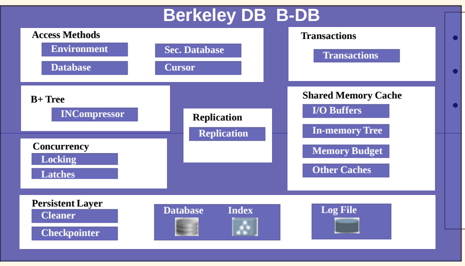
Preprocessing without some pitfalls

S. Jarzabek[1], Y. Xue[1], N. Shaikh[1], H. Zhang[2] and Y. Lee[2]

[1]National University of Singapore [2]Tsinghua University

- Preprocessing has well-known pitfalls
- Still, it is commonly used to mange software system variants
- So here are small enhancements that make preprocessing easier to use
- Limitations of our technique and potentials for drastic improvements

1. Example: Berkeley DB (B-DB)



- B-DB: a Java database engine
- 232 B-DB base source files
- Any of 38 optional features can be implemented in B-DB variants:
  IO, MemoryBudget, Evictor, CheckSum, Statistics, …

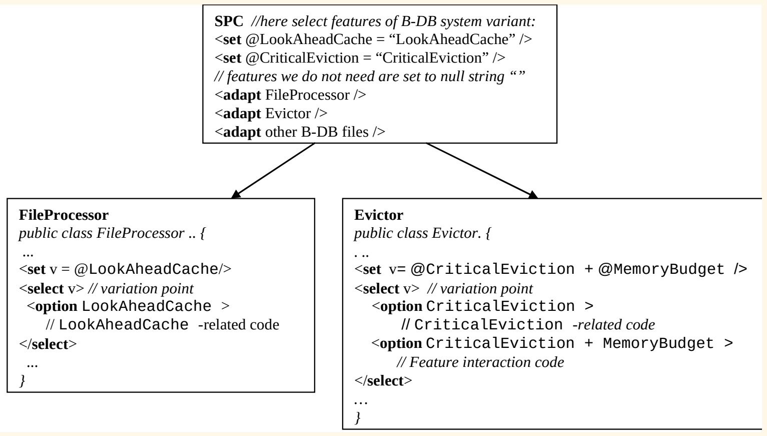
B-DB is a Product Line with many B-DB system variants!

The impact of features on B-DB base files:

Feature	# base files affected	# points
`MemoryBudget`	32	190
`Evictor`	12	28
`CheckSum`	10	28
`Statistics`	10	34
`CheckPointer`	5	34
`CpByteConfig`	4	6
`CpTimeConfig`	4	7

Feature	Interacting feature	# points
`CheckPointer`	Statistics	22
`MemoryBudget`	Evictor	5
`MemoryBudget`	CriticalEviction	1
`SyncIO`	IO	4
`EvictorDaemon`	Evictor	3

2. Managing B-DB variants with preprocessing



```
SPC  //here select features of B-DB system variant:
<set @LookAheadCache = “LookAheadCache” />
<set @CriticalEviction = “CriticalEviction” />
// features we do not need are set to null string “”
<adapt FileProcessor />
<adapt Evictor />
<adapt other B-DB files />
```

```
FileProcessor
public class FileProcessor .. {
 ...
<set v = @LookAheadCache/>
<select v> // variation point
  <option LookAheadCache >
     // LookAheadCache  -related code
</select>
  ...
}
```

```
Evictor
public class Evictor. {
. ..
<set  v= @CriticalEviction + @MemoryBudget />
<select v>  // variation point
   <option CriticalEviction >
       // CriticalEviction  -related code
   <option CriticalEviction + MemoryBudget >
       // Feature interaction code
</select>
…
}
```

Concept: Variation points in base files affected by features marked with feature names

Realities of managing system variants:

- The number of optional features may be huge
- Many-to-many mappings between features and base files
  - Each feature affects many base files, at many variation points
  - Each base file is affected by many features
- One feature depends on and interacts with other features

3. Preprocessing pitfalls

Suppose we select features for a custom system or need modify a certain feature:

- Feature code spreads through base files, we need visit all the relevant variation points
- We must understand feature interactions
- Base files heavily instrumented with variation points

What we need to know to reuse/maintain features, and maintain base files?

- How is a feature Evictor implemented?
- Which base files are affected by featureEvictor and at which variation points?
- Which features affect base file FileManager?
- Which features interact with which other features, in which base files and how?

➔ **Difficult to understand base files, difficult to reuse or modify features**

4. Easing pitfalls with feature analysis

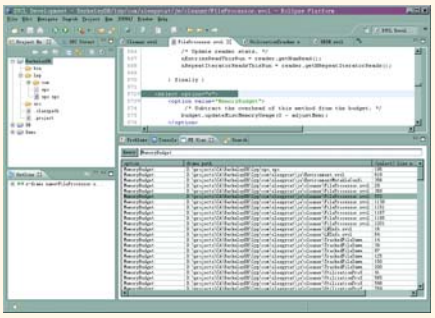
The concept of the solution: Feature Query Language (FQL) to analyze the preprocessing representation (analogous to program analysis), and answer above queries

Query examples:

```
declare base file x; option o;
select x, o
where o.f-names = “*” and Contains
(“Evictor”,o)
```

```
declare base file x; option o;
select x, o
where o.f-names = “*Evictor*” and
Contains (x,o)
```

```
declare option o
select x.name, o
Where o.f-names = ”*CriticalEviction*MemoryBudget*”
      and Contains (x,o)
```



IDE shows query results, assists in finding and analysis of feature code

5.Merits, Limitations and Potentials

- *Novelty*: treat preprocessing representation as first-class representation, rather than add-in to a programming language
- *Merit*: A tool can find feature-related code in various analysis contexts
  - improved readability, maintainability and reusability;
  - the solution can be applied to any preprocessing system, **but**:
- *Limitation*: Inherent complexity of a preprocessing representation remains
  - we do not cure the main problem which is scattering of feature code across base files, at multiple variation points
- Annotations, configuration parameter files, CVS/SVN have similar limitations
- Product Line approach is weak in streamlining and automating customizations
- *Potential*: enhance the solution into full-blown method for system variants
  - Example: XVCL , XML-based Variant Configuration Language