Securing DNS: DNSSEC

CS 161: Computer Security

Prof. David Wagner

Special request: Please spread out!
Pair up. Each pair, sit far away from anyone else.
If you're just arriving, sit next to someone who is alone.

Securing DNS Lookups

- Topic for today:
  How can we ensure that when clients look up names with DNS, they can trust the answers they receive?
- But first, a diversion…

An Experiment

- Today: Active learning + peer instruction
  - I'm going to ask you to work out how to secure DNS, on your own.
  - I'll give you a series of problems. I want you to break into groups of two, decide what you think a solution might be, then report back to the class.
  - TAs and I will circulate. Ask us for help!
  - Research suggests this might be more effective than lecturing. Let's give it a try!
- This is an experiment – I need your feedback on whether it helps you learn.

Outsourcing Data Lookups

- **Problem 1.** Berkeley has a database of all its graduates, $D = \{d_1, d_2, \ldots, d_n\}$, replicated across many mirror sites. Given a name x, any client should be able to query any mirror and learn whether $x \in D$. We don't trust the mirrors, so if answer to query is "yes" (i.e., if $x \in D$), client should receive a proof that it can verify. If answer is "no" (i.e., $x \notin D$), no proof is necessary. Make performance as good as possible.

Solutions

Give to the mirror:

- Signatures: d1,Sign(H(d1)),…,dn,Sign(H(dn))
- Signatures: d1,Sign(d1),…,dn,Sign(dn)

Outsourcing Data Lookups

- **Question 2.** Suppose we use your solution, with client connecting to mirror via HTTP – but there is a man-in-the-middle (on-path attacker). What can attacker do, without being detected?

  A. Can spoof both "yes" ($x \in D$) and "no" ($x \notin D$) responses.
  B. Can spoof "yes", but can't spoof "no".
  C. Can spoof "no", but can't spoof "yes".
  D. Can't spoof either kind of response.

Authenticating “Yes” and “No”

- **Problem 3.** Same as Problem 1, except now, if answer is “no” (i.e., $x \notin D$), client should receive a proof that it can verify.

Authenticating "Yes" and "No"

- **Problem 3.** Same as Problem 1, except now, if answer is "no" (i.e., $x \notin D$), client should receive a proof that it can verify.

  Hint: Organize the elements as a binary tree or hash table, then….

Solutions

Say D = {Alice, Bob, Jim, Xavier}.
Give to mirror:

- Sign(1, Alice), Sign(2, Bob), Sign(3, Jim), Sign(4, Xavier)
- Sign(Alice,Bob), Sign(Bob, Jim), Sign(Jim,Xavier)

To answer query "Doug":

- Doug -> no, Bob, Jim, Sign(2, Bob), Sign(3, Jim); or Doug -> no, Sign(Bob, Jim)

DNS

- **Problem 4.** Now Berkeley wants to protect its DNS records; how could it do it? What would be the advantages and disadvantages of your solution?

DNSSEC

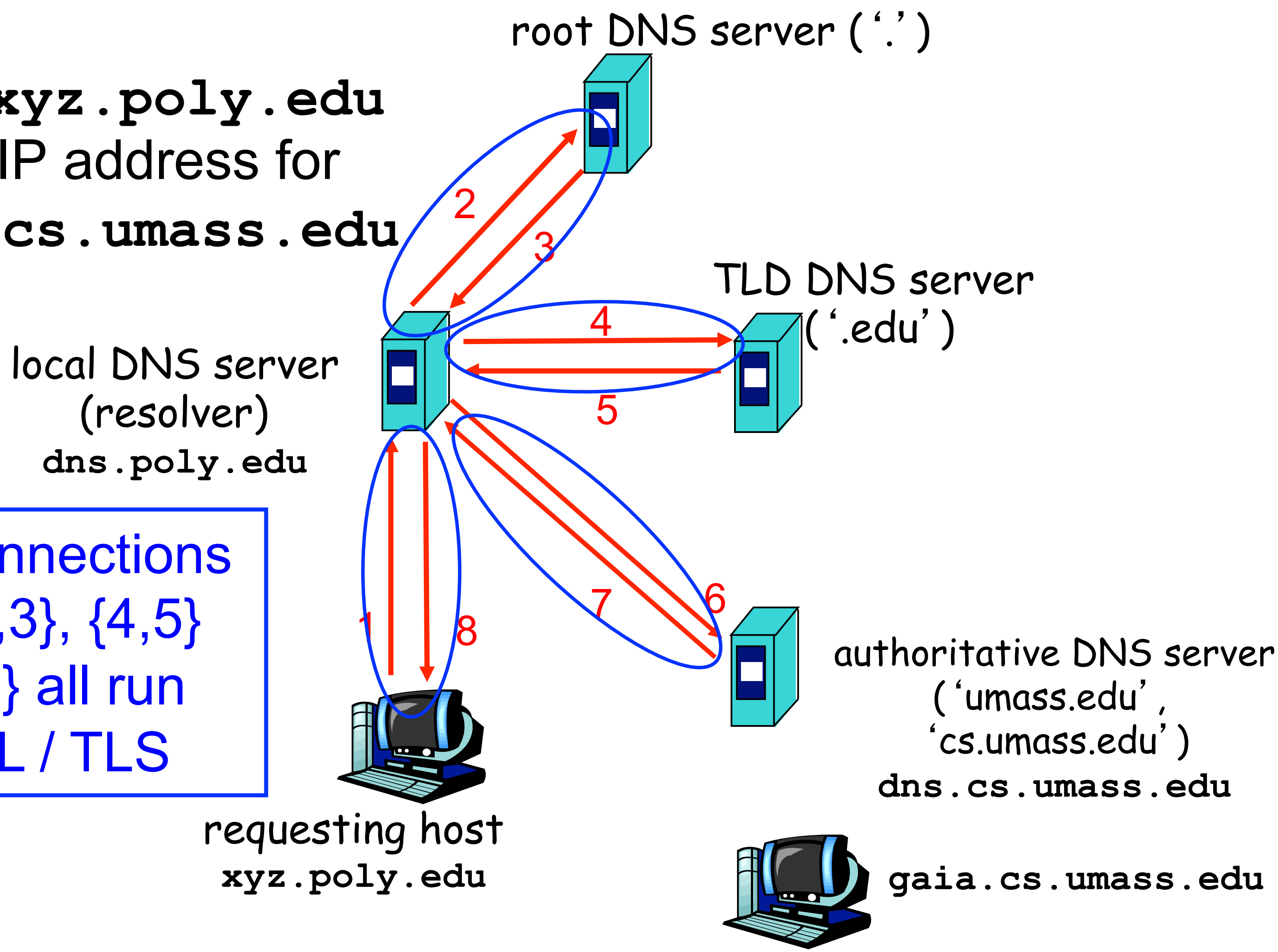
- Guess what – you just invented DNSSEC!
- Sign all DNS records. Signatures let you verify answer to DNS query, without having to trust the network or resolvers involved.

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 **`gaia.cs.umass.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

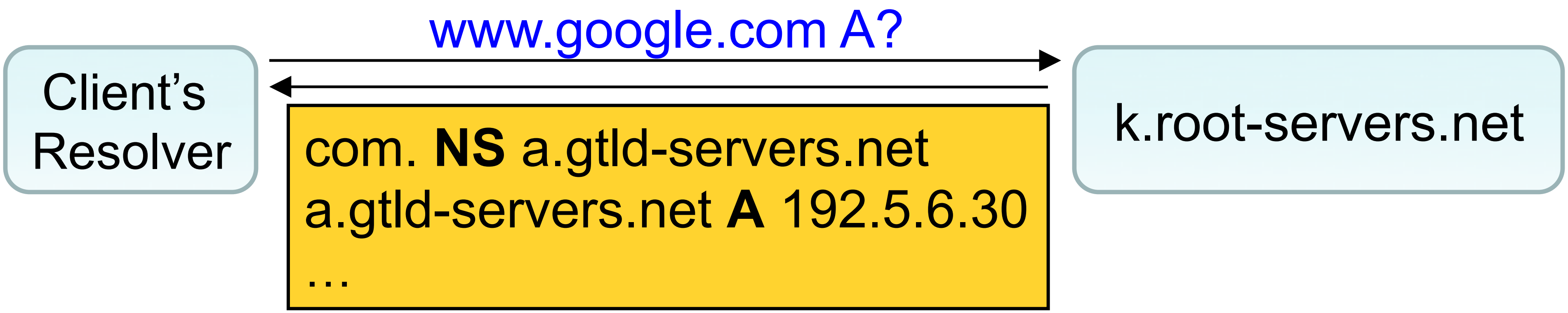
- DNSSEC = standardized DNS security extensions currently being deployed
- As a resolver works its way from DNS root down to final name server for a name, at each level it gets a signed statement regarding the key(s) used by the next level
  - This builds up a chain of trusted keys
  - Resolver has root's 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:

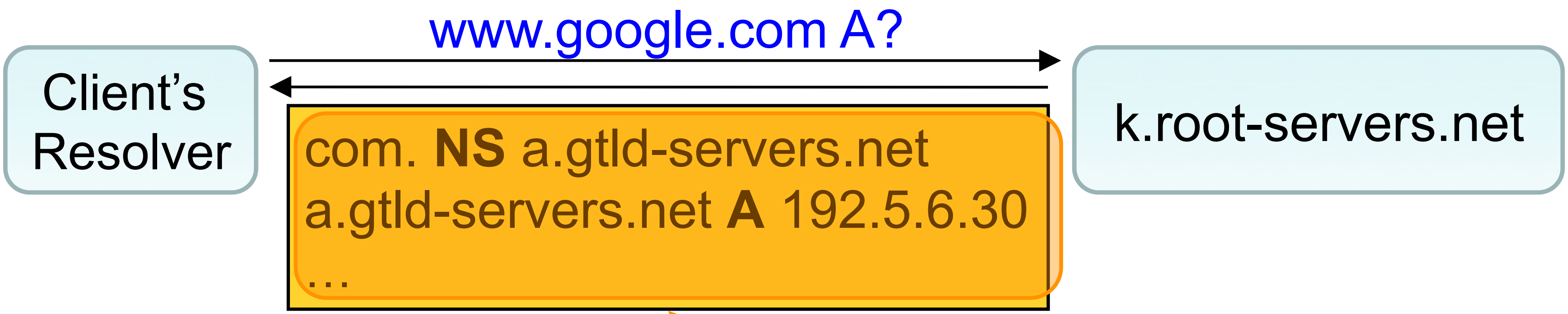
Ordinary DNS:

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:

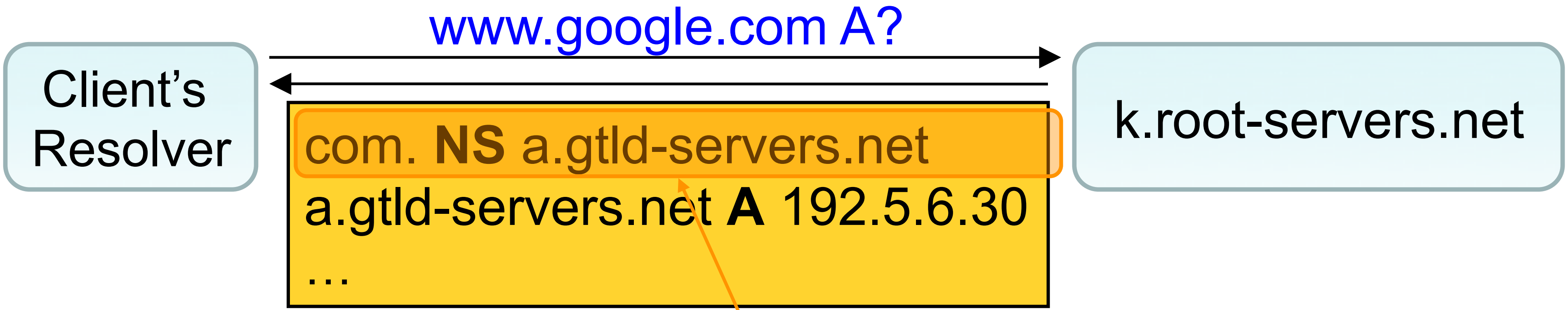


Ordinary DNS:



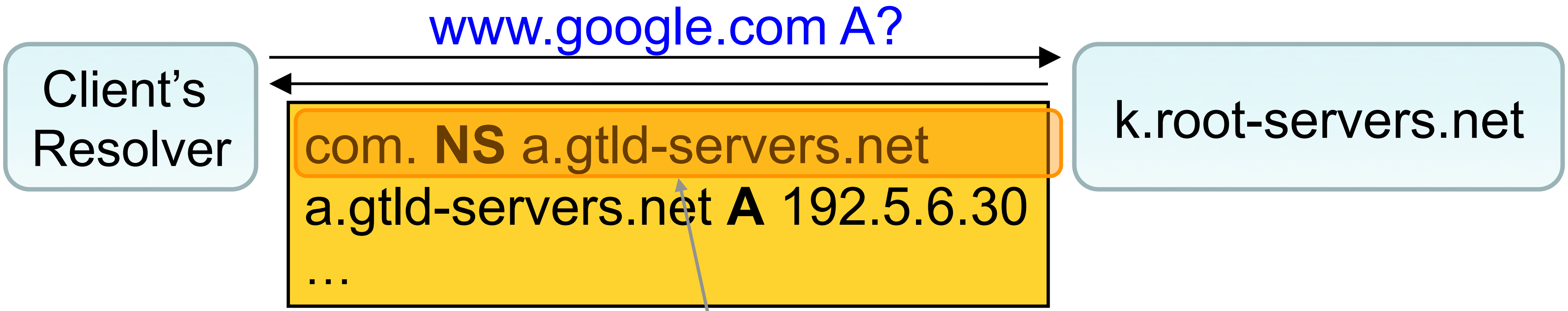
The reply *didn't include an answer* for `www.google.com`. That means that `k.root-servers.net` is instead telling us *where to ask next*, namely one of the name servers for `.com` specified in an **NS** record.

Ordinary DNS:



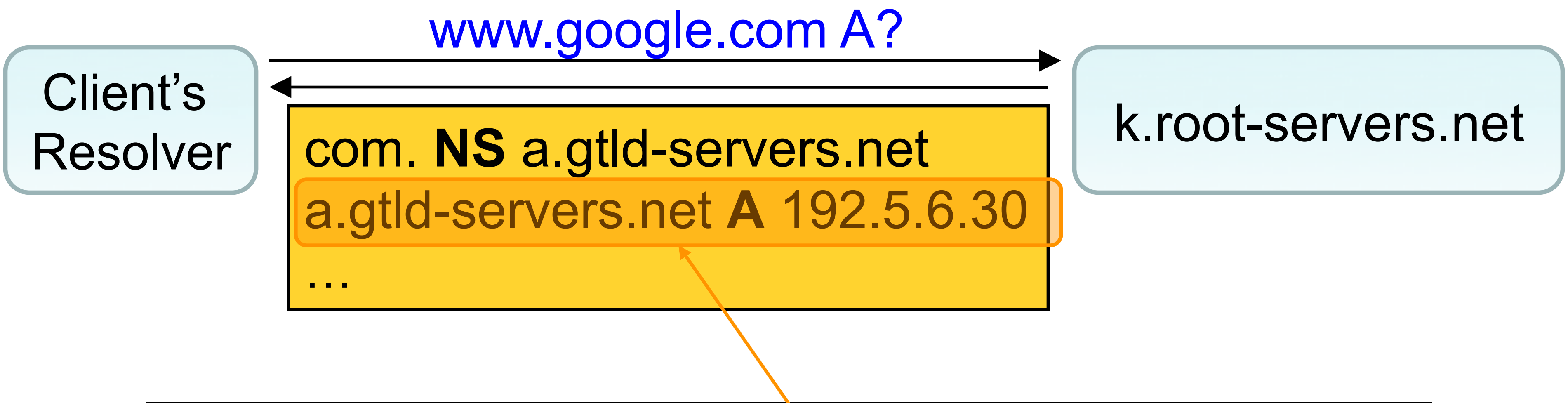
This *Resource Record* (**RR**) tells us that one of the name servers for `.com` is the host `a.gtld-servers.net`. (GTLD = Global Top Level Domain.)

Ordinary DNS:



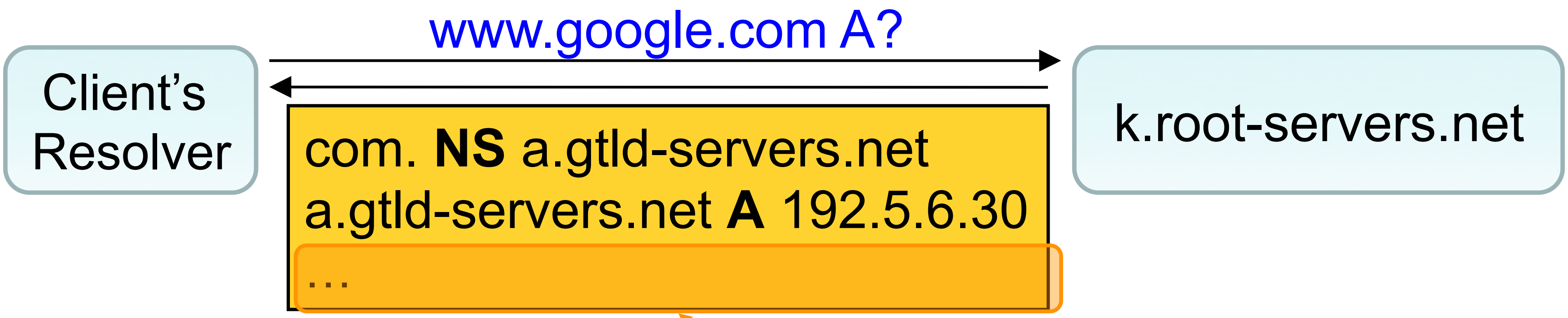
(The line above shows `com.` rather than `.com` because technically that's the actual name, and that's what the Unix `dig` utility shows; but the convention is to *call* it "dot-com")

Ordinary DNS:



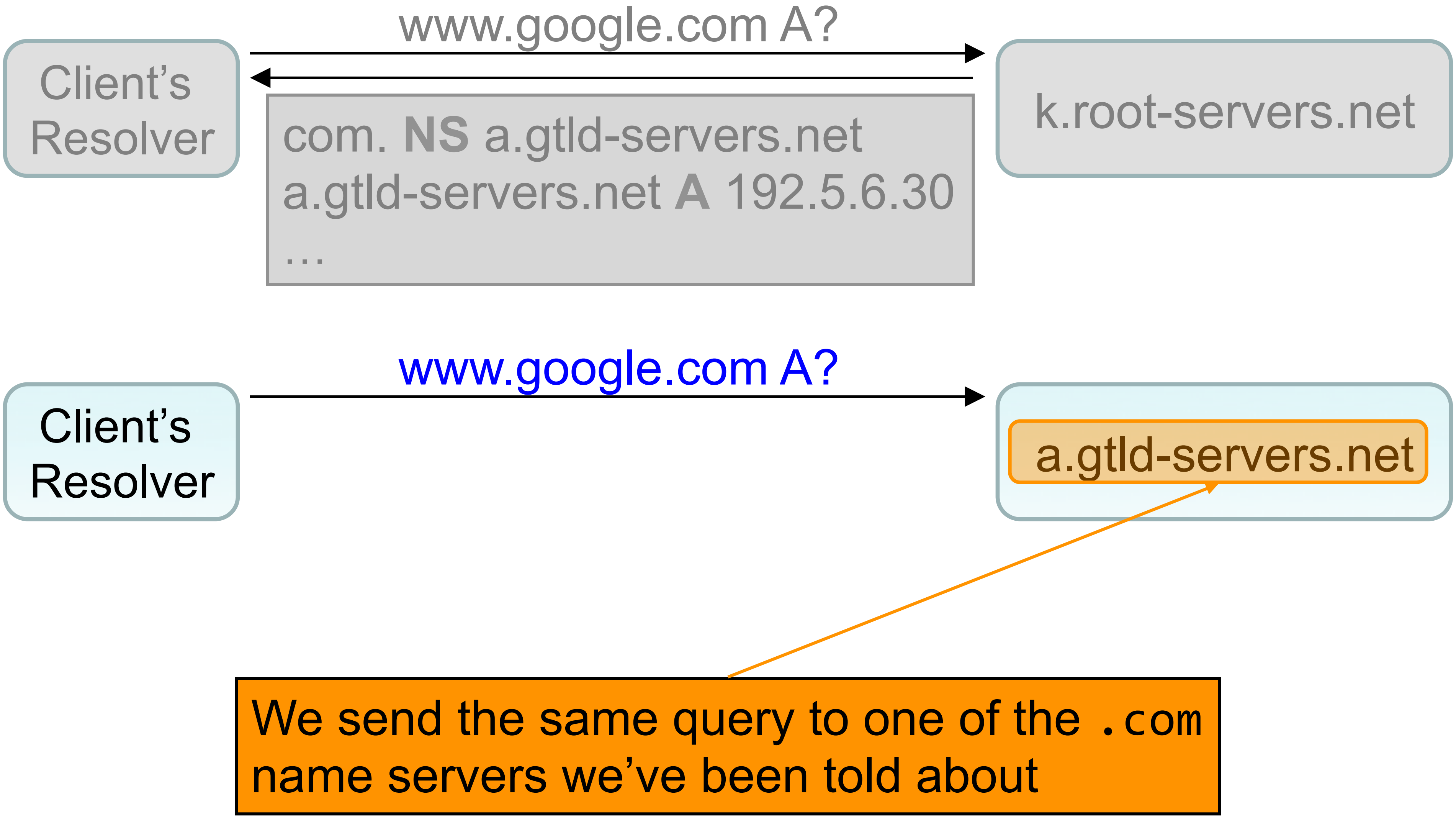
This **RR** tells us that an Internet address ("**A**" record) for `a.gtld-servers.net` is `192.5.6.30`. That allows us to know where to send our next query.

Ordinary DNS:

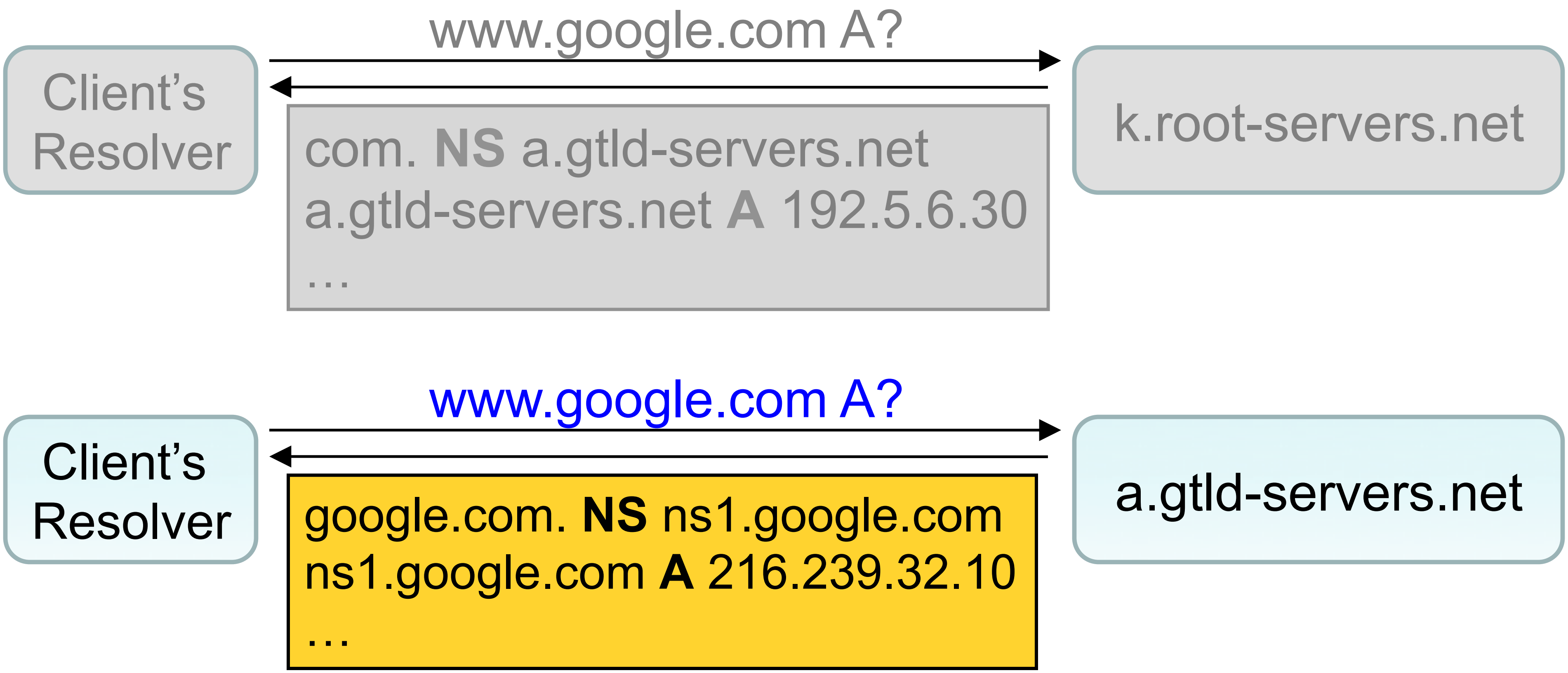


The actual response includes a bunch of **NS** and **A** records for additional `.com` name servers, which we omit here for simplicity.

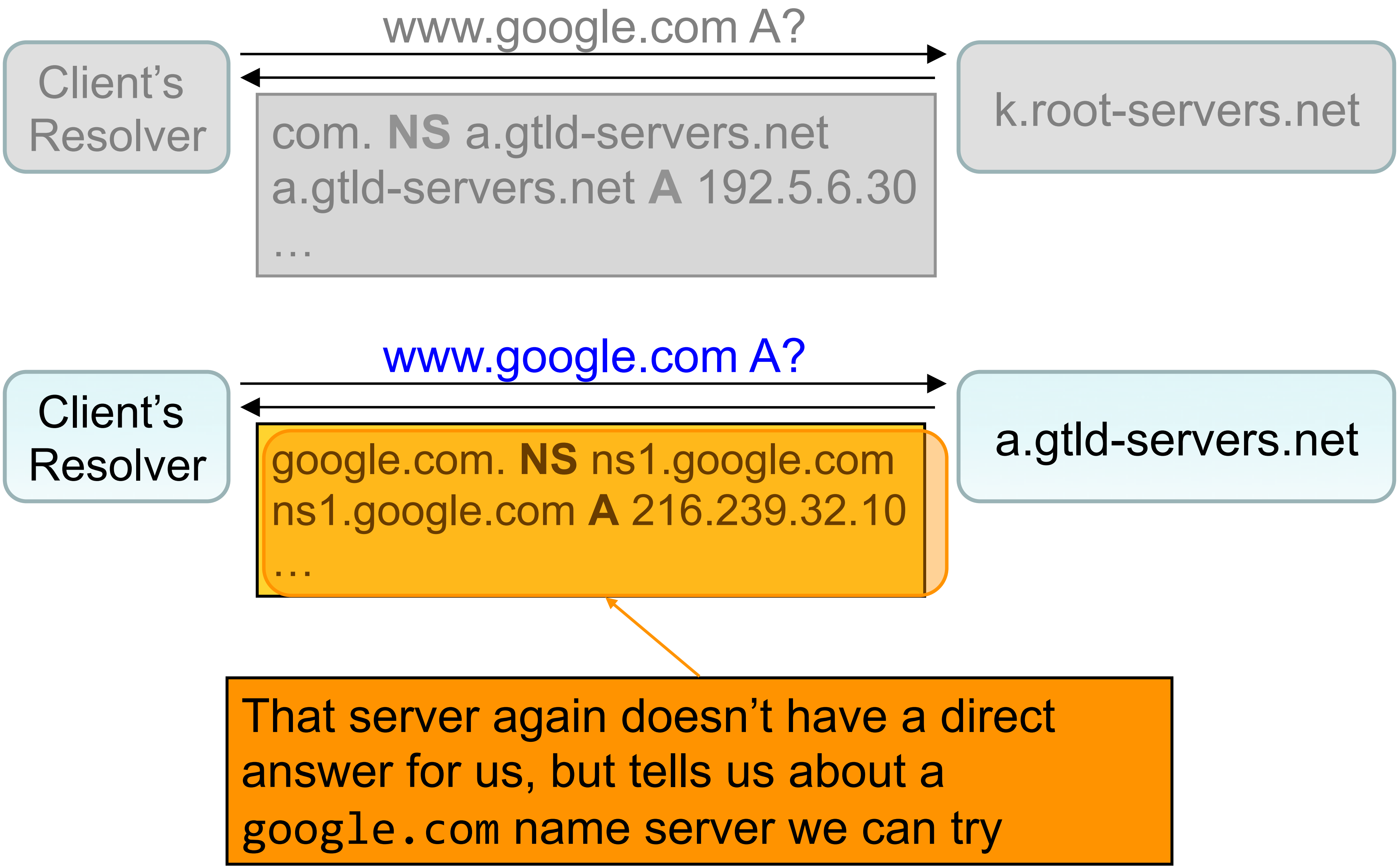
Ordinary DNS:



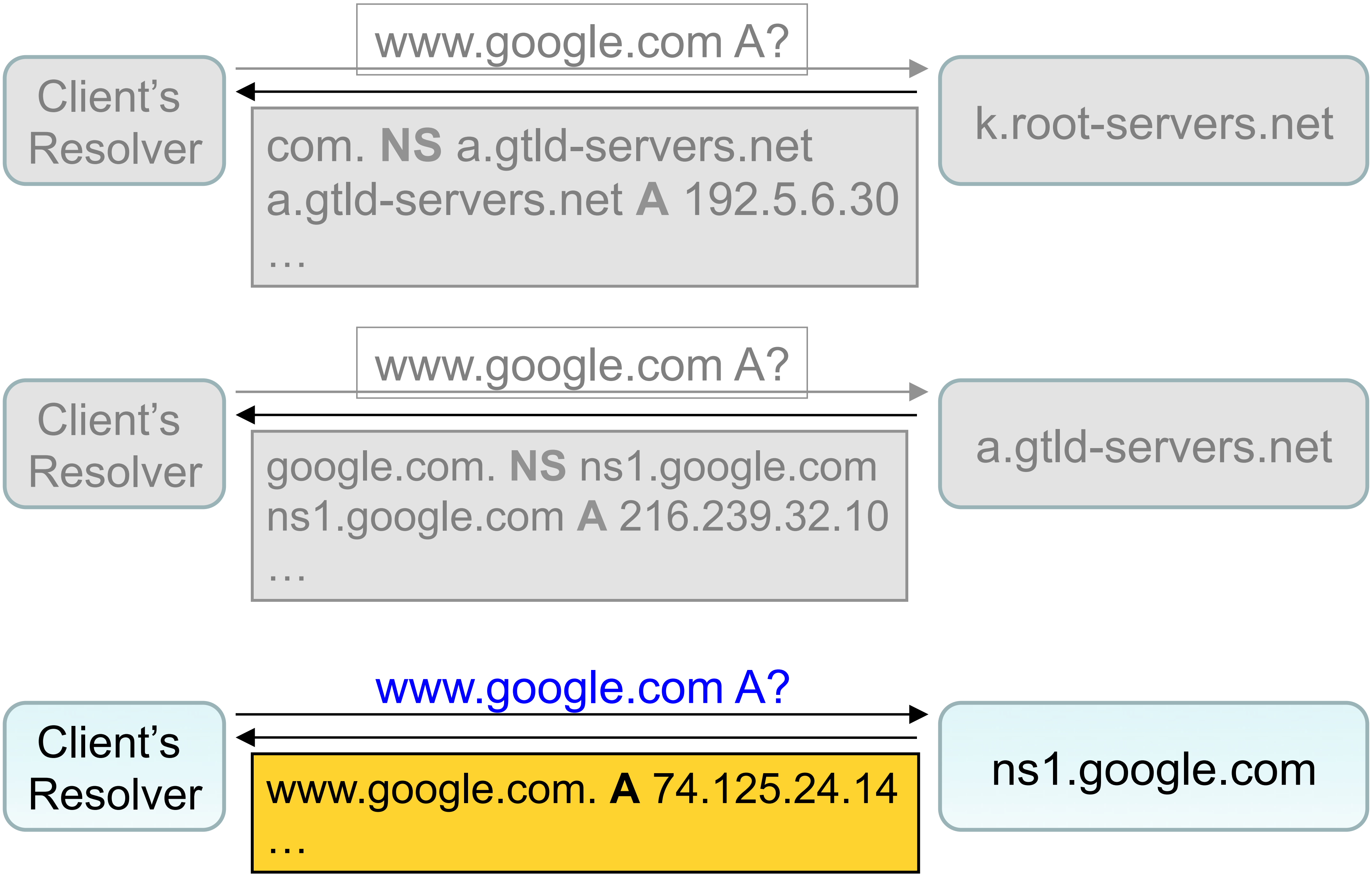
Ordinary DNS:



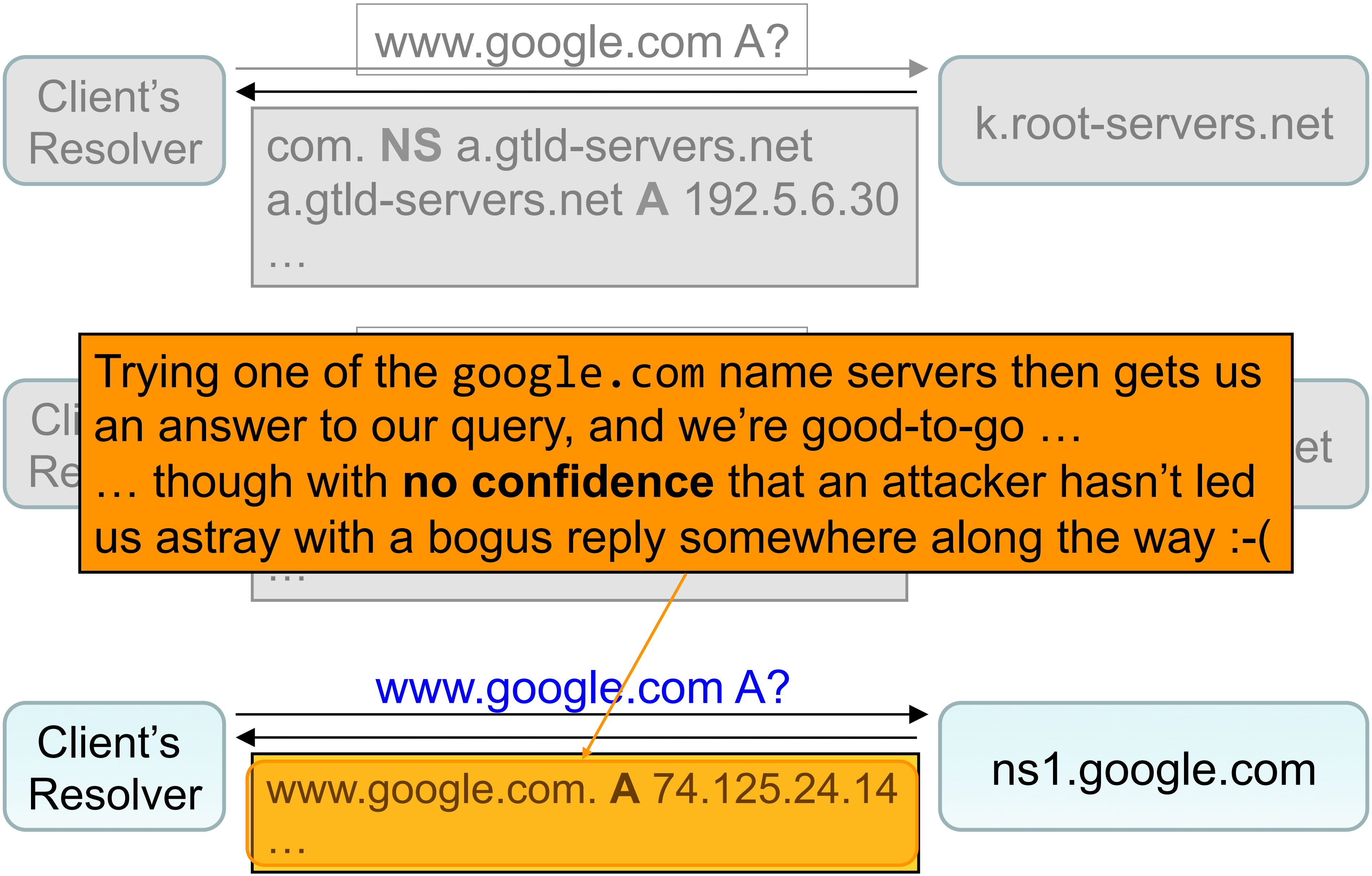
Ordinary DNS:



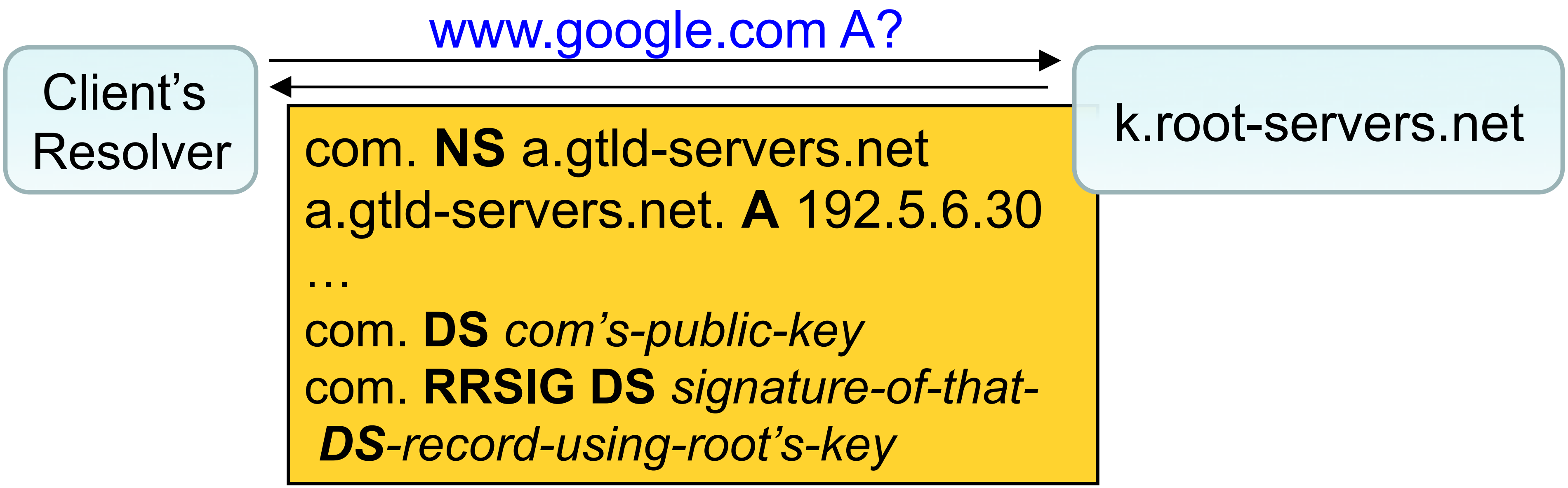
Ordinary DNS:



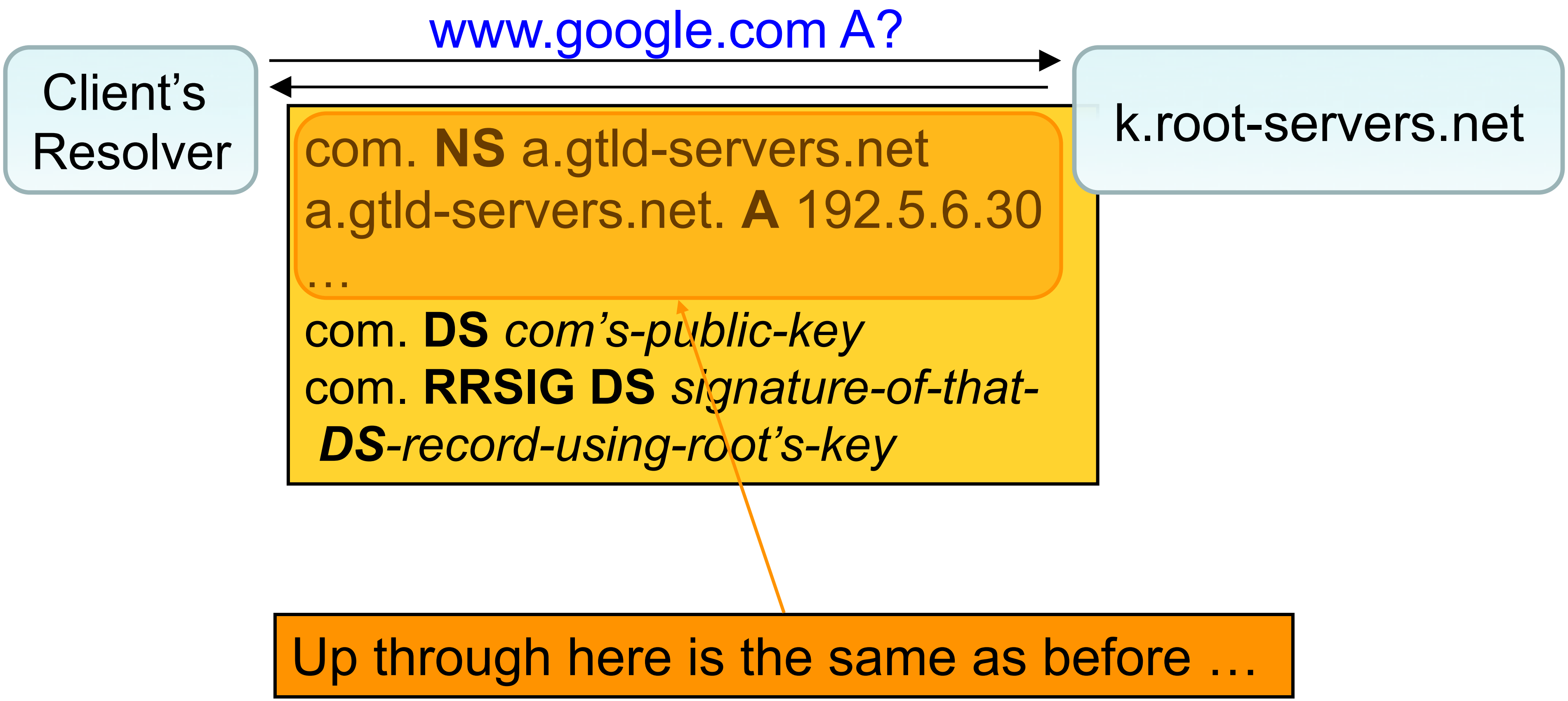
Ordinary DNS:



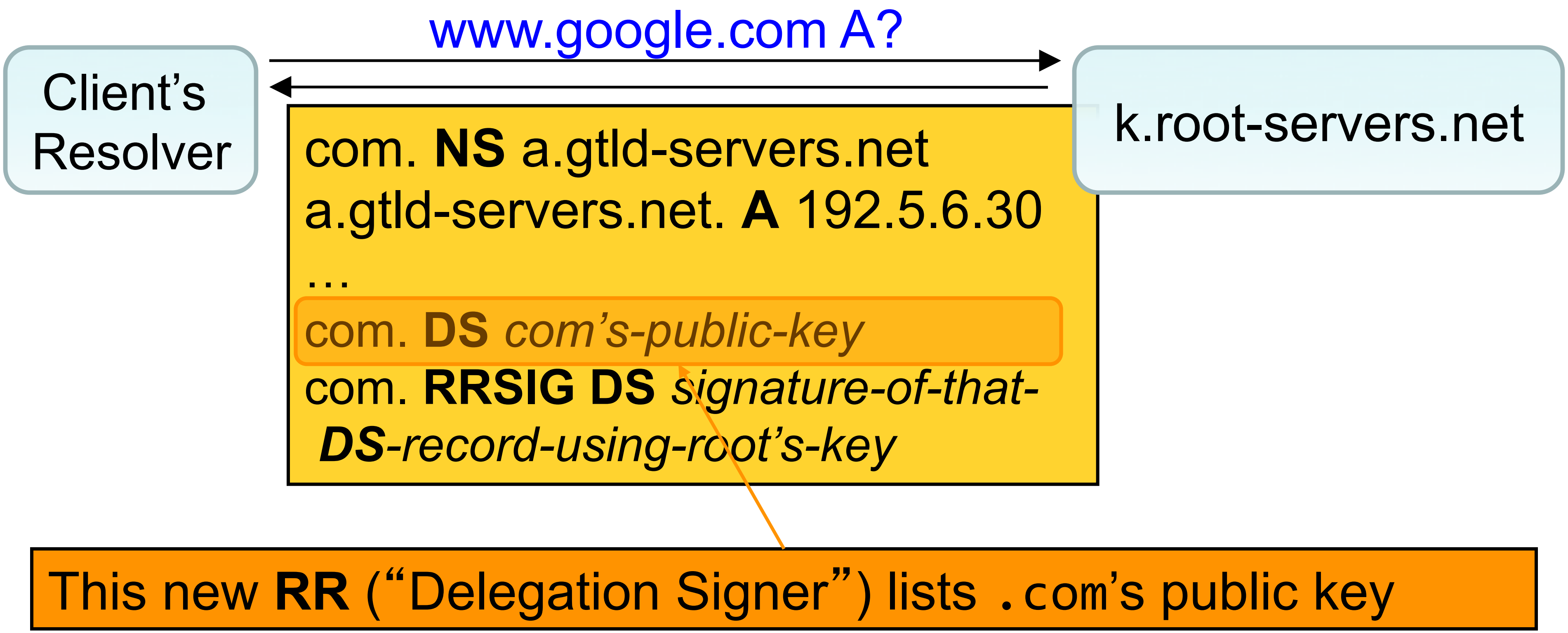
DNSSEC (with simplifications):



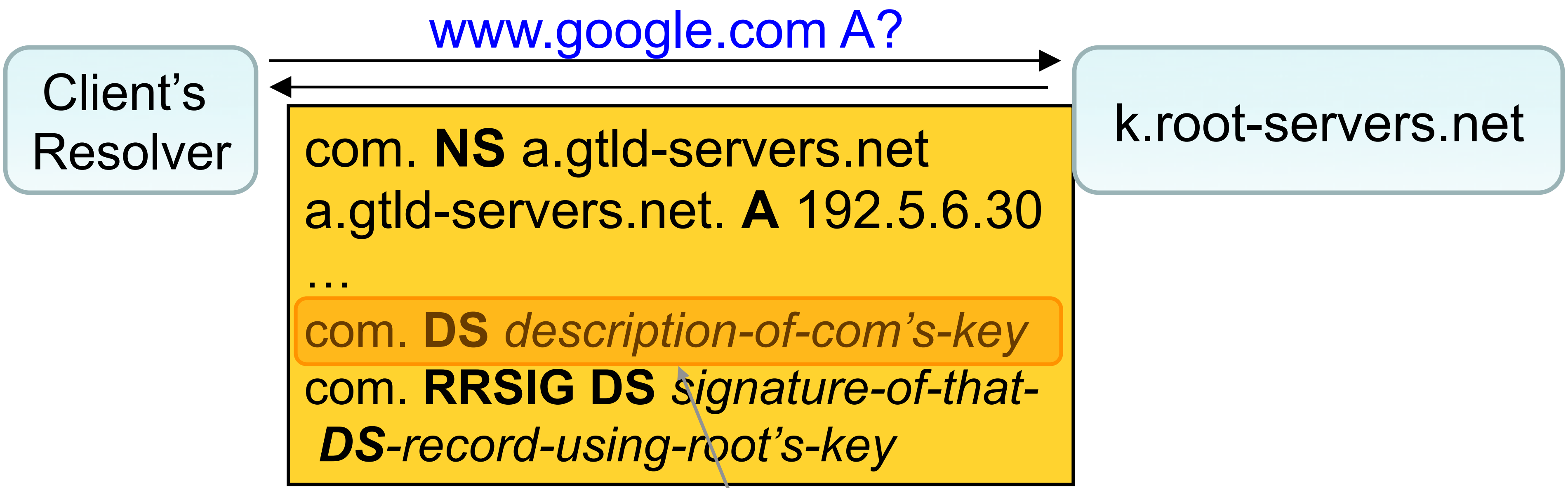
DNSSEC (with simplifications):



DNSSEC (with simplifications):

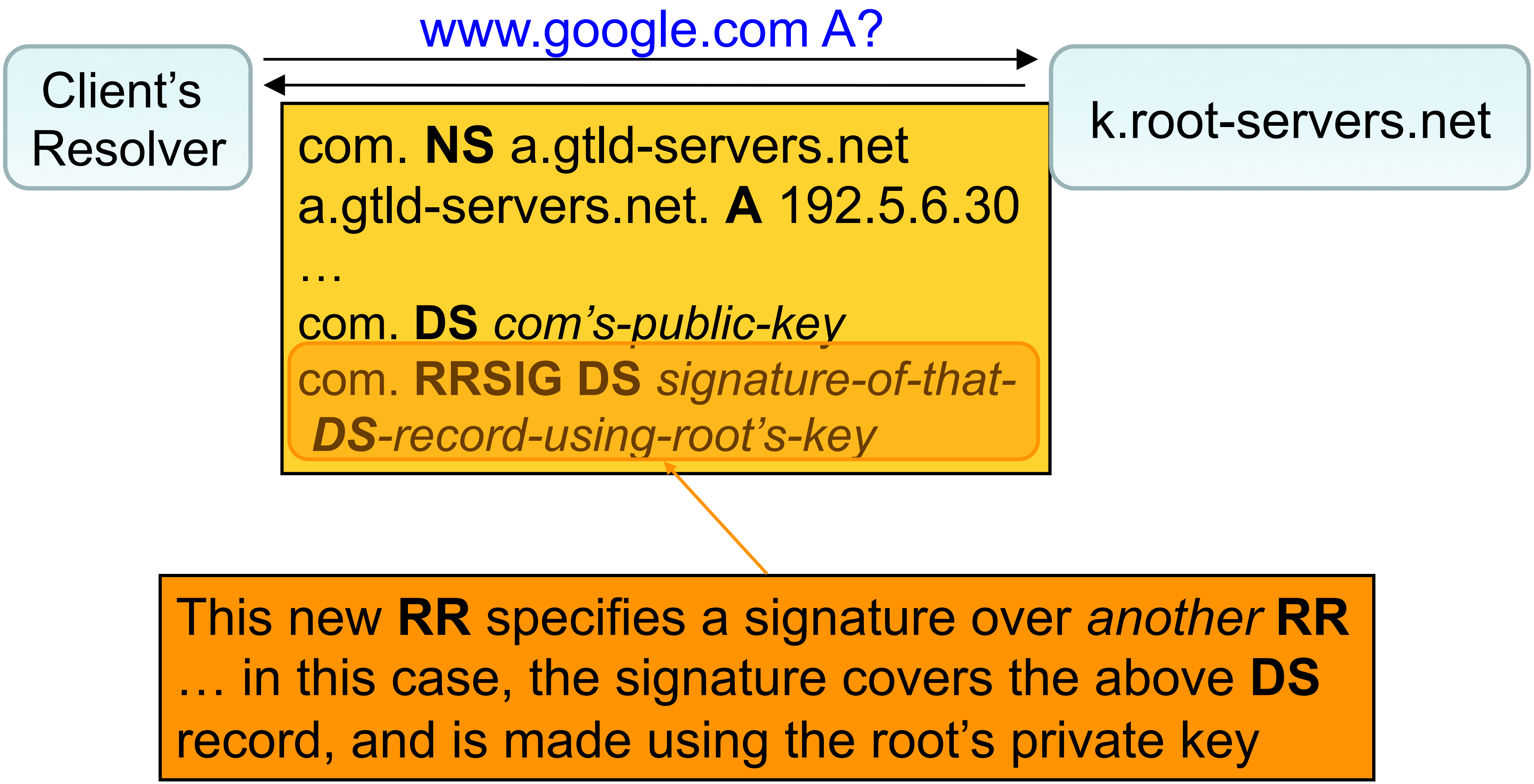


DNSSEC (with simplifications):

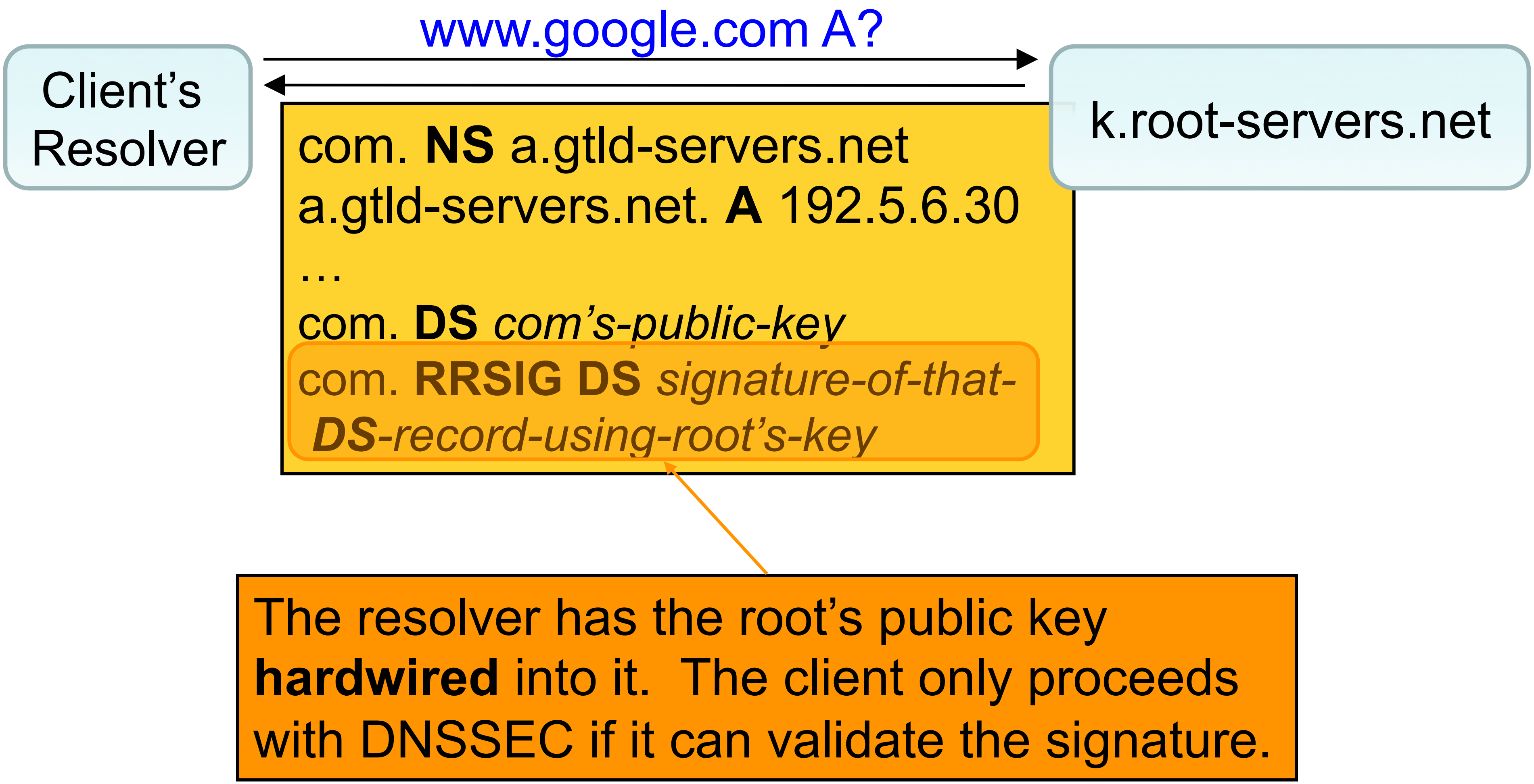


The actual process of retrieving .com's public key is complicated (actually involves multiple keys) but for our purposes doesn't change how things work

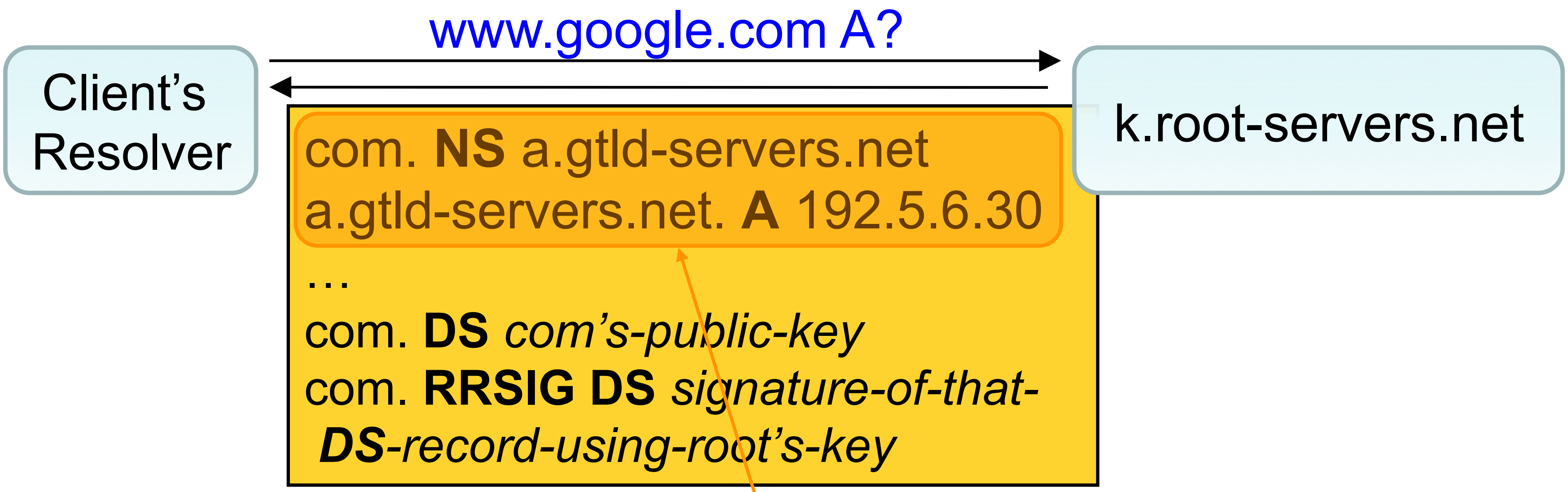
DNSSEC (with simplifications):



DNSSEC (with simplifications):



DNSSEC (with simplifications):



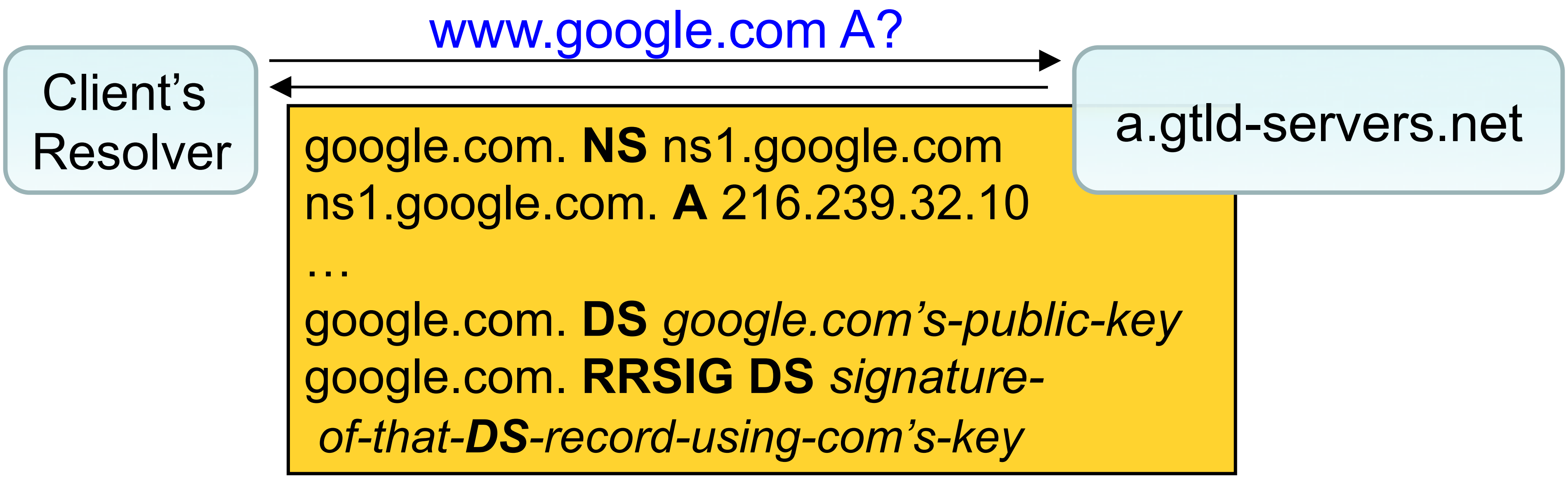
Note: there's no signature over the **NS** or **A** information! If an attacker has fiddled with those, the resolver will ultimately find it has a record for which it can't verify the signature.

DNSSEC (with simplifications):

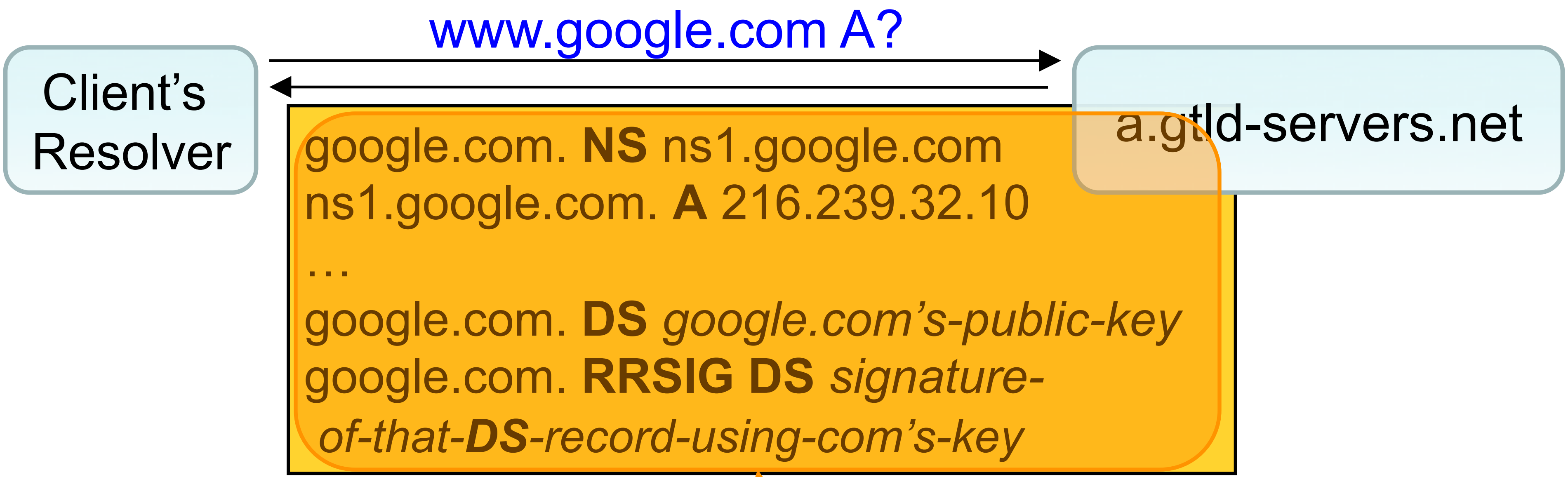
The resolver again proceeds to trying one of the name servers it's learned about.

Nothing guarantees this is a legitimate name server for the query!

DNSSEC (with simplifications):



DNSSEC (with simplifications):



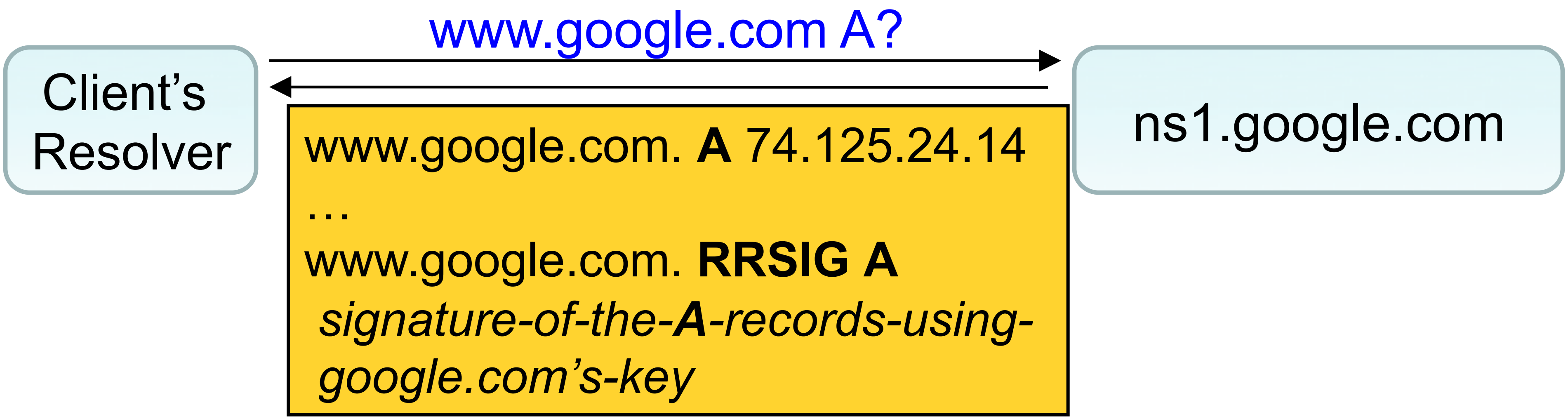
Back comes similar information as before: `google.com`'s public key, signed by `.com`'s key (which the resolver trusts because the root signed information about it)

DNSSEC (with simplifications):

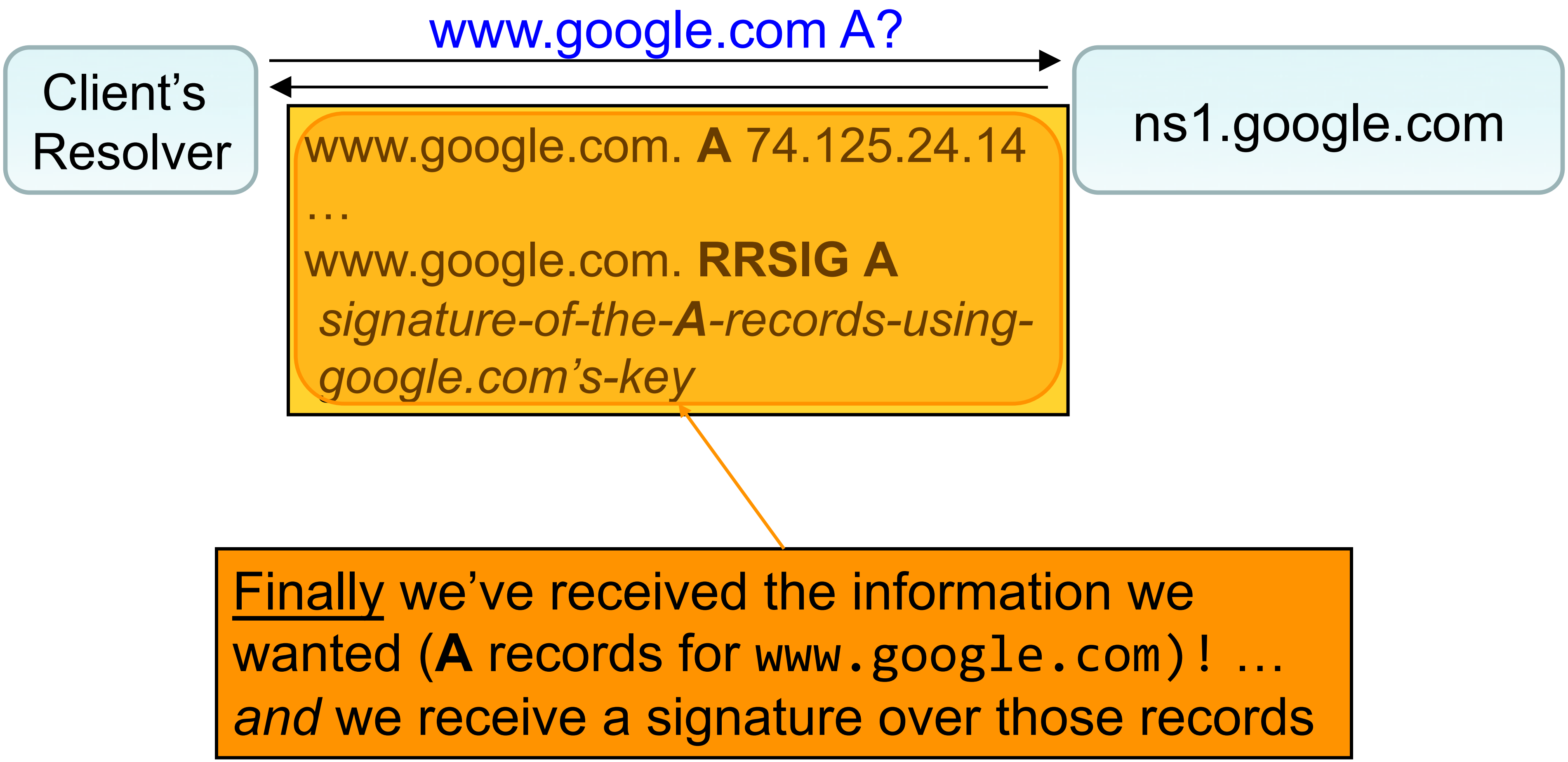
The resolver contacts one of the `google.com` name servers it's learned about.

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

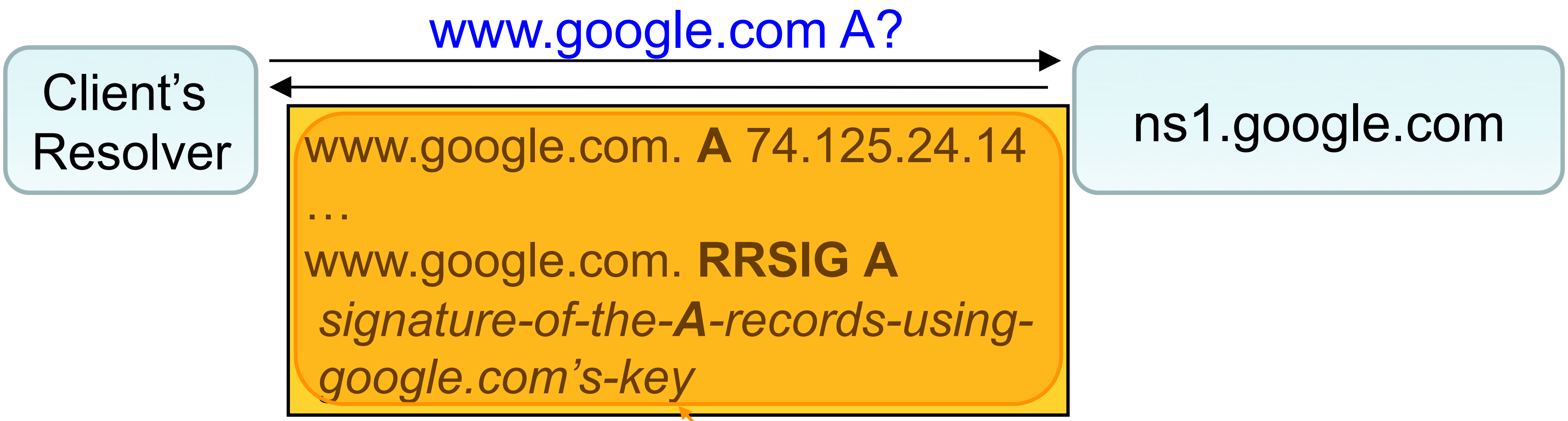
DNSSEC (with simplifications):



DNSSEC (with simplifications):



DNSSEC (with simplifications):



Assuming the signature validates, then because we believe (due to the signature chain) it's indeed from `google.com`'s key, we can trust that this is a correct set of **A** records … Regardless of what name server returned them to us!