Analysis Separation without Visitors

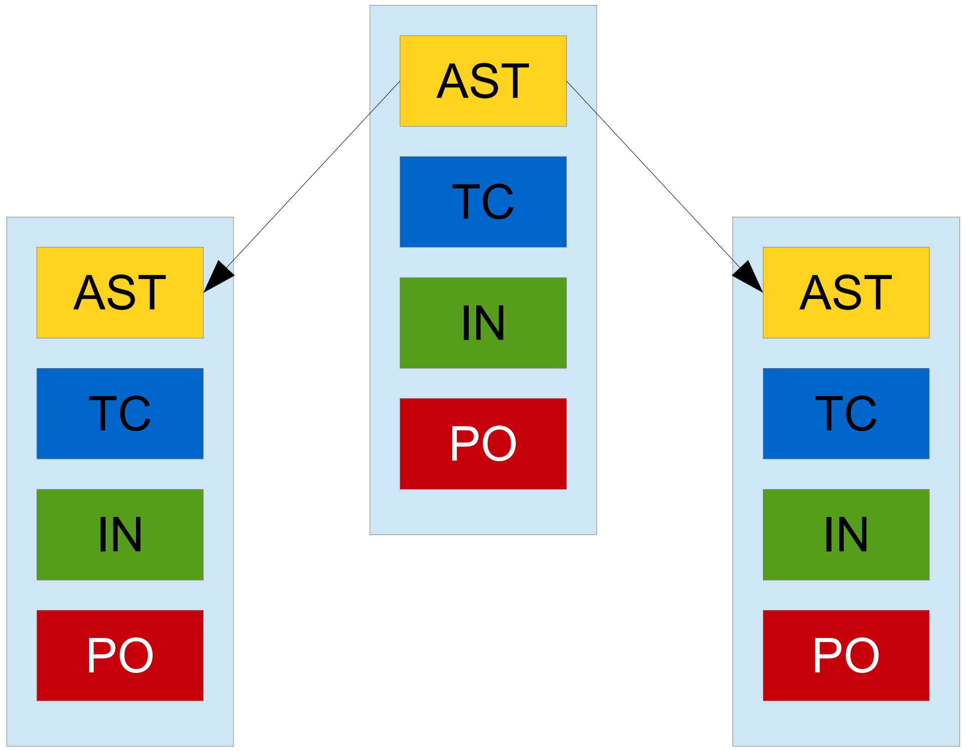
(Internal changes in VDMJ v4)

Nick Battle, Fujitsu UK

VDMJ version 3

1 + a

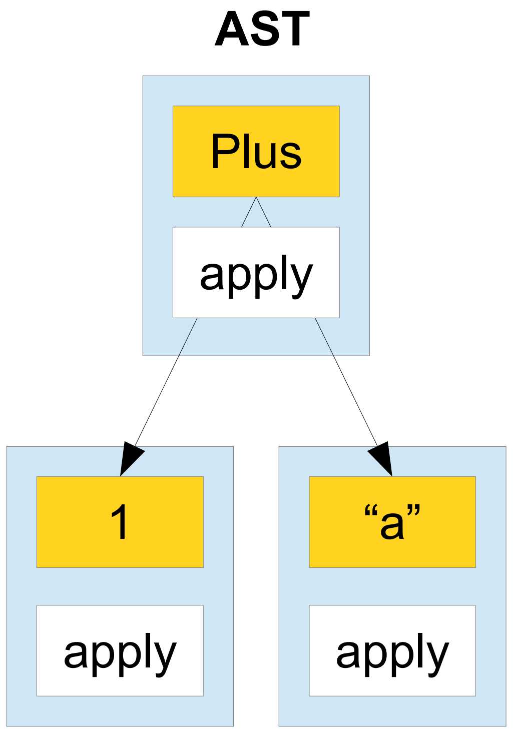
PlusExpression



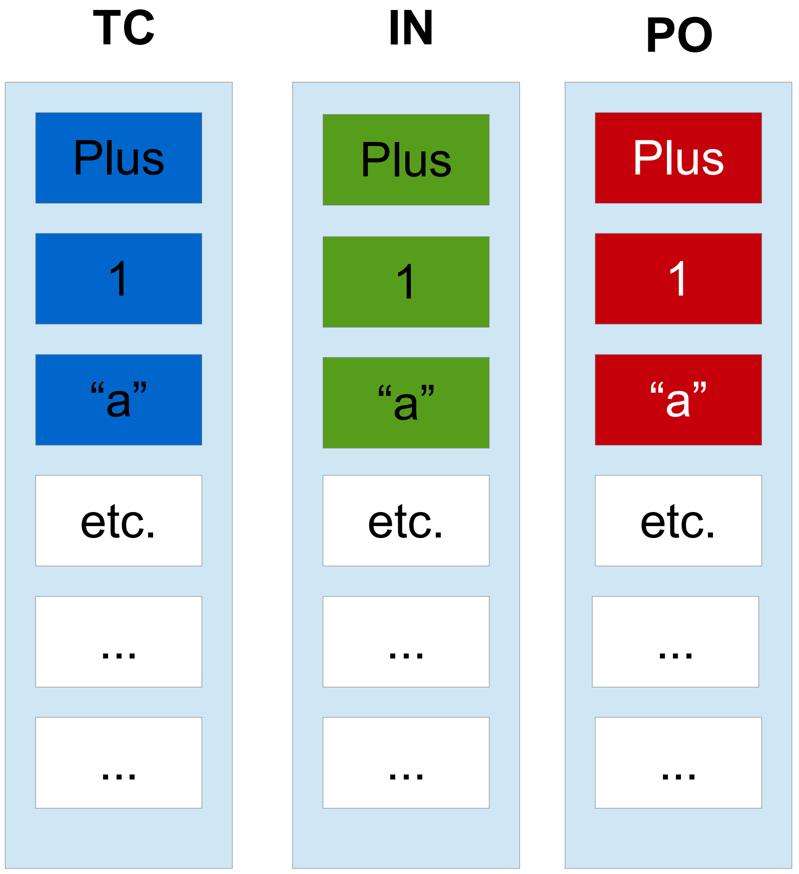
IntegerLiteralExpression **VariableExpression**

Overture2 with Visitors

1 + a



Analysis Visitors

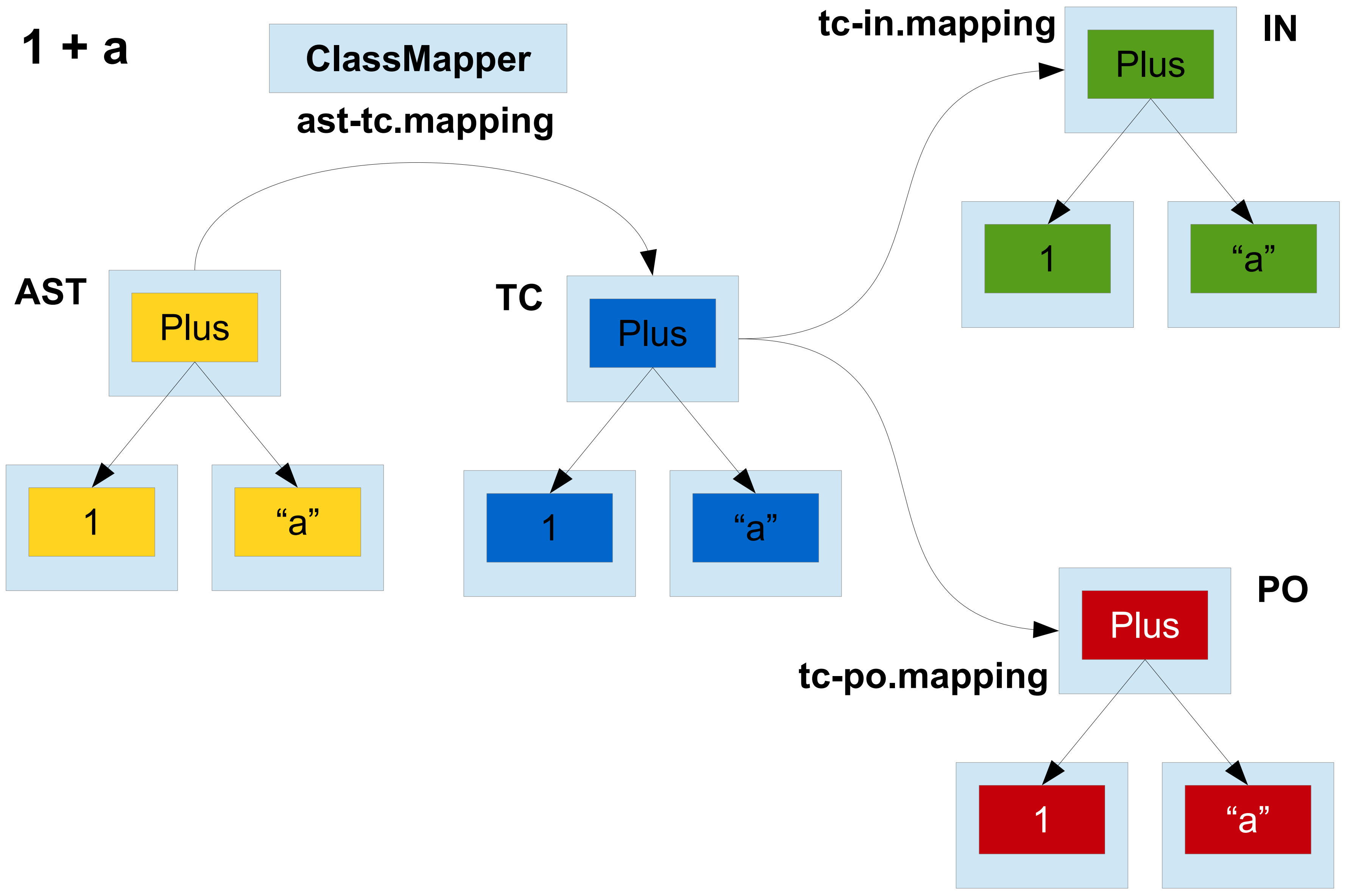


etc...

What could possibly go wrong?

- The Visitor pattern has problems with very rich ASTs:
  - VDM AST has ~300 types of node
  - Some visitor classes can get *very* large (so split)
  - Many small visitors needed too – over 120 of them
  - Flat namespace (sensible visitor names, but no structure)
- Common code is in separate *assistants* with factories
  - Many assistants – 66 of them
  - Flat namespace again
- There is nowhere obvious to store analysis working/output state
  - Type information added to AST – so implicit dependencies
  - Internal state information held in maps of node to state
- Analyses are slower (state map lookup, assistants, visitor calls)

VDMJ version 4



VDMJ version 4

```
##########################################################################################
# The class mapping definition for the Type Checker. See ClassMapper.
##########################################################################################

# expressions
package com.fujitsu.vdmj.ast.expressions to com.fujitsu.vdmj.tc.expressions;
map ASTPlusExpression{left, op, right} to TCPlusExpression(left, op, right);
map ASTIntegerLiteralExpression{value} to TCIntegerLiteralExpression(value);
map ASTVariableExpression{location, name} to TCVariableExpression(location, name);

# lex
package com.fujitsu.vdmj.ast.lex to com.fujitsu.vdmj.tc.lex;
map LexNameToken{} to TCNameToken(this);
unmapped com.fujitsu.vdmj.ast.lex.LexToken;
```

```java
public class ASTPlusExpression extends ASTNumericBinaryExpression
{
	public ASTPlusExpression(ASTExpression left, LexToken op, ASTExpression right)
	{
		...

public class TCPlusExpression extends TCNumericBinaryExpression
{
	public TCPlusExpression(TCExpression left, LexToken op, TCExpression right)
	{
		...

public class TCNameToken extends TCToken implements Comparable<TCNameToken>
{
	public TCNameToken(LexNameToken name)
	{
		...
```

How does this help?

- Analysis classes are *very* small (even smaller than VDMJ v3)
- Common code is in a natural class hierarchy
- Analysis state lives within its analysis tree
- Analysis dependencies are explicit (via mappings)
- Analyses are faster (same as VDMJ v3, no assistants, state lookup, etc.)
- Parser is 20-30% faster than VDMJ v3 (fewer fields to initialize)
- Code size roughly the same (4x classes, using same code)
- Some old problems solved: *LexNameToken* and *TCNameToken*

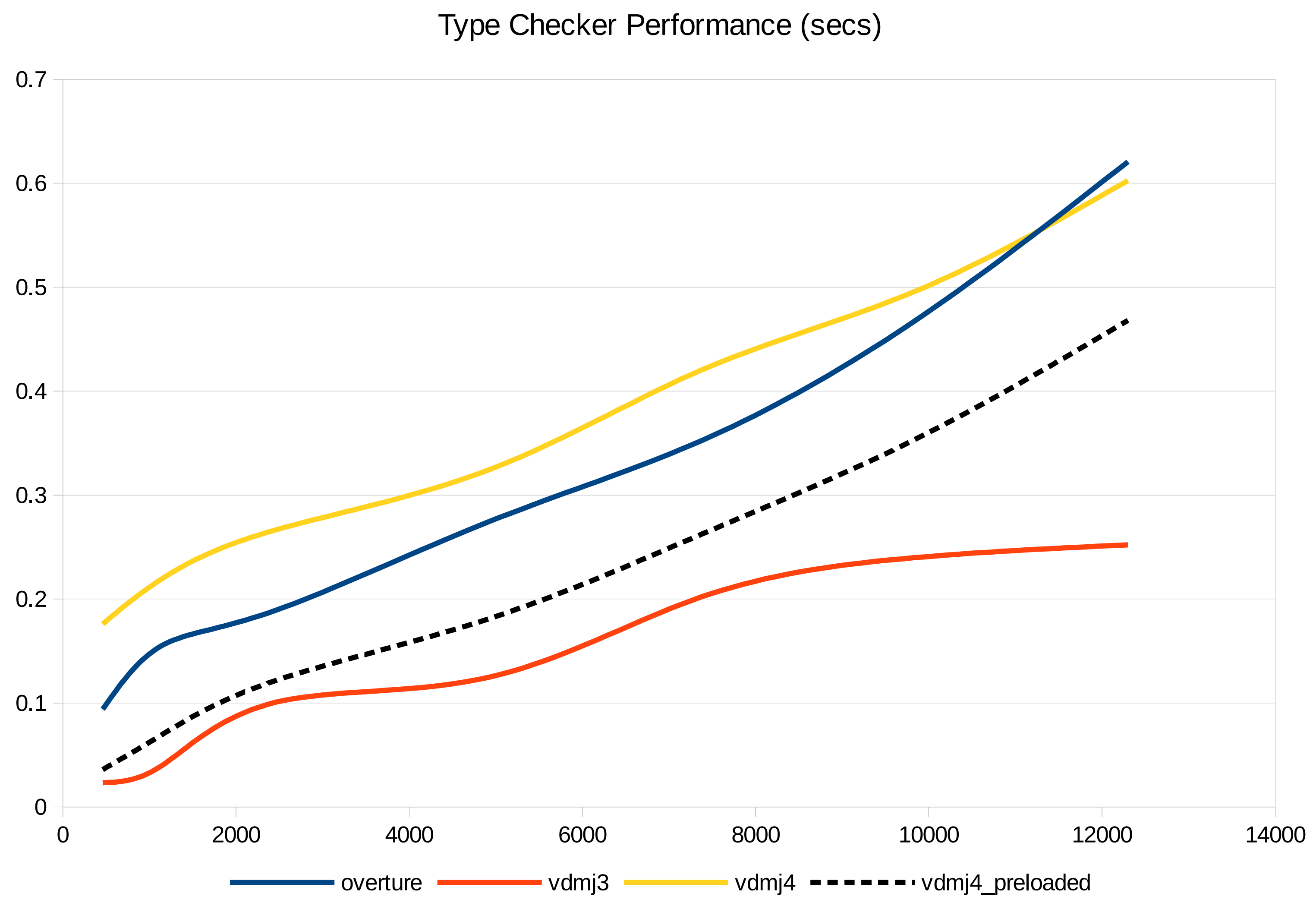
But…

- *It's an unproven non-standard technique (risks unclear)*
- *Small recursive processes are not modular (cf. small visitors)*
- *Slightly more memory is occupied (a few Mb)*
- *And it critically depends on how fast Java can create new objects...*

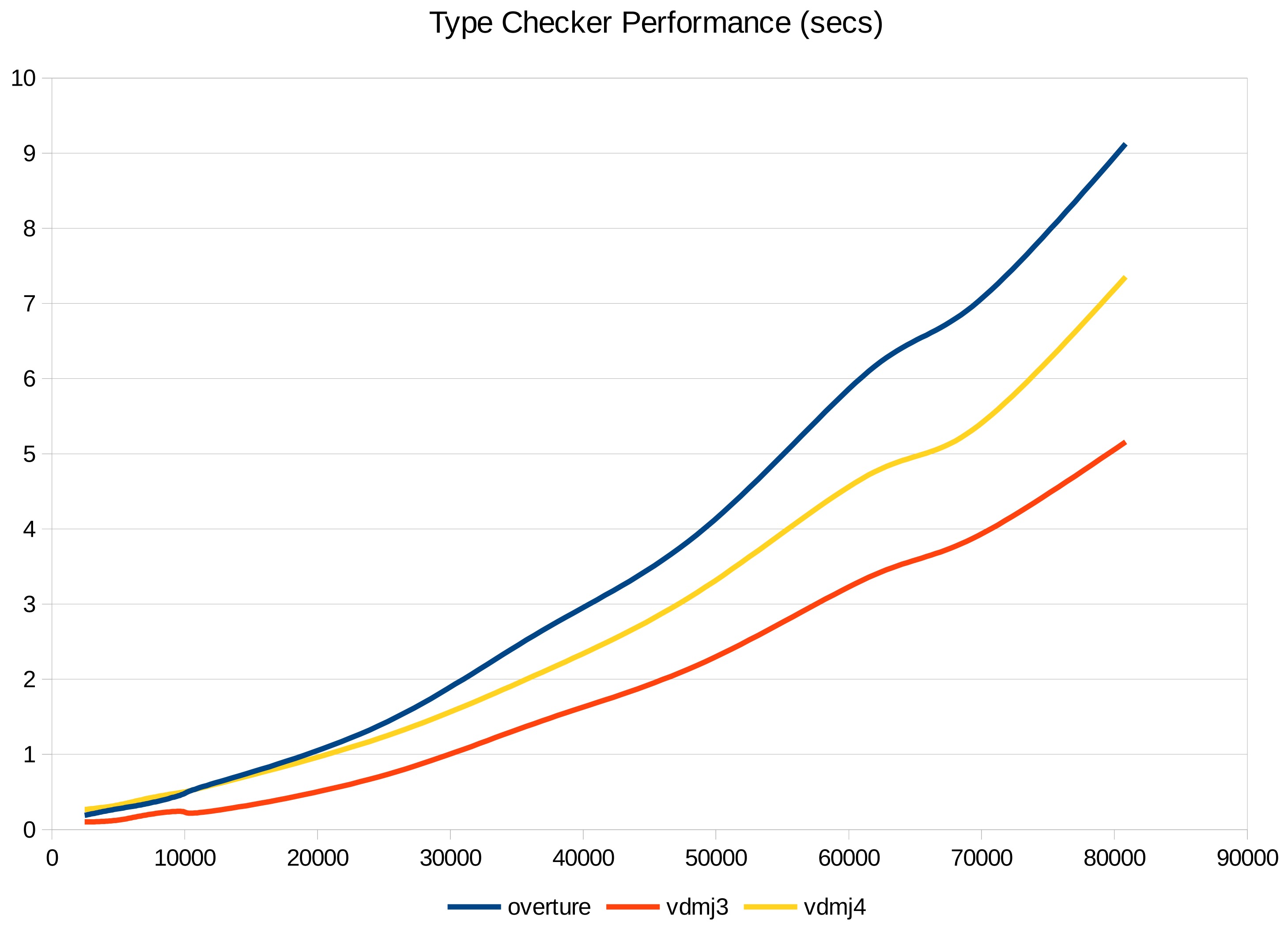
ClassMapper Performance

- Nodes mapped at 100-800K objects *per second*
  - 500K AST nodes roughly equivalent to 100,000 line spec
  - Conversion only happens once per analysis type
  - Delay is “between” analyses, not during analyses
- Mappings file loads in < 0.2 secs
  - Memory footprint of mappings is a few hundred Kb
  - All mappings loaded once (at startup?)
- Extra memory for trees is mostly extra linkage (cf. VDMJ v3)
  - Typically a few Mb, even for large specifications
  - “Copies” of state are just shared object references
  - Single-use trees can be removed (eg. AST or PO)

ClassMapper Performance



ClassMapper Performance



Where Next?

So performance may not be a big problem, but…

- Visitors can be better for small processes – use both?
- Overture's problems may not be due to its visitors
- We should check other dialects' mapping performance
- Mapping file/new analysis creation needs tool support
  - How often does a mapping need to change?
  - Implement a new analysis from scratch
- What if an analysis is derived from two or more trees?
- A plugin architecture should be investigated.

FUJITSU

shaping tomorrow with you