

YAHOO!

CROCKFORD ON JAVASCRIPT

DOUGLAS CROCKFORD, JAVASCRIPT ARCHITECT, YAHOO! INC.



YAHOO!

CROCKFORD ON JAVASCRIPT

DOUGLAS CROCKFORD, JAVASCRIPT ARCHITECT, YAHOO! INC.

Scene 6

Loopage

loop {

...

}

```
while (<condition>) {  
    <body>  
}
```

```
loop {  
    if (!(<condition>)) {  
        break;  
    }  
    <body>  
}
```

```
do {  
    <body>  
} while (<condition>);
```

```
loop {  
    <body>  
    if (!( <condition> )) {  
        break;  
    }  
}
```

```
for (<initialize>; <condition>; <increment>) {  
    <body>  
}
```

```
<initialize>;  
loop {  
    if (!(<condition>)) {  
        break;  
    }  
    <body>  
    <increment>;  
}
```

```
while (<condition>) {  
    <body>  
}
```

```
(function whiler() {  
    if (<condition>) {  
        <body>  
        return whiler();  
    }  
})();
```

Browser Event Loop

- **Event queue containing callback functions.**
(timer, ui, network)
- **Turn: Remove one callback from the queue. It runs to completion.**
- **Prime Directive: Never block. Never wait. Finish fast.**
- **The Event Loop is one of the best parts of the browser.**
- **Avoid: alert, confirm, prompt, XHR synchronous.**

```
var do_default = true;
event.bubbles = true;
event.preventDefault = function () {
    do_default = false;
};
event.stopPropagation = function () {
    event.bubbles = false;
};
event.currentTarget = event.target;
do {
    func = event.currentTarget['on' + event.type];
    if (typeof func === 'function') {
        func.call(event.currentTarget , event);
    } else if (typeof func === 'string') {
        new Function(func).apply(event.currentTarget);
    }
    event.currentTarget = event.currentTarget.parent;
} while (event.currentTarget && event.bubbles);
if (do_default) {
    event.target.default_action();
}
```



Courtesy WGBH

IBM Automatic Sequence Controlled Calculator





A-0

- **The first compiler.**
- **Compiled a set of subroutines from a tape library.**
- **Each subroutine had a call number.**
- **Subroutines can be open (include) or closed (function).**
- **In modern terms, it was more of a linker or loader.**

A-2

- **The first open source project.**
- **It was shared with people who bought Univac machines, and they contributed changes and suggestions.**
- **Lead to A-3, AT-3, and MATH-MATIC.**
- **IBM's FORTRAN.**

- **Some of the best programmers of the day rejected this approach.**

**Programming took too much
creativity and dexterity for the
human being to be replaced by
the very machine we are
manipulating. Waa.**

READ / WRITE

**Stop until the I/O operation is
completed.**

Black Box



Timesharing

- **Interactive programs are highly modal.**
- **The user can only enter what the program is expecting.**
- **The flow of the dialog is dictated by the program, not by the user.**

- **Timesharing depended on blocking READ.**
- **While one program is blocked, another may run.**

**Virtually all programming
languages have blocking I/O.**

READ / WRITE

unto every generation.

**Except C, which has
no I/O at all.**

**Unfortunately, it has `stdio`,
and `stdio` blocks.**

Meanwhile

Research and games.



**You have to write the
programs inside out! Waa!**

Let's go back to the command line.



Welcome to HyperCard

HyperCard is a unique software tool that allows you to do more with your computer.

With HyperCard, you use documents called **stacks**. Stacks can help you do many different things—for example, you could use a stack to keep track of your appointments, manage your expenses, learn a new language, or record and play sounds.

To learn more, click "What is HyperCard?" to the right.

©1993-1995 Apple Computer, Inc. All Rights Reserved.

Home



What is HyperCard?



Addresses



Audio Help

**Non-programmers were
incredibly productive with
HyperCard.**

**Suddenly, professional
programmers got smarter and
there was an explosion of Mac and
Windows applications.**

HyperCard was all about events

- Programs (aka stacks) were written as a collection of event handlers attached to visible objects.
- Events bubble up.
 - on mouseUp
 - on keyDown
 - on cardEnter
 - on idle
- If you don't provide a right way, the street will find its own way.

**HyperCard had a big impact
on the evolution of the
browser.**

JavaScript is well suited for this model.

**As awful as the DOM is,
JavaScript+DOM is effective.**

JavaScript+YUI3 is really effective.

**JavaScript does not have
READ.**

**That has always been seen as a
huge disadvantage, but it is
actually a wonderful thing.**

**READ blocks, and blocking is
bad for event loops.**

**JavaScript programmers are
smarter about using event loops
than programmers of other
languages.**

**Event loop is just one approach
to concurrency.**

**The most popular approach is
threading.**

Threading

Pro

- No rethinking is necessary.
- Blocking programs are ok.
- Execution continues as long as any thread is not blocked.

Con

- Stack memory per thread.
- If two threads use the same memory, a race *may* occur.
- To be continued...

Two threads

```
1. my_array[my_array.length] = 'a';
```

```
2. my_array[my_array.length] = 'b';
```

- ['a', 'b']
- ['b', 'a']

Two threads

```
1. my_array[my_array.length] = 'a';
```

```
2. my_array[my_array.length] = 'b';
```

- ['a', 'b']
- ['b', 'a']
- ['a']
- ['b']

```
my_array[my_array.length] = 'a';  
  
length_a = my_array.length;  
my_array[length_a] = 'a';  
if (length_a >= my_array.length) {  
    my_array.length = length_a + 1;  
}
```

```
my_array[my_array.length] = 'a';

length_a = my_array.length;
length_b = my_array.length;
my_array[length_a] = 'a';
if (length_a >= my_array.length) {
my_array[length_b] = 'b';
    my_array.length = length_a + 1;
}
if (length_b >= my_array.length) {
    my_array.length = length_b + 1;
}
```

**It is impossible to have
application integrity when
subject to race conditions.**

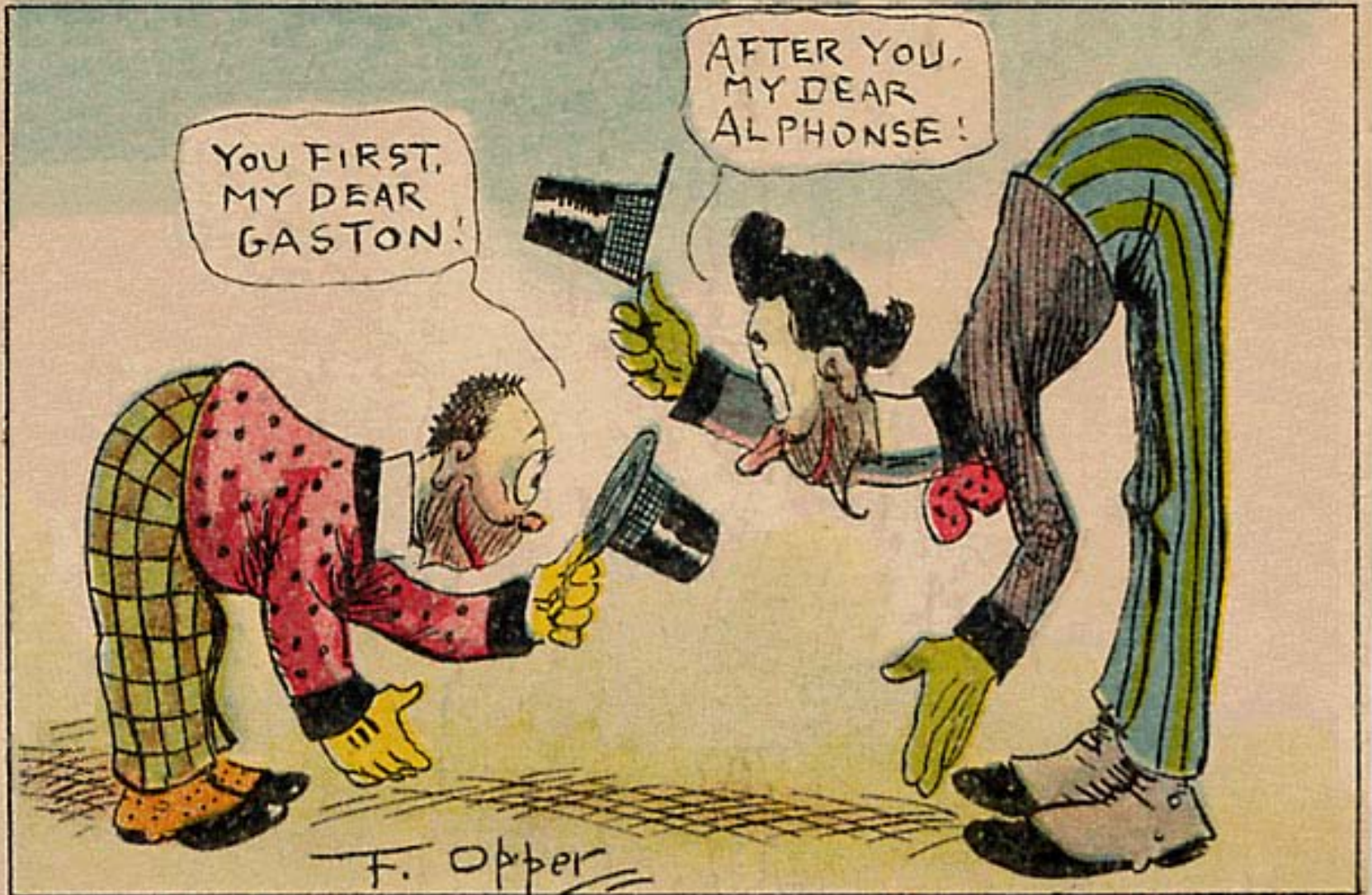
Mutual Exclusion

- semaphore
 - monitor
 - rendezvous
 - synchronization
-
- This used to be operating system stuff.
 - It has leaked into applications because of networking and the multi-core problem.

Mutual Exclusion

- **Only one thread can be executing on a critical section at a time.**
- **All other threads wanting to execute the critical section are blocked.**
- **If threads don't interact, then the program runs at full speed.**
- **If they do interact, then races will occur unless mutual exclusion is employed.**

Deadlock



COPYRIGHT, 1906, BY AMERICAN JOURNAL EXAMINER.

"YOU FIRST, MY DEAR."

Deadlock

- **Deadlock occurs when threads are waiting on each other.**
- **Races and deadlocks are difficult to reason about.**
- **They are the most difficult problems to identify, debug and correct.**
- **They are often unobservable during testing.**
- **Managing sequential logic is hard.
Managing temporal logic is really, really hard.**

Threading

Pro

- No rethinking is necessary.
- Blocking programs are ok.
- Execution continues as long as any thread is not blocked.

Con

- Stack memory per thread.
- If two threads use the same memory, a race *may* occur.
- Overhead.
- Deadlock.
- Thinking about reliability is extremely difficult.
- System/Application confusion.

**Fortunately, there is a model
that completely avoids all of
the reliability hazards of
threads.**

The Event Loop!

Event Loop

Pro

- Completely free of races and deadlocks.
- Only one stack.
- Very low overhead.
- Resilient. If a turn fails, the program can still go on.

Con

- Programs must never block.
- Programs are inside out!
Waa!
- Turns must finish quickly.

Long running tasks

- **Two solutions for long running programs:**
- **Eteration: Break the task into multiple turns.**
- **Move the task into a separate process (workers).**

Remote Procedure Call

- Combines two great ideas, functions and networking, producing a really bad idea.
- Like READ, attempts to isolate programs from time. The program blacks out.
- In reading the program, it is by design difficult to see where time is lost.
- This can result in a terrible experience for the user. Lost time === annoying delays.
- Keeping the user waiting without warning is disrespectful and rude.

Latency Compensation

- **At a minimum, acknowledge user's input immediately.**
- **Don't lock up the interaction while waiting for the server's response.**
- **In some applications, it is reasonable to predict the server's response and display it immediately. Later display a correction if the prediction was wrong.**

Security

XSS

XSS has two causes:

- 1. Sharing of the global object.**
- 2. Misinterpretation of HTML...**

**Tragically, HTML5 ignores
and worsens the XSS problem.**

**“...HTML doesn't ever have
markup injection
vulnerabilities...”**

<http://lists.w3.org/Archives/Public/public-webapps/2010AprJun/0648.html>

**The browser is a loaded gun
pointed at your head.**

This pulls the trigger:

<?= "bang" ?>

Page Templates

- **The page template systems (PHP, ASP, JSP...) are not a good match for the way we build modern sites.**
- **A template is too rigid a framework.**
- **It is too easy to insert text into a context where it can be misinterpreted and executed, completing an XSS attack.**

**Can we do better by using
JavaScript on the server?**

There are some obvious advantages:

**We can take advantage of our new
understanding of JavaScript.**

Server Side JavaScript

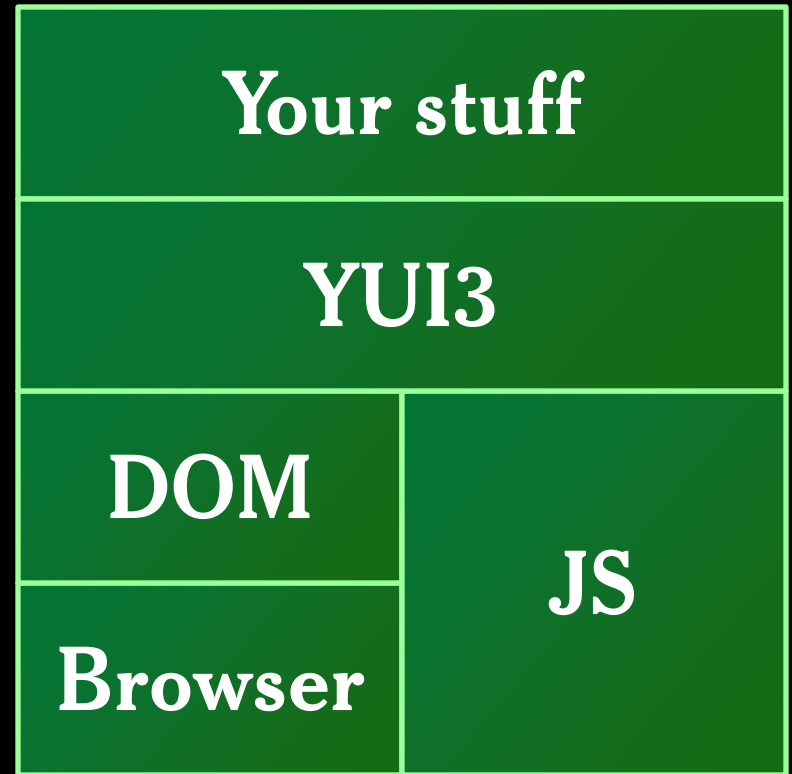
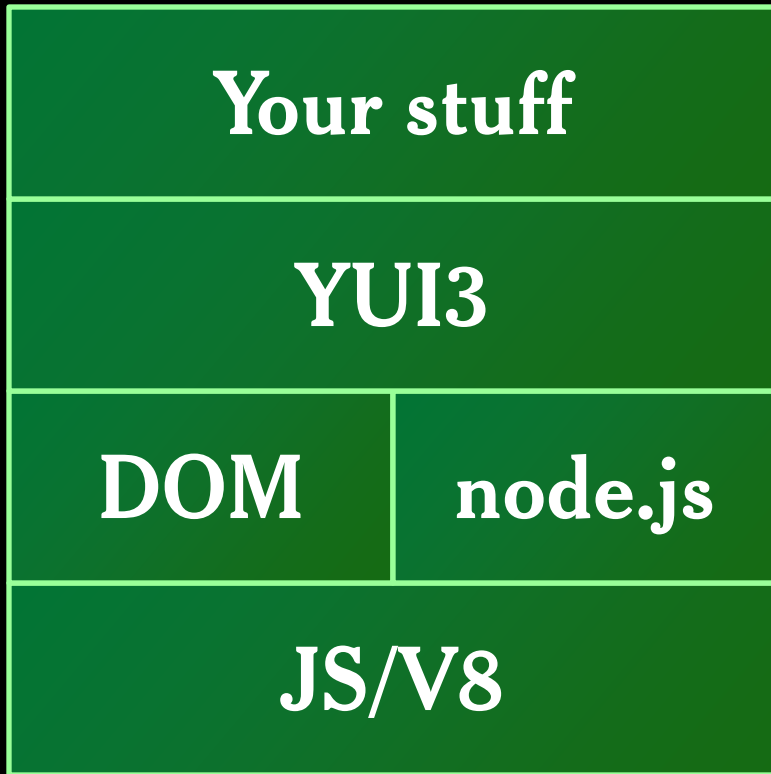
- This is not a new idea.
- Netscape offered an SSJS product in 1996.
- It was a page template system using a `<server>` tag and a `write` function to insert matter in the output stream.
- It had all of the disadvantages of the other page template systems with a really slow JS engine.

**What about Server Side
JavaScript with an Event
Loop?**

node.js

- node.js implements a web server in a JavaScript event loop.
- It is a high-performance event pump.
- `fs.read(fd, length, position, encoding, function (err, str, bytesRead) { ... })`
- Everything is (or can be) non-blocking.
- Except:
 - some synchronous functions
 - `require`

Your stuff runs on both sides



Stack of Protocols

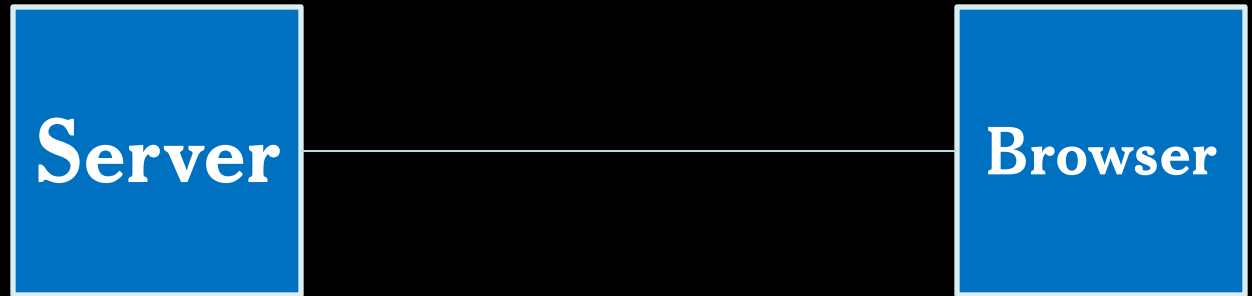
Ajax Stateful

HTTP Stateless

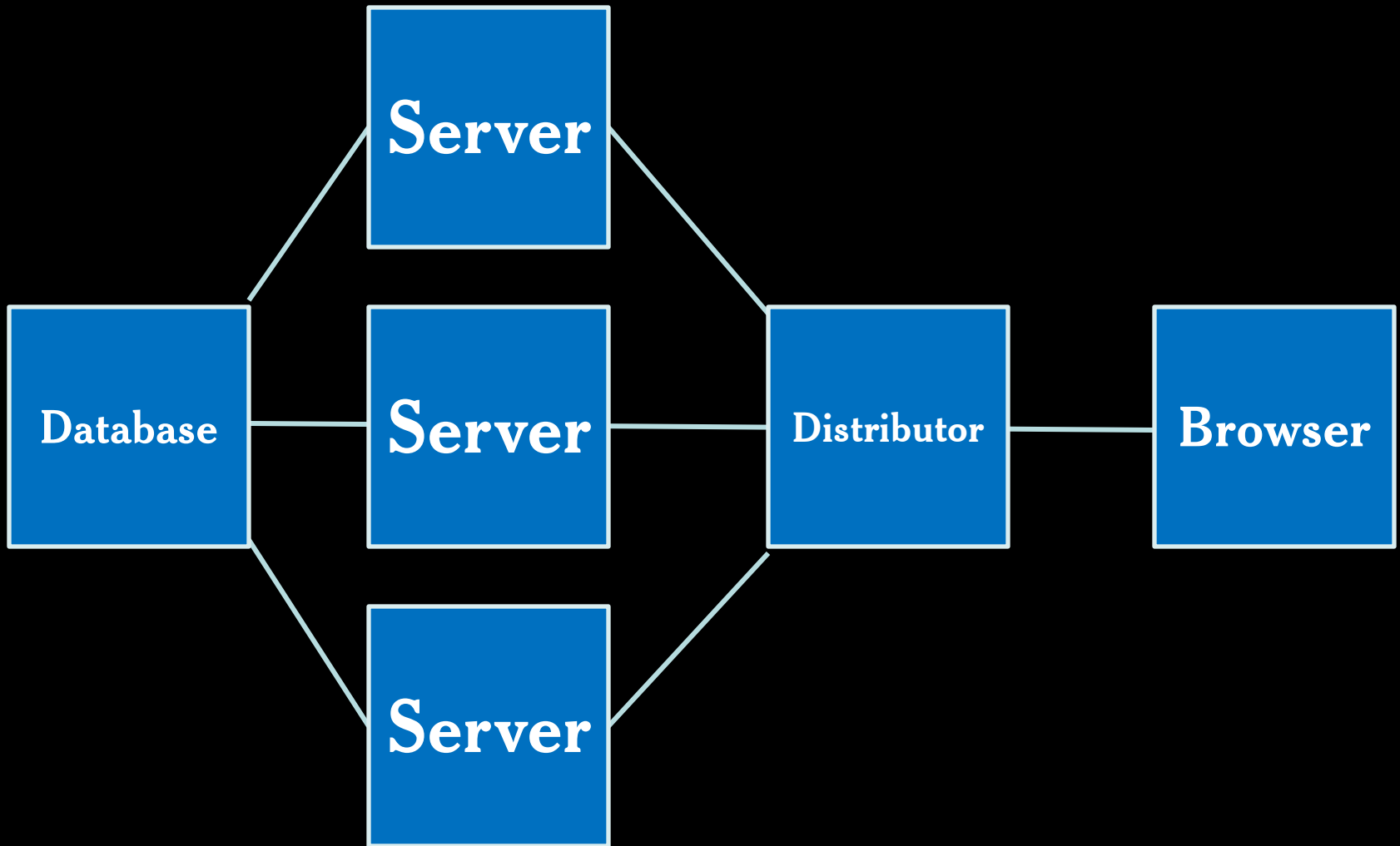
TCP Sessionful

IP Sessionless

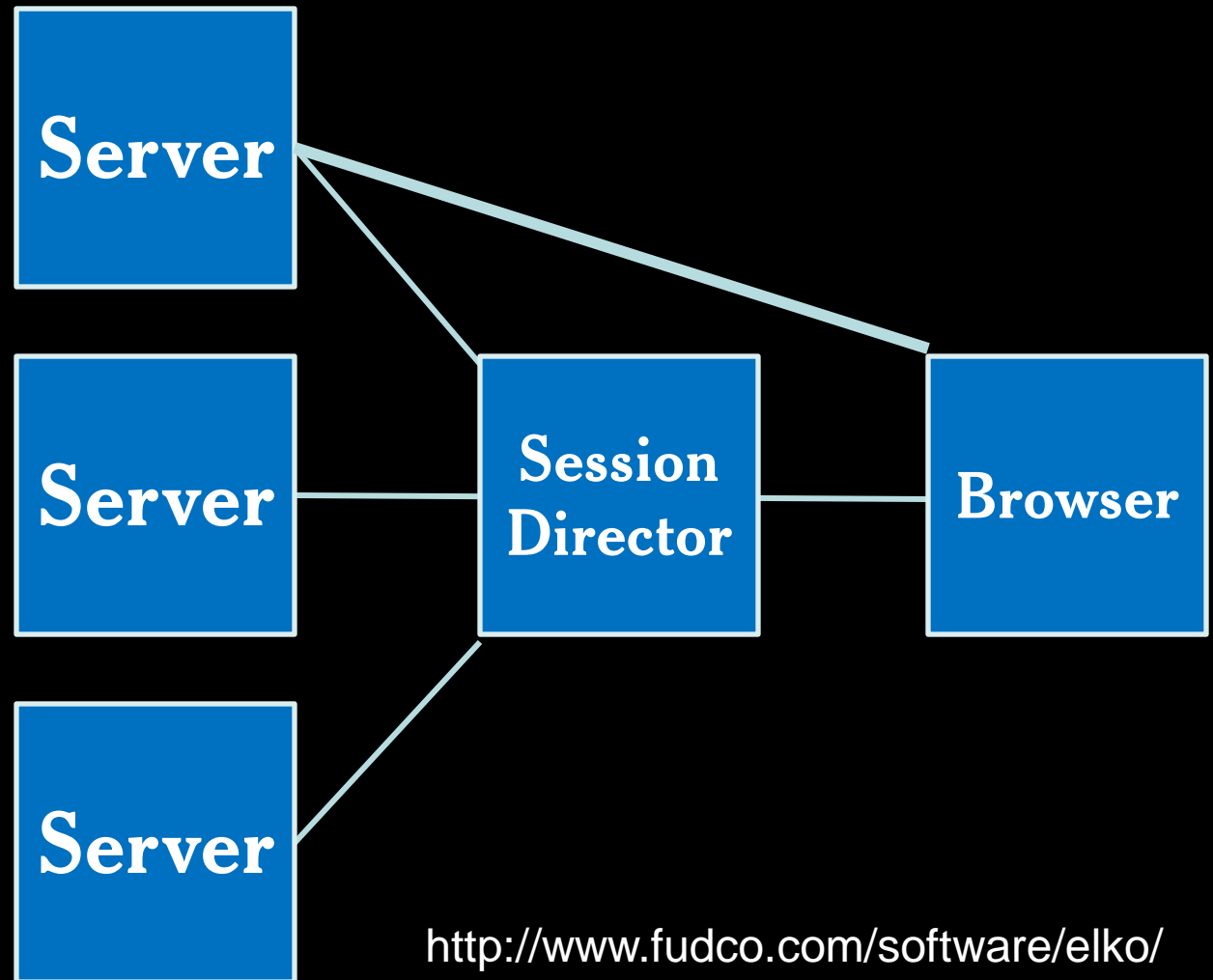
In the beginning...



Scaling out...



Elko



<http://www.fudco.com/software/elko/>

Final Thought



Courtesy WGBH

Thank you and good night.