CS5223
Distributed Systems

Lecture 2: Communication

Instructor: YU Haifeng

Today’s Roadmap

- Chapter 4 of textbook
- Review (Overview) of the OSI model
- Remote Procedure Call / Remote Method Invocation
  - If you don’t know Java, read some tutorials on Java (e.g., “Java in a Nutshell” https://www.amazon.com/Java-Nutshell-Desktop-Quick-Reference/dp/1098131002)
  - Should be quite easy if you already know C++
- Multicast
- Gossiping

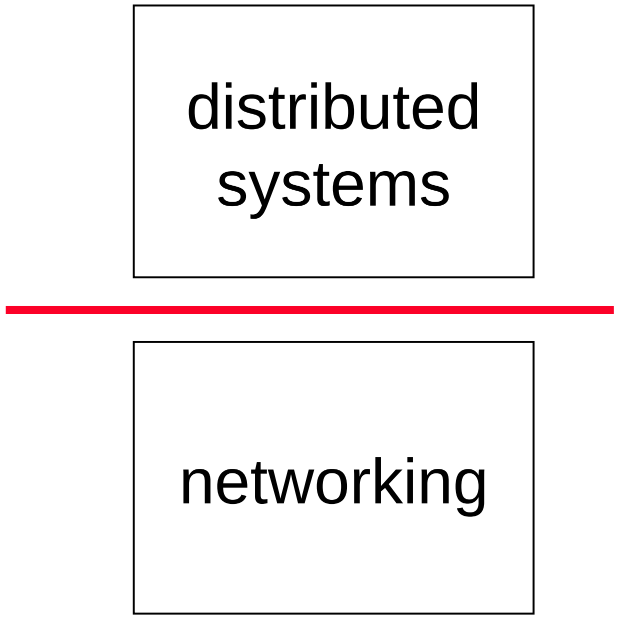
Motivation

- Distributed systems by definition need communication

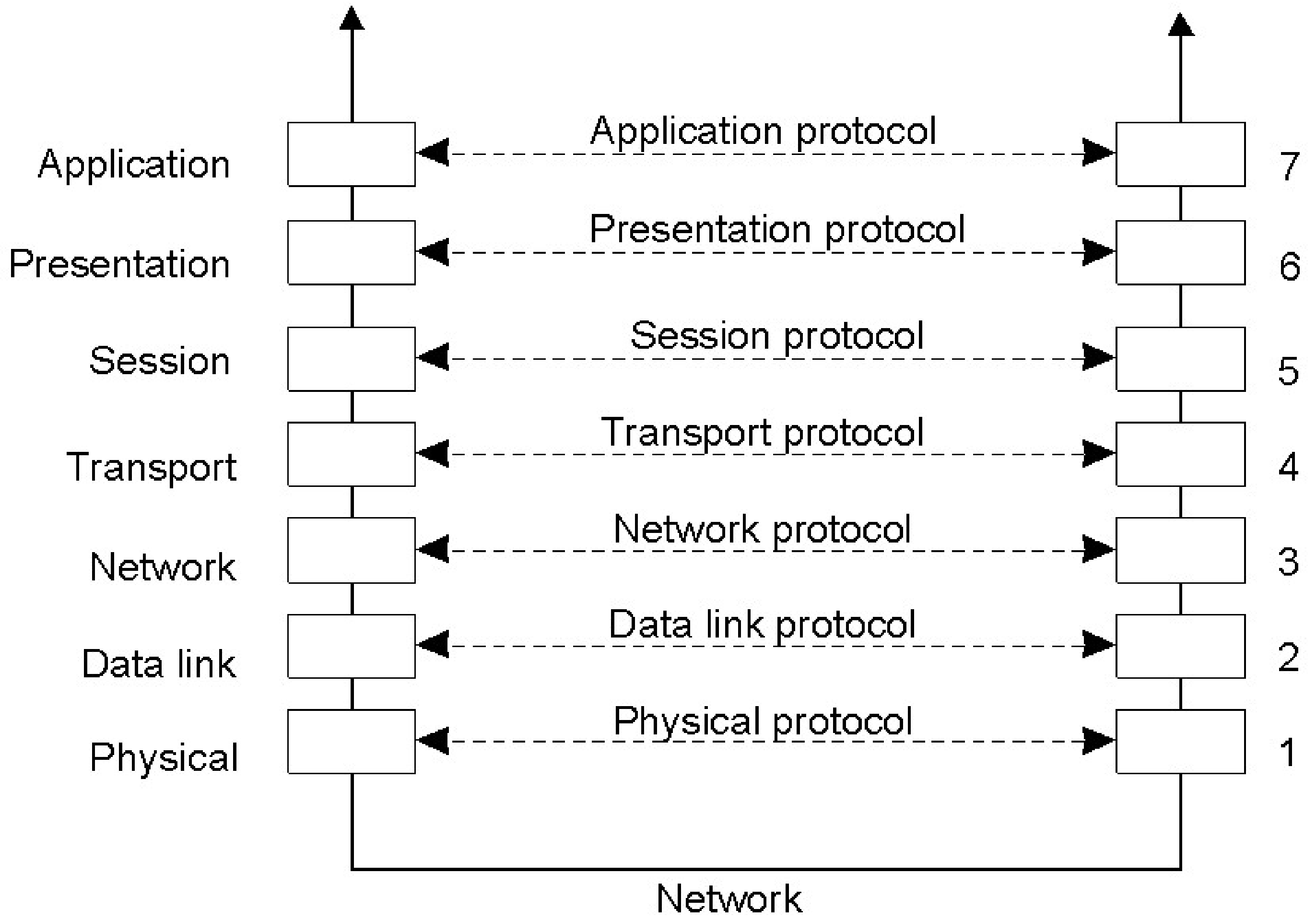
- Networking:
  - Aims at providing delivery of data
  - Does not care about the data content

- Distributed systems:
  - The layer above networking
  - Does not care how the data is delivered – sometimes treat the network as a black-box

- Pros and cons of treating network as a black box?
  - Example?
  - PollEv.com/haifengyu229



Overview (Review) of OSI Model



Open Systems Interconnection Reference Model

More Details on Each Layer

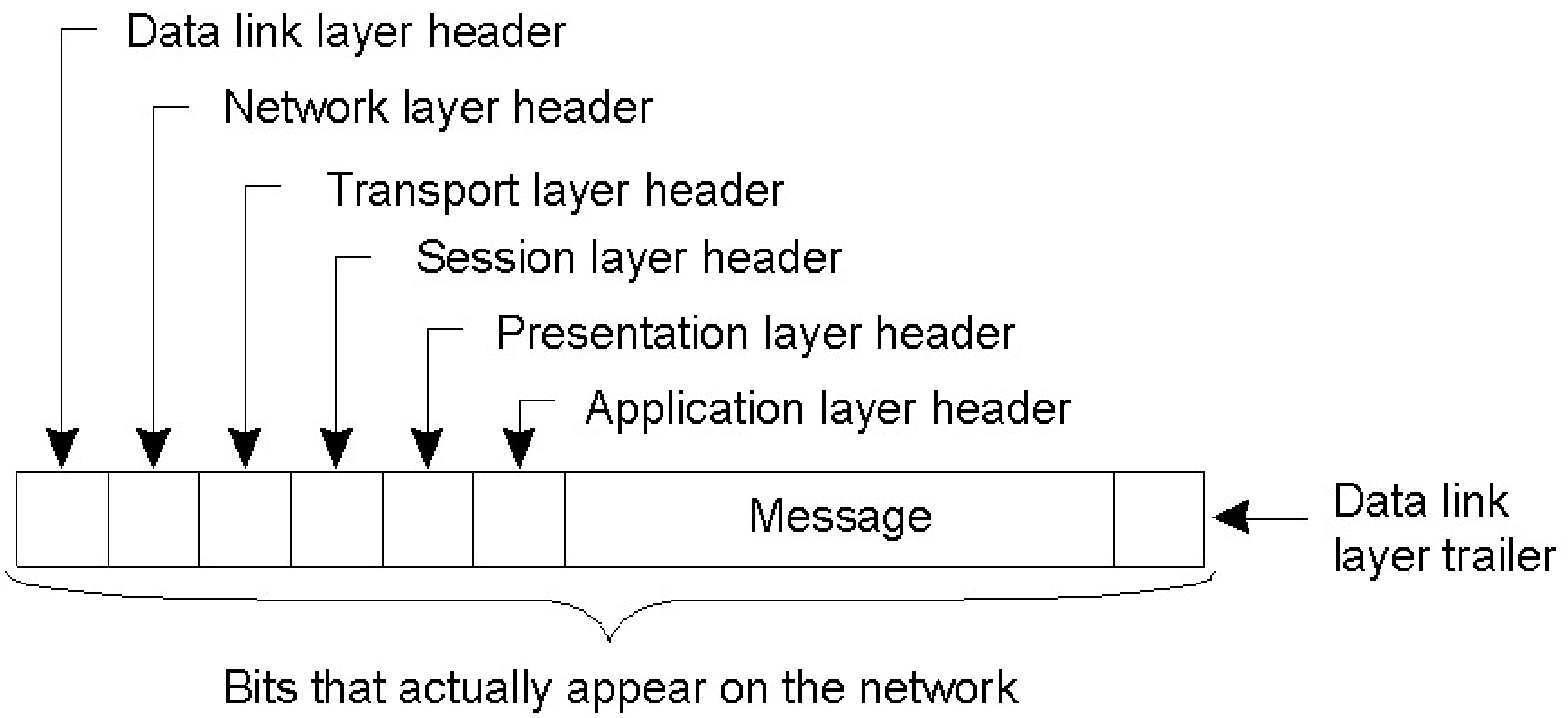
- Physical layer
  - Transmission of bits between sender and receiver
- Data link layer
  - Transmission of frames (series of bits) with error correction and detection
- Network layer
  - Describes how packets are *routed along multiple hops*
  - Prominent example: IP (Internet protocol)

Transport Layer

- Transport layer provides the actual communication facilities for most distributed systems

- Standard Internet transport-layer protocols
  - TCP (transmission control protocol): Reliable, in-order, with flow control, stateful
  - UDP (universal datagram protocol): Unreliable (best-effort) datagram connectionless communication

Messages in the OSI Model



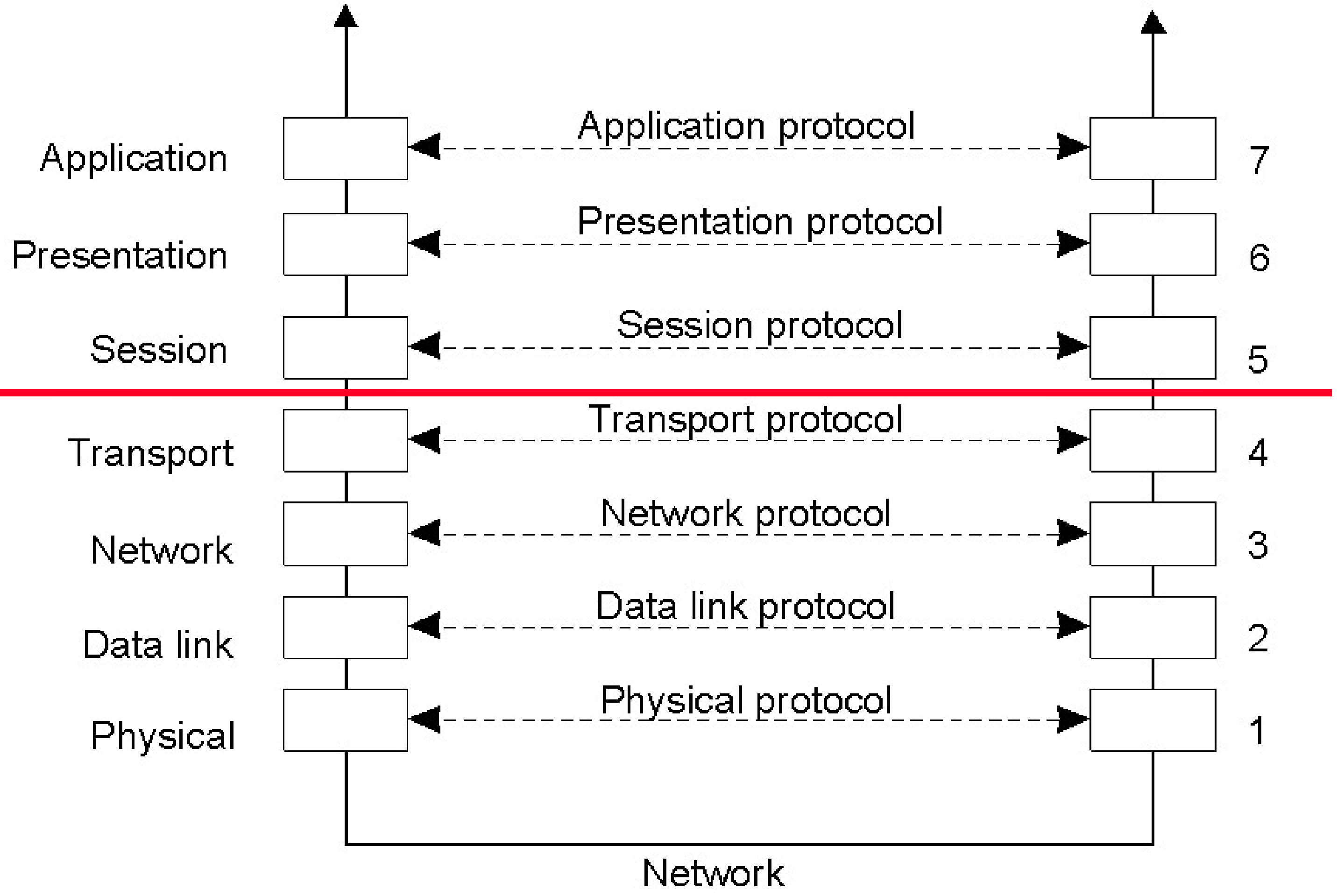
OSI Model - Continued

- The OSI model does not dictate how you implement each layer
- Nothing fancy – just the application of layered modular design principle
  - Neither the only model nor the best model (no single best model)
  - But it is widely accepted – you want to speak the same language as other people
- Should NOT view it as cast into stone: The layered modular design principle is more important than the model itself
- OSI aims for generality, and one can also customized it in different context
  - Why should we customize it?
  - Example?

The concepts of Application, Presentation, and Session layers are not widely used.

Distributed Systems -- may have its own layers

Networking



Motivation for More Powerful Communication Mechanisms

- Many distributed systems directly use TCP/UDP
  - Prominent examples: Distributed games, BitTorrent
  - In theory, all distributed systems can be built by directly using TCP/UDP

- But many distributed systems share common communication pattern / requirements
  - Middleware / libraries / tools / modules to simplify their design
  - Again, just a software engineering principle (what principle?)

More Powerful Communication Mechanisms

- Remote Procedure Call / Remote Method Invocation
- Multicast
- Gossiping

Motivation: Remote Procedure Call

- Many distributed systems involve the interactions between a client and a server
  - Example: Clients wants to read the content of a file on a remote machine
  - So common that we want to develop a common middleware / library / toolkit to simply their designs

- One approach (not the only approach): Remote procedure call
  - Example: read(fileid, buffer, numbytes)

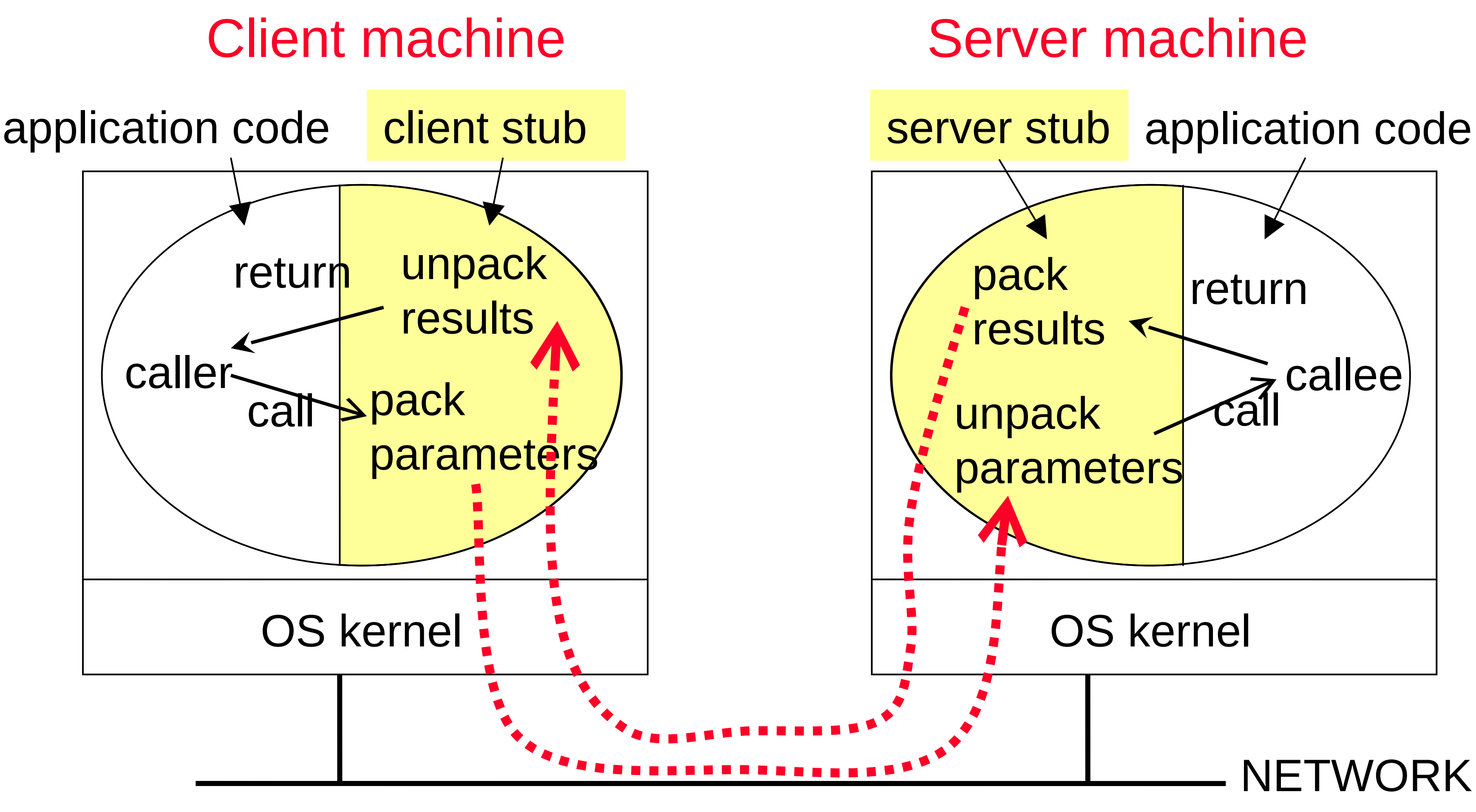
Motivation: Remote Procedure Call

- Sending the parameters to the remote machine and then wait for a reply
  - Simple (and potentially efficient) to implement
  - Can do so using a general middleware that work for all remote procedure calls

- Why RPCs are good
  - Application developers are familiar with procedure calls
  - RPCs operate in isolation – modular design

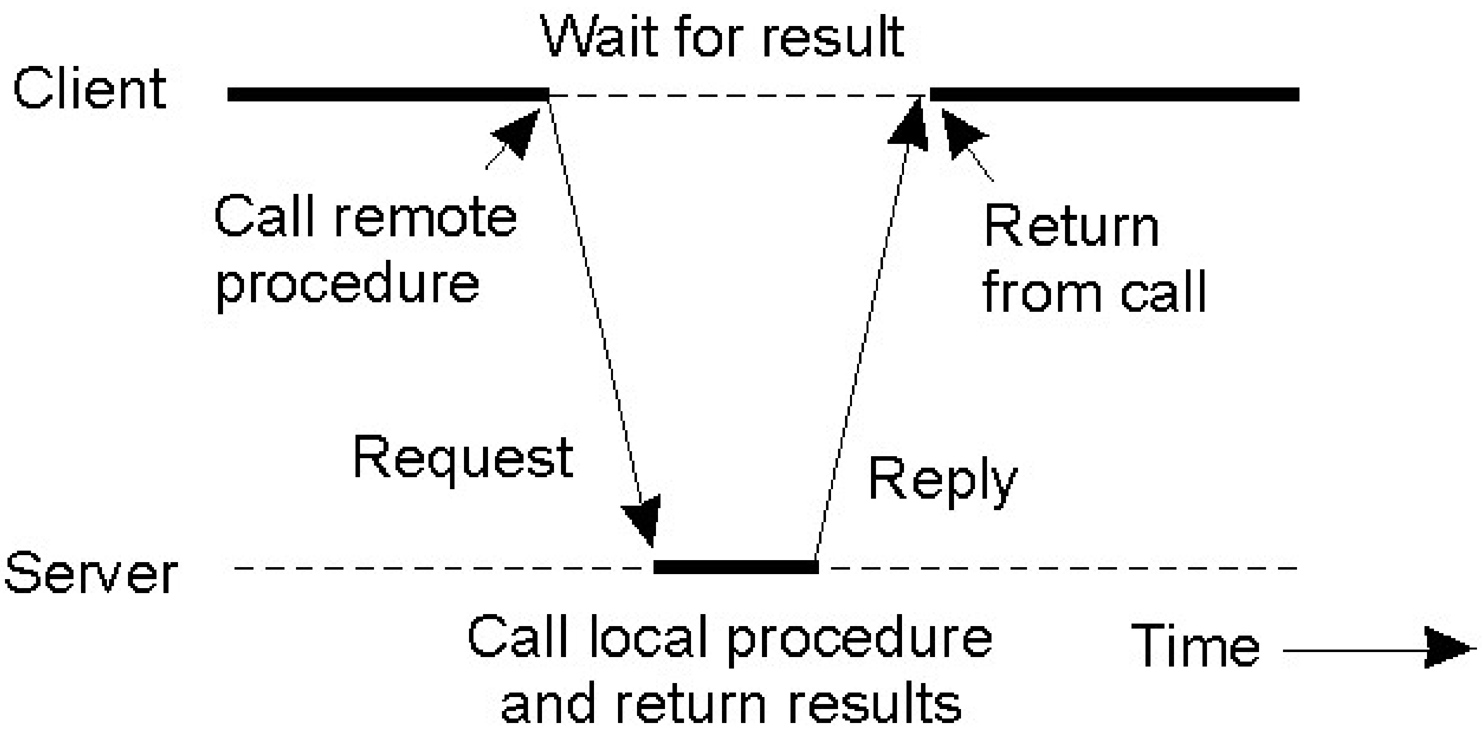
- History has proved the huge success of RPC
  - Java and C# have RPC support built into the language

Typical Architecture for Enabling RPC

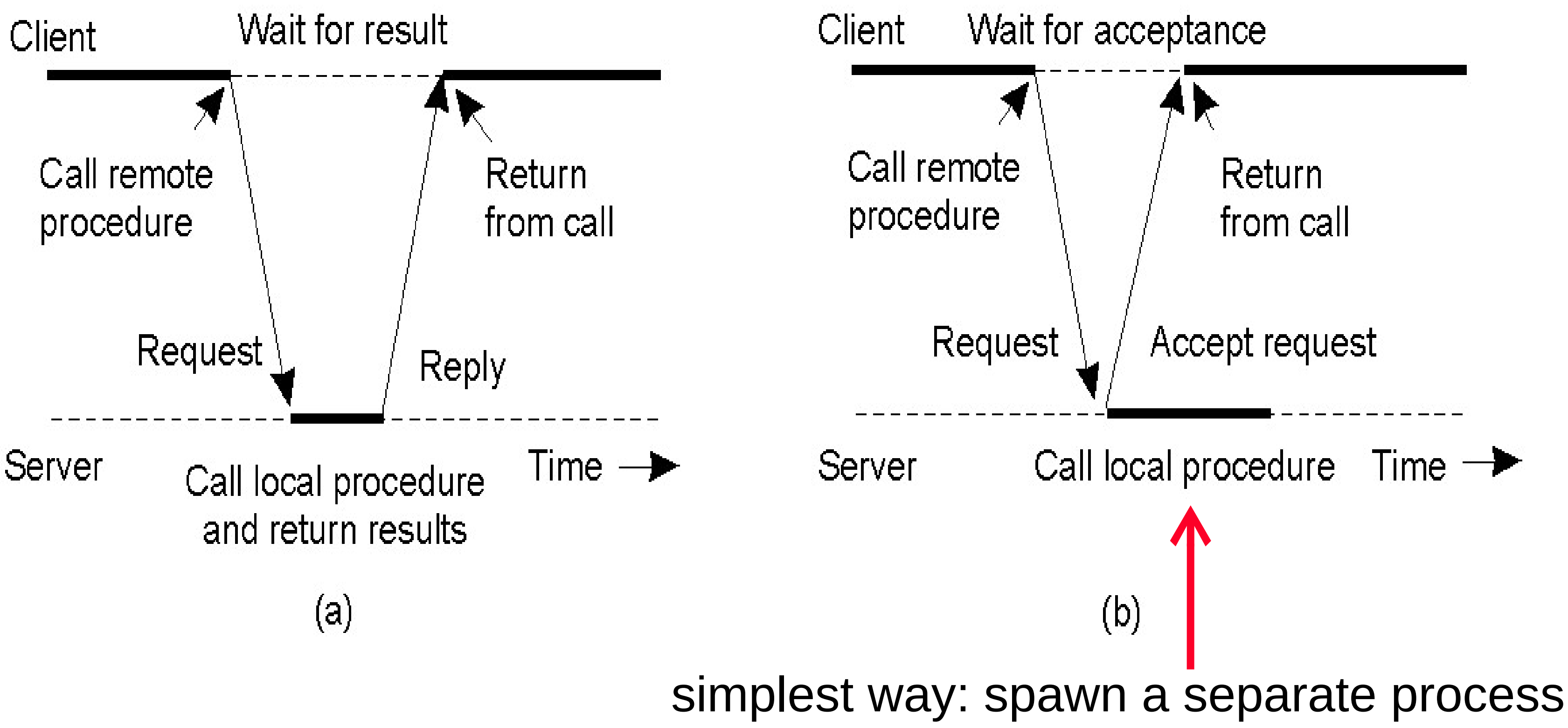


Yellow parts are provided by the RPC mechanism

RPCs are by definition blocking



Making RPCs Non-blocking



- Non-blocking RPCs depart from procedure call semantics
- If you use a lot of non-blocking RPCs, re-think whether you indeed want to use RPC

RMI: A Case Study for RPC

- RMI is just object-oriented RPC
  - Each function (i.e., method) in Java belongs to some object
  - read(file, buffer, numbytes) ⇔ file.read(buffer, numbytes)
- file is a reference to a “remote object”
- We will focus on RMI as an example from now on

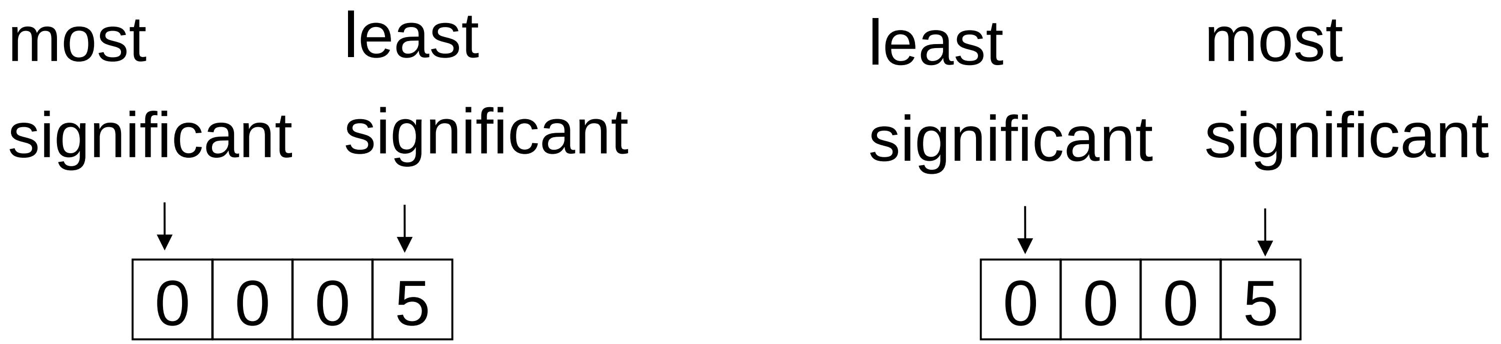
Java Remote Objects

- What is needed in a remote object reference
  - Need to ensure uniqueness over space – no other object has the same “name”
  - Need to ensure uniqueness over time – no other objects should have the same “name” in the past or in the future

IP addr	port num	creation time	obj num	methods

Marshaling: Primitive Data Types

- Marshaling: Client needs to send the parameters of the function call to the server
  - Wrap all parameters into a message and send the message
- Network transmits raw bytes: Client and server may have different representation of data



4-byte integer on Pentium 4-byte integer on SPARC

Marshaling: Java (Non-remote) Objects

- How to marshal complex data types (i.e., objects)
  - Example: A Vector of integers
  - Java object serialization: convert a Java object to a bit string (for sending over the network or storing on disk)

- Java serialization extremely useful:
  - How to store a binary tree?

- What about a vector of vectors?
  - When serializing an obj, all obj it references are serialized as well (and this goes on… how?)
  - Understand the object you are serializing

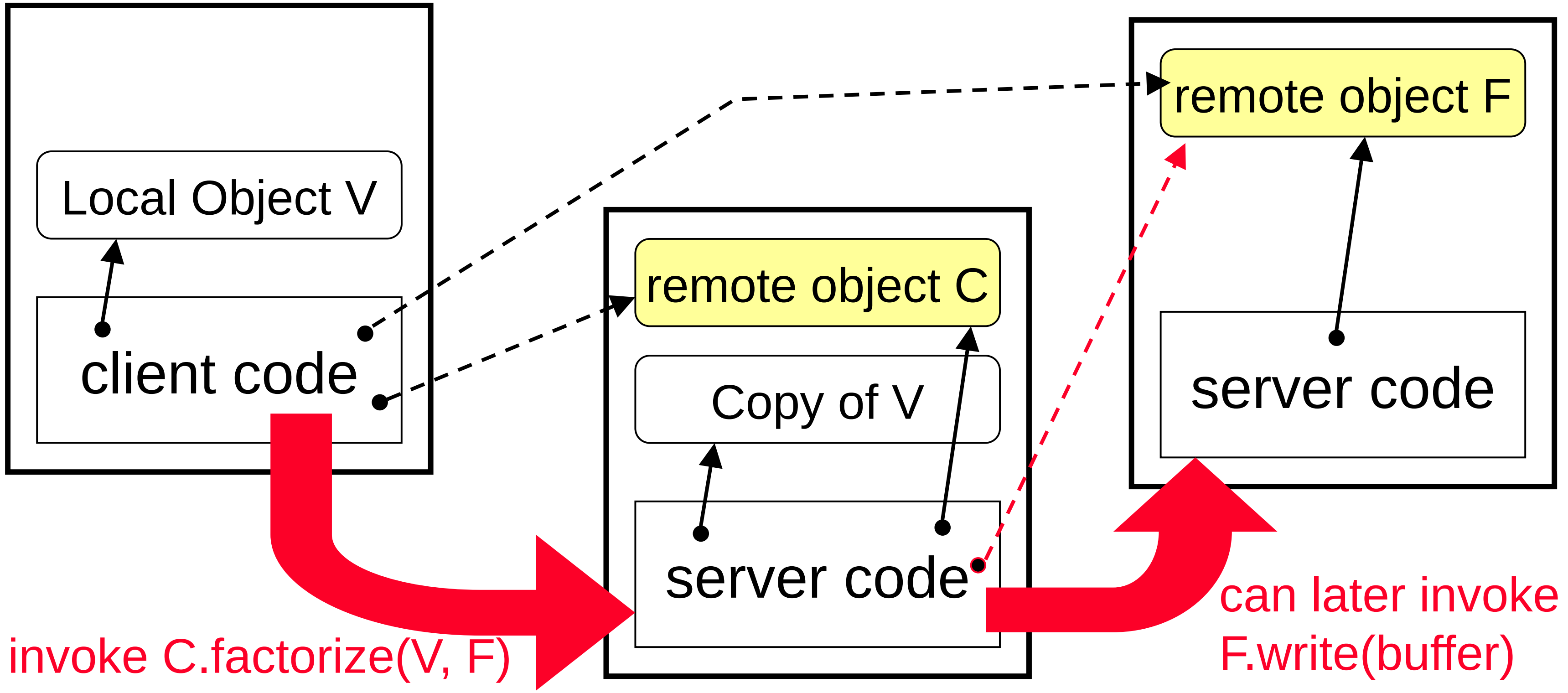
Marshaling: Java (Non-remote) Objects

- Unmarshaling on server: Use de-serialization and convert the bit stream back to object
  - A new object will be created on server

- Use of object serialization/de-serialization =
  Semantics of RMI is different from local method invocation
  - Server will work on the “cloned” object
  - The “clone” can then start differing from the original….

Marshaling: Java Remote Objects

- Remote objects are passed by reference when used as parameter
  - C.factorize(V, F)
  - C is a ComputeEngine object; V is a Vector of integers, F is output file
  - C and F are remote objects, V is a local object



Marshaling: Java Remote Objects

- Remote objects are always “interfaces”
- The remote object may originates from the client or some other server
  - Enables complex interactions (such as callbacks) – how?

Result Marshaling/Unmarshaling

- Exactly the same as parameter marshaling / unmarshaling

Naming in Java RMI

- Obtain a reference of a remote object for the first time
  - Server machine runs rmiregistry (a separate process from the Java process)

    rmiregistry [port]
  - Java prog on the server invokes

    Naming.rebind(“objname”, remote_obj);
  - rmiregistry will remember this name binding
  - Java prog on the client invokes

    obj = Naming.lookup(“//ip_addr:port/objname”);
  - rmiregistry will return the remote obj

- Lookup itself is like an RMI

Downloading of Classes

- Two scenario:
  - Receive a marshaled object, but don’t have the code (i.e., class file) for that object
  - Receive a reference of a remote object, but don’t have the code (i.e., class file) for the object

- Java will automatically “download” the class file from the machine where you get the object
  - Advantage: Allow automatic object upgrade
  - Disadvantage: If you already have a class file for that object, the local class file will be used – possibility of inconsistency

When we should NOT use RMI/RPC

- RPC aims for client/server communication:
  - Not suitable for multicast/broadcast
- Performance: Why RPC is not used for web browsing?
  - Sub-question: What exactly are we using for web browsing?
  - All about tradeoffs
- Blocking: A client cannot invoke RPCs in parallel
  - Non-blocking RPC solves the problem, but at the cost of design complexity – sometimes beats the exact purpose of RPC!

When we should NOT use RMI/RPC

- Failures: Perhaps the biggest vulnerability
  - Concept of RPC comes from local procedure call and intends to preserve its behavior
  - What if the server fails during execution?
  - Client has no idea whether the procedure has been invoked or not …example?
  - To ensure exactly-once semantics, careful /complex logging needs to be done

- Should we hide such failure or expose it?
  - Again, it is about tradeoffs

How to make up your mind?

Rule of thumb: Use RPC if the communication pattern is simple, if you do not expect/worry much about failures, and if performance overheads are acceptable.

History Readings (Non-compulsory)

- The original paper [Birrell and Nelson’84]:
  - “Implementing Remote Procedure Calls”, in ACM Transactions on Computer Systems, Feb 1984.

- SUN RPC:
  - http://www.rfc-editor.org/rfc/rfc1057.txt
- XML-RPC:
  - http://www.xmlrpc.com/

Motivation: Multicast

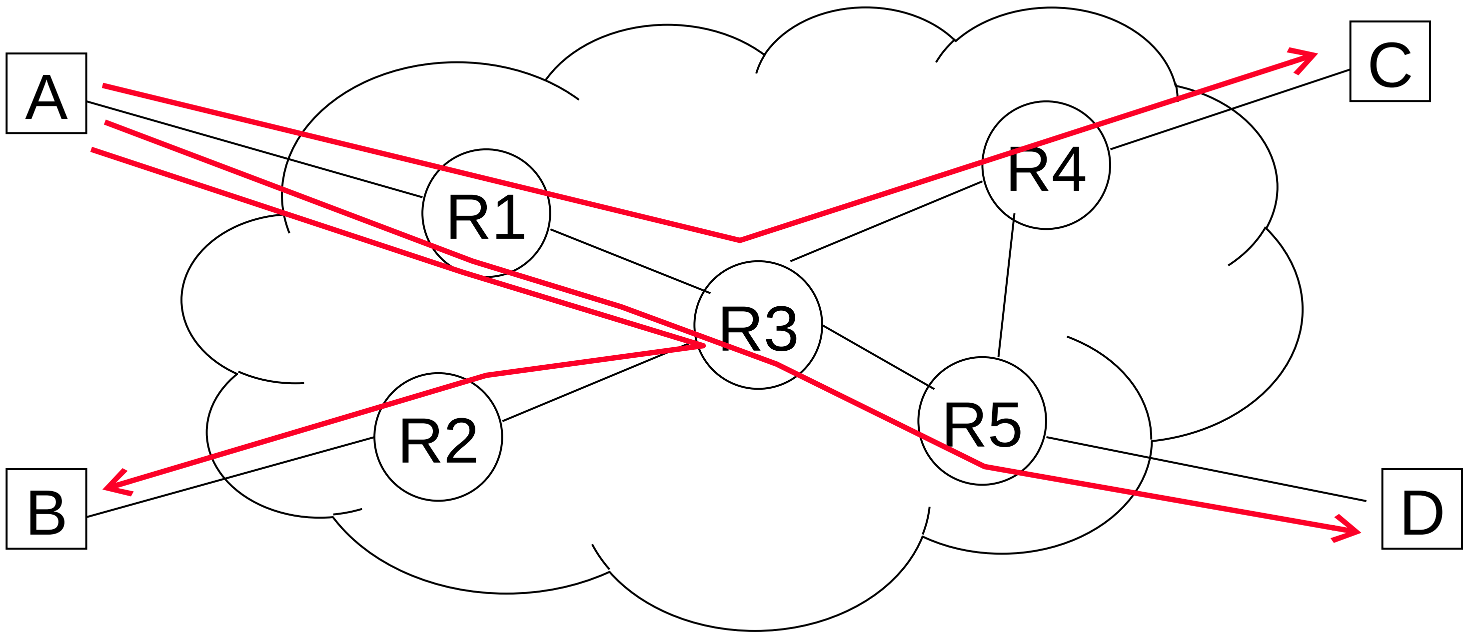
- Single sender sends the same data to n receivers
- Use n point-to-point messages
  - Overload the sender – throughput limited by

    server's bandwidth / n

- Multicast: Have the clients help forward the message to others

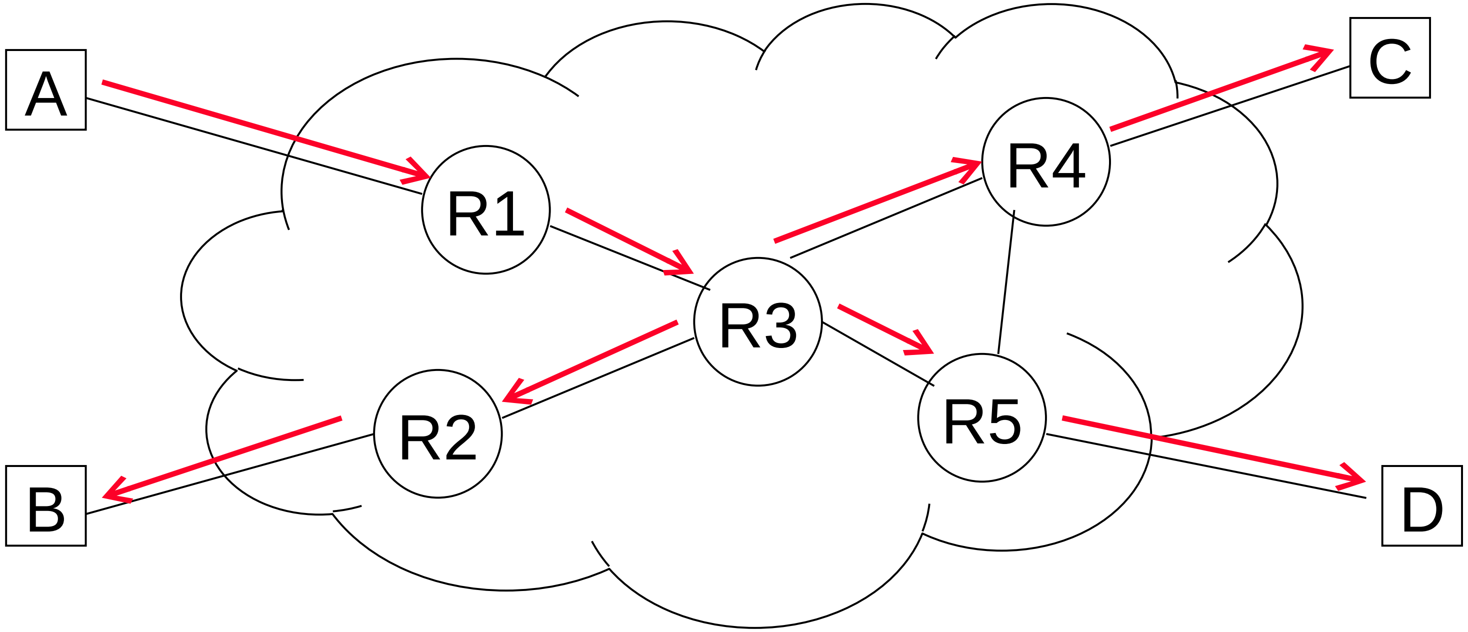
Without Multicast

- A uses its outgoing link 3 times
- Edge R1 - R3 is used 3 times as well



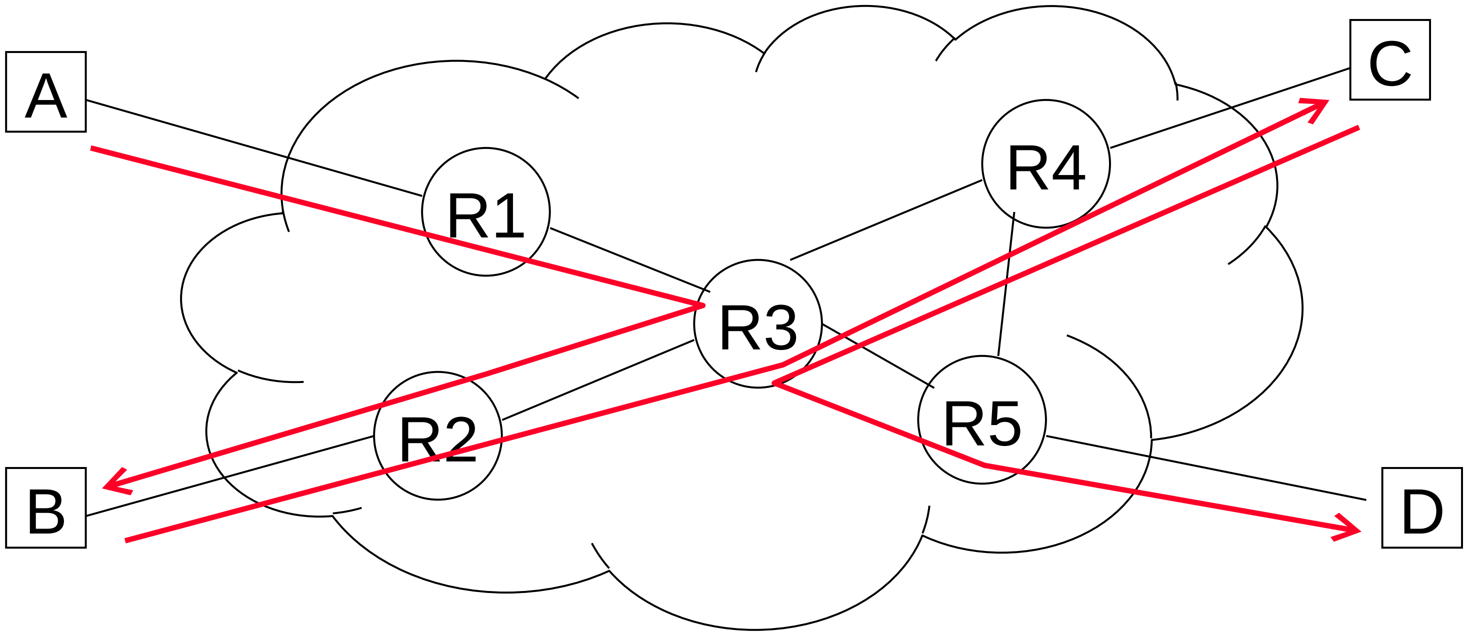
IP-Multicast

- A networking topic
- Very efficient – message never traverse same link twice
- Need support for routers – ISPs hate changes
- IP multicast was never hugely successful – a good technique does not always turn into products



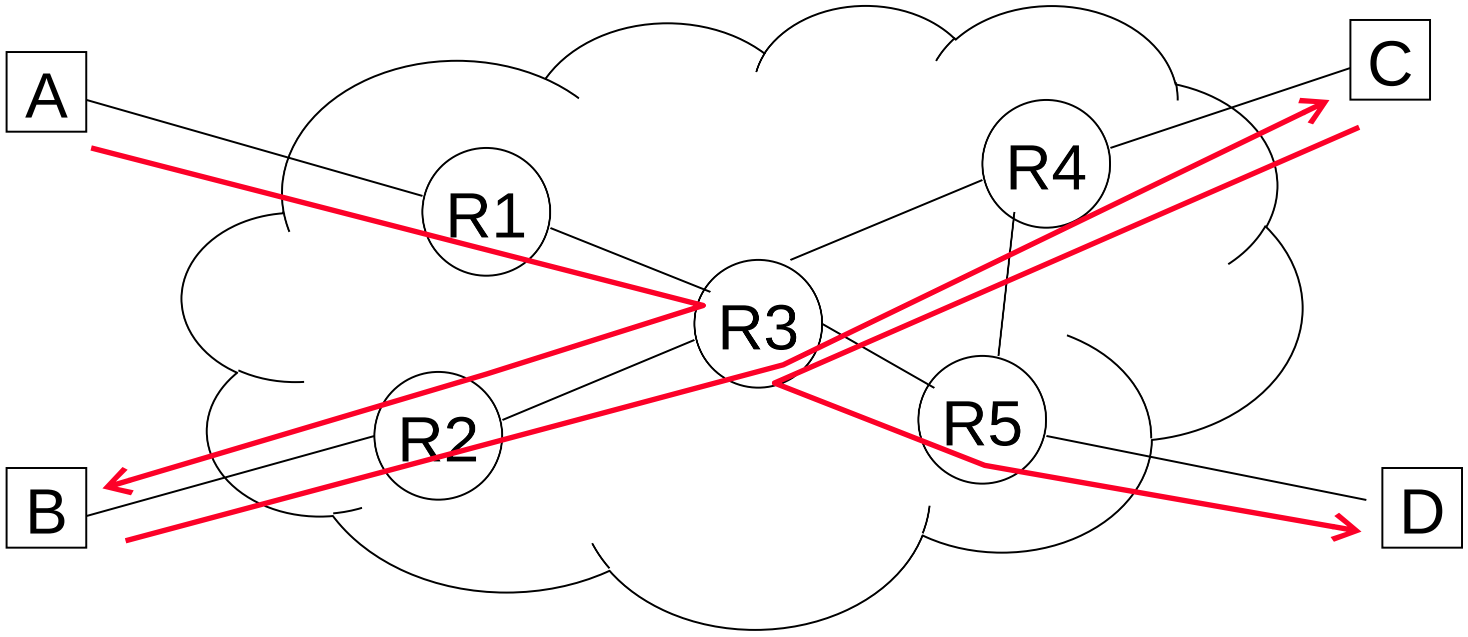
Application-Level Multicast

- Example: Server sends data to a small number of client, each client forwards the data to some other clients
- A distributed system topic



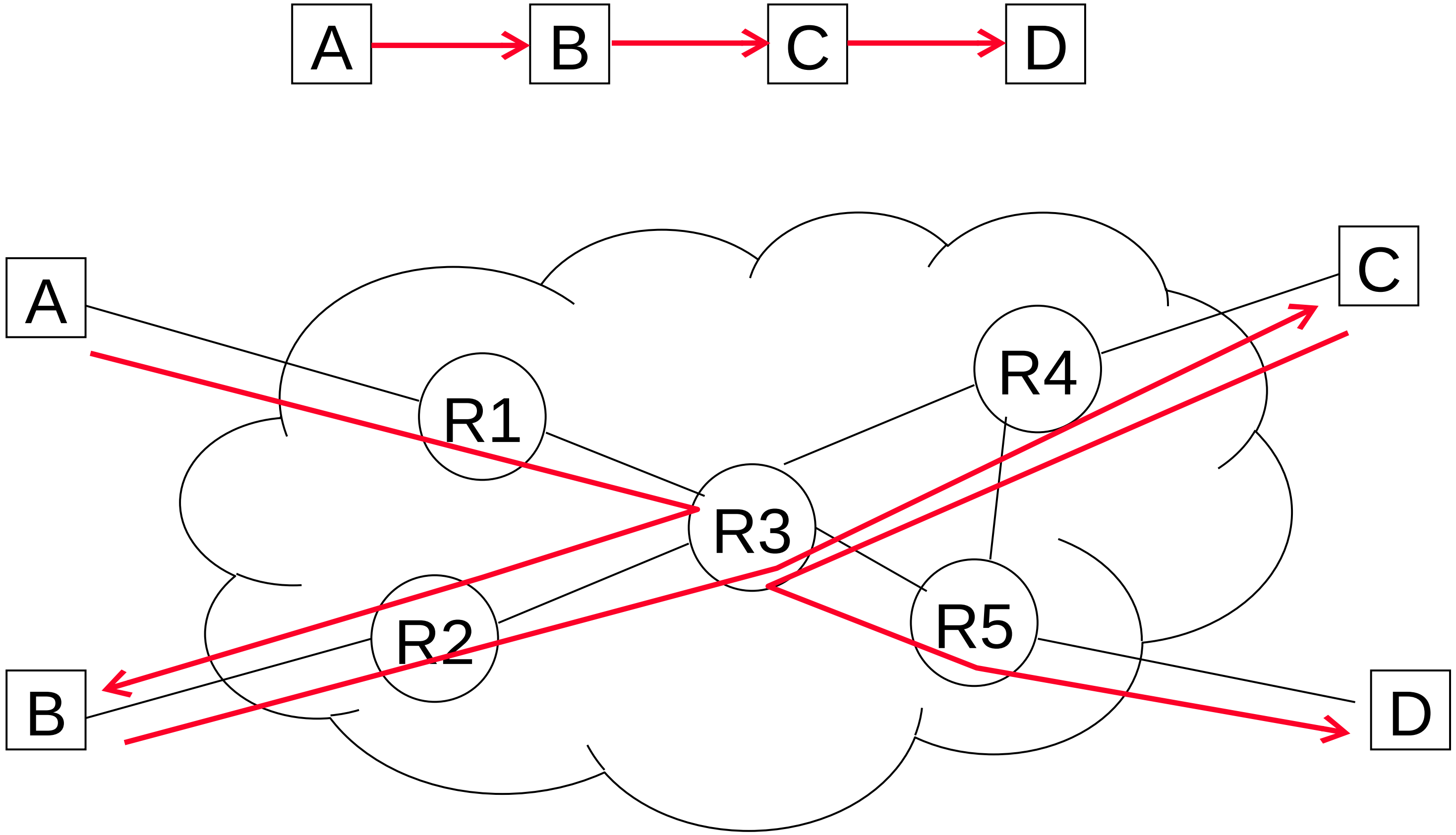
Application-Level Multicast

- Advantage:
  - Partly solves the problem with server bandwidth
  - No need to change routers
- Disadvantage:
  - A message may still traverse the same link twice
- Overall: Has been much more successful than IP multicast



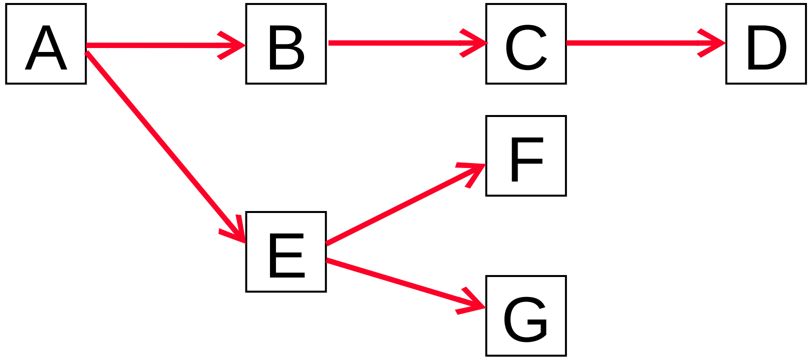
Designing Application-Level Multicast

- Multicast topology

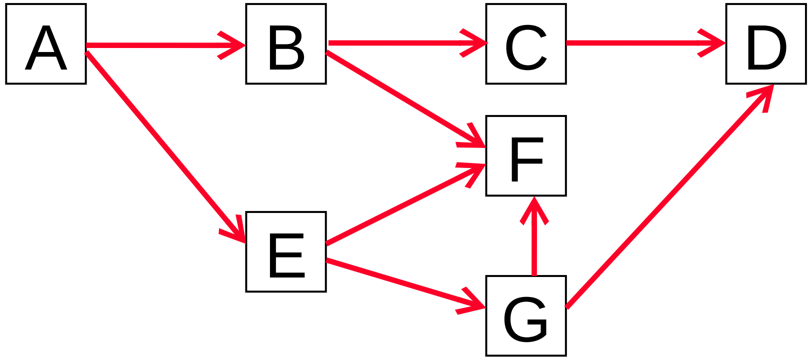


Multicast Topology

- Tree: Simple but not robust

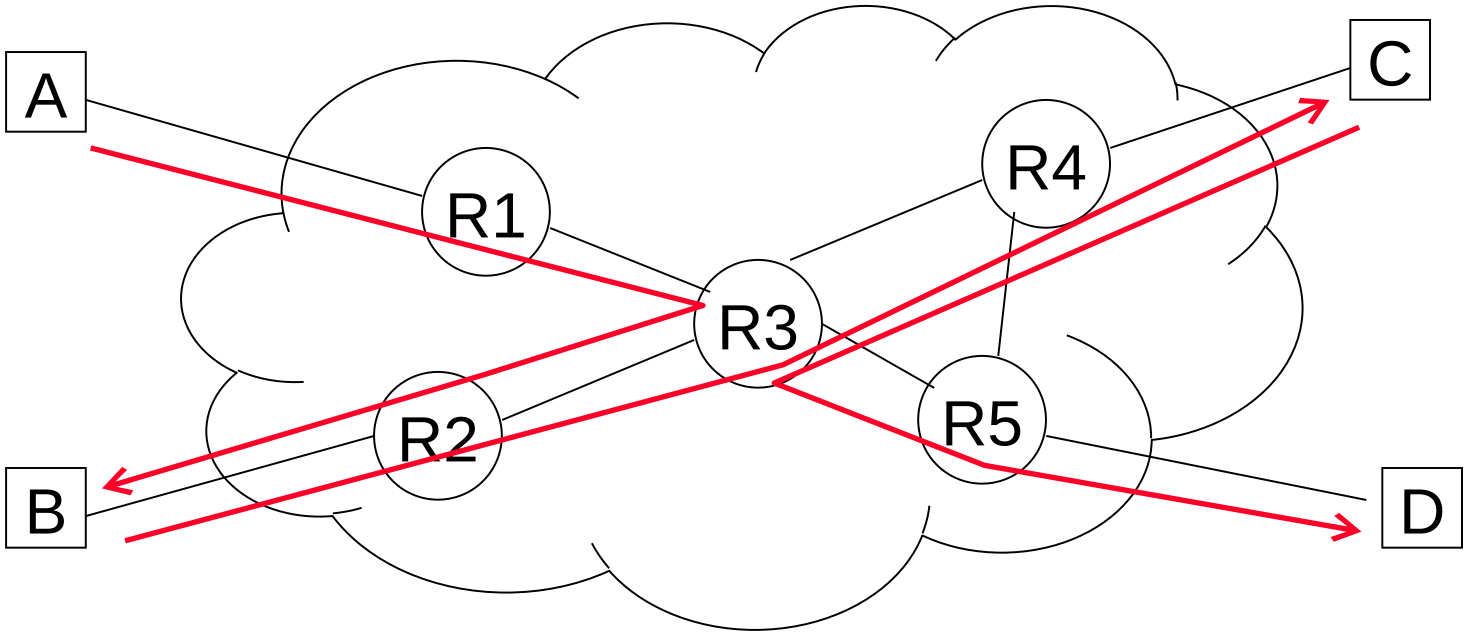


- Mesh: Robust but more complex – what data should I get from whom?



Designing Application-Level Multicast

- Quality of a multicast tree
  - Link stress (2): maximum # messages traversing the same edge
  - Stretch (12/4=3): Relative delay penalty given a pair of nodes
  - Tree cost (12): Total weight of edges – minimum spanning tree optimizes for tree cost



Motivation: Gossip

- Sender wants to send the same data to multiple receivers
  - Each node periodically send the data (if it has the data) to some other random node – usually called a round
- Advantage of gossip:
  - Reduces load of the sender
  - Extremely robust
  - Easy to verify from our everyday experience
  - Simple – no need to maintain a multicast topology
- Disadvantage – not as efficient as multicast

Designing a Gossip Protocol

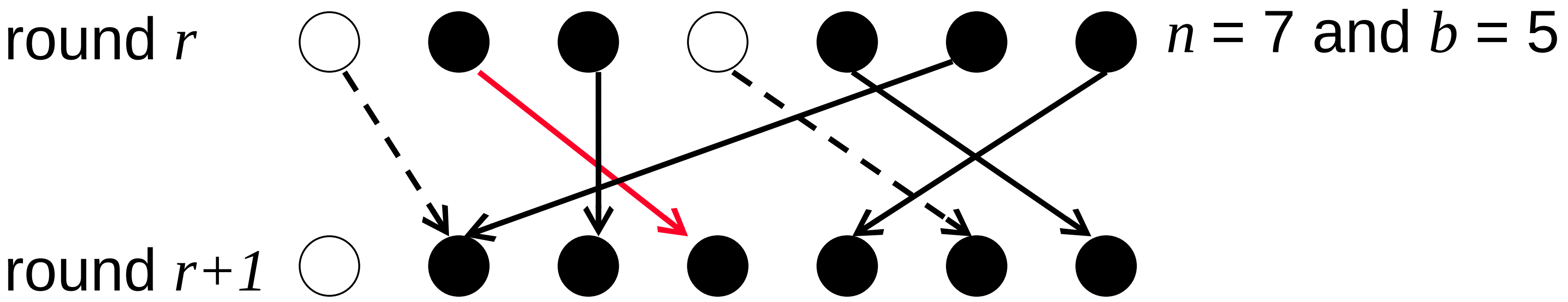
- Each gossip operation is between two nodes
- How to pick the other node to gossip with?
  - Usually picked uniformly at random
  - How to pick random nodes without global knowledge?
  - Performance will be influenced by the choice

- How to gossip? If P picks Q
  - P may gossip news to Q (push)
  - P may request news from Q (pull)
  - P and Q may update each other (push+pull)

Performance of Push-based Gossiping

- P pushes message to Q
  - Effective at the beginning, but not toward the end
  - Chance of a black P picking a white Q decreases with time
  - n: # total nodes; b: # black nodes;
  - Prob[a black P picks a white Q] = $(n-b)/n$ = 2/7
  - But the number of black P's increases as well !

    (Description on textbook is not complete)



Push: Be More Rigorous

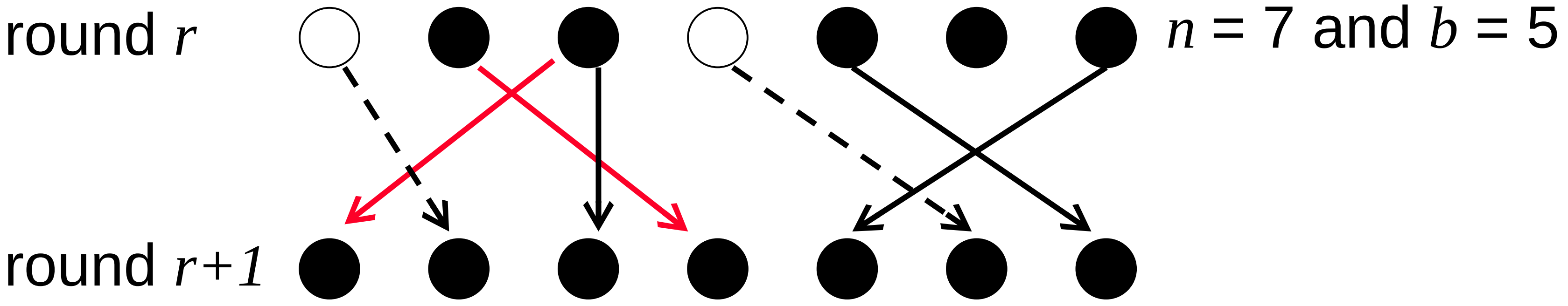
- Consider a white node Q at round r
  - Prob[Q remains white at round r+1] is $\left(1-\frac{1}{n}\right)^b$
- Changing the last remaining white node to black (i.e., from $b = n\text{-}1$ to $b = n$)

$$1-\left(1-\frac{1}{n}\right)^{n-1} \approx 1-\left(1-\frac{1}{n}\right)^{n} \approx 1-\frac{1}{e} \approx 0.63$$

  - On expectation 1/0.63 = 1.6 rounds needed
- Changing the first remaining white node to black (i.e., from $b = 1$ to $b = 2$)
  - On expectation $1/\left(1-\frac{1}{n}\right) \approx 1$ round needed

Performance of Pull-based Gossiping

- P pulles message to Q
  - Not effective at the beginning, but effective toward the end
  - Chance of a white P picking a black Q increase with time
  - n: # total nodes; b: # black nodes;
  - Prob[a white P picks a black Q] = b/n = 5/7



Pull: Be More Rigorous

- Changing the first remaining white node to black (i.e., from $b = 1$ to $b = 2$)
  - On expectation 1.6 rounds needed – why?

- Changing the last remaining white node to black (i.e., from $b = n\text{-}1$ to $\mathrm{b} = \mathrm{n}$)
  - On expectation $1/\left(1-\frac{1}{n}\right) \approx 1$ round needed

- One would naturally imagine that we want to use push+pull

Asymptotic Properties

- The asymptotic number of rounds needed to change all nodes to black is the same: $\theta(\log n)$
  - For push-only
  - For pull-only
  - For push+pull

- To prove this, consider two stages:
  - First stage: From $b = 1$ to $b = 0.1n$
  - Second stage: From $b = 0.1n$ to $b = n$

Second Stage

- At each round, imagine that the black nodes each throws a ball into a uniformly random bin (this is for push – pull is slightly different)
  - Total $0.1n$ balls thrown into n bins
  - The number of balls thrown at each round never below $0.1n$

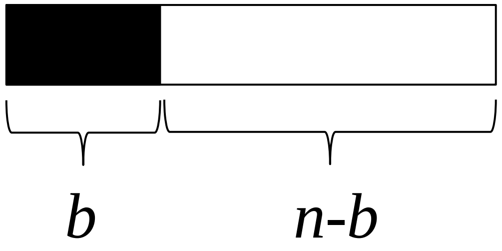
- Coupon collection problem tells us:
  - On expectation needs $\theta(n\log n)$ balls

- Thus on expectation, # rounds needed is at most

$$\frac{\theta(n\log n)}{0.1n} = \theta(\log n)$$

First Stage

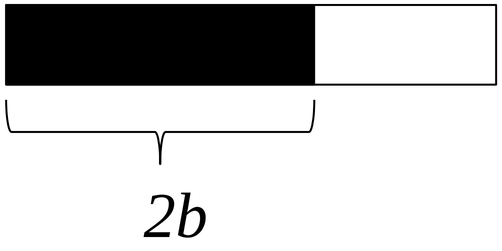
- Define X_1 to be # rounds needed from b = 1 to b = 2
- Define X_2 to be # rounds needed from b = 2 to b = 4
- …
- Can prove that E[X_i] < 1.3 for all i
  - Thus the first stage takes $\theta(\log n)$ rounds on expectation



Need b successes. Each ball has a success probability of at least (n-b)/n >= 0.8.

Need on expectation b/0.8 < 1.3b balls. Each round has at least b balls.

Need on expectation < 1.3 rounds.



History Readings (Non-compulsory)

- The original paper on gossiping:
  - “Epidemic Algorithms for Replicated Database Maintenance” in PODC’87
  - http://portal.acm.org/citation.cfm?id=41841

More Applications of Gossiping

- Gossiping not only good for propagating messages
- Another interesting application: Information aggregation

- Count the total number of nodes in the system
  - A distinguished node 0 starts from a value $x_0 = 1$
  - Others have $x_i = 0$

- Each node i periodically picks another node j
  - Node i: new $x_i = (x_i + x_j)/2$
  - Node j: new $x_j = (x_i + x_j)/2$

Counting the Number of Nodes

- Eventually all nodes will have the same x_i values
- Property from how we update x_i : Mass preserving
  - The sum of all x_i's are always 1
- Property from gossiping:
  - All x_i 's will eventually be the same
  - Takes logarithmic number of rounds to achieve this (a much harder proof!)

- Number of nodes = $\dfrac{1}{x_i}$

Computing Average

- Each node has a value (e.g., available disk space)
  - We want to know the total available disk space
- Same as before but set x_i to be the amount of disk space available
- What if we want to compute sum? PollEv.com/haifengyu229
- Drawback of this approach: Node leaves and node failures
  - Disrupts the mass preserving property – any way to patch?

Another Way of Computing Count

- Each node picks a random number between [0,1]
  - Use gossiping to find out the minimum
  - (The textbook uses maximum, which is the same – but conventionally people use minimum)
  - The larger the number, the smaller the minimum

- How to do sum and average? PollEv.com/haifengyu229

- Advantage: More robust against failures
- Other in-depth differences with the previous approach exist (e.g., specific forms of the asymptotic properties)

History Readings (Non-compulsory)

- The beautiful push-sum algorithm
  - "Gossip-based Aggregation in Large Dynamic Network"
  - http://portal.acm.org/citation.cfm?id=1082470

Summary

- Review (Overview) of the OSI model
- Remote Procedure Call / Remote Method Invocation
  - If you don't know Java, read some tutorials on Java
- Multicast
- Gossiping

Mandatory Homework For This Week

- Try out RMI by following tutorial at
  - Canvas\Files\AssignmentOne\RMI-tutorial.txt
  - You should be able to smoothly run and completely understand this example, otherwise you will have trouble with Assignment One
  - Email me or the TA (Sun Yucheng, sunyuch@comp.nus.edu.sg) if you need help on this homework --- everyone should complete this homework
- Resources:
  - Canvas\Discussions -- You are encouraged to discussed with other students about this homework
- Security warning – kill your processes after running:
  - Do not leave rmiregistry running forever
  - Do not leave your server code running forever