

***Building a Community Infrastructure for Scalable
On-Line Performance Analysis Tools***

also known as

Component Based Tool Framework

“CBTF”

Jim Galarowicz, Don Maghrak, Bill Hachfeld

The Krell Institute

- ***Project Goals***
- ***Project Team***
- ***Rationale for Project***
- ***Project Goals/Objectives***
- ***Last Years Status vs Current Status (Progress)***
- ***What is next?***

- 1) Krell gives overview of current state of project infrastructure
- 2) Discussion topics:
 - a) Tool start-up mechanisms
 - i) Relating to LW MRNet BE connection - it seems there isn't a start-up mechanism for the LW MRNet BE model
 - b) Binary rewriter integration into CBTF - as a component?
 - c) How to integrate Active Harmony?
- 3) Longer term issues
 - a) Publish a paper about CBTF (who, when, etc.)
 - b) What are the end goals -- what to show at the end of the 3 years (Oct 2012)
 - i) Report form
 - ii) Demonstrations
 - iii) Looking for feedback from those who have experience at this.
- 4) License issues - BSD type vs GPL/LGPL discussion

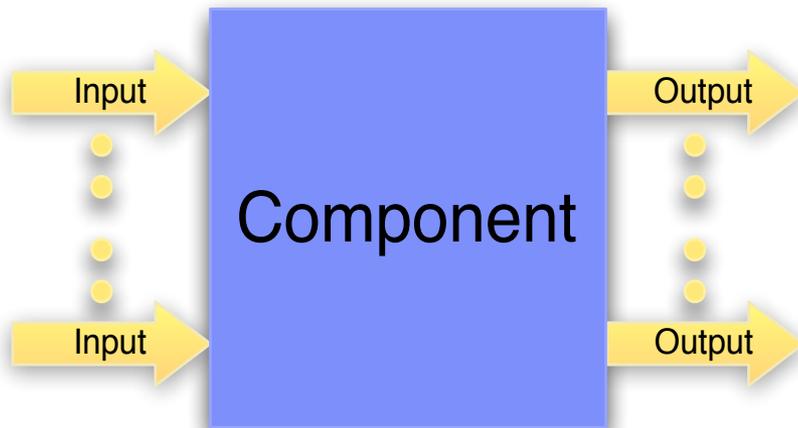
- **Project Goals/Objectives:**

- *Create a set of reusable components for building high-level end user tools and/or quickly build tool prototypes.*
- *Tools should be easily configurable without rebuilding core infrastructure.*
- *Able to mix components from several groups and/or vendors. Everyone would be able to contribute and use the new components.*
- *Recreate Open | SpeedShop from CBTF provided components and services.*

- **Project Team**

- *The Krell Institute*
- *University of Maryland*
- *University of Wisconsin*
- *Oak Ridge National Laboratory*
- *Lawrence Livermore National Laboratory*
- *Los Alamos National Laboratory*
- *Sandia National Laboratories*
- *Carnegie Mellon University*
- *Others welcome.....*

- **Why the need for the project?**
 - *Petascale environments need tool sets that are flexible*
 - *Need to quickly create new and specialized tools*
 - *Better availability of tools across more platforms*
 - *Helps in building modular tools and in the avoidance of creating stove pipe tools*



❖ Data-Flow Model

- Accepts Inputs
- Performs Processing
- Emits Outputs

❖ C++ Based

❖ Provide Metadata

- Type & Version
- Input Names & Types
- Output Names & Types

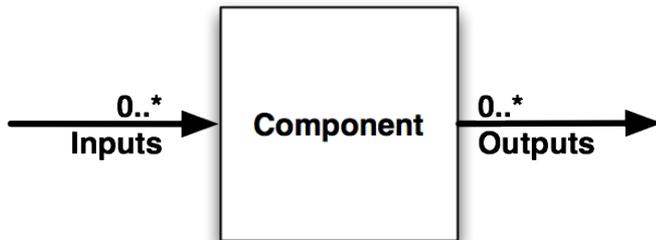
❖ Versioned

- Concurrent Versions

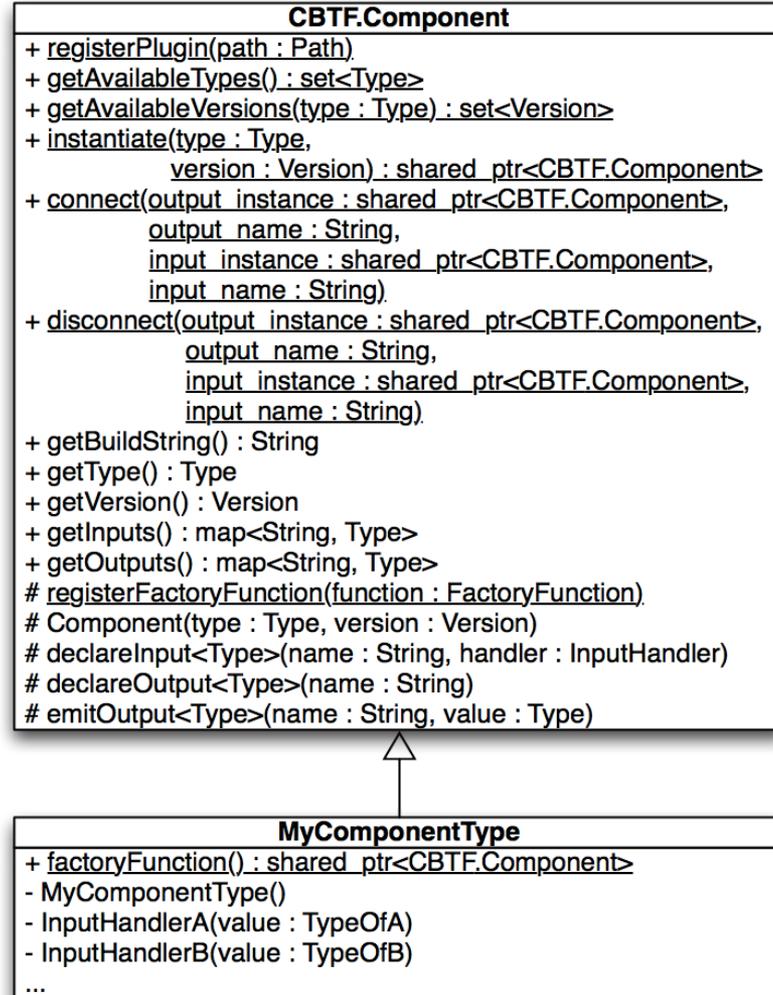
❖ Packaging

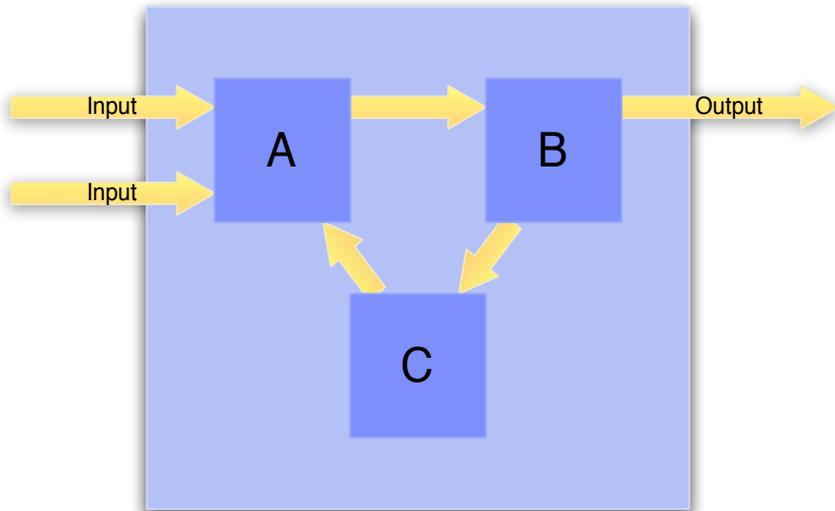
- Executable-Embedded
- Shared Library
- Runtime Plugin

Abstract



Concrete





- ❖ **Components**
 - Specific Versions
- ❖ **Connections**
 - Matching Types
- ❖ **Arbitrary Topology**
- ❖ **Recursive**
- ❖ **XML-Specified**

must tell cbtf about plugins avail.

registerPlugin(A)

registerPlugin(B)

create the instantiation of the plugin

instance_of_a1=instantiate(Type(A))

instance_of_a2=instantiate(Type(A))

...

instance_of_b2=instantiate(Type(B))

now connect the components

**connect(instance_of_a1, "out",
 instance_of_a2, "in")**

**connect(instance_of_a2, "out",
 instance_of_a3, "in")**

...

**connect(instance_of_b1, "out",
 instance_of_b3, "in");**

....

```
<Type>ExampleNetwork</Type>  
<Version>1.2.3</Version>
```

```
<SearchPath>./opt/myplugins</SearchPath>  
<Plugin>myplugin</Plugin>
```

```
<Component>
```

```
  <Name>A1</Name>
```

```
  <Type>TestComponentA</Type>
```

```
</Component>
```

...

...

```
<Connection>
```

```
  <From>
```

```
    <Name>A1</Name>
```

```
    <Output>out</Output>
```

```
  </From>
```

```
  <To>
```

```
    <Name>A2</Name>
```

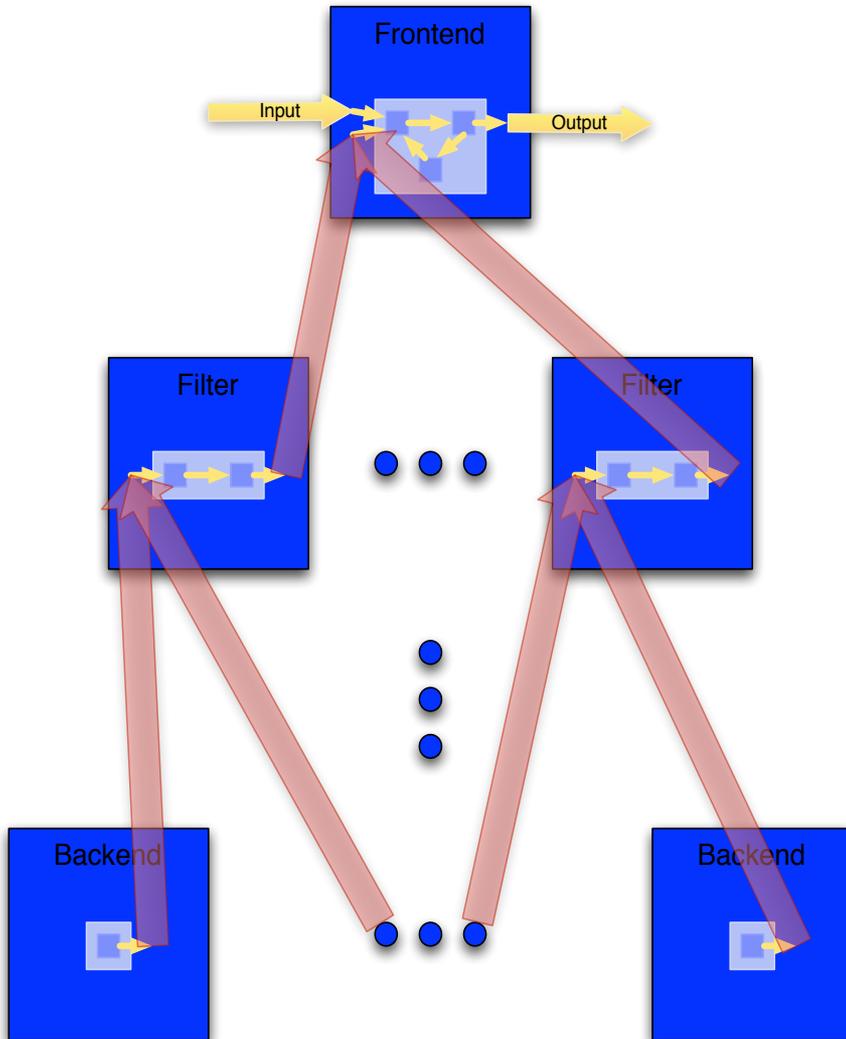
```
    <Input>in</Input>
```

```
  </To>
```

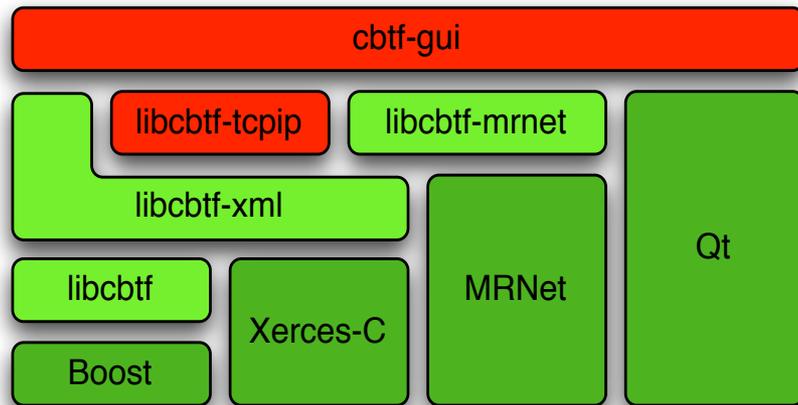
```
</Connection>
```

...

```
</Network>
```



- ❖ **MRNet Based**
- ❖ **Per-Node Networks**
 - Homogenous Within Tree Levels
 - Heterogeneous Between Tree Levels
- ❖ **Named Streams**
 - Up and Down
 - Connect Networks
- ❖ **Also Recursive**
- ❖ **Also XML-Specified**
- ❖ **Supports LW MRNet**



- ❖ **Minimal Dependencies**
 - Easier Builds
- ❖ **Tool-Type Independent**
 - Performance Tools
 - Debugging Tools
 - etc...
- ❖ **Completed Components**
 - Base Library
 - XML-Based Component Networks
 - MRNet Distributed Components
- ❖ **Planned Components**
 - TCP/IP Distributed Component Networks
 - GUI Definition of Component Networks

Collector  service plugin

Component  service plugin

❖ Services

- Libraries (C or C++) of functionality that don't fit into the data-flow model of the CBTF component framework.
- Collection services
 - Unwinding service
 - Timer service
 - HWC (PAPI) service
 -

❖ Messages

- Defines the data that is exchanged between data-flow components.

❖ Core

- C++ Base Classes
 - Time, Address, Blob, Path support

❖ **Open | SpeedShop**

- Using Services, Messages, Core, and CBTF
- Full fledged multipurpose performance tool

❖ **Customized Tools**

- Built from Services, Messages, Core, and CBTF
- Aimed at specific tool needs determined by application code teams

❖ **Last Years (April 2010) Status**

- *Held a number of Open | SpeedShop team design meetings*
- *Started holding extended CBTF team meetings to discuss ideas for component interfaces*
- *Created a CBTF wiki*
- *Started prototyping the component interface design*
- *Doing a number of improvements and decompositions in Open | SpeedShop for deployment in CBTF*

❖ **This Years (May 2011) Current Status**

- *CBTF design/status meetings with extended team*
- *Used Open | SpeedShop as a test bed driver for CBTF design*
- *Designed, Implemented and Tested three of the core libraries that define the CBTF infrastructure*
 - *libcbtf*
 - *libcbtf-xml*
 - *libcbtf-mrnet*
- *Designed another core library: libcbtf-tcp*
- *Writing components, services, and messages to use with the CBTF infrastructure with the goal of supporting program counter sampling, then other collection types.*

❖ **Next steps for CBTF:**

- Add support for LW MRNet's backend attach mode
- GUI tool for CBTF component network configuration. The XML files get tedious to write by hand...
- TCP/IP library implementation and test. (libcbtf-tcpip)
- Tool start up investigation and implementation (launchmon, libi, ...)
- Tool component creation to support more types of collection
- Filtering components for MRNet communication node deployment

❖ **MRNet and Dyninst Features/Requirements/Desires**

- *Implementing usage of the MRNet lightweight library*
 - *Prototyped using OSS, now implementing in CBTF*
- *Working with UW on the detach on the fly*
- *Plan to use the binary rewriter feature*
- *Plan to use symtabAPI*
- *Would like a floating point register fix up feature*
- *Would like a 1st party stackwalker API*
- *Plan to create an “new” CBTF tool/feature based on Active Harmony*

Questions?

jeg@krellinst.org

cbt_framework@krellinst.org

Supplemental Information

- **List of the Component Interface Libraries**

- **libcbtf**

- Provides Cores Services:
 - *Component Abstraction, Metadata, Interconnection*
- Minimal Dependencies (GNU Build Tools & Boost)

- **libcbtf-xml**

- Provides XML-Based Definition of Single-Process Component Networks
- Depends on libcbtf and Xerces-C

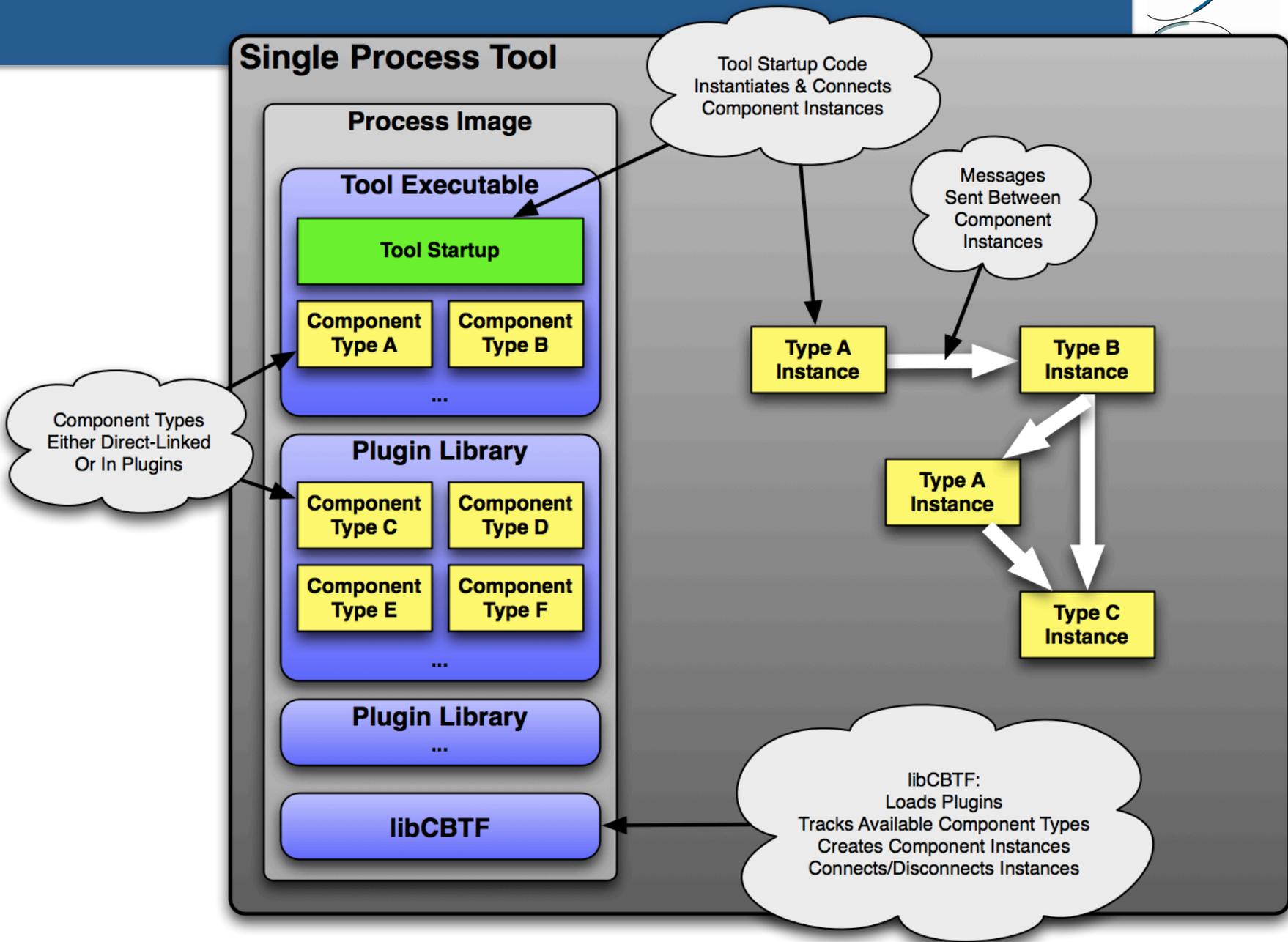
- **libcbtf-mrnet**

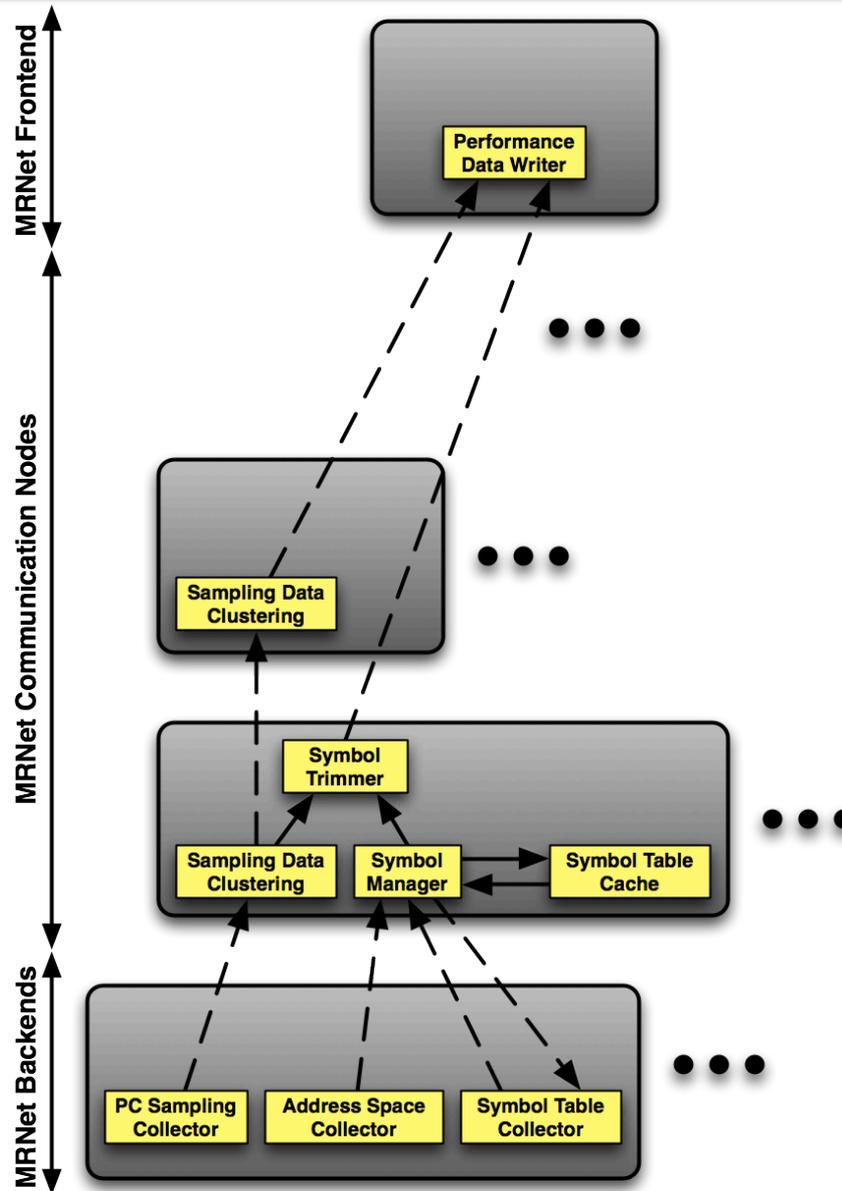
- Provides XML-Based Definition of MRNet-Based Distributed Component Networks
- Depends on libcbtf, libcbtf-xml, and MRNet

- **libcbtf-tcpip**

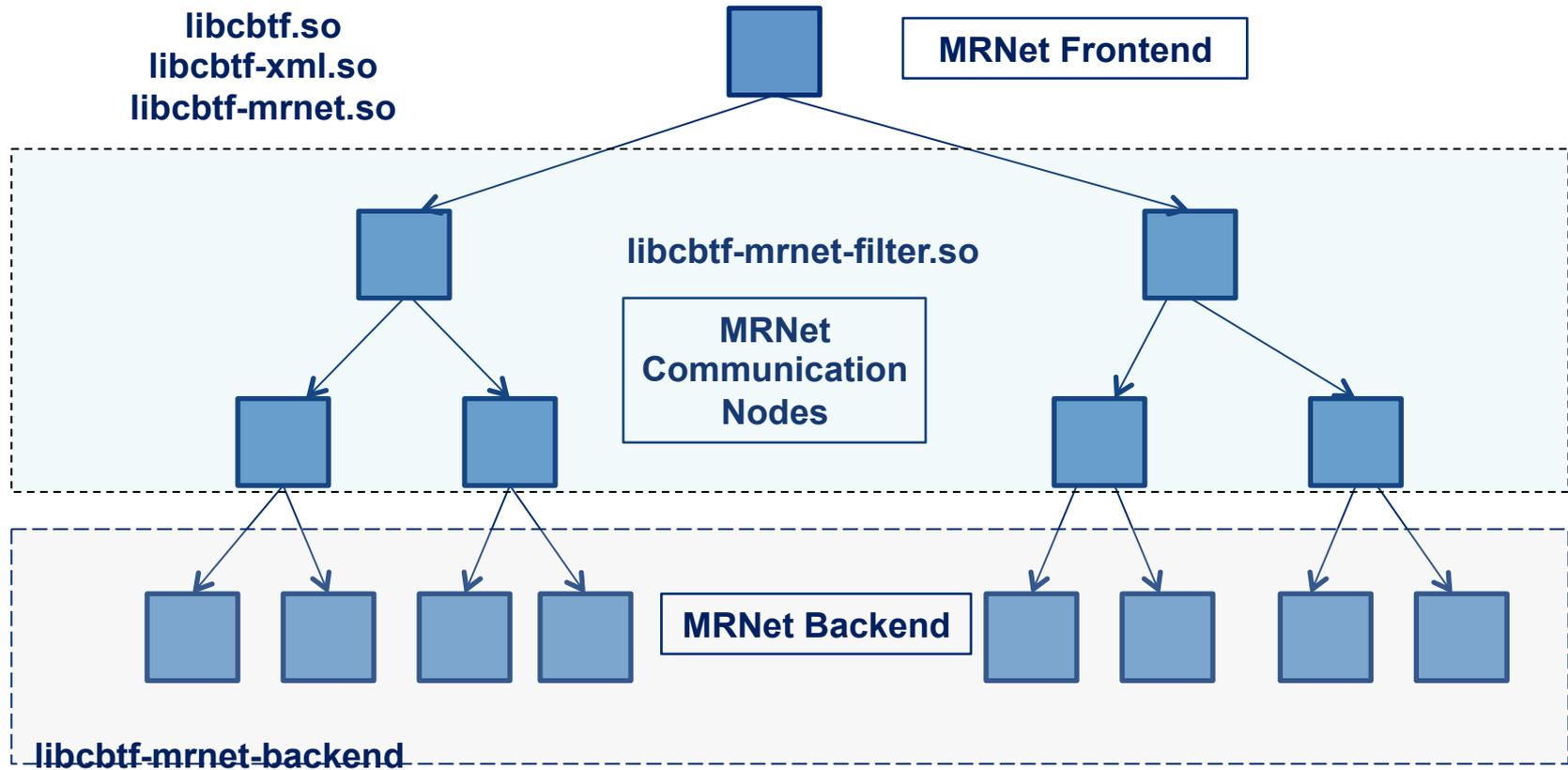
- Provides XML-Based Definition of TCP based networks
- Depends on libcbtf, libcbtf-xml, and TCP/IP

Single Process Tool

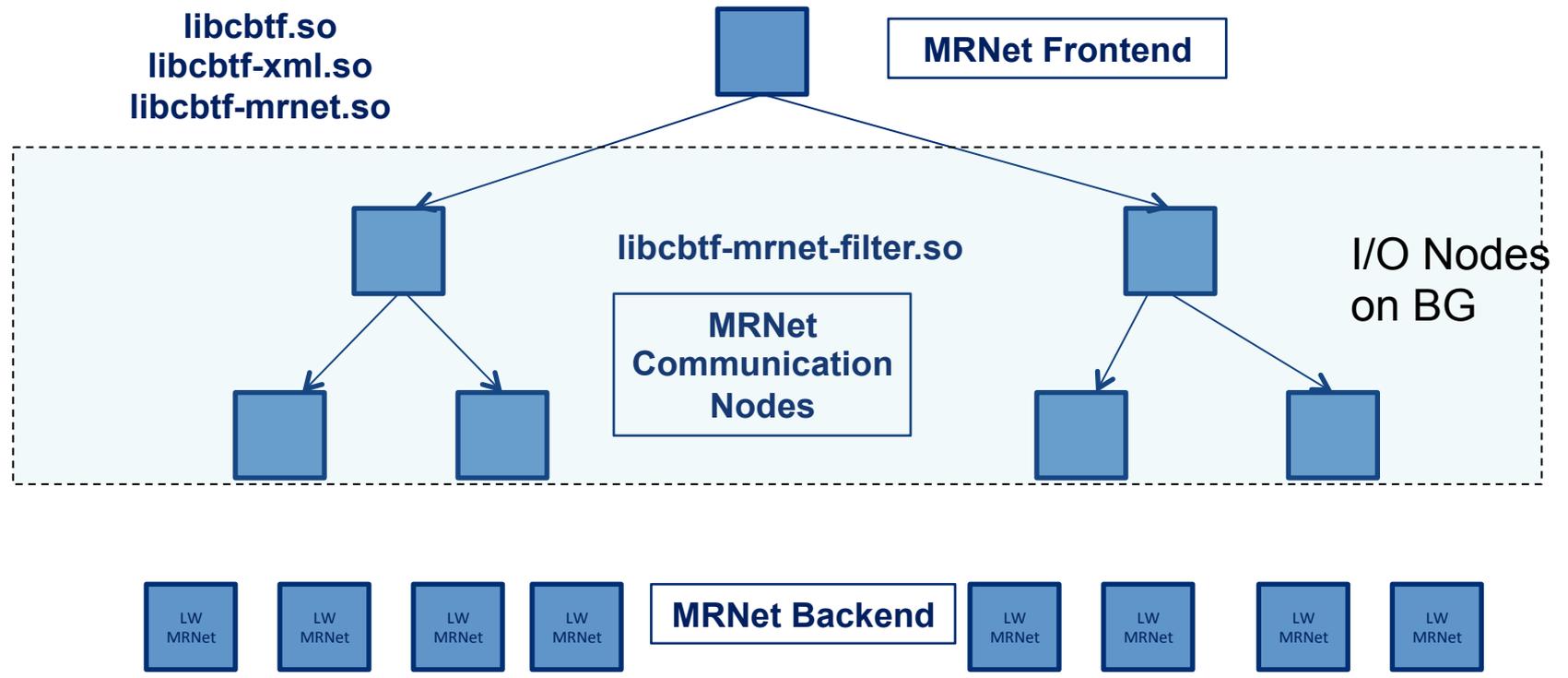




MRNet component network example



How to connect LW MRNet BEs at start up time



Application containing LW MRNet – will be started by mpriun/srun

Communication Node Filter Example

