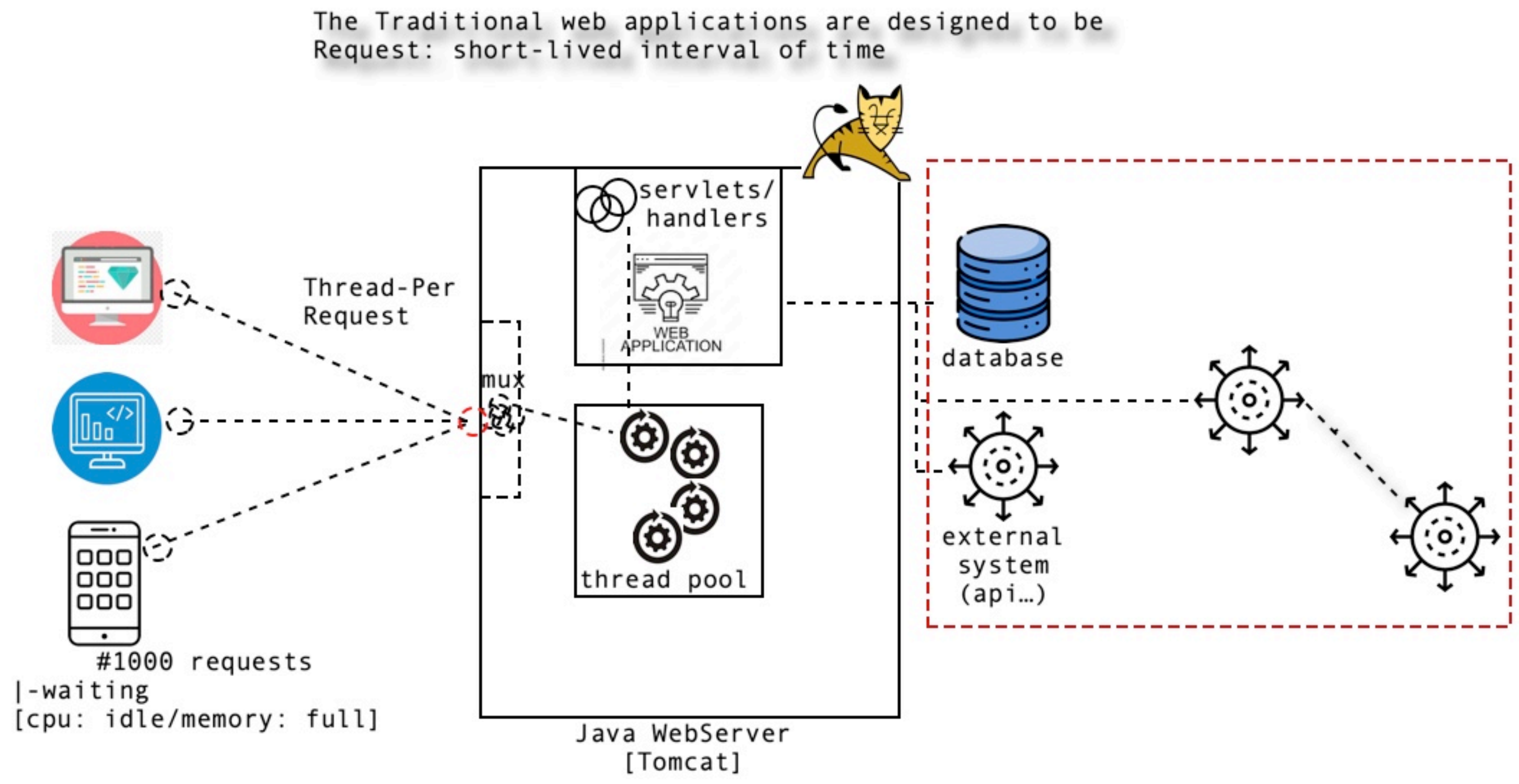
Traditional Way
Traditional way of handling the request/responses by a typical Java WebServer as below.



When a request comes in, the Webserver assigns a thread from the worker pool, asking to handle the request and dispatch the response to the client. The thread will be occupied with request until it finishes the processing.

This is called “Request per Thread” model, where all the traditional or gen#1 Java WebServers works based on this model in handling the request/response.

This model works well with traditional web applications where a request spans for shorter-interval of time. But in the modern world, the HTTP/Web standards are not being used only for building web applications, we build enterprise distribute component solutions by leveraging HTTP standards.

Usually these complex business components interacts with external resources or other apis/systems. They might talk to multiple other systems or services, aggregates the data, performs processing and builds the output. This might take long interval of time, which means the Http Request towards an application takes huge amount of time (or requests are long-lived)

So when the webservers handles such requests based on “Request-per-Thread” model, there are lot of waiting threads holding the precious system resources for getting the response from the application processes (Servlets/Handlers), which intern are waiting for response from external systems.

Due to which there are longer number of idle threads blocking the system resources, thus by all the new requests that are coming to the application will be kept waiting as there are no enough computing resources are available, this leads to thread-starvation and crash of the application.

Inorder to address this problem, we need to scale-out the application where the cost of scaling the application is very high as it creates an virtual load on the servers.

In a typical web server like an embedded tomcat server the worker thread pool size: 200, which means concurrently we can accomodate 200 max requests. If all the threads are blocked/waiting for the process to complete execution, inorder to scale the application we can increase these worker pool size.
But the problem each of these threads will be occuping system resources like cpu/ram and may not have any resources left for the jvm to perform operation. So increasing the worker pool size is not the right solution.

As a solution to the above problems, a team of developers, led by Jonas Boner came together and introduced a new programming paradigm, which is called as “Reactive Programming” has been founded.