UNIVERSITY OF COPENHAGEN

Communication: Message Queues & BASE

ACS, Yongluan Zhou

What should we learn today?

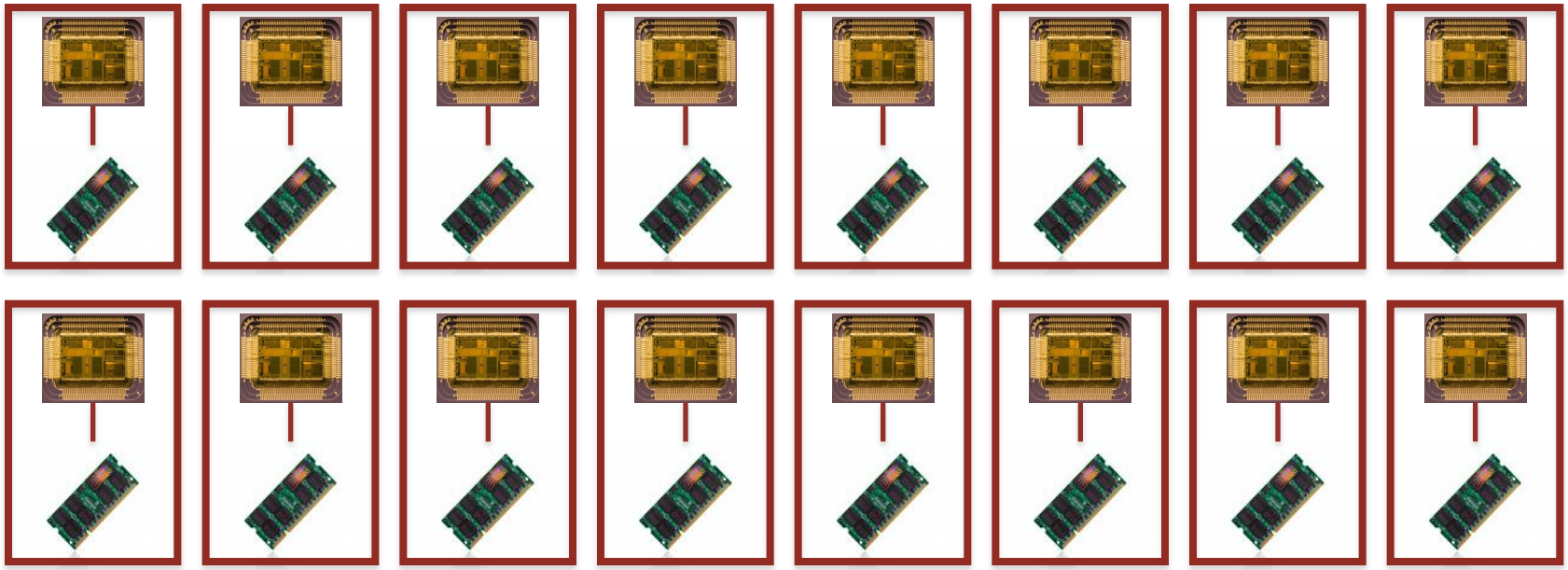
- Describe different approaches to design **communication abstractions**, e.g., transient vs. persistent and synchronous vs. asynchronous
- Explain the design and implementation of **message-oriented middleware** (MOM)
- Explain how to organize systems employing a **BASE** methodology
- Discuss the relationship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

Towards Distribution

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

Towards Distribution

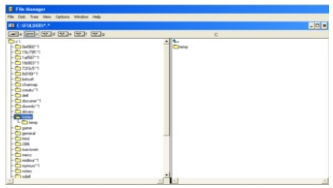
- Use independent services to store different data → **partitioning**
- Scalability and Availability improves but now coordination may be necessary
- Employ communication abstraction



Recall: Fundamental abstractions

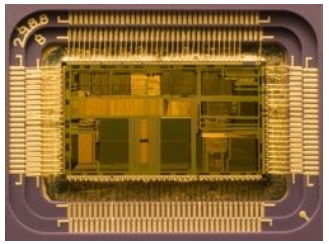
- Memory
  - Read/write





- Interpreter
  - Instruction repertoire
  - Environment
  - Instruction pointer

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



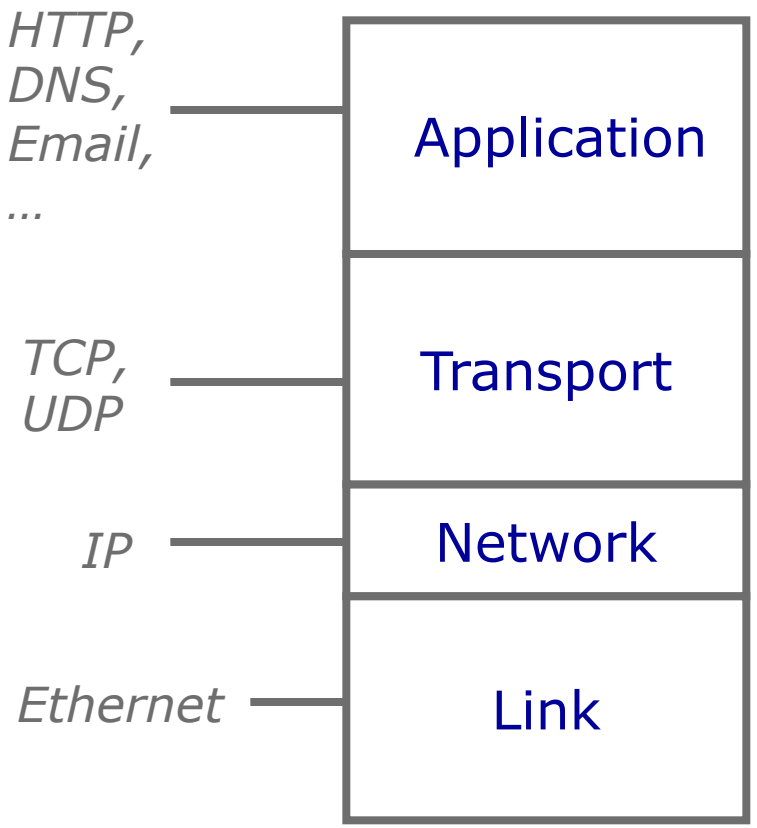
Networks implement link abstraction

- Communication links
  - Send/receive

Source: Saltzer & Kaashoek & Morris (partial)

Recall: 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

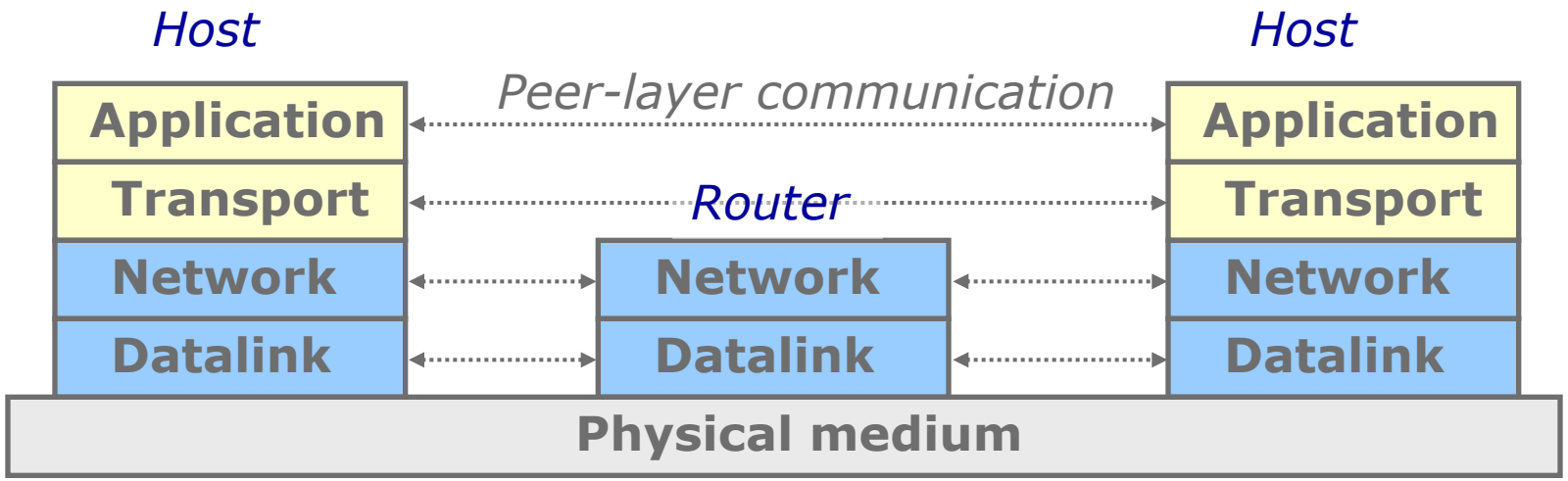


Do all servers in the network contain all layers?

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

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



Why?

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

End-to-End Argument

End-to-end argument

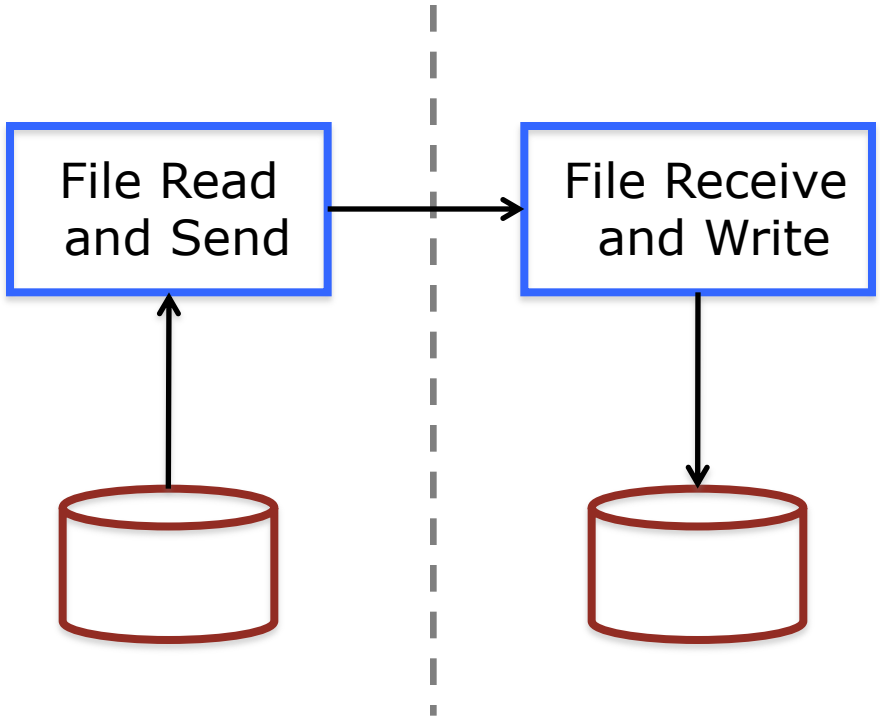
"The function in question can completely and correctly be implemented only **with the knowledge and help of the application** standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes 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

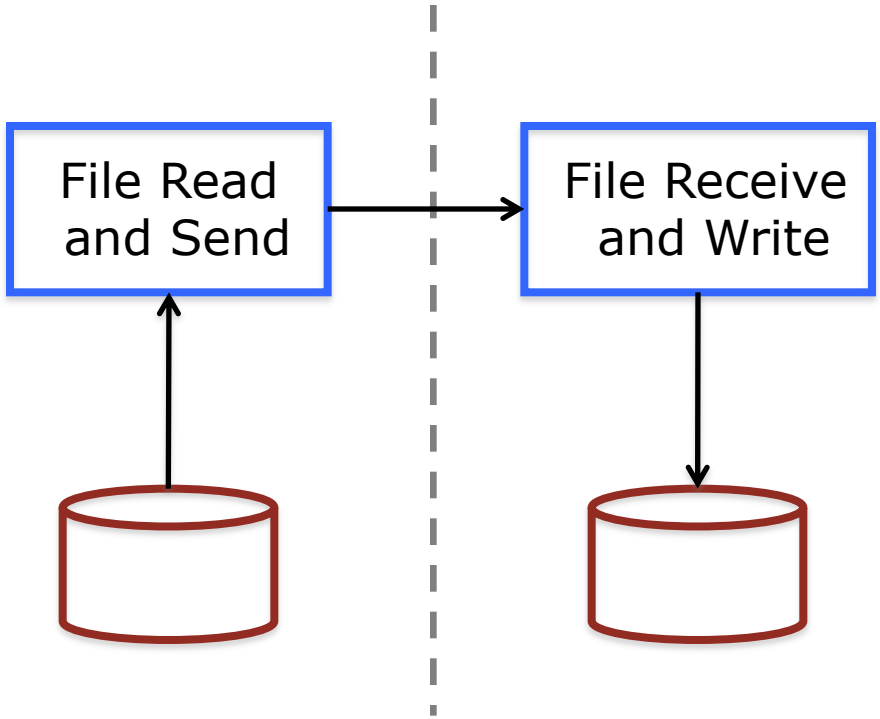
- Read from disk into memory
- Organize packets, pass to communication system
- Receive packets, re-organize file in memory
- Write file to disk



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

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**

Communication Abstractions

Source: Tanenbaum & Van Steen

Different Types of Communication

- Synchronous vs. Asynchronous
  - Synchronize at request submission (1)
  - Synchronize at request delivery (2)
  - Synchronize after being fully processed by recipient (3)
- Transient vs. Persistent (temporal decoupling)

Examples	Persistent	Transient
Asynchronous	Email	UDP Erlang
Synchronous	Message-queueing systems (1) Online Chat (1) + (2)	Asynchronous RPC (2) RPC (3)

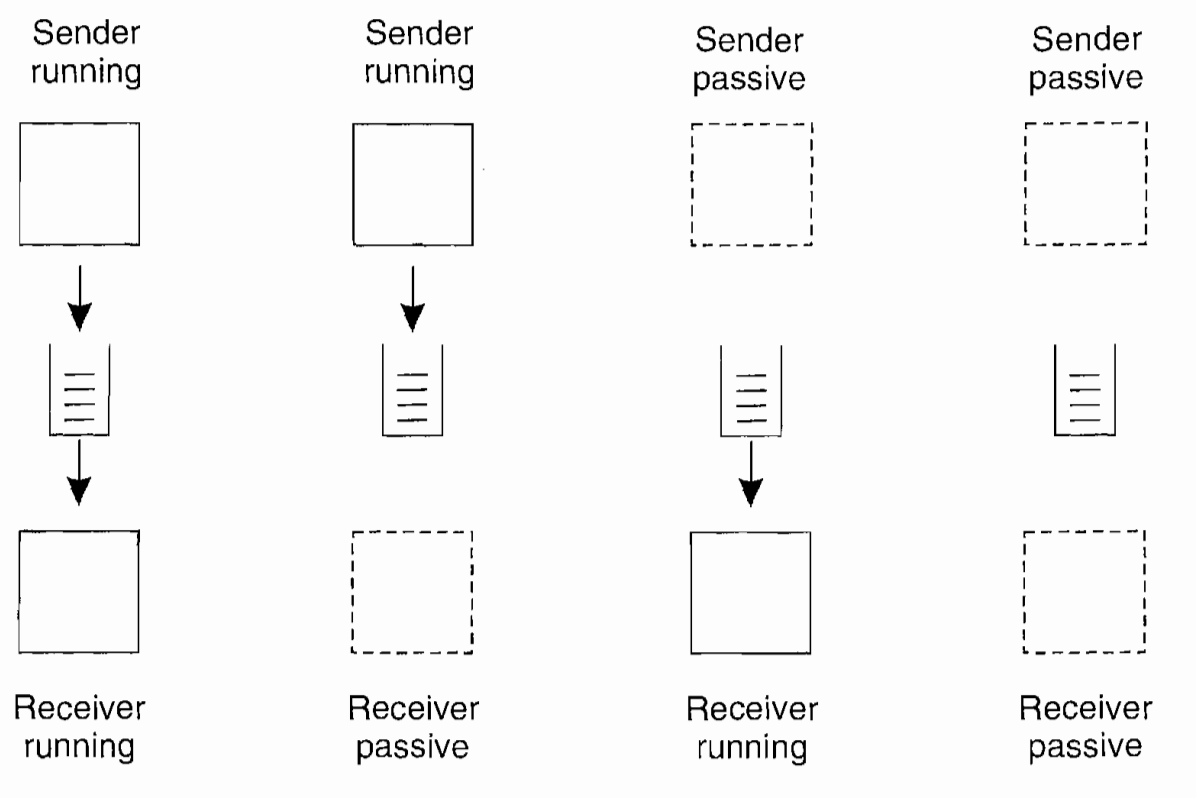
Two Types of Communication Abstractions

- **RPC**
  - Transparent and hidden communication
  - Synchronous
  - Transient
- **Message-Oriented**
  - Explicit communication `SEND/RECEIVE` of point-to-point messages
  - Synchronous vs. Asynchronous
  - Transient vs. Persistent

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

Message-Oriented Persistent Communication

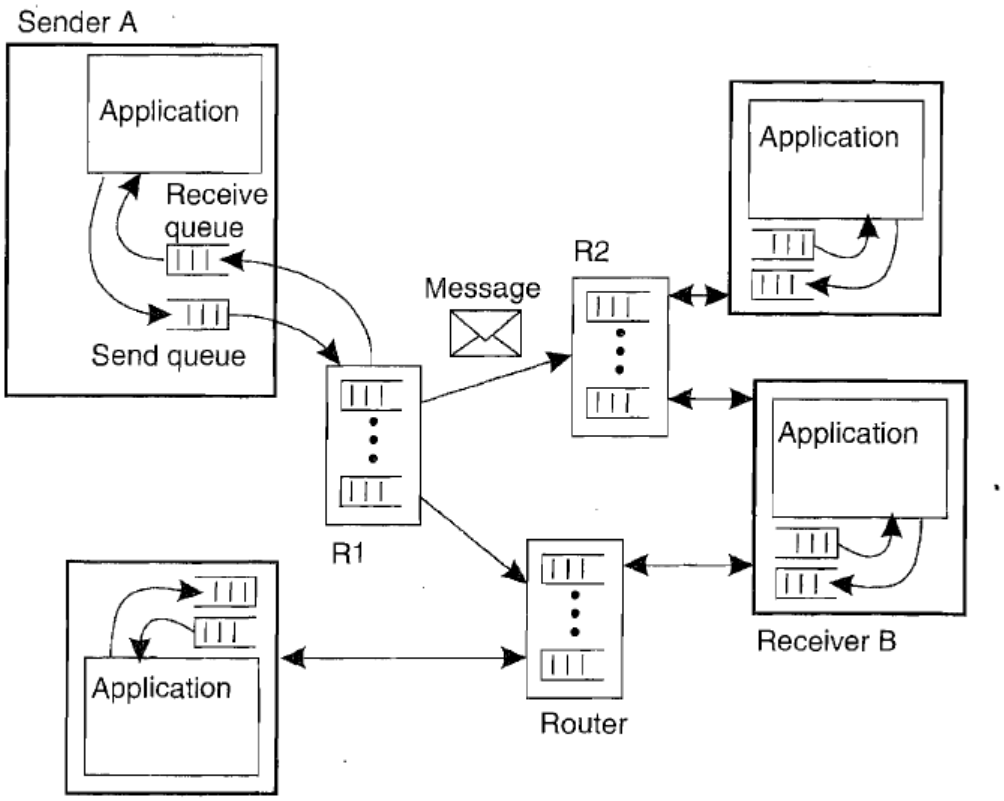
- Queues make sender and receiver loosely-coupled
- Modes of execution of sender/receiver:



Source: Tanenbaum & Van Steen

Queue Interface

Put	Put message in queue
Get	Remove first message from queue (blocking)
Poll	Check for message and remove first (non-blocking)
Notify	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

More Types of Communication Abstractions

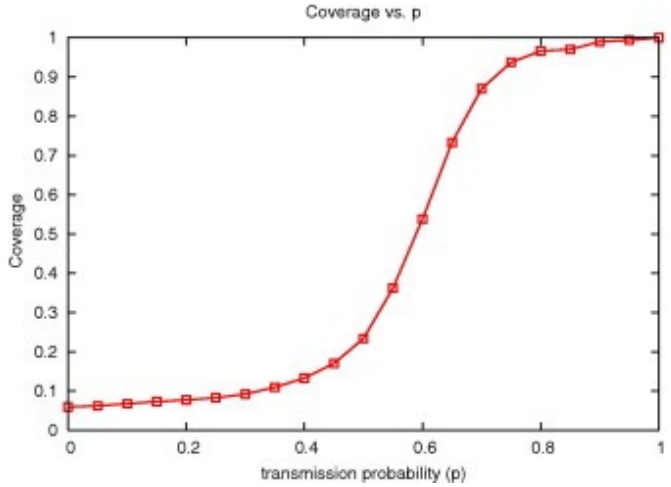
- **Stream-Oriented**
  - Continuous vs. discrete
  - Asynchronous vs. Synchronous vs. Isochronous
  - Simple vs. complex
- **Multicast**
  - `SEND/RECEIVE` over groups
  - Application-level multicast vs. gossip

More details in compendium

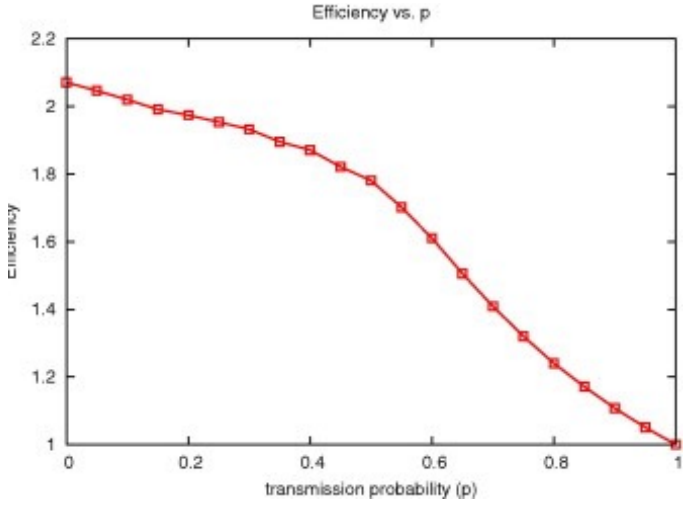
Source: Saltzer & Kaashoek & Morris (partial), Tanenbaum & Steen (partial)

A Word about Gossip

- **Epidemic protocols**
  - No central coordinator
- **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))
- **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



Coverage vs. trans prob



Efficiency vs. trans prob

Source: Haas et al. (partial)

Questions so far?

Coordinations

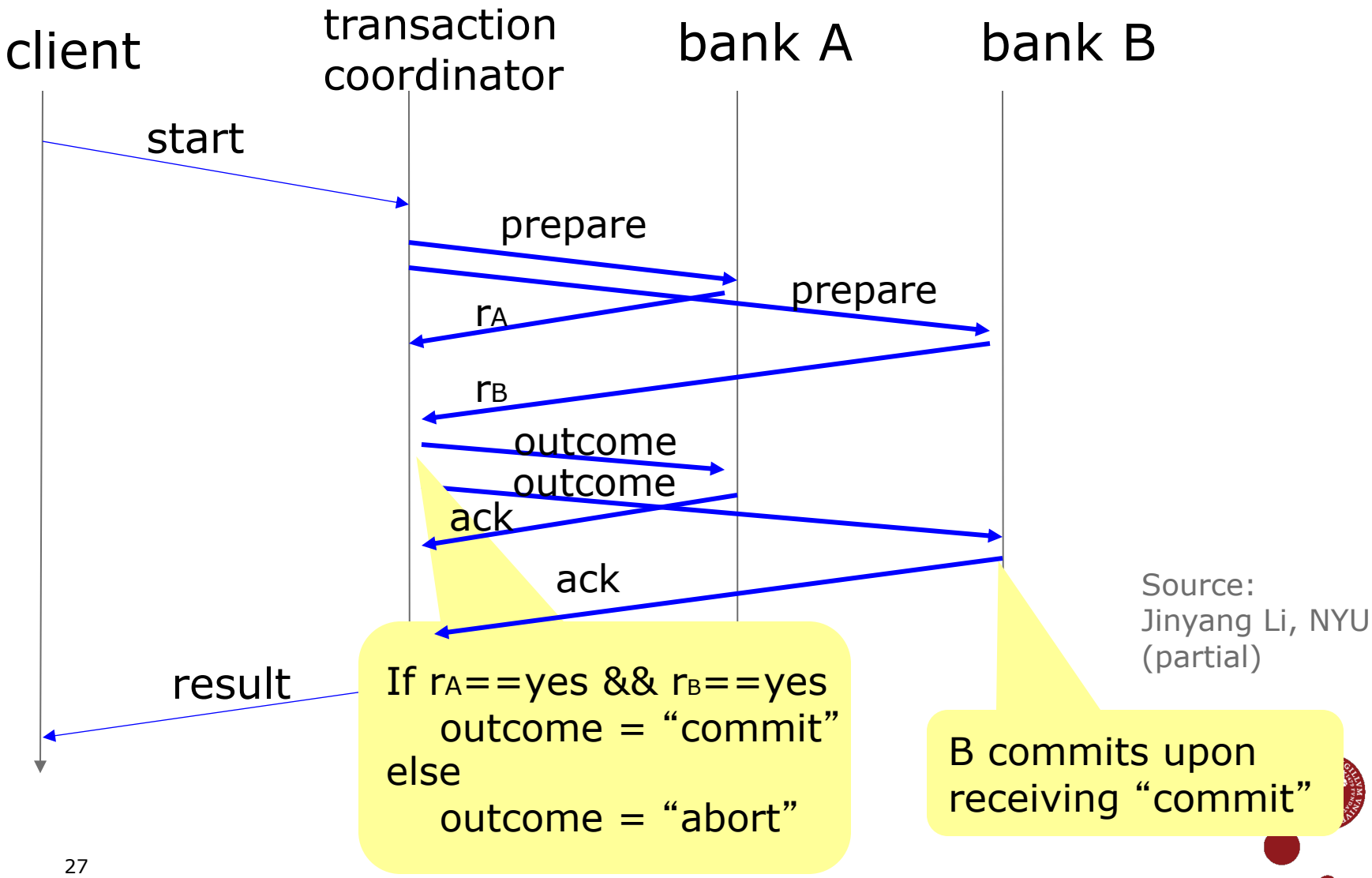
Employing Queues to Decouple Systems

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 the TRANSFER transaction using Message-Oriented Middleware (MOM)?
- How to achieve ACID?

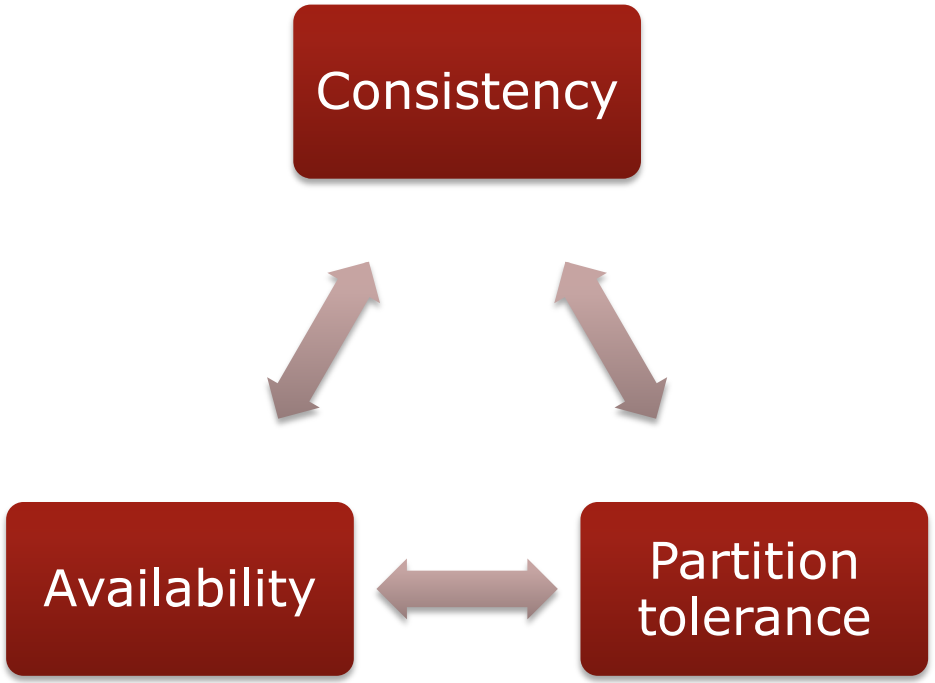
Recall: Two-Phase Commit (2PC)



Source: Jinyang Li, NYU (partial)

The CAP Theorem

- Consistency: client perceives set of operations occurred all at once (i.e., atomicity)
- Availability: every request must result in an intended response
- Partition tolerance: operations terminate, even if the network is partitioned.



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

Source: Pritchett (partial)

ACID

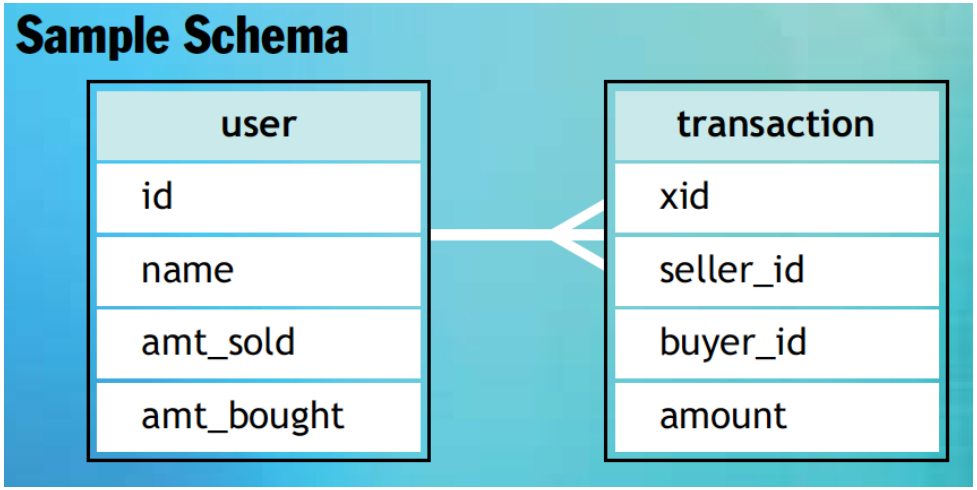
- Using 2PC guarantees atomicity (i.e. “C” in CAP theorem)
- If 2PC is used, a transaction is not guaranteed to complete in the case of network partition (either Abort or Blocked)
  - i.e. we choose C over A
- But what if an application can benefit from choosing A over C
  - How to achieve this?

From ACID to BASE

- BASE
  - Basically-Available: only components affected by failure become unavailable, not whole system
  - Soft-State: a component's 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

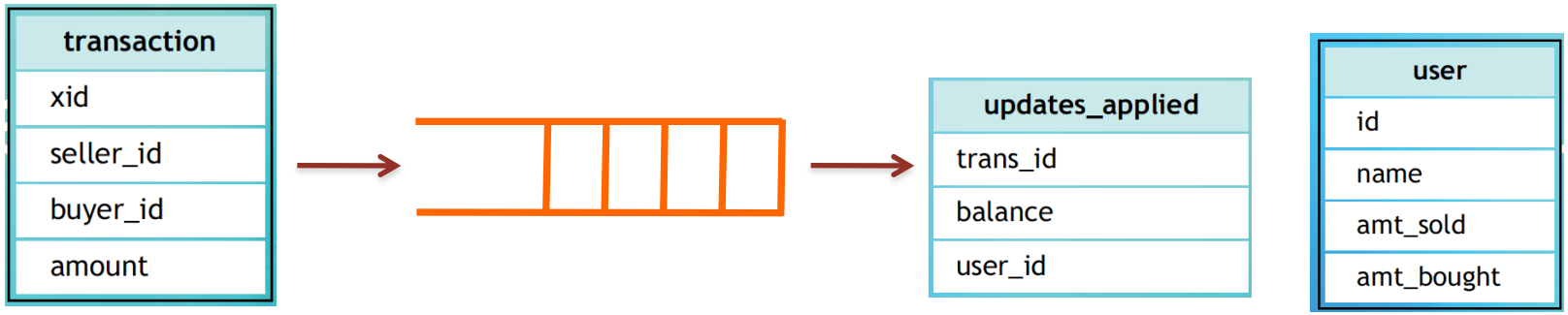
- Users buy and sell items
- Simple transaction for item exchange:



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 of msgs
  - Idempotence
  - Order

Source: Pritchett (partial)

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
- Identify alternative communication abstractions such as data streams and multicast / gossip