Ereignisbearbeitung

Proactor – Asychronous Completion Token – Acceptor Connector

Software Design Patterns – 01072009

Überblick

- Woher kommen diese Pattern? Welche Formen von I/O gibt es? Was ist Ereignisbearbeitung?
- Pattern Proactor
- Pattern Asynchronous Completion Token
- Pattern Acceptor–Connector
- 35 Folien (en–de) ~ 30 Minuten

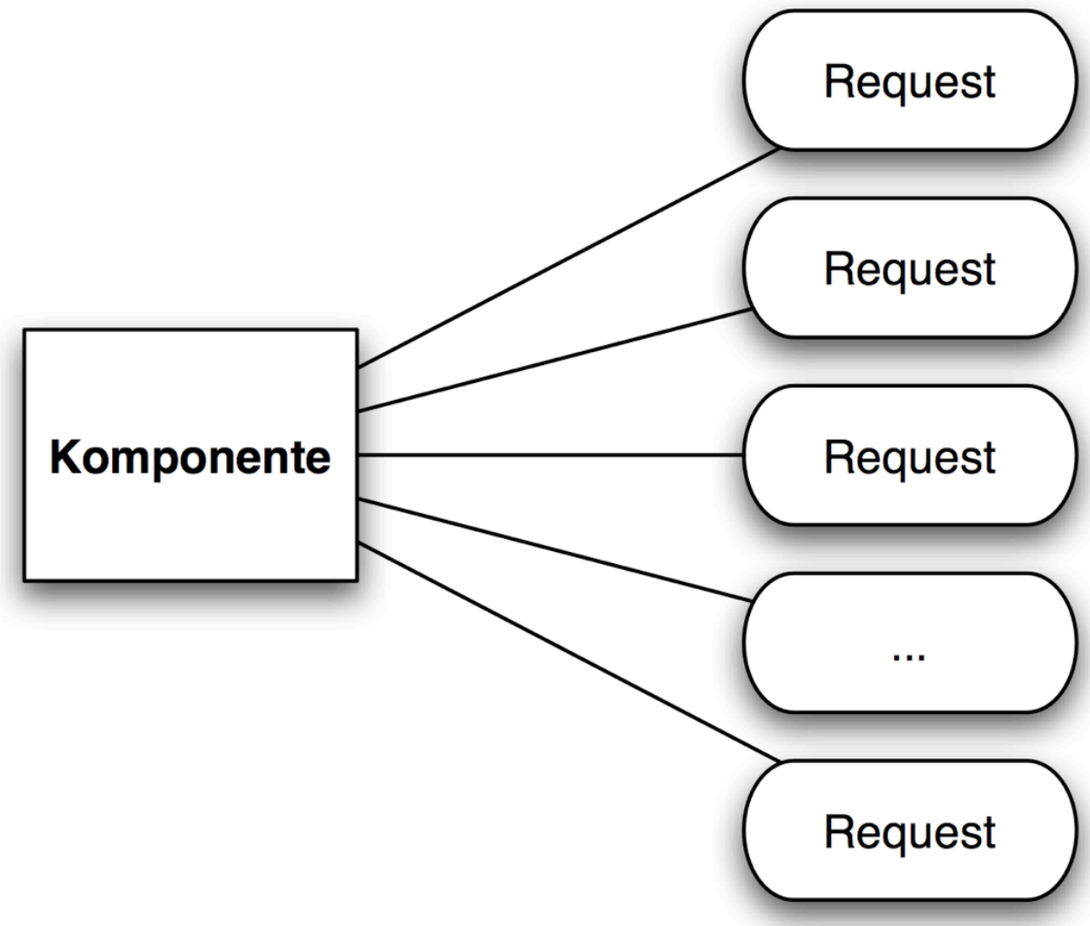
Entstehung der Pattern

- 1996–97
- Pattern Language of Programs (PLOP)
  http://hillside.net/plop
- D.C. Schmidt
  http://www.cs.wustl.edu/~schmidt

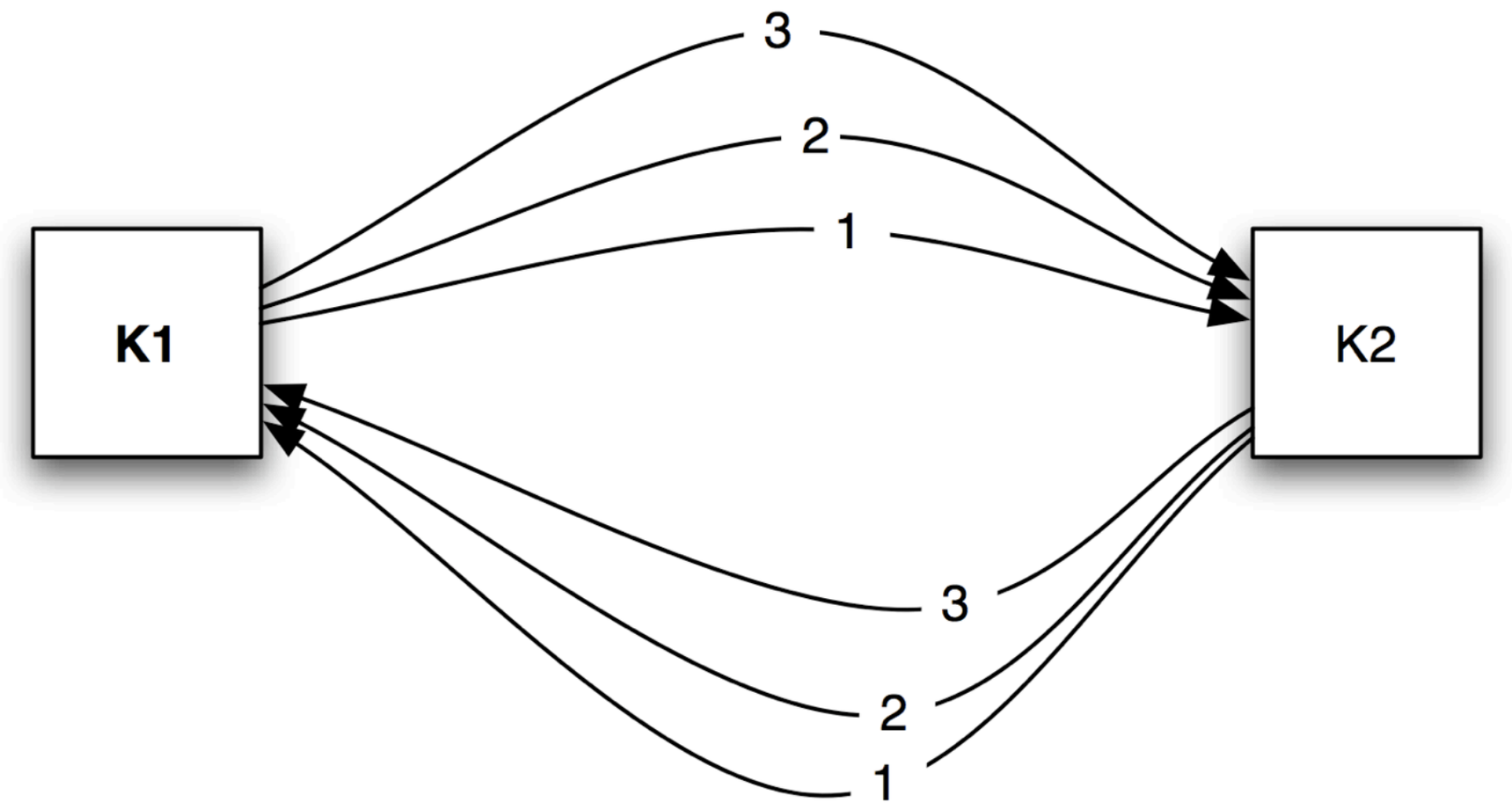
Ereignisbearbeitung

- tritt bevorzugt in Netzwerken auf
- Proactor 1 Komponente – N Requests
- ACT 2 Komponenten – N Nachrichten
- Acceptor–Connector 2 Komponenten – N Services

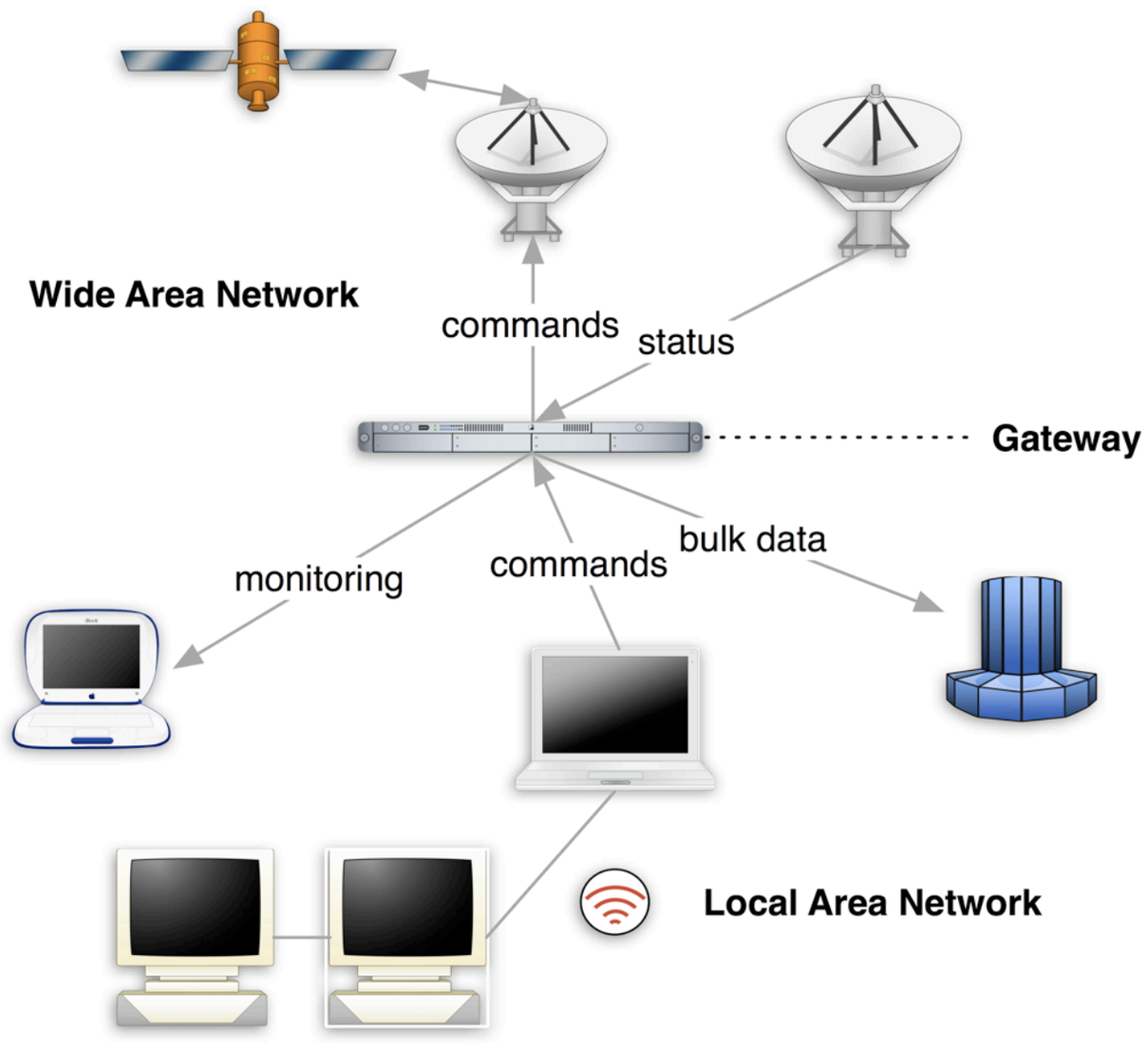
Ereignisbearbeitung
Proactor



Ereignisbearbeitung ACT



Ereignisbearbeitung Acceptor–Connector



I/O

- Synchronous – Asynchronous
  I/O completes – returns **vs.** I/O in parallel – data access through other mechanisms
- Blocking – Nonblocking (e.g. sockets)
  blocked until data is available **vs.** not blocked and notified (e.g. via polling, asynchronous notification)

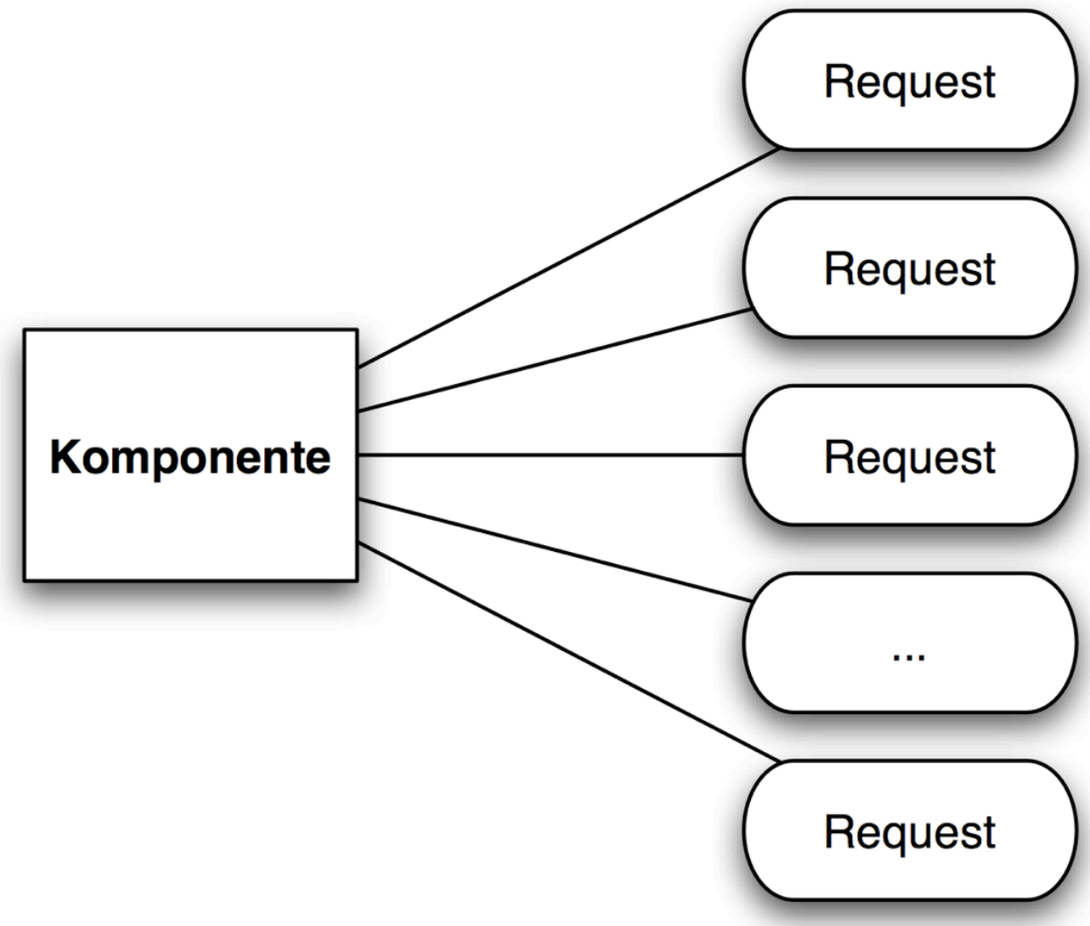
I/O

- blocking
  caller is blocked, no thread reuse
- non–blocking synchronous
  return to caller immediatly with data or some error code
- non–blocking asynchronous
  caller gets notified

Proactor
I/O Event Dispatching

- Kontext Ereignisgesteuerte Applikation, welche mehrere Serviceanfragen empfängt und asynchron verarbeitet
- Kräfte Skalierbarkeit, Effizienz, Änderungsanforderungen, Information Hiding

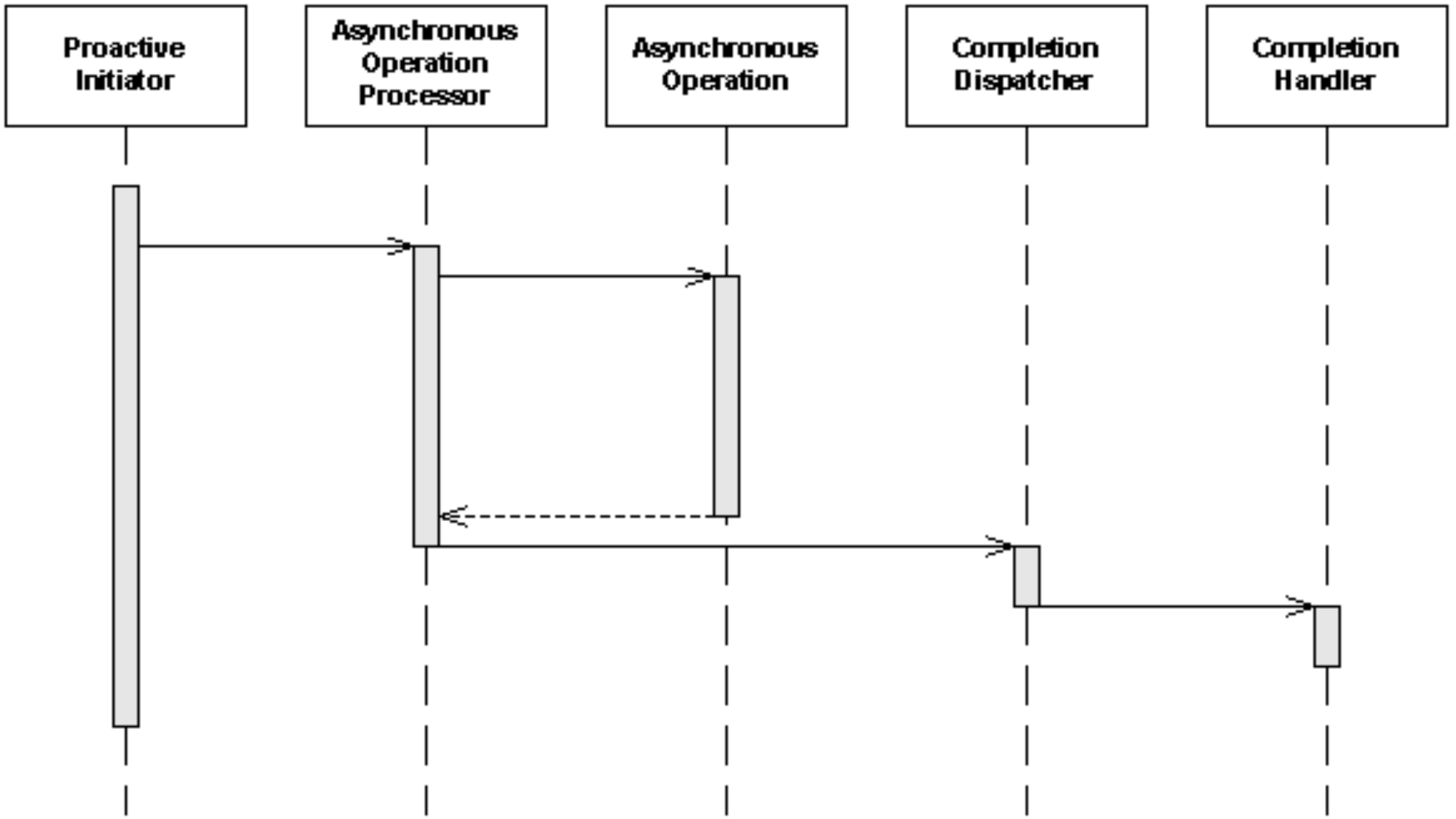
Proactor



Proactor

- **Handler** initiert **asynchrone Operation** (falls OS dies unterstützt)
- Ereignis–**Demulitplexer wartet** auf Fertigstellung
- Demultiplexer **ruft** entsprechenden Handler **auf**
- Handler **liest** Daten vom **Buffer**

Proactor


Proactive Initiator
Asynchronous Operation Processor
Asynchronous Operation
Completion Dispatcher
Completion Handler

Proactor in Java

- Time–Server & Client (37)
- **12916.55** req/second on ancient 32bit system, server and client on same machine

Proactor in Java

- via **java.nio**
- *"Unfortunately, there is a price to pay: an architecture based on I/O multiplexing is significantly harder to understand and to implement correctly than one based on thread pooling."*

http://www.onjava.com/pub/a/onjava/2004/09/01/nio.html

Proactor

- **Libraries**
  - ACE/Proactor
  - java.nio
  - asyncore, twisted
- **I/O Demultiplexing**
  - e.g. *select(2)*, *poll(2)*, */dev/poll*, *kqueue*

Proactor Example

- **HTTP GET**
  - HTTP Handler initiates AIO
  - **read** from socket asynchronously (which socket? which completion handler?)
  - OK, it **runs** ... meanwhile do sth. else
  - AOP **pass results** (bytes read, status, etc) and completion handler back to the proactor – e.g. **transmit** a file *(asynchronously, ...)*

Proactor/Reactor

- In **Reactor**, the event demultiplexor waits for events that indicate when a file descriptor or socket is ready for a read or write operation. The demultiplexor passes this event to the appropriate **handler, which is responsible for performing the actual read or write**.

Proactor/Reactor

- In the **Proactor** pattern, by contrast, the **handler**—or the event demultiplexor on behalf of the handler—**initiates asynchronous read and write** operations. The I/O operation itself is performed by the operating system (OS).

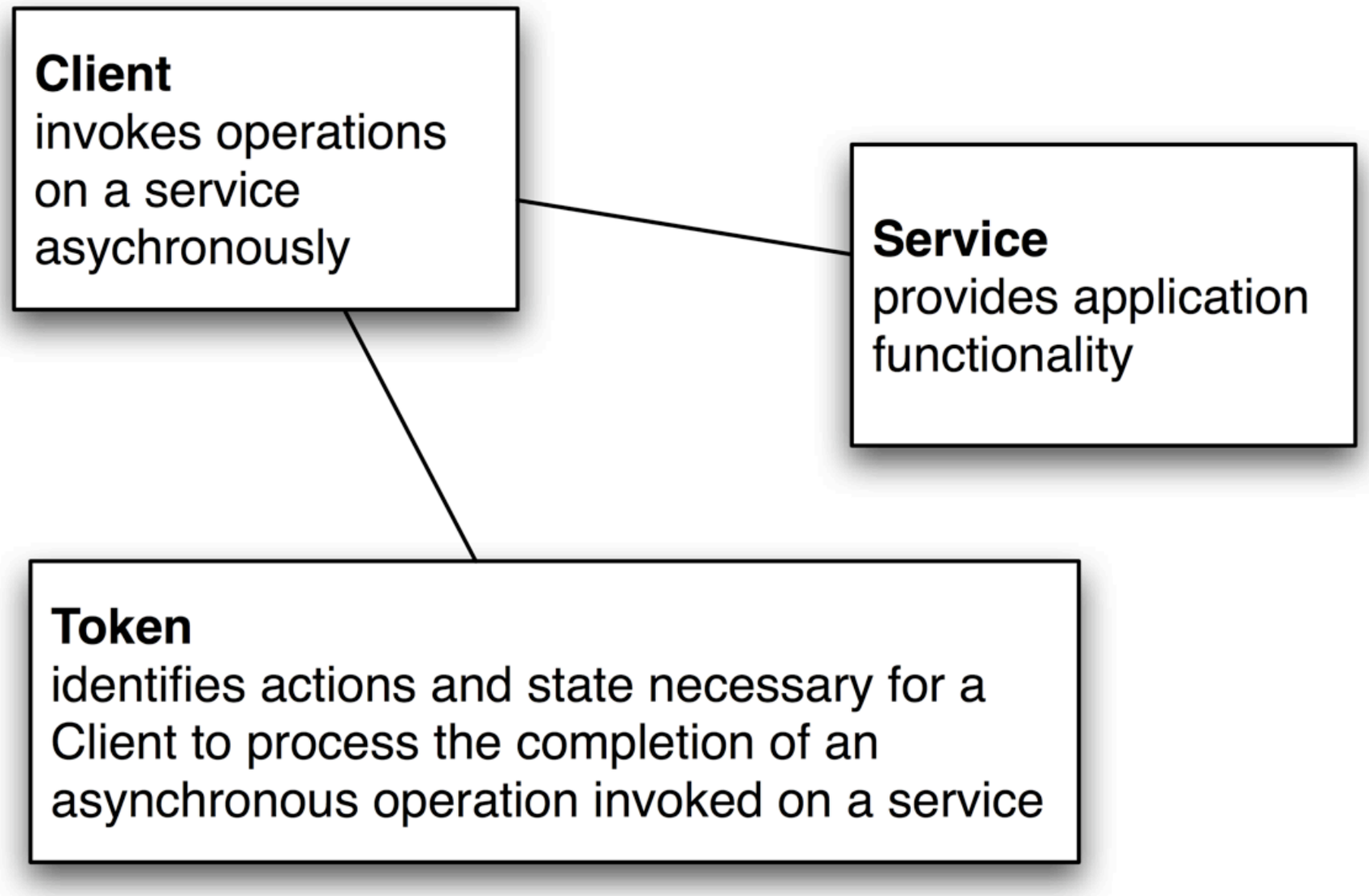
Proactor Pro/Con

- **Pro**
  - Scalable
  - High Performance
- **Con**
  - Complex

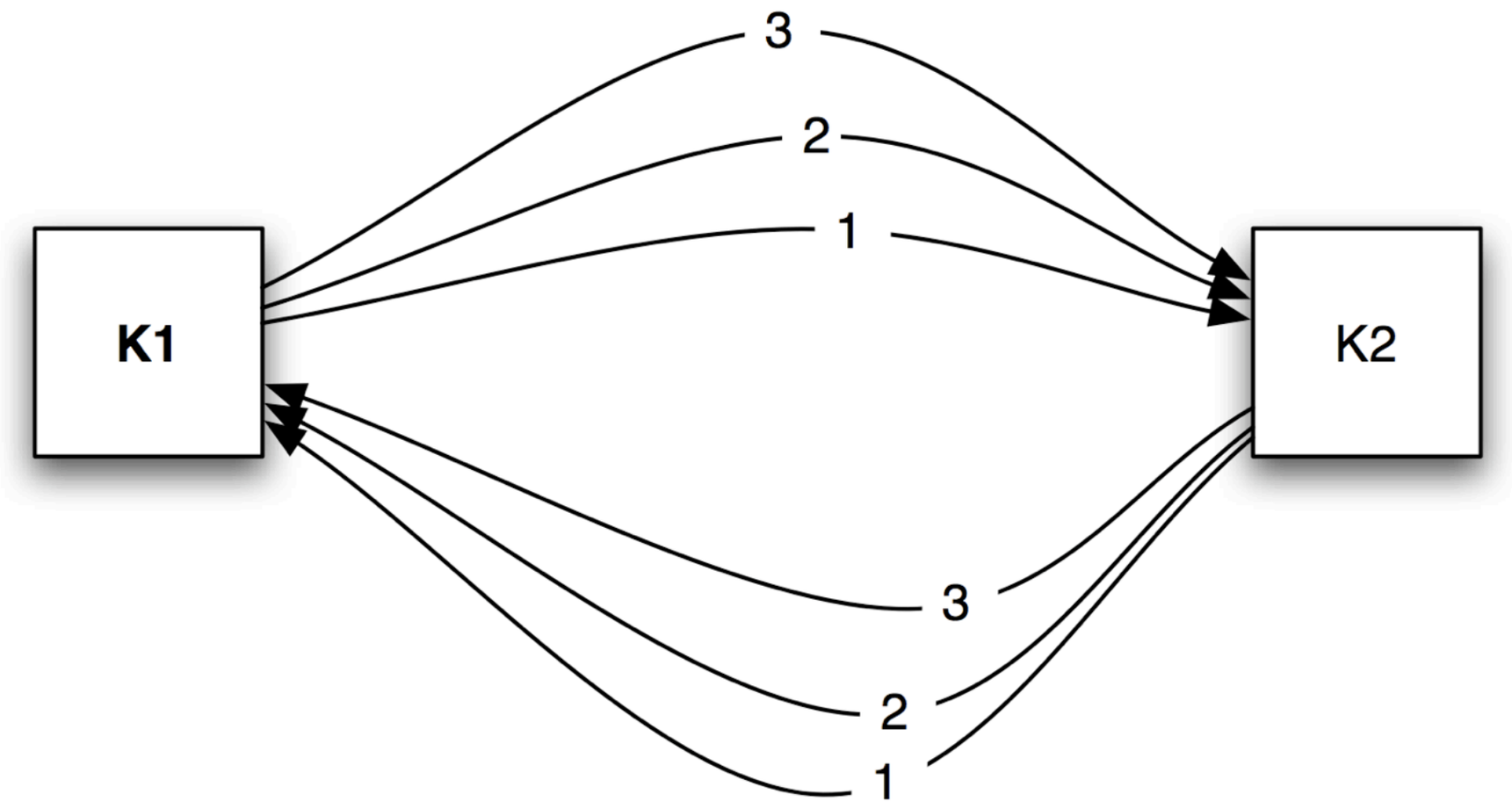
ACT
Active Multiplexing

- Kontext Eine verteilte Umgebung, in der Applikation asychron Services aufrufen und Antworten sukzessive verarbeiten
- Kräfte Effizienz, Separation of Concerns

ACT


Client
invokes operations on a service asychronously
Service
provides application functionality
Token
identifies actions and state necessary for a Client to process the completion of an asynchronous operation invoked on a service

ACT



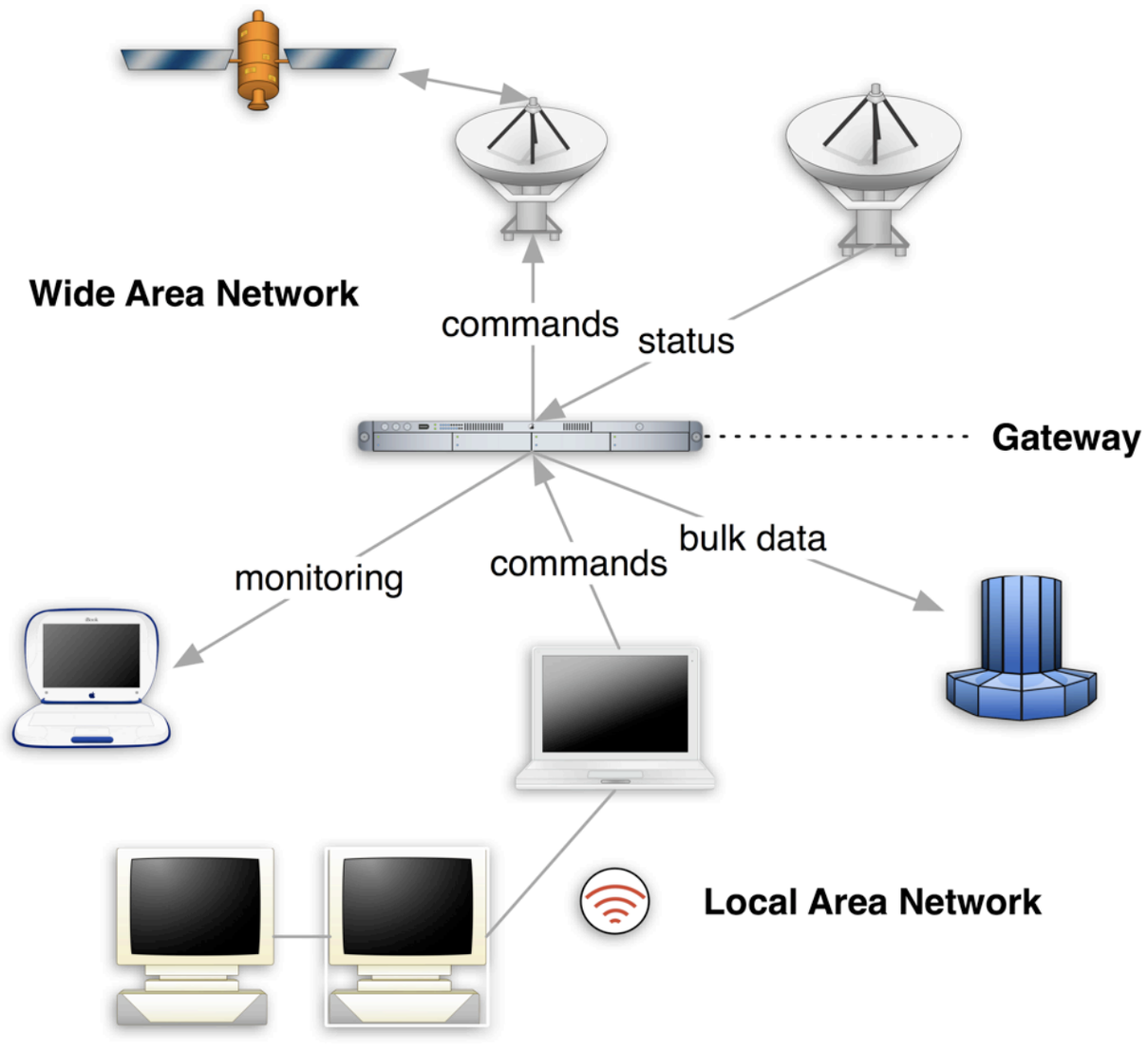
ACT

- **Generate**
  - for a specific State and Answer
    - a unique token
- and **associate** it
  - with the message

ACT Pro/Con

- **Pro**
  - Lightweight Datastructure
  - Efficient, Flexible
- **Con**
  - Memory–Efficiency (*dealloc()* etc.)
  - Persistence

Acceptor–Connector



Acceptor–Connector

- Connection Role – Communication Role
- Kräfte Änderbarkeit, Separation of Concerns, Plattform–Unabhängigkeit, Information Hiding, Skalierbarkeit

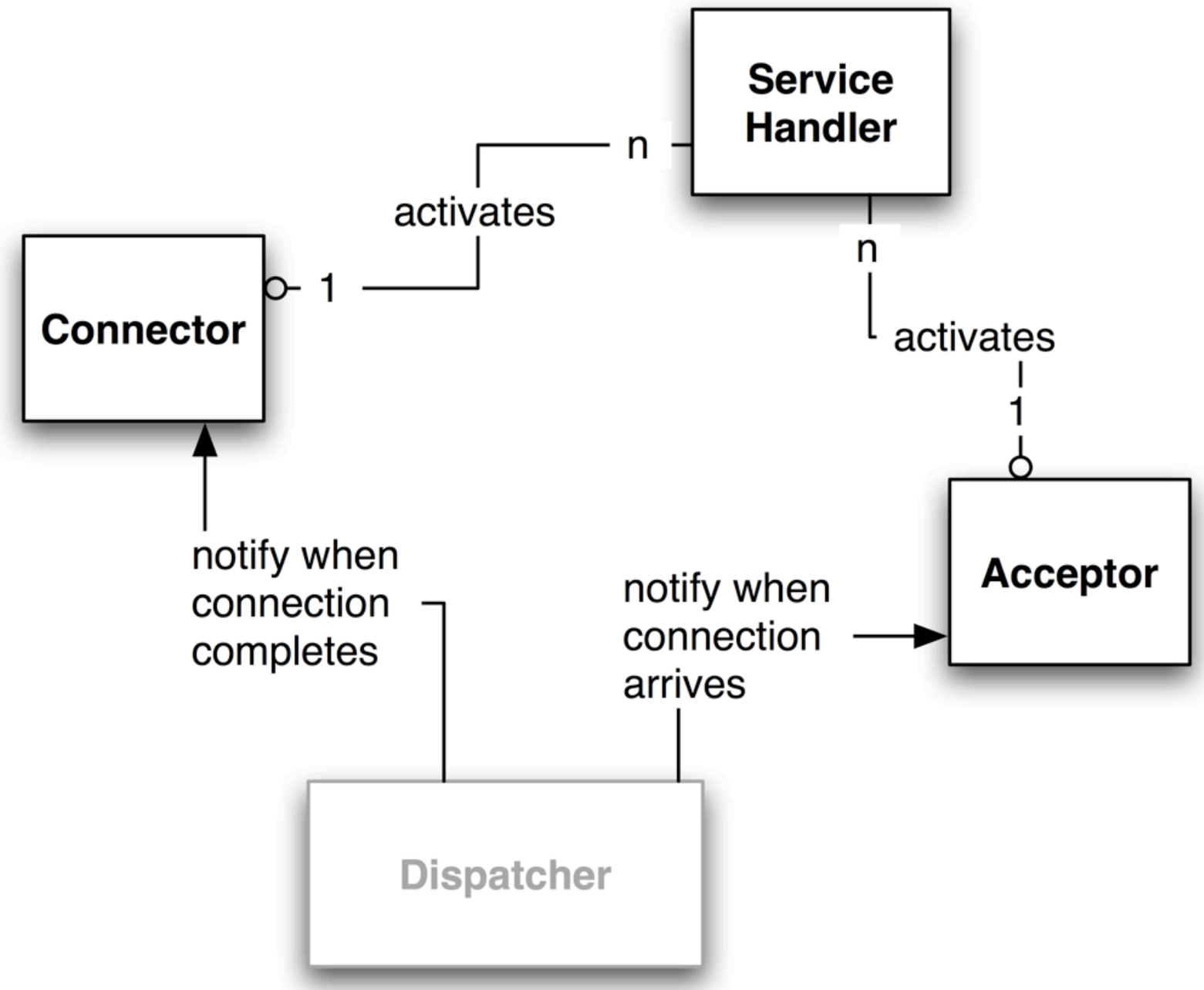
Acceptor/Connector

- **Clients can**
  - **initiate** connections through **various** protocols
  - **wait** for messages
- Client/Server Roles are somewhat **exchangeable**

Acceptor/Connector

- **Foreach (Service)**:
  - implement 2 fabrics: acceptor, connector
  - **acceptor** may create transport endpoints (passive, waiting)
  - **connector** may initiates connection to another transport endpoint (active)
  - both initialize a **service handler**

Acceptor–Connector



Acceptor/Connector

- 3 Steps
  - **Endpoint–Initialization**
    - *Result* Event–Loop
  - **Service–Initialization**
    - Incoming Connections – *accept* – allocate required resources
  - **Transport**
    - e.g. Layer 7 protocols (Service Handler)

Acceptor/Connector Pro/Con

- **Pro**
  - Reuse, Portability
  - Extendable
- **Con**
  - Another Layer of Indirection

Acceptor/Connector

- **Examples**
  - *inetd*
  - *CORBA ORB*
  - *ACE/Framework*
  - *Ericson EOS Call Center Management*

Questions?

Thank You.