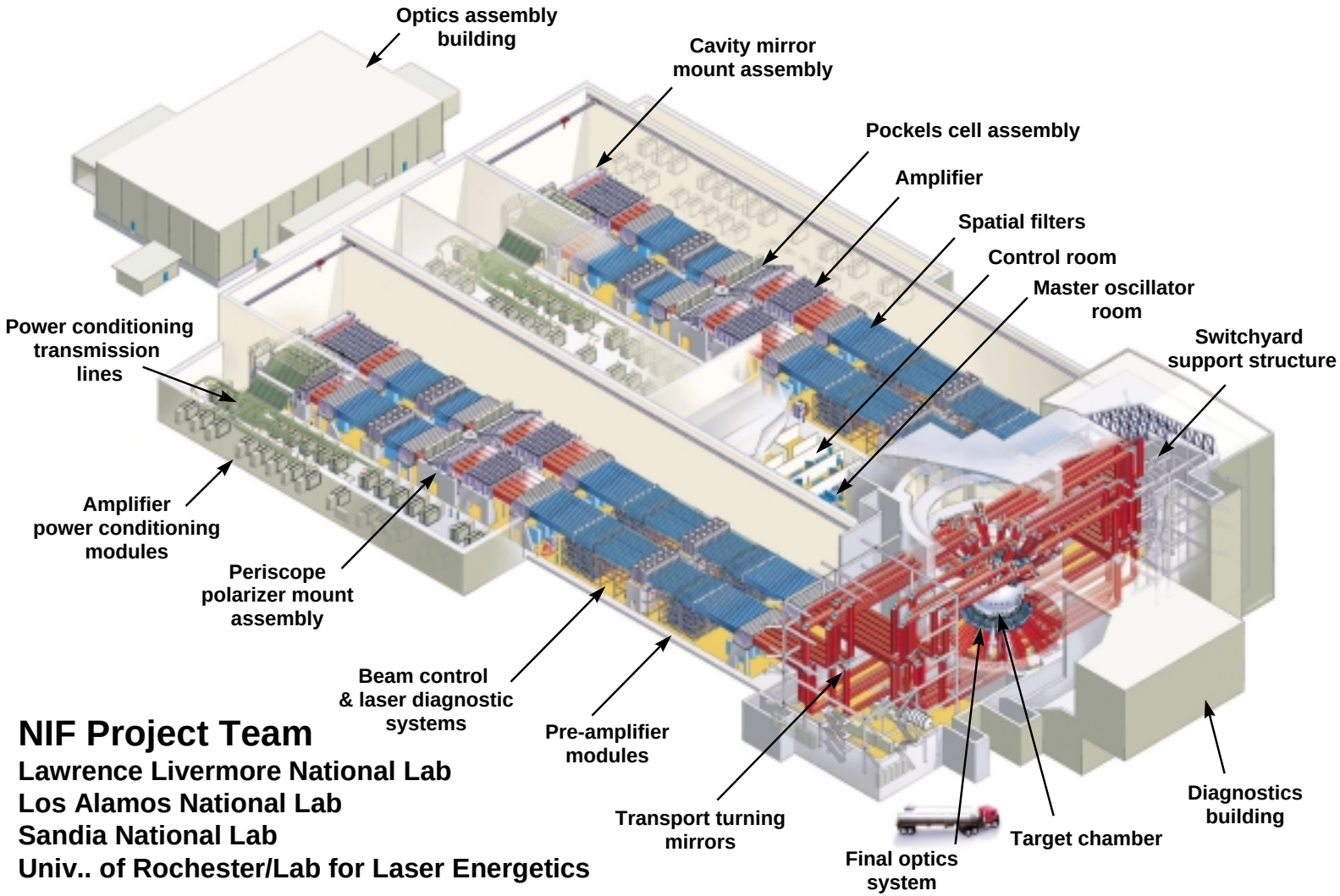
The National Ignition Facility
Integrated Computer Control System

John P. Woodruff
NIF Project Software Architect
Lawrence Livermore National Lab

Presented at Stanford Linear Accelerator Center
13 April 2000

The National Ignition Facility is a high-energy laser for inertial confinement fusion research



NIF Project Team

Lawrence Livermore National Lab
Los Alamos National Lab
Sandia National Lab
Univ.. of Rochester/Lab for Laser Energetics

NIF-0046265
2-jpw/jj

Agenda for Presentation

- **Requirements for NIF Computer Controls**
  - Subsystems and operational scenarios
  - Typical user interface

- **Integrated Timing System requirements and performance**

- **ICCS Software Architecture**
  - Distributed computational resources
  - Frameworks
  - Reusable abstractions
  - Construction of executable processes from generic templates

- **CORBA communication infrastructure**
  - Role of interoperable distributed objects
  - Performance measurements

Computer control system functional requirements

- Centralized control and monitoring of laser equipment
- Maintain machine configuration and operational history
- Coordinate shot countdown and data archiving
- Conduct shot in ‘real-time’ over 2 second period
- Conduct automated shot every 8 hours with 7 by 24 operation

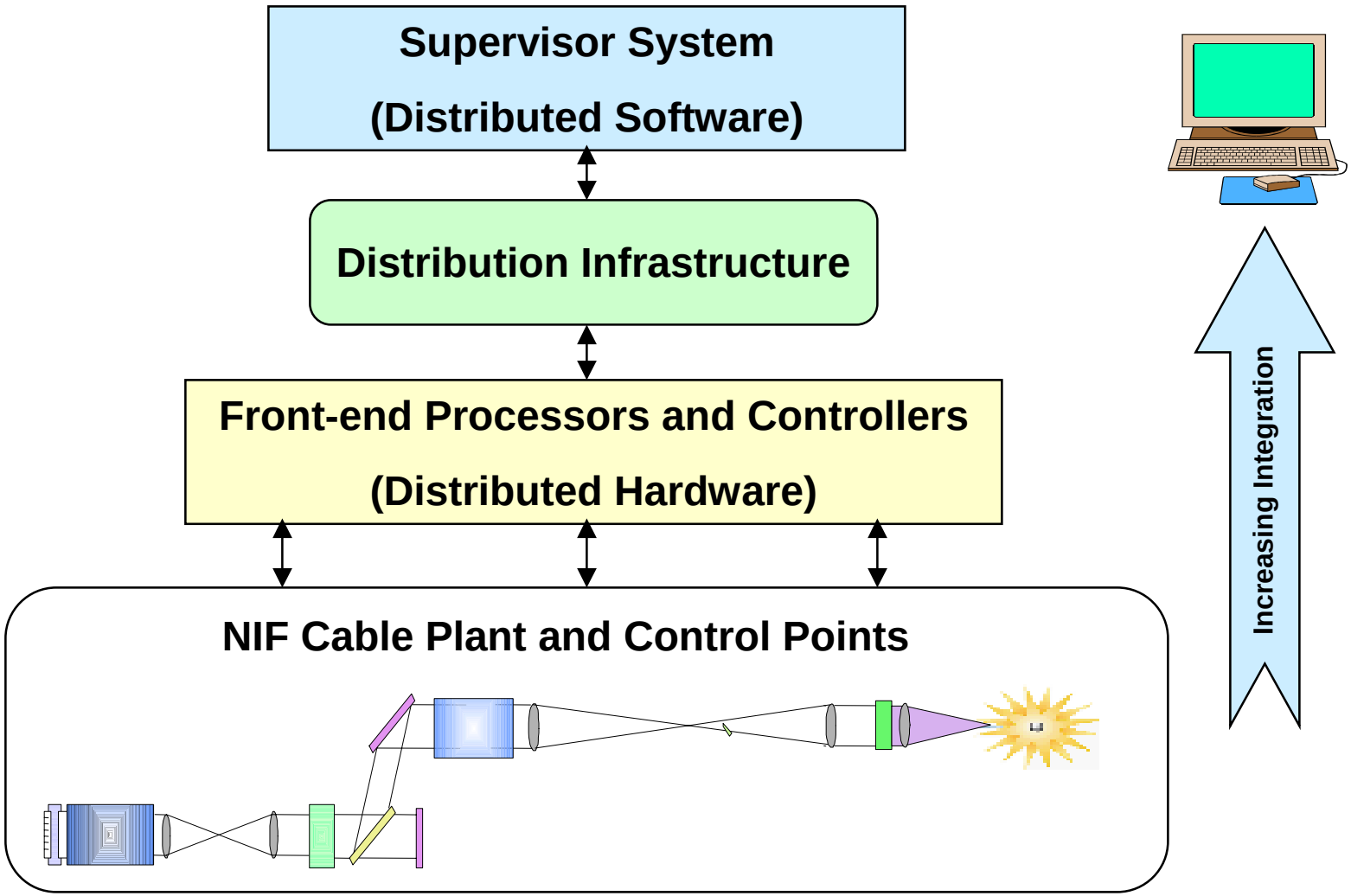
ICCS is a distributed system that does not have hard real time requirements

- **Supervisory software is event driven**
  - Operator-initiated actions and scripted sequences do not require specific response times
  - Speed requirements derive from operator needs for interactive response
  - Status information is propagated from the laser to updates on graphic user screens

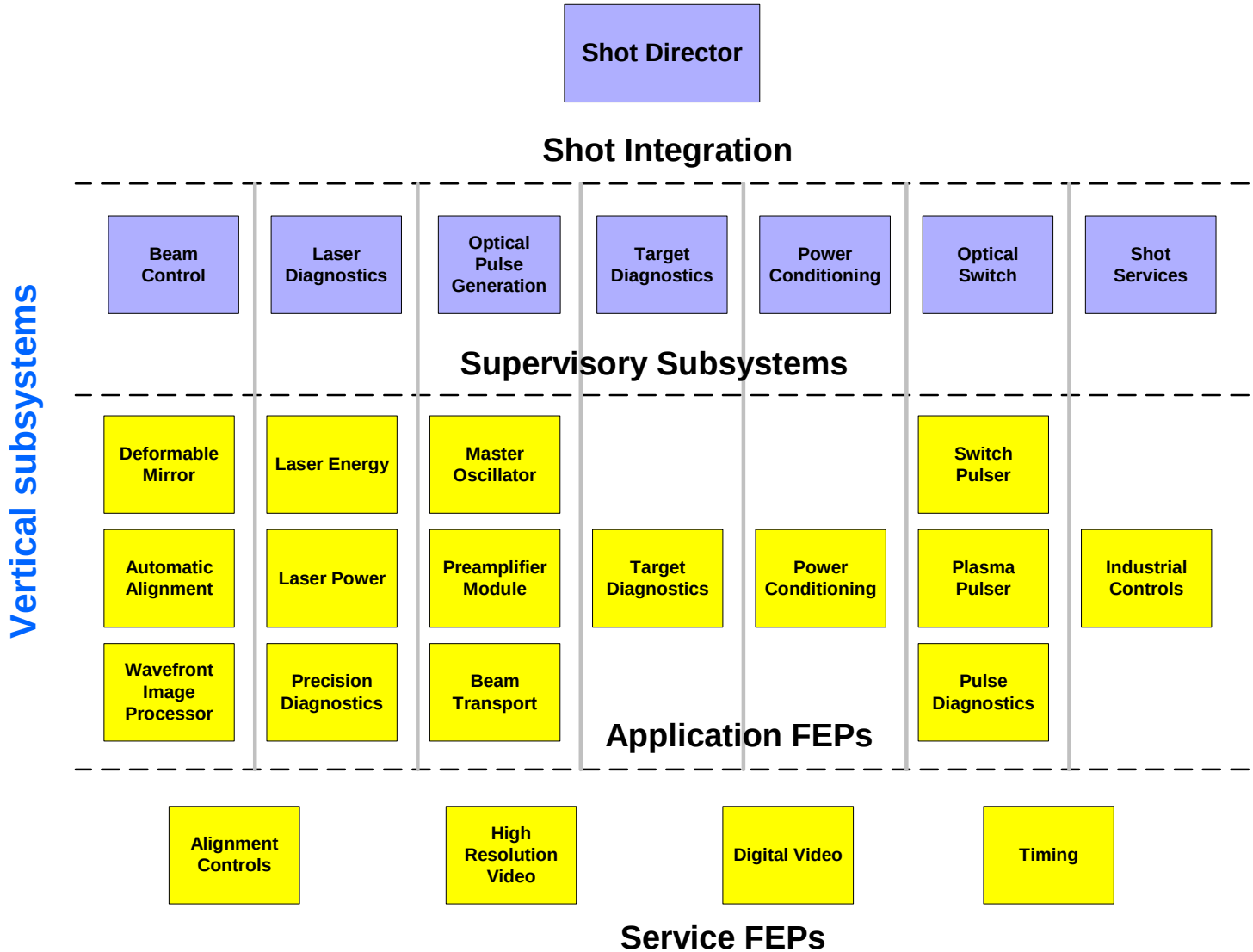
- **No process-related hard deadlines must be met**
  - Several hours of preparation precede shot
  - Shot executes in microseconds, controlled by dedicated hardware
  - Data gathering and reporting occurs in minutes after the shot

- **Some process controls are encapsulated in front-ends**
  - Automatic alignment
  - Capacitor charging

The functional system description of the control system maps to distributed architecture



Functionality is partitioned into subsystems



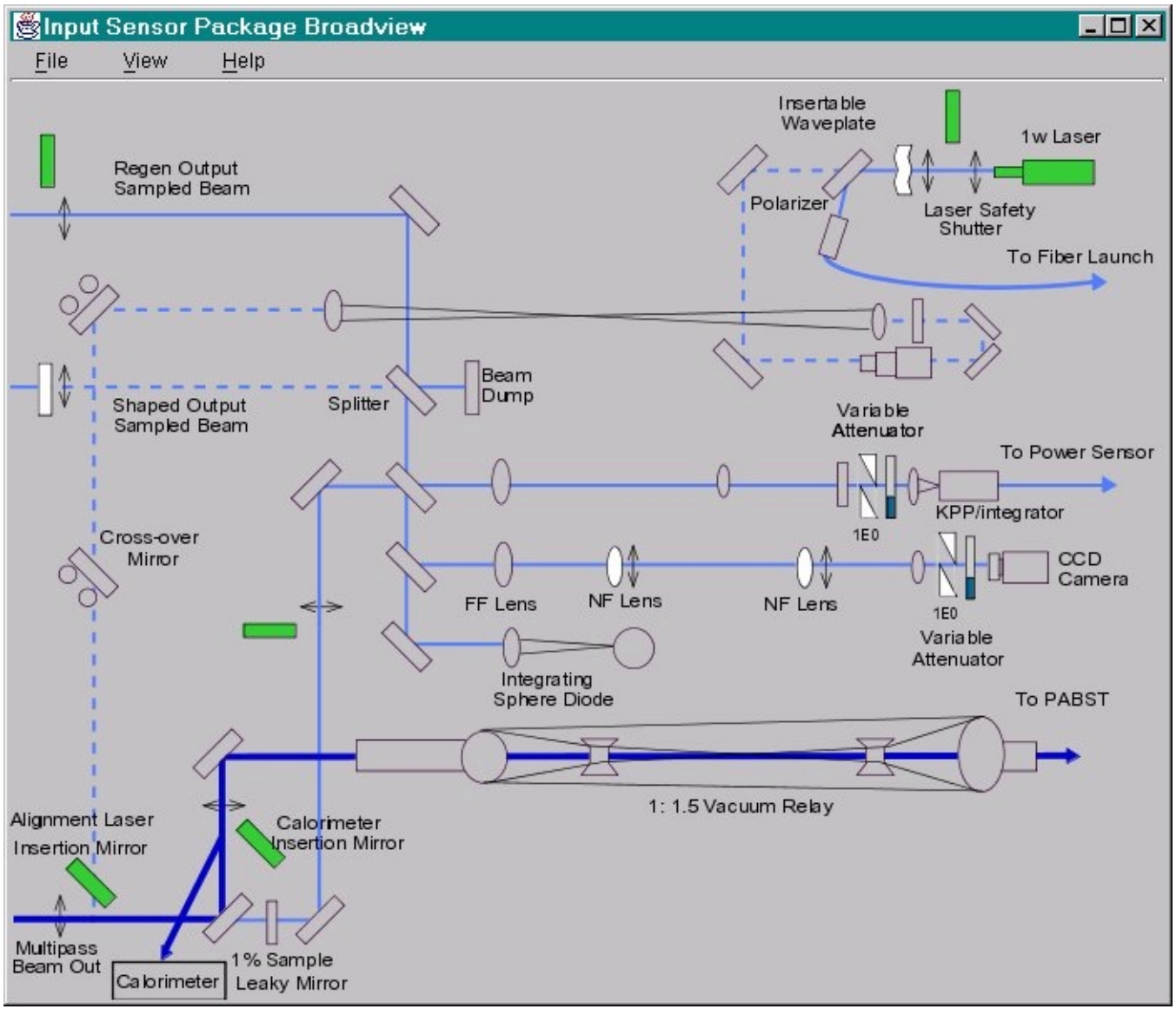
NIF-0046265
7-jpw/jj

The hardware boundary is the solid ground on which we build our software architecture

- **The control points are relatively inflexible**
  - NIF equipment will evolve only slowly
  - Changes to equipment will be expensive
    - Therefore the software can expect to evolve slowly along with equipment evolution

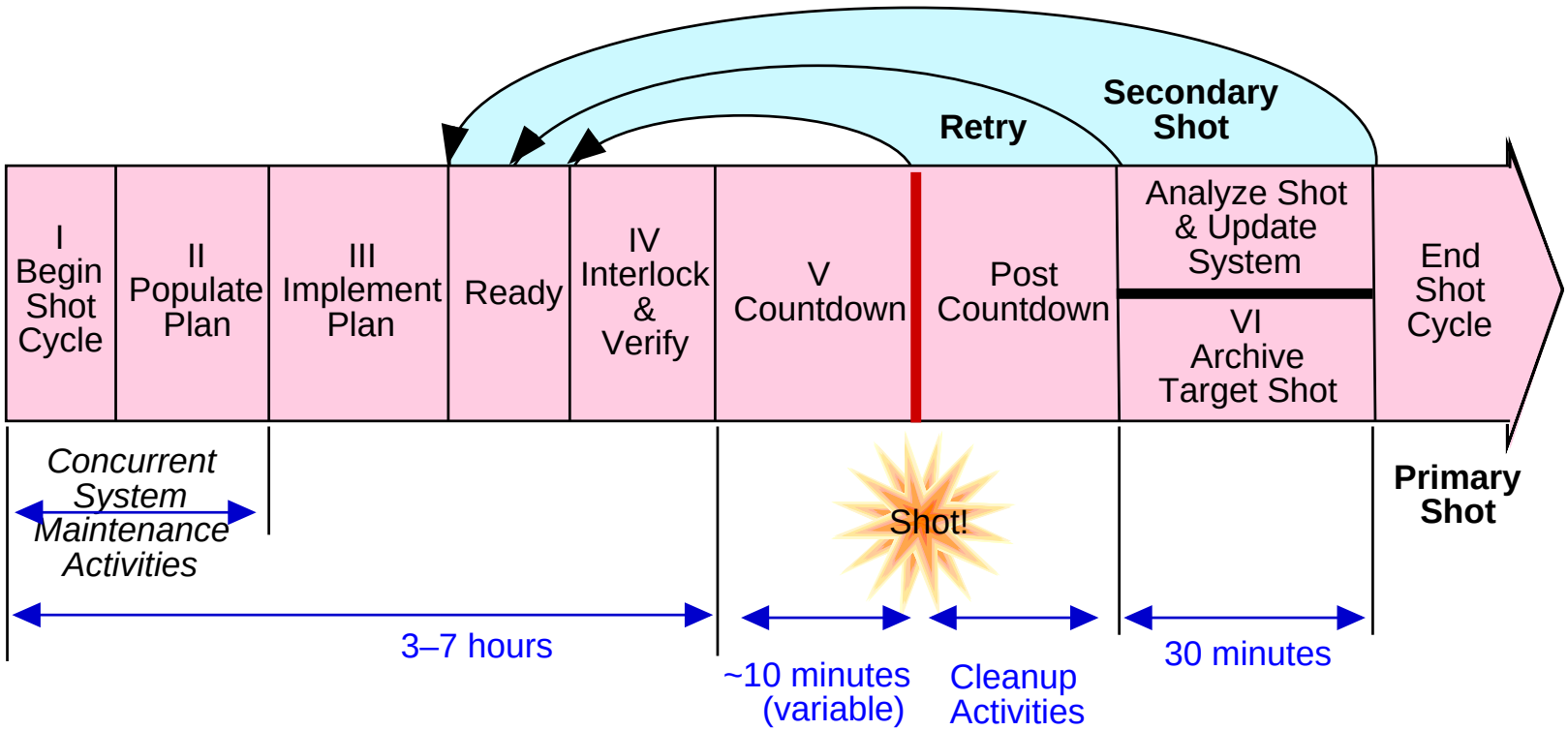
- **By contrast, the user interfaces and experimental execution plans will evolve more rapidly**
  - The user community will learn innovative ways to use the facility
  - Experimental campaigns will arise in response to researchers' creativity

A typical user interface shows broad-view status and offers pop-up control panels

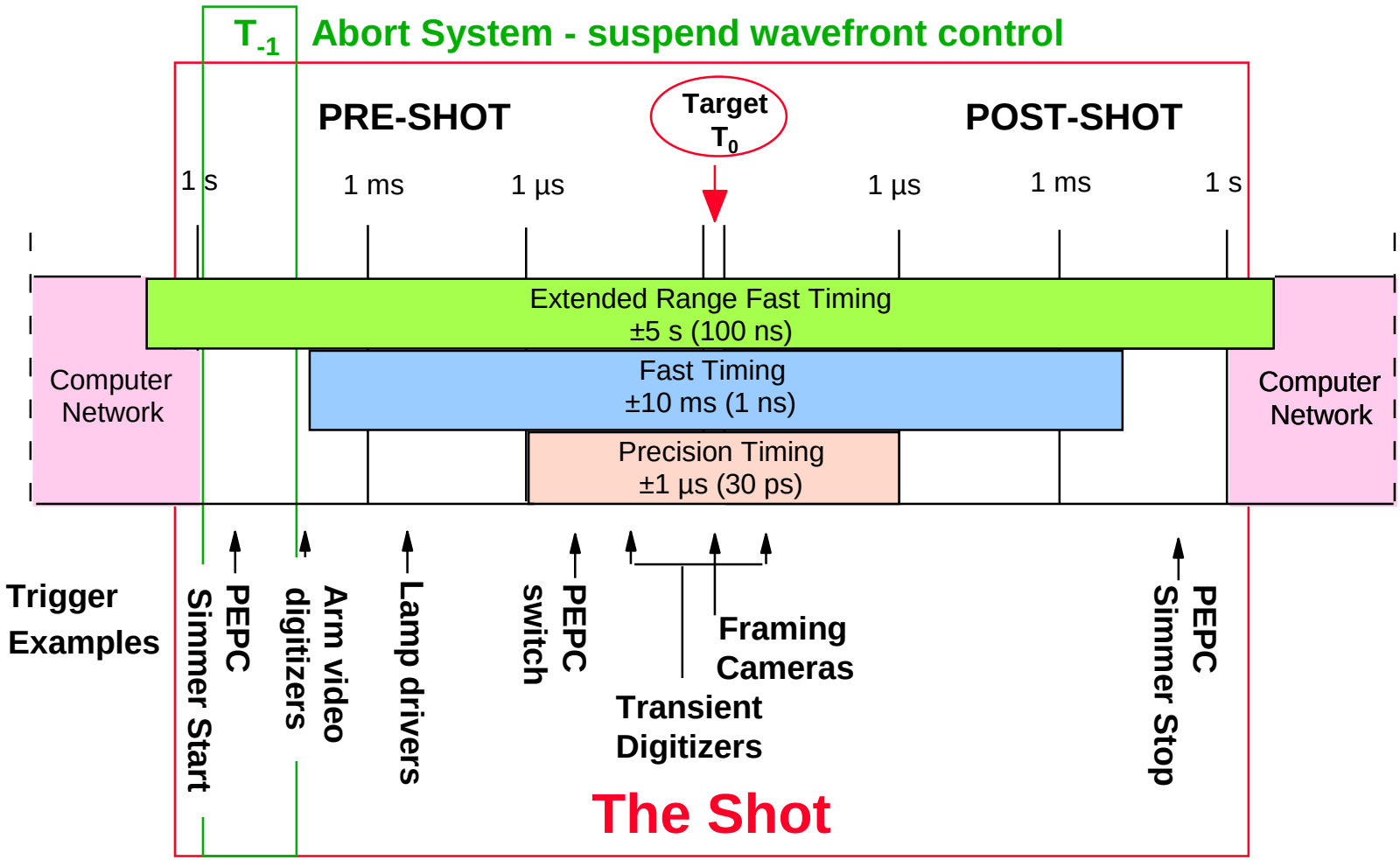


NIF-0046265
9-jpw/jj

Activities that constitute a shot cycle are defined as abstract state transitions

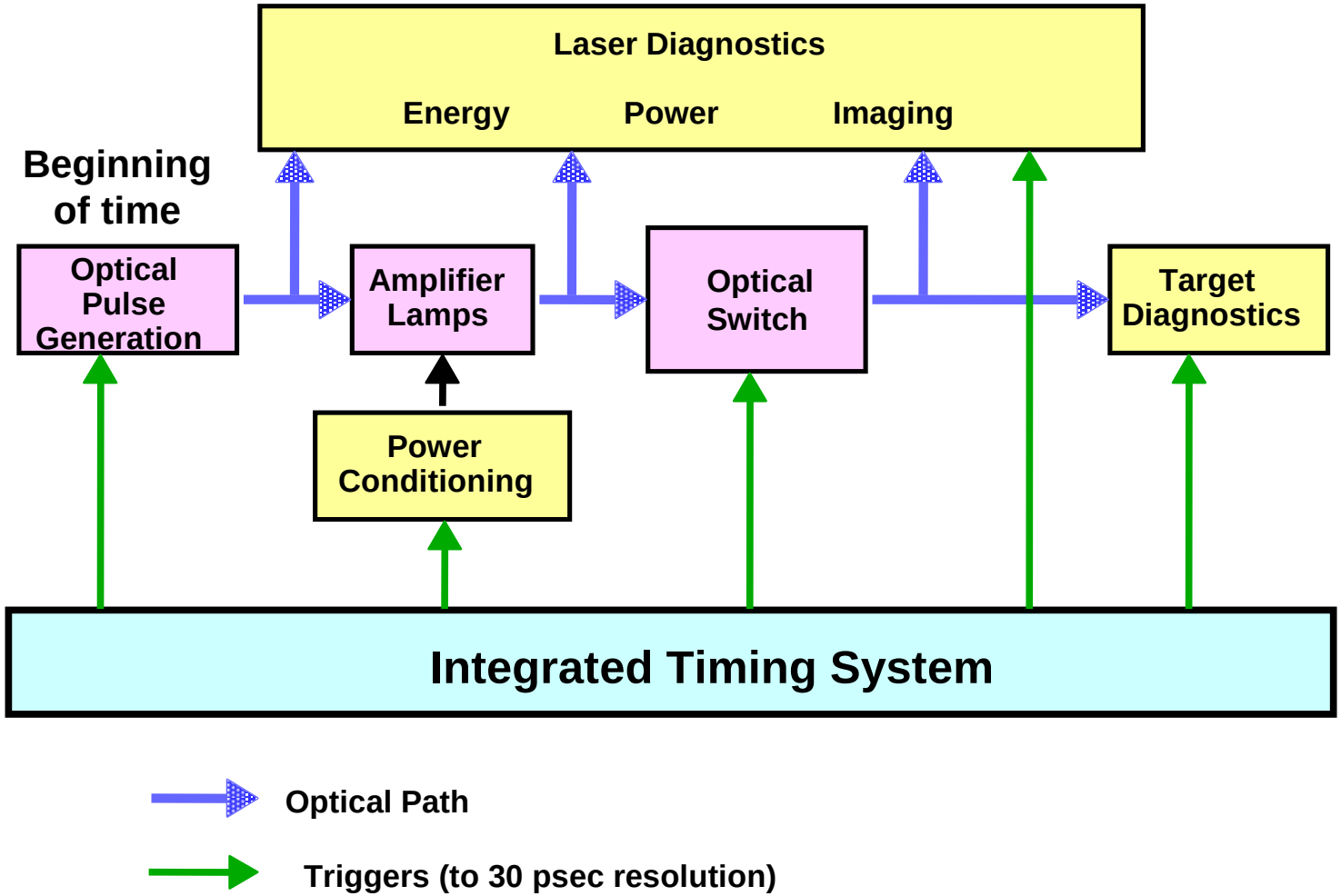


NIF shot in ‘real-time’ lasts 2 seconds under control of dedicated hardware



NIF-0046265
11-jpw/jj

The timing system orchestrates laser firing and triggering of diagnostics

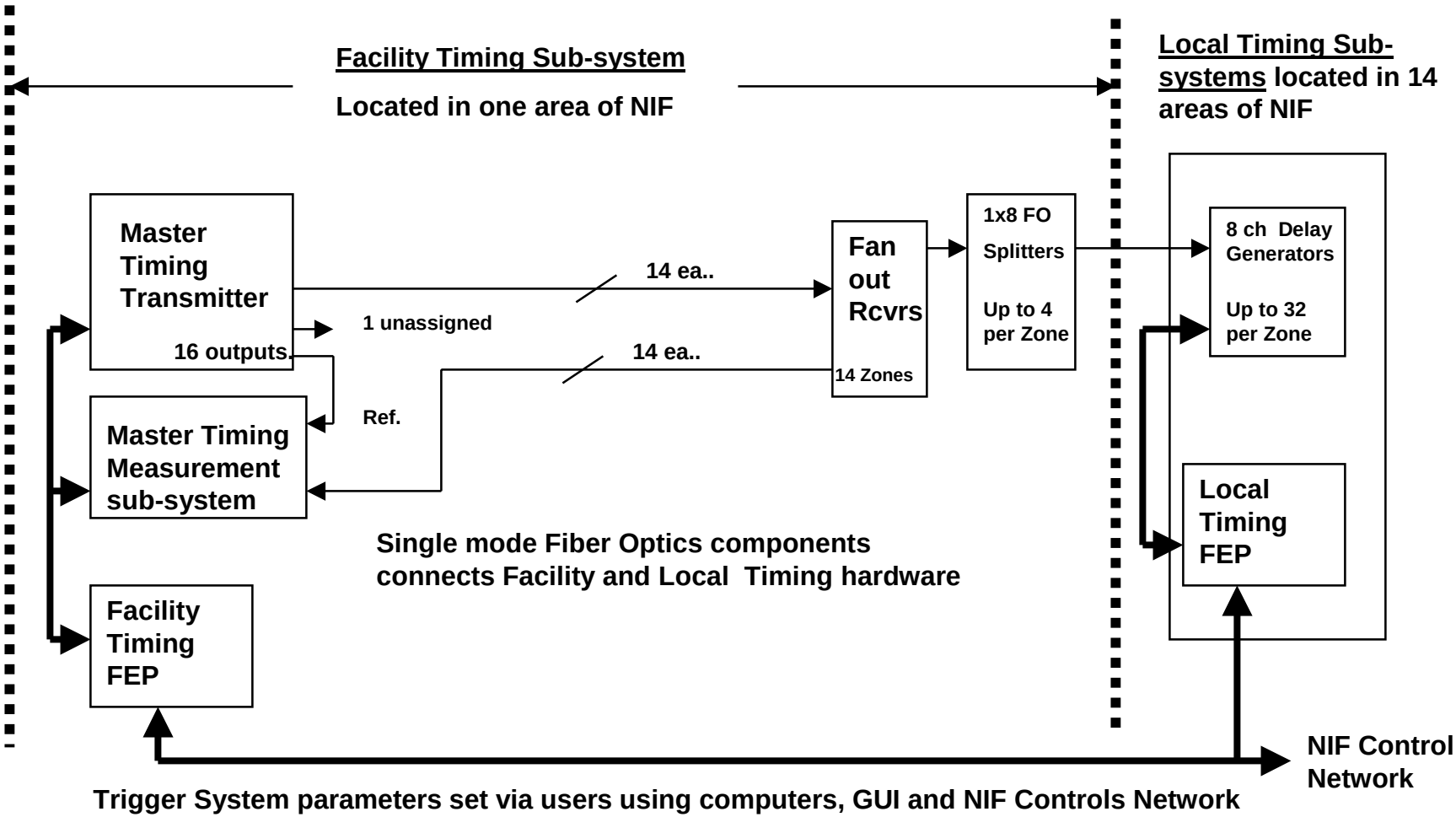


Trigger System Requirements

	Extended Range	Fast units	Precision units
# of channels	150	1900	50
Minimum range	+/- 1 sec.	+/- 55 msec.	+/- 10 usec.
Resolution (setting)	<100 ns	< 1 ns	20 ps
Stability (jitter)	<1 ms (jitter & wander)	<100 ps RMS (over 10 sec)	<20 ps RMS (over 10 sec)
Stability (wander)	See above	< 500 ps - pk to pk (over 7 days)	<100 ps 95 % (over 7 days)

Trigger system architecture

ITS Trigger system is divided into two functional sub-systems



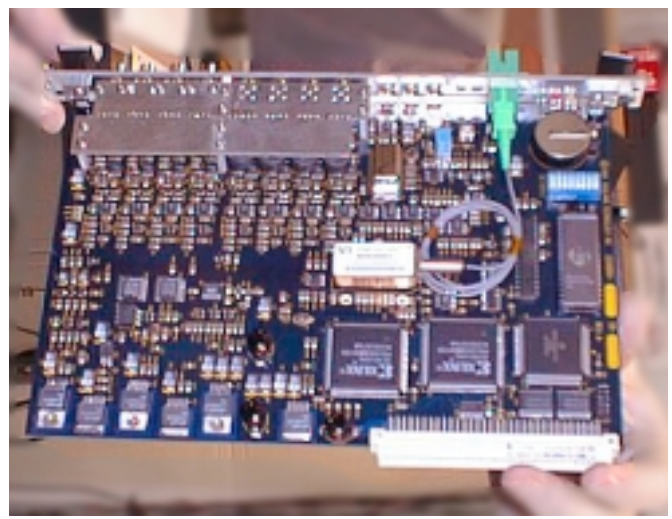
NIF-0046265
14-jpw/jj

First-article timing components have been demonstrated to exceed NIF requirements

Timing transmitter

Parameter	Specification	Verified
Delay range	2 sec	2 sec
Resolution	< 20 ps	7 ps
Short-term stability	< 20 ps RMS	5 ps RMS
Long-term stability	< 100 ps	< 50 ps

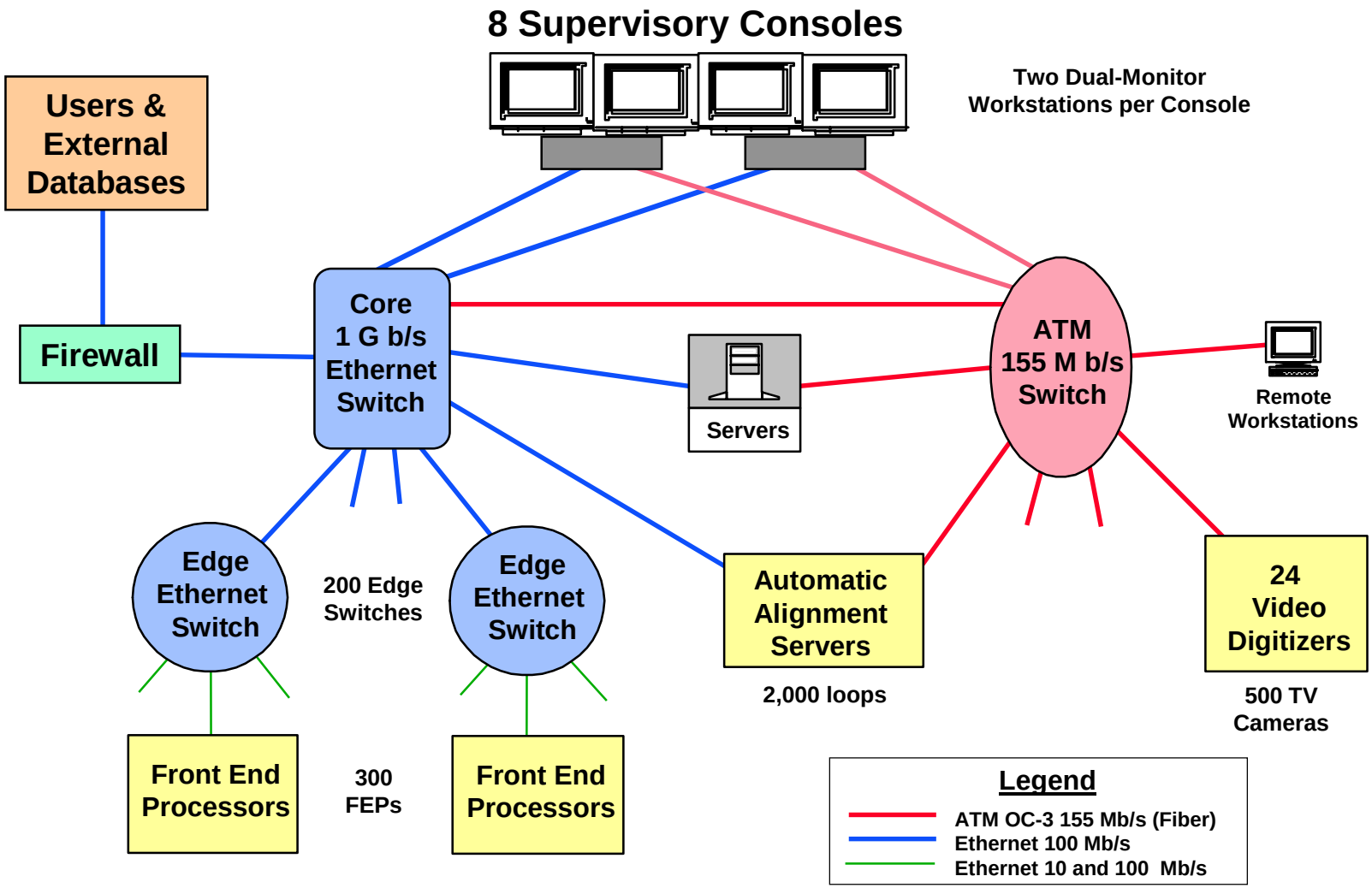
Measurement system



Precision delay generator

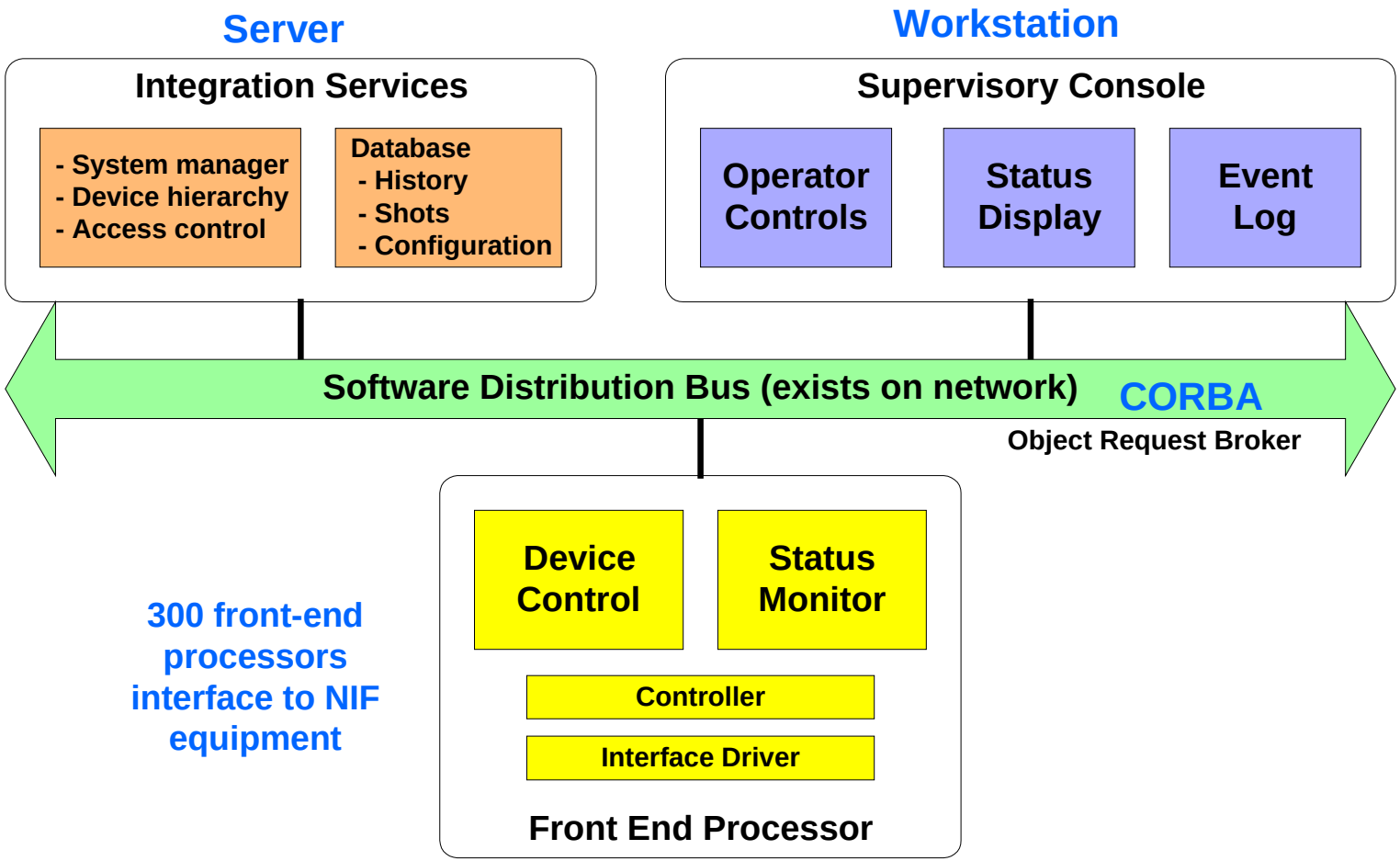
NIF-0046265
15-jpw/jj

The computer network employs switching technology to assure performance



NIF-0046265
16-jpw/jj

Software applications are built upon a framework of distributed services



Software objects representing control points "plug in" to the software distribution bus

The ICCS software architecture centers on widely used “Framework” components

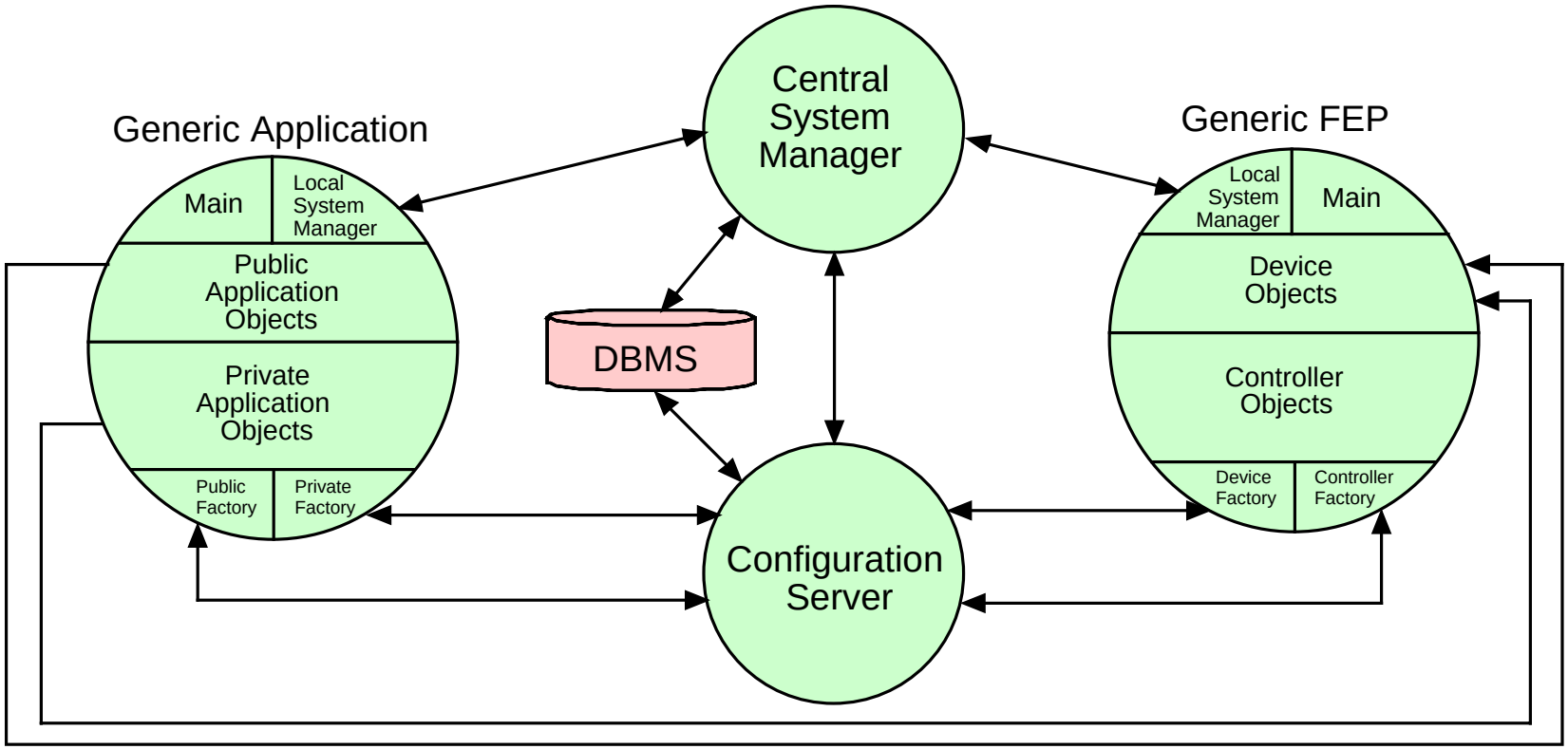
- **Our frameworks have been discovered by domain analysis**
  - Experience with similar experimental facilities
  - System requirements that span subsystems
  - Abstractions of services

- **The dozen frameworks fall into three categories**
  - Abstract services
    - “System Manager” starts processes, observes performance
  - Architecture - specific services
    - “Configuration” initializes the state of persistent objects
    - “Sequence control” embeds a scripting language into control objects
  - NIF - specific operational services
    - “Shot life cycle” abstracts the states that all subsystems enact in an experiment

These abstract frameworks are being built with prospective reuse in mind

- **Managing the lives of processes and application objects**
  - System manager
  - Generic main programs
  - Configuration: delivers database services
- **Organizing operational records**
  - Message Log
  - Machine history
  - Shot Data archive
- **Distributing up-to-date device status**
  - Status monitor: polls locally and pushes updates
- **Managing interactions with operators**
  - Graphic user interface
  - Reservation
  - Sequence control language
  - Alert notification
- **Implementing the state transitions in an experiment**

All supervisor and FEP programs are built by elaborating a generic template



Abstract frameworks are (largely) independent of each other

- **Four frameworks provide distinct information services**
  - Configuration: data to start devices
    - Example: signal level addresses, instrument calibration
  - Message log: audit trail of operator action and system responses
  - Machine history: service records of device performance
  - Shot archive: results of physics experiments

- **The different information services share common features**
  - Devices are named consistently in each data record
  - Records can be correlated, for example by time stamp

- **But the Policies that connect them are not inherent in the frameworks themselves**

- **Additional templates (for innovative frameworks) can be introduced without disturbing components already in place.**

Frameworks are constructed in layers to permit retargeting

NIF
The National Ignition Facility

ICCS Programs

Client and server mainlines

Supervisor applications,
Front End Processors,
Database servers,
etc

Framework Services

Customized for a specific system

Configuration Server,
System Manager,
GUI's,
etc

NIF Building blocks

Classes that model equipment

Devices,
Shot phases,
etc

Framework Templates Layer

Abstract classes for control systems

Abstract devices,
Configuration,
Monitor,
etc

Support layer

COTS and components

Oracle DBMS,
ORB*express*,
etc

The framework services layer is specific to NIF, built by extending reusable framework abstractions

- **The Framework Services are delivered when dispatching operations defined in the template are applied to concrete classes.**
  - The “Device” class is an abstract superclass
    - This base class defines interfaces applicable to all devices
      - for naming
      - for reserving on behalf of an operator
      - for multi-task safety
    - Several dozen derived classes control physical equipment
      - Diverse actions defined for motors, power supplies, diagnostic instruments, precision timing and triggering
      - Initialization from a central database
  - Descendents of the abstract device class provide actual operations to control physical NIF parts

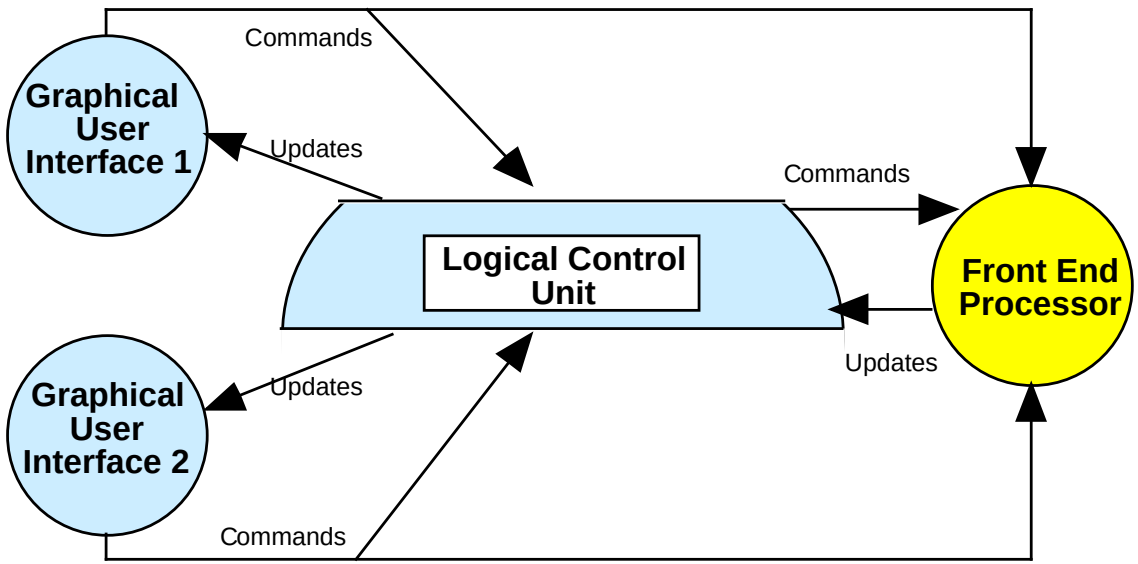
Dependencies between levels are strictly hierarchic

- **Subsystems within a particular layer can only depend upon subsystems at the same or a lower layer**
  - This allows classes at a given layer to be replaced or extended
  - Only layers above the replacement are affected

- **Replacement of the all the concrete classes derived from Device could make the "Application services" frameworks available for a different kind of experimental facility**
  - Adding new subclasses of Device enables evolution of the NIF
  - Replacement of state transition actions would produce new operational services

Numerous GUI's receive status updates from Supervisors

- Separate the operator interaction from system behavior
- Provide a consistent multi-display view of the system state
- Economize on message traffic by "pushing" status changes



Efforts to economize on message traffic

- **Status of every Device must be observable at multiple consoles**
  - Some status reports require latency as small as 0.1 second
  - Monitor objects are co-located with Devices
    - Local polling in the FEP
    - Notification of “significant” change

- **Supervisory objects collect and collate change reports**
  - GUI’s that display “broad view” status subscribe to these supervisors
  - GUI’s receive their status updates via “data push” from the supervisor

CORBA provides decentralized distribution services

- **A standard model of distributed objects resolves a major development risk**
  - ICCS software engineers are freed from building a “homebrew”communication infrastructure
  - Anticipate 30-year life of the standard

- **CORBA defines loose coupling between objects**
  - Communication becomes nearly invisible
    - Neither clients nor servers depend directly on communication infrastructure
    - Names of communicating objects hide locations
  - Transparent interoperability
    - IDL specifications are language-neutral interfaces
    - Data marshalling hides differences between hosts

- **Allocation of object implementations to processes can be deferred**

ICCS uses CORBA to distribute Ada-95 objects

- **Each of the 60,000 control points is controlled by one of the Front-End processors**
  - Each is implemented as an instance of a class derived from Device
  - These derived classes are specified using Interface Definition Language (IDL)

- **The ORBexpress IDL compiler translates the IDL to an Ada interface package and an implementation package**

- **“Abstract” classes that are defined by IDL translate to a concrete interface defining a classwide reference**

- **Framework objects perform their operations by dispatching calls on these classwide references**

- **ORBexpress produces invocations of the methods in the corresponding implementation**

The majority of NIF's CORBA objects are long-lived

- **60_000 objects implement the class Device in Front-End Processors**
  - About 130 subclasses
  - Each instance is initialized at system start-up
  - A framework manages data and naming
    - Oracle database maintains configuration
    - Persistence broker objects implement SQL queries on behalf of CORBA clients

- **A dead server is an error to be diagnosed and recovered**
  - Failover to a replacement of the same class is not automated

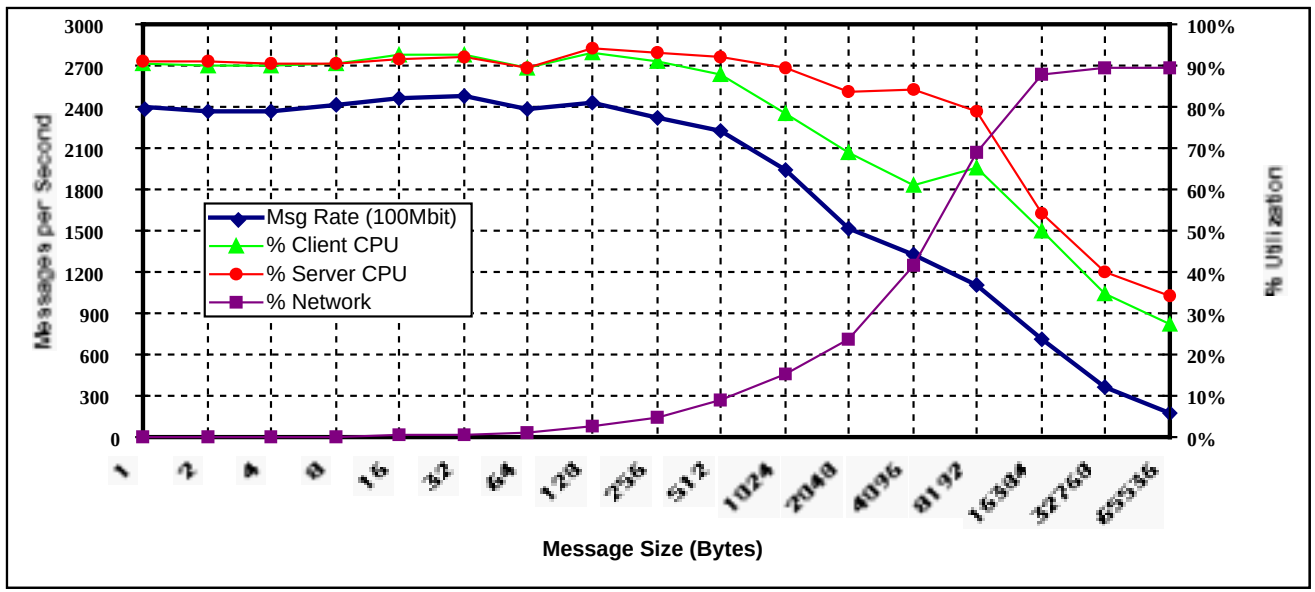
Using IDL to define interfaces implies some compromises

- **Interfaces must be declared in terms of IDL types**
  - These types “diffuse” into the rest of the system
  - IDL type model is less strict than Ada’s
    - No range constraints
    - No initial values for record components
  - No default parameter values
  - No operator overloading in interfaces

- **Configuration management must accommodate to the possibility that implementation details might be loaded into client processes**

Measurements of ORB*express* 2.0.1 confirm adequate performance

- **Network is 100 Megabit ethernet**
- **Both client and server are 2-processor Sun Enterprise 3000's**
  - Client runs 40 Ada tasks; server runs 5
  - Runtime is Apex 3.0
    - GNAT 3.11 is roughly 10% faster

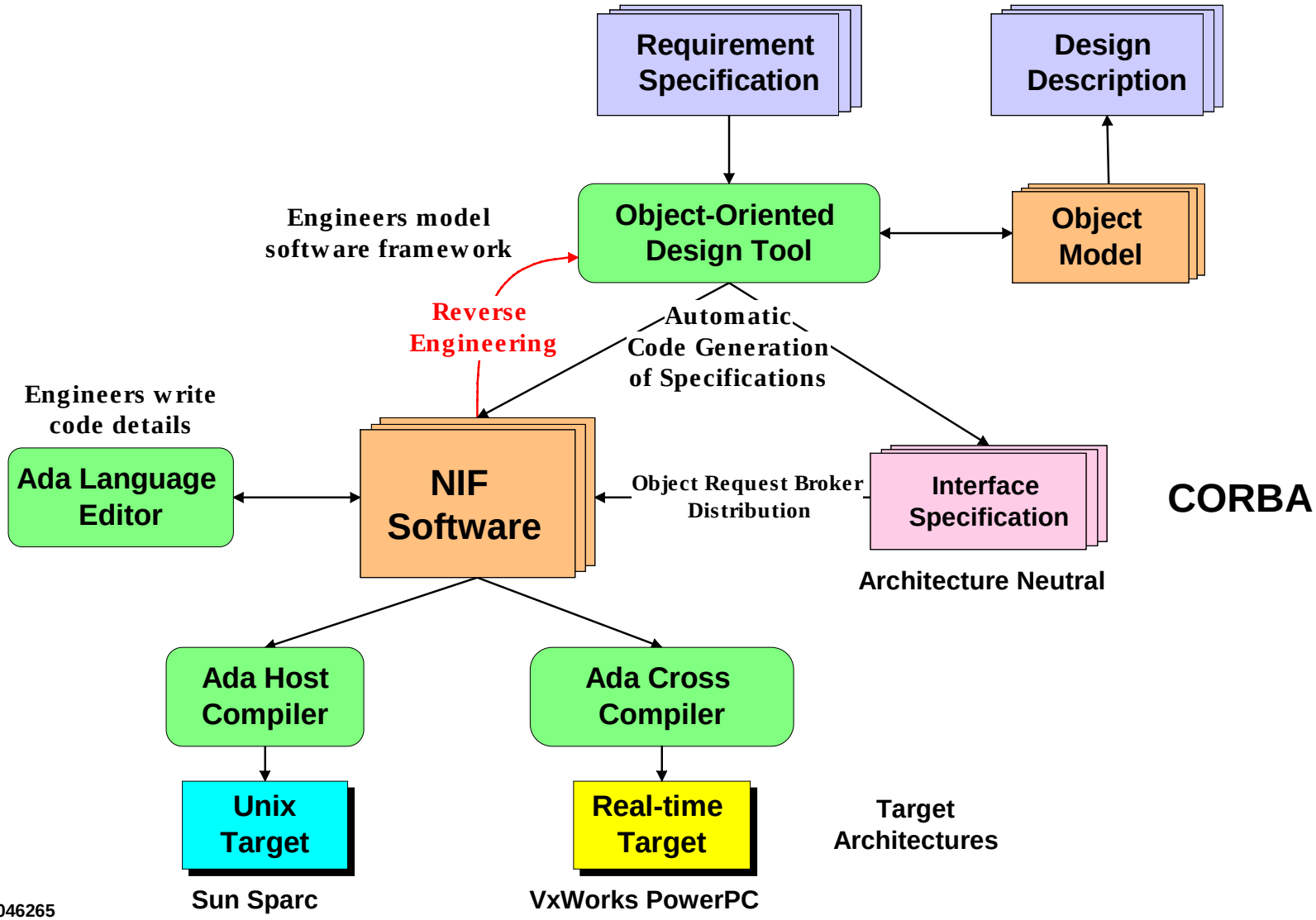


The ICCS strategy rests on two main decisions

- **Single unified architecture unites all subsystems**
  - Frameworks implement abstractions for widespread use
  - Distributed object-oriented system exploits CORBA
  - Design patterns embody programming choices
    - Publisher-subscriber relationships
    - Model-view-controller idiom for user interface

- **Managed process guides development**
  - Ada is the principle programming language
  - Documents are written and reviewed
  - Development proceeds incrementally
  - Code walkthroughs catch errors early in cycle
  - Each cycle of development is reviewed
    - Process is adjusted to incorporate lessons learned

A disciplined engineering process manages incremental construction and release of code



NIF-0046265
33-jpw/jj

DISCLAIMER
UCRL-VG-138473

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