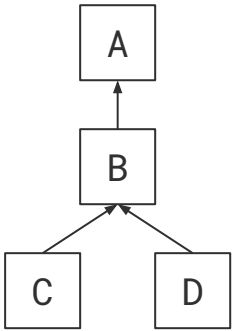
Compact Interface Method Table Layout

Ivan Trepakov Novosibirsk State University
Pavel Pavlov Huawei Novosibirsk Research Center

Inheritance

Single

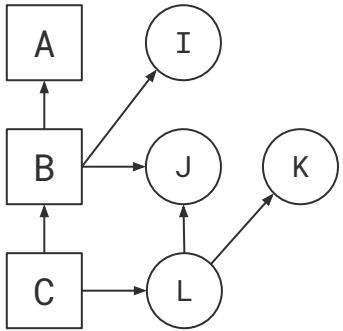


Modula 3

simula

Oberon-2

Interface



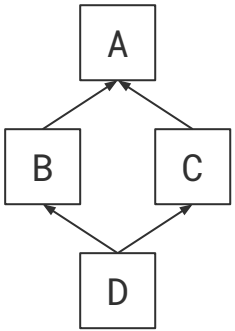
Java

Swift

Ruby

Free Pascal

Multiple



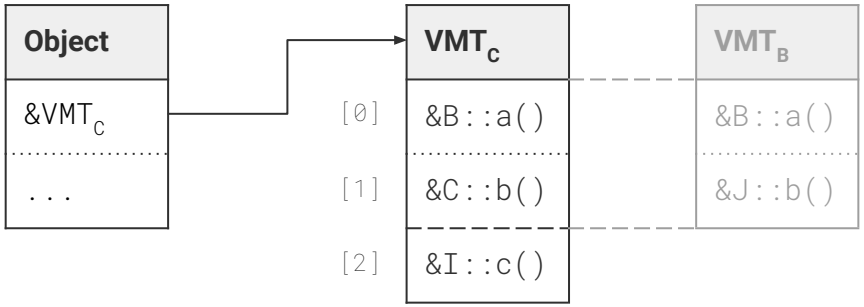
Eiffel

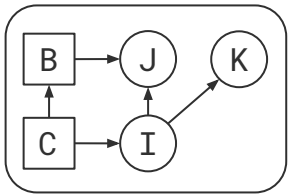
OCaml

Virtual and interface method tables

$\mathbf{VMT}_C$: *Array[Addr]*

$\mathbf{vnum}_C$: *Method* $\longrightarrow$ *Int*,
  C <: B, m $\in$ methods(B) $\Longrightarrow$
    $\mathbf{VMT}_C[\mathbf{vnum}_B(m)] = \mathbf{impl}_C(m)$





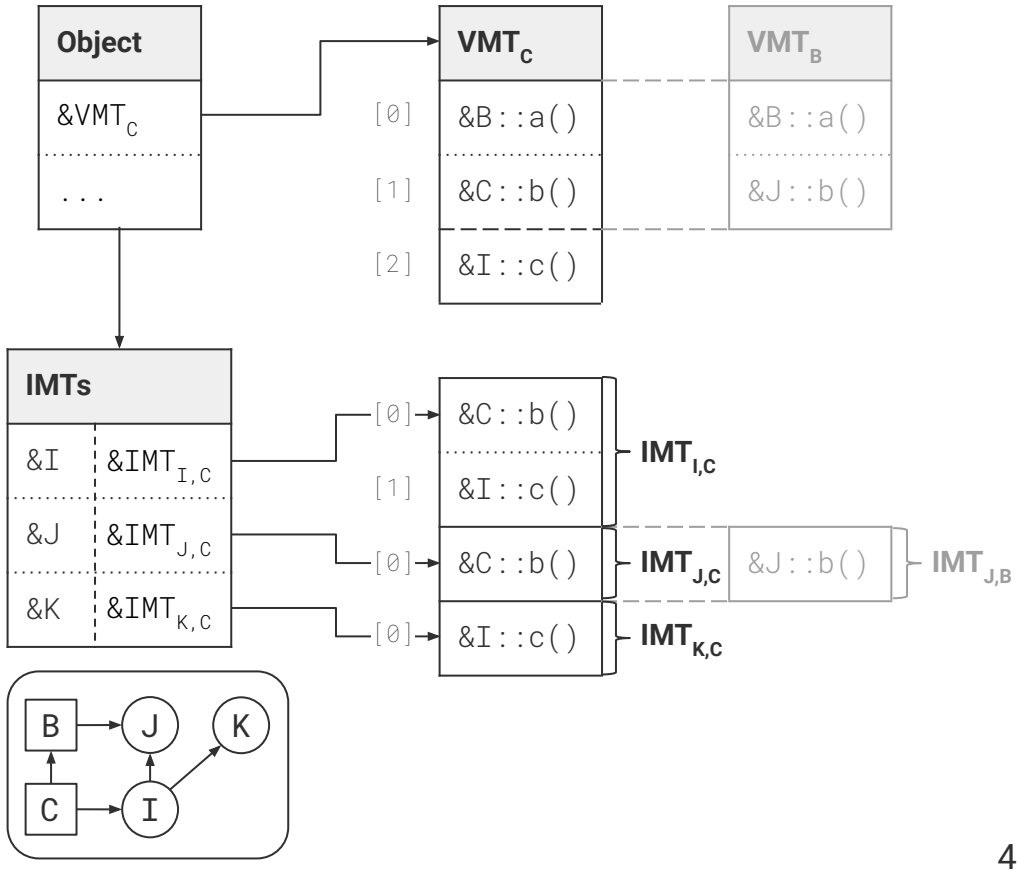
Virtual and interface method tables

$\mathtt{VMT_C}$: *Array[Addr]*

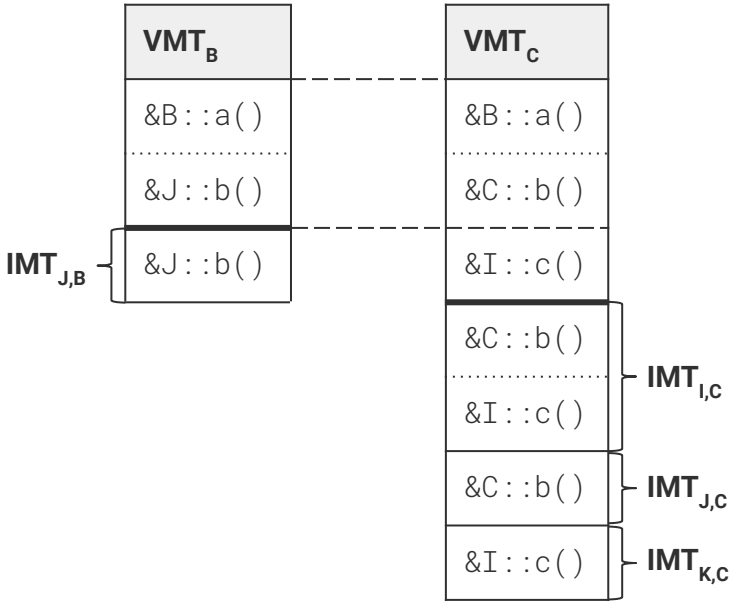
$\mathtt{vnum_C}$: *Method* $\longrightarrow$ *Int*,
  C <: B, m $\in$ methods(B) $\Longrightarrow$
    $\mathtt{VMT_C[vnum_B(m)]}$ = $\mathtt{impl_C(m)}$

$\mathtt{IMT_{I,C}}$: *Array[Addr]*

$\mathtt{vnum_I}$: *Method* $\longrightarrow$ *Int*,
  C <: I, m $\in$ methods(I) $\Longrightarrow$
    $\mathtt{IMT_{I,C}[vnum_I(m)]}$ = $\mathtt{impl_C(m)}$



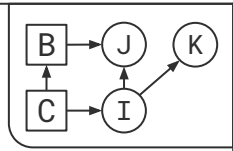
Compact layout



VMT layout

1. VMT_{super}
2. methods \ $methods_{super}$

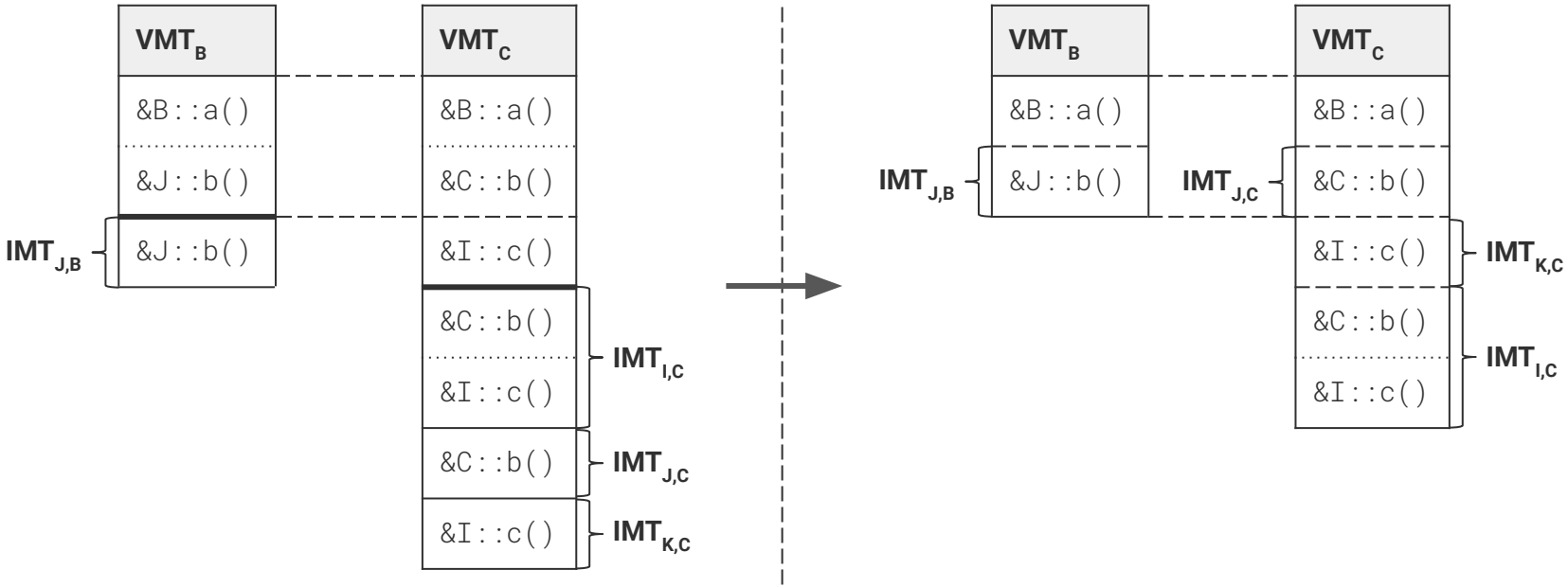
- Base layout



IMT layout

1. methods

Compact layout

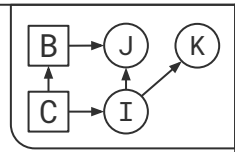


VMT layout

IMTs sorted by size ↑

1. VMT_{super}
2. $methods \setminus methods_{supers}$
3. $IMTs \setminus IMTs_{super}$

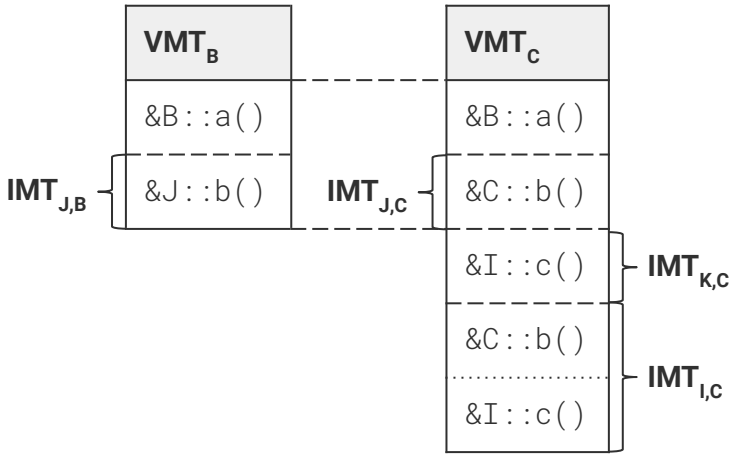
- Base layout
- Sharing VMT and IMT entries



IMT layout

1. methods

Compact layout

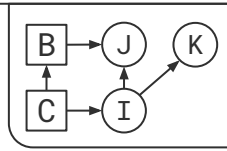


VMT layout

IMTs sorted by size ↑

1. VMT_{super}
2. $methods \setminus methods_{supers}$
3. $IMTs \setminus IMTs_{super}$

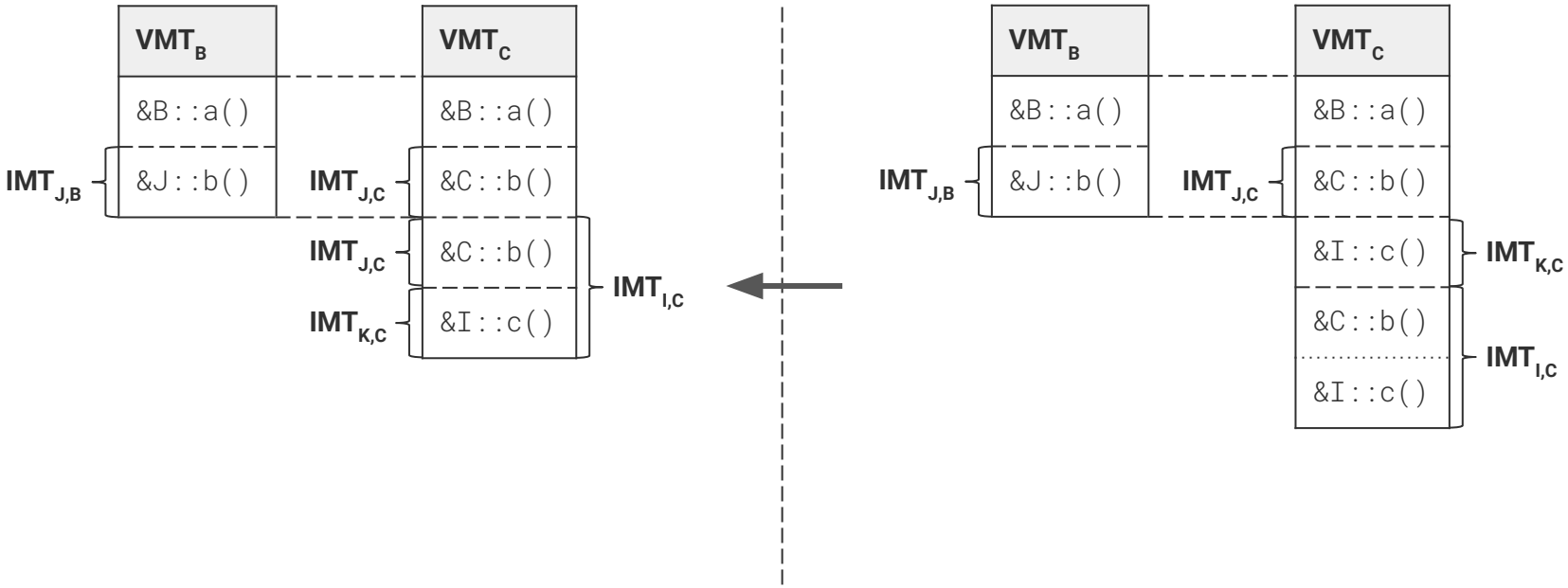
- Base layout
- Sharing VMT and IMT entries



IMT layout

1. methods

Compact layout

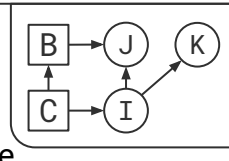


VMT layout

IMTs sorted by size ↑

1. VMT_{super}
2. methods \ $methods_{supers}$
3. IMTs \ $IMTs_{super}$

- Base layout
- Sharing VMT and IMT entries
- Superinterface layout inheritance

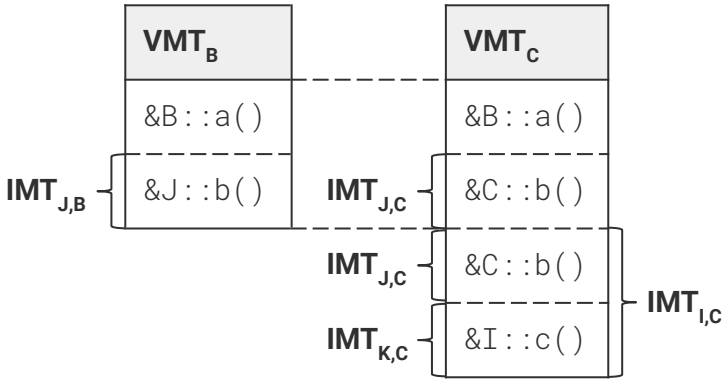


IMT layout

IMTs sorted by size ↓

1. *distinct*(IMTs)
2. methods \ $methods_{supers}$

Compact layout

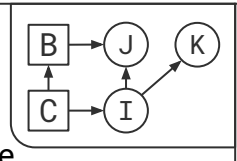


VMT layout

IMTs sorted by size ↑

1. $\text{VMT}_{\text{super}}$
2. $\text{methods} \setminus \text{methods}_{\text{supers}}$
3. $\text{IMTs} \setminus \text{IMTs}_{\text{super}}$

- Base layout
- Sharing VMT and IMT entries
- Superinterface layout inheritance

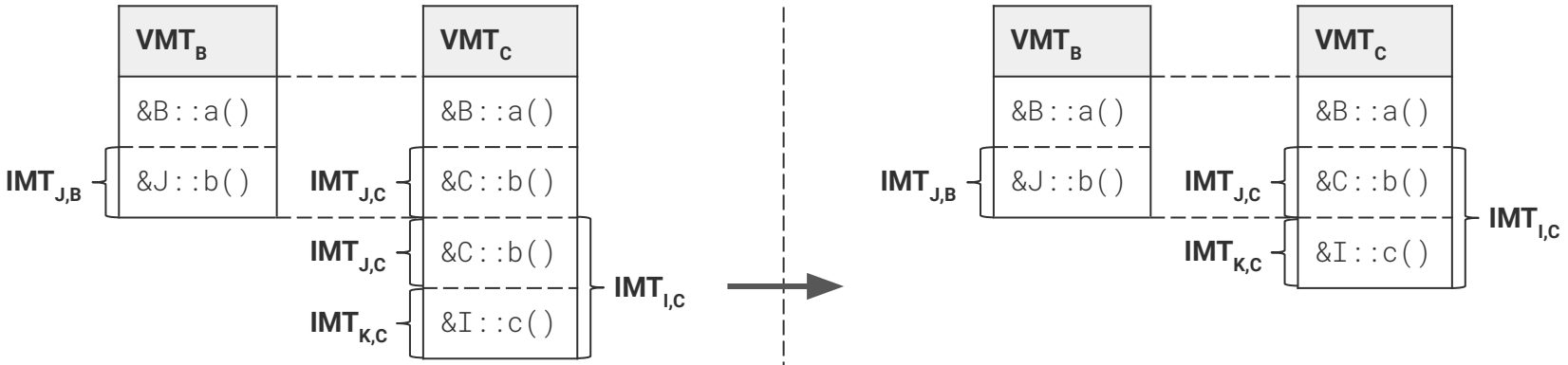


IMT layout

IMTs sorted by size ↓

1. *distinct*(IMTs)
2. $\text{methods} \setminus \text{methods}_{\text{supers}}$

Compact layout

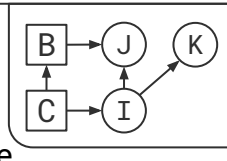


VMT layout

IMTs sorted by size ↑

1. VMT_{super}
2. **extend last IMT**
3. methods \ $methods_{supers}$
4. IMTs \ $IMTs_{super}$

- Base layout
- Sharing VMT and IMT entries
- Superinterface layout inheritance
- Last superclass IMT extension

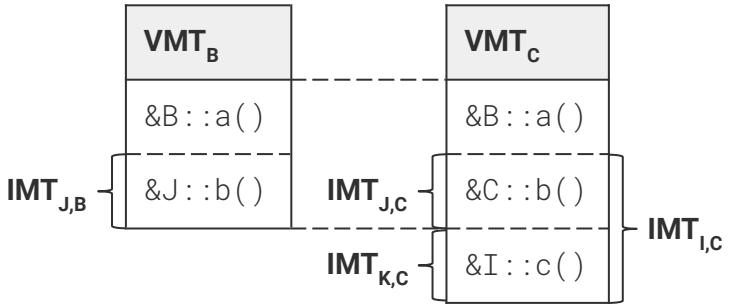


IMT layout

IMTs sorted by size ↓

1. *distinct*(IMTs)
2. methods \ $methods_{supers}$

Compact layout

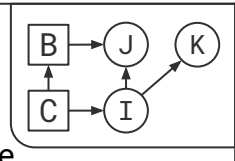


VMT layout

IMTs sorted by size ↑

1. VMT_{super}
2. extend last IMT
3. methods \ $methods_{supers}$
4. IMTs \ $IMTs_{super}$

- Base layout
- Sharing VMT and IMT entries
- Superinterface layout inheritance
- Last superclass IMT extension

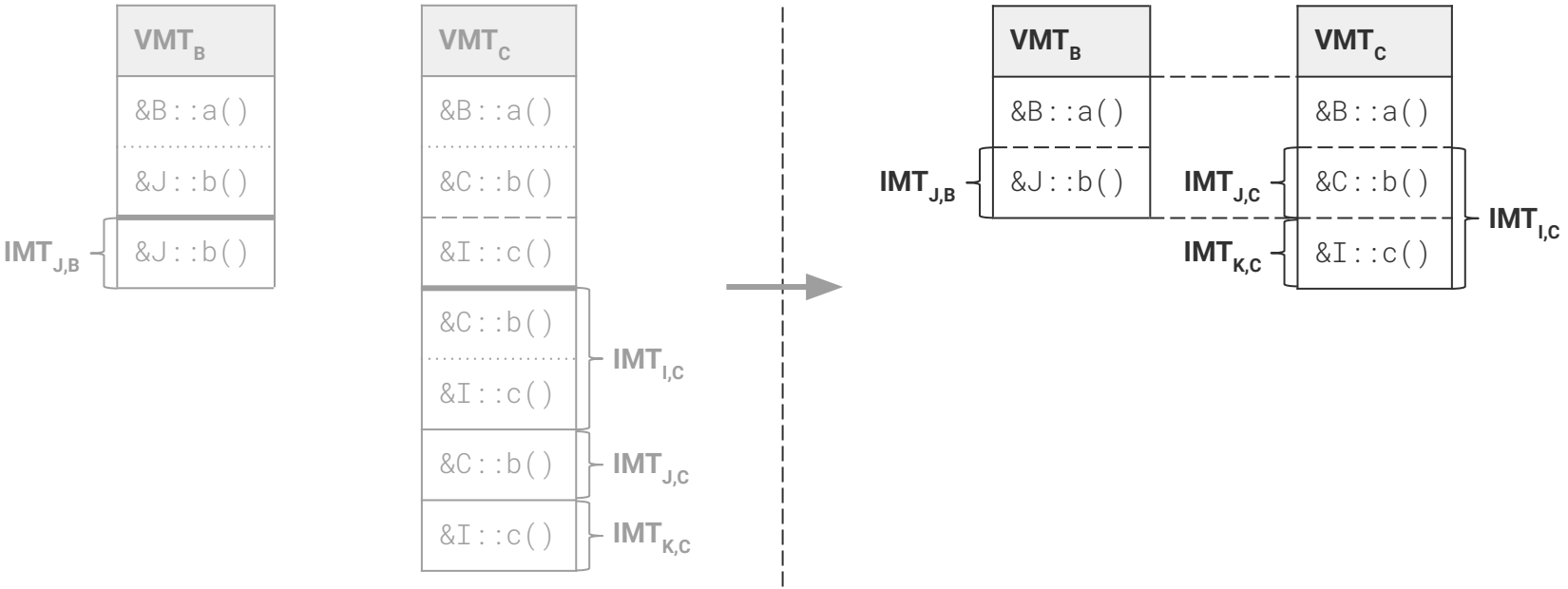


IMT layout

IMTs sorted by size ↓

1. *distinct*(IMTs)
2. methods \ $methods_{supers}$

Compact layout

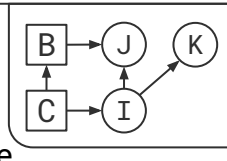


VMT layout

IMTs sorted by size ↑

1. $\text{VMT}_{\text{super}}$
2. extend last IMT
3. methods \ $\text{methods}_{\text{supers}}$
4. IMTs \ $\text{IMTs}_{\text{super}}$

- Base layout
- Sharing VMT and IMT entries
- Superinterface layout inheritance
- Last superclass IMT extension

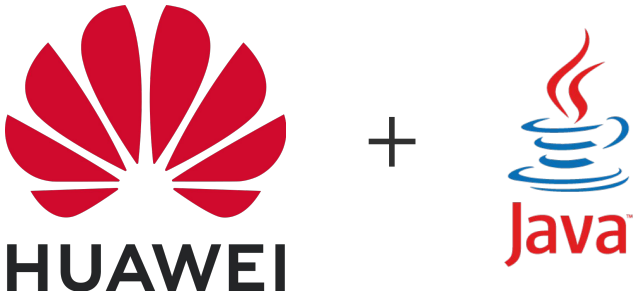


IMT layout

IMTs sorted by size ↓

1. *distinct*(IMTs)
2. methods \ $\text{methods}_{\text{supers}}$

Experimental setup



Experimental Huawei JDK Setup
(JVM + AOT-compiler)

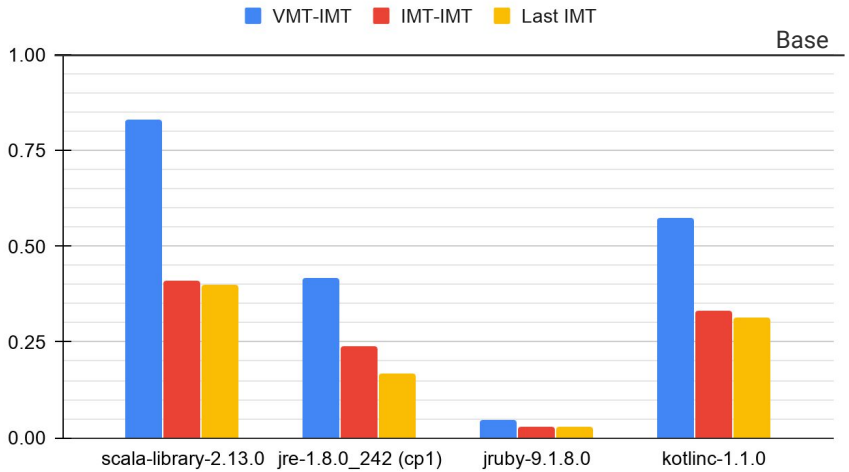
Experimental setup

Tested applications

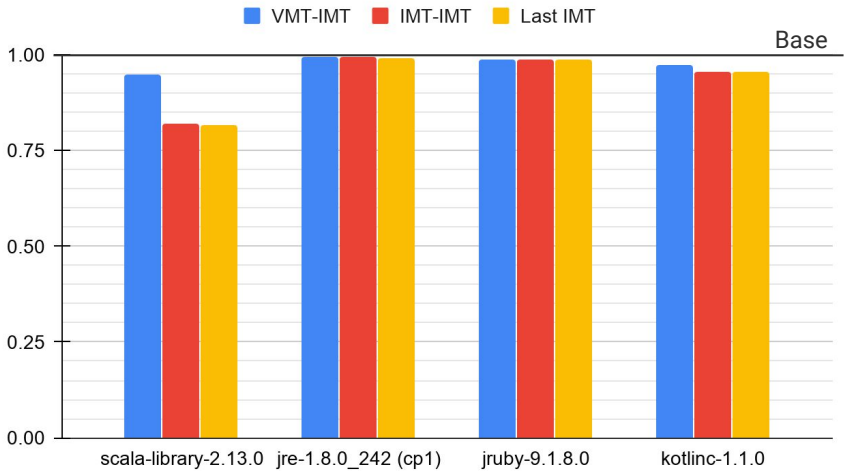
Application	Description	Languages	Executable size, KB
scala-library-2.13.0	Standard library for the Scala programming language	Scala	4 918
jre-1.8.0_242 (cp1)	Run-time environment for Java SE 8 with compact profile 1	Java	9 490
jruby-9.1.8.0	Implementation of the Ruby programming language on top of JVM	Ruby, Java	11 323
kotlinc-1.1.0	Compiler of the Kotlin programming language	Kotlin	13 247

Experimental results

Factor of entry duplication reduction

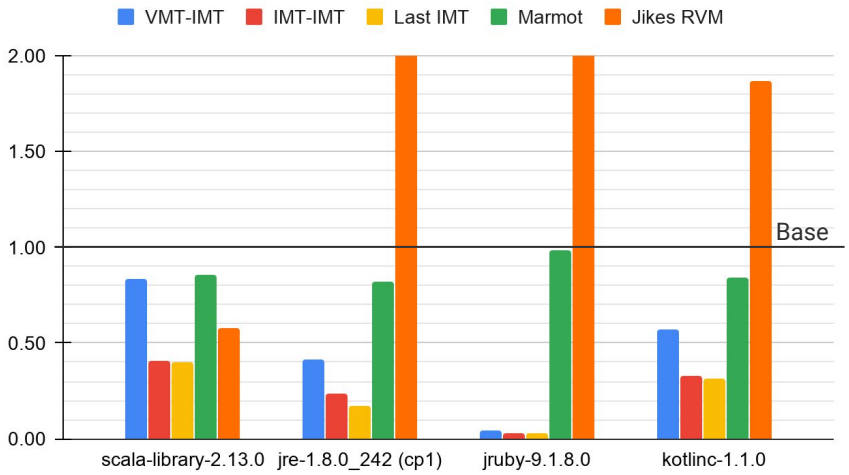


Factor of total executable size reduction

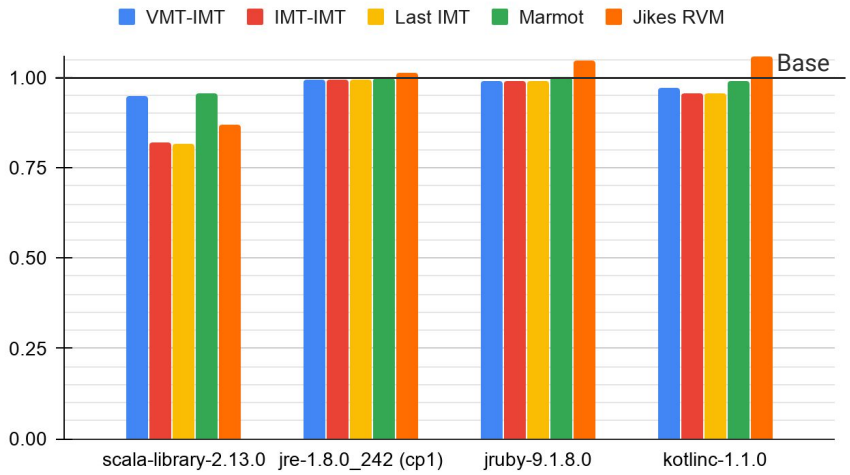


Experimental results against alternatives

Factor of entry duplication reduction



Factor of total executable size reduction



Marmot – research platform for studying the implementation of high level programming languages.
Jikes RVM – mature open source virtual machine that runs Java programs.

Conclusion and future work

Main results:

- We have implemented several IMT layout improvements targeted at overall table size reduction of one of the biggest run-time structures in the program.
- The proposed method was implemented in experimental Huawei JDK setup with AOT compilation support.
- The overall table size has been significantly reduced even compared to existing IMT implementations, resulting in a considerable executable size reduction.
- The described technique can be applied to a variety of object-oriented languages, increasing their viability for embedded development.

Future work:

- Further research is required to determine better IMT inheritance heuristics, instead of proposed “greedy” one.
- The resulting hierarchical IMT structure can be further exploited in certain optimization scenarios.

Thank you for your attention

Experimental results

Table I: Tested applications

Application	Description	Languages	Classes	Interfaces	Executable size, KB
scala-library-2.13.0	Standard library for the Scala programming language.	Scala	2 313	516	4 918
jre-1.8.0_242 (cp1)	Run-time environment for Java SE 8 with compact profile 1.	Java	4 858	610	9 490
jruby-9.1.8.0	Implementation of the Ruby programming language on top of JVM.	Ruby, Java	8 422	360	11 323
kotlinc-1.1.0	Compiler of the Kotlin programming language.	Kotlin	11 490	1 572	13 247

Table II: Factor of entry duplication reduction (< 1) or increase (> 1)

Application	VMT–IMT	IMT–IMT	Last IMT	Marmot+	Marmot	Jikes RVM
scala-library-2.13.0	**0.83**	0.41	**0.40**	0.69	0.86	0.58
jre-1.8.0_242 (cp1)	0.42	0.24	**0.17**	0.24	0.82	2.42
jruby-9.1.8.0	0.05	0.03	**0.03**	0.03	0.98	**5.23**
kotlinc-1.1.0	0.57	0.33	**0.31**	0.41	0.84	1.87

Table III: Total executable size, KB

Application	VMT–IMT	IMT–IMT	Last IMT	Marmot+	Marmot	Jikes RVM
scala-library-2.13.0	4 663 (0.95)	4 025 (0.82)	**4 012 (0.82)**	4 449 (0.90)	4 705 (0.96)	4 279 (0.87)
jre-1.8.0_242 (cp1)	9 440 (0.99)	9 424 (0.99)	9 418 (0.99)	9 424 (0.99)	9 475 (1.00)	9 614 (1.01)
jruby-9.1.8.0	11 200 (0.99)	11 198 (0.99)	11 198 (0.99)	11 198 (0.99)	11 321 (1.00)	11 870 (1.05)
kotlinc-1.1.0	12 881 (0.97)	12 672 (0.96)	12 657 (0.96)	12 743 (0.96)	13 110 (0.99)	**13 995 (1.06)**

Layout building time complexity

	Base	IMT - VMT	IMT - IMT	Last IMT
VMT Layout	m	$nm + nlogn$	$nm + nlogn$	$nm + nlogn$
IMT Layout	m	m	$nm + nlogn$	$nm + nlogn$
Σ	km	$k(nm + nlogn)$	$k(nm + nlogn)$	$k(nm + nlogn)$

$k = \#types;\ m = max_{types}(\#methods);\ n = max_{types}(\#superInterfaces)$

Superinterface distribution