



A Sequencer for the LHC Era (a technical presentation)

V. Baggiolini

R. Gorbonosov, D. Khasbulatov, R. Alemany, M. Lamont

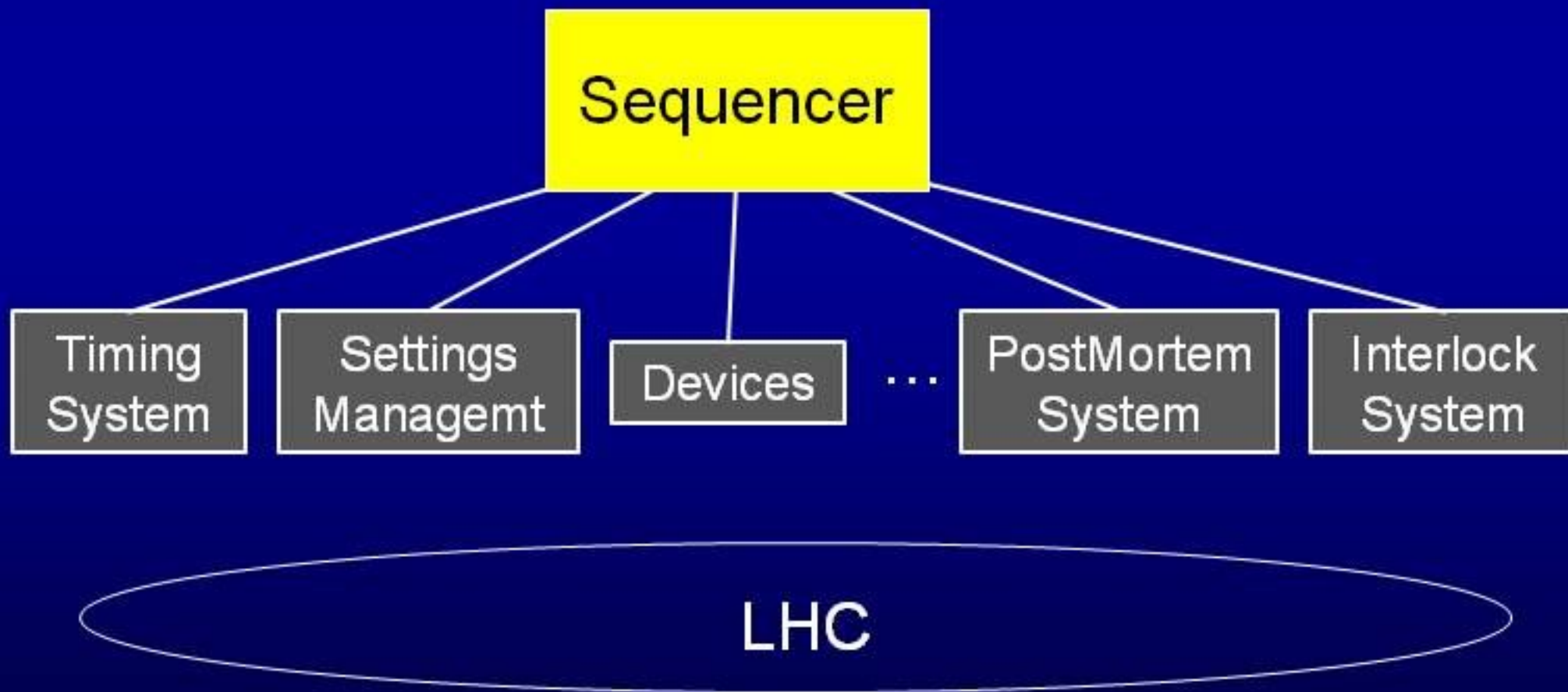
Controls and Operations Groups
Beams Department

CERN



What is the Sequencer?

- A tool that helps the operators and physicists commission and operate the LHC
 - Executes the countless tasks needed to produce beam
 - Interfaces with most sub-systems of LHC controls



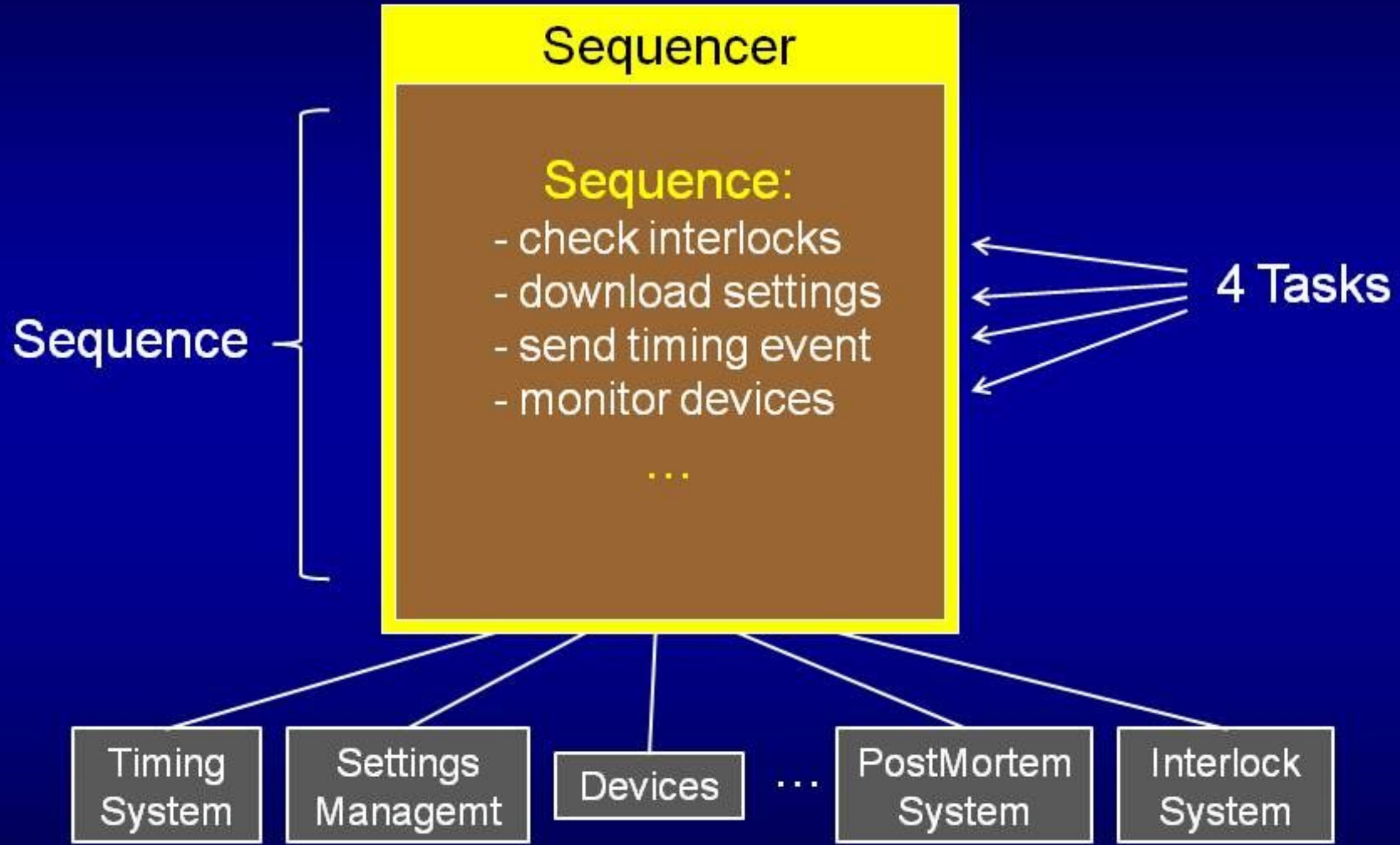


An Operational Tool

- Sequencer has been **in operations since early 2007**
- **LHC hardware commissioning**
 - 60 sequences (~ 10'000 executions to test the whole LHC)
- **LHC beam commissioning**
 - 400 sequences and sub-sequences developed so far by OP
- **A vital tool for operations from the start**
- **A modular system using advanced software technology**
- **Could be packaged as a stand-alone product**

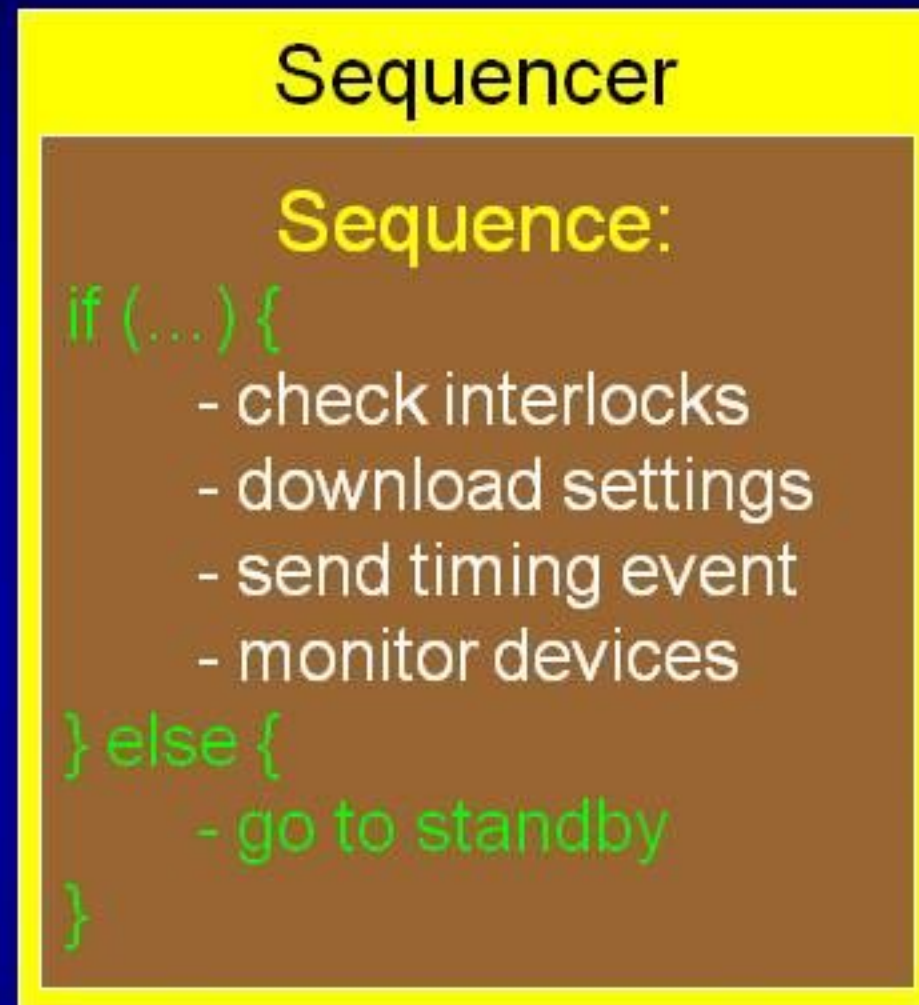


Sequencer runs Sequences with Tasks





Sequencer runs Sequences with Tasks



- if / else
- try / catch
- loops
- variables

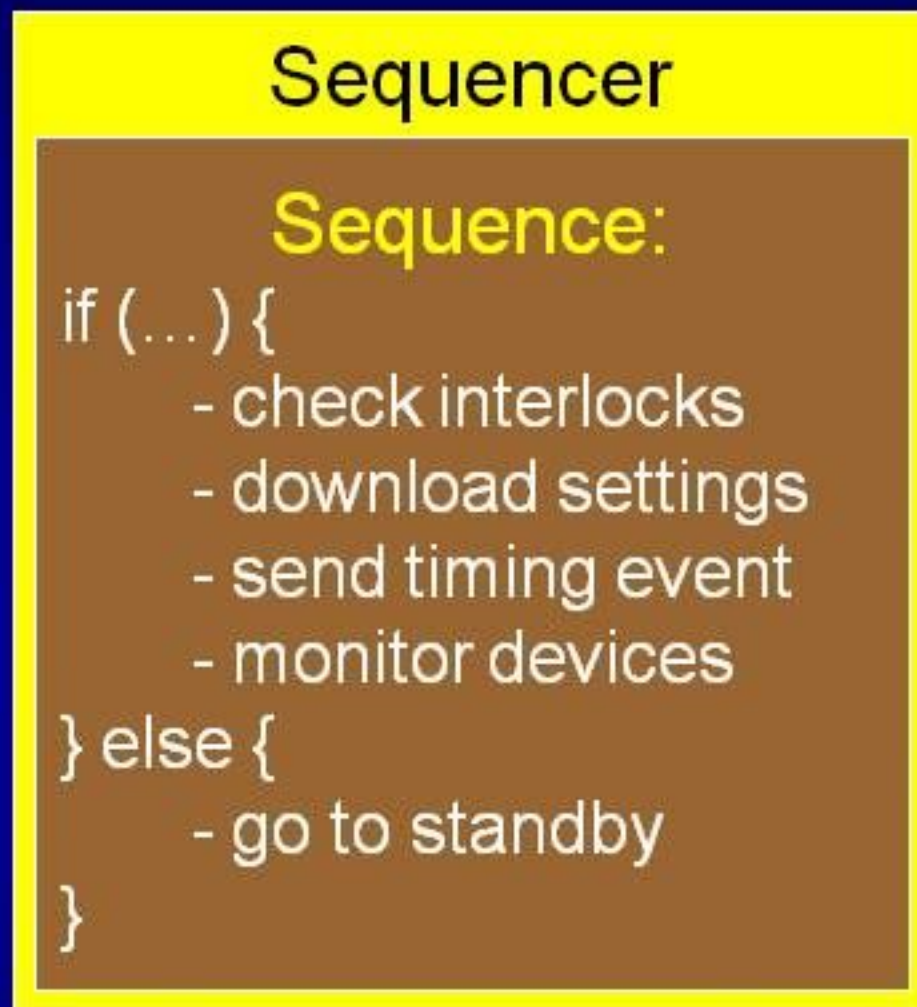




Sequencer ~ Enhanced Debugger

Debugger:

- Run-through
- Breakpoints
- Step-by-step



Enhanced features:

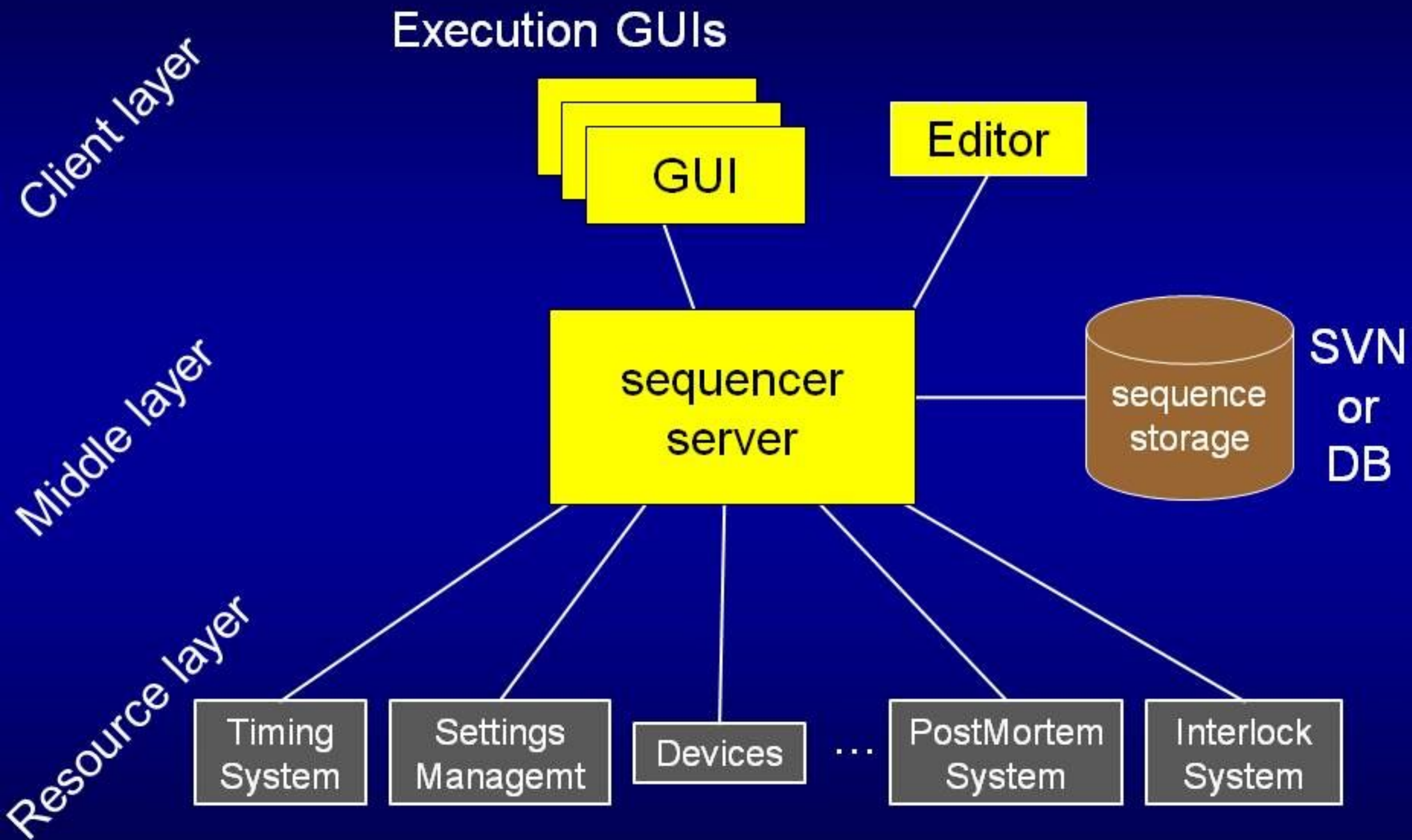
- Skipping
- Configurable on-error behavior
 - Stop (+ repeat)
 - Ignore
 - Recover
 - Abort
 - ...



Design Overview



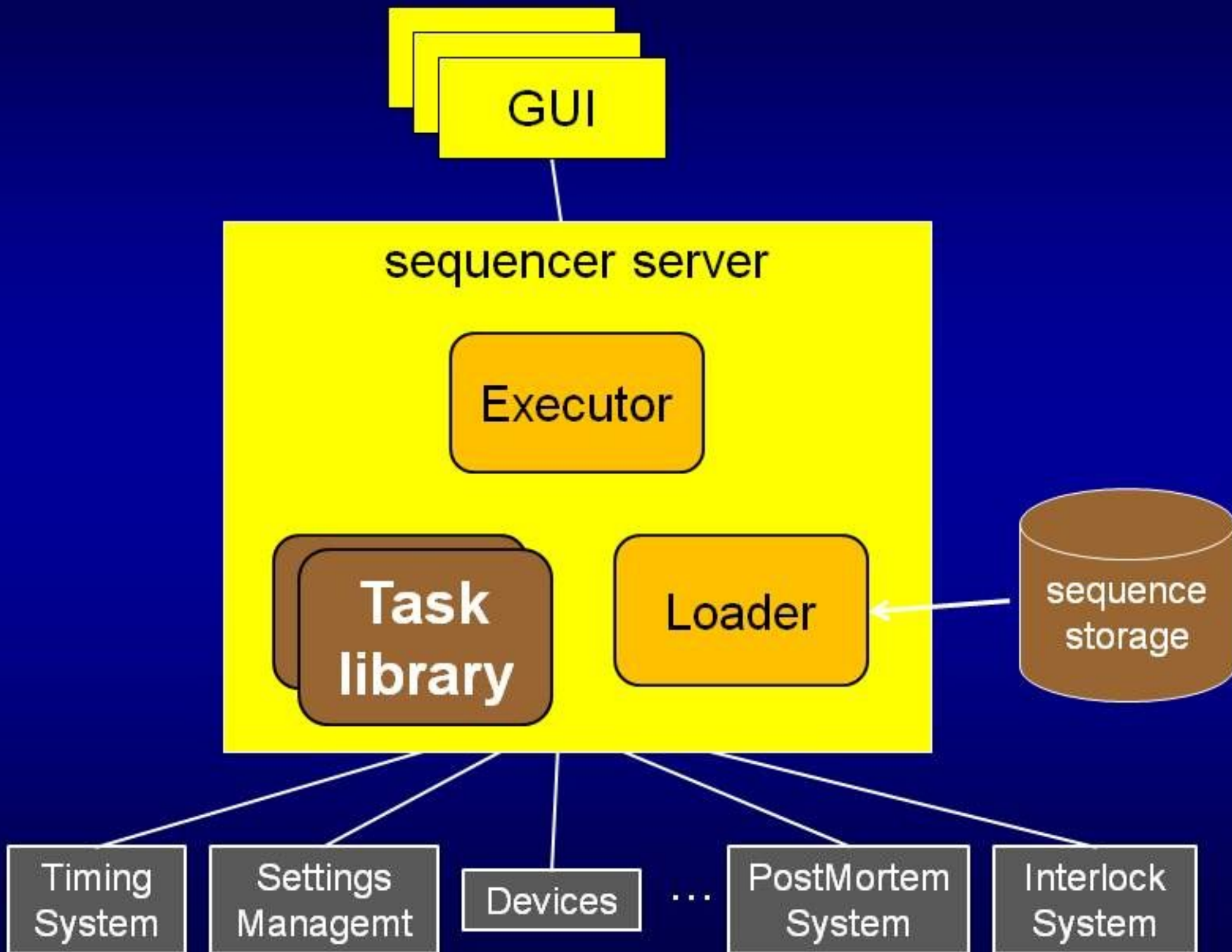
3-Layer Architecture



Based on Java and Spring Framework

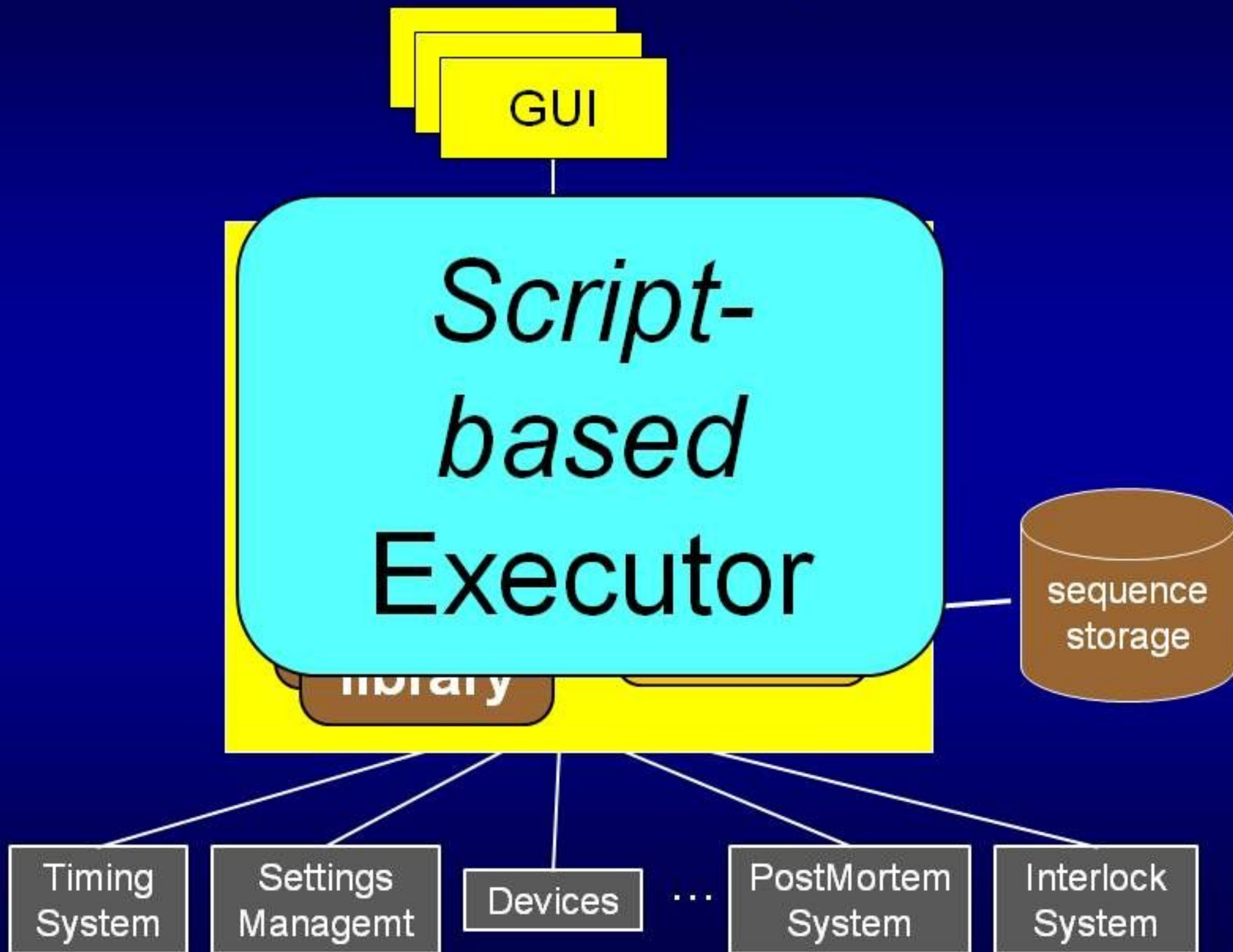


Modular Design






Modules are Exchangeable





Script-based Executor

- We did not want to implement scripting from scratch
 - Sequencer ~ Debugger → Use Debugger
 - We looked for an Script Interpreter with Debugger
- Found **Pnuts** written by Toyokazu Tomatsu, Sun 
- **Pnuts scripting language**
 - Syntax similar to Java, just simpler
 - Language features needed for sequences (if/else, try/catch, loops)
 - Pnuts can call Java methods
- **Pnuts Interpreter/Debugger**
 - Is written in Java
 - Can be embedded into a Java process
- **Script Executor** is based on Pnuts interpreter/debugger
 - Embedded into the sequencer server
 - Debugger **callbacks are redirected to our GUIs** (e.g. "on-new-line")₁



Enhancing the Pnuts Debugger

to add **skipping** and
configurable **on-error behavior**



Sequence = Pnuts Script

(for the script-based executor)

Sequence:

```
if (...) {  
  - check interlocks  
  - download settings  
  - send timing event  
  - monitor devices  
} else {  
  - go to standby  
}
```

Tasks

=

Pnuts Script:

```
if (...) {  
  checkInterlocks()  
  downloadSettings()  
  sendTimingEvent()  
  monitorDevices()  
} else {  
  goToStandby()  
}
```

=

Calls to **Java methods**
contained in Task Libraries



Steps of Execution: (1) Parsing

Pnuts
Interpreter

Tree-shaped data structure

```
Pnuts Script:  
if (...) {  
  checkInterlocks()  
  downloadSettings()  
  sendTimingEvent()  
  monitorDevices()  
} else {  
  errorRecovery()  
}
```

Parsing

Root

If-Block

MethodCall

MethodCall

MethodCall

MethodCall

Else-Block

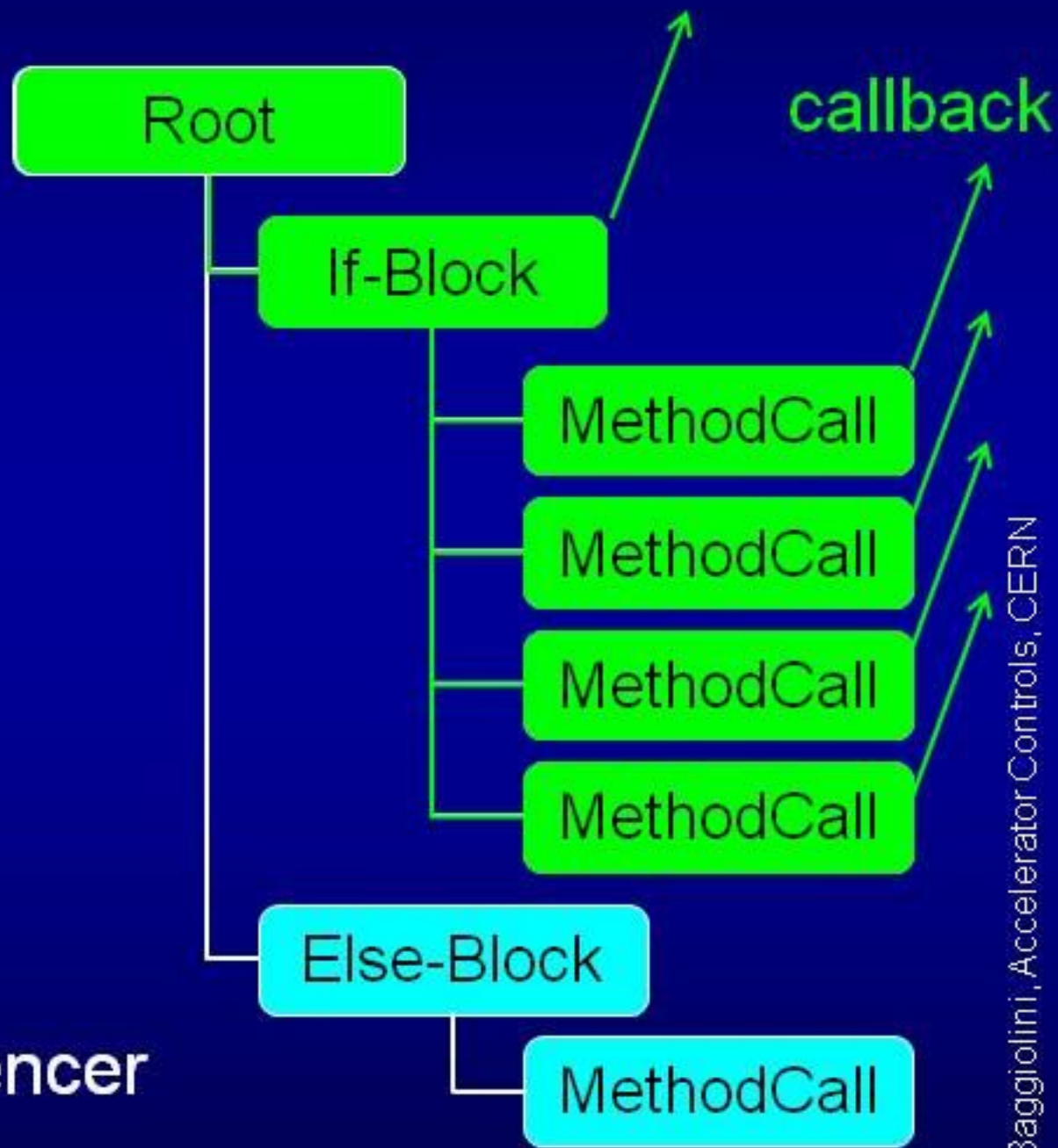
MethodCall

“Abstract Syntax Tree”



(2) Executing the Abstract Syntax Tree

Pnuts Interpreter

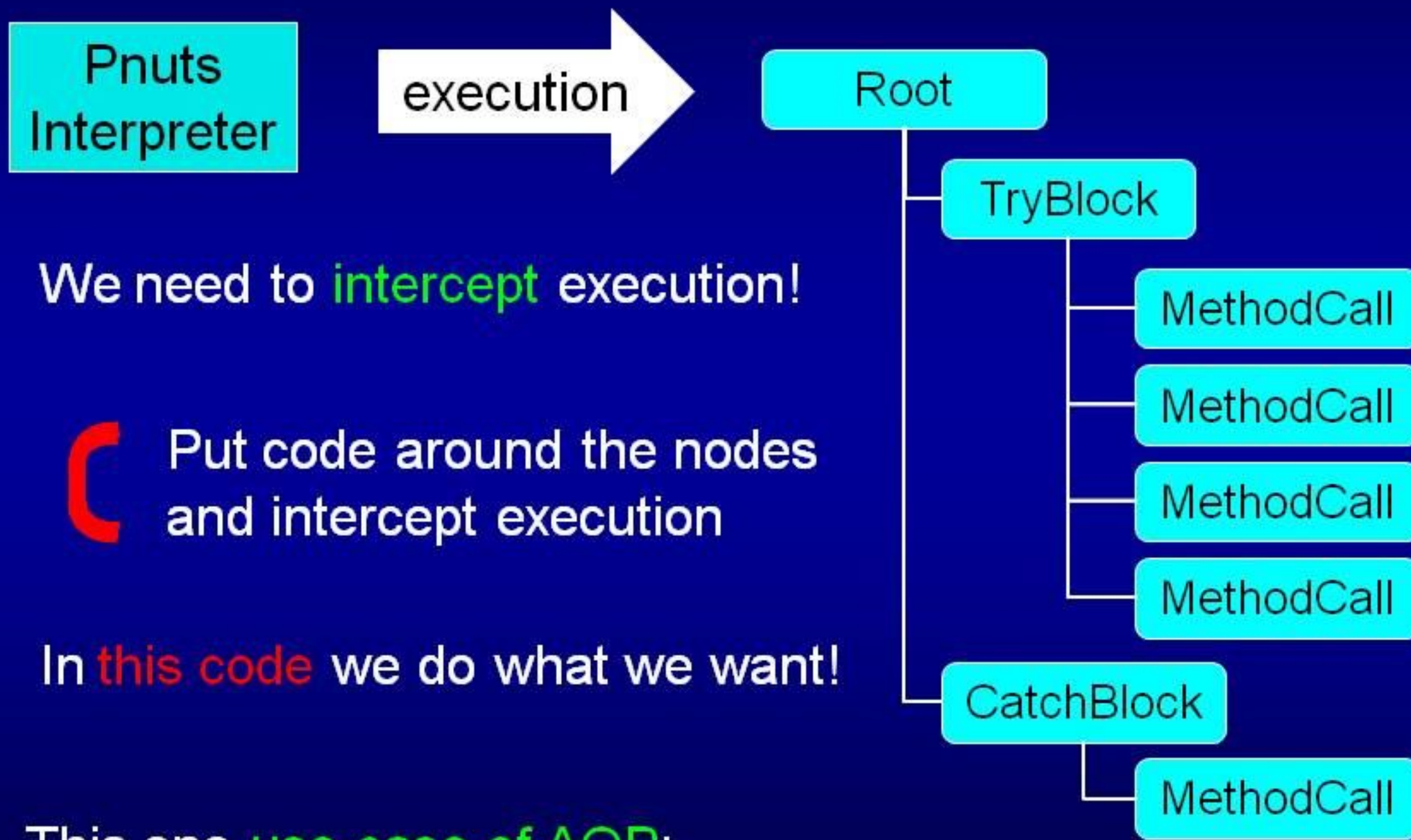


- Difficulty to implement **enhanced** sequencer features
 - Once the execution runs, it is **out of our control**
 - Callback happens only **after** execution of a node

- To implement **enhanced** sequencer features (e.g. skipping), we need to intervene **before** the node is executed!



Aspect-Oriented Programming (AOP)



We need to **intercept** execution!

C Put code around the nodes and intercept execution

In **this code** we do what we want!

This one **use-case** of AOP:

Add a **new aspect** (e.g. skipping) to an existing system



Why Aspect-Oriented Programming?

Why not **modify the Pnuts source code**?

1. We **might not have source code**
2. We **don't want to maintain 3rd party source code**

We used **AspectJ** (implementation of AOP for Java)

- The **(** code can be **inserted into an existing binary**
(e.g. into the Pnuts interpreter)



Summary

- The LHC sequencer is **operational since 2007** and a **vital tool** for operations
- **Aspect-Oriented Programming** is something to be aware of

Aspect-Oriented Programming

- A programming paradigm
- AOP vs Object-Oriented Programming (OO)
 - Not a replacement for OO but a complement
 - For things that cannot be handled well in OO
- For functionality that is typically scattered over a whole software source base, e.g.
 - Security checks (in all public methods entering a server)
 - Caching (in all methods that retrieve data over the network)
 - DB transaction management (in all methods that participate in a DB transaction)
- AOP approach:
 - Write the code for security/caching/transactions once
 - Tell AOP compiler where to insert it into the code base
- Exists for most languages

Sequences represented in Java

- We write sequences in Java, then translate to Pnuts
 - Well-known language ✓
 - Good development tools ✓
 - Compilation finds many possible mistakes ✓
- Translation based on **Abstract Syntax Tree**
 1. Parse Java Sequence into Abstract Syntax Tree
 2. Check AST for some rules
 3. Convert from AST to Pnuts source code
 - Implemented with Java Compiler Compiler (JavaCC)

A Snippet of a Sequence

/// Simulate powering failure on the power converter

```
fgc.simulateFault();
```

/// Wait to see the fault happened on the power converter

```
circuit.checkSimulatedFault();
```

/// Verify the converter maximum performance buffers

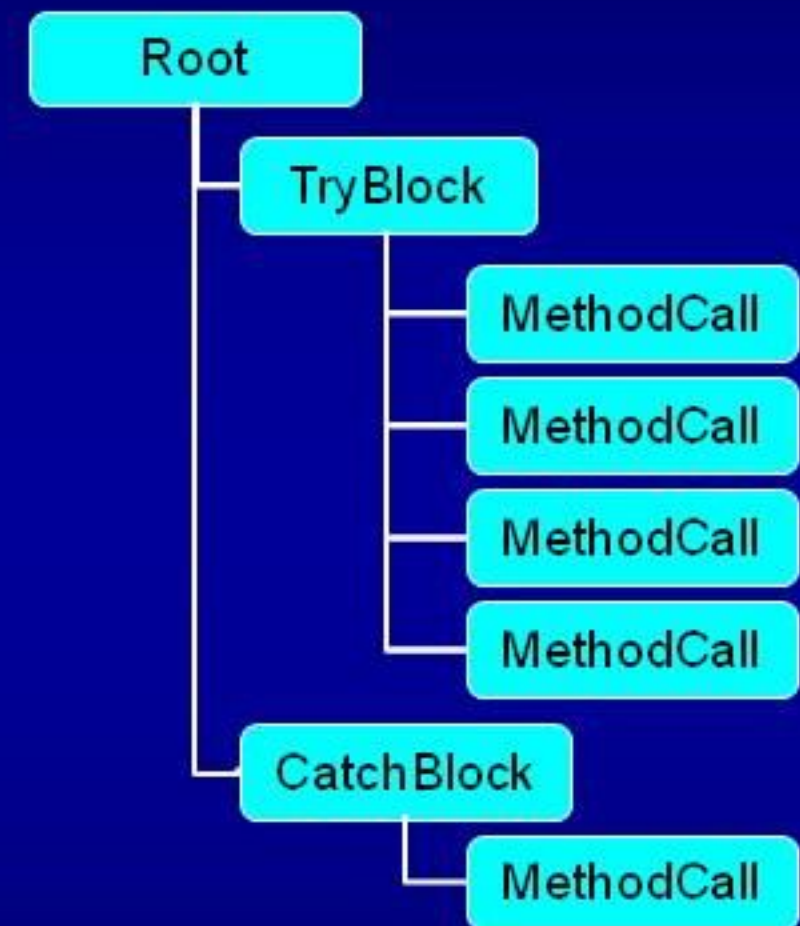
```
fgcMeasUtil.checkIErr(pcInfo.getIErrMax());
```

/// Wait until the power converter has finished sending PM data

```
fgc.waitPostMortemFinished();
```

Other usage of Abstract Syntax Tree

- Tools that “reason about” source code use an AST
 - IDE like Eclipse or Netbeans
 - Static source code checking tools
- Checking that sequences follow certain rules
- Sequence Editor Tool
 - E.g. code completion
- Automatic manipulation of sequences
 - Add boilerplate code



Parts are Developed by Different People

