Distributed Programming

- What is distributed programming?
- What are some problems it’s used for?
- How is it organized?
  - Multiple computers under the same management
  - Multiple computers run by different entities
- What are issues that must be solved?
- Examples
  - Seti@Home / BOINC
  - Google’s map/reduce
  - Teraflop-scale processors

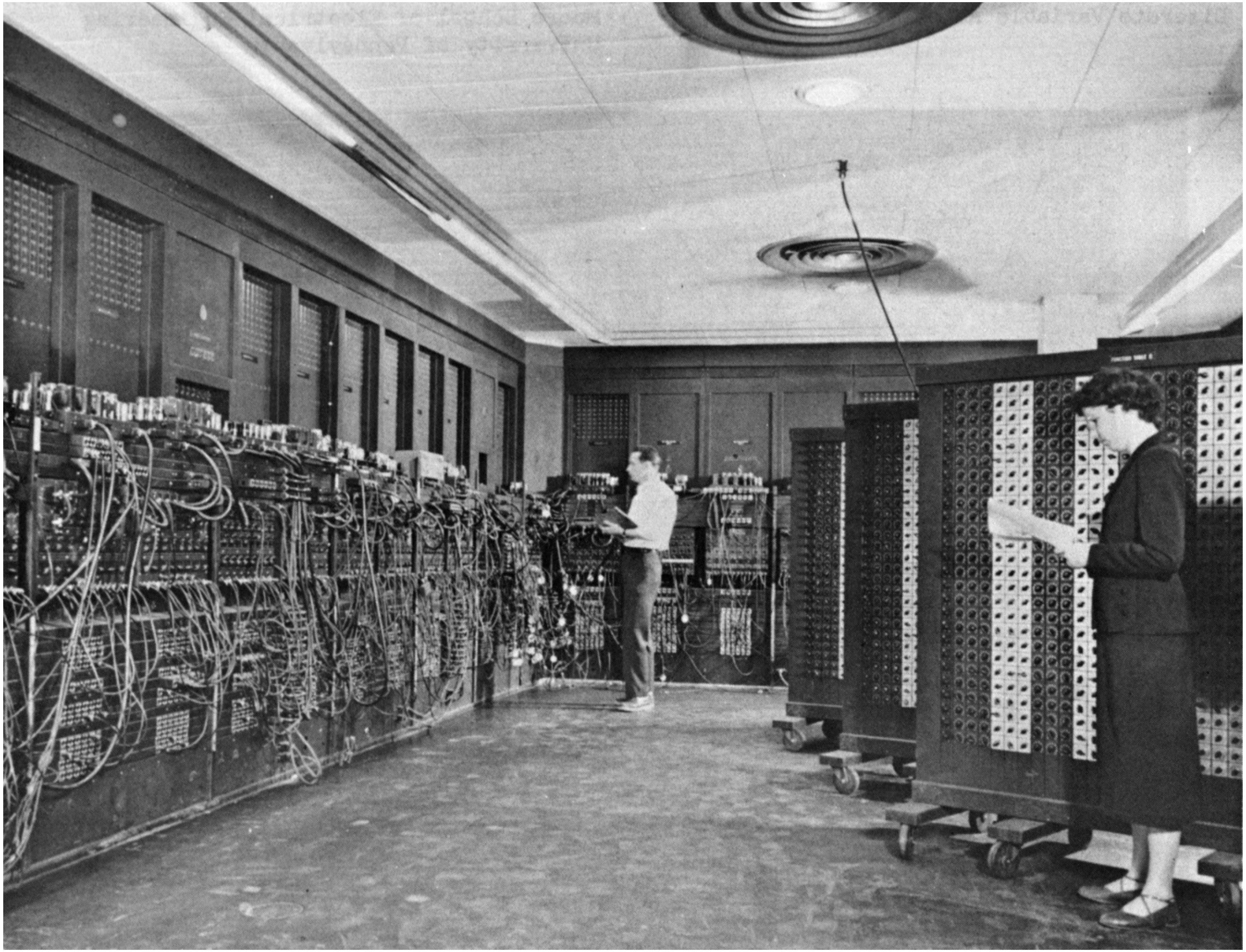
What Is It Used for?

- Large-scale scientific problems
  - Finite element models: simulate reality by following the laws of physics at each of millions of points
    - Weather modeling
    - Structural analysis
    - Earthquakes
  - Biochemistry models (protein folding, DNA sequencing)
- Large-scale searching
  - Internet
    - Store and search a piece of the Web on each node
    - Results are combined and returned
- Signal processing
  - Look for features in terabytes of signal

History of High-Performance Computing

- 1946: ENIAC (Electronic Numerical Integrator And Computer)
- 1960: Network of vacuum-tube computers US Air Force (SAGE)
- 1985-95: Supercomputers (Convex, Cray, Thinking Machines)
- 1985: VAX VMS clusters by Digital Equipment Corporation
- 1990: Clusters of Unix workstations (HP, SGI, SUN)
- 1990-95: Transputers by Inmos and Parsytec
- 1995-00: Beowulf concept: Linux based low budget PC's
- 2000-: Linux clusters: Linux based (very) high budget PC's

ENIAC



Osaka University Vacuum Tube Computer

CRAY X-MP/24

Beowulf Clusters 1

- Beowulf: Name of the first Linux cluster by NASA in 1994
- Main goal: high and reliable computing power at low costs
- Features:
  - Based upon standard off-the-shelf Intel PCs
  - Operating system Linux (sometimes with special drivers)
  - Dedicated interconnecting network (no external traffic allowed)
  - Master server accessible from outside and through monitor/keyboard
  - Other nodes (Slaves) only equipped with a network card
  - Master NFS shared file system: code transparent on all slaves
  - Message passing software: mostly MPI and PVM

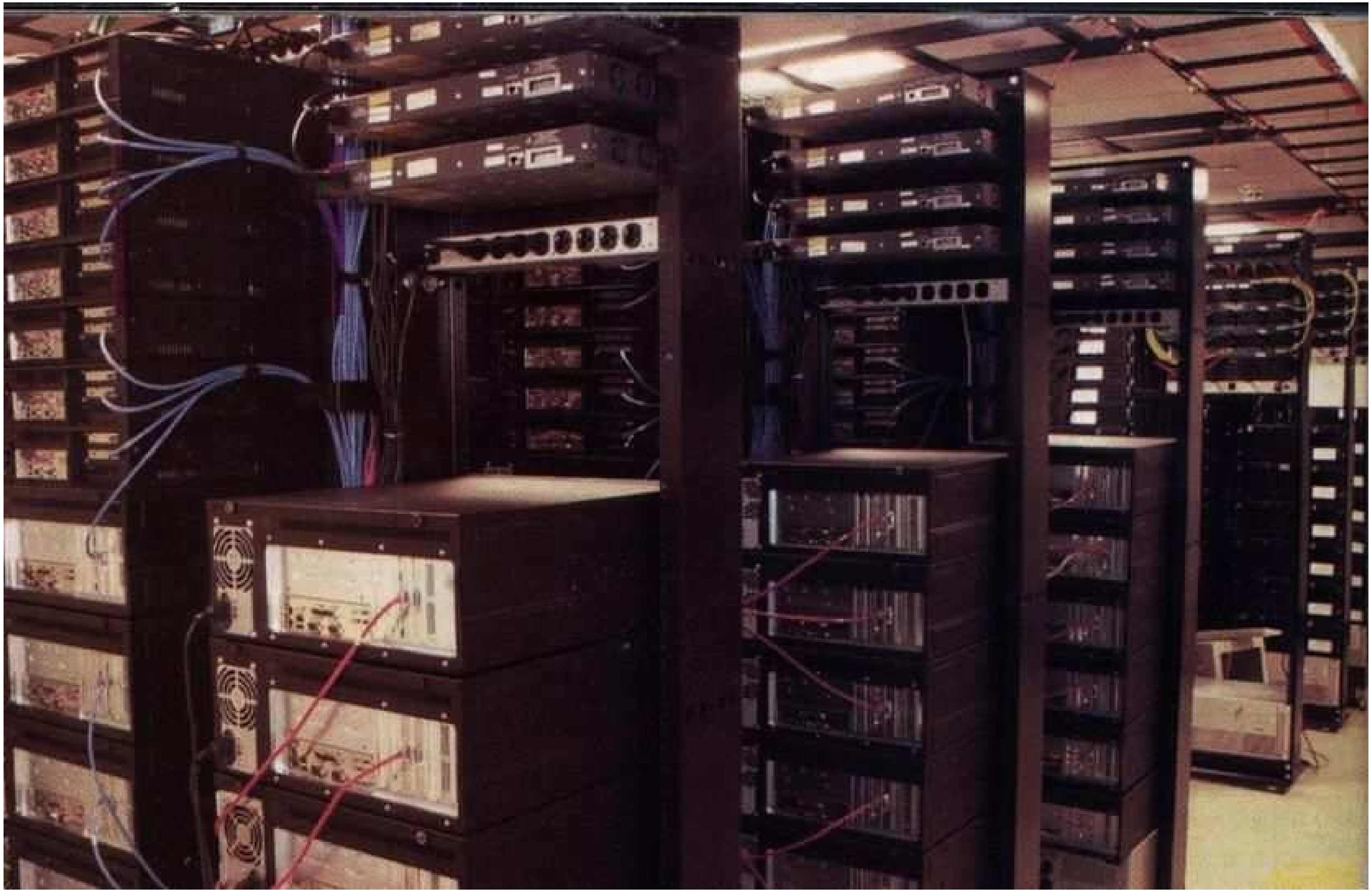
Beowulf Clusters 2

- Advantages
  - Hardware cheap compared with workstations or supercomputers
  - Software is mostly open source and free (Linux, MPI, PVM, GCC)
  - Easy to expand with new systems or join with other clusters
  - Many choices for the interconnecting network
  - Very reliable and stable (but depends on hardware)
- Disadvantages
  - PC hardware less reliable than workstation or supercomputer
  - Depending on network: higher latencies (e.g. for TCP/IP)
  - Failure detection on cluster nodes difficult (no monitor/keyboard)
  - Single entry point: only master accessible from outside

Beowulf Clusters 3

- Types of interconnecting networks
  - Serial or RS323 port connections or SLIP (0.1 Mb/s , 100 ms)
  - Parallel port connections or PLIP (1 Mb/s , 100 ms)
  - Ethernet 10Mbit, 100Mbit (10-100 Mb/s , 100 μs)
  - Ethernet Gigabit (1 Gb/s , 200 μs)
  - Shared versus Switched network routers
  - Dedicated low latency network cards:
    - Myrinet (10 Gb/s, 2 μs) and Infiniband (2 Gb/s/channel, 1 μs)

Computing Cluster at The American Museum of Natural History in New York City



Cluster of PCs at The ORNL - The Stone Souper Computer

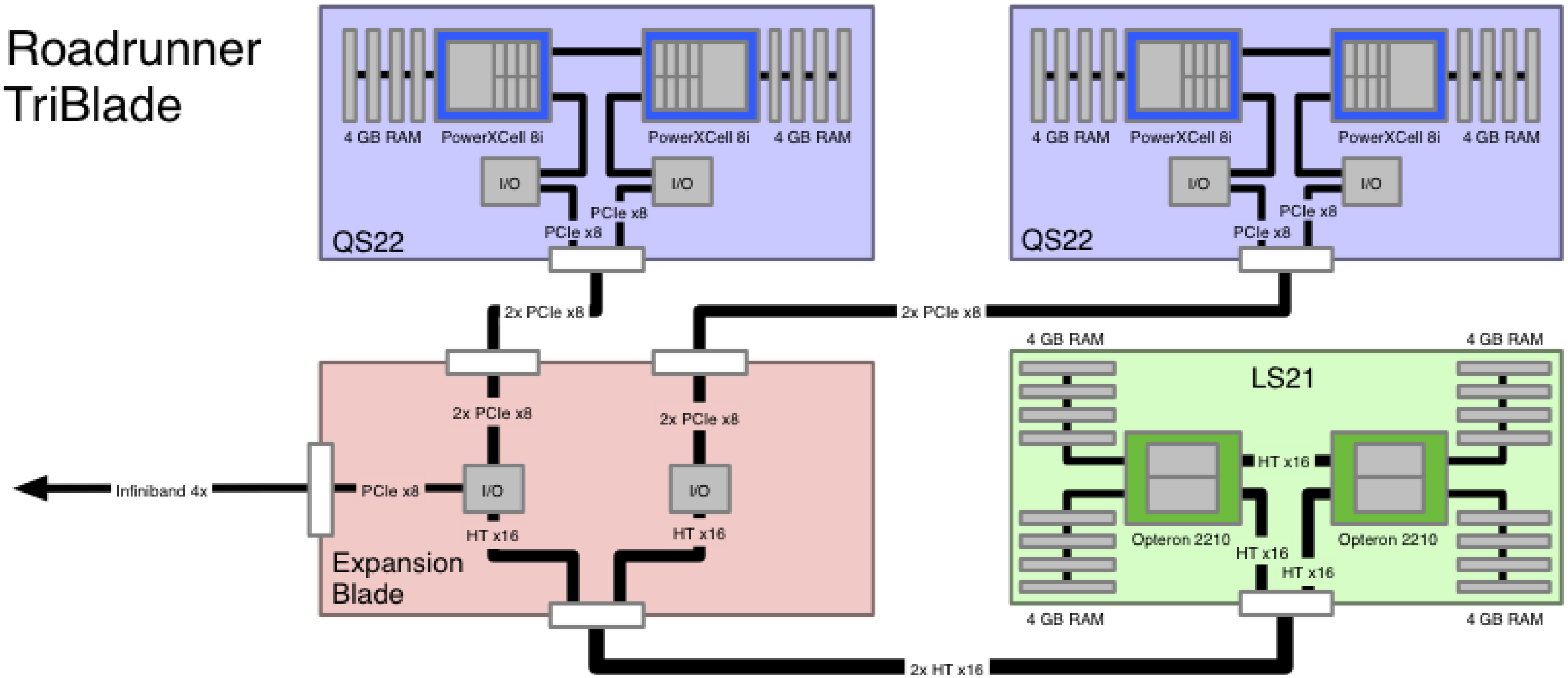
HP BladeSystem c7000 Populated with 16 Blades

IBM Roadrunner - 1.1 Petaflop Supercomputer

12,960 IBM PowerXCell 8i CPUs and 6,480 AMD Opteron dual-core processors

IBM Roadrunner - Schematic of A TriBlade

Roadrunner TriBlade



Organizing Distributed Programs

- Several available programming styles
  - Message-passing
  - Object-based
  - “Work packets”
- Each is best suited for different kinds of problems
  - Often, there’s overlap between programming styles
  - Choice depends on how tightly coupled the program must be

Coupling

- Coupling is the amount of coordination that individual nodes must have with each other
- Loosely coupled: not much coordination
  - Task can be neatly divided into pieces
  - Individual pieces don’t have much to do with one another
  - Results combined at the end
- Tightly coupled: closer coordination between nodes
  - Intermediate results depend on those from other nodes
  - Communication occurs frequently
  - Nodes need to be somewhat synchronized so one node doesn’t get ahead of others

Message Passing

- Work divided among nodes
- Nodes communicate via messages as needed
  - MPI (Message Passing Interface) commonly used
- Messages used to
  - Synchronize nodes
  - Parcel out work
  - Communicate intermediate results
- Code on individual nodes often, but not always the same
  - Typically, data is divided among nodes
  - Sometimes, functionality is divided
    - Mail filtering and delivery
    - Large-scale web service (DB, session handling, page serving)

Distributed Objects

- Similar to message passing
- Each object does its own processing locally
- Remote methods work like you'd expect
  - Remote objects run computations
- Typically slower for large-scale computations
  - Distributed objects usually synchronous
  - Easier to move data rather than moving small chunks of computation
- May get used in systems where function is different for each node

“Work Packets”

- Computation divided into many small independent pieces
  - Monte Carlo simulation
  - SETI
  - Large number factoring (decryption)
- Each node takes some units of work
  - Units are independent
  - Node completes them, reports back to coordinator
- Coordinator tracks who’s doing what
  - Reschedules work that hasn’t been done yet
  - Failures handled by redoing work units

Distributed Computing Organization

- Computers largely identical
  - Owned by a single organization
  - Configured the same way
  - Comparable abilities (speed, I/O, etc.)
- Computers somewhat different
  - Controlled by a single organization
  - Configured similarly
  - May have widely disparate abilities or network speeds
- Computers unrelated
  - Owned by different organizations
  - Configured differently
  - About the only constant is free CPU time

Identical Computers

- Often called parallel computing, especially if nodes connected by high-speed networks
- Same environment everywhere
  - Tools & binaries available on every node
  - File system looks the same everywhere
- Security provided by
  - Single username across nodes
  - Security at the “front door”
- Good for large-scale scientific workloads
- Typically very expensive!

Similar Computers

- Computers under single administrative domain
  - Similar available resources (users, files)
  - No worries about malicious nodes that might corrupt results
- Computers have different capabilities
  - Some nodes are much faster than others
  - Computation can’t proceed in “lock-step”
  - Need to take differences into account in scheduling
- May be able to use spare cycles on workstations along with “centralized” resources
  - “Night-time supercomputer”
- Jobs managed centrally
- Good fit for loosely coupled problems

Unrelated Computers

- No common configuration or administration
  - Different user name at each location
  - No set of common resources except those brought along explicitly
  - Capabilities wildly different
- Security issues
  - Malicious nodes might run the program incorrectly
  - Malicious nodes might steal your data!
- Communication can be dicey
  - Nodes may have slow network connections (or even fail)
  - Failures have to be handled by a coordinator
- Typically only done with “work units”

Issue: Dividing The Problem

- Need to take a big problem and chop it up
- Divide it arbitrarily
  - Job has n things that need to be done
- Divide it up “physically”
  - Usually works with problems that simulate the real world
  - Pieces correspond to regions of the phenomenon being simulated
  - May run into problems if some pieces are harder than others
- Divide it up into multiple runs
  - May rerun the same simulation lots of times (Monte Carlo method)
  - Simulations are largely independent

Issue: Combining The Results

- Each node does a small part of the work
- Results need to be combined
  - Assembled: “concatenate” results together with no additional processing: common for physical problems
  - Reduced: results from individual nodes are “merged”
    - Search
    - Simulations
- This part of the program is difficult to speed up

Issue: Coordination

- Distributed systems have to work together
- Who decides how they do this?
- Single (central) coordinator
  - Simple to program
  - Less efficient: central bottleneck
  - Prone to failure
- Multiple central coordinators
  - Must coordinate amongst themselves
  - More scalable than single coordinator
- Fully distributed: nodes coordinate amongst themselves
  - Difficult to program
  - Most efficient approach

Issue: Running Code

- Every node needs to run the right code!
  - Easy in a tightly coupled system
  - Difficult in a loosely coupled system, especially if it’s run by multiple organizations
- Solution: require each node to run code that loads other code as needed
  - Java Virtual Machine
  - Local daemon that downloads binaries as needed
- Solution: use a distributed file system
  - Every node can get the code it needs
  - Use the Web for this…

Issue: Security

- Two security challenges
  - Protect system from code running on it
  - Protect running code from the rest of the system
- Protecting the system from the distributed code
  - Trust the code?
    - Big risk
  - Limit privileges
    - Run as a user that has few (if any) abilities on the system
  - Restrict CPU usage
- Protecting the distributed code from the system
  - Prevent data and computation from corruption
  - Prevent the system from stealing the data (yes, this is a real issue in many distributed systems)

Google's distributed Web index

- Google has an index of the entire Internet (or at least a lot of it)
- Index is too large to fit on a single machine
  - Hundreds of terabytes of data
  - Thousands of disks
- Too much activity for only a single server
  - Need to divide the problem up
  - Handle requests on multiple servers
- Divide up data and requests

Google's Approach

- Divide Web index by batches of pages
  - Crawl the Web continuously
  - Each set of pages is on a single machine
  - Each machine has indices for all words in those pages
- Queries go to all machines
  - Quickly search individual machines for set of pages
  - Results merged separately
- Failures handled by ignoring that part of the Web
  - It'll be recrawled soon anyway
  - Recently, some redundancy was included

Google's Map/Reduce

- Google's approach is called map/reduce
  - Computation is mapped to all of the nodes
  - Results from all nodes are combined (reduced) into a single result set
  - Uses specialized coordinators to do it
- Google has techniques to do this quickly!
  - Make individual nodes fast
  - Redo computations that fail (or just don't respond)
- Google's file system is optimized for this
  - Special operations for appending to large files
  - Optimized for dealing with gigabyte files
  - Not good for smaller stuff….

Berkeley Open Infrastructure for Network Computing

- BOINC is a system for doing distributed computation across a large set of unrelated nodes
  - Distribute small units of computation to individual (home) computers
  - Gather results from them
- Platform is designed to run any software that can break up computation this way
  - SETI
  - Protein folding
  - Climate change

BOINC Details

- Jobs divided into very small tasks
  - Search a small portion of the sky
  - Try several protein folding combinations
- Clients download
  - Software for a particular task
  - List of small tasks to run
  - Data for the small tasks
- Clients run the software on the small tasks
  - Respond back with results when they’re finished
- Coordinator ships out tasks and collates results
  - Often hands tasks to multiple computers
    - Check accuracy
    - Deal with failure

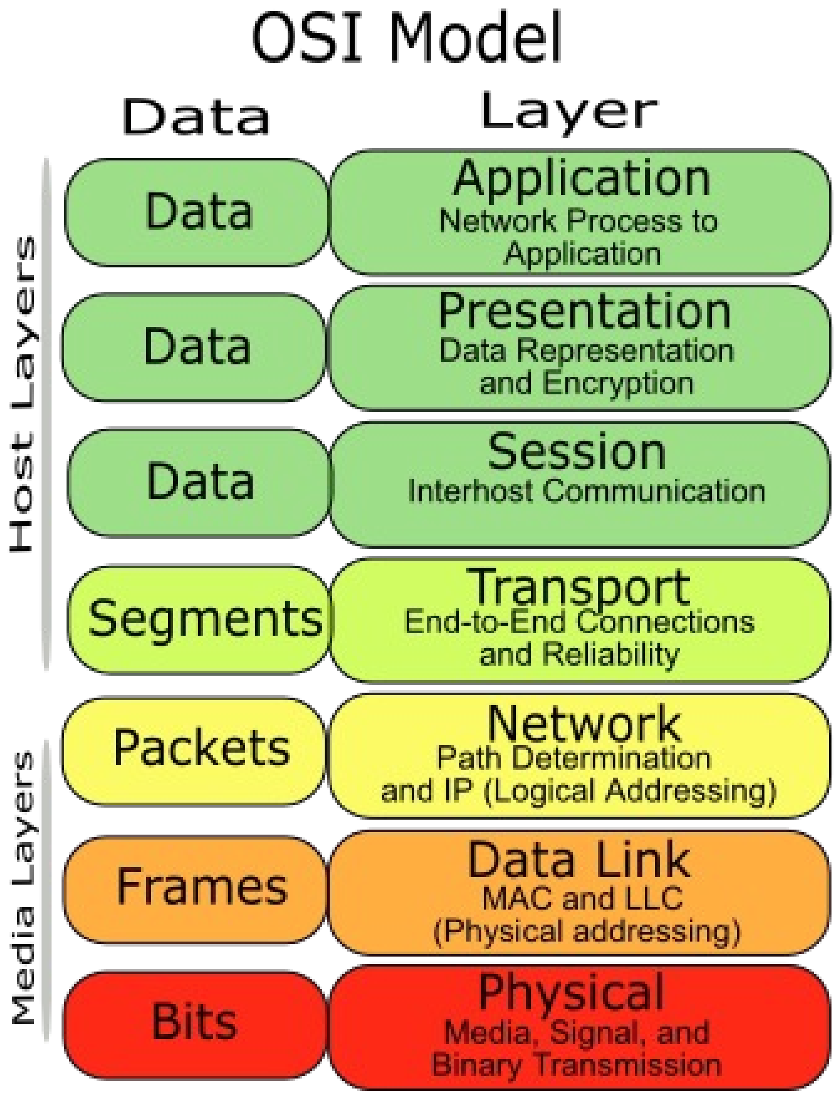
More on BOINC

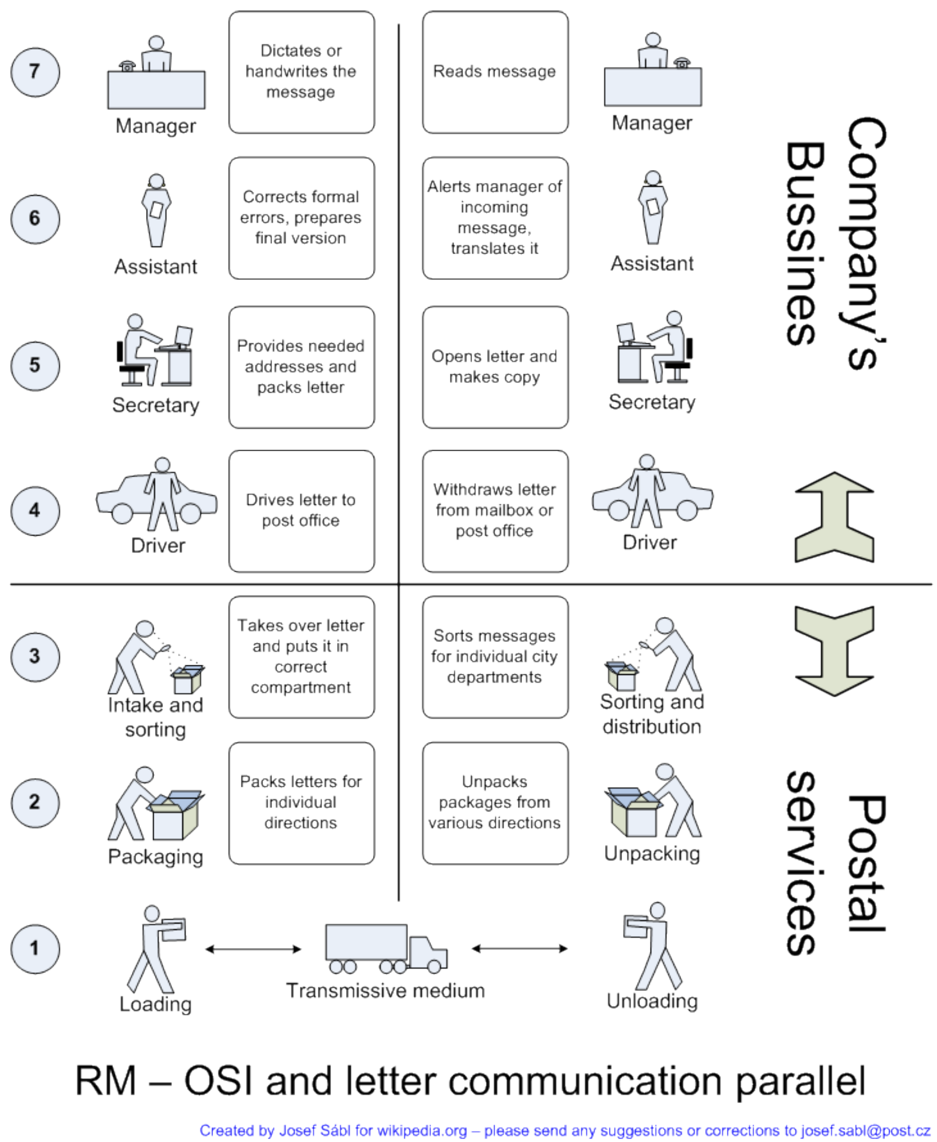
- System is open: others can create new programs to run across the world
- System runs on multiple platforms
  - Deal with byte ordering
  - Different assembly languages (PowerPC, x86, etc.)
  - Wide range of different speeds
- Reliability isn't so great
- Must be very loosely coupled
  - No guarantee on how long each task will take
  - Communication may be spotty
  - Data is not sensitive (no worry about client stealing it)

Communication in Distributed Systems

- Lack of shared memory, necessity to send messages
- Basic operations: Send() and Receive()
  - Client sends a request and waits for an answer
  - Server receives the request and sends a response
- Messages can be reliable or unreliable (when unreliable, higher layer must provide reliability of transmission)
- Can be based on a specialized protocol or a general-purpose protocol (such as TCP/IP)

OSI Reference Model





April 1st, 1990 - RFC 1149

Standard for the transmission of IP datagrams on avian carriers

Implementation: http://www.blug.linux.no/rfc1149/

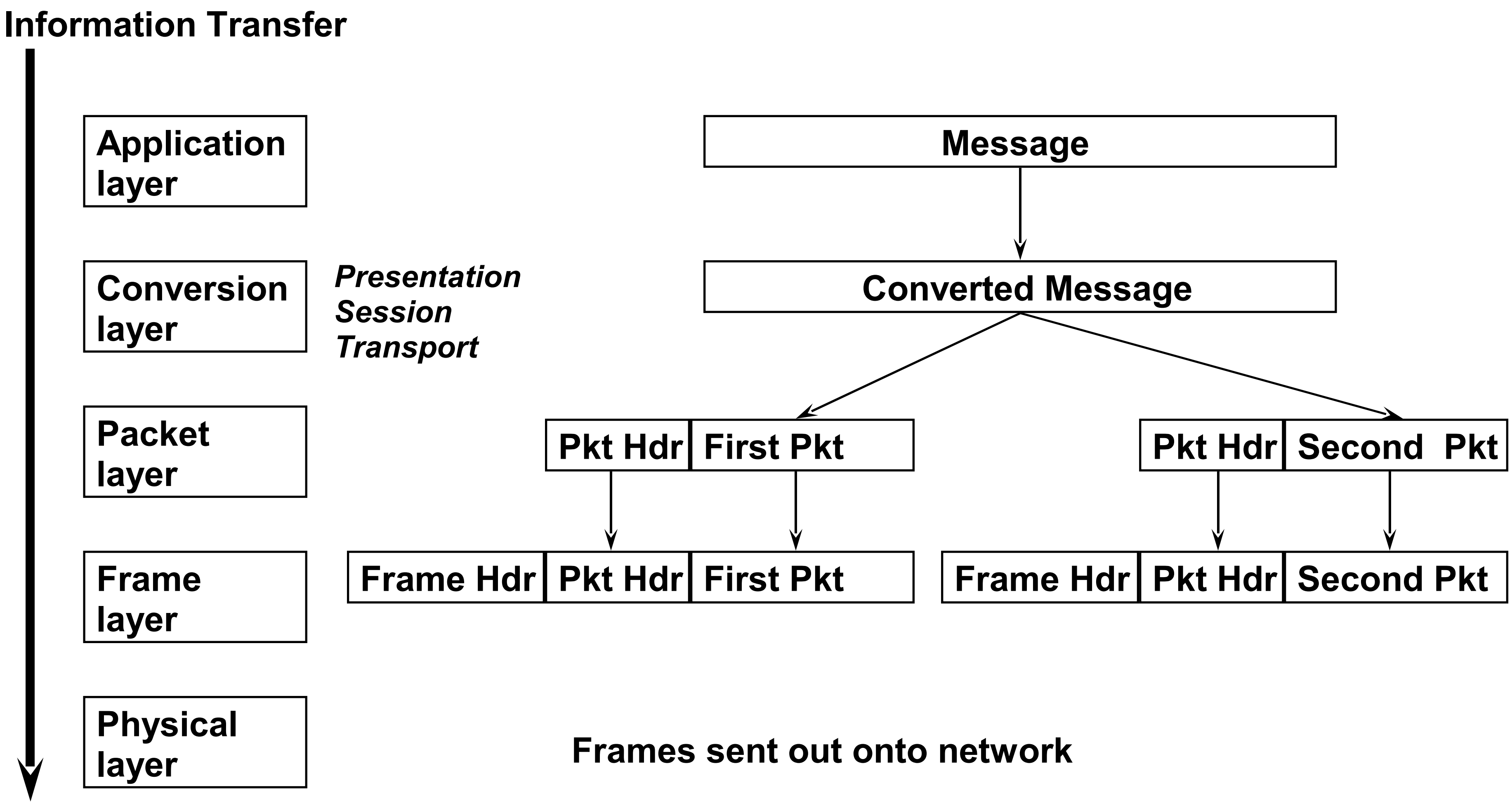
```
Script started on Sat Apr 28 11:24:09 2001
vegard@gyversalen:~$ /sbin/ifconfig tun0
tun0   Link encap:Point-to-Point Protocol
       inet addr:10.0.3.2  P-t-P:10.0.3.1  Mask:255.255.255.255
       UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:150  Metric:1
       RX packets:1 errors:0 dropped:0 overruns:0 frame:0
       TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0
       RX bytes:88 (88.0 b)  TX bytes:168 (168.0 b)

vegard@gyversalen:~$ ping -i 900 10.0.3.1
PING 10.0.3.1 (10.0.3.1): 56 data bytes
64 bytes from 10.0.3.1: icmp_seq=0 ttl=255 time=6165731.1 ms
64 bytes from 10.0.3.1: icmp_seq=4 ttl=255 time=3211900.8 ms
64 bytes from 10.0.3.1: icmp_seq=2 ttl=255 time=5124922.8 ms
64 bytes from 10.0.3.1: icmp_seq=1 ttl=255 time=6388671.9 ms

--- 10.0.3.1 ping statistics ---
9 packets transmitted, 4 packets received, 55% packet loss
round-trip min/avg/max = 3211900.8/5222806.6/6388671.9 ms
vegard@gyversalen:~$ exit

Script done on Sat Apr 28 14:14:28 2001
```

Implementation of Layers



Exercise 1

- Implement a server computing roots of real numbers and providing current date and time on server
- Based on TCP protocol
- To ensure portability between different processors, transmit all numbers in Big Endian format
- RQ ID allows to distinguish different requests

Request for square root:

0	0	0	1	RQ ID	Number (IEEE double)

Response:

1	0	0	1	RQ ID	Root (IEEE double)

Exercise 1

- Date and time sent in textual form, without terminating zero
- Length sent in Big Endian format
- During one connection several requests can be sent
- Order of responses can be different than order of requests

Request for time and date:

0	0	0	2	RQ ID

Response:

1	0	0	2	RQ ID	Length (BE)	Time and date