SWAT

Tolerance in Grammarware

PEM Colloquium
Vadim Zaytsev, SWAT, CWI

Grammar-based source code analysis

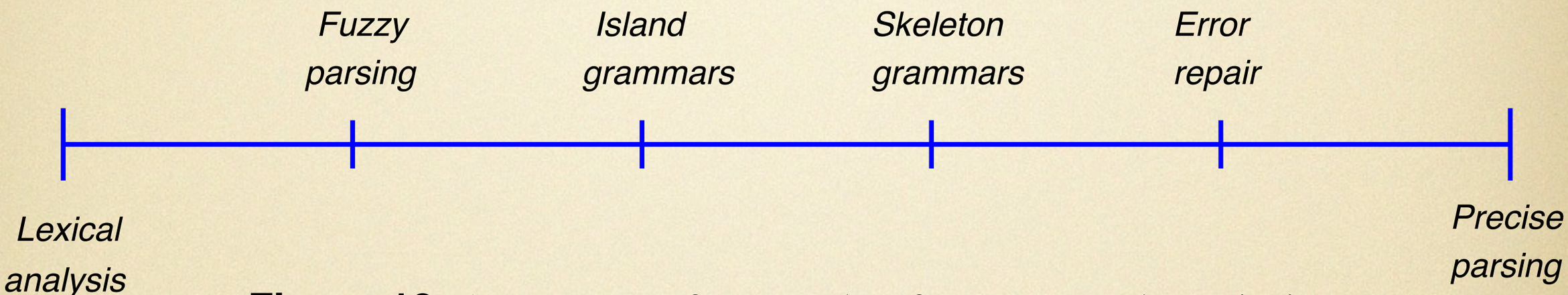


Figure 10. A spectrum of approaches for source code analysis.

- A spectrum of approaches w.r.t. tolerance
- We will go from right to left
- Figure borrowed (for extension) from:

S. Klusener, R. Lämmel, *Deriving Tolerant Grammars from a Base-line Grammar*, ICSM 2003

Precise parsing

A. V. Aho, J. D. Ullman, *The Theory of Parsing, Translation, and Compiling*, 1972.
A. V. Aho, J. D. Ullman, *Principles of Compiler Design*, 1977.
A. V. Aho, R. Sethi, J. D. Ullman, *Compilers*, 1985.
A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman, *Compilers*, 2006.
D. Grune, C. J. H. Jacobs, *Parsing Techniques: A Practical Guide*, 2008.

Error repair: panic mode

- The simplest method to detect multiple syntax errors
- Provide a list of synchronising tokens (beacon symbols)
  - ;
  - }
  - ...anything obvious and unambiguous
- In case of error, skip everything until the next synchronising token

A. V. Aho, R. Sethi, J. D. Ullman, *Compilers*, §4.1, 1985.

Error repair: phrase level

- Local correction
- Default options for symbols
- Typically
  - insert ; if it is not present
  - balance the brackets
  - ...most heuristics of later blocks of Grammar Hunter
- Sometimes, real error occurs before the detection point

A. V. Aho, R. Sethi, J. D. Ullman, *Compilers*, §4.1, 1985.

Permissive grammars

- Insertion recovery rules
- Substitution recovery rules
- Choose interpretation with minimum recoveries
- Aimed at error handling
  - error repair
  - error reporting

L. C. L. Kats, M. de Jonge, E. Nilsson-Nyman, E. Visser,
Providing Rapid Feedback in Generated Modular Language Environments, OOPSLA 2009

Global error correction

- Given string x and grammar G,
  - If $x \notin L(G)$,
  - construct string y such that
  - $y \in L(G)$
  - number of changes from x to y is minimal
- The closest program is not always the intended one

A. V. Aho, R. Sethi, J. D. Ullman, *Compilers*, §4.1, 1985.

Hierarchical error repair

- Think of a parser as a state machine
- For every state, there are transitions for allowed tokens
- If an error occurs, no transition for the input token
- A non-error transition is taken
  - based on synchronisation stack
- Compatible at least with LR and LL

D. T. Barnard, R. C. Holt, *Hierarchic Syntax Error Repair for LR Grammars*, IJCIS 11:4, 1982

Error productions

- Know your enemy
  - Define your enemy with a grammar
- Works well for known kinds of errors
- (Should this be a part of a language?)
  - permissive grammars

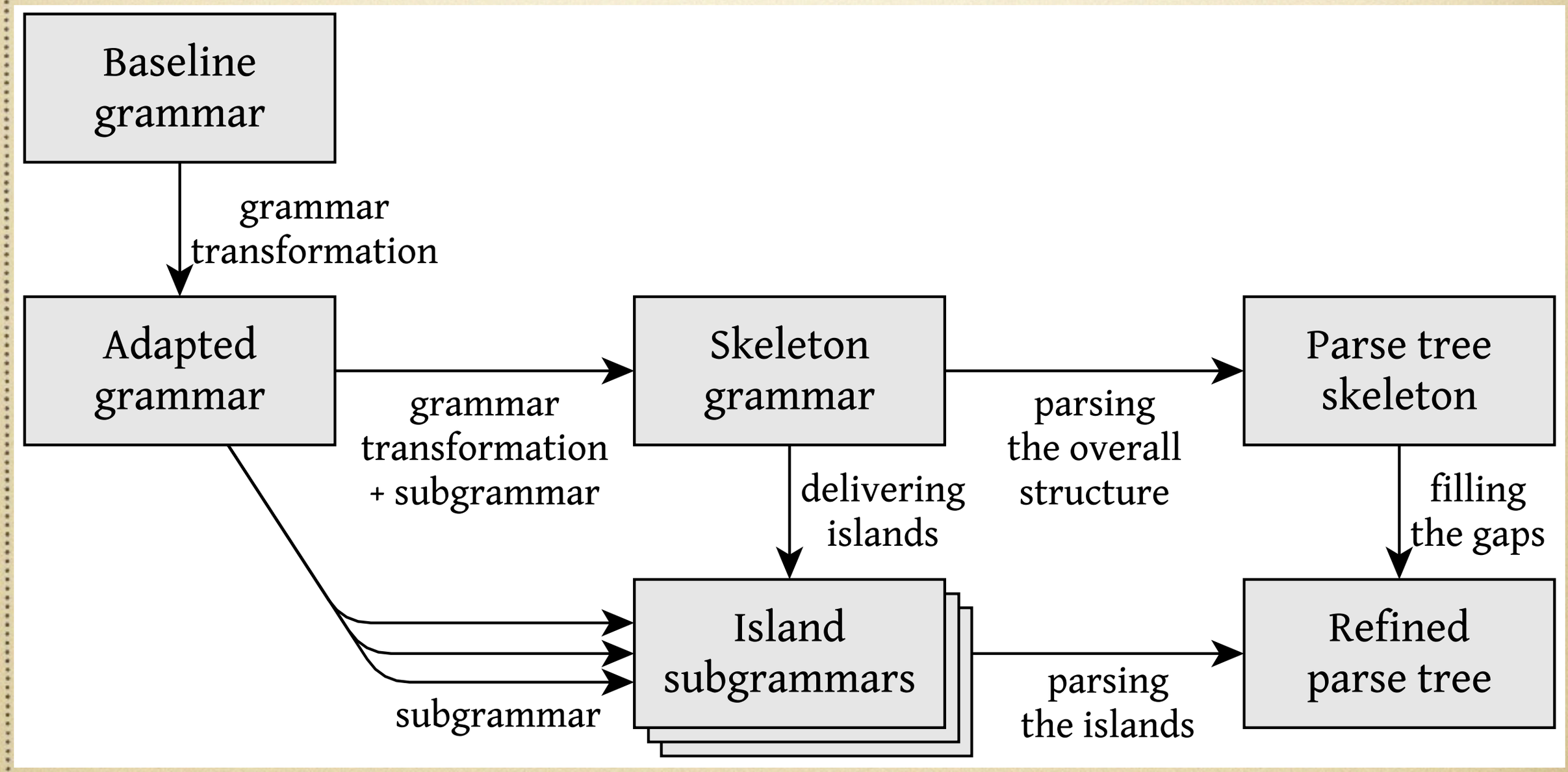
A. V. Aho, R. Sethi, J. D. Ullman, *Compilers*, §4.1, 1985.

Multiple passes

- Lazy iterative skeleton grammar parsing
- Parse first with a skeleton grammar
  - obtain the global structure
- Parse the islands with subgrammars
  - if possible
- Also enables
  - “grammarware as a service” and “parsing in the cloud”

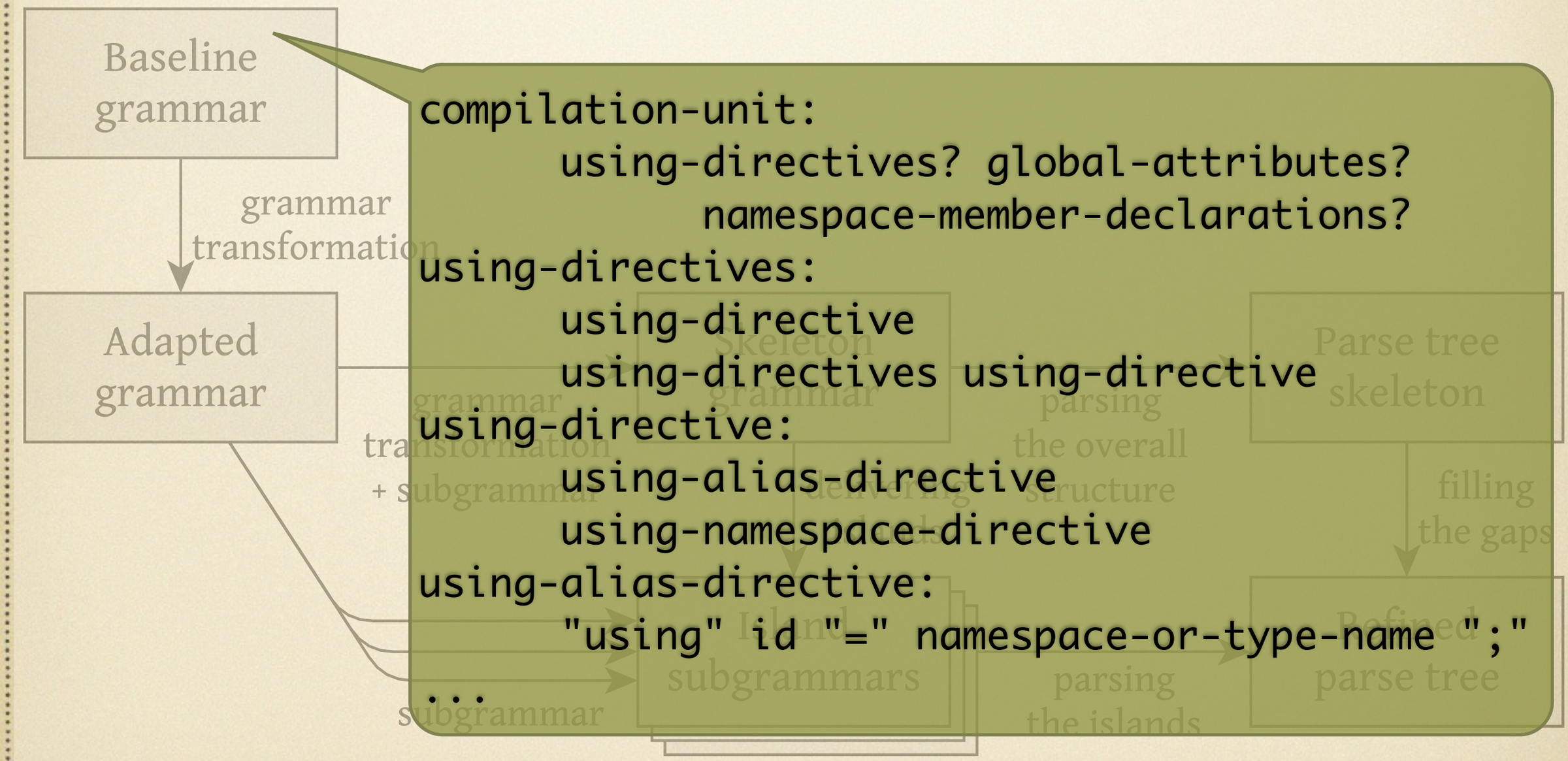
Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM’12

Parsing in the cloud



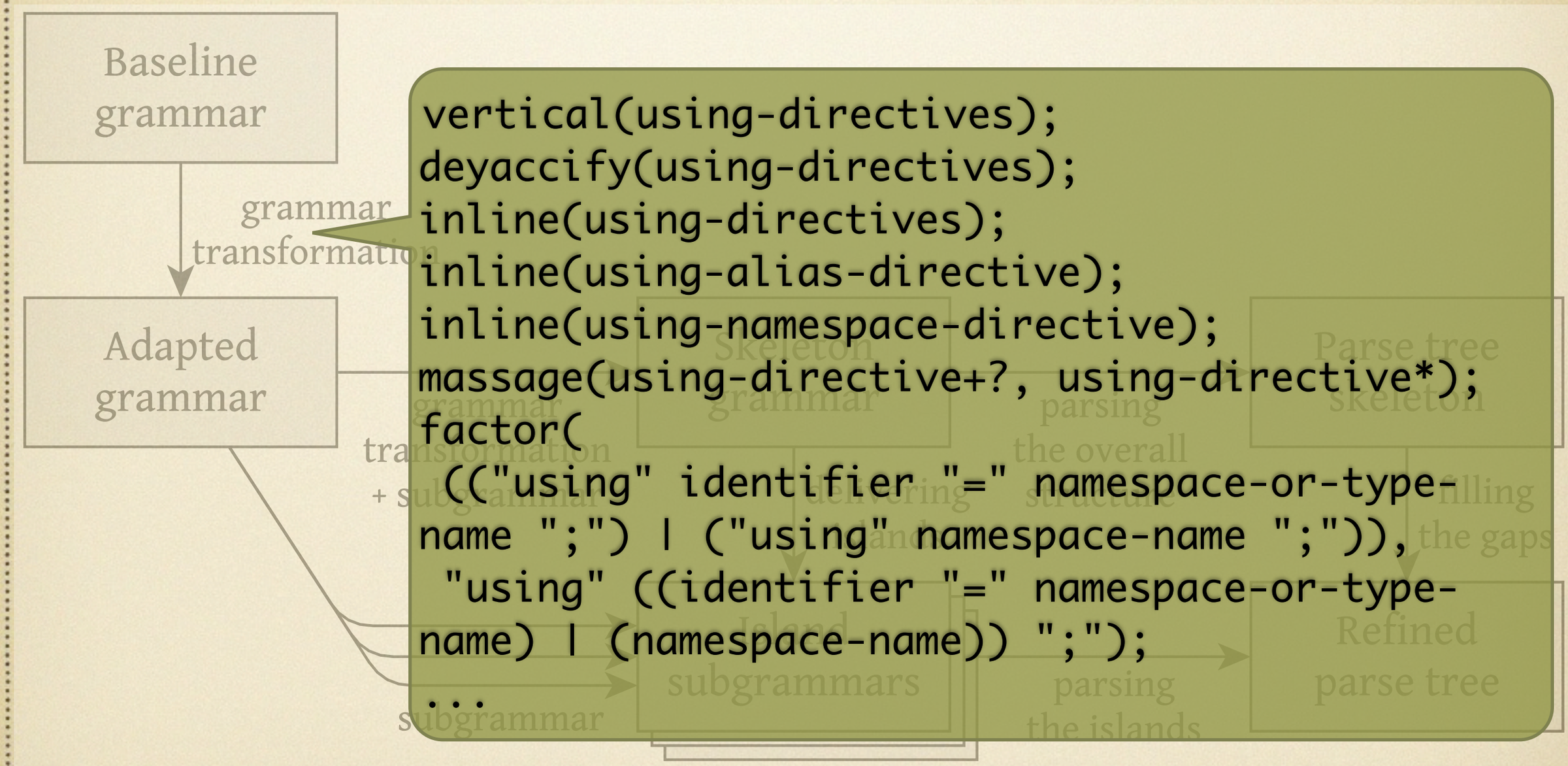
Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM'12

Parsing in the cloud



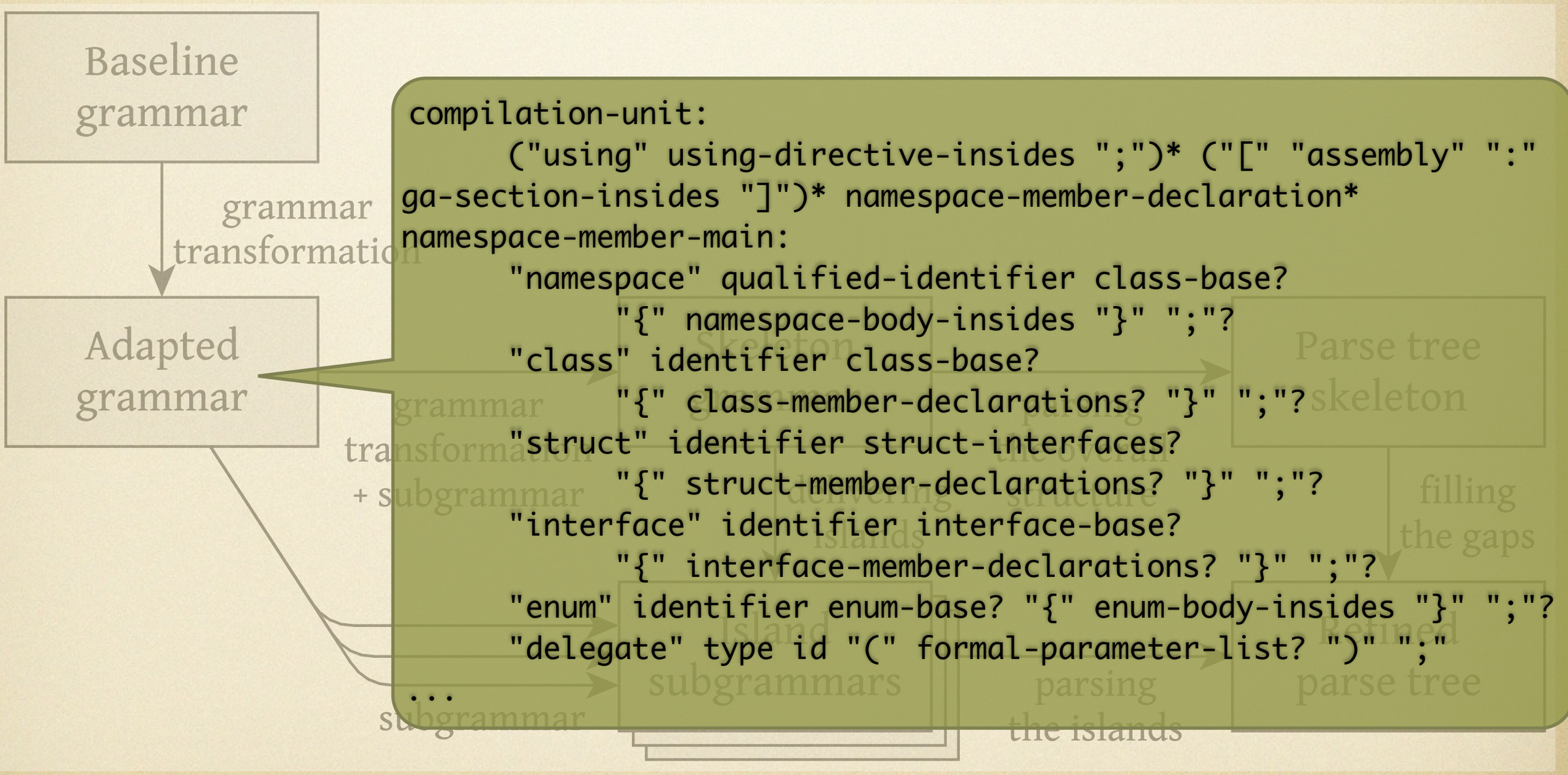
Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM'12

Parsing in the cloud



Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM'12

Parsing in the cloud



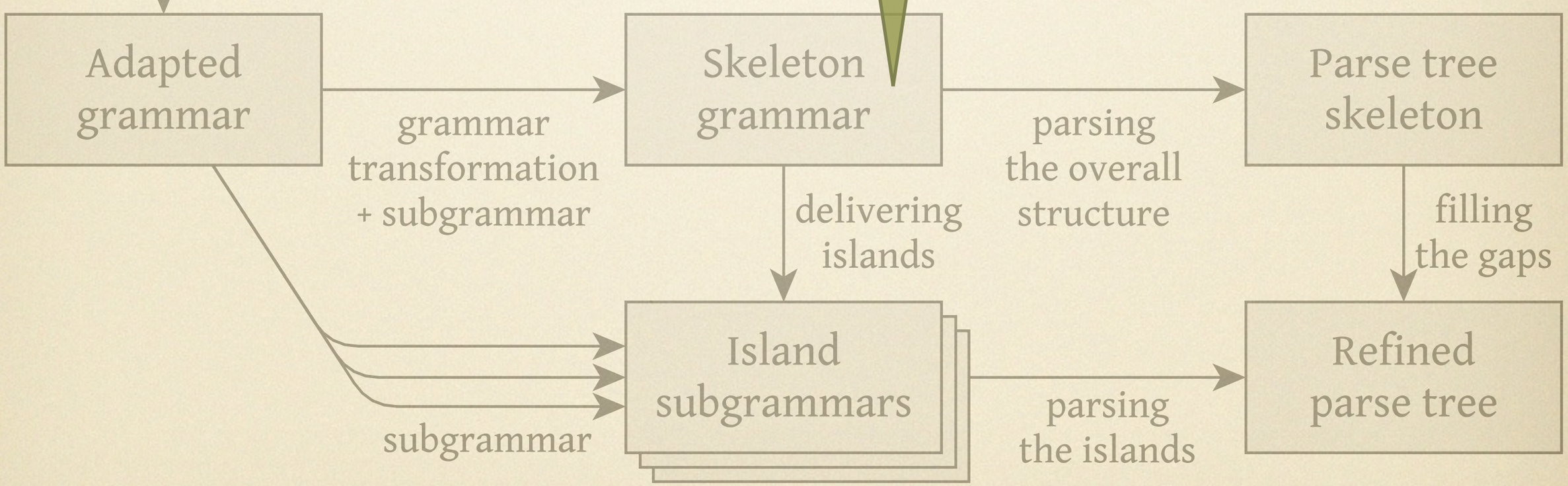
Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM'12

```
layout L = [\ \t\r\n]* !>> [\ \t\r\n] ;

syntax CompilationUnit = ("using" NotSemicolon ";")*
       ("[" "assembly" ":" NotRightSquareBracket "]")*
       NamespaceMemberDeclaration* ;

syntax NotRightSquareBracket = NRSBChunk+ () >> [\]];
lexical NRSBChunk = ![\]\ \t\r\n]+ >> [\]\ \t\r\n];

...
```



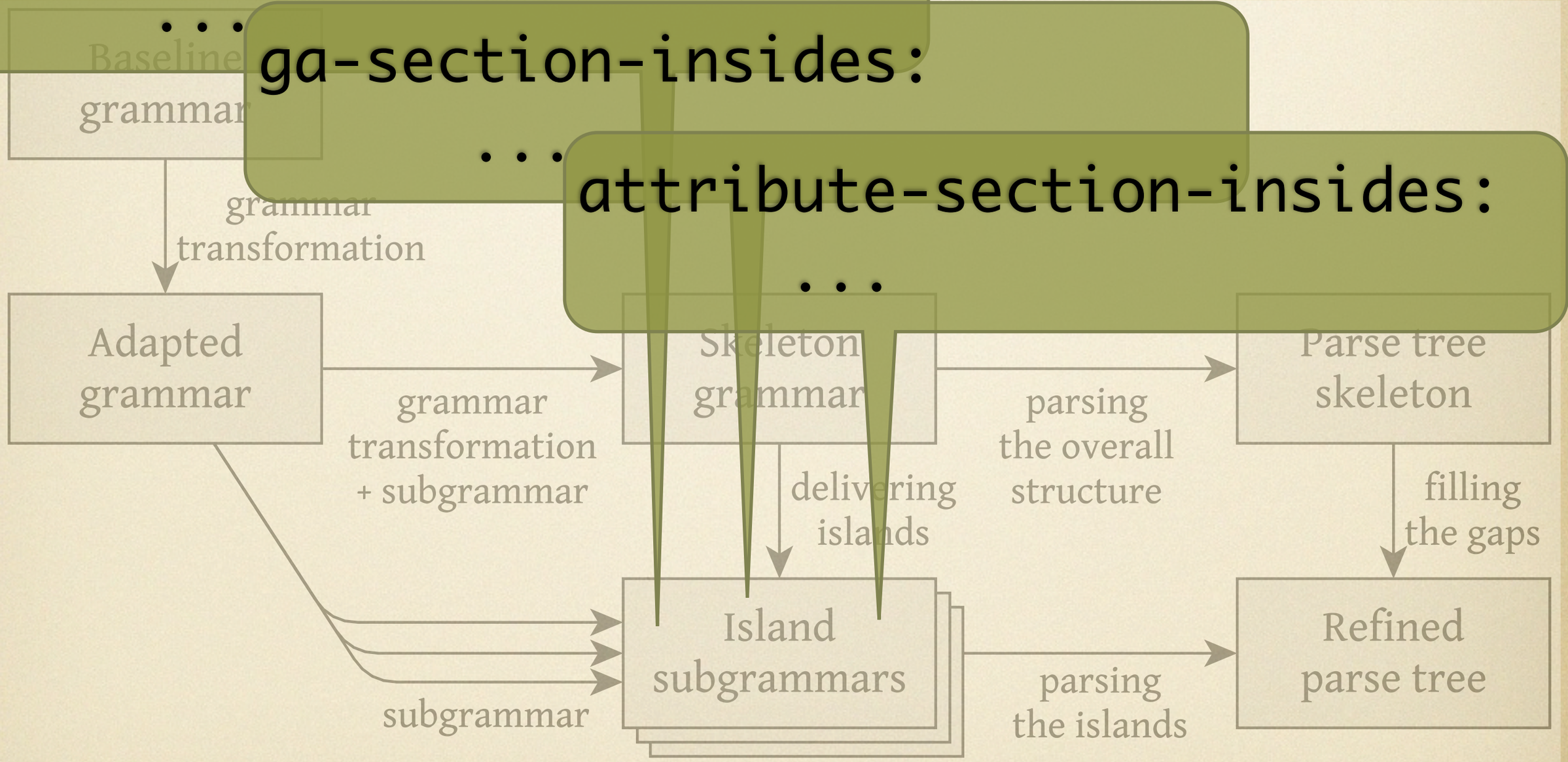
Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM'12

Parsing in the cloud

```
using-directive-insides:
  ...
```

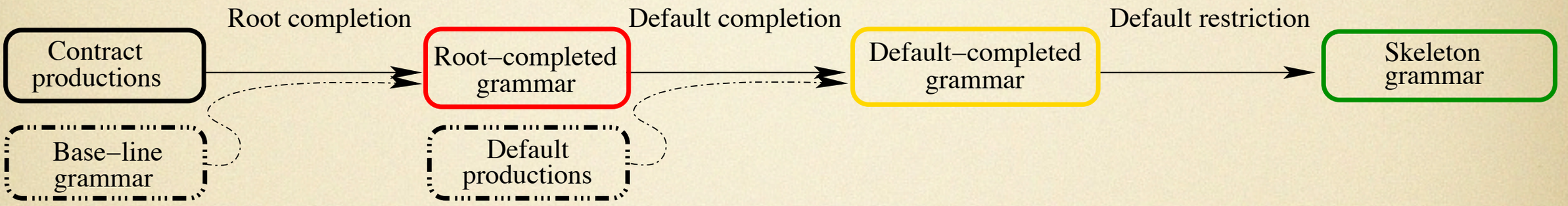
```
ga-section-insides:
  ...
```

```
attribute-section-insides:
  ...
```



Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM'12

Skeleton grammars



- Productions for interesting constructs are reused
- Default productions used for the rest
- Nonterminal mapping is maintained
  - facilitates reasoning about false positives & negatives

S. Klusener, R. Lämmel, *Deriving Tolerant Grammars from a Base-line Grammar*, ICSM 2003

Bridge grammars

- Bridges connect islands
  - can enclose other bridges but never cross
- Reefs add info to nearby islands
  - e.g., indentation and delimiters
- Can be further enhanced with
  - bridge repairer
  - artificial islands

E. Nilsson-Nyman, T. Ekman, G. Hedin. *Practical Scope Recovery using Bridge Parsing*, SLE'08

Robust parsing

- Combination of
  - error productions
  - island grammars for multiple languages
  - bridges between islands are parts of islands
- Works well for multilingual parsing
  - e.g., VB + JS + ASP + HTML

N. Synytskyy, J. R. Cordy, T. R. Dean, *Robust Multilingual Parsing Using Island Grammars*, CASCON'03

Island grammars

- Detailed production rules for interesting constructs
- Liberal production rules for the rest
  - `~[\.]+ [\.] → Statement`
  - `~[\ \t\n]+ → Water {avoid}`
- Minimal set of assumptions about the overall structure
  - (e.g., a program is a list of statements)

A. van Deursen, T. Kuipers, *Building Documentation Generators*, ICSM 1999.
L. Moonen, * *using Island Grammars*, WCRE 2001, IWPC 2002.

Fuzzy parsing

- Floating islands: no [information about] water
- Complete full lexical analysis
- Perform syntactic analysis of selected parts
  - triggered by anchor symbols
- Inspired (and used) by fact extractors

R. Koppler, *A Systematic Approach to Fuzzy Parsing*, SP&E 27:6, 1997.

Fuzzy parsing

```
declare function local:mccabe($w) {
1+count($w//if) +count($w//evaluate/when)
-count($w//evaluate/when[contains(@unparsed,"OTHER")])
+count($w//perform[contains(@unparsed,"TIMES")])
+count($w//perform[contains(@unparsed,"UNTIL")])
+count($w//search/when) +count($w//search/end)};

let $doc := doc("portfolio.xml") return <results>
{for $section in $doc//section, $para in $section/paragraph
let $cc := local:mccabe($para) where $cc>20
return <component>
<section> {data($section/@name)}</section>
<paragraph> {data($para/@label-name)} </paragraph>
<cc>{$cc} </cc>
</component> </results>
```

V. Zaytsev, *[Framework for] Using XQuery to Measure IT Portfolio Codebases,*
rejected from GTTSE 2007 & ICPC 2008.

Iterative lexical analysis

- Straightforward shortest pattern matching
- `{.*} → Block`
- Bottom-up language engineering
- Several levels of matching:
  - from “simple matches” (1) and “short matches” (2)
  - to “good guesses” (7) and “desperation” (8)
- Enables syntactic analysis of irregular code

A. Cox, C. Clarke, *Syntactic Approximation Using Iterative Lexical Analysis*, IWPC 2003

Hierarchical lexical analysis

- No syntactic constraints
- Works well for conceptual source models
- Even across languages
- Definition example:
  - `[ <type> ] <functionName> \( [ { <formalArg> }+ ] \) [ { <type> <argDecl> ; }+ ] \{`

G. Murphy, D. Notkin. *Lightweight Source Model Extraction,* FSE 1995.

Lexical analysis

```
grep " [0-9][0-9]*[A-Z0-9\-] * PRODCODE" *

grep "MOVE *[A-Z0-9\-]*PRODCODE" *

perl -pi -w -e 's/SEN/SWAT/i;' *

if ($current_line =~ /(MOVE|SET|IF|...)/)
        {...}
```

A. S. Klusener, R. Lämmel, C. Verhoef, *Architectural Modifications to Deployed Software*, SCP 54, 2005

To summarise

http://commons.wikimedia.org/wiki/File:Torii_kiyoshige_bando_hikosaburo_ii.jpg

one scale of tolerance?

To summarise

- Lexical analysis
- Hierarchical lexical analysis
- Iterative lexical analysis
- Fuzzy parsing
- Island grammars
- Robust parsing
- Bridge grammars
- Skeleton grammars
- Parsing in the cloud
- Error productions
- Hierarchical error repair
- Permissive grammars
- Panic mode
- Precise parsing

Stay tuned!

vadim@grammarware.net