

Event Handling Patterns

Reactor, Proactor

Kryštof Hrubý



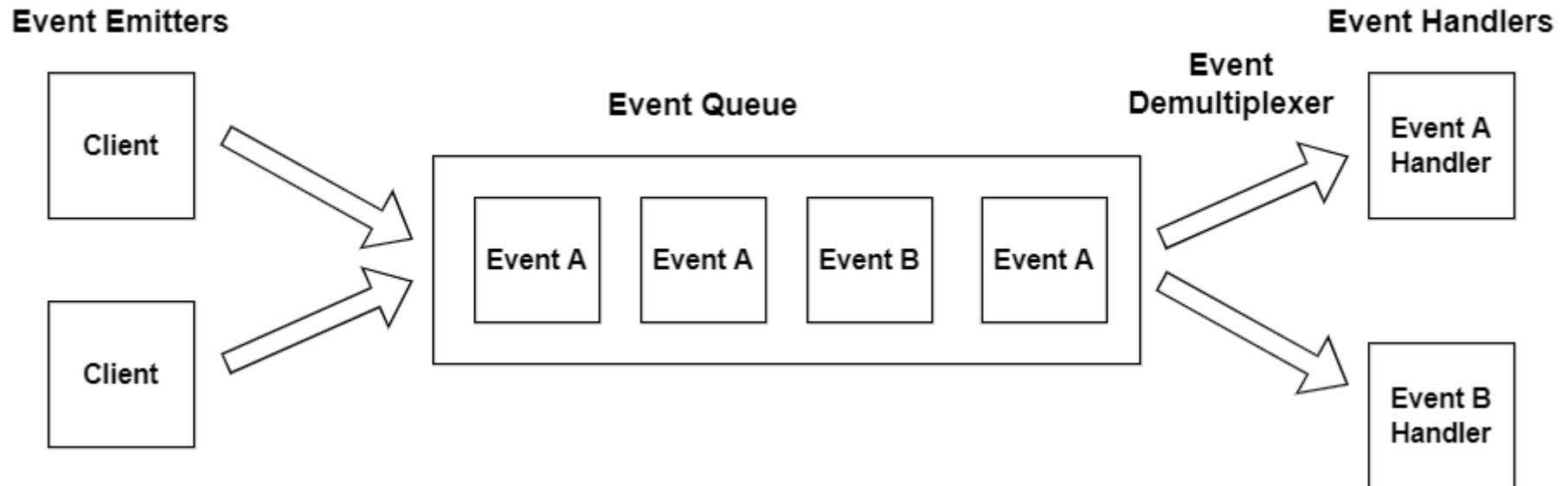
Osnova

- **Event Handling Patterns.**
- **Motivační příklad – Logovací server.**
- **Logovací server pomocí Reactor pattern.**
- **Reactor.**
- **Logovací serveru pomocí Proactoru.**
- **Proactor.**



Event Handling Patterns

- Popisují způsob, jakým iniciovat, přijmout, demultiplexovat, dispatchovat a zpracovat události (eventy).

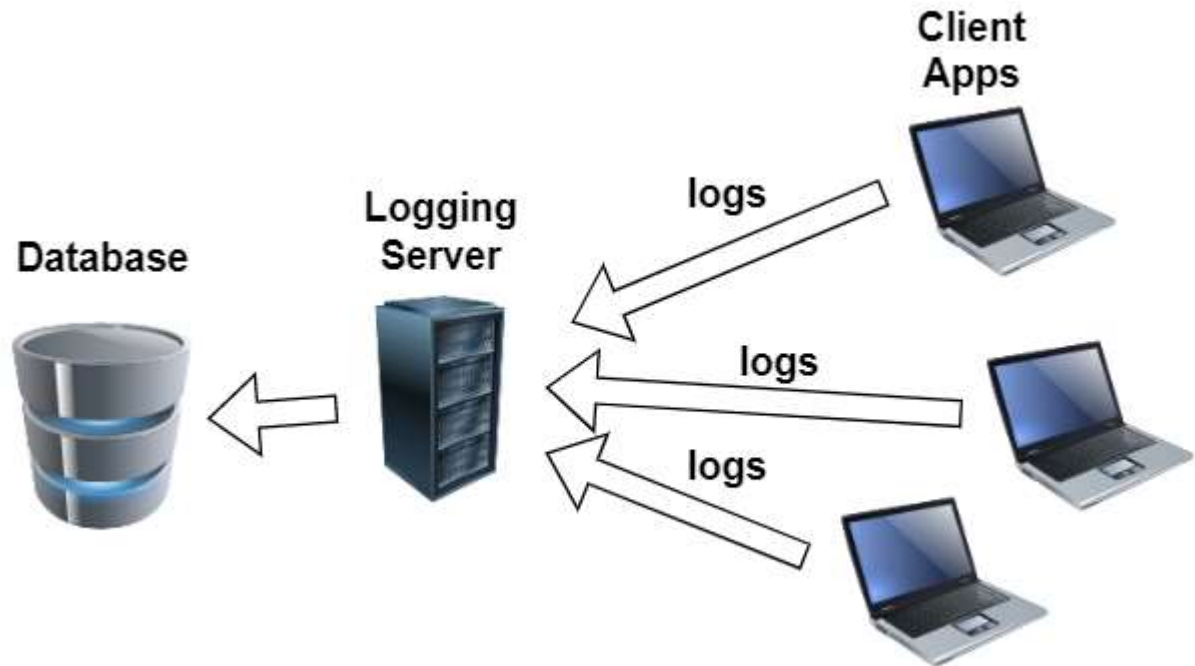




Logovací služba

■ Distribuovaná logovací služba

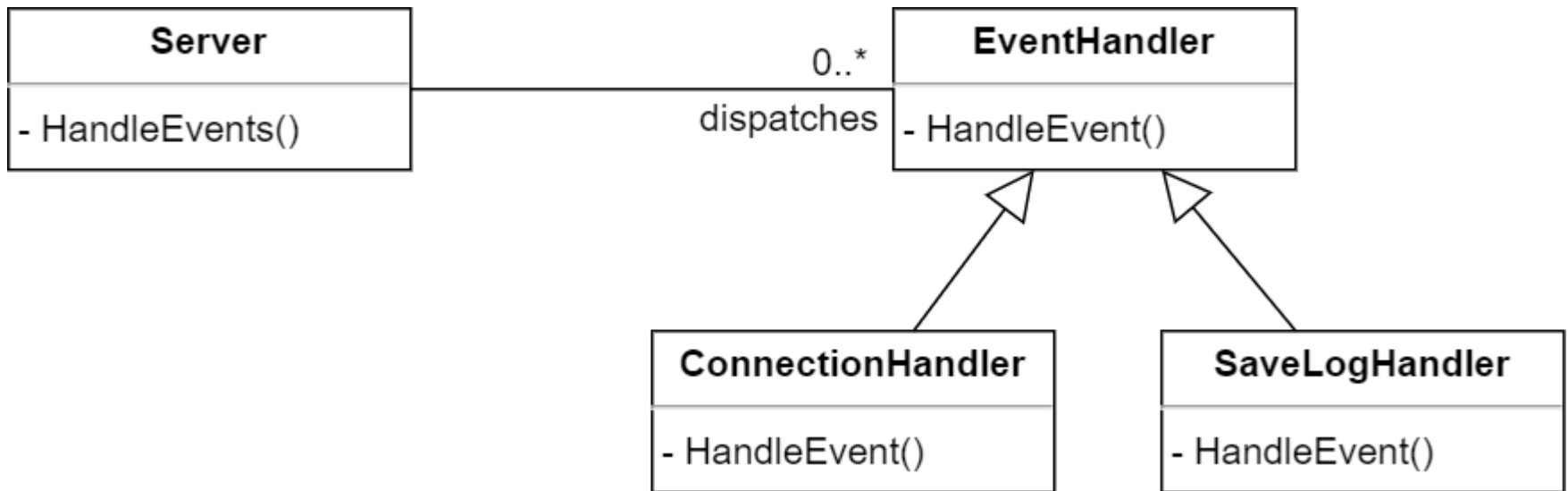
- Klienti posílají informace o jejich stavu na logovací server.
- Od klientů vznikají dva typy událostí:
 - Connect – Nový klient žádá o připojení.
 - Read – Klient má k dispozici nové položky do logu.
- Události mohou chodit najednou.
- Obsluha události má krátké trvání.





Logovací služba – jednovláknové řešení

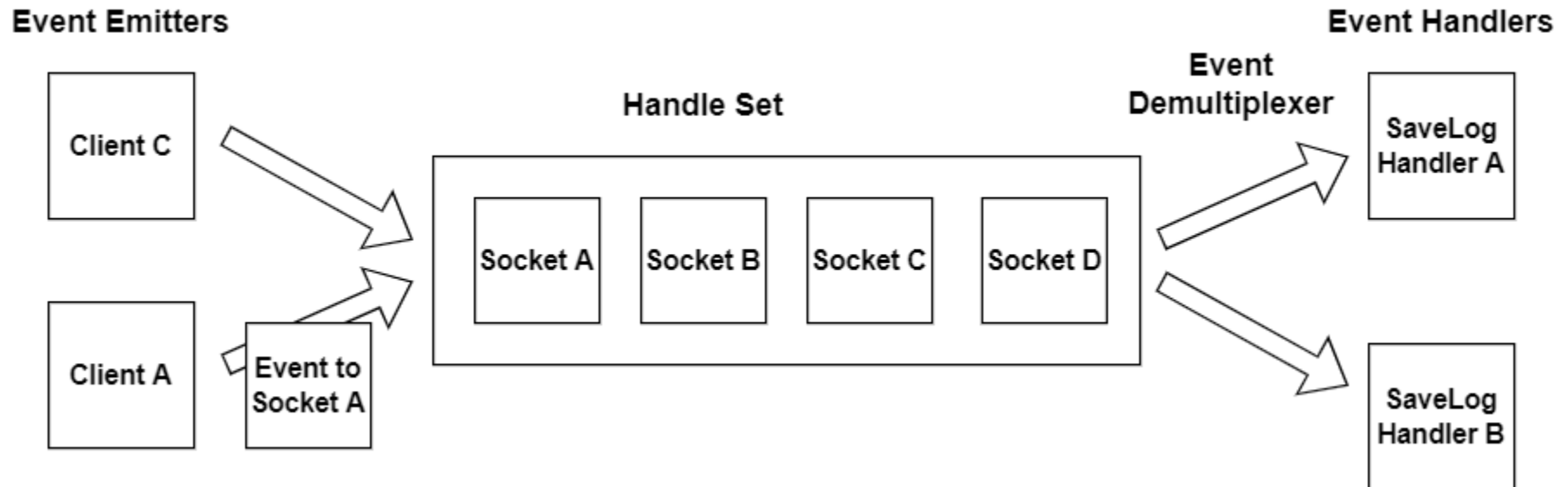
- **Server čte v jednom vlákně všechny requesty.**
- **Aplikační logika je forwardována na Handlers.**
- **Výhody:**
 - Oddělena aplikační logika a logika zpracovávání událostí.
- **Nevýhody:**
 - Pomalé - Veškeré zpracování se děje v jednom vlákně.





Logovací služba – myšlenka čekání

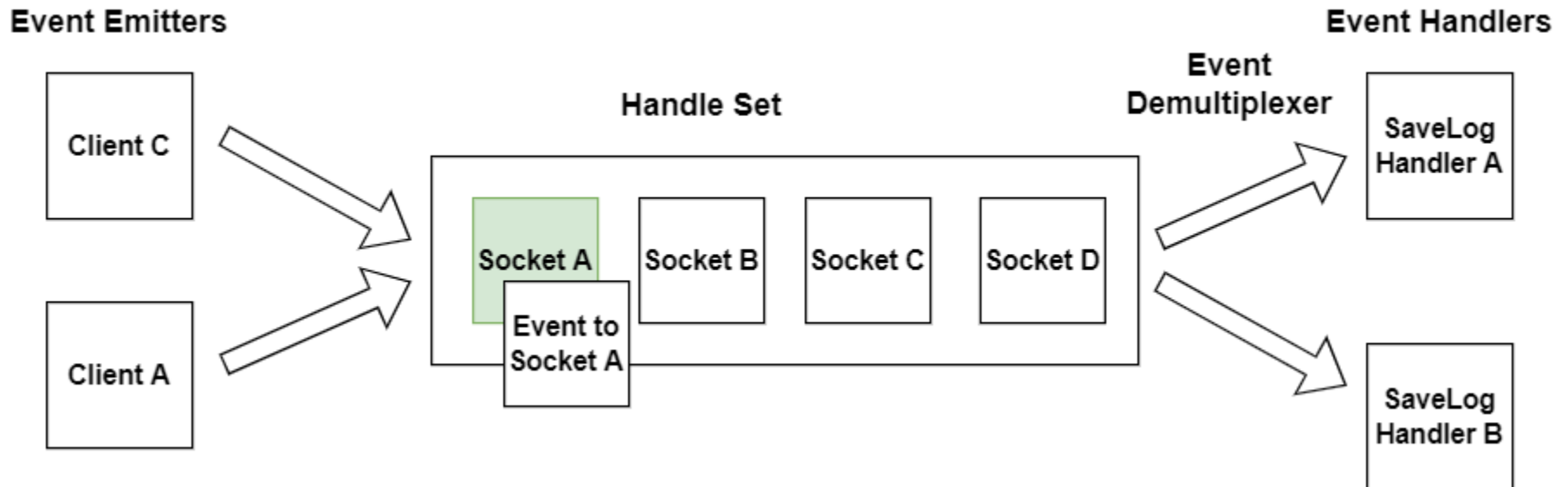
- Chceme efektivně čekat na nové eventy.
- Chceme být probuzeni, když nám přijde událost na jeden z použitých socketů.





Logovací služba – myšlenka čekání

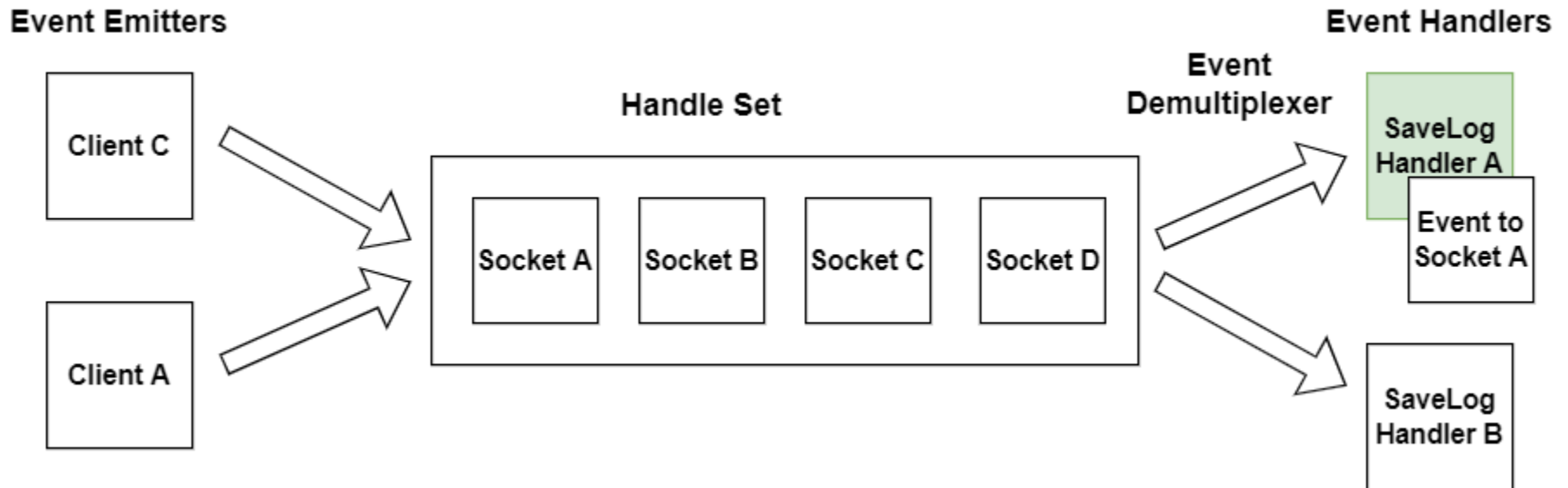
- Chceme efektivně čekat na nové eventy.
- Chceme být probuzeni, když nám přijde událost na jeden z použitých socketů.





Logovací služba – myšlenka čekání

- Chceme efektivně čekat na nové eventy.
- Chceme být probuzeni, když nám přijde událost na jeden z použitých socketů.





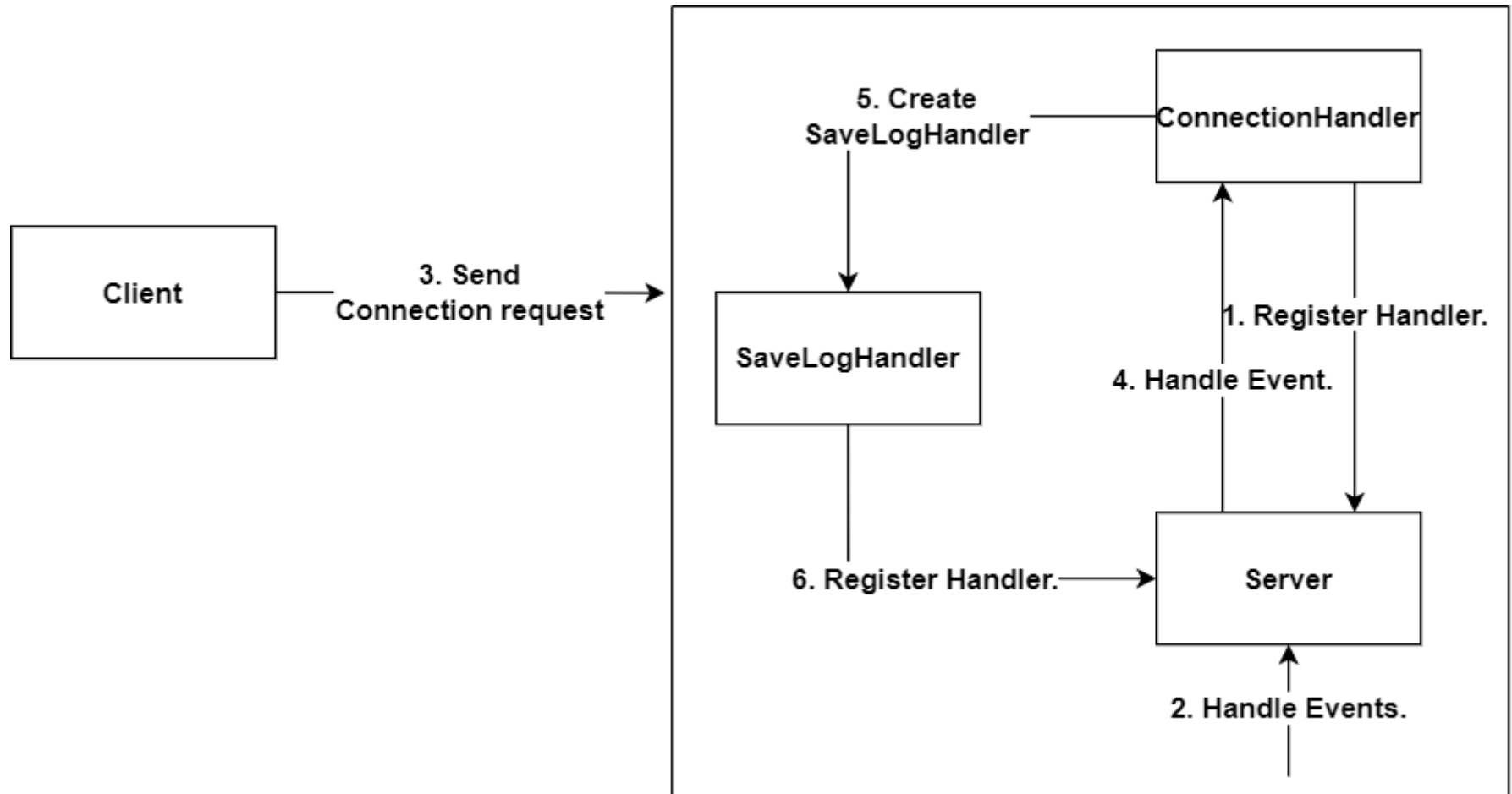
Logovací služba – vícevláknové řešení

- **Server pro každého klienta (connection) vytváří vlákno.**
- **Vlákno pak na přiděleném socketu zpracovává requesty.**
- **Výhody:**
 - Paralelní čekání na události.
- **Nevýhody:**
 - Neefektivní a neškálovatelné kvůli režii přepínání vláken a synchronizace.
 - Zpracovávání requestů není férové.



Reactor - workflow

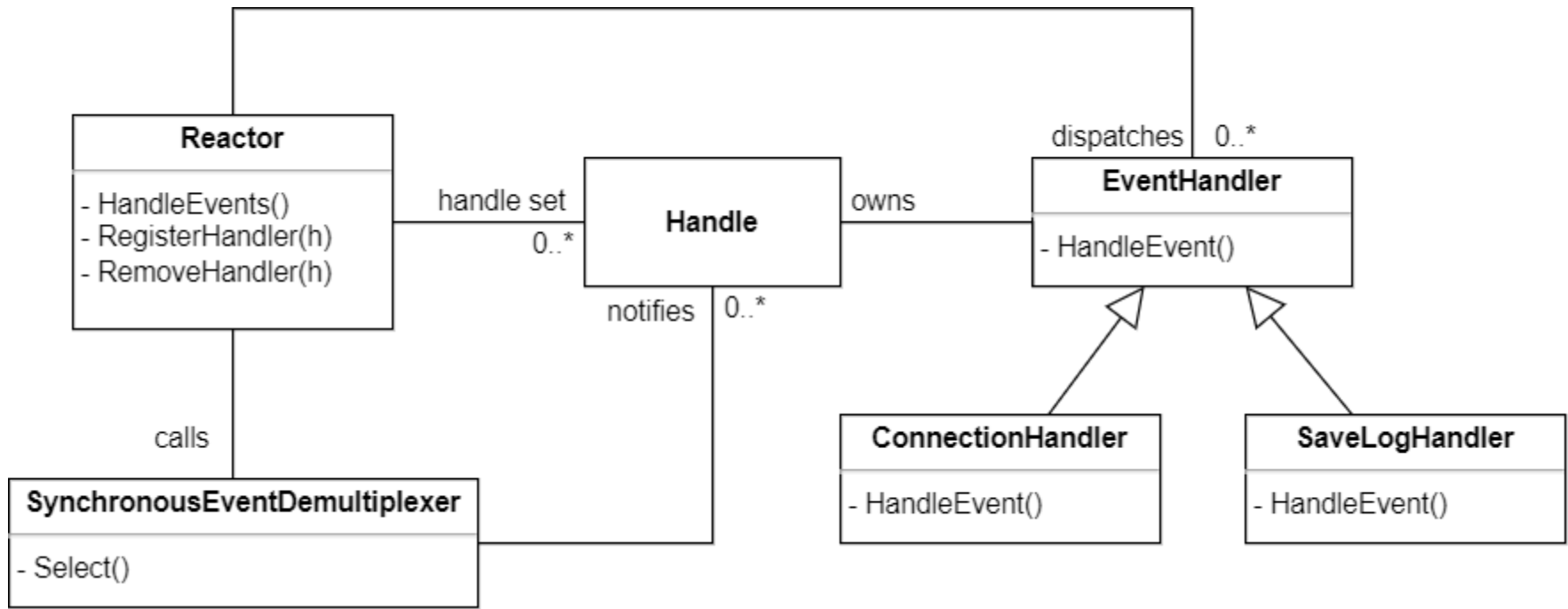
- Příklad připojení klienta k serveru – Connection událost.





Reactor

- Každý EventHandler má svůj Handle (socket).
- OS má funkci pro detekci, zda nějaký Handle je aktivní pro zpracování.
 - SynchronousEventDemultiplexer.
 - Demultiplexer blokuje a budí se při aktivním Handlu v rámci Handle set Reactoru.





Reactor – varianty

- **Event Handlery pracují ve vlastních vláknech.**
- **Program má tolik Reactorů, kolik má CPU jader.**



Reactor – použití

■ Reálné implementace:

- ❑ Reactor (Java)
- ❑ Twisted (Python)
- ❑ Nodejs I/O
- ❑ Nginx
- ❑ UI



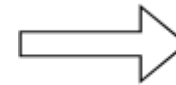
Reactor

■ Výhody:

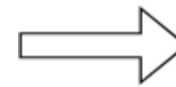
- ❑ Rozdělení zodpovědností mezi Reactor a Aplikaci.
 - ❑ Umožňuje modularitu, znovupoužitelnost, přenositelnost.
 - ❑ Aplikace implementuje pouze obsluhu.
- ❑ Umožňuje obsloužit více současných spojení bez režie více vláken.
 - ❑ Ale jen na principu serializace volání jednotlivých Event Handlerů.

■ Nevýhody:

- ❑ Event Handlery nemůžou dělat moc práce.
 - ❑ Jakékoliv blokující volání v obsluze zablokuje celý proces.
- ❑ Pro efektivní implementaci je třeba podpora OS.
- ❑ Zpracování příchozích událostí je v synchronní.
- ❑ Náročné ladění a testování.



Proactor



Proactor



Proactor

■ Potřebujeme OS podporu asynchronních operací.

- ❑ Asynchronous Operation Processor.
- ❑ OS výsledky async operací (event) dává do fronty (Event Queue).

■ Rozdělení aplikačních služeb na dvě části

- ❑ Asynchronní (můžou být dlouho trvající) operace, které jsou volány proaktivně.
- ❑ Completion Handlery – stará se o zpracování výsledku asynchronní operace.

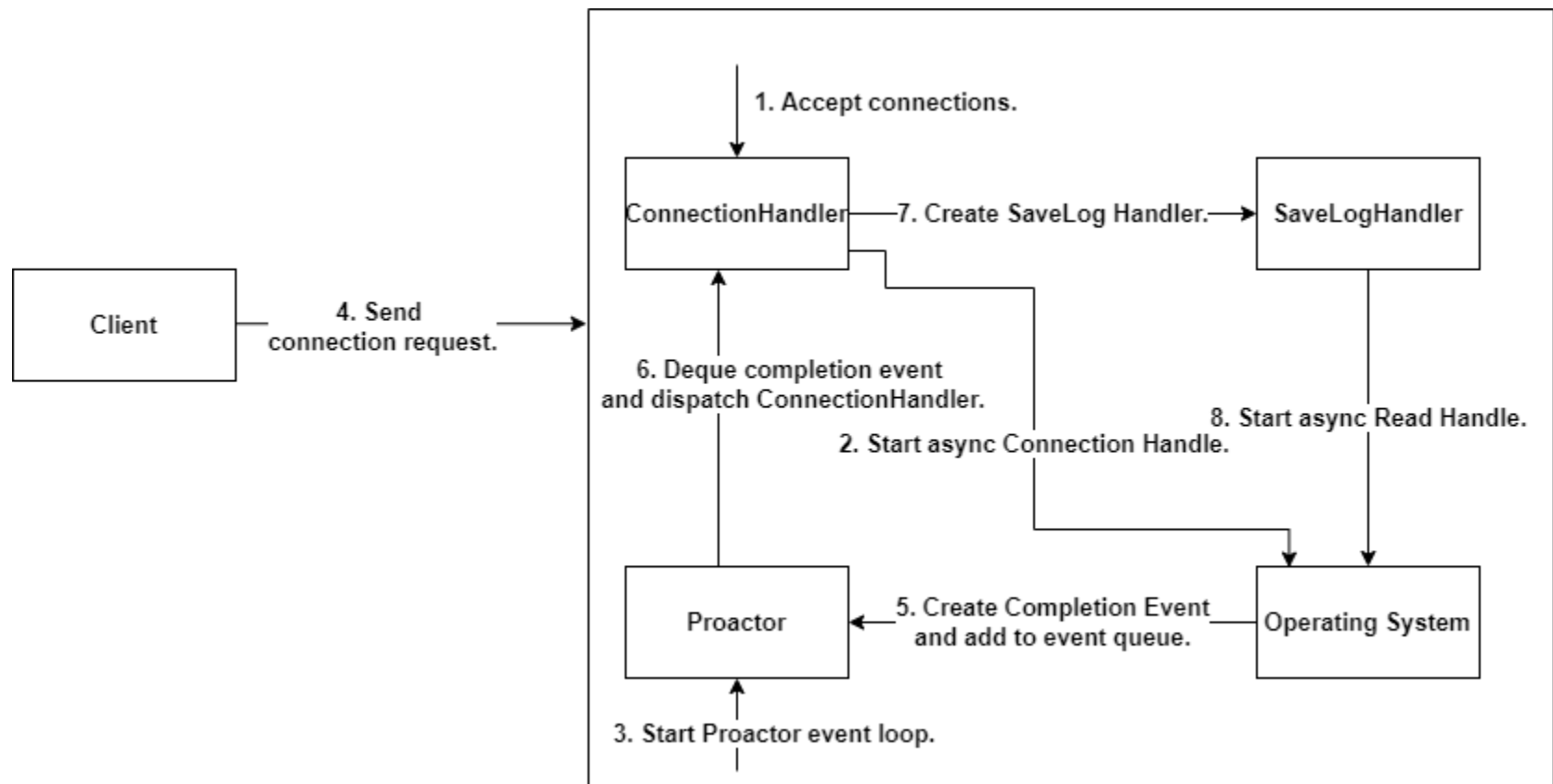
■ Completion Handler

- ❑ Stará se o zpracování výsledku asynchronní operace – Completion Event.



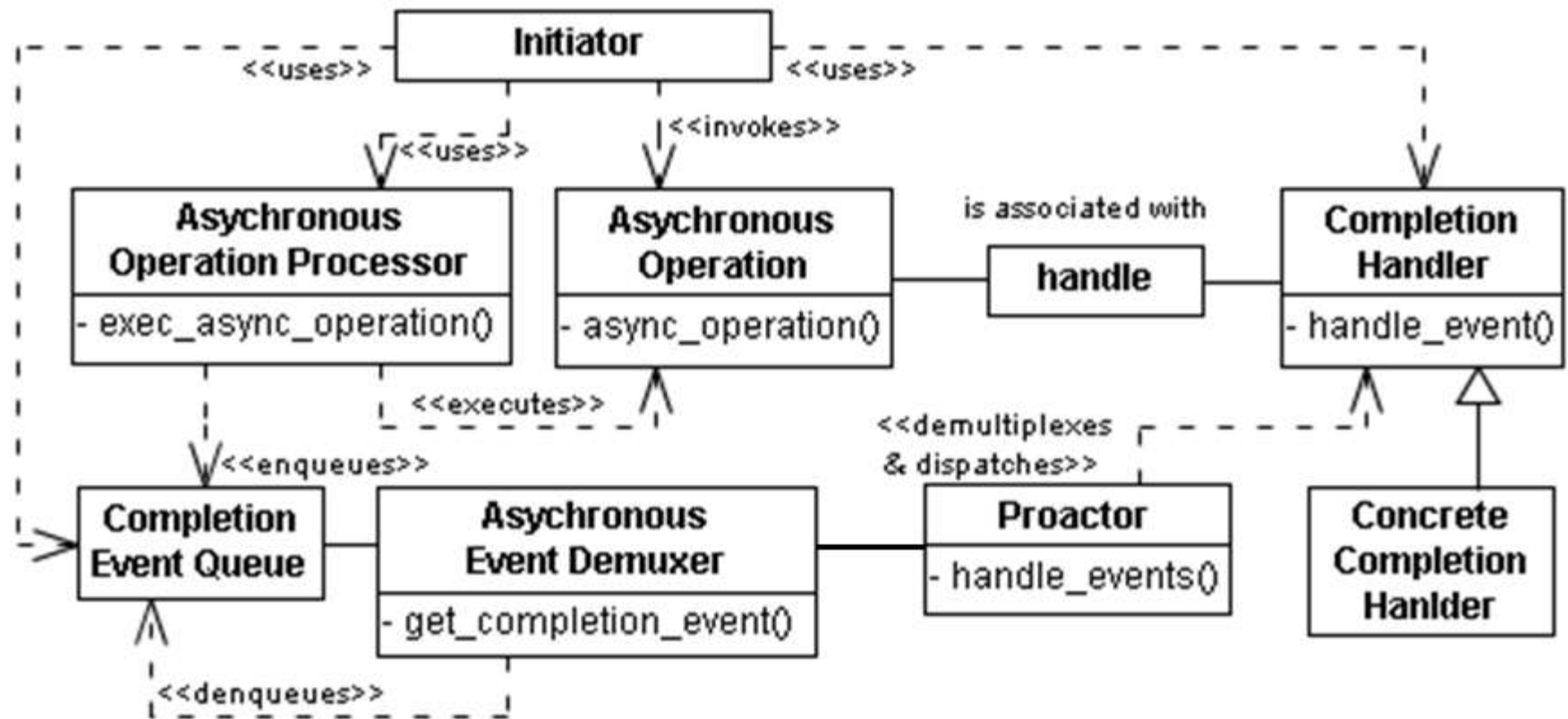
Proactor - workflow

■ Příklad připojení klienta k serveru – Connection událost.





Proactor





Proactor – varianty

- **Asynchronní Completion Handler.**

- Jsou to iniciátory async operací.

- **Vícevláknový asynchronní Demultiplexer.**

- Bazének vláken sdílející Asynchronous Event Demultiplexer.



Proactor – použití

- **Reálné implementace:**

- C++: Boost.Asio



Proactor

■ Výhody:

- Rozdělení zodpovědností mezi Proactor a Aplikaci.
 - Umožňuje modularitu, znovupoužitelnost, přenositelnost.
 - Aplikace implementuje pouze obsluhu.
- Umožňuje obsloužit více současných spojení bez režie více vláken.
- Performance.
- Jednodušší synchronizace.

■ Nevýhody:

- Nemáme kontrolu, jak přesně se plánují asynchronní operace.
- Pro efektivní implementaci je třeba podpora OS.
- Náročné ladění a testování.



Proactor vs Reactor

■ Společná myšlenka

- Snaha o nahrazení multithreadingu v event-driven systémech.

■ Reactor

- Čeká na informaci, že může vykonat nějakou operaci bez blokování, a pak ji synchronně vykoná.
- Samotná obsluha události pak probíhá synchronně v rámci programu.

■ Proactor

- Asynchronně vykonává operace a čeká na informaci, že tyto operace byly dokončeny.
- Vyžaduje o dost pokročilejší podporu operačního systému.



Děkuji za pozornost.