



# Communication: Message Queues & BASE, Streams, Multicast End-to-End Argument

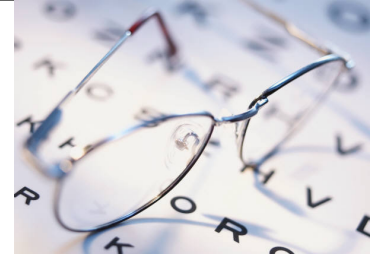
PCSD, Marcos Vaz Salles

# Do-it-yourself-recap: Achieving the ACID Properties

- **ACID Properties**
  - **Atomicity**: transactions are all-or-nothing
  - **Consistency**: transaction takes database from one consistent state to another
  - **Isolation**: transaction executes as if it were the only one in the system (aka before-or-after atomicity)
  - **Durability**: once transaction is done (“committed”), results are persistent in the database
- **Questions**
  - How did we achieve these properties? Which techniques were buying us which of the ACID properties?
  - What was the system model? (e.g., interpreter/memory/communication abstraction used, assumptions regarding distribution, durability)



## What should we learn today?



- Describe different approaches to design communication abstractions, e.g., transient vs. persistent and synchronous vs. asynchronous
- Explain the design and implementation of message-oriented middleware (MOM)
- Explain how to organize systems employing a BASE methodology
- Discuss the relationship of BASE to eventual consistency and the CAP theorem
- Discuss alternative communication abstractions such as data streams and multicast / gossip
- Apply the end-to-end argument to system design situations

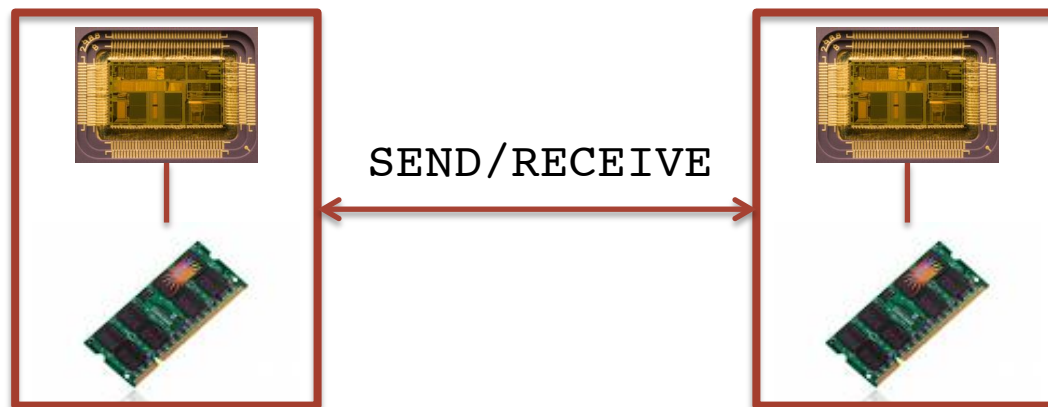
## Availability vs. Atomicity

- Last two weeks' property: **Atomicity**
- Next two weeks: **Availability**



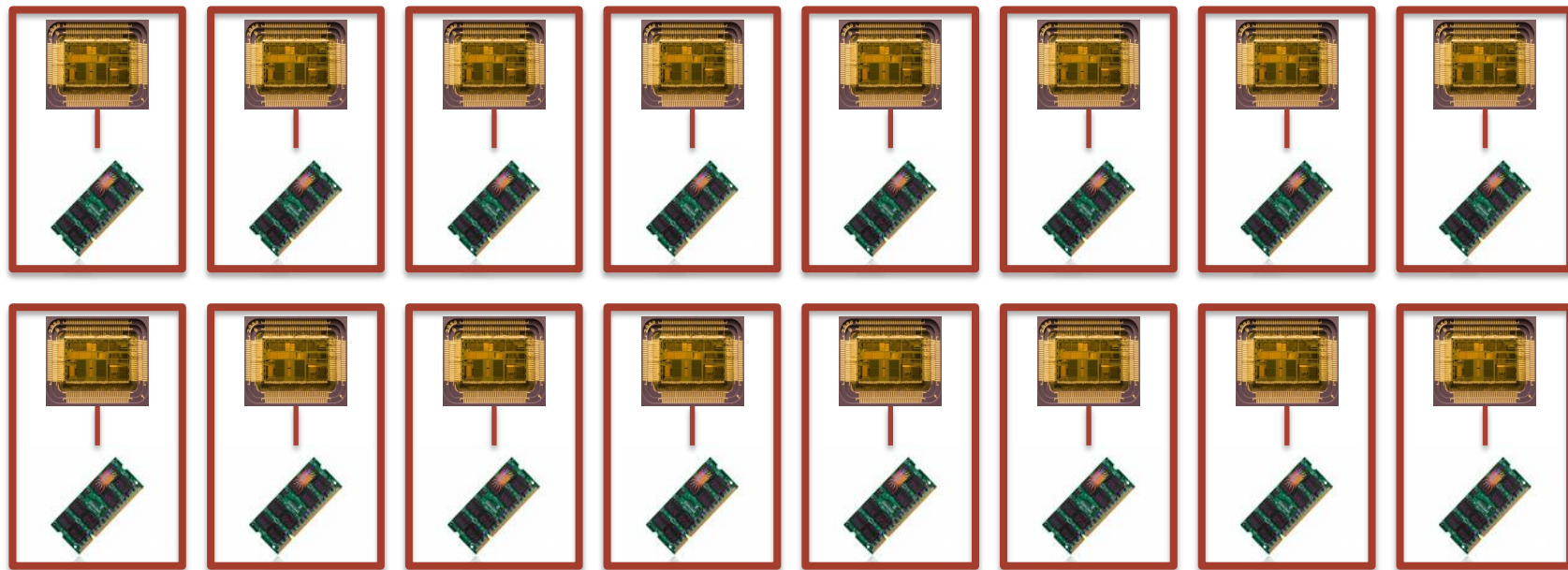
## Towards Distribution

- Use independent services to store different data → **partitioning**
- Availability improves but now coordination may be necessary
- Employ communication abstraction



## Towards Distribution

- Use independent services to store different data → **partitioning**
- Availability improves but now coordination may be necessary
- Employ communication abstraction



## Types of Communication Abstractions

- **Message-Oriented**
  - SEND/RECEIVE of point-to-point messages
  - Synchronous vs. Asynchronous
  - Transient vs. Persistent
- **Stream-Oriented**
  - Continuous vs. discrete
  - Asynchronous vs. Synchronous vs. Isochronous
  - Simple vs. complex
- **Multicast**
  - SEND/RECEIVE over groups
  - Application-level multicast vs. gossip

What about  
Remote-  
Procedure Call  
(RPC)?



Source: Saltzer & Kaashoek & Morris (partial),  
Tanenbaum & Steen (partial)



Source: Tanenbaum &  
Van Steen

## Message-Oriented Communication

- Middleware for different types of communication
  - Synchronous vs. Asynchronous
    - Synchronize at request submission (1)
    - Synchronize at request delivery (2)
    - Synchronize after processing by server (3)
  - Transient vs. Persistent

### Examples?

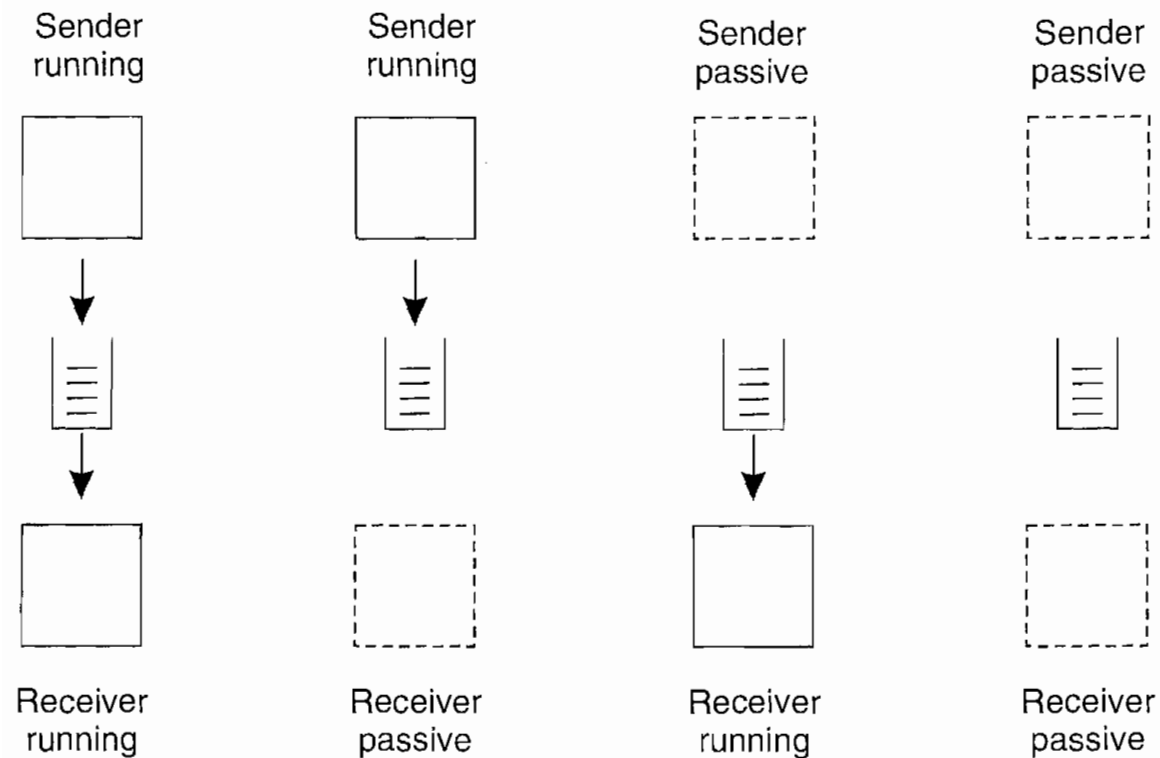
	Persistent	Transient
Asynchronous	E-mail	Erlang
Synchronous	Google chat (1)+(2)	RPC (3)





## Message-Oriented Persistent Communication

- Queues make sender and receiver loosely-coupled
- Modes of execution of sender/receiver:

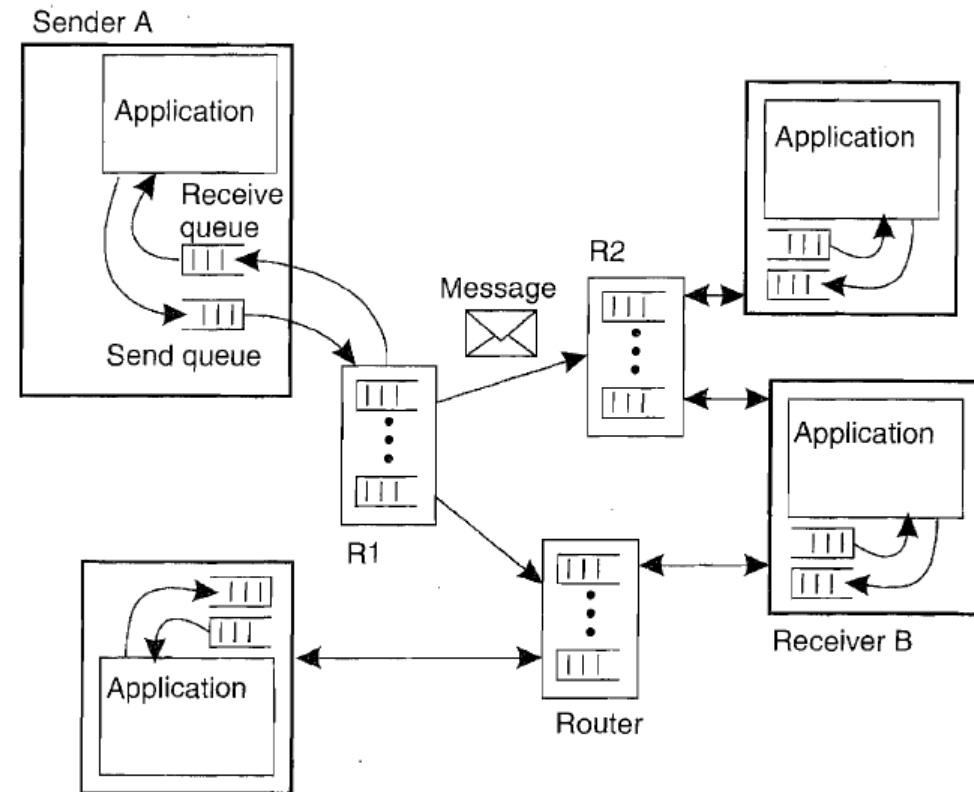


Source: Tanenbaum & Van Steen



## Queue Interface

<b>Put</b>	Put message in queue
<b>Get</b>	Remove first message from queue (blocking)
<b>Poll</b>	Check for message and remove first (non-blocking)
<b>Notify</b>	Handler that is called when message is added



- Source / destination decoupled by **queue names**
- **Relays:** store and forward messages
- **Brokers:** gateway to transform message formats



Source: Tanenbaum & Van Steen

## Employing Queues to Decouple Systems

### Transaction T1: TRANSFER

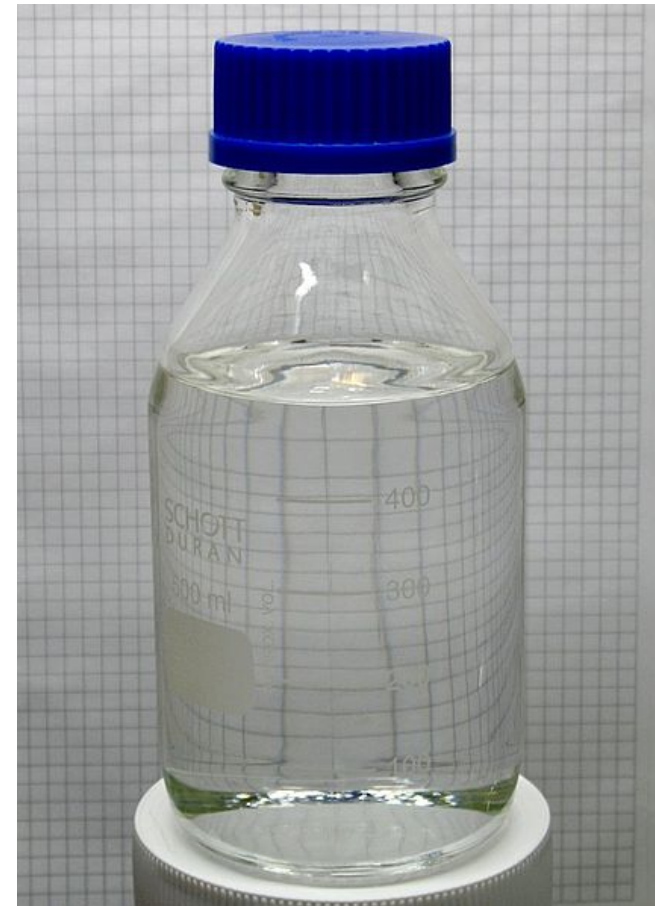
```
BEGIN
  UPDATE account
  SET bal = bal + 100
  WHERE account_id = 'A';
  --
  UPDATE account
  SET bal = bal - 100
  WHERE account_id = 'B';
COMMIT
```

- What if we partition accounts A and B across different computers?
- How would you execute TRANSFER using Message-Oriented Middleware (MOM)?
- What guarantees can you still give? What happens to atomicity?



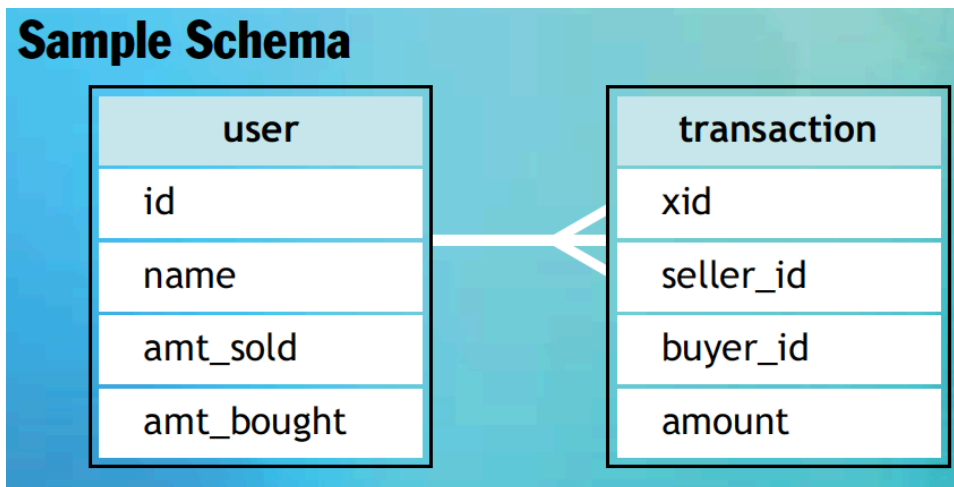
## From ACID to BASE

- **BASE**
  - **Basically-Available**: only partitions affected by failure become unavailable, not whole system
  - **Soft-State**: partition state may be out-of-date, and events may be lost without affecting availability
  - **Eventually Consistent**: under no further updates and no failures, partitions converge to consistent state



## A BASE Scenario

- Users buy and sell items
- Simple transaction for item exchange:



Begin transaction

Insert into transaction(xid, seller\_id, buyer\_id, amount);

Update user set amt\_sold=amt\_sold+\$amount where id=\$seller\_id;

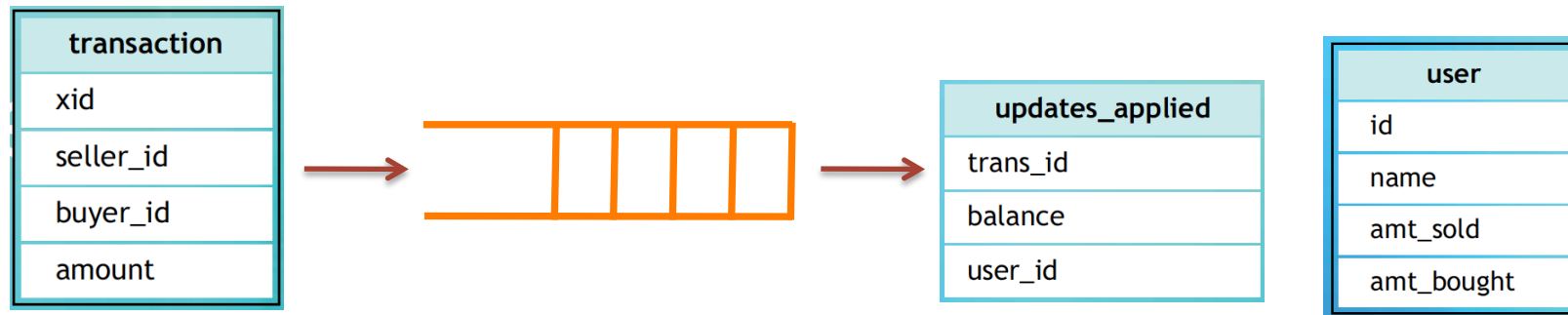
Update user set amt\_bought=amount\_bought+\$amount where id=\$buyer\_id;

End transaction

Source: Pritchett (partial)



## Decouple Item Exchange with Queues



Record Transaction  
Queue User updates

Peek Queue for message  
Check if update applied  
If not  
    Update User amounts  
    Record update as applied  
Remove message from queue

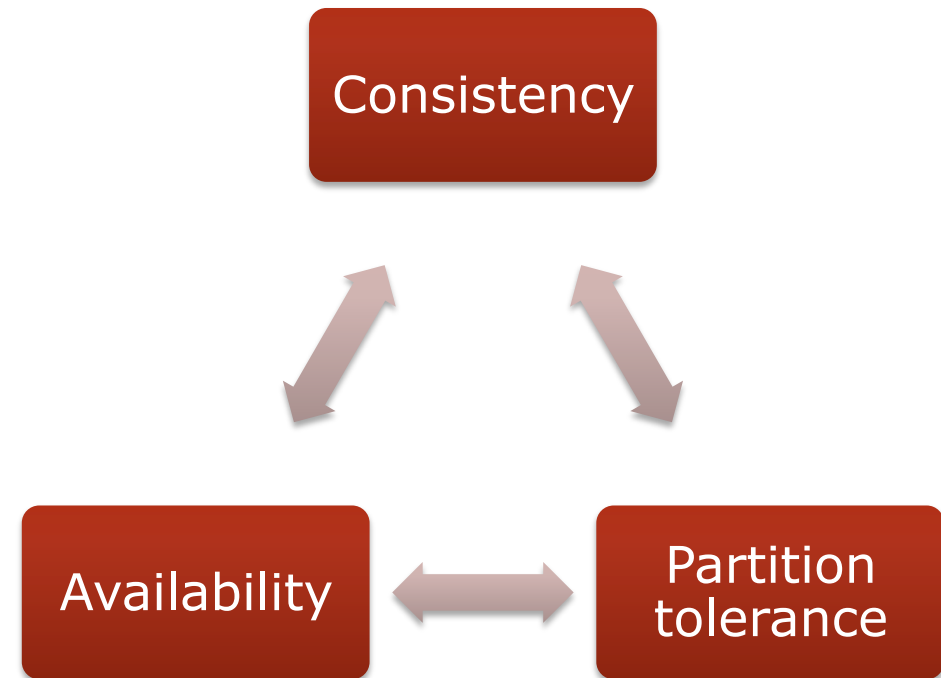
- **Issues**
  - Tolerance to loss
  - Idempotence
  - Order



Source: Pritchett (partial)

## Partitioned State and the CAP Theorem

- **Consistency**: client perceives set of operations occurred all at once (i.e., atomicity)
- **Availability**: every operation terminates in intended response
- **Partition tolerance**: operations complete, even if components are unavailable



**CAP Theorem:**  
A scalable service cannot achieve all three properties simultaneously



Source: Pritchett (partial)

## Types of Communication Abstractions

- **Message-Oriented**
  - SEND/RECEIVE of point-to-point messages
  - Synchronous vs. Asynchronous
  - Transient vs. Persistent
- **Stream-Oriented**
  - Continuous vs. discrete
  - Asynchronous vs. Synchronous vs. Isochronous
  - Simple vs. complex
- **Multicast**
  - SEND/RECEIVE over groups
  - Application-level multicast vs. gossip

More details in  
compendium



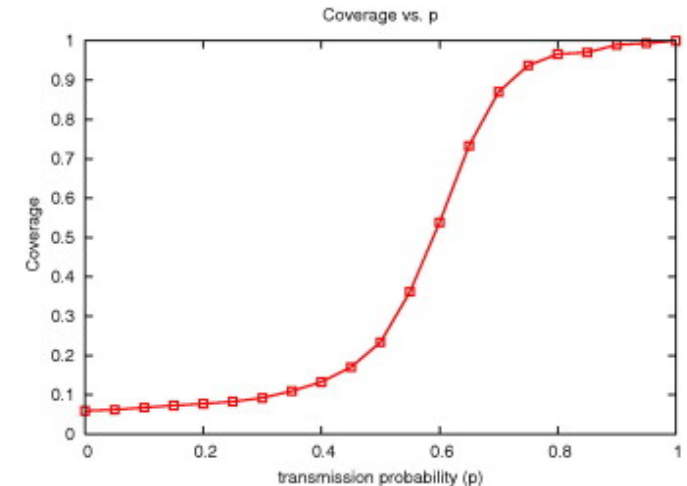
Source: Saltzer & Kaashoek & Morris (partial),  
Tanenbaum & Steen (partial)



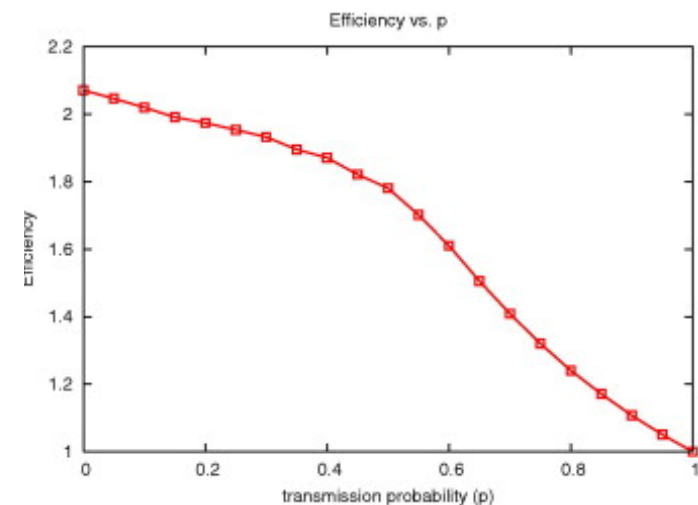


## A Word about Gossip

- **Epidemic protocols**
  - No central coordinator
- **Gossiping**
  - If P is updated it tells a random Q
  - If Q already knows, P can lose interest with some percentage
  - Not guaranteed that all nodes get infected by update
- **Anti-entropy**
  - Each node communicates with random node
  - Round: every node does the above
  - Pull vs. push vs. both
  - Spreading update from single to all nodes takes  $O(\log(N))$



Coverage vs. trans prob



Efficiency vs. trans prob



Source: Haas et al. (partial)

Questions so far?



## Recall: Fundamental abstractions

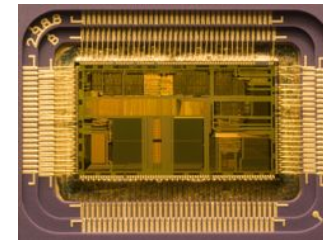
- Memory
  - Read/write



- Interpreter
  - Instruction repertoire
  - Environment
  - Instruction pointer

`(loop (print (eval (read))))`

Networks  
implement link  
abstraction



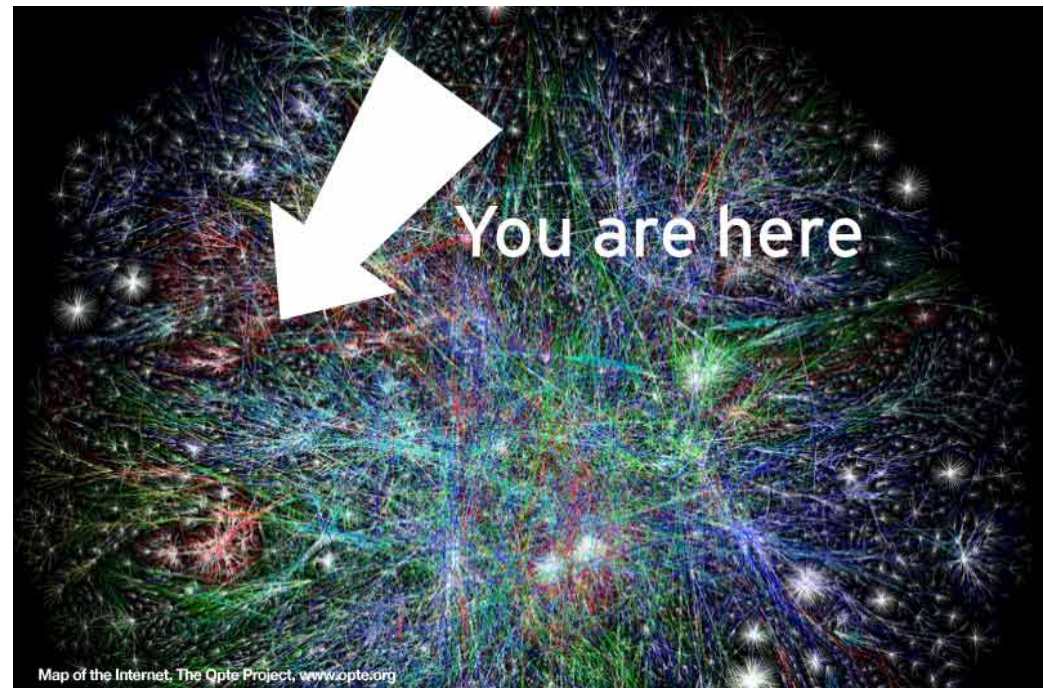
- Communication links
  - Send/receive



Source: Saltzer & Kaashoek & Morris (partial)

## Recall: Computer Networks

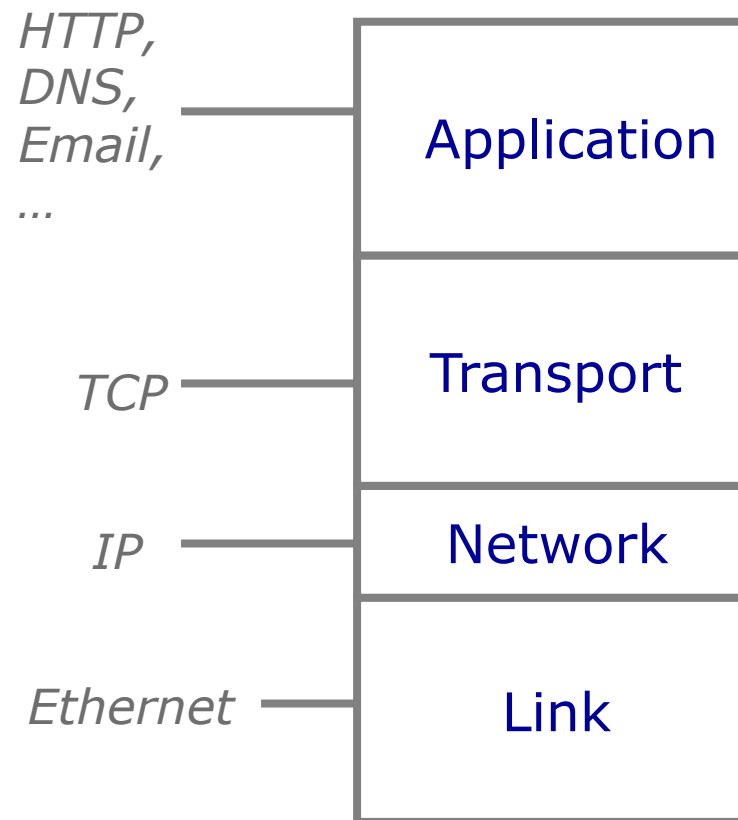
- Naming service to find services
- Clients anywhere can communicate to services anywhere else
- Layers of **protocols** deal with details of data transmission and multiplexing of links



Protocols vs. APIs?

## Recall: Protocols and Layering in the Internet

- Layering
  - System broken into vertical hierarchy of protocols
  - Service provided by one layer based solely on service provided by layer below
- Internet model
  - Four layers

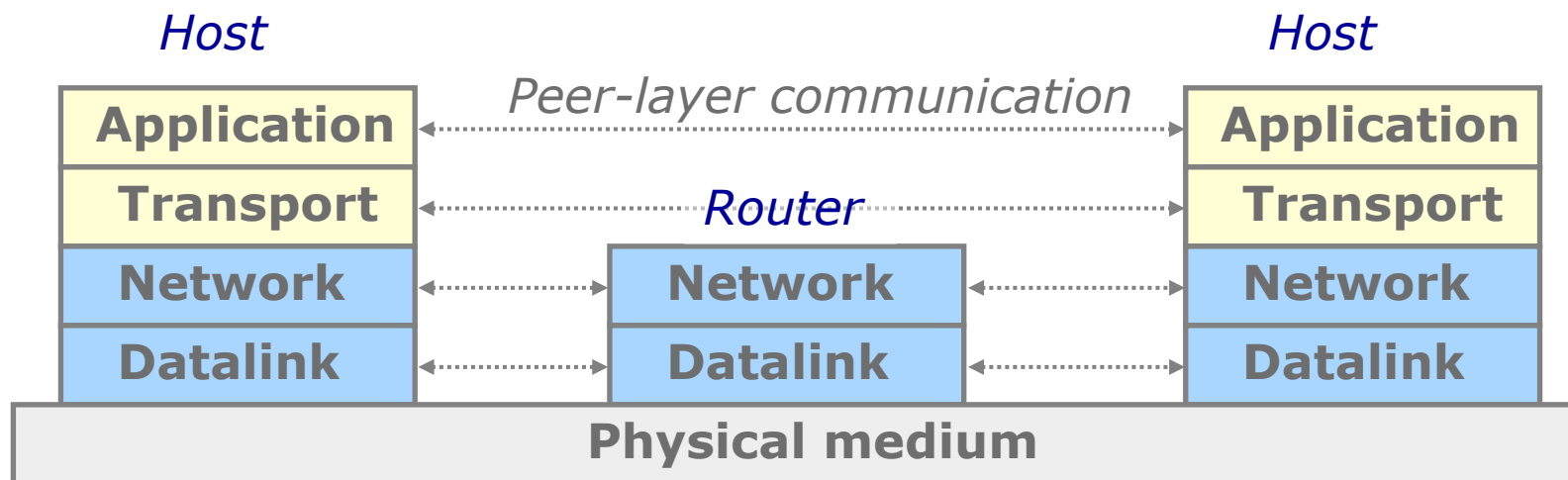


Do all servers in the network contain all layers?



## Recall: Layers in Hosts and Routers

- Link and network layers implemented everywhere
- End-to-end layer (i.e., transport and application) implemented only at hosts



Today's question: Why?



Source: Katabi & Kaashoek & Morris & others (partial)

## End-to-end argument

“The function in question can completely and correctly be implemented only **with the knowledge and help of the application** standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)”

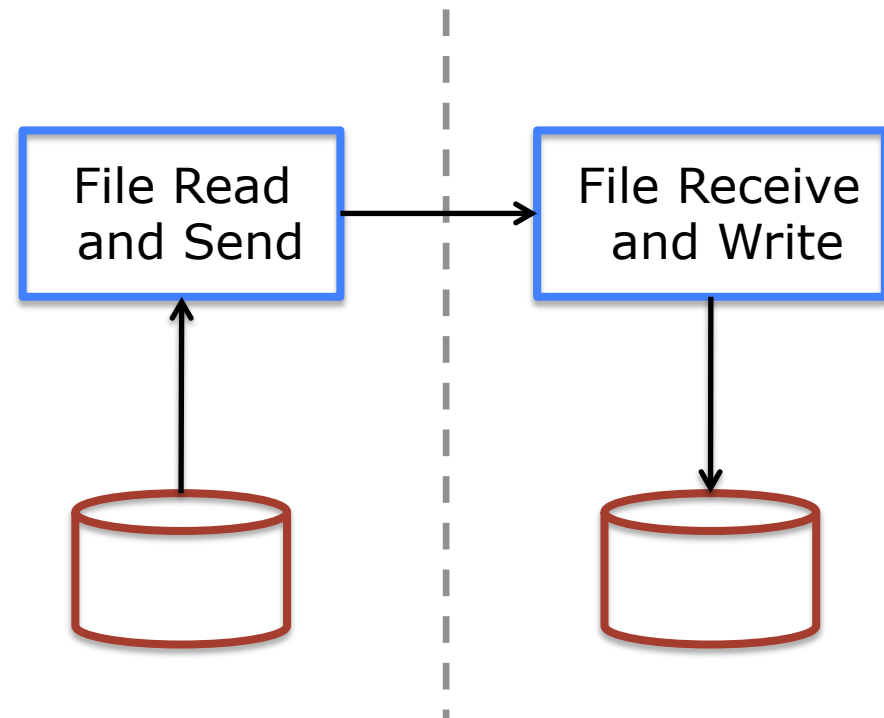
Saltzer, Reed, and Clark

i.e., the application knows best! 😊



## Careful File Transfer Application

- Read from disk into memory
- Organize packets, pass to communication system
- Receive packets, re-organize file in memory
- Write file to disk

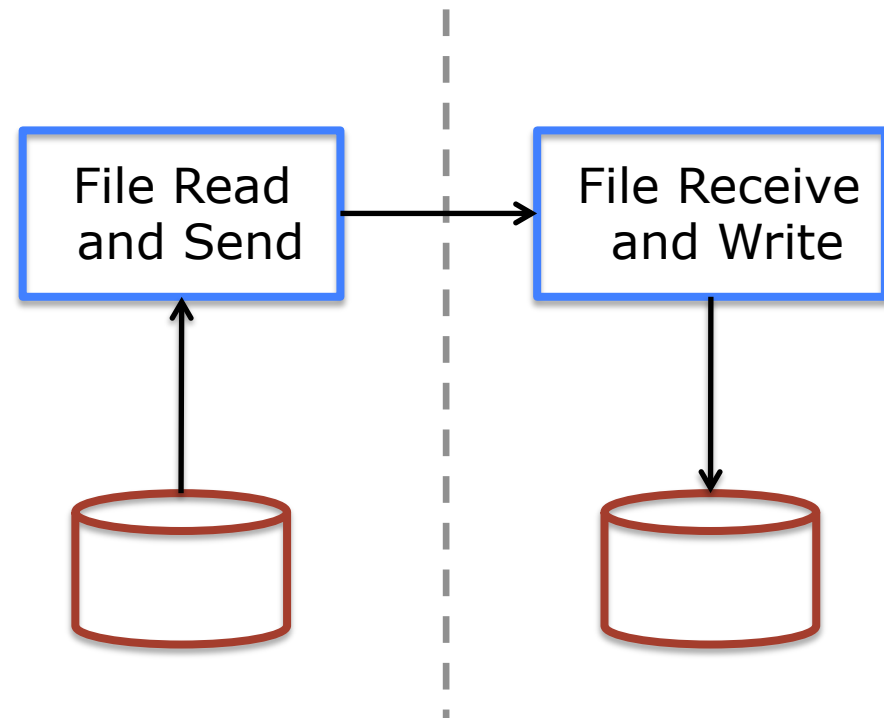


What can go wrong?  
How can you work around it?



## End-to-End Check and Retry

- Application tests if end-to-end function worked; if not, retries the whole function
- Careful file transfer uses a checksum
  - If checksum wrong, re-transfer the file
- Reliable data transmission not enough to eliminate all threats!



## Network-Level Checks as Performance Enhancements

- *Inevitability of application-level check means communication system does not have to do anything then?*
- Multiple re-transfers can cost a lot
  - Especially when almost the whole file went through!
- Argument: Communication system should offer “some” reliability to reduce *frequency* of retransmissions, but not “perfect” reliability
  - E.g., error detection and correction in link-layer



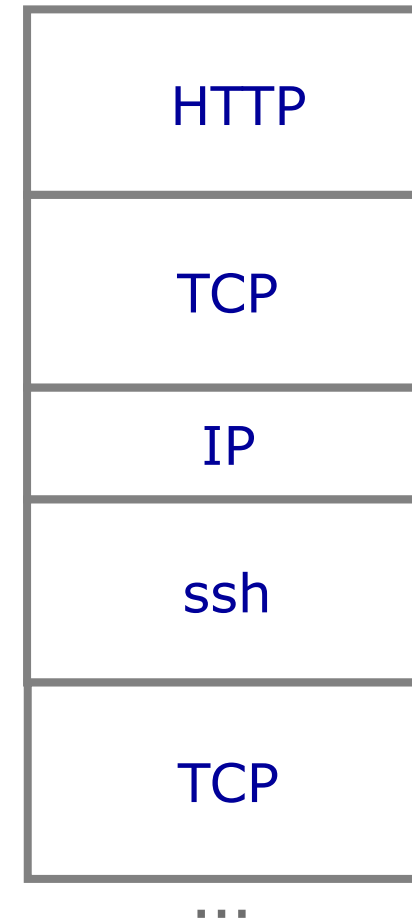
## End-to-End Transport Protocols

- UDP
  - Ports + checksums
- TCP
  - Ordering + no missing nor duplicate data
  - Flow control, congestion avoidance
- RTP
  - Turns UDP checksums off
  - Timestamps packets
- Connection to RPC semantics
  - Request/response
  - At least once delivery: timers
  - At most once delivery: duplicate suppression



## Power to the Edges: Clever Uses of Layering

- Nesting layers to the extreme: tunneling
  - Run link layer over TCP (Virtual Private Network)
- Router uses TCP as transport for routing protocol (e.g., BGP)
- P2P overlays (e.g., Skype)



## Other Examples of End-to-End Argument

- Secure transmission
- Distributed commit (e.g., 2PC)
- Exokernels
- Auditing logs and procedures
- Keeping multiple copies of data (e.g., replication and backup)

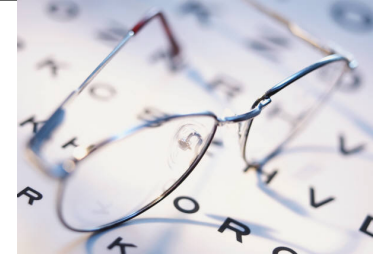


## Philosophical Summary of End-to-End Argument

- System will hardly eliminate all errors
  - So it should implement what it does really well
- Some functionality needs help from application
- High-level checks necessary, however possibly costly
- System can provide performance enhancements to **reduce error frequency**



## What should we learn today?



- Describe different approaches to design communication abstractions, e.g., transient vs. persistent and synchronous vs. asynchronous
- Explain the design and implementation of message-oriented middleware (MOM)
- Explain how to organize systems employing a BASE methodology
- Discuss the relationship of BASE to eventual consistency and the CAP theorem
- Discuss alternative communication abstractions such as data streams and multicast / gossip
- Apply the end-to-end argument to system design situations

