# SSA Form for the Java HotSpot™ Client Compiler

Christian Wimmer
cwimmer@uci.edu
www.christianwimmer.at

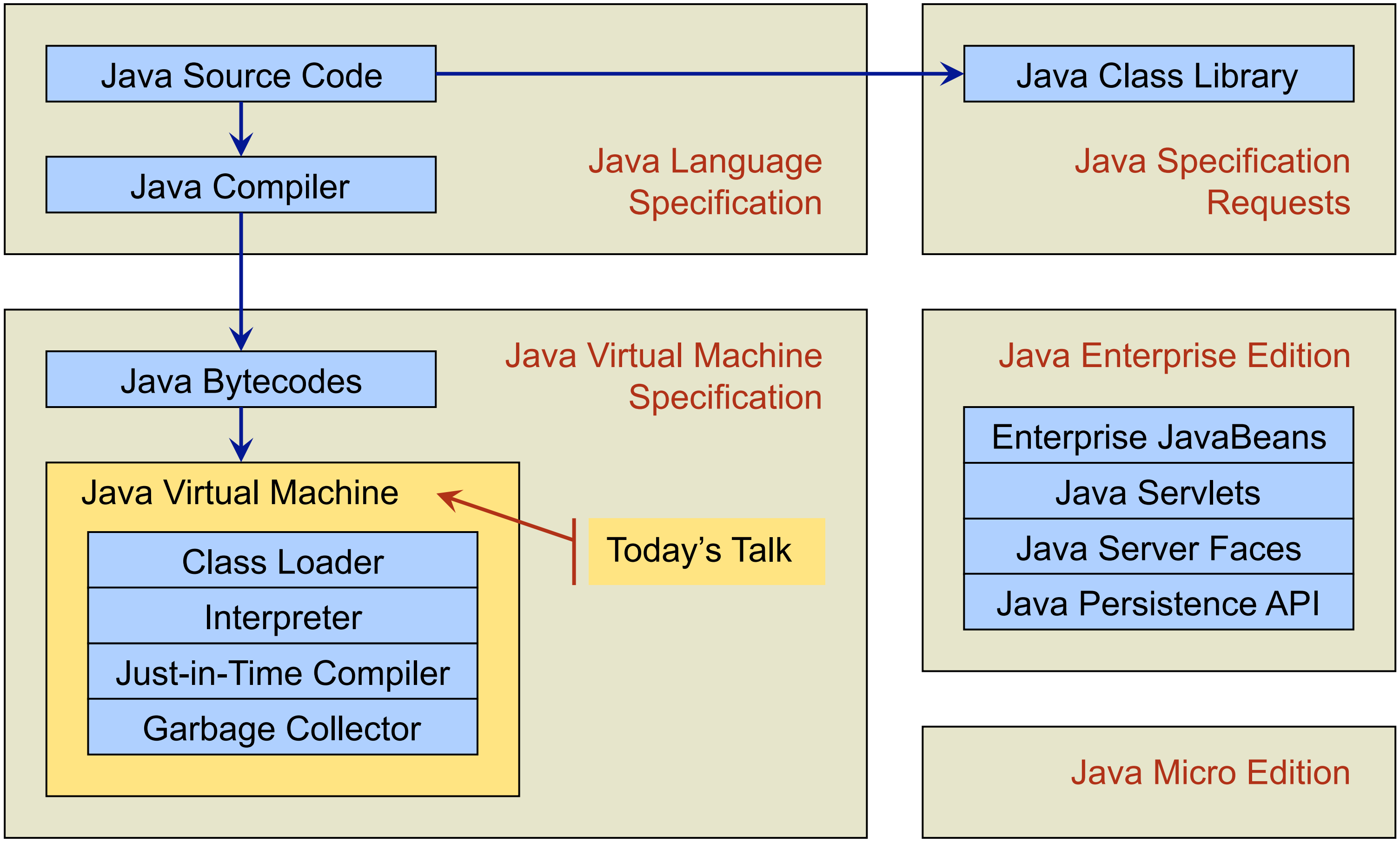
April 2009

Department of Computer Science
University of California, Irvine

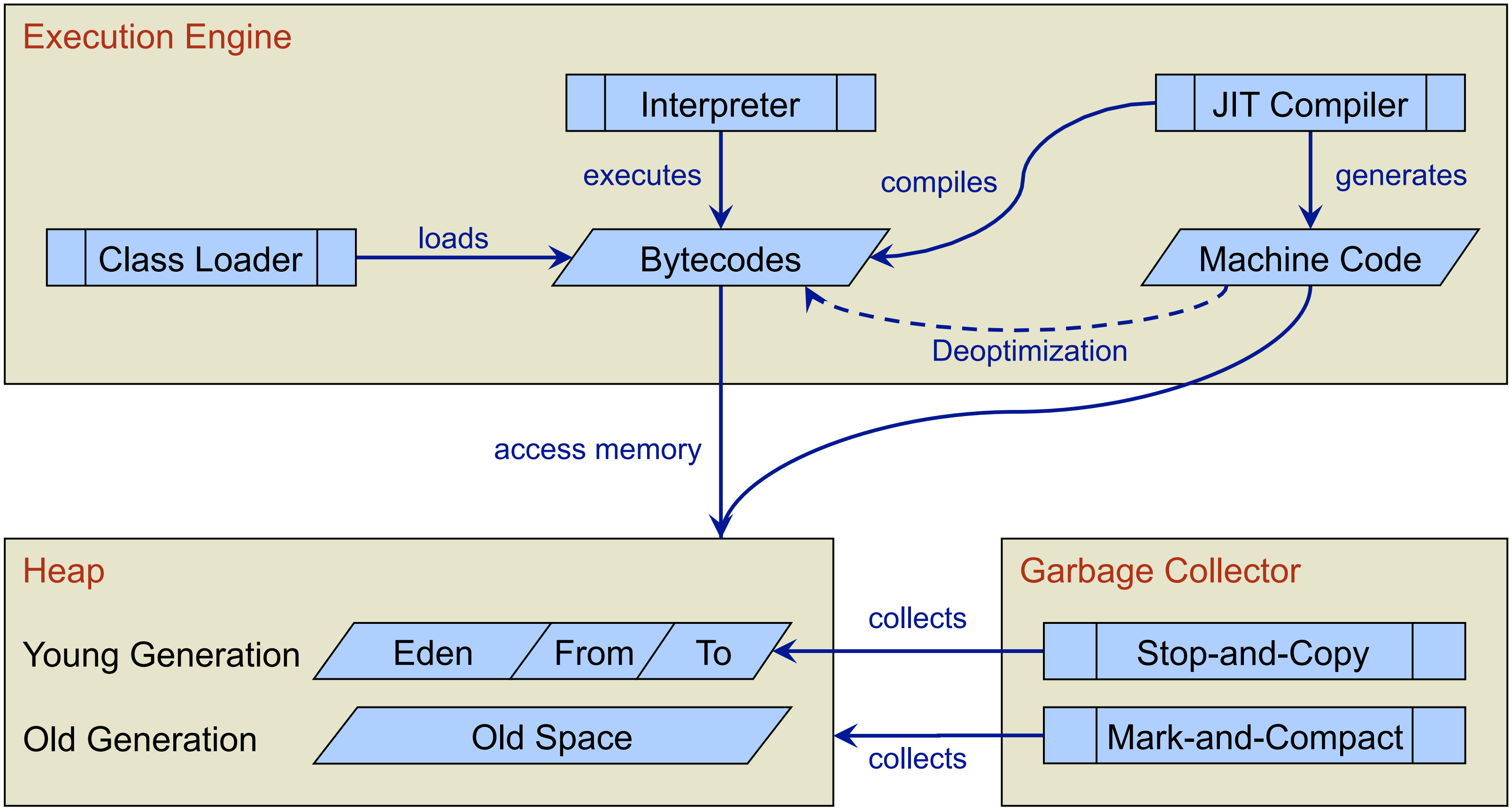
Institute for System Software
Johannes Kepler University Linz, Austria

Sun Microsystems, Inc.

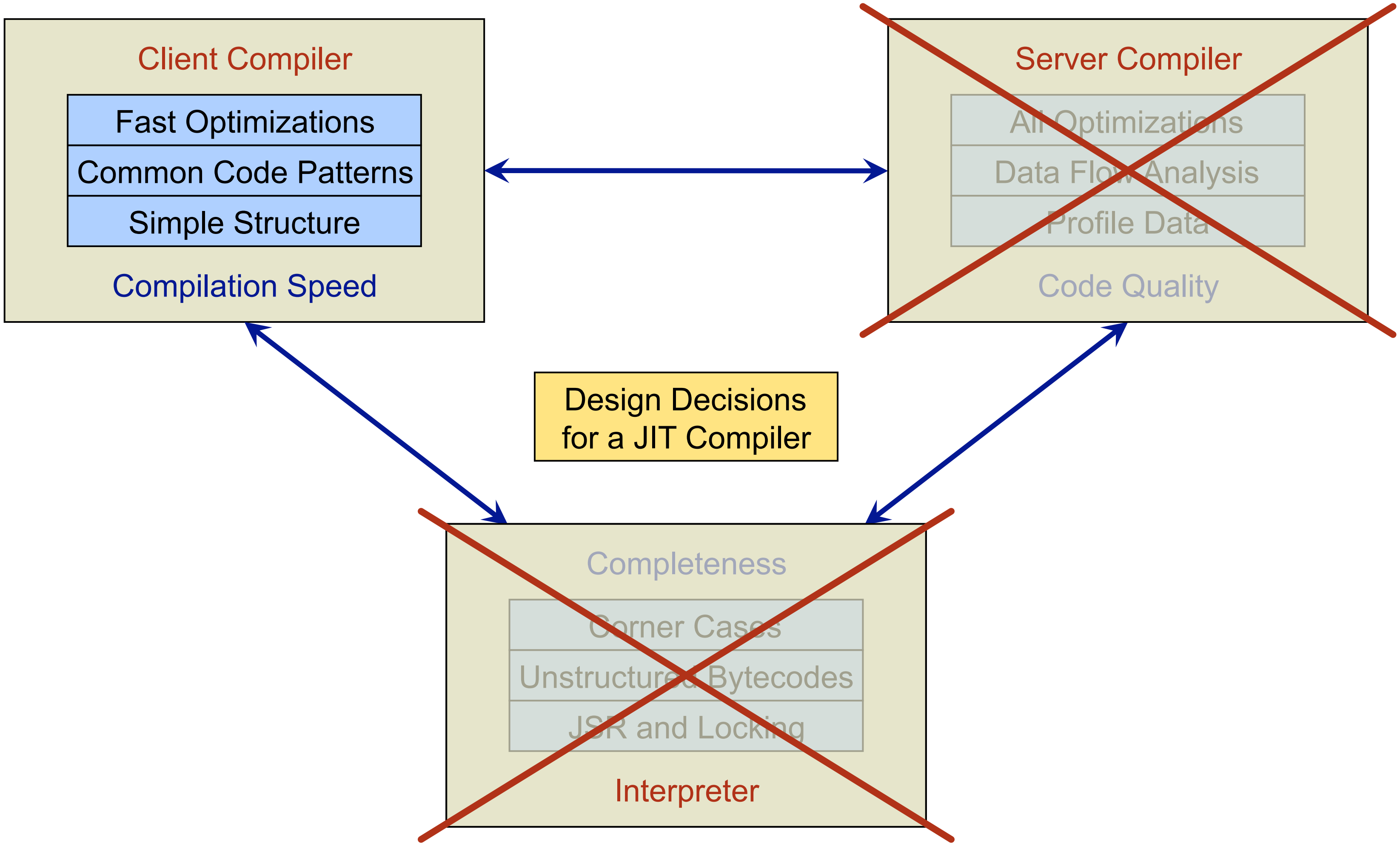
# Java is also an island…



# Java HotSpot™ VM



# Client Compiler – Design



Client Compiler

Fast Optimizations

Common Code Patterns

Simple Structure

Compilation Speed

~~Server Compiler~~

~~All Optimizations~~

~~Data Flow Analysis~~

~~Profile Data~~

~~Code Quality~~

Design Decisions for a JIT Compiler

~~Completeness~~

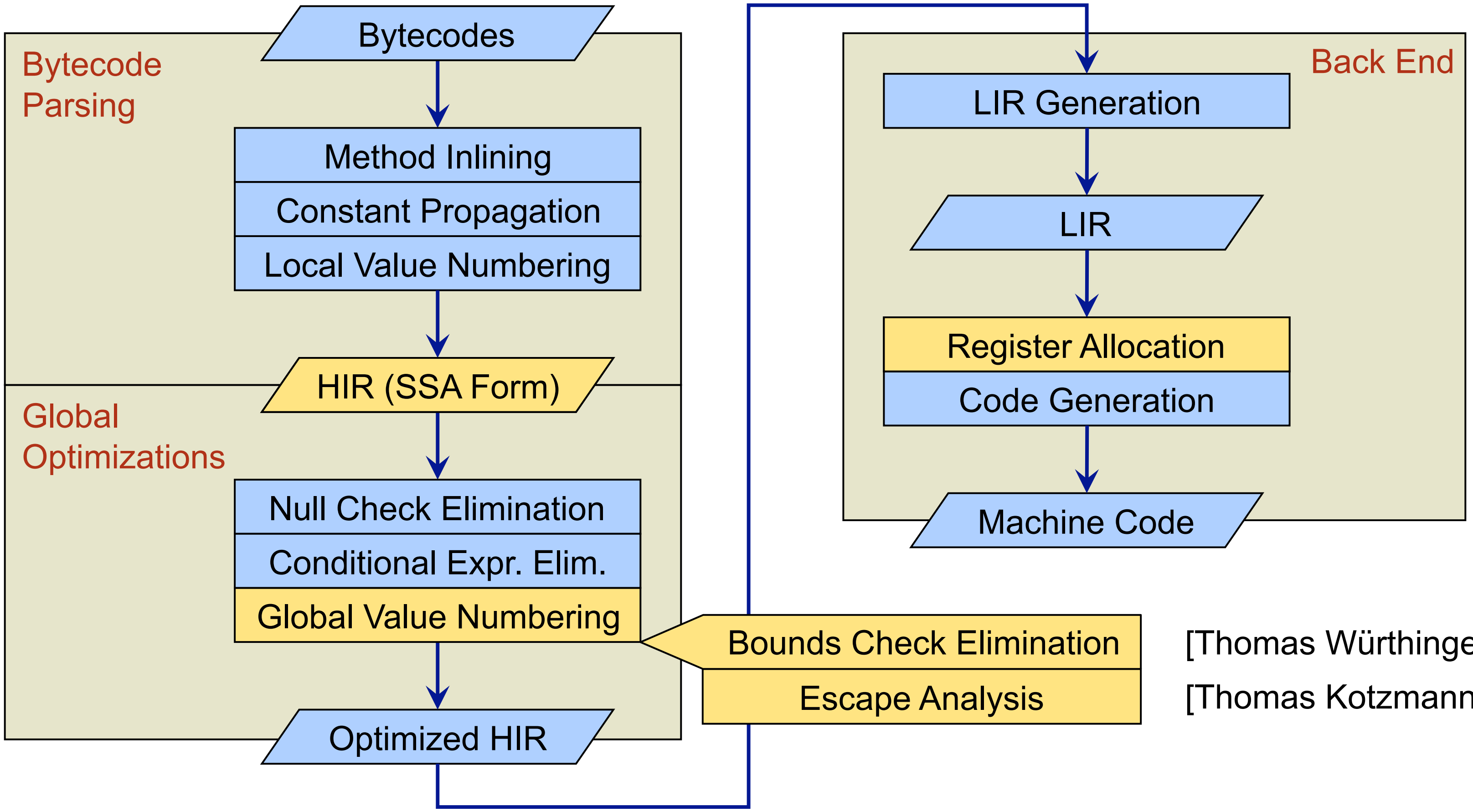
~~Corner Cases~~

~~Unstructured Bytecodes~~

~~JSR and Locking~~

~~Interpreter~~

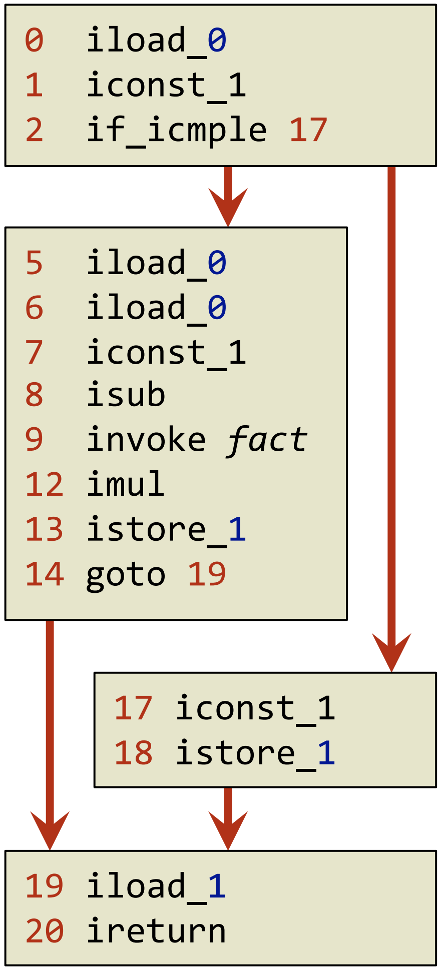
# Client Compiler – Structure



[Thomas Würthinger]

[Thomas Kotzmann]

# Phi Function Placement



```
0  iload_0
1  iconst_1
2  if_icmple 17
```

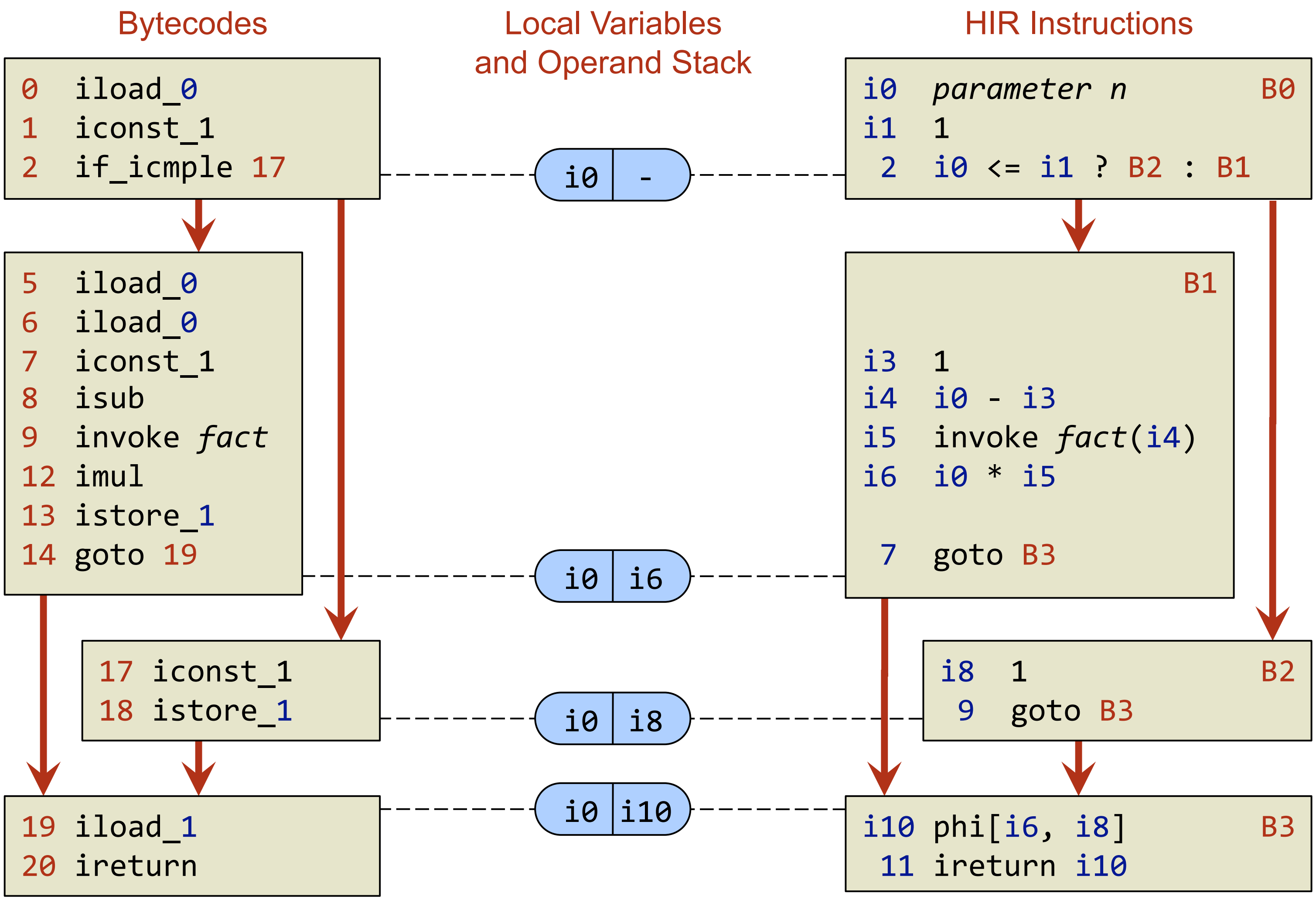
```
5  iload_0
6  iload_0
7  iconst_1
8  isub
9  invoke fact
12 imul
13 istore_1
14 goto 19
```

```
17 iconst_1
18 istore_1
```

```
19 iload_1
20 ireturn
```

```
static int fact(int n) {
  int p;
  if (n > 1) {
    p = n * fact(n - 1);
  } else {
    p = 1;
  }
  return p;
}
```

# Phi Function Placement



Bytecodes

```
0  iload_0
1  iconst_1
2  if_icmple 17
```

```
5  iload_0
6  iload_0
7  iconst_1
8  isub
9  invoke fact
12 imul
13 istore_1
14 goto 19
```

```
17 iconst_1
18 istore_1
```

```
19 iload_1
20 ireturn
```

Local Variables and Operand Stack

i0 | -

i0 | i6

i0 | i8

i0 | i10

HIR Instructions

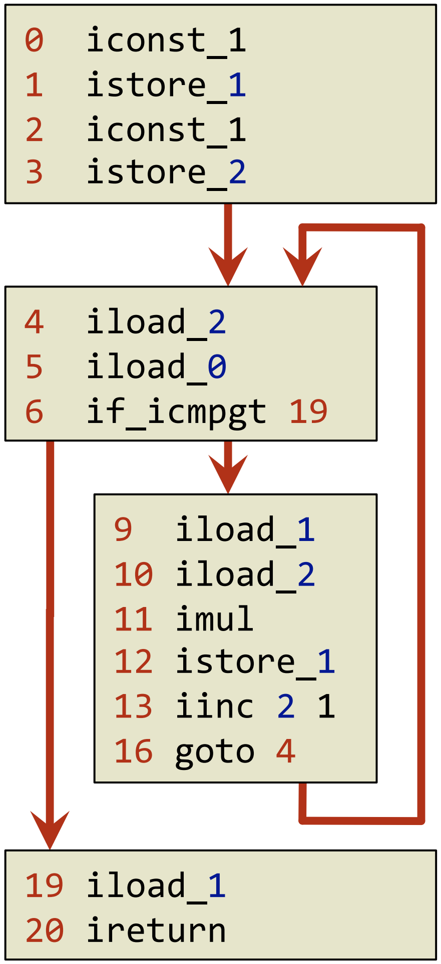
```
i0  parameter n             B0
i1  1
 2  i0 <= i1 ? B2 : B1
```

```
                       B1
i3  1
i4  i0 - i3
i5  invoke fact(i4)
i6  i0 * i5

 7  goto B3
```

```
i8  1                   B2
 9  goto B3
```

```
i10 phi[i6, i8]         B3
 11 ireturn i10
```

# Phi Function Placement – Loops



```
0  iconst_1
1  istore_1
2  iconst_1
3  istore_2
```

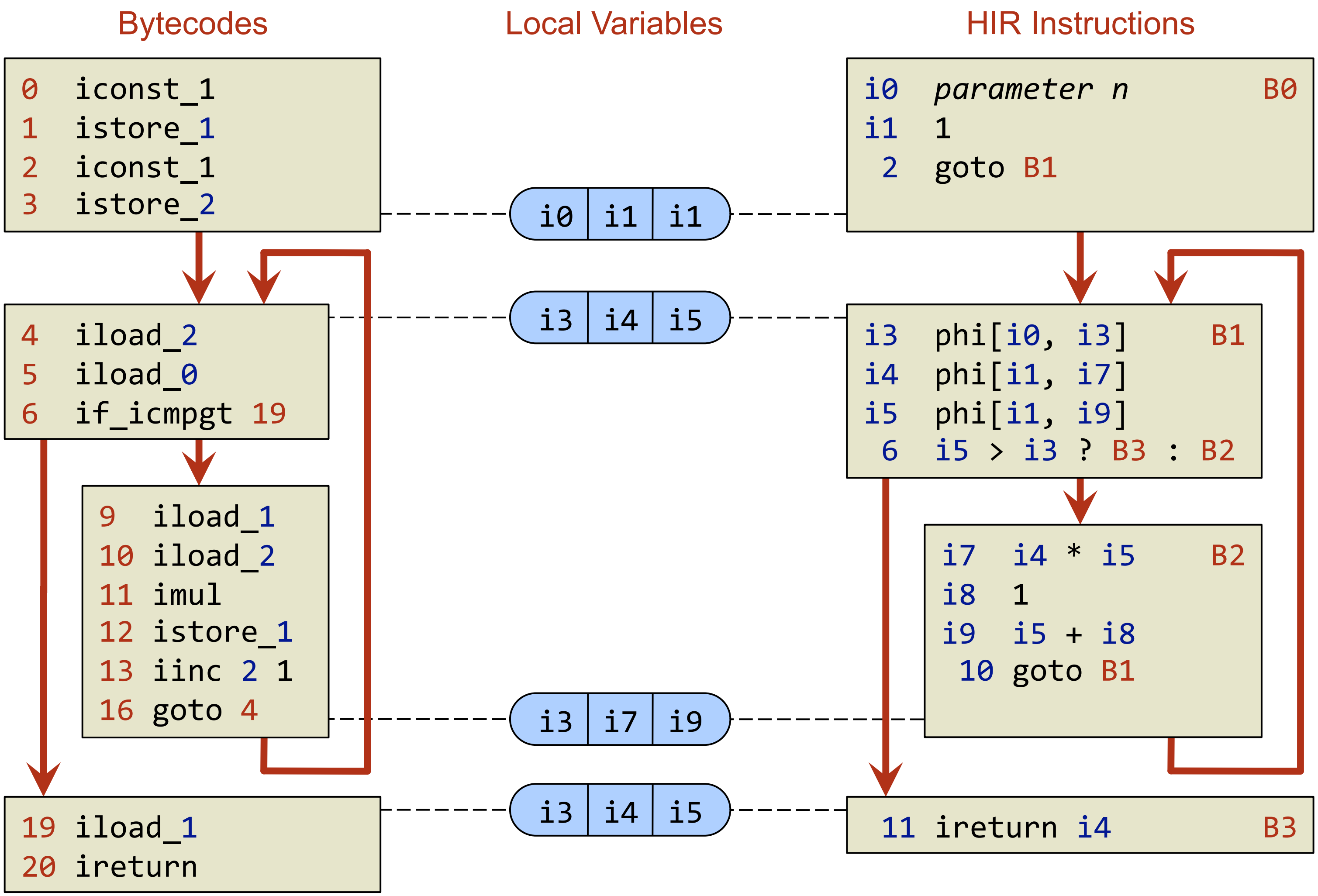
```
4  iload_2
5  iload_0
6  if_icmpgt 19
```

```
9  iload_1
10 iload_2
11 imul
12 istore_1
13 iinc 2 1
16 goto 4
```

```
19 iload_1
20 ireturn
```

```
static int fact(int n) {
  int p = 1;
  for (int i = 1; i <= n; i++) {
    p = p * i;
  }
  return p;
}
```

# Phi Function Placement – Loops



Bytecodes

```
0  iconst_1
1  istore_1
2  iconst_1
3  istore_2
```

```
4  iload_2
5  iload_0
6  if_icmpgt 19
```

```
9  iload_1
10 iload_2
11 imul
12 istore_1
13 iinc 2 1
16 goto 4
```

```
19 iload_1
20 ireturn
```

Local Variables

```
i0 | i1 | i1
i3 | i4 | i5
i3 | i7 | i9
i3 | i4 | i5
```

HIR Instructions

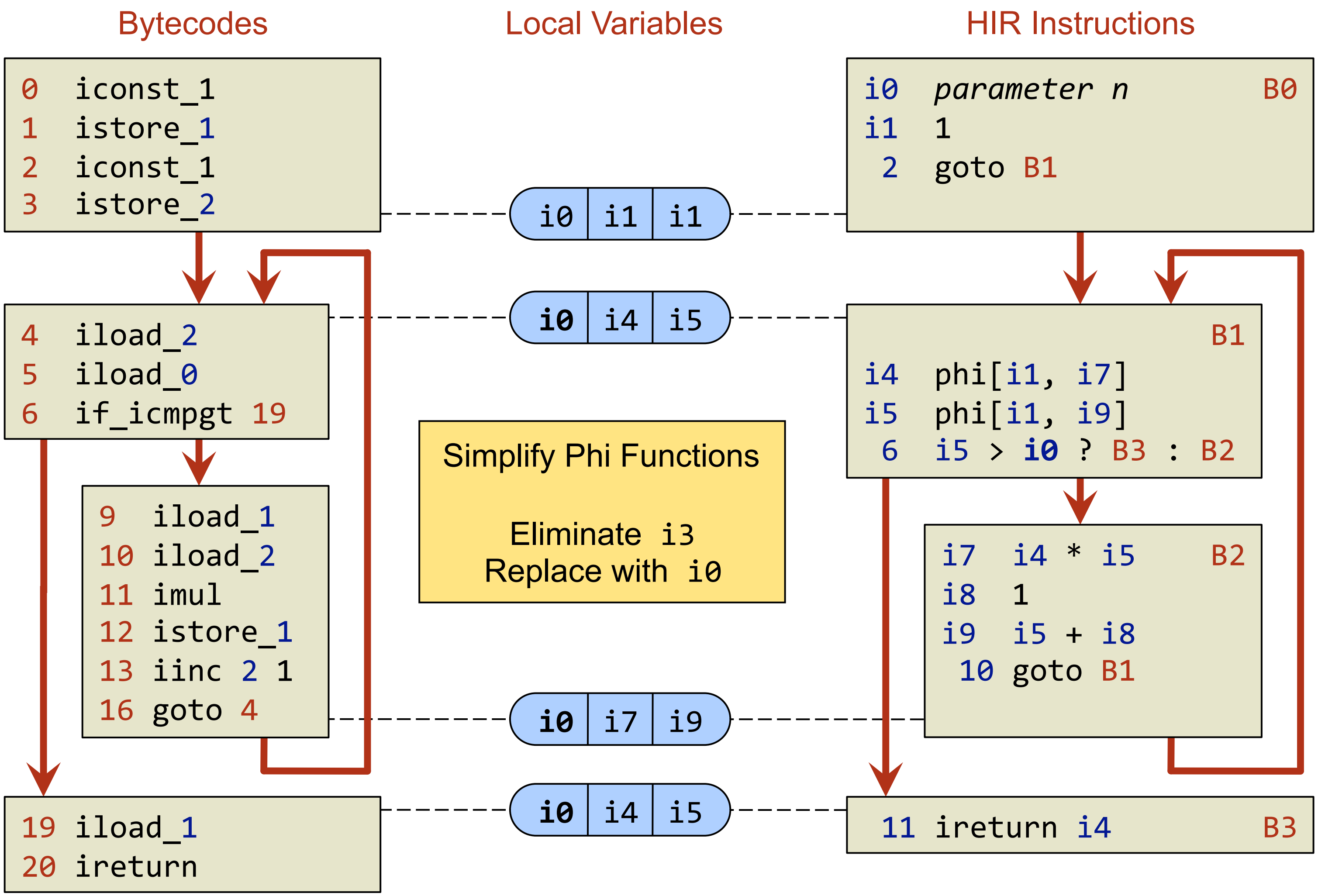
```
i0  parameter n         B0
i1  1
 2  goto B1
```

```
i3  phi[i0, i3]      B1
i4  phi[i1, i7]
i5  phi[i1, i9]
 6  i5 > i3 ? B3 : B2
```

```
i7  i4 * i5      B2
i8  1
i9  i5 + i8
 10 goto B1
```

```
 11 ireturn i4          B3
```

# Phi Function Placement – Loops



Bytecodes

```
0  iconst_1
1  istore_1
2  iconst_1
3  istore_2
```

```
4  iload_2
5  iload_0
6  if_icmpgt 19
```

```
9  iload_1
10 iload_2
11 imul
12 istore_1
13 iinc 2 1
16 goto 4
```

```
19 iload_1
20 ireturn
```

Local Variables

| | | |
|---|---|---|
| i0 | i1 | i1 |
| i0 | i4 | i5 |
| i0 | i7 | i9 |
| i0 | i4 | i5 |

Simplify Phi Functions

Eliminate i3
Replace with i0

HIR Instructions

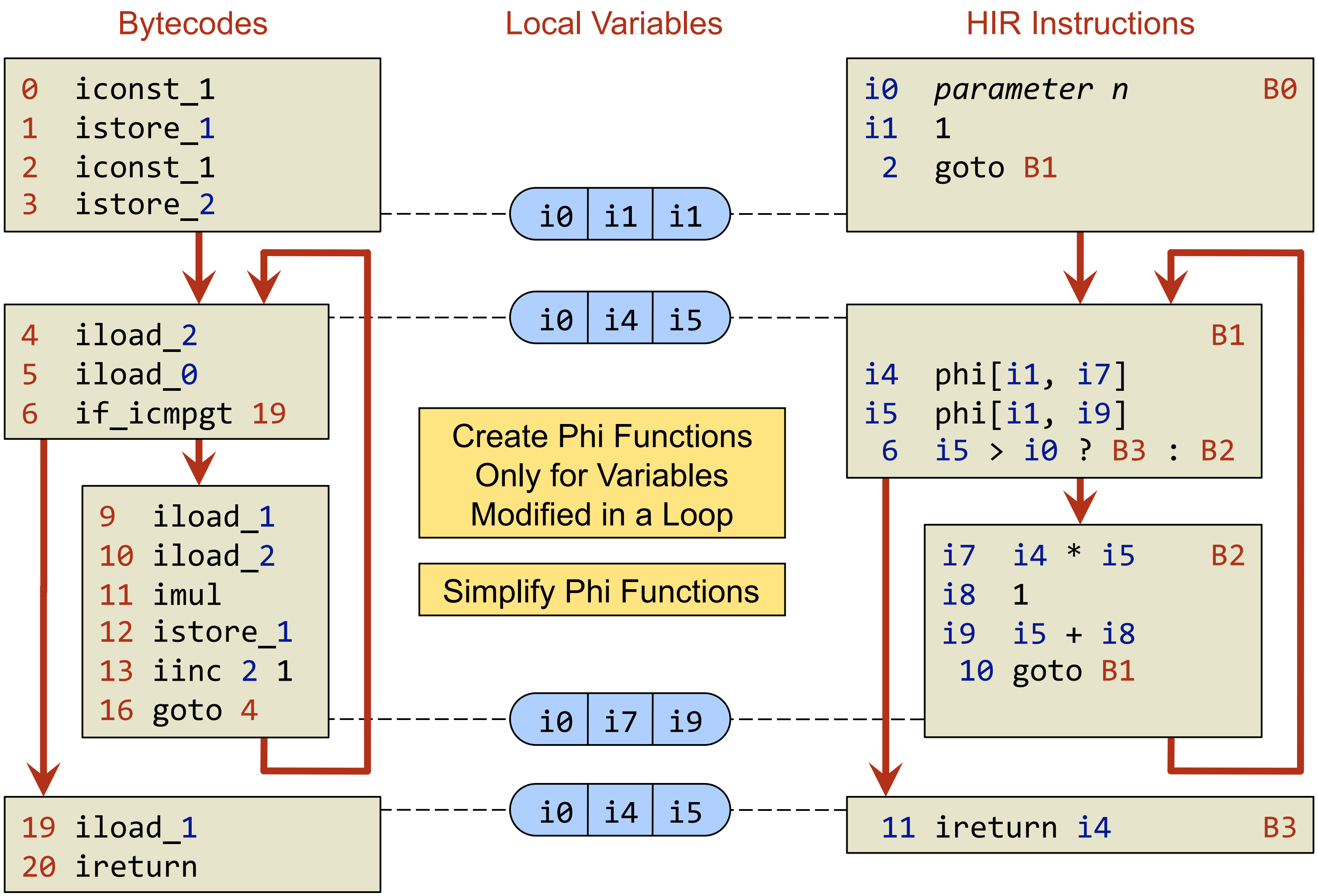
```
i0  parameter n         B0
i1  1
 2  goto B1
```

```
                    B1
i4  phi[i1, i7]
i5  phi[i1, i9]
 6  i5 > i0 ? B3 : B2
```

```
i7  i4 * i5       B2
i8  1
i9  i5 + i8
 10 goto B1
```

```
 11 ireturn i4          B3
```

# Phi Function Placement – Loops



Bytecodes

```
0  iconst_1
1  istore_1
2  iconst_1
3  istore_2
```

```
4  iload_2
5  iload_0
6  if_icmpgt 19
```

```
9  iload_1
10 iload_2
11 imul
12 istore_1
13 iinc 2 1
16 goto 4
```

```
19 iload_1
20 ireturn
```

Local Variables

Create Phi Functions Only for Variables Modified in a Loop

Simplify Phi Functions

HIR Instructions

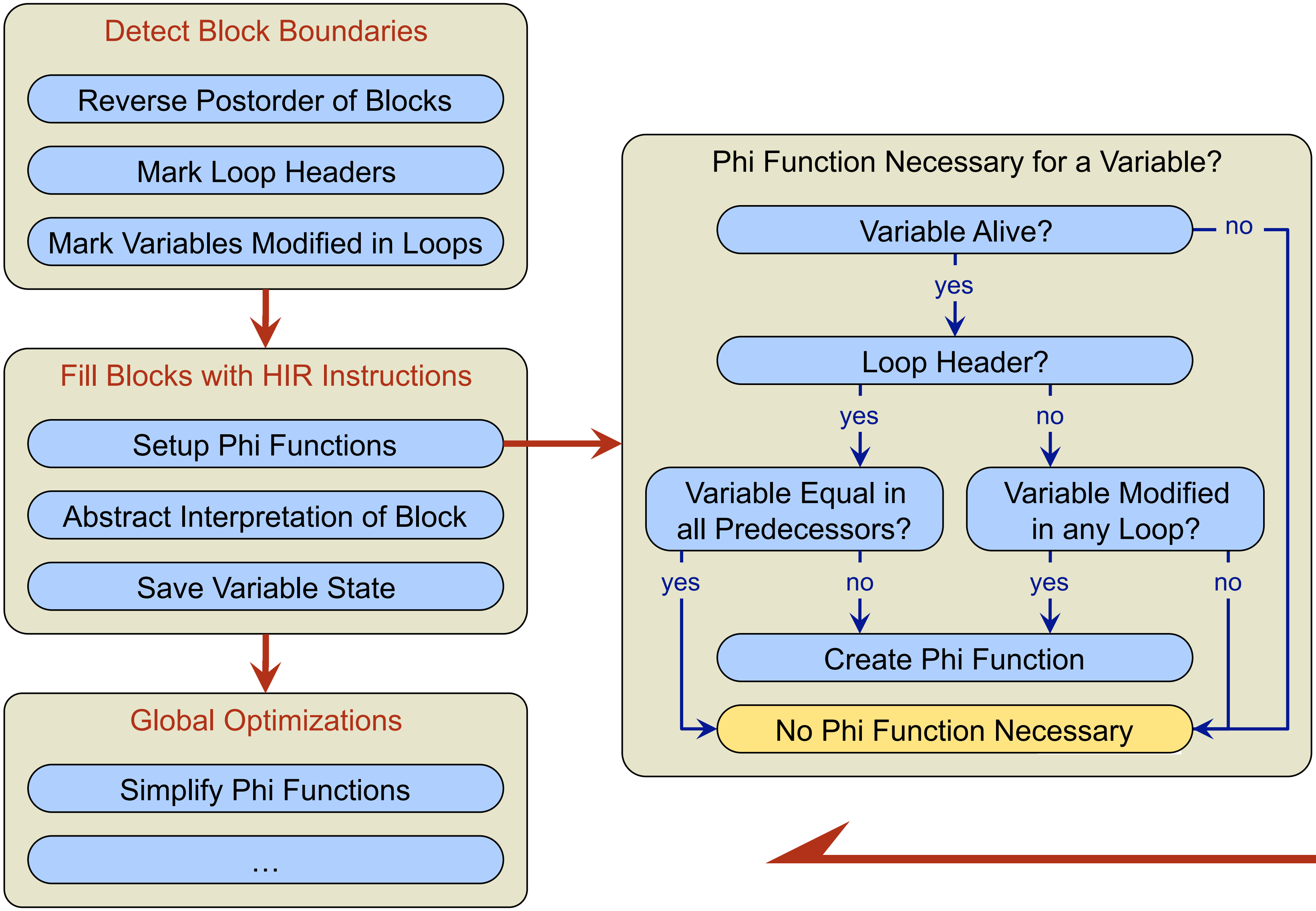
```
i0  parameter n         B0
i1  1
 2  goto B1
```

```
                     B1
i4  phi[i1, i7]
i5  phi[i1, i9]
 6  i5 > i0 ? B3 : B2
```

```
i7  i4 * i5       B2
i8  1
i9  i5 + i8
 10 goto B1
```

```
11 ireturn i4           B3
```

# SSA Construction



Detect Block Boundaries

- Reverse Postorder of Blocks
- Mark Loop Headers
- Mark Variables Modified in Loops

Fill Blocks with HIR Instructions

- Setup Phi Functions
- Abstract Interpretation of Block
- Save Variable State

Global Optimizations

- Simplify Phi Functions
- …

Phi Function Necessary for a Variable?

- Variable Alive?
- Loop Header?
- Variable Equal in all Predecessors?
- Variable Modified in any Loop?
- Create Phi Function
- No Phi Function Necessary

# Phi Function Statistics

One Run of SPECjvm98 (All Benchmarks)

~1000  Methods Compiled

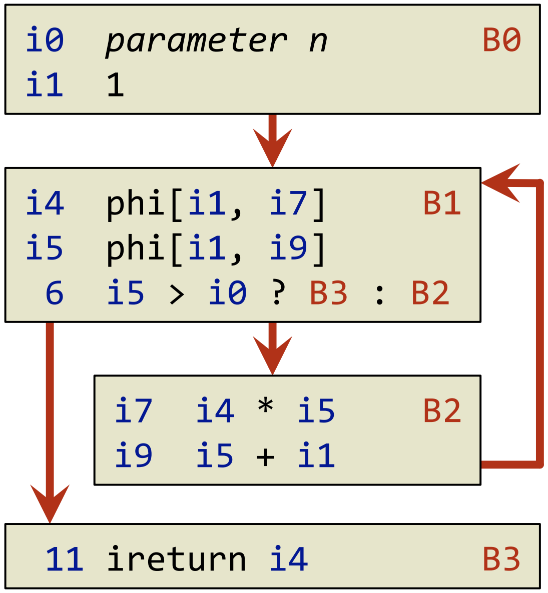
~ 210 KByte  Code Compiled

~ 122,000  HIR Instructions

| | Without Loop Optimization | | | With Loop Optimization | | |
|---|---|---|---|---|---|---|
| | Created | Simplified | | Created | Simplified | |
| Loop Phi Functions | 2843 | 2046 | 72% | 1224 | 429 | 35% |
| Other Phi Functions | 1930 | 376 | 20% | 1685 | 108 | 6% |
| Total | 4773 | 2422 | 50% | 2909 | 537 | 18% |

# SSA Deconstruction

## HIR Instructions



```
i0  parameter n      B0
i1  1
```

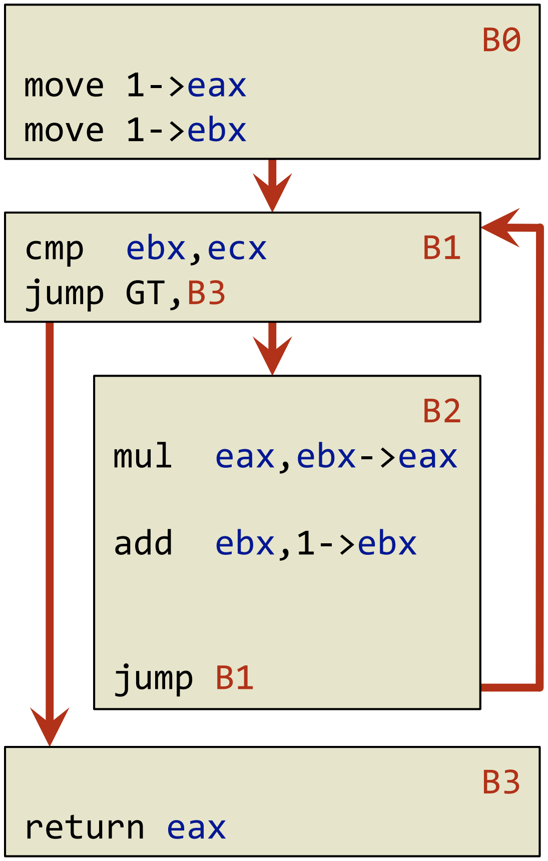
```
i4  phi[i1, i7]      B1
i5  phi[i1, i9]
 6  i5 > i0 ? B3 : B2
```

```
i7  i4 * i5          B2
i9  i5 + i1
```

```
 11 ireturn i4       B3
```

- LIR Register for Phi Function
- Moves in Predecessors
- No Coalescing
- Register Hints for Linear Scan Register Allocator
- Delete Unnecessary Moves

## LIR Operation



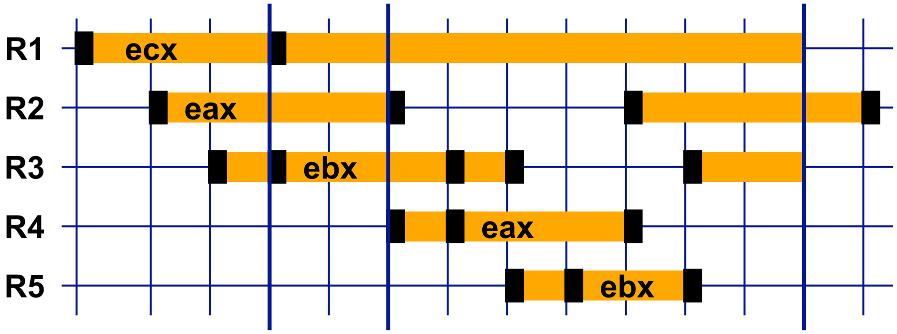
```
                     B0
move 1->eax
move 1->ebx
```

```
cmp  ebx,ecx         B1
jump GT,B3
```

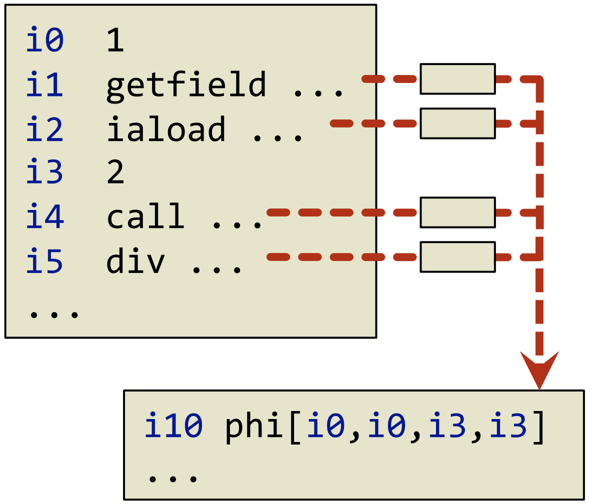
```
                     B2
mul  eax,ebx->eax

add  ebx,1->ebx


jump B1
```

```
                     B3
return eax
```

## Intervals for Linear Scan Register Allocation



# Exception Handling



```
i0  1
i1  getfield ...
i2  iaload ...
i3  2
i4  call ...
i5  div ...
...
```

```
i10 phi[i0,i0,i3,i3]
...
```

**Normal Control flow**

- All Critical Edges Split
- Save Position to Insert Moves

**Exception Edges**

- Start “In the Middle” of a Block
- Many Phi Function Operands

**Split Critical Exception Edges?**

- Would Lead to Many Blocks
- Most of them Finally Empty
- Overhead to Remove Them

**Phi Functions in LIR**

- Known by Register Allocator
- Create Adapter Blocks on Demand

# Array Bounds Check Elimination

Java Source Code

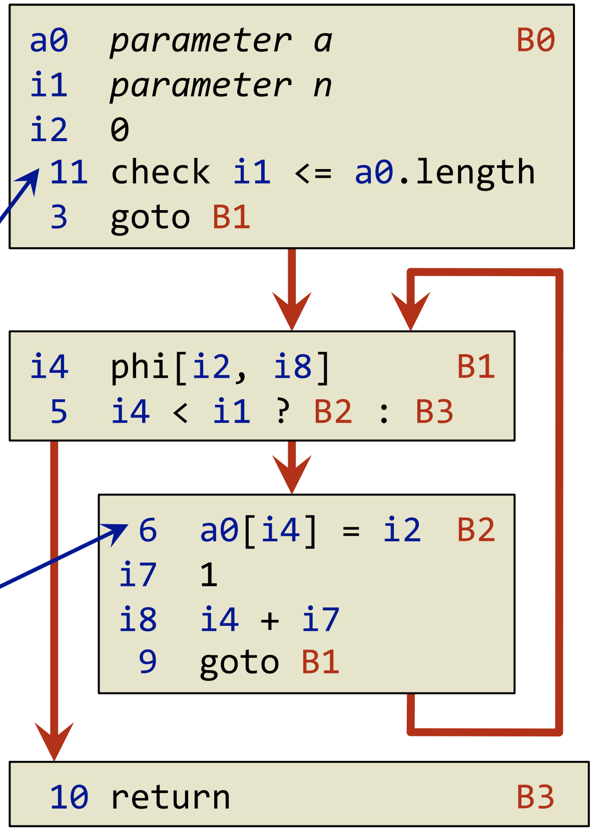
```
static void clear(int[] a, int n) {
  for (int i = 0; i < n; i++) {
    a[i] = 0;
  }
}
```

| Bounds for i4 at instruction 6 |
|---|
| Increasing (because of i8) |
| Lower Bound: i2 |
| Upper Bound: i1 |

| All Bounds Loop Invariant |
|---|
| Insert Check Before Loop |
| Remove Check Inside Loop |

| Java Exception Semantics |
|---|
| Integer Overflows |

HIR Instructions



```
a0  parameter a            B0
i1  parameter n
i2  0
 11 check i1 <= a0.length
 3  goto B1
```

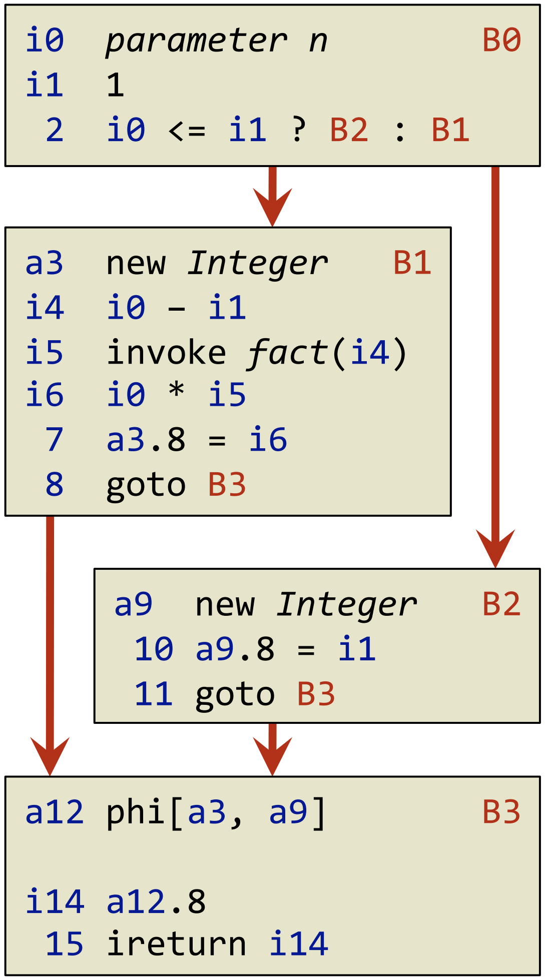
```
i4  phi[i2, i8]        B1
 5  i4 < i1 ? B2 : B3
```

```
 6  a0[i4] = i2  B2
i7  1
i8  i4 + i7
 9  goto B1
```

```
 10 return                 B3
```

# Escape Analysis

## HIR Instructions



```
i0   parameter n           B0
i1   1
 2   i0 <= i1 ? B2 : B1
```

```
a3   new Integer     B1
i4   i0 – i1
i5   invoke fact(i4)
i6   i0 * i5
 7   a3.8 = i6
 8   goto B3
```

```
a9   new Integer     B2
 10  a9.8 = i1
 11  goto B3
```

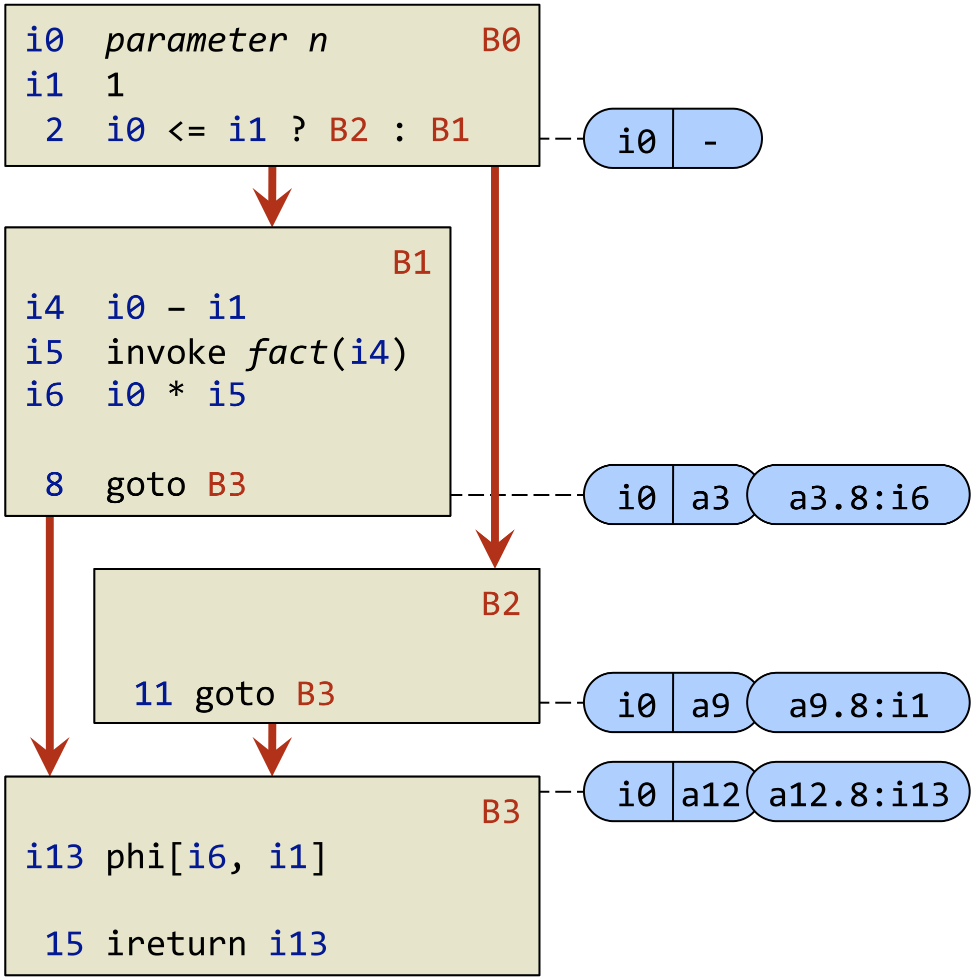
```
a12 phi[a3, a9]            B3

i14 a12.8
 15 ireturn i14
```

## Java Source Code

```
static int fact(int n) {
  Integer p;
  if (n > 1) {
    p = new Integer(n * fact(n - 1));
  } else {
    p = new Integer(1);
  }
  return p.intValue();
}
```

# Escape Analysis

HIR Instructions



```
i0  parameter n         B0
i1  1
 2  i0 <= i1 ? B2 : B1
```

```
                    B1
i4  i0 - i1
i5  invoke fact(i4)
i6  i0 * i5

 8  goto B3
```

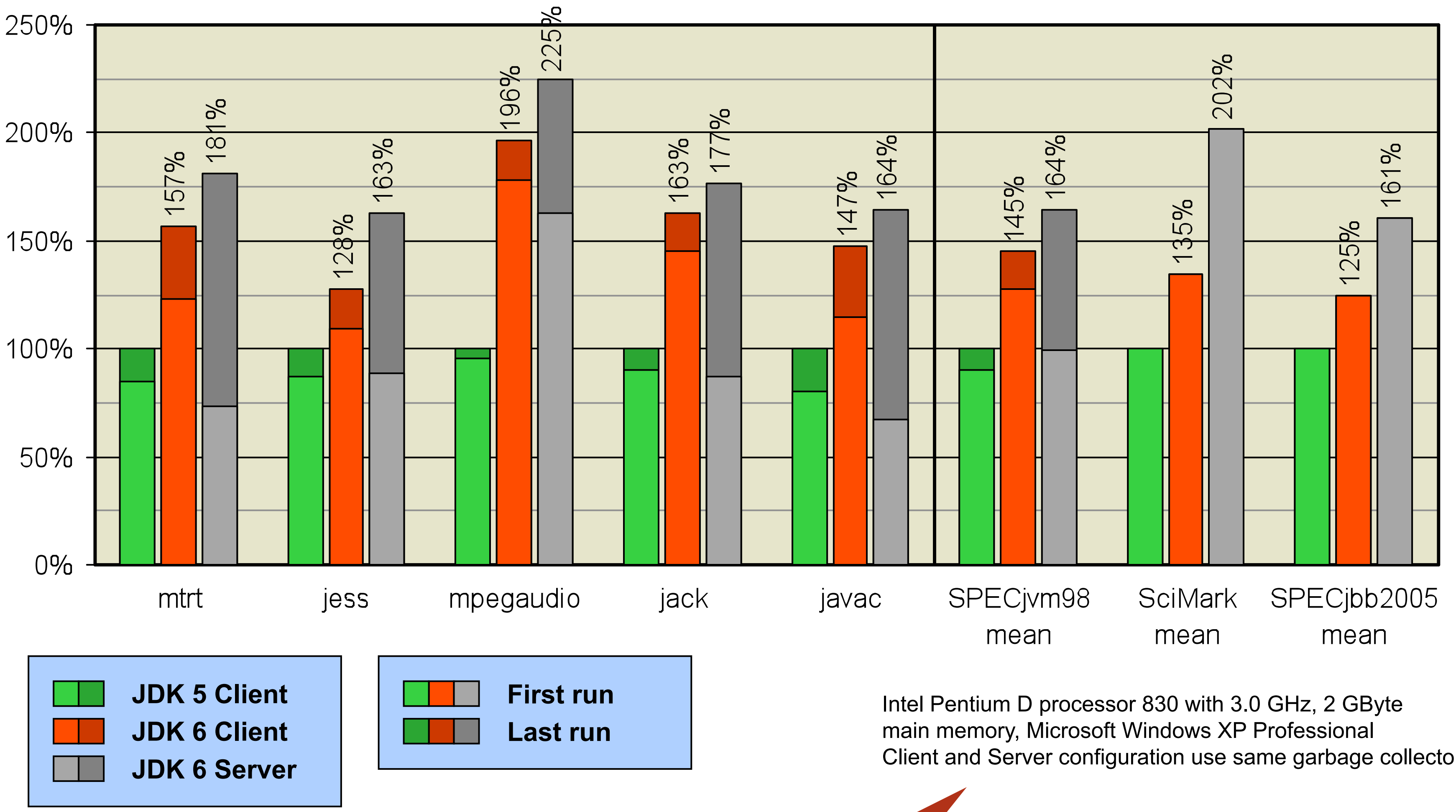
```
                B2

 11 goto B3
```

```
                    B3
i13 phi[i6, i1]

 15 ireturn i13
```

- Track Field Values
  - State Similar to Variables
  - Create Phi Functions
- Track Escape State
  - Method Local
  - Thread Local
  - Escaping
- Optimize Non-Escaping Objects
  - Eliminate Object Allocation
  - Eliminate Field Stores
  - Replace Field Loads
- Alias Effects
- Equi-Escape Sets

# Performance Comparison

Speedup of JDK 6 Client and Server relative to JDK 5 Client



Intel Pentium D processor 830 with 3.0 GHz, 2 GByte
main memory, Microsoft Windows XP Professional
Client and Server configuration use same garbage collector

# Selected Publications

- Client Compiler, SSA Form, Linear Scan Register Allocation
  - Thomas Kotzmann, Christian Wimmer, Hanspeter Mössenböck, Thomas Rodriguez, Kenneth Russell, David Cox: ***Design of the Java HotSpot™ Client Compiler for Java 6***. In *ACM Transactions on Architecture and Code Optimization*, volume 5, issue 1, article 7. ACM Press, 2008. doi: 10.1145/1369396.1370017
  - Christian Wimmer, Hanspeter Mössenböck: ***Optimized Interval Splitting in a Linear Scan Register Allocator***. In *Proceedings of the ACM/USENIX International Conference on Virtual Execution Environments*, pages 132-141. ACM Press, 2005. doi:10.1145/1064979.1064998

- Array Bounds Check Elimination
  - Thomas Würthinger, Christian Wimmer, Hanspeter Mössenböck: ***Array Bounds Check Elimination in the Context of Deoptimization***. In *Science of Computer Programming*, volume 74, issues 5-6, pages 279-295. Elsevier, 2009. doi:10.1016/j.scico.2009.01.002

- Escape Analysis
  - Thomas Kotzmann, Hanspeter Mössenböck: ***Escape Analysis in the Context of Dynamic Compilation and Deoptimization***. In *Proceedings of the ACM/USENIX International Conference on Virtual Execution Environments*, pages 111-120. ACM Press, 2005. doi:10.1145/1064979.1064996
  - Thomas Kotzmann, Hanspeter Mössenböck: ***Run-Time Support for Optimizations Based on Escape Analysis***. In *Proceedings of the International Symposium on Code Generation and Optimization*, pages 49-60. IEEE Computer Society, 2007. doi:10.1109/CGO.2007.34

- Complete List of Publications:
  - http://wikis.sun.com/display/HotSpotInternals/Publications+JKU