Improving the Scalability of the TotalView Debugger using TBONs

Michael J. Brim
Paradyn Project, University of Wisconsin

John DelSignore
Rogue Wave Software

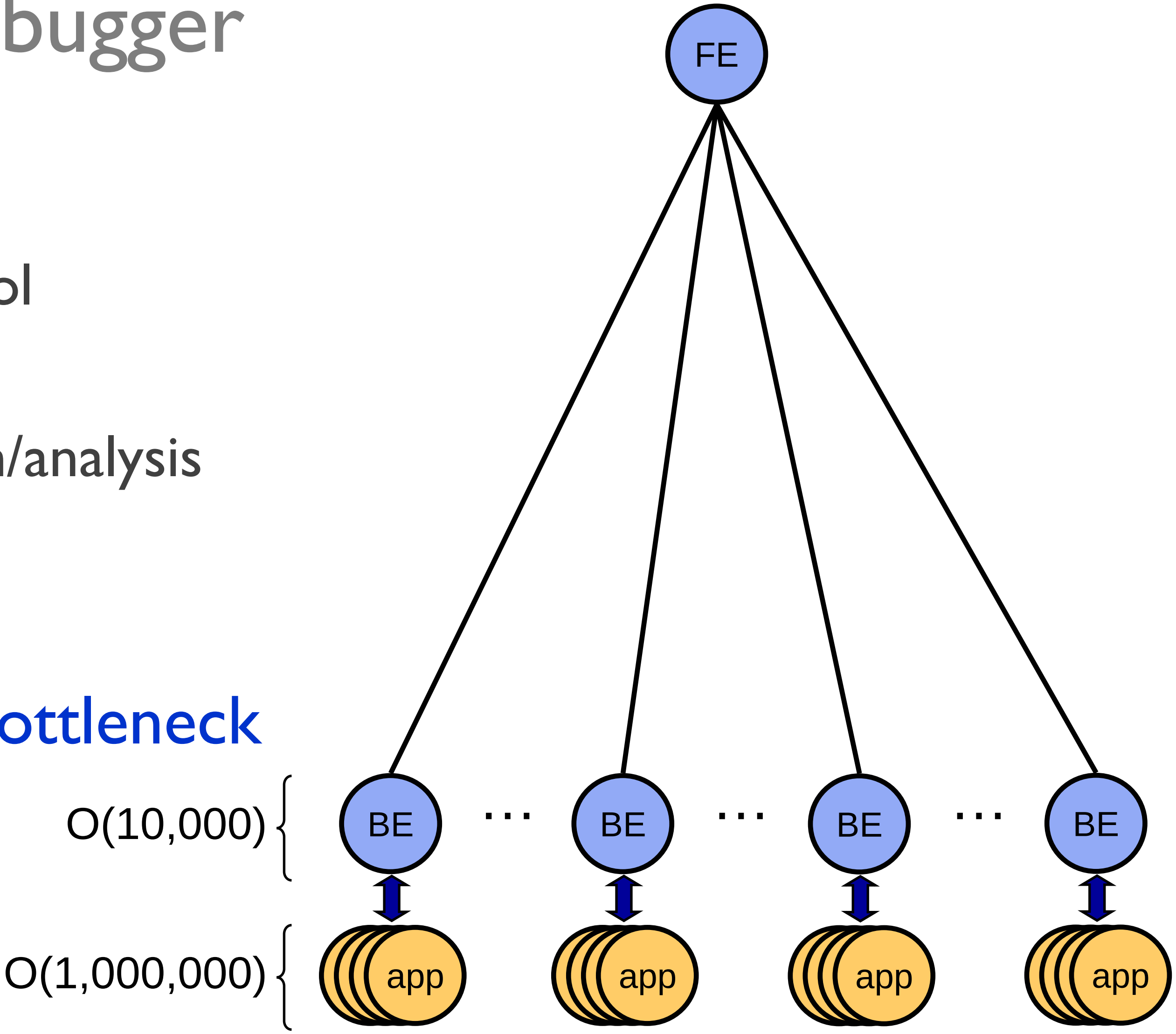
CScADS
August 1, 2011

The ~~Tool~~ Scalability Problem
TotalView Debugger

Key tasks:

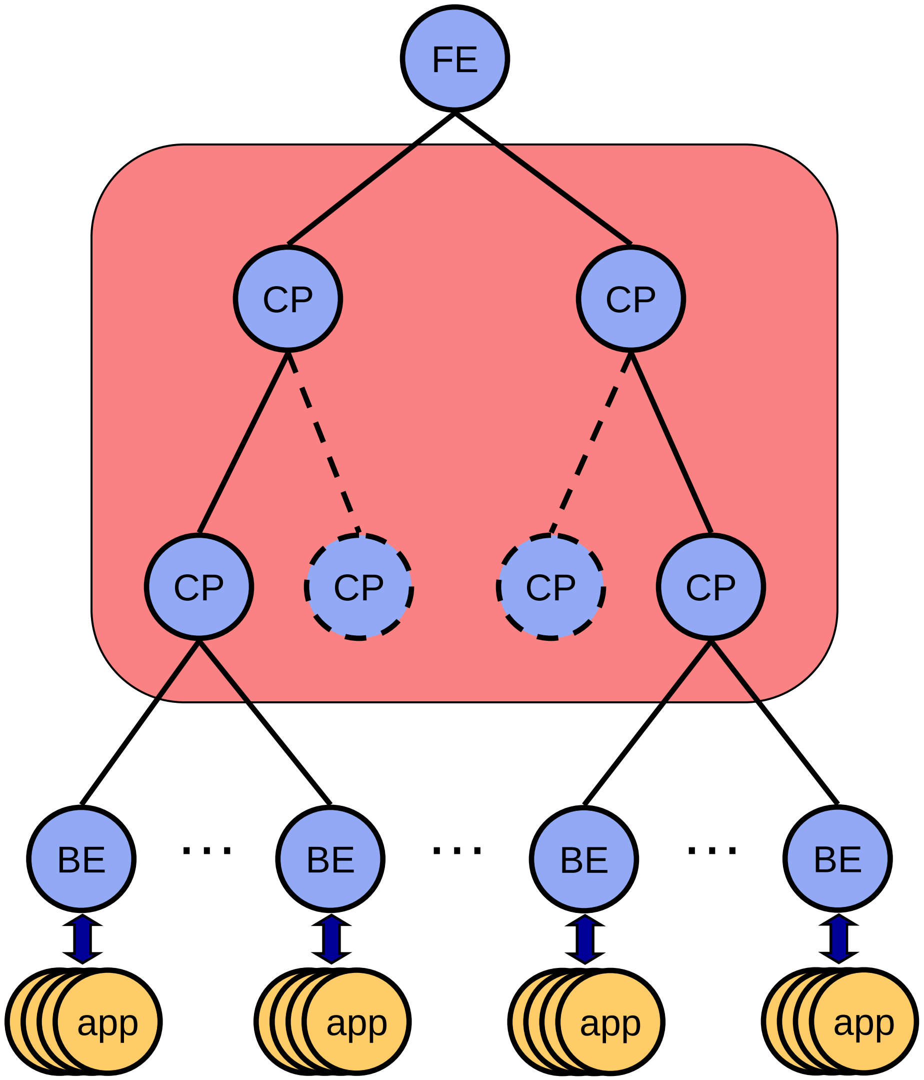
- Application Control
- Data collection
- Data centralization/analysis

As scale increases,
front-end becomes bottleneck



Tree-Based Overlay Networks (TBONs)

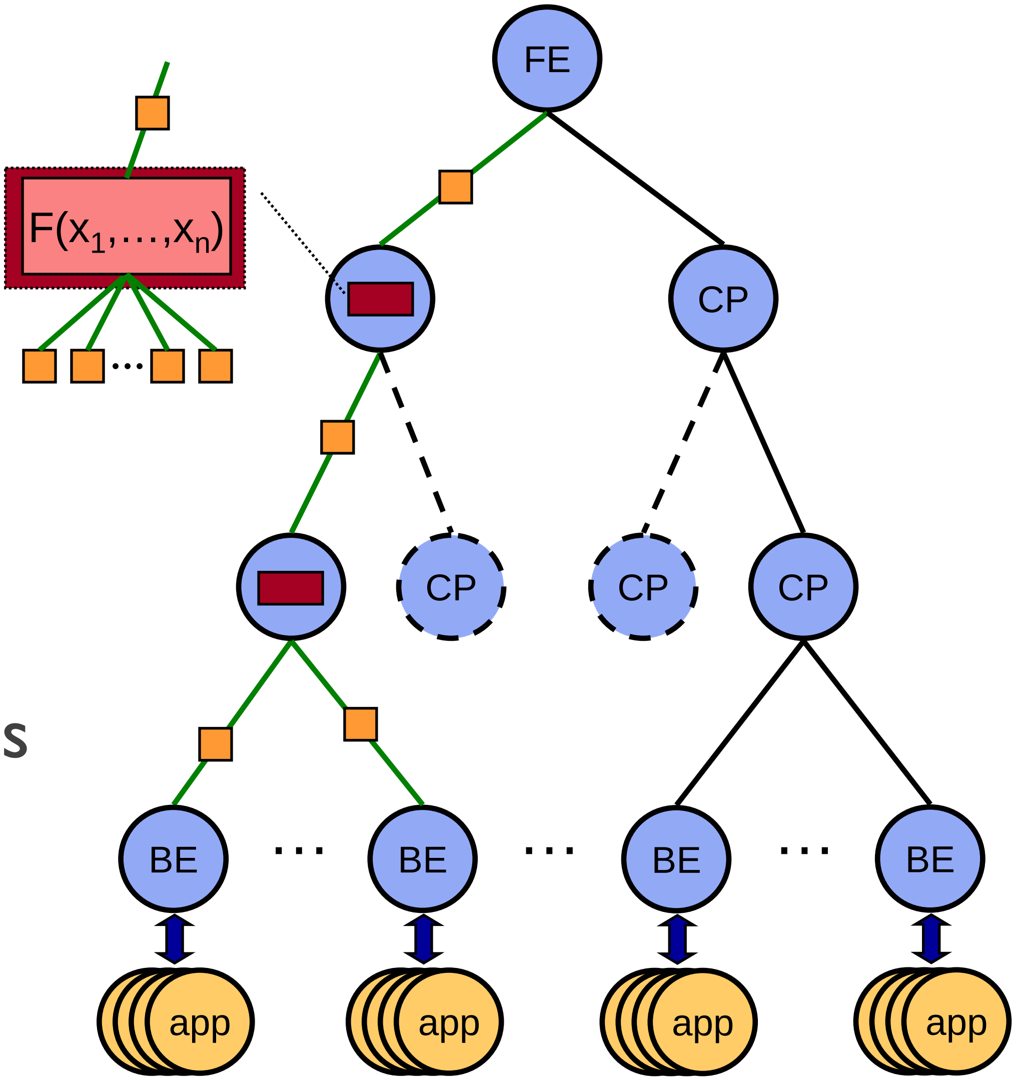
- Scalable multicast
- Scalable gather
- Scalable data aggregation
- Natural redundancy



MRNet – Multicast / Reduction Network

General-purpose TBON API

- Network: user-defined topology
- Stream: logical data channel
  - to a set of back-ends
  - multicast, gather, and custom reduction
- Packet: collection of data
- Filter: stream data operator
  - synchronization
  - transformation



Widely adopted by HPC tools

- CEPBA toolkit
- Cray ATP & CCDB
- Open|SpeedShop & CBTF
- STAT
- TAU
- …

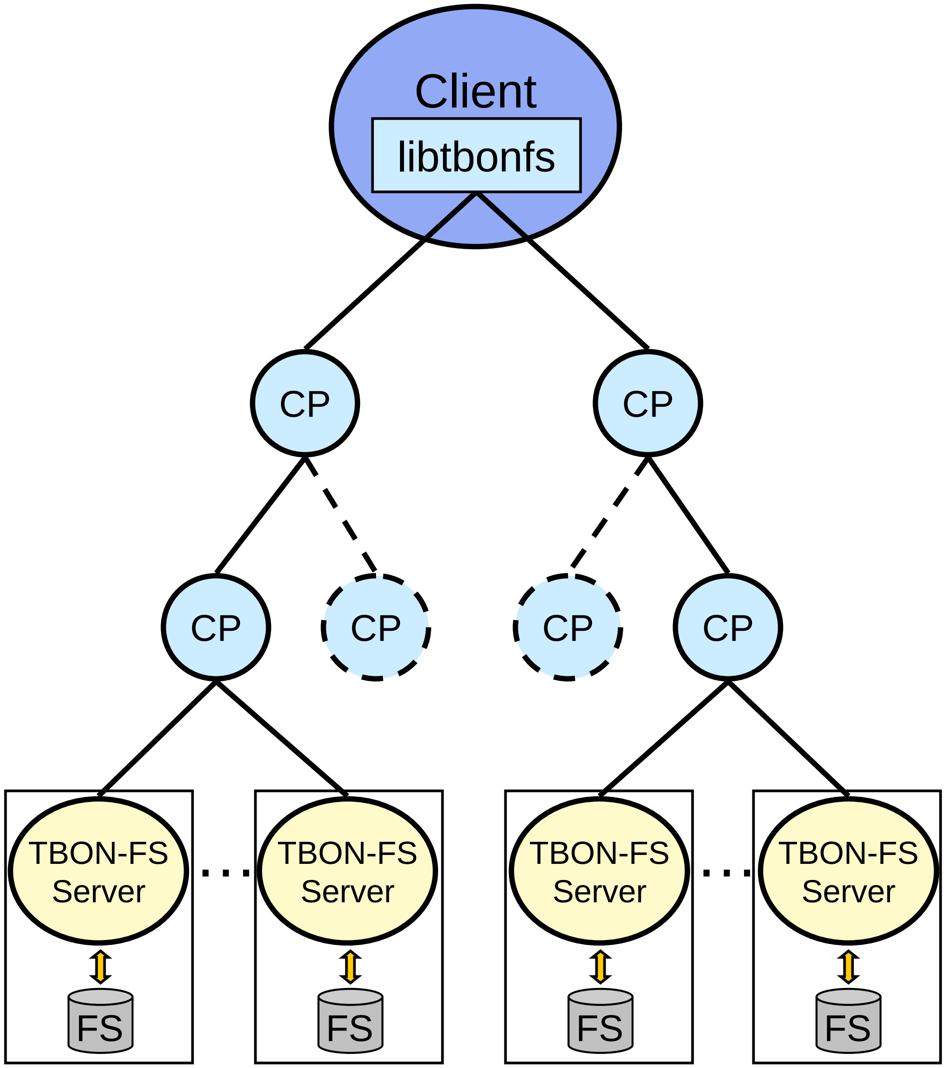
TBON-FS : the TBON File System

Specialized TBON for distributed file access

- back-end data sinks/sources are files
- simplifies tool front-end development by providing an intuitive interface based on POSIX I/O
- custom tool back-end functionality via synthetic file systems loaded into TBON-FS servers

Uses MRNet for:

- scalable unified name space composition
- scalable group file operations



Group File Operations

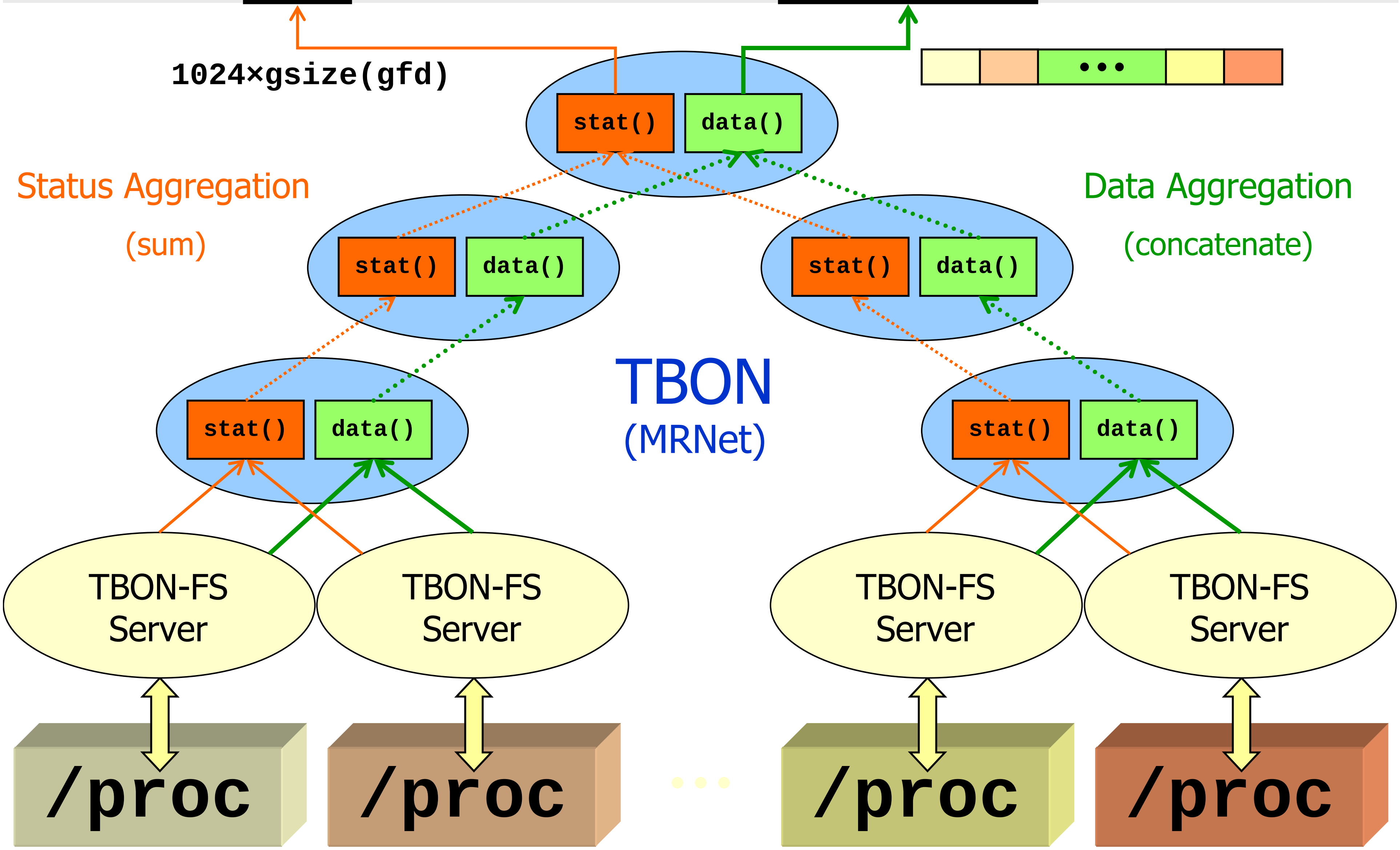
```
gfd = gopen(dir, flags, mode)
```

Operating on Groups

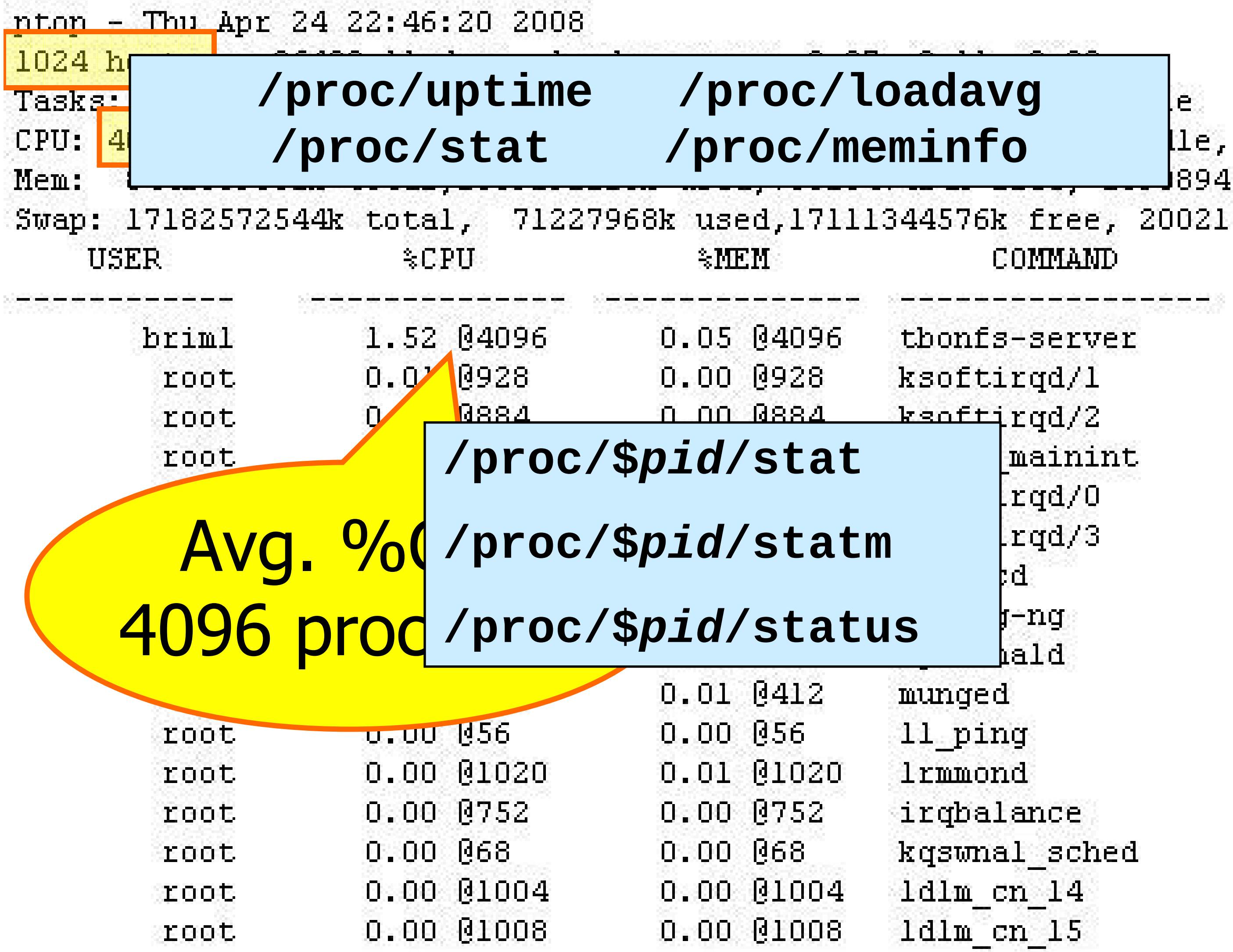
- Use group file descriptor with regular file operations (e.g., `read` and `write`)
  - avoids iteration, one system call per group operation
- Semantics
  - operation applied to each group member
  - user-controlled aggregation of status and data results

TBON-FS: Scalable Group File Operations

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



Scalable Distributed Process Monitoring: `ptop`



```
ptop - Thu Apr 24 22:46:20 2008
1024 h
Tasks:                                                    e
CPU: 4                                                    lle,
Mem:                                                      89408
Swap: 17182572544k total,  71227968k used,17111344576k free, 200214
    USER            %CPU              %MEM              COMMAND
------------    --------------    --------------    -----------------
       briml        1.52 @4096        0.05 @4096    tbonfs-server
        root        0.0  @928         0.00 @928     ksoftirqd/1
        root        0    @884         0.00 @884     ksoftirqd/2
        root                                        _mainint
                                                    rqd/0
                                                    rqd/3
                                                    d
                                                    -ng
                                                    ald
                                      0.01 @412     munged
        root             @56          0.00 @56      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
```

/proc/uptime /proc/loadavg
/proc/stat /proc/meminfo

Avg. %C
4096 proc

/proc/*$pid*/stat

/proc/*$pid*/statm

/proc/*$pid*/status

4,096
files

>1,000,000
files

Group Process Control & Inspection

`/proc` : a good starting point

- write to process/thread control file(s) to run/stop/signal
- read files containing process/thread status
- read/write process address space
- read/write thread registers

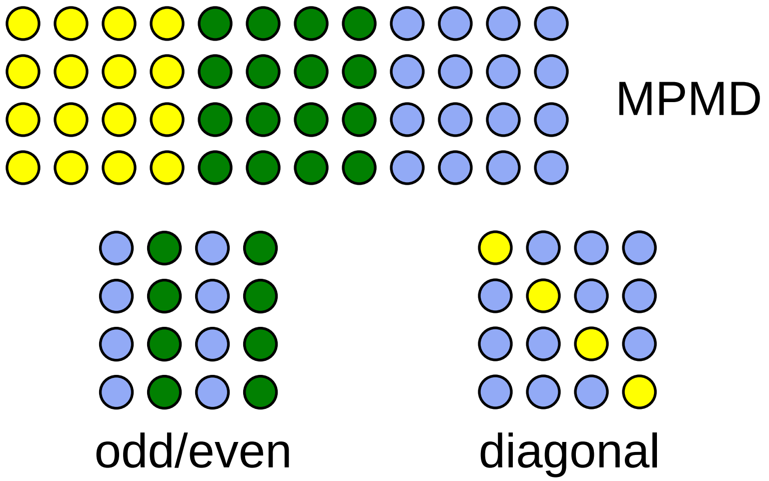
But,

- functionality differs by OS (e.g., no control on Linux)
- no notion of group operations
- always contains all host processes

proc++ : Synthetic File System for Process Control

Improvements over /proc

1. process/thread groups
   - explicit group management
   - directories containing members' control and inspection files automatically created
2. high-level debugger operations
   - breakpoints
   - stepping
   - stack walks
3. platform-independent interface



/proc	proc++

`foreach(member){`
`  restore_insn()`
`  step_target()`
`  insert_bkpt()`
`  run_target()`
`}`
	`run_group()`

Example: Continue group from breakpoint

proc++ : from the makers of Dyninst

Most capabilities provided by ProcControlAPI

- Cross-platform component library / C++ API
  - Linux, FreeBSD, BlueGene, Windows
- Process / thread control and inspection
  - Stop / continue processes, single-step threads
  - Read / write process memory, thread registers
  - Insert / remove breakpoints
  - Inferior remote procedure calls
  - Callbacks for asynchronous event notification

Thread stack walks (StackwalkerAPI)

TotalView Parallel Debugger

Commercial debugger from Rogue Wave Software

- Sequential, multi-threaded, and parallel programs
- Fortran, C, C++ code from various compilers
- pthreads, OpenMP, MPI, UPC

20+ years of engineering and HPC experience

- Advanced MPI debugging
- Built-in memory debugger
- Reverse debugging (application DVR)
- Recent support for GPGPU (CUDA) code

TotalView is a great case study

Most widely-used HPC debugger

- Lots of happy users

Known scalability limitations

- Lots of users that need it to work at full-scale on largest systems (i.e., @ 200K+ processes)

20+ years of engineering

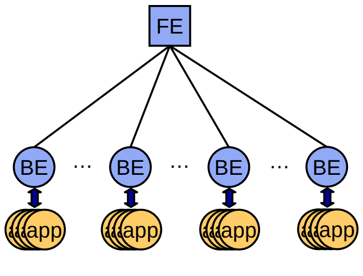
- A real tool that works on real applications
- Modular architecture that evolved over time
- Operations on process and thread groups are primary focus

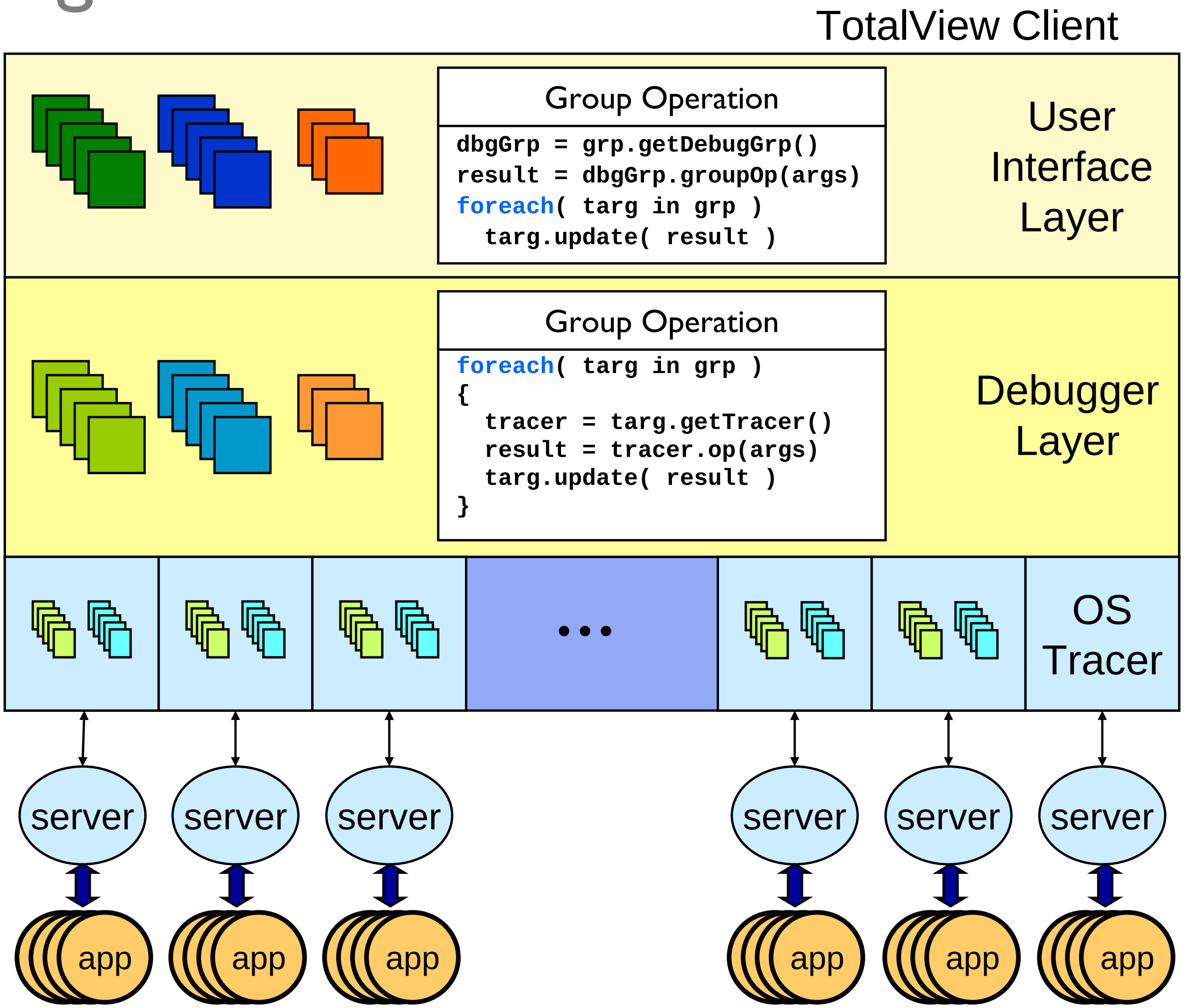
TotalView: Original Architecture

🟩 Process Object

🟦 Thread Object

🟧 Group Object





Group Operation

```
dbgGrp = grp.getDebugGrp()
result = dbgGrp.groupOp(args)
foreach( targ in grp )
  targ.update( result )
```

Group Operation

```
foreach( targ in grp )
{
  tracer = targ.getTracer()
  result = tracer.op(args)
  targ.update( result )
}
```

TotalView Integration Challenges

Group Operations

- no group operations at (lowest) tracer level
  - pushed groups down to use group file operations
- some group operations at UI level use iteration
  - added group operations at debugger level
- some group operations require process- or thread-specific context
  - extended proc++ interface and capabilities

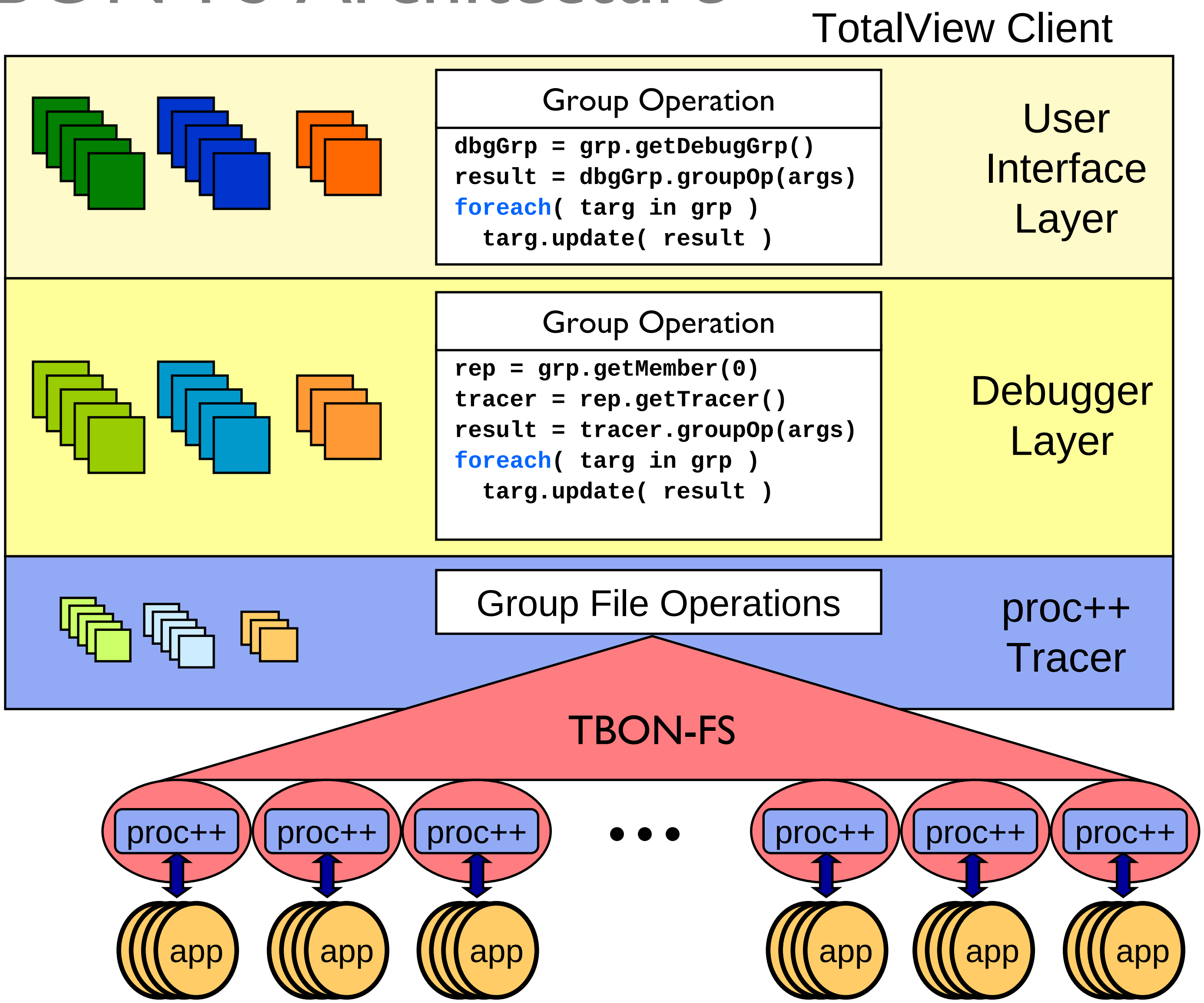
Multi-level object maintenance

TotalView: TBON-FS Architecture

🟩 Process Object

🟦 Thread Object

🟧 Group Object



```
dbgGrp = grp.getDebugGrp()
result = dbgGrp.groupOp(args)
foreach( targ in grp )
  targ.update( result )
```

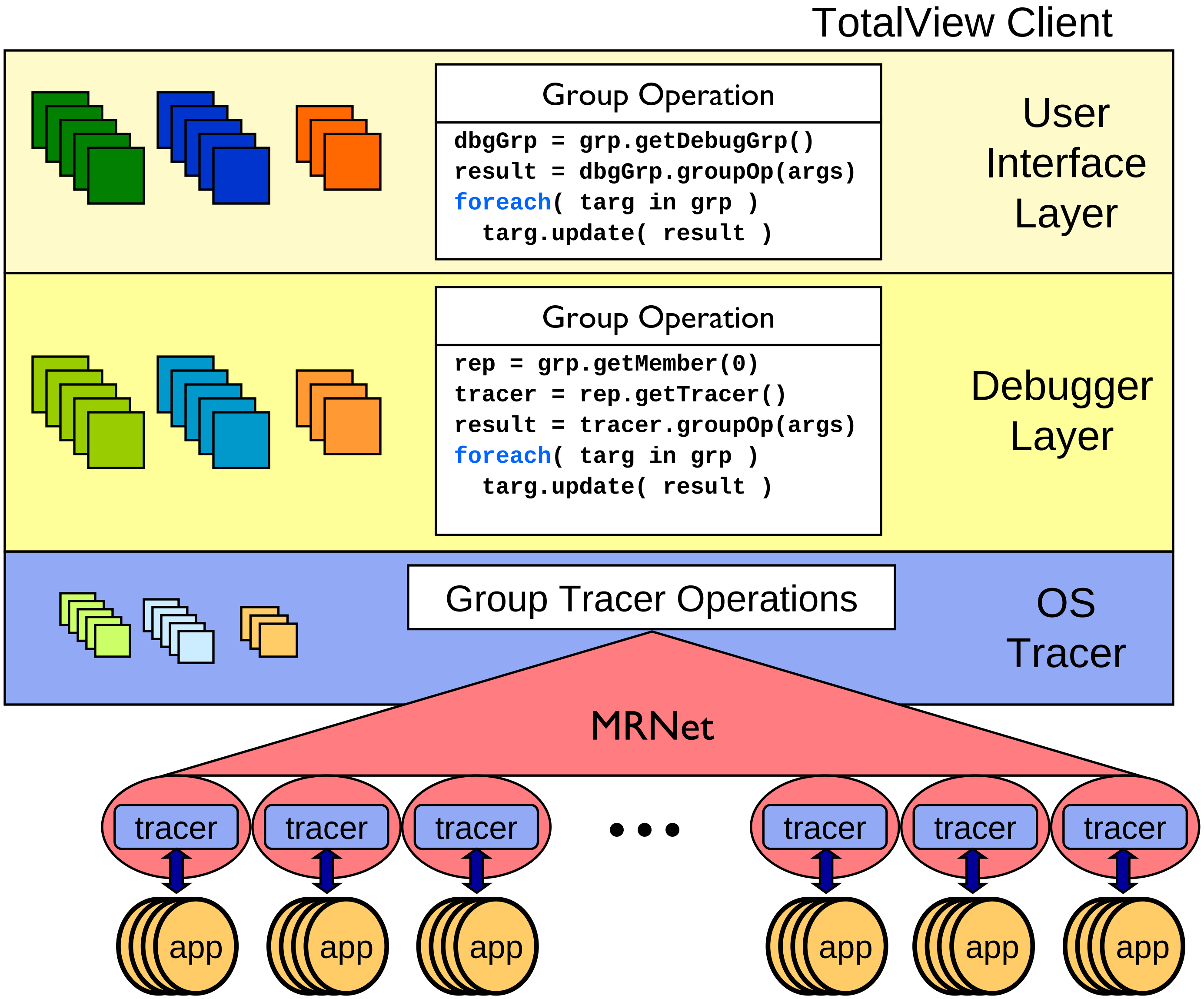
```
rep = grp.getMember(0)
tracer = rep.getTracer()
result = tracer.groupOp(args)
foreach( targ in grp )
  targ.update( result )
```

TotalView: MRNet Architecture

🟩 Process Object

🟦 Thread Object

🟧 Group Object



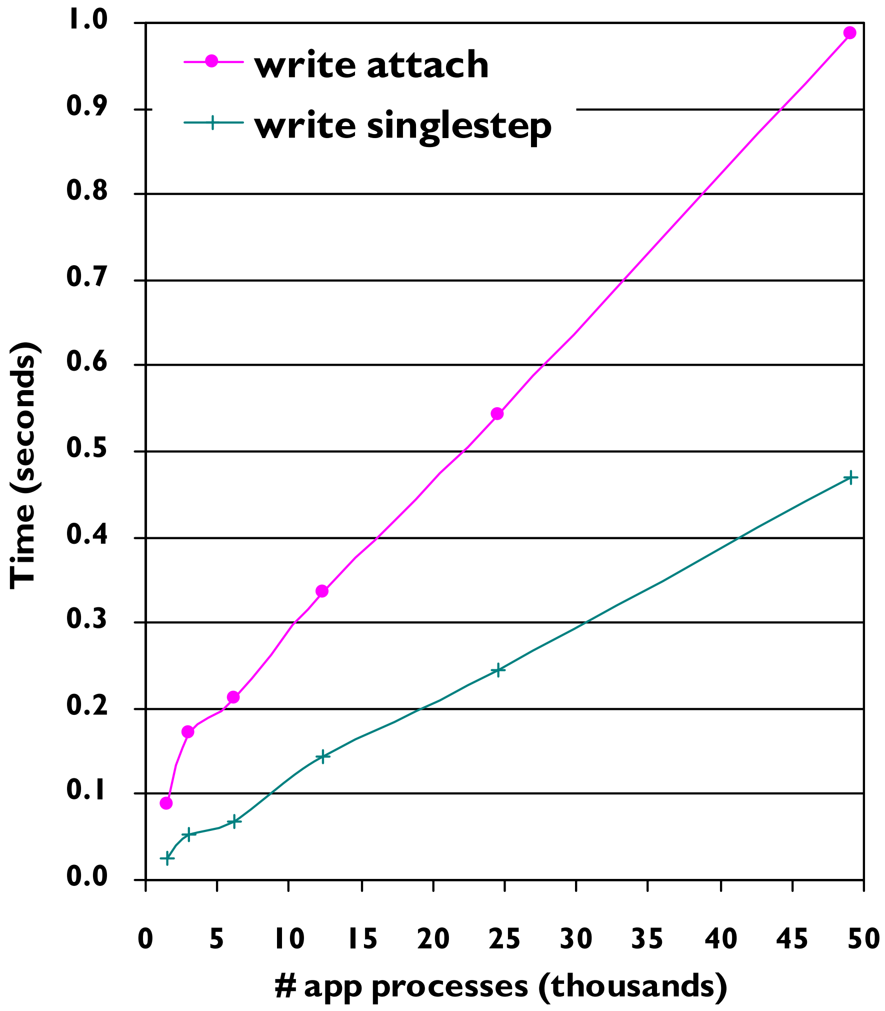
User Interface Layer, Group Operation:

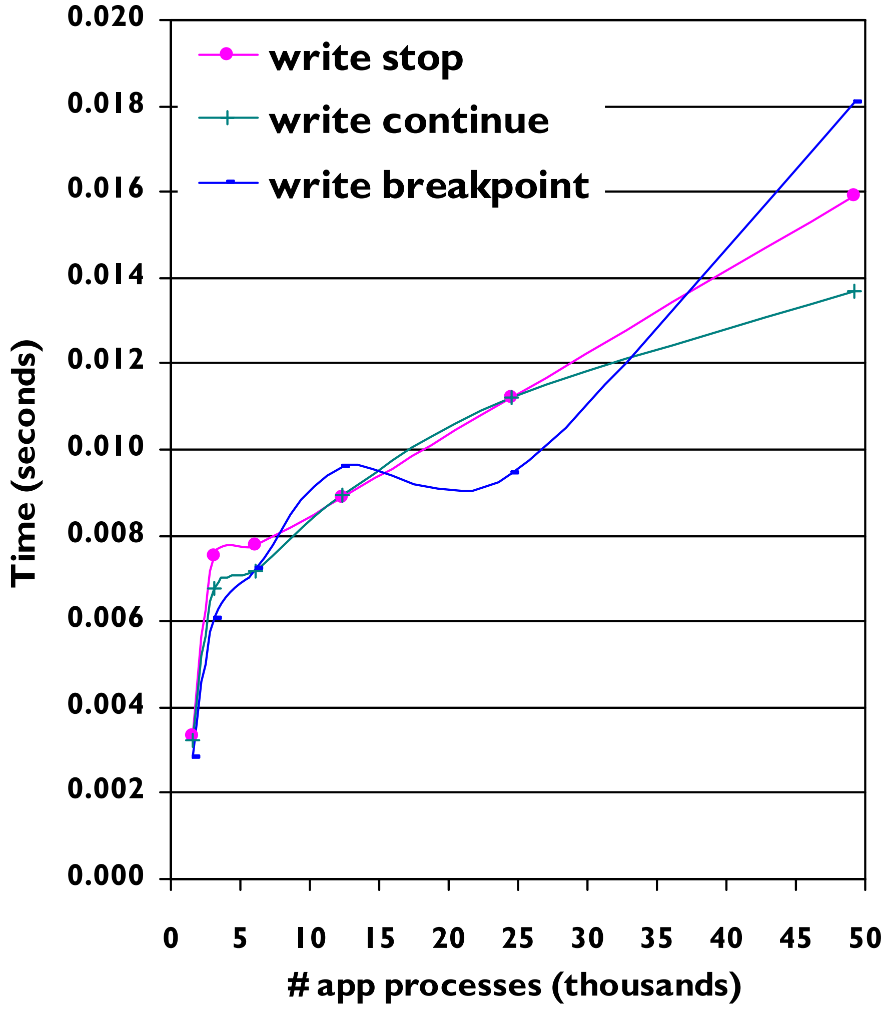
```
dbgGrp = grp.getDebugGrp()
result = dbgGrp.groupOp(args)
foreach( targ in grp )
  targ.update( result )
```

Debugger Layer, Group Operation:

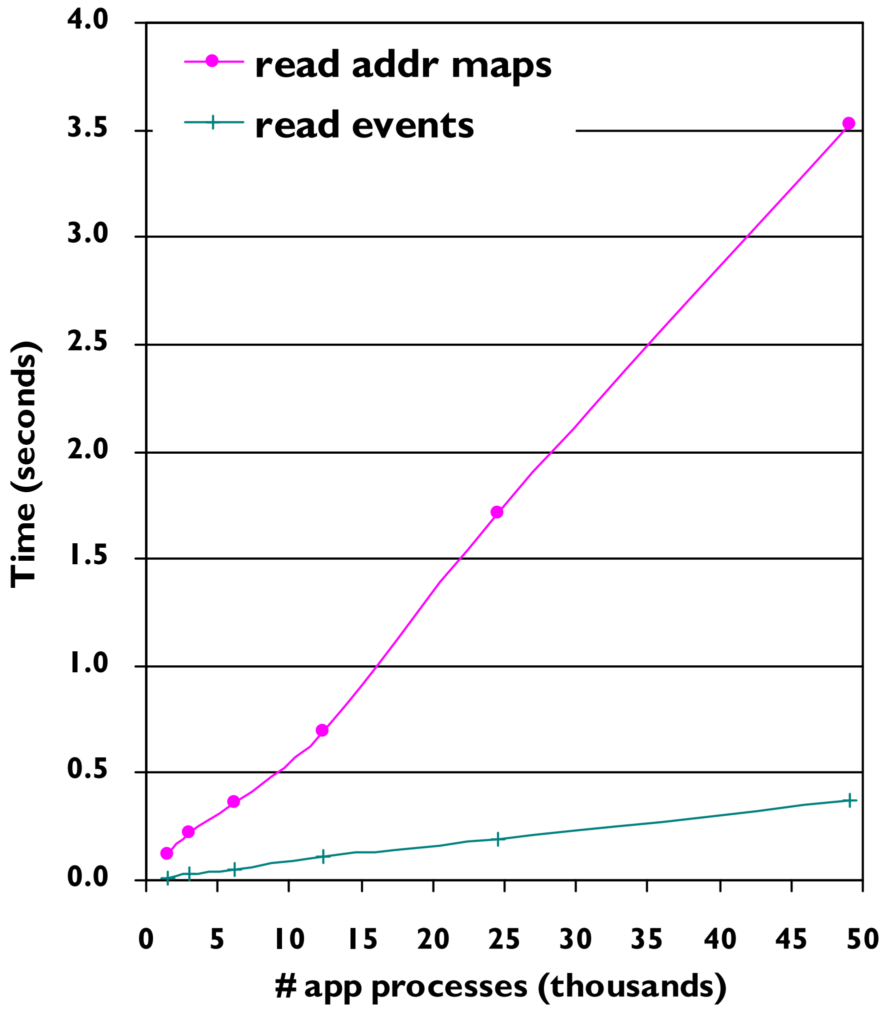
```
rep = grp.getMember(0)
tracer = rep.getTracer()
result = tracer.groupOp(args)
foreach( targ in grp )
  targ.update( result )
```

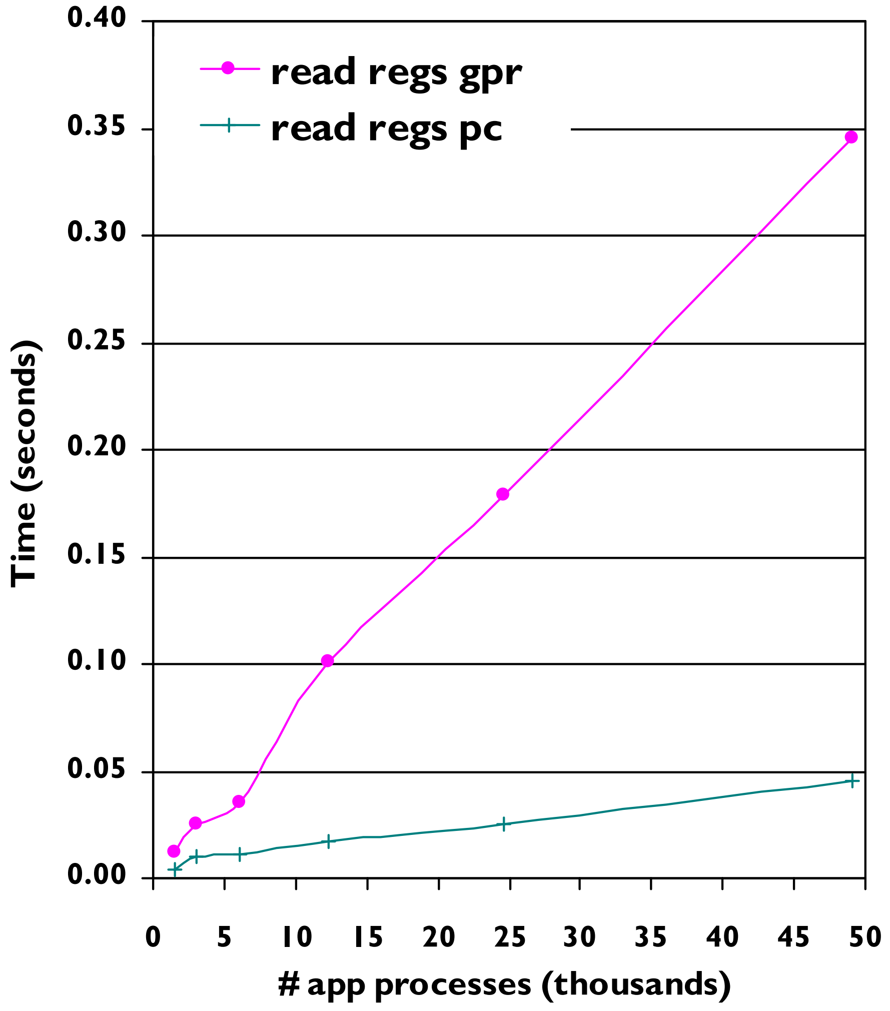
Scalability: proc++ group writes





Scalability: proc++ group reads

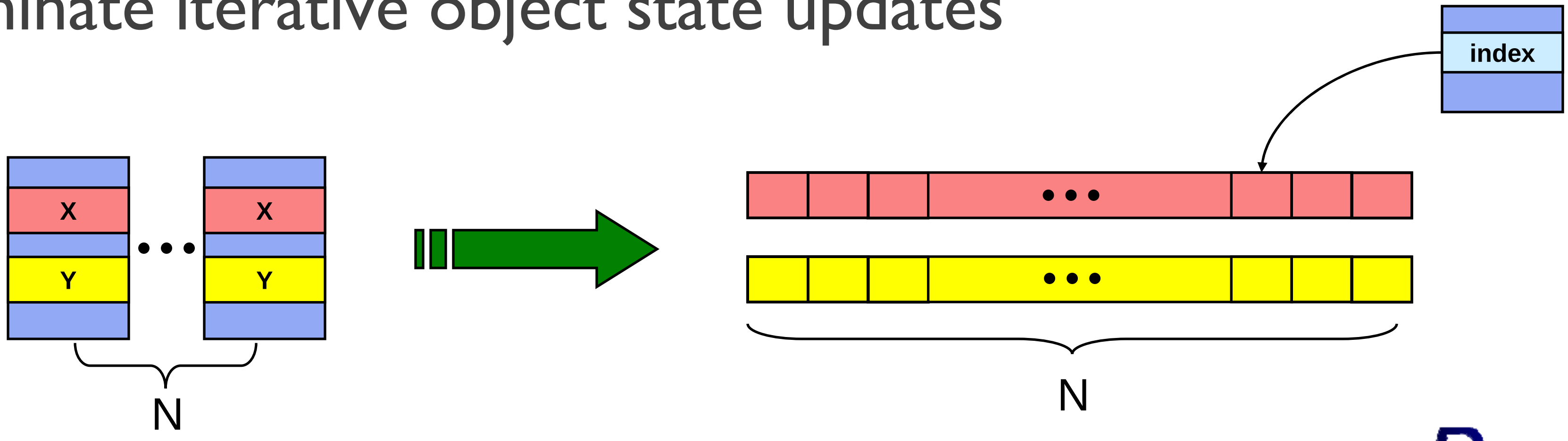




Amdahl's Law for Scalable Tools

Speed-up from using scalable group file operations is limited by front-end sequential behaviors

- reduce the number of objects per target
- reduce the state kept in those objects
- eliminate iterative allocation of objects
- eliminate iterative object state updates



Keys to Real Tool Scalability

"iteration is the bane of scalability"- me

- any operation requiring a linear number of steps is a show-stopper

1. Limited sequential behavior in tool front-end
2. Good group representation
   - efficient creation and update ⇒ distributed group state
3. Constant or logarithmic time group operations
   - parallel execution across group members
4. Constant or logarithmic size data at tool front-end
   - tool internal state: O(# of groups), not O(# of targets)
   - user display of group data: scalable aggregation is necessary

Tool Scalability "rules to live by"

1. Single-target operations must be efficient, but rarely used

2. On-demand data access (lazy evaluation)
   - do not collect or generate data that is never used

3. Data Caching
   - individual target data at tool front-end is a bad idea
     - leads to iterative cache invalidation and update
     - see rule #2
   - individual target data at tool back-ends is a time/space tradeoff
   - group data at tool front-end is a time/space tradeoff
     - caching within a TBON can limit both time and space

Questions?

Group File Operations & TBON-FS

- International Conference on High Performance Computing (HiPC 2009) Best Paper
- ftp://ftp.cs.wisc.edu/paradyn/papers/Brim09GroupFile.pdf

Scalable Composition of File System Name Spaces

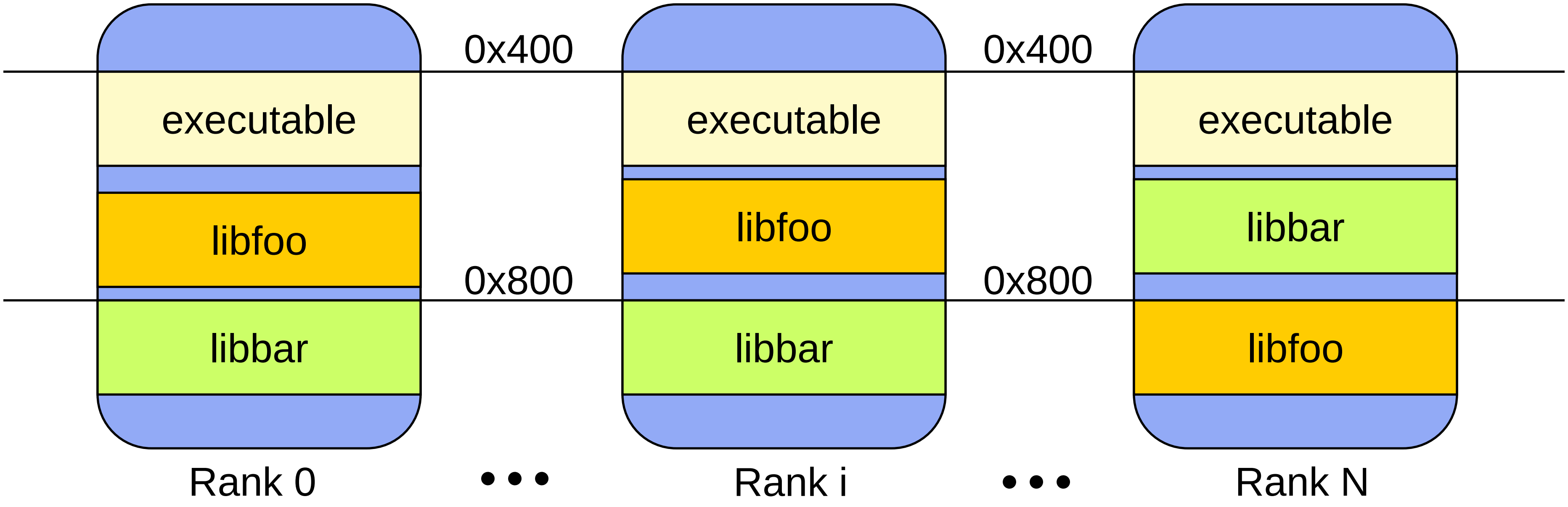
- International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2011)
- ftp://ftp.cs.wisc.edu/paradyn/papers/Brim11FinalNamespace.pdf

MRNet : http://www.paradyn.org/mrnet/

TBON-FS or proc++ Source Code (talk to me)

TotalView Integration: `proc++` Extensions

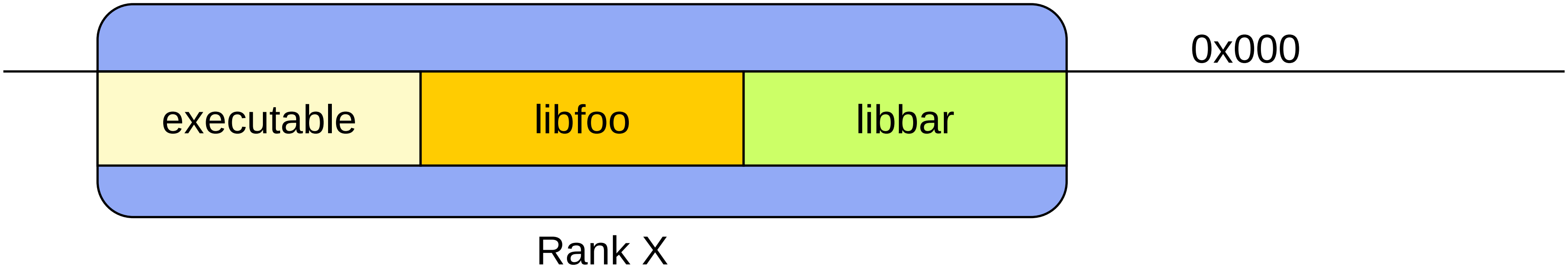
Problem: dynamic address space mappings



How can we do group address space write/read?

TotalView Integration: **proc++** Extensions

Solution: image files that hide dynamic mappings



- one file for each mapped code image
- zero offset corresponds to map base of image
- to read / write symbols in image, seek to the symbol offset