Building a Community Infrastructure for Scalable

On-Line Performance Analysis Tools

a.k.a.

Component Based Tool Framework

"CBTF"

Jim Galarowicz

Bill Hachfeld, Don Maghrak

The Krell Institute

CBTF Agenda

- Open|SpeedShop update
- CBTF project overview
- CBTF project current implementation
  - Usage Examples
- CBTF project future directions

Open|SpeedShop status

Highlights:

- Release 1.9.3.4 (www.openspeedshop.org)
- Ported to Cray-XT
- Porting to PPC and Blue Gene
  - Unwinding - signal context, getting at hardware counters
- Improvements to scaling (Pepc 12k run on Jaguar)
- Improvements to online (Dyninst/MRNet) version of Open|SpeedShop
  - Testing on multiple platforms, scaling

Project Goals/Objectives

- ***CBTF Project Goals/Objectives: (Big Picture)***
  - Attempting to build a framework that can be used to connect together the various components that make up a typical performance tool.
  - And, importantly, to be able to deploy them across a petascale system in various configurations, depending on the architecture of that system.
  - Framework should be relatively easy to use.

CBTF Project Team

- ***CBTF Project Team***
  - *Funded Team (jointly funded OASCR / NNSA)*
    - *The Krell Institute*
    - *University of Maryland*
    - *University of Wisconsin*
    - *Oak Ridge National Laboratory*
    - *Lawrence Livermore National Laboratory*
  - *Unfunded Collaborators*
    - *Los Alamos National Laboratory*
    - *Sandia National Laboratories*
    - *Carnegie Mellon University*

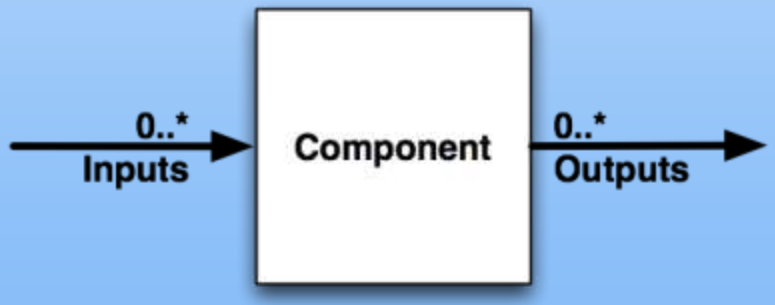
Current CBTF Implementation

- ***CBTF Implementation***
  - ***Krell:*** *Designing and implementing, with full CBTF team input*
    - *Set of libraries that define:*
      - *Component interfaces*
      - *Tool component binding mechanism*
      - *Initial transport network components and binding mechanism*
  - ***Krell:*** *Refactoring Open\|SpeedShop in the existing source tree. (componentizing)*
  - ***UW:*** *Design support, research on binary rewriting, new Dyninst features, as well as MRNet.*
  - ***UMD:*** *Design support, research into Active Harmony component or component network*
  - ***LLNL:*** *Design support, application & tool need identification*
  - ***ORNL:*** *Design support, application & tool need identification, source and wiki hosting.*

Current CBTF Implementation

- ***CBTF Implementation***
  - CBTF API overview
  - Single process API usage example
  - XML specification of component network for single process
  - Single process XML usage example
  - XML based specification of MRNet based component network

Abstract



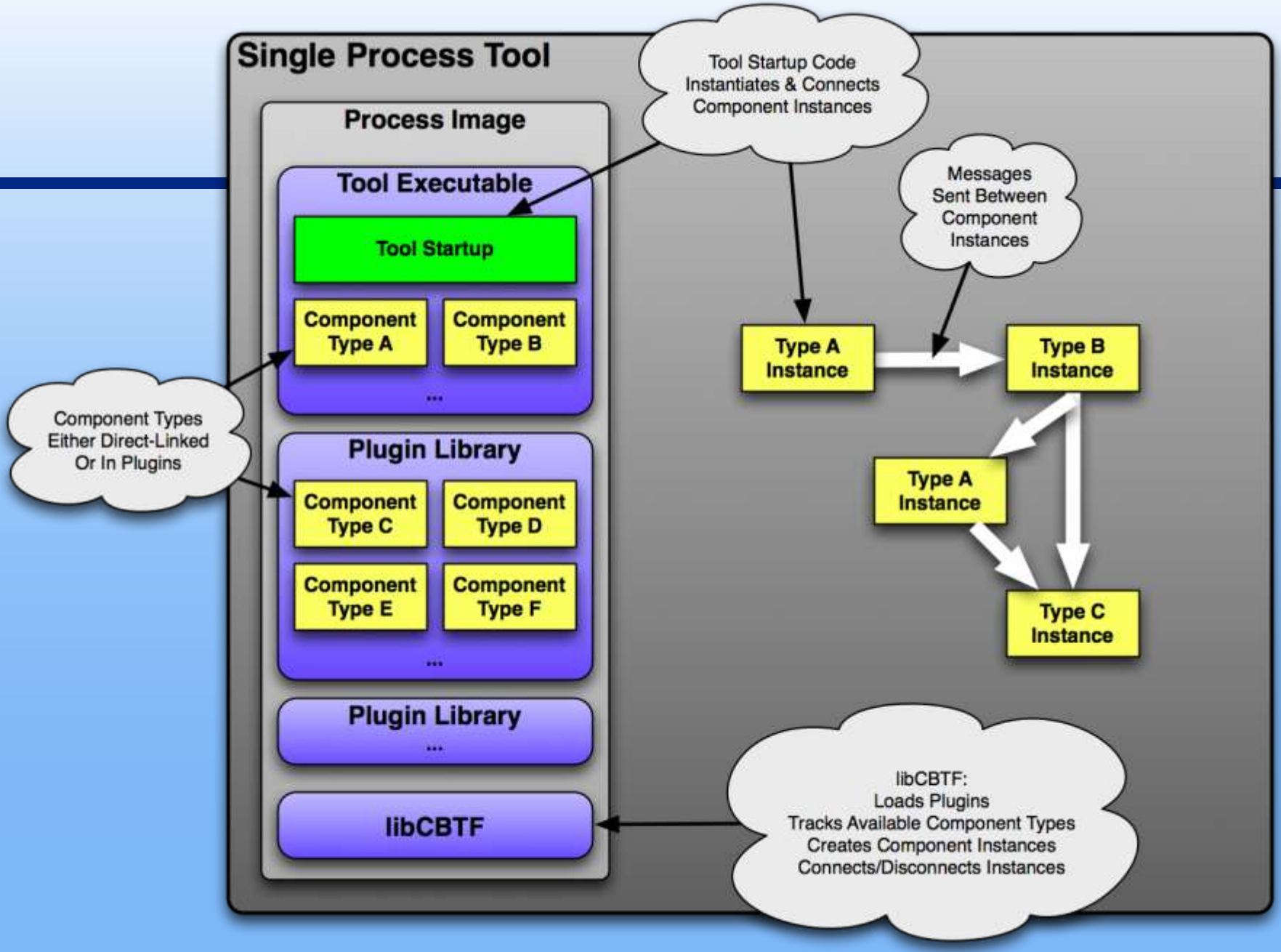
Concrete

CBTF.Component

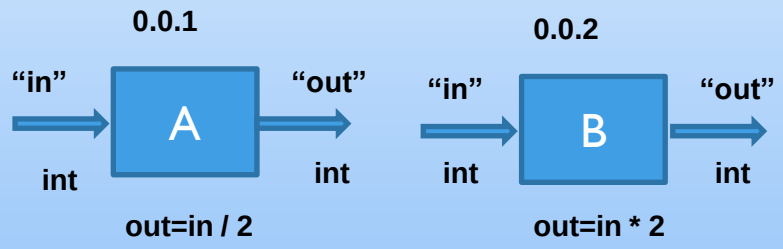
+ registerPlugin(path : Path)
+ getAvailableTypes() : set<Type>
+ getAvailableVersions(type : Type) : set<Version>
+ instantiate(type : Type,
  version : Version) : shared_ptr<CBTF.Component>
+ connect(output_instance : shared_ptr<CBTF.Component>,
  output_name : String,
  input_instance : shared_ptr<CBTF.Component>,
  input_name : String)
+ disconnect(output_instance : shared_ptr<CBTF.Component>,
  output_name : String,
  input_instance : shared_ptr<CBTF.Component>,
  input_name : String)
+ getBuildString() : String
+ getType() : Type
+ getVersion() : Version
+ getInputs() : map<String, Type>
+ getOutputs() : map<String, Type>
# registerFactoryFunction(function : FactoryFunction)
# Component(type : Type, version : Version)
# declareInput<Type>(name : String, handler : InputHandler)
# declareOutput<Type>(name : String)
# emitOutput<Type>(name : String, value : Type)

MyComponentType

+ factoryFunction() : shared_ptr<CBTF.Component>
- MyComponentType()
- InputHandlerA(value : TypeOfA)
- InputHandlerB(value : TypeOfB)
...

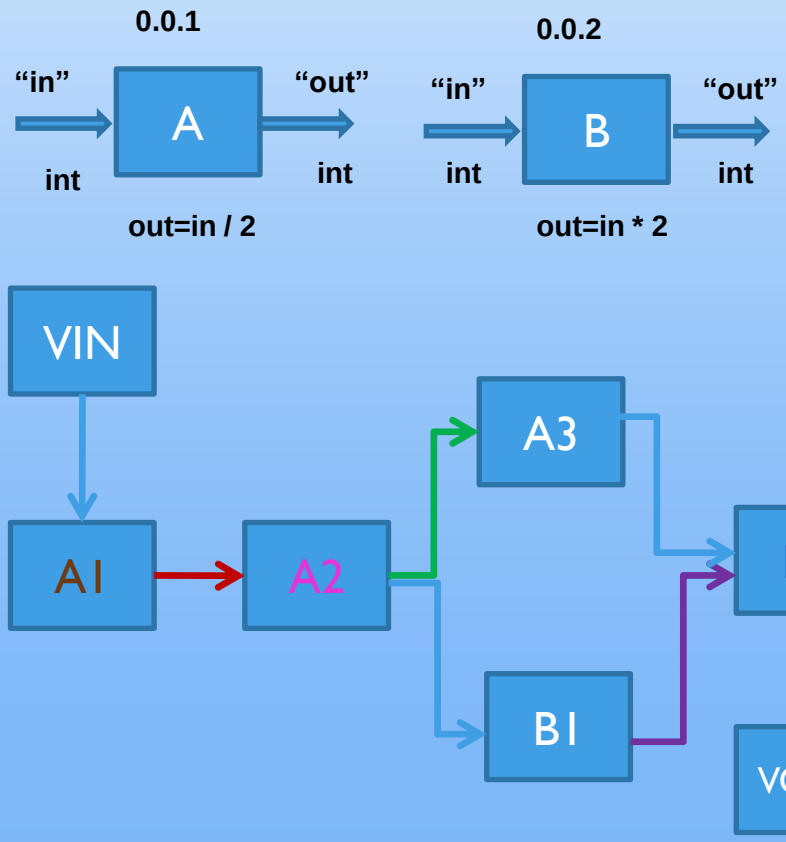

Single Process Tool
Process Image
Tool Executable
Tool Startup
Component Type A
Component Type B
...
Plugin Library
Component Type C
Component Type D
Component Type E
Component Type F
...
Plugin Library
...
libCBTF
Tool Startup Code Instantiates & Connects Component Instances
Messages Sent Between Component Instances
Component Types Either Direct-Linked Or In Plugins
Type A Instance
Type B Instance
Type A Instance
Type C Instance
libCBTF:
Loads Plugins
Tracks Available Component Types
Creates Component Instances
Connects/Disconnects Instances

CBTF API Usage Example




```
//  Component type used by the unit test for the Component class.
class __attribute__ ((visibility ("hidden"))) TestComponentA :
    public Component
{
public:
    /** Factory function for this component type. */
    static Component::Instance factoryFunction()
    { return Component::Instance(reinterpret_cast<Component*>(new TestComponentA()));}
private:
    /** Default constructor. */
    TestComponentA() :Component(Type(typeid(TestComponentA)), Version(0, 0, 1))
    {   declareInput<int>(
            "in", boost::bind(&TestComponentA::inHandler, this, _1)
            );
        declareOutput<int>("out");
    }
    /** Handler for the "in" input.*/
    void inHandler(const int& in)
    { printf("in an instance of TestComponentA, in=%d\n", in);
      emitOutput<int>("out", in / 2 ); }
}; // class TestComponentA
KRELL_INSTITUTE_CBTF_REGISTER_FACTORY_FUNCTION(TestComponentA)
```

CBTF API Usage Example

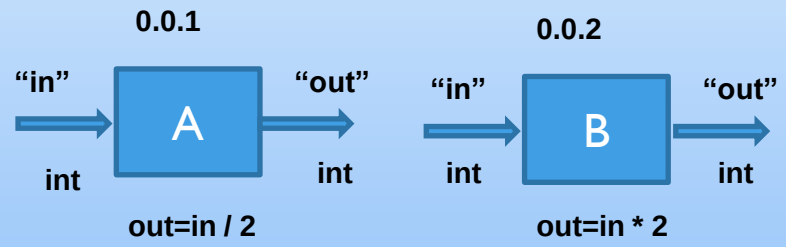


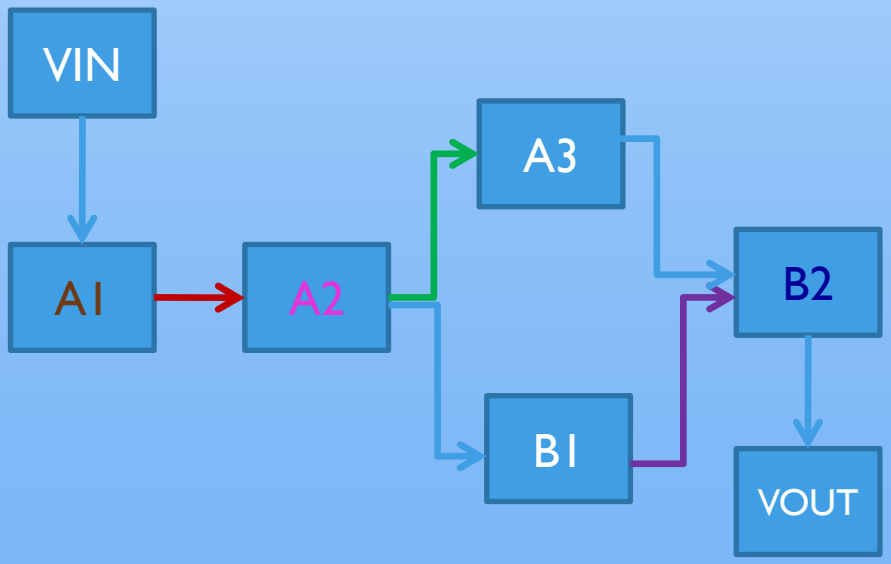

```
# must tell cbtf about plugins avail.
registerPlugin(A)
registerPlugin(B)

# 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");
```

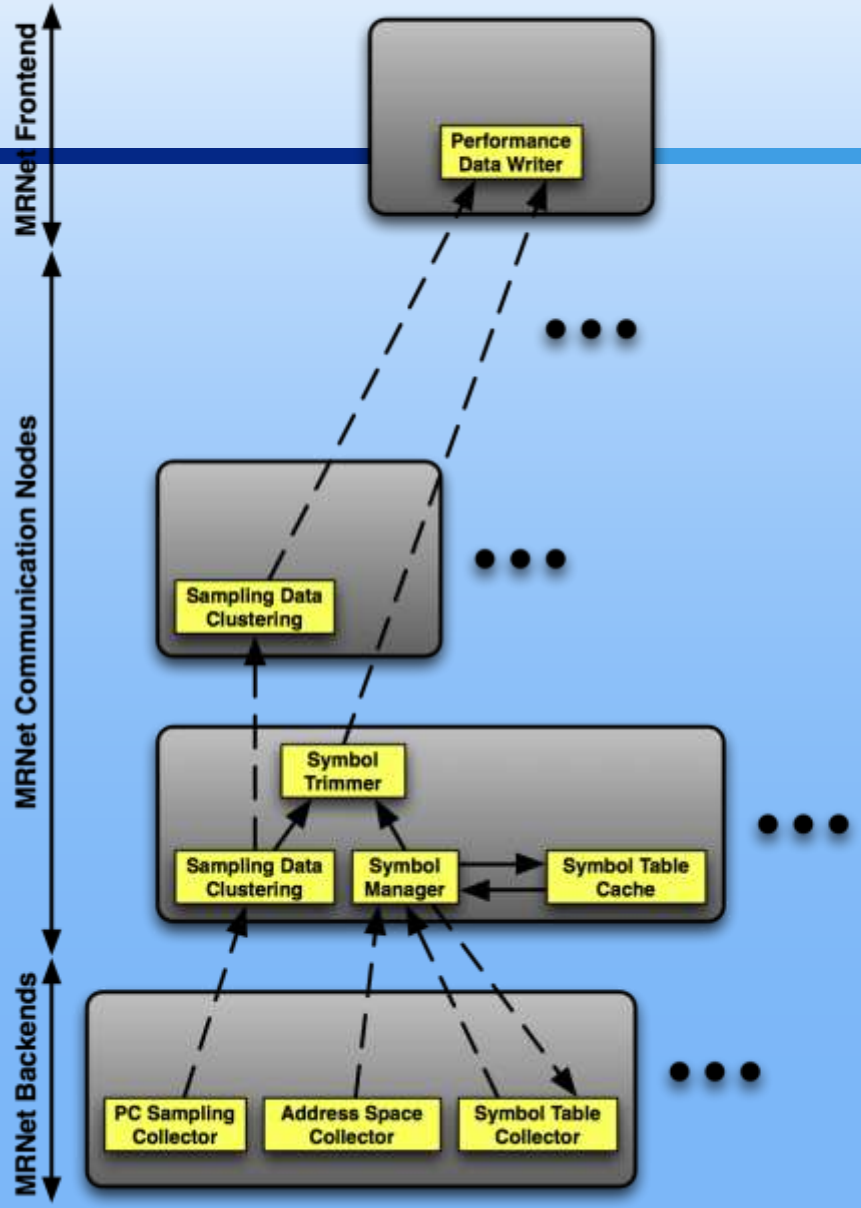
CBTF-XML Usage Example





The call:
Monolithic tool ("tool.xml");
creates the network

```
….
<MonolithicTool xmlns="…/XML/MonolithicTool.xsd">
 <Plugins xmlns="…/XML/Types.xsd">
  <SearchPath>.</SearchPath>
  <Required>plugin.la</Required>
  </Plugins>
  <Network xmlns="…/XML/Types.xsd">
    <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>
  </MonolithicTool>
```



MRNet component network example

libcbtf-mrnet XML example

- ***XML libcbtf-mrnet notes/highlights***
  - Similar to the single process XML definition
    - MRNet tool specification file
      - Specifies the component networks across the MRNet tree
        - What level to put the network in the tree
        - How to connect the components on that level
  - Topology neutral (passes the topology through to MRNet)
    - *Network specifications are relative to the backend, frontend node level*
    - *Can specify **all node levels** get a component network or **node levels based on a relative value***
  - *Only way to specify the MRNet tool networks is through XML*
    - *With single process tool, user could create a component network via the base library in addition to the XML specification file.*

- ***Summary 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

Current Status

- **Current Status**
  - *Held several design meetings with the extended CBTF team*
  - *Created a CBTF wiki hosted at ORNL (http://ft.ornl.gov/doku/cbtfw/start)*
  - *Doing a number of improvements and decompositions in Open|SpeedShop*
  - *Designing and developing the CBTF infrastructure*
    - *libcbtf, libcbtf-xml - first version near completion*
    - *libcbtf-mrnet – two to three months away*
    - *Source hosted at ORNL*

What are the next steps?

- ***Next steps** (Krell)*
  - *Continue to design and develop the CBTF infrastructure*
  - *Begin decomposing Open|SpeedShop internal structure and re-engineer into CBTF components*
    - *Components for Dyninst specific instrumentation*
    - *Components for Offline (libmonitor) instrumentation*
    - *Components for MRNet filtering*
    - *Component for distributed cluster analysis*
    - *Components for the internal Open|SpeedShop infrastructure*
      - *Database, Process Control, Instrumentor, UI clients, Symbol Proc.*

Questions?

Questions?

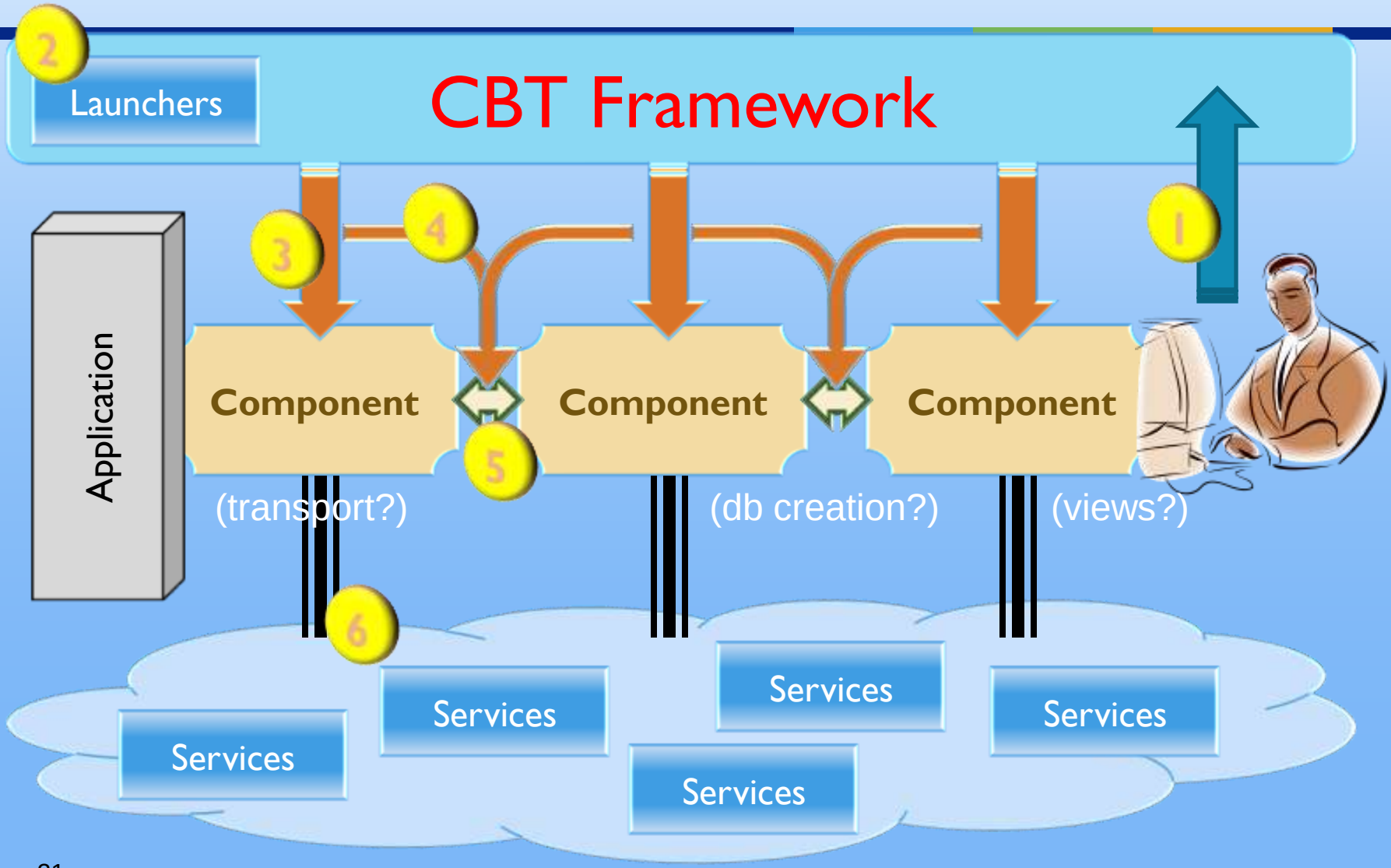
jeg@krellinst.org

Supplemental Information Section

Project Goals/Objectives

- ***Project Goals/Objectives:***
  - *Create a toolbox of components*
    - *For building high-level end user tools*
    - *Quickly build tool prototypes.*
  - *Tools should be easily configurable/adjustable w/o rebuilding.*
  - *Able to mix components from other groups.*
  - *We would like contributors to define the interfaces with us, or give feedback, so that we can share components later in both directions.*

Conceptual Picture



Initialization Order

1. User starts the tool which starts the framework
2. Framework uses launchers to start all components
3. Framework initiates pipeline components
4. Framework identifies data connections
5. Framework connects pipeline components
6. Pipeline components rely on service components (existing stand alone packages like libmonitor or a specific Dyninst component).

CBTF Component Interface Library

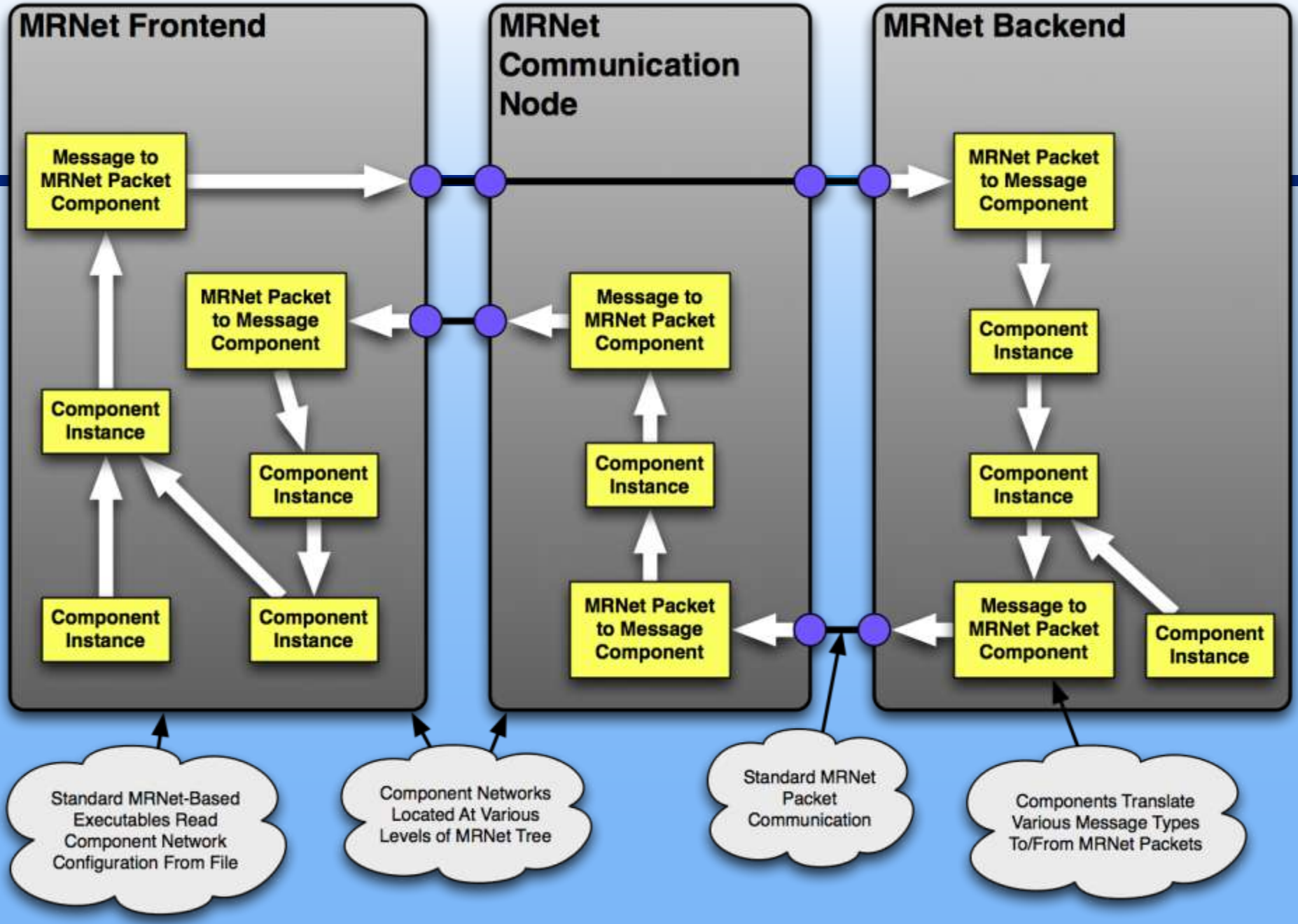
- ***Details of the Component Interface Library (libcbtf)***
  - Provides Cores Services:
    - Component Abstraction
      - Main class: Component.hpp
      - Key point: Do not need source header file, all that is needed is object code
    - Metadata
      - Information about the Component
        - Type, Version, Build String, Inputs, Outputs
    - Interconnection
      - Forms the connection between components
      - Arguments passed via C++ callback function (shared pointer)

CBTF Component Binding Interface

- ***Details of the Component Binding Interface (libcbtf-xml)***
  - Provides XML-Based Definition of Single-Process Component Networks
    - Gives ability to create a component network within a single process
      - Monolithic tool (tool.xml) creates the network for the user
    - Uses the low level library: libcbtf under the hood
    - XML schema defines what is a legal specification
    - Three options to find available plugins containing components:
      - Direct file specification
      - Plugin Path environment variable
      - Install Path

CBTF Transfer Mechanism Component

- ***CBTF Transfer Mechanism Component (libcbtf-mrnet)***
  - Provides XML-Based Definition of MRNet-Based Distributed Component Networks
    - Must use the XML definitions to define MRNet component network
      - Specifies what the components are
      - Where to put the components
      - How to connect the components
    - Assumptions:
      - At each level of the network we assume the same component network for all nodes (homogenous network)
  - Depends on libcbtf, libcbtf-xml, and MRNet


MRNet Frontend
MRNet Communication Node
MRNet Backend
Message to MRNet Packet Component
MRNet Packet to Message Component
Component Instance
Component Instance
Component Instance
Component Instance
MRNet Packet to Message Component
Message to MRNet Packet Component
Component Instance
MRNet Packet to Message Component
Component Instance
Component Instance
Message to MRNet Packet Component
Component Instance
Standard MRNet-Based Executables Read Component Network Configuration From File
Component Networks Located At Various Levels of MRNet Tree
Standard MRNet Packet Communication
Components Translate Various Message Types To/From MRNet Packets

Project Results/Deliverables

- ***Long Term Project Results/Deliverables:***
  - *Set of reusable components for creating performance tools*
  - *Proof of concept:*
    - *Special purpose tool based on need at ORNL*
    - *Modified version of gprof using reusable components*
    - *Tool or Open|SpeedShop experiment based on Active Harmony*
  - *Components that do online data aggregation, reduction, filtering, and transfer at high scale*
  - *A new, more modular Open|SpeedShop performance tool*