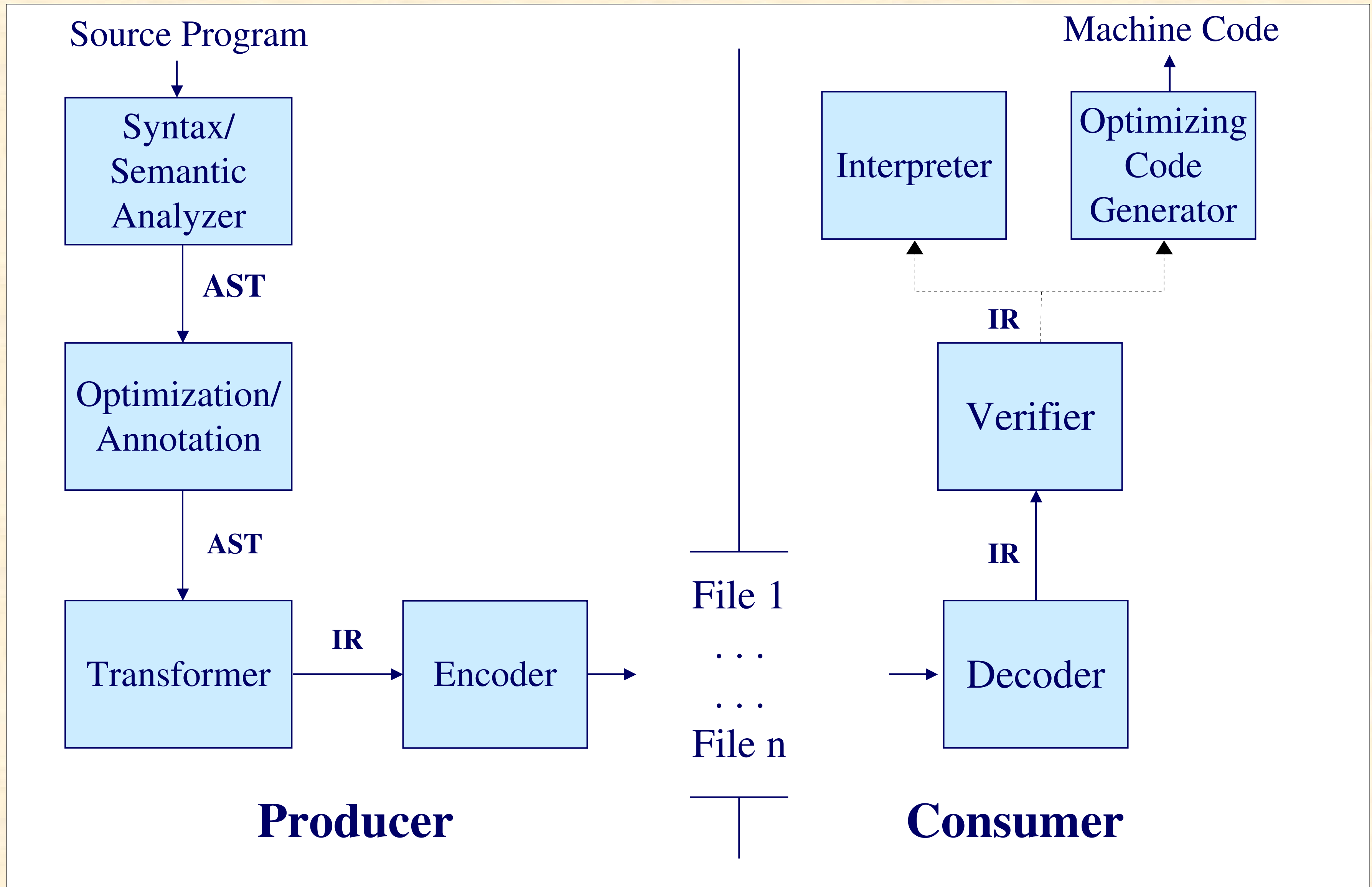
SSA-Based Mobile Code: Construction and Empirical Evaluation

Wolfram Amme
Friedrich-Schiller-University Jena, Germany

Michael Franz
Universit of California, Irvine, USA

Jeffery von Ronne
Universtity of Texas, San Antonio, USA

General System for the Transport of Mobile Code



Wolfram Amme, SSA-Based Mobile Code: Construction and Empirical Evaluation, Autrans, France, 2009.

Mobile Code Security

- **most approaches are based on some type-safe programming language**
- **program safety is usually defined as type safety**
- **objective: type safety,** i.e. no
  - invalid pointer accesses
  - illegal field accesses
  - operator application with illegal parameters
  - calling routines imported from elsewhere with illegal parameters

De facto Standard: Java's Bytecode

- **Java's bytecode format is the de facto standard for transporting mobile code**
- **however, it is far away from being an ideal mobile code representation**
  - stack model of the JVM leads to a time-consuming verification phase on the consumer side
  - limitation of accessing the top elements of the stack prevents the reuse of operands and code reordering
  - optimizing JIT compilers often transform Java bytecode internally into code for a register machine
  - many bytecode operations include sub-operations (null-checks, bounds checks)

SafeTSA: Facts

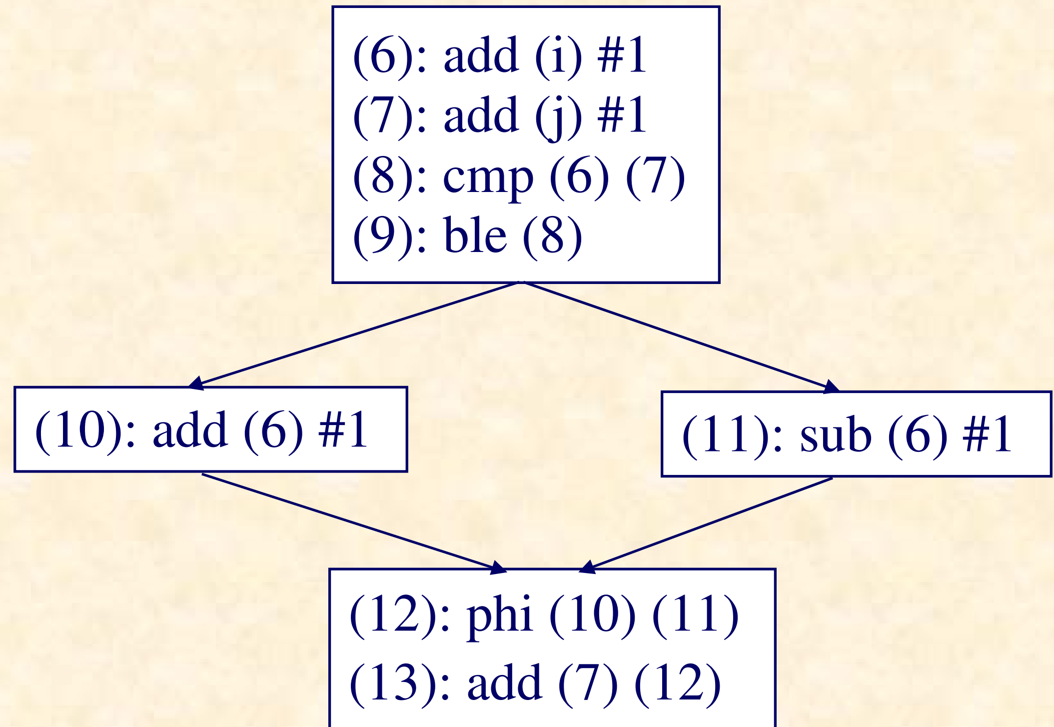
- **transportation format consisting of**
  - a symbol table
  - an abstract syntax tree
  - SSA-style instructions within basic blocks
- **SSA-based instruction format within basic blocks**
  - is reference-safe and type-safe with less verification effort than Java bytecode
  - allows to move CSE from code consumer to code producer
  - can transport results of null and bounds check elimination in a tamper-proof manner
  - can directly used for JIT compilation

SafeTSA - Construction

Program in SSA-Form

Example:

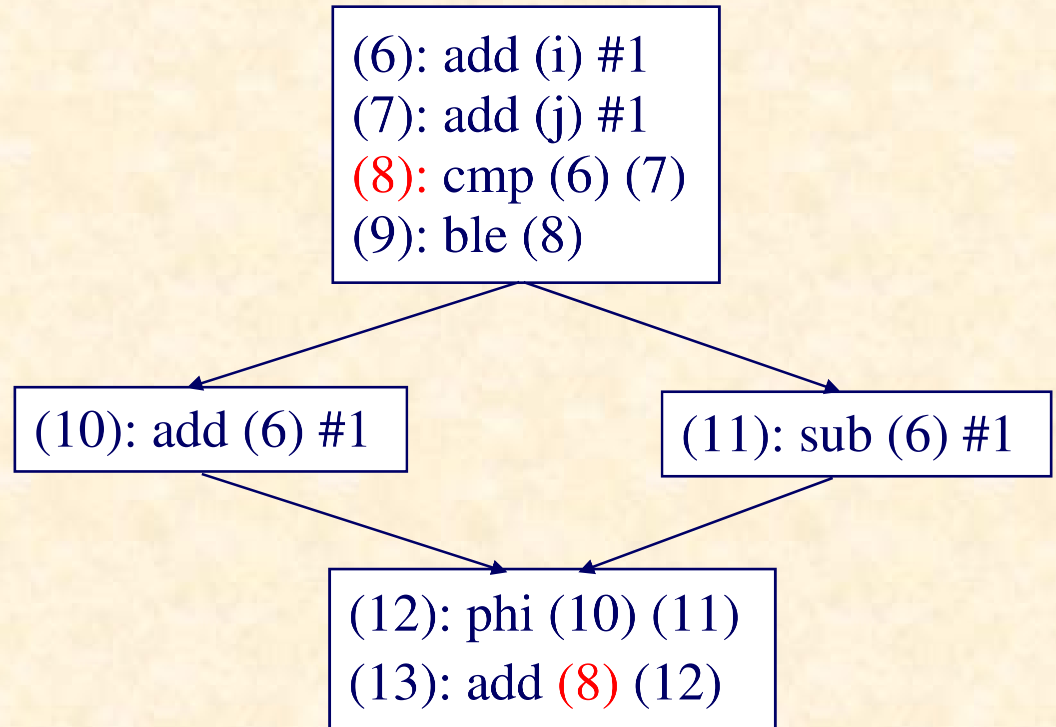
```
    . . .
i = i + 1;
j = j + 1;
if  (i<= j)
        i = i + 1;
else
        i = i - 1;
j = j + i;
```



Program in SSA-Form

Example:

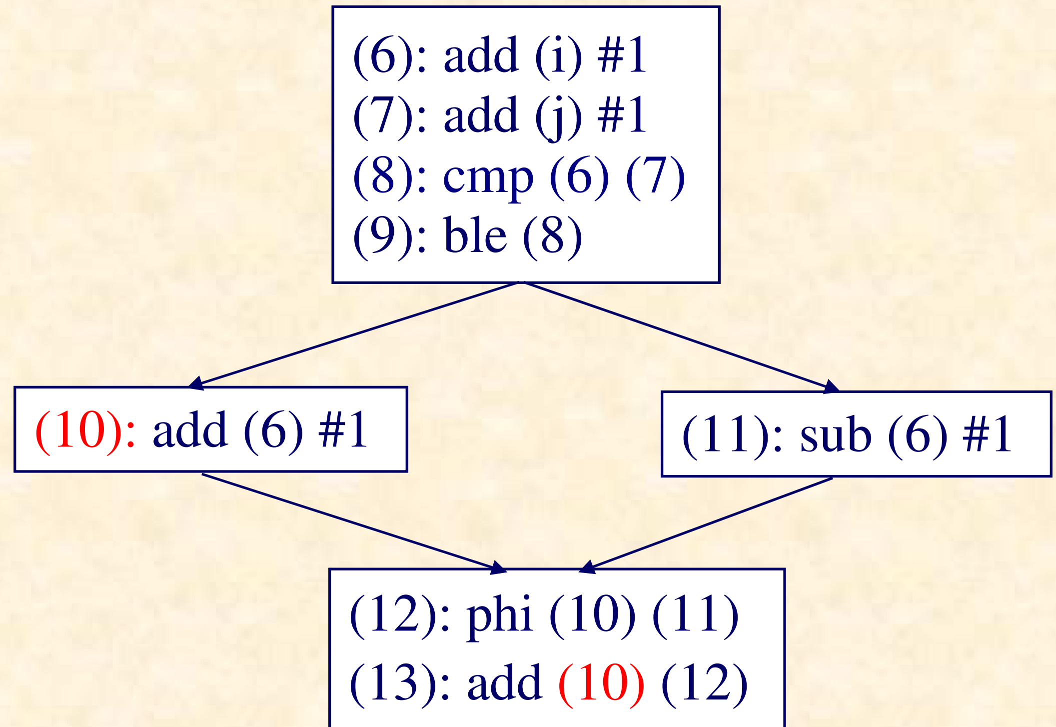
```
    . . .
i = i + 1;
j = j + 1;
if  (i<= j)
       i = i + 1;
else
       i = i - 1;
j = j + i;
```



Program in SSA-Form

Example:

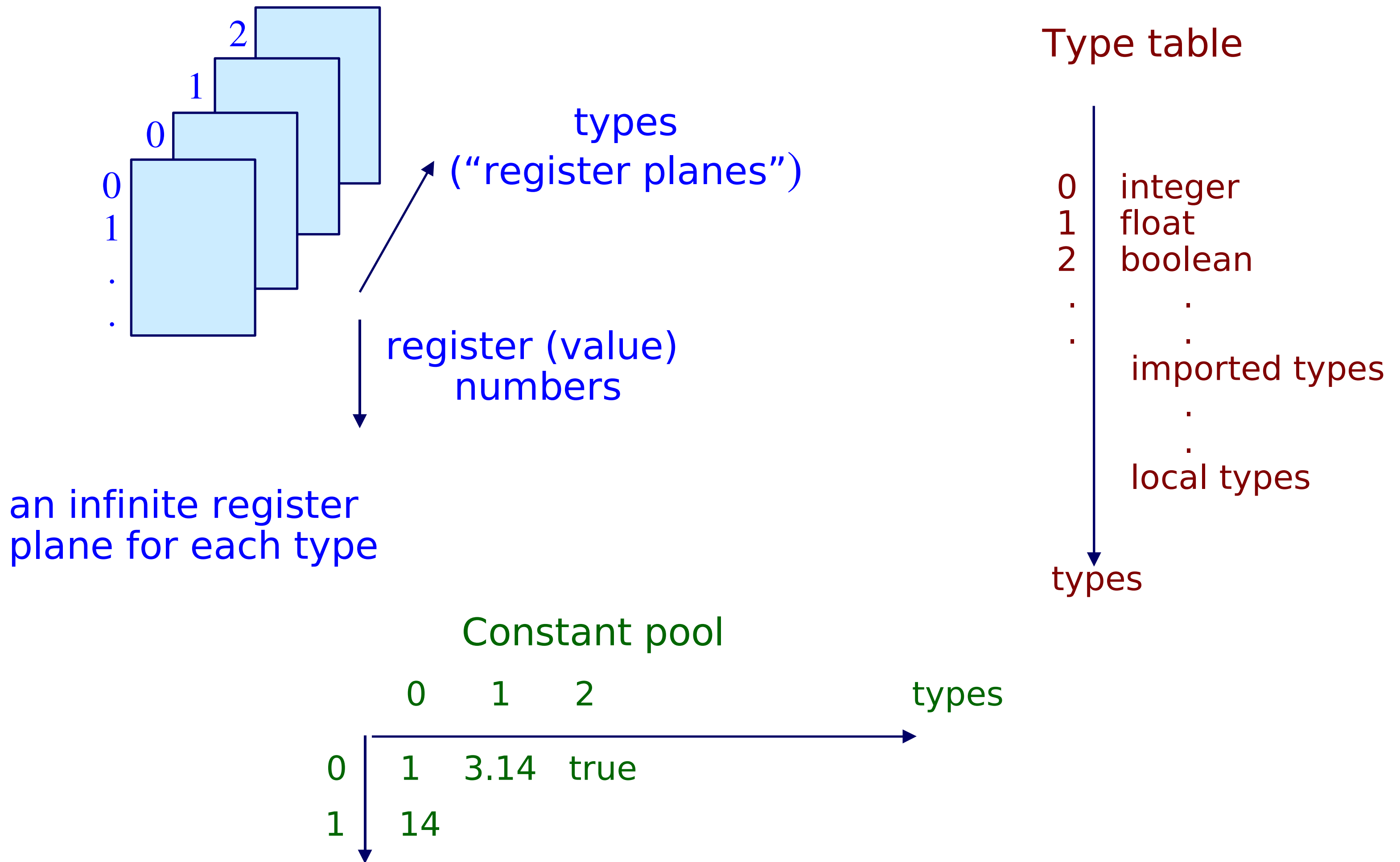
```
    . . .
i = i + 1;
j = j + 1;
if  (i<= j)
        i = i + 1;
else
        i = i - 1;
j = j + i;
```



Reference error

Wolfram Amme, SSA-Based Mobile Code: Construction and Empirical Evaluation, Autrans, France, 2009.

Extended Machine Model



Wolfram Amme, SSA-Based Mobile Code: Construction and Empirical Evaluation, Autrans, France, 2009.

Type Separation in SafeTSA

- **in SafeTSA all operations are strongly typed**
- **for all operations the following holds:**

  - a specific operation implicitly selects the register plane(s) from which the arguments are taken

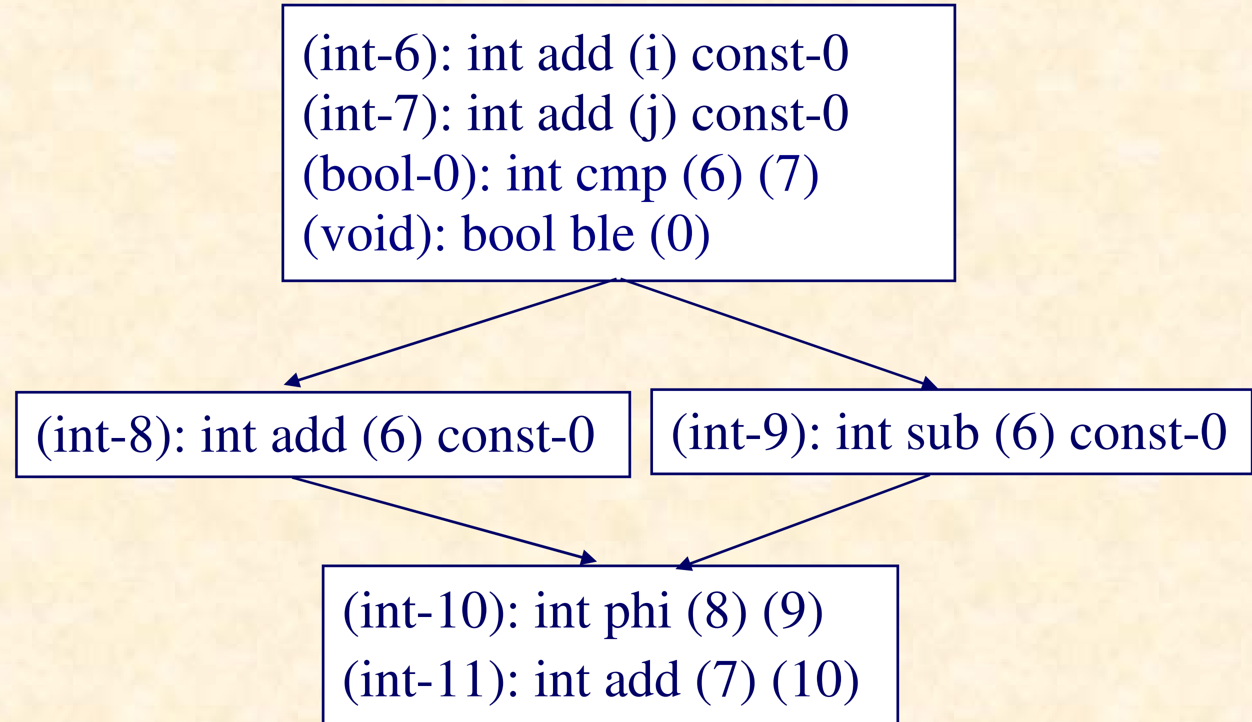
  - an operation merely specifies the register number(s) on that plane, but not the plane(s) involved

  - the result is deposited in the next available register on the plane that corresponds to the result of the operation

Type-Separated SSA

Example:

```
   . . .
i = i + 1;
j = j + 1;
if  (i<= j)
      i = i + 1;
else
      i = i - 1;
j = j + i;
```



Wolfram Amme, SSA-Based Mobile Code: Construction and Empirical Evaluation, Autrans, France, 2009.

Reference Safety: Construction

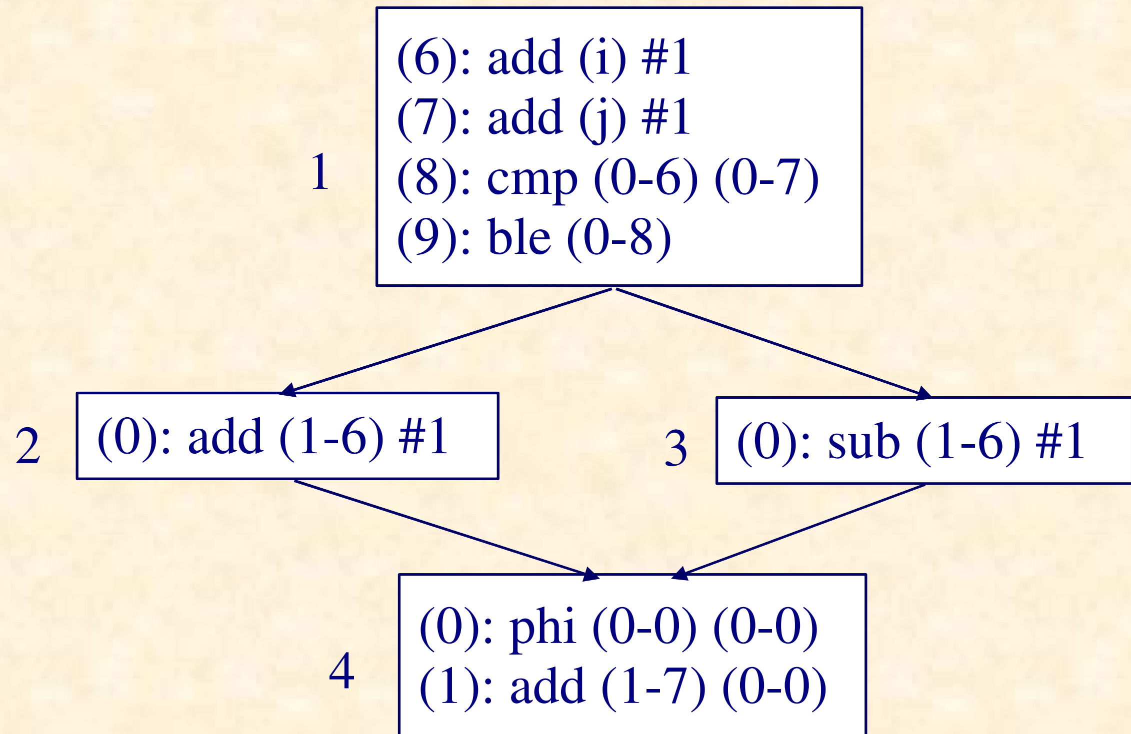
- **dominator tree of a program is used for safe access to values**
  - in a dominator tree all predecessors of a node, that represents a basic block A, stand for basic blocks which always will be executed before A
- **in reference safe SSA Form an operand access is a pair (*steps*,*value*), where**
  - *steps*: number of nodes, that starting with the actual basic block, have to be traversed the dominator tree backwards (until the basic block is found which defines the value)
  - *value*: a relative instruction number in that basic block

Wolfram Amme, SSA-Based Mobile Code: Construction and Empirical Evaluation, Autrans, France, 2009.

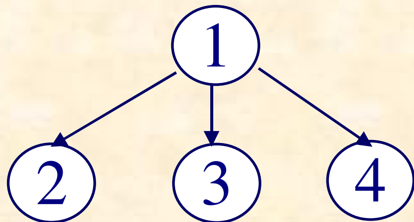
Reference-Safe SSA

Example:

```
  . . .
i = i + 1;
j = j + 1;
if  (i<= j)
      i = i + 1;
else
      i = i - 1;
j = j + i;
```



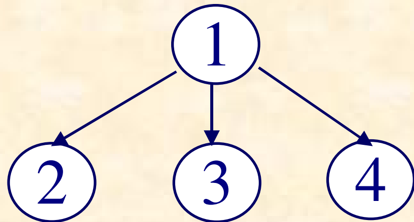
Dominator tree:



Wolfram Amme, SSA-Based Mobile Code: Construction and Empirical Evaluation, Autrans, France, 2009.

SafeTSA: Type-separated and Reference-safe SSA

Dominator tree:



1
(int-6): int add (i) const-0
(int-7): int add (j) const-0
(bool-0): int cmp (0-6) (0-7)
(void): bool ble (0-0)

2
(int-0): int add (1-6) const-0

3
(int-0): int sub (1-6) const-0

4
(int-0): int phi (0-0) (0-0)
(int-1): int add (1-7) (0-0)

Each basic block is assigned its own set of register planes

Instruction Set

Instruction Set: Operators

prim <type> <primfun> <param>*
xprim <type> <primfun> <param>*

Example:
(int-3): prim int add (0-2) (0-1)

- difference between prim and xprim is whether or not the operation may cause an exception

Instruction Set: Cast Operators

xupcast <type> <type> <object>
downcast <type> <integer> <object>

Example:
class B extends A { };

(ref-B): xupcast ref-A ref-B (...)
(ref-A): downcast ref-B 1 (...)

Instruction Set: Other Kind of Instructions

Memory Access

getfield <type> <object> <symbol>
setfield <type> <object> <symbol> <value>
getelt <type> <object> <position>
setelt <type> <object> <position> <value>

Method Call

xdispatch <type> <object> <fun> <param>*
xcall <type> <object> <fun> <param>*

Phi Instruction

phi <type> <value>*

Null and Bounds Check Elimination

Construction of Memory Safety

- **safe reference and safe index types**
  - for each reference type ref-T we introduce a safe reference type safe-T guaranteed not to be null
  - for each array object A we have a safe index type safe-index-A guaranteeing that the array's index value is within range (created when array is allocated)
- **null and range checking then become operations that take values from an unsafe value-plane and copy them (to the first available register) of the corresponding safe reference type's plane**
- **memory and array accesses take their operands always from the corresponding safe value-plane**

Example: Memory Access and Nullcheck Elimination

```
class A{
    int f;
}


A obj;
...                                  (safe-A-1): xupcast ref-A safe-A (...)
obj.f;                               (int-1): getfield A (0, safe-A-1) f
...                                  ...
obj.f;                               (safe-A-2): xupcast ref-A safe-A (...)
                                     (int-2): getfield A (0, safe-A-2) f
```

Example: Memory Access and Nullcheck Elimination

class A{	
int f;	
}	
A obj;	
...	(safe-A-1): xupcast ref-A safe-A (...)
obj.f;	(int-1): getfield A (0, safe-A-1) f
...	...
obj.f;	
	(int-2): getfield A (0, safe-A-1) f

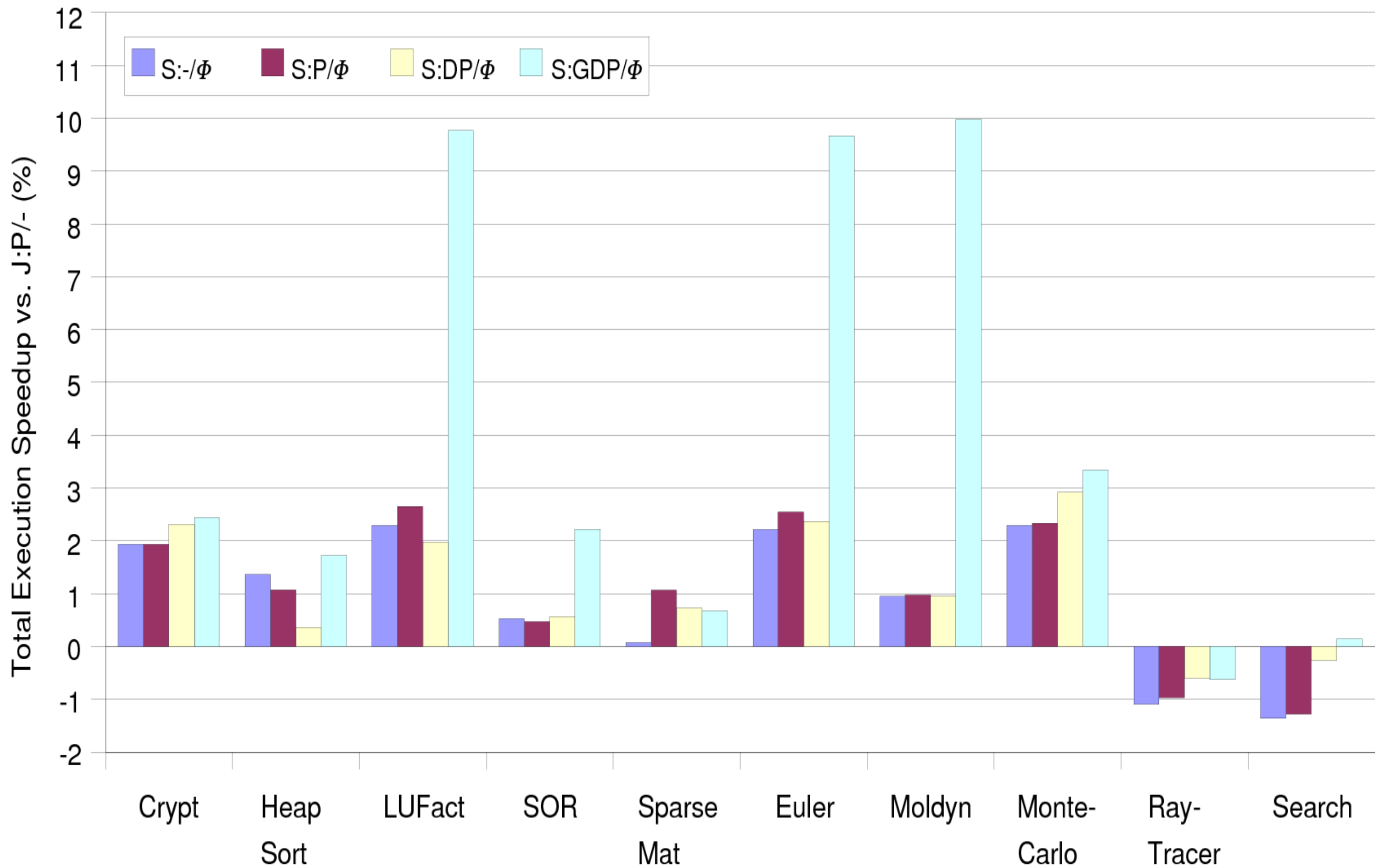
Implementation and Evaluation

SafeTSA: Implementation

- **a compiler that transforms Java programs into SafeTSA class files**
  - extension of *Pizza´s Java compiler*
  - Optimizations: constant propagation, deadcode elimination, and CSE

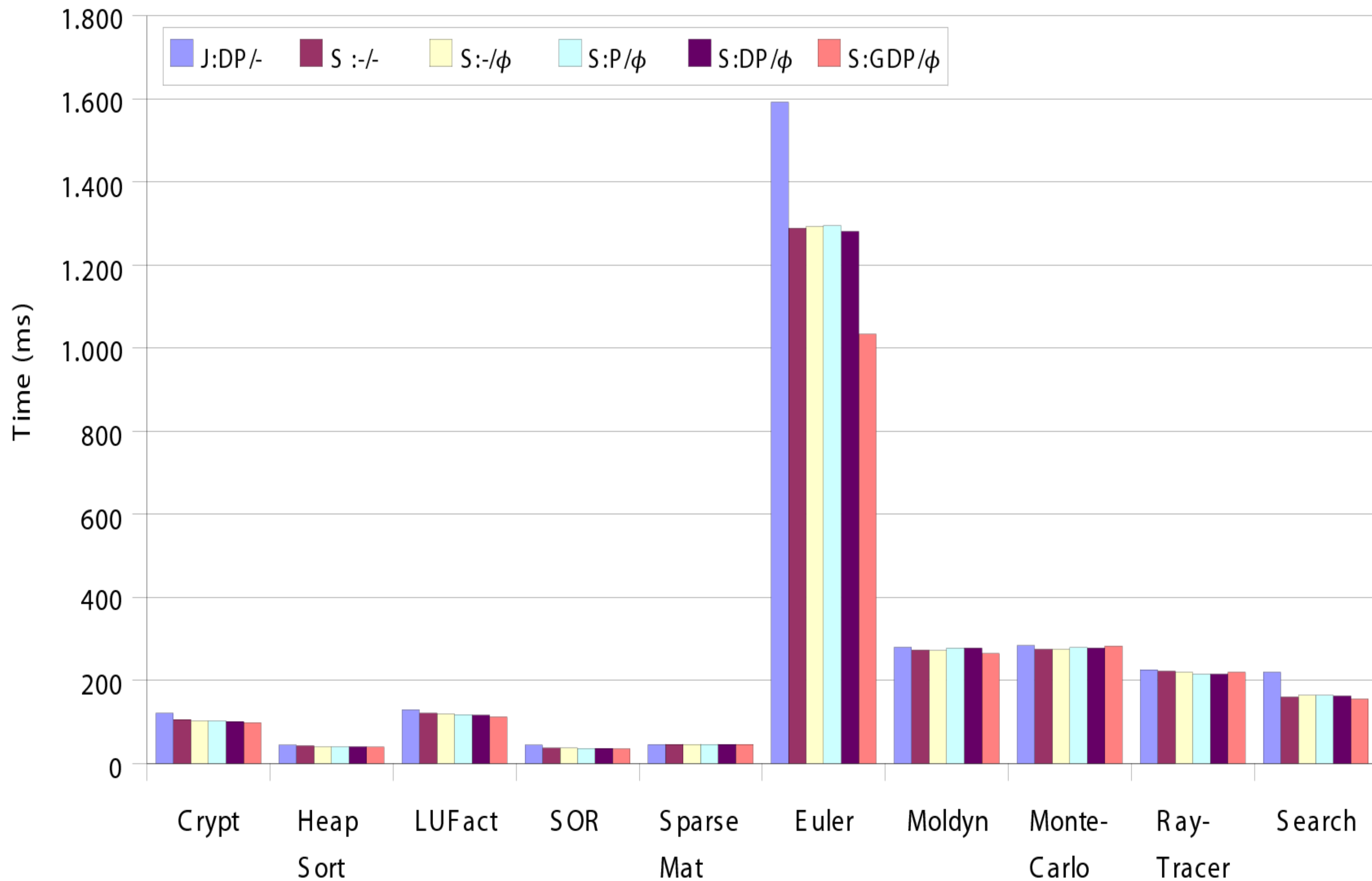
- **a JVM which is capable of executing heterogeneous of bytecode and SafeTSA class files**
  - extension of IBM´s Jikes RVM
  - Optimizations: method inlining, load and store elimination, global code motion, etc.

Results: Runtime Behavior



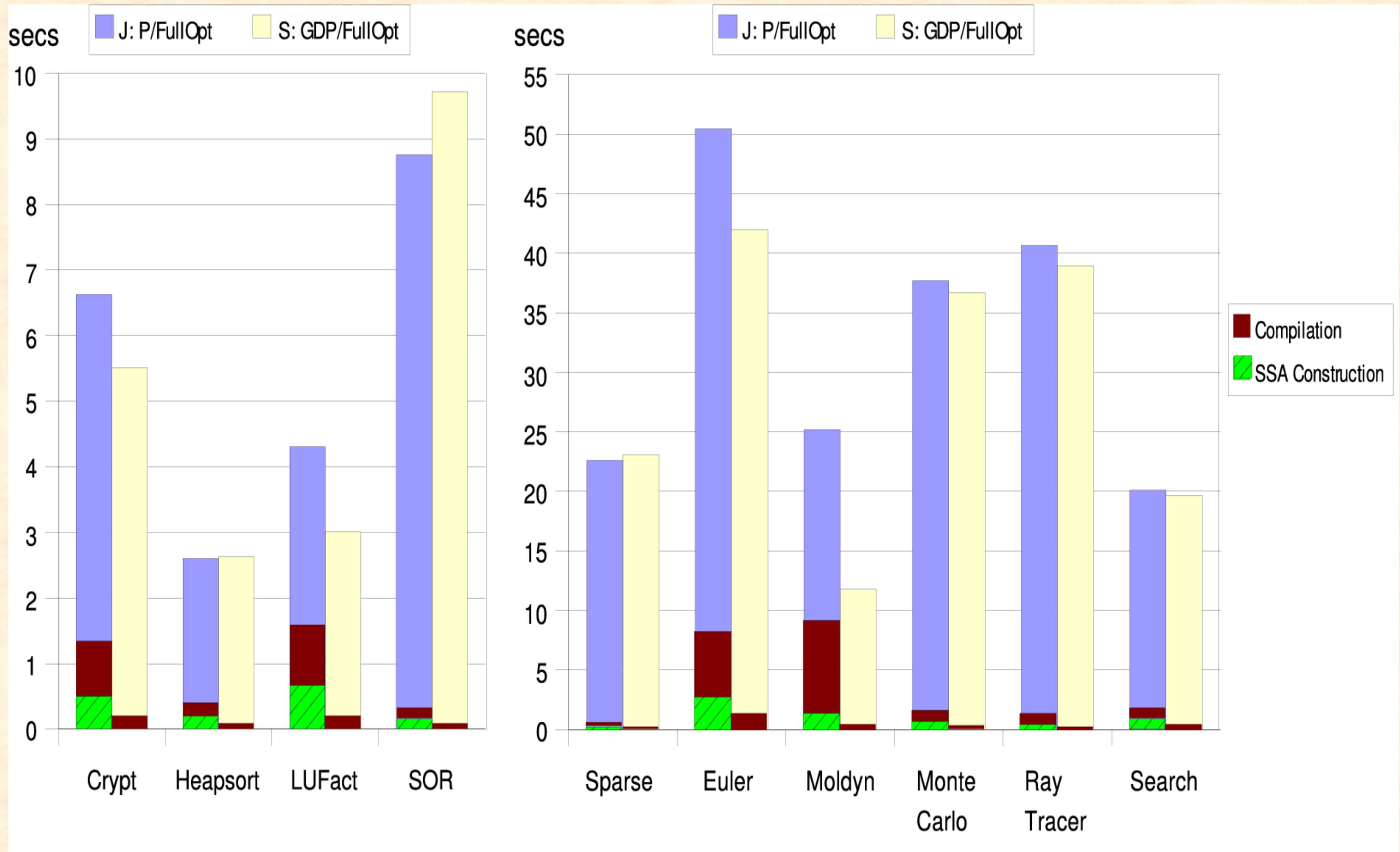
Wolfram Amme, SSA-Based Mobile Code: Construction and Empirical Evaluation, Autrans, France, 2009.

Results: Compilation Times



Wolfram Amme, SSA-Based Mobile Code: Construction and Empirical Evaluation, Autrans, France, 2009.

Results: Optimizing JIT Compilation



Wolfram Amme, SSA-Based Mobile Code: Construction and Empirical Evaluation, Autrans, France, 2009.