Generics: Past, Present and Future

@richardwarburto
@raoulUK

binarySearch(List<? extends Comparable<? super T>> list, T key)

Past

Present

Future

[thefacebook]

login register about

Email:

Password:

login

register

Welcome to Thefacebook!

[Welcome to Thefacebook]

Thefacebook is an online directory that connects people through social networks at colleges.

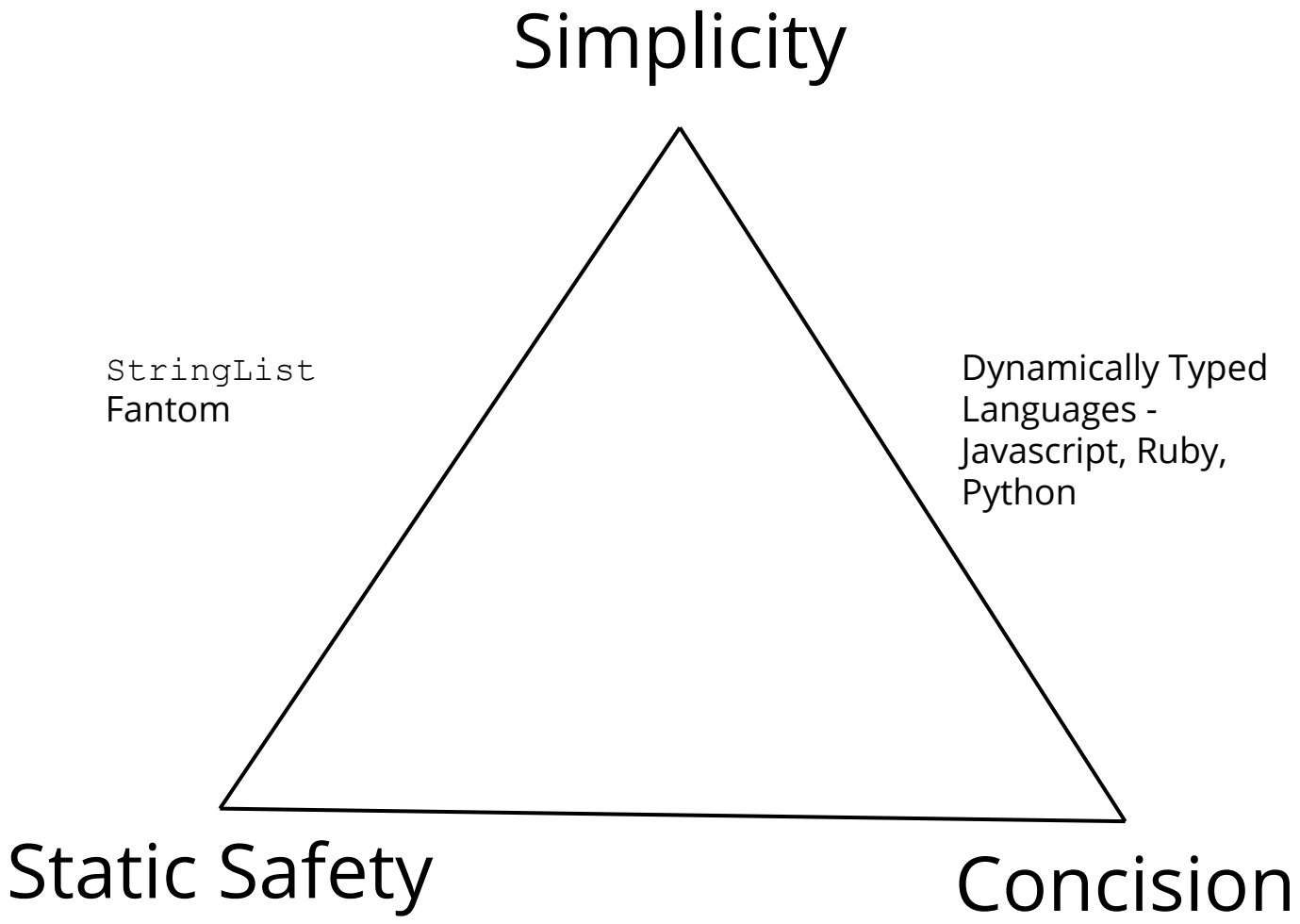
We have opened up Thefacebook for popular consumption at:

BC • Berkeley • Brown • BU • Chicago • Columbia • Cornell • Dartmouth • Duke
Emory • Florida • Georgetown • Harvard • Illinois • Michigan • Michigan State
MIT • Northeastern • Northwestern • NYU • Penn • Princeton • Rice • Stanford
Tulane • Tufts • UC Davis • UCLA • UC San Diego • UNC
UVA • WashU • Wellesley • Yale

Your facebook is limited to your own college or university.

... also generics are added to Java.

Yay!


Simplicity
StringList
Fantom
Dynamically Typed Languages - Javascript, Ruby, Python
Static Safety
Concision
Generics
Java, Scala, C#, C++
...

Past

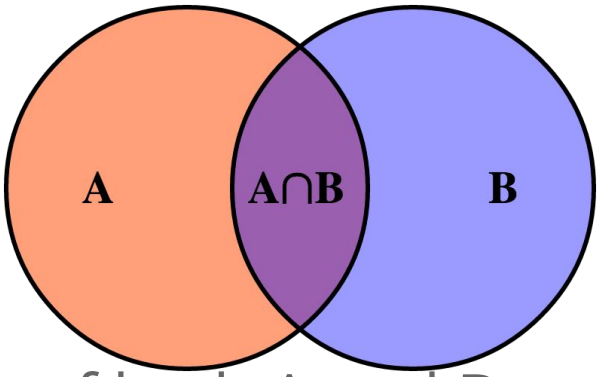
Present

Future

Intersection Types

Curiously Recurring Generics Pattern

Wildcards



Intersection

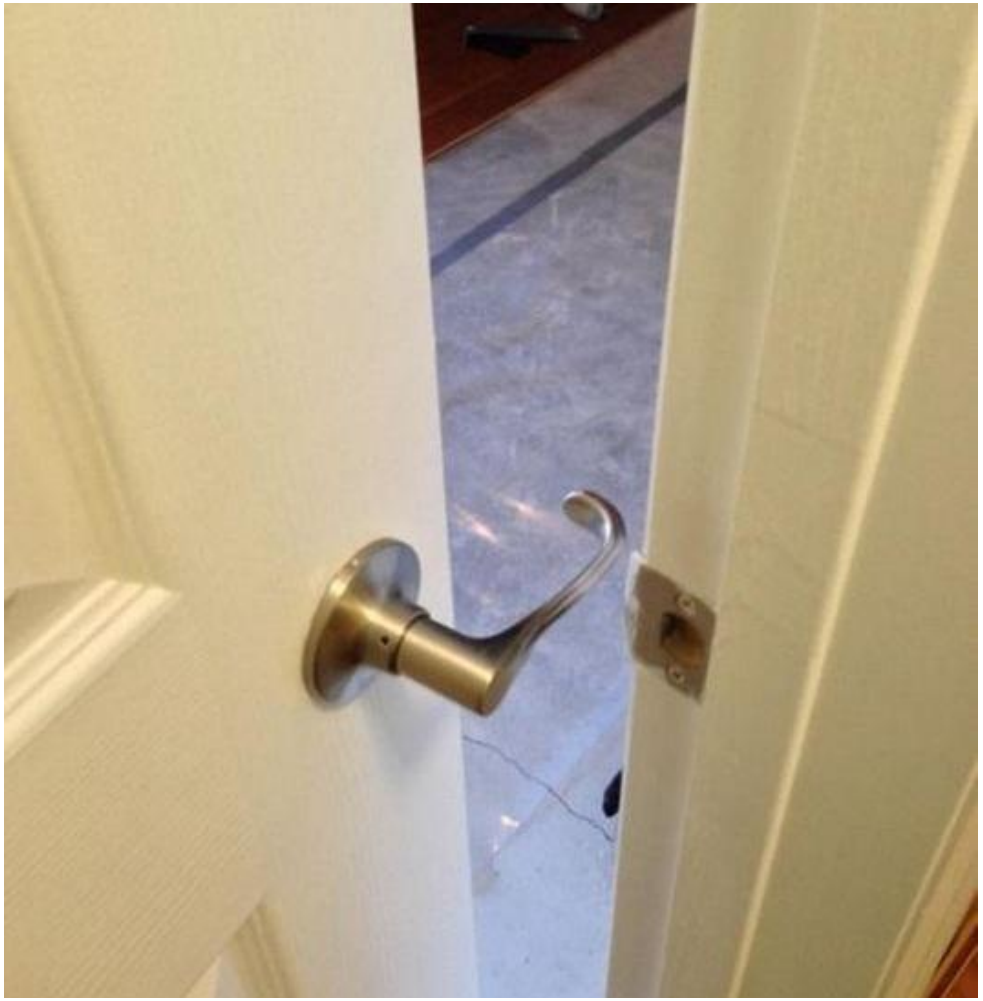
A ∩ B = elements has to be a member of both A and B

Intersection Type

<T extends A> = T has is a subtype of A

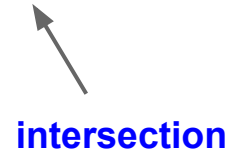
<T extends A **&** B> = T is a subtype of A **and** B

<T extends Object & Comparable<? super T>>
T max(Collection<? extends T> coll)



A Confusing Intersection Type

<T extends Object & Comparable<? super T>>



T max(Collection<? extends T> coll)

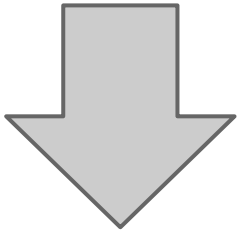
Signature pre-generics

```
public static Object max(Collection coll)
```

- `max` is stuck with this signature to preserve binary compatibility.
- Can only find the `max` if the objects are `Comparable`

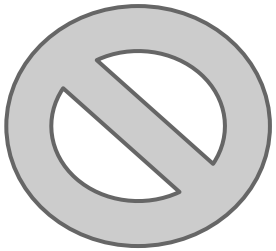
Type erasure

```
<T extends Comparable<? super T>>
T max(Collection<? extends T> coll)
```



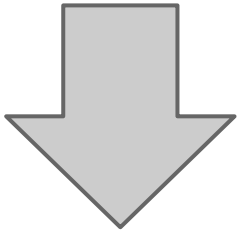
javac compilation

```
Comparable max(Collection coll)
```



Type erasure with intersection

```
<T extends Object & Comparable<? super T>>
T max(Collection<? extends T> coll)
```



javac compilation

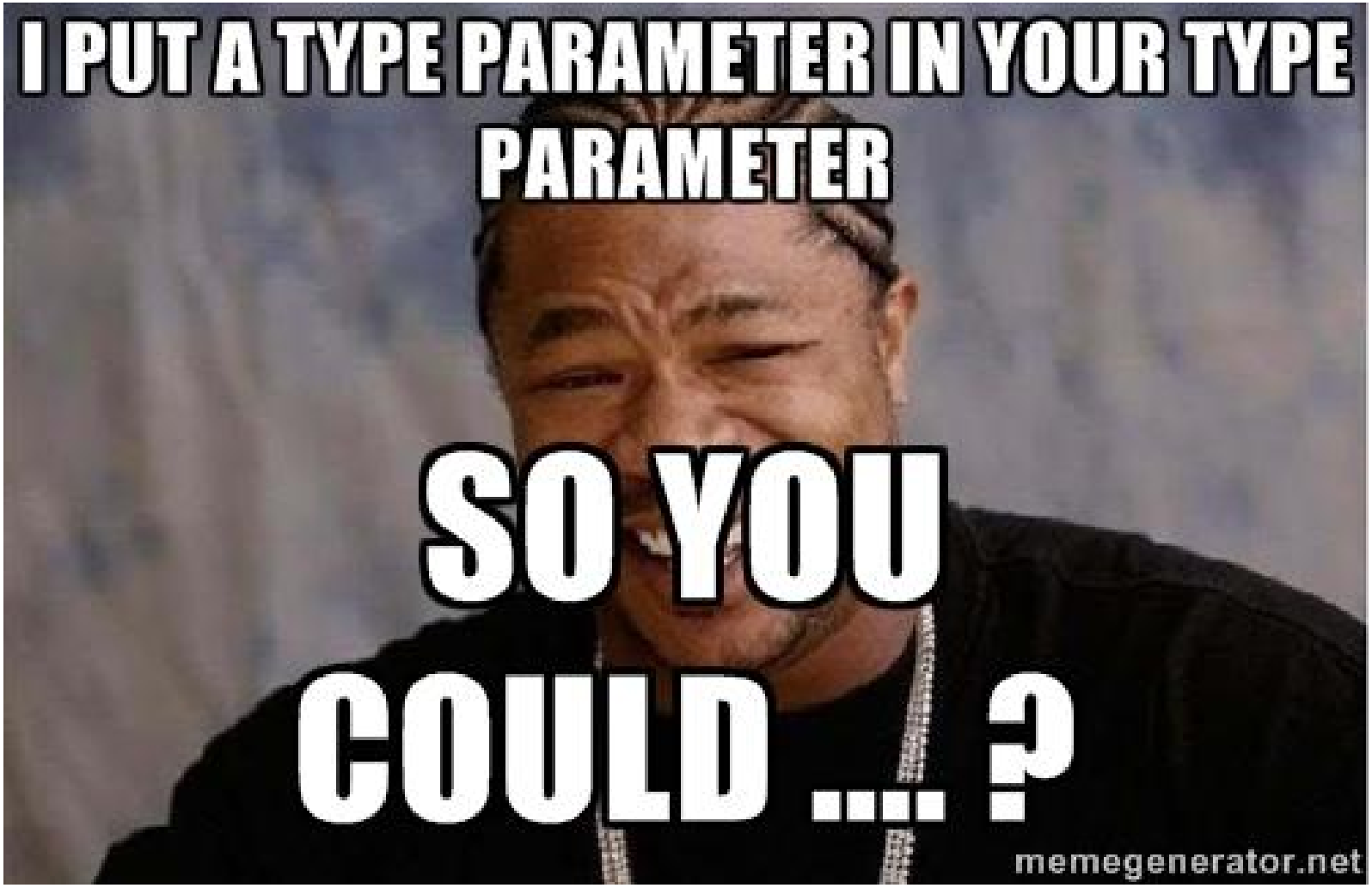
```
Object max(Collection coll)
```

Serializable lambdas

```
<T, U extends Comparable<? super U>> Comparator<T>
comparing(Function<? super T, ? extends U> keyExtractor) {
    Objects.requireNonNull(keyExtractor);
    return (Comparator<T> & Serializable)
            (c1, c2) ->
                    keyExtractor.apply(c1)
                                .compareTo(keyExtractor.apply(c2));
}
```

class Enum<E extends Enum<E>>

Curiously Recurring Generics Pattern



Bounded Wildcards

Examples

```
<T> List<T> unmodifiableList(List<? extends T> list)
```

```
<T> int binarySearch(List<? extends T> list, T key,
Comparator<? super T> c)
```

```
<T> int binarySearch(List<? extends Comparable<? super
T>> list, T key)
```

It’s all about subtyping!

? super

Commonly used for Functional Interfaces

`Comparator<Foo>`

- always `Comparator<? super Foo>`
- `int compare(T o1, T o2);`
- `Comparator<Message>` **<:** `Comparator<? super EmailMessage>`

`Predicate<Foo>`

- always `Predicate<? super Foo>`
- `boolean test(T t);`
- `Predicate<Message>` **<:** `Predicate<? super EmailMessage>`

Adoption and use of Java generics

90% generics use with Collections

- `List<String>, ArrayList<String>,`
- `HashMap<String,String>, Set<String>`

wildcards 10%

- `Class<?>`

Chris Parnin, Christian Bird, Emerson Murphy-Hill

Adoption and use of Java generics

http://www.cc.gatech.edu/~vector/papers/generics2.pdf

Intersection Types

Curiously Recurring Generics Pattern

Wildcards

Use-site variance

```
static void logAllWithAction(List<? extends Message> messages,
                             Consumer<? super Message> action) {
        messages.forEach(action);
}
```

Declaration-site variance

Library:

```
interface Consumer<? super T> {
    void accept(T t);
}
interface Iterator<? extends E> {
    E next();
    ...
}
```

User code:

```
static void logAllWithAction(Iterator<Message> messages,
                             Consumer<Message> action) {
        ...
}
```

Declaration-site variance

- User-site variance
  - variance complexity pushed to users
  - can add more verbosity due to annotations
- Declaration-site variance
  - variance complexity pushed to library level
  - List needs to be split in ReadOnly, WriteOnly
  - Adopted by C#, Scala

Improved variance for generic classes and interfaces

http://openjdk.java.net/jeps/8043488

Empirical Analysis for Declaration-site variance

- At least **27%** of generic classes and **53%** of generic interfaces in the examined libraries have an inherently variant type parameter.
- At least **39%** of wildcard uses in these libraries could be made unnecessary with declaration-site variance.

John Altidor, Shan Shan Huang, & Yannis Smaragdakis. *Taming the Wildcards: Combining Definition- and Use-Site Variance.*

http://jgaltidor.github.io/variance_pldi11.pdf

Type inference and Generics

```
Map<User, List<Transaction>> userToTransactions
    = new HashMap<User, List<Transaction>>();
```

Since Java 7:

```
Map<User, List<Transaction>> userToTransactions
    = new HashMap<>();
```

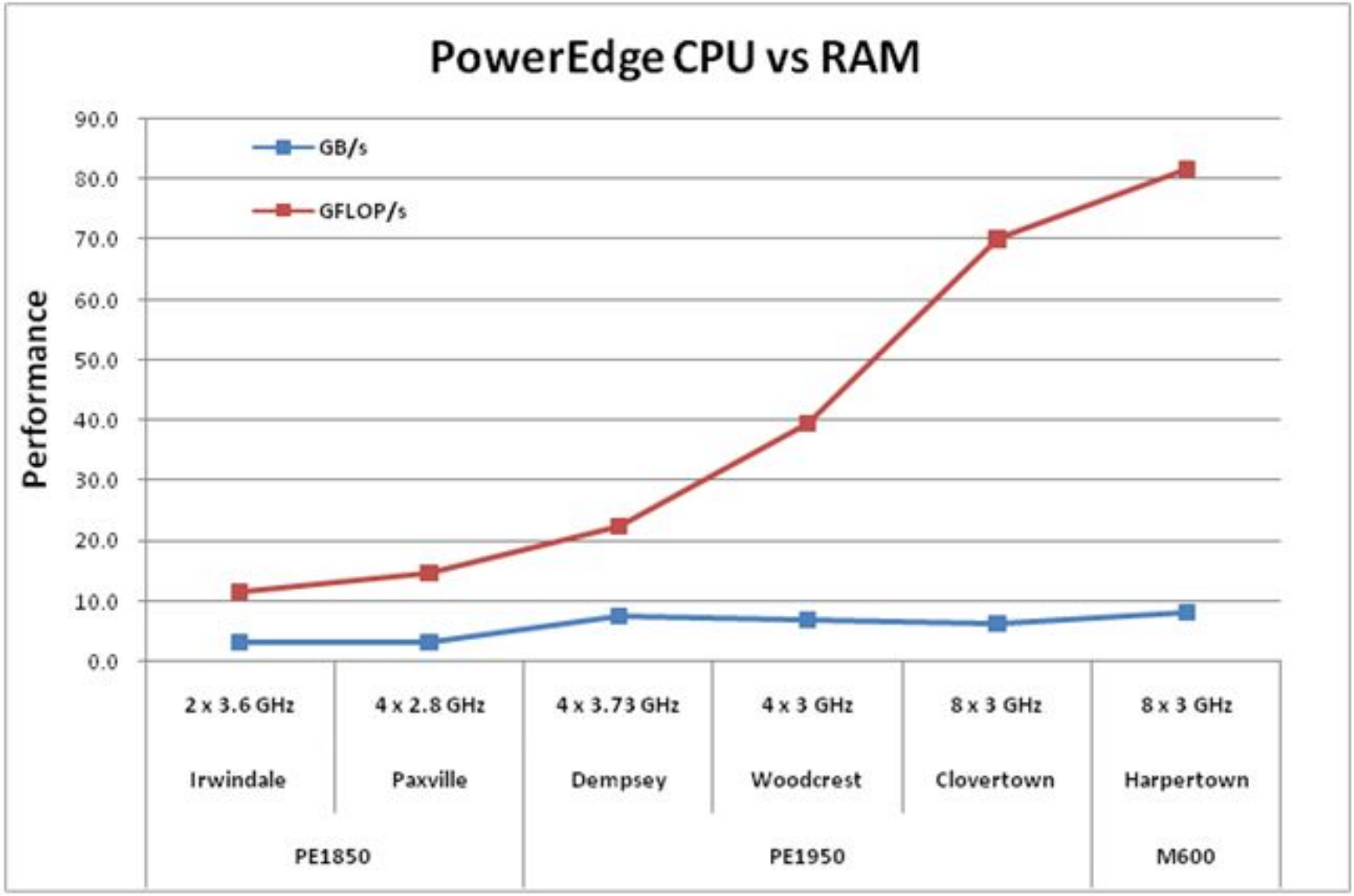
Since Java 8:

```
BiConsumer<User, Integer> transfer
    = (user, amount) -> {};
```

JEP 286: Local type inference

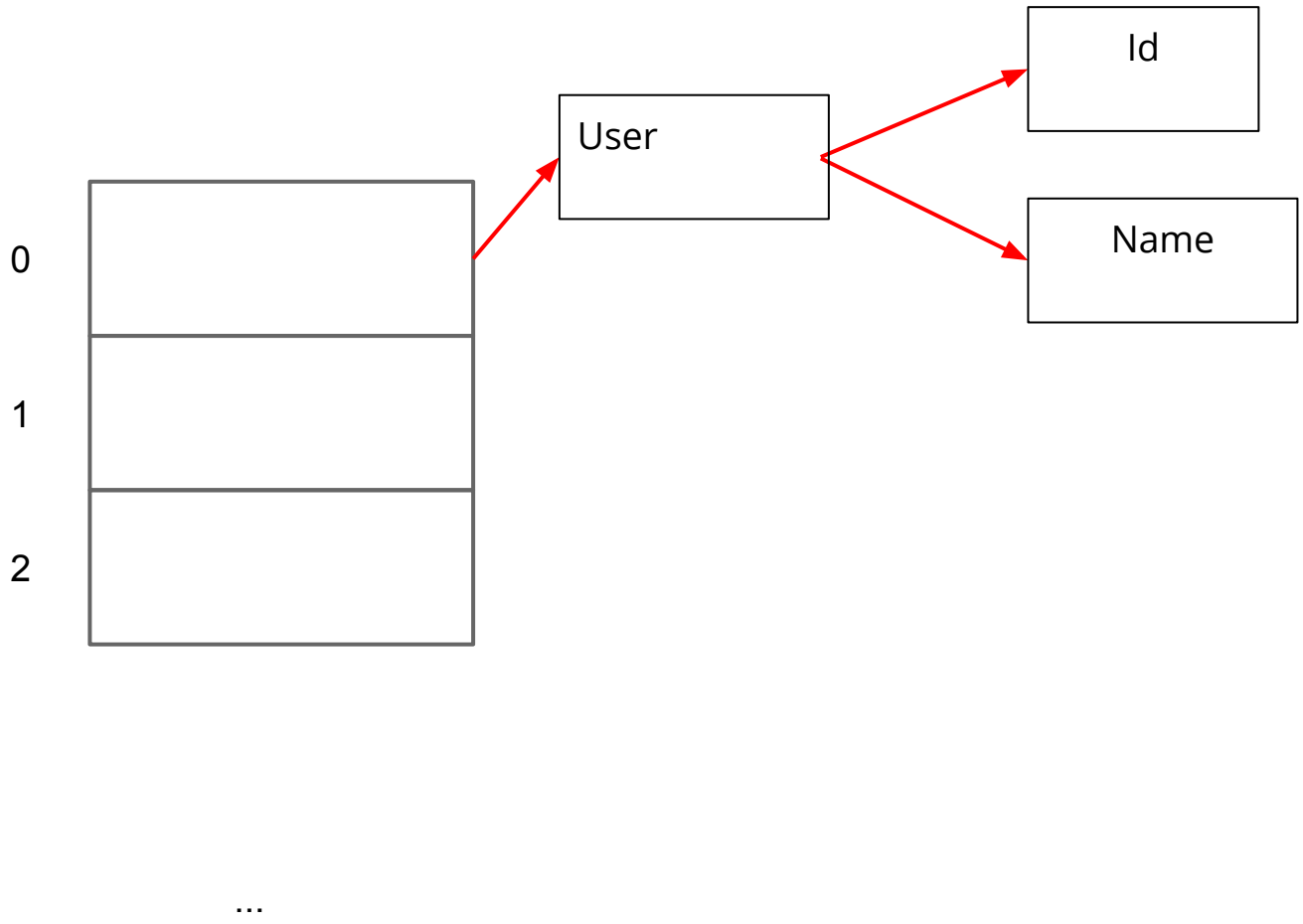
```
var userToTransactions
    = new HashMap<User, List<Transaction>>();

var answer = 42;

var names = Stream.of("Bob", "Marley");
```

Binary builds available at http://iteratrlearning.com/jep286.html



http://media.community.dell.com/en/dtc/ux9mpc-lbzbuhgas9izg4g41691.png

Poor Sequential Locality (Flatness)



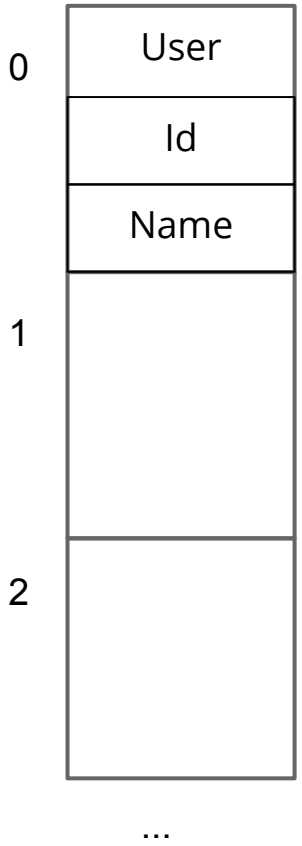
Value Types

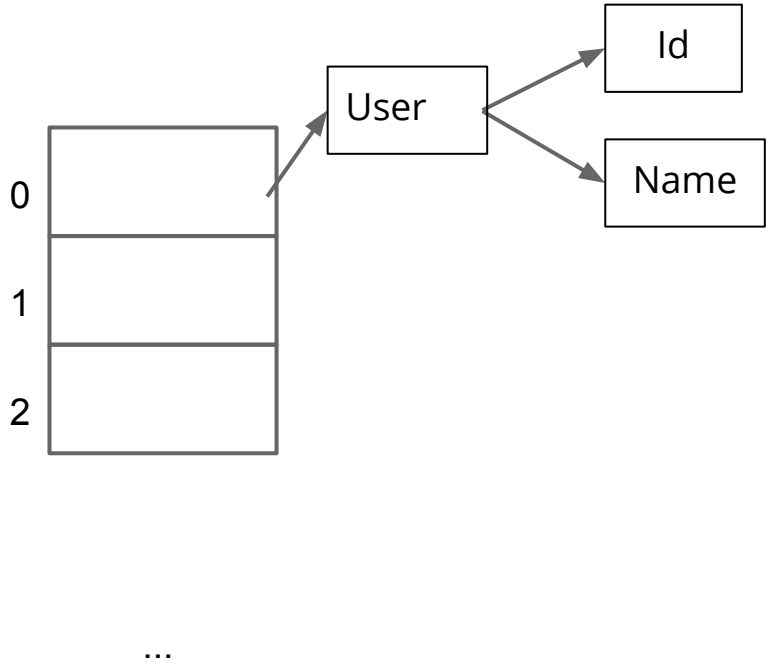
- *“codes like a class, works like an int”*
- No Identity
- Just a *struct* of values

Compactness (Less memory)

- No Mark Word
  - Locking
- No klass pointer
- Saving 8-16 bytes depending upon architecture/VM

Sequential Locality (Flatness)





```
class ArrayList<any T> implements List<T>
```

```
List<int> numbers = new ArrayList<>();
numbers.add(1);
numbers.add(2);
```

```
this.elementData =

new Object[initialCapacity];
```

```
null => T.default
```

What should `ArrayList<boolean>` store its data in?

You can help

http://cr.openjdk.java.net/~briangoetz/valhalla/specialization.html

http://openjdk.java.net/projects/valhalla/

For Reference

- Source Code
  - https://github.com/RichardWarburton/generics-examples
- Unbounded Wildcards
- Type Bounds
- Erasure Problems & Advantages
- Static safety failures
- Other Languages & Features (Lambda Cube)

Conclusions

- Usage patterns change as other features are added
- Generics usage continues to increase in both scale and complexity
- Most of the complexity burden is on library authors

Static Type-safety often involves a tradeoff

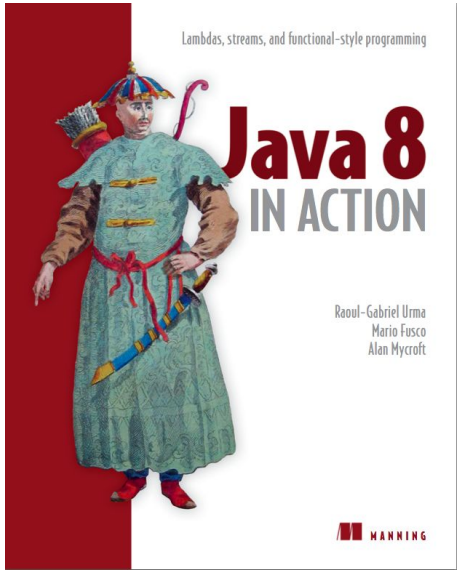
between simplicity and flexibility

Any Questions?

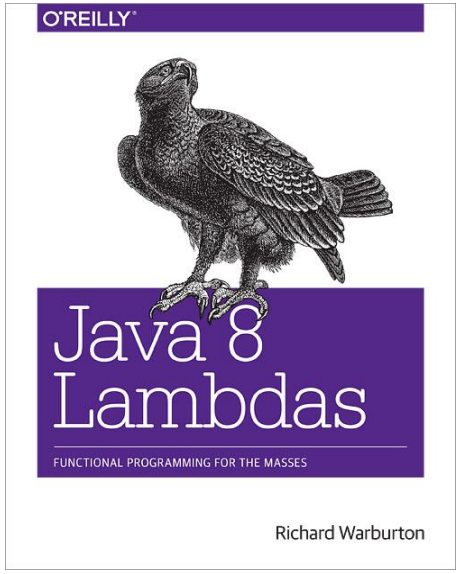
www.pluralsight.com/author/richard-warburton

www.cambridgecoding.com

www.iteratrlearning.com



http://manning.com/urma



http://tinyurl.com/java8lambdas

The End

Richard Warburton
(@richardwarburto)

Raoul-Gabriel Urma
(@raoulUK)

http://iteratrlearning.com

Mmmm

Java API

<T> List<T>

```
unmodifiableList(List<? extends T> list)
```

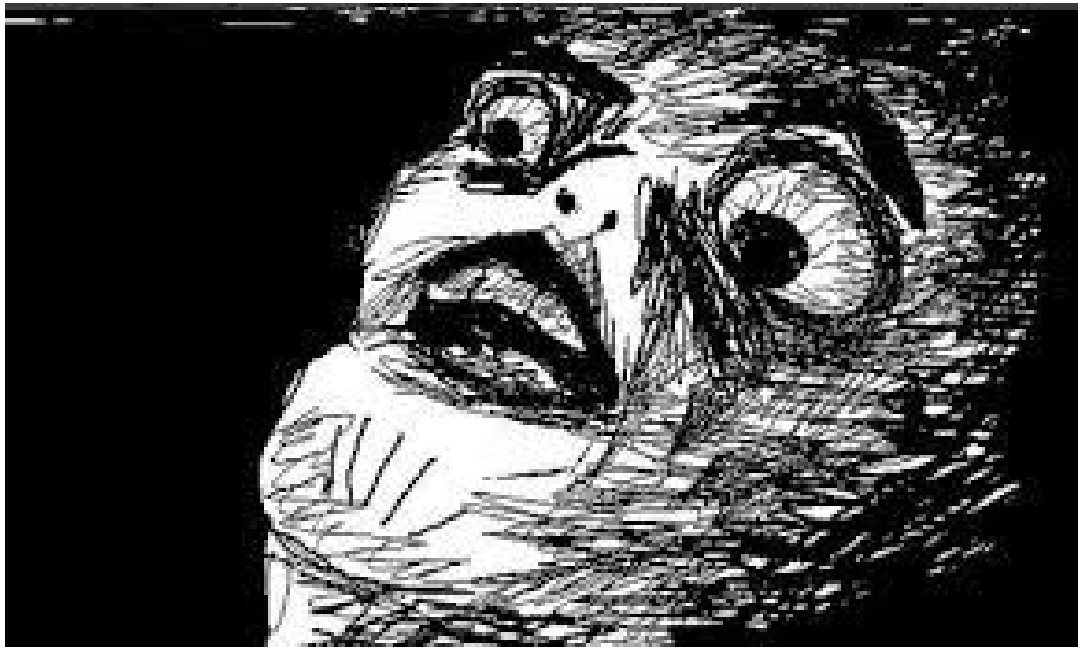
vs

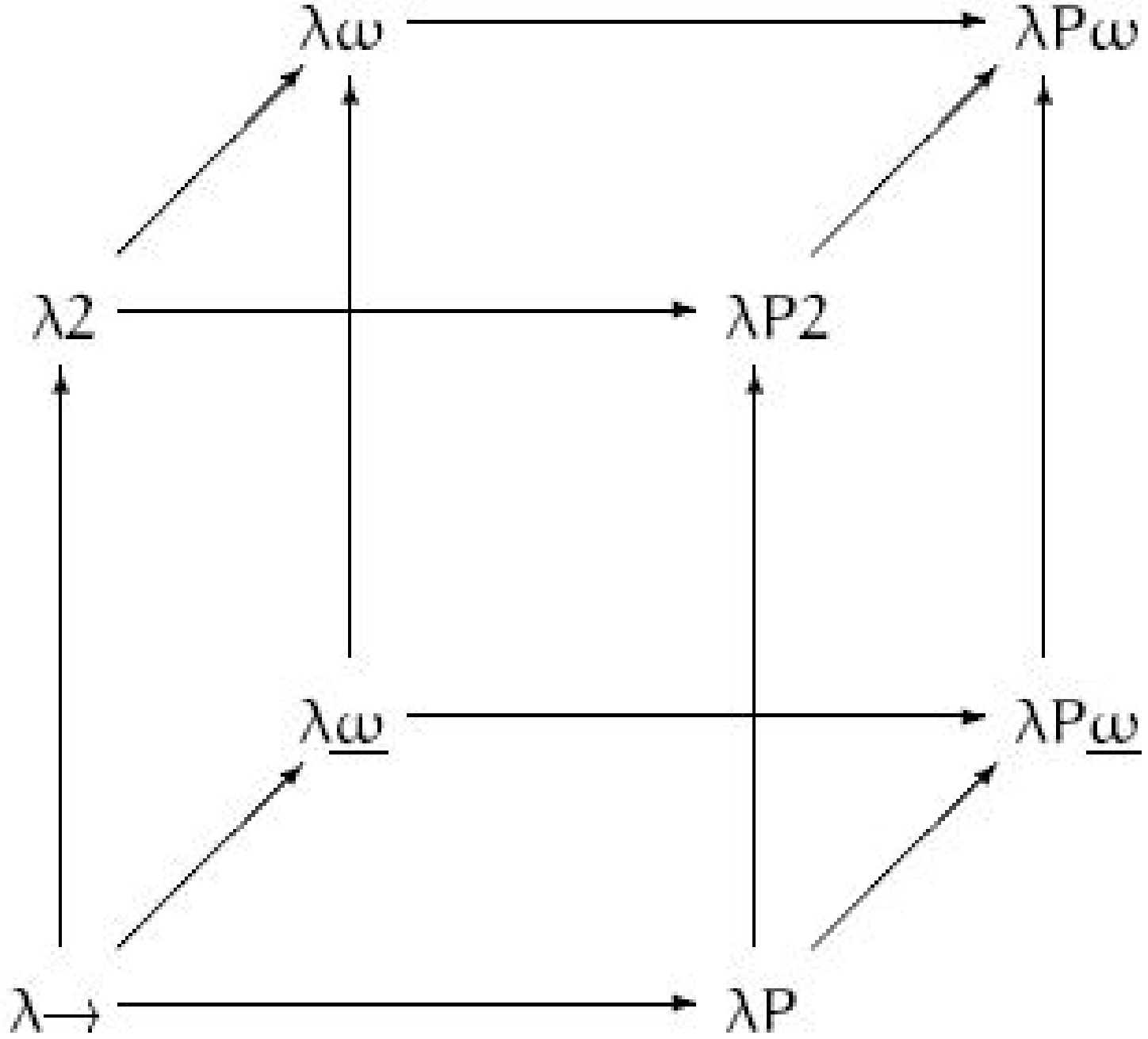
<T> List<? extends T>

```
unmodifiableList(List<? extends T> list)
```

From Java 8's Collectors

```
public static <T,K,U,M extends Map<K,U>> Collector<T,?,M>
toMap(Function<? super T,? extends K> keyMapper,
                Function<? super T, ? extends U> valueMapper,
                BinaryOperator<U> mergeFunction,
                Supplier<M> mapSupplier)
```




λω
λPω
λ2
λP2
λω
λPω
λ→
λP

Higher kinded types

```
trait Mapable[F[_]] {
  def map[A, B](fa: F[A])(f: A => B): F[B]
}

Stream[T] extends Mapable[Stream]
Option[T] extends Mapable[Option]
```