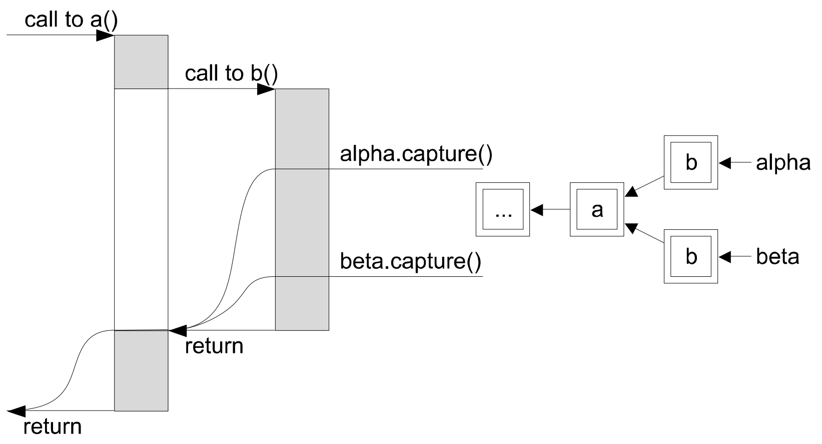
Lazy Continuations for Java Virtual Machines



Lukas Stadler

Johannes Kepler University Linz, Austria

Agenda

- Continuations
- Uses for continuations
- Common implementation techniques
- Our *lazy* approach
- Implementation
- Summary

Continuations

- Functional / dynamic languages
- “the rest of the computation”
- “everything thats going to happen from now on”
- In Java terminology: (part of) the contents of the stack of activation frames
  (method, bci, variables, expressions)
- Can be stored
- Can be reinstated (possibly more than once)
- Different types with different semantics

Continuations

```
Continuation alpha;

void method() {
    int value = 0;
    alpha.capture();
    System.out.println("current value: " + value);
    value += 1;
    alpha.resume();
}
```

Continuations

```java
Continuation alpha;

void method() {
    int value = 0;
    alpha.capture();
    System.out.println("current value: " + value);
    value += 1;
    alpha.resume();
}
```

```
current value: 0
current value: 0
current value: 0
current value: 0
...
...
```

Uses for continuations

- Functional languages: basic language features
  - return, exception handling, etc.
- Java: advanced features
  - green threads, coroutines, fibers, etc.
- Web servers
  - linearize complex interactions
  - “back button” problem
- Checkpointing, portable agents, etc.

Common Techniques

- One-shot continuations (via exceptions)
- Activation frames as objects (Smalltalk)
- Segments containing many activation frames allocated on heap
  (some Scheme environments)
- Most implementations: *Copy-all* approach

Common Techniques

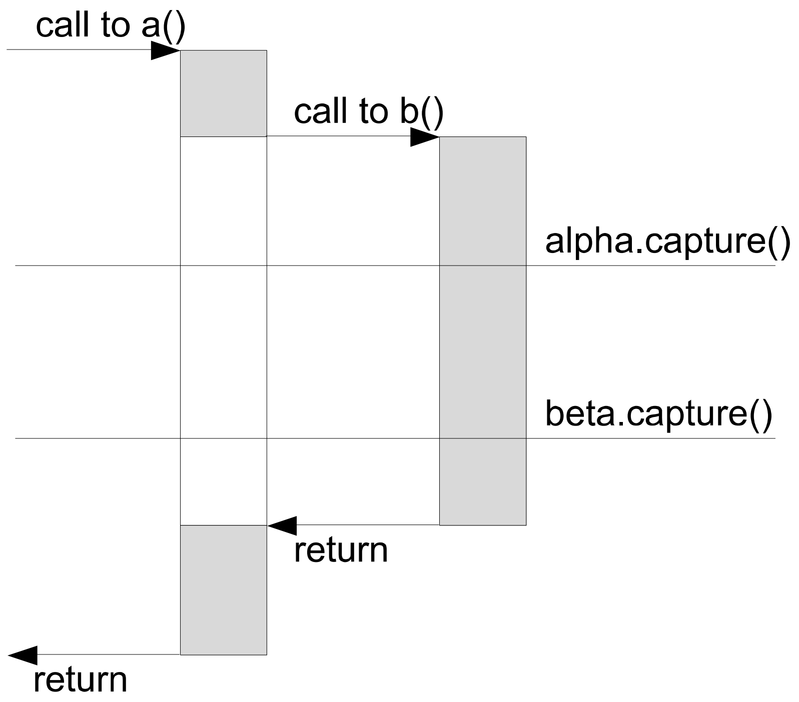
- Copy-all approach: example

```
Continuation alpha;
Continuation beta;

void a() {
    b();
}
void b() {
    alpha.capture();
    beta.capture();
}
```

Common Techniques

- Copy-all approach: example

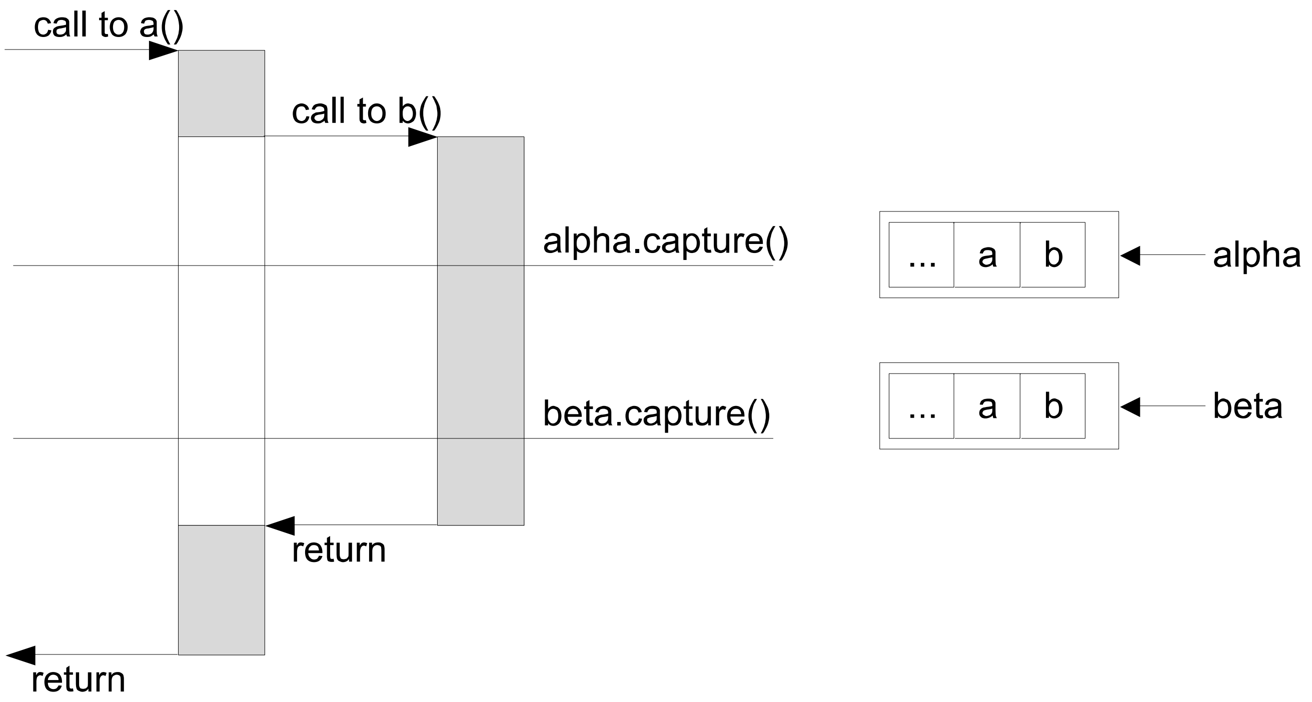
```
Continuation alpha;
Continuation beta;

void a() {
    b();
}
void b() {
    alpha.capture();
    beta.capture();
}
```



Common Techniques

- Copy-all approach: example

```
Continuation alpha;
Continuation beta;

void a() {
    b();
}
void b() {
    alpha.capture();
    beta.capture();
}
```



Common Techniques

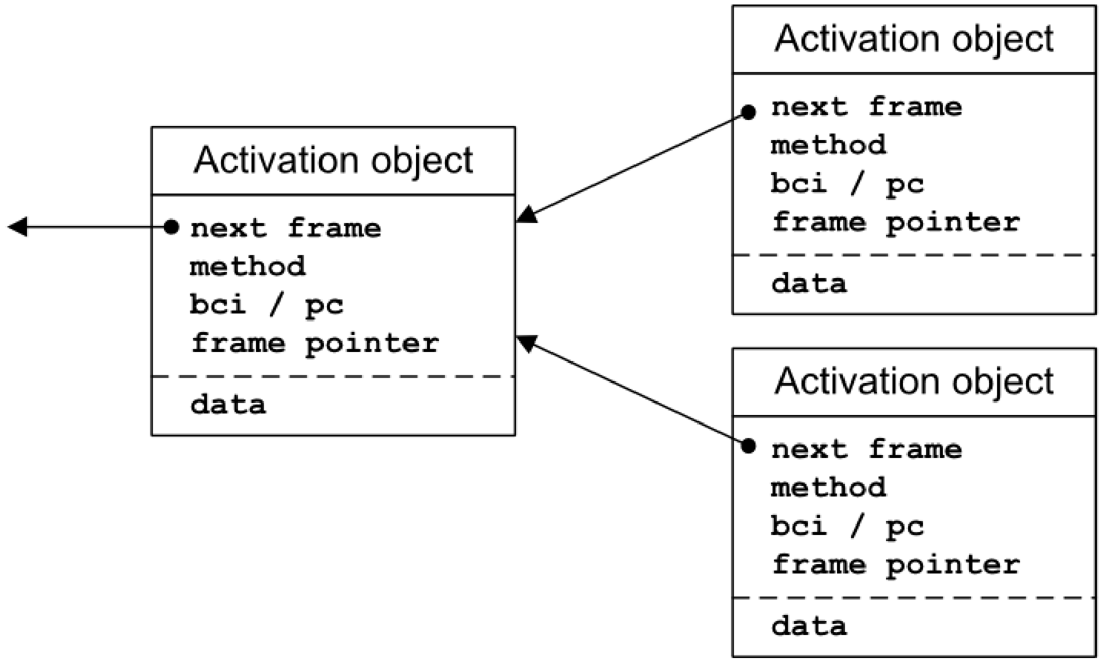
- Problems:
  - Immediate cost of continuation capture
  - Continuations often share activation frames
  - No way to tell if an activation frame needs to be restored
- Be Lazy!

Lazy Continuations

- Store activation frames as late as possible
- Intercept the return to an activation frame by patching the return address
- Call site - specific trampoline
- One Object per activation frame (called activation object): linked list
- The activation object for the next activation frame stored in the thread

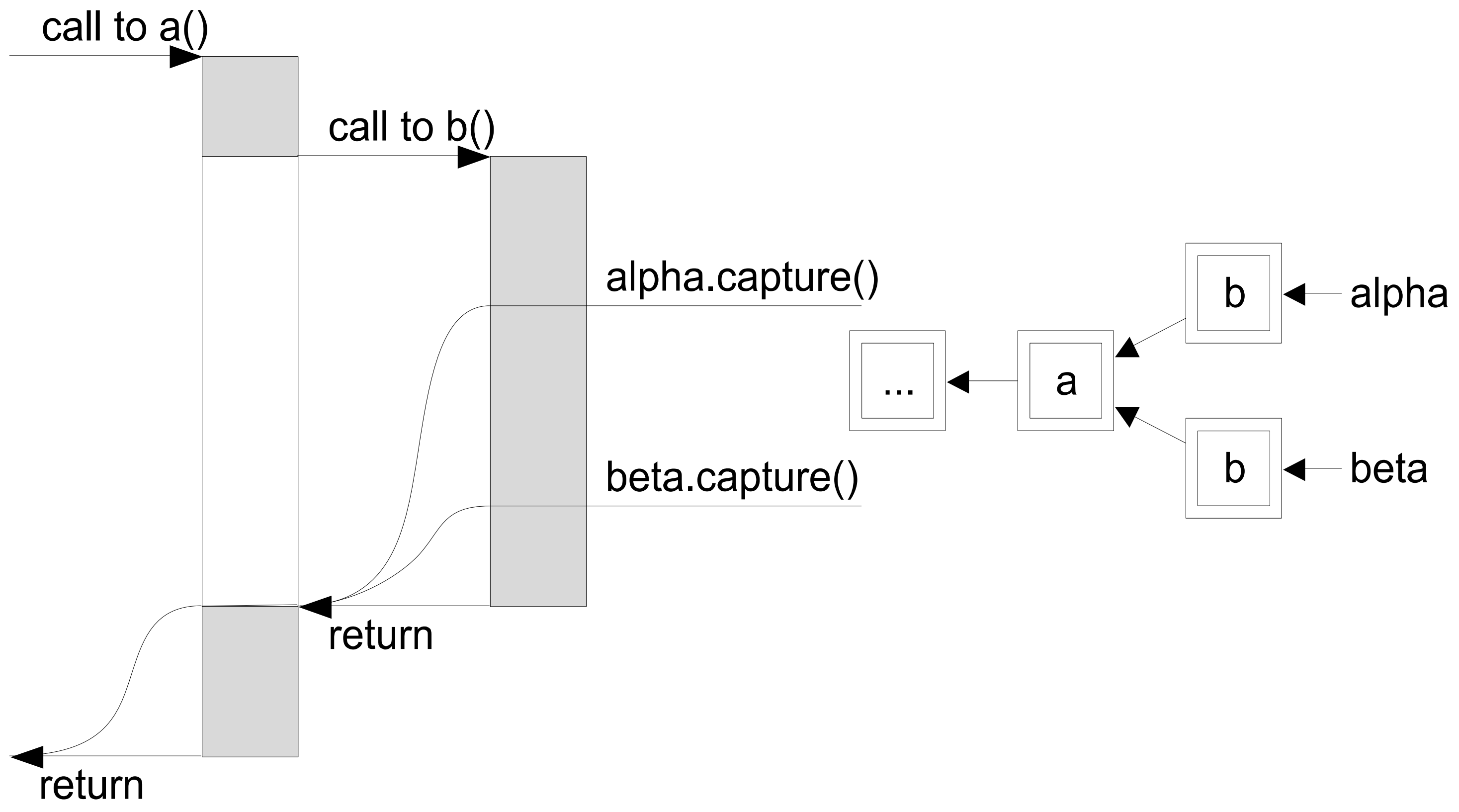
Lazy Continuations

- Continuations joined into tree structure



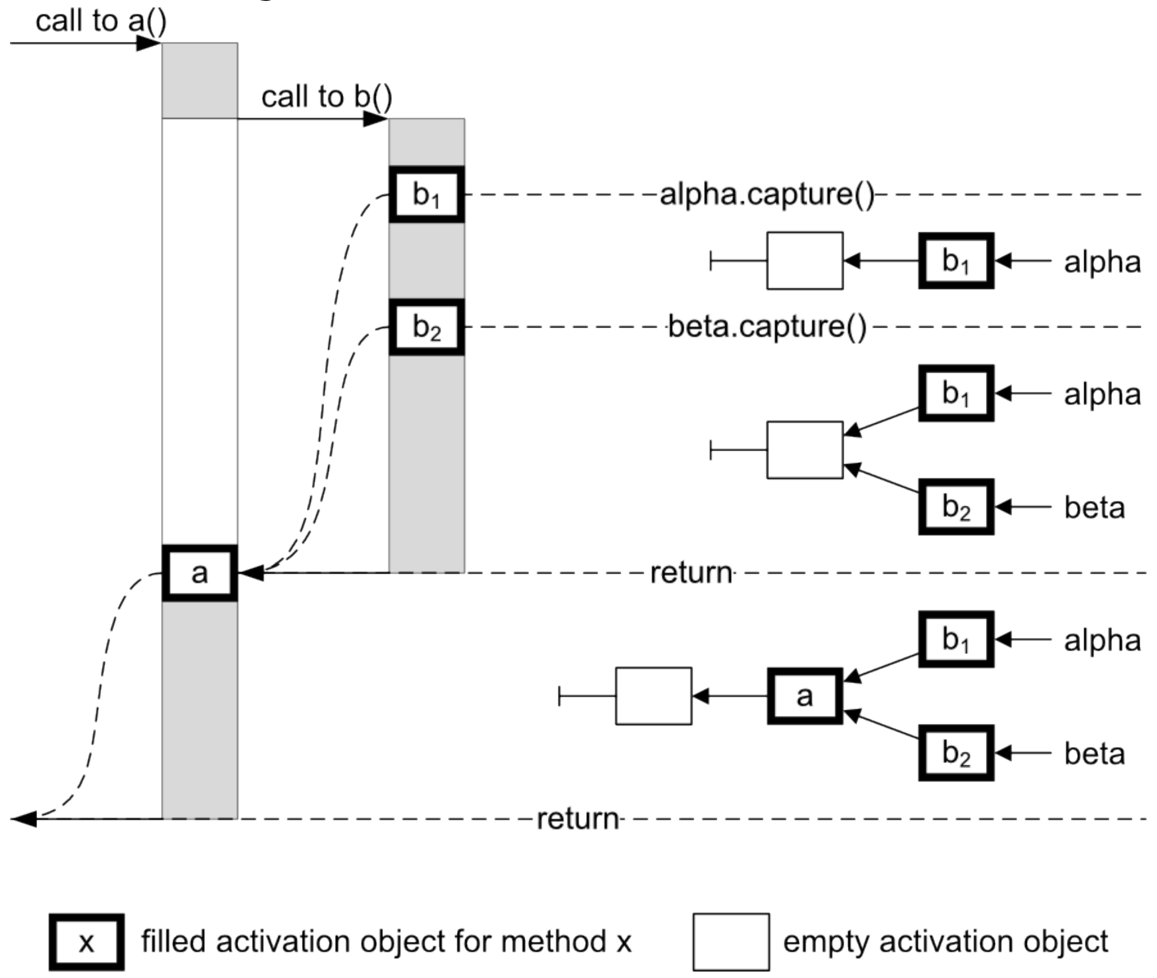
Lazy Continuations

- Continuations joined into tree structure



Lazy Continuations

- Continuations joined into tree structure



Java Interface

```
public class Continuation {
    public static final Object CAPTURED;
    public native Object capture();
    public native void resume(Object retVal);
}

public @interface Continuable {
}
```

- Passing a return value on resume
- Annotation to mark methods continuation - safe

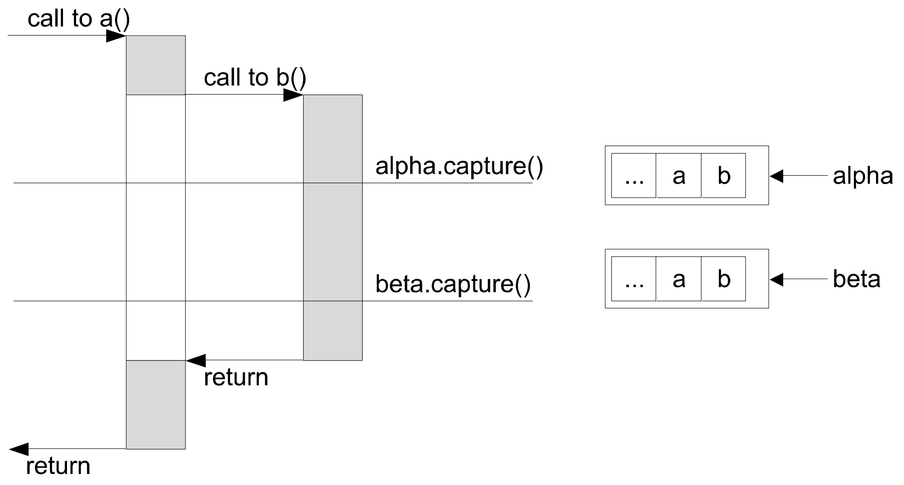
Java Interface

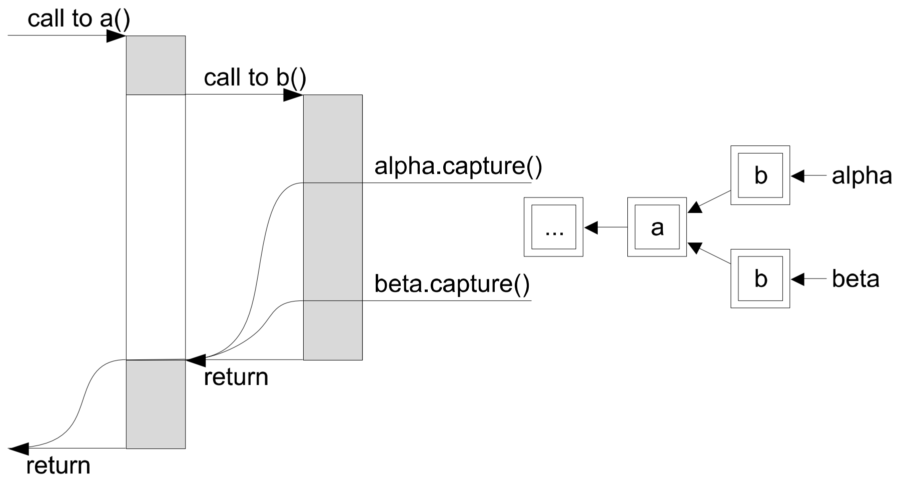
```
class Test {
    Continuation alpha = new Continuation();

    @Continuable
    public static void main() {
        System.out.println(“start”);
        if (alpha.capture() == Continuation.CAPTURED) {
            System.out.println(“captured”);
            alpha.resume(null);
        } else {
            System.out.println(“resumed”);
        }
        System.out.println(“end”);
    }
}
```

```
start
captured
resumed
end
```

Summary

- Saves time
- Saves memory
  (break even at ~30%)





Future

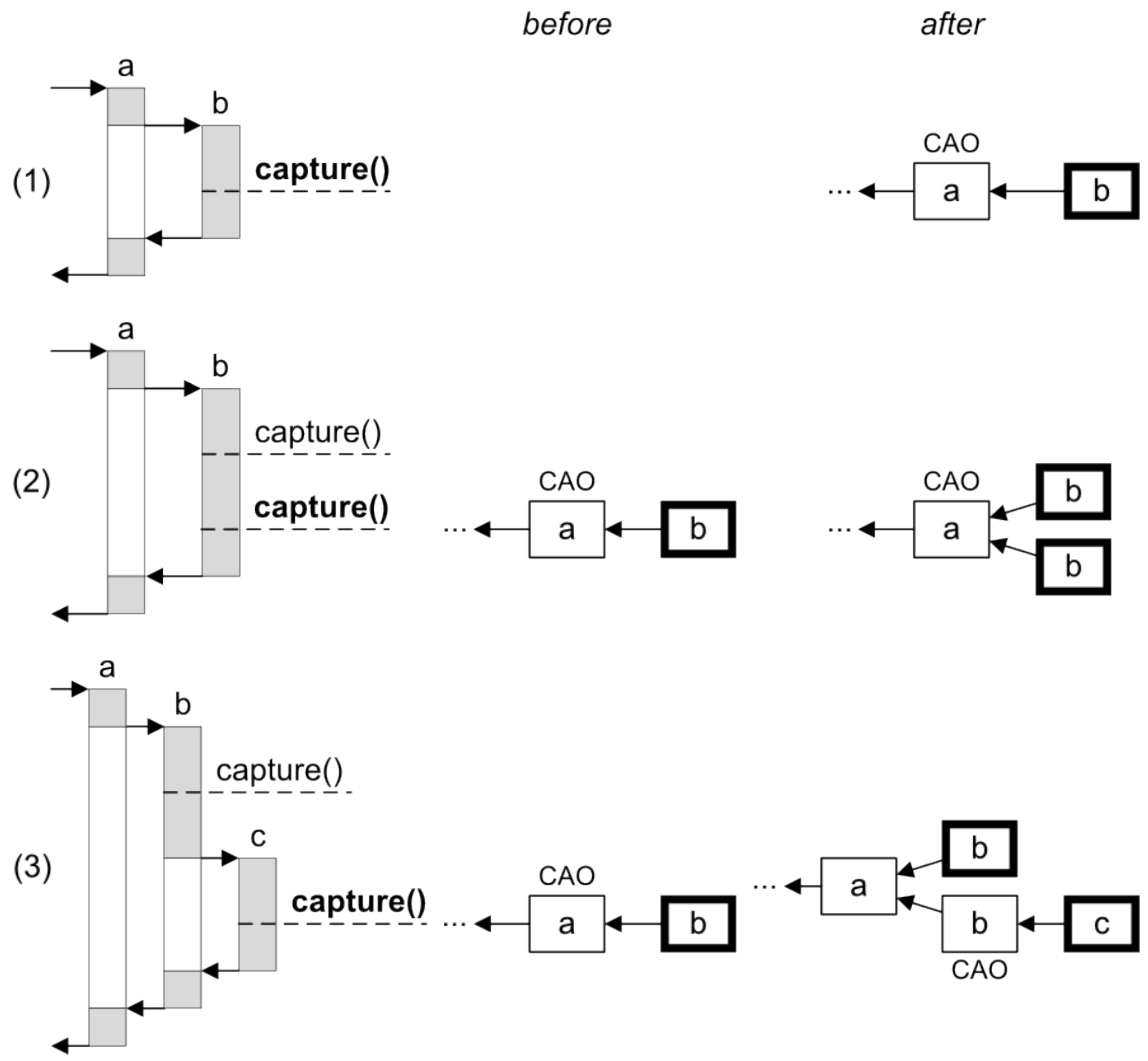
- Serialization
- Good for general case – what about special cases?
- Dealing with Java Security Model
  - Continuations break JVM assumptions

Thank you. Questions?

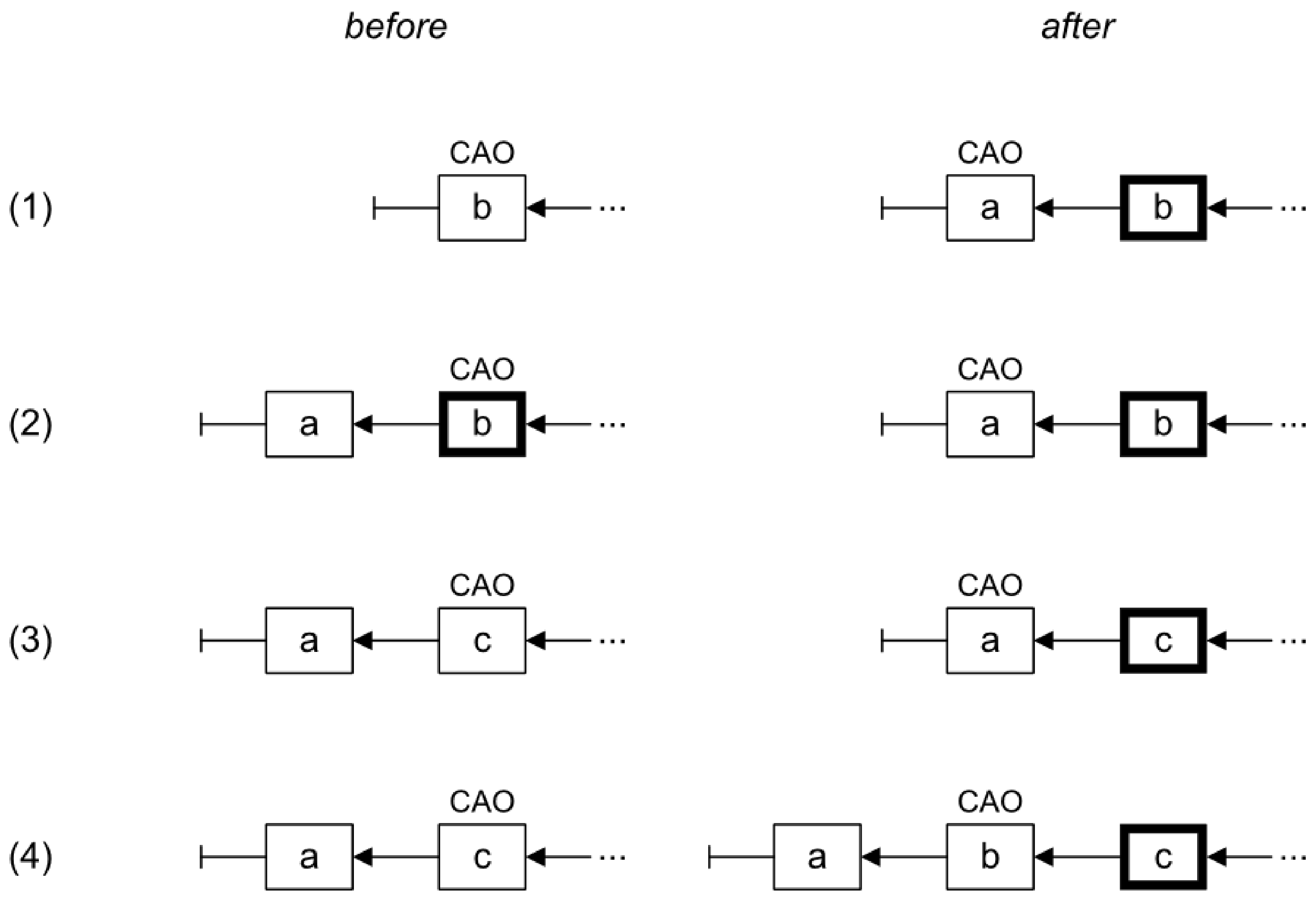
Lazy Continuations for Java Virtual Machines

Lukas Stadler
Johannes Kepler University Linz, Austria

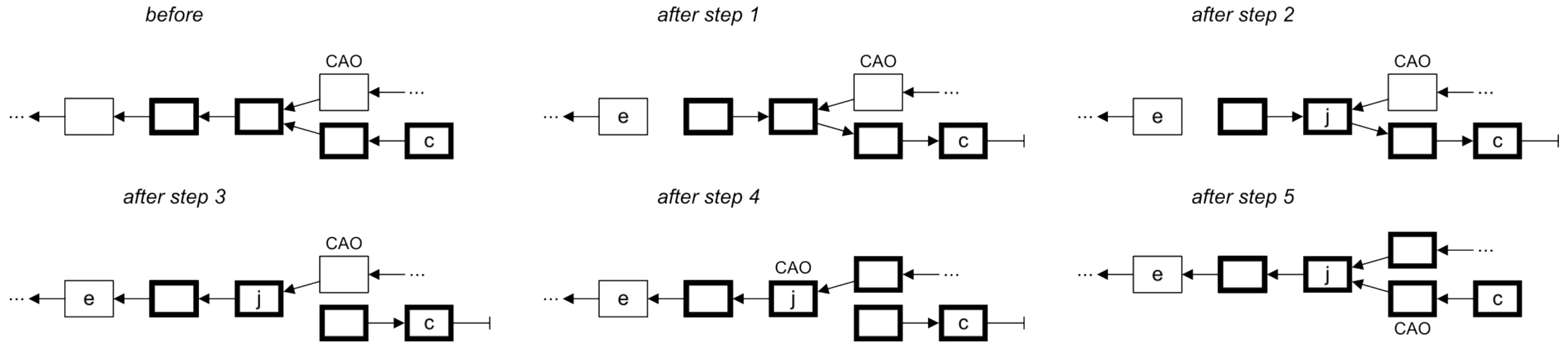
Copy cases


before
after
(1)
a
b
capture()
CAO
a
b
(2)
capture()
capture()
CAO
a
b
(3)
c
capture()
capture()
CAO
a
b
c
...

Frame storing cases



Resume cases



KPS 2009