Web Server Architectures

CS 4244: Internet Programming

Dr. Eli Tilevich

Based on “Flash: An Efficient and Portable Web Server,” Vivek S. Pai, Peter Druschel, and Willy Zwaenepoel, *1999 Annual Usenix Technical Conference*, Monterey, CA, June 1999.

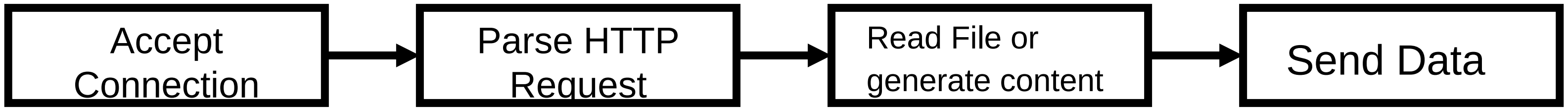
Design Goals

- Performance & Quality of Service (Systems)
  - Good responsiveness; low latency
  - Good concurrency
    - Can support multiple clients simultaneously
  - High throughput
  - Graceful degradation
  - Low memory consumption
- Ease of development (Software Engineering)
  - Simple to understand, fine-tune, add new features, debug, etc.

What Web Servers Do

- In response to a Web client request (e.g., http://google.com/index.html) a Web server:
  - **Accepts** network connection
  - **Parses** the request (index.html)
  - **Reads** file from disk or runs a dynamic content generator
  - **Sends** content (headers and body) back

Single-Threaded Web Server



- One process sequentially handles all client connections
- Simple –requires no synchronization
- Does not scale (one client at a time)

Optimizations?

- Caching
  - Pathname translation
  - Some dynamic content
  - File operations

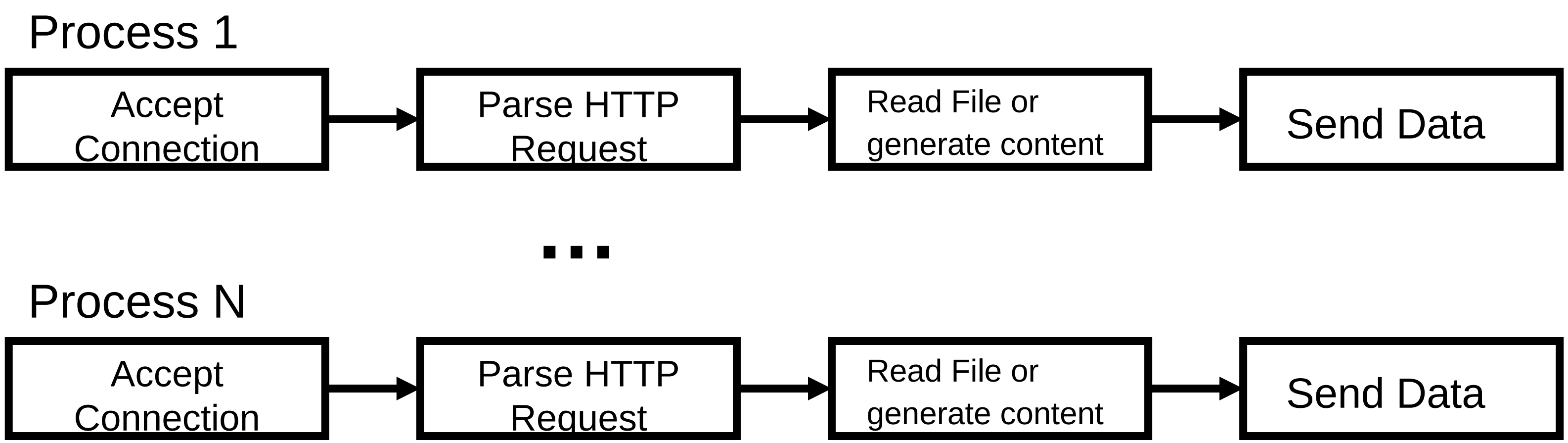
Additional Features of Web Servers

- Logging
- Security (e.g., access control)
- Traffic analysis
- Require centralized data structures to implement

Main Server Architectures

- Multi-process (Apache on Unix)
- Multi-threaded (Apache on NT/XP)
- Single process event driven (Zeus, thttpd)
- Asymmetric multi-process event-driven (Flash)

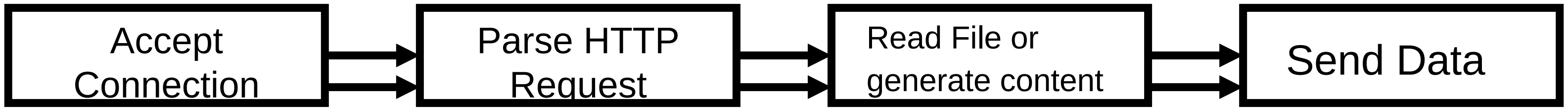
Multi-Process Architecture



- Utilizes multiple processors
- Easy to debug
- Can pre-fork a pool of processes

- Inter Process Communication is difficult and expensive
- High memory cost, context switches

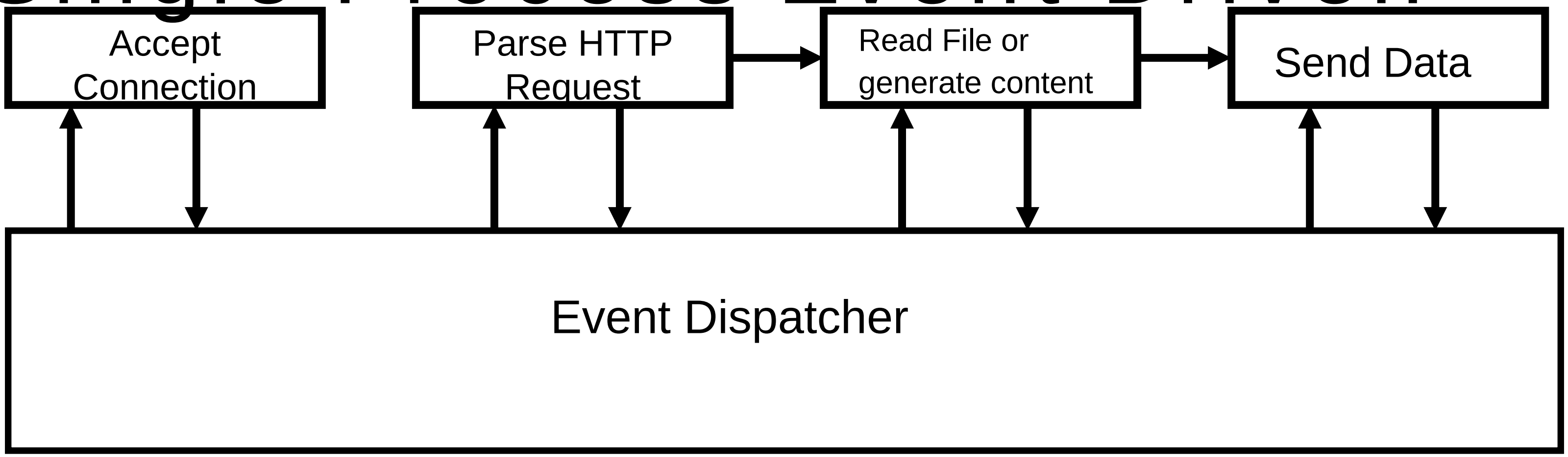
Multi-Threaded Architecture



- Utilizes multiple threads, good performance
- Easy to change threading policies

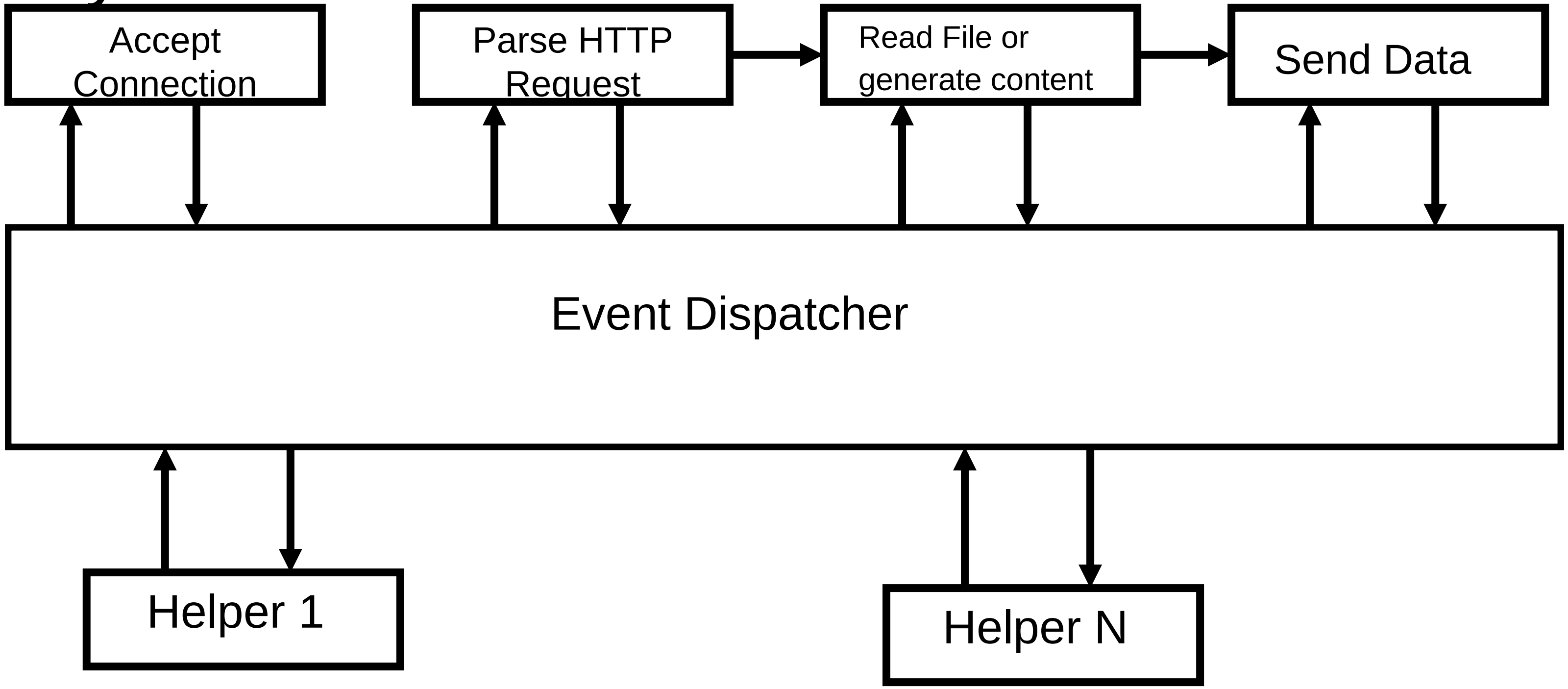
- Need to synchronize, to avoid data races
- Resources utilization (kernel and user-level):
  - memory consumption, context switches, startup
- Blocking I/O can cause deadlocks

Single Process Event-Driven



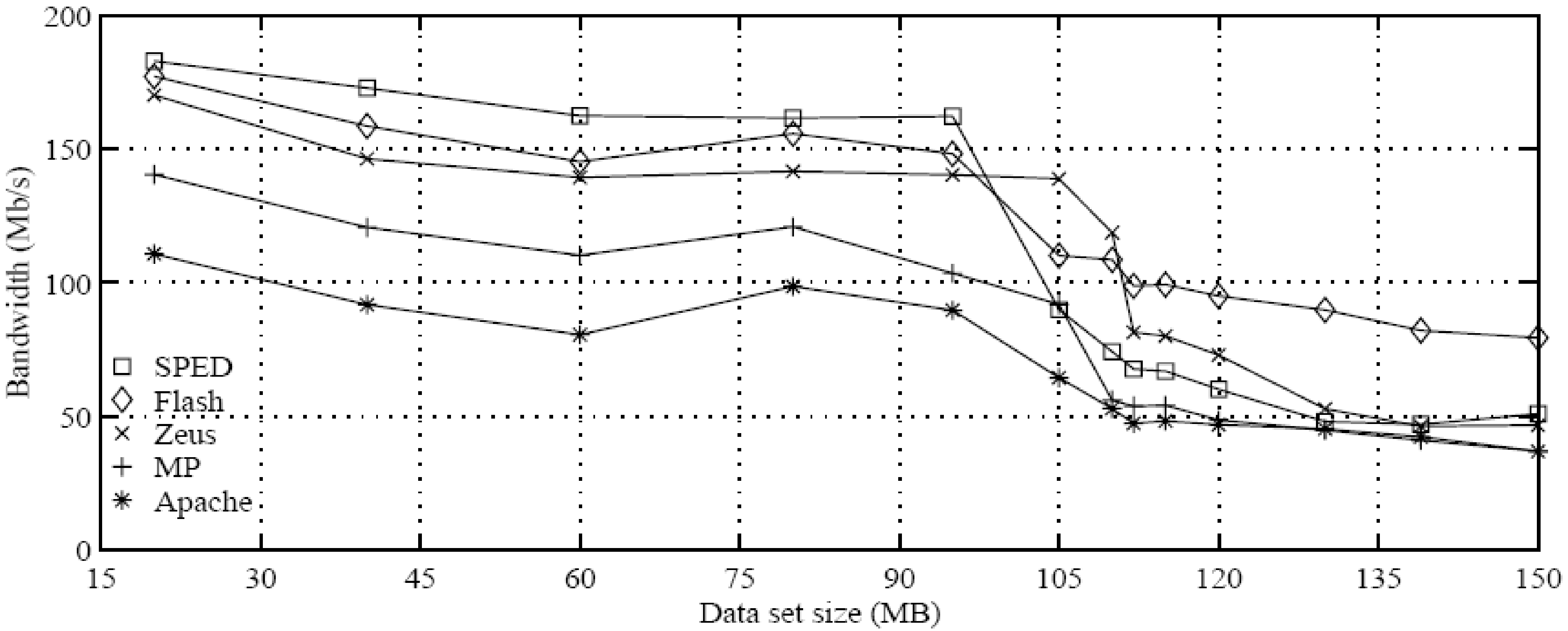
- Use a selector to check for ready file descriptors
- Uses a finite state machine to determine how to move to the next processing stage
- No context switching, no synchronization, single address space
- Modern OS do not provide adequate support for asynchronous disk operations

Asymmetric Multi-Process Event-Driven

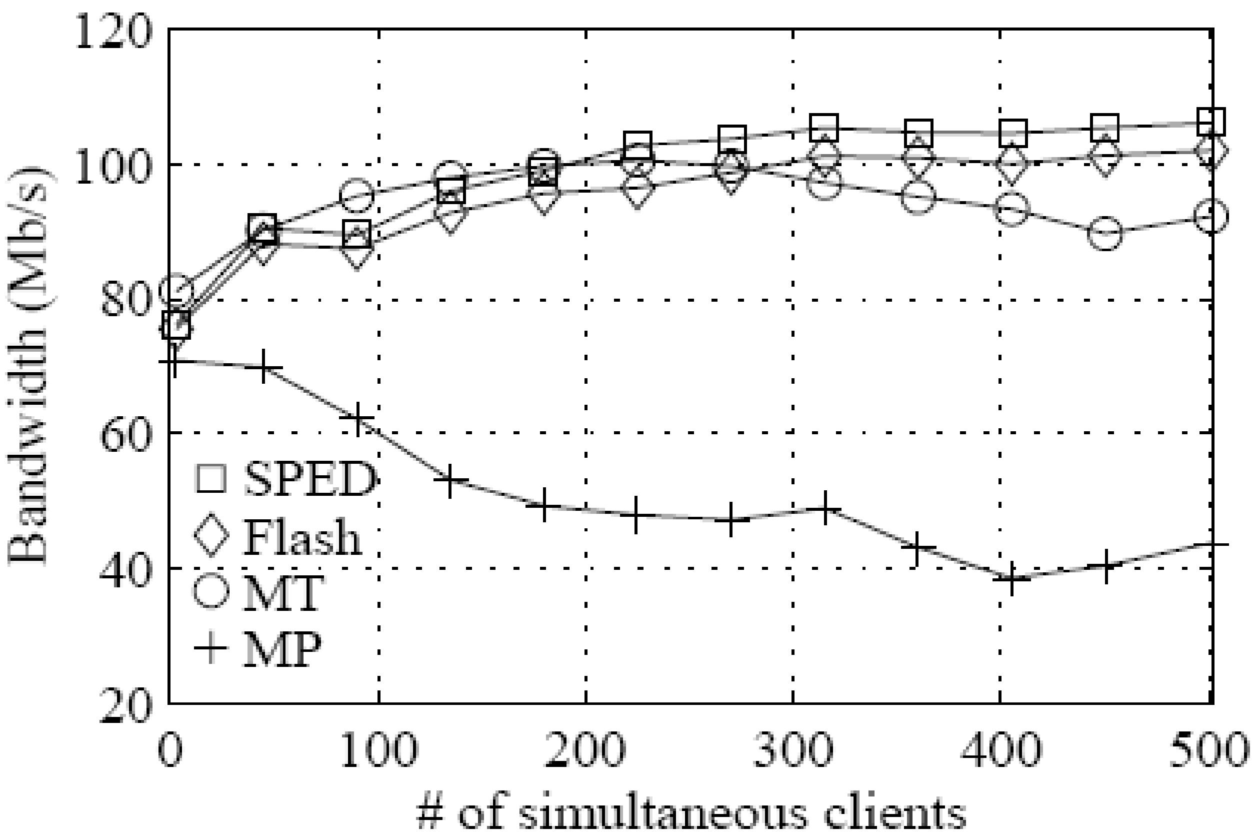


- Similar to Single Process Event-Driven but with helpers for blocking I/O (e.g., disk requests)

Real Workload (On Solaris)



Performance in WAN (Adding Clients)



Performance Comparison

	MP	MT	SPED	AMPED
Disk Blocking	+ (only one proc.)	+	- (whole server proc. blocks)	+
Memory Cons.	- (separate mem. space per proc.)	+	+	+
Disk Usage	+	+	- (one disk request at a time)	+

Challenges of Using Threads

- Need for synchronization
- Deadlocks, starvation
- Race conditions
- Scheduling can be tricky

Challenges of Using Events

- Only one process/thread is used
- High latency for long running handlers
- Control flow is obscure
- Difficult to write, understand, and modify

Hybrid Approach

- SEDA: Staged Event-Driven Architecture
  - **SEDA: An Architecture for Well-Conditioned, Scalable Internet Services**, Matt Welsh, David Culler, and Eric Brewer. In *Proceedings of the Eighteenth Symposium on Operating Systems Principles (SOSP-18)*, Banff, Canada, October, 2001.
- Uses thread pools and events
- Events organized into stages
- Each stage has a thread pool
- Load conditioning; graceful degradation