

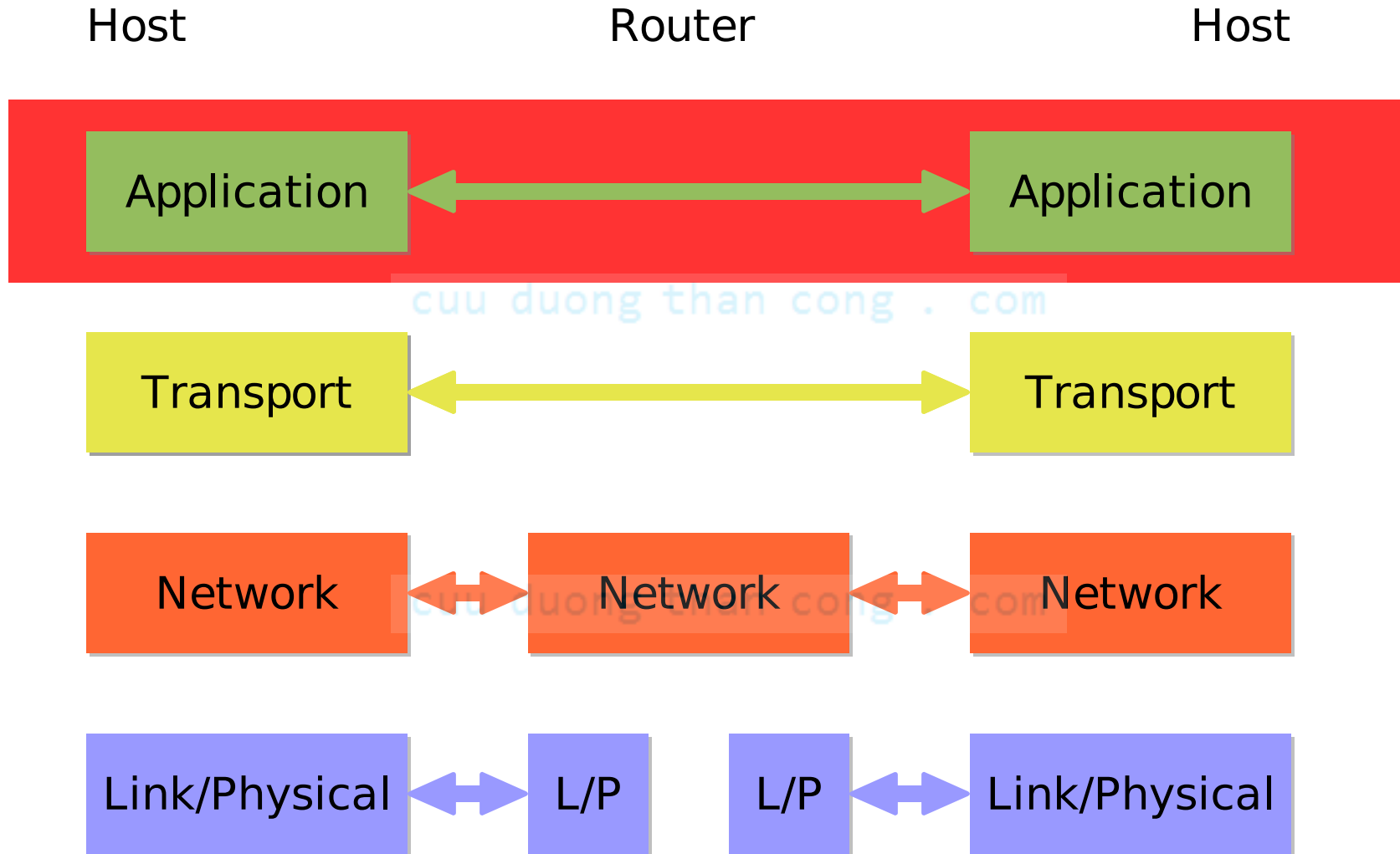
# Application Layer Protocols 2

# Today's objectives

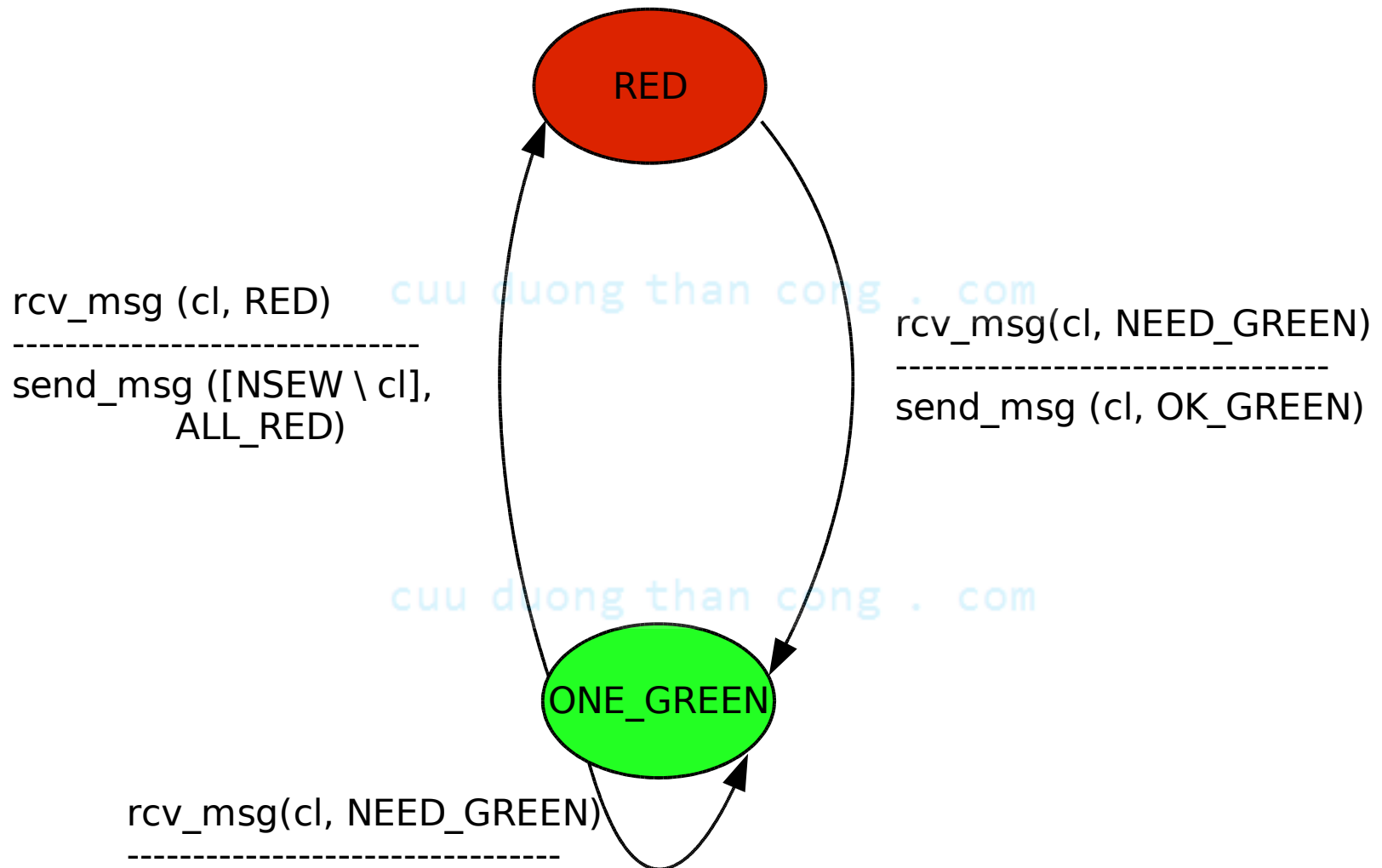
- More on FSMs
  - aggregate state machine
  - checking properties
- More application-layer protocols
  - remote login [cuu duong than cong . com](http://cuuduongthancong.com)
  - quick introduction to security
  - ssh: secure shell
  - peer-to-peer networks

[cuu duong than cong . com](http://cuuduongthancong.com)

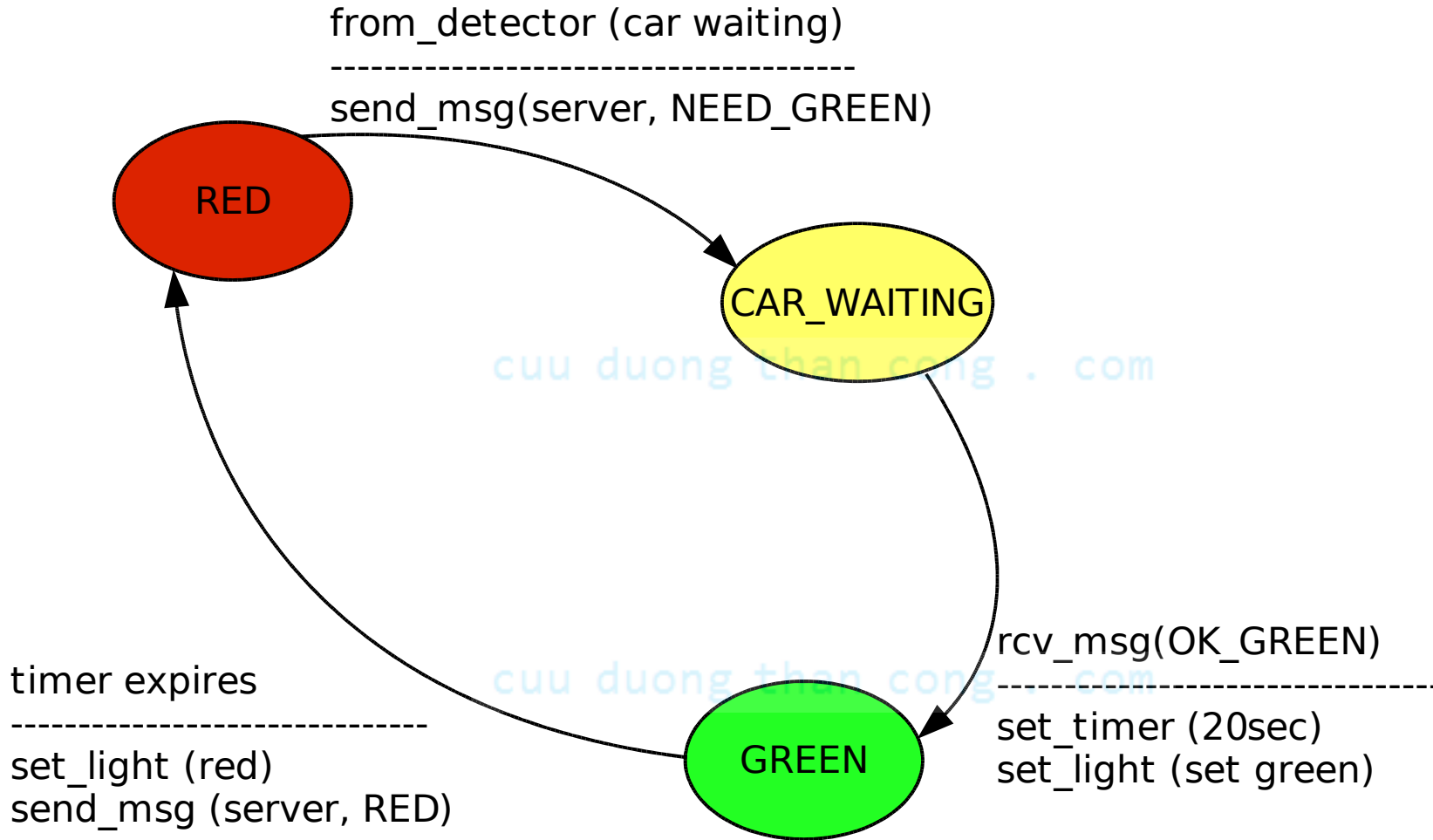
# Application layer



# Intersection server FSM #1



# Intersection client FSM #1



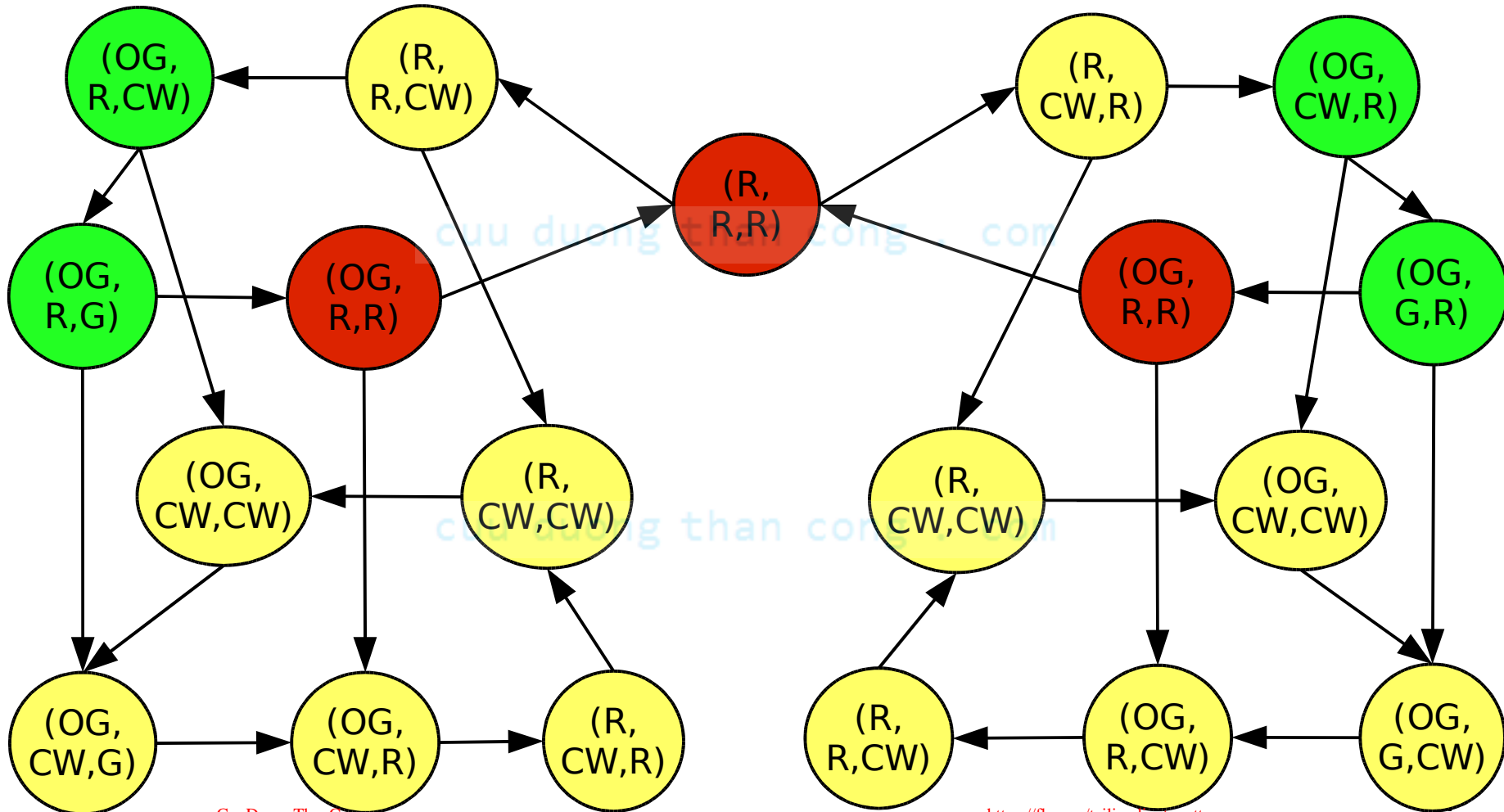
# Joint FSM

- Joint FSM:
  - An FSM that captures the evolution of the system as a whole
  - Each state  $S$  of the joint FSM corresponds to the set of states each component FSM  $1, \dots, n$  is in:
    - $S = (S_1, S_2, \dots, S_n)$
  - Every transition  $(S_1, S_2, \dots, S_n) \rightarrow (S_1', S_2', \dots, S_n')$  corresponds to one or several transitions in component FSMs
    - If several transitions, they have to be concurrent

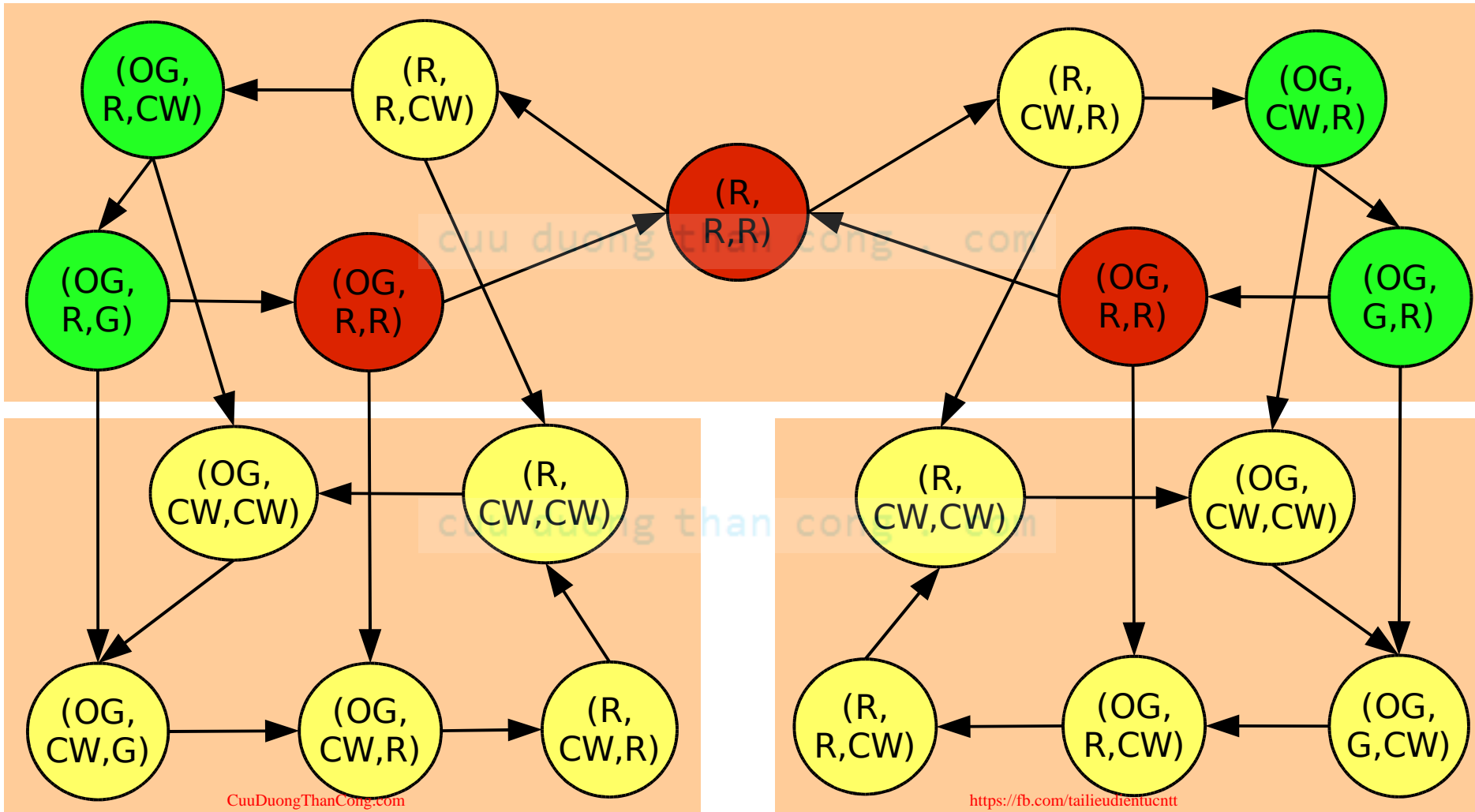
cuu duong than cong . com

# Joint FSM for intersection protocol #1

- State = (server, state\_client1, state\_client2)



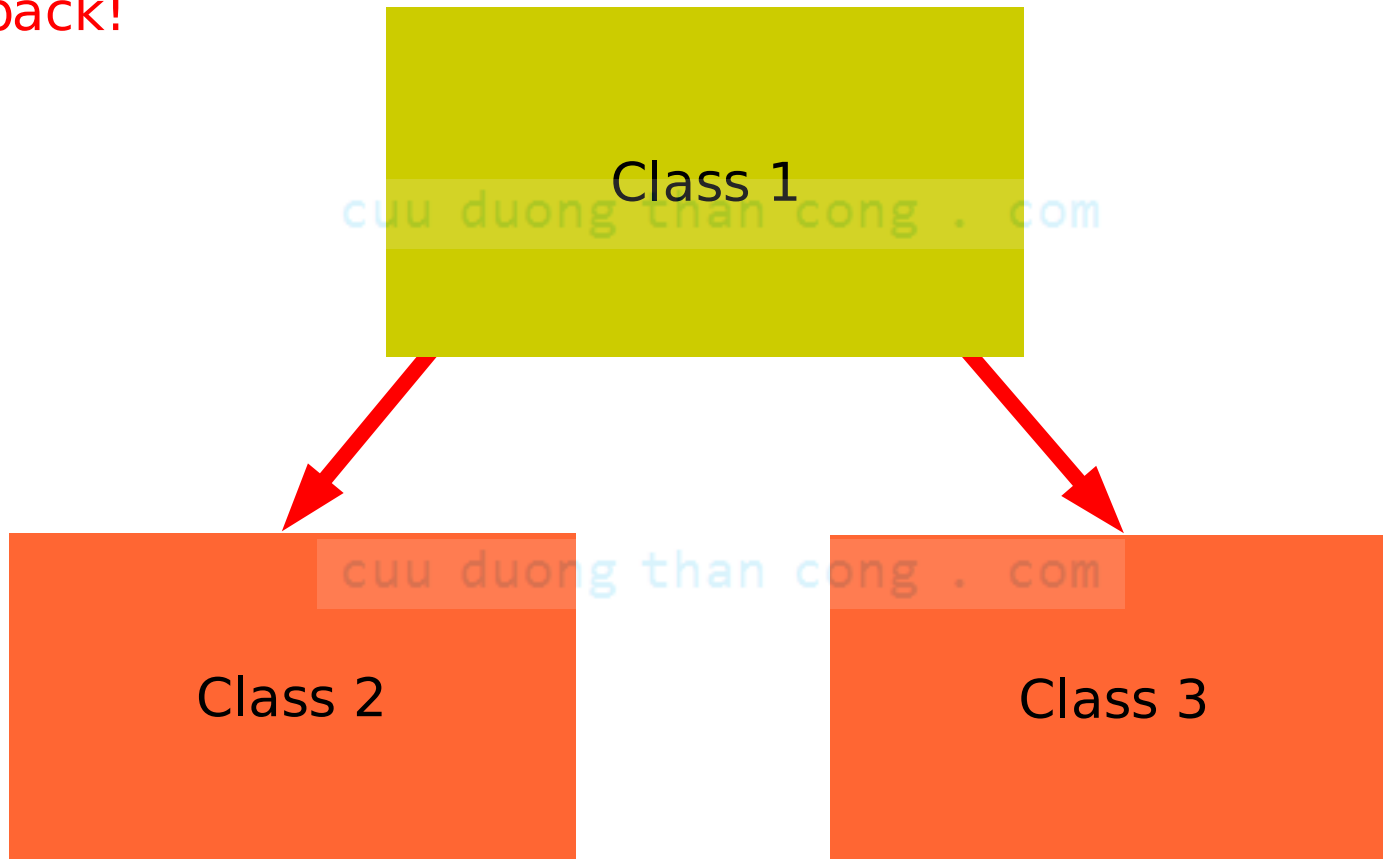
# Joint FSM for intersection protocol #1



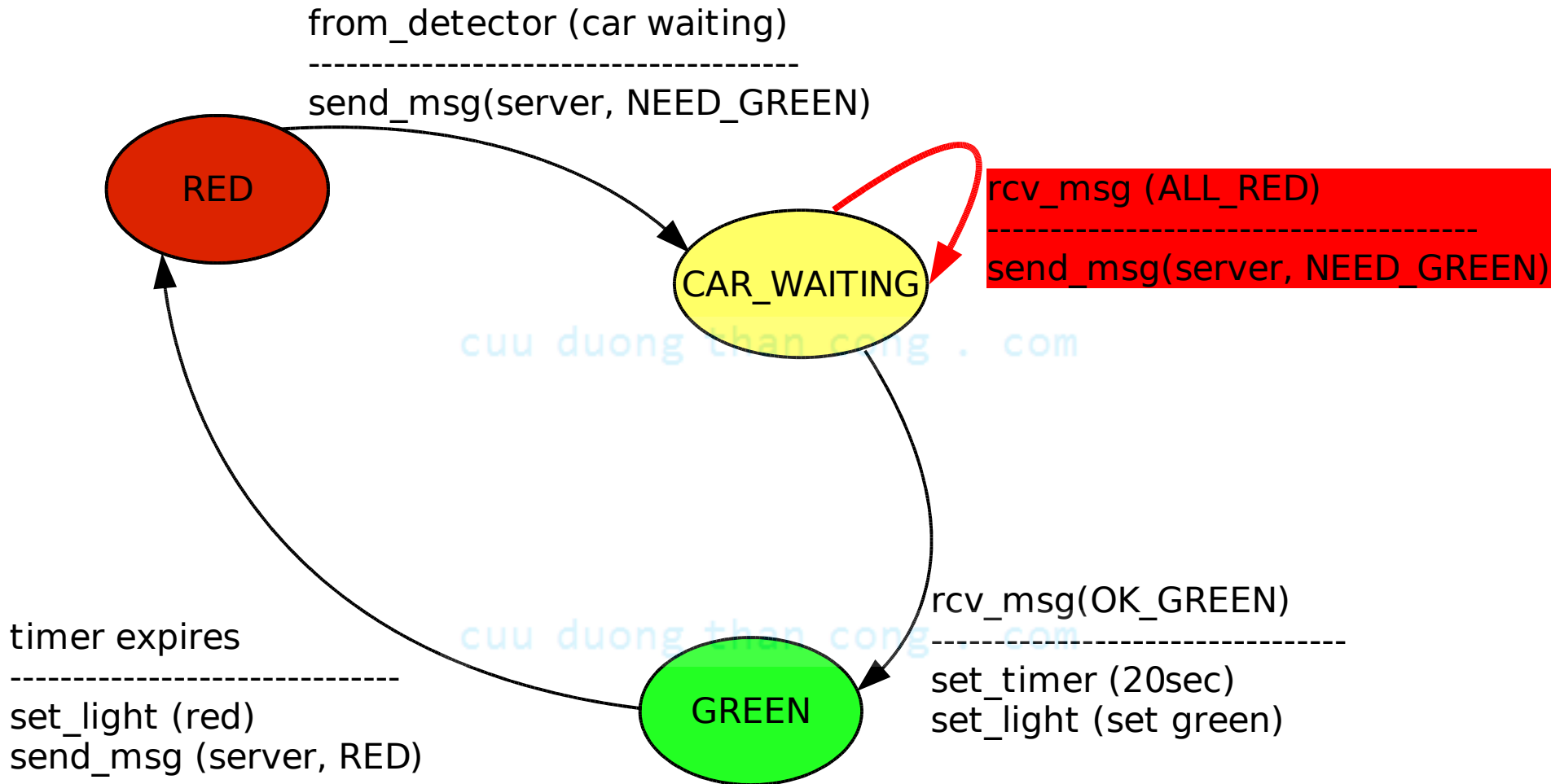


# Joint FSM for intersection protocol #1

- Classes of states
  - We can leave class 1 to either class 2 or 3 and never come back!

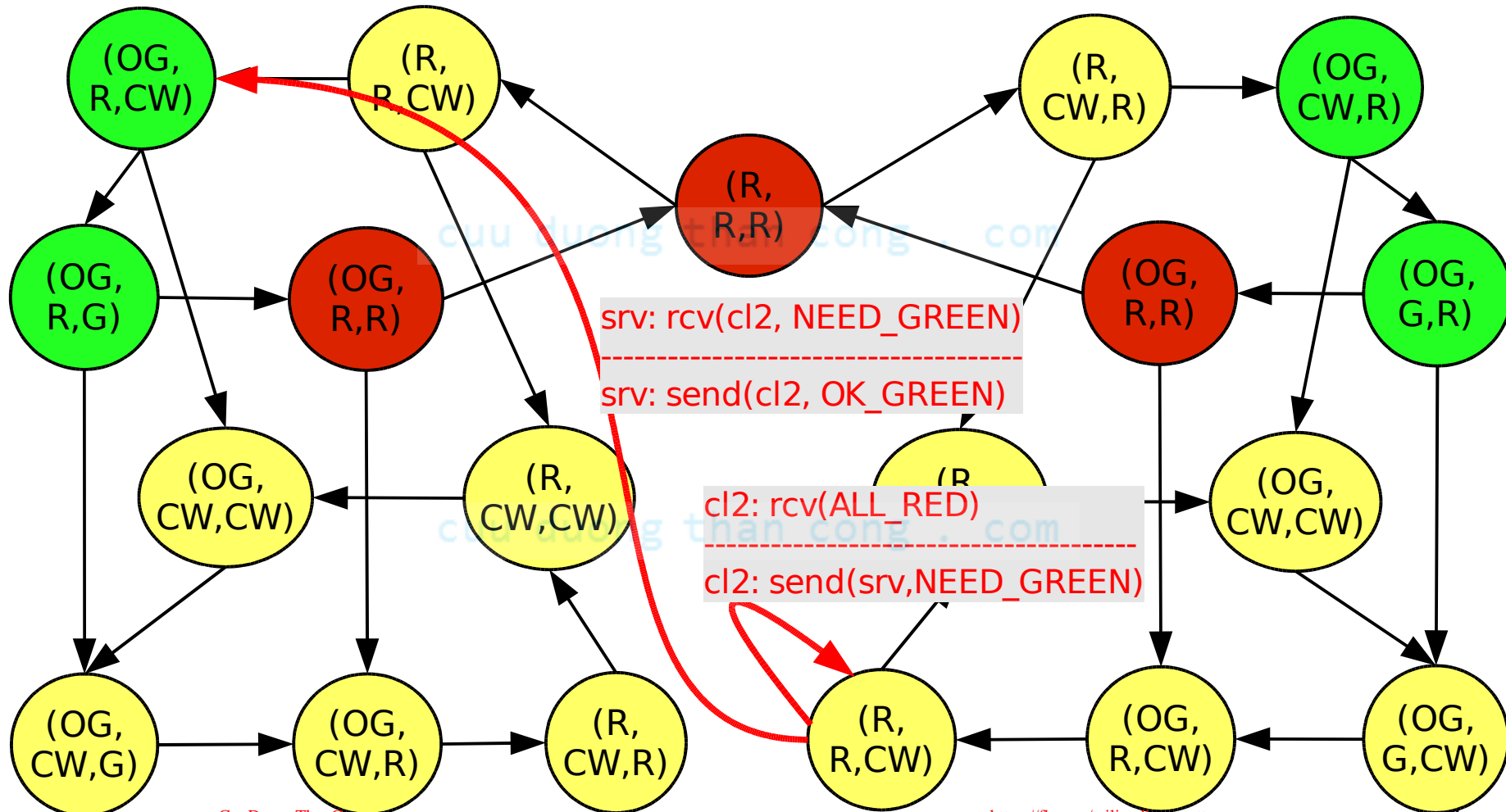


# Intersection client FSM #2, no blocking



# Joint FSM for intersection protocol #2

- State = (server, state\_client1, state\_client2)



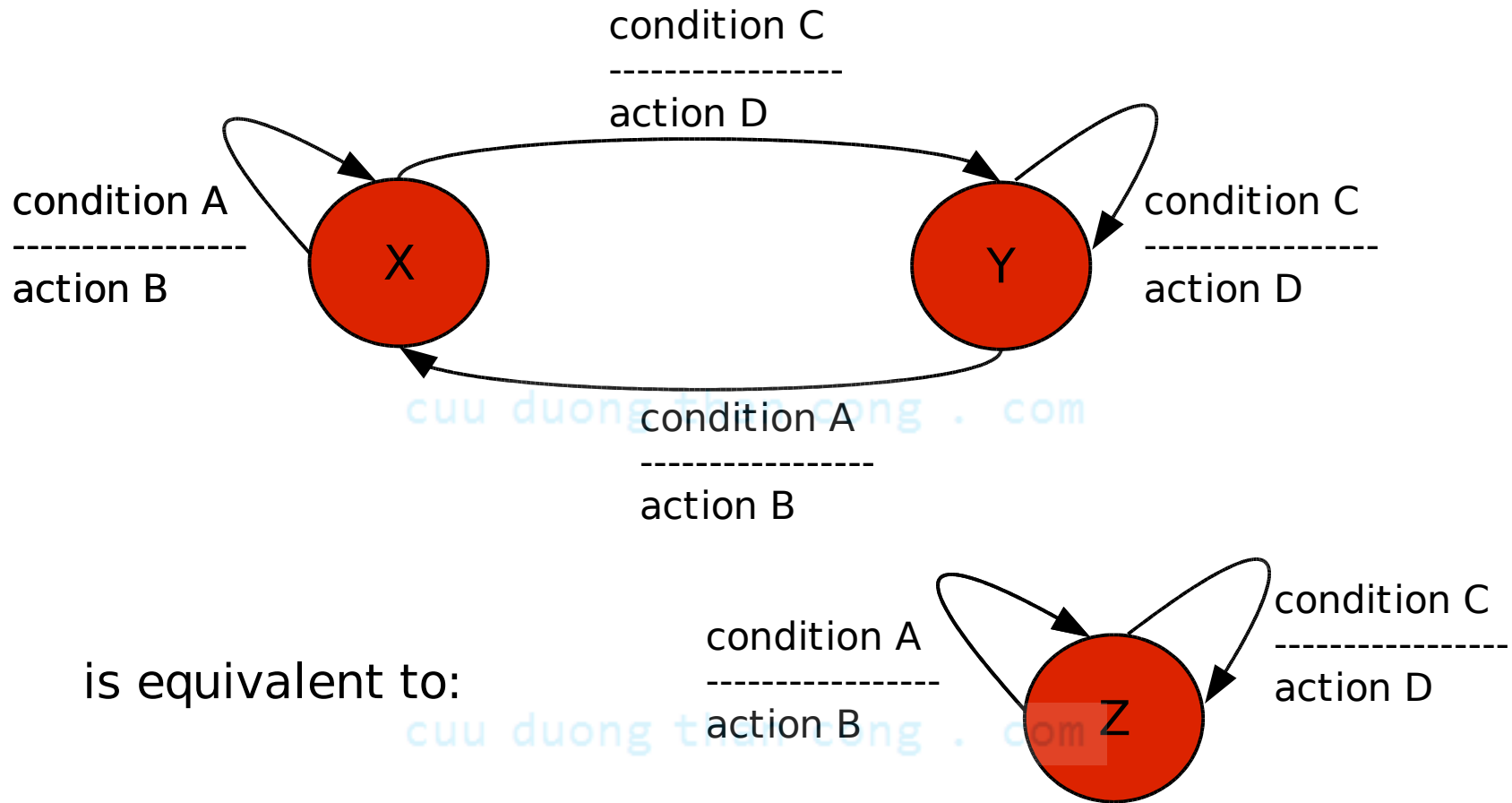
# Joint FSM for intersection protocol #2

- Single class of states
  - We can go from anywhere to anywhere, given the right “input”



Class 1

# Another example of FSM-based analysis



- All that matters is input->output
  - Can we get rid of internal states?
  - Automatic simplification

# FSM: summary

- Individual FSM:

- Describes individual protocol entity

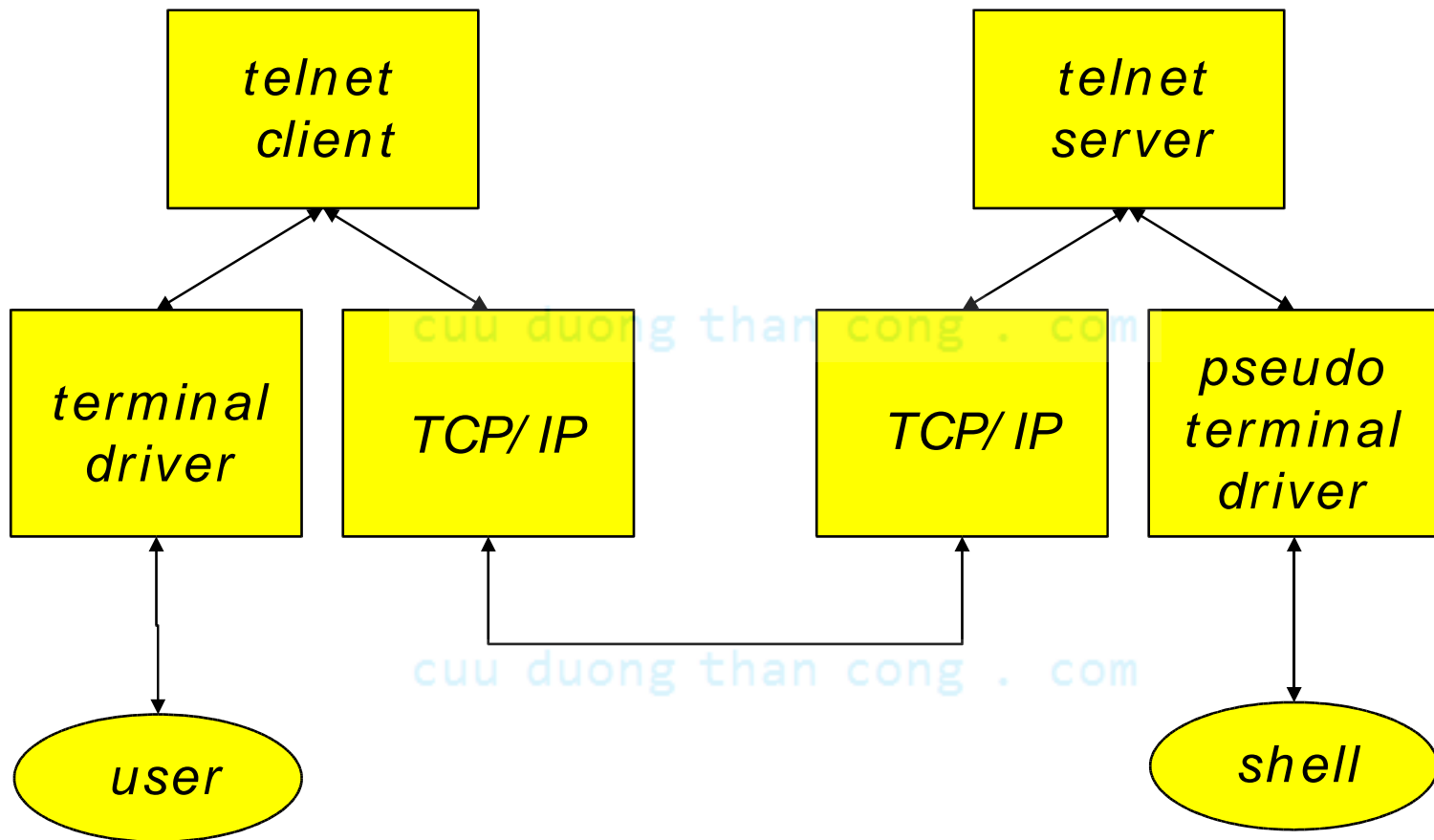
- Joint FSM:

- Can be generated automatically
  - Describes communicating set of entities
  - Protocol verification: check properties such as “no absorbing classes of states”; if such a class exists, what is sequence of events that leads there, etc.
  - Advantage:
    - Allows mathematical proof of these properties
    - Software tools, code generators
  - Disadvantage:
    - Computational complexity quickly becomes large as FSMs become more complex -> number of joint states explodes

# Remote session: telnet, rlogin, ssh

- Work on a remote system
- Applications
  - telnet
  - *R-commands* of Unix
    - rlogin, rsh, rcp
  - ssh
- Principles
  - send characters typed on the keyboard to the remote shell
  - receive characters from the remote shell and display

# rlogin, telnet, ssh





# rlogin

- Connection between UNIX systems
  - port 513
- Authentication
  - password (send in clear)
  - `.rhosts` file
    - host name
    - user name
- Typed character
  - echo sent by the remote system

# telnet

- Connection between any system
  - NVT (*Network Virtual Terminal*)
    - common denominator between different systems
    - NVT ASCII
      - 7 bits, end of line: CR, LF
      - also used by FTP, SMTP, finger, whois, HTTP
- Port 23
- Options
  - character mode, line mode
- Escape - interpreted by the client
  - Control-]

# SSH: what is network security?

**Confidentiality:** only sender, intended receiver should “understand” message contents

- sender encrypts message
- receiver decrypts message

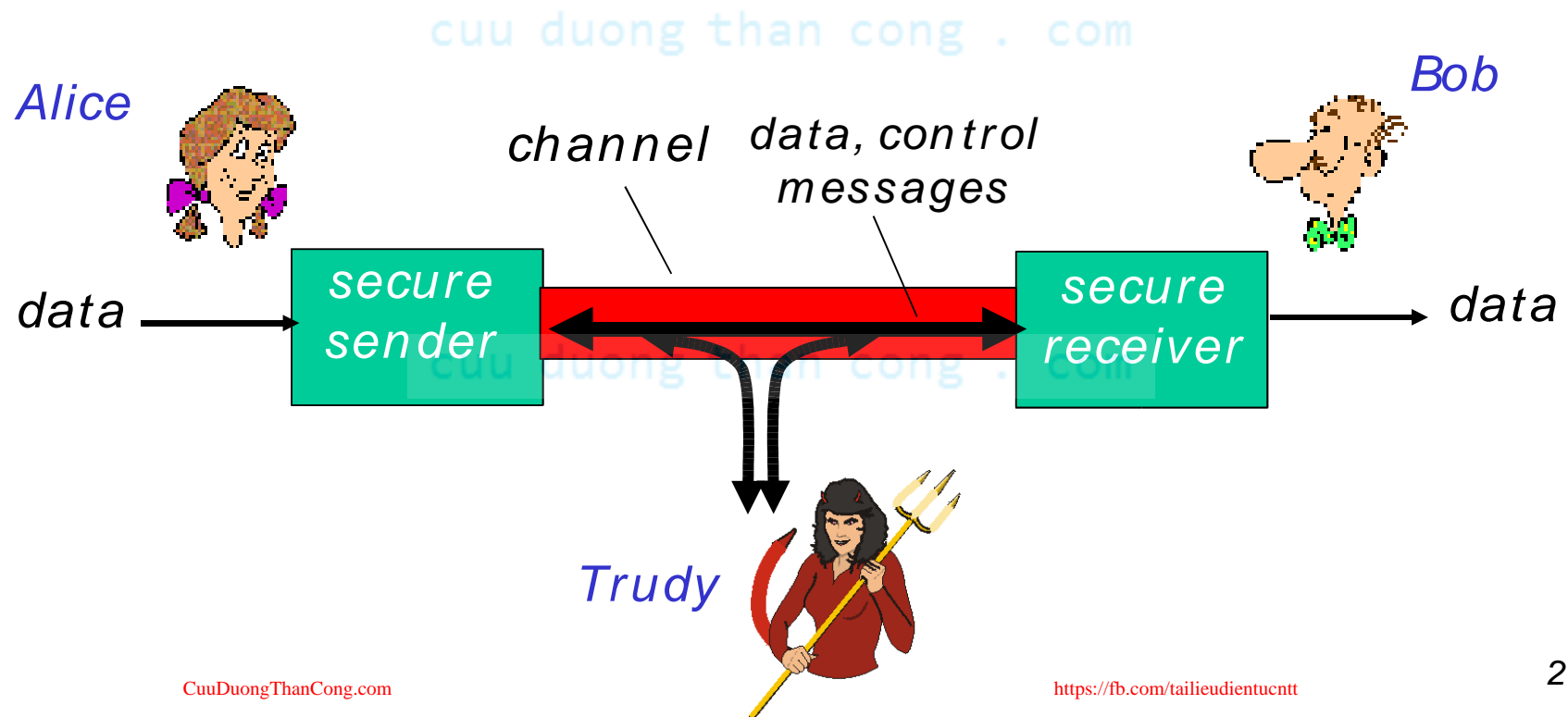
**Authentication:** sender, receiver want to confirm identity of each other

**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 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 [cuu.duong.than.cong.com](http://cuu.duong.than.cong.com)
- routers exchanging routing table updates
- other examples?

[cuu.duong.than.cong.com](http://cuu.duong.than.cong.com)

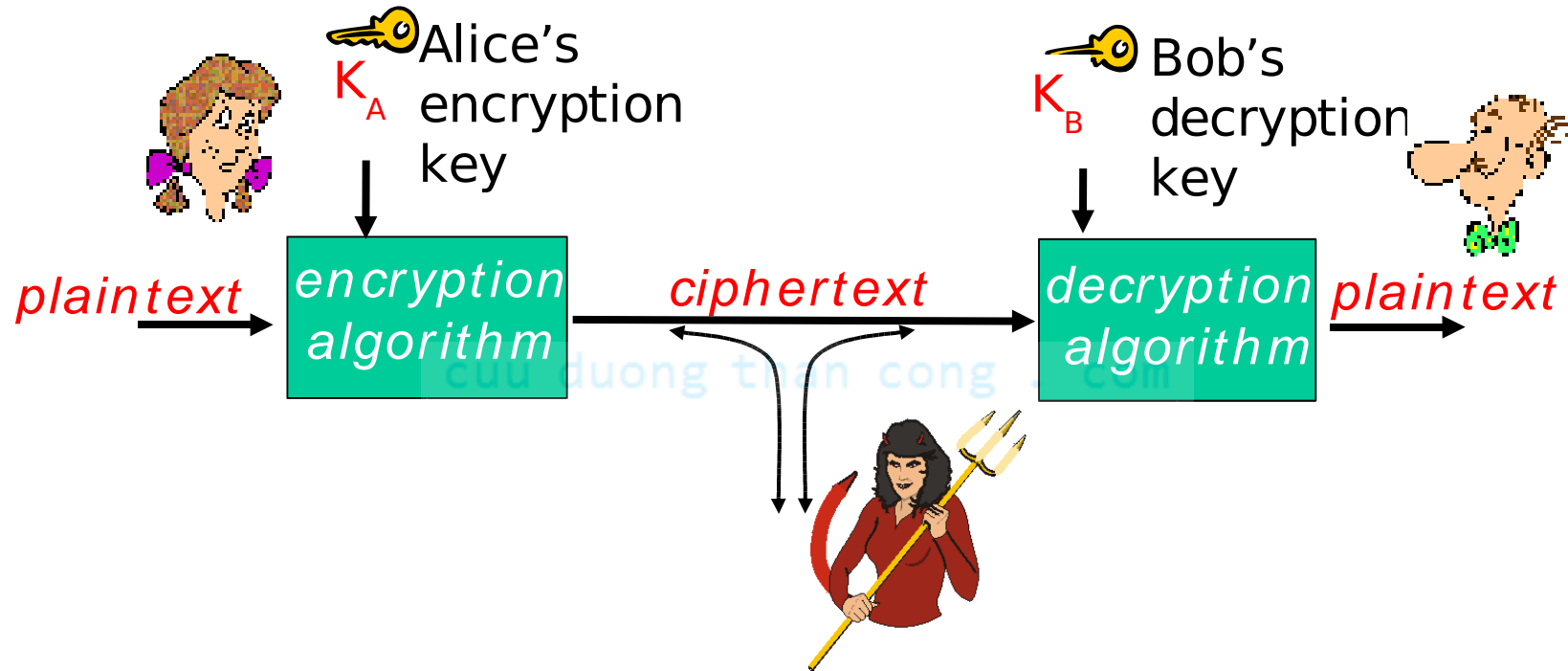
# There are bad guys (and girls) out there!

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)

# Symmetric key cryptography

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

**plaintext:**    abcdefghijklmnopqrstuvwxyz



**ciphertext:**   mnbvcxzasdfghjklpoiuytrewq

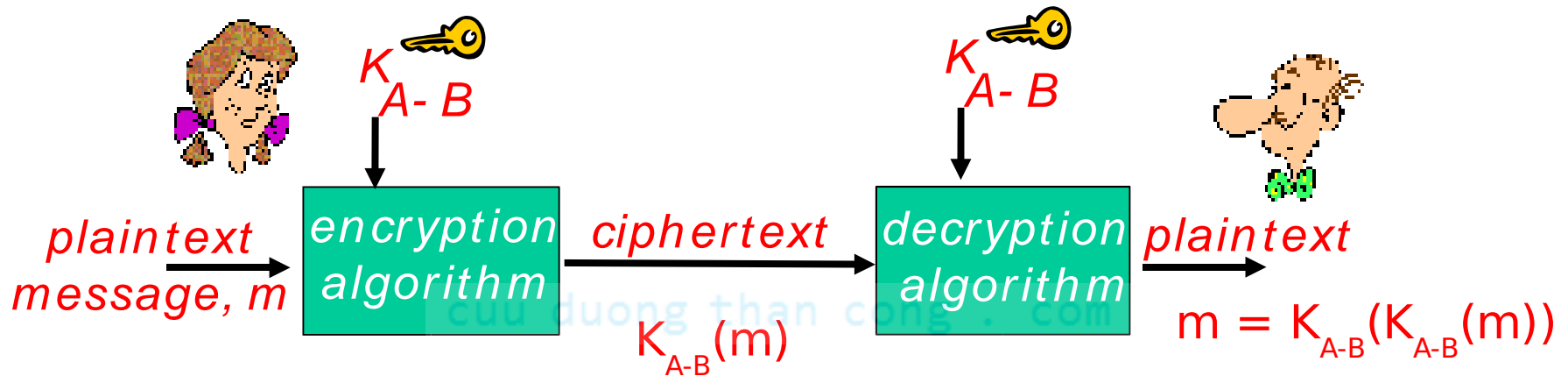
E.g.: Plaintext: bob. i love you. alice  
ciphertext: nkn. s gktc wky. mgsbc

Q: How hard to break this simple cipher?:

brute force (how hard?)  
other?



# Symmetric key cryptography



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

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

# Public key cryptography

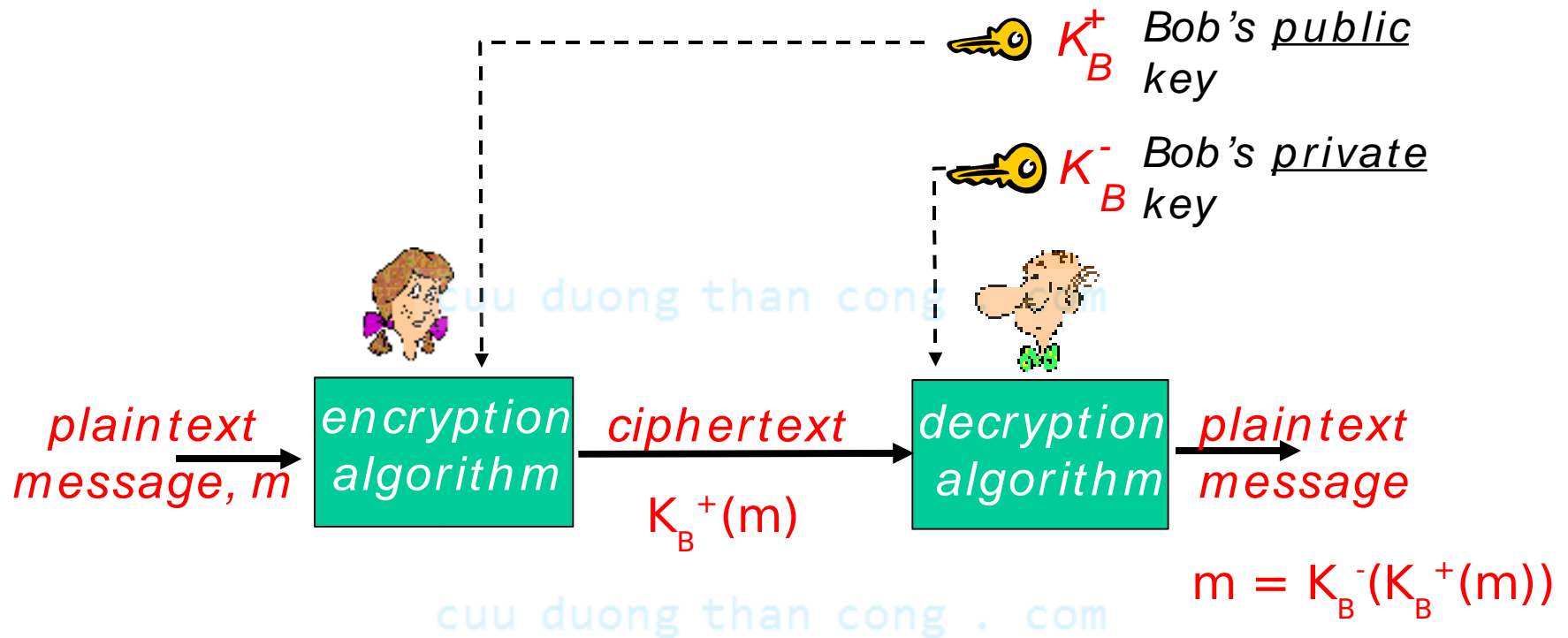
- Symmetric key crypto
  - requires sender, receiver know shared secret key
  - Q: how to agree on key in first place (particularly if never “met”)?

- 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  $K_B^+( )$  and  $K_B^-( )$  such that

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

- ② given public key  $K_B^+$ , it should be impossible to compute private key  $K_B^-$

cuu duong than cong . com

**RSA:** Rivest, Shamir, Adelson algorithm

# RSA: Choosing keys

1. Choose two large prime numbers  $p, q$ .  
(e.g., 1024 bits each)
2. Compute  $n = pq$ ,  $z = (p-1)(q-1)$
3. Choose  $e$  (with  $e < n$ ) that has no common factors with  $z$ . ( $e, z$  are “relatively prime”).
4. Choose  $d$  such that  $ed - 1$  is exactly divisible by  $z$ .  
(in other words:  $ed \bmod z = 1$  ).
5. Public key is  $(n, e)$ . Private key is  $(n, d)$ .

$K_B^+$

$K_B^-$

# RSA: Encryption, decryption

0. Given  $(n,e)$  and  $(n,d)$  as computed above

1. To encrypt bit pattern,  $m$ , compute

$$c = m^e \bmod n \quad (\text{i.e., remainder when } m^e \text{ is divided by } n)$$

cuu duong than cong . com

2. To decrypt received bit pattern,  $c$ , compute

$$m = c^d \bmod n \quad (\text{i.e., remainder when } c^d \text{ is divided by } n)$$

Magic  
happens!

$$m = \underbrace{(m^e \bmod n)}_c^d \bmod n$$

# RSA: another important property

*The following property will be **very** useful later:*

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

*use public key  
first, followed  
by private key*

*use private  
key first,  
followed by  
public key*

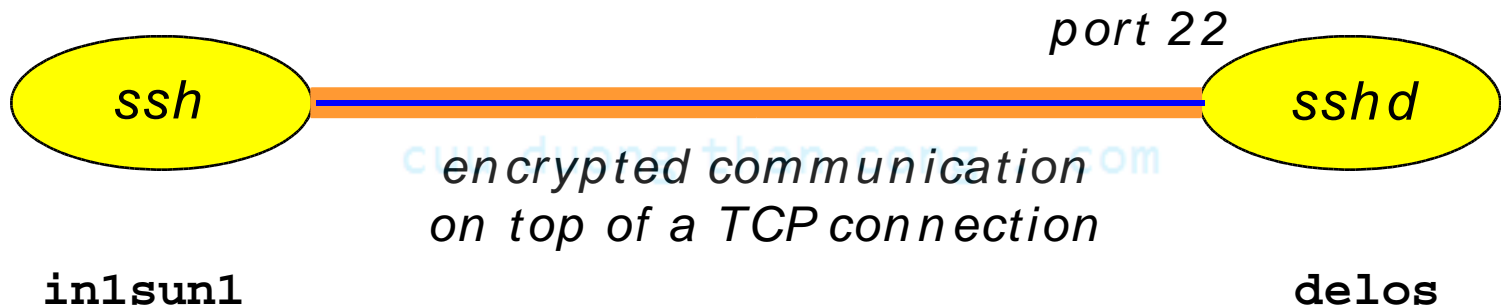
*Result is the  
same!*

# Back to SSH

- Secure remote session
  - encrypted connection, secret per session key
  - port 22
- Authentication
  - encrypted password
  - RSA public key
    - user puts its public key on the remote host
    - random challenge signed with the public key
    - only the user can decrypt it with its secret key
- Tunnels and port redirection
  - redirect the connections of other applications (e-mail)
  - automatic redirection of X connections

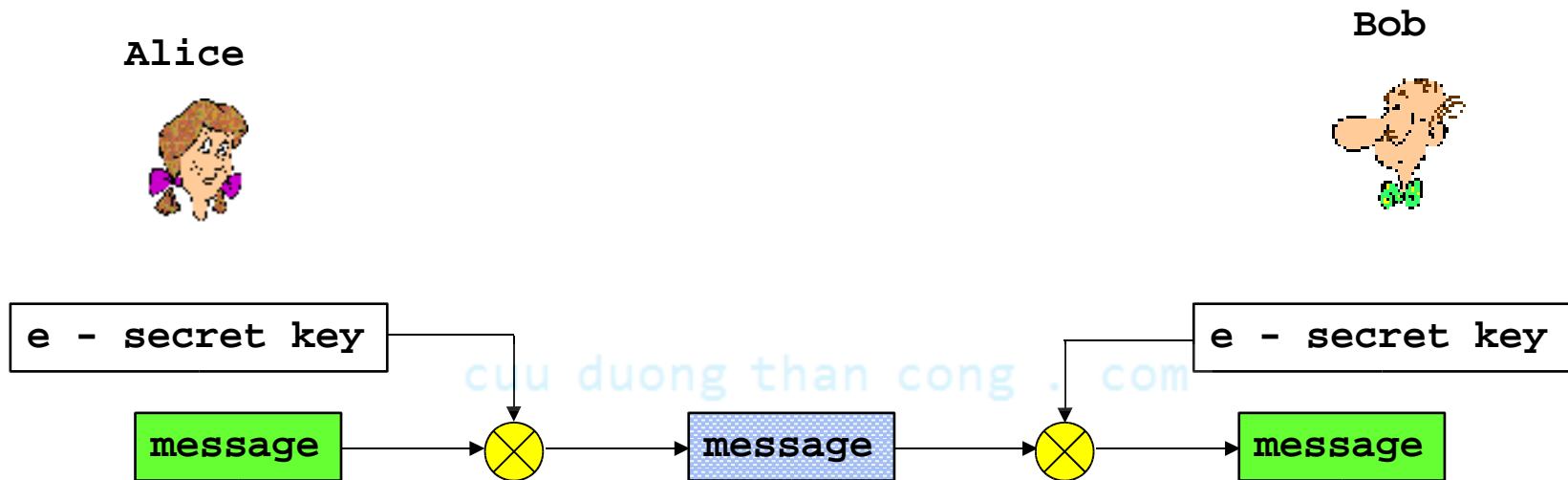


# Basic ssh connection



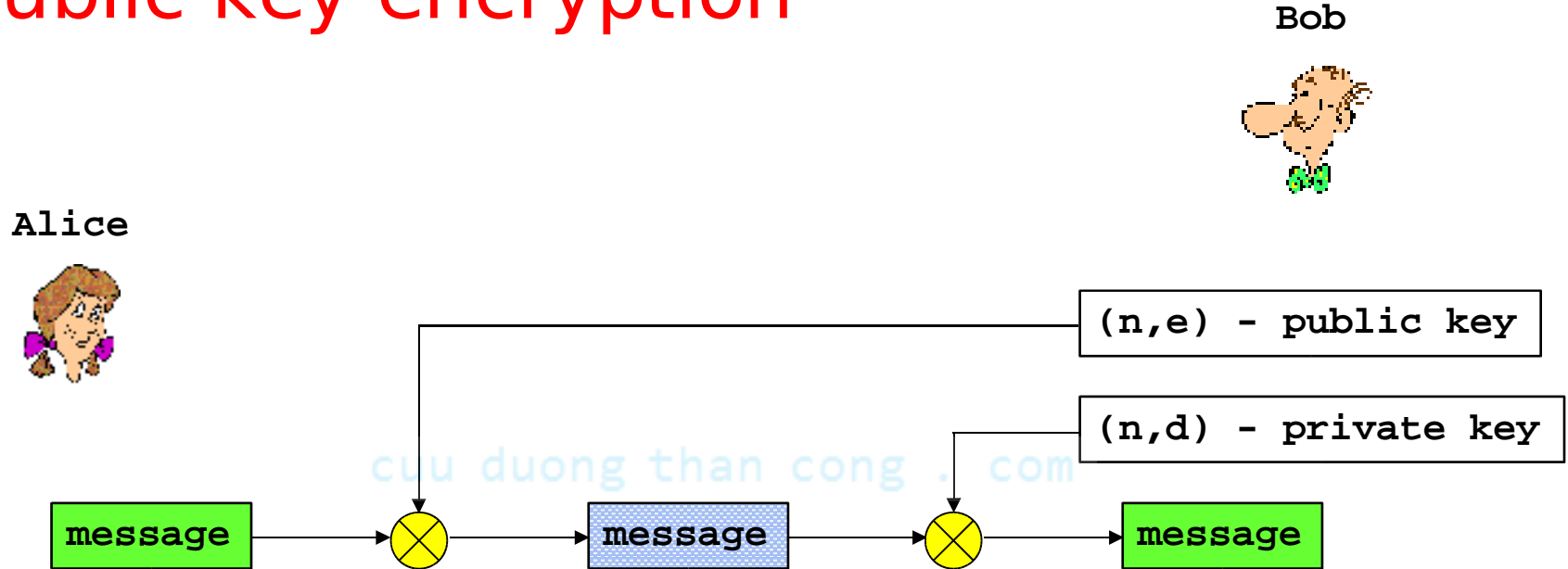
```
in1sun1% ssh delos.imag.fr
```

# Symmetric key encryption



- Secret key encryption (DES, 3DES,...)
  - encrypted message  $c = f(e, m)$
  - decrypted message  $m = f^{-1}(e, c)$
- Must exchange the key
- Efficient encryption

# Public key encryption



- RSA encryption

- encrypted message  $c = (m^e \bmod n)$
- decrypted message  $m = (c^d \bmod n)$

- Key property

- $(m^e)^d \bmod n = m$

- Slow

# Public key authentication

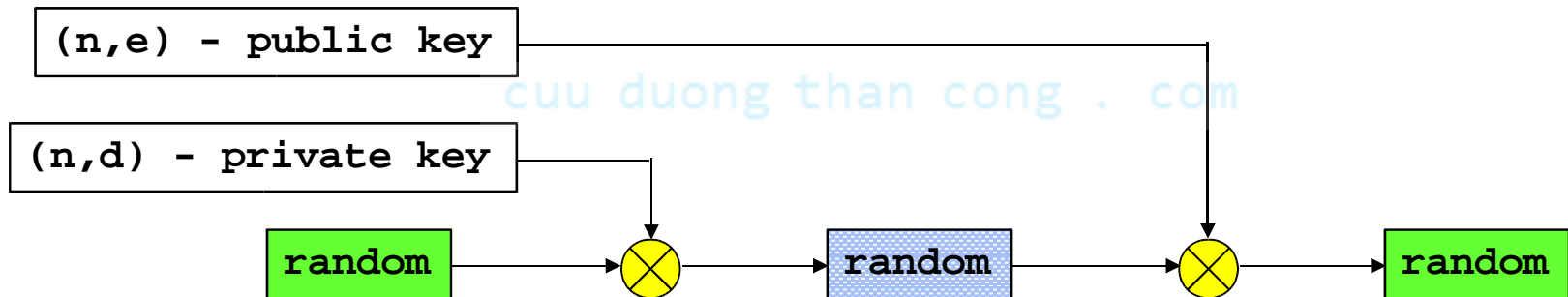
Alice



I'm Alice

random

Bob



- Authentication

- random challenge (nonce), used only once

- Bob verifies

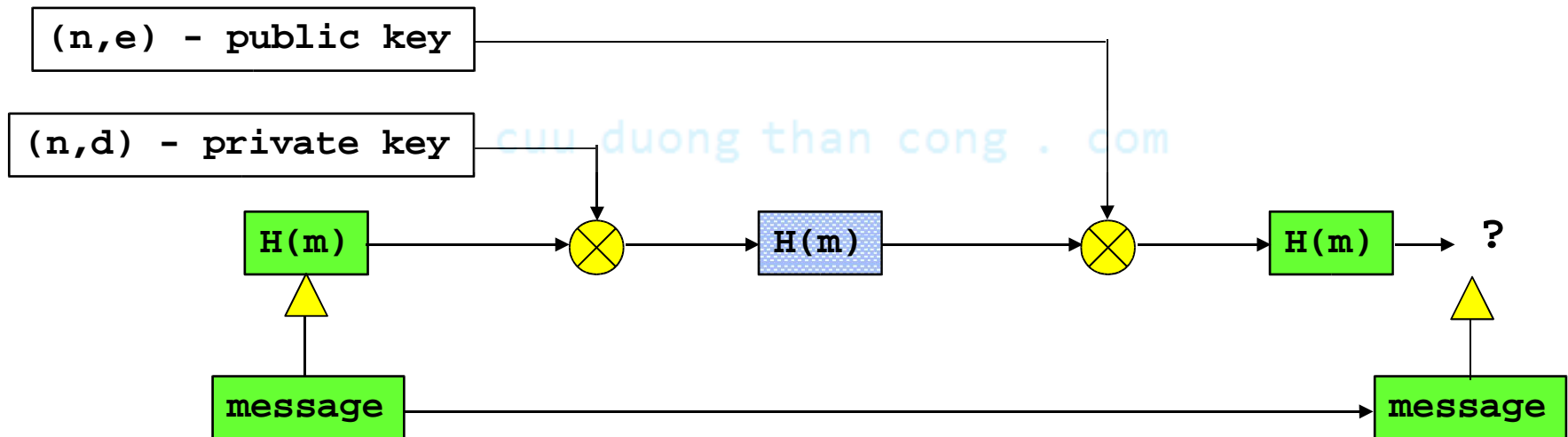
- $(r^d)^e \bmod n = r$

# Integrity - digital signature

Alice

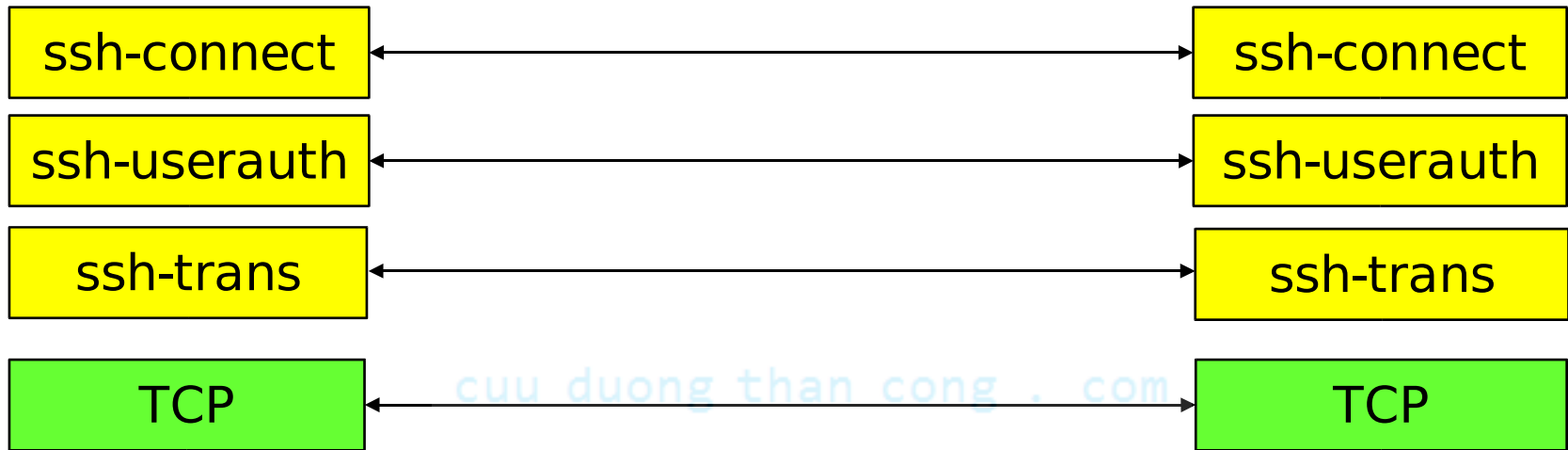


Bob



- Hash, digest, or MAC (Message Authentication Code)
  - 128 or 160 bits (MD5, SHA-1)
- Bob decrypts  $H(m)$  using the public key and verifies if
  - $H(m) = H(\text{message})$

# ssh architecture



- **ssh-trans**
  - server authentication, confidentiality, integrity
- **ssh-userauth**
  - authenticates the client-side user
- **ssh-connect**
  - multiplexes the encrypted tunnel into several logical channels (enables port redirection)

# ssh-trans

- Server authentication
  - each server host must have a host key
  - server host key is used during key exchange to verify that the client is really communicating with the correct server.
  - the client must have prior knowledge of the server's public host key:
    - client has a local database that associates each host name (as typed by the user) with the corresponding public host key.
    - host name - key association is certified by a trusted certification authority.
- Danger if the client talks to an unknown host
  - man-in-the-middle attack

# ssh-trans

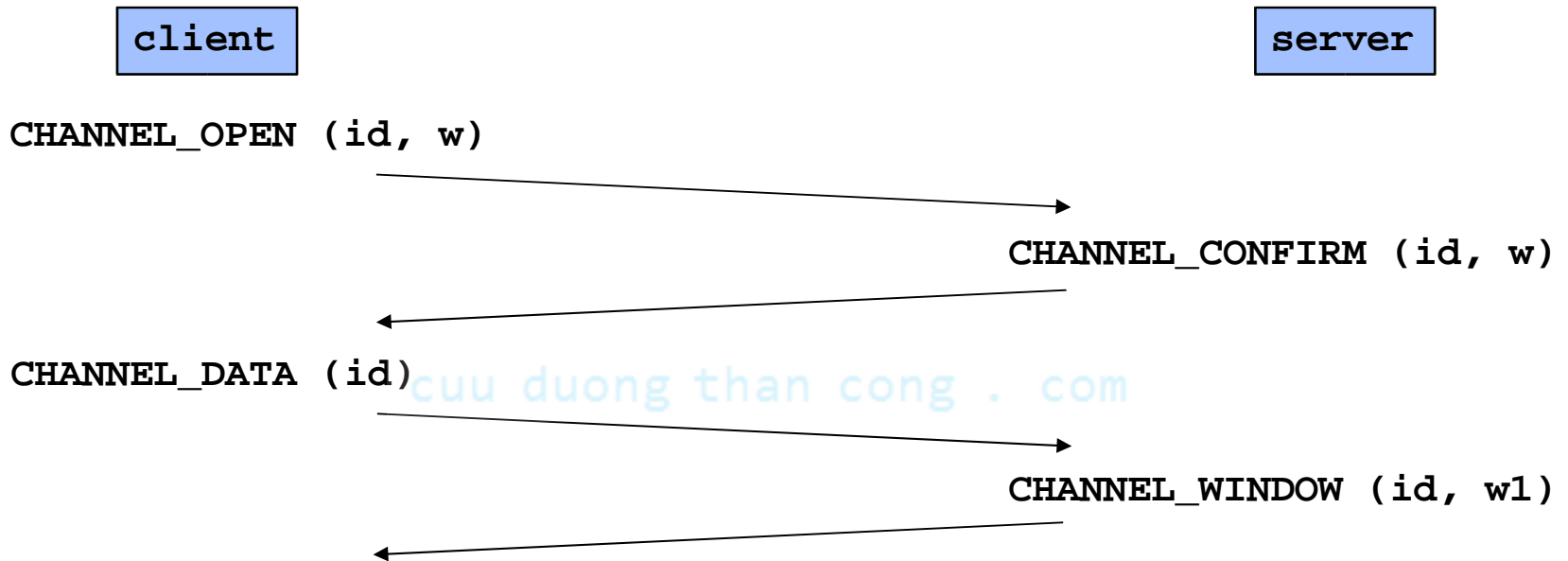
- Confidentiality
  - data encrypted using a one-time secret session key
- Key exchange phase
  - Diffie-Hellman method to create a secret key  $K$
  - $K$  used to derive a unique connection id
- Encryption
  - symmetric encryption using  $K$
  - several ciphers (e.g. 3DES)
- Integrity
  - MAC (Message Authentication Code) included with each packet
  - computed from the shared secret key, packet sequence number, the contents of the packet



# ssh-userauth

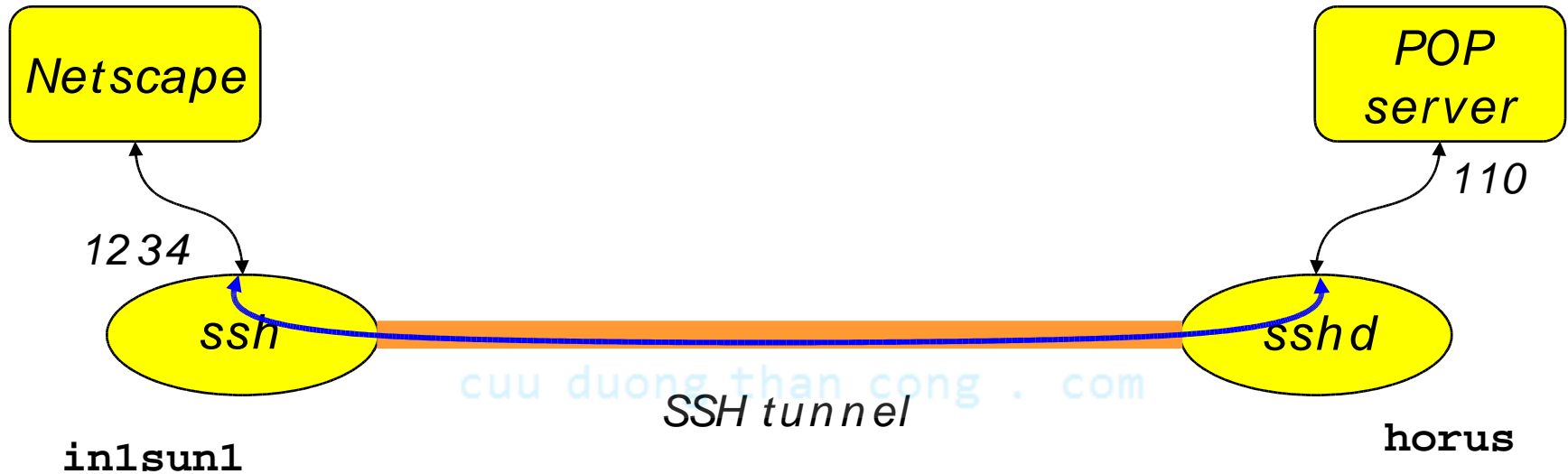
- Password
  - username, password on the remote system
- Public key authentication
  - user generates a pair of keys: public + secret
  - public key stored on the remote system
  - authentication request
    - signature by the secret key over session-id, username
    - the signature verified on the server by the public key
- Host based authentication
  - authentication request
    - signature by the client host secret key over session-id, hostname, username
    - the signature verified on the server by the public key

# ssh-connect



- Multiple channels multiplexed into a single connection at the ssh-trans level
- Channels identified by numbers on each end
- Channels are flow-controlled
  - window size - amount of data to send

# Local port redirection



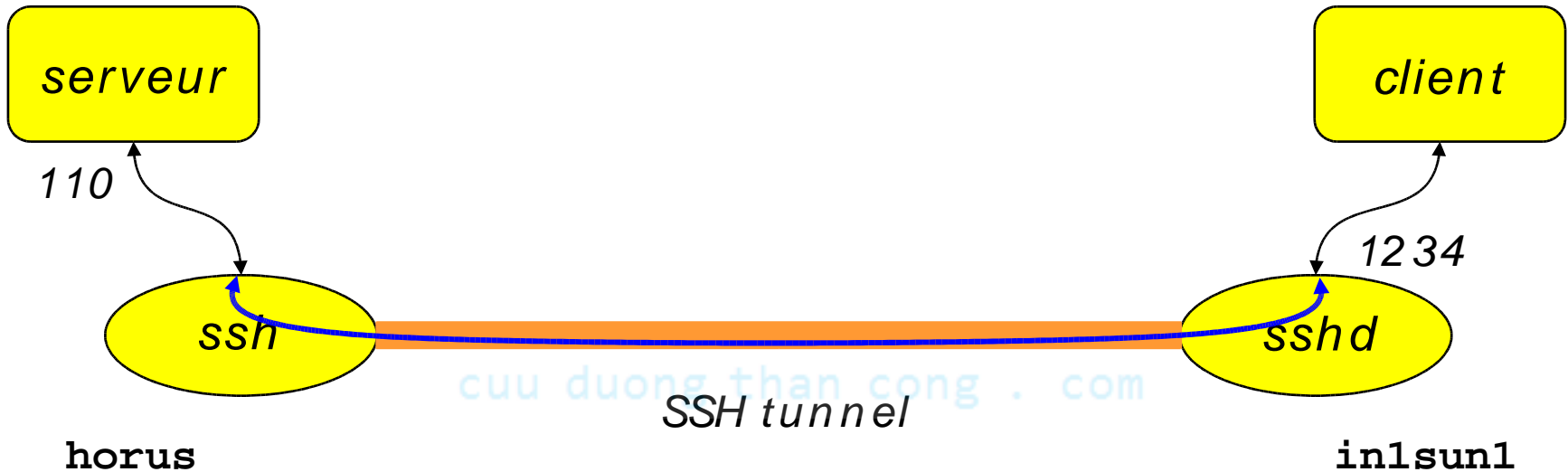
```
in1sun1% ssh -L 1234:horus.imag.fr:110
```

```
horus.imag.fr
```

config Netscape on `in1sun1` - read e-mail by POP on:  
`localhost`, port `1234`

e-mail will be read on `horus` through the ssh tunnel

# Remote port redirection



```
horus% su root
```

```
horus% ssh -R 1234:in1sun1.imag.fr:110  
in1sun1.imag.fr
```

Netscape on in1sun1: read e-mail by POP on  
localhost port 1234 (read in fact on horus)

# SSH: summary

- Excellent security
  - encryption and authentication
  - should be used instead of telnet/rlogin
- Integration with other applications
  - e-mail, X
- Known caveat
  - man in the middle attack:
    - intercept packets of both parties and generate packets so to make them think that they talk to each other
    - requires a possibility of packet intercepting

# Peer-to-peer file sharing

## ■ Example

- Alice runs P2P client application on her notebook computer
- Intermittently connects to Internet; gets new IP address for each connection
- Asks for “Hey Jude”
- Application displays other peers that have copy of Hey Jude.
- Alice chooses one of the peers, Bob.
- File is copied from Bob’s PC to Alice’s notebook: HTTP
- While Alice downloads, other users uploading from Alice.
- Alice’s peer is both a Web client and a transient Web server.
- All peers are servers = highly scalable!

# P2P: centralized directory

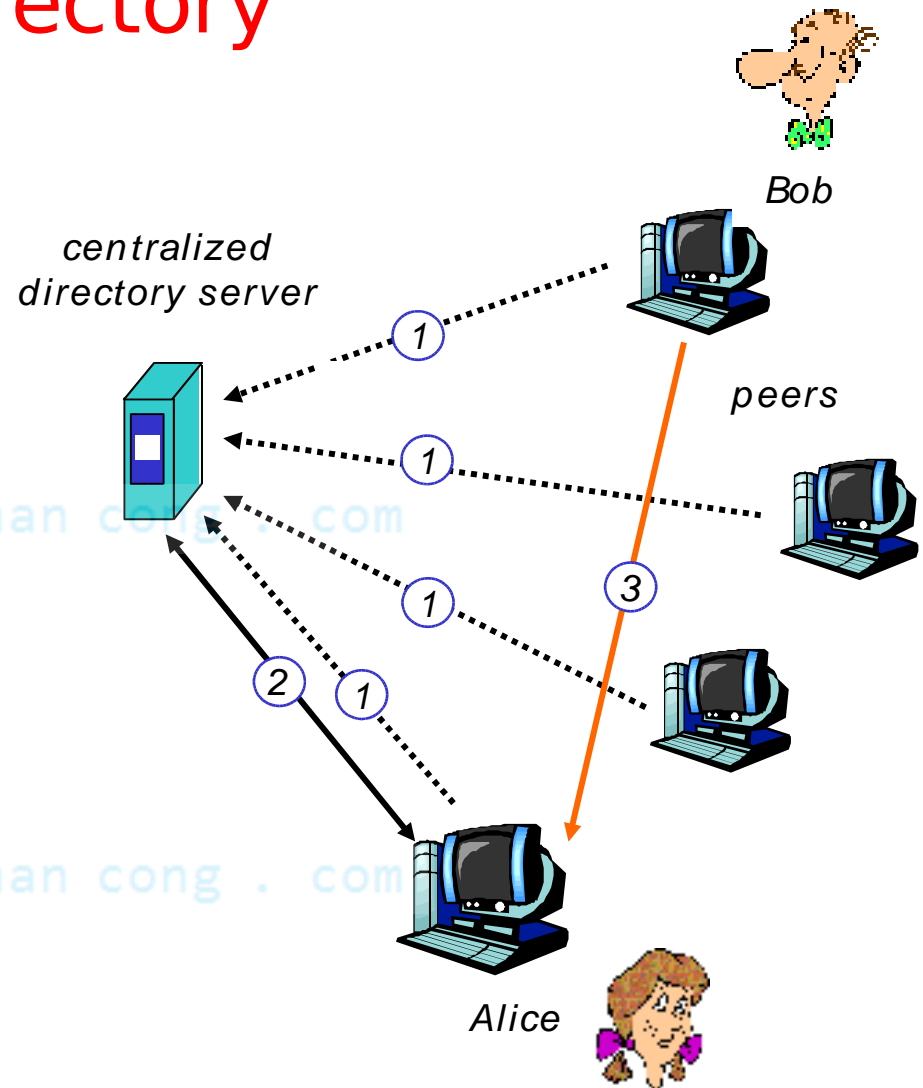
original “Napster”  
design

1) when peer connects,  
it informs central  
server:

- IP address
- content

2) Alice queries for “Hey  
Jude”

3) Alice requests file  
from Bob



# P2P: problems with centralized directory

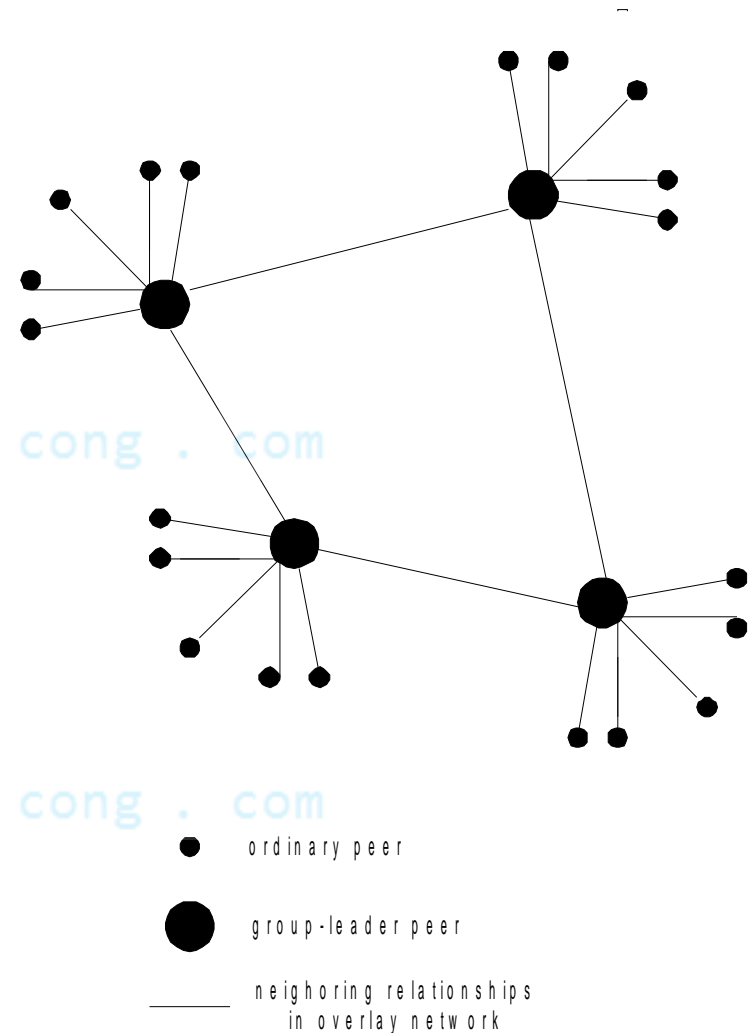
- Single point of failure
- Performance bottleneck
- Copyright infringement
  - Napster has been shut down by lawsuit

file transfer is decentralized, but locating content is highly centralized



# P2P: decentralized directory

- Each peer is either a group leader or assigned to a group leader.
- Group leader tracks the content in all its children.
- Peer queries group leader; group leader may query other group leaders.

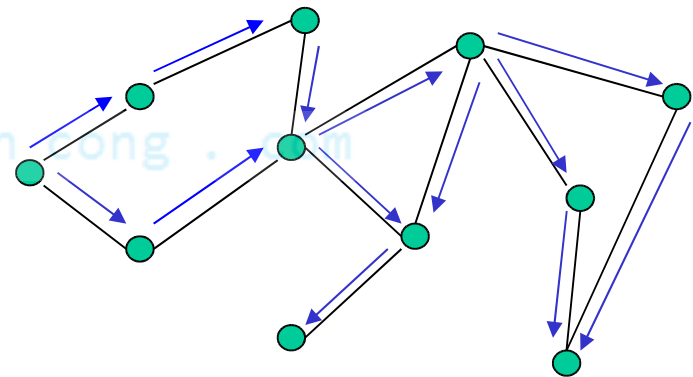
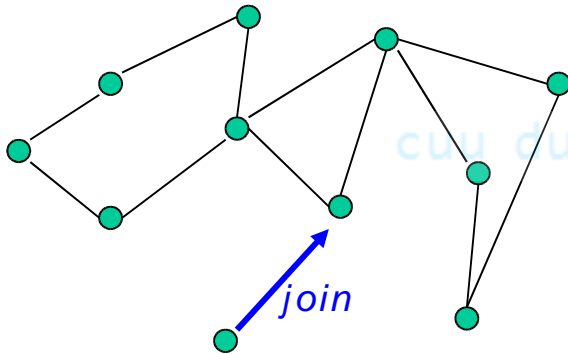


# More about decentralized directory

- Overlay network
  - peers are nodes
  - edges between peers and their group leaders
  - edges between some pairs of group leaders
  - virtual neighbors
- Bootstrap node
  - connecting peer is either assigned to a group leader or designated as leader
- Advantages of approach
  - no centralized directory server
    - location service distributed over peers
    - more difficult to shut down
- Disadvantages of approach
  - bootstrap node needed
  - group leaders can get overloaded

# P2P: Query flooding

- Example: Gnutella
- no hierarchy
- use bootstrap node to learn about others
- join message
- Send query to neighbors
- Neighbors forward query
- If queried peer has object, it sends message back to querying peer



# P2P: more on query flooding

## ■ Pros

- peers have similar responsibilities: no group leaders
- highly decentralized
- no peer maintains directory info

## ■ Cons

- excessive query traffic
- query radius: may not have content when present
- bootstrap node
- maintenance of overlay network

# Application layer: summary

Our study of networking applications now complete!

- Application service requirements:
  - reliability, bandwidth, delay
- Client-server paradigm
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- Specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - SSH
  - Peer-to-peer
- Finite State Machines
  - Formalism to describe & analyze protocols
  - Applies to all layers

# Application layer: summary

## Most importantly: learned about *protocols*

- Typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- Message formats:
  - headers: fields giving info about data
  - data: info being communicated
- Control vs. data msgs
  - in-band, out-of-band
- Centralized vs. decentralized
- Stateless vs. stateful
- Reliable vs. unreliable msg transfer
- “complexity at network edge”, “end-to-end”
- Security