

Dyninst as a Binary Rewriter

Matthew LeGendre

University of Wisconsin

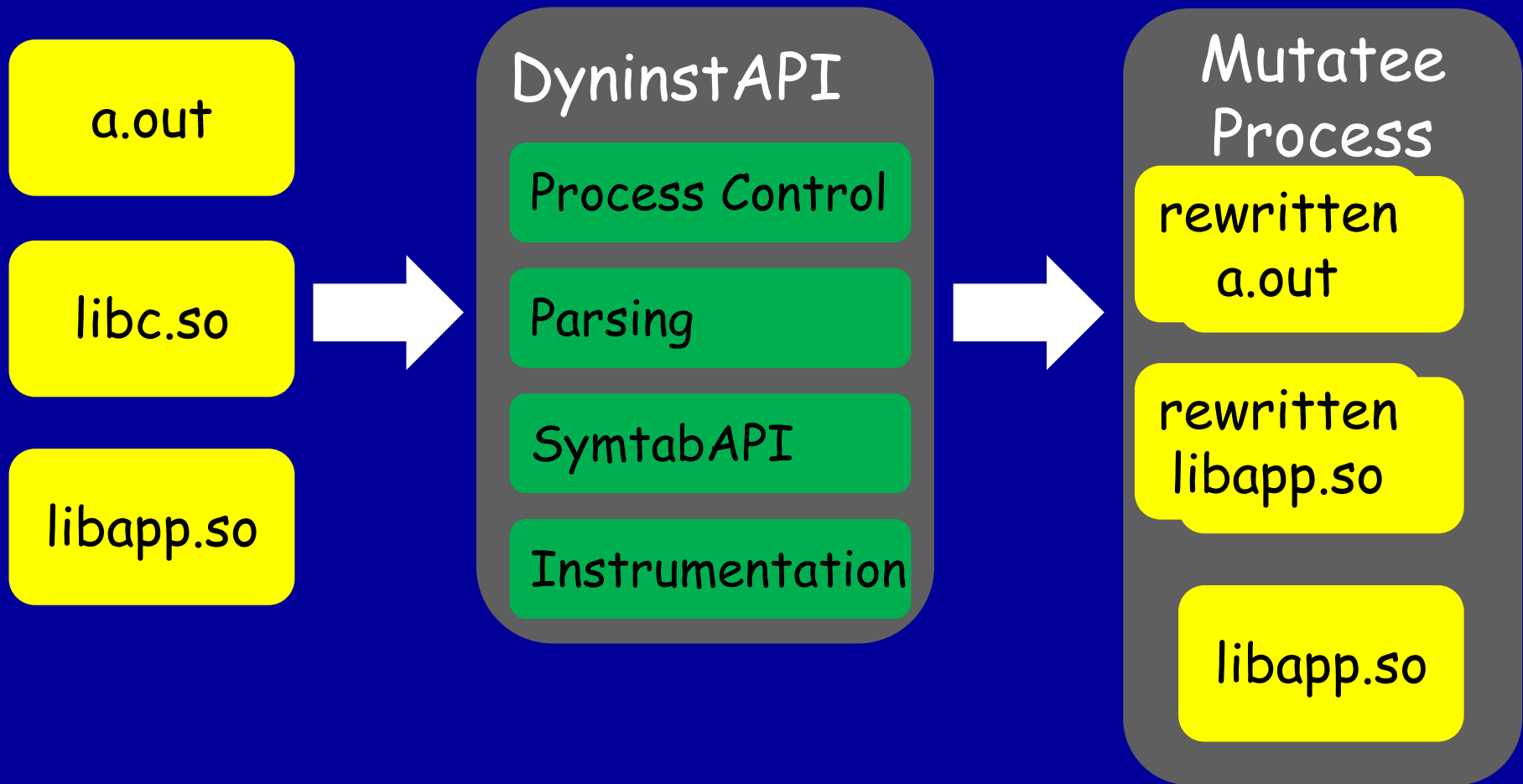
legendre@cs.wisc.edu

<http://www.paradyn.org>



April 2008

Static Binary Rewriting in Dyninst



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.

Static Vs. Dynamic Rewriting

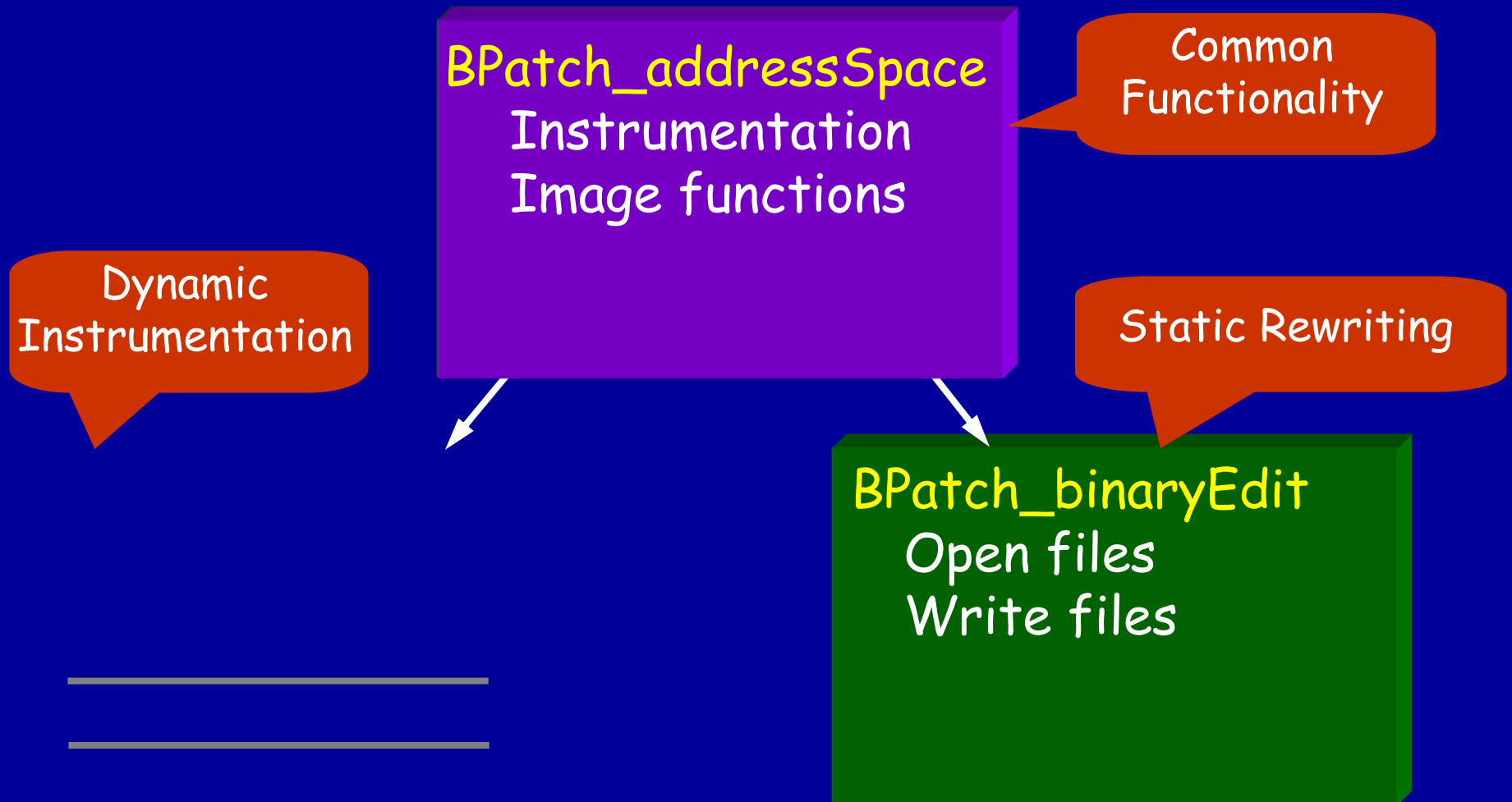
Static Rewriting

- ✓ Faster instrumentation insertion.
- ✓ Amortize parsing and instrumentation time across multiple runs.
- ✓ Easier to port.

Dynamic Instrumentation

- ✓ Insert and Remove instrumentation at run time.
- ✓ Execute instrumentation at a particular time (oneTimeCode).
- ✓ Respond to run time events (shared library loads, exec, ...).

The Binary Rewriter Interface



BPatch_addressSpace

- Use BPatch_addressSpace for static and dynamic code instrumentation.

```
if (use_bin_edit)
    addr_space = bpatch.openFile(...);
else
    addr_space = bpatch.attachProcess(...);

...

addr_space->getImage()->findFunction(...);
addr_space->insertSnippet(...);
addr_space->replaceFunction(...);
```

What's New in Dyninst's New Binary Rewriter

- Beta release in Dyninst 6.0 for Linux/x86, Linux/x86_64.
- Rewrite shared objects.
- Generate instrumentation that calls between shared objects.
- SymtabAPI Rewriter Support
- Windows rewriting support (coming in Dyninst 6.1).

Challenges in Static Rewriting

- **Allocate space** in for instrumentation and relocated code.
- **Insert libraries** as new binary dependencies
- **Generate intermodule** calls and data references in instrumentation
- **Preserve addresses** of original code and data objects
- **Gritty details** of writing ELF and PE files

Growing Sections

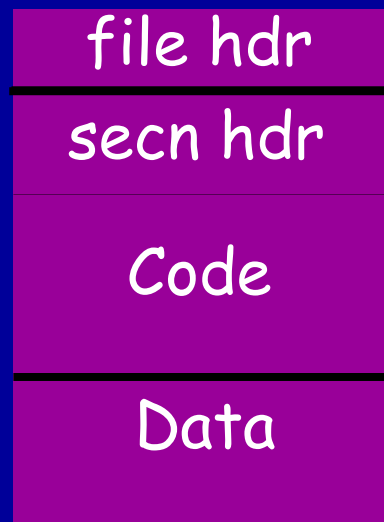
- Need to add new data to old sections
 - *Grow existing sections (dangerous).*
 - *Relocate copy of section to end of file.*



We shifted code and data!
Only works if binary has blank space we can fill in.

Growing Sections

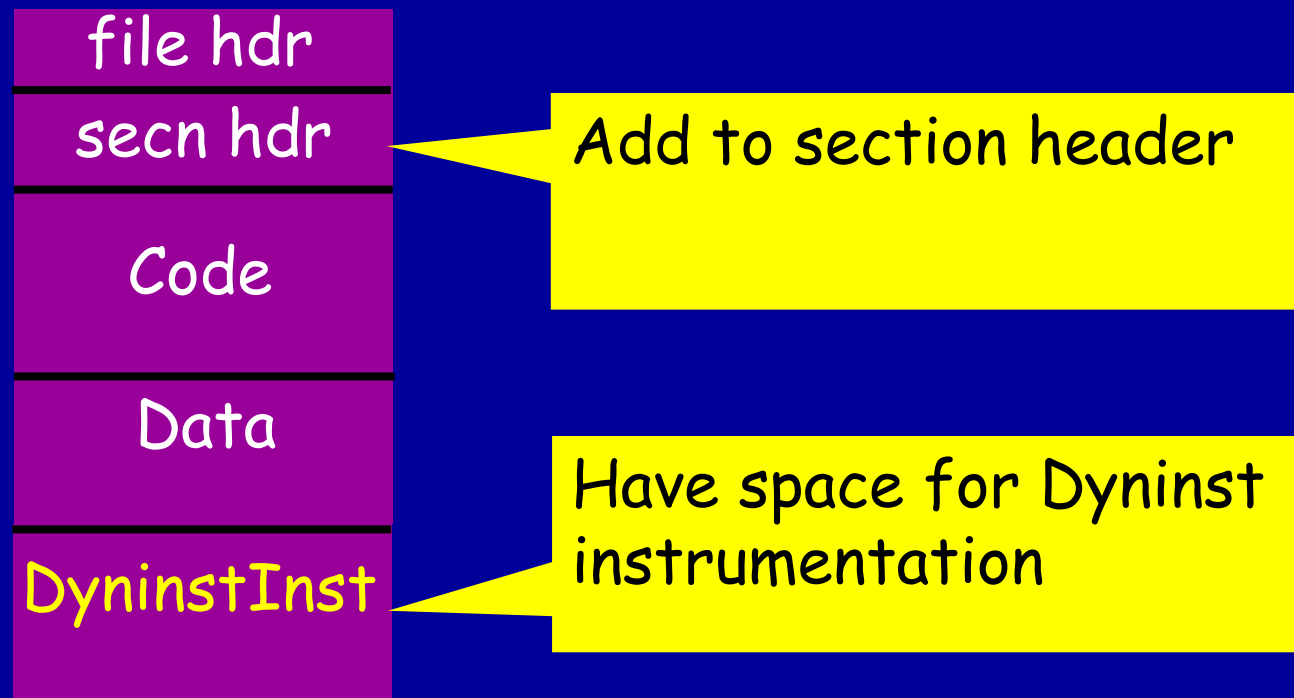
- Need to add new data to old sections
 - Grow existing sections (dangerous).
 - Relocate copy of section to end of file.



Code and data are unmoved
Have to update pointers to
section header

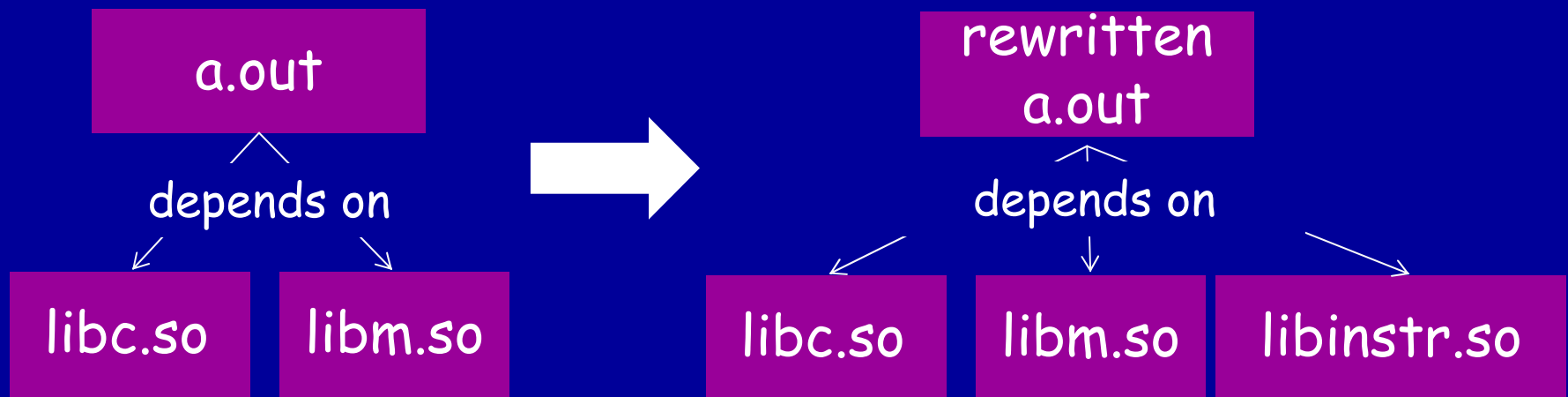
Adding Space

- Need space in binary for instrumentation, relocation and new copies of object file tables.
 - Add section and entry to section header table



Adding Libraries

- Add new libraries as dependencies
 - Typically used for instrumentation support.
 - Add new library to existing list of dependencies.



- Need to grow section that contains dependency lists.

Intermodule Calls

- Need to generate inter-module calls and data references in instrumentation.
 - E.g, instrumentation in a.out calls printf
 - Trivial in dynamic rewriting, hard in static rewriting.
- Output data structures to tell dynamic linker to link instrumentation call site to target function.

Intermodule Calls

- Use dynamic linker to patch intermodule calls
 1. **Allocate space** that will contain printf's address.
 2. Generate **instrumentation** that **makes an indirect call** through this address.
 3. Add a relocation that tells the **dynamic linker** where to **write the address of printf**.

①

dyninstInst

00000000

②

ld **addr** -> r1
call *r1
...

③

Relocation Entries

{ printf, **addr** }

Intermodule Calls (Linux)

- Adding a relocation requires:
 - **New relocation in .rel.dyn**
 - Lists relocations to be applied at load time.
 - **New symbol in .dynsym**
 - Symbols that are referenced by dynamic loader
 - **New string in .dynstr**
 - Contains names of symbols from .dynsym
 - **New entry in .hash**
 - A hash table for fast lookup of names into dynsym
- Copy modified sections to end of binary

Intermodule Calls (Windows)

- Adding a patch requires:
 - **New entry to Import Table**
 - Lists externally referenced libraries and functions
 - **New entry to Import Address Table**
 - Holds addresses of externally referenced functions, filled in at runtime.
- Move Import Table to end of binary
- Split Import Address Table into old and new sections.

The Devil in the Details

- Have to update pointers to moved sections.
- Store and regenerate old sections.
- Account for differences between standards and implementations.
 - Linux didn't let us move program headers to end of file.
- Reconstruct sections indirectly affected by our rewriting.
 - hash and symbol versioning sections on ELF.

SymtabAPI Rewriting

- Binary rewriting functionality available through SymtabAPI
 - Open existing binary
 - Add new symbols
 - Add library dependencies
 - Add new code and data regions
 - Add intermodule references
 - Modify existing code and data
 - Write binary

SymtabAPI Rewriting

- Add a function symbol to a binary:

```
/* Open a file */
```

```
Symtab *synt;
```

```
Symtab::openFile(synt, "a.out");
```

```
/* Add Symbol */
```

```
synt->createFunction("func1" /*name*/,  
                    0x1000 /*offset*/,  
                    100 /*size*/);
```

```
/* Write new binary */
```

```
synt->emit("rewritten.out");
```

Performance Comparison

- Instrument every block in SPECINT 2006 gcc:
Static: 0.0s Dynamic: 0.0s
 - Dynamic slow because of app startup and ptrace overhead.
- SPECINT 2006 gcc runtime:
Base: 0.0s Static: 0.0s Dynamic: 0.0s
 - Static slow because of overheads in inter-module calls

Future Work - Static Binaries

- Insert library into statically linked binaries
 - Static binaries especially common in HPC.
 - No existing infrastructure in static binaries for loading libraries.
- Ideas
 - Append inserted library to end of static binary.
 - Have Dyninst resolve inter-module references.
 - But what if original binary is stripped?

Future Work - Ports

- Windows/x86 under development
- Elf platforms
 - Linux PPC-64 & IA-64
 - Solaris/Sparc
- AIX support
 - Needs significant work for XCOFF rewriting

Questions?

Matthew LeGendre

University of Wisconsin

legendre@cs.wisc.edu

<http://www.paradyn.org>