

## ***Analysis 3 / CGI***

---

### Recall

Network Effect / Inertia  
Standards -> Competition

Q: Where did the Internet Come from?  
What is its growth engine?

A: Standards

**Any-Any Synergies**

Value is unlocked by the any-to-any connections enabled by a freely implementable standards.

Q: What Makes Standards Work?

e.g. SMTP and POP are nothing special -- how is it that they unlocked so much value so quickly and cheaply?

A: Voluntary Participation

The value is unlocked by the voluntary participation.

**It's the participants, stupid**

Without the voluntary participants, the system is worth nothing. As a result, the future depends on standards and incentives for participation more than technical merit.

## Q: What's so special about standards?

A:

Nobody knows for sure why standards work out so well. Here are some speculations

### **1. 100% much better than 90%**

Maybe having 100% compatibility is much better than the 90% that a proprietary, e.g., MS, standard can reach. Maybe the 100% unlocks any-synergies.

MS can never reach 100% because they are all about setting up tech tying (or we could say "compatibility") between products they control and leaving out systems they don't control -- this always leaves them at 90% at most. Perhaps leaving out the last 10% hurts disproportionately.

### **2. Ego vs. proprietary standard**

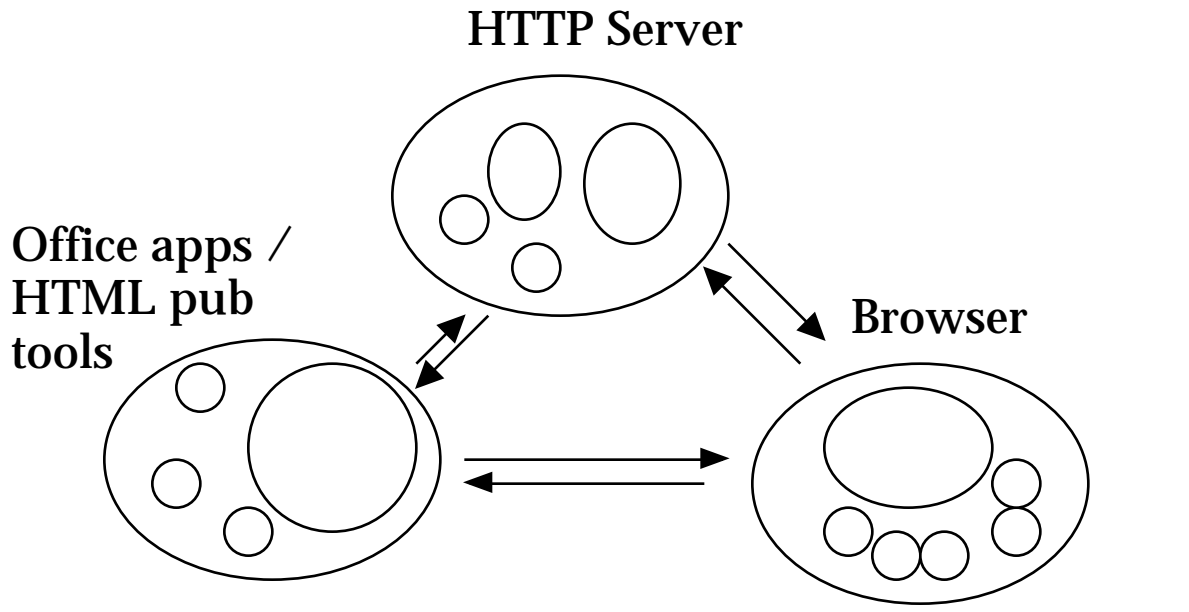
Maybe it's just unappealing to participate in a standard owned by someone else, so the critical mass of participation never quite blossoms. A public, free standard like POP is more appealing to the participants since you're not obviously feathering some else's bed. Theme: incentives for participation matter more than technical merit

### **3. Competition / Darwinism**

The standard makes implementations replaceable which leads to competition. One of the results over time is that the implementation price/quality gets better and better.

## Standards / Competition Picture

Compatible domains. Standards allow replaceability, so within each domain there is competition -- remember this picture



## Why didn't MS invent the Internet?

MS word has over 80% of the word-processing market share. Likewise, MS networking technology. Why didn't "The Internet" grow up around MS document interchange?

**A:**

See above -- apparently free standards pull in participants in a way that the MS standard doesn't quite.

MS is all about setting up synergies between its products and leaving out the products of others. Create synergies between MS products without helping or encouraging competition. From a sufficient position of strength, this gets the Network Effect working for MS's interests. All vendors would love to use this strategy given the chance.

# What Would Bill Gates Say?

## 1. Standards Are Bad

Standards have slow, designed-by-committee feel  
They must be dumbed down to the lowest common denominator

### Response

That's all true, but technical merit was not the point.  
Participation and standards-driven competition end up mattering more so long as the standard is of at least reasonable quality.

## 2. IE won on its merits

IE has more market share because it is better than Netscape.

### Response

IE is a high quality implementation. IE is dominating with a combination of tech merit and MS tying strategy. The tying is what's been ruled illegal. MS has also done every sort of technical and marketing tying/bundling imaginable to channel inertia and Network Effect MSs way as much as possible.

## 3. Merit Beats Inertia

MS products have 90% market share because they are better than the competition.

### Response

Since the DOS days, MS well understands the role of inertia and network effects in product success. Gates was the first to really understand and exploit the dynamics of a high-inertia marketplace.  
MS produces pretty good products, but the overall MS domination is a testament to proprietary network effect.  
If MS thinks its a merit-driven world, why are they so hot to get IE up to 90% market share as soon as possible? It's because they realize that whoever dominates a niche first, dominates it forever because the network effect dominates tech merit.

## Server Side HTTP

HTTP server sits there

request comes in

Typically HTTP server maps that request into its file system

The server decides what the MIME type is -- probably based on its file extension.

This "file system" model is sometimes called "static pages"

## Alternative Server Side - Dynamic

Suppose the server is just a database of classes "cs106, cs107, ..."

When a request comes in like "/class/cs106"...

1) Look up that row in the database

2) Dynamically generate a page containing that row's data and send it back as a reply

The client just specifies a request string such as "/" or "/class/cs106"

How the server interprets the request is ultimately up to the server

## CGI

Common Gateway Interface -- a standard which interfaces the HTTP server software with CGI programs which run on the server.

The CGI standard defines how the server communicates all the various facts about the requesting URL (date, referrer, accept) to the CGI program, and how the CGI program's output should be handled and sent back to the requesting client.

CGI is language independent: C, Perl, Unix shell script.

## Security

Security concerns -- because the CGI's run on the server, they need to take care not to allow a rogue client to compromise the server either by hurting the server or serving up information which was supposed to be secret.

## CGI Dialog

### 1. URL

The client has a URL as usual. However, the URL identifies not a page of HTML, but a CGI program on the server.

e.g. <http://www-cs-faculty.stanford.edu/cgi-bin/nick/hello.pl>

### 2. Client GET

The client does a GET request as usual

### 3. Server -> CGI

The server looks at the GET request, realizes its a CGI.

### 4. Run CGI

Server runs the CGI program, feeding it the GET request as input. The GET information is fed to the CGI through environment variables.

## 5. CGI program

The CGI looks at the input and computes what it wants to compute.

## 6. HTTP Header

First, the CGI prints the HTTP header fields it wants, followed by a blank line.

## 7. HTML content

Then the CGI prints the body content it wants and then exits

## 8. HTTP Server

The HTTP server gathers the output of the CGI, adds more fields to the header, and sends it along to the client.

# Trivial Example — `hello.pl`

```
#!/usr/bin/perl -w

## Hello.pl -- demonstrate a trivial CGI that prints
## out some HTML and the current time on this server.

use strict 'vars';

my($EOL) = "\015\012";

## This is a human-readable str of the current time
my($nowStr);
$nowStr = localtime();

## This line must be included in the header
print "Content-type: text/html$EOL$EOL";

## Write out the HTML content
print "<html><head><title>Hello.pl</title></head>\n";
print "<body>\n";
print "<h1>Hello.pl</h1>\n";
print "Hello there from CGI-land. It's currently '$nowStr'\n";
print "</body></html>\n";
```



## hello.pl CGI Process

This CGI doesn't look at any input, it just produces an HTTP header followed by the same HTML body every time.

For its HTTP header, it just writes the "content-type: text/html" line followed by a blank line. The HTTP server fixes up the rest of the header on its way out to the client.

## Input to CGI programs

Most likely, the CGI runs as a user called "WWW" or "CGI" which has limited read/write permission on the system. This helps limit some of the security danger, even if the CGI is poorly written.

Before the CGI is run, its environment variables are set to communicate the request — the primitive Unix method for programs to communicate with each other.

The QUERY\_STRING environment variable will be the text after the ? in the URL. There are many other environment variables set to indicate other things about the request -- see the DumpEnv example below.

## Shell Script CGI

Shell script is a simple Unix programming facility which pre-dates the popularity of Perl. Environment variables are available by prefixing their name with a \$.

Shell scripts are a poor choice for CGI programming because they are easy to write in a way which creates a security hole -- allowing the client to run an arbitrary expression such as "rm -rf \*" on the server.

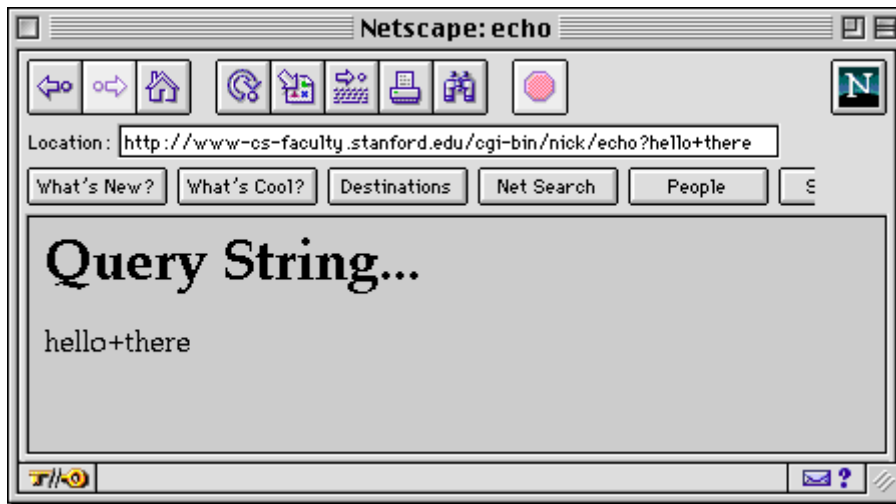
## Echo Query Shell Script

```
#!/bin/sh
# echo cgi script
echo "Content-type: text/html"
```

```
echo
echo "<html><body>"
echo "<h1>Query String...</h1>"
echo "$QUERY_STRING"
echo "</body></html>"
echo
# Uncomment the following line for the definition of a security hole...
# $QUERY_STRING
```

## Echo Query Example

Invoke: <http://www-cs-faculty.Stanford.EDU/cgi-bin/nick/echo?hello+there>



Usually, the query string encodes bindings with the "form" syntax:  
"n=10&pi=3.14"

# DumpEnv Perl Script

```
#!/usr/bin/perl
# Print out the values of all the environment variables
# in an HTML <ul>.
# Call from the shell or invoke as a CGI script.

# HTTP header section
print "content-type: text/html\r\n\r\n";

$header = <<EOT;
<html>
<head><title>DumpEnv</title></head>
<body>
EOT

$trailer = <<EOT;
</body>
</html>
EOT

# Emit an HTML <ul> for all the environment vars
# set up for the CGI
print $header

print "<ul>\n";
foreach $key (keys %ENV) {
    print "<li><b>$key</b> = $ENV{$key}\n";
}
print "</ul>\n";
print $trailer;
```

# Environment Vars

- Query\_String (after the ? in the URL)
- Script\_Name (eg /cgi-bin/nick/dumpenv.pl)
- Request\_Method (usually GET)
- Path\_Info (extra path after the name of the CGI in the URL)
- Remote\_Host -- The client's IP or DNS address -- from a privacy point of view, this is one thing the server will definitely know -- how else will it send the packets back to you but by knowing your IP addr?

# DumpEnv Output w/ "..?pi=3.14"

