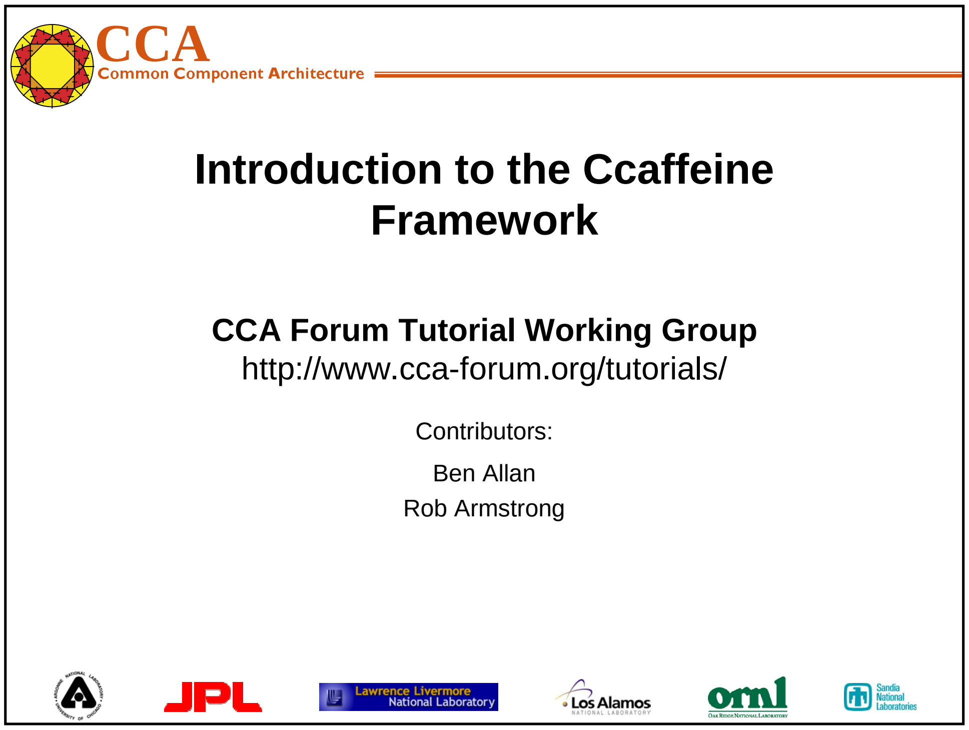


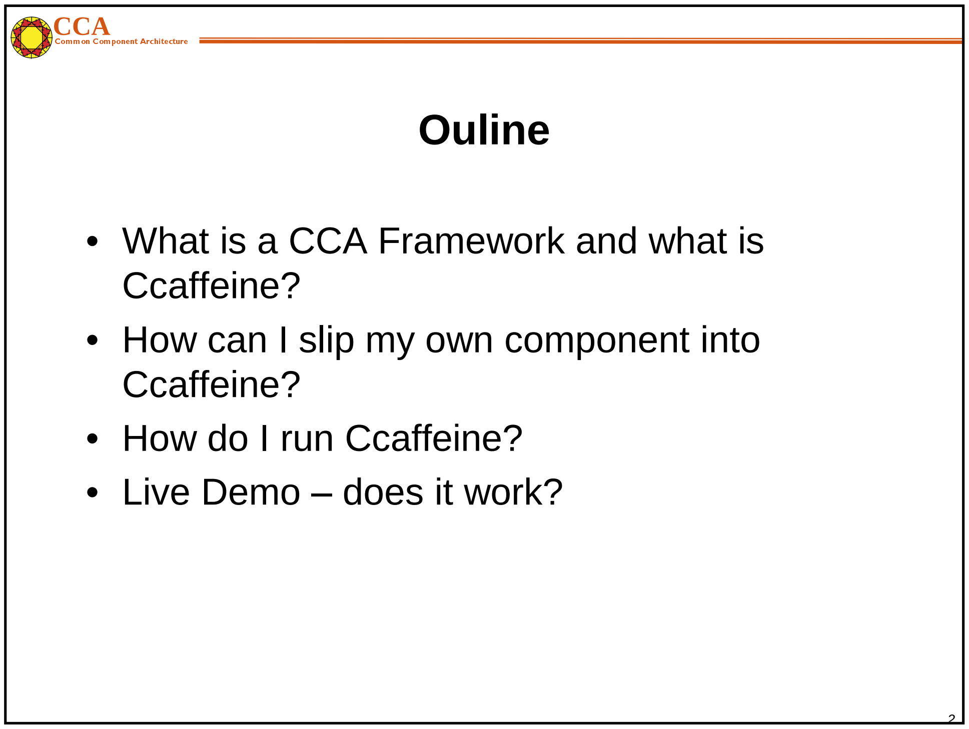
CCA
Common Component Architecture

Introduction to the Ccaffeine Framework

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/

Contributors:

Ben Allan
Rob Armstrong



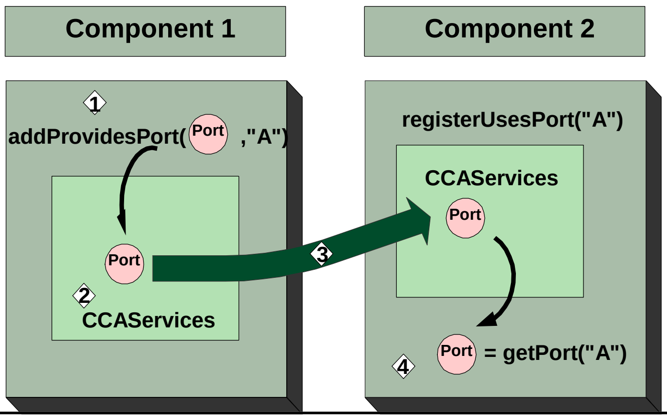
CCA
Common Component Architecture

Ouline

- What is a CCA Framework and what is Ccaffeine?
- How can I slip my own component into Ccaffeine?
- How do I run Ccaffeine?
- Live Demo – does it work?

CCA What CCA compliant framework is expected to do …

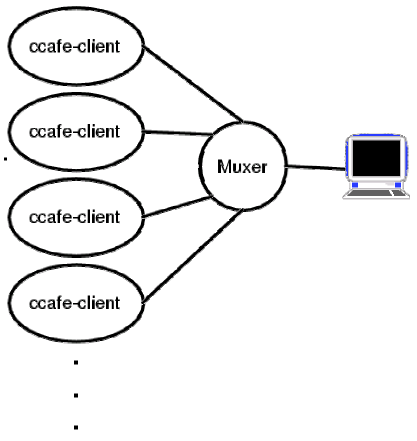
- Exchange interfaces among components without one component needing to know more about the other than the interface itself.



Interactive Parallel Components: what Ccaffeine does

- Executable ccafe-client:
  - PVM, MPI, or whatever is used for communication between clients.
  - Muxer enforces “single process image” of SPMD parallel computing.
- HOWTO:
  http://www.cca-forum.org/ccafe/
  - Build Ccaffeine
  - Run Ccaffeine

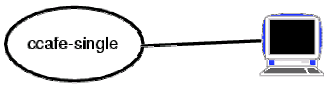
http://www.cca-forum.org/ccafe/



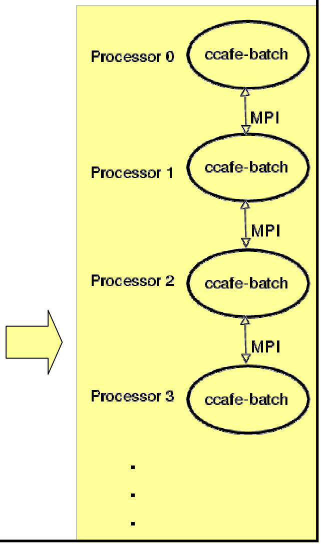
CCA
Common Component Architecture

Ccaffeine comes in two other “flavors” and a GUI.

- Single process executable: ccafe-single
  - really useful for debugging



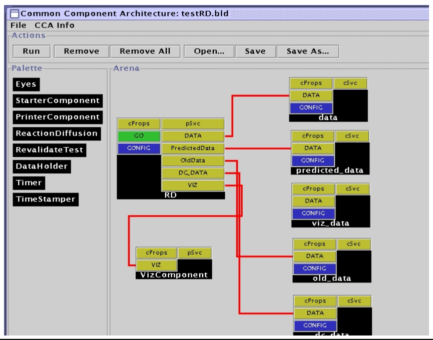
- Batch executable: ccafe-batch
  - when all you want to do is run it.

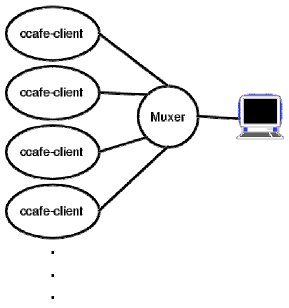


CCA
Common Component Architecture

How to run Ccaffeine:

- Ccaffeine interactive language: “benSpeak”
  - used to configure batch and interactive sessions.
  - Allows useful “defaults.”
  - Allows the GUI to talk over a socket.



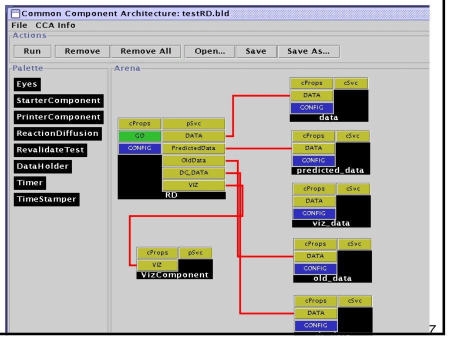


CCA
Common Component Architecture

Configuration Commands: interactive or as an “RC” file or as a Batch run

Sample:

```
#!ccaffeine bootstrap file.
# ------- don't change anything ABOVE this line.-------------
# where to find components:
path set /home/rob/cca/dccafe-classic/cxx/dc/component
# load components into the “pallet”
repository get StarterComponent
repository get TimeStamper
repository get Timer
repository get PrinterComponent
repository get RevalidateTest
```



7

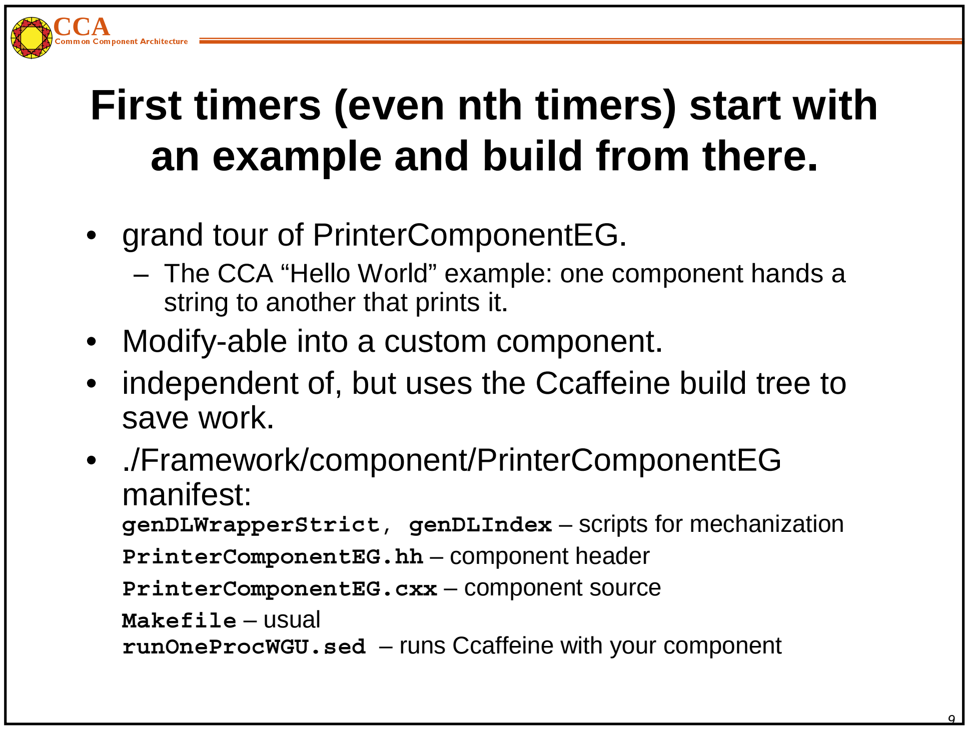
CCA
Common Component Architecture

Creating a Ccaffeine component*

- Beyond the CCA spec, Ccaffeine needs:
  – Your component to be built compatibly, consider:
    - Special libraries, shared and otherwise
    - Compiler compat. (i.e. use the same compiler)
    - Loaded dynamically, or statically?
    - Defaults: libblas, linpack, g++, dynamic .so components
  – To know where your component is
    - set path default/path:my/own/path
  – To know how to load your component
    - needs a “.cca” text file. (auto gen’d in the example)
- Whirlwind tour
  – start from the tutorial PrinterComponent example.

*classic style component not Babel

8

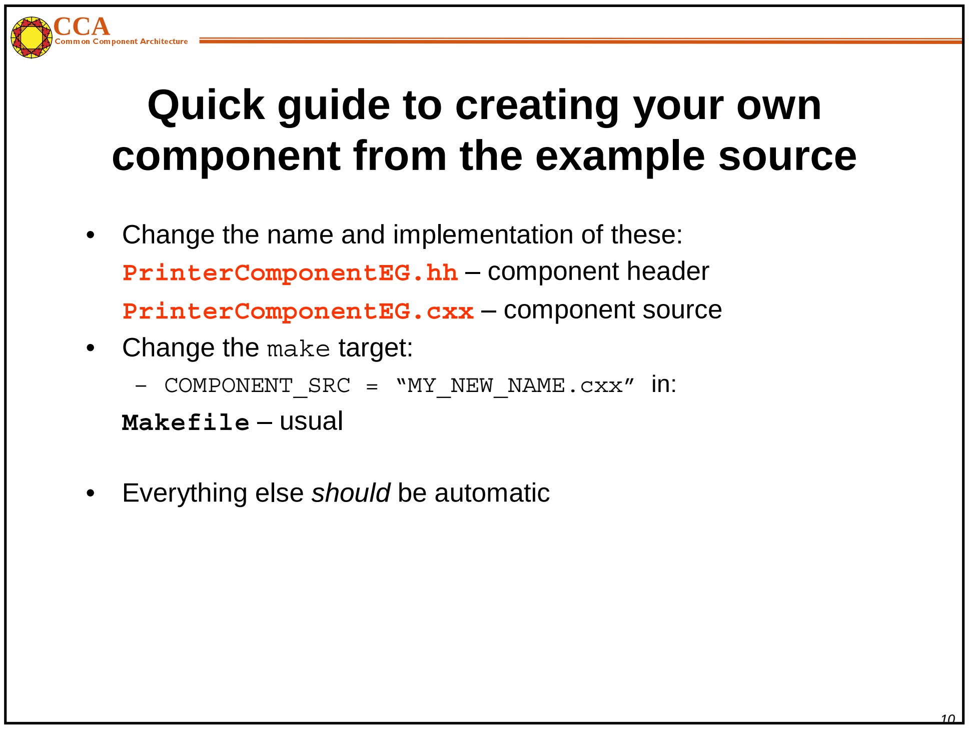


CCA Common Component Architecture

First timers (even nth timers) start with an example and build from there.

- grand tour of PrinterComponentEG.
  - The CCA “Hello World” example: one component hands a string to another that prints it.
- Modify-able into a custom component.
- independent of, but uses the Ccaffeine build tree to save work.
- ./Framework/component/PrinterComponentEG manifest:
  `genDLWrapperStrict`, **`genDLIndex`** – scripts for mechanization
  `PrinterComponentEG.hh` – component header
  `PrinterComponentEG.cxx` – component source
  `Makefile` – usual
  `runOneProcWGU.sed` – runs Ccaffeine with your component

9



CCA Common Component Architecture

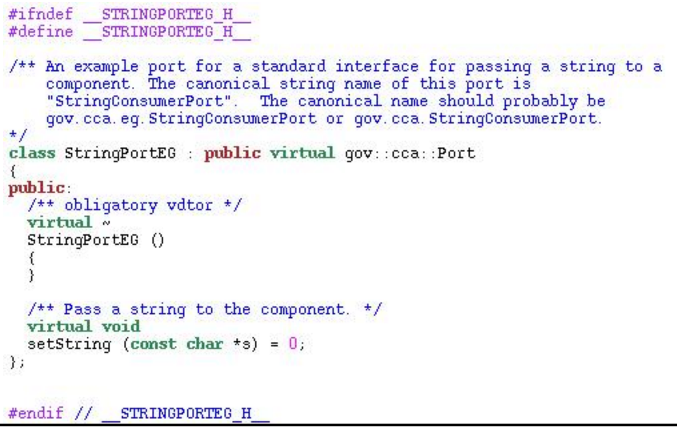
Quick guide to creating your own component from the example source

- Change the name and implementation of these:
  `PrinterComponentEG.hh` – component header
  `PrinterComponentEG.cxx` – component source
- Change the `make` target:
  - `COMPONENT_SRC = “MY_NEW_NAME.cxx”` in:
  `Makefile` – usual
- Everything else *should* be automatic

10

PrinterComponent takes a string and prints it out

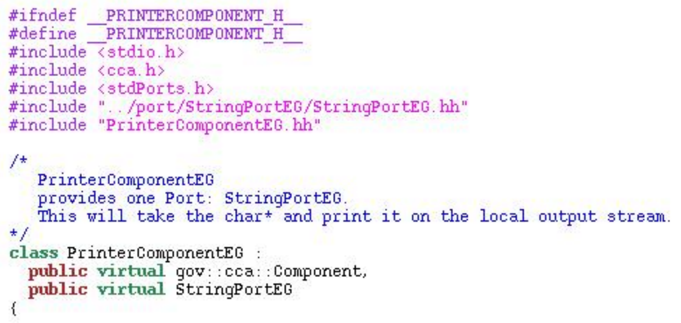
- It exposes a single interface for use: “StringPortEG” from the file StringPortEG.hh:



```
#ifndef __STRINGPORTEG_H__
#define __STRINGPORTEG_H__

/** An example port for a standard interface for passing a string to a
    component. The canonical string name of this port is
    "StringConsumerPort".  The canonical name should probably be
    gov.cca.eg.StringConsumerPort or gov.cca.StringConsumerPort.
*/
class StringPortEG : public virtual gov::cca::Port
{
public:
  /** obligatory vdtor */
  virtual ~
  StringPortEG ()
  {
  }

  /** Pass a string to the component. */
  virtual void
  setString (const char *s) = 0;
};


#endif // __STRINGPORTEG_H__
```

The component must inherit all the stuff of a “normal” CCA component

- Must implement gov::cca:Component
  - Choose to implement StringPortEG in the component
  - Header file: PrinterComponentEG.hh:



```
#ifndef __PRINTERCOMPONENT_H__
#define __PRINTERCOMPONENT_H__
#include <stdio.h>
#include <cca.h>
#include <stdPorts.h>
#include "../port/StringPortEG/StringPortEG.hh"
#include "PrinterComponentEG.hh"

/*
   PrinterComponentEG
   provides one Port: StringPortEG.
   This will take the char* and print it on the local output stream.
*/
class PrinterComponentEG :
  public virtual gov::cca::Component,
  public virtual StringPortEG
{
```

CCA
Common Component Architecture

The component must inherit all the stuff of a “normal” CCA component

- Implement setServices()
- Implement StringPortEG
  - implement setString()

```
private:

  gov::cca::Services *
    svc;

public:

  PrinterComponentEG ()
  {
    svc = 0;
  }

  virtual ~
  PrinterComponentEG ()
  {
  }

  virtual void
  setServices (gov::cca::Services * svc);

  /** Implements StringPortEG */
  virtual void
  setString (const char *s);

};

#endif // __PRINTERCOMPONENT_H__
```

CCA
Common Component Architecture

PrinterComponentEG Implementation

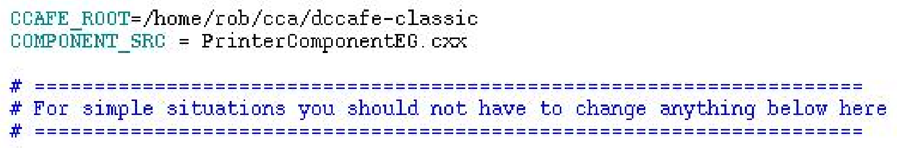
- File PrinterComponentEG.cxx:

```
#include <stdio.h>
#include <cca.h>
#include <stdPorts.h>
#include "../port/StringPortEG/StringPortEG.hh"
#include "PrinterComponentEG.hh"

void
PrinterComponentEG::setServices (gov::cca::Services * svc)
{
  this->svc = svc;
  svc->addProvidesPort (this,
                        svc->createPortInfo ("printer_port",
                                             "StringPortEG", 0));
}

void
PrinterComponentEG::setString (const char *s)
{
  FILE *fp = fopen ("/dev/tty", "w");
  fprintf (fp, "PrinterComponent says: %s\n", s);
  fclose (fp);
}
```

14

To change this implementation to your own, modify the Makefile

- Change the name of the Component from PrinterComponentEG to whatever
  - leave the extension .cxx
- Makefile (partial) listing:

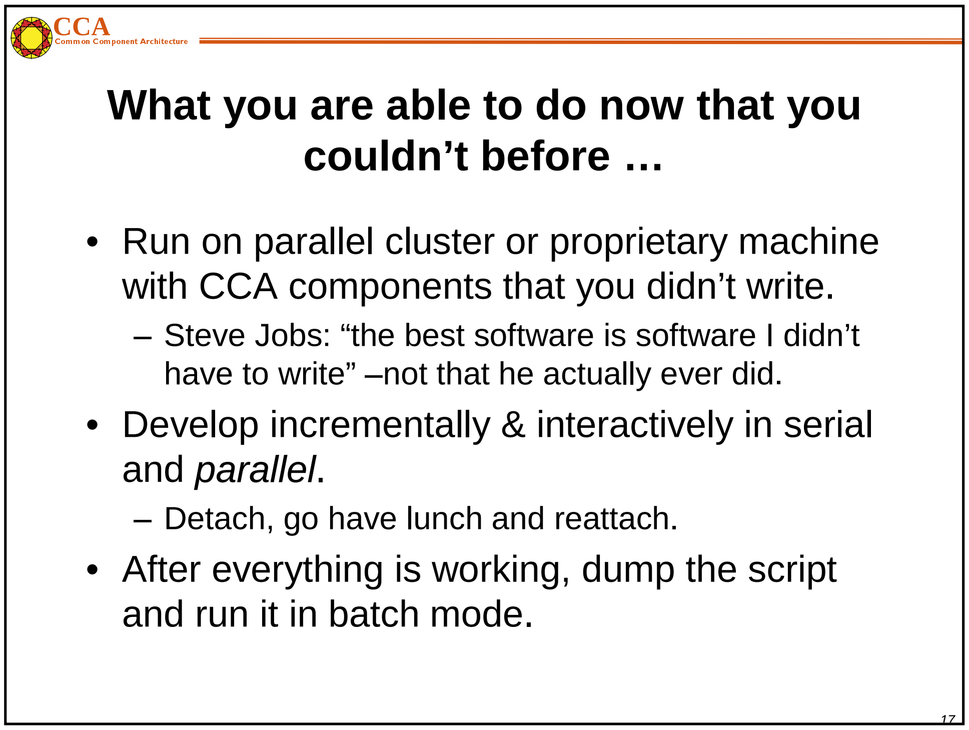


```
CCAFE_ROOT=/home/rob/cca/dccafe-classic
COMPONENT_SRC = PrinterComponentEG.cxx

# ====================================================================
# For simple situations you should not have to change anything below here
# ====================================================================
```

Type “make” and you’re ready to go.

Time to see if it works...

- Use the script runOneProcWGUI
  - searches for current component and any that are one dir level above current.
  - An identical example is in ./Framework/component/StarterComponent/ that the script will find.
- Creates a CcafeineRC file that initializes the framework with components (Ccaffeine standard and the examples here).



What you are able to do now that you couldn’t before …

- Run on parallel cluster or proprietary machine with CCA components that you didn’t write.
  - Steve Jobs: “the best software is software I didn’t have to write” –not that he actually ever did.
- Develop incrementally & interactively in serial and *parallel*.
  - Detach, go have lunch and reattach.
- After everything is working, dump the script and run it in batch mode.