The Latest in the Deconstruction of DyninstAPI

Drew Bernat and Matthew LeGendre

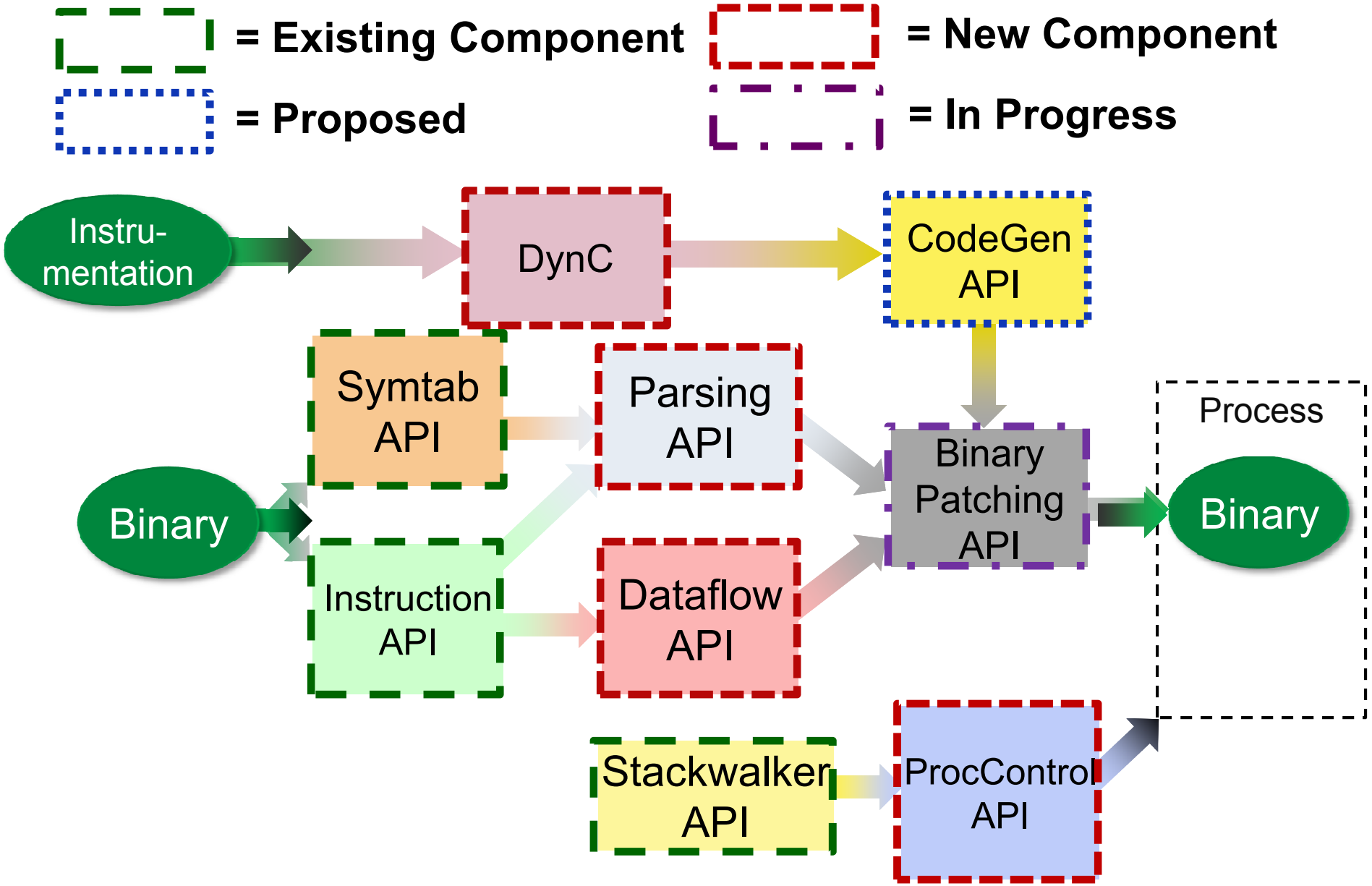
Jeff Hollingsworth and Bart Miller

David Aeschilmann, Mike Brim, Ray Chen, Todd Fredrick,Tugrul Ince, Emily Jacobson, Aishwarya Kumar, Madhavi Krishnan, Dan McNulty, Ramya Olichandran, Nathan Rosenblum, Kevin Roundy, James Waskiewicz

Paradyn

Dyninst

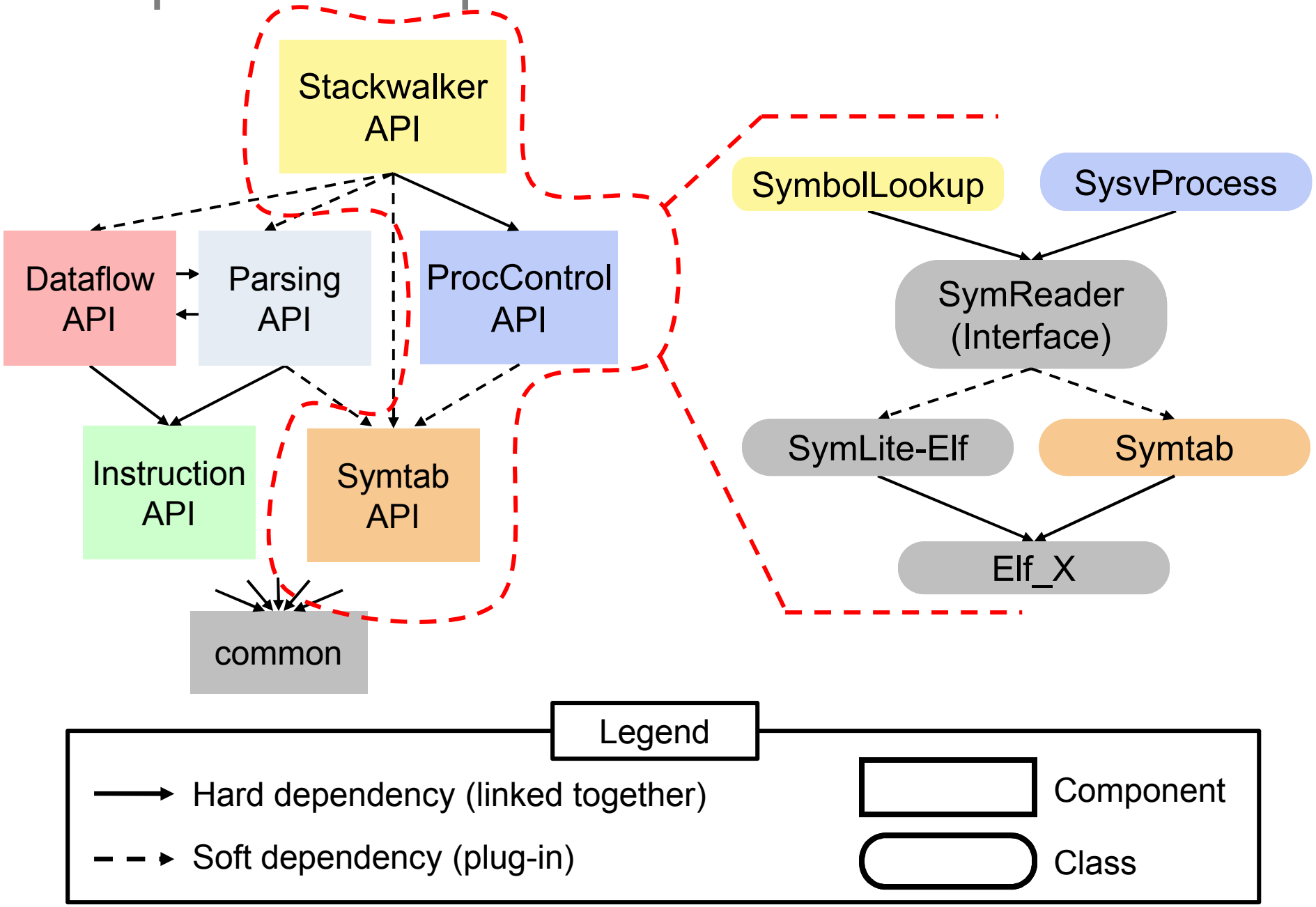
Dyninst and the Components



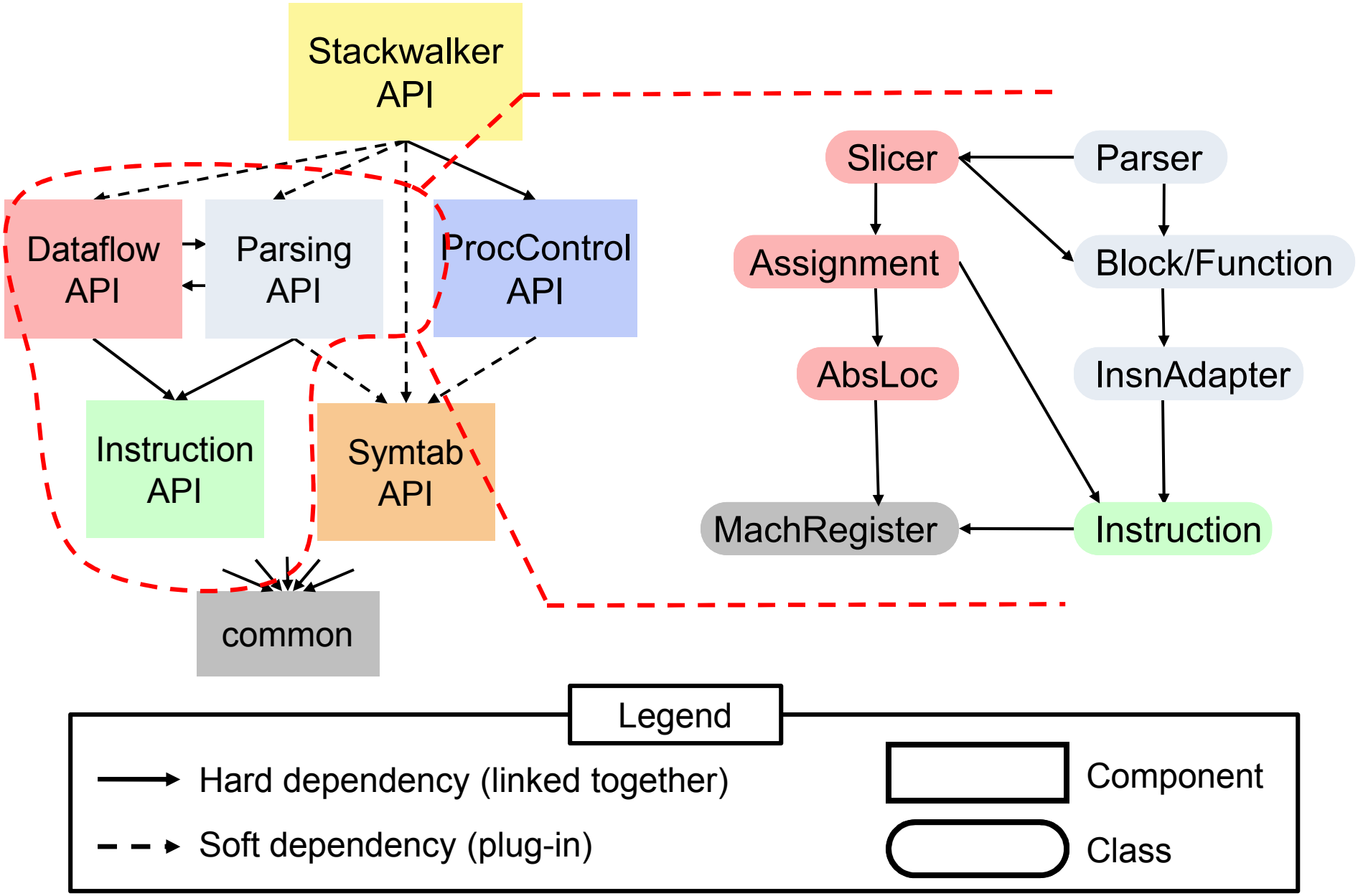
Talk Outline

- **Component Dependencies**
- **ProcControlAPI**
- **ParsingAPI**
- **DynC**
- **Wrapping C++ in C**
- **Other Components/Wrapup**

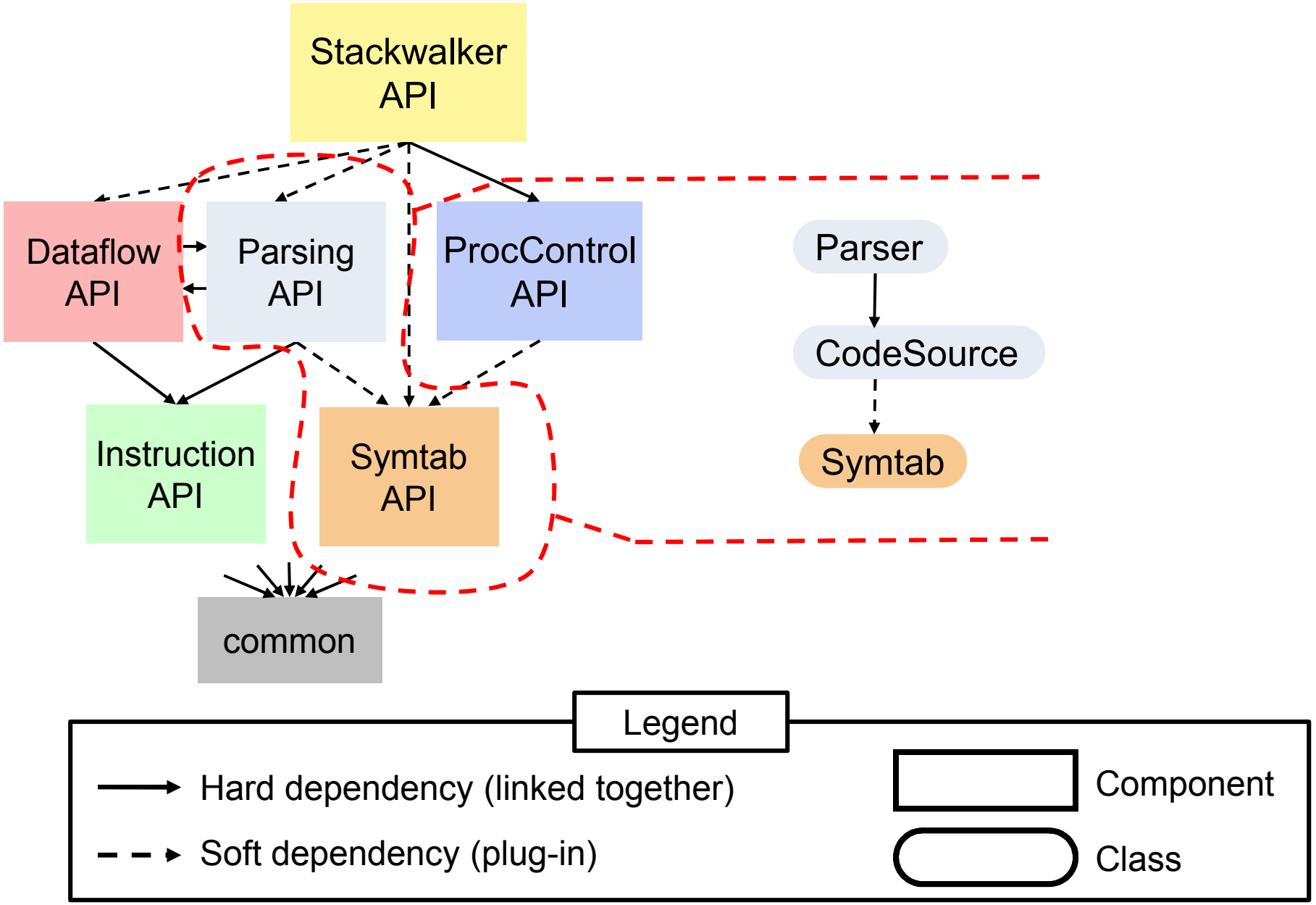
Component Dependencies



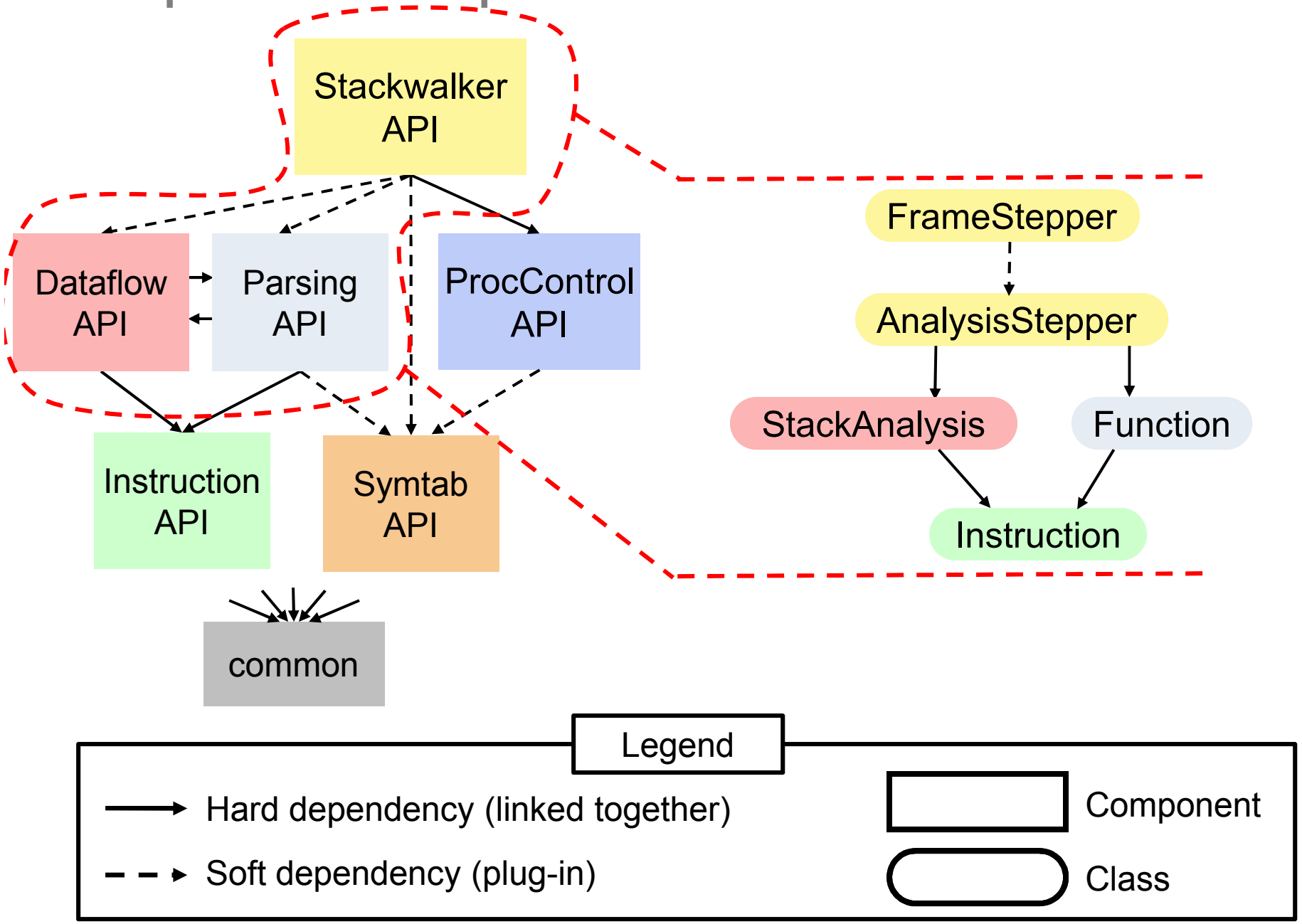
Component Dependencies



Component Dependencies



Component Dependencies

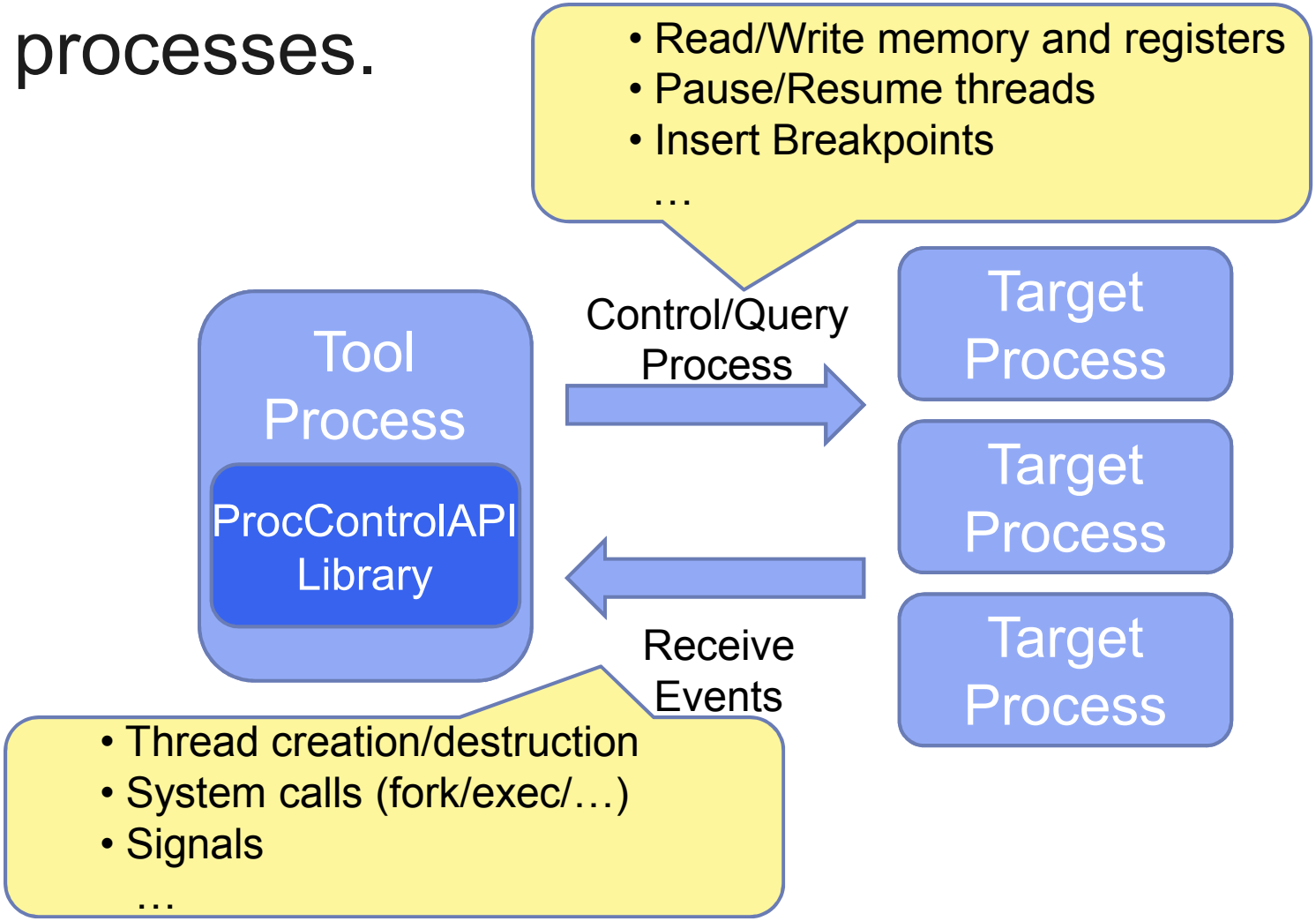


Talk Outline

- Component Dependencies
- **ProcControlAPI**
- ParsingAPI
- DynC
- Wrapping C++ in C
- Other Components/Wrapup

ProcControlAPI

- A library for controlling and monitoring processes.



ProcControlAPI Goals

- Platform Independent Interface
  - Implemented on: Linux, FreeBSD, CrayXT, BlueGene, AIX, Solaris, VXWorks, Windows
- Simple and Powerful
  - High-level abstractions in interface. E.g.,
    - Breakpoints
    - Inferior RPCs
    - Process Control
    - Events

Example: Single Step Threads

```
void singlestep_proc(int pid) {
  Process::ptr proc = Process::attachProcess(pid);
  Process::registerEventCallback(EventType::SingleStep,
                                 singlestep);
  ThreadPool::iterator i = proc->threads().begin();
  for (; i != proc->threads().end(); i++)
    (*i)->setSingleStepMode(true);

  proc->continueProc();
  while (!proc->isTerminated())
    Process::handleEvents(true);
}
```

Attach to a
Register a

For each thread running in the process
Enable single-stepping

Run the process
Block until process terminates

Example: Single Step Threads

```
Process::cb_ret_t on_singlestep(Event::ptr event)
{
  unsigned long cur_pc, thread_id;
  event->getThread()->getRegister(MachRegister::getPC(),
                                  cur_pc);
  thread_id = event->getThread()->getLWP();

  printf("Singlestep to %lx on thread %lx\n",
         cur_pc, thread_id);

  return Process::cbThreadContinue;
}
```

Called each time a thread executes an instruction

Get the thread's PC and LWP.

Tell ProcControlAPI to continue the thread

Process Class

- Handle for target process
- Typical operations
  - Create and attach to processes
  - Access process memory
  - Insert breakpoints
  - Track library loads/unloads
  - Track thread creation and destruction
  - Stop/Continue all threads
  - Detach from/terminate target process

Thread Class

- Handle for thread in target process
- Typical Operations
  - Stop/Continue individual thread
  - Get/Set registers
  - Single step

- Assuming 1:1 threading/LWP model
  - Thread object represents OS's LWP
  - Thread object represents user space thread

Inferior RPCs

- Insert and run code in target process
  - User provides machine code to run
  - ProcControlAPI allocates memory, saves registers, runs code, restores registers and memory
  - Returns result of iRPC

- gdb supports a subset of this

```
(gdb) call getpid()
$1 = 14218
(gdb)
```

Events

- Can register callbacks for process events:
  - Fork
  - Pre-Exec
  - Post-Exec
  - Pre-Exit
  - Post-Exit
  - Crash
  - Signal
  - Thread Create
  - Thread Destroy
  - Library Load
  - Library Unload
  - Breakpoint
  - RPC Completion
  - Single Step

Callbacks

- Events are delivered via callbacks
  - User registers callbacks for interesting events

- Restrictions on callback functions
  - Can not call anything that would recursively trigger more callbacks.
  - This prevents races

ProcControlAPI Status

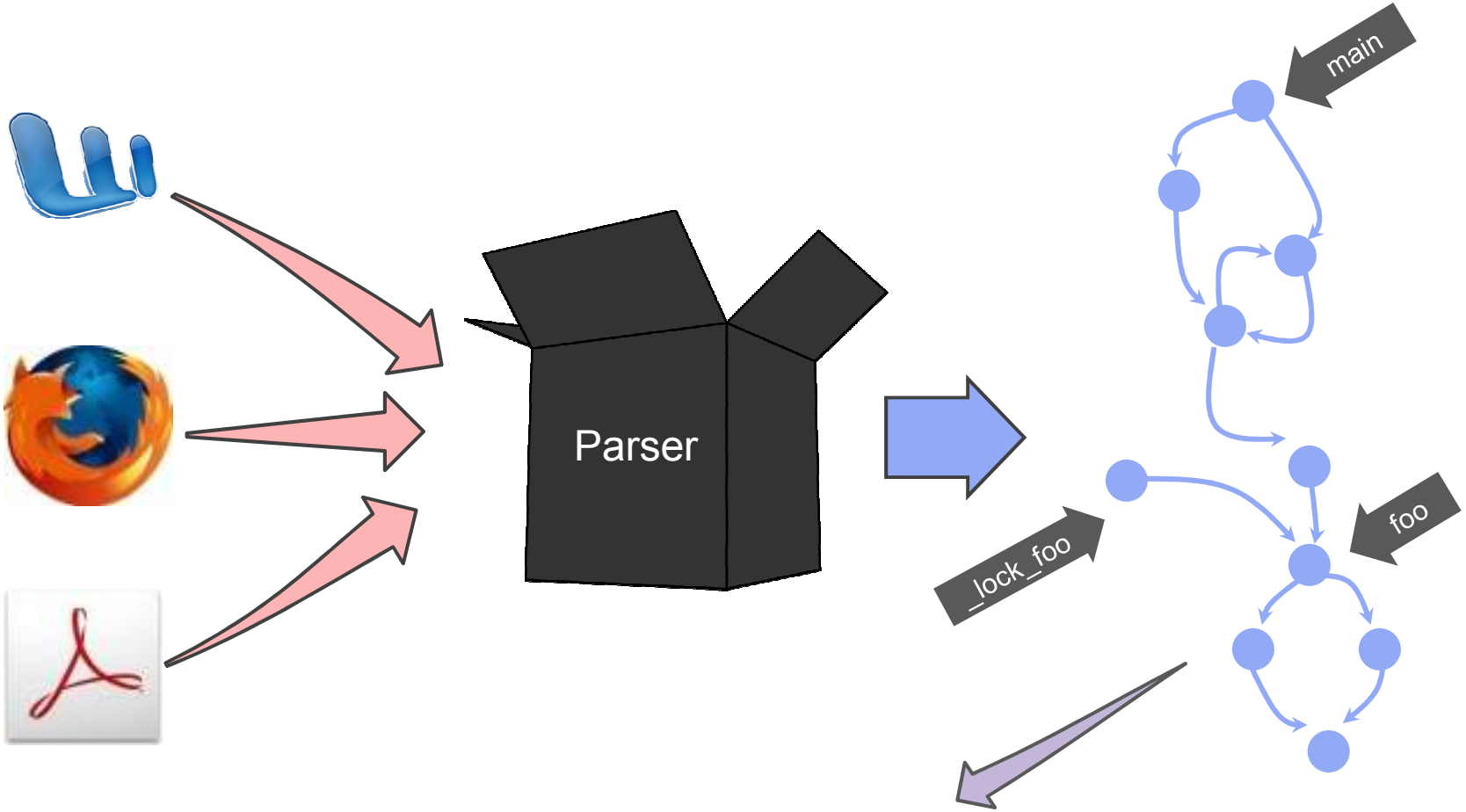
- Currently on Linux/x86, Linux/x86_64, FreeBSD,Cray/XT
- Next platforms are Linux/ppc, BlueGene.
- Windows and AIX support to follow.

- Beta release available upon request.

Talk Outline

- Component Dependencies
- ProcControlAPI
- **ParsingAPI**
- DynC
- Wrapping C++ in C
- Other Components

Binary parsing



dynamic instrumentation, debugger, static binary analysis tools, malware analysis, binary editor/rewriter, …

We've been down this road…

the Dyninst binary parser

recursive traversal parsing

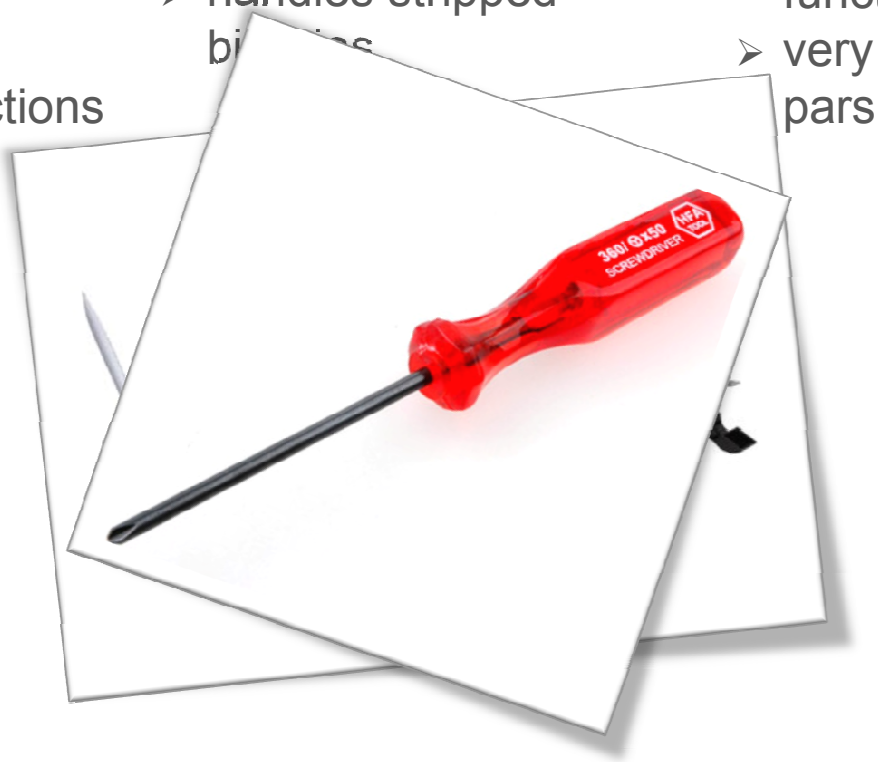
- non-contiguous functions
- code sharing
- non-returning functions

"gap" parsing heuristics

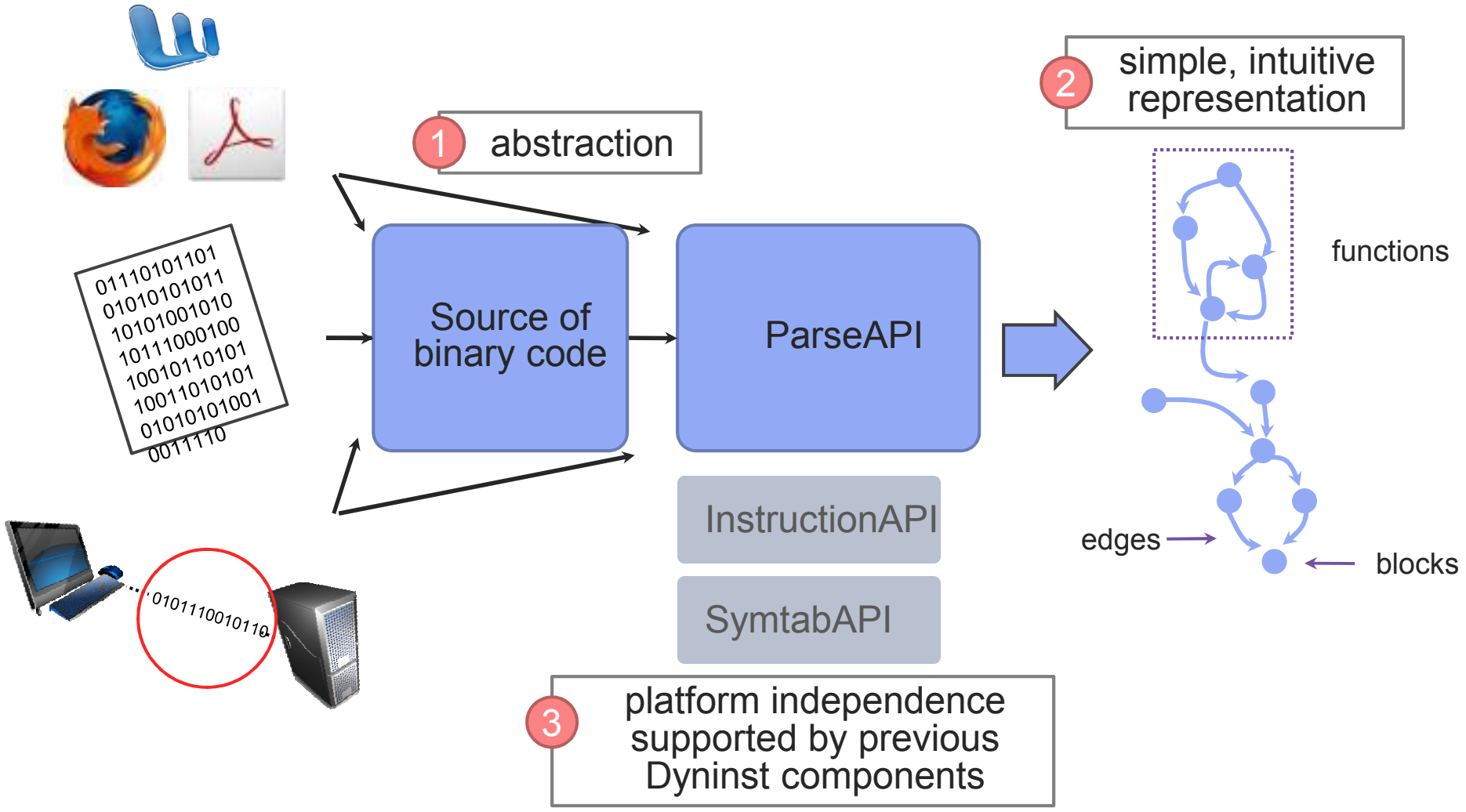
- preamble scanning
- handles stripped bi…

probabilistic code models

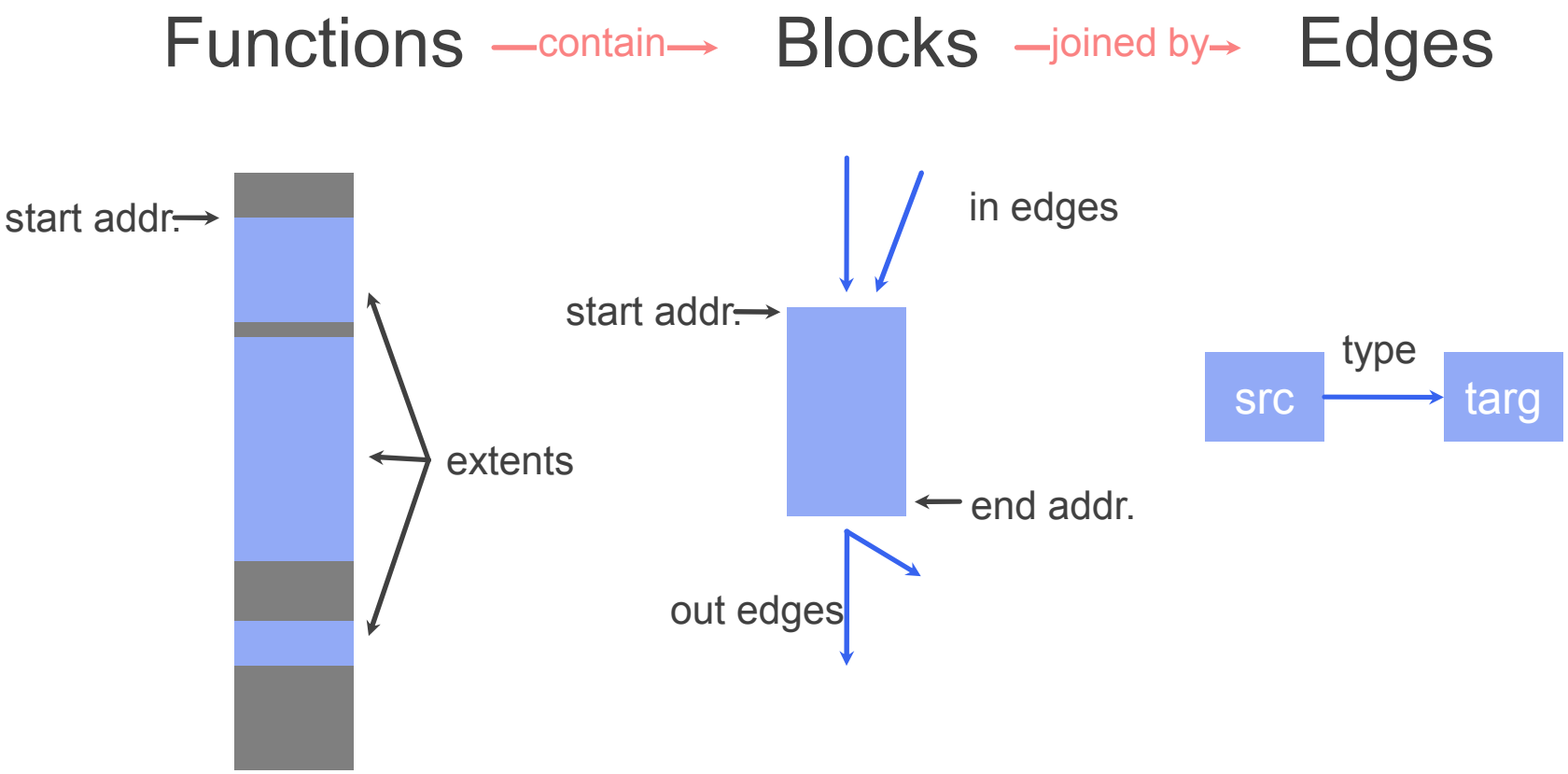
- learn to recognize function entry points
- very accurate gap parsing



What makes a parsing component?



Simple control flow interface



Parsing Interface

```
SymtabCodeSource s("/lib/libc.so.6");
CodeObject libc(&s);
Address entry_addr = ...;

// Find and parse the function @ entry_addr
libc.parse(entry_addr, NO_RECURSE);

// Parse the entire object
libc.parse ();

// Find "printf"
Function *printf_func = libc.findFuncByName("printf");

// Find the function starting at entry_addr
Function *other = libc.findFuncByEntry(entry_addr);
```

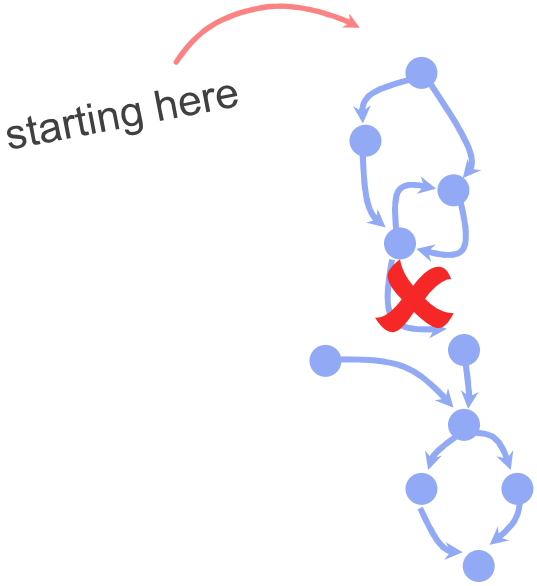
Views of control flow

*Walking an **intra**procedural control flow graph*

```
blocks.push_back(printf_func->entry());
while(!blocks.empty()) {
  Block *b = blocks.pop();

  /* do something with b */


  edgeiter eit = b->out().begin();
  while(eit != b->out().end()) {
    blocks.push(*eit++);
  }
}
```



What if we only want intraprocedural edges?

Edge predicates

*Walking an **intraprocedural** control flow graph*

```
blocks.push_back(printf_func->entry());
while(!blocks.empty()) {
  Block *b = blocks.pop();

  /* do something with b */

  IntraProc pred;
  edgeiter eit = b->out().begin(&pred);
  while(eit != b->out().end()) {
    blocks.push(*eit++);
  }
}
```

Edge Predicates

Tell iterator whether Edge argument should be returned
Composable (and, or)

Examples:

- Intraprocedural
- Single function context

What's in the box?* box to be released soon

Binary Parser

- recursive descent parsing
- speculative gap parsing
- cross platform: x86, x86-64, PPC, SPARC

SymtabAPI-based Code Source

- cross-platform
- supports ELF, PE, XCOFF formats

Control Flow Graph Representation

- graph interface
- extensible objects for easy tool integration

Dataflow Analysis

?

Dataflow applications

Inside parsing:

- Improved jump table parsing
- Virtual function identification
- Detecting return address modifications
- Identifying library functions via syscall patterns

Other:

- Relocating code safely
- Stack walking

Interface Questions

Low level concepts

- Abstract location
- Abstract region

Graph representation

- Data dependence
- Control dependence
- Slicing

Symbolic Evaluation

- Translate instructions to functions via semantics

Analysis Algorithms

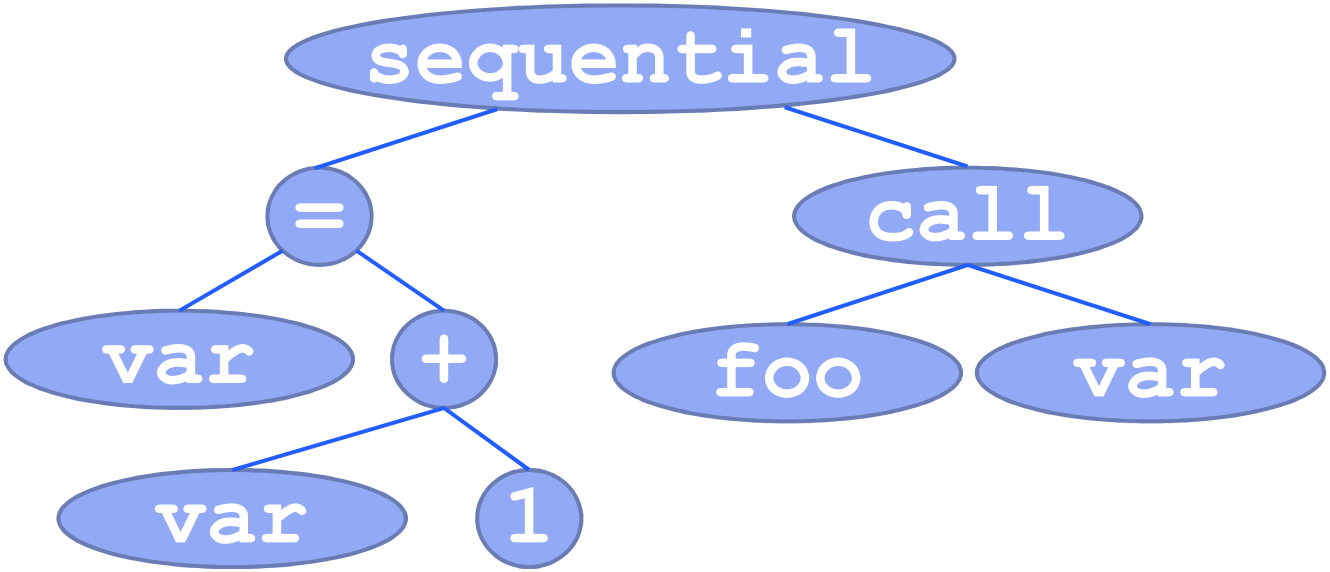
- Stack analysis
- …?

Talk Outline

- Component Dependencies
- ProcControlAPI
- ParsingAPI
- **DynC**
- Wrapping C++ in C
- Other Components/Wrapup

DynC – An Instrumentation Language

- Old ASTs



- DynC Language

```
`foo(++`var);
```

DynC Domains

- Reference symbols with domains (backtick operator):

```
global`printf("Open called on %s\n"
                param`pathname);
```

- Lone backticks have Dyninst search:

```
`printf("Loop index is %d\n"
        local`i);
```

- Idea from Cinquecento language

DynC Instrumentation Support

- Use $ operator to access built-in instrumentation functions

```
if ($thread_index == 0) {
   `printf("Entering function %s\n",
            $function_name);
}
```

- Declare variables in instrumentation

```
int i;
i++;
```

Talk Outline

- Component Dependencies
- ProcControlAPI
- ParsingAPI
- DynC
- **Wrapping C++ in C**
- Other Components/Wrapup

C++ in First-Party Tools

Good

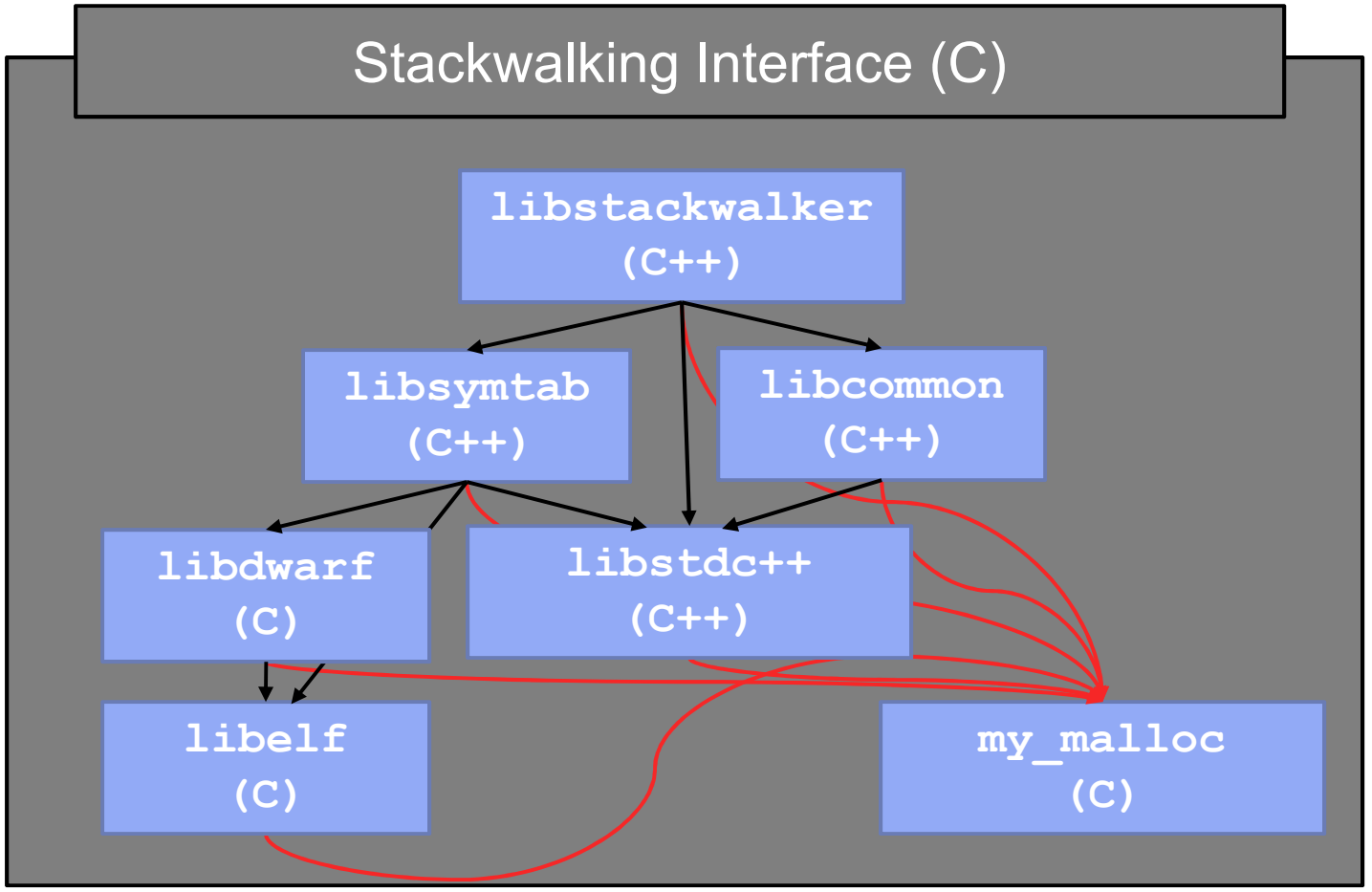
- A high-level language, good for large applications
- Large suite of support libraries (STL, Boost)
- Good performance

Bad

- Incompatibilities between libstdc++ versions
- Implicit dynamic memory allocations from STL
- Poor interoperability with other languages

- DyninstAPI and components are about ~460,000 lines of C++

Goal: Hide the C++



- Wrap collection of libraries into “wrapper” library
  - Only export minimal C interface

Recipe for Wrapper Library

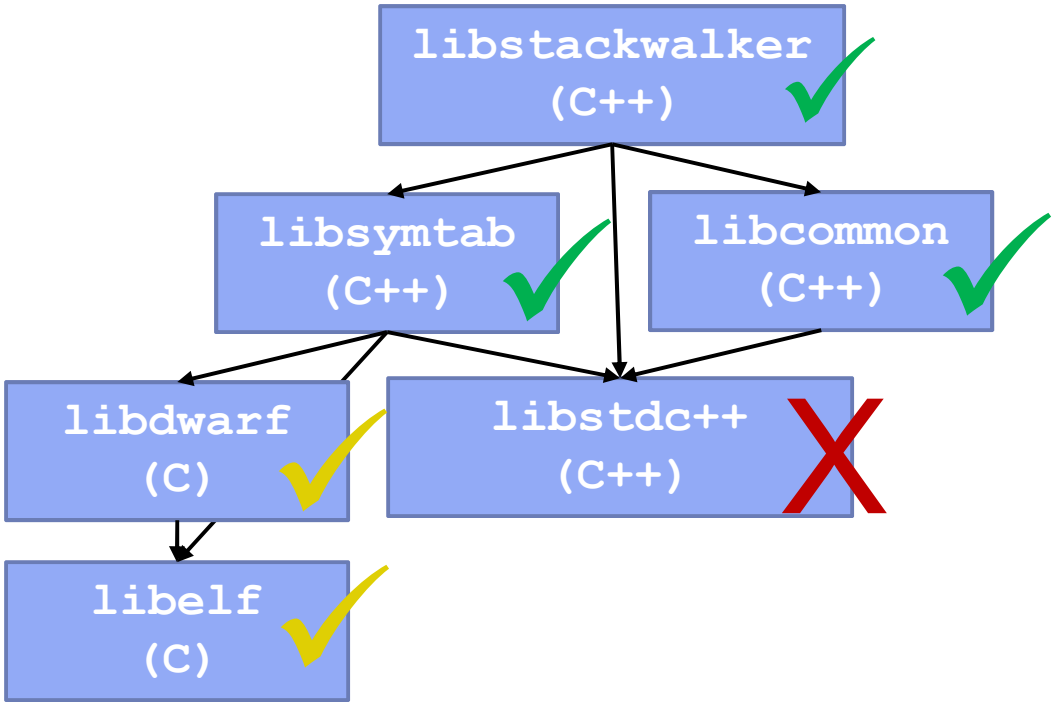
For each sub-library

- Build object files with –fPIC
- Link into static archive

Link sub-libraries into wrapper shared library

- Include custom malloc
- Include “C” interface
- Use GNU ld’s --exclude-libs option to hide exports from sub-libraries

The Catch: Building archives with -fPIC



✓ Written by us—easy to change

✓ Simple tool—not too difficult to change

X Part of gcc—hard to change

Building fPIC libstdc++.a

- GCC comes with:
  - libstdc++.so, built with –fPIC
  - libstdc++.a, built without –fPIC
- Build custom gcc with fPIC and libstdc++.a
  - Modify libstdc++ Makefile, add –fPIC to CFLAGS
- Distribute modified libstdc++ with tool
  - Or distribute wrapper library as binary code

Current Status

- StackwalkerAPI into wrapper library
  - No visible C++
  - Does not touch application heap
  - ~3MB on disk

- "C" interface to StackwalkerAPI partially completed.

Talk Outline

- Component Dependencies
- ProcControlAPI
- ParsingAPI
- DynC
- Wrapping C++ in C
- **Other Components/Wrapup**

StackwalkerAPI

- Current:
  - Linux/x86, Linux/x64, Linux/PPC32, Linux/PPC64, BlueGene/L, BlueGene/P, CrayXT
  - DWARF based stack walking
- Future:
  - Analysis based stack walking
  - No file access stack walking
  - ProcControlAPI for third-party

InstructionAPI and SymtabAPI

- InstructionAPI:
  - x86, x86_64, PPC32, PPC64 instruction decoding
  - Significant performance improvements over previous versions
- SymtabAPI
  - Object file and debug parsing for ELF, PE, XCOFF
  - Shared libraries, static libraries, executables, static binaries, object files
  - Rewriting interface

DyninstAPI

- Static and dynamic binary instrumentation
- Binary analysis framework

- New features:
  - LGPL licensed
  - Rewriting statically linked binaries
  - FreeBSD/x86_64 support
  - Linux/PPC static rewriting
  - BlueGene static rewriting

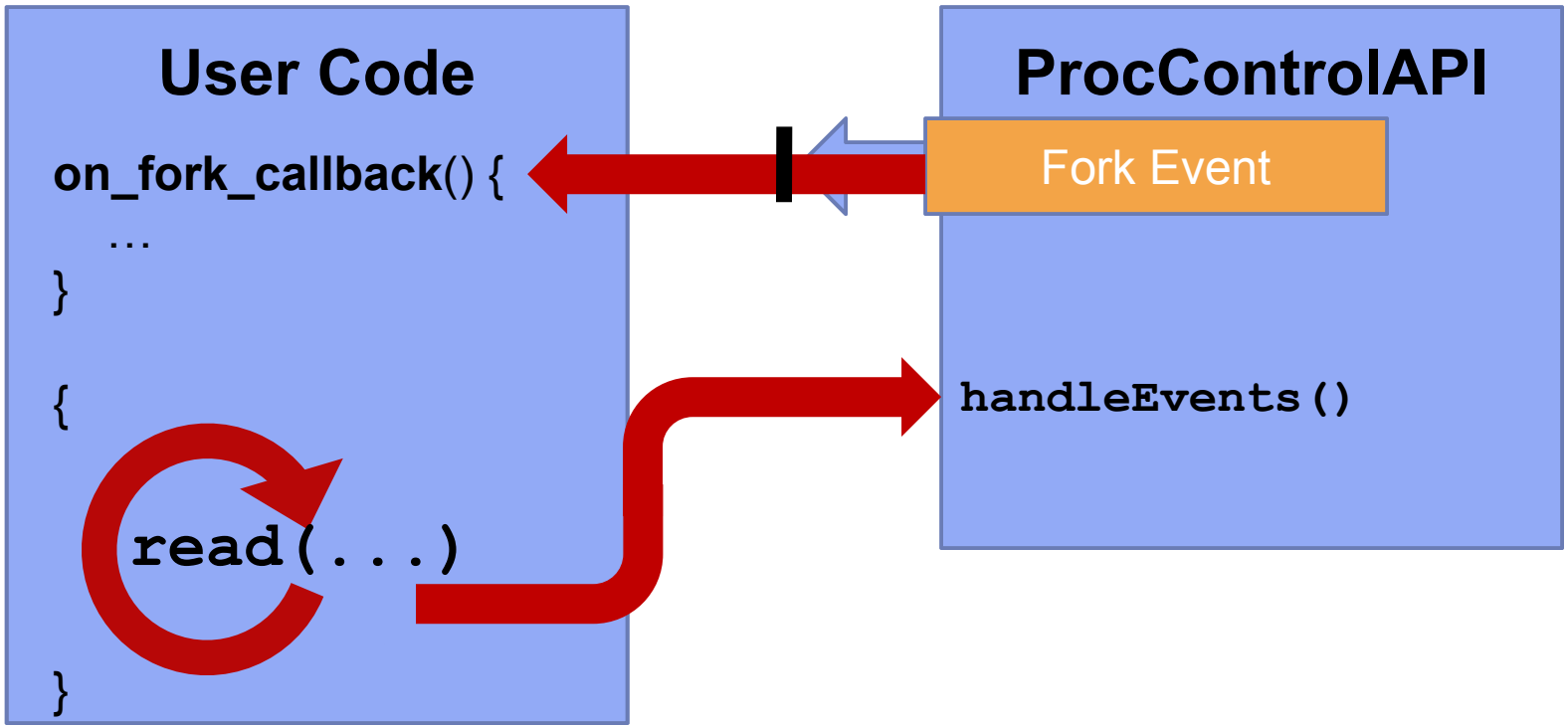
New MRNet features

- Cray XT support
- Lightweight MRNet back end (C-based)
- Per-stream data arrival notification
- Optimizations to startup efficiency
- Beta: topology information available at filters
- Timeout filters (finally!)

Questions?

Callback Notification

- Want to deliver callback on user thread
  - User thread may be busy or blocked



- ***Notify*** user that event is pending via Callback or FD

ProcControlAPI Use Cases

- StackwalkerAPI
  - Read stack memory
  - Access registers
  - Stop/Continue process
  - Track dynamic libraries
- DyninstAPI
  - Write instrumentation
  - Receive events
  - Track threads
  - Walk call stack
  - Stop/Continue process
  - OneTimeCode
  - Launch/Attach processes
  - Track dynamic libraries
  - Read/Set variable values
- Debugger
  - Receive Events
  - Track threads
  - Walk call stack
  - Stop/Continue process
  - Insert breakpoints
  - Track dynamic libraries
  - Read/Set variable values
  - Single step