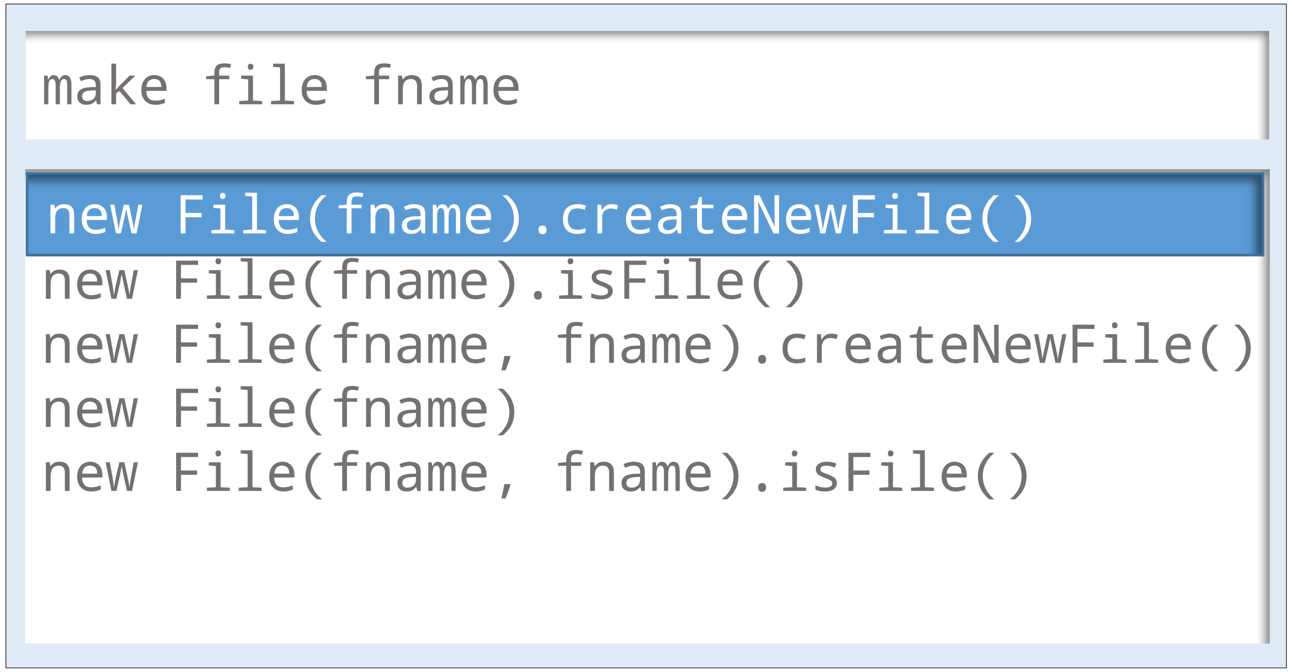
Synthesizing **Java** Expressions from Free-Form **Queries**

Tihomir Gvero
Viktor Kuncak

EPFL, Switzerland

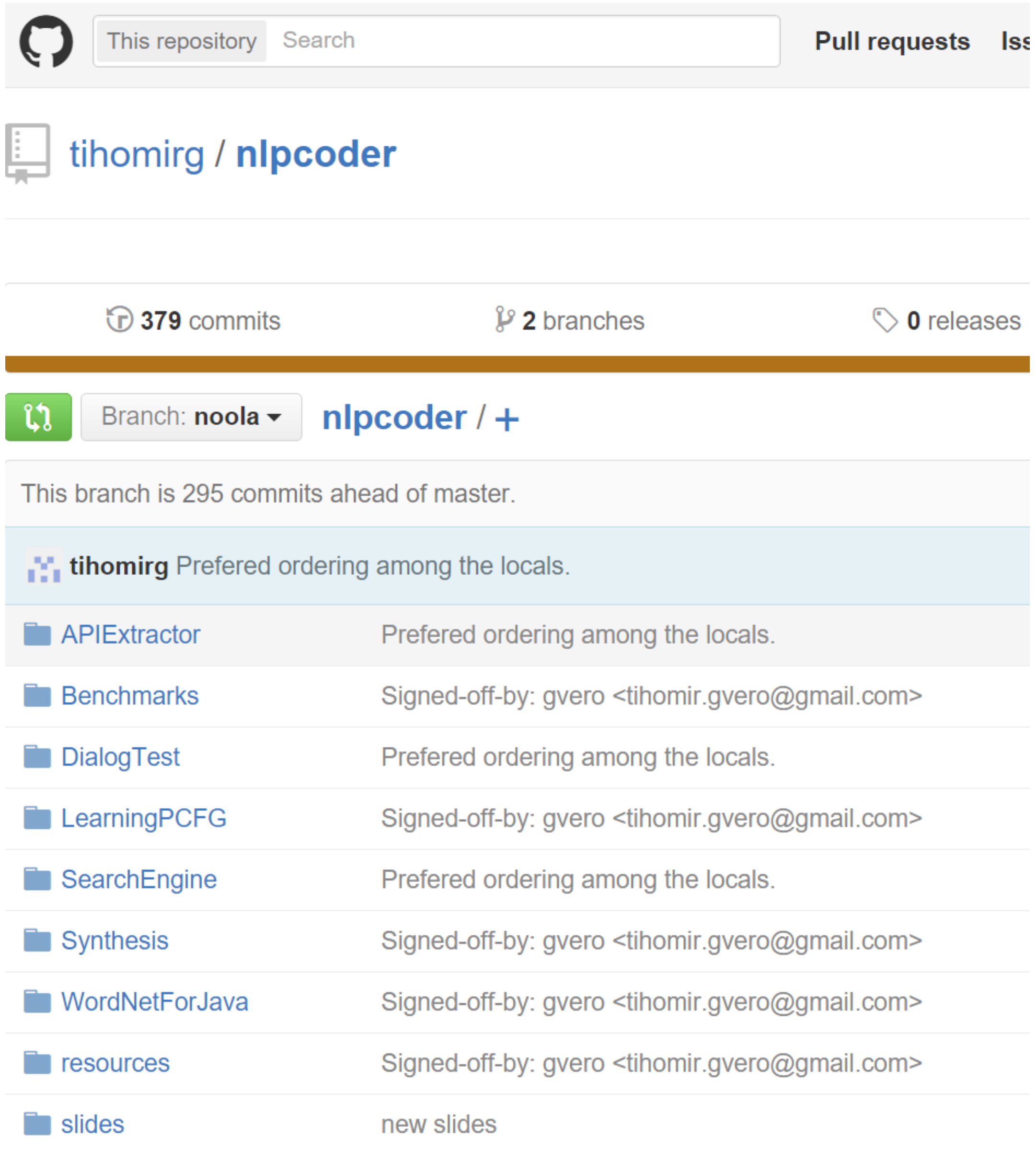


```
make file fname
```

```
new File(fname).createNewFile()
new File(fname).isFile()
new File(fname, fname).createNewFile()
new File(fname)
new File(fname, fname).isFile()
```

Tihomir Gvero

- Did all the work!
- Is not able to attend



tihomirg / **nlpcoder**

379 commits **2** branches **0** releases

Branch: **noola** nlpcoder / +

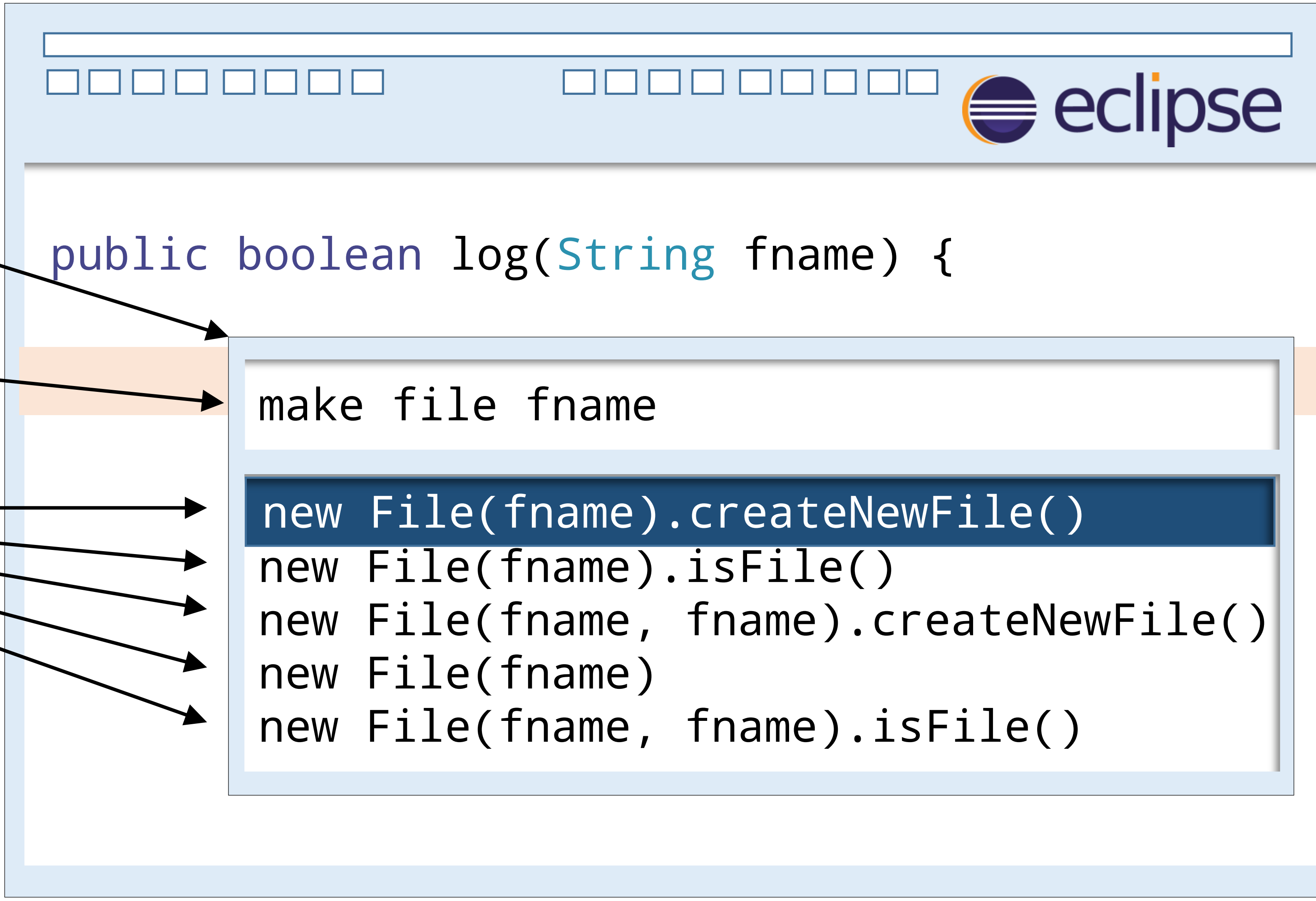
This branch is 295 commits ahead of master.

tihomirg Prefered ordering among the locals.

APIExtractor	Prefered ordering among the locals.
Benchmarks	Signed-off-by: gvero <tihomir.gvero@gmail.com>
DialogTest	Prefered ordering among the locals.
LearningPCFG	Signed-off-by: gvero <tihomir.gvero@gmail.com>
SearchEngine	Prefered ordering among the locals.
Synthesis	Signed-off-by: gvero <tihomir.gvero@gmail.com>
WordNetForJava	Signed-off-by: gvero <tihomir.gvero@gmail.com>
resources	Signed-off-by: gvero <tihomir.gvero@gmail.com>
slides	new slides

anyCode tool on GitHub

Synthesizing **Java** expressions from free-form **queries**



```
public boolean log(String fname) {
```

```
make file fname
```

```
new File(fname).createNewFile()
new File(fname).isFile()
new File(fname, fname).createNewFile()
new File(fname)
new File(fname, fname).isFile()
```

completion box

query
English + identifiers

Java Expressions
- query relevant
- type & scope correct
- statistically likely, yet not copy-pasted

Examples of results that anyCode gives

load class ”MyClass.class”	`Thread.currentThread()`
	`      .getContextClassLoader()`
	`      .loadClass(”MyClass.class”)`

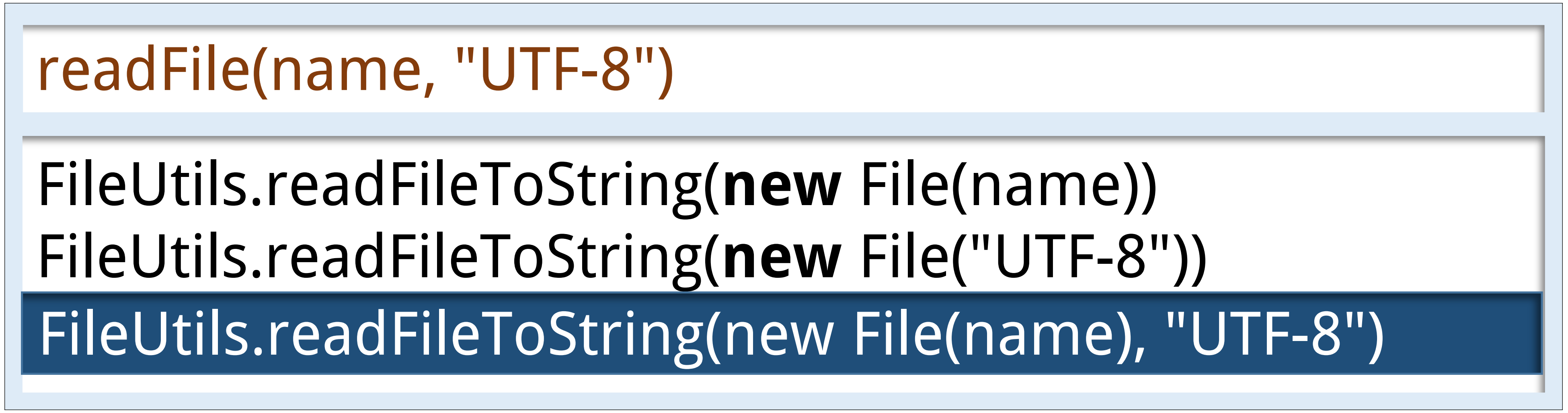
write ”hello” to file ”text.txt”	`FileUtils.writeStringToFile(`
	`     new File(”text.txt”), ”hello”)`

new buffered stream ”text.txt”	`new BufferedReader(`
	`  new InputStreamReader(`
	`  new BufferedInputStream(`
	`  new FileInputStream(”text.txt”))))`

set thread max priority	`Thread.currentThread()`
	`  .setPriority(Thread.MAX_PRIORITY)`

Can also help correct “sloppy Java”

```
public String prepareMessage(String name, String protocol)
                                                throws Exception {
    if (!protocol.equals("file"))
        return errorMessage(protocol);
    else
        return
}
```



```
readFile(name, "UTF-8")
```

```
FileUtils.readFileToString(new File(name))
FileUtils.readFileToString(new File("UTF-8"))
FileUtils.readFileToString(new File(name), "UTF-8")
```

How?

Translation problem

`make file fname` → `new File(fname).createNewFile()`

English queries:

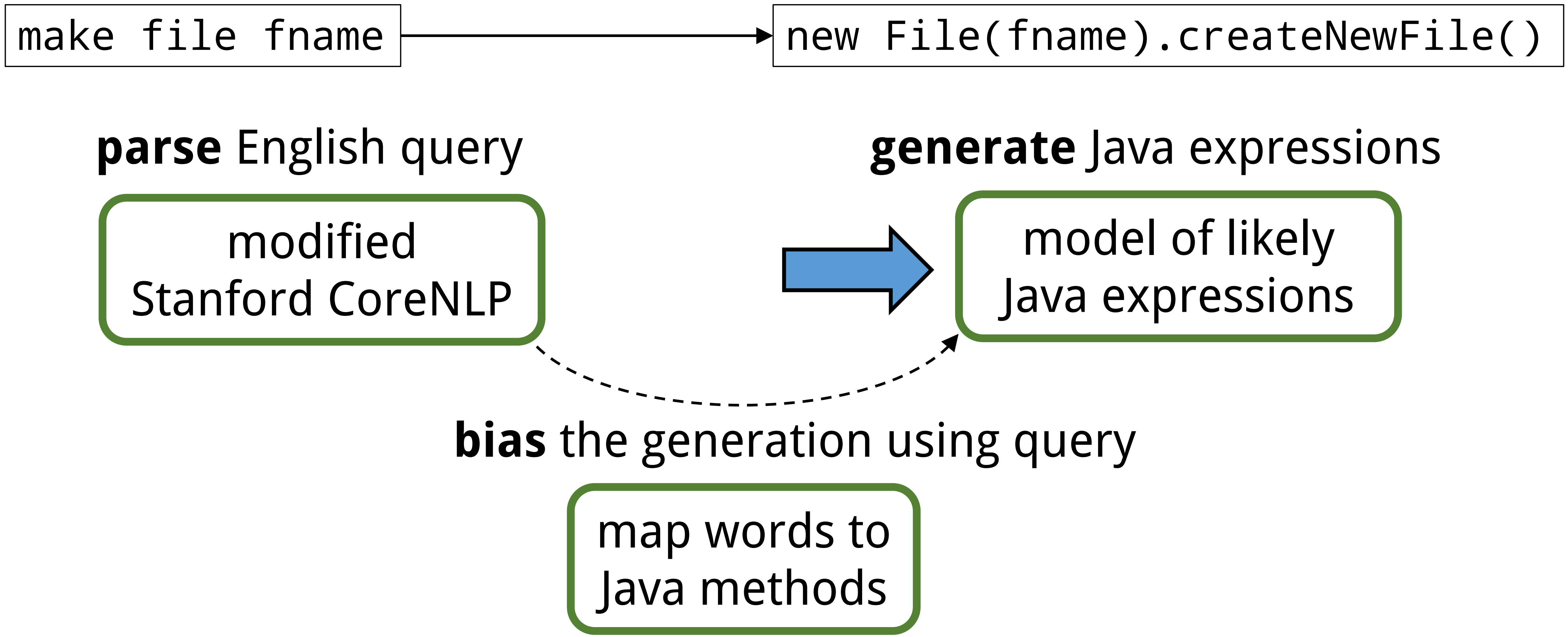
- English phrase structures
- English dictionary words
- identifiers in scope
- literals, e.g. 42 or “Hello”

Java expressions:

- scoping and type rules of Java
- API method names `camelCase`
- identifiers in scope
- literals, e.g. 42 or “Hello”

No readily available large-scale parallel corpus, unlike machine translation.

Key tasks in translation


make file fname
new File(fname).createNewFile()
parse English query
modified Stanford CoreNLP
generate Java expressions
model of likely Java expressions
bias the generation using query
map words to Java methods

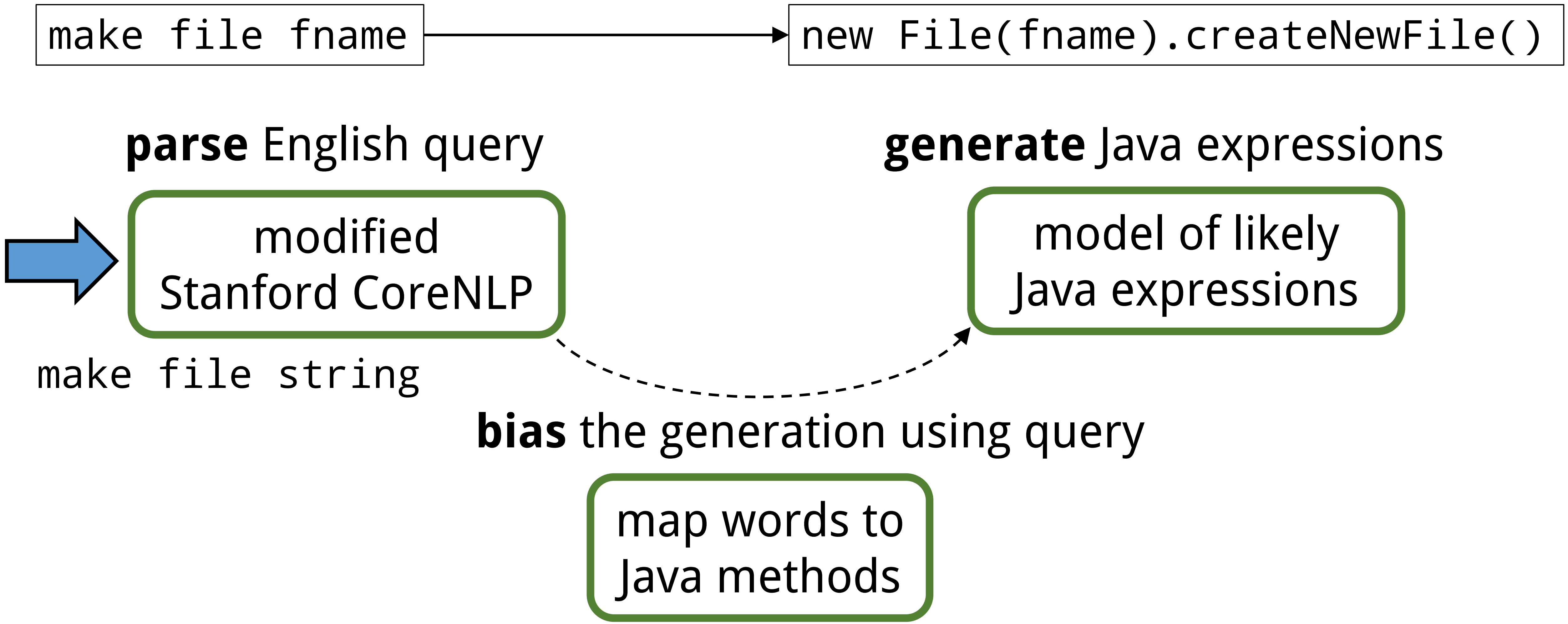
Which Java expressions do IDEs dream about?

```
Thread.currentThread()
        .setPriority(Thread.MIN PRIORITY)
```

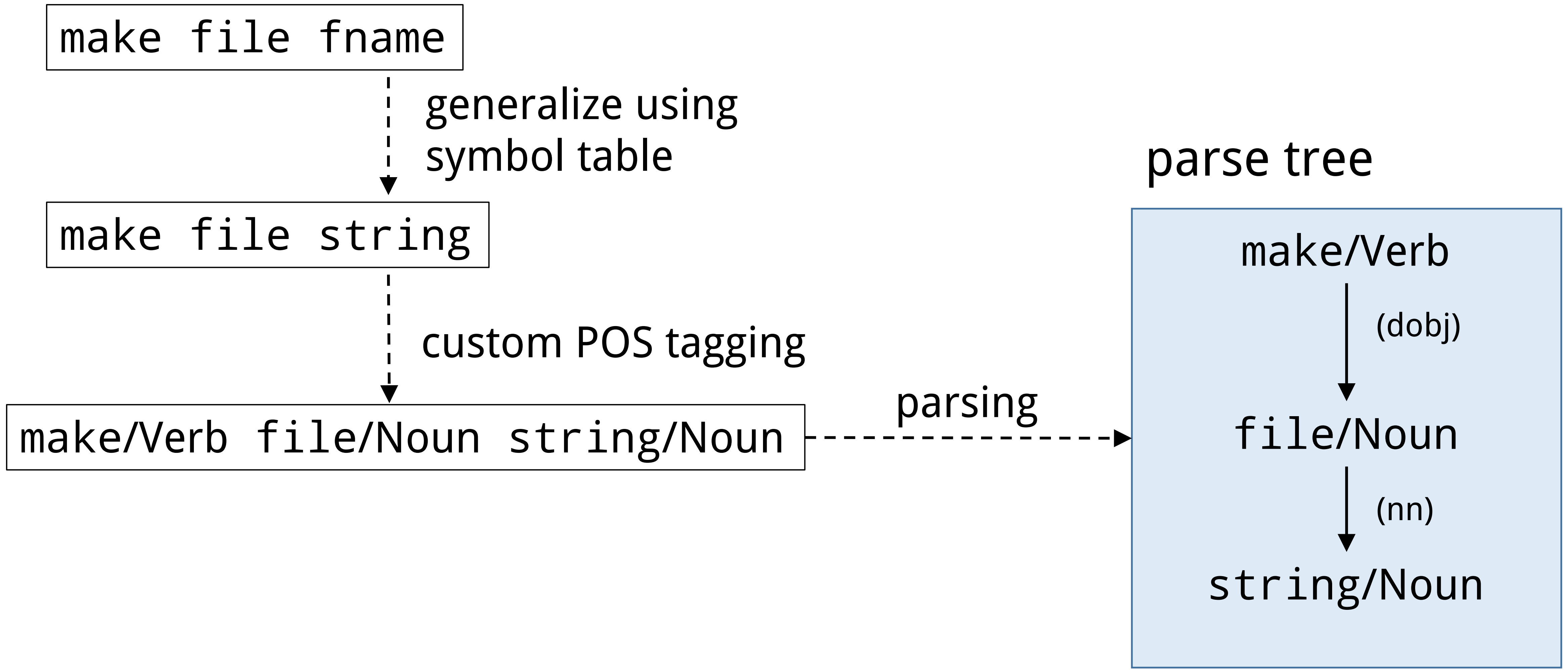
Distribution over all Java expressions

- Our prior work: declaration frequencies only (Gvero et al. PLDI’13)
- This work: computes additionally probabilistic context-free grammar (**PCFG**) describing likely composition of declarations
    - parse and type check 14’000 Java projects (~2M files)
    - extract PCFG from expressions, built after copy propagation on the files
    - splits Java types according to methods that return them
- Pr(expression) = product of Pr of rules used to build it
- Our model can be used for various program synthesis tasks
    - avoids bizarre solutions for highly underspecified queries
- Here: it gives baseline expression probability, in absence of a query
    - machine translation terminology: model for the target language

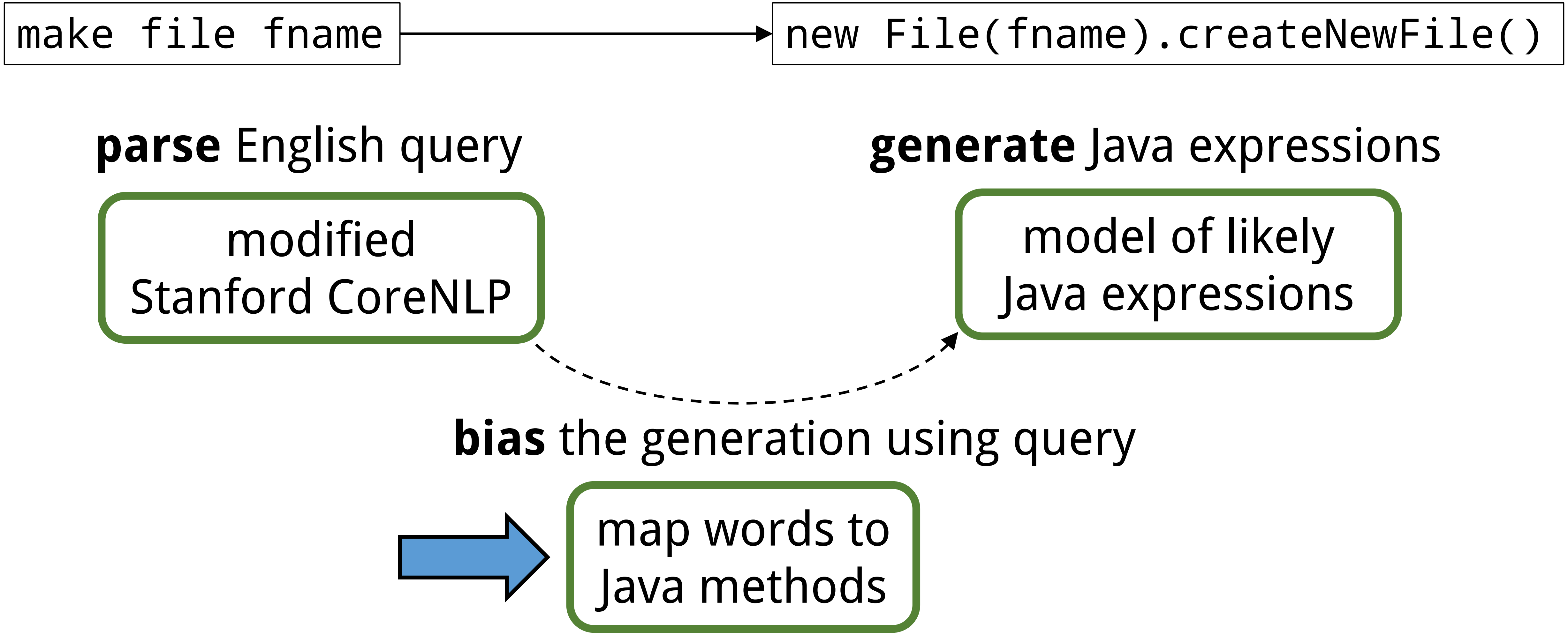
Key tasks in translation


make file fname
new File(fname).createNewFile()
parse English query
modified Stanford CoreNLP
make file string
generate Java expressions
model of likely Java expressions
bias the generation using query
map words to Java methods

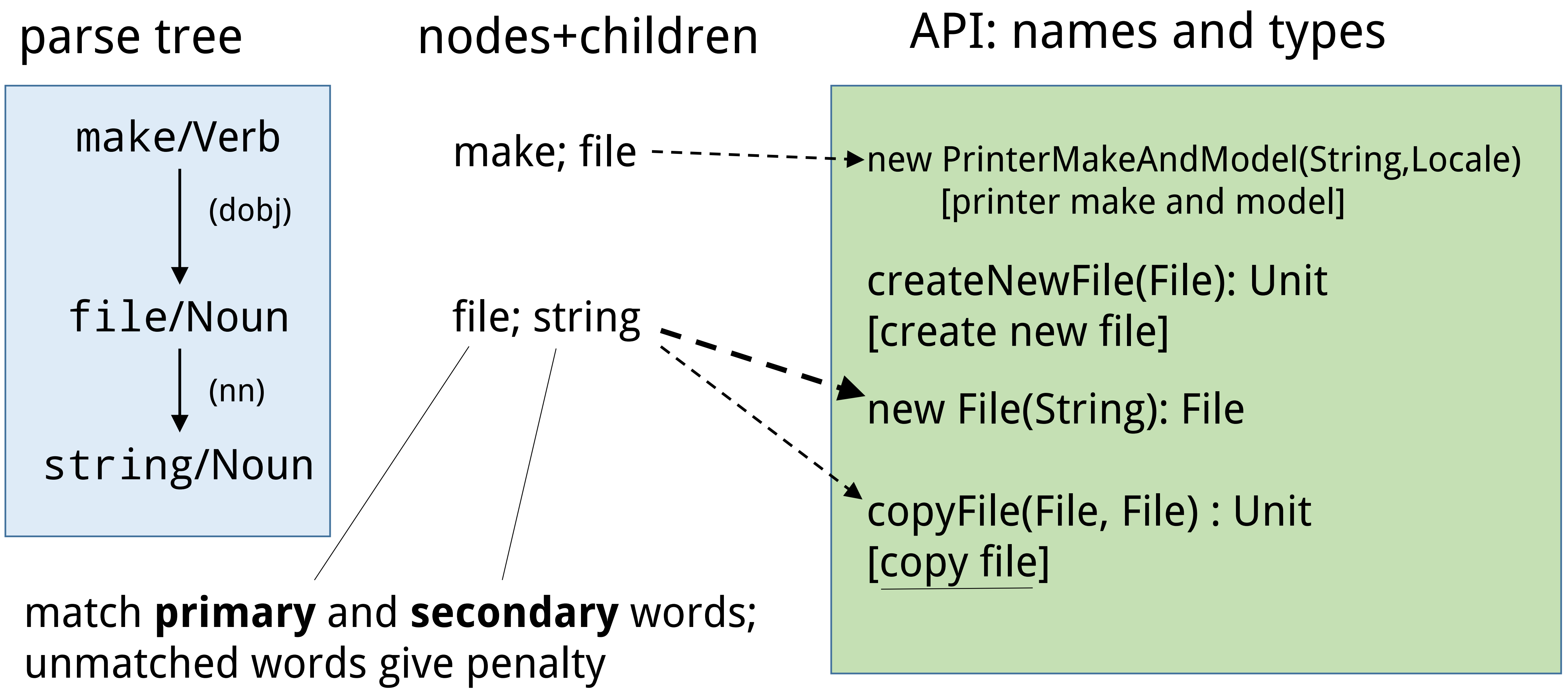
Parsing using modified CoreNLP toolkit



Key tasks in translation

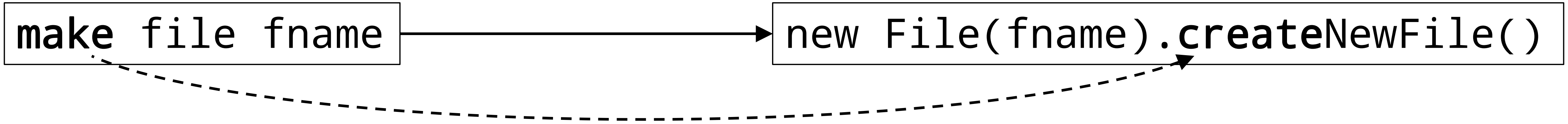

make file fname
new File(fname).createNewFile()
parse English query
modified Stanford CoreNLP
generate Java expressions
model of likely Java expressions
bias the generation using query
map words to Java methods

Map groups from parse tree to declarations



match **primary** and **secondary** words;
unmatched words give penalty

Supporting related words

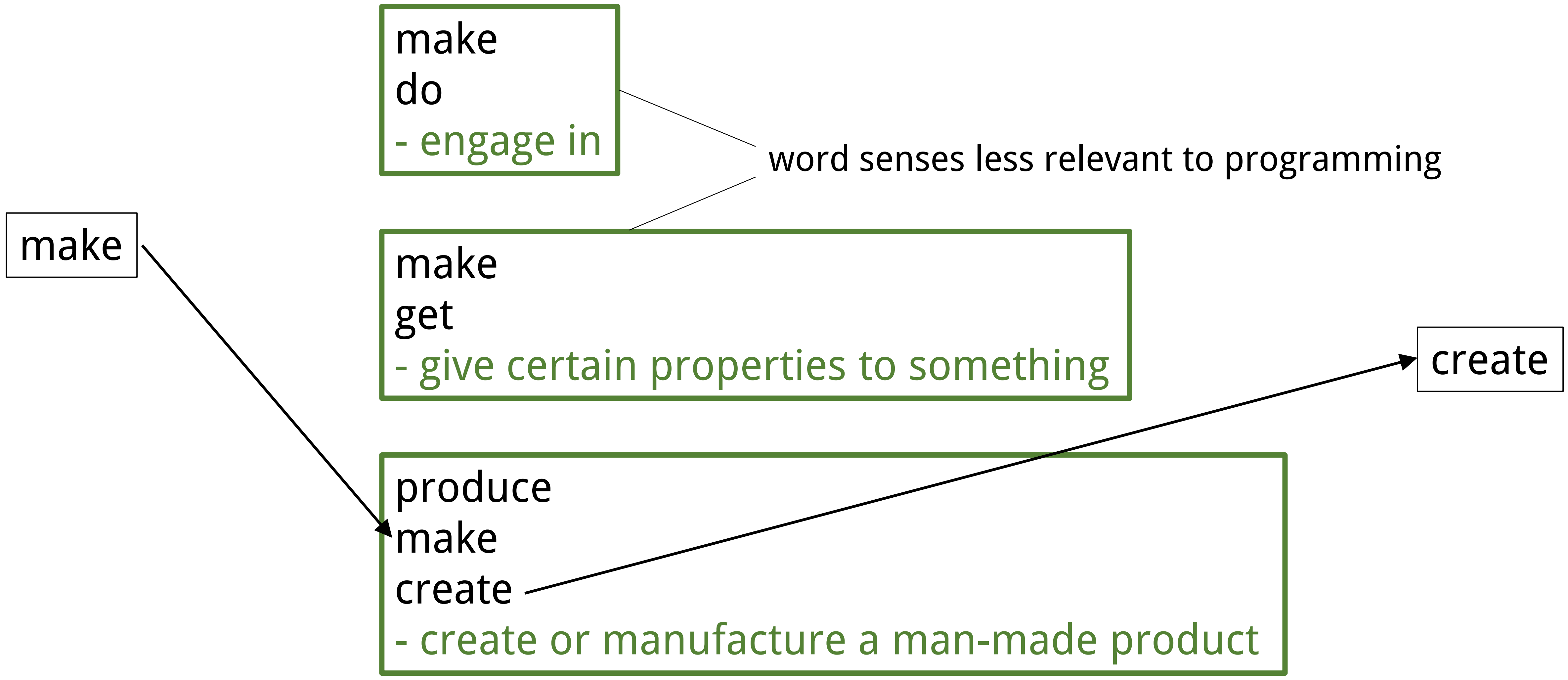


```
make file fname
```

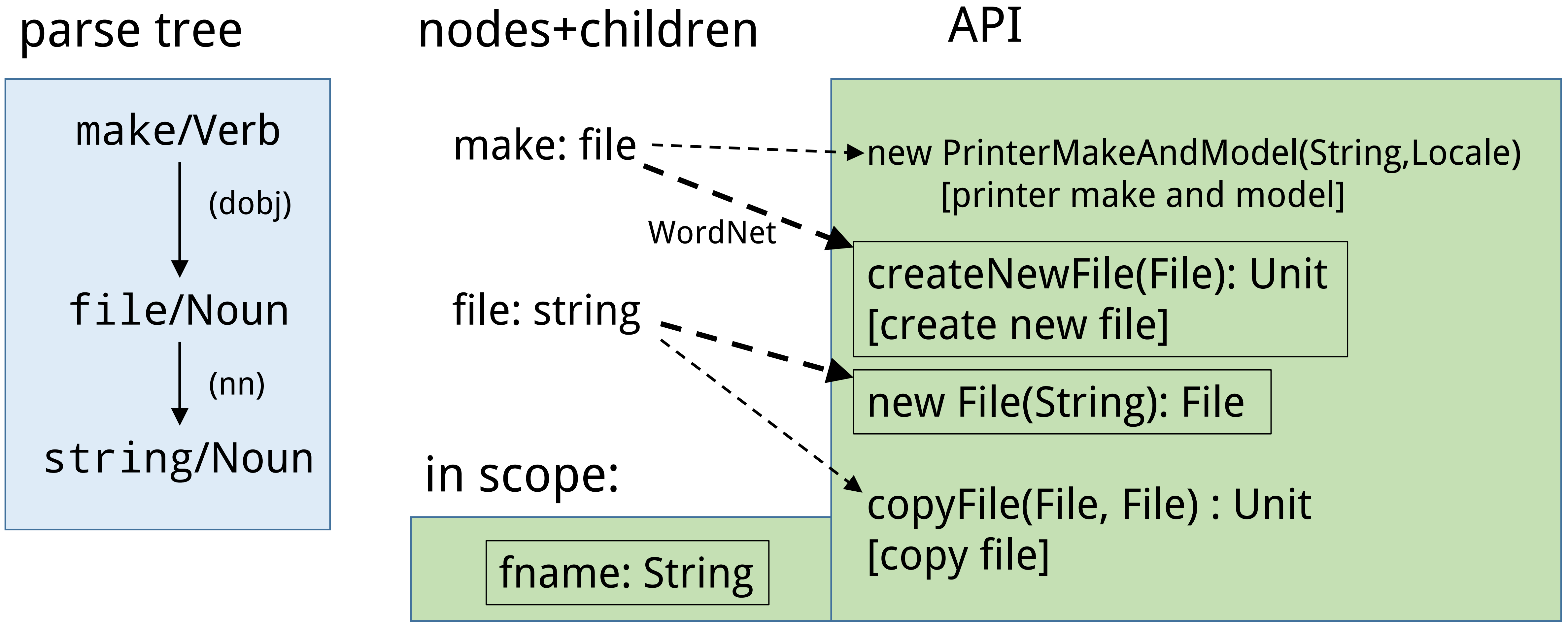
```
new File(fname).createNewFile()
```

- Approach so far would not support e.g. synonyms
- We therefore use WordNet (https://wordnet.princeton.edu/)
  - Groups words into sets of synonyms (synsets)
  - Each word may belong to multiple synsets (meanings of a word)
  - Relationships between synsets, such as "is-a"
  - Synsets have English descriptions, as in a dictionary
- When computing if words are related, we favor those synsets whose description uses API words – specialize to jargon of programming

Related words through WordNet synsets

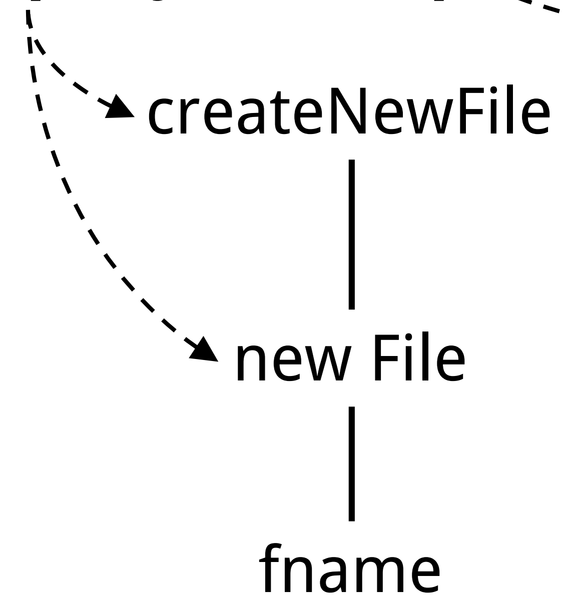

make
do
- engage in
word senses less relevant to programming
make
make
get
- give certain properties to something
create
produce
make
create
- create or manufacture a man-made product

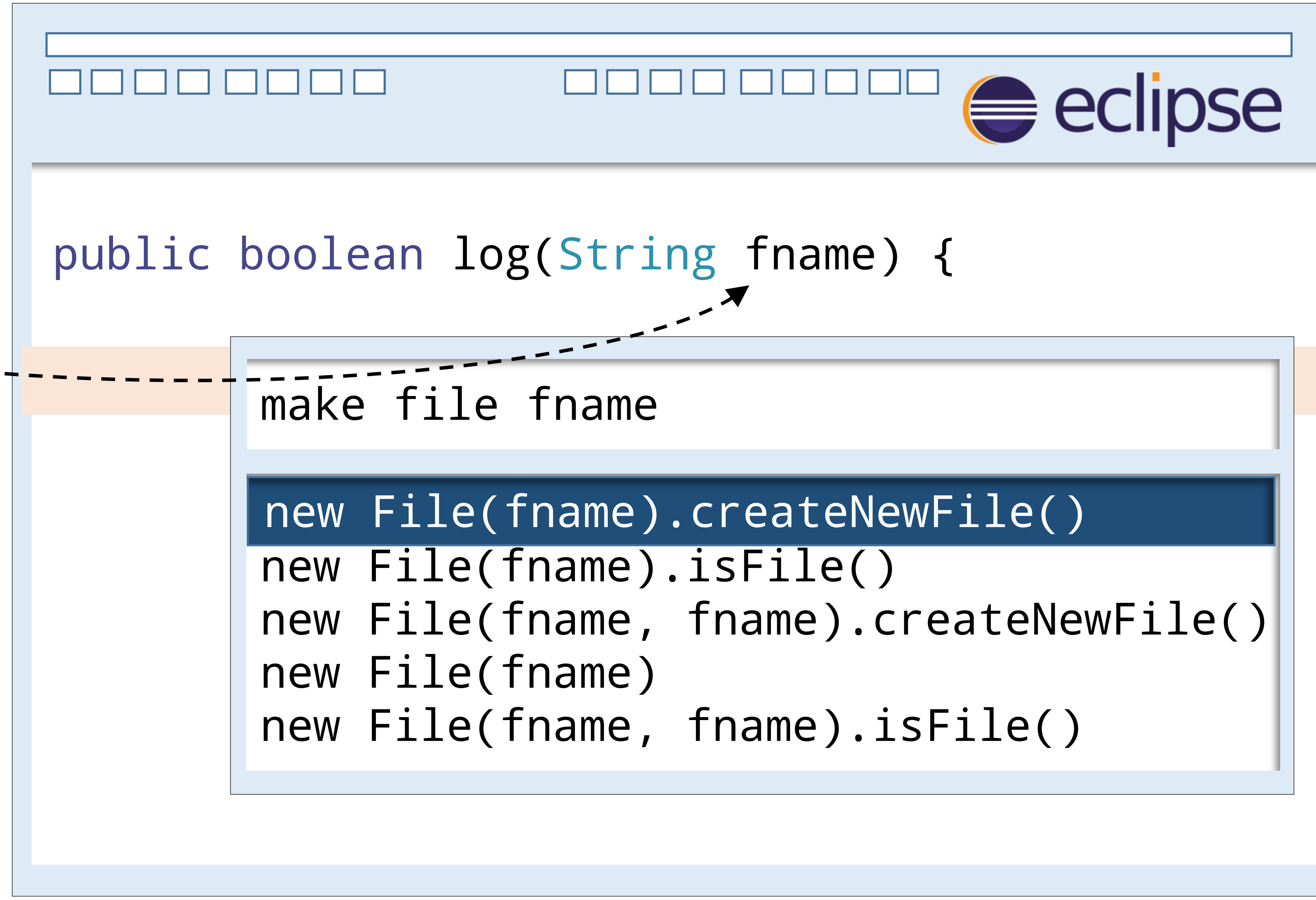
Map groups from parse tree to declarations



Combining declarations into expression

Find most likely word from a new PCFG:
PCFG for Java
extended with bias from:
query and **scope**





```
public boolean log(String fname) {
```

```
make file fname
```

```
new File(fname).createNewFile()
new File(fname).isFile()
new File(fname, fname).createNewFile()
new File(fname)
new File(fname, fname).isFile()
```

Parameters and tuning

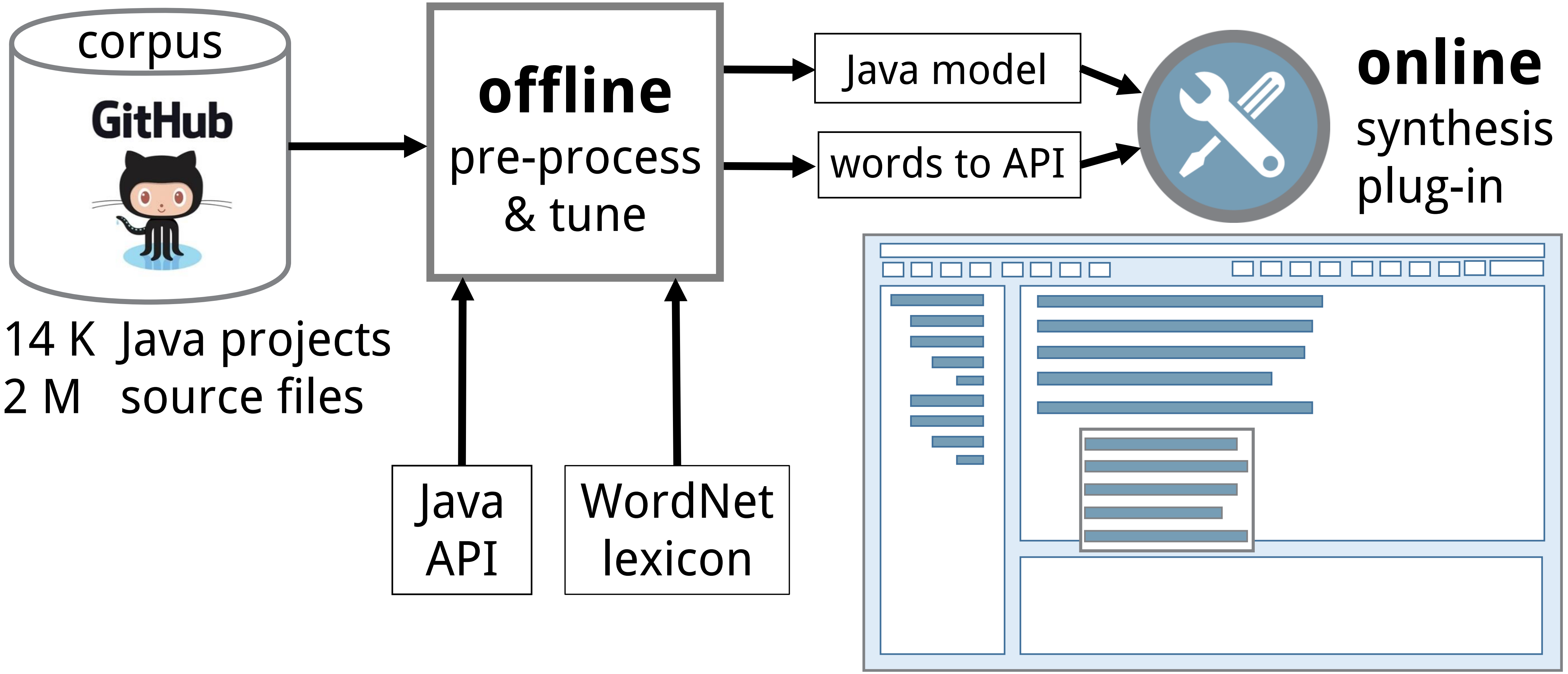
Parameters determine relative strength of different criteria

- matching of words to declarations: primary vs secondary words
- weights derived from corpus vs identifiers in scope
- order of parameters in input vs output – penalize inversion
- repetition of input elements undesired

A small number of parameters, <10

- system works even with our “best guess” values of parameters
- tuning: make it work better, by finding locally optimal values
- use local search, cost function as black box (discretize space)

Outline of our system



How well?

Evaluation

- We wrote a set of 90 (query, Java) pairs – all are shown in paper
- We split them in two parts:
  - 45 used for tuning relative weights of different aspects of translation
    45 used to evaluate the final system
- Results
  - in 82% cases: the desired expression found and ranked in top 10
  - in 20% of those cases: the expression ranked #1
  - running times 0.001 to 0.219 seconds, average 0.06 s
- Turning off PCFG brings success rate from 82% down to only 27%

Selected related work

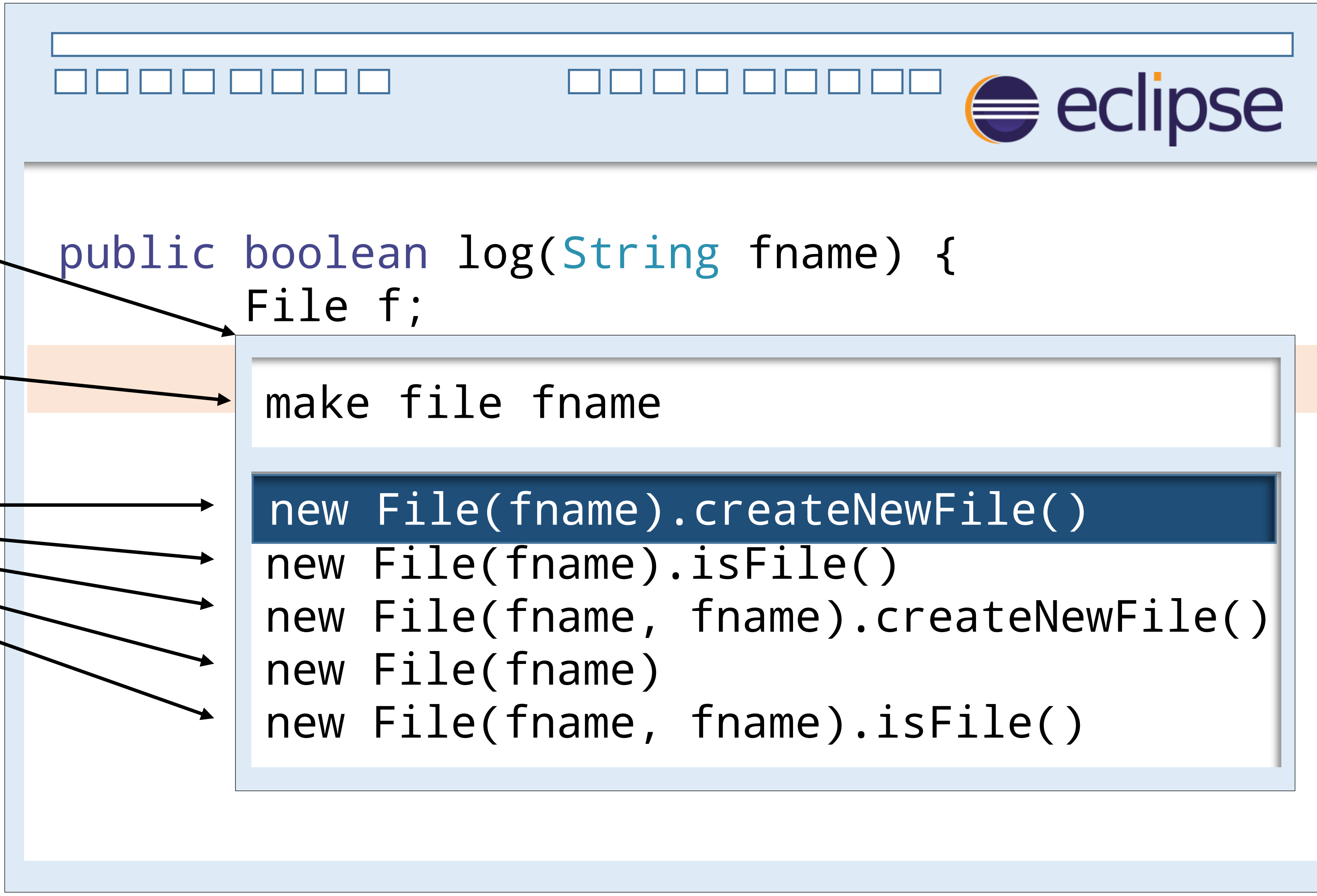
- G. Little and R. C. Miller: Keyword programming in Java (ASE ‘07)
  - Translates small number of *keywords* into a valid expression (no corpus)
- D. Price, E. Riloff, J. L. Zachary, and B. Harvey: NaturalJava (IUI ‘00)
  - Translation from a *restricted form of NL description* to Java edit statements
- V. Le, S. Gulwani, and Z. Su: SmartSynth (MobiSys ‘13)
  - Generates *smartphone automation scripts* from NL descriptions (bag of w.)
- A. Cozzie and S. T. King: Macho (TR ‘12)
  - Transforms NL descriptions into simple programs
  - Uses input-output *examples*
- V. Raychev, M. T. Vechev, and E. Yahav: SLANG (PLDI ‘14)
  - Uses N-gram language model to complete *holes in the program*

anyCode: a new point in the space

completion box

query
English + identifiers

Java Expressions
- query relevant
- type & scope correct
- statistically likely, yet not copy-pasted



```
public boolean log(String fname) {
        File f;
```

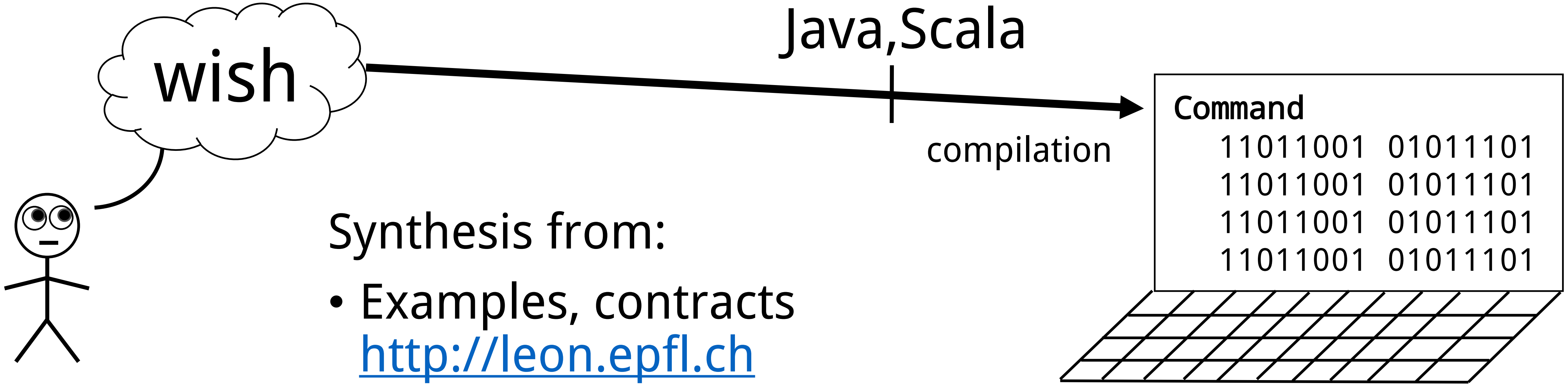
```
make file fname
```

```
new File(fname).createNewFile()
new File(fname).isFile()
new File(fname, fname).createNewFile()
new File(fname)
new File(fname, fname).isFile()
```

Some limitations

- Source: analyzes English no better than Stanford CoreNLP toolkit
  - no semantic analysis (like most NLP tools)
  - question of ontologies for programming tasks is wide open
- Translation
  - uses only one source syntax parse trees
  - only extracts sub-trees of height one, ignoring deeper nesting structure
  - relies on names in program being in English, as for API
- Target: use primarily the PCFG to guide synthesis
  - no use of input-output examples
  - no static analysis of e.g. method sequences

Agenda: “Your wish is my command”



Synthesis from:

- Examples, contracts
  http://leon.epfl.ch
- Natural language & corpus
  - this line of work

Two unsolvable problems put together?

Is it methodologically reasonable to try to solve at once both

- processing of natural language
- synthesis from specifications that have multiple solutions

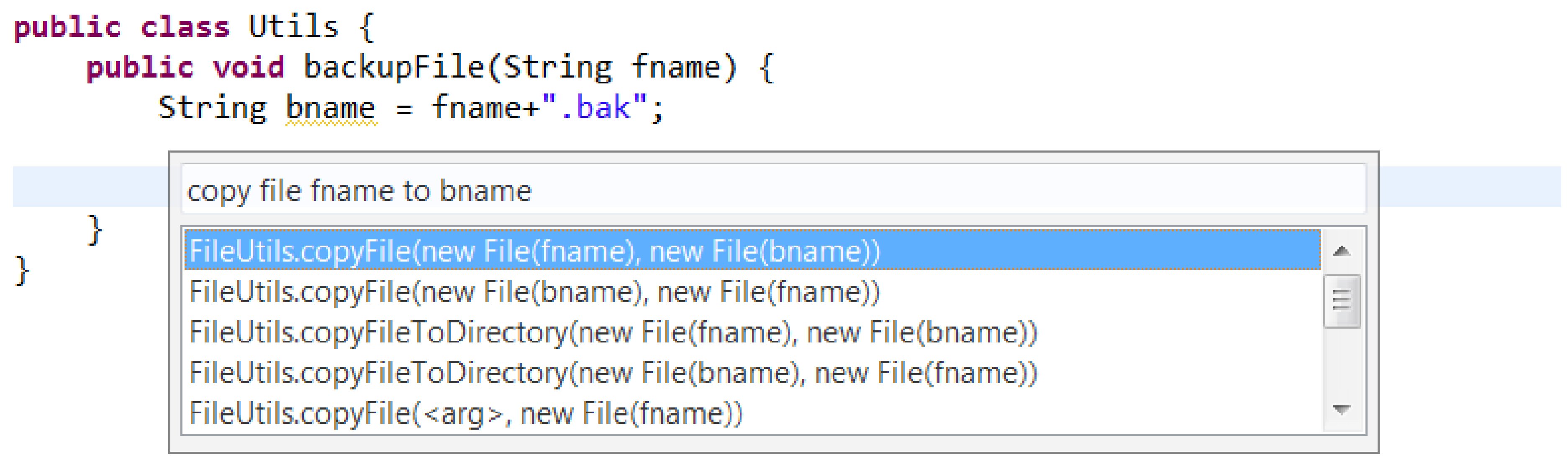
We claim: yes, they work well together

- natural language introduces multiple interpretations;
  synthesis can handle this ambiguity
- range of applications for synthesis is greater if we can avoid formal specifications in favor of English

Mapping English to code is feasible.

We need more research in this area!

Questions…

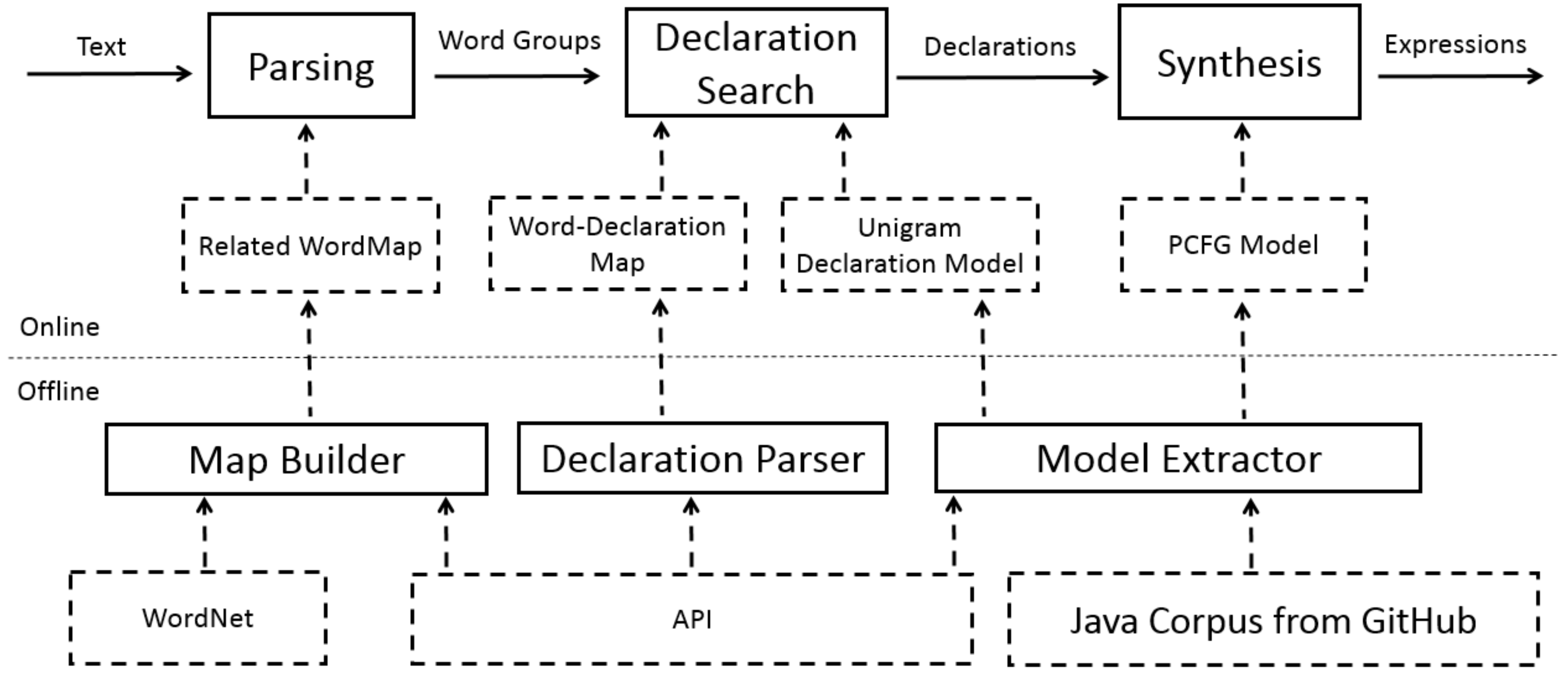


```java
public class Utils {
    public void backupFile(String fname) {
        String bname = fname+".bak";

    }
}
```

copy file fname to bname

```
FileUtils.copyFile(new File(fname), new File(bname))
FileUtils.copyFile(new File(bname), new File(fname))
FileUtils.copyFileToDirectory(new File(fname), new File(bname))
FileUtils.copyFileToDirectory(new File(bname), new File(fname))
FileUtils.copyFile(<arg>, new File(fname))
```



Corpus and Ranking

- Corpus:
  - Over 14’000 Java projects from GitHub (near 2 million source files)
- Declaration score

  + Frequency

  + Number of hit words

  – Number of missed words
- Expression score

  + PCFG score

  + Declaration scores

  + Input coverage score

  – Repetition score