Principi di progettazione di sistemi software

Corso di Ingegneria del Software

Anno Accademico 2012/13

La progettazione del sistema

- La progettazione è la fase in cui viene decisa e specificata la struttura del sistema
- Lo strumento fondamentale è la *divisione del sistema in sottosistemi* (divide et impera)
- Ciò avviene a vari livelli
- I sottosistemi sono chiamati anche *moduli* o *componenti*, anche a seconda del livello
- La struttura del sistema è spesso chiamata l'*architettura*

Architettura del sistema

- 3 livelli di architettura:
- Architettura hardware:
  - Uno o più calcolatori, tipo di rete, tipo di dispositivi…
- Architettura software di base:
  - Sistema operativo, protocolli di rete, linguaggi, …
- Architettura dell'applicazione:
  - I moduli in cui è scomposta l'applicazione da realizzare

Architettura hardware

- **Macchina singola.** Un utente su un computer (es. MS Word)
- **Mainframe**. Un computer e molti utenti connessi via terminali.
- **Internet**. Uno o più server HTML e accesso da più utenti tramite “Web browser”.
- **Rete di calcolatori**. Molti computer e altri dispositivi collegati in rete
- **Embedded**. Uno o più processori collegati direttamente a sensori e attuatori
- **Parallelismo**. Molti processori che eseguono calcoli intensi in parallelo
- **Cloud**. Server e dati sono custoditi da un fornitore esterno su Internet

Architettura software di base

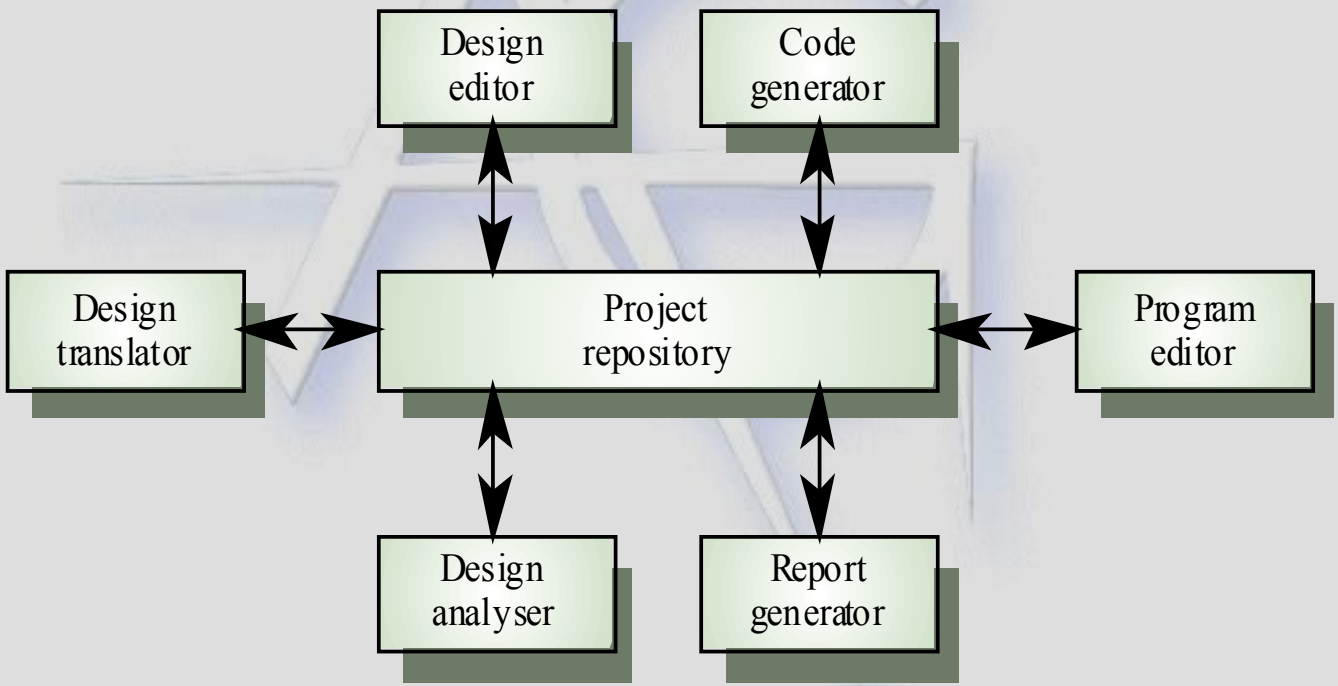
- I componenti da considerare sono:
  - Sistema operativo (Windows, Unix, Linux, Mac-OS…)
  - Protocollo di rete (TCP/IP, LAN proprietaria,…)
  - DBMS (file, relazionale, OO, XML, …)
  - Server Internet (Apache, MS IIS, WebLogic,…)
  - PDE (linguaggio + IDE)
  - Modello dei componenti (.net, Java EE, Corba,…)
  - Web services

Architettura software di base

- Vari modelli di architettura di base:
  - Central repository
  - Client/server
  - Basata su Internet
  - A strati (layered)

Central repository

- Utile se molti utenti devono agire sugli stessi dati
- Centrata su un *data store* (di solito un DBMS)
- Esempio: sistema CASE:



Modelli di *client*

- **Thin client**: quasi tutti i calcoli sono effettuati sul *data store*. I sistemi usati dall'utente per collegarsi hanno intelligenza molto limitata.
- **Intelligenza bilanciata**: sia il client che il data store effettuano calcoli. Il *client* effettua tutti i calcoli locali possibili, mentre il *data store* controlla le transazioni ed effettua tutte le elaborazioni che è ragionevole effettuare centralmente.
- **Fat client**: la maggiornaza delle elaborazioni sono effettuate sul client, ed il data store è un gestore di dati passivo. Talora il client esegue una copia locale dei dati.

Central repository

- Vantaggi:
  - Efficienza
  - Assenza di ridondanza dei dati
  - Gestione centralizzata dei dati (sicurezza, backup…). Le applicazioni non devono curarsene
  - Le applicazioni non devono “conoscersi”, perché scambiano i dati tramite il data store
- Svantaggi:
  - Il modello dei dati è comune a tutte le applicazioni, e può non essere ottimale per alcune di esse
  - L’evoluzione del modello dei dati è molto costosa
  - Se l'accesso ai dati è massiccio, ci possono essere problemi di carico
  - Molto sensibile all’indisponibilità del data store o della rete.

Central repository: esempio

- **Esempio monoutente: editor WYSWYG**
- **I dati sono contenuti in file (.doc, .odt...) nel file system**
- **L'applicativo presenta i dati all'utente tramite una finestra**
- **L'utente può creare e modificare i dati**
- **La cancellazione avviene al livello F.S.**

Central repository: esempio

- **Esempio multiutente: gestione magazzino**
- **I dati sono contenuti in un database, posto in un server dedicato**
- **Gli utenti eseguono l'applicazione sul proprio PC**
- **L'applicazione si collega al DB centrale, e presenta i dati all'utente tramite una finestra**
- **L'utente può creare, cancellare e modificare i dati**

Central repository: esempio

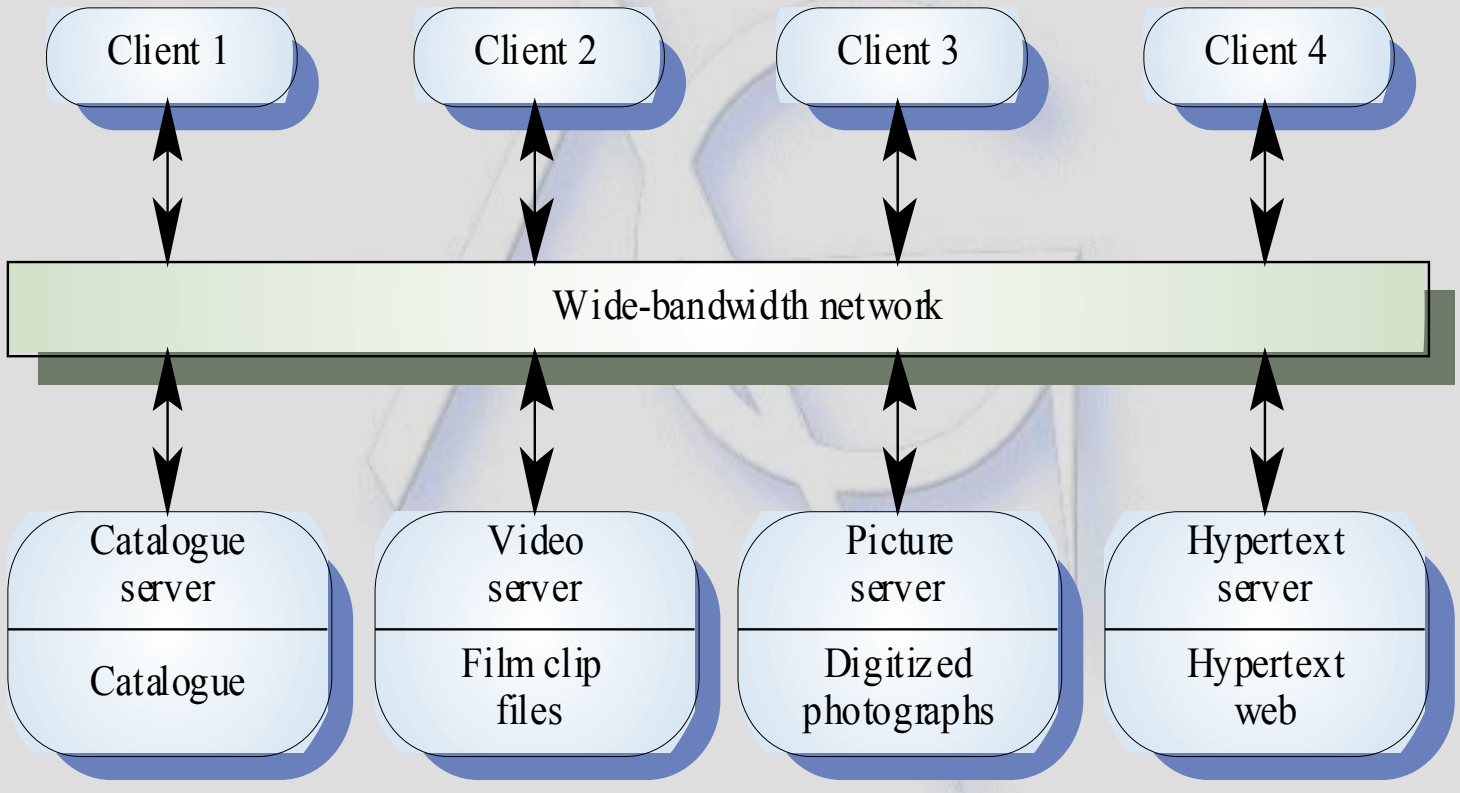
- **Esempio multiutente su Web: gestione magazzino**
- **I dati sono contenuti in un database, posto in un server dedicato**
- **L'applicazione è scritta su un Web server, e genera pagine Web usate per accedere ai dati**
- **Gli utenti si collegano sul proprio PC/TM usando un browser Web**
- **Ogni sessione crea un'istanza dell'applicazione, che si collega al DB e presenta i dati all'utente**
- **L'utente può creare, cancellare e modificare i dati**

Client-server

- **E' un modello *distribuito*, in cui le elaborazioni sono eseguite da più processori, che si scambiano i dati:**
- **Due tipi di componenti**
  - **I server, che offrono servizi di elaborazione: gestione dati, stampa, connessione a Internet, ecc.**
  - **I client, che usano tali servizi. Sono di solito PC o Wks connessi in rete su cui girano le applicazioni utente**
- **La distinzione tra client e server è logica e non fisica**
- **Il client conosce i servizi disponibili dei server**
- **I server non conoscono i client**

Client-server

- Libreria di film e immagini:



Client-server

- **Vantaggi:**
  - **Sistema distribuito e modulare**
  - **Se un server si blocca, il sistema continua a funzionare, seppur privo dei suoi servizi**
  - **E' facile cambiare la struttura interna di un server senza impatto sul resto del sistema**
- **Svantaggi:**
  - **Architettura complessa**
  - **Integrazione dei client coi server complessa, con le applicazioni che devono conoscere i servizi offerti**
  - **Costi di programmazione e gestioni alti**
  - **Problemi di scalabilità in caso di aggiunta di nuovi server e client**

Architettura basata su Internet

- **E' il modo moderno di realizzare il client-server**
- **Rete e comunicazioni basati su protocollo TCP-IP**
- **Uso su Internet o per Intranet aziendale**
- **I client sono i browser Web, eventualmente con opportuni *plug-in***
- **I server sono applicazioni interfacciate alla rete dal lato server, capaci di accettare connessioni**
- **Informazione scambiata spesso in formato XML**
- **Più recentemente: Service Oriented Architecture (SOA), basata su Web Services**

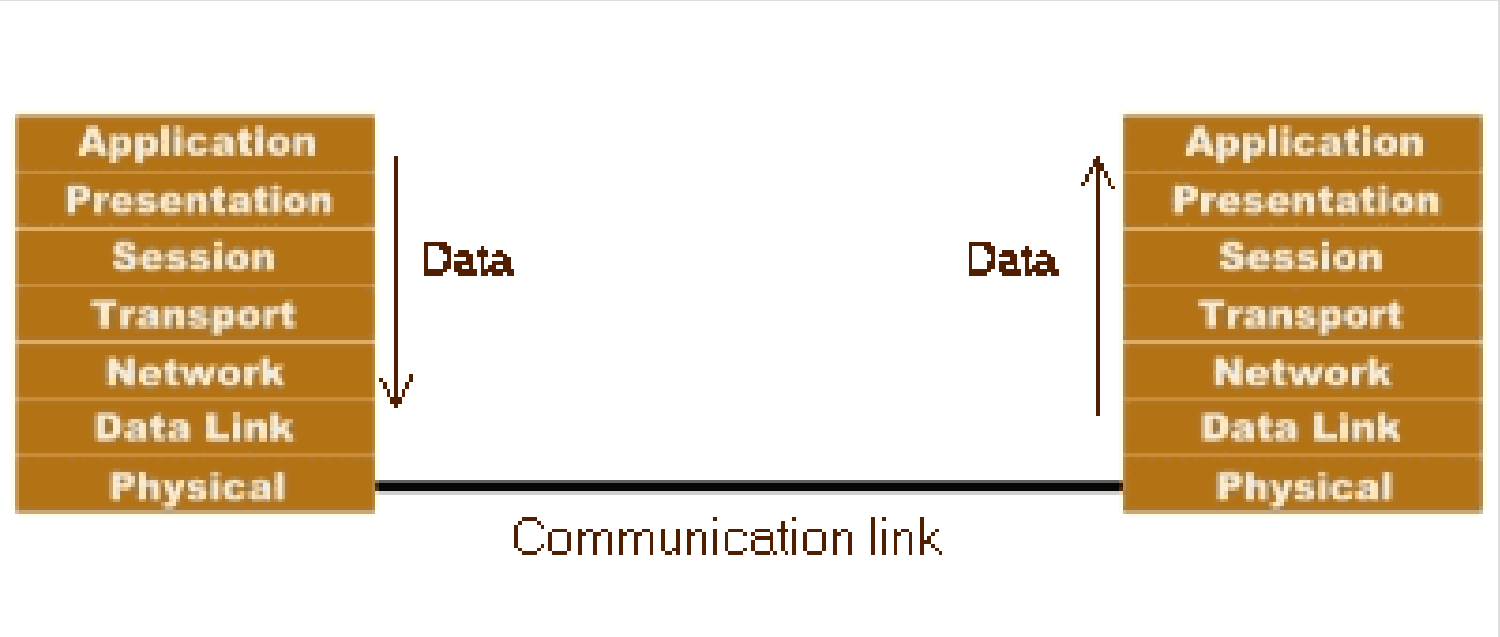
Architettura basata su Internet

- **Vantaggi:**
  - **Basato su standard aperti e di basso costo**
  - **Esistono applicazioni open-source per memorizzare e scambiare i dati**
  - **Formato per lo scambio dati basato sui caratteri**
  - **Facilmente scalabile**
- **Svantaggi:**
  - **Velocità di scambio dati può essere critica**
  - **Problemi di sicurezza per reti aperte all'esterno**
  - **Sviluppo di applicazioni complesso**

Architettura a strati (macchina astratta)

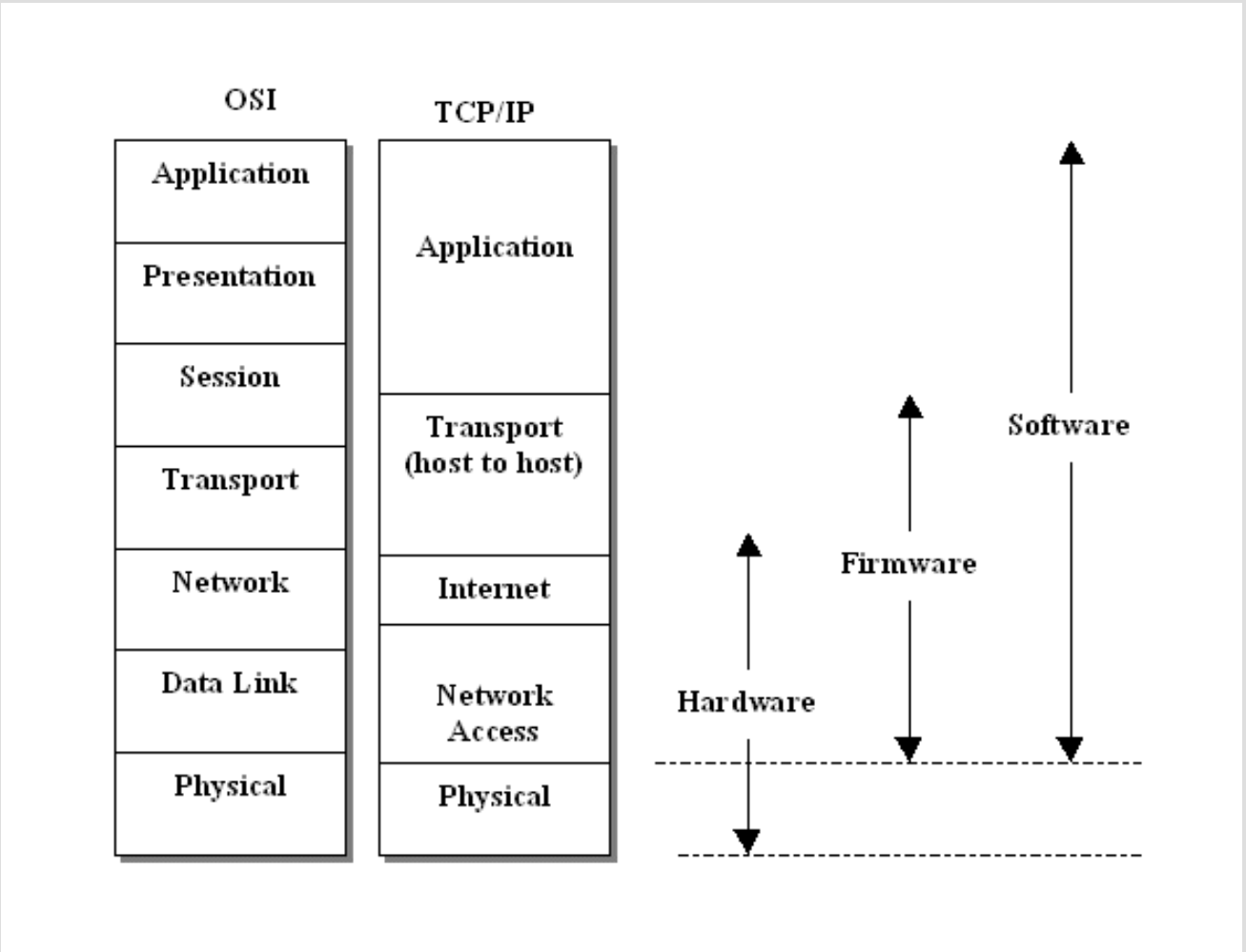
- **Il sistema è organizzato in vari *strati*, uno interno all'altro e ciascuno che fornisce dei servizi**
- **Gli strati più interni effettuano le operazioni più vicine all'hardware**
- **Gli strati più esterni forniscono servizi di livello sempre più elevato e astratto**
- **Composta da:**
  - **Servizi**
  - **Interfacce**
  - **Protocolli**

Esempio: l'architettura OSI



- *Open Systems Interconnection*: usata per definire sistemi di comunicazione via rete

OSI e TCP-IP


OSI
TCP/IP
Application
Presentation
Session
Transport
Network
Data Link
Physical
Application
Transport (host to host)
Internet
Network Access
Physical
Software
Firmware
Hardware

TCP-IP

Strato	Descrizione
Application Layer	Comprende vari protocolli di alto livello: • *File Transfer Protocol* (FTP) per lo scambio di file • *TELNET* per il *login* remoto. • *Simple Mail Transfer Protocol* (SMTP) per la posta elettronica. • *HyperText Transfer Protocol* (HTTP) per il World Wide Web. • Altri protocolli per news-group, gestione nomi dei domini e altre funzioni.
Transport Layer	Fornisce un servizio affidabile (TCP), per lo scambio di dati tra due nodi tramite pacchetti IP.
Internet Layer	Implementa una rete non affidabile per il trasporto di pacchetti IP.
Host-to-network Layer	Gestisce la connessione fisica del computer alla rete

Architettura dell'applicazione

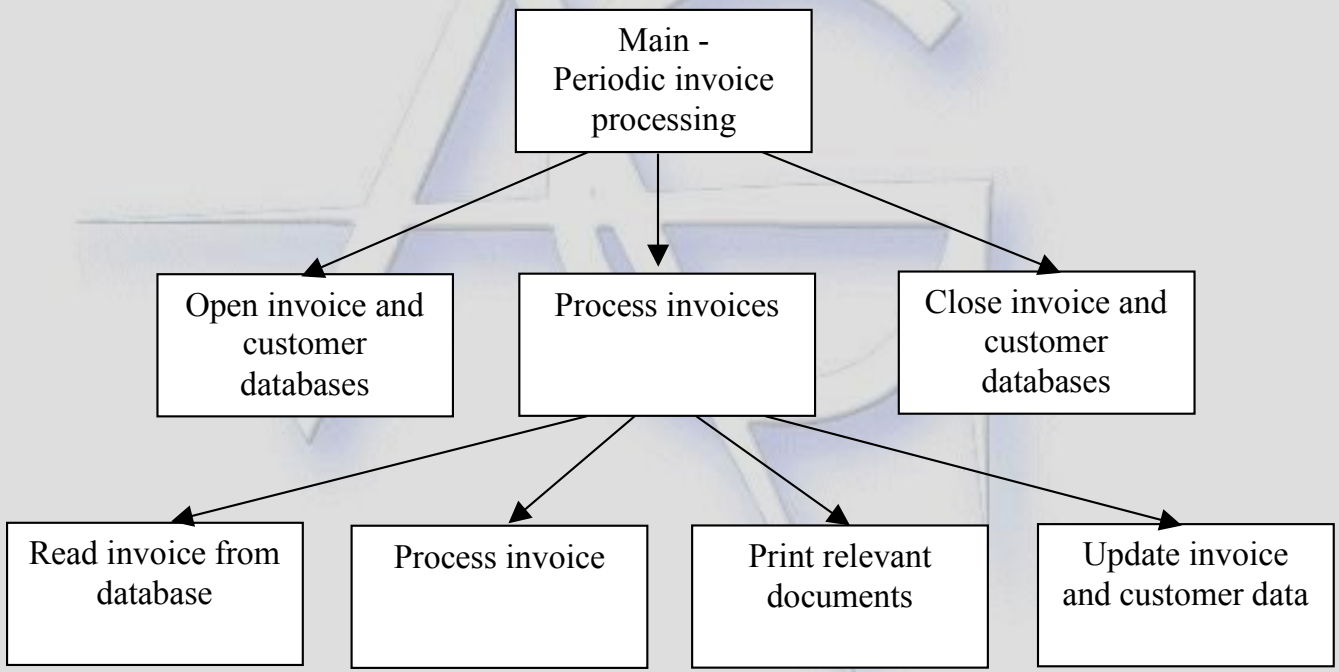
- Vi sono vari modelli di architettura dell'applicazione
- Una classe di modelli attiene al *controllo* dell'elaborazione
- Un'altra classe alla *scomposizione* dell'applicazione in moduli

Controllo centralizzato

- Un singolo modulo (controllore) ha il compito di controllare gli altri moduli
- Questi sono passivi, e rispondono solo al controllore
- Due tipi di controllo centralizzato:
  - Call and return
  - Manager model

Call and return

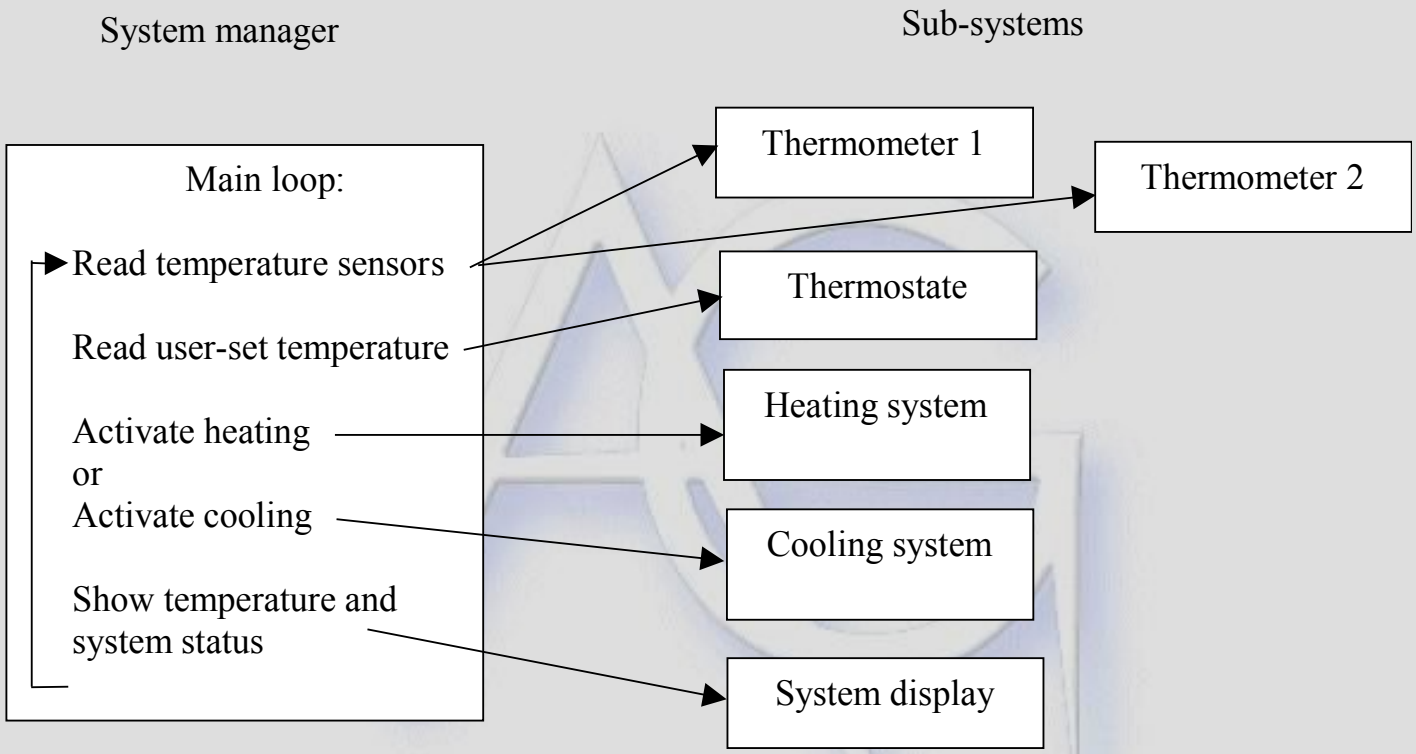
- Scomposizione in funzioni e sottofunzioni
- Il controllore è il *main()*:



Manager model

- Un modulo del sistema agisce come controllore
- Esso può essere una funzione, un oggetto o un sottosistema
- Usato spesso in sistemi di controllo senza vincoli di tempo critici
- Il manager ha un ciclo infinito in cui controlla lo stato dei sensori e dei moduli di input, ed esegue le opportune elaborazioni usando il proprio stato

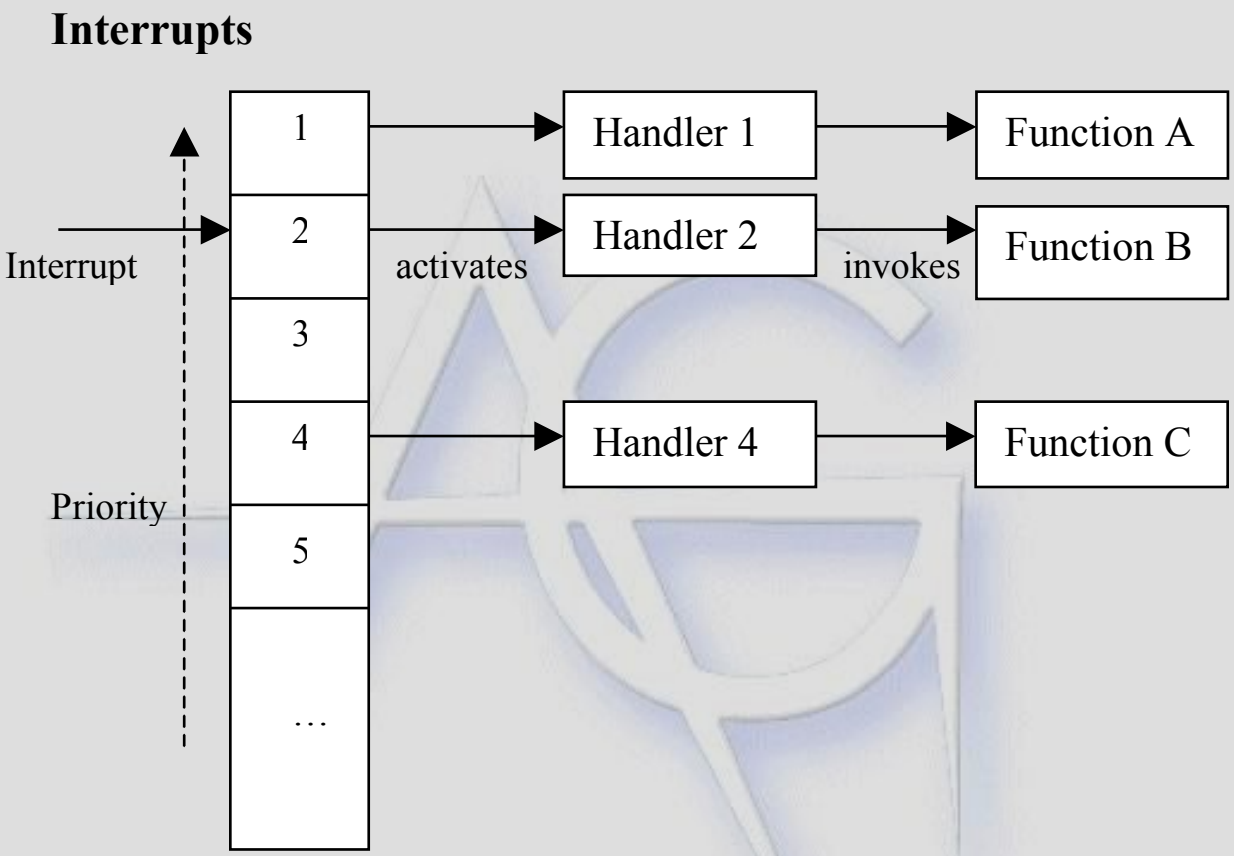
Un controllo di temperatura



Controllo ad eventi

- I vari moduli non chiamano direttamente altri moduli
- Essi si registrano nel sistema, associandosi a *eventi*
- Quando un evento succede, il modulo registrato per esso viene invocato
- Un evento può essere generato da un input esterno o da un'elaborazione interna
- A un evento possono essere associati dei dati, trasmessi al modulo che lo gestisce
- Modelli "broadcast" e ad "interrupt"

Un sistema ad *interrupt*



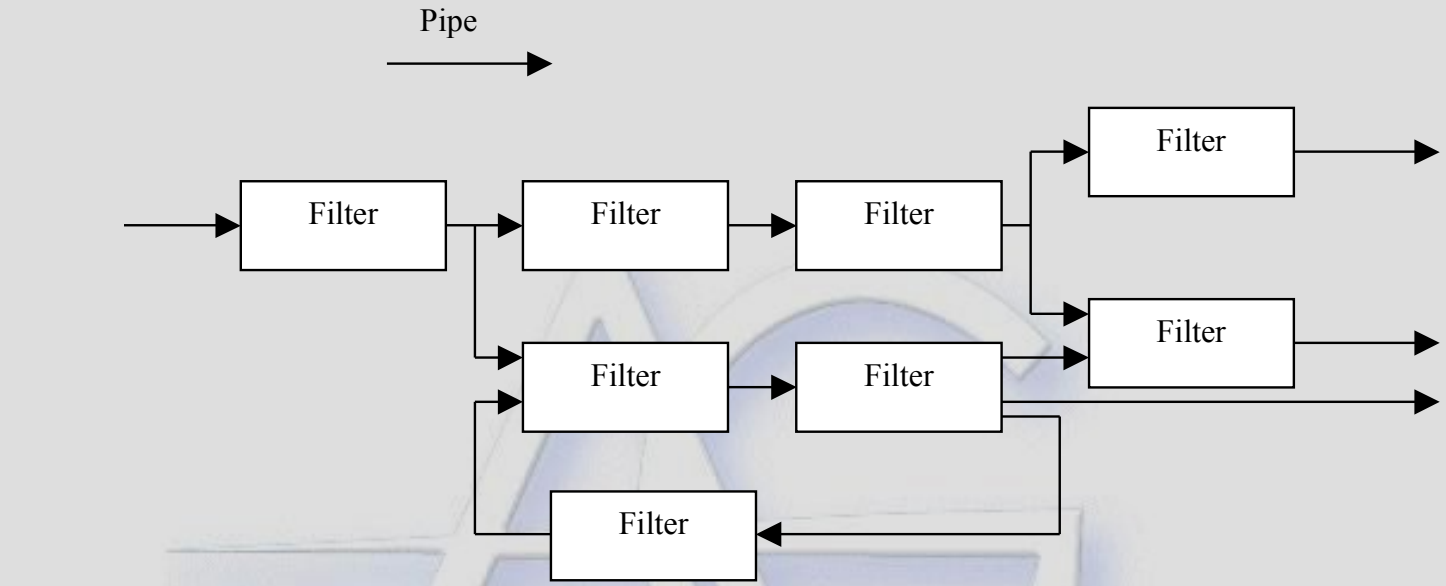
Scomposizione in moduli

- Una volta deciso il tipo di controllo, occorre scomporre un'applicazione complessa in sottosistemi, e poi in moduli
- Si tratta in effetti della stessa attività, solo eseguite a diversi livelli di dettaglio
- Vi sono due modelli fondamentali :
  - Il modello a flusso dei dati
  - Il modello ad oggetti

Modello a flusso dei dati

- Composto da canali per i dati e filtri (*pipes and filters*)
- Modellato tramite diagrammi DFD
- Se vi è una singola sequenza di moduli, si chiama *batch* sequenziale
- Orientato alla programmazione procedurale
- Le moderne applicazioni con GUI sono difficilmente rappresentabili

Pipes and filters



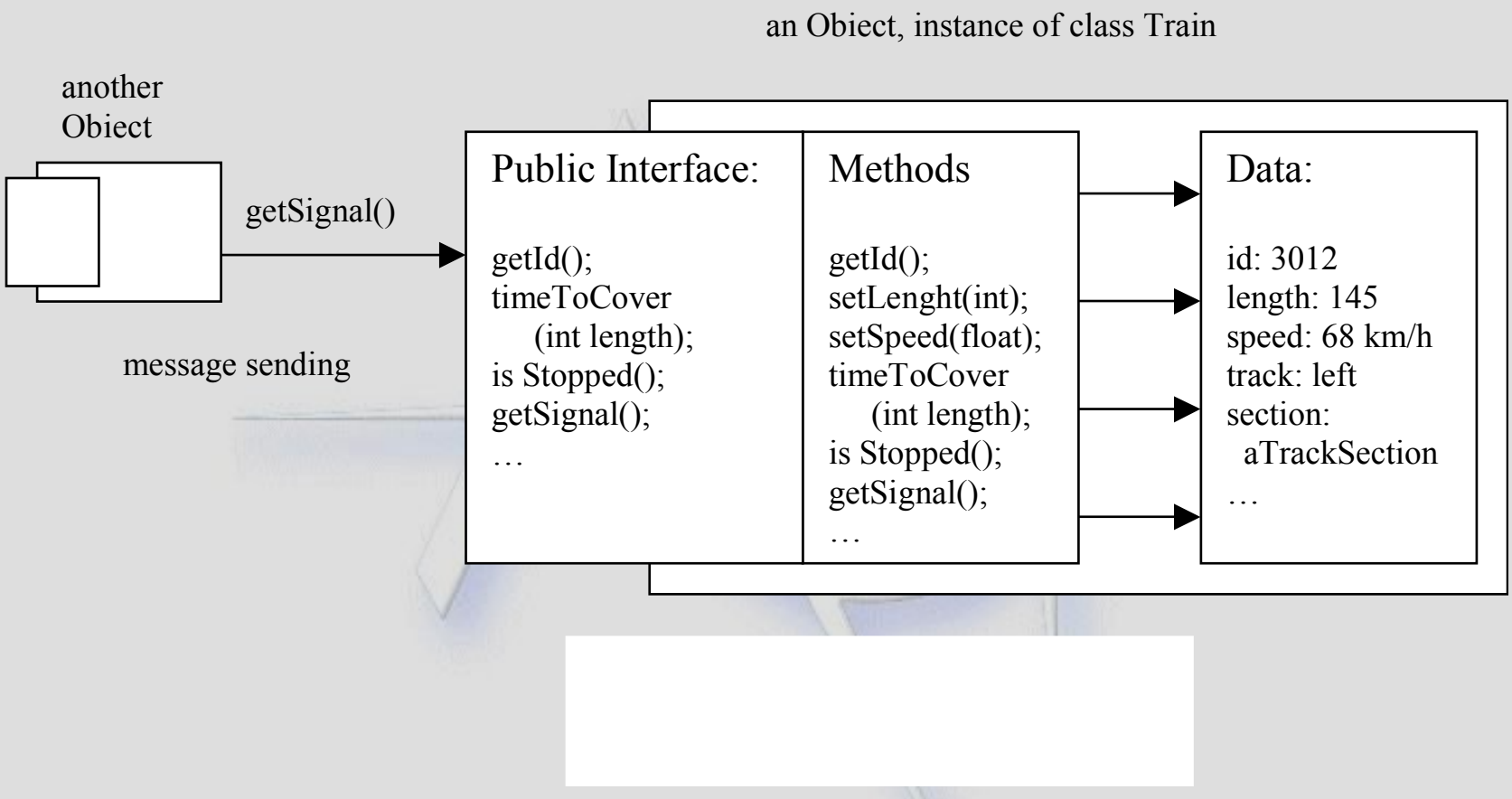
Generic data-flow architecture.

Batch sequential architecture.

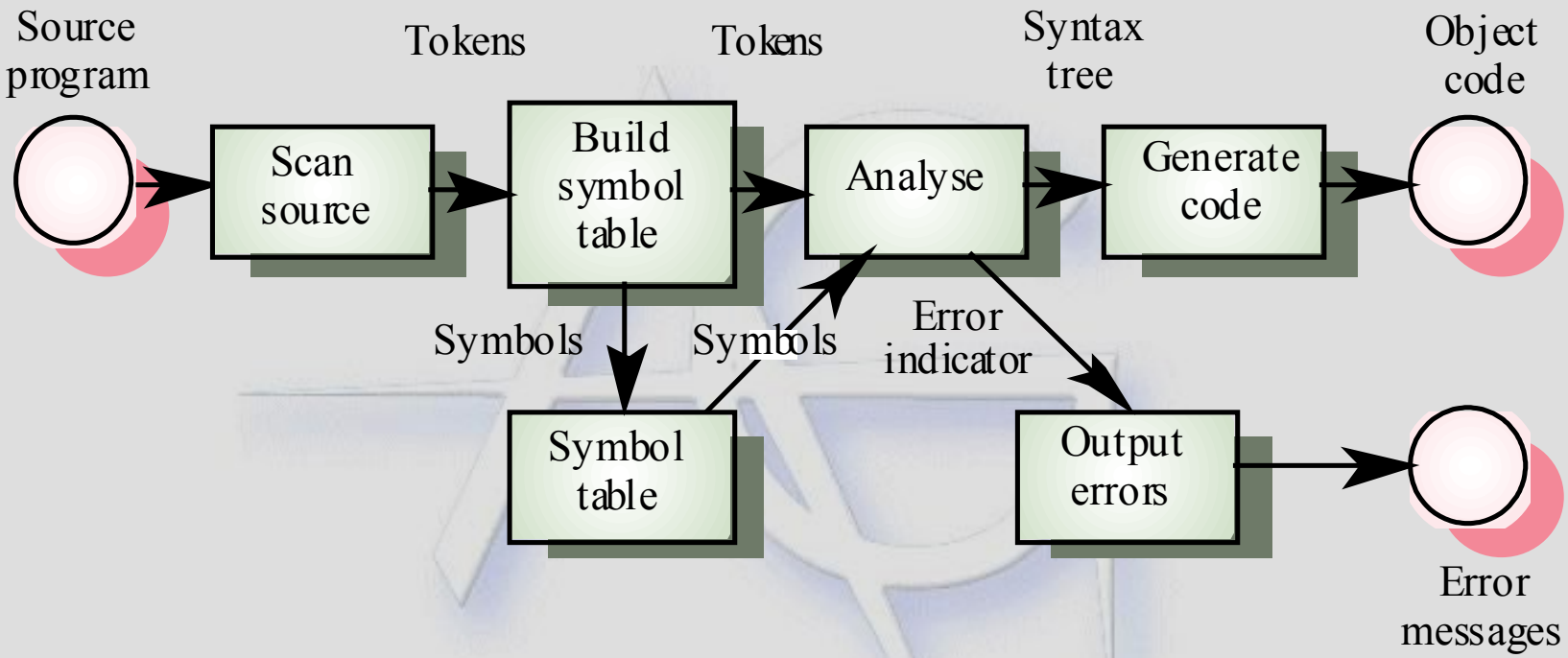
Modello a oggetti

- L'unità di scomposizione è l'*oggetto*
- Un oggetto è l'insieme di *dati*, *operazioni* (*metodi*) e *interfaccia esterna*
- Chiamare una data operazione su un oggetto si dice *inviare un messaggio all'oggetto*
- Per poter usare un oggetto, basta conoscerne l'interfaccia esterna
- Oggetti simili sono descritti da una *classe*
- Le classi possono essere in relazione di *ereditarietà*

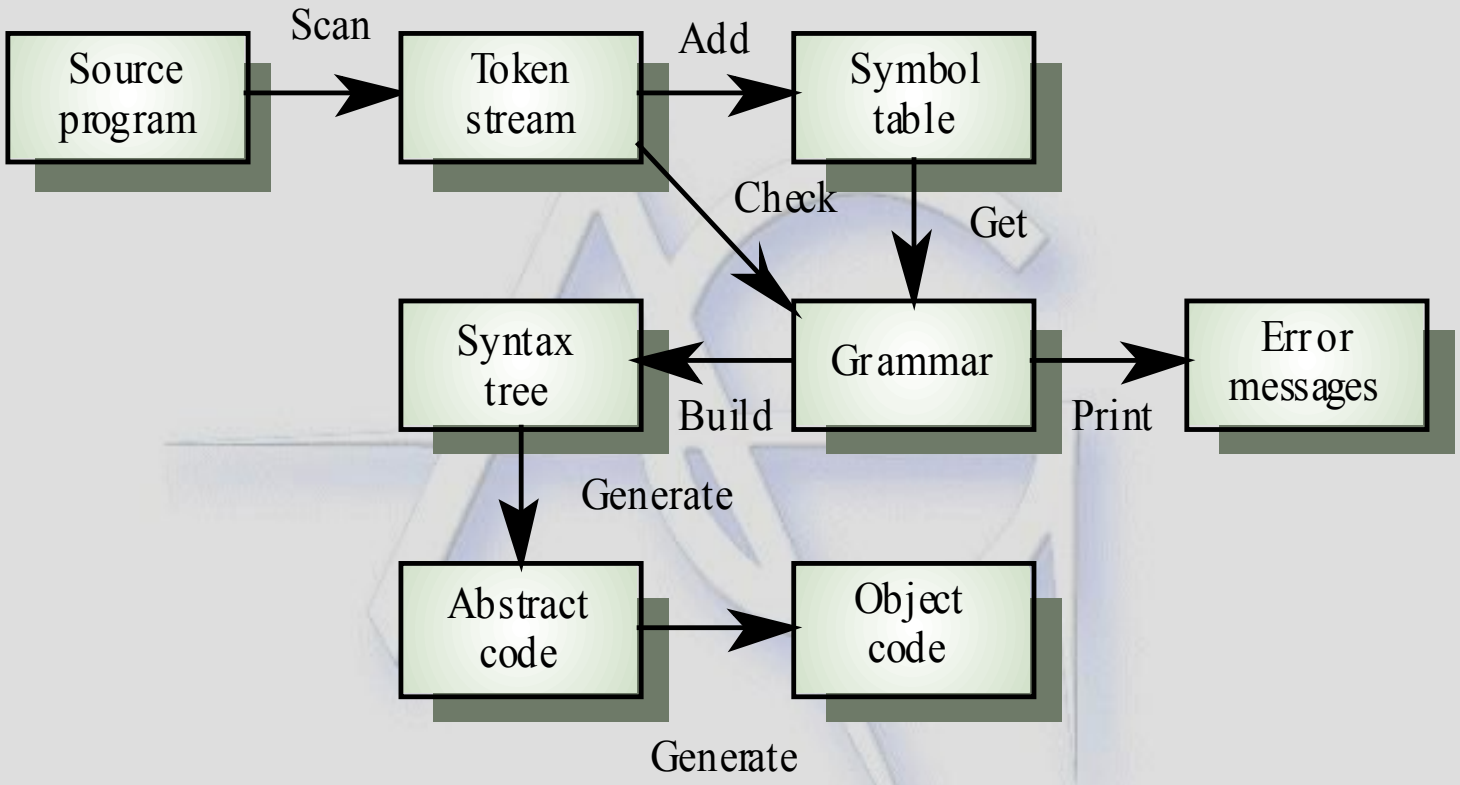
Un oggetto *Treno*



Vista funzionale di un compilatore



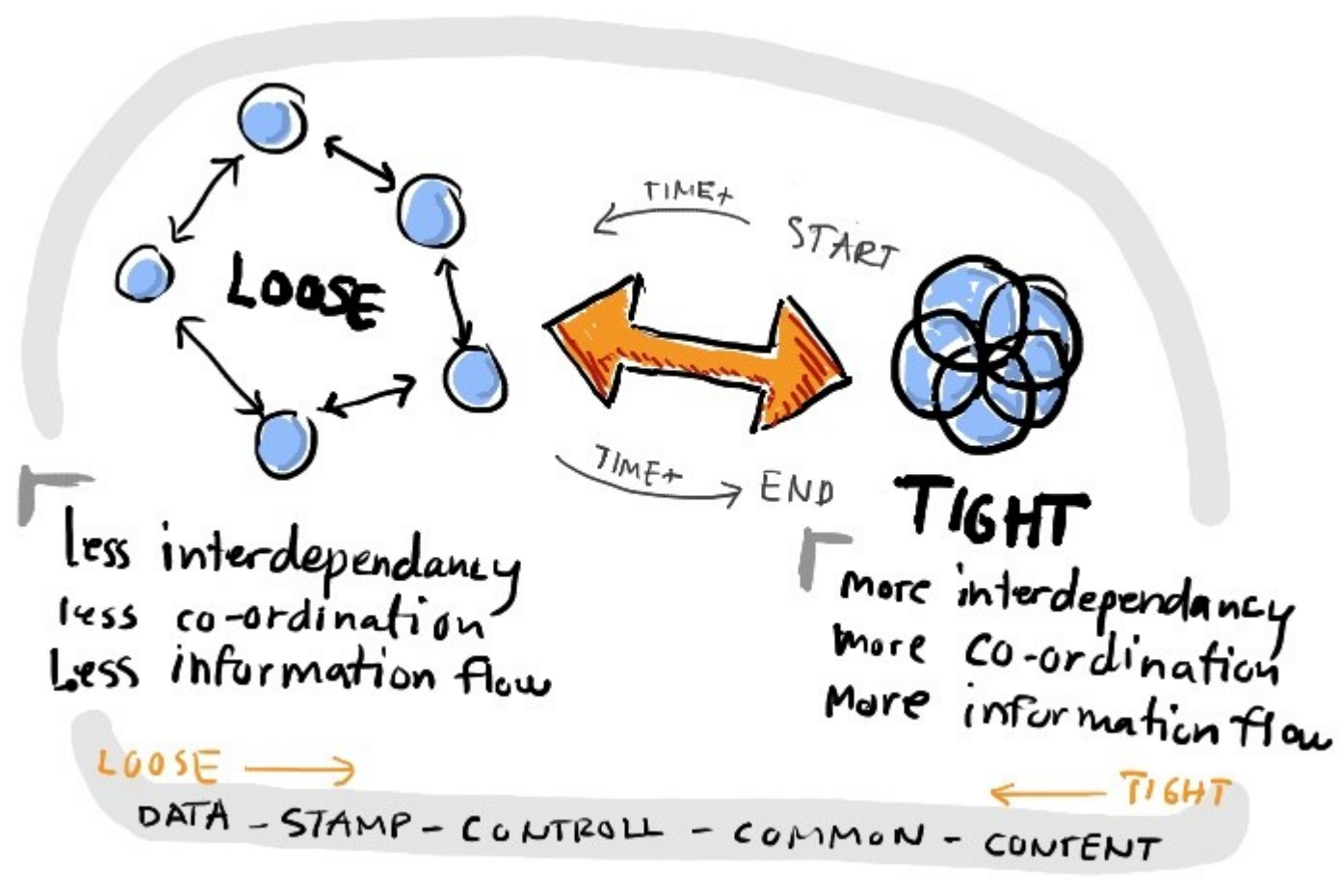
Vista ad oggetti di un compilatore



Principi di “buona”progettazione

- Occultamento dell’informazione
  - Un modulo deve occultare la propria implementazione
  - Cambiare un modulo senza cambiarne l’interfaccia non deve avere impatto sui moduli che lo usano
- Basso accoppiamento
  - Occorre minimizzare le interconnessioni e le dipendenze tra i vari moduli
- Coesione
  - Ogni modulo deve avere un singolo scopo
- Semplicità
  - Fa la cosa più semplice che possa funzionare
  - Non pensare a riusi futuri che non siano certi ora

Tipi di accoppiamento



Tipi di accoppiamento

- ***Message coupling: le procedure/metodi di un modulo chiamano le procedure/metodi di altri moduli***
- ***Data coupling: le procedure/metodi di un modulo hanno parametri, tipo di ritorno o variabili locali definiti in altri moduli***
- ***Subclass coupling: una classe definita in un modulo è sottoclasse di una classe definita in un altro modulo***

Tipi di accoppiamento

- ***Stamp (Data-structured) coupling: uno o più moduli o usano formati di dati e/o di comunicazione specifici***
- ***External coupling: uno o più moduli usano caratteristiche specifiche del compilatore, del SO o sono interfacciati a dispositivi specifici***
- ***Common coupling: due o più moduli condividono dati globali***
- ***Content coupling: uno o più moduli accedono direttamente a dati contenuti in altri moduli***

Coesione

- ***Funzionale: raggruppamento di funzionalità che contribuiscono tutte a un compito ben definito***
- ***Sequenziale: raggruppamento di funzionalità in cui l'input di una è l'output di un'altra***
- ***Comunicazionale: funzionalità raggruppate perché operano sugli stessi dati***
- ***Procedurale: raggruppamento di funzionalità chiamate in sequenza precisa***

Coesione

- ***Temporale: funzionalità raggruppate perché chiamate insieme quando eseguite***
- ***Logica: raggruppamento di funzionalità solo logicamente accoppiate (ad es. tutte le funzionalità di I/O)***
- ***Coincidentale: raggruppamento arbitrario di funzionalità (ad es. tutte quelle più usate)***