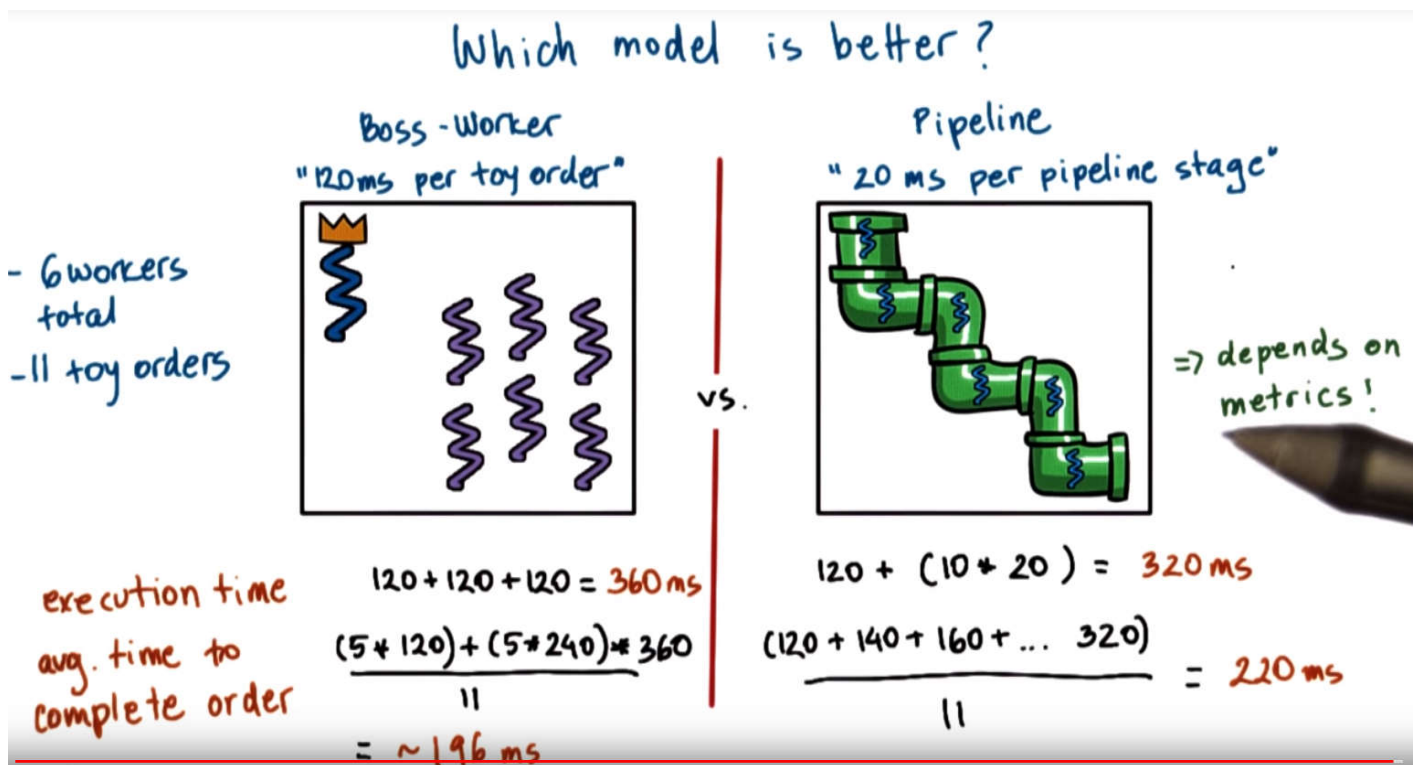


P2L5 Thread Performance Consideration

Goal:

- Performance Comparisons
 - multi-process
 - multi-threaded
 - event-driven
- Event-driven architectures
- Flash: Efficient and protable web server vs. Apache

1. How to compare models?



- For execution time (time it takes to process the same amount of job) pipeline model wins.
- For average time to complete order, Boss-Worker model wins.
- When making comparisons, what metrics we are considering if important.

Are threads useful?

- who is asking? – matrix multiplication application or a web service application?
- What do we care about?
- The answer is not that simple, it all depends
 - Depends on metrics
 - Depends on workload

- Different type of tasks (graphs, file patterns)

What is useful?

For a matrix multiply application...
⇒ execution time

⇒ depends on metrics

For a web service application...
⇒ number of client requests / time } average, max, min, 95%
⇒ response time

For hardware...
⇒ higher utilization (e.g., CPU)

Visual Metaphore

Visual Metaphor

"Metrics exist for operating systems and for toy shops"

Throughput

- process completion rate

Response time

- avg. time to respond to input (e.g., mouse click)

Utilization

- percentage of CPU

Many more...

Throughput

- How many toys per hour?

Response time

- Avg. time to react to a new order

Utilization

- Percent of workbenches in use over time

Many more...

Performance Metrics

Performance Metrics

metrics == a measurement standard

- measurable and/or quantifiable property...
- of the system we're interested in...
- that can be used to evaluate the system behavior

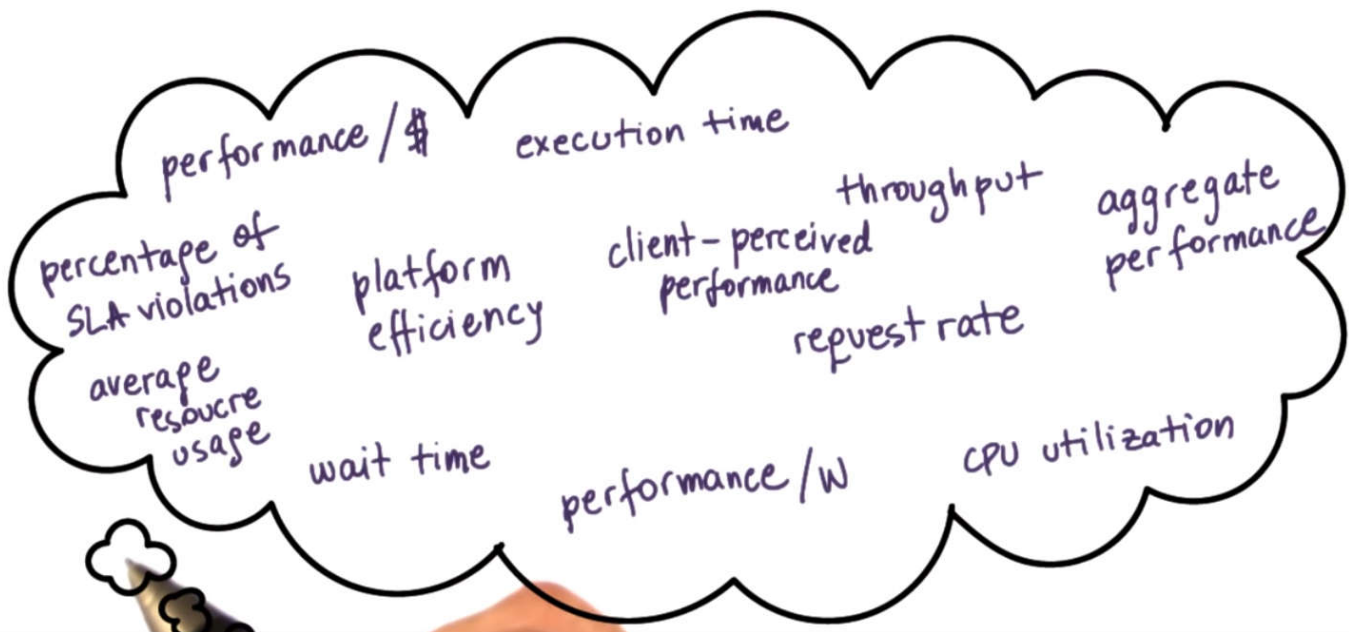
Examples

execution time

software
implementation
of a problem

its improvement
compared to
other
implementations

Performance Metrics

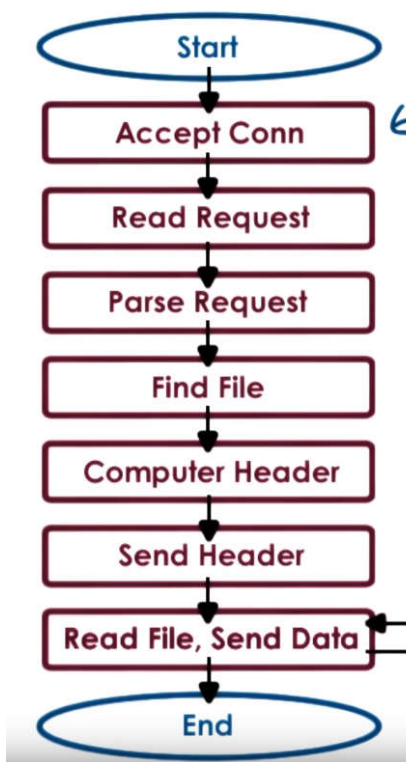


2. Multi-threading vs. Multi-Processing

How to best provide concurrency?

- Context: web server
- Task: concurrent processing of client requests

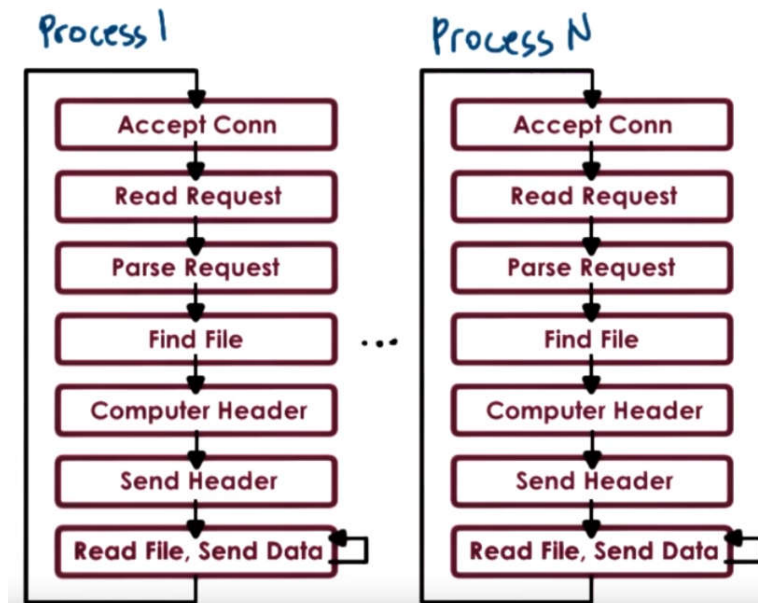
- Multiprocess vs. Multithreaded



Steps in a Simple Web Server

1. client/browser send request
2. web server accepts request
3. server processing steps
4. respond by sending file

Multi-process Web Server



Multi Process Web Server

Multi Process (MP)



simple programming

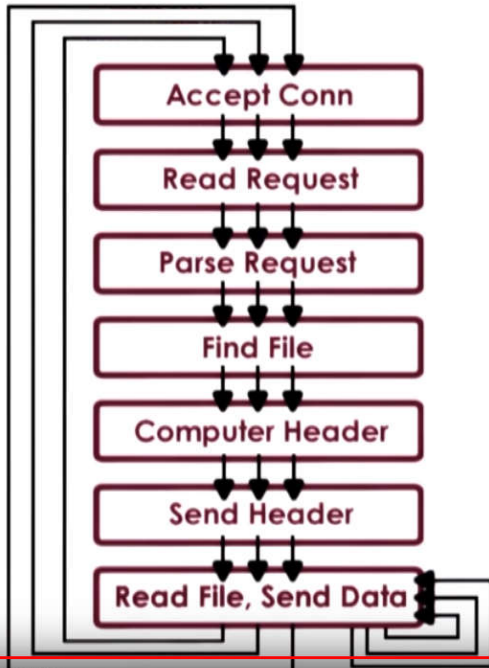


many processes =>
high memory usage
costly context switch
hard/costly to maintain shared state
(tricky port setup)

Multi-threaded Web Server

Multi Threaded Web Server

Multi Threaded (MT)



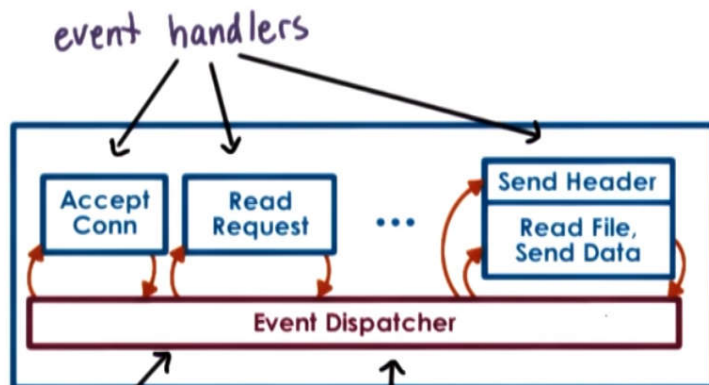
shared address space
shared state
cheap context switch



not simple implementation
requires synchronization
underlying support for threads

3. Event-Driven Model

Event - Driven Model

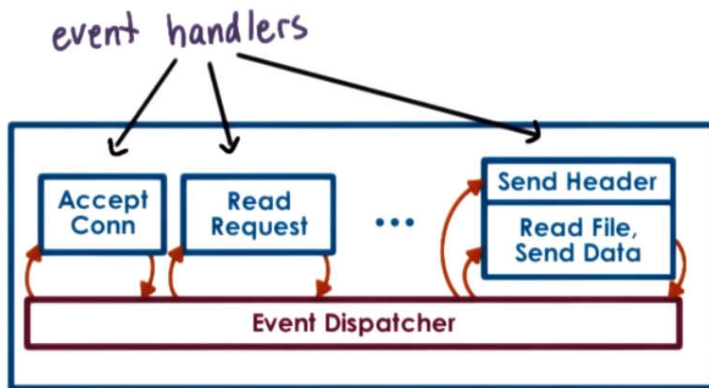


Dispatcher == state machine
external events
⇒ call handler == jump to code

Events

- receipt of request
- completion of send
- completion of disk read

Event - driven Model



Dispatcher == state machine
external events
⇒ call handler == jump to code

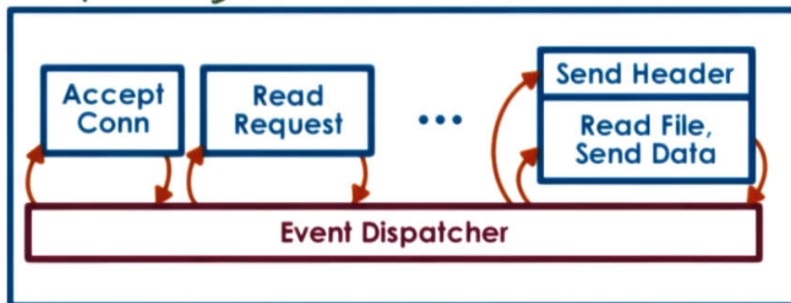
Handler

- run to completion
- if they need to block
⇒ initiate blocking operation and pass control to dispatch loop

Concurrency in Event-Driven Model

Concurrent Execution In Event-Driven Model

single thread switches among
processing of different requests



client C1 ⇒ wait on send
client C2 ⇒ wait on disk I/O
client C3 ⇒ wait on recv

MP and MT:
1 request per execution
context (process/thread)

Event-driven:
Many requests interleaved
in an execution context

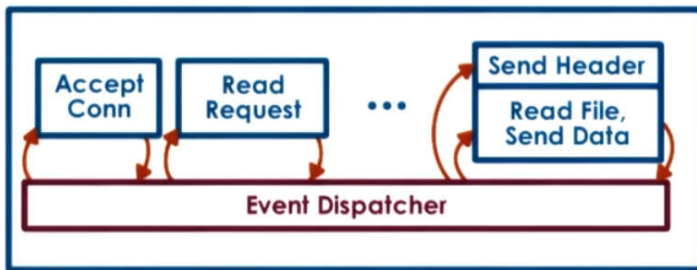
- Even though there is only one execution context/ single thread, there are concurrent executions of multiple requests that are handled interleave.

Why does this work?

- What is the benefit of having one thread switching between processing of different requests.

- How does this approach compare to assigning to different threads or even to different processes.

WHY does this work?



on 1 CPU "threads hide latency"
 if ($t_idle > 2 * t_ctx_switch$)
 \Rightarrow ctx-switch to hide latency
 if ($t_idle == \emptyset$)
 context switching just wastes cycles
 that could have been used for
 request processing

Event Driven:

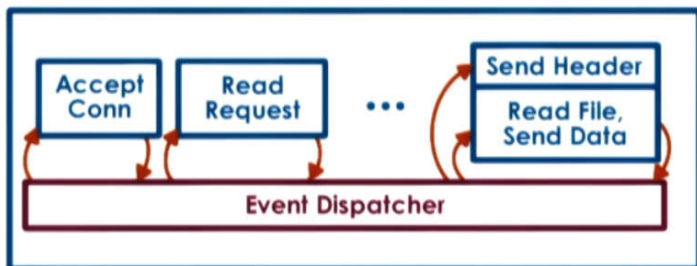
- process request until wait necessary
 then switch to another request

multiple CPUs \Rightarrow
 multiple event-driven processes

- In single threaded event-driven model, it'll process the requests until idle is necessary then switch to another request, hence avoid unnecessary context switching.
- If we have multiple CPUs, it still makes sense to use event-driven model especially when we have to handle more concurrent requests than the number of CPUs.
- Each CPU could host a single event-driven process and then handle multiple concurrent request within that one context (so that no context switch on the CPUs, reducing overhead).

How to implement Event-Driven Model?

HOW does this work?



event == input on file descriptor (FD)

Which file descriptor?

- select()
- poll()
- epoll()

file descriptors

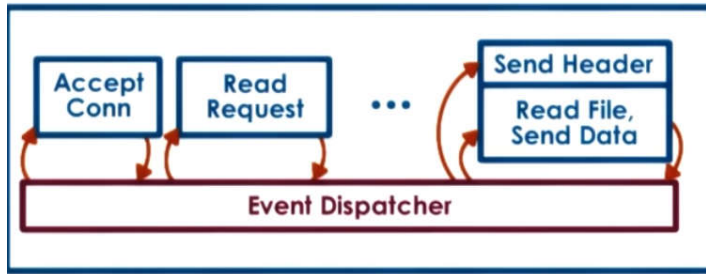
sockets

files

network

disk

Benefits of Event-Driven Model



single address space
single flow of control



smaller memory
requirement
no context
switching

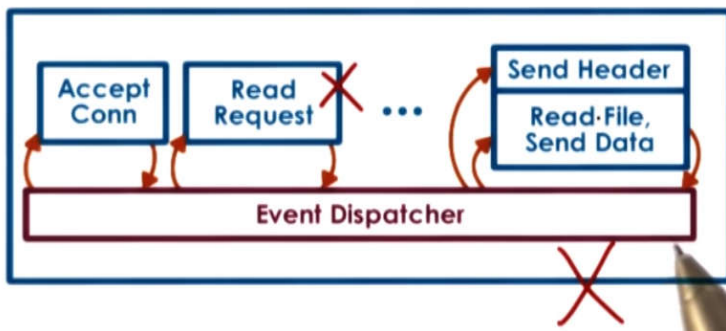


no synchronization

- Jumping over the code base of the server

Problem with Event-Driven Model

problem with EVENT-Driven Model



- a blocking
request/handler
will block the
entire process

Solution:

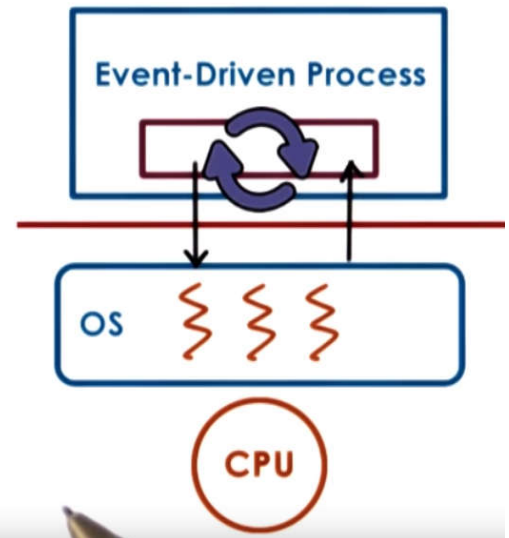
Asynchronous I/O Operations

Asynchronous System Call

- process/thread makes system call
- OS obtains all relevant info from stack, and either learns where to return results, or tells caller where to get results later
- process/thread can continue

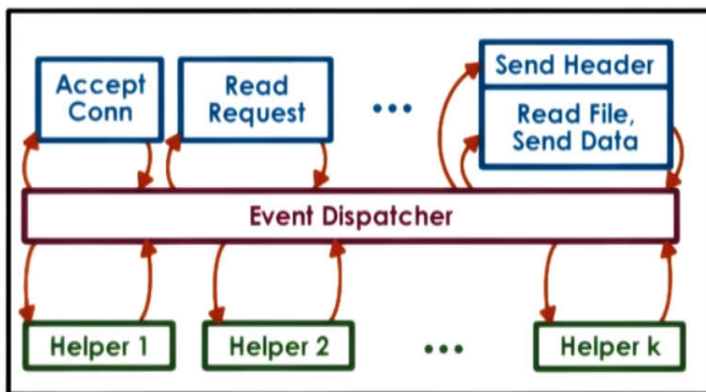
Requires support from **kernel** (e.g., threads) and/or **device** (e.g., DMA)

=> Fits nicely with event-driven model



- Only possible because the OS kernel itself is multi-threaded, so while the caller thread continues execution, another kernel thread does all the necessary work and waiting that's needed to perform the I/O operation
- Asynchronous I/O operation fits nicely with event-driven model
- Asynchronous calls may not always be available

What if Async Calls Are Not Available ?



Asymmetric Multi-Process
Event-Driven Model

(AMPEM) / (AMTED)

Helpers

- designated for blocking I/O operations only
- pipe/socket based comm. w/ event dispatcher => `select()`/`poll()` still ok!
- **helper blocks**, but main event loop (and process) will not!

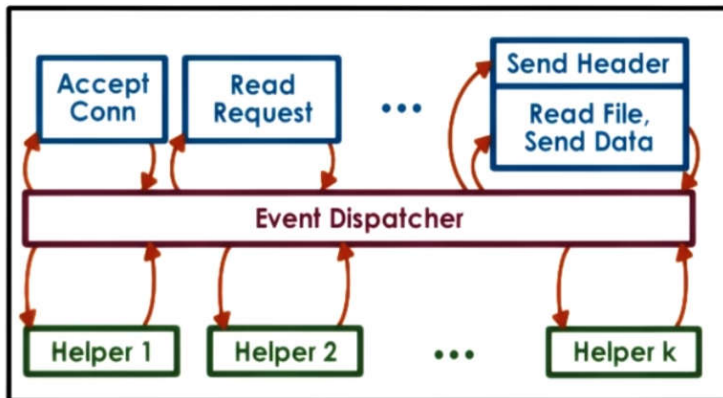
Asymmetric Model:

- At the time the Flash paper is written, not all kernels are multi-processed, so they make the helpers processes.

- Asymmetric: The helper side only deals with blocking I/O operations and the other side (upper side) deals with everything else.

Asymmetric Model Benefits & Downsides:

AMPED / AMTED



Helper Threads / Processes



- + resolves portability limitations of basic event-driven model
- + smaller footprint than regular worker thread



- applicability to certain classes of applications
- event routing on multi CPU systems

4. Flash & Apache Web Server

Flash Web Server



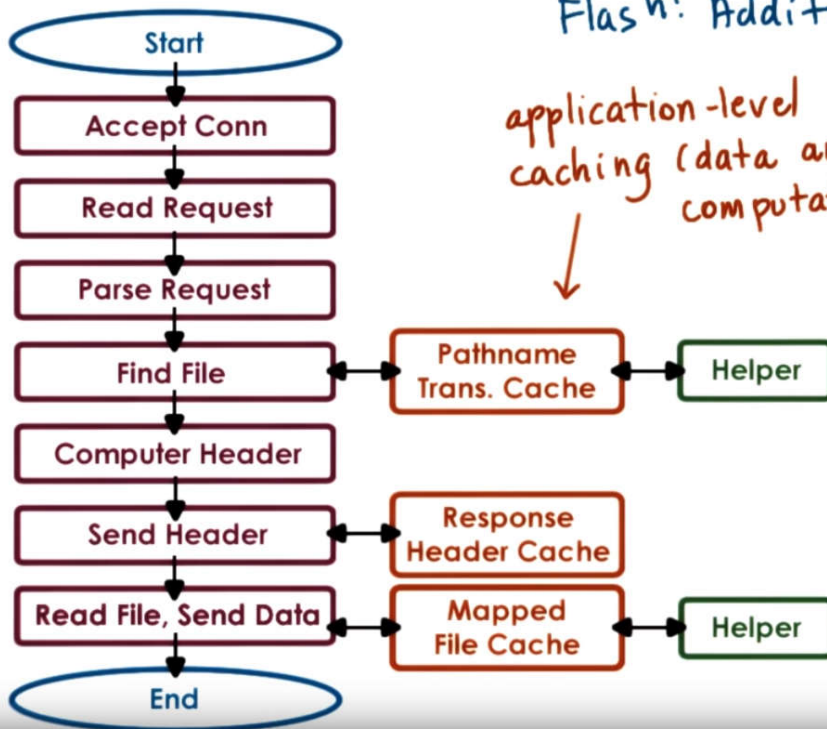
Flash: Event-Driven Web Server

- an event-driven webserver (AMPED)
- with asymmetric helper processes
- helpers used for disk reads
- pipes used for comm. w/ dispatcher
- helper reads file in memory (via mmap)
- dispatcher checks (via mincore) if pages are in mm. to decide 'local' handler or helper

=> possible BIG savings



Flash: Additional Optimizations



application-level
caching (data and
computation)

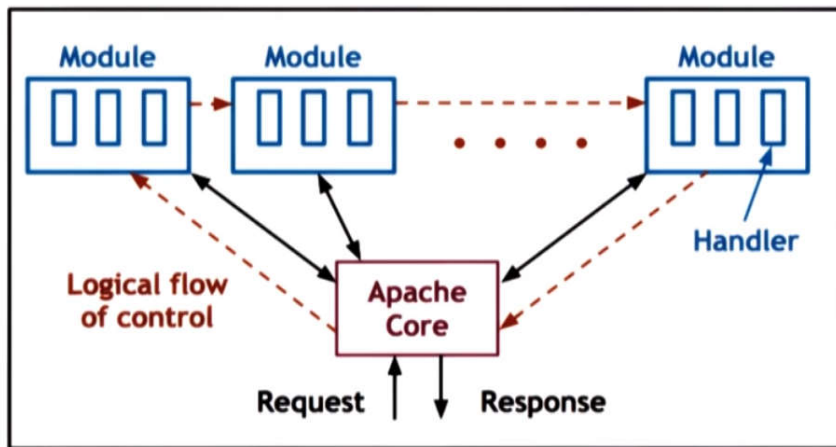
=> Now fairly
common optimizations

+ alignment for DMA
+ use of DMA w/
scatter-gather =>
vector I/O operations

Apache Web Server



Web Server



Core = basic server skeleton
Modules: per functionality

Flow of control: similar to event driven model

BUT

Combination of MP+MT

- each process == boss/worker w/ dynamic thread pool

- # of processes can also be dynamically adjusted

5. Experimental Methodology w Flash Paper as Example

Setting up Performance Comparisons

Define Comparison points

WHAT

systems are you comparing?

Define inputs

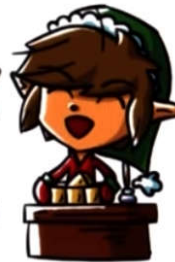
WHAT

workloads will be used?

Define metrics

HOW

will you measure performance?



Flash: define Comparison Points

WHAT systems are you comparing?

- MP (each process single thread)
 - MT (boss-worker)
 - Single Process Event-Driven (SPED)
 - Zeus (SPED w/ 2 processes)
 - Apache (v1.3.1, MP) *
- => compare against Flash (AMPED model)



* for all but Apache optimizations available

Flash: define Inputs

WHAT workloads will be used?

- CS Web Server trace (Rice Univ.)
- owlnet trace (Rice Univ.)
- synthetic workload

Realistic request workload

=> distribution of web page accesses over time

Controlled, reproducible workload

=> trace-based (from real web servers)



Flash: define Metrics

HOW will you measure performance?

Bandwidth == bytes / time

=> total bytes xfered from files / total time

Connection Rate == request / time

=> total client conn / total time



Evaluate both as a function of file size

- larger file size => amortize per connection cost => higher bandwidth
- larger file size => more work per connection => lower connection rate

Experiment Results

Cases:

Flash: define Comparison Points

WHAT systems are you comparing?

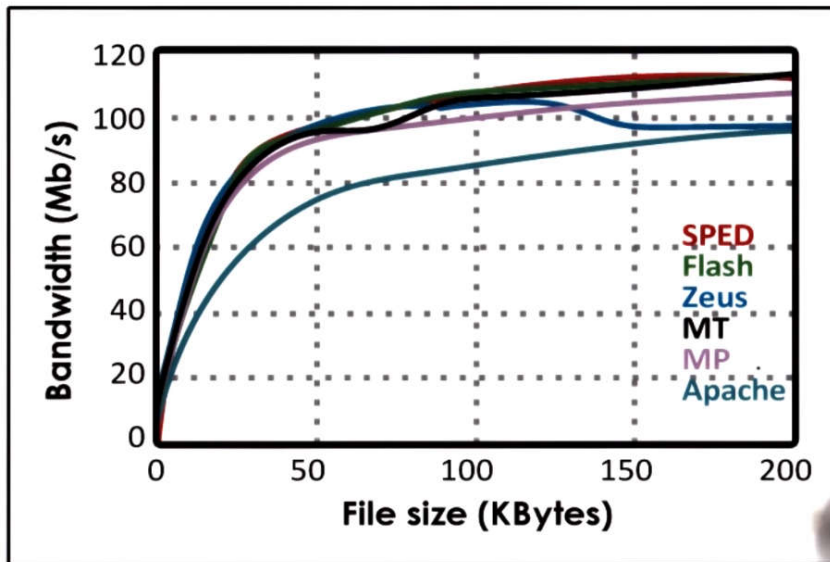
- MP (each process single thread)
 - MT (boss-worker)
 - Single Process Event-Driven (SPED)
 - Zeus (SPED w/ 2 processes)
 - Apache (v1.3.1, MP) *
- => compare against Flash (AMPED model)



* for all but Apache optimizations available

Best Case:

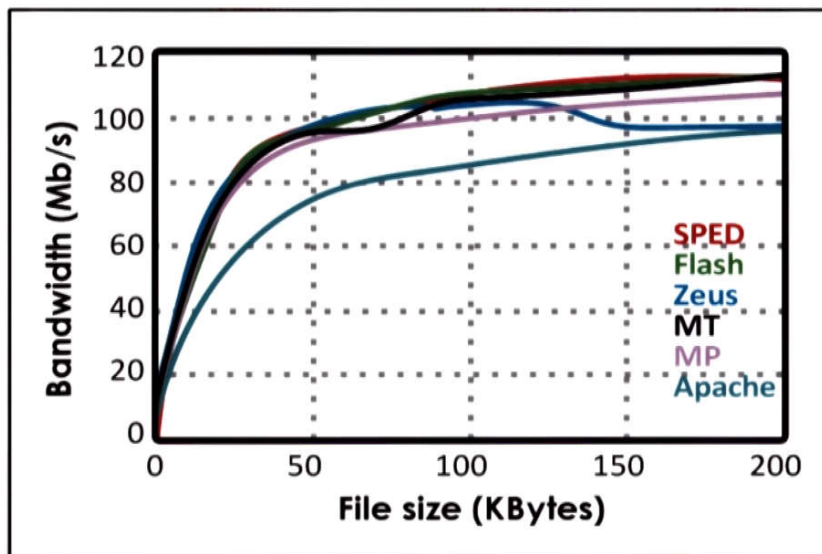
Best Case Numbers



Synthetic load:
- N requests for same file
=> BEST CASE

Measure Bandwidth
- $BW = N * \text{bytes}(F) / \text{time}$
- File size 0-200 KB
=> vary work per request

Best Case Numbers



Observations:
- All exhibit similar results
- SPED has best performance
- Flash AMPED extra check for memory presence
- Zeus has anomaly
- MT/MP extra sync & context switching
- Apache lacks optimizations

OwlNet Trace:

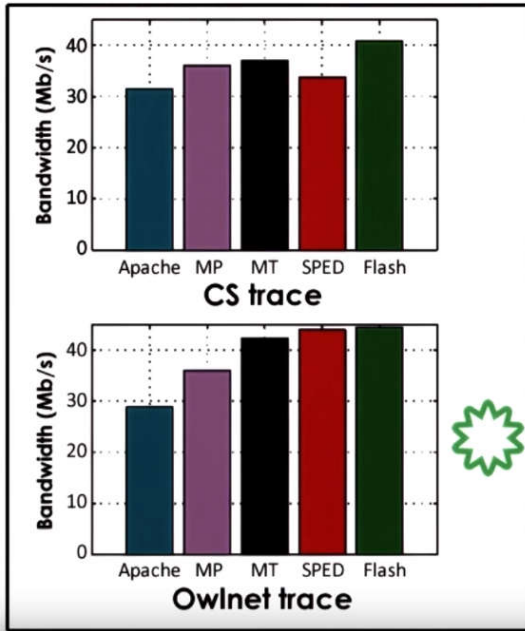
Owlnet Trace

Observations:

- trends similar to "best" case
- small trace, mostly fits in cache
- sometimes blocking I/O is required

=> SPED will block

=> Flash's helpers resolve the problem



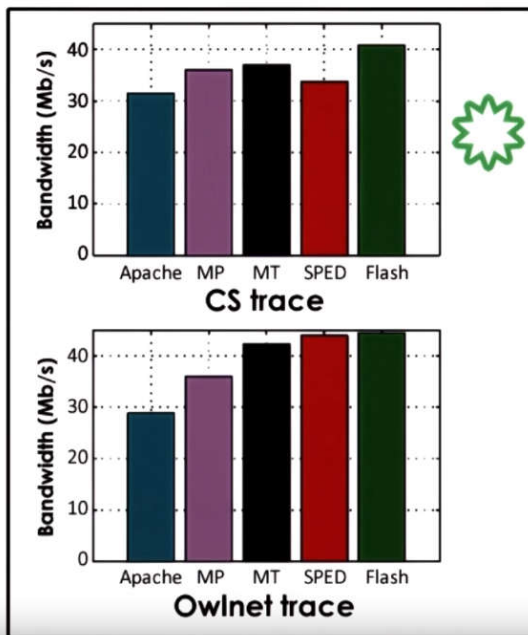
CS Trace:

CS Trace

Observations:

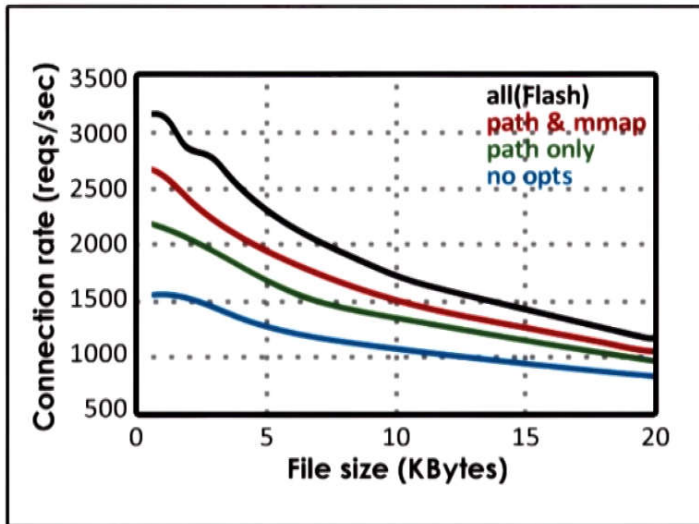
- larger trace, mostly requires I/O
- SPED worst => lack of async I/O
- MT better than MP
 - => memory footprint
 - => cheaper (faster) sync
- Flash best

=> smaller mem footprint
 => more memory for caching
 => fewer requests -> blocking I/O
 => no sync. needed



Impact of Optimizations performed in Flash

Impact of Optimizations



Flash w/ optimizations:
 path == directory lookup
 caching

path & mmap == directory
 lookup + file

all == directory lookup +
 file + header

=> optimizations are
 important!

=> Apache would have
 benefited too!

Summary:

Summary of performance Results

When data is in cache:

- SPED >> Amped Flash
 - unnecessary test for memory presence
- SPED and AMPED Flash >> MT/MP
 - Sync. & context switching overhead

With disk-bound workload

- AMPED Flash >> SPED
 - blocks b/c no async I/O
- AMPED Flash >> MT/MP
 - more memory efficient and less context switching



Advices on Designing Experiments

Design Relevant Experiments

It's Easy ... just run test cases,
gather metrics, and show results!

Not so fast!!!

Relevant experiments \Rightarrow statements
about a solution, that others believe
in, and care for.



Purpose of Relevant Experiments

Example: Web Server Experiment

- Clients: + response time
- Operators: + throughput

Possible goals:

- +response time, +throughput \Rightarrow great!
- +response time \Rightarrow will buy that too.
- +response time, -throughput \Rightarrow may be useful?
- maintains response time when request rate increases



goals \Rightarrow metrics & configuration of experiments

Picking the Right Metrics

"Rule of Thumb" for Picking metrics:

- "Standard" metrics
⇒ broader audience
- Metrics answering the "why? what? who?" questions
 - client performance → response time, # timeout request...
 - operator costs → throughput, costs



Picking the Right Configuration Space

System Resources:

- hardware (CPU, mem...) &
- software (#threads, queue sizes...)

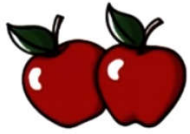
Workload:

- web server: ⇒ request rate, # concurrent requests, file size, access pattern

Now Pick!

- choose a subset of configuration parameters
- pick ranges for each variable factor
- pick relevant workload
- include best/worst case scenarios





Are you comparing Apples to Apples?

Pick useful combinations of factors
- many just reiterate the same point

Compare Apples to Apples!

Poor Example:

- Combo 1: large workload, small resource size
 - Combo 2: small workload, large resource size
- ⇒ conclusion: performance improves when
I increase resources

⇒ WRONG!



What about the competition?
What about the baseline?

Compare system to...

- state-of-the-art
- or most common practice
- ideal best/worst case scenario



Advice on Running Experiments

I've Designed the Experiments...
Now What?

Now, it is Easy:

- run test cases n times
- compute metrics
- represent results.

And don't forget about
making conclusions!

