

# Tool Componentization and Experiences with Dyninst

Matthew LeGendre

University of Wisconsin

[legendre@cs.wisc.edu](mailto:legendre@cs.wisc.edu)

<http://www.paradyn.org>



April 2008

# We're Rebuilding the Same Infrastructure

- Many HPC tools need:
  - Scalability
  - Data Collection/Instrumentation
  - Job Management and Launching
  - Visualization
  - Data Analysis
  
- We could share a lot of common infrastructure

# Sharing Components Between Tools

Performance  
Analysis

Debugger

Object Parser

Scalability

Debug Interface

Static Analysis

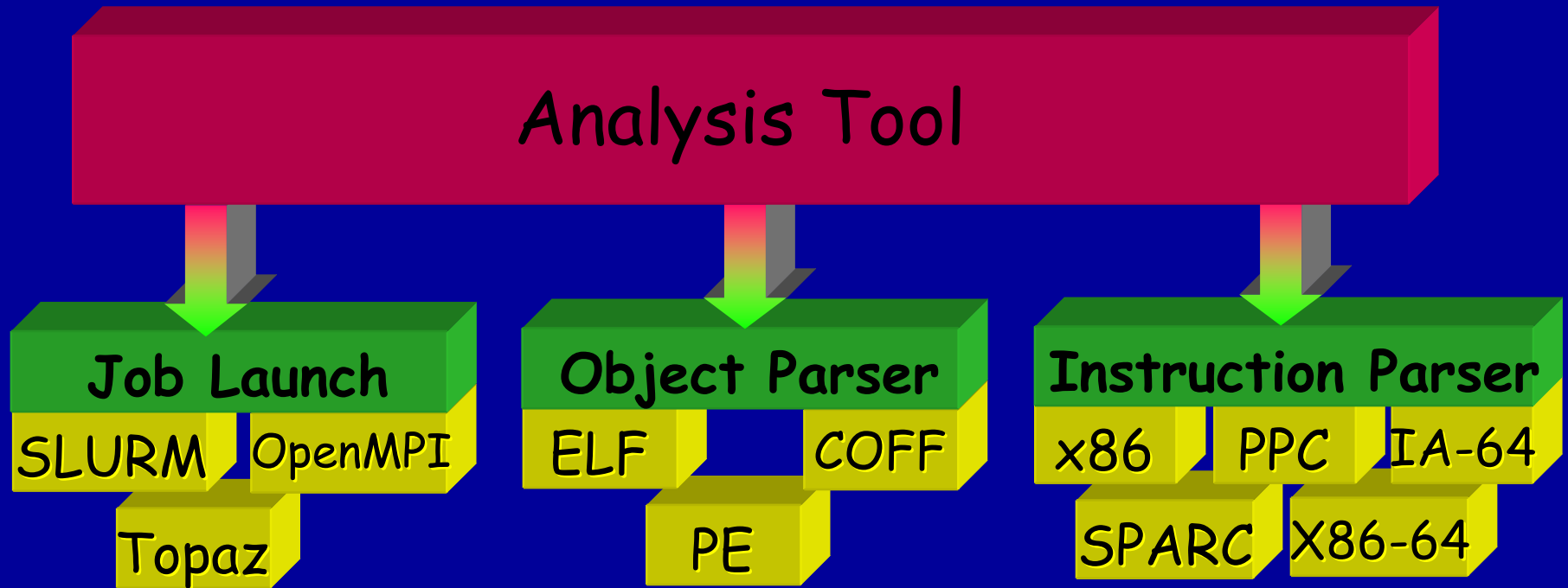
Object Parser

Instruction  
Disassembler

Analysis

# Abstract Interfaces

- Use a simplified interface to hide complexity and cross-platform development issues.
  - Interfaces need to provide abstractions, not just access to platform specific data structures.

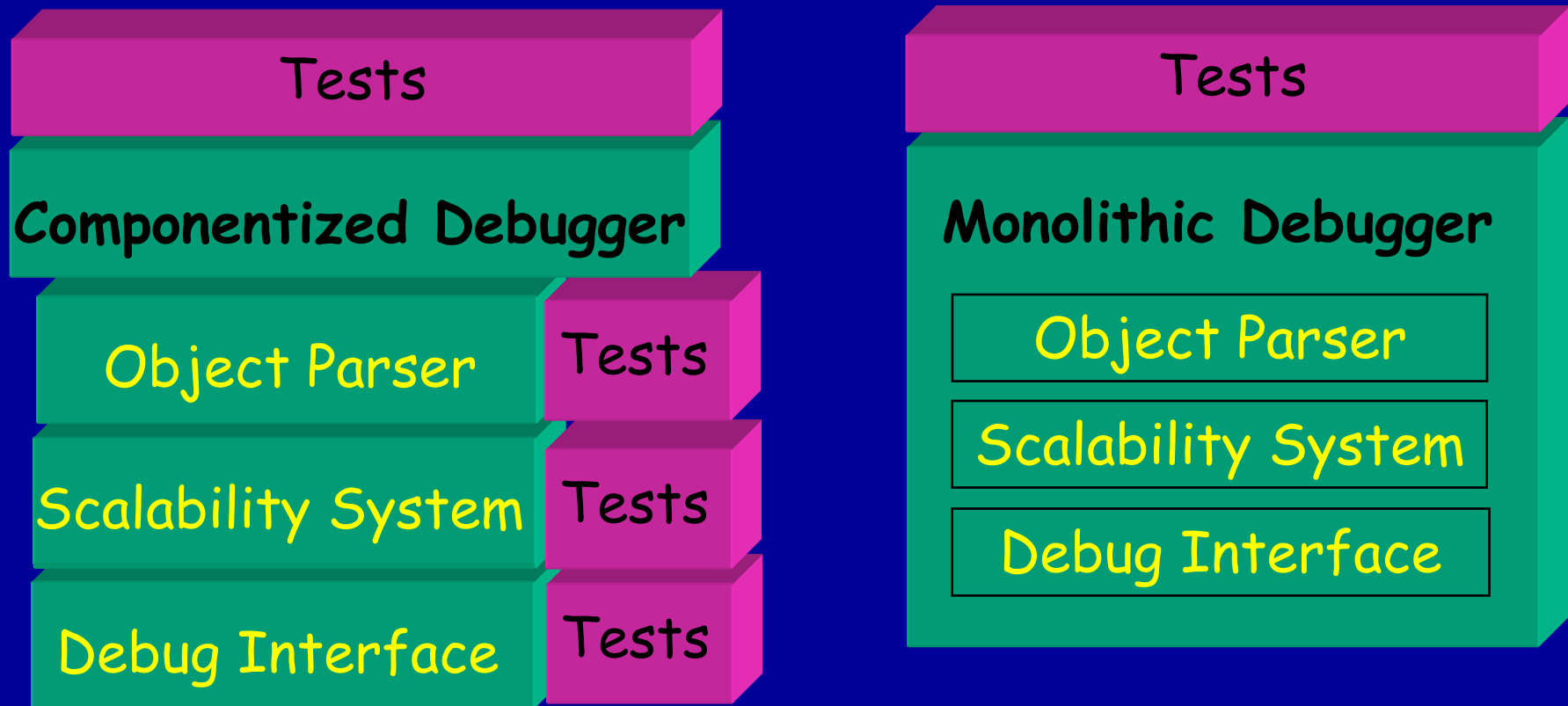


# Small, Independent Components

- Components provide specific pieces of core infrastructure.
  - Larger, monolithic components lack flexibility
  - Smaller components can be combined
- Components should be able to stand-alone.
  - Tightly integrated components suffer the same problems as monolithic software.
  - E.g. A visualizer that requires a specific data collector.

# Testing Components

- Well componentized software is easier to test

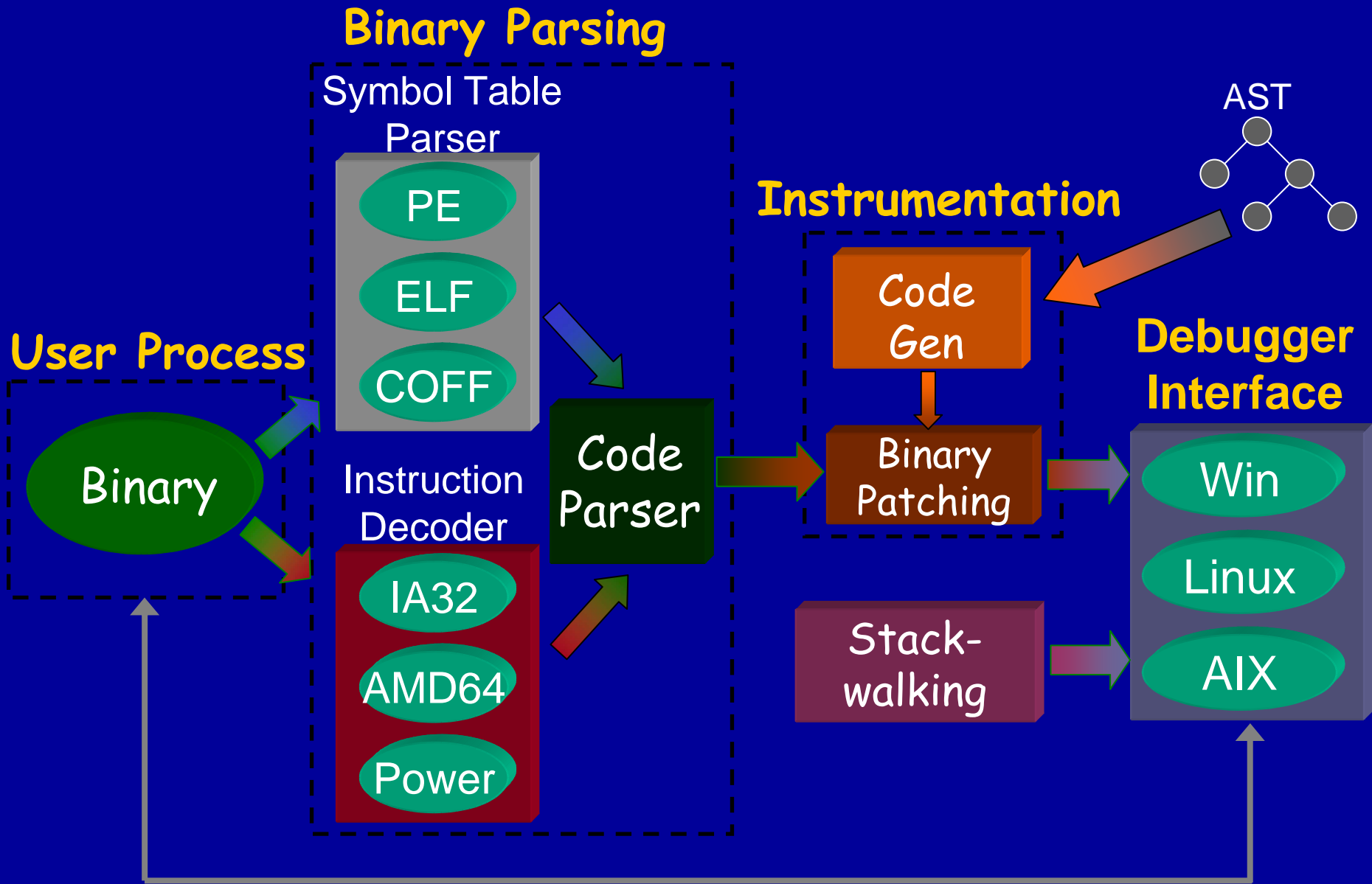


# Licensing and Support

- Users want to know that relying on your components won't leave them stranded.
  - Open Source software allows people to fix bugs and add features they need.
  - Reliable support allows users to trust the software you're giving them.

# Our Starting Point: Dyninst

- A platform-independent library for machine level code patching.
  - Functions for binary code analysis
  - Functions for binary code patching
- Clean abstractions to encapsulate the tool complexity.
- Originally designed as part of the Paradyn performance profiling tool, but now widely used in many tools, including Open|SpeedShop.



# Dyninst and Paradyne Components

- Splitting **Dyninst** up into component libraries:
  - **SymtabAPI**: parses symbol and debug information from object files
  - **StackwalkerAPI**: walks call stacks
  - **InstructionAPI**: disassembles byte sequences into instructions
- Component libraries from **Paradyne**:
  - **MRNet**: provides scalable communication for distributed tools

# SymtabAPI

- Cross-platform interface for reading and writing symbol and debug information.
  - ELF, XCOFF, PE (Linux, Solaris, AIX, Windows).
- Unifies the data found under different file formats into a single high-level abstraction.
  - E.g. Associates local variables with function symbol.
- Release SymtabAPI 2.0 in Fall '07

# StackwalkerAPI

- Cross-platform API for collecting first and third party stackwalks.
- Callback interface allows users to plug in their own stack walking mechanisms, e.g:
  - Walking through non-standard stack frames created by optimized functions.
  - Use stackwalking debug information provided by another library
- Beta version is currently available

# InstructionAPI

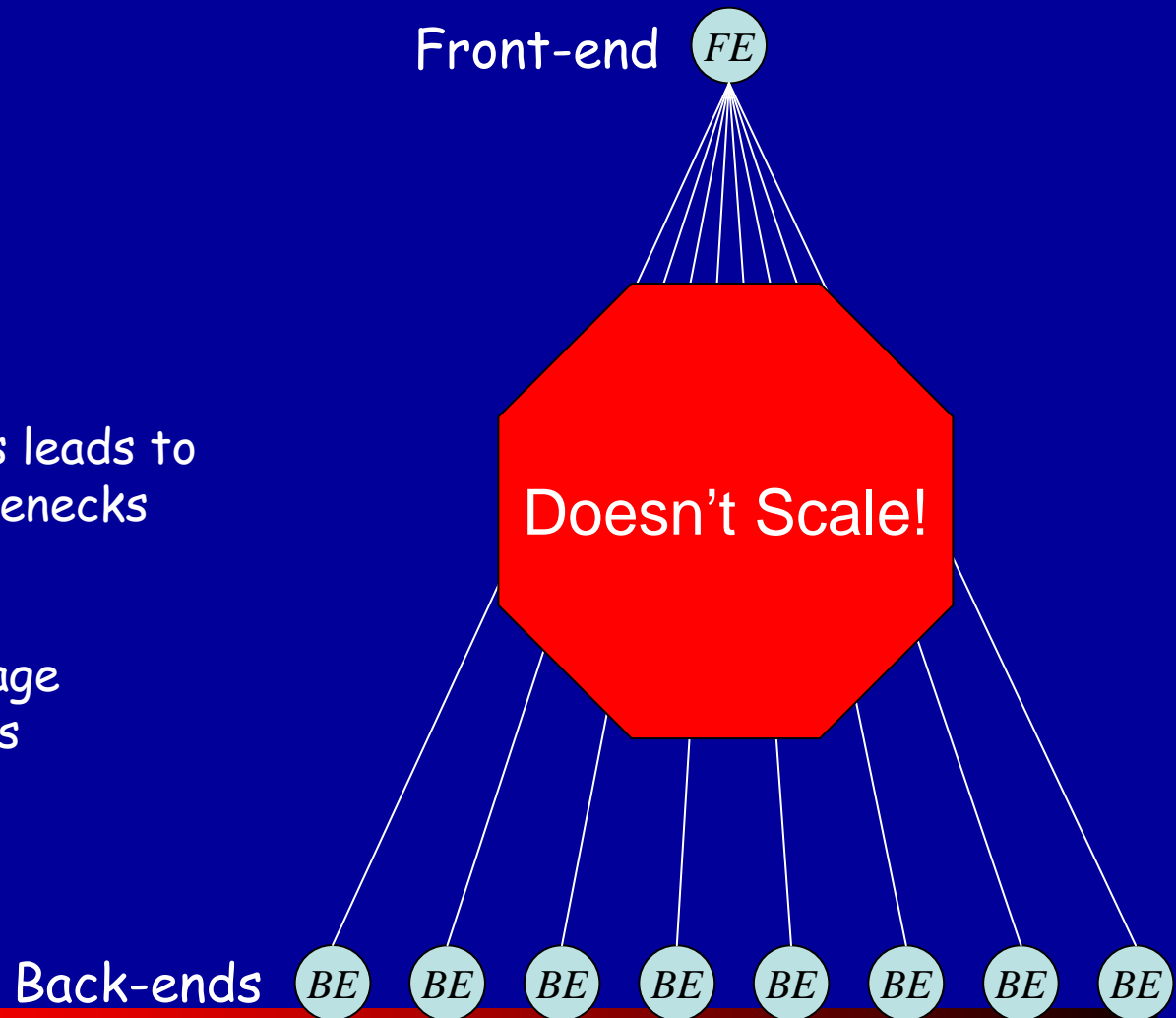
- Decodes machine code into abstract, platform independent, instruction representation
- Interface allows straightforward data flow and control flow analysis
  - Query interface is designed for analysis, e.g.:
    - Control flow targets
    - Registers read/written
    - Memory addresses accessed
- Instruction semantics are modeled with ASTs of operations

# MRNet

- Provides infrastructure for scalable communication between a front end and back ends.
- Tree Based Overlay Network (TBON) model works with tools, applications.
- Challenges have involved providing an interface flexible enough for users needs.

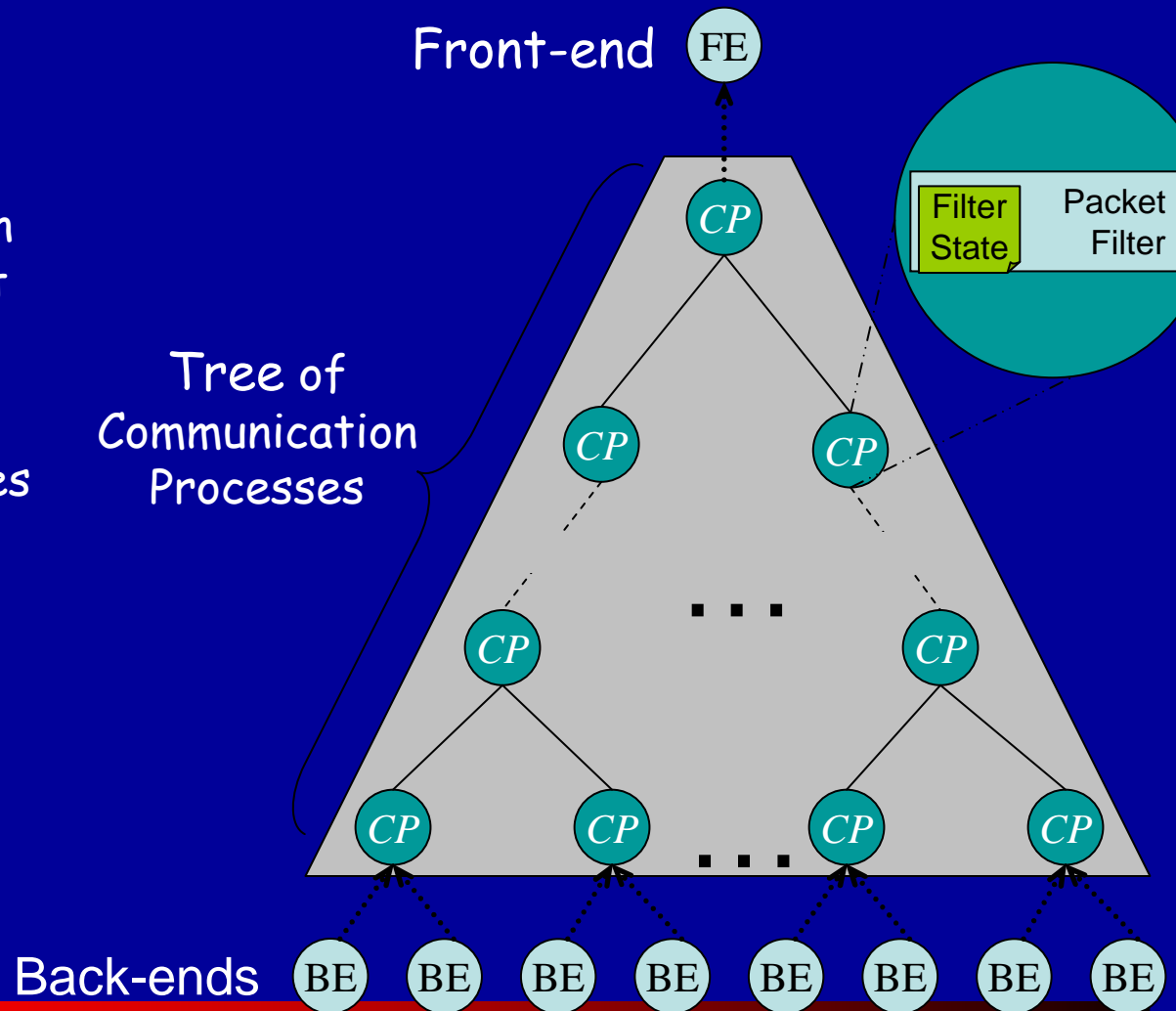
# Tree Based Overlay Networks

- Data managements
  - Large volumes
- Data analysis
  - Centralized analysis leads to computational bottlenecks
- Many resources to manage
  - E.g. control channels



# Tree Based Overlay Networks

- Hierarchical decomposition for scalable data multicast and aggregation
- Flexible, dynamic topologies
- User-defined filters
- Trade-off: extra processing nodes for performance



# Conclusions

- Tool componentization allows for
  - Easier development
  - More flexible tools
- Challenges include
  - Building good interfaces
  - Identifying component granularity
- Dyninst/Paradyn Componentization
  - Generated SyntabAPI, StackwalkerAPI, InstructionAPI, MRNet
  - Starting to see wide use

# Questions?

Matthew LeGendre

University of Wisconsin

[legendre@cs.wisc.edu](mailto:legendre@cs.wisc.edu)

<http://www.paradyn.org>