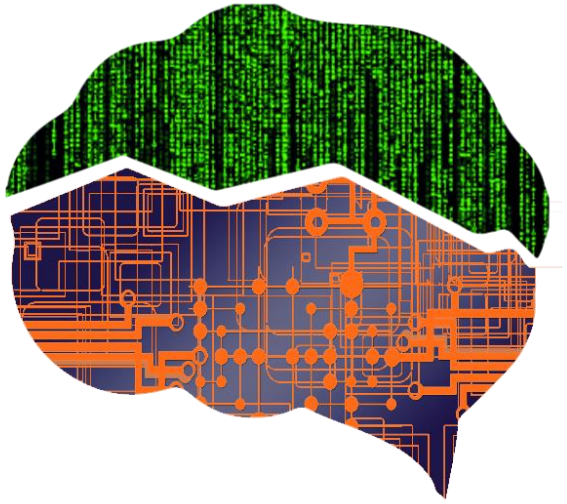
Software Product Line Engineering

Version Control and Build Systems

Christian Kästner (Carnegie Mellon University)
Sven Apel (Universität Passau)
Norbert Siegmund (Bauhaus-Universität Weimar)
Gunter Saake (Universität Magdeburg)



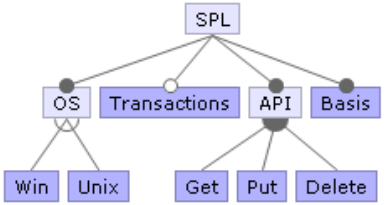
Bauhaus-Universität Weimar

How to implement variability?

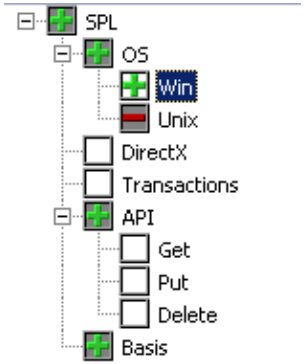
Domain Eng.

Application Eng.

Feature Model

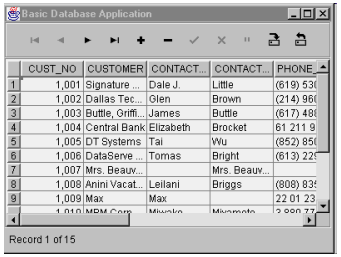


Reusable Implementation artifacts



Feature Selection

Generator



Resulting Program

Recap: Variability at Runtime

- Parameter
  - Global configuration vs. propagating parameters
- Design Patterns
  - Observer (Observer and Subject)
  - Template Method (inheritance)
  - Strategy Pattern
    (Context + Interface for alternative algorithms)
  - Decorator Pattern (Delegation + Interface)

Variability at Compile Time

- Goal: Only actually required source code gets compiled
- Small, optimized variants
- Source code gets selected and composed

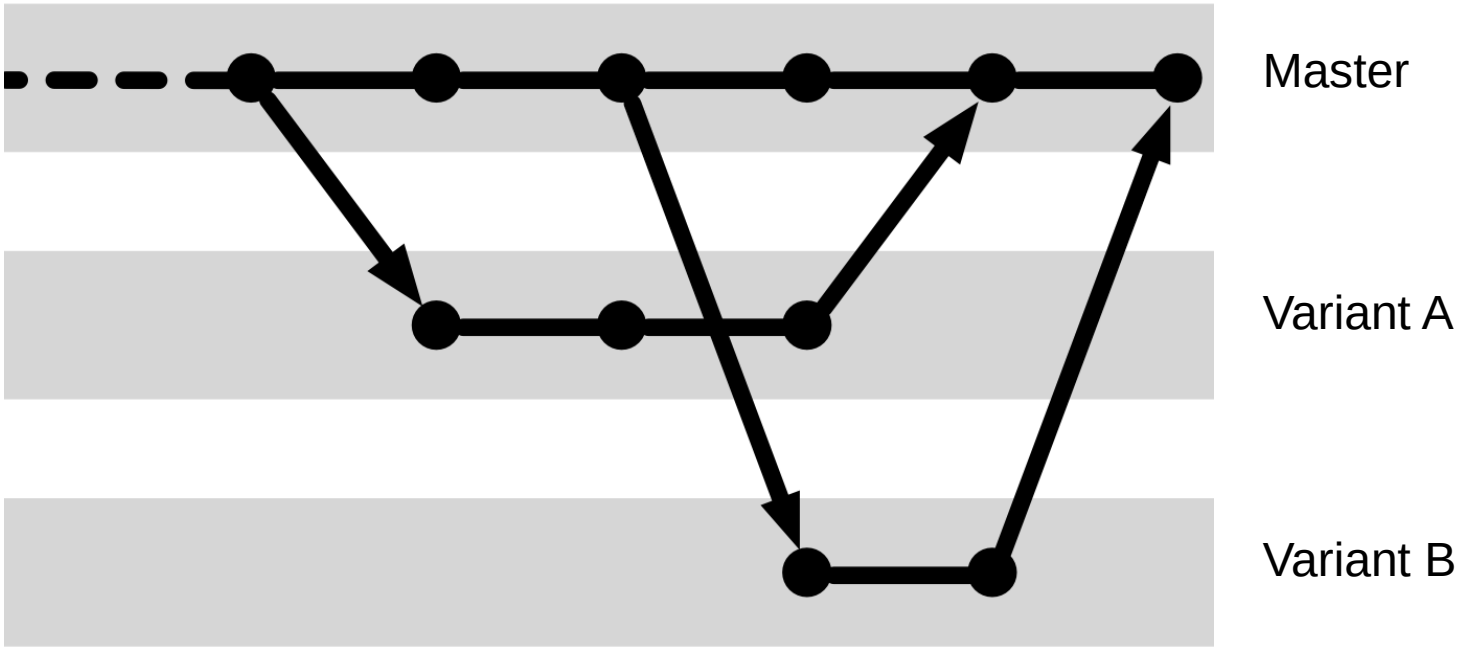
- How to implemented optional and alternative code?
- Here: simple methods for few variants

Version Control Systems

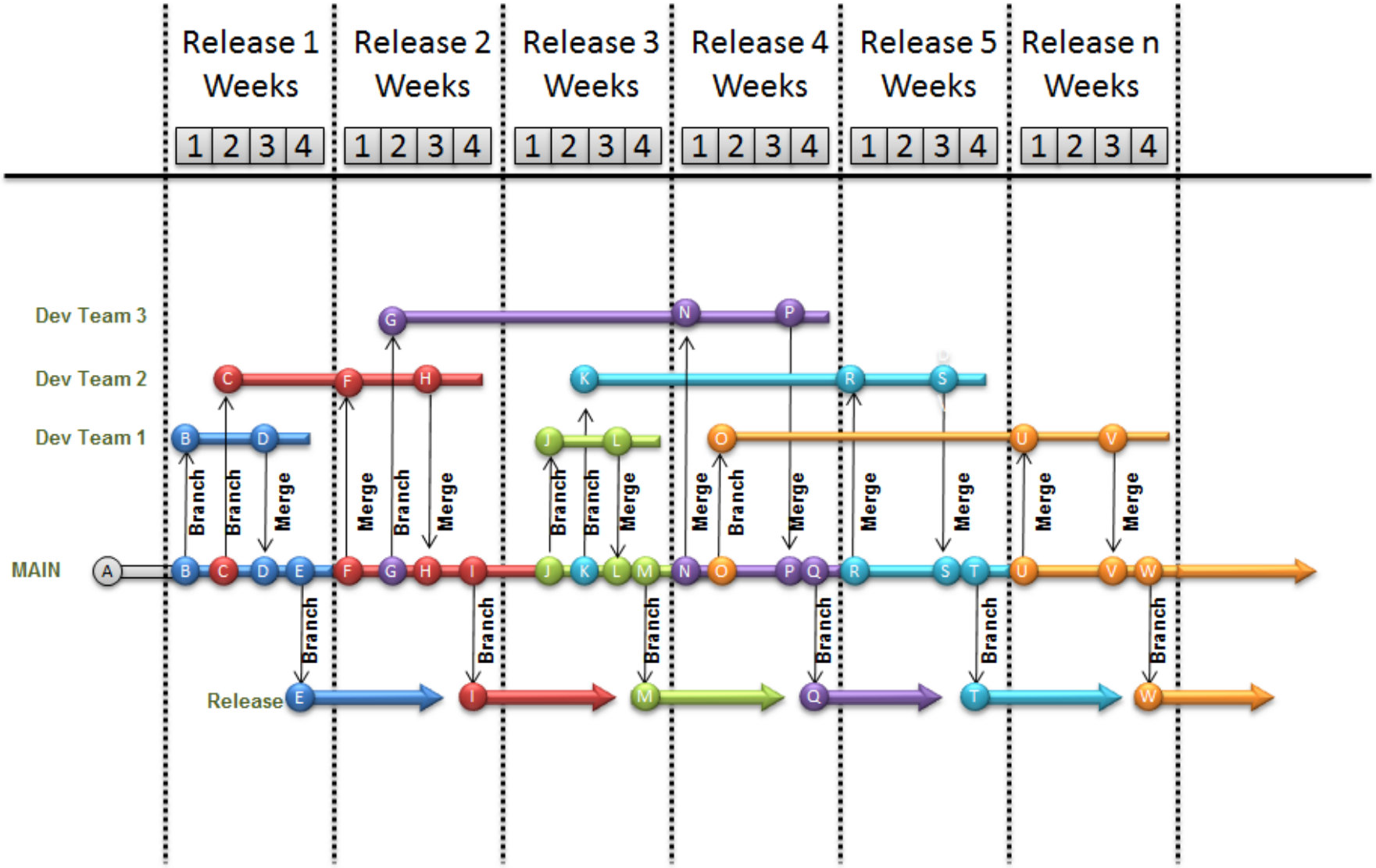
Version Control System

- Part of the configuration management
- Versioning of source-code files
- Shared development
- Archive of old code versions
  - Time stamp and user id
  - Changes as deltas
- Process: Checkout – Change – Commit – Update – Change – Commit - …
- Examples: CVS, SVN, Visual SourceSafe, Perforce, SCCS, Git, Mercurial

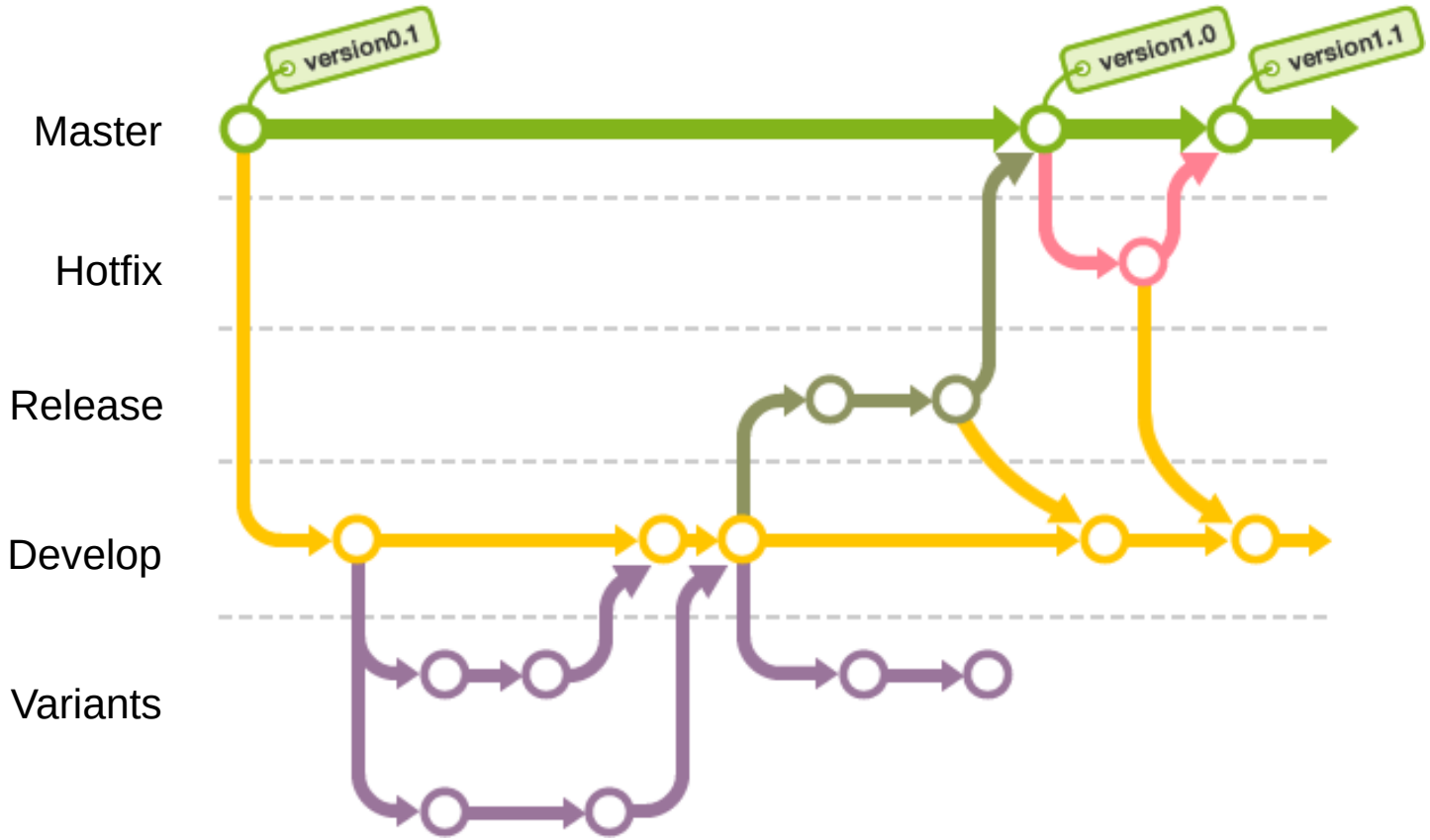
Branching & Merging



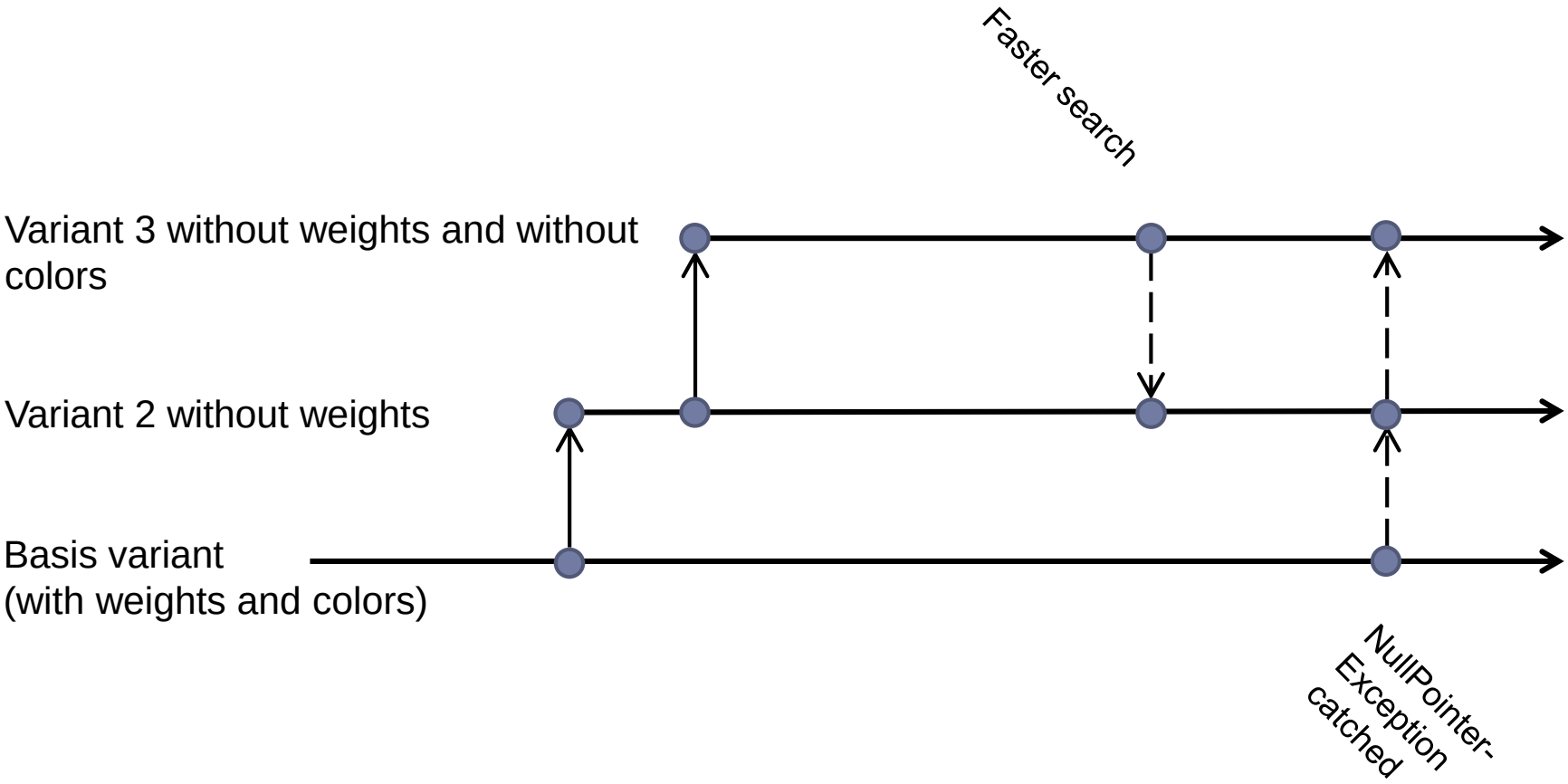
Parallel Development



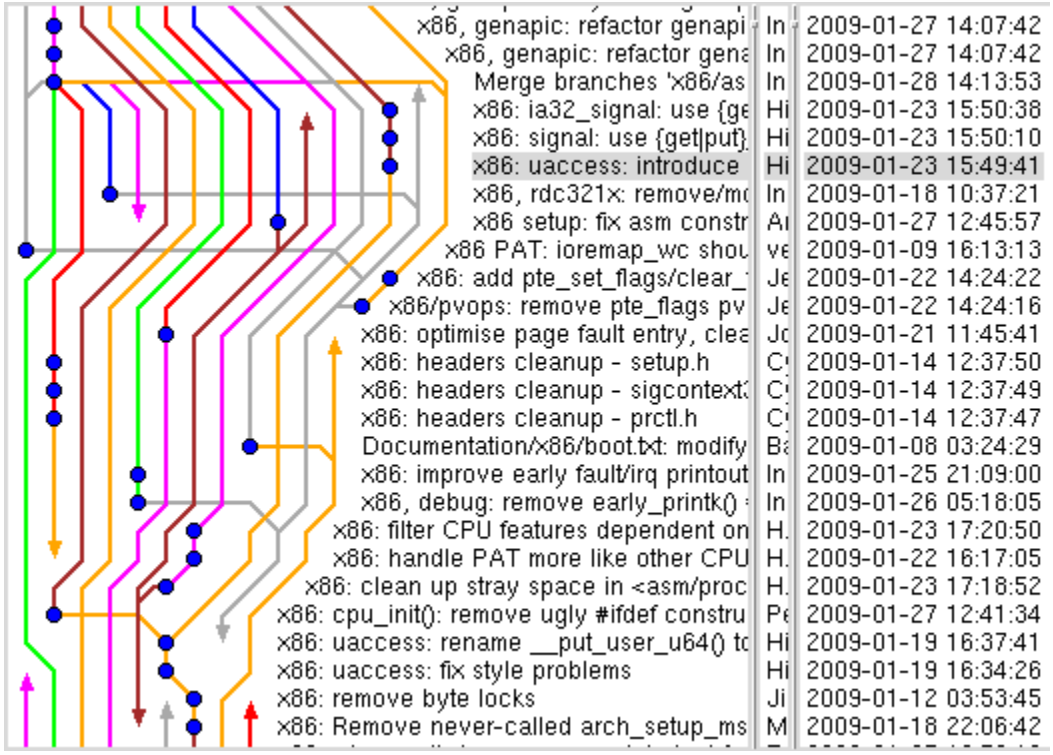
Fixes, Variants, Releases



Variants of Graph Library



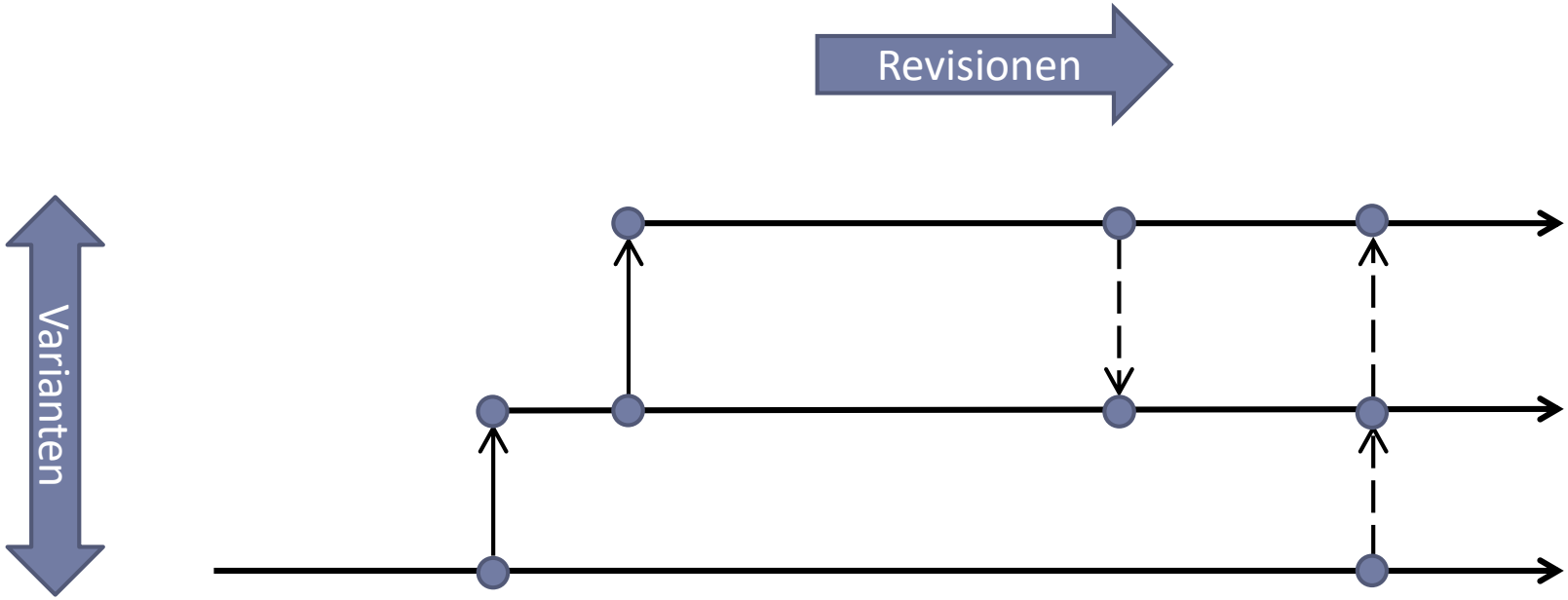
Scaling?


x86, genapic: refactor genapi
x86, genapic: refactor gena
Merge branches 'x86/as
x86: ia32_signal: use {ge
x86: signal: use {get|put}
x86: uaccess: introduce
x86, rdc321x: remove/mo
x86 setup: fix asm constr
x86 PAT: ioremap_wc shou
x86: add pte_set_flags/clear_
x86/pvops: remove pte_flags pv
x86: optimise page fault entry, clea
x86: headers cleanup - setup.h
x86: headers cleanup - sigcontext
x86: headers cleanup - prctl.h
Documentation/x86/boot.txt: modify
x86: improve early fault/irq printout
x86, debug: remove early_printk()
x86: filter CPU features dependent on
x86: handle PAT more like other CPU
x86: clean up stray space in <asm/proc
x86: cpu_init(): remove ugly #ifdef constru
x86: uaccess: rename __put_user_u64() to
x86: uaccess: fix style problems
x86: remove byte locks
x86: Remove never-called arch_setup_ms
2009-01-27 14:07:42
2009-01-27 14:07:42
2009-01-28 14:13:53
2009-01-23 15:50:38
2009-01-23 15:50:10
2009-01-23 15:49:41
2009-01-18 10:37:21
2009-01-27 12:45:57
2009-01-09 16:13:13
2009-01-22 14:24:22
2009-01-22 14:24:16
2009-01-21 11:45:41
2009-01-14 12:37:50
2009-01-14 12:37:49
2009-01-14 12:37:47
2009-01-08 03:24:29
2009-01-25 21:09:00
2009-01-26 05:18:05
2009-01-23 17:20:50
2009-01-22 16:17:05
2009-01-23 17:18:52
2009-01-27 12:41:34
2009-01-19 16:37:41
2009-01-19 16:34:26
2009-01-12 03:53:45
2009-01-18 22:06:42

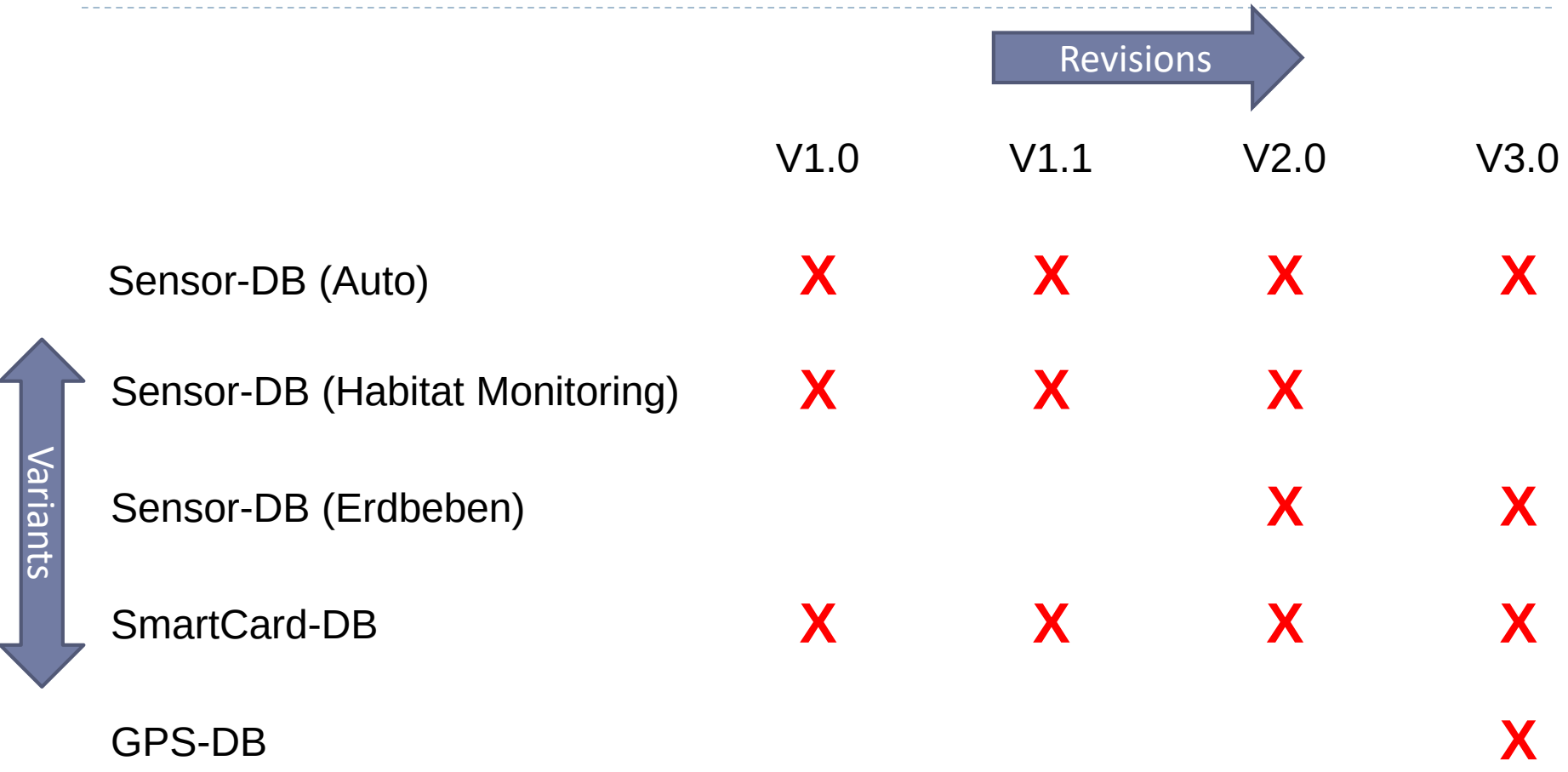
Code and Non-Code Files

- Java Code
- Documentation
- Models
- Build scripts: Ant/Makefile/Maven/Graddle
- Licenses
- Grammars
- Compiled files (be careful! Usually not advised)
- HTML, JavaScript, CSS
- For binary files is conflict resolution and merging problematic

Variants vs. Revisions

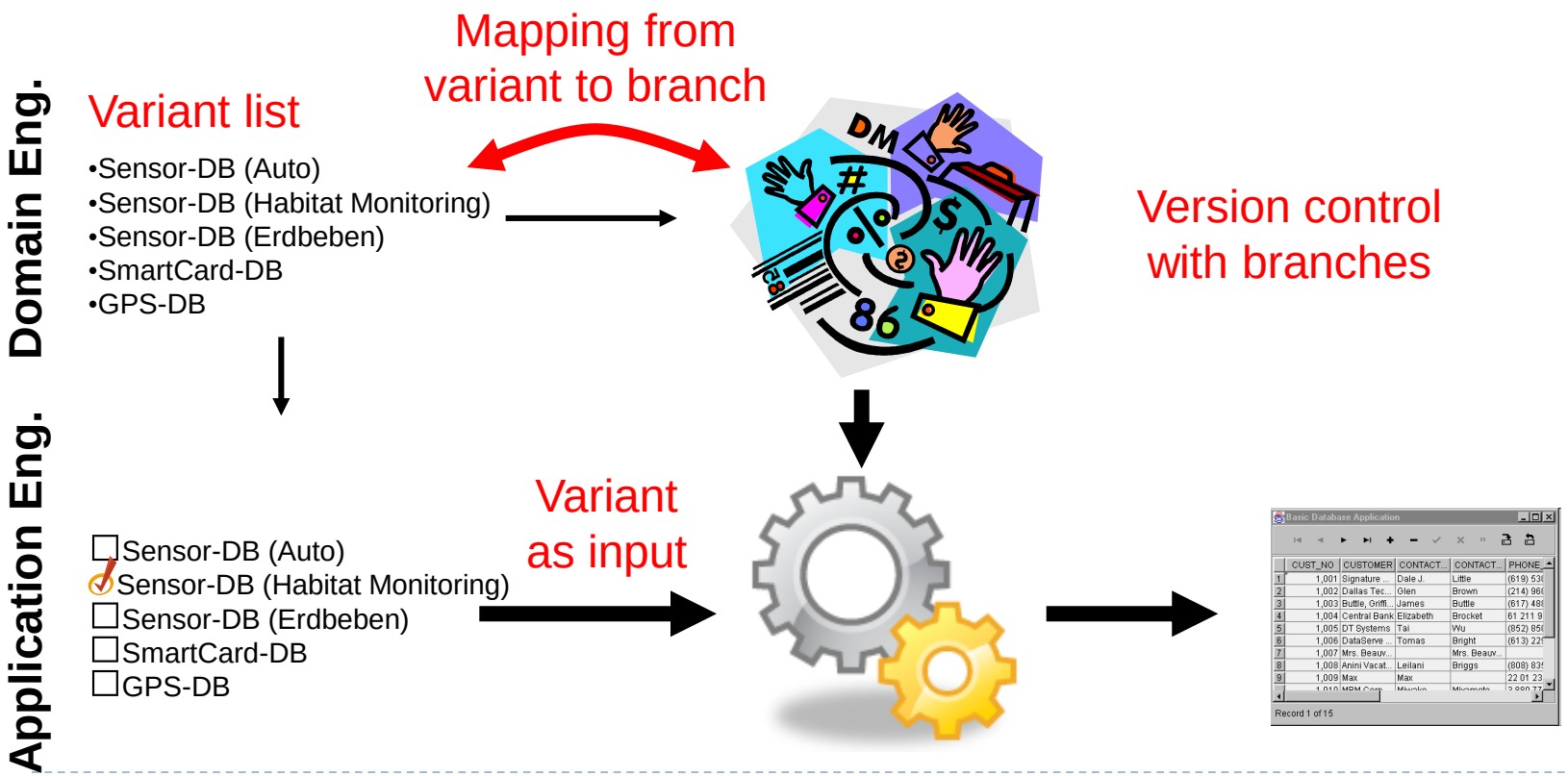


Variants vs. Revisions



Product Lines with Version Control

- Development of variants in branches
- Merge of changes between branches



Product Line with Version Control – Discussion

- Advantages
  - Established, reliable system
  - Know process
  - Excellent workflow and tool integration
- Drawbacks
  - Mixed revisions with variants
  - Development of variants, not (smaller) features: flexible combination of features is not possible
  - No systematic reuse mechanism (Copy & Edit)
  - High maintenance effort (Merging)

Build Systems

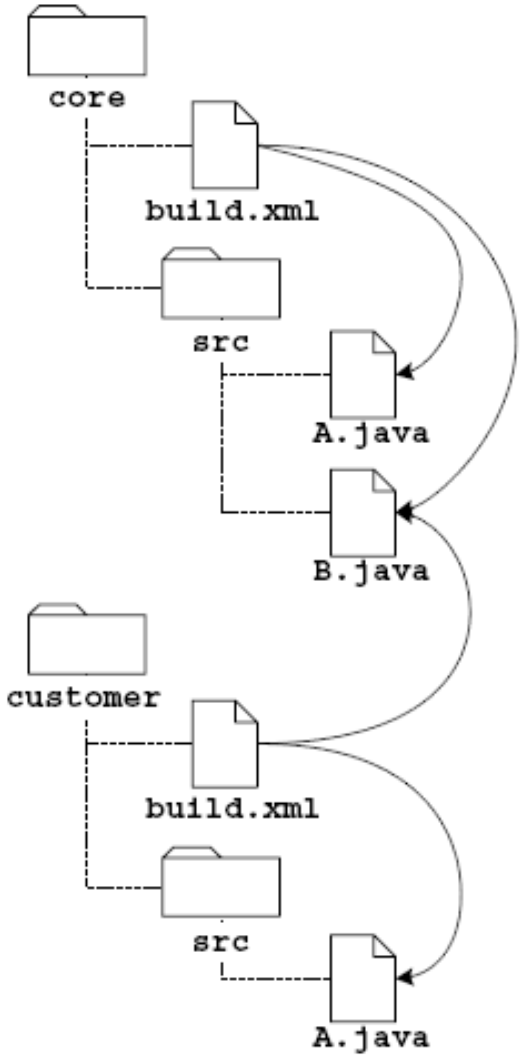
Build-Systeme

- Automating the build process
- Copies files, integrates libraries, handles dependencies, calls compiler, executes addition tools…
- Multiple steps with dependencies and requirements
- Tools: GNU make, Ant, Maven, Gradle, …

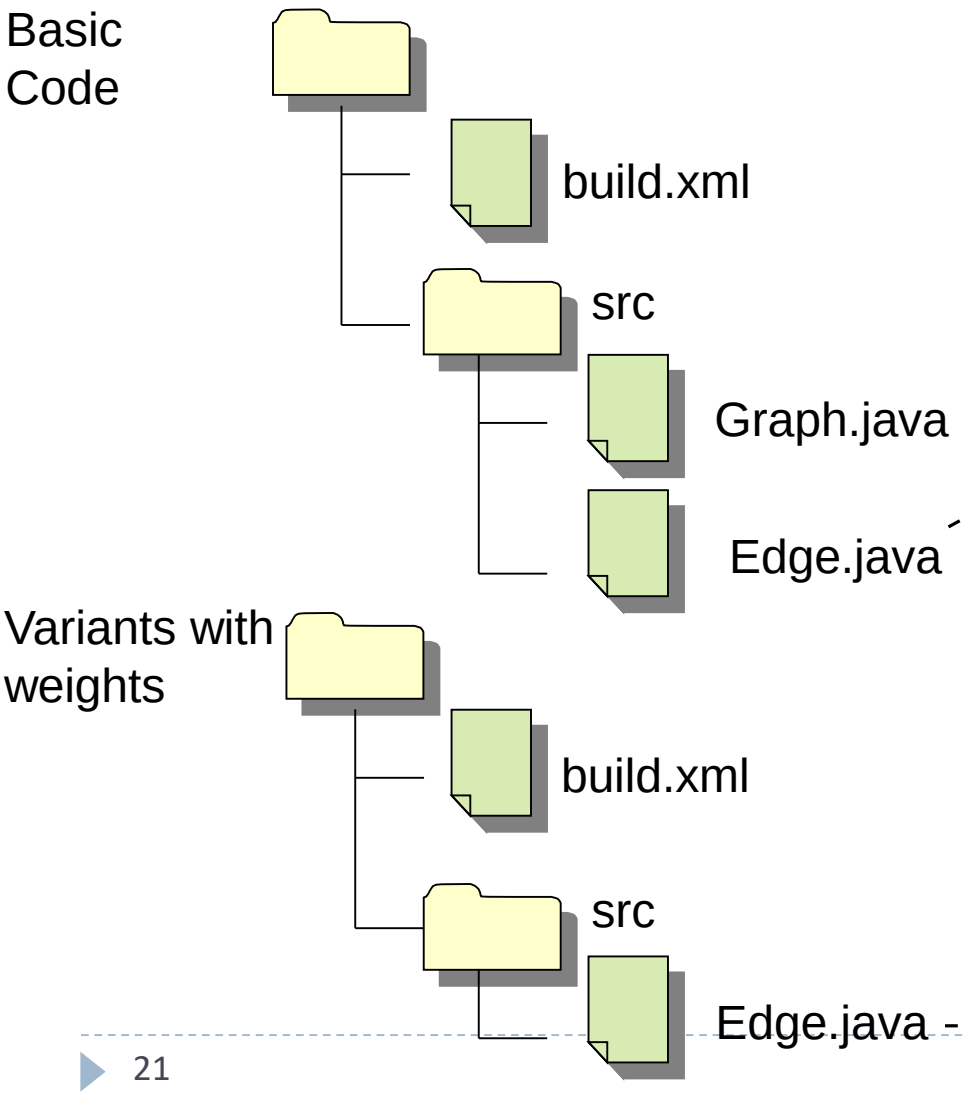
```xml
<?xml version="1.0"?>
<project name="Ant-Test" default="main" basedir=".">
    <!-- Sets variables which can later be used. -->
    <!-- The value of a property is accessed via ${} -->
    <property name="src.dir" location="src" />
    <property name="build.dir" location="bin" />
    <property name="dist.dir" location="dist" />
    <property name="docs.dir" location="docs" />

    <!-- Deletes the existing build, docs and dist directory-->
    <target name="clean">
        <delete dir="${build.dir}" />
        <delete dir="${docs.dir}" />
        <delete dir="${dist.dir}" />
    </target>

    <!-- Creates the build, docs and dist directory-->
    <target name="makedir">
        <mkdir dir="${build.dir}" />
        <mkdir dir="${docs.dir}" />
        <mkdir dir="${dist.dir}" />
    </target>

    <!-- Compiles the java code (including the usage of library for JUnit -->
    <target name="compile" depends="clean, makedir">
         <javac srcdir="${src.dir}" destdir="${build.dir}">
        </javac>
    </target>

    …

</project>
```

Product Lines with Build Systems

- One configuration file / build script for each variant
- Compilation process integrates or ignores files
- Overwriting files with product-specific variants



Example: Graph Library

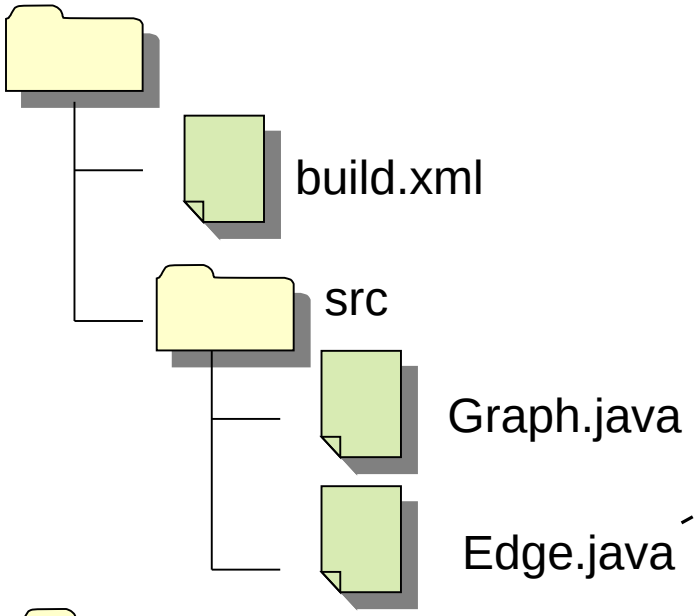



```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    a.print(); b.print();
  }
}
```

```
class Edge {
  Node a, b;
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    a.print(); b.print();
    weight.print();
  }
}
```

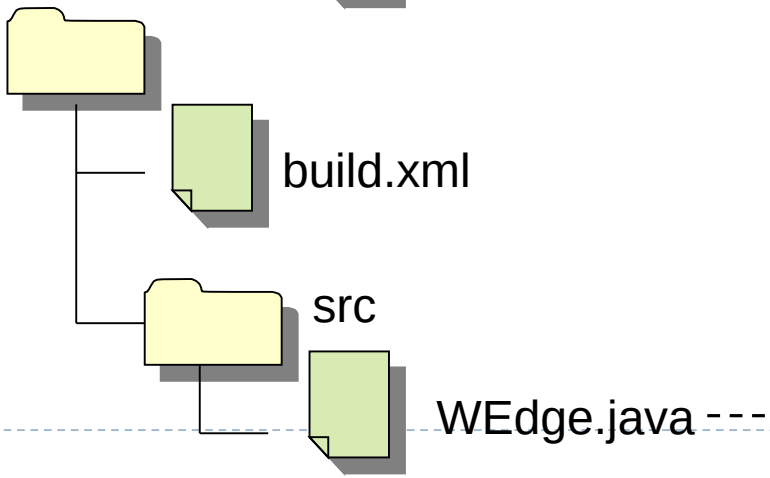
Alternative Example: Graph Library

Basic code




```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    a.print(); b.print();
  }
}
```

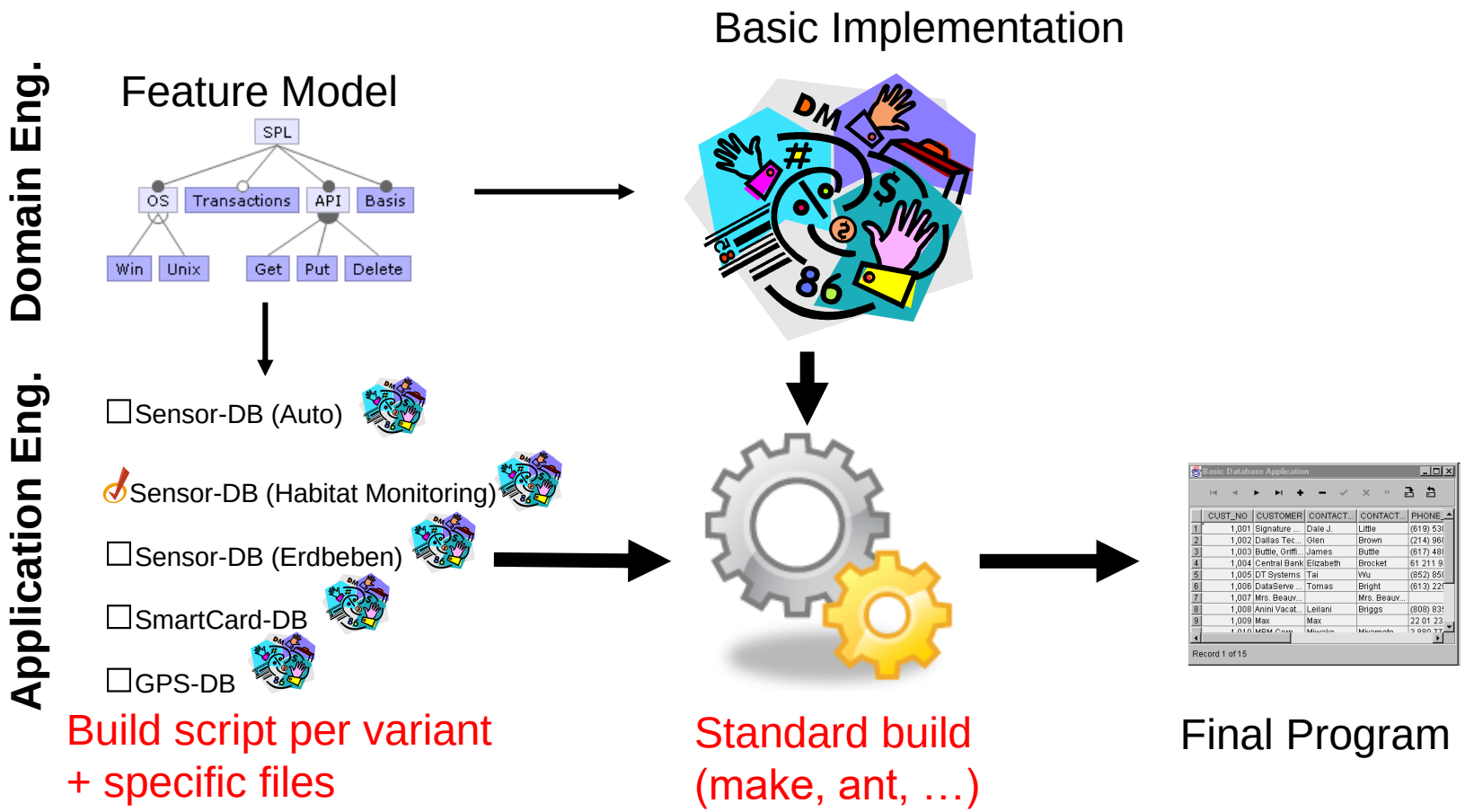
Variant with weights




```
class WEdge extends Edge {
  Weight weight;
  void print() {
    super.print();
    weight.print();
  }
}
```

…and adapt all constructor calls;
use of Factory Pattern could help

Product Line with Build System



Product Lines with Build Systems – Discussion

- Relatively simple mechanism
- High flexibility– arbitrary changes for variants
- Little preplanning possible
- Development for each variant separated, could lead to high development effort during Application Engineering
- Changes at file level (overwriting of whole files)
- Changes at base implementation difficult

Outlook

- Additional methods for implementing variability at compile time
- But: Developing features instead of variants

Literature

- M. Staples, D. Hill. Experiences adopting software product line development without a product line architecture. Proceedings APSEC, pp. 176—183, 2004
  [Industrial experience with version control and build systems for product line development]

- T. Dhaliwal, F. Khomh, Y. Zou, A. Hassan. Recovering commit dependencies for selective code integration in software product lines. Proceedings ICSM, 202—211, 2012
  [Assignment of commints to features; Dependency analysis]

Quiz

- Should we use branches for the development of program variants or for the development of individual features? List pros and cons.

- How do version control systems, build systems, and runtime parameters interact?

- What is the role of granularity of changes in version control or build systems?