Tracking the Flow of Ideas through the Programming Languages Literature

Michael Greenberg, Kathleen Fisher, and David Walker

How can we understand the PL literature?

Is there more **related work** should I cite?

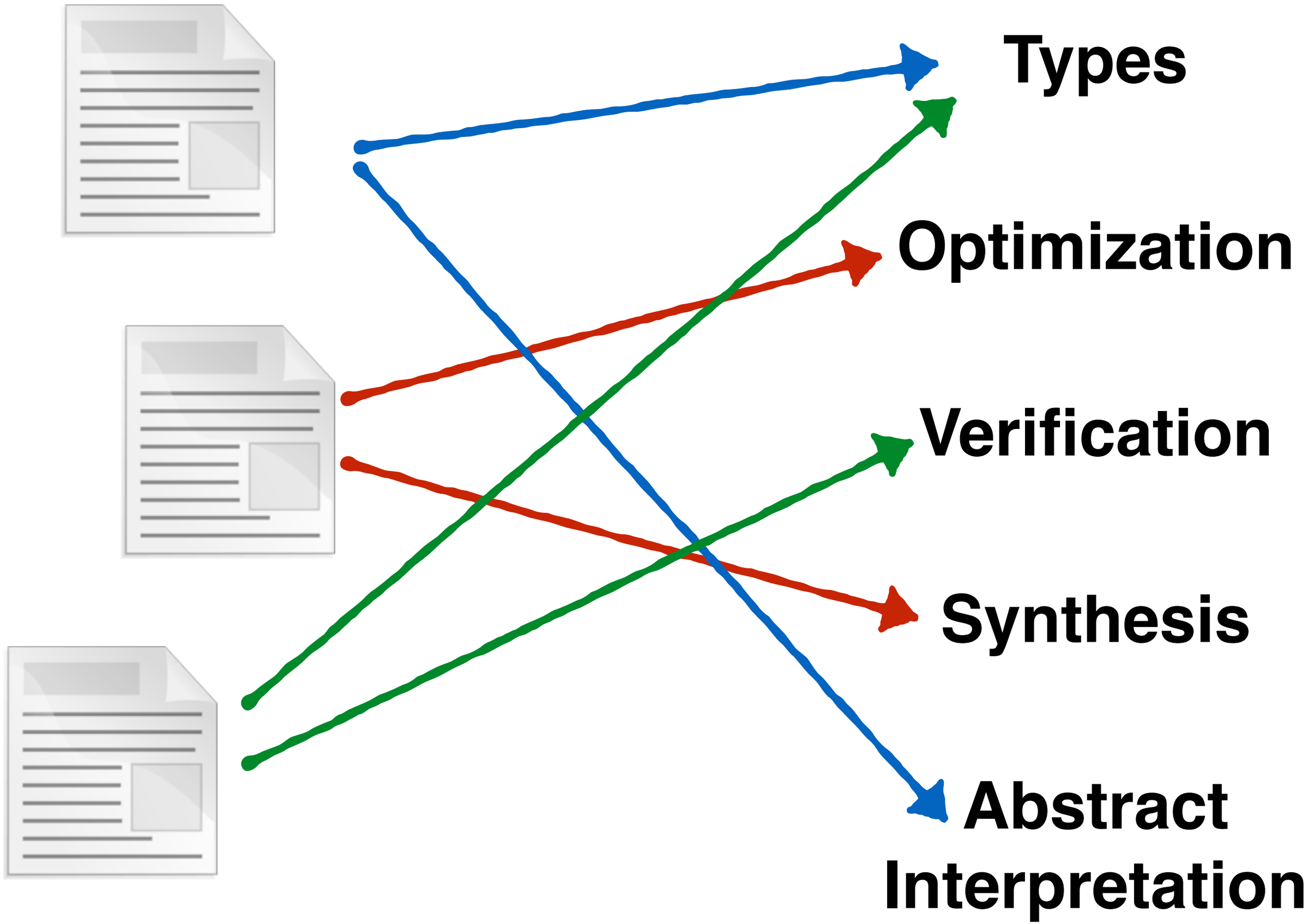
Is my work a better fit for **PLDI** or **POPL**?

Who should I invite to this **PC**?

Who should **review** this paper?

Was this a **typical year** for ICFP?

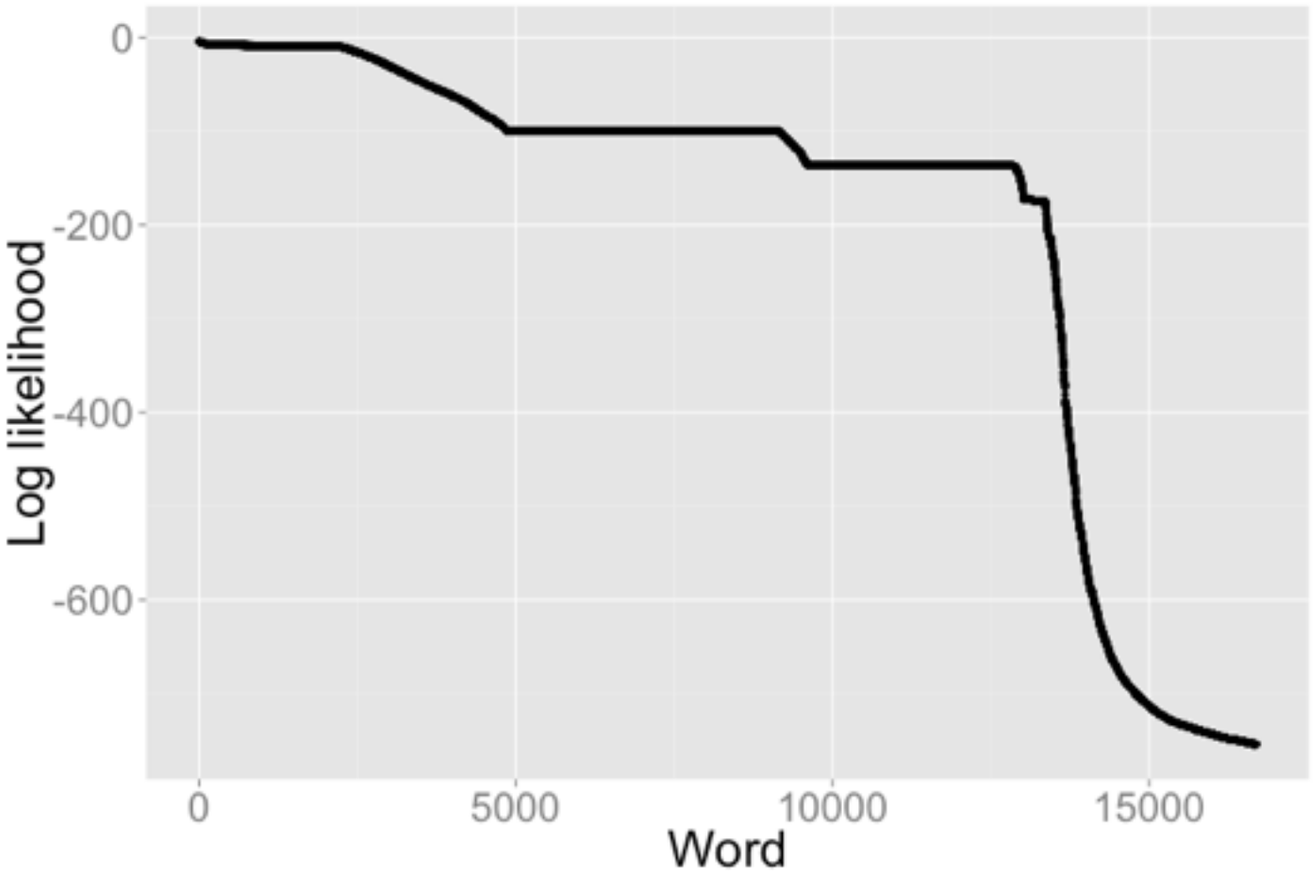
How has **OOPSLA** changed over the years?


Types
Optimization
Verification
Synthesis
Abstract
Interpretation

What is a ‘topic’ in a document?

Word	Count
type	120
system	83
check	34
static	21

Topics are distributions of words



“Parsing” topic	
Word	**Log likelihood**
grammar	-3.905040
language	-4.206531
structure	-4.308618
parser	-4.513348
…	…

Documents are a mix of topics

.6

type systems	
Word	**Count**
type	120
system	83
check	34
static	21

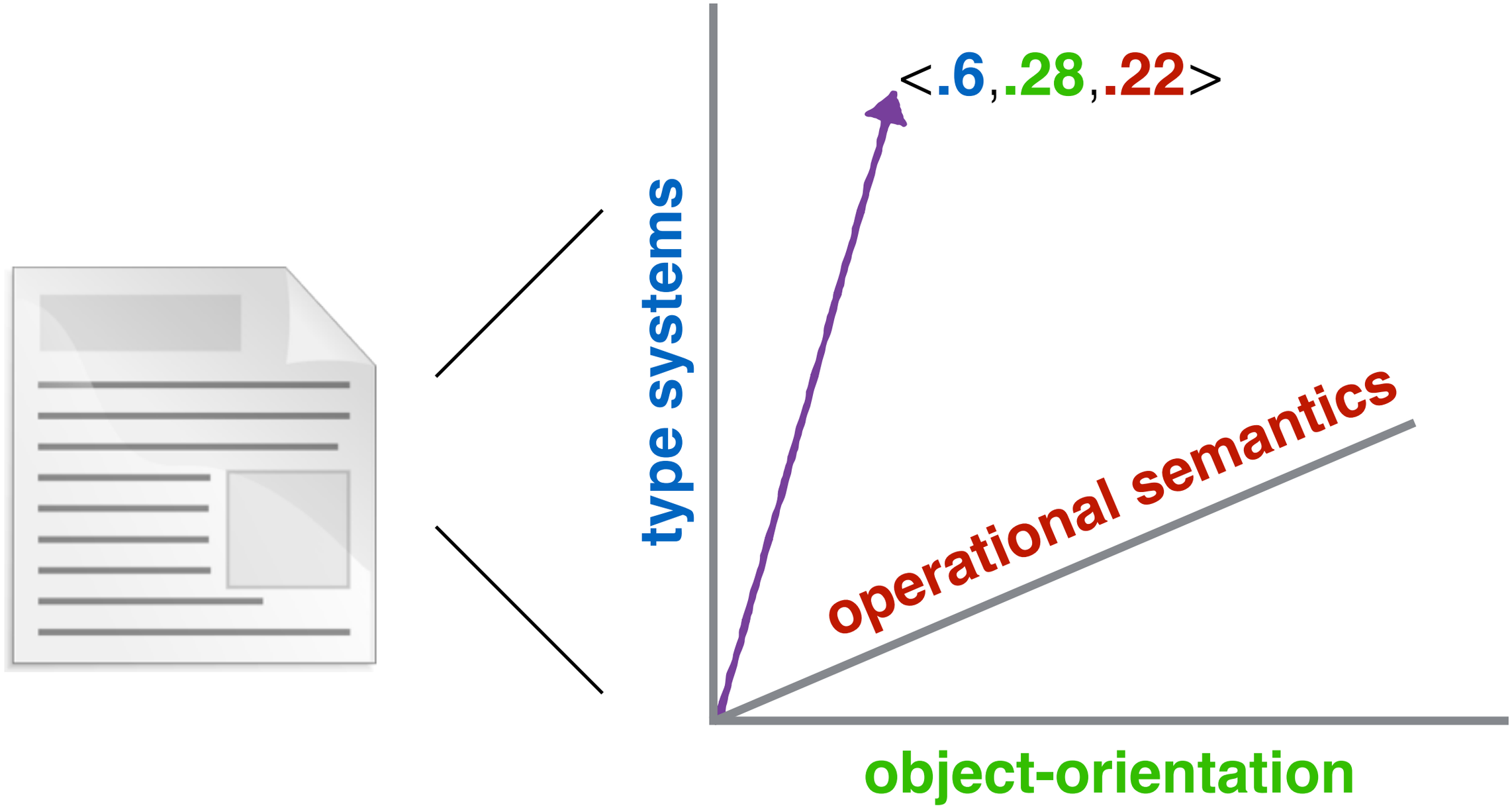
.28

operational semantics	
Word	**Count**
semantics	90
step	45
reduce	38
evaluate	19

.22

object-orientation	
Word	**Count**
object	88
class	13
instance	12
method	7

Documents are a mix of topics



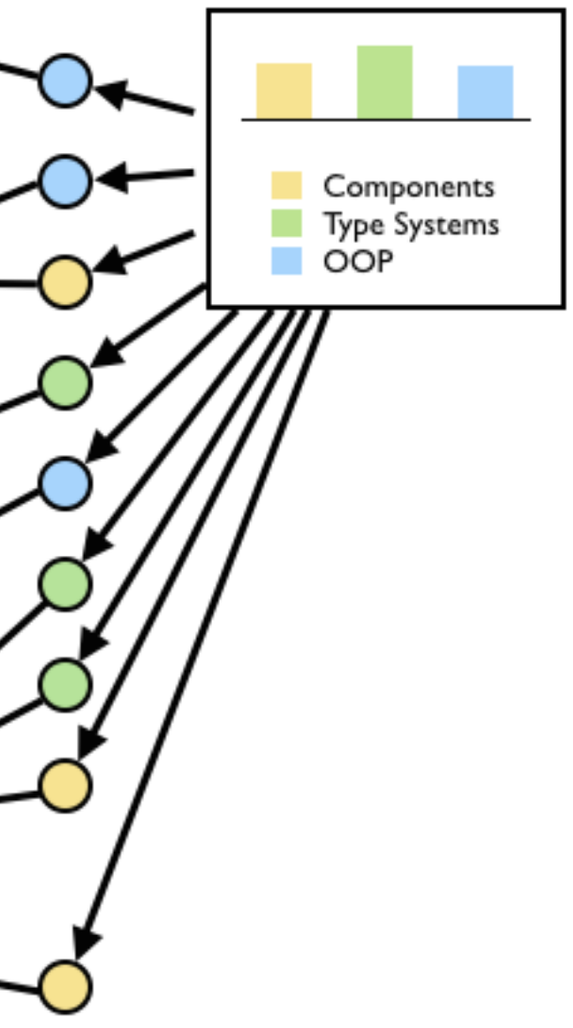
Generative LDA topic model

Gradual Typing for First-Class Classes *

Abstract

Dynamic type-checking and object-oriented programming often go hand-in-hand; scripting languages such as Python, Ruby, and JavaScript all embrace object-oriented (OO) programming. When scripts written in such languages grow and evolve into large programs, the lack of a static type discipline reduces maintainability. A programmer may thus wish to migrate parts of such scripts to a sister language with a static type system. Unfortunately, existing type systems neither support the flexible OO composition mechanisms found in scripting languages nor accommodate sound interoperation with untyped code.

In this paper, we present the design of a gradual typing system that supports sound interaction between statically- and dynamically-typed units of class-based code. The type system uses row polymorphism for classes and thus supports mixin-based OO composition. To protect migration of mixins from typed to untyped components, the system employs a novel form of contracts that partially seal classes. The design comes with a theorem that guarantees the soundness of the type system even in the presence of untyped components.



OOPSLA'12, October 19–26, 2012, Tucson, Arizona, USA.

Takikawa, Strickland, Dimoulas, Tobin-Hochstadt, and Felleisen
Gradual typing for first-class classes. OOPSLA 2012.

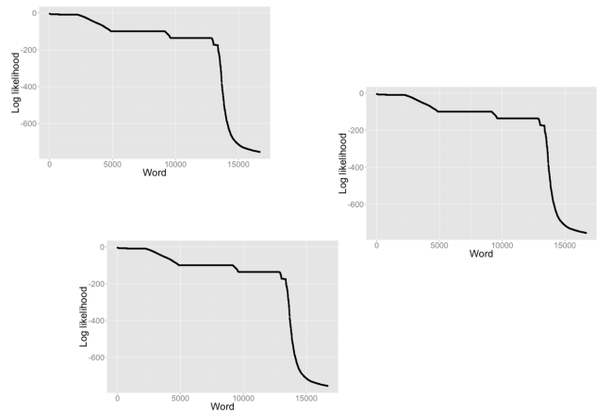
Inference with LDA

N bags of words

LDA-C*

k

N vectors, **k**-dimensional space

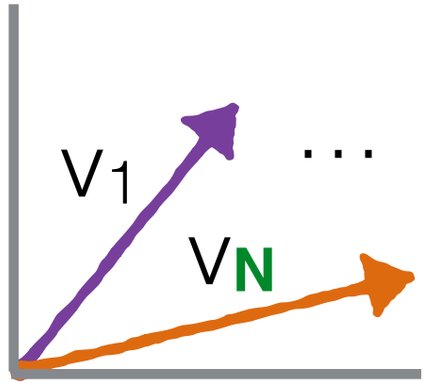


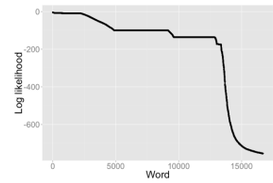
k topics

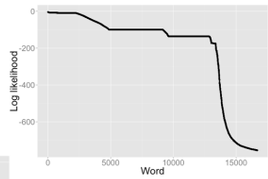
*http://www.cs.princeton.edu/~blei/lda-c/

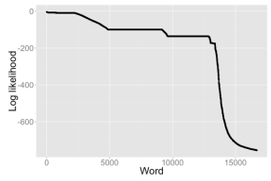
parse

LDA-C
k









N vectors

k topics

post

corpus
N docs

N bags of words

combined vocabulary

k top words
k top papers

aggregate vectors
by year
by conference

by hand

k topic names

Parsing

- Parsing drops standard **stopwords**
  - Added some extra ones with TF-IDF
- **Stemmed** words using nltk*
  - Removes plurals, etc.

a
about
above
after
again
against
…

calculi → calculus
goes → go

*http://www.nltk.org/

Our corpora

- **Abstracts:** ICFP, OOPSLA, PLDI, POPL
  - 4,355 documents
    - Imperfect data in the ACM Digital Library
- **Fulltext:** PLDI, POPL
  - 2,257 documents
  - Imperfect PDF-to-text conversion

Let’s name a topic!

object

heap

region

memor

pointer

collector

garbage

collection

allocation

reference

Space overhead

ent with

n

r systems

llection

and their

ound garbage collection gridlock

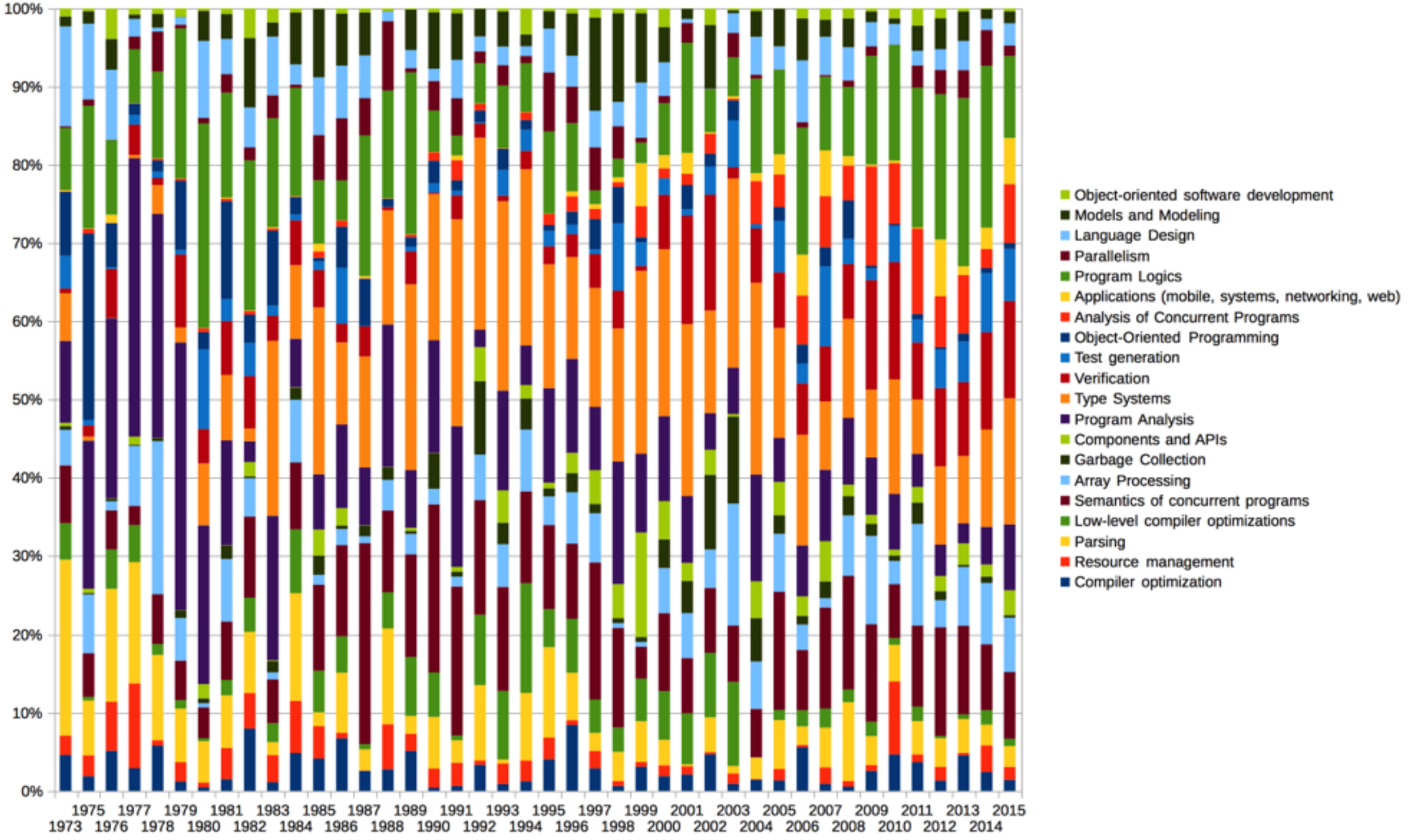
On bounding time and space for multiprocessor garbage collection

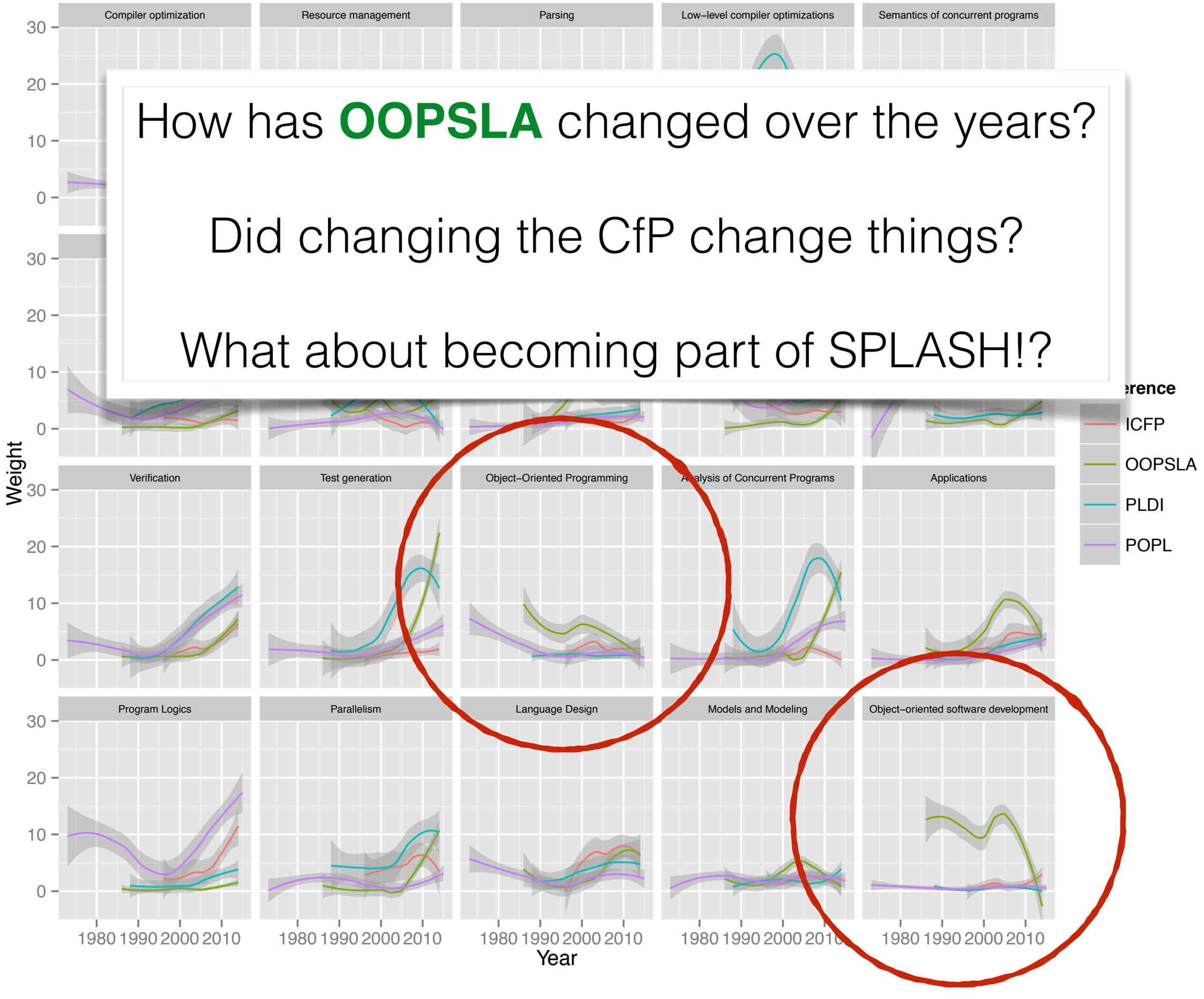
Garbage collection without paging

Garbage collection!

Topic names for k=20, abstracts

Compiler optimization	Array Processing	Verification	Program Logics
Resource management	Garbage Collection	Test generation	Parallelism
Parsing	Components and APIs	Object-Oriented Programming	Language Design
Low-level compiler optimizations	Program Analysis	Analysis of Concurrent Programs	Models and Modeling
Semantics of concurrent programs	Type Systems	Applications	Object-oriented software development


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
1973 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015
Object-oriented software development
Models and Modeling
Language Design
Parallelism
Program Logics
Applications (mobile, systems, networking, web)
Analysis of Concurrent Programs
Object-Oriented Programming
Test generation
Verification
Type Systems
Program Analysis
Components and APIs
Garbage Collection
Array Processing
Semantics of concurrent programs
Low-level compiler optimizations
Parsing
Resource management
Compiler optimization



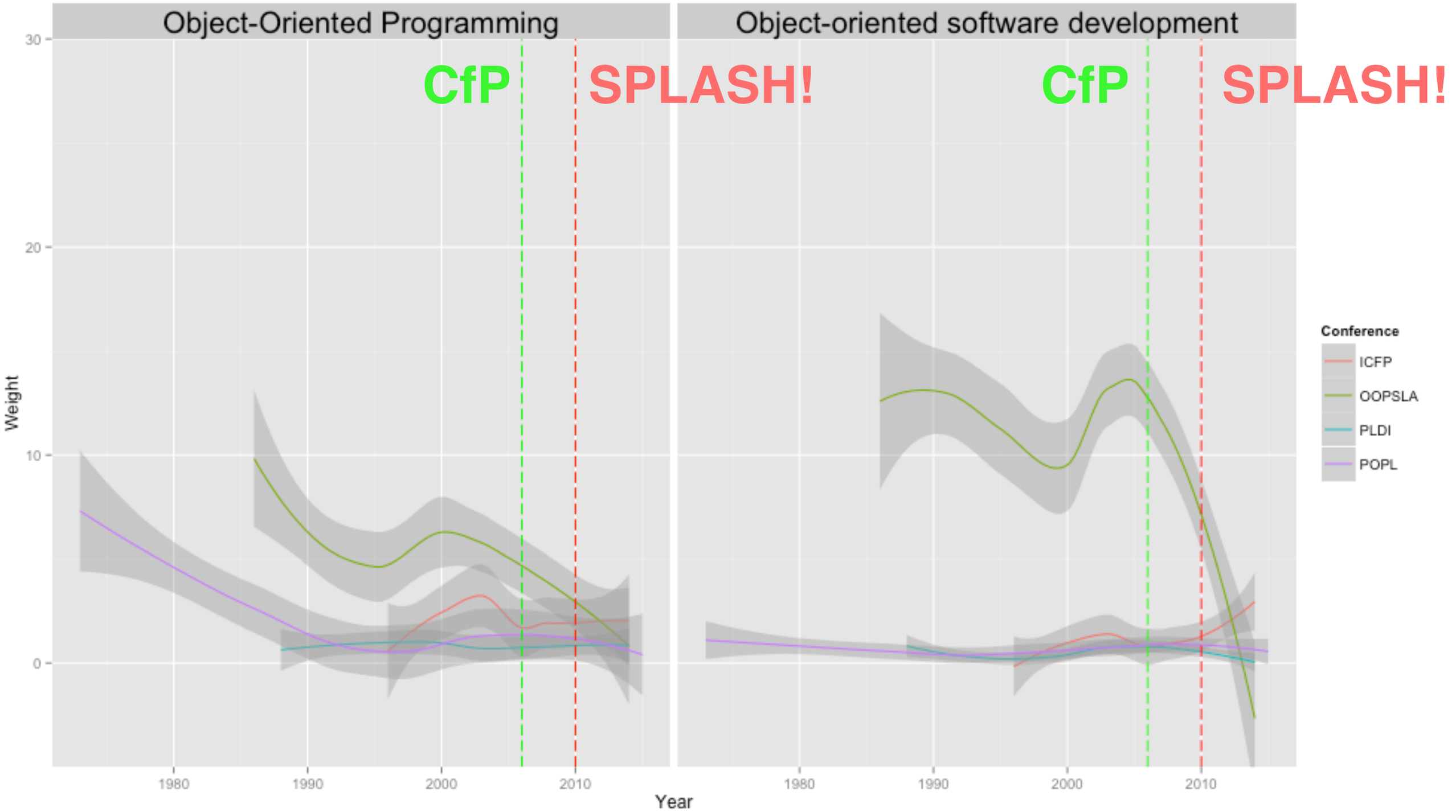
How has **OOPSLA** changed over the years?

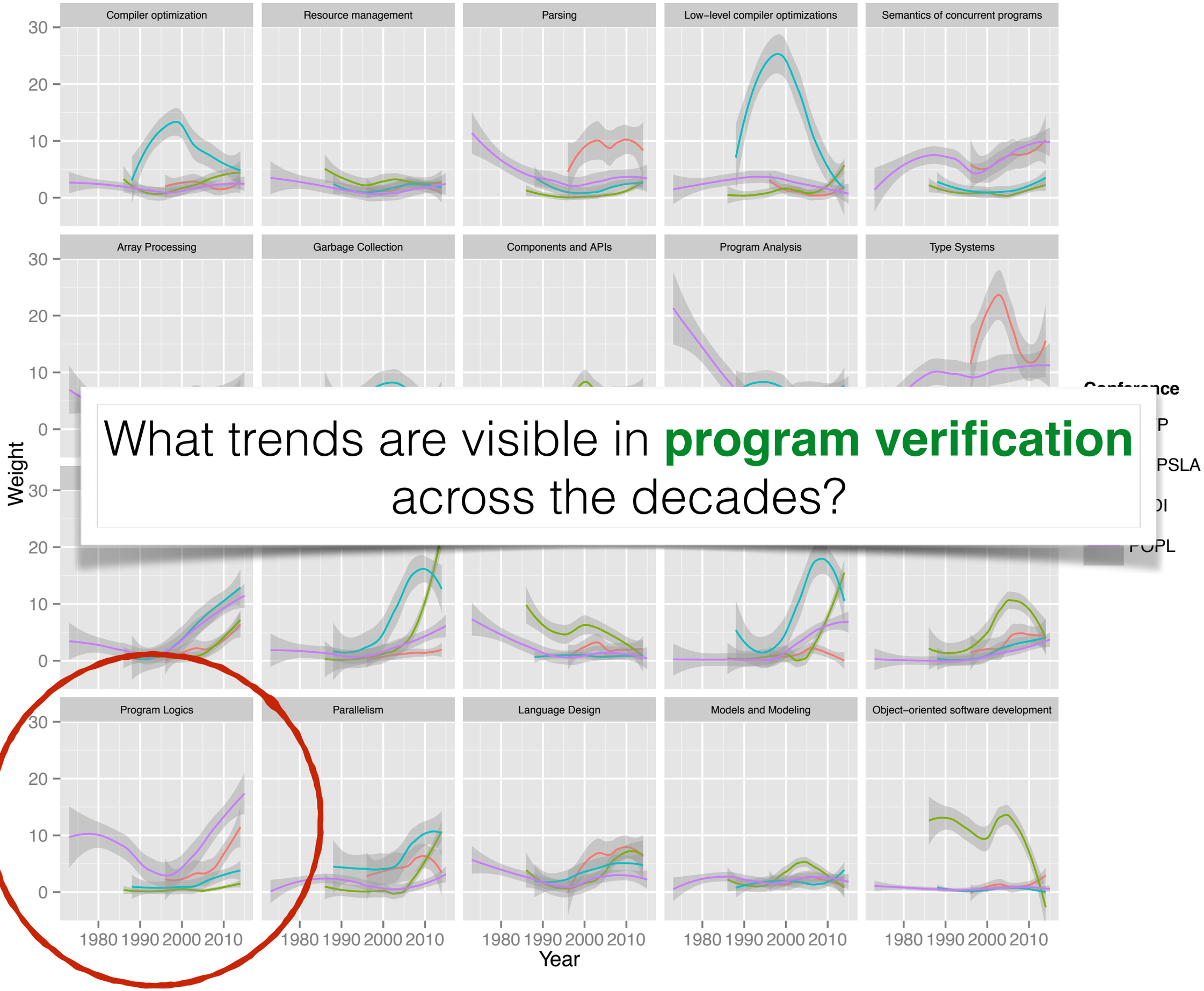
Did changing the CfP change things?

What about becoming part of SPLASH!?

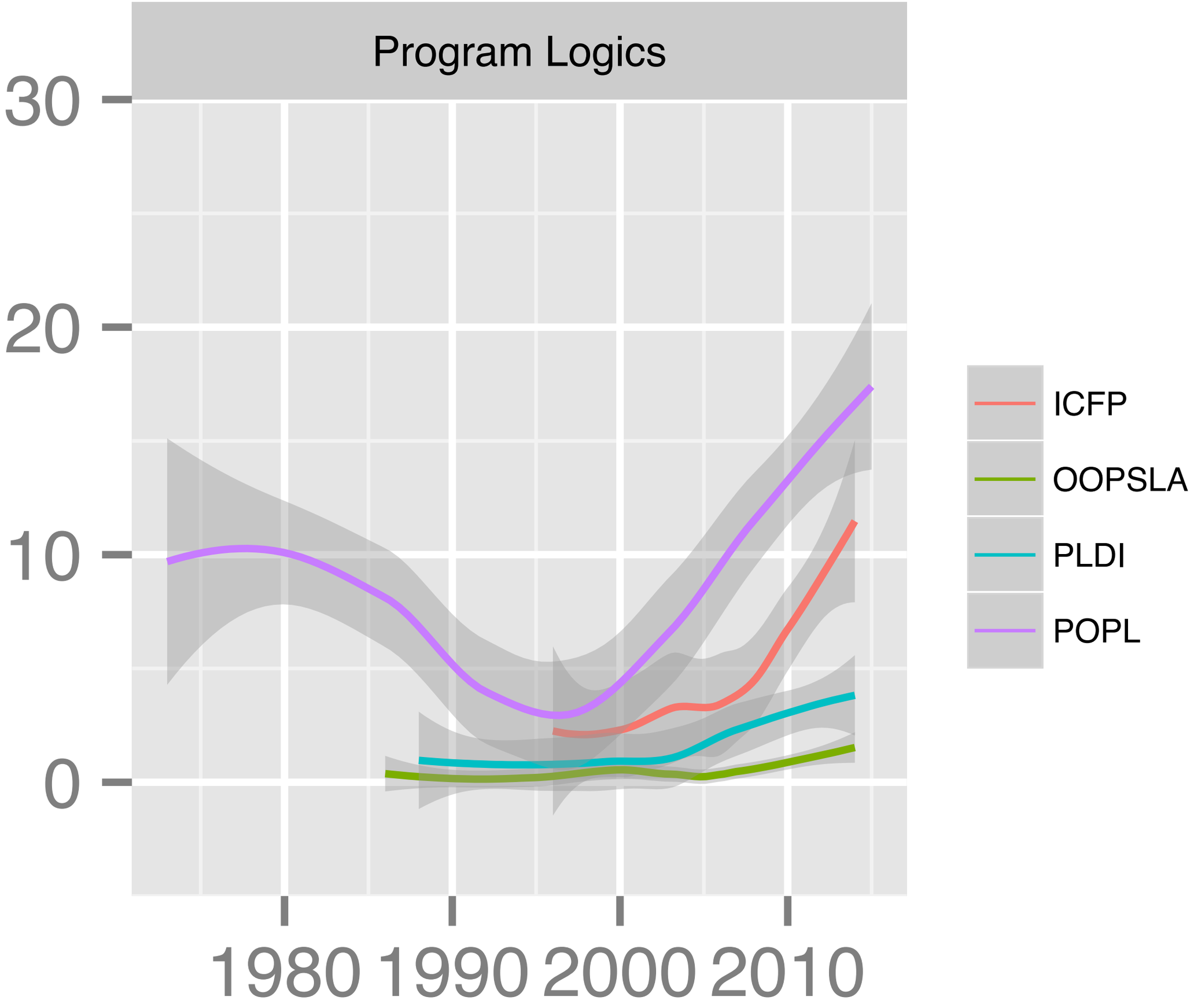
OOPSLA Call for Papers

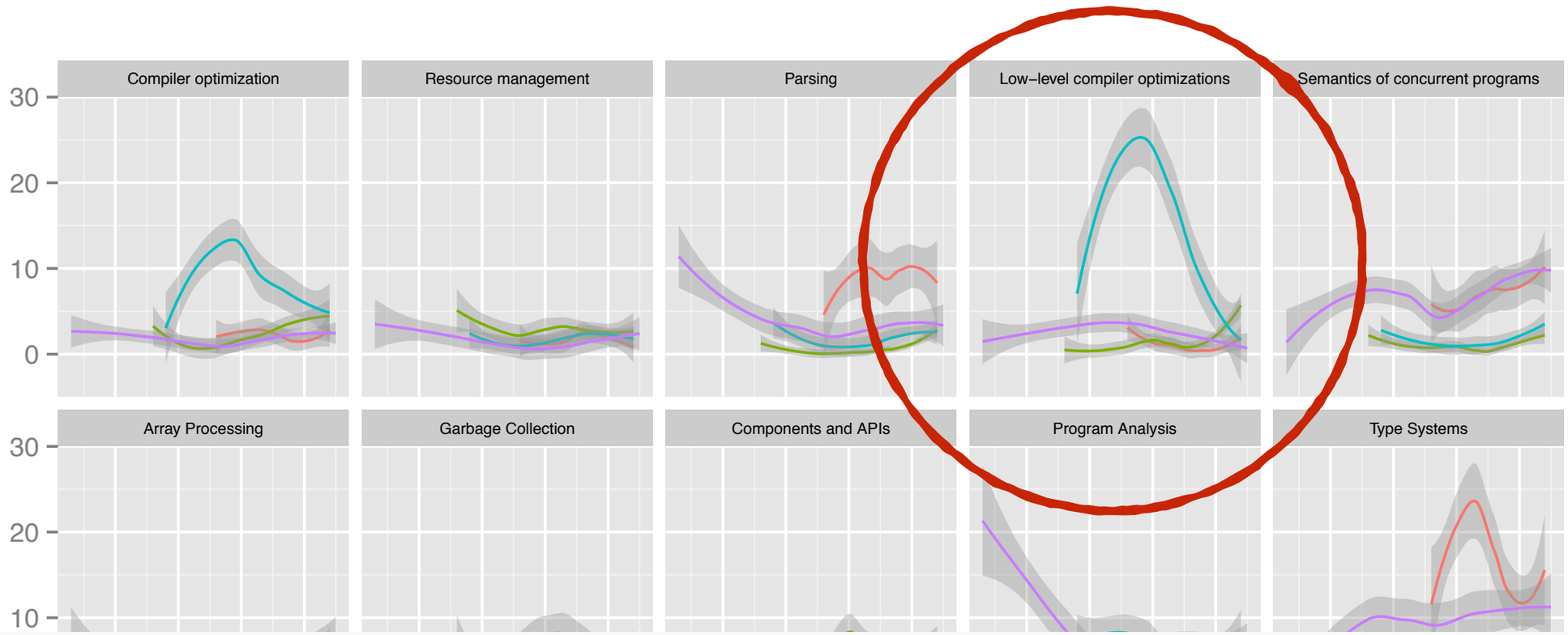
2006	2007	2010
foundations of **object and related technologies**	paradigms **beyond the traditional concept** of object-oriented programming	**all aspects** of programming languages and software engineering, **broadly construed**


Object-Oriented Programming
Object-oriented software development
CfP
SPLASH!
CfP
SPLASH!
Weight
30
20
10
0
1980
1990
2000
2010
Year
Conference
ICFP
OOPSLA
PLDI
POPL



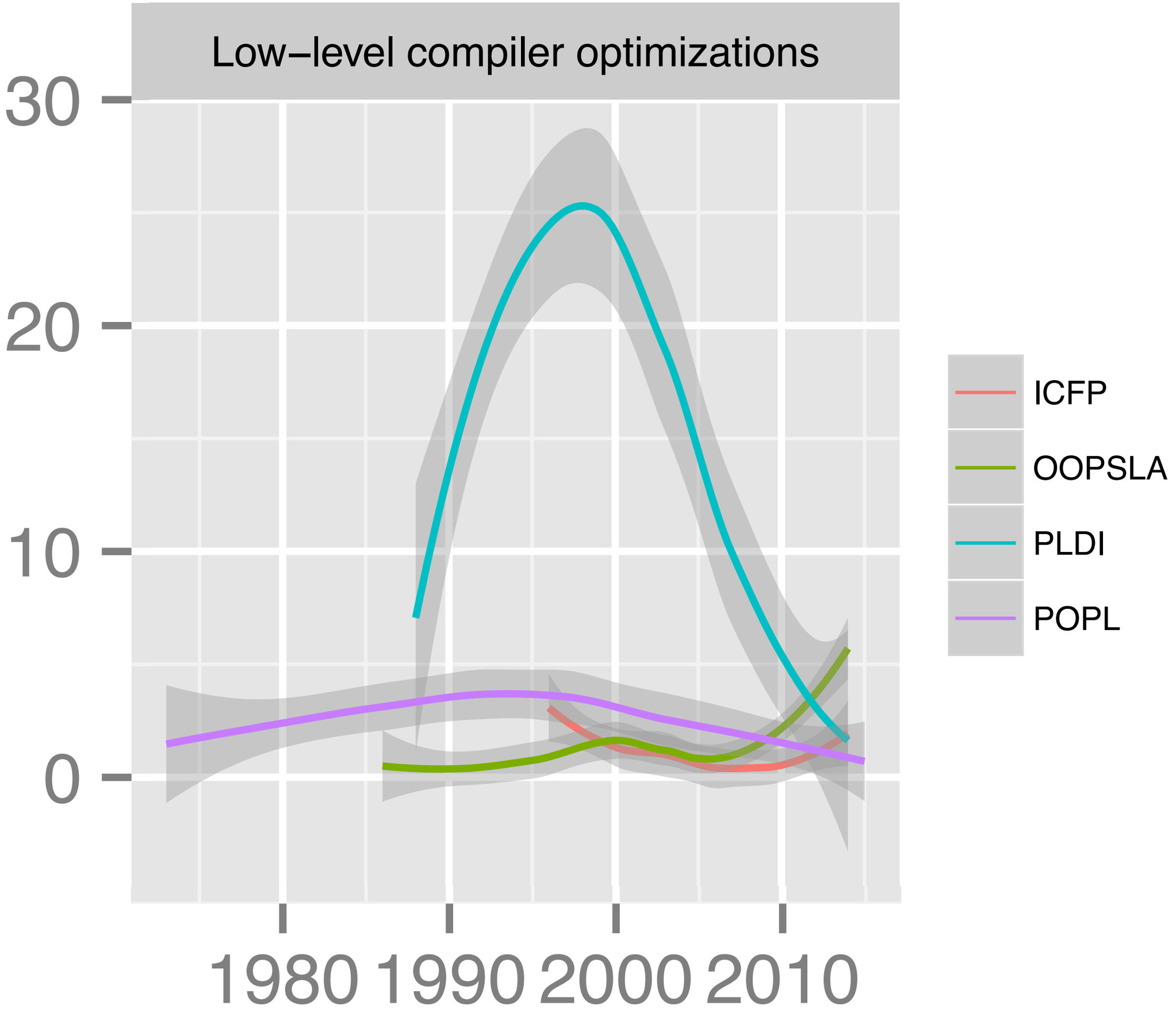
What trends are visible in **program verification** across the decades?


Program Logics
30
20
10
0
1980
1990
2000
2010
ICFP
OOPSLA
PLDI
POPL



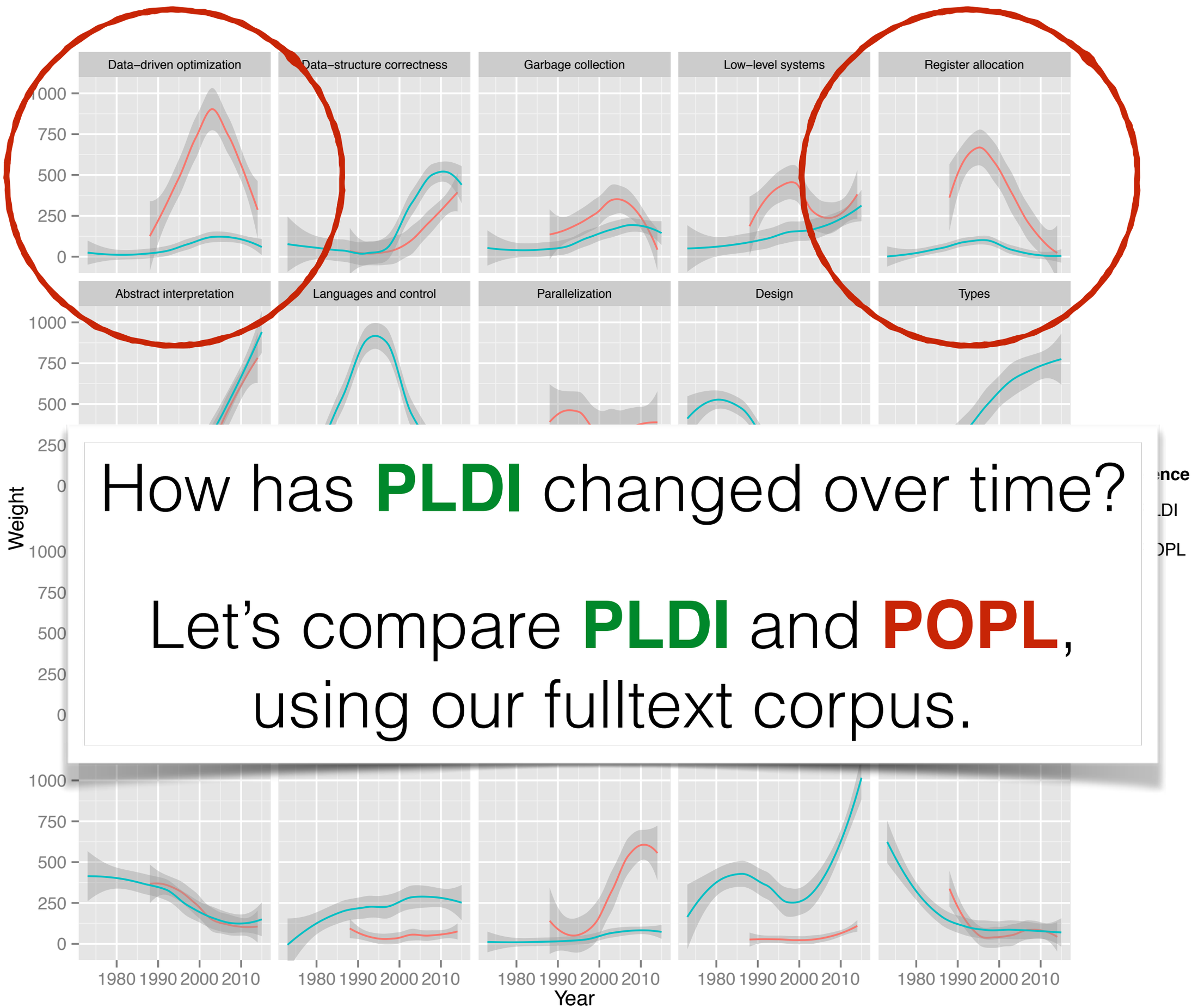
How has **PLDI** changed over time?

Per “Future of PLDI” session in Edinburgh, what is the state of the community?


Low−level compiler optimizations
30
20
10
0
1980
1990
2000
2010
ICFP
OOPSLA
PLDI
POPL

Topic names for k=20, full text

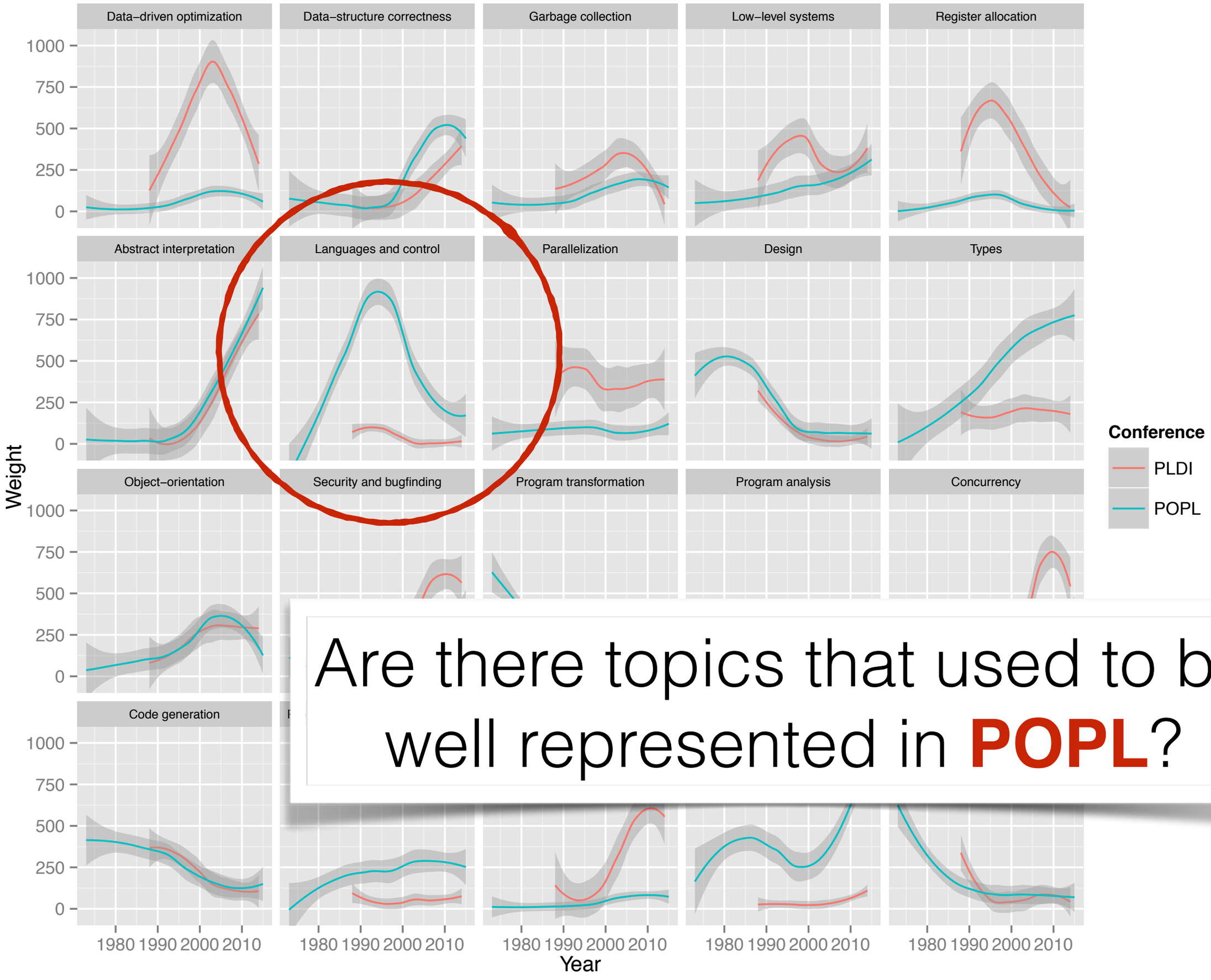
Data-driven optimization	Abstract interpretation	Object-orientation	Code generation
Data-structure correctness	Languages and control	Security and bugfinding	Processes and message passing
Garbage collection	Parallelization	Program transformation	Dynamic analysis
Low-level systems	Design	Program analysis	Proofs and models
Register allocation	Types	Concurrency	Parsing



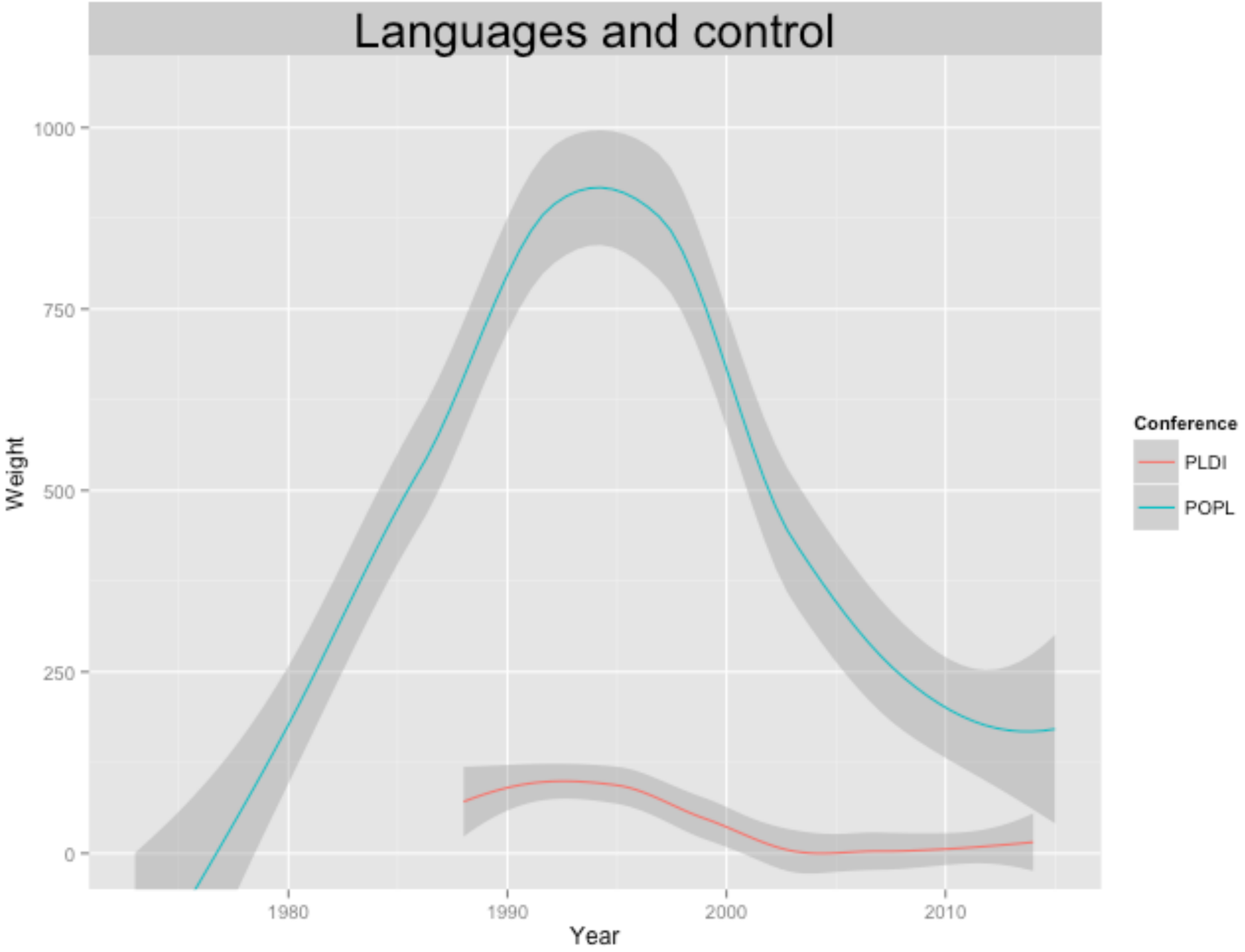
How has **PLDI** changed over time?

Let's compare **PLDI** and **POPL**, using our fulltext corpus.

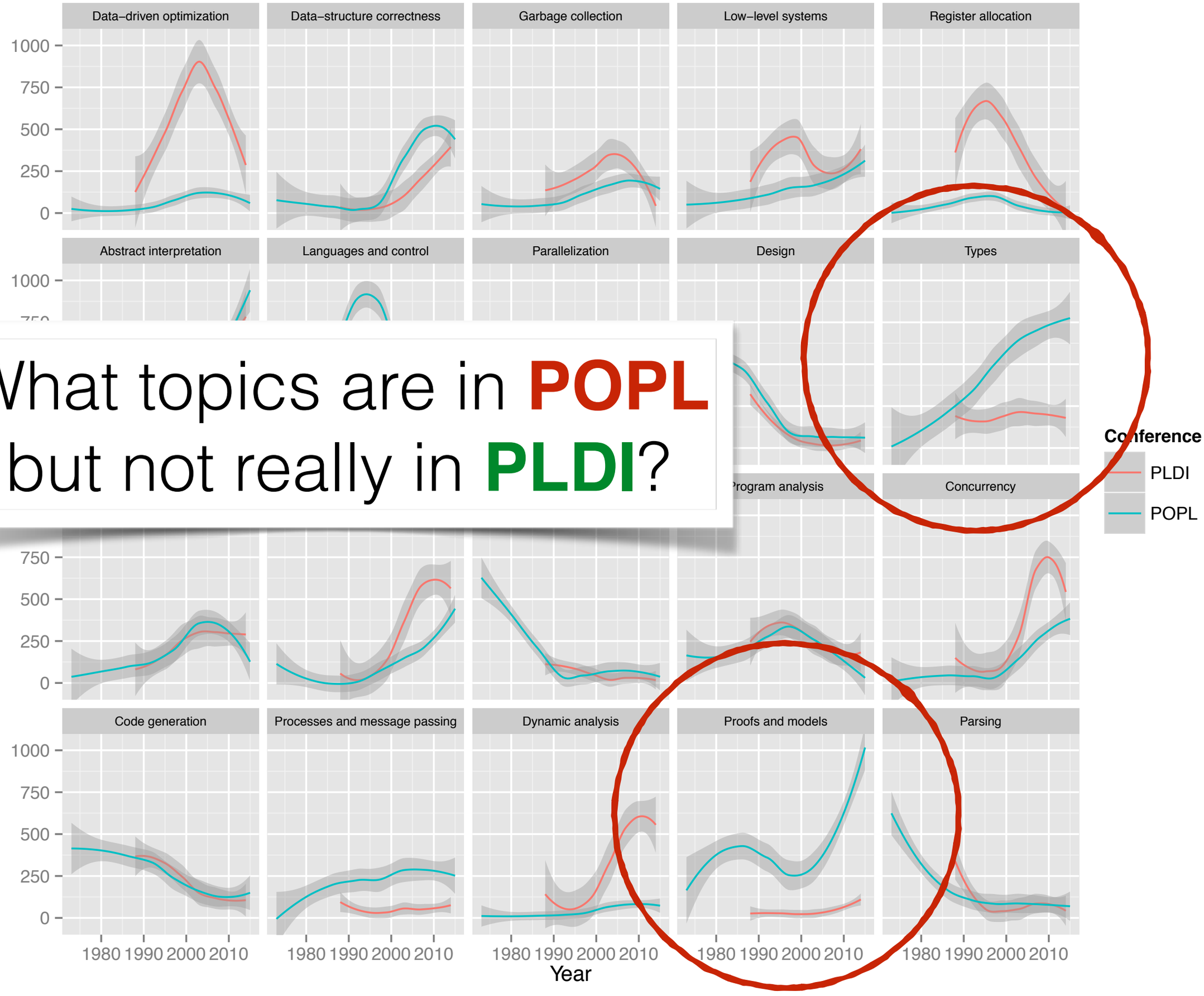

Data-driven optimization
Register allocation
Weight
1000
750
500
250
0
1980
1990
2000
2010
Year
Conference
PLDI
POPL

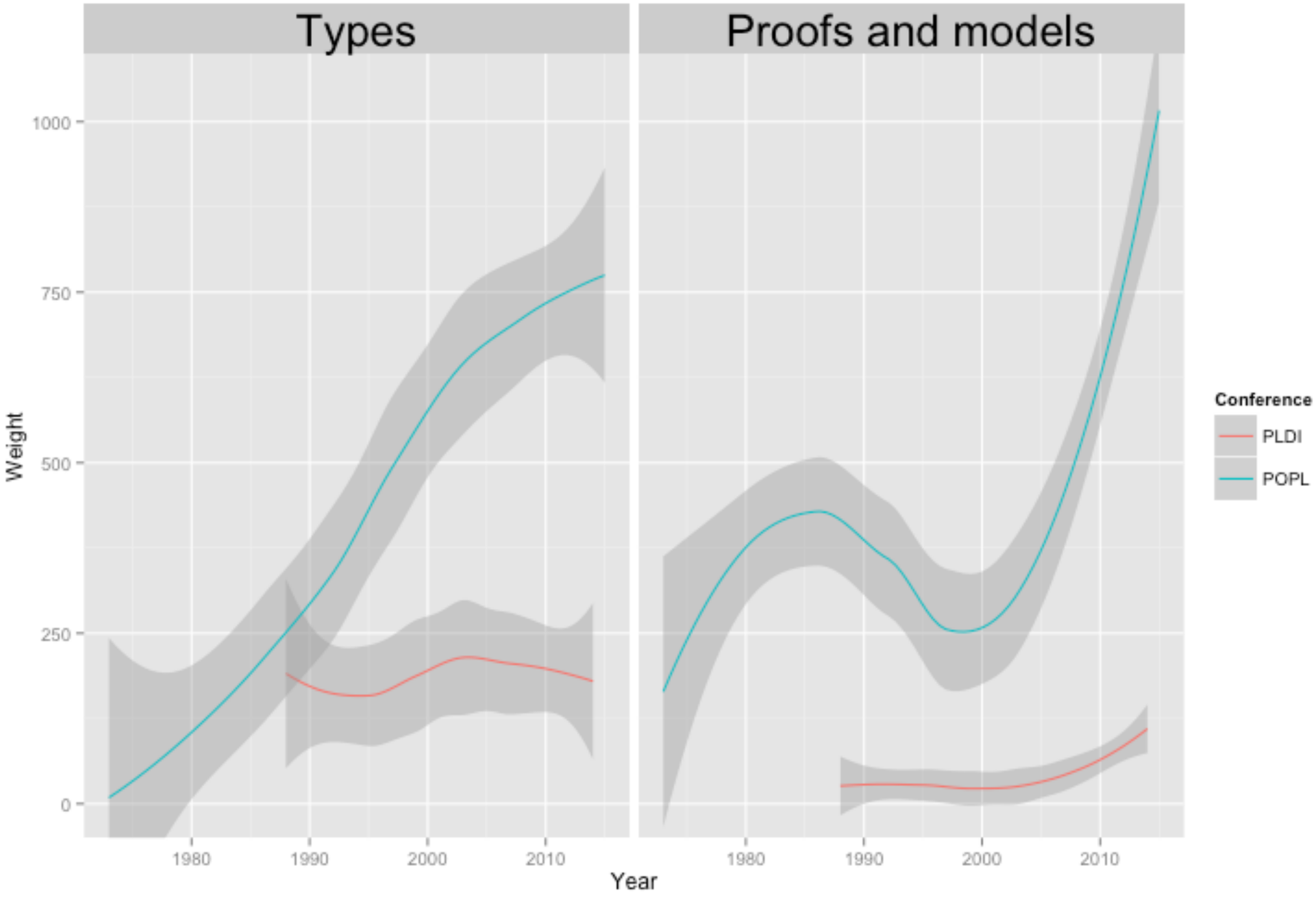


Are there topics that used to be well represented in **POPL**?

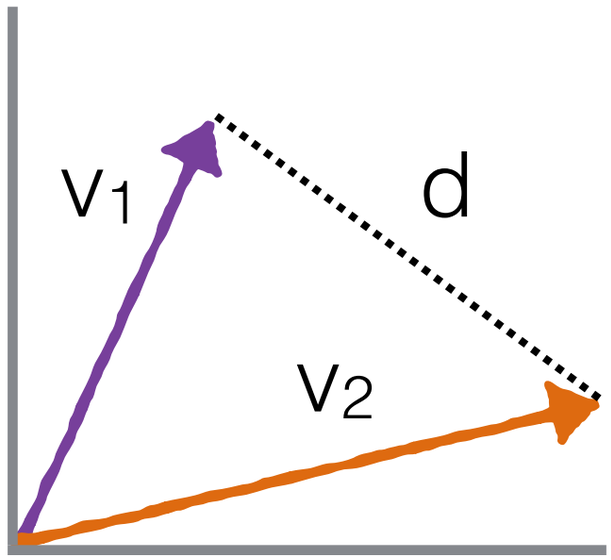

Languages and control
1000
750
500
250
0
Weight
1980
1990
2000
2010
Year
Conference
PLDI
POPL

What topics are in **POPL** but not really in **PLDI**?



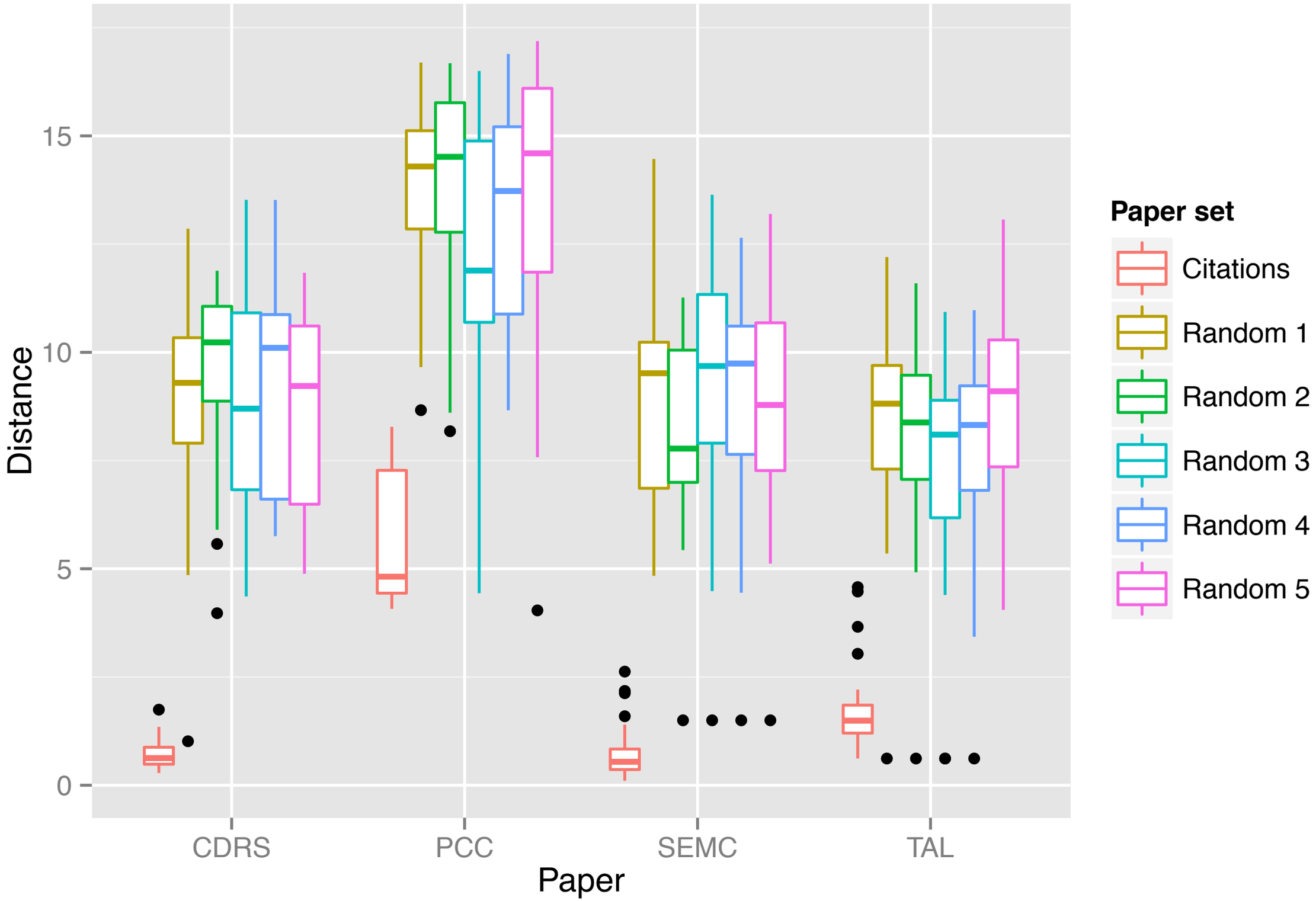

Types
Proofs and models
1000
750
500
250
0
Weight
1980
1990
2000
2010
Year
Conference
PLDI
POPL

Comparing documents



Are papers with **close** topic vectors **related**?

Measure distance using *Symmetrized KL divergence*, which gives less weight to dimensions with small magnitude.


Distance
15
10
5
0
CDRS
PCC
SEMC
TAL
Paper
Paper set
Citations
Random 1
Random 2
Random 3
Random 4
Random 5

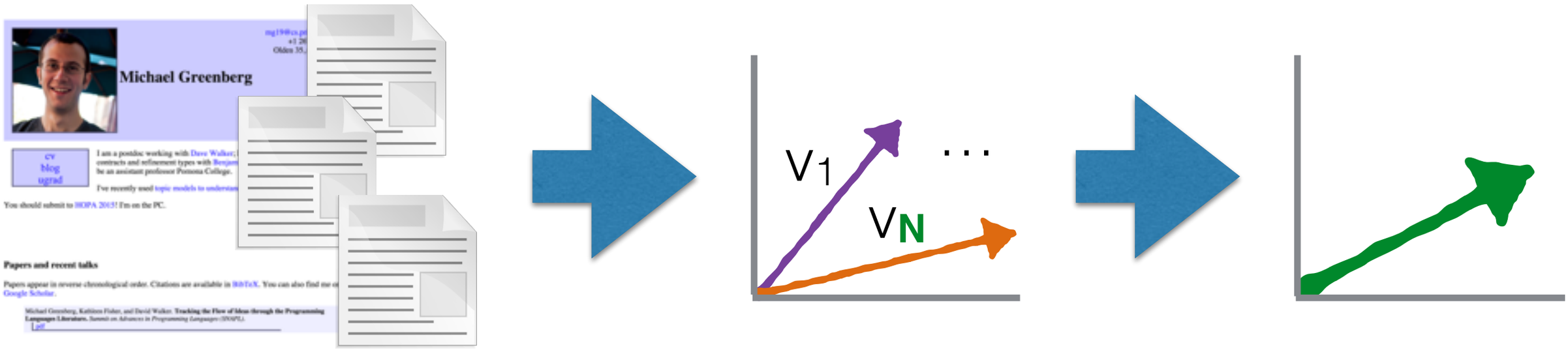
http://tmpl.weaselhat.com

Ideas and plans

Beginning of a new project

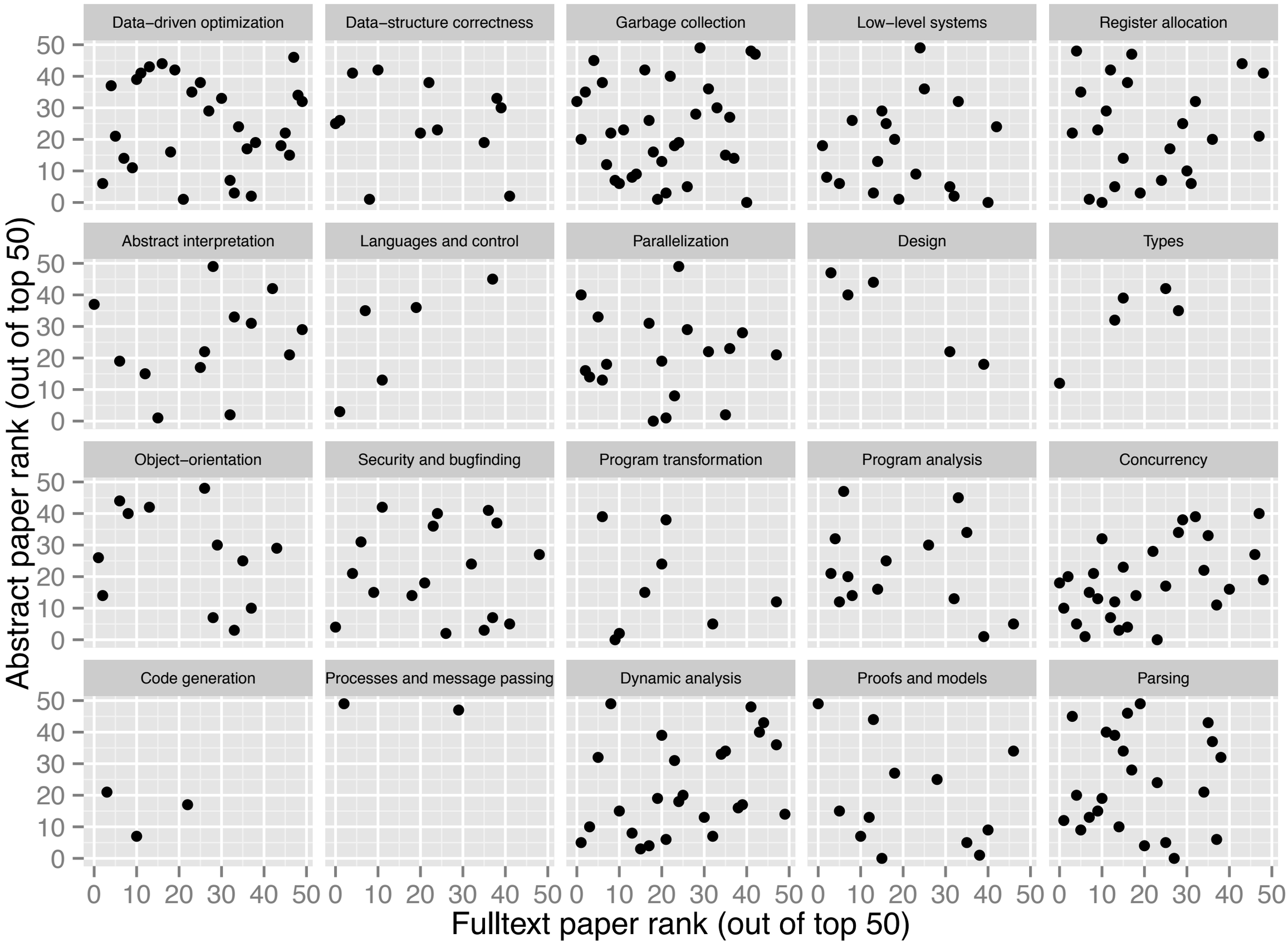
What do *you* think we should do?

Models for researchers



Limitations/problems

- ACM DL is missing data
  - No programmatic access
- Unclear choices about models
  - Abstracts or fulltext? k=20? k=30? k=200?
  - Which documents should ‘seed’ LDA?


Data−driven optimization
Data−structure correctness
Garbage collection
Low−level systems
Register allocation
Abstract interpretation
Languages and control
Parallelization
Design
Types
Object−orientation
Security and bugfinding
Program transformation
Program analysis
Concurrency
Code generation
Processes and message passing
Dynamic analysis
Proofs and models
Parsing
Abstract paper rank (out of top 50)
Fulltext paper rank (out of top 50)
0
10
20
30
40
50

(More) Questions?