TCP e UDP: il livello trasporto dell'architettura TCP/IP

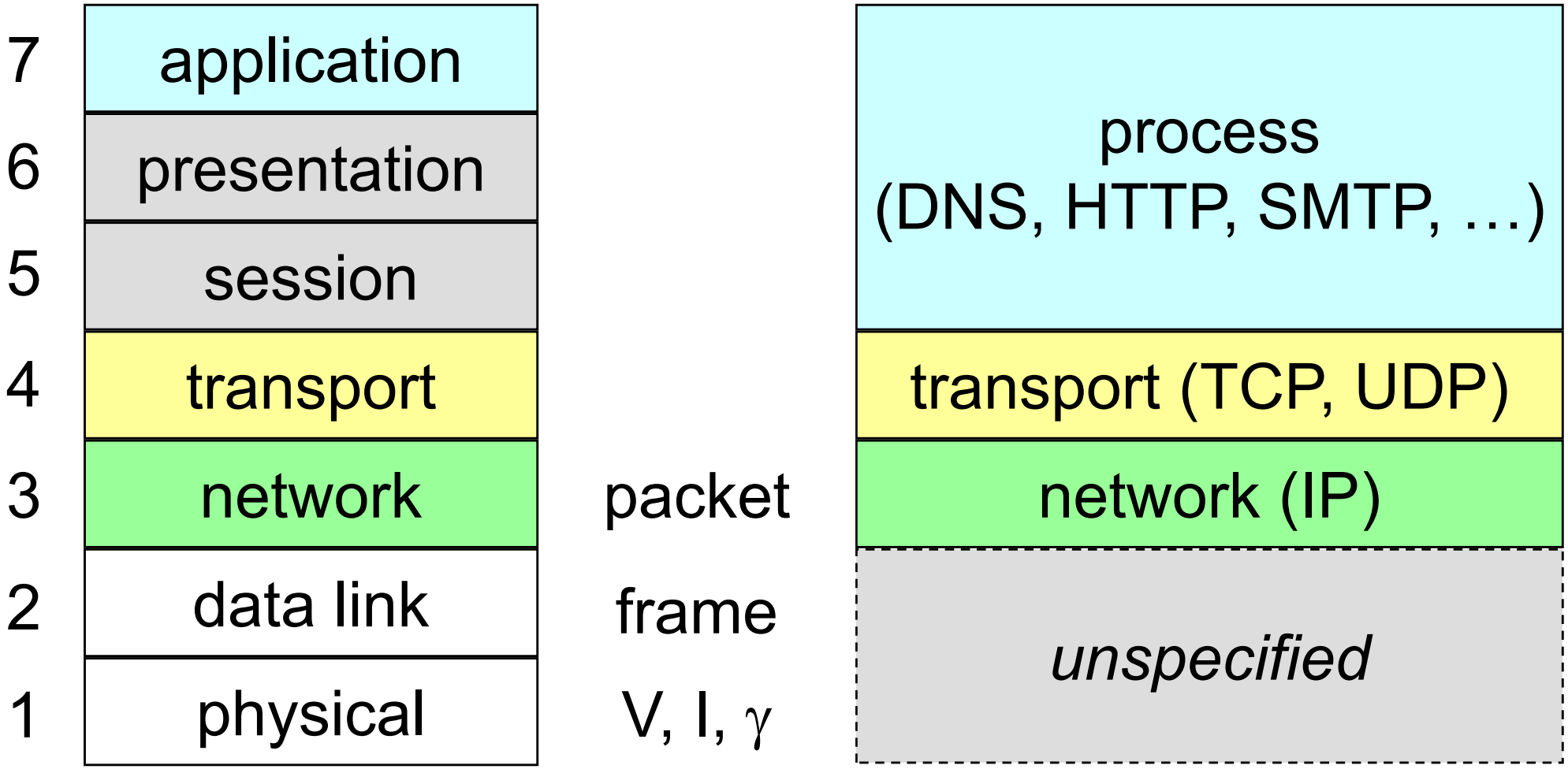
Antonio Lioy

< lioy@polito.it >

Politecnico di Torino

Dip. Automatica e Informatica

OSI vs. TCP/IP

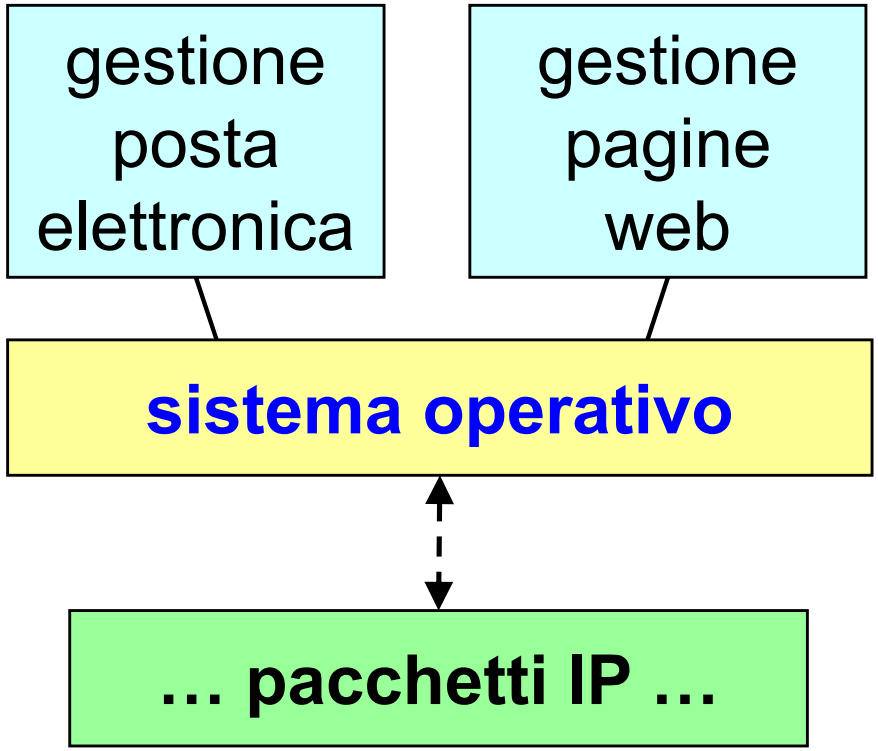


Transport layer

- **canale logico end-to-end**
  - utile per gli sviluppatori applicativi … che altrimenti dovrebbero risolversi da soli i problemi di IP (es. dimensione limitata, pacchetti persi, duplicati, fuori sequenza)
- **multiplexing/demultiplexing**
  - uso di un solo indirizzo di rete per inviare/ricevere da parte dei diversi processi applicativi di un host
- **controllo di flusso**
  - il mittente cerca di non intasare il ricevente
- **controllo di congestione**
  - il mittente cerca di non intasare la rete

Transport layer

- **il livello rete (L3) fa comunicare nodi di rete**
- **il livello trasporto (L4):**
  - fa comunicare processi
  - offre una API (Application Programming Interface) standard ai programmatori – i socket



Transport layer: protocolli

- **due protocolli L4 più usati:**
  - TCP
    - privilegia l'affidabilità (=garantire la ricezione dei dati o sapere che non sono stati ricevuti)
    - risolve (o almeno affronta) tutti i problemi
  - UDP
    - privilegia la latenza
    - risolve solo alcuni dei problemi, lasciando gli altri agli sviluppatori applicativi ed ai gestori di rete
- **ma ne esistono altri (es. SCTP = Stream Control Transmission Protocol, MPTCP = Multipath TCP)**

Velocità, latenza e throughput

- **velocità di rete V = massima quantità di dati per unità di tempo trasmissibili**
  - es. Ethernet 10M = 10 Mbps
- **banda B = frazione di V dedicata ad uno specifico flusso di rete**
  - $B \leq V$
- **latenza L = tempo che intercorre tra gli istanti di invio e ricezione di un dato**
  - $L = t_R - t_S$
- **throughput T = quantità di dati trasmessa nell'unità di tempo**
  - $T = |\ dati\ | / (\ t_R - t_S\ ) = D / L$

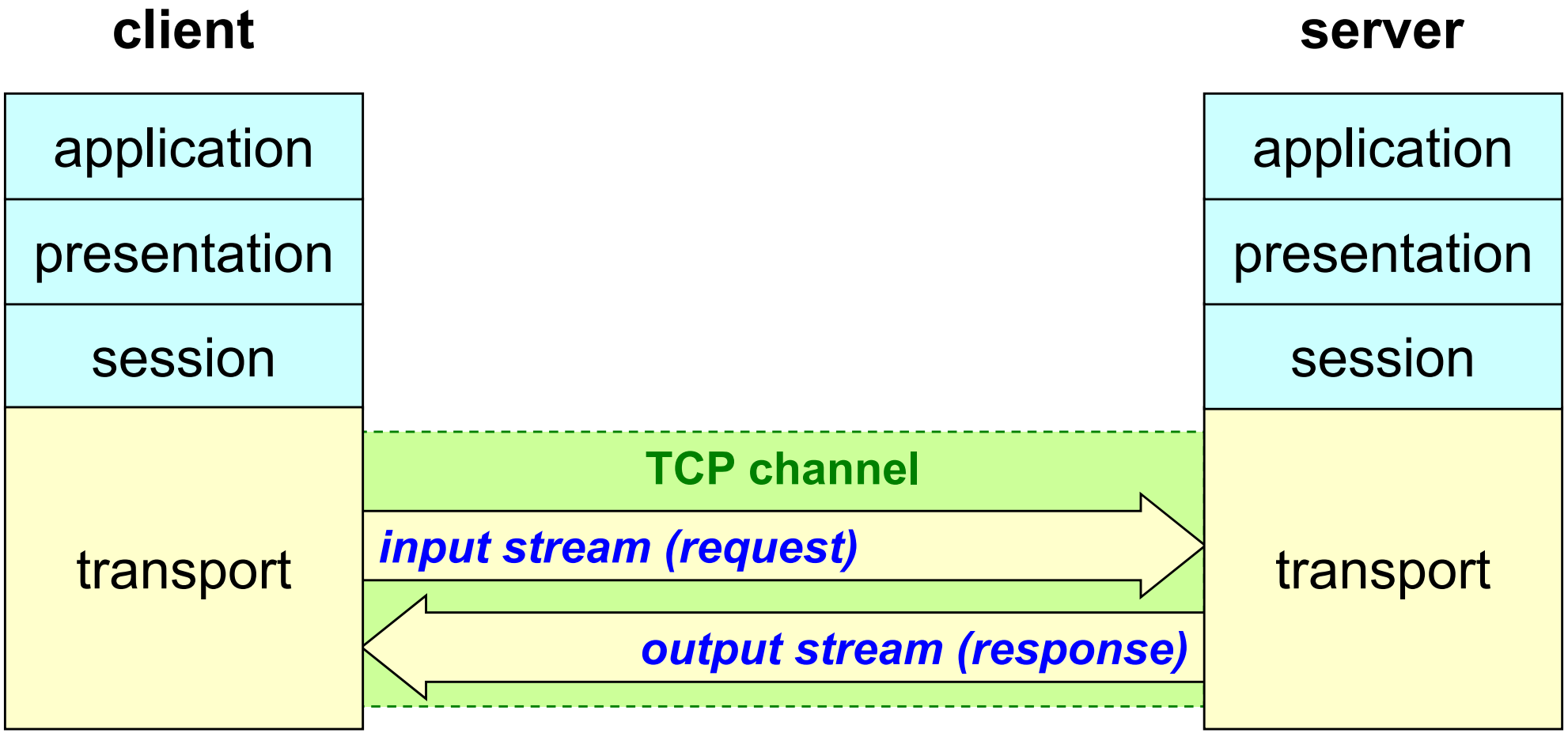
Cosa misuriamo?

- **interessa il payload quindi T e L misurati ai vari livelli dello stack differiscono (a causa dell'overhead introdotto dagli header)**
- **es. invio file 100kB su rete Ethernet a 10 Mbps**
  - in teoria L = 100 x 1024 x 8 / 10 M = 82 ms
  - … ma ci sono 14 B di header Ethernet
  - se invio payload da 16 B per volta
    - 100 x 1024 / 16 = 6,400 pacchetti
    - 6,400 x (16 + 14) = 192,000 B inviati
    - L = 192,000 * 8 / 10 M = 154 ms
  - se invio payload da 128 B per volta allora L = 91 ms (ragionevole, spreco circa il 10% della banda teorica)

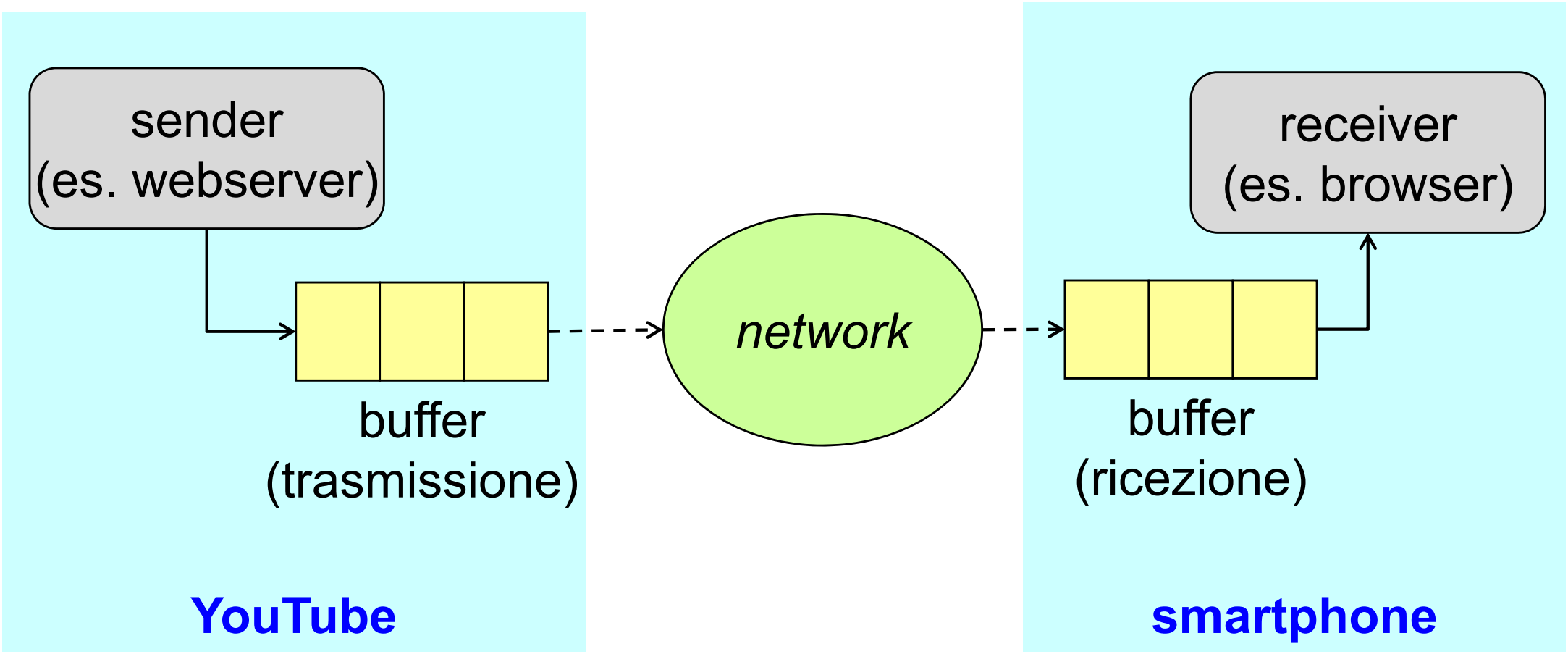
TCP (Transmission Control Protocol)

- **canale logico end-to-end**
  - API come file sequenziale forward-only (read, write)
- **stream bi-direzionale simultaneo di byte tra due componenti distribuite**
- **protocollo affidabile ma lento**
- **buffering (=memorizzazione temporanea dei dati nello stack di rete, prima di inviarli in rete o all'applicazione) sia in trasmissione sia in ricezione**
  - disaccoppia la velocità computazionale (mittente e ricevente possono avere velocità di calcolo diverse)
- **il più usato dalle applicazioni Internet**
  - web, posta elettronica, trasferimento file, ...

Uso di TCP per client-server



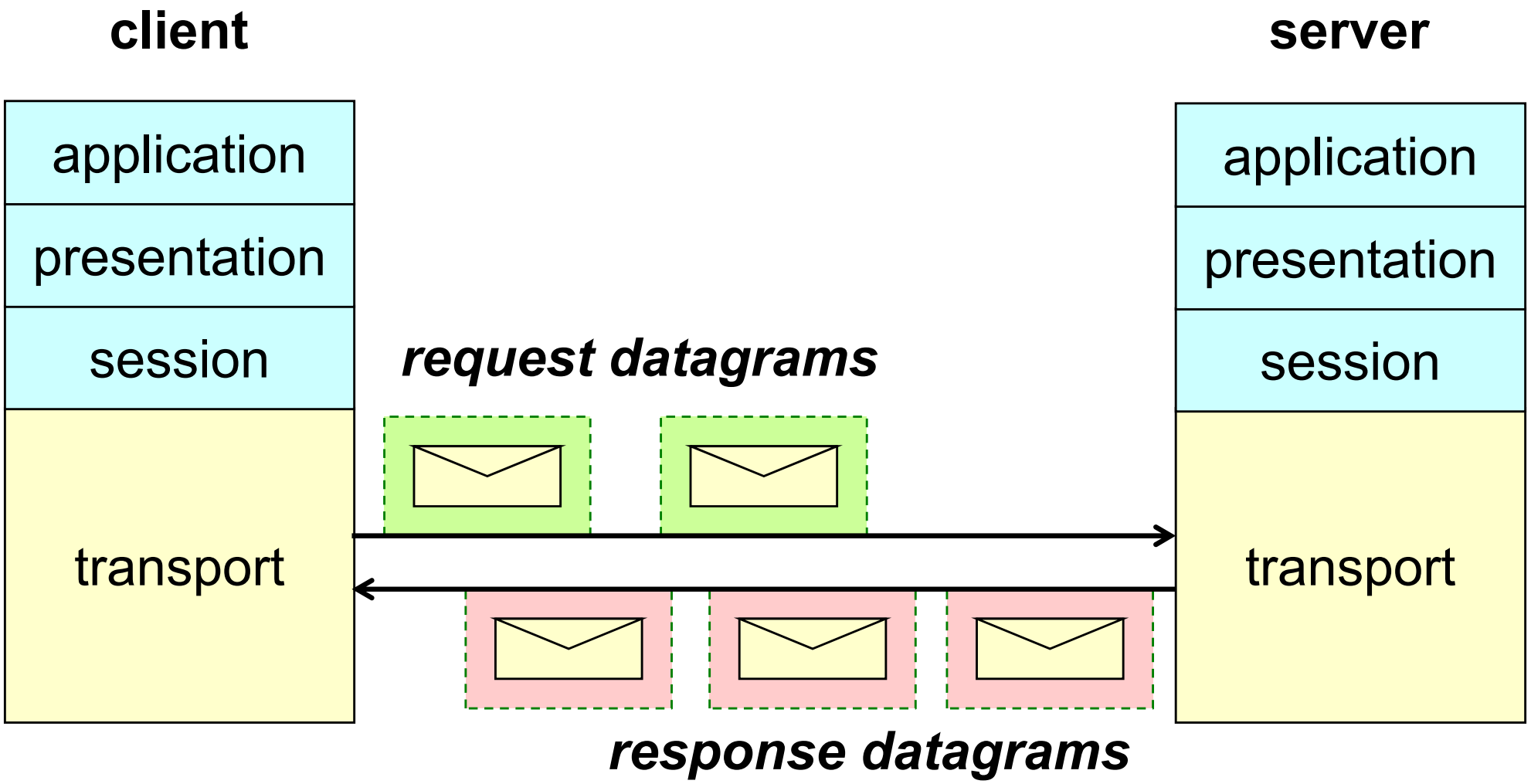
TCP buffering



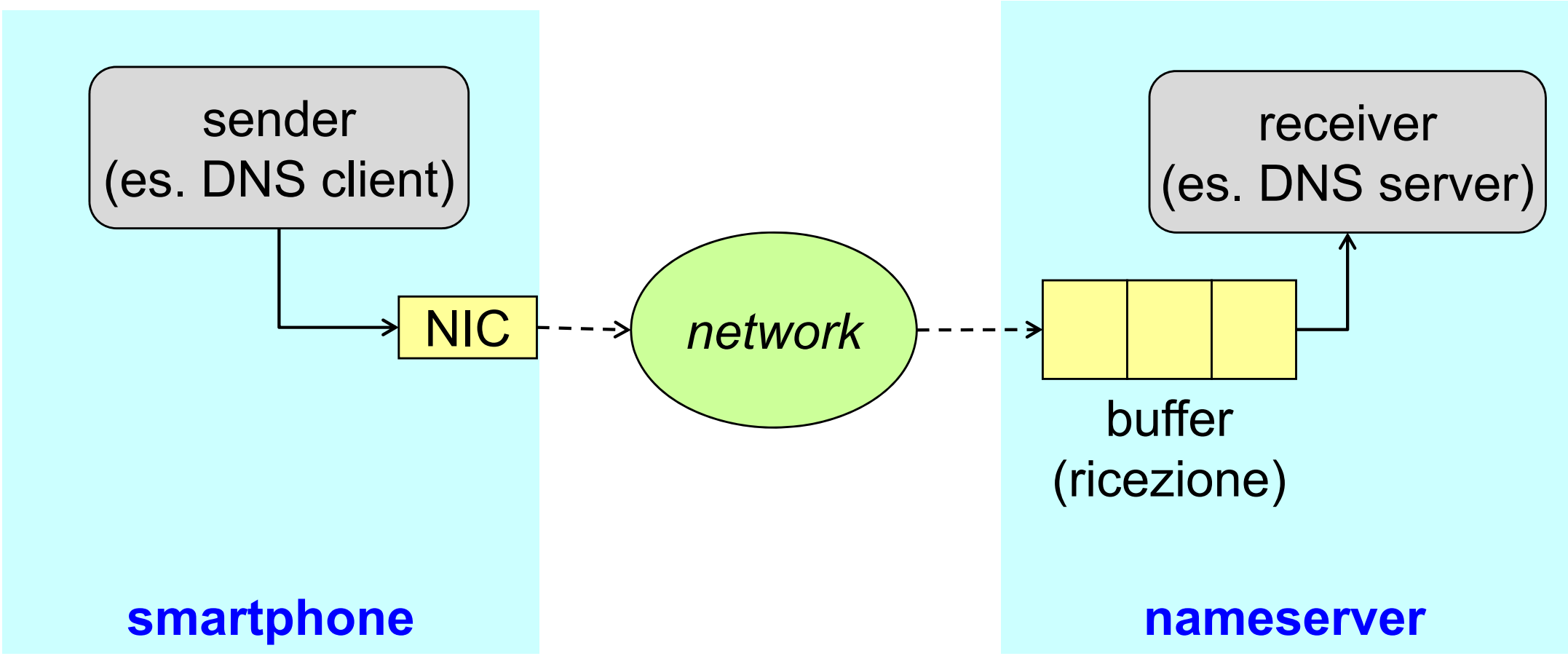
UDP (User Datagram Protocol)

- **permette alle componenti di scambiarsi messaggi (datagrammi) contenenti un insieme di byte**
  - API a messaggi (sendto, recvfrom)
- **destinatario identificato internamente al messaggio**
- **protocollo inaffidabile ma veloce**
- **limitata lunghezza del messaggio (max 64 kB)**
- **accodamento solo al destinatario (buffer di ricezione)**
  - rischio di "over-run" o "out-of-space"
- **usato per applicazioni in cui la ritrasmissione o perdita non è un problema (es. DNS, NTP)**

Uso di UDP per client-server



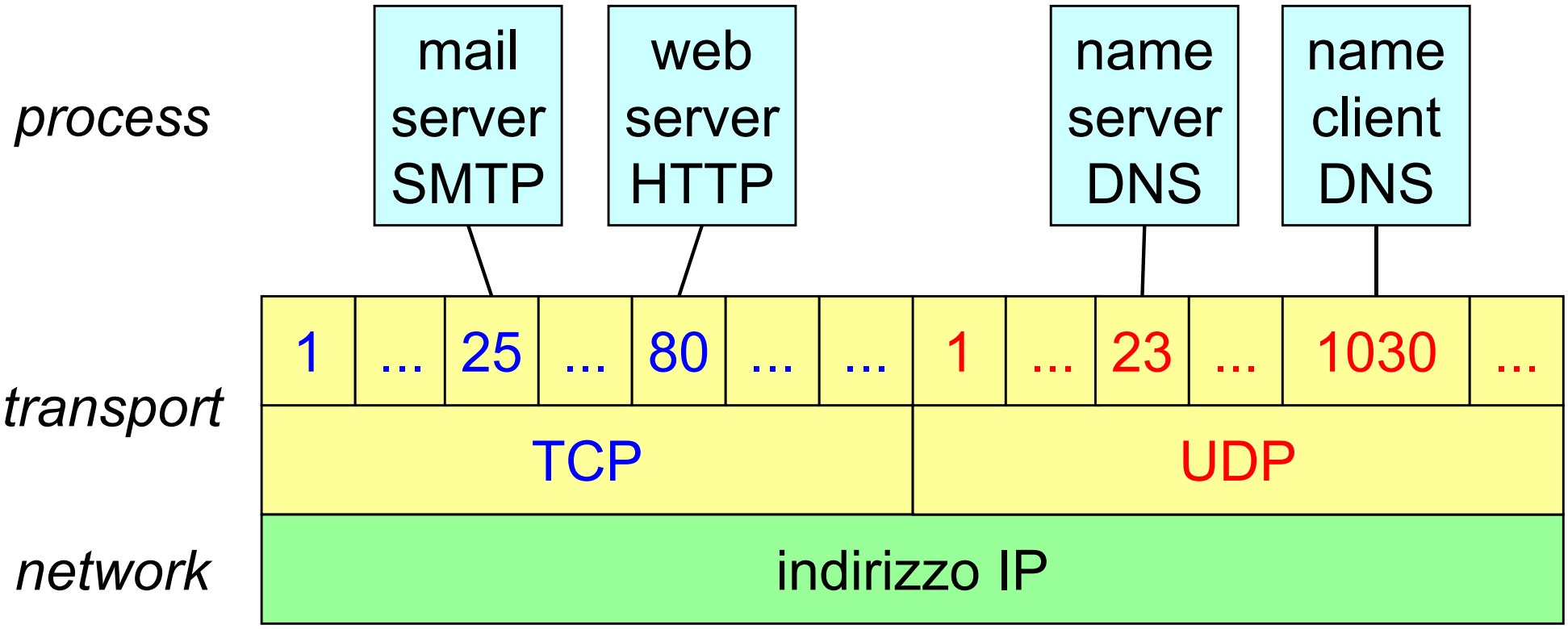
UDP buffering



TCP e UDP

- **due protocolli di trasporto alternativi**
- **realizzano funzionalità comuni a tutti gli applicativi**
- **usabili simultaneamente da applicativi diversi**
- **per distinguere i dati generati da / destinati ad una specifica applicazione su un determinato nodo si usa il concetto di porta (*multiplexing*)**
- **ad esempio un browser che voglia connettersi ad un server web deve indicare:**
  - l'indirizzo IP del server web
  - il protocollo di trasporto (TCP)
  - il numero della porta associata al servizio web (80)

Porte e multiplexing

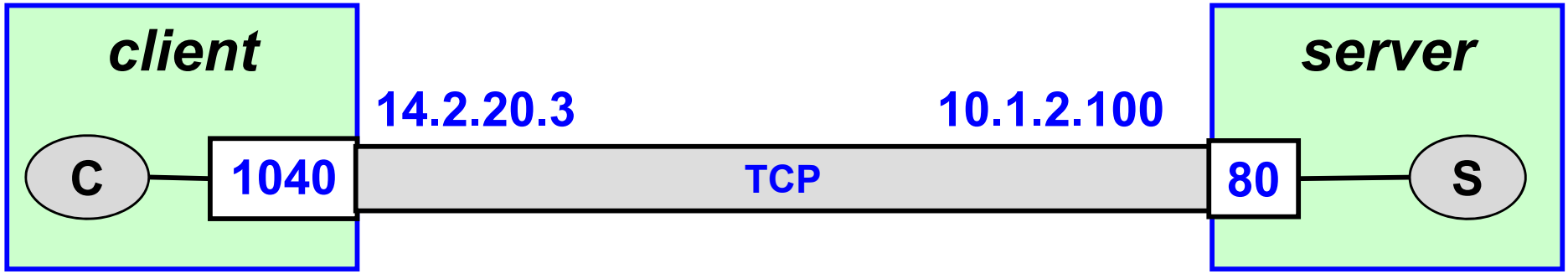


Caratteristiche delle porte TCP e UDP

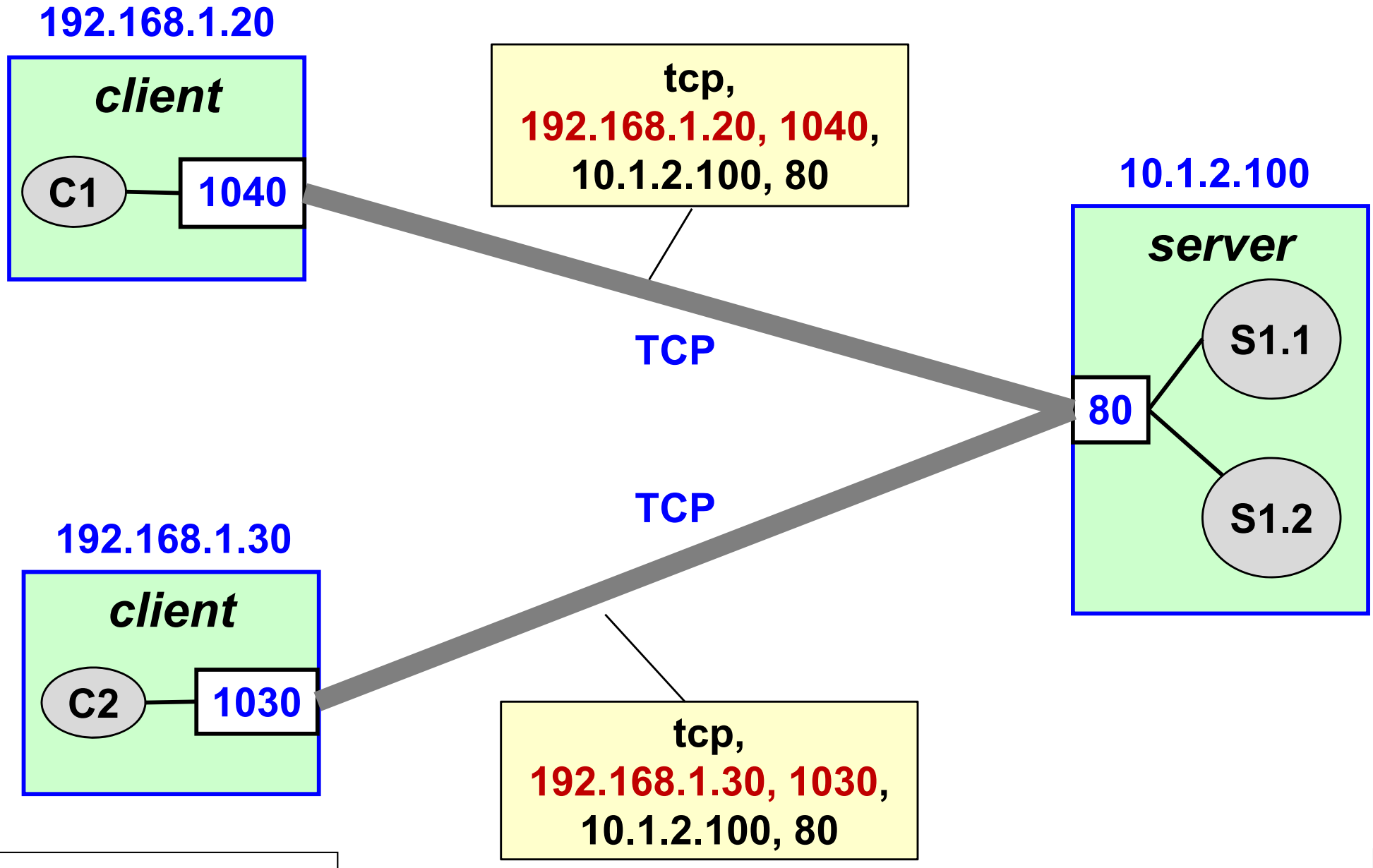
- **identificate da un numero intero su 16 bit**
- **0 ... 1023 = porte privilegiate**
  - usabili solo da processi di sistema
- **1024 ... 65535 = porte utente**
  - usabili da qualunque processo
- **porte statiche**
  - quelle dove un server è in ascolto
- **porte dinamiche**
  - quelle usate per completare una richiesta di connessione e svolgere un lavoro

Connessione TCP (o messaggio UDP)

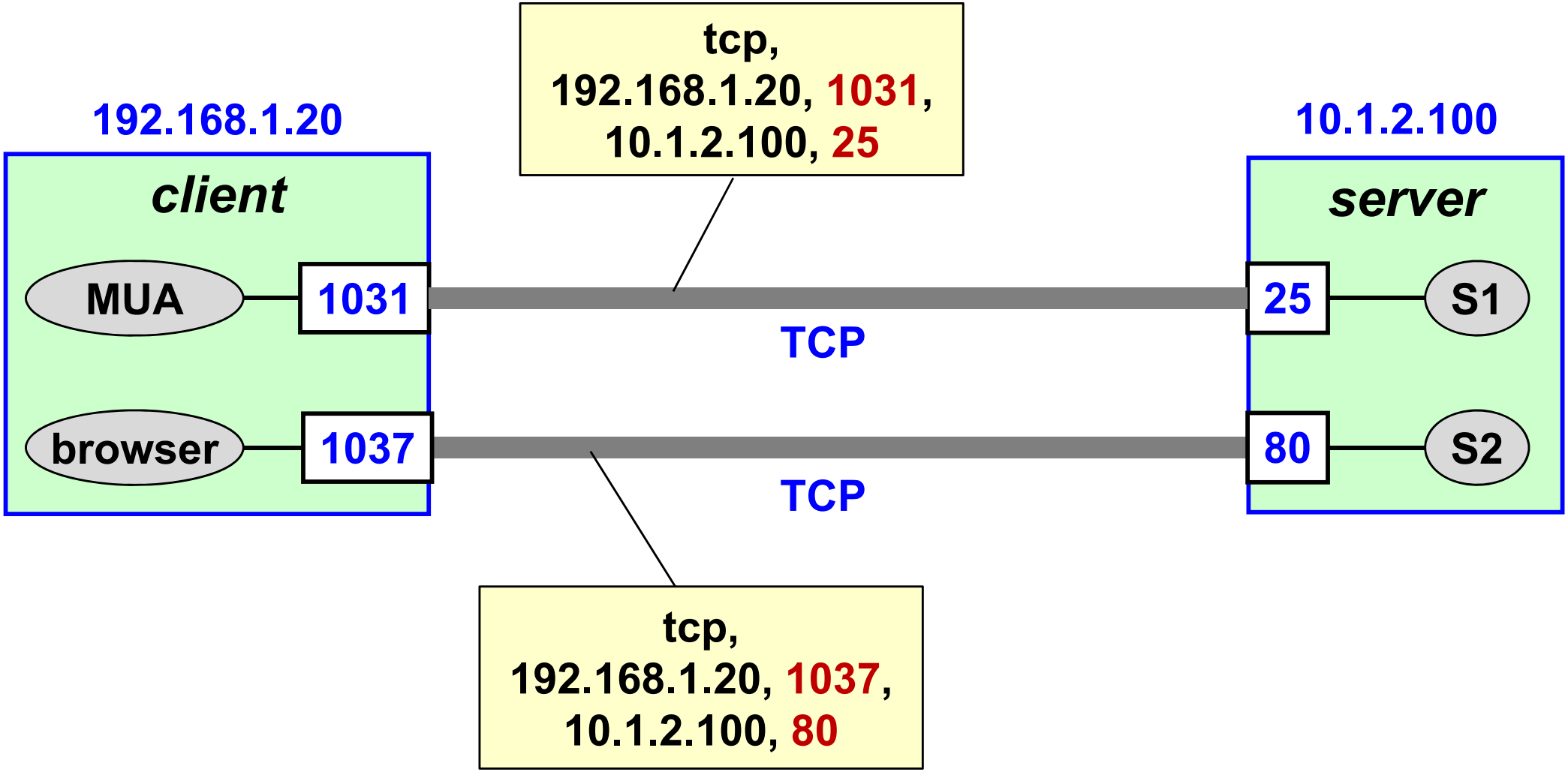
- **rappresentate da una quintupla:**
  - protocollo (TCP o UDP)
  - indirizzo IP (32 bit) e porta (16 bit) del client
  - indirizzo IP (32 bit) e porta (16 bit) del server
- **ad esempio, per un collegamento HTTP:**
  - (TCP, 14.2.20.3, 1040, 10.1.2.100, 80)



Multiplexing: N client, 1 server, 1 servizio



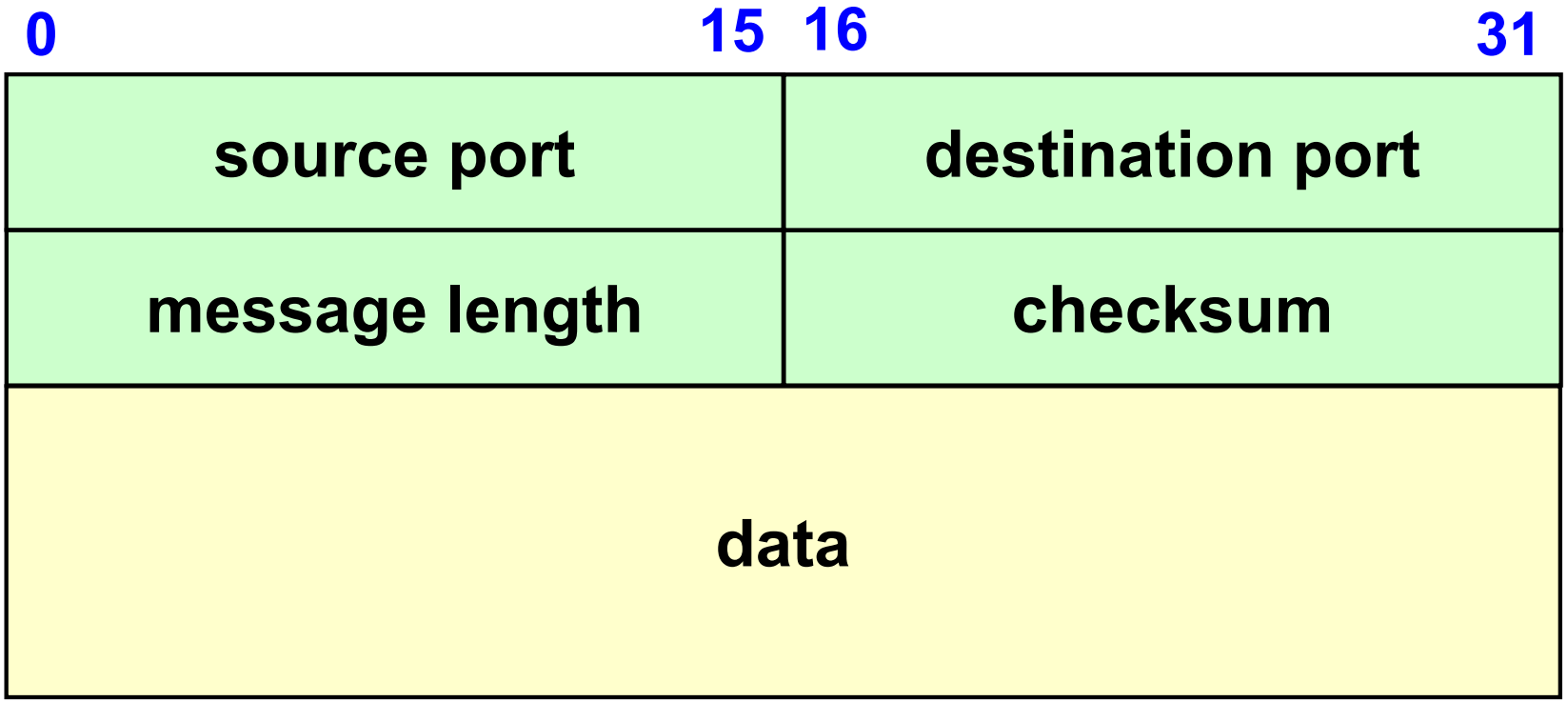
Multiplexing: N client, 1 server, 2 servizi



UDP: User Datagram Protocol

- **un protocollo di trasporto:**
  - orientato ai messaggi
  - non connesso
    - stato del destinatario ignoto
    - nessun accordo preliminare
  - non affidabile
    - datagrammi persi, duplicati, fuori sequenza
- **aggiunge due funzionalità a quelle di IP:**
  - multiplexing delle informazioni tra le varie applicazioni tramite il concetto di porta
  - checksum (opzionale) per verificare l'integrità dei dati (utile contro errori di trasmissione, non attacchi)

UDP: PDU (datagram)



Campi dell'header UDP

- **source port (16 bit)**
- **destination port (16 bit)**
- **message length (16 bit)**
  - lunghezza totale della PDU (header + payload)
  - payload massimo teorico =
    - 64 kB (massima lunghezza PDU IP)
    - ... – 8 B (header UDP)
    - ... – 20 B (minimo header IP)
    - = 65,507 B
- **checksum (16 bit, tutti zero se non usata)**
  - protegge header e payload (da errori, non attacchi!)

UDP: applicabilità

- **utile quando:**
  - si opera su rete affidabile (es. LAN o punto-punto)
  - singola PDU può contenere tutti i dati applicativi
  - non importa che tutti i dati arrivino a destinazione
  - l'applicazione gestisce meccanismi di ritrasmissione
- **maggior problema di UDP = il controllo di congestione**
  - mittente mantiene il proprio tasso di trasmissione (elevato) anche se la rete è intasata, contribuendo ad intasarla maggiormente
  - proposto DCCP (Datagram Congestion Control Protocol)

UDP: applicazioni

Le principali applicazioni che usano UDP sono:

- **DNS (Domain Name System)**
  - traduzioni nomi – indirizzi IP
- **NFS (Network File System)**
  - dischi di rete (in ambiente Unix)
- **SNMP (Simple Network Management Protocol)**
  - gestione apparecchiature di rete (router, switch, …)
- **molte applicazioni di streaming audio e video**
  - è importante la bassa latenza
  - è accettabile la perdita di alcuni dati (ridondanza delle codifiche audio e video)

TCP: Transmission Control Protocol

- **un protocollo di trasporto:**
  - byte-stream-oriented
  - connesso
  - affidabile
- **usato da applicativi che richiedono la trasmissione affidabile dell'informazione:**
  - telnet = terminale virtuale
  - FTP (File Transfer Protocol) = trasferimento file
  - SMTP (Simple Mail Transfer Protocol) = trasmissione e-mail
  - HTTP (Hyper-Text Transfer Protocol) = scambio dati tra browser e server web

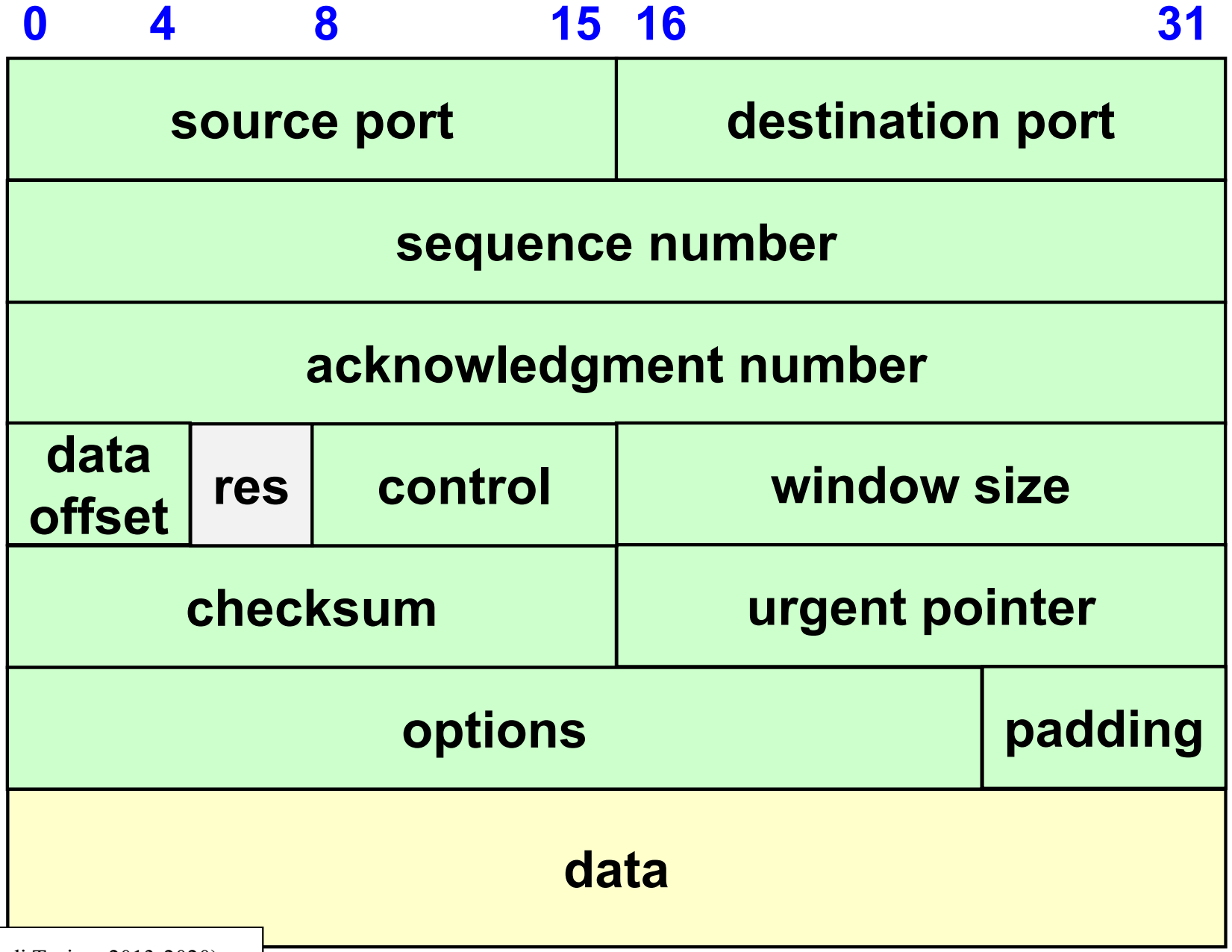
TCP: funzionalità

- **funzionalità TCP:**
  - supporto della connessione tramite circuiti virtuali
  - controllo di errore
  - controllo di flusso
  - multiplazione e demultiplazione
  - controllo di stato e di sincronizzazione
- **TCP garantisce la consegna dei dati, UDP no!**

TCP: caratteristiche

- **come UDP ha il concetto di porta**
- **il TCP di un nodo, quando deve comunicare con il TCP di un altro nodo, crea un *circuito virtuale***
- **al circuito virtuale è associato un protocollo**
  - full-duplex
  - acknowledgement
  - controllo di flusso
- **segmenta e riassembla i dati secondo necessità:**
  - nessuna relazione tra il numero di read e di write
- **usa sliding window, timeout e ritrasmissione**
- **TCP richiede più banda e più CPU di UDP**

TCP: PDU (segment)



Campi dell'header TCP (I)

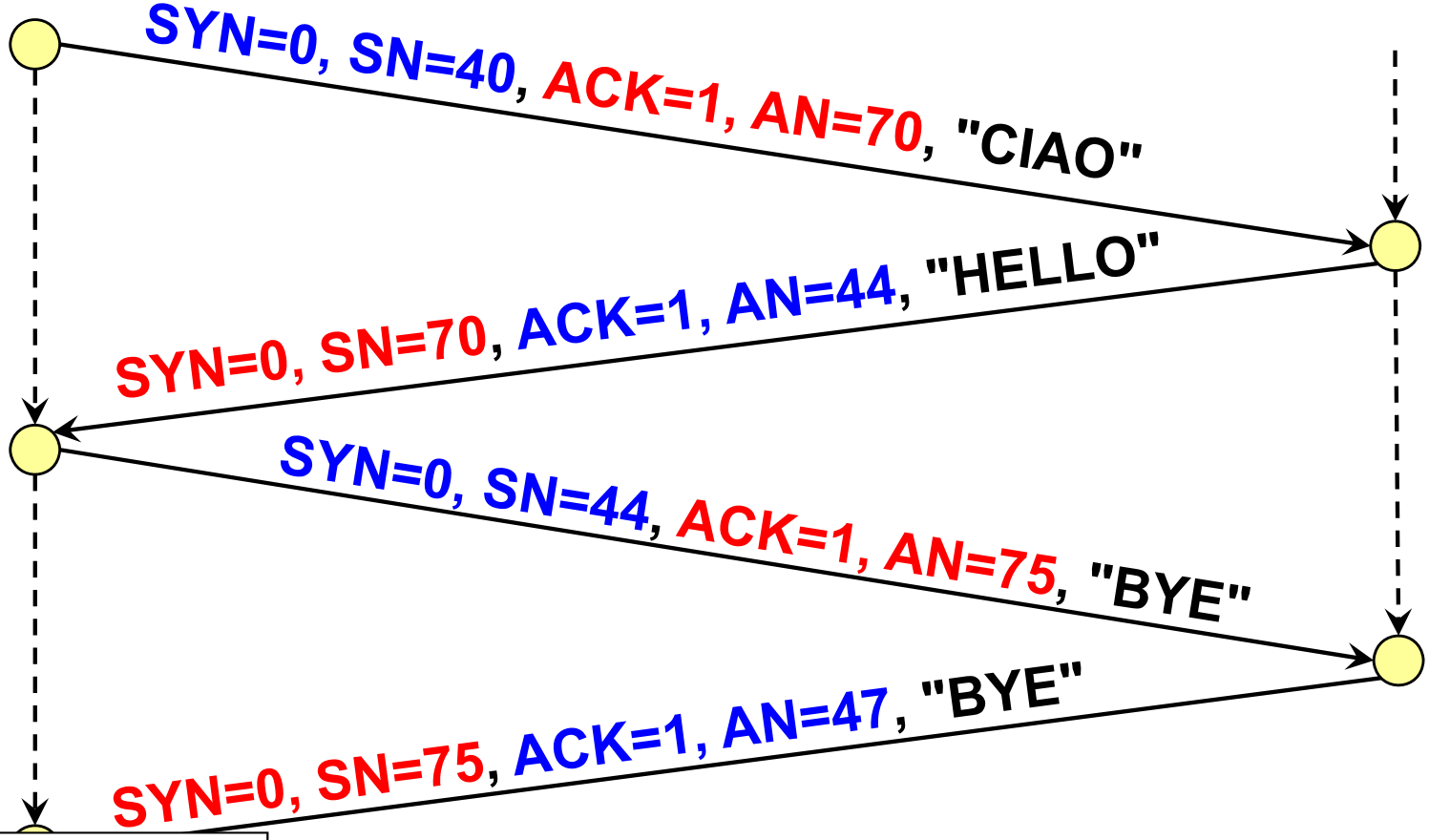
- **source port (16 bit)**
- **destination port (16 bit)**
- **sequence number (32 bit)**
  - (SYN=1) valore iniziale
  - (SYN=0) posizione nello stream del primo data byte
- **acknowledgment number (32 bit)**
  - (ACK=1) posizione nello stream del primo data byte da ricevere (tutti quelli precedenti sono OK)
  - (ACK=0) non significativo
- **checksum (16 bit, tutti zero se non usata)**
  - protegge sia header sia payload

Campi dell'header TCP (II)

- **data offset (4 bit)**
  - lunghezza dell'header TCP in word da 32 bit
  - valore 5...15 (20...60 B, con max 40 B di opzioni)
- **control (9 bit) = insieme di flag**
  - SYN = sincronizzare i Sequence Number
  - ACK = campo Acknowledgment Number valido
  - FIN = non verranno trasmessi altri dati
  - RST = reset della connessione
  - PSH = inviare i dati nel buffer all'applicazione
  - URG = campo Urgent Pointer valido
  - NS, CWR, ECE (controllo avanzato di congestione)

TCP sequence number

- SN (Seq.Num.) = posizione primo byte inviato
- AN (Ack.Num.) = posizione primo byte libero nel buffer di ricezione



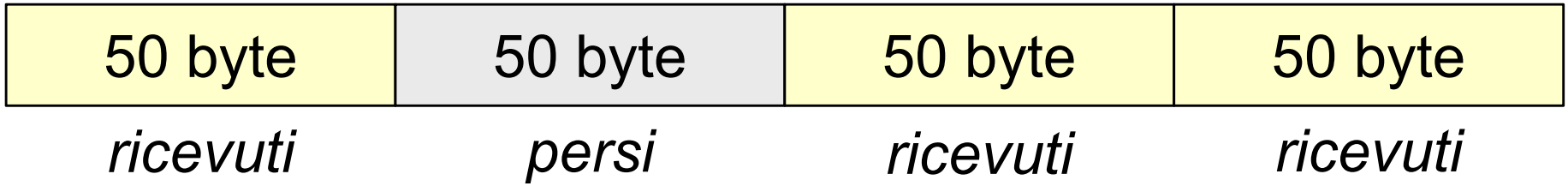
Campi dell'header TCP (III)

- **window size (16 bit)**
  - spazio ancora disponibile nella receive window
  - il mittente può mandare al massimo questi byte prima di attendere un ACK ed una nuova WIN
    ... oppure lo scadere del timeout!
- **options (0-320 bit, in multipli di byte)**
  - opzioni varie
  - es. Timestamp, Selective Acknowledgment
- **padding**
  - byte a zero per rendere l'header multiplo di 32 bit

TCP Selective Acknowledgment

- **anche detto SACK**
- **definito in RFC-2018**
- **utile quando viene perso un segmento ma ricevuti correttamente quelli successivi**
  - l'opzione indica un intervallo di byte ricevuti
  - aggiuntiva rispetto ad ACK

Esempio SACK

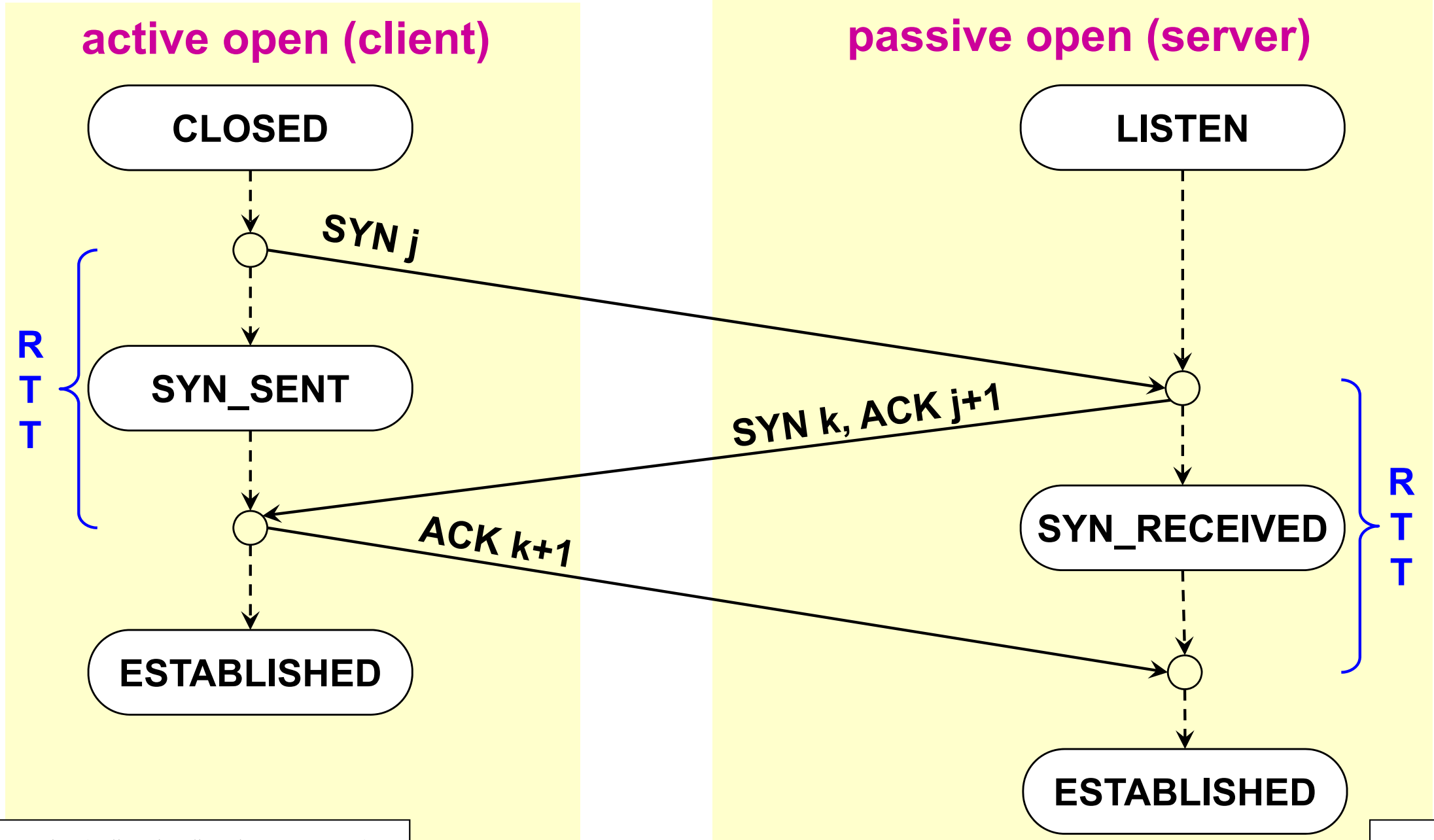


- **(normale) ACK=50**
  - sender ritrasmette 51-200 (ossia 150 byte)
- **(con SACK) ACK=50, SACK=101-200**
  - sender ritrasmette 51-100 (ossia 50 byte)

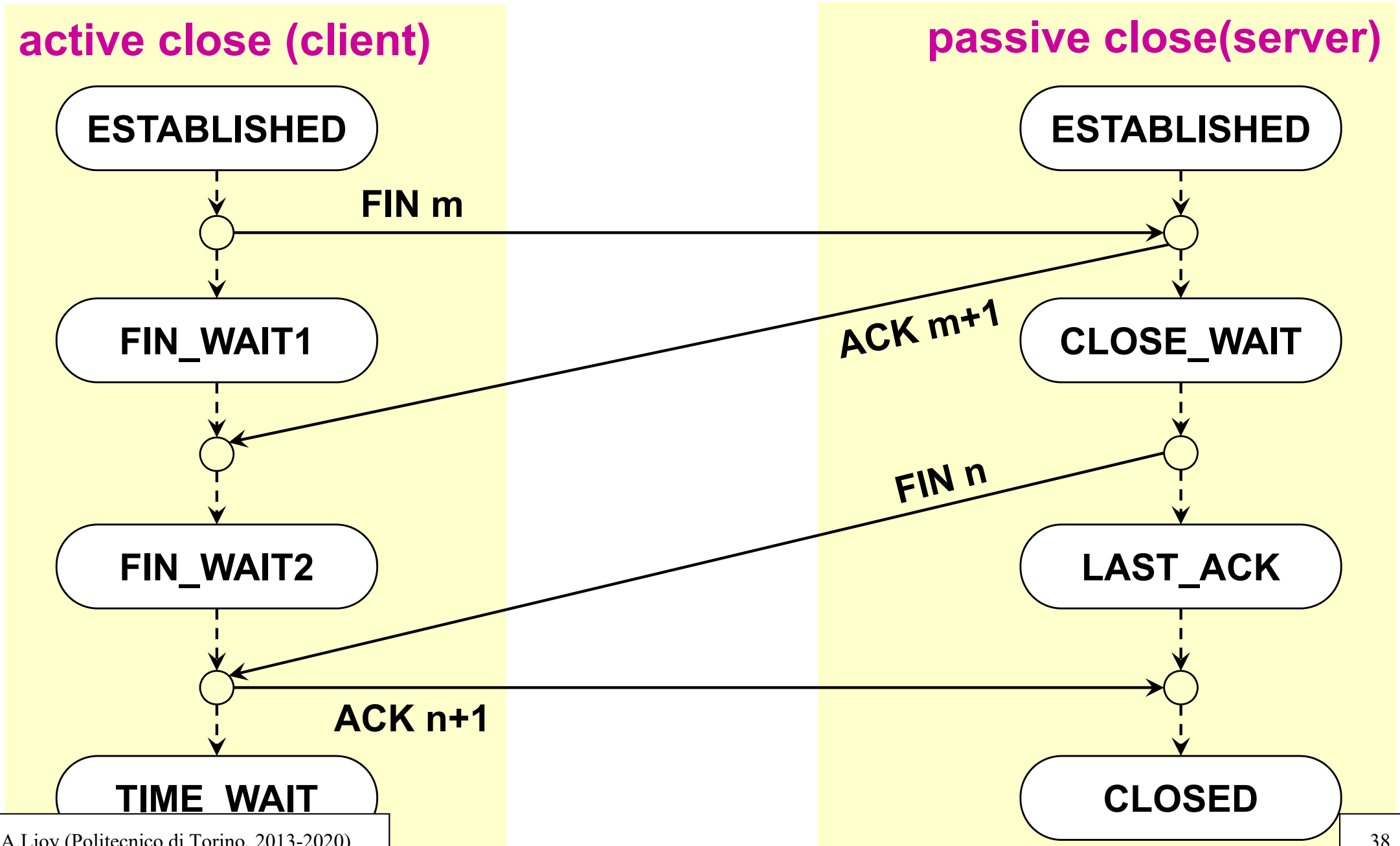
TCP: Urgent Pointer

- **con URG=1, indica che nel segmento ci sono uno o più byte "urgenti"**
- **tipicamente associati ad eventi asincroni:**
  - interrupt
- **byte da trattare prima di quelli già ricevuti ma ancora nel buffer**
- **è un meccanismo per "saltare la coda" sul ricevente …**
- **… ma non ha effetto sulle code in rete**
- **… quindi è praticamente inutile!**

TCP: three-way handshake



TCP: four-way teardown



Lo stato TIME_WAIT

- **dallo stato TIME_WAIT si esce solo per timeout:**
  - durata pari a 2 x MSL (Max Segment Lifetime)
  - MSL = 2 minuti (RFC-1122), 30 secondi (BSD)
  - quindi timeout 1…4 minuti
- **esiste per risolvere due problemi:**
  - implementare la chiusura TCP full-duplex
    - l'ultimo ACK potrebbe venir perso ed il client ricevere un nuovo FIN
  - permettere a pacchetti duplicati di "spirare"
    - potrebbero essere interpretati come parte di una nuova incarnazione della stessa connessione

Come si creano i segmenti TCP?

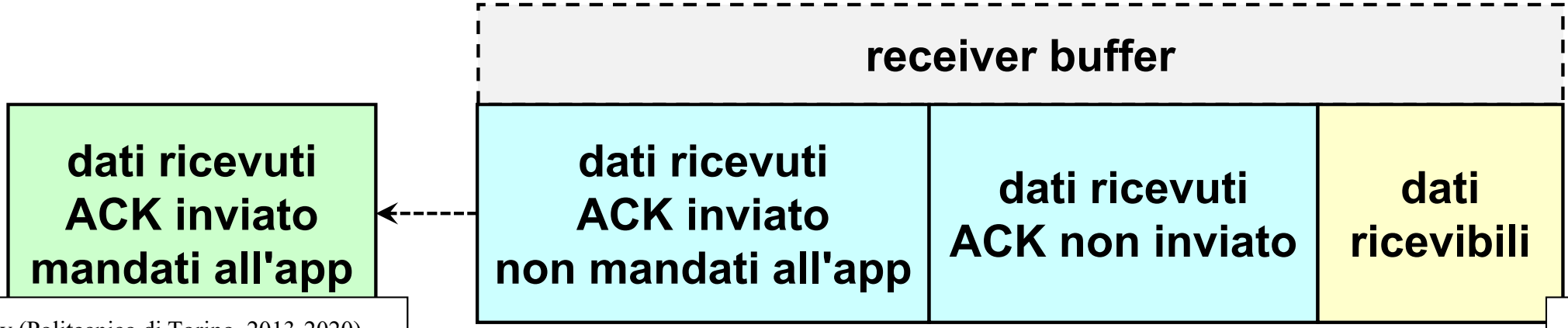
- **TCP riceve un byte-stream dal livello superiore**
- **quando invia un segmento? con quanti byte?**
  - invia quando è pieno un segmento massimo (MSS)
  - invia alla scadenza di un timeout (200 ms)
  - invia quando si riceve un comando esplicito (flush) dal livello superiore
  - invia non appena ci sono dati disponibili (se la connessione è marcata TCP_NODELAY)
- **segmenti grossi: maggiore latenza, migliore throughput (viceversa per segmenti piccoli)**
  - es. download vs. on-line gaming

TCP Maximum Segment Size (MSS)

- **massima quantità di dati (payload) presenti in un segmento TCP:**
  - 576 B (max pacchetto IP trattabile da tutti)
  - ... – 20 B (min header IP)
  - ... – 20 B (min header TCP)
  - = 536 B

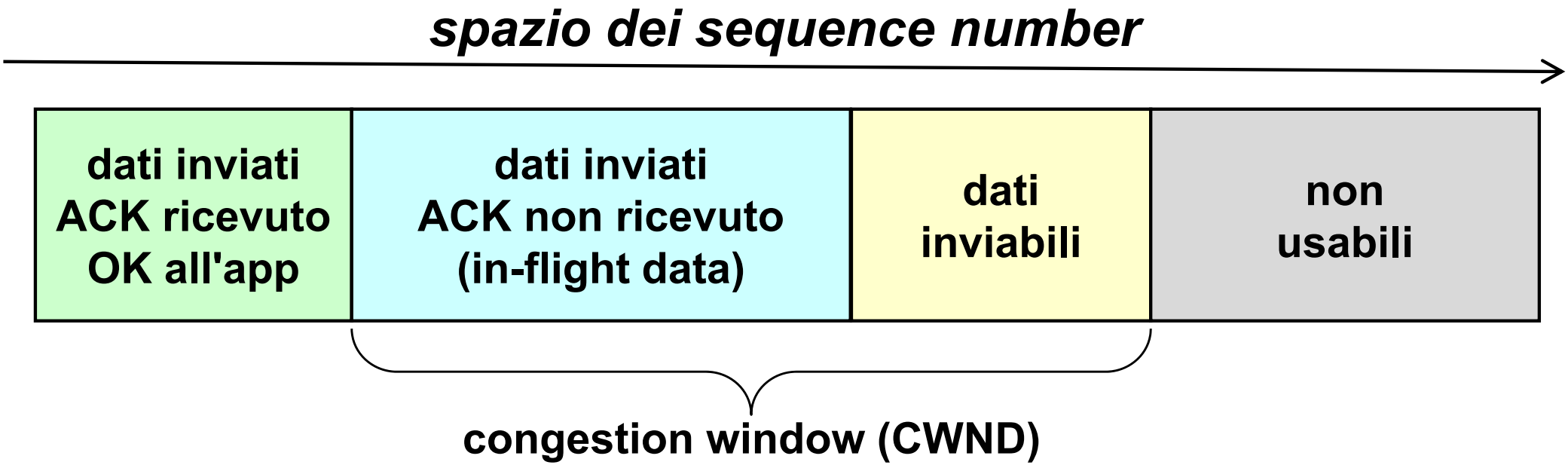
TCP: sliding window

- **il ricevente ha un buffer limitato**
- **lo spazio disponibile varia in base a:**
  - ACK inviati
  - dati prelevati dall'applicazione
- **non manda ACK per singoli byte ma cerca di**
  - raggruppare più byte ricevuti
  - usare "piggyback" (quando ci sono dati da inviare)



TCP: sliding window

- **il mittente deve autolimitarsi per non intasare il ricevente**
- **invia al massimo CWND byte e poi aspetta gli ACK per RTT**
  - quindi (senza ritrasmissione) T ~ CWND / RTT



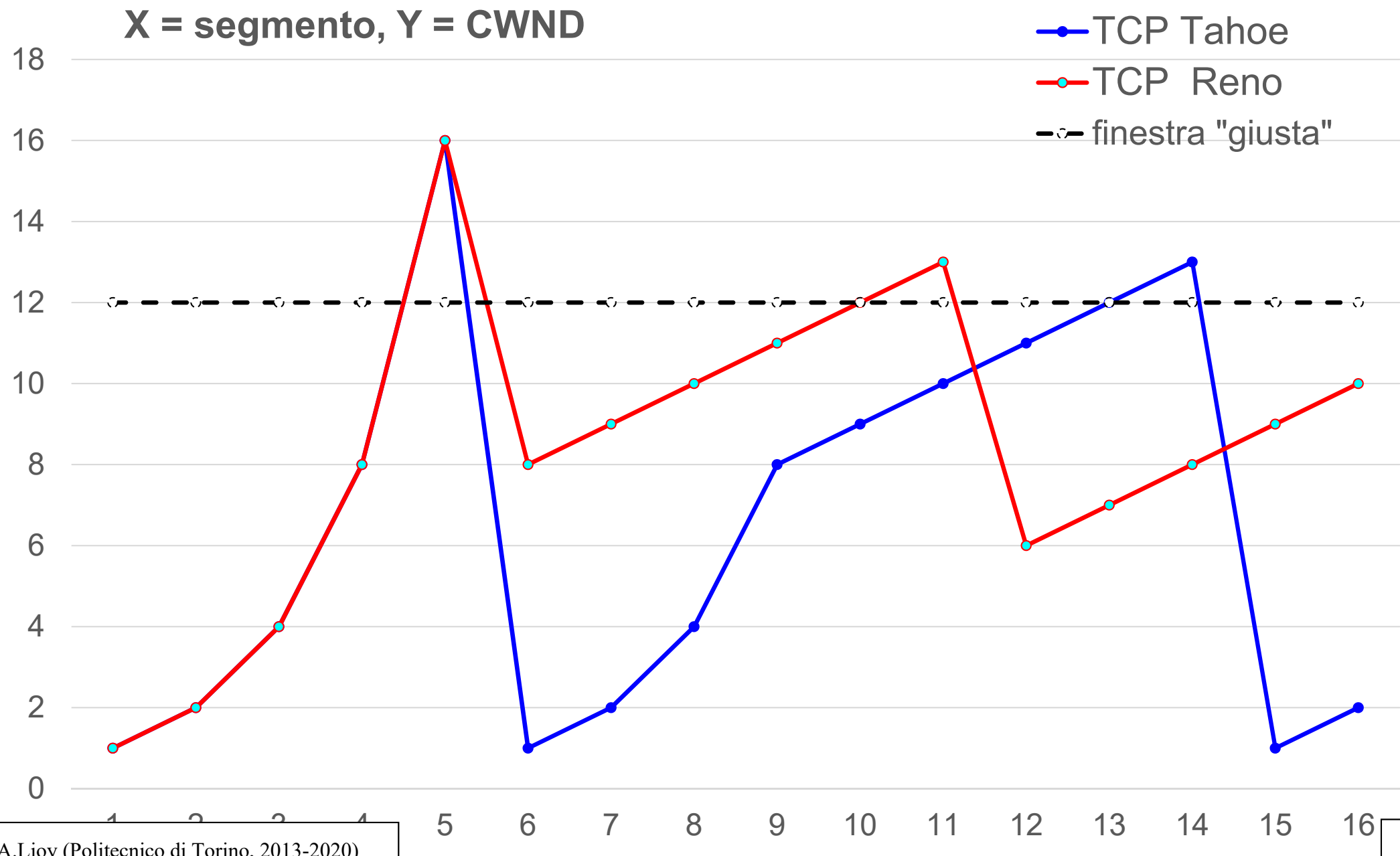
TCP: slow start

- **le prime versioni di TCP quando andavano in timeout ritrasmettevano l'intera window**
- **questo poteva causare gravi congestioni della rete:**
  - nell'ottobre 1986 Arpanet fu bloccata da una congestione (da 32 kbps a 40 bps)
- **inizio: CWND = 1 MSS**
- **CWND raddoppia ogni RTT**
  - realizzato facendo CWND += 1 MSS per ogni ACK ricevuto
- **TCP parte piano ma accelera velocemente**

TCP: ritrasmissione

- **problema rilevante per le connessioni wireless, più soggette a timeout di quelle cablate**
- **quando TCP non riceve ACK entro il timeout**
  - TCP Tahoe
    - CWND = 1
    - poi cresce esponenzialmente (x2) sino al limite
    - poi cresce linearmente (+1) fino a nuovo timeout
  - TCP Reno
    - CWND = ½ ultima CWND valida
    - poi cresce linearmente (+1) fino a nuovo timeout

TCP slow start + ritrasmissione



Equità di trattamento in TCP (fairness)

- **col meccanismo di ritrasmissione adottato, se N flussi condividono la medesima banda B, ciascun flusso ne userà una frazione B/N**
- **esempio su un link con banda totale B:**
  - 9 processi con un flusso a testa, ognuno usa B/9
  - nuovo processo apre 11 flussi
  - questo processo ottiene non 0.1*B ma 0.55*B (perché usa 11 flussi su 20 totali)
  - trucco fatto da molti browser per avere più banda
- **applicazioni multimediali non usano TCP perché non vogliono che il throughput vari, ma usano UDP inviando dati a velocità costante ed accettando delle perdite (ridondanza della codifica)**

Miglioramenti di TCP

- **perdita di pacchetti = congestione?**
  - vero in passato, non più così vero oggi
- **prestazioni possono essere limitate da:**
  - applicazione (non manda abbastanza dati da sfruttare tutta la banda disponibile)
  - collo di bottiglia (link a minor velocità di trasmissione)
  - buffer (troppo piccolo per tenere tutti i dati ricevuti)
- **oggi esistono ben 12 algoritmi per controllo di congestione**
- **… e poi arriva G con BBR (Bottleneck Bandwidth and Round-trip propagation time)**
  - migliora prestazioni 5-10% … ma è unfair verso flussi non BBR