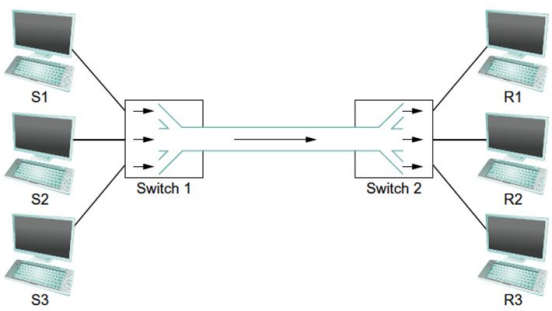
Following from last lecture… How to build a computer from the groud up?

Challenge: **Resource Sharing & Efficiency**

More specifically…
How do *all the host*s that want to communicate share the network, especially if they want to use it **at the same time**?
How do *several hosts* that **share the same link** when they all want to *use it at the same time*?

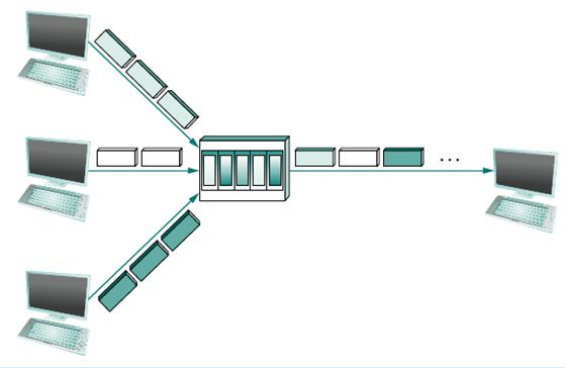
Solution: **multiplexing**!
(Other real world examples of resource sharing and multiplexing: Telephone networks: TDMA, CDMA; Operating systems: CPU scheduling)



Method 1: Synchronous Time Division Multiplexing (STDM)
Method 2: Frequency Division Multiplexing (FDM)

Potential problems: What happens if one of the flow doesnt have any data? Maximum number of flow is fixed and assumed to be known?

Another Solution: **Statistical Multiplexing** (Send on Demand!)



Potential problem: Problem of letting each host send on demand? The hosts other than the sending host may not get a chance to send
Q: So, how do we ensure every host has a chance to send?
A: The concept of "packets" -> limit the amount of data each host can send!

Q: How to implement the "CAP": which packet to send next at a switch?
A: FIFO, Round-Robin, etc.

Q: What happens if the switch receives faster than it sends?
A: Store and Send; Congestion Control

Q: Computer network is more than packet delibery b/w computers; applications on hosts need to communicate too!
A: Instead of building all functionality into each application, we develop *common services* and build applications using these services

Some thinking… what functionality should a channel provide to applications?
E.g. guarantee for message delivery? msg receive in same order as they're sent? requirement on msg deliver within a strict time bound? ensure no 3rd party can eavesdrop on the channel?

Two application examples:
File transfer protocol (FTP) - request/reply channel
Video on demand - Streaming channel

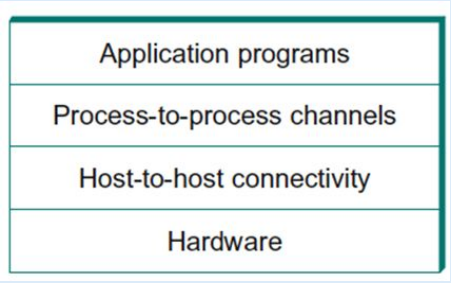
Requirements	Request/Reply (FTP)	Streaming (Video)
Allow message deliver failure?	No	Yes
Allow message received out of order?	Yes	No
Strict time bound on message delivery?	No	Yes
Need protect privacy and integrity?	Yes	Yes

Q: Where is the function of channel implemented, network or end host?
A: Option 1: simple network + complex end hosts
Option 2: complex network + simple end hosts
Option 3: a tradeoff bw the 2 options

About **reliability**:
Possible reasons a computer network could fail: machine can crash and reboot; fibers and be cut out; switches run out of bufer space; bits are corrupted during transmission; bugs in software…
3 kinds of error: bit error (1->0 or 0->1) very rare; packet error (distinguish bw late and lost packets); node/link error (distinguish bw failed and slow nodes)
requirement of network: recover from these failures automatically so that applications dont have to worry about them!

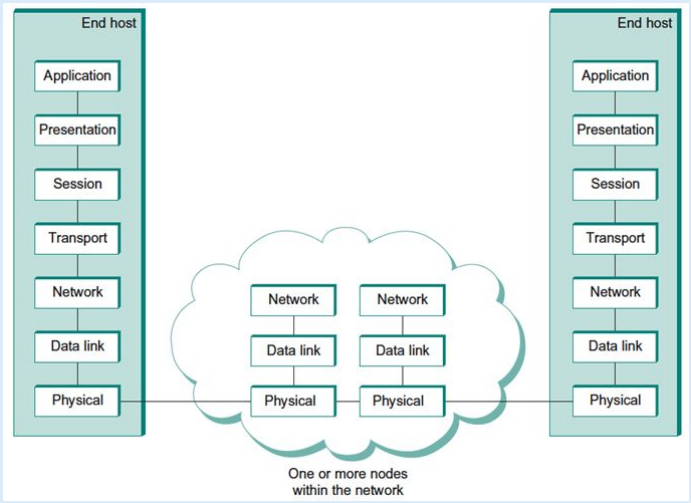
Network Architecture

- Layering
  - abstraction naturally leads to laying
  - starts with services offered by hardware and add a sequence of layers
  - services offered at high layers are implemented in terms of services provided at lower layers

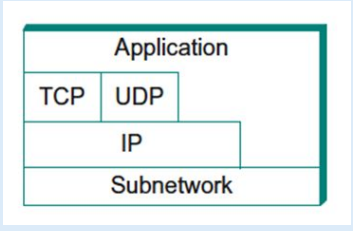


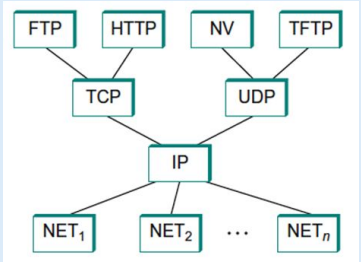
- Features of laying
  - decompose complex problem of building a network into more manageable components
  - provides a modular design (reuse functions at other layers)

- 7-Layer OSI (Open System Interconnection) Model



- Internet Architecture (TCP/IP Architecture)





As we can this, this architecture is <u>wide at top and bottom, and narrow in the middle</u>

Summary:

The goal is build a *general purpose computer network* that can support many different applications

What do we expect from networks? cost-effective and scalable connectivity; reliable channel, easy management

How to figure against the complexity of building the networks? **OSI 7-layer model** vs **Internet architecture** (TCP/IP)

How to implement the network protocol? Network API (socket programming)

How to evaluate the network performance? Latency, bandwidth, Jitter, etc.