3. Event Handling Patterns

Event-handling patterns

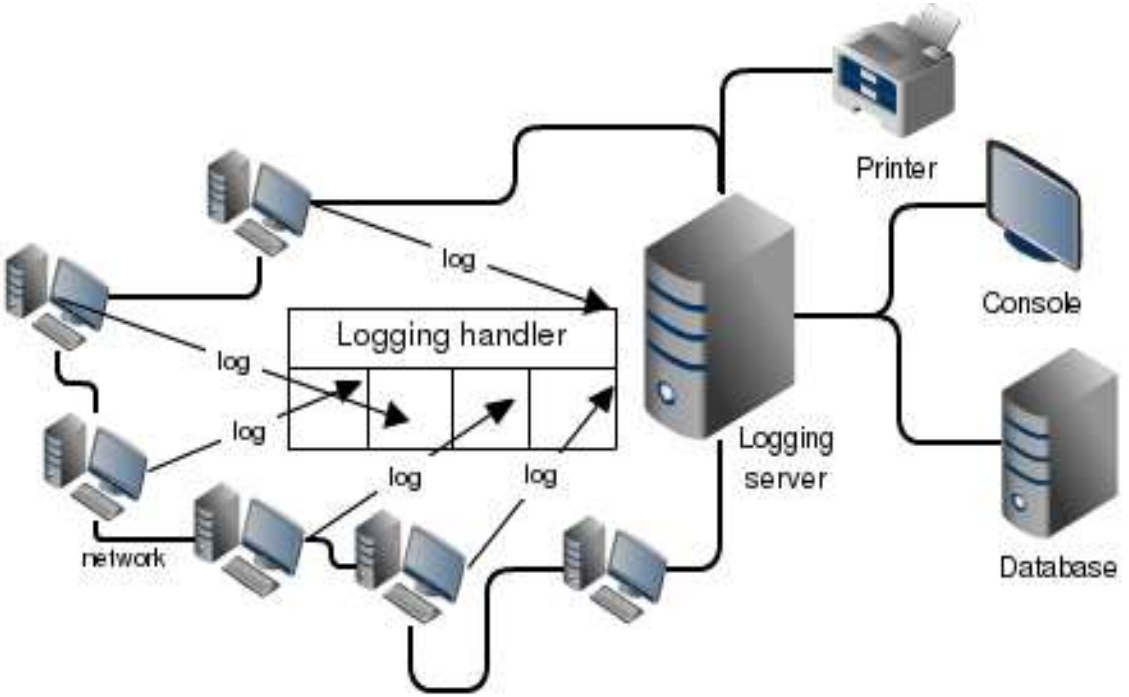
- **Naše zaměření:**
  - Síťové orientované systémy

- **Návrhové vzory**
  - Reactor
  - Proactor
  - Asynchronous Completion Token
  - Acceptor-Connector

- **Popisují způsob, jakým iniciovat, přijmout, demultiplexovat, dispatchovat a zpracovat události (eventy) v síťově orientovaném systému**

Reactor – Motivační příklad

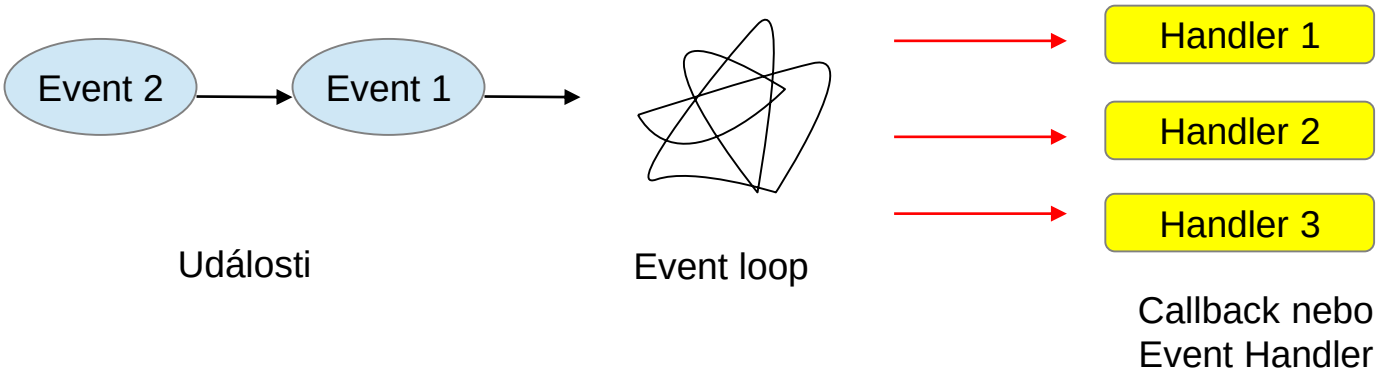
- **Distribuovaná logovací služba**
  - klienti posílají logy na centrální logovací server
  - Pouze dvě události
    - connect - připojení klienta
    - read – přečti logovací záznam
  - simultánní příchod události
  - obsluha má krátké trvání



Rešení

- **Pro každé připojení vytvořit nové vlákno**
  - nutno zabezpečit thread-safe
  - nelze měnit kontext
  - již při pár tisících požadavků neúnosná režie
  - vlákna nejsou podporována na všech systémech

- **Reactor**
  - single thread
  - simultánní zpracování událostí
  - event demultiplexer



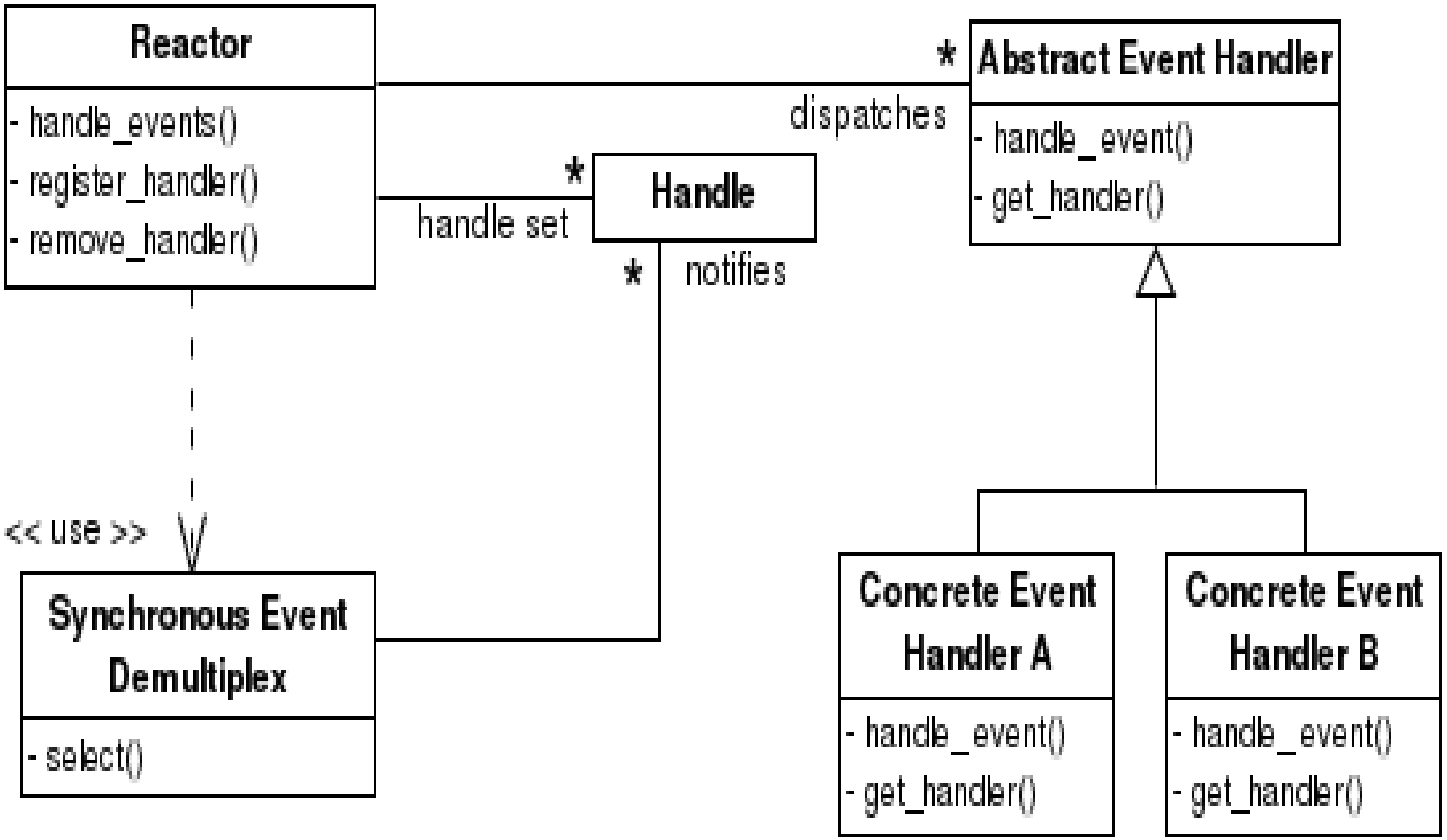
Reactor

- **Reactor řeší rozdělení (demultiplexing) a následné zpracování (dispatching) příchozích eventů, aplikace jen poskytuje event handlery, které obsluhují tyto události**
- **Využívá vzor Inversion of Control (Hollywoodský princip - “Don't call us, we'll call you”)**
- **Základní vlastnosti**
  - čeká synchronně na událost, když nastane událost, aktivuje handlery
  - události jsou typicky z různých zdrojů
  - single thread
- **Také známý jako Dispatcher nebo Notifier**

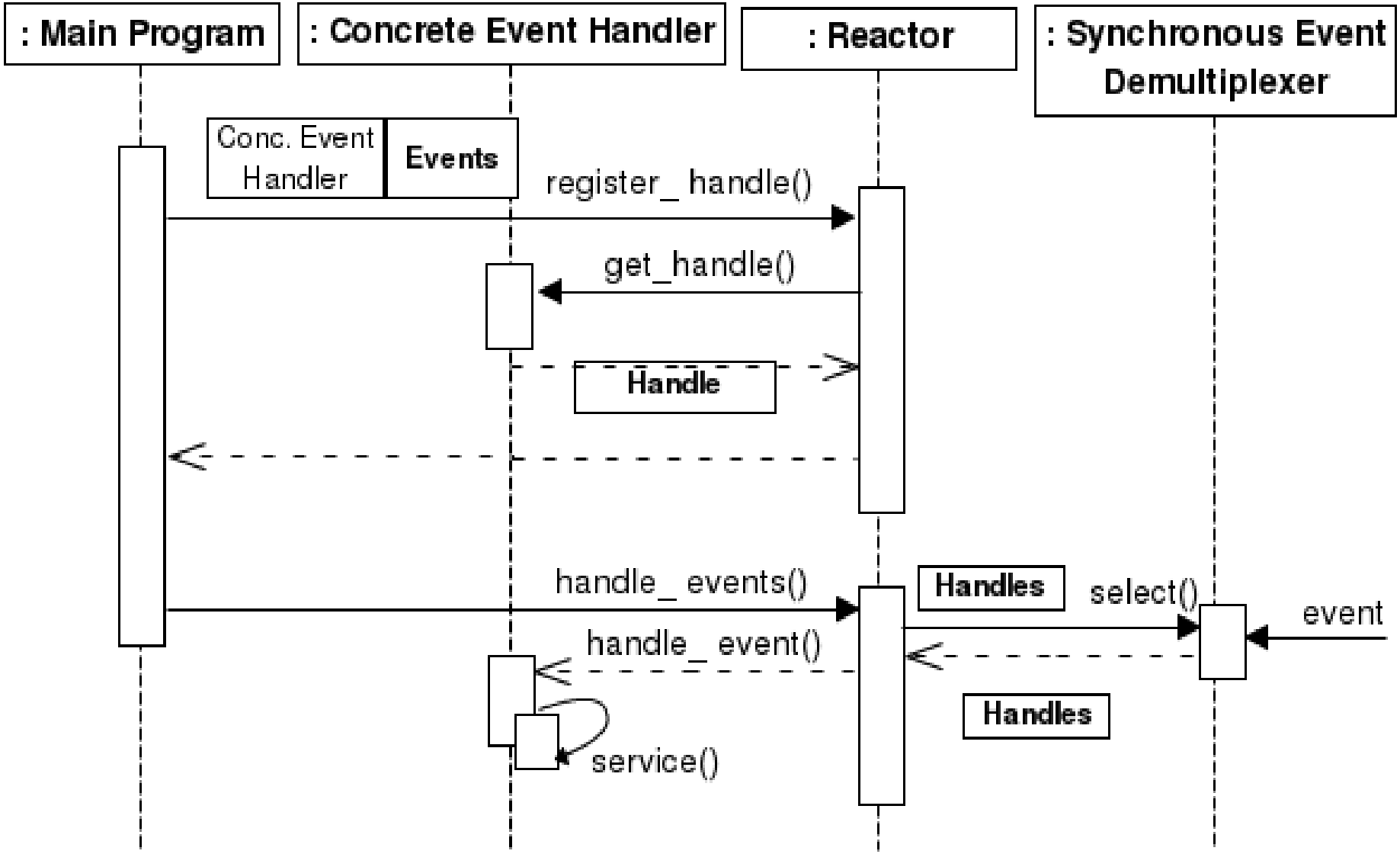
Reactor - struktura

- **Handle**
  - libovolný zdroj (řízený OS) schopný vstupu a/nebo výstupu
  - obvykle síťová připojení, soubory
- **Synchronous Event Demultiplexer**
  - event loop
  - blokuje zdroje
  - jakmile je možné začít synchronní operaci na handli bez blokování, pošle handle na dispatcher
- **Dispatcher**
  - udržuje si tabulku registrovaných handlerů
  - odešle handle z demultiplexoru na asociovaný event handler
- **Event handler**
  - komponenta aplikace implementující zpracování daného eventu
  - aplikace využívá demultiplexing a dispatching poskytnutý reactorem

Reactor – struktura diagram



Reactor – časový diagram



Reactor

- **Výhody**
  - Rozdělení zodpovědností mezi reactor a aplikaci
    - umožňuje modularitu, znovupoužitelnost, přenositelnost
  - Umožnuje obsloužit více současných spojení bez režie multiple threads
    - ale jen na principu serializace volání jednotlivých event handlerů

- **Nevýhody**
  - Nepreempetivní - při implementaci je dobře vyvarovat se
    - cyklení nad velikými kolekcemi objektů
    - výpočetně náročným operacím
    - voláni synchronních I/O metod
    - sleep / pause výrazy
  - Komplexní ladění, testování

Reactor – Příklady, použití

- **Příklad - telefonní linka**
  - telefonní síť je reactor.
  - Vy jste event handler registrovaný telefonním číslem (handle).
  - pokud se Vám někdo pokouší dovolat, telefonní síť Vás na hovor upozorní zvoněním a Vy tuto událost zpracujete například tím, že zvednete telefon

- **Reálné implementace**
  - Reactor (Java), Twisted (Python), ReactPHP (PHP), Xt (UI toolkit v UNIXu)

Proactor – Motivační příklad

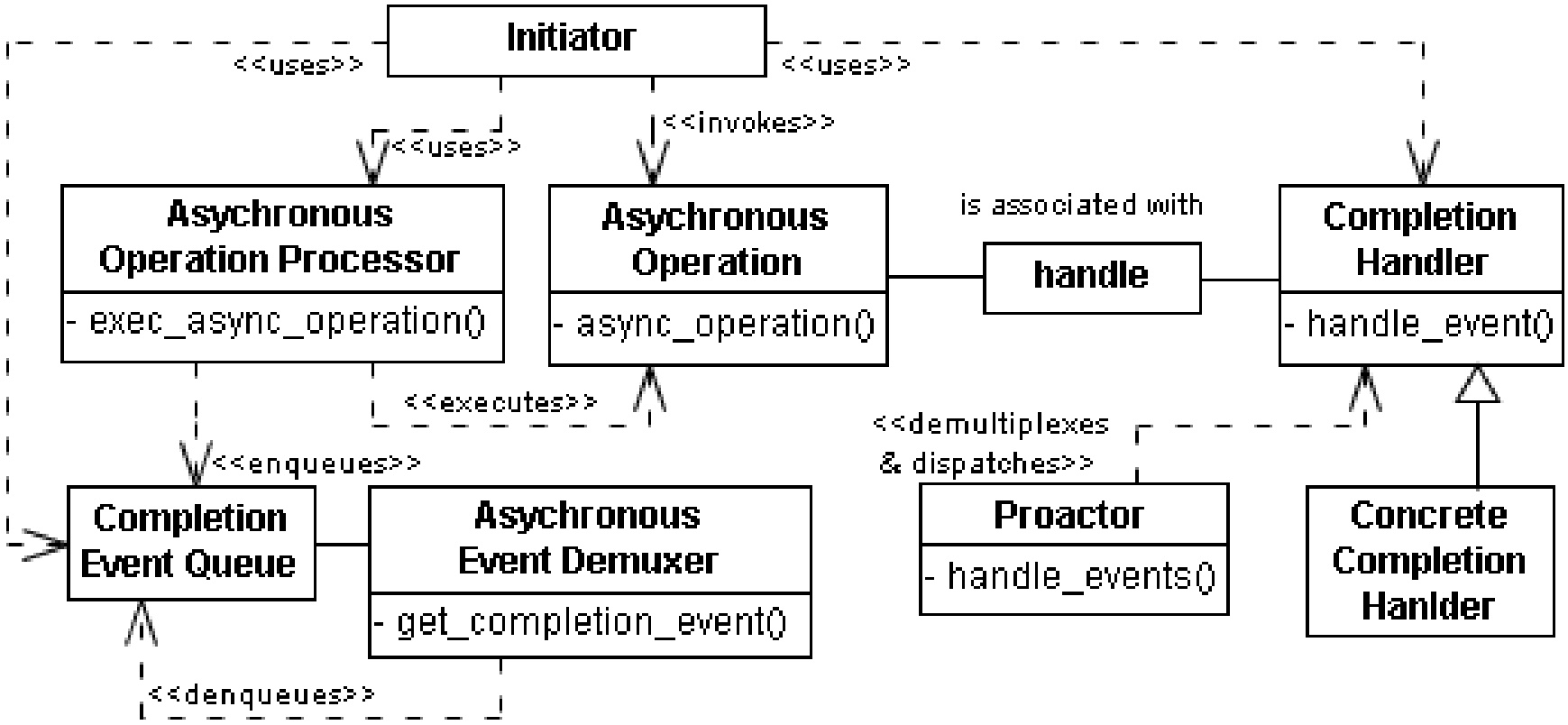
- **Web server**
  - klient zašle požadavek na spojení
  - server spojení akceptuje
  - server otevře a čte příslušný soubor
  - nakonec pošle server obsah souboru zpět klientovi

- **Možné implementace**
  - Synchronní multi-threading
    - problémy se škálovatelností, komplexitou kódu a přenositelností
  - Reactor
    - nepodporuje zpracování více eventů zároveň
    - nepodporuje dlouhé zpracovaní klientských požadavků
  - Proactor
    - asynchronní varianta Reactoru

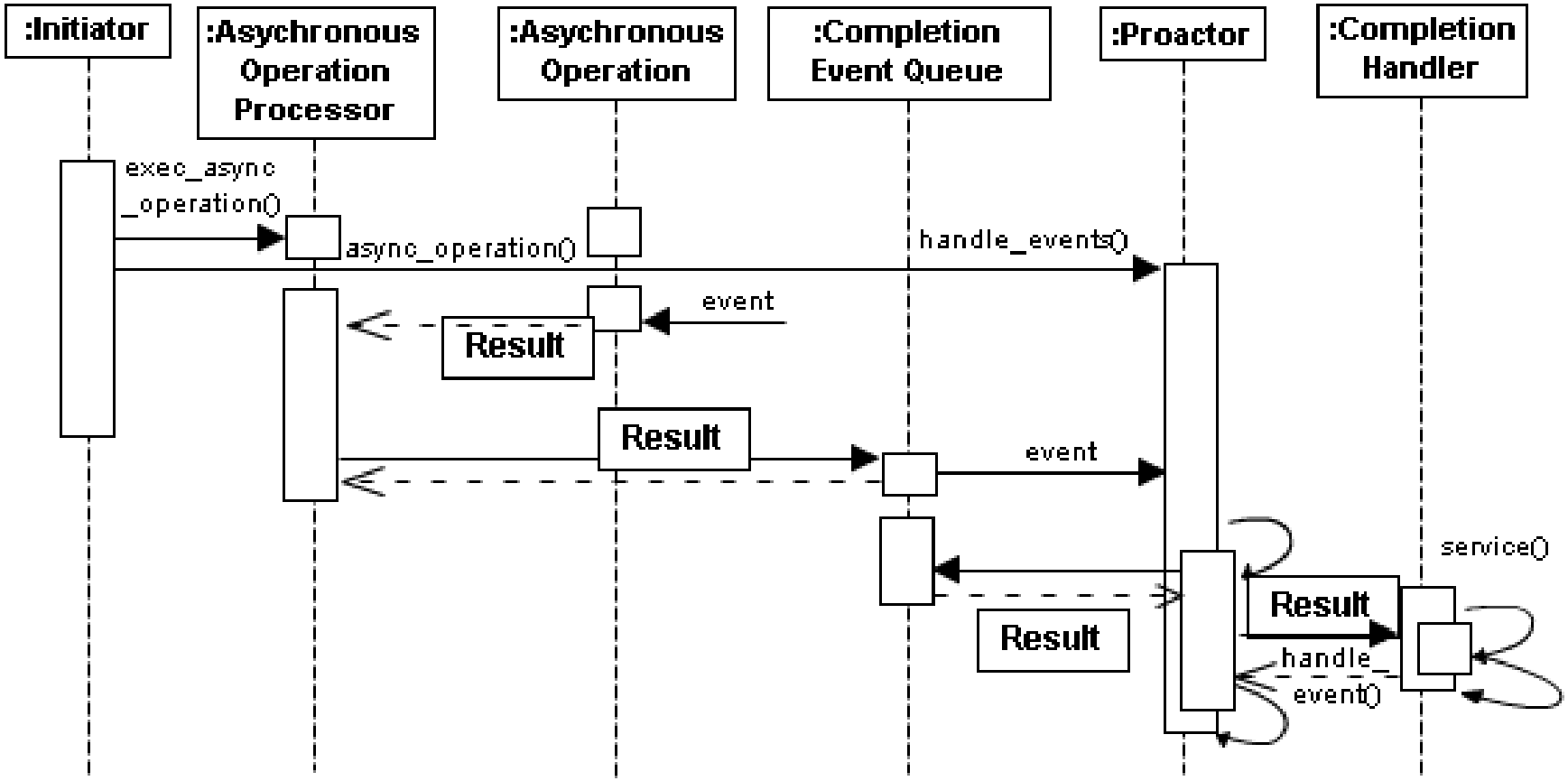
Proactor

- **Aktivní event demultiplexing and dispatching model**
- **Využívá Asynchronous Operation Processor**
  - typicky I/O
- **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
- **Vhodno použít, jestliže aplikace**
  - vykonává asynchronní operace bez zablokovaní volajícího vlákna
  - aplikaci stačí notifikace, když asynchronní operace skončí

Proactor - struktura



Proactor – v čase



Proactor – Příklad ze života, použití

- **Upozornění nepřijatých hovorů**
  - zavolám momentálně nedostupné kamarádce
  - nechám ji (jako initiator) vzkaz na záznamníku (zde asynchronous operation processor), ať zavolá zpět
  - během čekání mohu vykonávat libovolnou jinou činnost
  - kamarádka zavolá zpět (Proactor)
  - zvednu telefon a představím se jako Completion handler ☺
- **Reálné implementace**
  - Windows: completion ports
  - UNIX: POSIX AIO (asynchronous I/O)
  - Obsluha přerušení v OS
  - C++: Boost.Asio, ACE

Proactor vs Reactor

- **Společná myšlenka**
  - snaha o nahrazení multithreadingu v event-driven systémech
- **Reactor**
  - synchronní události
  - pasivní event demultiplexing and dispatching model
  - vhodné jen krátce trvající operace
- **Proactor**
  - asynchronní události
  - aktivní event demultexing and dispatching model
  - vhodné i pro dlouho trvající operace

Asynchronous Completion Token – Motivační příklad

- **Internetový burzovní systém**
  - musíme monitorovat všechny servery pro okamžité řešení selhání
  - na každém serveru běží management agent
  - management aplikace sbírá data z management agentů

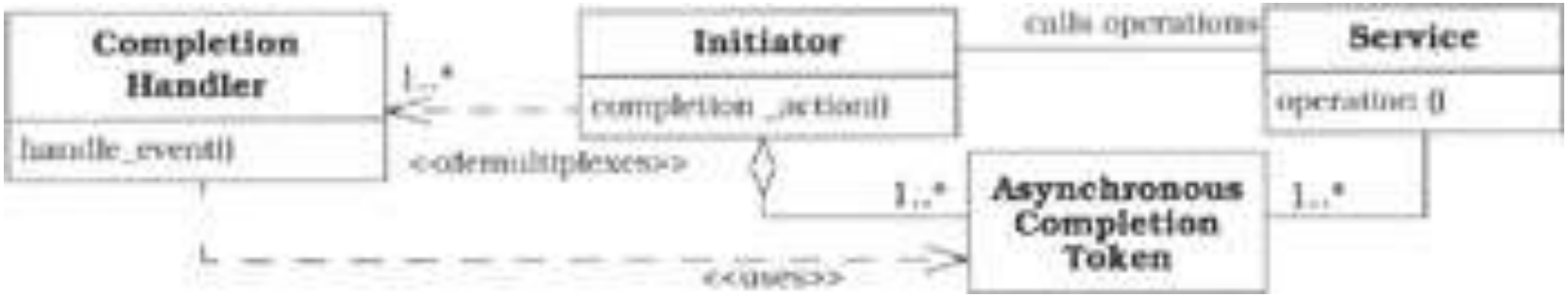
- **Spojení management aplikace s management agenty**
  - thread v aplikaci pro každého agenta, synchronní spojení
    - špatně škáluje
  - asynchronní spojení, ale jak efektivně demultiplexovat completion event?
    - řešení – Asynchronous Completion Token

Asynchronous Completion Token (ACT)

- **Také znám jako: Active Demultiplexing, Magic Cookie**

- **Iniciátor při asynchronním volání služby vytvoří ACT**
  - ACT identifikuje completion handler na straně iniciátora
  - služba při completion události vrátí iniciátoru ACT
  - ACT umožní efektivní demultiplexing completion eventu na straně iniciátora

ACT - Struktura



ACT – Příklad ze života, reálné implementace

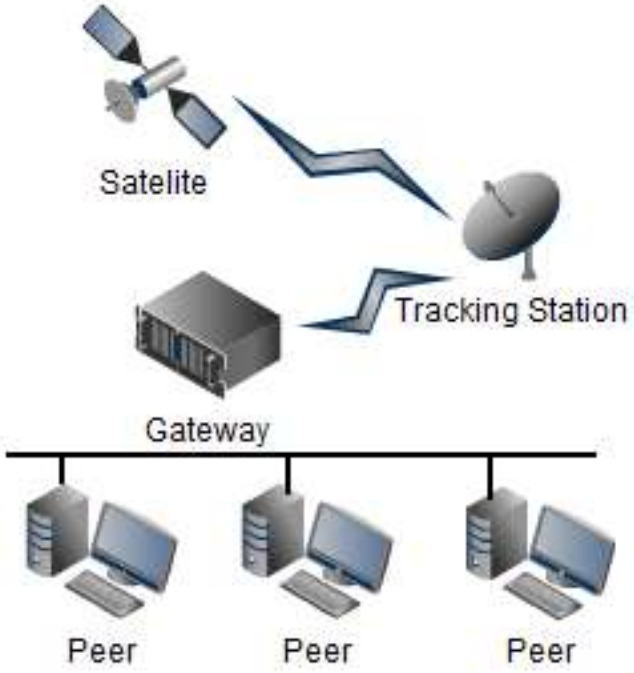
- **FedEx**
  - při odeslání zásilky může odesílatel vyplnit vlastní identifikátor
  - při doručení je identifikátor odeslán odesílateli
- **Reálné implementace**
  - HTTP koláčky
  - Windows: handles, overlapped I/O, I/O completion ports
  - UNIX: POSIX asynchronous I/O

Asynchronous Comletin Token - shrnutí

- **Výhody**
  - jednoduchost, prostorová nenáročnost, flexibilita
  - efektivní uložení a obnovení stavu
- **Nevýhody**
  - potenciální memory leak
  - nutnost autentizace tokenu v případě nedůvěryhodné služby
  - pokud je ACT pointer, nesmí být přemapována paměť (např. restart aplikace)

Acceptor-Connector – motivační příklad

- **Gateway**
  - každá peer služba používá gateway na odesílaní a přijímaní dat, příkazů a informací o stavu, například k satelitu
- **Problém**
  - navázání spojení se satelitem je komplexní
  - kód pro navázání sopjení je téměř úplně nezávislý na kódu samotných služeb
- **Řešení?**
  - Acceptor-Connector

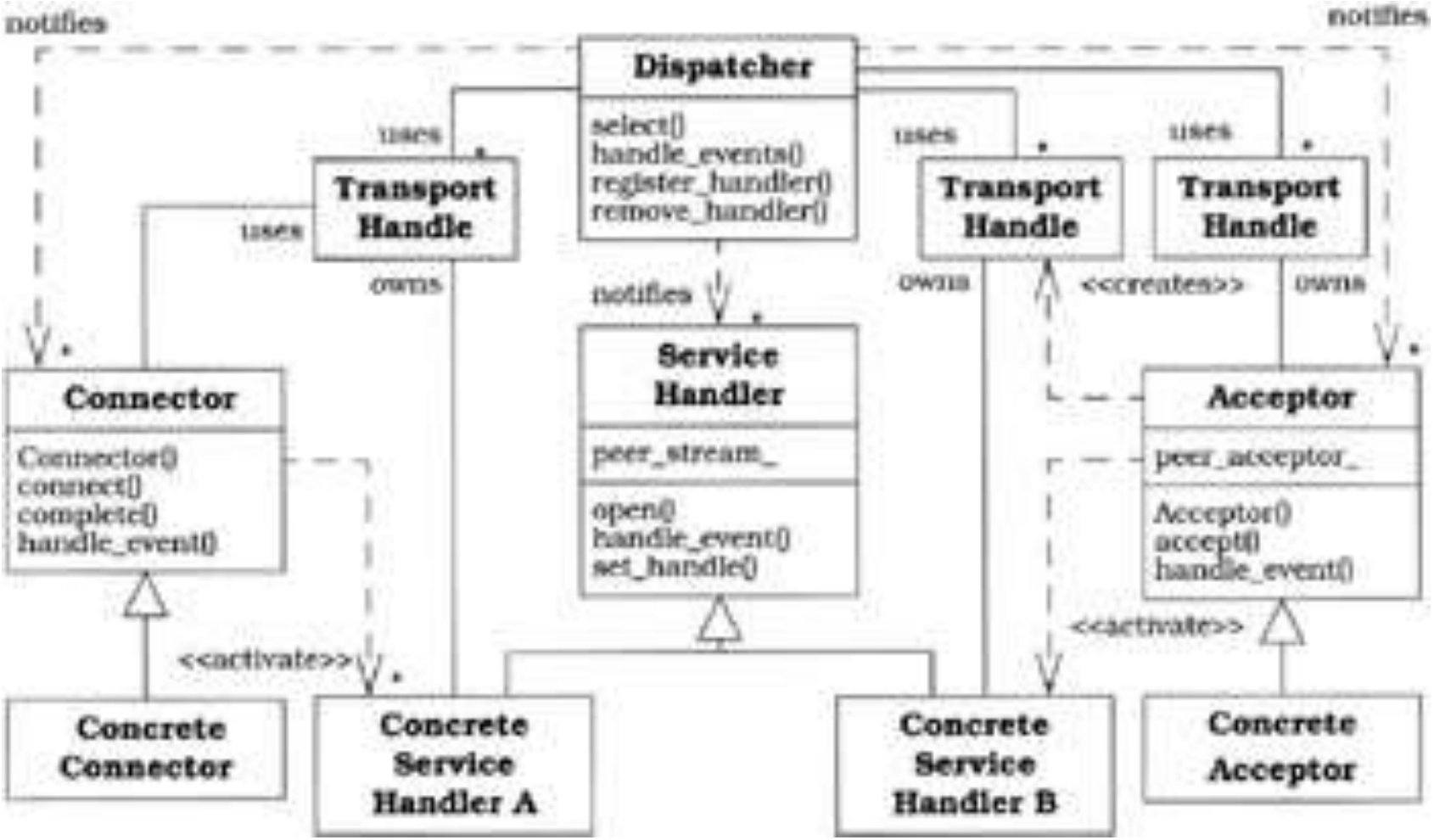


Acceptor-Connector

- **Vlastnosti**
  - odděluje zřízení připojení a inicializaci od vlastní služby
  - zapouzdření aplikačních služeb pomocí peer service handlerů

- **Struktura**
  - Acceptor přijímá pasivně spojení
  - Connector vytváří aktivně spojení
  - service handlers implementují vlastní službu

Acceptor-Connector - Struktura



Acceptor-Connector – Příklad, reálné implementace

- **Manažeři a sekretářky**
  - manažerka Alice chce hovořit s manažerem Bobem
  - manažerka Alice požádá svou sekterářku (Connector) o uskutečnění hovoru
  - sekterářka zavolá Bobově sekretářce (Acceptor)
  - ta předá telefon Bobovi
  - Alice a Bob jsou peer service handlers

- **Reálné implementace**
  - ACE (C++), Java CE
  - UNIX – Inetd
    - master acceptor process
    - service handlers: FTP, TELNET, DAYTIME a ECHO