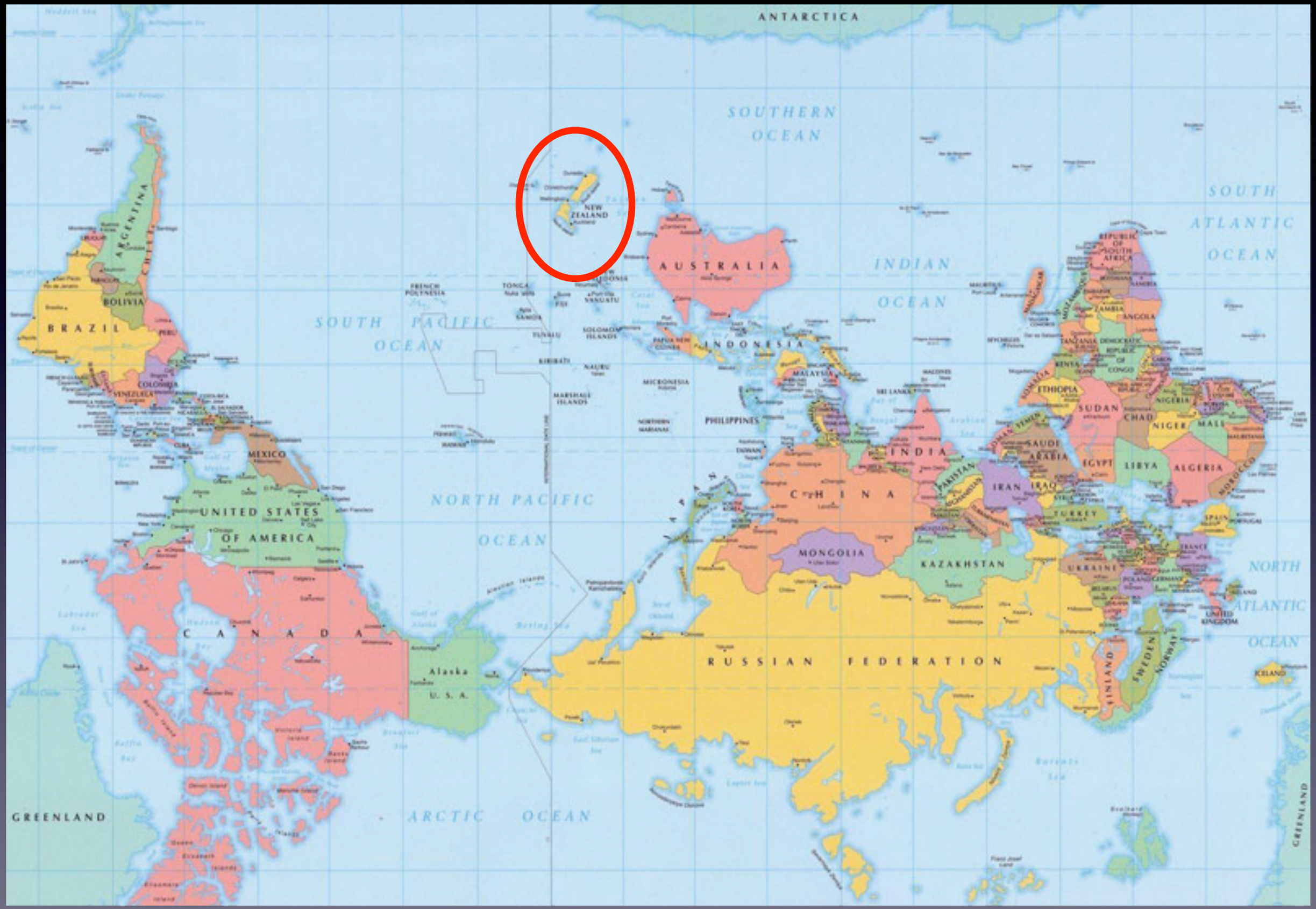
Subsuming Methods: Finding New Optimisation Opportunities in OO Software

David Maplesden
John Hosking
Ewan Tempero
John Grundy

The University of Auckland
ICPE 2015

Performance is Important

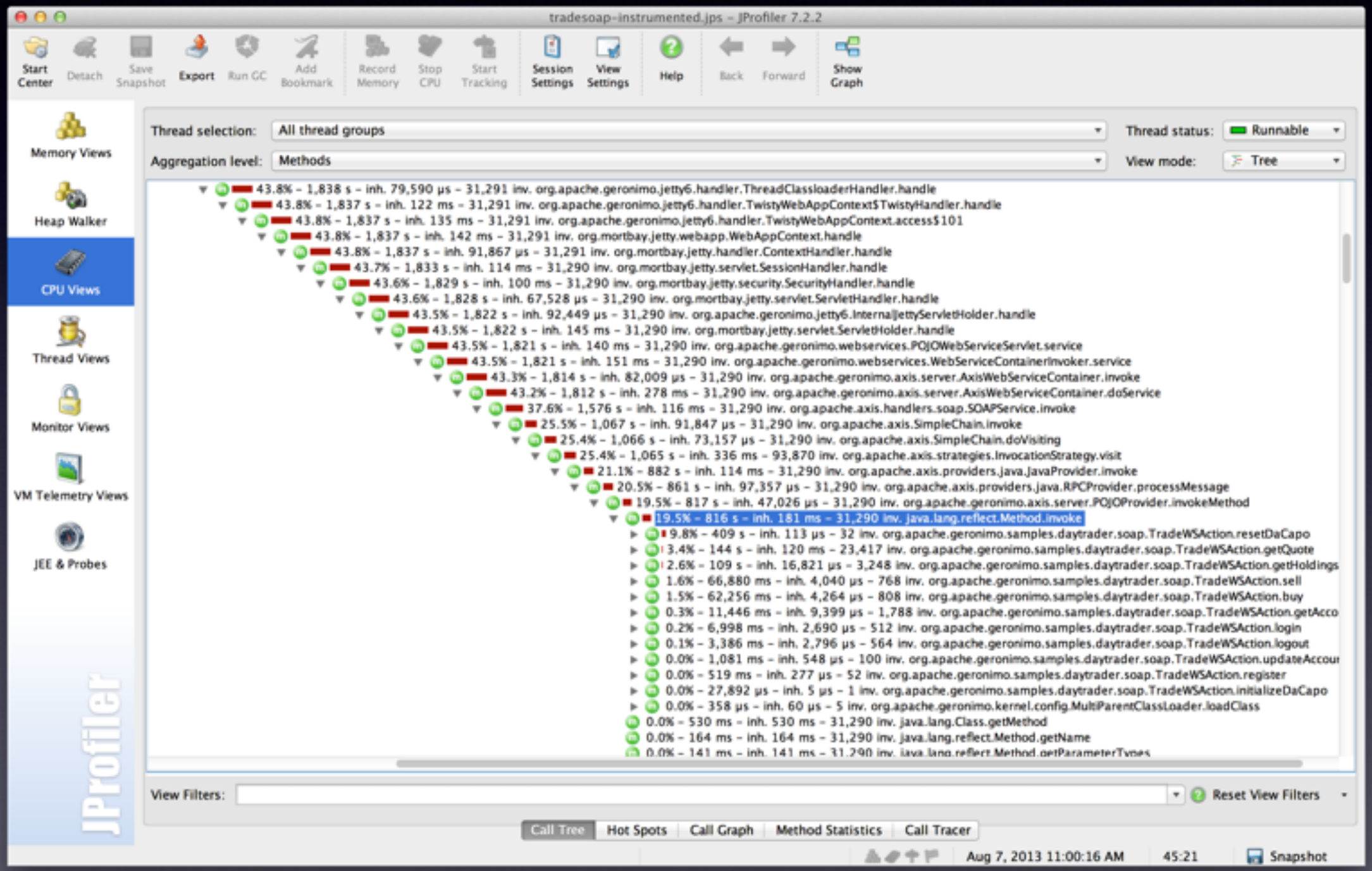
- Cloud computing costs
- Resource constrained environments
    - Mobile applications
- Lost business - Amazon 100ms delay = 1% sales
- Failed projects


ANTARCTICA
SOUTHERN OCEAN
SOUTH ATLANTIC OCEAN
INDIAN OCEAN
SOUTH PACIFIC OCEAN
NORTH PACIFIC OCEAN
NORTH ATLANTIC OCEAN
ARCTIC OCEAN
NEW ZEALAND
AUSTRALIA
INDONESIA
PHILIPPINES
FRENCH POLYNESIA
TONGA
FIJI
VANUATU
SOLOMON ISLANDS
PAPUA NEW GUINEA
KIRIBATI
NAURU
MICRONESIA
MARSHALL ISLANDS
NORTHERN MARIANAS
MALAYSIA
SRI LANKA
MALDIVES
INDIA
CHINA
MONGOLIA
KAZAKHSTAN
RUSSIAN FEDERATION
PAKISTAN
AFGHANISTAN
IRAN
IRAQ
SAUDI ARABIA
YEMEN
OMAN
TURKEY
SYRIA
EGYPT
LIBYA
ALGERIA
MOROCCO
SPAIN
PORTUGAL
FRANCE
UKRAINE
POLAND
GERMANY
UNITED KINGDOM
FINLAND
SWEDEN
NORWAY
ICELAND
GREENLAND
REPUBLIC OF SOUTH AFRICA
ZAMBIA
ANGOLA
TANZANIA
DEMOCRATIC REPUBLIC OF CONGO
KENYA
ETHIOPIA
SUDAN
CHAD
NIGERIA
NIGER
MALI
MAURITANIA
MADAGASCAR
MOZAMBIQUE
ARGENTINA
CHILE
BOLIVIA
BRAZIL
PERU
ECUADOR
COLOMBIA
VENEZUELA
GUYANA
SURINAME
PARAGUAY
URUGUAY
COSTA RICA
EL SALVADOR
NICARAGUA
CUBA
MEXICO
UNITED STATES OF AMERICA
CANADA
Alaska
U. S. A.

Focus

- Large-scale object-oriented software
  - Ubiquitous in industry
- Late-cycle empirical performance analysis
  - a.k.a. profiling and tuning
  - Complementary to model-based predictive methods
- *Analysis* - the neglected backward path

A ‘modern’ profiler



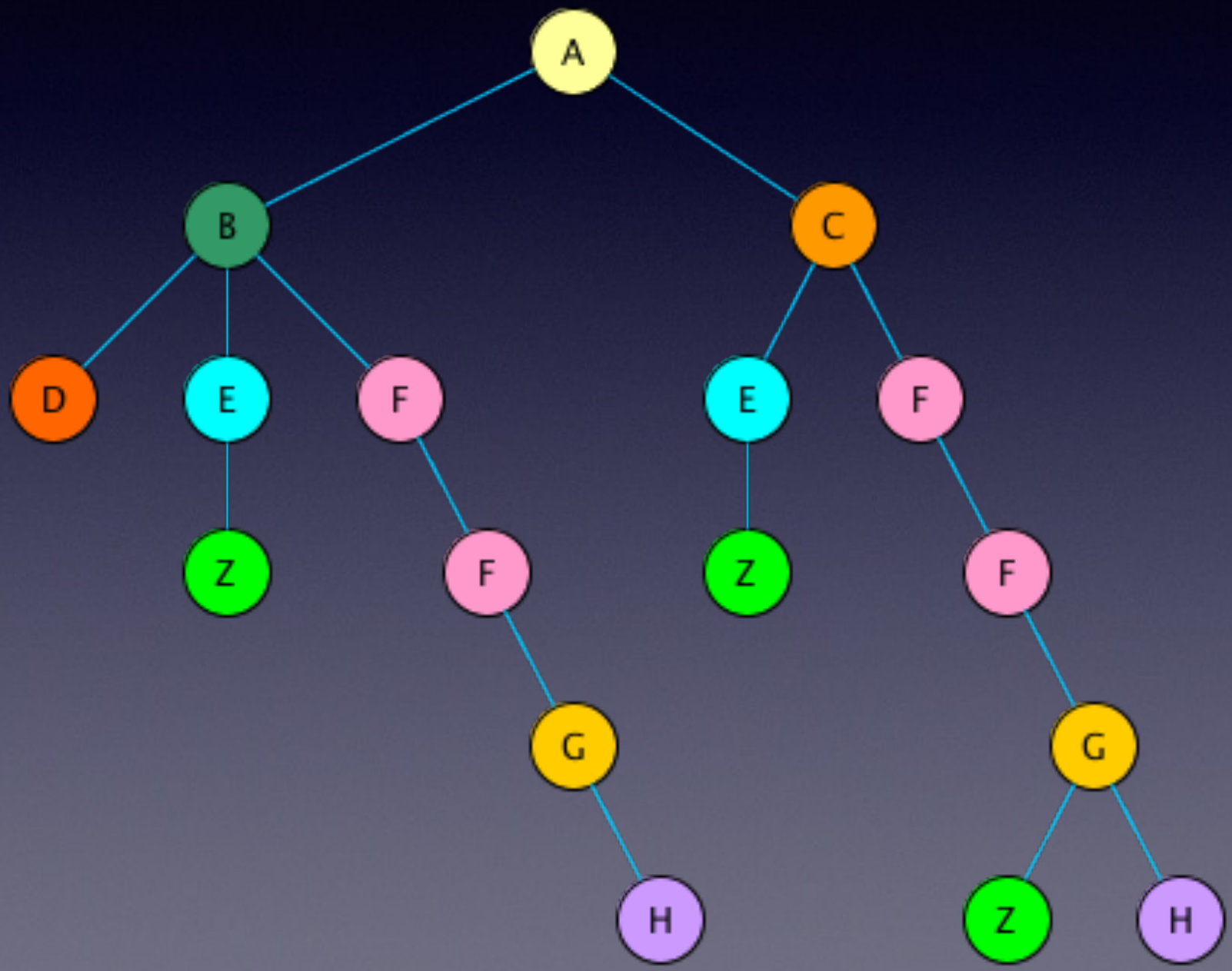
Challenges of OO software

- Numerous small methods
- Heavily layered architecture
    - Engineered for maintainability and reuse
    - Reusable frameworks, more abstractions
    - *Runtime bloat*
- Complex, thinly distributed, runtime behaviour
- Challenging to identify optimisation opportunities

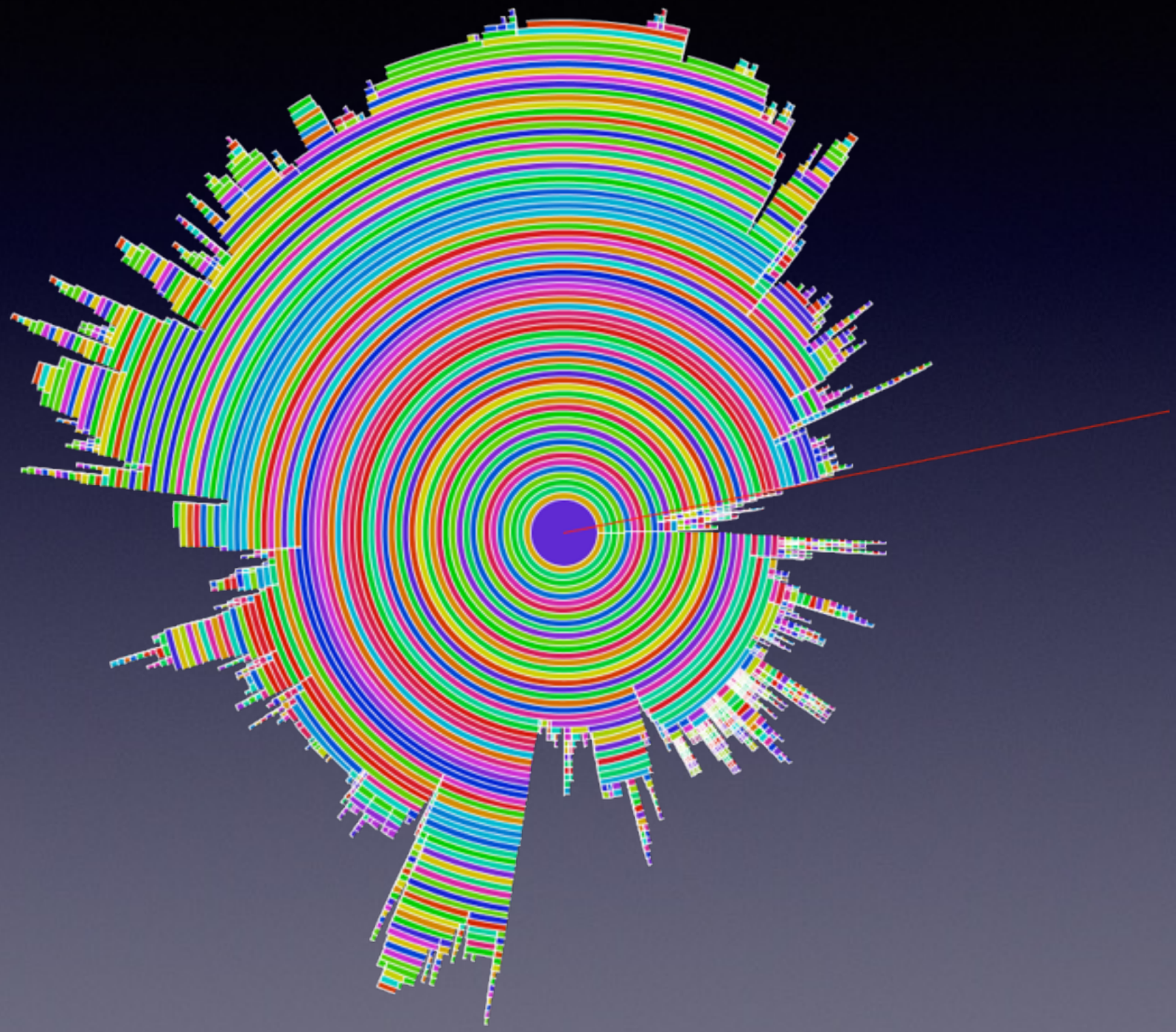
Hot methods - DaCapo fop benchmark

Method	Occurrences in CCT	% Exclusive Time
sun.misc.FloatingDecimal.dtoa	348	6.876
java.text.DigitList.set	374	5.245
java.text.DecimalFormat.subformat	374	3.110
org.apache.fop.fo.properties.PropertyMaker.findProperty	1501	2.461
java.lang.String.equals	4666	1.853
sun.nio.cs.US_ASCII$Encoder.encode	568	1.788
sun.misc.FloatingDecimal.countBits	348	1.556
java.util.HashMap.hash	10663	1.506
java.util.HashMap.getEntry	6081	1.342
java.lang.String.indexOf	3343	1.295

Background - Calling Context Trees



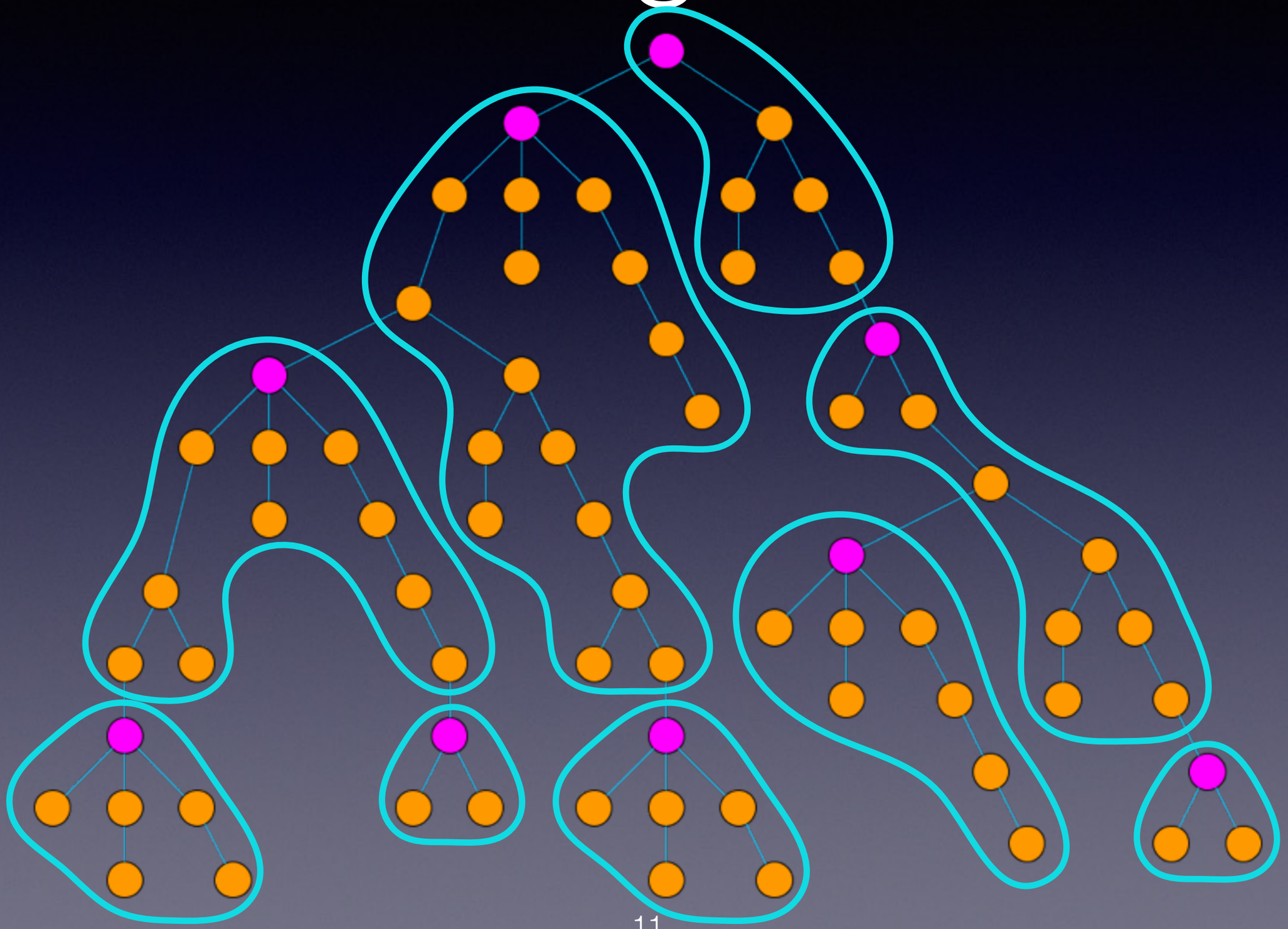
Calling Context Ring Chart



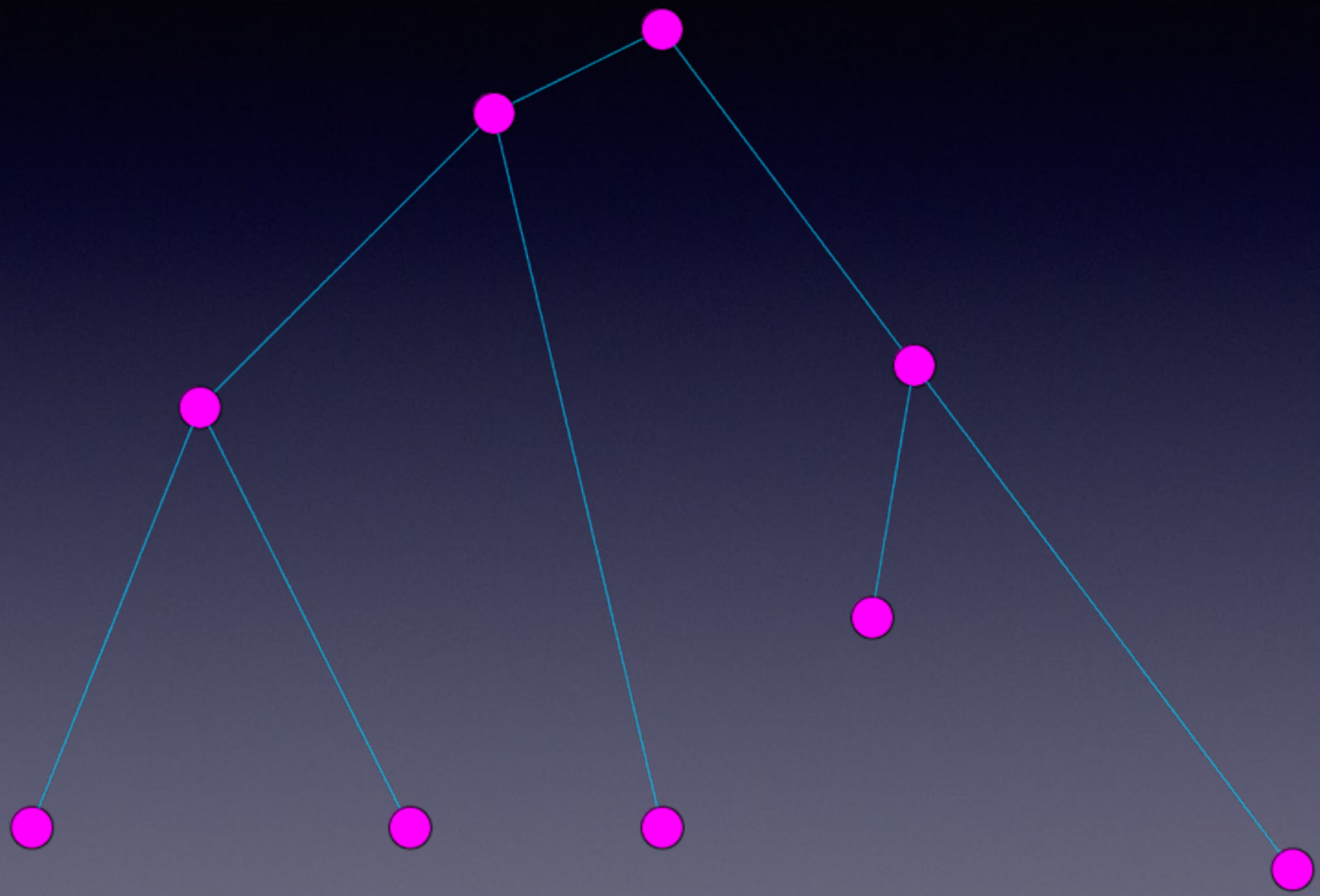
Key Idea

- CCTs aren’t just random data
- There are patterns within the calling context tree
  - induced by the design of the software
  - compact
  - repeated in multiple locations
  - expensive when aggregated

Subsuming Methods

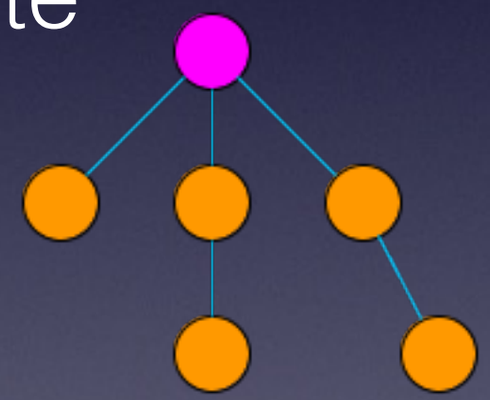


Consolidated Tree



Subsuming Methods

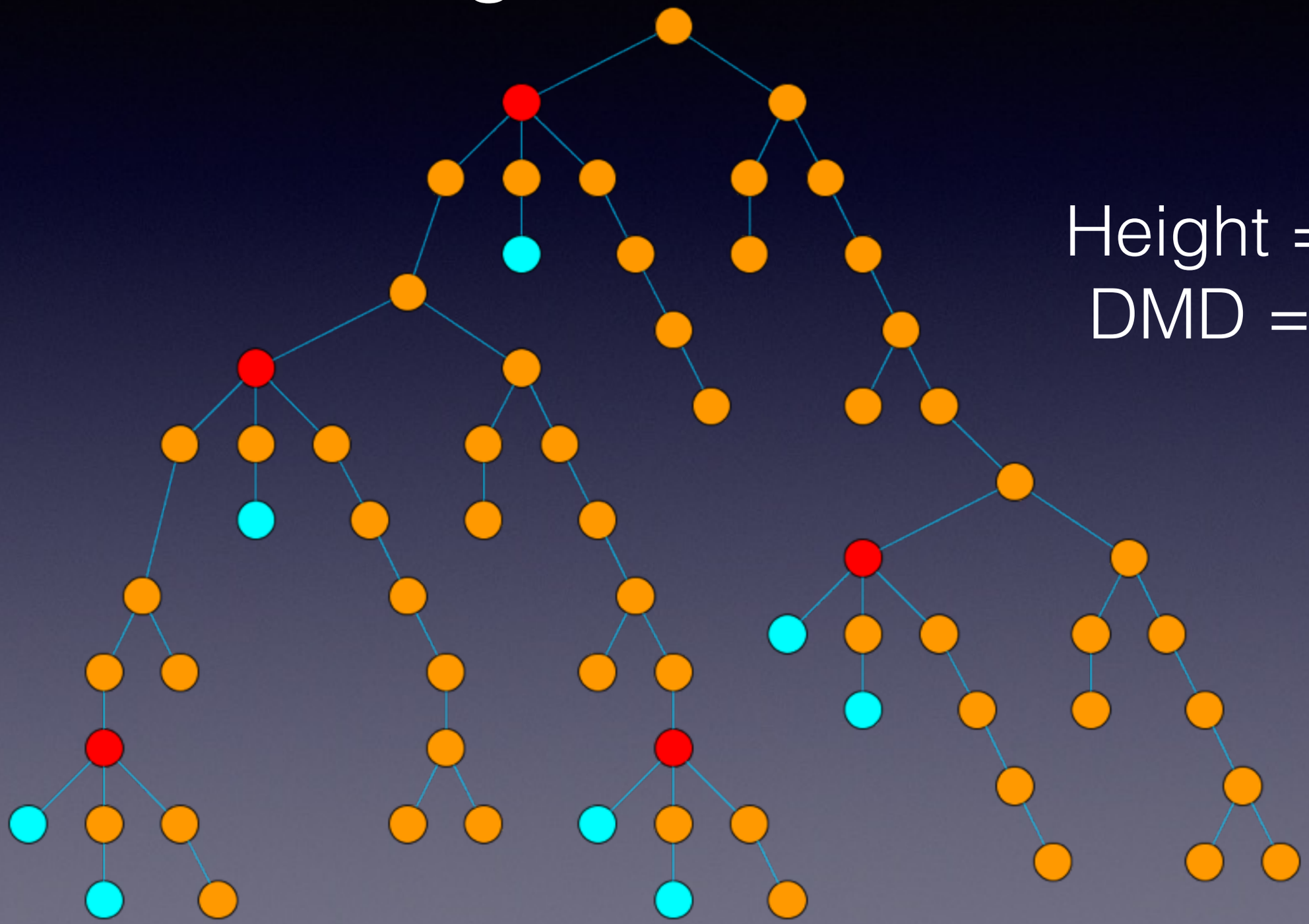
- Partition the CCT into areas of related functionality
- Induced time is very efficient to calculate
- Each subsuming method represents a repeated pattern
- How do we choose our subsuming methods?



Subsuming Attributes

- ‘Elementary’ methods - induce a limited range of behaviour
  - Approximated using height of method in CCT
  - Trivial case (height = 0) - makes no method calls - a leaf method
  - getters, setters, hashCode(), equals()
  - ~30% of all methods are leafs ~70% have height <= 4
- ‘Subordinate’ methods - called in a predictable manner
  - Every call to the method can be attributed to a (nearby) dominating method which is responsible for the invocation
  - Measured using novel metric - dominating method distance
  - Trivial case is when a method is only ever called from a single call site
  - ~70% of all methods have a single parent ~77% have DMD <= 4

Height and Dominating Method Distance



Height = 0
DMD = 2

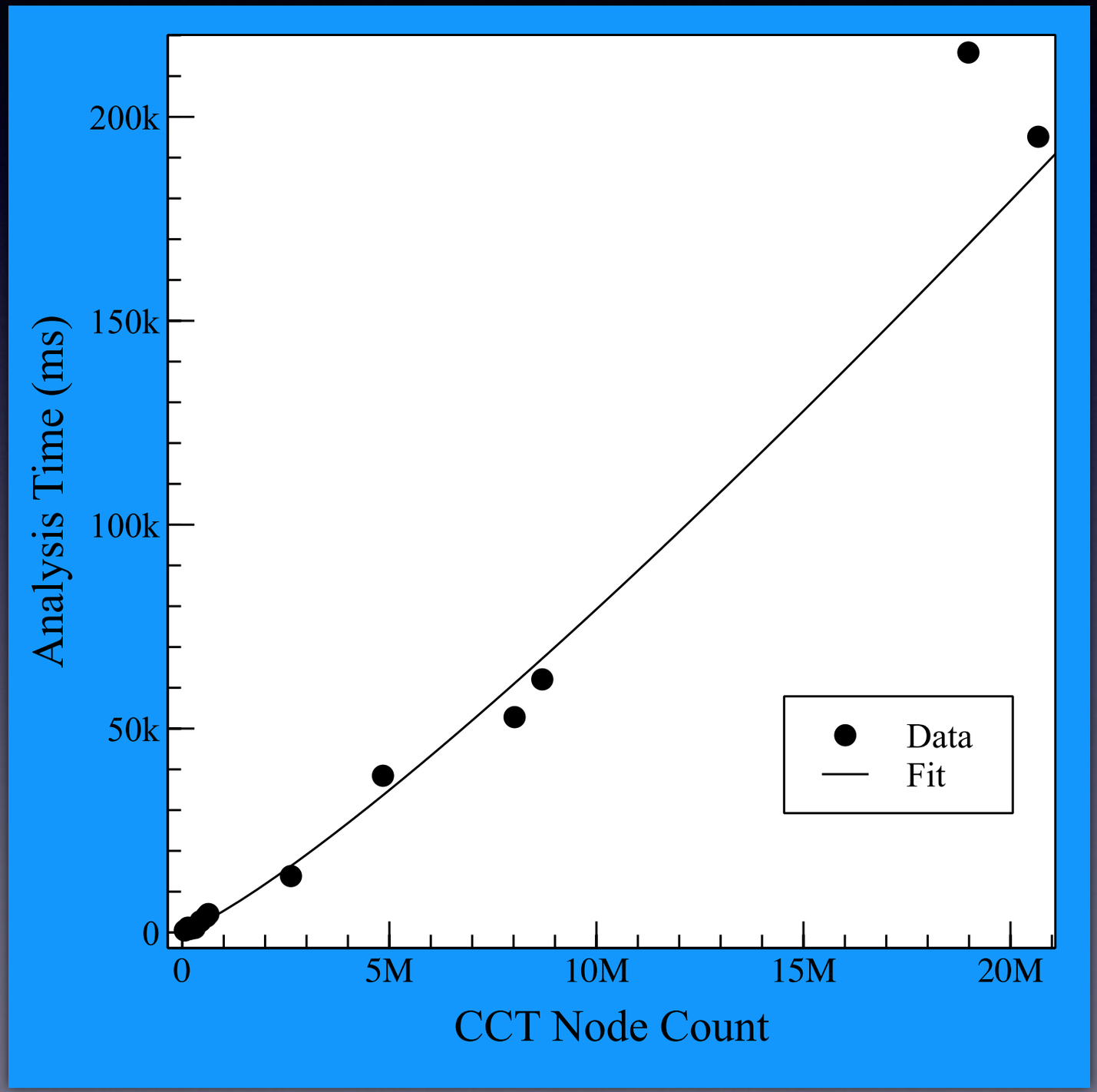
Experimental Evaluation

- DaCapo-9.12-bach (2009) benchmark suite
  - 14 different Java benchmark applications
- JP2 profiler to capture CCT profiles
  - Very consistent, reproducible results
- Apply our subsuming methods analysis
- 5 runs for each benchmark
- Constraints: height > 4 and DMD > 4

Results Summary

- Across the 14 benchmarks:
  - 6.12% of all methods were subsuming
  - 11.82% of nodes in the CCT were subsuming
    - => subsuming methods aggregation greatly simplifies profile information
  - 15 of the top 20 subsuming methods were not in the top 20 inclusive or exclusive cost methods
    - => new optimisation opportunities are identified
- https://www.cs.auckland.ac.nz/~dmap001/subsuming/

Results - Analysis Efficiency



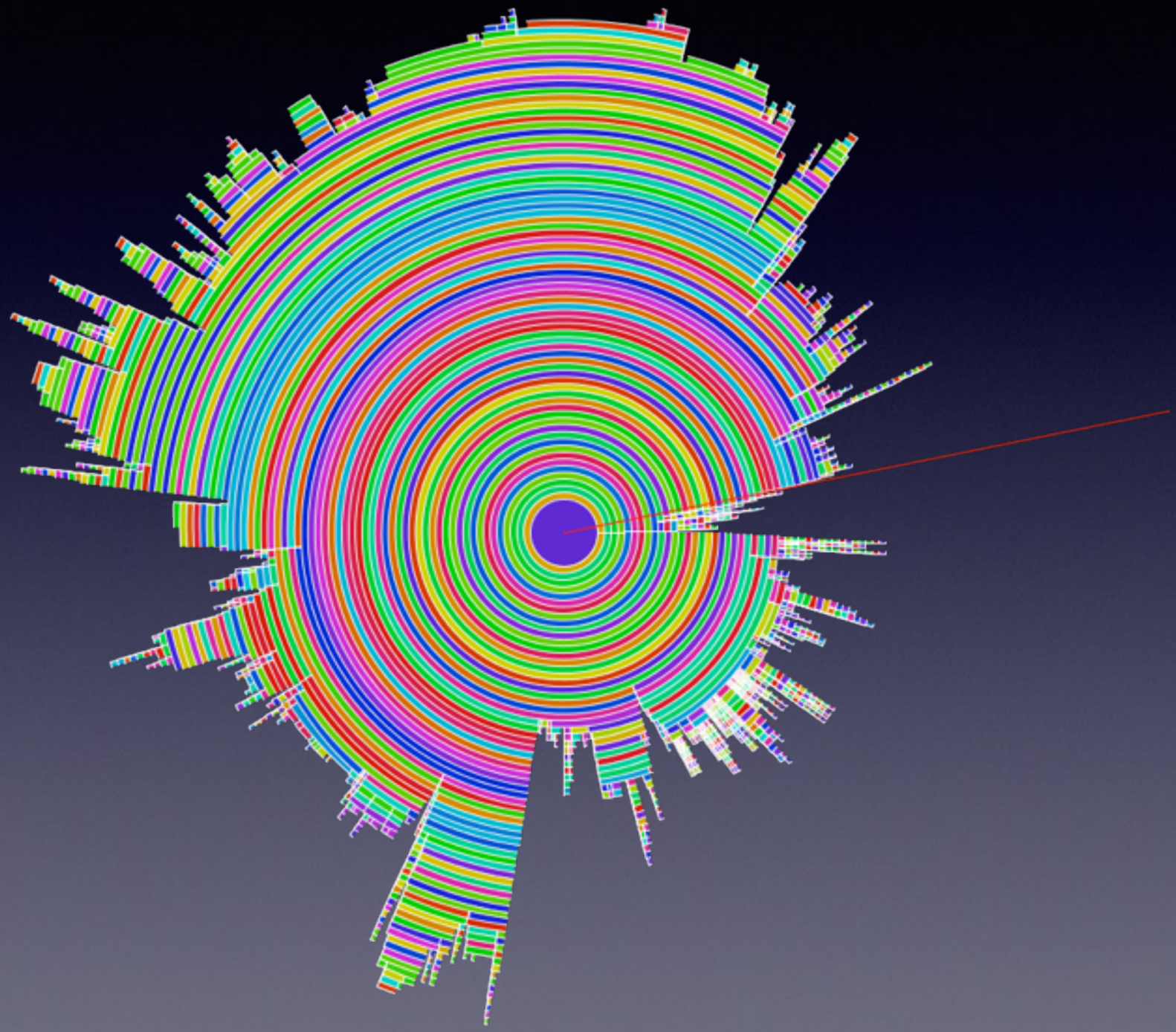
Results - DaCapo fop benchmark

	Full CCT	Subsuming CCT	Ratio
Nodes	628751	71430	11.36%
Height	111	25	22.52%
Unique Methods	6709	345	5.14%

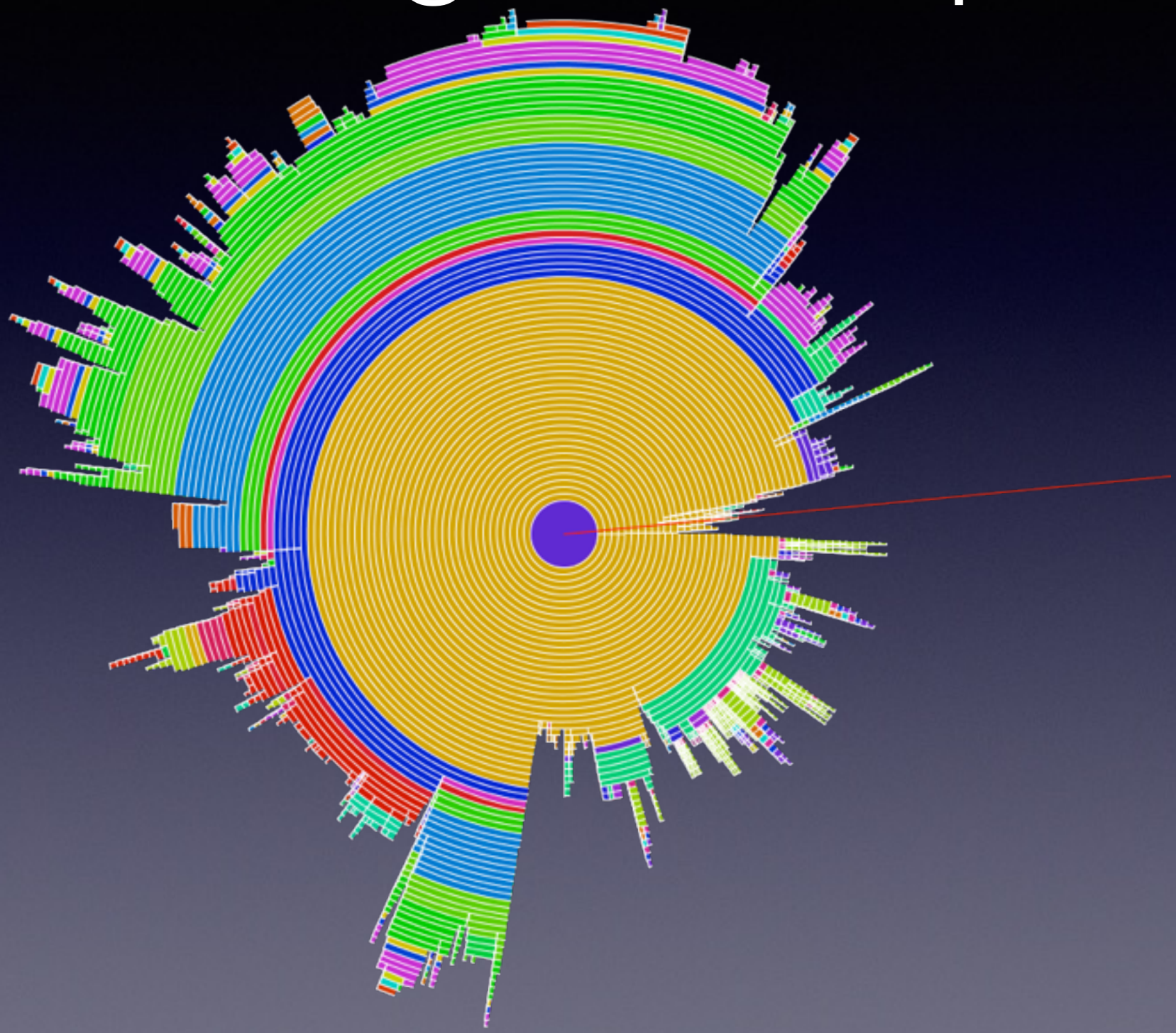
Top Subsuming Methods - DaCapo fop benchmark

Method	Occurrences in CCT	% Exclusive Time	% Induced Time
java.text.DecimalFormat.format	374	0.169	13.644
org.apache.fop.fo.StaticPropertyList.get	1691	1.228	8.884
sun.misc.FloatingDecimal.dtoa	348	6.876	8.871
org.apache.fop.layoutmgr.BlockStackingLayoutManager.getNextKnuthElements	12	0.068	6.381
org.apache.fop.render.AbstractRenderer.renderInlineArea	42	0.041	4.449
org.apache.fop.layoutmgr.BreakingAlgorithm.findBreakingPoints	16	0.002	4.340

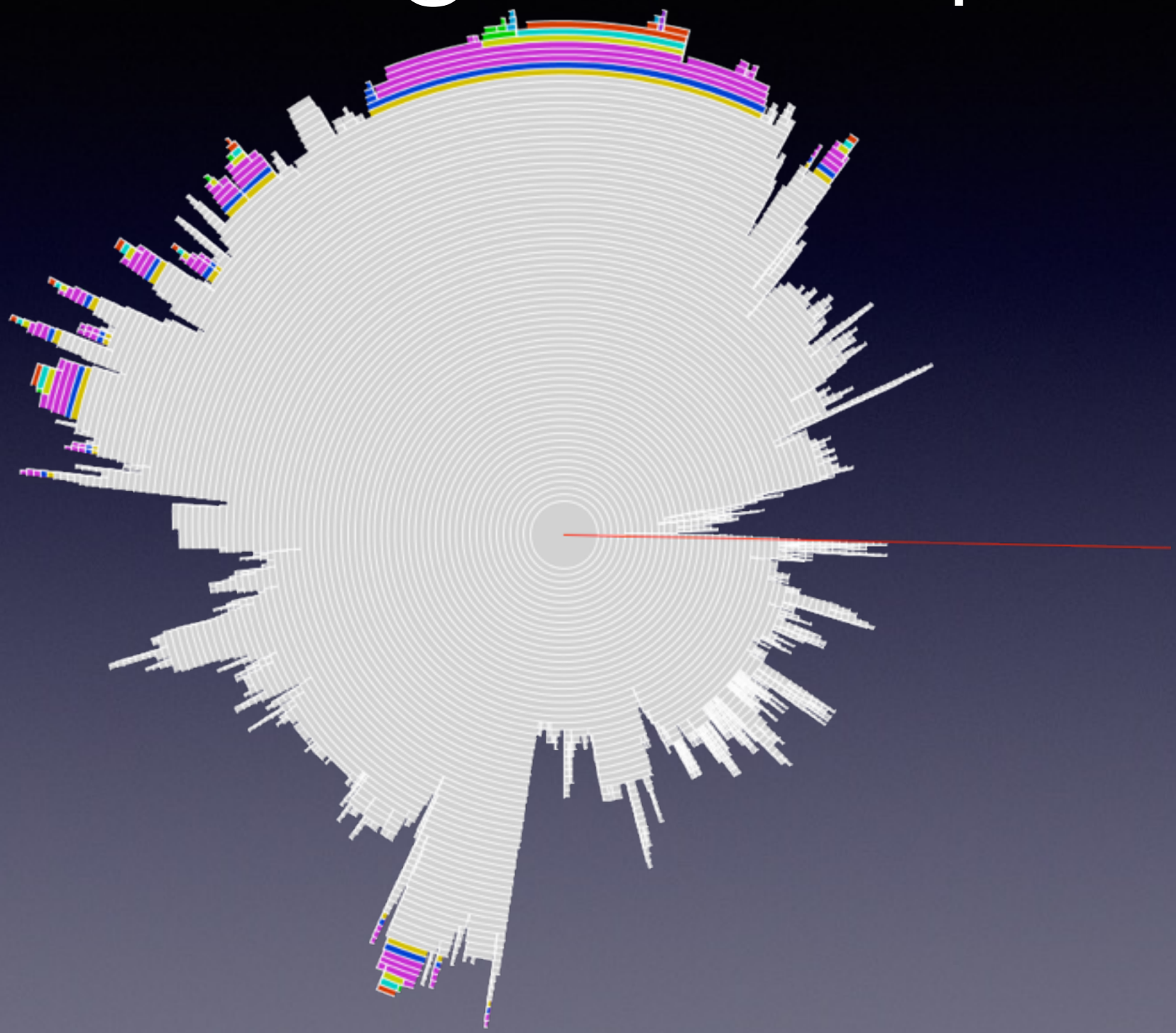
Calling Context Ring Chart



Subsuming - DaCapo fop



Subsuming - DaCapo fop



DaCapo fop - Improvements

- Top subsuming method - java.text.DecimalFormat.format
- Highly complex general purpose number formatter
- Called 98% of the time from one location to produce a very specific (and simple) 2 decimal place format
  - org.apache.xmlgraphics.ps.PSGenerator.formatDouble
- Accounts for 26% of the total benchmark cost
- Replace with a more specialised implementation
  - 22% reduction in benchmark cost

Summary

- Subsuming Methods
  - Empirical performance analysis approach
    - Helps identify repeated patterns within a CCT profile
    - Efficient offline analysis
    - Complementary to existing approaches
    - Applicable to a wide range of data
- Preliminary evaluation with DaCapo benchmark

Industry Case Study

- letterboxd.com
  - 125,700 registered members
  - 3.6 million requests per day
  - 54.8% reduction in CPU load
  - 49.6% reduction in response time
- Paper accepted at ICSE 2015 - SEIP track
  - “Performance Analysis using Subsuming Methods: An Industrial Case Study” - Maplesden et al

User Study

- Test the effectiveness of subsuming methods analysis in aiding software engineers
- Implemented as an on-line test and questionnaire
- Mid 2015
- If interested please volunteer!
  - Contact: david@maplesden.co.nz

Thank you!

Questions?

Related Work

- Very broad domain (100 venues in our SLR)
- Relevant work from HPC, Compiler, Programming Language domains
- Majority of approaches provide simple metrics
  - Lack of actionable feedback
- Very few approaches leverage static analysis
- Runtime bloat research focussed on data-flow

Systematic Mapping

- “Performance Analysis for Object-Oriented Software: A Systematic Mapping”
- Empirical methods focus
- Accepted for publication in TSE
- http://dx.doi.org/10.1109/TSE.2015.2396514

Runtime Bloat Research

- Tackle problem of excessive activity to achieve seeming simple functionality
- Data-flow centric approaches
- Efficiency of data structures
- Object pooling opportunities
- Copy profiling
- Reference propagation profiling

Existing Approaches

- Lin et al (2010). Towards Anomaly Comprehension: Using Structural Compression to Navigate Profiling Call-Trees.
  - Aggregation by package and class name
- Srinivas & Srinivasan (2005). Summarizing application performance from a components perspective.
  - Thresholding and filtering