Progettazione architetturale

Alberto Gianoli

Univ. Ferrara - Corso di Laurea in Informatica

Dove?

- Pressmann, cap. 12
- Sommerville, cap. 11

Obiettivo

- La progettazione dell'architettura rappresenta la struttura dei dati e i componenti del programma necessari
- Considera lo stile architetturale che si intende usare per il sistema, la struttura e le proprietà dei componenti e lele relazioni tra componenti
- Se il sistema è grande e/o complesso si può avere un progettista di database (o data warehouse) per i dati e un "architetto di sistemi" per il resto del sistema
- Cosa vogliamo ottenere alla fine? Modello dell'architettura (dati e struttura programma), con le proprietà e le relazioni fra i componenti

L'architettura del software

- La definizione dell'architettura non ha lo scopo di creare del software operativo, piuttosto
  - analizzare l'efficacia del progetto nel soddisfare i vari requisiti
  - permettere la valutazione di tutte le possibili alternative architetturali del sistema in una fase iniziale in cui le modifiche sono relativamente poco costose
- Quindi l'architettura **non è** la stesura del codice
- ...ma una buona architettura facilita la successiva stesura del codice
- Architettura del software == progetto dei dati + progetto architetturale

Perché è importante?

Sostanzialmente tre motivi chiave:

- La rappresentazione dell'architettura è il mezzo che le parti interessate allo sviluppo usano per comunicare
- L'architettura mette in evidenza le decisioni progettuali preliminari: queste avranno un impatto nel successivo lavoro di sviluppo
- L'architettura è un modello relativamente conciso e facilmente comprensibile di come il sistema sarà strutturato e di come i vari componenti collaboreranno tra loro

Progetto dei dati

- “La progettazione dei dati a livello dei componenti si concentra sulla rappresentazione delle strutture dati il cui accesso avviene direttamente da parte di uno i più componenti software”
- Sono stati individuati una serie di principi per la progettazione dei dati

Principi di progettazione dei dati

- Si applicano gli stessi principi di analisi sistematica che si usano per le funzionalità e per il comportamento
- Bisogna individuare tutte le strutture di dati e le operazioni da svolgersi su ciascuna
  - nel fare la struttura tenere conto delle operazioni che si vuole svolgere sulla struttura stessa
- Occorre compilare un dizionario dei dati, e utilizzarlo per definire il progetto dei dati e del programma
  - class diagram: definiscono oggetti-dato e le operazioni applicate
- Le decisioni di basso livello sul progetto dei dati devono essere rimandate alle ultime fasi di progettazione
  - raffinare per passi successivi: l'organizzazione è definita durante analisi req., raffinata in questa fase, e specificata nei dettagli poi

Principi di progettazione dei dati

- La rappresentazione delle strutture dati deve essere nota solo ai moduli che hanno bisogno di utilizzarle direttamente
  - information hiding e coupling
- E' vantaggioso pensare in maniera modulare, pensando da subito a possibili riutilizzi
  - sviluppare librerie di strutture dati utili e di operazioni
- La specifica e l'implementazione degli ADT deve basarsi su un progetto del software e su un linguaggio di programmazione
  - implementare un ADT complicato può dipendere dal supporto che un determinato linguaggio di programmazione offre

Stili architetturali

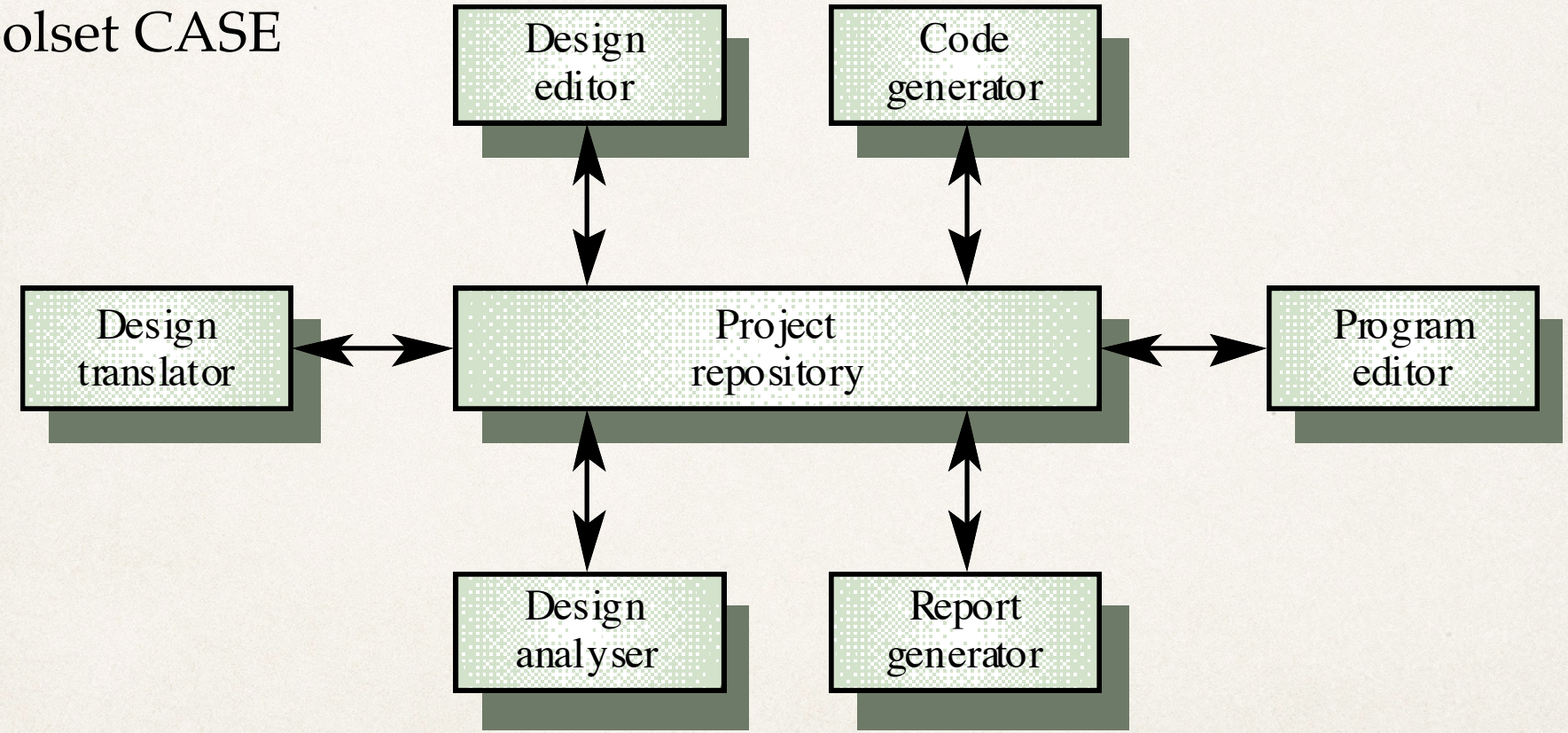
- Uno stile architetturale è composto da:
  - un insieme di componenti che implementano le funzionalità richieste
  - un insieme di connettori che consentono la comunicazione (e quindi la cooperazione) tra i componenti
  - un insieme di vincoli che definiscono i modi in cui i vari componenti possono essere integrati fra loro per formare il sistema
  - dei modelli semantici che permettono al progettista di comprendere il funzionamento del sistema sulla base delle sue componenti
- Gli stili sono quasi infiniti, ma si possono classificare in un numero ridotto di stili base

Architettura basata sui dati
(Architettura a repository)

- Il sistema è centrato su un archivio di dati
- Le componenti accedono all'archivio operando indipendentemente tra loro
- L'archivio può essere passivo (tutte le operazioni sono in mano ai client) oppure attivo (l'archivio notifica ai client le variazioni nei dati)
- Il vantaggio è l'indipendenza tra i vari moduli (integrabilità): si può aggiungere un modulo o intervenire su uno di essi senza che gli altri ne risentano
- L'accoppiamento tra le componenti può avvenire mediante un meccanismo a blackboard

Architettura a repository

Architettura di un toolset CASE



Architettura a repository

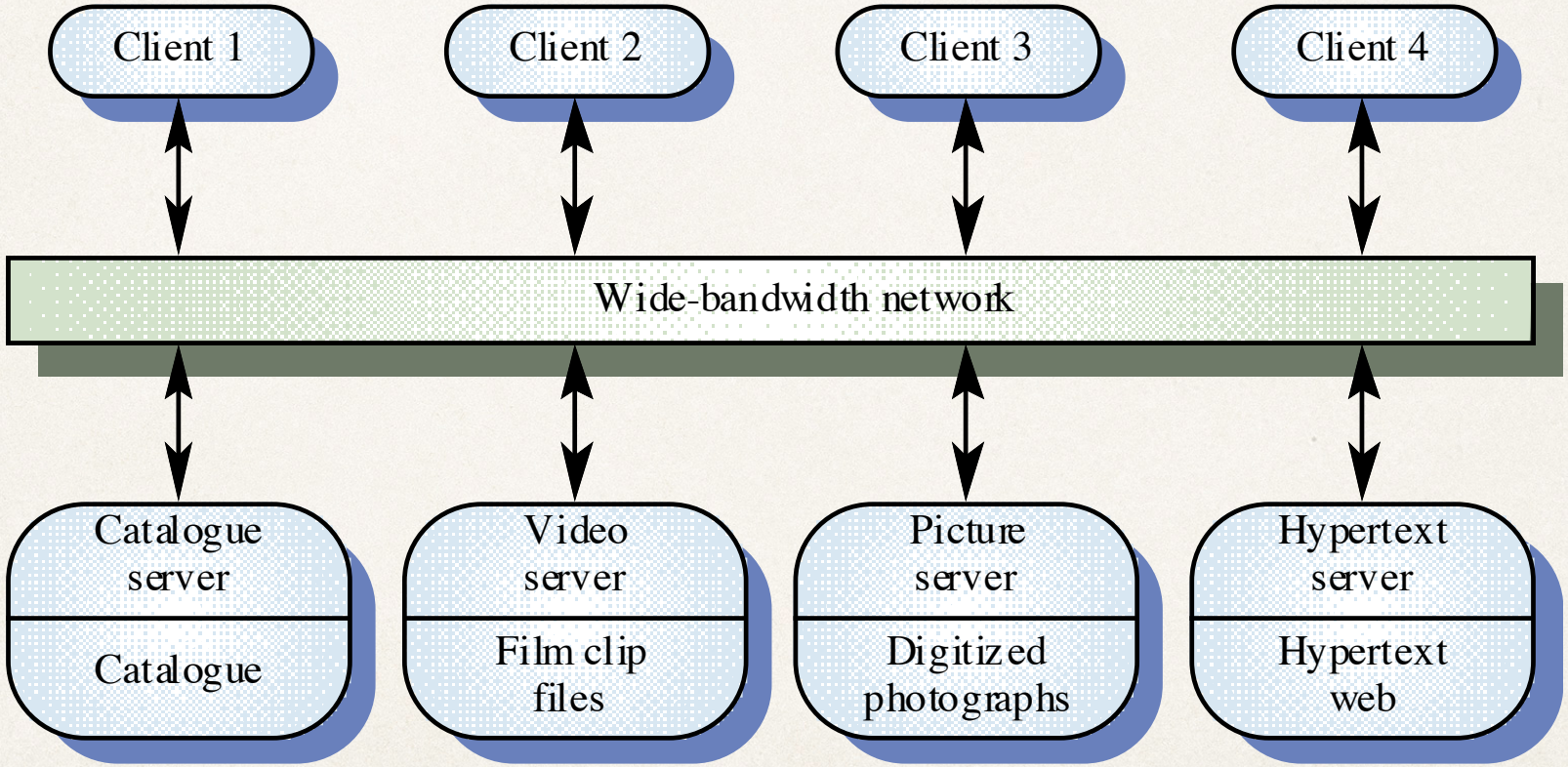
- Vantaggi
  - modo efficiente di condividere grandi quantità di dati
  - i sottosistemi possono disinteressarsi a come i dati vengono condivisi, la gestione è centralizzata (backup, sicurezza, …
- Svantaggi
  - i sottosistemi devono concordare su un modello dei dati che inevitabilmente è un compromesso tra esigenze diverse
  - modificare la struttura (schema) dei dati è difficile e dispendioso
  - non c'è spazio per politiche specifiche di accesso ai dati
  - bassa scalabilità: il repository centrale spesso è un collo di bottiglia

Architettura client-server

- Modello per sistemi distribuiti, mostra come dati e processi sono distribuiti su un insieme di componenti
  - insieme di server autonomi che offrono servizi specifici
  - insieme di clienti che richiedono questi servizi
  - una rete di comunicazione che permette ai clienti di accedere ai server

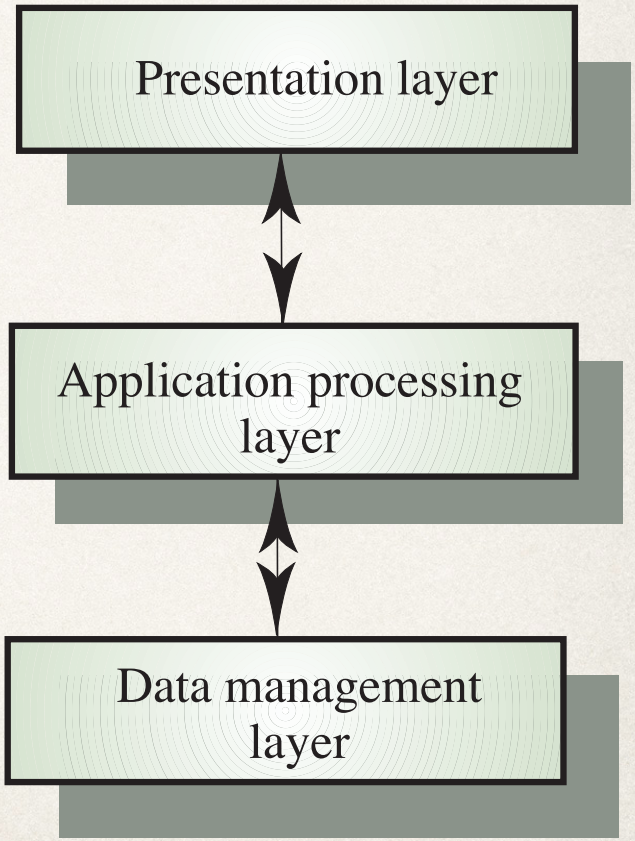
Architettura client-server

Libreria digitale

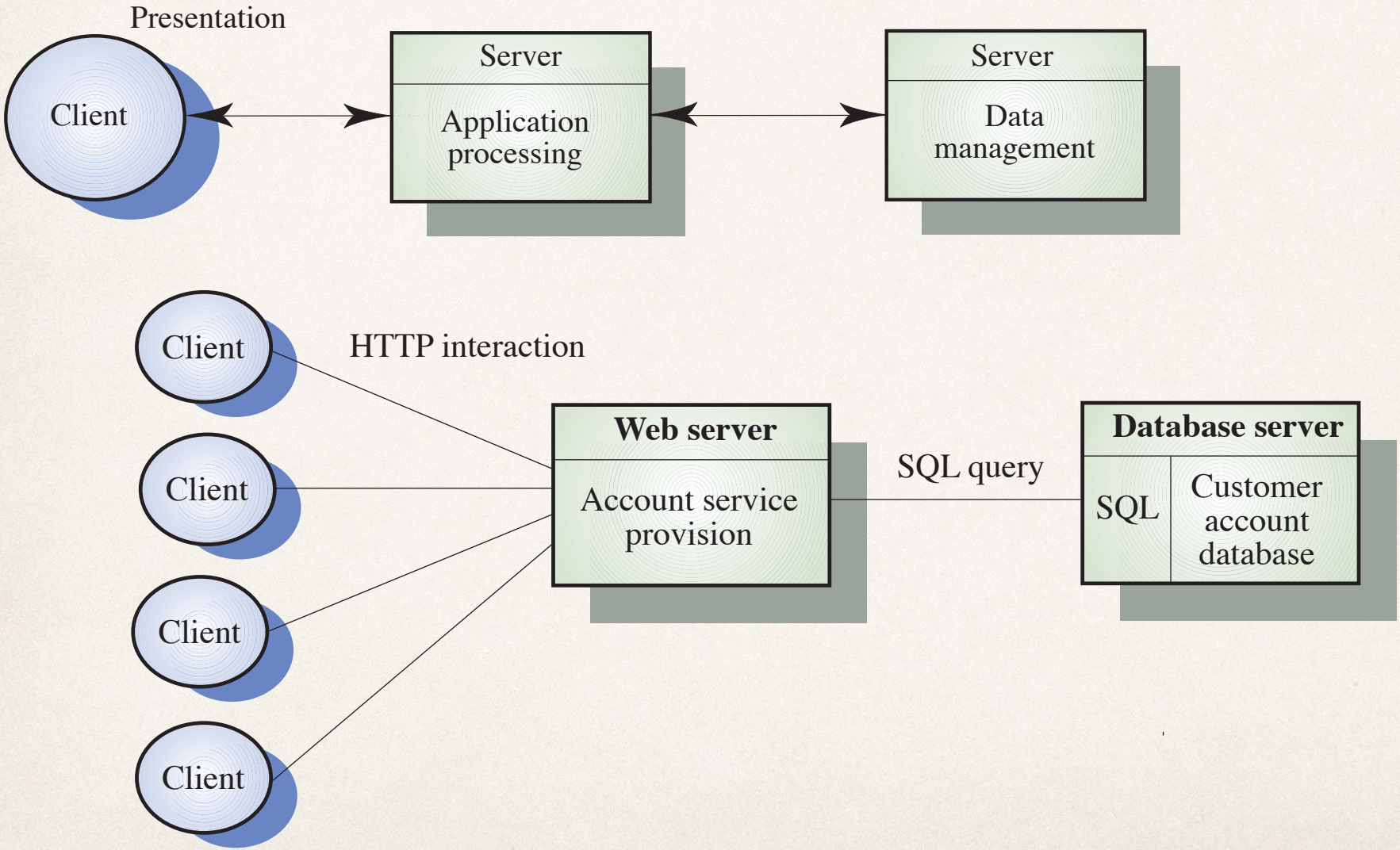


Strati funzionali nell'architettura client-server (three tier architecture)

- Presentation layer
  - si occupa di presentare i risultati della computazione agli utenti del sistema e di gestire gli input da parte degli utenti
- Application processing layer
  - si occupa di offrire le funzionalità specifiche dell'applicazione
- Data management layer
  - si occupa di gestire la comunicazione con il DBMS



Esempi three-tier architecture



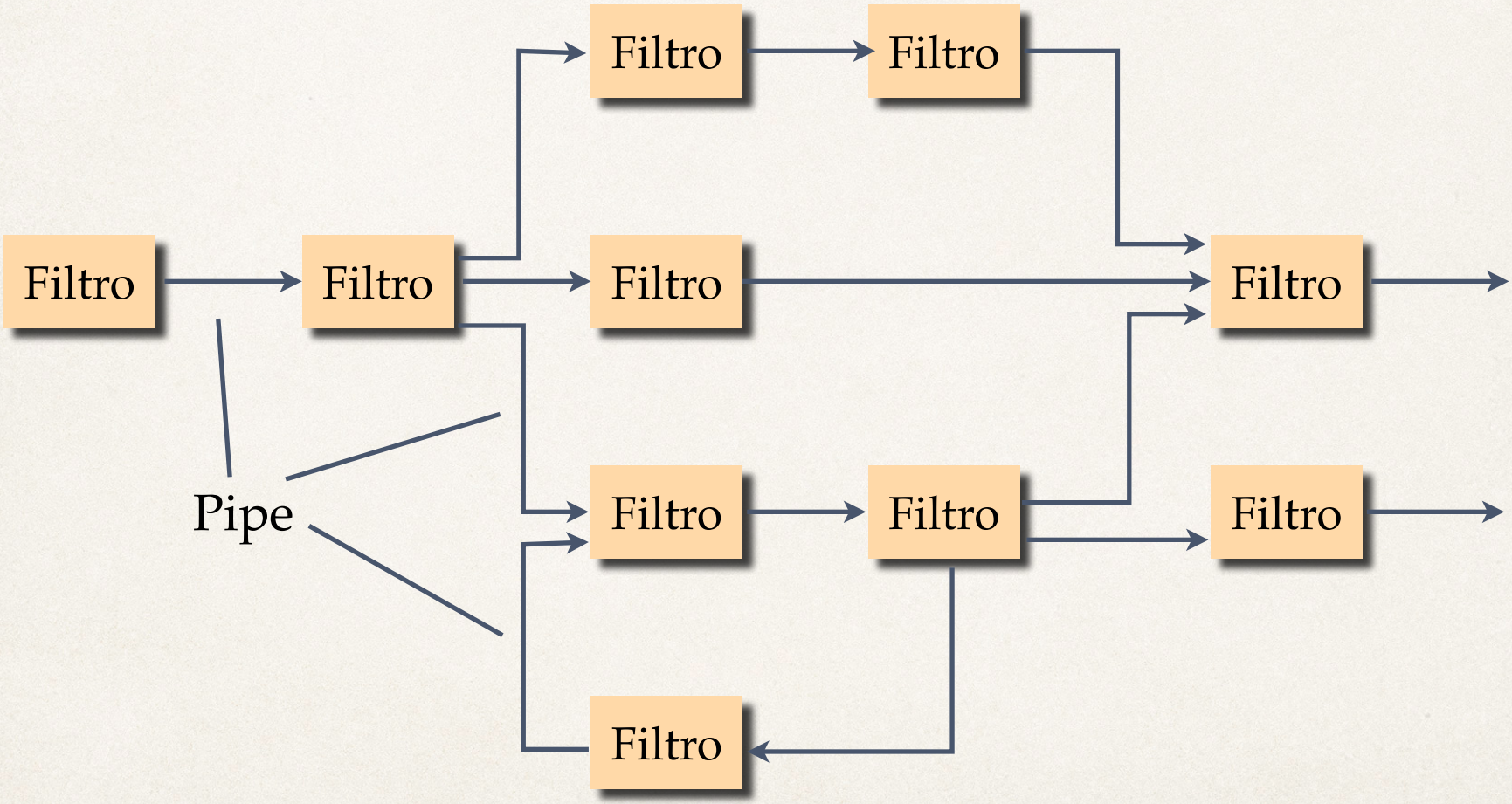
Caratteristiche del modello client-server

- Vantaggi
  - la distribuzione dei dati è molto semplice
  - fa un uso effettivo del sistema di rete e può richiedere hw economico (molti nodi a bassa potenza per effettuare computazioni complesse)
  - è facile aggiungere nuovi server o fare l'upgrade dei server esistenti
- Svantaggi
  - non c'è un unico modello condiviso dei dati, quindi ogni sottosistema fa uso di un modello proprio, e lo scambio di dati può essere inefficiente
  - alcune attività di gestione dei dati devono essere replicate
  - non c'è un registro centrale dei nomi e servizi: può essere difficile sapere quali dati/servizi sono disponibili

Architettura a flusso di dati
(architettura pipe-and-filter)

- Il sistema è modellato sul flusso di dati, dalla fase di input a quella di output
- I moduli si comportano da filtri connessi da pipe di dati
- Ogni filtro si attende solo dati in input con un certo formato e produce dati in output di formato prefissato
- Ogni filtro lavora senza occuparsi di cosa lo precede o lo segue
- Se il flusso dati degenera in una unica catena di filtri, allora l'architettura si dice "a filtro batch sequenziale"

Esempio architettura pipe-and-filter



Caratteristiche dell'architettura a flusso di dati

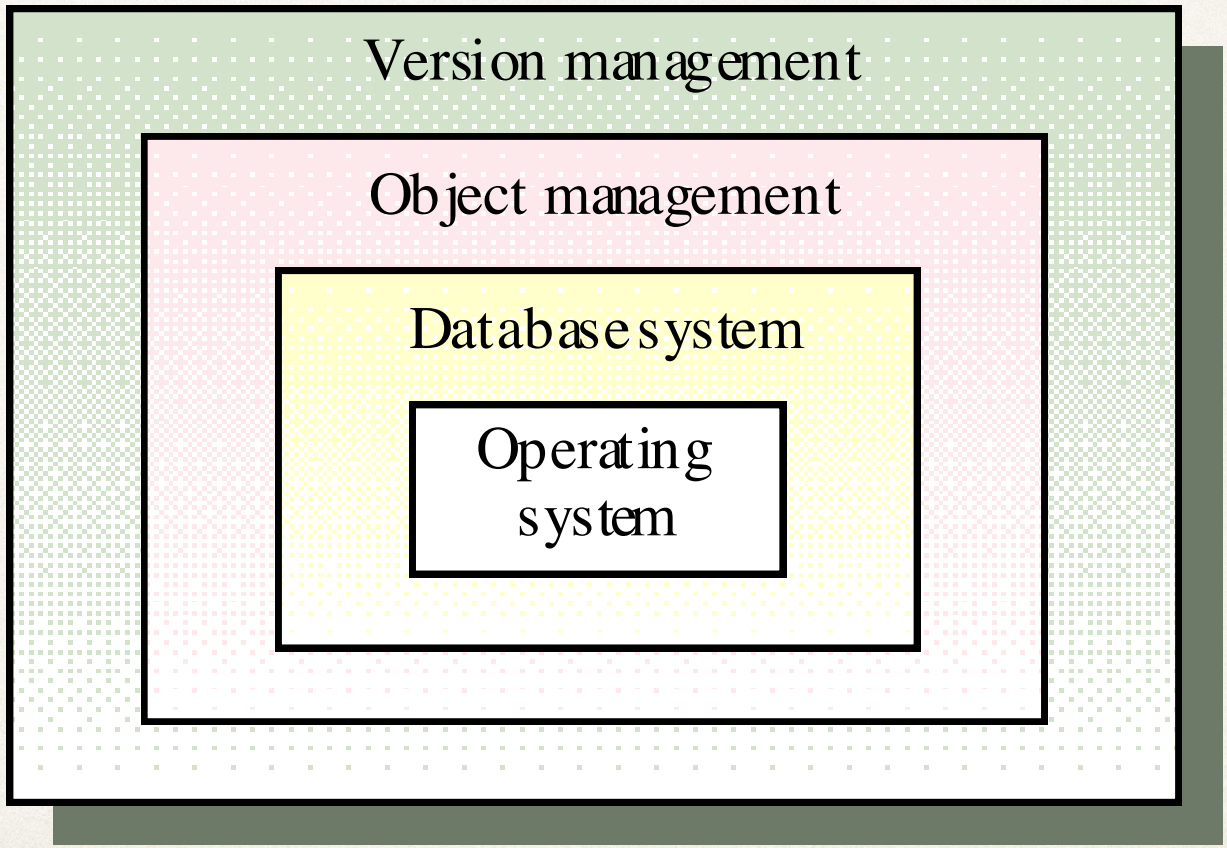
- Vantaggi
  - è facile costruire computazioni complesse mediante concatenazione di filtri semplici
  - ogni filtro è un "scatola nera" riutilizzabile in altre situazioni
  - se ben programmati, i filtri non condividono lo stato (basso accoppiamento)
- Svantaggi
  - i formati di dati in input e output di filtri collegati devono essere compatibili
  - se la struttura di controllo non può essere "linearizzata", questo modello non è adeguato

Architettura a macchina astratta
(architettura a livelli)

DETTA ANCHE "A STRATI"

- Usato per modellare l'interfaccia tra sottosistemi
- Organizza il sistema in un insieme di strati (o macchine astratte) ognuno dei quali offre un insieme di servizi
- Supporta lo sviluppo incrementale di sottosistemi a livelli diversi: se l'interfaccia di un livello cambia, ne risulta affetto solo il livello adiacente

Esempio di architettura a livelli



OPPURE PENSATE ALL'ISO-OSI

Caratteristiche dell'architettura a livelli

- Vantaggi
  - supporta lo sviluppo incrementale, livello dopo livello
  - se si cambia l'interfaccia di un livello, solo quello adiacente ne risente
- Svantaggi
  - spesso può essere complicato strutturare il sistema in questo modo
  - può essere restrittivo pensare che un livello possa interagire solo col precedente o successivo

Riassumendo (1)

- Modelli di struttura di un sistema
  - repository
  - client-server
  - flusso dati
  - macchina astratta (strati)

Scomposizione modulare

- Ulteriore raffinamento a livello strutturale: i sottosistemi sono scomposti in moduli
- Sottosistema - modulo: distinzione non chiara ma
  - sottosistema è un sistema di diritto: sono composti da moduli, hanno interfacce ben definite per comunicare con altri sottosistemi
  - modulo di solito è componente sottosistema e fornisce uno o più servizi ad altri moduli, o usa servizi di altri moduli; non è indipendente

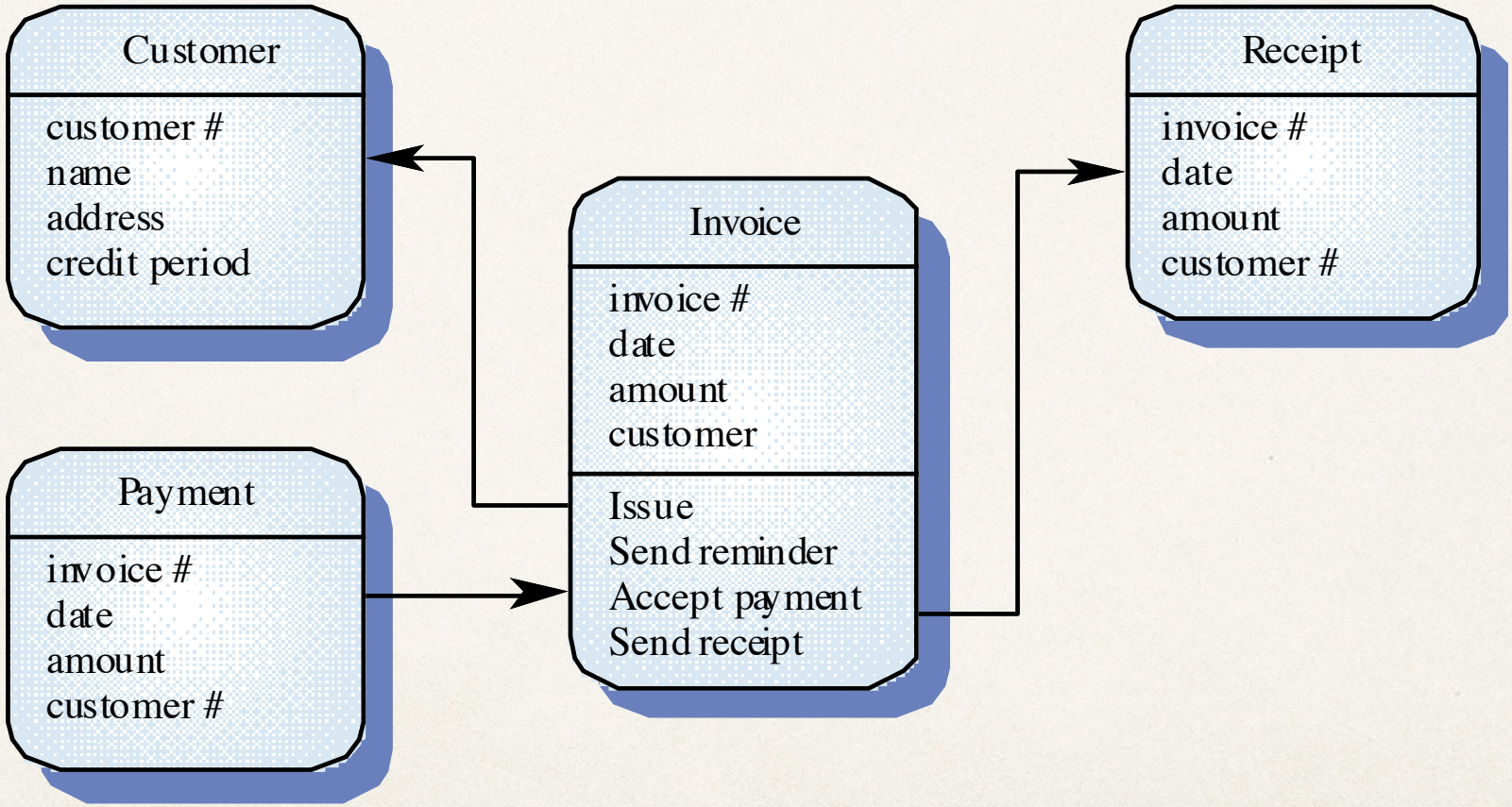
Scomposizione modulare

- Due modelli di scomposizione:
  - modello a oggetti, dove il sistema è scomposto in oggetti che interagiscono
  - modello data-flow, dove il sistema è scomposto in modelli funzionali che trasformano input in output (modelli pipeline)
- Normalmente ogni decisione rispetto alla concorrenza dovrebbe essere posticipata fino alla fase di implementazione dei moduli

Modelli a oggetti

- Si struttura il sottosistema come un insieme di oggetti con accoppiamento lasco e interfacce ben definite
- La scomposizione orientata ad oggetti significa identificare le classi degli oggetti, gli attributi (campi) e le operazioni (metodi)
- Quando vengono implementati, gli oggetti sono istanze di queste classi e qualche modello di controllo è usato per coordinare i metodi

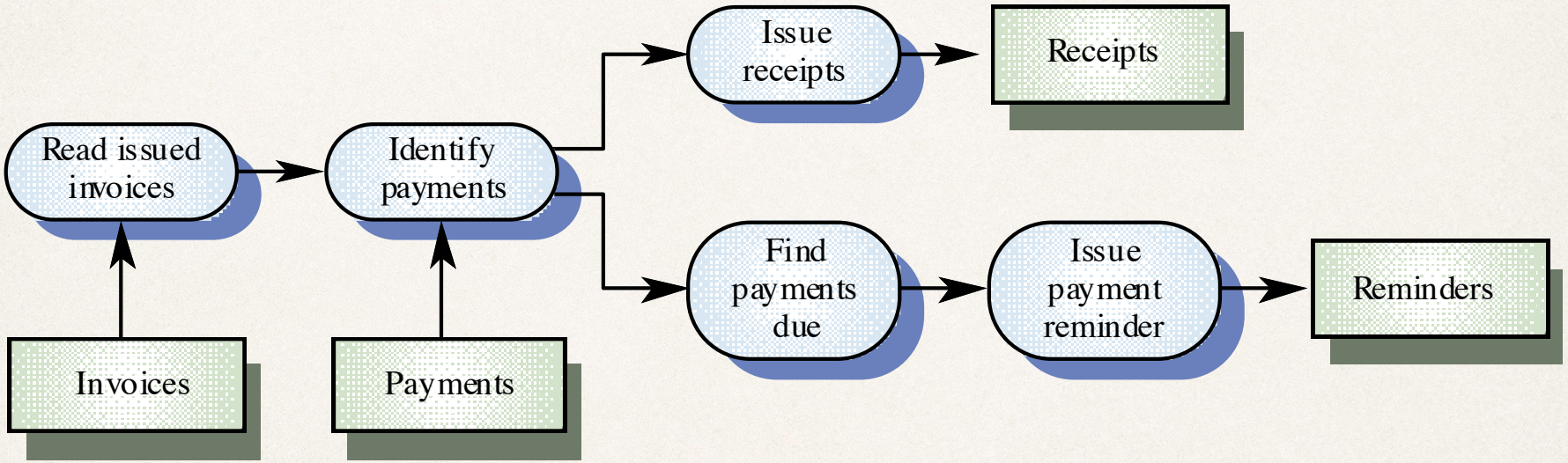
Esempio: sistema fatturazione



Modelli “data-flow”

- Trasformazioni funzionali che producono un output a partire da un input
- Possono riferirsi ad un modello a “pipe” e a filtri (pensate alla shell di unix)
- Se c’è un’unica trasformazione sequenziale, è un modello batch sequenziale: molto usato nei sistemi di gestione dei dati
- Non si presta bene a sistemi interattivi

Esempio: sistema fatturazione



Modelli di controllo (tra sottosistemi)

- Descrivono il modo con cui fluisce il controllo tra i sottosistemi
  - controllo centralizzato
    un sottosistema ha la responsabilità del controllo globale e avvia e disattiva i sottosistemi
  - controllo basato a eventi
    ogni sottosistema può rispondere a eventi generati da altri sottosistemi o dall'ambiente esterno

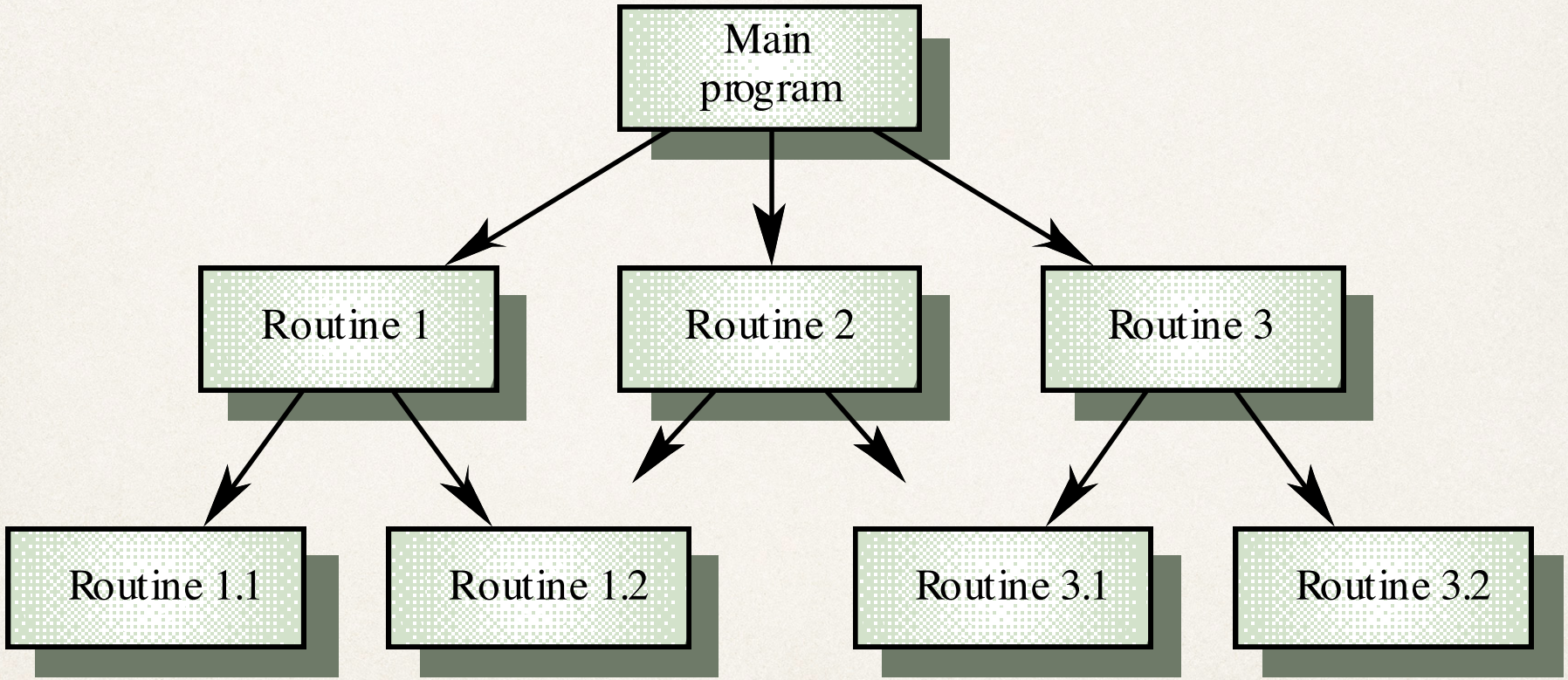
Controllo centralizzato

- Un sistema ha la responsabilità di gestire il controllo complessivo
- modello *"call-return"*
  - gerarchia di procedure, il controllo inizialmente parte dalla routine alla base della gerarchia e si sposta verso il basso. Questo tipo di controllo si può applicare a sistemi sequenziali
- modello *"manager"*
  - una componente del sistema controlla l'interruzione, l'inizio e il coordinamento degli altri processi. Applicabile a sistemi concorrenti. Può essere implementato in sistemi sequenziali come un blocco "case"

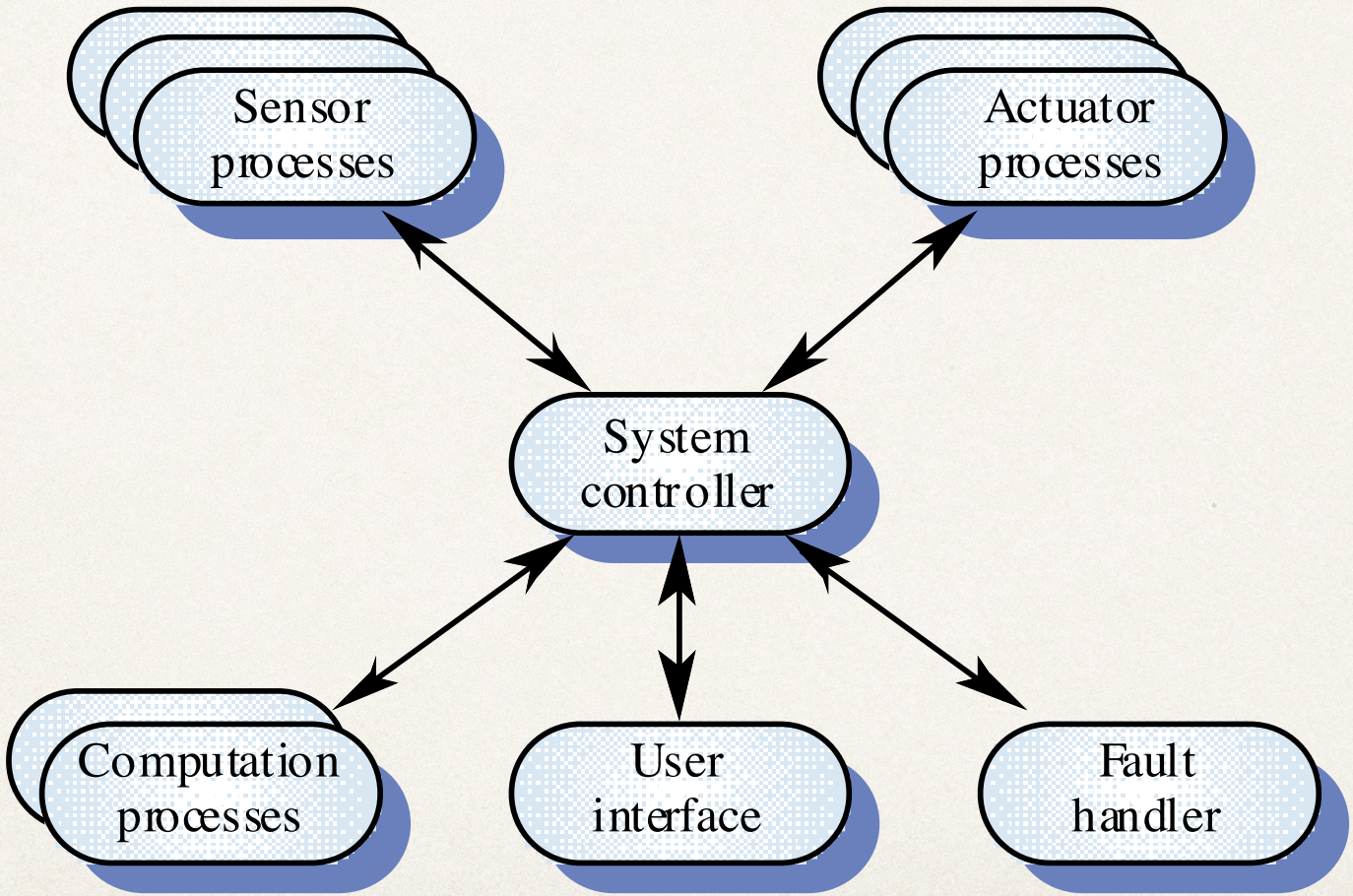
Modello call-return

- Si basa su una struttura gerarchica in cui un programma principale richiama una serie di procedure
- Il caso più classico è quello di una struttura a subroutine annidate
- Le varie componenti possono anche essere attive su nodi distribuiti: in questo caso l'architettura viene detta "a chiamata di procedure remote"
- Partendo da questo schema architetturale si sono evolute le architetture orientate agli oggetti

Esempio di call-return



Esempio di modello manager



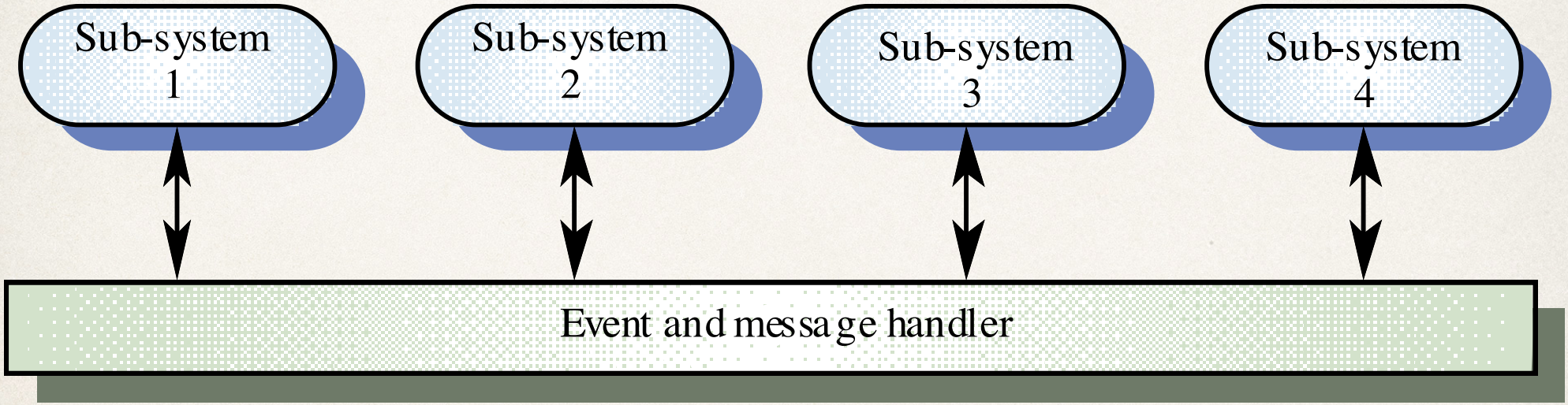
Modello basato su eventi

- E' guidato da eventi generati dall'esterno, dove la temporizzazione degli eventi è fuori dal controllo dei sottosistemi che gestiscono l'evento
- Due modelli "event-driven" principali
  - modello "broadcast": un evento è trasmesso a tutti i sottosistemi; ogni sottosistema in grado di gestire l'evento può farlo
  - modello "interrupt-driven": usato nei sistemi real-time dove le interruzioni sono raccolti da un gestore di interruzioni e passate a un componente responsabile per processarle

Modello “broadcast”

- E’ utile per integrare diversi sistemi collegati in rete
- Ciascun sottosistema si registra per riceve particolari tipi di evento: quando questi eventi si verificano, il controllo viene passato ai sottosistemi registrati
- I sottosistemi decidono gli eventi di interesse; il gestore degli eventi deve solo registrare l’interesse dei sottosistemi per certi tipi di eventi e avvisarli quando ne accade uno

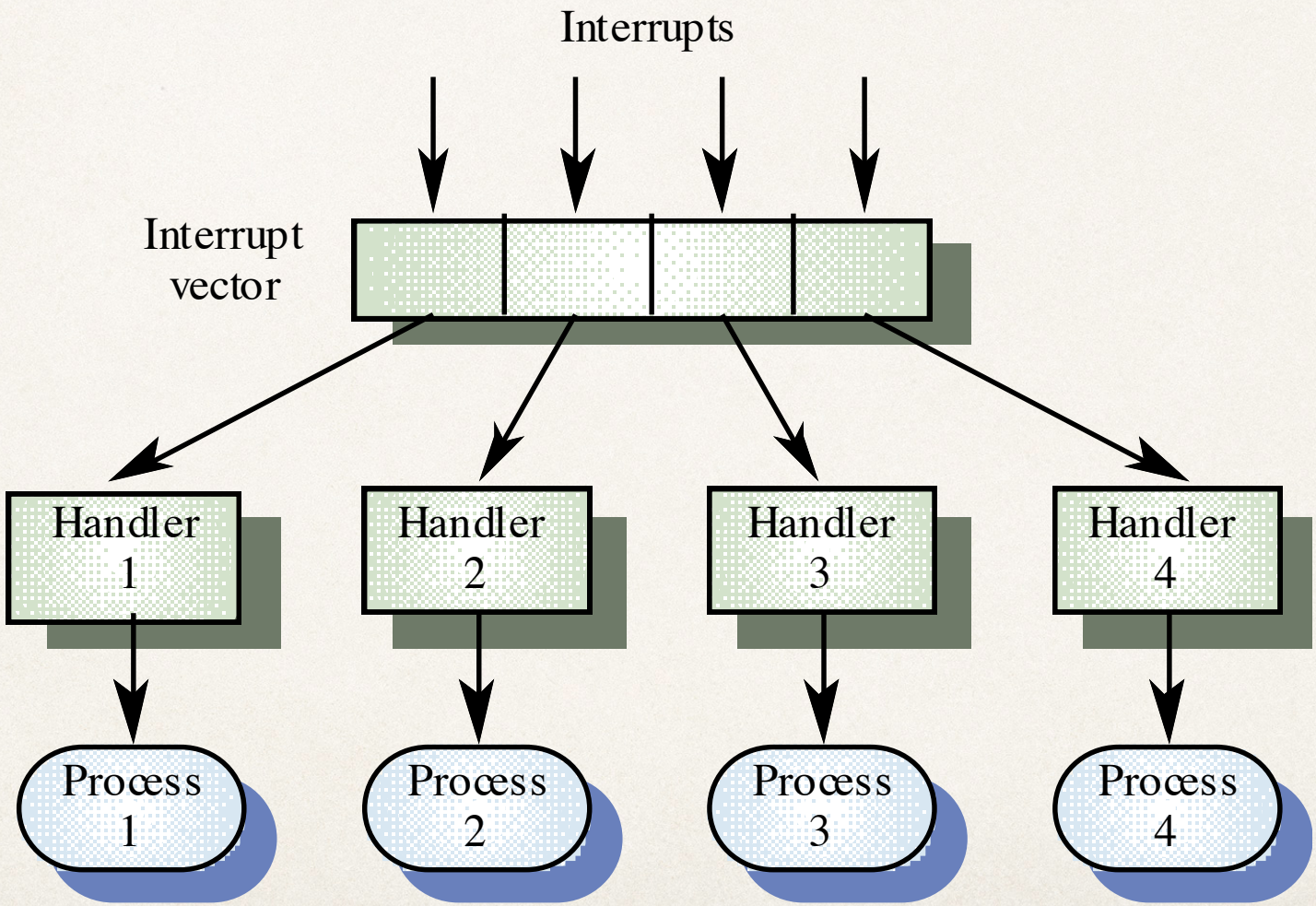
Esempio di broadcast



Modello “interrupt-driven”

- Usati in sistemi real-time dove la velocità di risposta a un evento e la latenza sono essenziali
- Devono essere definiti tutti i tipi di interruzioni noti, e a ciascun tipo occorre associare un modulo capace di gestire quel particolare tipo di interruzione
- Ogni tipo è associato a una locazione di memoria e uno switch hardware lo trasferisce al suo gestore
- Consentono rapidità di risposta ma sono complicati da programmare e difficili da validare

Esempi di modello interrupt-driven



Riassumendo (2)

- Modelli di controllo **centralizzato**
  - call-return
  - manager
- modelli di controllo **basato su eventi**
  - broadcast
  - interrupt-driven

Organizzazione e raffinamento

- Spesso si ha una serie di alternative: come chiarirsi le idee per scegliere uno stile architetturale?
- Controllo
  - come viene gestito il controllo all'interno dell'architettura?
  - esiste una gerarchia di controllo? qual'è il ruolo dei componenti?
  - come viene condiviso il controllo tra i componenti?
  - il controllo è sincronizzato o i componenti sono asincroni?
  - .....
- Dati
  - i componenti come si passano i dati?
  - il data-flow è continuo o i dati passano saltuariamente?
  - qual'è la modalità di trasferimento dei dati? (singolarmente o globalmente)
  - ...

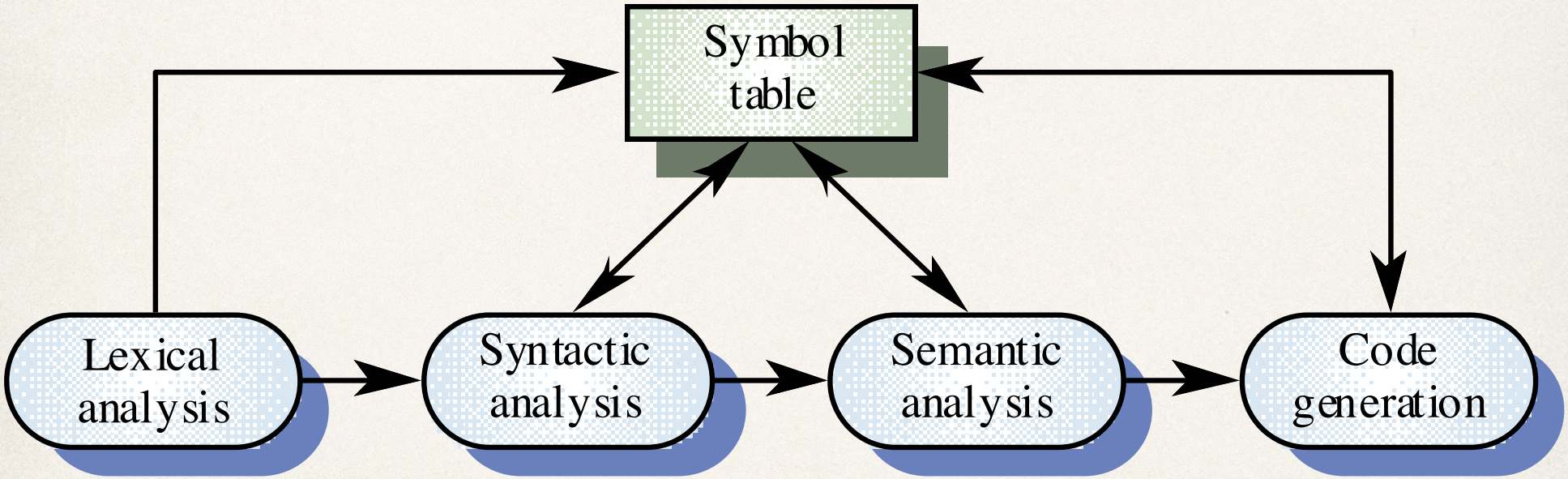
Architetture “domain-specific”

- Fino ad ora visti modelli generici
- Modelli di architettura di sistema che sono molto specifici a qualche dominio di applicazione
- Due tipi
  - modelli generici: sono astrazioni da un certo insieme di sistemi reali e contengono le caratteristiche principali di questi sistemi. Di solito sono modelli bottom-up
  - modelli di riferimento: sono modelli idealizzati, più astratti. Offrono informazioni su quella classe di sistemi e permettono di confrontare diverse architetture.
    Di solito sono modelli top-down

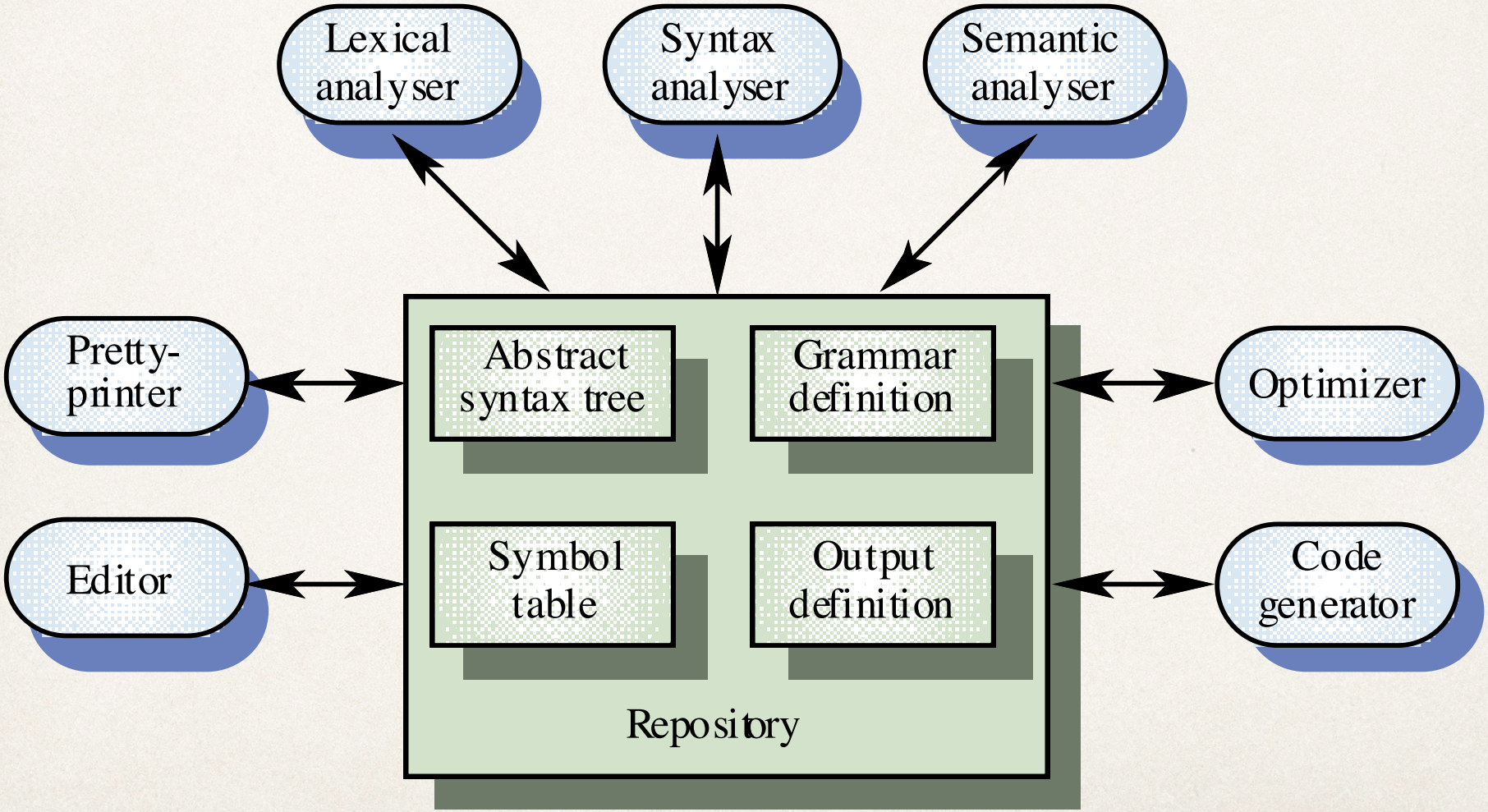
Modelli generici

- Consideriamo un compilatore. E' composto da:
  - analizzatore lessicale
  - tabella dei simboli
  - analizzatore sintattico
  - albero di derivazione
  - analizzatore semantico
  - generatore di codice
- Il modello generico di un compilatore può essere organizzato in base a diversi modelli architetturali

Compilatore basato su architettura a filtri



Compilatore basato sul modello a repository



Modelli di riferimento

- Derivano dallo studio del determinato dominio di applicazione piuttosto che dallo studio di sistemi esistenti
- Possono essere usati come base per implementare sistemi o per confrontare diversi sistemi
  - Hanno un ruolo di “standard” rispetto al quale valutare un sistema
- p.e.: il modello OSI è un modello di riferimento per i sistemi di comunicazione

Modelli di riferimento: OSI