



db4o Performance Tuning

Being as fast as possible...



Break Out Session 1
Stefan Edlich, Carl Rosenberger

Result will differ!

| Handle the following results with care because:

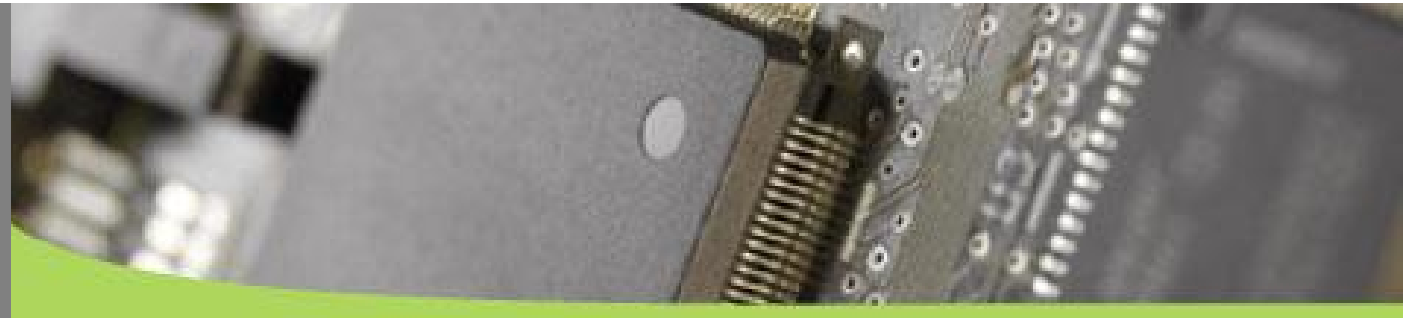
| Your system differs

| C# and Java will behave different

| Your objects differ

| You general system load will differ

| db4o improves faster (e.g. the marshaller)



INTRODUCTION

DESIGN

OBJECT

SEARCH

CASCADE

CONFIGURATION

REFERENCES

FILE

SUMMARY

The Performance Tuning Catalog

Will be inserted and cultivated in the Wiki
using your experiences

DESIGN/call defragment

DESIGN/in memory

DESIGN/split objects

DESIGN/separate logic

OBJECT / avoid inheritance

OBJECT / avoid fields

OBJECT / typesafe

OBJECT / use db4o collections

SEARCH / use index

SEARCH / sub index

SEARCH / don't search and+or

SEARCH / opt nativeQueries

SEARCH / adjust bTree settings

CONFIGURATION / blocksize

CONFIGURATION / constructorCall

CONFIGURATION / constructorTest

CONFIGURATION / schema

CONFIGURATION / freespace

CASCADE / activation

CASCADE / update+deletion

REFERENCES / weakReference

REFERENCES / purge

FILE / new adapter

FILE / flushing

Performance Tests

Using a special version of Polepos

- | a new classloader can run different db4o.jar's

- | A new configuration system can handle special configurations over

Using CarreraBahn for special purposes

- | Has a five level deep object tree

- | Writes the objects (the amount you want 100, 1000, 10000, etc.)

- | Searches and activates 10% of the objects at top level

- | Searches and updates 10% of the objects at second level

- | Searches and deletes 10% of the objects at fifth level

All tested on Athlon XP 3000+

- | 2,1 GHz

- | 1 GB RAM

INTRODUCTION

DESIGN

OBJECT

SEARCH

CASCADE

CONFIGURATION

REFERENCES

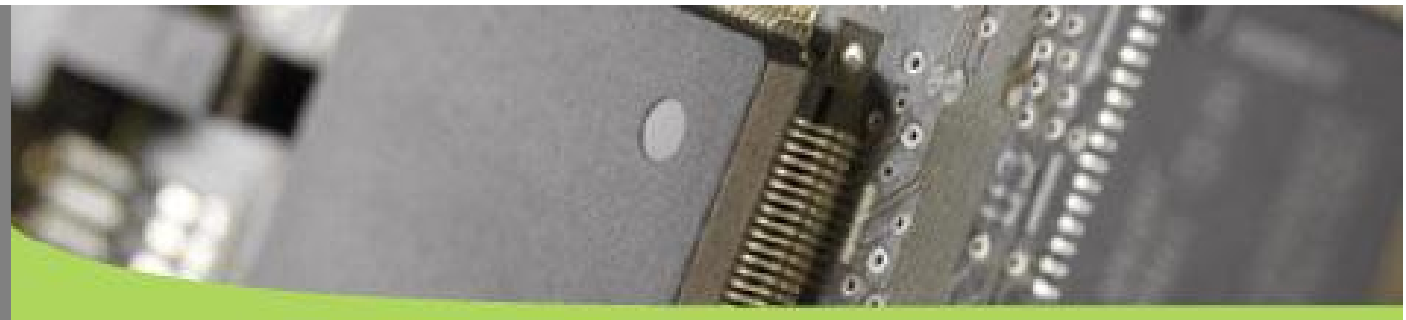
FILE

SUMMARY

DESIGN / Call Defragment

- | Defragment deletes unused references
- | Defragment shrinks the filesize

- | New fast Defragment is currently being worked on and available soon. Run it frequently!



DESIGN / Use In Memory Mode

- | Good for temporal data
- | In-Memory mode is not so fast as expected
- | Carrera Bahn test shows a 1/3 (here 37%) improvement

TestConfiguration

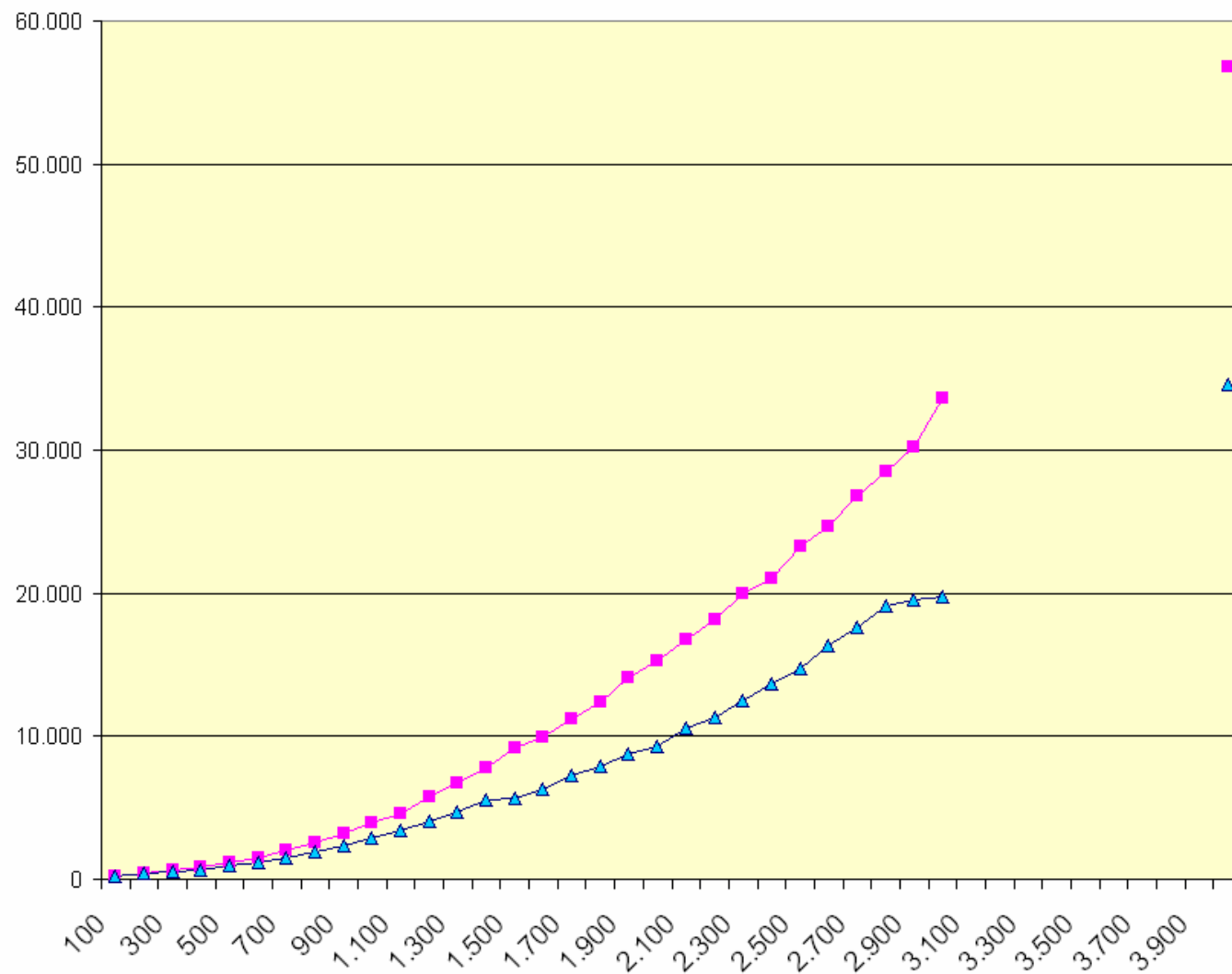
```
Db4o.configure().io(new MemoryIoAdapter());  
db = Db4o.openFile(filename);
```

This is even slower than the default FileIO! So don't use this!

```
MemoryFile file = new MemoryFile();  
ObjectContainer db = ExtDb4o.openMemoryFile(file);
```

db4objects

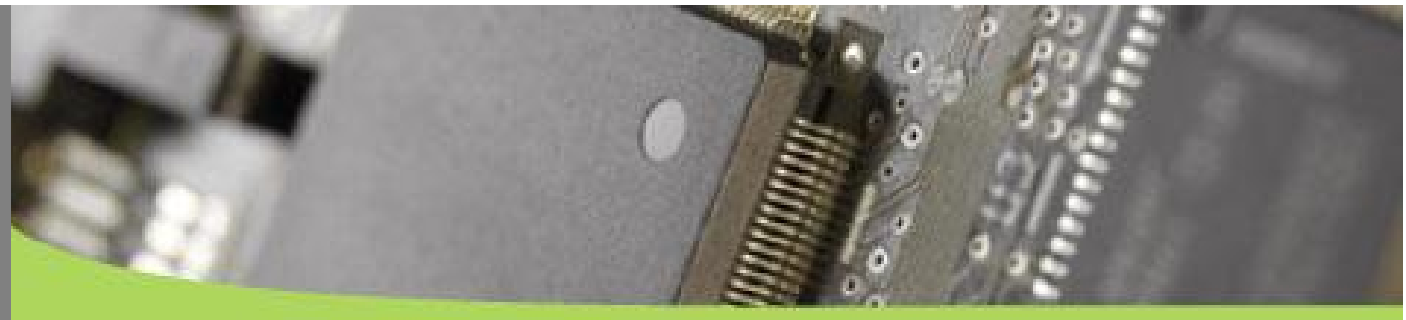
- INTRODUCTION
- DESIGN
- OBJECT
- SEARCH
- CASCADE
- CONFIGURATION
- REFERENCES
- FILE
- SUMMARY



DESIGN / Split objects

- | Group and save objects to different files
- | Suitable if the filesize becomes really big
- | Open N containers and manage up to $N \times 256$ GB
- | The difference is really hard to measure...
 - | Need help here!
- | Result:
 - | Smaller database file, better to backup alone
 - | Services can be scalable

**This only works when objects are
NOT interconnected!**

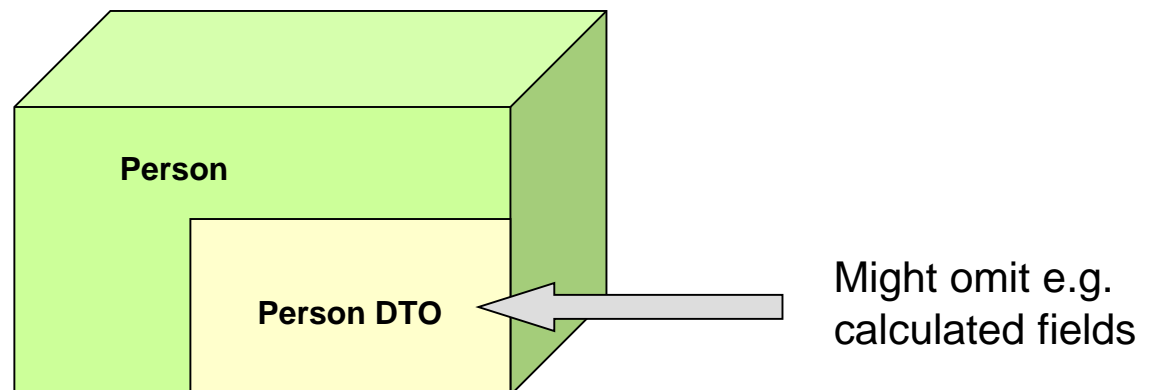


Example

```
public static void main(String[] args) {  
    ObjectContainer db1 = Db4o.openFile(dirname+"file1.yap");  
    ObjectContainer db2 = Db4o.openFile(dirname+"file2.yap");  
    db1.set(new Person("Mike", 24));  
    db1.set(new Person("Ted", 36));  
    db2.set(new Animal("Elephant", true));  
    db2.set(new Animal("Dolphin", false));  
    db1.commit();  
    db2.commit();  
    db1.close();  
    db2.close();  
}
```

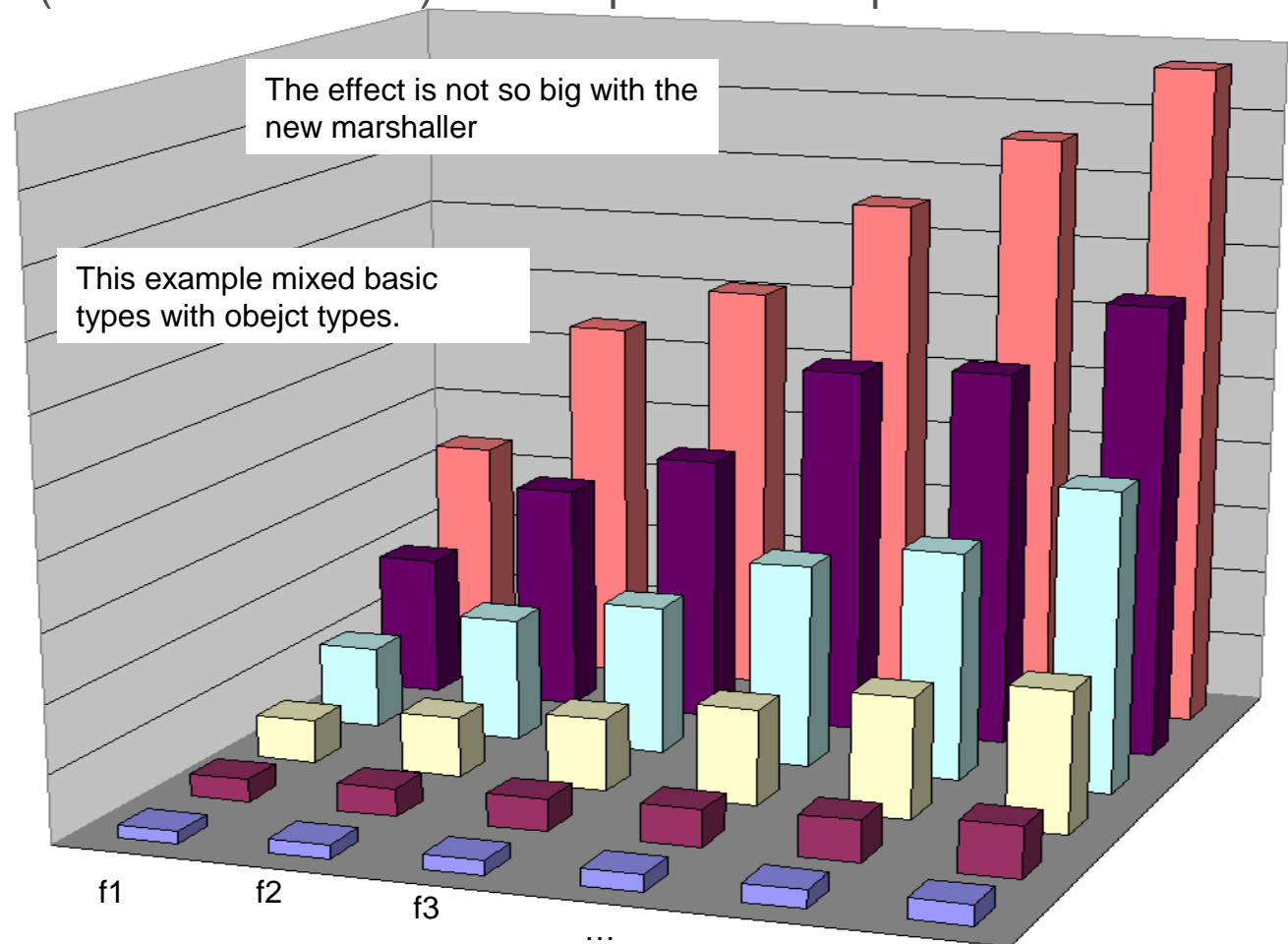
DESIGN / Separate Logic OBJECT / Avoid Fields

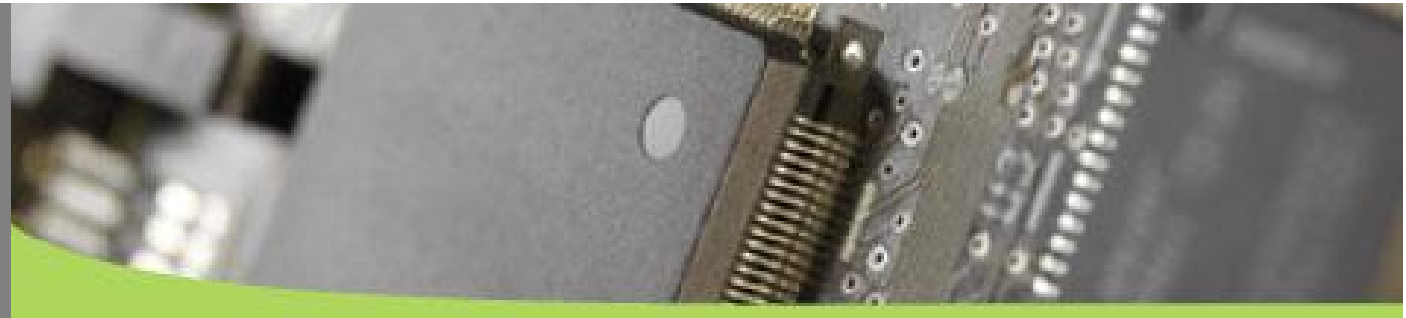
- | Try to think if you should separate persistent classes from business logic classes
- | Avoiding fields or better **mark them as transient** means smaller database and faster instantiation
- | For example: Let the original object „Person“ have a method that creates „PersonDTO“ by copying only the relevant fields out



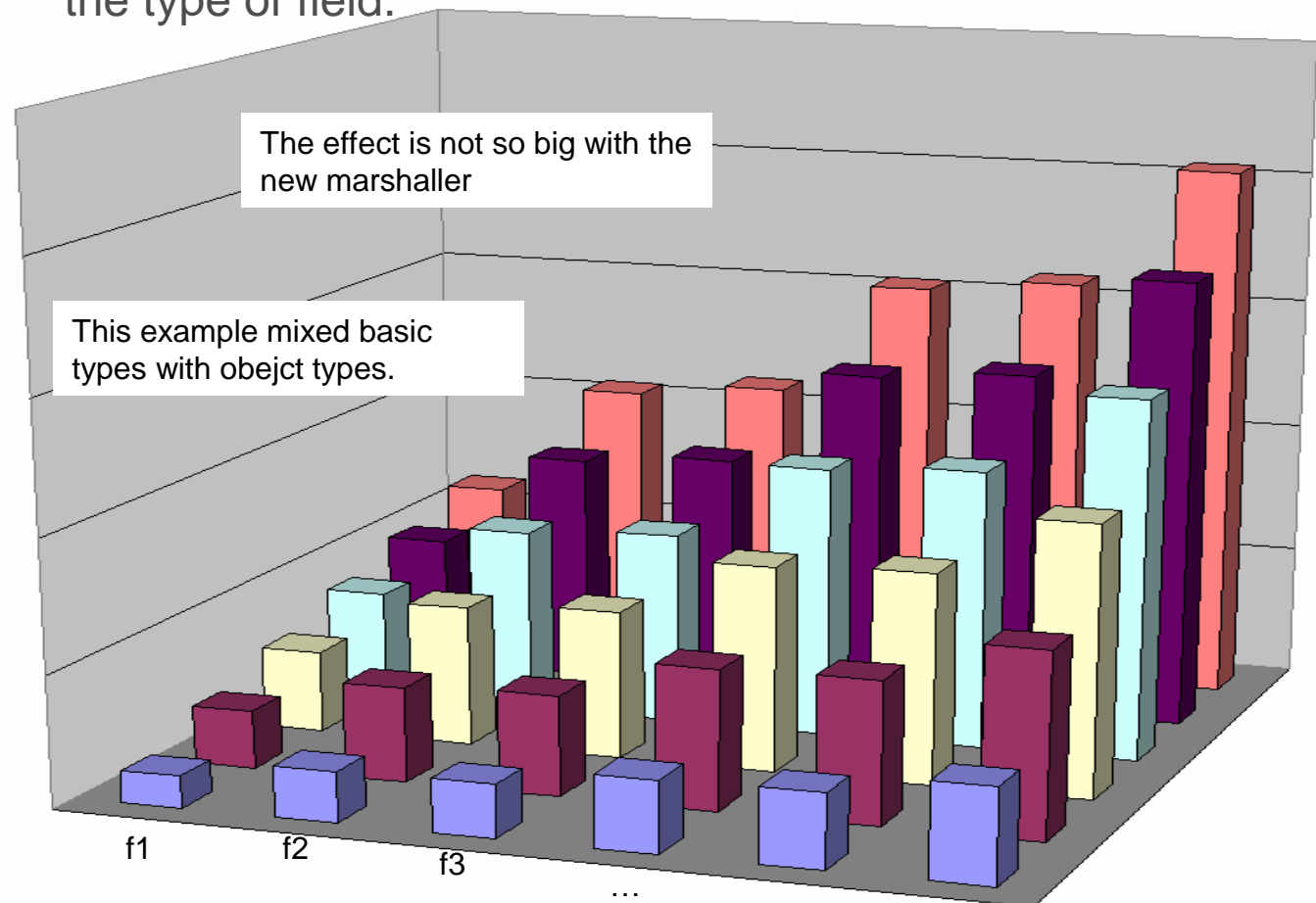


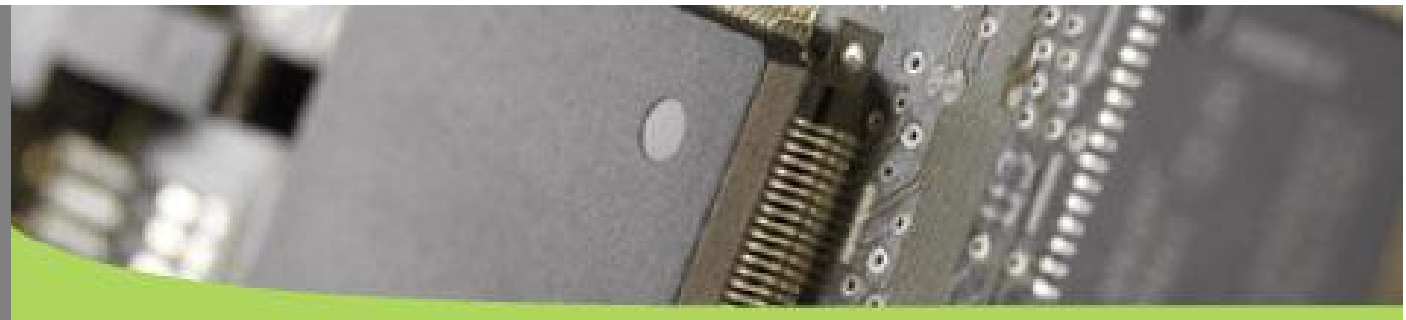
Each field slows down 17,9% speed in the average (min 5% max 37%) in this specific example





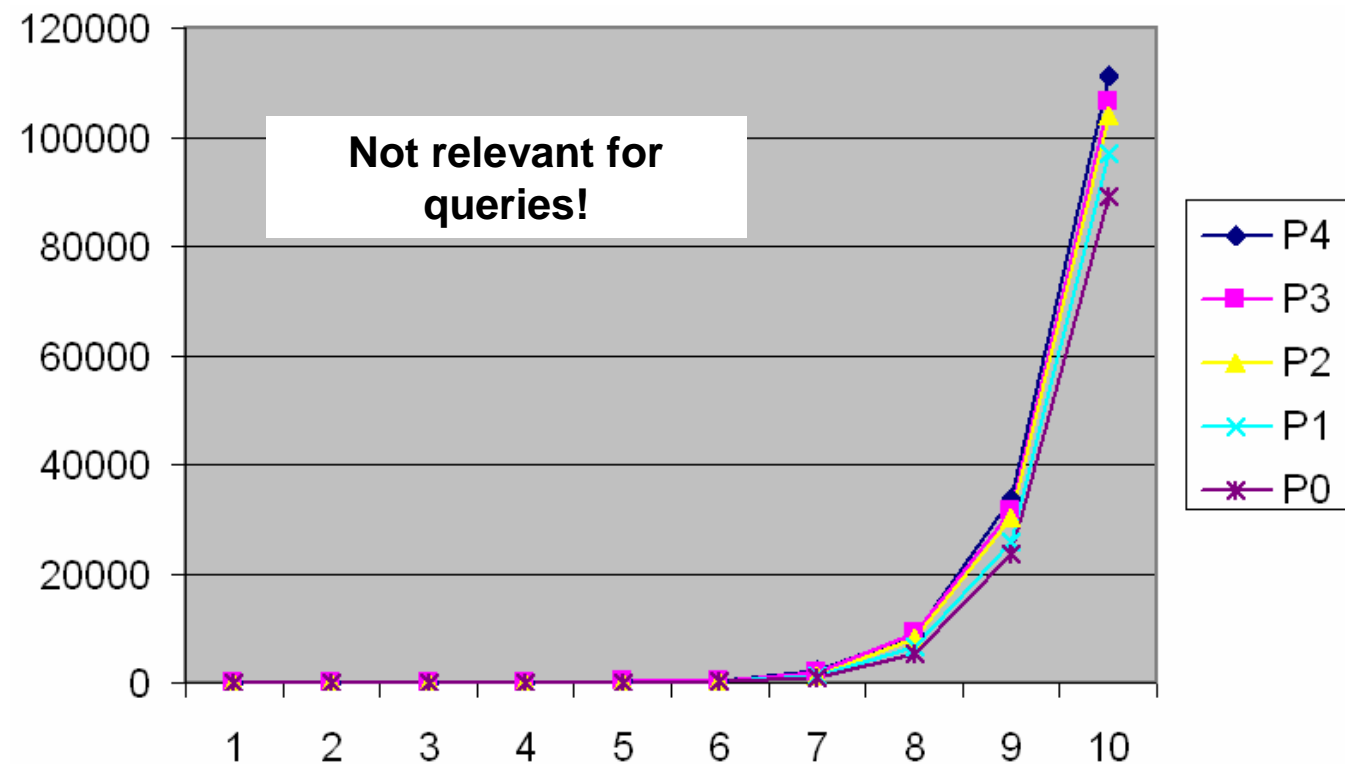
Each field increases filesize by 20% in the average for one class (min 5% max 42%) but this highly depends on the type of field.





DESIGN / Avoid Interheritance

Very very big variance but adding one level of inheritance costs some performance in average.

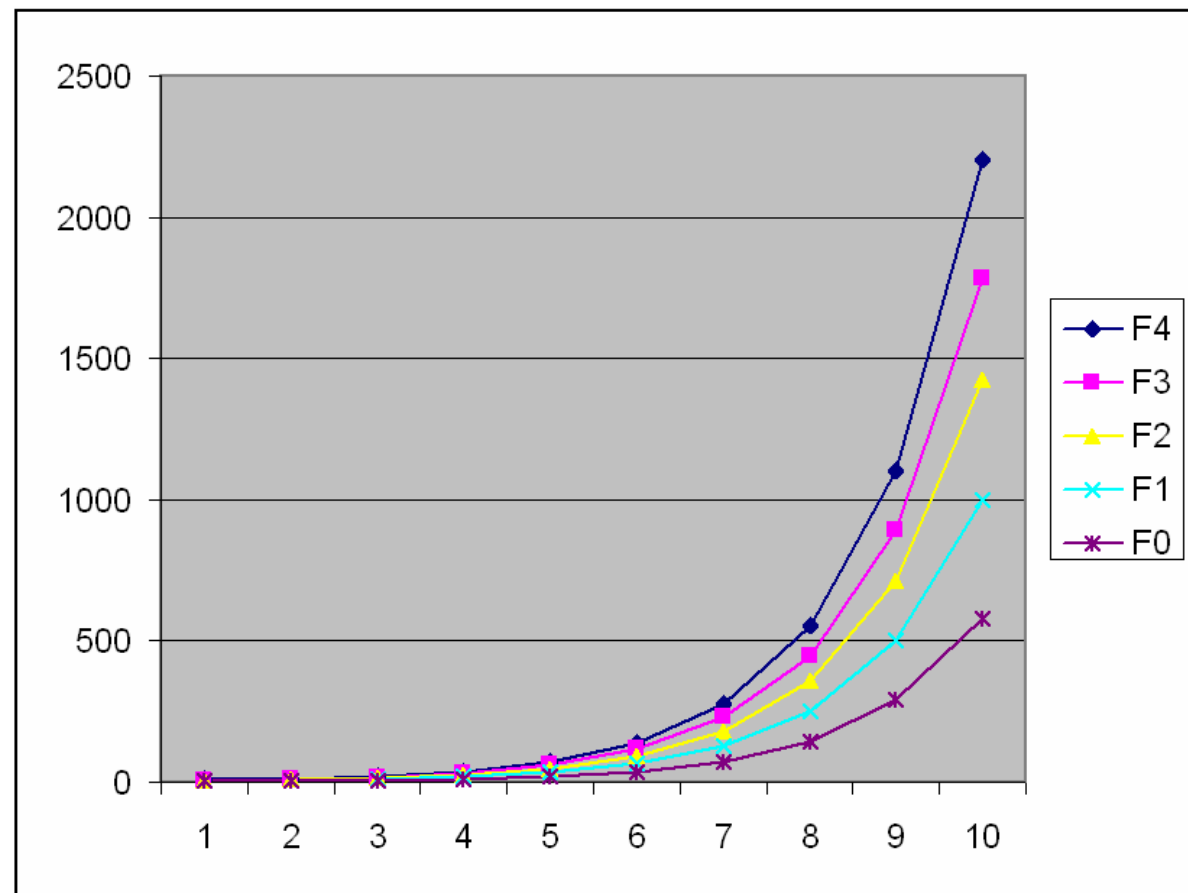




- INTRODUCTION
- DESIGN
- OBJECT
- SEARCH
- CASCADE
- CONFIGURATION
- REFERENCES
- FILE
- SUMMARY

DESIGN / Avoid Interhritance

Filesize increases





Typesafe

| Declare fields typesafe where possible.

not type safe:

```
class Foo{
    Object bar;
    Object fly;
    Object high;
}
```

type safe:

```
class Foo{
    int bar;
    String fly;
    FooBar high;
}
```

Use db4o Collections

| Only declare the interfaces in your parent classes:

| `java.util.List`

| `System.Collections.UList`

| Use the following in your code:

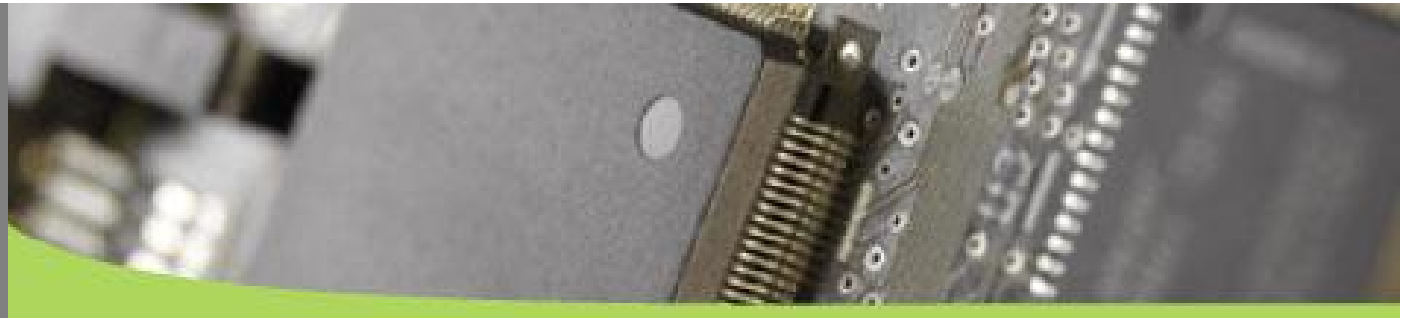
| `new ArrayList` and

| `ExtObjectContainer#collections().newLinkedList()`

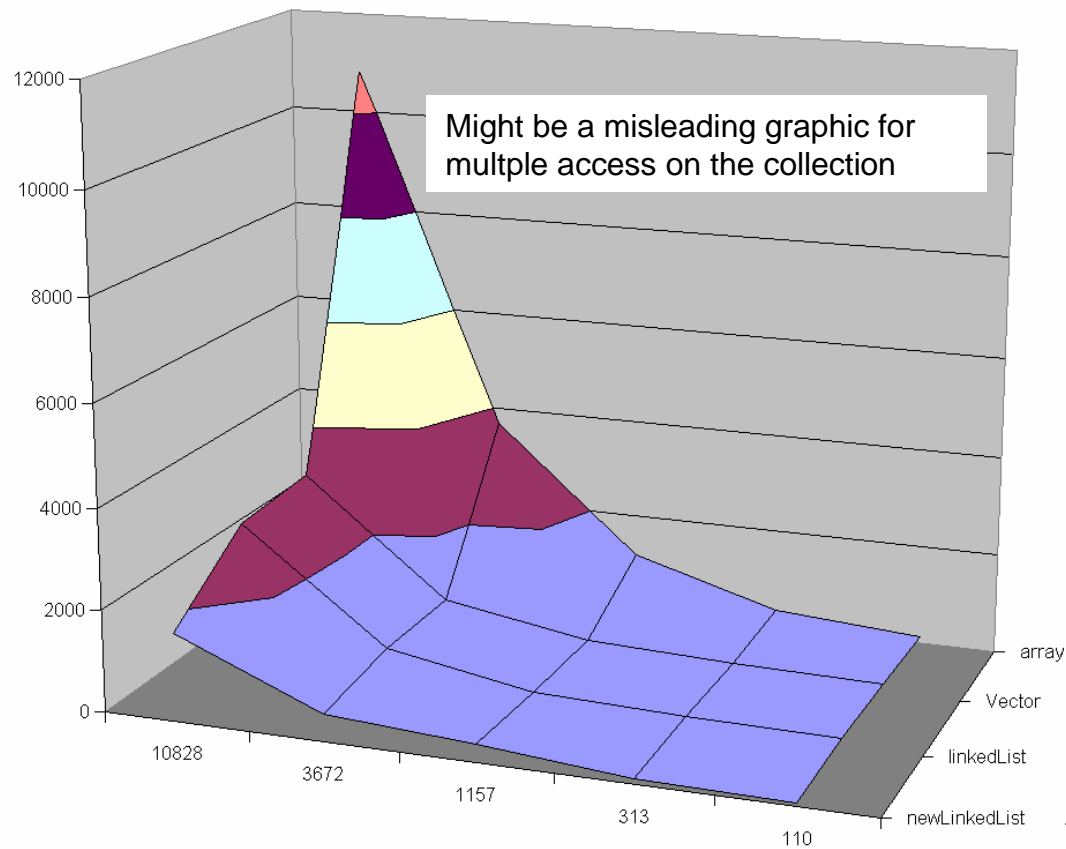
| `ExtObjectContainer#collections().newHashMap()`

| 2006 will see transparent activation!

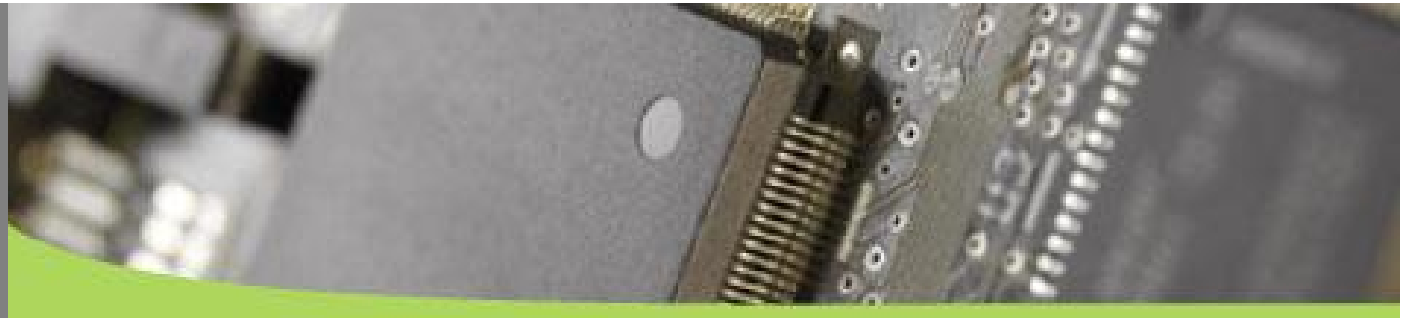
| it will optimize more and deliver only the objects necessary



Db4o linked list is mostly at least 50% faster then other collections (Vector, linkedList or arrays)

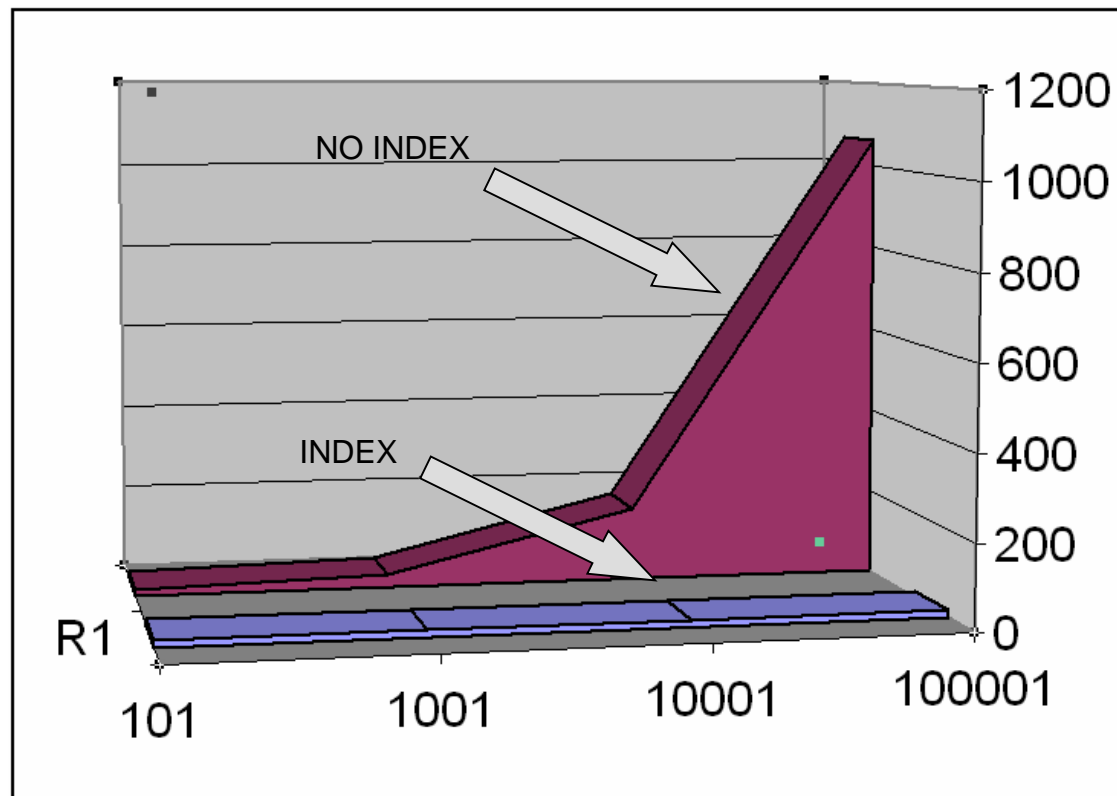


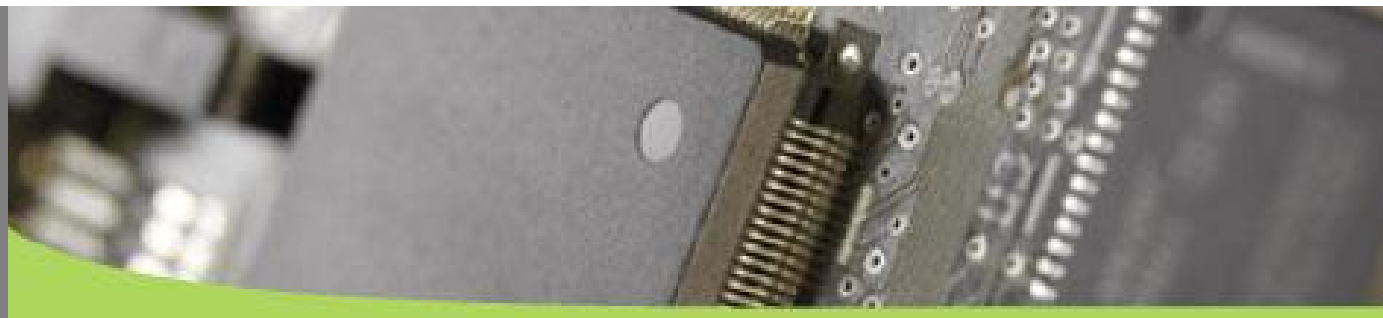
But it needs substantially more initial initialization time!



Use Index on fields that will be searched

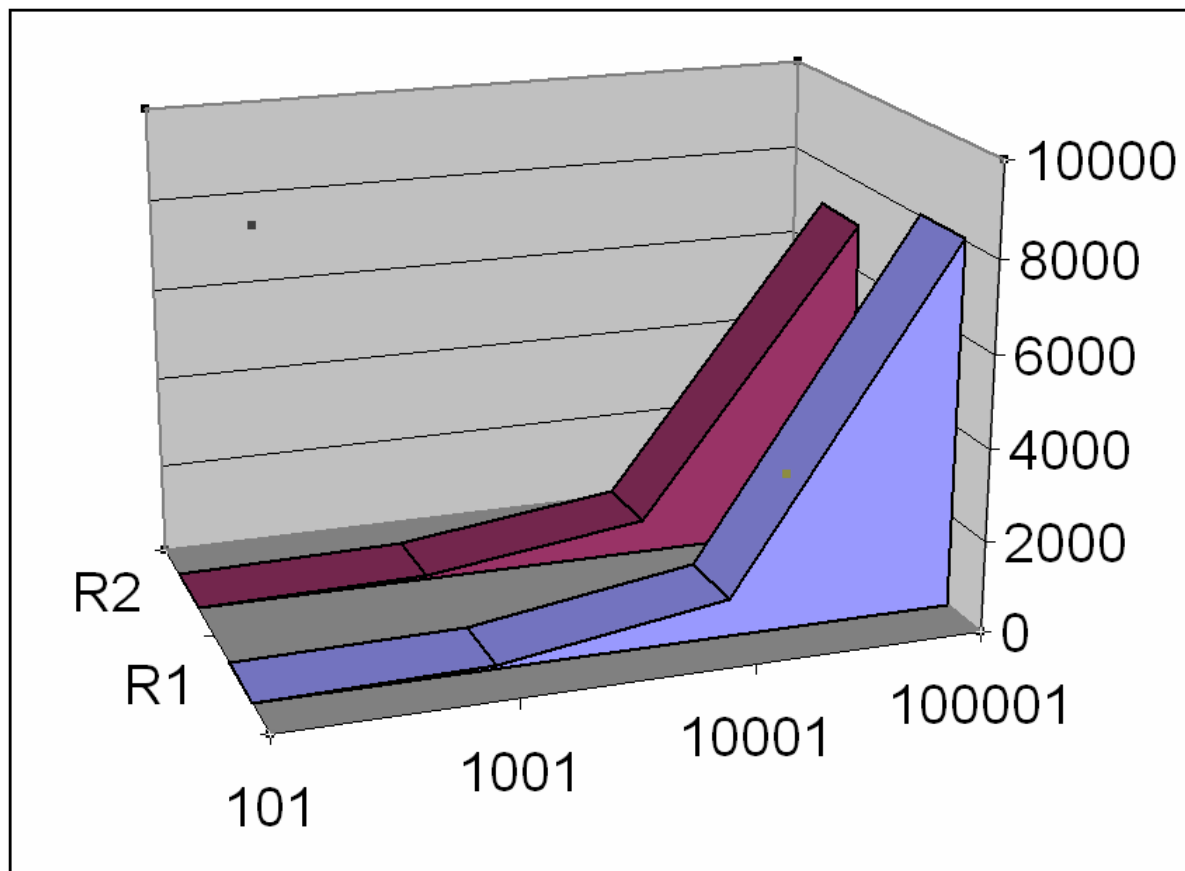
- | Search one field in 1000, 10000, 100000, ... classes
- | Index simply makes log faster





One Index makes filesize about 9-12% bigger for one class

- INTRODUCTION
- DESIGN
- OBJECT
- SEARCH
- CASCADE
- CONFIGURATION
- REFERENCES
- FILE
- SUMMARY



Use class.field.field as index

- | Field indexes also work for class.field.field
- | Example:

```
Db4o.configure().objectClass(Address.class)
    .objectField(„str“).indexed(true);
```



Beispiel

```
Class Person {  
    field1...  
    Address ad;  
}
```

```
Class Address {  
    field1...  
    String Street str;  
    ...  
}
```

```
Query q = db.query();  
q.constrain(Person.class);  
q.descend("address").descend("street").  
constrain("Ocean Blvd.");
```

Don't search and+or / Optimize Native Queries

| Joining constraints with `#and()` and `#or()` is not perfectly optimized at this time. We will work on that still in 2006.
| If you can avoid `#and()` and `#or()` do so.

| It's clear that you should optimize queries for performance.

| The performance gain depends on the query

| The wiki will show some example and performance results...

BTrees

- | Btree settings affect RAM consumption versus performance
- | You can change:
 - | bTreeCachHeight
 - | bTreeNodeSize
- | Used for Class Indexes
- | Will be used for Field indexes, Freespace Management, Collections and Collection Indexes in later versions
- | First polepos results show mixed results.
- | A bigger node size will improve performance mostly

Activation (update / delete issues)

| Activation can be a bottleneck. For now there are three viable strategies:

1. constant depth 5 as is, configurable to more or less
2. **activation depth to zero**, only activate manually when you really need an object
3. activate partial graphs fully in one go, using `#activate(object, Integer.MAX_VALUE)`

| Use db4o Collections to avoid activation

`newHashMap` / `newLinkedList`

| Use Transparent activation (completed second half of 2006) will activate when objects are accessed through public or package methods.

Does Blocksize have a reasonable performance effect?

```
Db4o.configuration().blockSize(16)
```

The standard setting is 1 allowing for a maximum database file size of 2GB. This value can be increased up to 127 (=254 GB) to allow larger database files, although some space will be lost to padding because the size of some stored objects will not be an exact multiple of the block size.

Let's run polepos with blocksize 1, 8, 32, 64 and 127

First results:

Blocksize should be the highest size needed to account for the anticipated number of objects stored.

8 is a good setting, since it corresponds to internal pointer length.

db4objects

INTRODUCTION

DESIGN

OBJECT

SEARCH

CASCADE

CONFIGURATION

REFERENCES

FILE

SUMMARY

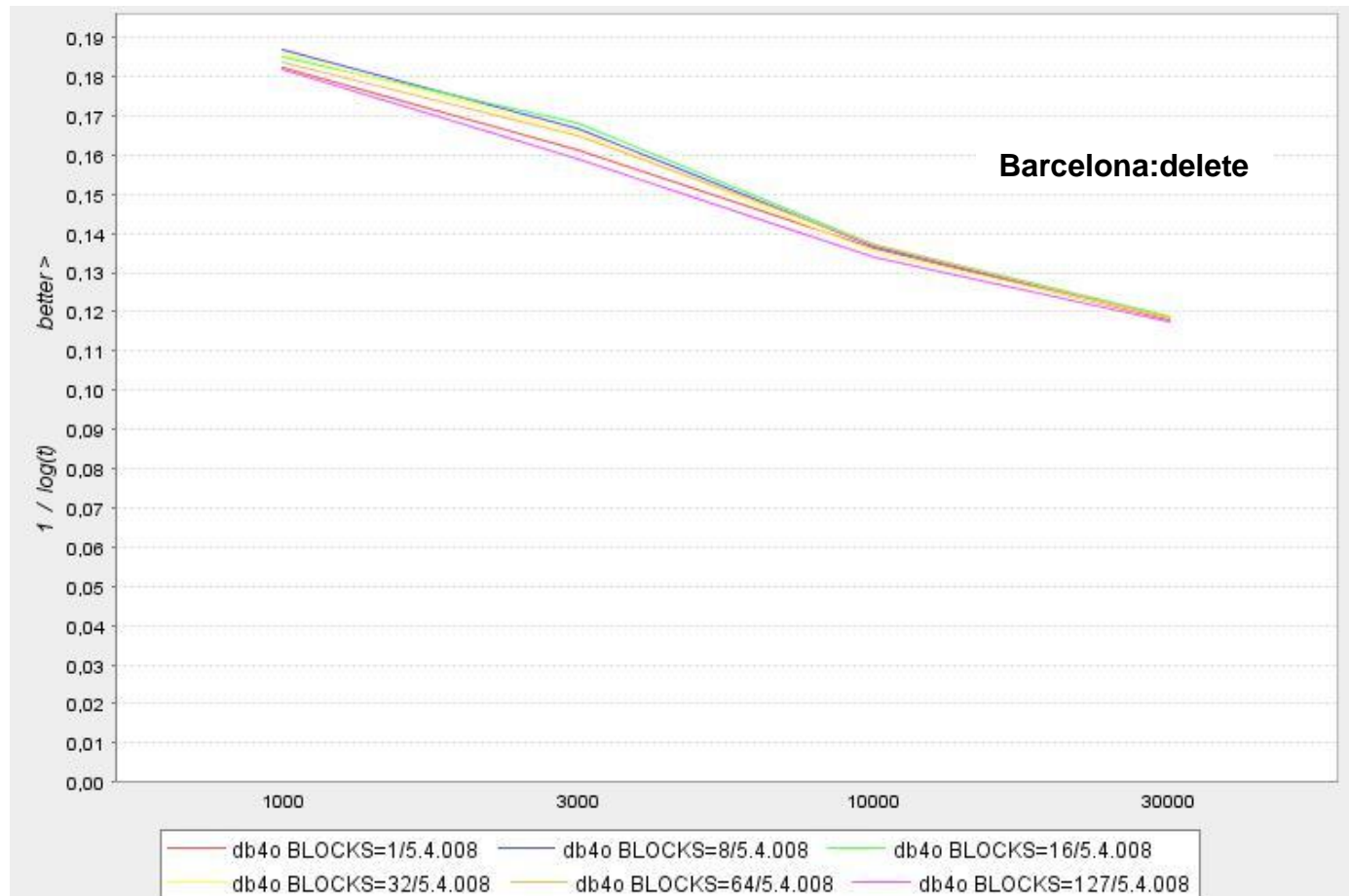


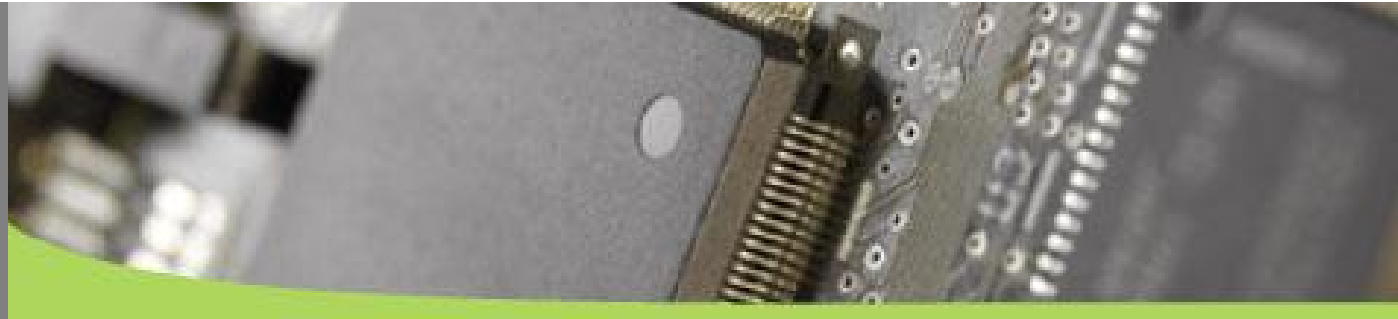
Melbourne	writes, reads and deletes unstructured flat objects of one kind in bulk mode
write	no effect
read	avoid blocksize 1
read_hot	8 or 16 could be better
delete	no effect
Sepang	writes, reads and then deletes an object tree
write	no effect
read	no effect
read_hot	64 is running crazy (Object trees (5-10) depth might like blocksize 64))
delete	no effect
Bahrain	write, query, update and delete simple flat objects individually
write	no effect
query_indexed_string	(avoid 64) possible maverick
query_indexed_int	(avoid 32) possible maverick
update	(avoid blocksize 1) possible maverick
delete	no effect
Imola	retrieves objects by native id
retrieve	no effect
Barcelona	writes, reads, queries and deletes objects with a 5 level inheritance structure
write	no effect
read	(avoid blocksize 1) possible maverick
query	(avoid blocksize 1) possible maverick
delete	no effect



Result

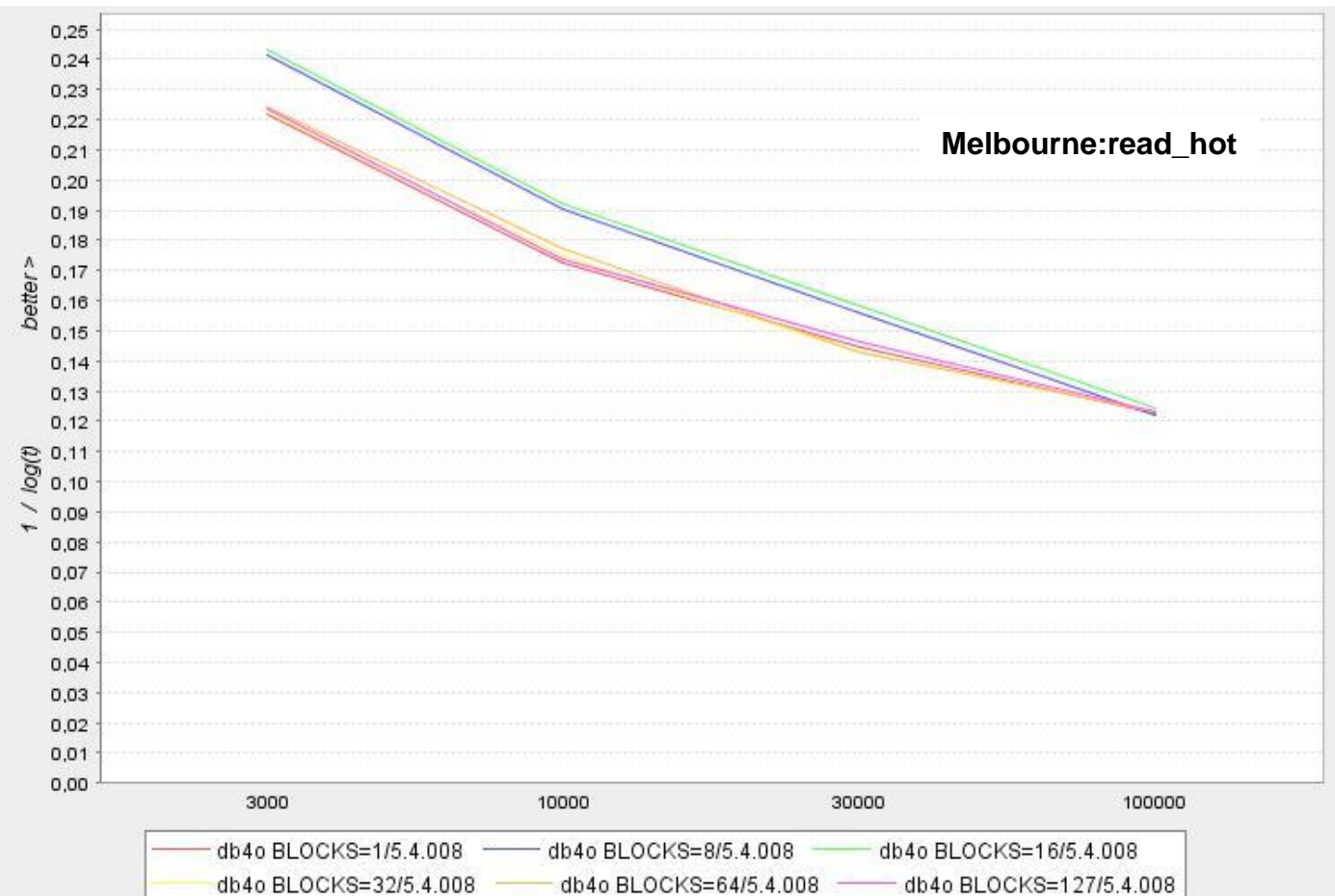
| Blocksize does really not have a big effect! Example:

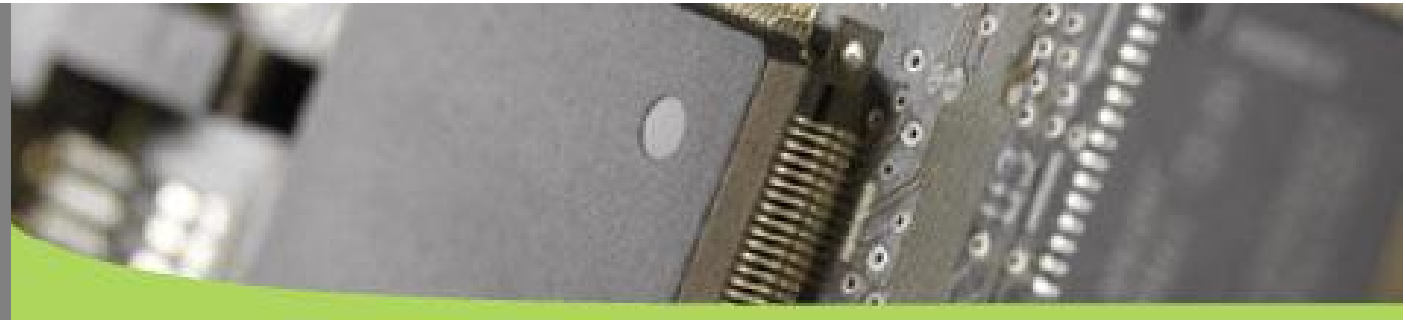




A recommended setting for large database files is 8, since internal pointers have this length.

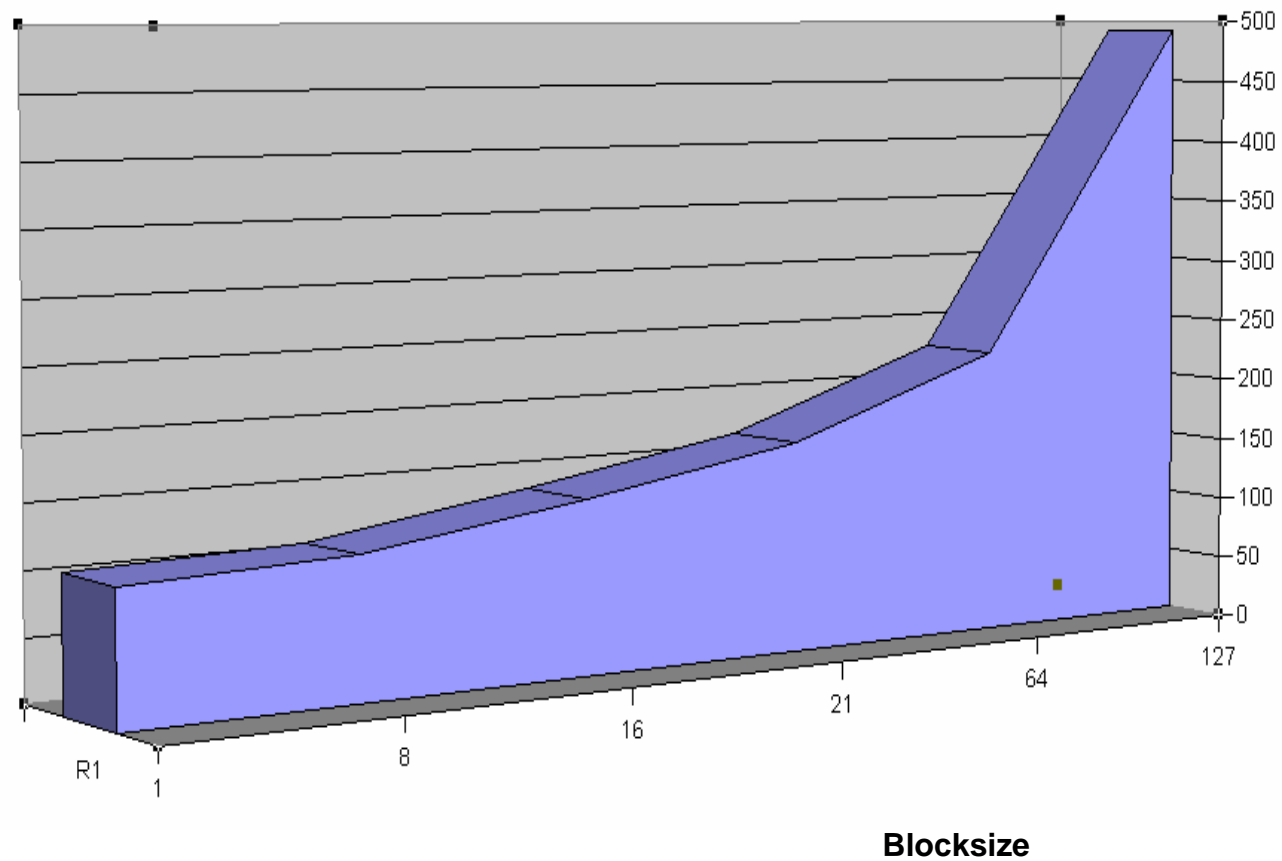
Mostly 8 and 16 are on the top (but the difference is small!)






But of course the filesize changes...

- INTRODUCTION
- DESIGN
- OBJECT
- SEARCH
- CASCADE
- CONFIGURATION
- REFERENCES
- FILE
- SUMMARY





```
callConstructors()  
testConstructors()  
detectSchemaChanges()
```

| These tuning features are for a production environment

| They are only relevant during the startup phase!

Freespace Management

- | `Db4o.configure().freespace().discardSmallerThan(...);`

- | Default is zero.

- | Free slots have to be managed which costs time and RAM

- | The Byte Argument sets the level when slots smaller are discarded

- | Increasing the value gains performance but increases the filesize

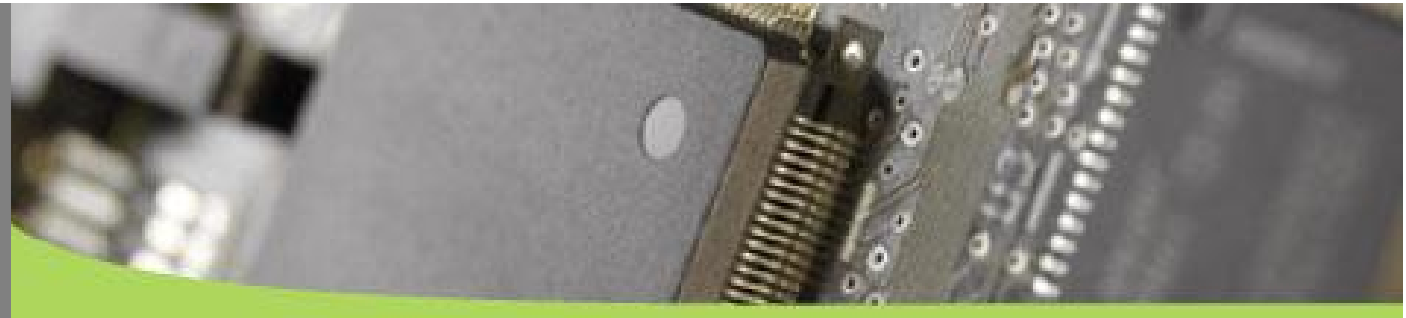
- | According to Carl, everything bigger then 50 will be a fiasco

`weakReferences()`

`weakReferenceCollectionInterval()`

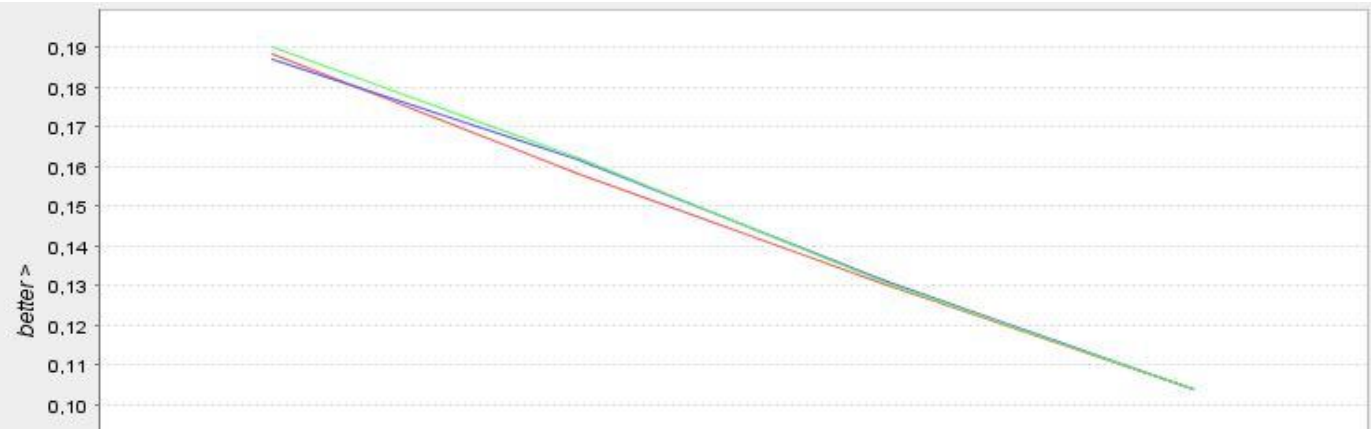
`purge()`

- | If you just write in one specific component and don't search, you can turn `weakReferences` off
- | Test the `weakReferenceCollectionInterval` to determine the best value
- | If you really need the `weakReference Management System` but not for some objects, the you can `purge` the object in your code.



Results on Polepos

weakReferenceCollectionInterval()



It's clear that a larger collection time needs more RAM

It shows that 1000 is a good setting

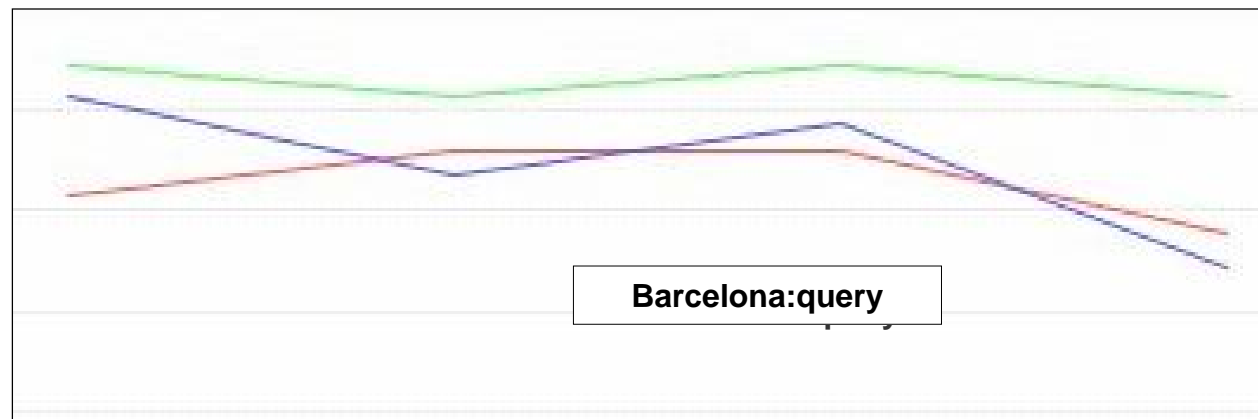
The additional RAM amount for polepos not notable (Some 10-20 MB).

In general, there is no big performance effect if you change the weak reference collection interval (e.g. from 100 to 1,000 to 10,000 in polepos).



But it can be useful to test...
to get the last 1% performance out of your system ...

- INTRODUCTION
- DESIGN
- OBJECT
- SEARCH
- CASCADE
- CONFIGURATION
- REFERENCES
- FILE
- SUMMARY



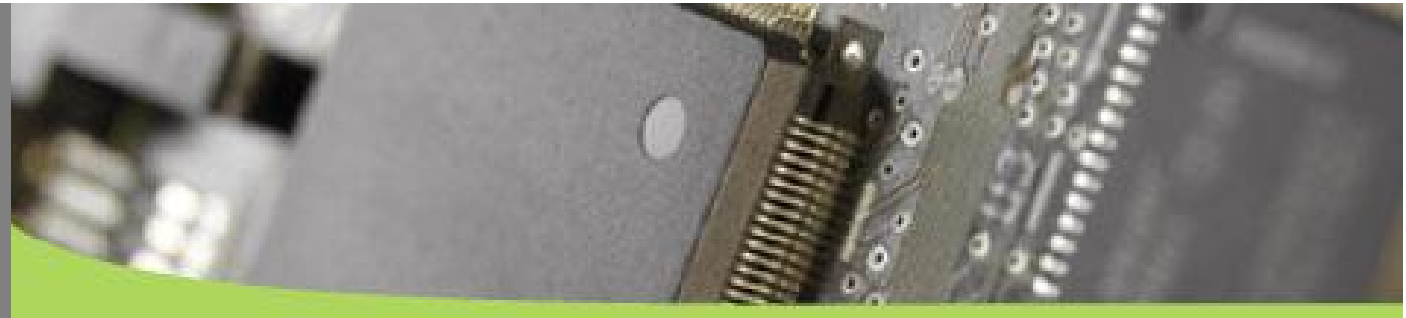
FILE: native Adapter

| (A) Write a new native file adapter

```
| Db4o.configure().io( ...your  
FileAdapter()...)
```

| Native code sample in the
db4o SVN.

| Can also be used to mirror to
two files



flushFileBuffers

(B) `Db4o.configure().flushFileBuffers()`

| File flushing: Turning off improves speed slightly, but is **no longer recommended**. db4o only does 4 flushes on commit and the performance loss is negligible.

| Can jeopardize ACID behaviour!

| In more than 80% of the cases, the gain is smaller than 6%

t [time in ms]	selects:100 objects:1000	selects:100 objects:3000	selects:100 objects:10000	selects:100 objects:30000
db4o 5.4.008	33	77	240	729
db4o noflush 5.4.008	31	76	243	901

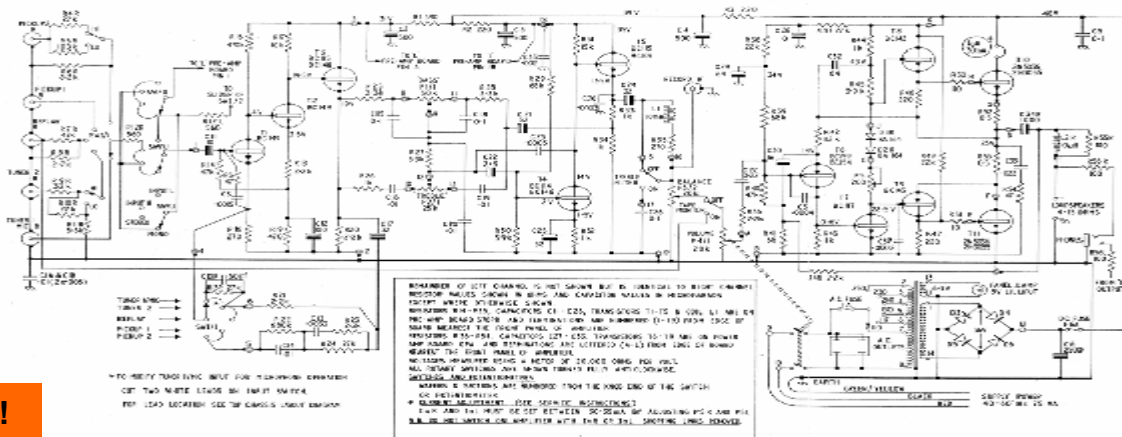




- INTRODUCTION
- DESIGN
- OBJECT
- SEARCH
- CASCADE
- CONFIGURATION
- REFERENCES
- FILE
- SUMMARY

Summary

The global advice for the 24 db4o performance tuning tricks follows this algorithm:



But!

Rules of thumb: Invest in **OO-Design** is easy and gives huge performance boosts.

Writing **tests** for your specific data, RAM and access / operation modes and remaining configurations with db4o is easy.



Performance is most important accordingly to the user survey. So if a customer thinks db4o is too slow in this specific case, then **db4o** accepts code to **optimize** against it!

Please visit the catalog:
http://developer.db4o.com/ProjectSpaces/view.aspx/Performance_Catalog/Performance_Catalog

Thanks for listening!