# Notes on DoD

## General Notes:

* Watch polymorphic procedures from Jonathan blow
* OOP encourages AoS (Array of Struct) pattern which is said to be slow

## Research:

* Streaming prefetch bonus and general CPU bonuses
* How to analyze compiler

## Mike Acton talk:

* Where there is one there many (solve the most common problem first)
* The more context you have the better the solution, don’t throw away data or constraints that you need
* OOP confuses Maintenance with Understanding properties of data at the benefit of very little bonus to maintenance the cost is it makes it hard to solve performance problems

## Post Hand in:

### AoS vs SoA

* Maybe try to apply parallelization to AoS vs SoA example to see if there is a bigger boost to one memory access pattern than another
* Write about stack/heap
* Experiments
  + Mess around with the iterations
  + Make the same experiment with objects and figure out if there is a measurable difference between thing being stored on the stack vs the heap
  + Introduce more/less data/variables
  + Introduce different kinds of data
  + Introduce operations together with the data (which is typically against the data-oriented philosophy)

### OOP vs DoD example (Stocks)

* Make everything into structs
* Only use flat linear arrays and see if separating huge chunks into separate arrays yield better results
* Look at code metrics
* Make unit tests to show difference

### Disposition

* Introduce the subject
  + Why is it interesting, and what led me to it?
    - Mention the fact that it seems like the “end user” in OOP always gets referred to as the software designer and not the user of the actual program which always seemed counterintuitive
  + How I wanted to approach it
* I really wanted to be able to look at the compiler, why that wasn’t feasible
* I watched another talk from Mike Acton who was hired by Unity shortly after his talk from CppCon because they have many problems with performance. Unity also uses C# and Acton and his team are looking to make what they call High Performant C# (HPC#), and a compiler for it without the use of Garbage Collection.
* Since I couldn’t look under the hood properly, I just tried to apply the basic principles or at least the ones I was able to find to make a DoD version of Pers Stock example
* Show the project
  + Structs/No classes (stack instead of heap)
  + Linear data structures (basically only arrays)
  + Data and operation/function/method separation
  + Minimal states (Look for bools)
* Conclusion
  + C# is not suited well for dataoriented design because of garbage collection

## References

### Mike Acton Talk:

<https://www.youtube.com/watch?v=rX0ItVEVjHc>

<https://www.youtube.com/watch?v=p65Yt20pw0g>

### Stoyan Nikolov Talk:

<https://www.youtube.com/watch?v=yy8jQgmhbAU>

### Jonathan Blow Talks:

<https://www.youtube.com/watch?v=TH9VCN6UkyQ&list=PLmV5I2fxaiCKfxMBrNsU1kgKJXD3PkyxO>

### What Is CPU Cache? Article:

<https://www.makeuseof.com/tag/what-is-cpu-cache/>

### Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP):

<http://gamesfromwithin.com/data-oriented-design>

### Forum Posts:

<https://stackoverflow.com/questions/29192679/memory-layout-optimization-in-c-sharp>

<https://stackoverflow.com/questions/56255090/does-c-sharp-allocate-array-of-structs-as-structure-of-arrays-in-memory>

### AoS vs Soa:

<https://www.youtube.com/watch?v=ScvpoiTbMKc>

<https://www.youtube.com/watch?v=Dj3lkfIEUyA&t=1s>