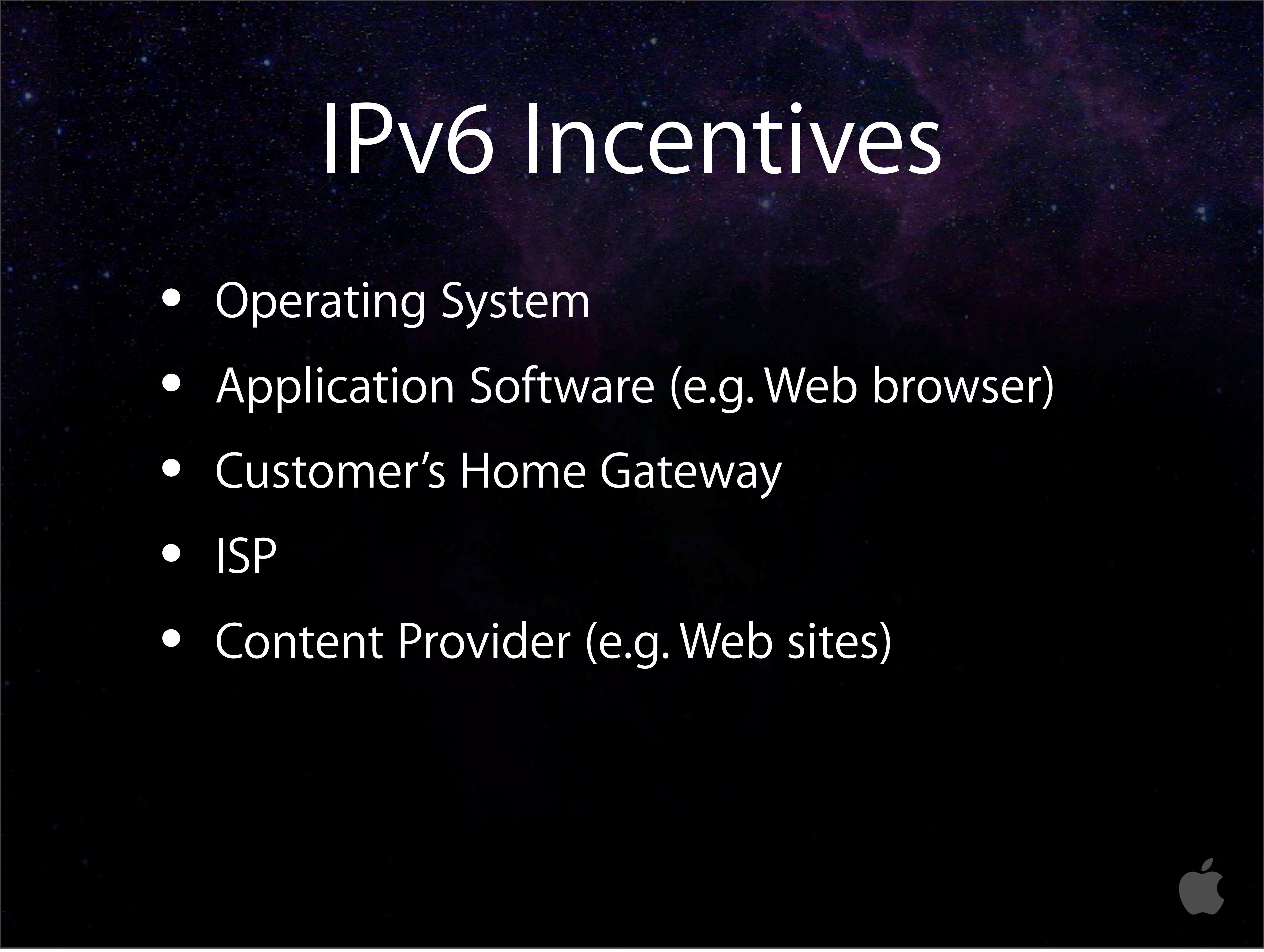
Connect-By-Name for IPv6

Stuart Cheshire, Apple

79th IETF, 8th November 2010, Beijing

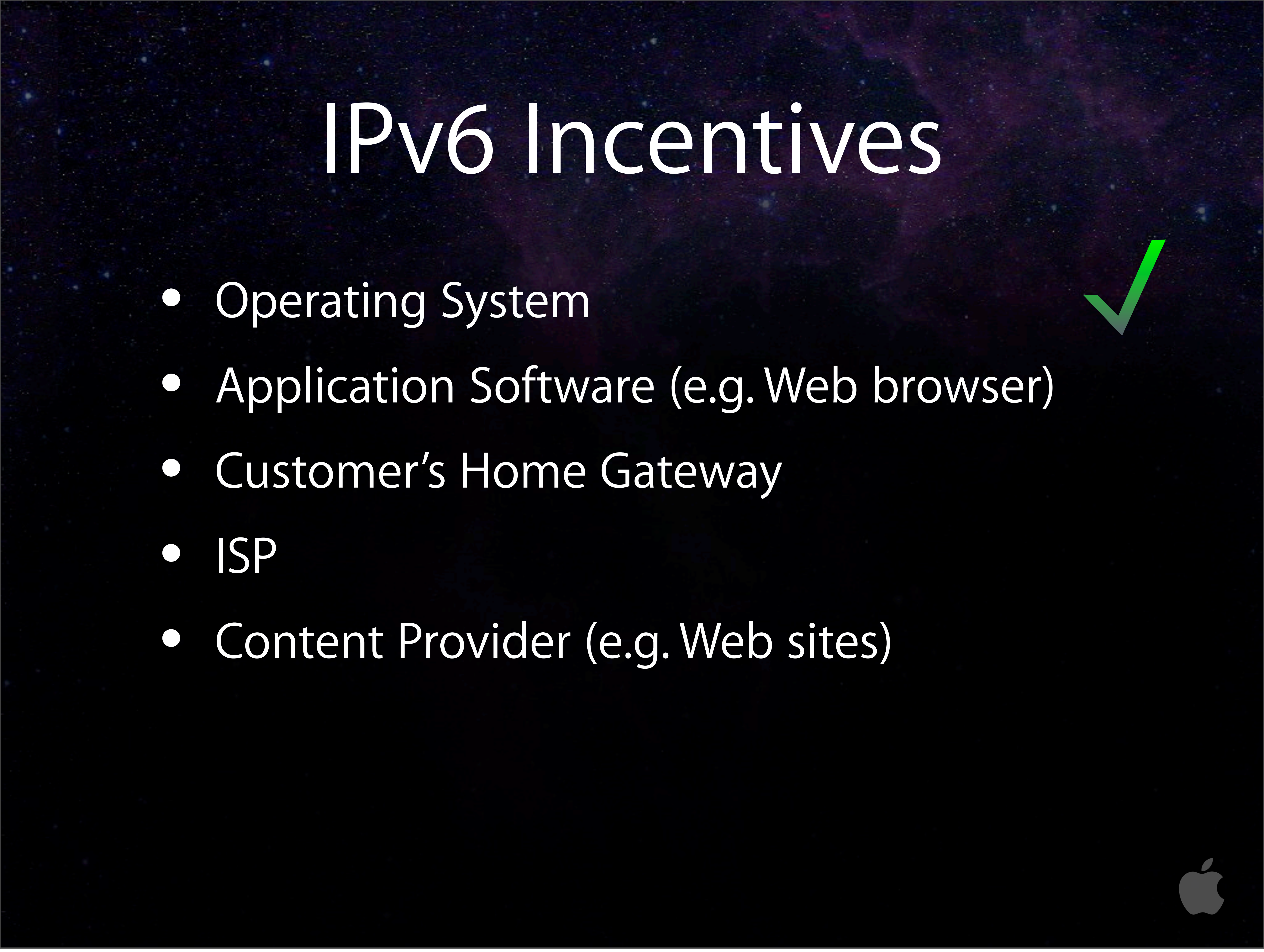
Adapted from IETF 72 IAB plenary presentation, available with audio soundtrack at
http://www.stuartcheshire.org/IETF72/



IPv6 Incentives

- Operating System
- Application Software (e.g. Web browser)
- Customer’s Home Gateway
- ISP
- Content Provider (e.g. Web sites)

Apple’s business is largely end-users at home using iMacs.
In Enterprise markets this picture might look a bit different, but in this context we have five players who need to cooperate to make IPv6 a success.
So, in reverse order:
* We need content, and I’ll use the web as the common example of what people use the Internet for. We need web sites that have content, that are reachable by IPv6
* but there’s no point having that web site if no ISP offers IPv6 to customers
* and there’s no point an ISP offering it if the customer’s home gateway doesn’t support IPv6,
* and there’s no point doing that if the web browser doesn’t do IPv6
* and you can’t do a web browser that supports IPv6 unless the operating system does
so everybody needs to collaborate here
Now, there’s no incentive for any of these players to move first, because any one of these players does all this work, and the user gets no benefit because the other people haven’t done their bit.
So, the problem we have is providing incentives.
Well, IPv6 is cool, and thankfully, in many cases, that’s enough
↳and that’s why Apple has IPv6 in the operating system
↳and that’s why most of our network applications already support IPv6 too
but now we need the other players to catch up
and we need incentives for them to move to IPv6

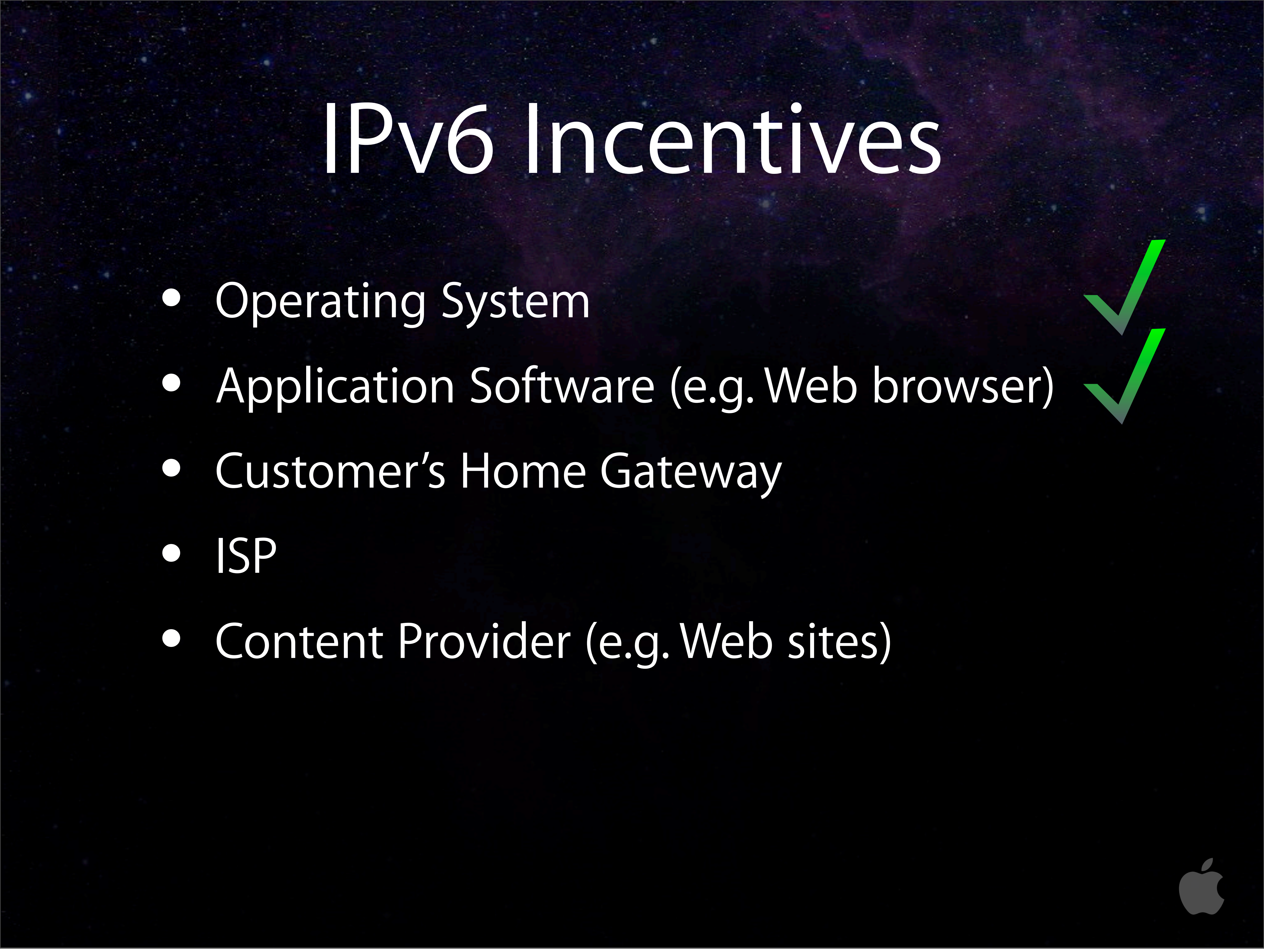


IPv6 Incentives

- Operating System ✓
- Application Software (e.g. Web browser)
- Customer’s Home Gateway
- ISP
- Content Provider (e.g. Web sites)

Sunday, 7 November, 2010

Apple’s business is largely end-users at home using iMacs.
In Enterprise markets this picture might look a bit different, but in this context we have five players who need to cooperate to make IPv6 a success.
So, in reverse order:
* We need content, and I’ll use the web as the common example of what people use the Internet for. We need web sites that have content, that are reachable by IPv6
* but there’s no point having that web site if no ISP offers IPv6 to customers
* and there’s no point an ISP offering it if the customer’s home gateway doesn’t support IPv6,
* and there’s no point doing that if the web browser doesn’t do IPv6
* and you can’t do a web browser that supports IPv6 unless the operating system does
so everybody needs to collaborate here
Now, there’s no incentive for any of these players to move first, because any one of these players does all this work, and the user gets no benefit because the other people haven’t done their bit.
So, the problem we have is providing incentives.
Well, IPv6 is cool, and thankfully, in many cases, that’s enough
↳and that’s why Apple has IPv6 in the operating system
↳and that’s why most of our network applications already support IPv6 too
but now we need the other players to catch up
and we need incentives for them to move to IPv6

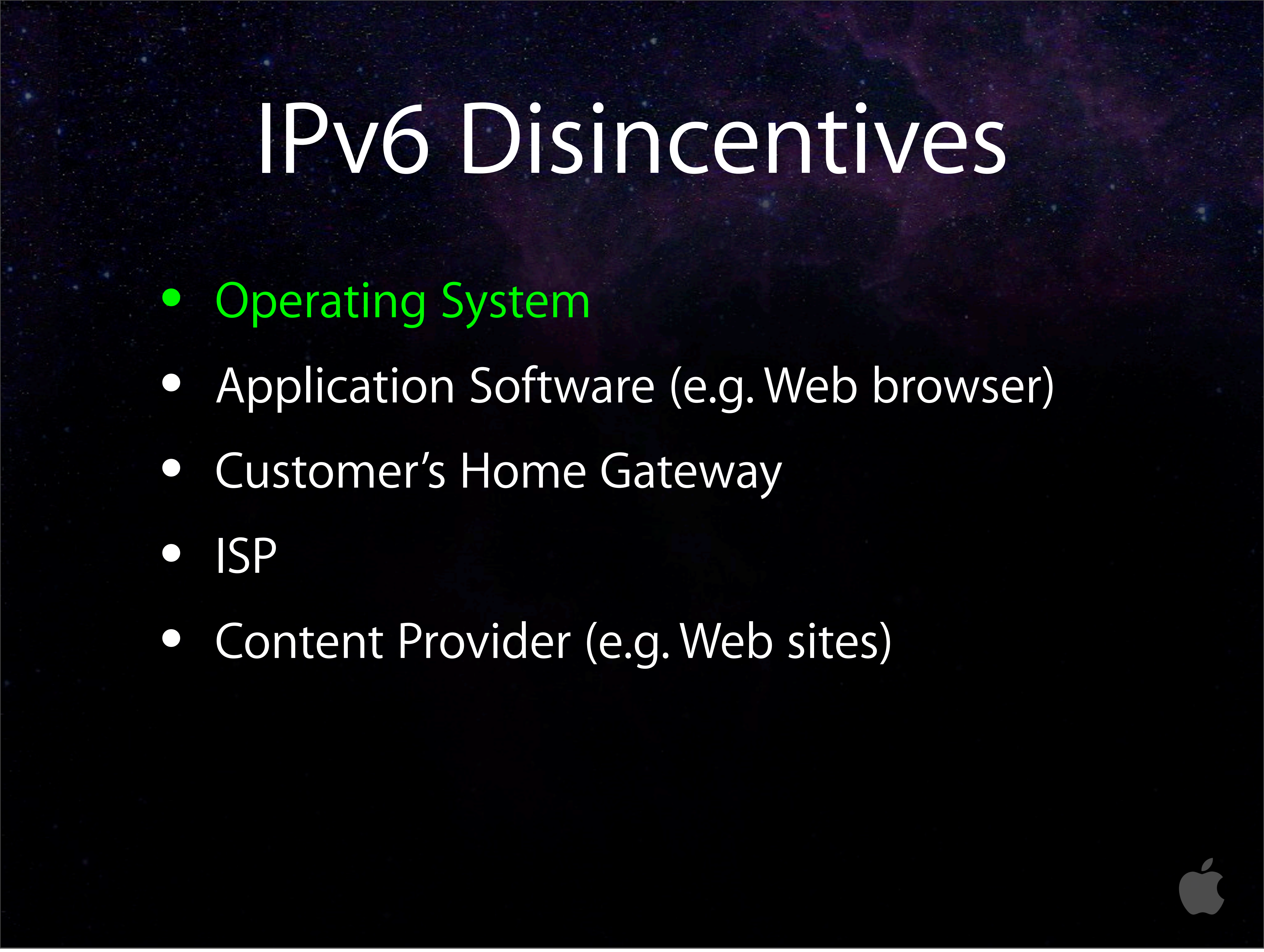


IPv6 Incentives

- Operating System ✓
- Application Software (e.g. Web browser) ✓
- Customer’s Home Gateway
- ISP
- Content Provider (e.g. Web sites)

Sunday, 7 November, 2010

Apple’s business is largely end-users at home using iMacs.
In Enterprise markets this picture might look a bit different, but in this context we have five players who need to cooperate to make IPv6 a success.
So, in reverse order:
* We need content, and I’ll use the web as the common example of what people use the Internet for. We need web sites that have content, that are reachable by IPv6
* but there’s no point having that web site if no ISP offers IPv6 to customers
* and there’s no point an ISP offering it if the customer’s home gateway doesn’t support IPv6,
* and there’s no point doing that if the web browser doesn’t do IPv6
* and you can’t do a web browser that supports IPv6 unless the operating system does
so everybody needs to collaborate here
Now, there’s no incentive for any of these players to move first, because any one of these players does all this work, and the user gets no benefit because the other people haven’t done their bit.
So, the problem we have is providing incentives.
Well, IPv6 is cool, and thankfully, in many cases, that’s enough
↳and that’s why Apple has IPv6 in the operating system
↳and that’s why most of our network applications already support IPv6 too
but now we need the other players to catch up
and we need incentives for them to move to IPv6



IPv6 Disincentives

- Operating System
- Application Software (e.g. Web browser)
- Customer's Home Gateway
- ISP
- Content Provider (e.g. Web sites)

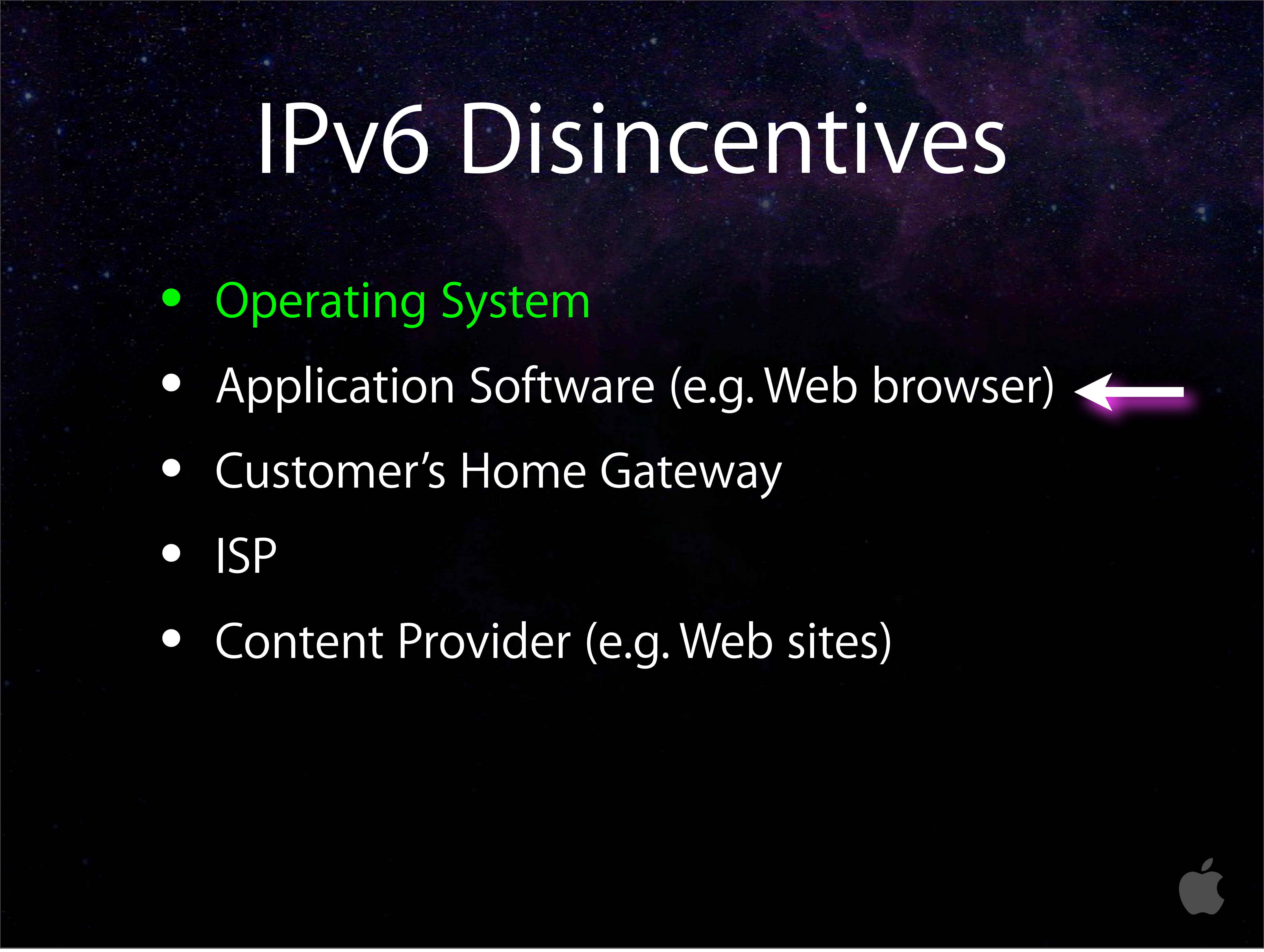
Sunday, 7 November, 2010

Or at least, we don't want there to be disincentives

And this is the focus of my talk. We don't want adopting IPv6 to result in a bad user experience, where networking is slow or broken.

Adopting IPv6 in the OS is easy. There's really no downside. If no applications are using it, then just having it in the OS doesn't really hurt anything, and this is why Mac OS X, Windows, and pretty much all other major operating systems now support IPv6.

The next step is what happens when applications start to use IPv6, ↳and that's what I'm going to talk about today.



IPv6 Disincentives

- Operating System
- Application Software (e.g. Web browser) ←
- Customer’s Home Gateway
- ISP
- Content Provider (e.g. Web sites)

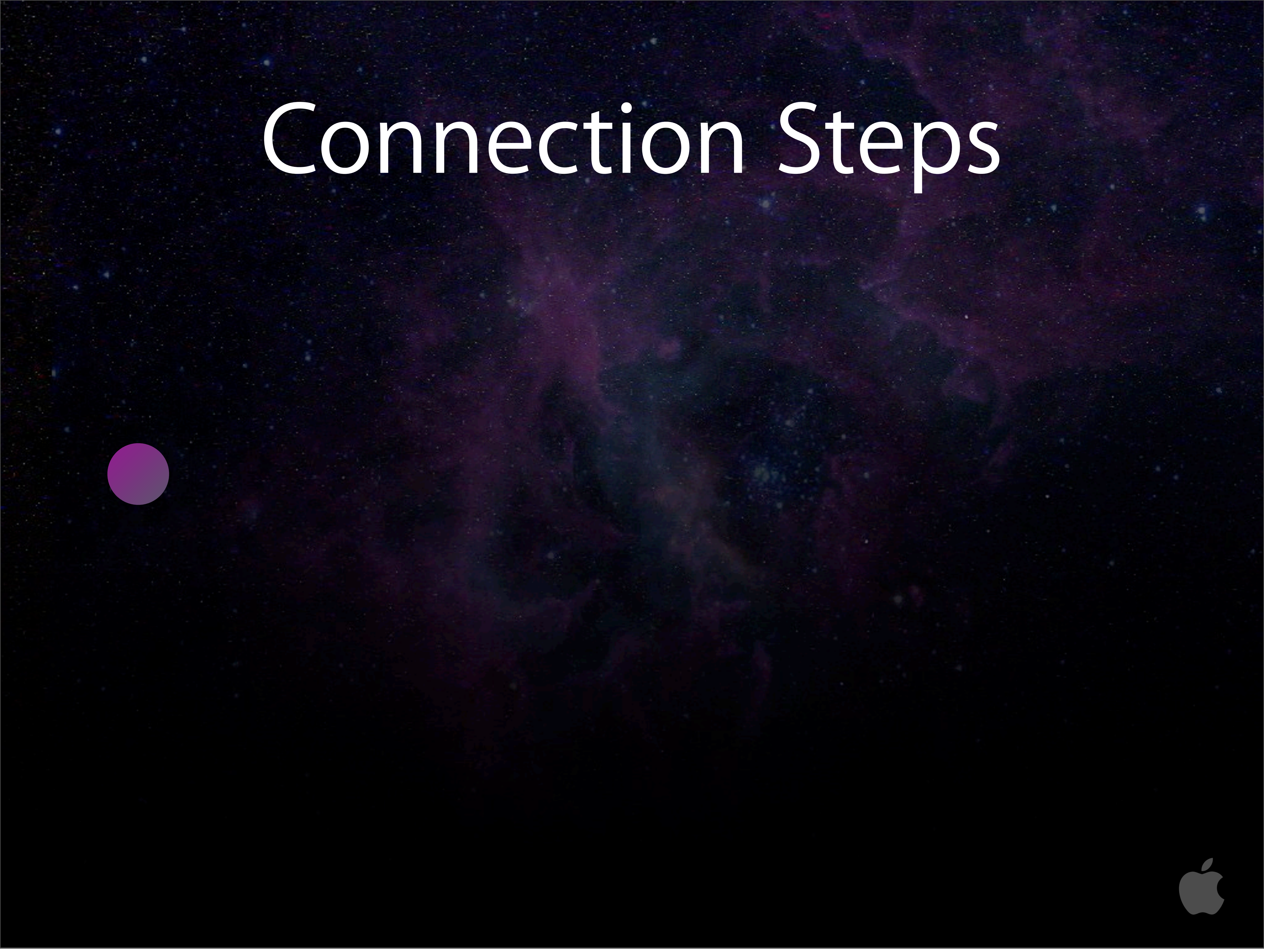
Sunday, 7 November, 2010

Or at least, we don’t want there to be disincentives

And this is the focus of my talk. We don’t want adopting IPv6 to result in a bad user experience, where networking is slow or broken.

Adopting IPv6 in the OS is easy. There’s really no downside. If no applications are using it, then just having it in the OS doesn’t really hurt anything, and this is why Mac OS X, Windows, and pretty much all other major operating systems now support IPv6.

The next step is what happens when applications start to use IPv6, ↳and that’s what I’m going to talk about today.



Connection Steps

Sunday, 7 November, 2010

This is an example of what might happen when you view a web page in your web browser.

First your machine looks up the IPv6 quad-A address record
Then it looks up the IPv4 address record
Then it tries to connect with IPv6… but that connection might fail
So then it connects with IPv4

And this picture is great — as long as that whole timeline completes nearly instantaneously, so that the user doesn't notice.

The problem is that that IPv6 failure in the middle might not happen instantaneously. It might take a minute, or two, or three minutes to time out, depending on the TCP stack, and the user isn't going to be willing to wait three minutes.

Connection Steps

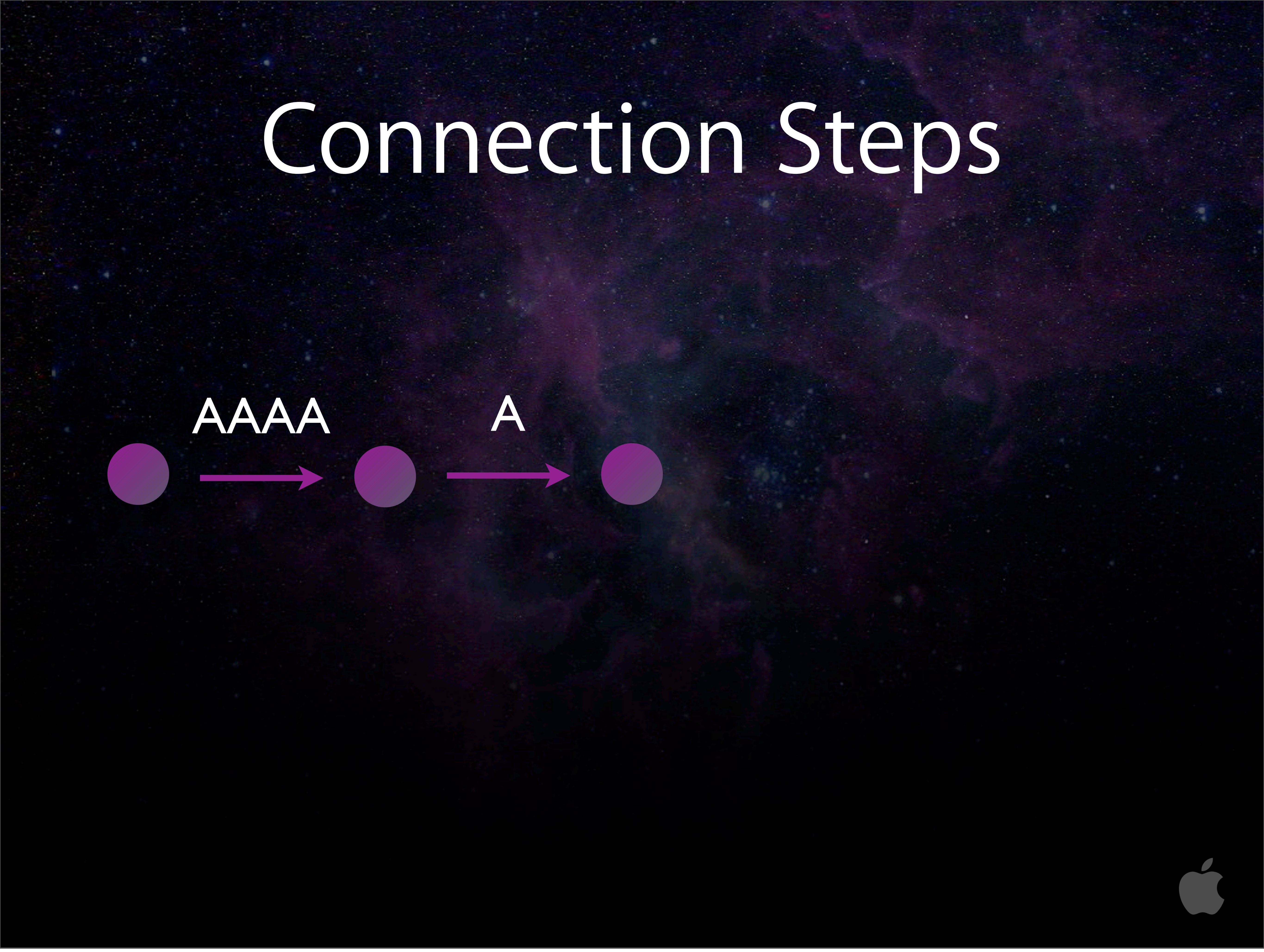
AAAA

This is an example of what might happen when you view a web page in your web browser.

First your machine looks up the IPv6 quad-A address record
Then it looks up the IPv4 address record
Then it tries to connect with IPv6… but that connection might fail
So then it connects with IPv4

And this picture is great — as long as that whole timeline completes nearly instantaneously, so that the user doesn't notice.

The problem is that that IPv6 failure in the middle might not happen instantaneously. It might take a minute, or two, or three minutes to time out, depending on the TCP stack, and the user isn't going to be willing to wait three minutes.



Connection Steps

Sunday, 7 November, 2010

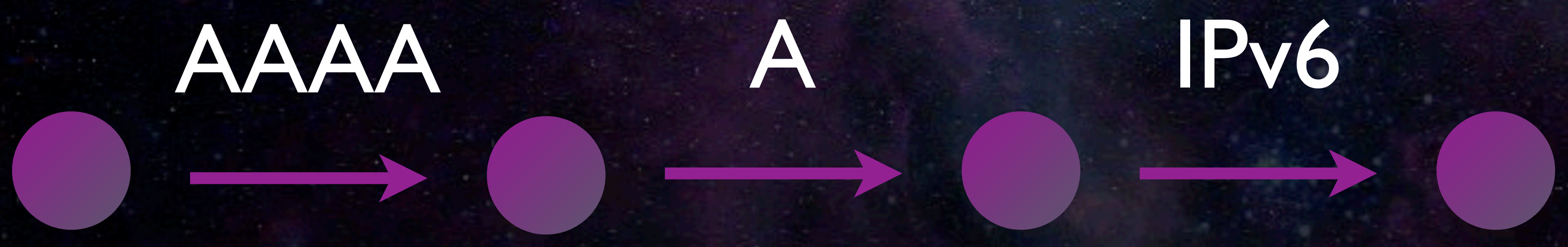
This is an example of what might happen when you view a web page in your web browser.

First your machine looks up the IPv6 quad-A address record
Then it looks up the IPv4 address record
Then it tries to connect with IPv6… but that connection might fail
So then it connects with IPv4

And this picture is great — as long as that whole timeline completes nearly instantaneously, so that the user doesn't notice.

The problem is that that IPv6 failure in the middle might not happen instantaneously. It might take a minute, or two, or three minutes to time out, depending on the TCP stack, and the user isn't going to be willing to wait three minutes.

Connection Steps



AAAA

A

IPv6

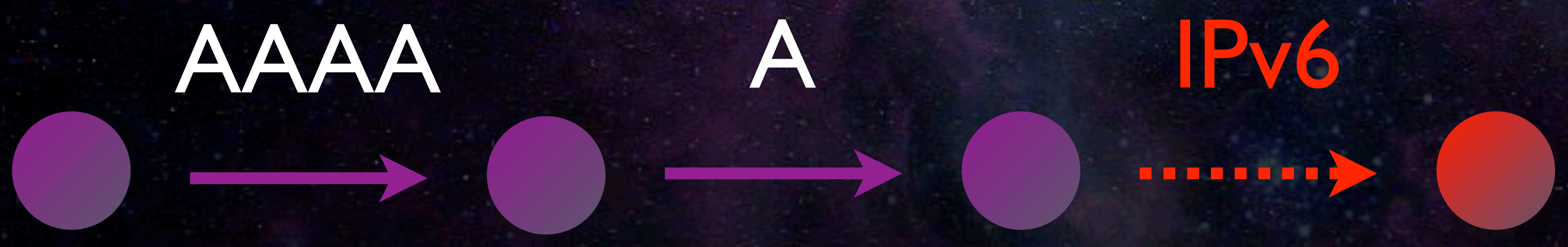
This is an example of what might happen when you view a web page in your web browser.

First your machine looks up the IPv6 quad-A address record
Then it looks up the IPv4 address record
Then it tries to connect with IPv6… but that connection might fail
So then it connects with IPv4

And this picture is great — as long as that whole timeline completes nearly instantaneously, so that the user doesn’t notice.

The problem is that that IPv6 failure in the middle might not happen instantaneously. It might take a minute, or two, or three minutes to time out, depending on the TCP stack, and the user isn’t going to be willing to wait three minutes.

Connection Steps



Sunday, 7 November, 2010

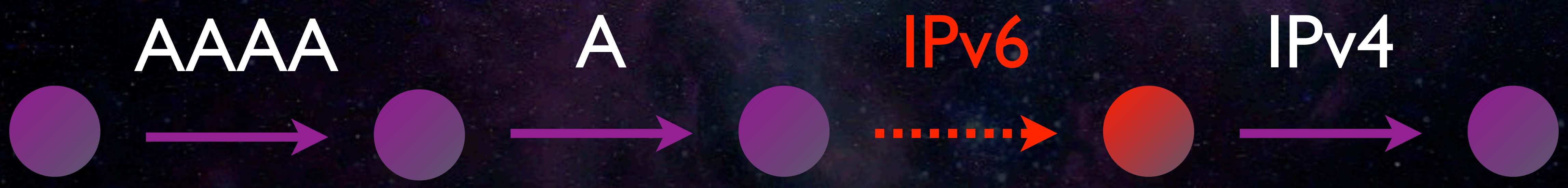
This is an example of what might happen when you view a web page in your web browser.

First your machine looks up the IPv6 quad-A address record
Then it looks up the IPv4 address record
Then it tries to connect with IPv6… but that connection might fail
So then it connects with IPv4

And this picture is great — as long as that whole timeline completes nearly instantaneously, so that the user doesn’t notice.

The problem is that that IPv6 failure in the middle might not happen instantaneously. It might take a minute, or two, or three minutes to time out, depending on the TCP stack, and the user isn’t going to be willing to wait three minutes.

Connection Steps



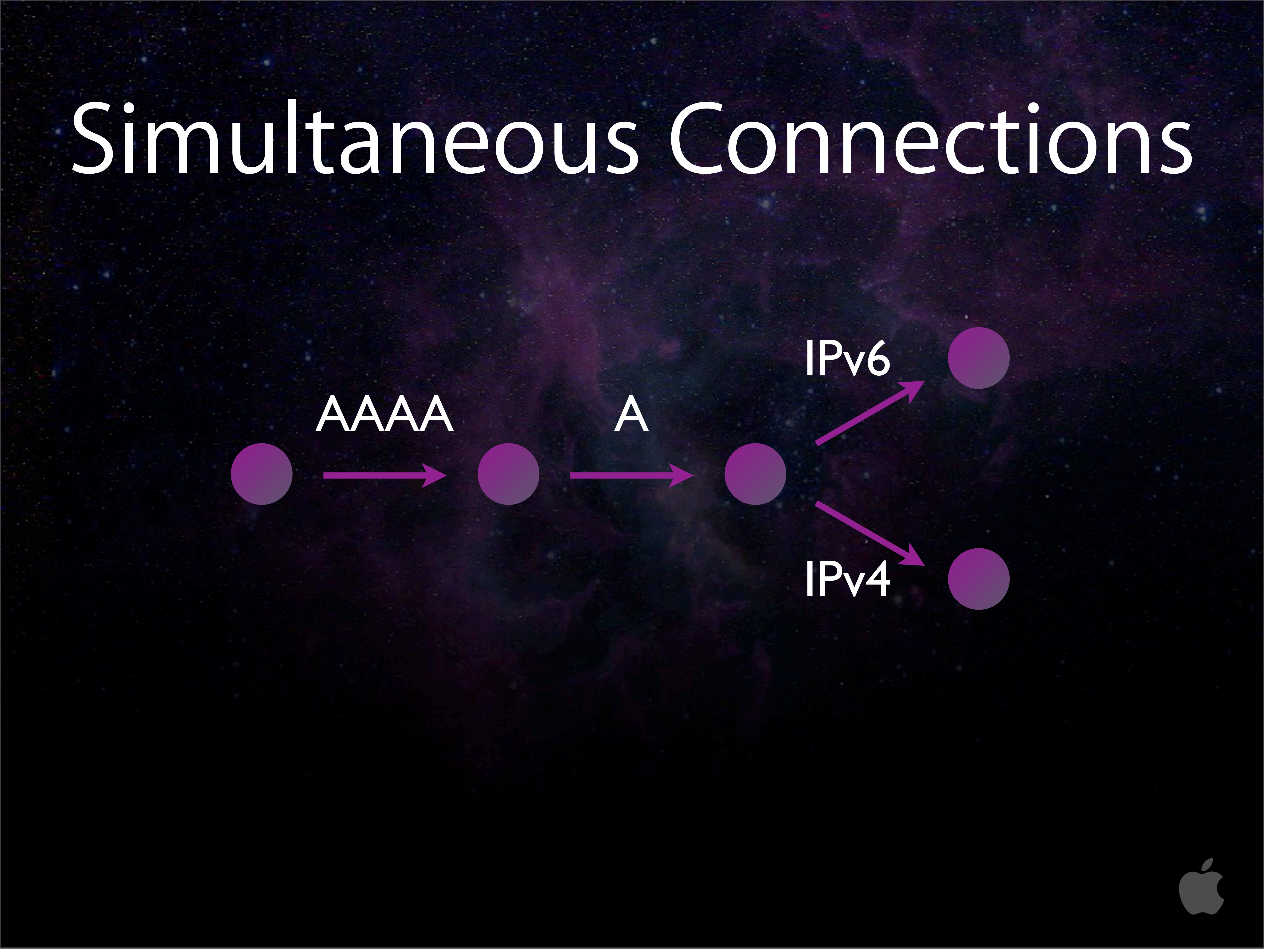
Sunday, 7 November, 2010

This is an example of what might happen when you view a web page in your web browser.

First your machine looks up the IPv6 quad-A address record
Then it looks up the IPv4 address record
Then it tries to connect with IPv6… but that connection might fail
So then it connects with IPv4

And this picture is great — as long as that whole timeline completes nearly instantaneously, so that the user doesn’t notice.

The problem is that that IPv6 failure in the middle might not happen instantaneously. It might take a minute, or two, or three minutes to time out, depending on the TCP stack, and the user isn’t going to be willing to wait three minutes.

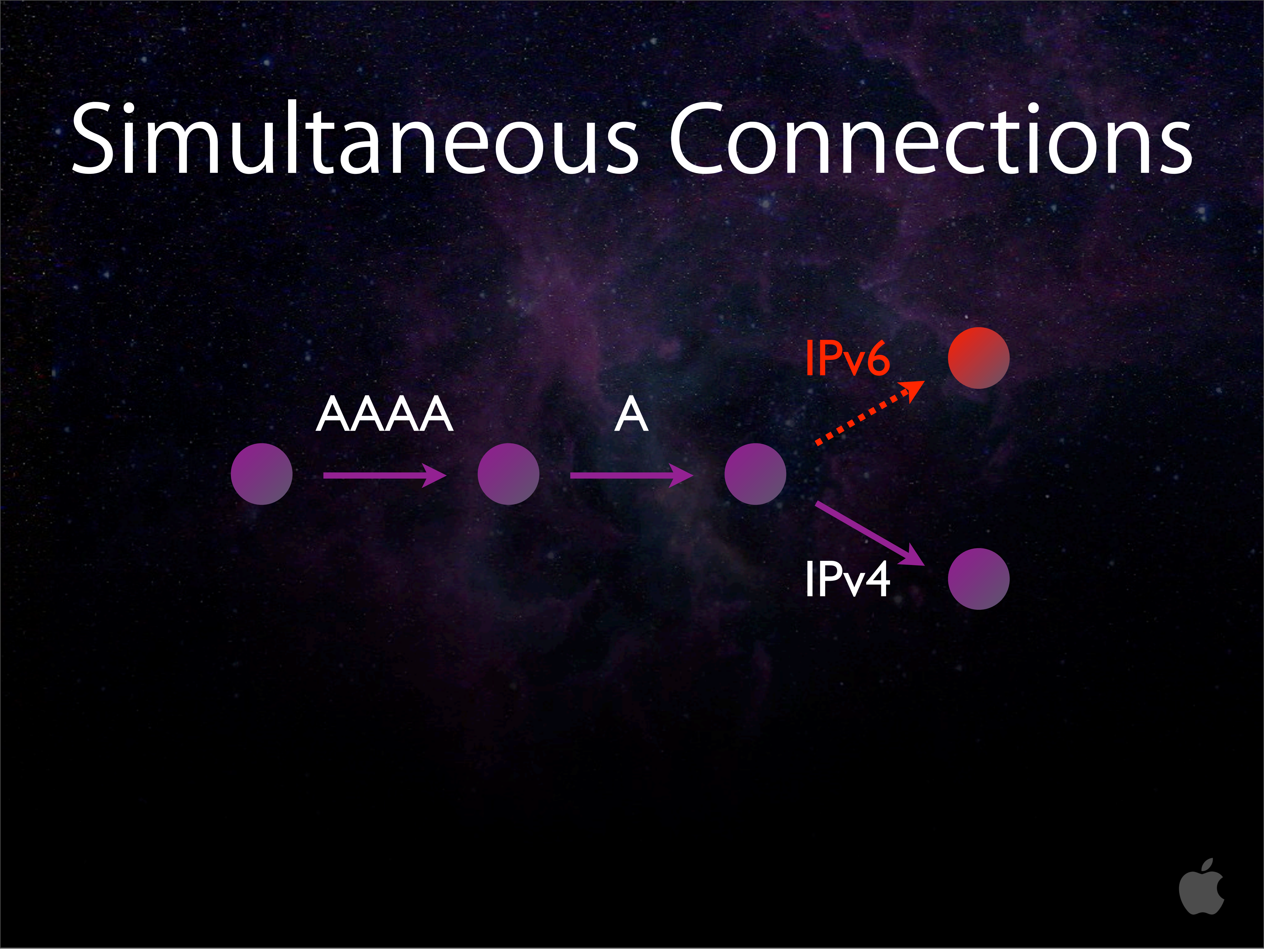


Simultaneous Connections

Sunday, 7 November, 2010

We can solve that by doing the two connections in parallel, and that way, if the IPv6 connection fails, that doesn’t block the IPv4 connection from completing in a timely fashion.

Doing those DNS queries sequentially is a bit of a waste of time, and we can solve that by doing those in parallel too.



Simultaneous Connections

AAAA

A

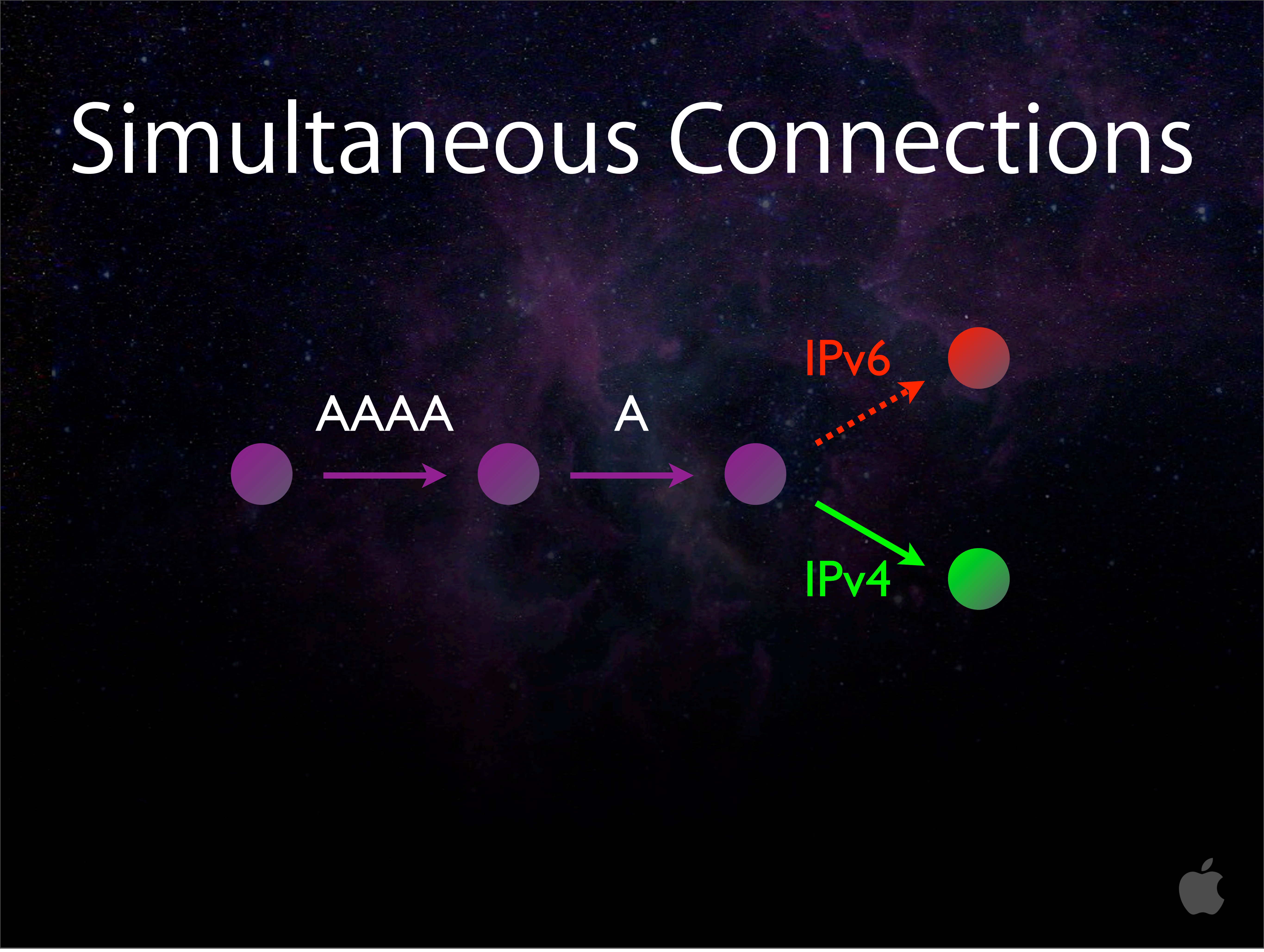
IPv6

IPv4

Sunday, 7 November, 2010

We can solve that by doing the two connections in parallel, and that way, if the IPv6 connection fails, that doesn’t block the IPv4 connection from completing in a timely fashion.

Doing those DNS queries sequentially is a bit of a waste of time, and we can solve that by doing those in parallel too.

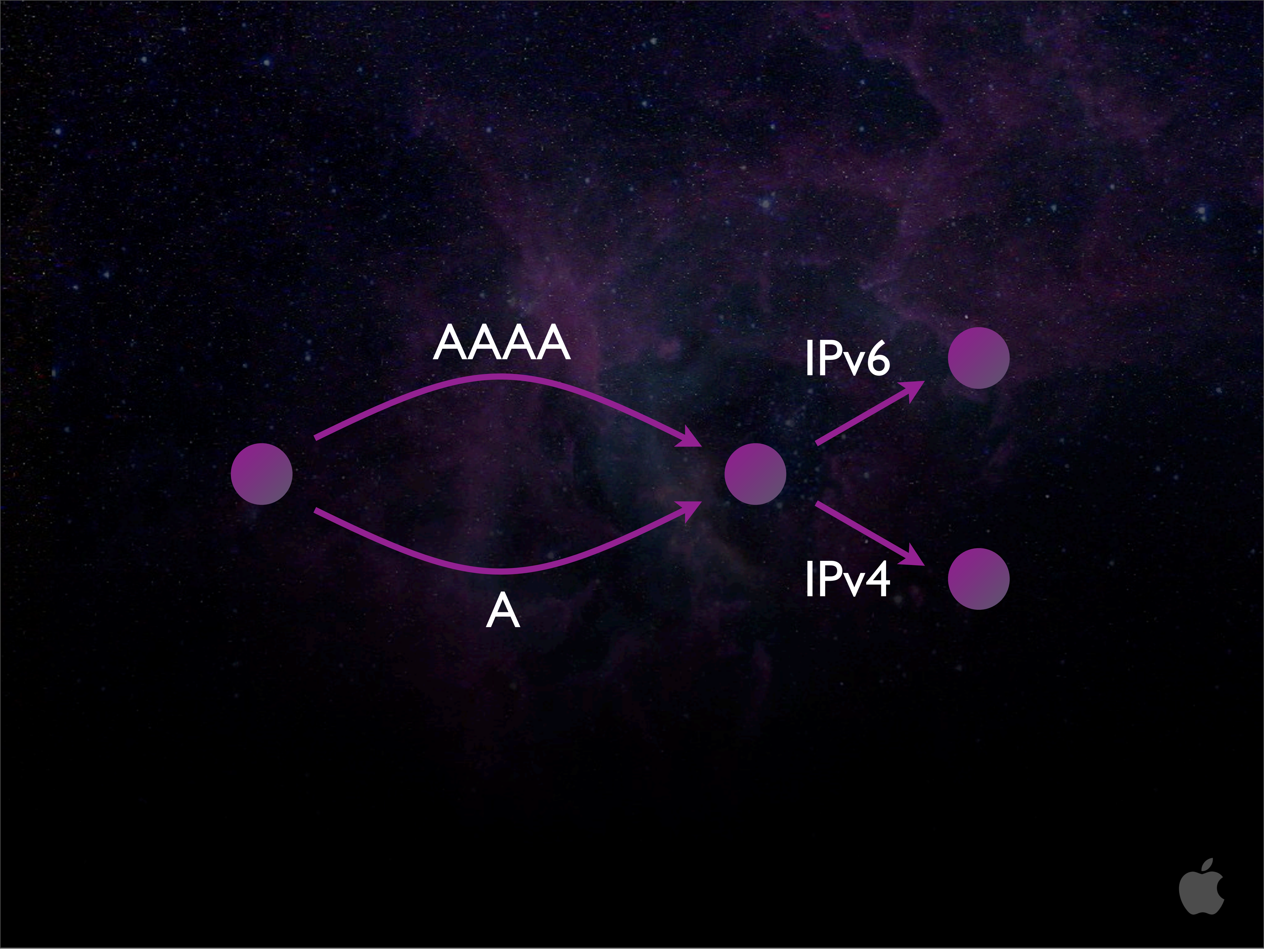


Simultaneous Connections

Sunday, 7 November, 2010

We can solve that by doing the two connections in parallel, and that way, if the IPv6 connection fails, that doesn't block the IPv4 connection from completing in a timely fashion.

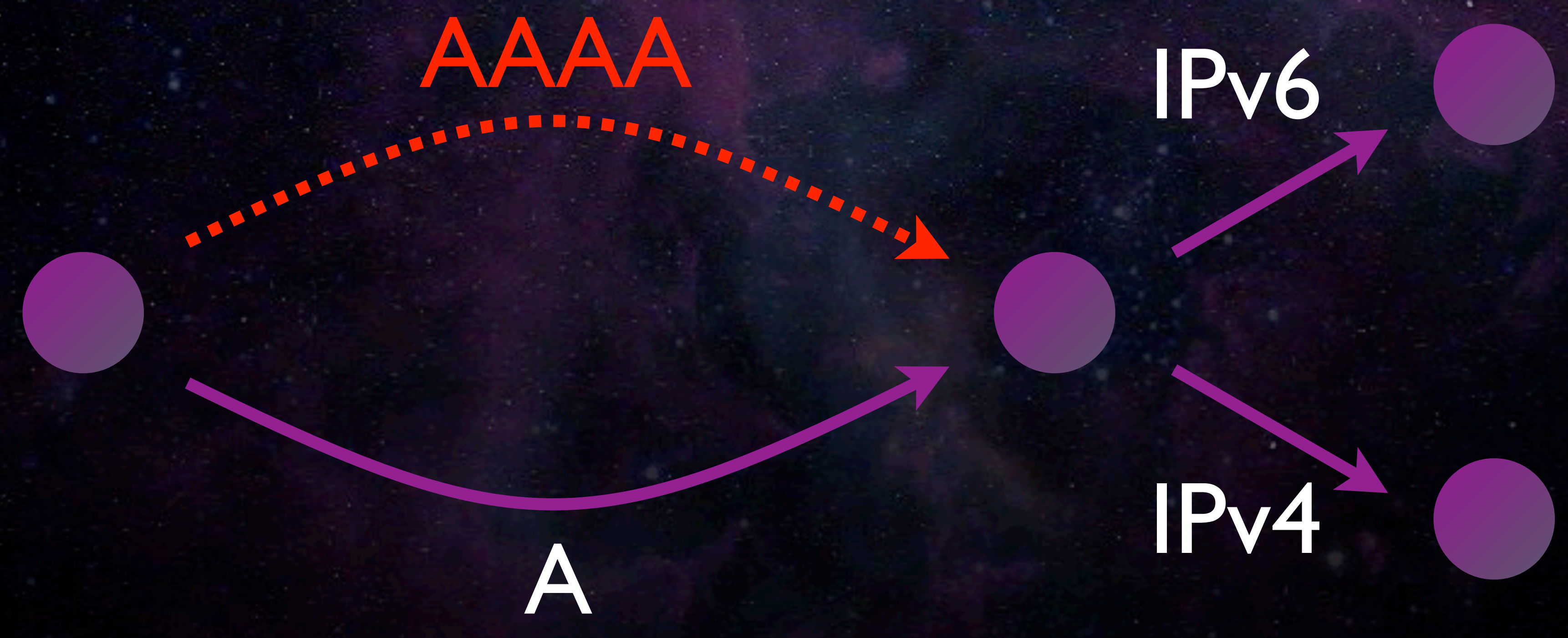
Doing those DNS queries sequentially is a bit of a waste of time, and we can solve that by doing those in parallel too.



Sunday, 7 November, 2010

So, this is good, but there's still a problem.
There are many cases, we have found, with not just small web sites, but big-name web sites, where those quad-A queries fail. They don't give an error. They don't give a negative response. The queries are just ignored, so the client retransmits, and it fails with a timeout, and that takes a long time, and the problem there is that it's blocking progress on the IPv4 connection while we're waiting for an IPv6 address that isn't coming. People try to work out heuristics for how long to wait like, maybe, how long the address query takes as a predictor of how long the quad-A will take, but that's not a good predictor because quite often the address record is cached in the local DNS cache but the quad-A isn't. So, we don't know how long to wait, and even waiting a couple of seconds is too long. The Safari engineers work late nights and weekends to shave every millisecond off the page load times, and if they've got a particular page loading in 0.7 seconds, for us to come along and say, "Oh, yeah, we want to add a five-second pause on that," is completely unacceptable. It's like telling Ferrari engineers that you want to make a little change to their car, and by-the-way, the top speed will now be 12 miles per hour. They're not going to go for that.

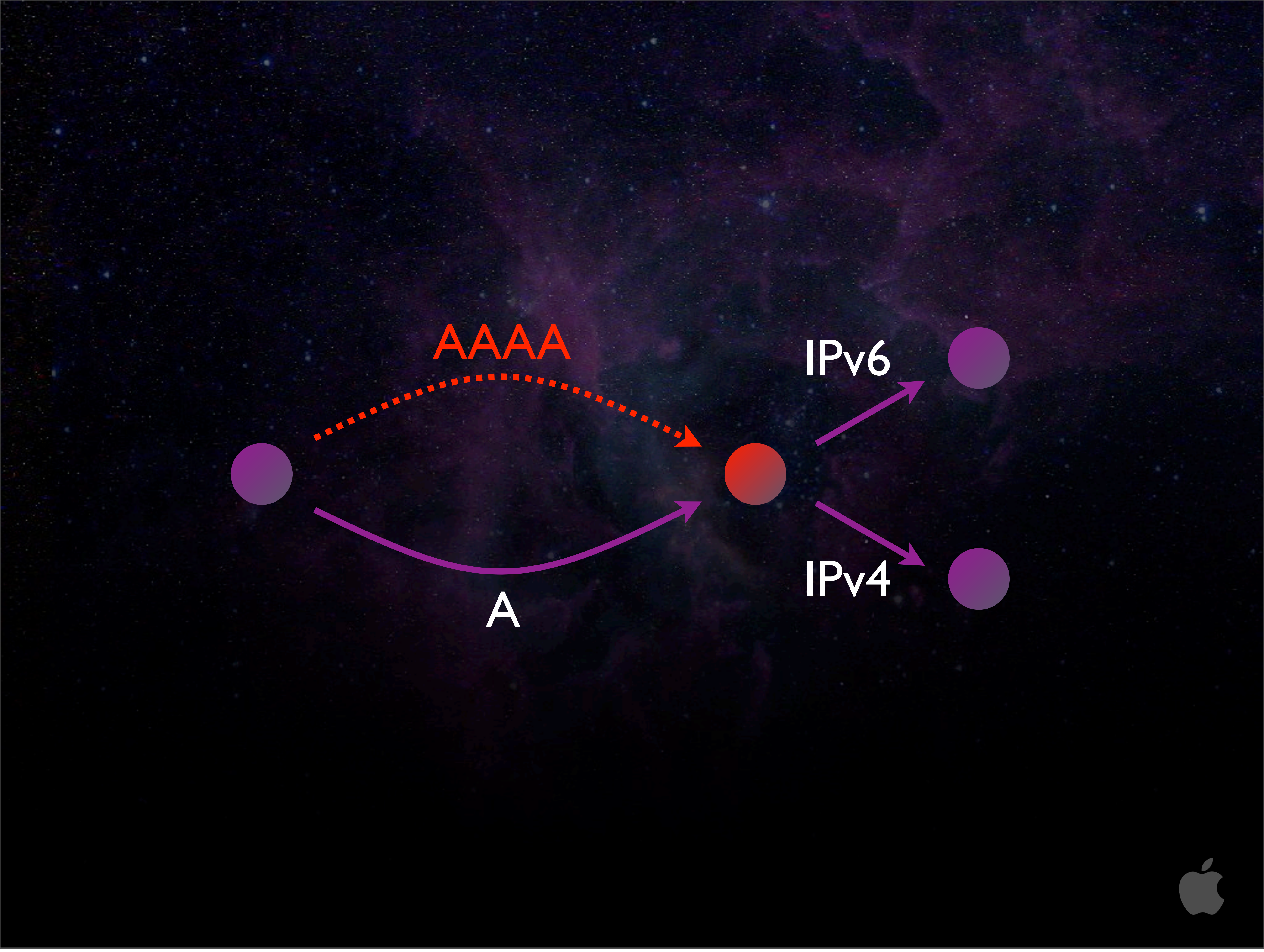
That picture really describes how getaddrinfo() works, and fundamentally the problem is that it is blocked waiting for information that isn't coming.



Sunday, 7 November, 2010

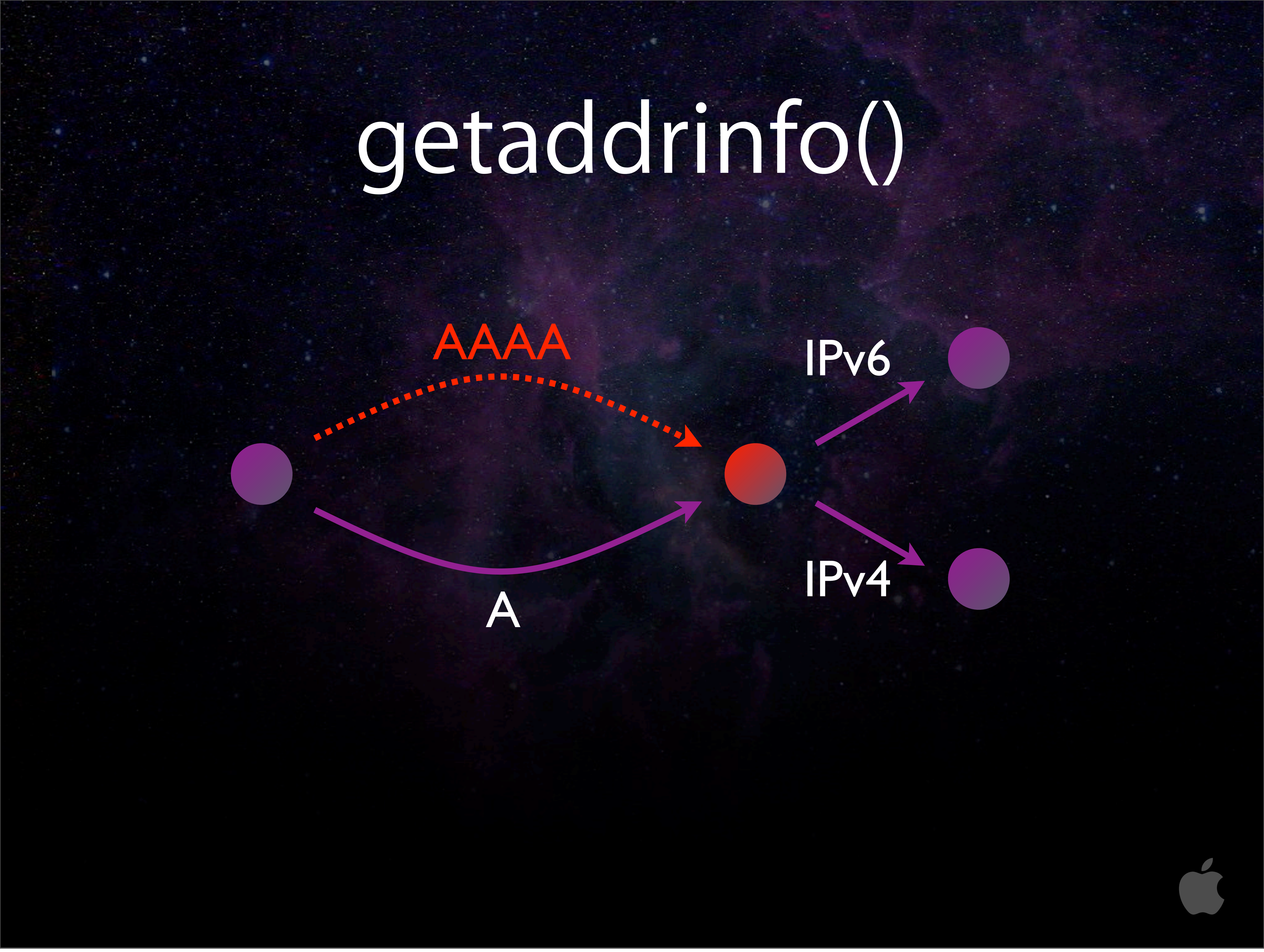
So, this is good, but there’s still a problem.
There are many cases, we have found, with not just small web sites, but big-name web sites, where those quad-A queries fail. They don’t give an error. They don’t give a negative response. The queries are just ignored, so the client retransmits, and it fails with a timeout, and that takes a long time, and the problem there is that it’s blocking progress on the IPv4 connection while we’re waiting for an IPv6 address that isn’t coming. People try to work out heuristics for how long to wait like, maybe, how long the address query takes as a predictor of how long the quad-A will take, but that’s not a good predictor because quite often the address record is cached in the local DNS cache but the quad-A isn’t. So, we don’t know how long to wait, and even waiting a couple of seconds is too long. The Safari engineers work late nights and weekends to shave every millisecond off the page load times, and if they’ve got a particular page loading in 0.7 seconds, for us to come along and say, “Oh, yeah, we want to add a five-second pause on that,” is completely unacceptable. It’s like telling Ferrari engineers that you want to make a little change to their car, and by-the-way, the top speed will now be 12 miles per hour. They’re not going to go for that.

That picture really describes how getaddrinfo() works, and fundamentally the problem is that it is blocked waiting for information that isn’t coming.



So, this is good, but there's still a problem.
There are many cases, we have found, with not just small web sites, but big-name web sites, where those quad-A queries fail. They don't give an error. They don't give a negative response. The queries are just ignored, so the client retransmits, and it fails with a timeout, and that takes a long time, and the problem there is that it's blocking progress on the IPv4 connection while we're waiting for an IPv6 address that isn't coming. People try to work out heuristics for how long to wait like, maybe, how long the address query takes as a predictor of how long the quad-A will take, but that's not a good predictor because quite often the address record is cached in the local DNS cache but the quad-A isn't. So, we don't know how long to wait, and even waiting a couple of seconds is too long. The Safari engineers work late nights and weekends to shave every millisecond off the page load times, and if they've got a particular page loading in 0.7 seconds, for us to come along and say, "Oh, yeah, we want to add a five-second pause on that," is completely unacceptable. It's like telling Ferrari engineers that you want to make a little change to their car, and by-the-way, the top speed will now be 12 miles per hour. They're not going to go for that.

That picture really describes how getaddrinfo() works, and fundamentally the problem is that it is blocked waiting for information that isn't coming.

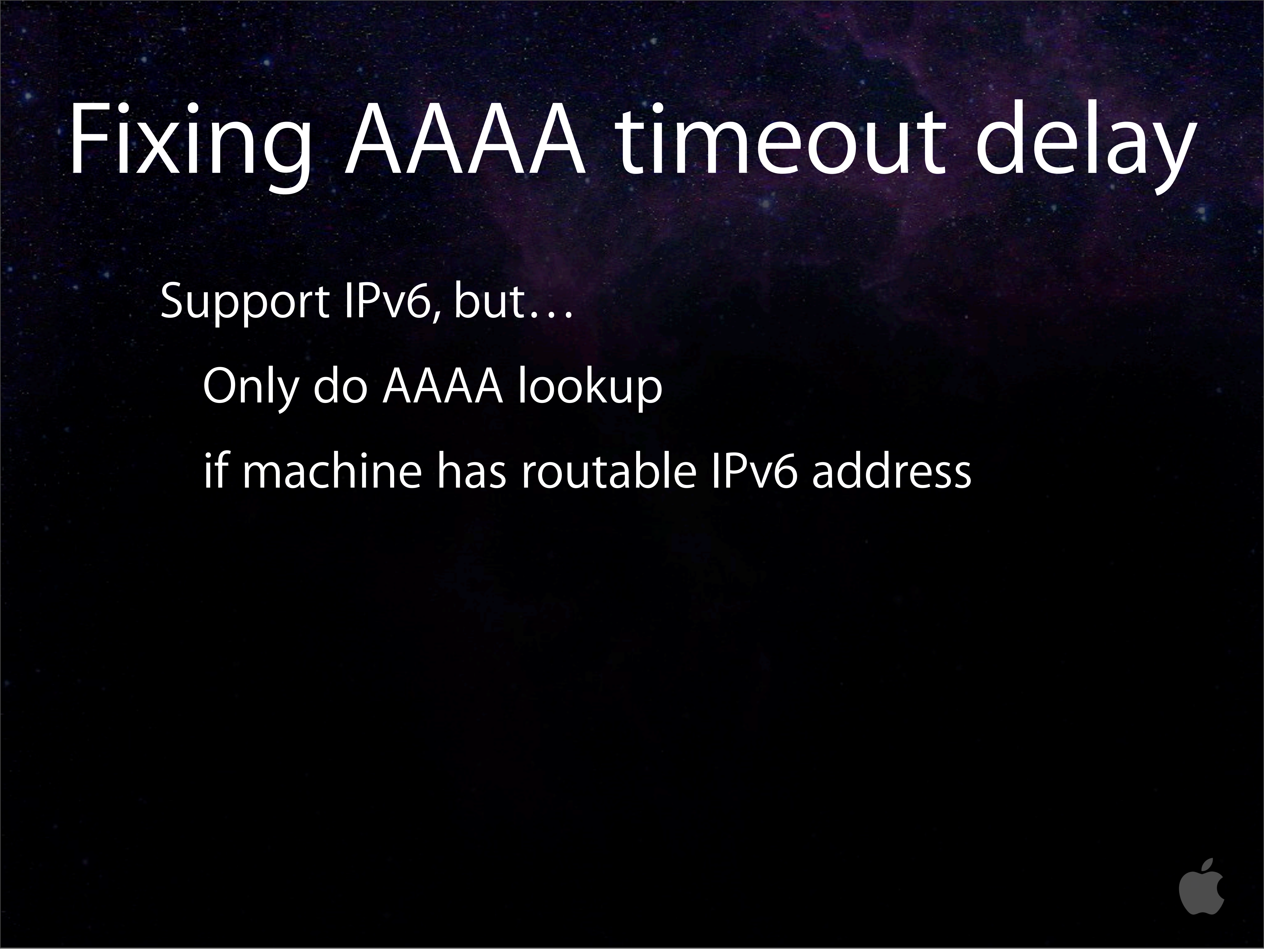


getaddrinfo()

Sunday, 7 November, 2010

So, this is good, but there’s still a problem.
There are many cases, we have found, with not just small web sites, but big-name web sites, where those quad-A queries fail. They don’t give an error. They don’t give a negative response. The queries are just ignored, so the client retransmits, and it fails with a timeout, and that takes a long time, and the problem there is that it’s blocking progress on the IPv4 connection while we’re waiting for an IPv6 address that isn’t coming. People try to work out heuristics for how long to wait like, maybe, how long the address query takes as a predictor of how long the quad-A will take, but that’s not a good predictor because quite often the address record is cached in the local DNS cache but the quad-A isn’t. So, we don’t know how long to wait, and even waiting a couple of seconds is too long. The Safari engineers work late nights and weekends to shave every millisecond off the page load times, and if they’ve got a particular page loading in 0.7 seconds, for us to come along and say, “Oh, yeah, we want to add a five-second pause on that,” is completely unacceptable. It’s like telling Ferrari engineers that you want to make a little change to their car, and by-the-way, the top speed will now be 12 miles per hour. They’re not going to go for that.

That picture really describes how getaddrinfo() works, and fundamentally the problem is that it is blocked waiting for information that isn’t coming.



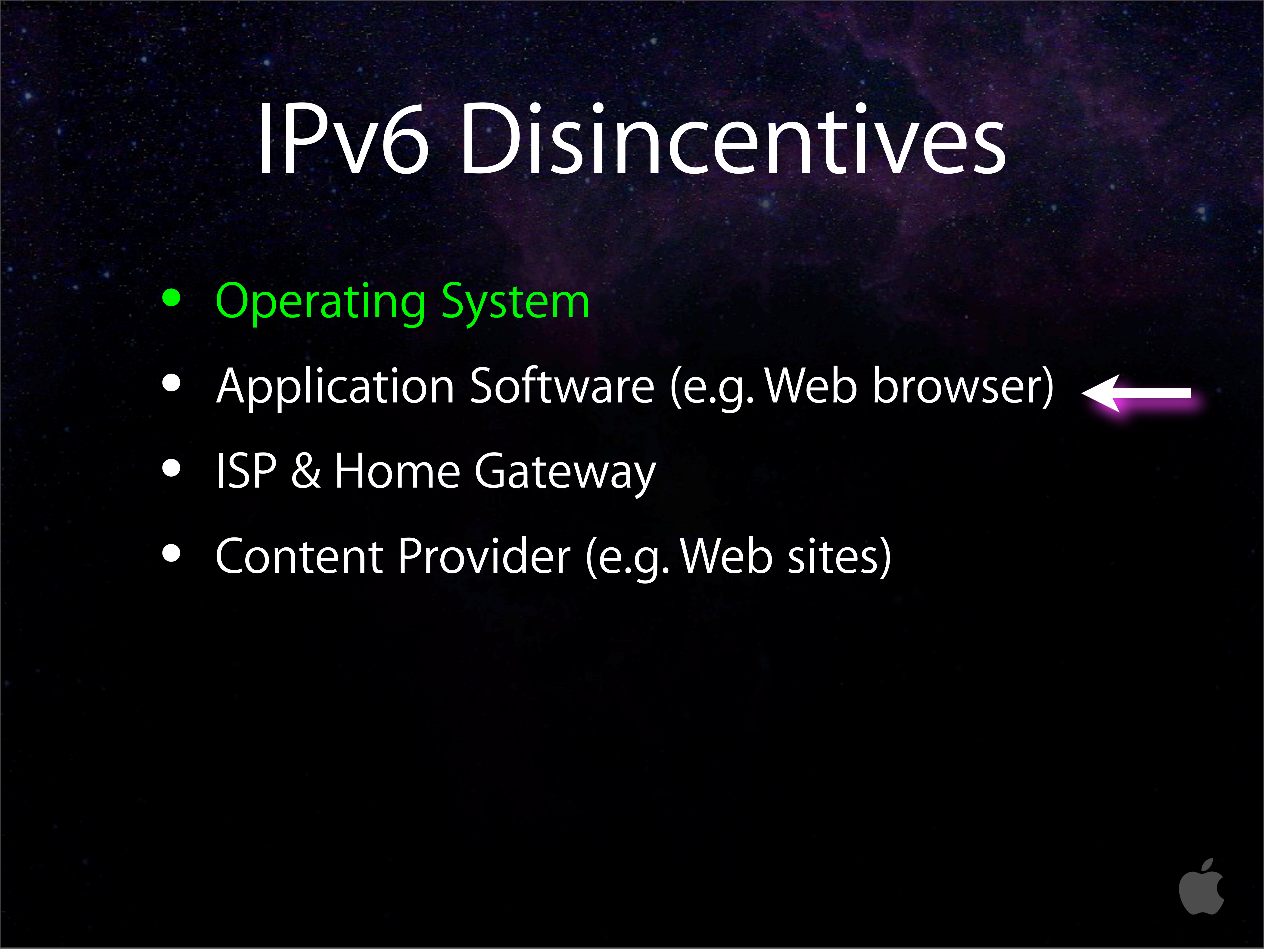
Fixing AAAA timeout delay

- Support IPv6, but…
  - Only do AAAA lookup
  - if machine has routable IPv6 address

Sunday, 7 November, 2010

Now, the standard solution that people give for this is that they say, “If the machine doesn’t have an IPv6 address, then there’s no point doing the quad-A address lookup.”

That masks the symptom, but the irony is that it’s a solution that makes IPv6 work acceptably well, only for people who’re **not using IPv6**. The moment you have an IPv6 address, the problem comes back.



IPv6 Disincentives

- Operating System
- Application Software (e.g. Web browser)
- ISP & Home Gateway
- Content Provider (e.g. Web sites)

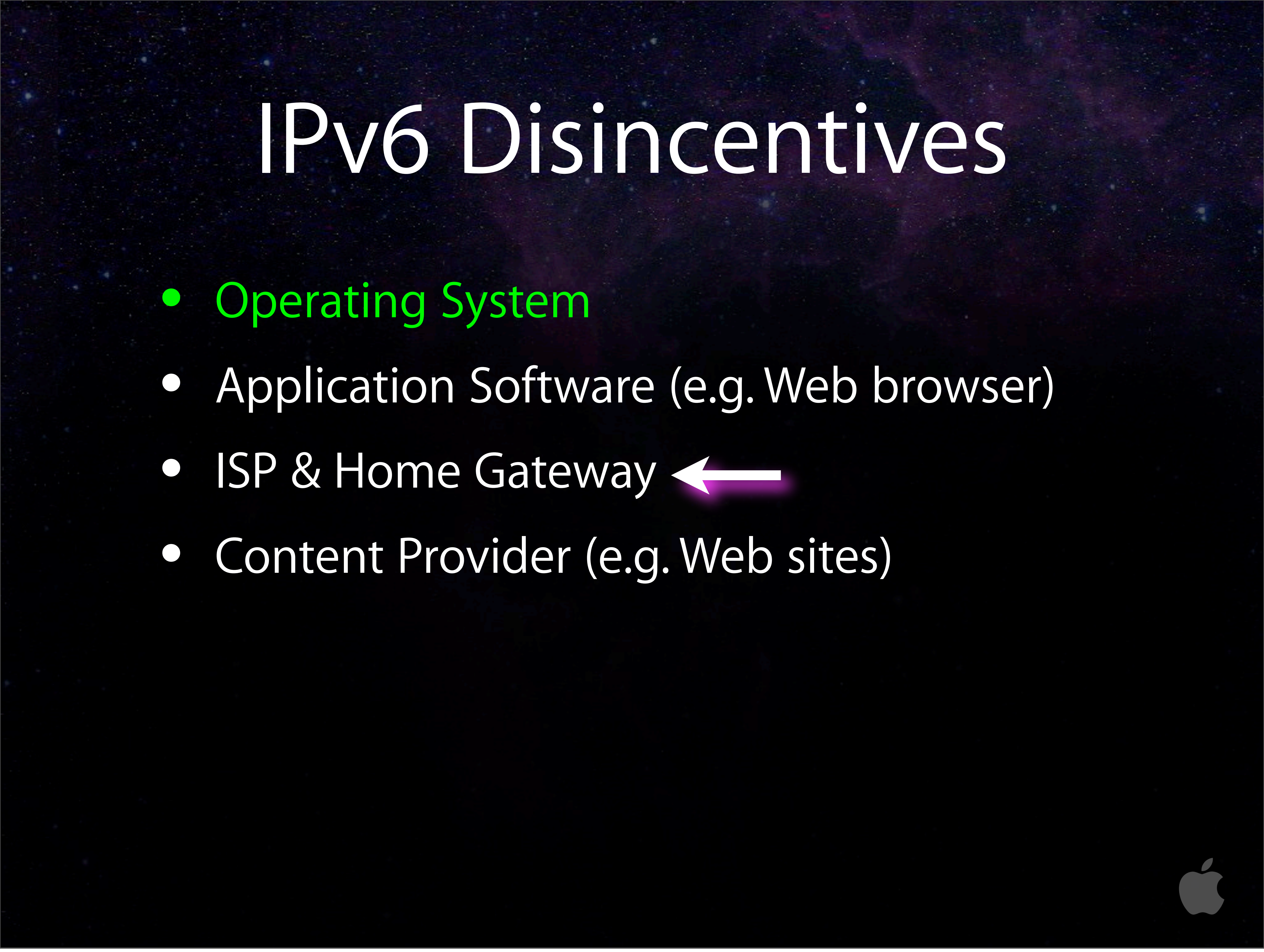
Sunday, 7 November, 2010

And this just pushes problem from the application layer to the ISP.

Why is this bad?

The problem is that when an ISP decides to try deploying IPv6, they start getting customer complains that the web is really slow, and then the ISP turns off IPv6 and swears never to make that mistake again.

We want to make sure that doesn’t happen.



IPv6 Disincentives

- Operating System
- Application Software (e.g. Web browser)
- ISP & Home Gateway ←
- Content Provider (e.g. Web sites)

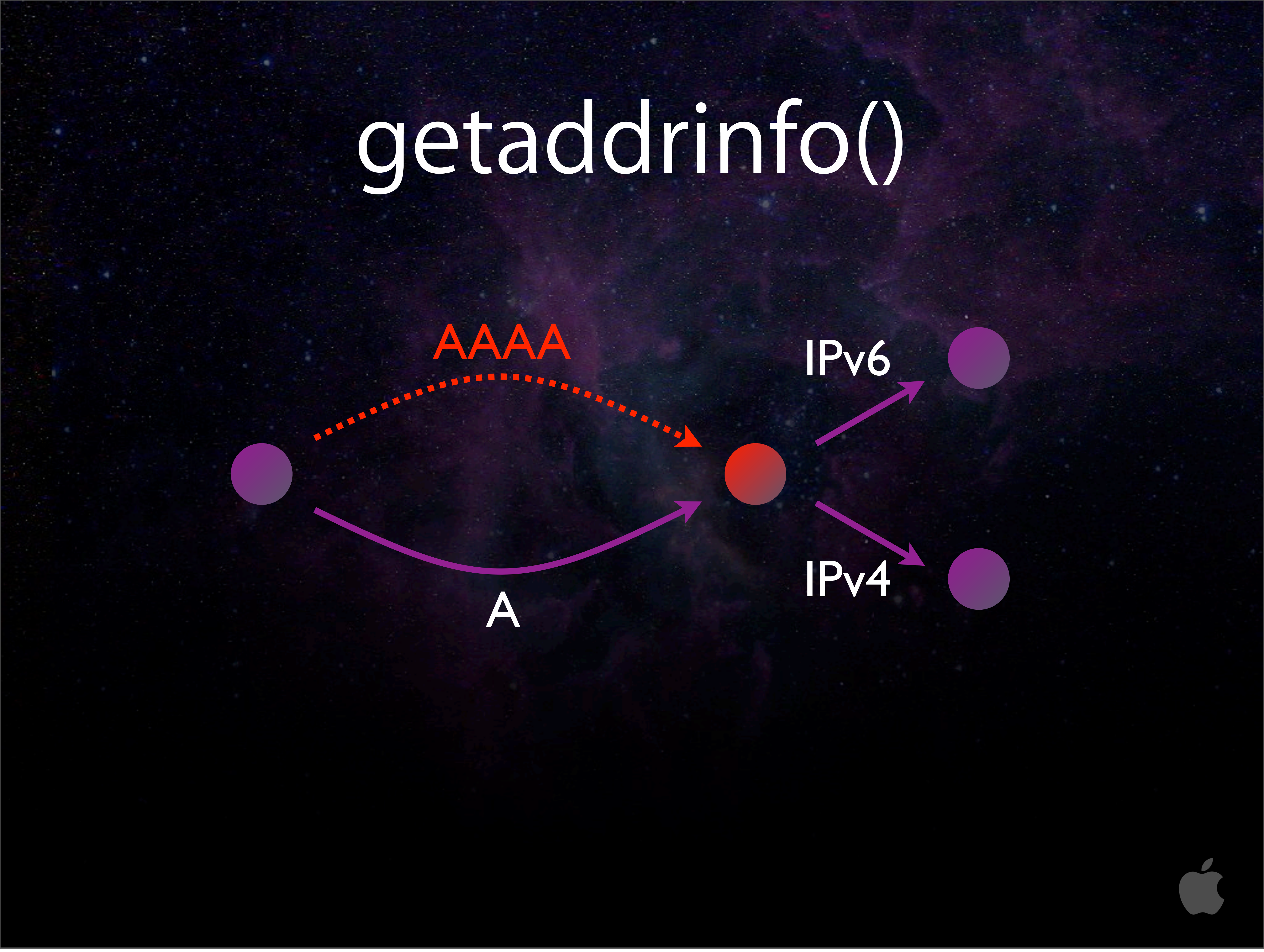
Sunday, 7 November, 2010

And this just pushes problem from the application layer to the ISP.

Why is this bad?

The problem is that when an ISP decides to try deploying IPv6, they start getting customer complains that the web is really slow, and then the ISP turns off IPv6 and swears never to make that mistake again.

We want to make sure that doesn’t happen.



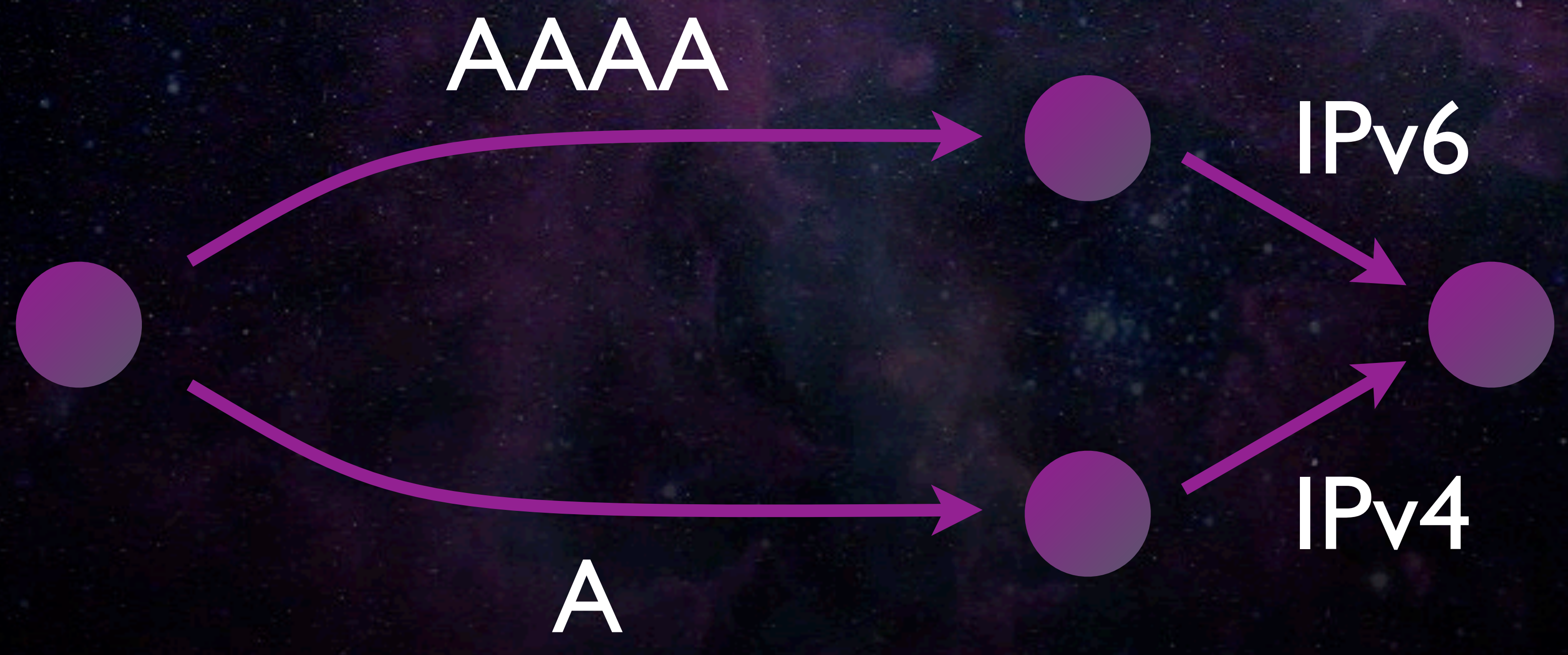
getaddrinfo()

Sunday, 7 November, 2010

getaddrinfo() fundamentally has this problem that:
the v4 connection is blocked waiting for a v6 address it doesn’t need, and
the v6 connection is blocked waiting for a v4 address it doesn’t need

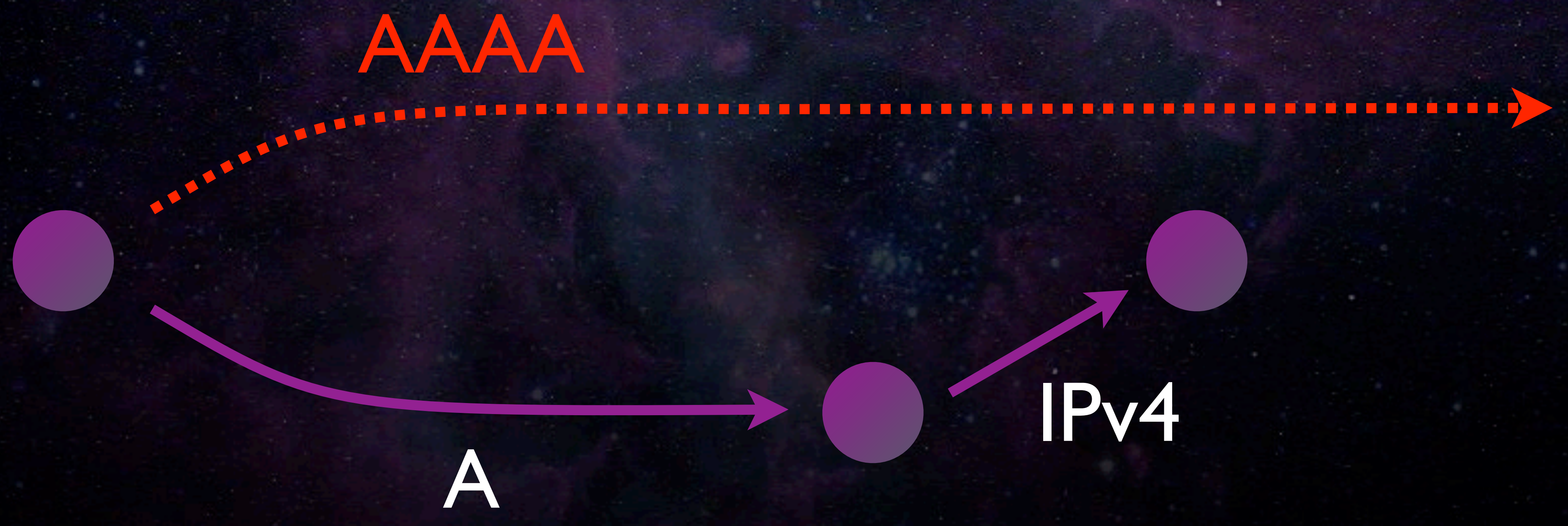
We need to blow up that red node in the middle
and go to more of a dataflow model

Dataflow Model



and now, if the IPv6 quad-A lookup times out, it doesn't block the IPv4 goodness

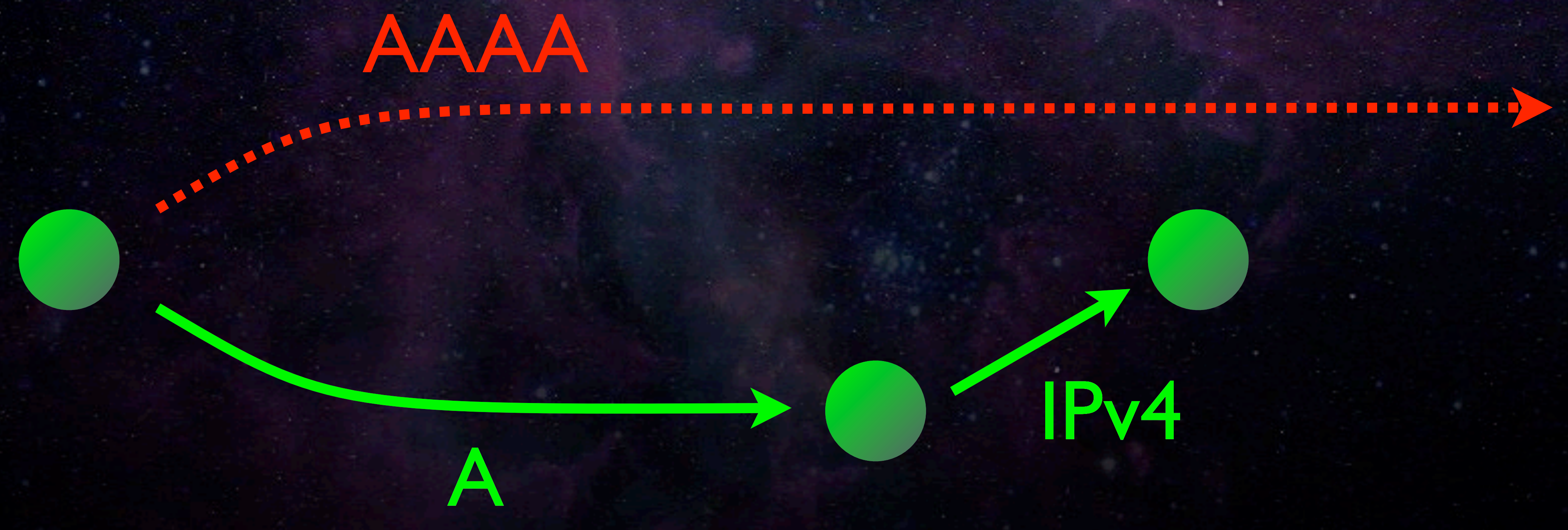
Dataflow Model



Sunday, 7 November, 2010

and now, if the IPv6 quad-A lookup times out, it doesn’t block the IPv4 goodness

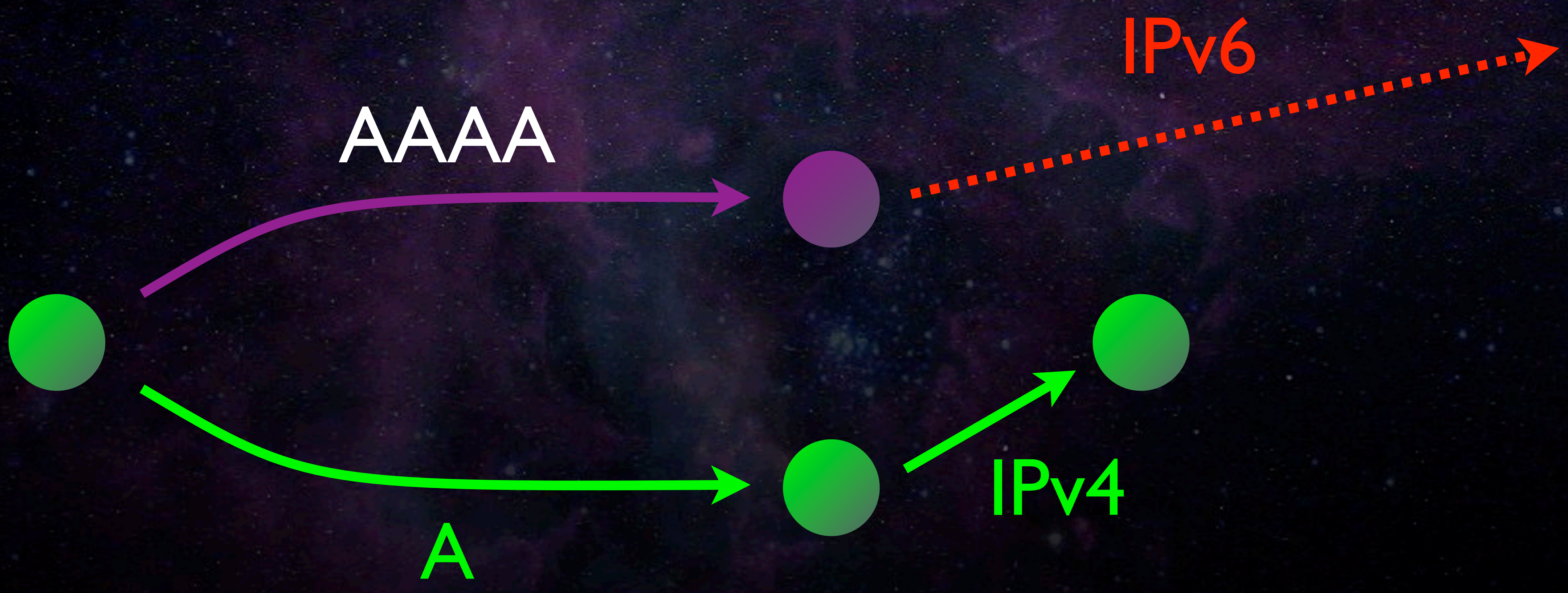
Dataflow Model



Sunday, 7 November, 2010

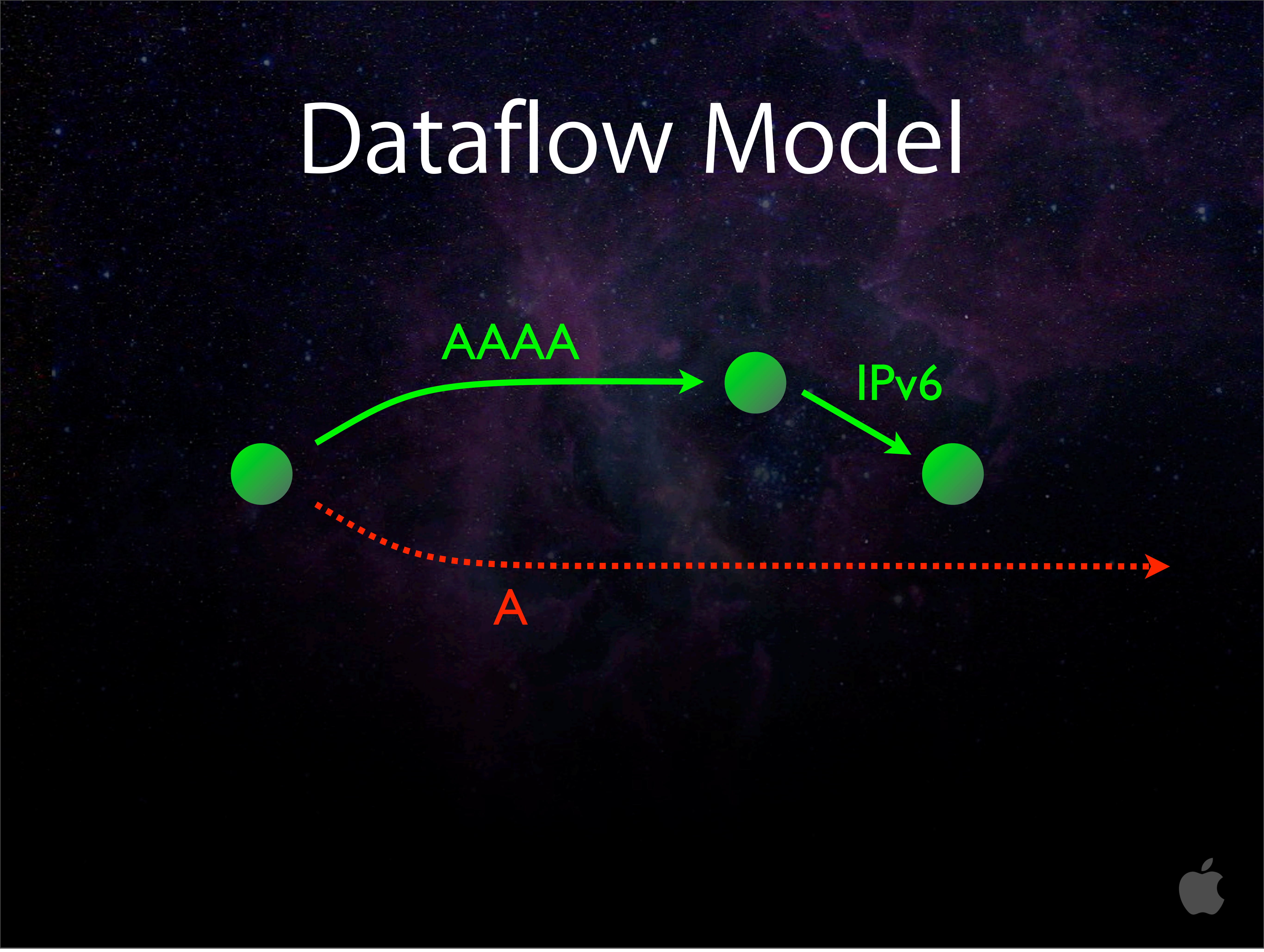
and now, if the IPv6 quad-A lookup times out, it doesn't block the IPv4 goodness

Dataflow Model



Sunday, 7 November, 2010

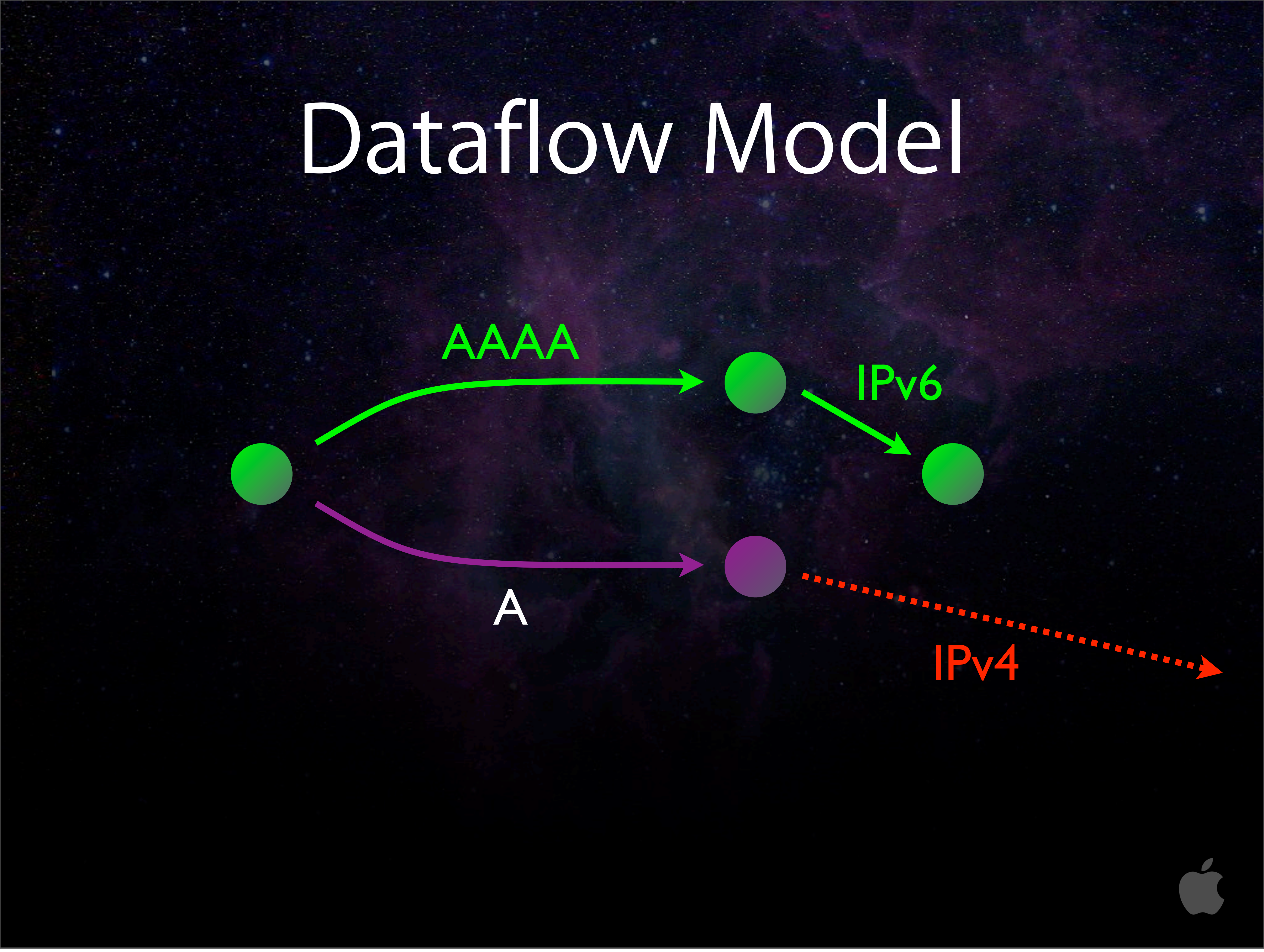
or… the DNS may succeed but the TCP connection times out — it still doesn't block IPv4



Dataflow Model

Sunday, 7 November, 2010

I don’t want to sound like I’m picking on IPv6 here, because the same is true in reverse
maybe IPv4 could fail, and IPv6 would still be able to complete in a timely fashion



Dataflow Model

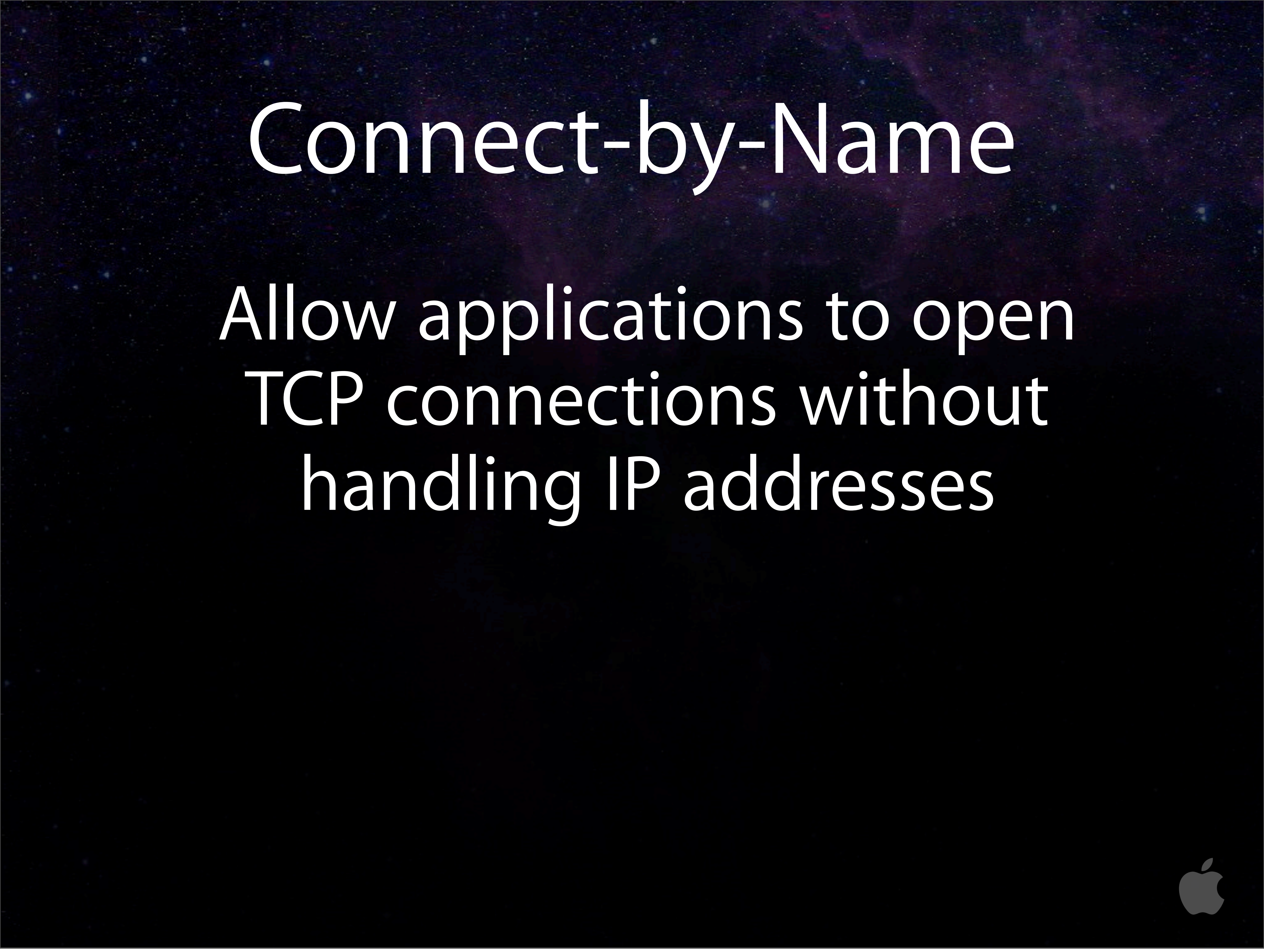
Sunday, 7 November, 2010

or maybe the DNS works but the IPv4 TCP connection times out.

The problem here is that the getaddrinfo API forces applications to be aware of IP addresses.

When you send an IP packet, the kernel ARPs for you to find the Ethernet address that goes with that IP address, and application isn't involved with that process — it's automatic and invisible to the application.

Similarly here, when an application wants to connect to a given hostname, it shouldn't have to get involved with the mechanics of how that's achieved.



Connect-by-Name

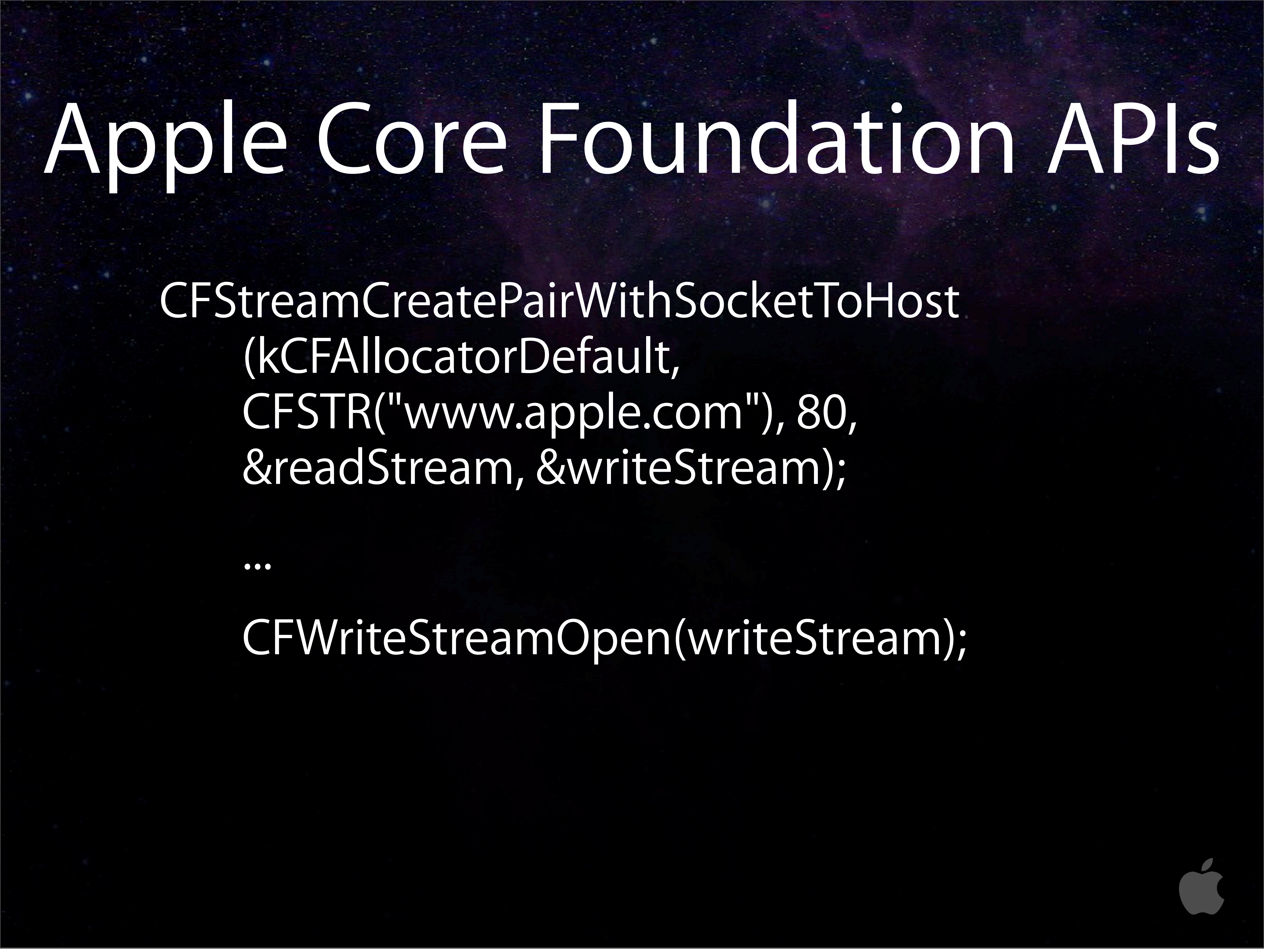
Allow applications to open TCP connections without handling IP addresses

Sunday, 7 November, 2010

The way we should do this is using what I call “connect-by-name” APIs.

Applications say, “Here’s the hostname; connect me to it.”

This is not some future goal — these APIs already exist. We just need more applications to start using them.



Apple Core Foundation APIs

```
CFStreamCreatePairWithSocketToHost
    (kCFAllocatorDefault,
    CFSTR("www.apple.com"), 80,
    &readStream, &writeStream);

    ...

    CFWriteStreamOpen(writeStream);
```

Sunday, 7 November, 2010

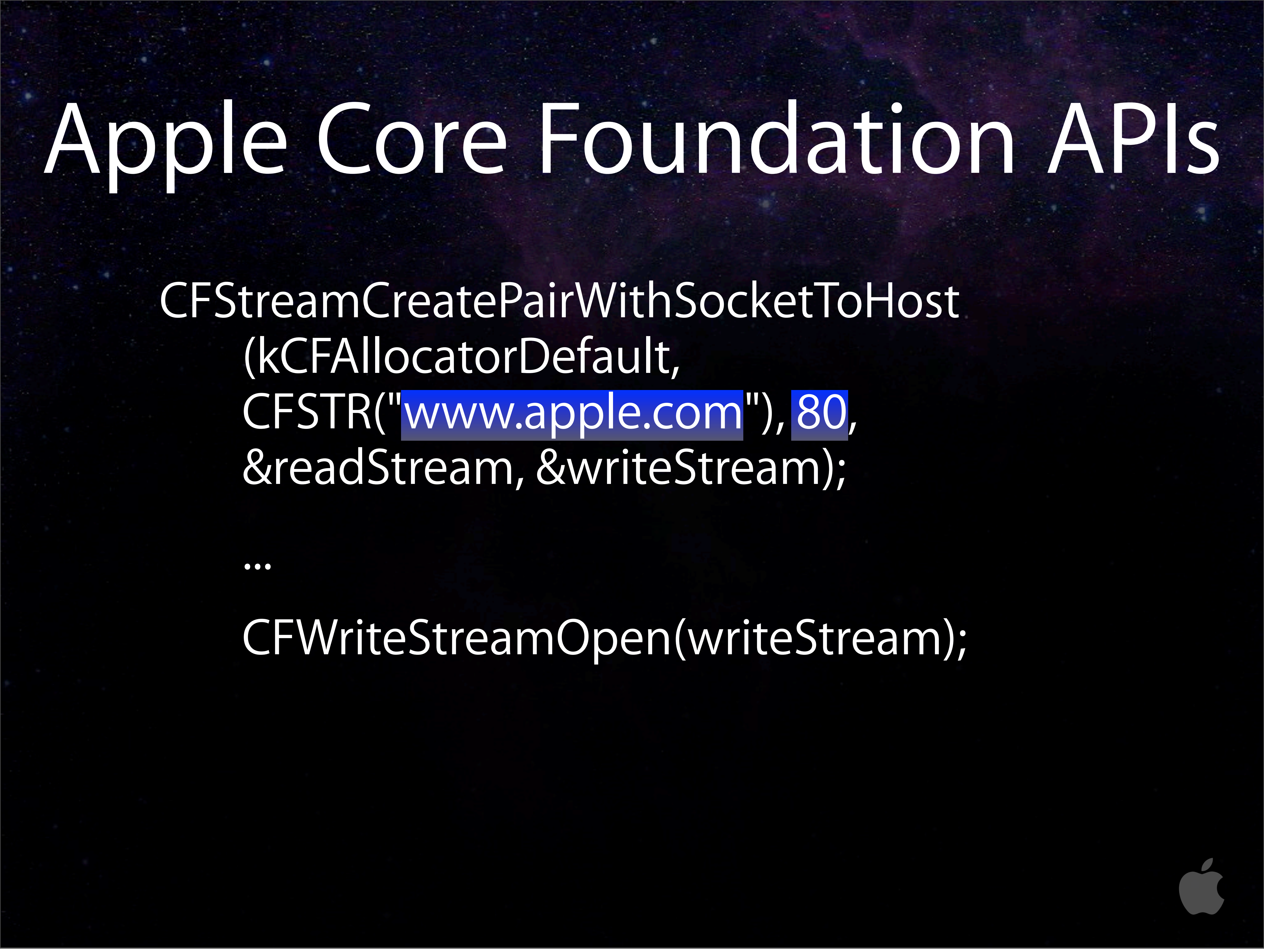
In Apple’s Core Foundation APIs, this is how you do it.

You make this opaque object called a CFStream, and you tell it the hostname and the port you want to connect to.

Then, when you open that stream, it asynchronously does whatever magic is necessary to get you a connection.

The application here never sees an address. It doesn’t know whether it’s IPv4 or IPv6.

This is not limited to Apple APIs.



Apple Core Foundation APIs

```
CFStreamCreatePairWithSocketToHost
    (kCFAllocatorDefault,
    CFSTR("www.apple.com"), 80,
    &readStream, &writeStream);

    ...

    CFWriteStreamOpen(writeStream);
```

Sunday, 7 November, 2010

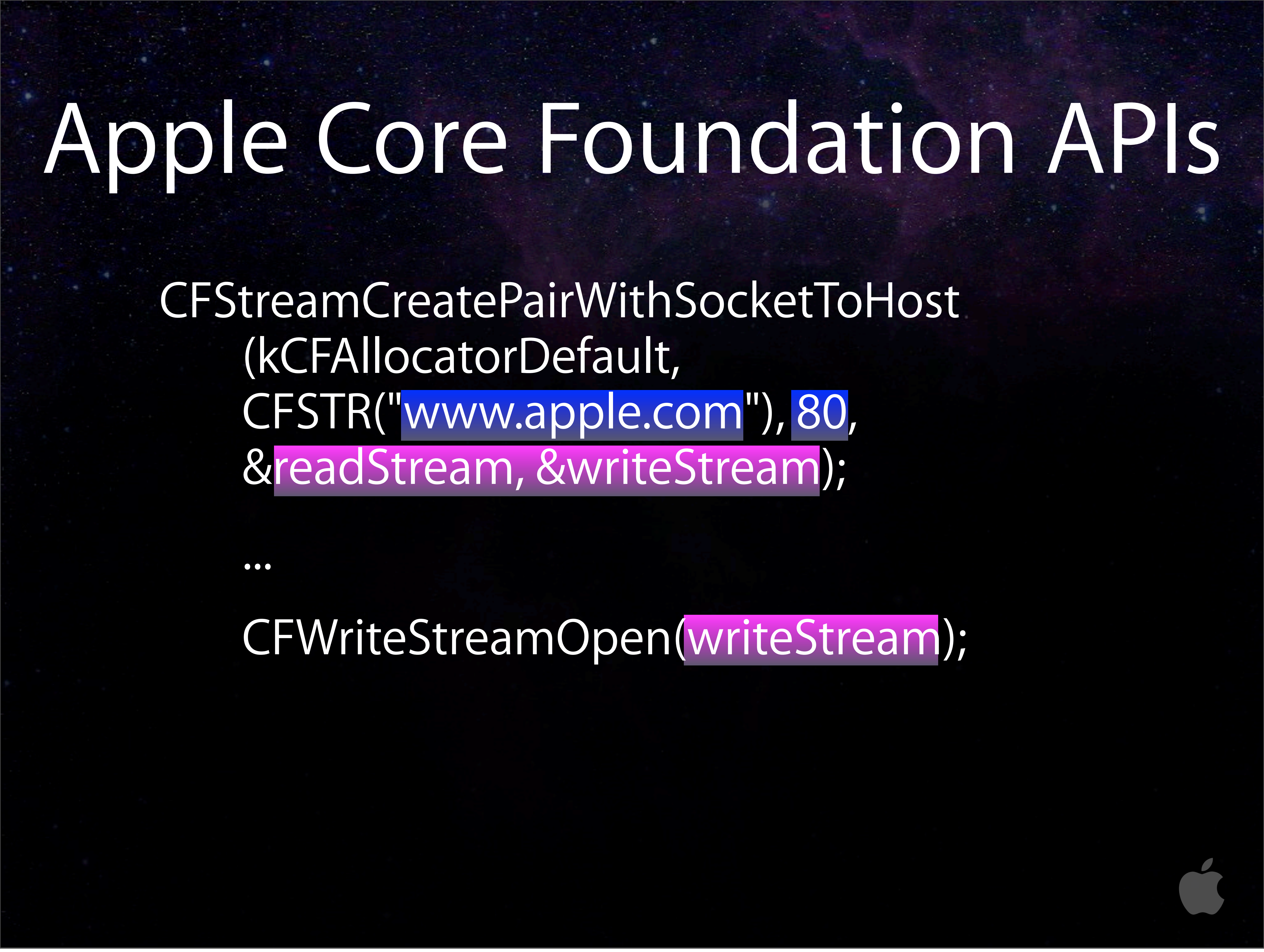
In Apple’s Core Foundation APIs, this is how you do it.

You make this opaque object called a CFStream, and you tell it the hostname and the port you want to connect to.

Then, when you open that stream, it asynchronously does whatever magic is necessary to get you a connection.

The application here never sees an address. It doesn’t know whether it’s IPv4 or IPv6.

This is not limited to Apple APIs.



Apple Core Foundation APIs

```
CFStreamCreatePairWithSocketToHost
    (kCFAllocatorDefault,
    CFSTR("www.apple.com"), 80,
    &readStream, &writeStream);

    ...

    CFWriteStreamOpen(writeStream);
```

Sunday, 7 November, 2010

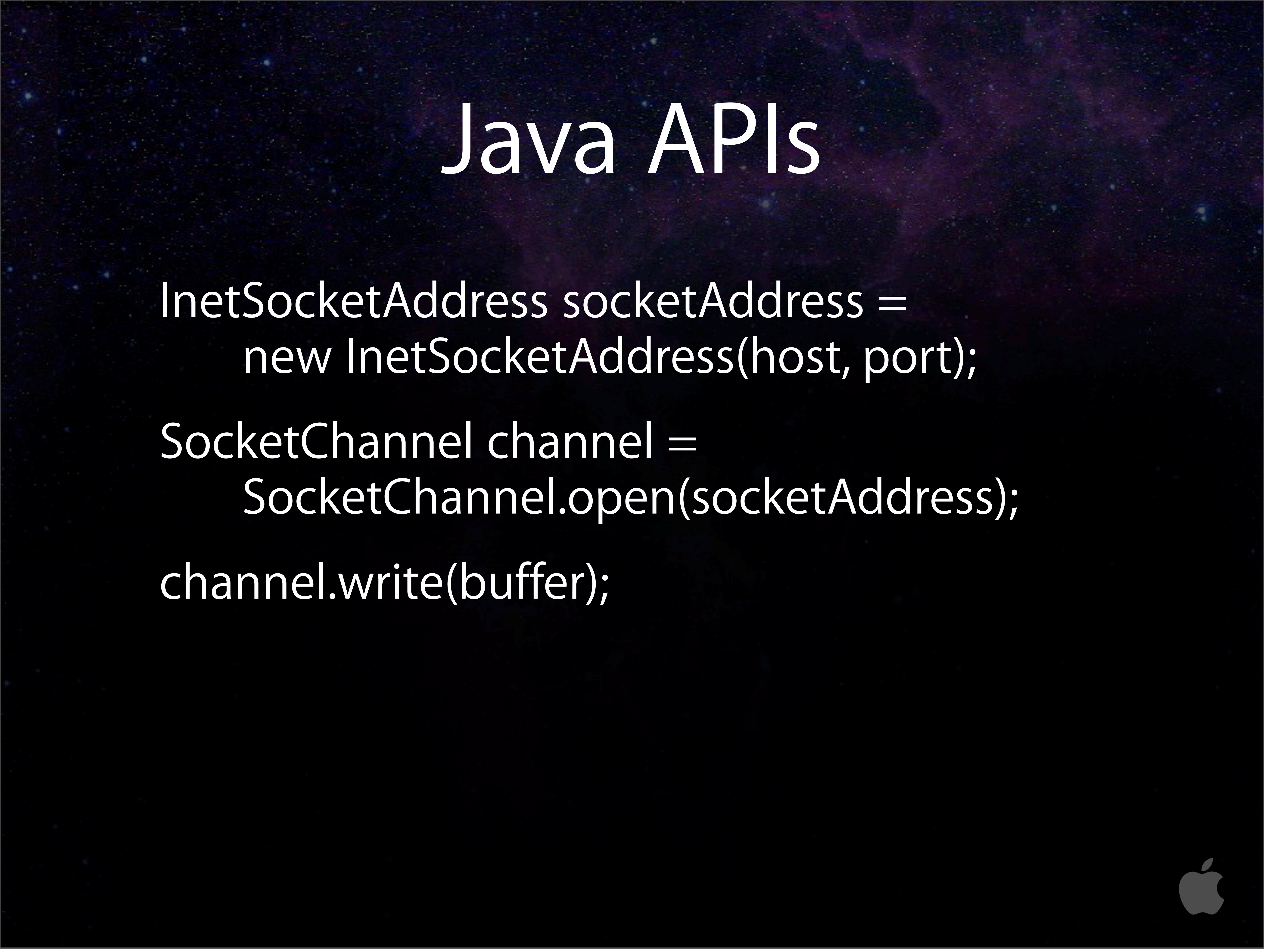
In Apple’s Core Foundation APIs, this is how you do it.

You make this opaque object called a CFStream, and you tell it the hostname and the port you want to connect to.

Then, when you open that stream, it asynchronously does whatever magic is necessary to get you a connection.

The application here never sees an address. It doesn’t know whether it’s IPv4 or IPv6.

This is not limited to Apple APIs.



Java APIs

```
InetSocketAddress socketAddress =
      new InetSocketAddress(host, port);

SocketChannel channel =
      SocketChannel.open(socketAddress);

channel.write(buffer);
```

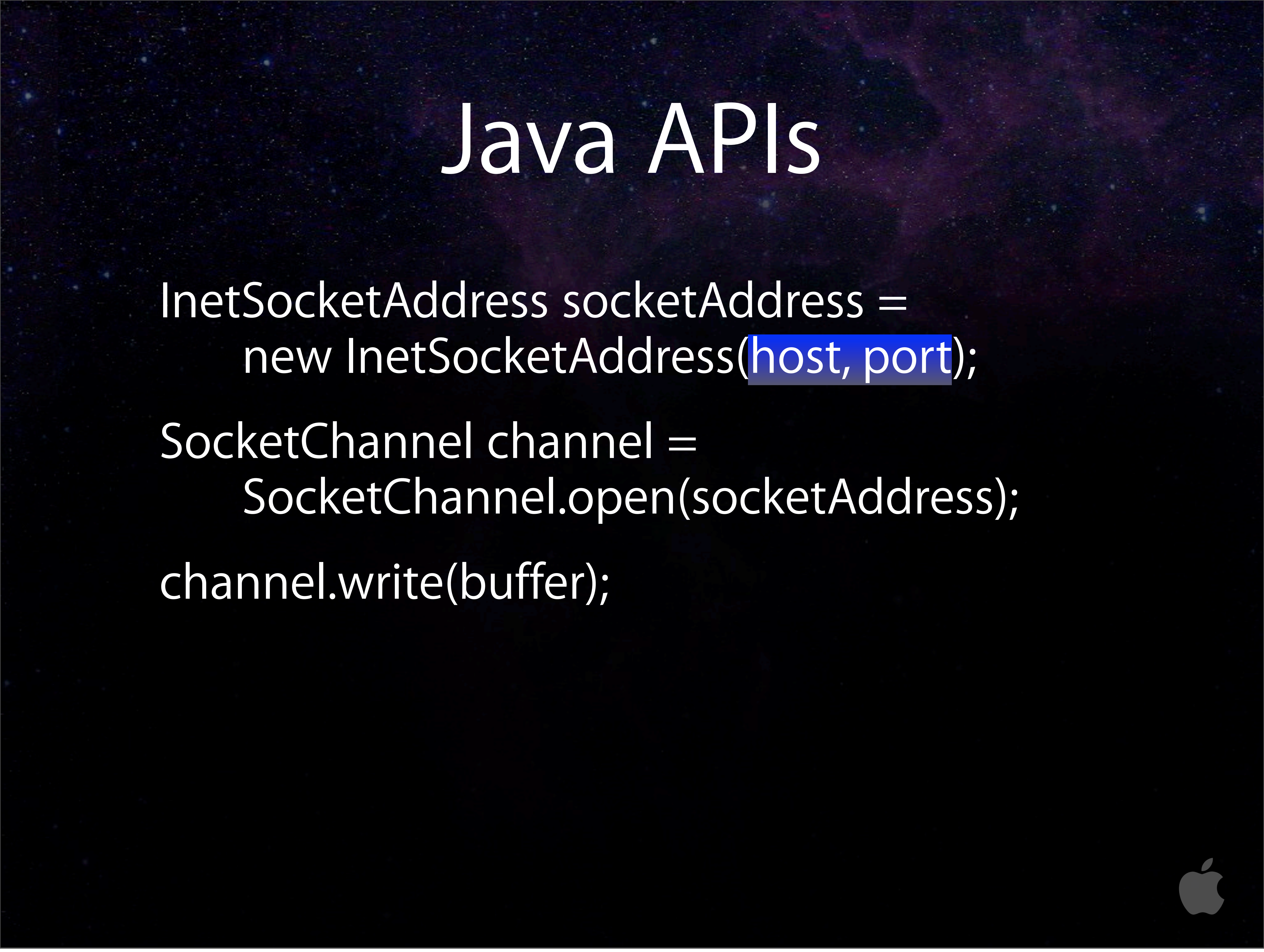
Java has a similar thing.

Here we make an opaque object called an InetSocketAddress from a host and port, and then when we open that SocketChannel, that can complete under the covers, doing whatever is necessary, without the application ever seeing an IP address.

Windows also has connect-by-name APIs. I don’t have code fragments for those here.

Now, I’m not saying that all implementations of these APIs necessarily do the right thing today, but if applications are using these APIs, then the implementations can be improved over time.

The difference with getaddrinfo() and similar APIs is that they fundamentally can’t be improved over time. The API definition is that they return you a full list of addresses, so they have to wait until they have that full list to give you. There’s no way getaddrinfo can return you a partial list and then later give you some more.



Java APIs

```
InetSocketAddress socketAddress =
      new InetSocketAddress(host, port);

SocketChannel channel =
      SocketChannel.open(socketAddress);

channel.write(buffer);
```

Sunday, 7 November, 2010

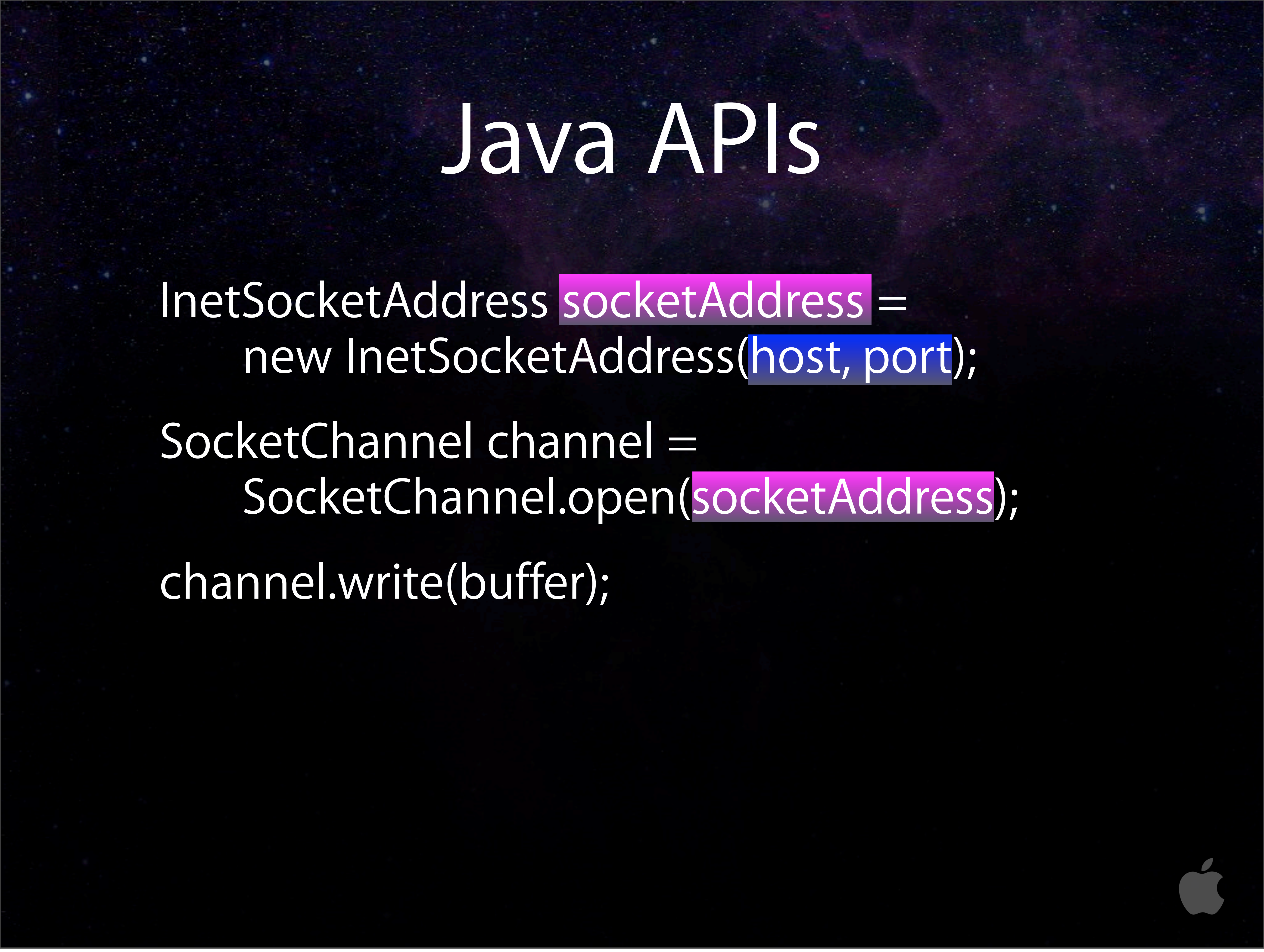
Java has a similar thing.

Here we make an opaque object called an InetSocketAddress from a host and port, and then when we open that SocketChannel, that can complete under the covers, doing whatever is necessary, without the application ever seeing an IP address.

Windows also has connect-by-name APIs. I don't have code fragments for those here.

Now, I'm not saying that all implementations of these APIs necessarily do the right thing today, but if applications are using these APIs, then the implementations can be improved over time.

The difference with getaddrinfo() and similar APIs is that they fundamentally can't be improved over time. The API definition is that they return you a full list of addresses, so they have to wait until they have that full list to give you. There's no way getaddrinfo can return you a partial list and then later give you some more.



Java APIs

```
InetSocketAddress socketAddress =
    new InetSocketAddress(host, port);

SocketChannel channel =
    SocketChannel.open(socketAddress);

channel.write(buffer);
```

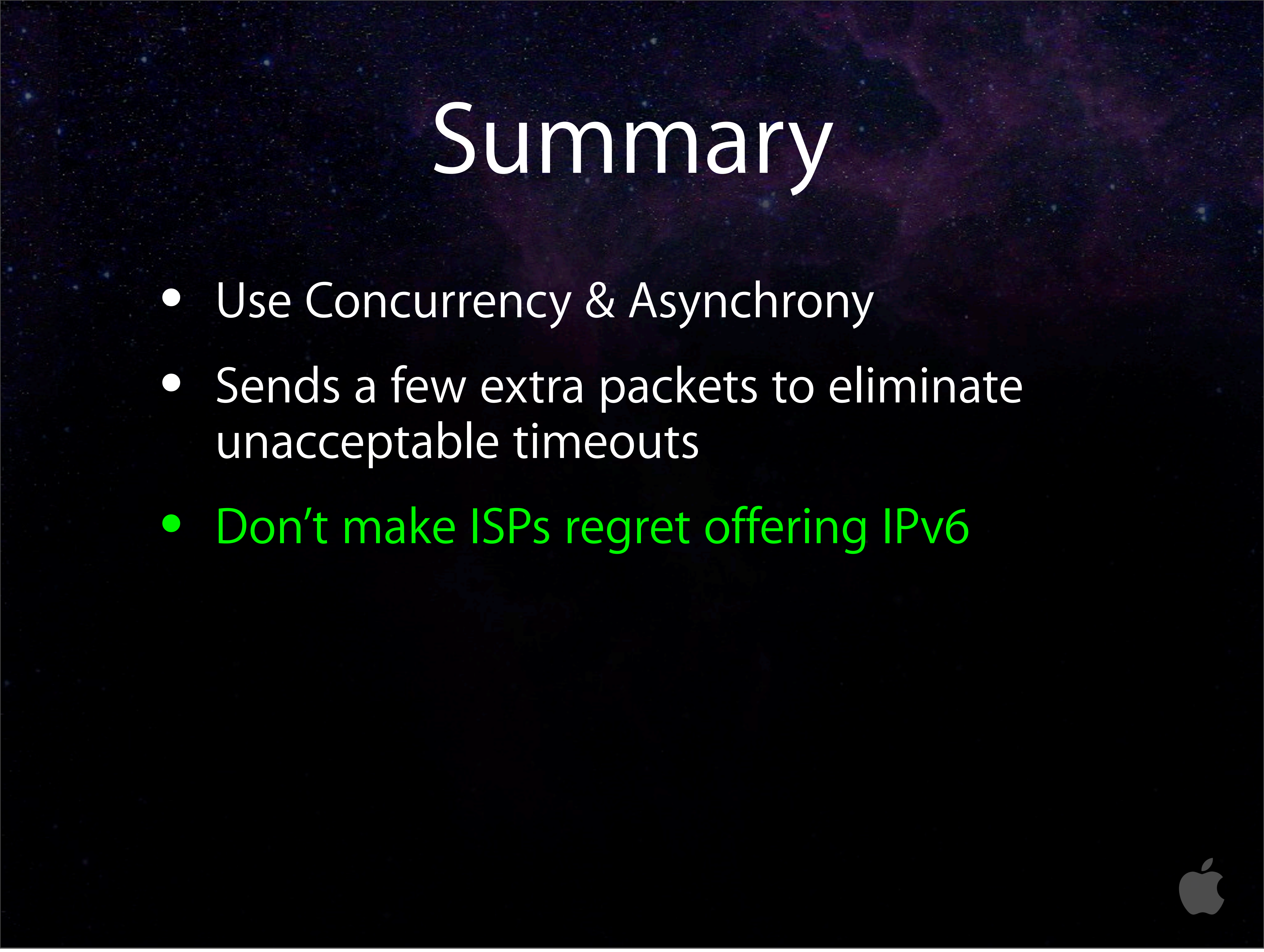
Java has a similar thing.

Here we make an opaque object called an InetSocketAddress from a host and port, and then when we open that SocketChannel, that can complete under the covers, doing whatever is necessary, without the application ever seeing an IP address.

Windows also has connect-by-name APIs. I don’t have code fragments for those here.

Now, I’m not saying that all implementations of these APIs necessarily do the right thing today, but if applications are using these APIs, then the implementations can be improved over time.

The difference with getaddrinfo() and similar APIs is that they fundamentally can’t be improved over time. The API definition is that they return you a full list of addresses, so they have to wait until they have that full list to give you. There’s no way getaddrinfo can return you a partial list and then later give you some more.



Summary

- Use Concurrency & Asynchrony
- Sends a few extra packets to eliminate unacceptable timeouts
- Don’t make ISPs regret offering IPv6

So this is the summary of my message today:
* Application programmers: Use Concurrency & Asynchrony to give your users a good user experience.
* Opening multiple connections in parallel, and then resetting the ones you don’t need, does mean that we send a few extra packets on the network, but that’s the price we pay to make an acceptable IPv6 user experience, that people will be willing to live with.
* And the high-level message is: We want ISPs to start offering IPv6 to their customers, and we don’t want them to regret doing that when they try it.