Data-Oriented Design

July 2018
Kirk McCulloch

An Aperture Science Computer Efficiency Enrichment Strategy

Data Oriented Design is...

- A software design methodology
- Separates data from the requisite logic
- Brings the data to the forefront
- Offers performance benefits over raw OOP
- Can be used alongside OOP and other patterns
- First popularized in the video game industry

A shift of focus

- Focus on how data is
  - Represented
  - Moved
  - Shared
  - Transformed
- Not on
  - What objects do
  - How objects interact

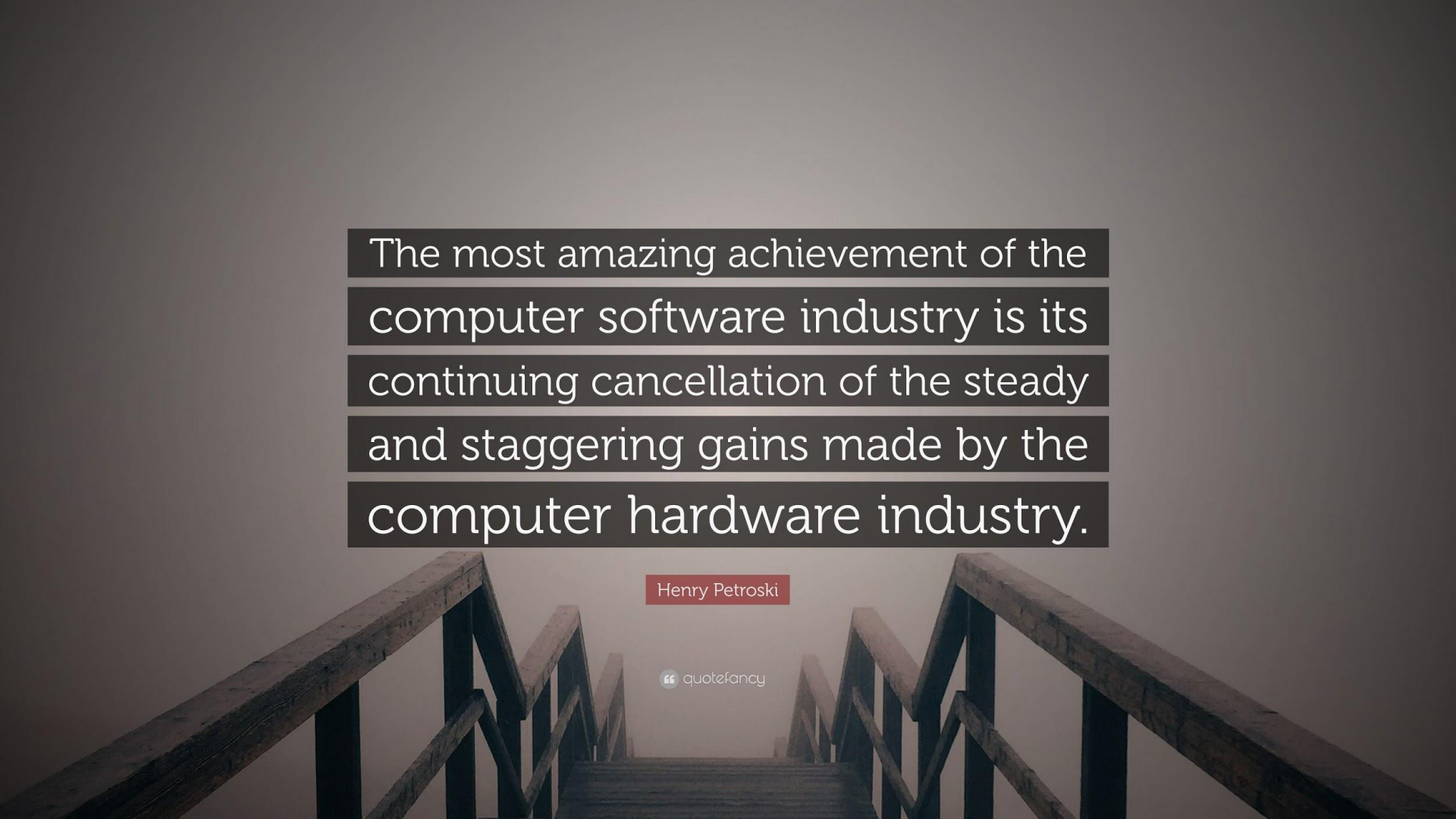
A shift of focus

- Focus on how data is
  - Represented
  - Moved
  - Shared
  - Transformed
- Not on
  - What objects do
  - How objects interact

A shift of focus

- Focus on how data is
  - Represented
  - Moved
  - Shared
  - Transformed
- Not on
  - What objects do
  - How objects interact

Why do this?

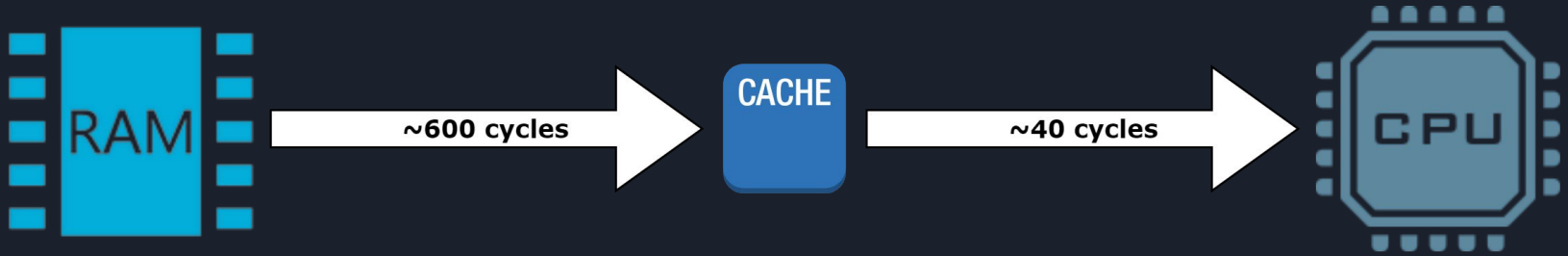

The most amazing achievement of the computer software industry is its continuing cancellation of the steady and staggering gains made by the computer hardware industry.
Henry Petroski
quotefancy

Why do this?

- It’s all about real world performance
- Improved utilization of CPU cache
- Parallelization becomes much easier
- Code is generally more modular
- Modular code is easier to test!

Why do this?

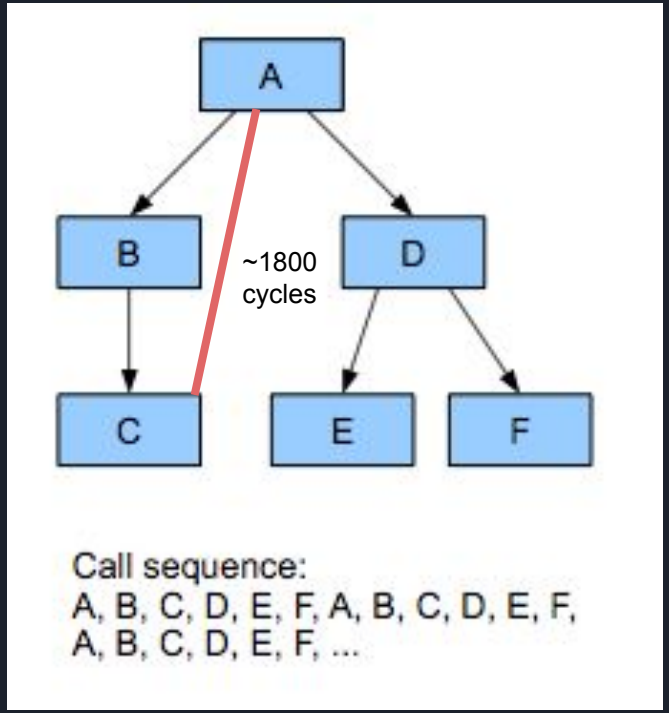
- Organize data in a CPU friendly way
  - Memory is slow, lookups are expensive
  - CPU cache is fast, lookups are cheap
- CPU friendly is also developer friendly
- Cut down on “cache misses”



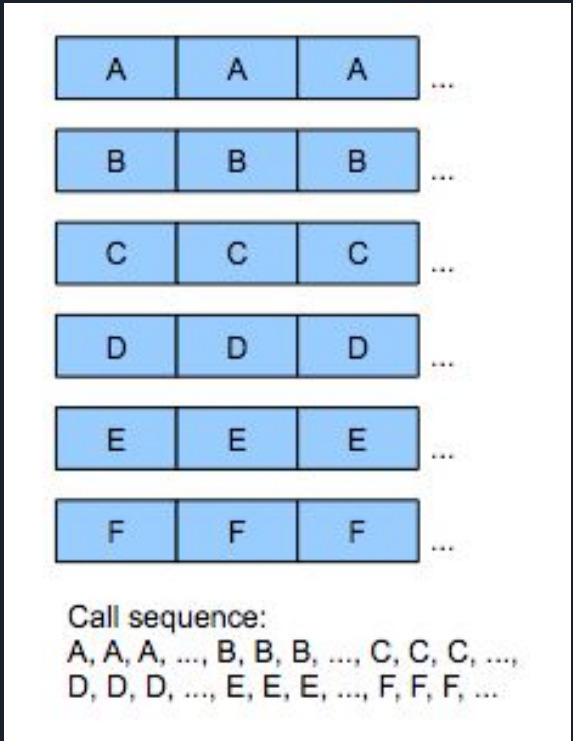
Data Layouts

The drawback of OOP

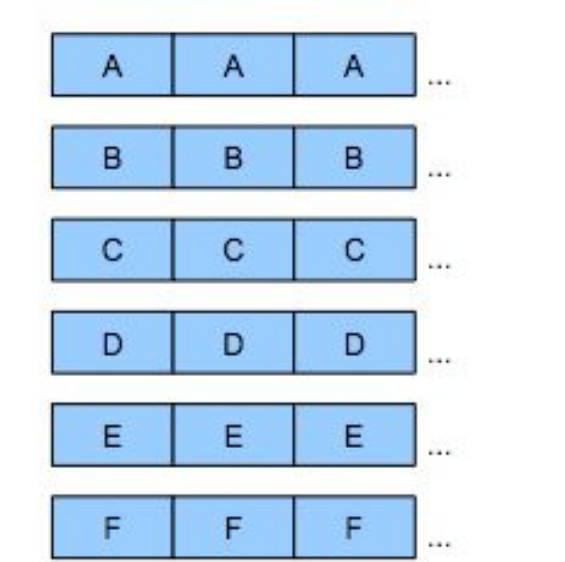
- Data naturally laid out in a tree
- Becomes deeply nested
- Many kinds of data together
- Complex lookups
- Most ideal for human brains
- Hard to parallelize, non-sequential



- Code is much less important
- Data is most important
  - Types of data
  - How data is read
  - How data is processed
- Design for the access pattern

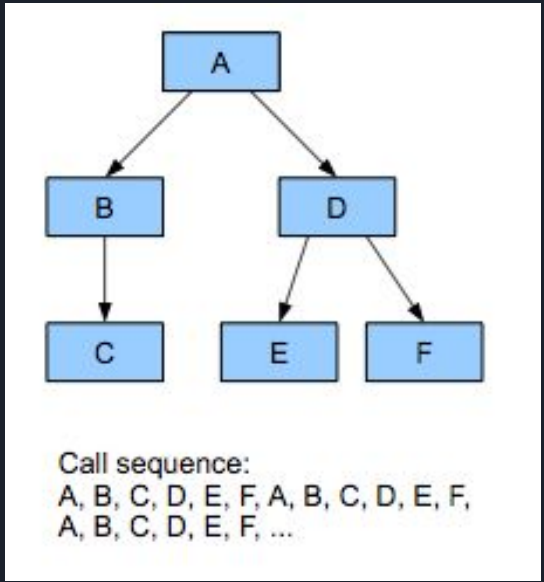


- Data laid out in linear arrays
- Similar data packed together
- Lookups are trivial
- Ideal for robot brains
- Easy to parallelize, sequential



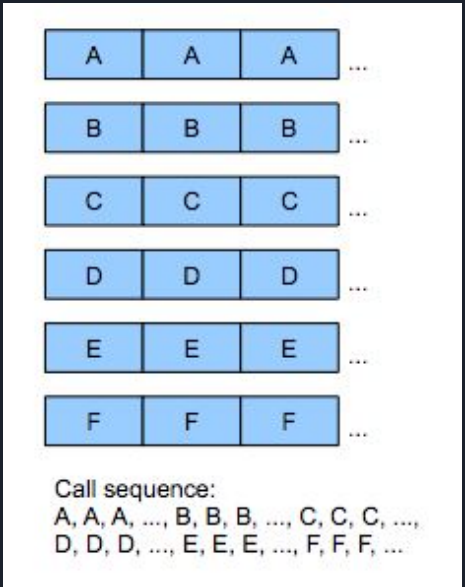
Design for the access pattern

deep and complex



Incidental processing of diverse data

flat and contiguous



Repetitious transformation of many items

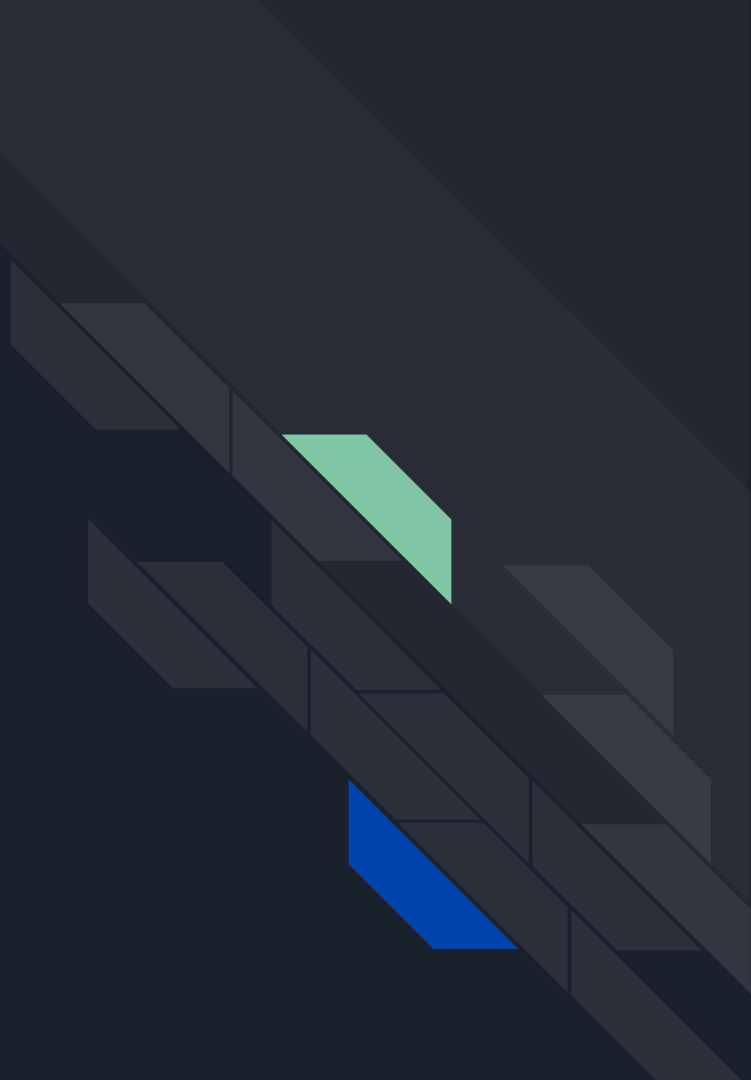
Won’t this make code harder to understand and write?

YNEOS!

- Requires you to think differently than OOP
- You have to train your brain
- Old habits die hard

- Flexible and modular code
- Threadability!

Modularity



Separation of logic and state

- Think of code as transformations of data
- Naturally more functional patterns emerge
- Data agnostic functions are more reusable
- Code that acts on one data set easily acts on many
- Code Lego!

Parallelization

- Keeping track of data accessors is tough
- Resource locking is counter productive
- Sync points negate the effect of threads
- Chaining cuts down on sync points
- Need to map out dependencies and control workflow

- DOD helps find dependency chains
- Keep units of data on one thread
- Take advantage of order independent execution
- Isolate independent code paths, execute separately
- When you know the flow of data it becomes easier to verify that your threads are correct and safe

Sauces

- DICE
  http://www.dice.se/wp-content/uploads/2014/12/Introduction_to_Data-Oriented_Design.pdf
- Noel Llopis
  http://gamesfromwithin.com/data-oriented-design
- Noel Llopis
  http://gamesfromwithin.com/data-oriented-design-now-and-in-the-future
- Niklas Frykholm
  http://gamedevs.org/uploads/practical-examples-in-data-oriented-design.pdf
- Mike Acton
  https://www.youtube.com/watch?v=rX0ItVEVjHc
- Marinara
  https://www.bonappetit.com/recipe/classic-marinara-sauce
- Extra materials
  https://github.com/dbartolini/data-oriented-design

https://media.giphy.com/media/K3StLJ7MtN1Li/giphy.gif