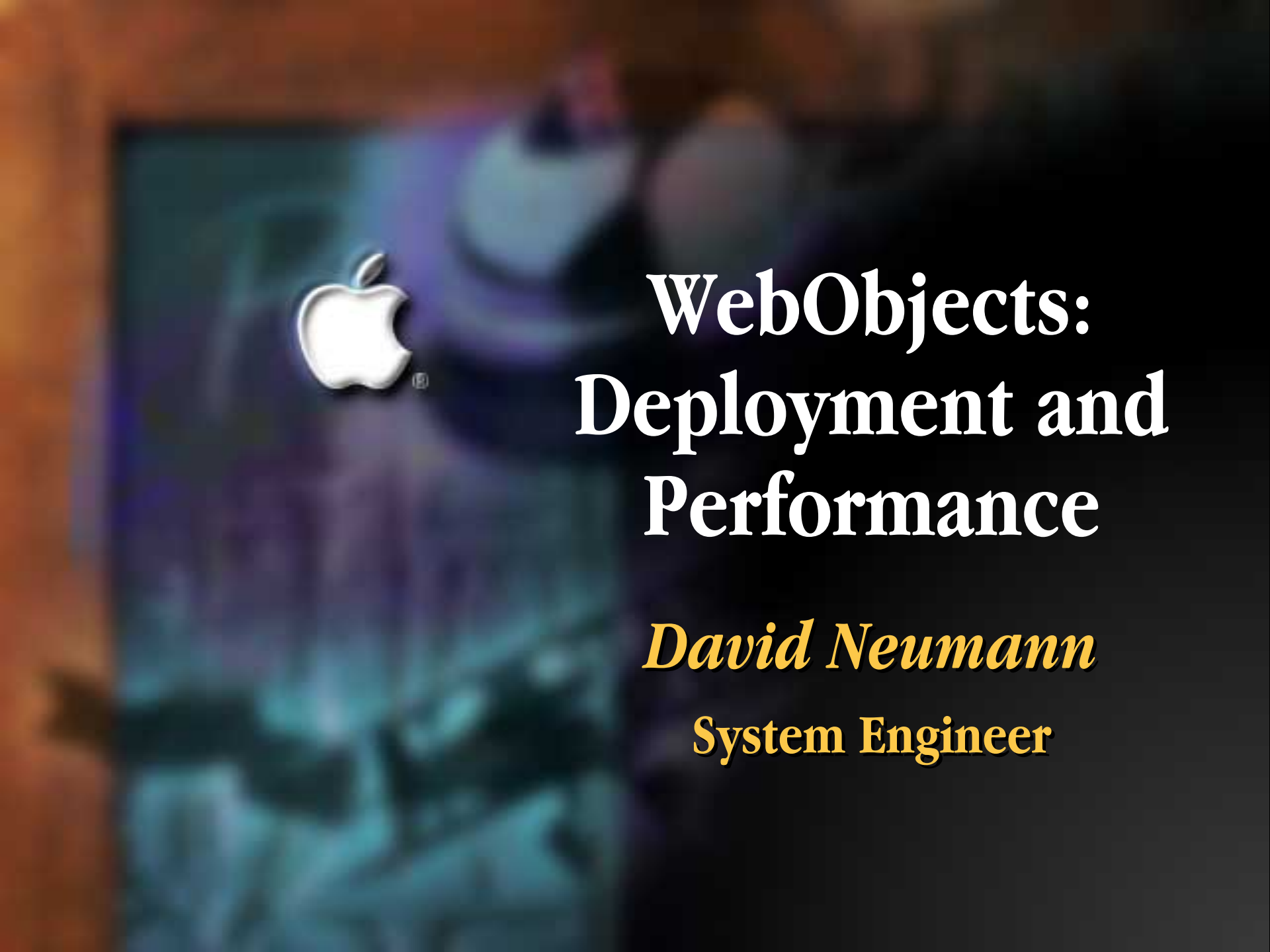

Worldwide
Developers
Conference



WebObjects: Deployment and Performance

David Neumann

System Engineer

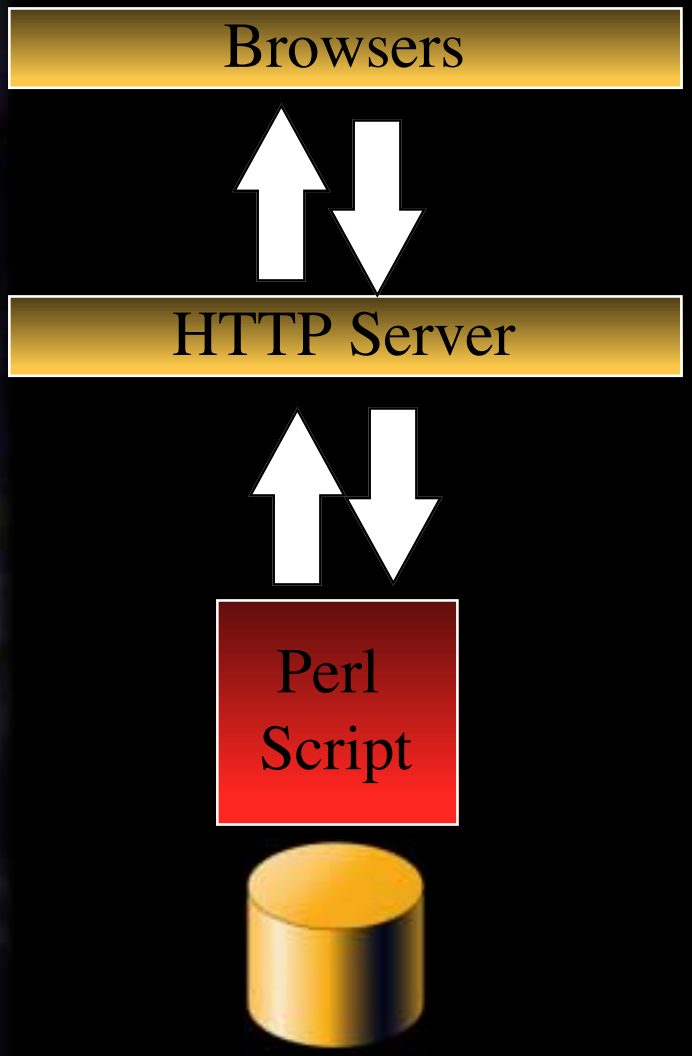
Two Kinds of Scalability

- Logical scalability
  - If you have it, you can manage the largest and most complex of applications
- Physical scalability
  - If you have it, you can provide the largest number of users with acceptable response times

Overall Deployment Architecture

The Well-known CGI Problem

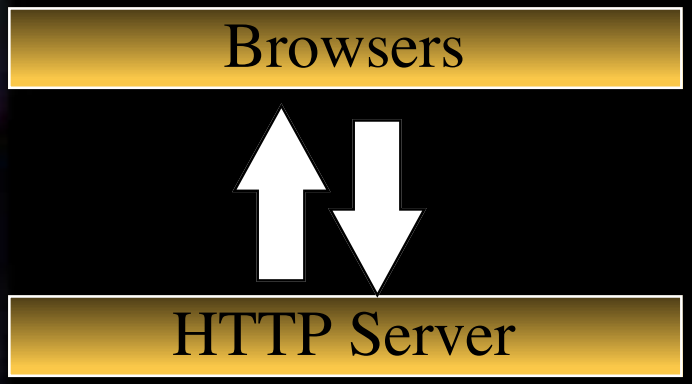
Your app as a Perl script



- HTTP Servers launch CGI processes for every request

The Well-known CGI Problem

Your app as a Perl script

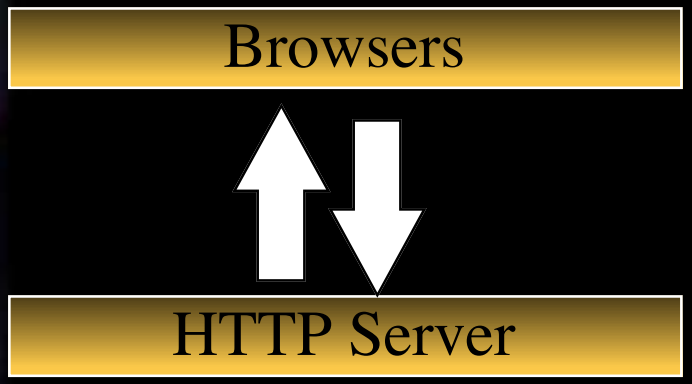


- HTTP Servers launch CGI processes for every request

- When the request is over, the CGI process ends

The Well-known CGI Problem

Your app as a Perl script



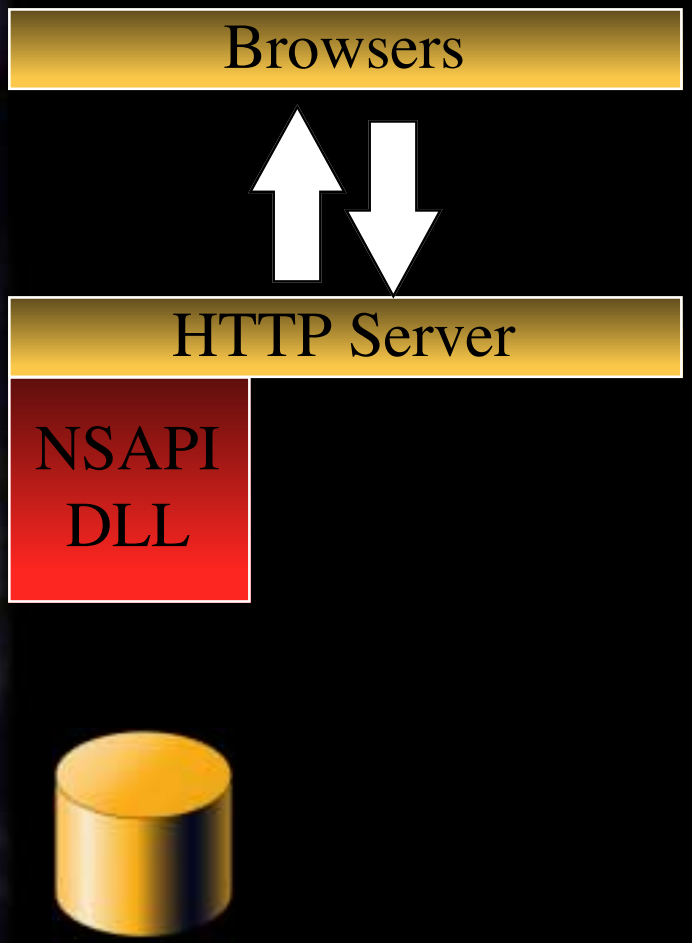
- What about state?

- What about massive overhead to essentially restart your application for every user event?

- What about database connections?

The Server API "Solution"

Your app as a server plug-in



BUT…

1) You are now dependent on a proprietary API

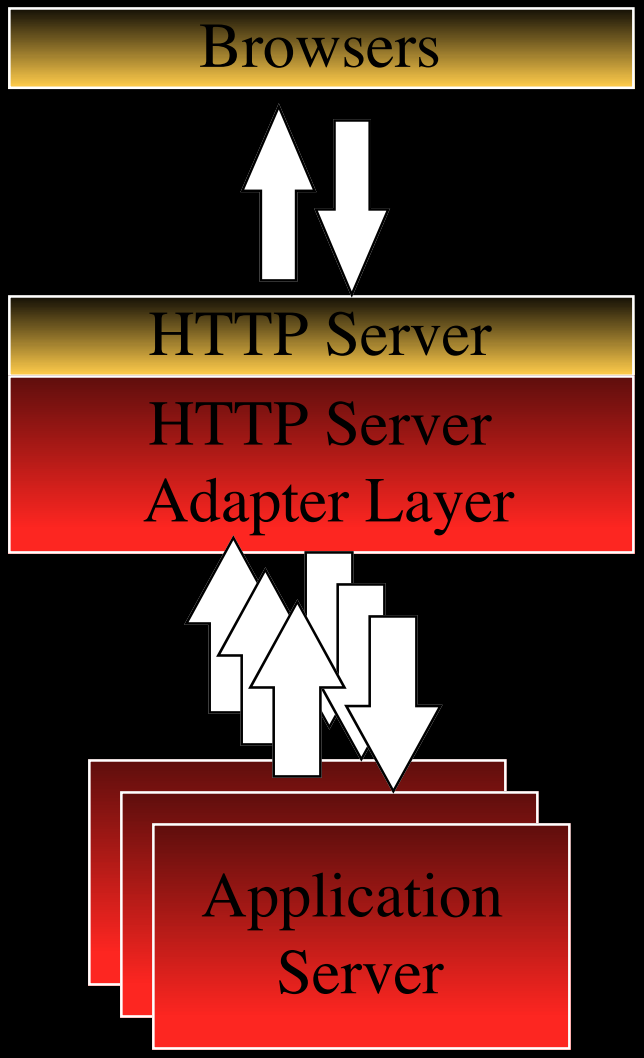
2) Like CGI, can't scale to processors other than those running the HTTP Server

3) Managing state still an issue

4) A failure in one thread will not only take out that thread but also potentially ***the entire site***

A Better Solution

Your app as a WebObjects Server



Use an intermediate load balancing layer

Advantages:

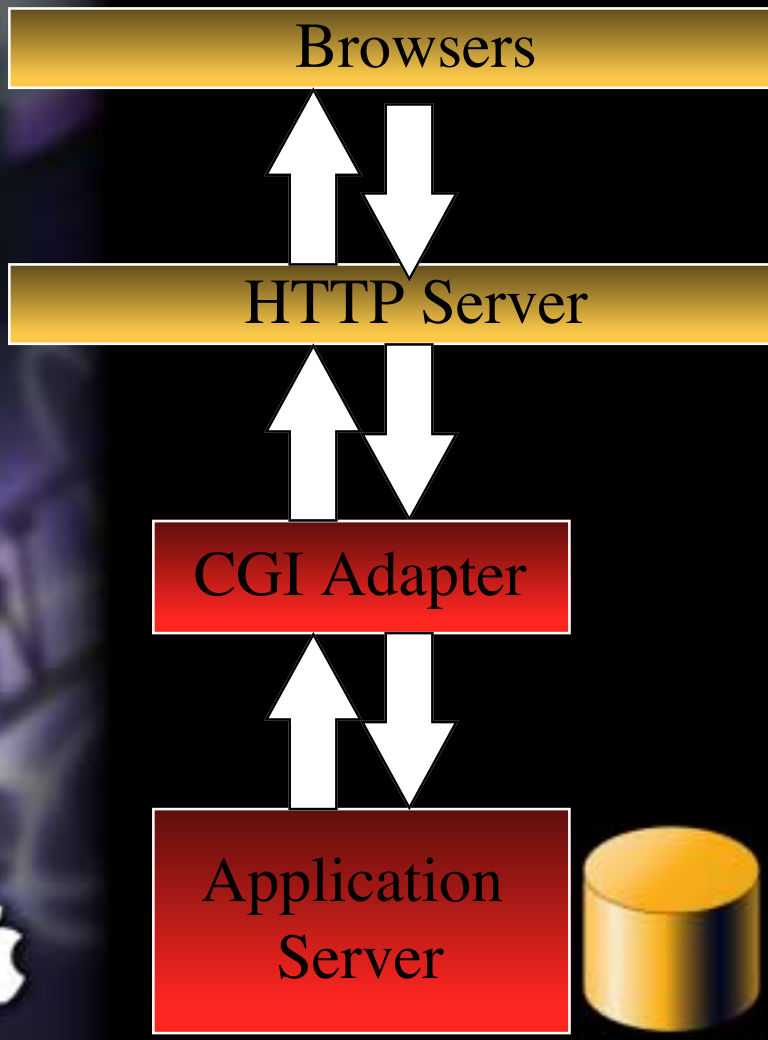
- Application servers stay up and preserve their connections

- You can have as many Application server processes as you need …but no more than you don't

- You can cluster application servers on multiple machines to offer linear horse power increases

- A problem in one process cannot affect any other process

How It Works with WebObjects…



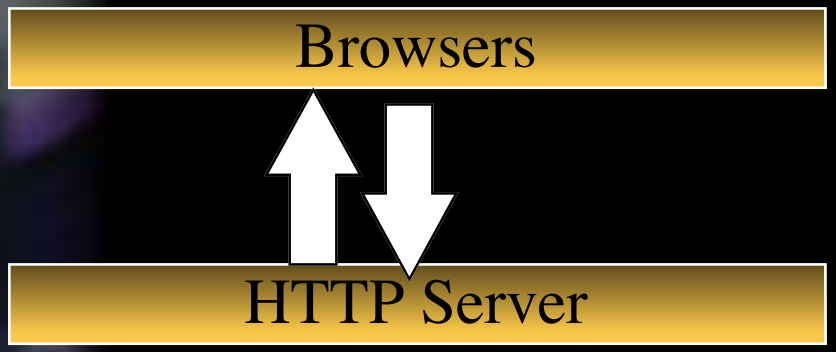
With CGI …when a browser sends a request to the server:

1) WebObjects CGI Adapter launched by HTTP Server

2) Adapter forwards request to existing Application Server

3) Application sends response to CGI Adapter when it's done processing

How It Works with WebObjects…



With CGI …when a browser sends a request to the server:

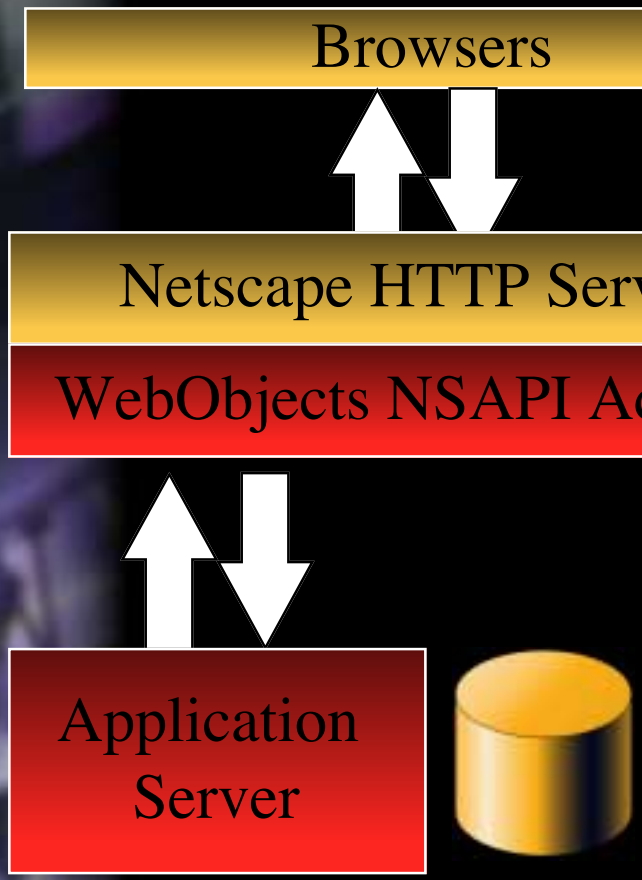
1) WebObjects CGI Adapter launched by HTTP Server

2) Adapter forwards request to existing Application Server

3) Application sends response to CGI Adapter when it's done processing

CGI process closes …but Application Server does not

How It Works with WebObjects…

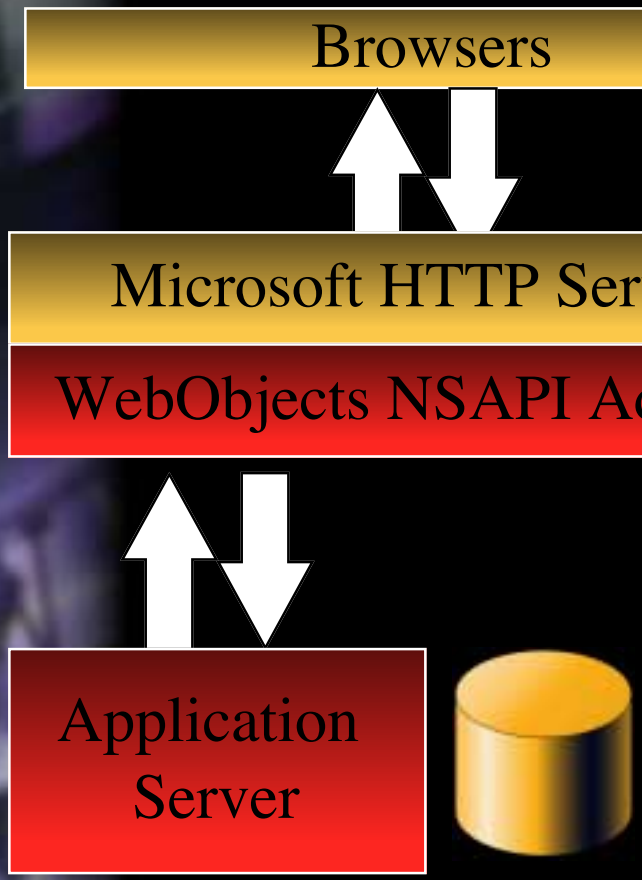


With HTTP Server APIs …

Use either:

- The API from Netscape

How It Works with WebObjects…



With HTTP Server APIs …

Use either:

- The API from Netscape

or…

- The API from Microsoft

- You never write against any proprietary API

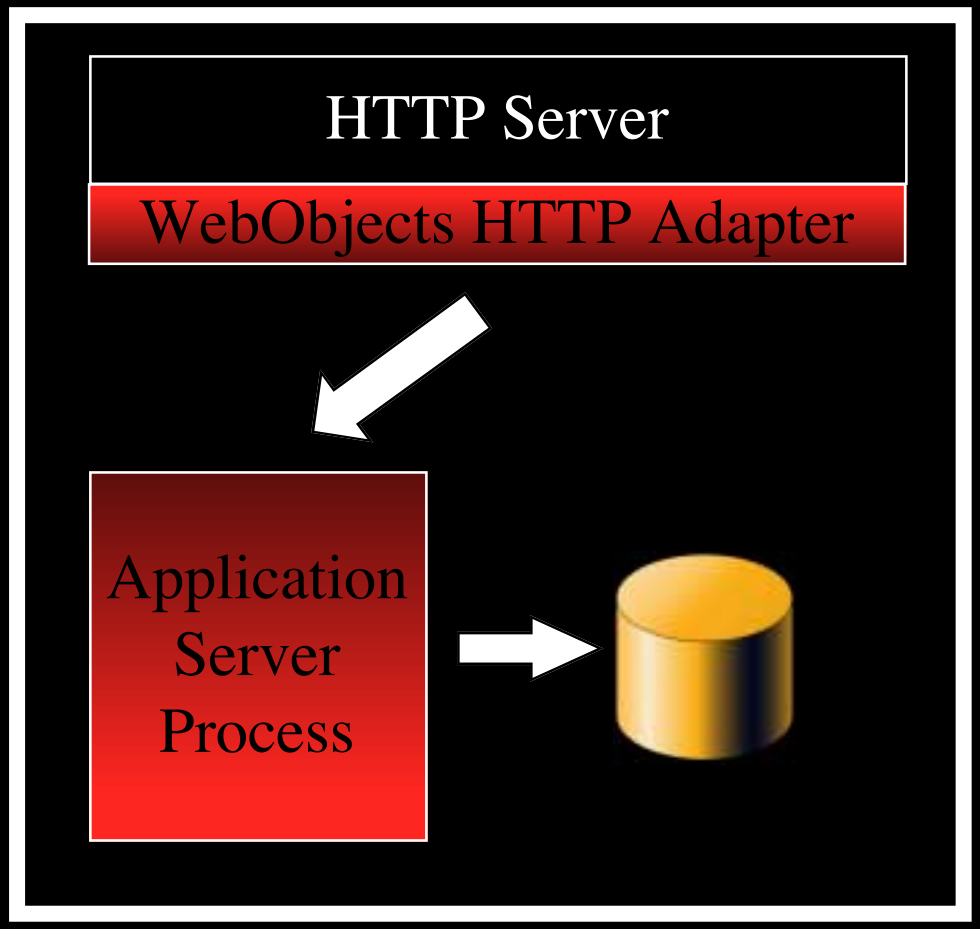
- No CGI process launching overhead at all

- Server API offer ultimate in transactional throughput

Physical Scalability

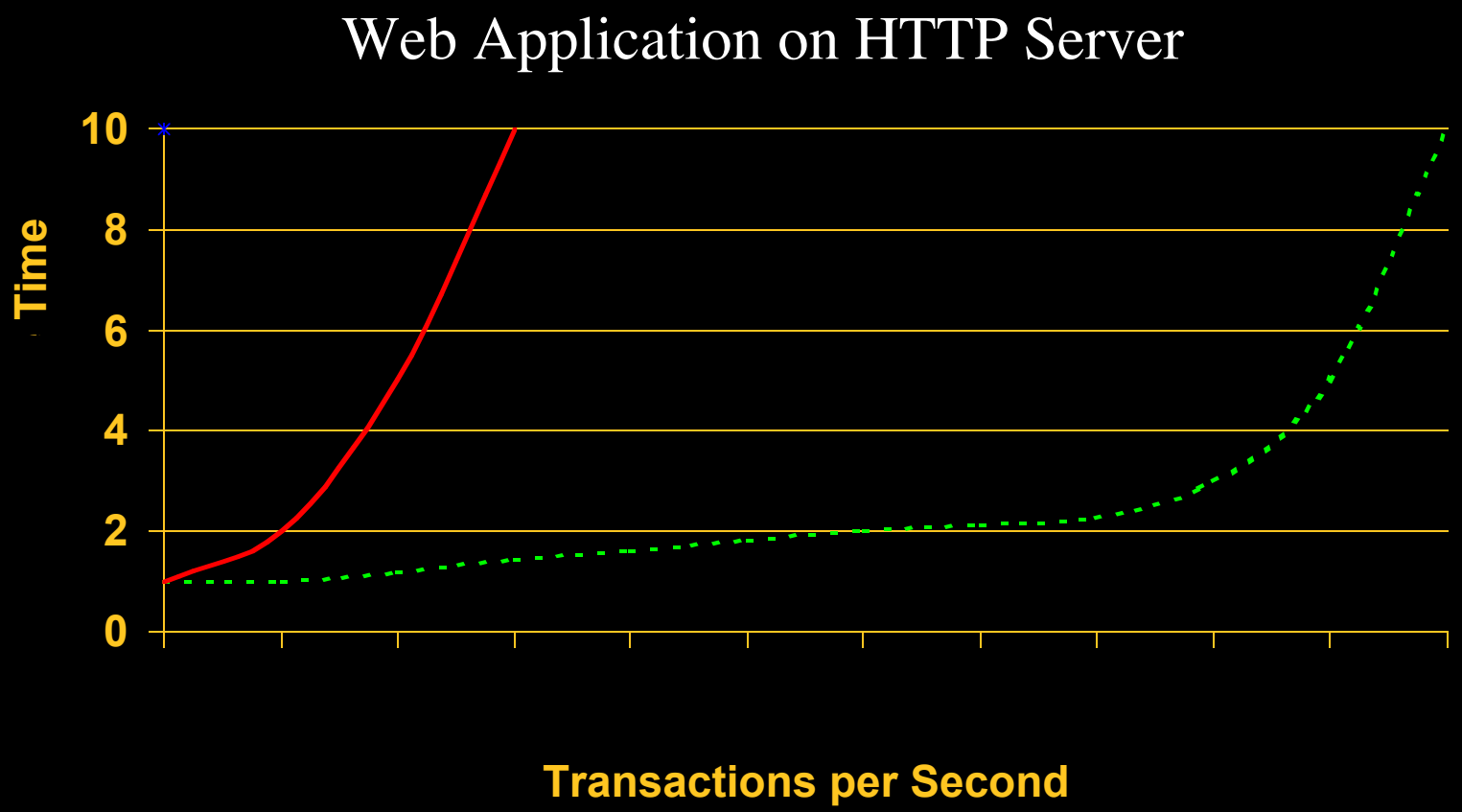
- Transaction throughput

Your App Handling a Workgroup

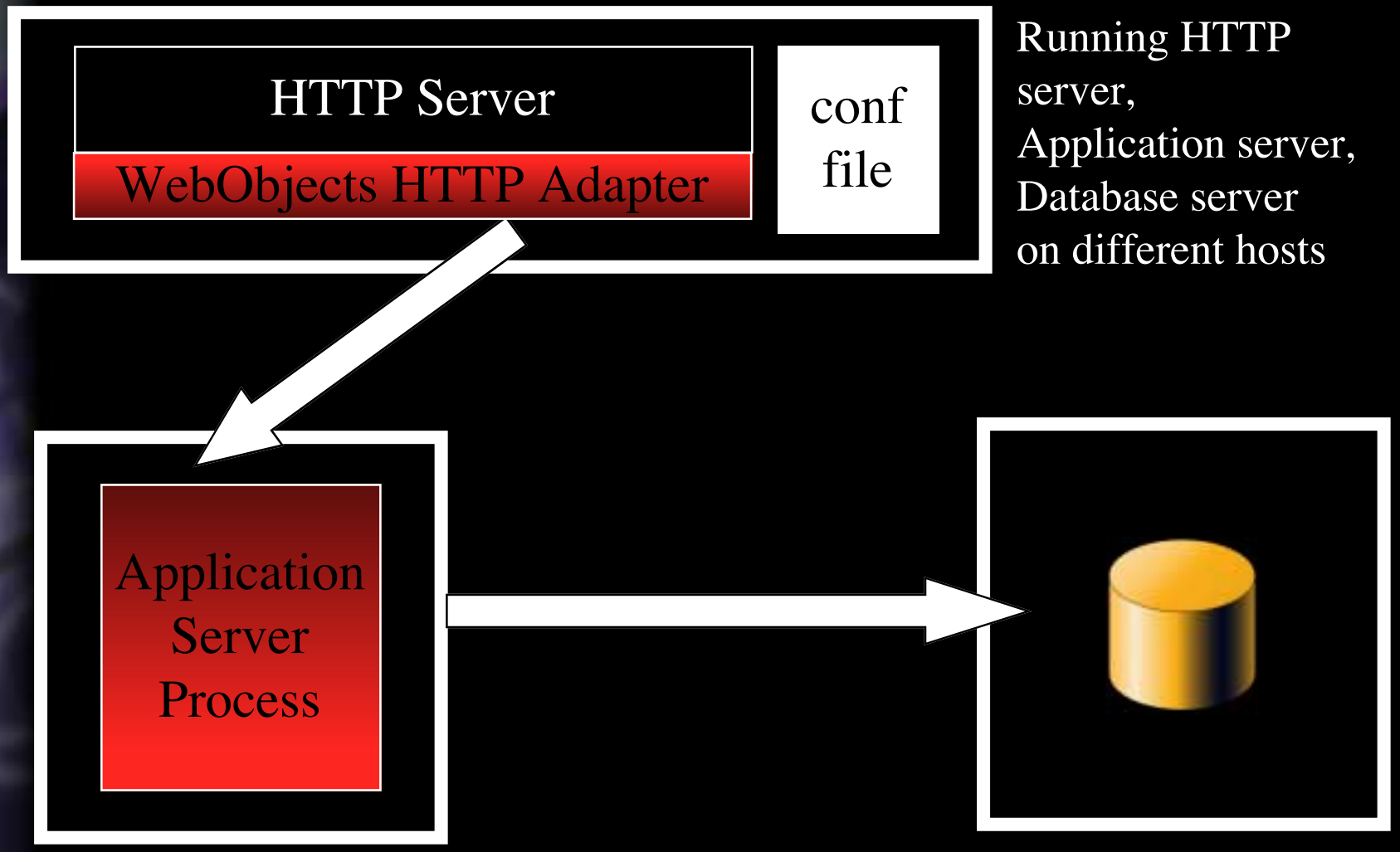


Running HTTP server,
Application server,
Database server
on same host

Response Time Curves

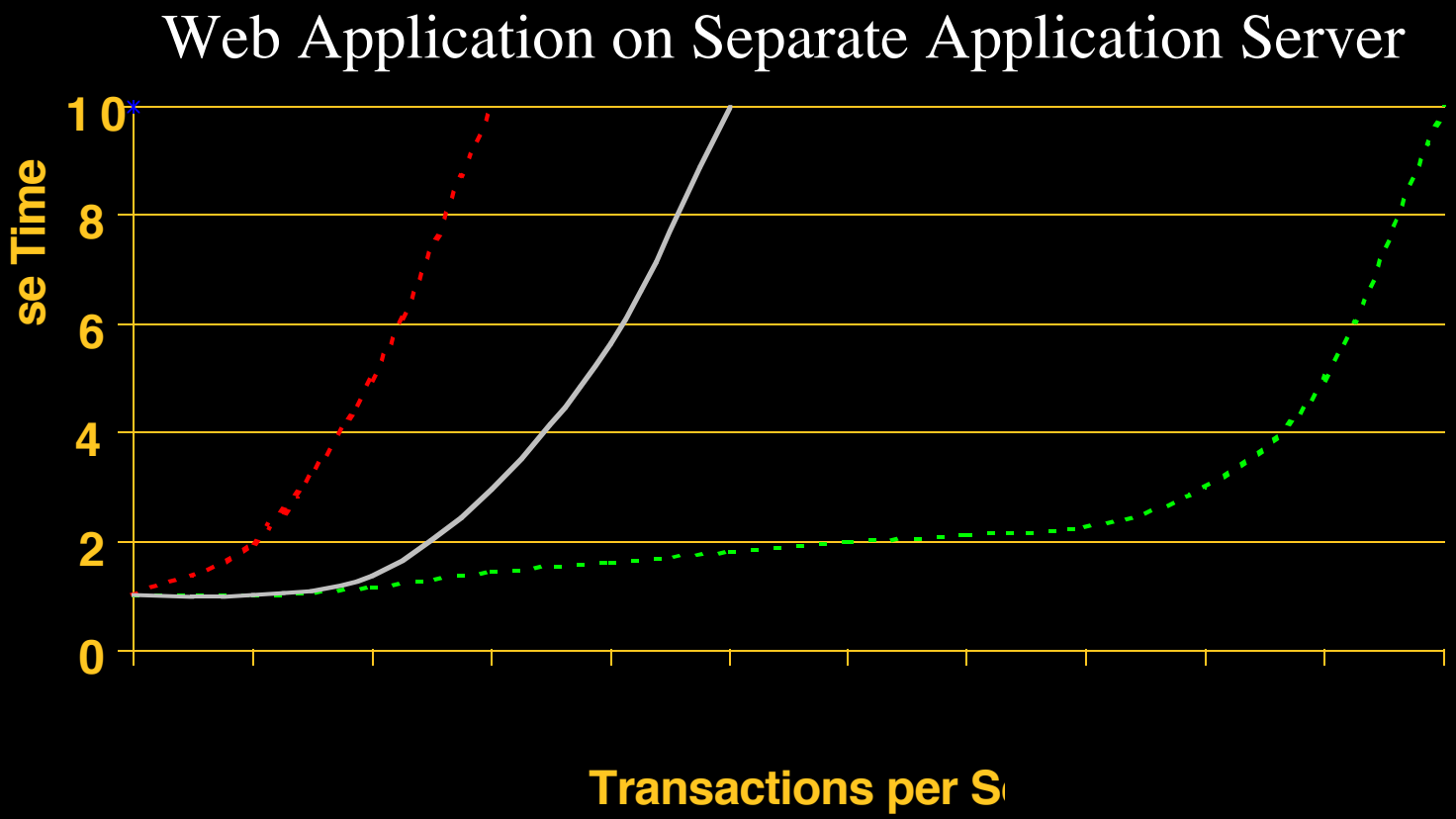


Scaling to More Users

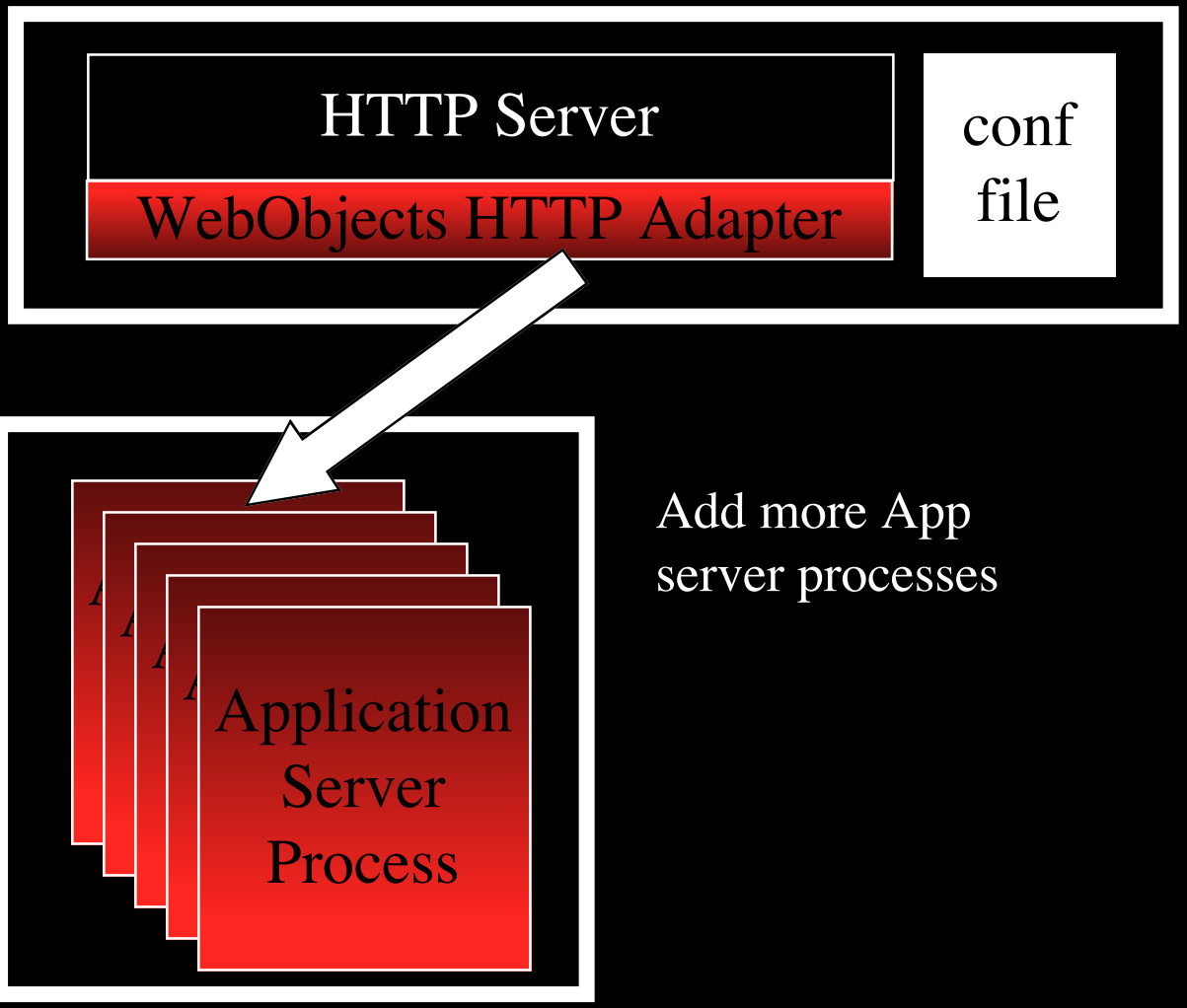


Running HTTP server,
Application server,
Database server
on different hosts

Response Time Curves

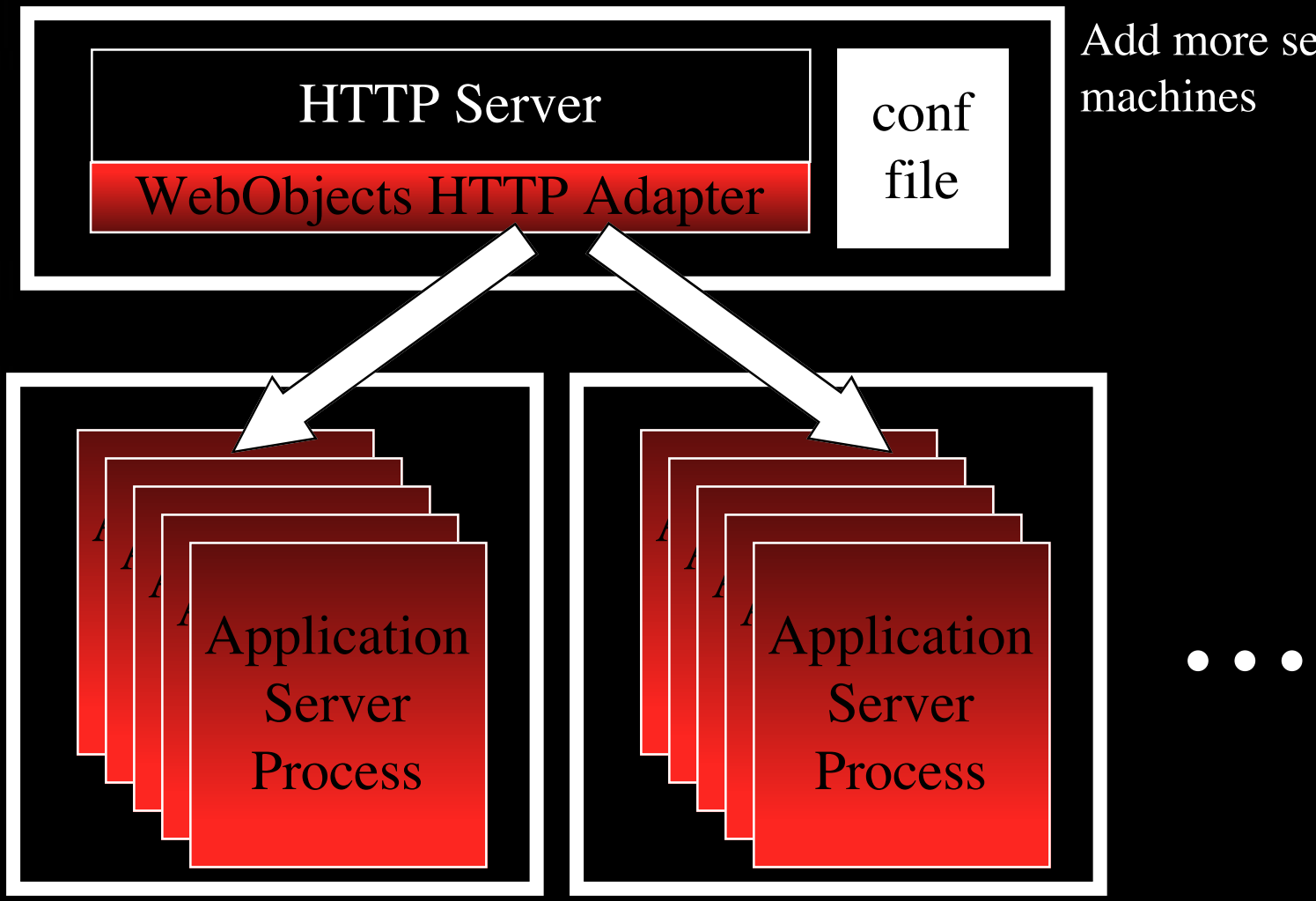


Scaling to More Users



Add more App server processes

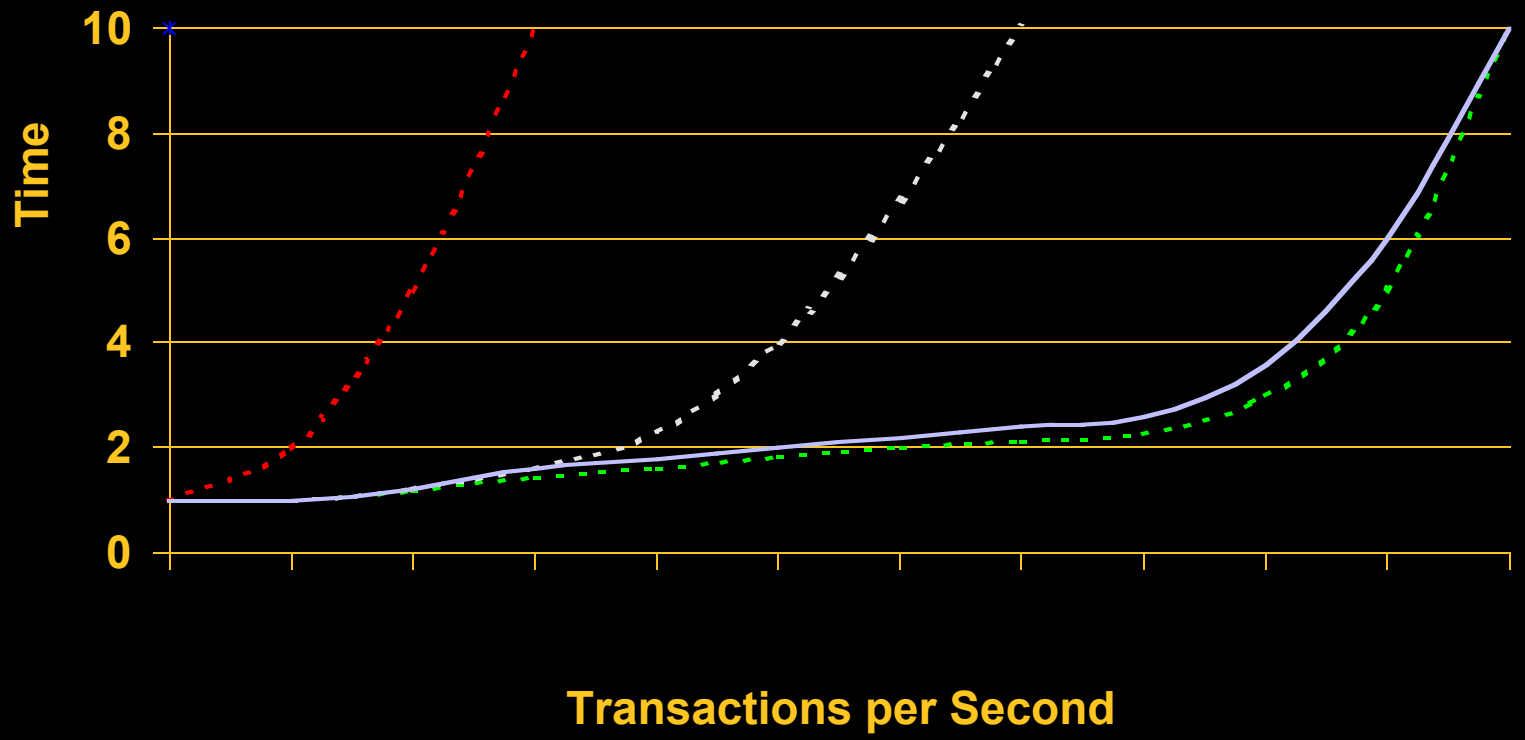
Scaling to Even More Users



Add more server machines

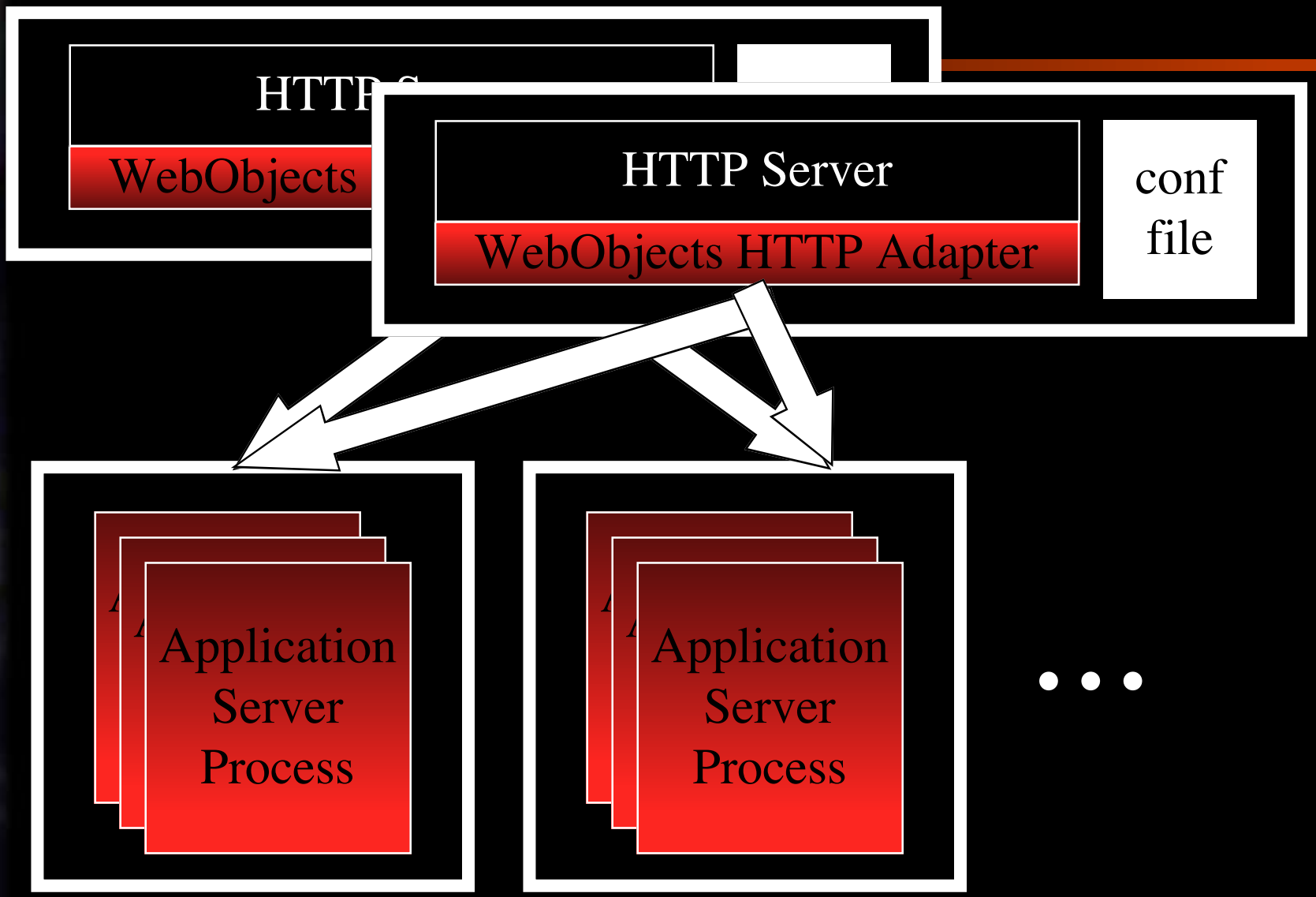
Response Time Curves

WebObjects Application Distributed to Multiple Servers



Approaches maximum throughput of HTTP server

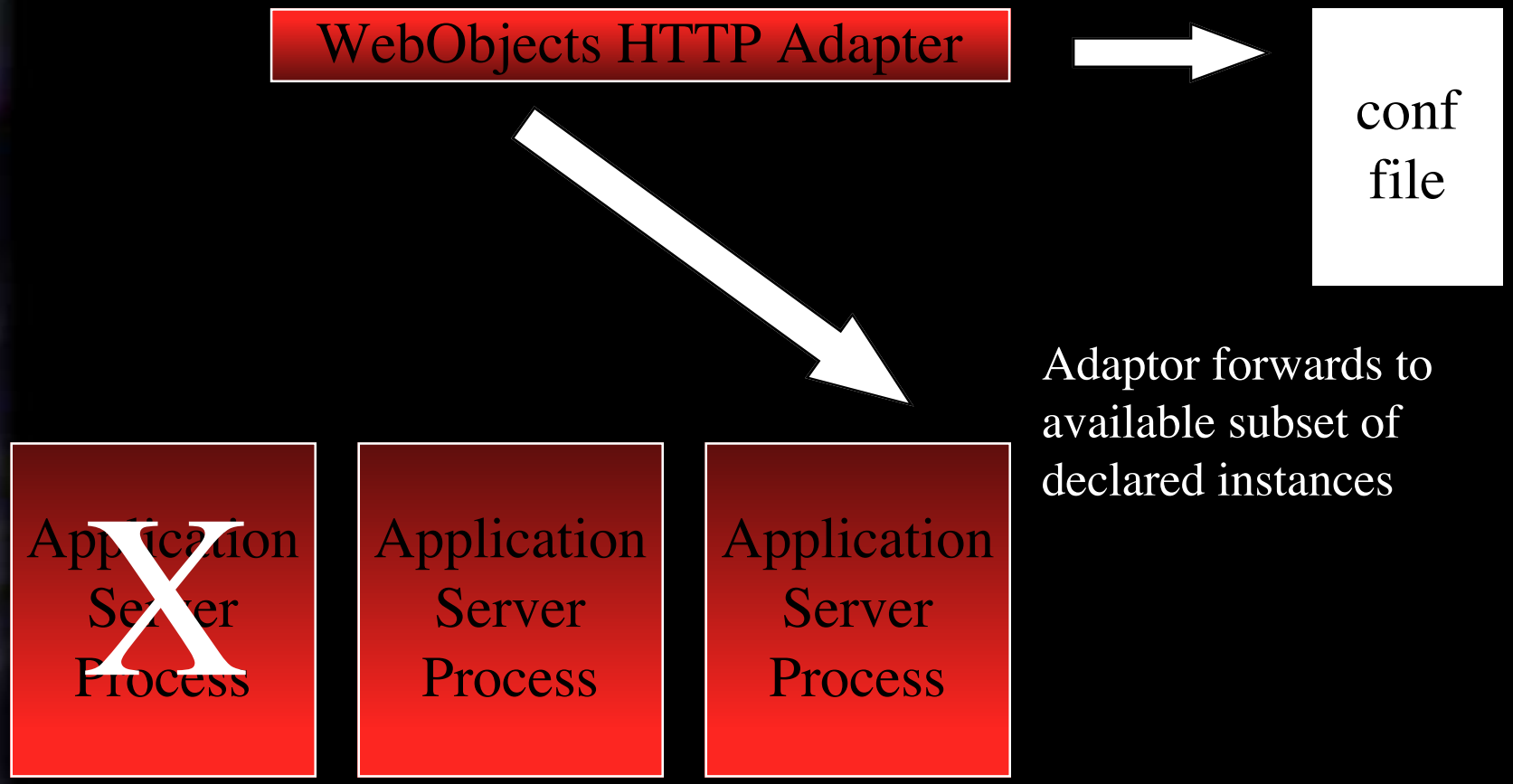
If the HTTP Server Bottlenecks…



Physical Scalability

- Transaction throughput
- Reliability

Fail Over at Adaptor Level

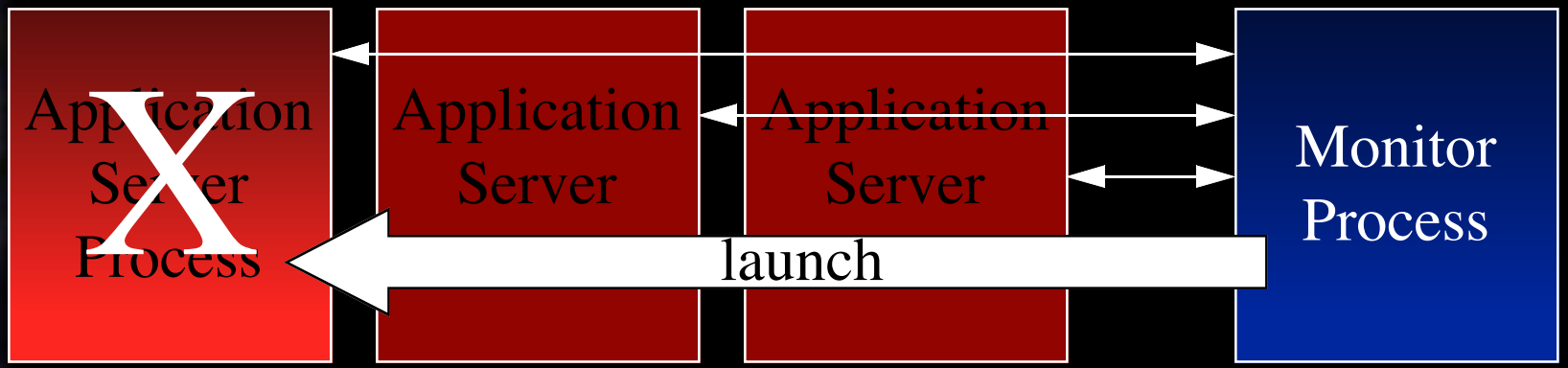


Fail Over at Monitor Level

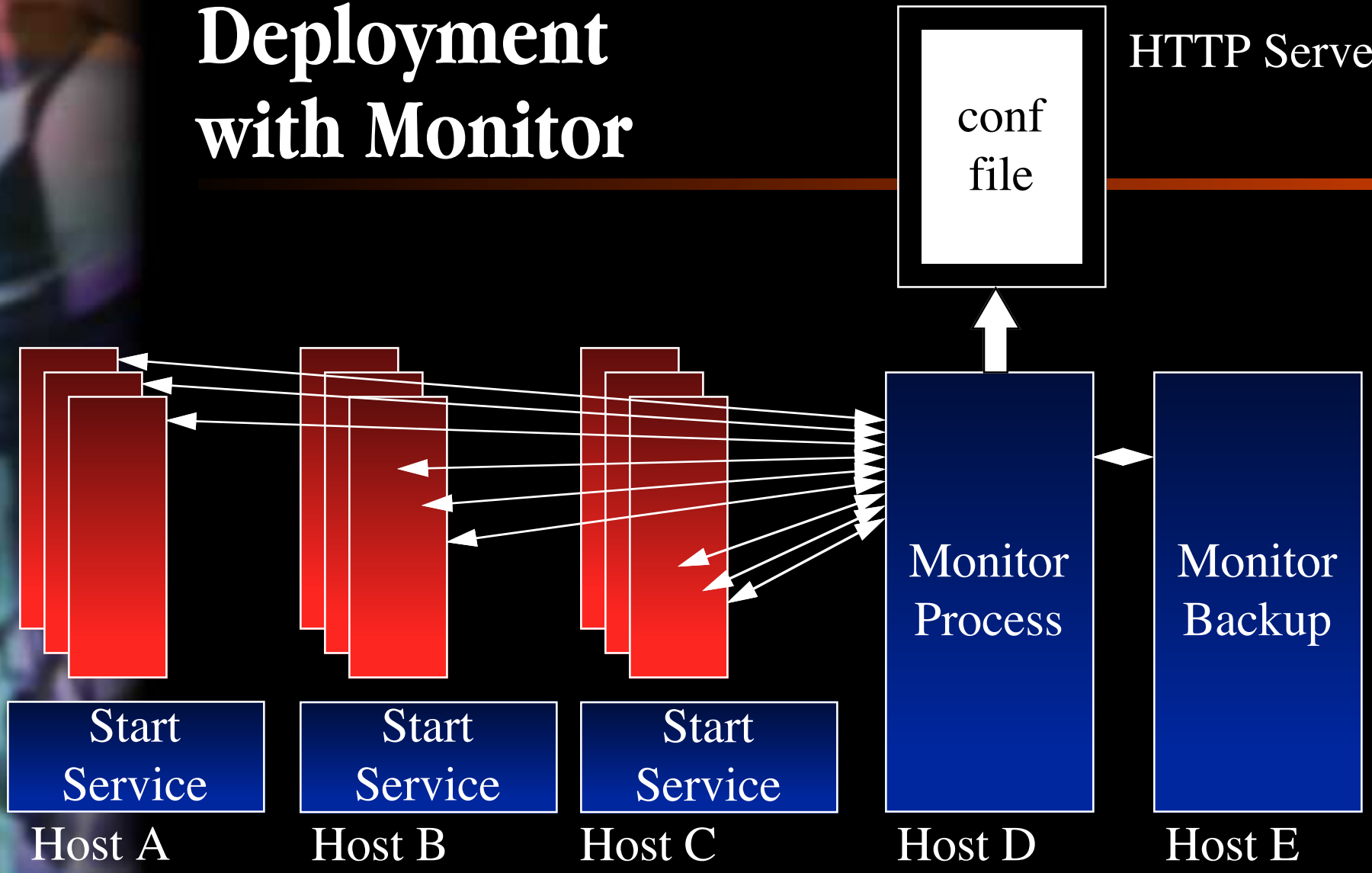
WebObjects HTTP Adapter

conf file

Monitor detects death and will immediately attempt an instance restart



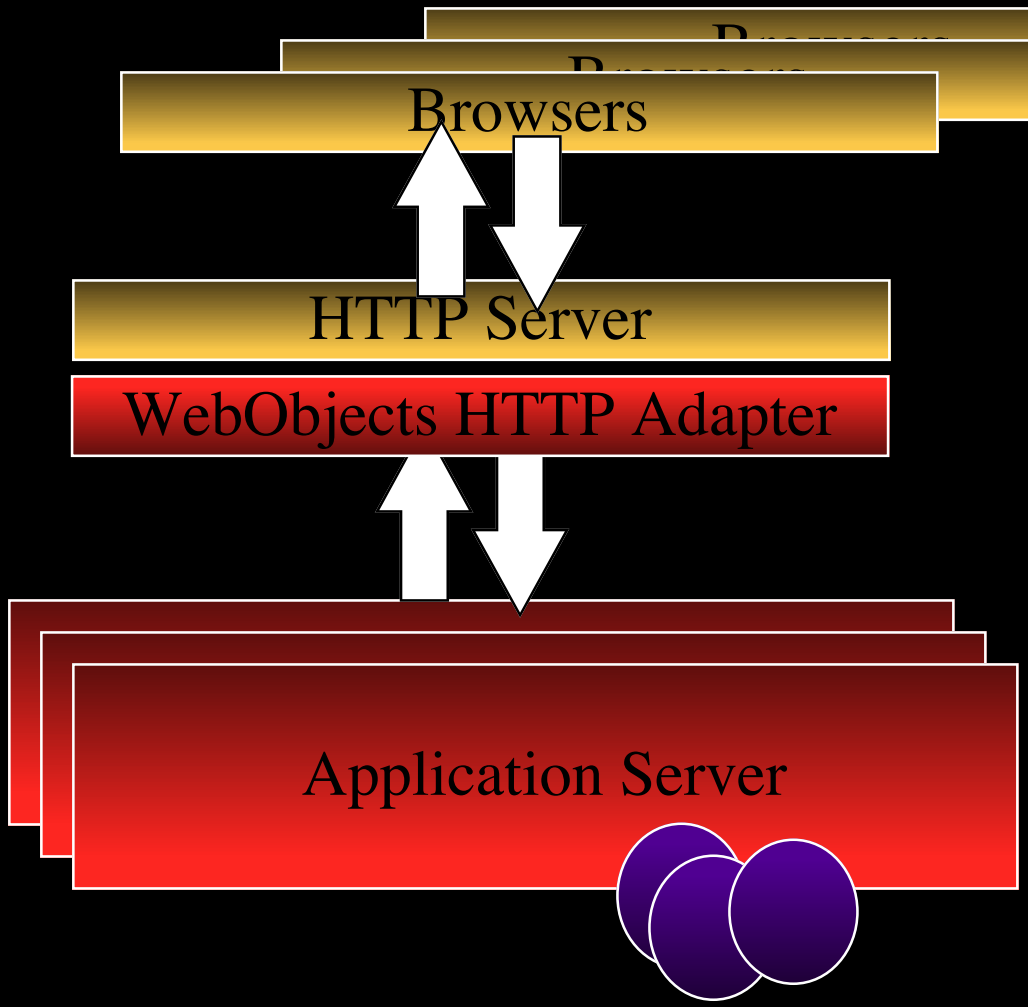
Deployment with Monitor



Physical Scalability

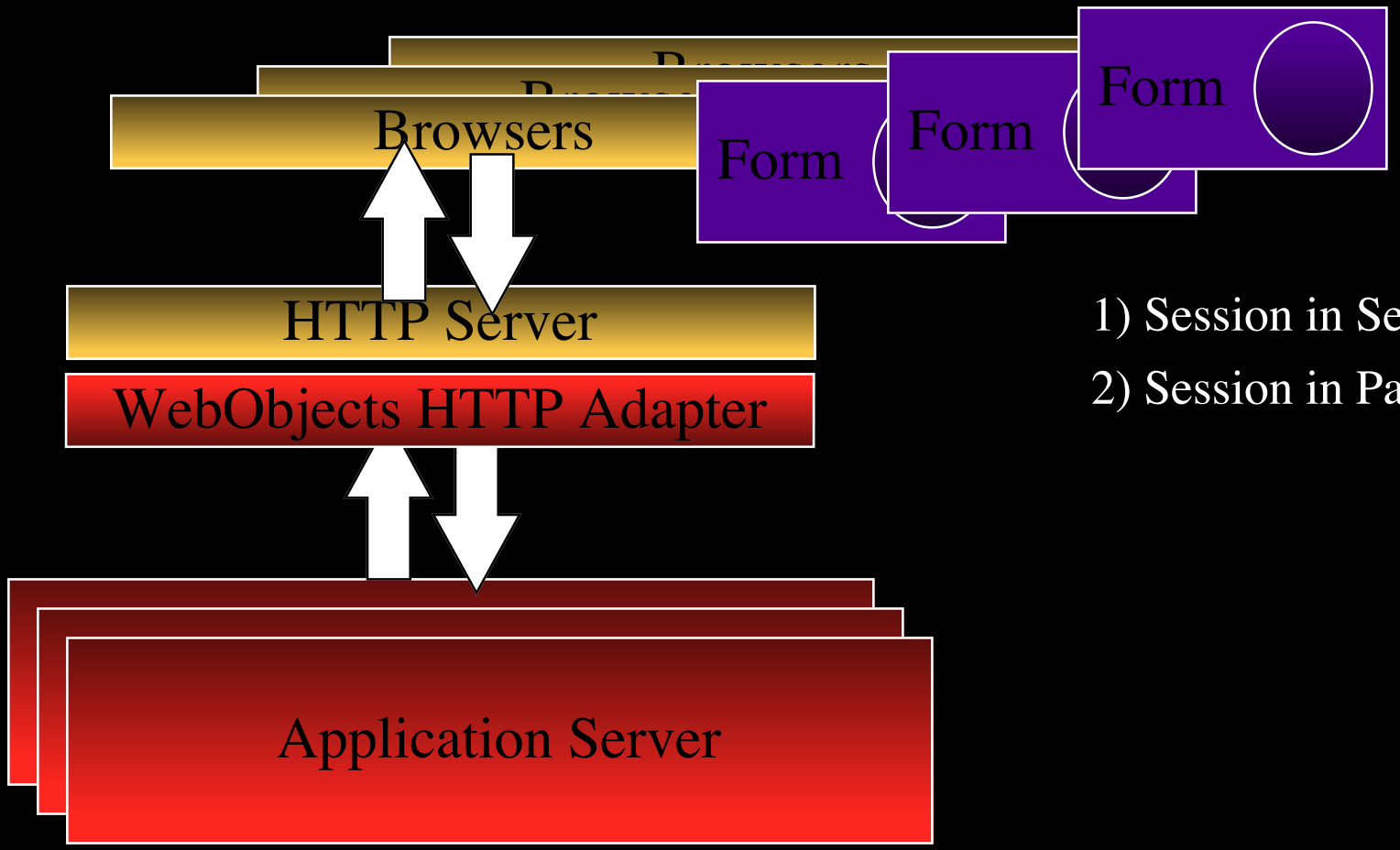
- Transaction throughput
- Reliability
- High performance state management architecture

State Management Policy



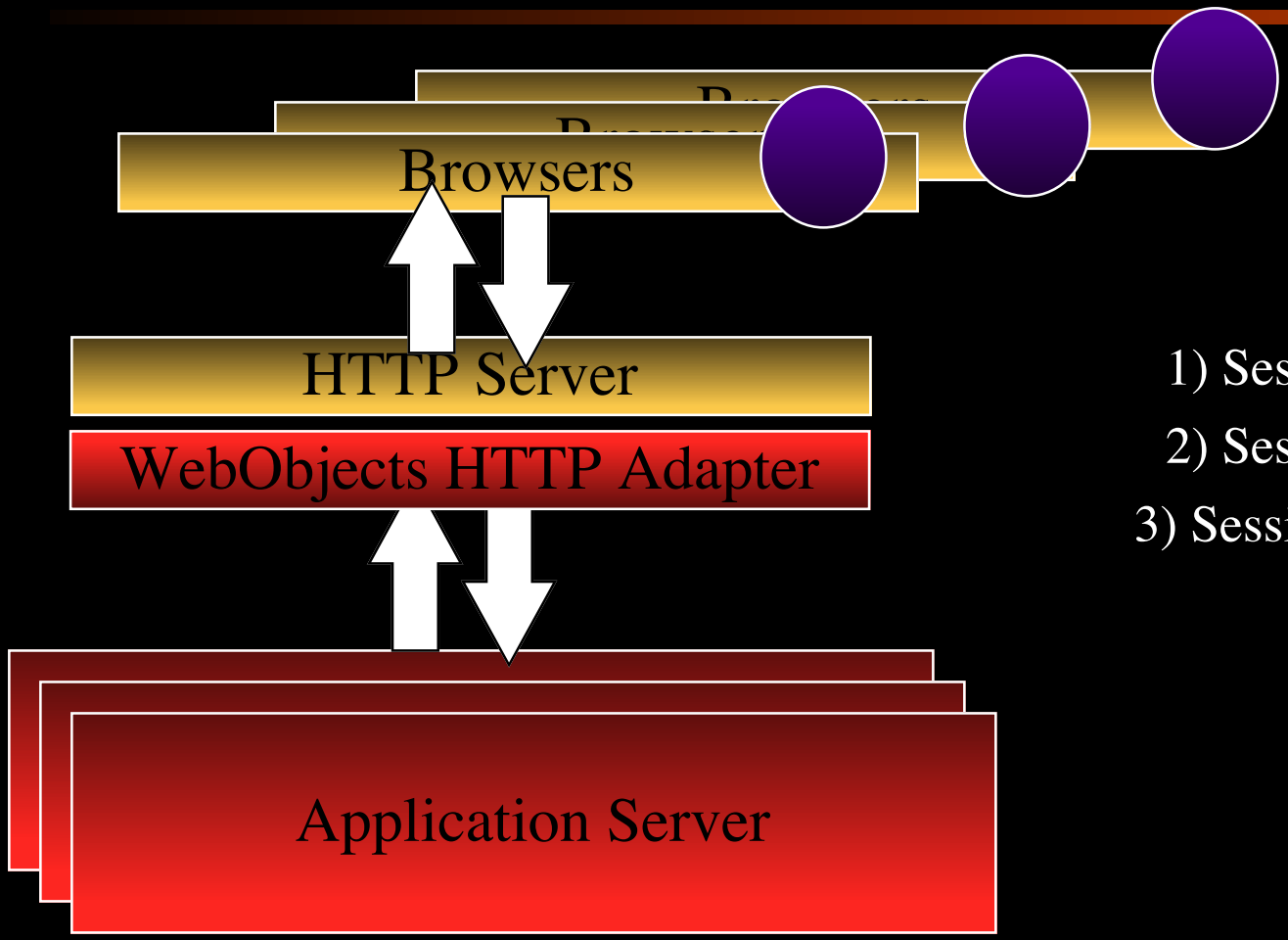
1) Session in Server

State Management Policy



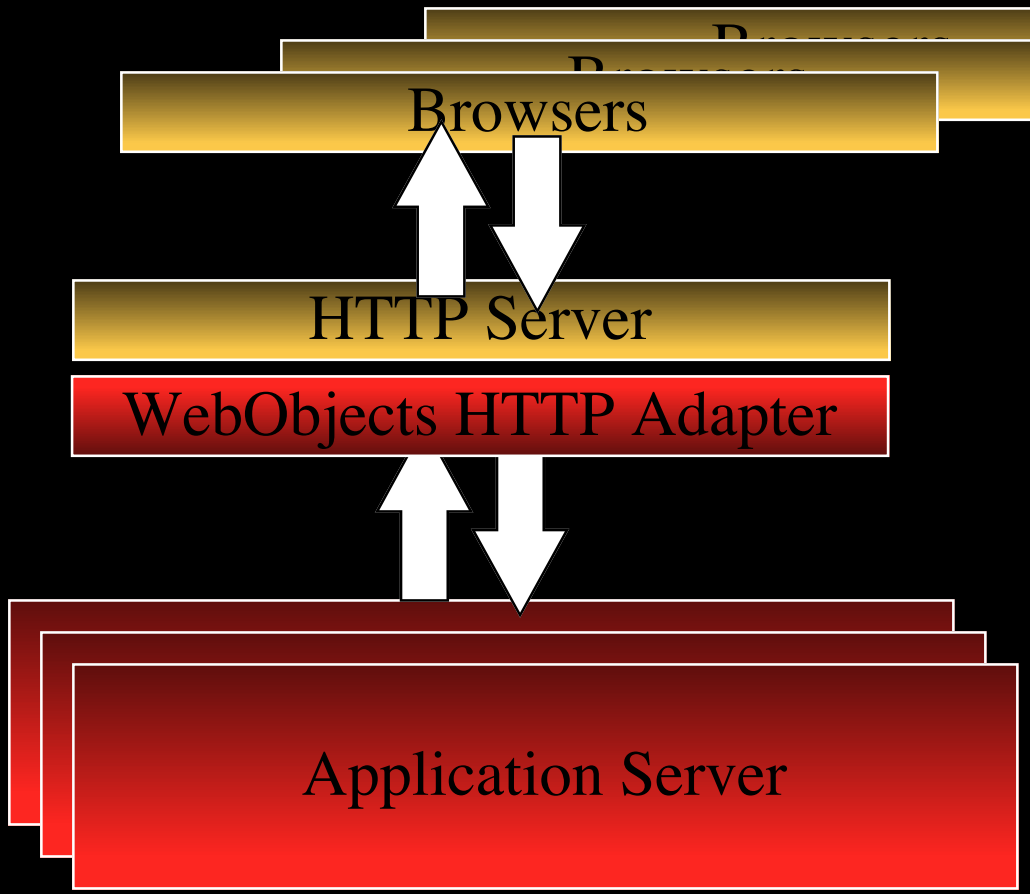
1) Session in Server
2) Session in Page

State Management Policy



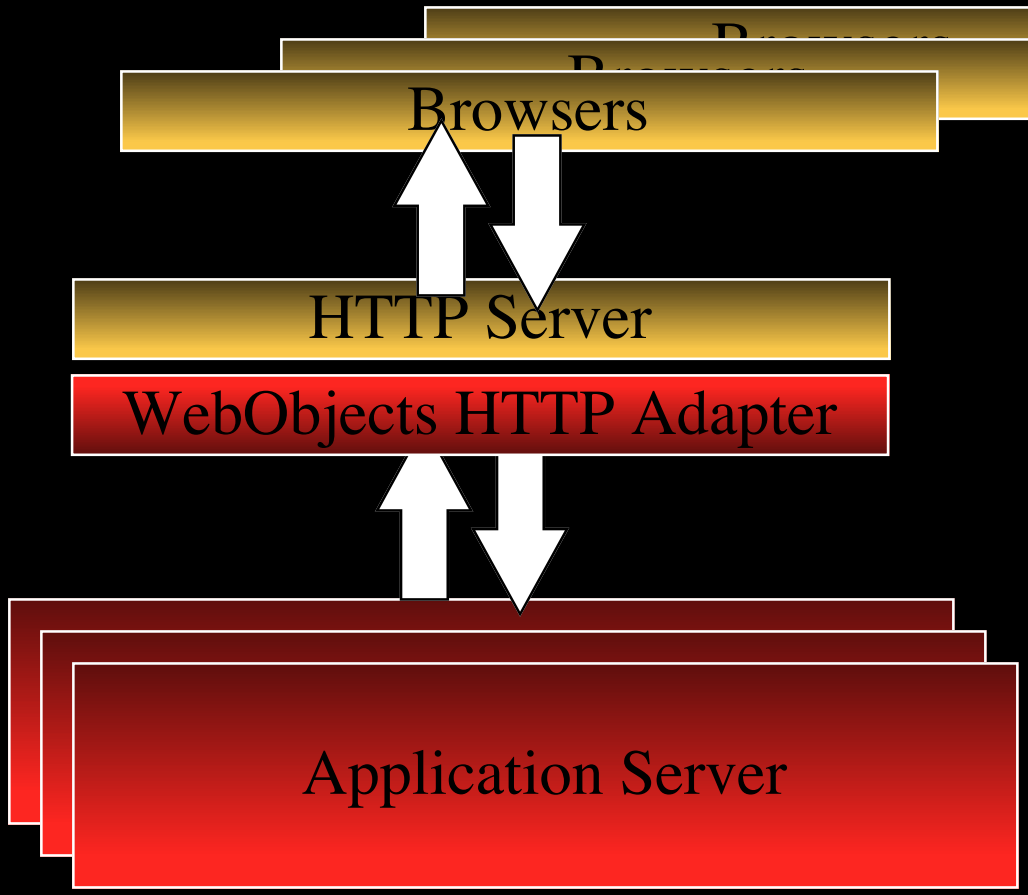
1) Session in Server
2) Session in Page
3) Session in Cookies

State Management Policy

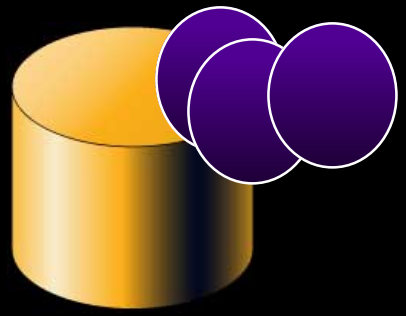


1) Session in Server
2) Session in Page
3) Session in Cookies
4) Session in File System

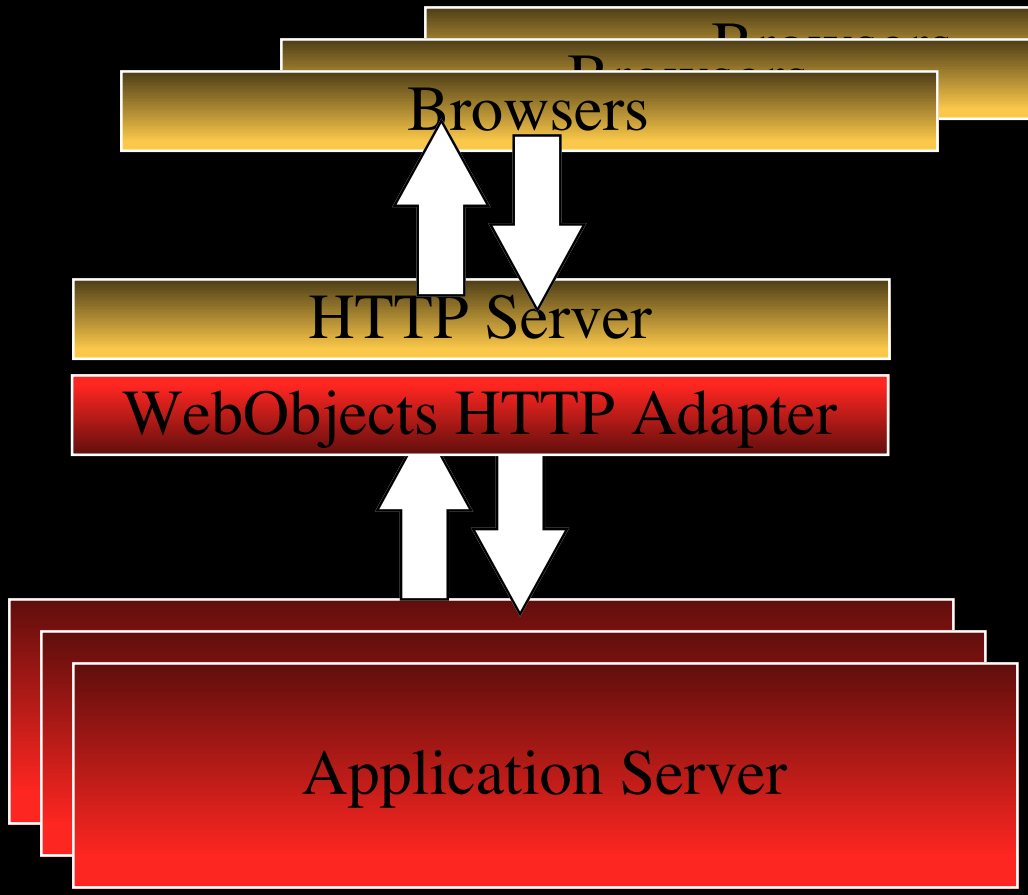
State Management Policy



1) Session in Server
2) Session in Page
3) Session in Cookies
4) Session in File System
5) Session in Database



State Management Policy



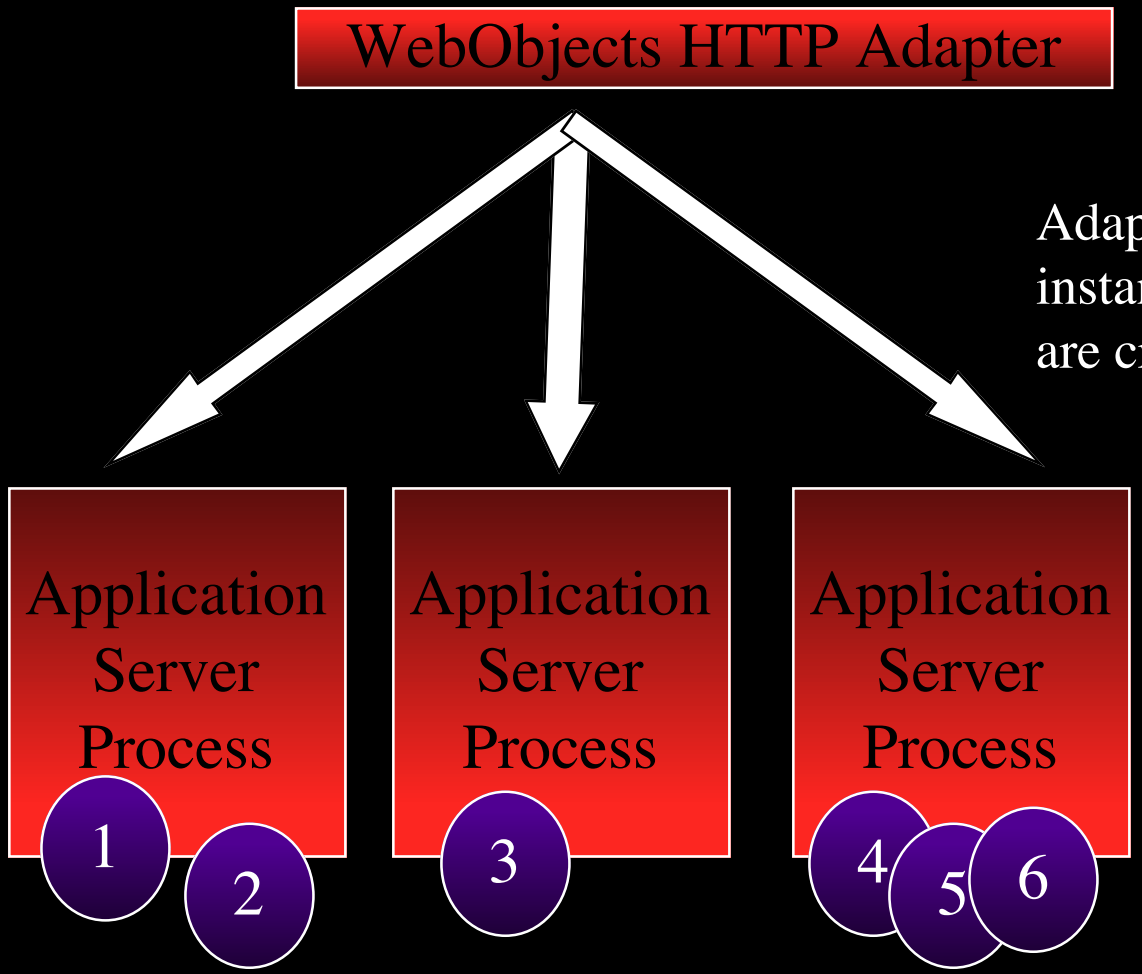
1) Session in Server
2) Session in Page
3) Session in Cookies
4) Session in File System
5) Session in Database
6) Other

State Policy and Load Balancing

State Management Policy can affect:

1) Granularity of load balancing
2) Availability of sessions

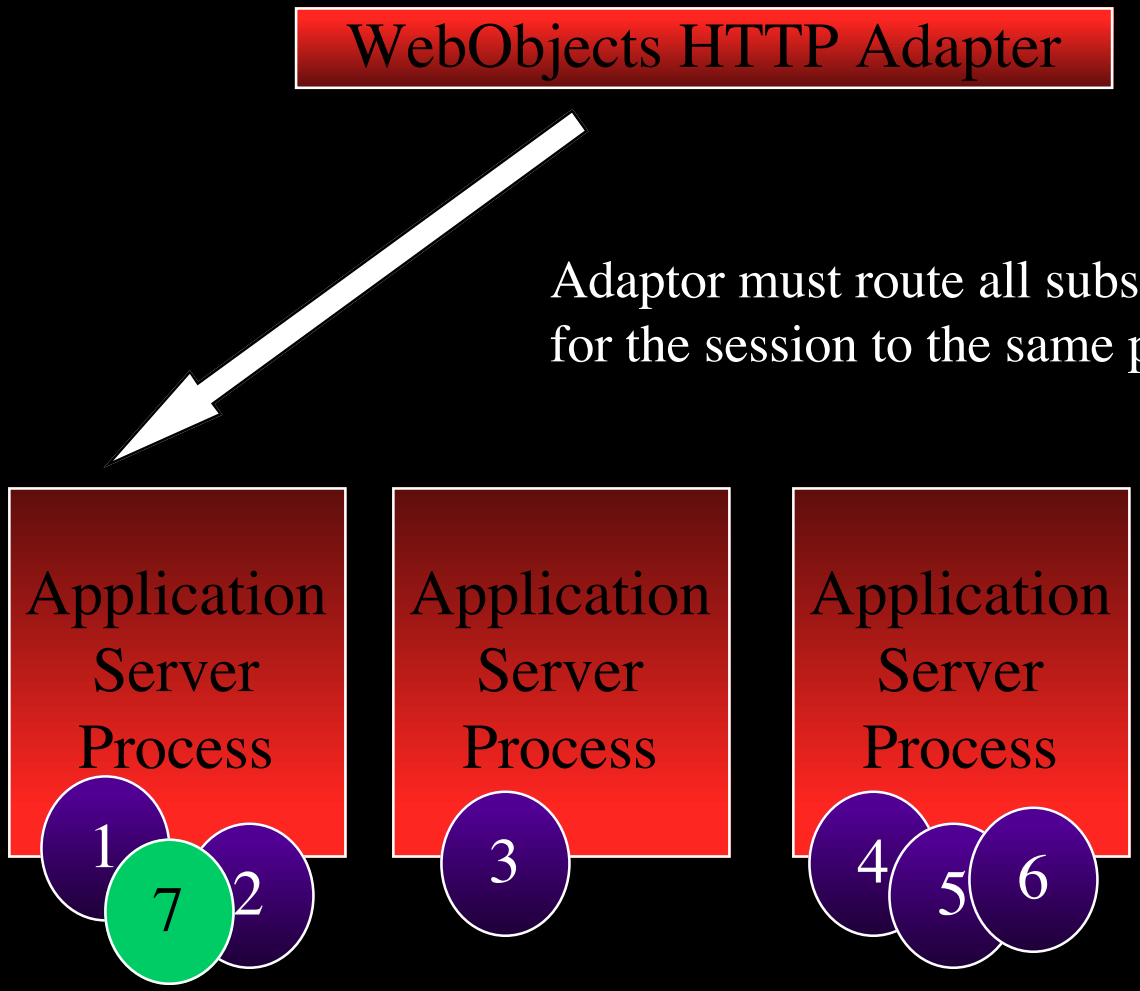
State in Server Process



Adaptor can pick any instance when sessions are created

State in Server Process

Load Balancing Constrained

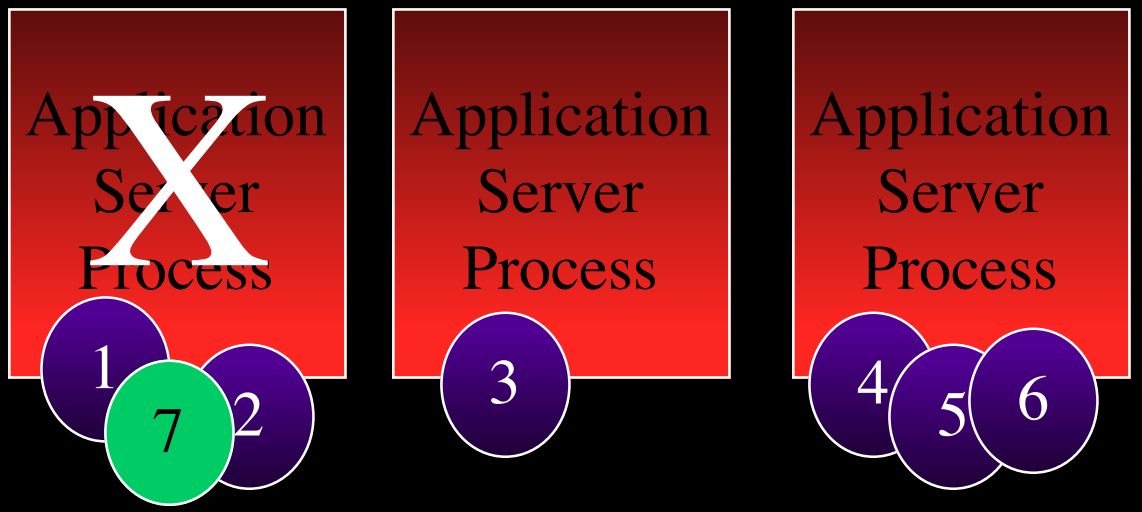


State in Server Process

Availability of Sessions Limited

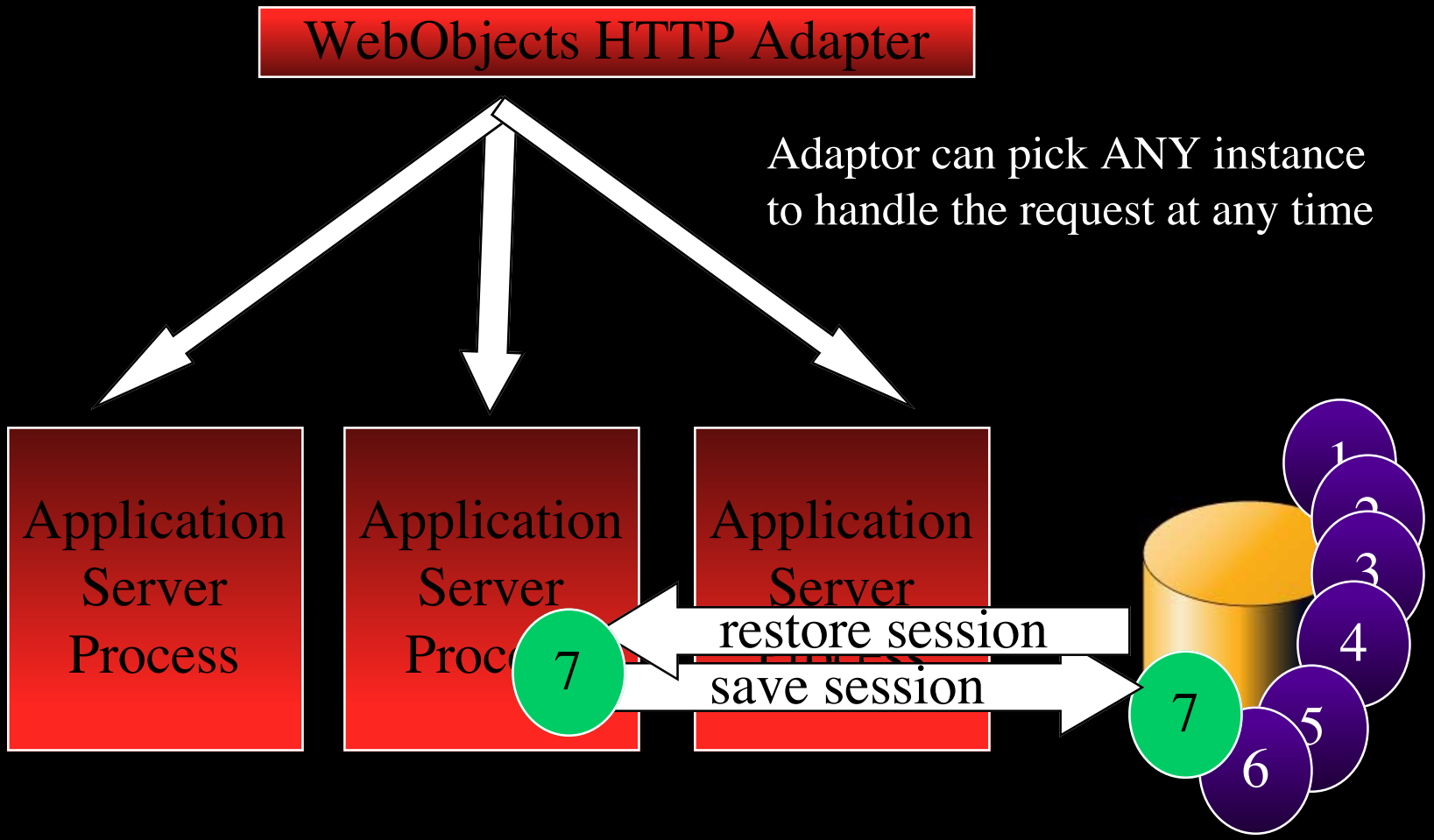
WebObjects HTTP Adapter

If process exits, all sessions in it are lost … site stays up but clients must start new sessions



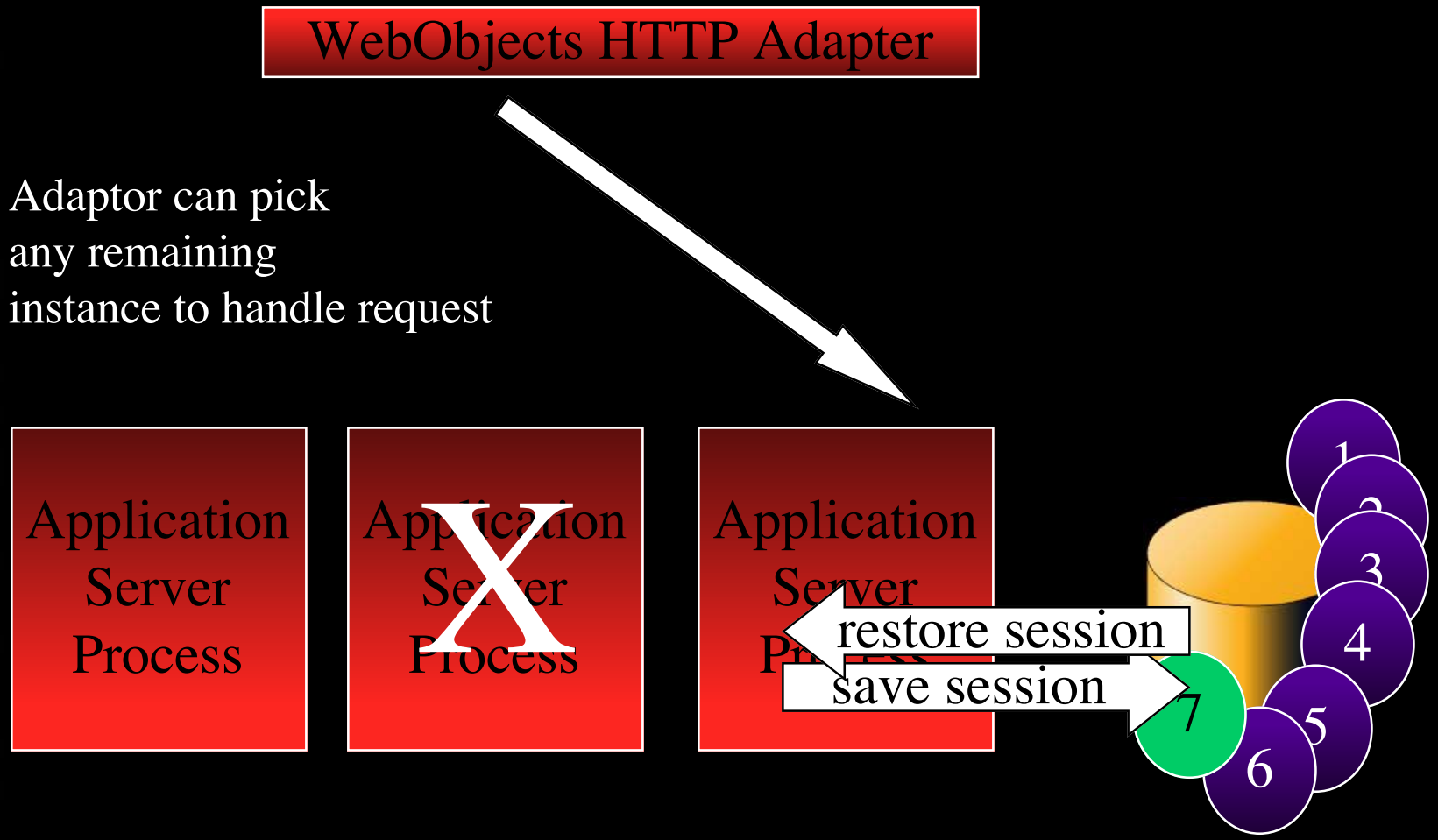
State in External Store

Load Balancing Unconstrained



State in External Store

Availability of Sessions Unlimited



State Management Trade-Offs

Server Side State
(RAM)

+ Works regardless of client
+ Inherently more secure
+ May be faster if RAM resource not an issue/addressed

- Can create very large footprints on the server if timeout not used
- Requires load balancing at the granularity of the session

State Management Trade-Offs

Client Side State

+ Page/Cookie state keeps Server footprints to a minimum
+ Allows the user to load balance at the granularity of the request

- Some overhead required to archive/unarchive the state
- Requires a FORM or Cookie support in the browser

State Management Trade-Offs

Server Side State
(External Store)

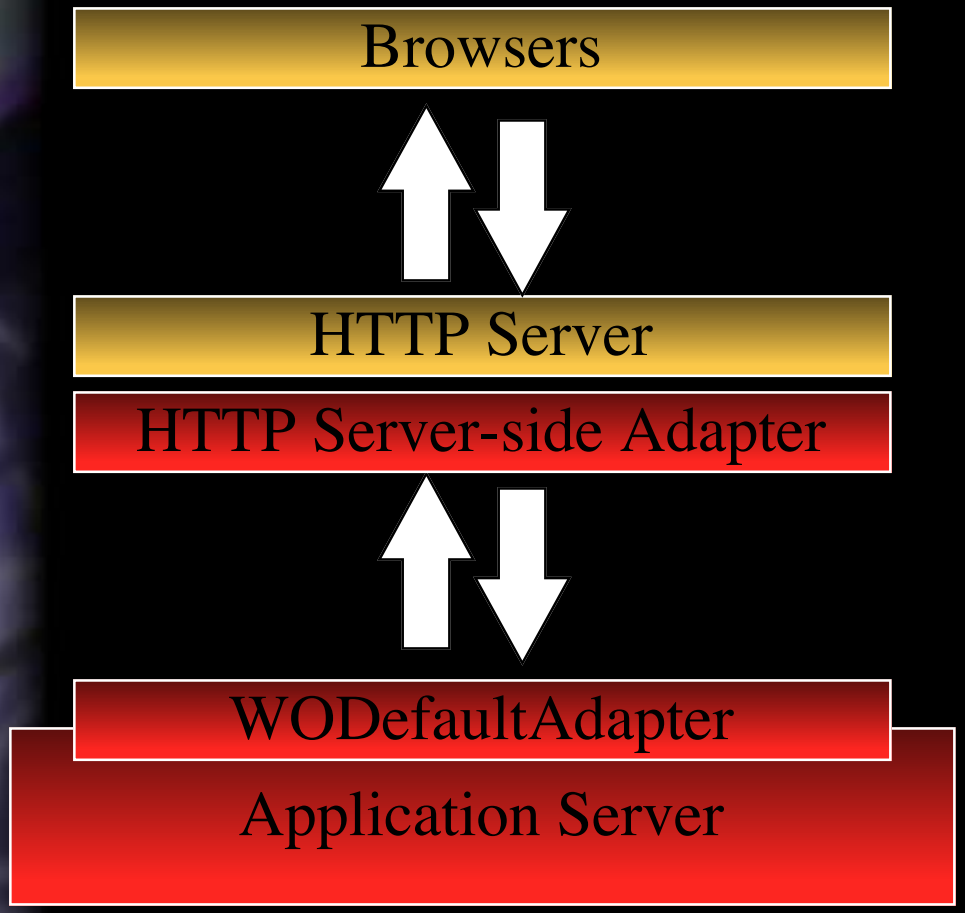
+ Works regardless of client
+ Nearly as secure as RAM state
+ Instances do NOT grow
+ Depending on store used, can be very fast
+ Built-in session recovery
+ Get to load balance at the granularity of the request

- Overhead required to archive/unarchive state from store

Physical Scalability

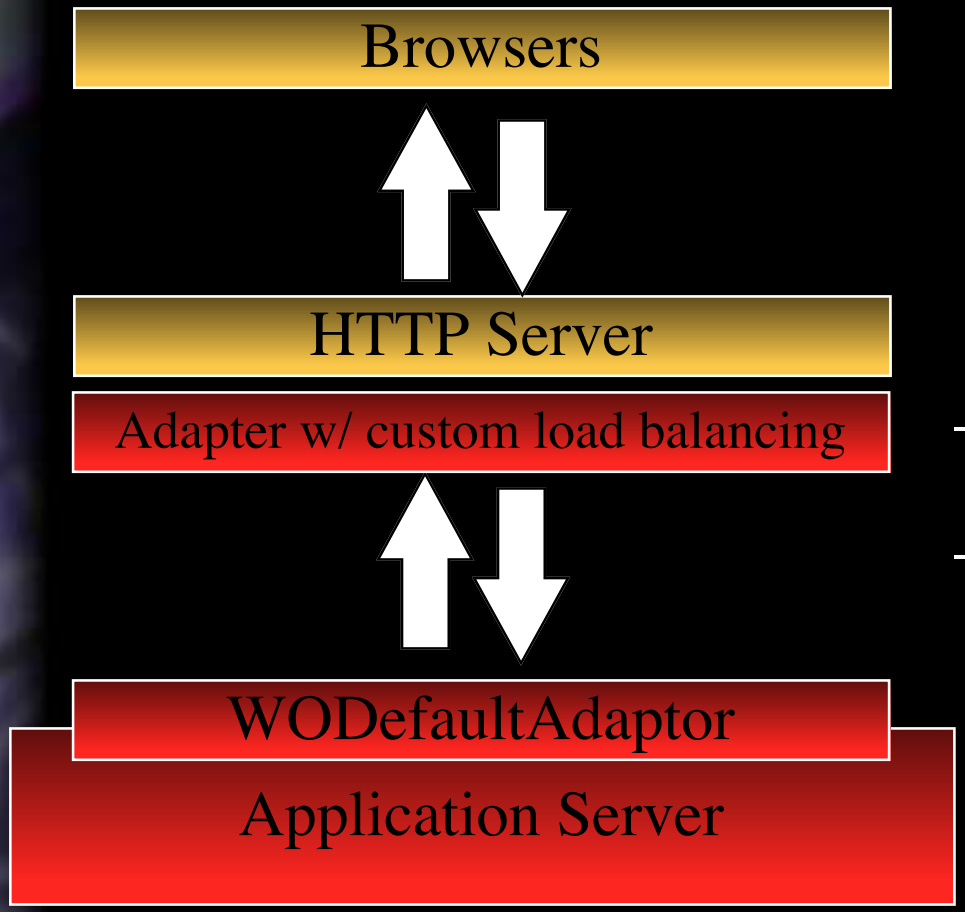
- Transaction throughput
- Reliability
- High performance state management architecture
- Extensible load balancing architecture

Extensible Load Balancing in WebObjects



- Default Adapter uses sockets and rapid rand selection strategy
- HTTP-and App-side source provided
- Comes in CGI 1.1, ISAPI, NSAPI 1.0 and 2.0 “flavors”

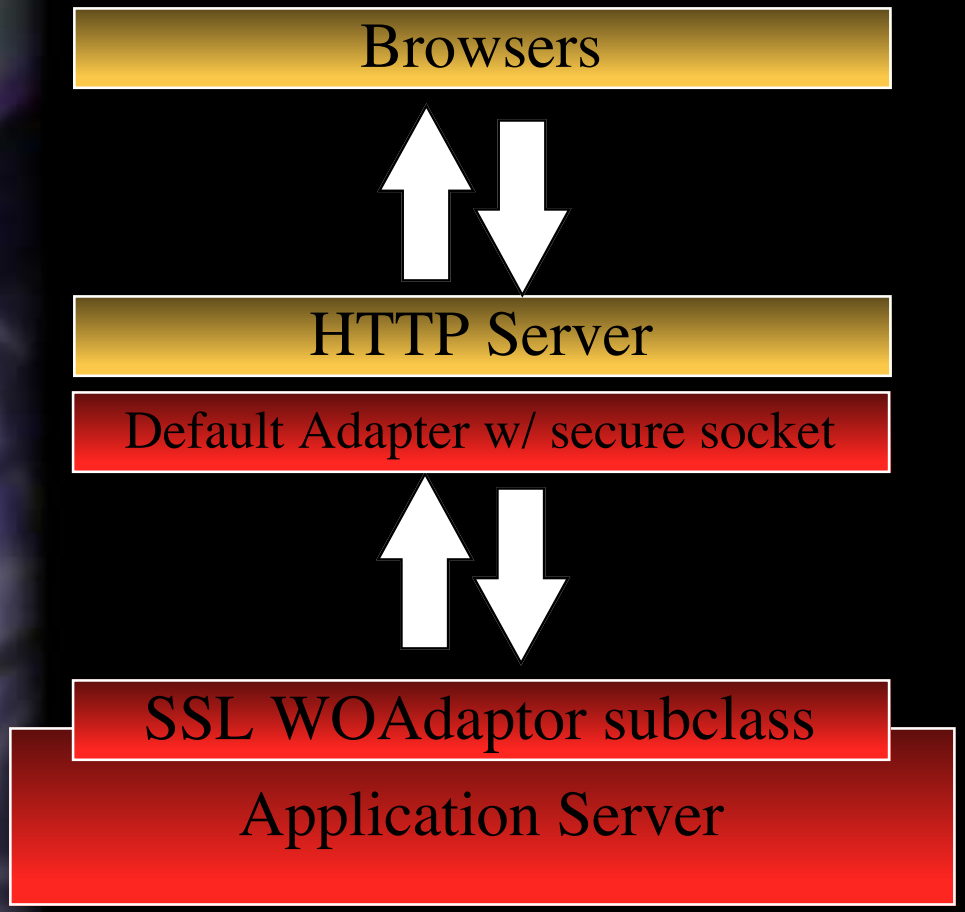
Extensible Load Balancing in WebObjects



Other Possibilities:

- Create a custom load balancing scheme based on provided source
- Can continue to use default WOAdaptor subclass

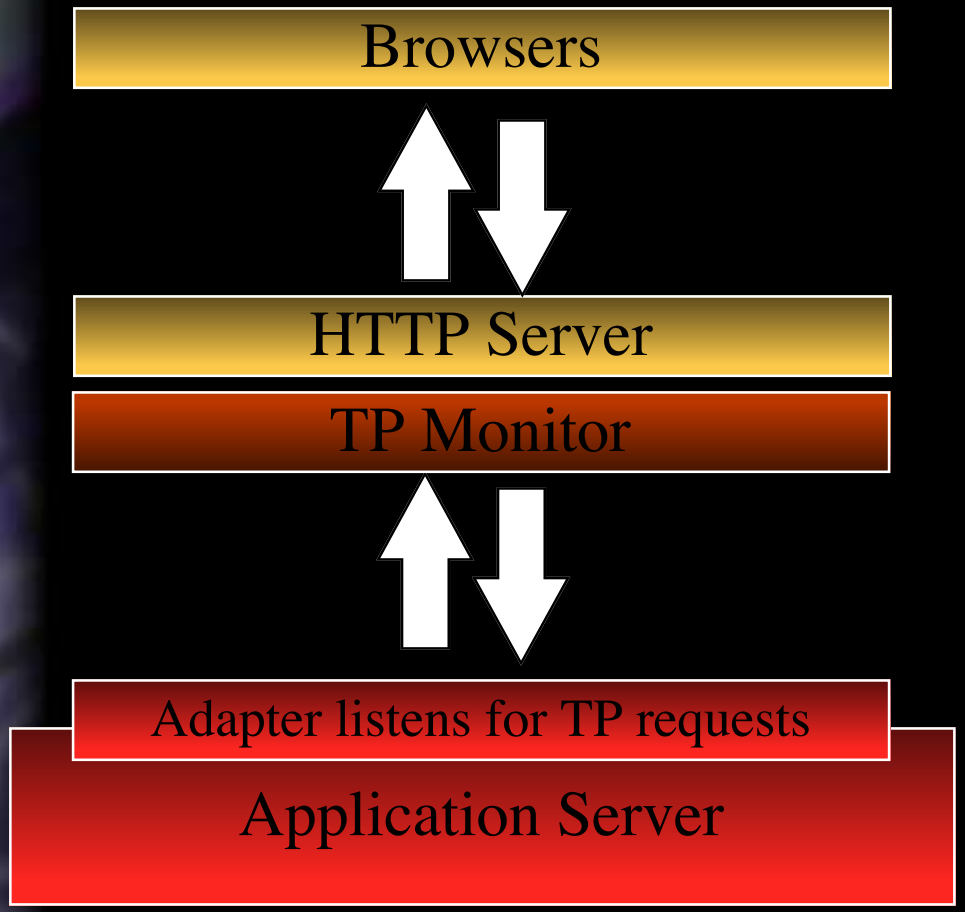
Extensible Load Balancing in WebObjects



Other Possibilities:

- SSL Adapter

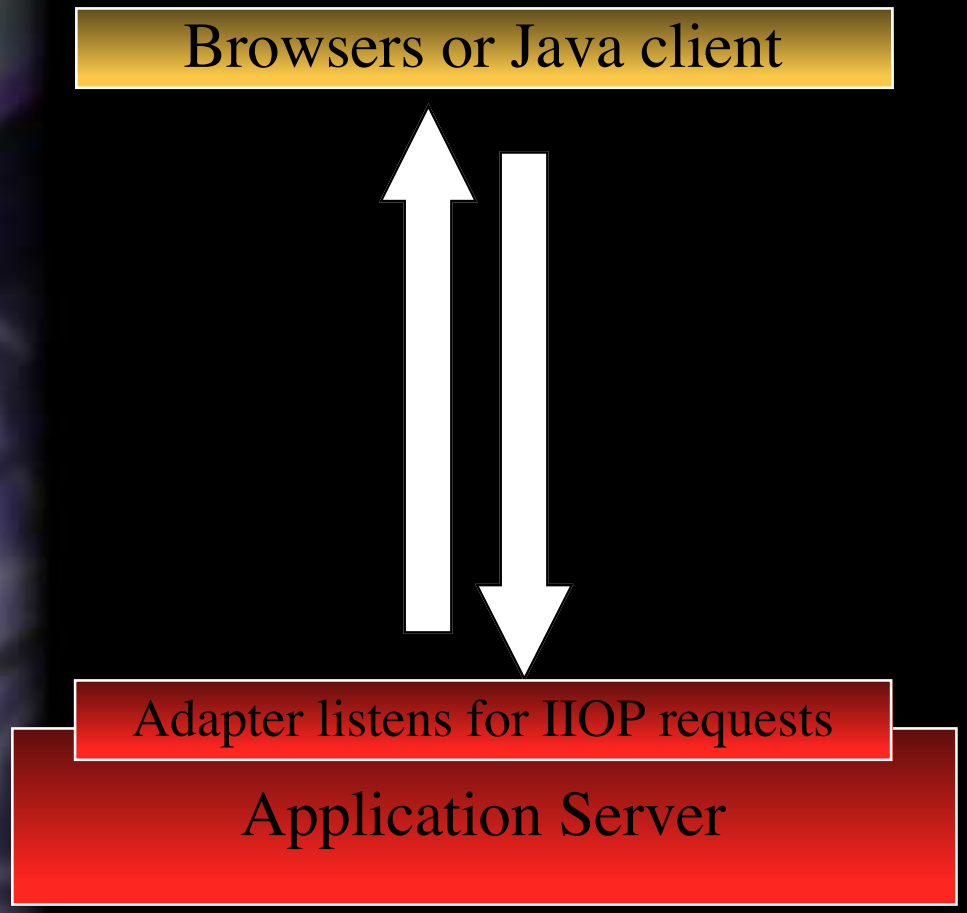
Extensible Load Balancing in WebObjects



Other Possibilities:

- Use existing TP Monitor to do load balancing strategy
- Create WOAdaptor subclass to listen to TP Monitor

Extensible Load Balancing in WebObjects



Other Possibilities:

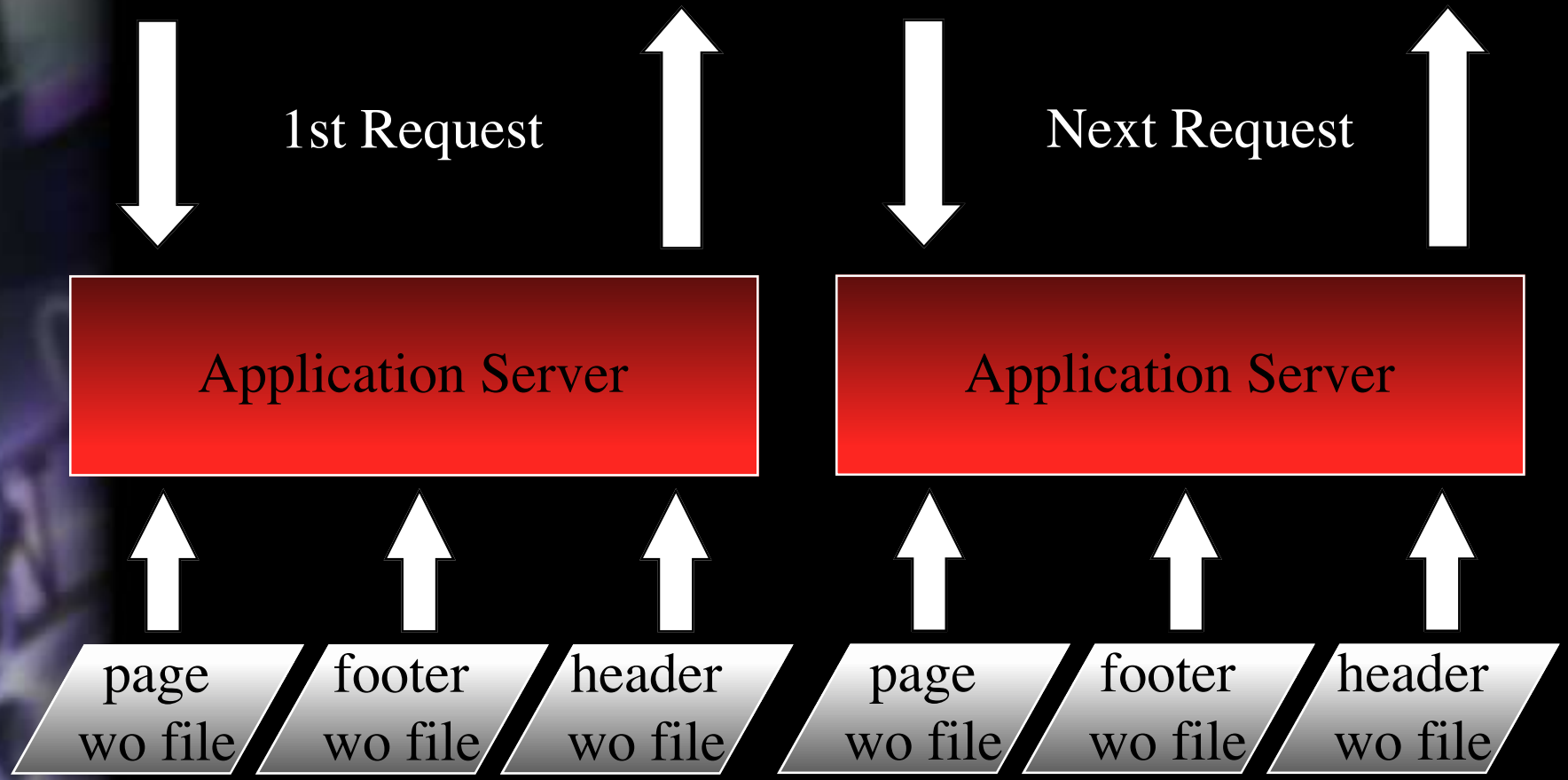
- Use IIOP and client/server CORBA available in Java VMs
- Create WOAdaptor subclass to listen to IIOP messages

Physical Scalability

- Transaction throughput
- Reliability
- High performance state management architecture
- Extensible load balancing architecture
- Application tuning

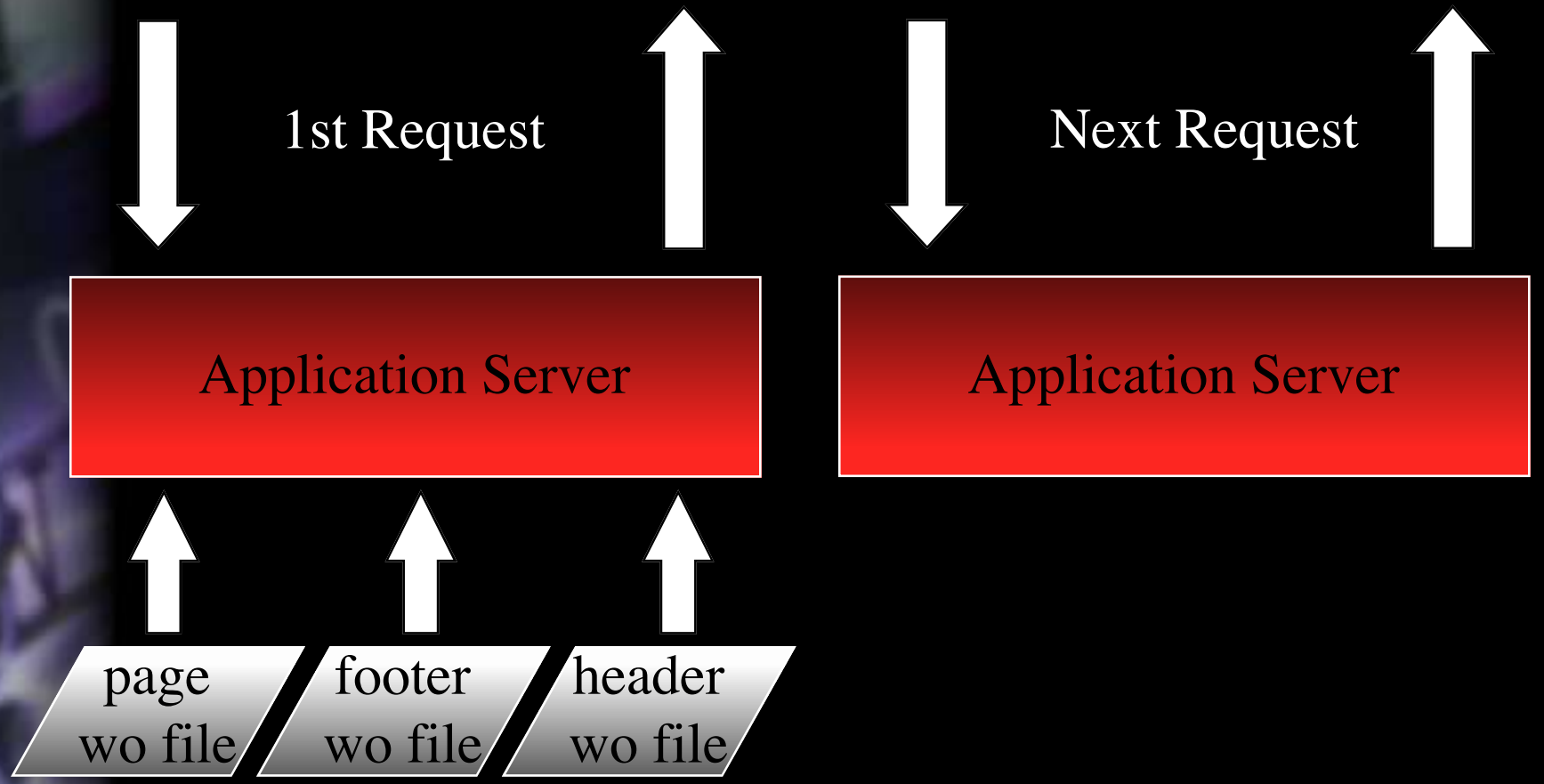
Page Definition Caching

Development Mode: Caching Turned OFF



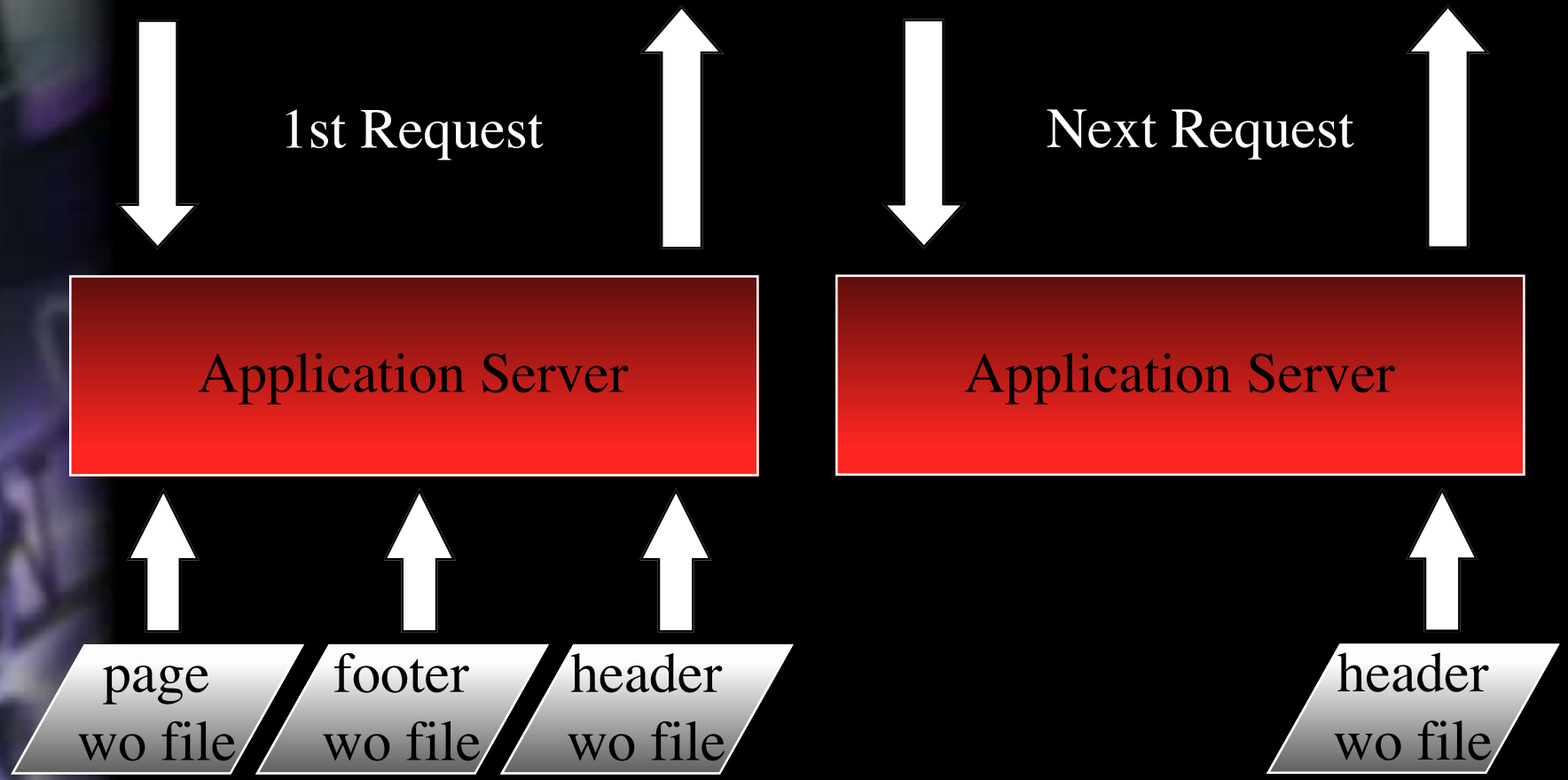
Page Definition Caching

Deployment Mode: Caching Turned ON



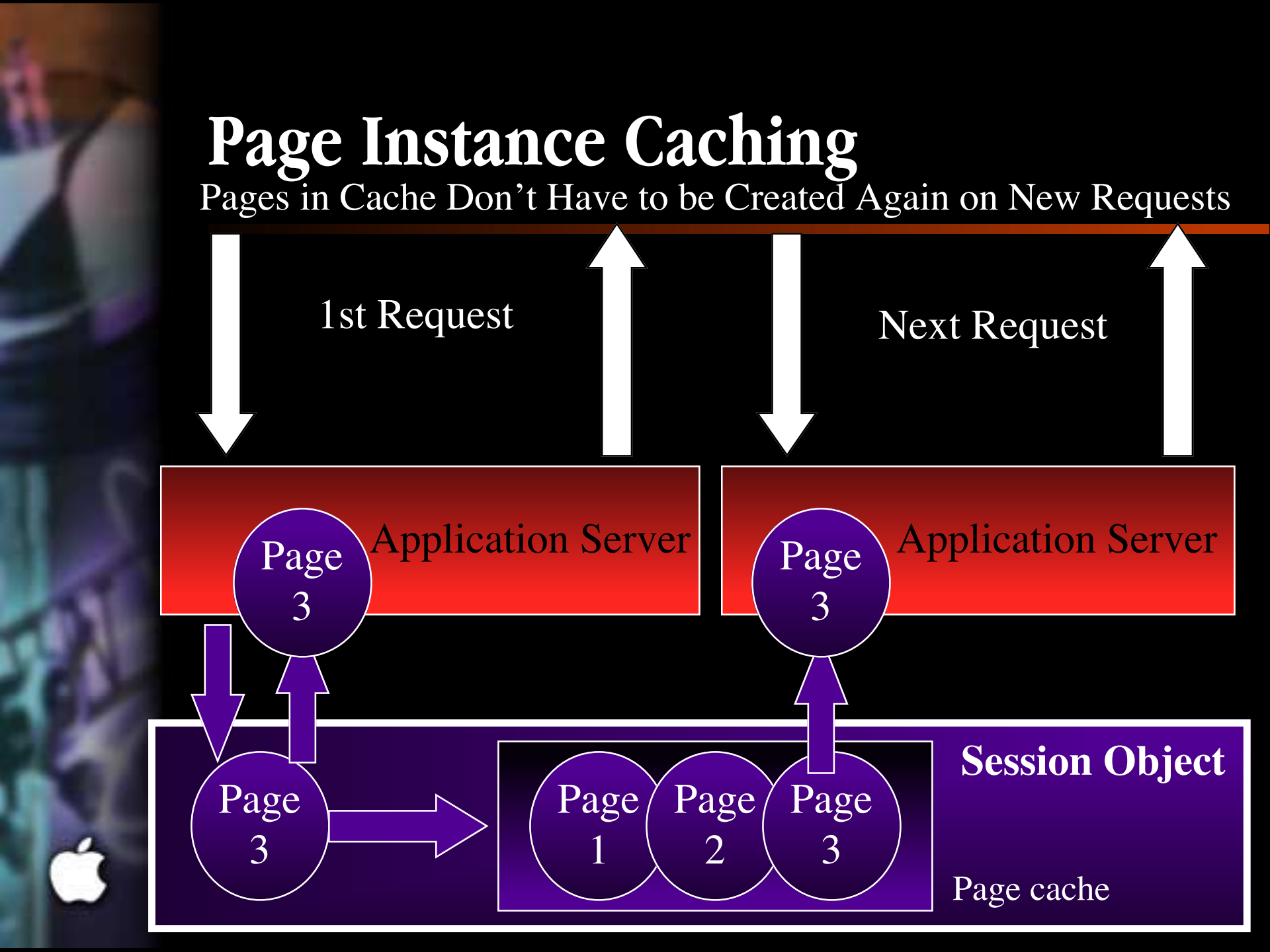
Page Definition Caching

Deployment Mode: Caching Turned ON Selectively



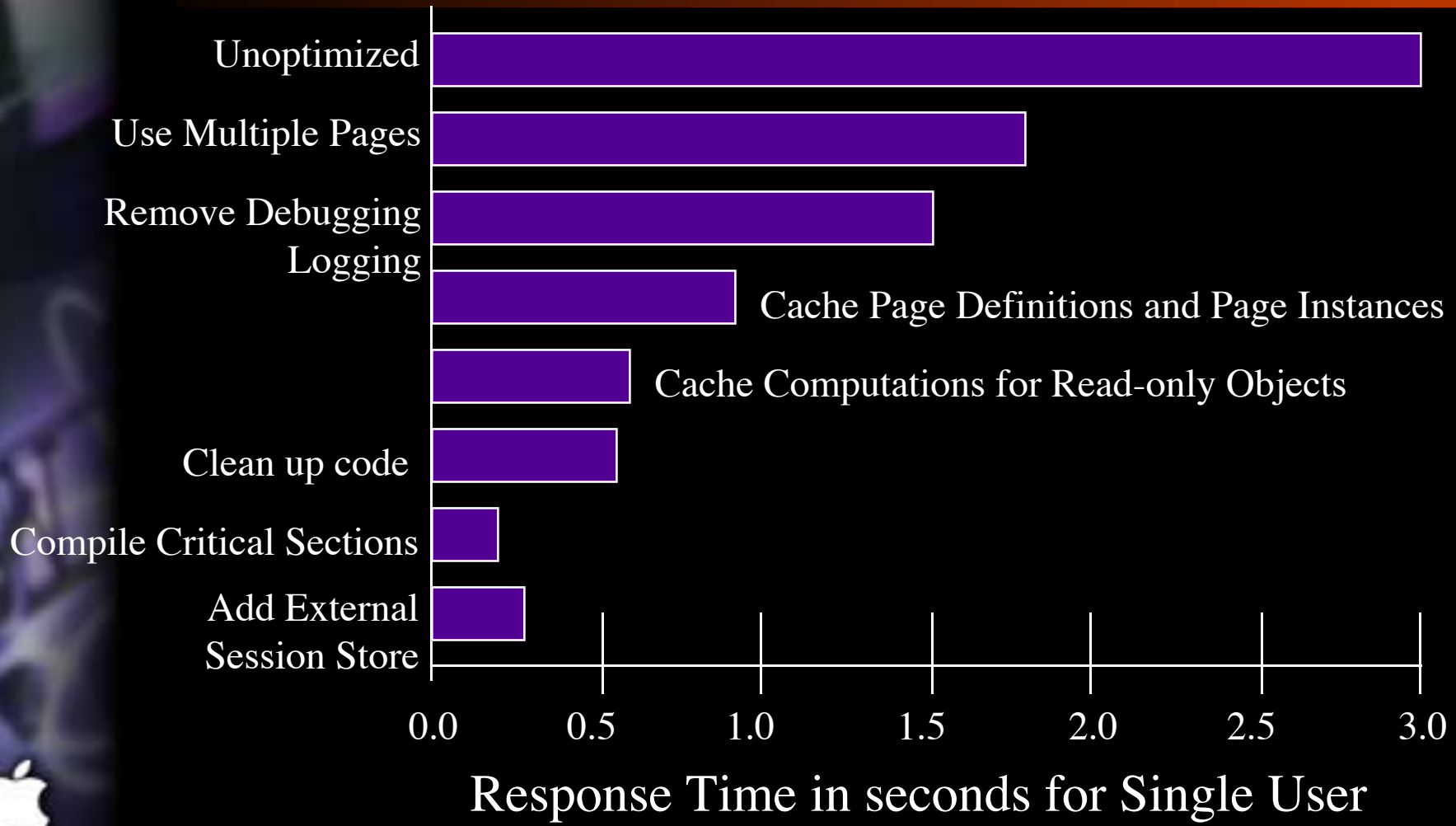
Page Instance Caching

Pages in Cache Don't Have to be Created Again on New Requests



Optimizing Page Gen Time

A Case Study in Optimizing the "Worst" Page



Optimizing Database Utilization

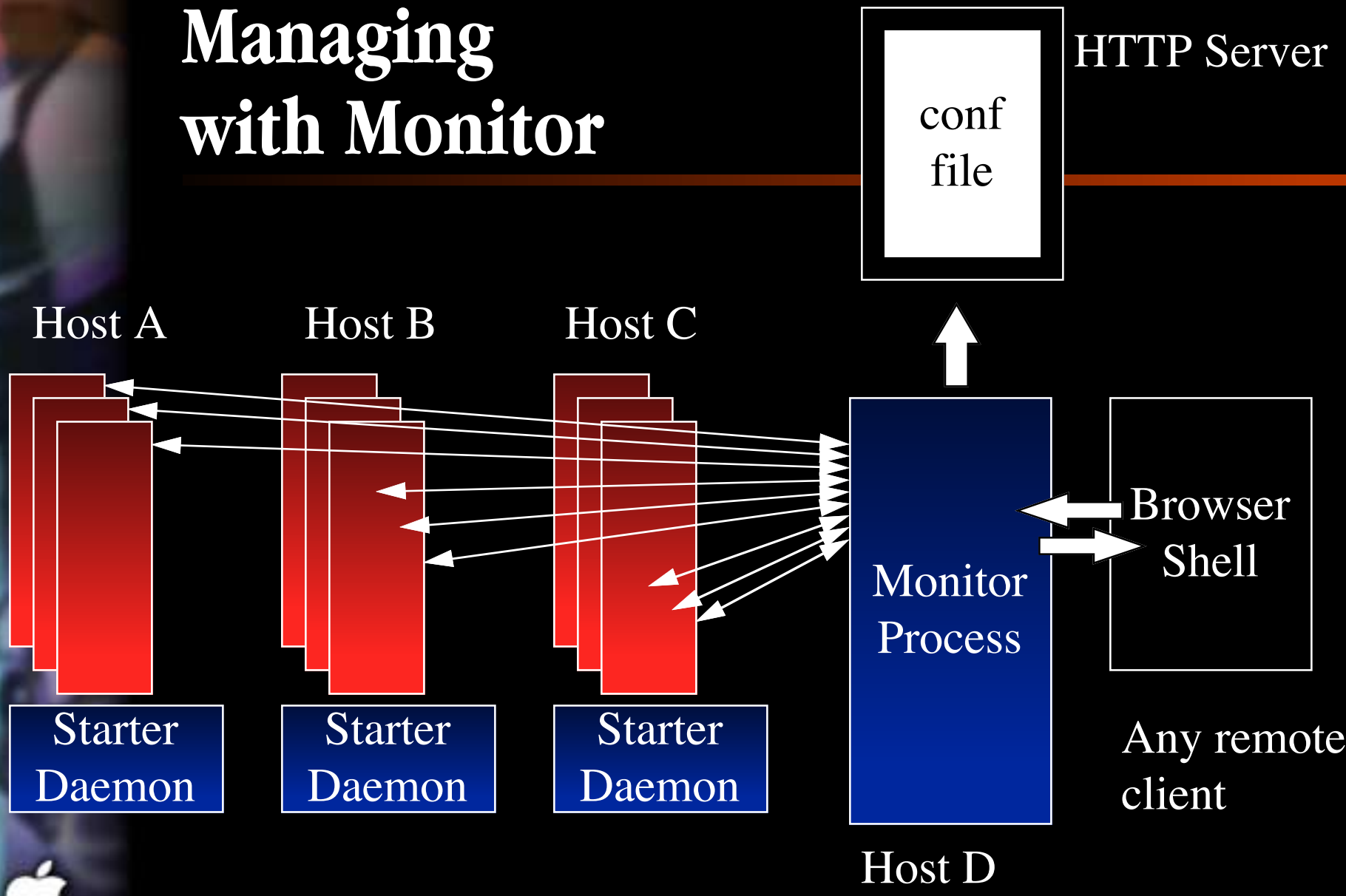
What EOF Technology Gives Us for Free

- Sessions multiplex across one database connection per instance
  Conserve database connections
- Optimistic locking with snapshots default update strategy
  Avoid deadlocks; minimize waiting for record release
- Changes made in memory propagated in batch to database
  Database hits minimized; unwanted operations never hit database
- Small Tables may be cached in memory
  Avoid unnecessary round trips to database
- In memory sorting and querying
  Avoid round trips for filtering and presentation purposes
- Database sorting and querying
  For extremely large datasets when server processing more efficient
- Objects uniqued per row in memory
  Minimize need to make trips to the database; App servers smart enough to hand you a prefetched object instead of going to the database every time

Physical Scalability

- Transaction throughput
- Reliability
- High performance state management architecture
- Extensible load balancing architecture
- Application tuning
- Managing site changes

Managing with Monitor



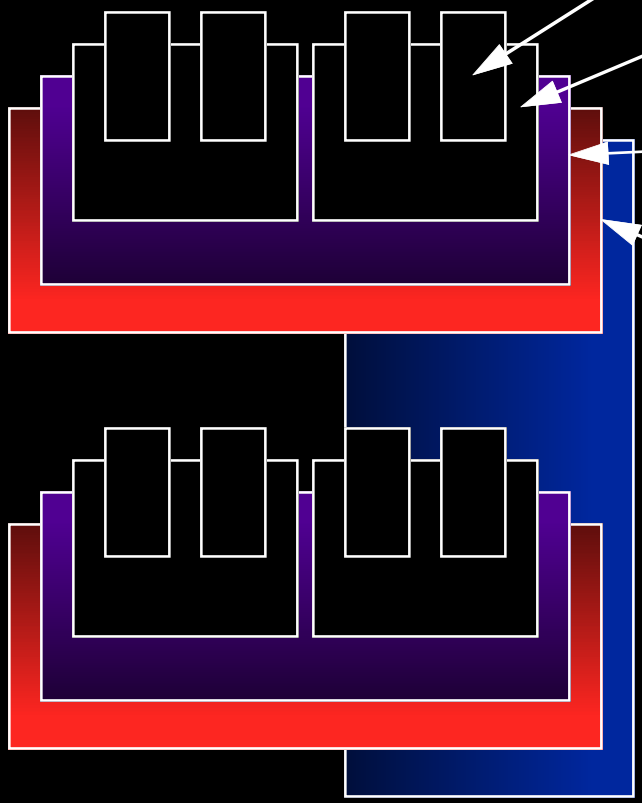
Managing with Monitor

What You Can Administrate Remotely

- Add, remove, configure instances
  Scale your site for more or less users
- Set always ON/always OFF settings on per instance basis
  Force certain instances to always be ON when the Monitor runs
- Automate instance cycling on per instance basis
  Avoid runaway foot-print due to undiscovered leaks
- View performance statistics for running instances
  Provide feedback to instance configuration decisions
- Provide and Control “super application” state
  i.e. Toggle page definition caching

Super Application State

What Is It?



Method handler scope

Page scope
(disappears after request unless cached in Session)

Session scope
state shared across all trans-ctions (Resides in session store)

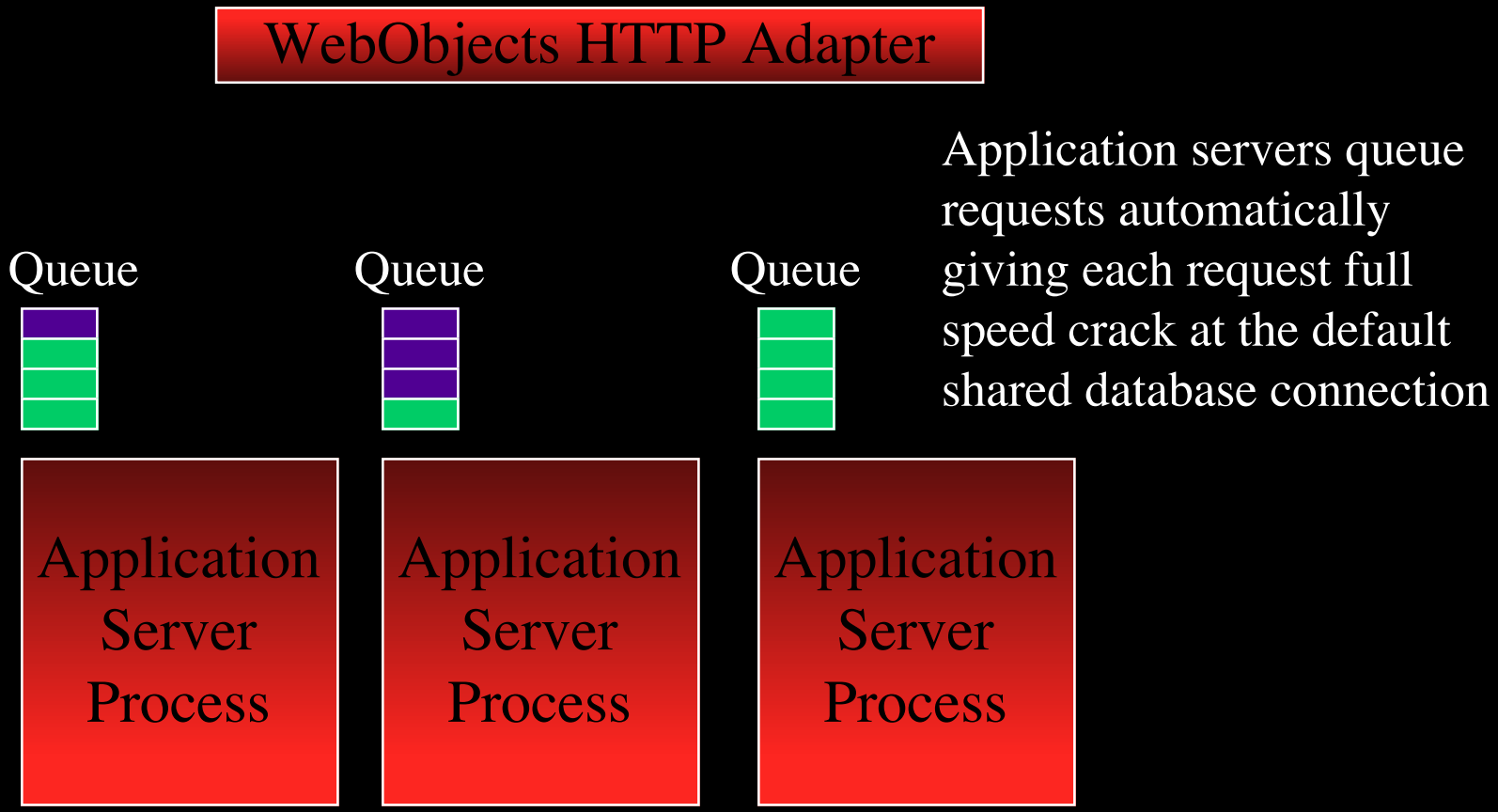
Application scope
state shared by all sessions (Resides in process RAM)

Super Application scope
state shared by all running processes (Resides in Monitor process)

Physical Scalability

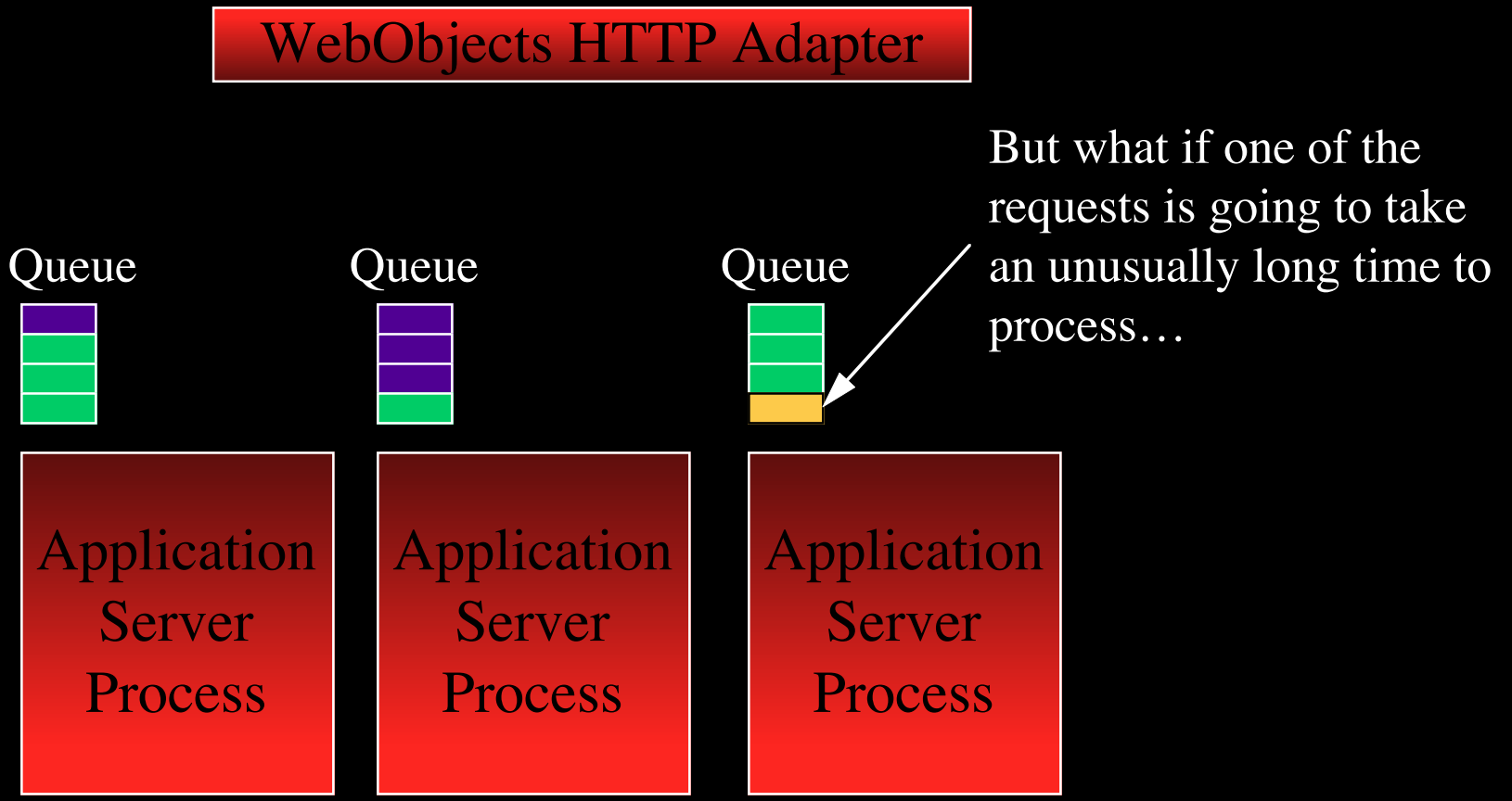
- Transaction throughput
- Reliability
- High performance state management architecture
- Extensible load balancing architecture
- Application tuning
- Managing site changes
- Handling pathological response times

Application Event Processing



Application servers queue requests automatically giving each request full speed crack at the default shared database connection

Application Event Processing



Pathological Response Time

What to do

- If response time due to large fetch, specify a fetch record limit

  Avoid massive fetches on under-qualified fetch specifications

- If response time due to unavoidable calculation or slow resource, use Multi-threading

  Allows user to submit a long running process without blocking the submitting user or any other users that queue into the shared process

- Use custom load balancing scheme to dedicate each session with its own process

  Prevent other users from queuing behind one user's long response time


POWERED BY
WEBOBJECTS
NeXT
TM


Q&A

Tuning Tools in WebObjects