Dyninst's Binary Rewriter and Open|SpeedShop

Matthew LeGendre
University of Wisconsin
legendre@cs.wisc.edu
http://www.paradyn.org

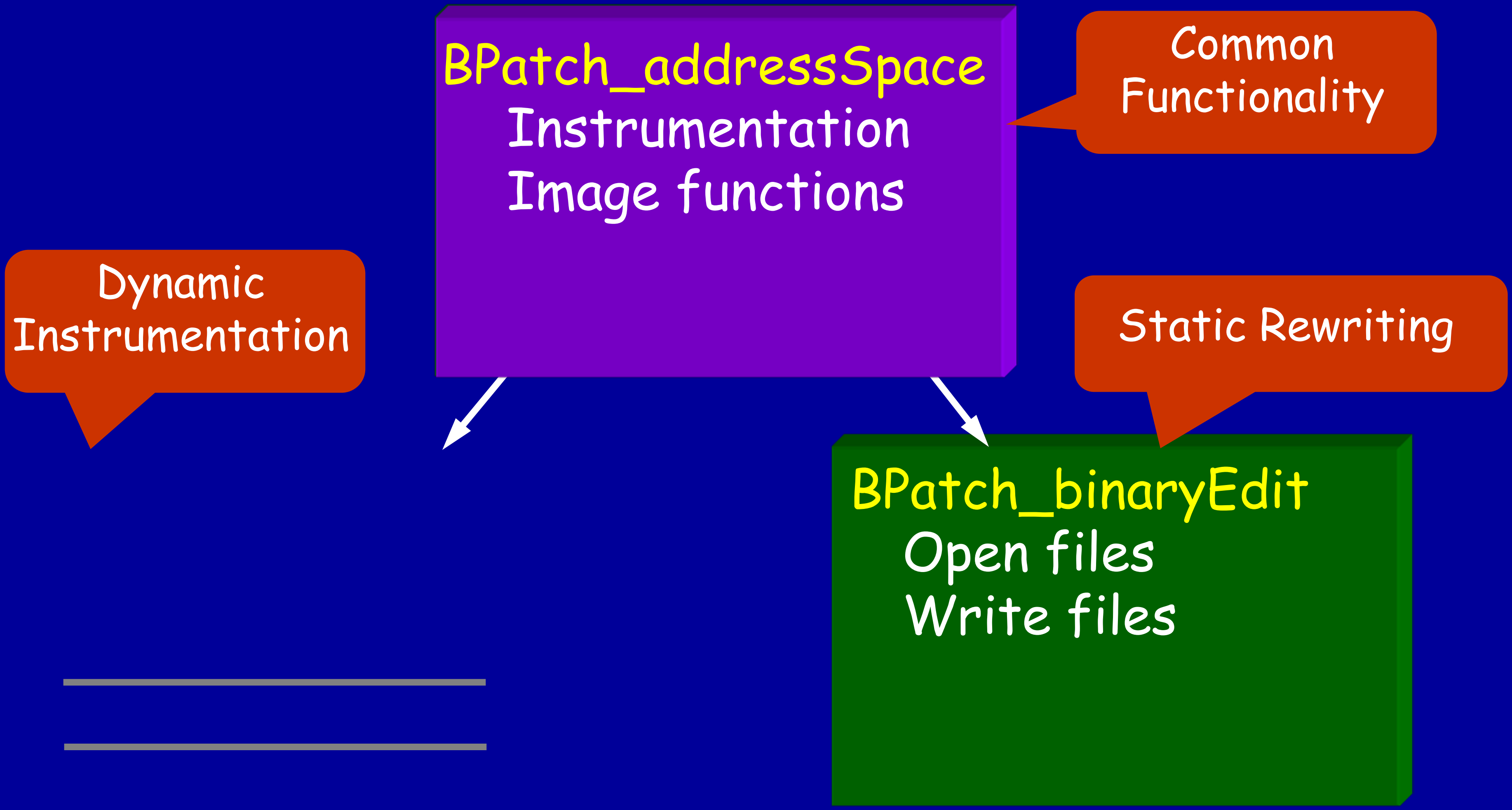
A Static Binary Rewriter

- Binary Rewriter Capabilities
  - Instrument once, run many times
  - Run instrumented binaries on systems without dynamic instrumentation (e.g. BlueGene).
  - Perform static analysis without running a binary

- Operates on unmodified binaries.
  - No debug information required
  - No linker relocations required
  - No symbols required

- Uses the same abstractions and interfaces as Dyninst.

The Binary Rewriter Interface



Bpatch_binaryEdit

- A set of libraries/executable (DSOs) that make up an app.
  - Each DSO represented by a Bpatch_module
  - Contains statically determinable libraries

- Can choose which DSOs to rewrite.
  - E.g, a single Bpatch_binaryEdit contains:
    - a.out
    - libc.so
    - libpthread.so
  - Choose to rewrite a.out and libc.so

Converting Code: Original

```
//Initialize
BPatch_process *a;
a = bpatch.createProcess(file, args);

//Real Work
BPatch_image *img = a->getImage();
BPatch_variableExpr *expr = a->malloc(4);
Bpatch_function *f = img->findFunction(...);
f->insertSnippet(...);

//Finalize
a->continueExecution();
while (!a->isTerminated())
   bpatch.waitForStatusChange();
```

Converting Code: Initialization

- Old:

```
//Initialize
BPatch_process *a;
a = bpatch.createProcess(file, args);
```

- New:

```
//Initialize
BPatch_addressSpace *a;
if (dynamic)
   a = bpatch.createProcess(file, args);
else
   a = bpatch.openBinary(file);
```

Converting Code: Finalization

- Old:

```
a->continueExecution();
while (!a->isTerminated())
   bpatch.waitForStatusChange();
```

- New:

```
if (dynamic) {
  BPatch_process *proc =
    dynamic_cast<Bpatch_process *>(a);
  proc->continueExecution();
  while (!proc->isTerminated())
     bpatch.waitForStatusChange();
} else {
  BPatch_binaryEdit *bin = ;
    dynamic_cast<BPatch_binaryEdit>(a);
  bin->writeFile(“outfile”);
}
```

Changes Relevant to O|SS

- No runtime events (pre-fork, thread create…)
  - Can instrument equivalent functions (fork entry, pthread_create, …)

- No oneTimeCode
  - Can do initialization and finalization by instrumenting main and exit
  - Or do initialization with library constructors
  - How to flush data on a signal (SIGSEGV) exit?

Changes Relevant to O|SS

- Rewriter can run on different machine than mutatee.
  - You want to do rewriting from front-end, not daemons.
  - Need to match processor architecture family between mutator/mutatee.
- Instrumentation must done a-priori, cannot be changed or removed.
  - Experiments must be done a-priori
  - O|SS's thread specific instrumentation implementation won't work.

Features

- Beta versions for Linux/x86, Linux/x86_64, CrayXT in Dyninst 6.0.
- Linux/PPC32, Linux/PPC64, BG/L, BG/P under development and available soon.
- ia64 not currently planned.
  - Needed for O|SS?
- Partial support for statically linked binaries
  - Cannot yet insert new library dependencies

Questions?

Matthew LeGendre
University of Wisconsin
legendre@cs.wisc.edu
http://www.paradyn.org