Fill in the empty cells of the comparison table.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Multithreaded**  **(either dynamic threads or**  **using a pool of worker threads)** | **Event-driven** |
| Number of threads | multiple | one |
| Memory usage | in general, worse | in general, better |
| Location where application-level state is stored for a blocked action | Individual Thread Stack + TCB | “continuation” data structure  (i.e., current state for the action plus a pointer to the next code to run) |
| Method of storing/restoring application-level state | automatically, as part of  thread switch | Manual; must be explicitly programmed |
| Type of I/O | synchronous (i.e., blocking) | Asynchronous I/O |
| Exploits parallelism available between a processor and I/O devices? | yes | yes |
| Exploits parallelism available among multiple processors? | Yes | No (Only has 1 thread) |
| Convenient framework for a mix of foreground and background tasks? | Yes | No (see above) |
| Is synchronization required when updating data shared between threads? | Yes | No (see above) |
| Example | Apache | Node.js  (from the JavaScript perspective) |

Note: in practice, many system designs are a mix of each approach.