Binary Component Adaptation

Modifying Components On The Fly

Ralph Keller and Urs Hölzle
Department of Computer Science
University of California, Santa Barbara
http://www.cs.ucsb.edu/oocsb

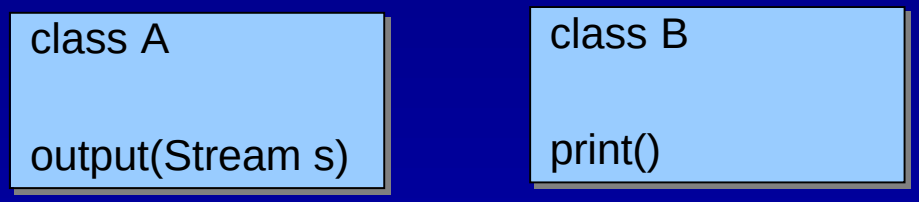
Motivation

- OOP vision:
  - Pervasive component reuse
  - Large-scale component market
  - Programs are built mostly of existing components
- Unfortunately: Combining independently developed components is hard
- Binary Component Adaptation makes integration easier!

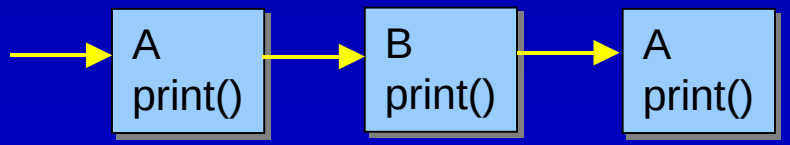
Talk Outline

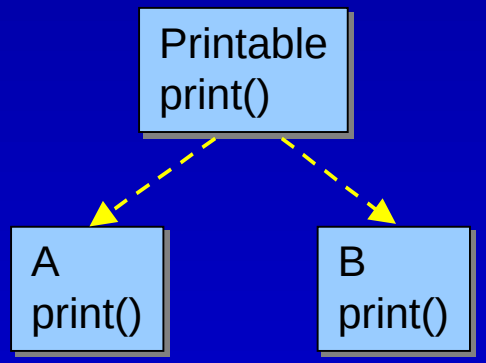
- Problems with component-based programming
- Binary Component Adaptation
- Performance
- Conclusions

The Integration Problem



- Both components provide functionality for printing
- Goal: Integrate A and B in list:

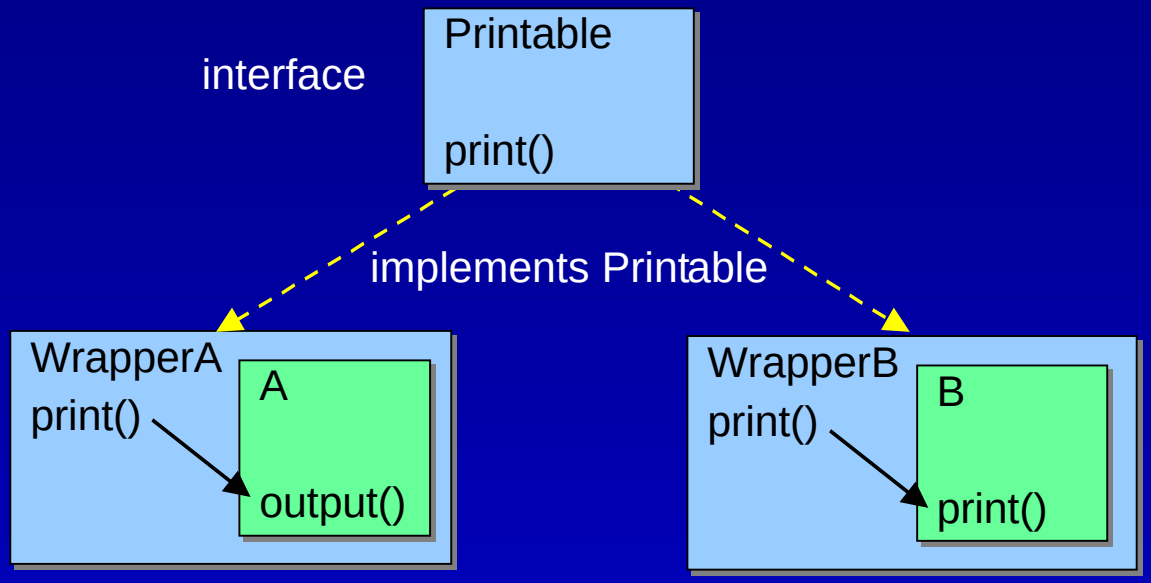




desired type hierarchy

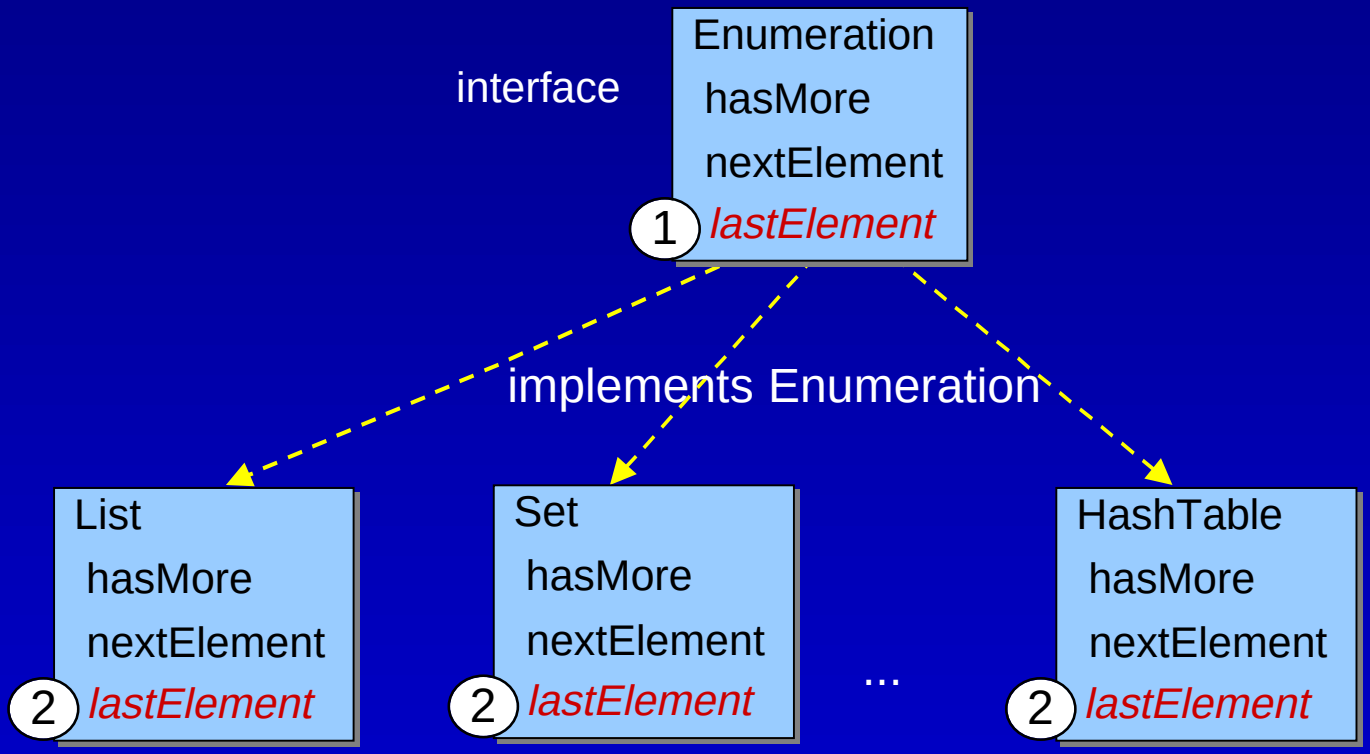
- But:
  - Interface mismatch
  - No common supertype

Using Wrapper Classes



- Disadvantages:
  - Additional programming effort
  - Overhead (wrapping and unwrapping)
  - Redundant type hierarchy
  - Difficult subtyping problems [Hölzle, ECOOP ‘93]

The Interface Evolution Problem



Interface Evolution Problem

- Interface change must be reflected immediately by all classes
- Default implementation for each class:

```
Object lastElement() {
  Object o = null;
  while (hasMore()) o = nextElement();
  return o;
}
```

- But:
  - Generally no source code
  - Open set of classes (dynamic linking)

Problem Summary

- Minor incompatibilities prevent pervasive reuse
- Each re-user may have different requirements
- Source code modifications are not practical
  - not available
  - needs recompile before executing
  - need to keep track of different compiled versions
  - tracking new releases is tedious

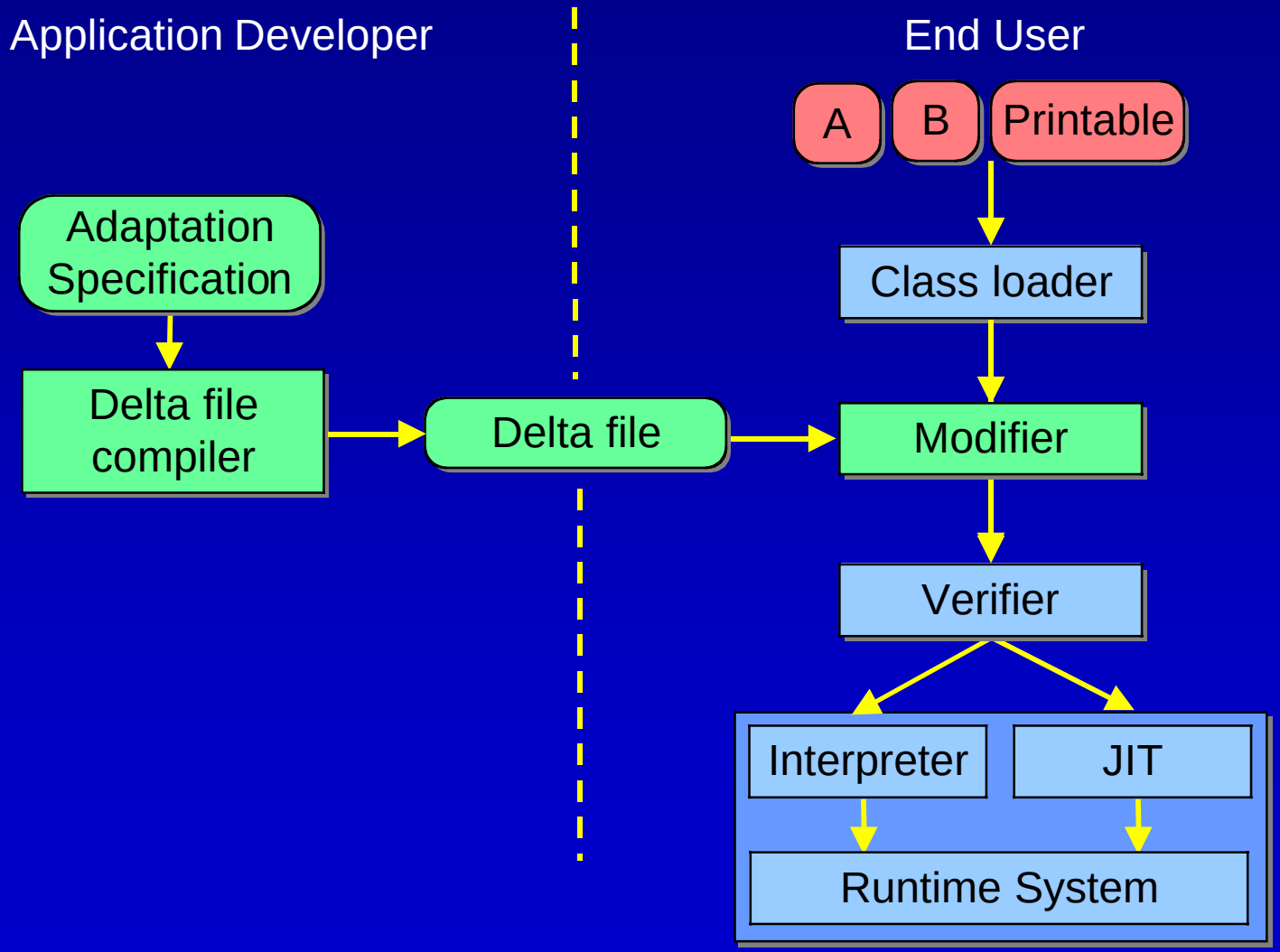
What We Want

- Easy-to-use, flexible adaptation mechanism
- No source code required
- “In place” changes of type (no wrappers)
- Automatic integration with future releases

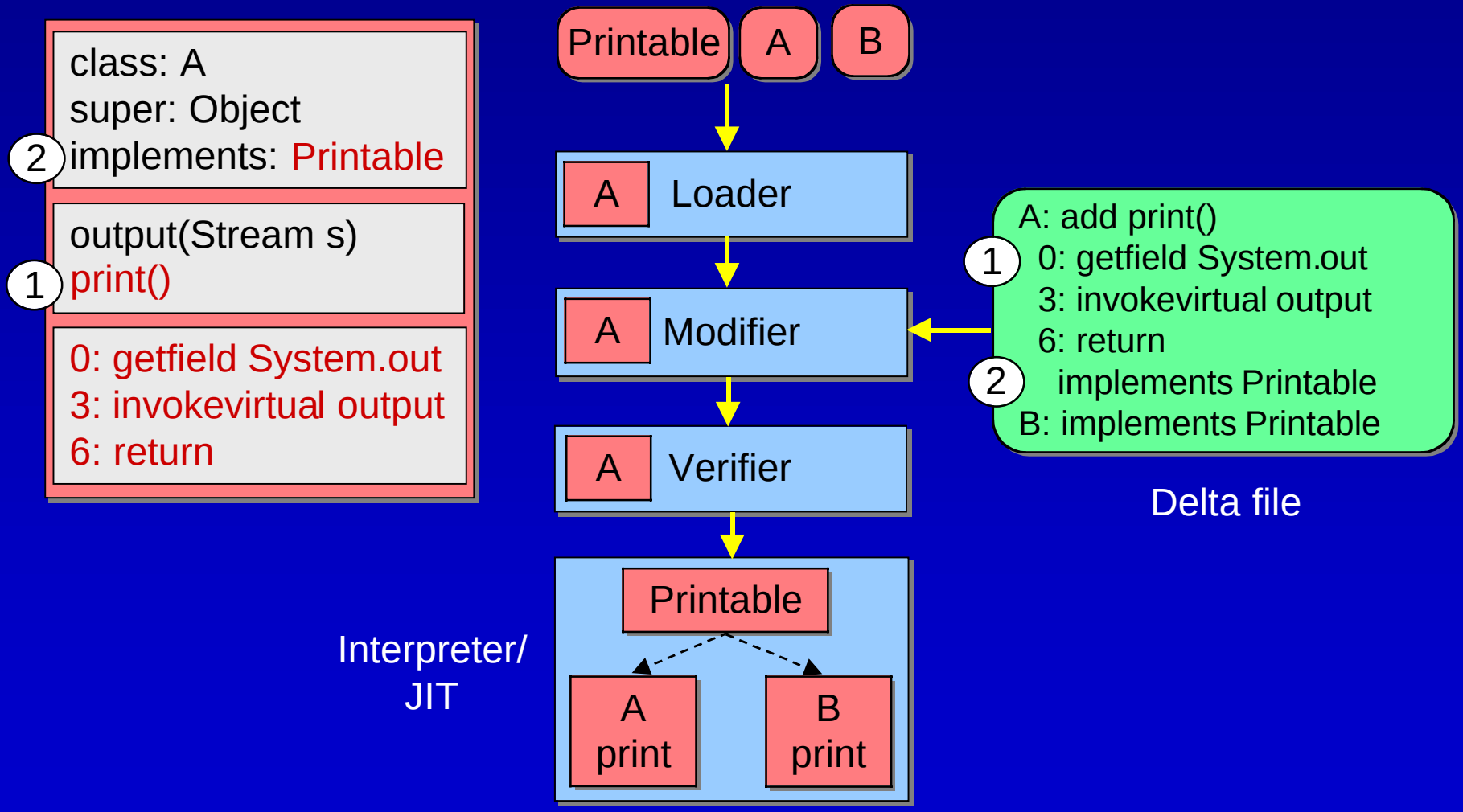
Talk Outline

- Motivation: Problems with component-based programming
- Binary Component Adaptation
- Performance
- Conclusions

BCA Overview



Modifying Components On The Fly



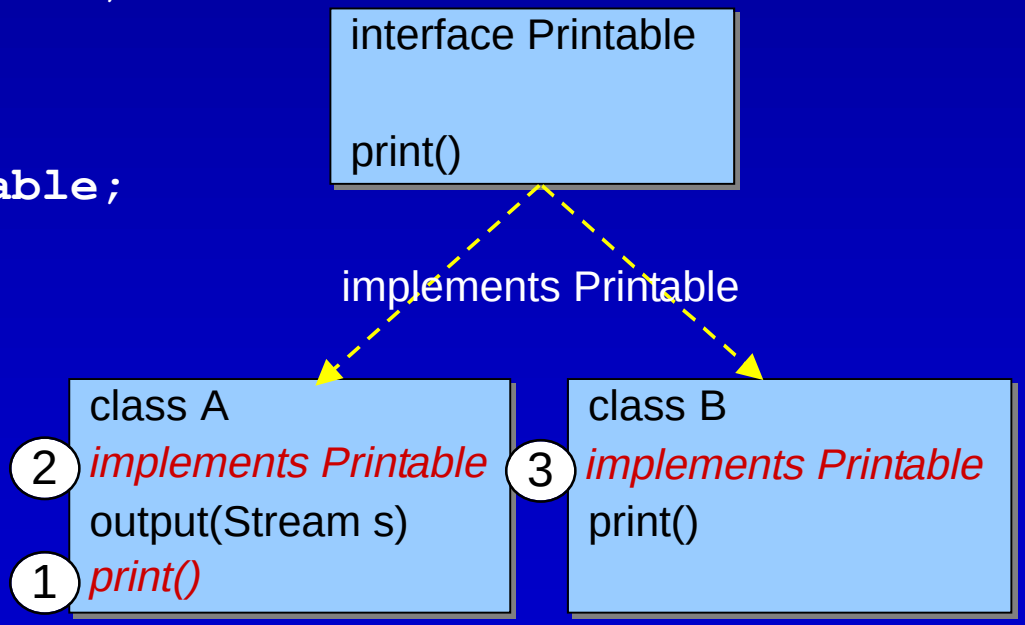
Binary Component Adaptation

- Directly modifies component
- Uses type information in compiled component
  - Requires no source code
  - Works even on third-party components
- Keeps changes separately from components
  - No changed components delivered to client
- Performs component modification during class loading

Adaptation for Integration Problem

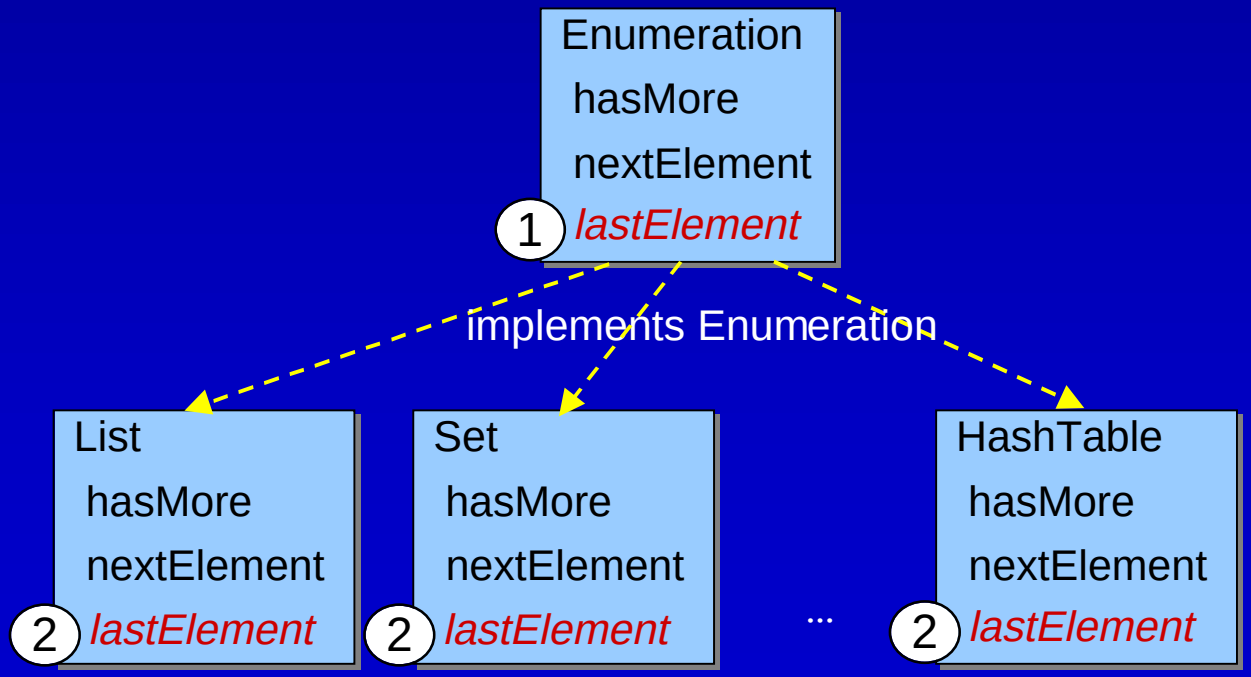
Language to specify changes for classes

```
delta class A {
1   add method void print() { output(System.out); }
2   add implements Printable;
}
delta class B {
3   add implements Printable;
}
```



Adaptation for Evolution Problem

```
delta interface Enumeration {
1   add method Object lastElement();
}
delta implements Enumeration {
2   add method Object lastElement() { … while(hasMore()… }
}
```

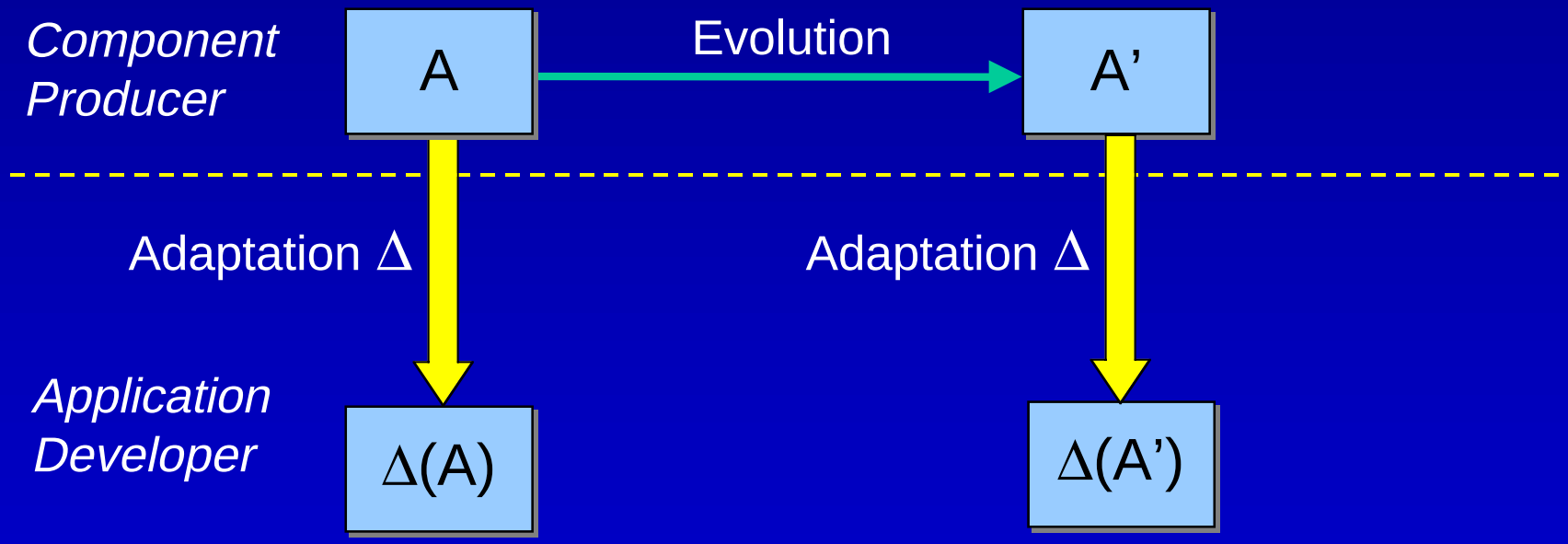


What Modifications Are Possible?

- Limited by amount of symbolic information
- For Java: can change almost everything
  - Add fields, methods
  - Add implemented interfaces
  - Delete fields, methods
  - Rewrite byte codes
- But: Binary compatibility should be preserved

Binary Compatibility

- Goal: Adaptation is valid on any future release of A



Binary Compatible Modifications

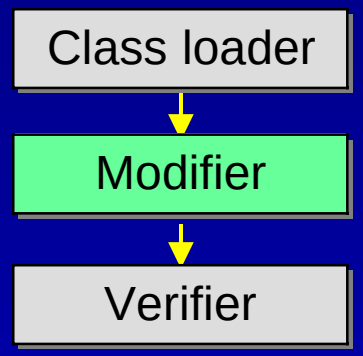
- Very flexible even with binary compatibility restriction:
  - Add fields & methods, reimplement methods
  - Change type hierarchy, …
- Even handles open systems
  - Rename classes or class members
  - Extend interfaces
- More flexible than source code!

Talk Outline

- Problems with component-based programming
- Binary Component Adaptation
- Performance
- Conclusions

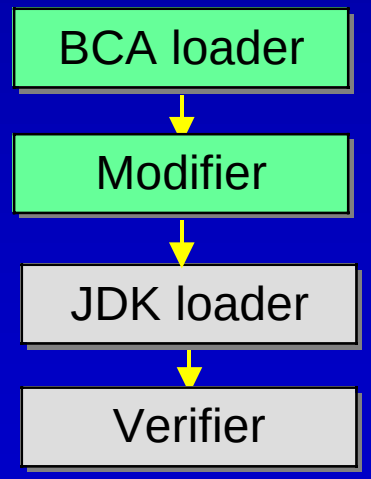
Overhead

- Introduced by modifier at load-time
- But code executes at full speed



BCA model

- Our implementation:
  - Easy to integrate into JDK
  - Independent of VM data structures



Extra Class Loader

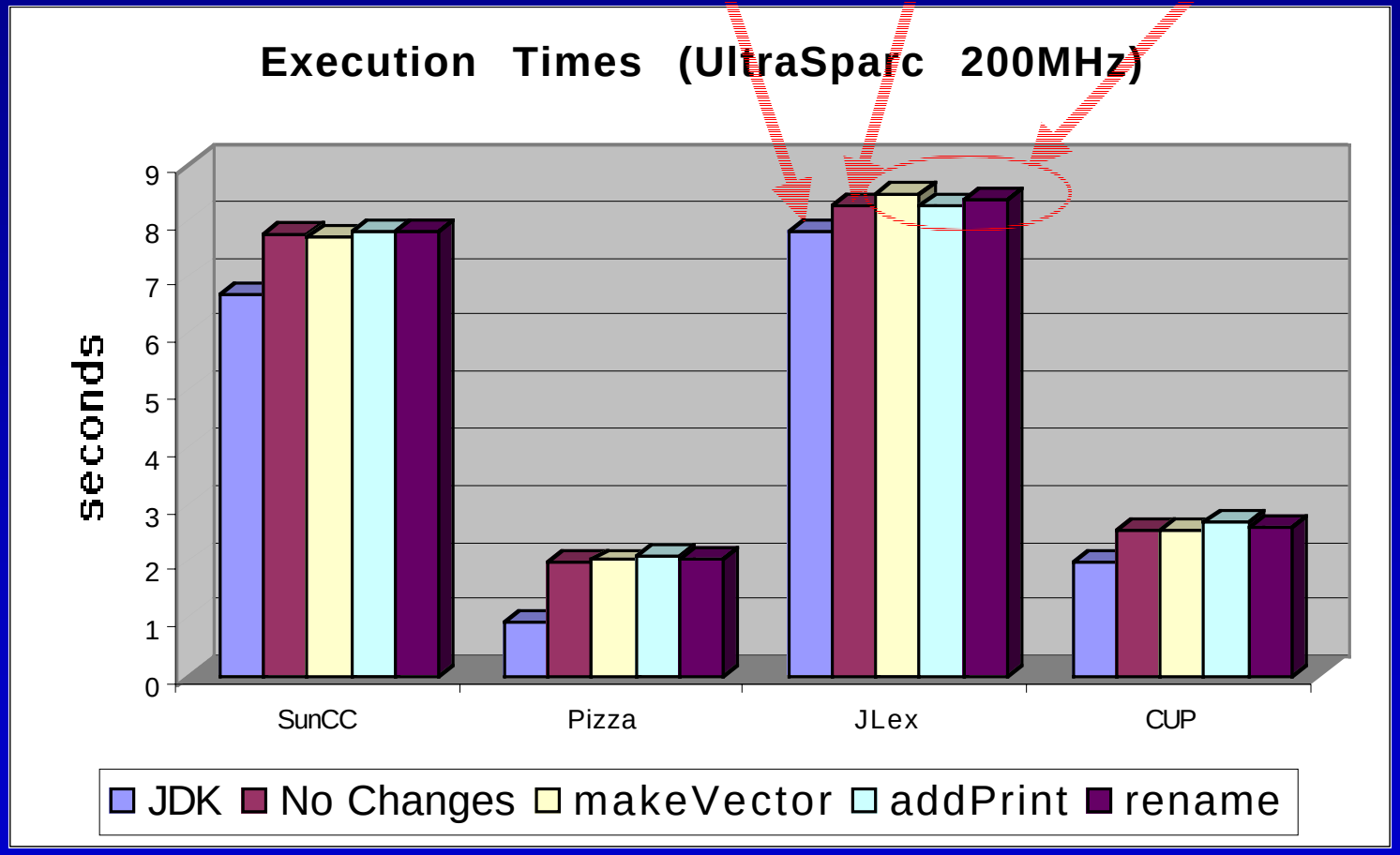
Current implementation

Performance

Sun JDK

BCA
no modifications

BCA with
modifications



Status and Future Work

- Implemented:
  - BCA-aware virtual machine
  - Delta file compiler
  - BCA-modified javac
- Freely available for SPARC/Solaris:
  - http://www.cs.ucsb.edu/oocsb/bca
- Future:
  - Type checker for delta files
  - More modifications (such as refactorings)

Conclusions

Binary Component Adaptation

- Is an adaptation mechanism for components
- Modifies components “in place” (type identity)
- Uses type information in compiled component
  - Requires no source code
  - Works on any component
- Guarantees release-to-release compatibility
- Is efficient enough to be performed at load-time

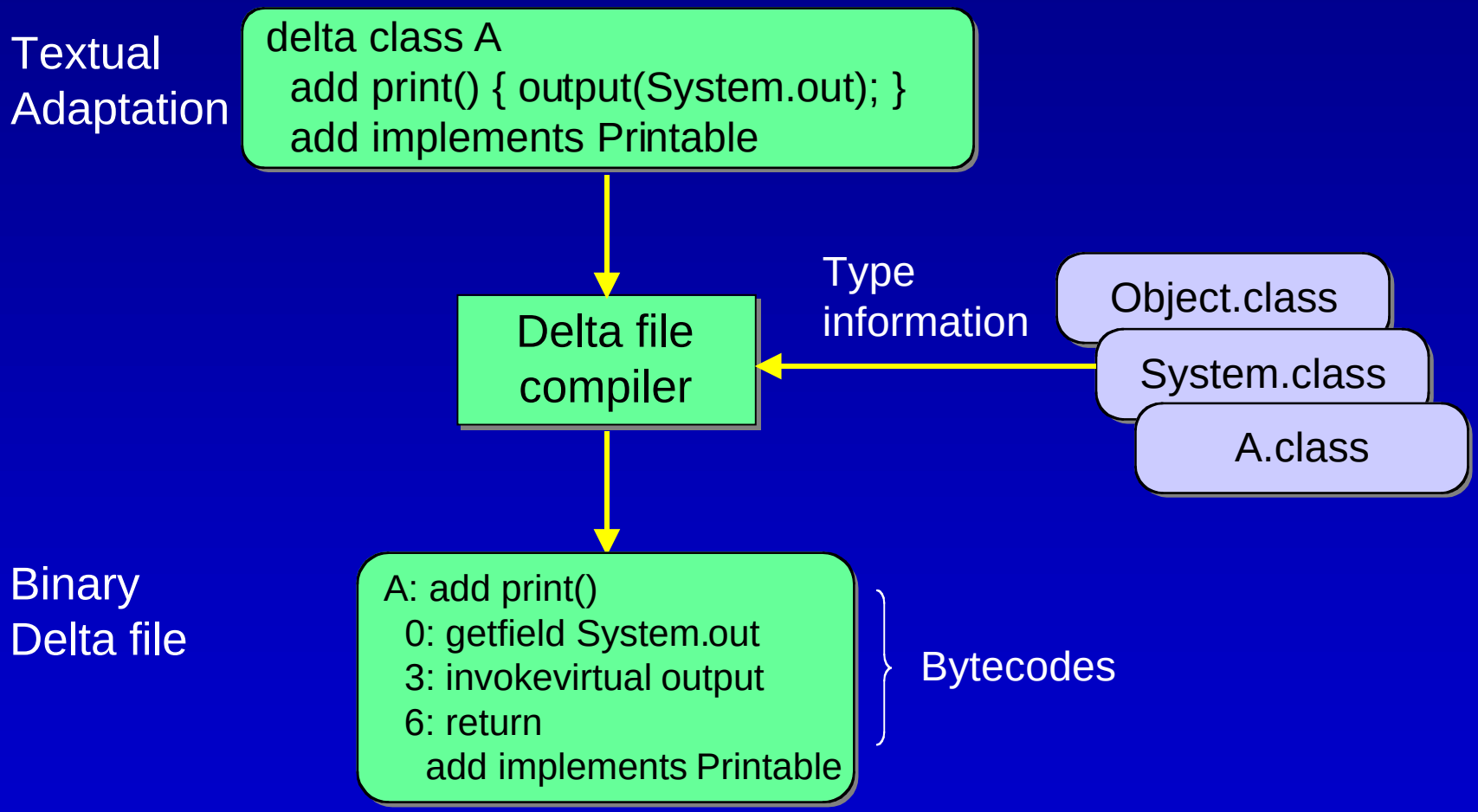
BCA could significantly improve reusability of components

Further Information

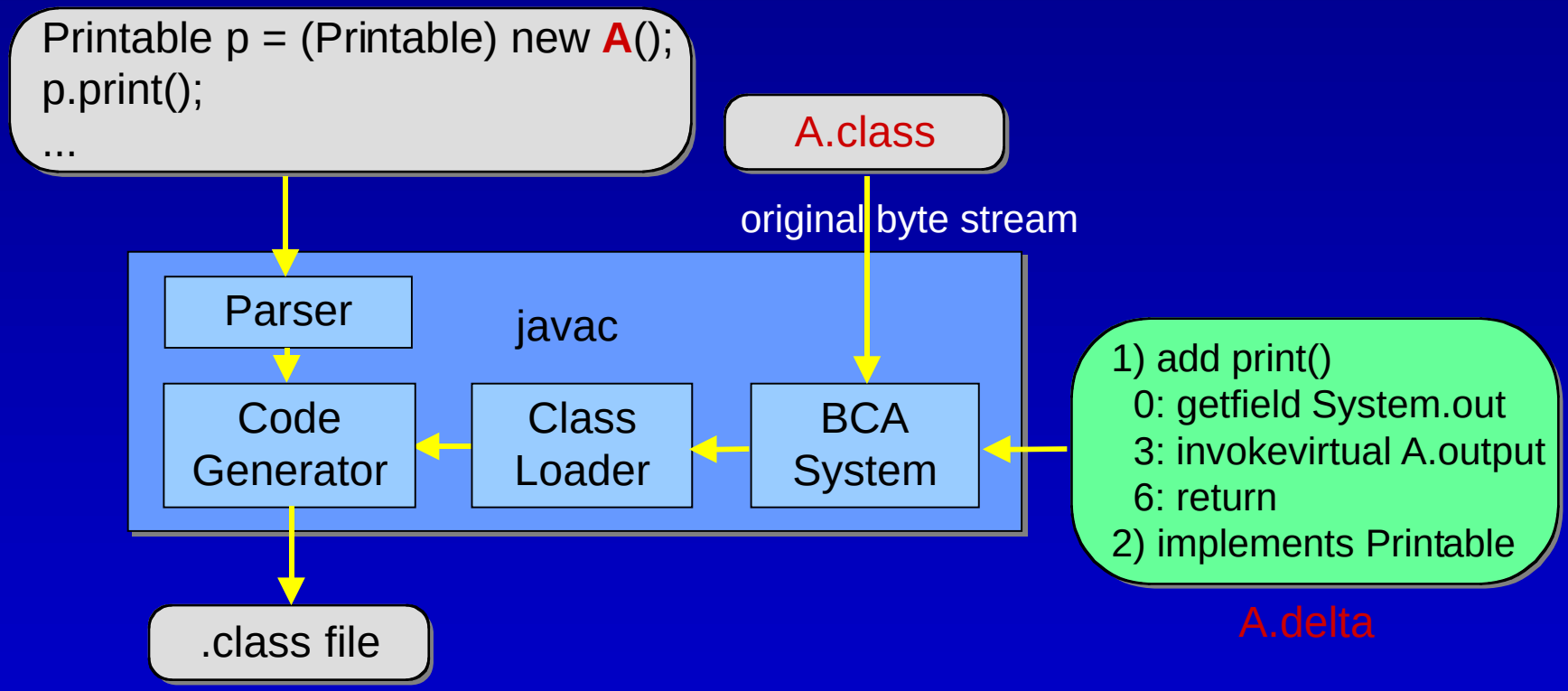
- World Wide Web
  - http://www.cs.ucsb.edu/oocsb/bca

Backup Slides

Delta File Compilation

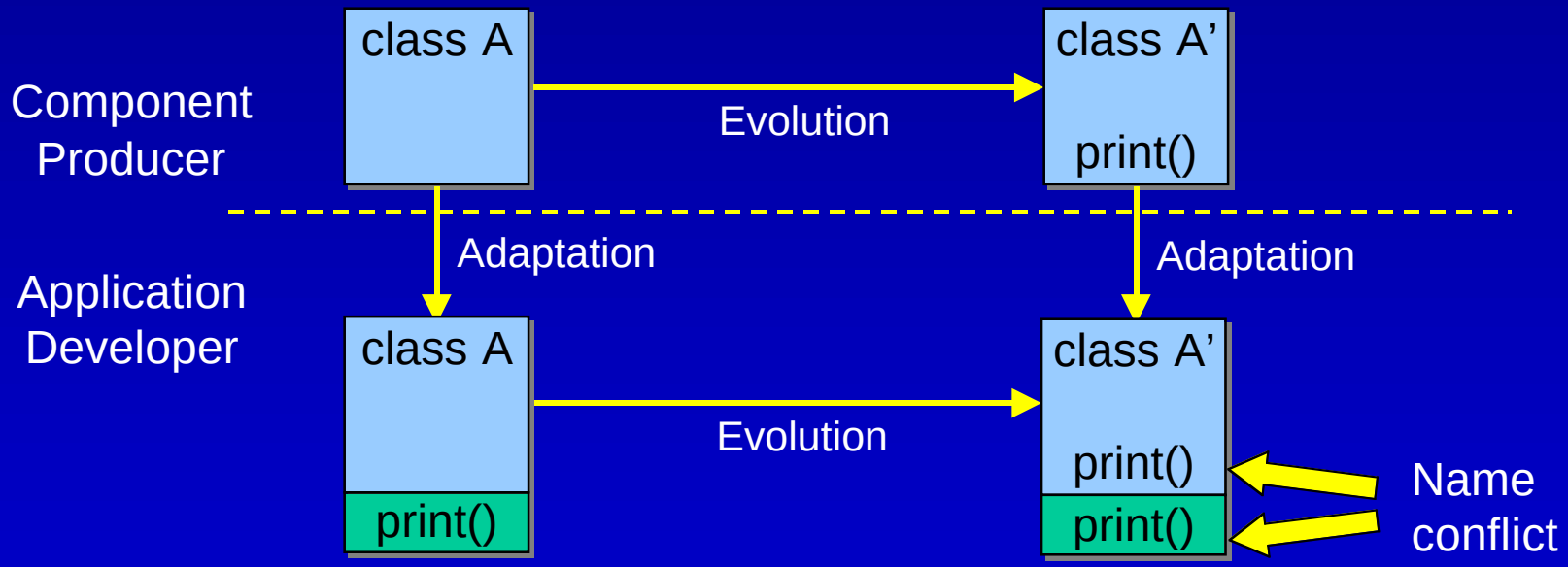


Compiling Against Adapted Classes



Enabling Binary Compatibility

- Problem: Changes between Evolution and Adaptation may lead to name conflicts



Solution: Mark Class Files

- Mark class files compiled against adapted class
- Conflicts resolved by renaming