Data-Oriented Design

Mike Acton, *Data-Oriented Design and C++*, CppCon 2014

TLDR

In FP/OOP, you focus on organizing your *code*.

In DOD, you focus on organizing your *data*.

The “Why” of Data-Oriented

The Problem

> *Picture this: Toward the end of the development cycle, your game crawls, but you don't see any obvious hotspots in the profiler. The culprit?* ***Random memory access patterns and constant cache misses.*** *In an attempt to improve performance, you try to parallelize parts of the code, but it takes heroic efforts, and, in the end, you barely get much of a speed-up due to all the synchronization you had to add. To top it off, the code is so complex that fixing bugs creates more problems, and the thought of adding new features is discarded right away. Sound familiar?*

—Noel Llopis, *Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP)*

The Problem

> *That scenario pretty accurately describes almost every game I've been involved with for the last 10 years. The reasons aren't the programming languages we're using, nor the development tools, nor even a lack of discipline. In my experience,* ***it's object-oriented programming (OOP) and the culture that surrounds it that is in large part to blame*** *for those problems. OOP could be hindering your project rather than helping it!*

—Noel Llopis, *Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP)*

The Problem

Summary

- Game runs too slowly
- Why? Too many cache misses
- Why? Random—rather than sequential and predictable—accesses to memory
- Why? Common OO usage allocates objects in a tree/graph independent of the memory hierarchy

The problem with OO culture

Micro-example

An example of common OO design advice.

> *Depend on abstractions, not on concretions[1].*

Uncle Bob

[1] BTW: Concretion is a geological term; it's not the opposite of abstraction.

Micro-example

```
static private double arrayAvg(double[] it) {
    double sum = 0.0;
    for (double x: it) {
        sum += x;
    }
    return sum / it.length;
}

/* VERSUS */

static private double iteratorAvg(Iterator<Double> it) {
    double sum = 0.0;
    int len = 0;
    while (it.hasNext()) {
        sum += it.next();
        len += 1;
    }
    return sum / len;
}
```

Micro-example

Method	Time
Array	0.6 sec
Iterator	1.2 sec

- The “good OO” implementation is twice as slow
- The array version can be optimized (SIMD, split array + spawn threads)
- The interface version cannot be optimized.
- What happens in a system where *everything* follows this advice?

Micro-example

> *Different problems require different solutions.*
>
> *Solving problems you probably don't have creates more problems you definitely do have.*

Mike Acton

The “What” of Data-Oriented

What is data-oriented design?

> *Data-oriented design is an approach to optimising programs by carefully considering the memory layout of data structures*

James McMurray, *An introduction to Data Oriented Design with Rust*

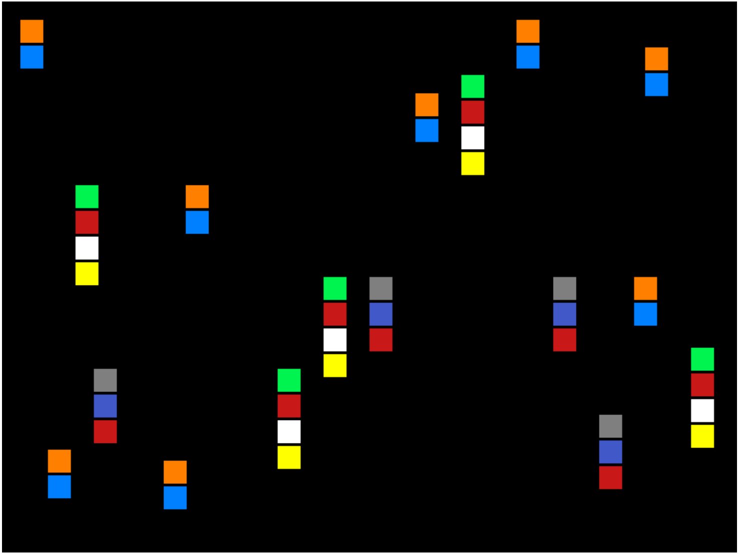
What is data-oriented design?

Data-oriented design is an approach to ***organizing*** *programs by carefully considering the memory layout of data structures*

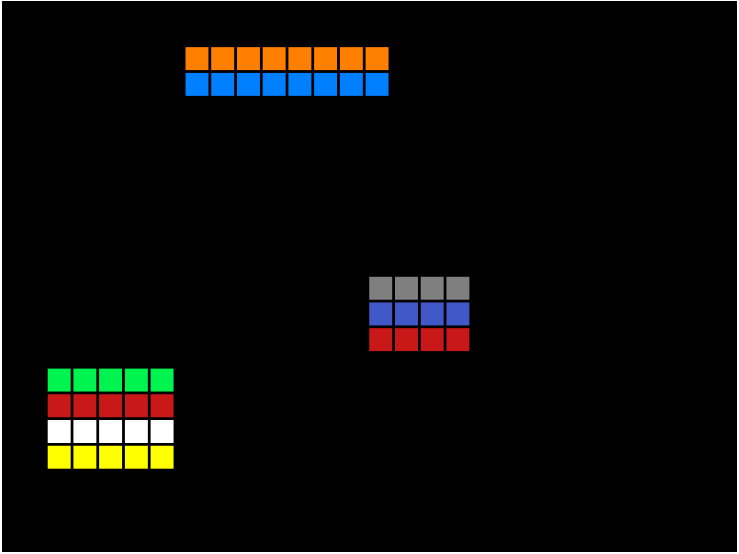
What is data-oriented design?

Carefully considering the memory layout of data

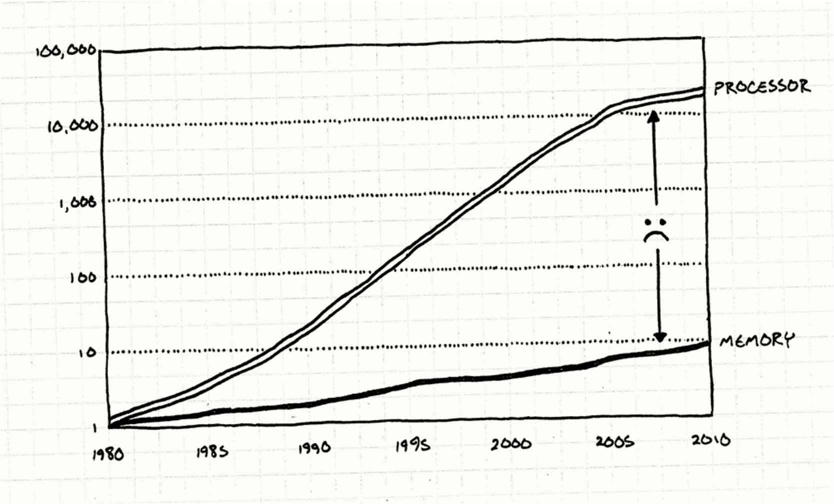
What is data-oriented design?



What is data-oriented design?

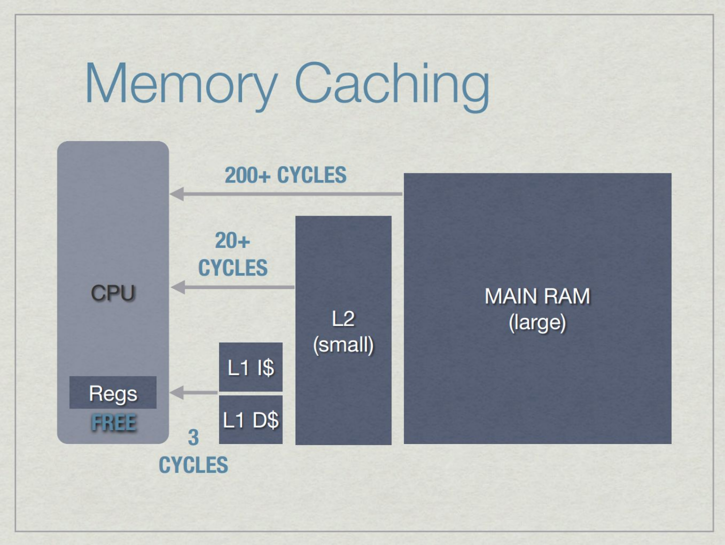


Computer Memory in Modern Times



Source: `http://gameprogrammingpatterns.com/data-locality.html`

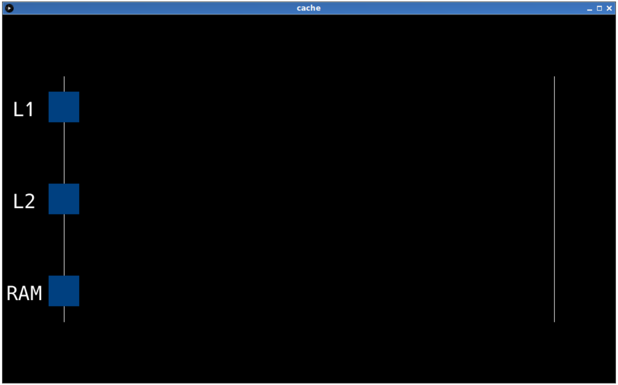
Computer Memory in Modern Times



Source: Jason Gregory, *Game Engine Architecture*

Computer Memory in Modern Times

Demo



The “How” of Data-Oriented

Better cache use example

Cache example

Find the user that logged in most recently

The input file

```
0:alicea_brittnee:/home/alicea_brittnee:/bin/bash:1576347215
1:carolee_ange:/home/carolee_ange:/bin/bash:1572506918
2:claudie_halsted:/home/claudie_halsted:/bin/bash:1576263841
3:brianne_hizar:/home/brianne_hizar:/bin/bash:1576619633
4:marika_barraza:/home/marika_barraza:/bin/bash:1566035342
5:cristiona_randolf:/home/cristiona_randolf:/bin/bash:1573717947
6:meghan_lief:/home/meghan_lief:/bin/bash:1572370708
7:elly_lemmuela:/home/elly_lemmuela:/bin/bash:1590743916
8:rora_baily:/home/rora_baily:/bin/bash:1567493247
9:sharron_medor:/home/sharron_medor:/bin/bash:1573733695
...
```

Cache example

Array of Structs ($\sim$ Postgresql)

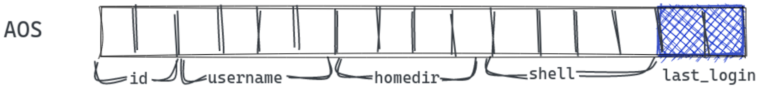
```
type User = struct {
        id         int64
        username   string
        homedir    string
        shell      string
        last_login int64
}


latest_timestamp := int64(0)
latest_username := ""
for _, user := range users {
        if user.last_login > latest_timestamp {
                latest_timestamp = user.last_login
                latest_username = user.username
        }
}
```

Cache example

Array of Structs

```
:) for _ in {1..5}; do go run most_recent_login.go; done
last_login: 1596901628; user: eolanda_pinchas, time: 8.43612ms
last_login: 1596901628; user: eolanda_pinchas, time: 11.060461ms
last_login: 1596901628; user: eolanda_pinchas, time: 11.229049ms
last_login: 1596901628; user: eolanda_pinchas, time: 8.513542ms
last_login: 1596901628; user: eolanda_pinchas, time: 8.479247ms
```



Cache example

Hot/Cold Split

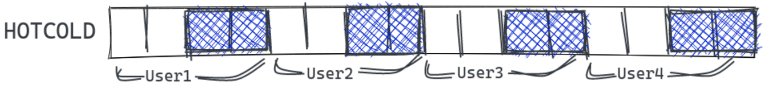
```
type UserInfo = struct {
        id       int64
        username string
        homedir  string
        shell    string
}
type User = struct {
        info       *UserInfo
        last_login int64
}


latest_timestamp := int64(0)
latest_username := ""
for _, user := range users {
        if user.last_login > latest_timestamp {
                latest_timestamp = user.last_login
                latest_username = user.info.username
        }
}
```

Cache example

Hot/Cold Split

```
:) for _ in {1..5}; do go run most_recent_login.go; done
last_login: 1596901628; user: eolanda_pinchas; time: 2.549558ms
last_login: 1596901628; user: eolanda_pinchas; time: 2.53031ms
last_login: 1596901628; user: eolanda_pinchas; time: 2.588776ms
last_login: 1596901628; user: eolanda_pinchas; time: 2.536942ms
last_login: 1596901628; user: eolanda_pinchas; time: 2.549696ms
```



Cache example

Struct of Arrays ($\sim$ Vertica)

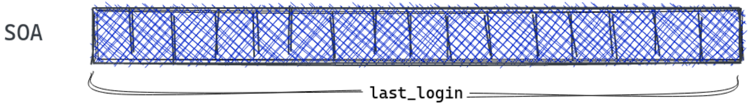
```
type Users = struct {
        id         []int64
        username   []string
        homedir    []string
        shell      []string
        last_login []int64
}


latest_timestamp := int64(0)
latest_entity := 0
for i, last_login := range users.last_login {
        if last_login > latest_timestamp {
                latest_timestamp = last_login
                latest_entity = i
        }
}
```

Cache example

Struct of Arrays

```
:) for _ in {1..5}; do go run most_recent_login.go; done
last_login: 1596901628; user: eolanda_pinchas; time: 1.45387ms
last_login: 1596901628; user: eolanda_pinchas; time: 1.449812ms
last_login: 1596901628; user: eolanda_pinchas; time: 1.464322ms
last_login: 1596901628; user: eolanda_pinchas; time: 1.460788ms
last_login: 1596901628; user: eolanda_pinchas; time: 1.446045ms
```



Cache example

Takeaways

- Same complexity ($O(n)$), but in practice constants *do* matter
- By considering the layout of data, we got a 6x speed-up
- *We* made the program 6x faster, not the compiler

An example that has nothing to do with cache

An example that has nothing to do with cache

The Problem

segvault-client

Richard found that a lot of time in rtb-gateway was spent calling the function `setelement`

```
FUNCTION                            CALLS        %
--------                            -----  -------
...
cowboy_handler:execute/2              146     5.01
segvault_protocol:collect/2        129136     7.72
erlang:setelement/3                143367    10.49
```

An example that has nothing to do with cache

The Problem

```
collect([Seg|More], #segvault_response {
  lotame = Lotame,
  adobe = Adobe, ...} = Acc) ->
  {SegmentType, SegmentId} = {
    decode_segment_type(Seg bsr ?SEGMENT_BITS),
    Seg band ?SEGMENT_MASK
  },
  Acc2 = case SegmentType of
    lotame ->
      Acc#segvault_response { lotame = [SegmentId | Lotame] };
    adobe ->
      Acc#segvault_response { adobe = [SegmentId | Adobe] };
```

For those who don’t read Erlang:

- We process one segment at a time
- We cons the segment to the appropriate list
- We setelement the new list in the response

An example that has nothing to do with cache

The Data

```
ADOBE:11111
ADOBE:11112
LOTAME:22221
LOTAME:22222
LOTAME:22223
LOTAME:22224
LIVERAMP3P:33331
LIVERAMP3P:33332
LIVERAMP3P:33333
...
```

Holy fish! The segments are grouped by segment type!

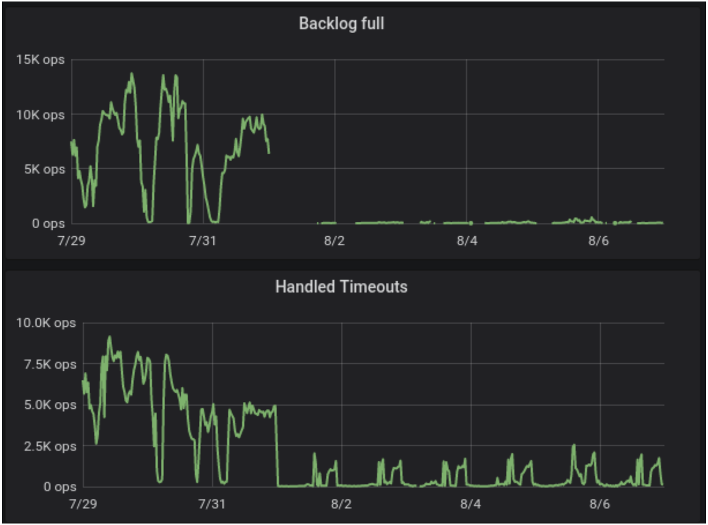
An example that has nothing to do with cache

The Solution

- Accumulate segments in a list (one cons per segment)
- When the segment type changes, store list in accumulator, empty list (one setelement per *segment type*)

An example that has nothing to do with cache

The Result



Extra slides

Random wisdom from Mike Acton

The purpose of all programs, and all parts of those programs, is to transform data from one form to another.

If you don’t understand the data you don’t understand the problem.

Conversly, understand the problem by understanding the data.

Different problems require different solutions.

If you have different data, you have a different problem.

Where there is one there are many.

The more context you have, the better you can make the solution.

There is no ideal, abstraction solution to the problem.

You can't "future proof"

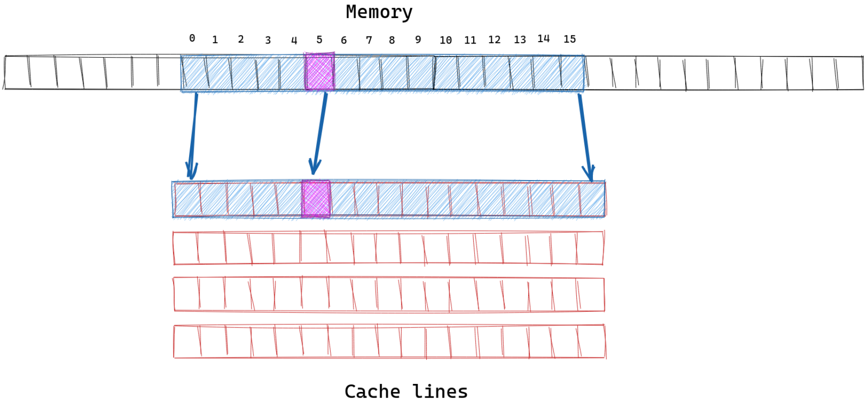
Solve the most common case first, not the most generic.

Computer Memory in Modern Times

2-minute cache course

- If a memory address is cached, don't go to memory
- When fetching from memory, bring an entire cache line (64 bytes) into the cache
- Best way to take advantage of the cache: keep data contiguous in memory

Computer Memory in Modern Times


Memory
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Cache lines