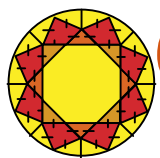
Exploring Applicability of CCA to TSI

Gary Kumfert, Tamara Dahlgren, and Thomas Epperly

Lawrence Livermore National Laboratory

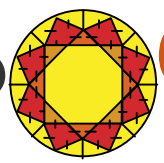
UCRL-PRES-203090

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.



CCA
Common Component Architecture

What Are

- Components
- The likely Costs and Benefits of Componentizing your code
- 
  CCA
  Common Component Architecture
- 

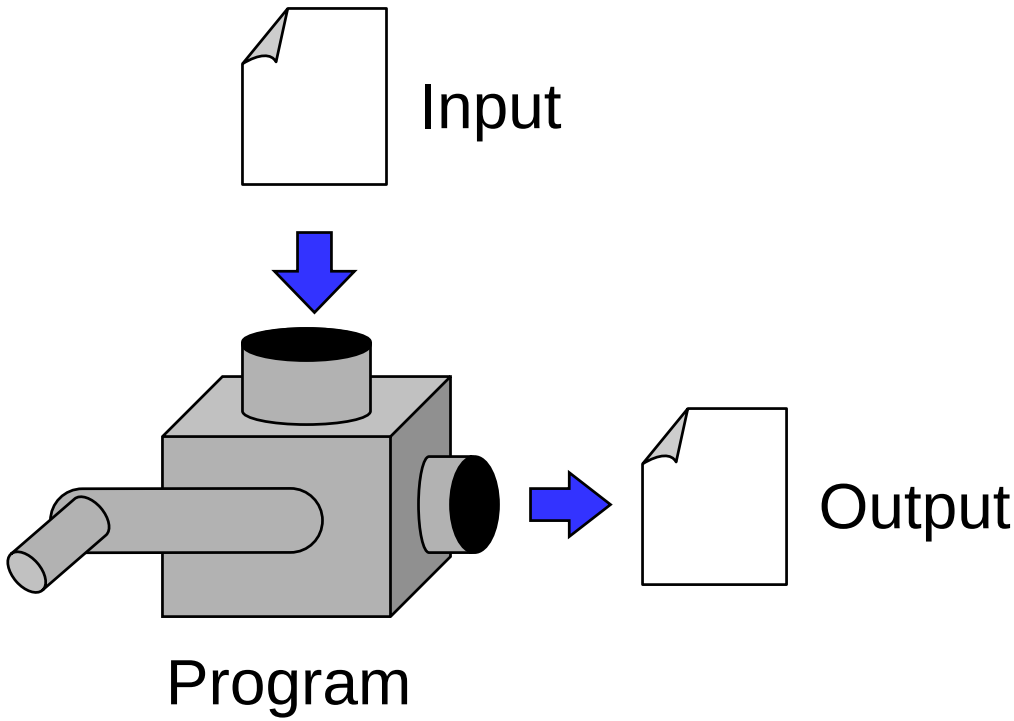
What Are Components?

- **Hard Question**
  - **Unintentionally Vague**
- **Component Technology is a Concept**
- **Easier questions:**
  - **What's a COM Component**
  - **What's a .NET Component**
  - **What's the difference between Components in CORBA and Enterprize Java Beans?**
  - **What's a CCA Component?**

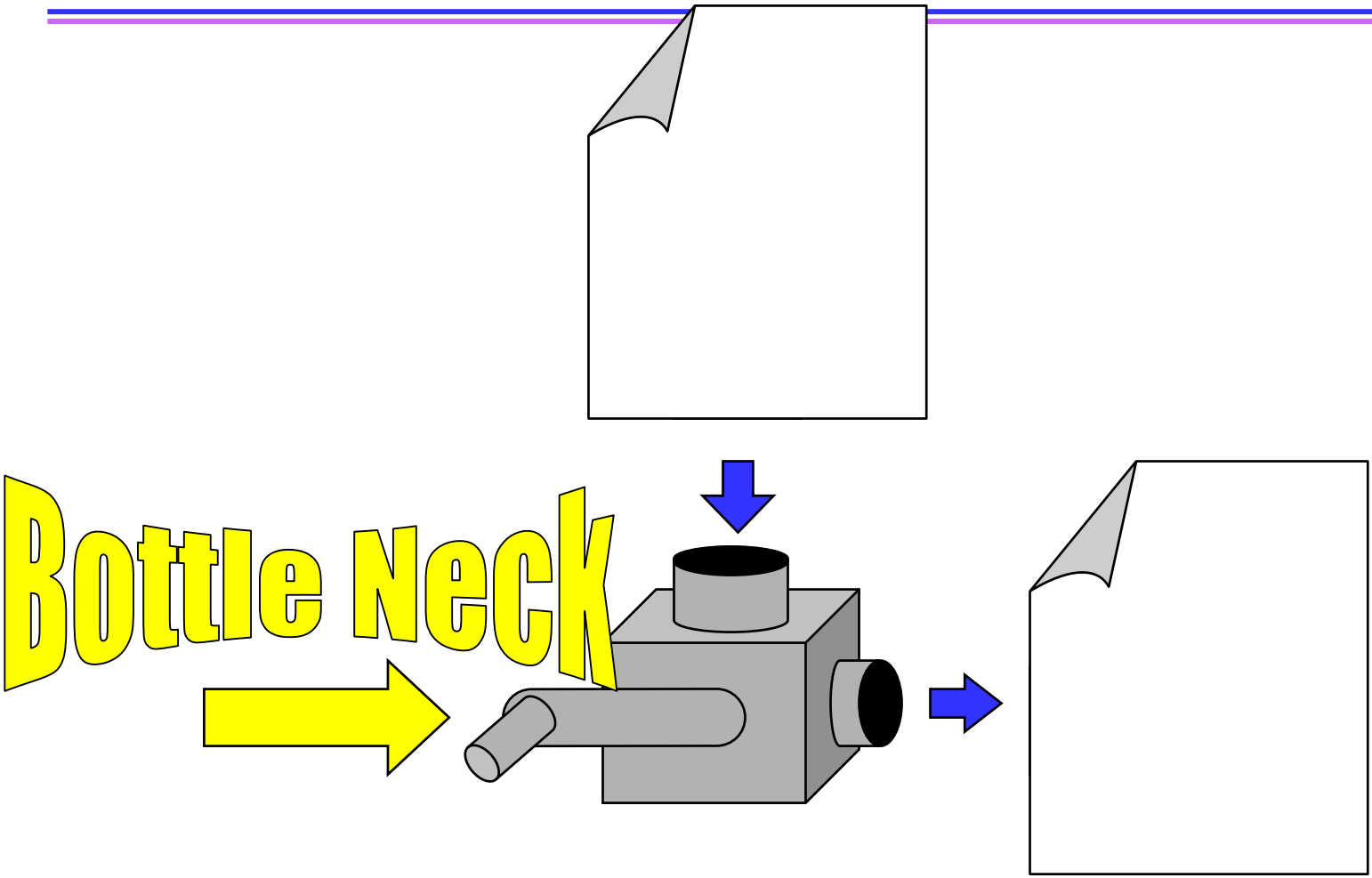
What Are Components?

- A Pictorial Introduction
- (aka Gary's Sausage Grinder Talk)
- There will be a quiz at the end!

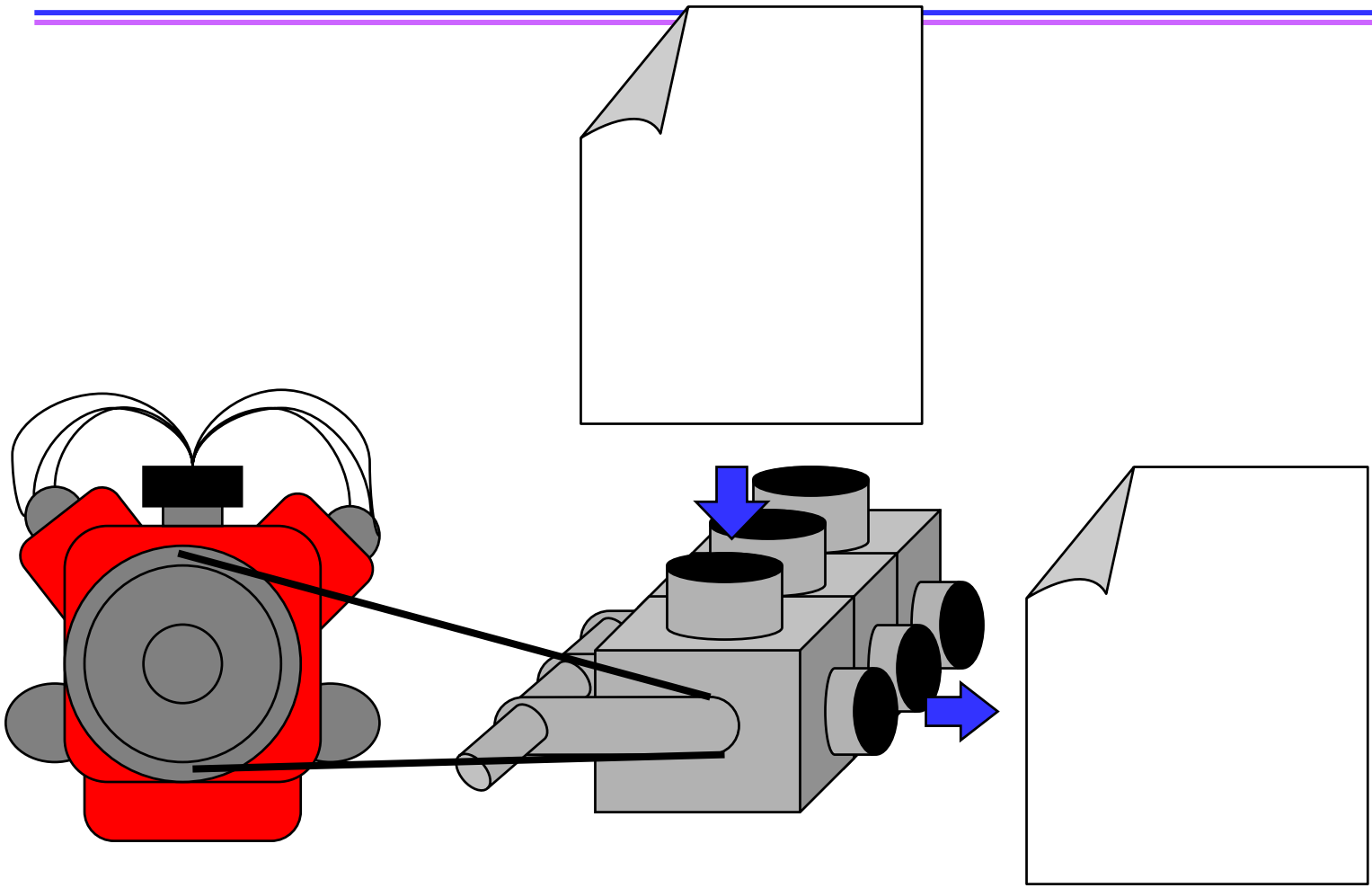
Once upon a time...



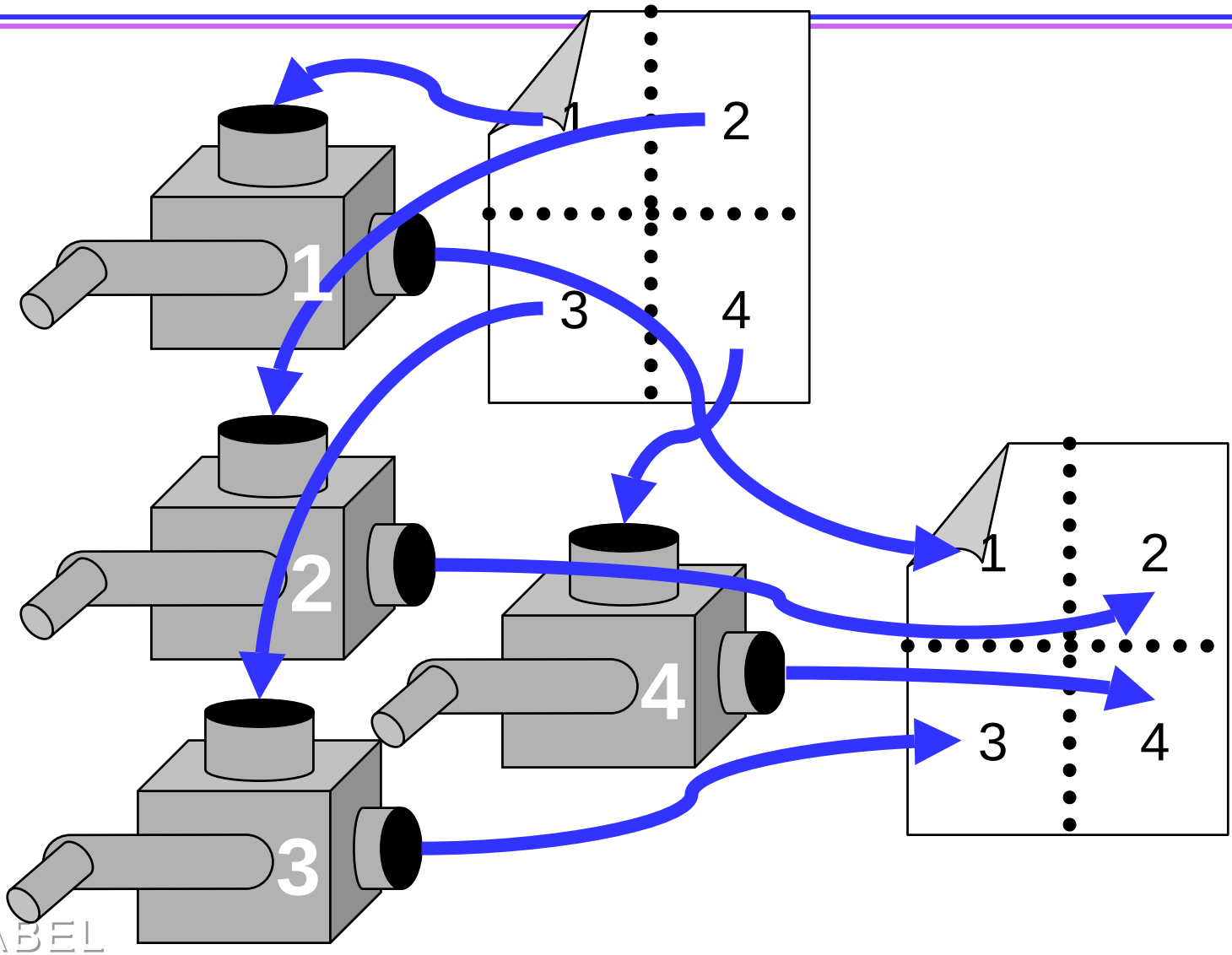
As Scientific Computing grew...



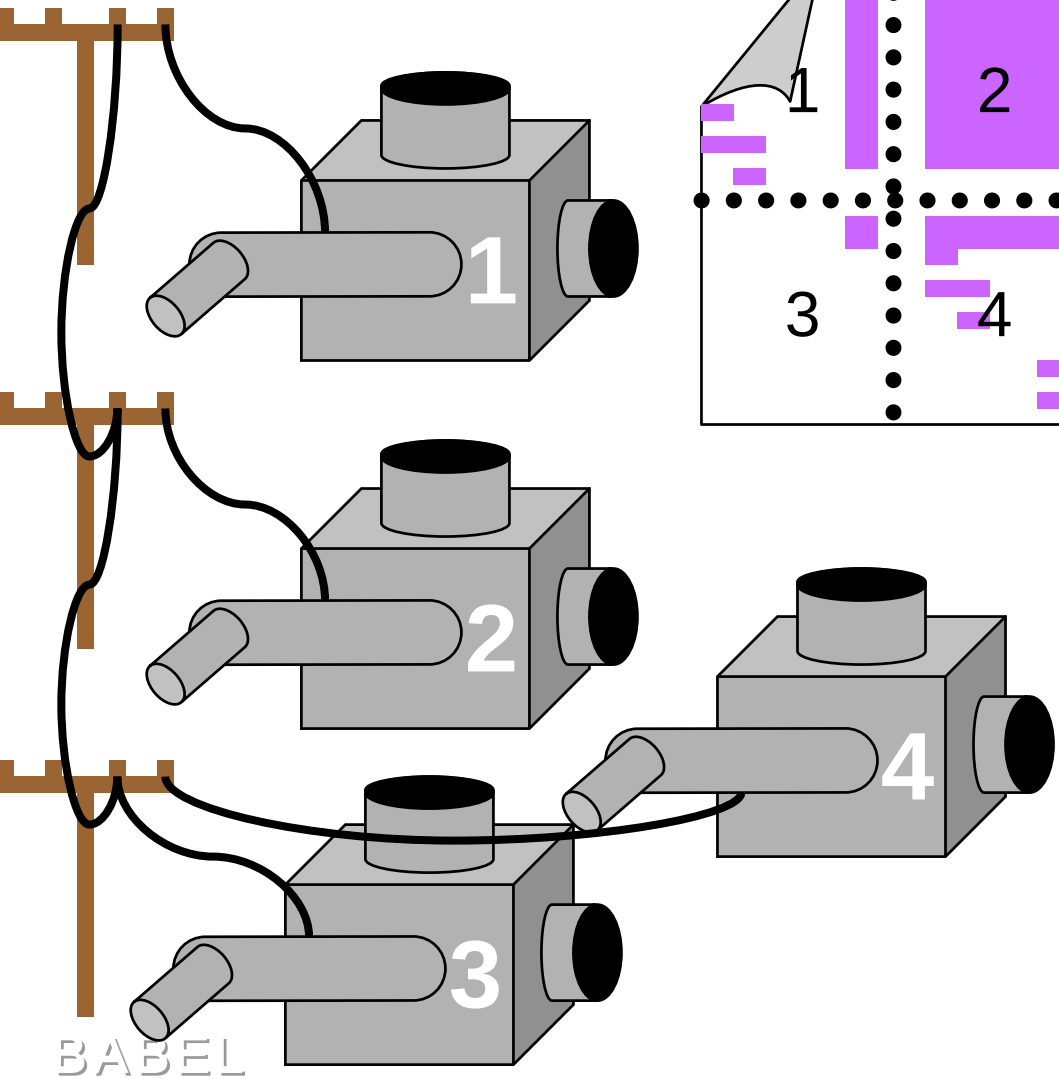
Tried to ease the bottle neck



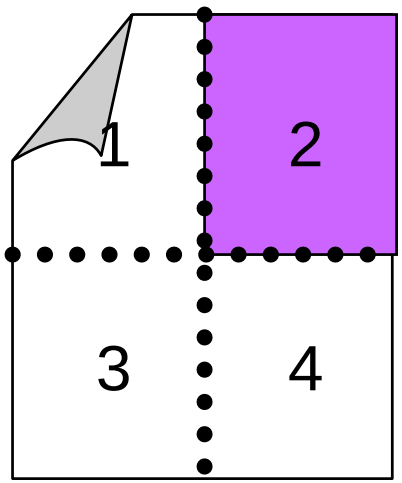
SPMD was born.



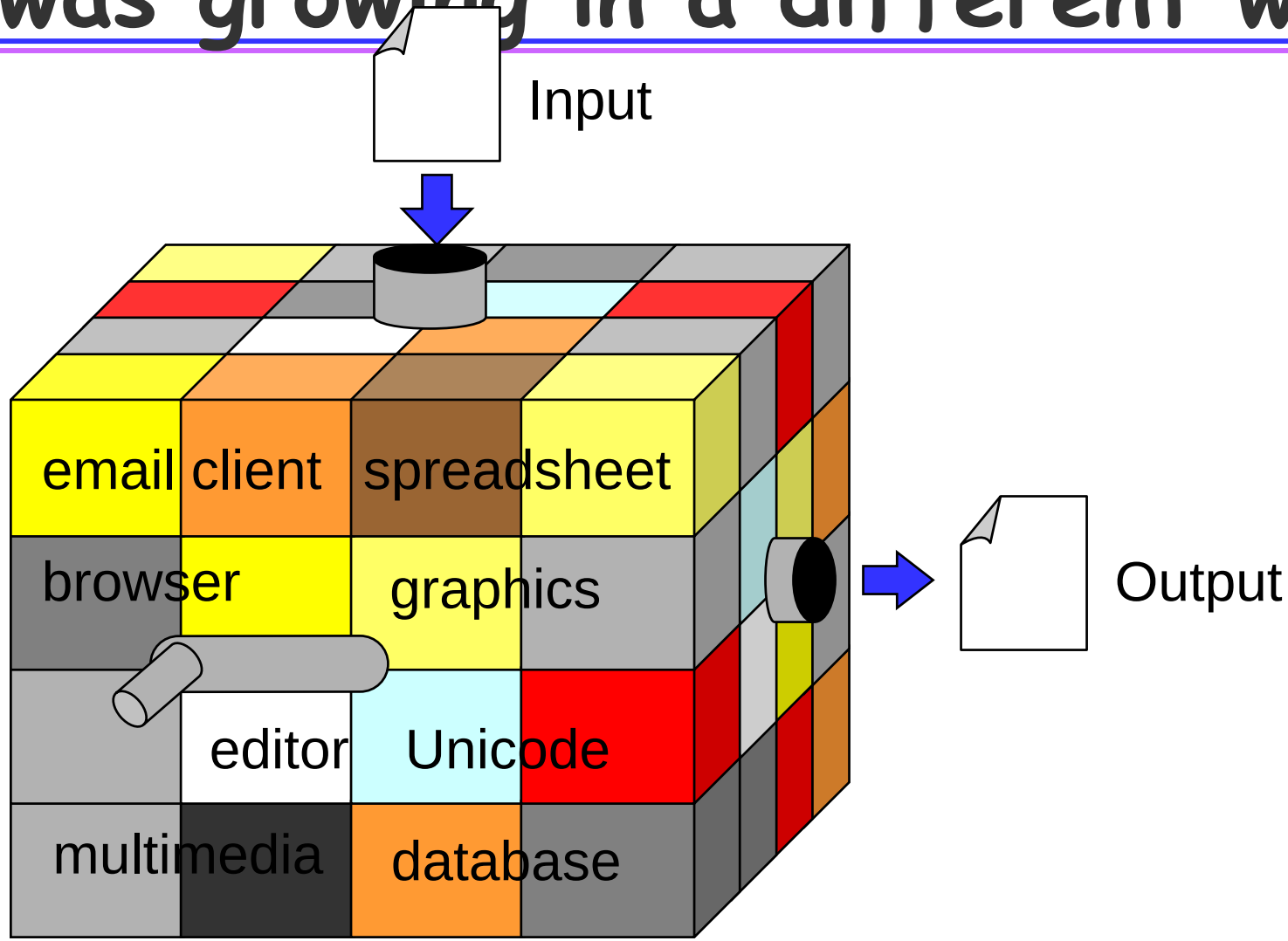
SPMD worked



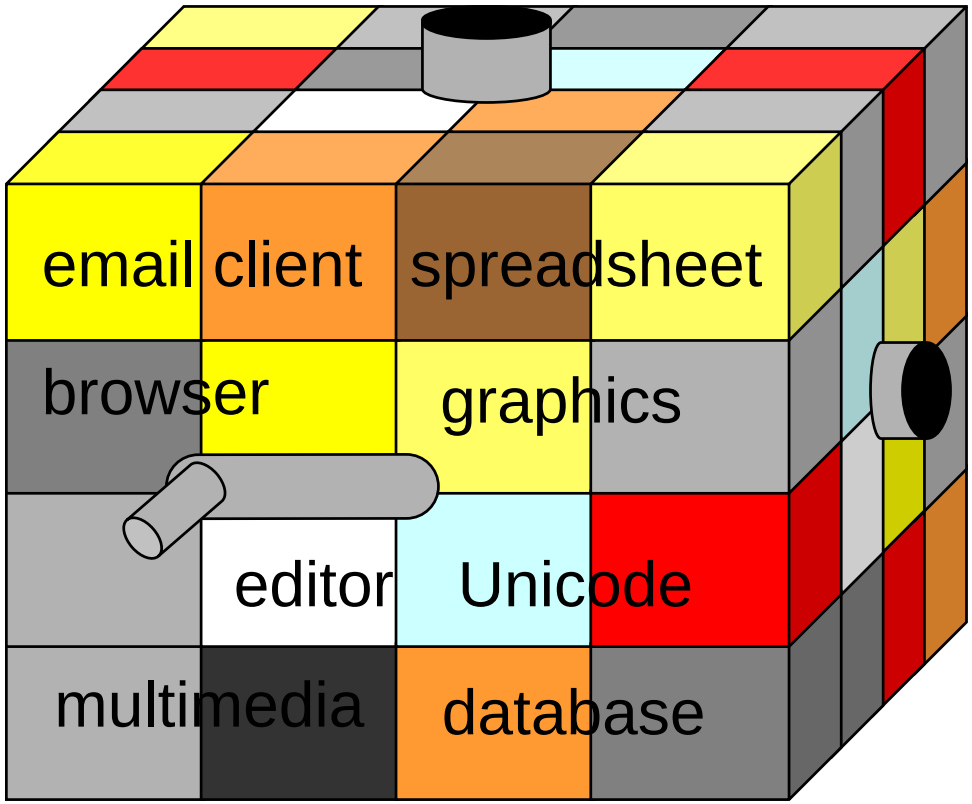
But it isn't easy!!!



Meanwhile, corporate computing was growing in a different way

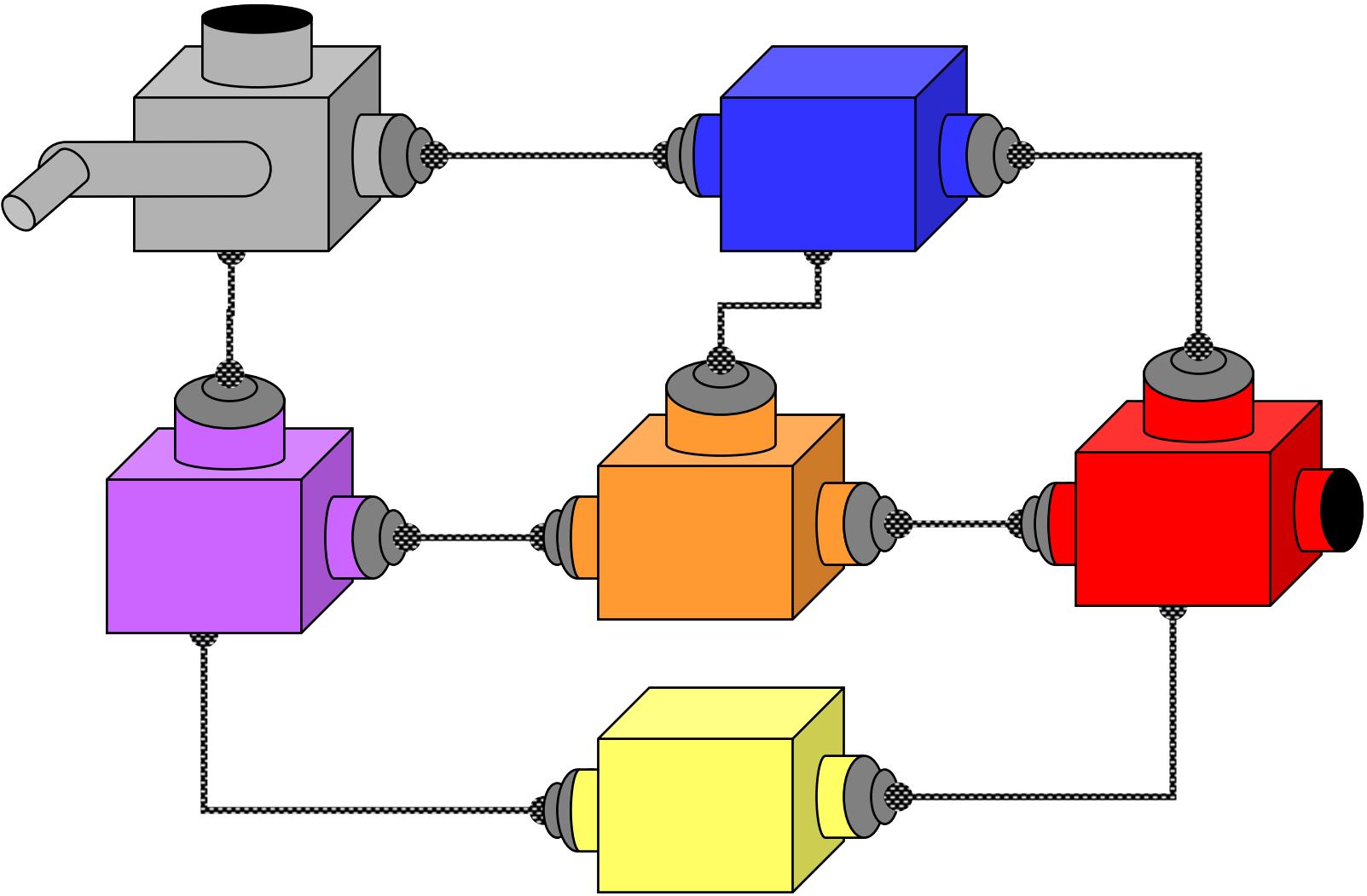


This created a whole new set of problems...

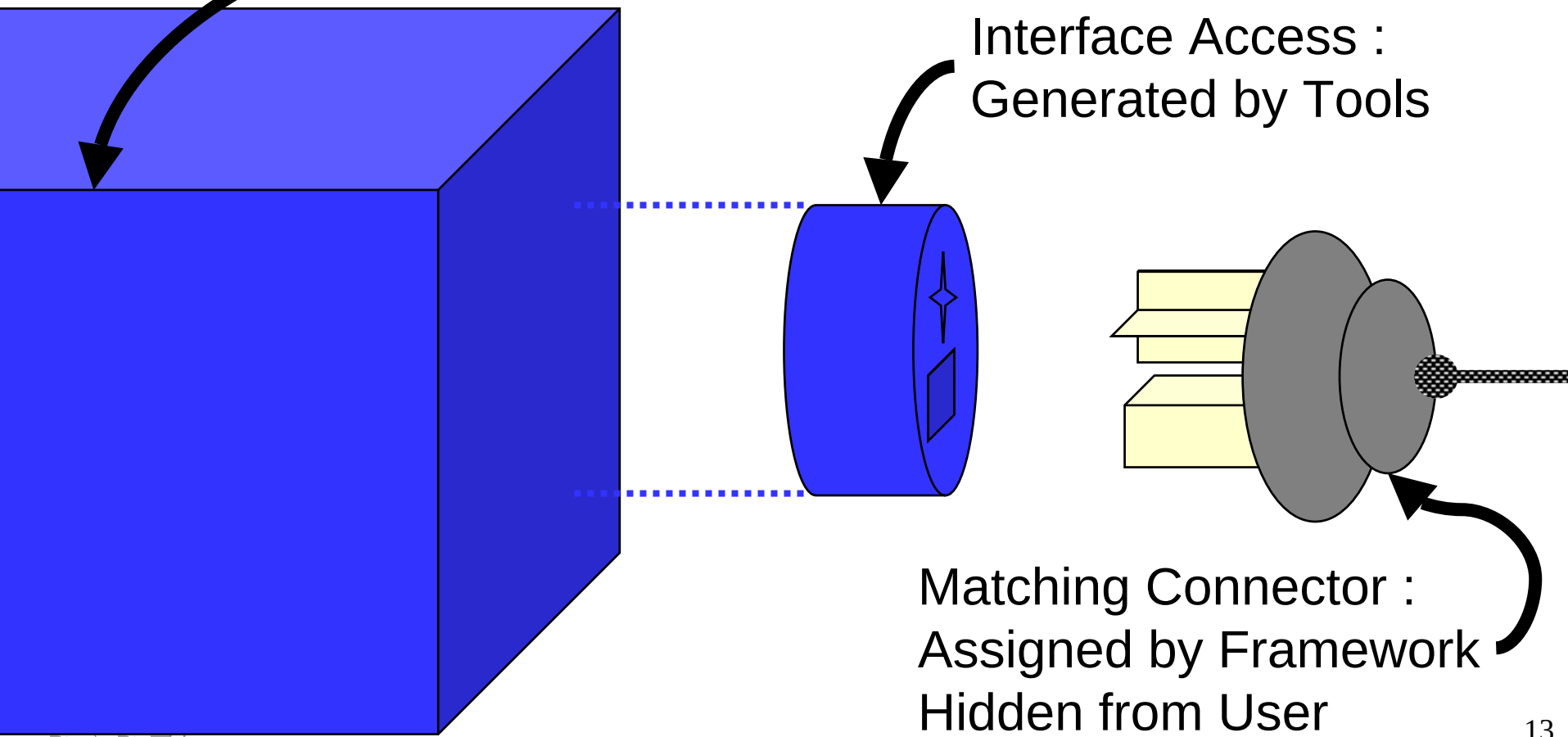


- **Interoperability across multiple languages**
- **Interoperability across multiple platforms**
- **Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)**

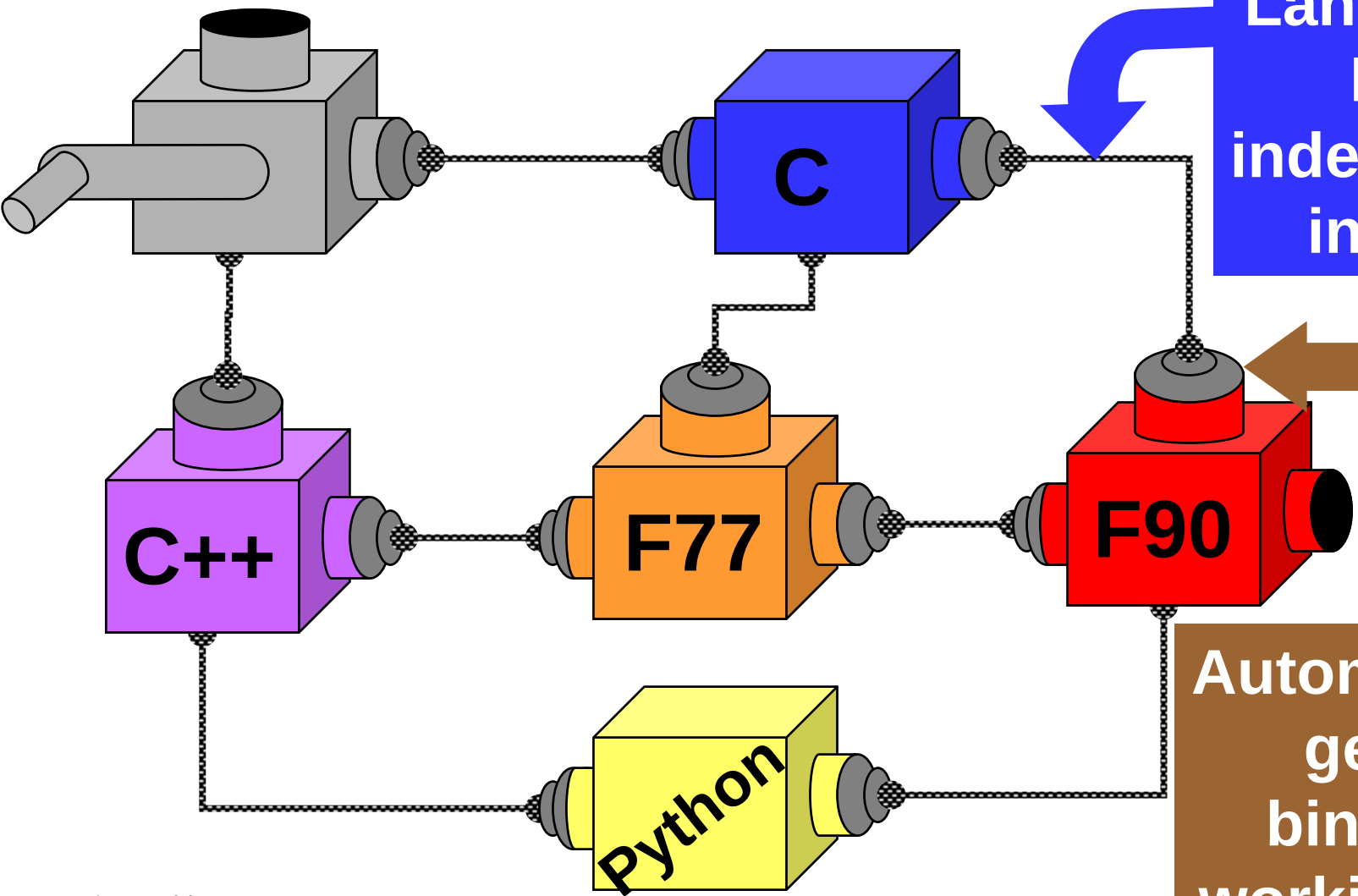
Component Technology addresses these problems



So what's a component ???



1. Interoperability across multiple languages

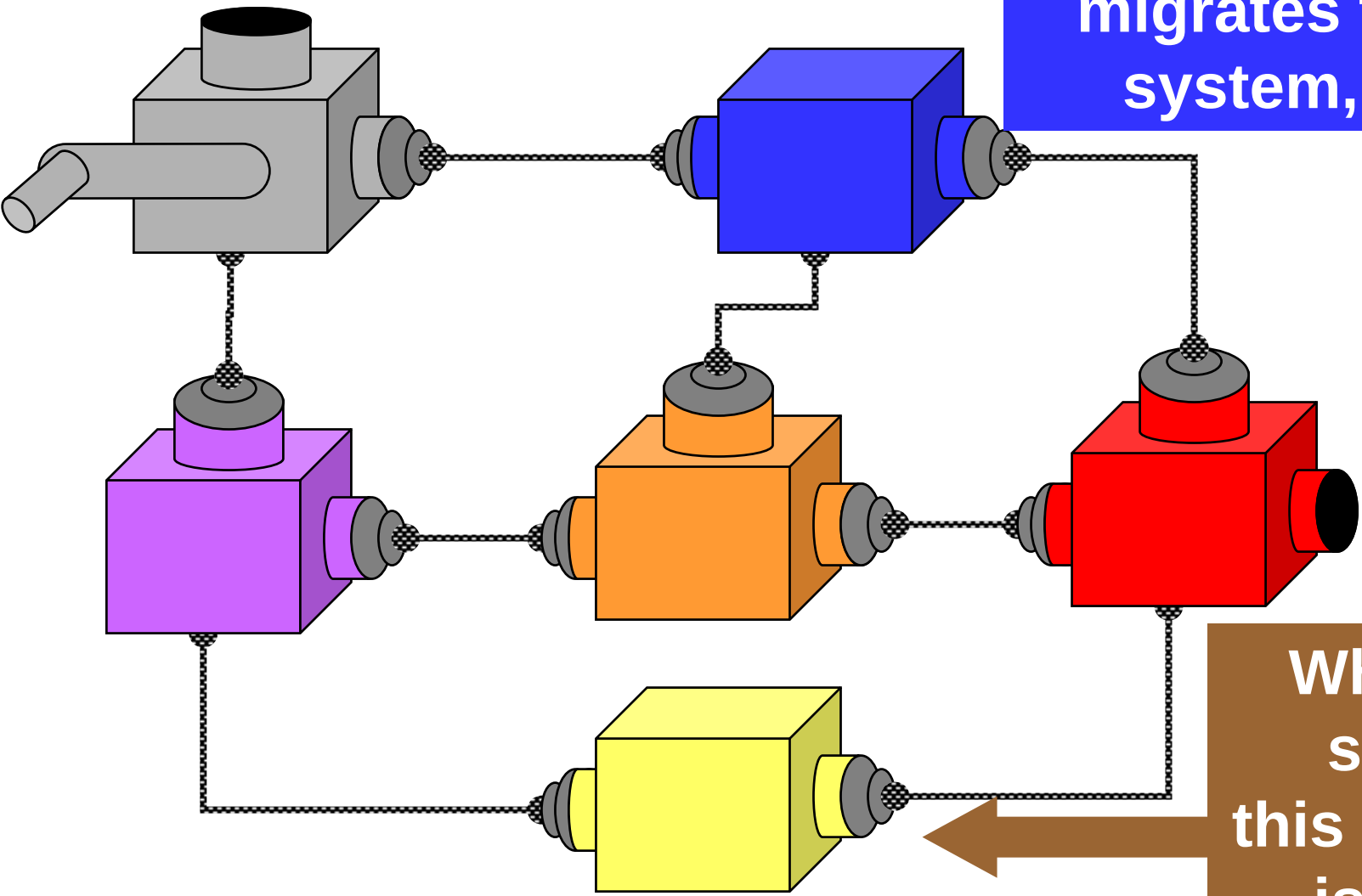


Language & Platform independent interfaces

Automatically generated bindings to working code

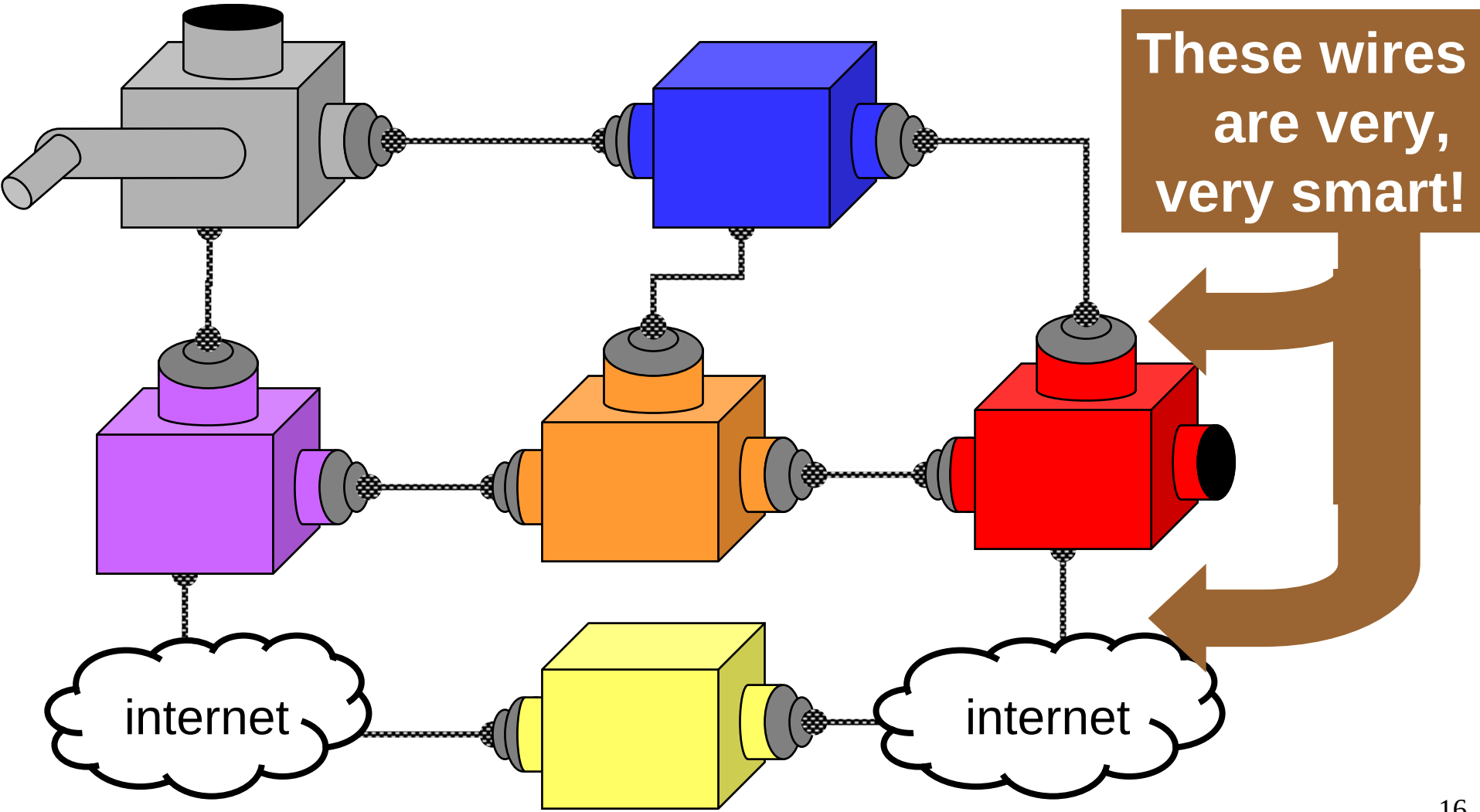
2. Interoperability Across Multiple Platforms

Imagine a company migrates to a new system, OS, etc.

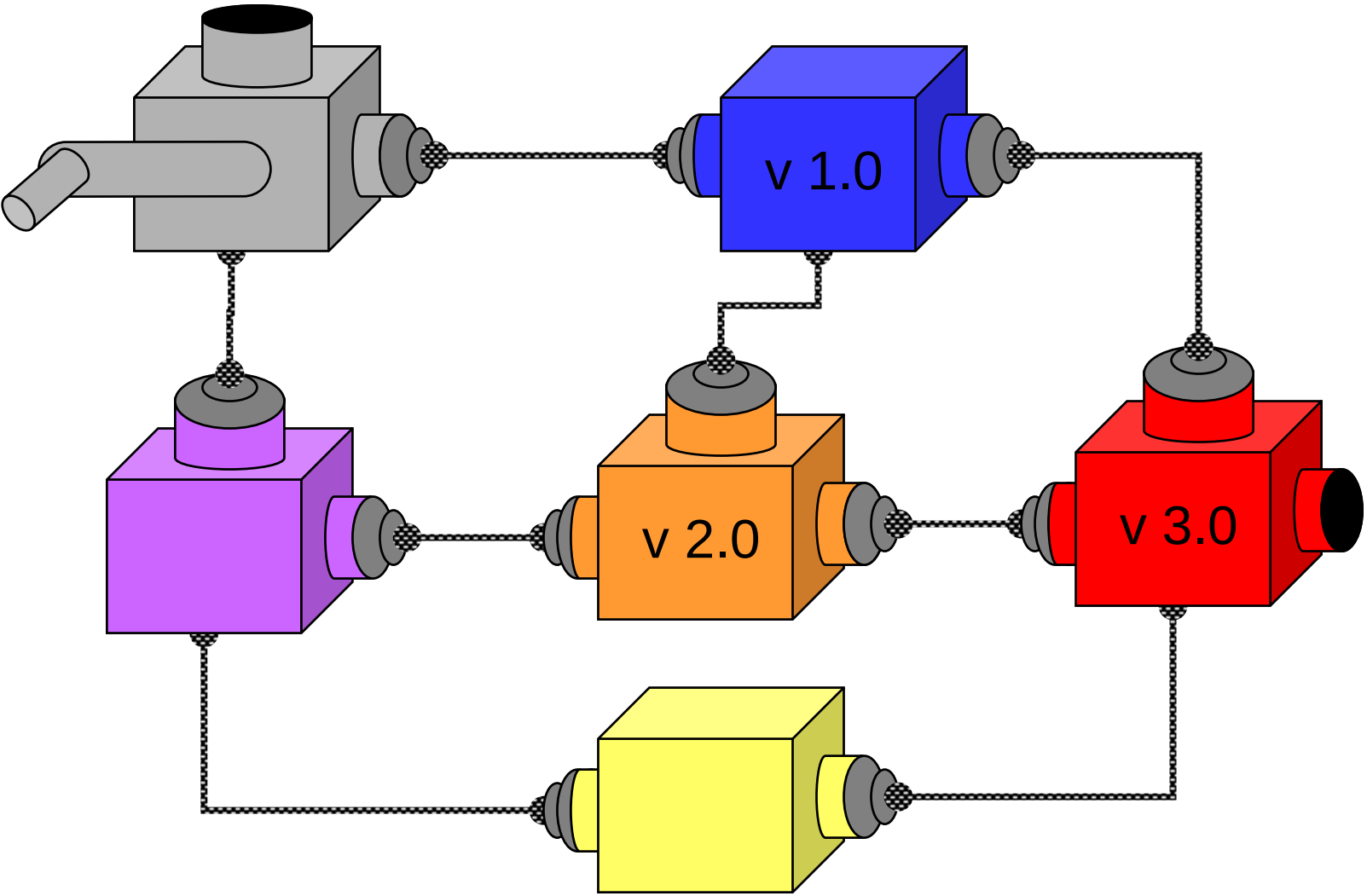


What if the source to this one part is lost???

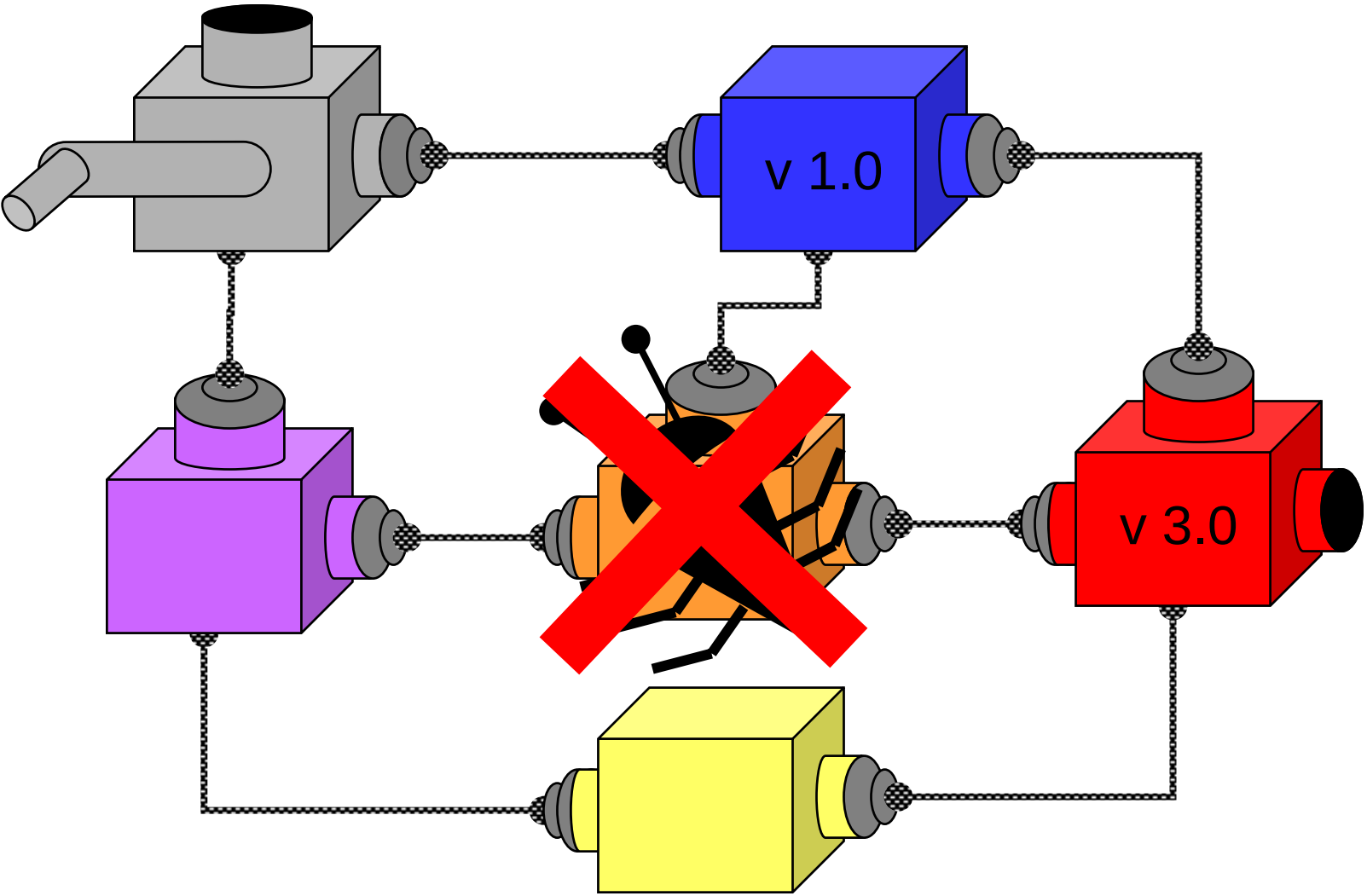
Transparent Distributed Computing



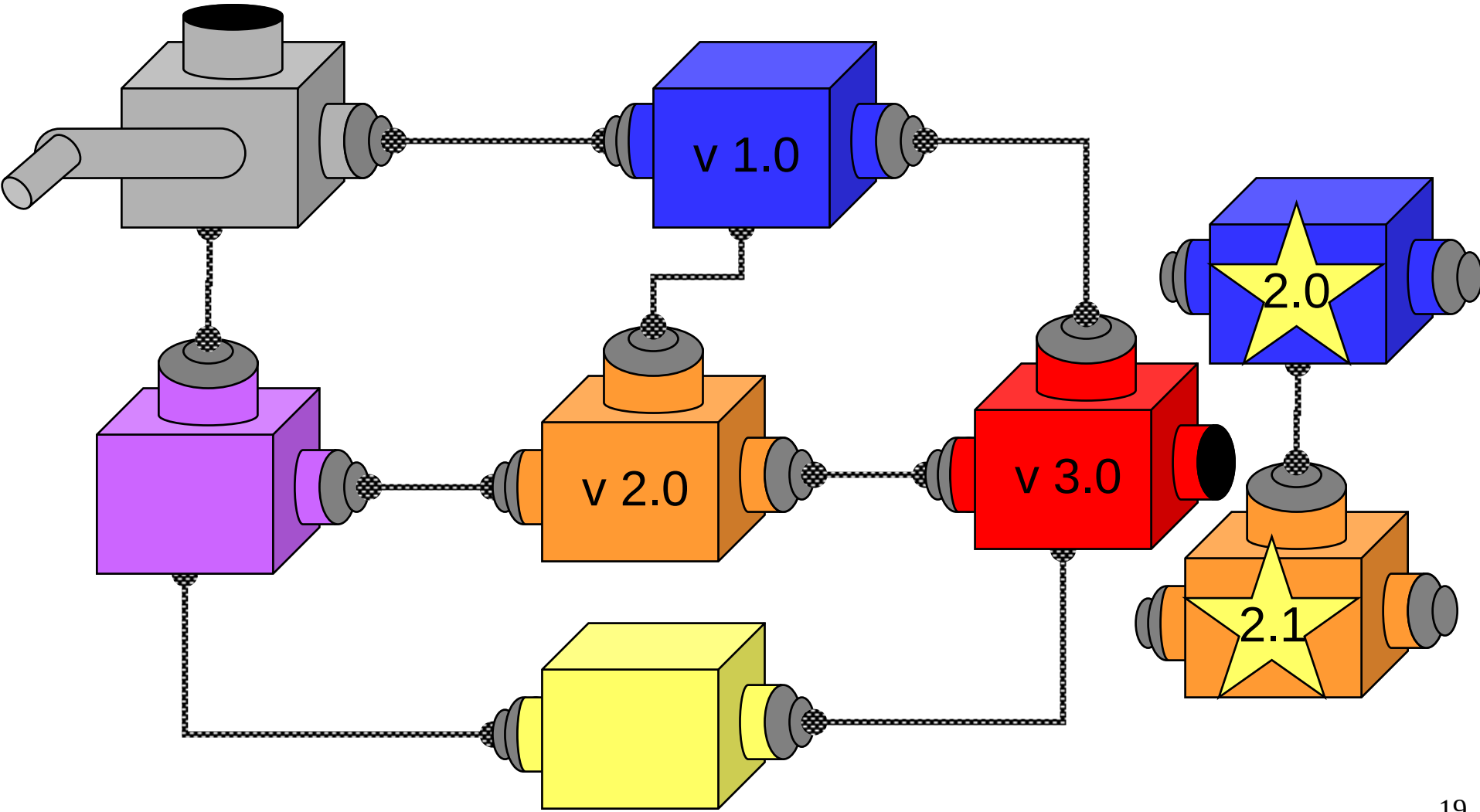
3. Incremental Evolution With Multiple 3rd party software



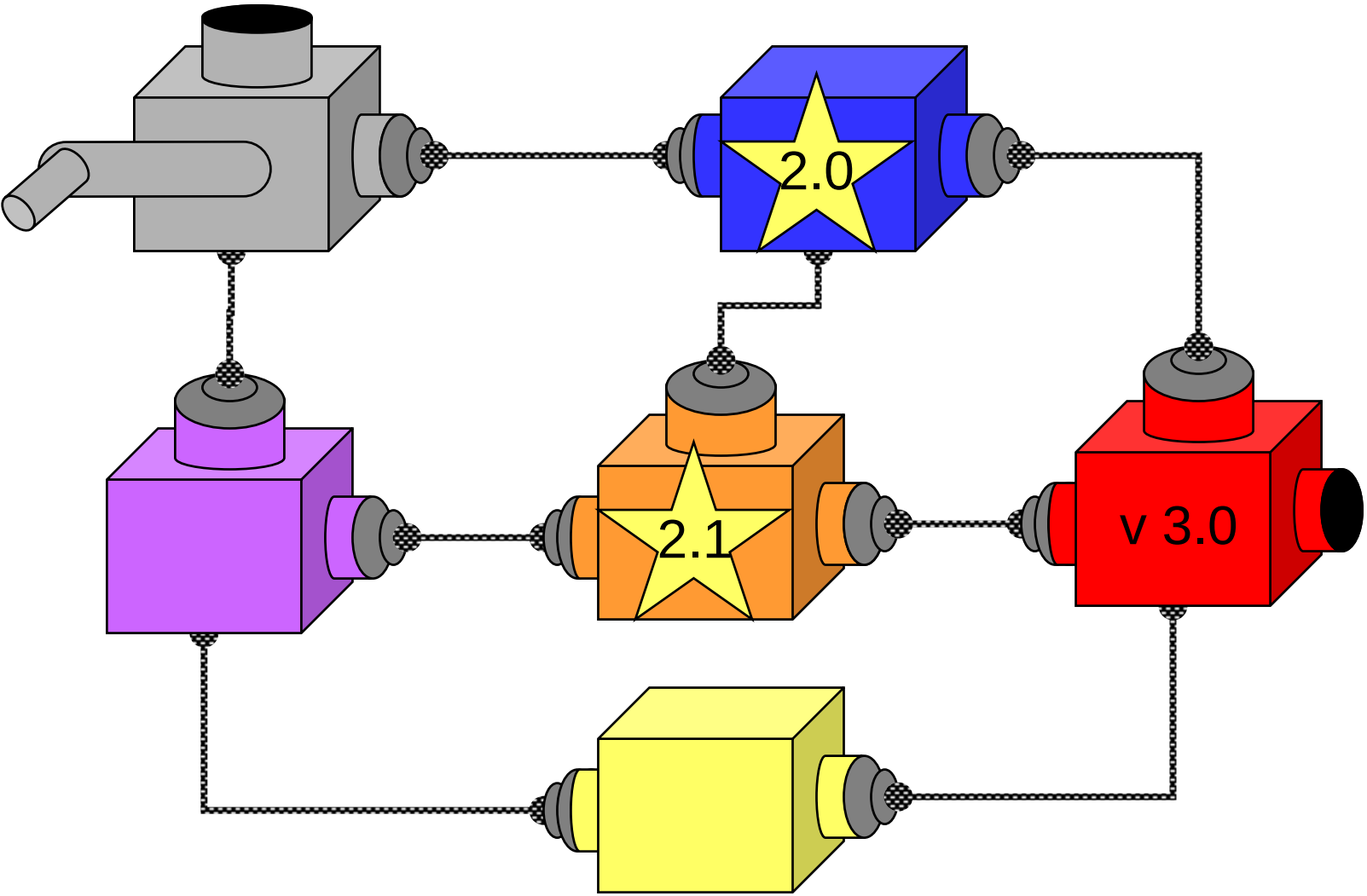
Now suppose you find this bug...



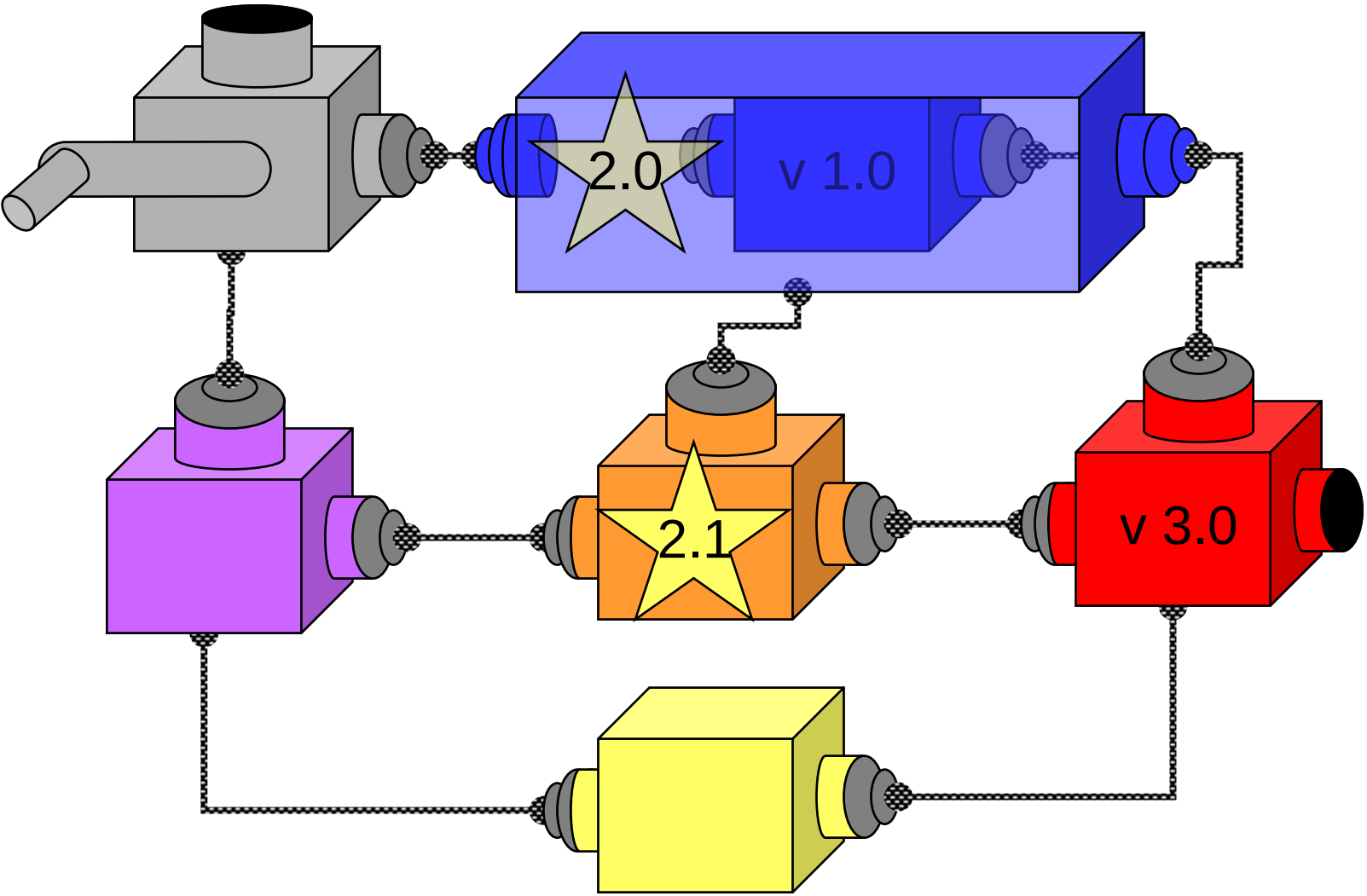
Good news: an upgrade available
Bad news: there's a dependency



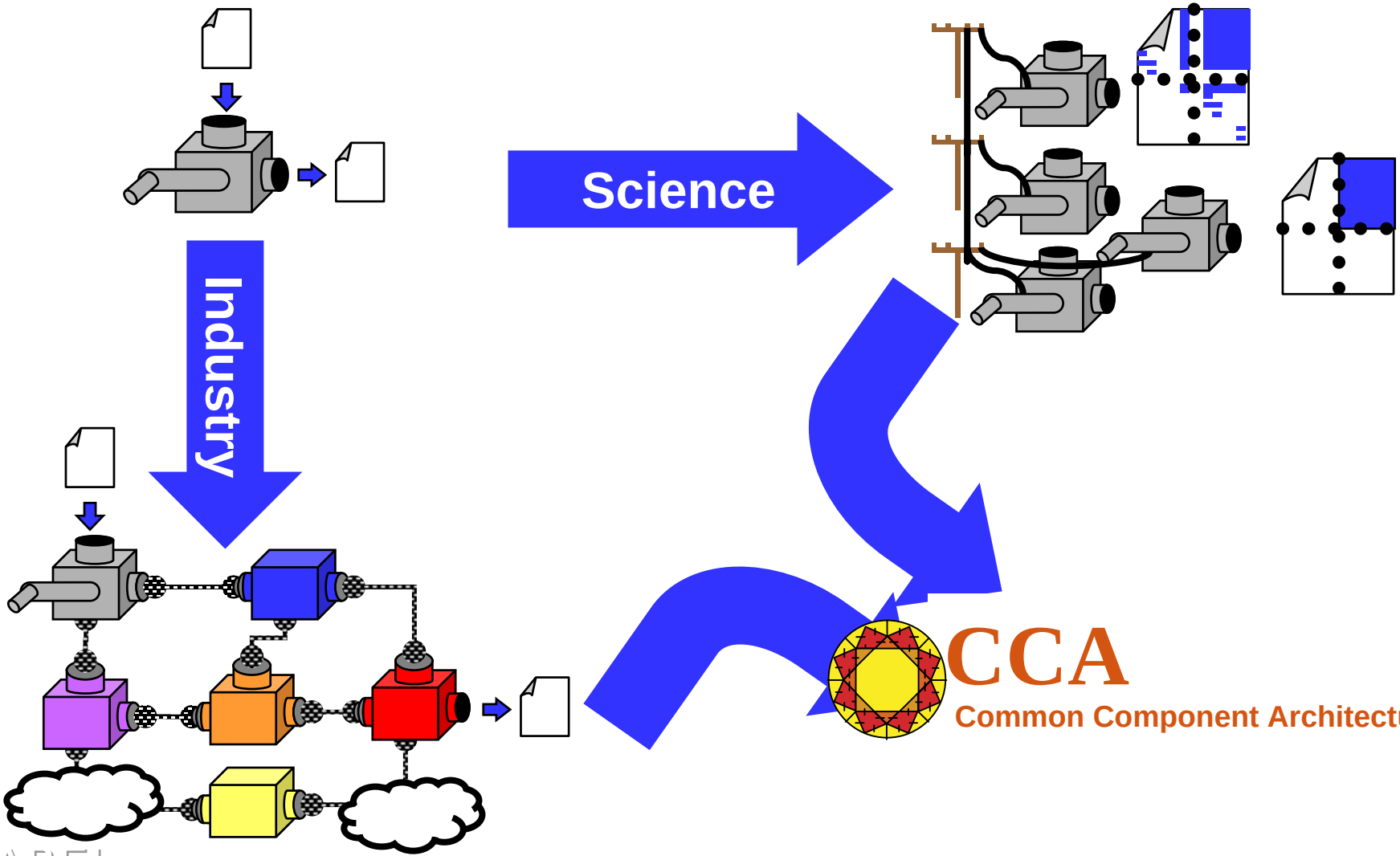
Great News: Solvable with Components



Great News: Solvable with Components

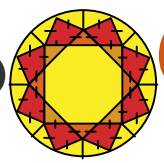


The Model for Scientific Component Programming

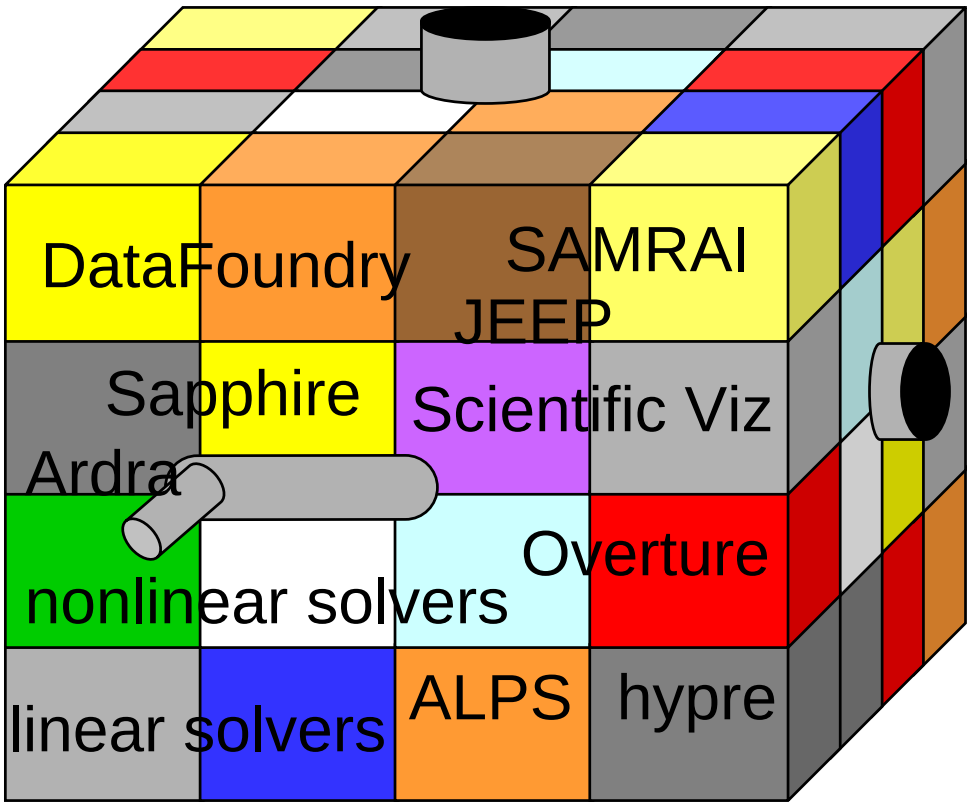


What Are

We are here

- Components
- The likely Costs and Benefits of Componentizing your code
- 
  CCA
  Common Component Architecture
- 

Why Components for Scientific Computing?



- **Interoperability across multiple languages**
- **Interoperability across multiple platforms**
- **Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)**

Why Components for Scientific Computing?

"Change-Oriented Software"

- integration of small systems to large ones
- amenability to change
- manage correctness in the face of change

- **Interoperability across multiple languages**
- **Interoperability across multiple platforms**
- **Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)**

When componentization might make sense for you

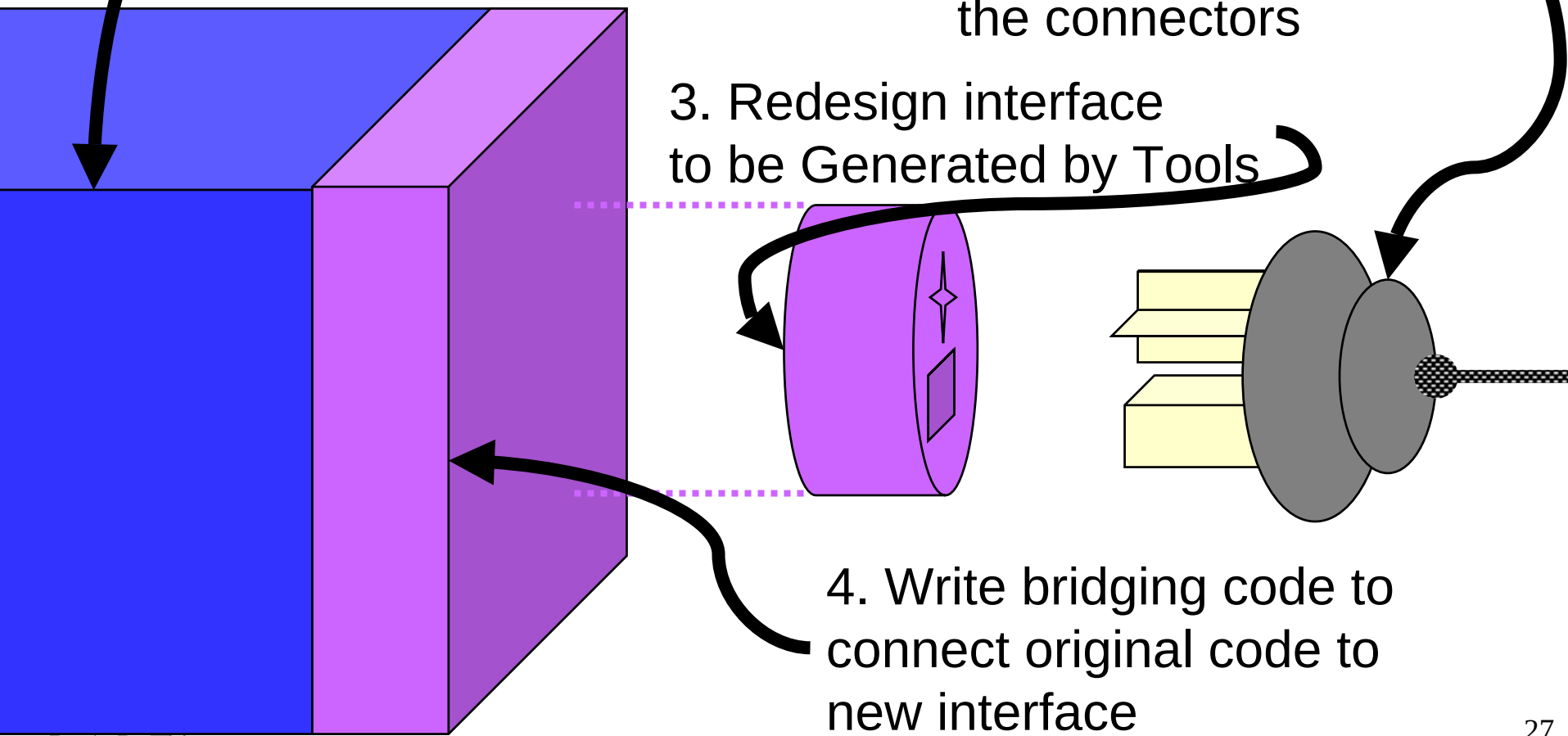
- Componentization is not automatic
- Makes sense if:
  - You develop a library for wide-spread use
  - You mix your code with lots of others
  - You maintain a large code that will evolve with your scientific pursuits
- Doesn't make sense for
  - Disposable, one-off codes
  - Software that is standalone & fixed (not incl bugs)

What happens with componentization?

1. Original Implementation

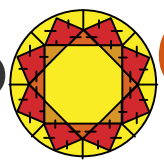
2. Learn the technology to properly design for the connectors

3. Redesign interface to be Generated by Tools



4. Write bridging code to connect original code to new interface

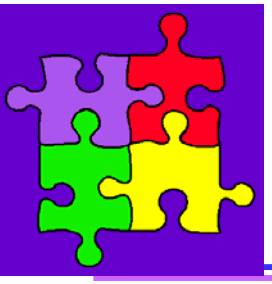
What Are

- Components
- The likely Costs and Benefits of Componentizing your code
- 
  CCA
  Common Component Architecture
- BABEL

We are here

What is the CCA?

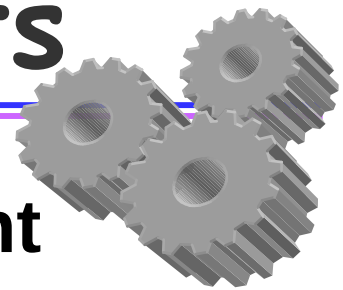
- Common Component Architecture
  - Is a “research” standard
- CCA Forum
  - The grass-roots body
  - Voting membership: requires attendance at 2 out of the last three quarterly meetings.
- CCTTSS is “official” name for the SciDAC ISIC.
  - Rob Armstrong, Sandia, PI



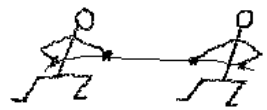
CCTTSS Research Thrust Areas and Main Working Groups

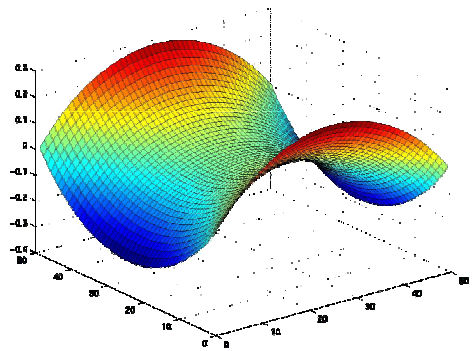
- **Scientific Components**
  - **Scientific Data Objects**
  - **Lois Curfman McInnes, ANL (curfman@mcs.anl.gov)**
- **"MxN" Parallel Data Redistribution**
  - **Jim Kohl, ORNL (kohlja@ornl.gov)**
- **Frameworks**
  - **Language Interoperability / Babel / SIDL**
  - **Component Deployment / Repository**
  - **Gary Kumfert, LLNL (kumfert@llnl.gov)**
- **User Outreach**
  - **David Bernholdt, ORNL (bernholdtde@ornl.gov)**

Scientific Components

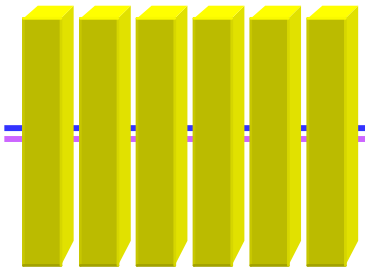


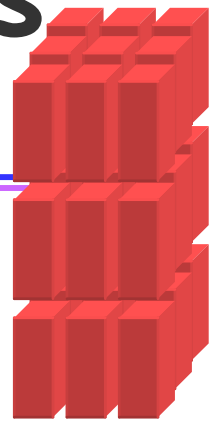
- **Abstract Interfaces and Component Implementations**
  - **Mesh management**
  - **Linear, nonlinear, and optimization solvers**
  - **Multi-threading and load redistribution**
  - **Visualization and computational steering**
- **Quality of Service Research**
- **Fault Tolerance**
  - **Components and Frameworks**



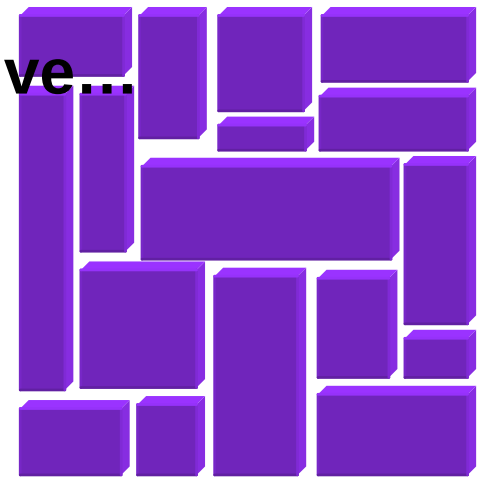


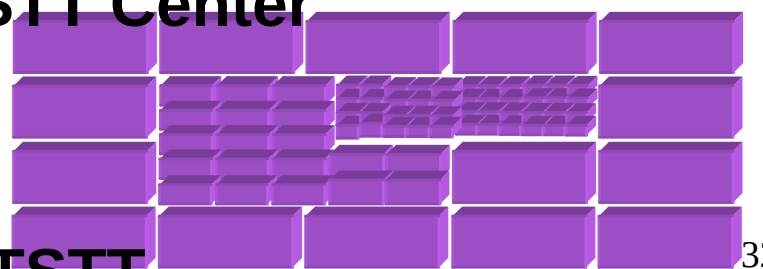
Scientific Data Objects & Interfaces



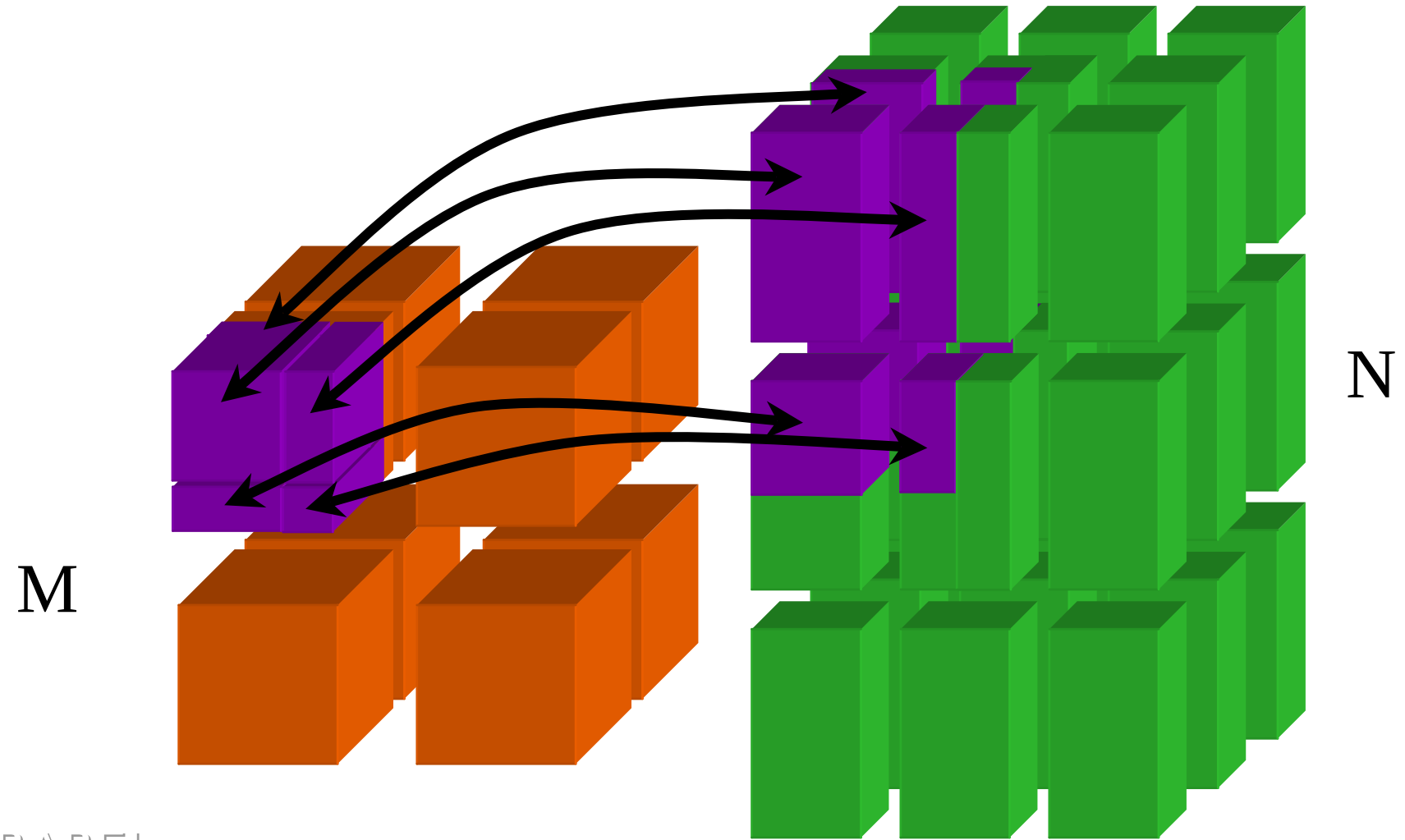


- **Define "Standard" Interfaces for HPC Scientific Data**
  - **Descriptive, Not (Necessarily) Generative…**
- **Basic Scientific Data Object**
  - **David Bernholdt, ORNL**
- **Structured & Unstructured Mesh**
  - **Lori Freitag, ANL**
  - **Collaboration with SciDAC TSTT Center**
- **Block Structured AMR**
  - **Phil Colella, LBNL**
  - **Collaboration with APDEC & TSTT**





"MxN" Parallel Data Redistribution: The Problem…



"MxN" Parallel Data Redistribution: The Problem…

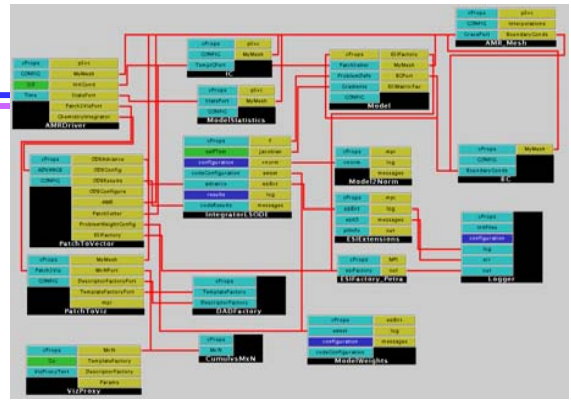
- Create complex scientific simulations by coupling together multiple parallel component models
  - Share data on "M" processors with data on "N"
    - M != N ~ Distinct Resources (Pronounced "M by N")
  - Model coupling, e.g., climate, solver / optimizer
  - Collecting data for visualization
    - Mx1; increasingly MxN (parallel rendering clusters)
- Define "standard" interface
  - Fundamental operations for any parallel data coupler
    - Full range of synchronization and communication options

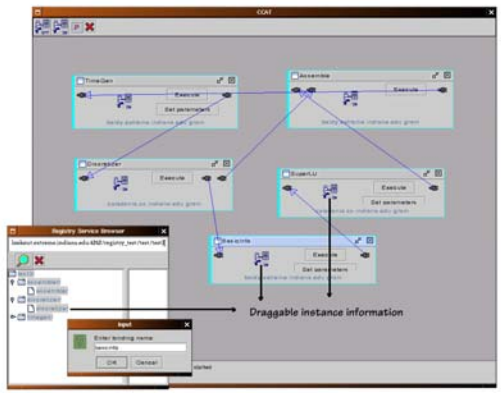
CCA Frameworks

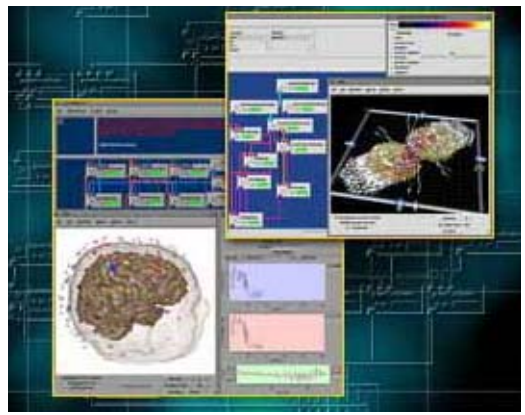
- **Component Containers & Run-Time Environments**
- **Research Areas:**
  - **Integration of prototype frameworks**
    - **SCMD/parallel with distributed, bridged for one application**
    - **Unify framework services & interactions…**
  - **Language interoperability tools**
    - **Babel/SIDL, incorporate difficult languages (F90…)**
    - **Production-scale requirement for application areas**
  - **Component deployment**
    - **Component repository, interface lookup & semantics**

CCA Frameworks

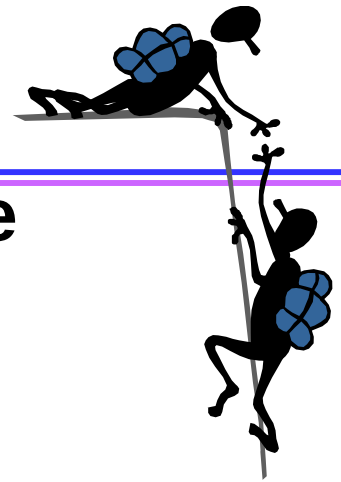
- Ccaffeine
  - SPMD/SCMD parallel
  - Direct connectio
- CCAT / XCAT
  - Distributed
  - Network connect
- SCIRun
  - Parallel, multithreaded
  - Direct connection
- Decaf, DCS, Dune, Uintah, LegionCCA







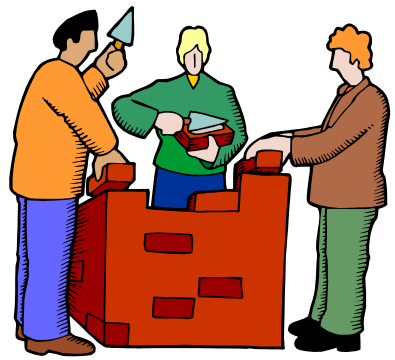
Outreach and Applications Integration



- Tools Not Just "Thrown Over The Fence"…
- Several Outreach Efforts:
  - General education and awareness
    - Tutorials, like this one!
    - Papers, conference presentations
  - Strong liaison with adopting groups
    - Beyond superficial exchanges
    - Real production requirements & feedback
  - Chemistry and climate work within CCTTSS
    - Actual application development work ($$$)
- SciDAC Emphasis
  - More vital *applied* advanced computing research!

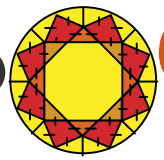
Active CCA Forum Working Groups

- Adaptive Mesh Refinement
- Generalized Data Objects
- Tutorial Presentations
- Application Domain Groups:
  - Climate, Chemistry
- MxN Data Redistribution
- Embeddable Scripting
- Fortran Users
- Babel Development & Users
- Deployment / XML Schemas
- Ccaffeine Open Framework
- Component-Based Debugging…



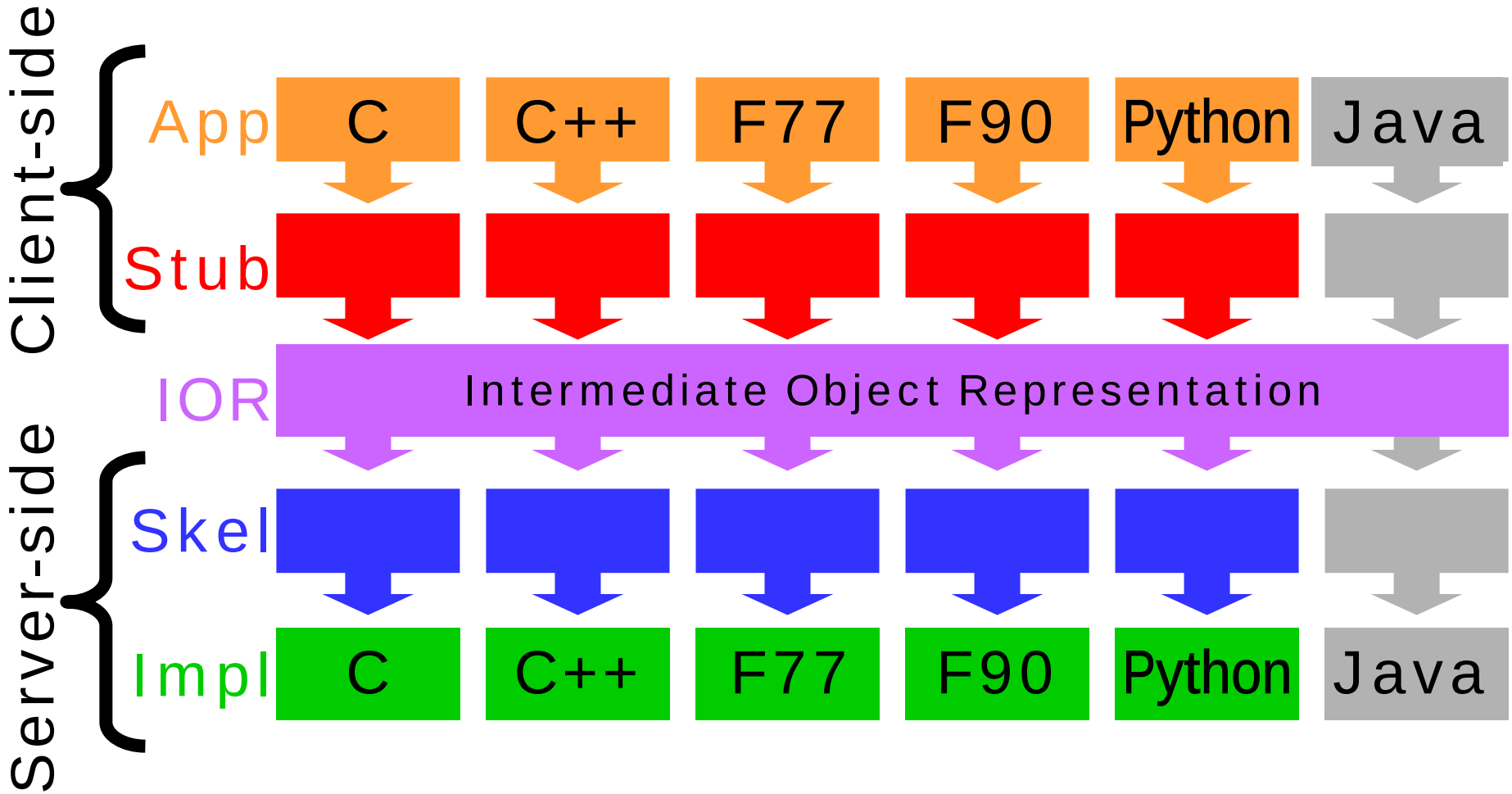
See http://www.cca-forum.org/working_groups.html for more info.

What Are

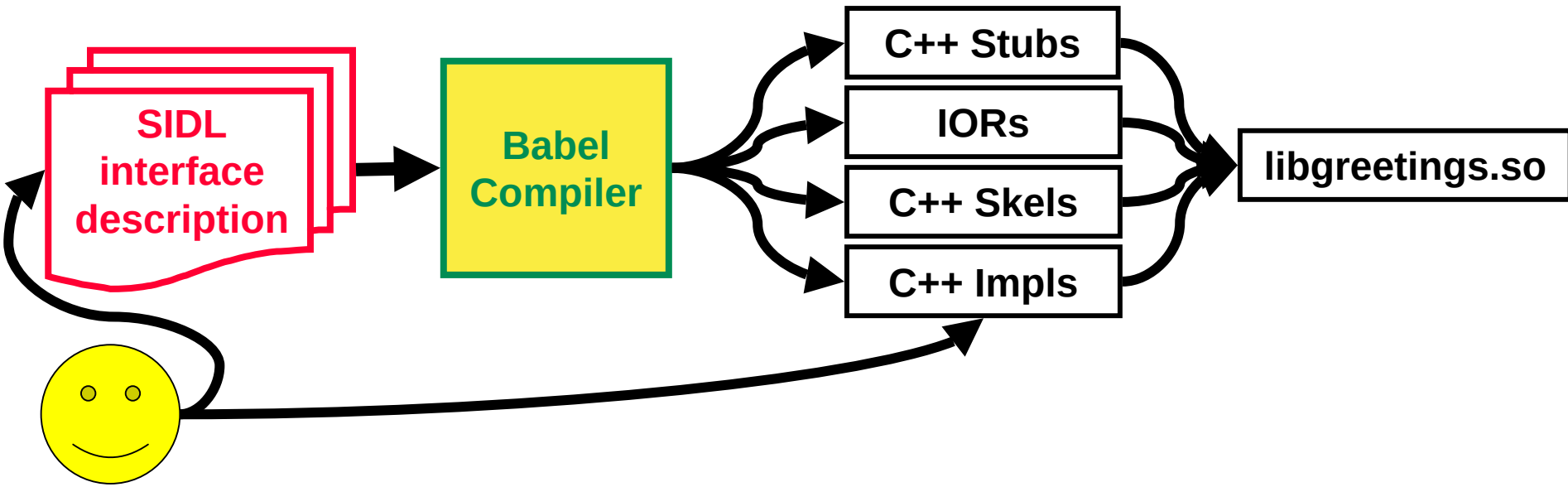
- Components
- The likely Costs and Benefits of Componentizing your code
- 
  CCA
  Common Component Architecture
- 

We are here

Babel Architecture



Library Developer Does This...



1. Write SIDL File
2. `babel --server=C++ greetings.sidl`
3. Add implementation details
4. Compile & Link into Library/DLL

greetings.sidl: A Sample SIDL File

```
package greetings version 1.0 {
    interface Hello {
        void setName( in string name );
        string sayIt ( );
    }
    class English implements-all Hello {  }
}
```

Adding the Implementation

```
namespace greetings {
class English_impl {
  private:
    // DO-NOT-DELETE splicer.begin(greetings.English._impl)
    ::std::string d_name;
    // DO-NOT-DELETE splicer.end(greetings.English._impl)
```

```
::std::string
greetings::English_impl::sayIt()
throw ()
{
  // DO-NOT-DELETE splicer.begin(greetings.English.sayIt)
  ::std::string msg("Hello ");
  return msg + d_name + "!";
  // DO-NOT-DELETE splicer.end(greetings.English.sayIt)
}
```

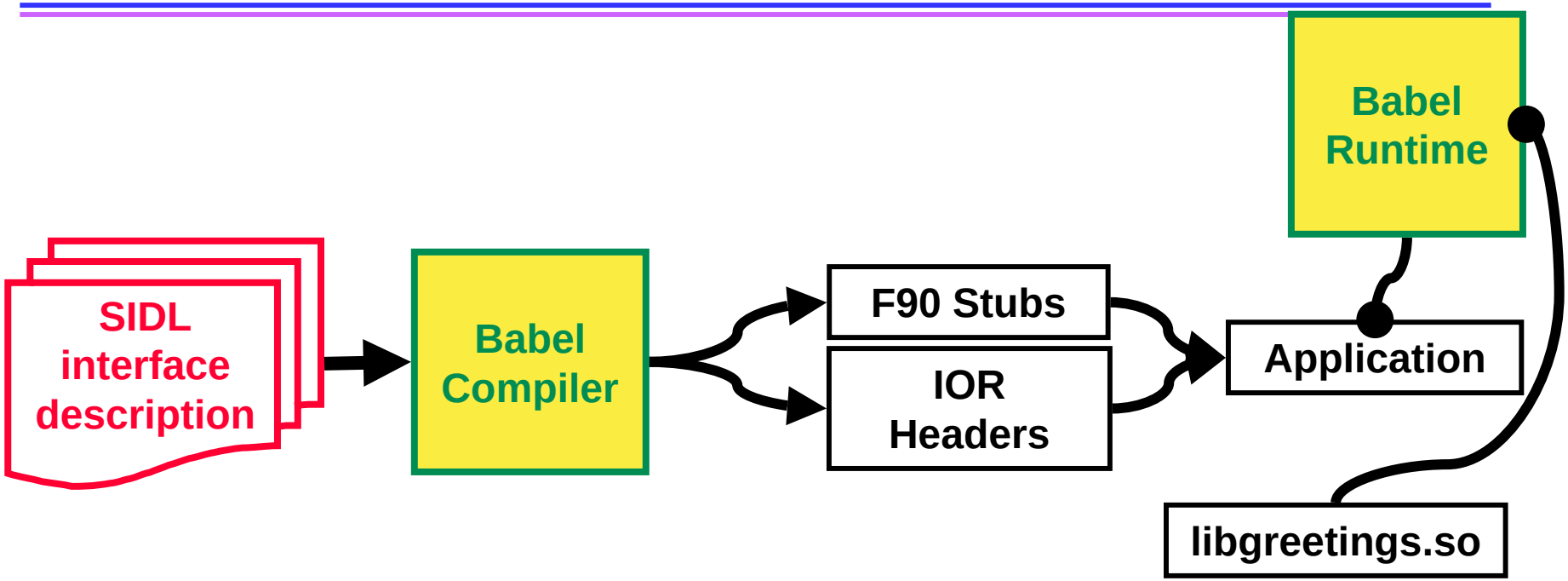
Adding th

```
package greetings version 1.0 {
    interface Hello {
        void setName( in string name );
        string sayIt ( );
    }
    class English implements-all Hello {  }
}
```

```
namespace greetings {
class English_impl {
  private:
    // DO-NOT-DELETE splicer.begin(greetings.English._impl)
    ::std::string d_name;
    // DO-NOT-DELETE splicer.end(greetings.English._impl)
```

```
::std::string
greetings::English_impl::sayIt()
throw ()
{
  // DO-NOT-DELETE splicer.begin(greetings.English.sayIt)
  ::std::string msg("Hello ");
  return msg + d_name + "!";
  // DO-NOT-DELETE splicer.end(greetings.English.sayIt)
}
```

Library User Does This...



1. `babel --client=F90 greetings.sidl`
2. Compile & Link generated Code & Runtime
3. Place DLL in suitable location

F90/Babel "Hello World" Application

```
program helloclient
  use greetings_English
  implicit none
  type(greetings_English_t) :: obj
  character (len=80)        :: msg
  character (len=20)        :: name

  name='World'
  call new( obj )
  call setName( obj, name )
  call sayIt( obj, msg )
  call deleteRef( obj )
  print *, msg

end program helloclient
```

These subroutines come from directly from the SIDL

Some other subroutines are "built in" to every SIDL class/interface

F90/Babe
A

```
package greetings version 1.0 {
    interface Hello {
        void setName( in string name );
        string sayIt ( );
    }
    class English implements-all Hello {  }
}
```

```
program helloclient
  use greetings_English
  implicit none
  type(greetings_English_t) :: obj
  character (len=80)        :: msg
  character (len=20)        :: name

  name='World'
  call new( obj )
  call setName( obj, name )
  call sayIt( obj, msg )
  call deleteRef( obj )
  print *, msg

end program helloclient
```

These subroutines come from directly from the SIDL

Some other subroutines are “built in” to every SIDL class/interface

SWIG v. Babel

(David Beazley @ U Chicago)

- Call from Tcl, Perl, Python, Java, Ruby, mzscheme, or Guile
- Implement in C, C++
- Reads existing code
  - Library User can do independently
  - C++ “type system”
  - Auxiliary .i files fill in details
- Better suited for fast prototyping

- Call from C, C++, F77, F90, Python, and Java
- Implement in C, C++, F77, F90, and Python
- Hand-written SIDL
  - Library Developer task (or “motivated” user?)
  - SIDL “object model”
  - SIDL is self contained, no extra hints needed
- Better suited for production use

Change Oriented Software

- **Absorb change without losing correctness**
- **Empower and exploit the creativity of users**
- **Reduce dependency entanglement among developers**

Babel's Contributions to Change-Oriented Software

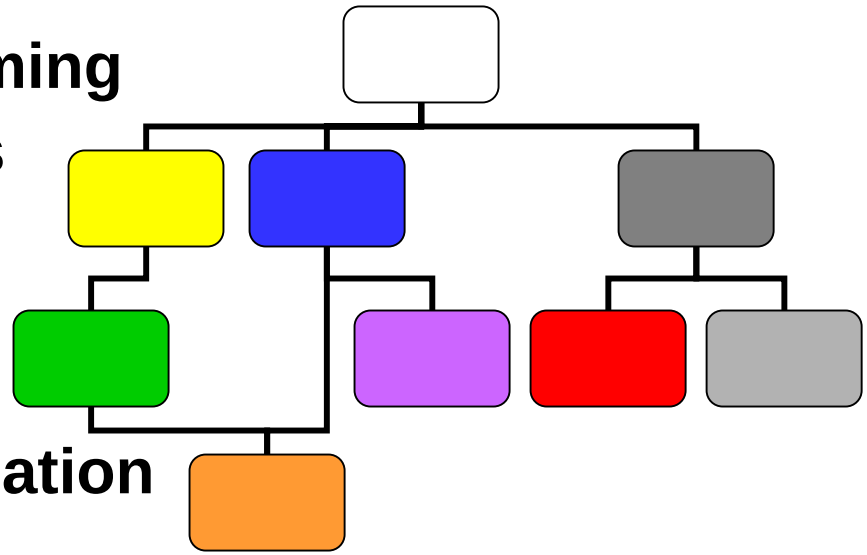
- **SIDL**
  - **Compilable Software Contract btwn developer and user**
  - **Language Independent Standards**
    - **CCA Specification in SIDL**
  - **Version Management of Interfaces**
  - **Ongoing Research: Adding semantic specifications**

Babel's Contributions to Change-Oriented Software

- **Language Transparent Software**
  - **Keeps implementation details from driving the design**
  - **Lowers integration barriers**
- **Stories:**
  - **Babel helps NWChem mix F77 w/ F77**
  - **Babel in Adaptive Algorithm Research**

CCA's Contributions to Change-Oriented Software

- **Pure Babel**
  - still imperative programming
  - assembly of call graph is embedded in code
- **CCA**
  - separates component development from application assembly
  - application assembly can be deferred to last minute (like scripting)
  - Loosely coupled systems are inherently more changeable



For More on CCA

- **CCA tutorial at SIAM Parallel Processing at San Francisco (late Feb)**
- **CCA Quarterly meetings.**
  - **Next one hosted by NCAR in Colorado April 15-16.**

Contact Info

- **CCA Forum: http://www.cca-forum.org**
  - **cca-forum@cca-forum.org**
- **Babel (&stuff): http://www.llnl.gov/CASC/components**
  - **components@llnl.gov ← my team**
  - **kumfert@llnl.gov ← me**