Making Resource Analysis Practical for Real-Time Java

Rody Kersten, Olha Shkaravska, Bernard van Gastel, Manuel Montenegro and Marko van Eekelen

Institute for Computing and Information Sciences
Radboud University Nijmegen

October 25, 2012

CHARTER

- **C**ritical and **H**igh **A**ssurance **R**equirements **T**ransformed through **E**ngineering **R**igour
- ARTEMIS Embedded Computing Systems Initiative project
- April 2009 – June 2012
- 3 times in a row the highest ratings
- Project partners: aicas GmbH, Atego Ltd, Chalmers University of Technology, Impronova AB, Lero at Dundalk Institute of Technology, Luminis, NLR, QRTECH AB, Radboud Universiteit Nijmegen, The Open Group, Universiteit Twente

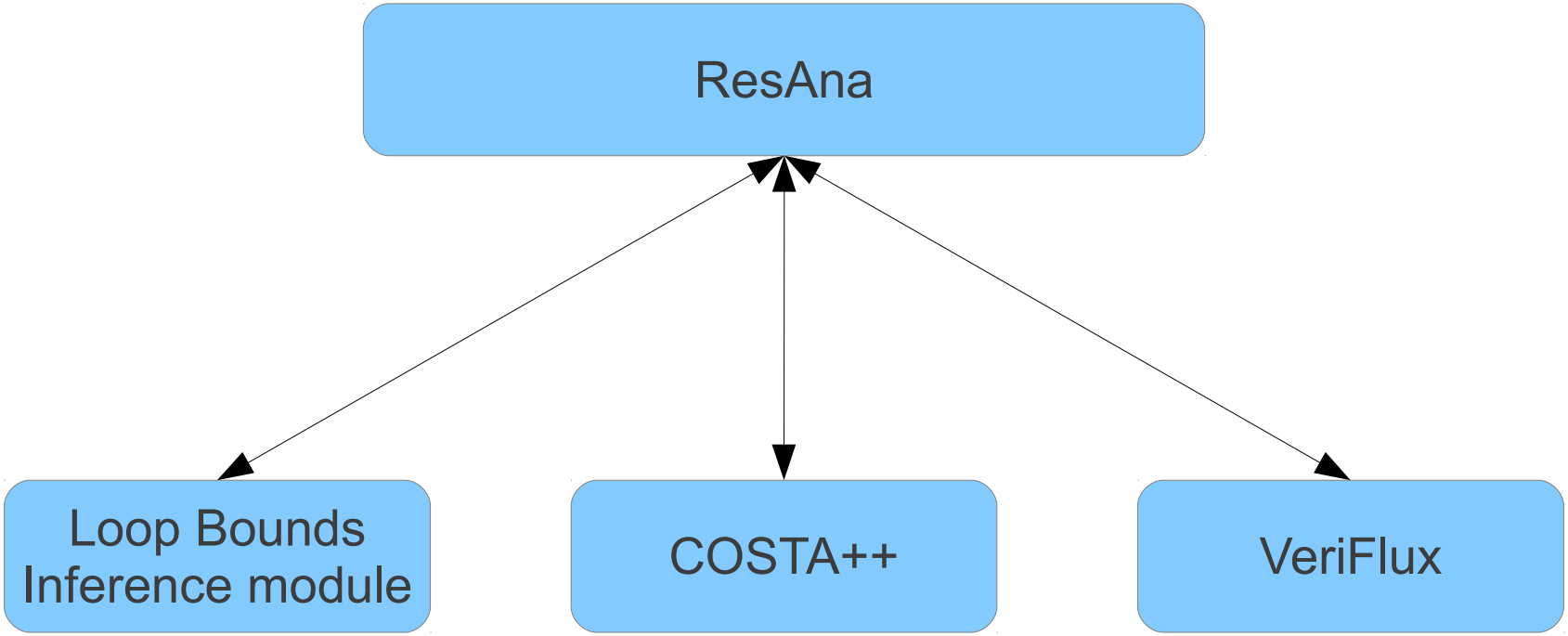
CHARTER

- Aim: improve certification process of critical embedded systems
- Automotive, health care, avionics, surveillance
- By means of:
  - Model Driven Development
  - Rule Based Compilation
  - Formal Verification
  - Focus on Real-Time Java (JamaicaVM)
- Radboud University's focus was on resource analysis and formal specification of the Real-Time Java API

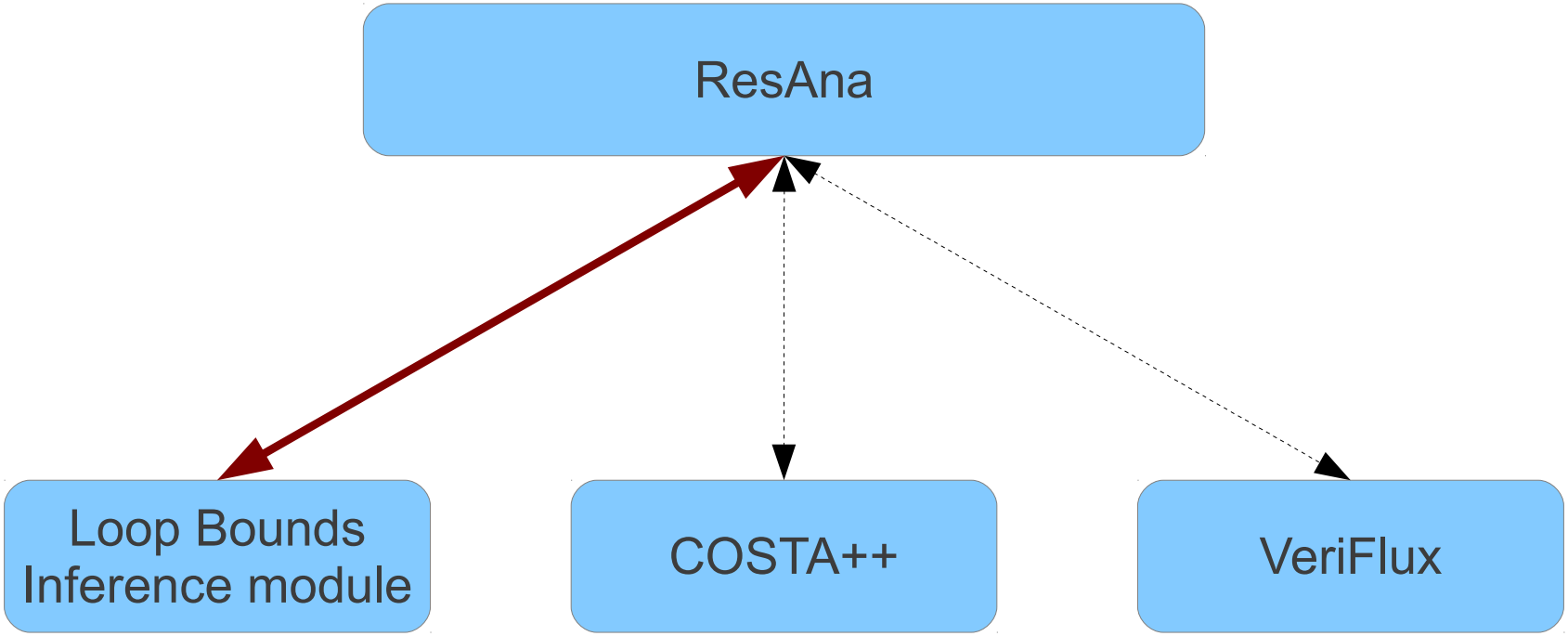
Resource Analysis in CHARTER

- Loop-Bound Analysis
  - Prove termination
  - Prerequisite for other resource analysis
- Memory Usage Analysis (Heap and Stack)
  - In safety and security critical applications: to prevent abrupt termination due to the lack of memory, because output and intermediate structures are too large (DOS attack)
  - To optimise memory management, e.g. by allocation in advance chunks of a heap
  - To be able to physically configure embedded systems for small devices in an optimal way

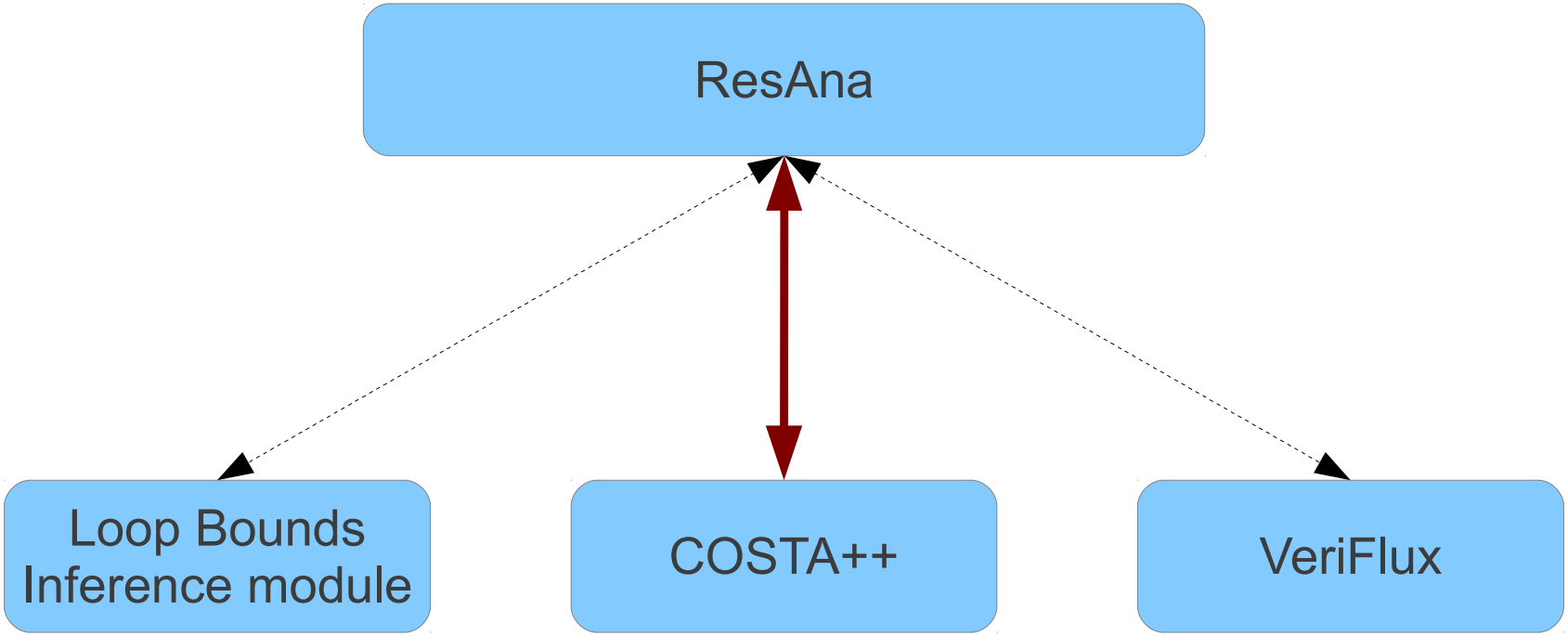
ResAna



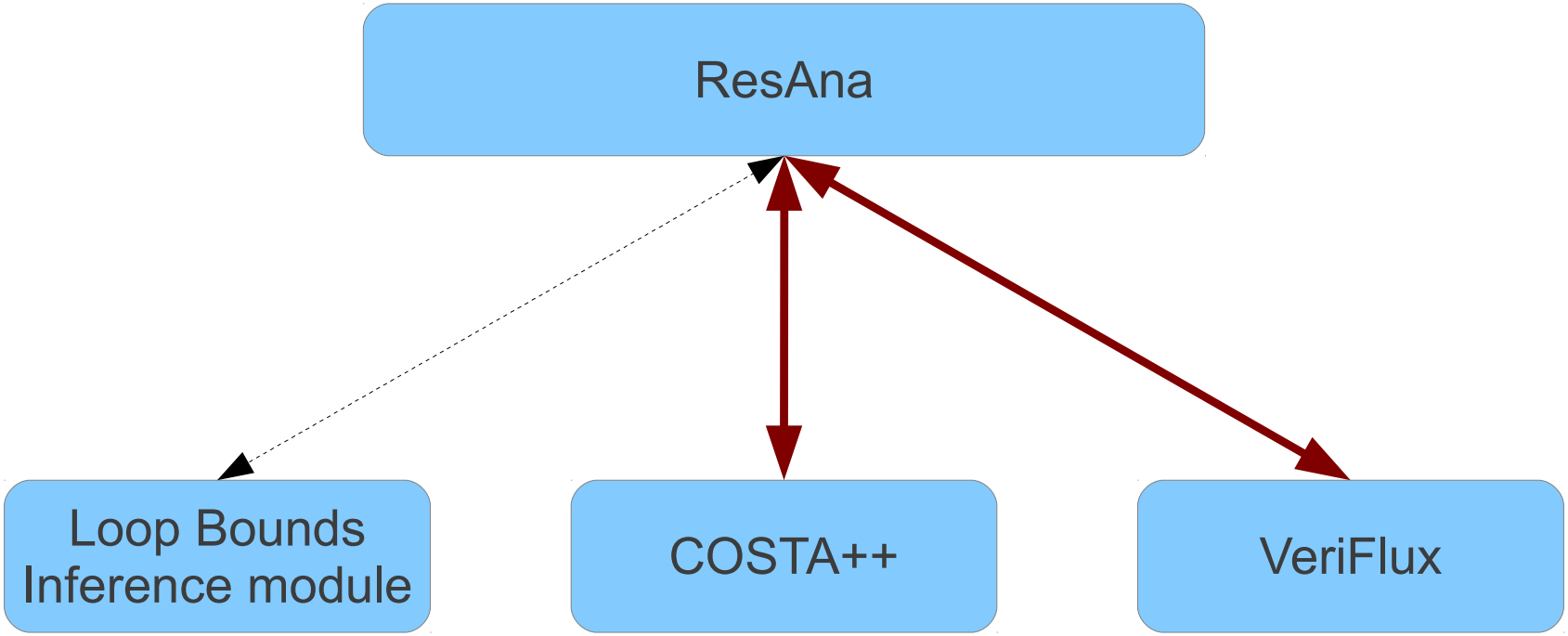
ResAna: Loop Bound Analysis



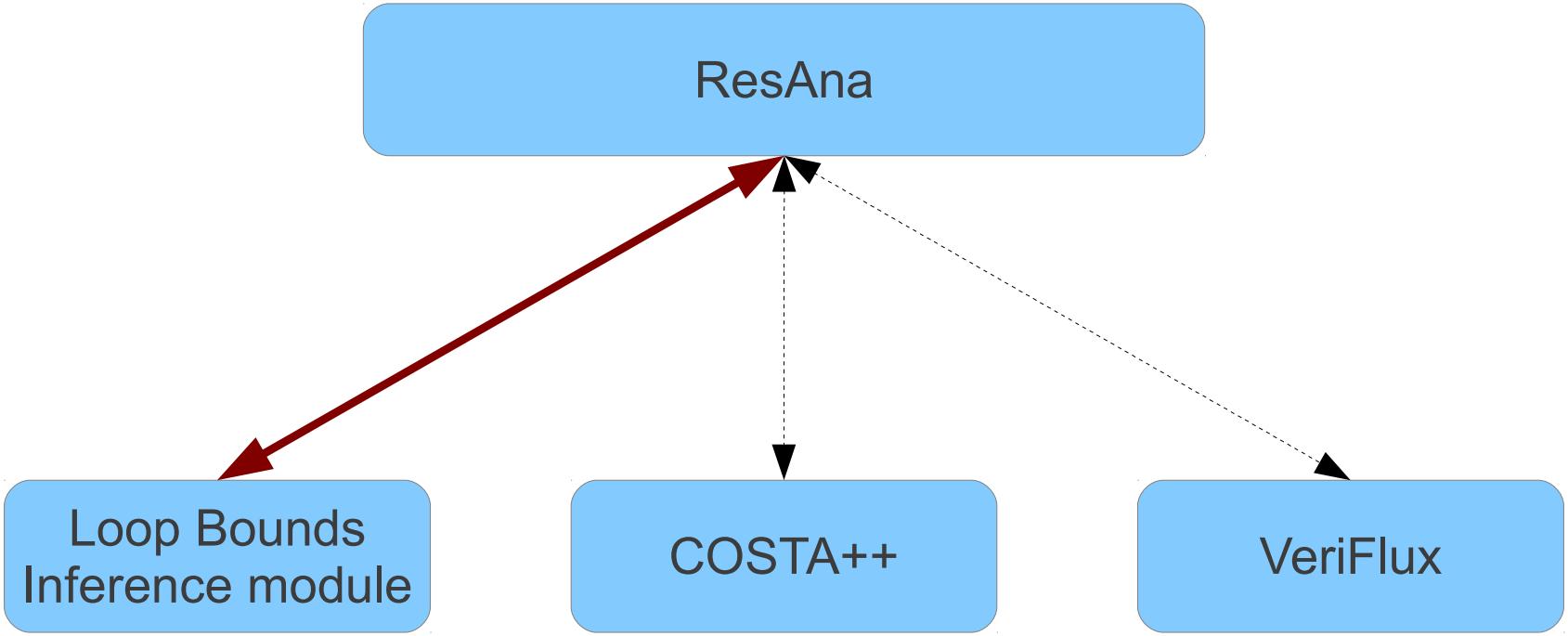
ResAna: Heap Analysis



ResAna: Stack Analysis



Loop Bound Analysis



Ranking Function

- Decreases in every basic block
- Here: in every loop iteration
- Bounded by zero

```
while (i < 15) {
  i++;
}
```

- Ranking function for the loop above is $15 - i$

Prerequisite for Resource Analysis

```
while (i < 15) {
  consumeResource();
  i++;
}
```

Applicable Loops

- The basic polynomial interpolation method considers loops with conditions in the following form:

$$C := sC \mid C_1 \wedge C_2$$
$$sC := e_1 \; [<, >, \leq, \geq, =, \neq] \; e_2$$

- where e_i are arithmetical expressions
- i.e. conjunctions over arithmetical (in)equalities

Test-Based Approach

1. Instrument loop with a counter
2. Do test runs for a set of $N_d^k = \binom{d+k}{k}$ input values satisfying **NCA** and the exit condition
3. Interpolate a polynomial from the results

Test-Based Approach

1. Instrument loop with a counter
2. Do test runs for a set of $N_d^k = \binom{d+k}{k}$ input values satisfying **NCA** and the exit condition
3. Interpolate a polynomial from the results

Test-Based Approach

1. Instrument loop with a counter
2. Do test runs for a set of $N_d^k = \binom{d+k}{k}$ input values satisfying **NCA** and the exit condition
3. Interpolate a polynomial from the results

Quadratic Example

```
public void m(int a, int b, int c) {
        while (a > 0 && c <= b && c > 0) {
                if ( c == b ) { a−−; c = 0; }
                c++;
        }
}
```

```
public int m(int a, int b, int c) {
        int count=0;
        while (a > 0 && c <= b && c > 0) {
                if ( c == b ) { a−−; c = 0; }
                c++;
                count++;
        }
        return count;
}
```

Expected degree of polynomial (here: d=2)

Test runs

1st group: degree 2 NCA on plane
a=1, b=1, c=1 => count=1
a=1, b=1, c=2 => count=2
a=1, b=1, c=3 => count=3
a=1, b=2, c=2 => count=1
a=1, b=2, c=3 => count=2
a=1, b=3, c=3 => count=1

2nd group: degree 1 NCA on plane
a=2, b=1, c=1 => count=2
a=2, b=1, c=2 => count=4
a=2, b=2, c=2 => count=3

3rd group: degree 0 NCA on plane
a=3, b=1, c=1 => count=3

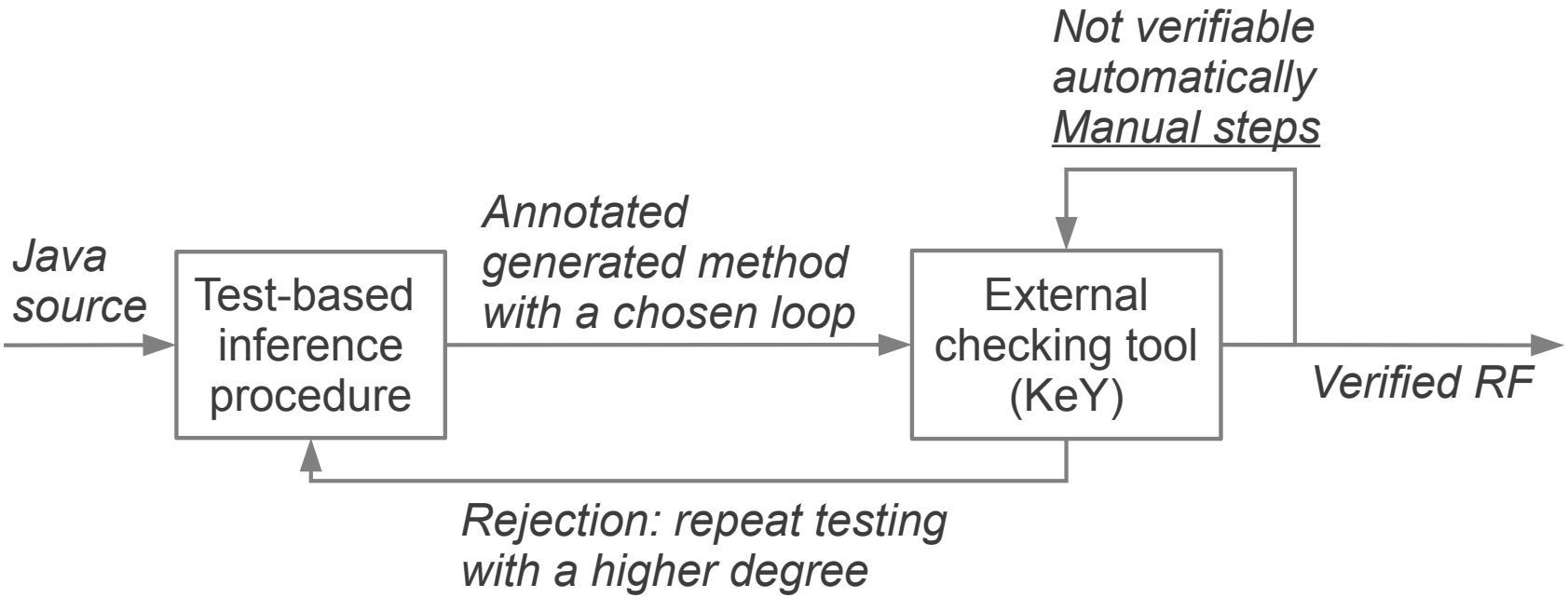
Find the interpolating polynomial and generate the method annotated with the corresponding ranking function:
RF(a, b, c) = a*b – c + 1

Soundness

- The procedure itself is unsound
- Use external prover to verify the inferred ranking functions
- KeY: `http://www.key-project.org/`
- Ranking function can be expressed in JML as a `decreases` clause

```
//@ decreases i < 15 ? 15 - i : 0;
while (i < 15) {
  i++;
}
```

Helicopter View

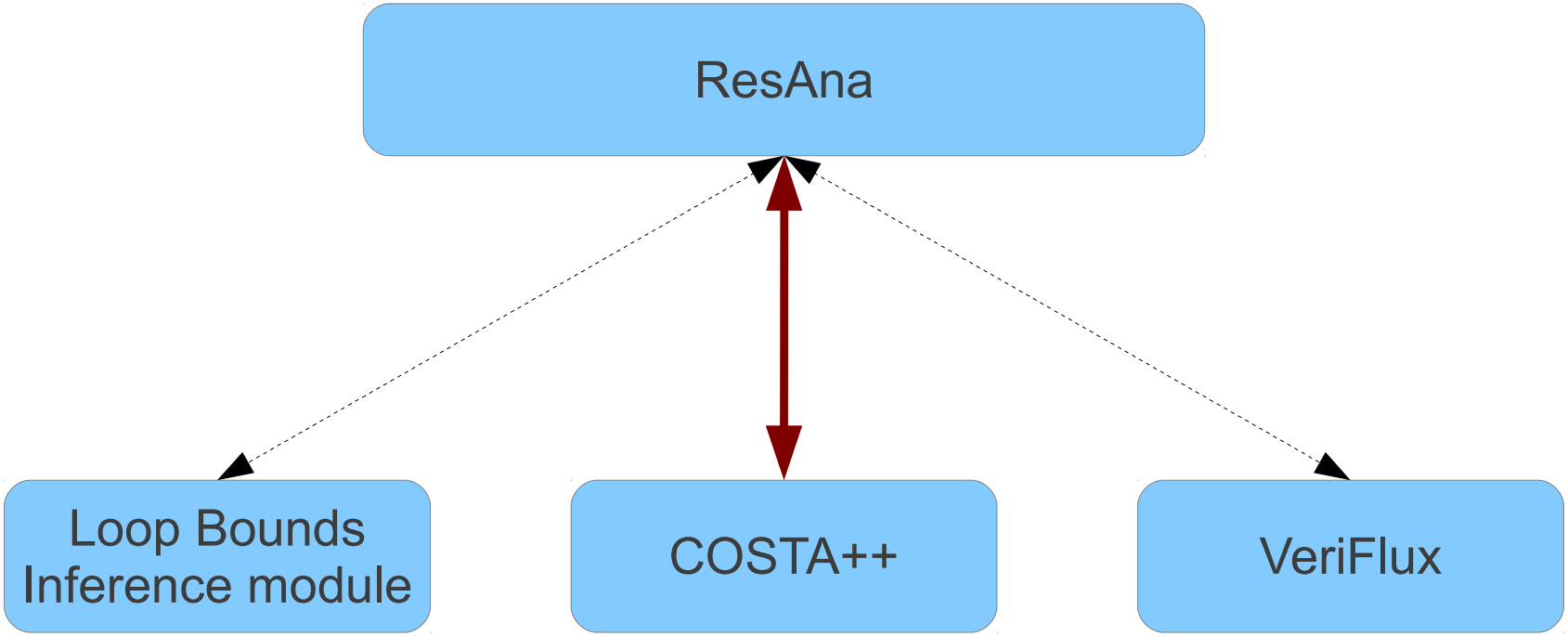


Further Reading

O. Shkaravska, R. Kersten, M. van Eekelen.
Test-Based Inference of Polynomial Loop-Bound Functions.
PPPJ'10: Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java

http://resourceanalysis.cs.ru.nl/

Heap Analysis



The COSTA System

COSTA = COSt and Termination Analyzer for Java Bytecode

Universidad Complutense de Madrid (UCM)

Elvira Albert
Puri Arenas
Samir Genaim
Diego Alonso
Jesús Correas
Miguel Gómez-Zamalloa

Universidad Politécnica de Madrid (UPM)

Germán Puebla
Damiano Zanardini
Abu Naser Masud
Diana Ramrez
José Miguel Rojas
Guillermo Román

Aim: Compute an upper bound to the cost of a given program in terms of the size of the input.

The COSTA System

Approach based on [Wegbreit 1975]:

1. Given a program and a cost model, produce a set of equations specifying the cost of the program.

```
public void f(int n) {
  while (n > 0) {
    int[] array = new int[n];
    n--;
  }
}
```

COSTA

2. Compute a nonrecursive form of the solution (closed form)

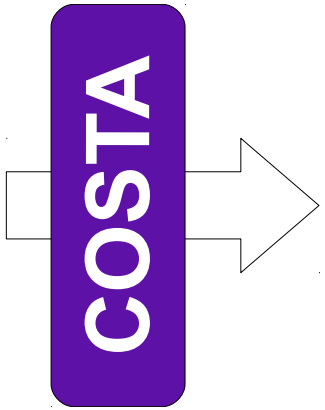
$$f(n) = 4n^2$$

The COSTA System

Approach based on [Wegbreit 1975]:

1. Given a program and a cost model, produce a set of equations specifying the cost of the program.

```
public void f(int n) {
  while (n > 0) {
    int[] array = new int[n];
    n--;
  }
}
```



Recurrence relation:

$$f(n)=0 \quad \{n \leqslant 0\}$$
$$f(n)=4n+f(n-1) \quad \{n>0\}$$

2. Compute a nonrecursive form of the solution (closed form)

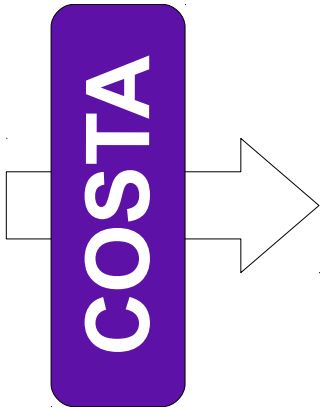
$$f(n) = 4n^2$$

The COSTA System

Approach based on [Wegbreit 1975]:

1. Given a program and a cost model, produce a set of equations specifying the cost of the program.

```
public void f(int n) {
  while (n > 0) {
    int[] array = new int[n];
    n--;
  }
}
```



Recurrence relation:

$$f(n)=0 \quad \{n \leqslant 0\}$$
$$f(n)=4n+f(n-1) \quad \{n>0\}$$

guards

2. Compute a nonrecursive form of the solution (closed form)

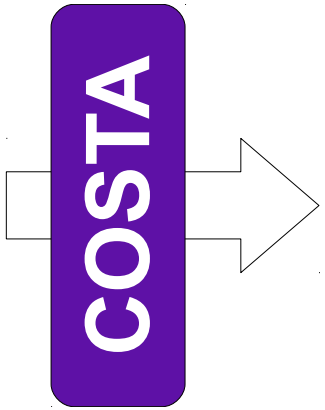
$$f(n) = 4n^2$$

The COSTA System

Approach based on [Wegbreit 1975]:

1. Given a program and a cost model, produce a set of equations specifying the cost of the program.

```
public void f(int n) {
  while (n > 0) {
    int[] array = new int[n];
    n--;
  }
}
```



Recurrence relation:

$$f(n)=0 \quad \{n \leqslant 0\}$$
$$f(n)=4n+f(n-1) \quad \{n>0\}$$

guards

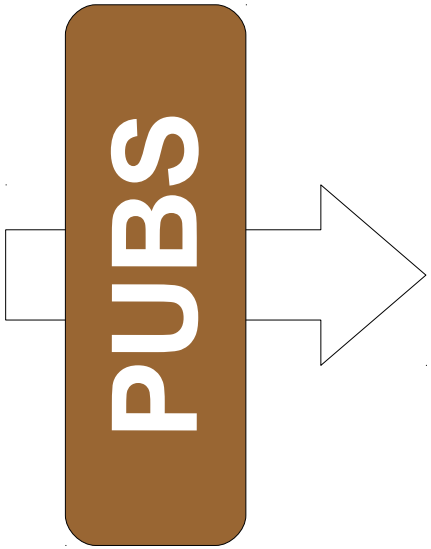
2. Compute a nonrecursive form of the solution (closed form)

$$f(n) = 4n^2$$

PUBS System

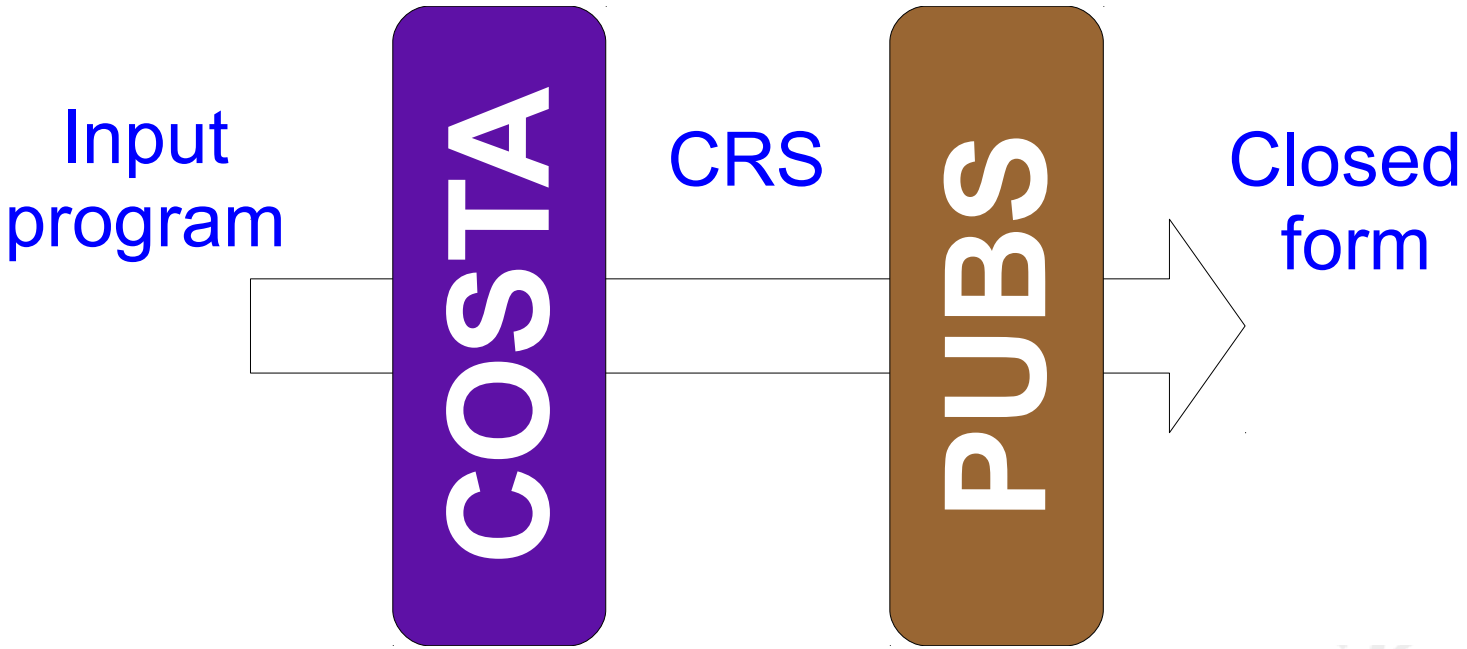
PUBS = Practical Upper Bounds Solver

$$f(n)=0 \quad \{n \leqslant 0\}$$
$$f(n)=4n+f(n-1) \quad \{n>0\}$$



$$f(n)=4n^2$$

Overview of COSTA architecture



Our extensions

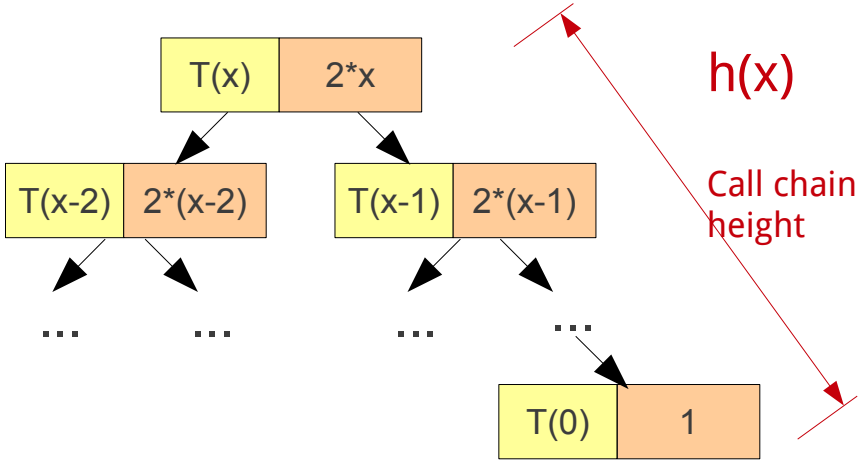
- Interpolation-based height analysis (COSTA++)
- Simplification of bounds
- Correct array-size analysis
- Virtual-machine specialisation

Interpolation-based height analysis

- PUBS is based on the notion of evaluation trees.

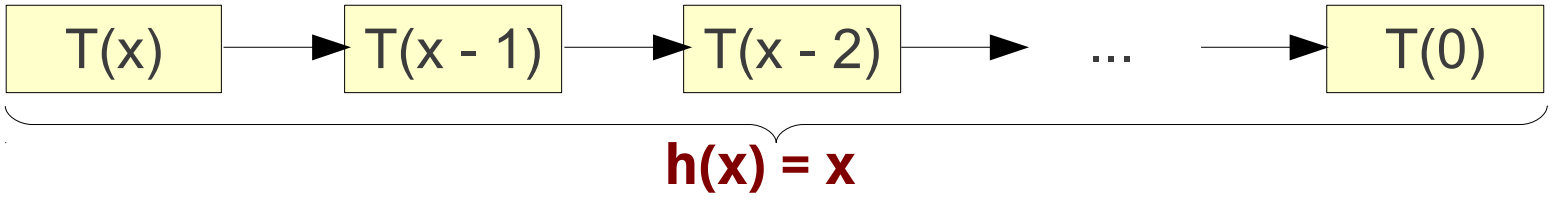
$T(x) = 1 \quad \{ x \leq 1 \}$

$T(x) = 2x + T(x-1) + T(x-2) \quad \{ x > 1 \}$



Interpolation-based height analysis

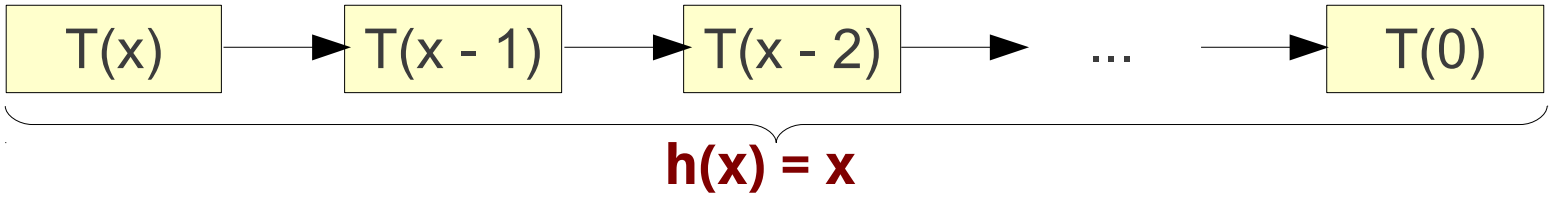
- Call-chain height $h(x)$: Upper-bound to the maximum number of unfoldings that may be undergone to reach a base case.
- It is closely related to the concept of ranking function.



A. Podelski, A. Rybalchenko.
A Complete Method for the Synthesis of Linear Ranking Functions.
5th International Conference on Verification, Model Checking and Abstract Interpretation, VMCAI 2004

Interpolation-based height analysis

- Call-chain height $h(x)$: Upper-bound to the maximum number of unfoldings that may be undergone to reach a base case.
- It is closely related to the concept of ranking function.



A. Podelski, A. Rybalchenko.
A Complete Method for the Synthesis of Linear Ranking Functions.
5th International Conference on Verification, Model Checking and Abstract Interpretation, VMCAI 2004

Interpolation-based height analysis

- The length of the maximal call chain associated with a CRS may not linearly depend on the sizes of the arguments.

Example

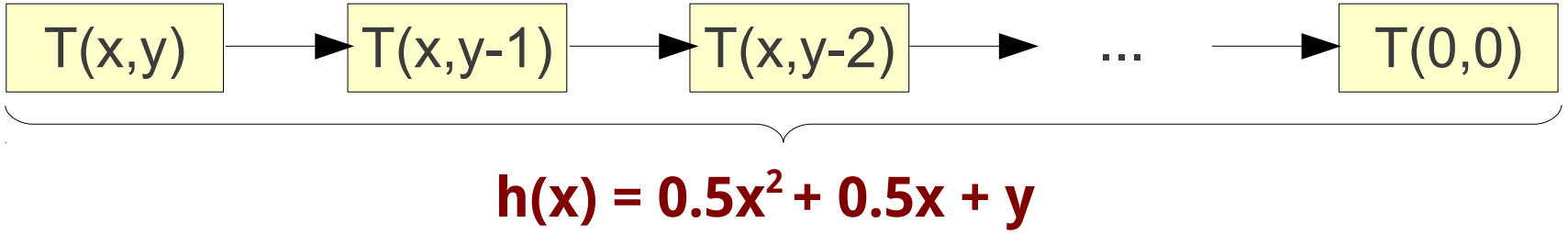
$$
\begin{array}{lcll}
T(x,y) & = & \mathsf{nat}(x) & \{x=0, y=0\} \\
T(x,y) & = & \mathsf{nat}(x) + T(x-1, x-1) & \{x>0, y=0\} \\
T(x,y) & = & \mathsf{nat}(x) + T(x, y-1) & \{x \geq 0, y>0\}
\end{array}
$$

Interpolation-based height analysis

- The length of the maximal call chain associated with a CRS may not linearly depend on the sizes of the arguments.

Example

$$
\begin{array}{lcll}
T(x,y) & = & \mathsf{nat}(x) & \{x=0, y=0\} \\
T(x,y) & = & \mathsf{nat}(x) + T(x-1, x-1) & \{x>0, y=0\} \\
T(x,y) & = & \mathsf{nat}(x) + T(x, y-1) & \{x \geq 0, y>0\}
\end{array}
$$



Simplification of bounds

$$
\begin{aligned}
&(n) * \\
&\quad ((n) * (c((\texttt{java.lang.Object}, 1)) + \\
&\qquad\qquad c((\texttt{java.lang.Object}, 2))) + \\
&\quad (n) * (c((\texttt{java.lang.Object}, 1)) + \\
&\qquad\qquad c((\texttt{java.lang.Object}, 2))) + \\
&\quad (n) * (c((\texttt{java.lang.Object}, 1)) + \\
&\qquad\qquad c((\texttt{java.lang.Object}, 2))))
\end{aligned}
$$

$$\Downarrow$$

$$6n^2 * (\texttt{java.lang.Object})$$

Correct array-size analysis

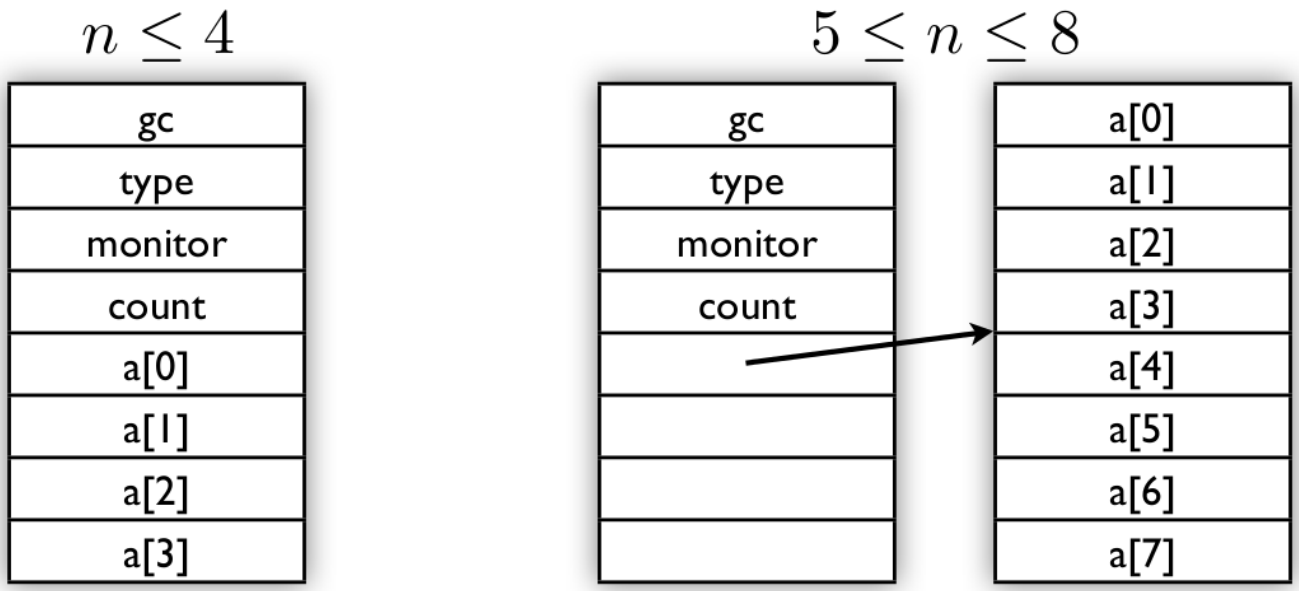
- Arrays are packed into data structures of the virtual machine
- Currently Costa calculates memory cost for an array using $new\ \ T\,[n] \Rightarrow n * c(size(primitiveType(T), 1)$
- Cost model should be $new\ \ T\,[n] \Rightarrow array(n, c(size(primitiveType(T), 1))$

Jamaica VM

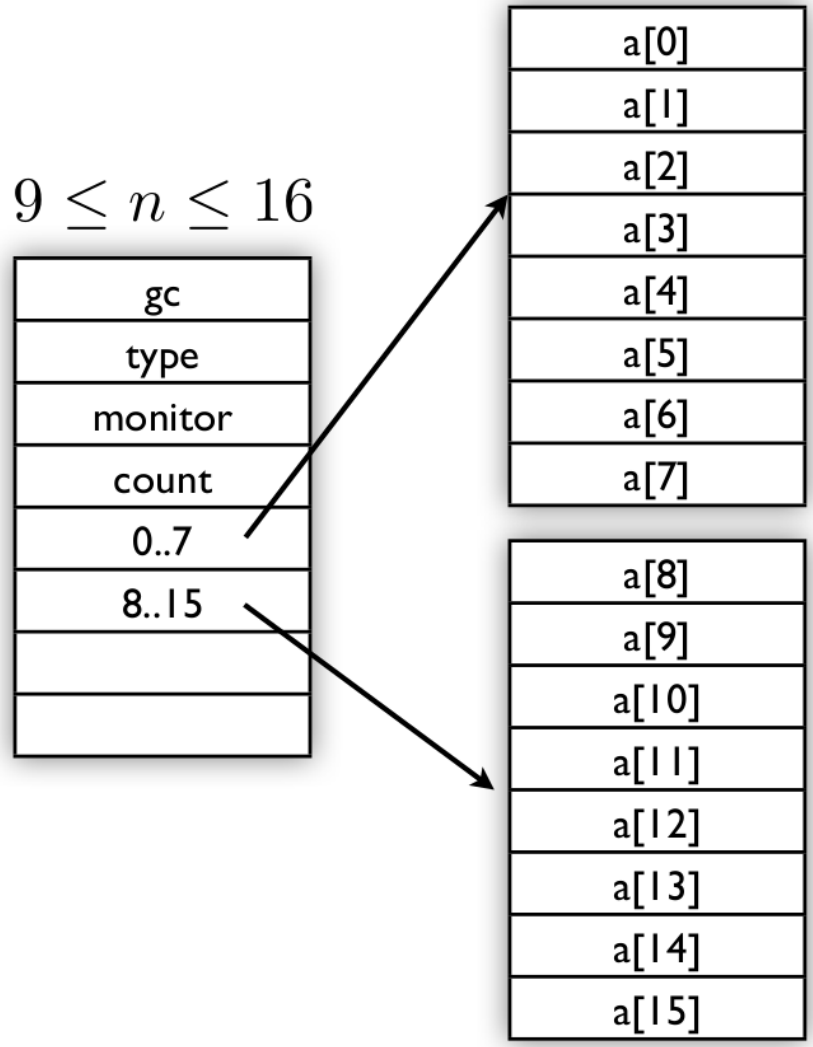
- Real-Time Java virtual machine
- Developed by Aicas GmbH (partner in CHARTER)
- Real-Time Garbage Collection by using an innovative memory allocator
- Many RTOS: VxWorks, QNX, Linux-variants, RTEMS
- Support for multicore execution

Virtual-machine specialisation

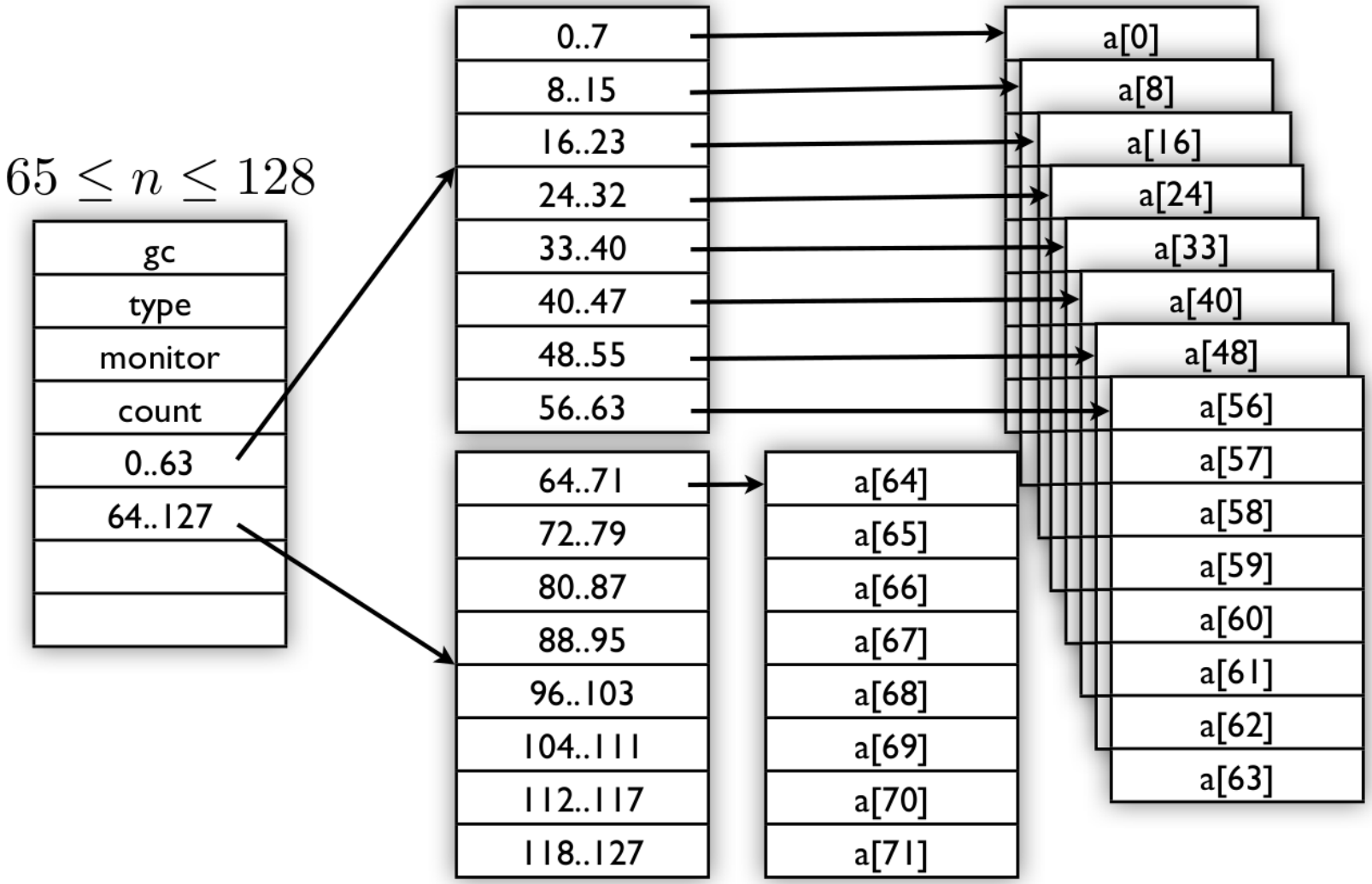
- An array is saved as an 8-tree with data stored only at the leafs (JamaicaVM)
- Every object/memory chunk in JamaicaVM is 32 bytes



Virtual-machine specialisation



Virtual-machine specialisation



Virtual-machine specialisation

- Specialise COSTA's results for JamaicaVM
- Calculate sound upper-bounds for array sizes:

$$\begin{aligned} \text{arrayblocks}(1..4) &= 1 \\ \text{arrayblocks}(n) &= \left\lceil \frac{n}{8} \right\rceil + \text{arrayblocks}(\left\lceil \frac{n}{8} \right\rceil) \end{aligned}$$

- Replace symbolic object sizes by actual sizes

$$6n^2 * (\texttt{java.lang.Object})$$

$$\Downarrow$$

$$192n^2$$

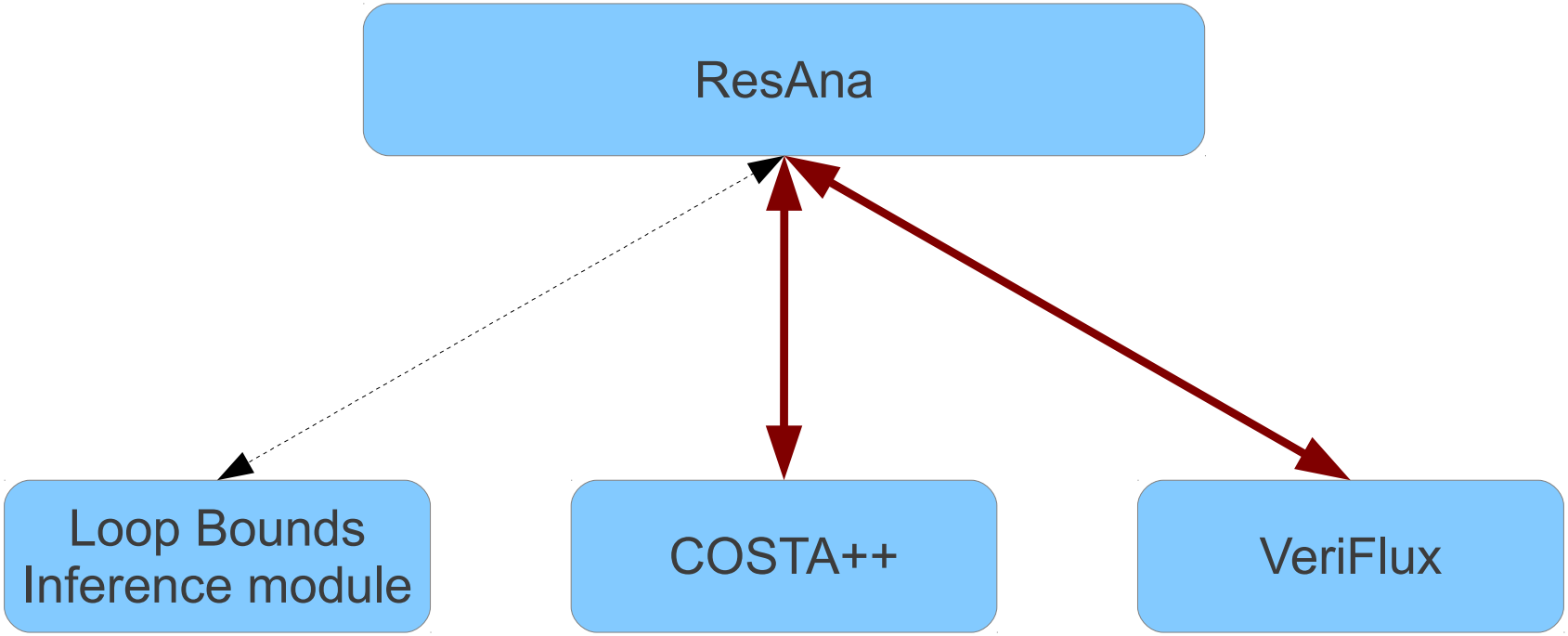
Further reading

E. Albert, P. Arenas, S. Genaim, G. Puebla., D.Zanardini
Cost Analysis of Java Bytecode.
16th European Symposium on Programming, ESOP 2007

M. Montenegro, O. Shkaravska, M. van Eekelen, R. Peña
Interpolation-based height analysis for improving a recurrence solver.
2nd International Workshop on Foundational Practical Aspects of Resource Analysis, FOPARA 2011

http://costa.ls.fi.upm.es/
http://resourceanalysis.cs.ru.nl/

Stack Analysis



VeriFlux

- Aicas GmbH Karlsruhe
- Combines various static analyses in one tool
  - Data-flow
  - Control-flow
  - ...
- Finds various problems
  - Run-time exceptions
  - Deadlocks
  - Race conditions
  - ...

`http://www.aicas.com/veriflux.html`

Stack Analysis

- COSTA++ outputs symbolic upper bound on the depth of recursion for a given method (height of the call tree)
- Use as `measured_by` annotation
  - Ranking function
  - Similar to `decreases` annotation
  - But for recursion instead of iterations
- Use VeriFlux's data-flow analysis to combine with input values
- Use VeriFlux to combine with stack-frame sizes

Stack Analysis

```
1 static int fib(int n) {
2     if (n < 2)
3         return n;
4     return fib(n-1) + fib(n-2);
5 }
```

```
1 public static void main(String [] args) {
2   fib(21);
3 }
```

Stack Analysis

```
1 //@ measured_by n;
2 static int fib(int n) {
3     if (n < 2)
4         return n;
5     return fib(n-1) + fib(n-2);
6 }
```

```
1 public static void main(String [] args) {
2   fib(21);
3 }
```

Stack Analysis

```
1 //@ measured_by n;
2 static int fib(int n) {
3     if (n < 2)
4         return n;
5     return fib(n-1) + fib(n-2);
6 }
```

```
1 public static void main(String [] args) {
2   fib(21);
3 }
```

$$21 \cdot 56 + 36 = 1212 \text{ bytes}$$

Practical Experience

- Loop-Bound Inference tested on several case-studies: could analyse 66% of the loops
  - Collision detector from Hunt et al. *Provably correct loops bounds for realtime Java programs*, JTRES'06
  - DIANA avionics package
  - CD_X Collision Detector package
- ResAna was used by NLR in the development of a safety-critical avionics application
  - Found easy to use
  - Feedback led to several improvements

Future Work

- Loop Bound Analysis
  - Larger case study to identify most needed improvements
  - Combine with syntactical pattern-matching method (Fulara and Jakubczyk. *Practically Applicable Formal Methods*, SOFSEM'10)
  - Use JML annotations
- Heap Space Analysis
  - Specialisation for OpenJDK

Conclusions

- Various resources analyses combined into one tool: ResAna
- Loop-bound analysis
  - Polynomial ranking functions for loops
  - Based on interpolation of test-run results
- Heap analysis
  - Enhanced version of COSTA
  - Interpolation-based height analysis
  - Simplification of bounds
  - Correct array-size analysis
  - Virtual-machine specialisation
- Stack analysis
  - Use COSTA to generate measured_by annotation
  - Use VeriFlux to calculate stack usage

http://resourceanalysis.cs.ru.nl/resana

Conclusions

- Various resources analyses combined into one tool: ResAna
- Loop-bound analysis
  - Polynomial ranking functions for loops
  - Based on interpolation of test-run results
- Heap analysis
  - Enhanced version of COSTA
  - Interpolation-based height analysis
  - Simplification of bounds
  - Correct array-size analysis
  - Virtual-machine specialisation
- Stack analysis
  - Use COSTA to generate measured_by annotation
  - Use VeriFlux to calculate stack usage

http://resourceanalysis.cs.ru.nl/resana

Conclusions

- Various resources analyses combined into one tool: ResAna
- Loop-bound analysis
  - Polynomial ranking functions for loops
  - Based on interpolation of test-run results
- Heap analysis
  - Enhanced version of COSTA
  - Interpolation-based height analysis
  - Simplification of bounds
  - Correct array-size analysis
  - Virtual-machine specialisation
- Stack analysis
  - Use COSTA to generate `measured_by` annotation
  - Use VeriFlux to calculate stack usage

http://resourceanalysis.cs.ru.nl/resana

Conclusions

- Various resources analyses combined into one tool: ResAna
- Loop-bound analysis
  - Polynomial ranking functions for loops
  - Based on interpolation of test-run results
- Heap analysis
  - Enhanced version of COSTA
  - Interpolation-based height analysis
  - Simplification of bounds
  - Correct array-size analysis
  - Virtual-machine specialisation
- Stack analysis
  - Use COSTA to generate `measured_by` annotation
  - Use VeriFlux to calculate stack usage

http://resourceanalysis.cs.ru.nl/resana

Conclusions

- Various resources analyses combined into one tool: ResAna
- Loop-bound analysis
  - Polynomial ranking functions for loops
  - Based on interpolation of test-run results
- Heap analysis
  - Enhanced version of COSTA
  - Interpolation-based height analysis
  - Simplification of bounds
  - Correct array-size analysis
  - Virtual-machine specialisation
- Stack analysis
  - Use COSTA to generate `measured_by` annotation
  - Use VeriFlux to calculate stack usage

`http://resourceanalysis.cs.ru.nl/resana`