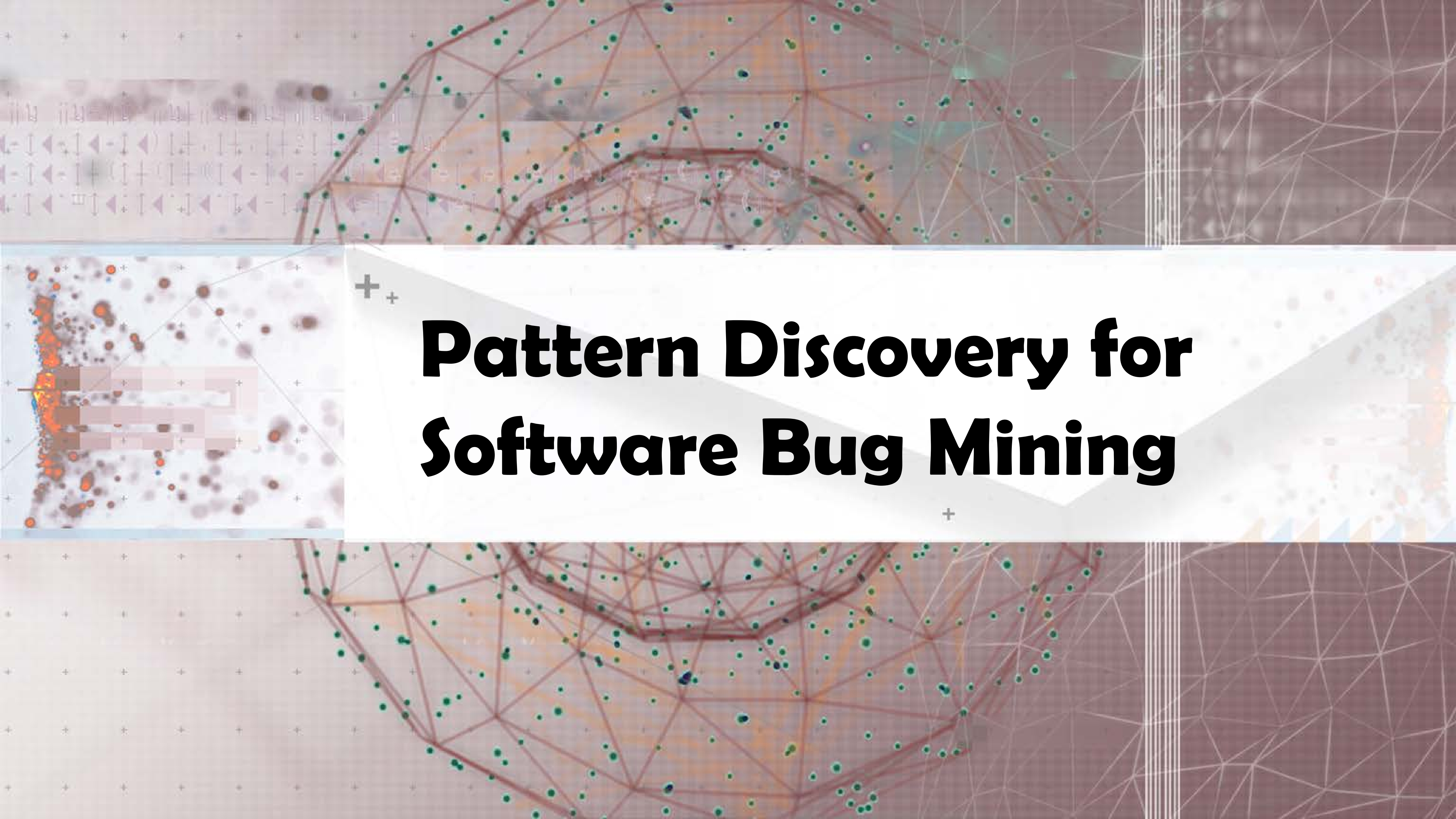


Pattern Discovery for Software Bug Mining

Pattern Discovery for Software Bug Mining

- Software is complex, and its runtime data is larger and more complex!
- Finding bugs is challenging: Often no clear specifications or properties; need substantial human efforts in analyzing data
- Software reliability analysis
  - Static bug detection: Check the code
  - Dynamic bug detection or testing: Run the code
  - Debugging: Given symptoms or failures, pinpoint the bug locations in the code
- Why pattern mining?—Code or running sequences contain hidden patterns
  - Common patterns → likely specification or property
  - Violations (anomalies comparing to patterns) → likely bugs
  - Mining patterns to narrow down the scope of inspection
    - Code locations or predicates that happen more in failing runs but less in passing runs are suspicious bug locations

Typical Software Bug Detection Methods

- **Mining rules from source code**
  - Bugs as deviant behavior (e.g., by statistical analysis)
  - Mining programming rules (e.g., by frequent itemset mining)
  - Mining function precedence protocols (e.g., by frequent subsequence mining)
  - Revealing neglected conditions (e.g., by frequent itemset/subgraph mining)
- **Mining rules from revision histories**
  - By frequent itemset mining
- **Mining copy-paste patterns from source code**
  - Find copy-paste bugs (e.g., CP-Miner [Li et al., OSDI'04]) (to be discussed here)
    - Reference: Z. Li, S. Lu, S. Myagmar, Y. Zhou, "CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code", OSDI'04

Mining Copy-and-Paste Bugs

- Copy-pasting is common
  - 12% in Linux file system
  - 19% in X Window system
- Copy-pasted code is error-prone
- Mine *"forget-to-change"* bugs by sequential pattern mining
  - Build a sequence database from source code
  - Mining sequential patterns
  - Finding mismatched identifier names & bugs

```
void __init prom_meminit(void)
{
   ……
   for (i=0; i<n; i++) {
      total[i].adr = list[i].addr;
      total[i].bytes = list[i].size;
      total[i].more = &total[i+1];
   }
   ……
```

```
for (i=0; i<n; i++) {
      taken[i].adr = list[i].addr;
      taken[i].bytes = list[i].size;
      taken[i].more = &total[i+1];
   }
```

Code copy-and-pasted but **forget to change** "id"!

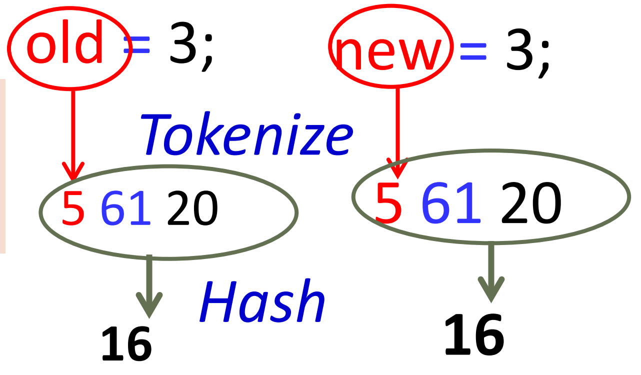
(Simplified example from *linux-2.6.6/arch/sparc/prom/memory.c*)

Courtesy of Yuanyuan Zhou@UCSD

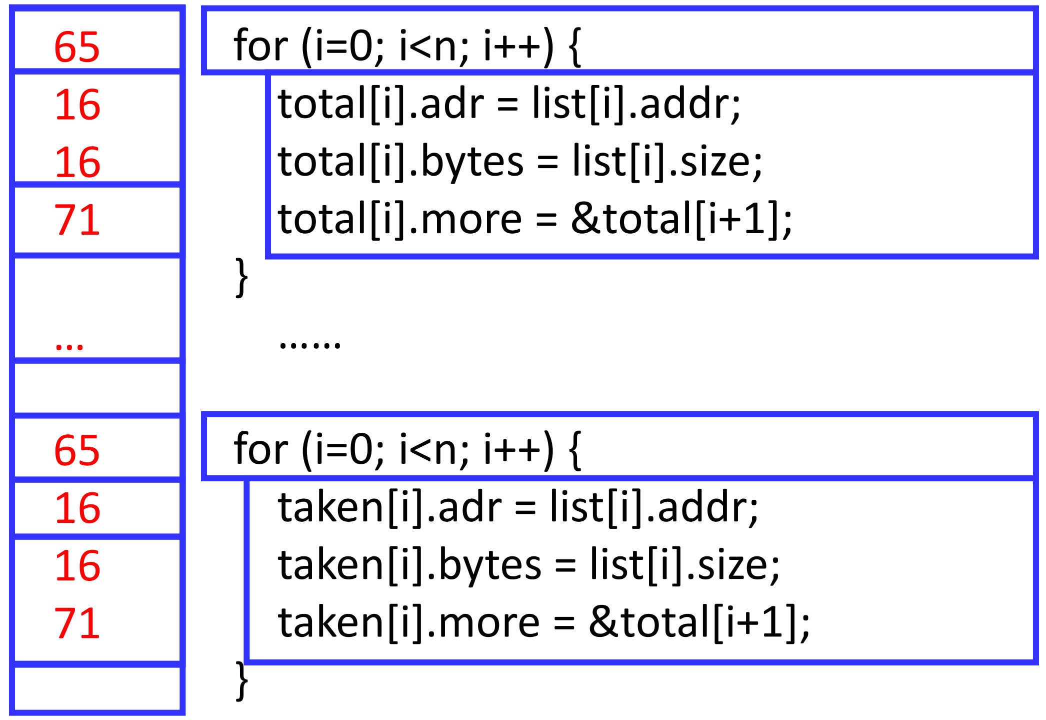
Building Sequence Database from Source Code

- Statement $\xrightarrow{\text{(mapped to)}}$ number
- Tokenize each component
  - Different operators, constants, key words → different tokens
  - Same type of identifiers → same token
- Program → A long sequence
  - Cut the long sequence by blocks

Map a statement to a number



Hash values



```
for (i=0; i<n; i++) {
    total[i].adr = list[i].addr;
    total[i].bytes = list[i].size;
    total[i].more = &total[i+1];
}
    ……
for (i=0; i<n; i++) {
    taken[i].adr = list[i].addr;
    taken[i].bytes = list[i].size;
    taken[i].more = &total[i+1];
}
```

Final sequence DB:
(65)
(16, 16, 71)
…
(65)
(16, 16, 71)

Courtesy of Yuanyuan Zhou@UCSD

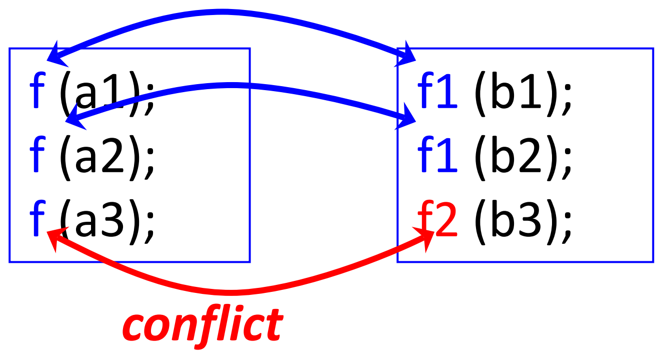
Sequential Pattern Mining & Detecting "Forget-to-Change" Bugs

- Modification to the *sequence pattern mining algorithm*
  - Constrain the max gap

(16, 16, 71)
……
(16, 16, 10, 71)

Allow a maximal gap: inserting statements in copy-and-paste

- Composing Larger Copy-Pasted Segments
  - Combine the neighboring copy-pasted segments repeatedly
- Find conflicts: Identify names that cannot be mapped to the corresponding ones
  - E.g., 1 out of 4 "**total**" is unchanged, *unchanged ratio* = 0.25
  - If 0 < *unchanged ratio* < *threshold*, then report it as a bug
- CP-Miner reported many C-P bugs in Linux, Apache, … out of millions of LOC (lines of code)



Courtesy of Yuanyuan Zhou@UCSD