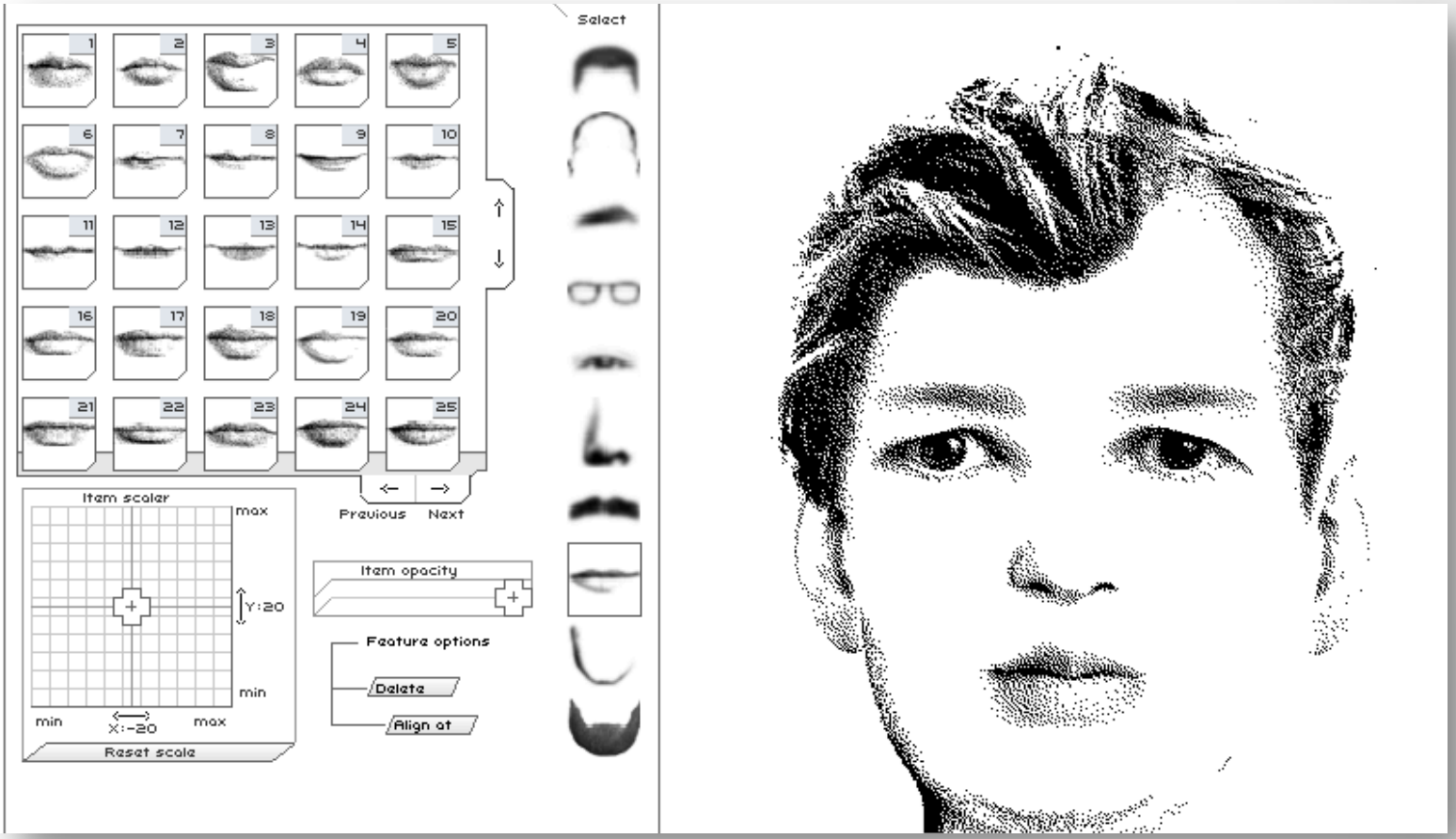
Language Design is Trade-Offs

andrey.breslav@JetBRAINS.com

Simply Universal

Full of Shortcuts

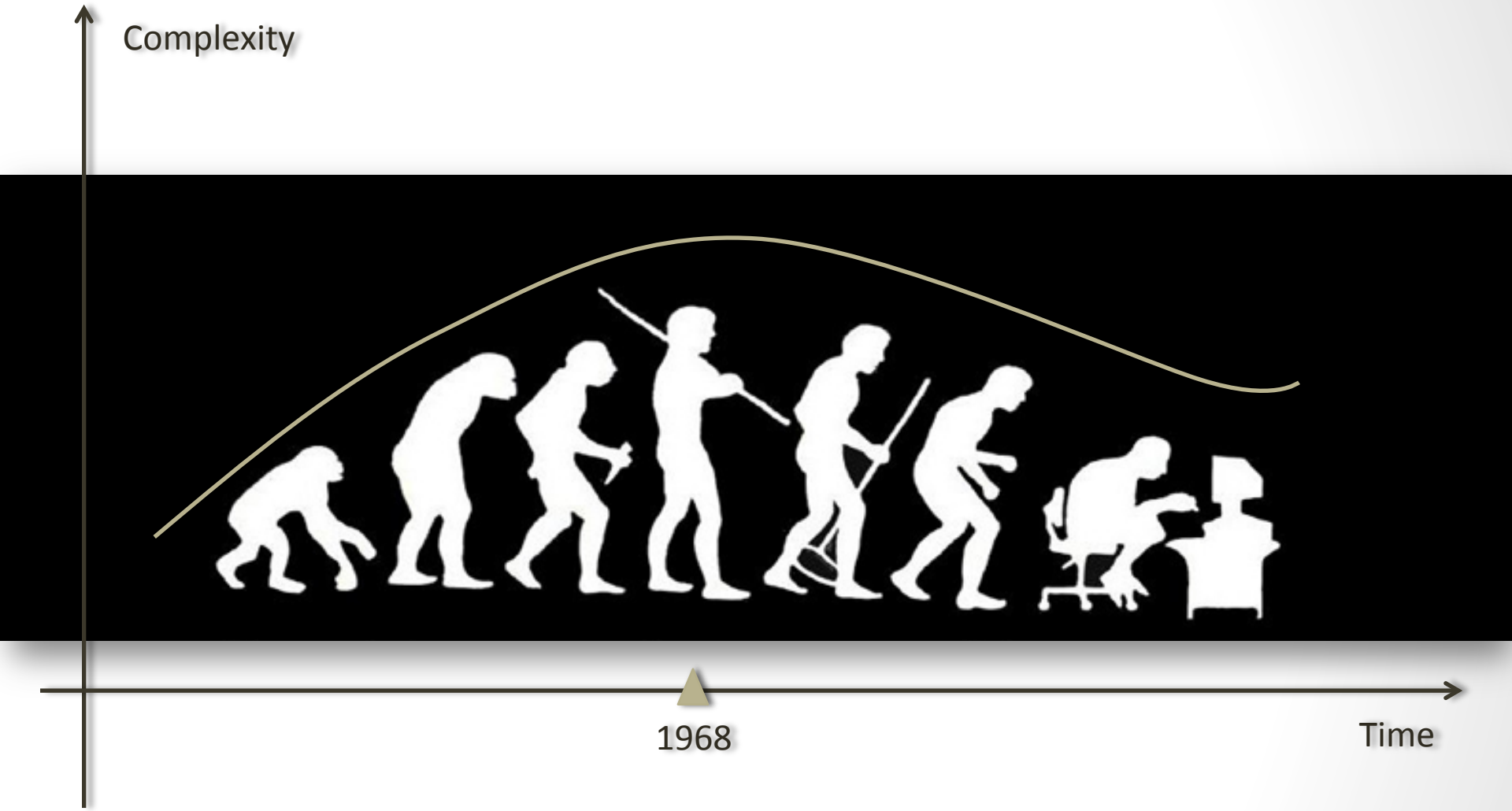
Simply Universal



Full of Shortcuts

Elegance VS Convenience

Trade-Off #0


Complexity
1968
Time

Cool

Vs

Practical

Clever

Readable

Simple

Fast

VS

Shiny New

Good Old

Ground-Breaking

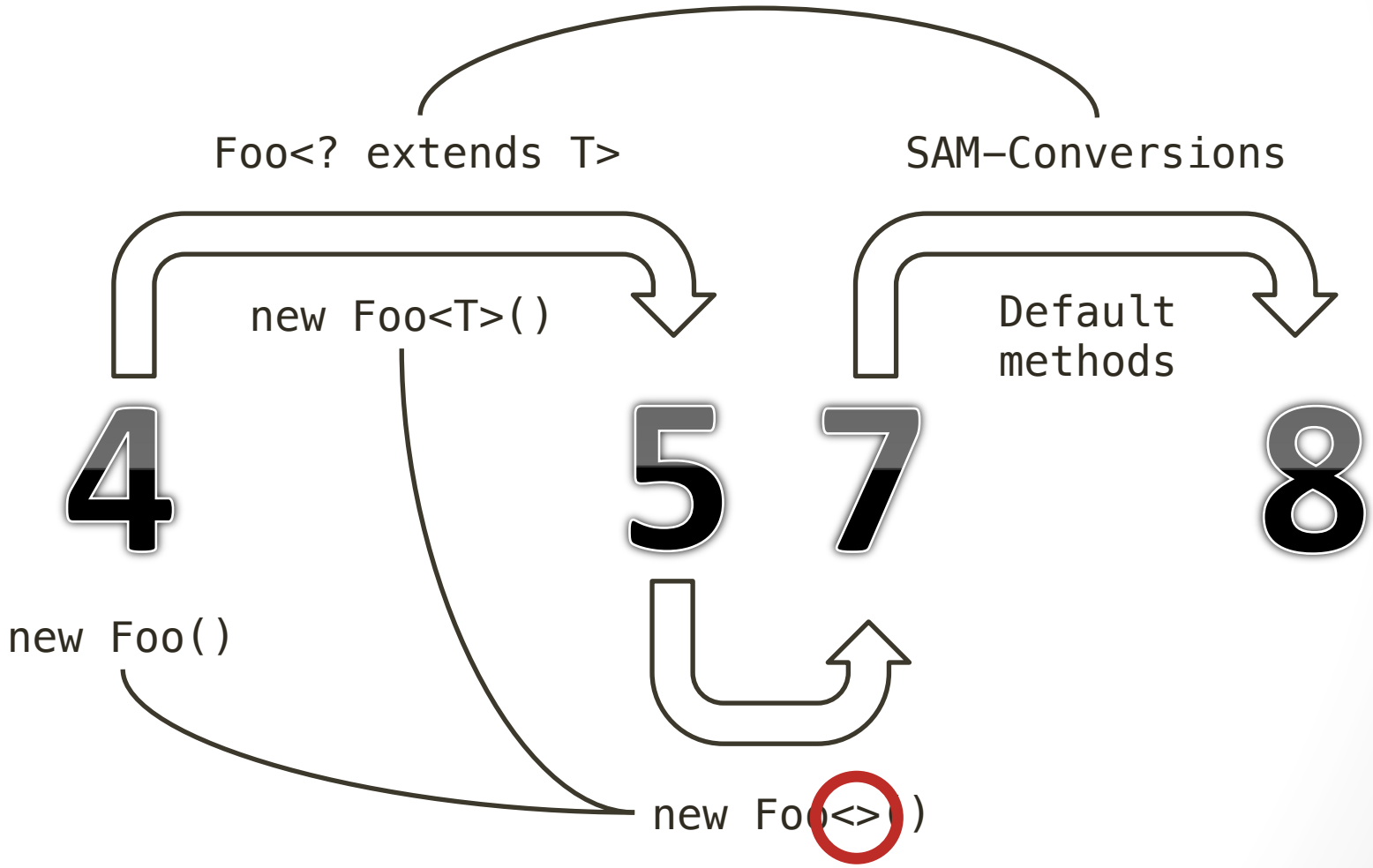
Compatible

VS

Shiny New

Good Old

As Java Evolved…



Lambdas & SAM-Conversions

```
foo(g -> new SubBaz(g));
```

```
void foo(Function<? super Bar, ? extends Baz> g)
```

SAM-type

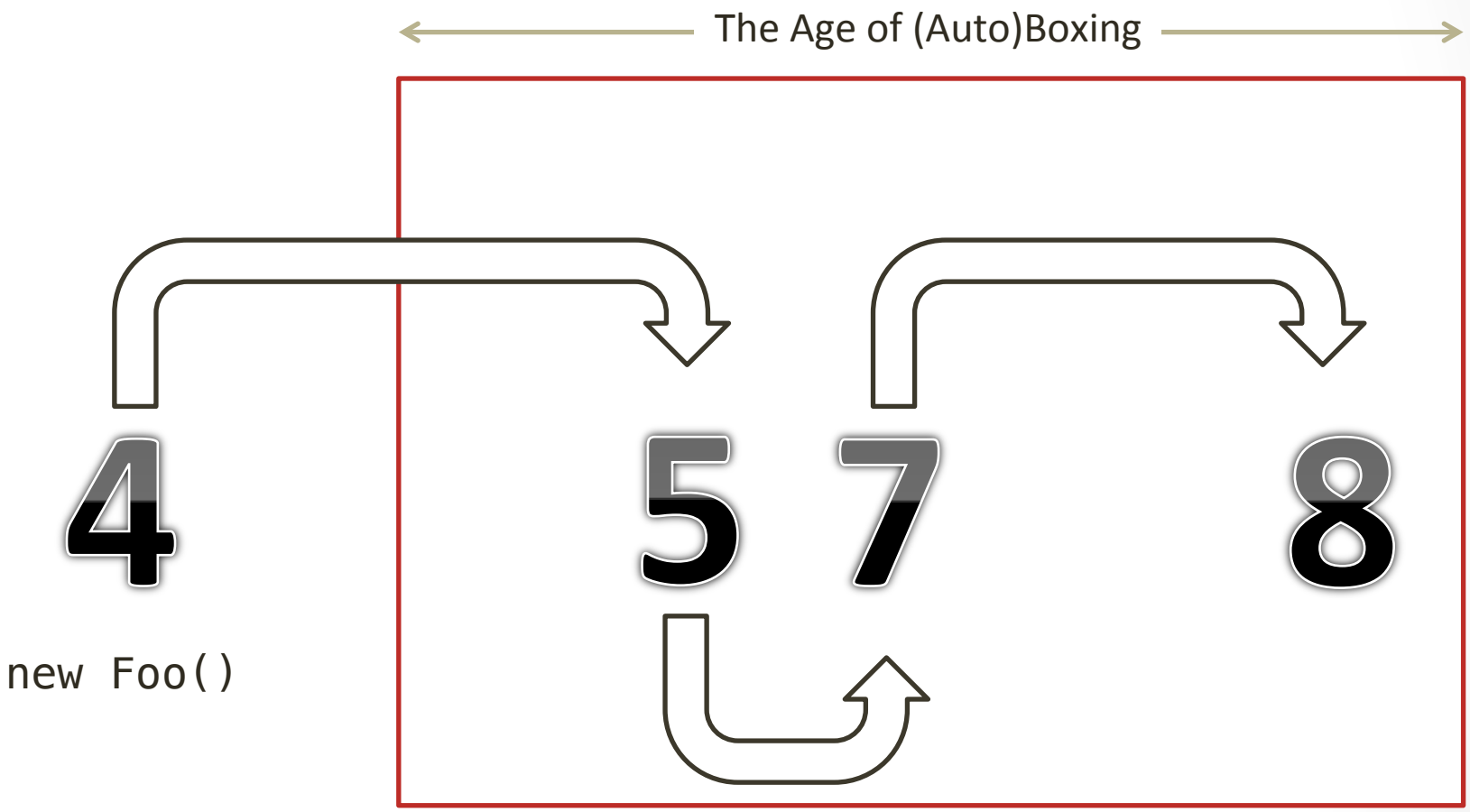

Compatibility
is a drag,
son…

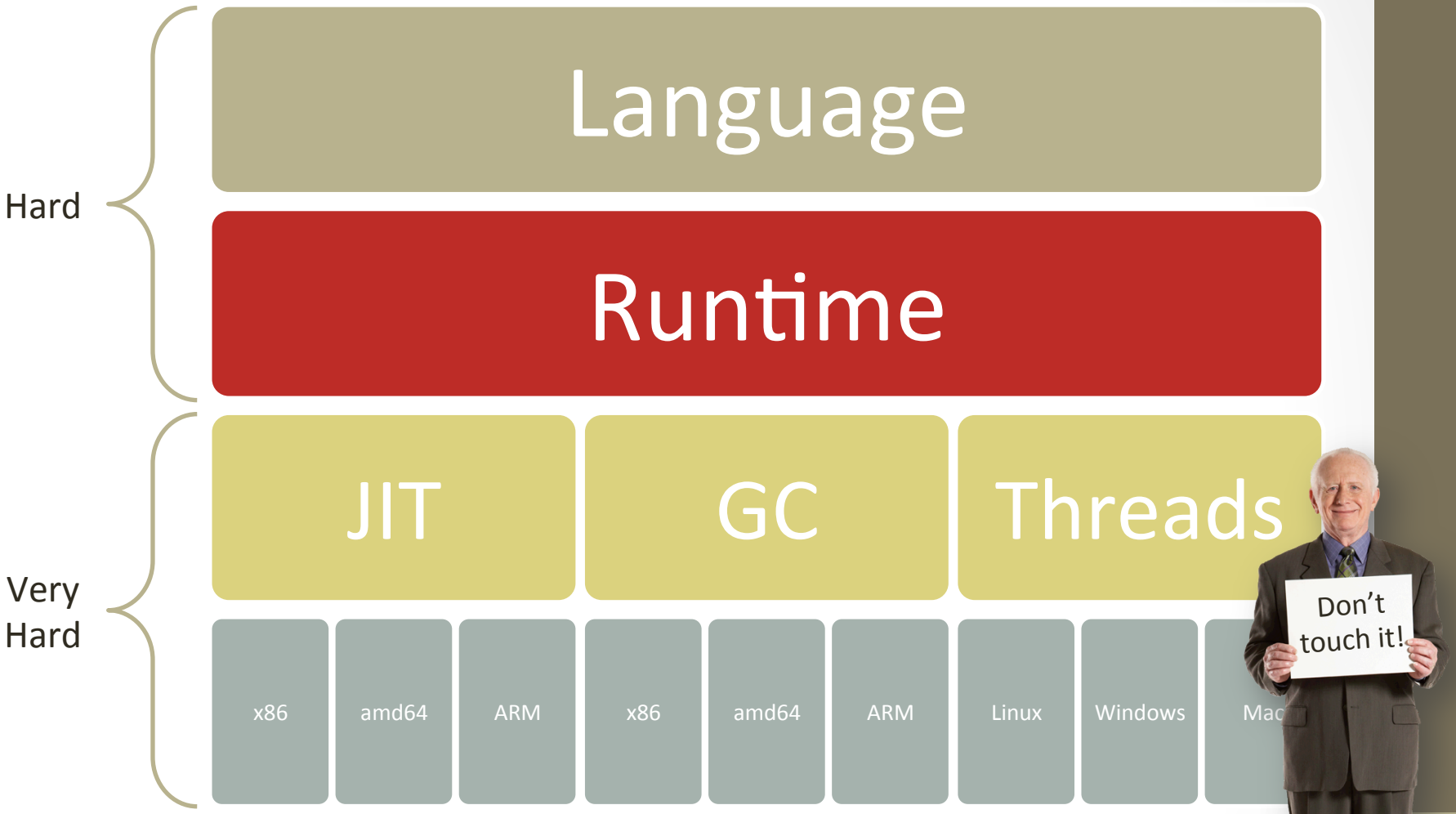
VS

Simple

Fast

Boxing for Primitives




Hard
Language
Runtime
Very Hard
JIT
GC
Threads
x86
amd64
ARM
x86
amd64
ARM
Linux
Windows
Mac
Don't touch it!

VS

Ground-Breaking

Compatible

JDK Collections

```
interface List<E> {

    E get(int index);

    E set(int index, E value);

}
```

Read-only

```
/** read-only */
List<Foo> getFoos() {
    return unmodifiableList(l);
}
```

Errors

At Runtime

Cool Collections

```
interface List<E> {
   E get(int index);
}

interface MutableList<E>
              extends List<E> {
   E set(int index, E value);
}
```

Read-only

```
List<Foo> getFoos() {
    return l; // may be mutable
}
```

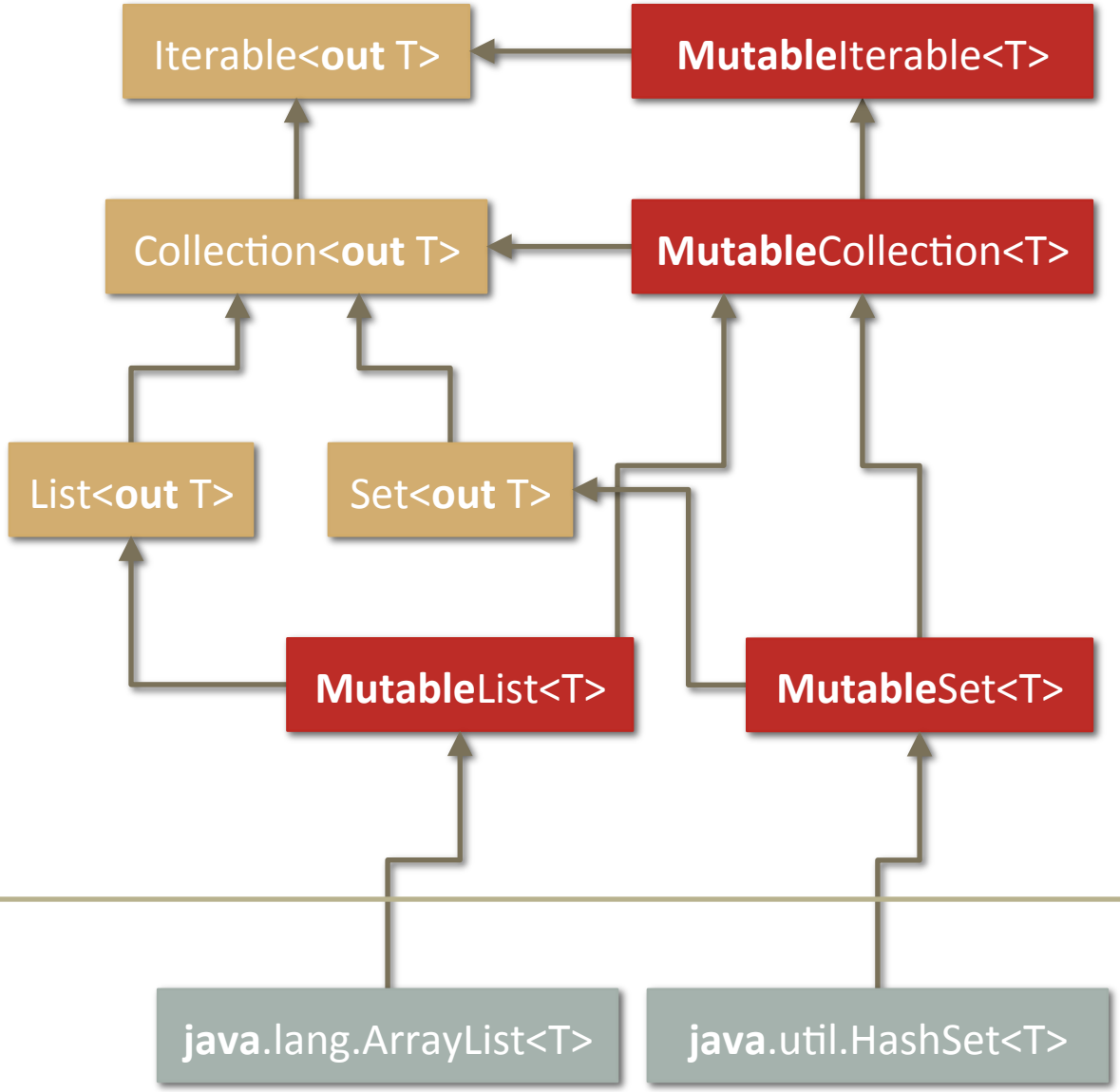
Errors

At Compile Time

Cool Collections	JDK Collections
Safe	Compatible
catch errors early self-documenting code	no wrapping/repackaging existing API's work
Intuitive	? extends Foo
`print(foos: List<Foo>)`	`print(List<? extends Foo> foos)`

Kotlin Collections

JDK Collections



Kotlin Collections =  & 

Safe

Compatible

Co-variant

Java

```
String s = null;

s.length();
```

Errors At Runtime

Kotlin

```
val s: String = null
s.length()
```

```
val s: String? = null
s.length()
```

Nullable type

Errors At Compile Time

Check and use

```
val s: String? = …
if (s != null) {
    s.length()
}
```

Check and exit

```
if (s == null) return

s.length()
```

Rock'n'Roll

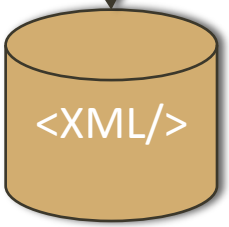
```
s?.length()
```

```
class Foo {
  String getName() { … }
}
```

Nullable or Not?

JAR File

@NotNull



VS

Dynamic

Static

Library

*Util class

Util class

```
Collections2.transform(
    Collections2.filter(
        list,
        (foo) -> foo.isValid()),
    (foo) -> new Bar(foo)));
```

No Util Class

```
scala.collection.Iterable[T]
```

declares 116 methods

VS

Ugly Interfaces

Ugly Code

People Need Extensions

Java
```
static char last(String s)
```

C#
```
static char last(string this)
```

Kotlin
```
fun String.last(): Char {
    return this[this.size - 1]
}
```

Usage
```
“abc”.last()
```

Access privates	No
Virtual	No
Owner	Package
Many Applicable	Overloading error
Member Clashes	Member wins

= static utility

```
File("...").
    appendBytes(data: jet.ByteArray) for java.io.File in k…   jet.Unit
    appendText(text: jet.String, encoding: java.nio.charset.Charset)
    appendText(text: jet.String, encoding: jet.String = ...) for java.io.File
    canonicalPath                                              jet.String
    copyTo(file: java.io.File, bufferSize: jet.Int = ...) …    jet.Long
    directory                                                  java.io.File
    equals(other: jet.Any?) for jet.Any? in jet                jet.Boolean
    extension                                                  jet.String
    forEachBlock(closure: (jet.ByteArray, jet.Int) -> jet.Unit)   jet…
    forEachLine(charset: jet.String = ..., closure: (jet.String) ->…
    hashCode() for jet.Any in kotlin                           jet.Int
Use ⇧⌘⏎ to syntactically correct your code after completing (balance parentheses etc.)
```

Modularity

Clean Code

IDE Support

VS

Theory

Practice

Java

```
new List<String>()
Collections.<String>emptyList()
```

Why different?

Scala

```
new List[String]()
emptyList[String]()
arr(0)
```

Why square brackets?

```
foo(bar<A, B>(x + 1))
```


```
foo(bar < a, b > (x + 1))
```

Java Familiar (from C++)
Sometimes ugly

Scala Always pretty
Unfamiliar

C# and Familiar
Kotlin Always* pretty

```
emptyList<String>()
```

```
* foo((bar < A), B > (x+1))
```

Practice is nicer than Theory

Language Design is Trade-Offs

kotlin.jetbrains.org

andrey.breslav@JetBRAINS.com