Dynamic Code Evolution

for the

Java HotSpot™ Virtual Machine

Thomas Wuerthinger (wuerthinger@ssw.jku.at)
Institute for System Software
Johannes Kepler University Linz, Austria
09/16/09

My Background

Array Bounds Check Elimination

http://wikis.sun.com/display/HotSpotInternals/Publications+JKU

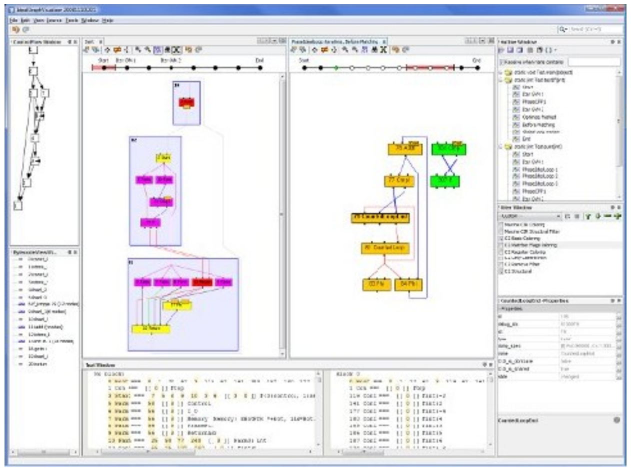
- for the HotSpot client compiler

Ideal Graph Visualizer

http://www.kenai.com/projects/igv

- focused on HotSpot server compiler graph

- visualization of evolving graph data structures



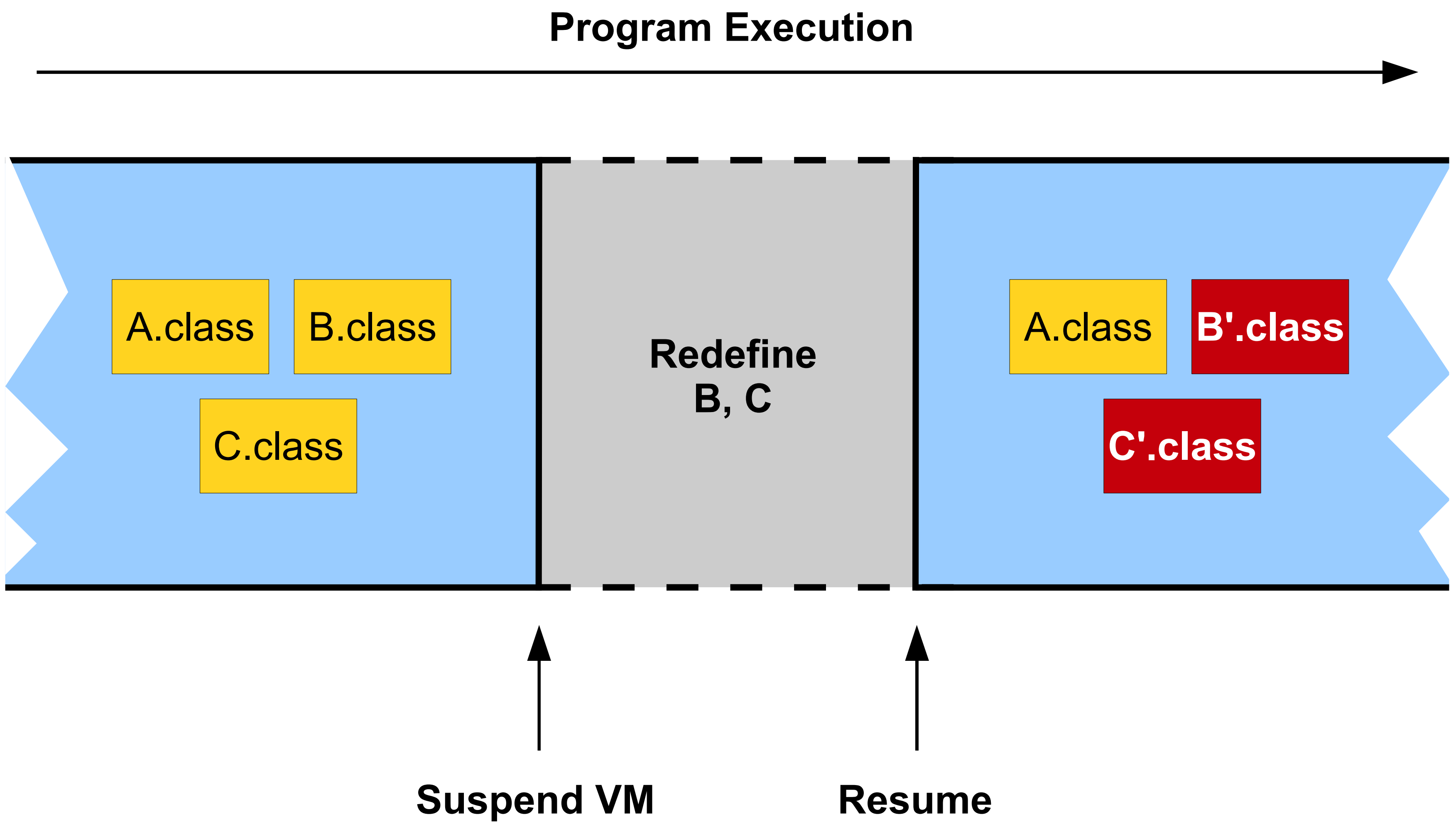
Maxine

http://www.kenai.com/projects/maxine

- meta-cirtular virtual machine written in Java

- recent focus: Porting the HotSpot client compiler to Java

Class Evolution



Applications

Debugging

Atomic changes of a set of classes

No restrictions on the type of changes

Update at specific point (e.g. at a break point)

No additional indirections (e.g. that produce strange stack traces)

Long-Living Server Applications

No performance penalty on normal execution

Find a „good“ update point

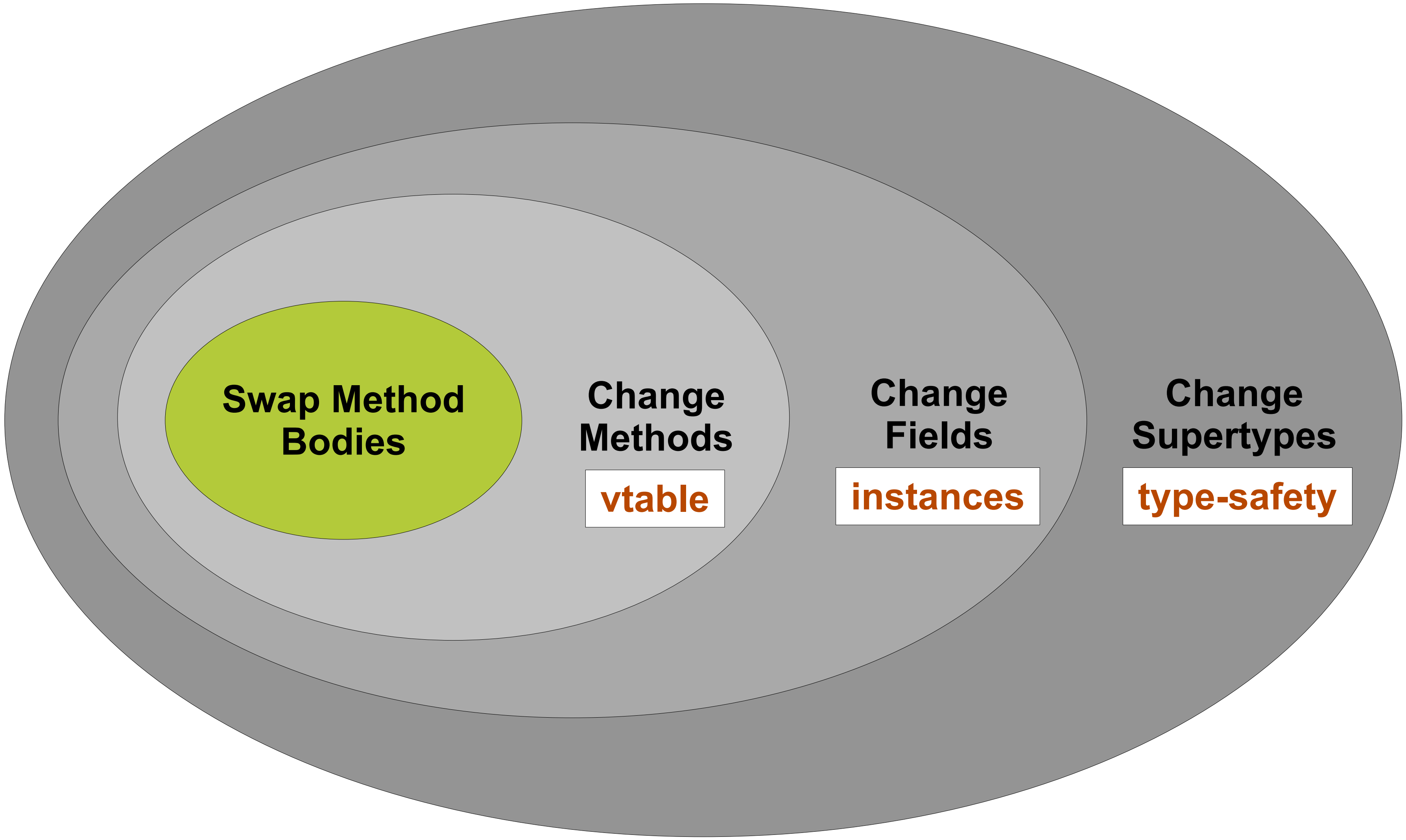
Stability and security issues

Scripting Languages

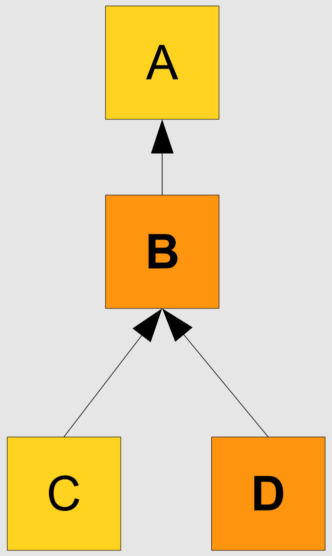
Fast, small, incremental changes

Focus on additions (new methods, new fields, ...)

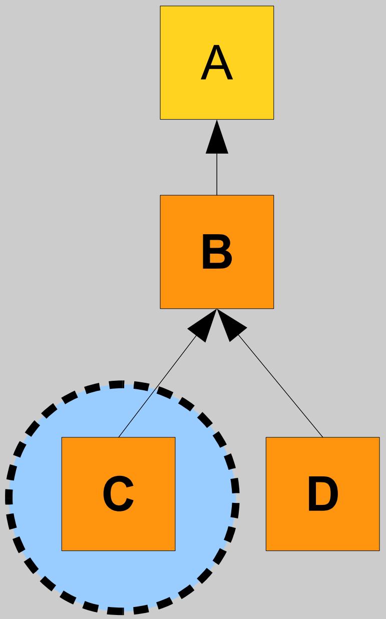
Levels


Swap Method Bodies
Change Methods
vtable
Change Fields
instances
Change Supertypes
type-safety

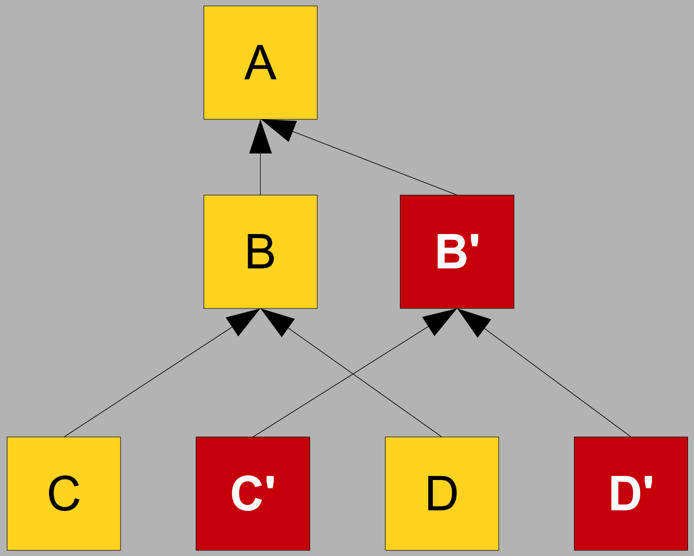
Redefine B, D

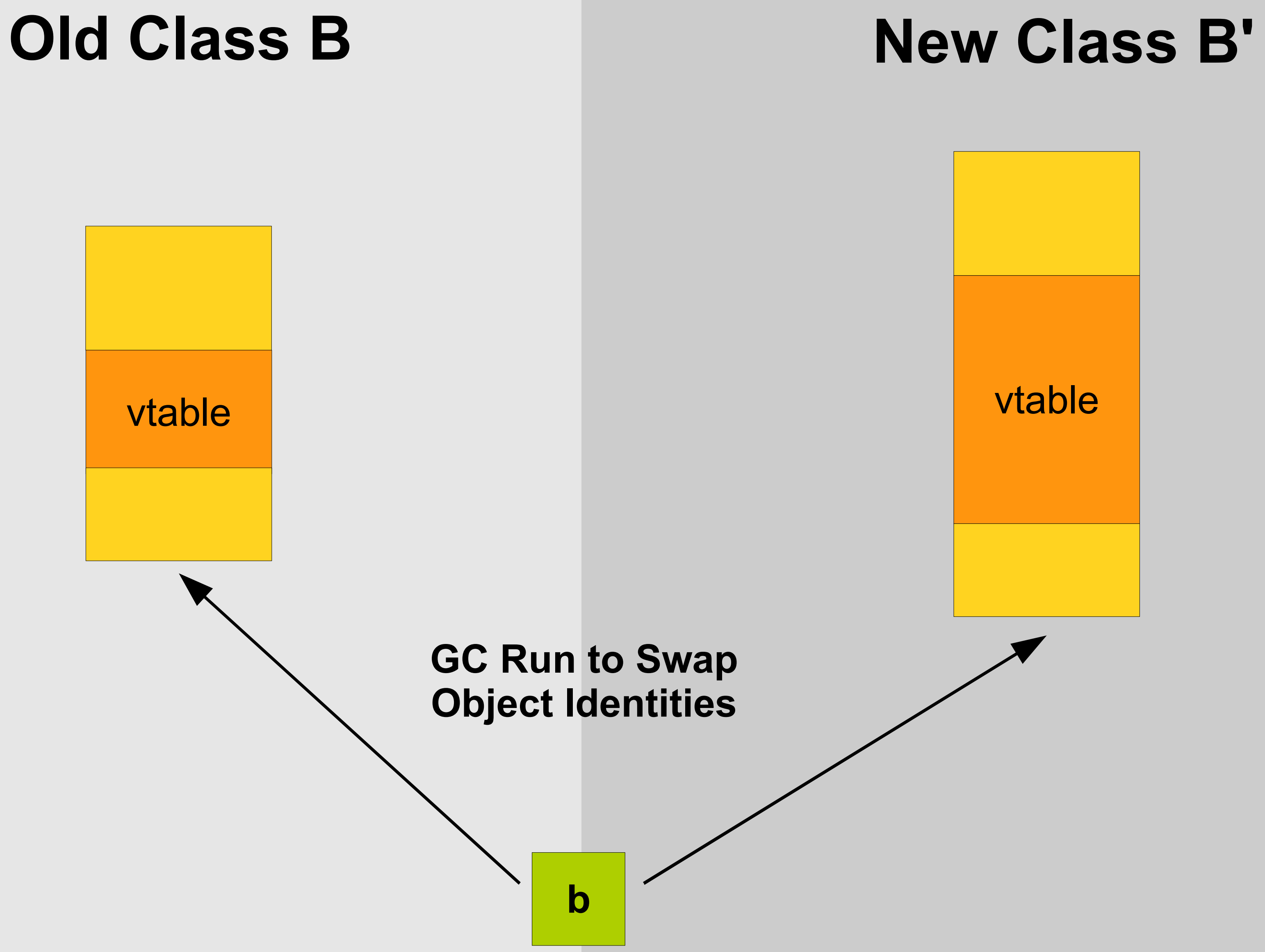


Find Affected Classes

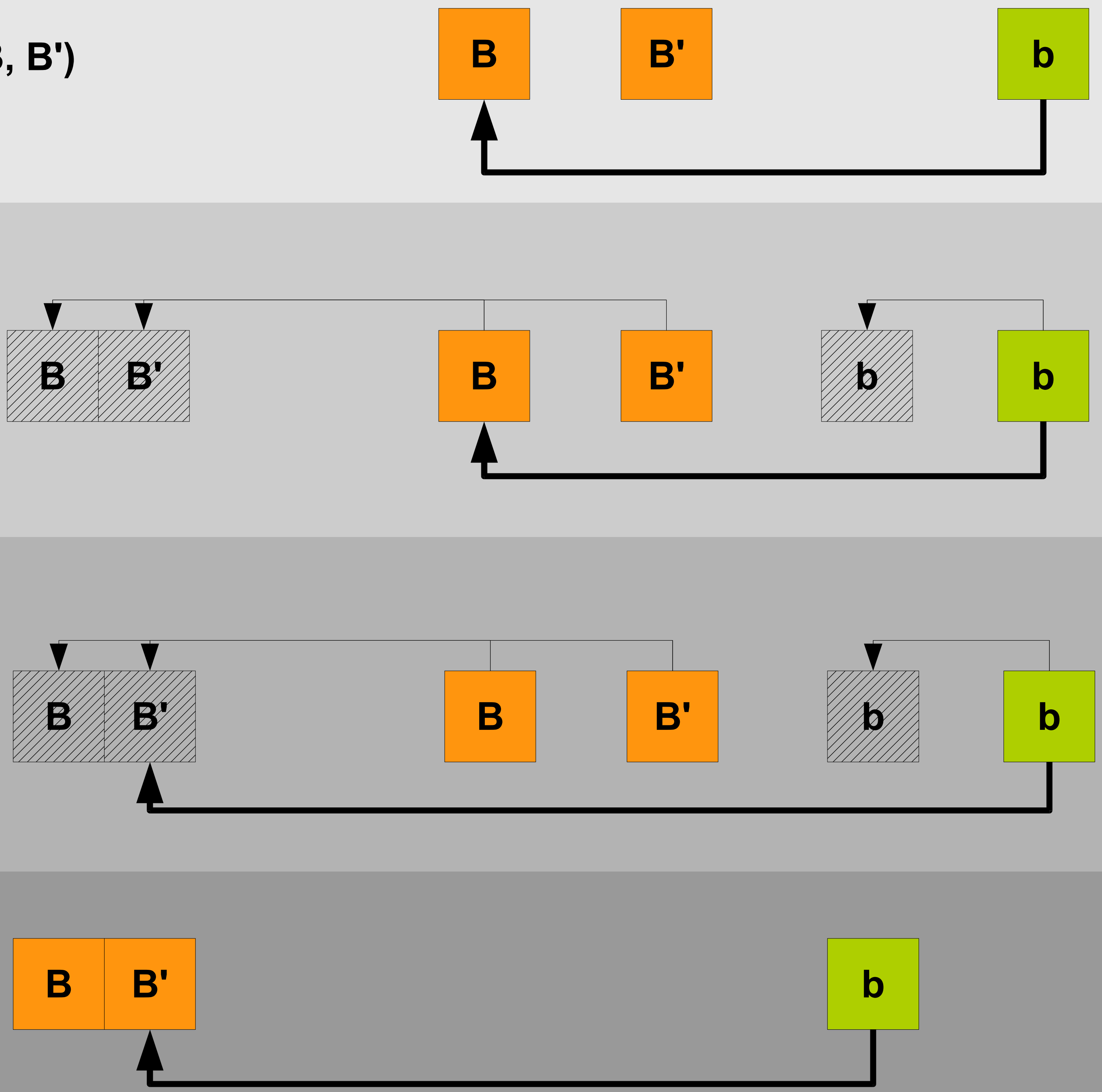


Build Side Universe

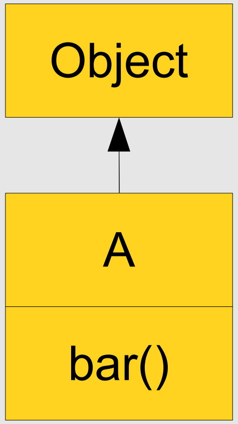


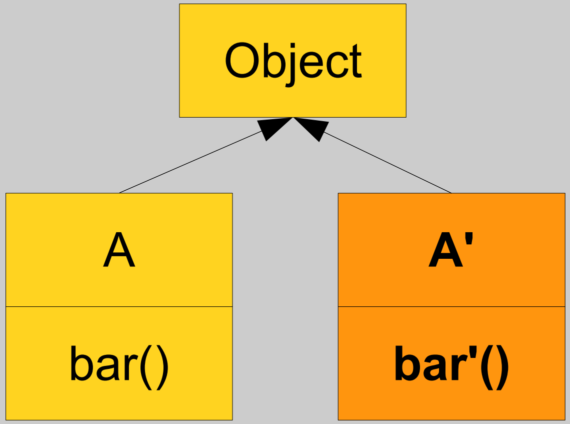

Old Class B
New Class B'
vtable
vtable
GC Run to Swap Object Identities
b

Swap (B, B')



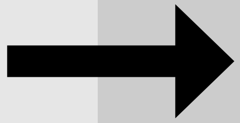
Changing Active Method





```
void foo() {
   // <= REDEFINE
   bar();
   ...
}
```

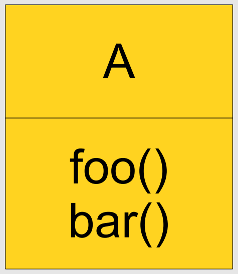
Deoptimization

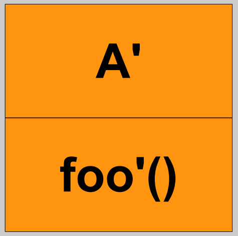


Update Constant Pool Cache

```
void foo() {
   // <= RESUME
   bar'();
   ...
}
```

Removing a Method





```
int foo() {
   // <= REDEFINE
   return bar();
}
```

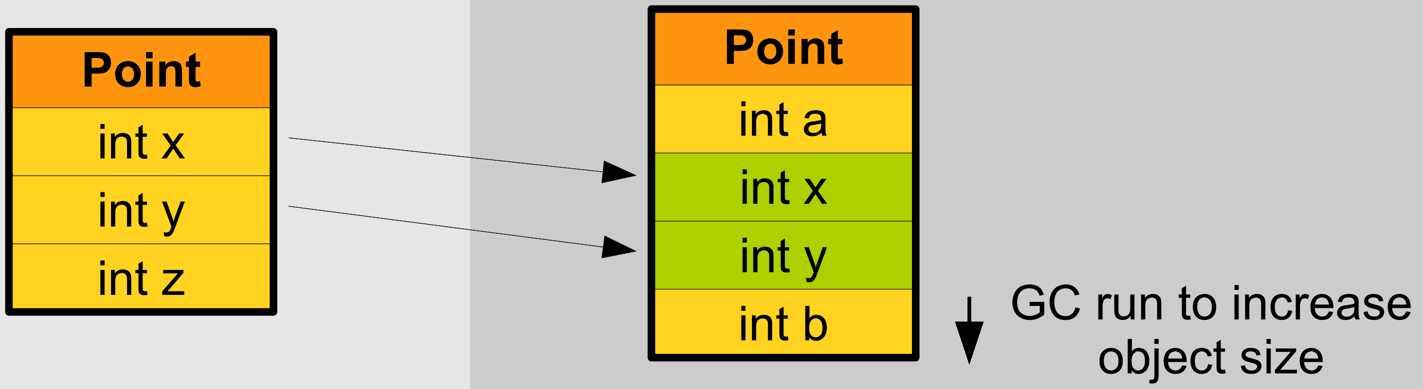
```
int foo'() {
   return 5;
}
```

throws NoSuchMethodException in foo()

Possible alternative solutions:

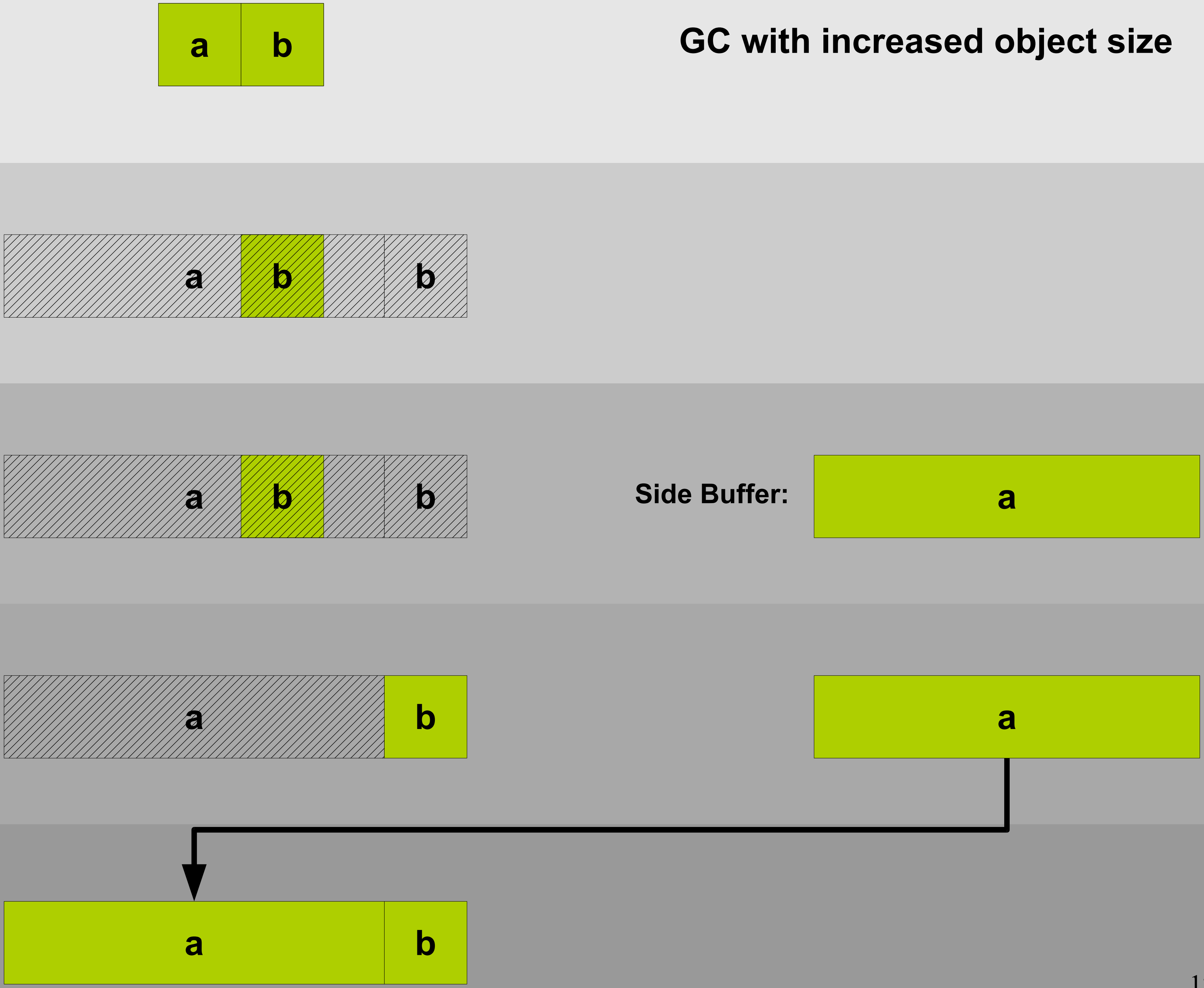
- Continue executing „deleted“ methods
- Replace method invocation by a constant value
- Delay code evolution until invalid old code is no longer active

Instance Updates

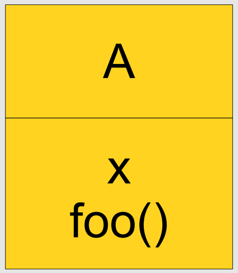


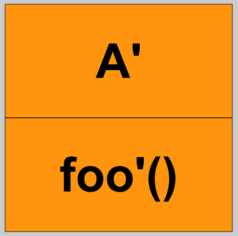
Future Plans: Connect with the NetBeans refactoring facilities

GC with increased object size



Removing a Field





```
int foo() {
   // <= REDEFINE
   return x;
}
```

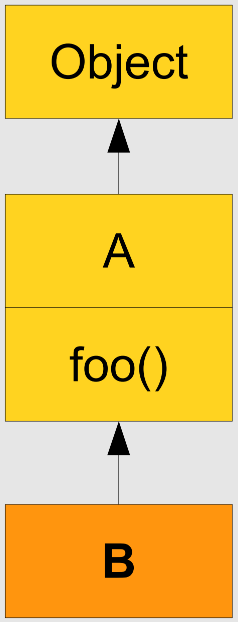
```
int foo'() {
   return 5;
}
```

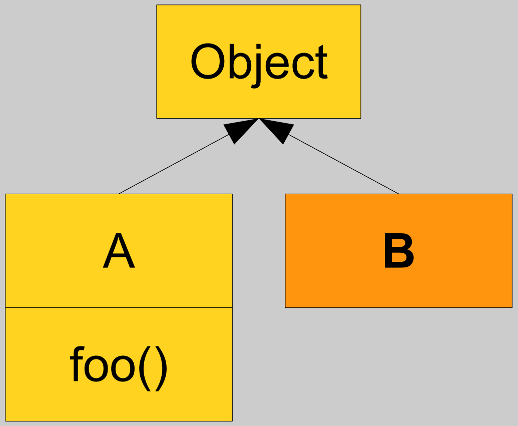
throws NoSuchFieldException in foo()

Possible alternative solutions:

- Keep the deleted field in „old“ objects
- Replace the field access by a constant value (e.g. 0)

Supertype Change Problem





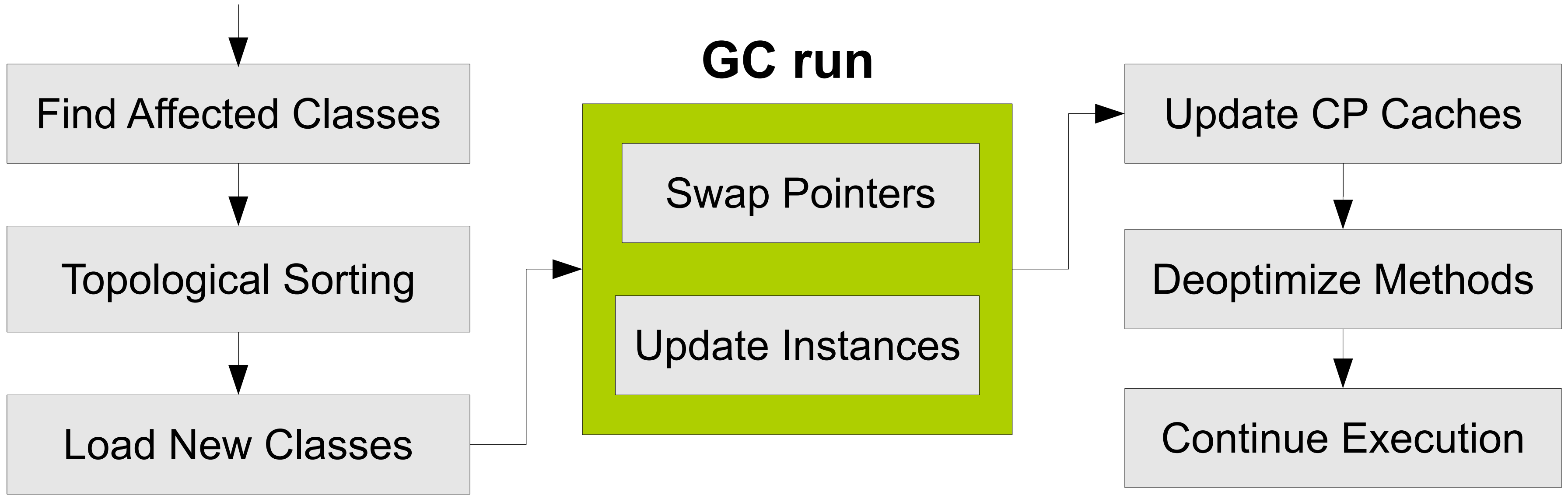
```
A a = new B();
```

a.foo(); // ?!

Possible solutions:

- Prohibit type narrowing through hotswapping if instances exist
- Replace type system violating object pointers with null

Summary



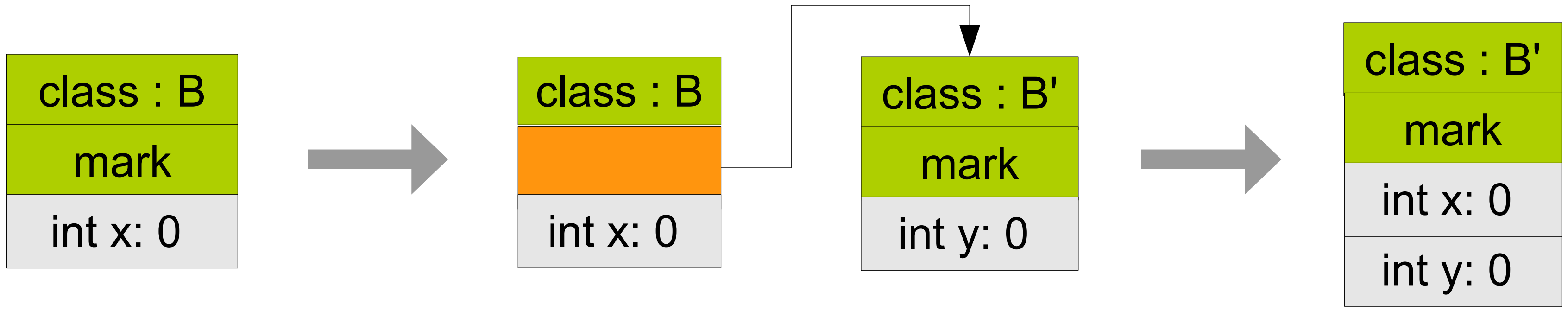
Arbitrary Changes Possible

No Additional Runtime Overhead

Future Work

Performance

Remove need for GC run in case of „small“ updates (adding methods or fields)



Improved IDE Support

Connect with the NetBeans refactoring facilities

Security

API for selecting „safe“ program execution points

Patch + detailed technical documentation available at

http://wikis.sun.com/display/mlvm/HotSwap

Feel free to post any questions to

mlvm-dev@openjdk.java.net or wuerthinger@ssw.jku.at

$$\begin{tikzcd}
\textbf{Thanks} \arrow[d] & \\
\text{for} \arrow[r] & \textbf{your} \arrow[d] \\
& \text{attention!}
\end{tikzcd}$$

Q & A