ALMA Common Software

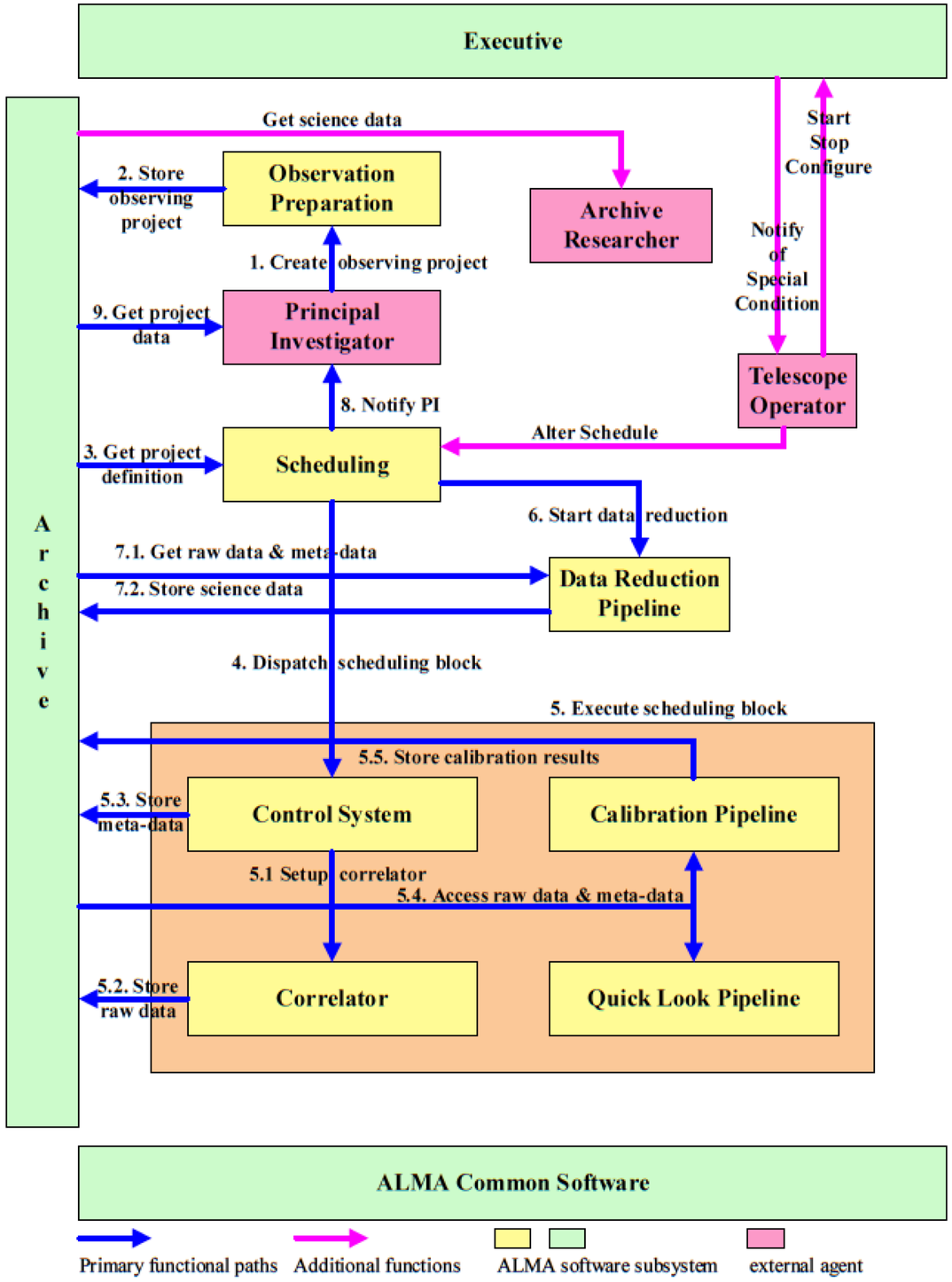
Basic Track

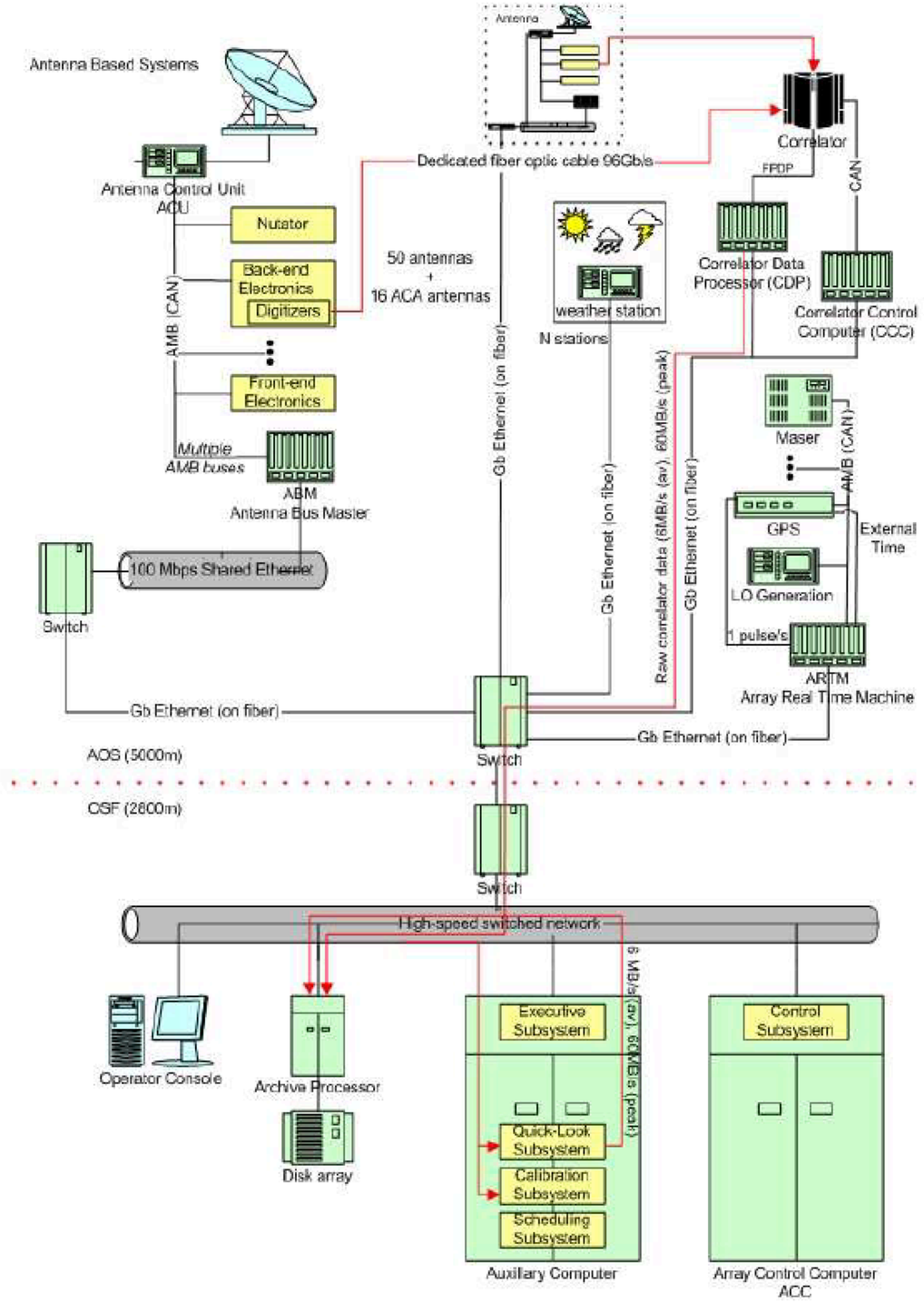
Introduction to the ACS Framework

The Modern Observatory

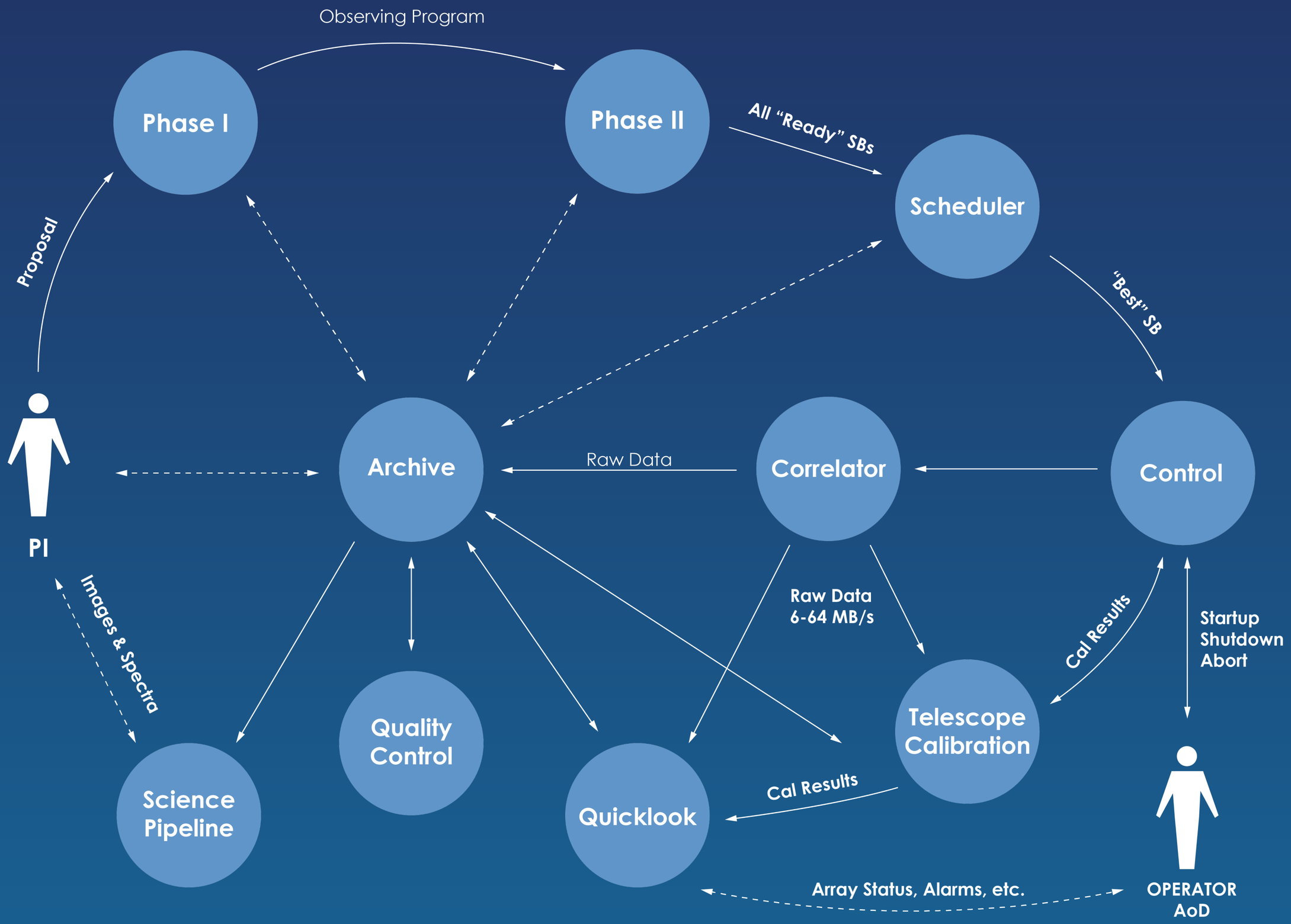
- ✧ Integrated and distributed architecture (end to end data flow)
- ✧ Archive and virtual observatory integration.
- ✧ Feedback systems and distributed control loops.
- ✧ Big projects and small projects: what are the differences?

Example: ALMA Software and Physical Architecture





Example: ALMA Data Flow



Distributed System

Requirement: the observatory is a distributed system.

Servers and clients are distributed on different machines:

- ✧ Possibly in different locations
- ✧ With different purpose and functionality
- ✧ With different requirements on performance and reliability

This leads to a…

Heterogeneous Distributed System

Requirement: the observatory shall be a heterogeneous distributed system.

Servers and clients may use different:

- ✧ Hardware
- ✧ System software
- ✧ Programming languages

Transparent Heterogeneous Distribution

- Developers of clients should be unaware of the underlying server architecture & vice-versa
- It should be possible to change the architecture of a server transparently to the client
- Client developers should not even need to know whether a server is local or remote.

Functional and Technical Architecture

The separation of functional from technical concerns is a strategy for enabling the application developer to concentrate on the specific aspects of the observatory

✧ Expressing the complexity in software of an observatory is difficult

✧ Having to know the subfields of computer science associated with distributed object architecture is also very challenging

Conclusion

Let system developers take care of the computer science-related tasks

Functional Architecture

A Functional Software Architecture (FSA) is a model that identifies enterprise functions, interactions and corresponding information technology needs.

- ✧ Software components/subsystems
  - ✧ Responsibilities
  - ✧ Interfaces
  - ✧ Primary relationships and interactions
- ✧ Physics and algorithms
- ✧ Hardware deployment and distribution

It is developed by architect and subsystem leaders

Technical Architecture

The functional architecture must be supported by a technical architecture that describes (and implements) the technical aspects of the software

✧ Access remote resources
✧ Store and retrieve data (Database technology)
✧ Manage security needs
✧ Communication mechanisms and networking
✧ Software deployment and activation
✧ Programming model

It is provided by the technical team

Separation of concerns

- ✧ Functional and technical architecture: two views
- ✧ Subsystem teams should concentrate on function
- ✧ Technical architecture provides them with simple and standard ways to, for example:
  - ✧ Access remote resources
  - ✧ Store and retrieve data
  - ✧ Manage security needs
- ✧ Keep the two concerns separate!

The key to the separation between Functional and Technical Architecture

Purpose of a framework is to:

- ✧ provide a programming model
  - ✧ Ensure that the same thing is done in the same way in all the development locations
- ✧ provide common paradigm abstractions
- ✧ mask heterogeneity
- ✧ satisfy performance, reliability and security requirements

The ALMA Common Software (ACS) Framework

- ACS provides the basic services needed for object oriented distributed computing. Among these:
  - Transparent remote object invocation
  - Object deployment and location based on a container/ component model
  - Distributed error and alarm handling
  - Distributed logging
  - Distributed events
- The ACS framework is based on CORBA and built on top of free CORBA implementations.
- Model driven development with code generation

Supported Platforms

- Operating system: RHEL 5.5 and 6.5 (32 and 64 bits)
  - CentOS/SL 5 and 6 binary compatible
  - Other linux versions supported by external projects
  - Windows added also by external initiatives
- Real-time: RTAI
  - VxWorks supported by and for APEX
- Languages: C++, Java, Python
- CORBA middleware: TAO (C++), JacORB (Java), Omniorb (Python), CORBA services.
- Embedded ACS Container: PC104, Debian, 300Mhz Geode, 256MB RAM, 256 MB flash (CosyLAB microIOC), …

LGPL and open source software

The strategy to provide common features to users is:

- Use as much as possible open-source tools, instead of implementing things.
  - Do not reinvent the wheel
  - Reuse experience of other projects
  - Do not pay for licenses
  - Support from user community
- Identify the best way to perform a task among the possibilities
- Wrap with convenience and unifying APIs

ACS is distributed under the LGPL license

Open source software may have drawbacks:

- Fast lifecycle and support only of the newest
- Free/commercial support
- Documentation not as good as commercial products

Separation of roles

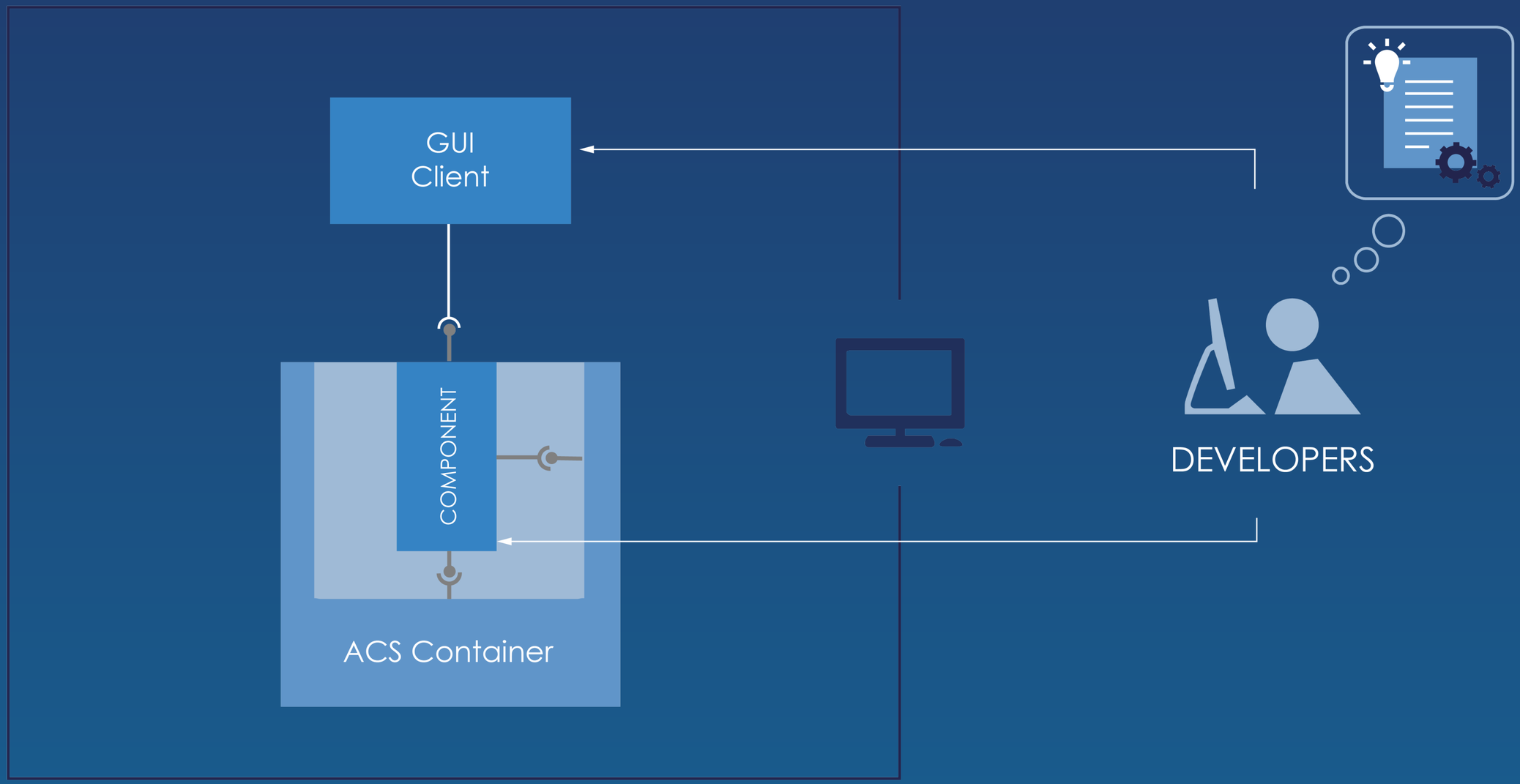
ACS keeps separate 3 roles/phases:

- ✧ Development by software developers
- ✧ Deployment by deployment engineers
- ✧ Runtime by system operators

Development

- ✧ Developers write components and graphical user interfaces clients in C++, Java, or Python.
- ✧ ACS provides an integrated build environment based on application code modules.
- ✧ Communication from an application to a component, and among components, uses ACS as middleware.
- ✧ No thinking about starting and stopping components, or on which machine they should run later.

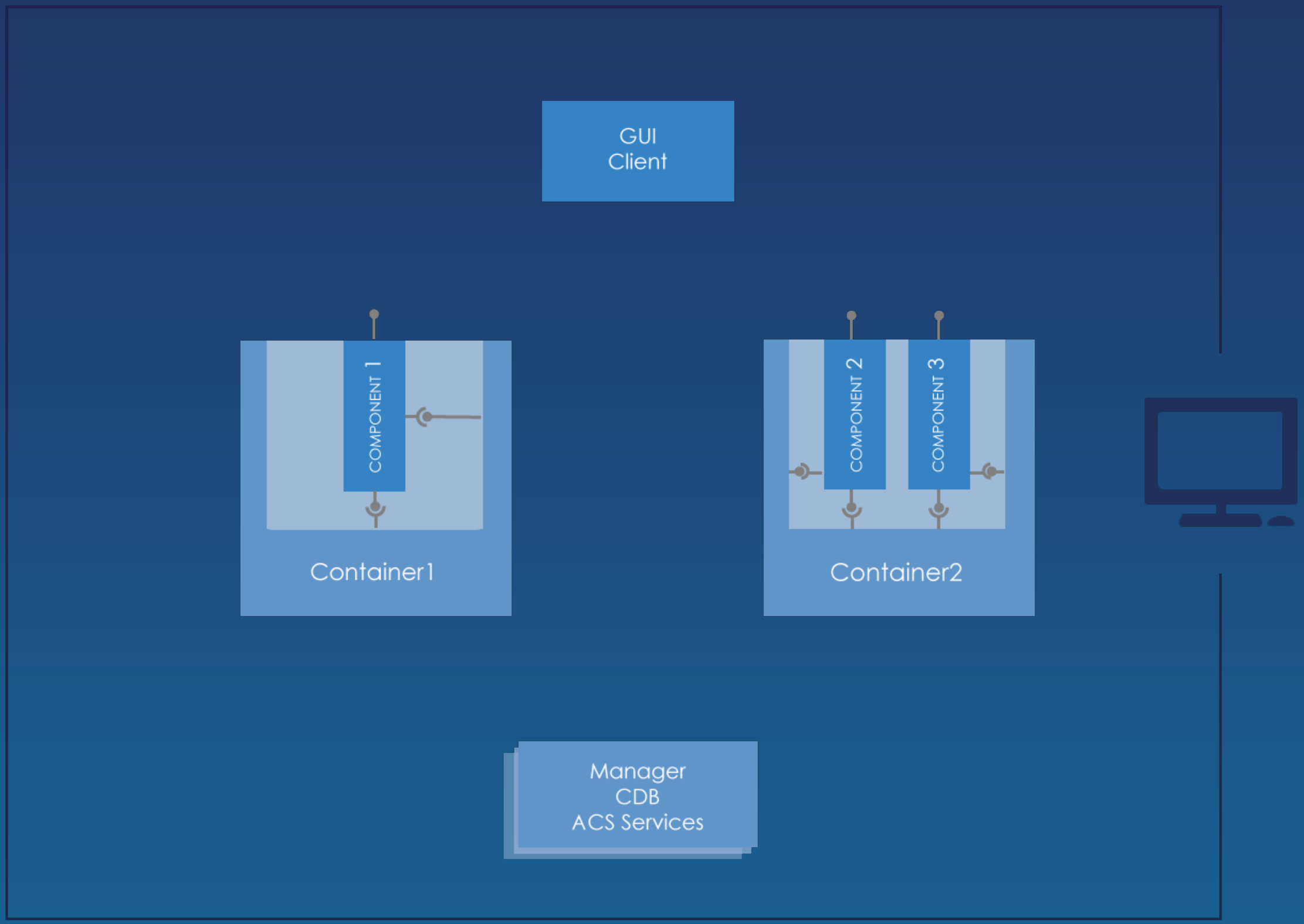
Development



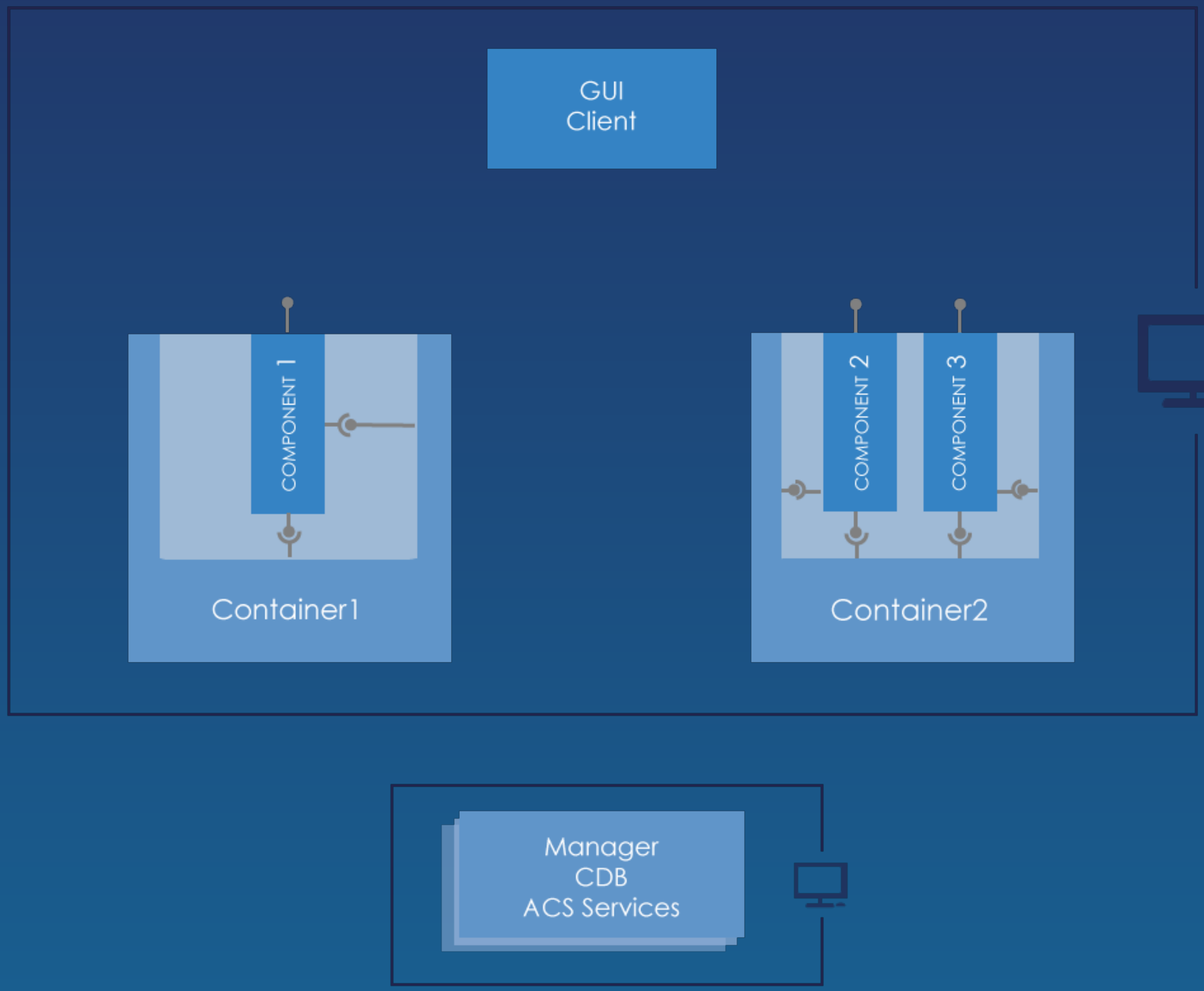
Deployment

- ✧ One or more containers get assigned to each computer.
- ✧ Components get assigned to containers.
- ✧ This location information is stored centrally in the Configuration Database (CDB).
- ✧ Other configuration data for containers and components are also stored in the CDB.
- ✧ There can be different deployments for unit tests, system tests, and various stages of the production system.

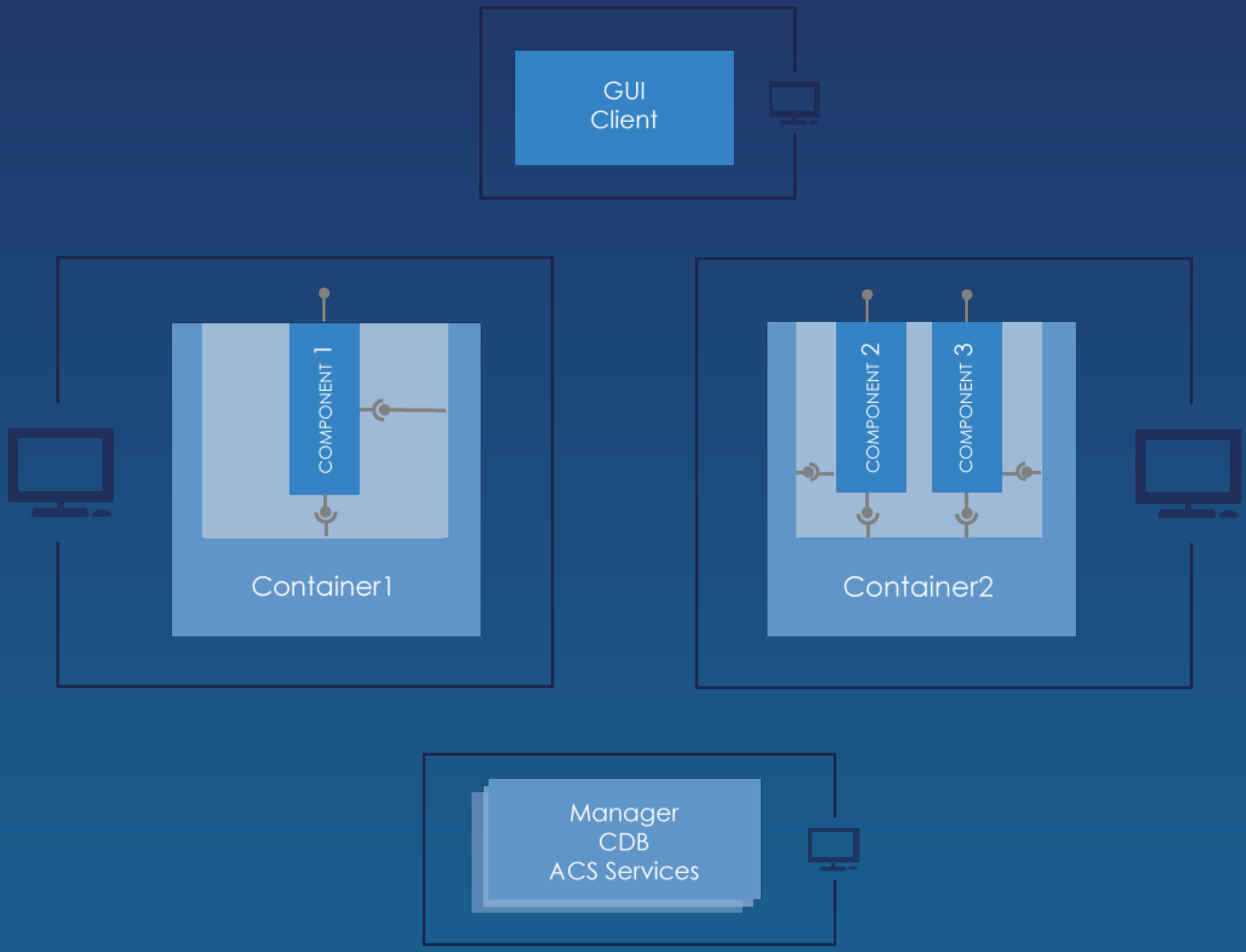
Deployment


GUI
Client
COMPONENT 1
Container1
COMPONENT 2
COMPONENT 3
Container2
Manager
CDB
ACS Services

Deployment


GUI
Client
COMPONENT 1
Container1
COMPONENT 2
COMPONENT 3
Container2
Manager
CDB
ACS Services

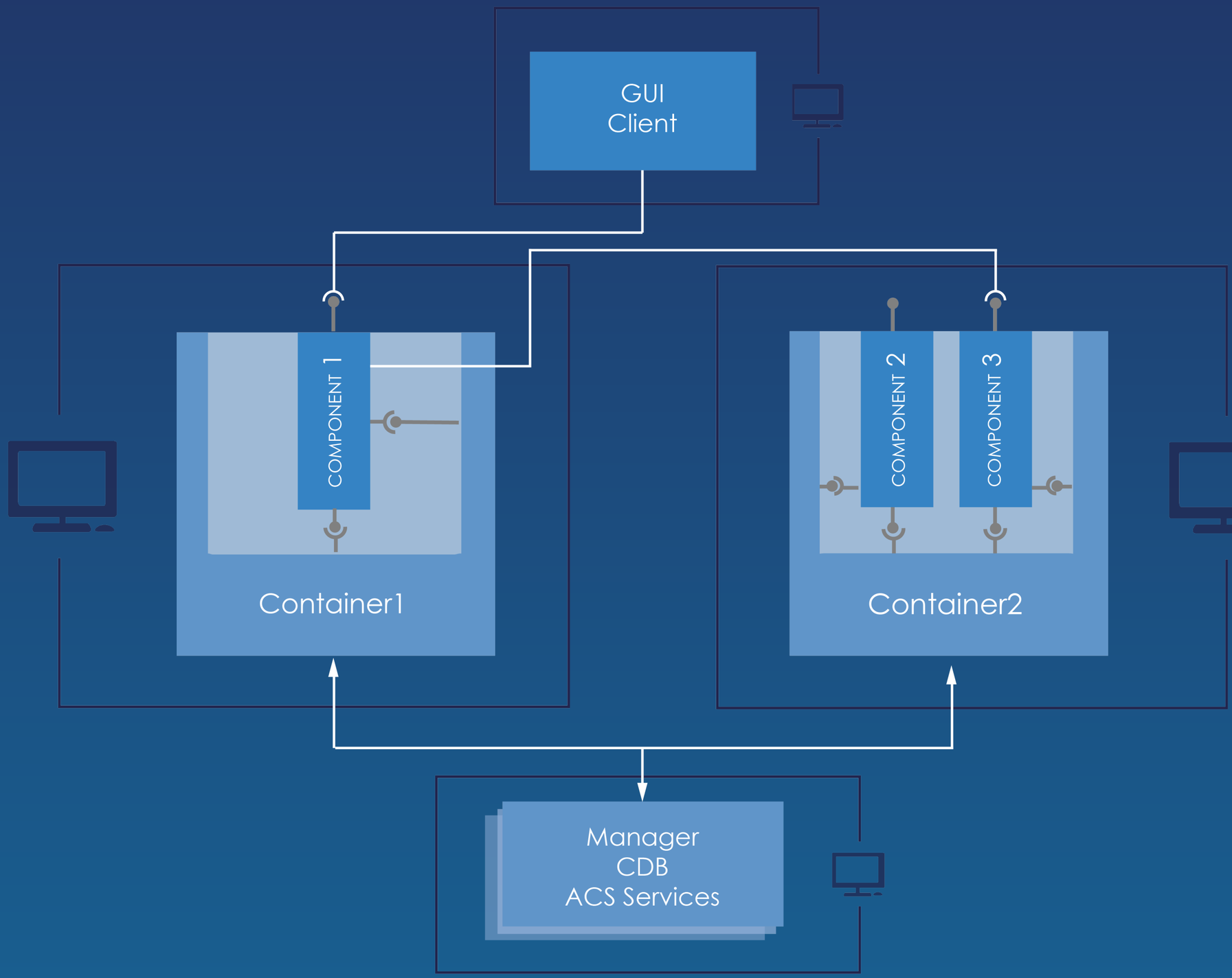
Deployment


GUI
Client
COMPONENT 1
Container1
COMPONENT 2
COMPONENT 3
Container2
Manager
CDB
ACS Services

Runtime

- ACS containers start and stop components (lifecycle management) as needed.
- Containers provide components and clients with references to other components.
- The Manager is the central intelligence point that keeps the system together. Components never see it directly.
- Manager, CDB, and other services, are started with the “acsStart” command or with the ACS command center GUI.

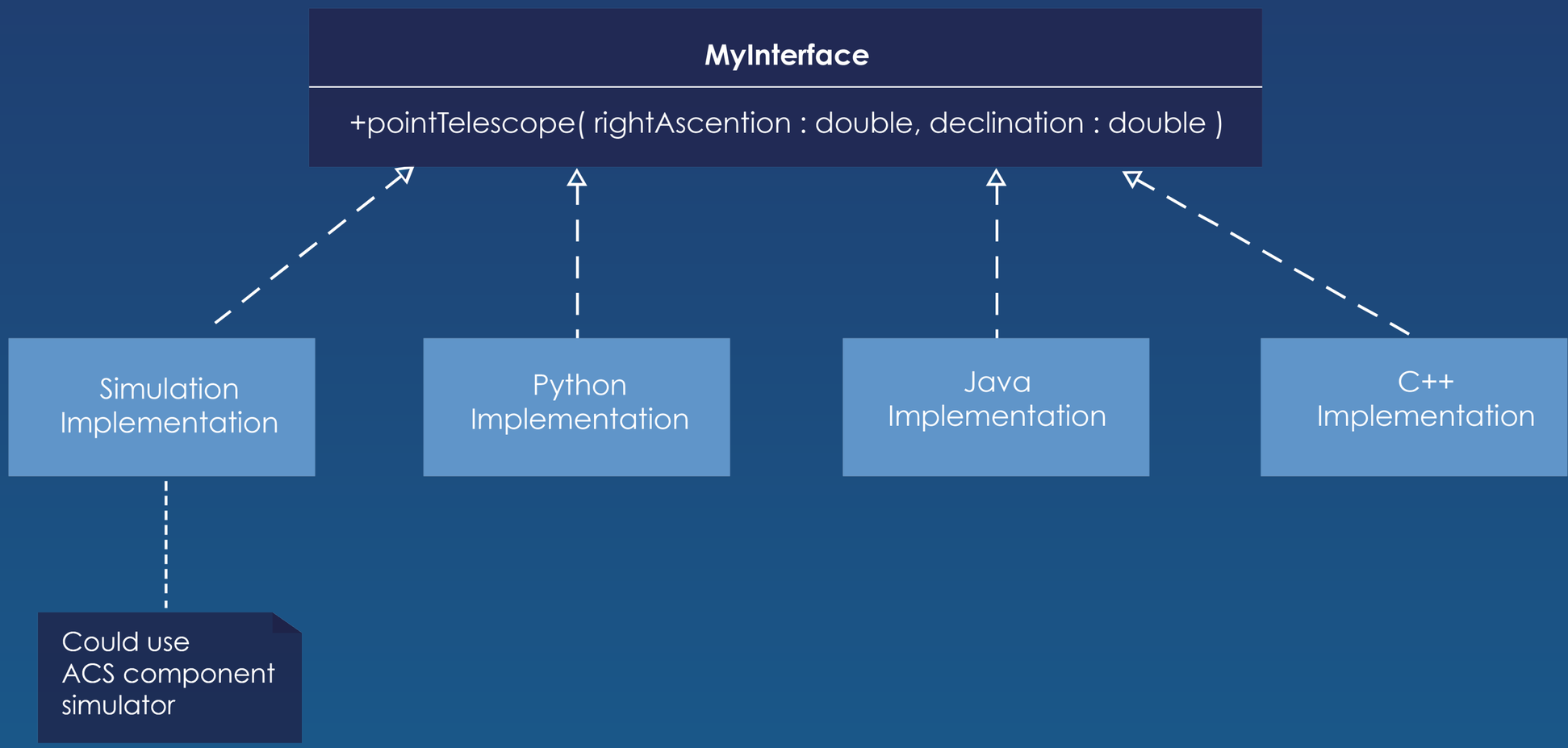
Runtime


GUI
Client
COMPONENT 1
COMPONENT 2
COMPONENT 3
Container1
Container2
Manager
CDB
ACS Services

Interfaces versus Implementations

- First step: Identify objects
  - Mount
  - Camera
  - Telescope
  - Observation
  - Exposure
- Second step: Define interfaces
  - Implementation comes later and is independent of interface
  - Deployment is also independent of interface definitions
  - Interfaces shall be kept as stable as possible, but it must be possible to have them evolve when needed.
  - A formal interface definition language is needed

One Interface, many implementations



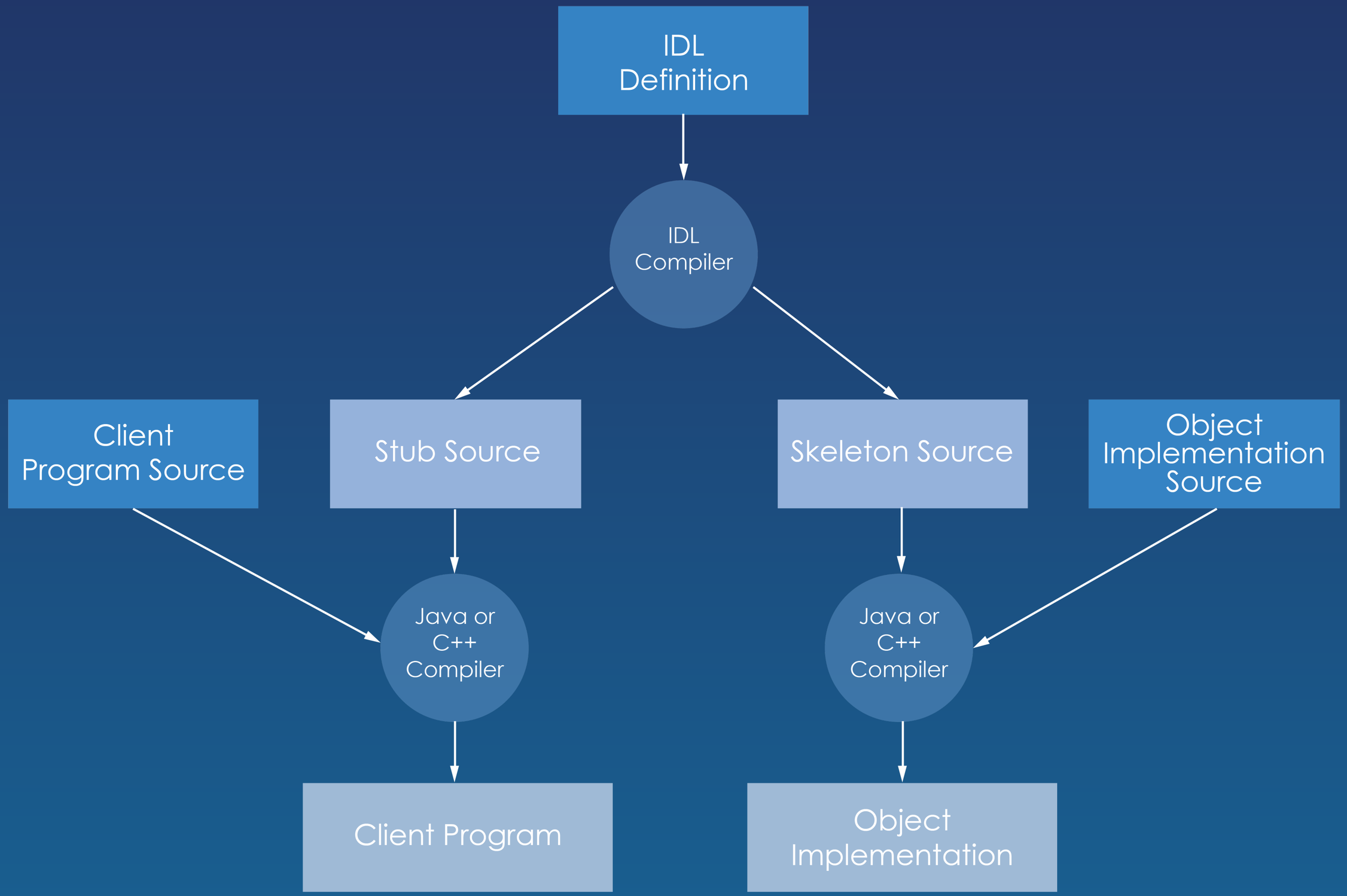
Interface Definition Language (IDL)

- ✧ CORBA and DDS both use the same formal interface definition language: IDL

  IDL forms a ‘contract’ between client and servant or publisher and subscriber

- ✧ IDL reconciles diverse object models and programming languages
- ✧ Imposes the same object model on all supported languages
- ✧ Programming language independent means of describing data types and object interfaces
  - ✧ purely descriptive - no procedural components
  - ✧ provides abstraction from implementation
  - ✧ allows multiple language bindings to be defined
- ✧ A way to integrate and share objects from different object models and languages

Development Process



Questions?

Acknowledgements
ACS presentations were originally developed by the ALMA Common Software development team and has been used in many instances of training courses since 2004. Main contributors are (listed in alphabetical order): Jorge Avarias, Alessandro Caproni, Gianluca Chiozzi, Jorge Ibsen, Thomas Jürgens, Matias Mora, Joseph Schwarz, Heiko Sommer.

The Atacama Large Millimeter/submillimeter Array (ALMA), an international astronomy facility, is a partnership of Europe, North America and East Asia in cooperation with the Republic of Chile. ALMA is funded in Europe by the European Organization for Astronomical Research in the Southern Hemisphere (ESO), in North America by the U.S. National Science Foundation (NSF) in cooperation with the National Research Council of Canada (NRC) and the National Science Council of Taiwan (NSC) and in East Asia by the National Institutes of Natural Sciences (NINS) of Japan in cooperation with the Academia Sinica (AS) in Taiwan. ALMA construction and operations are led on behalf of Europe by ESO, on behalf of North America by the National Radio Astronomy Observatory (NRAO), which is managed by Associated Universities, Inc. (AUI) and on behalf of East Asia by the National Astronomical Observatory of Japan (NAOJ). The Joint ALMA Observatory (JAO) provides the unified leadership and management of the construction, commissioning and operation of ALMA.