Attacks on DNS

D. J. Bernstein
University of Illinois at Chicago

The Domain Name System

`cert.org` wants to see `http://www.lsec.be`.

Browser at `cert.org`

↑ "The web server `www.lsec.be` has IP address 81.246.94.54."

Administrator at `lsec.be`

Now `cert.org` retrieves web page from IP address 81.246.94.54.

s on DNS

Bernstein

sity of Illinois at Chicago

The Domain Name System

`cert.org` wants to see
`http://www.lsec.be`.

Browser at `cert.org`

"The web server
`www.lsec.be`
has IP address
81.246.94.54."

Administrator at `lsec.be`

Now `cert.org`
retrieves web page from
IP address 81.246.94.54.

Same f

`cert.o`
to deliv

Mail

Admin

Now ce
delivers
IP addr

ois at Chicago

The Domain Name System

`cert.org` wants to see `http://www.lsec.be`.

Browser at `cert.org`

↑

"The web server
`www.lsec.be`
has IP address
81.246.94.54."

Administrator at `lsec.be`

Now `cert.org` retrieves web page from IP address 81.246.94.54.

Same for Interne

`cert.org` has m
to deliver to `som`

Mail client at

↑

"The mail
`lsec`
has IP a
80.92.66

Administrator

Now `cert.org`
delivers mail to
IP address 80.92

cago

The Domain Name System

`cert.org` wants to see `http://www.lsec.be`.

Browser at `cert.org`

↑

“The web server
`www.lsec.be`
has IP address
81.246.94.54.”

Administrator at `lsec.be`

Now `cert.org`
retrieves web page from
IP address 81.246.94.54.

Same for Internet mail.

`cert.org` has mail
to deliver to `someone@lse`

Mail client at `cert.or`

↑

“The mail server for
`lsec.be`
has IP address
80.92.66.174.”

Administrator at `lsec.b`

Now `cert.org`
delivers mail to
IP address 80.92.66.174.

<u>The Domain Name System</u>

`cert.org` wants to see `http://www.lsec.be`.

Browser at `cert.org`

↑

"The web server `www.lsec.be` has IP address 81.246.94.54."

Administrator at `lsec.be`

Now `cert.org` retrieves web page from IP address 81.246.94.54.

Same for Internet mail.

`cert.org` has mail to deliver to `someone@lsec.be`.

Mail client at `cert.org`

↑

"The mail server for `lsec.be` has IP address 80.92.66.174."

Administrator at `lsec.be`

Now `cert.org` delivers mail to IP address 80.92.66.174.

omain Name System

org wants to see

//www.lsec.be.

wser at cert.org

“The web server

www.lsec.be

has IP address

81.246.94.54.”

istrator at lsec.be

ert.org

es web page from

ress 81.246.94.54.

Same for Internet mail.

cert.org has mail

to deliver to someone@lsec.be.

Mail client at cert.org

“The mail server for

lsec.be

has IP address

80.92.66.174.”

Administrator at lsec.be

Now cert.org

delivers mail to

IP address 80.92.66.174.

Forging

cert.c

to deliv

Mail c

Attac

Now ce

delivers

IP addr

actually

me System

to see

ec.be.

cert.org

server

c.be

dress

.54."

at `lsec.be`

ge from

6.94.54.

Same for Internet mail.

`cert.org` has mail

to deliver to `someone@lsec.be`.

Mail client at `cert.org`

"The mail server for

`lsec.be`

has IP address

80.92.66.174."

Administrator at `lsec.be`

Now `cert.org`

delivers mail to

IP address 80.92.66.174.

Forging DNS pac

`cert.org` has m

to deliver to `som`

Mail client at c

"The mail se

`lsec.b`

has IP ad

157.22.245

Attacker anyw

Now `cert.org`

delivers mail to

IP address 157.22

actually the attac

n

be

Same for Internet mail.

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

"The mail server for
`lsec.be`
has IP address
80.92.66.174."

Administrator at `lsec.be`

Now `cert.org`
delivers mail to
IP address 80.92.66.174.

Forging DNS packets

`cert.org` has mail
to deliver to `someone@lse`

Mail client at `cert.org`

"The mail server for
`lsec.be`
has IP address
157.22.245.20."

Attacker anywhere on n

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker's mac

Same for Internet mail.

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

"The mail server for
`lsec.be`
has IP address
80.92.66.174."

Administrator at `lsec.be`

Now `cert.org`
delivers mail to
IP address 80.92.66.174.

Forging DNS packets

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

"The mail server for
`lsec.be`
has IP address
157.22.245.20."

Attacker anywhere on network

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker's machine.

for Internet mail.

org has mail

ver to someone@lsec.be.

client at cert.org

“The mail server for

lsec.be

has IP address

80.92.66.174.”

nistrator at lsec.be

ert.org

s mail to

ress 80.92.66.174.

Forging DNS packets

cert.org has mail

to deliver to someone@lsec.be.

Mail client at cert.org

“The mail server for

lsec.be

has IP address

157.22.245.20.”

Attacker anywhere on network

Now cert.org

delivers mail to

IP address 157.22.245.20,

actually the attacker’s machine.

“Can a

t mail.

ail
eone@lsec.be.

t cert.org

server for
.be
address
6.174.”

at lsec.be

.66.174.

Forging DNS packets

cert.org has mail
to deliver to someone@lsec.be.

Mail client at cert.org

“The mail server for
lsec.be
has IP address
157.22.245.20.”

Attacker anywhere on network

Now cert.org
delivers mail to
IP address 157.22.245.20,
actually the attacker's machine.

“Can attackers d

c.be.

g

be

Forging DNS packets

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

↑

"The mail server for
`lsec.be`
has IP address
157.22.245.20."

Attacker anywhere on network

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker's machine.

"Can attackers do that?"

Forging DNS packets

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

“The mail server for
`lsec.be`
has IP address
157.22.245.20.”

Attacker anywhere on network

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker’s machine.

“Can attackers do that?”

<u>Forging DNS packets</u>

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

↑ "The mail server for `lsec.be` has IP address 157.22.245.20."

Attacker anywhere on network

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker's machine.

"Can attackers do that?"
— Yes.

Forging DNS packets

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

↑

"The mail server for
`lsec.be`
has IP address
157.22.245.20."

Attacker anywhere on network

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker's machine.

"Can attackers do that?"
— Yes.

"Really?"

Forging DNS packets

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

"The mail server for
`lsec.be`
has IP address
157.22.245.20."

Attacker anywhere on network

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker's machine.

"Can attackers do that?"
— Yes.

"Really?" — Yes.

Forging DNS packets

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

"The mail server for
`lsec.be`
has IP address
157.22.245.20."

Attacker anywhere on network

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker's machine.

"Can attackers do that?"
— Yes.

"Really?" — Yes.

"Don't the clients check
who's sending information?"

Forging DNS packets

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

“The mail server for
`lsec.be`
has IP address
157.22.245.20.”

Attacker anywhere on network

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker’s machine.

“Can attackers do that?”
— Yes.

“Really?” — Yes.

“Don’t the clients check
who’s sending information?”

— Yes, but the attacker
forges the sender address;
as easy as forging address
on a physically mailed postcard.

g DNS packets

org has mail
ver to someone@lsec.be.

lient at cert.org

"The mail server for
lsec.be
has IP address
157.22.245.20."

cker anywhere on network

ert.org
s mail to
ress 157.22.245.20,
y the attacker's machine.

"Can attackers do that?"
— Yes.

"Really?" — Yes.

"Don't the clients check
who's sending information?"

— Yes, but the attacker
forges the sender address;
as easy as forging address
on a physically mailed postcard.

Real po

From: 1
To:

The ma
IP addr
to have

Forged

From: 1
To:

The ma
IP addr
to have

ckets

ail

eone@lsec.be.

cert.org

erver for

be

dress

5.20."

vhere on network

2.245.20,

cker's machine.

"Can attackers do that?"

— Yes.

"Really?" — Yes.

"Don't the clients check
who's sending information?"

— Yes, but the attacker
forges the sender address;
as easy as forging address
on a physically mailed postcard.

Real postcard fro

From: lsec.be adr
To: cert.org
The mail server for
IP address 80.92.66
to have informed yo

Forged postcard

From: lsec.be adr
To: cert.org
The mail server for
IP address 157.22.2
to have informed yo

c.be.

network

chine.

"Can attackers do that?"
— Yes.

"Really?" — Yes.

"Don't the clients check who's sending information?"

— Yes, but the attacker forges the sender address; as easy as forging address on a physically mailed postcard.

Real postcard from admini

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` ha
IP address 80.92.66.174. Hopin
to have informed you sufficiently

Forged postcard from attac

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` ha
IP address 157.22.245.20. Hopi
to have informed you sufficiently

"Can attackers do that?"
— Yes.

"Really?" — Yes.

"Don't the clients check who's sending information?"

— Yes, but the attacker forges the sender address; as easy as forging address on a physically mailed postcard.

Real postcard from administrator:

> From: `lsec.be` admin
>
> To: `cert.org` mail client
>
> ---
>
> The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged postcard from attacker:

> From: `lsec.be` admin
>
> To: `cert.org` mail client
>
> ---
>
> The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

ttackers do that?"

.

?" — Yes.

the clients check
sending information?"

, but the attacker
the sender address;
as forging address
ysically mailed postcard.

Real postcard from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged postcard from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

Real pa

From: l

To:

The ma

IP addr

to have

Forged

From: l

To:

The ma

IP addr

to have

o that?"

s.

ts check

formation?"

attacker

r address;

g address

nailed postcard.

Real postcard from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged postcard from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

Real packet from

From: `lsec.be` adr

To: `cert.org`

The mail server for

IP address 80.92.66

to have informed yo

Forged packet fro

From: `lsec.be` adr

To: `cert.org`

The mail server for

IP address 157.22.2

to have informed yo

?"

tcard.

Real postcard from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged postcard from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

Real packet from administr

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` ha

IP address 80.92.66.174. Hopin

to have informed you sufficiently

Forged packet from attack

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` ha

IP address 157.22.245.20. Hopi

to have informed you sufficiently

Real postcard from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged postcard from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

Real packet from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged packet from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

ostcard from administrator:

lsec.be admin

cert.org mail client

il server for lsec.be has

ess 80.92.66.174. Hoping

informed you sufficiently!

postcard from attacker:

lsec.be admin

cert.org mail client

il server for lsec.be has

ess 157.22.245.20. Hoping

informed you sufficiently!

Real packet from administrator:

From: lsec.be admin

To: cert.org mail client

The mail server for lsec.be has

IP address 80.92.66.174. Hoping

to have informed you sufficiently!

Forged packet from attacker:

From: lsec.be admin

To: cert.org mail client

The mail server for lsec.be has

IP address 157.22.245.20. Hoping

to have informed you sufficiently!

“Is the

listenin

lsec.b

om administrator:

min

mail client

---

lsec.be has

.174. Hoping

u sufficiently!

from attacker:

min

mail client

---

lsec.be has

45.20. Hoping

u sufficiently!

Real packet from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged packet from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

“Is the client alw

listening for the a

`lsec.be`?”

strator:

s

g

y!

cker:

s

ng

y!

Real packet from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged packet from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

"Is the client always listening for the address of `lsec.be`?"

Real packet from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged packet from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

"Is the client always listening for the address of `lsec.be`?"

Real packet from administrator:

From: `lsec.be` admin To: `cert.org` mail client
The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged packet from attacker:

From: `lsec.be` admin To: `cert.org` mail client
The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

“Is the client always listening for the address of `lsec.be`?”

— No.

When client wants to know address of `lsec.be`,
it sends a query
to the administrator,
and listens for the response.

Forged `lsec.be` information is effective *if* it arrives at this time.

acket from administrator:

lsec.be admin

cert.org mail client

il server for lsec.be has

ess 80.92.66.174. Hoping

informed you sufficiently!

packet from attacker:

lsec.be admin

cert.org mail client

il server for lsec.be has

ess 157.22.245.20. Hoping

informed you sufficiently!

"Is the client always listening for the address of `lsec.be`?"

— No.

When client wants to know address of `lsec.be`, it sends a query to the administrator, and listens for the response.

Forged `lsec.be` information is effective *if* it arrives at this time.

Many v

time fo

1. Atta

One of

arrive a

2. Pok

to trigg

3. Atta

a long

n administrator:

min
mail client
lsec.be has
.174. Hoping
ou sufficiently!

om attacker:

min
mail client
lsec.be has
45.20. Hoping
ou sufficiently!

"Is the client always listening for the address of `lsec.be`?"

— No.

When client wants to know address of `lsec.be`, it sends a query to the administrator, and listens for the response.

Forged `lsec.be` information is effective *if* it arrives at this time.

Many ways for at
time forgeries pro

1. Attack repeat
One of the forge
arrive at the righ

2. Poke the clien
to trigger a know

3. Attack caches
a long time in ad

rator:

s

g

y!

er:

s

ng

y!

"Is the client always listening for the address of `lsec.be`?"

— No.

When client wants to know address of `lsec.be`, it sends a query to the administrator, and listens for the response.

Forged `lsec.be` information is effective *if* it arrives at this time.

Many ways for attackers to time forgeries properly:

1. Attack repeatedly. One of the forgeries will arrive at the right time.

2. Poke the client to trigger a known lookup.

3. Attack caches a long time in advance.

"Is the client always listening for the address of `lsec.be`?"

— No.

When client wants to know address of `lsec.be`, it sends a query to the administrator, and listens for the response.

Forged `lsec.be` information is effective *if* it arrives at this time.

Many ways for attackers to time forgeries properly:

1. Attack repeatedly.
One of the forgeries will arrive at the right time.

2. Poke the client
to trigger a known lookup.

3. Attack caches
a long time in advance.

"Is the client always listening for the address of `lsec.be`?"

— No.

When client wants to know address of `lsec.be`,
it sends a query
to the administrator,
and listens for the response.

Forged `lsec.be` information is effective *if* it arrives at this time.

Many ways for attackers to time forgeries properly:

1. Attack repeatedly.
One of the forgeries will arrive at the right time.

2. Poke the client
to trigger a known lookup.

3. Attack caches
a long time in advance.

4. Easy, succeeds instantly:
Sniff the network.

client always
g for the address of
be?”

client wants to know
s of `lsec.be`,
s a query
administrator,
tens for the response.

`lsec.be` information is
ve *if* it arrives at this time.

Many ways for attackers to
time forgeries properly:

1. Attack repeatedly.
One of the forgeries will
arrive at the right time.

2. Poke the client
to trigger a known lookup.

3. Attack caches
a long time in advance.

4. Easy, succeeds instantly:
Sniff the network.

Br

DNS

Admi

Browse
DNS c

Cache
adminis
doesn’t

ays
address of

ts to know
be,

ator,
e response.

information is
ives at this time.

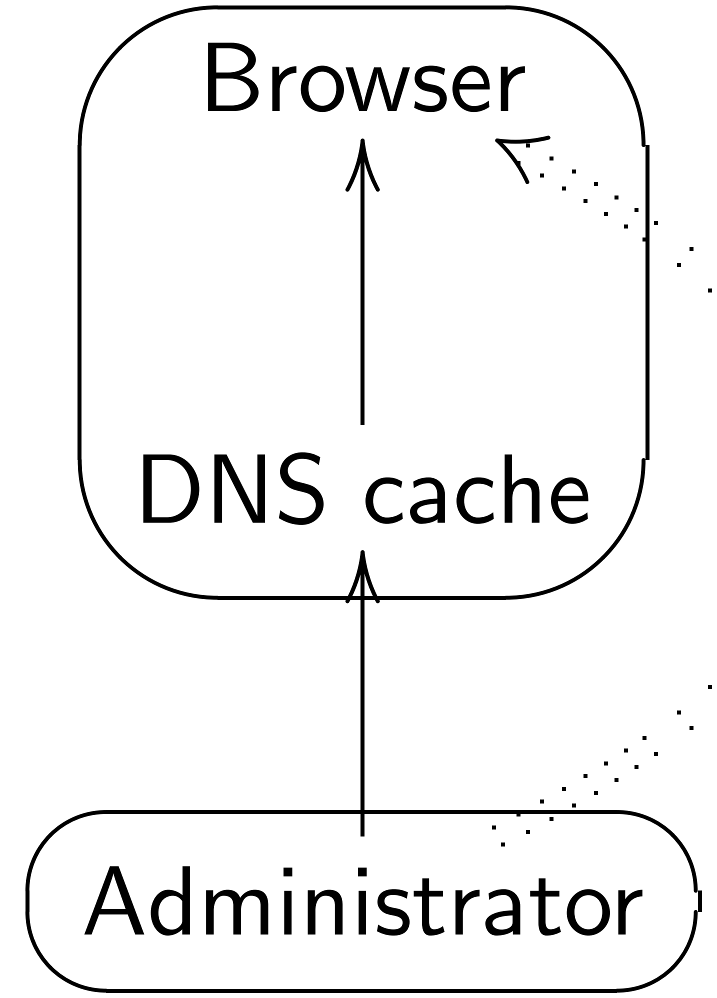
Many ways for attackers to
time forgeries properly:

1. Attack repeatedly.
One of the forgeries will
arrive at the right time.

2. Poke the client
to trigger a known lookup.

3. Attack caches
a long time in advance.

4. Easy, succeeds instantly:
Sniff the network.



Browser pulls dat
DNS cache at ce

Cache pulls data
administrator *if*
doesn't already h

v

e.

on is

s time.

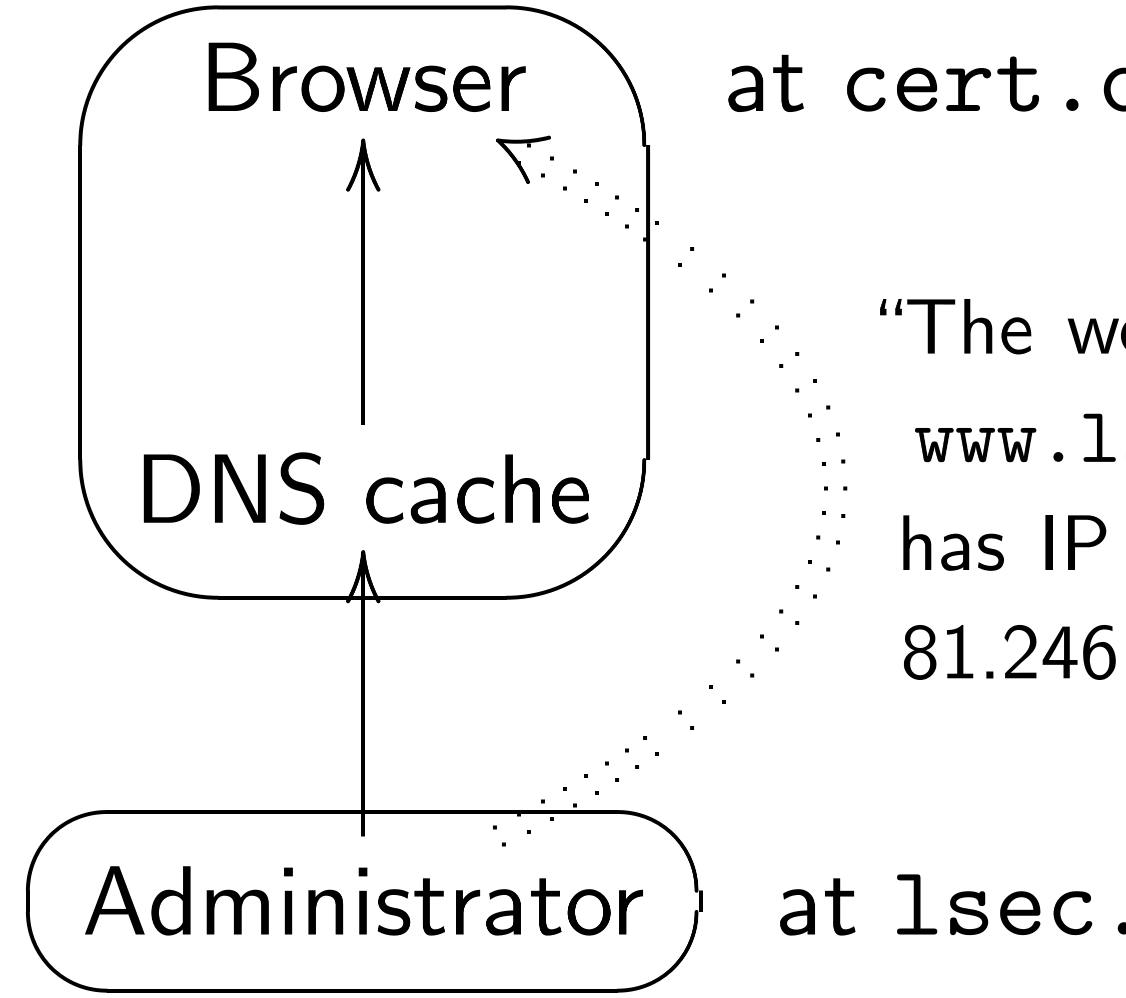
Many ways for attackers to time forgeries properly:

1. Attack repeatedly. One of the forgeries will arrive at the right time.

2. Poke the client to trigger a known lookup.

3. Attack caches a long time in advance.

4. Easy, succeeds instantly: Sniff the network.



Browser pulls data from DNS cache at `cert.org`.

Cache pulls data from administrator *if* it doesn't already have the da

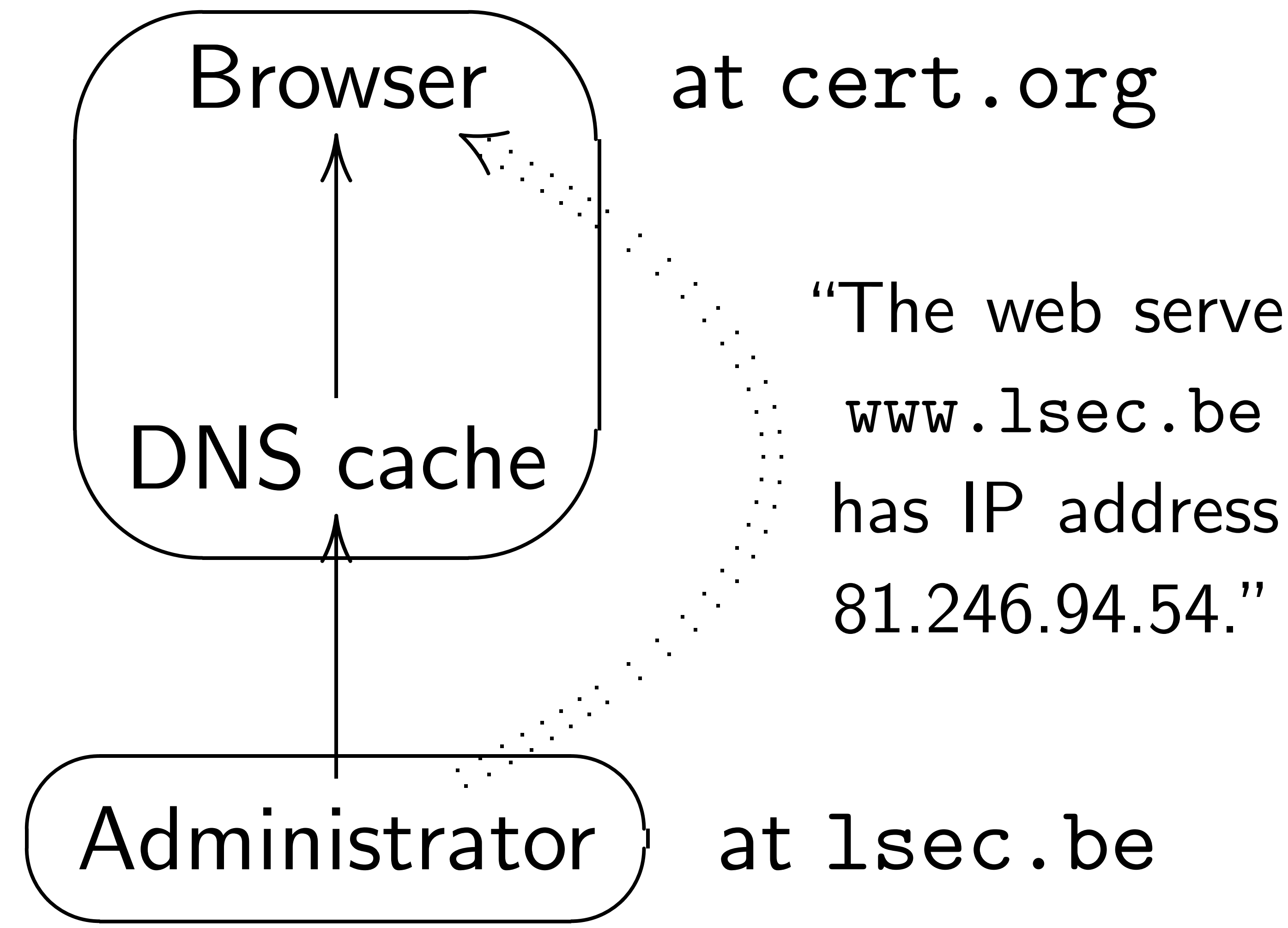
Many ways for attackers to time forgeries properly:

1. Attack repeatedly.
One of the forgeries will arrive at the right time.

2. Poke the client
to trigger a known lookup.

3. Attack caches
a long time in advance.

4. Easy, succeeds instantly:
Sniff the network.



Browser pulls data from DNS cache at `cert.org`.

Cache pulls data from administrator *if* it doesn't already have the data.

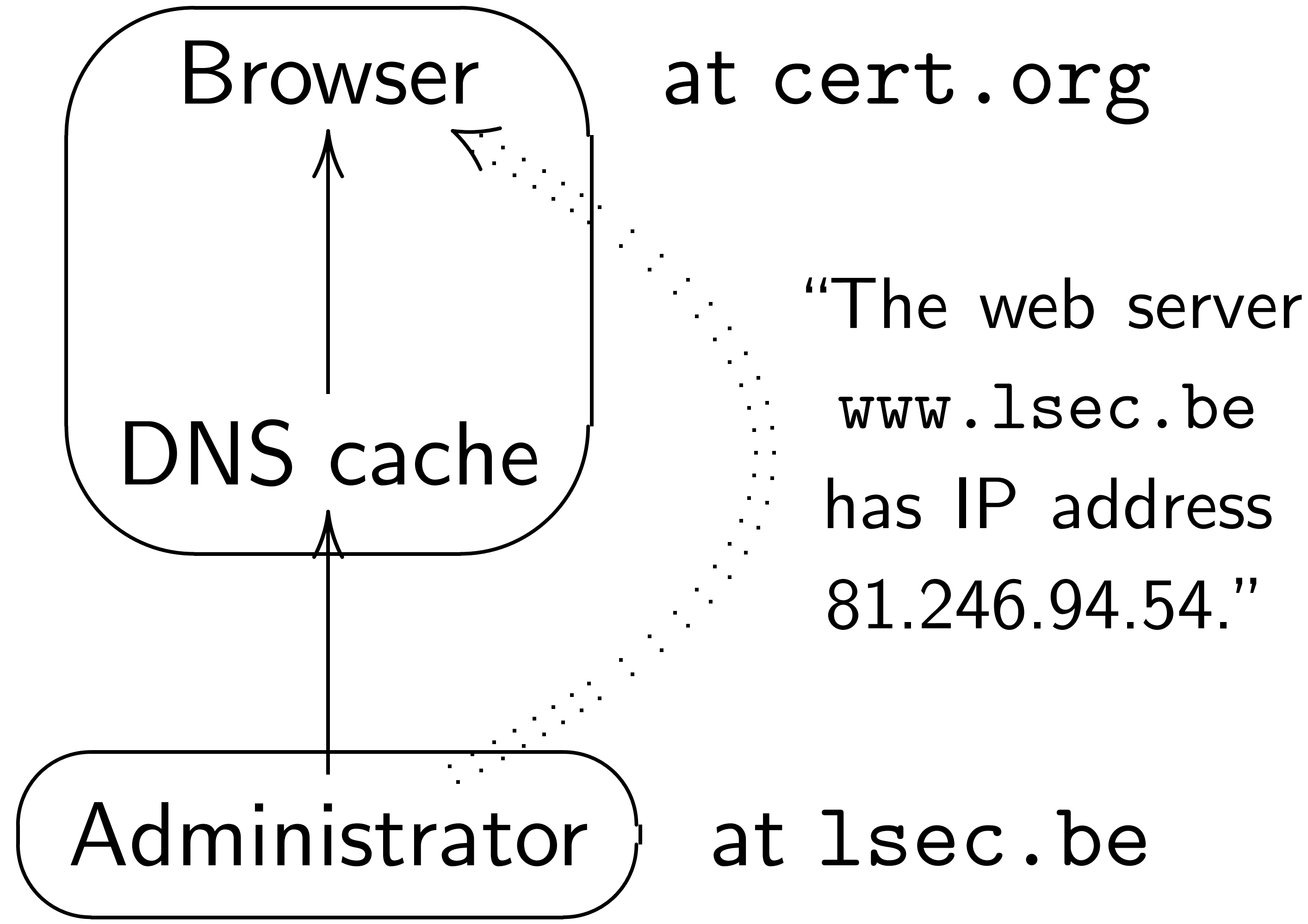
ways for attackers to
orgeries properly:

ack repeatedly.
f the forgeries will
at the right time.

e the client
ger a known lookup.

ack caches
time in advance.

y, succeeds instantly:
he network.



Browser pulls data from
DNS cache at `cert.org`.

Cache pulls data from
administrator *if* it
doesn't already have the data.

A typic

Attack
supers
includi
from w

Victim
supers

Attack
sends w
waits a
ask cac
and se
forged

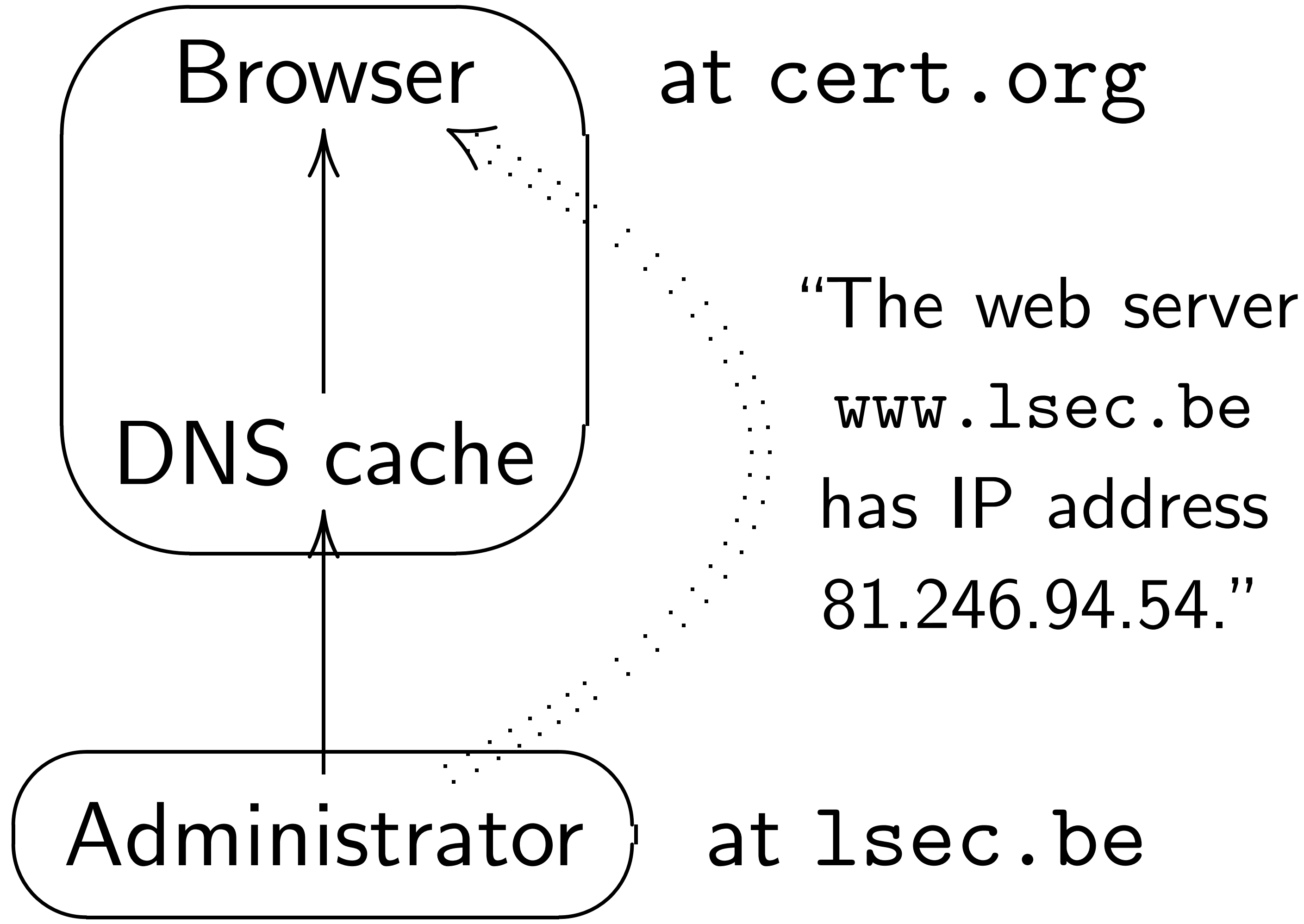
ttackers to
pperly:

edly.
ries will
t time.

nt
wn lookup.

s
dvance.

s instantly:
k.



Browser pulls data from DNS cache at `cert.org`.

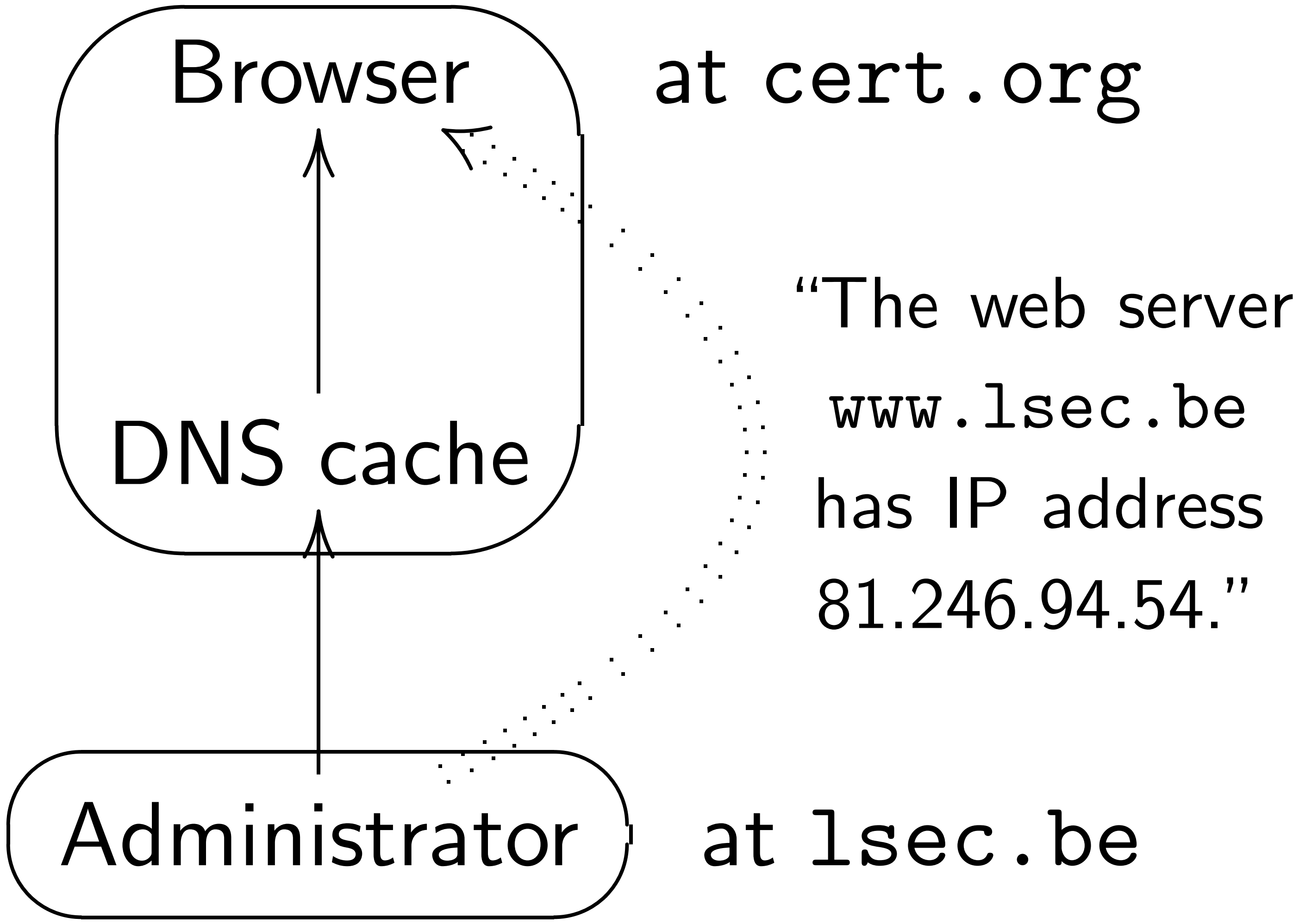
Cache pulls data from administrator *if* it doesn't already have the data.

A typical blind at

Attacker sets up
`supersecurityt`
including an inlin
from `www.lsec.`

Victim asks brow
`supersecurityt`

Attacker sees HT
sends web page t
waits a moment
ask cache about
and sends the DN
forged data for w



Browser pulls data from DNS cache at `cert.org`.

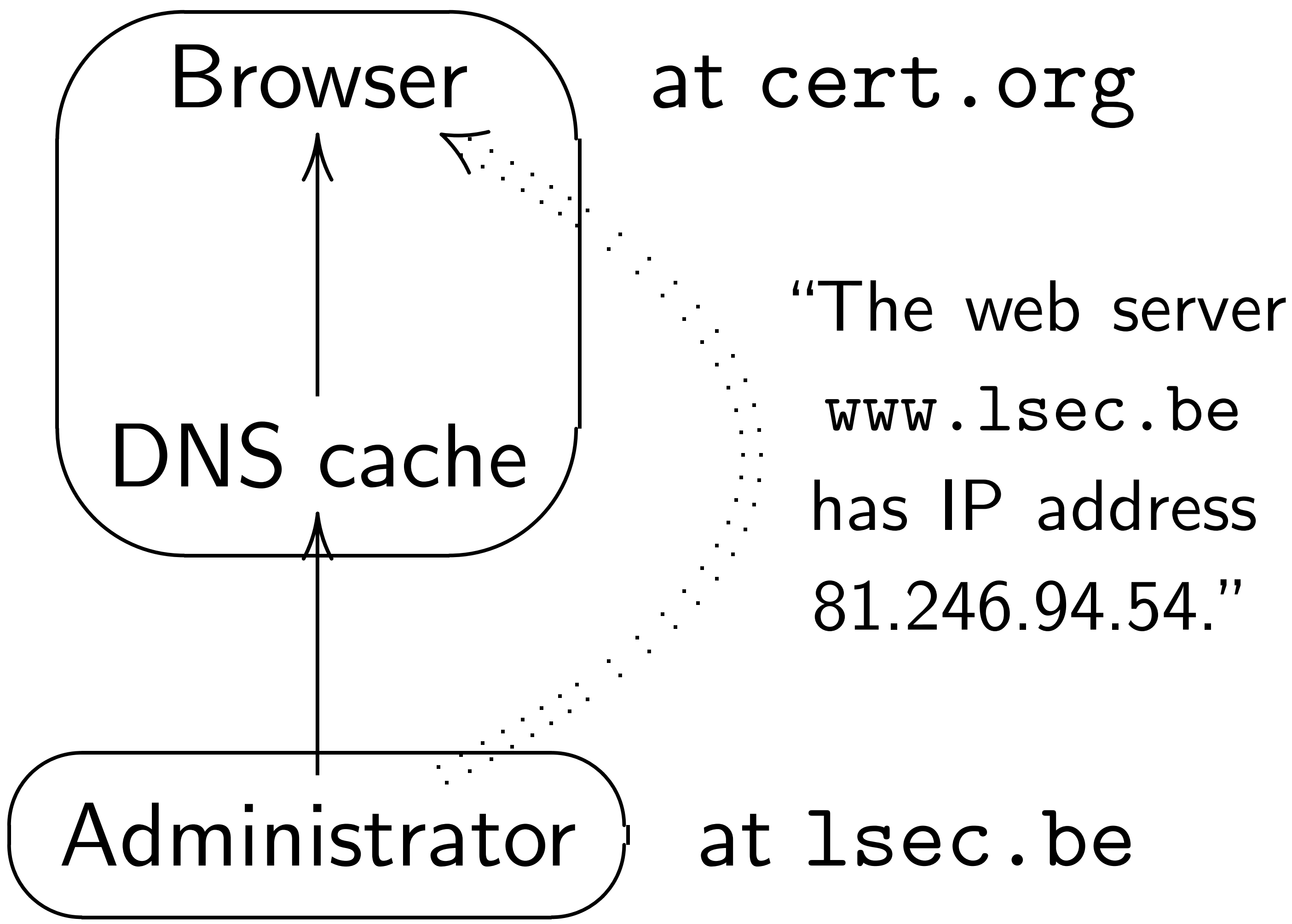
Cache pulls data from administrator *if* it doesn't already have the data.

A typical blind attack:

Attacker sets up a web pag
`supersecuritytools.to`,
including an inline image
from `www.lsec.be`.

Victim asks browser to view
`supersecuritytools.to`.

Attacker sees HTTP reque
sends web page to browser
waits a moment (for brows
ask cache about `www.lsec`
and sends the DNS cache
forged data for `www.lsec.`



Browser pulls data from DNS cache at `cert.org`.

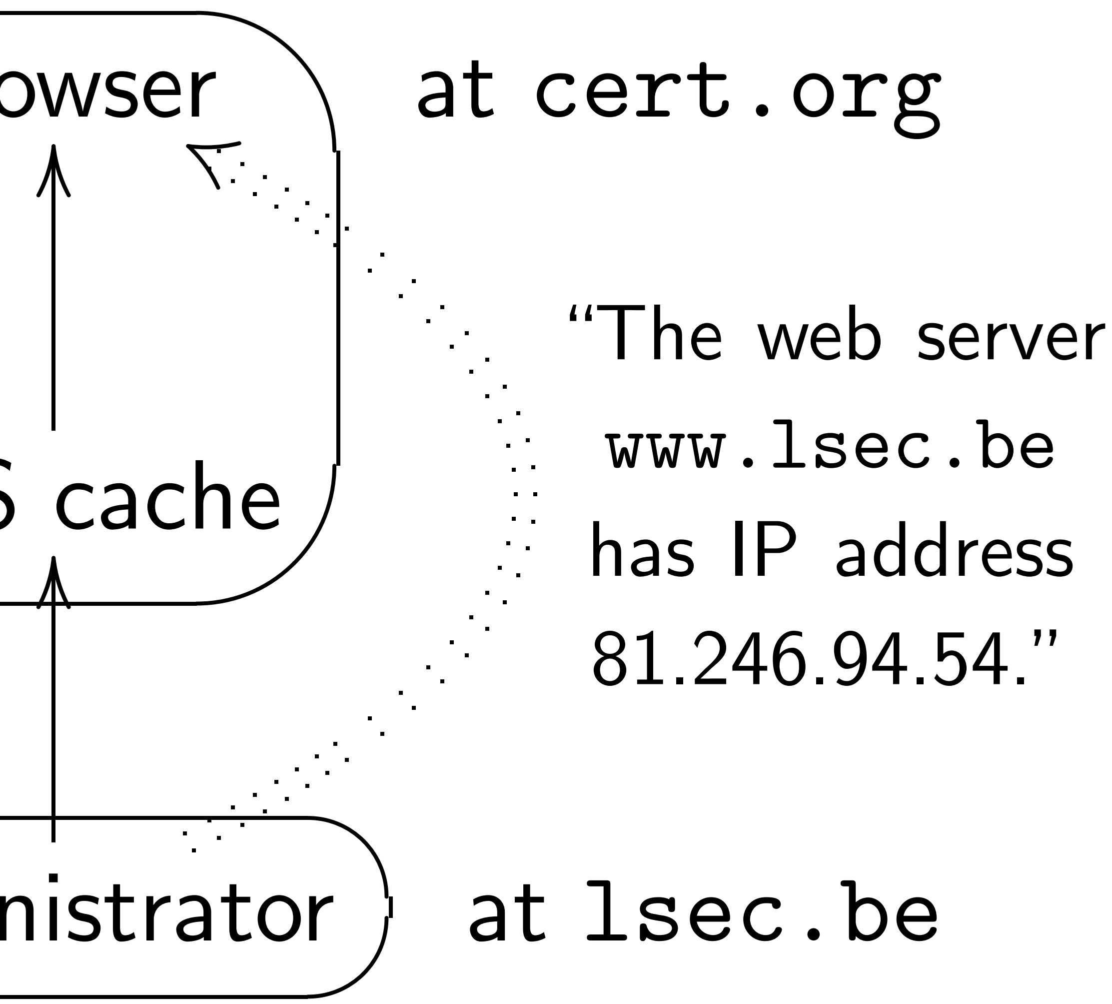
Cache pulls data from administrator *if* it doesn't already have the data.

A typical blind attack:

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees HTTP request, sends web page to browser, waits a moment (for browser to ask cache about `www.lsec.be`), and sends the DNS cache forged data for `www.lsec.be`.



er pulls data from
ache at cert.org.

pulls data from
strator *if* it
t already have the data.

A typical blind attack:

Attacker sets up a web page
supersecuritytools.to,
including an inline image
from www.lsec.be.

Victim asks browser to view
supersecuritytools.to.

Attacker sees HTTP request,
sends web page to browser,
waits a moment (for browser to
ask cache about www.lsec.be),
and sends the DNS cache
forged data for www.lsec.be.

“Doesn
win a r
legitim
from th
lsec.b

at `cert.org`

"The web server `www.lsec.be` has IP address 81.246.94.54."

at `lsec.be`

ta from
`ert.org`.

from
it
ave the data.

A typical blind attack:

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees HTTP request, sends web page to browser, waits a moment (for browser to ask cache about `www.lsec.be`), and sends the DNS cache forged data for `www.lsec.be`.

"Doesn't the att
win a race agains
legitimate DNS p
from the adminis
`lsec.be`?"

org

eb server
sec.be
address
.94.54."

.be

ata.

A typical blind attack:

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees HTTP request, sends web page to browser, waits a moment (for browser to ask cache about `www.lsec.be`), and sends the DNS cache forged data for `www.lsec.be`.

"Doesn't the attacker have win a race against the legitimate DNS packets from the administrator at `lsec.be`?"

A typical blind attack:

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees HTTP request, sends web page to browser, waits a moment (for browser to ask cache about `www.lsec.be`), and sends the DNS cache forged data for `www.lsec.be`.

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at `lsec.be`?"

A typical blind attack:

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees HTTP request, sends web page to browser, waits a moment (for browser to ask cache about `www.lsec.be`), and sends the DNS cache forged data for `www.lsec.be`.

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at `lsec.be`?"

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.
2. Mute the legitimate server.
3. Poke-jab-jab-jab-jab-jab.

A typical blind attack:

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees HTTP request, sends web page to browser, waits a moment (for browser to ask cache about `www.lsec.be`), and sends the DNS cache forged data for `www.lsec.be`.

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at `lsec.be`?"

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.

2. Mute the legitimate server.

3. Poke-jab-jab-jab-jab-jab.

4. Easy, succeeds instantly: Sniff the network.

cal blind attack:

er sets up a web page
securitytools.to,
ng an inline image
ww.lsec.be.

asks browser to view
securitytools.to.

er sees HTTP request,
web page to browser,
moment (for browser to
che about www.lsec.be),
nds the DNS cache
data for www.lsec.be.

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at lsec.be?"

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.
2. Mute the legitimate server.
3. Poke-jab-jab-jab-jab-jab.
4. Easy, succeeds instantly: Sniff the network.

Typical

Attacke
with qu
all avai
or flood
with pa
all avai

Attacke
trigger
Attacke
a series
to the

ttack:

a web page
tools.to,
ne image
be.

vser to view
tools.to.

TTP request,
to browser,
(for browser to
www.lsec.be),
NS cache
ww.lsec.be.

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at lsec.be?"

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.
2. Mute the legitimate server.
3. Poke-jab-jab-jab-jab-jab.
4. Easy, succeeds instantly: Sniff the network.

Typical combined

Attacker floods l
with queries that
all available CPU
or floods lsec.b
with packets that
all available netw

Attacker pokes th
trigger an lsec.
Attacker immedia
a series of forged
to the DNS cach

ge

,

w

.

st,

,

ser to

:.be),

be.

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at `lsec.be`?"

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.
2. Mute the legitimate server.
3. Poke-jab-jab-jab-jab-jab.
4. Easy, succeeds instantly: Sniff the network.

Typical combined blind att

Attacker floods `lsec.be` s
with queries that consume
all available CPU time,
or floods `lsec.be` network
with packets that consume
all available network capac

Attacker pokes the client t
trigger an `lsec.be` lookup
Attacker immediately send
a series of forged packets
to the DNS cache.

"Doesn't the attacker have to win a race against the legitimate DNS packets from the administrator at `lsec.be`?"

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.

2. Mute the legitimate server.

3. Poke-jab-jab-jab-jab-jab.

4. Easy, succeeds instantly: Sniff the network.

Typical combined blind attack:

Attacker floods `lsec.be` servers with queries that consume all available CPU time, or floods `lsec.be` network with packets that consume all available network capacity.

Attacker pokes the client to trigger an `lsec.be` lookup. Attacker immediately sends a series of forged packets to the DNS cache.

n't the attacker have to
race against the
ate DNS packets
he administrator at
be?"

, but many ways for
ers to win race:

fen the legitimate server.

te the legitimate server.

e-jab-jab-jab-jab-jab.

y, succeeds instantly:
he network.

Typical combined blind attack:

Attacker floods `lsec.be` servers
with queries that consume
all available CPU time,
or floods `lsec.be` network
with packets that consume
all available network capacity.

Attacker pokes the client to
trigger an `lsec.be` lookup.
Attacker immediately sends
a series of forged packets
to the DNS cache.

"What

— Mar
to cont

1. He a

2. He a
the sar

3. He a
the sar

With a
numbe
increas

acker have to
st the
packets
strator at

y ways for
race:

gitimate server.

timate server.

ab-jab-jab.

s instantly:
k.

Typical combined blind attack:

Attacker floods `lsec.be` servers with queries that consume all available CPU time, or floods `lsec.be` network with packets that consume all available network capacity.

Attacker pokes the client to trigger an `lsec.be` lookup. Attacker immediately sends a series of forged packets to the DNS cache.

"What if attacke

— Many ways fo
to continue his a

1. He attacks an

2. He attacks an
the same cache.

3. He attacks th
the same cache,

With any of thes
number of cache
increases linearly

e to

erver.

ver.

.

:

Typical combined blind attack:

Attacker floods `lsec.be` servers with queries that consume all available CPU time, or floods `lsec.be` network with packets that consume all available network capacity.

Attacker pokes the client to trigger an `lsec.be` lookup. Attacker immediately sends a series of forged packets to the DNS cache.

“What if attacker loses rac

— Many ways for attacker to continue his attack:

1. He attacks another cach

2. He attacks another nam the same cache.

3. He attacks the same na the same cache, sideways.

With any of these approac number of cached forgerie increases linearly over time

Typical combined blind attack:

Attacker floods `lsec.be` servers with queries that consume all available CPU time, or floods `lsec.be` network with packets that consume all available network capacity.

Attacker pokes the client to trigger an `lsec.be` lookup. Attacker immediately sends a series of forged packets to the DNS cache.

"What if attacker loses race?"

— Many ways for attacker to continue his attack:

1. He attacks another cache.

2. He attacks another name on the same cache.

3. He attacks the same name on the same cache, sideways.

With any of these approaches, number of cached forgeries increases linearly over time.

l combined blind attack:

er floods lsec.be servers
ueries that consume
ilable CPU time,
ds lsec.be network
ackets that consume
ilable network capacity.

er pokes the client to
an lsec.be lookup.
er immediately sends
s of forged packets
DNS cache.

"What if attacker loses race?"

— Many ways for attacker to continue his attack:

1. He attacks another cache.

2. He attacks another name on the same cache.

3. He attacks the same name on the same cache, sideways.

With any of these approaches, number of cached forgeries increases linearly over time.

Sidewa
in 2008

Attacke
trigger
867530

Attacke
867530
with ex
www.ls

For var
DNS ca
to acce

d blind attack:

.sec.be servers
t consume
J time,
oe network
t consume
vork capacity.

he client to
be lookup.
ately sends
l packets
e.

"What if attacker loses race?"

— Many ways for attacker
to continue his attack:

1. He attacks another cache.

2. He attacks another name on
the same cache.

3. He attacks the same name on
the same cache, sideways.

With any of these approaches,
number of cached forgeries
increases linearly over time.

Sideways attacks
in 2008 by Dan K

Attacker pokes th
trigger a DNS lo
`8675309.lsec.b`

Attacker forges r
`8675309.lsec.b`
with extra inform
`www.lsec.be`.

For various perfo
DNS caches are w
to accept the ext

ack:

ervers

k

e

ity.

o

.

s

"What if attacker loses race?"

— Many ways for attacker to continue his attack:

1. He attacks another cache.

2. He attacks another name on the same cache.

3. He attacks the same name on the same cache, sideways.

With any of these approaches, number of cached forgeries increases linearly over time.

Sideways attacks were pop in 2008 by Dan Kaminsky.

Attacker pokes the client t trigger a DNS lookup for `8675309.lsec.be`.

Attacker forges response fo `8675309.lsec.be` with extra information abo `www.lsec.be`.

For various performance re DNS caches are willing to accept the extra informa

"What if attacker loses race?"

— Many ways for attacker
to continue his attack:

1. He attacks another cache.

2. He attacks another name on the same cache.

3. He attacks the same name on the same cache, sideways.

With any of these approaches, number of cached forgeries increases linearly over time.

Sideways attacks were popularized in 2008 by Dan Kaminsky.

Attacker pokes the client to trigger a DNS lookup for `8675309.lsec.be`.

Attacker forges response for `8675309.lsec.be` with extra information about `www.lsec.be`.

For various performance reasons, DNS caches are willing to accept the extra information.

if attacker loses race?"

ny ways for attacker
tinue his attack:

attacks another cache.

attacks another name on
me cache.

attacks the same name on
me cache, sideways.

ny of these approaches,
r of cached forgeries
es linearly over time.

Sideways attacks were popularized in 2008 by Dan Kaminsky.

Attacker pokes the client to trigger a DNS lookup for `8675309.lsec.be`.

Attacker forges response for `8675309.lsec.be` with extra information about `www.lsec.be`.

For various performance reasons, DNS caches are willing to accept the extra information.

Interlu

Confid
cannot

Integri
silently
User d

Availa
cannot
User se

r loses race?"

r attacker
ttack:

other cache.

other name on

e same name on
sideways.

e approaches,
d forgeries
over time.

Sideways attacks were popularized in 2008 by Dan Kaminsky.

Attacker pokes the client to trigger a DNS lookup for `8675309.lsec.be`.

Attacker forges response for `8675309.lsec.be` with extra information about `www.lsec.be`.

For various performance reasons, DNS caches are willing to accept the extra information.

Interlude: types

Confidentiality:
cannot see this i

Integrity: The a
silently modify t
User doesn't see

Availability: Th
cannot modify th
User sees the rig

ce?"

ne.

ne on

me on

hes,

s

.

Sideways attacks were popularized in 2008 by Dan Kaminsky.

Attacker pokes the client to trigger a DNS lookup for `8675309.lsec.be`.

Attacker forges response for `8675309.lsec.be` with extra information about `www.lsec.be`.

For various performance reasons, DNS caches are willing to accept the extra information.

Interlude: types of security

Confidentiality: The attac cannot see this informatior

Integrity: The attacker ca *silently* modify this inform User doesn't see wrong dat

Availability: The attacker cannot modify this informa User sees the right data.

Sideways attacks were popularized in 2008 by Dan Kaminsky.

Attacker pokes the client to trigger a DNS lookup for `8675309.lsec.be`.

Attacker forges response for
`8675309.lsec.be`
with extra information about
`www.lsec.be`.

For various performance reasons,
DNS caches are willing
to accept the extra information.

Interlude: types of security

Confidentiality: The attacker cannot see this information.

Integrity: The attacker cannot *silently* modify this information. User doesn't see wrong data.

Availability: The attacker cannot modify this information. User sees the right data.

ys attacks were popularized
3 by Dan Kaminsky.

er pokes the client to
a DNS lookup for
09.lsec.be.

er forges response for
09.lsec.be
xtra information about
sec.be.

rious performance reasons,
aches are willing
ept the extra information.

Interlude: types of security

Confidentiality: The attacker cannot see this information.

Integrity: The attacker cannot *silently* modify this information. User doesn't see wrong data.

Availability: The attacker cannot modify this information. User sees the right data.

Attacke
is comp
("Deni

Attacke
DNS p
is comp

Attacke
is comp
attacke
Also co
user do

. were popularized
Kaminsky.

he client to
okup for
pe.

esponse for
pe
nation about

rmance reasons,
willing
tra information.

Interlude: types of security

Confidentiality: The attacker cannot see this information.

Integrity: The attacker cannot *silently* modify this information. User doesn't see wrong data.

Availability: The attacker cannot modify this information. User sees the right data.

Attacker flooding
is compromising
("Denial of servi

Attacker successf
DNS packets of
is compromising

Attacker stealing
is compromising
attacker sees the
Also compromisi
user doesn't see

ularized

o

or

ut

asons,

ation.

Interlude: types of security

Confidentiality: The attacker cannot see this information.

Integrity: The attacker cannot *silently* modify this information. User doesn't see wrong data.

Availability: The attacker cannot modify this information. User sees the right data.

Attacker flooding a networ
is compromising availabilit
("Denial of service.")

Attacker successfully forgin
DNS packets of `lsec.be`
is compromising integrity.

Attacker stealing email
is compromising confidenti
attacker sees the email.
Also compromising availab
user doesn't see the email.

Interlude: types of security

Confidentiality: The attacker cannot see this information.

Integrity: The attacker cannot *silently* modify this information. User doesn't see wrong data.

Availability: The attacker cannot modify this information. User sees the right data.

Attacker flooding a network is compromising availability. ("Denial of service.")

Attacker successfully forging DNS packets of `lsec.be` is compromising integrity.

Attacker stealing email is compromising confidentiality: attacker sees the email.
Also compromising availability: user doesn't see the email.

de: types of security

lentiality: The attacker
see this information.

ity: The attacker cannot
modify this information.
oesn't see wrong data.

bility: The attacker
modify this information.
ees the right data.

Attacker flooding a network
is compromising availability.
("Denial of service.")

Attacker successfully forging
DNS packets of `lsec.be`
is compromising integrity.

Attacker stealing email
is compromising confidentiality:
attacker sees the email.
Also compromising availability:
user doesn't see the email.

Lack o
helps c
e.g., flo
can ass

Lack o
helps c
e.g., sn
makes

Lack o
helps c
e.g., fo
allows

etc.

of security

The attacker
nformation.

ttacker cannot
his information.
wrong data.

e attacker
his information.
ht data.

Attacker flooding a network
is compromising availability.
("Denial of service.")

Attacker successfully forging
DNS packets of `lsec.be`
is compromising integrity.

Attacker stealing email
is compromising confidentiality:
attacker sees the email.
Also compromising availability:
user doesn't see the email.

Lack of availabili
helps compromis
e.g., flooding a s
can assist in DN

Lack of confiden
helps compromis
e.g., sniffing DN
makes forgeries t

Lack of integrity
helps compromis
e.g., forging DN
allows redirectin

etc.

y

cker

n.

nnot

ation.

ta.

ntion.

Attacker flooding a network
is compromising availability.
("Denial of service.")

Attacker successfully forging
DNS packets of `lsec.be`
is compromising integrity.

Attacker stealing email
is compromising confidentiality:
attacker sees the email.
Also compromising availability:
user doesn't see the email.

Lack of availability often
helps compromise integrity
e.g., flooding a server
can assist in DNS forgeries

Lack of confidentiality ofte
helps compromise integrity
e.g., sniffing DNS queries
makes forgeries trivial.

Lack of integrity often
helps compromise confiden
e.g., forging DNS packets
allows redirecting mail.

etc.

Attacker flooding a network
is compromising availability.
("Denial of service.")

Attacker successfully forging
DNS packets of `lsec.be`
is compromising integrity.

Attacker stealing email
is compromising confidentiality:
attacker sees the email.
Also compromising availability:
user doesn't see the email.

Lack of availability often
helps compromise integrity:
e.g., flooding a server
can assist in DNS forgeries.

Lack of confidentiality often
helps compromise integrity:
e.g., sniffing DNS queries
makes forgeries trivial.

Lack of integrity often
helps compromise confidentiality:
e.g., forging DNS packets
allows redirecting mail.

etc.

er flooding a network
promising availability.
al of service.")

er successfully forging
ackets of `lsec.be`
promising integrity.

er stealing email
promising confidentiality:
er sees the email.
ompromising availability:
oesn't see the email.

Lack of availability often
helps compromise integrity:
e.g., flooding a server
can assist in DNS forgeries.

Lack of confidentiality often
helps compromise integrity:
e.g., sniffing DNS queries
makes forgeries trivial.

Lack of integrity often
helps compromise confidentiality:
e.g., forging DNS packets
allows redirecting mail.

etc.

PGP-e
can pro
Attack
still wo

Also in
Attack

But it
The em

Retroa
doesn't

g a network
availability.
ce.")

fully forging
lsec.be
integrity.

email
confidentiality:
email.
ng availability:
the email.

Lack of availability often
helps compromise integrity:
e.g., flooding a server
can assist in DNS forgeries.

Lack of confidentiality often
helps compromise integrity:
e.g., sniffing DNS queries
makes forgeries trivial.

Lack of integrity often
helps compromise confidentiality:
e.g., forging DNS packets
allows redirecting mail.

etc.

PGP-encrypting
can provide confi
Attacker who ste
still won't unders

Also integrity.
Attacker can't m

But it won't prov
The email silentl

Retroactively che
doesn't restore a

k
y.

ng

ality:

ility:

Lack of availability often
helps compromise integrity:
e.g., flooding a server
can assist in DNS forgeries.

Lack of confidentiality often
helps compromise integrity:
e.g., sniffing DNS queries
makes forgeries trivial.

Lack of integrity often
helps compromise confidentiality:
e.g., forging DNS packets
allows redirecting mail.

etc.

PGP-encrypting your emai
can provide confidentiality.
Attacker who steals email
still won't understand it.

Also integrity.
Attacker can't modify ema

But it won't provide availa
The email silently disappea

Retroactively checking inte
doesn't restore availability.

Lack of availability often
helps compromise integrity:
e.g., flooding a server
can assist in DNS forgeries.

Lack of confidentiality often
helps compromise integrity:
e.g., sniffing DNS queries
makes forgeries trivial.

Lack of integrity often
helps compromise confidentiality:
e.g., forging DNS packets
allows redirecting mail.

etc.

PGP-encrypting your email
can provide confidentiality.
Attacker who steals email
still won't understand it.

Also integrity.
Attacker can't modify email.

But it won't provide availability.
The email silently disappeared!

Retroactively checking integrity
doesn't restore availability.

f availability often
ompromise integrity:
ooding a server
sist in DNS forgeries.

f confidentiality often
ompromise integrity:
niffing DNS queries
forgeries trivial.

f integrity often
ompromise confidentiality:
rging DNS packets
redirecting mail.

PGP-encrypting your email
can provide confidentiality.
Attacker who steals email
still won't understand it.

Also integrity.
Attacker can't modify email.

But it won't provide availability.
The email silently disappeared!

Retroactively checking integrity
doesn't restore availability.

What a

Cache's
contai

RFC 1
is copi
used b
replies

Traditi
1, 2, 3

Cache
that h

"How
guess t

ty often
e integrity:
erver
S forgeries.

tiality often
e integrity:
S queries
rivial.

often
e confidentiality:
S packets
g mail.

PGP-encrypting your email
can provide confidentiality.
Attacker who steals email
still won't understand it.

Also integrity.
Attacker can't modify email.

But it won't provide availability.
The email silently disappeared!

Retroactively checking integrity
doesn't restore availability.

What about cool

Cache's DNS que
contains a 16-bit

RFC 1035 (1987)
is copied [to the]
used by the reque
replies to outstan

Traditional ID se
1, 2, 3, 4, 5, etc.

Cache discards a
that has the wro

"How does the a
guess the right ID

:

s.

en

:

tiality:

PGP-encrypting your email
can provide confidentiality.
Attacker who steals email
still won't understand it.

Also integrity.
Attacker can't modify email.

But it won't provide availability.
The email silently disappeared!

Retroactively checking integrity
doesn't restore availability.

What about cookies?

Cache's DNS query packet
contains a 16-bit ID.

RFC 1035 (1987): "This i
is copied [to the] reply and
used by the requester to m
replies to outstanding quer

Traditional ID sequence:
1, 2, 3, 4, 5, etc.

Cache discards any reply
that has the wrong ID.

"How does the attacker
guess the right ID?"

PGP-encrypting your email can provide confidentiality. Attacker who steals email still won't understand it.

Also integrity. Attacker can't modify email.

But it won't provide availability. The email silently disappeared!

Retroactively checking integrity doesn't restore availability.

What about cookies?

Cache's DNS query packet contains a 16-bit ID.

RFC 1035 (1987): "This identifier is copied [to the] reply and can be used by the requester to match up replies to outstanding queries."

Traditional ID sequence: 1, 2, 3, 4, 5, etc.

Cache discards any reply that has the wrong ID.

"How does the attacker guess the right ID?"

ncrypting your email
ovide confidentiality.
er who steals email
on't understand it.

tegrity.
er can't modify email.

won't provide availability.
nail silently disappeared!

ctively checking integrity
t restore availability.

What about cookies?

Cache's DNS query packet
contains a 16-bit ID.

RFC 1035 (1987): "This identifier is copied [to the] reply and can be used by the requester to match up replies to outstanding queries."

Traditional ID sequence:
1, 2, 3, 4, 5, etc.

Cache discards any reply
that has the wrong ID.

"How does the attacker
guess the right ID?"

Attack
`supers`
includi
from `w`

Attack
`supers`
from h

Victim
`supers`

Attack
`supers`
DNS q
predict

your email
identiality.
eals email
stand it.

odify email.

vide availability.
y disappeared!

ecking integrity
vailability.

What about cookies?

Cache's DNS query packet
contains a 16-bit ID.

RFC 1035 (1987): "This identifier is copied [to the] reply and can be used by the requester to match up replies to outstanding queries."

Traditional ID sequence:
1, 2, 3, 4, 5, etc.

Cache discards any reply
that has the wrong ID.

"How does the attacker
guess the right ID?"

Attacker sets up
`supersecurityt`
including an inlin
from `www.lsec.`

Attacker provides
`supersecurityt`
from his own DN

Victim asks brow
`supersecurityt`

Attacker sees cac
`supersecurityt`
DNS query. Atta
predicts ID for `ls`

l

il.

bility.

ared!

egrity

What about cookies?

Cache's DNS query packet contains a 16-bit ID.

RFC 1035 (1987): "This identifier is copied [to the] reply and can be used by the requester to match up replies to outstanding queries."

Traditional ID sequence: 1, 2, 3, 4, 5, etc.

Cache discards any reply that has the wrong ID.

"How does the attacker guess the right ID?"

Attacker sets up a web pag
`supersecuritytools.to`,
including an inline image
from `www.lsec.be`.

Attacker provides DNS dat
`supersecuritytools.to`
from his own DNS servers.

Victim asks browser to vie
`supersecuritytools.to`.

Attacker sees cache's ID fo
`supersecuritytools.to`
DNS query. Attacker then
predicts ID for `lsec.be` qu

What about cookies?

Cache's DNS query packet contains a 16-bit ID.

RFC 1035 (1987): "This identifier is copied [to the] reply and can be used by the requester to match up replies to outstanding queries."

Traditional ID sequence: 1, 2, 3, 4, 5, etc.

Cache discards any reply that has the wrong ID.

"How does the attacker guess the right ID?"

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Attacker provides DNS data for `supersecuritytools.to` from his own DNS servers.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees cache's ID for `supersecuritytools.to` DNS query. Attacker then predicts ID for `lsec.be` query.

about cookies?

s DNS query packet
ns a 16-bit ID.

035 (1987): "This identifier
ed [to the] reply and can be
y the requester to match up
to outstanding queries."

onal ID sequence:
, 4, 5, etc.

discards any reply
as the wrong ID.

does the attacker
the right ID?"

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Attacker provides DNS data for `supersecuritytools.to` from his own DNS servers.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees cache's ID for `supersecuritytools.to` DNS query. Attacker then predicts ID for `lsec.be` query.

Ca
"ID 47
(and

Bro

Ca
"ID 476
Ca
"ID 476
Ca
"ID 476
Ca
"ID 476
Ca
"ID 476

kies?

ery packet
ID.

): “This identifier
reply and can be
ester to match up
nding queries.”

quence:

ny reply
ng ID.

ttacker
D?”

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Attacker provides DNS data for `supersecuritytools.to` from his own DNS servers.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees cache’s ID for `supersecuritytools.to` DNS query. Attacker then predicts ID for `lsec.be` query.

“`ID 47603:`
Cache ←
“`ID 47603: SST.t`
`(and please don`
“`http://`
Browser ←
“`... lsec`
Cache ←
“ID 47604: `lsec.b`
Cache ←
“ID 47604: `lsec.b`
Cache ←
“ID 47604: `lsec.b`
Cache ←
“ID 47604: `lsec.b`
Cache ←
“ID 47604: `lsec.b`

dentifier
can be
atch up
ies.”

Attacker sets up a web page supersecuritytools.to, including an inline image from www.lsec.be.

Attacker provides DNS data for supersecuritytools.to from his own DNS servers.

Victim asks browser to view supersecuritytools.to.

Attacker sees cache’s ID for supersecuritytools.to DNS query. Attacker then predicts ID for lsec.be query.

“ID 47603: SST.to?”

Cache $\longrightarrow$ Attac

Cache $\longleftarrow$ Attac

“ID 47603: SST.to 157.22.2

(and please don’t cache th

“http://SST.to”

Browser $\longrightarrow$ Attac

Browser $\longleftarrow$ Attac

“... lsec.be ...”

Cache $\longleftarrow$ Attac

“ID 47604: lsec.be 157.22.2

Cache $\longleftarrow$ Attac

“ID 47604: lsec.be 157.22.2

Cache $\longleftarrow$ Attac

“ID 47604: lsec.be 157.22.2

Cache $\longleftarrow$ Attac

“ID 47604: lsec.be 157.22.2

Cache $\longleftarrow$ Attac

“ID 47604: lsec.be 157.22.2

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Attacker provides DNS data for `supersecuritytools.to` from his own DNS servers.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees cache's ID for `supersecuritytools.to` DNS query. Attacker then predicts ID for `lsec.be` query.

“`ID 47603: SST.to?`”

Cache → Attacker

Cache ← Attacker

“`ID 47603: SST.to 157.22.245.20`
`(and please don't cache this)`”

“`http://SST.to`”

Browser → Attacker

Browser ← Attacker

“`... lsec.be ...`”

Cache ← Attacker

“ID 47604: `lsec.be 157.22.245.20`”

Cache ← Attacker

“ID 47604: `lsec.be 157.22.245.20`”

Cache ← Attacker

“ID 47604: `lsec.be 157.22.245.20`”

Cache ← Attacker

“ID 47604: `lsec.be 157.22.245.20`”

Cache ← Attacker

“ID 47604: `lsec.be 157.22.245.20`”

er sets up a web page `securitytools.to`, ng an inline image `ww.lsec.be`.

er provides DNS data for `securitytools.to` is own DNS servers.

asks browser to view `securitytools.to`.

er sees cache's ID for `securitytools.to` uery. Attacker then s ID for `lsec.be` query.

"`ID 47603: SST.to?`"

Cache → Attacker

Cache ← Attacker

"`ID 47603: SST.to 157.22.245.20`

`(and please don't cache this)`"

"`http://SST.to`"

Browser → Attacker

Browser ← Attacker

"`... lsec.be ...`"

Cache ← Attacker

"ID 47604: `lsec.be 157.22.245.20`"

Cache ← Attacker

"ID 47604: `lsec.be 157.22.245.20`"

Cache ← Attacker

"ID 47604: `lsec.be 157.22.245.20`"

Cache ← Attacker

"ID 47604: `lsec.be 157.22.245.20`"

Cache ← Attacker

"ID 47604: `lsec.be 157.22.245.20`"

More r "Hey, l the att forge a

Can us to expa into a l "rando AES-C Salsa20

Output attacke next ID entire s

a web page
tools.to,
he image
be.

s DNS data for
tools.to
IS servers.

wser to view
tools.to.

che's ID for
tools.to
cker then
sec.be query.

“`ID 47603: SST.to?`”

Cache $\xrightarrow{\hspace{3em}}$ Attacker
Cache $\xleftarrow{\hspace{3em}}$ Attacker

“`ID 47603: SST.to 157.22.245.20`
`(and please don't cache this)`”

“`http://SST.to`”

Browser $\xrightarrow{\hspace{3em}}$ Attacker
Browser $\xleftarrow{\hspace{3em}}$ Attacker

“`... lsec.be ...`”

Cache $\xleftarrow{\hspace{3em}}$ Attacker
“ID 47604: `lsec.be 157.22.245.20`”

Cache $\xleftarrow{\hspace{3em}}$ Attacker
“ID 47604: `lsec.be 157.22.245.20`”

Cache $\xleftarrow{\hspace{3em}}$ Attacker
“ID 47604: `lsec.be 157.22.245.20`”

Cache $\xleftarrow{\hspace{3em}}$ Attacker
“ID 47604: `lsec.be 157.22.245.20`”

Cache $\xleftarrow{\hspace{3em}}$ Attacker
“ID 47604: `lsec.be 157.22.245.20`”

More recent idea
“Hey, let's use ra
the attacker won
forge a packet w

Can use any goo
to expand a shor
into a long seque
“random” numbe
AES-CTR: $\approx$ 10
Salsa20/12: $\approx$ 3

Output is very ha
attacker has no i
next ID will be, e
entire sequence o

ge
,

ta for

w
r

or

uery.

"ID 47603: SST.to?"

Cache $\longrightarrow$ Attacker

Cache $\longleftarrow$ Attacker

"ID 47603: SST.to 157.22.245.20

(and please don't cache this)"

"http://SST.to"

Browser $\longrightarrow$ Attacker

Browser $\longleftarrow$ Attacker

"... lsec.be ..."

Cache $\longleftarrow$ Attacker

"ID 47604: lsec.be 157.22.245.20"

Cache $\longleftarrow$ Attacker

"ID 47604: lsec.be 157.22.245.20"

Cache $\longleftarrow$ Attacker

"ID 47604: lsec.be 157.22.245.20"

Cache $\longleftarrow$ Attacker

"ID 47604: lsec.be 157.22.245.20"

Cache $\longleftarrow$ Attacker

"ID 47604: lsec.be 157.22.245.20"

More recent idea:
"Hey, let's use random IDs
the attacker won't be able
forge a packet with the rig

Can use any good stream c
to expand a short secret ke
into a long sequence of
"random" numbers.
AES-CTR: $\approx$ 10 cycles/byt
Salsa20/12: $\approx$ 3 cycles/by

Output is very hard to pred
attacker has no idea what
next ID will be, even after
entire sequence of previous

“`ID 47603: SST.to?`”

Cache $\longrightarrow$ Attacker
Cache $\longleftarrow$ Attacker

“`ID 47603: SST.to 157.22.245.20`
`(and please don’t cache this)`”

“`http://SST.to`”

Browser $\longrightarrow$ Attacker
Browser $\longleftarrow$ Attacker

“`... lsec.be ...`”

Cache $\longleftarrow$ Attacker

“ID 47604: `lsec.be 157.22.245.20`”

Cache $\longleftarrow$ Attacker

“ID 47604: `lsec.be 157.22.245.20`”

Cache $\longleftarrow$ Attacker

“ID 47604: `lsec.be 157.22.245.20`”

Cache $\longleftarrow$ Attacker

“ID 47604: `lsec.be 157.22.245.20`”

Cache $\longleftarrow$ Attacker

“ID 47604: `lsec.be 157.22.245.20`”

More recent idea:
“Hey, let’s use random IDs! Then the attacker won’t be able to forge a packet with the right ID!”

Can use any good stream cipher to expand a short secret key into a long sequence of “random” numbers.
AES-CTR: $\approx$ 10 cycles/byte.
Salsa20/12: $\approx$ 3 cycles/byte.

Output is very hard to predict: attacker has no idea what the next ID will be, even after seeing entire sequence of previous IDs.

"ID 47603: SST.to?"

ache $\overrightarrow{\longleftarrow}$ Attacker

`603: SST.to 157.22.245.20`

`please don't cache this)"`

"`http://SST.to`"

owser $\overrightarrow{\longleftarrow}$ Attacker

"`... lsec.be ...`"

ache $\longleftarrow$ Attacker

604: `lsec.be 157.22.245.20`"

ache $\longleftarrow$ Attacker

604: `lsec.be 157.22.245.20`"

ache $\longleftarrow$ Attacker

604: `lsec.be 157.22.245.20`"

ache $\longleftarrow$ Attacker

604: `lsec.be 157.22.245.20`"

ache $\longleftarrow$ Attacker

604: `lsec.be 157.22.245.20`"

More recent idea:
"Hey, let's use random IDs! Then the attacker won't be able to forge a packet with the right ID!"

Can use any good stream cipher to expand a short secret key into a long sequence of "random" numbers.
AES-CTR: $\approx$ 10 cycles/byte.
Salsa20/12: $\approx$ 3 cycles/byte.

Output is very hard to predict: attacker has no idea what the next ID will be, even after seeing entire sequence of previous IDs.

Client

16-bit

16-bit

Implem

in `djbd`

and in

Same f

in "em

Micros

most C

New Y

"WITH

A PUS

SST.to?"

Attacker

to 157.22.245.20

't cache this)"

/SST.to"

Attacker

c.be ..."

Attacker

be 157.22.245.20"

Attacker

be 157.22.245.20"

Attacker

be 157.22.245.20"

Attacker

be 157.22.245.20"

Attacker

be 157.22.245.20"

More recent idea:
"Hey, let's use random IDs! Then the attacker won't be able to forge a packet with the right ID!"

Can use any good stream cipher to expand a short secret key into a long sequence of "random" numbers.
AES-CTR: ≈ 10 cycles/byte.
Salsa20/12: ≈ 3 cycles/byte.

Output is very hard to predict: attacker has no idea what the next ID will be, even after seeing entire sequence of previous IDs.

Client can rando
16-bit ID *and*
16-bit UDP sour

Implemented and
in `djbdns` since
and in PowerDN

Same feature ad
in "emergency"
Microsoft DNS,
most Cisco prod

New York Times
"WITH SECURI
A PUSH TO PA

ker
245.20
his)"

ker

ker
45.20"
ker
45.20"
ker
45.20"
ker
45.20"
ker
45.20"

More recent idea:
"Hey, let's use random IDs! Then the attacker won't be able to forge a packet with the right ID!"

Can use any good stream cipher to expand a short secret key into a long sequence of "random" numbers.
AES-CTR: ≈ 10 cycles/byte.
Salsa20/12: ≈ 3 cycles/byte.

Output is very hard to predict: attacker has no idea what the next ID will be, even after seeing entire sequence of previous IDs.

Client can randomize
16-bit ID *and*
16-bit UDP source port.

Implemented and advertise
in `djbdns` since 1999,
and in PowerDNS since 20

Same feature added 2008.0
in "emergency" upgrade to
Microsoft DNS, Nominum
most Cisco products, etc.

New York Times headline:
"WITH SECURITY AT RI
A PUSH TO PATCH THE

More recent idea:
"Hey, let's use random IDs! Then the attacker won't be able to forge a packet with the right ID!"

Can use any good stream cipher to expand a short secret key into a long sequence of "random" numbers.
AES-CTR: $\approx$ 10 cycles/byte.
Salsa20/12: $\approx$ 3 cycles/byte.

Output is very hard to predict: attacker has no idea what the next ID will be, even after seeing entire sequence of previous IDs.

Client can randomize 16-bit ID *and* 16-bit UDP source port.

Implemented and advertised in `djbdns` since 1999, and in PowerDNS since 2006.

Same feature added 2008.07 in "emergency" upgrade to BIND, Microsoft DNS, Nominum CNS, most Cisco products, etc.

New York Times headline:
"WITH SECURITY AT RISK,
A PUSH TO PATCH THE WEB"

ecent idea:
et's use random IDs! Then
acker won't be able to
packet with the right ID!"

e any good stream cipher
and a short secret key
long sequence of
m" numbers.
TR: $\approx$ 10 cycles/byte.
0/12: $\approx$ 3 cycles/byte.

t is very hard to predict:
er has no idea what the
D will be, even after seeing
sequence of previous IDs.

Client can randomize
16-bit ID *and*
16-bit UDP source port.

Implemented and advertised
in `djbdns` since 1999,
and in PowerDNS since 2006.

Same feature added 2008.07
in "emergency" upgrade to BIND,
Microsoft DNS, Nominum CNS,
most Cisco products, etc.

New York Times headline:
"WITH SECURITY AT RISK,
A PUSH TO PATCH THE WEB"

Bad ne
often w
See, e.g
leading
"emerg

In e
(sinc
oppo
of th
know
stop
"mu
gene
bilat
gene

The
take
to e
13-1

:
andom IDs! Then
't be able to
ith the right ID!"

d stream cipher
t secret key
ence of
ers.
cycles/byte.
cycles/byte.

ard to predict:
dea what the
even after seeing
of previous IDs.

Client can randomize
16-bit ID *and*
16-bit UDP source port.

Implemented and advertised
in `djbdns` since 1999,
and in PowerDNS since 2006.

Same feature added 2008.07
in "emergency" upgrade to BIND,
Microsoft DNS, Nominum CNS,
most Cisco products, etc.

New York Times headline:
"WITH SECURITY AT RISK,
A PUSH TO PATCH THE WEB"

Bad news: Ignora
often whip up bre
See, e.g., Klein's
leading to 2007.0
"emergency" BIN

> In essence, this
> (since the outpu
> opposed to the
> of the well studi
> known by many
> stop/go (LFSR)
> "mutually clock
> generator" and
> bilateral) step-1/
> generator". ...
>
> The Perl script
> takes around 10-
> to extract the in
> 13-15 consecutiv

s! Then
to
ht ID!"

cipher
ey

te.
te.

dict:
the
seeing
IDs.

Client can randomize
16-bit ID *and*
16-bit UDP source port.

Implemented and advertised
in `djbdns` since 1999,
and in PowerDNS since 2006.

Same feature added 2008.07
in "emergency" upgrade to BIND,
Microsoft DNS, Nominum CNS,
most Cisco products, etc.

New York Times headline:
"WITH SECURITY AT RISK,
A PUSH TO PATCH THE WEB"

Bad news: Ignorant develo
often whip up breakable cip
See, e.g., Klein's analysis
leading to 2007.07.24
"emergency" BIND 9 upgr

> In essence, this is a weak ve
> (since the output is 16 bits,
> opposed to the traditional 1
> of the well studied cryptosys
> known by many names: "bil
> stop/go (LFSR) generator",
> "mutually clock controlled (
> generator" and "mutual (or
> bilateral) step-1/step-2 (LFS
> generator". ...
>
> The Perl script in Appendix
> takes around 10-15 milliseco
> to extract the internal state
> 13-15 consecutive transactio

Client can randomize
16-bit ID *and*
16-bit UDP source port.

Implemented and advertised in `djbdns` since 1999, and in PowerDNS since 2006.

Same feature added 2008.07 in "emergency" upgrade to BIND, Microsoft DNS, Nominum CNS, most Cisco products, etc.

New York Times headline: "WITH SECURITY AT RISK, A PUSH TO PATCH THE WEB"

Bad news: Ignorant developers often whip up breakable ciphers. See, e.g., Klein's analysis leading to 2007.07.24 "emergency" BIND 9 upgrade:

> In essence, this is a weak version (since the output is 16 bits, as opposed to the traditional 1 bit) of the well studied cryptosystem known by many names: "bilateral stop/go (LFSR) generator", "mutually clock controlled (LFSR) generator" and "mutual (or bilateral) step-1/step-2 (LFSR) generator". ...
>
> The Perl script in Appendix C takes around 10-15 milliseconds ... to extract the internal state from 13-15 consecutive transaction IDs.

can randomize
ID *and*
UDP source port.

nented and advertised
dns since 1999,
PowerDNS since 2006.

feature added 2008.07
ergency" upgrade to BIND,
oft DNS, Nominum CNS,
Cisco products, etc.

ork Times headline:
H SECURITY AT RISK,
H TO PATCH THE WEB"

Bad news: Ignorant developers often whip up breakable ciphers. See, e.g., Klein's analysis leading to 2007.07.24 "emergency" BIND 9 upgrade:

> In essence, this is a weak version (since the output is 16 bits, as opposed to the traditional 1 bit) of the well studied cryptosystem known by many names: "bilateral stop/go (LFSR) generator", "mutually clock controlled (LFSR) generator" and "mutual (or bilateral) step-1/step-2 (LFSR) generator". ...
>
> The Perl script in Appendix C takes around 10-15 milliseconds ... to extract the internal state from 13-15 consecutive transaction IDs.

Also K
analysi
NetBSI

> Ope
> thei
> thei
> tran
> deci
> prov
> Con
> Tha
> BIN
> does
> proa
> strik
> seri
> PRN
> to p

Also K
analyse

mize

ce port.

l advertised
1999,
S since 2006.

ded 2008.07
upgrade to BIND,
Nominum CNS,
ucts, etc.

headline:
TY AT RISK,
TCH THE WEB"

Bad news: Ignorant developers often whip up breakable ciphers. See, e.g., Klein's analysis leading to 2007.07.24 "emergency" BIND 9 upgrade:

> In essence, this is a weak version (since the output is 16 bits, as opposed to the traditional 1 bit) of the well studied cryptosystem known by many names: "bilateral stop/go (LFSR) generator", "mutually clock controlled (LFSR) generator" and "mutual (or bilateral) step-1/step-2 (LFSR) generator". ...
>
> The Perl script in Appendix C takes around 10-15 milliseconds ... to extract the internal state from 13-15 consecutive transaction IDs.

Also Klein's 2008
analysis of IDs in
NetBSD, FreeBS

> OpenBSD porte
> their code tree,
> their own PRNG
> transaction ID fi
> decided ... to us
> proven algorithm
> Congruential Ge
> Thanks to this w
> BIND 9 shipped
> does not have th
> proactive securit
> strikes again." .
> serious weakness
> PRNG, which al
> to predict the ne

Also Klein's 2007
analyses of Micro

ed

06.

07

BIND,

CNS,

SK,

WEB"

Bad news: Ignorant developers often whip up breakable ciphers. See, e.g., Klein's analysis leading to 2007.07.24 "emergency" BIND 9 upgrade:

> In essence, this is a weak version (since the output is 16 bits, as opposed to the traditional 1 bit) of the well studied cryptosystem known by many names: "bilateral stop/go (LFSR) generator", "mutually clock controlled (LFSR) generator" and "mutual (or bilateral) step-1/step-2 (LFSR) generator". ...
>
> The Perl script in Appendix C takes around 10-15 milliseconds ... to extract the internal state from 13-15 consecutive transaction IDs.

Also Klein's 2008.02.06 analysis of IDs in OpenBSD NetBSD, FreeBSD, MacOS

> OpenBSD ported BIND 9 in their code tree, but rolled their own PRNG for the DN transaction ID field). ... "W decided ... to use a more proven algorithm (LCG, Line Congruential Generator) inst Thanks to this wise decision BIND 9 shipped with OpenB does not have this weakness proactive security of OpenBS strikes again." ... I discover serious weakness in OpenBS PRNG, which allows an atta to predict the next transacti

Also Klein's 2007 and 2008 analyses of Microsoft IDs.

Bad news: Ignorant developers often whip up breakable ciphers. See, e.g., Klein's analysis leading to 2007.07.24 "emergency" BIND 9 upgrade:

> In essence, this is a weak version (since the output is 16 bits, as opposed to the traditional 1 bit) of the well studied cryptosystem known by many names: "bilateral stop/go (LFSR) generator", "mutually clock controlled (LFSR) generator" and "mutual (or bilateral) step-1/step-2 (LFSR) generator". ...
>
> The Perl script in Appendix C takes around 10-15 milliseconds ... to extract the internal state from 13-15 consecutive transaction IDs.

Also Klein's 2008.02.06 analysis of IDs in OpenBSD, NetBSD, FreeBSD, MacOS X:

> OpenBSD ported BIND 9 into their code tree, but rolled their own PRNG for the DNS transaction ID field). ... "We decided ... to use a more proven algorithm (LCG, Linear Congruential Generator) instead. Thanks to this wise decision, the BIND 9 shipped with OpenBSD does not have this weakness. The proactive security of OpenBSD strikes again." ... I discovered a serious weakness in OpenBSD's PRNG, which allows an attacker to predict the next transaction ID.

Also Klein's 2007 and 2008 analyses of Microsoft IDs.

ews: Ignorant developers
whip up breakable ciphers.

g., Klein's analysis
g to 2007.07.24
gency" BIND 9 upgrade:

ssence, this is a weak version
ce the output is 16 bits, as
osed to the traditional 1 bit)
he well studied cryptosystem
wn by many names: "bilateral
/go (LFSR) generator",
utually clock controlled (LFSR)
erator" and "mutual (or
teral) step-1/step-2 (LFSR)
erator". ...

Perl script in Appendix C
s around 10-15 milliseconds ...
xtract the internal state from
15 consecutive transaction IDs.

Also Klein's 2008.02.06
analysis of IDs in OpenBSD,
NetBSD, FreeBSD, MacOS X:

> OpenBSD ported BIND 9 into their code tree, but rolled their own PRNG for the DNS transaction ID field). ... "We decided ... to use a more proven algorithm (LCG, Linear Congruential Generator) instead. Thanks to this wise decision, the BIND 9 shipped with OpenBSD does not have this weakness. The proactive security of OpenBSD strikes again." ... I discovered a serious weakness in OpenBSD's PRNG, which allows an attacker to predict the next transaction ID.

Also Klein's 2007 and 2008
analyses of Microsoft IDs.

Bad ne
Many v
to beat
even if

1. Atta
"An at
billion
to succ
millions
with a

2. Allo
to othe

ant developers
eakable ciphers.
analysis
07.24
ND 9 upgrade:

is a weak version
t is 16 bits, as
traditional 1 bit)
'ed cryptosystem
names: "bilateral
generator",
controlled (LFSR)
"mutual (or
/step-2 (LFSR)

in Appendix C
-15 milliseconds ...
nternal state from
ve transaction IDs.

Also Klein's 2008.02.06 analysis of IDs in OpenBSD, NetBSD, FreeBSD, MacOS X:

> OpenBSD ported BIND 9 into their code tree, but rolled their own PRNG for the DNS transaction ID field). ... "We decided ... to use a more proven algorithm (LCG, Linear Congruential Generator) instead. Thanks to this wise decision, the BIND 9 shipped with OpenBSD does not have this weakness. The proactive security of OpenBSD strikes again." ... I discovered a serious weakness in OpenBSD's PRNG, which allows an attacker to predict the next transaction ID.

Also Klein's 2007 and 2008 analyses of Microsoft IDs.

Bad news, contir
Many ways for at
to beat ID+port
even if it's crypto

1. Attack repeat
"An attacker wh
billion random gu
to succeed at lea
millions of guesse
with a colliding a

2. Allocate most
to other tasks, n

opers
phers.

ade:

ersion
as
bit)
stem
ateral

LFSR)

SR)

C
nds …
from
n IDs.

Also Klein's 2008.02.06 analysis of IDs in OpenBSD, NetBSD, FreeBSD, MacOS X:

> OpenBSD ported BIND 9 into their code tree, but rolled their own PRNG for the DNS transaction ID field). … "We decided … to use a more proven algorithm (LCG, Linear Congruential Generator) instead. Thanks to this wise decision, the BIND 9 shipped with OpenBSD does not have this weakness. The proactive security of OpenBSD strikes again." … I discovered a serious weakness in OpenBSD's PRNG, which allows an attacker to predict the next transaction ID.

Also Klein's 2007 and 2008 analyses of Microsoft IDs.

Bad news, continued:
Many ways for attackers
to beat ID+port randomiza
even if it's cryptographic.

1. Attack repeatedly.
"An attacker who makes a
billion random guesses is li
to succeed at least once; te
millions of guesses are ade
with a colliding attack;" et

2. Allocate most UDP por
to other tasks, non-reusabl

Also Klein's 2008.02.06 analysis of IDs in OpenBSD, NetBSD, FreeBSD, MacOS X:

> OpenBSD ported BIND 9 into their code tree, but rolled their own PRNG for the DNS transaction ID field). … "We decided … to use a more proven algorithm (LCG, Linear Congruential Generator) instead. Thanks to this wise decision, the BIND 9 shipped with OpenBSD does not have this weakness. The proactive security of OpenBSD strikes again." … I discovered a serious weakness in OpenBSD's PRNG, which allows an attacker to predict the next transaction ID.

Also Klein's 2007 and 2008 analyses of Microsoft IDs.

Bad news, continued:
Many ways for attackers to beat ID+port randomization, even if it's cryptographic.

1. Attack repeatedly.
"An attacker who makes a few billion random guesses is likely to succeed at least once; tens of millions of guesses are adequate with a colliding attack;" etc.

2. Allocate most UDP ports to other tasks, non-reusably.

Also Klein's 2008.02.06 analysis of IDs in OpenBSD, NetBSD, FreeBSD, MacOS X:

> OpenBSD ported BIND 9 into their code tree, but rolled their own PRNG for the DNS transaction ID field). ... "We decided ... to use a more proven algorithm (LCG, Linear Congruential Generator) instead. Thanks to this wise decision, the BIND 9 shipped with OpenBSD does not have this weakness. The proactive security of OpenBSD strikes again." ... I discovered a serious weakness in OpenBSD's PRNG, which allows an attacker to predict the next transaction ID.

Also Klein's 2007 and 2008 analyses of Microsoft IDs.

Bad news, continued:
Many ways for attackers to beat ID+port randomization, even if it's cryptographic.

1. Attack repeatedly.
"An attacker who makes a few billion random guesses is likely to succeed at least once; tens of millions of guesses are adequate with a colliding attack;" etc.

2. Allocate most UDP ports to other tasks, non-reusably.

3. Easy, succeeds instantly: Sniff the network.

lein's 2008.02.06
s of IDs in OpenBSD,
D, FreeBSD, MacOS X:

nBSD ported BIND 9 into
r code tree, but rolled
r own PRNG for the DNS
saction ID field). … "We
ded … to use a more
ven algorithm (LCG, Linear
gruential Generator) instead.
nks to this wise decision, the
D 9 shipped with OpenBSD
s not have this weakness. The
ctive security of OpenBSD
kes again." … I discovered a
ous weakness in OpenBSD's
NG, which allows an attacker
predict the next transaction ID.

lein's 2007 and 2008
es of Microsoft IDs.

Bad news, continued:
Many ways for attackers
to beat ID+port randomization,
even if it's cryptographic.

1. Attack repeatedly.
"An attacker who makes a few
billion random guesses is likely
to succeed at least once; tens of
millions of guesses are adequate
with a colliding attack;" etc.

2. Allocate most UDP ports
to other tasks, non-reusably.

3. Easy, succeeds instantly:
Sniff the network.

Collidi
(2001
aka "bi

Attack
for one
Typica
using 2

Attack
to man
for this

Any ID
i.e., ea
has 20

3.02.06

n OpenBSD,

D, MacOS X:

d BIND 9 into
but rolled
G for the DNS
ield). ... "We
se a more
n (LCG, Linear
nerator) instead.
wise decision, the
with OpenBSD
his weakness. The
y of OpenBSD
.. I discovered a
s in OpenBSD's
lows an attacker
ext transaction ID.

7 and 2008

osoft IDs.

Bad news, continued:
Many ways for attackers
to beat ID+port randomization,
even if it's cryptographic.

1. Attack repeatedly.
"An attacker who makes a few
billion random guesses is likely
to succeed at least once; tens of
millions of guesses are adequate
with a colliding attack;" etc.

2. Allocate most UDP ports
to other tasks, non-reusably.

3. Easy, succeeds instantly:
Sniff the network.

Colliding attacks
(2001 Bernstein)
aka "birthday att

Attacker triggers
for one name `lse`
Typical cache all
using 200 ID+po

Attacker sends fo
to many ID+port
for this name `ls`

Any ID+port col
i.e., each forgery
has $200/2^{32}$ char

D,
S X:

to

IS
Ve

ear
ead.
, the
BSD
. The
SD
red a
D's
cker
on ID.

8

Bad news, continued:

Many ways for attackers
to beat ID+port randomization,
even if it's cryptographic.

1. Attack repeatedly.
"An attacker who makes a few
billion random guesses is likely
to succeed at least once; tens of
millions of guesses are adequate
with a colliding attack;" etc.

2. Allocate most UDP ports
to other tasks, non-reusably.

3. Easy, succeeds instantly:
Sniff the network.

Colliding attacks on caches
(2001 Bernstein),
aka "birthday attacks":

Attacker triggers many que
for one name `lsec.be`.
Typical cache allows 200 q
using 200 ID+port combin

Attacker sends forgeries
to many ID+port combina
for this name `lsec.be`.

Any ID+port collision succ
i.e., each forgery attempt
has $200/2^{32}$ chance of suc

Bad news, continued:
Many ways for attackers to beat ID+port randomization, even if it's cryptographic.

1. Attack repeatedly.
"An attacker who makes a few billion random guesses is likely to succeed at least once; tens of millions of guesses are adequate with a colliding attack;" etc.

2. Allocate most UDP ports to other tasks, non-reusably.

3. Easy, succeeds instantly:
Sniff the network.

Colliding attacks on caches (2001 Bernstein), aka "birthday attacks":

Attacker triggers many queries for one name `lsec.be`.
Typical cache allows 200 queries, using 200 ID+port combinations.

Attacker sends forgeries to many ID+port combinations for this name `lsec.be`.

Any ID+port collision succeeds; i.e., each forgery attempt has $200/2^{32}$ chance of success.

ews, continued:
ways for attackers
t ID+port randomization,
it's cryptographic.

ack repeatedly.
tacker who makes a few
random guesses is likely
ceed at least once; tens of
s of guesses are adequate
colliding attack;" etc.

ocate most UDP ports
er tasks, non-reusably.

y, succeeds instantly:
ne network.

Colliding attacks on caches
(2001 Bernstein),
aka "birthday attacks":

Attacker triggers many queries
for one name `lsec.be`.
Typical cache allows 200 queries,
using 200 ID+port combinations.

Attacker sends forgeries
to many ID+port combinations
for this name `lsec.be`.

Any ID+port collision succeeds;
i.e., each forgery attempt
has $200/2^{32}$ chance of success.

Port-al
(2008.0

Compu
usually

Attack
to talk
tens of
Not all
but *alm*

Compu
reuse t
Cache
Attack
to *thos*

nued:

ttackers

randomization,

ographic.

edly.

o makes a few

uesses is likely

st once; tens of

es are adequate

attack;" etc.

UDP ports

on-reusably.

s instantly:

k.

Colliding attacks on caches
(2001 Bernstein),
aka "birthday attacks":

Attacker triggers many queries
for one name `lsec.be`.
Typical cache allows 200 queries,
using 200 ID+port combinations.

Attacker sends forgeries
to many ID+port combinations
for this name `lsec.be`.

Any ID+port collision succeeds;
i.e., each forgery attempt
has $200/2^{32}$ chance of success.

Port-allocation a
(2008.08 Bernste

Computer with a
usually has more

Attacker convinc
to talk to the at
tens of thousand
Not all available
but *almost* all.

Computer doesn'
reuse those UDP
Cache chooses o
Attacker sends D
to *those* UDP p

ation,

few
kely
ens of
quate
tc.

ts
y.

:

Colliding attacks on caches
(2001 Bernstein),
aka "birthday attacks":

Attacker triggers many queries
for one name `lsec.be`.
Typical cache allows 200 queries,
using 200 ID+port combinations.

Attacker sends forgeries
to many ID+port combinations
for this name `lsec.be`.

Any ID+port collision succeeds;
i.e., each forgery attempt
has $200/2^{32}$ chance of success.

Port-allocation attacks
(2008.08 Bernstein):

Computer with a DNS cac
usually has more servers.

Attacker convinces those s
to talk to the attacker on
tens of thousands of UDP
Not all available UDP port
but *almost* all.

Computer doesn't let the c
reuse those UDP ports.
Cache chooses other UDP
Attacker sends DNS forger
to *those* UDP ports.

Colliding attacks on caches
(2001 Bernstein),
aka "birthday attacks":

Attacker triggers many queries
for one name `lsec.be`.
Typical cache allows 200 queries,
using 200 ID+port combinations.

Attacker sends forgeries
to many ID+port combinations
for this name `lsec.be`.

Any ID+port collision succeeds;
i.e., each forgery attempt
has $200/2^{32}$ chance of success.

Port-allocation attacks
(2008.08 Bernstein):

Computer with a DNS cache
usually has more servers.

Attacker convinces those servers
to talk to the attacker on
tens of thousands of UDP ports.
Not all available UDP ports,
but *almost* all.

Computer doesn't let the cache
reuse those UDP ports.
Cache chooses other UDP ports.
Attacker sends DNS forgeries
to *those* UDP ports.

ng attacks on caches
Bernstein),
irthday attacks":

er triggers many queries
name `lsec.be`.
cache allows 200 queries,
200 ID+port combinations.

er sends forgeries
ny ID+port combinations
name `lsec.be`.

+port collision succeeds;
ch forgery attempt
$0/2^{32}$ chance of success.

Port-allocation attacks
(2008.08 Bernstein):

Computer with a DNS cache
usually has more servers.

Attacker convinces those servers
to talk to the attacker on
tens of thousands of UDP ports.
Not all available UDP ports,
but *almost* all.

Computer doesn't let the cache
reuse those UDP ports.
Cache chooses other UDP ports.
Attacker sends DNS forgeries
to *those* UDP ports.

Clients
a forge
suppres
random
by, e.g.
cache e
limit ca

Many p
Many i
Many b

Mostly
smart b
all com
against

on caches
,
tacks":

many queries
ec.be.
ows 200 queries,
rt combinations.

orgeries
t combinations
ec.be.

lision succeeds;
attempt
nce of success.

Port-allocation attacks
(2008.08 Bernstein):

Computer with a DNS cache
usually has more servers.

Attacker convinces those servers
to talk to the attacker on
tens of thousands of UDP ports.
Not all available UDP ports,
but *almost* all.

Computer doesn't let the cache
reuse those UDP ports.
Cache chooses other UDP ports.
Attacker sends DNS forgeries
to *those* UDP ports.

Clients can try to
a forgery's succes
suppress duplicat
randomly replace
by, e.g., `GooGLe.`
cache entries in c
limit caching; ask

Many performanc
Many interoperab
Many bogus secu

Mostly ineffective
smart blind attac
all completely ine
against sniffing a

s

eries

ueries,
ations.

tions

ceeds;

cess.

Port-allocation attacks (2008.08 Bernstein):

Computer with a DNS cache usually has more servers.

Attacker convinces those servers to talk to the attacker on tens of thousands of UDP ports. Not all available UDP ports, but *almost* all.

Computer doesn't let the cache reuse those UDP ports. Cache chooses other UDP ports. Attacker sends DNS forgeries to *those* UDP ports.

Clients can try to reduce a forgery's success chance: suppress duplicate queries; randomly replace `google.` by, e.g., `GooGLe.cOm`; remo cache entries in case of do limit caching; ask twice; et

Many performance problem
Many interoperability prob
Many bogus security analy

Mostly ineffective against smart blind attackers, and all completely ineffective against sniffing attackers.

Port-allocation attacks (2008.08 Bernstein):

Computer with a DNS cache usually has more servers.

Attacker convinces those servers to talk to the attacker on tens of thousands of UDP ports. Not all available UDP ports, but *almost* all.

Computer doesn't let the cache reuse those UDP ports. Cache chooses other UDP ports. Attacker sends DNS forgeries to *those* UDP ports.

Clients can try to reduce a forgery's success chance: suppress duplicate queries; randomly replace `google.com` by, e.g., `GooGLe.cOm`; remove cache entries in case of doubt; limit caching; ask twice; etc.

Many performance problems. Many interoperability problems. Many bogus security analyses.

Mostly ineffective against smart blind attackers, and all completely ineffective against sniffing attackers.

location attacks
08 Bernstein):

uter with a DNS cache
has more servers.

er convinces those servers
to the attacker on
thousands of UDP ports.
available UDP ports,
most all.

uter doesn't let the cache
hose UDP ports.
chooses other UDP ports.
er sends DNS forgeries
se UDP ports.

Clients can try to reduce a forgery's success chance: suppress duplicate queries; randomly replace `google.com` by, e.g., `GooGLe.cOm`; remove cache entries in case of doubt; limit caching; ask twice; etc.

Many performance problems.
Many interoperability problems.
Many bogus security analyses.

Mostly ineffective against smart blind attackers, and all completely ineffective against sniffing attackers.

Who d

What v
Attack
have tr
Blind a
also ha

ttacks
ein):

DNS cache
servers.

es those servers
tacker on
s of UDP ports.
UDP ports,

't let the cache
ports.
ther UDP ports.
NS forgeries
rts.

Clients can try to reduce a forgery's success chance: suppress duplicate queries; randomly replace `google.com` by, e.g., `GooGLe.cOm`; remove cache entries in case of doubt; limit caching; ask twice; etc.

Many performance problems.
Many interoperability problems.
Many bogus security analyses.

Mostly ineffective against smart blind attackers, and all completely ineffective against sniffing attackers.

Who does DNS t

What we've learn
Attackers sniffing
have trivial contr
Blind attackers a
also have some c

he

ervers

ports.
s,

cache

ports.
ies

Clients can try to reduce
a forgery's success chance:
suppress duplicate queries;
randomly replace `google.com`
by, e.g., `GooGLe.cOm`; remove
cache entries in case of doubt;
limit caching; ask twice; etc.

Many performance problems.
Many interoperability problems.
Many bogus security analyses.

Mostly ineffective against
smart blind attackers, and
all completely ineffective
against sniffing attackers.

<u>Who does DNS trust?</u>

What we've learned:
Attackers sniffing the netw
have trivial control over DI
Blind attackers around the
also have some control.

Clients can try to reduce a forgery's success chance: suppress duplicate queries; randomly replace `google.com` by, e.g., `GooGLe.cOm`; remove cache entries in case of doubt; limit caching; ask twice; etc.

Many performance problems. Many interoperability problems. Many bogus security analyses.

Mostly ineffective against smart blind attackers, and all completely ineffective against sniffing attackers.

Who does DNS trust?

What we've learned: Attackers sniffing the network have trivial control over DNS. Blind attackers around the world also have some control.

Clients can try to reduce a forgery's success chance: suppress duplicate queries; randomly replace `google.com` by, e.g., `GooGLe.cOm`; remove cache entries in case of doubt; limit caching; ask twice; etc.

Many performance problems.
Many interoperability problems.
Many bogus security analyses.

Mostly ineffective against smart blind attackers, and all completely ineffective against sniffing attackers.

Who does DNS trust?

What we've learned:
Attackers sniffing the network have trivial control over DNS.
Blind attackers around the world also have some control.

What if packet forgeries were magically eliminated?
What if all DNS packets had unforgeable sender addresses?
Who would still control DNS?

can try to reduce
ry's success chance:
ss duplicate queries;
nly replace `google.com`
., `GooGLe.cOm`; remove
entries in case of doubt;
aching; ask twice; etc.

performance problems.
nteroperability problems.
bogus security analyses.

ineffective against
blind attackers, and
npletely ineffective
t sniffing attackers.

Who does DNS trust?

What we've learned:
Attackers sniffing the network
have trivial control over DNS.
Blind attackers around the world
also have some control.

What if packet forgeries
were magically eliminated?
What if all DNS packets had
unforgeable sender addresses?
Who would still control DNS?

Origina
specifie
allowec
to cont

Cache
about
Server
canonic
which
`157.22`

Cache
`www.ls`
later as
receive

o reduce
ss chance:
te queries;
e google.com
.c0m; remove
case of doubt;
k twice; etc.

ce problems.
bility problems.
urity analyses.

e against
ckers, and
effective
ttackers.

<u>Who does DNS trust?</u>

What we've learned:
Attackers sniffing the network
have trivial control over DNS.
Blind attackers around the world
also have some control.

What if packet forgeries
were magically eliminated?
What if all DNS packets had
unforgeable sender addresses?
Who would still control DNS?

Original DNS cac
specified in RFC
allowed any DNS
to control all DN

Cache asks SST.
about www.SST.
Server says: www
canonical name w
which has addres
157.22.245.20.

Cache records ad
www.lsec.be. B
later asks about
receives 157.22.

com
ove
ubt;
c.
ns.
lems.
ses.

Who does DNS trust?

What we've learned:
Attackers sniffing the network have trivial control over DNS.
Blind attackers around the world also have some control.

What if packet forgeries were magically eliminated?
What if all DNS packets had unforgeable sender addresses?
Who would still control DNS?

Original DNS cache algorit
specified in RFC 1034
allowed any DNS server
to control all DNS records.

Cache asks `SST.to` DNS s
about `www.SST.to`.
Server says: `www.SST.to` h
canonical name `www.lsec`
which has address
`157.22.245.20`.

Cache records address of
`www.lsec.be`. Browser
later asks about `www.lsec`
receives `157.22.245.20`.

Who does DNS trust?

What we've learned:
Attackers sniffing the network
have trivial control over DNS.
Blind attackers around the world
also have some control.

What if packet forgeries
were magically eliminated?
What if all DNS packets had
unforgeable sender addresses?
Who would still control DNS?

Original DNS cache algorithms
specified in RFC 1034
allowed any DNS server
to control all DNS records.

Cache asks `SST.to` DNS server
about `www.SST.to`.
Server says: `www.SST.to` has
canonical name `www.lsec.be`,
which has address
`157.22.245.20`.

Cache records address of
`www.lsec.be`. Browser
later asks about `www.lsec.be`,
receives `157.22.245.20`.

oes DNS trust?

we've learned:
ers sniffing the network
rivial control over DNS.
ttackers around the world
ve some control.

f packet forgeries
nagically eliminated?
f all DNS packets had
eable sender addresses?
vould still control DNS?

Original DNS cache algorithms
specified in RFC 1034
allowed any DNS server
to control all DNS records.

Cache asks `SST.to` DNS server
about `www.SST.to`.
Server says: `www.SST.to` has
canonical name `www.lsec.be`,
which has address
`157.22.245.20`.

Cache records address of
`www.lsec.be`. Browser
later asks about `www.lsec.be`,
receives `157.22.245.20`.

The "b
(1997

The SS
are aut
the nar
and na
Not au
www.ls

Caches
from th

Bugs c
e.g. BI
micros
that go

trust?

ned:
g the network
rol over DNS.
round the world
control.

orgeries
liminated?
packets had
er addresses?
control DNS?

Original DNS cache algorithms specified in RFC 1034 allowed any DNS server to control all DNS records.

Cache asks SST.to DNS server about www.SST.to.
Server says: www.SST.to has canonical name www.lsec.be, which has address 157.22.245.20.

Cache records address of www.lsec.be. Browser later asks about www.lsec.be, receives 157.22.245.20.

The "bailiwick" f
(1997 BIND; 200

The SST.to DNS
are authorized to
the name SST.to
and names endin
Not authorized to
www.lsec.be.

Caches reject ww
from the SST.to

Bugs continue cr
e.g. BIND bug fi
microsoft.com
that google.com

ork
NS.
world

ad
es?
NS?

Original DNS cache algorithms specified in RFC 1034 allowed any DNS server to control all DNS records.

Cache asks `SST.to` DNS server about `www.SST.to`. Server says: `www.SST.to` has canonical name `www.lsec.be`, which has address `157.22.245.20`.

Cache records address of `www.lsec.be`. Browser later asks about `www.lsec.be`, receives `157.22.245.20`.

The “bailiwick” fix
(1997 BIND; 2003 Microso

The `SST.to` DNS servers are authorized to control the name `SST.to` and names ending `.SST.t`
Not authorized to control `www.lsec.be`.

Caches reject `www.lsec.b`
from the `SST.to` DNS serv

Bugs continue cropping up
e.g. BIND bug fixed 2003:
`microsoft.com` server car
that `google.com` has no a

Original DNS cache algorithms specified in RFC 1034 allowed any DNS server to control all DNS records.

Cache asks `SST.to` DNS server about `www.SST.to`. Server says: `www.SST.to` has canonical name `www.lsec.be`, which has address `157.22.245.20`.

Cache records address of `www.lsec.be`. Browser later asks about `www.lsec.be`, receives `157.22.245.20`.

The "bailiwick" fix (1997 BIND; 2003 Microsoft):

The `SST.to` DNS servers are authorized to control the name `SST.to` and names ending `.SST.to`. Not authorized to control `www.lsec.be`.

Caches reject `www.lsec.be` data from the `SST.to` DNS servers.

Bugs continue cropping up. e.g. BIND bug fixed 2003: `microsoft.com` server can say that `google.com` has no address.

al DNS cache algorithms
ed in RFC 1034
d any DNS server
trol all DNS records.

asks SST.to DNS server
www.SST.to.
says: www.SST.to has
cal name www.lsec.be,
has address
2.245.20.

records address of
sec.be. Browser
sks about www.lsec.be,
s 157.22.245.20.

The "bailiwick" fix
(1997 BIND; 2003 Microsoft):

The SST.to DNS servers
are authorized to control
the name SST.to
and names ending .SST.to.
Not authorized to control
www.lsec.be.

Caches reject www.lsec.be data
from the SST.to DNS servers.

Bugs continue cropping up.
e.g. BIND bug fixed 2003:
microsoft.com server can say
that google.com has no address.

For per
adminis
third-p

e.g. Th
set up
one of
and a t

In 200
broke i
and mi

The rs
no long

che algorithms
1034
S server
IS records.

to DNS server
to.
.SST.to has
www.lsec.be,
ss

ldress of
Browser
www.lsec.be,
245.20.

The “bailiwick” fix
(1997 BIND; 2003 Microsoft):

The `SST.to` DNS servers
are authorized to control
the name `SST.to`
and names ending `.SST.to`.
Not authorized to control
`www.lsec.be`.

Caches reject `www.lsec.be` data
from the `SST.to` DNS servers.

Bugs continue cropping up.
e.g. BIND bug fixed 2003:
`microsoft.com` server can say
that `google.com` has no address.

For performance
administrators so
third-party DNS

e.g. The rsa.co
set up two rsa.c
one of his own co
and a third-party

In 2000, an attac
broke into the th
and misdirected

The rsa.com ad
no longer uses th

thms

erver

has

.be,

.be,

The “bailiwick” fix
(1997 BIND; 2003 Microsoft):

The `SST.to` DNS servers
are authorized to control
the name `SST.to`
and names ending `.SST.to`.
Not authorized to control
`www.lsec.be`.

Caches reject `www.lsec.be` data
from the `SST.to` DNS servers.

Bugs continue cropping up.
e.g. BIND bug fixed 2003:
`microsoft.com` server can say
that `google.com` has no address.

For performance reasons,
administrators sometimes s
third-party DNS servers.

e.g. The `rsa.com` adminis
set up two `rsa.com` server
one of his own computers
and a third-party compute

In 2000, an attacker
broke into the third-party s
and misdirected `www.rsa.`

The `rsa.com` administrato
no longer uses third-party s

The “bailiwick” fix
(1997 BIND; 2003 Microsoft):

The `SST.to` DNS servers
are authorized to control
the name `SST.to`
and names ending `.SST.to`.
Not authorized to control
`www.lsec.be`.

Caches reject `www.lsec.be` data
from the `SST.to` DNS servers.

Bugs continue cropping up.
e.g. BIND bug fixed 2003:
`microsoft.com` server can say
that `google.com` has no address.

For performance reasons,
administrators sometimes set up
third-party DNS servers.

e.g. The `rsa.com` administrator
set up two `rsa.com` servers:
one of his own computers
and a third-party computer.

In 2000, an attacker
broke into the third-party server
and misdirected `www.rsa.com`.

The `rsa.com` administrator
no longer uses third-party servers.

ailiwick" fix
BIND; 2003 Microsoft):

T.to DNS servers
thorized to control
me SST.to
mes ending .SST.to.
thorized to control
sec.be.

reject www.lsec.be data
he SST.to DNS servers.

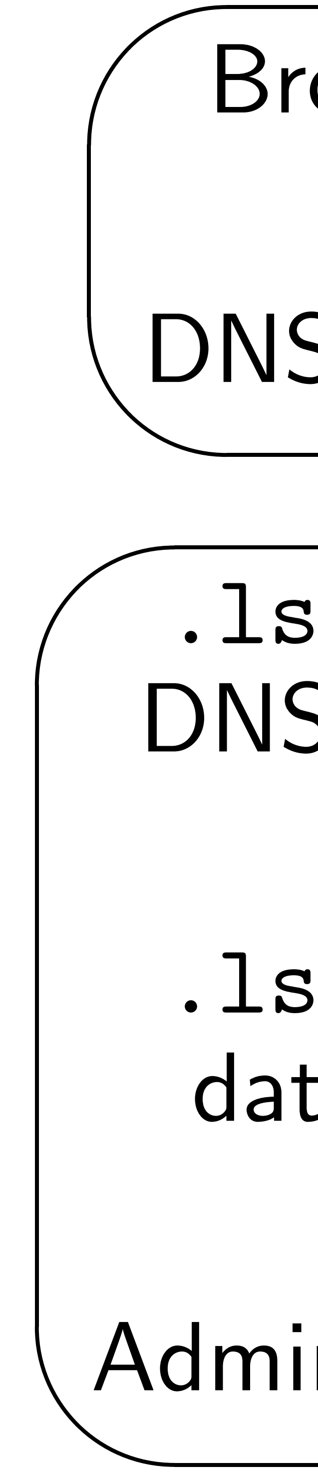
ontinue cropping up.
ND bug fixed 2003:
soft.com server can say
ogle.com has no address.

For performance reasons,
administrators sometimes set up
third-party DNS servers.

e.g. The rsa.com administrator
set up two rsa.com servers:
one of his own computers
and a third-party computer.

In 2000, an attacker
broke into the third-party server
and misdirected www.rsa.com.

The rsa.com administrator
no longer uses third-party servers.



fix
03 Microsoft):

S servers
control
o
g `.SST.to`.
o control

w`.lsec.be` data
DNS servers.

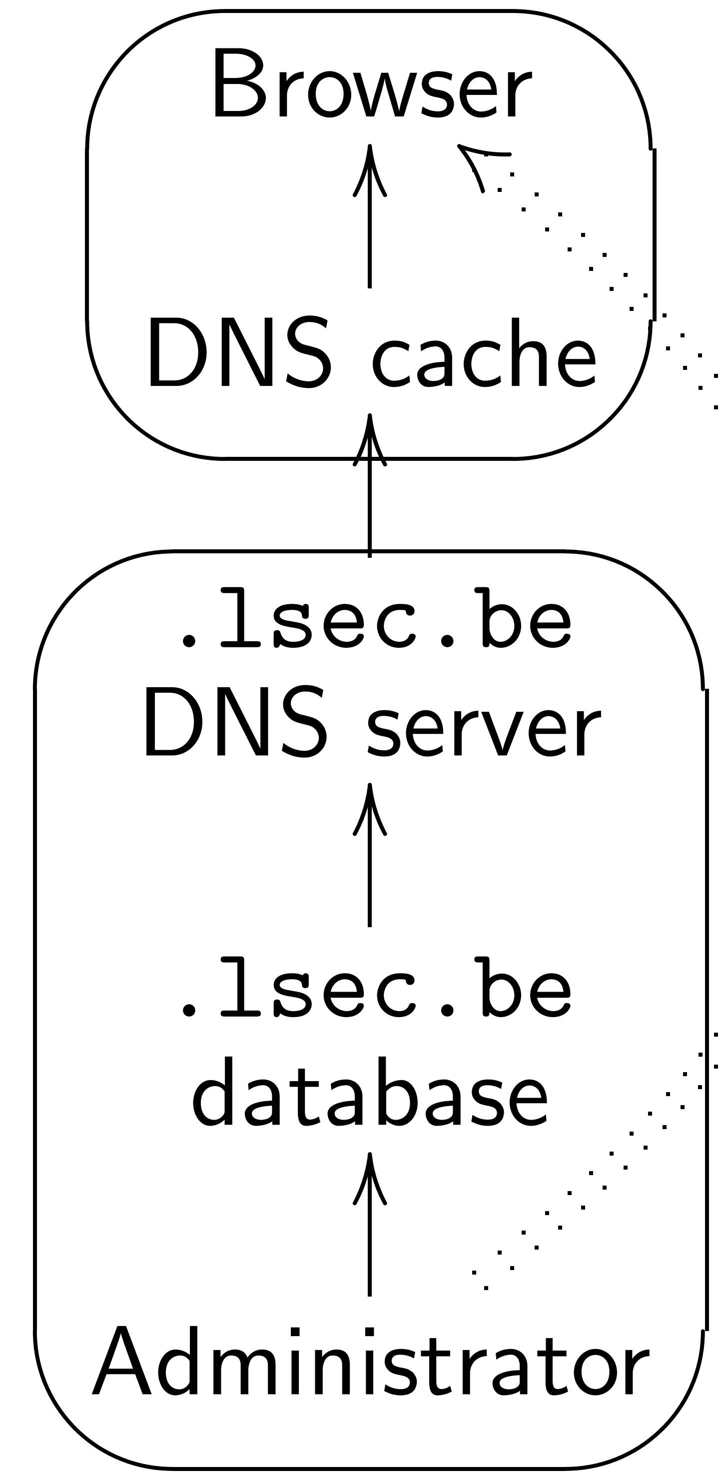
ropping up.
ixed 2003:
server can say
n has no address.

For performance reasons, administrators sometimes set up third-party DNS servers.

e.g. The `rsa.com` administrator set up two `rsa.com` servers: one of his own computers and a third-party computer.

In 2000, an attacker broke into the third-party server and misdirected `www.rsa.com`.

The `rsa.com` administrator no longer uses third-party servers.



oft):

o.

e data
vers.

.

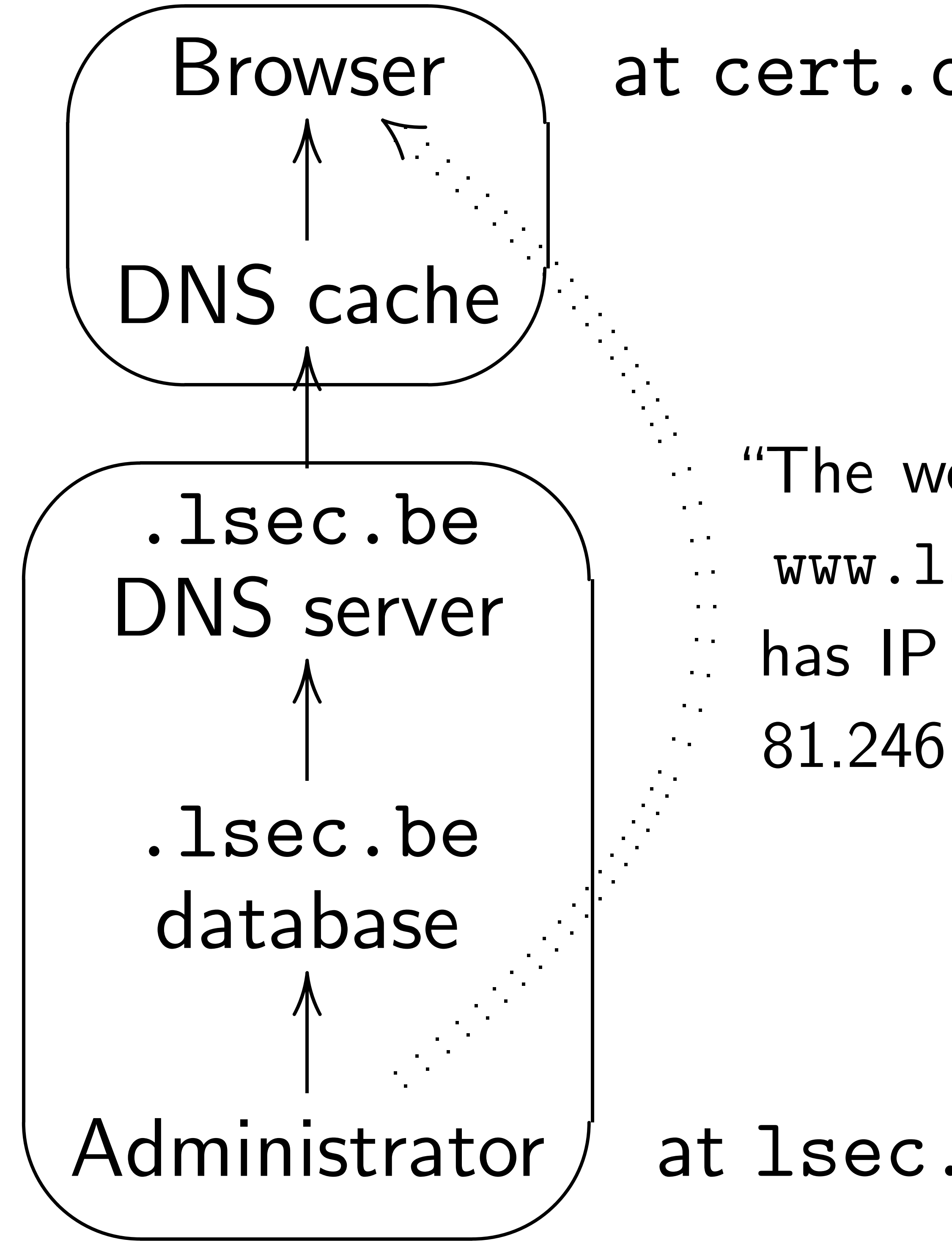
n say
ddress.

For performance reasons, administrators sometimes set up third-party DNS servers.

e.g. The `rsa.com` administrator set up two `rsa.com` servers: one of his own computers and a third-party computer.

In 2000, an attacker broke into the third-party server and misdirected `www.rsa.com`.

The `rsa.com` administrator no longer uses third-party servers.

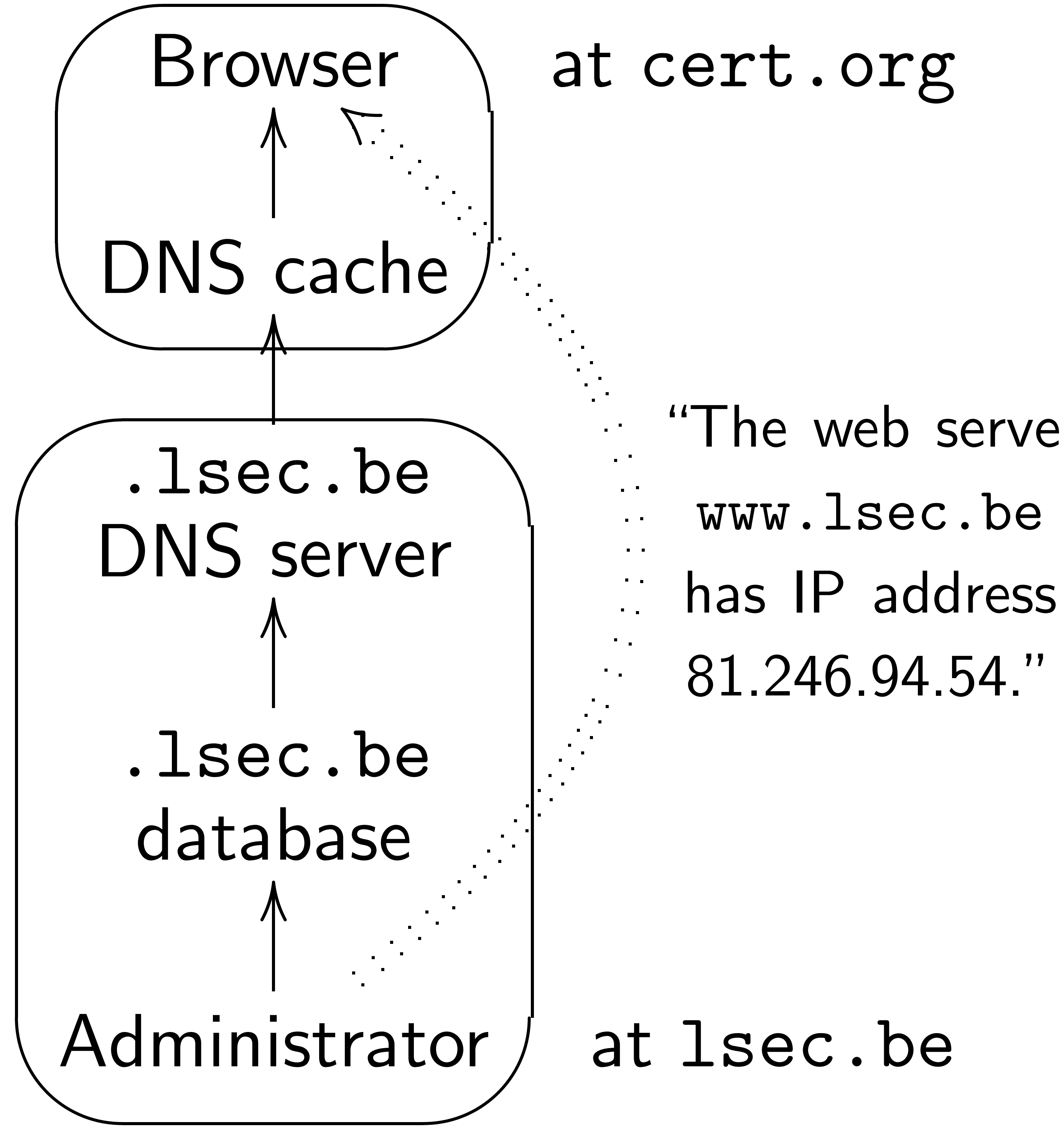


For performance reasons, administrators sometimes set up third-party DNS servers.

e.g. The `rsa.com` administrator set up two `rsa.com` servers: one of his own computers and a third-party computer.

In 2000, an attacker broke into the third-party server and misdirected `www.rsa.com`.

The `rsa.com` administrator no longer uses third-party servers.

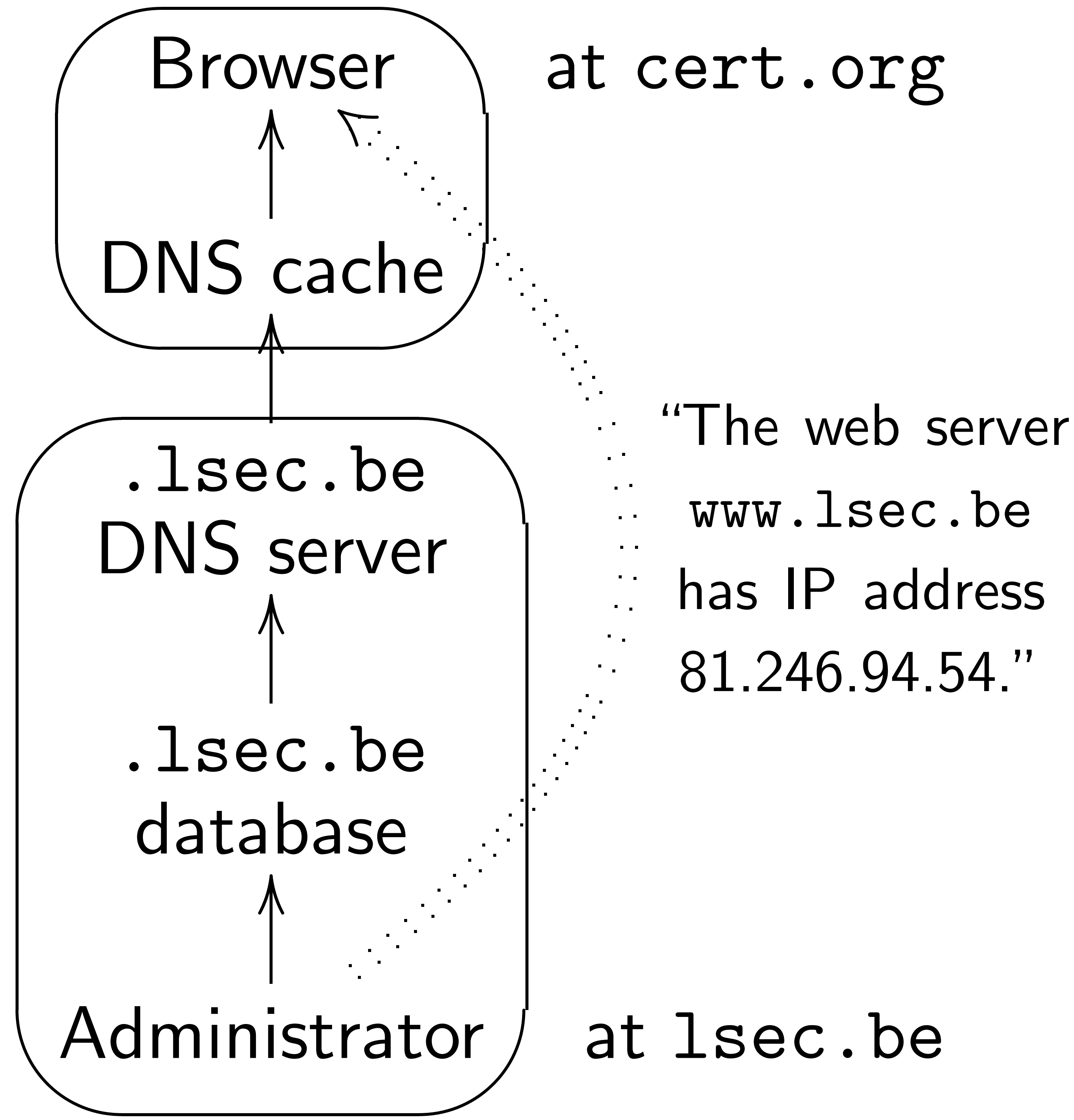


rformance reasons,
strators sometimes set up
arty DNS servers.

he rsa.com administrator
two rsa.com servers:
his own computers
third-party computer.

0, an attacker
nto the third-party server
sdirected www.rsa.com.

sa.com administrator
ger uses third-party servers.



DNS ca
.lsec.
.be DN

a

“The D
for .l
is
with IP
80.92.

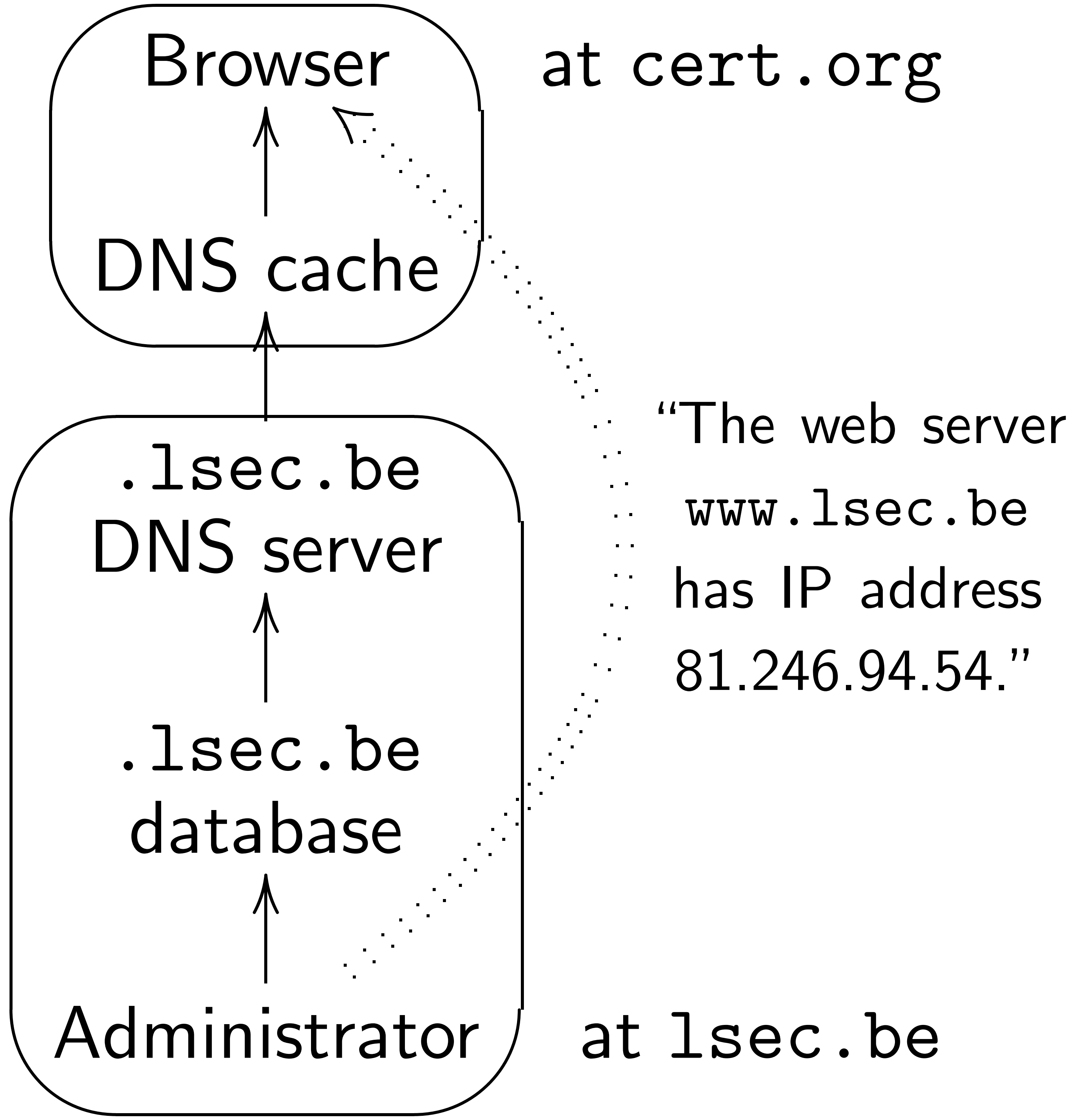
a

reasons,
ometimes set up
servers.

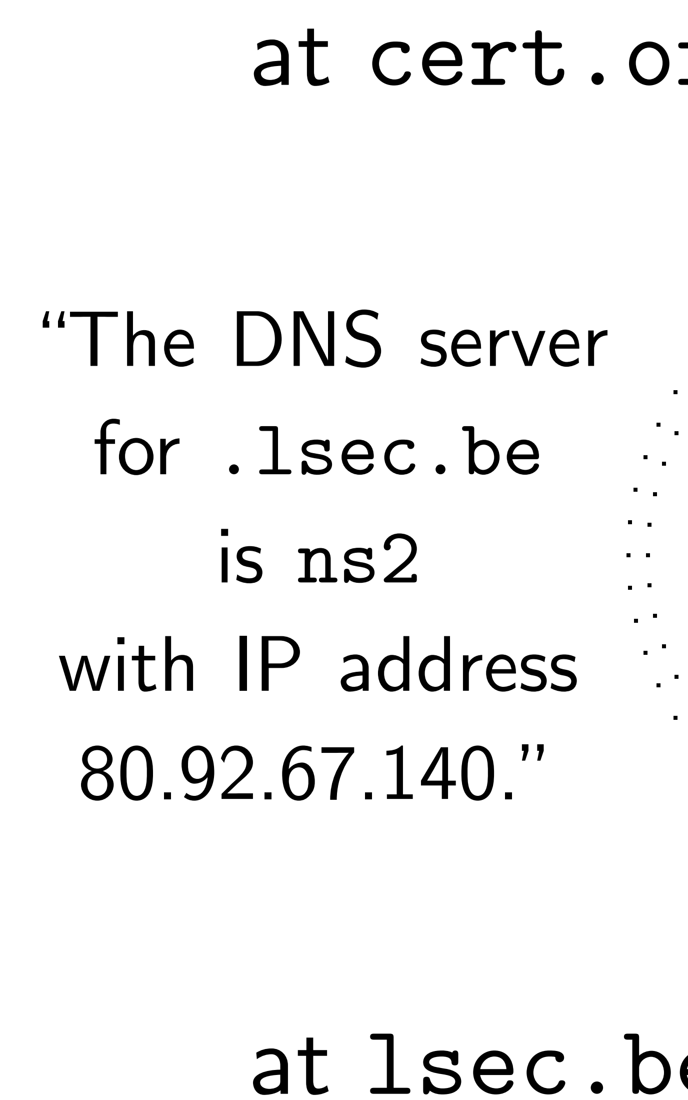
om administrator
com servers:
omputers
computer.

cker
ird-party server
www.rsa.com.

ministrator
hird-party servers.



DNS cache learn
.lsec.be DNS s
.be DNS server:



set up

trator

s:

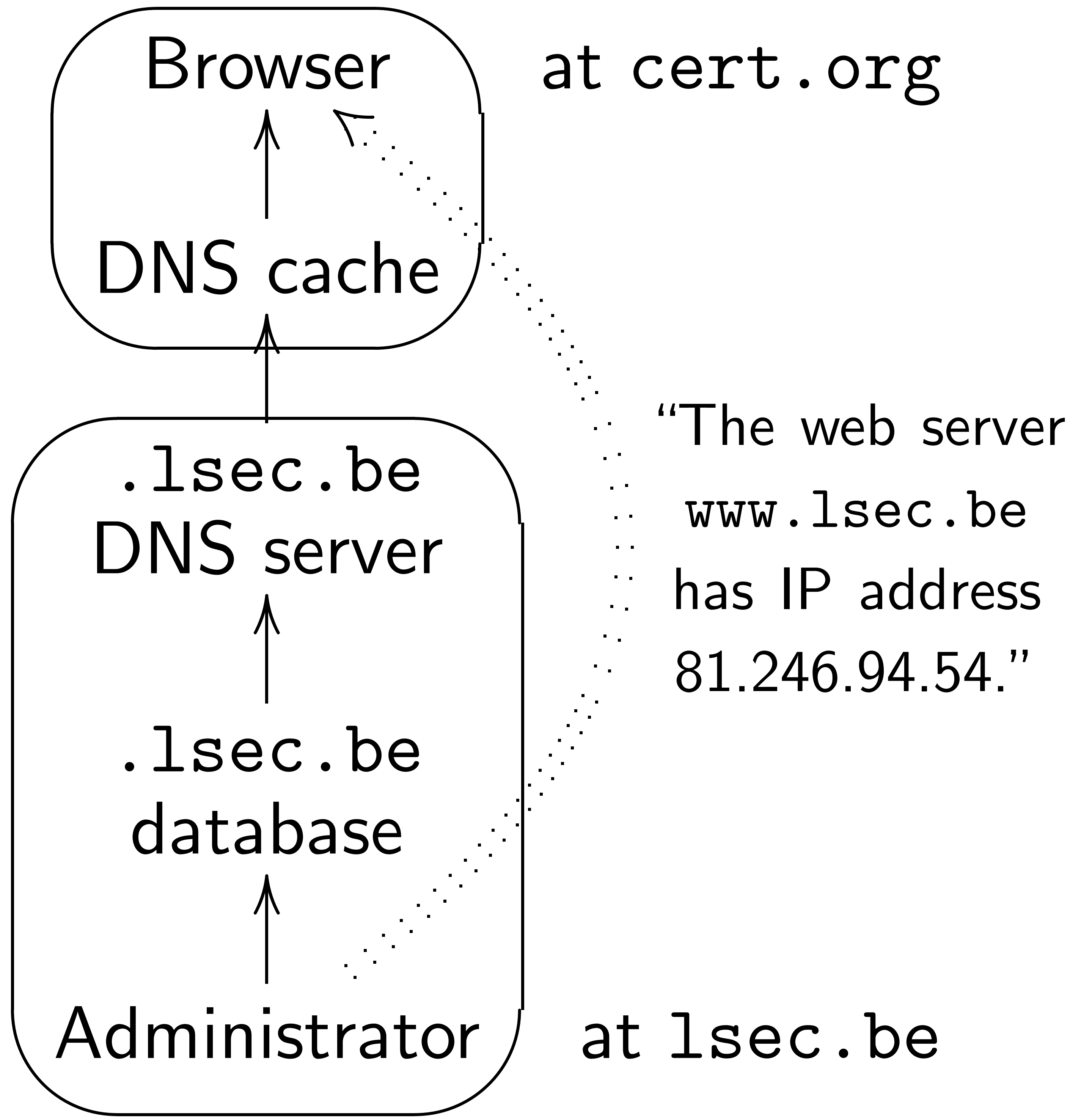
r.

server

com.

r

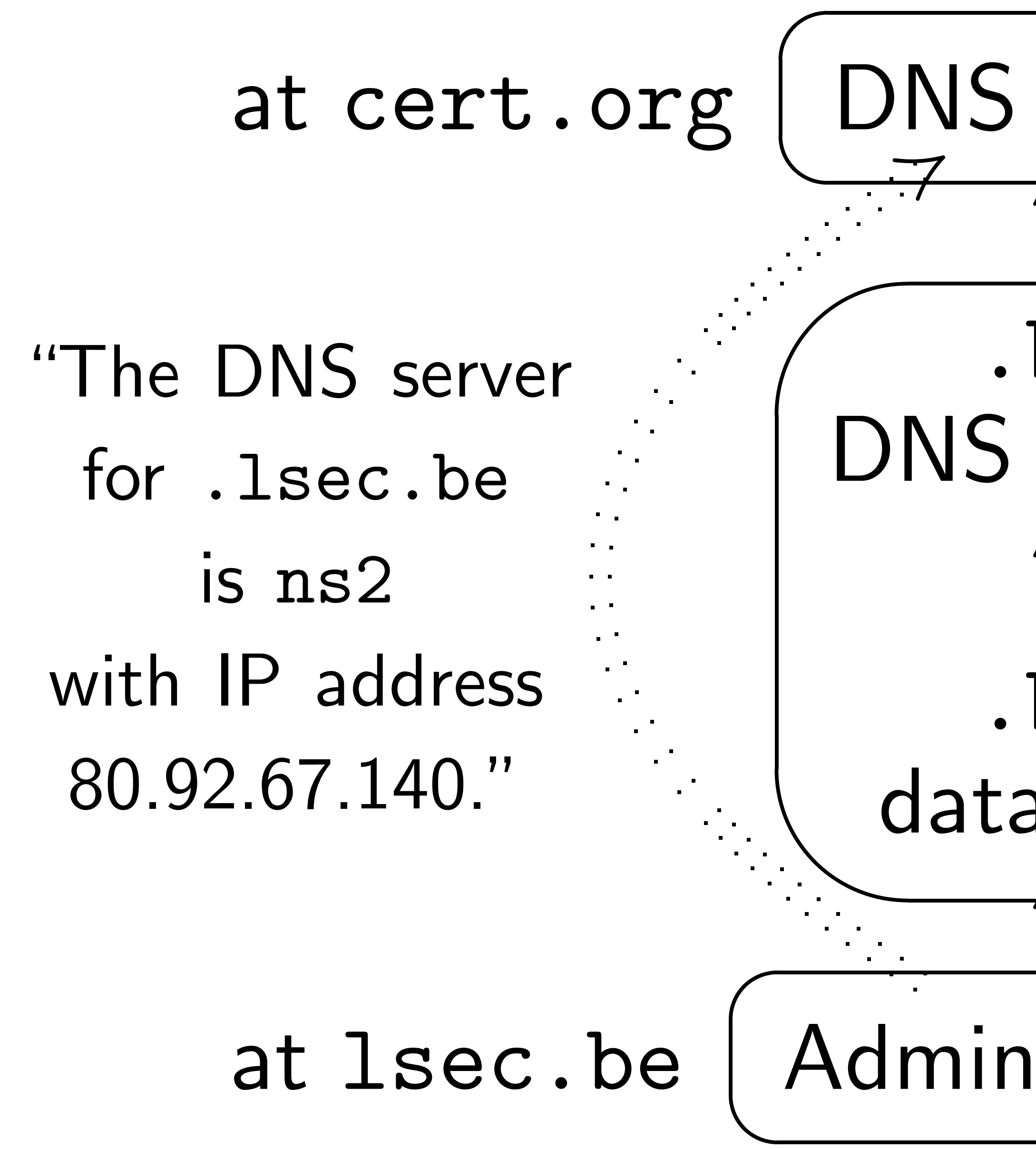
servers.

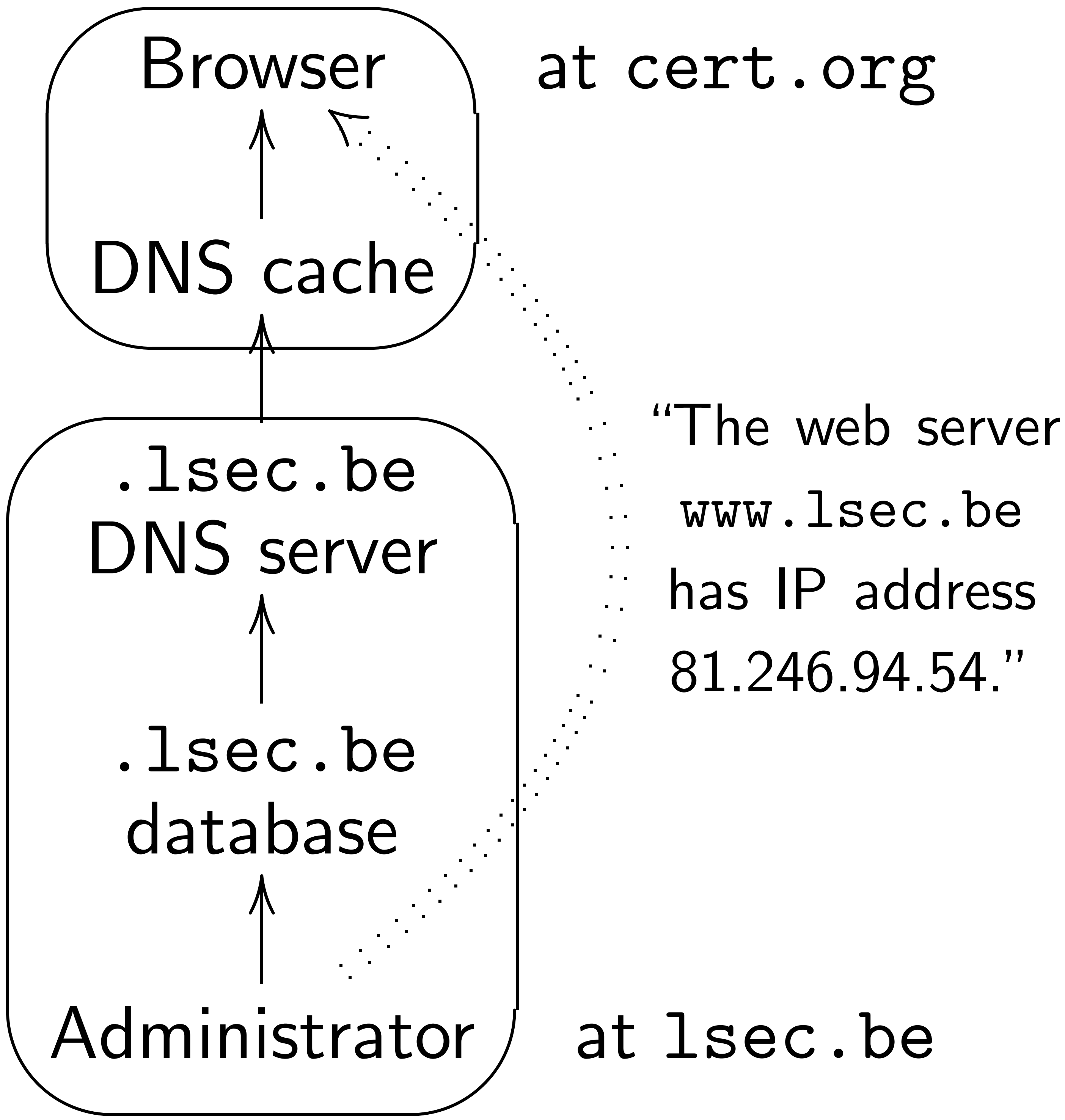


DNS cache learns location

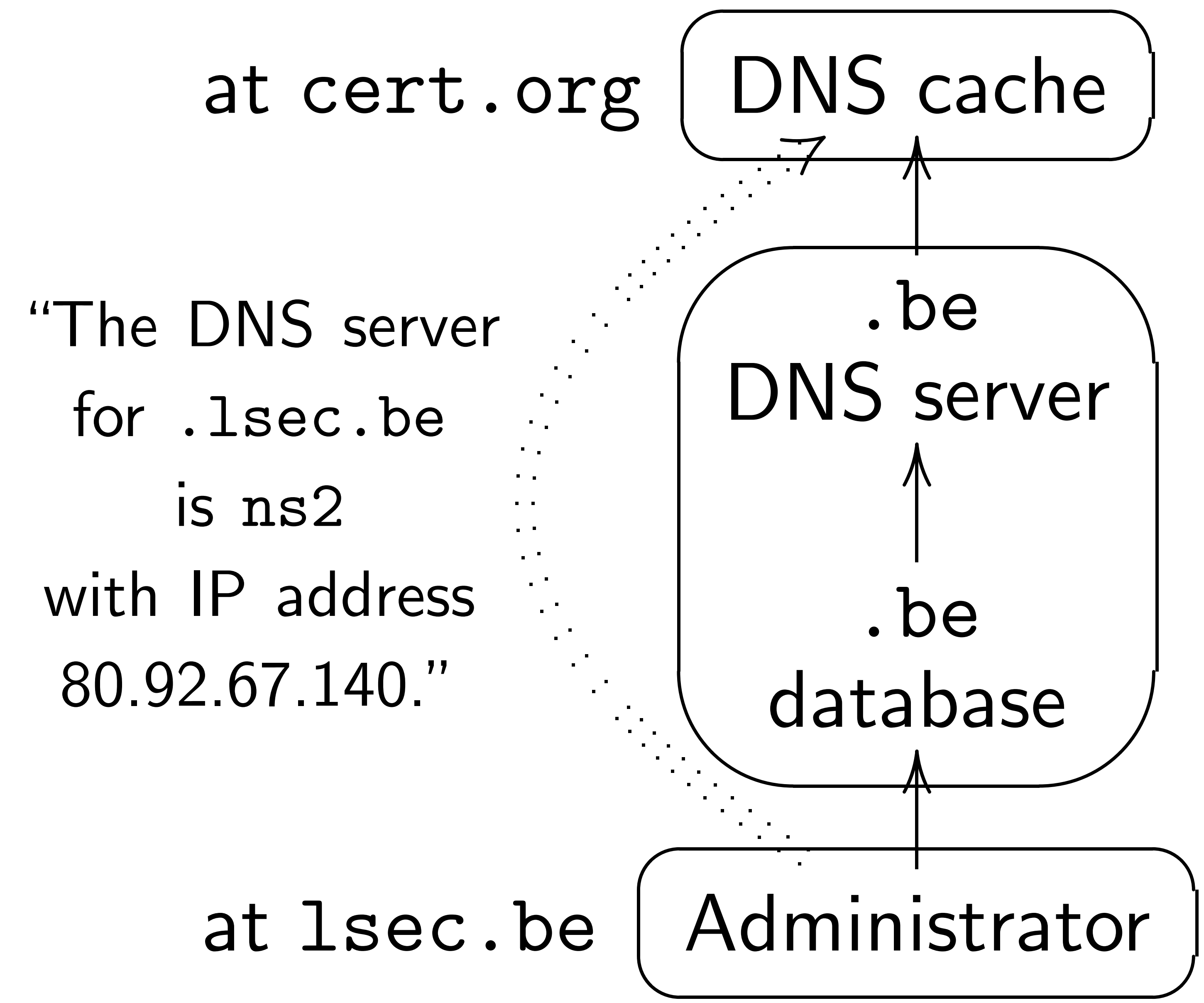
.lsec.be DNS server fron

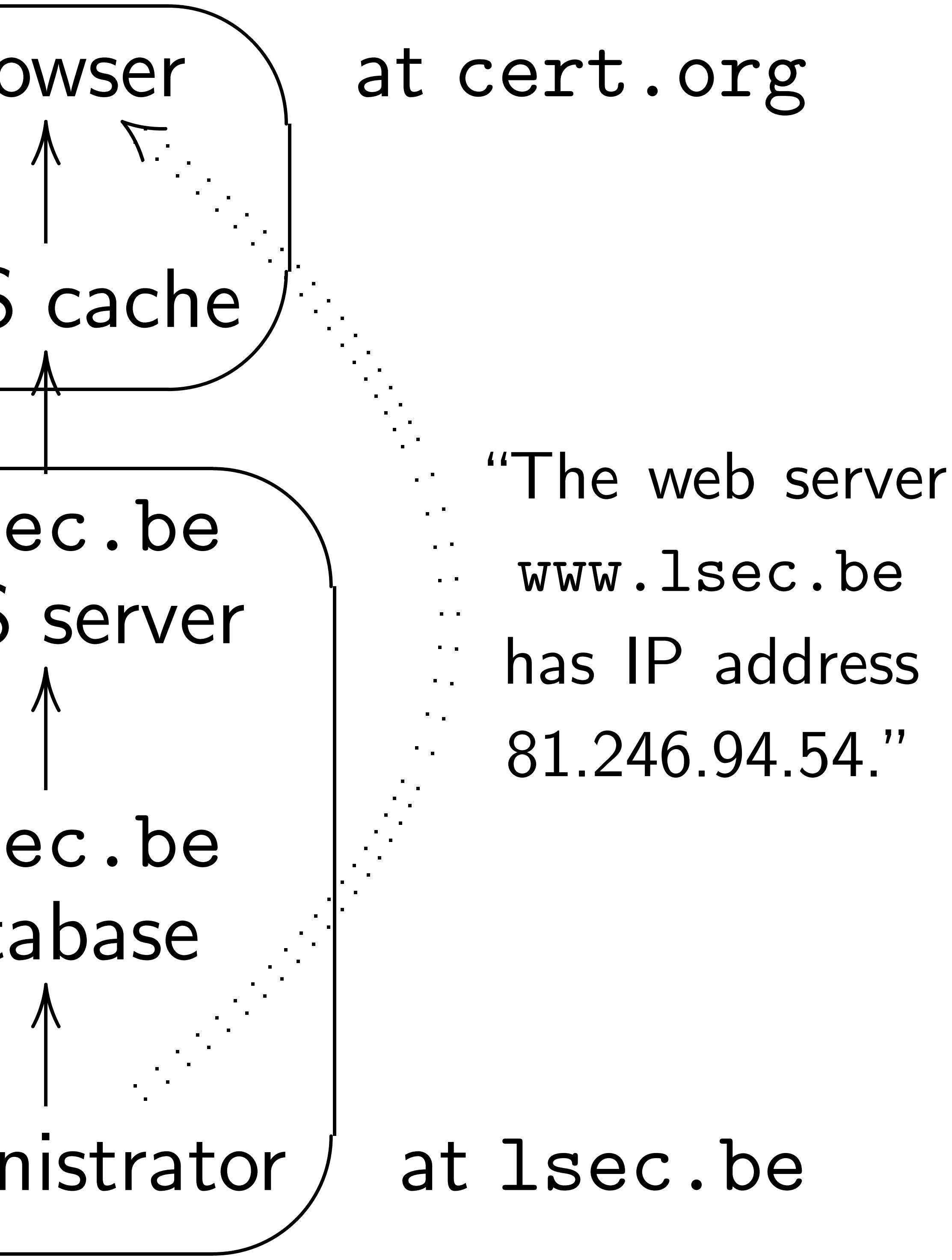
.be DNS server:



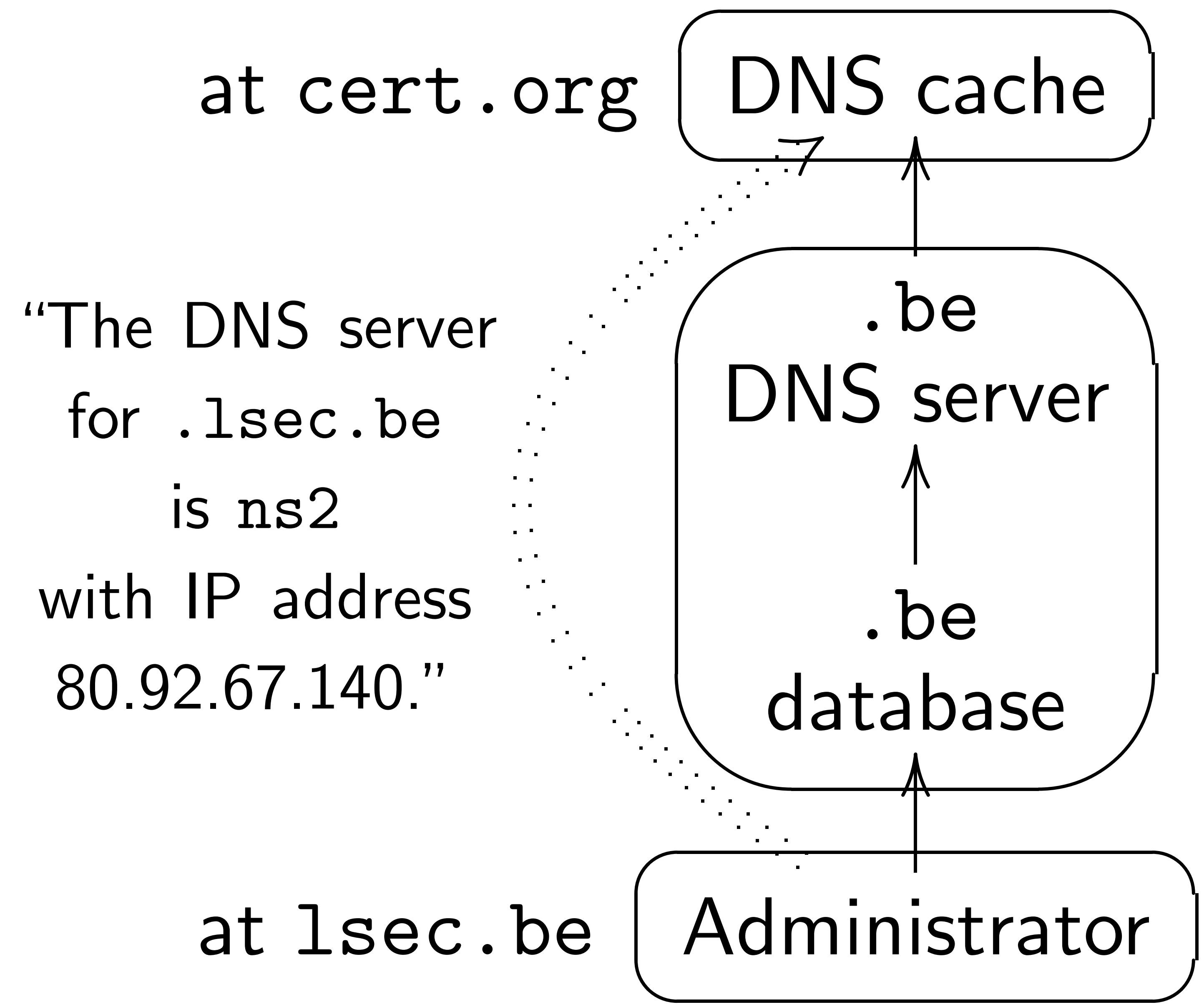


DNS cache learns location of `.lsec.be` DNS server from `.be` DNS server:



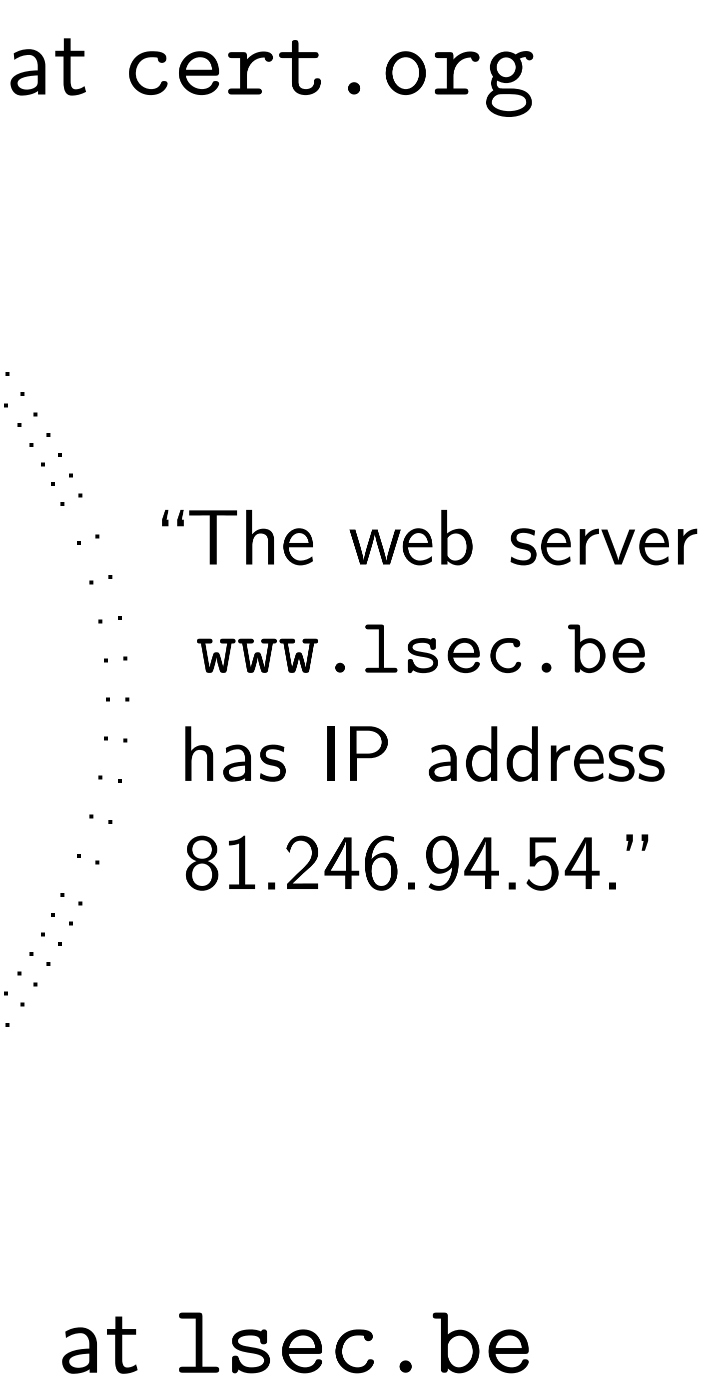


DNS cache learns location of
.lsec.be DNS server from
.be DNS server:

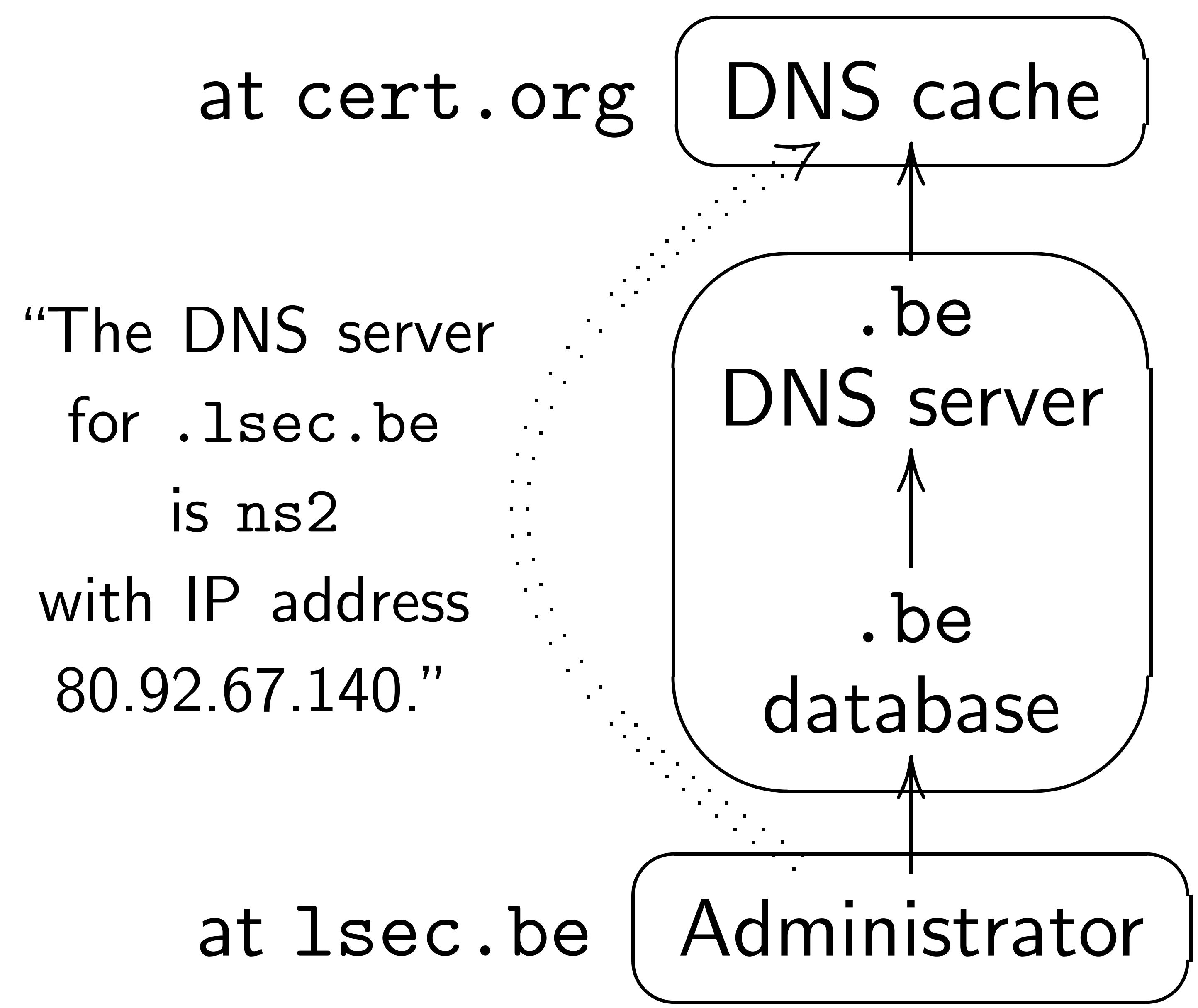


All pac

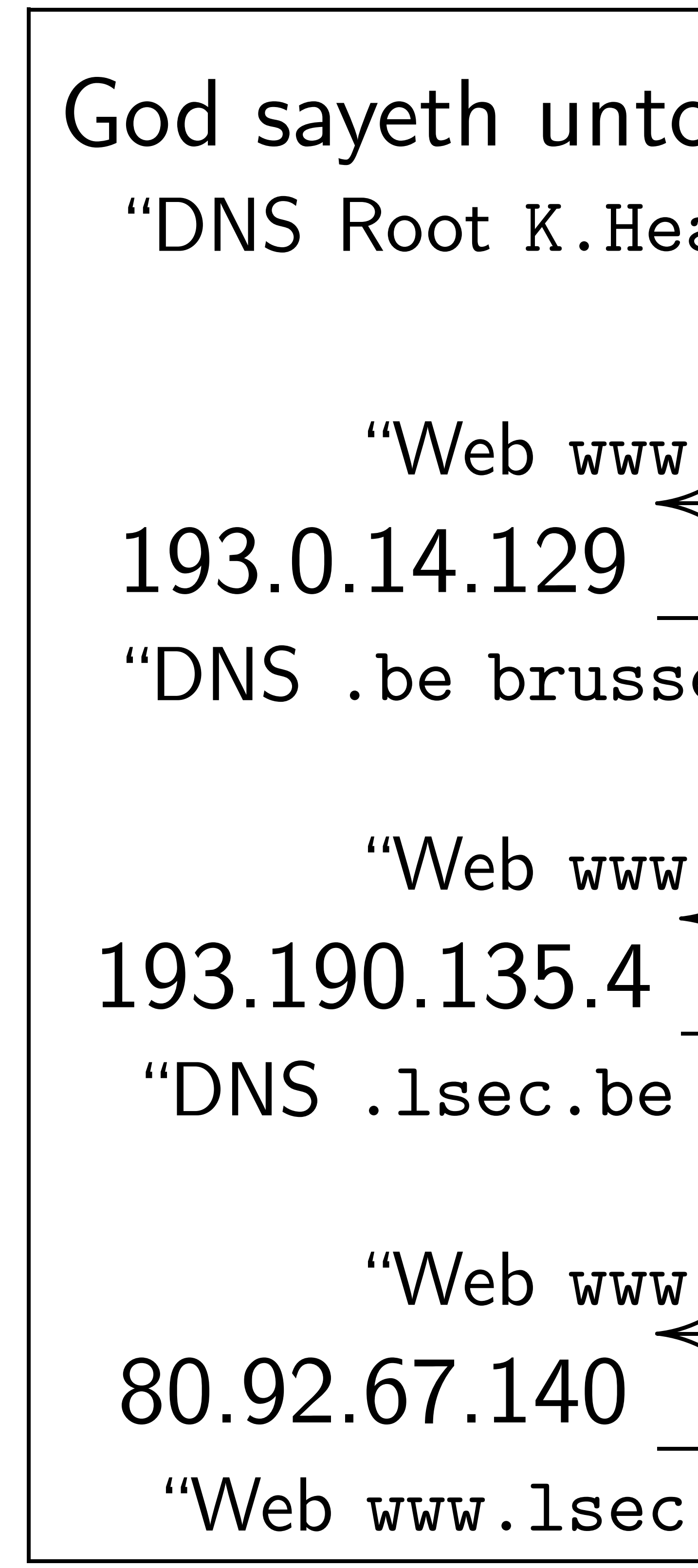
God sa
"DNS
193.0
"DNS
193.1
"DNS
80.92
"Wel



DNS cache learns location of
.lsec.be DNS server from
.be DNS server:



All packets to/fr

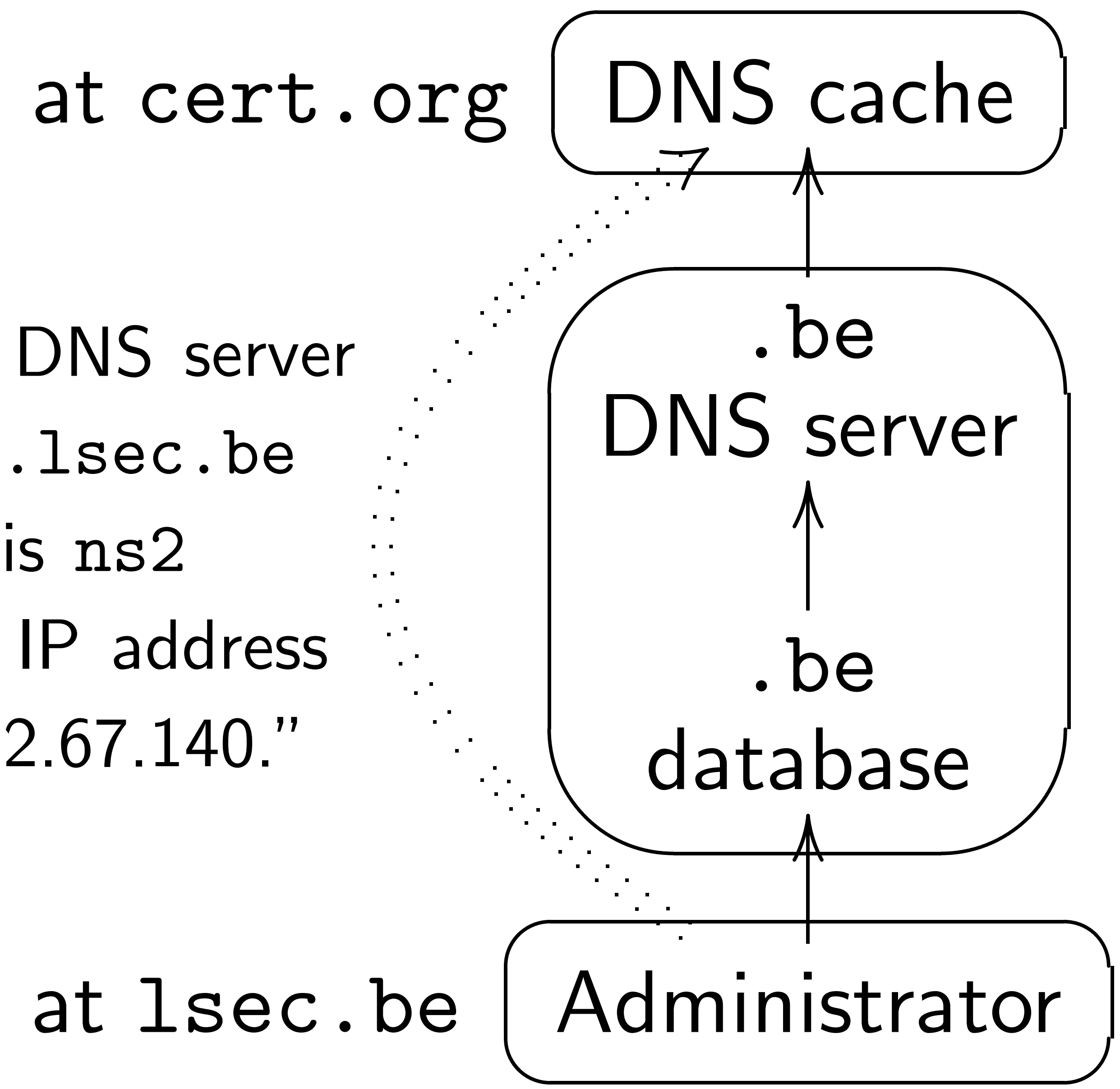


org

eb server
sec.be
address
.94.54.”

.be

DNS cache learns location of
.lsec.be DNS server from
.be DNS server:



All packets to/from DNS c

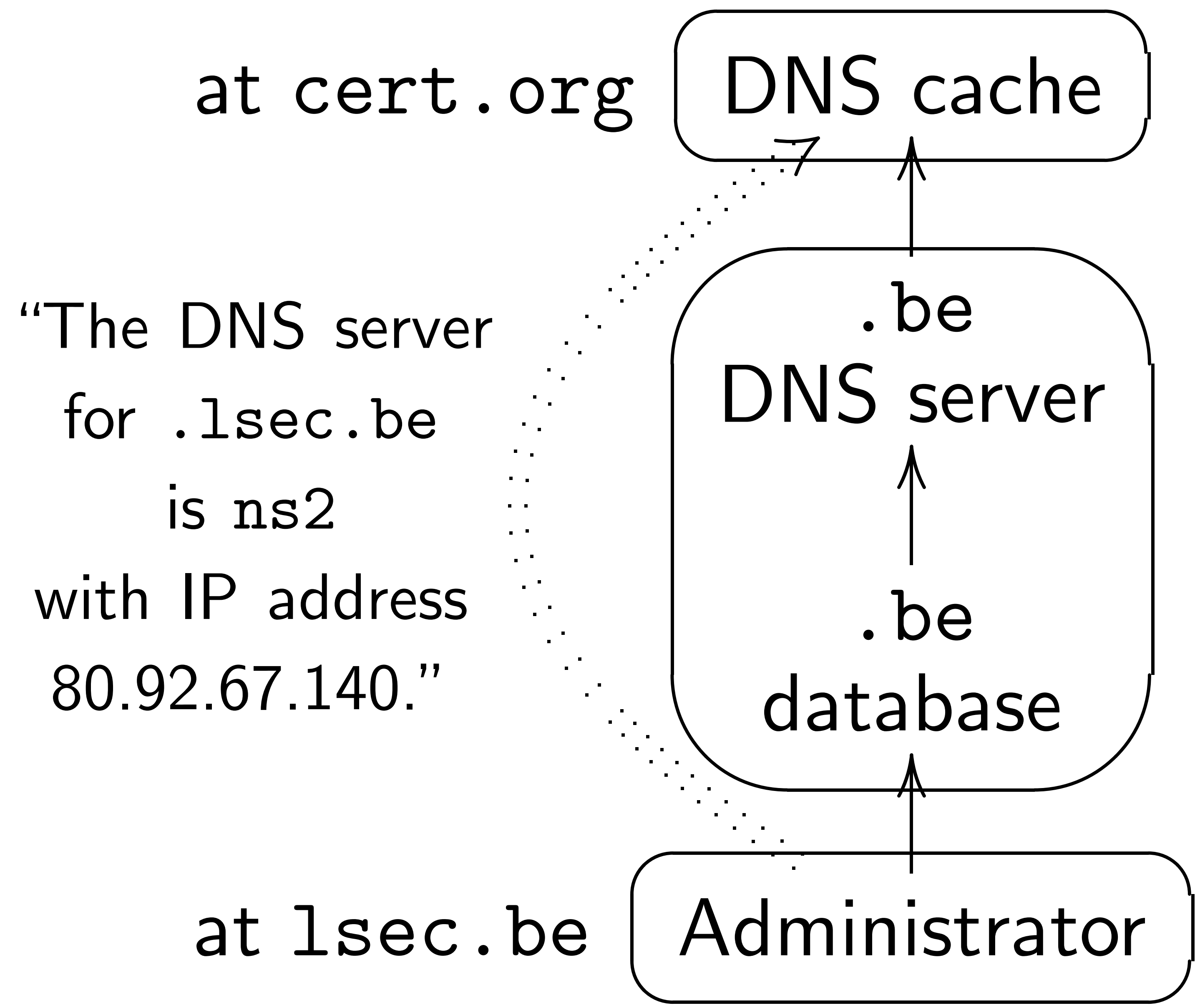
God sayeth unto the DNS
“DNS Root K.Heaven 193.0.1

“Web www.lsec.be?”
193.0.14.129 DNS c
“DNS .be brussels 193.190

“Web www.lsec.be?”
193.190.135.4 DNS c
“DNS .lsec.be ns2 80.92.6

“Web www.lsec.be?”
80.92.67.140 DNS c
“Web www.lsec.be 81.246.9

DNS cache learns location of `.lsec.be` DNS server from `.be` DNS server:



All packets to/from DNS cache:

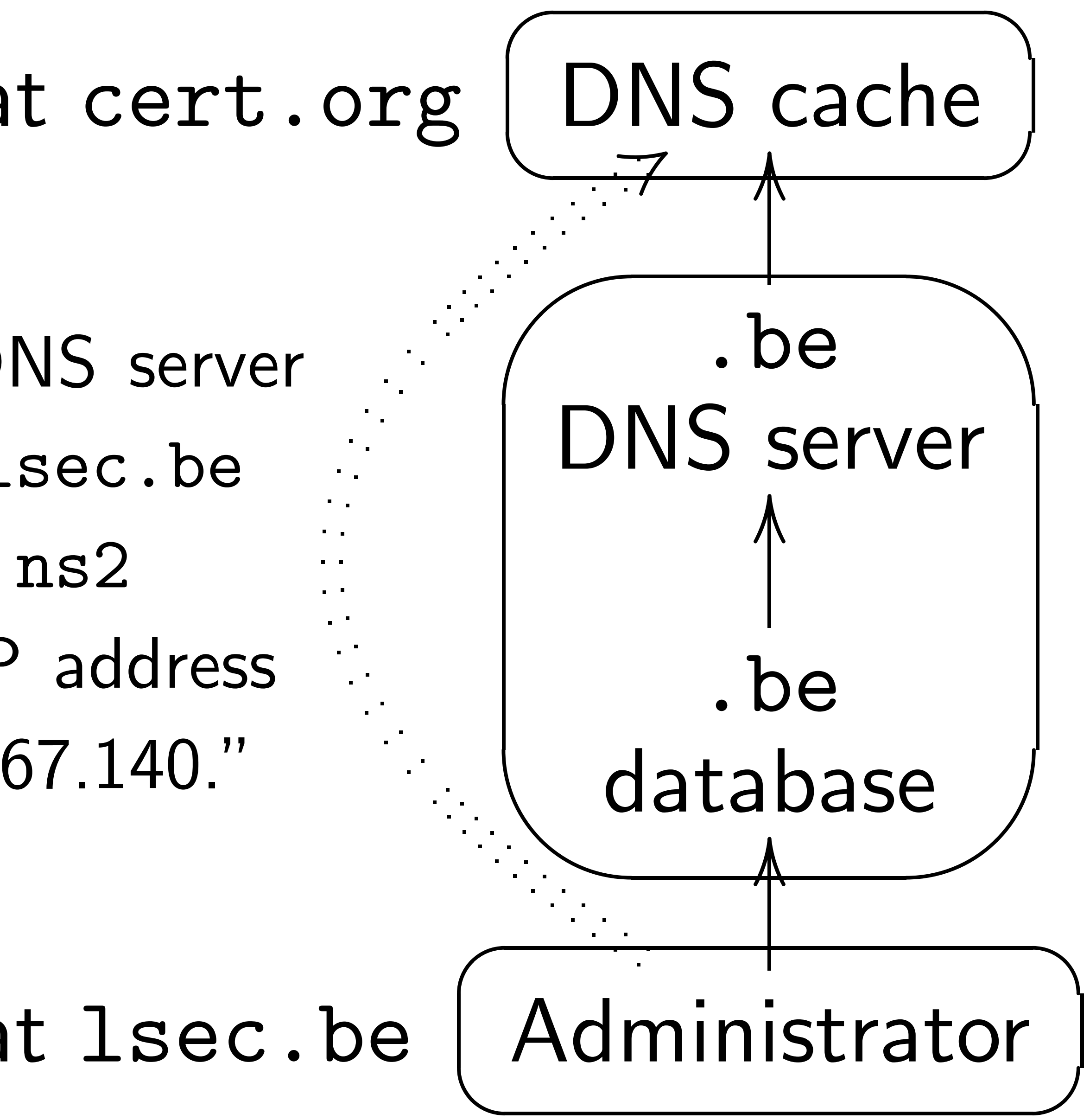
God sayeth unto the DNS cache:
"DNS Root `K.Heaven` 193.0.14.129"

"Web `www.lsec.be`?"
193.0.14.129 ⟵ DNS cache
193.0.14.129 ⟶ DNS cache
"DNS `.be brussels` 193.190.135.4"

"Web `www.lsec.be`?"
193.190.135.4 ⟵ DNS cache
193.190.135.4 ⟶ DNS cache
"DNS `.lsec.be ns2` 80.92.67.140"

"Web `www.lsec.be`?"
80.92.67.140 ⟵ DNS cache
80.92.67.140 ⟶ DNS cache
"Web `www.lsec.be` 81.246.94.54"

ache learns location of
.be DNS server from
NS server:



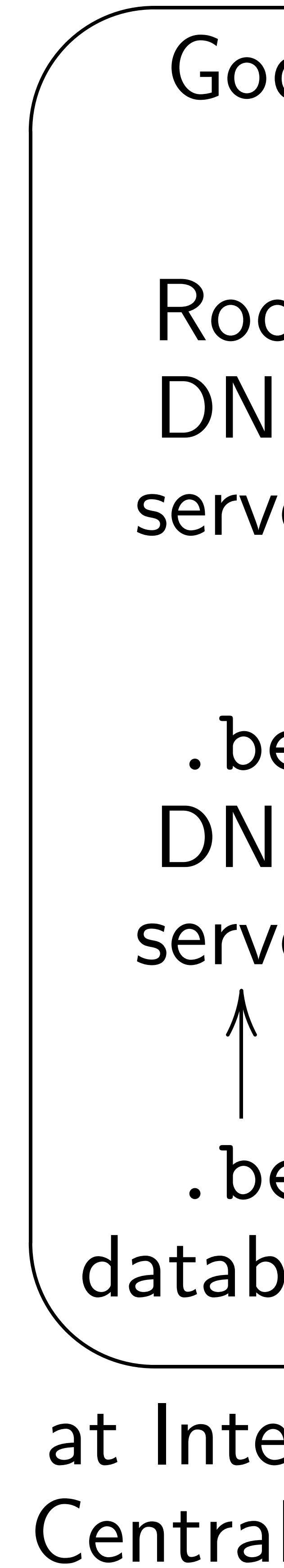
All packets to/from DNS cache:

God sayeth unto the DNS cache:
"DNS Root `K.Heaven` 193.0.14.129"

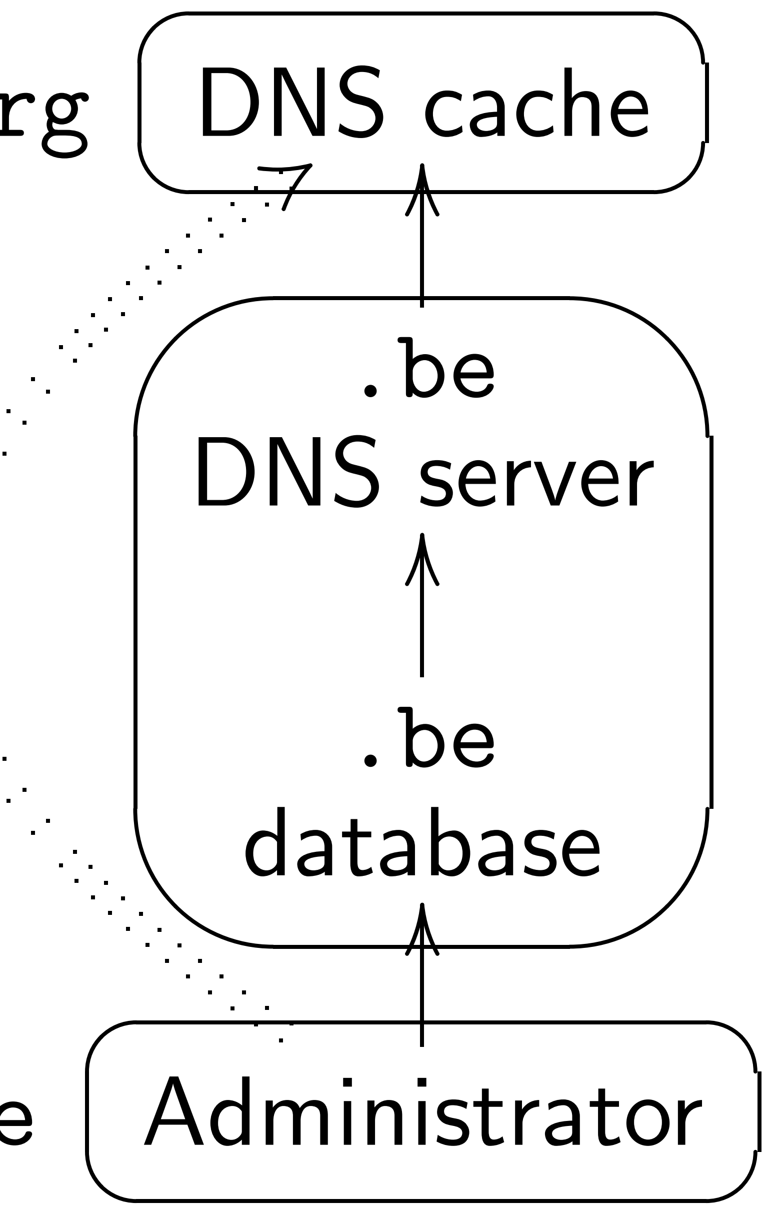
"Web `www.lsec.be`?"
193.0.14.129 ⟵ ⟶ DNS cache
"DNS `.be brussels` 193.190.135.4"

"Web `www.lsec.be`?"
193.190.135.4 ⟵ ⟶ DNS cache
"DNS `.lsec.be ns2` 80.92.67.140"

"Web `www.lsec.be`?"
80.92.67.140 ⟵ ⟶ DNS cache
"Web `www.lsec.be` 81.246.94.54"



s location of
server from



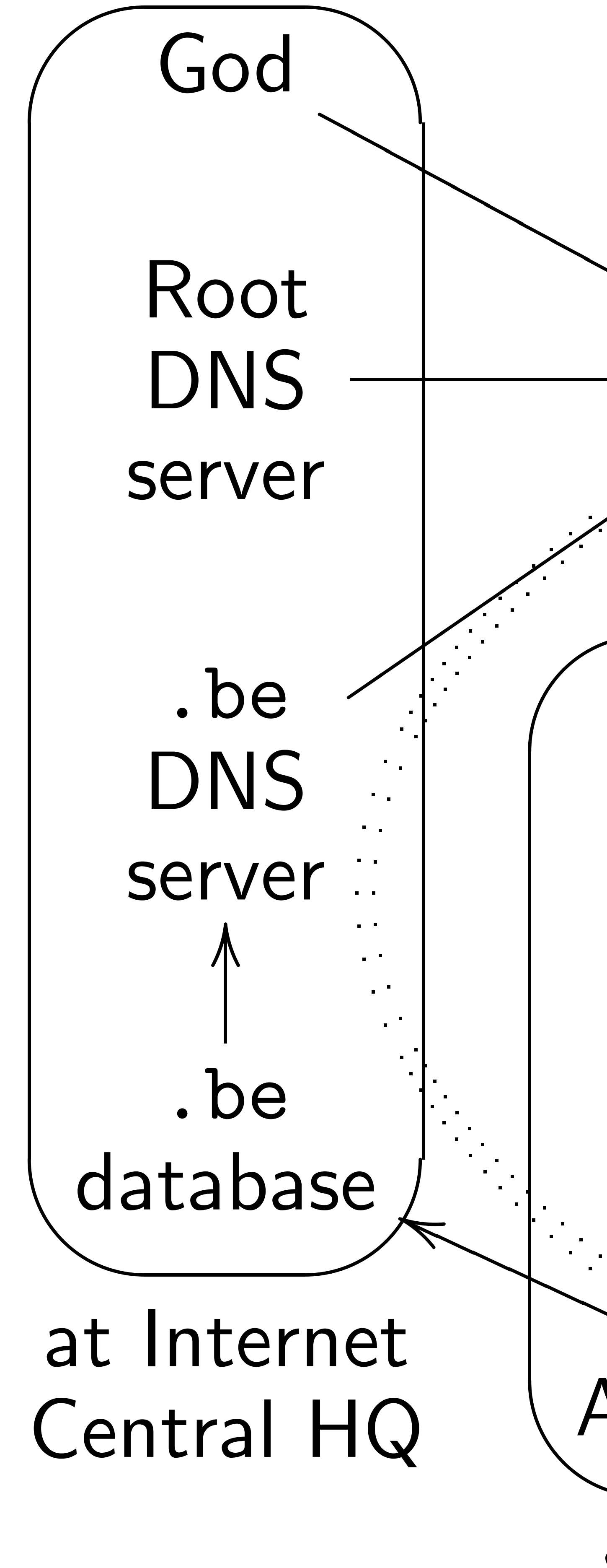
All packets to/from DNS cache:

God sayeth unto the DNS cache:
"DNS Root `K.Heaven` 193.0.14.129"

"Web `www.lsec.be`?"
193.0.14.129 ← → DNS cache
"DNS `.be brussels` 193.190.135.4"

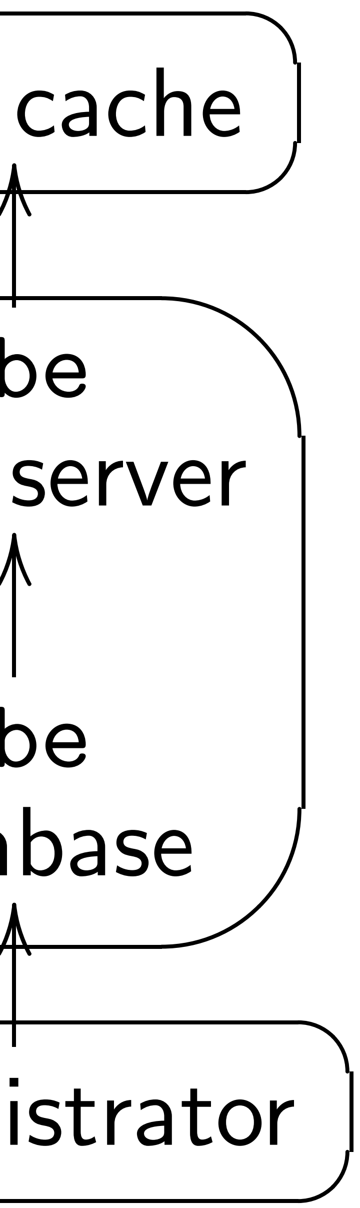
"Web `www.lsec.be`?"
193.190.135.4 ← → DNS cache
"DNS `.lsec.be ns2` 80.92.67.140"

"Web `www.lsec.be`?"
80.92.67.140 ← → DNS cache
"Web `www.lsec.be` 81.246.94.54"



of

n



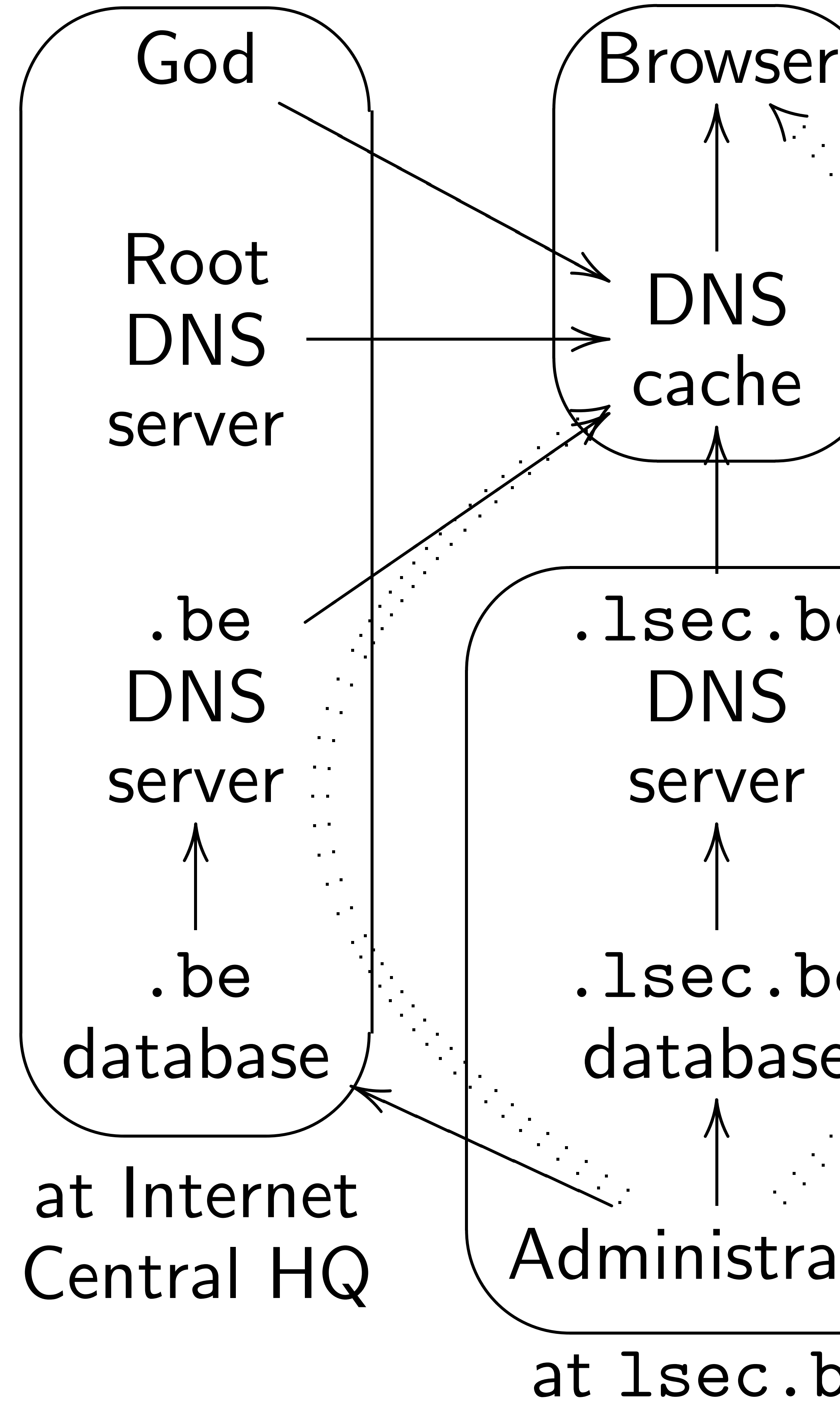
All packets to/from DNS cache:

God sayeth unto the DNS cache:
"DNS Root K.Heaven 193.0.14.129"

"Web www.lsec.be?"
193.0.14.129 ⇄ DNS cache
"DNS .be brussels 193.190.135.4"

"Web www.lsec.be?"
193.190.135.4 ⇄ DNS cache
"DNS .lsec.be ns2 80.92.67.140"

"Web www.lsec.be?"
80.92.67.140 ⇄ DNS cache
"Web www.lsec.be 81.246.94.54"



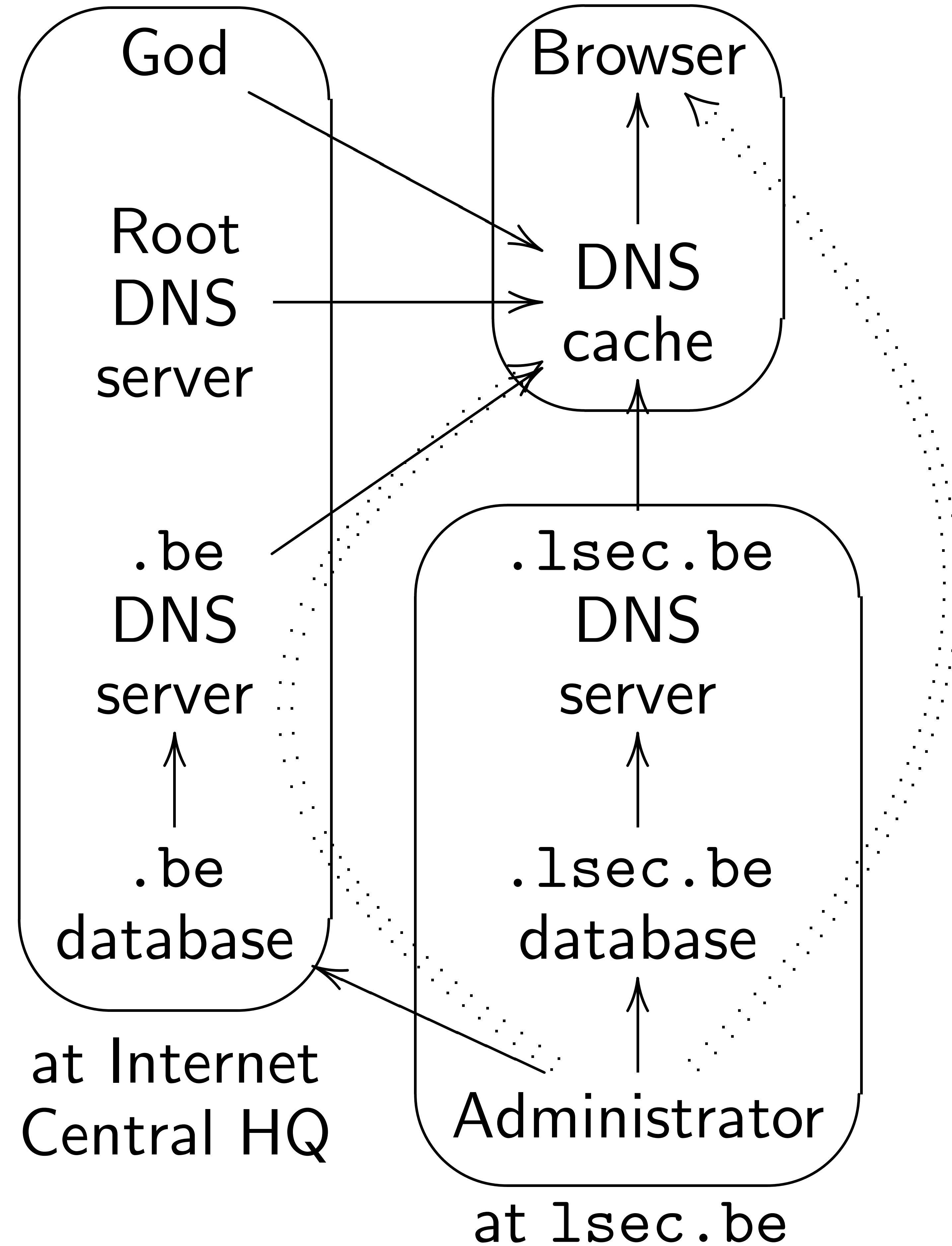
All packets to/from DNS cache:

God sayeth unto the DNS cache:
“DNS Root `K.Heaven` 193.0.14.129”

“Web `www.lsec.be`?”
193.0.14.129 ⟵ ⟶ DNS cache
“DNS `.be brussels` 193.190.135.4”

“Web `www.lsec.be`?”
193.190.135.4 ⟵ ⟶ DNS cache
“DNS `.lsec.be ns2` 80.92.67.140”

“Web `www.lsec.be`?”
80.92.67.140 ⟵ ⟶ DNS cache
“Web `www.lsec.be` 81.246.94.54”



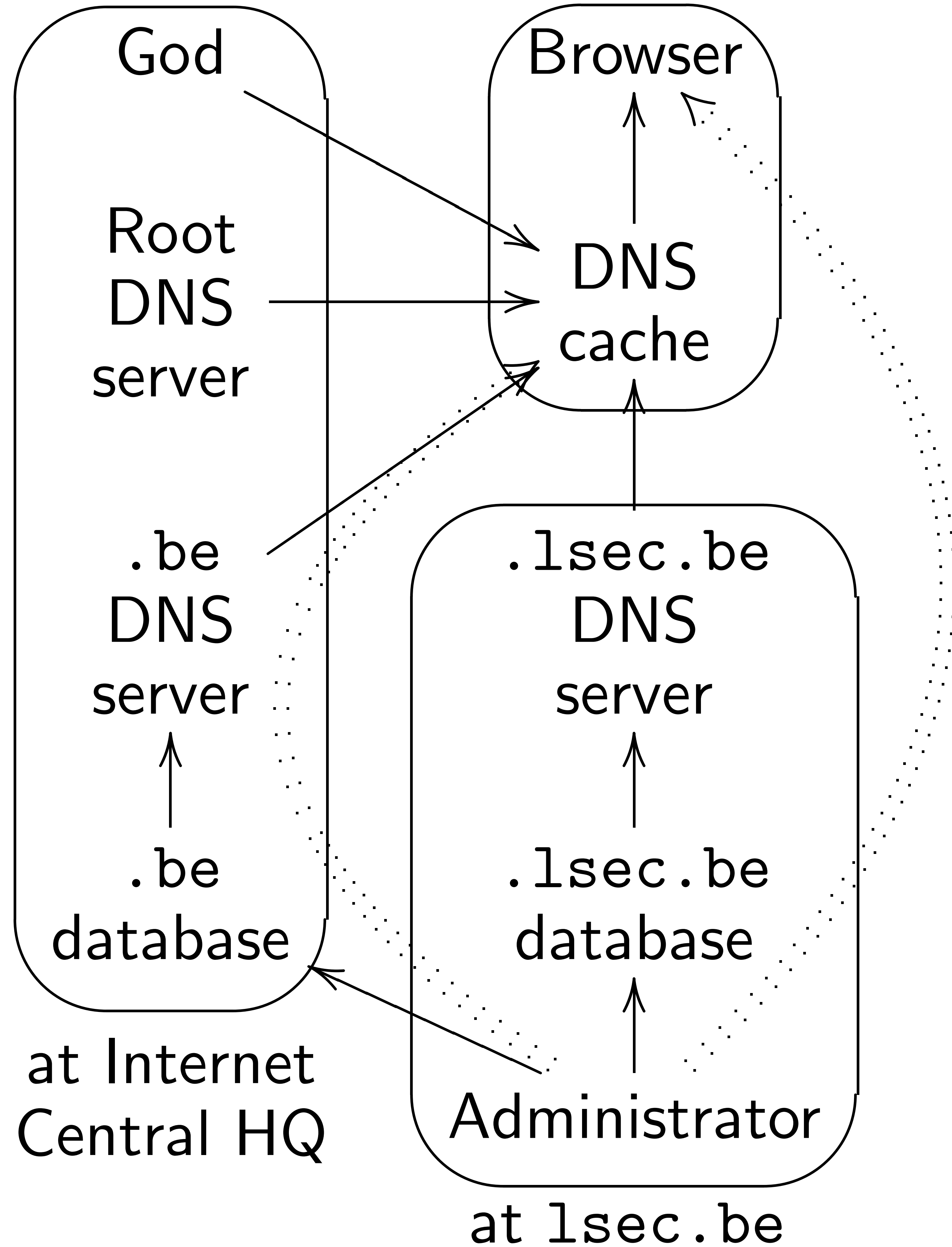
ckets to/from DNS cache:

ayeth unto the DNS cache:
Root `K.Heaven` 193.0.14.129”

“Web `www.lsec.be`?”
0.14.129 $\longleftarrow$ $\longrightarrow$ DNS cache
`.be brussels` 193.190.135.4”

“Web `www.lsec.be`?”
.90.135.4 $\longleftarrow$ $\longrightarrow$ DNS cache
S `.lsec.be ns2` 80.92.67.140”

“Web `www.lsec.be`?”
2.67.140 $\longleftarrow$ $\longrightarrow$ DNS cache
b `www.lsec.be` 81.246.94.54”



This ar
the `www`
is cont
by the
by the
by the

This is
`lsec.b`

e.g. 20
An atta
Interne
into ac
for `mic`

om DNS cache:

o the DNS cache:

aven 193.0.14.129"

.lsec.be?"

DNS cache

els 193.190.135.4"

.lsec.be?"

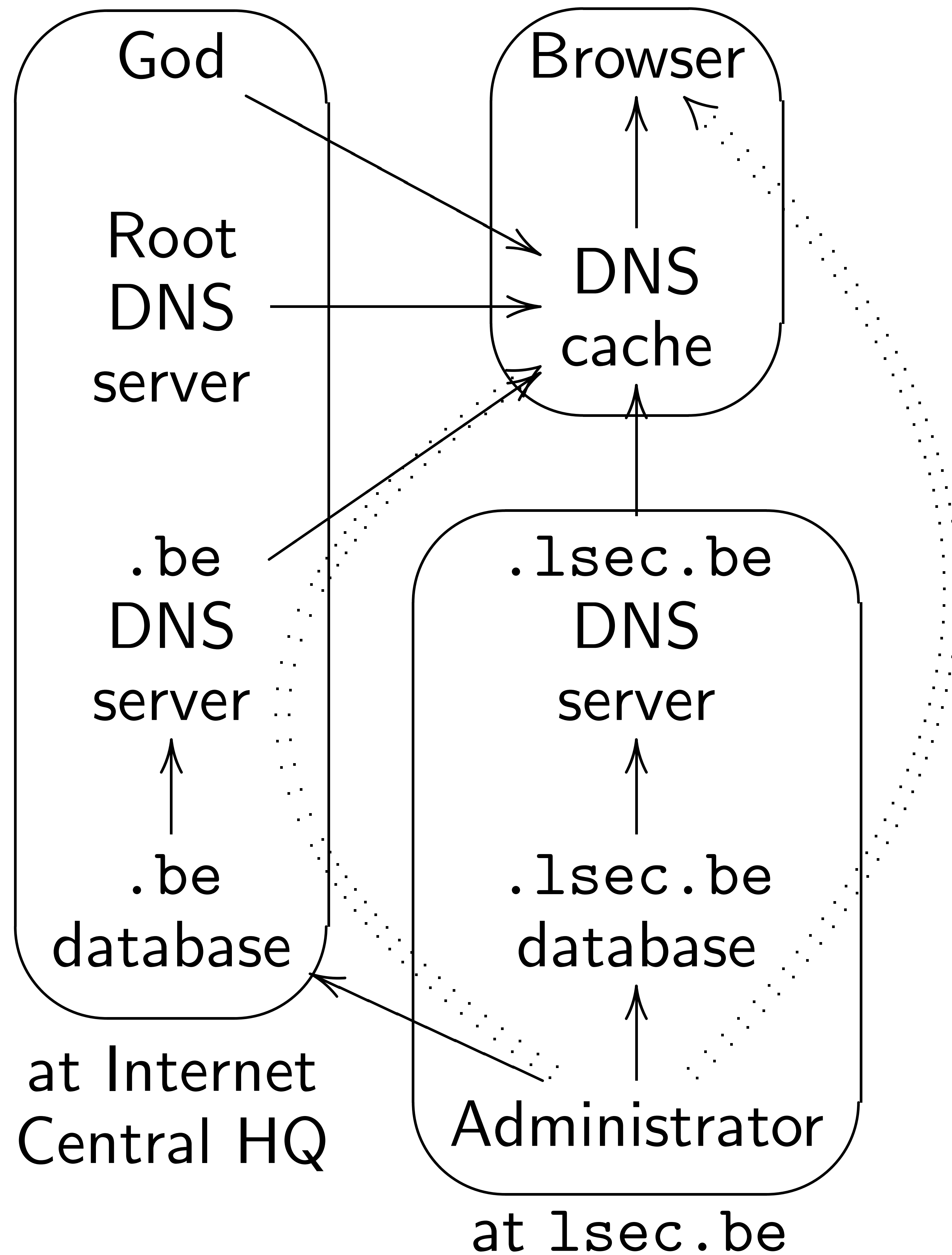
DNS cache

ns2 80.92.67.140"

.lsec.be?"

DNS cache

.be 81.246.94.54"



This architecture

the www.lsec.b

is controlled

by the DNS root

by the .be DNS

by the lsec.be

This isn't just th

lsec.be DNS se

e.g. 2001 incider

An attacker foole

Internet Central

into accepting fa

for microsoft.c

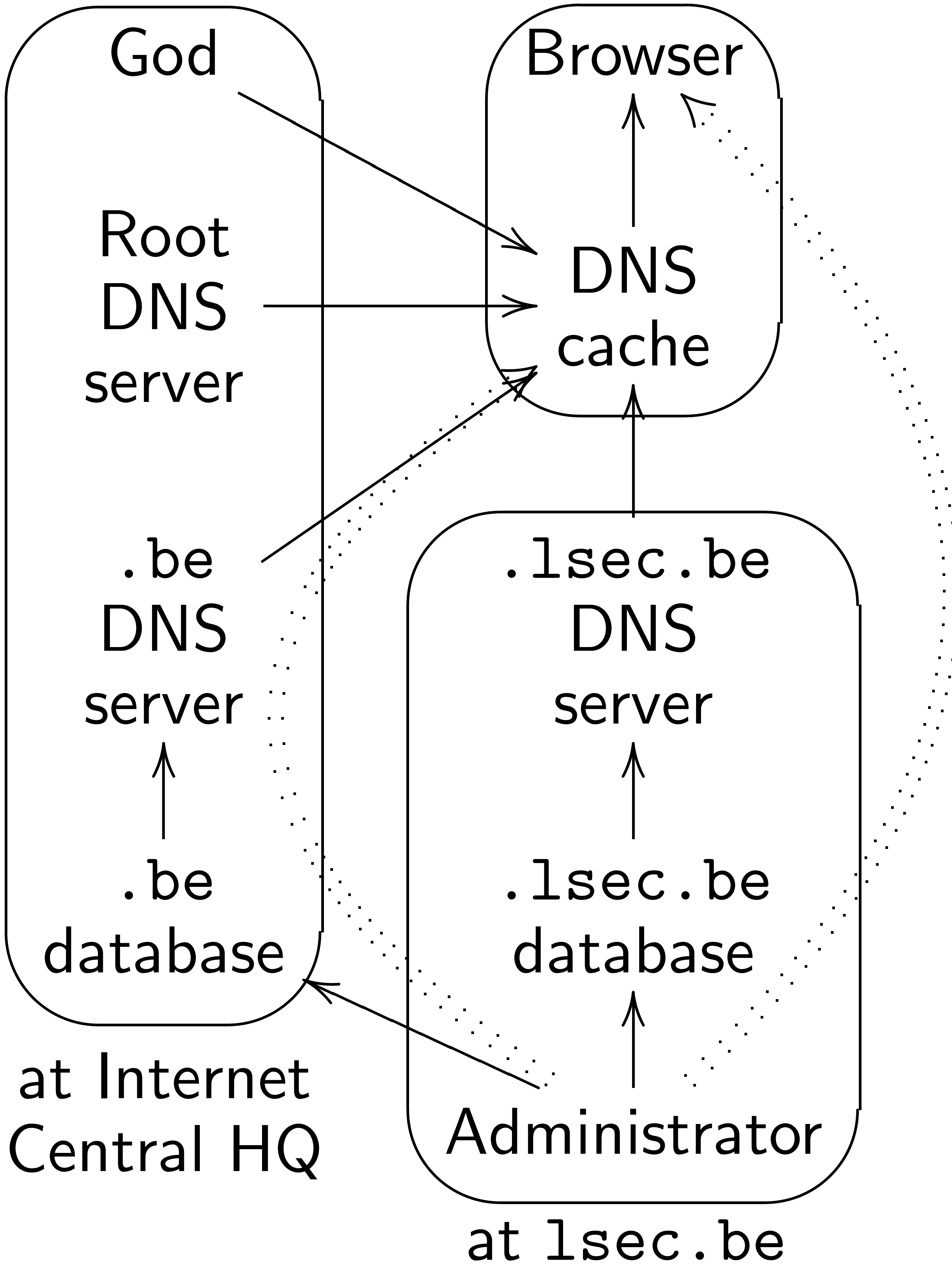
cache:

cache:
14.129"

cache
.135.4"

cache
7.140"

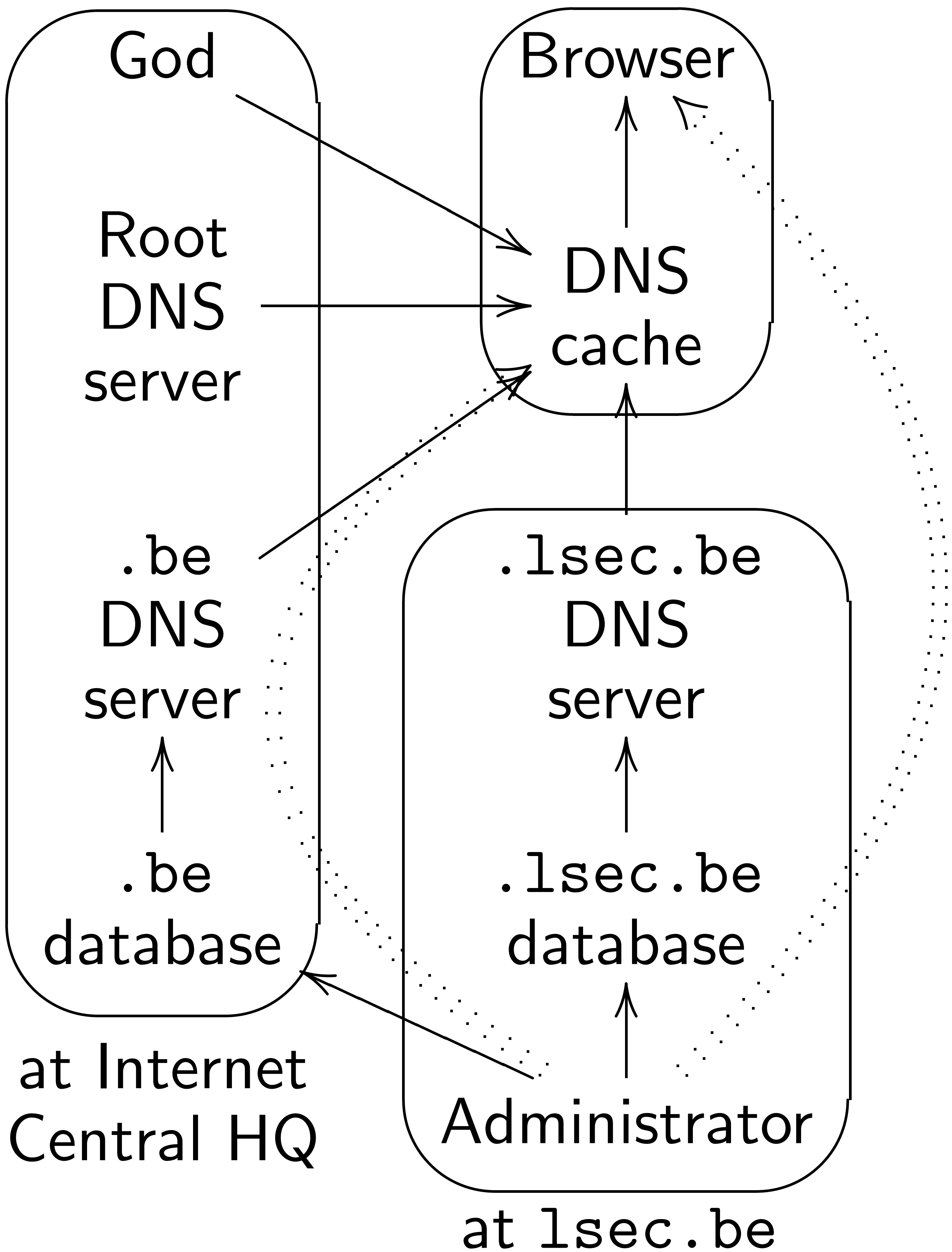
cache
94.54"



This architecture means th
the `www.lsec.be` address
is controlled
by the DNS root server;
by the `.be` DNS server; an
by the `lsec.be` DNS serve

This isn't just the
`lsec.be` DNS server!

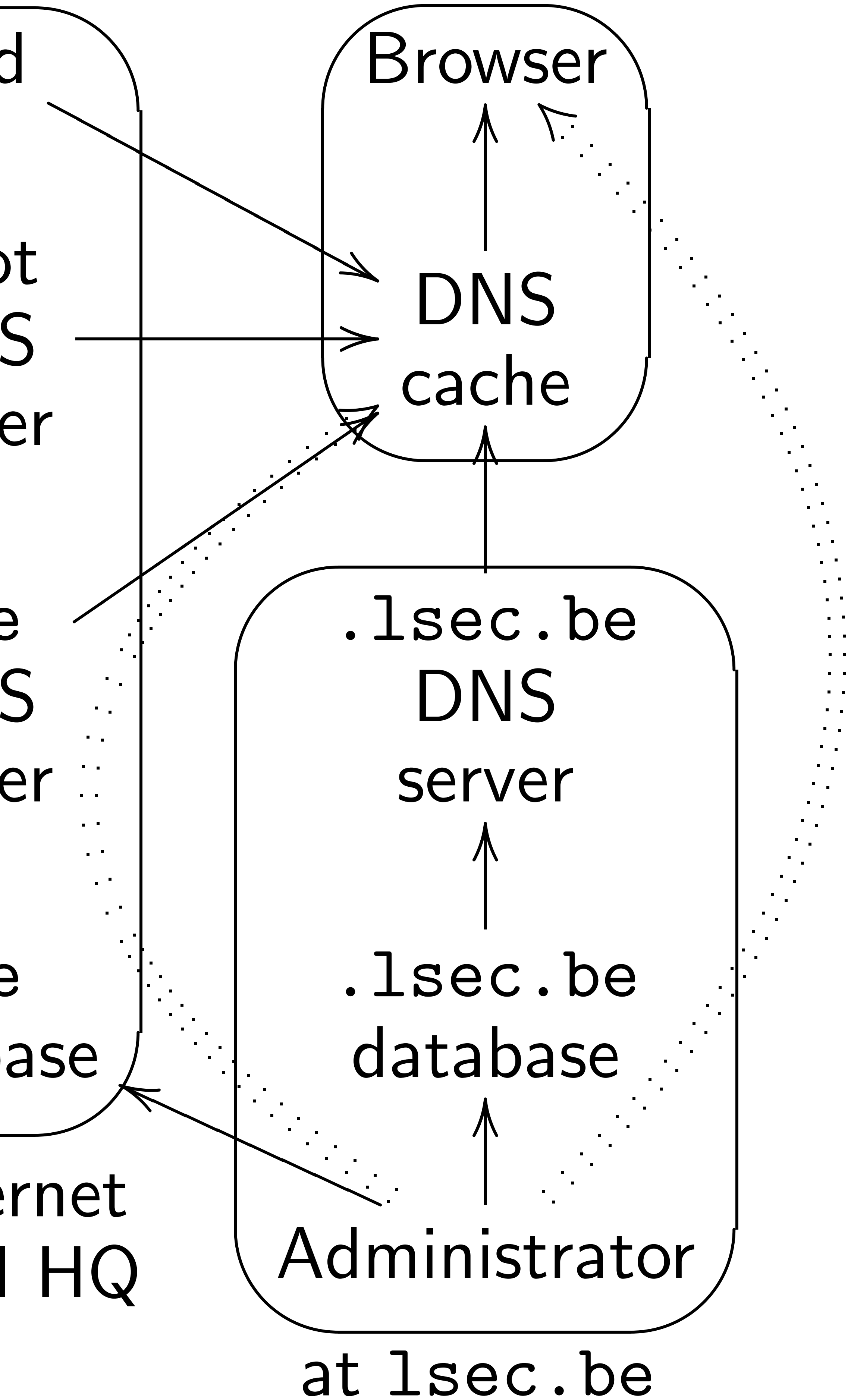
e.g. 2001 incident:
An attacker fooled
Internet Central Headquart
into accepting fake data
for `microsoft.com`.



This architecture means that the `www.lsec.be` address is controlled by the DNS root server; by the `.be` DNS server; and by the `lsec.be` DNS server.

This isn't just the `lsec.be` DNS server!

e.g. 2001 incident: An attacker fooled Internet Central Headquarters into accepting fake data for `microsoft.com`.



This architecture means that
the `www.lsec.be` address
is controlled
by the DNS root server;
by the `.be` DNS server; and
by the `lsec.be` DNS server.

This isn't just the
`lsec.be` DNS server!

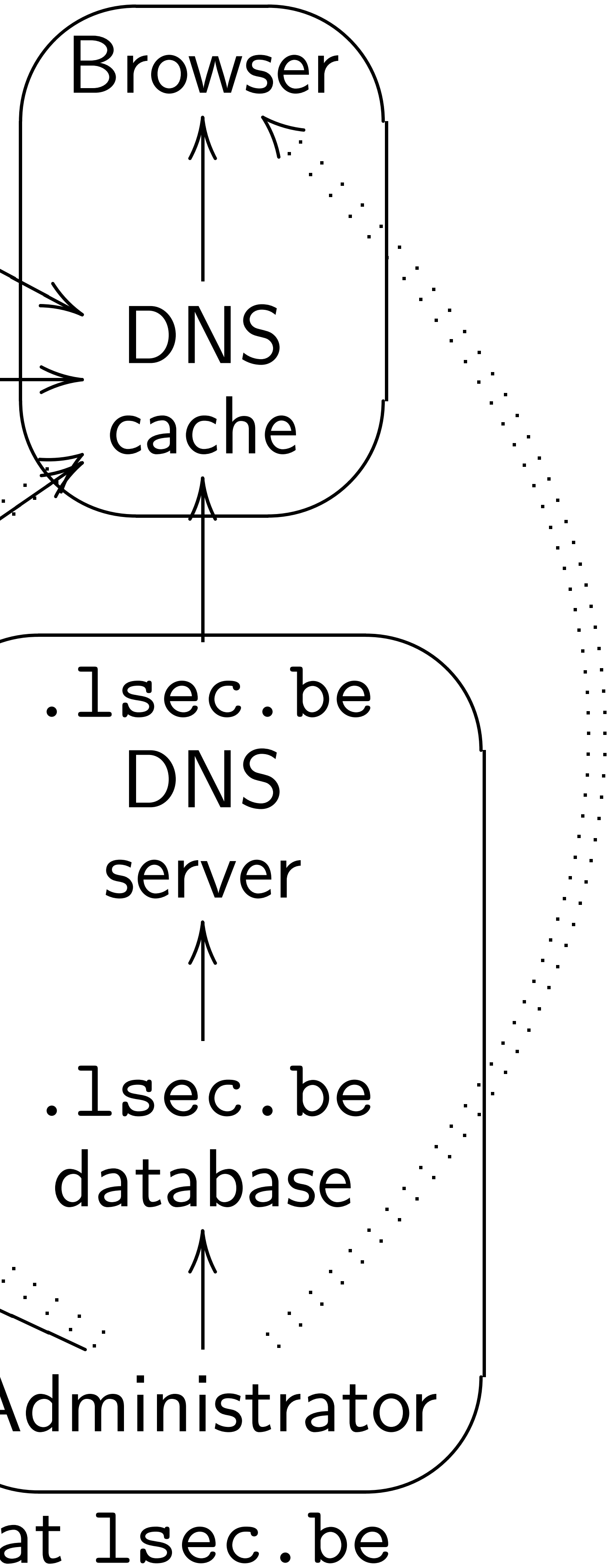
e.g. 2001 incident:
An attacker fooled
Internet Central Headquarters
into accepting fake data
for `microsoft.com`.

But wa
Recall
for `lse`

a

"The D
for `.l`
is
with IP
80.92.

a

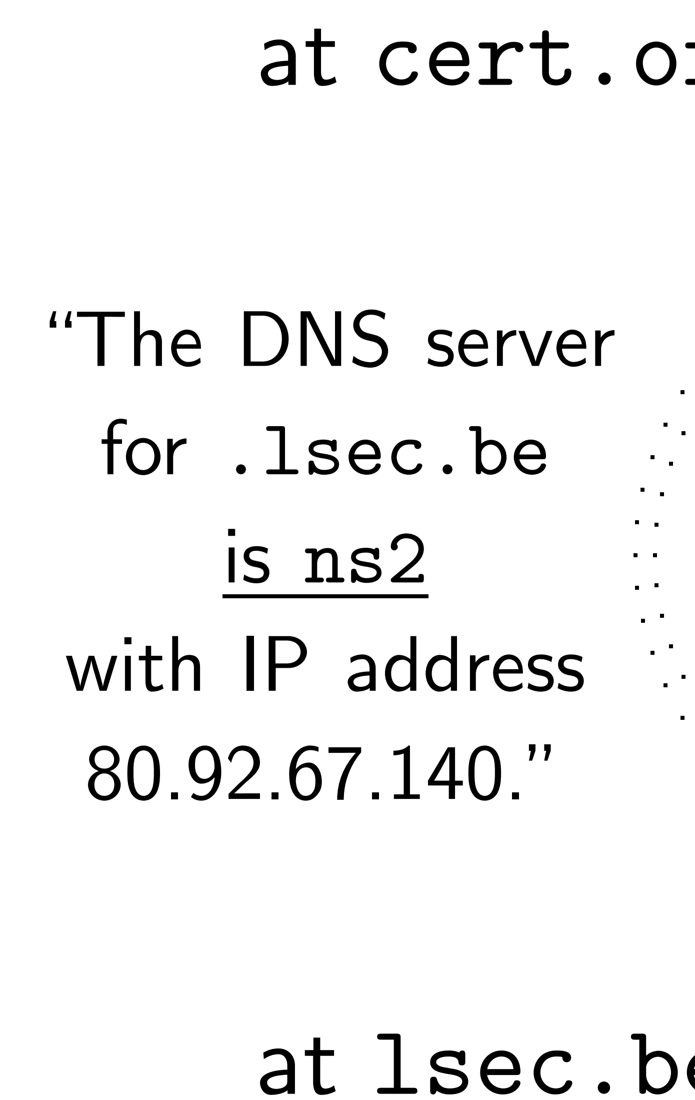


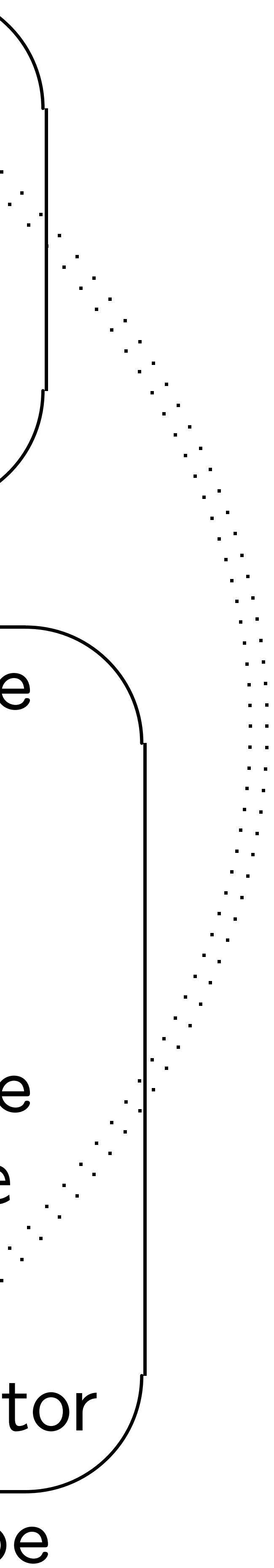
This architecture means that the `www.lsec.be` address is controlled
by the DNS root server;
by the `.be` DNS server; and
by the `lsec.be` DNS server.

This isn't just the `lsec.be` DNS server!

e.g. 2001 incident:
An attacker fooled Internet Central Headquarters into accepting fake data for `microsoft.com`.

But wait, there's
Recall that the D
for `lsec.be` hav



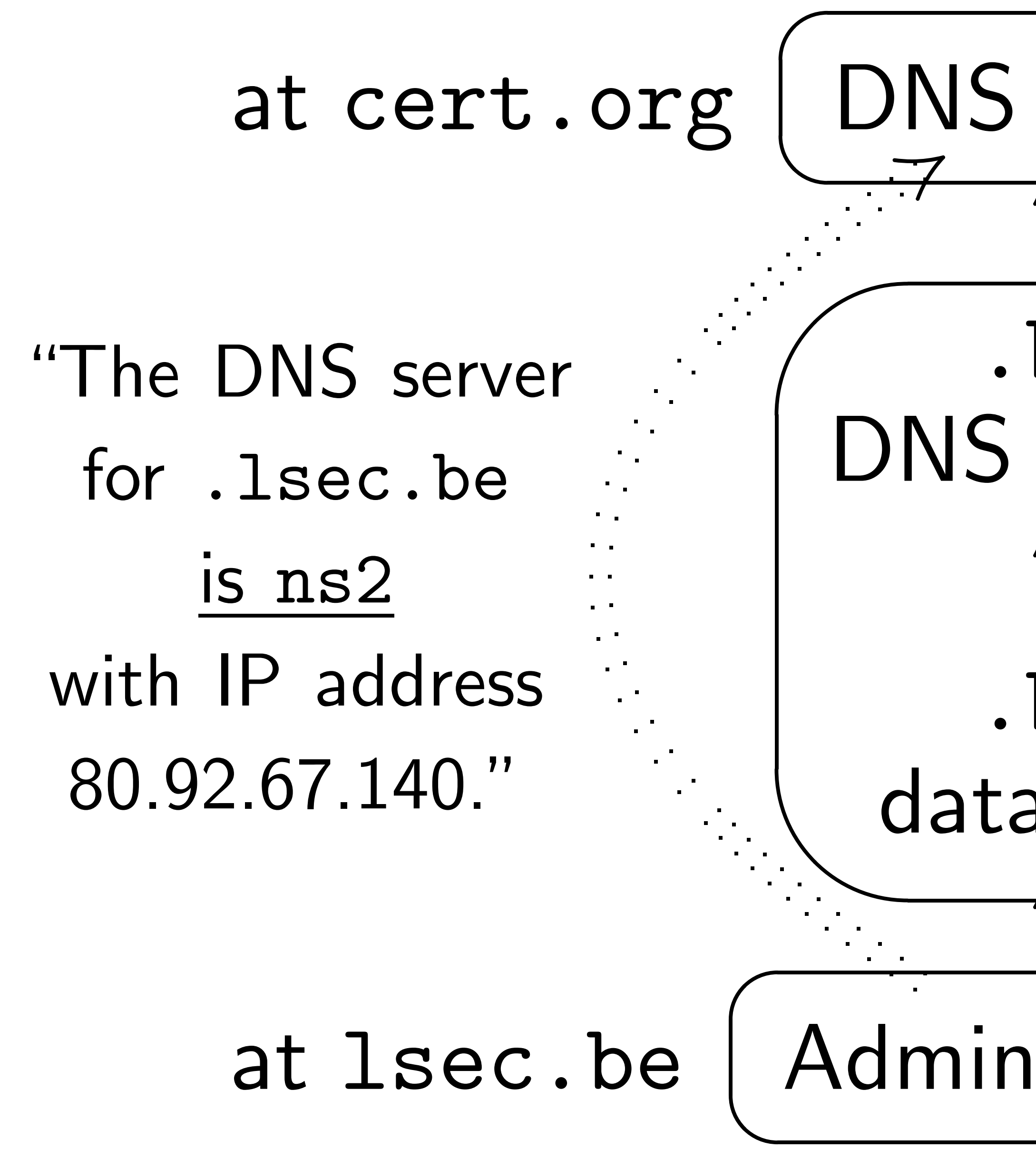


This architecture means that the `www.lsec.be` address is controlled
by the DNS root server;
by the `.be` DNS server; and
by the `lsec.be` DNS server.

This isn't just the `lsec.be` DNS server!

e.g. 2001 incident:
An attacker fooled Internet Central Headquarters into accepting fake data for `microsoft.com`.

But wait, there's more!
Recall that the DNS server for `lsec.be` have names.

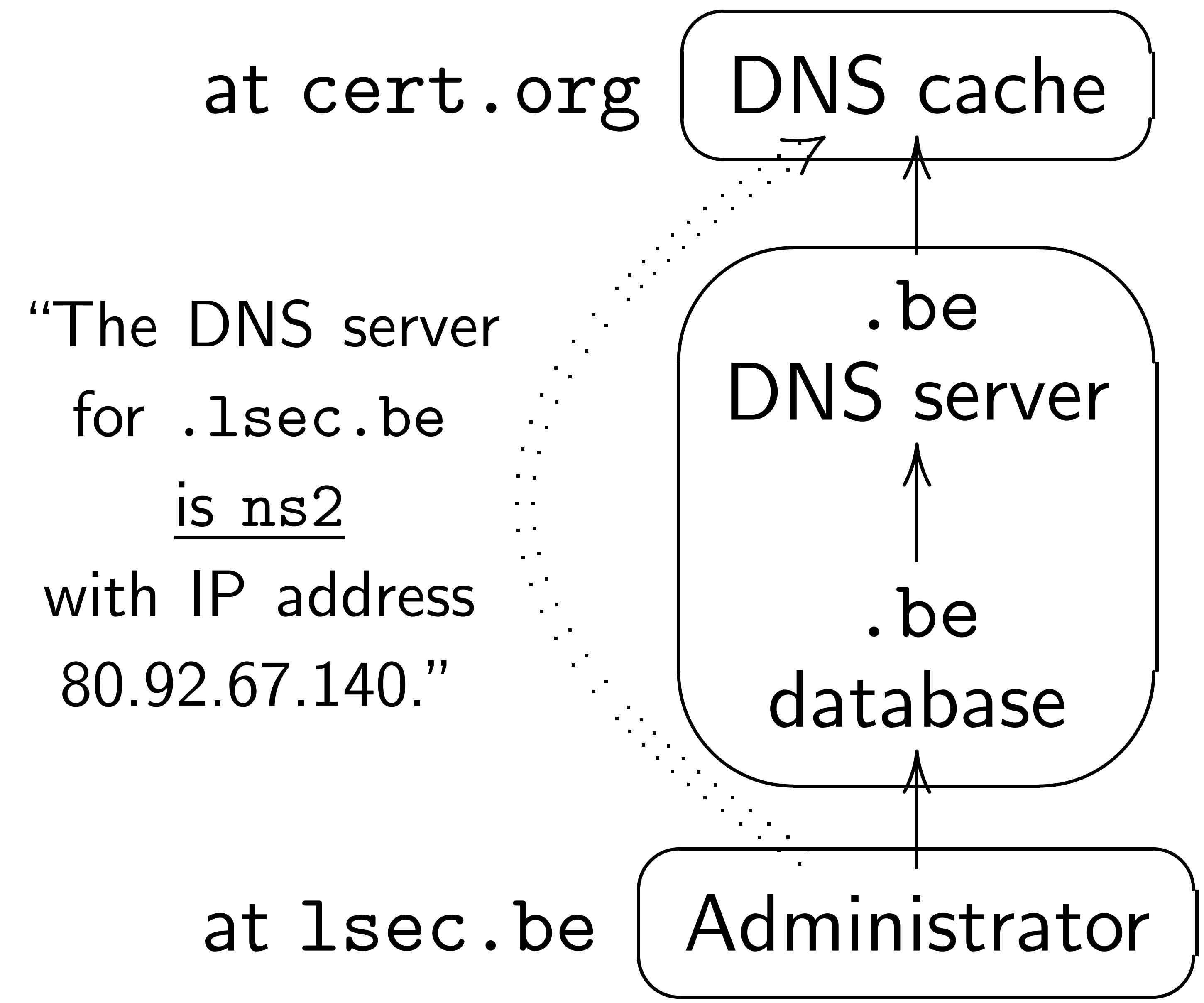


This architecture means that the `www.lsec.be` address is controlled
by the DNS root server;
by the `.be` DNS server; and
by the `lsec.be` DNS server.

This isn't just the `lsec.be` DNS server!

e.g. 2001 incident:
An attacker fooled Internet Central Headquarters into accepting fake data for `microsoft.com`.

But wait, there's more!
Recall that the DNS servers for `lsec.be` have names.

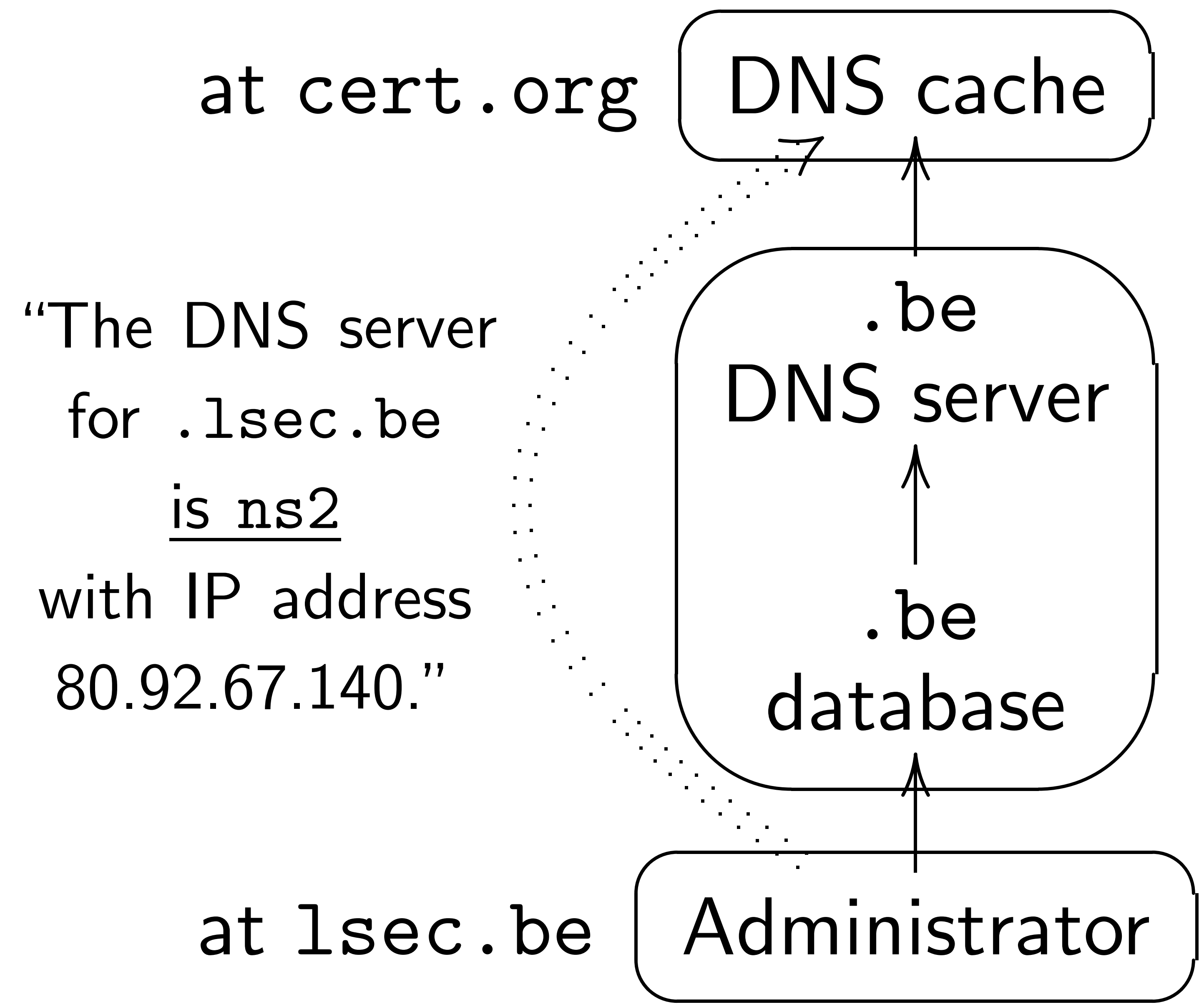


rchitecture means that
`w.lsec.be` address
rolled
DNS root server;
`.be` DNS server; and
`lsec.be` DNS server.

n't just the
be DNS server!

001 incident:
acker fooled
t Central Headquarters
cepting fake data
`crosoft.com`.

But wait, there's more!
Recall that the DNS servers
for `lsec.be` have names.



These
outside

One of
for `w3.`
`w3csur`

One of
for `ac.`
`ns.eu.`

One of
for `eu.`
`sunic.`

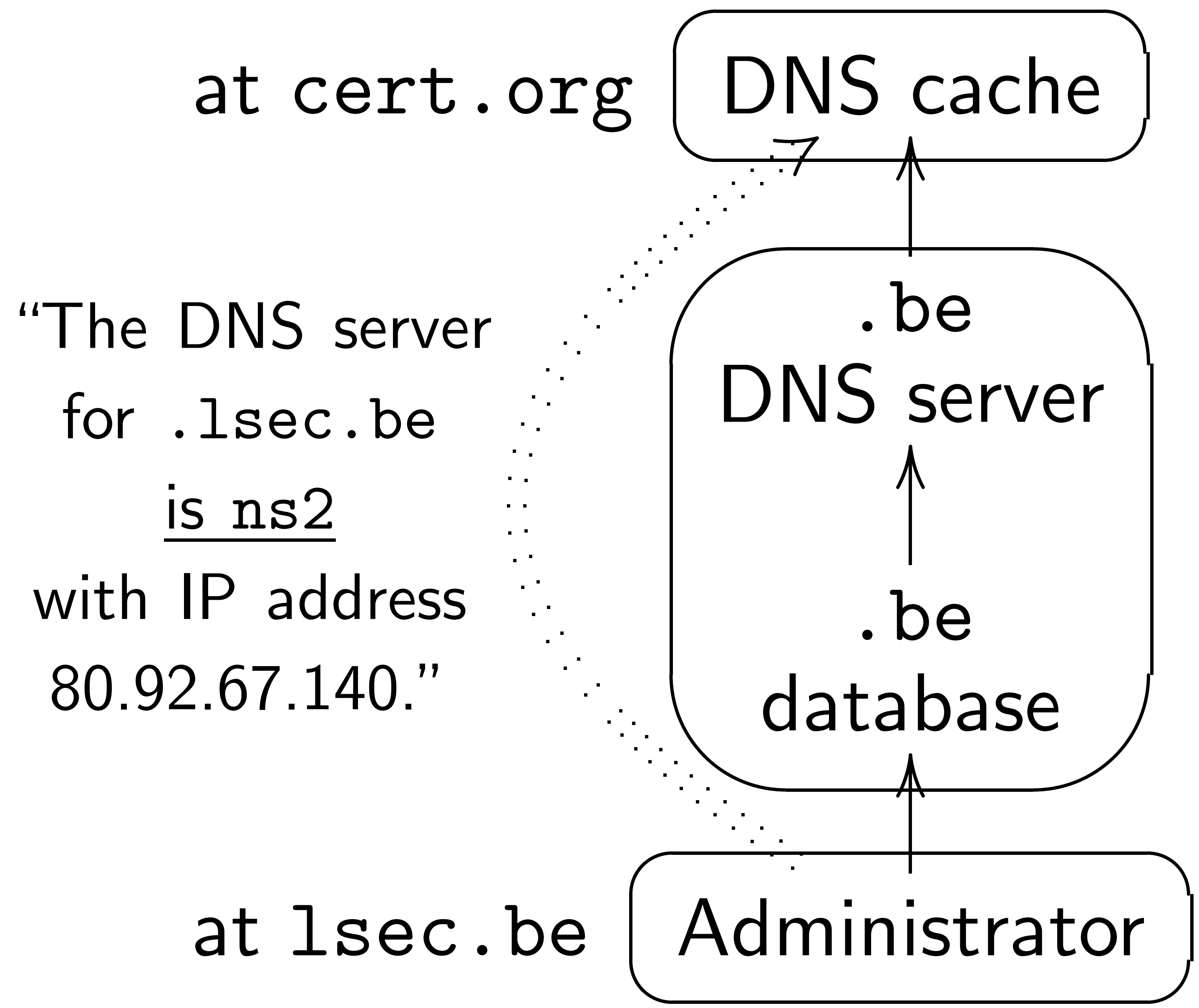
e means that
e address

server;
server; and
DNS server.

e
rver!

nt:
ed
Headquarters
ke data
com.

But wait, there's more!
Recall that the DNS servers
for `lsec.be` have names.



These names car
outside `lsec.be`

One of the DNS
for `w3.org` is nar
`w3csun1.cis.rl`

One of the DNS
for `ac.uk` is nam
`ns.eu.net`.

One of the DNS
for `eu.net` is nar
`sunic.sunet.se`

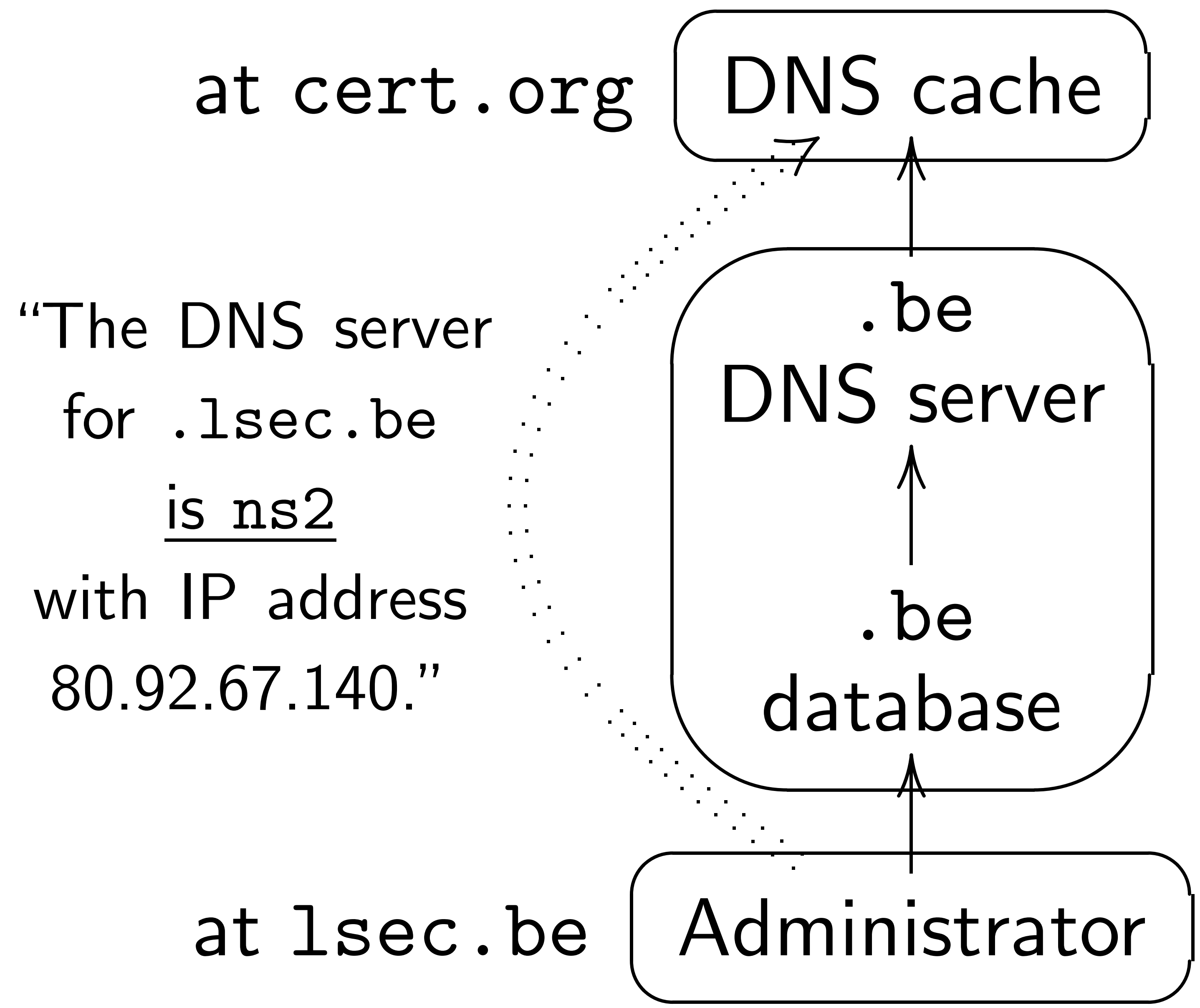
at

d

er.

ters

But wait, there's more!
Recall that the DNS servers
for `lsec.be` have names.



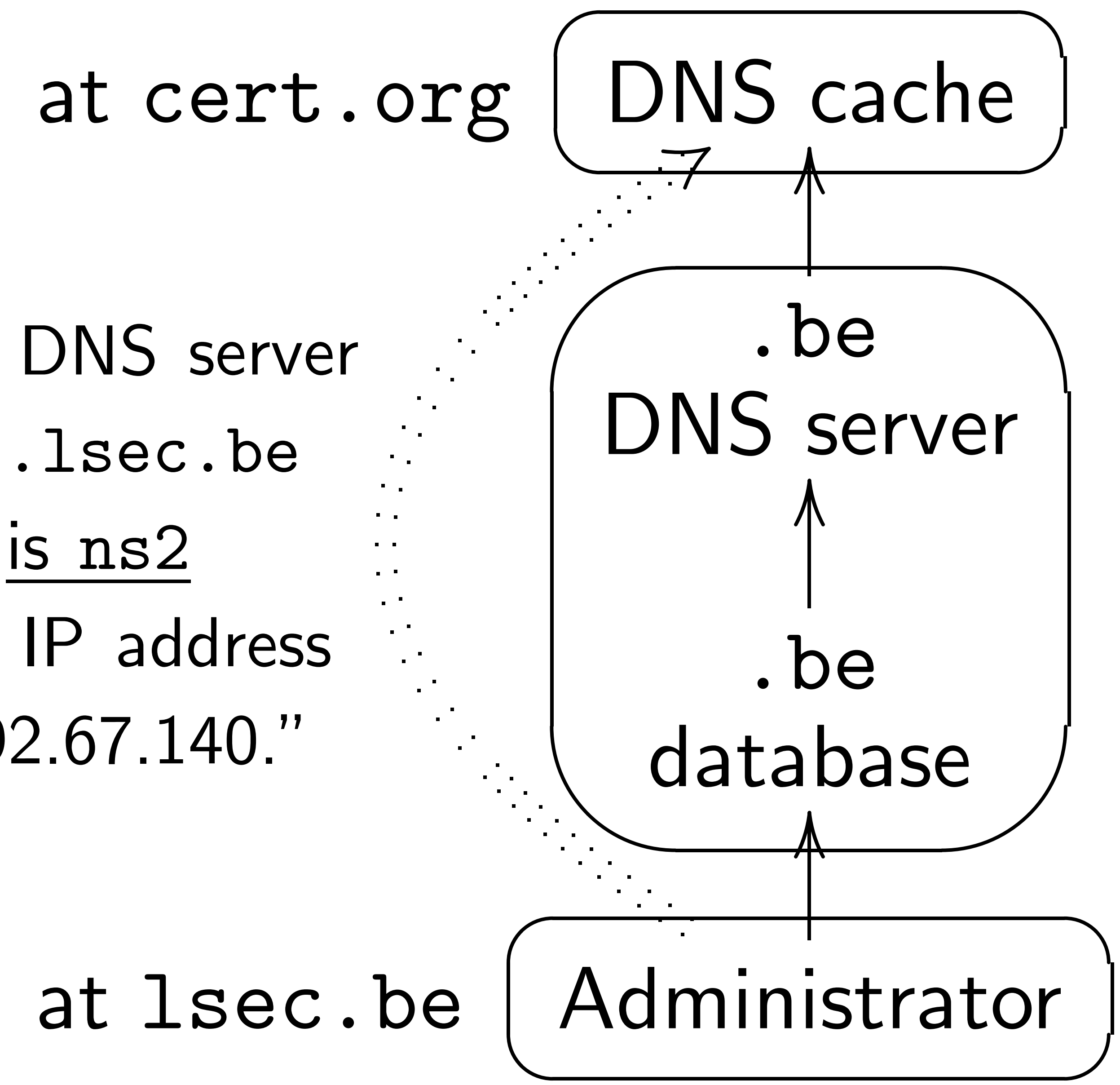
These names can be outside `lsec.be`.

One of the DNS servers for `w3.org` is named `w3csun1.cis.rl.ac.uk`.

One of the DNS servers for `ac.uk` is named `ns.eu.net`.

One of the DNS servers for `eu.net` is named `sunic.sunet.se`.

But wait, there's more!
Recall that the DNS servers
for `lsec.be` have names.



These names can be
outside `lsec.be`.

One of the DNS servers
for `w3.org` is named
`w3csun1.cis.rl.ac.uk`.

One of the DNS servers
for `ac.uk` is named
`ns.eu.net`.

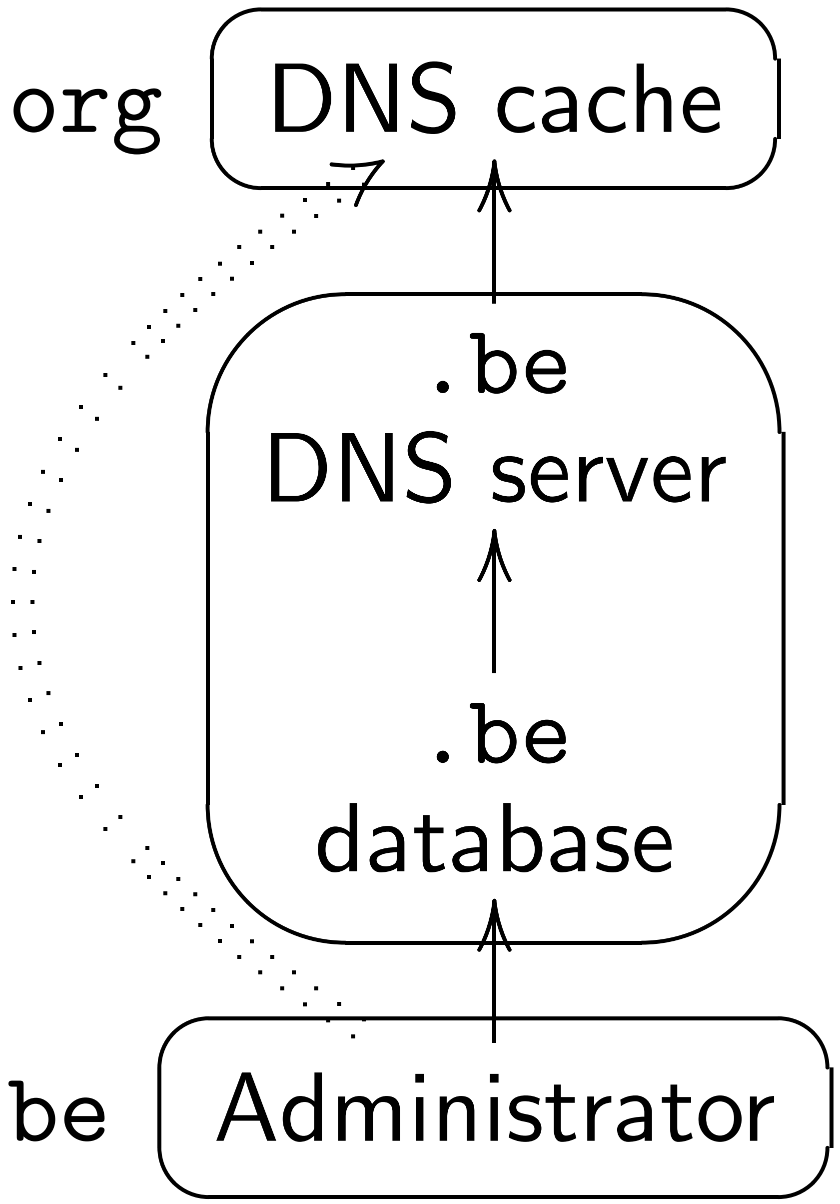
One of the DNS servers
for `eu.net` is named
`sunic.sunet.se`.

nit, there's more!
that the DNS servers
ec.be have names.

at cert.org

DNS server
.sec.be
ns2
P address
67.140."

at lsec.be



These names can be outside lsec.be.

One of the DNS servers for w3.org is named w3csun1.cis.rl.ac.uk.

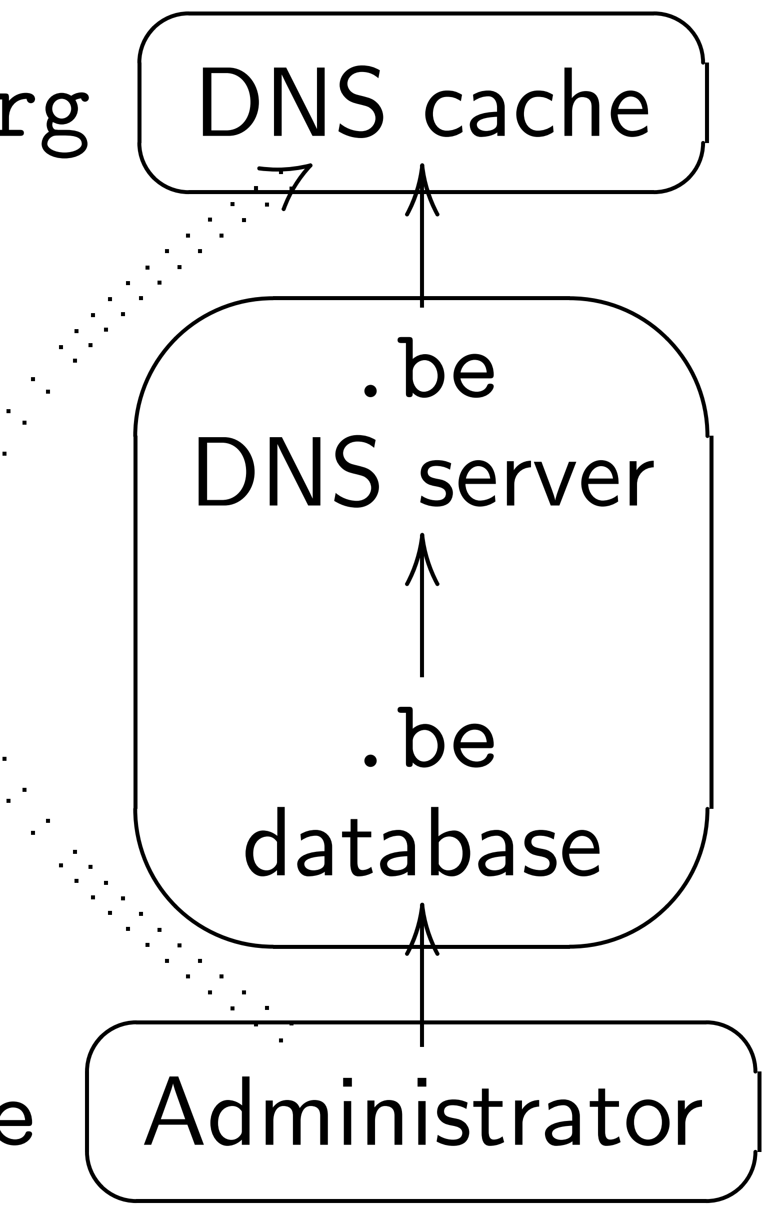
One of the DNS servers for ac.uk is named ns.eu.net.

One of the DNS servers for eu.net is named sunic.sunet.se.

One of
for sun
beer.p
and is

Attacke
beer.p
tells DI
for sun
tells DI
for ns.
tells DI
for w3c
tells DI
for w3.

more!
NS servers
e names.



These names can be outside `lsec.be`.

One of the DNS servers for `w3.org` is named `w3csun1.cis.rl.ac.uk`.

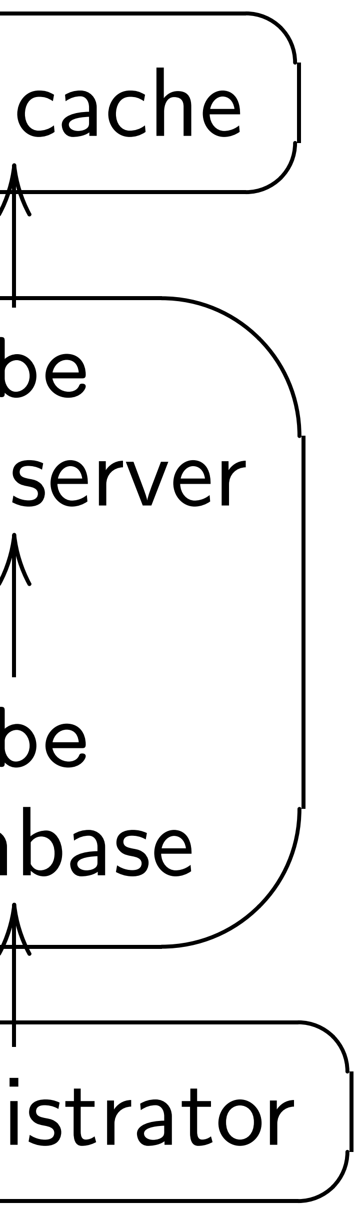
One of the DNS servers for `ac.uk` is named `ns.eu.net`.

One of the DNS servers for `eu.net` is named `sunic.sunet.se`.

One of the DNS
for `sunet.se` is
`beer.pilsnet.s`
and is horribly in

Attacker takes co
`beer.pilsnet.s`
tells DNS cache
for `sunic.sunet`
tells DNS cache
for `ns.eu.net`;
tells DNS cache
for `w3csun1.cis`
tells DNS cache
for `w3.org`.

rs



These names can be outside `lsec.be`.

One of the DNS servers for `w3.org` is named `w3csun1.cis.rl.ac.uk`.

One of the DNS servers for `ac.uk` is named `ns.eu.net`.

One of the DNS servers for `eu.net` is named `sunic.sunet.se`.

One of the DNS servers for `sunet.se` is named `beer.pilsnet.sunet.se` and is horribly insecure.

Attacker takes control of `beer.pilsnet.sunet.se`;
tells DNS cache a fake ad
for `sunic.sunet.se`;
tells DNS cache a fake ad
for `ns.eu.net`;
tells DNS cache a fake ad
for `w3csun1.cis.rl.ac.u`
tells DNS cache a fake ad
for `w3.org`.

These names can be outside `lsec.be`.

One of the DNS servers for `w3.org` is named `w3csun1.cis.rl.ac.uk`.

One of the DNS servers for `ac.uk` is named `ns.eu.net`.

One of the DNS servers for `eu.net` is named `sunic.sunet.se`.

One of the DNS servers for `sunet.se` is named `beer.pilsnet.sunet.se` and is horribly insecure.

Attacker takes control of `beer.pilsnet.sunet.se`;
tells DNS cache a fake address for `sunic.sunet.se`;
tells DNS cache a fake address for `ns.eu.net`;
tells DNS cache a fake address for `w3csun1.cis.rl.ac.uk`;
tells DNS cache a fake address for `w3.org`.

names can be
e lsec.be.

f the DNS servers
org is named
n1.cis.rl.ac.uk.

f the DNS servers
uk is named
.net.

f the DNS servers
net is named
.sunet.se.

One of the DNS servers for sunet.se is named beer.pilsnet.sunet.se and is horribly insecure.

Attacker takes control of beer.pilsnet.sunet.se;
tells DNS cache a fake address for sunic.sunet.se;
tells DNS cache a fake address for ns.eu.net;
tells DNS cache a fake address for w3csun1.cis.rl.ac.uk;
tells DNS cache a fake address for w3.org.

2000 B
control
via serv
Many c
run old

Lesson
Don't u
names

.com w
Eventu
this exa

2006 R
"Perils
Problem

n be
.

servers
med
l.ac.uk.

servers
ned

servers
med
e.

One of the DNS servers
for `sunet.se` is named
`beer.pilsnet.sunet.se`
and is horribly insecure.

Attacker takes control of
`beer.pilsnet.sunet.se`;
tells DNS cache a fake address
for `sunic.sunet.se`;
tells DNS cache a fake address
for `ns.eu.net`;
tells DNS cache a fake address
for `w3csun1.cis.rl.ac.uk`;
tells DNS cache a fake address
for `w3.org`.

2000 Bernstein:
controlled by $>$ 2
via server-name s
Many of these co
run old breakable

Lesson to admini
Don't use out-of-
names for DNS s

`.com` was then fi
Eventually `w3.or`
this example no l

2006 Ramasubra
"Perils of transiti
Problem is still w

One of the DNS servers
for `sunet.se` is named
`beer.pilsnet.sunet.se`
and is horribly insecure.

Attacker takes control of
`beer.pilsnet.sunet.se`;
tells DNS cache a fake address
for `sunic.sunet.se`;
tells DNS cache a fake address
for `ns.eu.net`;
tells DNS cache a fake address
for `w3csun1.cis.rl.ac.uk`;
tells DNS cache a fake address
for `w3.org`.

2000 Bernstein: `.com` etc.
controlled by >200 compu
via server-name server trus
Many of these computers
run old breakable servers.

Lesson to administrators:
Don't use out-of-bailiwick
names for DNS servers.

`.com` was then fixed.
Eventually `w3.org` was fixe
this example no longer wor

2006 Ramasubramanian–Si
"Perils of transitive trust":
Problem is still widespread

One of the DNS servers for `sunet.se` is named `beer.pilsnet.sunet.se` and is horribly insecure.

Attacker takes control of `beer.pilsnet.sunet.se`; tells DNS cache a fake address for `sunic.sunet.se`; tells DNS cache a fake address for `ns.eu.net`; tells DNS cache a fake address for `w3csun1.cis.rl.ac.uk`; tells DNS cache a fake address for `w3.org`.

2000 Bernstein: `.com` etc. are controlled by > 200 computers via server-name server trust. Many of these computers run old breakable servers.

Lesson to administrators: Don't use out-of-bailiwick names for DNS servers.

`.com` was then fixed. Eventually `w3.org` was fixed; this example no longer works.

2006 Ramasubramanian–Sirer "Perils of transitive trust": Problem is still widespread.

the DNS servers
et.se is named
ilsnet.sunet.se
horribly insecure.

er takes control of
ilsnet.sunet.se;
NS cache a fake address
ic.sunet.se;
NS cache a fake address
eu.net;
NS cache a fake address
sun1.cis.rl.ac.uk;
NS cache a fake address
org.

2000 Bernstein: `.com` etc. are controlled by $>$ 200 computers via server-name server trust. Many of these computers run old breakable servers.

Lesson to administrators: Don't use out-of-bailiwick names for DNS servers.

`.com` was then fixed. Eventually `w3.org` was fixed; this example no longer works.

2006 Ramasubramanian–Sirer "Perils of transitive trust": Problem is still widespread.

What's

"Can w
elimina
— Sec
Crypto

"What
and otl
— Thi
design

servers
named
sunet.se
secure.

ontrol of
sunet.se;
a fake address
t.se;
a fake address

a fake address
s.rl.ac.uk;
a fake address

2000 Bernstein: `.com` etc. are controlled by $>$ 200 computers via server-name server trust. Many of these computers run old breakable servers.

Lesson to administrators: Don't use out-of-bailiwick names for DNS servers.

`.com` was then fixed. Eventually `w3.org` was fixed; this example no longer works.

2006 Ramasubramanian–Sirer "Perils of transitive trust": Problem is still widespread.

What's coming u

"Can we detect a
eliminate forged
— Second talk:
Cryptography in

"What about bu
and other softwa
— Third talk: Se
design and codin

;
dress

dress

dress
uk;
dress

2000 Bernstein: `.com` etc. are controlled by $>$ 200 computers via server-name server trust. Many of these computers run old breakable servers.

Lesson to administrators: Don't use out-of-bailiwick names for DNS servers.

`.com` was then fixed. Eventually `w3.org` was fixed; this example no longer works.

2006 Ramasubramanian–Sirer "Perils of transitive trust": Problem is still widespread.

What's coming up

"Can we detect and eliminate forged packets?" — Second talk: Cryptography in DNS.

"What about buffer overflo and other software problem — Third talk: Secure design and coding for DNS

2000 Bernstein: `.com` etc. are controlled by >200 computers via server-name server trust. Many of these computers run old breakable servers.

Lesson to administrators: Don't use out-of-bailiwick names for DNS servers.

`.com` was then fixed. Eventually `w3.org` was fixed; this example no longer works.

2006 Ramasubramanian–Sirer "Perils of transitive trust": Problem is still widespread.

What's coming up

"Can we detect and eliminate forged packets?"
— Second talk: Cryptography in DNS.

"What about buffer overflows and other software problems?"
— Third talk: Secure design and coding for DNS.