The Design, Implementation and Performance Evaluation of Internet Services

Tim Brecht

University of Waterloo

http://cs.uwaterloo.ca/~brecht

Announcement

- Summer Research Internships at U Waterloo
  – Several Different areas
  – Competing and possibly working with top students from around the world

See:

http://blizzard.cs.uwaterloo.ca/intern07/info.html

Introduction

- **Tim Brecht** (pronounced brek-t)
  - brecht@cs.uwaterloo.ca
  - http://cs.uwaterloo.ca/~brecht

    (see my 497 link for assignment due next lecture)
- Background
  - B.Sc. (Sask.), M.Math (Waterloo), Ph.D. (Toronto)
  - On faculty (York & Waterloo)
  - Visiting Scientist (IBM)
  - Sabbatical & Research Scientist (HP Labs) 1+2 yrs
- Research Interests: performance, operating systems, networking, parallel and distributed computing

Introduction

- My research described here done with many people:
- **Ugrads**: Craig Barkhouse (UW),
  Siddharth Gupta (IIT Guwahati)
- **UW grad students**: Michal Ostrowski, David Pariag,
  Amol Shukla, Jialin Song, Elad Lehav,
  Weihan Wong, Ashif Harji, Gary Yeung
- **UW Faculty**: Martin Karsten, Peter Buhr,
- **UW Staff**: Louay Gammo, Mark Groves
- **HP Labs**: Brian Lynn, John Janakiraman,
  Yoshio Turner
- **Intel Labs**: Greg Regnier, Vikram Saletore

Outline

- **Part I: Background**
  - **Web Server Example: HTTP/1.1**
  - **Server Architectures**
  - **Performance Evaluation**
- Part II: A Flavour of some Current Research
  - Performance of Different Server Architectures
  - Improving Operating System Support for I/O Centric Servers (if time permits)
  - Possible Avenues for Future Research

Outline

- **Part I: Background**
  - Web Server Example: HTTP/1.1
  - Server Architectures
  - Performance Evaluation

How to build a fast Internet Service

- Types of services
  - Web Servers
  - Streaming Audio/Video Services
  - Game Services
  - Domain Name System (DNS) (i.e., name lookups)
  - Mail: SMTP / IMAP / POP
  - Chat Servers (Text)
  - Voice over IP
  - File Sharing (i.e., music stealing)

Example Internet Service: Web Server

- Simple to understand
- Easy to implement
- Widely used:
  - **106,875,138** Web Sites [Netcraft, January 2007]
  - Dominant Internet Service/Application
  - UW traffic [ist.uwaterloo.ca/cn/Stats/extvol.html]
    - http 62%, other 18%, ssh 6% Jan 10, 2007
    - http 52%, other 38%, p2p 3% March, 2005
    - http 63%, other 20%, ftp 7% March, 1998

Outline

- Part I: Background
  - **Web Server Example: HTTP/1.1**
  - Server Architectures
  - Performance Evaluation

Simple Web Server Request/Response

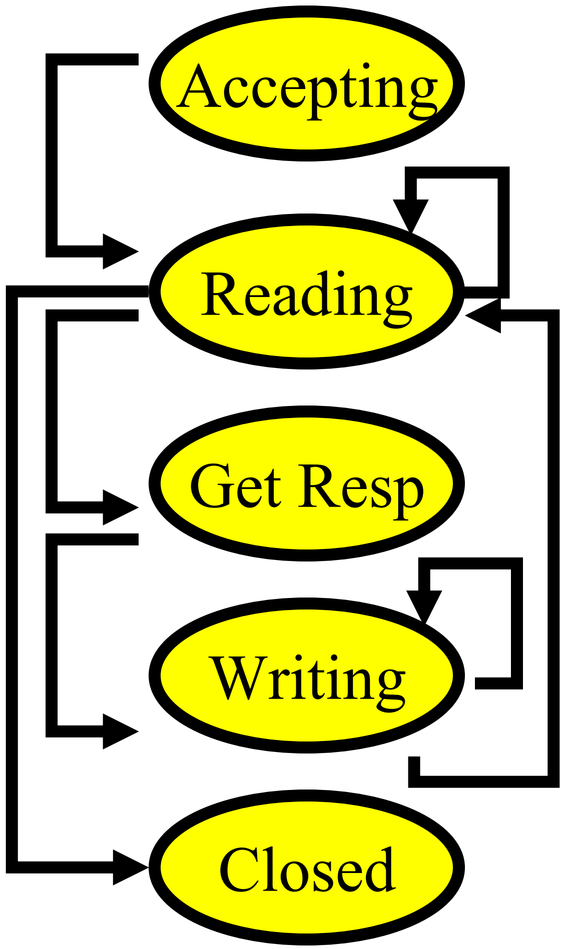
Client sends to server:

```
GET docs/10B.txt HTTP/1.1
User-Agent: httperf/0.8.4
Host: 127.0.0.1
<cr><lf>
```

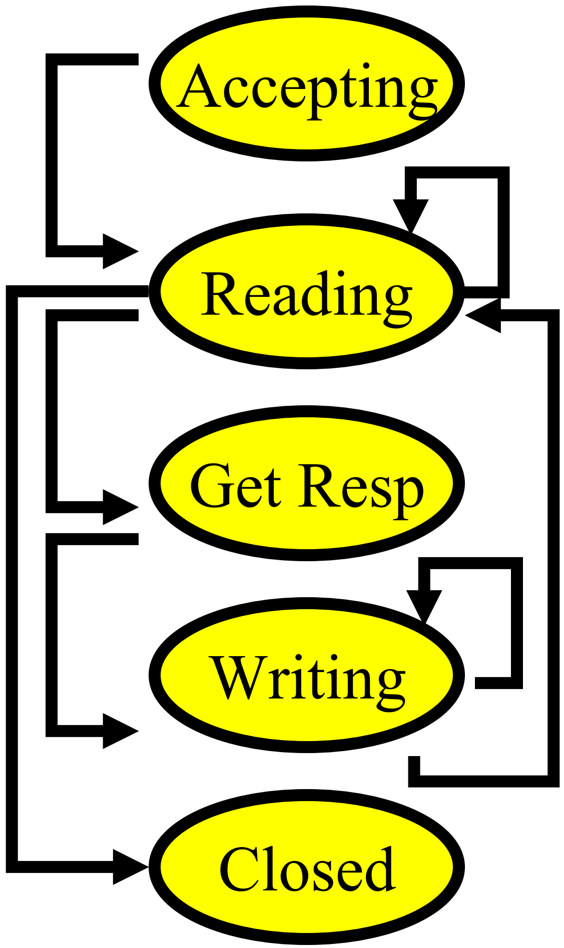
Server replies to client:

```
HTTP/1.1 200 OK
Server: userver-0.5.2
Content-Length: 10

012345678
```

HTTP/1.1: State Machine



HTTP/1.1: State Machine



Fairly easy to translate this into a simple server

Outline

- Part I: Background
  - Web Server Example: HTTP/1.1
  - **Server Architectures**
  - Performance Evaluation

A Simple Server

```
server_sd = socket(); bind(server_sd);
listen(server_sd);

for (;;) {
   // wait for new connection request
   sd = accept(server_sd);
   handle_requests(sd);
}

handle_requests(int sd)
{
   while(read_request(sd, inbuf)) {
       parse_request(inbuf);

       // get or compute response

       write_response(sd, outbuf);
   }
   close(sd);
}
```

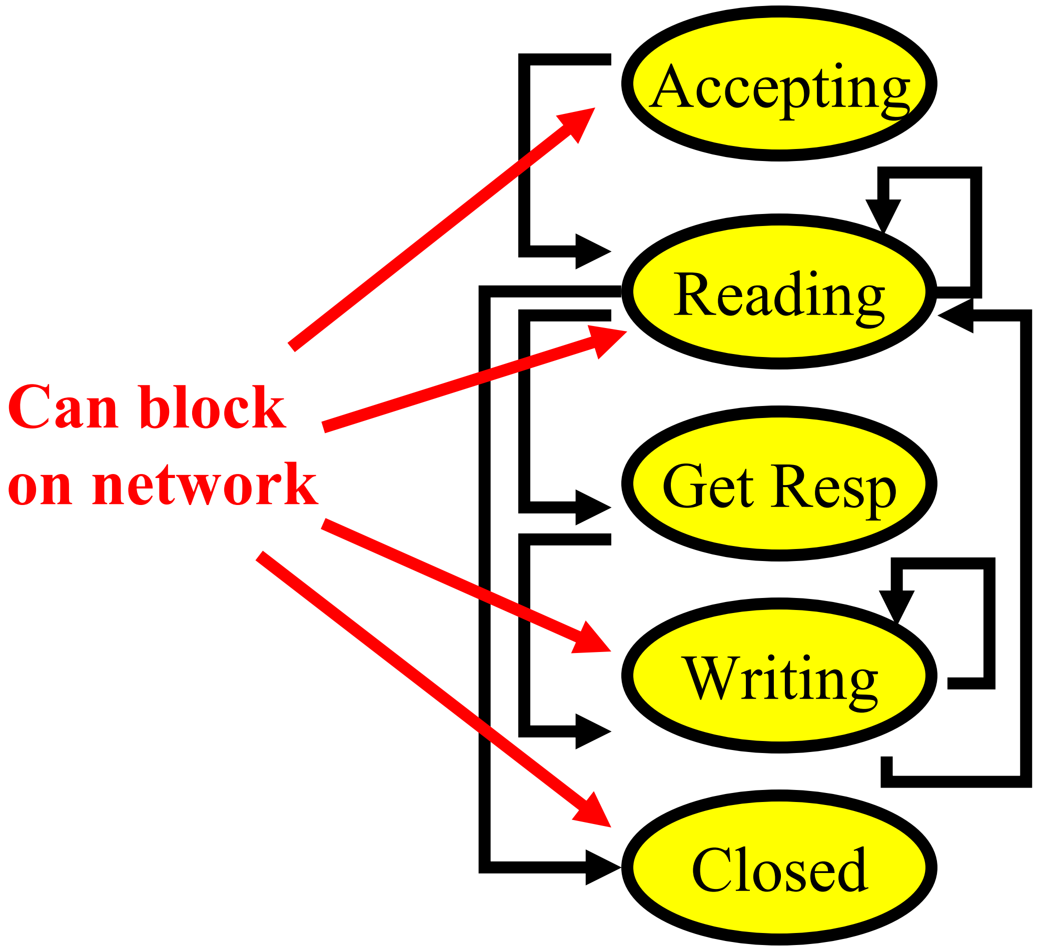
A Simple Server

```
server_sd = socket(); bind(server_sd);
listen(server_sd);

for (;;) {
   // wait for new connection request
   sd = accept(server_sd);
   handle_requests(sd);
}

handle_requests(int sd)
{
   while(read_request(sd, inbuf)) {
       parse_request(inbuf);

       // get or compute response

       write_response(sd, outbuf);
   }
   close(sd);
}
```

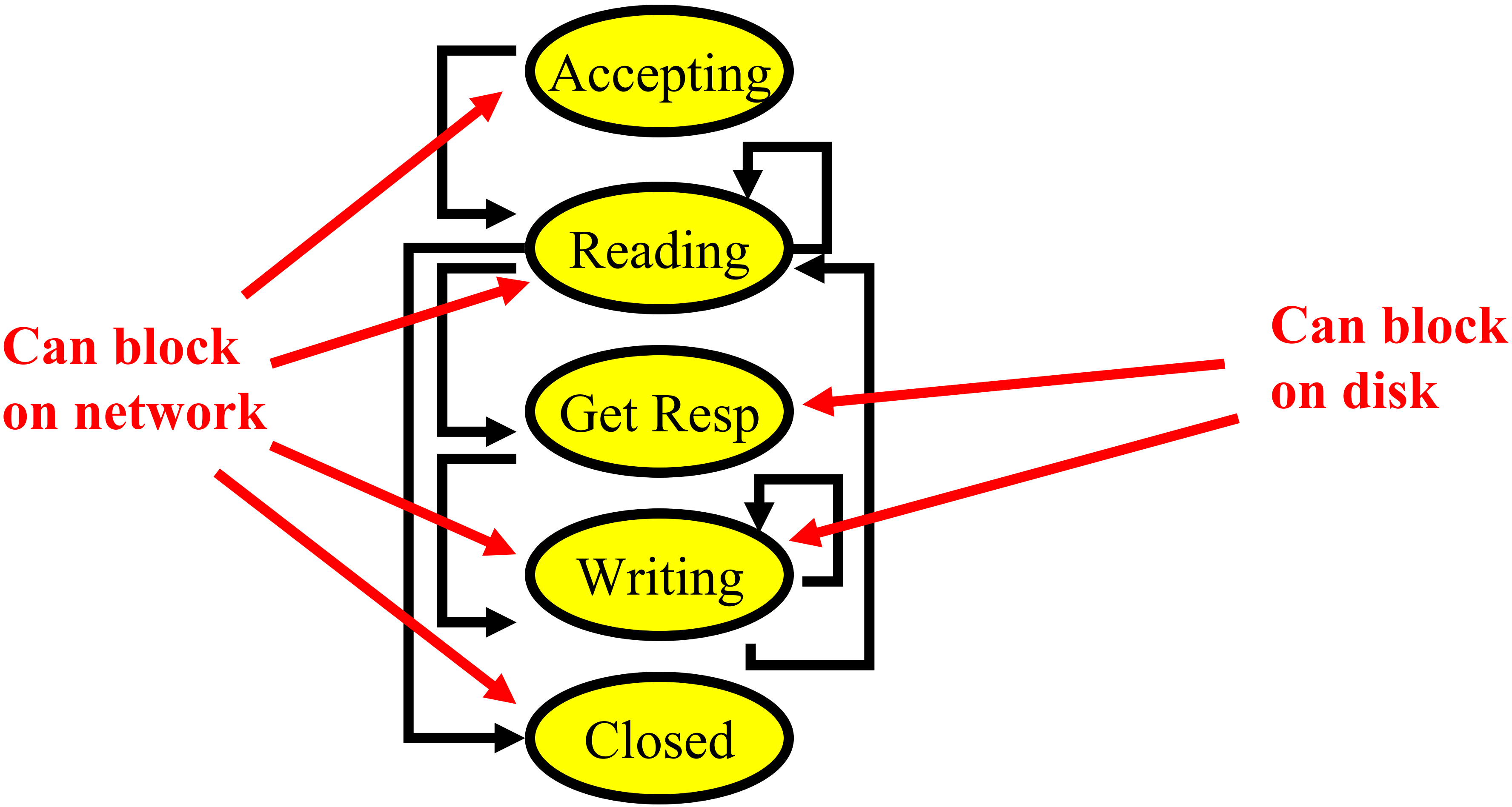
What's good about this approach?

What's bad about this approach?

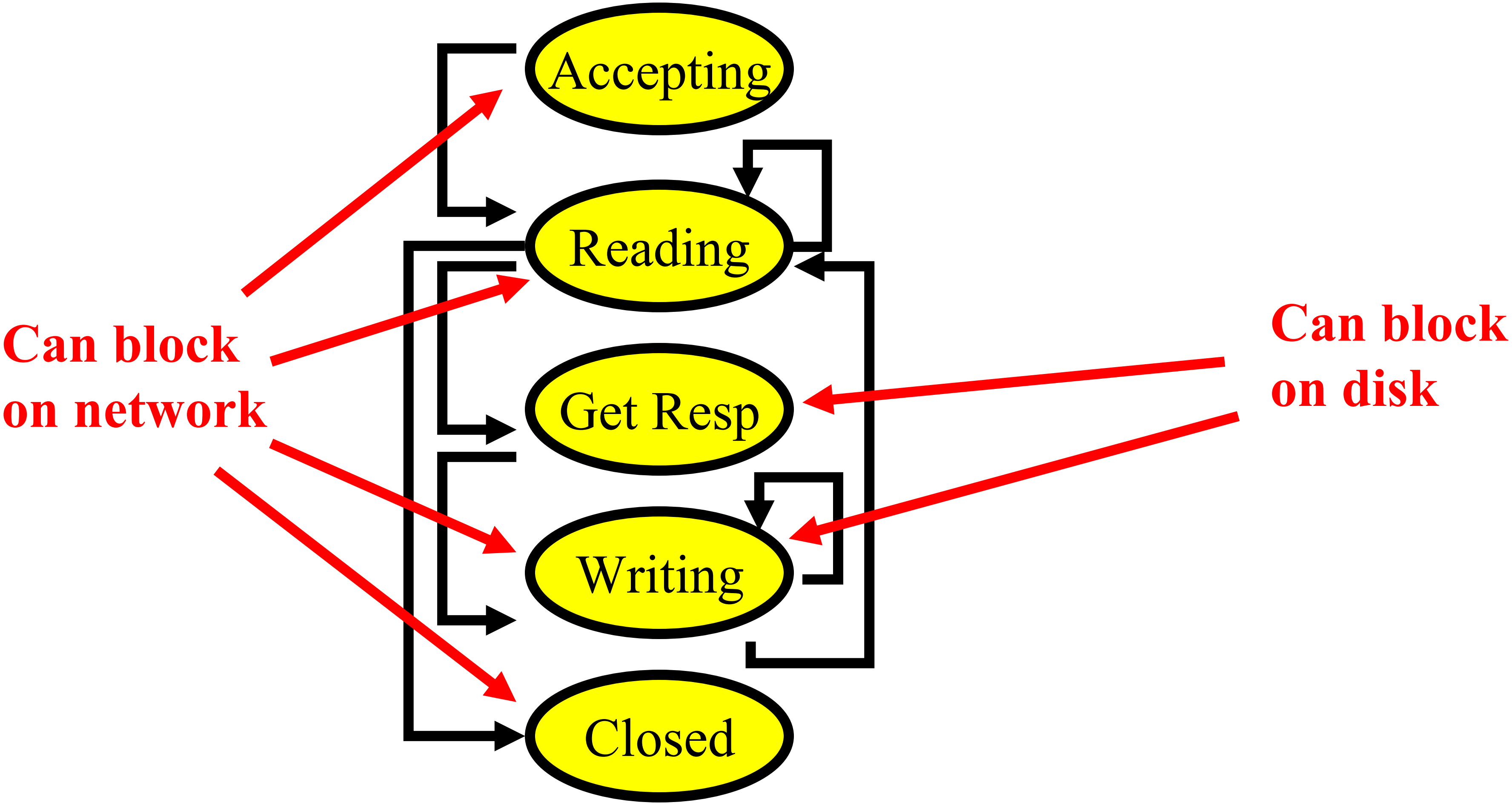
HTTP/1.1: State Machine



HTTP/1.1: State Machine

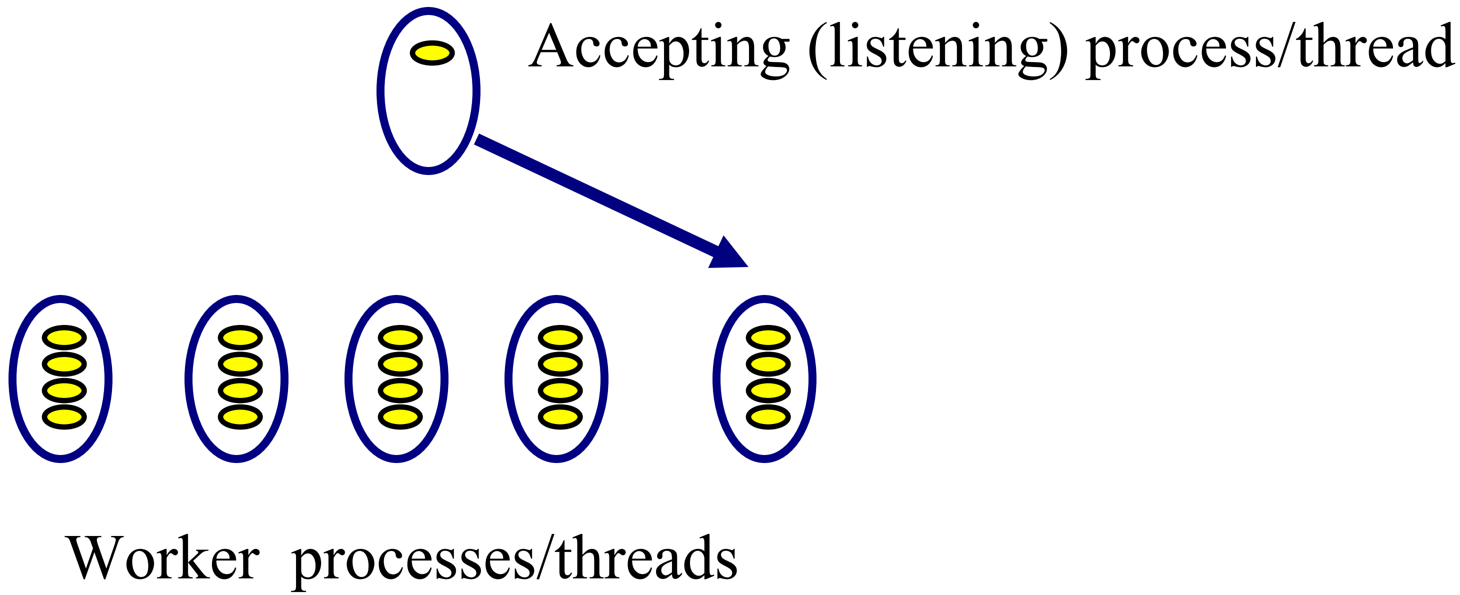


HTTP/1.1: State Machine



Possible Solutions?

A Forking Server



A Forking Server

```
for (;;) {
   // wait for connection
    sd = accept(server_sd);

   // fork/create child to handle request
   fork/create(handle_requests, sd);
}
```

A Forking Server

```
for (;;) {
   // wait for connection
    sd = accept(server_sd);

   // fork/create child to handle request
   fork/create(handle_requests, sd);
}
```

What's wrong with this approach?

A Forking Server

```
for (;;) {
   // wait for connection
    sd = accept(server_sd);

   // fork/create child to handle request
   fork/create(handle_requests, sd);
}
```

What's wrong with this approach?

How many simultaneous connections can be supported?

A Forking Server

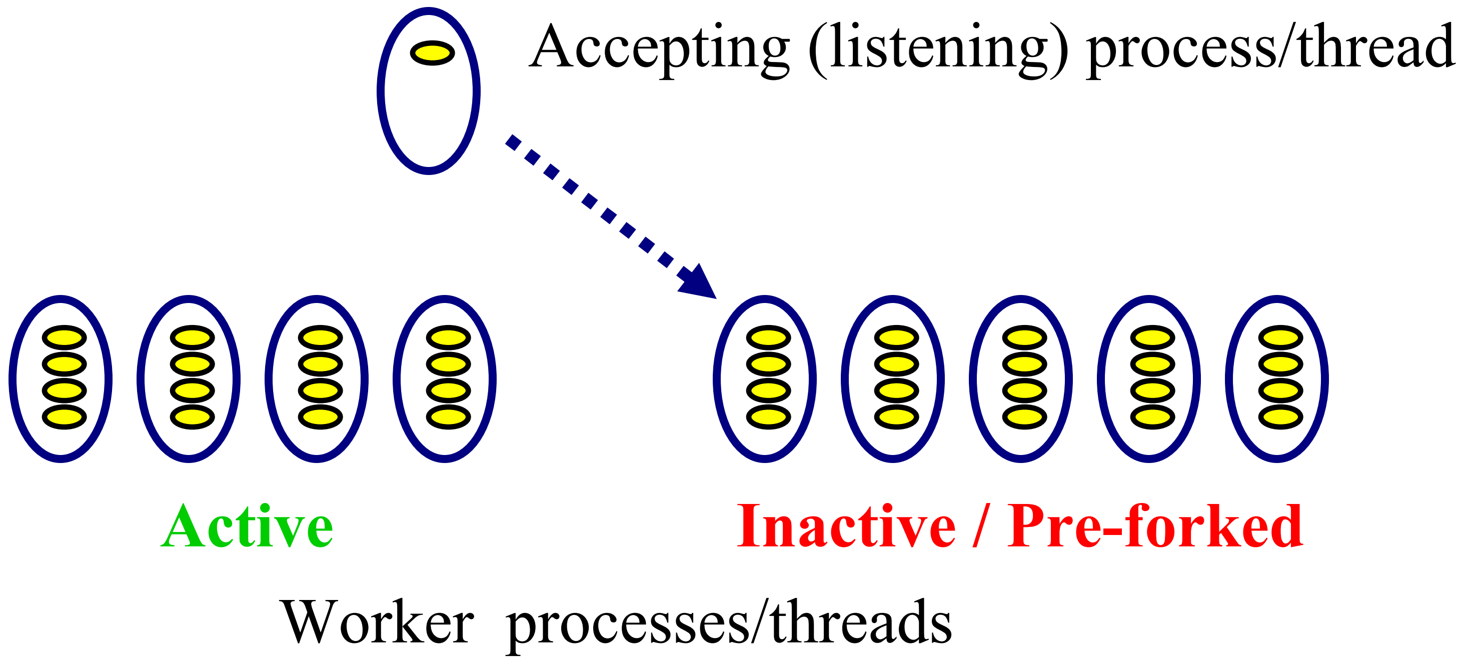
```
for (;;) {
   // wait for connection
    sd = accept(server_sd);

   // fork/create child to handle request
   fork/create(handle_requests, sd);
}
```

What's wrong with this approach?

How many simultaneous connections can be supported?

Should this server limit resource consumption? Which ones?

A Pre-Forking Server



A Pre-Forking Server

```
for (i=0; i<P; i++) {
   // fork/create a worker process
}

for (;;) {
   // wait for connection
    sd = accept(server_sd);

   // find idle worker to handle request
   pass_to_worker(sd);
}
```

A Pre-Forking Server

```
for (i=0; i<P; i++) {
   // fork/create a worker process
}

for (;;) {
   // wait for connection
    sd = accept(server_sd);

   // find idle worker to handle request
   pass_to_worker(sd);
}
```

What's wrong with this approach?

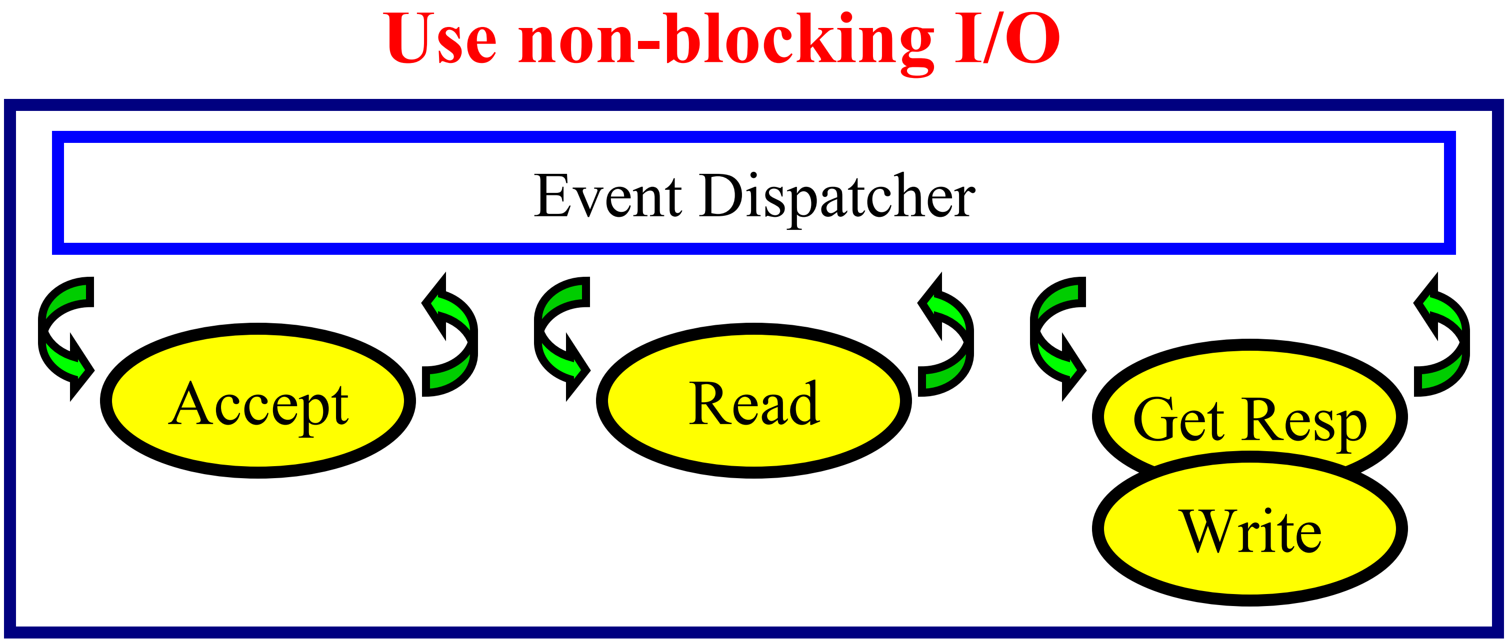
A Pre-Forking Server

```
for (i=0; i<P; i++) {
   // fork/create a worker process
}

for (;;) {
   // wait for connection
    sd = accept(server_sd);

   // find idle worker to handle request
   pass_to_worker(sd);
}
```

What's wrong with this approach?

What is a good value for P (# of workers)?

Single Process Event Driven (SPED)



Single Process Event Driven (SPED)

```
for (;;) {
  n = get_events(&eventlist);

  for (i=0; i<n; i++) {
    sd = eventlist[i].fd;
    if (is_read_event(eventlist[i])) {
      if (sd == server_sd) {
        // get new connection
        newsd = accept(server_sd);
      } else {
        read_request(sd); parse_request();
      }
    }
    if (is_write_event(eventlist[i])) {
      get_and_write_response(sd);
    }
  }
}
```

Single Process Event Driven (SPED)

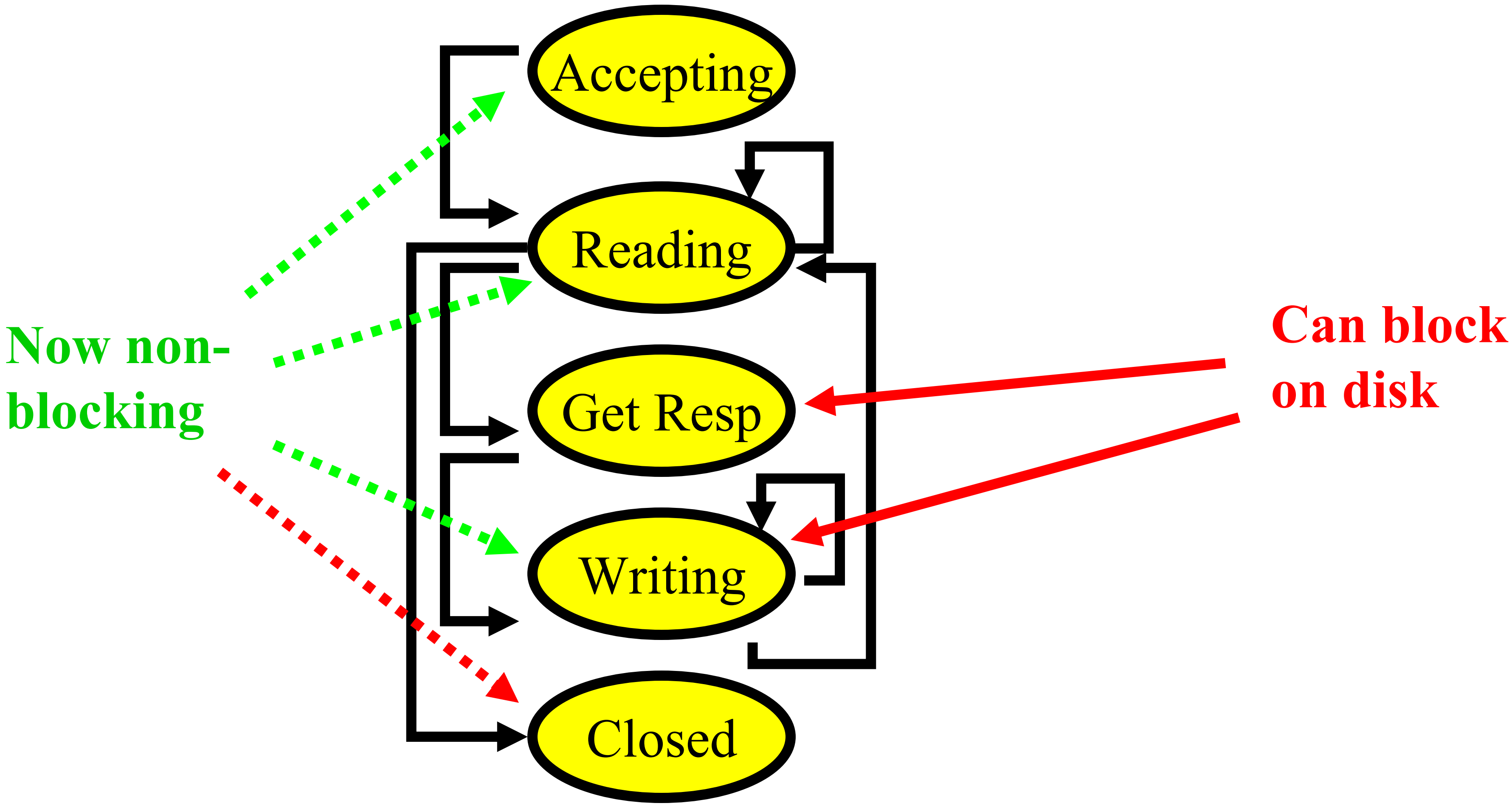
```
for (;;) {
  readfdset = rdset; writefdset = wrset;
  n = select(max_sd, &readfdset, &writefdset,
             &exceptfds, &timeout);
  for (i=0; i<max_sd; i++) {
    if (FD_ISSET(i, &readfdset)) {
      if (i == server_sd) {
        // get new connection
        sd = accept(server_sd);
        FD_SET(sd, &rdset); FD_SET(sd, &wrset);
      } else {
        read_request(i); parse_request();
      }
    }
    if (FD_ISSET(i, &writefdset)) {
      get_and_write_response(i);
    }
  }
}
```

Single Process Event Driven (SPED)

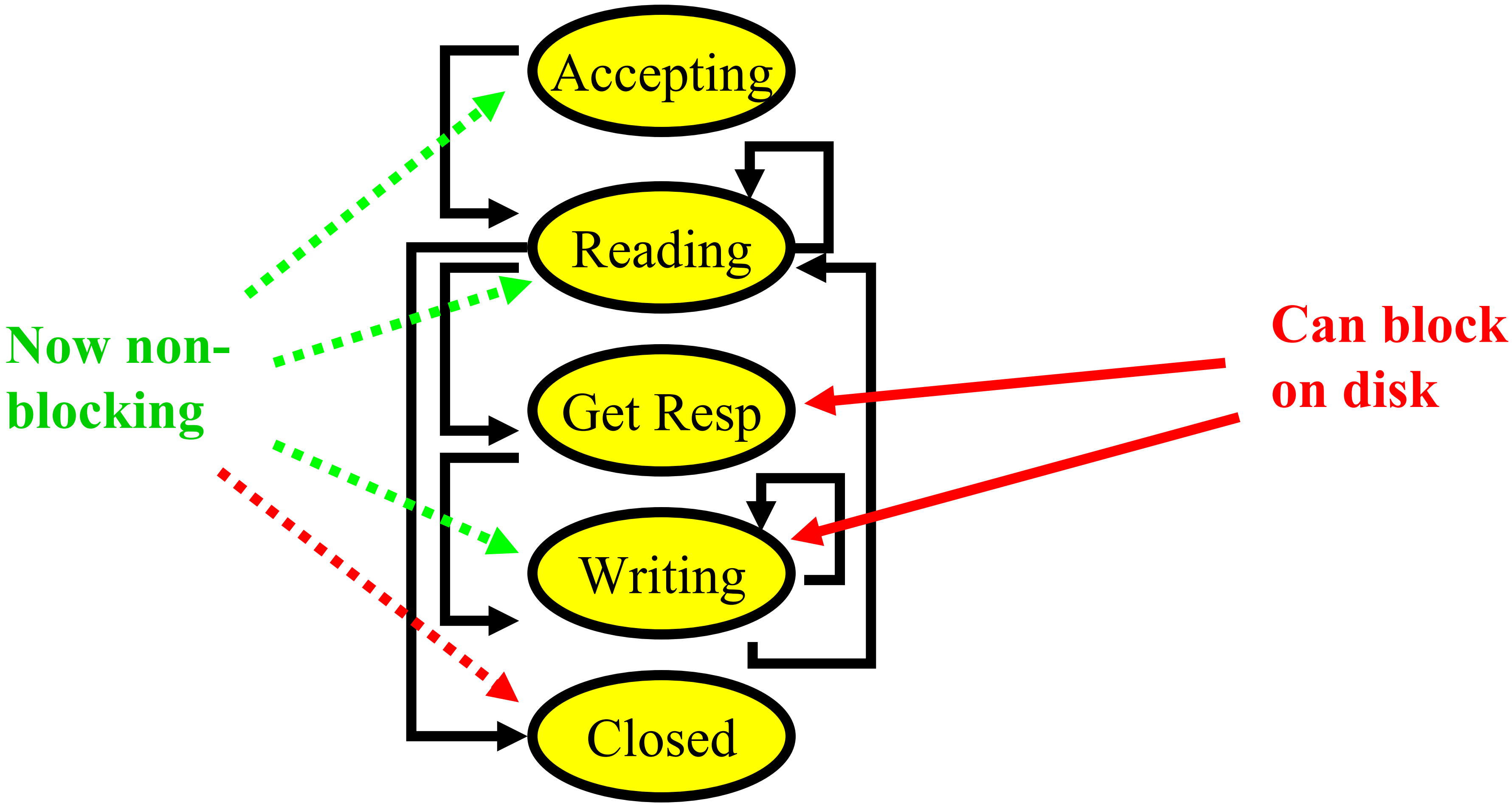
```
for (;;) {
  readfdset = rdset; writefdset = wrset;
  n = select(max_sd, &readfdset, &writefdset,
             &exceptfds, &timeout);
  for (i=0; i<max_sd; i++) {
    if (FD_ISSET(i, &readfdset)) {
      if (i == server_sd) {
        // get new connection
        sd = accept(server_sd);
        FD_SET(sd, &rdset); FD_SET(sd, &wrset);
      } else {
        read_request(i); parse_request();
      }
    }
    if (FD_ISSET(i, &writefdset)) {
      get_and_write_response(i);
    }
  }
}
```

What are the pros and cons of this approach?

Single Process Event Driven (SPED)

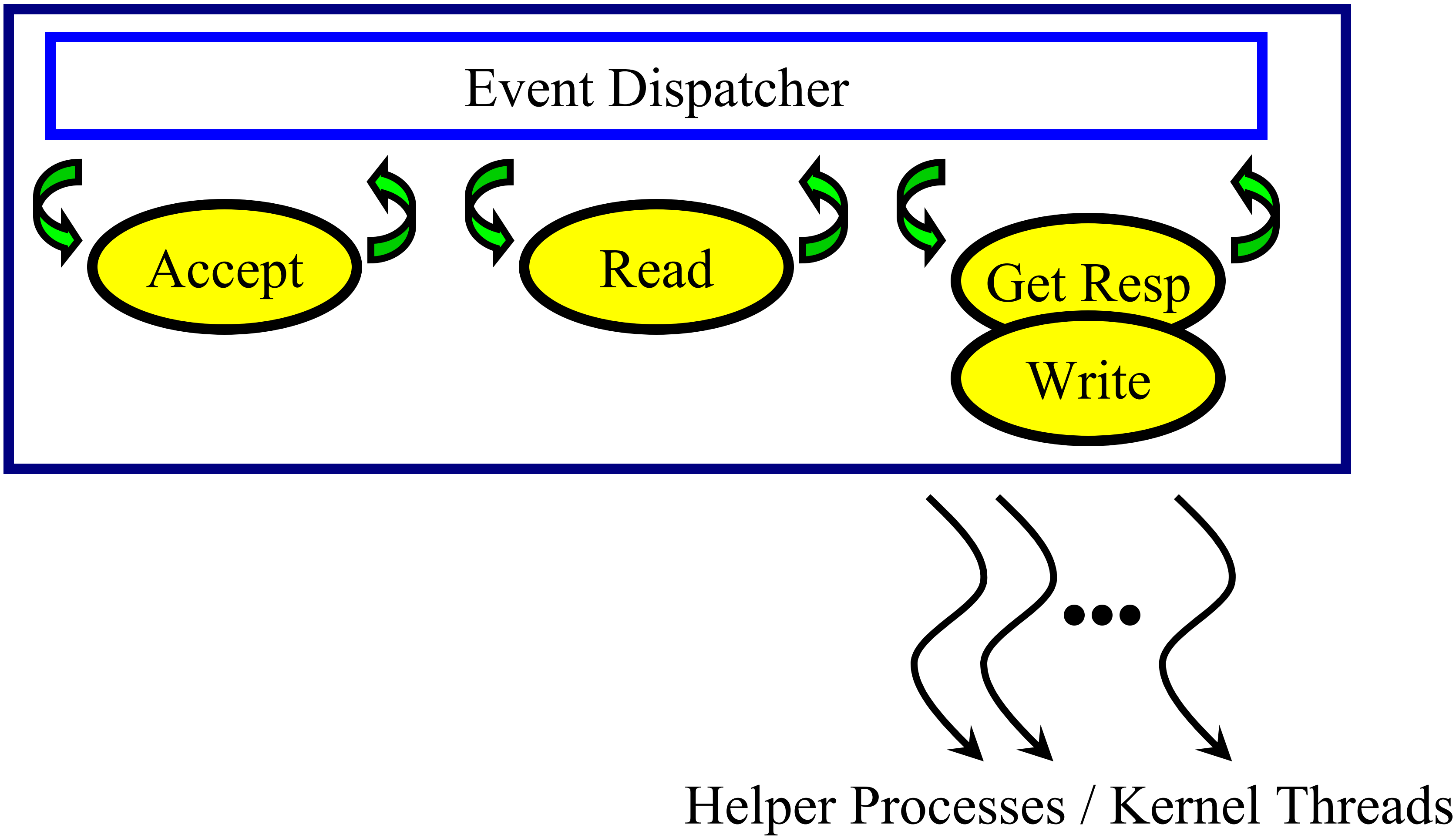


Single Process Event Driven (SPED)

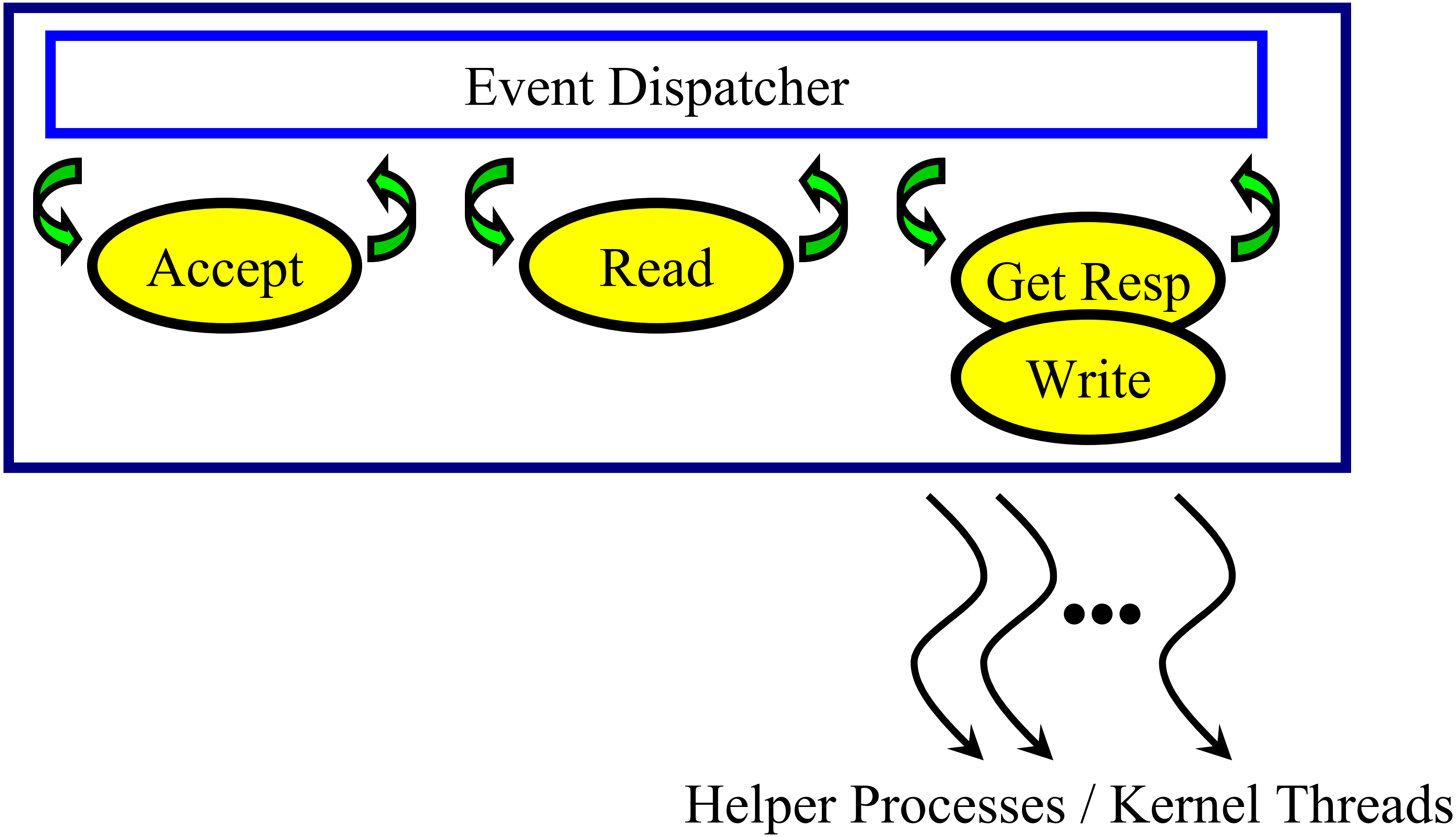


Possible Solutions?

Asymmetric MP Event Driven (AMPED)

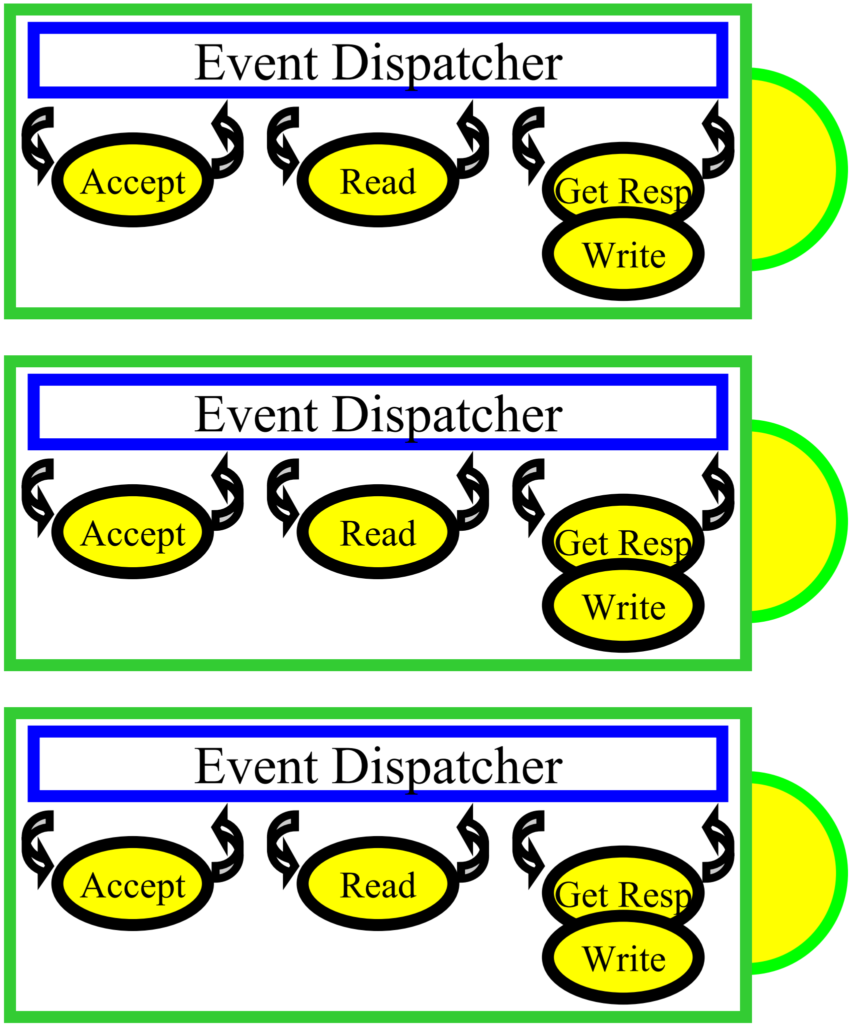


Asymmetric MP Event Driven (AMPED)

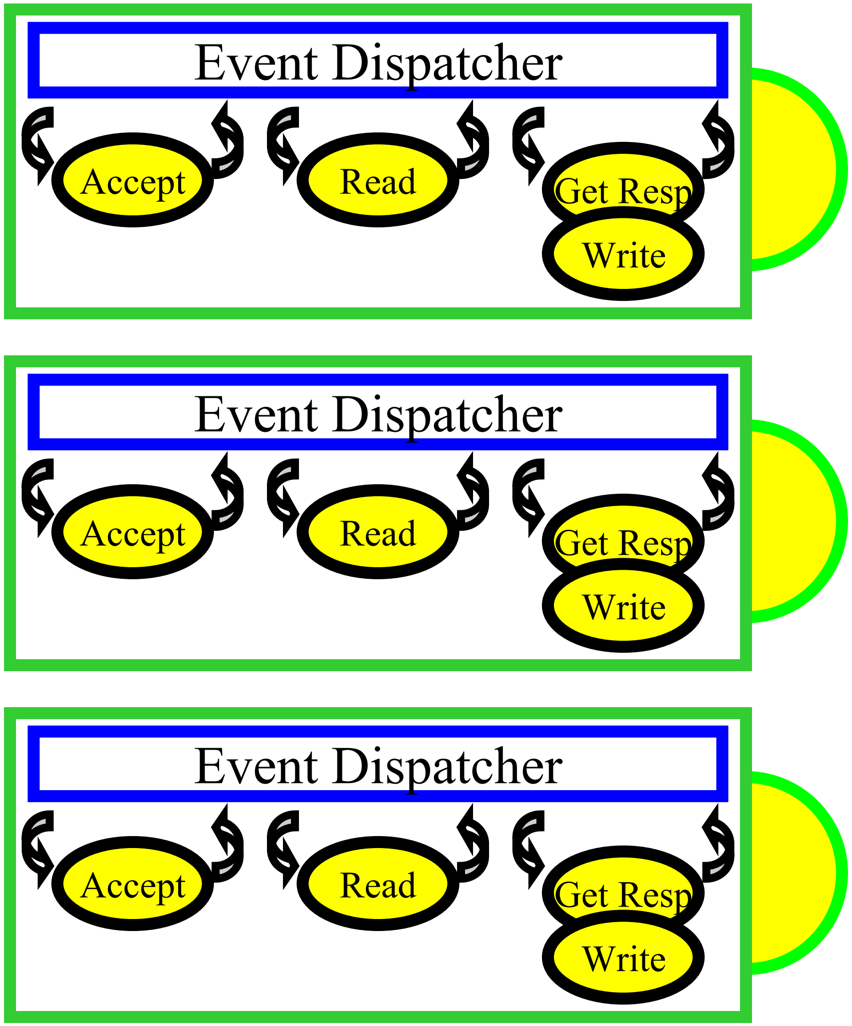


What are the pros and cons of this approach?

N-Copy: 1 SPED Server per CPU

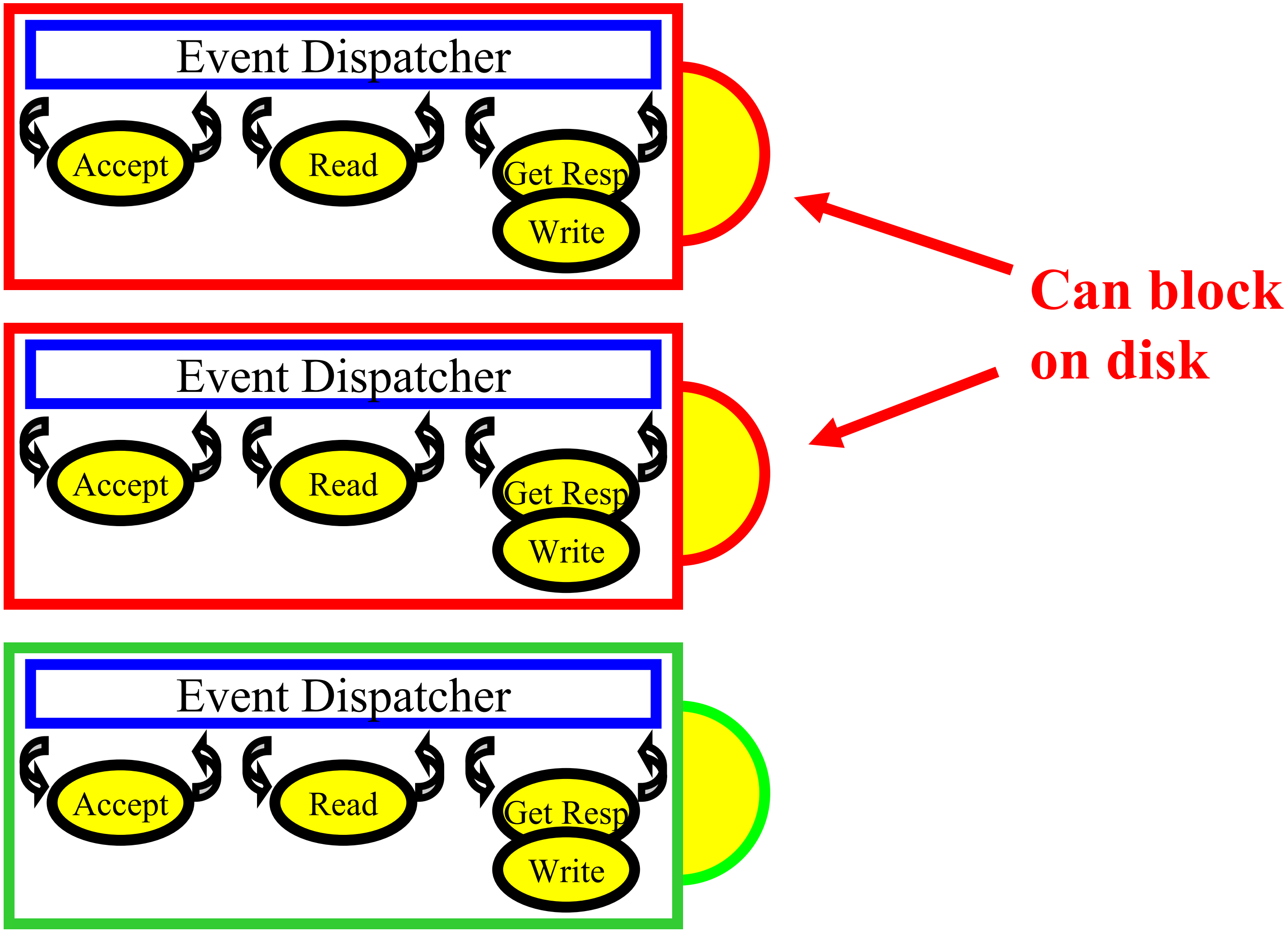


N-Copy: 1 SPED Server per CPU



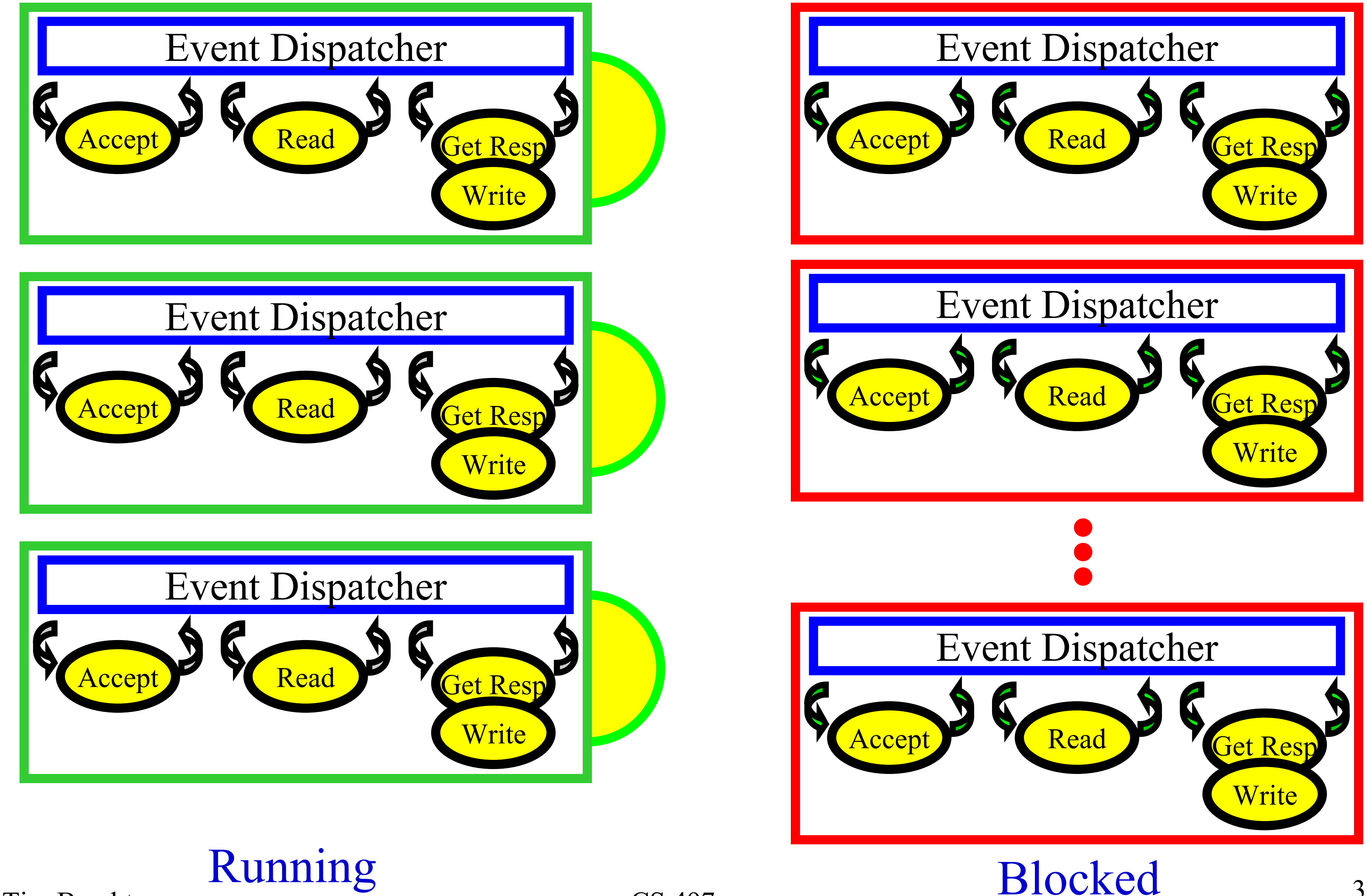
What are the pros and cons of this approach?

N-Copy: 1 SPED Server per CPU



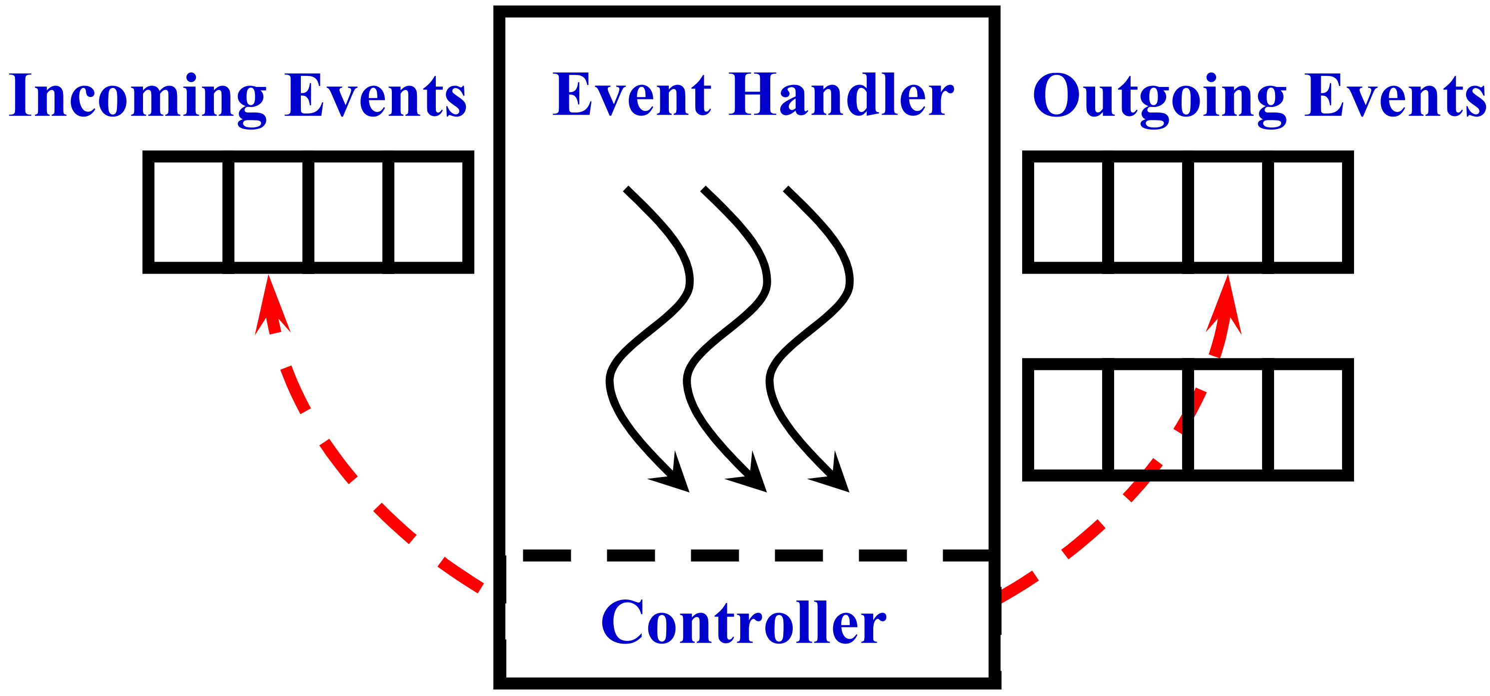
What are the pros and cons of this approach?

SYmmetric MP Event Driven (SYMPED)



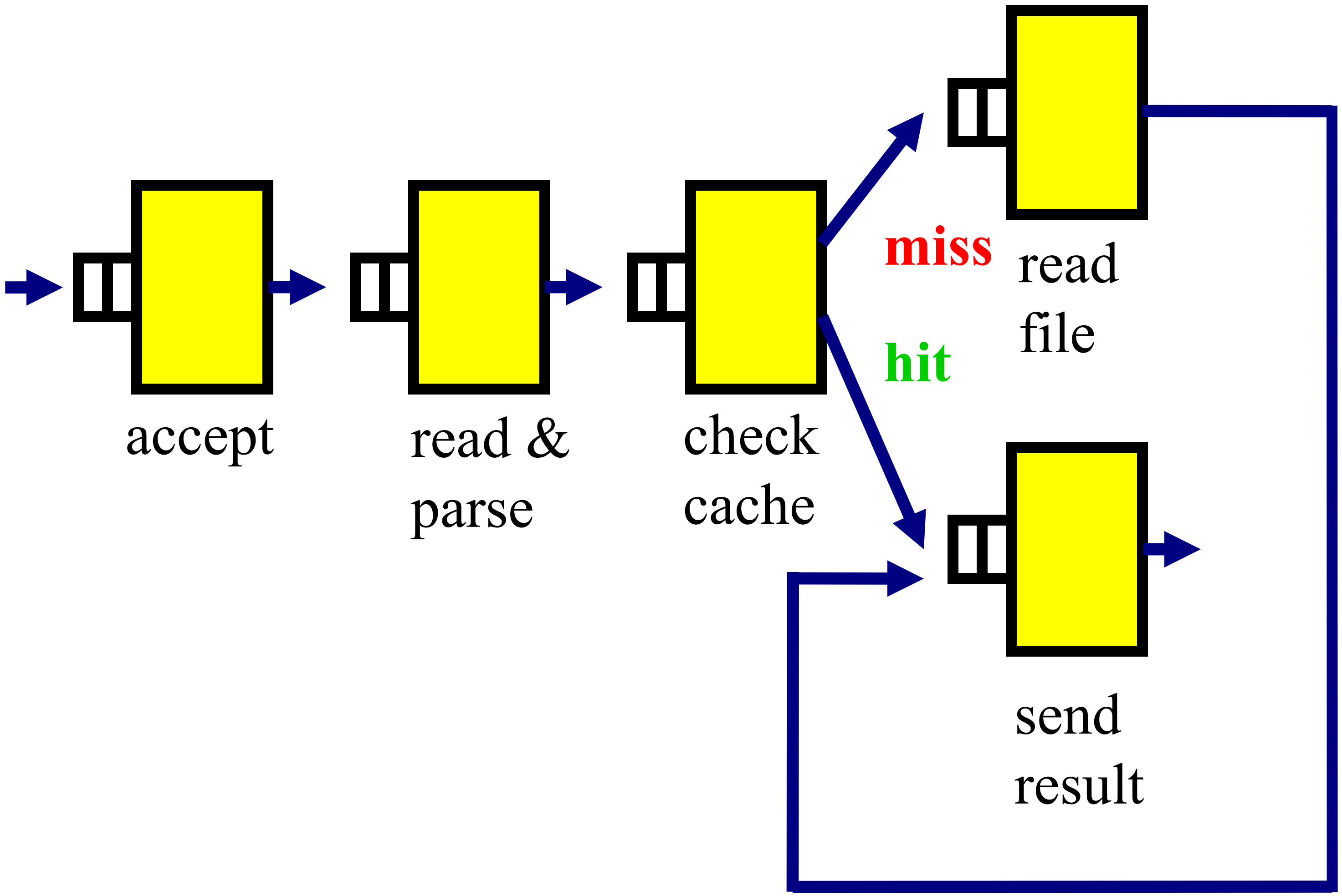
A Hybrid Server: Pipelined (SEDA)

An example of a stage:



- control # of threads
- shed load if needed

Hybrid Servers: Haboob / WatPipe



Overview of Some Servers

- Multi-Thread/Process: one thread/process per conn
  - (MT/MP) **Apache, Knot** [apache.org, von Behren et 03]
- Single Process Event Driven [www.zeus.com]
  - (SPED) **Zeus, Original Harvest/Squid** [Wessels, 96]
  - Asymmetric Multi-Process Event Driven
  - (AMPED) **Flash** [Pai et al, 99]
- One copy per CPU [Zeldovich et al, 03]
  - (N-Copy) **? Rock Web Server ?** [accoria.com]
- SYmmetric Multi-Process Event Driven
  - (SYMPED & Shared-SYMPED) **userver** [UW:Brecht et,]
- Hybrid: Staged Event Driven Architecture / Pipelined
  - (SEDA) **Haboob, WatPipe** [Welsh et 01, Pariag et 07]

Outline

- Part I: Background
  - Web Server Example: HTTP/1.1
  - Server Architectures
  - **Performance Evaluation**

How to Evaluate these Designs?

How to Evaluate these Designs?

- Performance?
- Ease of implementation?
- Ease of maintenance?
- Robustness?

What does Performance Mean?

"**We have improved performance by 48%.**"

What does Performance Mean?

"**We have improved performance by 48%.**"

- What is the performance metrics?
- What is the basis of comparision?
- Under what conditions is this statement true?
- Will this statement be true for you?

What does Performance Mean?

- How does one evaluate the performance of a car?

What does Performance Mean?

- How does one evaluate the performance of a car?
  - Horsepower, Torque?
  - 0-60 mph times? 60-0 times?
  - 0-100 mph times?
  - 0-200 mph times?
  - Track lap times?
  - Track lap times on an icy surface?
  - Number of speakers?
  - Crash test results?
  - Stereo decibel output?
  - Many others?

Which is the best?

What does Performance Mean?

- It means different things to different people
- What are some Web server performance metrics?

What does Performance Mean?

- It means different things to different people
- What are some Web server performance metrics?
  - **Throughput**: requests serviced per unit time
    (server operator / owner / hosting service provider)
  - **Response time**: how long to get response/result
    (user / client)
  - **Revenue**: e.g., dollars of income per unit time
    (owner, executives of the company)
  - **Reliability** (e.g., MTTF), **Recovery time** (crash recovery)
  - (owner, executives e.g., CFO, sys admins)
  - **Many others**

Q: mean, maximum, minimum, distributions?

How to Evaluate Performance?

- **Analytic Model** [Jain, The Art of Comp Sys Perf, 91]
  - mathematical model [CS 457]
  - high-level abstraction capturing the essence
  - (+) easy to change, (+) runs quickly,
    (-) may not capture important details
- **Simulation**
  - must capture key components of behaviour
  - (+) easy to change, (+/-) runtime,
    (-) may be difficult/expensive to capture important details
- **Experimental Evaluation**
  - run experiments on actual hardware
  - (-) hard to change, (-) can run for a long time,
    (+) captures details

Experimental Performance Evaluation

- Benchmark

Experimental Performance Evaluation

- Benchmark
  - A program or set of programs designed to be used to compare performance
  - Meant to be in some way representative of reality
  - Micro-benchmarks
    - small test of idea in isolation (outside of real application and environment)
  - Macro-benchmarks (benchmarks)
    - larger test of a real application in representative environment

Designing a Web Server Benchmark

- What is the goal?
- What is needed?

Benchmark: What is the goal?

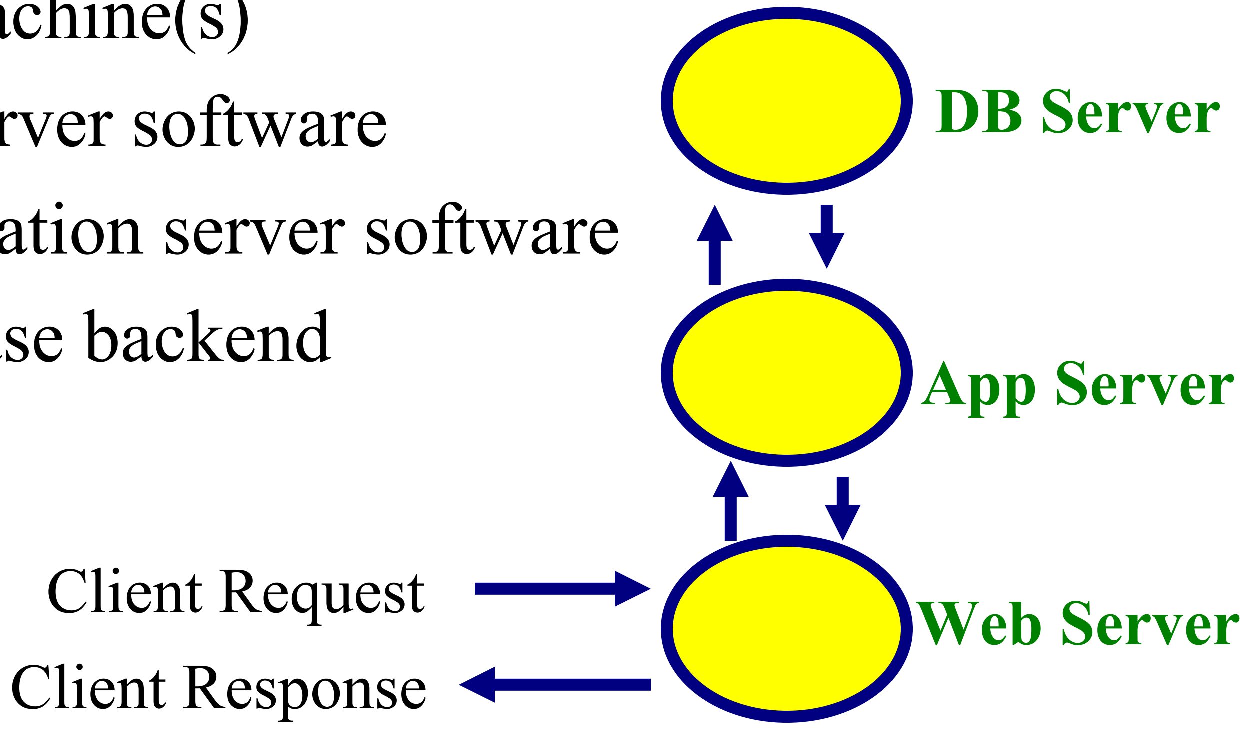
- Compare the performance of one or more of:
  - different machines
  - different web servers
  - different operating systems
  - improvements to web server implementation

Benchmark: What is the goal?

- Compare the performance of one or more of:
  - different machines
  - different web servers
  - different operating systems
  - improvements to web server implementation
- **HOW?**
  - **simulate real users accessing the web site**

Benchmark: What is needed?

- Experimental Environment:
  - **Server**
    - Host/machine(s)
    - Web Server software
    - ?Application server software
    - ?Database backend

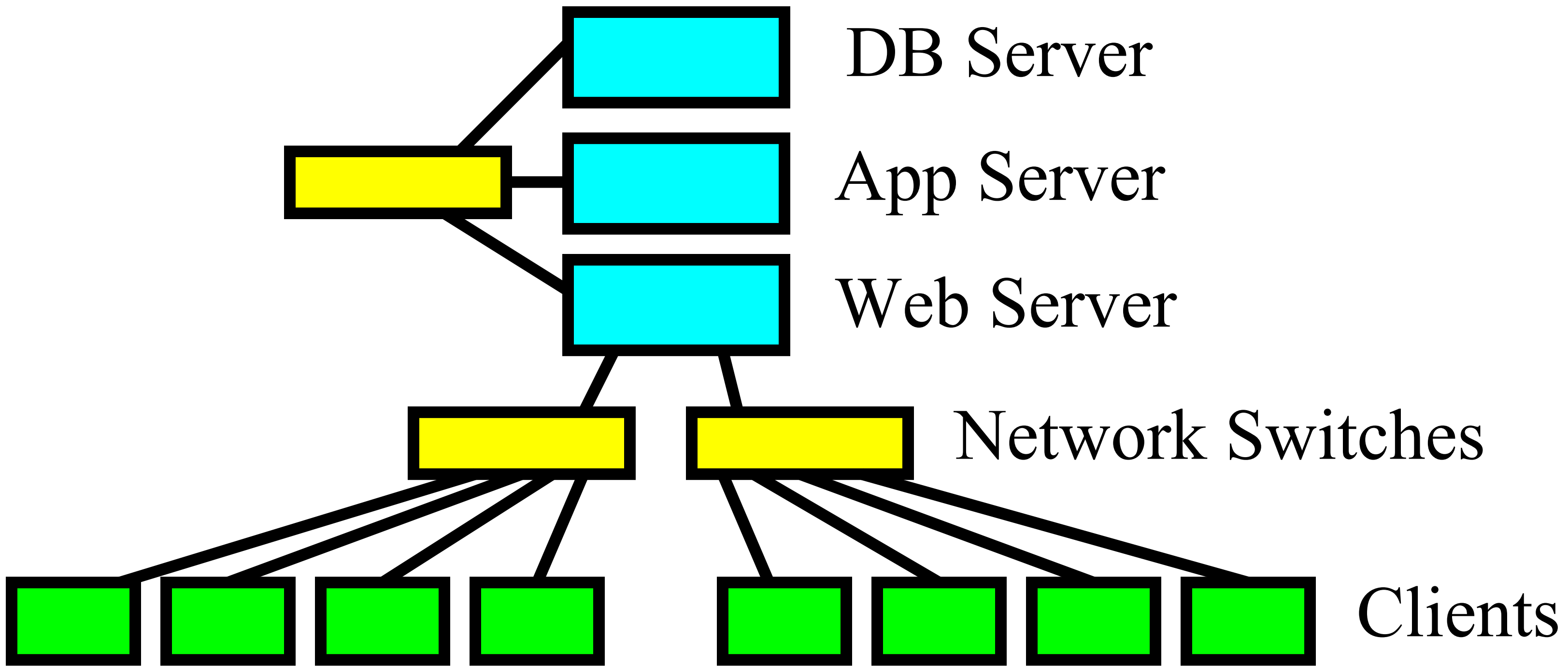


Benchmark: What is needed?

- Experimental Environment:
  - **Clients**
    - Hosts/Machines ... how many?
    - Client simulator software
  - **Networks(s)** to connect clients to servers
    - Server Network Interface Cards (NICs)
    - Client Network Interface Cards (NICs)
    - Network Switches and cables

Benchmark: What is needed?

Example Hardware Configuration/Environement



Benchmark: What is needed?

- Experimental Environment:
  - **Data required on the Server**
    - Files and info for clients to request
    - Data for the database (e.g., things to buy, cost)
  - **Data/Info required for simulated clients**
    - What to request?
    - Which Server NIC to talk to?
    - How long to wait for response?

Benchmark: What is needed?

- Experimental Environment:
  - **Data required on the Server**
    - Files and info for clients to request
    - Data for the database (e.g., things to buy, cost)
  - **Data/Info required for simulated clients**
    - What to request?
    - How long to wait for response?
    - Which Server NIC to talk to?

Q: Where does this data and info come from?

Some Types of Benchmarks

- Trace driven
  - collect requests and times
  - play requests back by clients

    (+) real stream of requests

    (-) can be difficult to modify in meaningful ways
- Characterization driven
  - collect requests and times
  - compute useful stats, use stats to drive workload

Workload Characterization

- Study an environment to try to determine workload
  - Workload is the load inflicted on a service
- Capture the essence of the workload with parameters
- May do this by observing/monitoring
  - Server
  - Client / Users
  - Network traffic

Workload Characterization

- Want benchmarks representative of real environments
  – Modify software (add instrumentation) to track
    - files accessed, when, by who (log file)
    - post process to get relevant info/stats
  – Run this on a real server
    - Ideally a bunch of different servers
  – Collect & analyze data: use in representative bmark

Workload Characterization

- Some server side characteristics:
  - file/info request sequences, rates & distributions
  - number of embedded objects
  - object types (e.g., html, jpg, mpg, etc.)
  - file sizes and distributions (usually by file types)

[Arlitt & Williamson: Invariants 1996]
[Arlitt & Jin: World Cup Soccer 1998]
[Arlitt, Krshnamurthy & Rolia: Shopping 2001]
[Veloso, et al., Streaming media, 2006]

Workload Characterization

- Behaviours of clients:
  - How long does a user typically:
    - Wait for a response?
    - Spend looking at a page?
  - How does browser fetch embedded objects
    - HTTP/1.1 one at a time
    - HTTP/1.0 all in parallel
    - HTTP/1.1 (pipelined – 1 req for N files)

[Cunha, Bestavros & Crovella, Client-based traces, 1995]

Workload Characterization

- Behaviour of network:
  - Network link speeds?
  - How long for a request to reach the server?
  - How long for a response to reach the client?
  - Packet drop rates?
  - What gets dropped, when?

Some Benchmarks

- Standard Performance Evaluation Corporation (SPEC) [spec.org]
  - SPECWeb96 (Static)
  - SPECweb99/_SSL (70% Static, 30% Dynamic)
  - SPECweb2005
    - Banking, Ecommerce, Support
    - Multi-tiered
- Transaction Processing Performance Council
  - TPC-W (Database oriented) [www.tpc.org]

Some Benchmark Clients

- SPECweb clients [SPEC: 96, 99, 2005/6]
- httperf [Mosberger & Jin: 98]
- s-client [Banga & Drushel: 97]
- Surge [Barford & Crovella: 98]

Some Research (Past, Present, Future)

- Server design and implementation (understanding!!)
  - best architecture for performance
  - how to avoid server meltdown under overload
- Client workload generator design and implemenation
  - small # of hosts to simulate large # of users
- Workload characterization
  - What is a representative workload?
  - What does it represent? How do we know?
- Improving operating system support
  - spending large % of execution time in OS / Why?

Part I: The End