

Multiple Processor Systems

To do ...

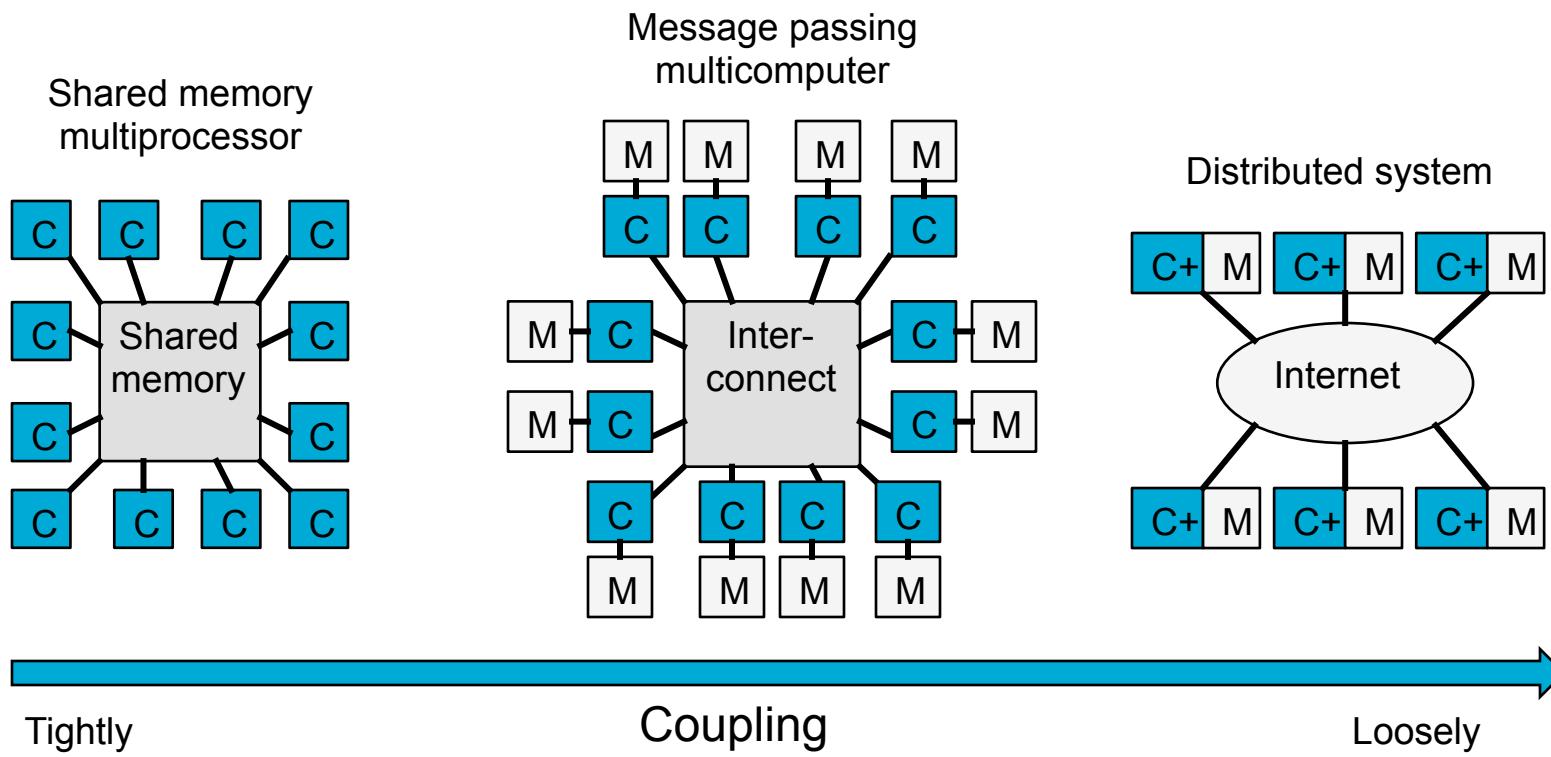
- Basic hardware and OS issues in
- Multi-processors
- Multi-computers
- Distributed systems
- Next: ?

Multiple processor systems

- We always need more computation power
- Making clocks run faster won't do it for long
 - Special theory of relativity limits signals' speed to ~20cm/nsec on wire
 - 1GHz computer – signals cannot travel more than 20cm within one cycle (10^9 cycles/second, 10^9 nanoseconds/sec)
 - 10GHz – 2cm, 100GHz – 2mm, ... to let the signal do a roundtrip within one cycle
 - Even if we can make them that small, temperature/heat dissipation is another issue
- Alternative – Pile machines together, as
 - Massively parallel machines
 - Over the Internet

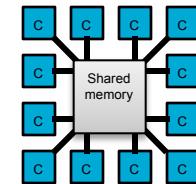
Multiple processor systems

- Challenge – Let them talk to each other to tackle a common problem
 - All communication between as message exchange
 - Differences in the time, distance and logical organization



Shared-memory systems

- Shared-memory – Nodes have equal access to memory, typically managed as paged virtual address space
- Uniform & Nonuniform Memory Access (N/UMA)
 - In UMA all memory words can be read as fast as every other
- Interconnecting nodes
 - UMA – Bus-based (a few 10s), cross-bar or multistage switching (still need quite a few switches)
 - NUMA – Add local memory, forget uniform access time!
 - With/without caching to hide differences; which must be somehow kept consistent
 - Multicore chips – aka chip-level multiprocessors (CMPs)



Multiprocessor OS types

- Each CPU with its own OS
 - Shared OS code, separate OS data structures
 - Simplest, but poor performance
 - E.g. Disk blocks? Each OS has its own cache – inconsistency; No caches – good for consistency, bad for performance
- Master-slave
 - A master holds the one copy of OS and tables, and distributes tasks among slaves
 - No synchronization issues, but master becomes a bottleneck
- Symmetric multiprocessing (SMP)
 - One copy of the OS, but any CPU can run it
 - On a sys call, the CPU where it was made traps to the kernel and processes it
 - Problem – need to synchronize access to shared OS state

Multiprocessor OS issues – Synchronization

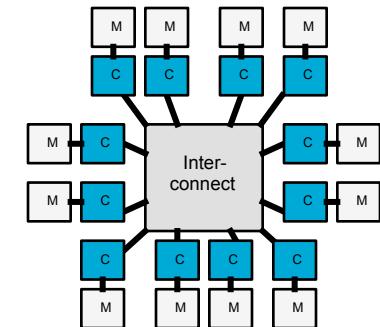
- CPUs in MPs frequently need to synchronize
 - E.g. when accessing kernel critical regions
- Disabling interrupts won't do
- TSL – Need to lock the bus to implement it
 - But spin-lock blocks everybody trying to use the bus
 - And cache invalidation generates a lot of traffic between holder and requesters of the lock (caches work with 32-64B blocks)
 - Test (read) before TSL? Read only poll is cheaper, only invalidated when lock owners frees the lock/writes
- Spinning or switching
 - Testing repeatedly is not productive and CPU switching wastes cycles – can you learn from past behavior (how long did you spin last)?

Multiprocessor OS issues – Scheduling

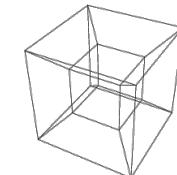
- Now bi-dimensional – which thread *and* where?
 - Timesharing – Simplest, a single ready queue
 - Contention for shared data structure, affinity to preserve state
 - Space sharing
 - Split CPUs dynamically to run related threads at the same time in different slices
- Gang scheduling – Time and space
 - Group of related threads in gangs, scheduled as a unit
 - Gang members start and end their time slices together
 - All members of a gang run simultaneously, on different timeshared CPUs
 - So, communicating members can request/reply almost immediately

Message passing multicomputer

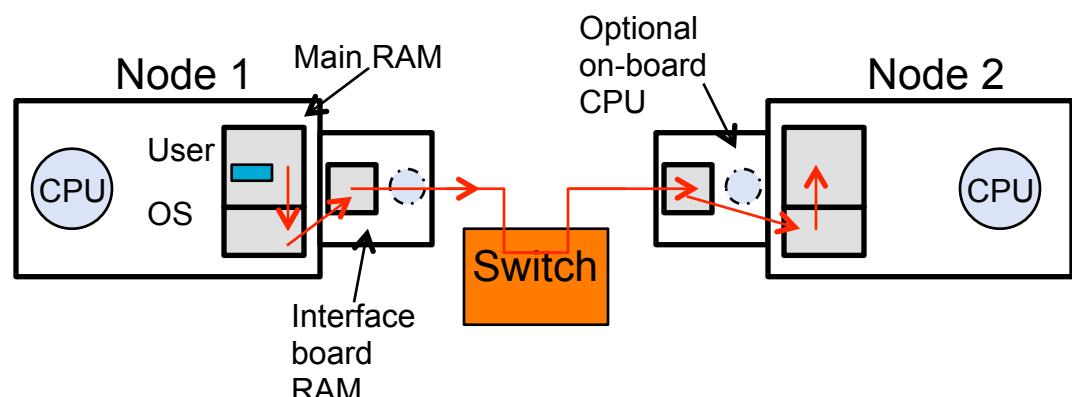
- CPU-Mem pairs connected by a high-speed network (clusters, COWS)
 - Easier to build/cheaper than MP
 - A stripped down PC (no keyboard, mouse, or monitor)
 - Cloud computing services are build on them



- Between CPUs, a high-speed interconnect
 - Organizations – Single switch, ring, mesh, hypercube ...
 - Switching – Store-and-forward or circuit switching
 - Network interfaces – RAM, DMA, network processors, ...

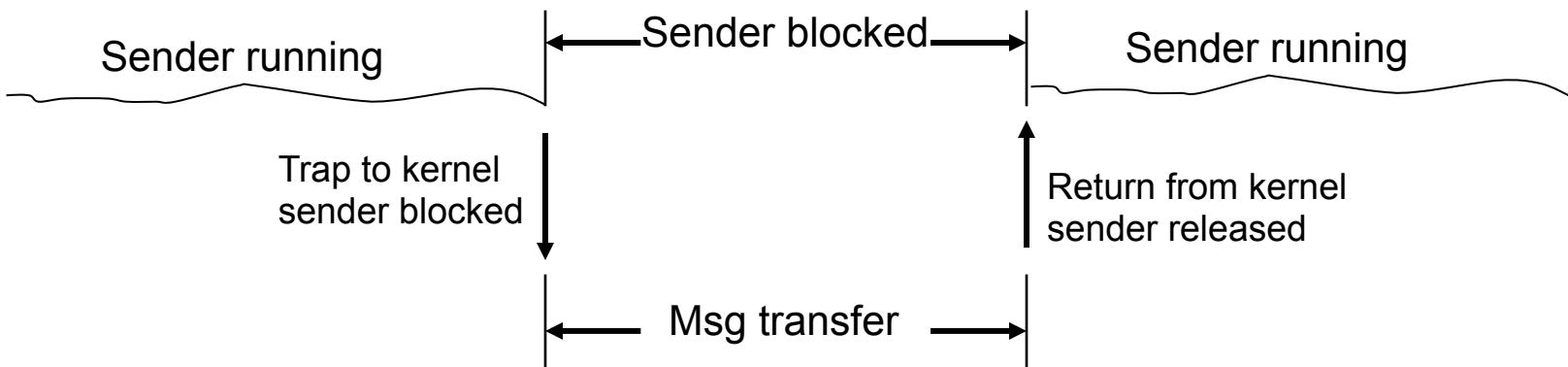


unnecessary
copying of
messages



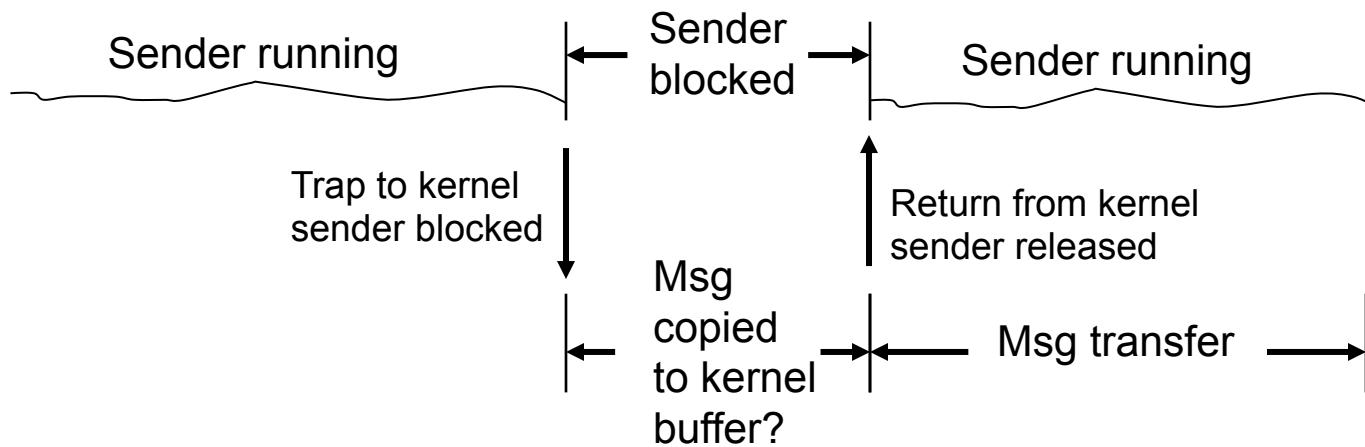
Multicomputer OS issues – Communication

- No shared memory, communication via msg passing
 - Easier to build, harder to program
- Send & receive, at the barest minimum
 - How to address a target node? cpu# and port# ?
 - Send/receive blocks thread until msg is sent/received



Multicomputer OS issues – Communication

- Send & receive, at the barest minimum
 - Non-blocking sender – send returns control to sender immediately, before msg is sent

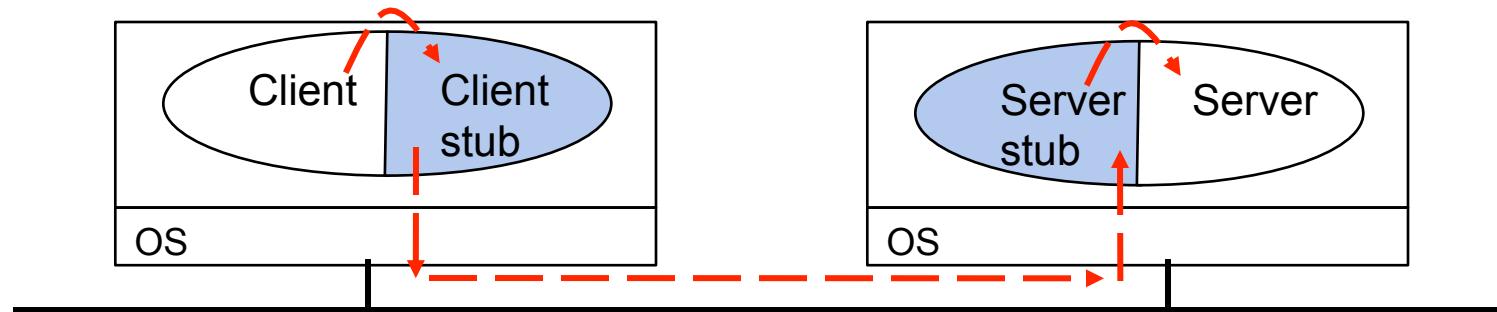


- Sender can go on with its work, but what about the unsent msg? Copy to kernel space? \$\$, COW?
- Non-blocking receive
 - Interrupts – costly, slow and complicated
 - Poll for arrived messages – how often?
 - Pop-up threads and active messages

Multicomputer OS issues – Communication

- Remote Procedure Calls

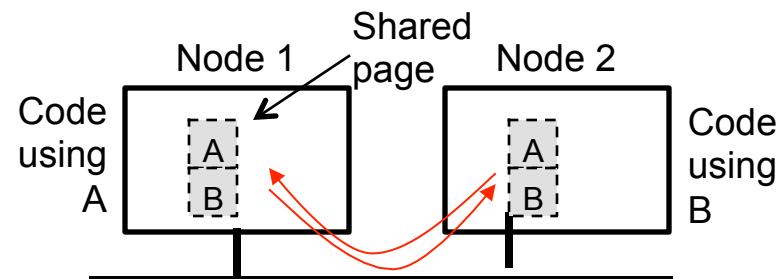
- Msg passing is I/O, for many, the wrong programming model
- RPC (Birrell, Nelson '84) – simple idea, subtle implications
 - Allow a process to call procedures in another machine
- To make it look like a procedure call - client and server stubs that pack and unpack the parameters and deal w/ the call



- Some problems
 - Pointers, weakly typed languages (array length?), ...
 - Global variables (now not longer shared)

Multicomputer OS issues – Memory

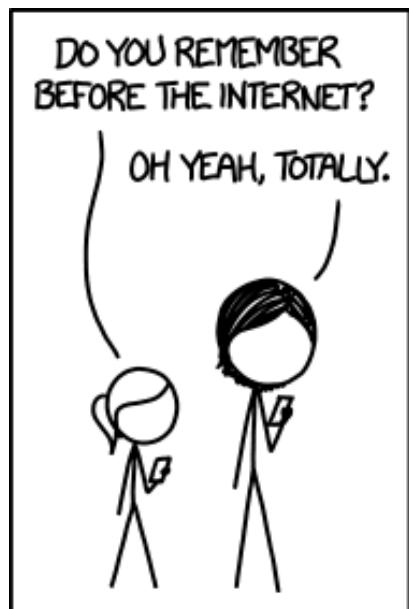
- Many programmers still prefer shared memory
 - Distributed shared memory without physically sharing it
- Basic idea
 - Each page is located in one of the memories
 - Each machine has its own virtual mem and page tables
 - When a page fault occurs, OS locates the page and requests it from the CPU holding it (remote memory, rather than disk)
- Improving performance
 - Replication of read only pages – easy
 - Replicating read/write pages – special handling to ensure consistency when pages are modified
 - False sharing
 - Failing machines ...



Multicomputer OS issues – Scheduling

- Scheduling done per node, each has its own set of processes – easier
 - But allocation of processes to nodes is key
- Allocation can be done in a number of ways
 - A graph-theoretic approach
 - Split graph (of processes/traffic xchng) as tightly coupled clusters
 - Schedule them to nodes
 - More distributed models
 - Sender-initiated – at process creation, if origin node is overloaded, pick a random node, if load is bellow threshold, move it there
 - Receiver-initiated – if a node loads is too low, pick another one at random and ask for extra work
 - At runtime – load balancing can be done at run-time if you expect a process to last for long enough to justify migration

And now a short break ...

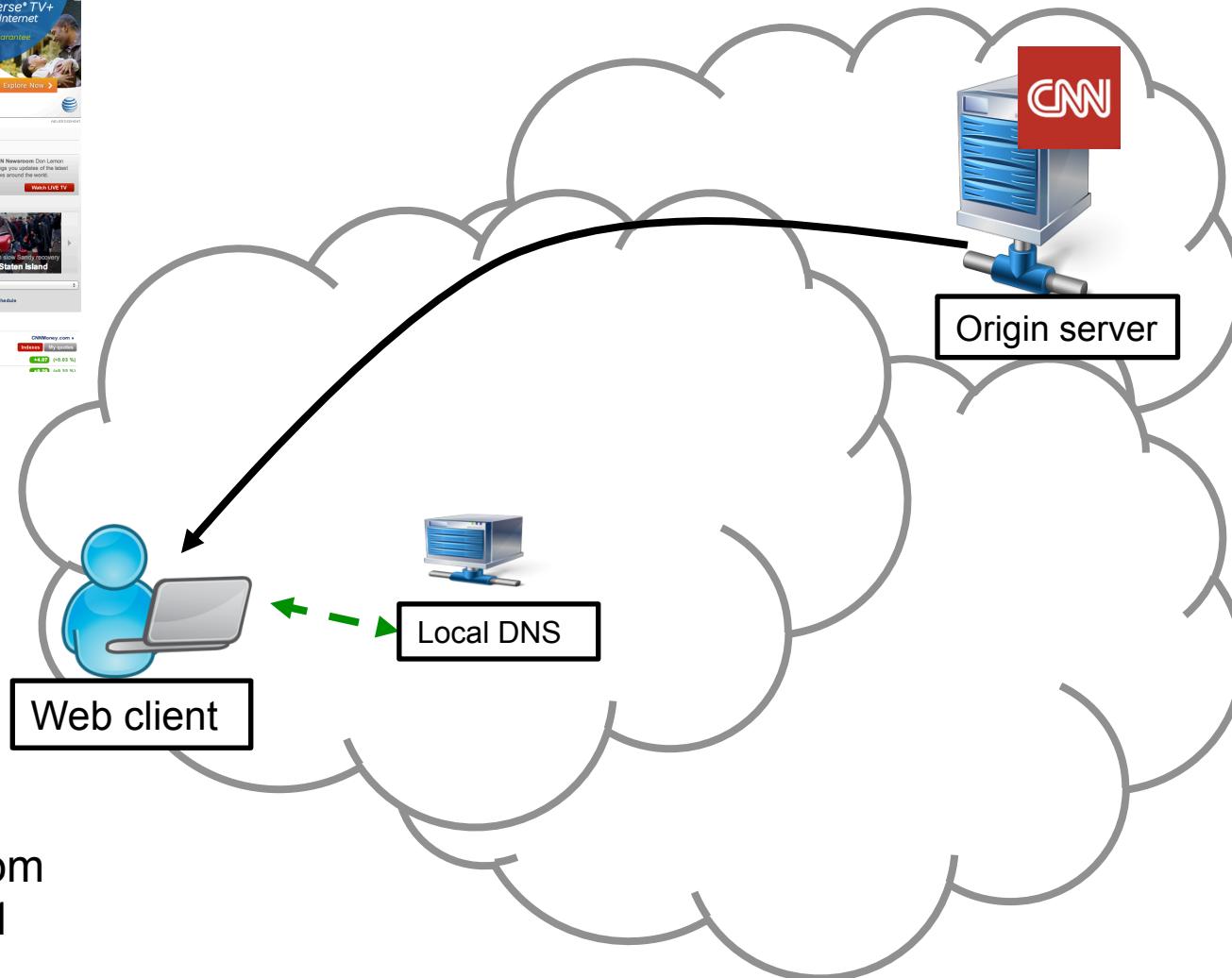


A “simple” visit to cnn.com

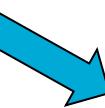


cnn.com?

% nslookup cnn.com
Server 192.168.1.1



A “simple” visit to cnn.com



Chartbeat

insightexpress[®]
discover tomorrow, today™

SCORECARDRESEARCH

TRUSTe
CERTIFIED PRIVACY

CNN

atlas.[®]
quigo
Deep Web Technologies

TURNER
BROADCASTING SYSTEM, INC.

nielsen

AOL

facebook

amazon
web services™



Akamai
AudienceScience[®]

Level(3)SM
COMMUNICATIONS

Google[™]

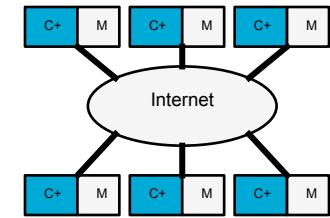
34 DNS lookups

204 HTTP requests

520 KB of data
downloaded

Distributed systems

- Very broad definition
 - Collection of independent, interconnected processors that communicate and coordinate through msg exchanges
 - ... appears to its users as a single coherent system
- Loosely-coupled - each system is completely autonomous
 - Each host runs an independent autonomous OS
 - Computers don't really trust each other
 - Some resources are shared, but most are not
 - The system may look differently from different hosts
 - Typically, communication times are long



Distributed systems – what for?

- Resource sharing – both, physical resources and information
- Computation speedup – to solve large problems, we will need many cooperating machines
- Reliability – machines fail frequently
- Communication – people collaborating remotely
- Many applications are by their nature distributed (ATMs, airline ticket reservation, etc)

Distributed systems challenges

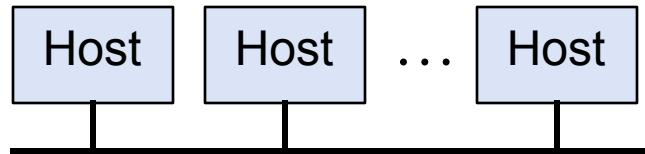
- Making resources available
 - A key goal of DS – making convenient to share resources
- Scalability
 - In numbers (users, resources), geographic span and administration complexity
- Providing transparency
 - Hide the fact that the system **is** distributed
 - Some types of transparency
 - Location – Where's the resource located?
 - Replication – Are there multiple copies?
 - Concurrency – Is there anybody else accessing the resource now?
 - Failure – Has it been working all along?
 - How *much* transparency?

Distributed systems challenges

- Openness
 - Services should follow agreed-upon rules on component syntax & semantics
- Security
 - Sharing, as always, introduces security issues
- Adding to this, common false assumptions
 - The network is reliable / secure / homogenous
 - The topology does not change
 - Latency is zero / Bandwidth is infinite /Transport cost is zero
 - There is one administrator

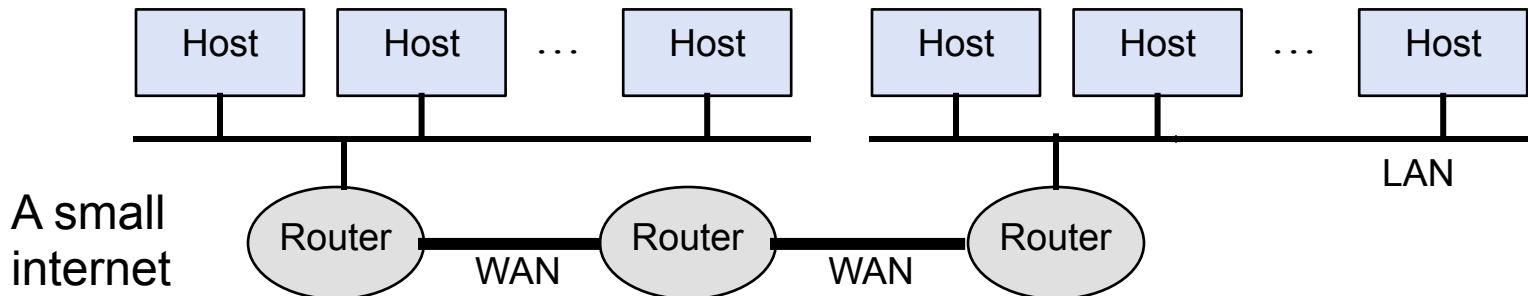
The network underneath

- Connecting hosts in a distributed system – a network
 - A hierarchical system (once) organized by geo proximity
- Lowest level – Local Area Network (LAN)
 - Most popular LAN technology – Ethernet
 - Ethernet segment – hosts connected through adapters and wires to a hub; segments can be bridge

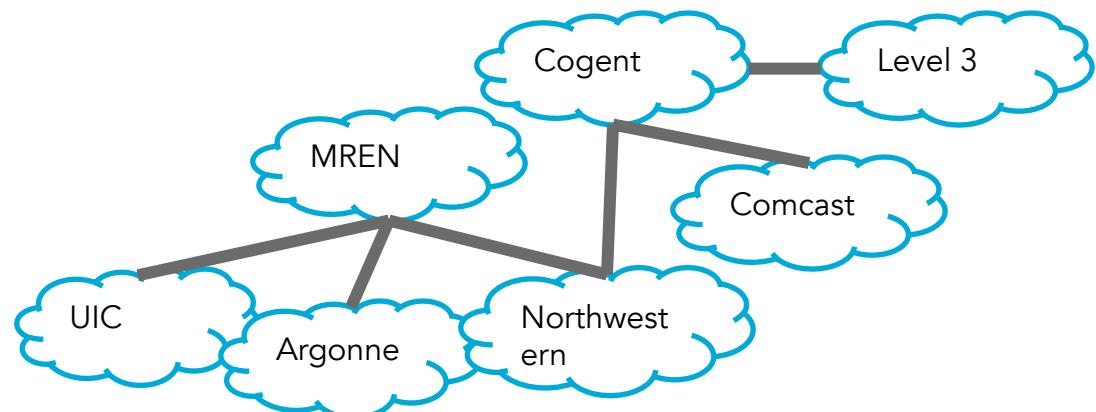


- Packetized – fixed packets (frames) include header + payload
- Each adapters has a globally unique address (MAC)
- Broadcast network – Carrier Sense Multiple Access with Collision Detection (CSMA-CD)

The network underneath

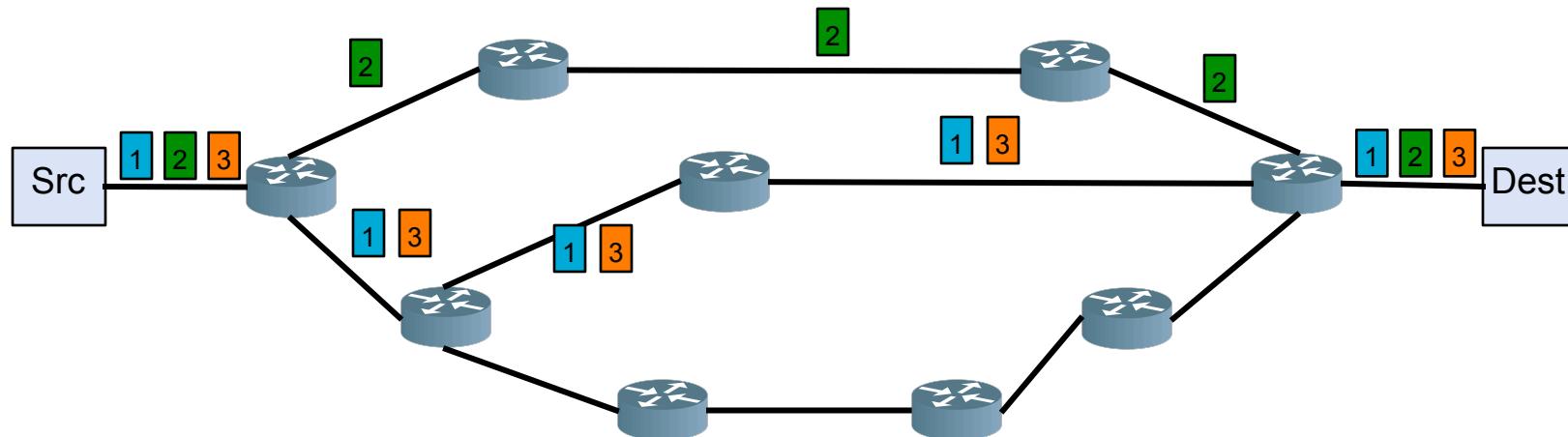


- Higher in the hierarchy, LANs connected through specialized computers *routers* to form an internet
- Internet is made of collections of independently managed internets/networks – aka autonomous systems (AS)



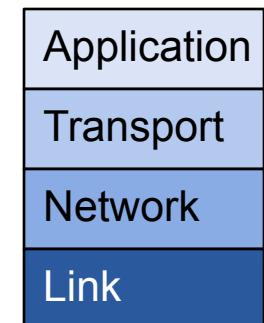
A packet switched network

- Data is broken into packets, each packet is routed separately from src to dest
- Contrast with circuit switching – a physical circuit is reserved from src to dest (e.g., buffer space)
- Links and routers
 - Links physically move packets from place to place
 - Routers receive packets from incoming links and put them on outgoing ones toward the dest



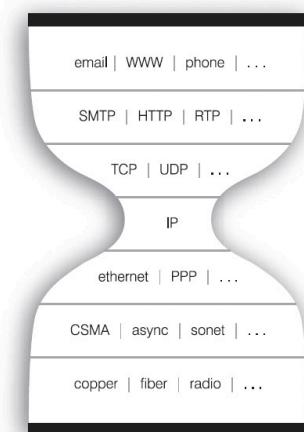
From source to destination

- Using packet switching networks to provide reliable, efficient communication is complex ...
- Layering protocols
 - Link – physically interfacing with communication medium to move a packet over a link; Ethernet, WiFi 802.11
 - Network – moving packets over multiple links, from src to destination; IP (Internet Protocol)
 - Transport – provides a msg service to apps; apps don't need to care that communication is in packets; TCP, UDP
 - Application – implementing app need; Web, DNS, P2P, email ... and protocols like HTTP, DNS, SMTP ...



Network layer – Internet protocol

- An unreliable, best-effort service
 - “Sorry, buffer’s full so I got to drop your packet”
- Each connection between a node and a network, an *interface*, every interface a unique address (IP)
 - A 32bit for IPv4 NU webserver 129.105.215.254
 - Often group by their prefixes 129.105.0.0/16
- The IP hourglass, a design principle
 - Overarching goal of Internet tech development – connectivity
 - “IP over everything”



The network is unreliable

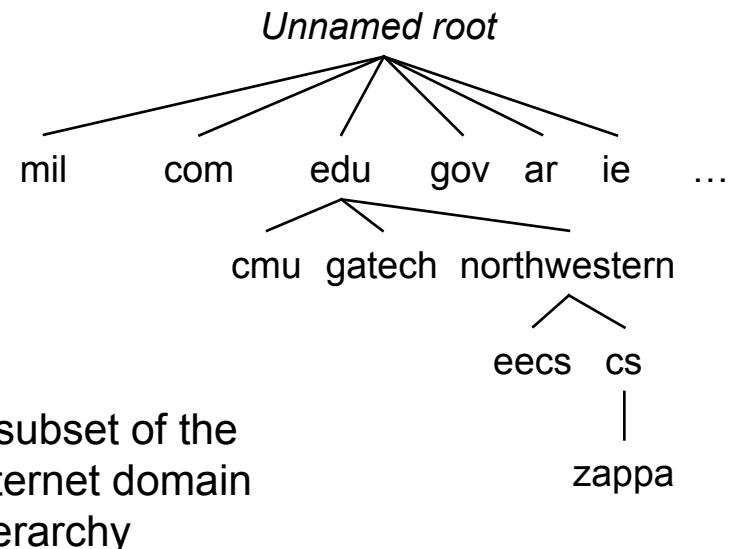
- Packets get lost or corrupted on the way
 - Flipped bits, links down, buffers overflow, ...
- One approach, ignore it – the application can handle it
 - UDP/IP (User Datagram Protocol)
- For reliability
 - How does the sender knows the receiver got it? ACK
 - What if the ACK gets lost? Timeout
 - What if the ACK just gets delayed? Counters
 - ...

Reliable transport on an unreliable network

- Transmission Control Protocol (TCP)
 - Reliable multi-packet messages from unreliable single-packet datagrams
- With TCP/IP and lower layers we get
 - Reliable delivered of ...
 - multi-packet messages between ..
 - two address spaces in two different machines over heterogeneous networks;
 - all without caring for the details
- Higher protocol layers facilitate specific services
 - SMTP for email, HTTP for Web, FTP for file transfer, ...

From names to IP addresses

- Large strings of integers (IP) are hard to remember
- To make it easier, use human friendly domain names
zappa.cs.northwestern.edu (165.124.180.8)
- Originally mapping was kept in a text file (HOSTS.TXT)
- Since 1998 through a distributed DB – Domain Name System (DNS)



Now that you can communicate ...

- How do you order events happening in different machines without a common clock?
 - Keeping all clocks somewhat synchronized
 - Ignore real-time clocks, just logical ordering could be enough
- How do you ensure mutual exclusion?
 - E.g., Token passing, voting, ...
- Replication for fault tolerance or performance
 - How do you read the NYT when you are in Europe?
 - Where do you place replicas?
 - How do you keep multiple replicas consistent?
- ...
- A whole class just to get you started ...

More? Take EECS 340, 345 or 3/495s

- 340 – Introduction to networking
 - The nitty-gritty details on how machines talk to each other
- 3/495 – Internet-scale experimentation
 - Designing, building and measuring Internet-scale distributed systems and their underlying network
- 345 – Distributed systems
 - Naming things, communicating, ordering of events, some reliability, self-destructing data for increased privacy ...
- 3/495 – DS in challenging environments
 - Hot and crazy ideas pushing distributed systems to uncomfortable spaces (outer-space, under-water, downtown, volcanoes, ...)