CScADS 2011: CBTF Presentation

Component Based Tool Framework CScADS 2011 Workshop August 1, 2011

David Montoya, LANL

Jim Galarowicz, Krell Institute

CBTF Project Goals and Objectives

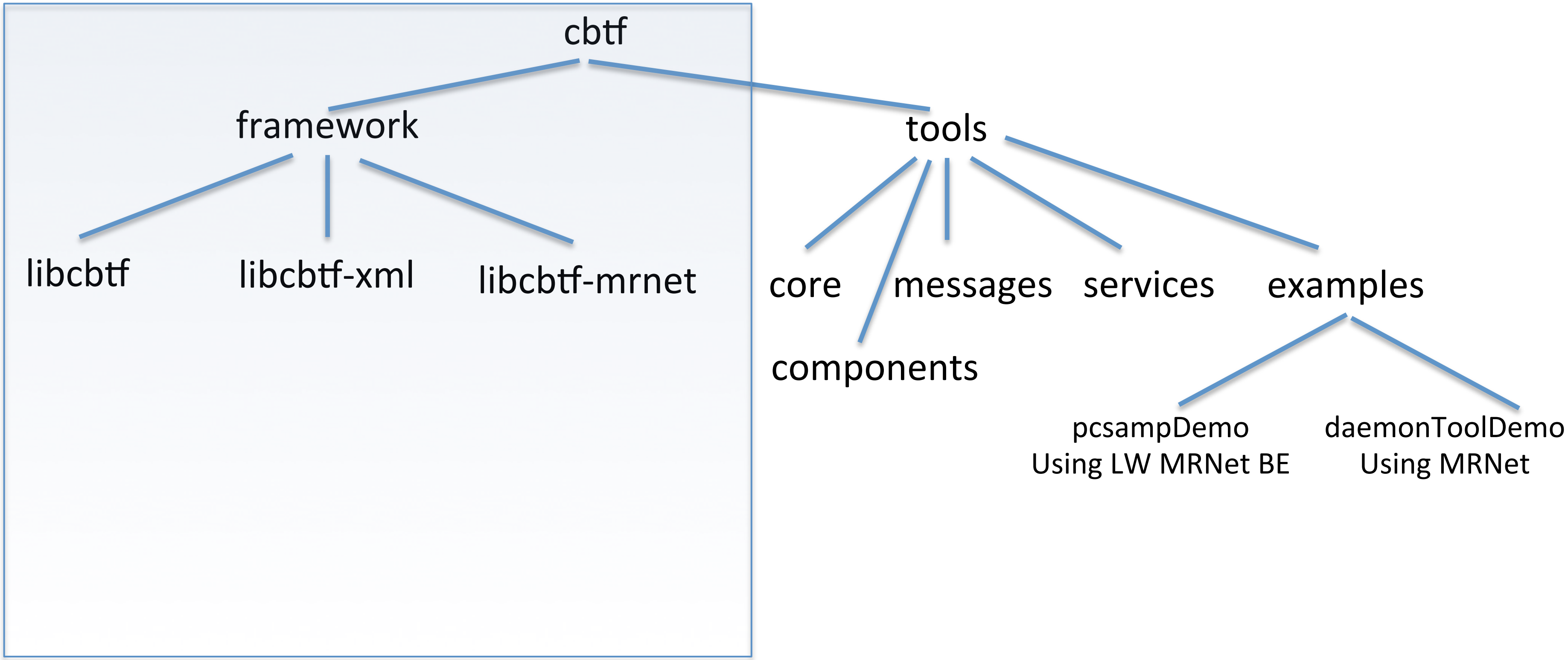
- ***Project Goals/Objectives:***
  - *Provide an infrastructure and mechanisms to:*
    - *Create a set of highly scalable, reusable components for building high-level end user tools and/or quickly building tool prototypes.*
    - *Establish a compatibility constraint structure for component integration.*
    - *Recreate Open|SpeedShop from the CBTF provided components and services*
  - *Allow for tools to be easily developed by creating a network of components without rebuilding core infrastructure.*
  - *Ability to integrate components from several groups and/or vendors into new tools.*

CBTF Project Team

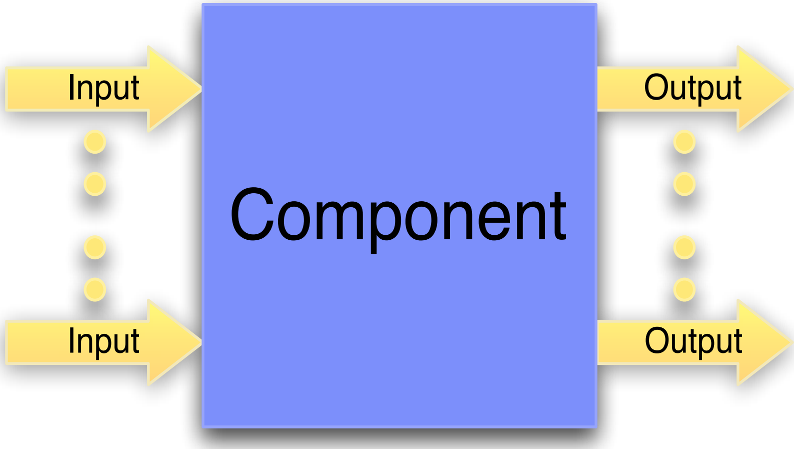
- **Project Team**
  - The Krell Institute
  - University of Maryland
  - University of Wisconsin
  - Oak Ridge National Laboratory
  - Lawrence Livermore National Laboratory
  - Los Alamos National Laboratory
  - Sandia National Laboratories
  - Carnegie Mellon University
  - Others welcome……
- **Co-funded by NNSA and Office of Science**

CBTF: Framework Infrastructure

- Infrastructure libraries defining/supporting: components, component networks, and distributed component networks.



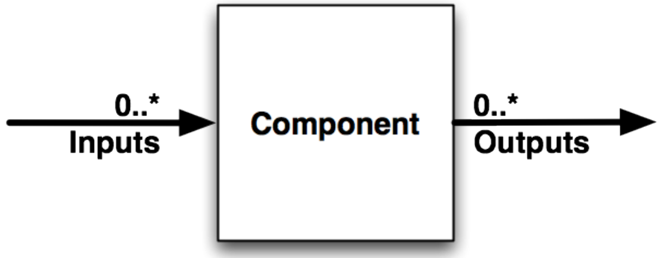
CBFT: Components



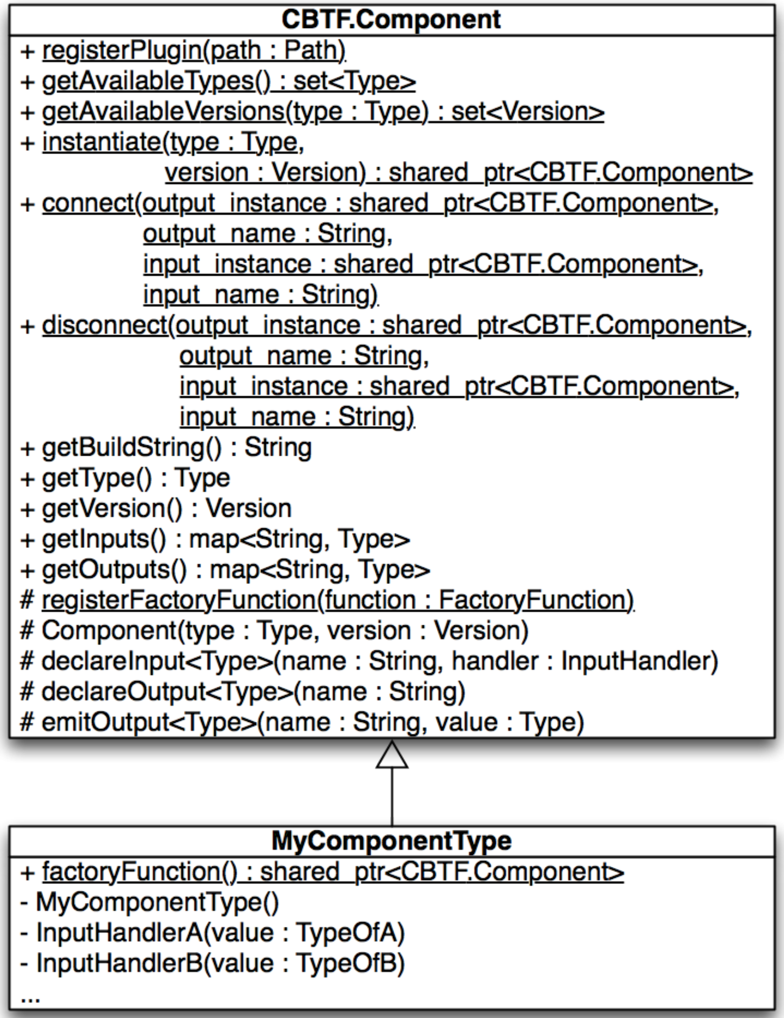
- **Data-Flow Model**
  - Accepts Inputs
  - Performs Processing
  - Emits Outputs
- **C++ Based**
- **Provide Metadata**
  - Type & Version
  - Input Names & Types
  - Output Names & Types
- **Versioned**
  - Concurrent Versions
- **Packaging**
  - Executable-Embedded
  - Shared Library
  - Runtime Plugin

CBTF Component Interfaces

Abstract



Concrete



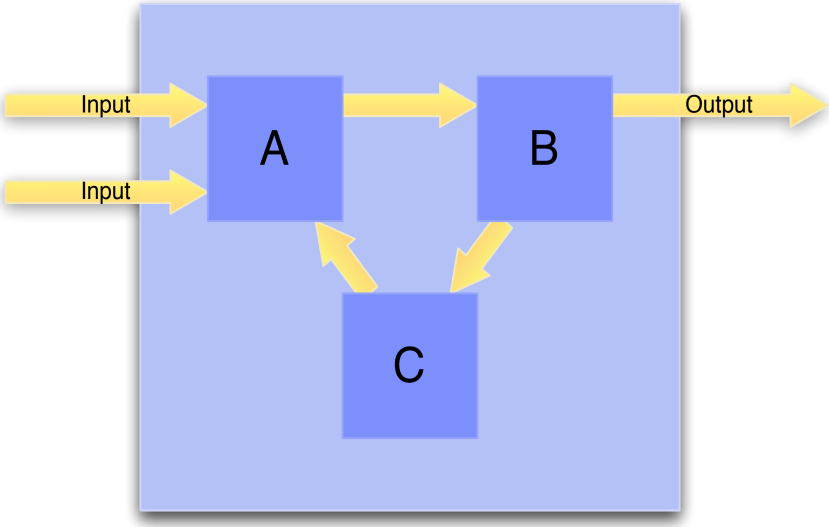
CBTF.Component

```
+ 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)
...
```

CBTF: Component Networks



- **Components**
  - Specific Versions
- **Connections**
  - Matching Types
- **Arbitrary Component Topology**
  - Pipelines
  - Graphs with cycles
  - ….
- **Recursive**
  - Network itself is a component
- **XML-Specified**

CBTF: Component Networks (API)

```
# 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”);
```

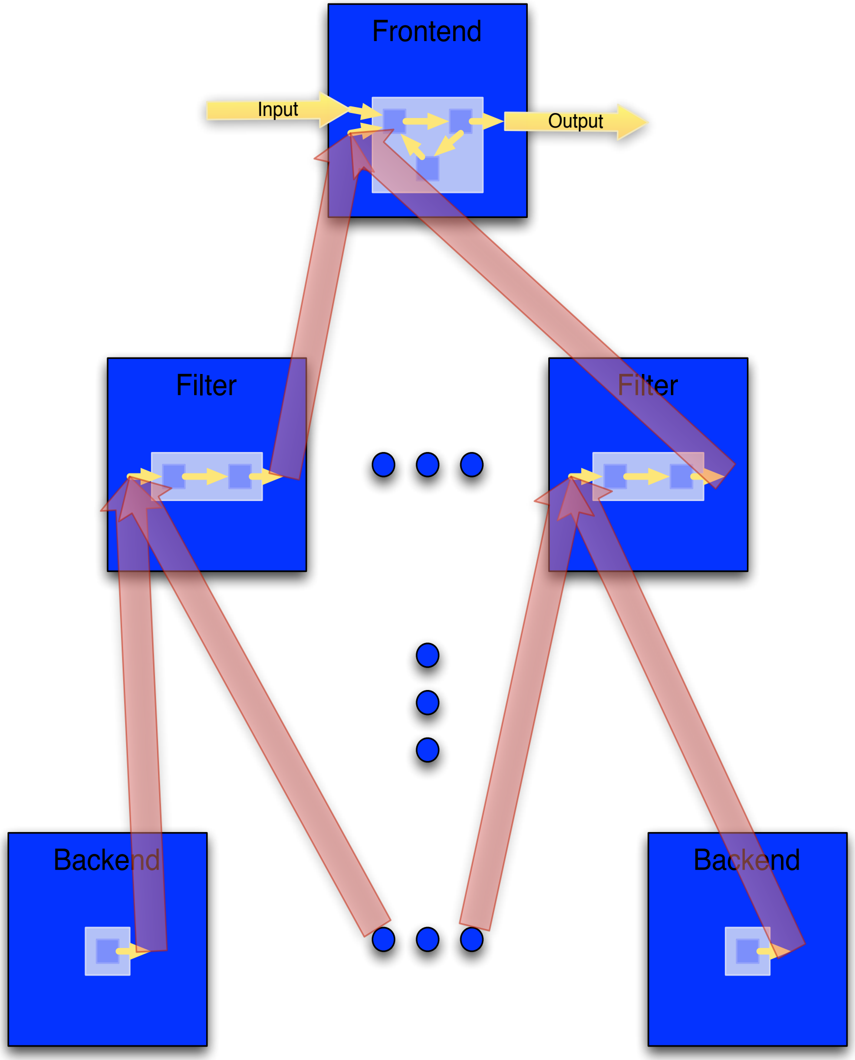
- Source code snippet for example component network creation and connection using the CBTF API
- Full source is available in source tree.
- Key points:
  - Using the API alone will create single process component networks.
  - Components operate on the data and then push it to the next components which are connected to it.
  - Messages passed between components within a single process are simple direct handing off of pointers to C++ objects.

CBTF: Component Networks (XML)

```
….
<Type>ExampleNetwork</Type>
<Version>1.2.3</Version>
<SearchPath>.:/opt/myplugins</SearchPath>
<Plugin>myplugin</Plugin>
    <Component>
      <Name>Component-A1</Name>
     <Type>TestComponentA</Type>
    </Component>
…
 <Network>
…
    <Connection>
      <From>
        <Name>Component-A1</Name>
        <Output>out</Output>
      </From>
      <To>
          <Name>Component-A2</Name>
          <Input>in</Input>
       </To>
     </Connection>
…
</Network>
```

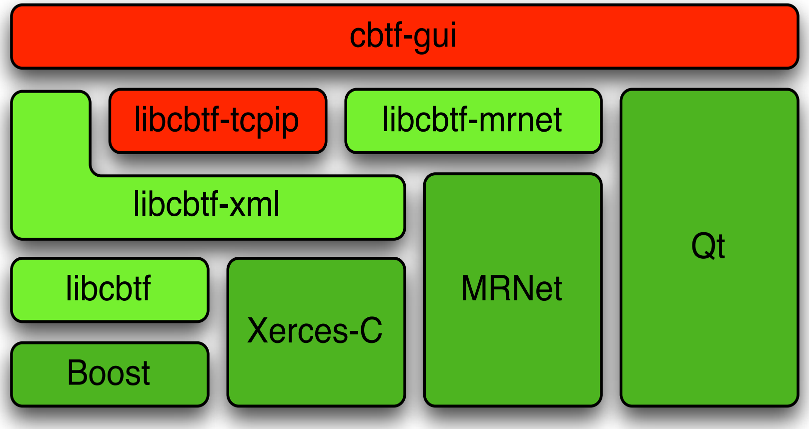
- XML code snippet for example component network creation and connection using XML specification files.
- Full XML source is available in source tree.
- Key points:
  - Does all of the CBTF definition and component connection by automatically generating the single process component network from a network specification written in XML.
  - XML specification file contains all the necessary information about each of the components and how they are connected to each other

CBTF: Distributed Component Networks



- **Transport Layer (MRNet based)**
- **Per-Node Component Networks**
  - Homogenous Within Tree Levels
  - Heterogeneous Between Tree Levels
- **Named Streams**
  - Up and Down
  - Connect Networks
- **Also Recursive**
- **Also XML-Specified**
- **Supports LW MRNet**

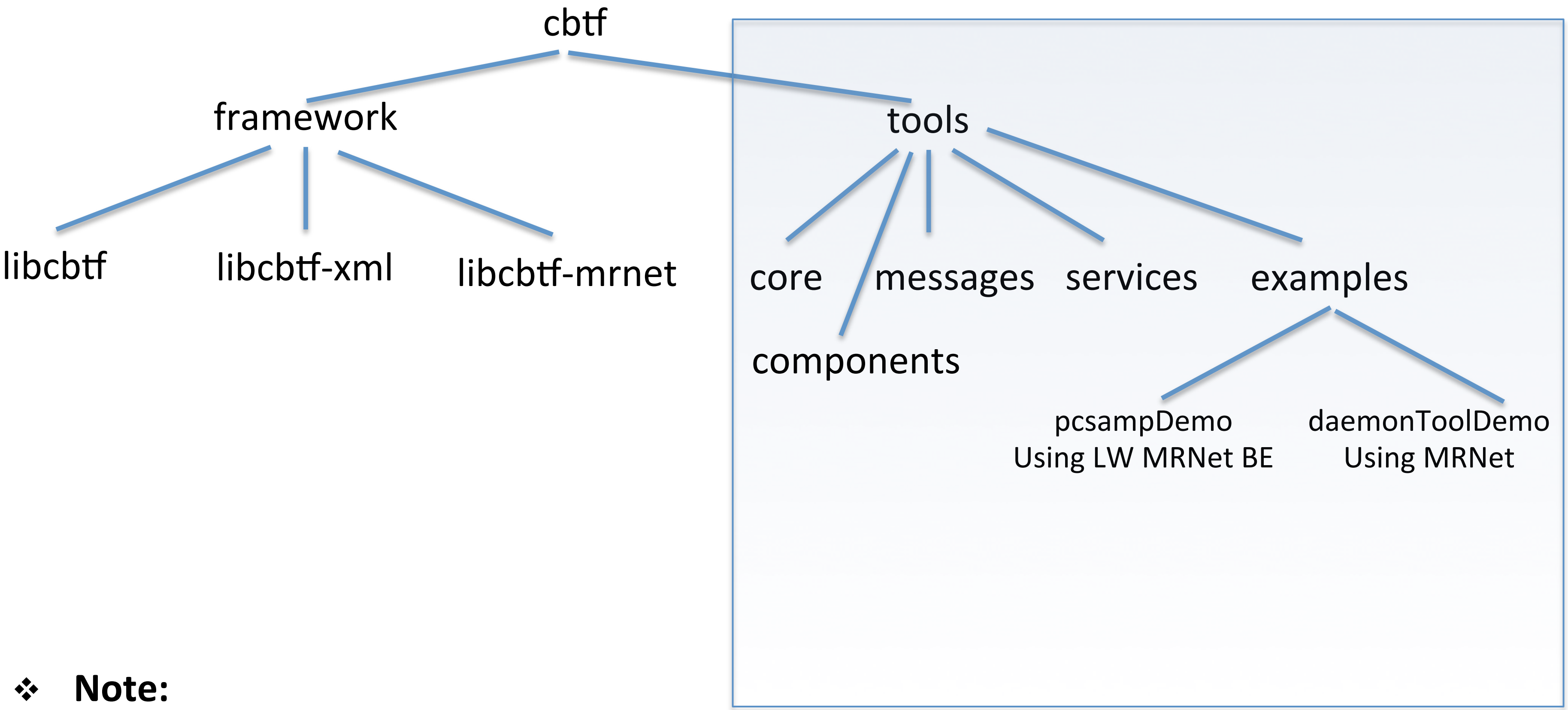
CBTF: Software Stack (Framework)



- **Tool-Type Independent**
  - Performance Tools
  - Debugging Tools
  - etc…
- **Completed Components**
  - Base Library (libcbtf)
  - XML-Based Component Networks (libcbtf-xml)
  - MRNet Distributed Components (libcbtf-mrnet)
- **Planned Components**
  - TCP/IP Distributed Component Networks
  - GUI Definition of Component Networks

CBTF: Tool Building Support

- **To enable tool builders to get started**
  - Add a tool building side to CBTF (tools subdirectory under cbtf)



- **Note:**
  - The directory structure is subject to change
  - daemonToolDemo doesn't rely on any service, message, or core "tools" code

CBT: Software Stack (Services,Messages,Components, Core)

Examples:

timer service: service library

pcsamp collector: collector plugin

pcsamp data: message type

aggregated address data: message type

❖ Services

- ➢ Libraries (C or C++) of functionality that don't fit into the data-flow model
- ➢ Collection services (Unwinding, Timer, HWC (PAPI), …)

❖ Messages

- ➢ Defines the data that is exchanged between data-flow components.
  - Performance data
  - Event notification (thread state, …
  - Control (process, thread, instrumentation,…)

❖ Components

- ➢ Follow proper dataflow model
- ➢ Exchange messages
- ➢ Examples: Filter components (aggregator)

❖ Core

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

CBT: Software Stack (Future Tools)

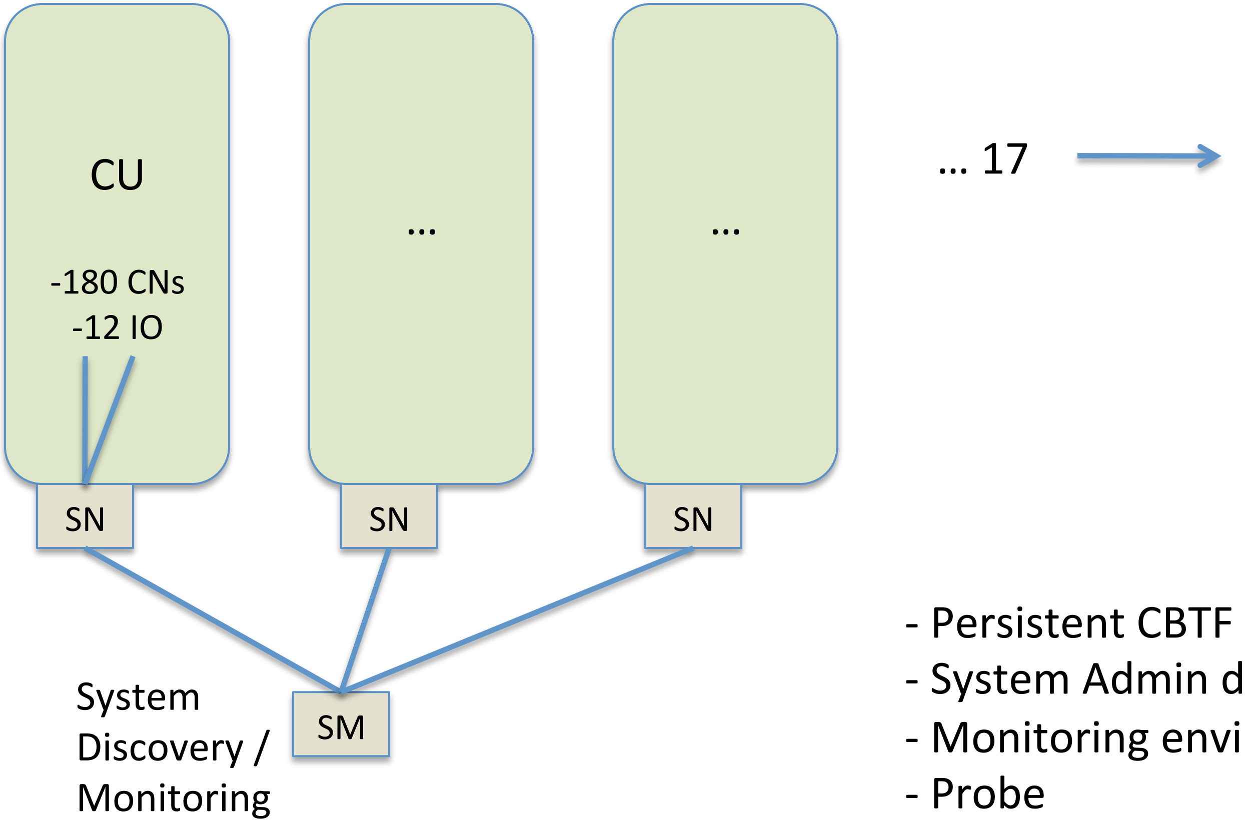
- **Open|SpeedShop**
  - Using Services, Messages, Core built using CBTF infrastructure
  - Full fledged multipurpose performance tool
- **Customized Tools**
  - Use the CBTF infrastructure, not necessarily any support from the ***tools*** support sub-directories
  - If tool creator sees a useful service in ***tools***, they can choose to use it (along with any message and/or core library).
  - Aimed at specific tool needs determined by application code teams

Initial Tool Implementation

- **Usage Scenarios (non-traditional)**
  - System Monitoring Services (persistent)
  - System Administration Investigation service
- **Targets**
  - Persistent /non-persistent implementation
  - Scalable discovery environment
  - Assess CBTF integration as a service
  - How does it fit in an integrated services stack

RoadRunner example

System Discovery / Monitoring Implementation

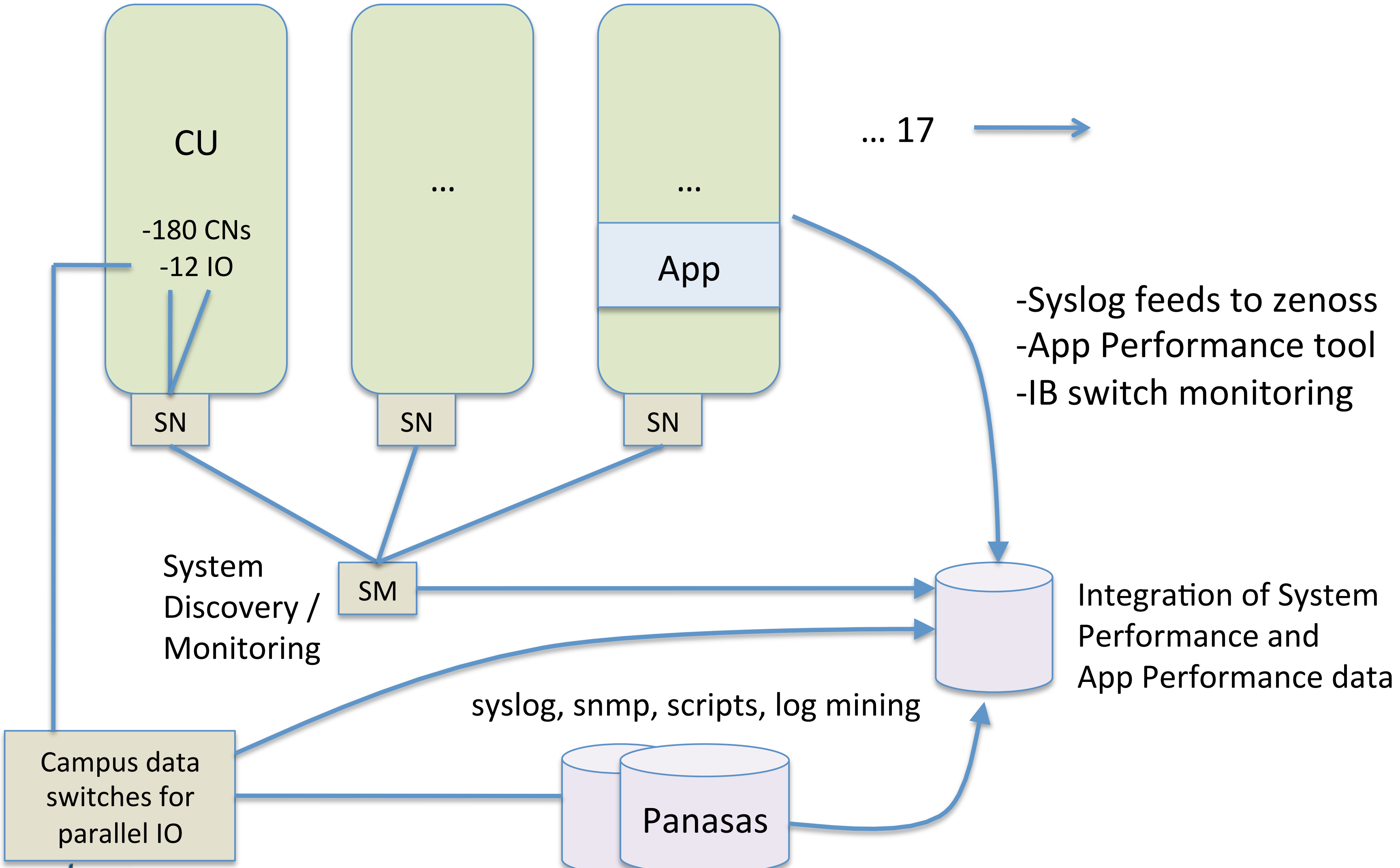


- Persistent CBTF implementation
- System Admin discovery tool
- Monitoring environment (focused)
- Probe

SN - Service node

RoadRunner example

Integrated Health and Performance Implementation


CU
-180 CNs
-12 IO
…
…
App
SN
SN
SN
… 17
SM
Campus data switches for parallel IO
Panasas
syslog, snmp, scripts, log mining

System Discovery / Monitoring

-Syslog feeds to zenoss
-App Performance tool
-IB switch monitoring

Integration of System Performance and App Performance data

Initial Tool Implementation

- **Assessment Approach**
  - Utilize non-tools developers
  - Assess architecture understanding
  - Documentation
  - Installation
  - Demo components
  - Integration Ability
  - Ease of development
- **After a month**
  - Learning curve points identified
  - Small successes
  - Lot's of work to do…..

CBTF Next Steps

- ***Next steps for CBTF:***
  - Create CBTF Tutorial, Step by Step Instructional Info
  - More detailed documentation of examples, demo tools
  - GUI tool for CBTF component network configuration. The XML files get tedious to write and verify by hand…
  - TCP/IP library implementation and test. (libcbtf-tcpip)
  - Tool start up investigation/implementation (launchmon, libi, …)
    - Several variations dependent on platform type (BG/P, Cray)
  - Tool services, messages, component creation to support more types of collection
  - Continue porting to Cray and Blue Gene platforms
  - More filtering components for MRNet communication node deployment

CBTF Information

- ***Where to find information***
  - CBTF wiki: http://ft.ornl.gov/doku/cbtfw/start
- ***Source Access***
  - ***Friendly access available through request***
  - ***Source hosted at ORNL git repository***
- ***Tutorial coming***
- ***Technical paper being worked on***

CBTF Questions?

Questions?

jeg@krellinst.org

dmont@lanl.gov

cbt_framework@krellinst.org