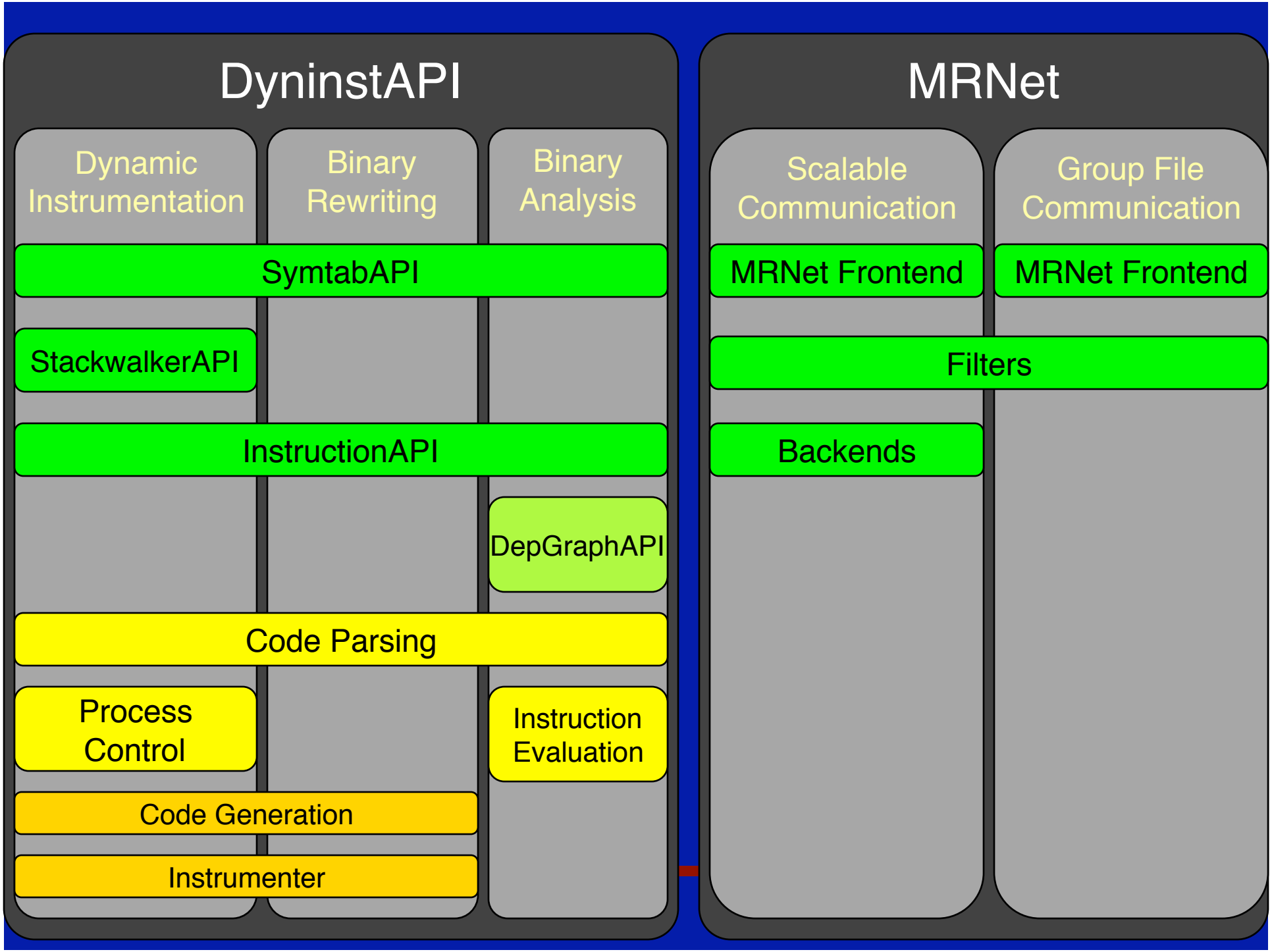
The Latest and Greatest in the Dyninst Binary Code Toolkit

Madhavi Krishnan
Matthew LeGendre
Bill Williams

Paradyn™


DyninstAPI
Dynamic Instrumentation
Binary Rewriting
Binary Analysis
SymtabAPI
StackwalkerAPI
InstructionAPI
DepGraphAPI
Code Parsing
Process Control
Instruction Evaluation
Code Generation
Instrumenter
MRNet
Scalable Communication
Group File Communication
MRNet Frontend
MRNet Frontend
Filters
Backends

Dyninst Components

- Updates on existing components
  - SymtabAPI
  - InstructionAPI
  - StackwalkerAPI
- New component
  - DepGraphAPI
- Proposed new components
  - Parsing
  - Instruction Semantics
  - Process Control

Updates

- SymtabAPI 6.0
  - New: binary modification interface
  - New: function, variable abstractions
- InstructionAPI 1.0
- StackwalkerAPI 1.0
  - Initial releases

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 *symt;
Symtab::openFile(symt, "a.out");

/* Add Symbol */
symt->createFunction("func1" /*name*/,
                     0x1000 /*offset*/,
                     100 /*size*/);

/* Write new binary */
symt->emit("rewritten.out");
```

DepGraphAPI

- A little static analysis can save a ton of instrumentation
- What if we need more detail than a CFG?
  - Example: stack pointer aliasing
- Construct dependence graphs to solve these problems

Example: Stack Pointer Aliasing

```
/* Build the PDG for a function */
PDG::Ptr pdg = PDG::analyze(func);

/* Find the node for initial SP */
pdg->find(entryAddr, sp, nodeBegin, nodeEnd);

/* Find the forward slice using the SP */
pdg->forwardClosure(*nodeBegin, sliceBegin,
                    sliceEnd);
for( ; sliceBegin != sliceEnd; ++sliceBegin)
   /* Do stuff */
```

Paradyn

DepGraphAPI Features

- Current
  - Builds data dependence graph based on:
    - Register reads/writes
    - Stack reads/writes
  - Builds control dependence graphs
  - Builds program dependence graphs
- Future
  - Tuning: precision vs. speed
  - *Other feature requests?*

Isolating Dyninst's Parsing

Dyninst Functionality

High-level Dyninst

instrumentation, process control, etc.

Low-level Dyninst

gritty binary code details

SymtabAPI, InsructionAPI

Map binary code to useful structures

CFG creation

Function lookup

Resolving indirect control flow

A parsing component?

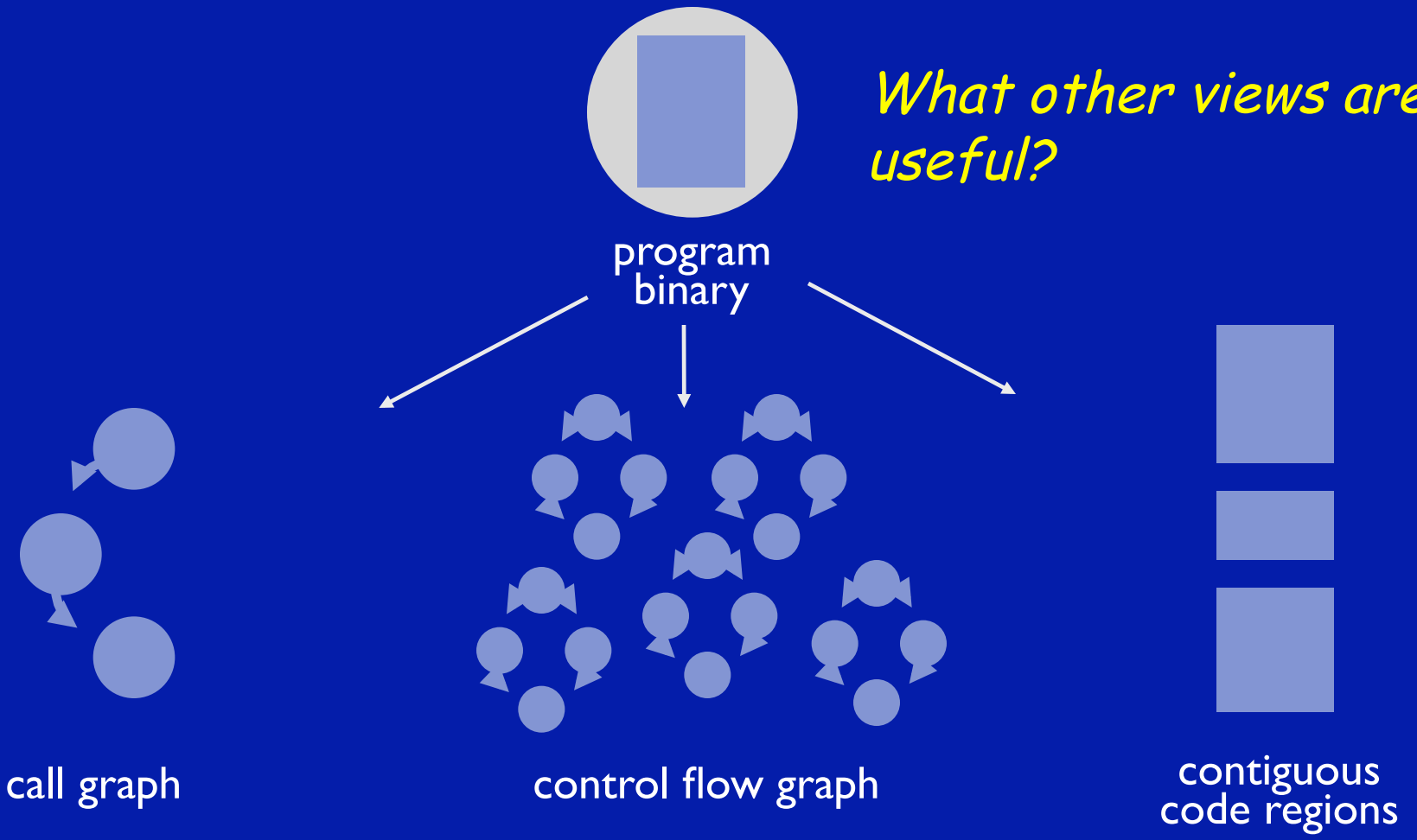
ParsingAPI Features

- Builds CFG from binary code
- Fine-grain lookup interface
  - functions, basic blocks, instructions
- Engineered to support "weird" binaries
  - Highly optimized code
  - Stripped binaries
- "Views"
  - (more on this in a moment)
  - Easily updatable representations

Paradyn

Binary Code Views

Support multiple abstractions of code



What other views are useful?

Instruction Semantics

- Detailed analysis beyond InstructionAPI

- Example: stack height analysis
  - Effect of push/pop on SP
  - Effect of stack arithmetic
  - Aliasing

Design: Open Questions

- Internal: we evaluate what it does
  - Symbolic evaluation
- External: we describe what it does
  - Transfer function
  - ASTs

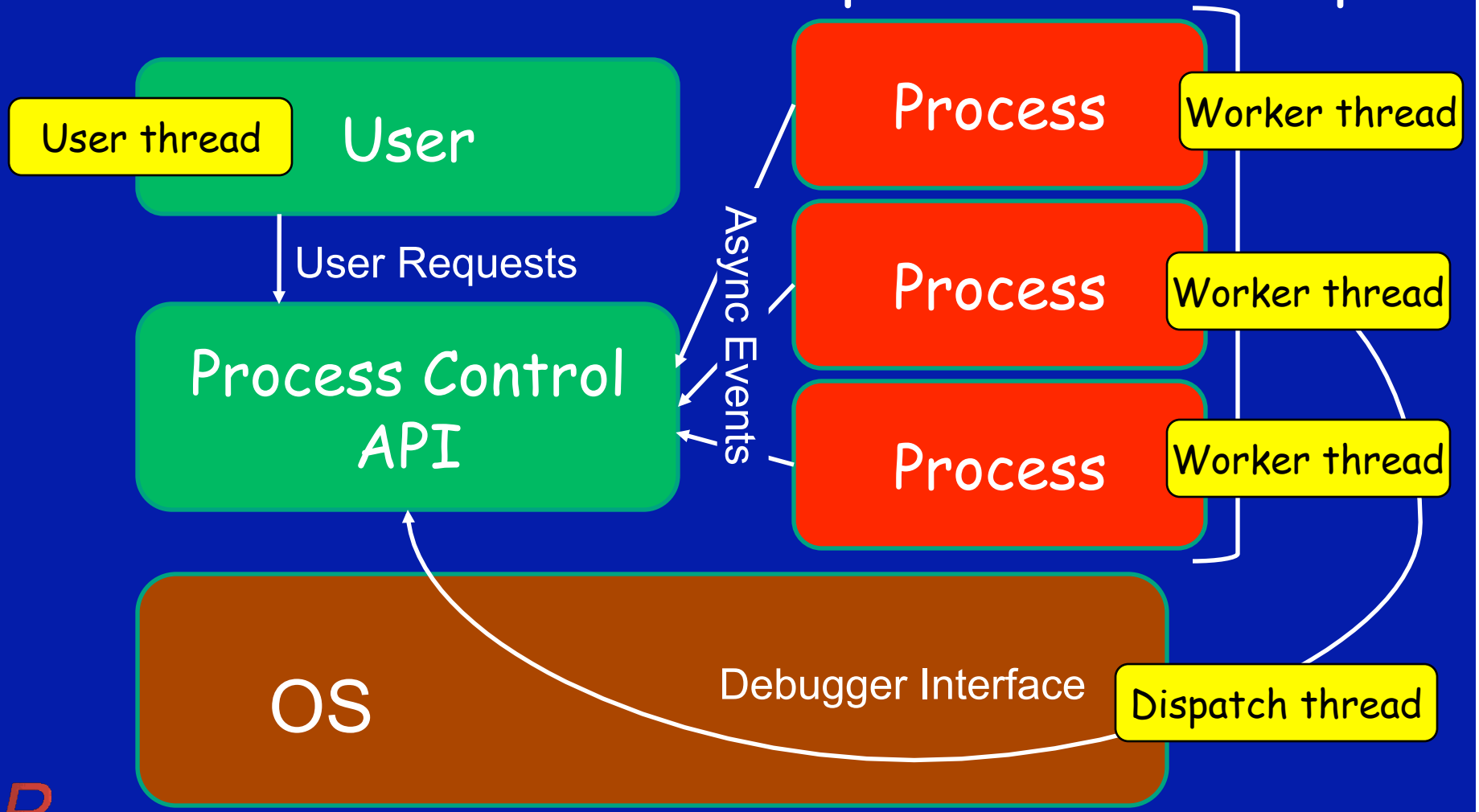
- *What views are useful to the community?*

Process Control: Goals

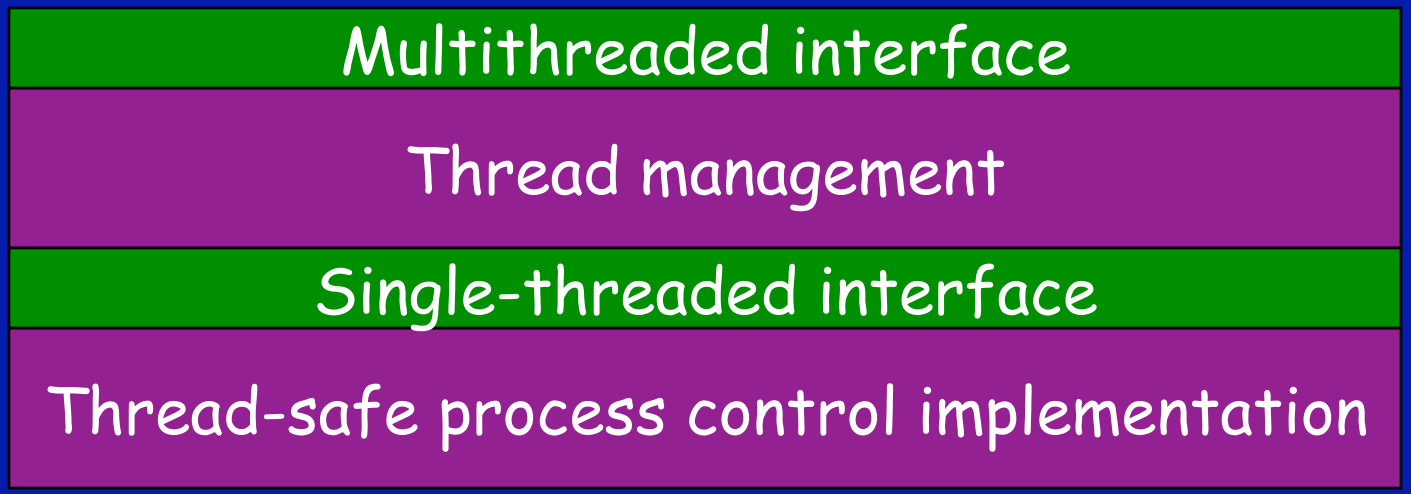
- Develop an API to manage processes and events
  - Control the process:
    - Start/stop
    - Attach/detach
    - …
  - Modify the process:
    - Read/write address space
  - Monitor the process
    - Fork/exec
    - Thread create/destroy
    - Library load/unload
    - Signals
  - …
- Use OS's debugger interface.

Threading Challenges

- Will have to deal with multiple sources of input



Proposed Architecture

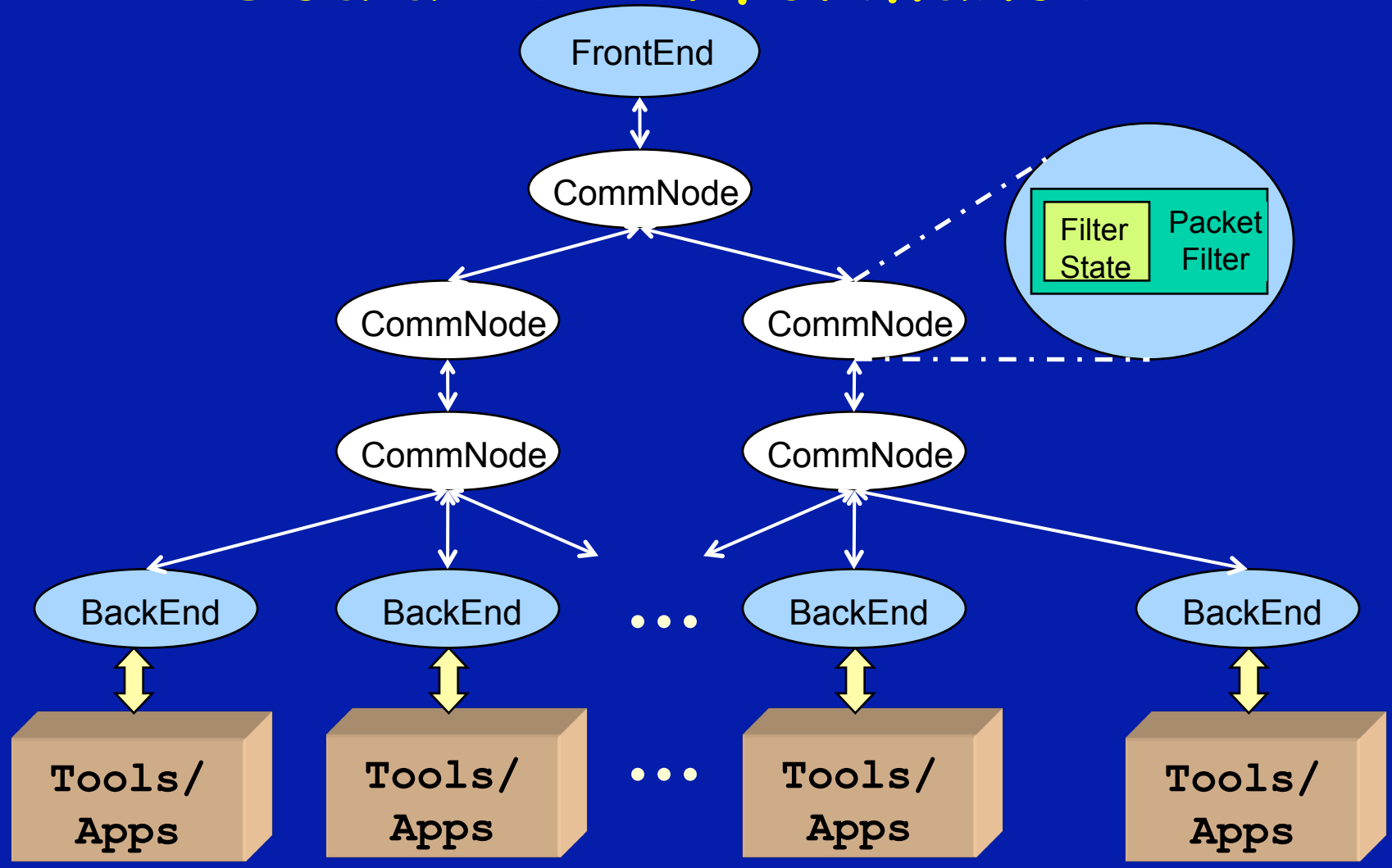


Bases for Implementation

- DyninstAPI implementation
  - Supports Linux, AIX, Solaris, Windows
  - Already has a working (but complex) threading model
  - Old and well tested
- StackwalkerAPI's debugger interface
  - Supports Linux, BlueGene
  - Simpler design
  - Already has a component interface
- Likely to build something that descends from both

Questions?

Tree-based Overlay Networks for Scalable Performance



Paradyn

MRNet Features

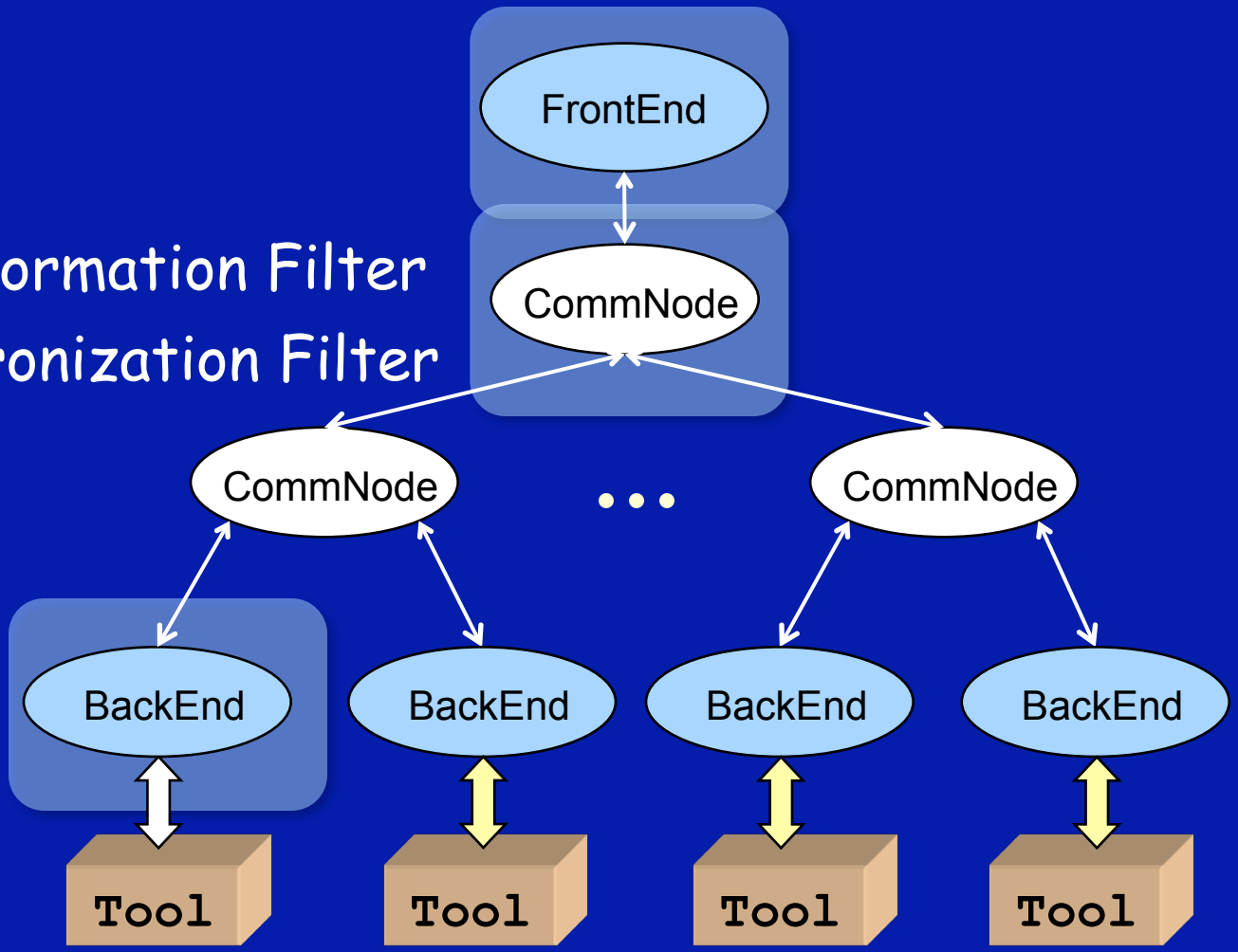
- Scalable multicast and aggregation
- Flexible topologies
- Reliability during node failures
- Filters:
  - Dynamically configurable
  - Stateful
- Built-in and User-defined filters:
  - Transformation Filters
  - Synchronization Filters

Paradyn™

Example: Tracing Tool

Transformation Filter

Synchronization Filter



Programming with MRNet: Front End

```
/* Setup */
network = new Network( topology_file, backend_exe, …);
transFilter = network->load_FilterFunc ("FuncTraceFilter",… );
syncFilter = network->load_FilterFunc ("SyncTraceFilter",… );
comm = network->get_BroadcastCommunicator( );
stream = network->new_Stream(comm, transFilter, syncFilter,…);

/* Send */
tag = PROT_START_TRACE;
stream->send( tag, "%d", type_func_trace)

/* Receive */
retval = stream->recv(&tag, packet);
char **func_trace; int func_trace_len;
packet->unpack( "%as", &func_trace, &func_trace_len) ;

storeFunctionTrace(func_trace, func_trace_len);
```

Programming with MRNet: Back End

```
/* Setup */
network = new Network(...);

/* Receive Request*/
network->recv(&tag, packet, &stream);

/* Process Request */
switch(tag){
    case  PROT_START_TRACE:
       packet->unpack("%d", &trace_type);
        collectTrace(trace_type, &trace);
       /* Send */
       stream->send(tag, "%as", trace);
       break;
    case ...:
}
```

Tracing: Transformation Filter

```
void FuncTraceFilter( packets_in, packets_out,
                      filter_state, config_params)
{
  /* Receive and Process Input Packets */
  for (i=0; i < packets_in.size(); i++) {
   cur_packet = packets_in[i];
       cur_packet->unpack("%as", &trace);
   mergeTraces(&trace);
  }

  /* Send Output Packet */
  PacketPtr new_packet = new Packet (trace, …);
  packets_out.push_back(new_packet);
  return;
}
```

Tracing: Synchronization Filter

```
void SyncFilter( packets_in, packets_out,
                 filter_state, config_params)
{
   /* Get saved packets from filter state*/
   batch_size = getBatchSize(config_params);
   packets = getPrevPackets(filter_state);
   packets.push_back(packets_in);

   /* Batch up packets */
   if( packets.size() >= batch_size ) {
      packets_out.push_back(packets);
      packets.clear();
   }

   updateFilterState(filter_state, packets);
   return;
}
```

Group File Operations

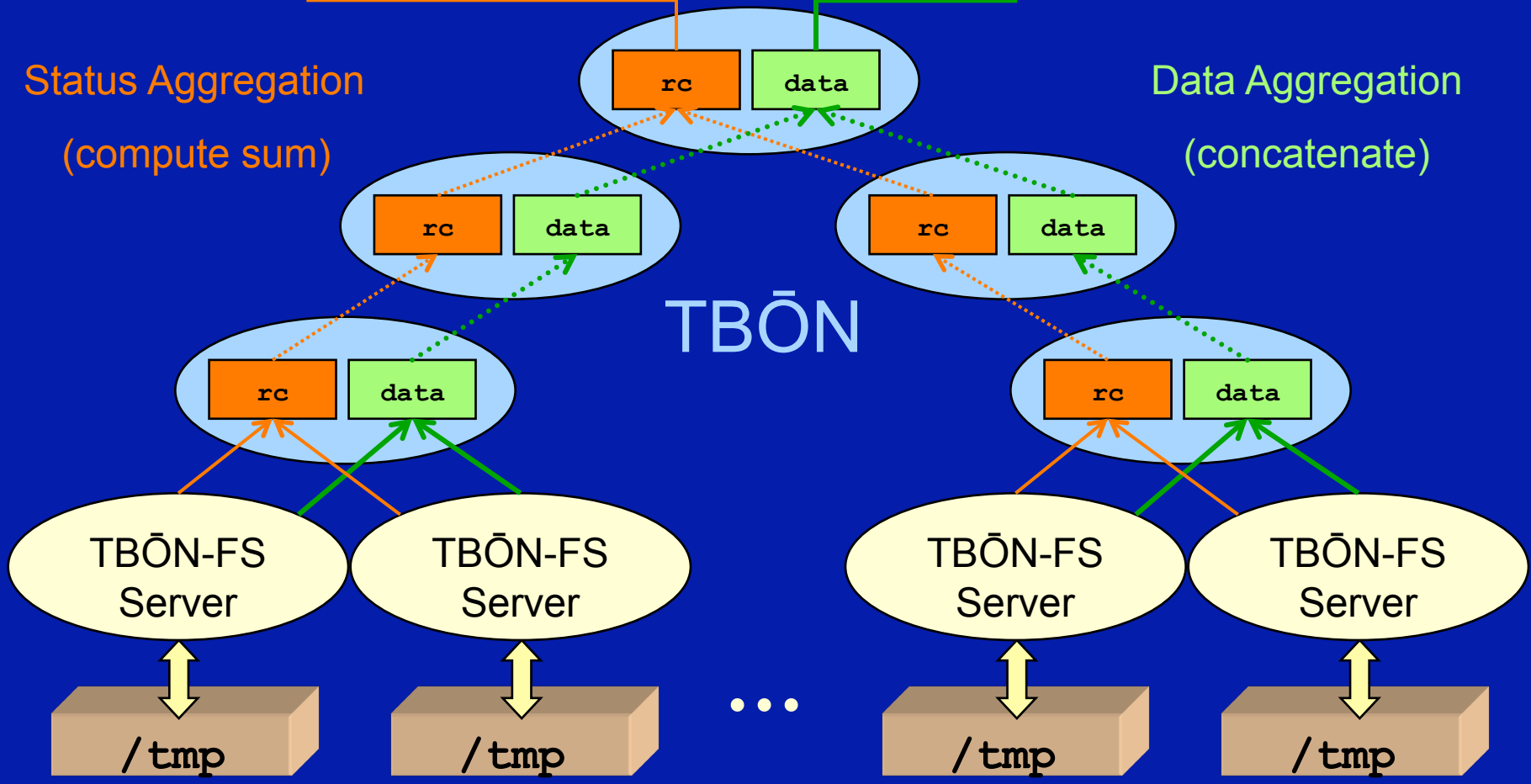
```
gfd = gopen(char* dir, flags)
```

- Directory: a natural file system group abstraction
- Operating on Groups
  - Pass group file descriptor to file operations
  - Explicit aggregation of group file operation results
- Fit existing interfaces

- Scalable access and operations
  - TBŌNs are scalable => TBON-FS

TBŌN-FS: Scalable Group File Operations

```
int rc = read(gfd, databuf, 1024)
```



Example Group File Operations

- Identify equivalent files at BEs
  - group read to checksum files or compare contents
  - e.g., group equivalent binary executables
- Analyze trace log files
  - group read fixed-size records
  - custom trace aggregation
- Distributed Debugger
  - group write breakpoint
  - group read process memory
    - variable value equivalence class

Paradyn™

Scalable Distributed Monitoring

```
ntop - Thu Apr 24 22:46:20 2008
1024 hosts up 96430.11 days, load average: 0.27, 0.11, 0.08
Tasks: 338112 total, 4296 run, 333816 sleep, 0 stopped, 0 zombie
CPU: 4096 cpu(s), 78.72% user, 0.86% sys, 0.00% nice, 19.65% idle, 0.76% wait
Mem:  84418395552k total,1059192128k used,7382647424k free, 169089408k buffers
Swap: 17182572544k total,  71227968k used,17111344576k free, 200214464k cached
    USER              %CPU              %MEM              COMMAND
-----------    --------------    --------------    -----------------
       briml        1.52 @4096        0.05 @4096    tbonfs-server
        root        0.01 @928         0.00 @928     ksoftirqd/1
        root        0.01 @884         0.00 @884     ksoftirqd/2
        root        0.01 @444         0.00 @444     elan4_mainint
        root        0.01 @524         0.00 @524     ksoftirqd/0
        root        0.01 @812         0.00 @812     ksoftirqd/3
        root        0.00 @1016        0.00 @1016    events/0
        root        0.00 @1020        0.00 @1020    khelper
        root        0.00 @1012        0.00 @1012    kthread
        root        0.00 @1016        0.00 @1016    kblockd/0
        root        0.00 @1020        0.00 @1020    kacpid
        root        0.00 @1020        0.00 @1020    ll_ping
        root        0.00 @1020        0.01 @1020    lrmmond
        root        0.00 @752         0.00 @752     irqbalance
        root        0.00 @68          0.00 @68      kqswnal_sched
        root        0.00 @1004        0.00 @1004    ldlm_cn_14
        root        0.00 @1008        0.00 @1008    ldlm_cn_15
```

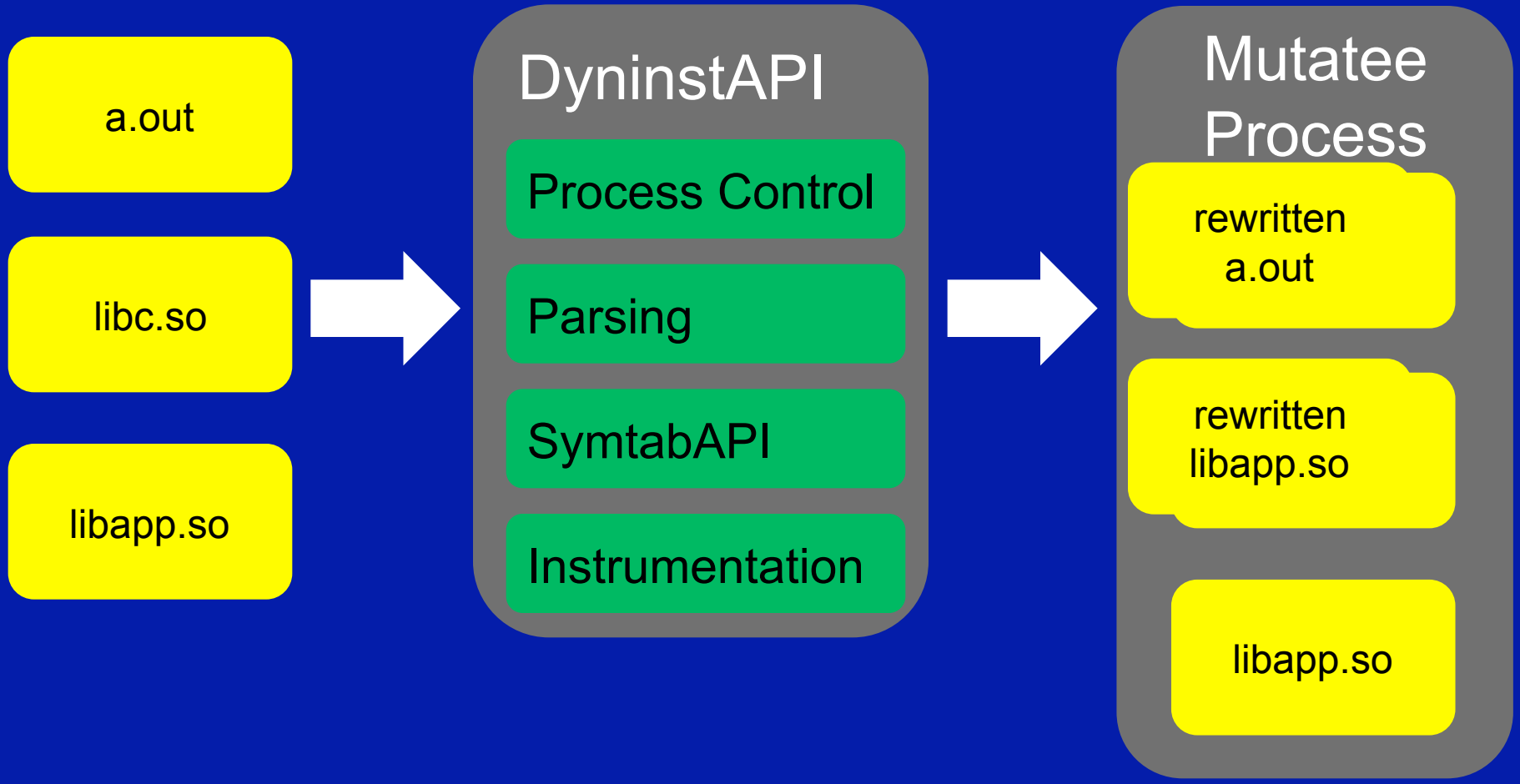
Avg. %MEM
4096 processes

Case Studies

- Parallel Linux Tools
  - `ptop` : observe resource utilization
  - `pgrep` : inspect file contents
  - `ptail -f` : follow file activity
  - `pcp, psync` : duplicate files
- Ganglia distributed monitor

Questions

Static Binary Rewriting in Dyninst



Features

- Same Interface
- Instrument shared objects and executables
- Add new libraries to rewritten shared objects
- Generate calls between shared objects

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

- Convenient for doing static binary analysis.

Paradyn™

PLACEHOLDER SLIDE

- Run Demo here:
  - Rewrite emacs and associated .so's to generate an OTF trace.
- Major Points to illustrate:
  - How easy it is to use
    - cat mutator source, should be fairly small
    - Note that it uses the regular Dyninst interface
  - How the binary changed
    - Readelf/ldd on the binary, show how new libraries were added, .dyninst and other sections were added and moved.

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?

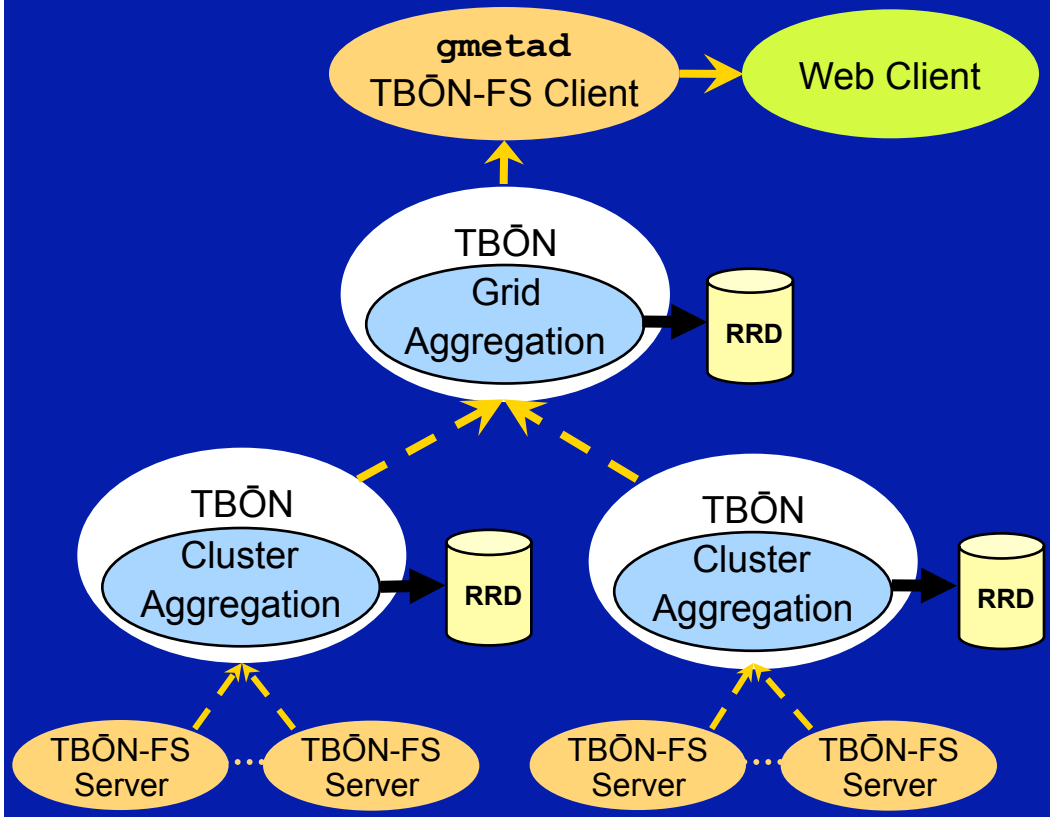
Future Work - Ports

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

Questions?

Madhavi Krishnan
Matthew LeGendre
Bill Williams

Ganglia-tbonfs



@ 500 hosts

- metrics collected at Ganglia default rates

gmetad

50% less CPU use due to TBON aggregation

gmond (tbonfs-server)

steady CPU use (.35%) vs. original `gmond` that increases linearly in cluster size