Bidirectional Object Layout for Separate Compilation

Andrew Myers

MIT Lab for Computer Science

Contributions

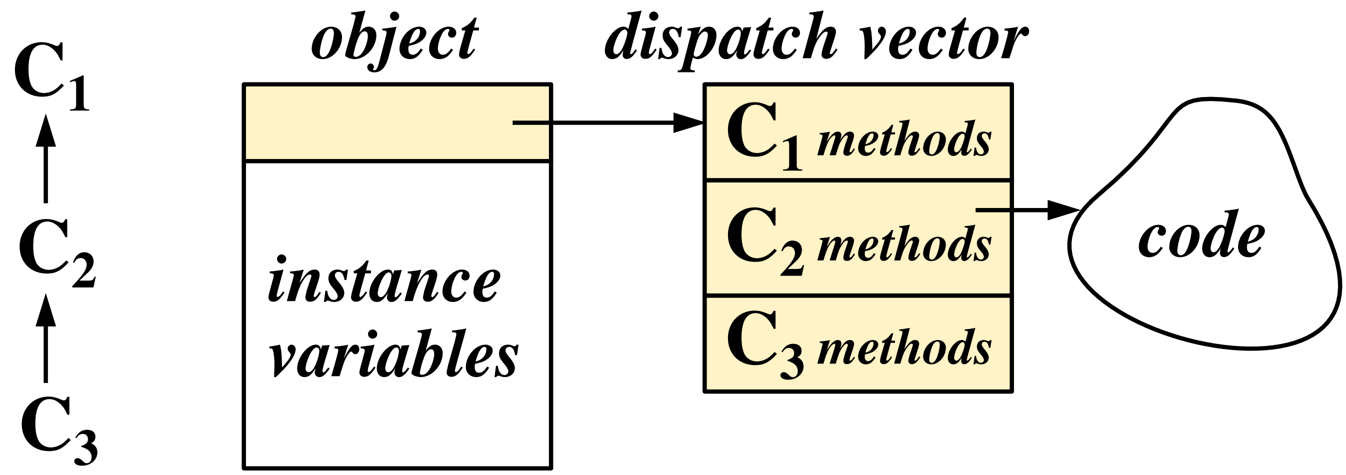
Bidirectional Object Layout (BOL)

- Compact objects
- Faster method dispatch
- Cheap incremental extension
  - no global analysis

Uses

- Complex type hierarchy
- Large/distributed/persistent systems
  - Separate compilation

Ideal Object Layout



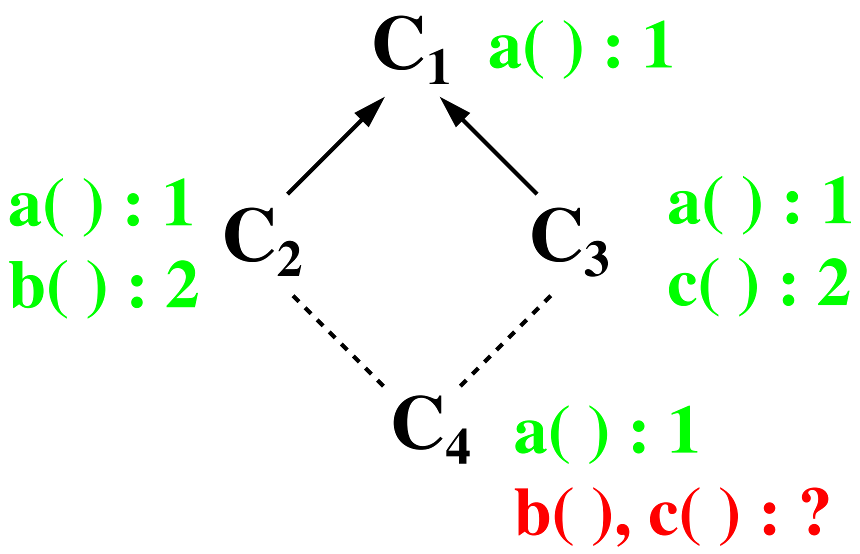
- Objects share a *dispatch vector (DV)*
- Methods accessed via *method indices*

\+ One word of dispatch overhead/object

− Doesn't scale to large systems

– complex type hierarchies

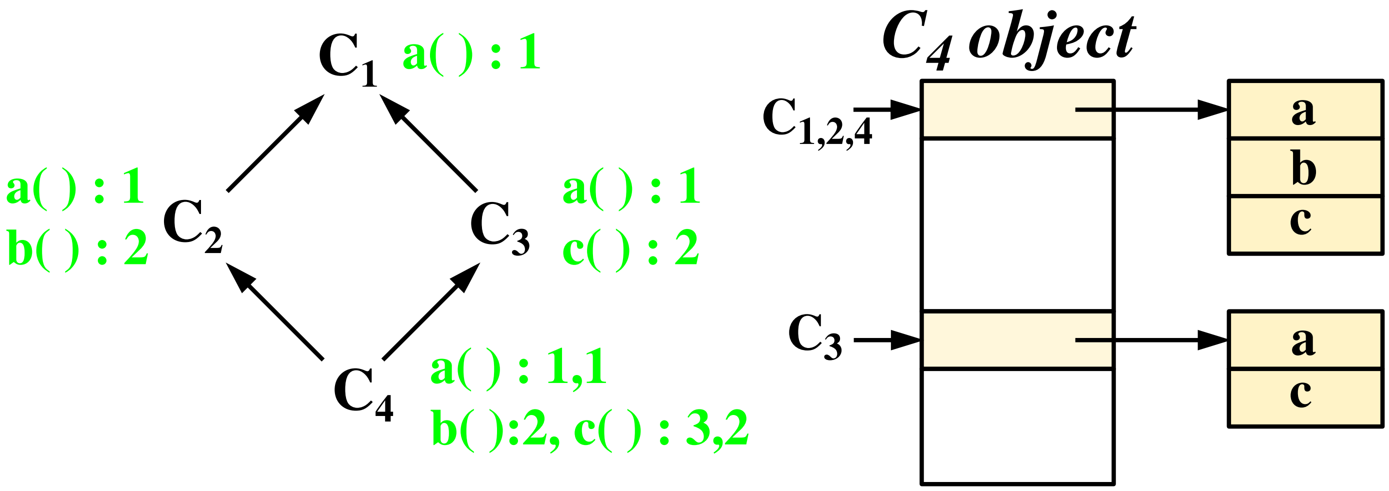
Problem: Conflicting Indices



- Conflict in superclass method indices
- Add $\mathbf{C_4} \rightarrow$ must change $\mathbf{C_2}$, $\mathbf{C_3}$ layout
- Efficient layouts hard to compute: global analysis
- Existing objects invalidated

C++ Solution: Multiple DV's

Idea: Resolve conflict with multiple DVs.



+ Dependency only on superclasses
+ Separate compilation possible
– Increasing space overhead
– Slower dispatch

Bidirectional Object Layout

- Multiple DV technique
- Local (superclass) dependency only
  - no global analysis

Advantages over C++ layout:

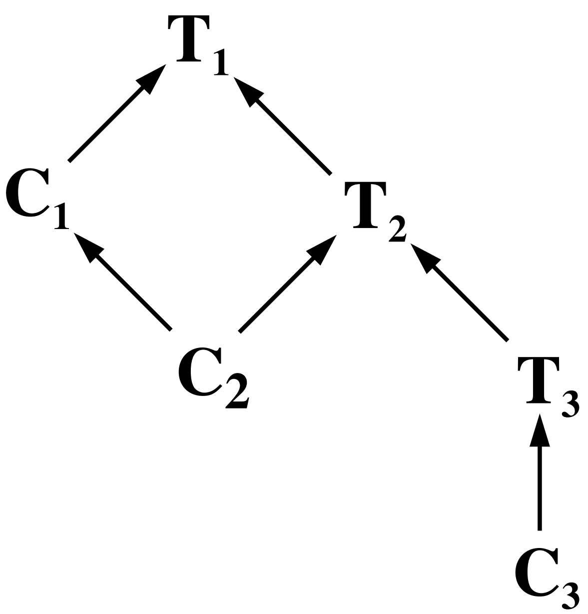
- Compact objects, faster dispatch
- Optimized for common hierarchies
- Automatically merges dispatch vectors

- BOL never worse than C++ layout
- Six formal layout rules
- Implemented for Theta, a persistent OOPL

Insight: Uses of Inheritance

Subtyping	$\begin{array}{c} \mathbf{T_1} \\ \uparrow \\ \mathbf{T_2} \end{array}$	Extends interface.
Abstraction	$\begin{array}{c} \mathbf{T} \\ \uparrow \\ \mathbf{C} \end{array}$	Separates interface and implementation. Allows reimplementation of **T**.
Implementation	$\begin{array}{c} \mathbf{C_1} \\ \uparrow \\ \mathbf{C_2} \end{array}$	Extends implementation to form a new class.

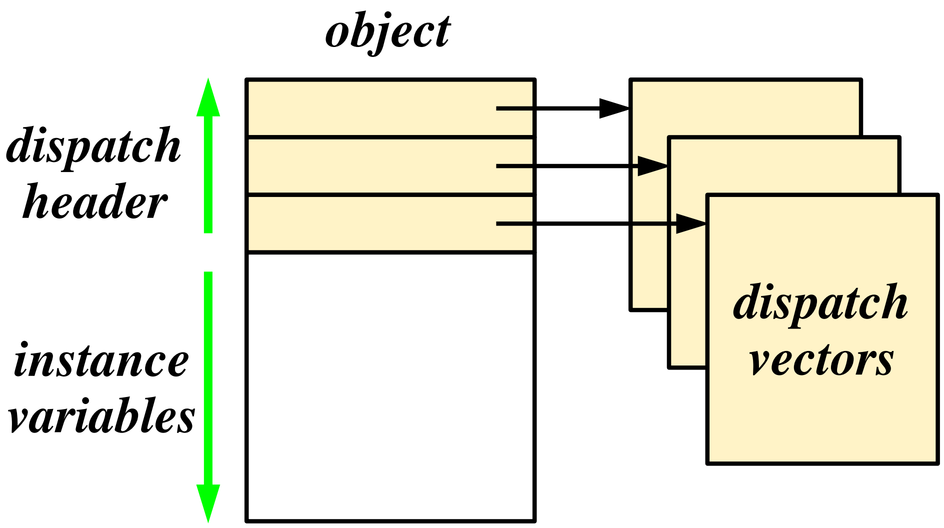
Common Hierarchies



- **Optimization Assumptions**
  - Separate interface/implementation hierarchies
  - Hierarchies mostly parallel

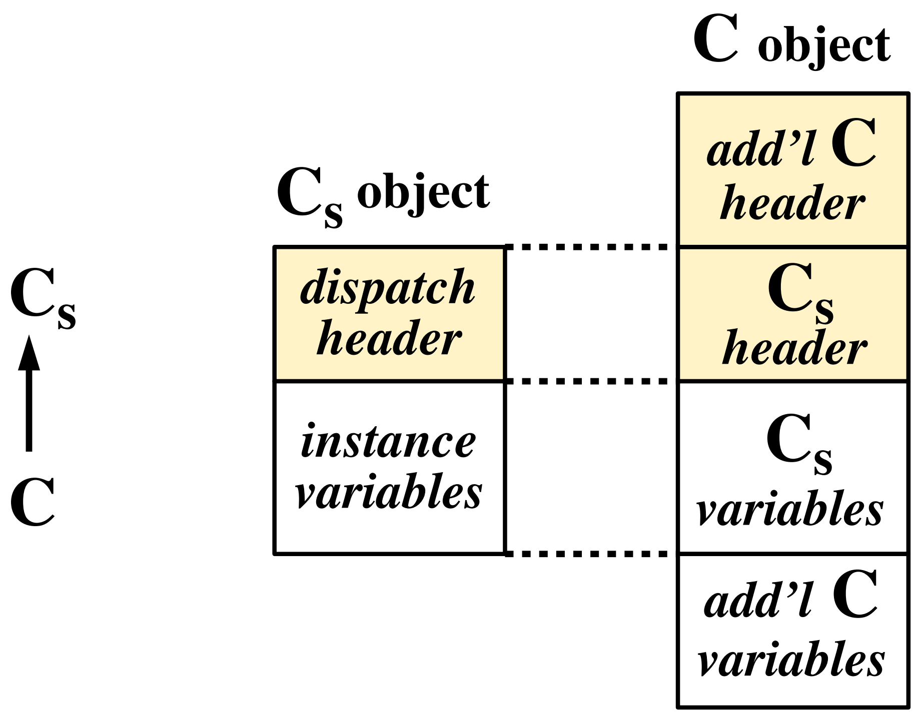
- **BOL never worse than C++ layout**

Bidirectional Object Layout



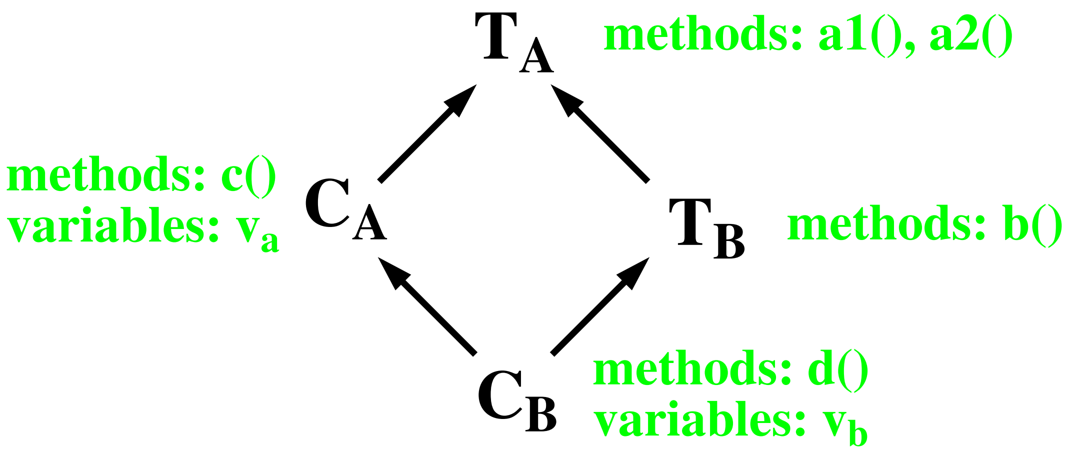
- Object composed of two separate pieces:
  - Dispatch header
  - Instance variables
- Grow in opposite directions
- Facilitates dispatch–vector merging

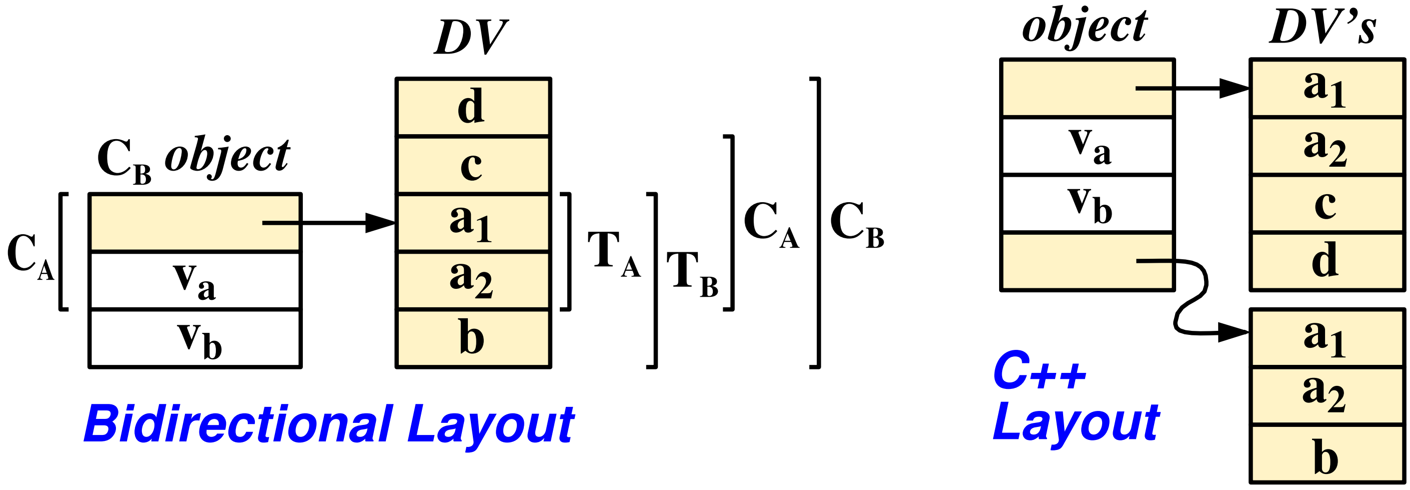
Subclass Compatibility



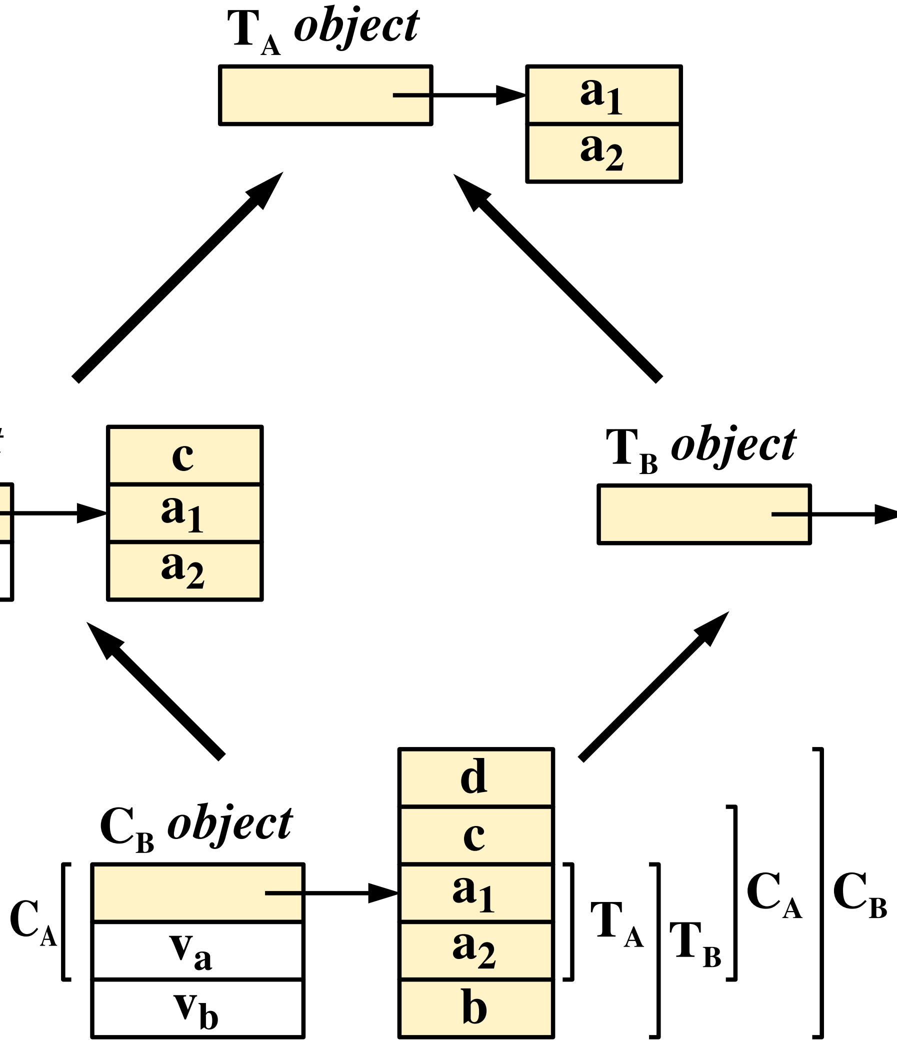
- Superclass $\mathbf{C_s}$ layout embedded in **C** layout
  - – no recompilation for inheritance

Example Layout

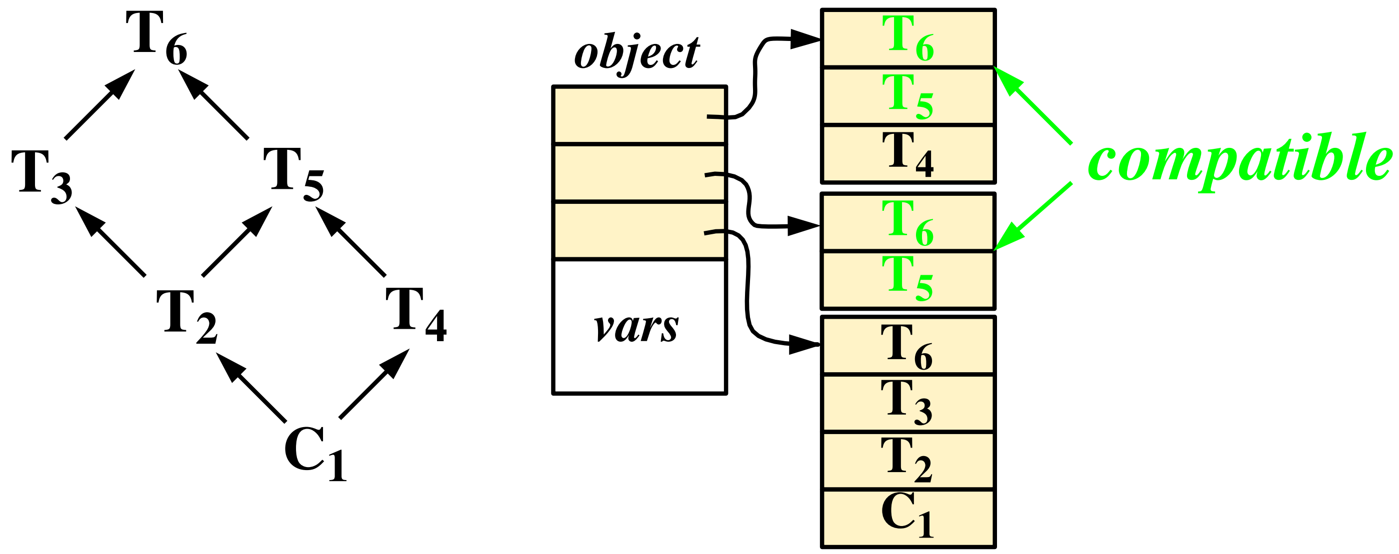




BOL Construction

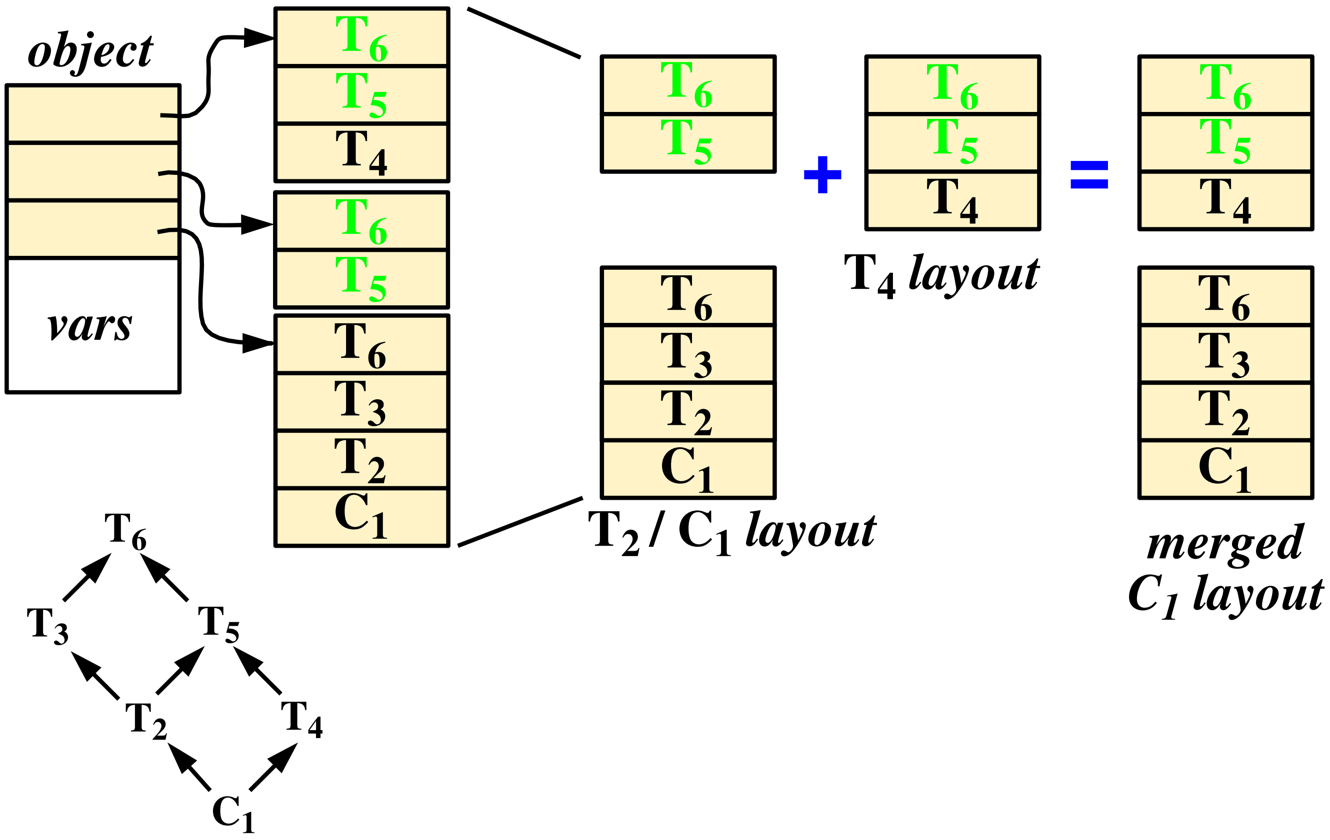


Merging



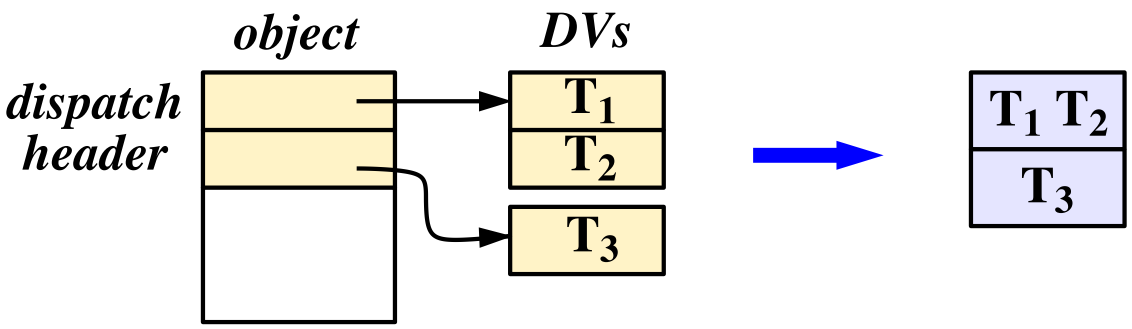
- *Lattices* result from separate hierarchies
- Redundant dispatch vectors
- Can *merge* dispatch vectors, compacting objects

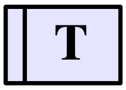
Constructing Layouts



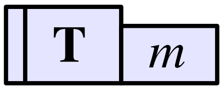
- Incrementally merge in superclass headers
- If no possible merge, append header

Notation



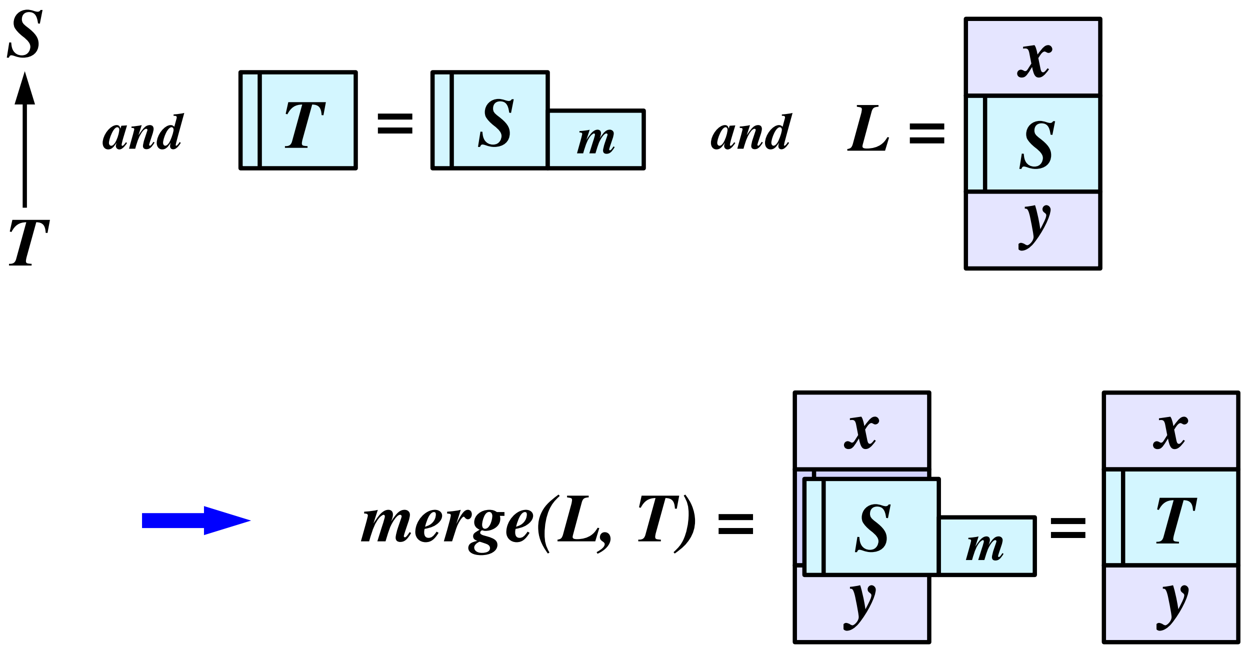
- Dispatch header: 2D array (box)
- Height of box is number of dispatch vectors
- Rows are dispatch vectors
- Layout of header for *T* written as 

- Horizontal abutment → appends to last DV

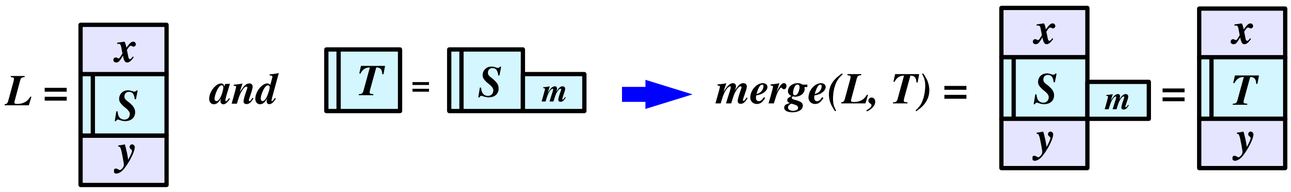


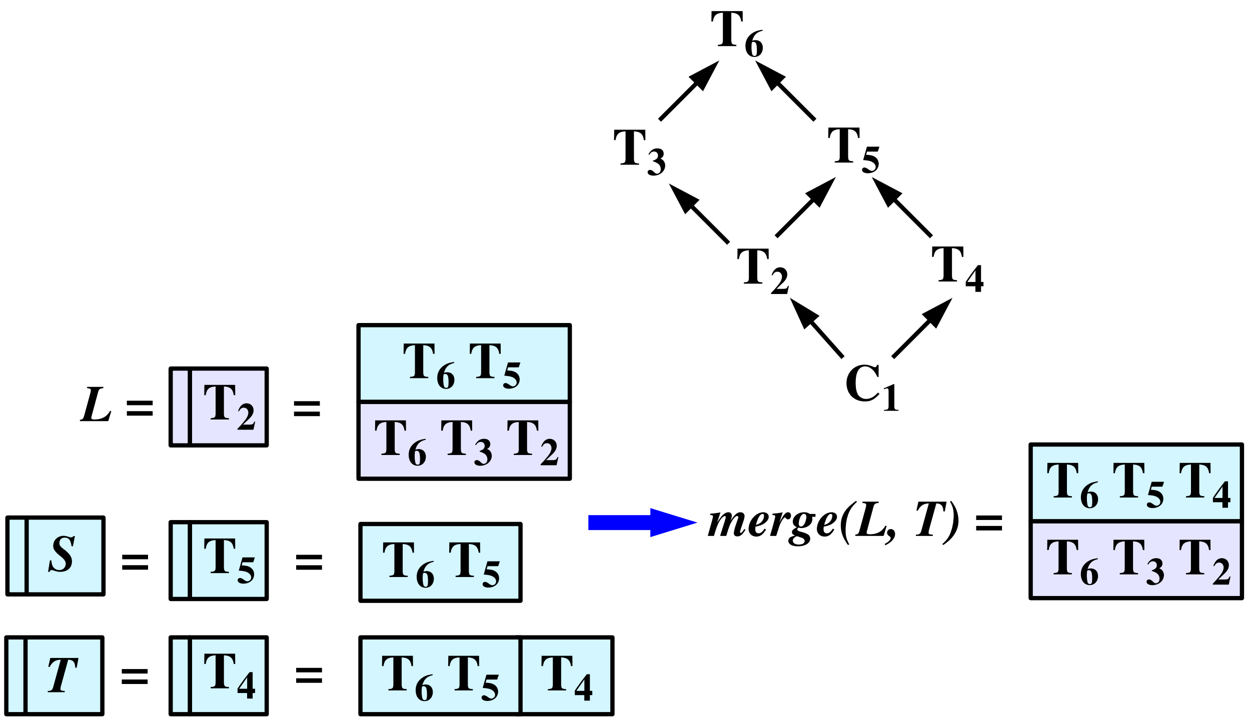
Merge Condition 1

Merging layout for T into existing layout L:

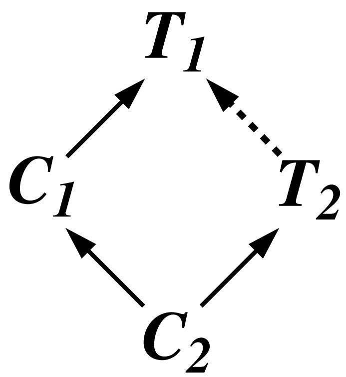


Merge Example





Results



- BOL objects have one dispatch vector in parallel type/class lattices
- Dispatch as fast, objects as compact as in ideal layout
- Handles complex types, no global analysis

Experiments

- Compared BOL, C++ layouts in two class libraries

Theta library

- ~100 types & classes
- Theta: enforced interface separation

Interviews 3.1

- ~430 classes
- C++: mixture of styles, much single–inheritance

Experimental Results

Theta

Overhead (words)	BOL	C++
1	44	23
2	1	17
3		4
4		1

Interviews 3.1

Overhead (words)	BOL	C++
1	399	372
2	17	24
3	8	17
4	2	6
5		3
6		1
≥7		3

- Theta classes significantly more compact
- Up to 4 words saved per object
- Generous to C++
- Much C++ overhead from hierarchy separation

Additional Contributions

- Formal object layout rules
- Showed correctness
- Multiple inheritance
- Method dispatch details

- Efficient implementation of signature refinement
  - contravariant arguments, covariant results

Related Work

B. Stroustrup. Multiple inheritance for C++. *Spring '87 EUSUGC*.

R. Dixon, T. McKee, P. Schweizer, M. Vaughan. A fast method dispatcher for languages with multiple inheritance. *OOPSLA '89*.

W. Pugh, G. Weddell. Two–directional record layout for multiple inheritance. *PLDI '90*.

Conclusion

Bidirectional Object Layout

- Compact objects
- Fast dispatch
- No global analysis: cheap extension

Applicability

- Most statically–typed OO languages
  – complex type hierarchy
- Separate compilation
  – large/distributed/persistent systems

Andrew Myers

MIT Lab for Computer Science

Mail: andru@lcs.mit.edu
Web: http://www.pmg.lcs.mit.edu/~andru