

<bigwig>
A Programming Language
for Developing
Interactive Web Services

Claus Brabrand

BRICS, University of Aarhus, Denmark

Plan

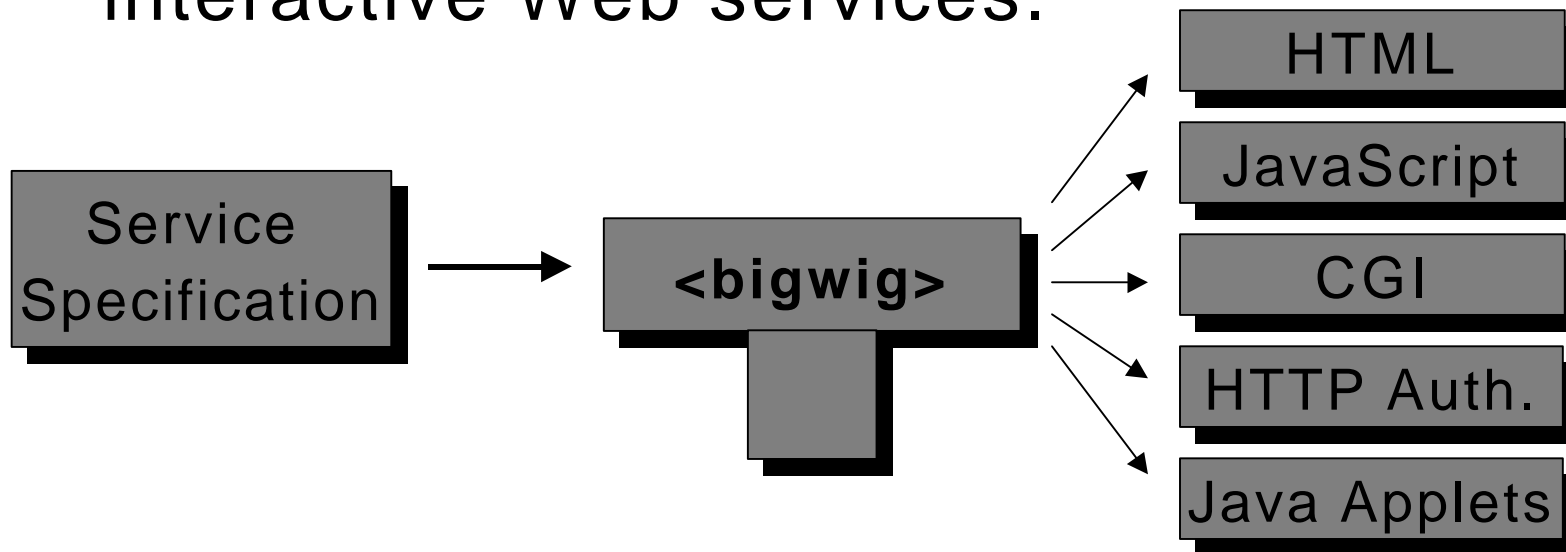
- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

Plan

- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

What is <bigwig>?

- A domain-specific high-level programming language for developing interactive Web services.



A collection of DSLs

- C-like skeleton language with
 - Runtime system
 - Concurrency control
 - Database
 - Dynamic documents: *DynDoc*
 - Input validation language: *PowerForms*
 - Security
 - Cryptographic security
 - Syntactic-level macros

A collection of DSLs

- C-like skeleton language *with*
 - *Runtime system*
 - Concurrency control
 - Database
 - *Dynamic documents: DynDoc*
 - *Input validation language: PowerForms*
 - Security
 - Cryptographic security
 - Syntactic-level macros

DSL vs. GPL

- DSL ::= Domain Specific Language
- GPL ::= General Purpose Language
- DSL?
 - Targeted for specific problem domain
 - Abstraction level match problem domain
- Examples: Lex/Yacc, LaTeX

DSL vs. GPL

- DSL ::= Domain Specific Language
- GPL ::= General Purpose Language
- DSL?
 - Targeted for specific problem domain
 - Abstraction level match problem domain
- Examples: Lex/Yacc, LaTeX, <bigwig>

DSL Advantages

... vs. GPL + library

- Syntax
- Analysis
- Implementation

Goals

- Lower development time (= cost):
 - Targeted at Web services
 - Low-level → high-level
- Increase functionality:
 - Compiler does “the dirty work”
- Reliability:
 - Catch errors during development
 - » Runtime errors → Compile-time errors

Assumptions

- “Rules of engagement”:
 - Lowest common denominator
 - Any browser/Web server combination
 - Only include basic primitives
 - Syntactic macro language does the rest

Target Audience

- Programmers!
 - No expert Web knowledge required
 - No multiple choice questionnaires or drag'n'drop

“Reduce Web service development to a standard programming task.”

Core Language Features

- C-like to minimize syntactic burdens
- Features:
 - Garbage collection
 - Relations, vectors, tuples
 - Strong type checking

People

- 1x Michael I. Schwartzbach
- 1x Post Doc.
- 3x Ph.D. students
- 2x Programmers
- 2x Testers
- 1x External contributor

Plan

- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

Plan

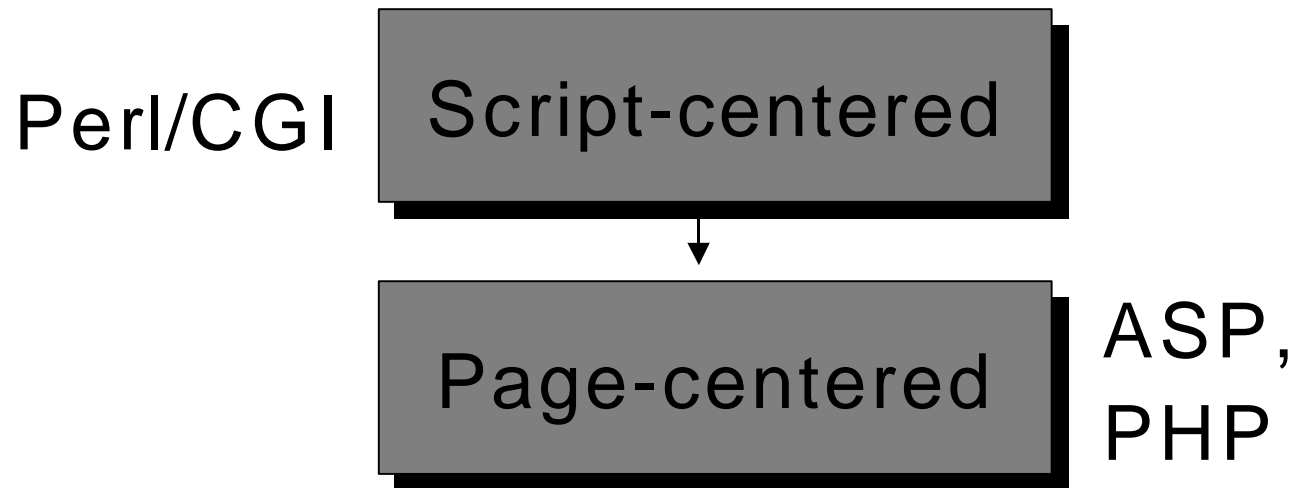
- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

3 Approaches

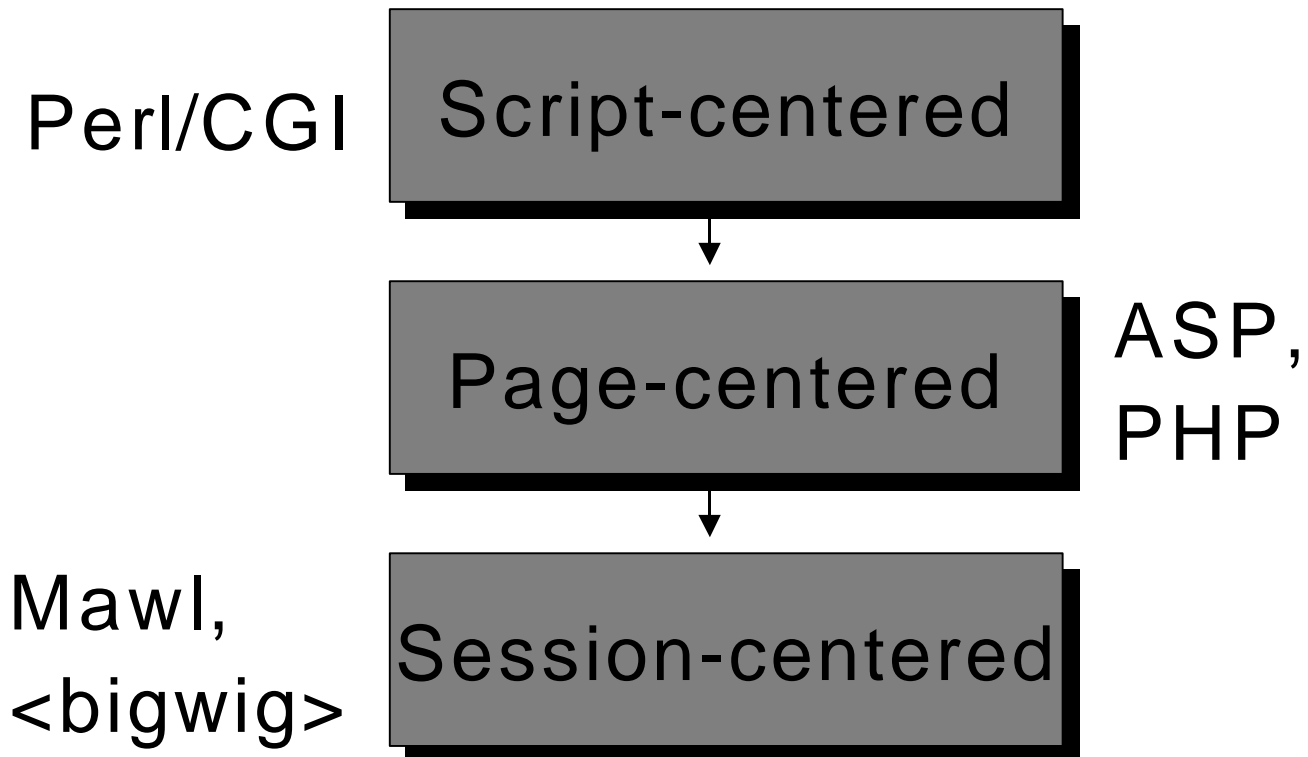
Perl/CGI

Script-centered

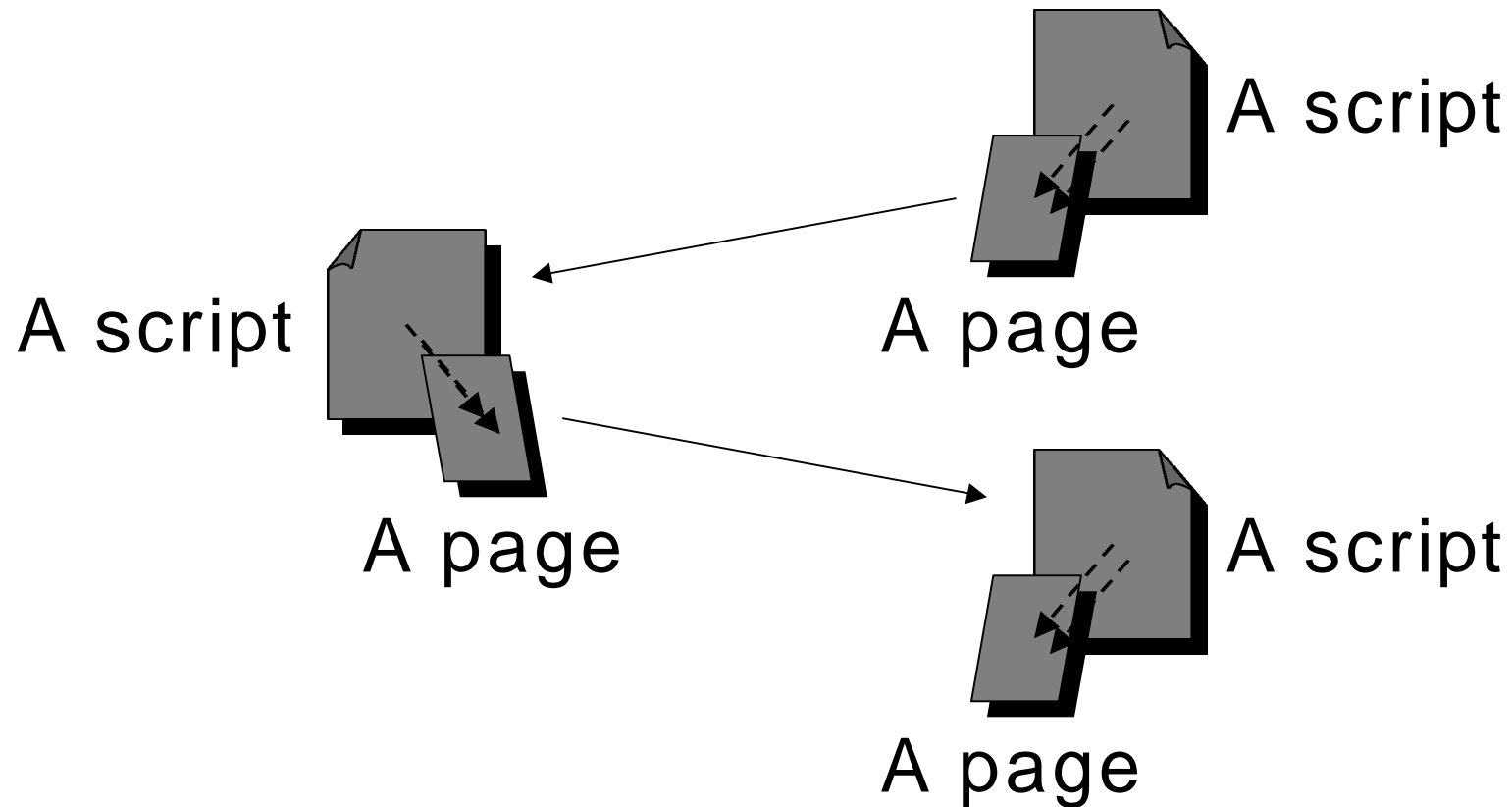
3 Approaches



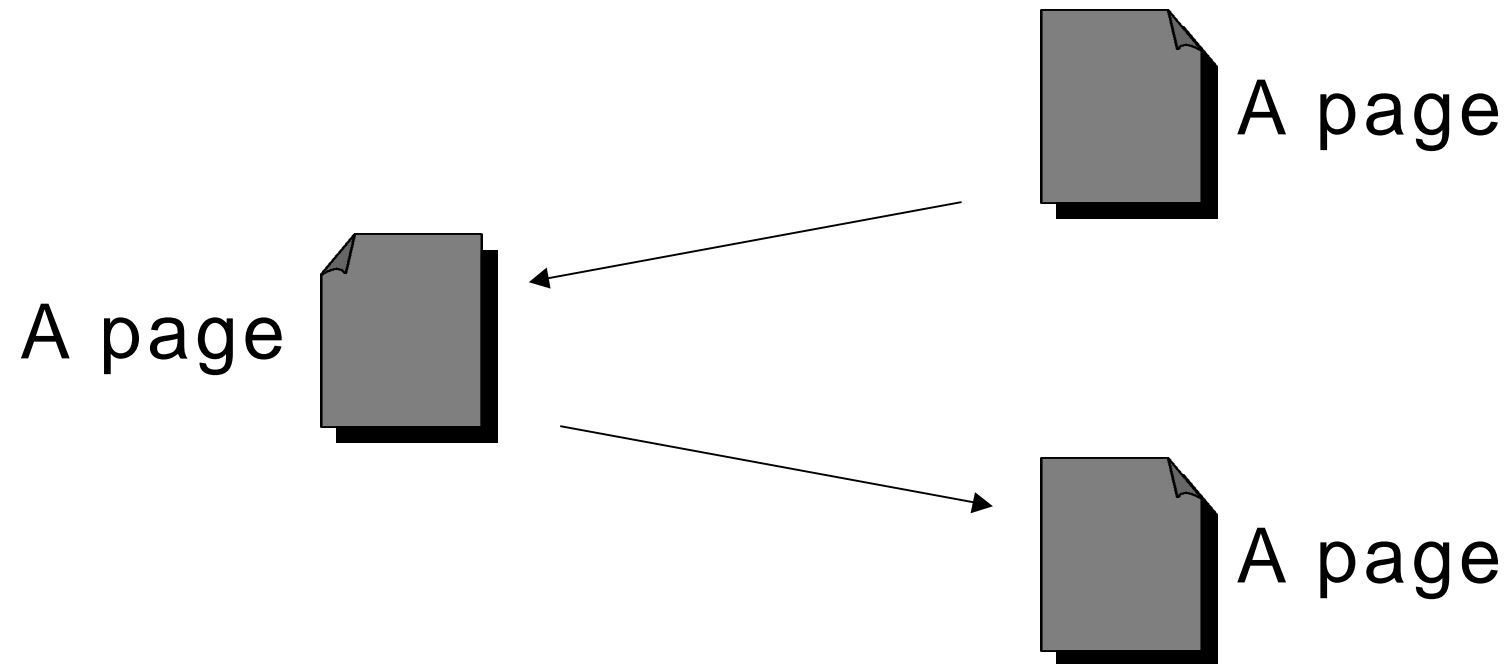
3 Approaches



Script-Centered



Page-Centered



Page-Centered

“Service code embedded in tags and interpreted by specialized Web server”

- Increased level of abstraction
- Easy to add dynamics to static pages
- Scalability

Script / Page-Centered

- As the service complexity increases:
 - Page-centered → Script-centered
- Script-centered:
 - default programming, escape printing.
- Page-centered:
 - default printing, escape programming.

(Fundamental) Problems

- Interpretation-based:
 - Typically no static checks
 - (Efficiency)
- Not domain specific:
 - Cannot check Web related issues
- Implicit control-flow:
 - A service = A collection of scripts/pages!

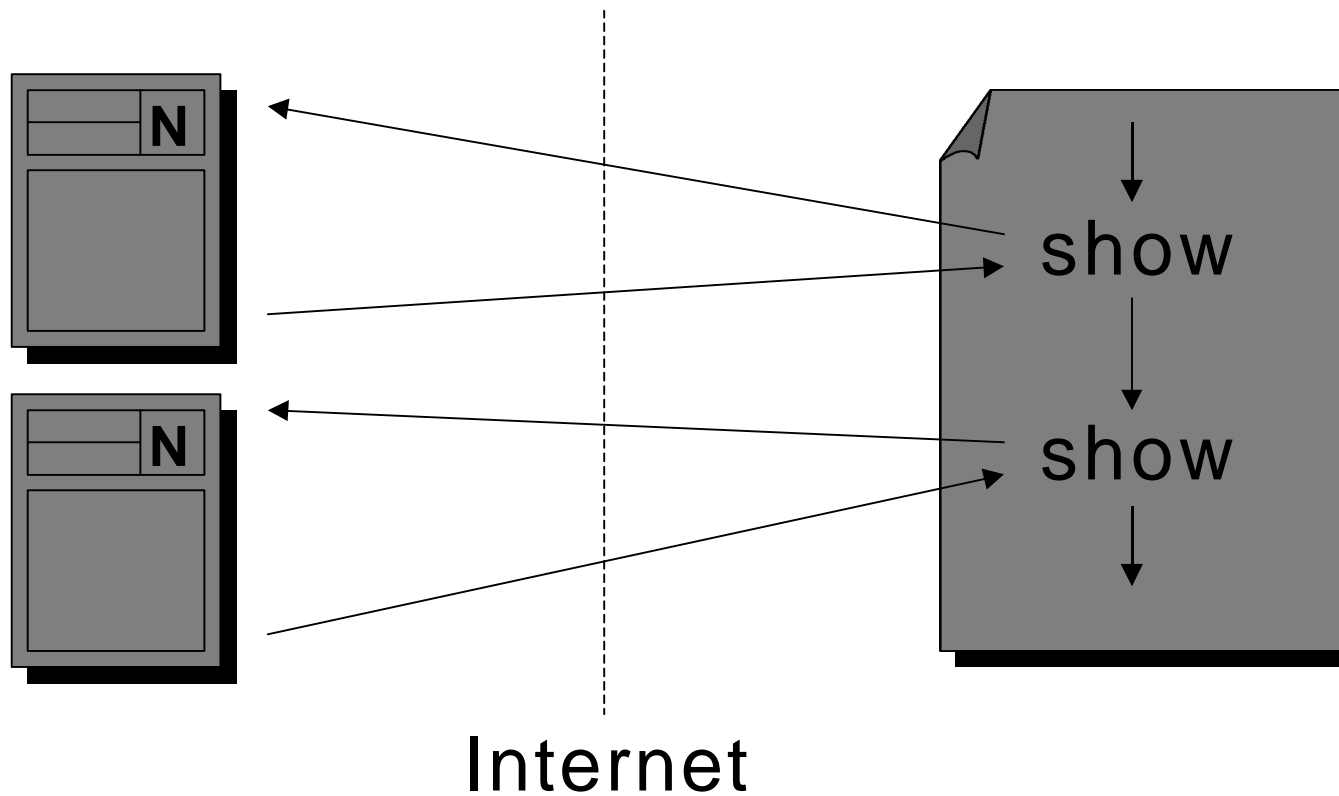
Language Requirements

- Compilation-based:
 - Static checks
 - (Efficiency)
- Domain specific:
 - Check Web related issues
- Explicit control-flow:
 - A clear service specification

Session-Centered

client:

server:



Hello World

```
service {  
    session Hello() {  
        html D = <html>Hello World!</html>;  
        show D;  
    }  
}
```

Hello World

```
service {  
    session Hello() {  
        show <html>Hello World!</html>;  
    }  
}
```

A Page Counter

```
service {  
  session Access() {  
    global int counter;  
    string name;  
    show EnterName receive [name=name];  
    counter = counter + 1;  
    show AccessDoc <[counter = counter];  
  }  
}
```

A Page Counter

```
:  
if (counter == 100) {  
    show Congratulations <[name = name];  
    counter = 0;  
} else {  
    show EnterName receive [name=name];  
}  
:
```

CGI Shortcomings

- Stateless protocol
 - Session model requires state
- No bookmarking
 - CGI, not HTML URL
- Back-button problem
 - “Step-back-in-time” does not make sense

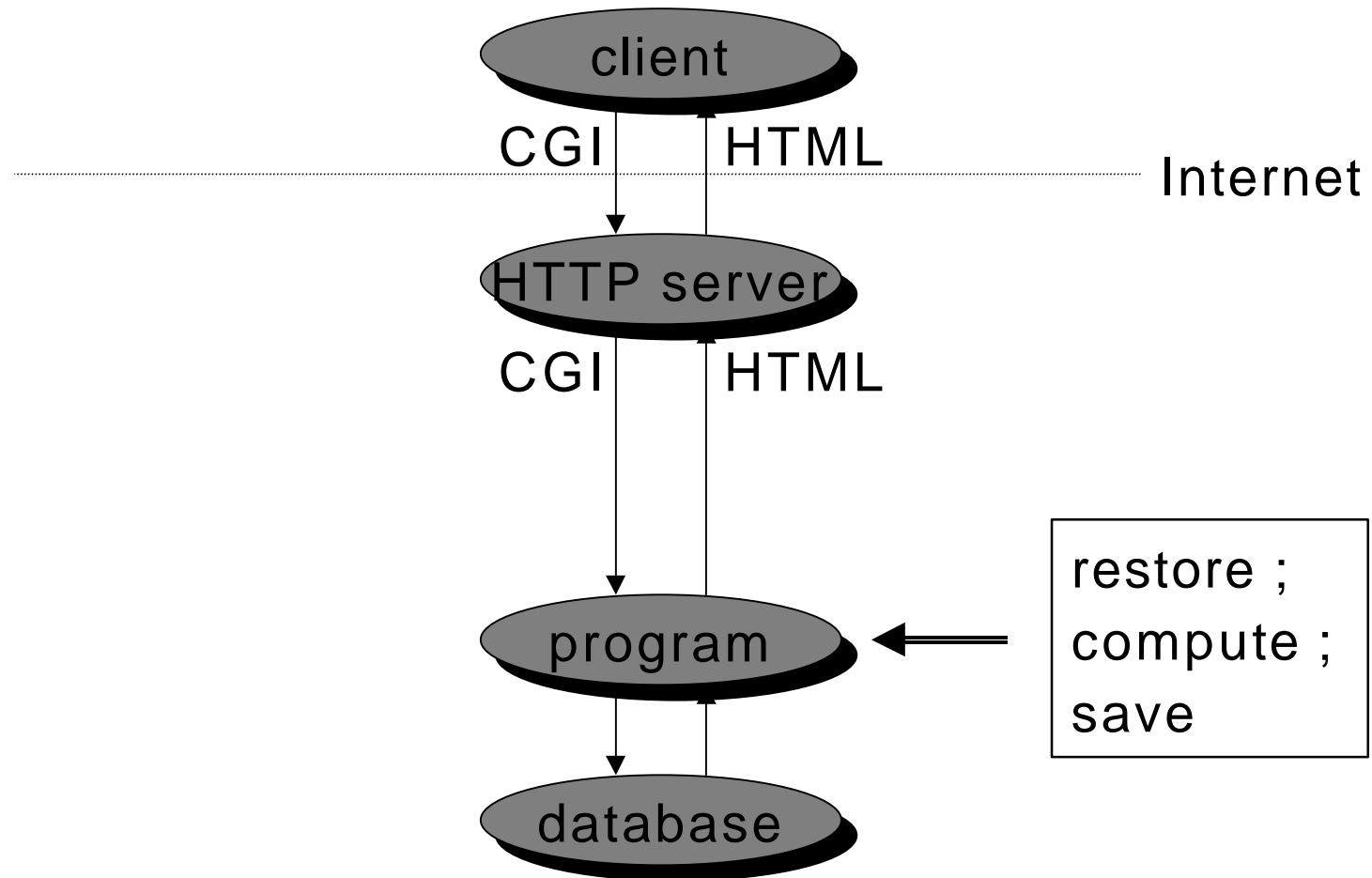
CGI Shortcomings

- Stateless protocol
- No bookmarking
- Back-button problem

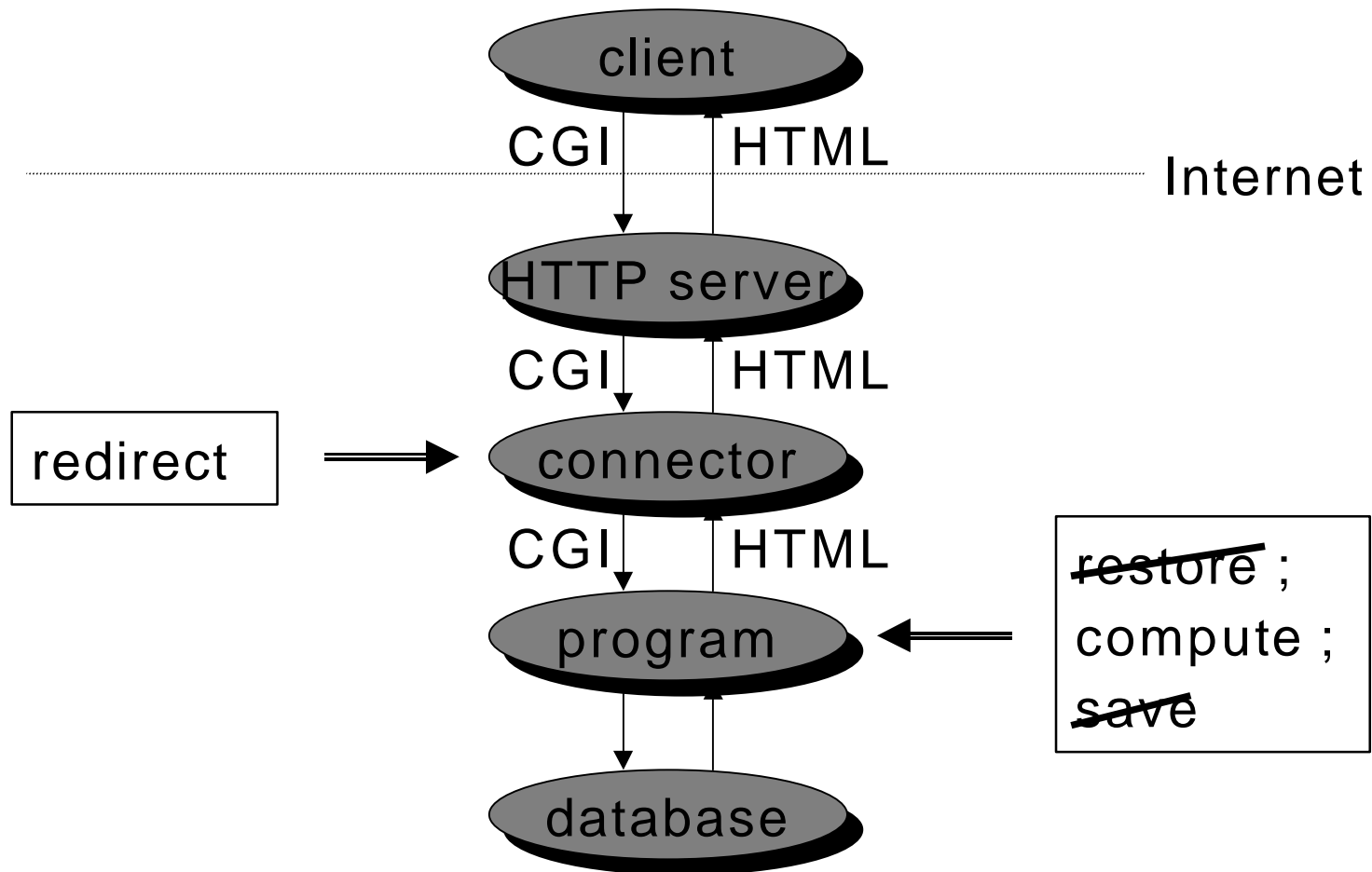
CGI Shortcomings

- Stateless protocol
- No bookmarking
- Back-button problem

Components



Adding a "Connector"



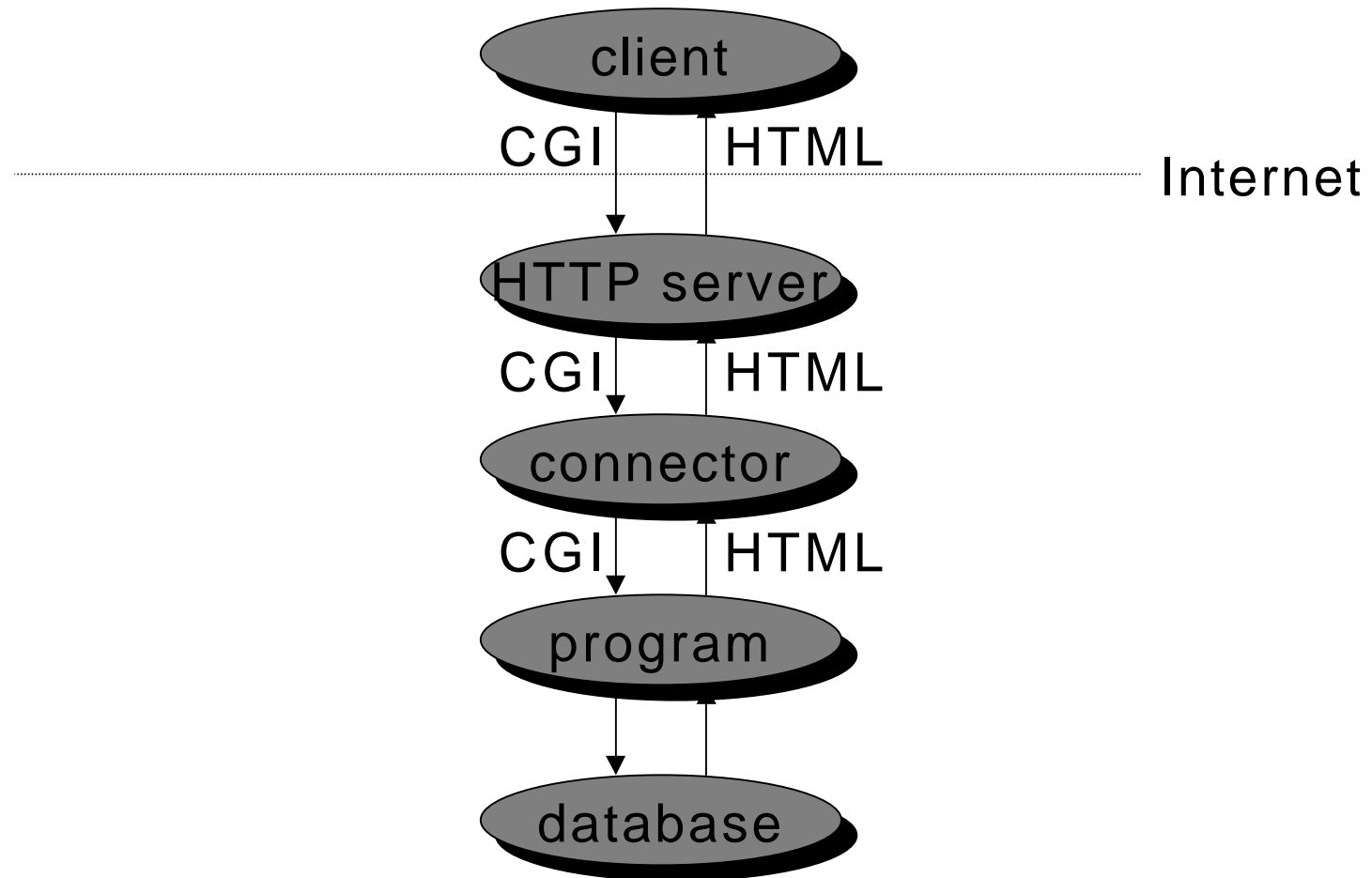
CGI Shortcomings

- Stateless protocol
- No bookmarking
- Back-button problem

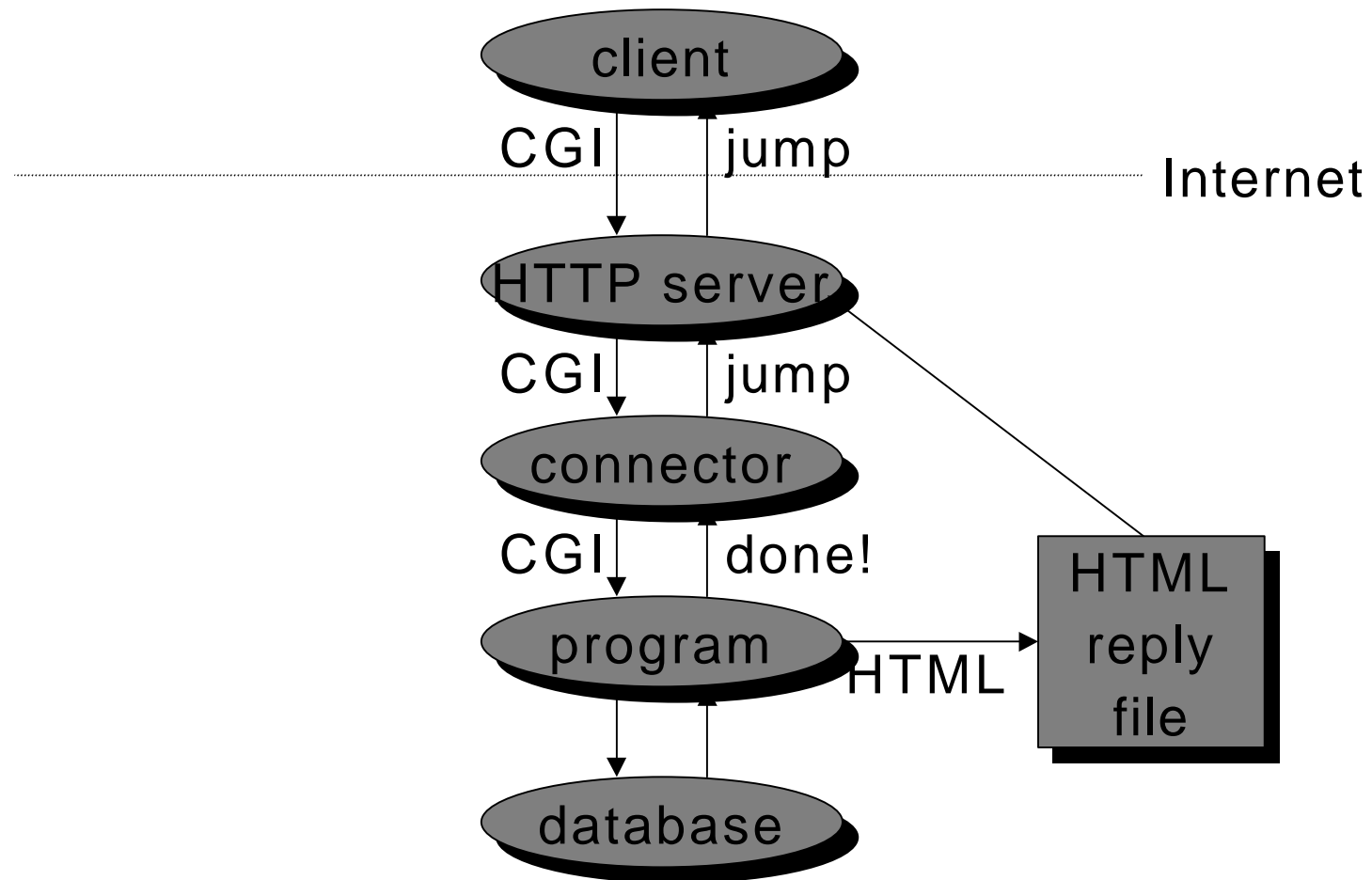
CGI Shortcomings

- Stateless protocol
- No bookmarking
- Back-button problem

Components



Adding an HTML "Reply File"



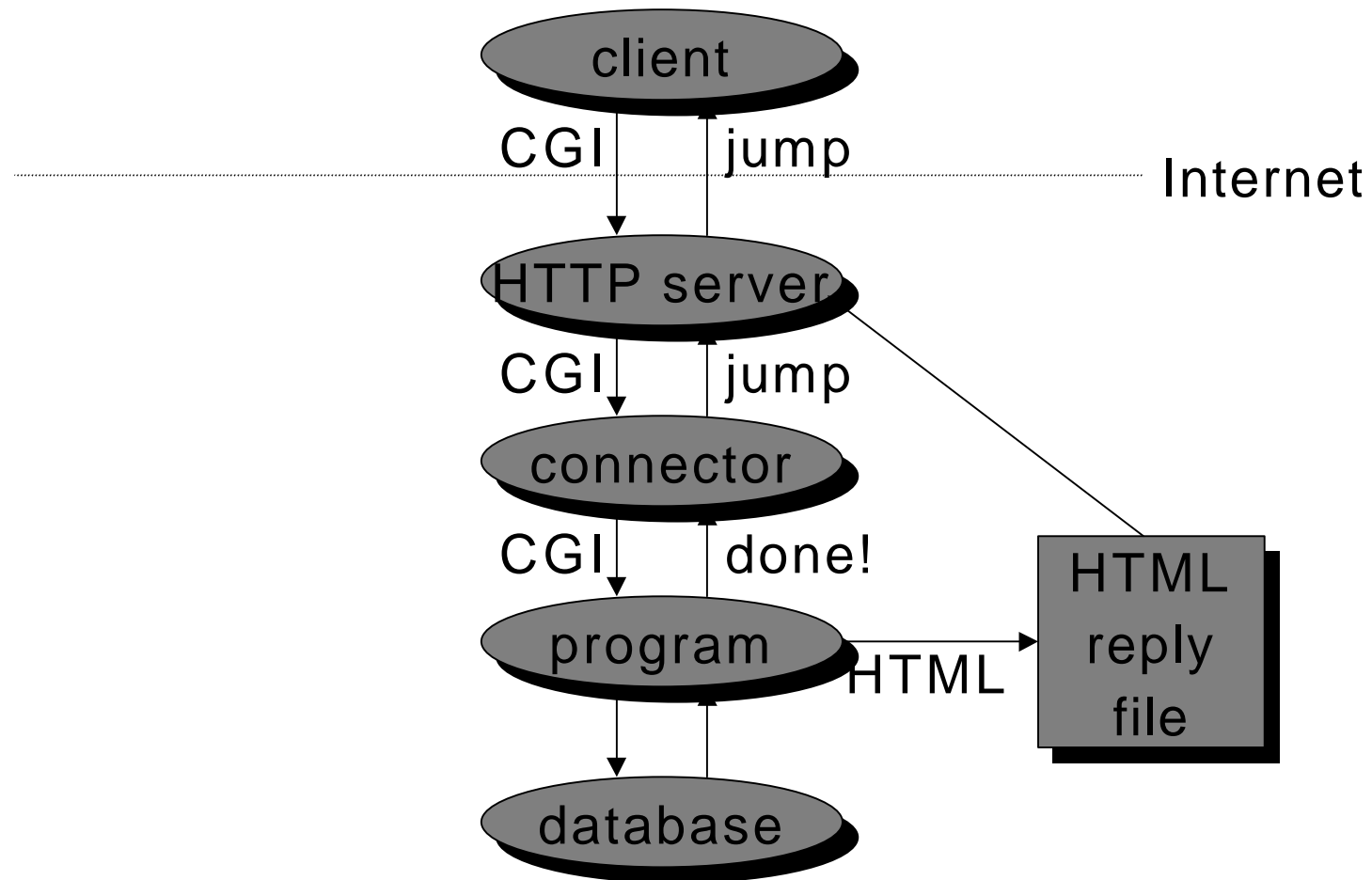
CGI Shortcomings

- Stateless protocol
- No bookmarking
- Back-button problem

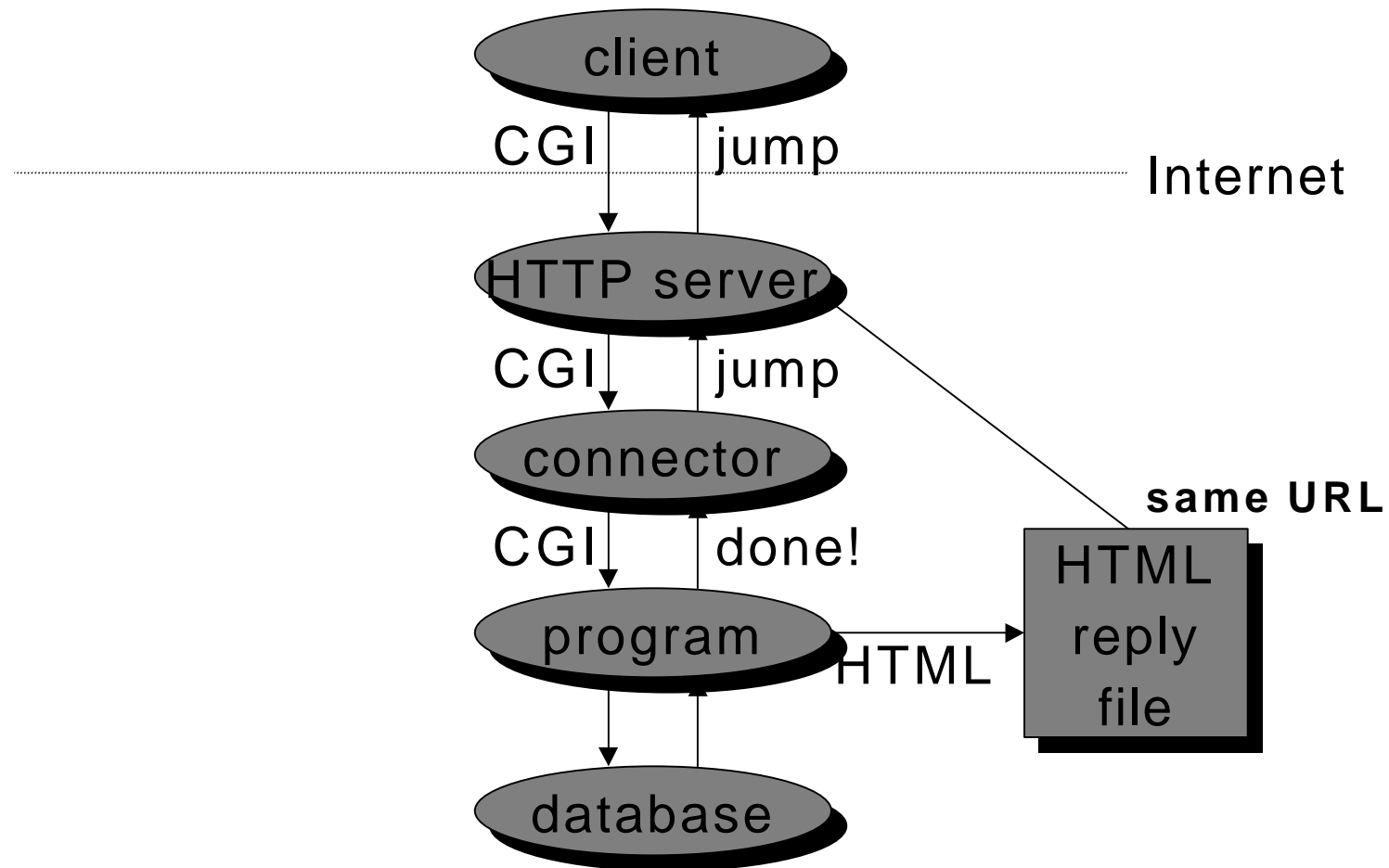
CGI Shortcomings

- Stateless protocol
- No bookmarking
- Back-button problem

Components



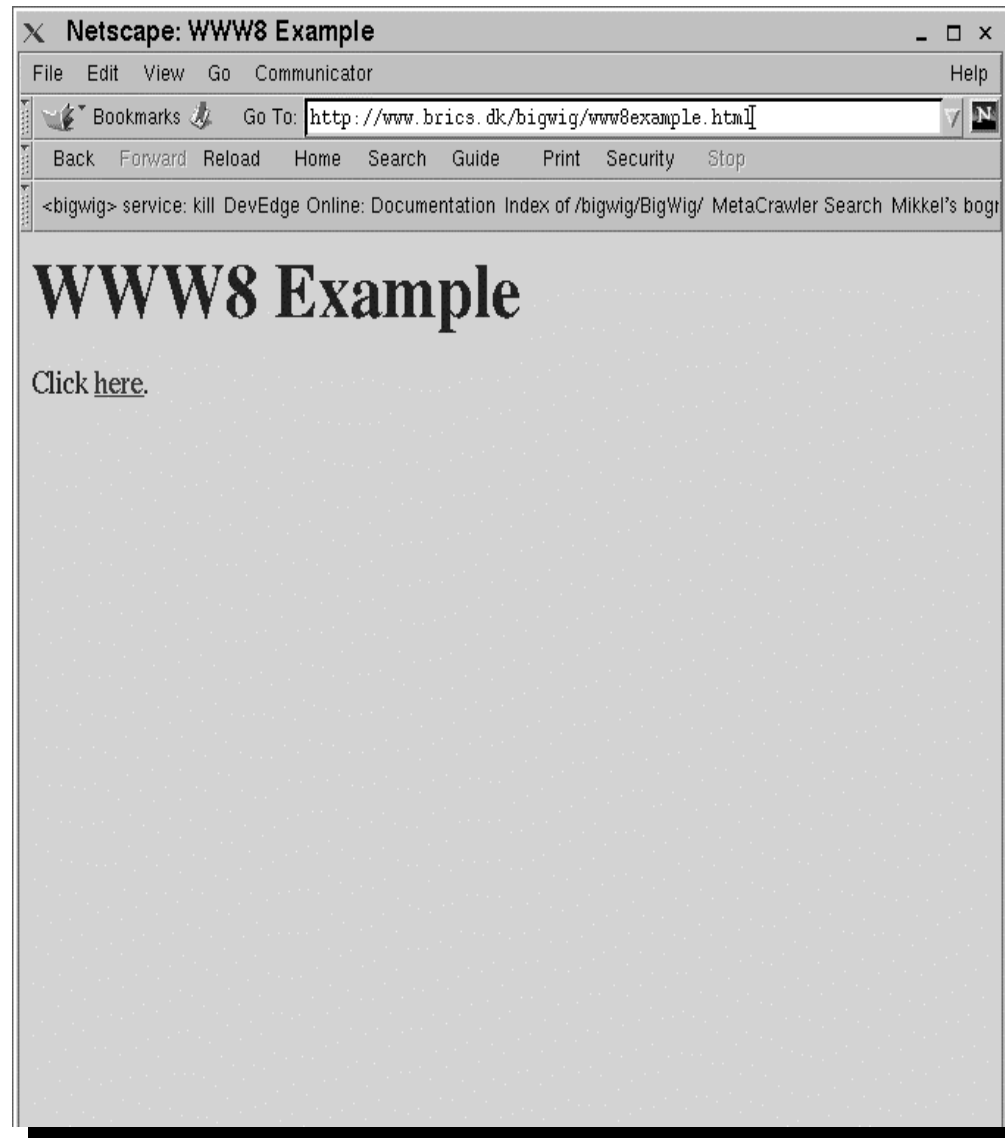
Components



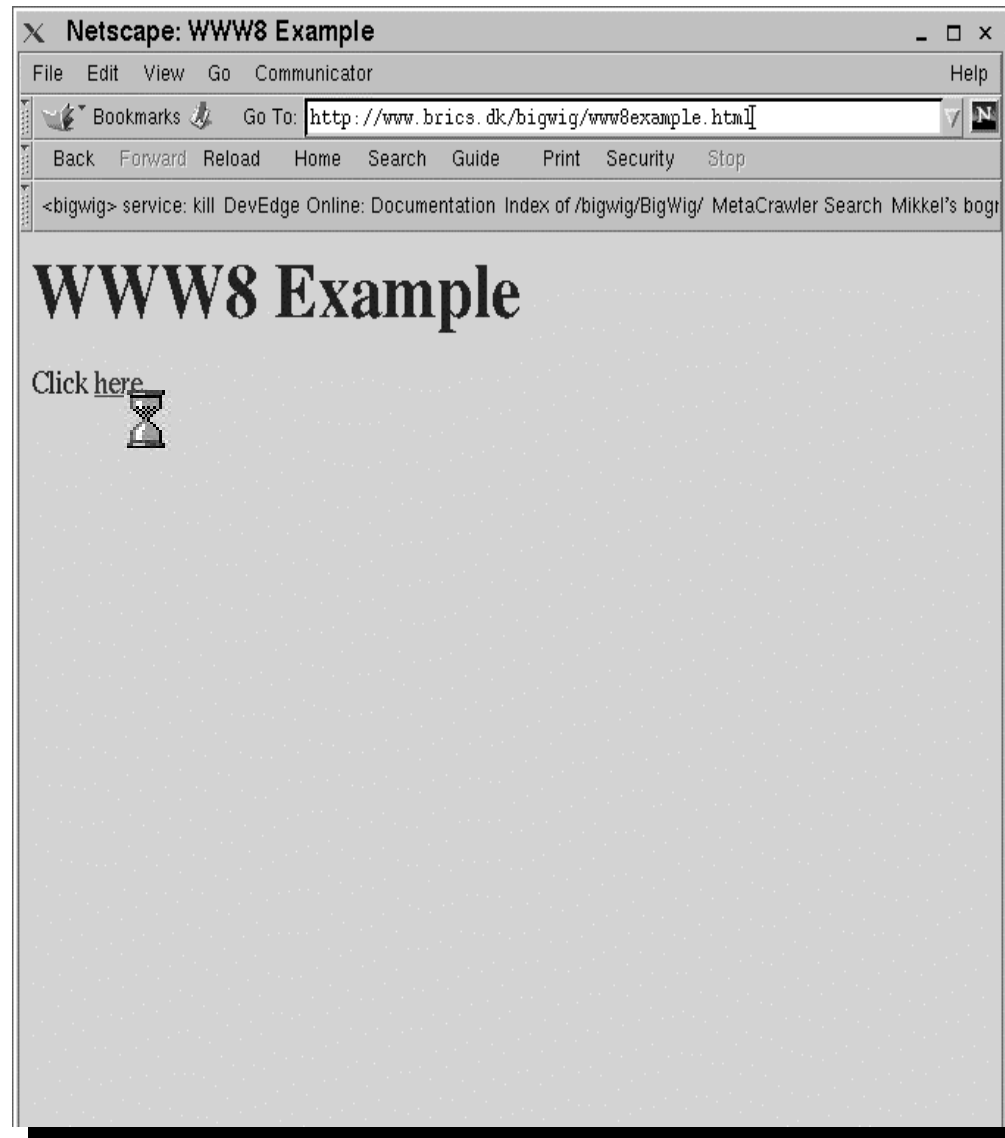
CGI Shortcomings

- Stateless protocol
- No bookmarking
- Back-button problem

Additional Problems



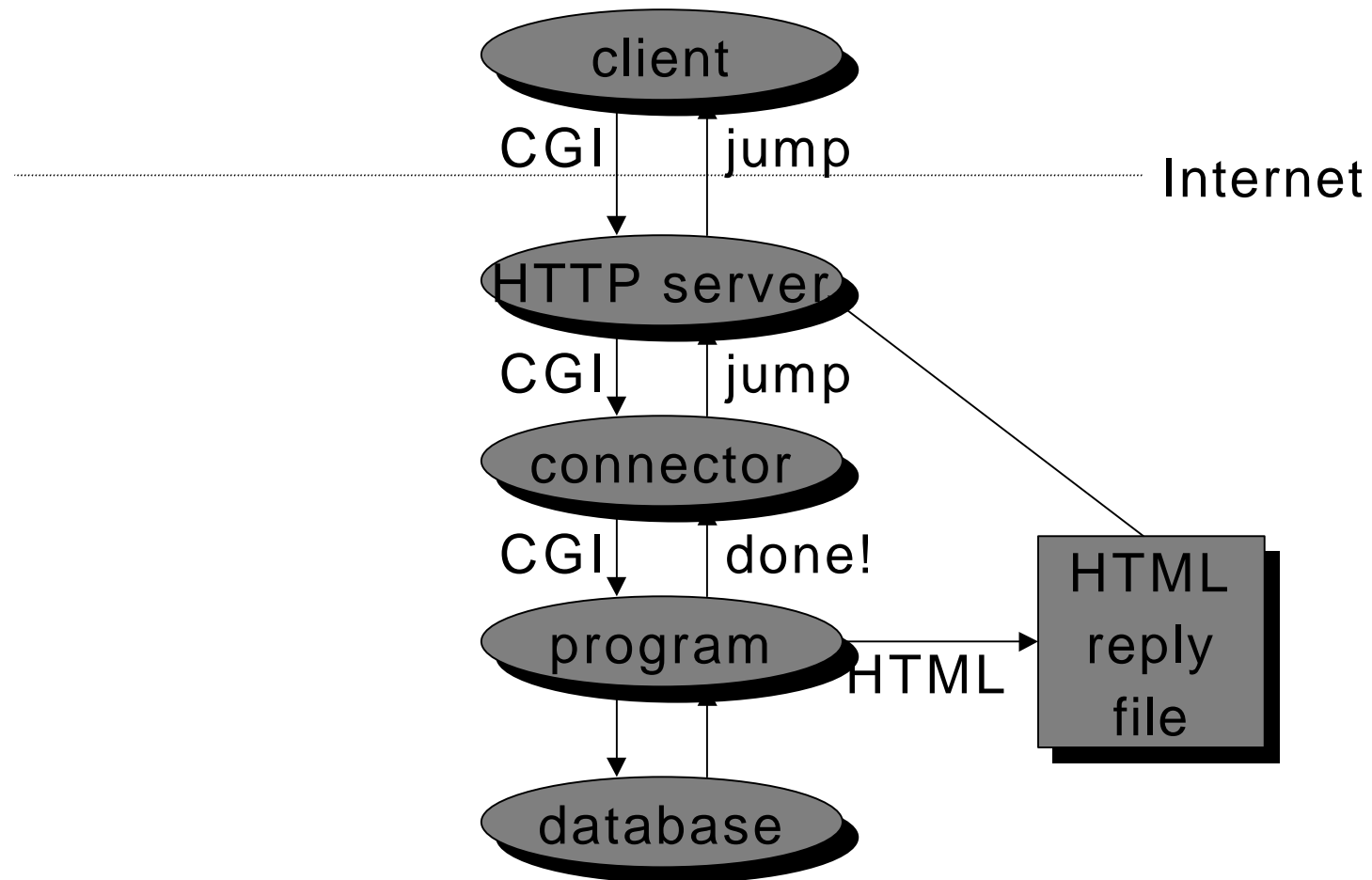
Additional Problems



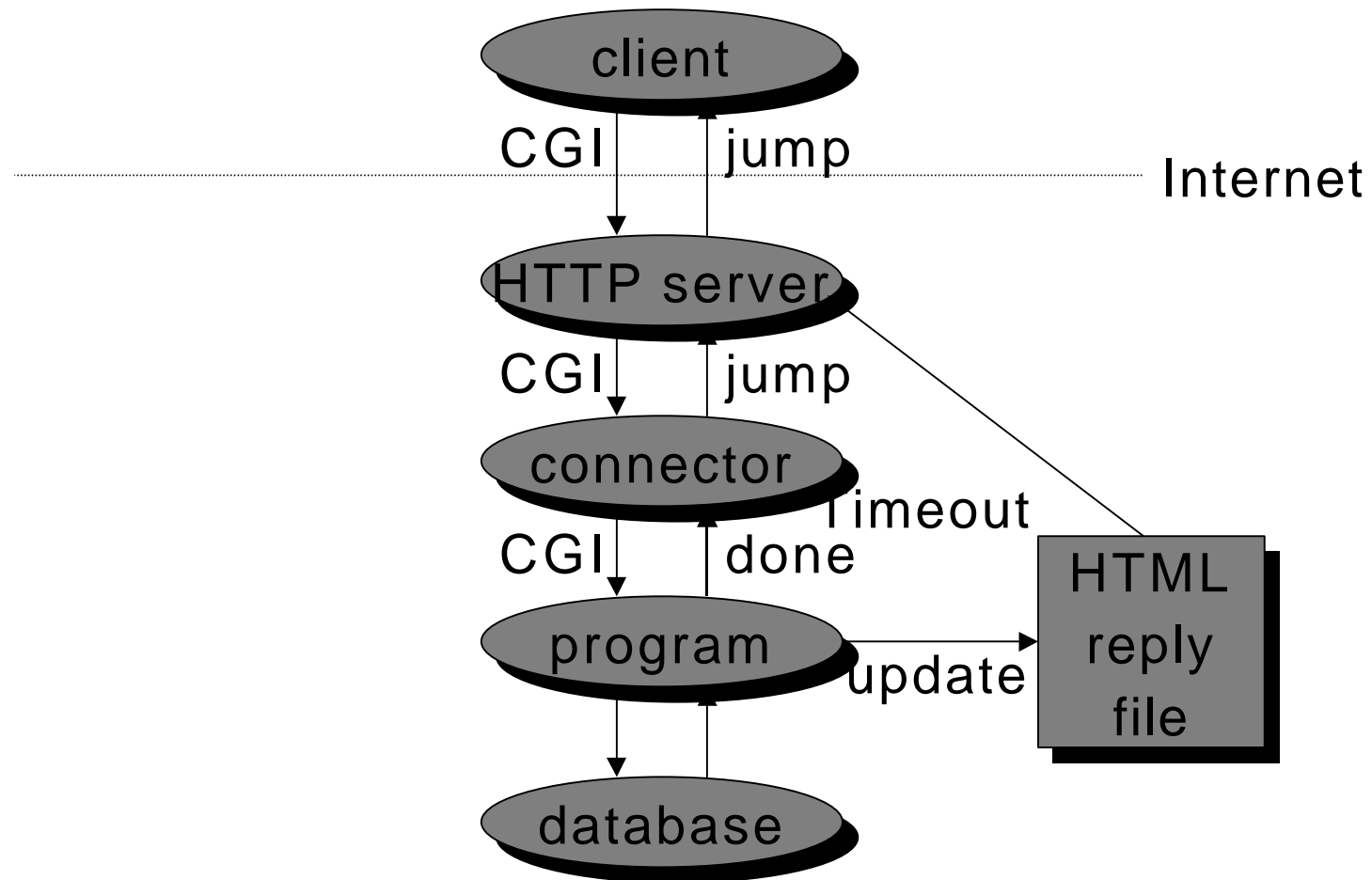
What is going on?

- Error?
 - Package Lost, Service Crash, Connection down?
- Ok?
 - Searching Database, Long Computation, Waiting for a Resource?
- Would like to explain delays:
 - “searching database, please wait...”

Components



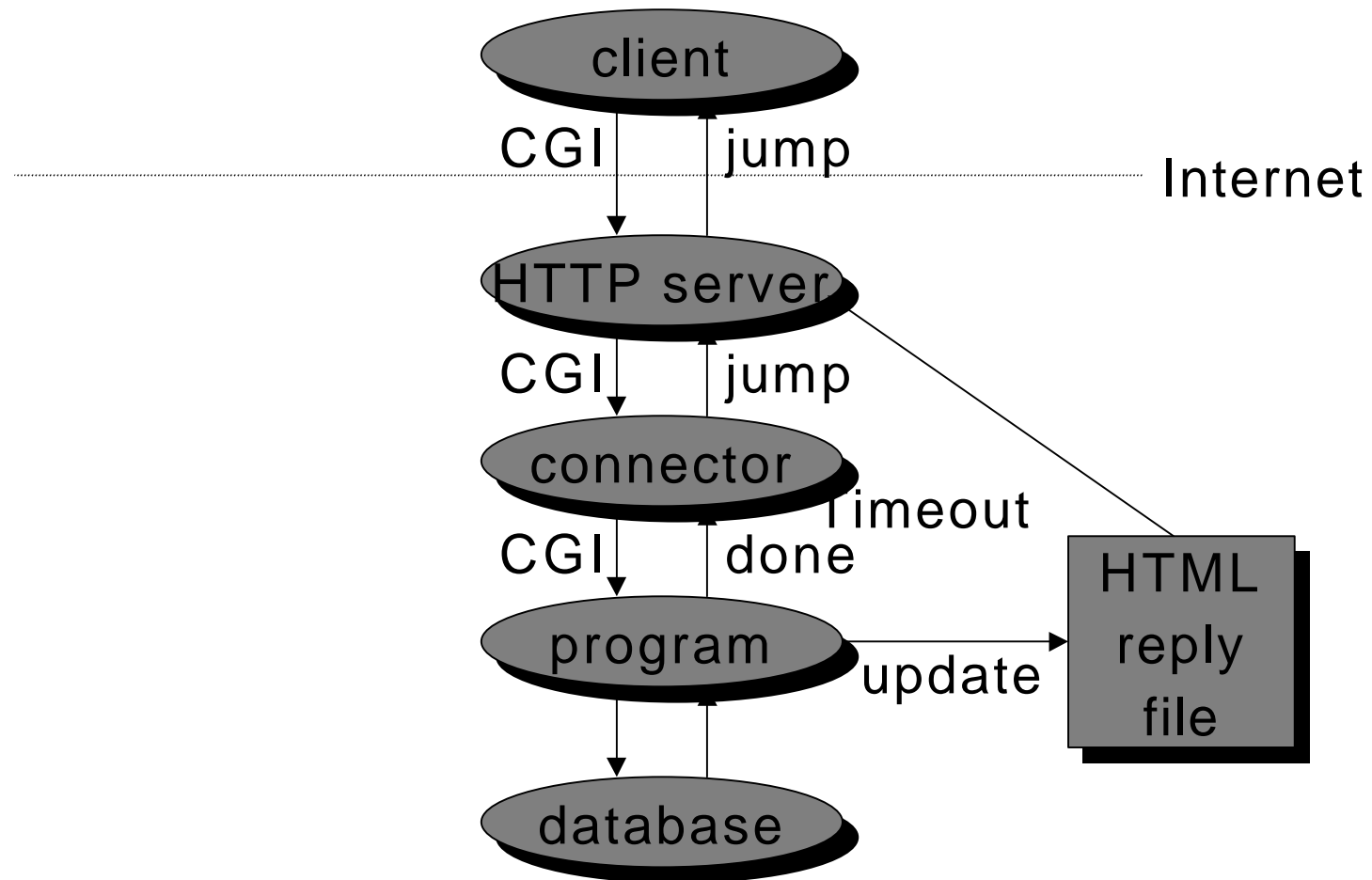
Adding a Connector Timeout



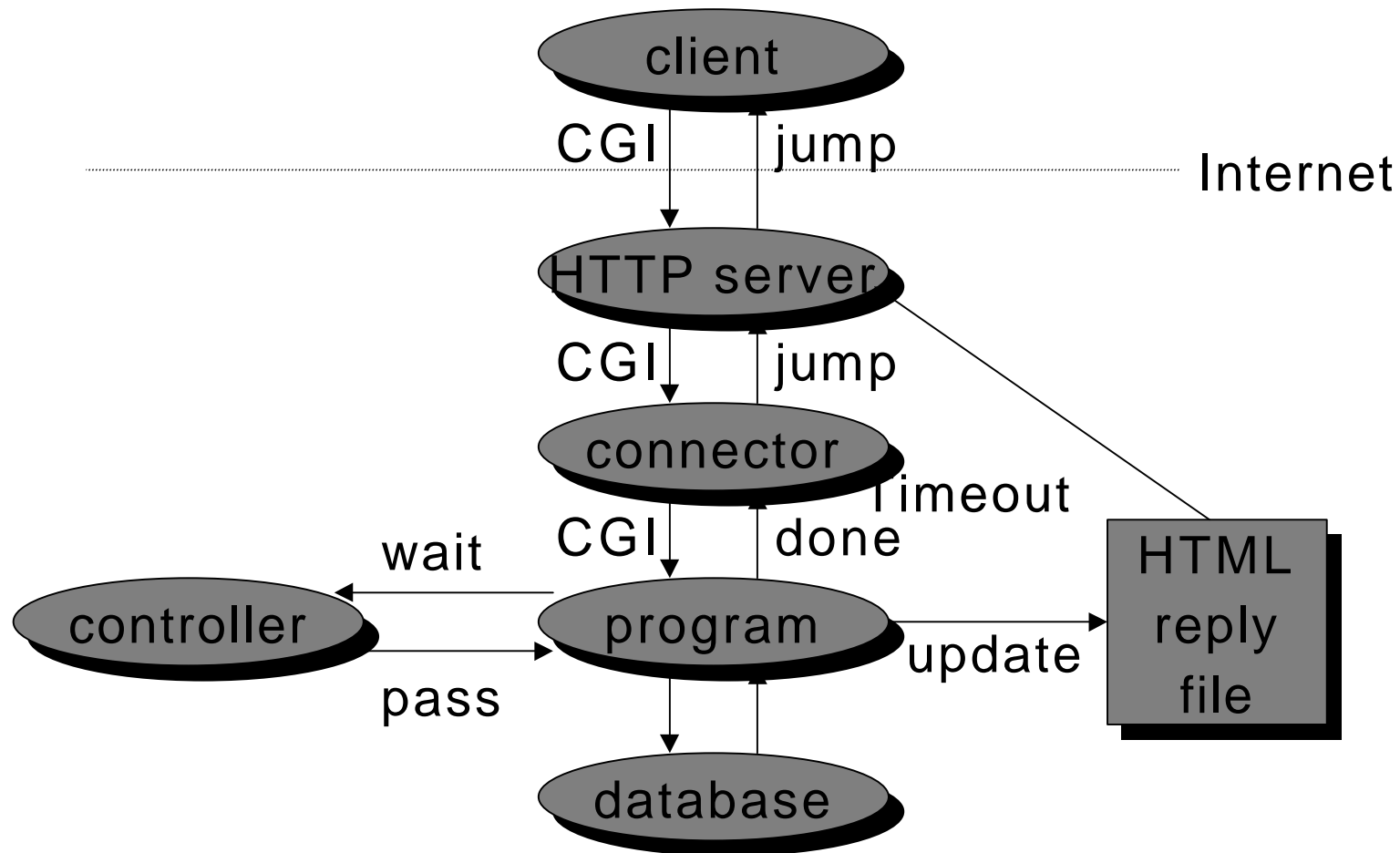
Concurrency Control

- Problem: Parallel service processes.
 - Access shared resources.
 - Require synchronization.
- Solution:
 - Specification of checkpoints & constraints
 - Synthesize centralized **safety controller**
 - Ensures that service obeys constraints.

Components



Adding a Safety Controller



Concurrency Control

:

```
counter = counter + 1;
```

:

Concurrency Control

```
:  
wait A;  
counter = counter + 1;  
wait B;  
:
```

Concurrency Control

$$\forall t, t'': A(t) \wedge A(t'') \Rightarrow \exists t': t < t' < t'' \wedge B(t')$$

:

wait A;

counter = counter + 1;

wait B;

:

A Page Counter

```
service {  
    session Access() {  
        global int counter;  
        :  
        counter = counter + 1;  
        :  
    }  
}
```

A Page Counter (with Macros)

```
service {  
    session Access() {  
        region global int counter;  
        :  
        exclusive (counter) {  
            counter = counter + 1;  
        }  
    }  
}
```

Demo Example: Mutex

```
int i;  
bool quit;  
while (!quit) {  
    flash WaitToEnterDoc;  
    wait A;  
    show DocA <[no=i] receive [quit = quit];  
    i++;  
}  
show GoodByeDoc;
```

Runtime System

- Availability:
 - In <bigwig> compiler and
 - As stand-alone package

Plan

- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

Plan

- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

Documents

- Traditionally: printf / <% print(...) %>
- Problems:
 - Only linear construction
 - Programmer/Designer tasks not separated
 - No Show/Receive correspondence
 - Legal/sensible HTML generated?

Documents

- Traditionally: printf / <% print(...) %>
- Problems:
 - Only linear construction
 - Programmer/Designer tasks not separated
 - No Show/Receive correspondence
 - Legal/sensible HTML generated?

Our Solution: Document Templates

- HTML → HTML with named gaps

```
<html>  
  <body bgcolor=[bgcolor]>  
    <h1>Hello <[what]>!</h1>  
  </body>  
</html>
```

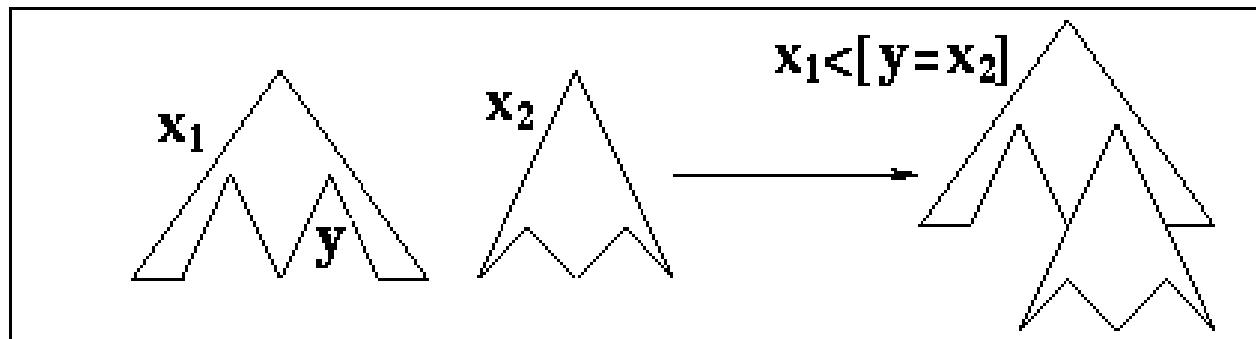
- ...gaps plugged at runtime.

Dynamic Documents

- Domain specific type: html (with gaps)
 - $type ::= \underline{int} \mid \underline{float} \mid \underline{string} \mid \dots \mid \underline{html}$
- Domain specific (sub)language: *DynDoc*
 - $exp ::= \dots \mid c \mid id \mid id = exp \mid exp <[id = exp]$
 - $stm ::= \dots \mid \underline{show} \ exp; \mid$
 $\underline{show} \ exp \underline{receive} \ [\ id = id \];$

Plug

- Syntax:
 - $exp ::= exp <[id = exp]$
- Semantics:

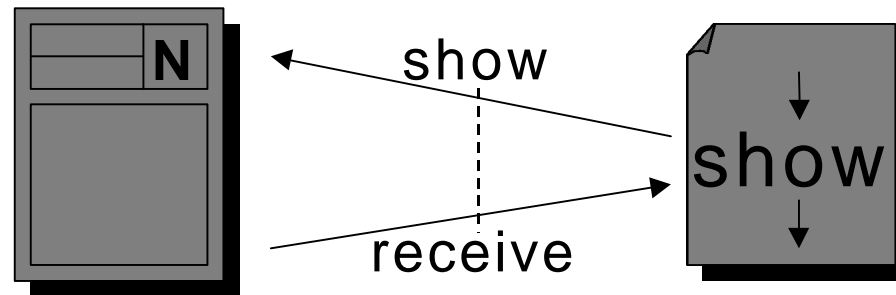


Show / Show-Receive

- Syntax:

– $stm ::= \underline{\text{show}} \text{ exp}; \mid$
 $\underline{\text{show}} \text{ exp} \underline{\text{receive}} [id = id];$

- Semantics:



Hello World (revisited)

```
session Hello() {  
  html H;  
  html D = <html>Hello <[what]>!</html>;  
  H = D <[what = "World"]>;  
  show H;  
}
```

Example: EnterData

string name, email;

html Input = <html>

name: <input name="your_name">

email: <input name="your_email">

</html>;

show Input receive [name = your_name,
email = your_email];

Rec. Example: Genealogy

```
html GenDoc = <html><ul><li>...</ul></html>;
```

```
html genTree(int n, string s) {  
    if (n == 0) return <html></html>;  
    else return GenDoc <[mother = s + "mother",  
        mothers_tree = genTree(n-1, "mother's"),  
        father = s + "father",  
        father_tree = genTree(n-1, "father's")];  
}
```

Documents

- Problems:
 - Only linear construction
 - Programmer/Designer tasks not separated
 - No Show/Receive correspondence
 - Legal/sensible HTML generated?

Documents

- Problems:
 - Only linear construction ✓
 - Programmer/Designer tasks not separated
 - No Show/Receive correspondence
 - Legal/sensible HTML generated?

Documents

- Problems:
 - Only linear construction ✓
 - Programmer/Designer tasks not separated ✓
 - No Show/Receive correspondence
 - Legal/sensible HTML generated?

Documents

- Problems:
 - Only linear construction ✓
 - Programmer/Designer tasks not separated ✓
 - No Show/Receive correspondence
 - Legal/sensible HTML generated?

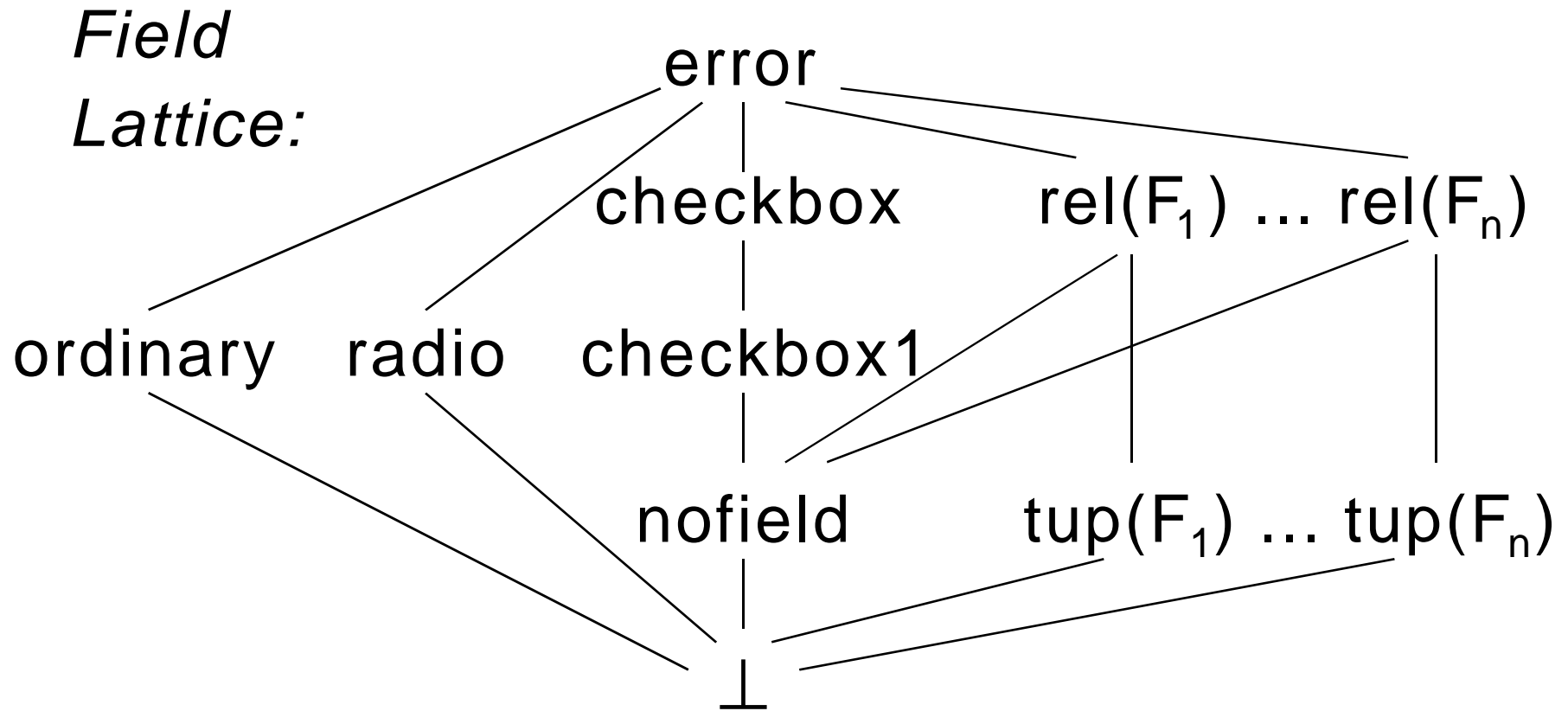
Static Guarantees?

- Documents Well-formed:
 - Two gaps with same name?
 - Field consistency?
- Plug operation:
 - Gap present?
 - Well-defined gap/field union?
- Show/Receive correspondence:
 - All fields received?
 - Receive types match?

Domain Specific Analysis

- Interprocedural data-flow analysis:
 - Infer exact types of all documents in program: (gaps, fields).
 - check:
 - documents well-formed
 - plug operations
 - show/receive correspondence

Highly Domain Specific



Documents

- Problems:
 - Only linear construction ✓
 - Programmer/Designer tasks not separated ✓
 - No Show/Receive correspondence
 - Legal/sensible HTML generated?

Documents

- Problems:
 - Only linear construction ✓
 - Programmer/Designer tasks not separated ✓
 - No Show/Receive correspondence ✓
 - Legal/sensible HTML generated?

Documents

- Problems:
 - Only linear construction ✓
 - Programmer/Designer tasks not separated ✓
 - No Show/Receive correspondence ✓
 - Legal/sensible HTML generated? (✓)

Future Plan

- Analyze generated HTML documents
 - with respect to:
 - HTML 3.0 / 4.0 / ...
 - DTD / DSD / ...
- Ensure that only “legal” documents are generated

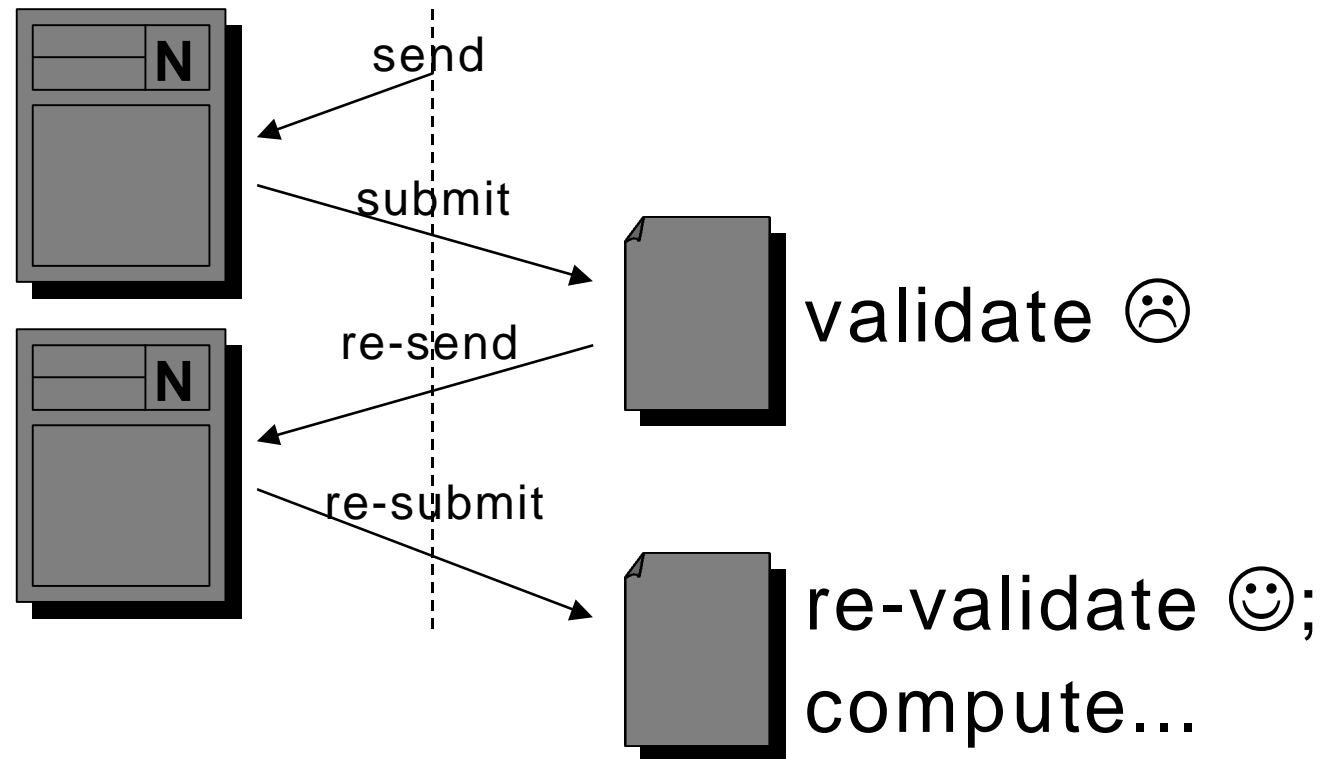
Plan

- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

Plan

- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

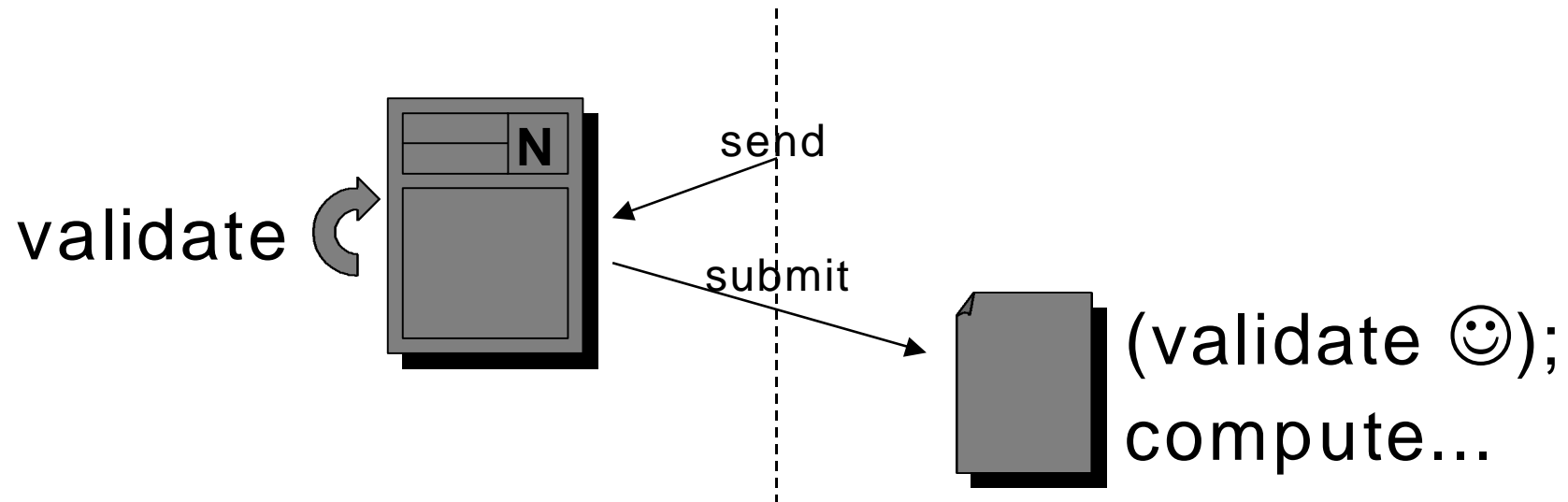
Server-side Input Validation



Drawbacks

- It takes time
- Excess network traffic
- Requires explicit programming
 - Affects all parties involved:
 - client
 - server
 - programmer

Client-side Input Validation



Drawbacks

- It takes time
- Excess network traffic
- Requires explicit programming

Drawbacks

- It takes time $\sqrt{\quad}$
- Excess network traffic
- Requires explicit programming

Drawbacks

- It takes time ✓
- Excess network traffic ✓
- Requires explicit programming

Drawbacks

- It takes time ✓
- Excess network traffic ✓
- Requires explicit programming:
 - re-showing of pages
 - actual validation

Drawbacks

- It takes time ✓
- Excess network traffic ✓
- Requires explicit programming:
 - re-showing of pages ✓
 - actual validation

Drawbacks

- It takes time ✓
- Excess network traffic ✓
- Requires explicit programming:
 - re-showing of pages ✓
 - actual validation ☹

Drawbacks

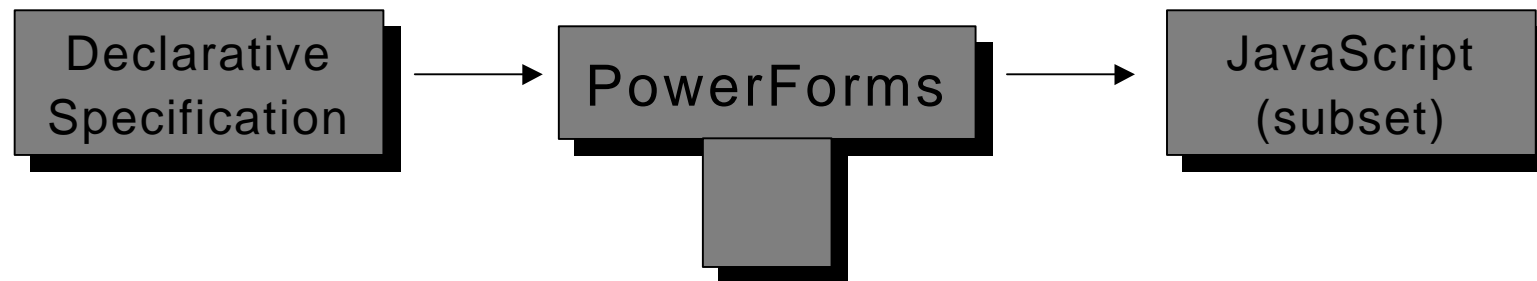
- It takes time ✓
 - Excess network traffic ✓
 - Requires explicit programming:
 - re-showing of pages ✓
 - actual validation ☹️
- Client, server: 😊
- Programmer: ☹️

Obvious Language: JavaScript

- Why avoid JavaScript?:
 - GPL for very specific task
 - Operational form
 - Diverging Browser Implementations:
 - Netscape vs. Explorer

Our Solution: *PowerForms*

- Domain specific language:
 - targeted uniquely for input validation
- Declarative nature (regexps):
 - abstracts away operational details



Syntax

- $decl ::= \underline{\text{format}} id = regexp ;$
- $regexp ::= id \mid \text{stringconst}$
 $\mid \text{union} (regexp^*)$
 $\mid \text{concat} (regexp^*)$
 $\mid \text{star} (regexp) \mid \dots$

`<input type="text" name="N" format="F">`

Example: Email Format

```
format Alpha = union(range('a','z'),range('A','Z'));  
format Word = ...;  
format Email = concat(Word,"@",Word,  
                        star(".",Word));
```

Example: EnterData (revisited)

html Input = <html>

name: <input name="name">

email: <input name="email">

</html>;

show Input receive [name=name,email=email];

Example: EnterData (revisited)

format **Email** = ...;

html Input = <html>

name: <input name="name">

email: <input name="email" **format="Email"**>

</html>;

show Input receive [name=name,email=email];

Field Interdependency

Have you attended past WWW conferences? Yes No

If Yes, how did WWW8 compare? Better Same Worse

...usually only handled on server-side

PowerForms (also)

- Extend (declarative specification):
 - formats depend on values of other fields
 - Update accordingly (fixed-point process):
 - text / password: status icons *updated*
 - radio / checkbox: illegal options *deselected*
 - select: illegal options *filtered* (and *deselected*)

Example Demos

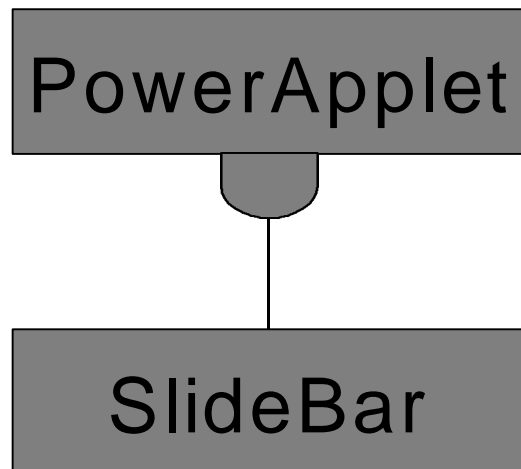
...speak for themselves...

- “Spouse”
 - Basic interdependency
- “Vowels and Consonants”
 - Select filtering
- “NYC Office”
 - Complex interdependency

Applets Integration

“Treat applets as regular input fields”

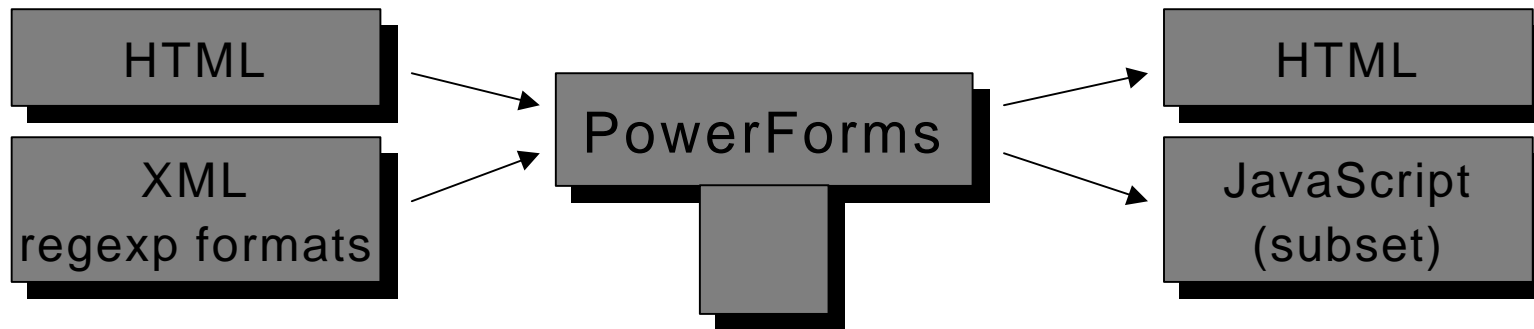
Java:



(PowerForms) HTML:

```
<applet code="SlideBar.class">
  <param name="low" value="32">
  <param name="high" value="212">
  <result name="choice">
</applet>
```

PowerForms: Stand-alone Tool



Plan

- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

Plan

- Introduction
- Runtime System
- Dynamic Documents
- PowerForms
- Conclusion

<bigwig> Publications

- <bigwig> *...submitted*
 - Runtime System WWW 8, Toronto
 - Concurrency Control FASE'98, Lisbon
 - Database (IPL'92)
 - Dynamic Documents POPL'00, Boston
 - PowerForms *...submitted*
 - Macros *...underway*
- Planned:
 - Security / Cryptographic security / Workflow

Professional Tool

- BUT,
 - Also a research project
- Not (yet) ideal for:
 - Heavy access
 - Runtime system: CGI → Spec. Web Server
 - Huge datasets
 - External database

Availability?

- <bigwig>: 1.5 MB C source
 - for UNIX/Linux
 - Also as stand-alone packages:
 - Runtime System
 - PowerForms
- License?
 - “Almost do what you want”

YOU:

- Free interesting tool
 - Use all of <bigwig> *or*
 - Use stand-alone packages
 - Runtime System
 - PowerForms
- Look out for future developments

US:

- Feedback
- New ideas
- Spread the word!

What is <bigwig>?

- Runtime System
- Concurrency Control
- Database
- Dynamic Documents
- PowerForms
- Security / Cryptographic security
- Syntactic-level Macros

<http://www.brics.dk/bigwig/>