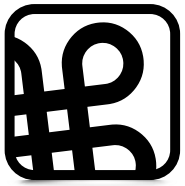
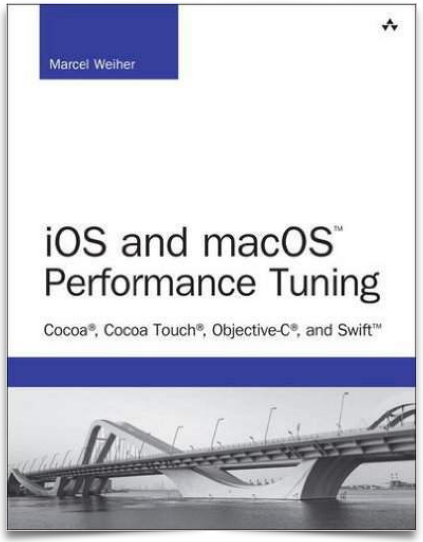


Macoun

Auf die Pladde, feddich, los!

Marcel Weiher
@mpweiher



I/O und Serialisierung

- Parameter
- Apple Techniken
- Alternativen

I/O und Serialisierung

- **Parameter**
- Apple Techniken
- Alternativen: Streams + Stores

Parameter

- I/O ist langsam
- CPU ist schnell
- Datenmenge minimieren

Parameter

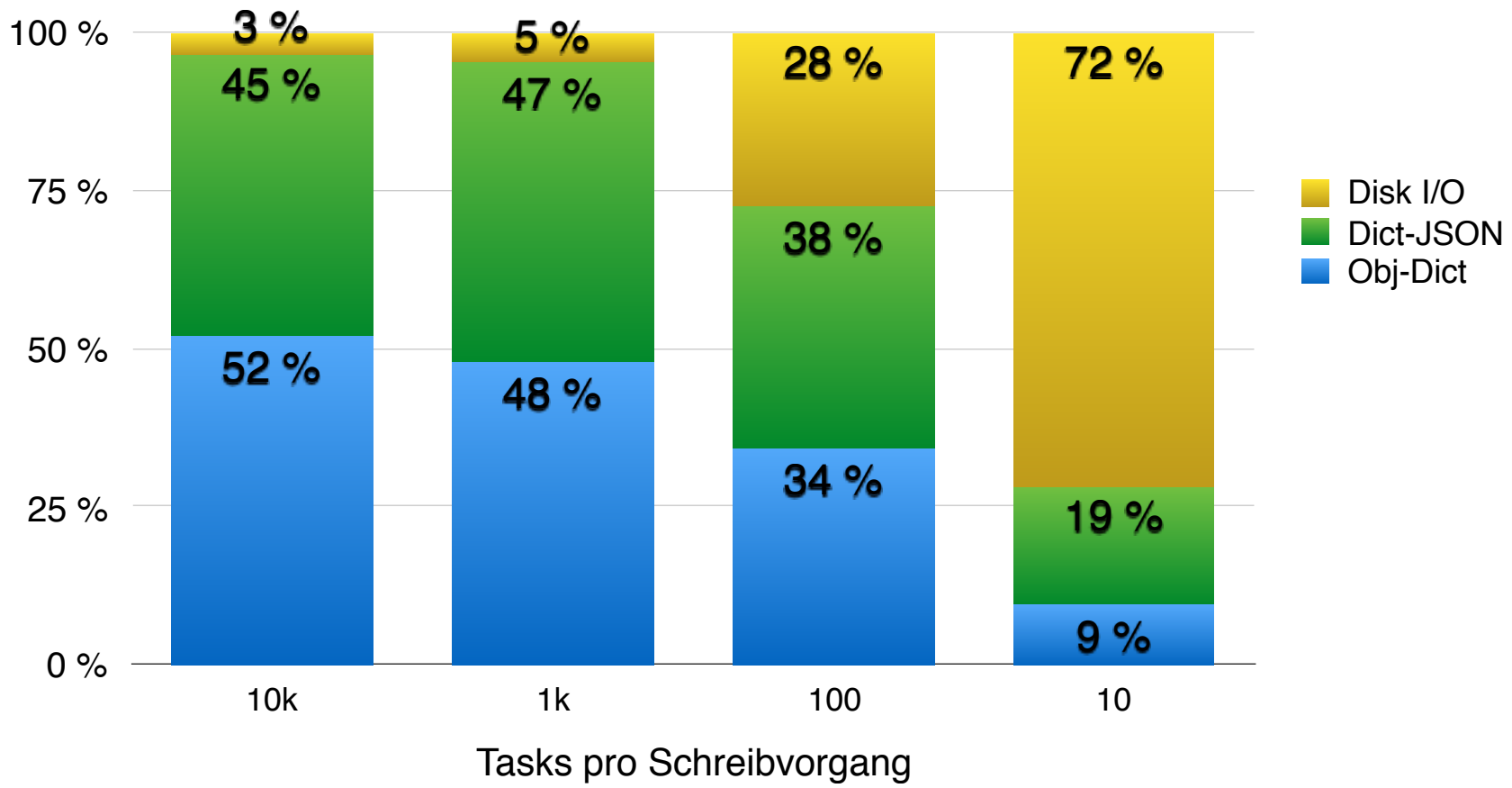
- **I/O ist langsam**
- CPU ist schnell
- Datenmenge minimieren

Demo 1

Moby

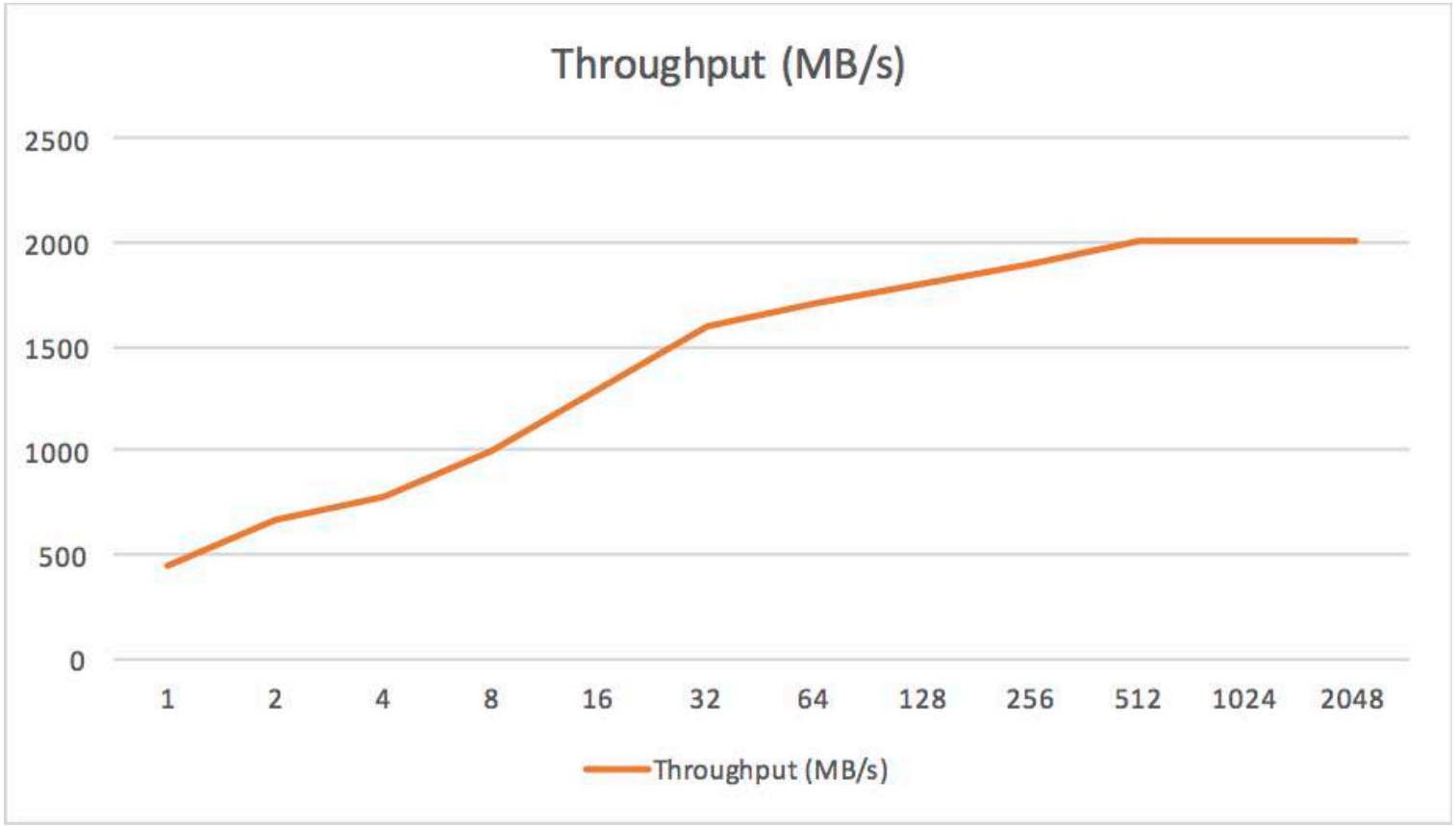
Parameter

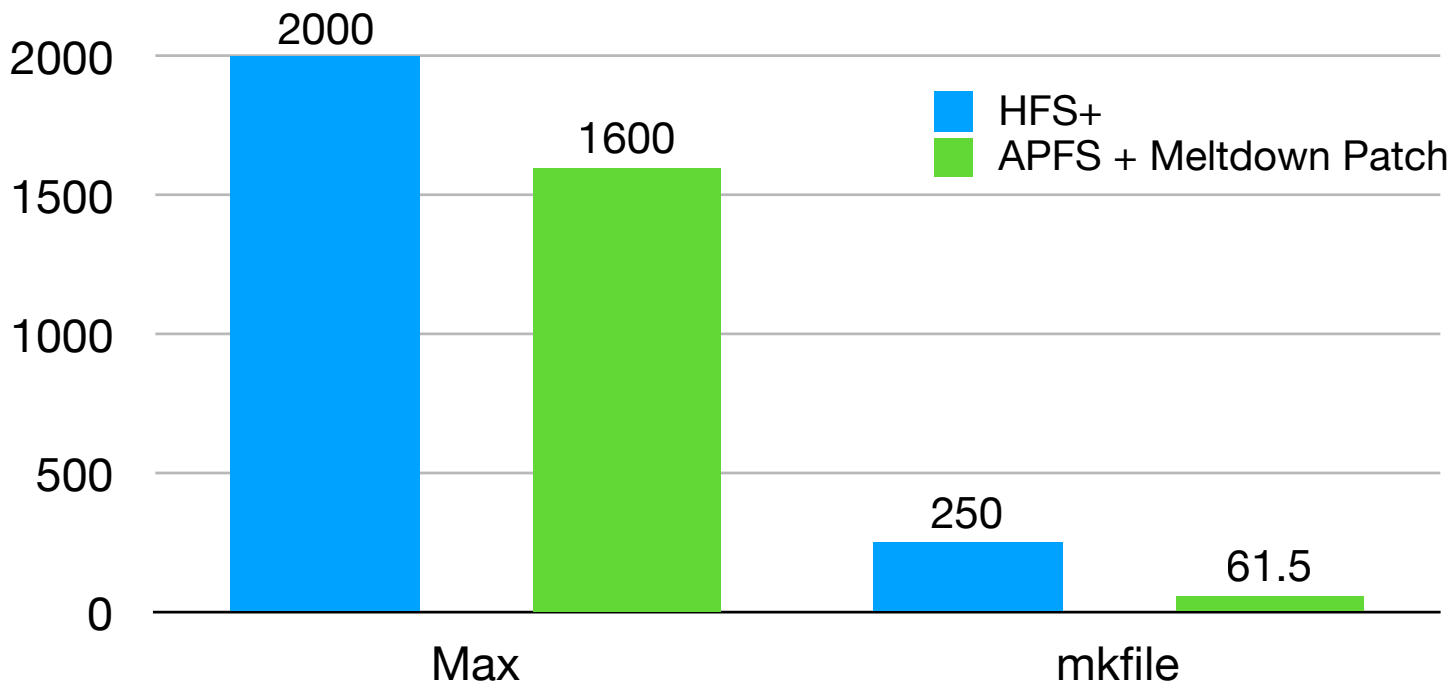
- I/O ist langsam
- **CPU ist schnell**
- Datenmenge minimieren


100 %
75 %
50 %
25 %
0 %
3 %
45 %
52 %
5 %
47 %
48 %
28 %
38 %
34 %
72 %
19 %
9 %
10k
1k
100
10
Tasks pro Schreibvorgang
Disk I/O
Dict-JSON
Obj-Dict

Demo 2

mkfile


Throughput (MB/s)
2500
2000
1500
1000
500
0
1
2
4
8
16
32
64
128
256
512
1024
2048
Throughput (MB/s)


2000
1600
250
61.5
2000
1500
1000
500
0
HFS+
APFS + Meltdown Patch
Max
mkfile

Parameter

- I/O ist langsam
- CPU ist schnell
- **Datenmenge minimieren**

Optimierungsgleichung

- Optimierungsgleichung: $n \times m$
- Optimierung = Faktorisierung $(m = o + p)$

$$n \times (o + p) \rightarrow n \times o + n \times p$$

Overhead wird nur einmal benötigt

$$\rightarrow o + n \times p$$

I/O und Serialisierung

- Parameter
- **Apple Techniken**
- Alternativen: Streams + Stores

Apple Techniken

- Property Lists / NS*Serialization
- Archiving
- Core Data
- Swift Coding

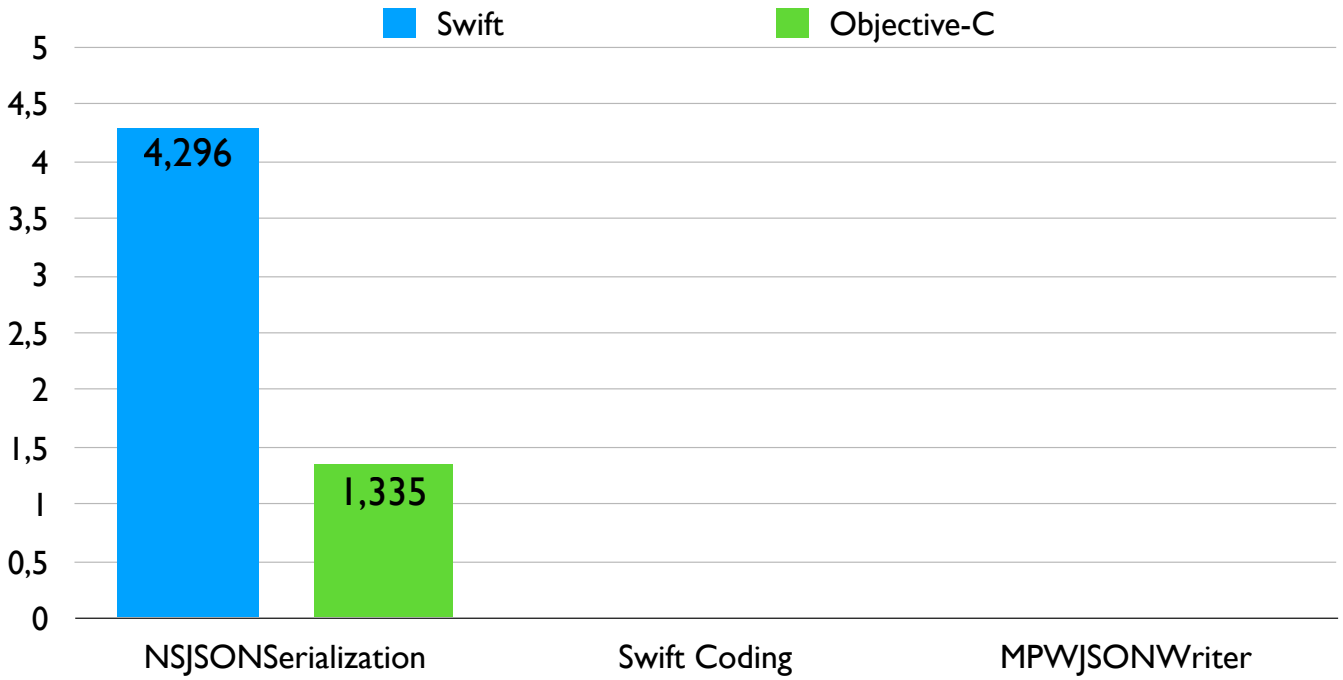
Apple: Property Lists / NS*Serialization

- NSJSONSerialization: schnellste* Apple Serialiserungsmethode
- API Problem: call/return, atomar

Demo 3

Swiftoun
Plist

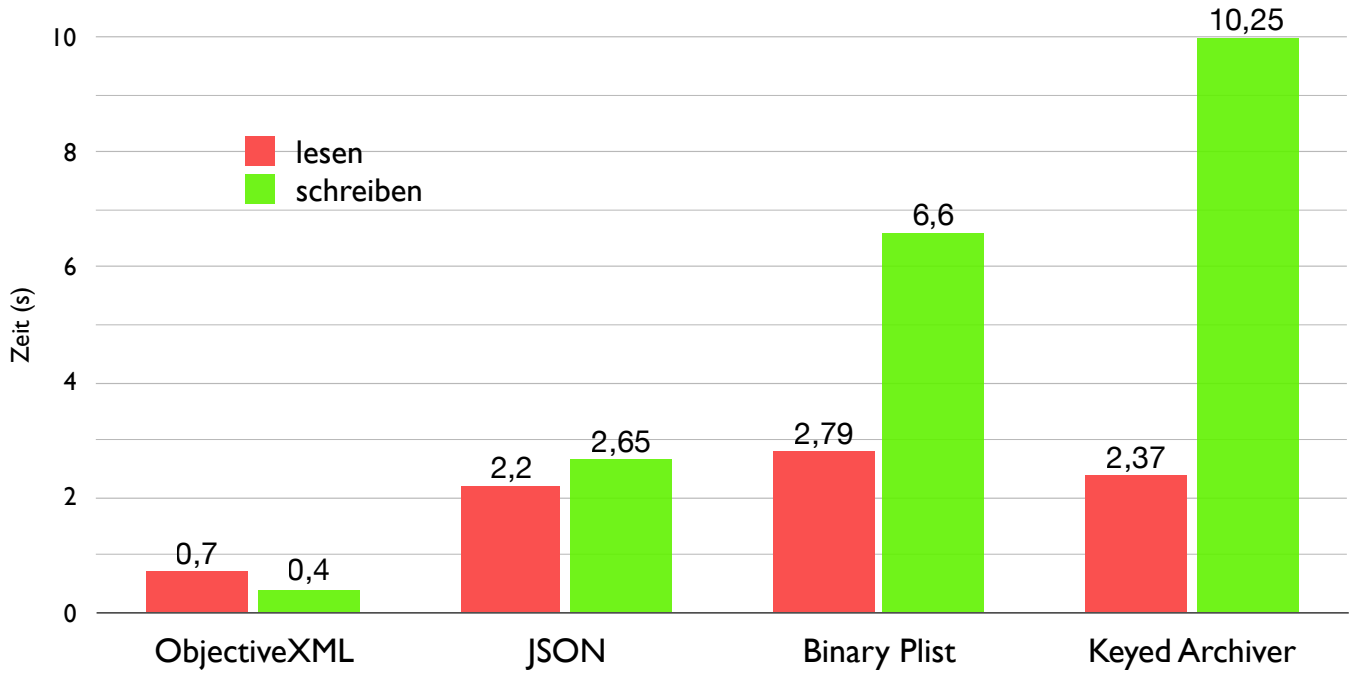
Apple JSON


Swift
Objective-C
5
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
4,296
1,335
NSJSONSerialization
Swift Coding
MPWJSONWriter

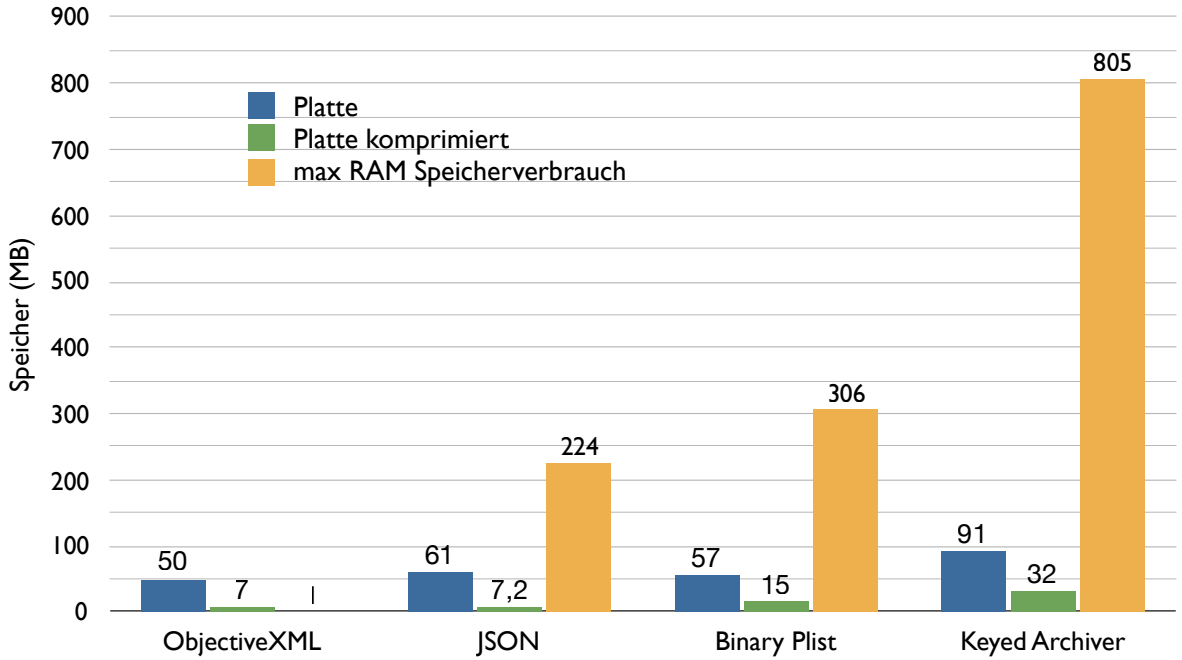
Apple: Archiving

- Von Apple für größere Datenmengen als PLists empfohlen
- Baut aber intern erst ein großes Dictionary und dann daraus eine Property List. 🤔

Apple Archiving: Zeit



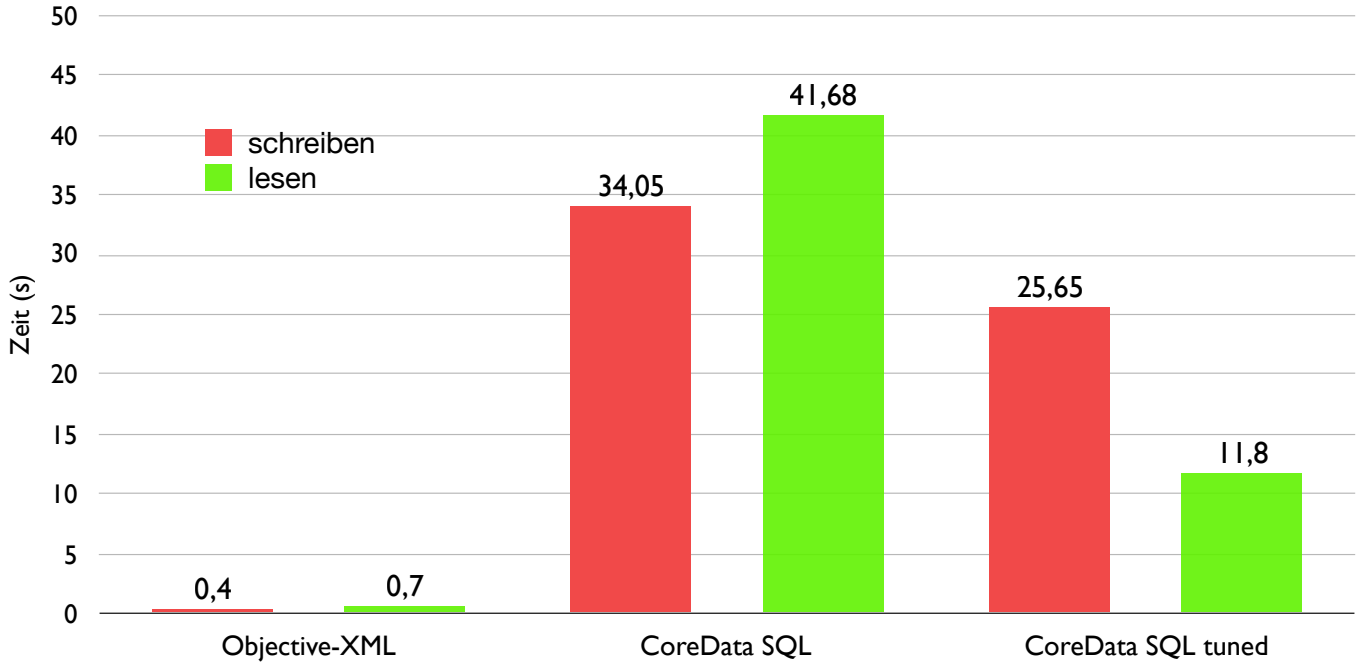
Apple Archiving: Speicher



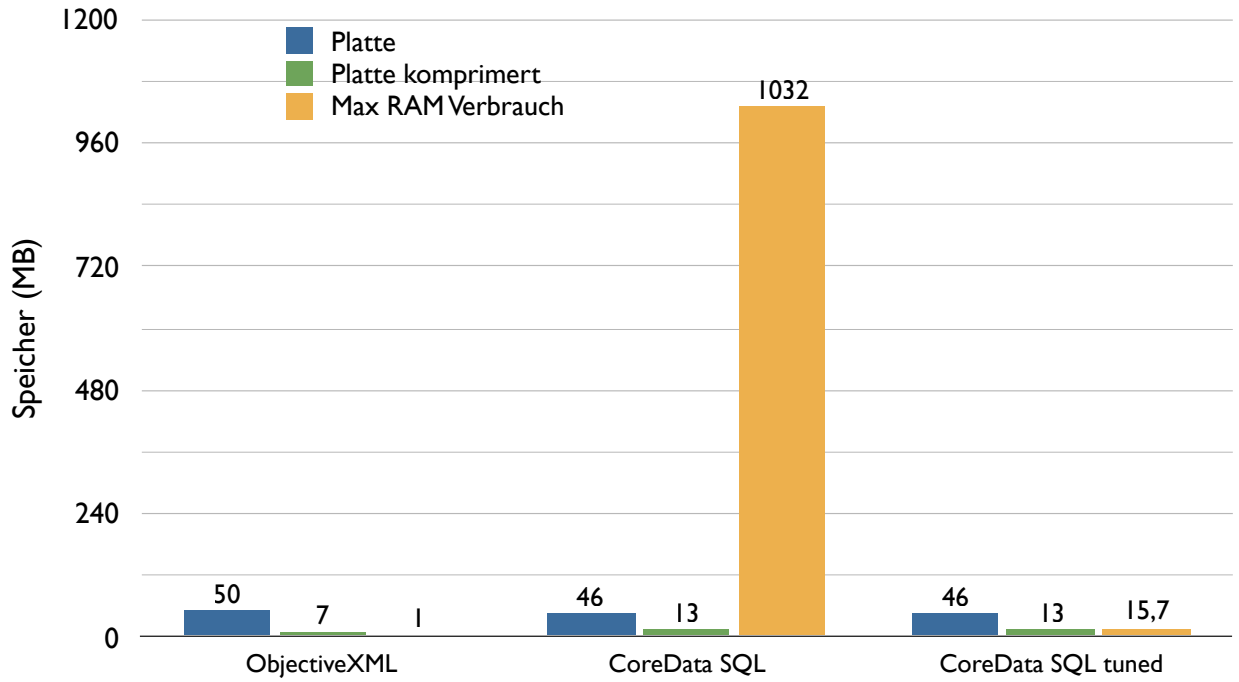
Apple: CoreData

- Wird von Apple für große Datenmengen empfohlen
- Sehr träge, optimiert durch Reduzierung von n

Apple CoreData: Zeit



Apple CoreData: Speicher



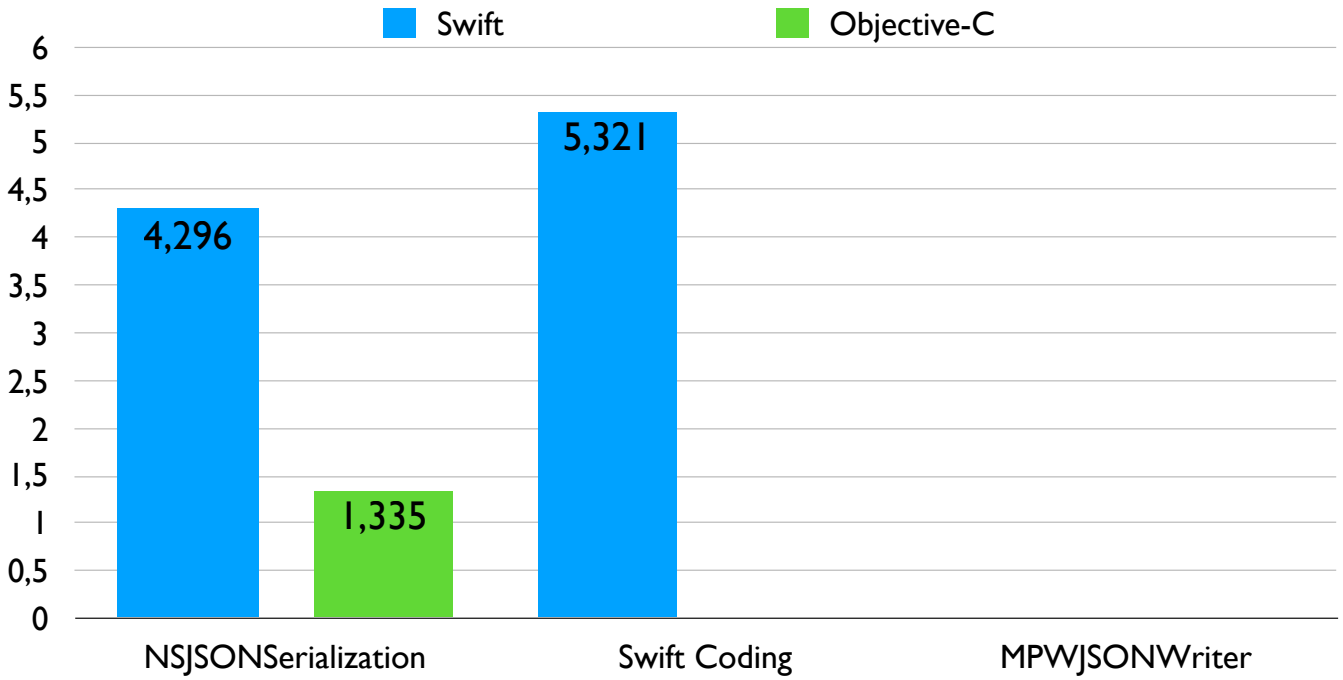
Apple: Swift Coding

- Sehr bequem (Codable Protokoll adoptieren)
- inkrementelle APIs, wie Archiver

Demo 4

Swiftoun
Coding

Apple: Swift Coding



I/O und Serialisierung

- Parameter
- Apple Techniken
- **Alternativen: Streams + Stores**

Alternativen

- (De-)Serialisierung
- Persistente Programmiersprachen
- In-Process REST

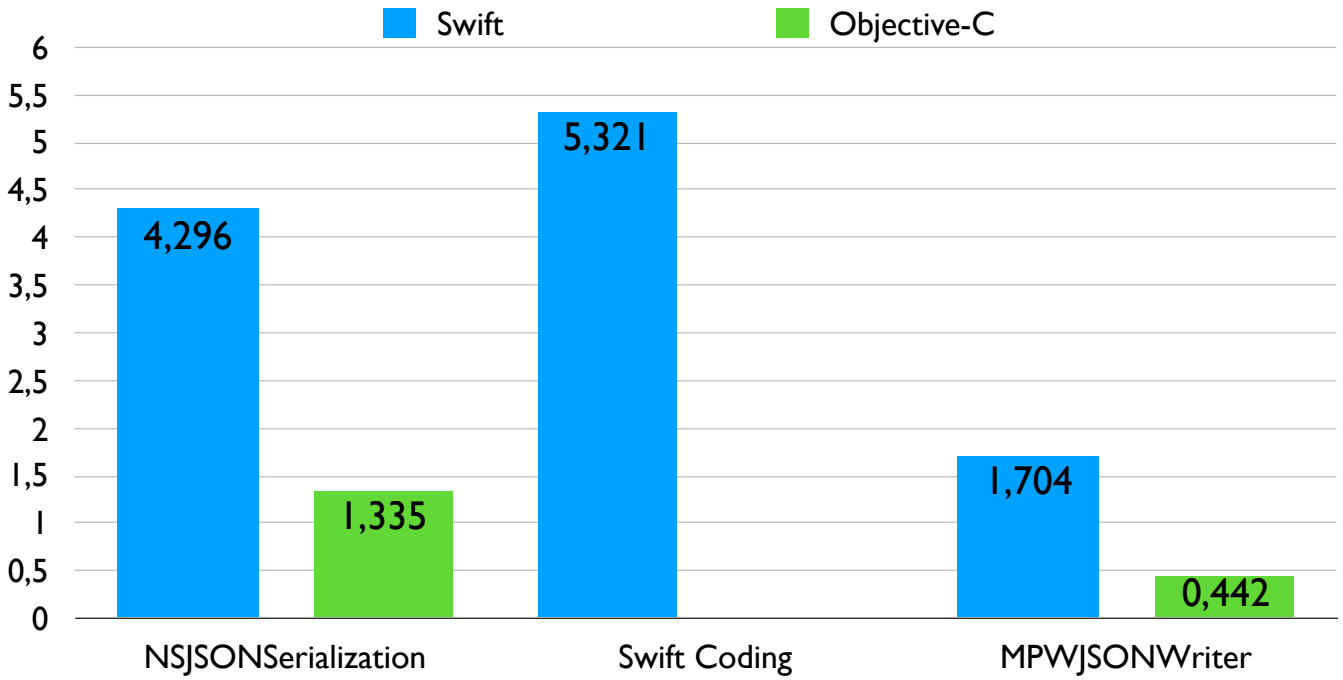
Alternativen: Serialisierung

- Andere Formate: Protocol Buffers, Cap'n Proto
- Eigene Implementierungen der existierenden Datenformate
- JSON (writing)
- Binary Property List

Demo 5

JSONWriter
Swift + ObjC

Alternativen: JSONWriter



Alternative: Binary Property List

- Binär
- Referenzen
- Index
- Type tags

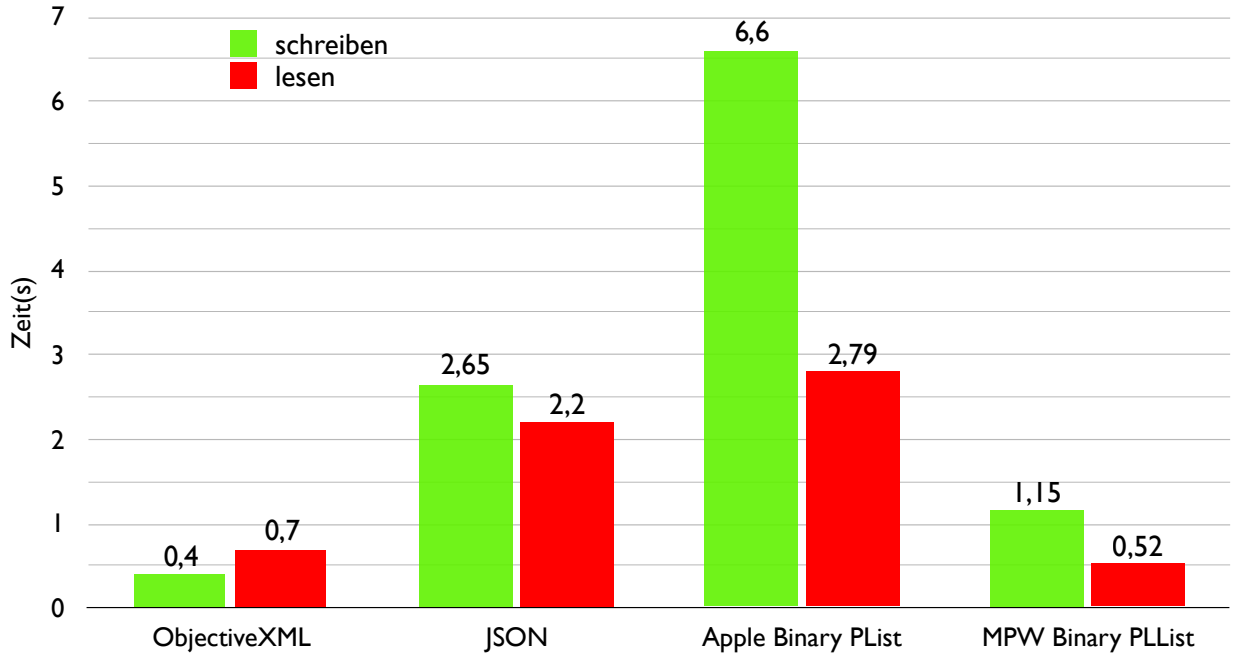
Lazy BPList

```
@interface MPWLazyBListArray : NSArray
{
    NSUInteger count;
    MPWBinaryPlist      *plist;
    MPWIntArray         *offsets;
    id  *objs;

}
@end
```

Lazy BPList

```
-objectAtIndex:(NSUInteger)anIndex
{
    id obj=nil;
    if ( anIndex < count) {
        obj=objs[anIndex];
        if ( obj == nil)  {
            obj = [plist objectAtIndex:[offsets integerAtIndex:(int)anIndex]];
            objs[anIndex]=[obj retain];
        }
    } else {
        [NSException raise:@"outofbounds" format:@"index %tu out of bounds",anIndex];
    }
    return obj;
}
```

Alternative: Binary PList



Alternative: Persistente Sprachen

- *"One of the more interesting parts (to me) about Eve wasn't the interactive programming or the literate programming, but that it included a database inside of the programming language." (HN)*
- *"I completely agree that the semantics are probably the most overall useful thing we presented in this version of Eve. Having the world just be views over a datastore that you don't have to manage in any meaningful way is hugely liberating coming from the way we program now."* – Chris Granger

Alternative: persistente Sprachen

- Squeak
- Mumps

```
set x2(“ducks”)=123
set ^x2("ducks")=123
```

- Objective-Smalltalk

Demo 6

stsh

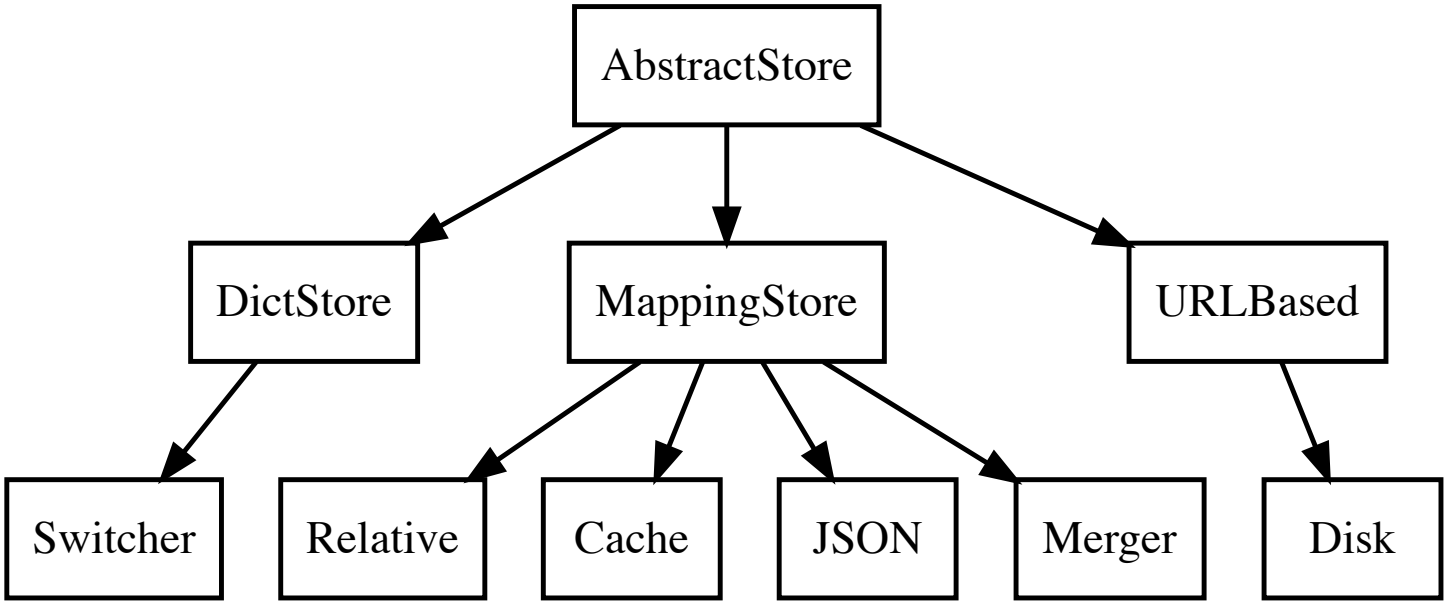
Objective-Smalltalk → Storage

- URLs → Object References
- Scheme Handler → Stores

Store: Protokoll

```
@protocol MPWStorage

-objectForReference:(id <MPWReferencing>)aReference;
-(void)setObject:theObject forReference:(id <MPWReferencing>)aReference;
-(void)mergeObject:theObject forReference:(id <MPWReferencing>)aReference;
-(void)deleteObjectForReference:(id <MPWReferencing>)aReference;
-(id <MPWReferencing>)referenceForPath:(NSString*)path;

@end
```

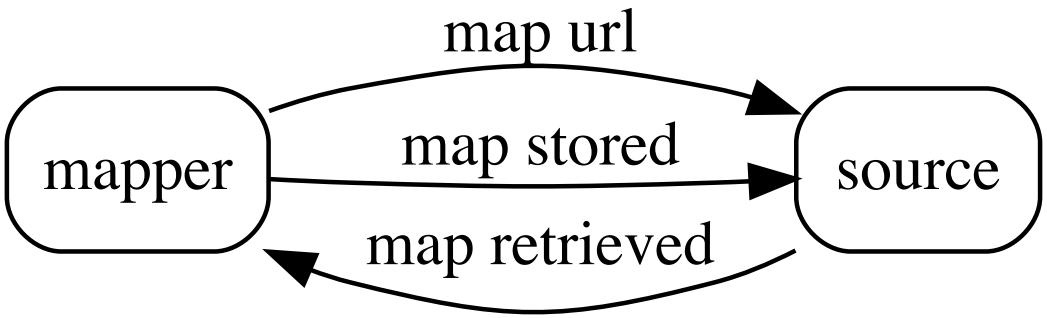
Store Klassen



Store: Dict

```
-objectForReference:(MPWReference*)aReference
{
    return self.dict[[self referenceToKey:aReference]];
}

-(void)setObject:theObject forReference:(MPWReference*)aReference
{
    self.dict[[self referenceToKey:aReference]]=theObject;
}

-(void)deleteObjectForReference:(MPWReference*)aReference
{
    self.dict[[self referenceToKey:aReference]]=nil;
}
```

Store: Map



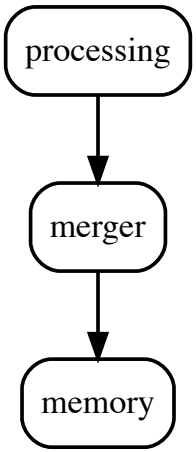
Store: Map

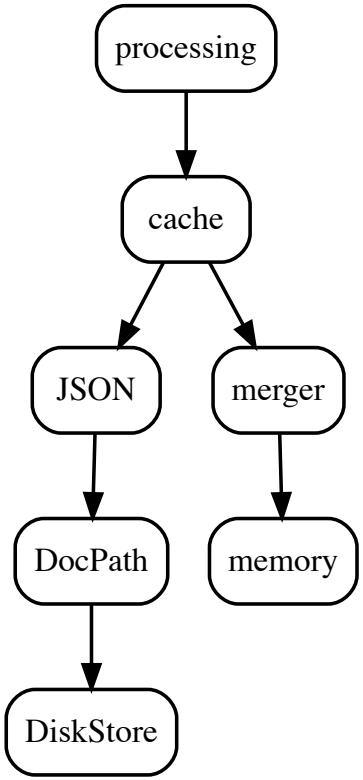
```
-objectForReference:(id <MPWReferencing>)aReference
{
    return [self mapRetrievedObject:[self.source objectForReference:[self
        mapReference:aReference]] forReference:aReference];
}

-(void)setObject:theObject forReference:(id <MPWReferencing>)aReference
{
    [self.source setObject:[self mapObjectToStore:theObject forReference:aReference]
        forReference:[self mapReference:aReference]];
}

-(void)mergeObject:theObject forReference:(id <MPWReferencing>)aReference
{
    [self.source mergeObject:[self mapObjectToStore:theObject forReference:aReference]
        forReference:[self mapReference:aReference]];
}
```

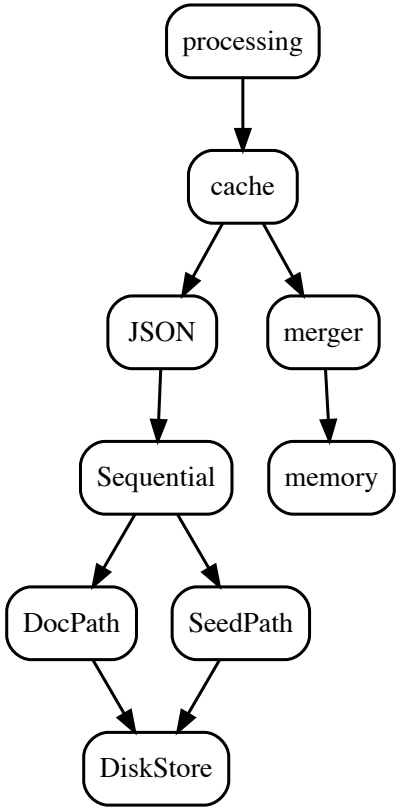
Store: Relative

```
-(id <MPWReferencing>)mapReference:(MPWGenericReference *)aReference
{
    return [self.baseReference referenceByAppendingReference:aReference];
}
```

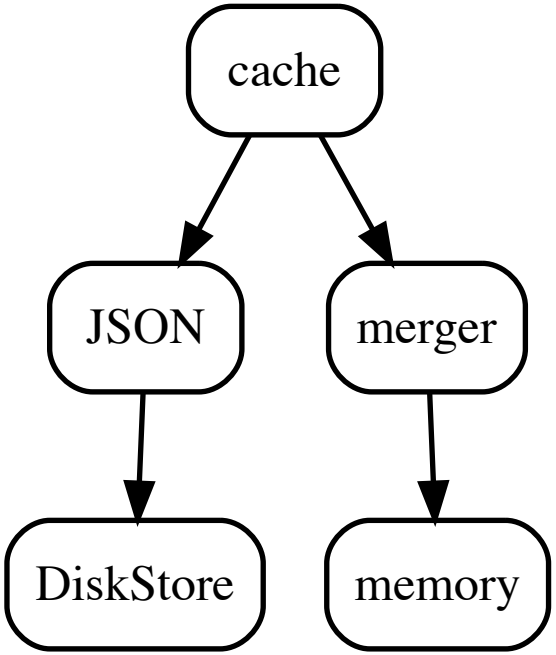
Store: Hierarchien

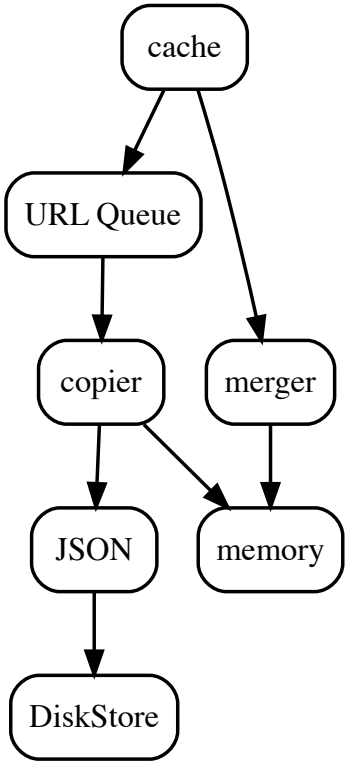




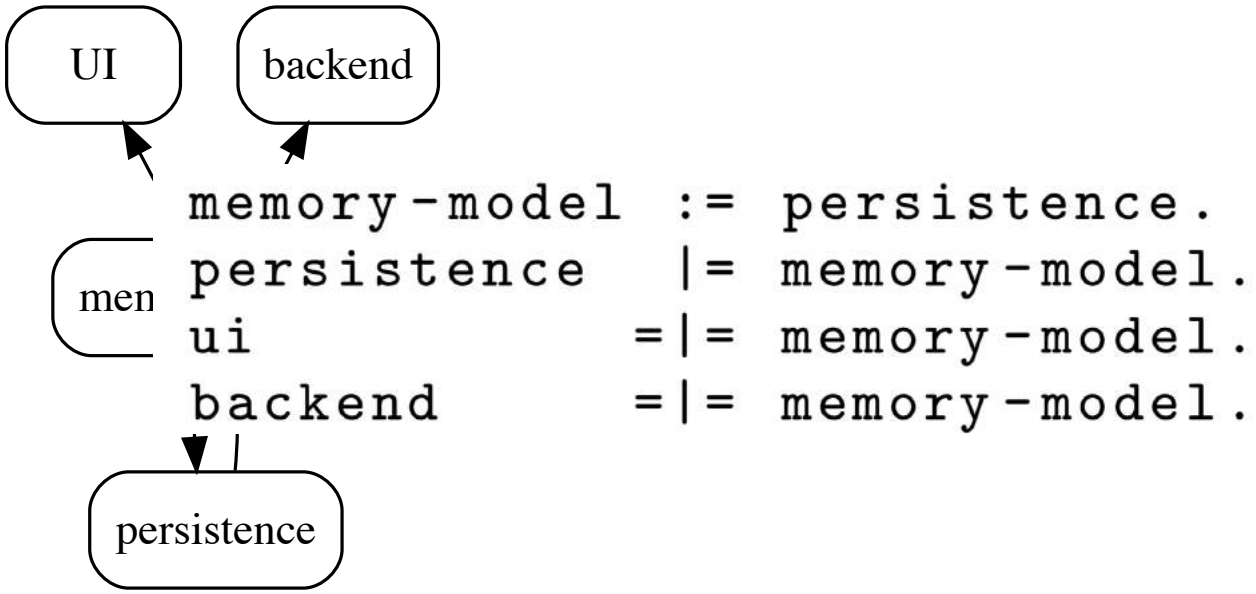


Asynchrones Schreiben





App Architektur



Fragen?

- I/O: rasend schnell, CPU-limitiert
- Apple: geht schneller
- Streams + Stores

@mpweiher