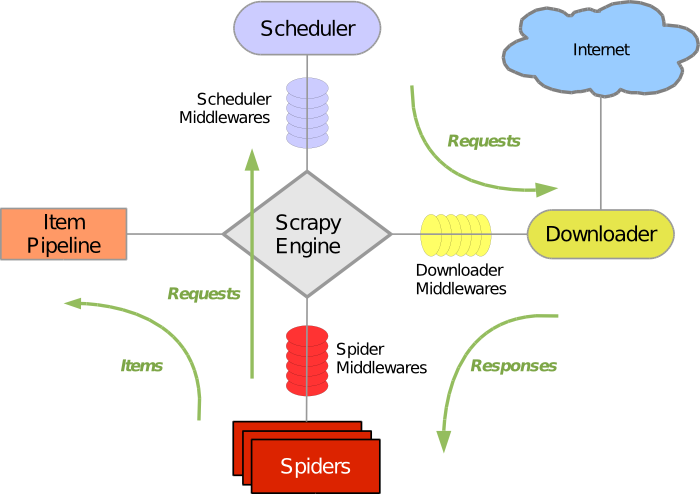
# Architecture overview[¶](http://doc.scrapy.org/topics/architecture.html#architecture-overview)

This document describes the architecture of Scrapy and how its components interact.

## Overview[¶](http://doc.scrapy.org/topics/architecture.html#overview)

The following diagram shows an overview of the Scrapy architecture with its components and an outline of the data flow that takes place inside the system (shown by the green arrows). A brief description of the components is included below with links for more detailed information about them. The data flow is also described below.



## Components[¶](http://doc.scrapy.org/topics/architecture.html#components)

### Scrapy Engine[¶](http://doc.scrapy.org/topics/architecture.html#scrapy-engine)

The engine is responsible for controlling the data flow between all components of the system, and triggering events when certain actions occur. See the Data Flow section below for more details.

### Scheduler[¶](http://doc.scrapy.org/topics/architecture.html#scheduler)

The Scheduler receives requests from the engine and enqueues them for feeding them later (also to the engine) when the engine requests them.

### Downloader[¶](http://doc.scrapy.org/topics/architecture.html#downloader)

The Downloader is responsible for fetching web pages and feeding them to the engine which, in turn, feeds them to the spiders.

### Spiders[¶](http://doc.scrapy.org/topics/architecture.html#spiders)

Spiders are custom classes written by Scrapy users to parse responses and extract items (aka scraped items) from them or additional URLs (requests) to follow. Each spider is able to handle a specific domain (or group of domains). For more information see [Spiders](http://doc.scrapy.org/topics/spiders.html#topics-spiders).

### Item Pipeline[¶](http://doc.scrapy.org/topics/architecture.html#item-pipeline)

The Item Pipeline is responsible for processing the items once they have been extracted (or scraped) by the spiders. Typical tasks include cleansing, validation and persistence (like storing the item in a database). For more information see [Item Pipeline](http://doc.scrapy.org/topics/item-pipeline.html#topics-item-pipeline).

### Downloader middlewares[¶](http://doc.scrapy.org/topics/architecture.html#downloader-middlewares)

Downloader middlewares are specific hooks that sit between the Engine and the Downloader and process requests when they pass from the Engine to the Downloader, and responses that pass from Downloader to the Engine. They provide a convenient mechanism for extending Scrapy functionality by plugging custom code. For more information see [Downloader Middleware](http://doc.scrapy.org/topics/downloader-middleware.html#topics-downloader-middleware).

### Spider middlewares[¶](http://doc.scrapy.org/topics/architecture.html#spider-middlewares)

Spider middlewares are specific hooks that sit between the Engine and the Spiders and are able to process spider input (responses) and output (items and requests). They provide a convenient mechanism for extending Scrapy functionality by plugging custom code. For more information see [Spider Middleware](http://doc.scrapy.org/topics/spider-middleware.html#topics-spider-middleware).

### Scheduler middlewares[¶](http://doc.scrapy.org/topics/architecture.html#scheduler-middlewares)

Scheduler middlewares are specific hooks that sit between the Engine and the Scheduler and process requests when they pass from the Engine to the Scheduler and vice-versa. They provide a convenient mechanism for extending Scrapy functionality by plugging custom code.

## Data flow[¶](http://doc.scrapy.org/topics/architecture.html#data-flow)

The data flow in Scrapy is controlled by the Engine, and goes like this:

1. The Engine opens a domain, locates the Spider that handles that domain, and asks the spider for the first URLs to crawl.
2. The Engine gets the first URLs to crawl from the Spider and schedules them in the Scheduler, as Requests.
3. The Engine asks the Scheduler for the next URLs to crawl.
4. The Scheduler returns the next URLs to crawl to the Engine and the Engine sends them to the Downloader, passing through the Downloader Middleware (request direction).
5. Once the page finishes downloading the Downloader generates a Response (with that page) and sends it to the Engine, passing through the Downloader Middleware (response direction).
6. The Engine receives the Response from the Downloader and sends it to the Spider for processing, passing through the Spider Middleware (input direction).
7. The Spider processes the Response and returns scraped Items and new Requests (to follow) to the Engine.
8. The Engine sends scraped Items (returned by the Spider) to the Item Pipeline and Requests (returned by spider) to the Scheduler
9. The process repeats (from step 2) until there are no more requests from the Scheduler, and the Engine closes the domain.

## Event-driven networking[¶](http://doc.scrapy.org/topics/architecture.html#event-driven-networking)

Scrapy is written with [Twisted](http://twistedmatrix.com/trac/), a popular event-driven networking framework for Python. Thus, it’s implemented using a non-blocking (aka asynchronous) code for concurrency.

For more information about asynchronous programming and Twisted see these links:

* [Asynchronous Programming with Twisted](http://twistedmatrix.com/projects/core/documentation/howto/async.html)
* [Twisted - hello, asynchronous programming](http://jessenoller.com/2009/02/11/twisted-hello-asynchronous-programming/)