### The StackwalkerAPI

- **Is a cross-platform library for walking call stacks.**
- **Works through a debugger interface or in its own process.**
- **Is customizable and extensible, easily integrating into pre-existing tools.**

### Customizable Plug-in Interface

- **Plug-ins allow StackwalkerAPI to be integrated easily into other tools.**
- **Customizes how StackwalkerAPI looks up symbol names, accesses a process, or walks through types of stack frames.**
- **Customizes how symbol names are looked up for each stack frame.**
- **Default uses SymtabAPI.**
- **Customizes how StackwalkerAPI reads from a process.**
- **Defaults use ProcControlAPI or read from current process.**

### Frame Steppers:

- **Describe how to walk through a type of frame.**
- **Find address ranges of code that this Frame Stepper can be used over.**

### Symbol Name Lookup

- **Process Access (Current Process):**
- **Process Access (Via Debugger):**
- **Frame Stepper (Normal Functions):**
- **Frame Stepper (Optimized Functions):**
- **Frame Stepper (Signal Handlers):**
- **Frame Stepper (Instrumentation):**
- **Frame Stepper (System Calls):**
- **Frame Stepper (Debug Information):**

### Example User Tool

```cpp
std::vector<Frame> stackwalk;
string s;
Walker *walker = Walker::newWalker();
walker->walkStack(stackwalk);
for (int i=0; i<stackwalk.size(); i++) {
    stackwalk[i].getFuncName(s);
    cout << "Function " << s << endl;
}
```

### Handling Optimized Functions

#### Solution 1: Static analysis

- **Analyze the function to understand how the stack changes as it executes.**
- **For each instruction, determine the distance, \( \Delta \), from the top of the current frame to the bottom of the current frame.**

#### Example Address Space

<table>
<thead>
<tr>
<th>Address</th>
<th>Words in the Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>080483f5: call foo</td>
<td>...</td>
</tr>
<tr>
<td>080483f9: ...</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x00000000</td>
<td>0x080483f9</td>
</tr>
<tr>
<td>0x080483f9</td>
<td>0x0f60b3b0</td>
</tr>
<tr>
<td>0x0f60b3b0</td>
<td>0x4010a7f0</td>
</tr>
<tr>
<td>0x4010a7f0</td>
<td>0x80000000</td>
</tr>
<tr>
<td>0x80000000</td>
<td>0x0000000c</td>
</tr>
</tbody>
</table>

#### An address is likely the top of a frame if ...

- ... it points to an instruction that follows a call, and
- ... the next word in the stack points into the stack.

### User Tool

**Frame Stepper:**

- **Normal Functions:**
- **Optimized Functions:**
- **Signal Handlers:**
- **Instrumentation:**
- **System Calls:**
- **Debug Information:**

**User Tool:**

```cpp
std::vector<Frame> stackwalk;
string s;
Walker *walker = Walker::newWalker();
walker->walkStack(stackwalk);
for (int i=0; i<stackwalk.size(); i++) {
    stackwalk[i].getFuncName(s);
    cout << "Function " << s << endl;
}
```

**Solution 2: Stack Value Inspection**

- **For each word in the stack, use a heuristic to determine if it could be a return address that was generated by a call instruction.**
- **Useful if there is no other way to walk through a stack frame, but prone to false positives.**

### Stack Value Inspection

- **An address is likely the top of a frame if ...**
  - ... it points to an instruction that follows a call, and
  - ... the next word in the stack points into the stack.