CS 755 – System and Network Architectures and Implementation

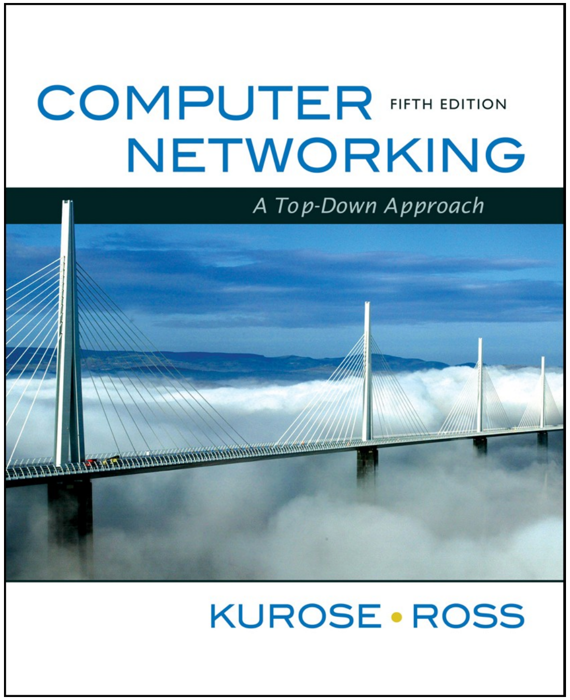
Module 4 – Remote Services

Martin Karsten

mkarsten@uwaterloo.ca

Notice

Some figures are taken from third-party slide sets. In this module, parts are taken from the Kurose/Ross and the Tanenbaum/van Steen slide set. See details on next slides...



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.

DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition
ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 4
Communication

Overview

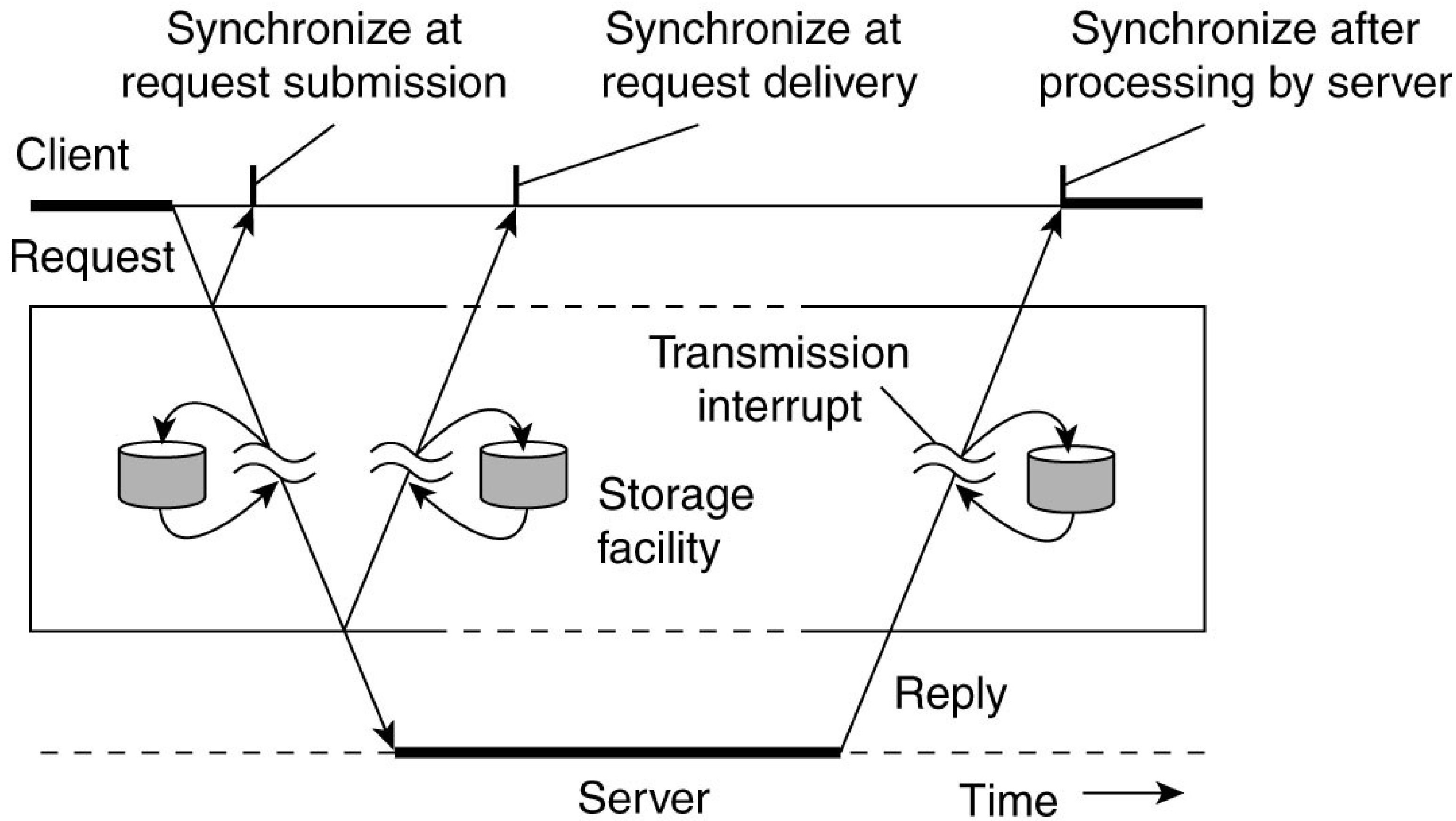
- messaging / message queueing
- remote procedure call
- security

Transport - Review

- multiplexing, virtual channel
  - process-to-process communication
- reliability
- flow and congestion control
- connection management

- participants: online and available!

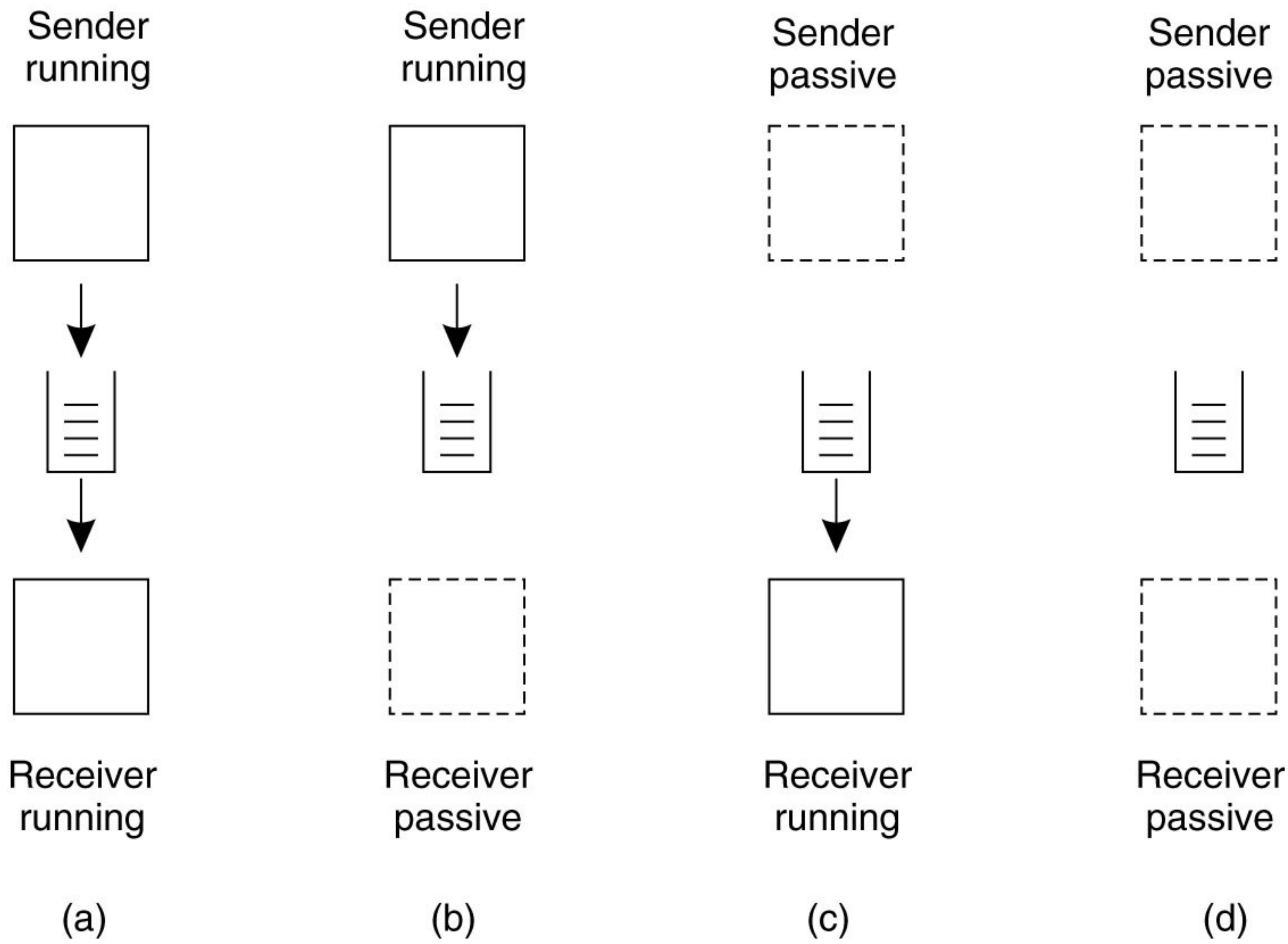
Communication - Synchronization



Messaging

- *persistent* communication
  - sender can terminate after sending message
  - receiver does not need to be online
  - vs. *transient* communication
- *asynchronous* communication
  - sender can continue other work after sending
    - vs. sender waits for acknowledgement
  - receiver is notified when message is available
    - vs. receiver blocks waiting for message

Persistency and Synchronization



Messaging Middleware

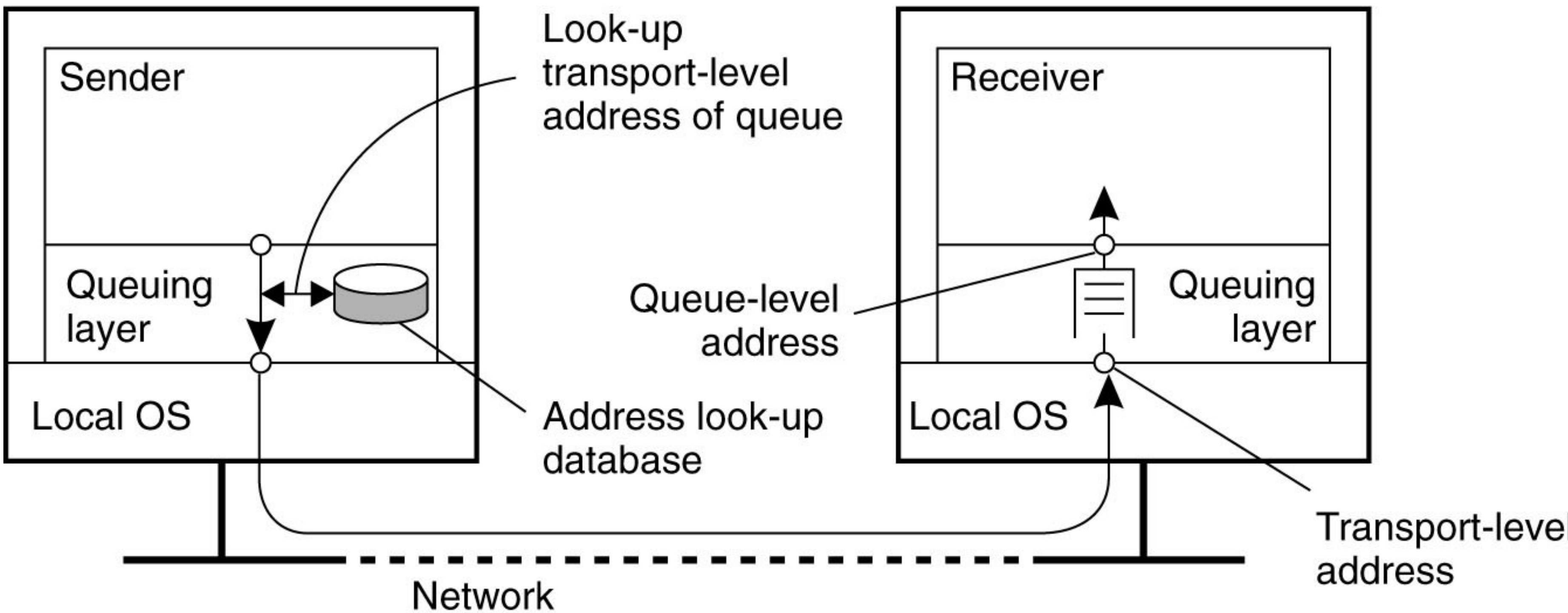
- persistence – reliability
- management, tracing, availability
- flexible integration with heterogeneous systems
  - OS, network, programming language, etc.
- group communication: publish / subscribe
  - underlying distribution model: unicast vs. broadcast

Messaging Queueing Primitives

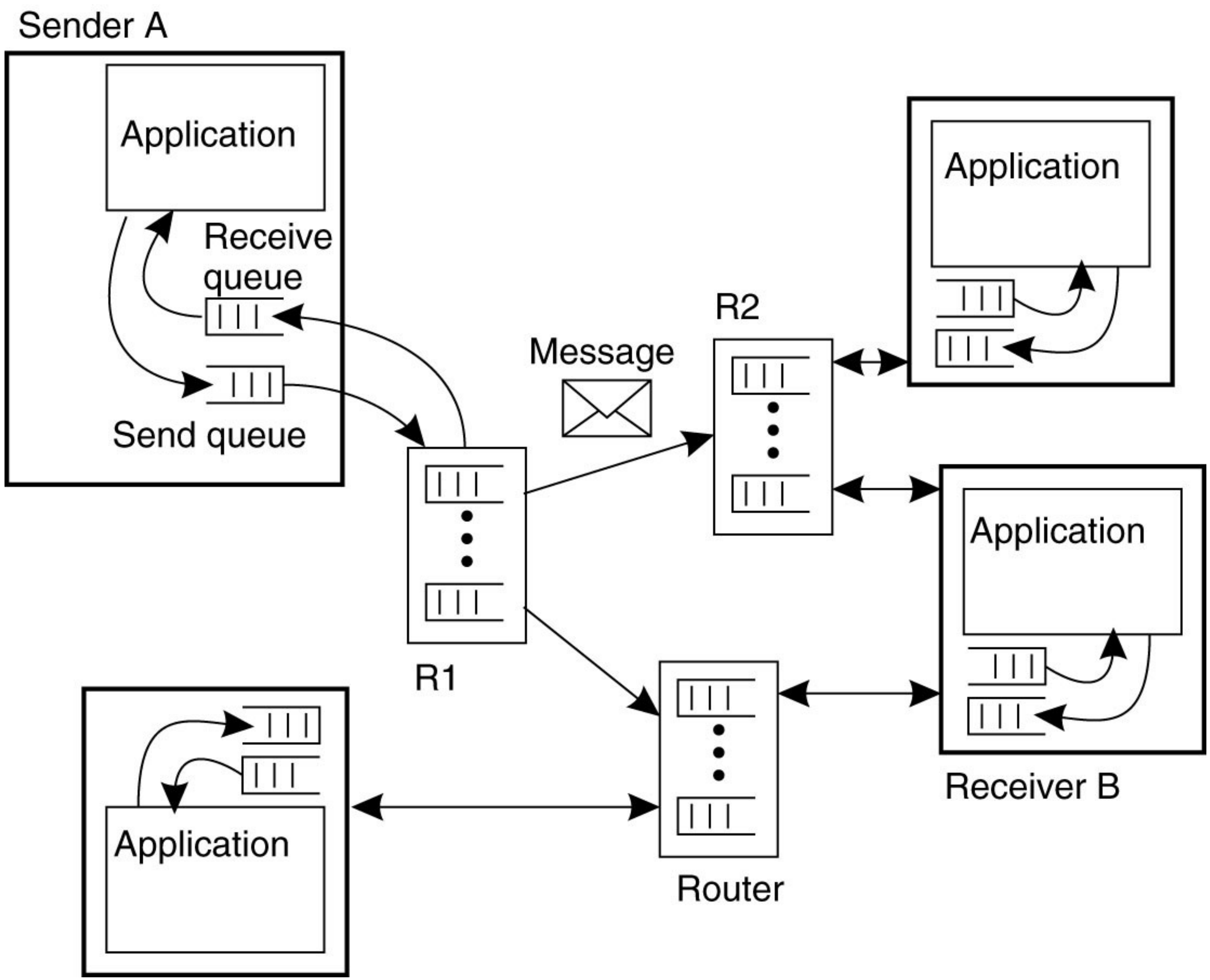
- Put – append message to queue (send)
- Get – retrieve message from queue (receive)
- Poll – check queue(s) for message availability
- Notify – install asynchronous retrieve handler

- need buffer decoupled from sender, receiver
- relay nodes for larger networks
  - addressing, routing, forwarding, etc., as usual

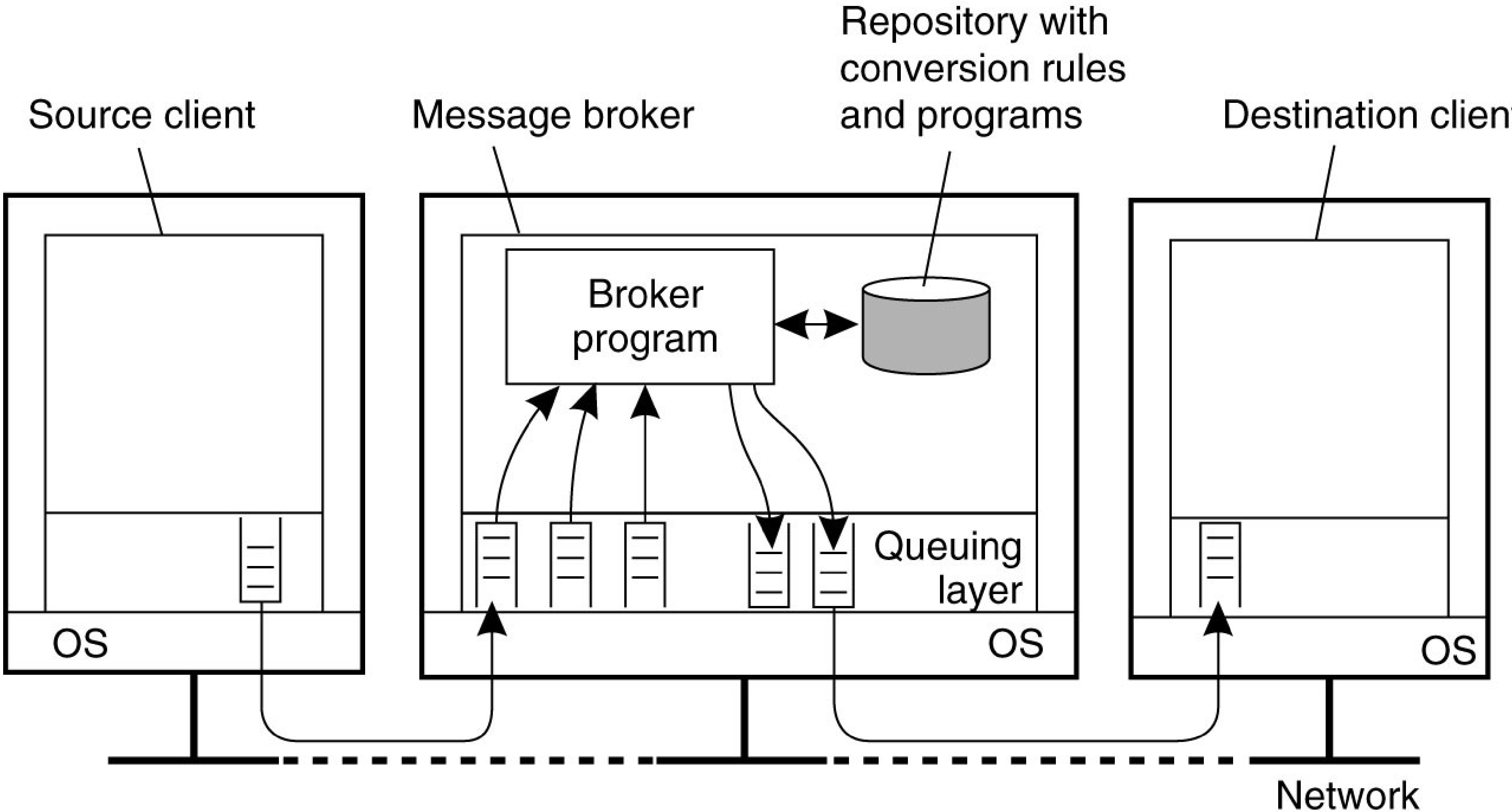
Architecture



Architecture



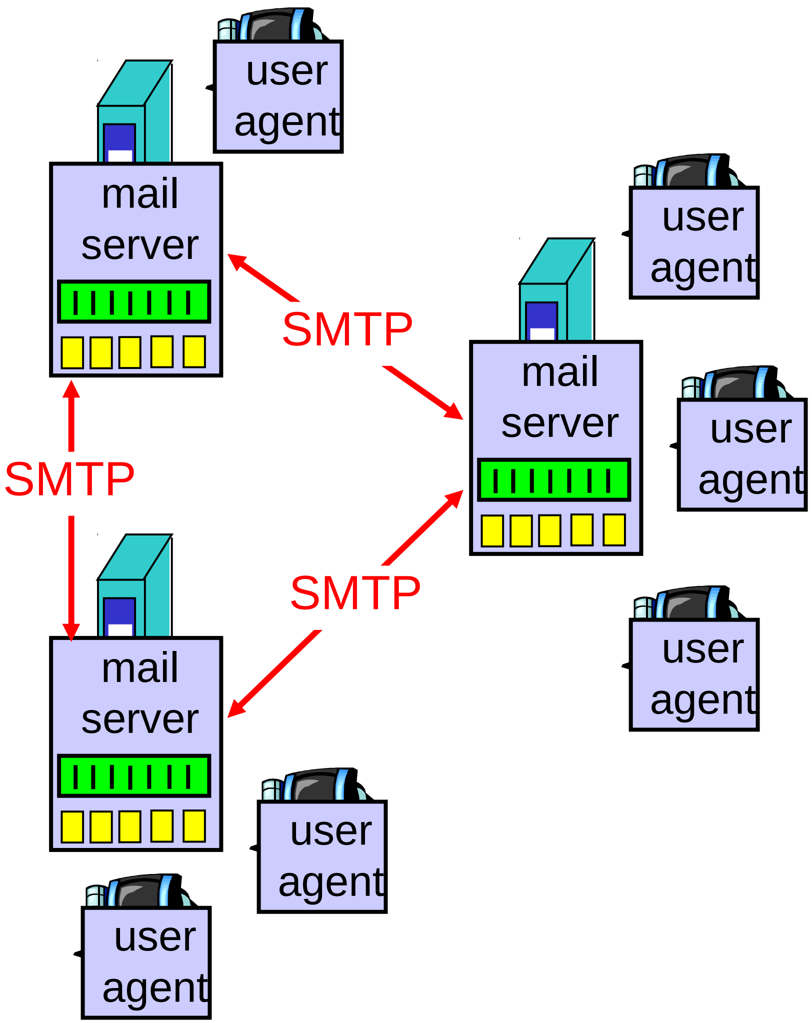
Message Broker



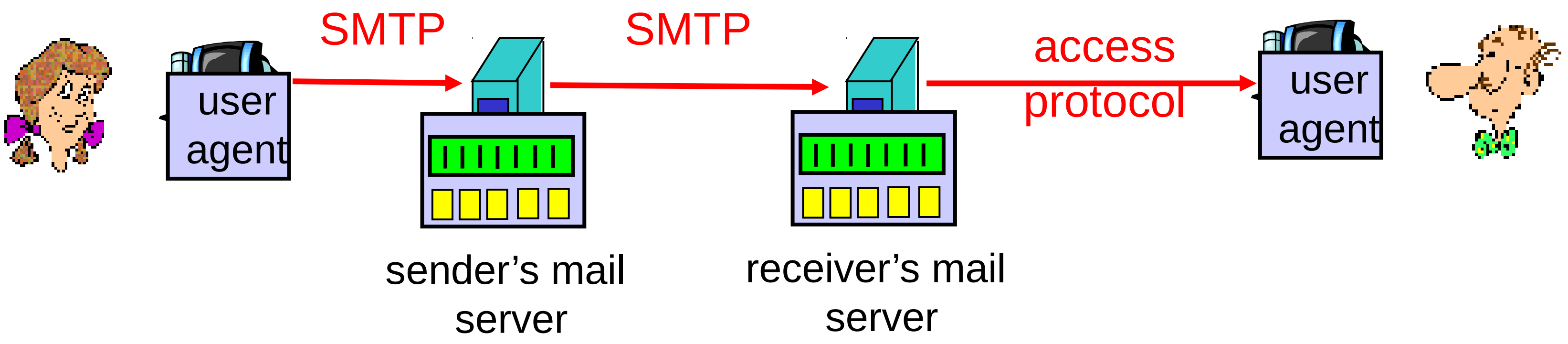
Example: Email

mail servers

- incoming messages mailbox
- outgoing message queue
- communication protocol: SMTP
  - reliable server-to-server transfer

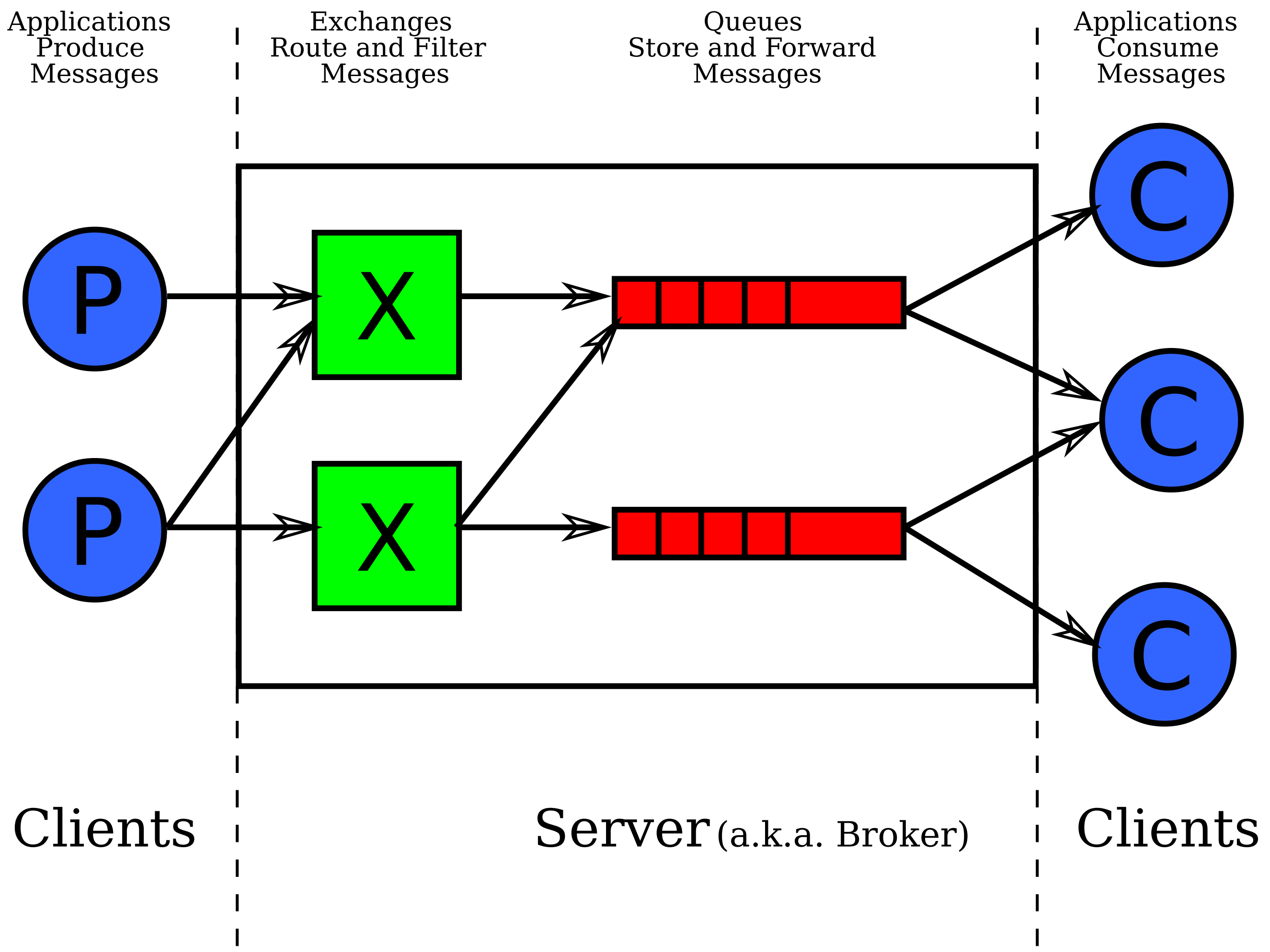


Email Access Protocols



- sender: synchronous, transient to server
- receiver: asynchronous, persistent from server
  - Post Office Protocol (POP) – old & simple
  - Internet Mail Access Protocol (IMAP) – better
  - HTTP – POP, IMAP, etc in background
  - remote file system and file-based (elm, pine, etc.)

Advanced Message Queuing Protocol (AMQP)



Message Passing Interface (MPI)

- portable abstraction of socket interface
- weaker semantics than message queueing

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

Publish/Subscribe

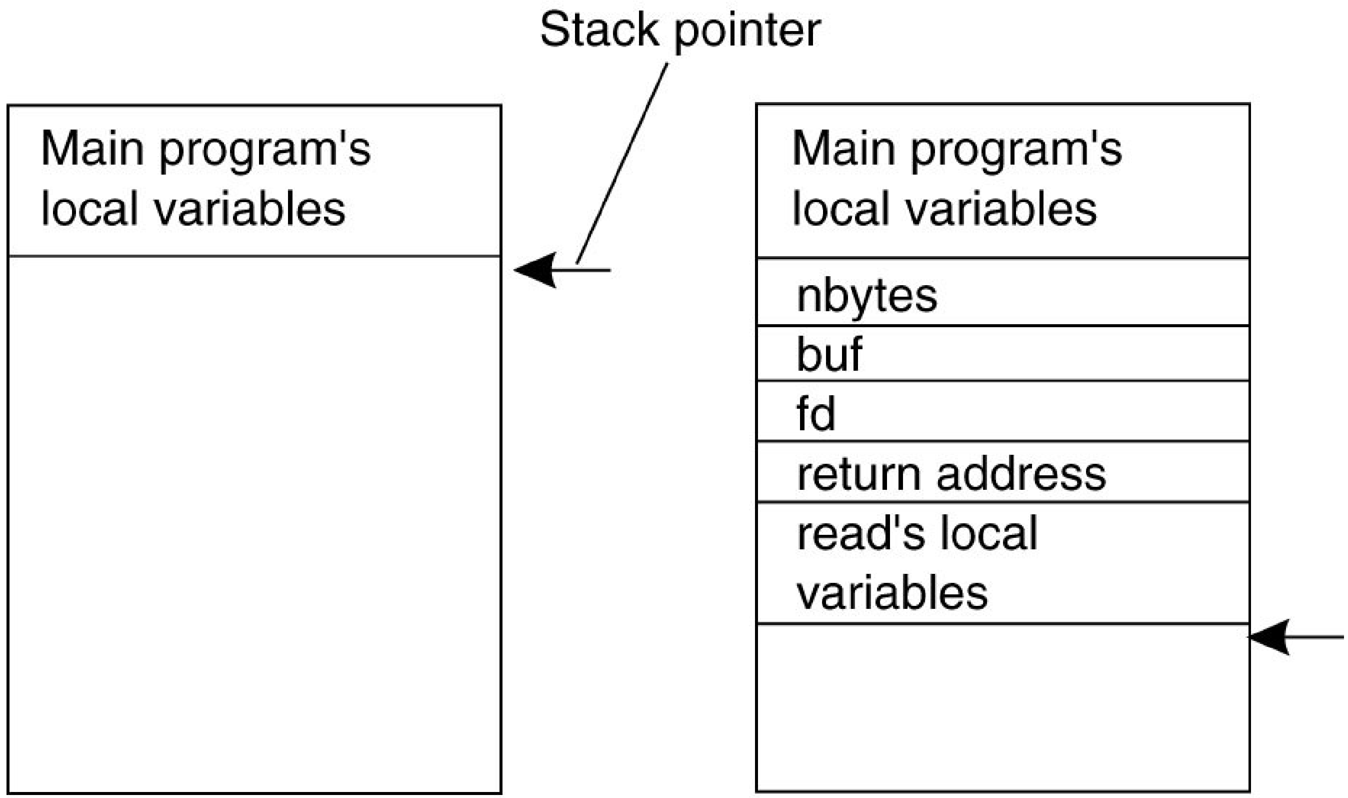
- special case of messaging
- notion of “queue” replaced by arbitrary filter
  - structured / topic
  - unstructured / content

Remote Procedure Call

- transparent execution of remote functionality

- example: Sun RPC aka ONC RPC
- classic UNIX RPC system
  - developed with/for Network File System (NFS)
- available on most UNIX systems
- see: `man rpc`

Conventional Procedure Call



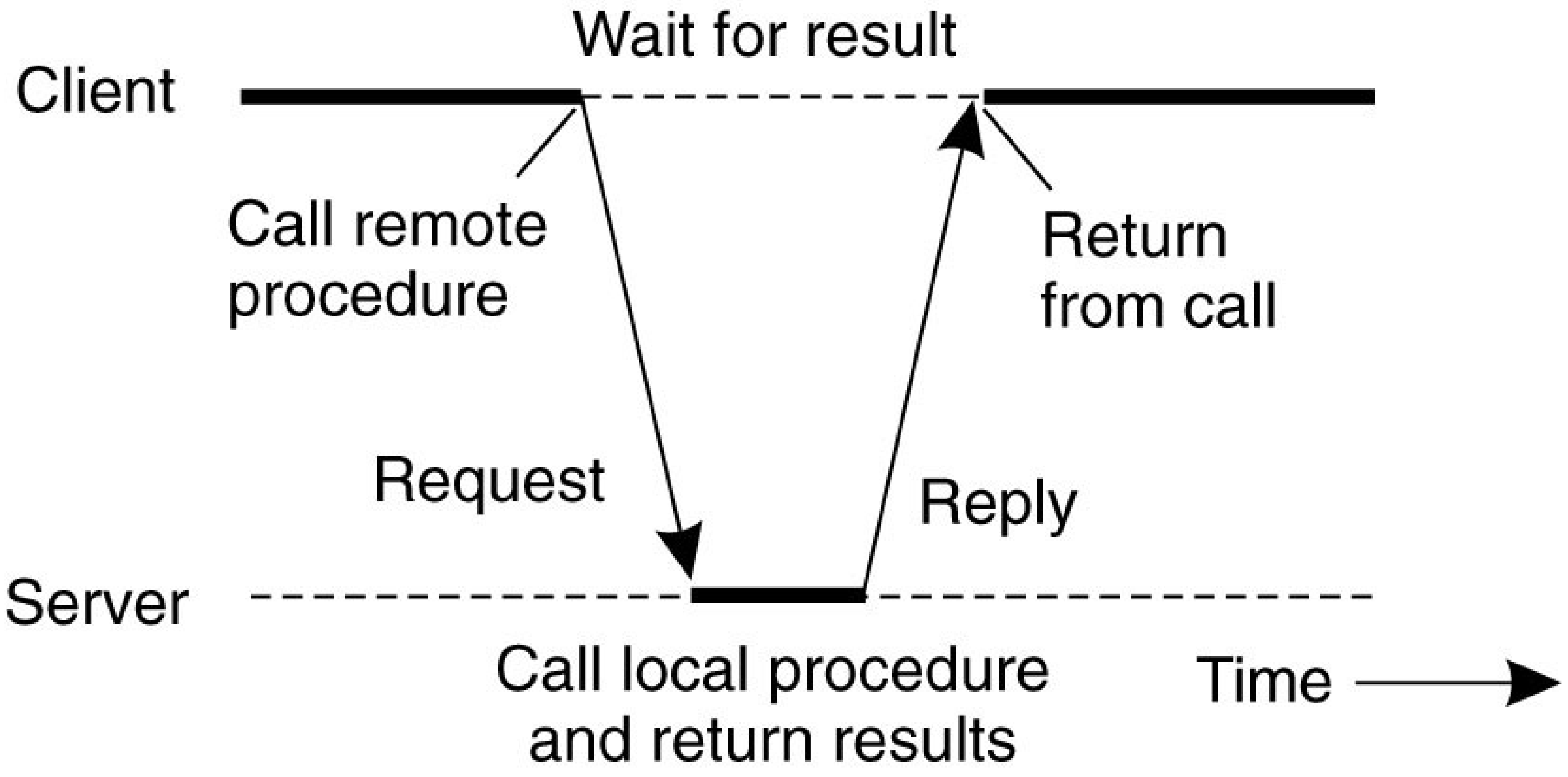
int len = read(fd, buf, nbytes);

RPC - Challenges

- machine architecture
- address space
- parameter passing
- independent failures

- goal: transparency

Remote Invocation



RPC Details

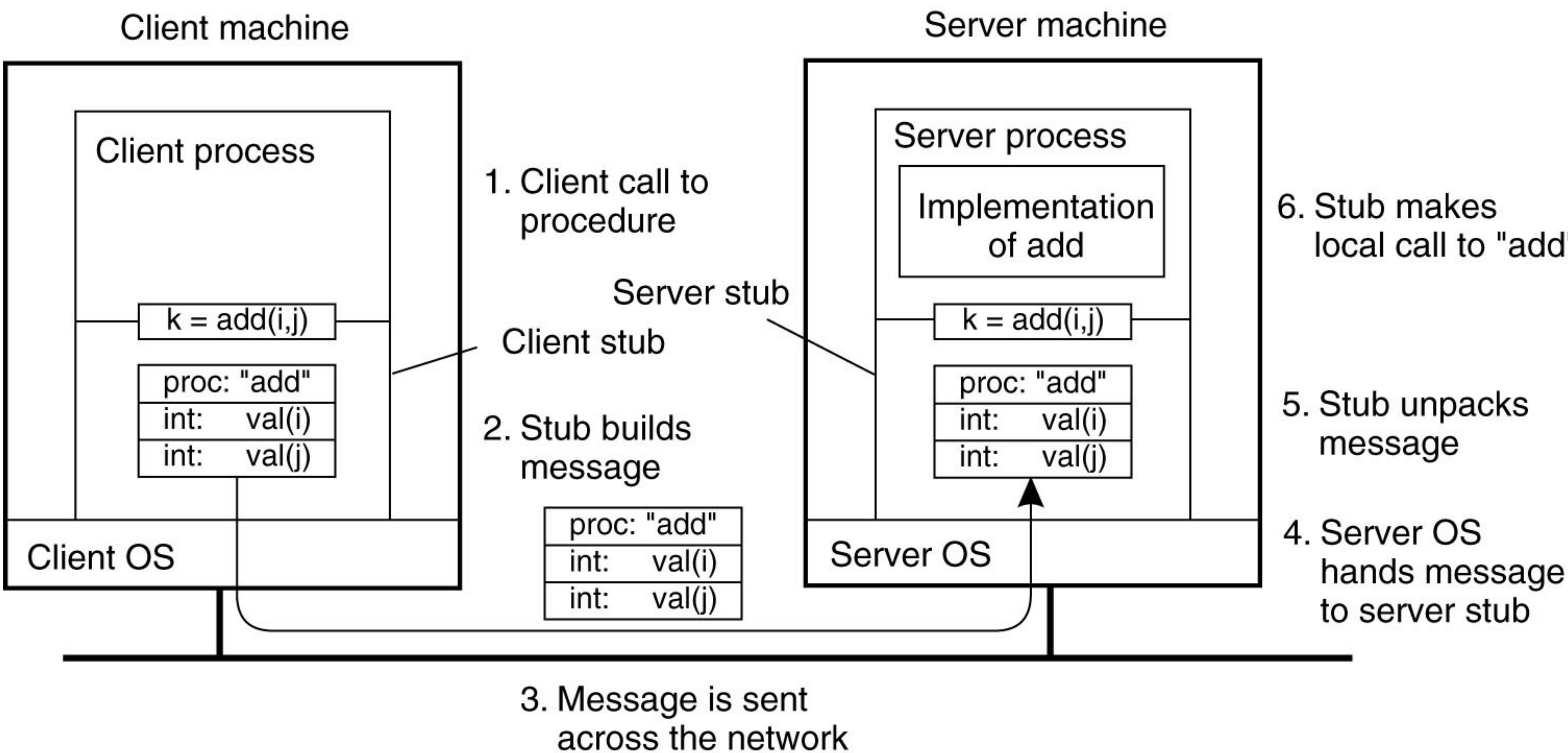
1. Client procedure calls *client stub* locally.
2. Client stub builds message and calls local OS.
   - *marshalling*: parameters -> message
3. Client OS sends message to server OS.
4. Server OS gives message to *server stub*.
5. Server stub unpacks parameters and calls server routine.
   - *de/unmarshalling*: message -> parameters

...

RPC Details

6. Server routine executes and returns to stub.
7. Server stub builds message and calls local OS.
8. Server OS sends message to client OS.
9. Client OS gives message to client stub.
10. Client stub unpacks result and returns to client.

RPC Details


Client machine
Server machine
Client process
k = add(i,j)
proc: "add"
int: val(i)
int: val(j)
Client OS
1. Client call to procedure
Server stub
Client stub
2. Stub builds message
proc: "add"
int: val(i)
int: val(j)
Server process
Implementation of add
k = add(i,j)
proc: "add"
int: val(i)
int: val(j)
Server OS
6. Stub makes local call to "add"
5. Stub unpacks message
4. Server OS hands message to server stub
3. Message is sent across the network

Data Representation

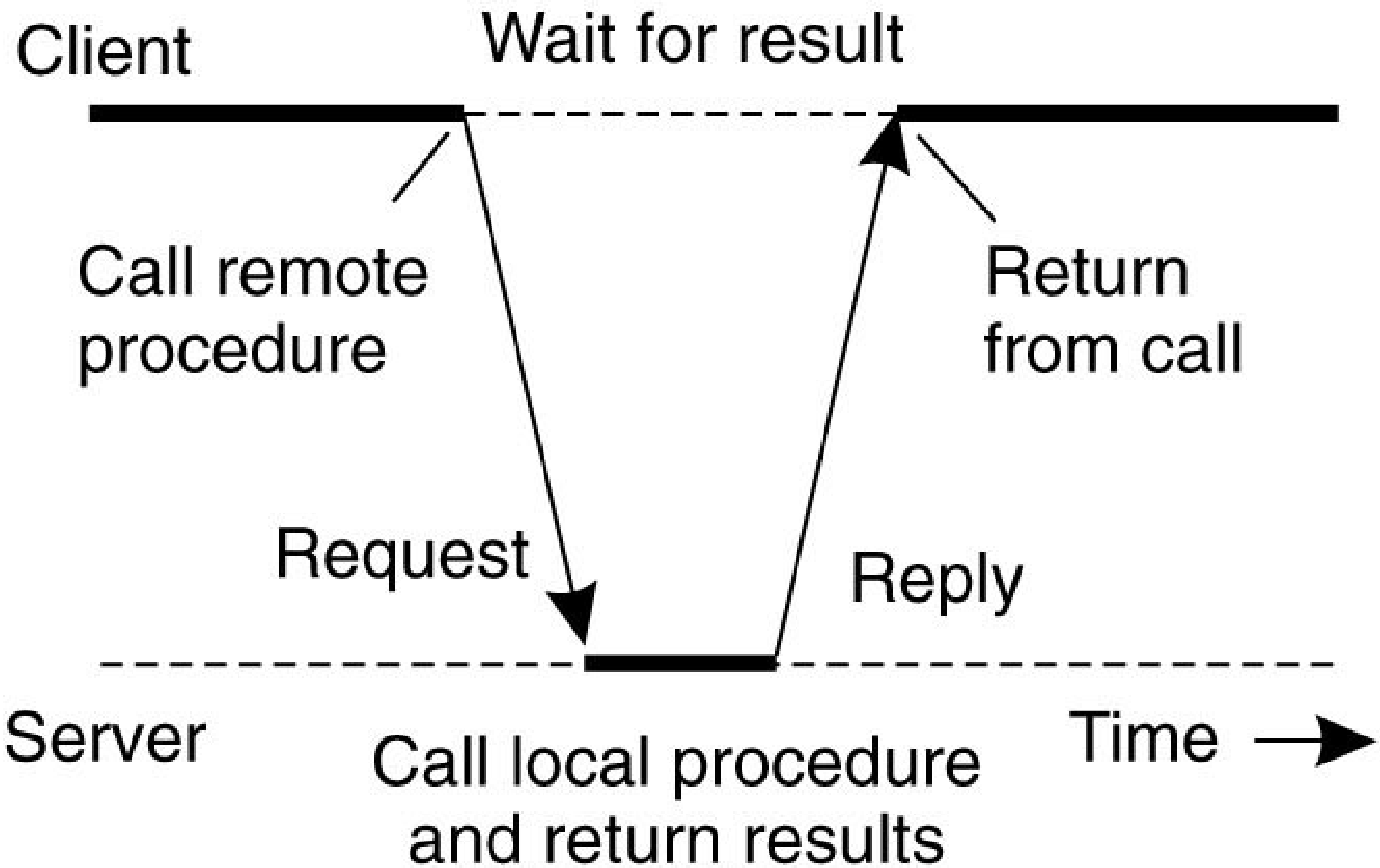
- transparency across platforms
  - Sun RPC: eXtensible Data Representation (XDR)

- hardware architecture
- operating system
- programming language
- runtime environment

Data Represenation

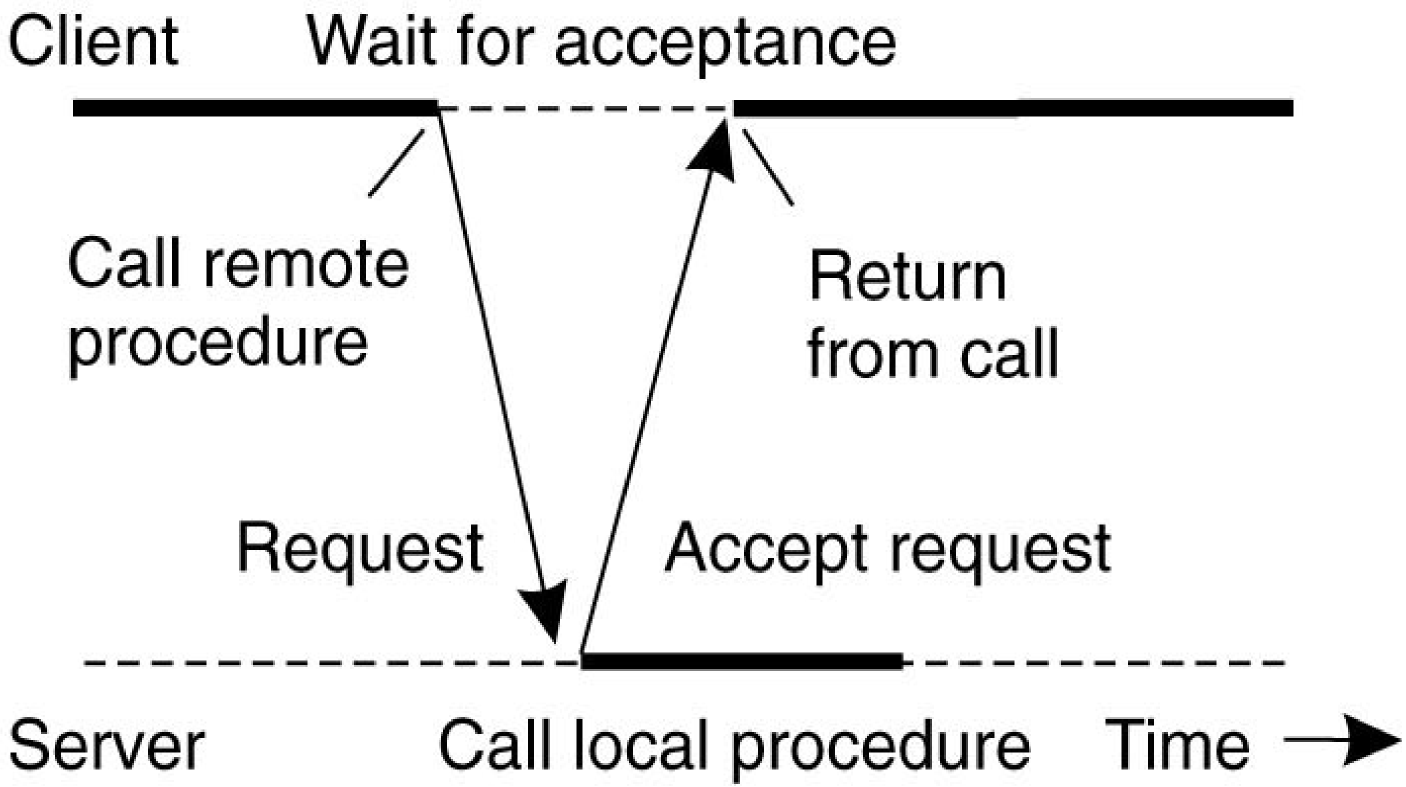
- common example: integer representation
  - *little endian* vs. *big endian*
- others: float, string, structures...
- dynamic data structures: list, tree, etc.
- objects?

Synchronous RPC



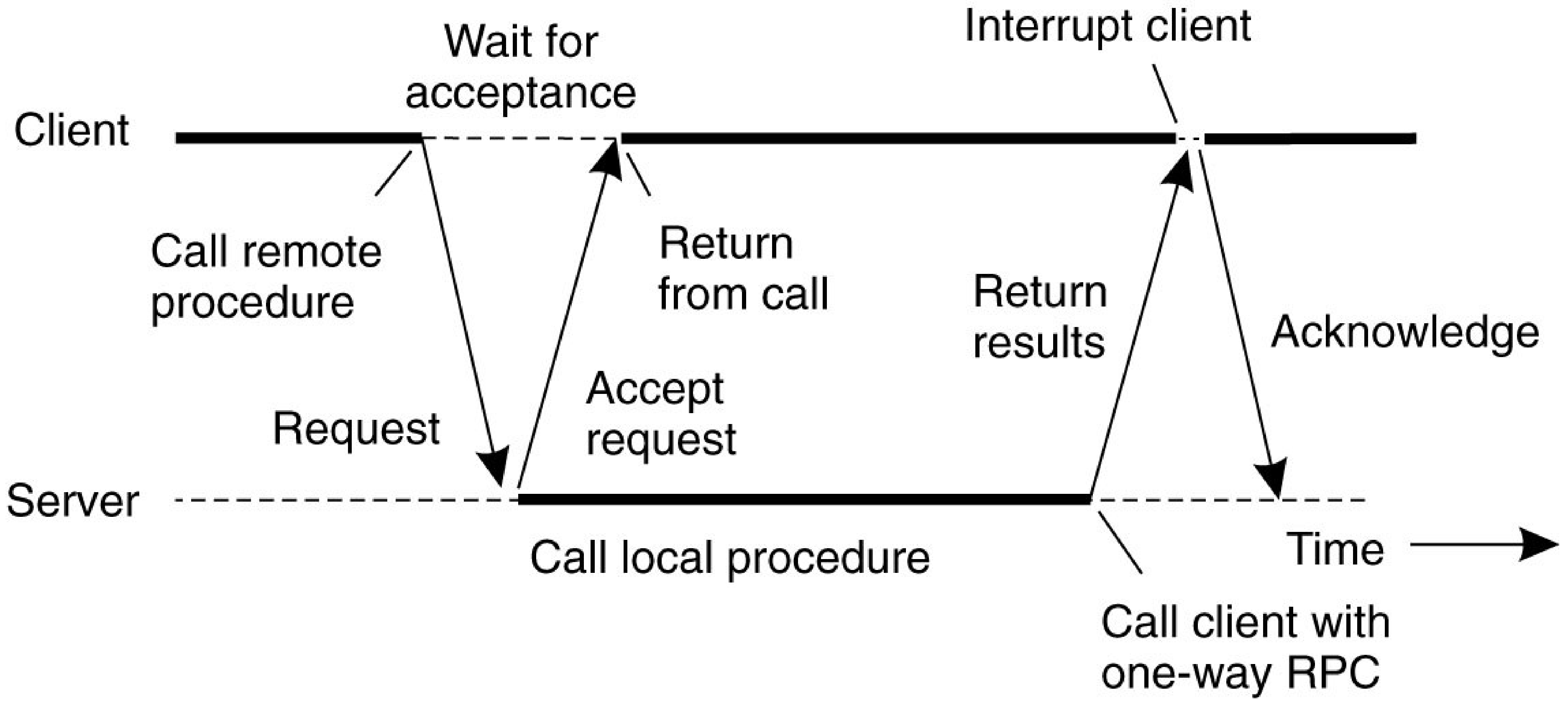
(a)

Asynchronous RPC

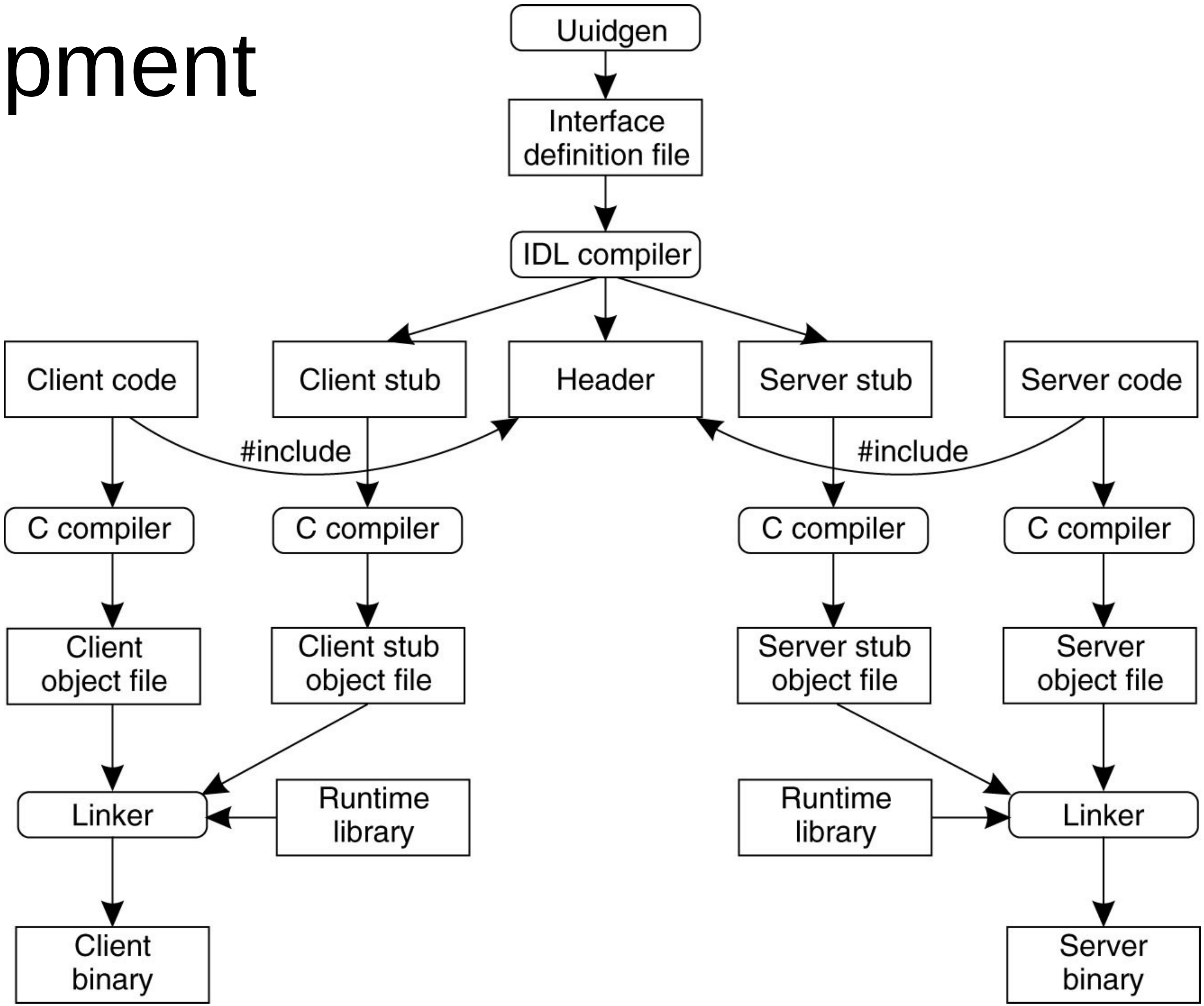


(b)

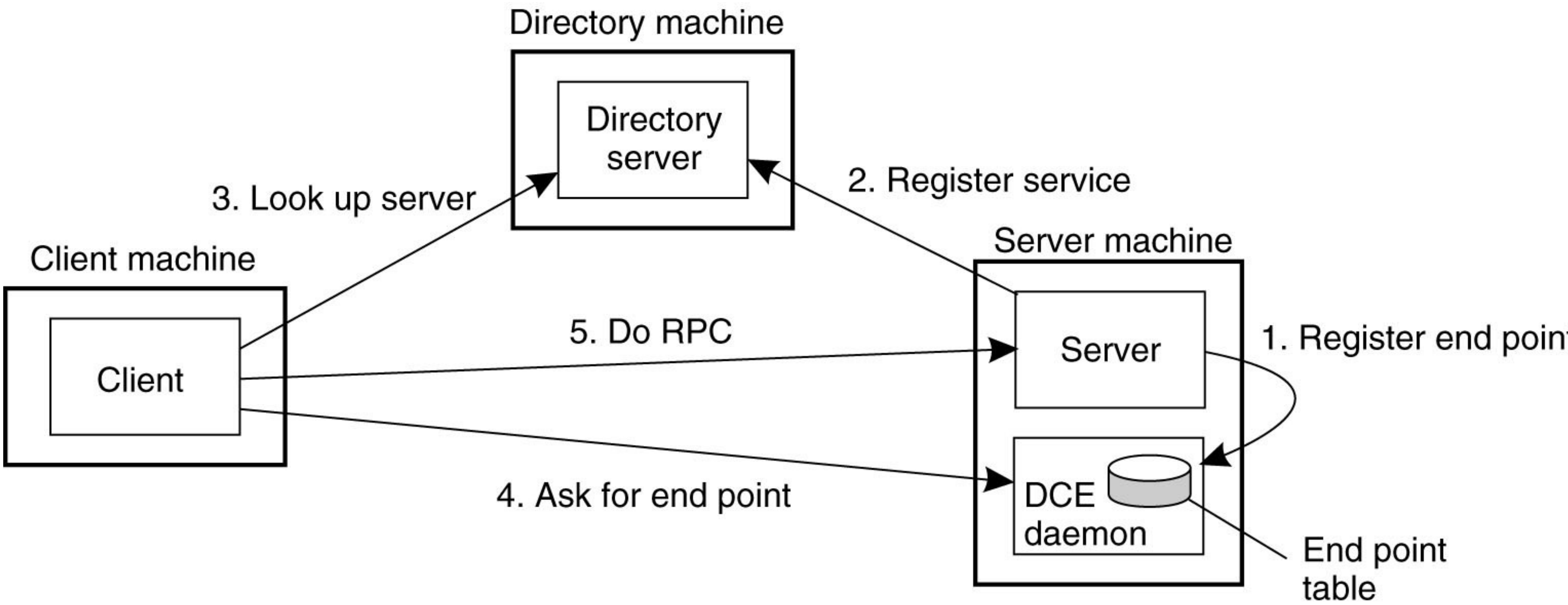
Two-Way Asynchronous RPC



Development

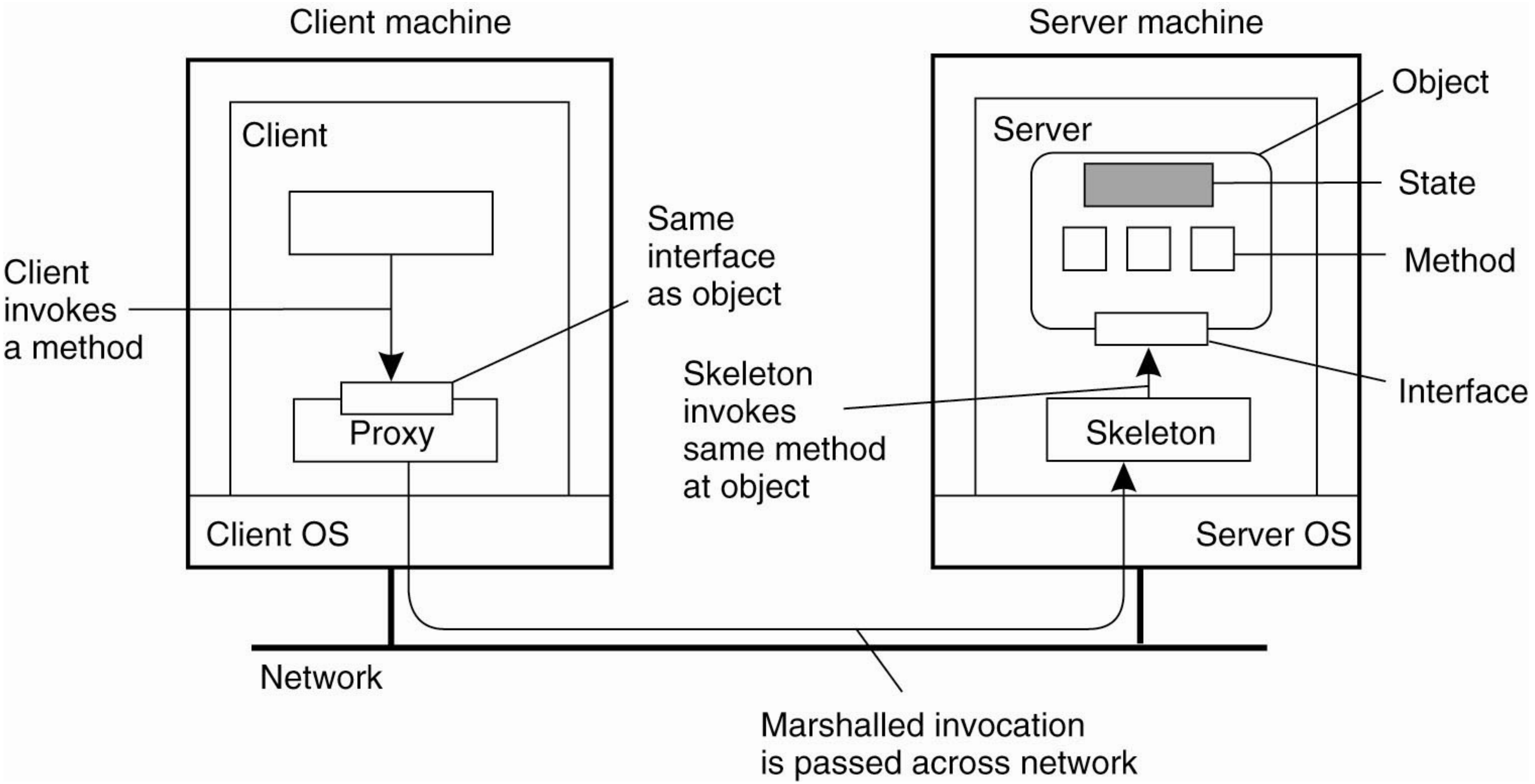


Runtime

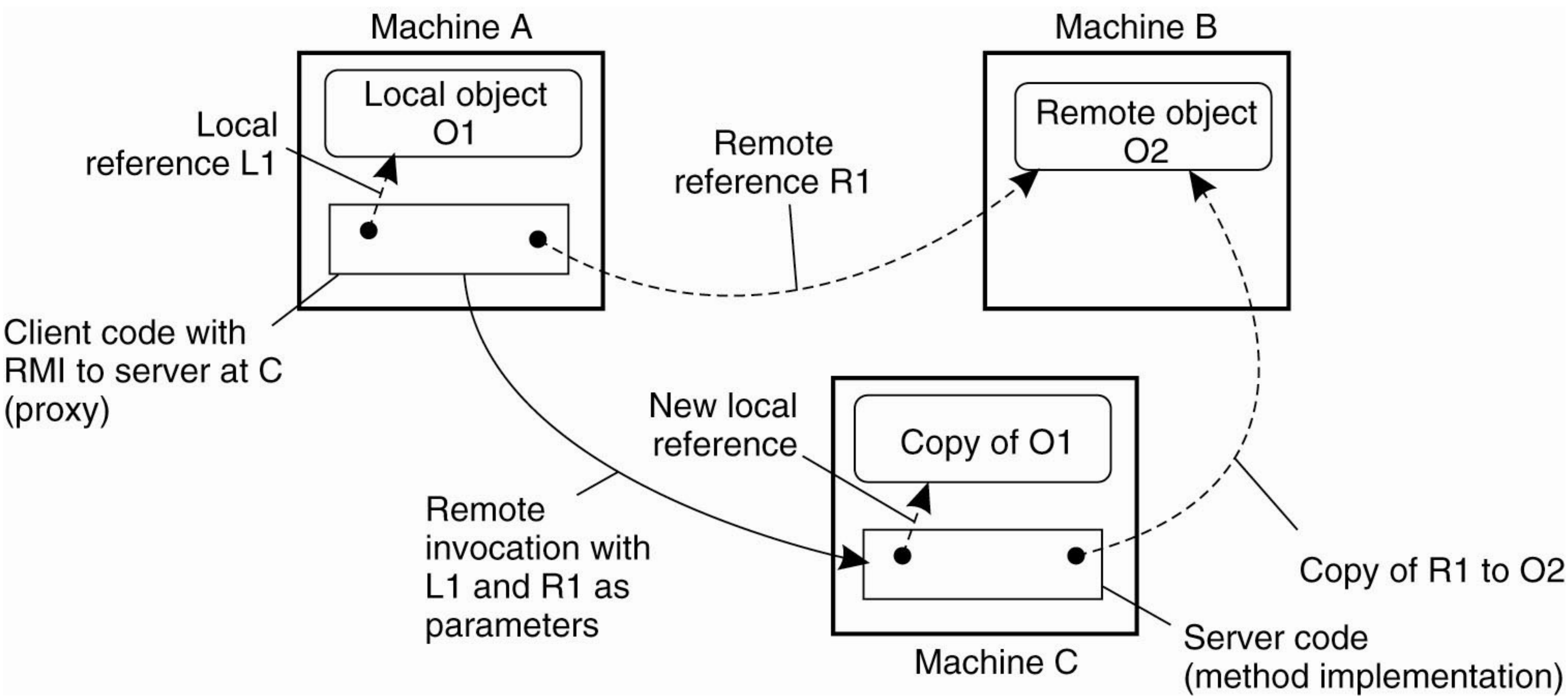


Directory machine
Directory server
3. Look up server
2. Register service
Server machine
Client machine
5. Do RPC
1. Register end point
Client
Server
DCE daemon
4. Ask for end point
End point table

Distributed Objects



Object References



Other RPC-Type Systems

- DCE -> DCOM/ODBC
- CORBA
- Java RMI
- SOAP

- Data Representation: XML

What is network security?

Authentication: sender, receiver want to confirm identity of each other

Confidentiality: only sender, intended receiver should "understand" message contents

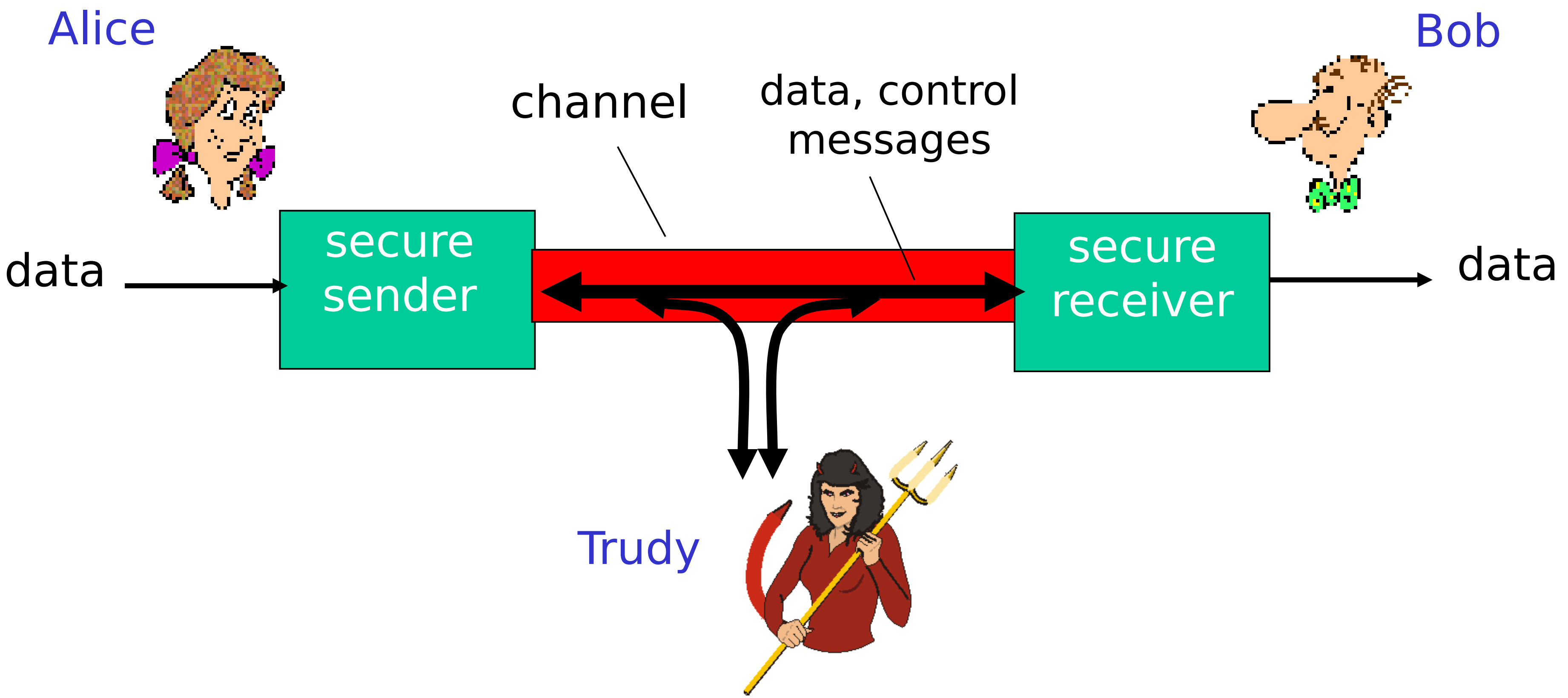
- sender encrypts message
- receiver decrypts message

Message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

Access and availability: services must be accessible and available to users

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



Who might Bob, Alice be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- etc...

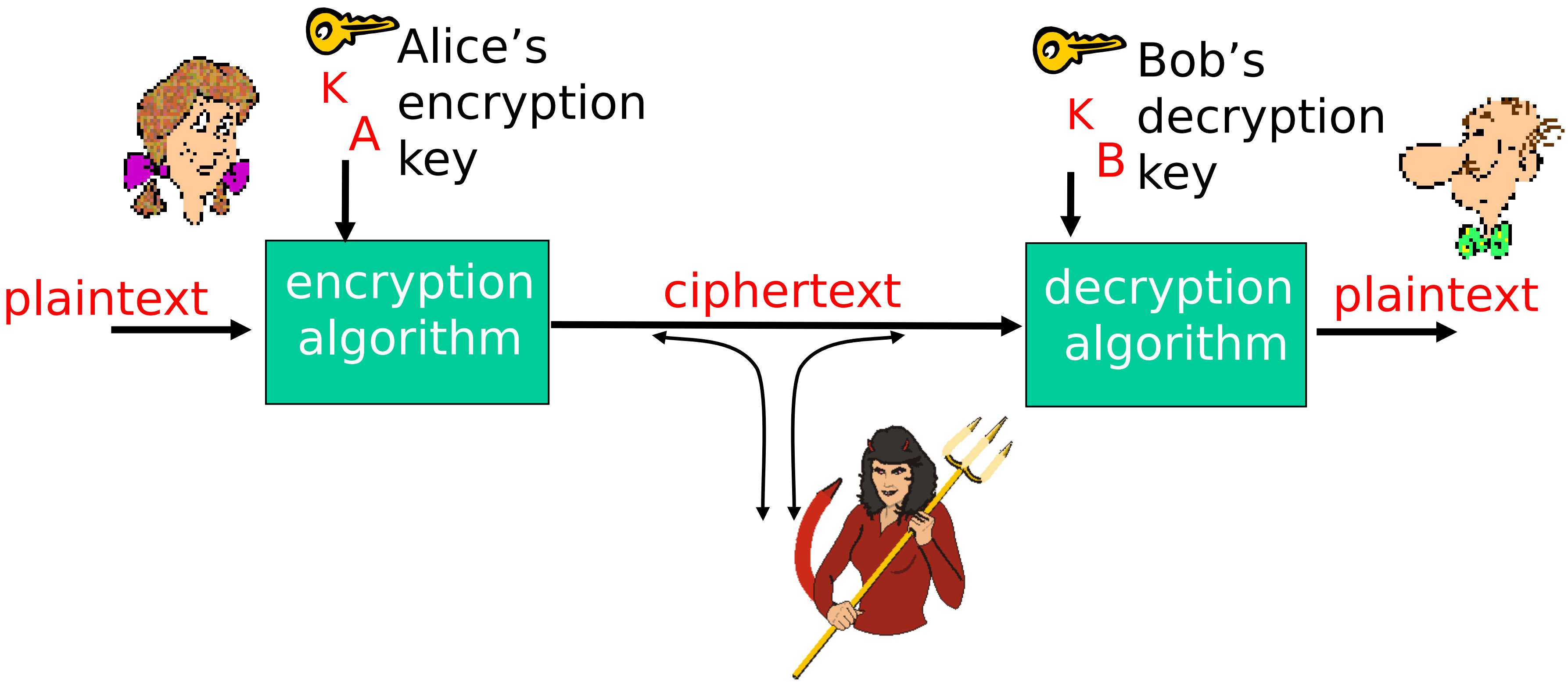
There are bad guys (and girls) out there!

Q: What can a "bad guy" do?

A: a lot!

- *eavesdrop:* intercept messages
- actively *insert* messages into connection
- *impersonation:* can fake (spoof) source address in packet (or any field in packet)
- *hijacking:* "take over" ongoing connection by removing sender or receiver, inserting himself in place
- *denial of service*: prevent service from being used by others (e.g., by overloading resources)

The language of cryptography



symmetric key crypto: sender, receiver keys *identical*

public-key crypto: encryption key *public*, decryption key *secret* (private) – or vice versa

Symmetric key cryptography

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

```
 plaintext:  abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

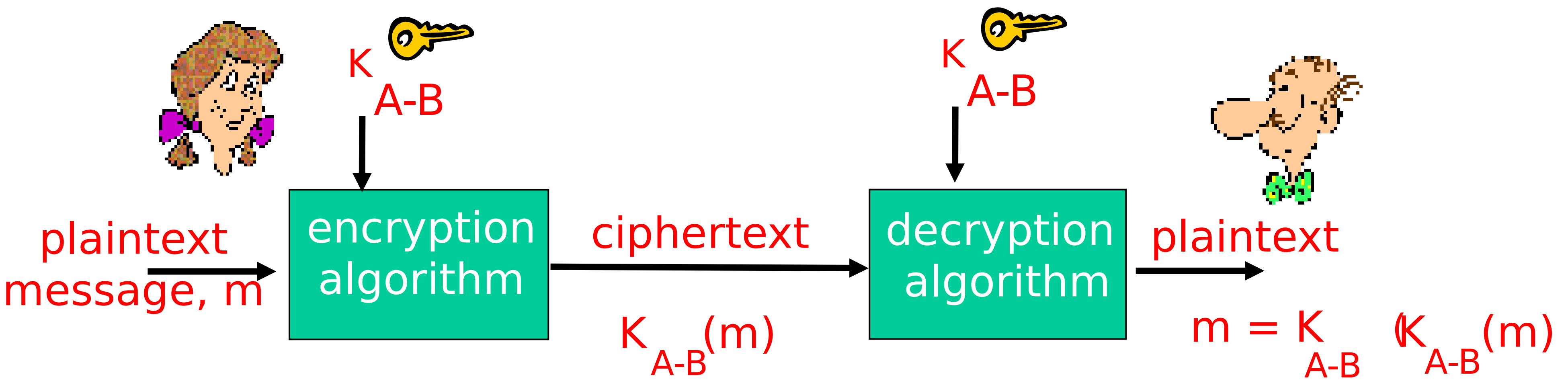
E.g.:

```
 Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Q: How hard to break this simple cipher?:

- brute force (how hard?)
- ciphertext-only vs known-plaintext vs chosen-plaintext

Symmetric key cryptography



symmetric key crypto: Bob and Alice share know same (symmetric) key: $K_{A\text{-}B}$

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher
- Q: how do Bob and Alice agree on key value?

Symmetric key crypto: DES

DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase (“Strong cryptography makes the world a safer place”) decrypted (brute force) in 4 months
  - no known “backdoor” decryption approach
- making DES more secure:
  - use three keys sequentially (3-DES) on each datum
  - use cipher-block chaining

AES: Advanced Encryption Standard

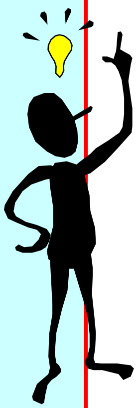
- new (Nov. 2001) symmetric-key NIST standard, replacing DES
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

Public key cryptography

symmetric key crypto

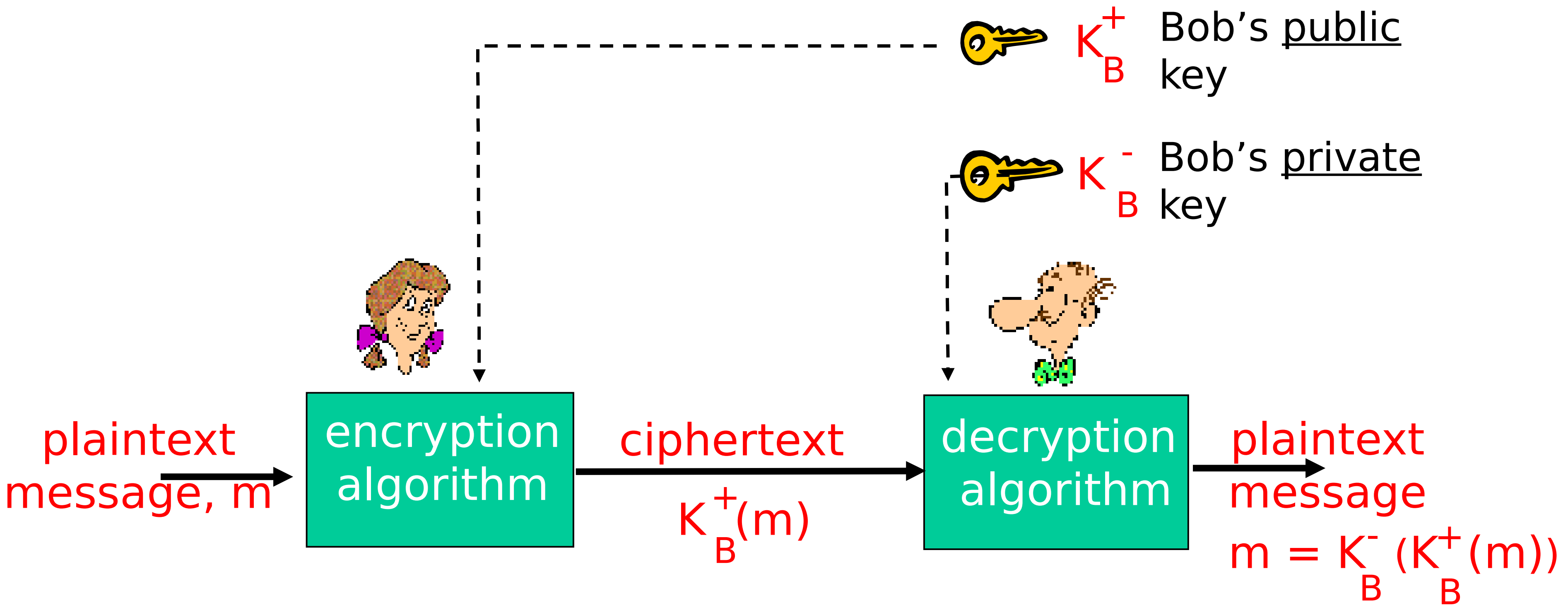
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?

public key cryptography



- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver

Public key cryptography



Public key encryption algorithms

Requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given K_B^+, cannot easily compute K_B^-

RSA: Rivest, Shamir, Adleman algorithm

RSA: another important property

The following property will be *very* useful later:

$$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$$

use public key first, followed by private key

use private key first, followed by public key

Result is the same!

Message Integrity

Bob receives msg from Alice, wants to ensure:

- message originally came from Alice
- message not changed since sent by Alice

Cryptographic Hash:

- takes input m, produces fixed length value, H(m)
  - e.g., as in Internet checksum
- computationally infeasible to find two different messages, x, y such that H(x) = H(y)
  - equivalently: given m = H(x), (x unknown), can not determine x.
  - note: Internet checksum *fails* this requirement!

Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of message
- is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

message	ASCII format
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 4F 42
	B2 C1 D2 AC

message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 4F 42
	B2 C1 D2 AC

different messages but identical checksums!

Digital Signatures

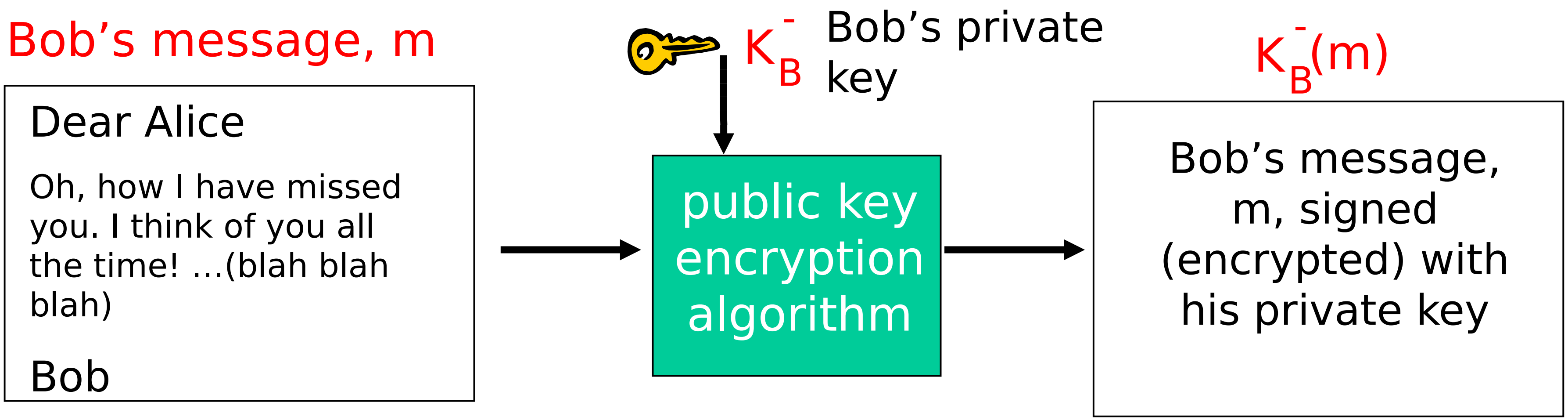
cryptographic technique analogous to hand-written signatures.

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- verifiable, nonforgeable: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital Signatures

simple digital signature for message m:

- Bob “signs” m by encrypting with his private key K_B^-, creating “signed” message, $K_B^-(m)$



Digital Signatures (more)

- suppose Alice receives msg m, digital signature $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- if $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

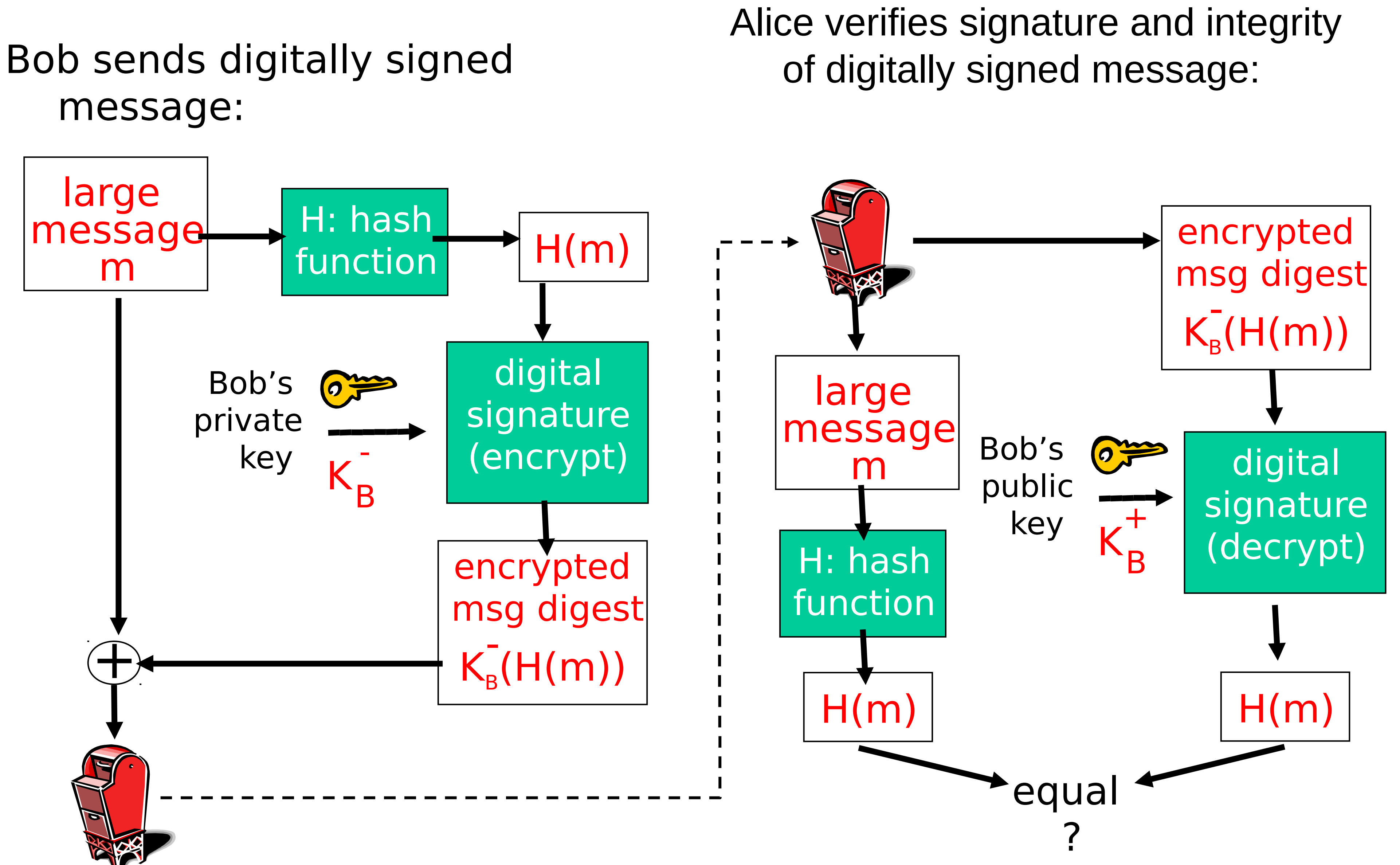
Alice thus verifies that:

- Bob signed m.
- No one else signed m.
- Bob signed m and not m'.

non-repudiation:

- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m.

Digital signature



Symmetric vs. Public Key

- symmetric (shared) key
  - less computational overhead
- public/private key
  - easier to set up
- typical compromise
  - both: key “wear-and-tear”, information leakage
  - use private key during session setup
  - negotiate shared key for session duration

Authentication

Goal: Bob wants Alice to “prove” her identity to him

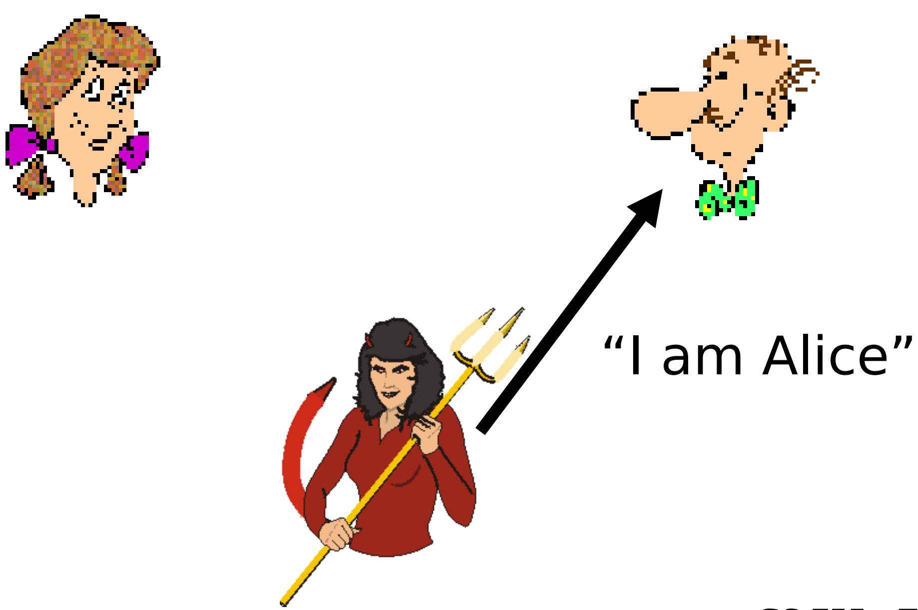
Protocol ap1.0: Alice says “I am Alice”

Failure scenario??

Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”



in a network,
Bob can not “see” Alice, so
Trudy simply declares
herself to be Alice

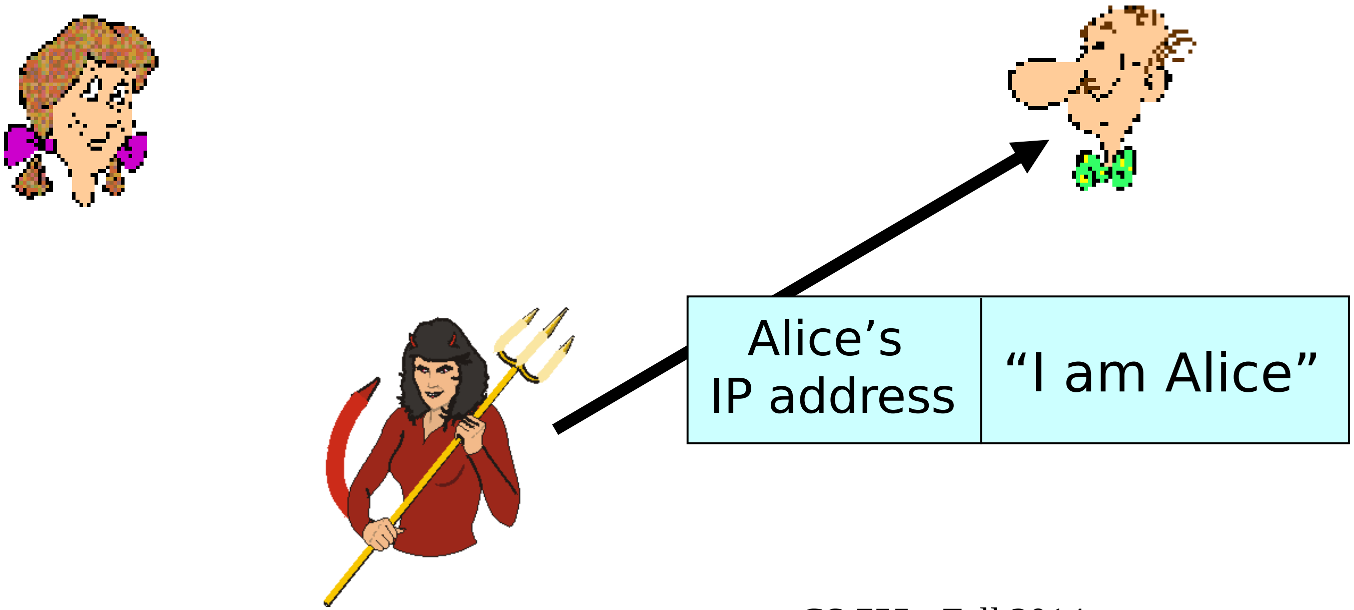
Authentication: another try

<u>Protocol ap2.0:</u> Alice says “I am Alice” in an IP packet containing her source IP address

Failure scenario??

Authentication: another try

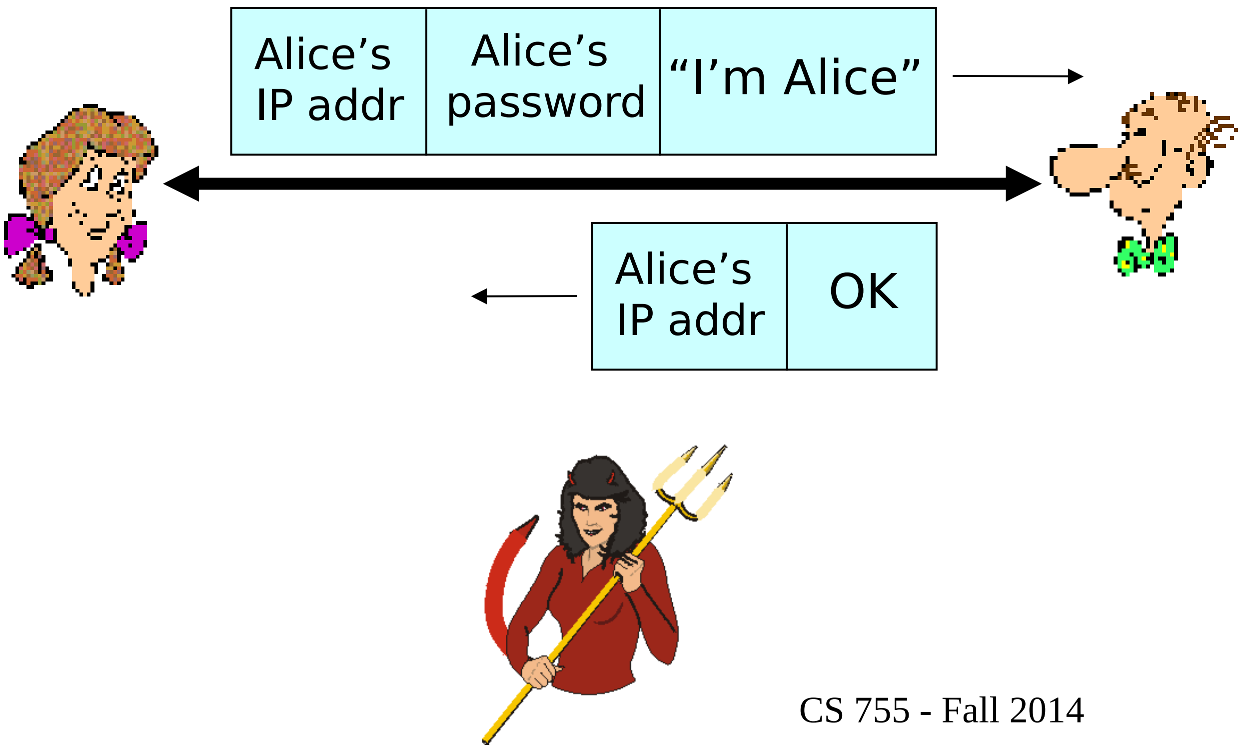
Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address



Trudy can create a packet “spoofing” Alice’s address

Authentication: another try

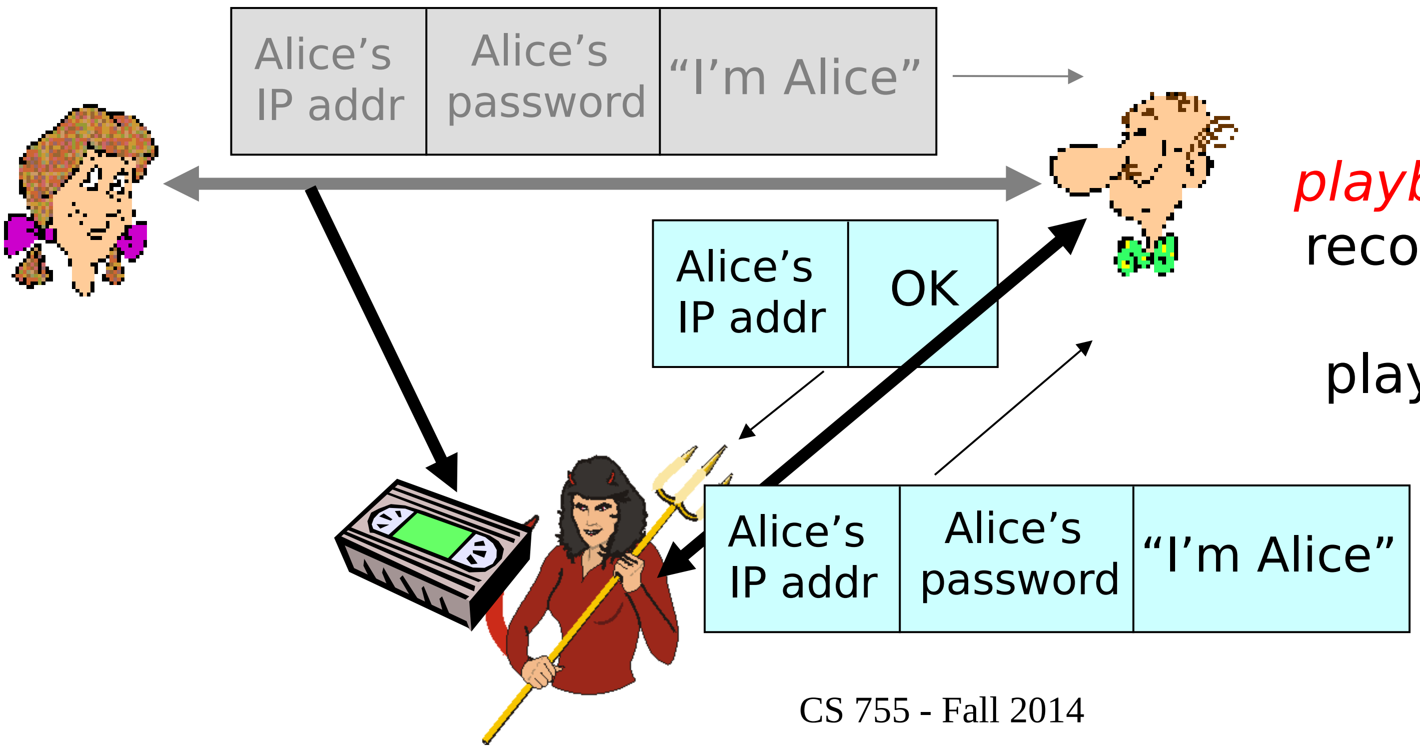
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: another try

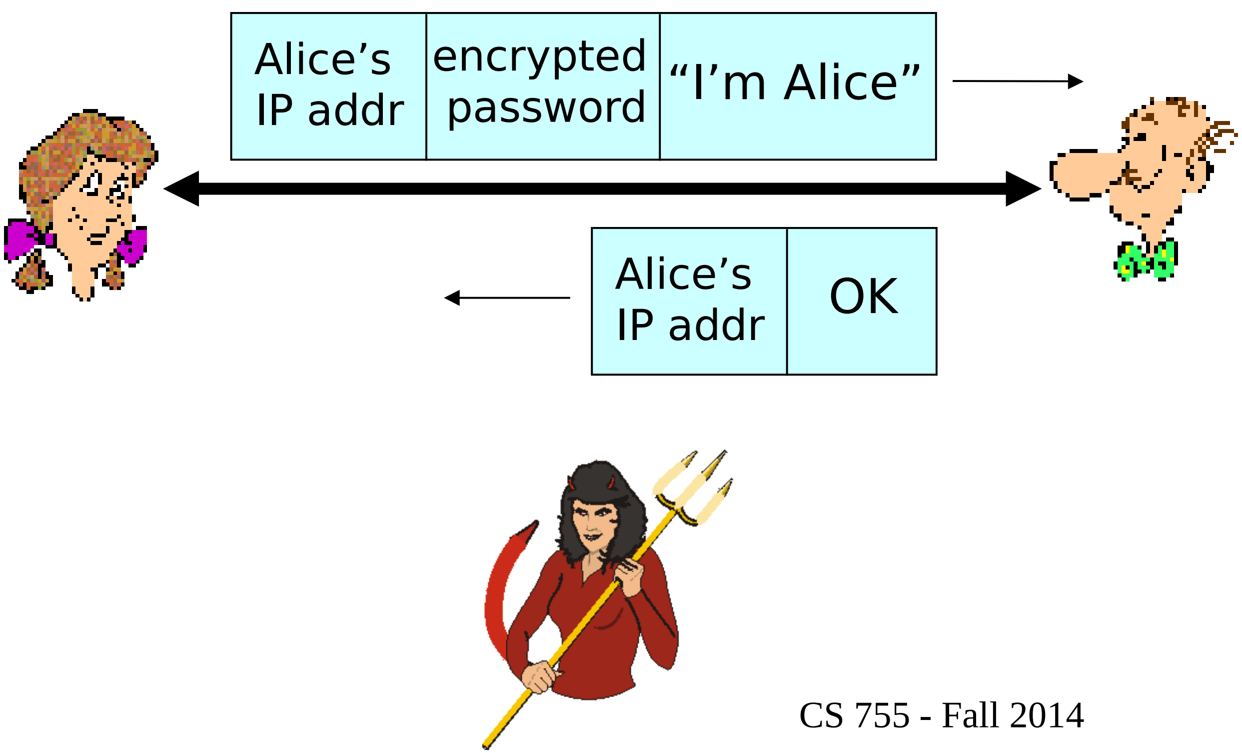
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



playback attack: Trudy records Alice's packet and later plays it back to Bob

Authentication: yet another try

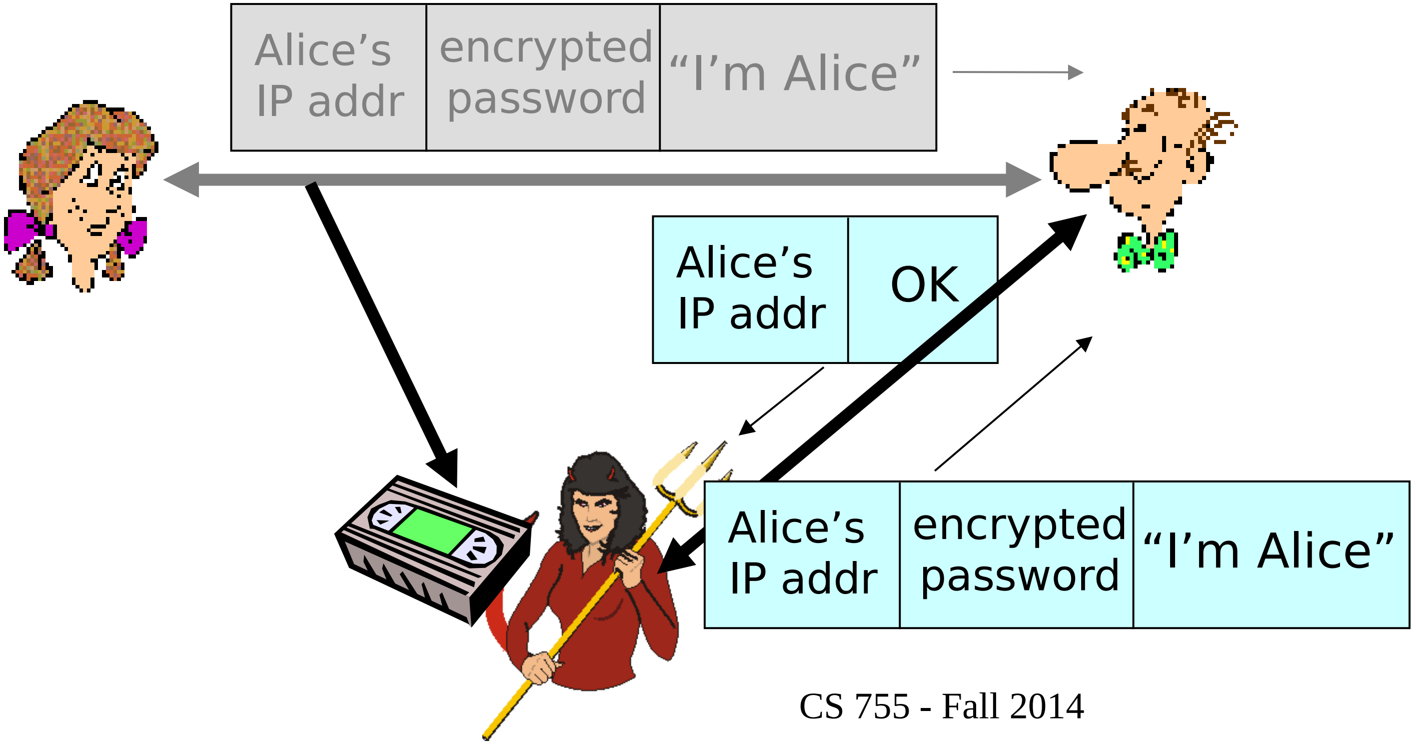
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: another try

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



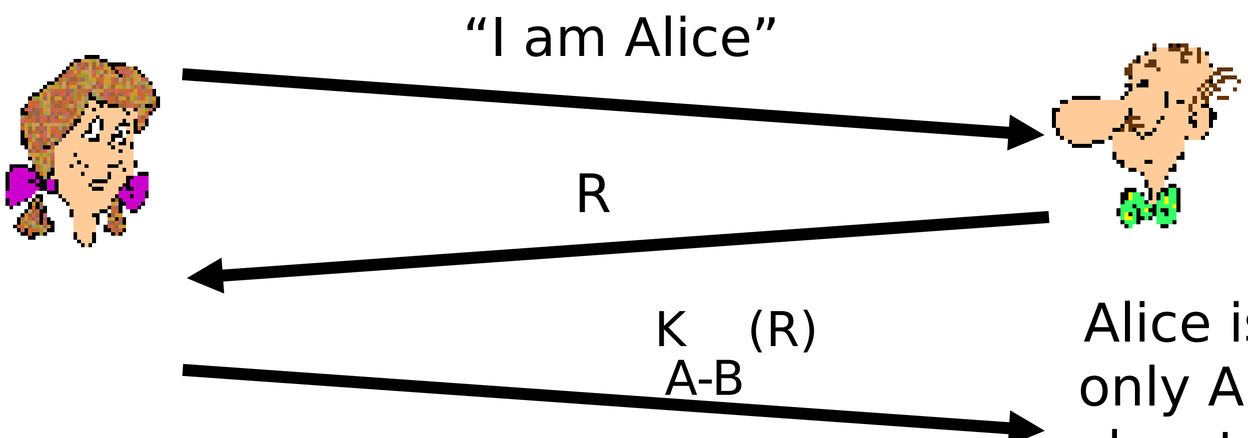
record and playback still works!

Authentication: yet another try

Goal: avoid playback attack

Nonce: number (R) used only *once –in-a-lifetime*

ap4.0: to prove Alice "live", Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key



Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

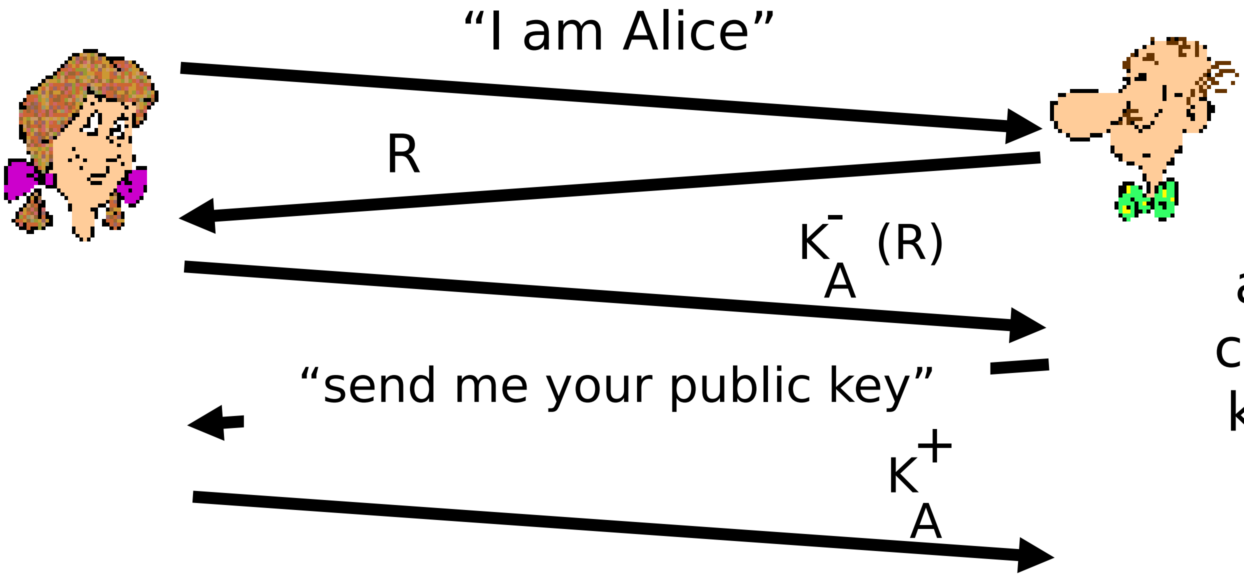
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



Bob computes

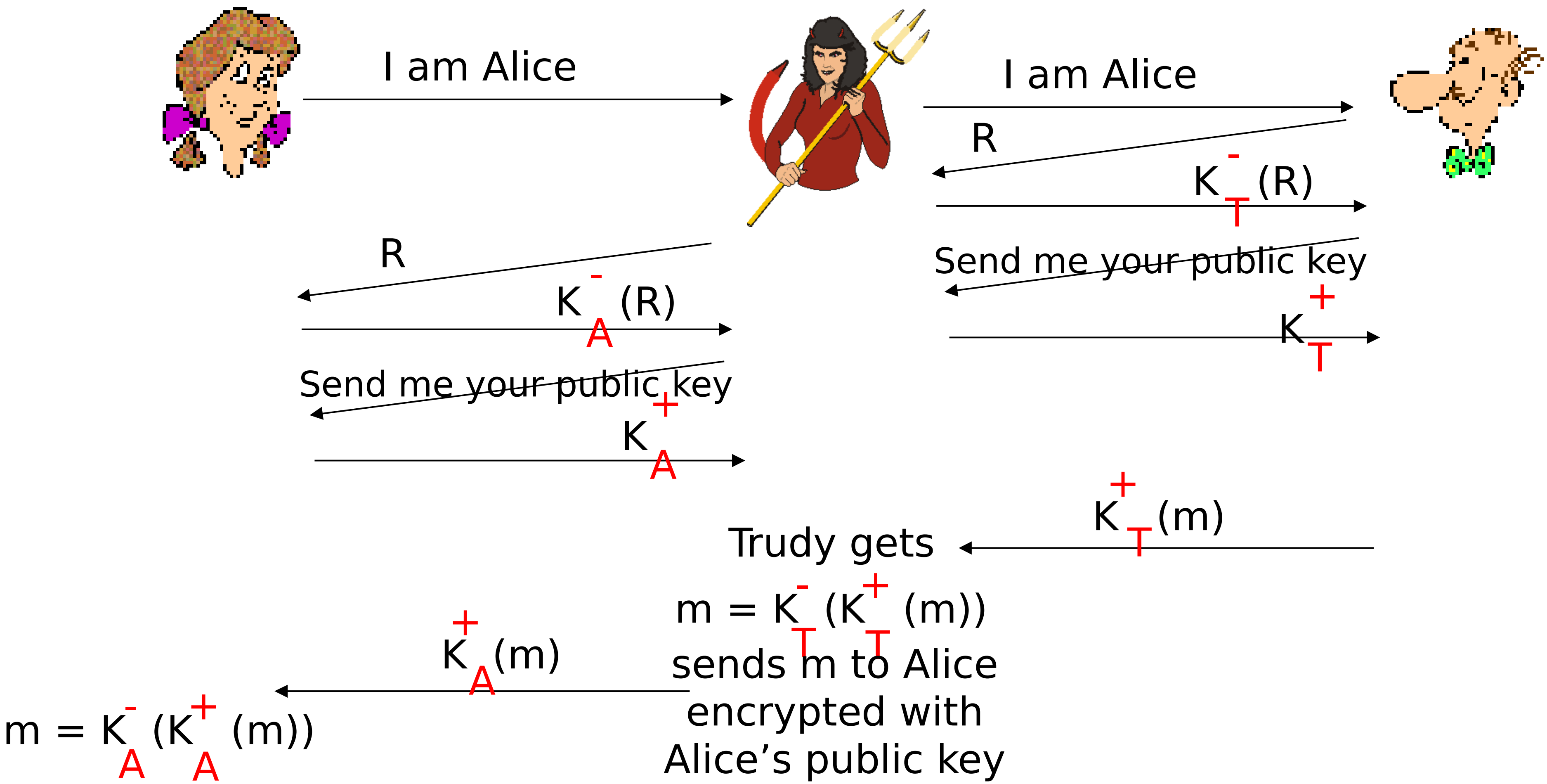
$K_A^+(K_A^-(R)) = R$

and knows only Alice could have the private key, that encrypted R such that

$K_A^+(K_A^-(R)) = R$

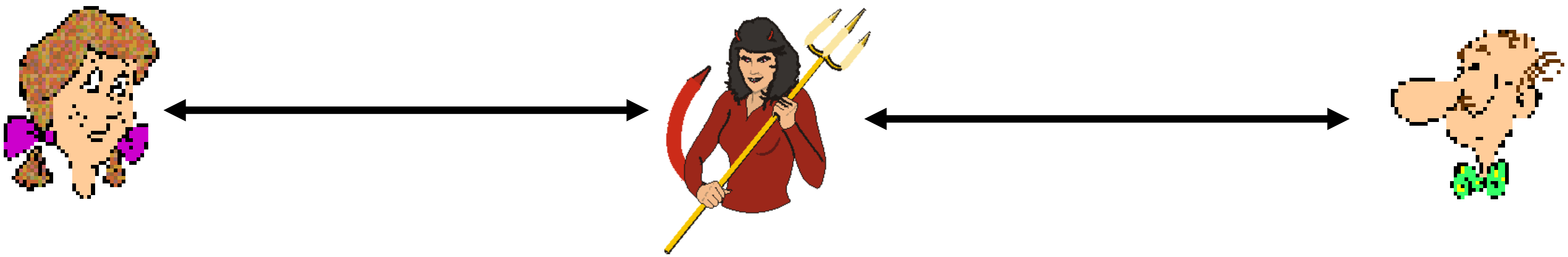
ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



Difficult to detect:

- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation)
- problem is that Trudy receives all messages as well!

Public Key Certification

public key problem:

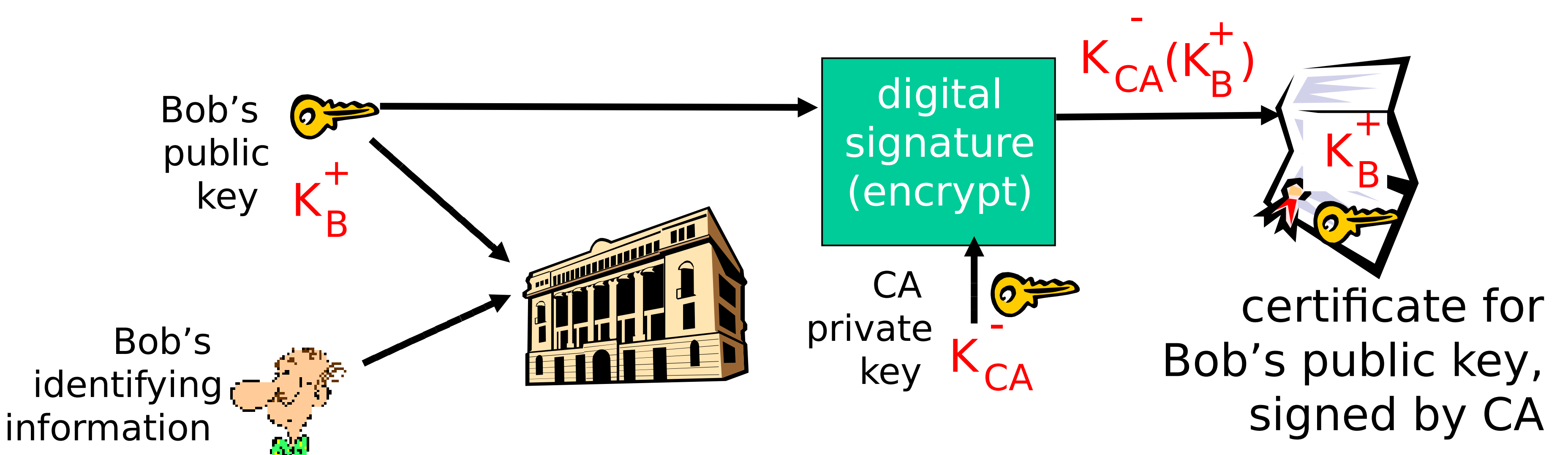
- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she *know* it is Bob's public key, not Trudy's?

solution:

- trusted certification authority (CA)

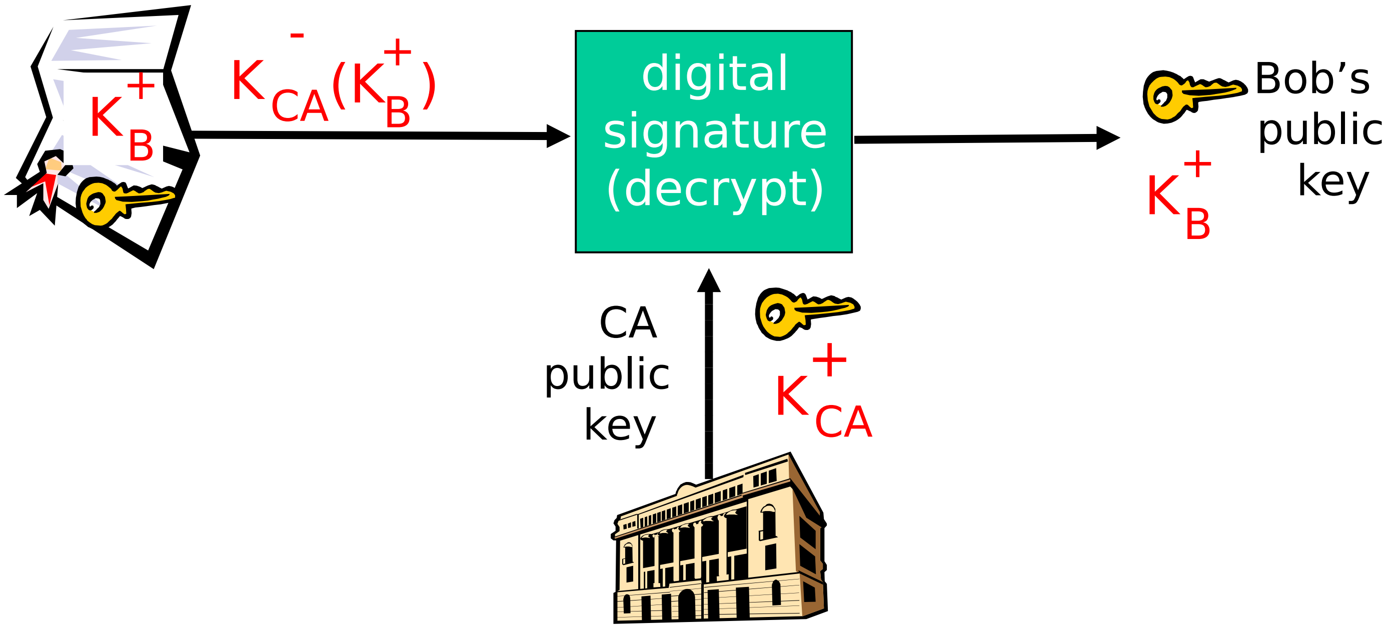
Certification Authorities

- Certification Authority (CA): binds public key to particular entity, E.
- E registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA: CA says "This is E's public key."



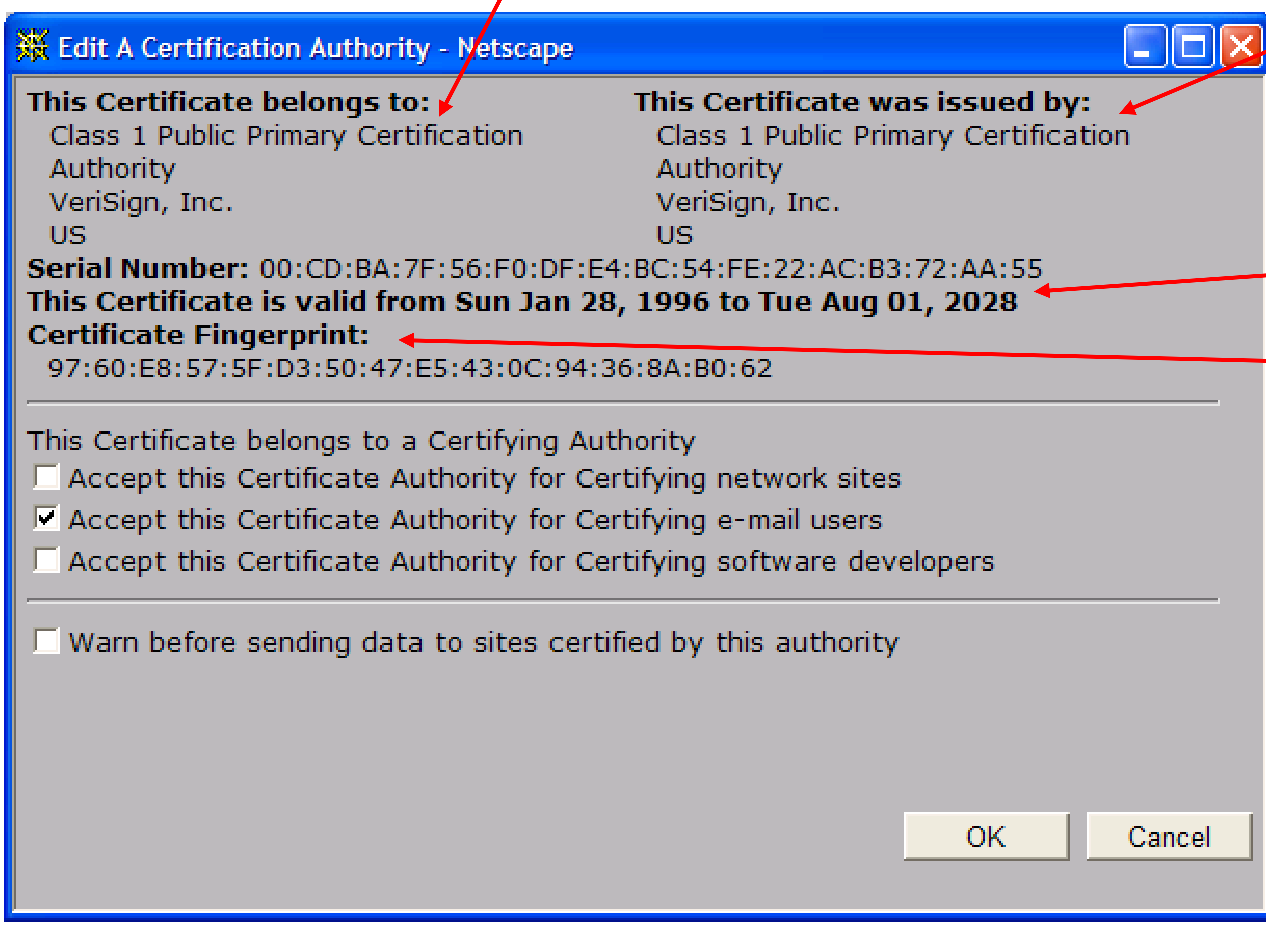
Certification Authorities

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



A certificate contains:

- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)



- info about certificate issuer
- valid dates
- digital signature by issuer