# DNSSEC

*CS 161: Computer Security*

**Prof. David Wagner**

April 11, 2016

# DNSSEC

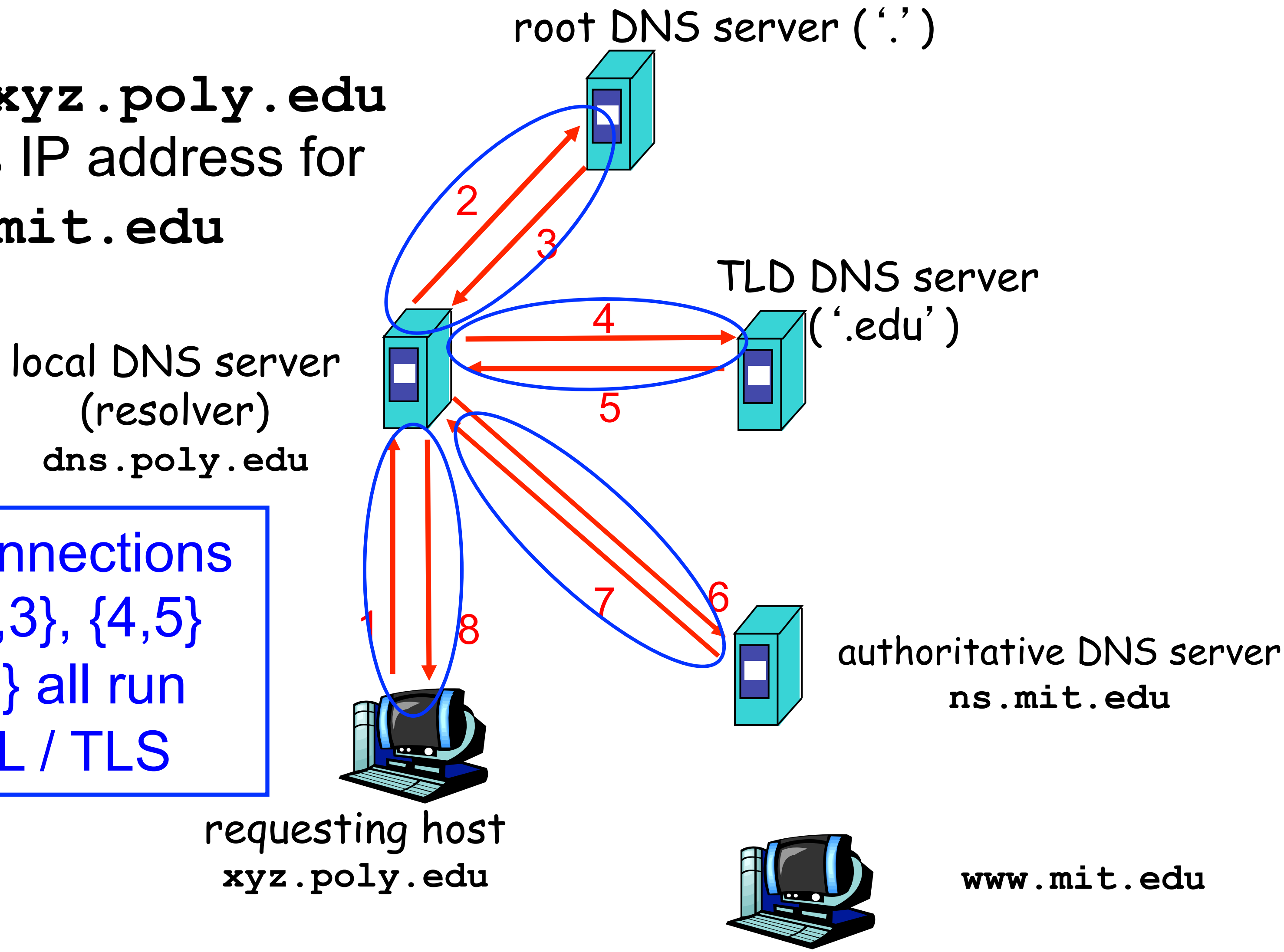
- Last lecture, you invented DNSSEC. Well, the basic ideas, anyway:
  - Sign all DNS records. Signatures let you verify answer to DNS query, without having to trust the network or resolvers involved.
- Remaining challenges:
  - DNS records change over time
  - Distributed database: No single central source of truth
- Today: how DNSSEC works

# Securing DNS Lookups

- How can we ensure that when clients look up names with DNS, they can trust the answers they receive?
- Idea #1: do DNS lookups over TLS (SSL)

# Securing DNS Using SSL/TLS

Host at **xyz.poly.edu** wants IP address for **www.mit.edu**




Idea: connections {1,8}, {2,3}, {4,5} and {6,7} all run over SSL / TLS

# Securing DNS Lookups

- How can we ensure that when clients look up names with DNS, they can trust the answers they receive?
- Idea #1: do DNS lookups over TLS (SSL)
  - Performance: DNS is very lightweight. TLS is not.
  - Caching: crucial for DNS scaling. But then how do we keep authentication assurances?
  - Security: must trust the resolver.
    ***Object*** *security vs.* ***Channel*** *security*
- Idea #2: make DNS results like *certs*
  - I.e., a verifiable signature that guarantees who generated a piece of data; signing happens **off-line**

# Operation of DNSSEC

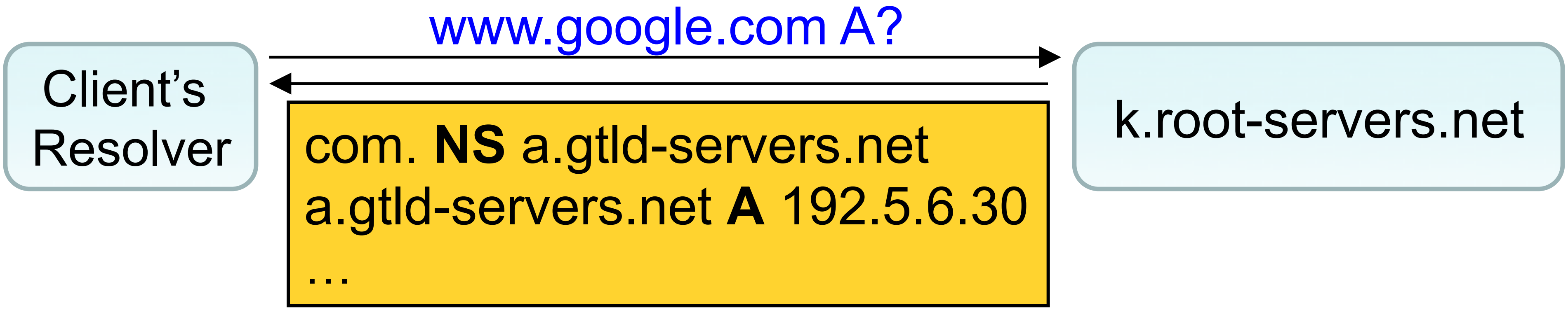
- DNSSEC = standardized DNS security extensions currently being deployed
- As a resolver works its way from DNS root down to final name server for a name, at each level it gets a signed statement regarding the key(s) used by the next level
  - This builds up a chain of trusted keys
  - Resolver has root's key wired into it
- The final answer that the resolver receives is signed by that level's key
  - Resolver can trust it's the right key because of chain of support from higher levels
- All keys *as well as signed results* are *cacheable*

# Ordinary DNS:

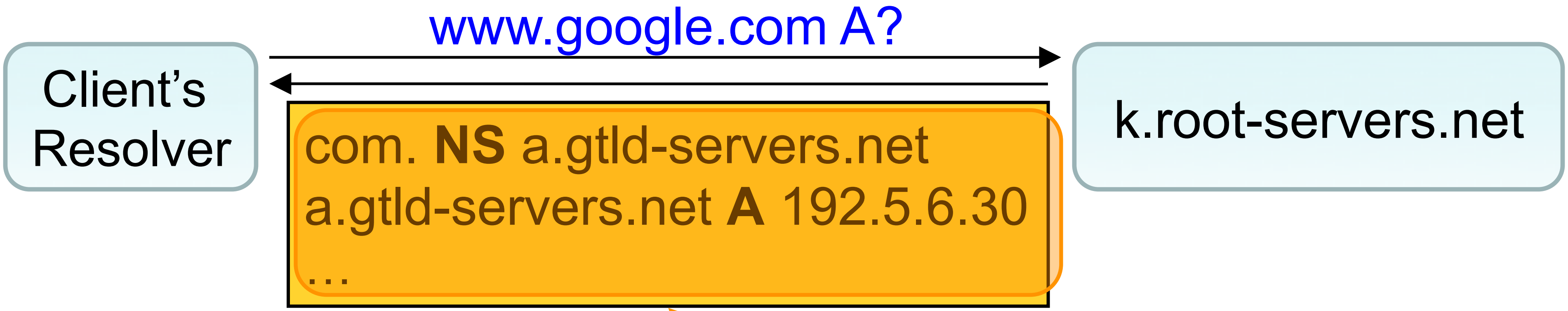
# Ordinary DNS:


We start off by sending the query to one of the root name servers. These range from `a.root-servers.net` through `m.root-servers.net`. Here we just picked one.

# Ordinary DNS:

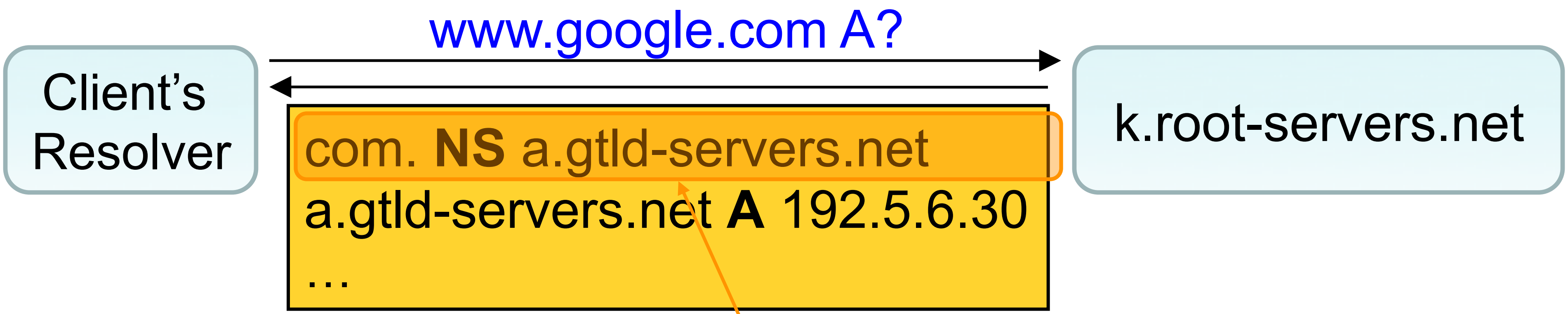


Ordinary DNS:



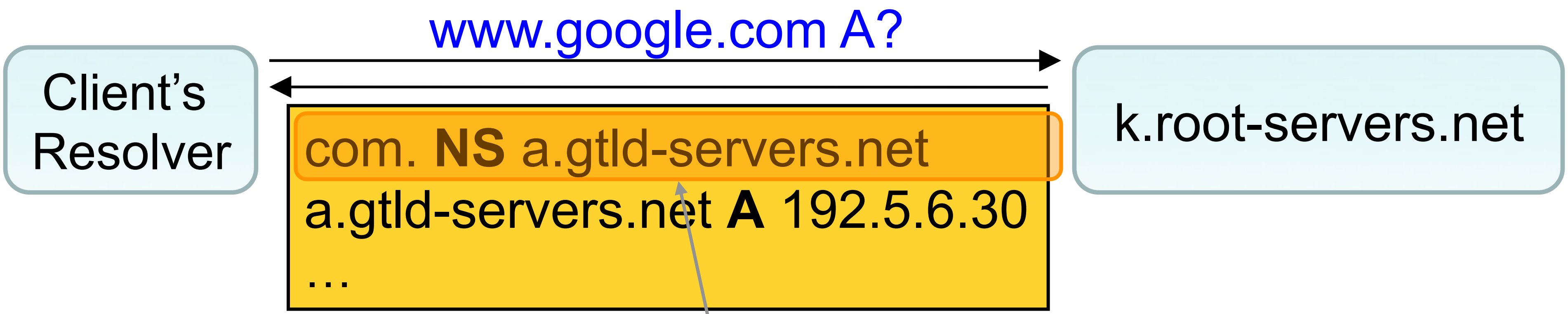

The reply *didn't include an answer* for `www.google.com`. That means that `k.root-servers.net` is instead telling us *where to ask next*, namely one of the name servers for `.com` specified in an **NS** record.

# Ordinary DNS:



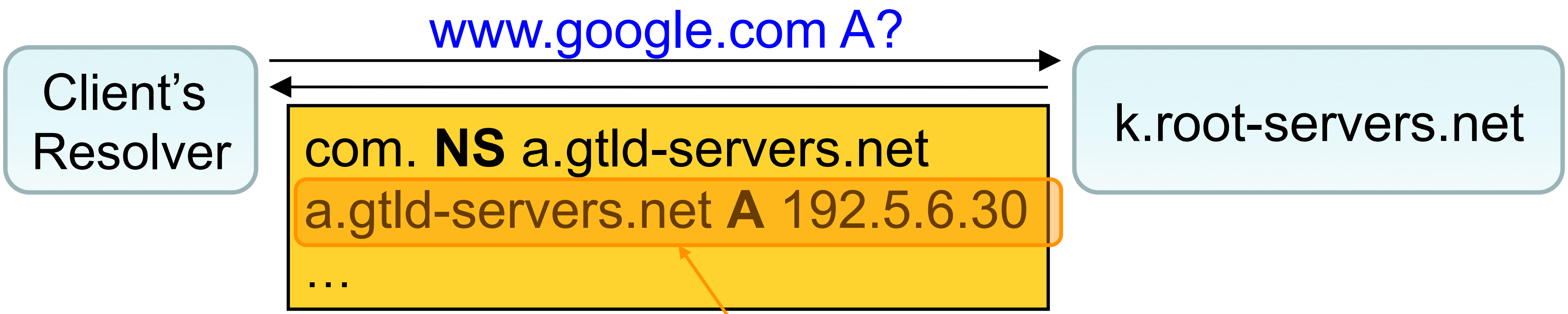

This *Resource Record* (**RR**) tells us that one of the name servers for `.com` is the host `a.gtld-servers.net`. (GTLD = Global Top Level Domain.)

# Ordinary DNS:



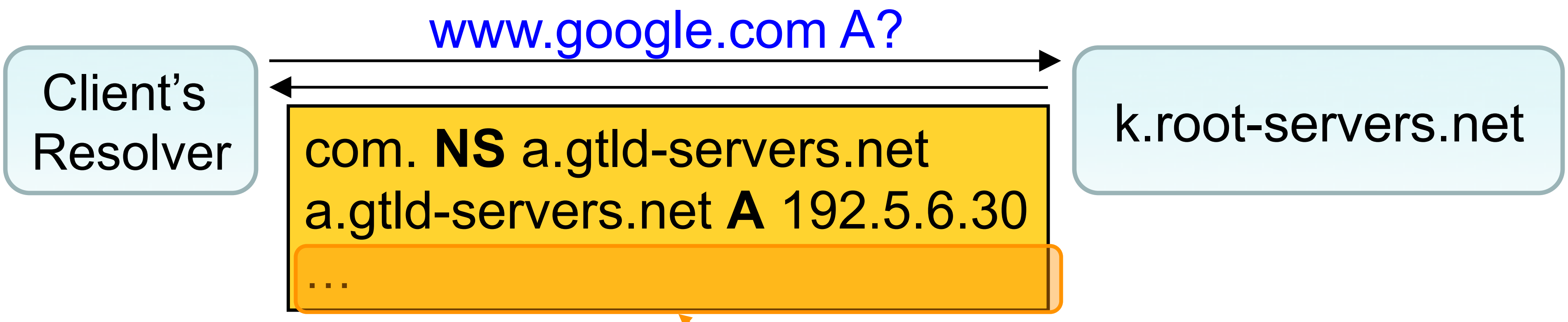

(The line above shows `com.` rather than `.com` because technically that's the actual name, and that's what the Unix `dig` utility shows; but the convention is to *call* it "dot-com")

# Ordinary DNS:



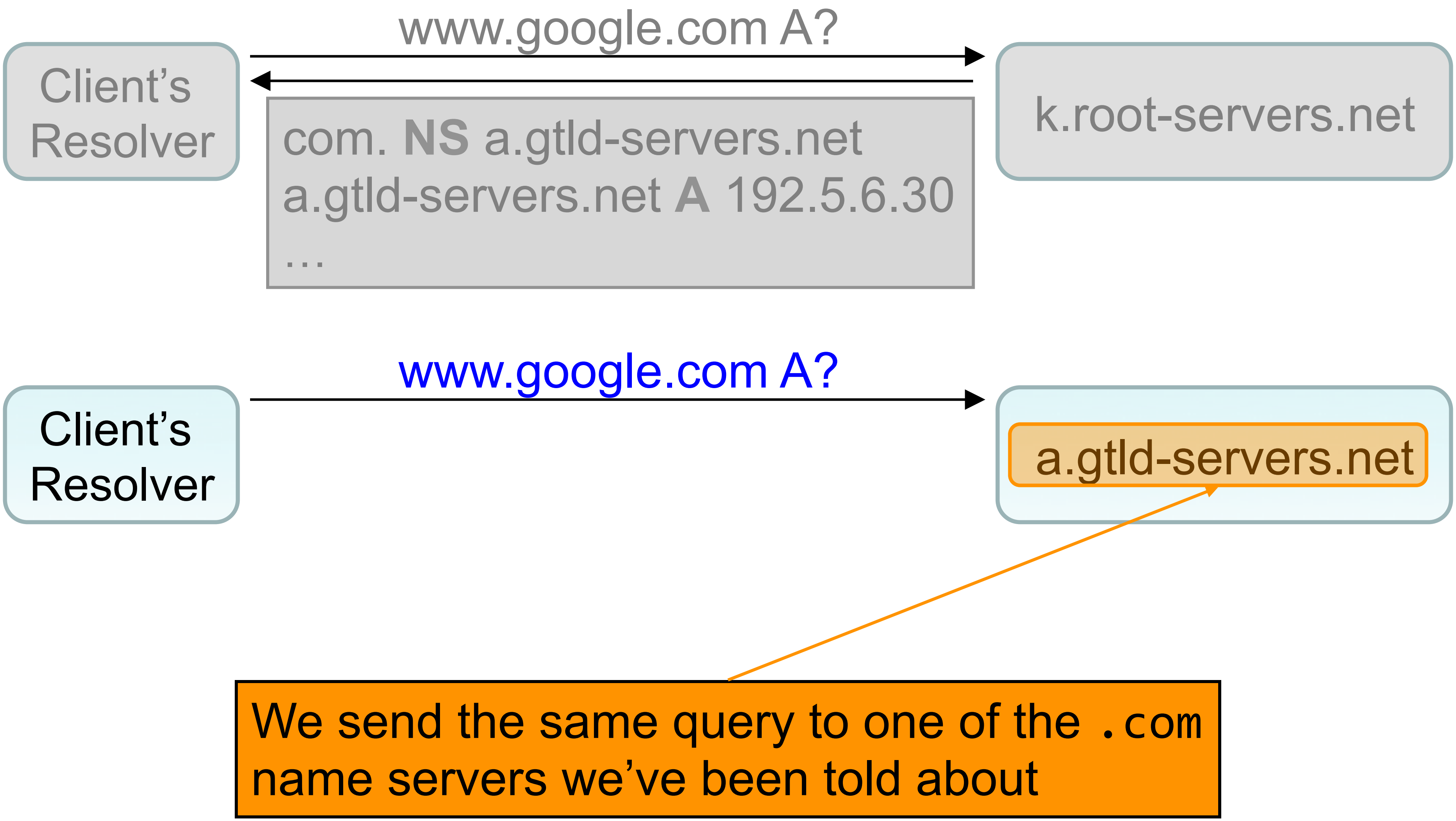

This **RR** tells us that an Internet address ("**A**" record) for `a.gtld-servers.net` is `192.5.6.30`. That allows us to know where to send our next query.

# Ordinary DNS:

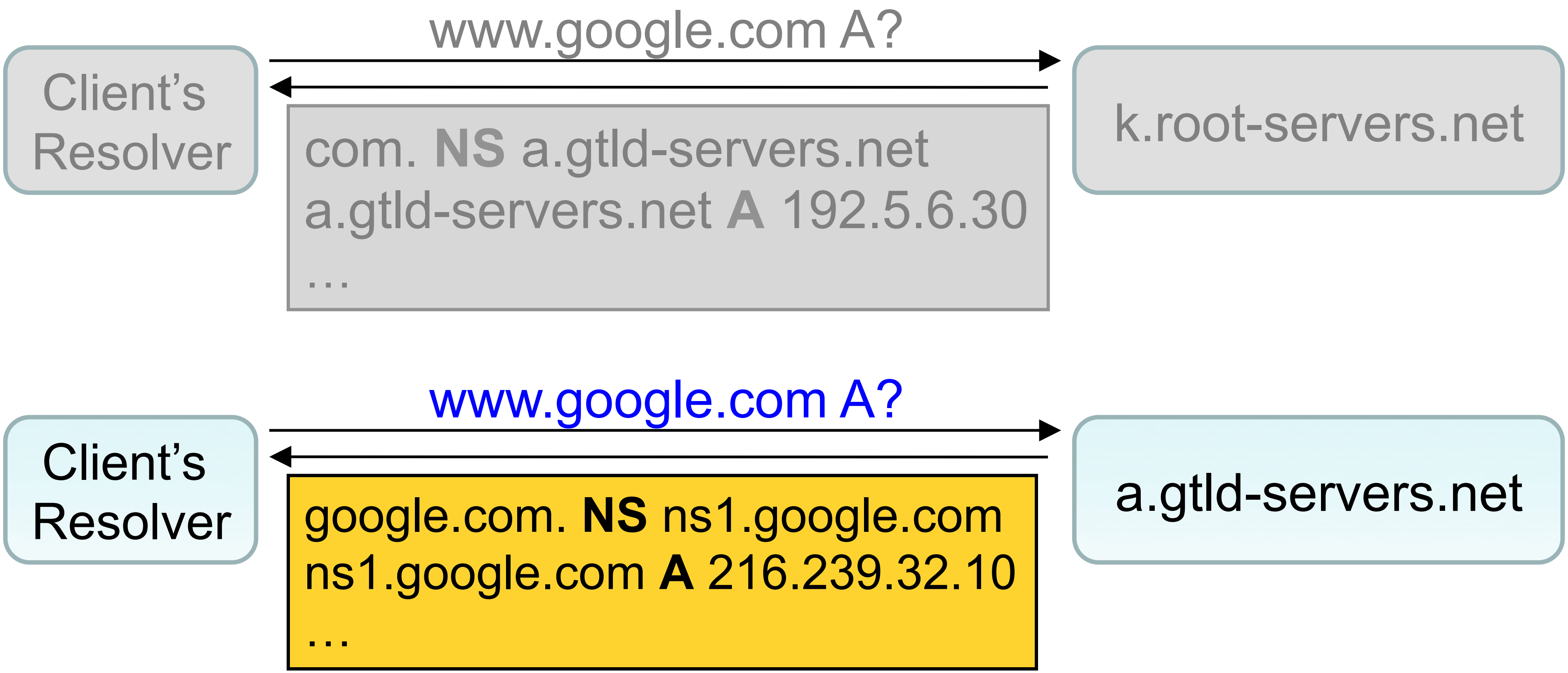



The actual response includes a bunch of **NS** and **A** records for additional `.com` name servers, which we omit here for simplicity.

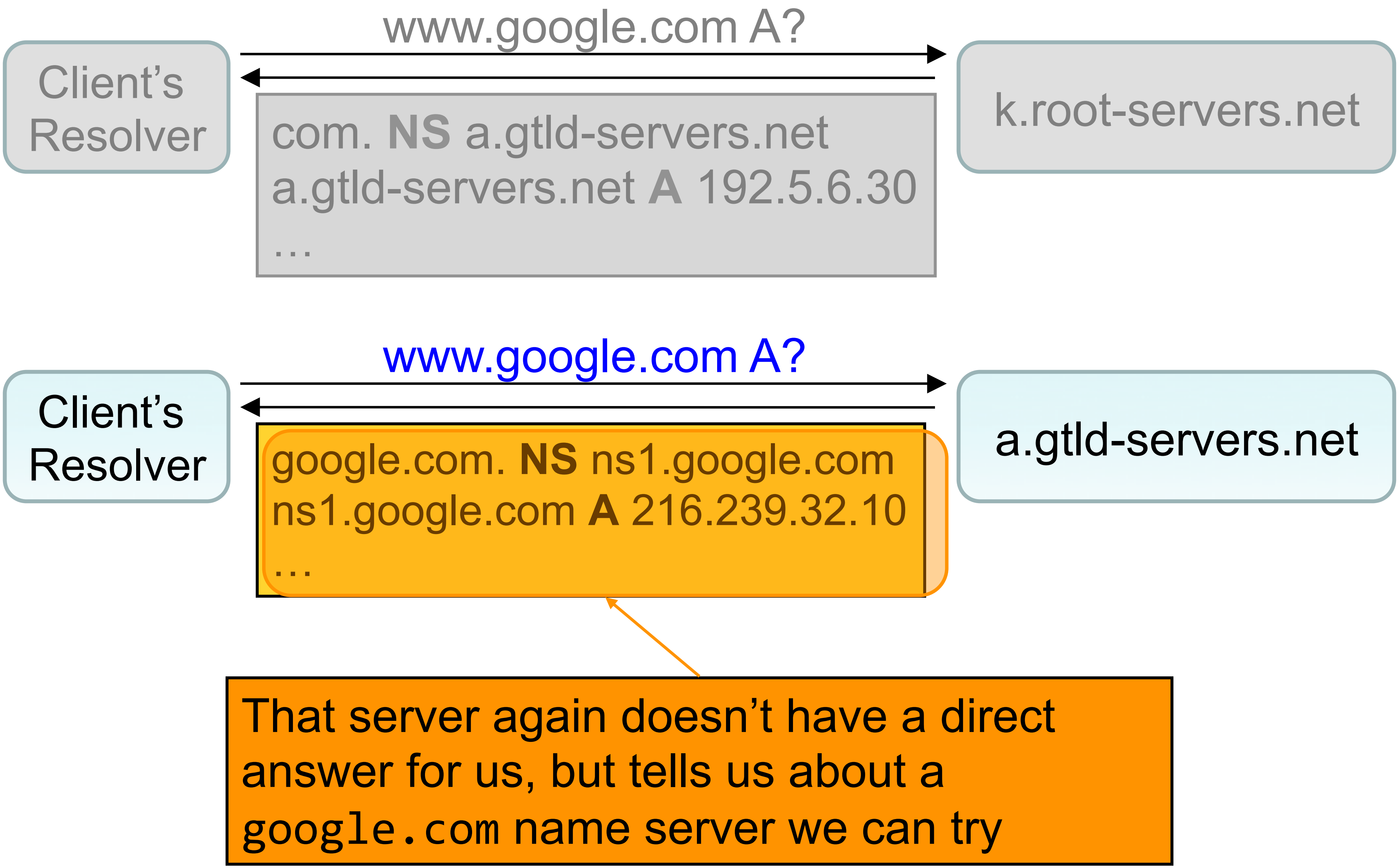
# Ordinary DNS:



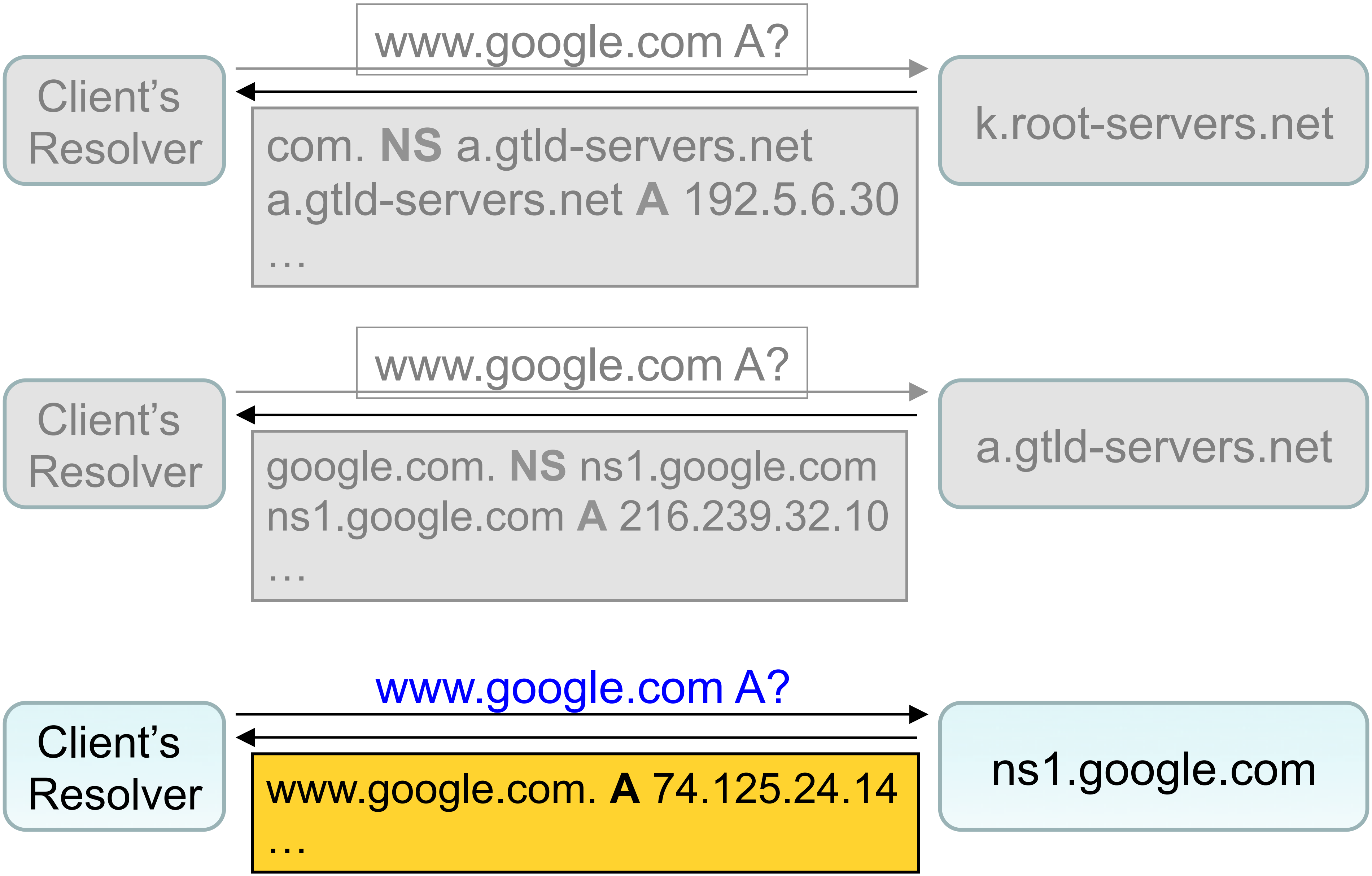
Ordinary DNS:



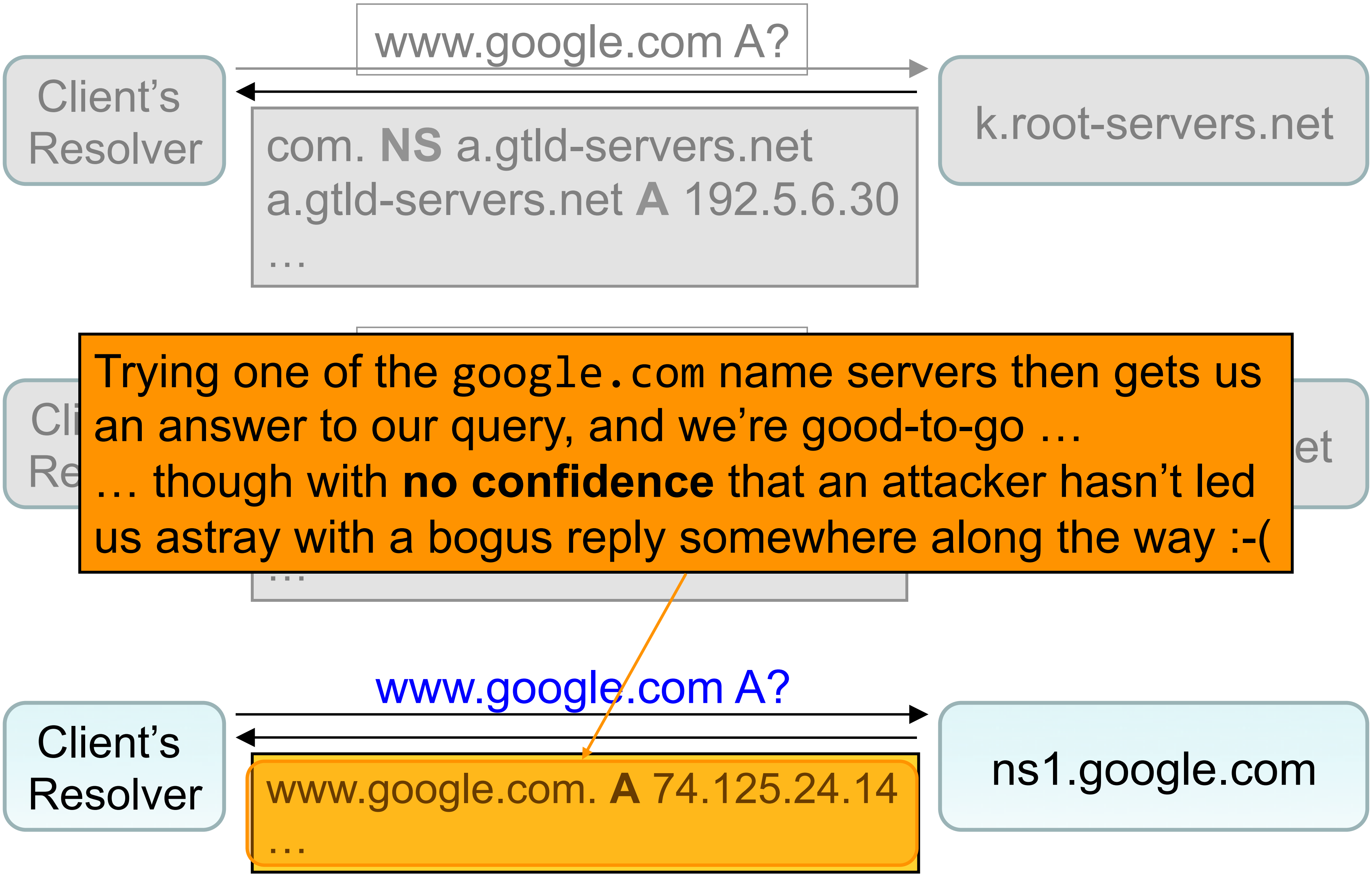
# Ordinary DNS:



# Ordinary DNS:

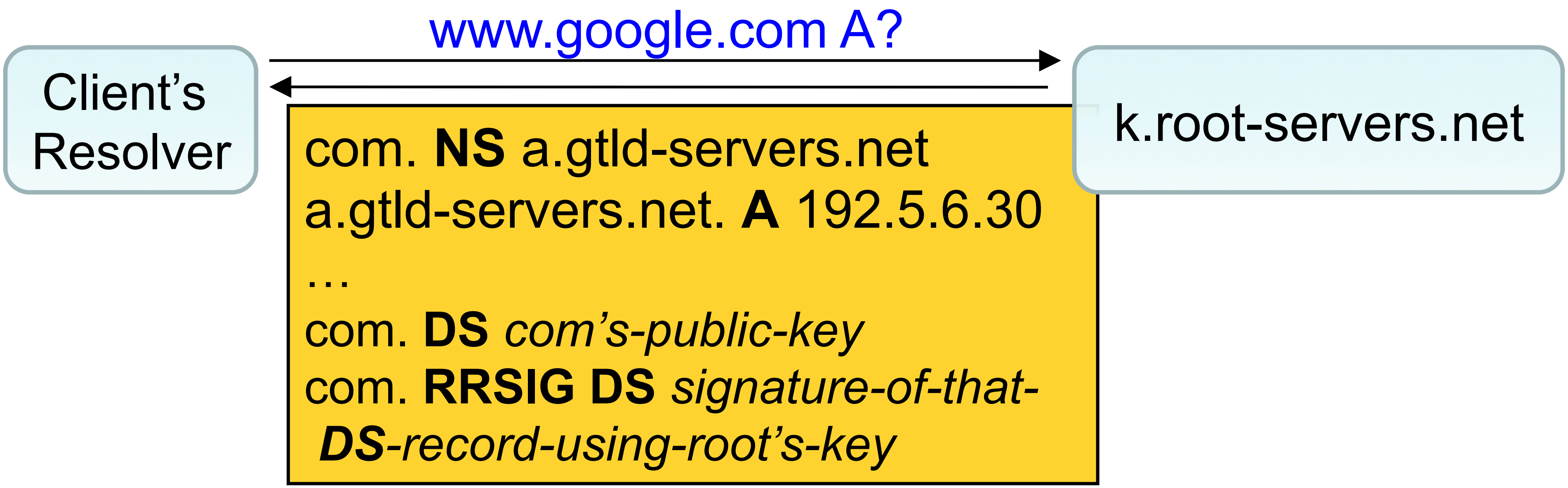


# Ordinary DNS:

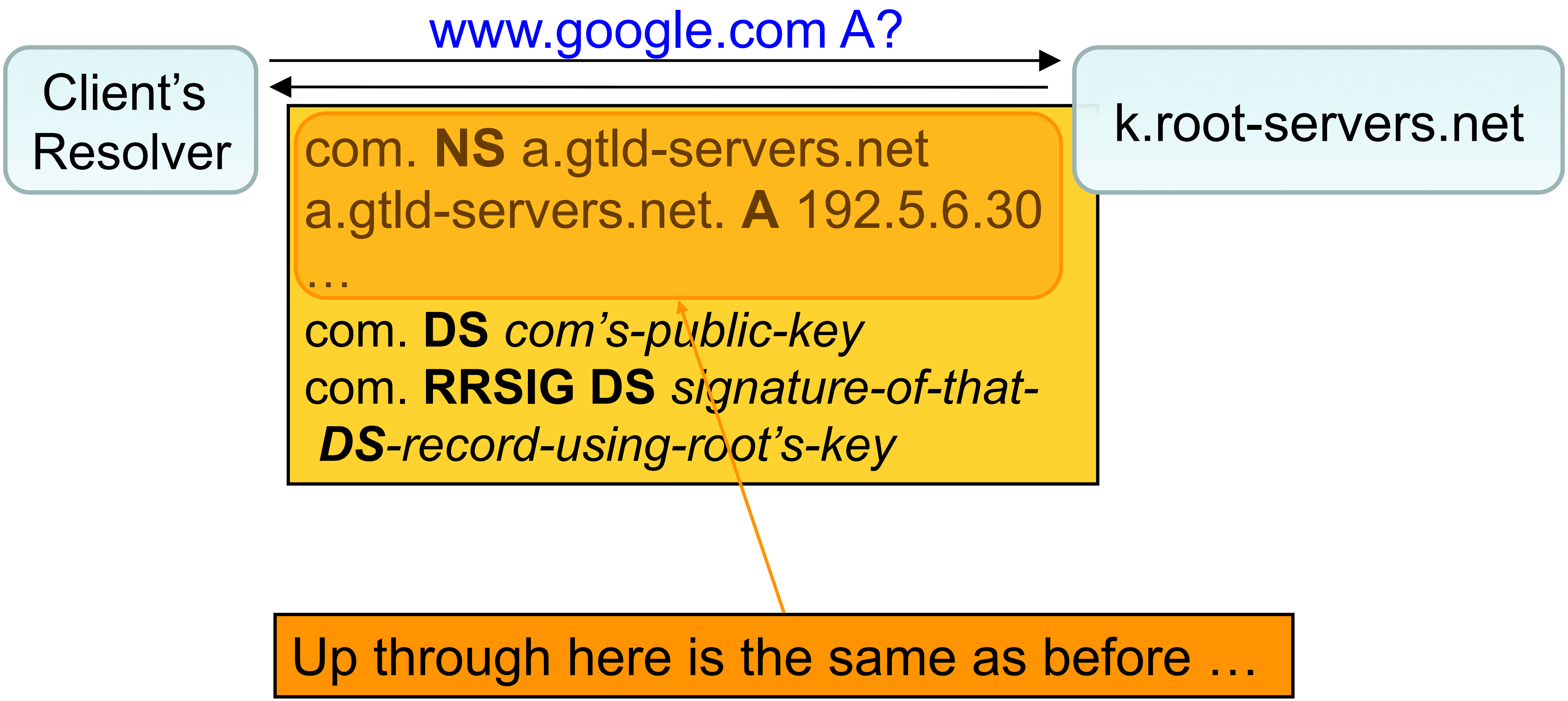



Trying one of the `google.com` name servers then gets us an answer to our query, and we're good-to-go …
… though with **no confidence** that an attacker hasn't led us astray with a bogus reply somewhere along the way :-(

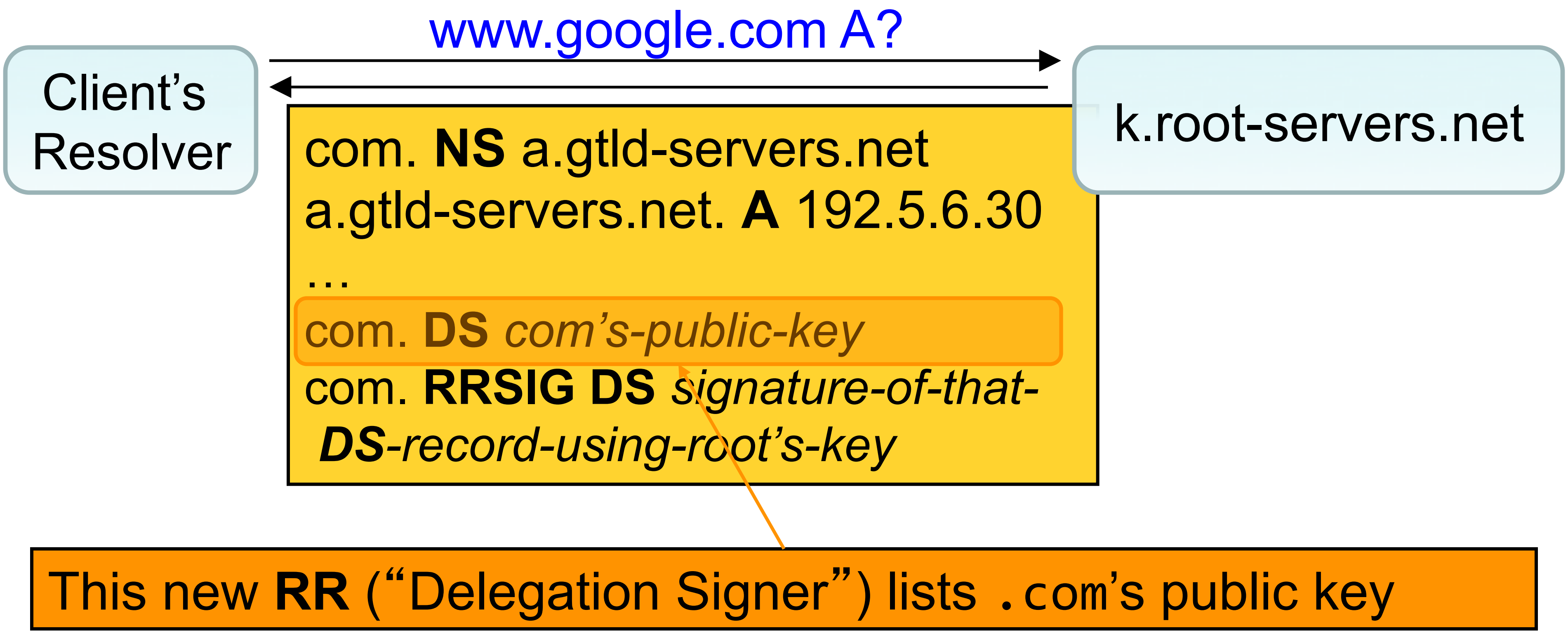
# DNSSEC (with simplifications):



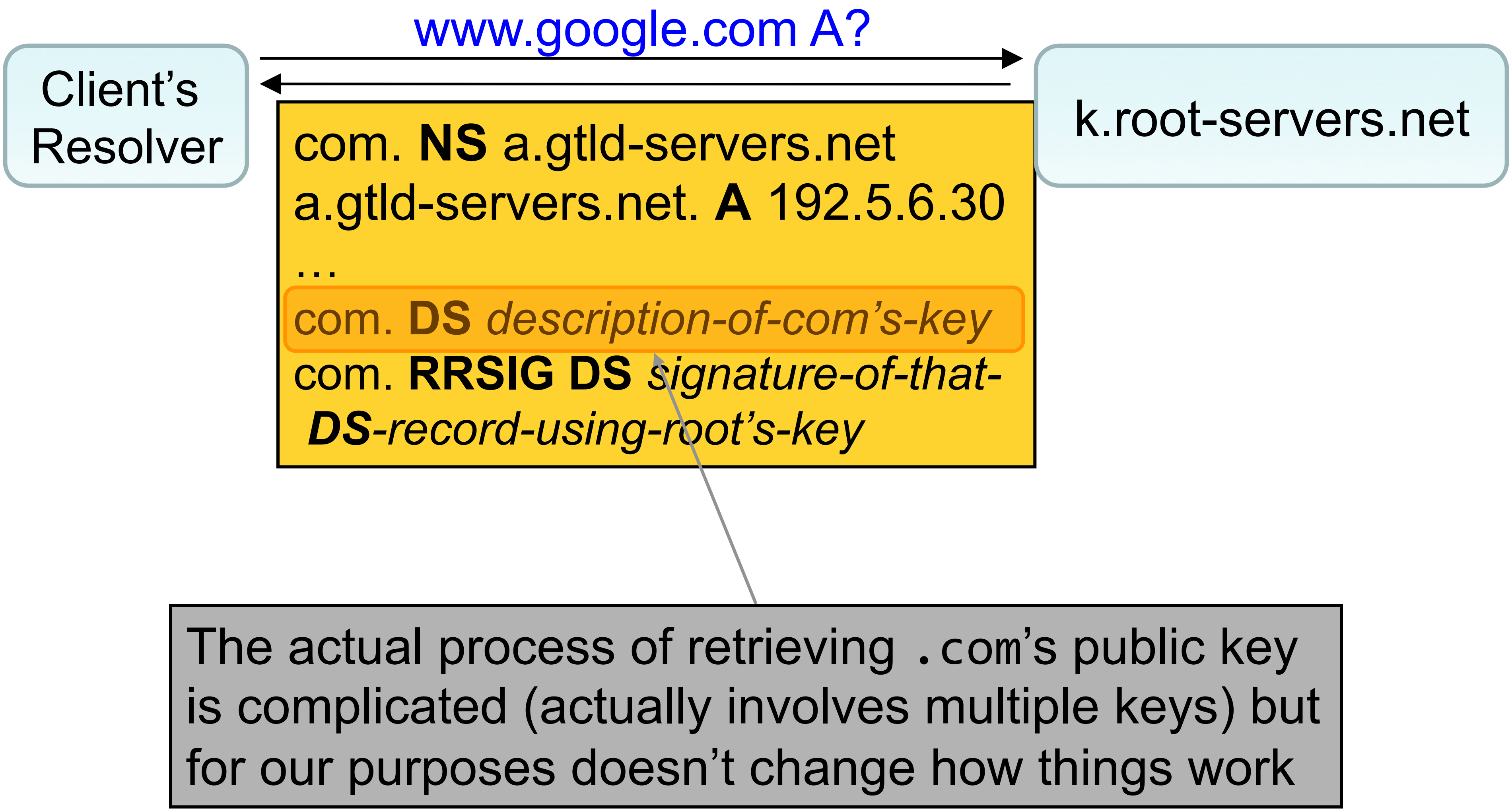
# DNSSEC (with simplifications):



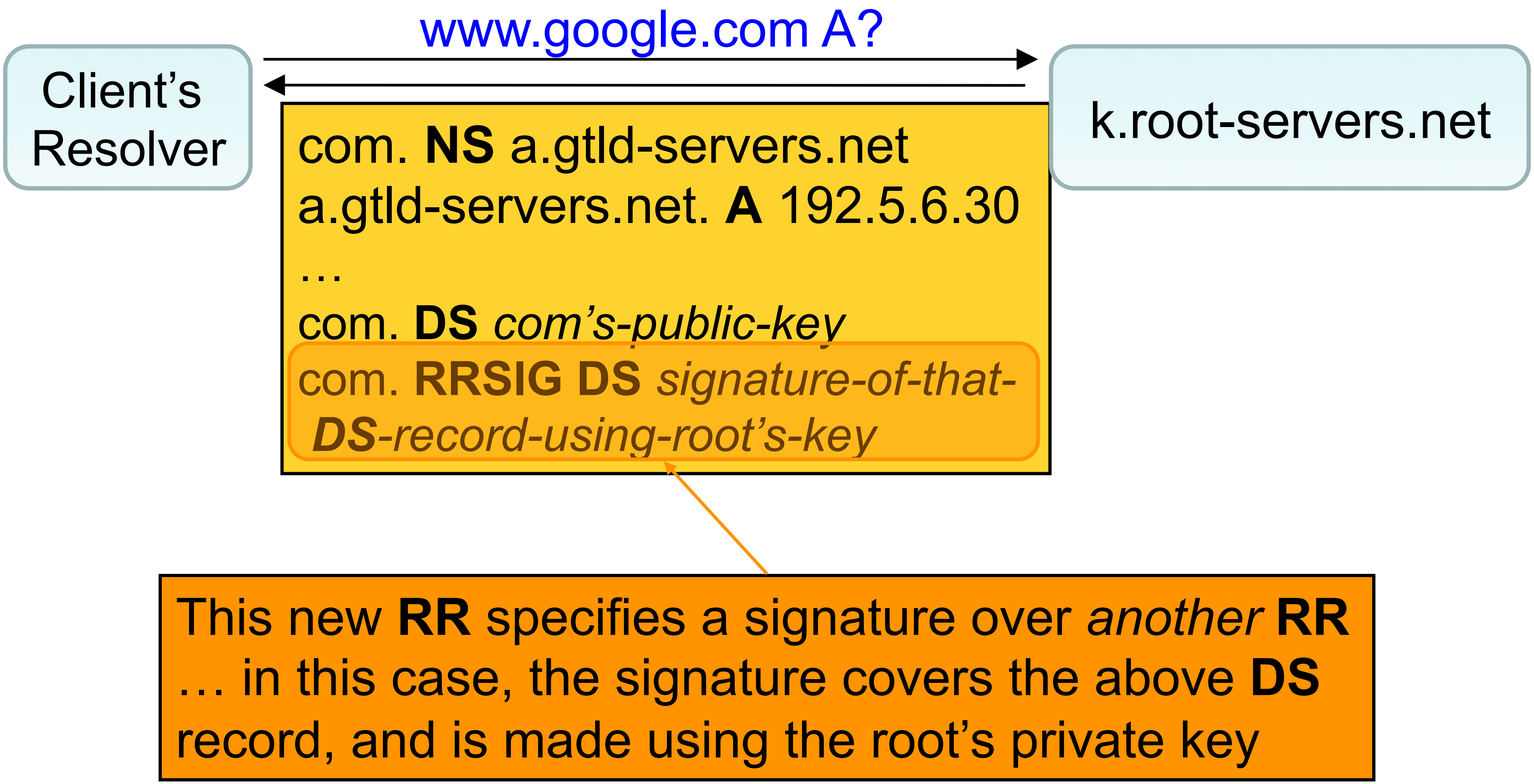
# DNSSEC (with simplifications):



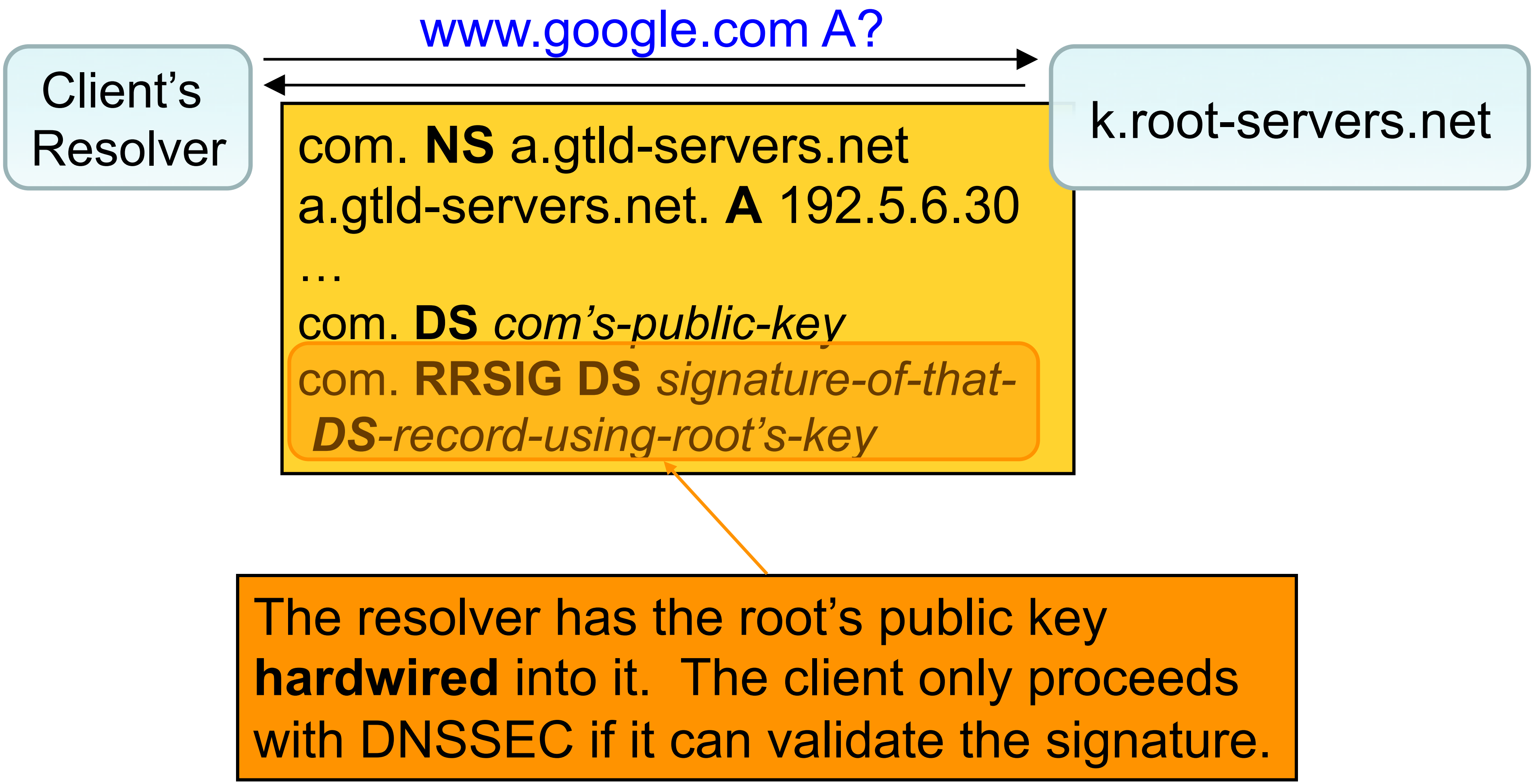
# DNSSEC (with simplifications):



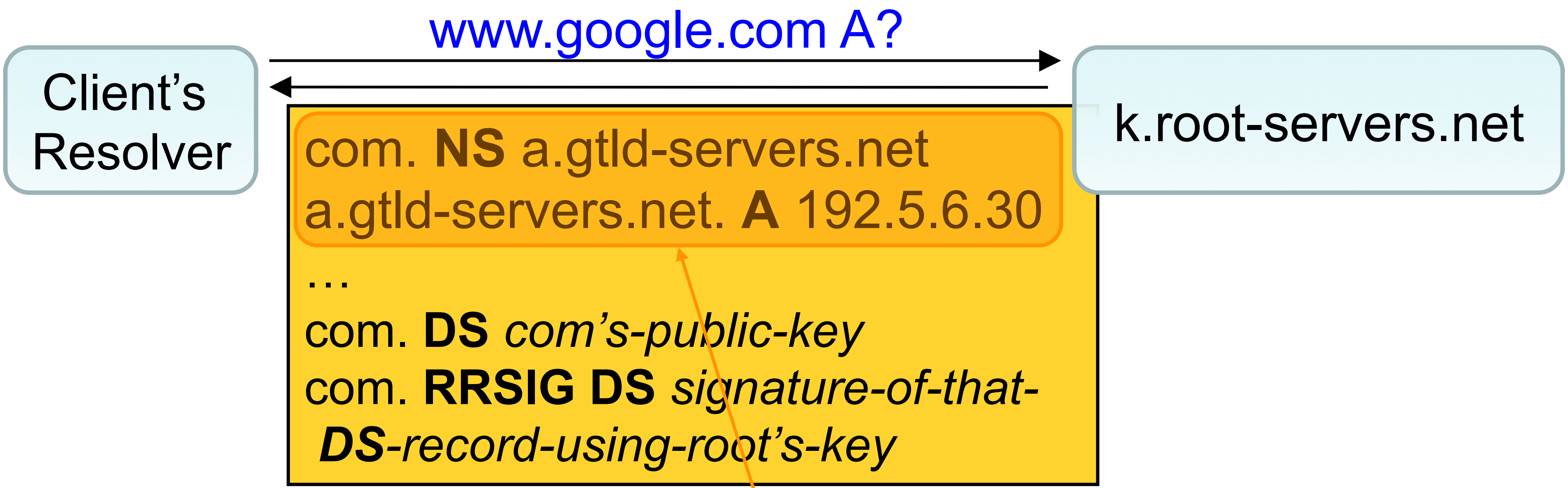
# DNSSEC (with simplifications):



# DNSSEC (with simplifications):

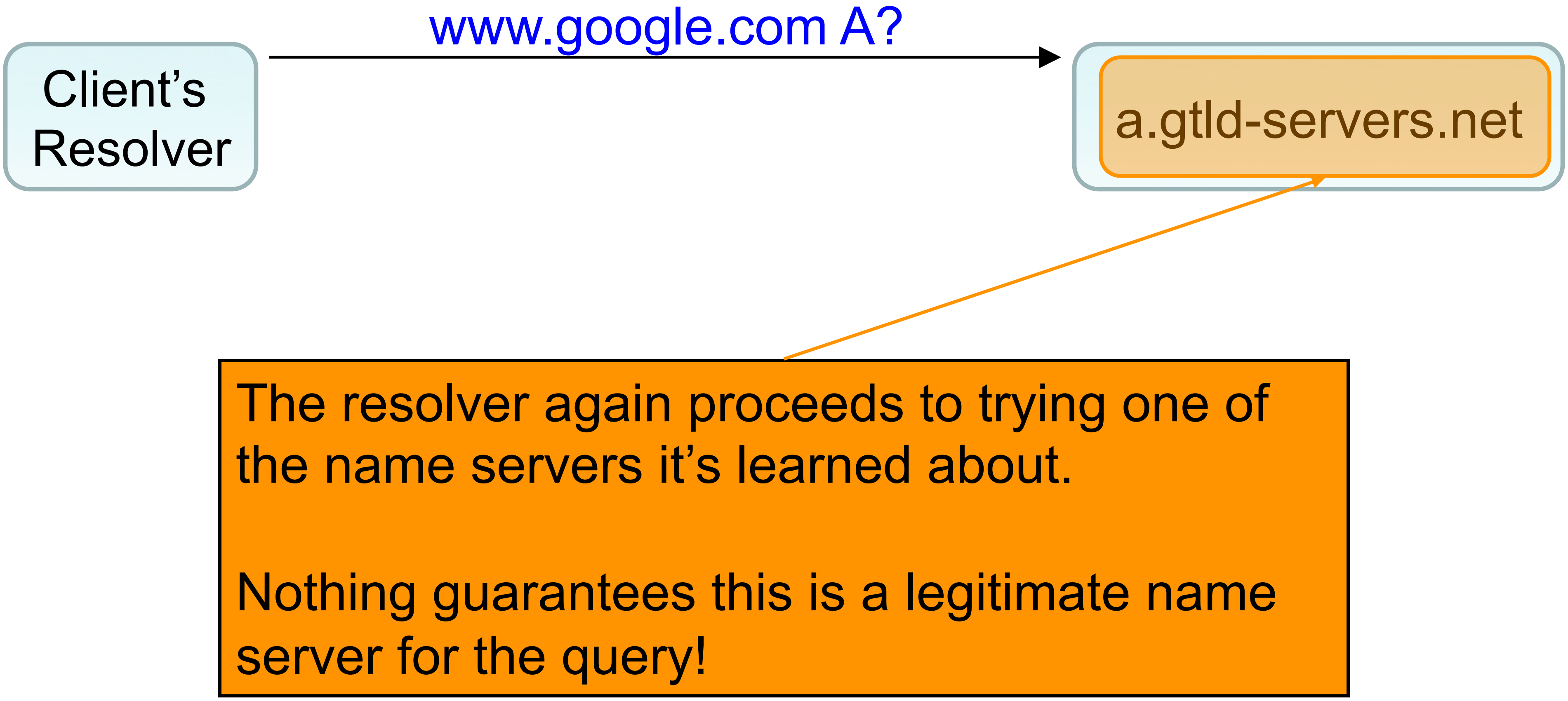


# DNSSEC (with simplifications):

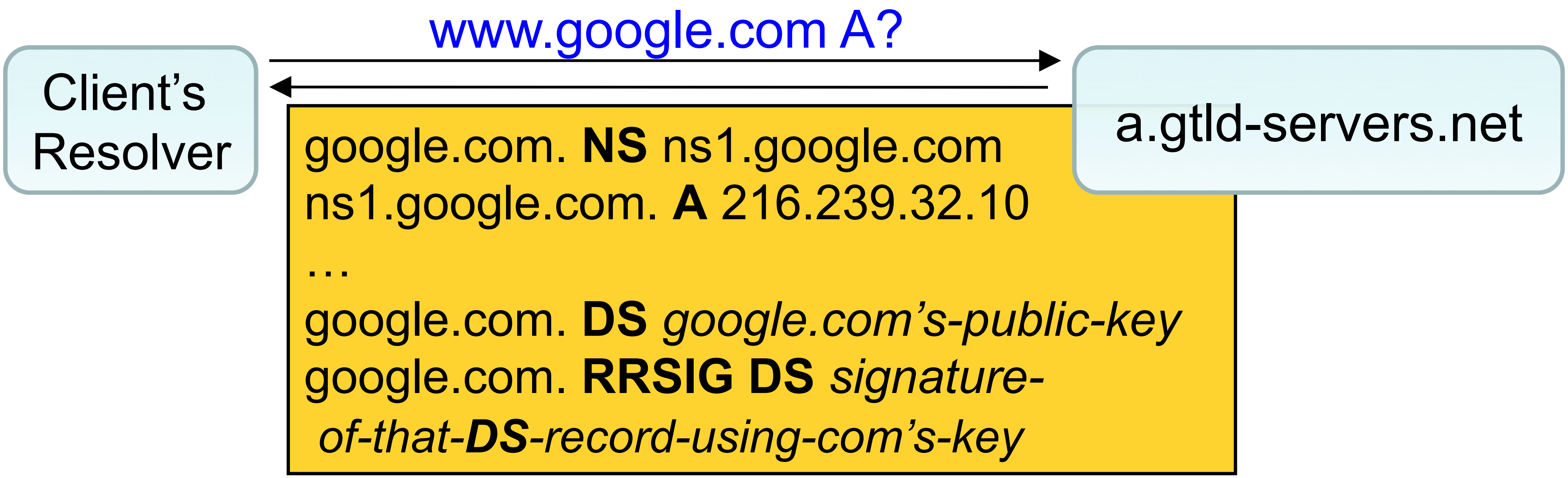



Note: there's no signature over the **NS** or **A** information! If an attacker has fiddled with those, the resolver will ultimately find it has a record for which it can't verify the signature.

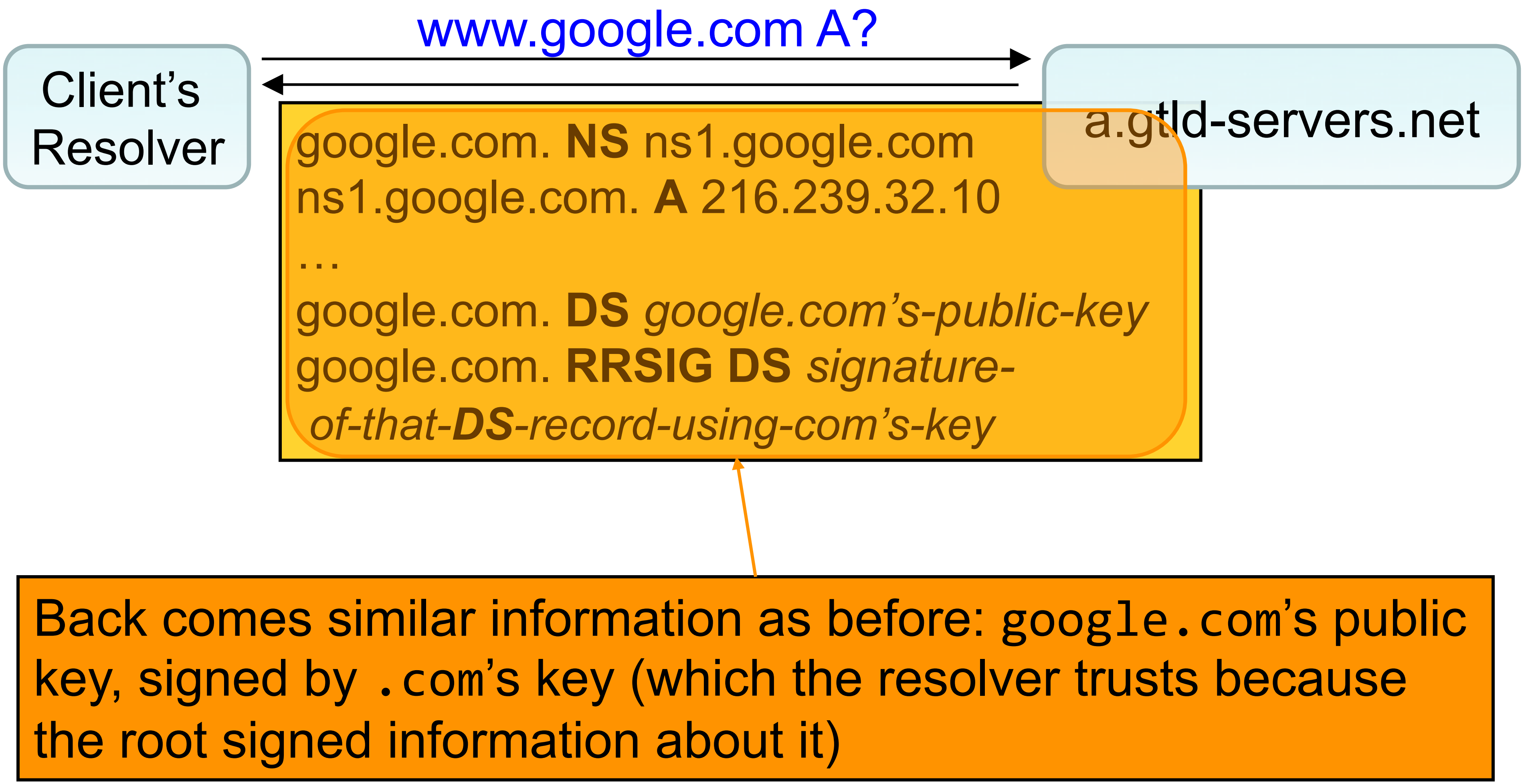
# DNSSEC (with simplifications):



# DNSSEC (with simplifications):



# DNSSEC (with simplifications):

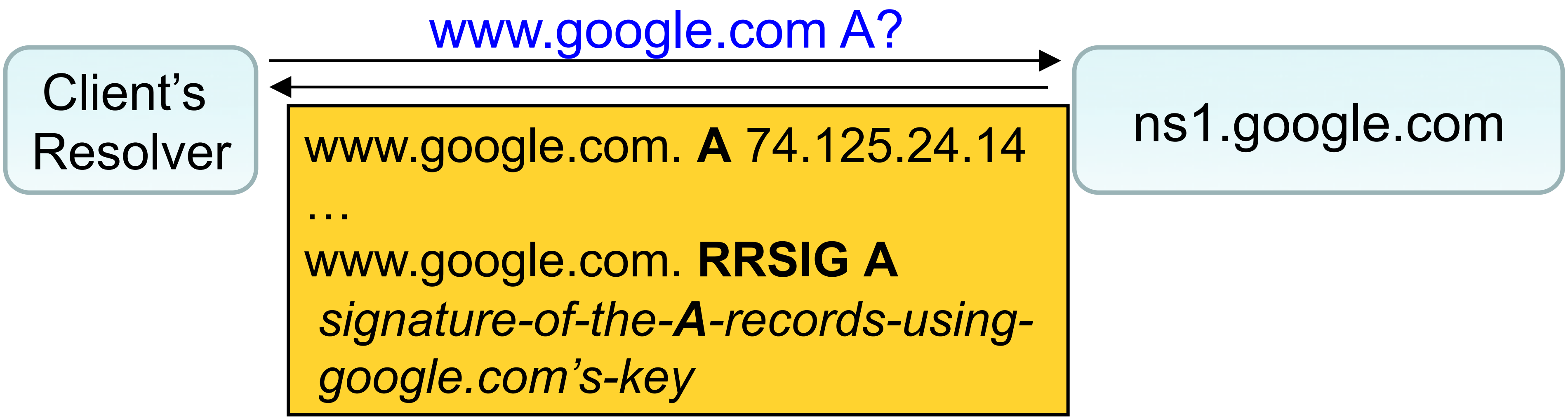


# DNSSEC (with simplifications):

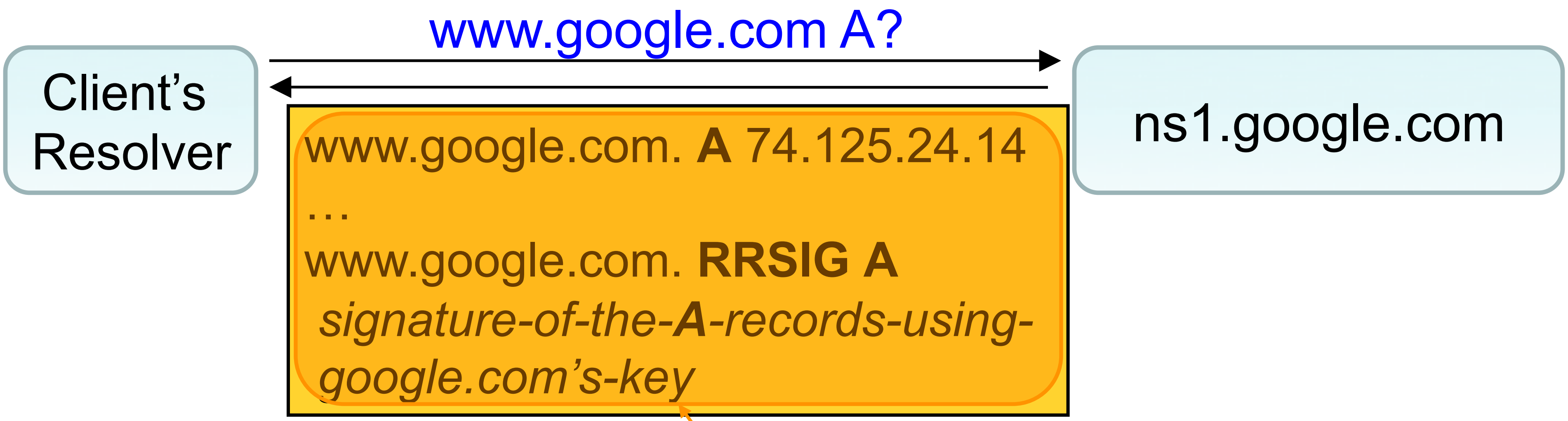

The resolver contacts one of the `google.com` name servers it's learned about.

Again, nothing guarantees this is a legitimate name server for the query!

# DNSSEC (with simplifications):

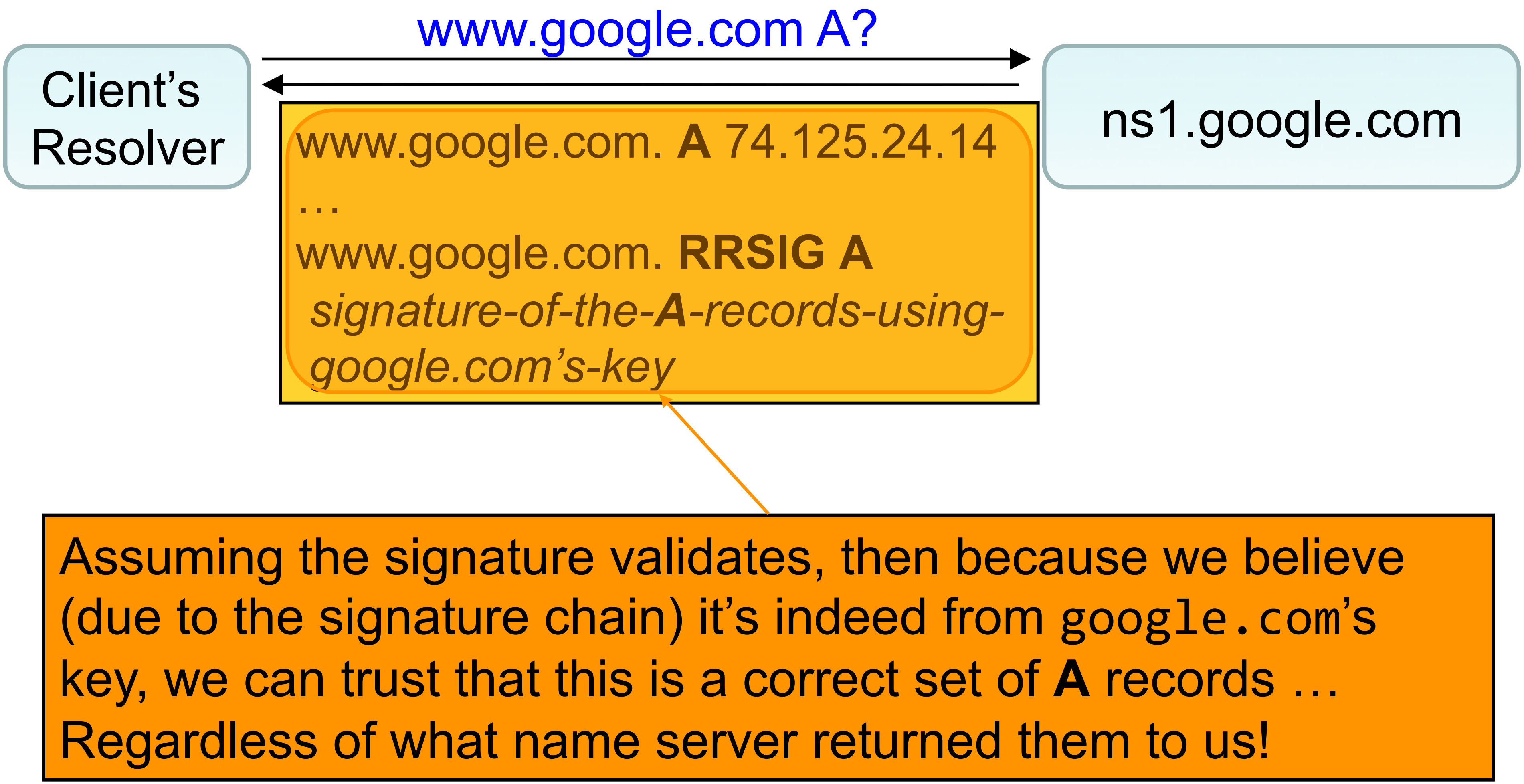


# DNSSEC (with simplifications):

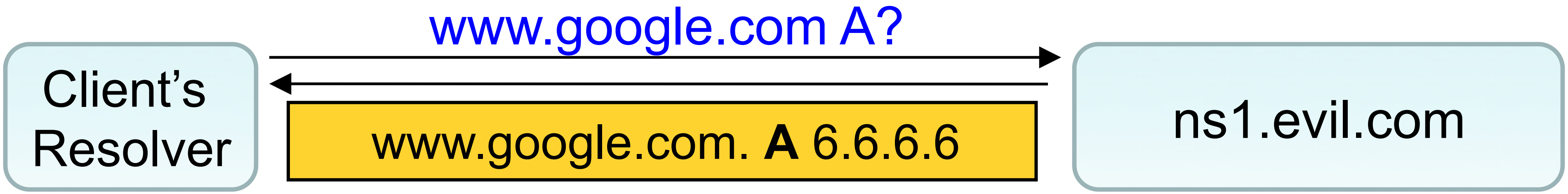



Finally we've received the information we wanted (**A** records for `www.google.com`)! … *and* we receive a signature over those records

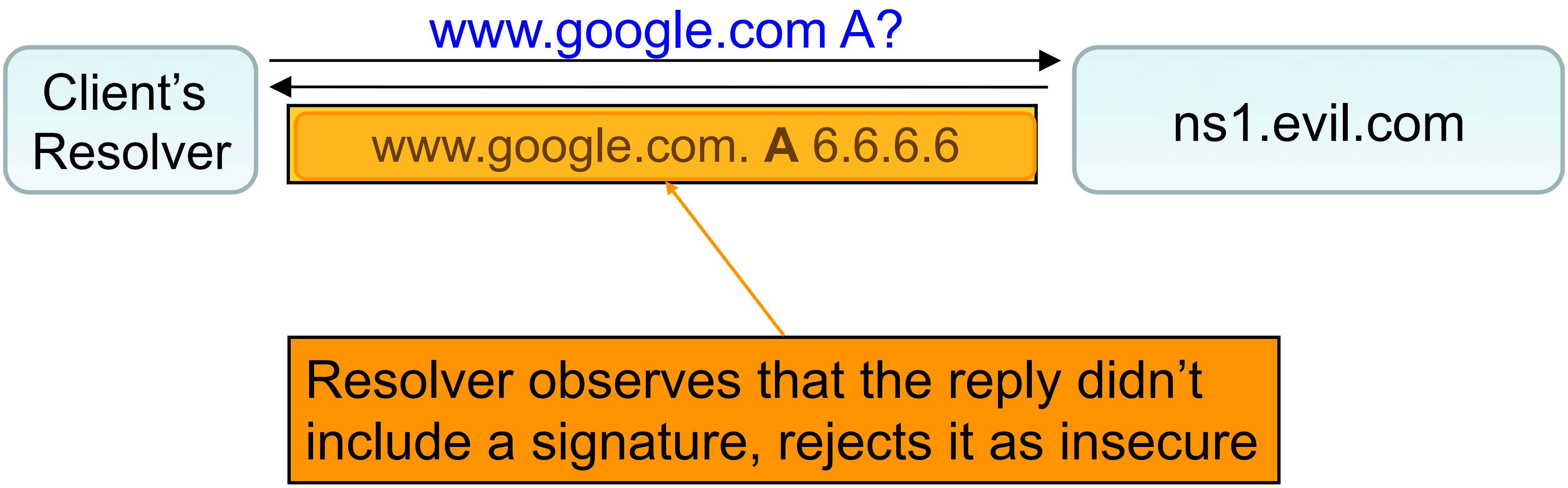
# DNSSEC (with simplifications):



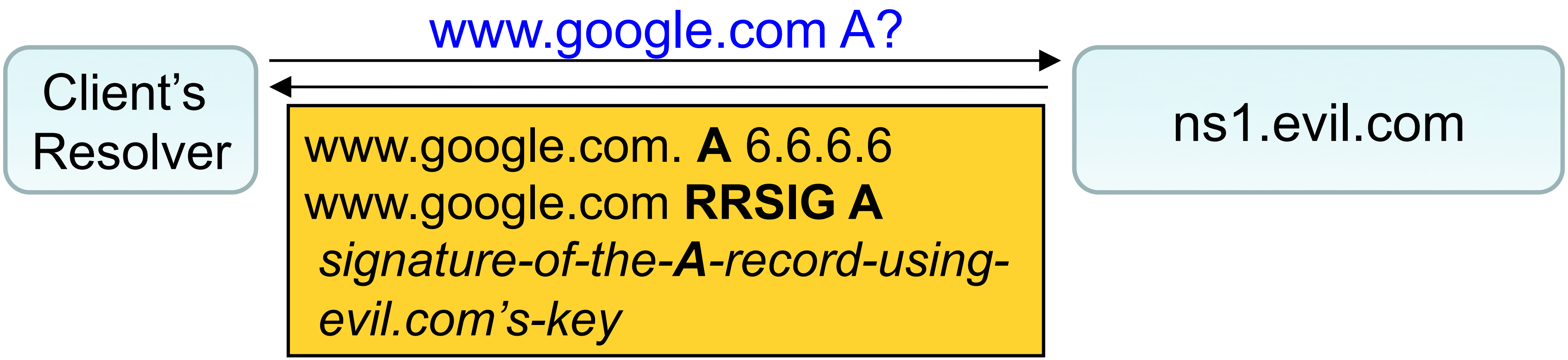
# DNSSEC – Mallory attacks!



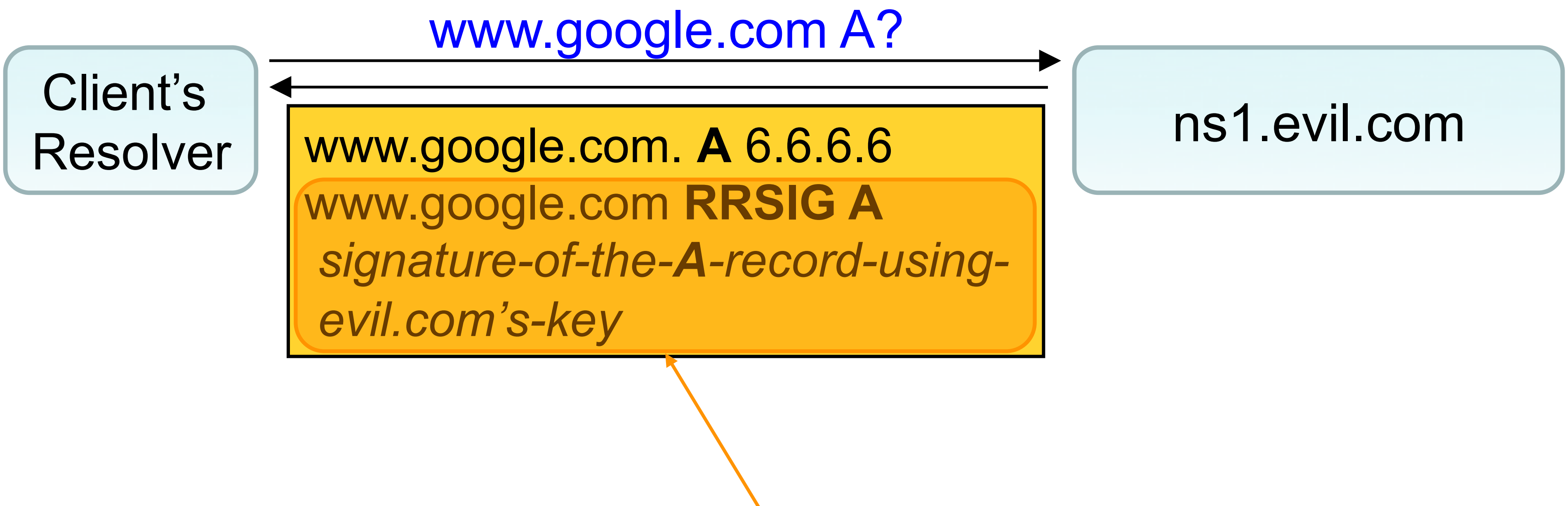
# DNSSEC – Mallory attacks!



# DNSSEC – Mallory attacks!

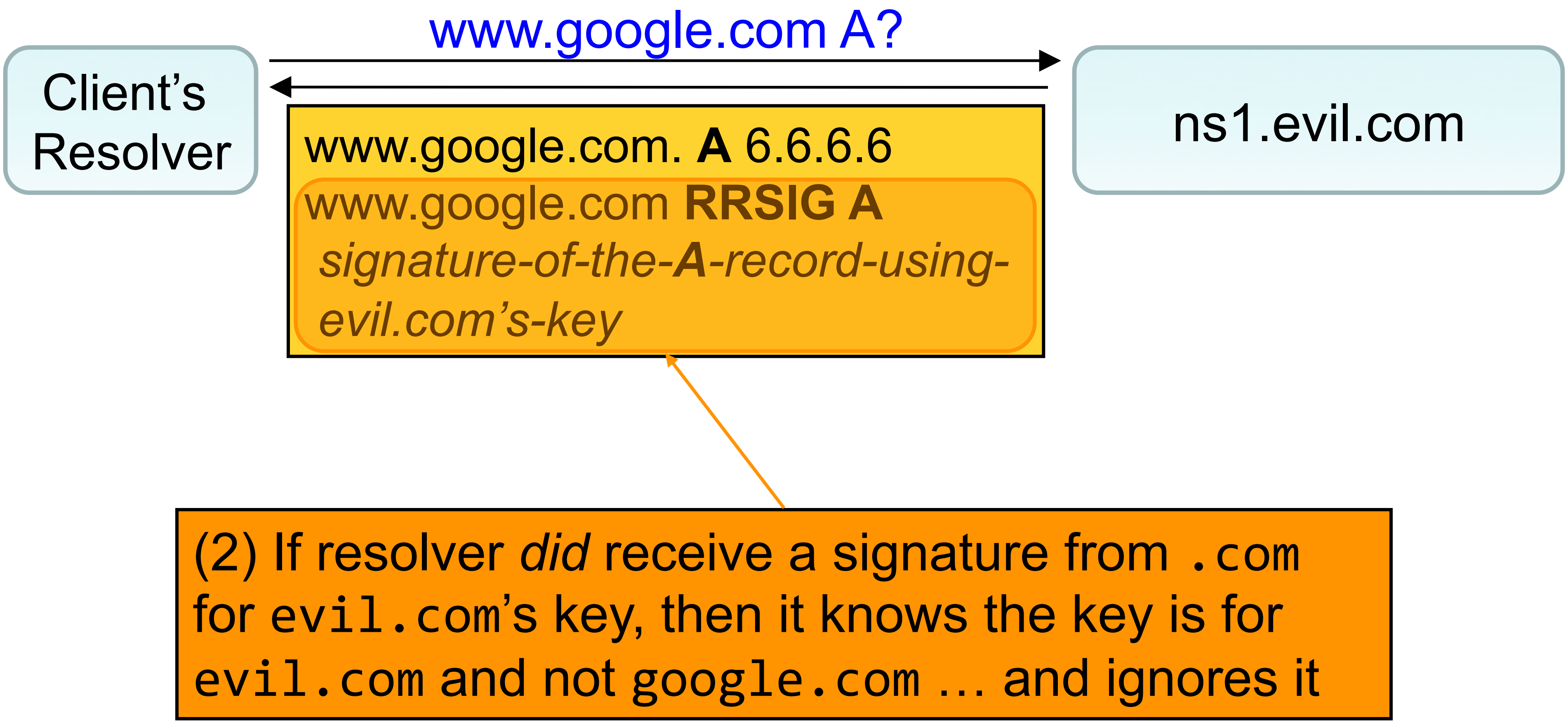


# DNSSEC – Mallory attacks!

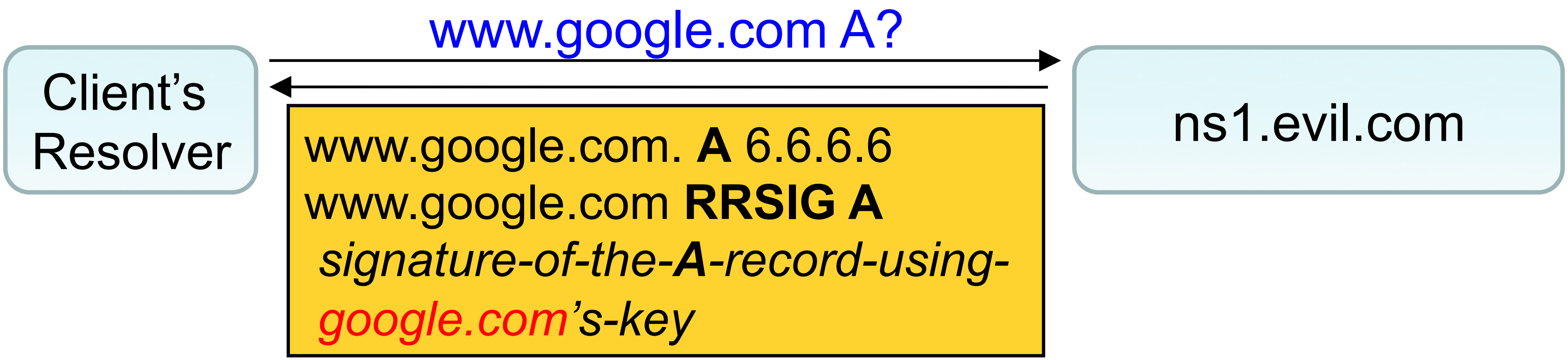



(1) If resolver didn't receive a signature from `.com` for `evil.com`'s key, then it can't validate this signature & ignores reply since it's not properly signed …

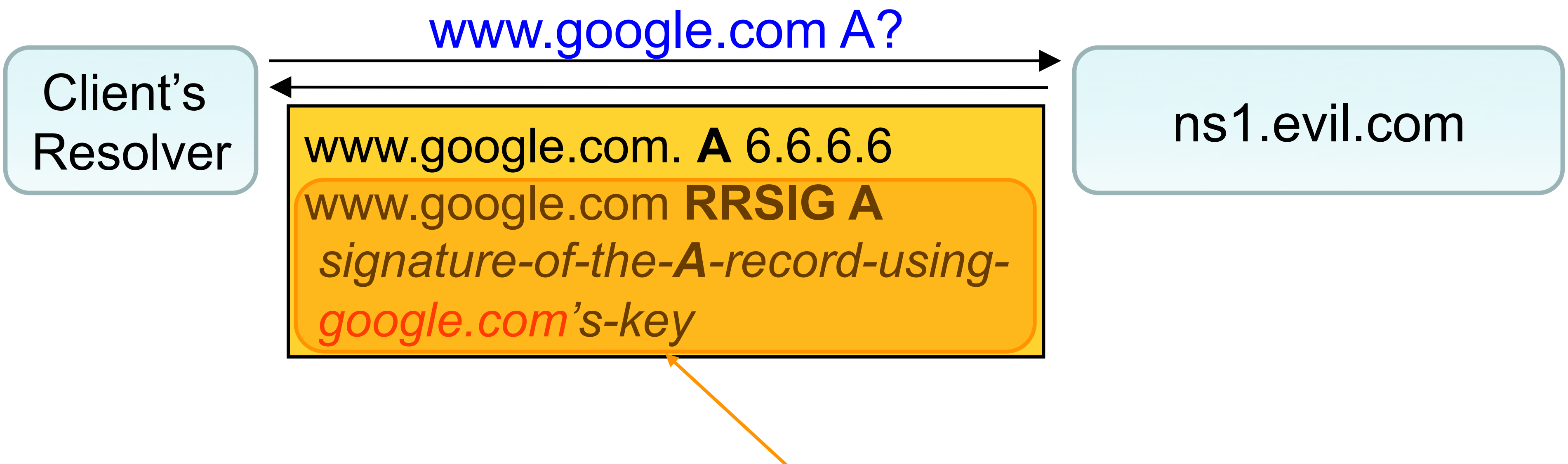
# DNSSEC – Mallory attacks!



# DNSSEC – Mallory attacks!



# DNSSEC – Mallory attacks!




If signature **actually** comes from `google.com`'s key, resolver will believe it …

… but no such signature should exist unless either:

(1) `google.com` *intended* to sign the RR, or

(2) `google.com`'s private key was compromised

# Issues With DNSSEC ?

- Issue #1: Replies are Big
  - E.g., "`dig +dnssec berkeley.edu`" can return 2100+ B
  - DoS amplification
  - Increased latency on low-capacity links
  - Headaches w/ older libraries that assume replies < 512B
- Issue #2: *Partial deployment*
  - Suppose `.com` not signing, though `google.com` is
  - Major practical concern. What do we do?
  - Can wire additional key into resolver (doesn't scale)
  - Or: outsource to trusted third party ("lookaside")
    - Wire their key into resolver, they sign numerous early adopters

# Issues With DNSSEC, cont.

- Issue #1: *Partial deployment*
  - Suppose `.com` not signing, though `google.com` is. Or, suppose `.com` and `google.com` are signing, but `cnn.com` isn't. Major practical concern. What do we do?
  - What do you do with unsigned/unvalidated results?
  - If you trust them, weakens incentive to upgrade (man-in-the-middle attacker can defeat security even for google.com, by sending forged but unsigned response)
  - If you don't trust them, a whole lot of things break

# Issues With DNSSEC, cont.

- Issue #2: Negative results ("no such name")
  - What statement does the nameserver sign?
  - If "`gabluph.google.com`" doesn't exist, then have to do dynamic key-signing (expensive) for any bogus request
  - Instead, sign (off-line) statements about order of names
    - E.g., sign "`gabby.google.com` is followed by `gabrunk.google.com`"
    - Thus, can see that `gabluph.google.com` can't exist
  - But: now attacker can enumerate all names that exist :-(

# Summary of TLS & DNSSEC Technologies

- **TLS**: provides channel security (for communication over TCP)
  - Confidentiality, integrity, authentication
  - Client & server agree on crypto, session keys
  - Underlying security dependent on:
    - Trust in Certificate Authorities / decisions to sign keys
    - (as well as implementors)
- **DNSSEC**: provides object security (for DNS results)
  - Just integrity & authentication, not confidentiality
  - No client/server setup "dialog"
  - Tailored to be caching-friendly
  - Underlying security dependent on trust in Root Name Server's key, and all other signing keys

# Summary of TLS & DNSSEC Technologies

- **TLS**: provides channel security (for communication over TCP)
  - Confidentiality, integrity, authentication
  - Client & server agree on crypto, session keys
  - Underlying security dependent on:
    - Trust in Certificate Authorities / decisions to sign keys
    - (as well as implementors)
- **DNSSEC**: provides object security (for DNS results)
  - Just integrity & authentication, not confidentiality
  - No client/server setup "dialog"
  - Tailored to be caching-friendly
  - Underlying security dependent on trust in Root Name Server's key, and all other signing keys

# Takeaways

- Channel security vs object security
- PKI organization should follow existing line of authority
- Adoption: two-sided adoption requirement makes tech transition tough; network effects

# A Tangent:
# How Can I Prove I Am Rich?

# Math Puzzle – Proof of Work

- **Problem.** To prove to Bob I'm not a spammer, Bob wants me to do 10 seconds of computation before I can send him an email. How can I prove to Bob that I wasted 10 seconds of CPU time, in a way that he can verify in milliseconds?

# Math Puzzle – Proof of Work

- **Problem.** To prove to Bob I'm not a spammer, Bob wants me to do 10 seconds of computation before I can send him an email. How can I prove to Bob that I wasted 10 seconds of CPU time, in a way that he can verify in milliseconds?

- Hint: Computing 1 billion SHA256 hashes might take 10 seconds.