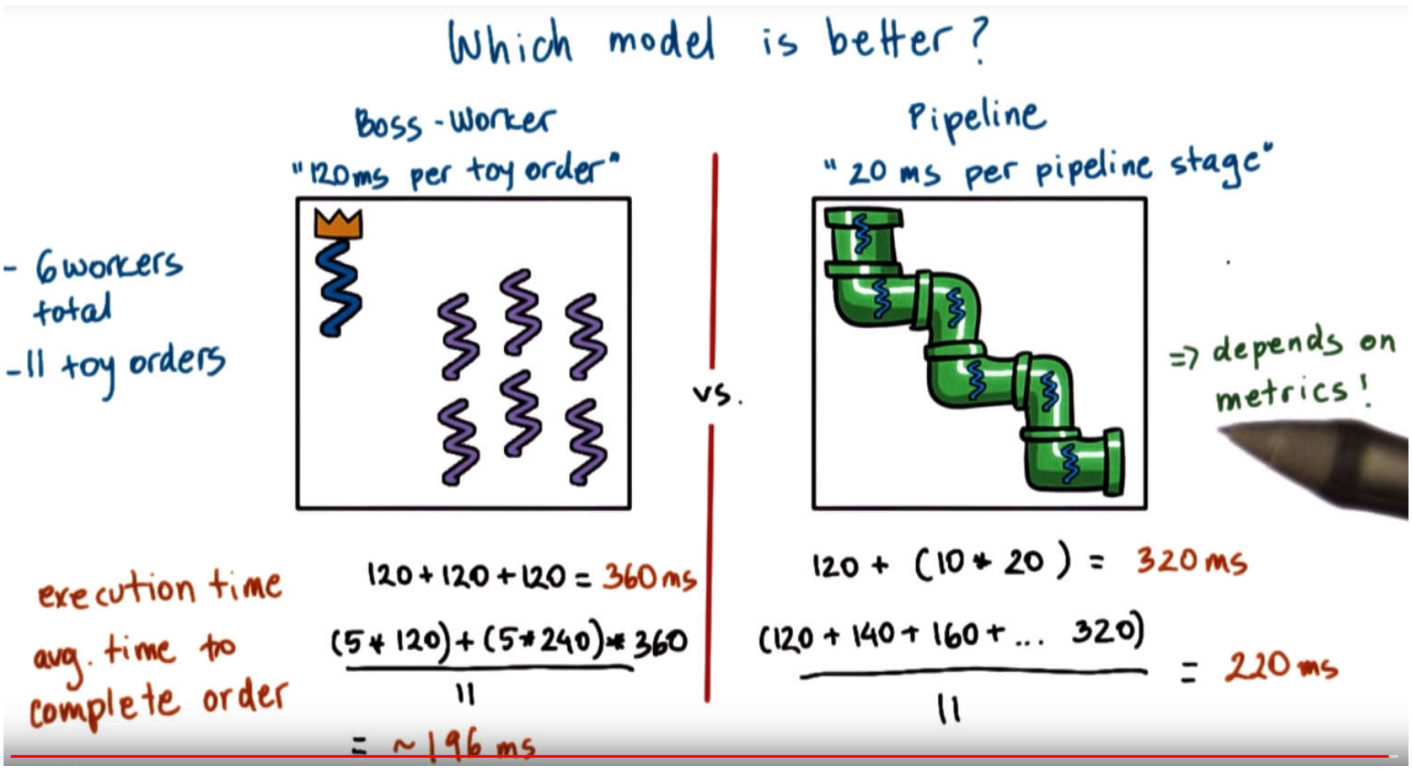
P2L5 Thread Performance Consideration

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?

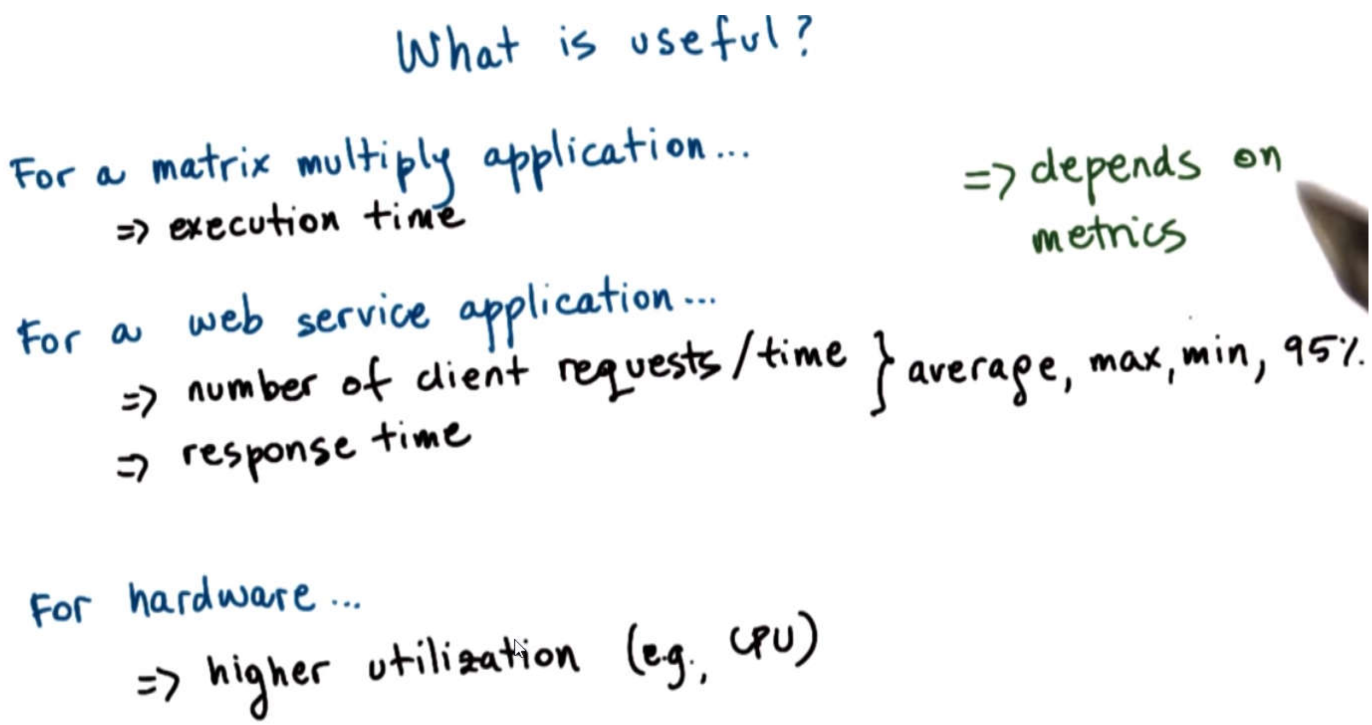


- For execution time(time it takes to process the same amount of job) pipeline model wins.
- For average time to complete order, Boss-Worker model wins.
- When making comparisons, what metrics we are considering if important.

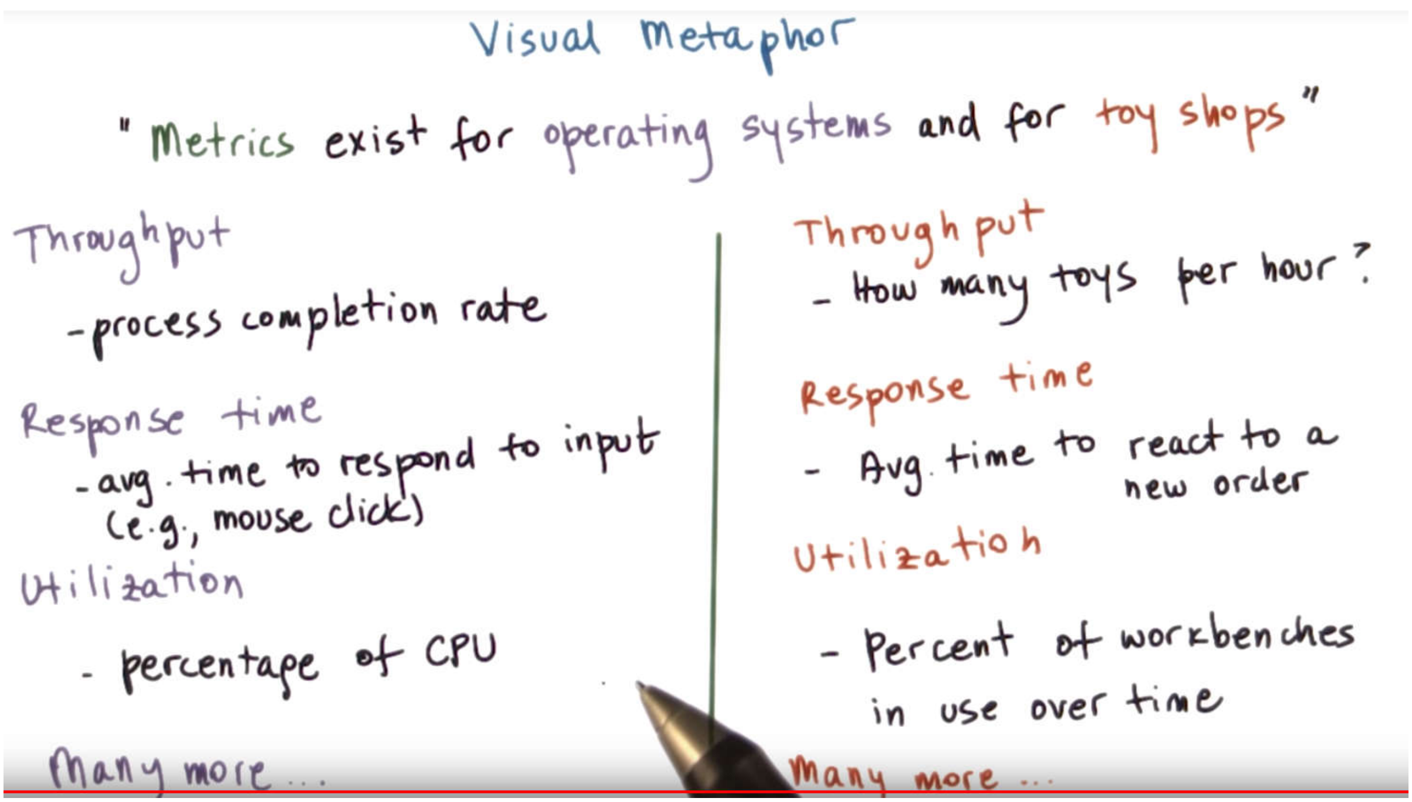
Are threads useful?

- 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

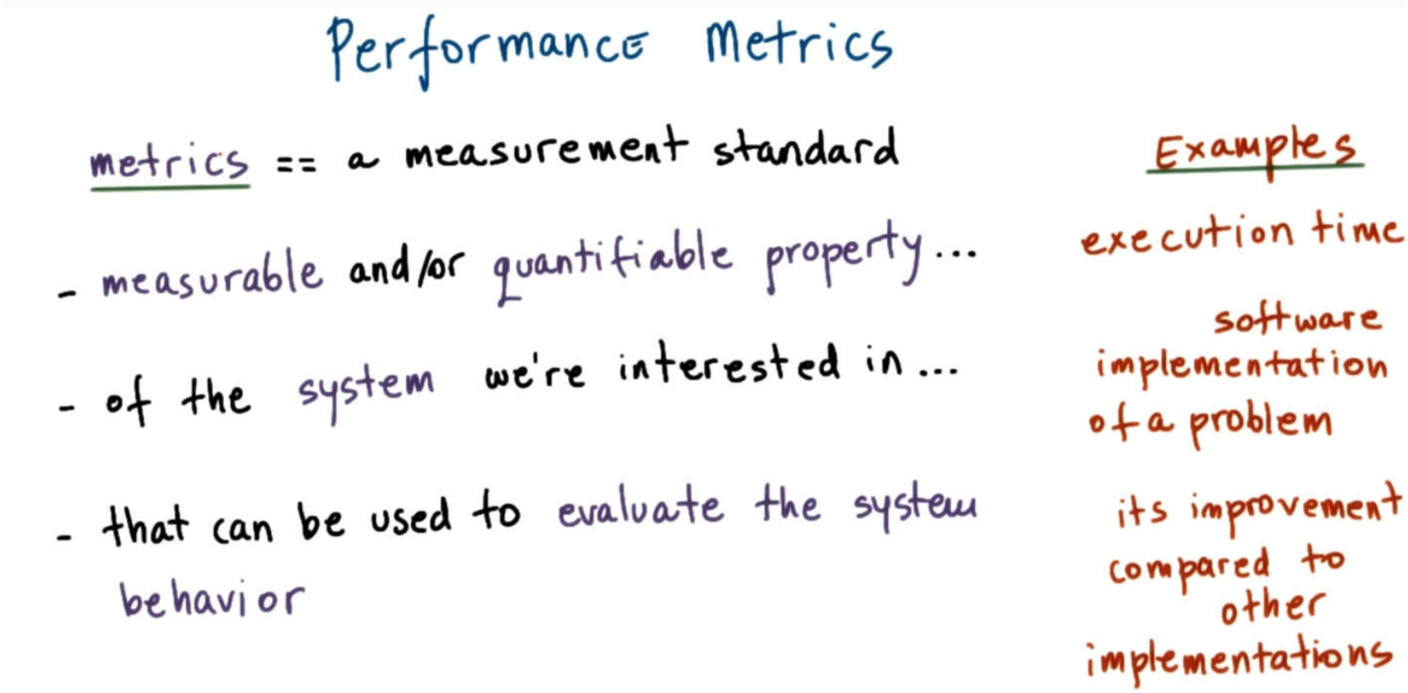
- Differen type of tasks (graphs, file patterns)

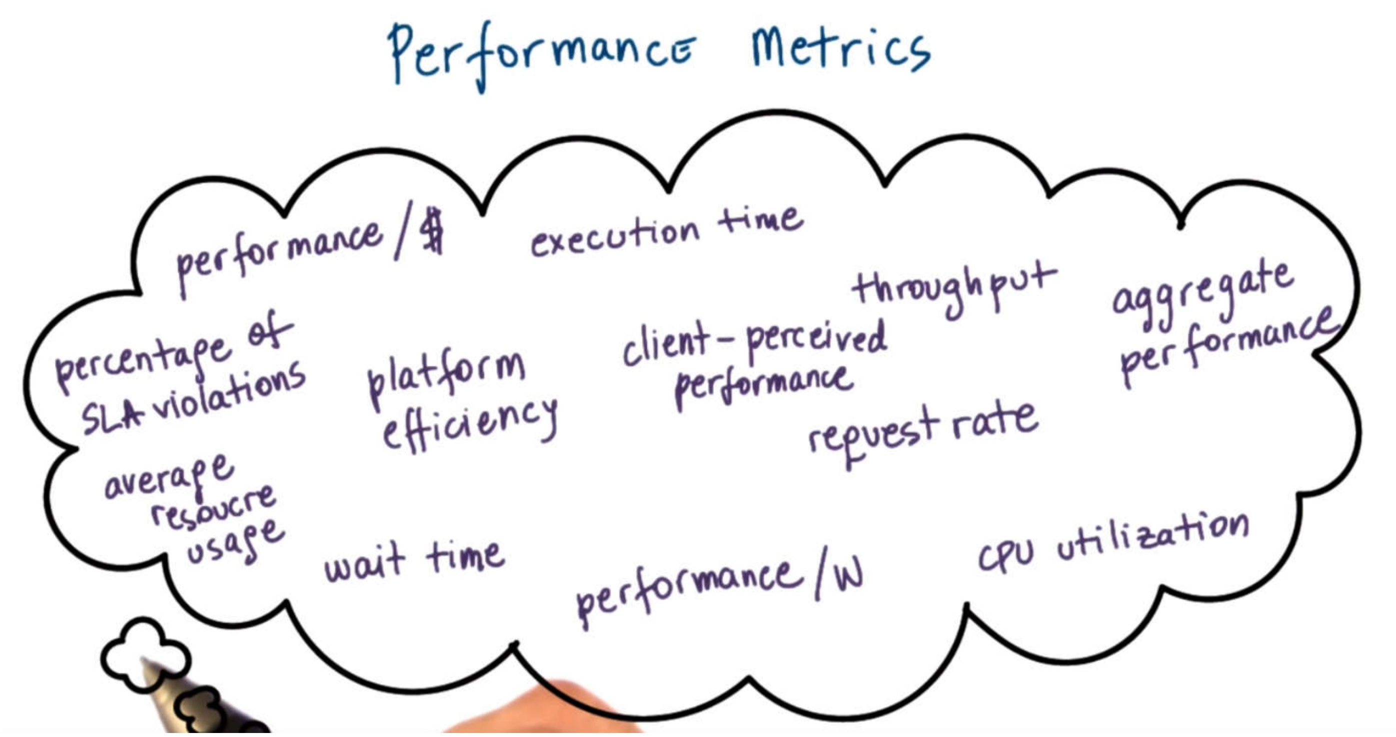


Visual Metaphore



Performance Metrics



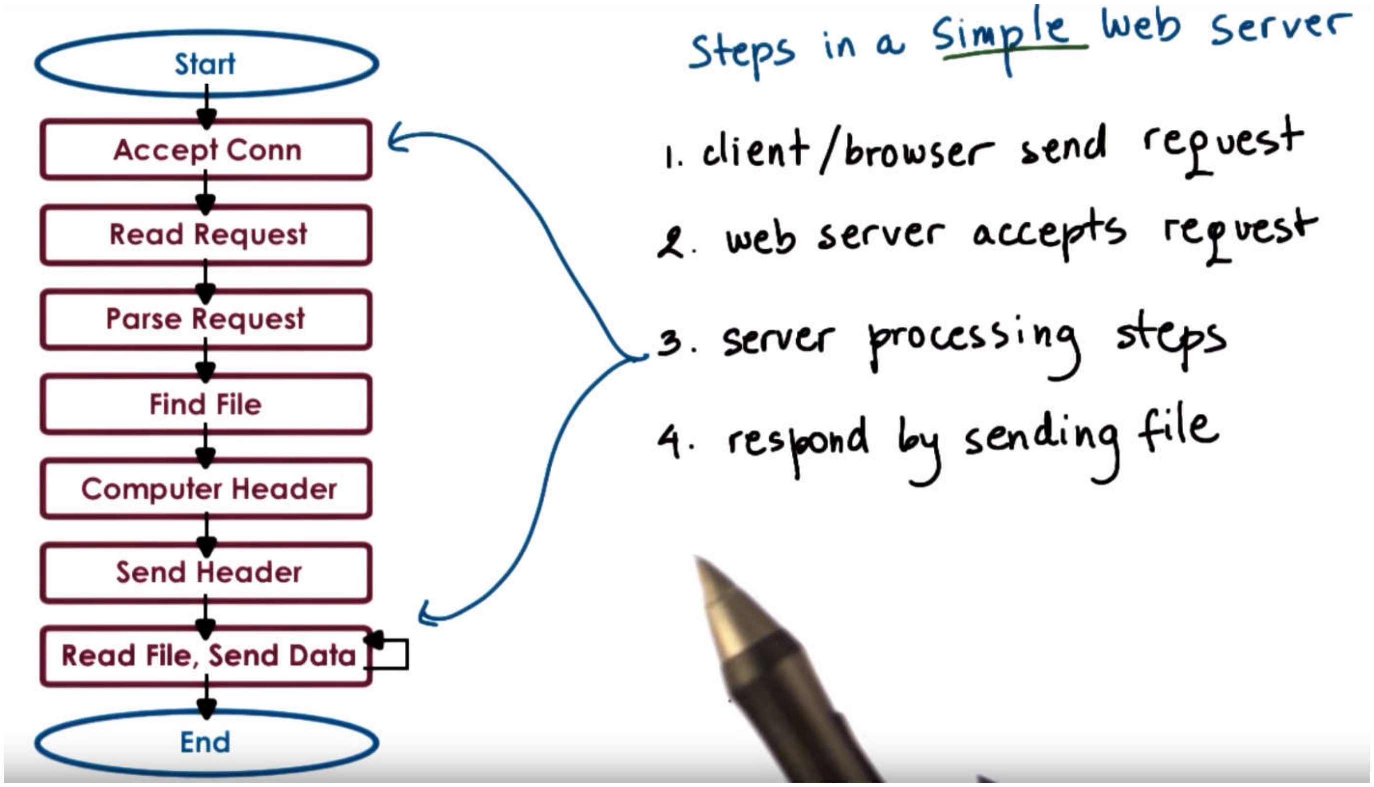


2. Multi-threading vs. Multi-Processing

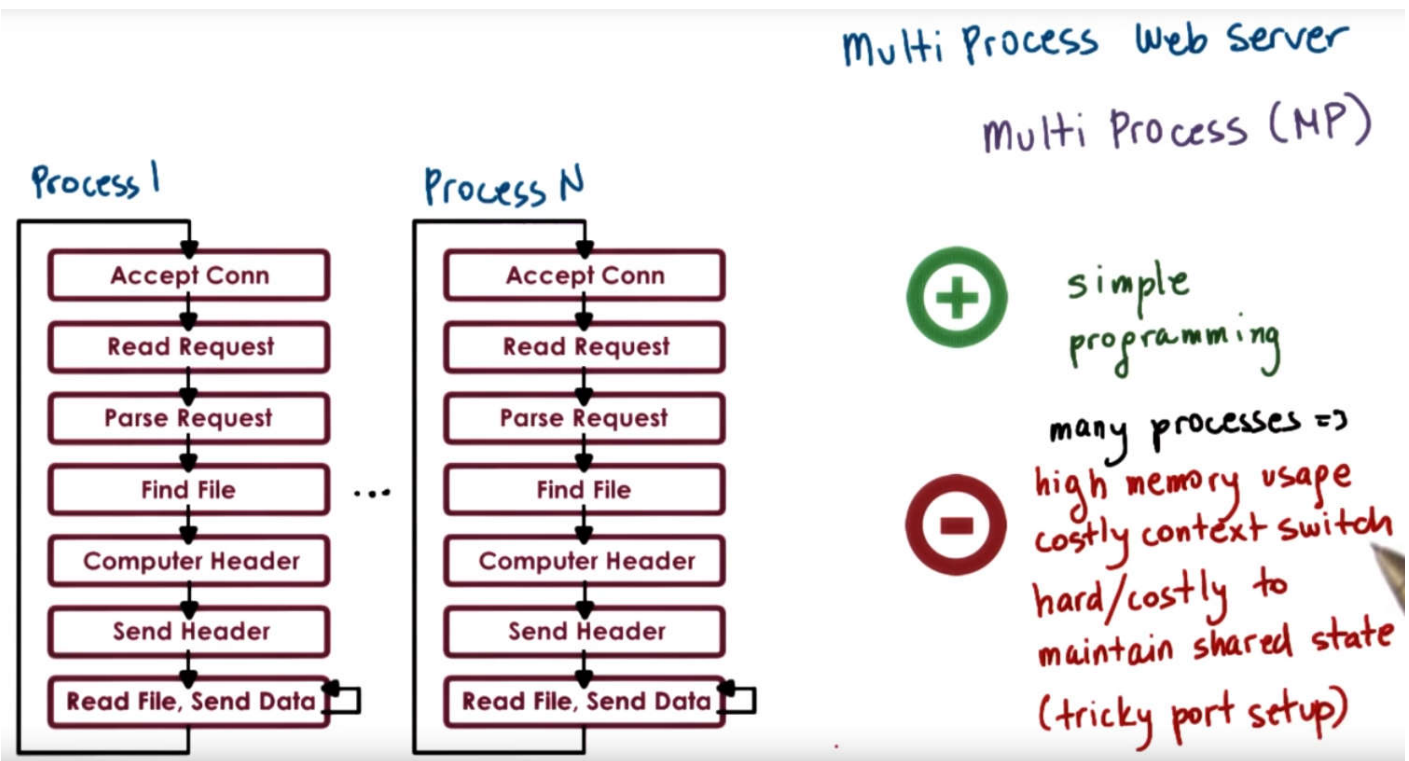
How to best provide concurrency?

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

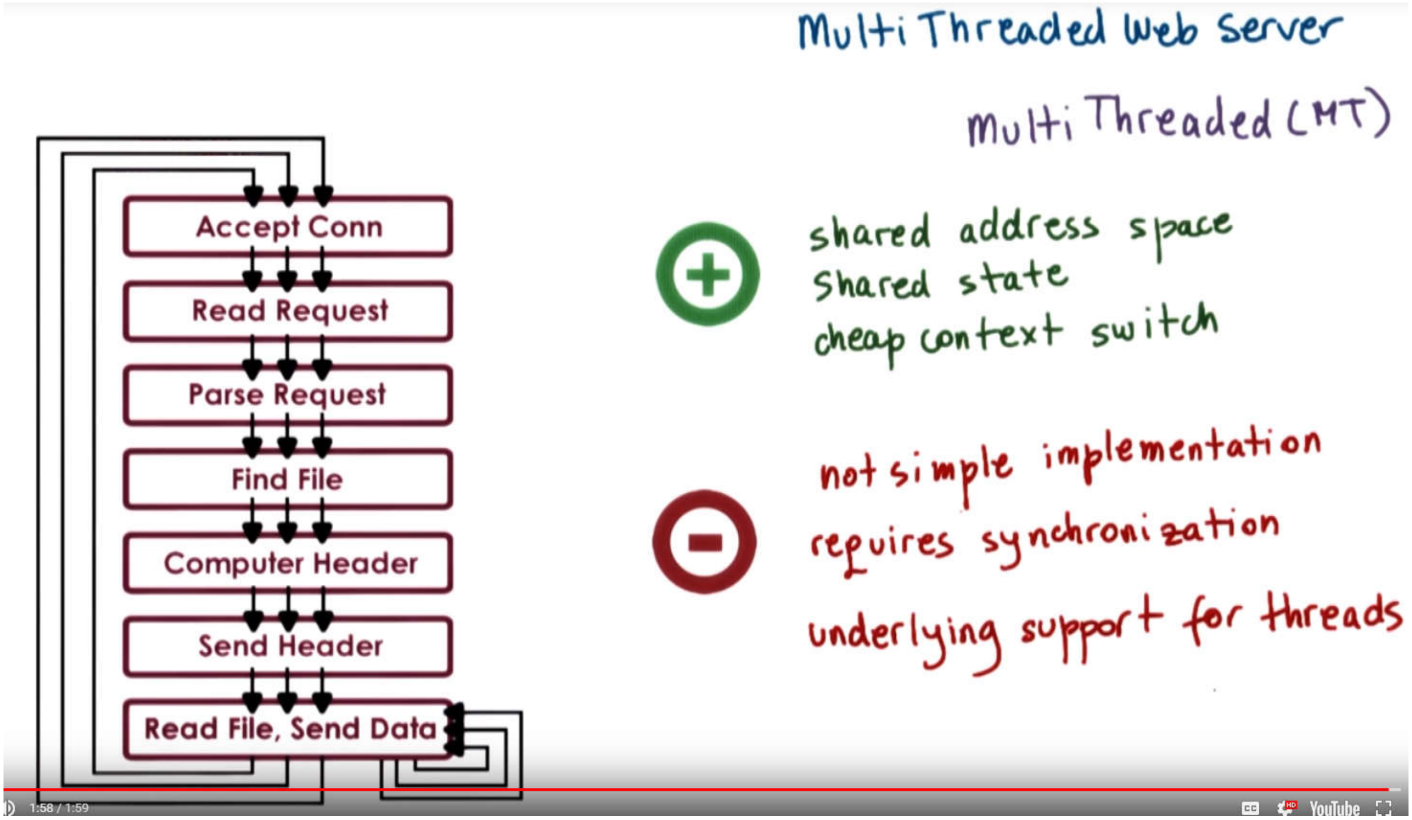
- Multiprocess vs. Multithreaded



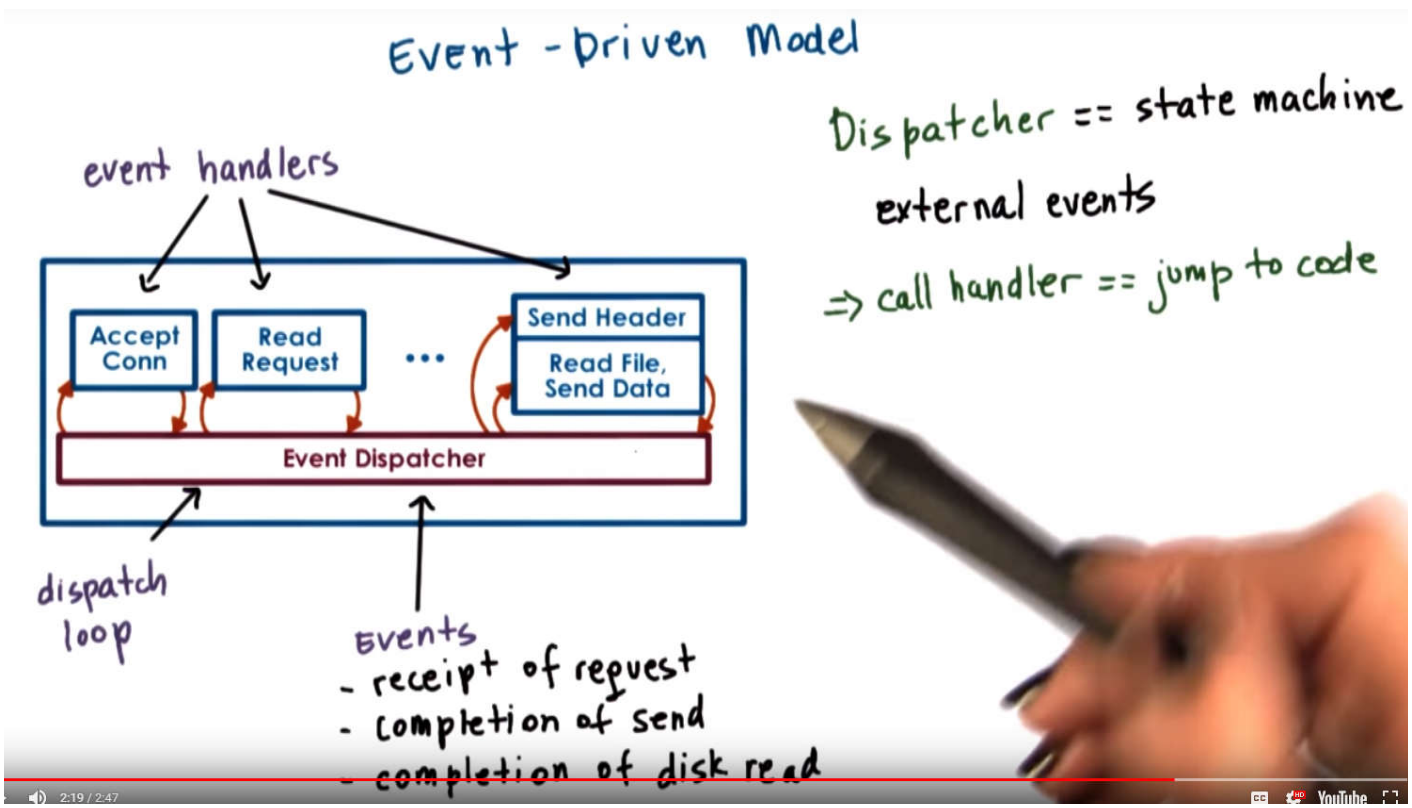
Multi-process Web Server

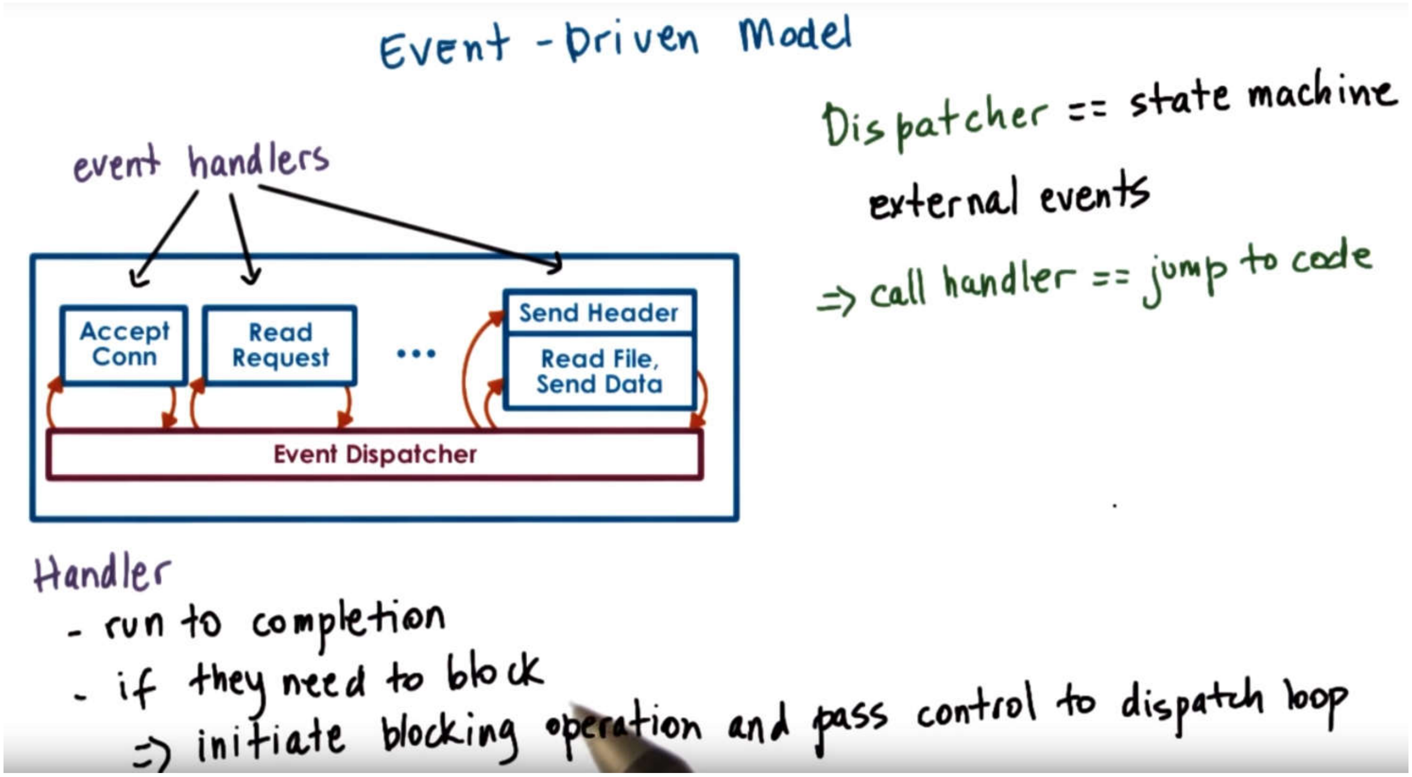


Multi-threaded Web Server

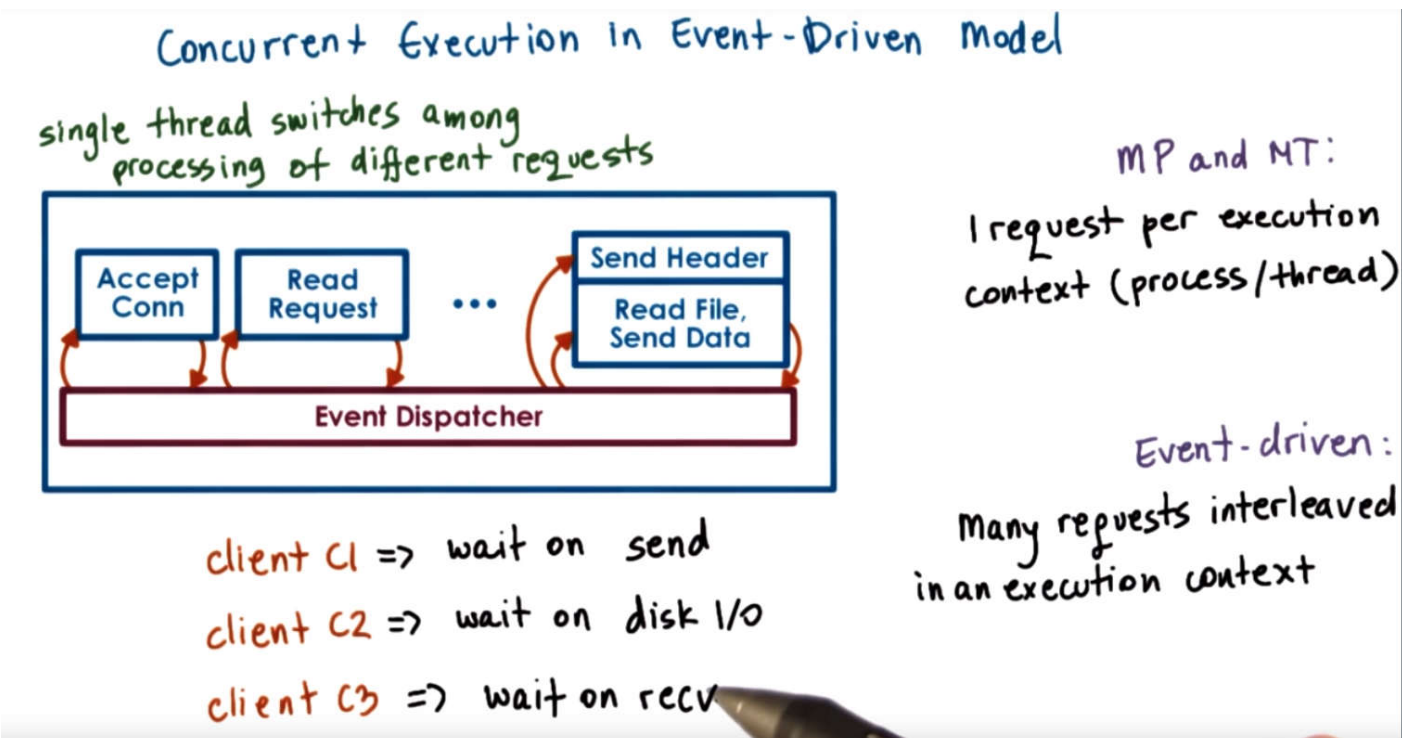


3. Event-Driven Model





Concurrency in Event-Driven Model

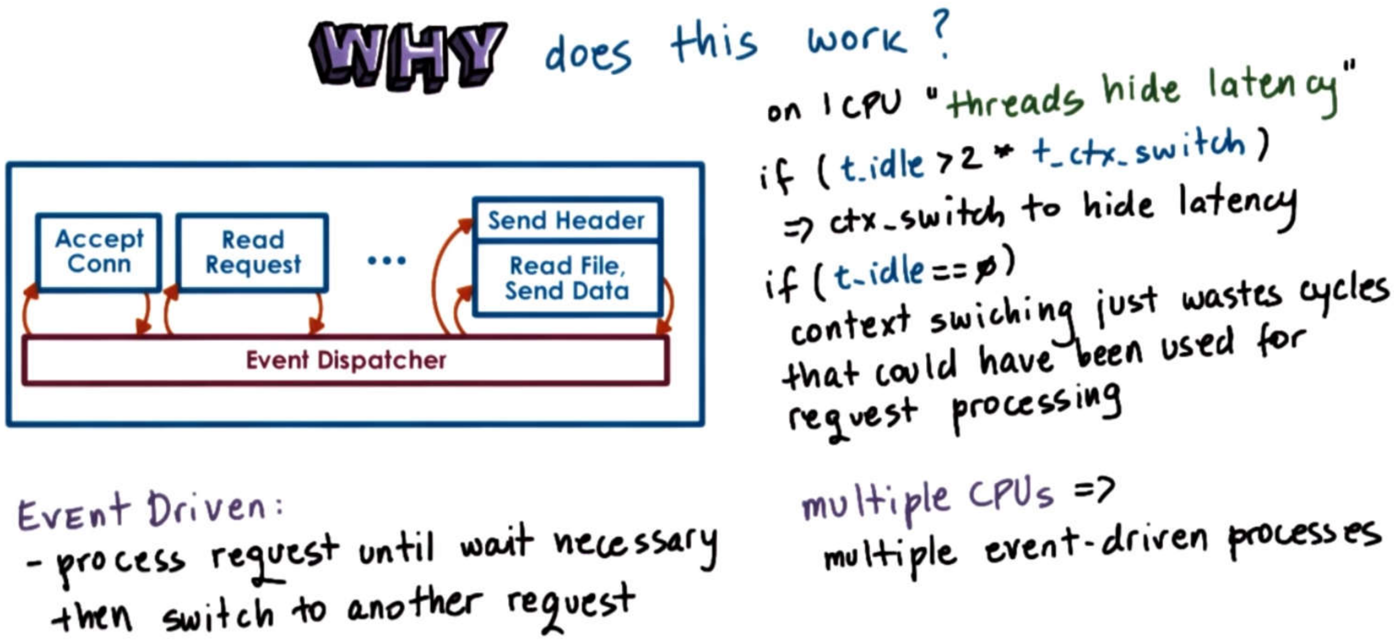


- Even though there is only one execution context/ single thread, there are concurrent executions of multiple requests that are handled interleave.

Why does this work?

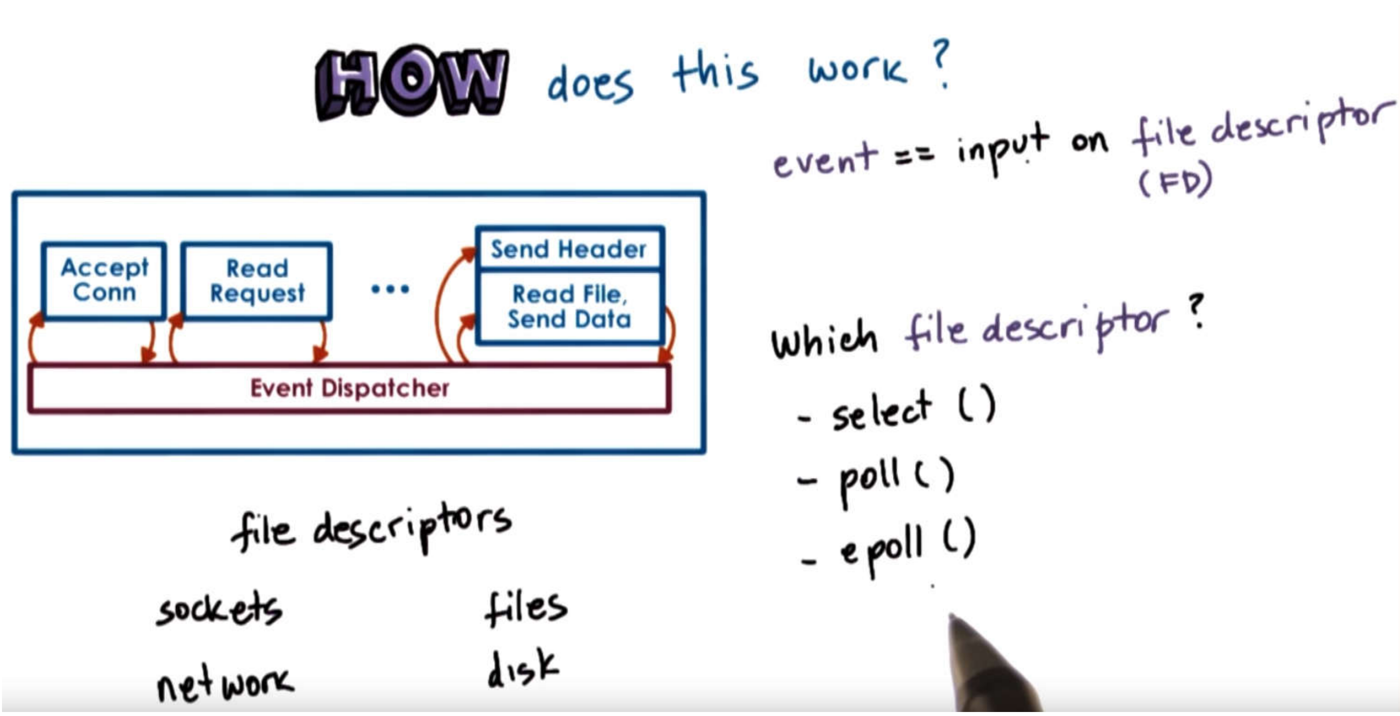
- What is the benefit of having one thread switching between processing of different requests.

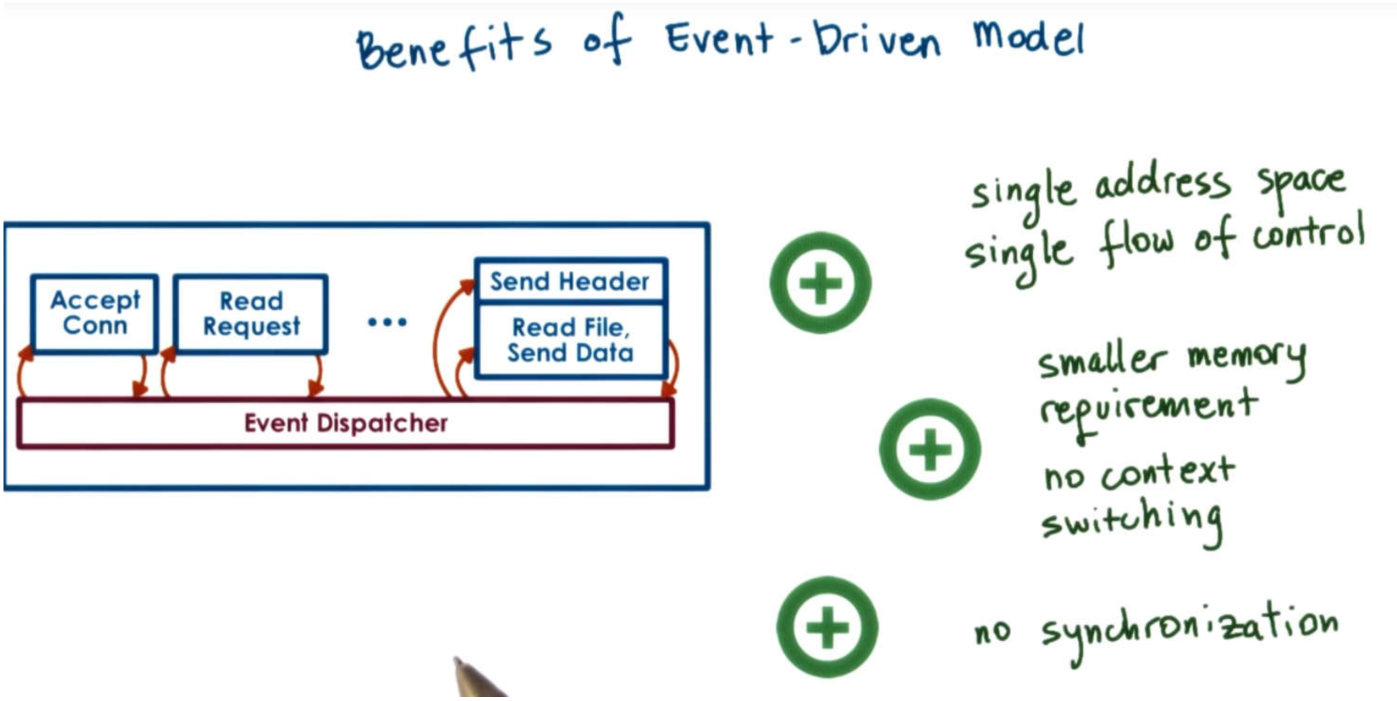
- How does this approach compare to assigning to different threads or even to different processes.



- In single threaded event-driven model, it'll process the requests until idle is necessary then switch to another request, hence avoid unecessary 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).

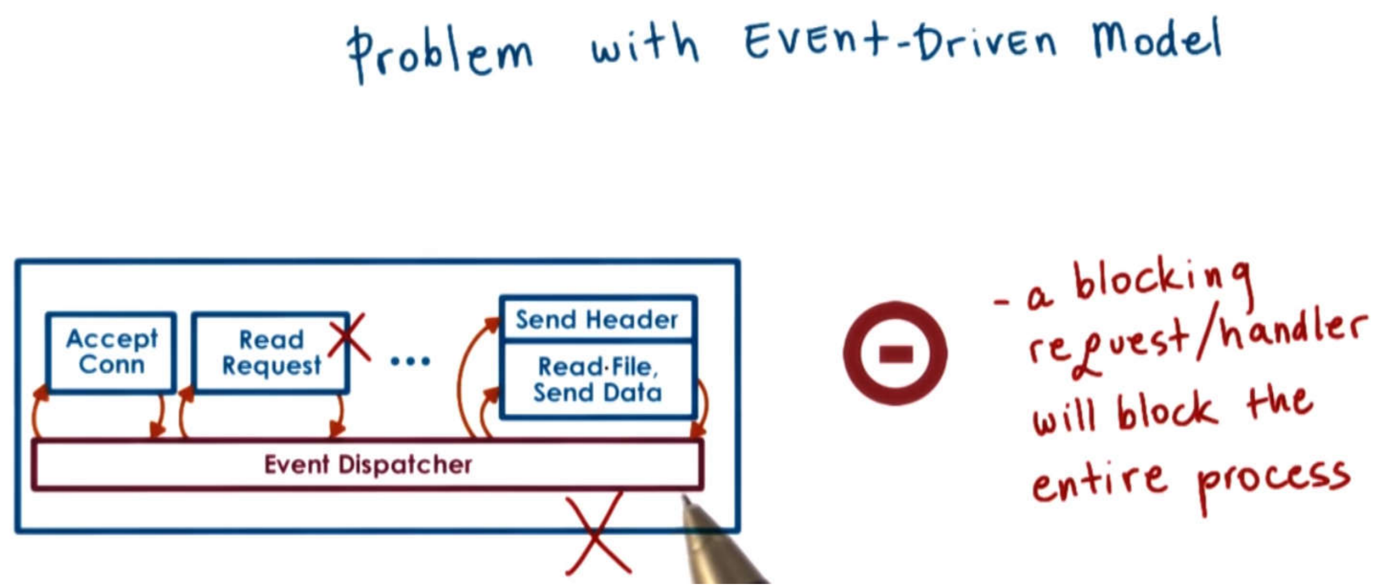
How to implement Event-Driven Model?



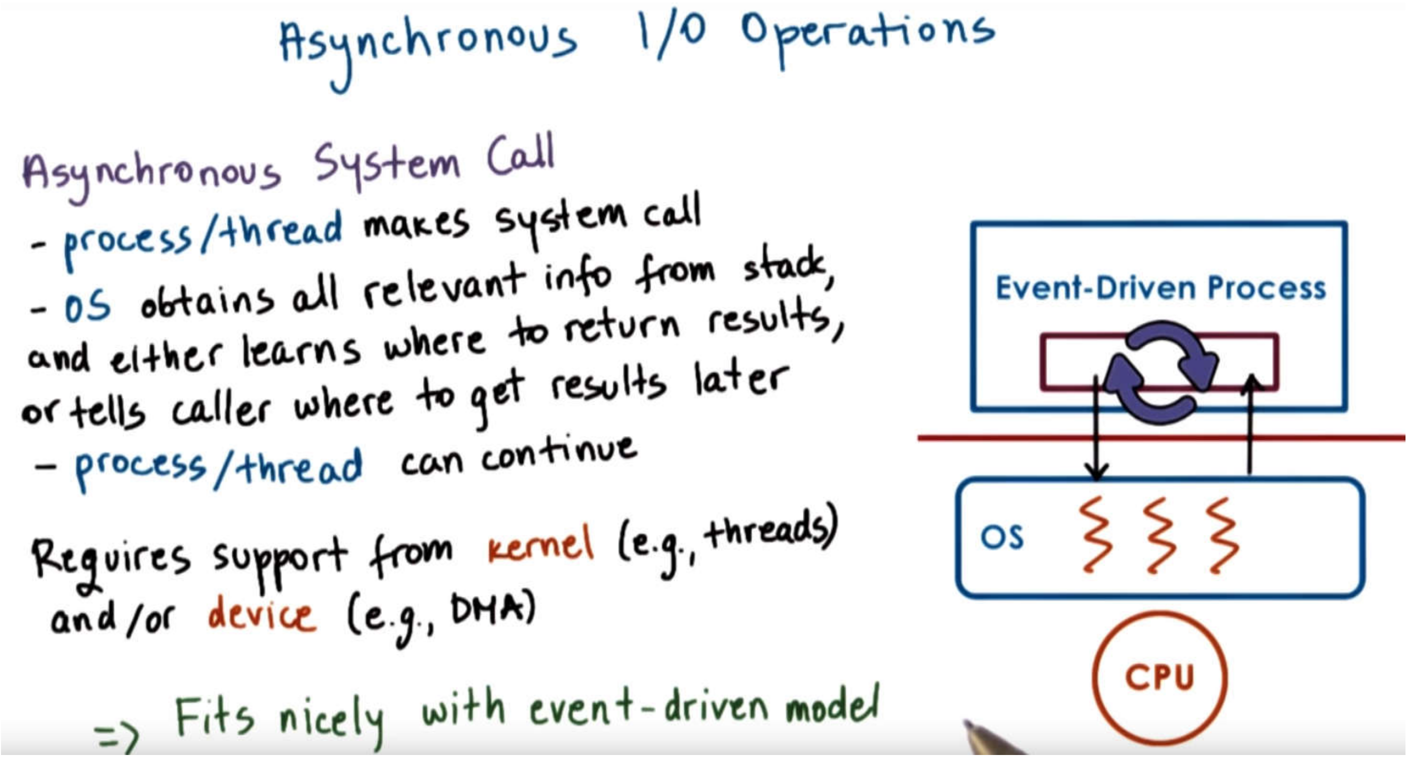


- Jumping over the code base of the server

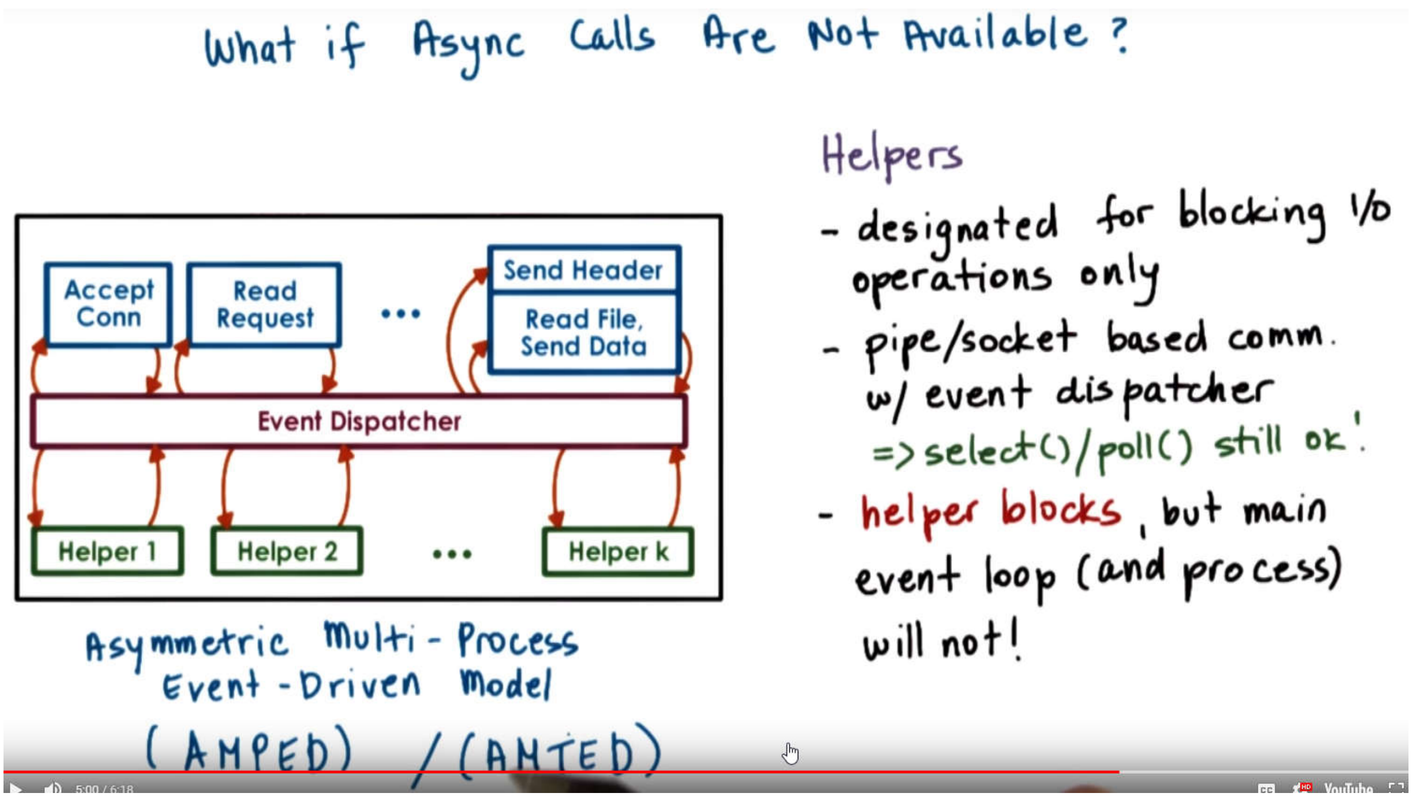
Problem with Event-Driven Model



Solution:



- Only possible because the OS kernel itself is mult-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

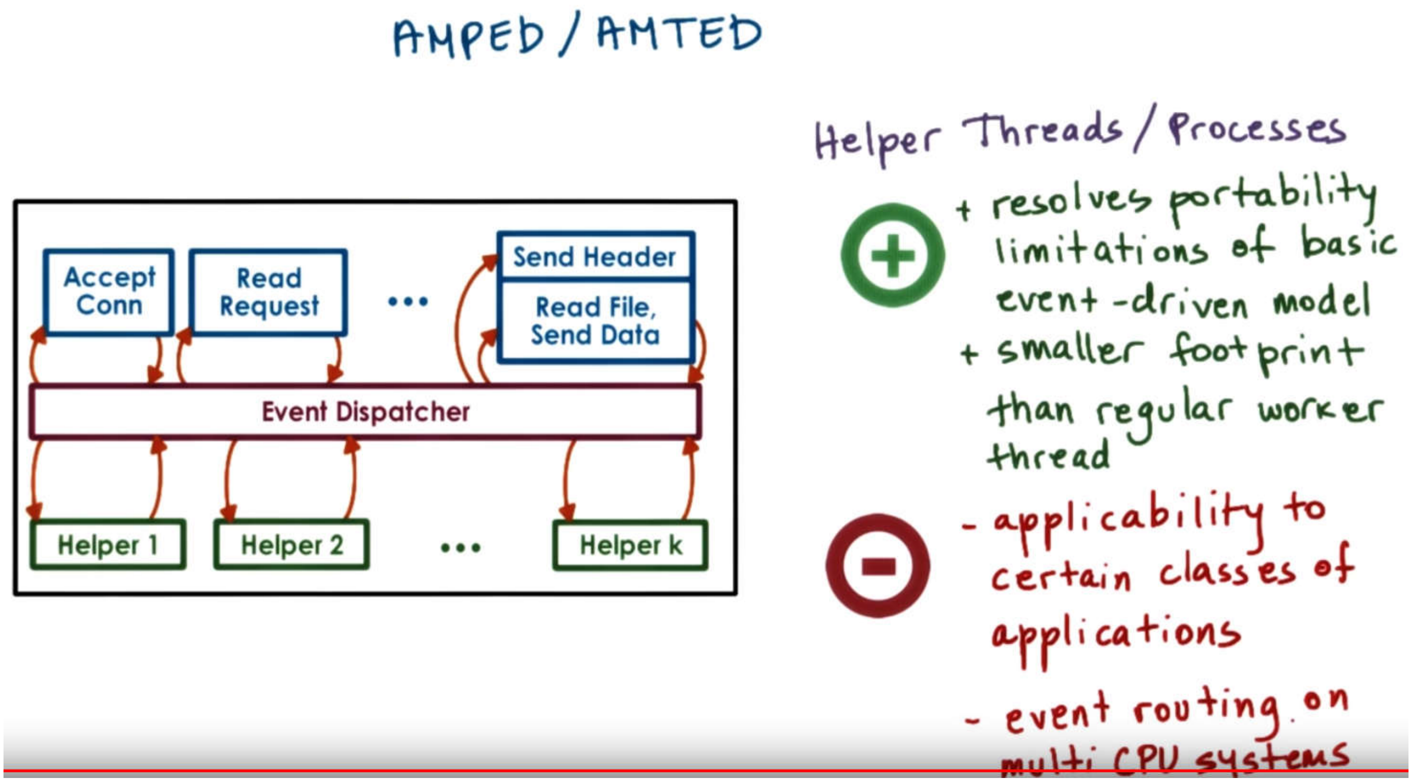


Asymmetric Model:

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

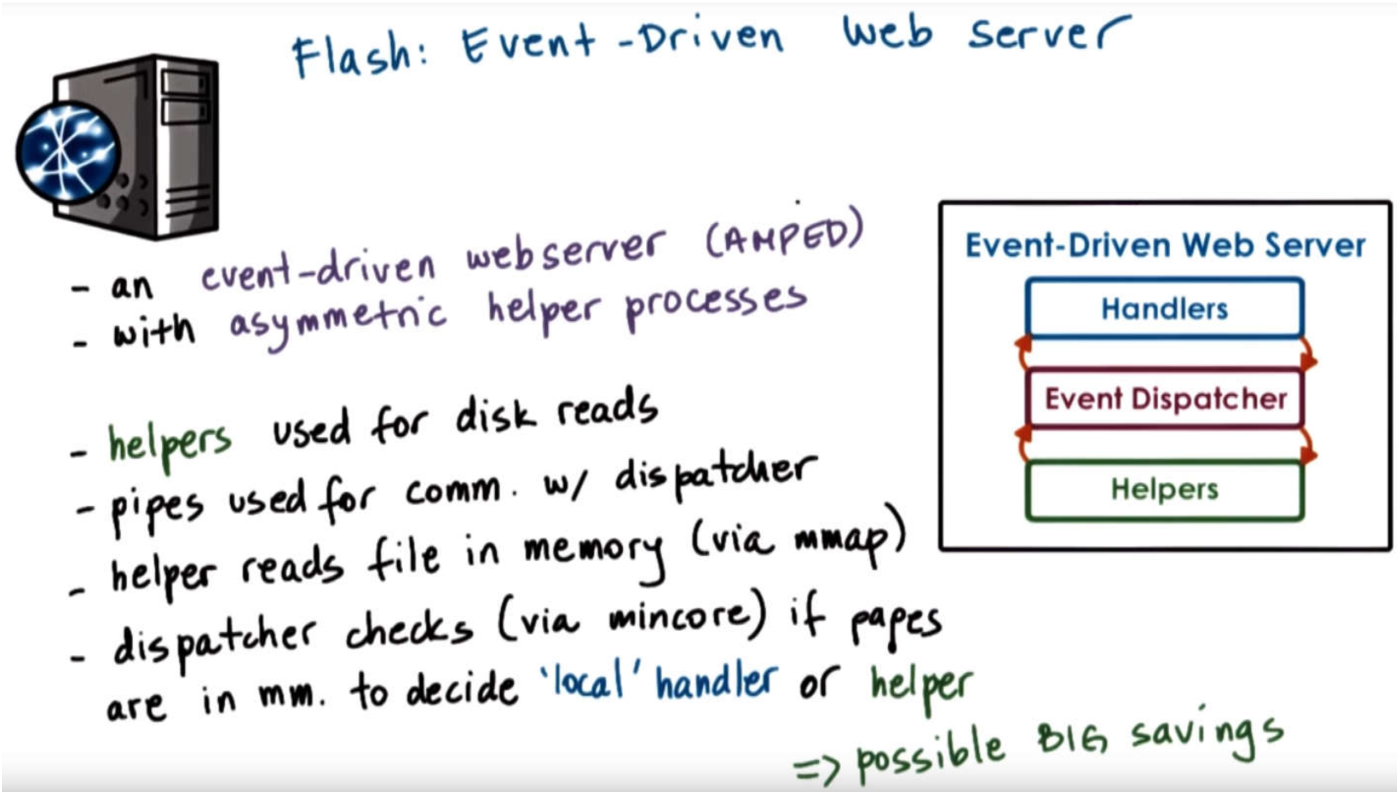
- Asymmetric: The helper side only deals with blocking I/O operations and the other side (upper side) deals with everything else.

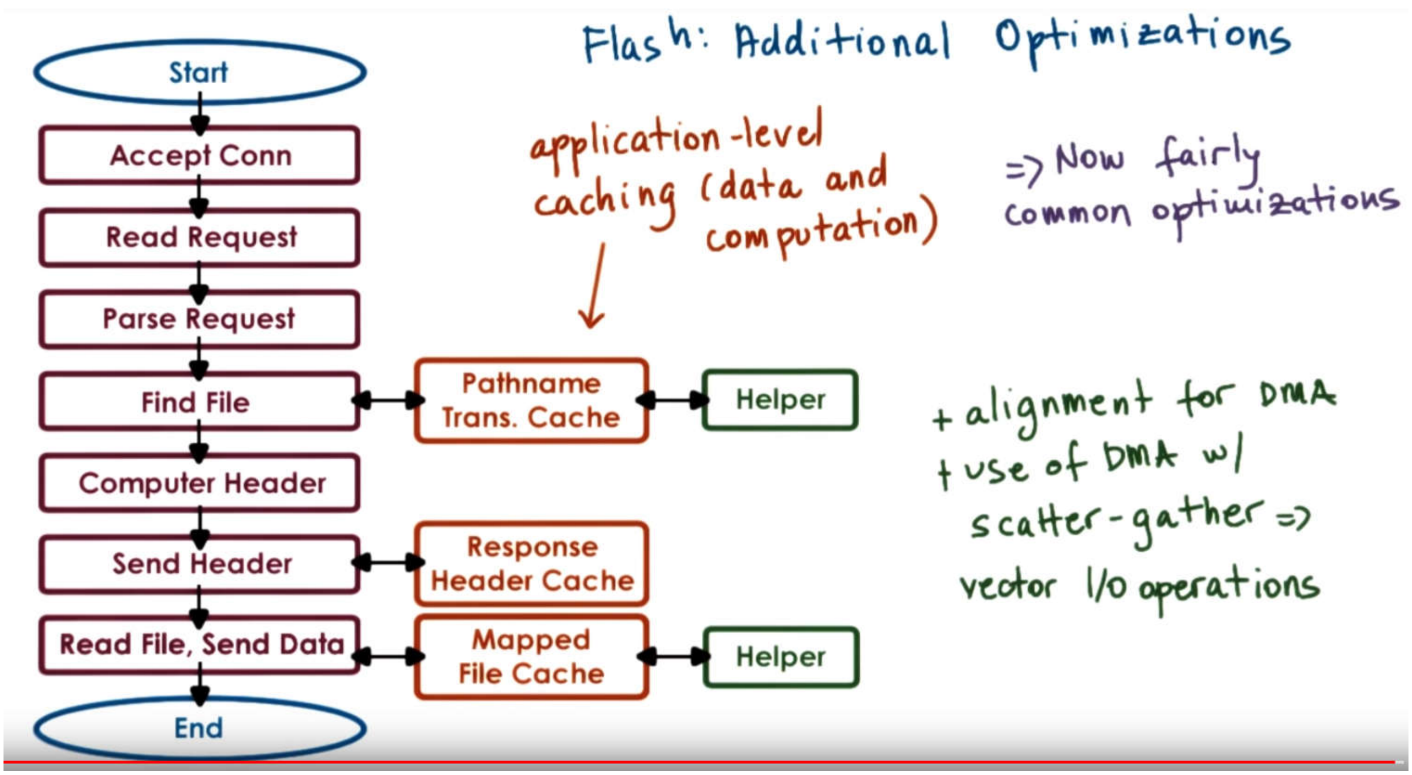
Asymmetric Model Benefits & Downsides:



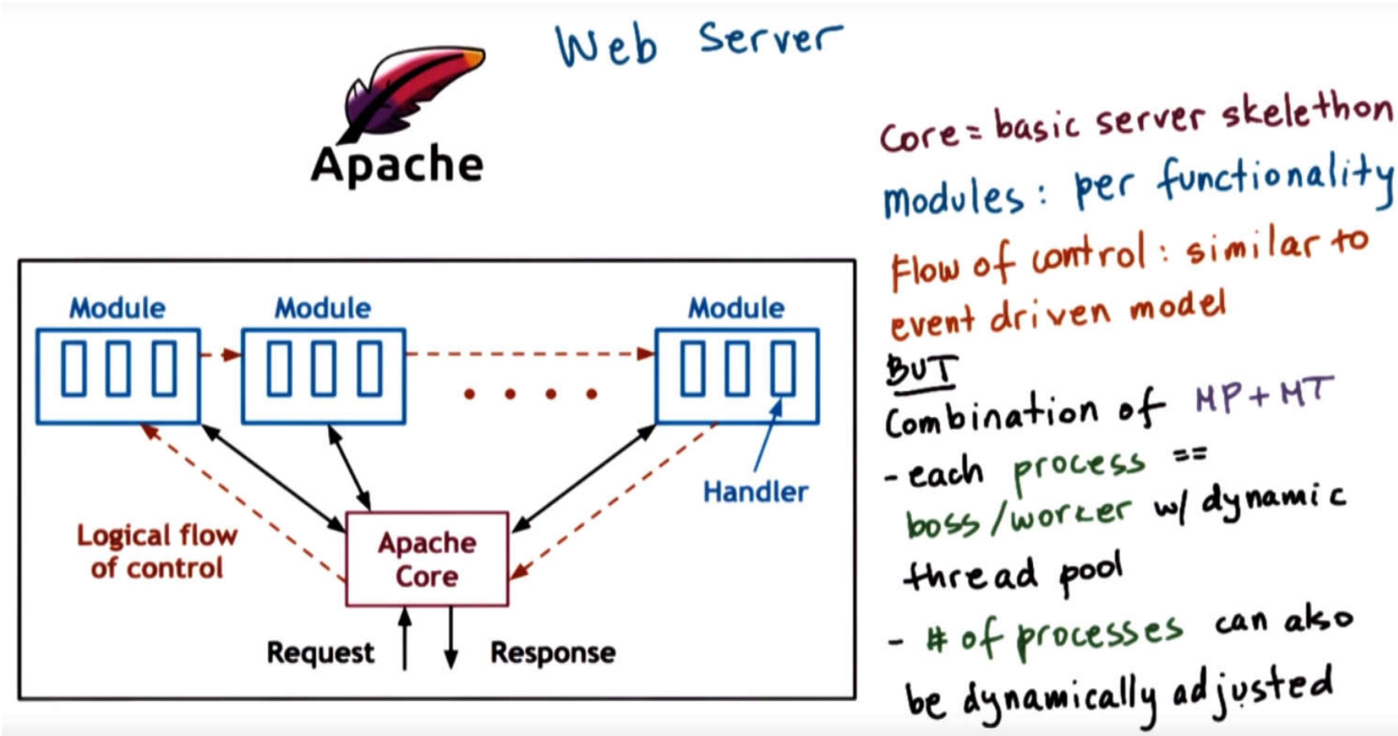
4. Flash & Apache Web Server

Flash Web Server

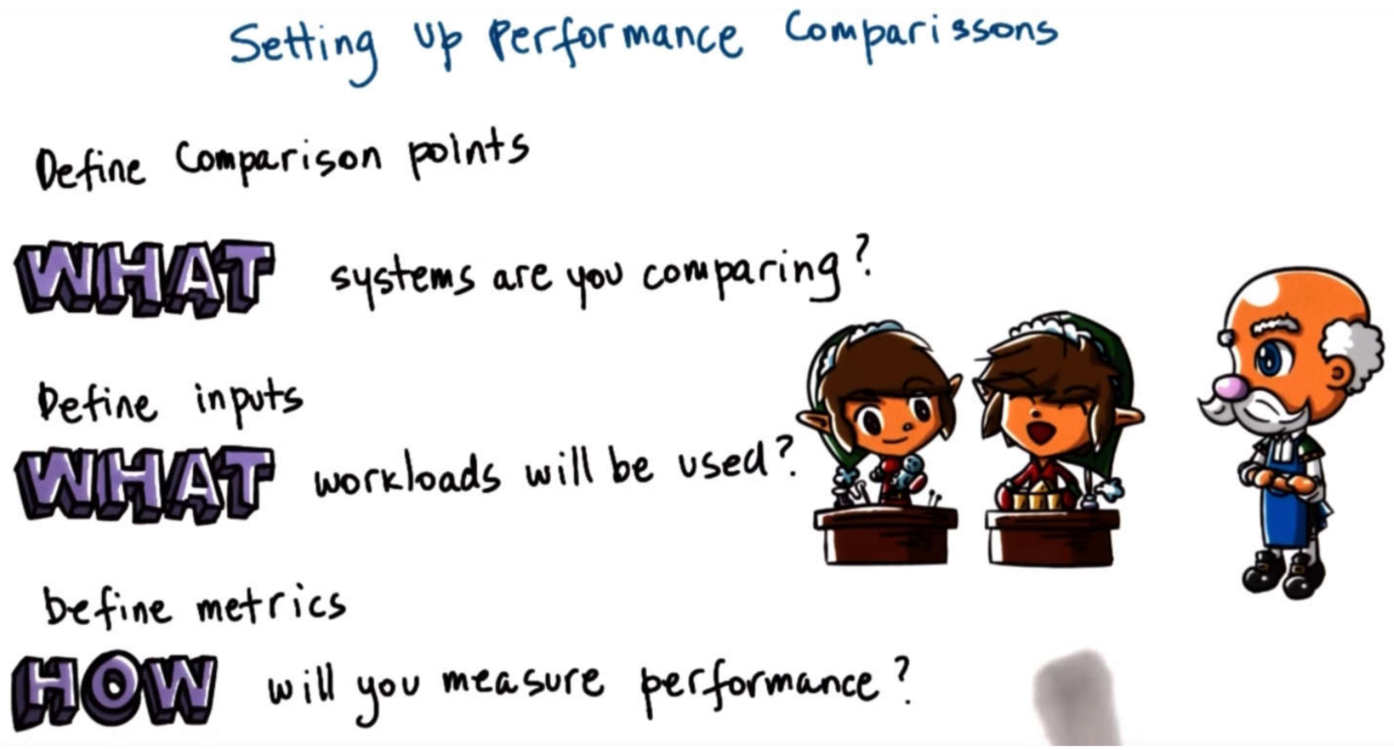




Apache Web Server



5. Experimental Methodology w Flash Paper as Example



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?

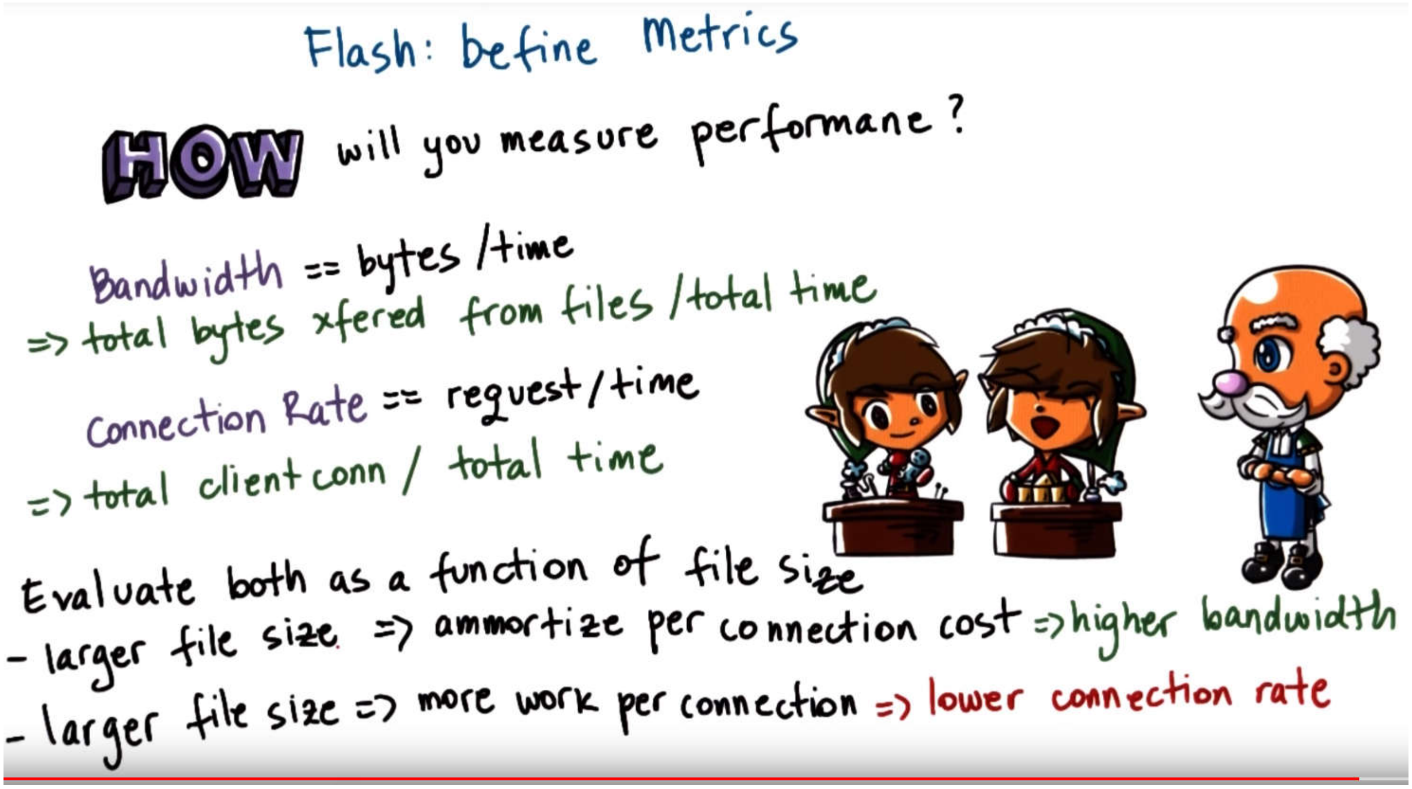
- CS Web Server trace (Rice Univ.)
- Owlnet trace (Rice Univ.)
- synthetic workload

Realistic request workload

=> distribution of web page accesses over time

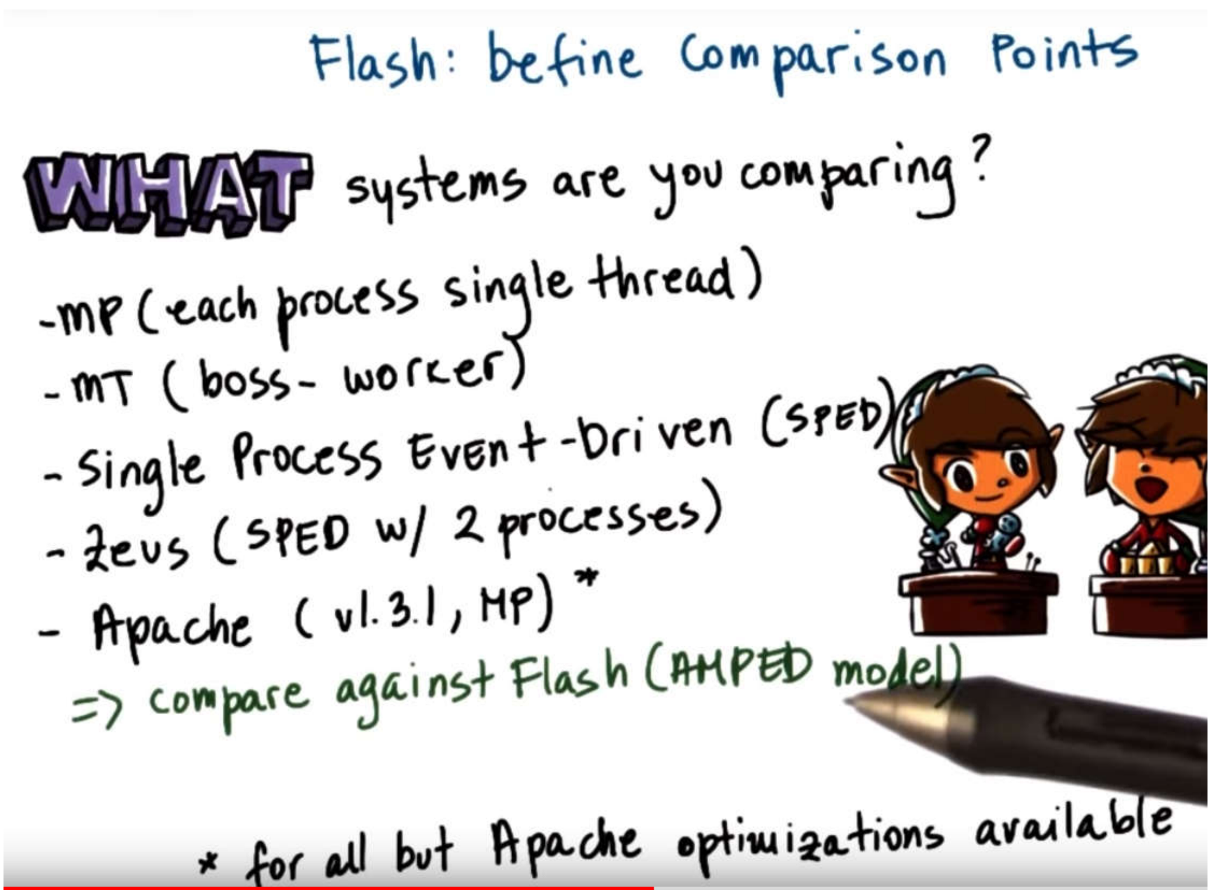
Controlled, reproducible workload

=> trace-based (from real web servers)

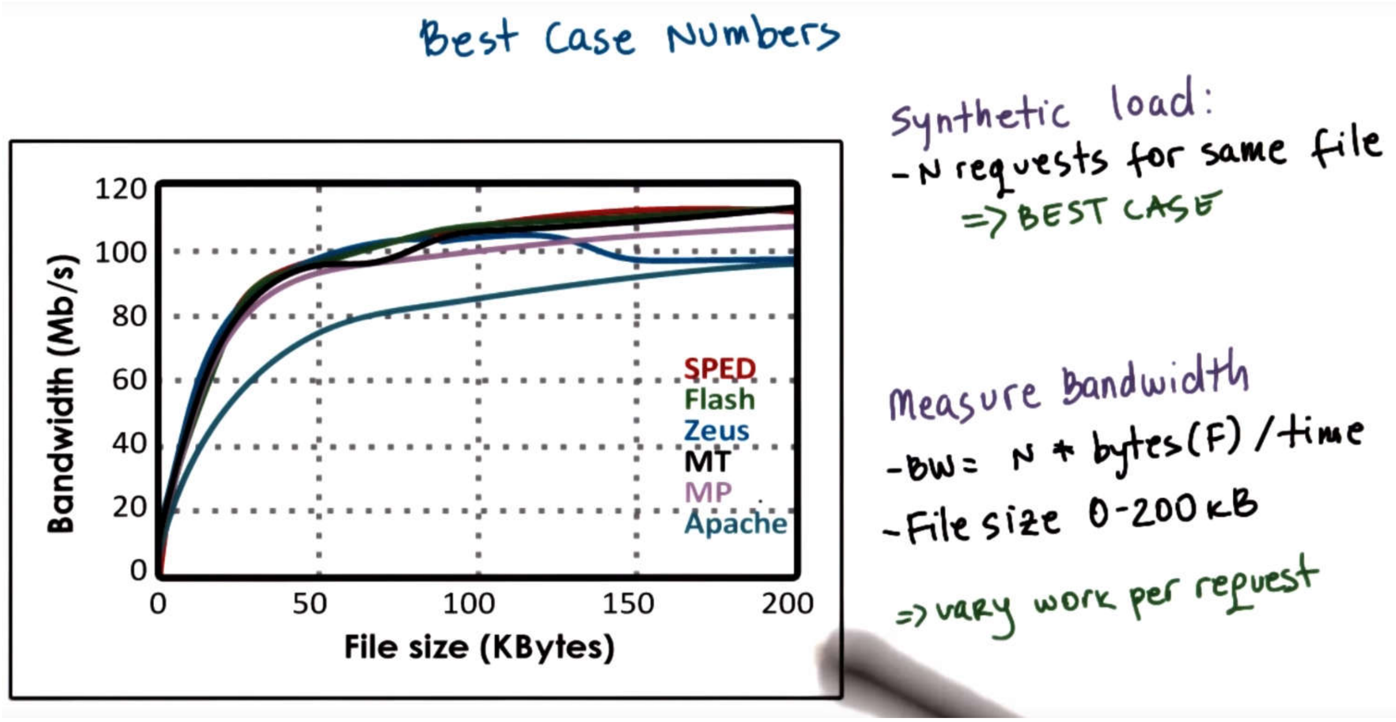


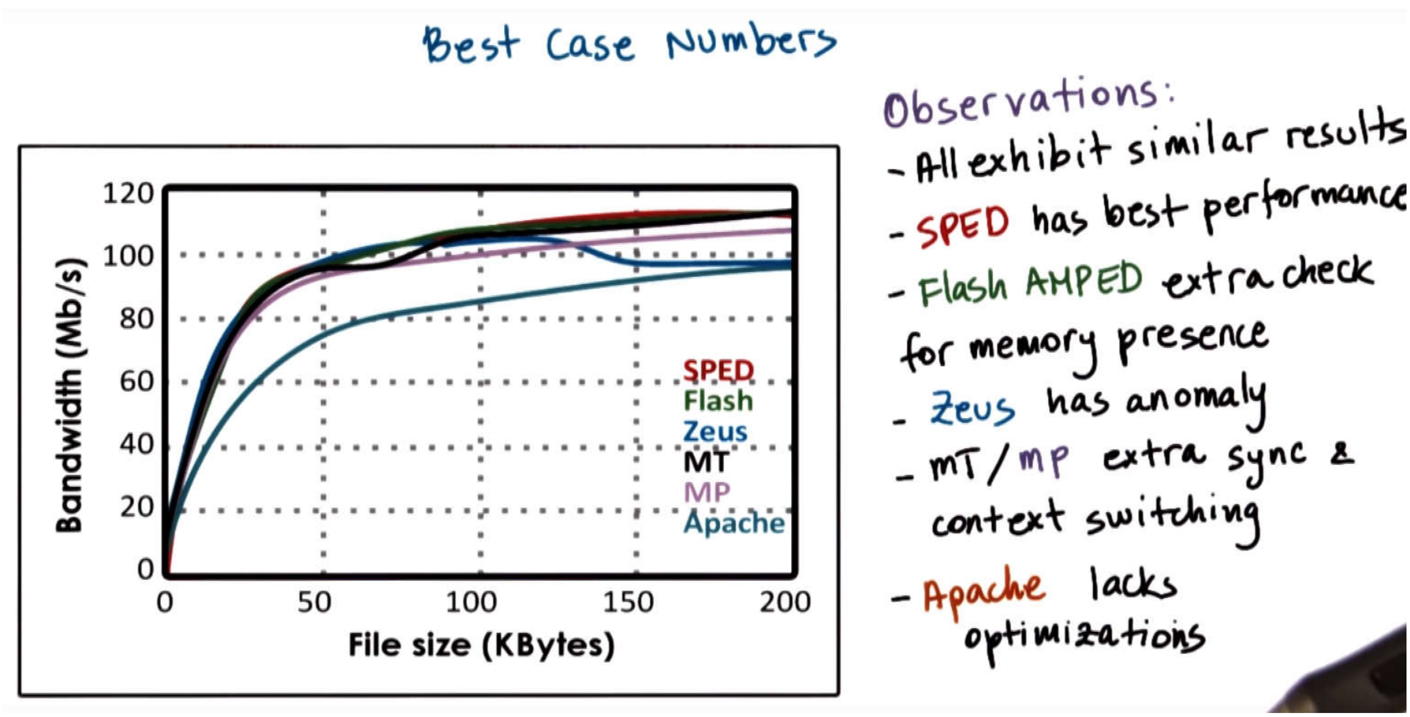
Experiment Results

Cases:

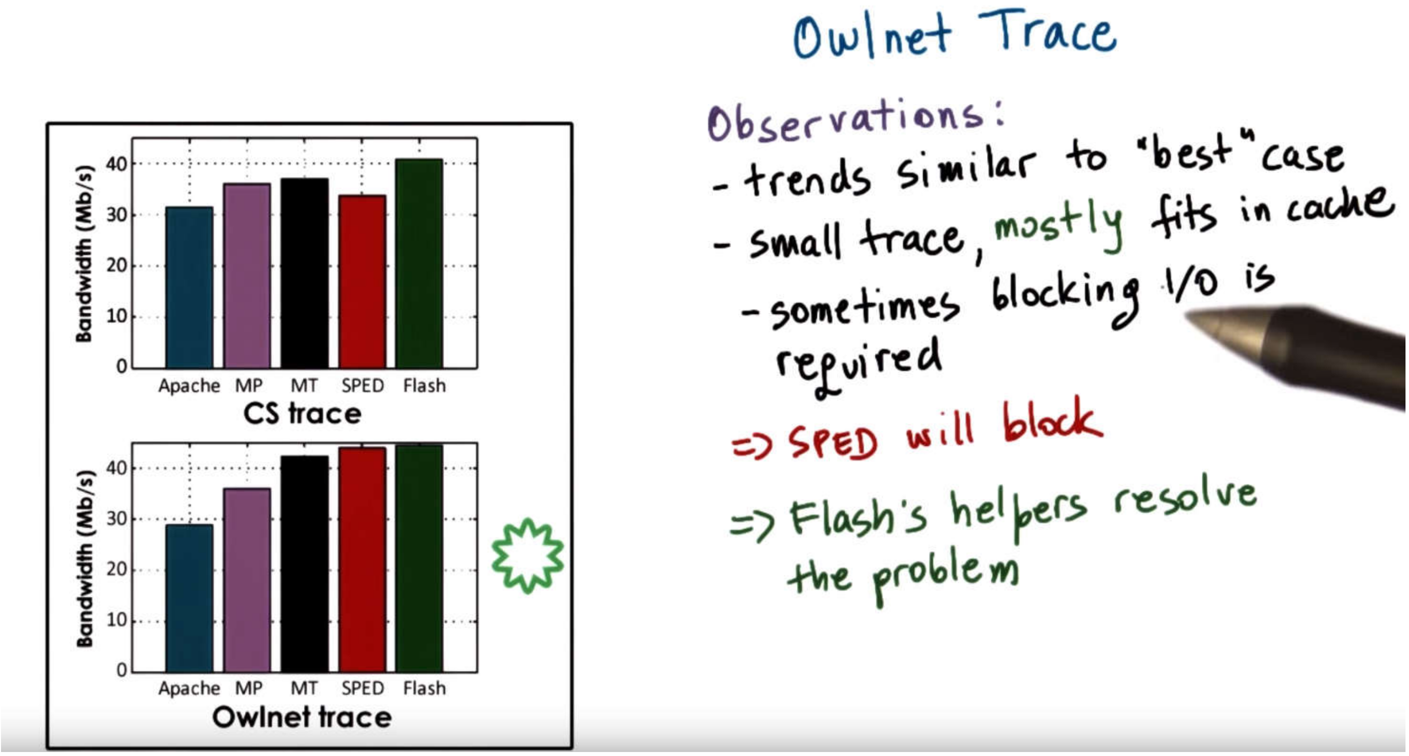


Best Case:

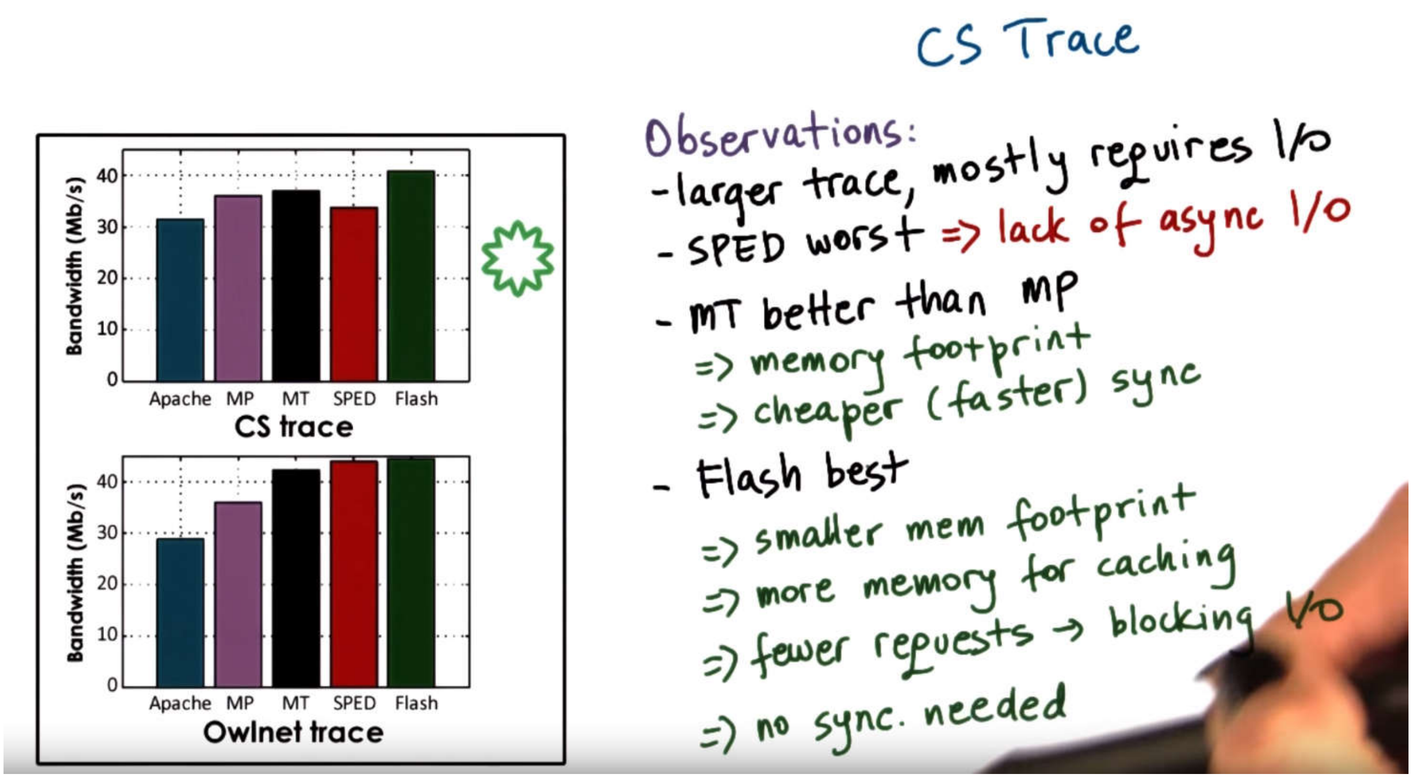




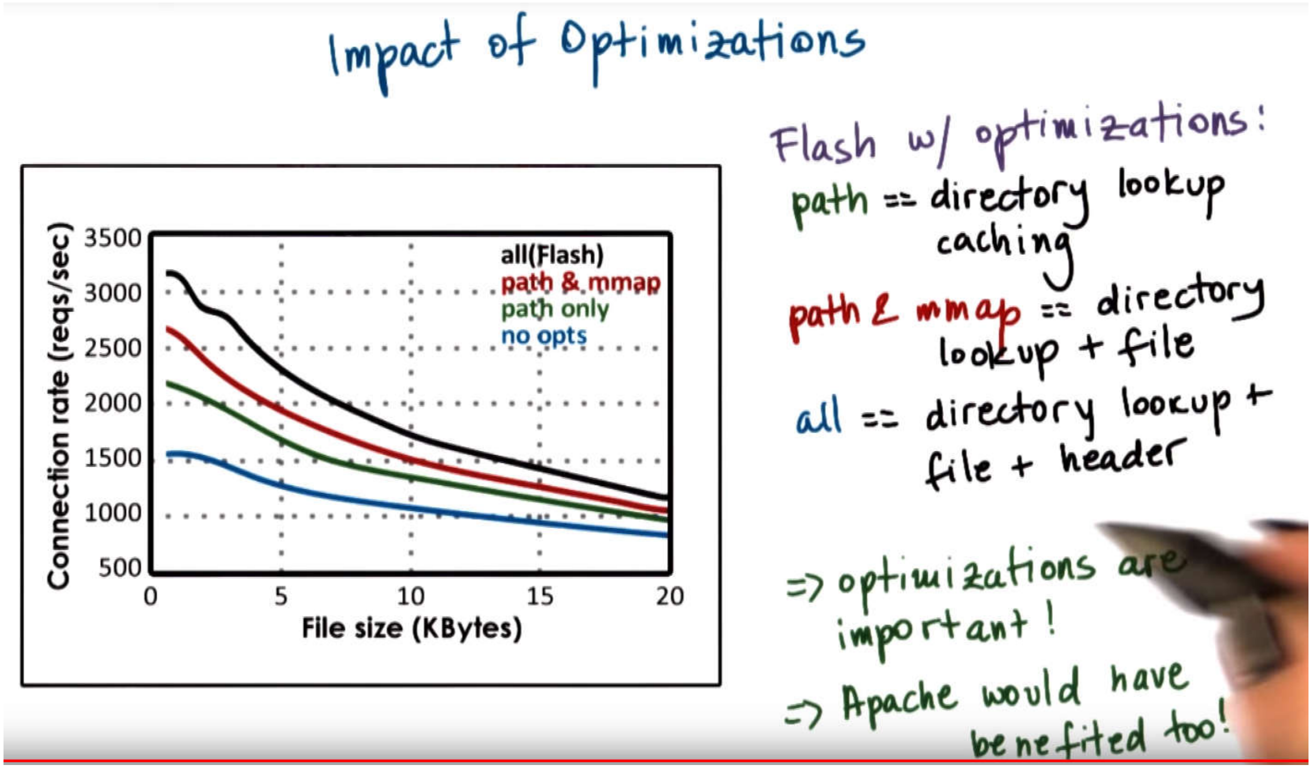
Owlnet Trace:



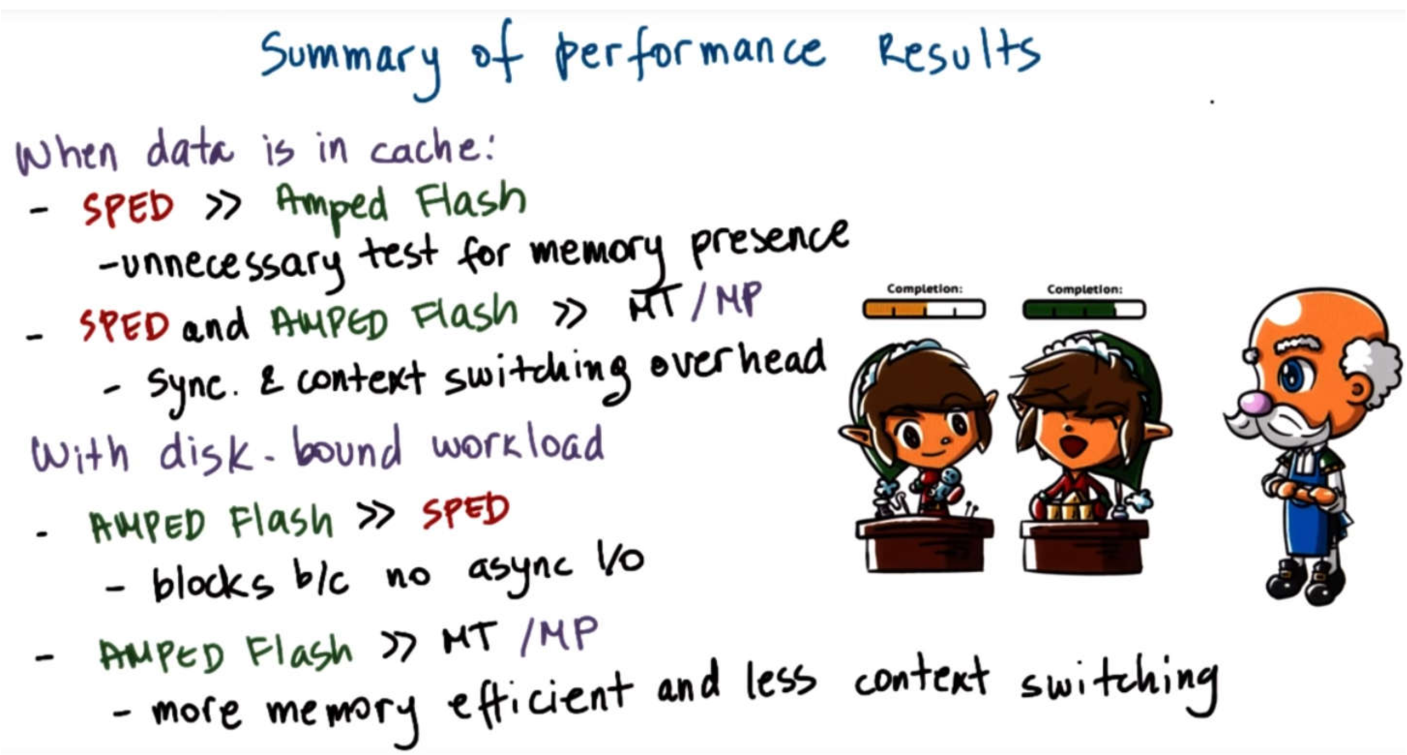
CS Trace:



Impact of Optimizations performed in Flash



Summary:



Advices on Designing Experiments

Design Relevant Experiments

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

Not so fast!!!

Relevant experiments => 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 => great!
- +response time => will buy that too.
- +response time, -throughput => may be useful?
- maintains response time when request rate increases

goals => 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, # timedout repuest...
  - operator costs -> throughput, costs

Picking the Right Configuration Space

System Resources:

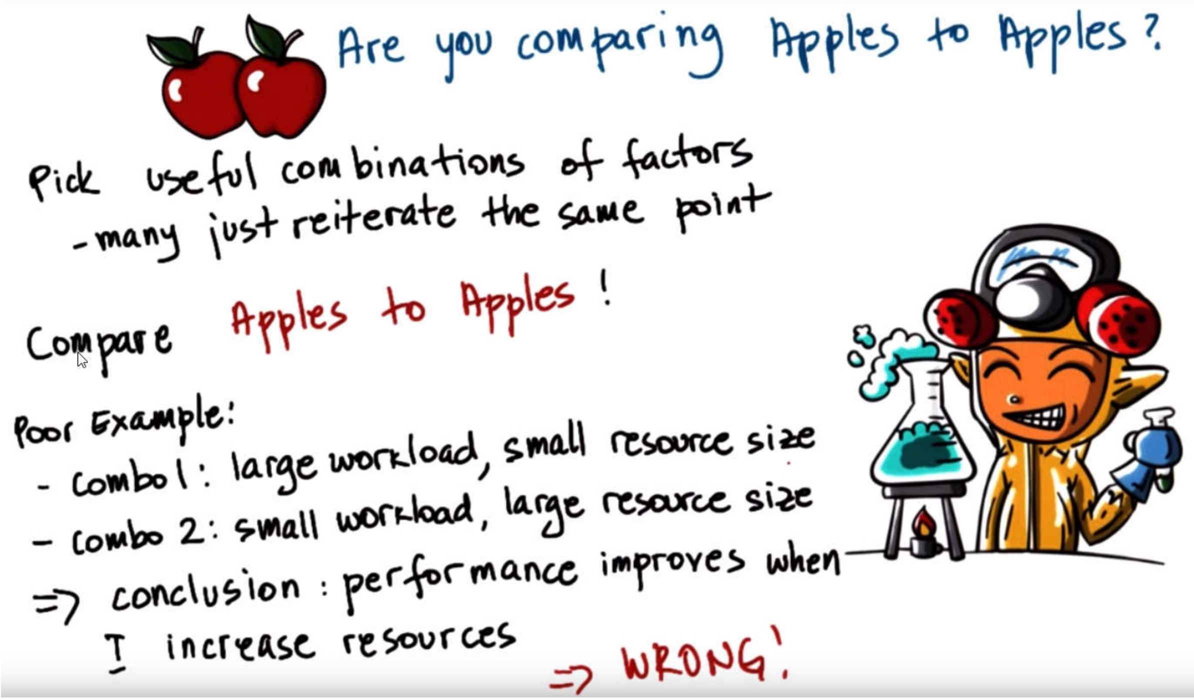
- hardware (CPU, mm...) & software (#threads, queue sizes...)

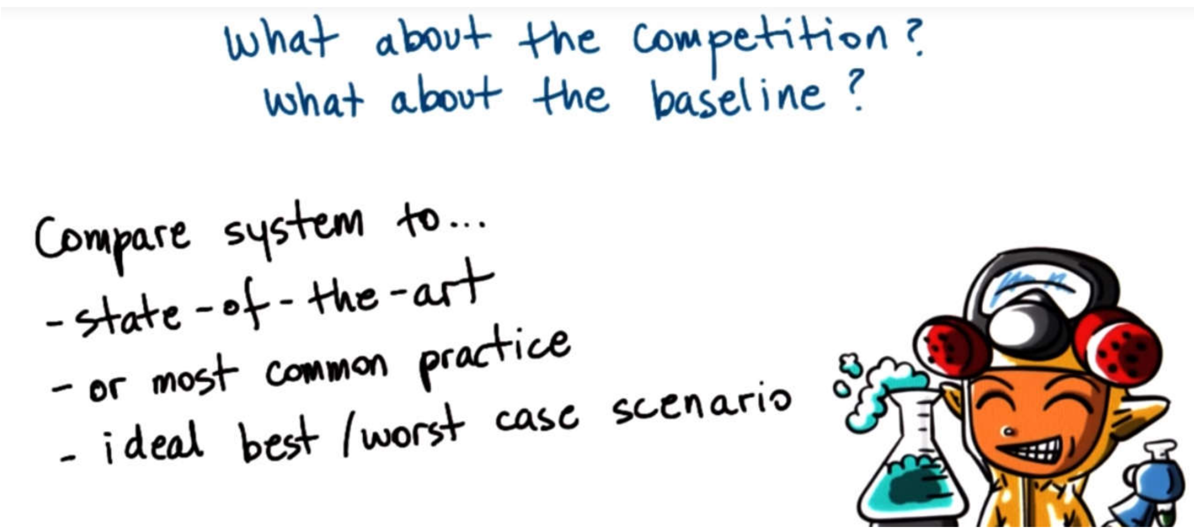
Workload:

- web server: => repuest 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





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!