The Introduction of RobotSDK

A Top-down Modular Software Development Framework
一种自顶向下的模块化软件开发框架

HMW-Alexander
何梦文

Carnegie Mellon University
卡内基梅隆大学

October 3, 2016

报告大纲

1. 前言：关于本报告
2. 什么是 RobotSDK
   - 问题提出
   - 问题总结，目标与要求
   - 关于 RobotSDK
     - 自顶向下的模块化开发理念
     - 基本概念
     - 基本架构
     - 使用方法与实例（请通过网上视频与资料深入了解）
3. RobotSDK 开源开发计划
   - RobotSDK 的发展简史
     - 北京大学时期
     - 名古屋大学时期
     - 目前现状
   - RobotSDK 目前的开发计划
   - RobotSDK 开源开发计划

Carnegie Mellon University CyLab Security and Privacy Institute

关于本报告

报告内容

- 介绍什么是模块化的软件开发理念。
- 介绍什么是自顶向下的软件开发理念。
- 介绍 RobotSDK 是如何实现自顶向下的模块化软件开发框架。
- 介绍 RobotSDK 开源开发计划。

关于本报告

报告内容

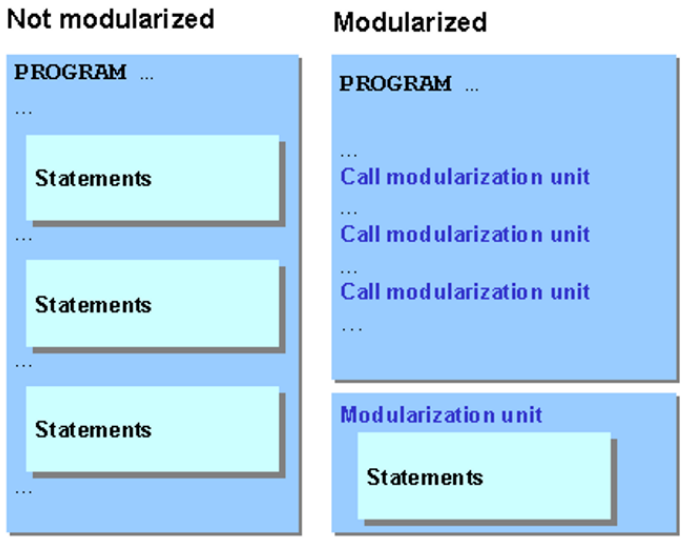
- 介绍什么是模块化的软件开发理念。
- 介绍什么是自顶向下的软件开发理念。
- 介绍 RobotSDK 是如何实现自顶向下的模块化软件开发框架。
- 介绍 RobotSDK 开源开发计划。

注意事项

- 自顶向下的模块化软件开发理念是本报告的关键。
- 而 RobotSDK 本身并不是本报告的重点：
  - RobotSDK 只是实现了自顶向下的模块化软件开发理念。但其目前还是实验室的试验品。
  - 虽然经过多次技术升级，但是毕竟个人能力有限，所以从技术层面上是无法与成熟的软件系统开发平台 (如 ROS) 相比的。
- 报告的最终目的是希望推广自顶向下的模块化软件开发理念，并希望大家能够参与到 RobotSDK 的开源开发计划，从而能够在技术和用户群方面可以和 ROS 相当。

问题提出

当刚学会 C/C++ 编程的时候，我们是如何完成大作业的呢？

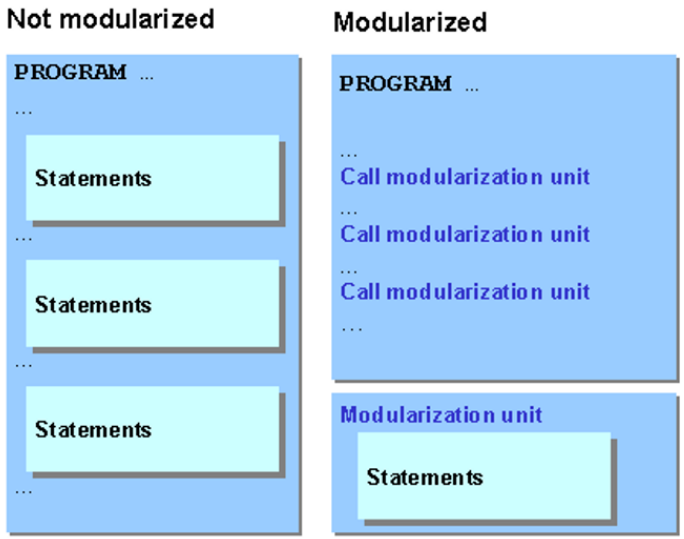


Carnegie Mellon University CyLab Security and Privacy Institute

问题提出

当刚学会 C/C++ 编程的时候，我们是如何完成大作业的呢？

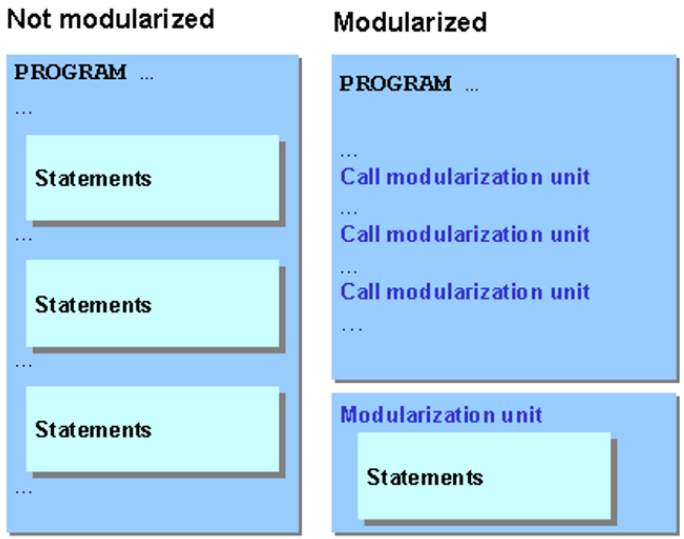
- 我们往往会根据题目的要求，直接写出一个浑然一体的程序。
- 结果往往是给一个新的大作业，就需要几乎重写一个新的程序。



问题提出

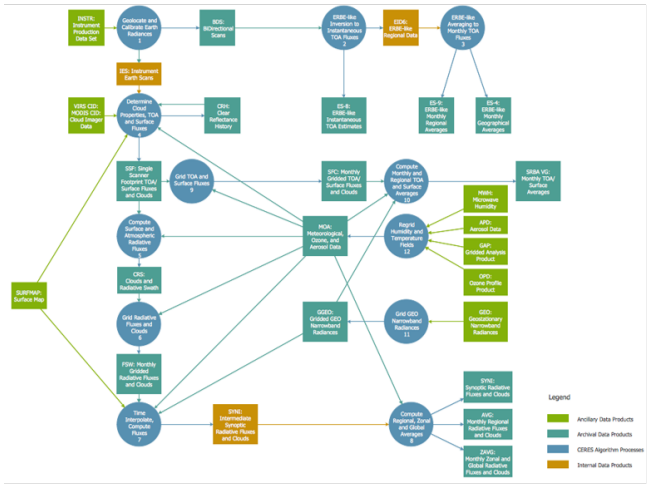
当刚学会 C/C++ 编程的时候，我们是如何完成大作业的呢？

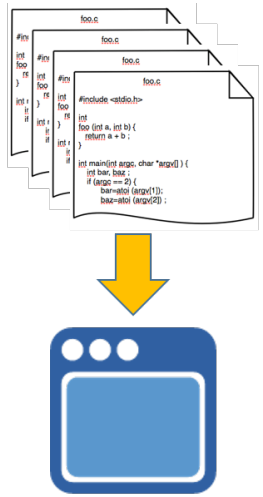
- 我们往往会根据题目的要求，直接写出一个浑然一体的程序。
- 结果往往是给一个新的大作业，就需要几乎重写一个新的程序。
- 这样的软件开发方式是低效率的，而且也不适合开发大的软件系统。
- 所以我们知道了使用“模块化”的方式提高软件编写效率和代码复用率。

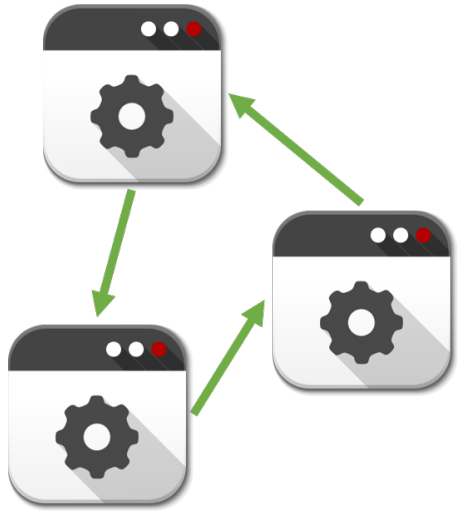


问题提出

当有了基本的软件开发知识和经验，我们是如何组队完成软件开发的呢？



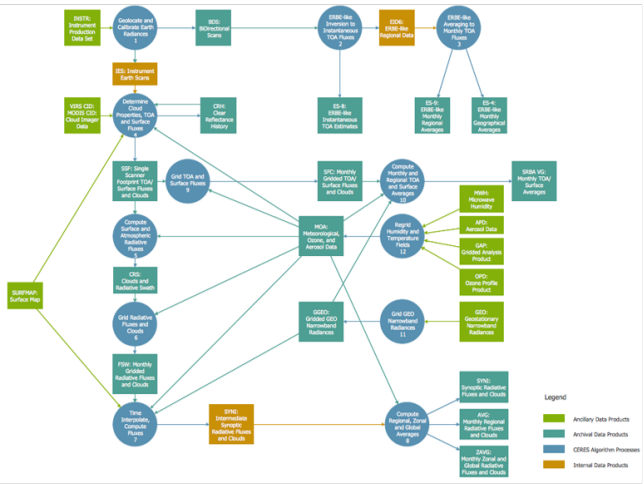


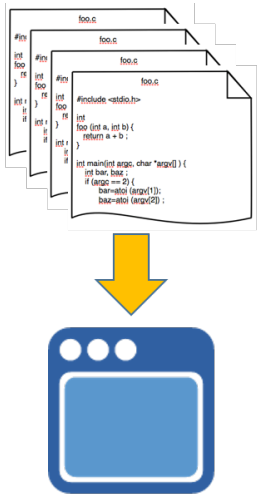


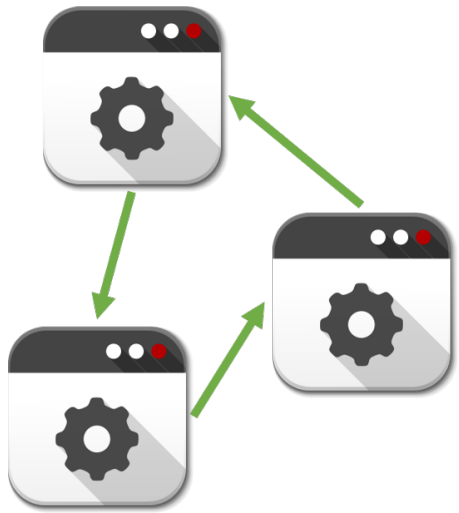
问题提出

当有了基本的软件开发知识和经验，我们是如何组队完成软件开发的呢？

- 首先我们需要讨论软件的模块化设计问题，最直观的方法就是用“图模型”：
  - 一个节点表示一个模块，定义输入输出和功能
  - 一条有向边表示一个数据通路，定义起止节点和数据传输类型



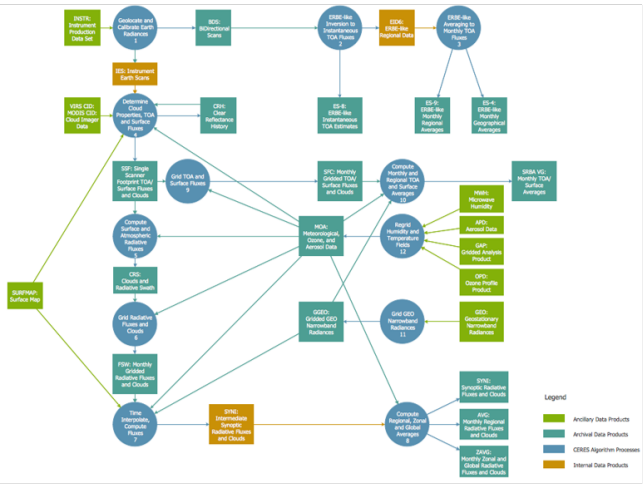


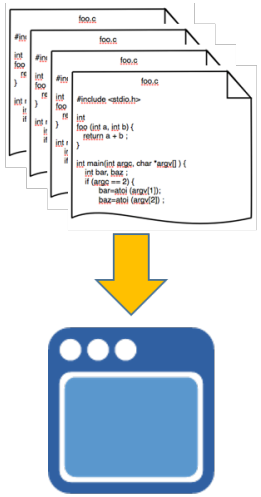


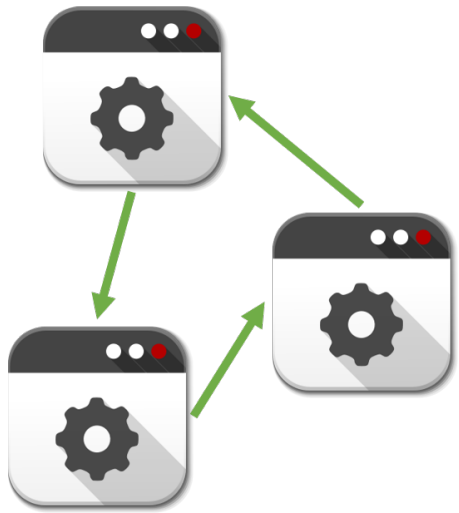
问题提出

当有了基本的软件开发知识和经验，我们是如何组队完成软件开发的呢？

- 首先我们需要讨论软件的模块化设计问题，最直观的方法就是用“图模型”：
  - 一个节点表示一个模块，定义输入输出和功能
  - 一条有向边表示一个数据通路，定义起止节点和数据传输类型
- 然后我们需要根据设计的模块与通信协议开始分工合作开发，一般有三种方式：
  - 每人编写自己对应的函数体/对象代码，最后合在一起编译成一个程序（代码方式）
  - 每人编写一个程序/动态库，然后用某种通信方式把所有模块组织起来（程序方式）
  - 上述两种方式的混合方式。

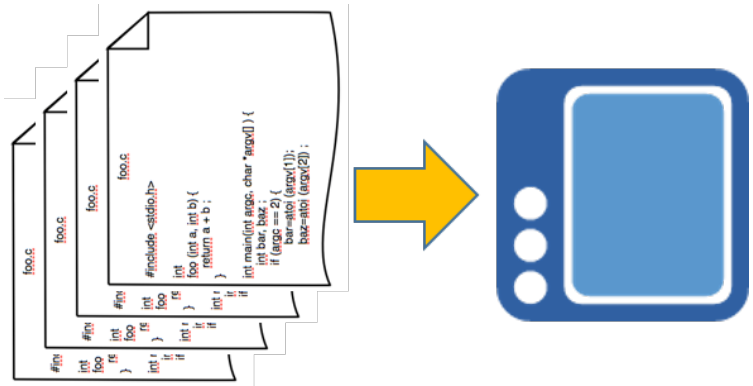






问题提出

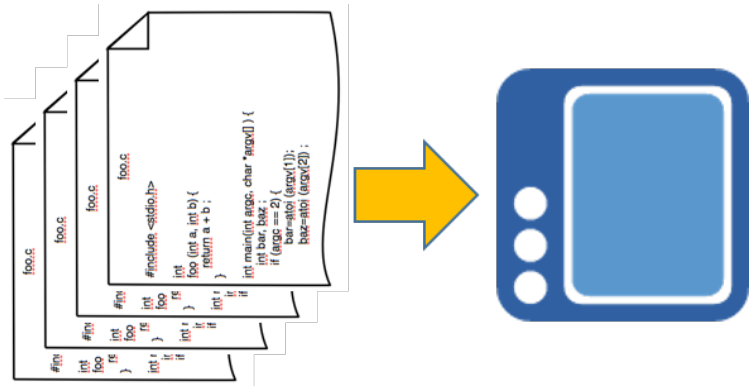
一、代码方式合作开发软件



问题提出

一、代码方式合作开发软件

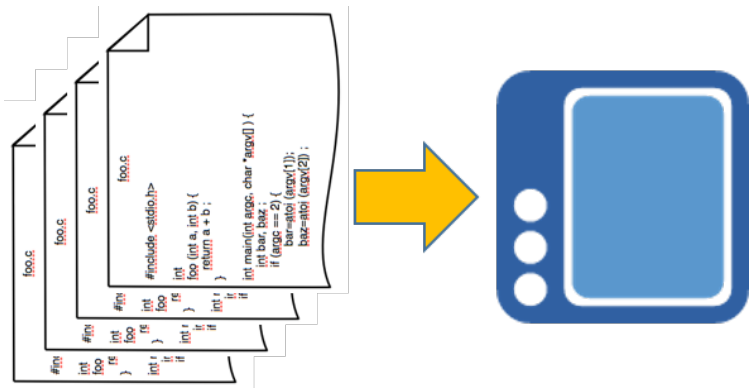
- 优点：
  - 模块的功能定义非常清晰，一般直接对应算法代码本身，不需要无关辅助功能。
  - 模块编写可以尽可能细分直到最小的函数体或对象。
  - 数据传输一般通过参数的形式在调用函数时传递，不受数据量大小影响速度。
  - 模块代码可以编译到静态链接库进而整合到最终软件，之后还可以被反复使用。



问题提出

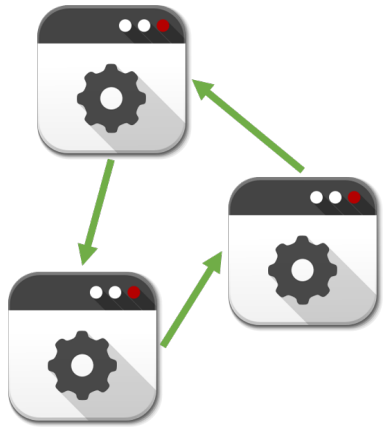
一、代码方式合作开发软件

- 优点：
  - 模块的功能定义非常清晰，一般直接对应算法代码本身，不需要无关辅助功能。
  - 模块编写可以尽可能细分直到最小的函数体或对象。
  - 数据传输一般通过参数的形式在调用函数时传递，不受数据量大小影响速度。
  - 模块代码可以编译到静态链接库进而整合到最终软件，之后还可以被反复使用。
- 缺点：
  - 模块的更新一般需要重新编译整个软件，而大型软件的编译速度通常是很慢的。
  - 模块与软件的测试一般需要等待所有相关模块编写完成并整合在一起。
  - 软件架构（图模型的程序化）与模块高度整合，往往无法在其它工程被复用。



问题提出

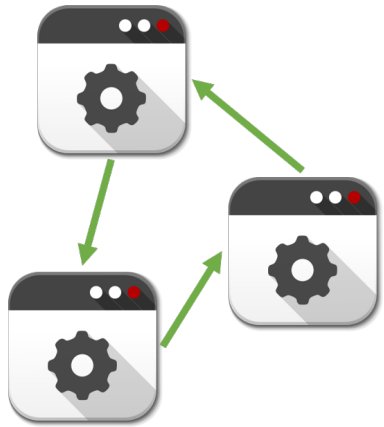
二、程序方式合作开发软件（例如 ROS）



问题提出

二、程序方式合作开发软件（例如 ROS）

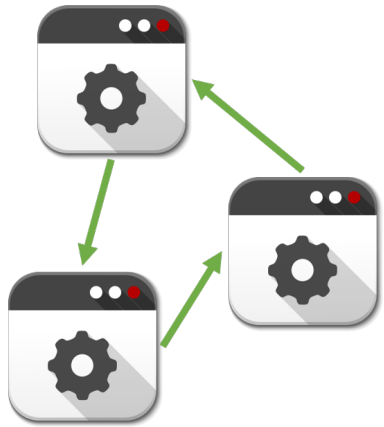
- 优点：
  - 模块的更新无需重新编译整个软件系统。
  - 程序化模块完成后可以自行做测试，因为其本身是独立可运行程序。
  - 软件架构与模块是分离的，所以可以在其它工程被复用（如 ROS Launch）。
  - 程序化/动态库化模块可以被反复使用。（如 ROS Package）



问题提出

二、程序方式合作开发软件（例如 ROS）

- 优点：
  - 模块的更新无需重新编译整个软件系统。
  - 程序化模块完成后可以自行做测试，因为其本身是独立可运行程序。
  - 软件架构与模块是分离的，所以可以在其它工程被复用（如 ROS Launch）。
  - 程序化/动态库化模块可以被反复使用。（如 ROS Package）
- 缺点：
  - 程序化模块还必须添加无关的辅助功能使其成为可运行程序并能与外界进行通讯。
  - 模块编写可以尽可能细分，但为了开发效率，一般是将相关功能集整合到一个模块中。
  - 基于网路的数据实体传输，速度会受数据量大小影响。（ROS 是基于网络的指针传输）



问题提出

两种方式的共有缺点

- 软件架构只针对于特定功能的模块集合，不能泛化到类似图模型的实现。
- 这里涉及到自顶（高层抽象）向下（底层实现）的软件开发理念，我们先讨论“泛化”概念。

问题提出

两种方式的共有缺点

- 软件架构只针对于特定功能的模块集合，不能泛化到类似图模型的实现。
- 这里涉及到自顶（高层抽象）向下（底层实现）的软件开发理念，我们先讨论“泛化”概念。

概念解释：软件架构的泛化

- 软件架构：即程序化（实例化）的图模型，其将各个功能模块实体组织在一起发挥软件的功能。
- 泛化：我们通过下面一个简单例子来解释这个概念。

问题提出

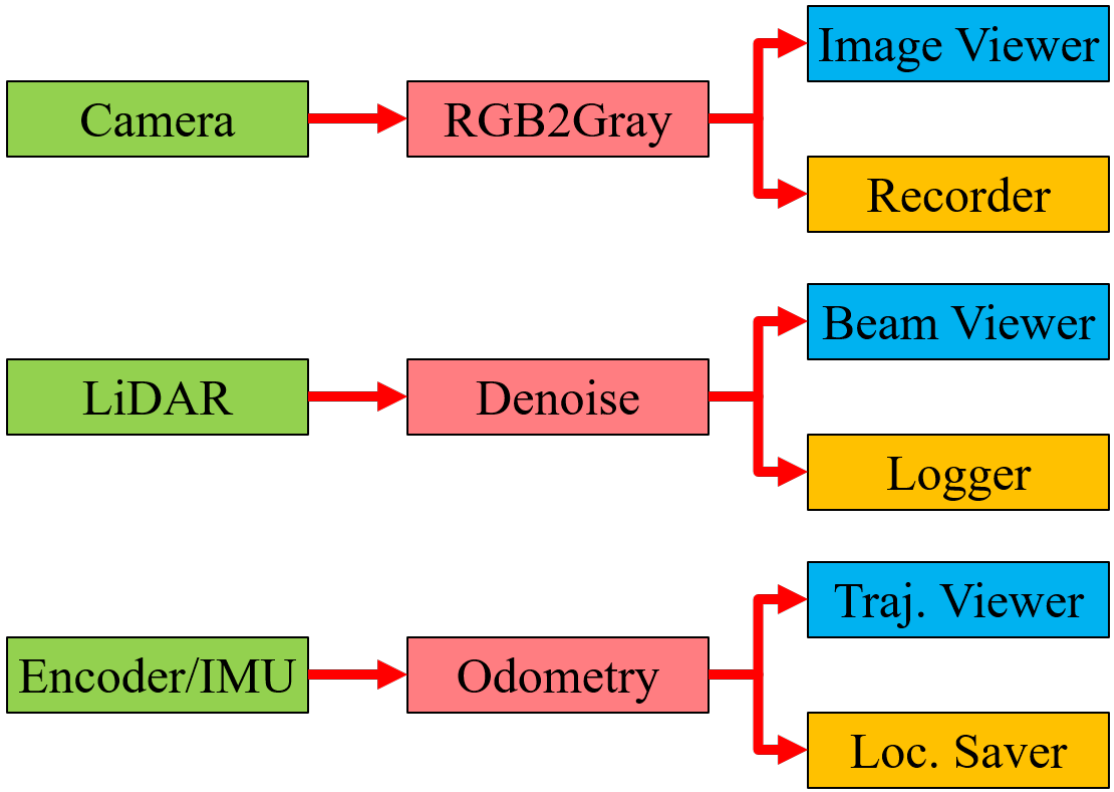
机器人数据采集可视化存储软件系统开发实例

- 现在有一个机器人装有 Camera、LiDAR、Encoder/IMU 等传感器。

问题提出

机器人数据采集可视化存储软件系统开发实例

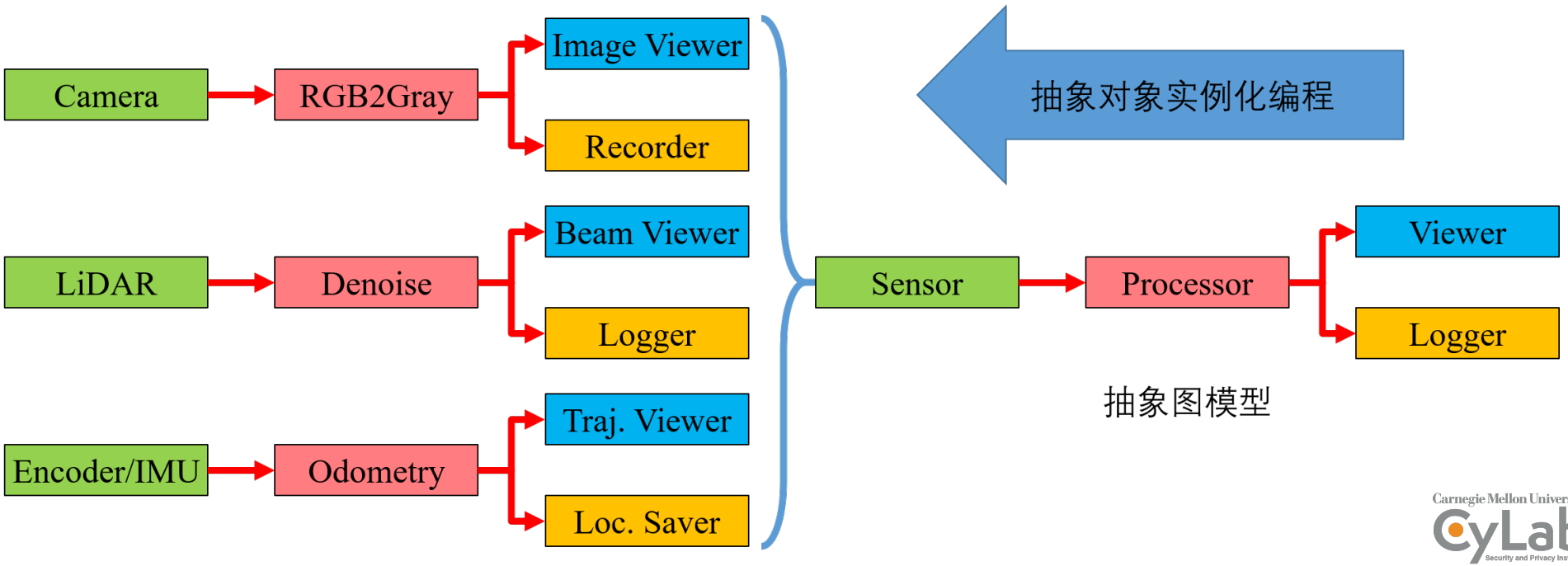
- 现在有一个机器人装有 Camera、LiDAR、Encoder/IMU 等传感器。
- 我们一般会设计出如下图模型：



问题提出

机器人数据采集可视化存储软件系统开发实例

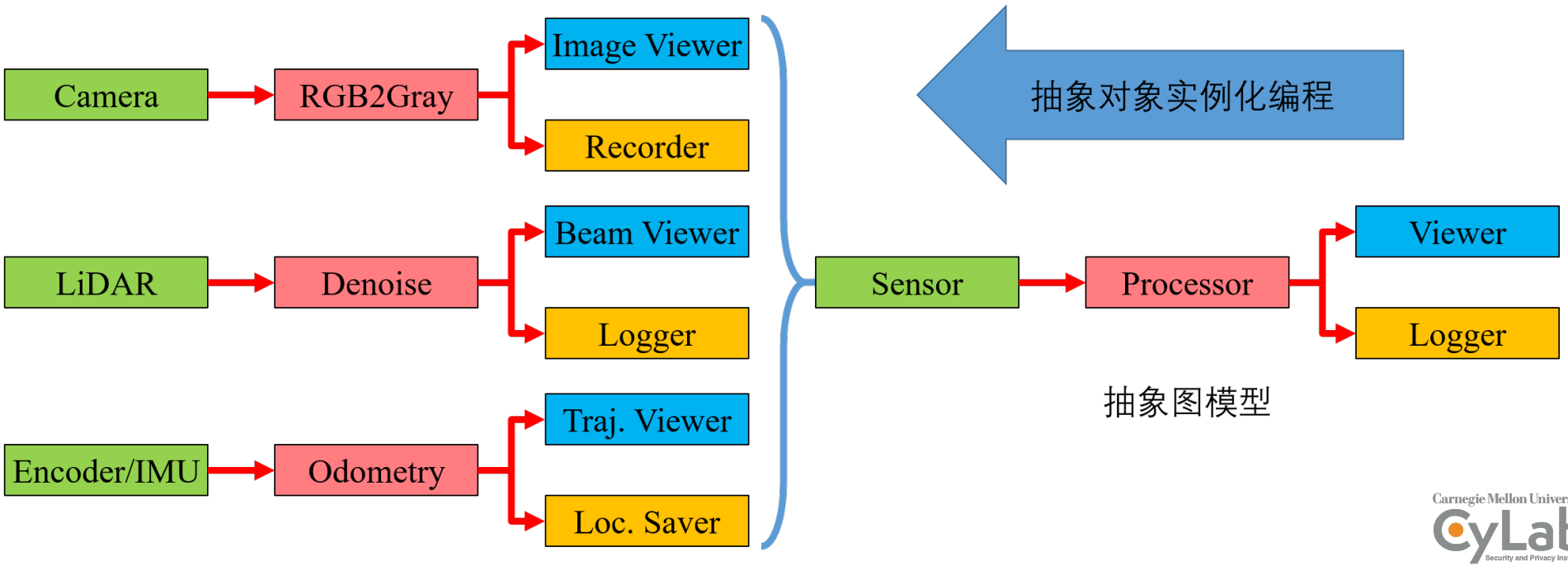
- 按照上述图模型，我们需要编写三个功能不同的软件架构，对应各传感器。



问题提出

机器人数据采集可视化存储软件系统开发实例

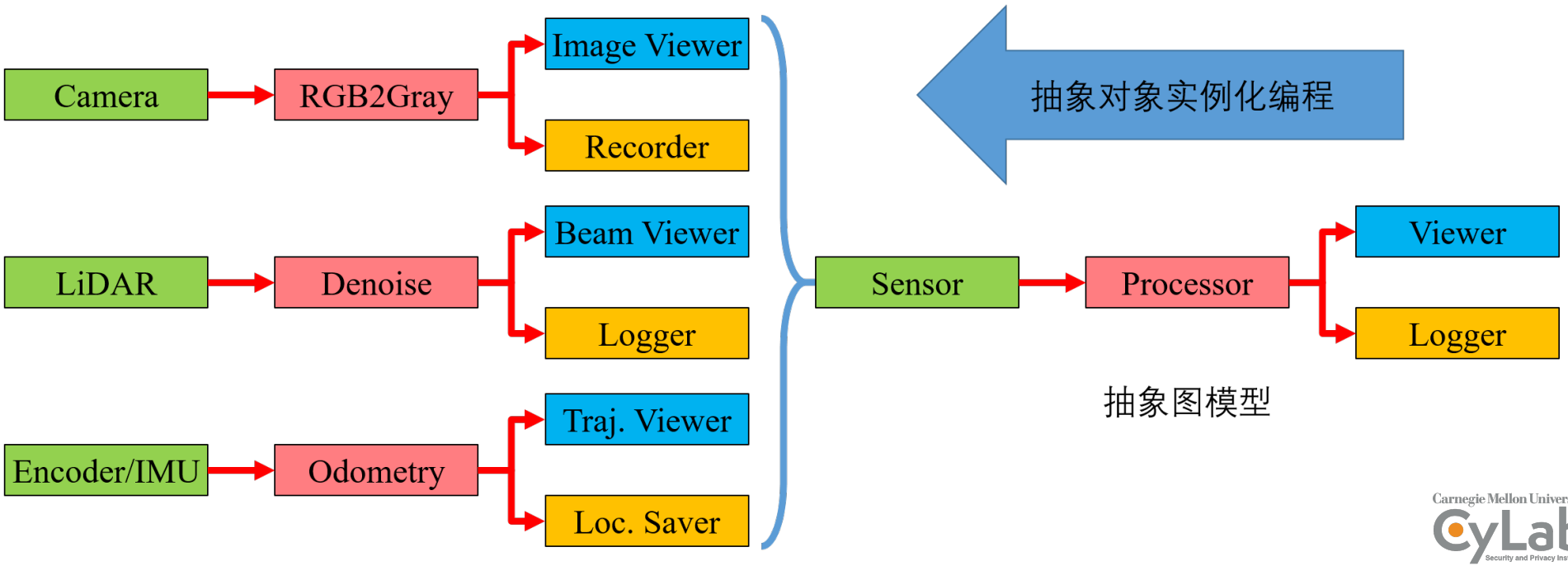
- 按照上述图模型，我们需要编写三个功能不同的软件架构，对应各传感器。
- 有经验的人会用面向对象的方式高效的编写出上述三个软件架构（代码层抽象）。
- 这很棒，因为你已经看出来了这三个软件架构具有相似的“抽象图模型”。



问题提出

机器人数据采集可视化存储软件系统开发实例

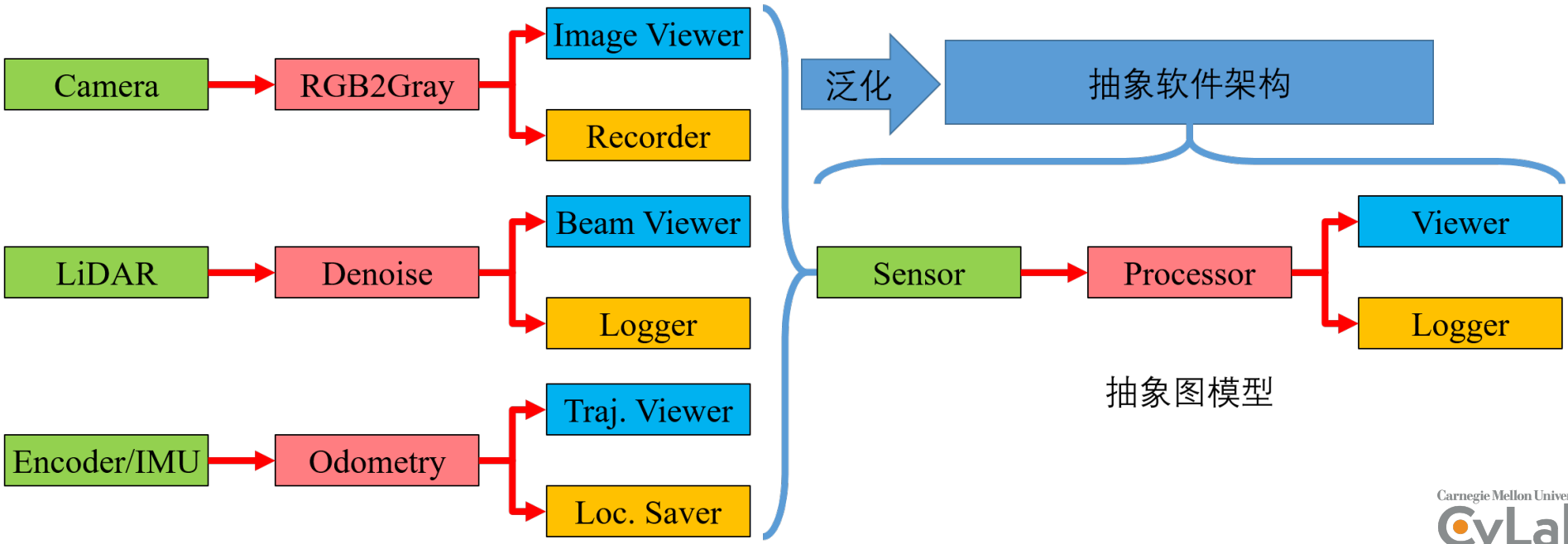
- 按照上述图模型，我们需要编写三个功能不同的软件架构，对应各传感器。
- 有经验的人会用面向对象的方式高效的编写出上述三个软件架构（代码层抽象）。
- 这很棒，因为你已经看出来了这三个软件架构具有相似的“抽象图模型”。
- 当来了新传感器，仍需要再编写相似结构的软件架构。（这个简单例子不是大问题）



问题提出

机器人数据采集可视化存储软件系统开发实例

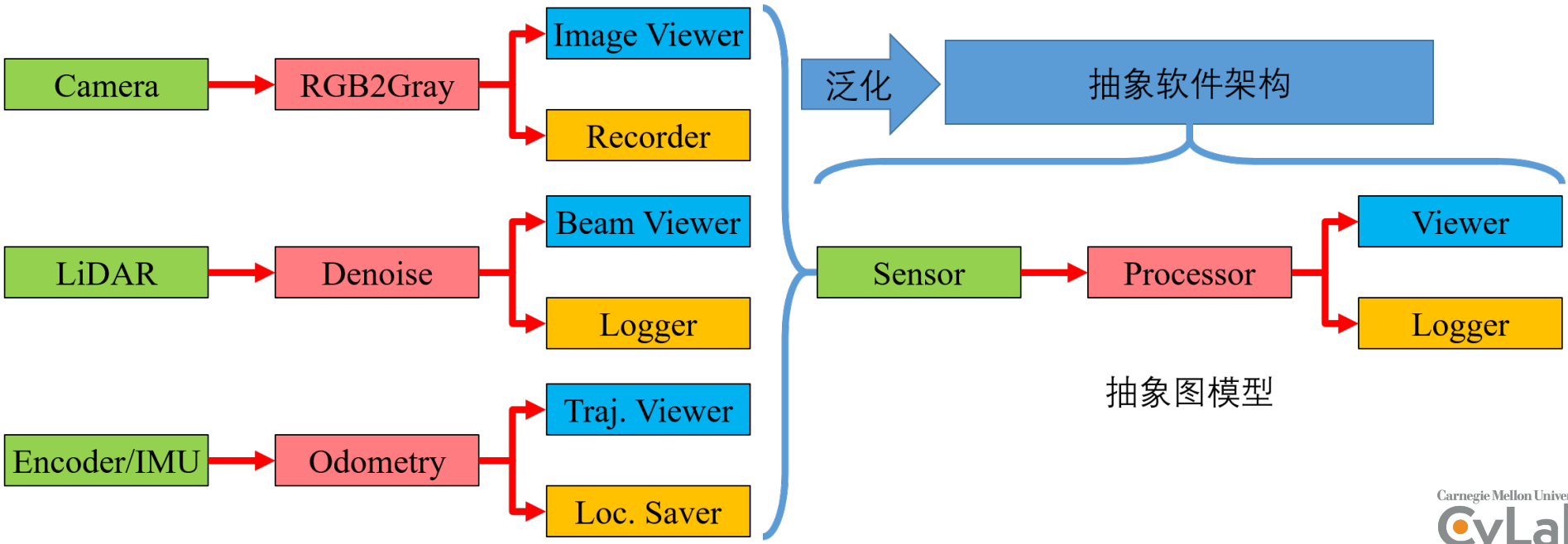
- 实际开发中往往能发现很多软件是基于相似但很复杂的“抽象图模型”。



问题提出

机器人数据采集可视化存储软件系统开发实例

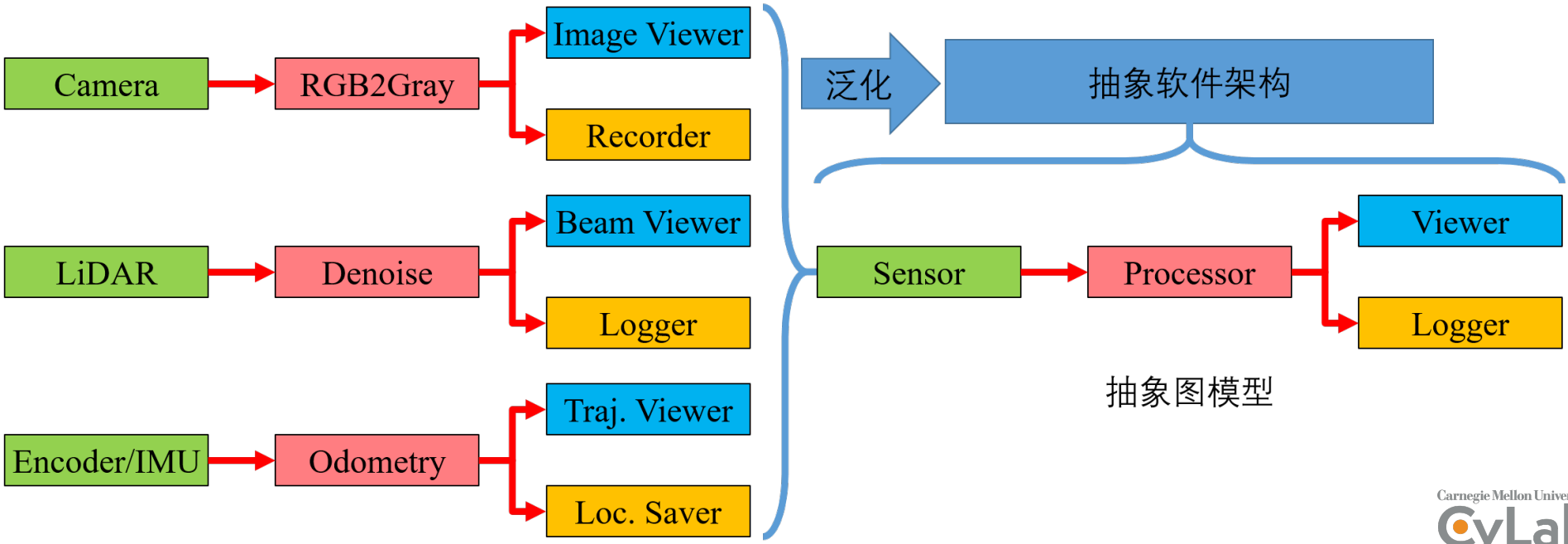
- 实际开发中往往能发现很多软件是基于相似但很复杂的“抽象图模型”。
- 这时候，代码层抽象只能方便我们设计软件，但无法减少重复编程。



问题提出

机器人数据采集可视化存储软件系统开发实例

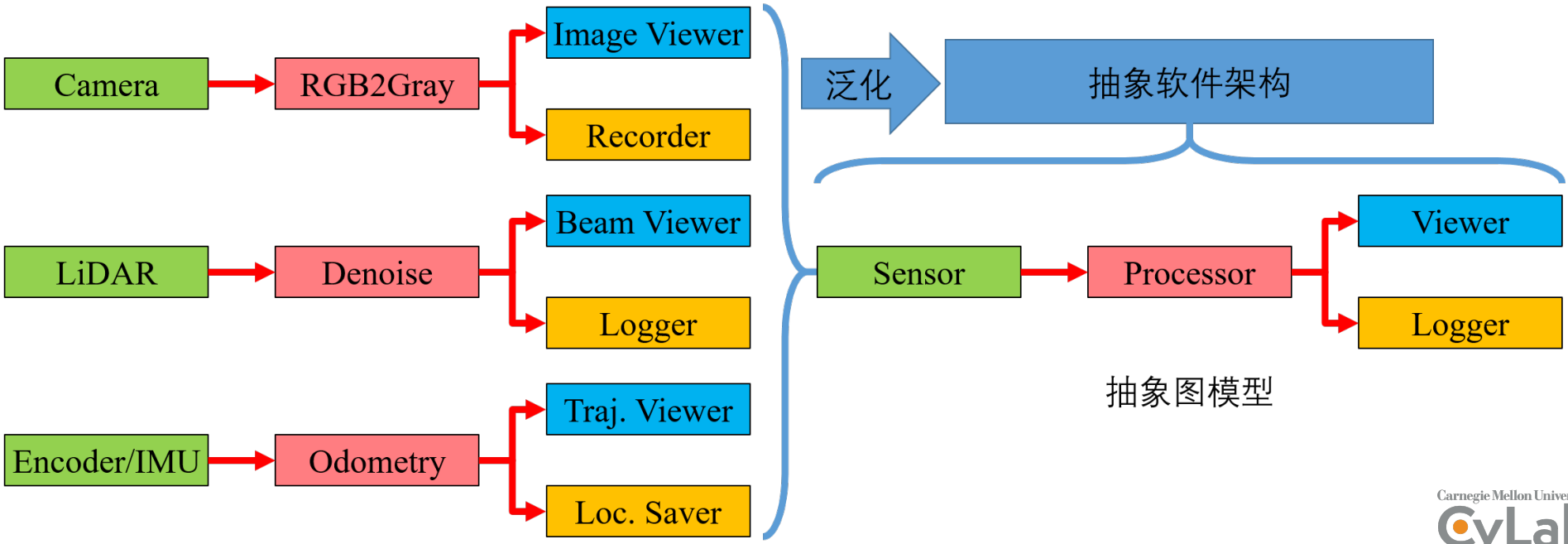
- 实际开发中往往能发现很多软件是基于相似但很复杂的“抽象图模型”。
- 这时候，代码层抽象只能方便我们设计软件，但无法减少重复编程。
- 能否对于某个图模型实现软件架构后，该软件架构能够泛化应用到相似图模型呢？



问题提出

机器人数据采集可视化存储软件系统开发实例

- 实际开发中往往能发现很多软件是基于相似但很复杂的“抽象图模型”。
- 这时候，代码层抽象只能方便我们设计软件，但无法减少重复编程。
- 能否对于某个图模型实现软件架构后，该软件架构能够泛化应用到相似图模型呢？
- 答案是可以，即通过程序层抽象实现抽象图模型的抽象软件架构。



问题总结，目标与要求

问题总结

- 模块化设计是目前通用的软件设计方法，一般基于图模型设计。
- 模块化开发是高效的分工合作软件开发方式，两种开发方式各有自己的优缺点。
- 软件架构的泛化能够减少不必要的对相似图模型的编程。

问题总结，目标与要求

问题总结

- 模块化设计是目前通用的软件设计方法，一般基于图模型设计。
- 模块化开发是高效的分工合作软件开发方式，两种开发方式各有自己的优缺点。
- 软件架构的泛化能够减少不必要的对相似图模型的编程。

目标与要求

开发一种高效的软件开发框架：

问题总结，目标与要求

问题总结

- 模块化设计是目前通用的软件设计方法，一般基于图模型设计。
- 模块化开发是高效的分工合作软件开发方式，两种开发方式各有自己的优缺点。
- 软件架构的泛化能够减少不必要的对相似图模型的编程。

目标与要求

开发一种高效的软件开发框架：

- 提供基于图模型的模块化软件设计，并支持抽象图模型。

问题总结，目标与要求

问题总结

- 模块化设计是目前通用的软件设计方法，一般基于图模型设计。
- 模块化开发是高效的分工合作软件开发方式，两种开发方式各有自己的优缺点。
- 软件架构的泛化能够减少不必要的对相似图模型的编程。

目标与要求

开发一种高效的软件开发框架：

- 提供基于图模型的模块化软件设计，并支持抽象图模型。
- 使其具有两种模块化开发方式的优点：
  - 模块直接对应算法代码本身，不需要无关辅助功能。
  - 模块可以高效的细分到最小函数体或对象。
  - 模块的更新不需要重新编译整个系统。
  - 模块间数据传输不受数据量大小影响速度。
  - 模块可以被不同工程反复使用。
  - 软件架构与模块分离，可以被其它工程复用。

问题总结，目标与要求

问题总结

- 模块化设计是目前通用的软件设计方法，一般基于图模型设计。
- 模块化开发是高效的分工合作软件开发方式，两种开发方式各有自己的优缺点。
- 软件架构的泛化能够减少不必要的对相似图模型的编程。

目标与要求

开发一种高效的软件开发框架：

- 提供基于图模型的模块化软件设计，并支持抽象图模型。
- 使其具有两种模块化开发方式的优点：
  - 模块直接对应算法代码本身，不需要无关辅助功能。
  - 模块可以高效的细分到最小函数体或对象。
  - 模块的更新不需要重新编译整个系统。
  - 模块间数据传输不受数据量大小影响速度。
  - 模块可以被不同工程反复使用。
  - 软件架构与模块分离，可以被其它工程复用。
- 利用基于抽象图模型的抽象软件架构编程实现软件架构的泛化。

关于 RobotSDK

自顶向下的模块化开发理念与简单说明

- 模块化：
  - 模块化开发理念无需赘言。
  - RobotSDK 通过预制代码模板直接封装算法代码到动态库实现模块化开发。
  - 动态库模块通过显示链接可以自由嵌入抽象软件架构（RobotSDK 称为高层程序）。
  - 这种模块化开发方式满足了上述的目标与要求。

关于 RobotSDK

自顶向下的模块化开发理念与简单说明

- 自顶向下:

关于 RobotSDK

自顶向下的模块化开发理念与简单说明

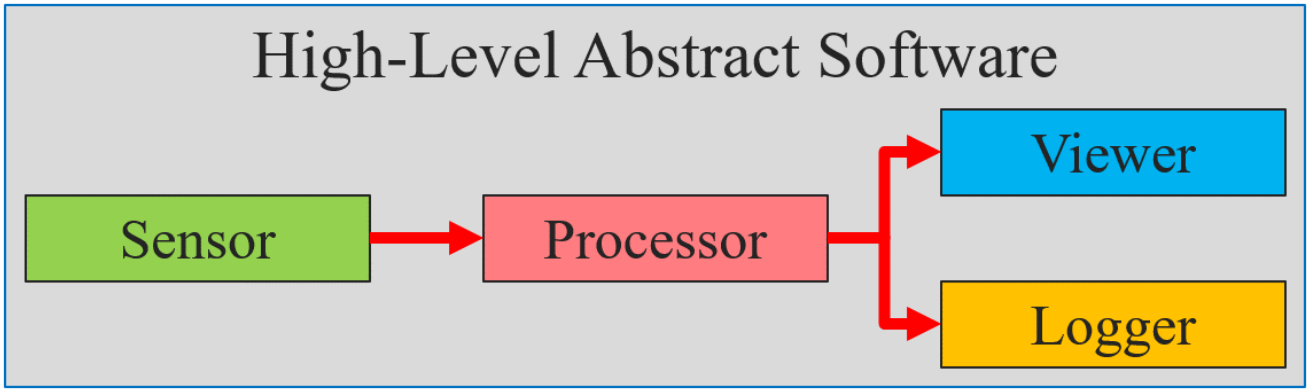
- 自顶向下:
  - 首先是一种从高层到底层的软件开发流程。
    - 【高层】高层图模型的设计与高层程序的编写，只关注图模型的实现，不具有具体功能。
    - 【底层】底层模块开发，只关注模块算法本身的编写和封装，不用考虑高层设计架构。

关于 RobotSDK

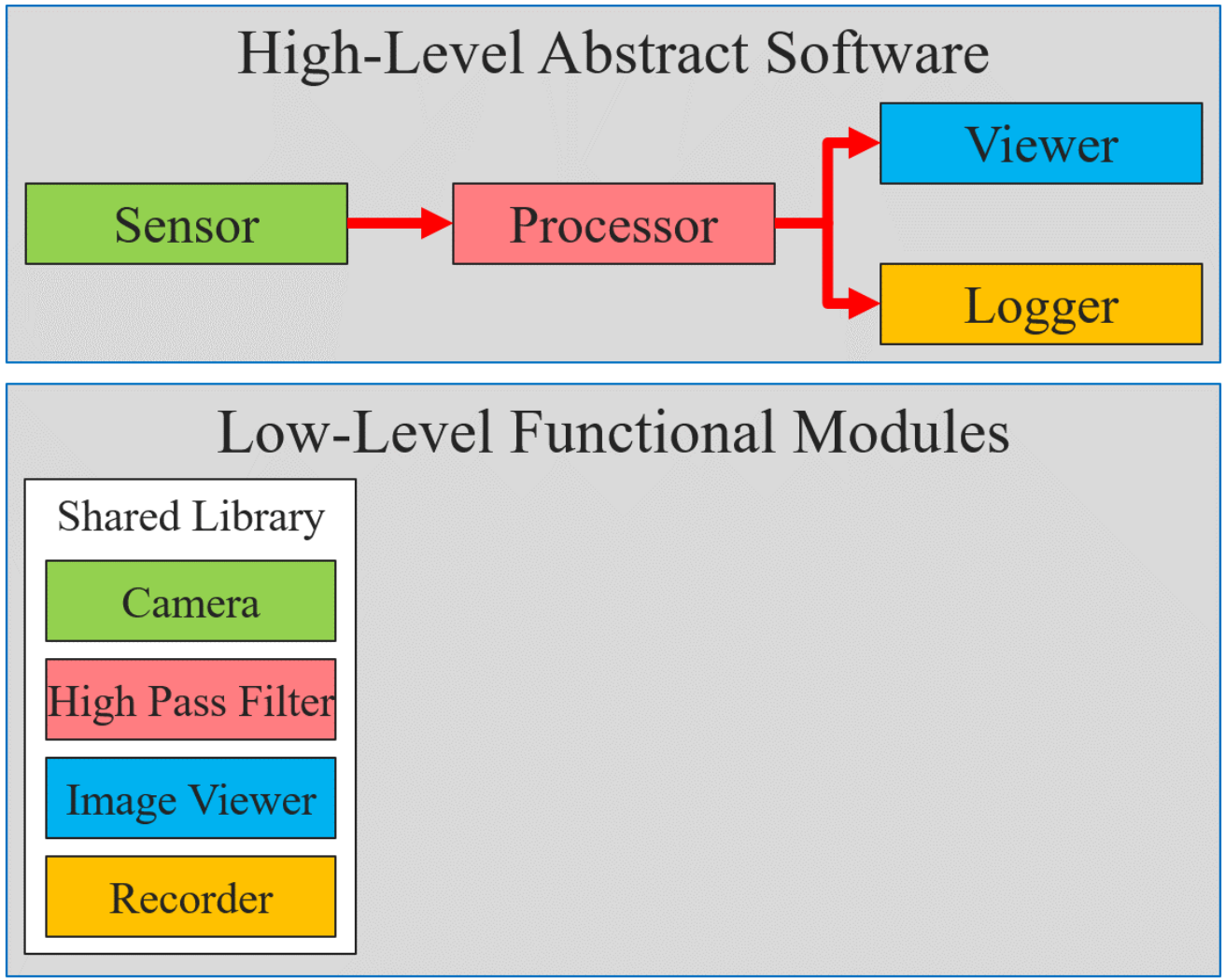
自顶向下的模块化开发理念与简单说明

- 自顶向下：
  - 首先是一种从高层到底层的软件开发流程。
    - 【高层】高层图模型的设计与高层程序的编写，只关注图模型的实现，不具有具体功能。
    - 【底层】底层模块开发，只关注模块算法本身的编写和封装，不用考虑高层设计架构。
  - 其次是一种从抽象到具体的软件开发流程。
    - 【抽象】抽象的高层图模型与对应的抽象软件架构实现，只关注图模型的结构实现，不具有功能。
    - 【具体】泛化的抽象软件架构在嵌入具体的底层功能模块后，拥有了对应具体图模型的软件功能。

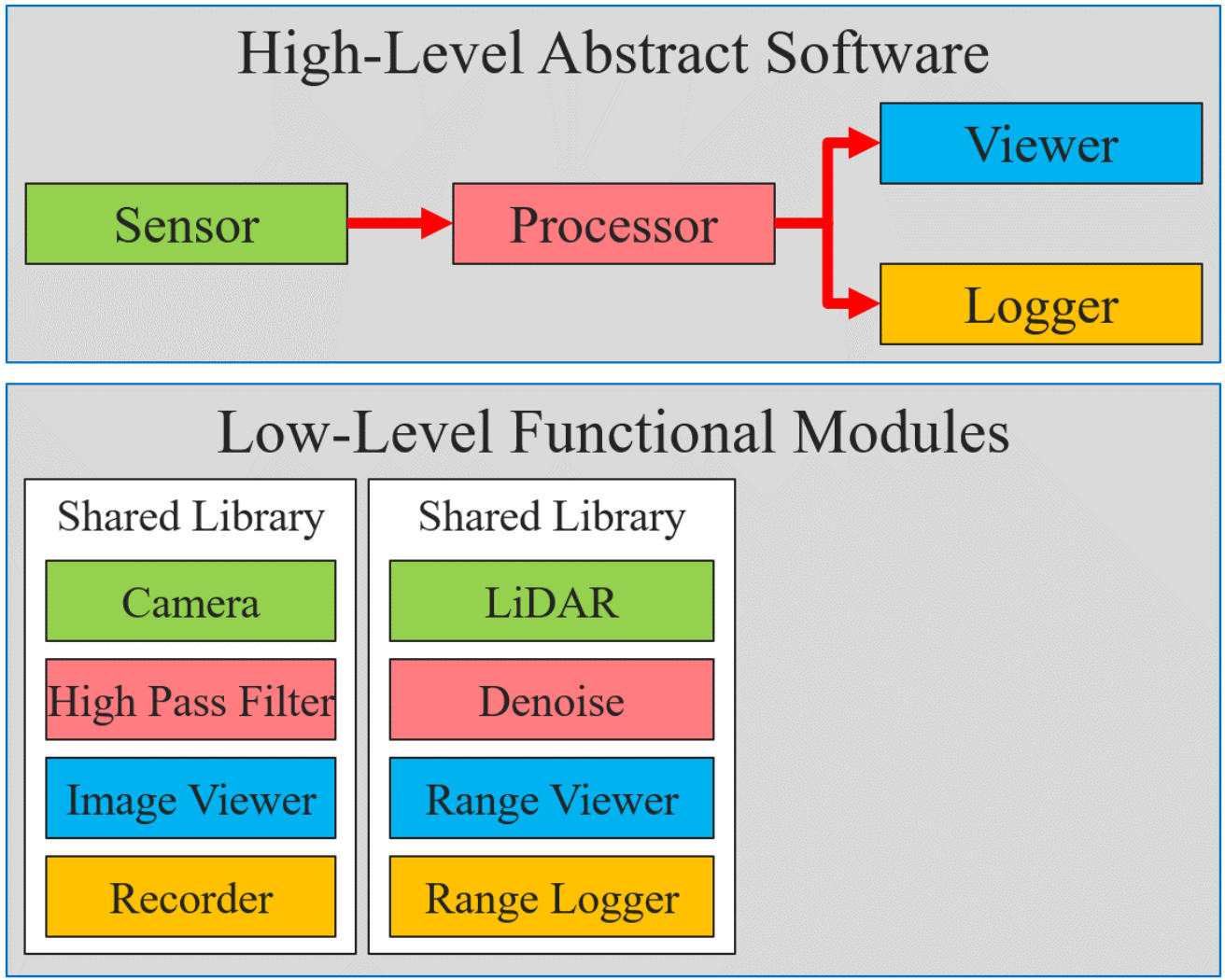
关于 RobotSDK—自顶向下的模块化开发理念实例 I



关于 RobotSDK—自顶向下的模块化开发理念实例 II

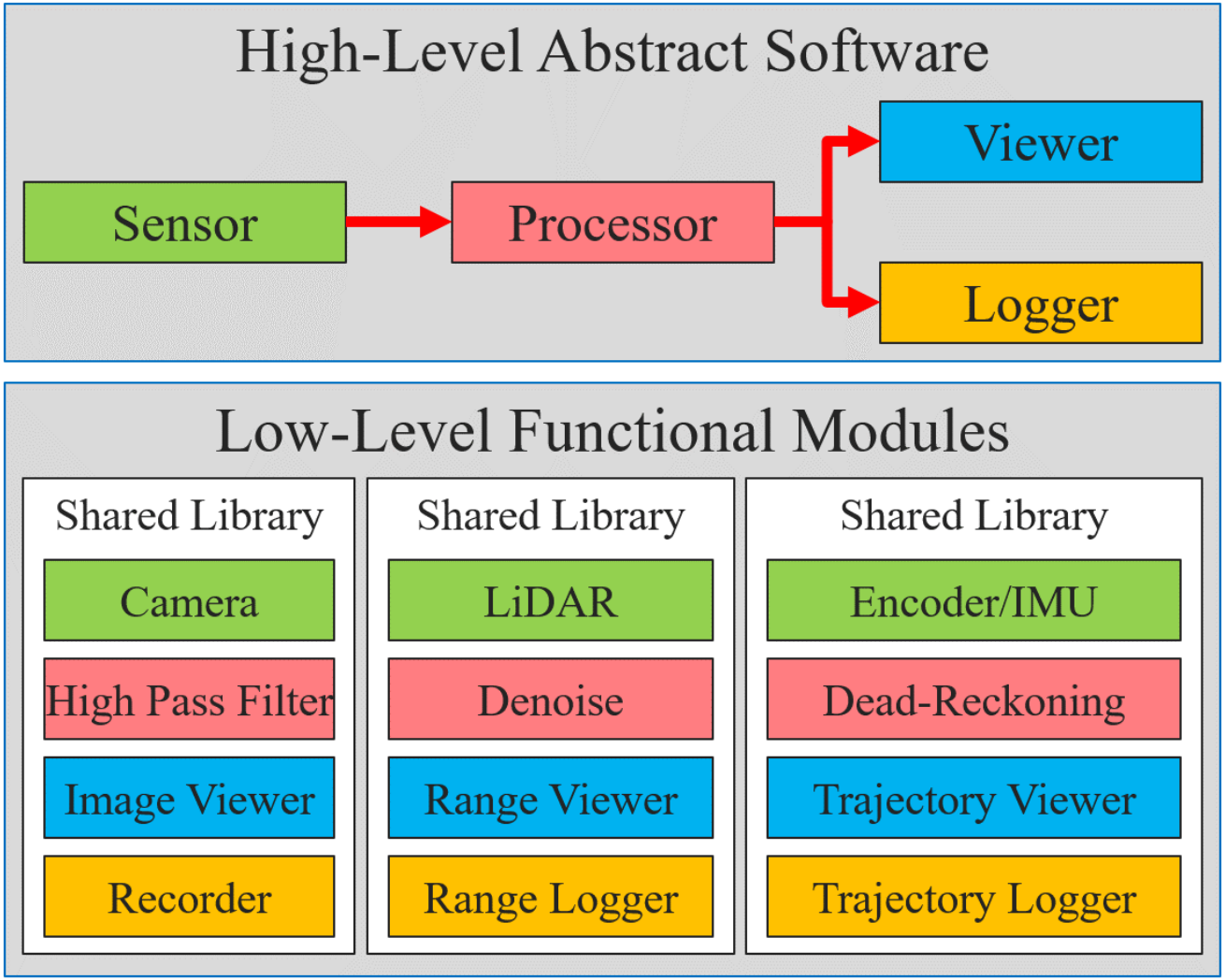


关于 RobotSDK—自顶向下的模块化开发理念实例 III



Carnegie Mellon University CyLab Security and Privacy Institute

关于 RobotSDK—自顶向下的模块化开发理念实例 IV



关于 RobotSDK—基本概念

高层图模型与高层程序

关于 RobotSDK—基本概念

高层图模型与高层程序

- 节点：功能模块的嵌入槽，同时定义了输入与输出端口。

关于 RobotSDK—基本概念

高层图模型与高层程序

- 节点：功能模块的嵌入槽，同时定义了输入与输出端口。
- 有向边：通用数据指针的通讯连接。所有不同类型的数据都存放在统一的数据池中，通信只传递指针，所以通信速度不会受到数据量影响。

关于 RobotSDK—基本概念

高层图模型与高层程序

- 节点：功能模块的嵌入槽，同时定义了输入与输出端口。
- 有向边：通用数据指针的通讯连接。所有不同类型的数据都存放在统一的数据池中，通信只传递指针，所以通信速度不会受到数据量影响。
- 高层程序：只实现抽象图模型，但不具有任何实际功能。当嵌入封装了功能模块的动态库后，才具有实际功能。另外可以只插入部分功能模块，从而实现其子图的功能，可以用来做子系统测试。

关于 RobotSDK—基本概念

高层图模型与高层程序

- 节点：功能模块的嵌入槽，同时定义了输入与输出端口。
- 有向边：通用数据指针的通讯连接。所有不同类型的数据都存放在统一的数据池中，通信只传递指针，所以通信速度不会受到数据量影响。
- 高层程序：只实现抽象图模型，但不具有任何实际功能。当嵌入封装了功能模块的动态库后，才具有实际功能。另外可以只插入部分功能模块，从而实现其子图的功能，可以用来做子系统测试。
- Robot-X：在最新的 4.0 版本，由于实现了脚本化高层程序，所以无需再编程实现，而是直接画图后解析即可得到高层抽象软件。Robot-X 就是抽象图模型的解析器，解析完后就成为了对应该图模型的高层程序。另外，Robot-X 可以同时解析多个抽象图模型，并进行图合并操作（相同节点会被合并），从而实现将简单子系统融合成复杂系统的功能。

关于 RobotSDK—基本概念

底层功能模块

关于 RobotSDK—基本概念

底层功能模块

- 预制代码模板：由 RobotSDK 根据图模型中的节点自动生成，其定义了功能模块与抽象节点的通信接口（底层模块开发者无需关注和了解）。底层模块开发者只需要将自己的算法代码填入算法封装区，并设计由模块“main”函数调用即可。

关于 RobotSDK—基本概念

底层功能模块

- 预制代码模板：由 RobotSDK 根据图模型中的节点自动生成，其定义了功能模块与抽象节点的通信接口（底层模块开发者无需关注和了解）。底层模块开发者只需要将自己的算法代码填入算法封装区，并设计由模块“main”函数调用即可。
- 动态库：通过模板代码将功能模块的实现算法封装入动态库，从而通过显示链接（explicit link）的方式将功能模块嵌入高层抽象软件的节点。一个动态库可以封装任意多的功能模块，一个动态库可以被任意多的高层抽象软件使用。注意：链接了的动态库如果被更新，更新内容只会在高层抽象软件重启后才起作用，这是目前还没有解决的热插拔问题。

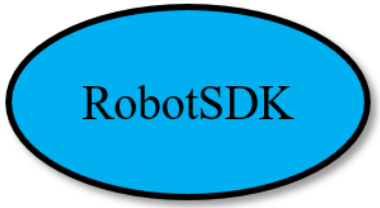
关于 RobotSDK—基本概念

底层功能模块

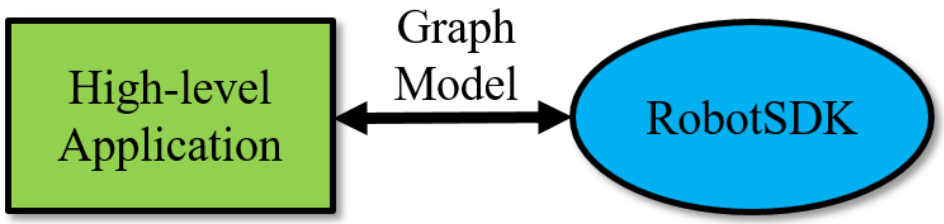
- 预制代码模板：由 RobotSDK 根据图模型中的节点自动生成，其定义了功能模块与抽象节点的通信接口（底层模块开发者无需关注和了解）。底层模块开发者只需要将自己的算法代码填入算法封装区，并设计由模块“main”函数调用即可。
- 动态库：通过模板代码将功能模块的实现算法封装入动态库，从而通过显示链接（explicit link）的方式将功能模块嵌入高层抽象软件的节点。一个动态库可以封装任意多的功能模块，一个动态库可以被任意多的高层抽象软件使用。注意：链接了的动态库如果被更新，更新内容只会在高层抽象软件重启后才起作用，这是目前还没有解决的热插拔问题。
- 多线程管理与数据流驱动：Robot-X 自动为每个嵌入的功能模块提供单独的线程，无需底层开发者额外设计。同时基于图模型的软件架构本身存在“源”和“漏”两种节点，所以处于不同线程的模块是通过从“源”到“漏”的数据流驱动进行同步的，这一点也无需底层开发者做额外设计。

Carnegie Mellon University CyLab Security and Privacy Institute

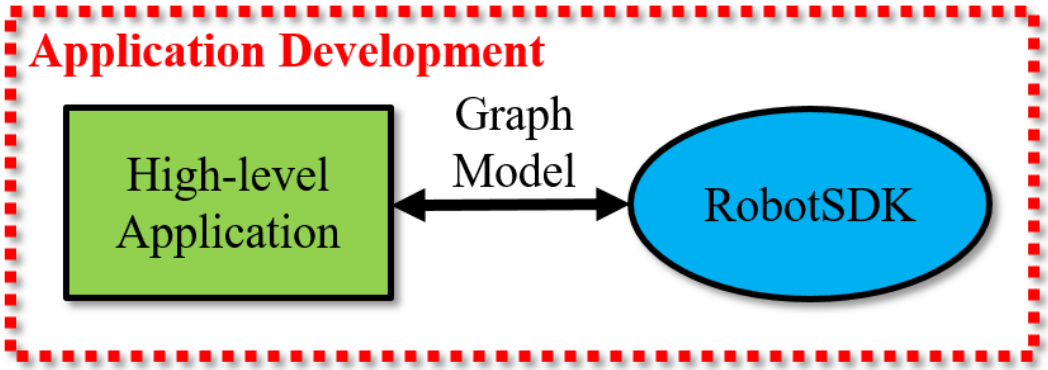
什么是 RobotSDK—基本架构 I



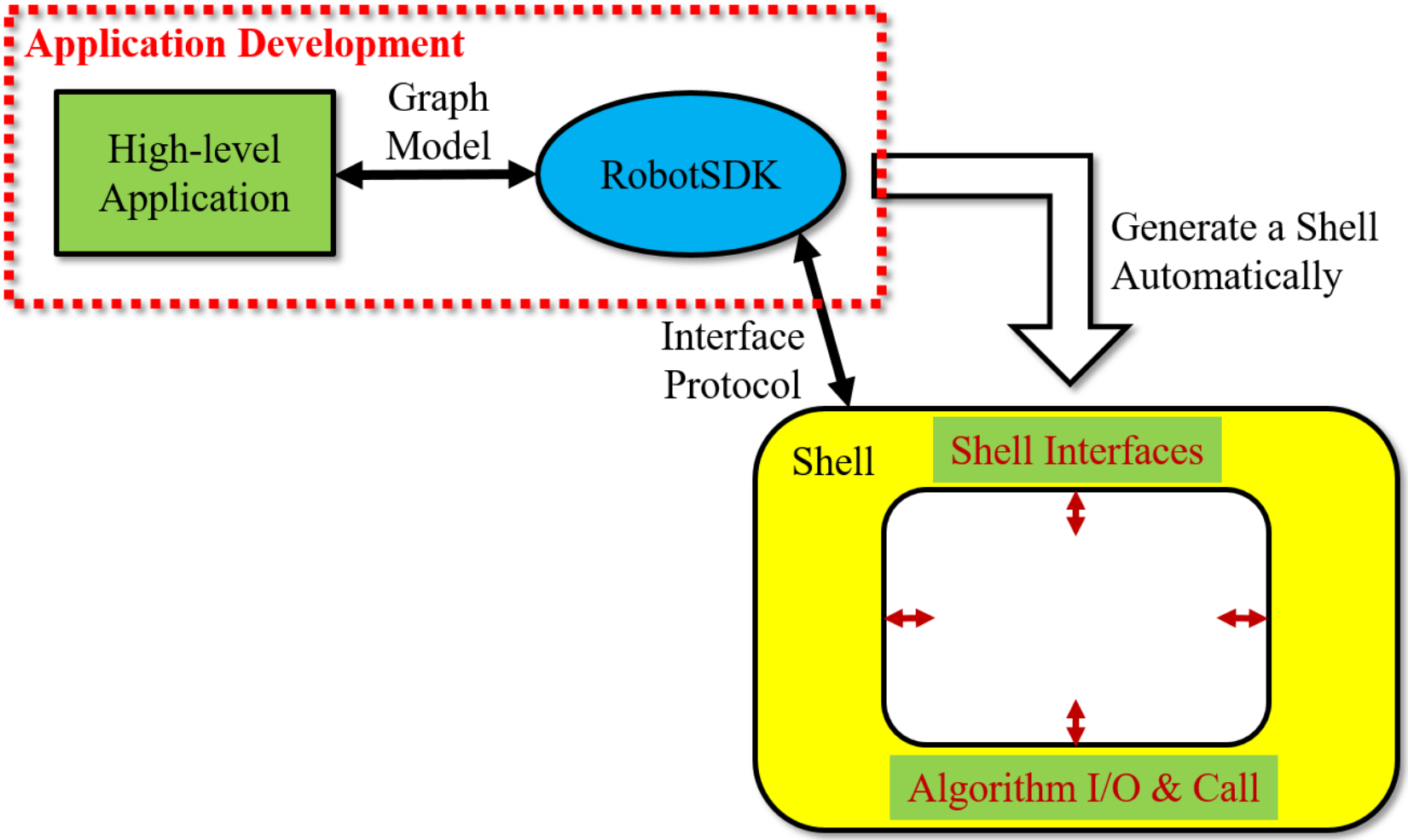
什么是 RobotSDK—基本架构 II



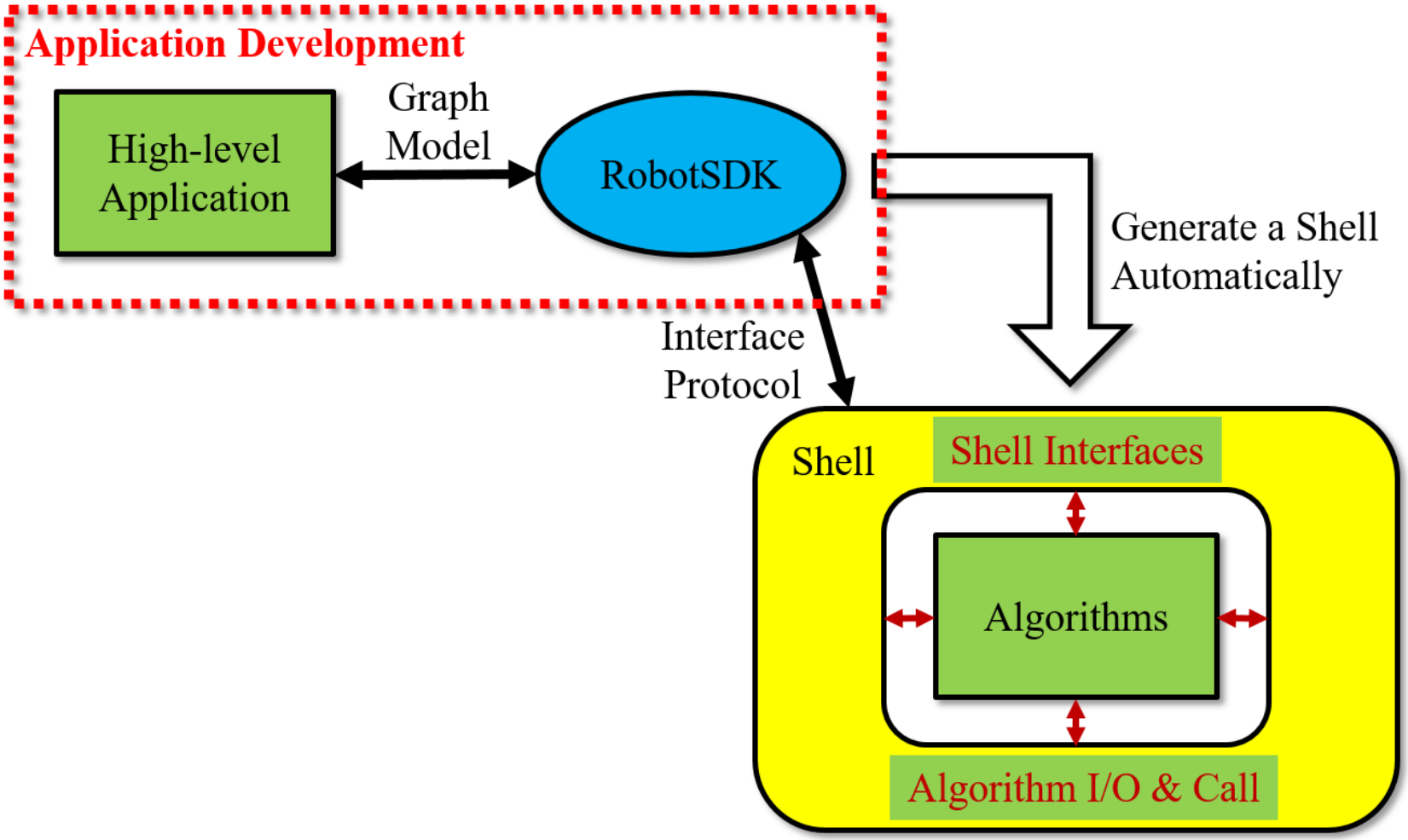
什么是 RobotSDK—基本架构 III



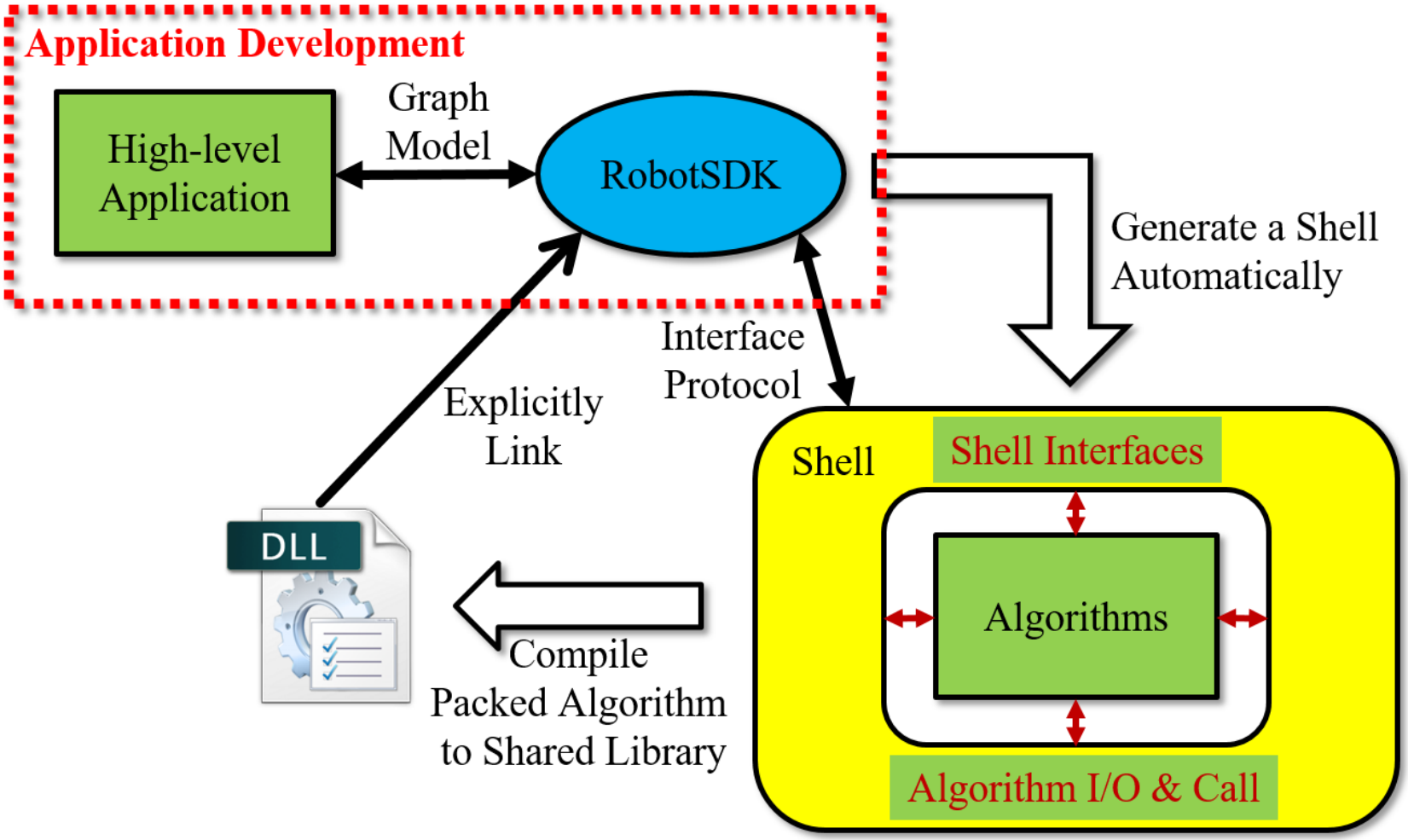
什么是 RobotSDK—基本架构 IV



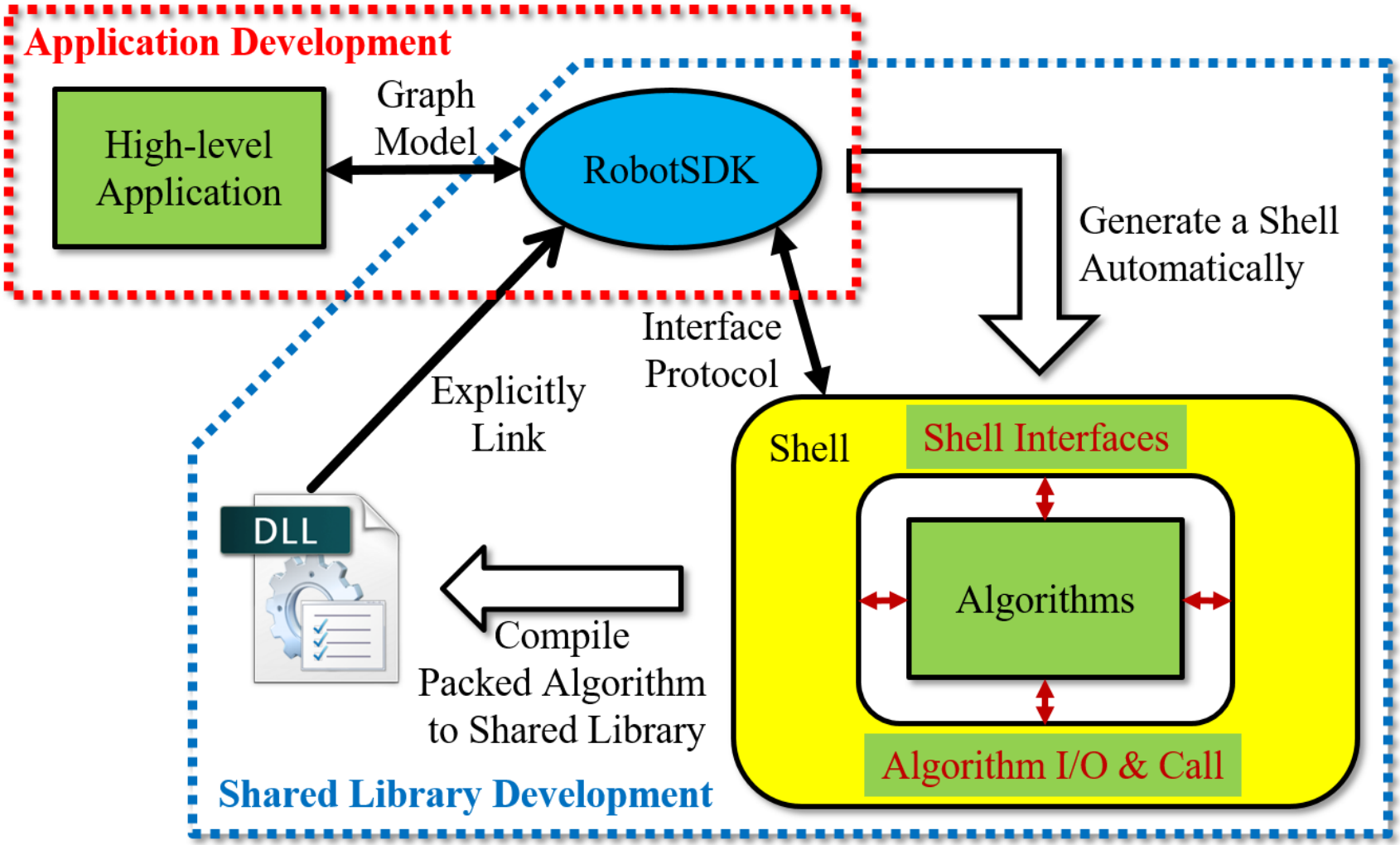
什么是 RobotSDK—基本架构 V



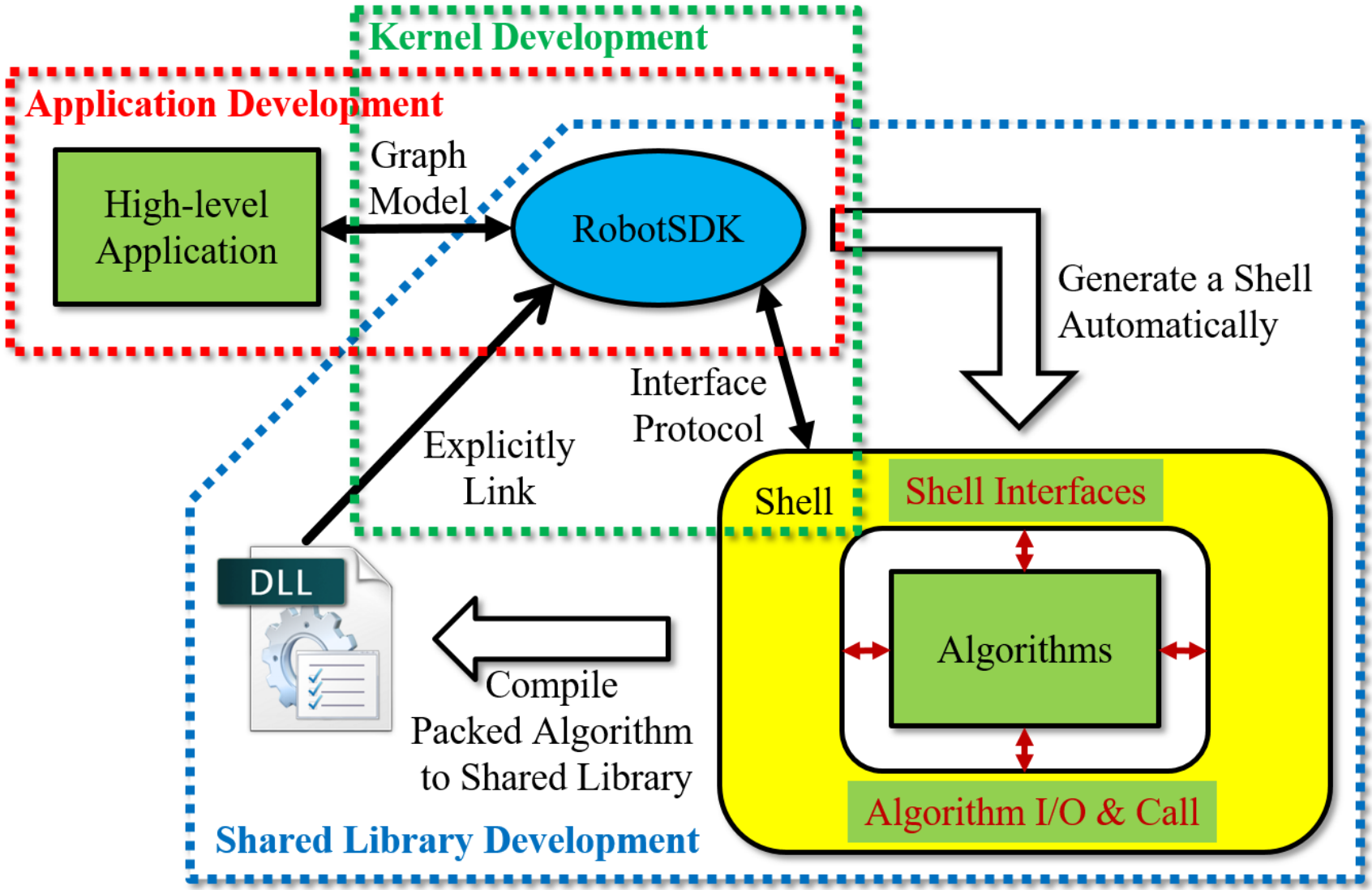
什么是 RobotSDK—基本架构 VI



什么是 RobotSDK—基本架构 VII



什么是 RobotSDK—基本架构 VIII



什么是 RobotSDK—使用方法与实例

- RobotSDK 4.0 的使用方法介绍视频：https://youtu.be/uf7t8gWjN3U
- RobotSDK 4.0 快速入门手册：https://raw.githubusercontent.com/RobotSDK/RobotSDK/RobotSDK_4.0/RobotSDK4_QuickManual.pdf
- RobotSDK 3.0 在北大 Goblin 小机器人上的使用（开发者方永 {方方土}【kun 显示不出来】）：https://youtu.be/ER-vk2F8UQU
- RobotSDK 3.0 用于 KITTI Dataset 的处理和可视化：https://youtu.be/Vgil3ObP3RE
- RobotSDK 4.0 在筑波挑战赛（名古屋大学预赛）上使用：
  - https://youtu.be/syAKQ_JizRM
  - https://youtu.be/yaWJ6Ofll_I
- RobotSDK 4.0 的第一个外部需求订单，DPM 训练样本收集工具：https://youtu.be/mjkYXEXnPMs
- RobotSDK 4.0 在本人研究中的使用：https://youtu.be/rAkuc5CNbl0

Carnegie Mellon University CyLab Security and Privacy Institute

RobotSDK 的发展简史—北京大学时期

筑波挑战赛（2013.08-2013.09）

- SensingPlatform SDK (RobotSDK 1.0)
  - 尝试使用 RobotSDK 作为标准的软件开发工具（感谢陈向羽的耐心使用）。
  - 只用于传感器数据的采集和可视化，算法编写没有采用 RobotSDK。
- 主要特点：
  - 模块化软件设计基于非抽象的底层图模型。
  - 模块化开发方式是基于静态库的代码方式。
  - 无模板代码，接口需要自己写，同时 RobotSDK 内核代码需要嵌入工程，开发非常不便。
  - 缺少数据池、数据流、数据缓冲和多线程的管理。

RobotSDK 的发展简史—北京大学时期

筑波挑战赛之后（2013.09-2014.01）

- SensorPlatform SDK (RobotSDK 2.0)
  - 根据陈向羽的反馈和比赛中出的问题进行的一次重大升级。
  - 部分实现了自顶向下的软件设计理念。
  - 由代码方式转变为程序方式。
  - 可以模块化算法代码。
- 特点：
  - 模块化软件设计基于非抽象高层图模型。
  - 模块化开发方式基于动态库的程序方式。
  - RobotSDK 内核代码与工程实现了分离。
  - 无模板代码，动态库开发不方便。
  - 缺少数据池、数据流、数据缓冲和多线程的管理。
  - 部署方法非常麻烦。

RobotSDK 的发展简史—北京大学时期

离开北大前（2014.01-2014.08）

- RobotSDK 2.1, 2.2, 3.0
  - 完全实现了自顶向下的模块化软件开发理念。
  - 开发了大量高层与底层辅助开发程序，用于高层与底层程序的开发。
  - 较为成熟的 RobotSDK 3.0 以北京大学名义申请了软件著作权
  - 并用于了本科生机器人课程教学。
- 特点：
  - 模块化软件设计基于抽象高层图模型。
  - 模块化开发方式引入了预制代码模板，动态库开发与封装非常方便。
  - 实现了基于抽象高层图模型的抽象软件架构编程，高层程序具有泛化能力。
  - 部分解决了数据池、数据流、数据缓冲和多线程的管理问题。
  - 模块参数 XML 文件化，参数调整可以通过修改通用的 XML 文件实现。
  - 节点类型仍然不够抽象（显式地分为“源”、“漏”、“中间节点”）。

Carnegie Mellon University CyLab Security and Privacy Institute

RobotSDK 的发展简史—名古屋大学时期

名古屋大学时期 (2014.09-2016.04)

- RobotSDK 4.0
- 名古屋大学的研究需要 RobotSDK 能够处理高速多模块的情况，同时在使用 ROS 的过程中得到了一些启发。
- 在保留了 RobotSDK 3.0 的自顶向下模块化软件开发理念下，重写了内核，彻底解决了数据池、数据流、数据缓冲和多线程的管理问题。
- 实现了高层程序的脚本化，并开发了通用高层软件 Robot-X，从而免去了编写高层程序的步骤。
- 多类节点被统一为一种通用节点（脚本化的前提）。
- 模块参数 UI 化（基于 C++11 技术），参数直接可以通过 Robot-X 的通用参数修改器修改。
- 开发了一套完整的RobotSDK Syntax进一步简化了动态库的开发（基于 C++11 技术），使开发者只用关注于算法本身。
- 架设了ROS 接口，RobotSDK 可以非常方便的与基于 ROS 的程序进行通信，同时本身也可以作为 ROS 的节点程序。

RobotSDK 的发展简史—目前现状

好的方面

- 经过了 3 年实践共 6 代升级，RobotSDK 4.0 已经基本实现了一个稳定可靠的自顶向下模块化软件开发框架，最初的理念也已经实现。
- 北大方面，实验室的一些项目开始使用 RobotSDK 进行开发。
- 名大方面，他们开发的 Autoware（基于 ROS 的无人车软件集）也集成了 RobotSDK 的部分功能。
- 我研究工作中的软件开发大部分基于 RobotSDK 实现。

不足的方面

- 图模型的设计界面不够美观和强大，需要使用更专业的图模型库代替。
- RobotSDK 基于队列的消息通信无法满足实时系统。
- 与 ROS 的融合度仍然不够。(ROS 是一种程序间的模块化组织，RobotSDK 是一种程序内的模块化组织，所以两者应该是互补的关系)

RobotSDK 目前开发计划—卡耐基梅隆大学期间

1. Java 版的 RobotSDK-JE（毕竟 Java 在 CMU 比较流行，但是实时系统就算了吧）
2. RobotSDK-4.1：
   - 使用新的图模型库 GOJS 代替 GraphViz；
   - 使用 ROS 数据（Message）类型定义进行深度 ROS 融合；
   - 开发独立的 IDE 平台 Robot-Workshop，用于集成高层图模型设计（RobotSDK-4.0 由 Robot-X 代劳）和底层模块编程工具（目前借由 QtCreator 平台进行编程）。
3. RobotSDK-4.2：
   - 基于高层抽象图模型的系统可行性认证与实时系统的时间规划（目前和 UPenn 在合作）；
   - 改进基于队列的通信为实时通信；
   - 系统运行实时监控工具。

RobotSDK 开源开发计划

- RobotSDK 目前只有我一个人负责开发和维护，而博士的工作越来越繁重，很多想法都没有时间实现。
- 同时本人也不是专业的软件工程出身，只是凭着个人兴趣和有限的能力在做这件事。
- 但是通过实践，个人认为这种自顶向下的模块化开发理念是有道理和优势的。
- 所以希望能有越来越多的人以 RobotSDK 为起始平台参与到这个项目的开发：
  - 你可以来做内核开发和优化，以及目前最关键的是让它能够在实时系统下工作。
  - 你可以来做辅助工具开发，让基于 RobotSDK 的开发更简单更有效率。
  - 你可以来使用 RobotSDK 开发软件，同时分享自己的功能模块到社区，供别人参考使用。
- RobotSDK 的 GitHub 地址：`https://github.com/RobotSDK/RobotSDK`
- 谢谢大家！

Carnegie Mellon University CyLab Security and Privacy Institute