CS193E
Lecture 18

Web Kit and
Networking with
Bonjour &
Distributed Objects

Web Kit

N
NE
E
SE
S
SW
W
NW

Web Kit

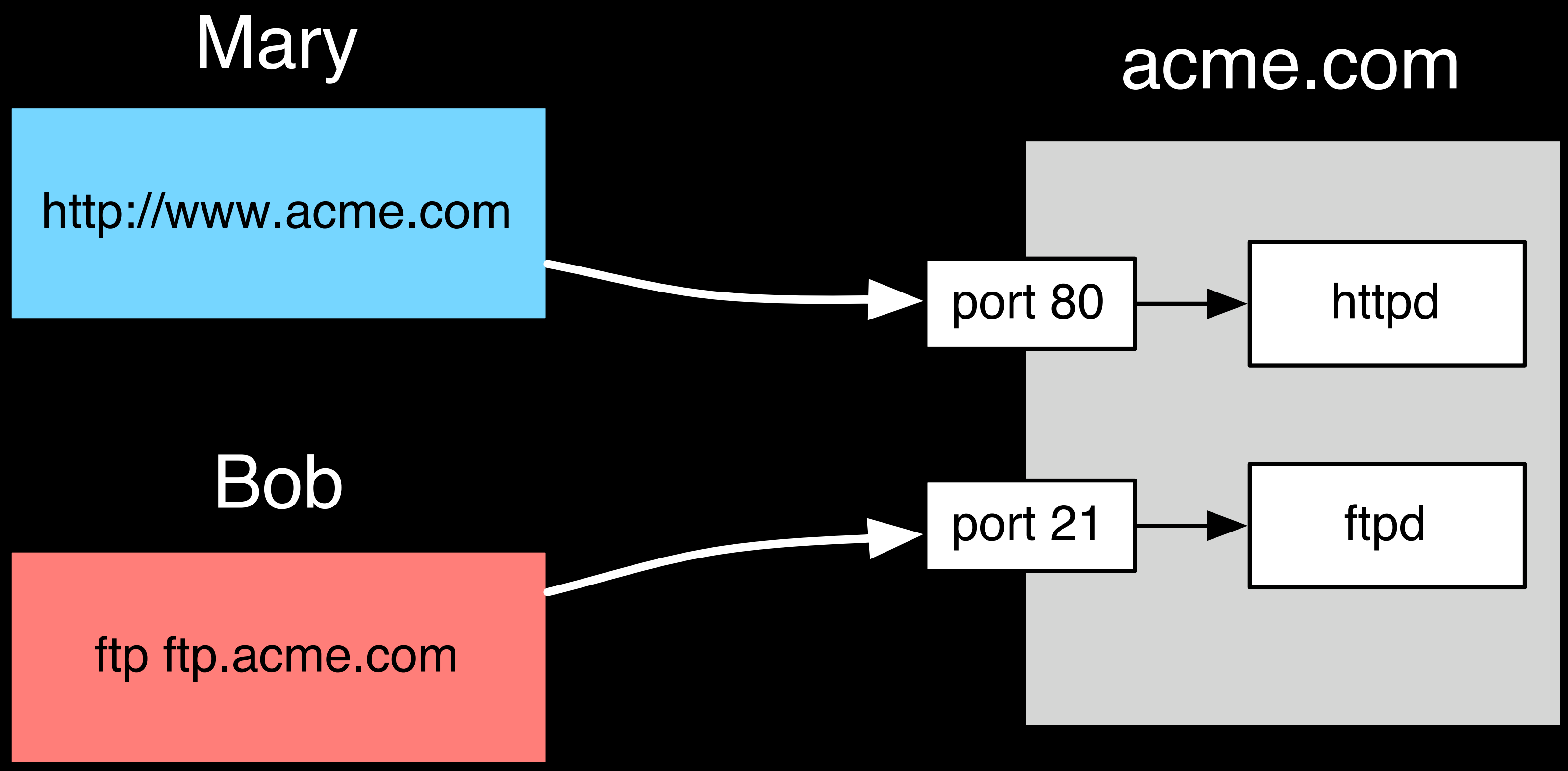
- Framework for handling web content
- Provides the core of Safari functionality
- Open Source project
  http://webkit.org/

Demo

Socket APIs

- Low level UNIX Socket APIs
  - C-based unix system calls
  - Sockets represented as file descriptors (ints)
- OS X CFSocket
  - C-based core foundation functionality
  - Sockets represented by CFSocket structs

Networking Example



Problems with Socket Programming

- You have to know the host address and port for a given service
  - DNS provides "host name" to IP address resolution
- Once a connection is established, you have very primitive means of transferring data (read and write raw data blocks)

Bonjour

- Three main functions:
    - Automate address distribution and name mapping
    - Publish availability of a service
    - Discover available services
- Open protocol Apple submitted to IETF
    - www.zeroconf.org

Bonjour

- Makes LANs self configuring
  - Requires no administration
  - Assign addresses without a DHCP server
  - Map names to addresses without a DNS server
  - Find services without a directory server

Automatic Addressing

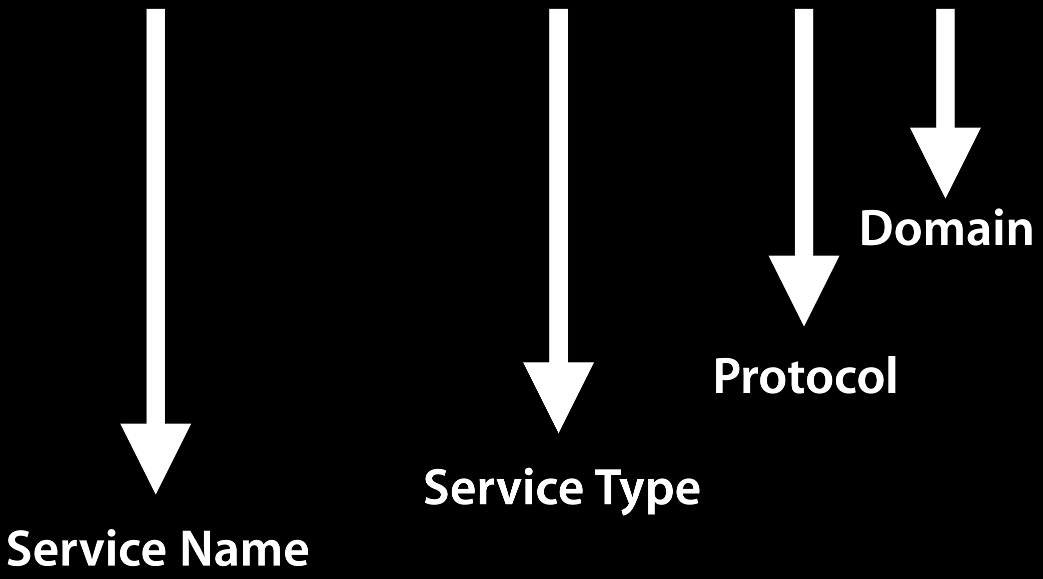
- Bonjour will pick a random address, see if it is in use
  - If it is not in use, it’s yours
  - If it is in use, try again
- Uses “.local” as a virtual domain
  - For example: powerbook.local

Advertising Services

- Applications provide a service name and port
- Follows same DNS specific-to-general model
- ServiceName._ServiceType._TransportProtocolName.
- Service Name: human readable descriptive name
- Service Type: IANA registered protocol name
- Transport Protocol Name: TCP or UDP

Service Naming

HP LaserJet 3330._printer._tcp.local.



Publishing a Service

- NSNetService is used to publish services via Bonjour

```
NSNetService *service;

service = [[NSNetService alloc] initWithDomain:@””
            type:@”_printer._tcp”
            name:@”HP LaserJet 3300”
            port:4721];

// Delegate is informed of status asynchronously
[service setDelegate:self];

[service publish];
```

Publish Delegate Methods

- NSNetService is always asynchronous (i.e. calls to publish return immediately)
- Conflict resolution handled automatically
- Status is communicated to the delegate

```
 - (void)netServiceWillPublish:(NSNetService *)sender
- (void)netServiceDidPublish:(NSNetService *)sender

- (void)netService:(NSNetService *)sender
      didNotPublish:(NSDictionary *)errorDict
```

Service Discovery

- Applications register service names with local daemon which handles responding to lookup queries
- Service discovery is completely independent of service implementation
- Resolving a service gives you an address and a port
  - The rest is up to you!

Browsing for Services

- NSNetServiceBrowser is used to search for services on the network.

```
 NSNetServiceBrowser *browser;

browser = [[NSNetServiceBrowser alloc] init];

[browser setDelegate:self];
[browser searchForServicesOfType:@"_printer._tcp."
         inDomain:@""];
```

- Note, the trailing period is required!

NSNetServiceBrowser Delegate Methods

- NSNetServiceBrowser browsing is also asynchronous
- Delegate methods called as services come and go

```
- (void)netServiceBrowserWillSearch:(NSNetServiceBrowser *)browser
- (void)netServiceBrowserDidStopSearch:(NSNetServiceBrowser *)browser

- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
             didNotSearch:(NSDictionary *)errorInfo

- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
           didFindService:(NSNetService *)service
               moreComing:(BOOL)more

- (void)netServiceBrowser:(NSNetServiceBrowser *)browser
         didRemoveService:(NSNetService *)service
               moreComing:(BOOL)more
```

Service Resolution

- NSNetServices found by NSNetServiceBrowser must have their addresses resolved before use:

```
[netService setDelegate:self];
[netService resolveWithTimeout:5];
```

- Status communicated aynschronously to delegate:

```
- (void)netService:(NSNetService *)sender
        didNotResolve:(NSDictionary *)errorDict;

- (void)netServiceDidResolveAddress:(NSNetService *)sender;
```

- Once a service has been resolved you can use the address information to connect to it

Bonjour TreeGenerator

Générateur d’arbre de Bonjour

Bonjour helps you find it, but how to you use to it?

Distributed Objects

Distributed Objects

- Allow messaging between objects in different processes/ machines as if they were local
- A server can vend any of its objects to the outside world
- A client can connect to an object in another process or machine and message it
  - Like magic!

Vending Named Objects

- If your server has an object to vend you can use NSConnection and simply vend the object by name:

```
NSConnection *connection;

connection = [NSConnection defaultConnection];
[connection setRootObject:serverObject];
[connection registerName:@"ServerObjectName"];
```

- Clients connect by doing:

```
id proxy;

proxy = [NSConnection rootProxyForConnectionWithRegisteredName:
                                   @"ServerObjectName"];
[proxy retain];
[proxy setProtocolForProxy:@protocol(ServerProtocol)];
```

Vending Multiple Objects

- If you have multiple objects to vend (as in the TreeGenerator example) you have to create different NSConnections for each object
  - For machine to machine vending, use NSSocketPort when creating connections

```
NSSocketPort *port;
NSConnection *connection;

port = [[NSSocketPort alloc] initWithTCPPort:12345];
     // Or specify 0 for the port and one will be assigned for you

connection = [[NSConnection alloc] initWithReceivePort:port
                                              sendPort:nil];
[connection setRootObject:serverObject];
```

Connecting Using Sockets

- Use NSSocketPort and NSConnection to connect to server
- Then ask connection for its rootProxy

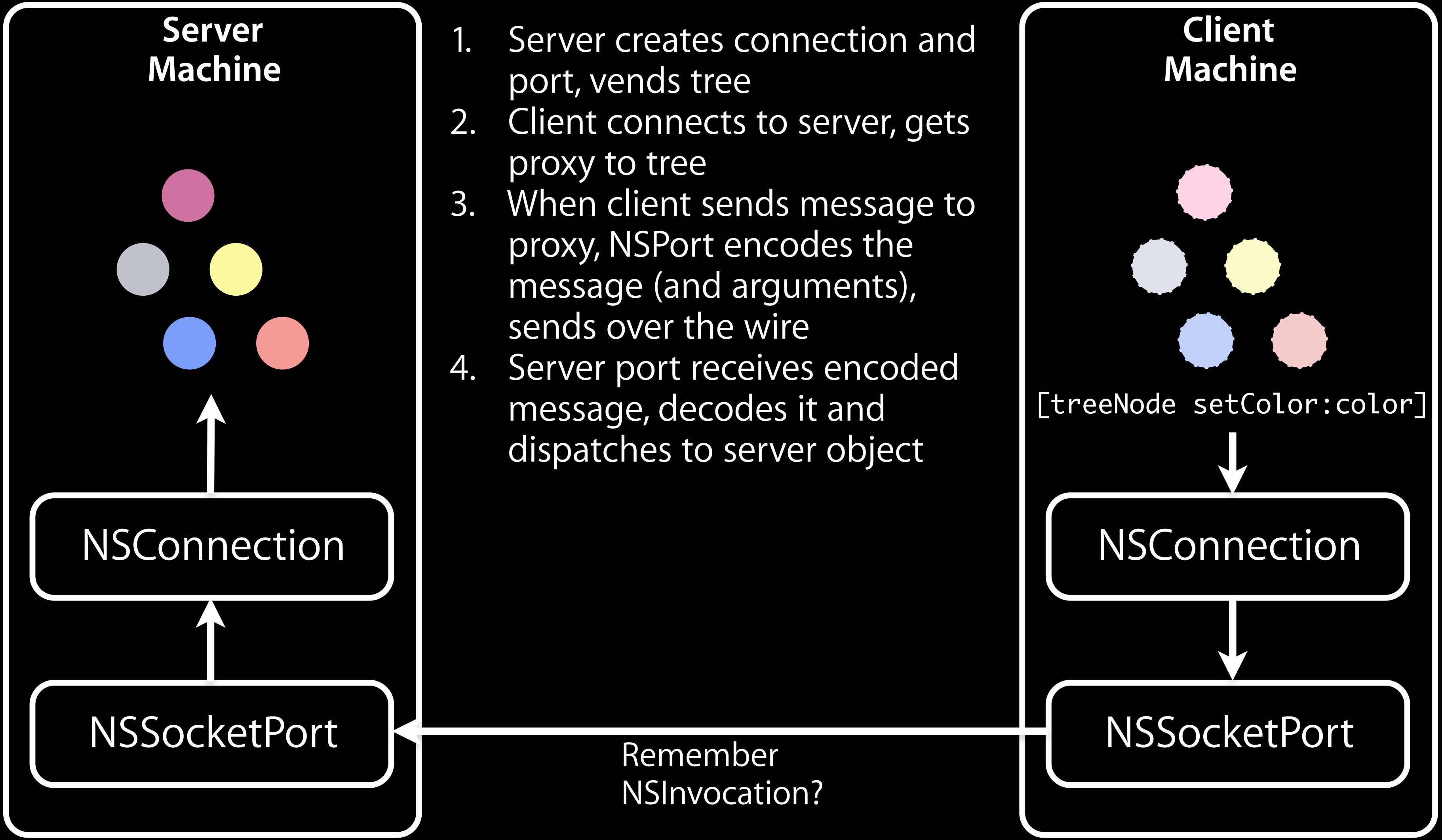
```
NSSocketPort *port;
NSConnection *connection;

port = [[NSSocketPort alloc] initRemoteWithTCPPort:12345
                                              host:@”server.foo.com”];

connection = [[NSConnection alloc] initWithReceivePort:nil
                                              sendPort:port];

// Now get the proxy object
id theProxy;
theProxy = [connection rootProxy];
```

Messaging Remote Objects



1. Server creates connection and port, vends tree
2. Client connects to server, gets proxy to tree
3. When client sends message to proxy, NSPort encodes the message (and arguments), sends over the wire
4. Server port receives encoded message, decodes it and dispatches to server object

Arguments & Return Values

- When a method returns an object, a proxy is returned

```
// Now get the proxy object
id remoteTree;

remoteTree = [connection rootProxy];
[remoteTree setProtocolForProxy:@protocol(TreeProtocol)];

TreeNode *treeNode = [remoteTree rootNode];
```

- The treeNode returned is actually a proxy to the remote TreeNode object on the server
- Similar behavior for arguments to methods

DO Optimizations

- -[NSDistantObject setProtocolForProxy:]
  - Optimizes introspection chatter
- Objective-C protocol method decls can have DO specific keywords in them:

```
- (oneway void)process:(bycopy Shape *)shape;
```

- oneway
  - Sends message but returns immediately
- bycopy
  - Copy the object rather than send a proxy

DO Robustness

- Timeouts
  - [NSConnection setRequest/ReplyTimeout:]
  - NSPortTimeoutException
- NSConnectionDidDieNotification
  - Not posted for NSSocketPorts
- [NSConnection setIndependentConversationQueueing:YES]

Questions?