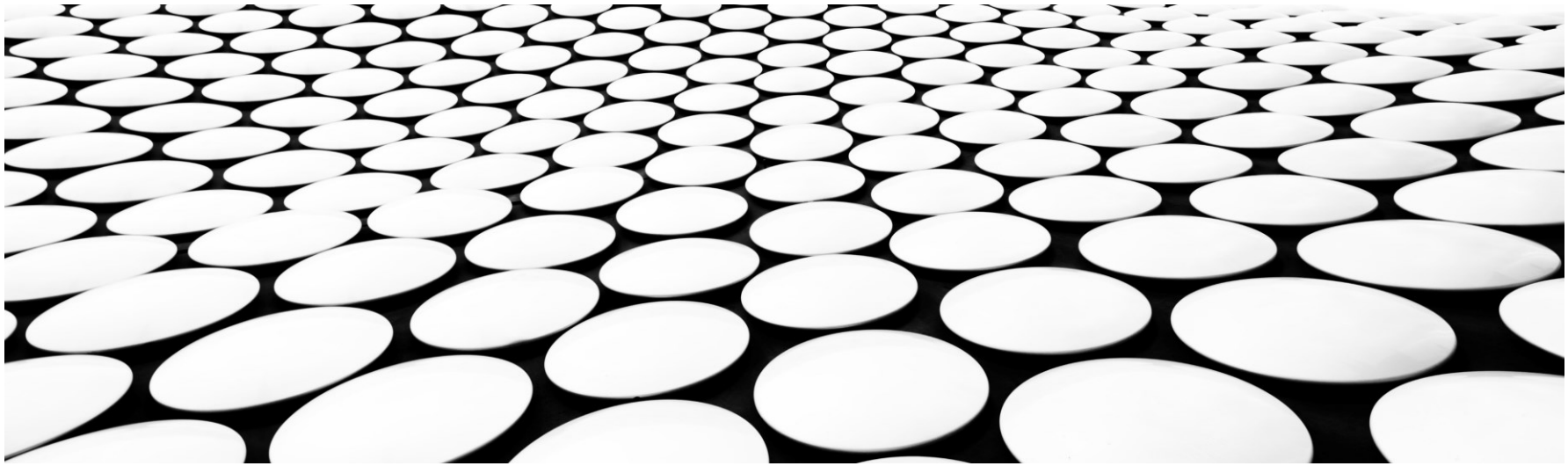
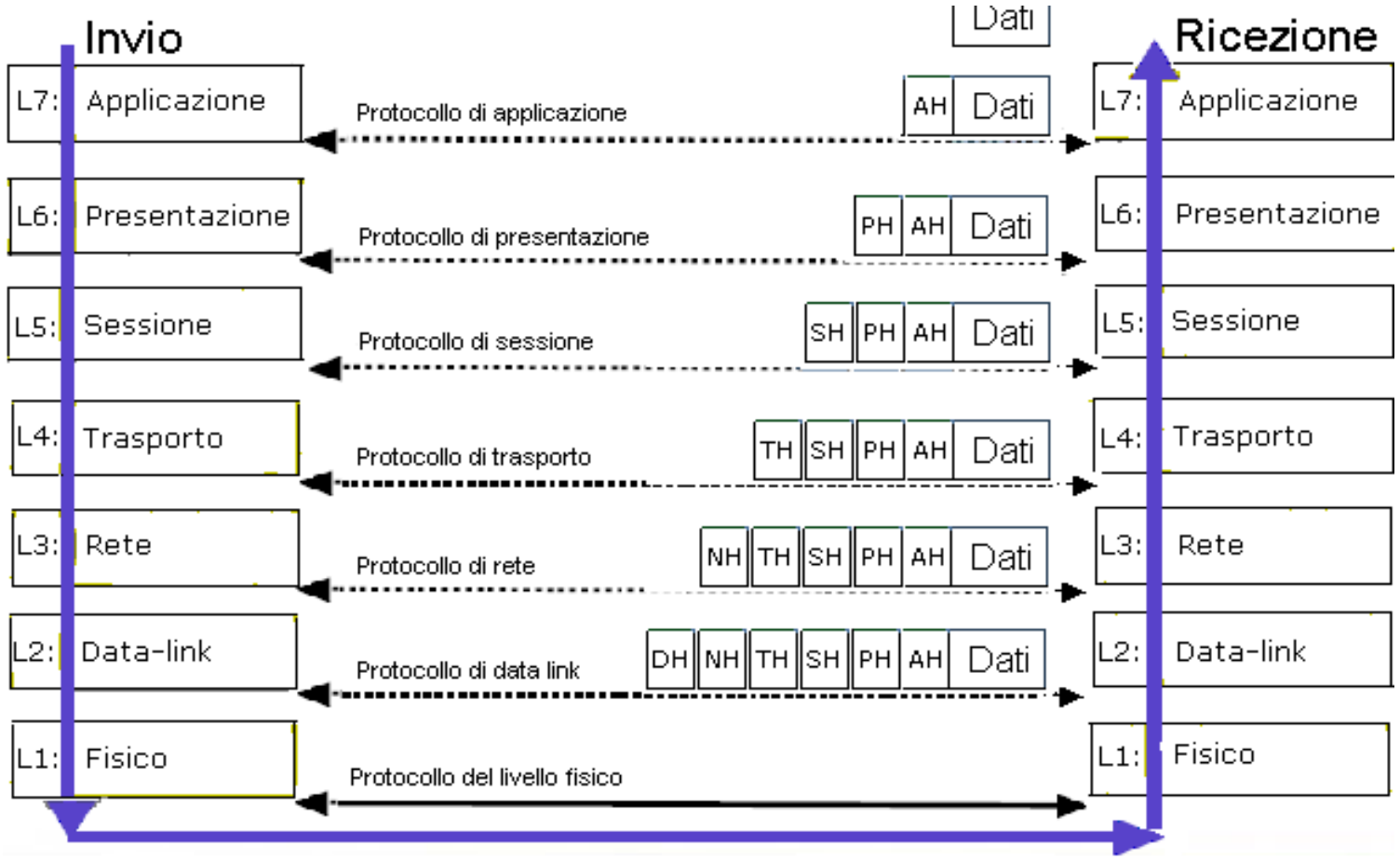
PARADIGMA CLIENT-SERVER

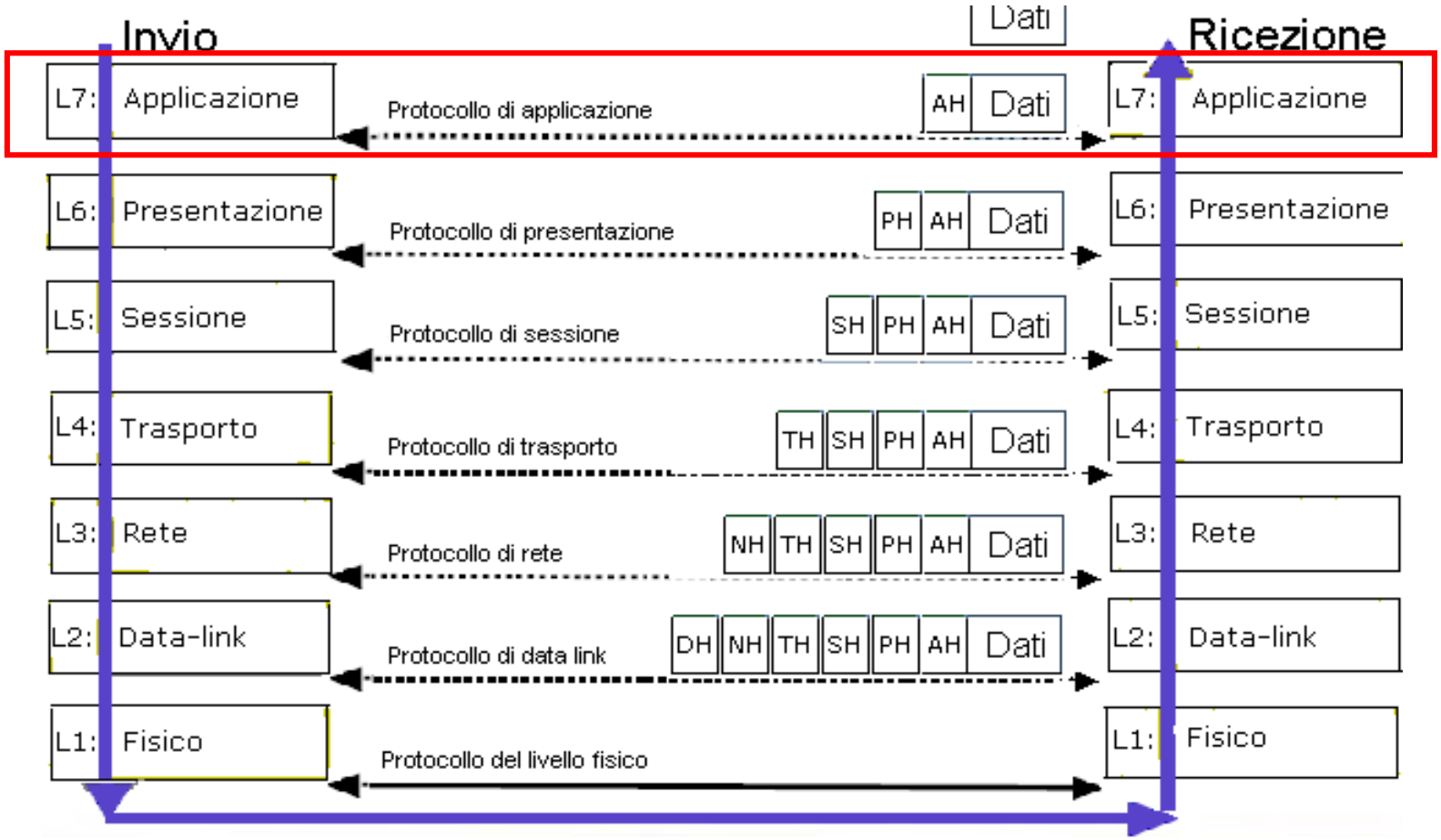
VINCENZO CALABRÒ



Standard ISO/OSI



Standard ISO/OSI



Livello applicativo

- Il livello OSI più vicino all'utente
  - Sia il livello applicativo che l'utente interagiscono direttamente con l'applicativo software
- Interagisce con l'applicativo software che implementa il componente di comunicazione
  - Applicazioni sono out of scope del modello OSI
- Funzionalità del livello applicativo includono
  - Identificare i partner della comunicazione
  - Determinare la disponibilità di risorse
  - Sincronizzare la comunicazione
  - Quality of service, autenticazione, privacy

Livello applicativo

- Identificazione dei partner di comunicazione
  - Il livello applicativo identifica e controlla la disponibilità dei partner di comunicazione per un'applicazione che deve trasmettere dati
- Disponibilità di risorse
  - Il livello applicativo decide se le risorse di rete disponibili sono sufficienti per la comunicazione
- Sincronizzazione della comunicazione
  - Le comunicazioni tra applicazioni richiedono cooperazione che è gestita dal livello applicativo

Protocolli di livello applicativo

- Tipo dei messaggi scambiati
  - Ad es., messaggi di richiesta e risposta
- Sintassi dei tipi di messaggi
  - Quali campi e come sono distribuiti
- Semantica dei campi
  - Significato delle informazioni nei campi
- Regole che guidano i processi nell'invio di e risposte a messaggi

Protocolli di livello applicativo

- Public-domain protocol
  - Definiti in RFC
  - Permettono interoperabilità
  - Ad es., HTTP, SMTP
- Protocolli proprietari
  - Ad es., KaZaA

Protocolli di livello applicativo

- Esempi
  - Domain Name System (DNS)
  - Hypertext Transfer Protocol (HTTP)
  - File Transfer Protocol (FTP)
  - Simple Mail Transfer Protocol (SMTP)

Applicazioni di rete

- E-mail
- Web
- Instant messaging
- Remote login
- P2P file sharing
- Multi-user network games
- Streaming stored video clips
- VOIP
- Real-time video conference
- Massive parallel computing

Livello di trasporto

- Perdita di dati
  - Alcune app (ad es., audio) possono tollerare alcune perdite
  - Altre app (ad es., file transfer, telnet) richiedono un trasferimento affidabile al 100%
- Timing
  - Alcune app (ad es., VOIP, interactive game) richiedono basso delay per essere utilizzabili
- Bandwidth
  - Alcune app (ad es., multimedia) richiedono una minima quantità di banda per essere efficaci
  - Altre app (“elastic app”) usano qualunque banda ottengano

Livello di trasporto

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100’s msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100’s msec
instant messaging	no loss	elastic	yes and no

Protocolli del livello di trasporto

- TCP
  - **Orientato alla connessione***:* richiesto setup tra processo client e server
  - **Trasporto affidabile** tra processo mittente e destinatario
  - **Controllo del flusso***:* mittente non inonderà il ricevente
  - **Congestion control***:* mittente considera stato della rete quando deve inviare nuovi messaggi
  - **Non fornisce***:* timing, garanzie su banda minima

Protocolli del livello di trasporto

- UDP
  - **Trasferimento dati non affidabile** tra processo mittente e destinatario
  - **Non fornisce**: setup della connessione, affidabilità, controllo del flusso, controllo della congestion, timing e garanzie sulla banda

Protocolli del livello di trasporto

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

Paradigma di comunicazione client-server

Paradigma client-server

- Base concettuale per (quasi) tutti i sistemi distribuiti
- Un programma inizia la comunicazione (client) verso un altro programma (server)
- Nota: applicazioni ‘‘peer-to-peer’’ internamente usano il paradigma client-server

Definizioni

- Client
  - Programma applicativo
  - Contatta il server
  - Crea e invia una richiesta
  - Attende una risposta
- Server
  - Di solito un programma specializzato che offre un servizio
  - Aspetta una richiesta
  - Crea la risposta
  - Invia la risposta

Server: caratteristiche

- Di solito implementato come un programma applicativo
  - Processo, processo utente, task
  - Possono essere eseguiti su sistemi che supportano comunicazioni TCP/IP
    - Sistemi timesharing
    - Personal computer

- Server con lo stesso servizio possono essere eseguiti sulla stessa o diverse macchine

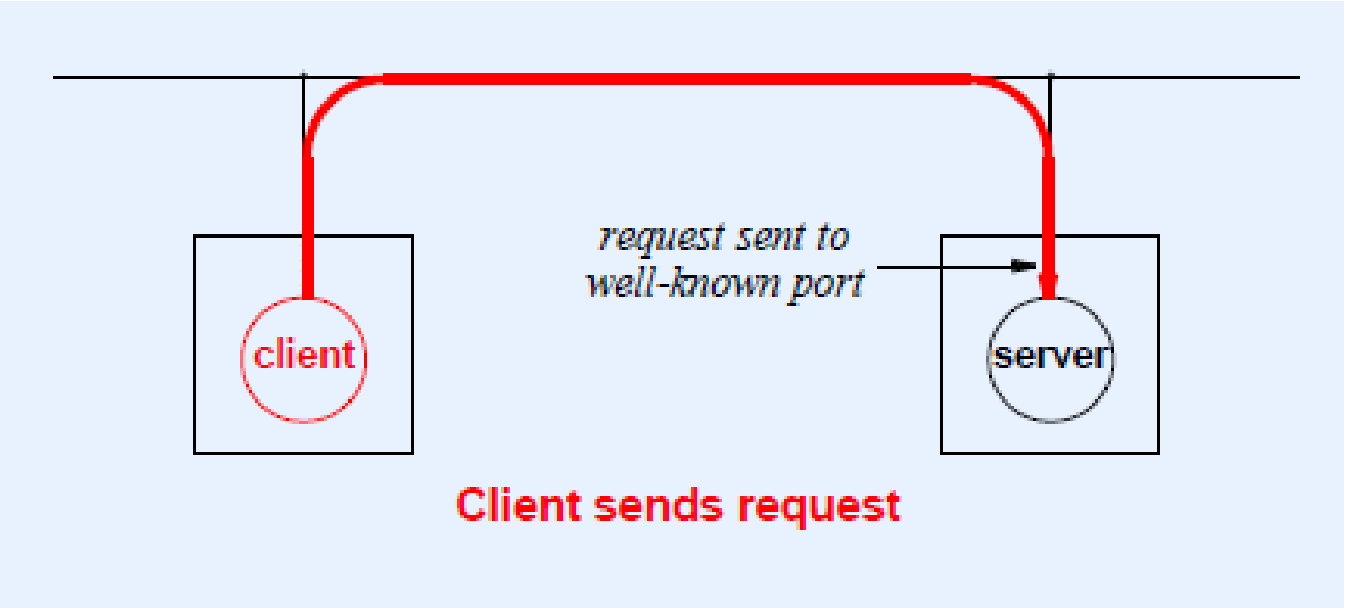
Persistenza del server

- Un server inizia l'esecuzione prima che l'interazione con il client inizi e (di solito) continua ad accettare richieste e inviare risposte senza terminare mai

- Un client è un programma che fa richieste, aspetta risposte e (di solito) termina dopo aver usato il server un numero finito di volte

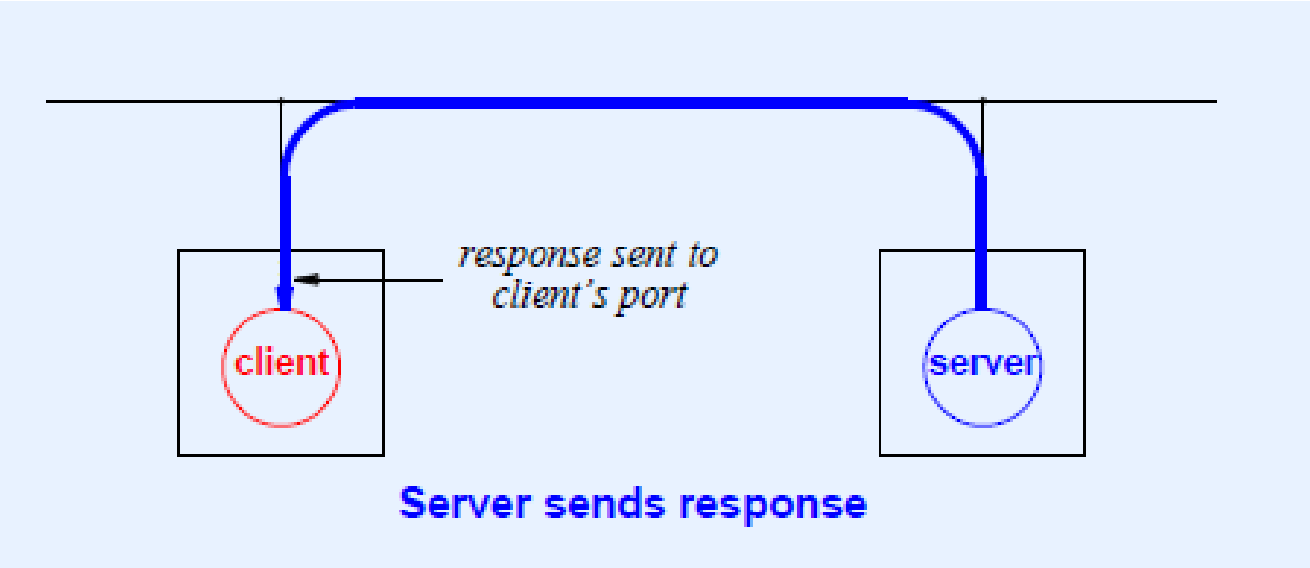
Paradigma client-server

- Client invia una richiesta attraverso la rete
  - Indirizzo IP + porta



Paradigma client-server

- Server invia una risposta attraverso la rete
  - Indirizzo IP + porta



Porte di protocollo

- Un server aspetta una richiesta ad una porta riservata (well-known port) per il servizio che deve offrire

- Un client alloca una porta non riservata, arbitraria e inutilizzata per le sue comunicazioni

Esempio: UDP Echo Server

- La forma più semplice di interazione client-server
  - Consegna di datagrammi non affidabili per la comunicazione tra client e server

- UDP Echo Server process
  - Riserva una porta UDP
  - Processo a 3 passi
    - Aspetta un datagramma
    - Inverte indirizzo sorgente e destinazione
    - Ritorna il datagramma ricevuto

Esempio: UDP Echo Client

- Alloca una porta UDP non usata
- Invia un messaggio UDP
- Aspetta la risposta contenente lo stesso dato inviato

Altri esempi

- Time-of-day Server
  - Semplice
  - Richiesta e risposta in un singolo datagramma
  - Rappresentazione del tempo come i secondi dopo una certa epoca

- Web Server
  - Complesso
  - Scambio di pagine web

Client side

- Ogni programma applicativo può diventare un client
- Deve conoscere come raggiungere il server
  - Indirizzo IP
  - Numero di porta del protocollo
- Facile da sviluppare

Server side

- Trova la locazione del client attraverso la richiesta
- Può essere implementato all'interno di un programma applicativo o il sistema operativo
- Inizia l'esecuzione prima dell'arrivo della richiesta
- Assicura che il client sia autorizzato
- Implementa regole di protezione
- Gestisce richieste multiple e concorrenti
- Di solito difficile da progettare e implementare

Server concorrenti

- Esempi precedenti erano sequenziali
  - Sistema operativo accoda richieste concorrenti
- Per il server è più difficile
  - Deve gestire le richieste concorrenti
- Server ha due parti
  - Programma master
  - Uno o più programmi slave
  - Processi slave sono concorrenti

Server concorrenti

1. Aprono una well-known port
2. Aspettano la prossima richiesta del client
3. Se necessario, creano una nuova porta locale per il client
4. Creano un thread / processo che gestisce la richiesta (slave)
5. Continua con una *wait* (Passo 2)

Complessità dei server

- Richieste concorrenti non sono l'unica cosa che influenzano la complessità del server
- Il server deve implementare e far rispettare autorizzazioni e regole di protezione (ad es., controllo dell'accesso)
- Server eseguono con il più alto privilegio (read file system, access data, keep logs…)
- Server non possono semplicemente rispondere a tutte le richieste che arrivano dalla controparte

Complessità dei server

- Server (programmi) devono proteggere il sistema operativo e i suoi dati facendo rispettare politiche di accesso e di protezione
- Server devono autoproteggersi da richieste malformate
  - File Server sviluppato alla Purdue University
    - Richiedeva di aprire un /dev/tty che causava un abort del server
  - Worm sviluppato da uno studente alla Cornell University
    - Novembre 1988 Morris worm
    - 10% dei PC infettati

Complessità dei server

- Server più difficili da implementare dei client perché, nonostante siano implementati tramite programmi applicativi, devono gestire politiche di accesso e di protezione del sistema su cui vengono eseguiti e devono autoproteggersi da errori, fallimenti, attacchi

Conclusioni

- Paradigma client-server alla base delle applicazioni distribuite
- Server è un programma/processo specializzato e complesso che offre servizi
- Applicazioni arbitrarie possono diventare un cliente contattando un server e inviando una richiesta
- Server usualmente concorrenti