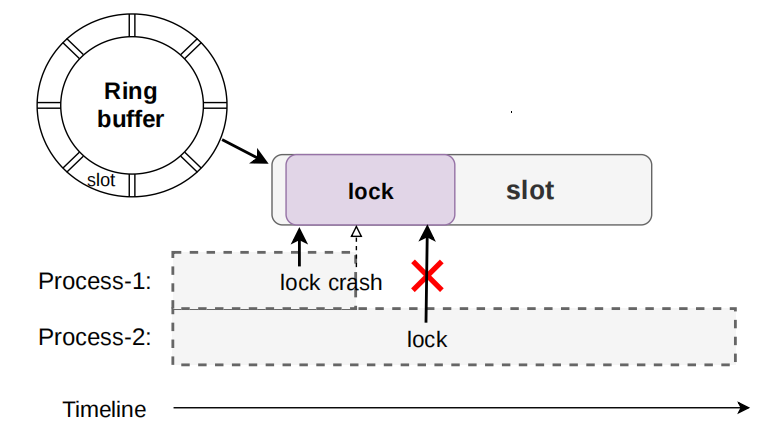
**一、Cyber RT共享内存设计**



它提出了一个与发布者和订阅者具有环状结构关联的共享缓冲区。共享缓冲区的环状结构可以分为几个slot，如图所示，其中存储了辅助信息和真实的数据。诸如读写锁、读计数、写标志、大小等辅助信息被用于管理slot的状态。当发布者发布消息时，它将slot的状态设置为写入，以阻止其他进程访问它。类似地，当订阅者从一个slot读取消息时，它将状态改为读取，这也防止对其同时写入。值得注意的是，锁和计数使共享内存是有状态的。因此，如果发布者或订阅者意外退出，状态总是存在并且不可改变，这可能导致严重的情况，即slot不可用，无法恢复以接收新的新数据。此外，环缓冲区作为其设计特性，以固定的slot大小来实现，这使得很难处理不同大小的消息。

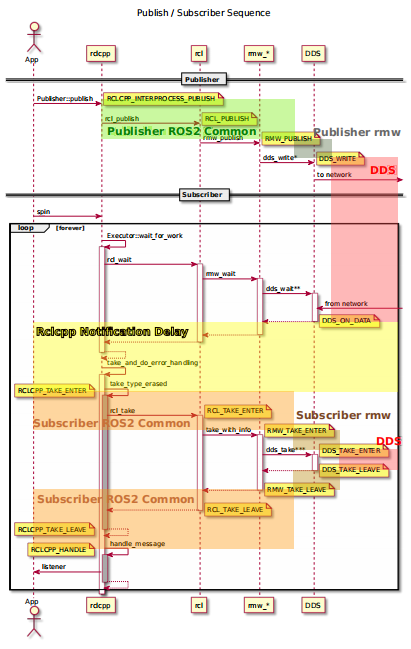
**二、共享内存设计方案**

共享内存池

1. 先出先出。在我们的IPC过程中，首先先输出(FIFO)是一个非常重要的特性，它表明第一个分配的内存首先是被释放的。基于这一特性，我们新设计的体系结构可以避免内存管理中的大量片段。
2. 可变的长度和无限长的slot。为了解决基于环的共享内存的局限性，我们需要设计一种新的架构，它可以支持每个slot的可变长度和无限数量的slot的数据传输。

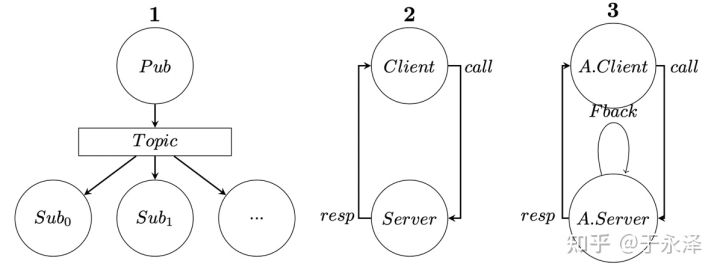
3、publisher向slot传入信息后通过socket发送地址信息给subscriber

**三、ROS2 pub/sub函数调用流程图**



**四、ROS通信机制和缺点**

（1）ROS2中有三种通信机制



Publish-Subscribe：发布者不直接发送数据给订阅者，而是分类为Topic. 订阅了该Topic的订阅者将获得数据。

Service-Client：Client向Server发送请求，Server执行任务，等任务完成后Server把响应返回给Client

Actions：该通信机制基于Service-Client，只不过在任务执行过程中的产生多次反馈。

（2）ROS2的调度模型（单线程执行器）

单线程执行器（Single Thread Executor）根据回调函数的优先级来决定它们的执行顺序。 回调函数的优先级取决于： 回调的类型和注册顺序（Registration Order）。

回调的类型（按照优先级降序排列）：

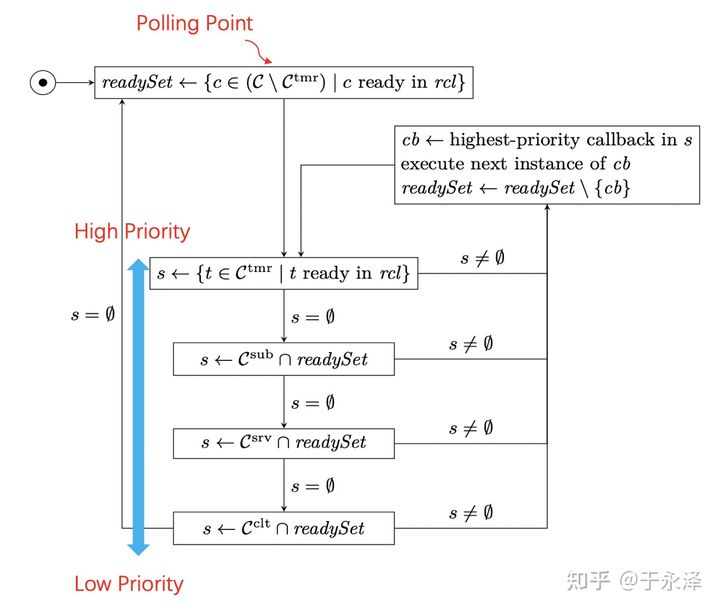
1. 定时器回调  ：由系统定时器触发，一般是周期性触发。一个回调链由多个回调组成。而第一个回调通常是定时器回调。 它在 4 种回调类型中具有最高优先级。

2. 订阅型回调  ：基于Publish-Subscribe这种通信机制。优先级排第二。

3. 服务型回调  : 基于Service-Client和Actions这两种通信机制中的Service.优先级排第三。

4. 客户型回调  ：基于Service-Client和Actions这两种通信机制中的Client. 最低优先级。

如果两个回调的类型不同，则比较它们的类型。如果它们的类型相同，则比较他们的注册顺序。先注册的优先级更高。



首先，如果Readyset 为空，那么执行器将所有就绪（ready）的常规回调放入Readyset。常规回调是除了定时器回调之外的回调。

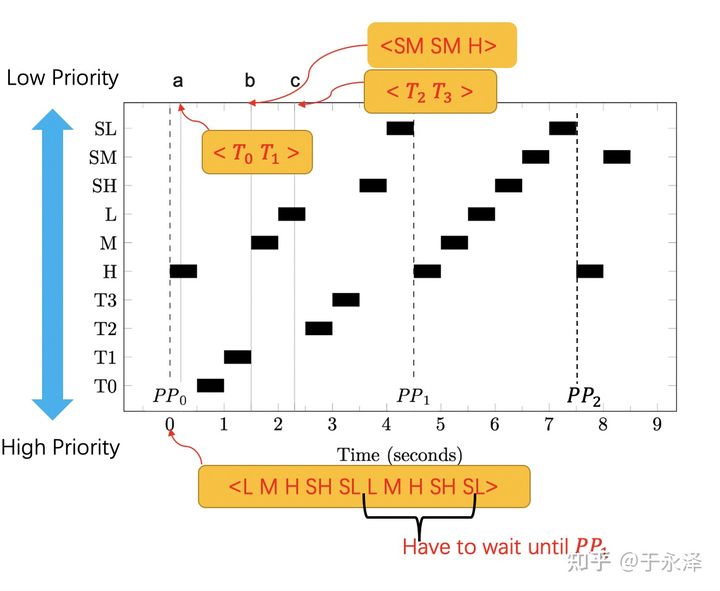
Readyset 更新的时间点称为轮询点。两个相邻轮询点之间的时间间隔称为处理窗口(Processing Window)。定时器回调不需要等待轮询点。

总是首先执行定时器回调，然后是订阅型回调，然后是服务型，最后是客户型。

如果某些回调具有相同的类型，将根据它们的注册顺序执行它们。更早在执行器上注册的回调则具有更高的优先级。如果回调有多个实例，则只有第一个实例可以执行。

如果 Readyset 中所有回调的第一个实例都执行了，也就是说ReadySet 是空集，那我们就到达下一个轮询点，新的处理窗口将开始。

## （3）例子一



T0、T1、T2是注册顺序不同的定时器回调。H、M 和 L 是具有不同注册顺序的订阅型回调。 SH, SM, SL 是不同注册顺序的服务型回调。 T0 的优先级最高，SL 的优先级最低。

在轮询点PP0，单线程执行器得到消息序列<L M H SH SL L M H SH SL>，并且只将每个回调的一个实例放入就绪集中。所以第二个L、M、H、SH、SL要等待下一个轮询点PP1。

因此，我们首先执行H。

在 a 时刻，定时器回调 T0 和 T1触发，并在 H 完成后执行它们。计时器回调不需要等待轮询点。

在 b 时刻，我们收到了新的消息序列 {SM SM H}，但我们无法将它们放入到ReadySet中。它们必须等待下一个轮询点PP1。

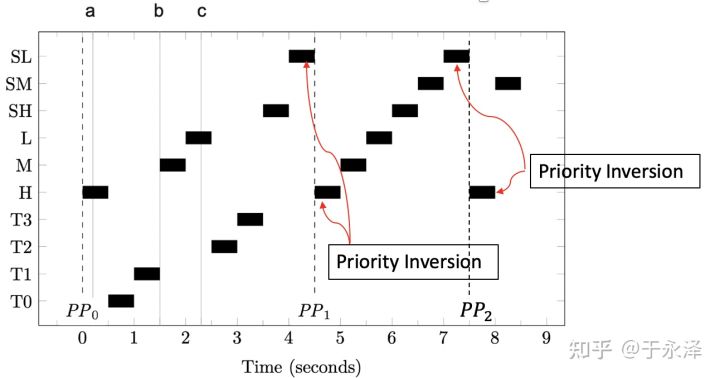
所以我们执行 M 和 L。

在 c 时刻，我们得到定时器回调 T2 和 T3，并在L完成后执行它们，然后执行SH和SL。

现在所有的第一个实例都已执行。其余实例将在接下来的处理窗口中执行。

## （4）ROS2调度模型的缺点

第一，优先级反转，因为所有常规回调都必须等待轮询点，并且每个处理窗口中每个回调只有一个实例可以执行。这就导致了高优先级的回调实例可能比低优先级的回调实例之行的更晚。比如说下图中的PP1轮询点和PP2轮询点附近产生了优先级反转。总的来说，轮询点的存在造成了优先级反转。



第二，隐式优先级。 回调的优先级取决于回调类型和注册顺序。 这往往对ROS应用开发者用户来说是不直观的。

第三，不可抢占。 回调执行期间不能被更高优先级的回调抢占并中断，这也是优先级反转的一种情况。

**五、调度改进方案**

定理：链Γc：=[τ1，...τi，...，τj，...，τN]的回调函数都在相同的CPU，新实例开始执行之前，旧链实例保证完成执行需要满足τj回调优先级高于τi(j>i)

如果执行者只有单链的回调函数Γc=：[τ1，...，τi，...，τj，...，τj，...，τN]，这些回调的优先顺序应该反向分配，即τj(j>i)将比τi获得更高的优先级，以满足引理1。

如果单链的回调函数包含了timer回调函数和常规回调函数，那么常规回调函数的优先级应该高于timer回调函数。

多链回调函数情况下，高优先级的链中的所有回调的优先级应该高于低优先级的链中的所有回调的优先级。

1. **底层通信机制**

进程间：dds

进程中：

总览：

为每个subscriber创建一个buffer，当消息被pub到一个topic中时，消息的publisher把消息push到每个关联到这个topic的subscriber的buffer中，并且产生一个notify，唤醒executor，这个executor可以把消息从buffer中弹出，并trigger subscriber中的callback

概念：

Node：发布者或订阅者的实例。包含一组callback

Executor：OS调度层级的调度实体（如线程），通过将node分配给executor来完成执行

细节：

在subscriber被创建时，用SubscriptionIntraProcess继承Waitable类，并加入node 的Waitable interface。SubscriptionIntraProcess对象 有指向对应subscriber的callback函数的指针 和 ringbuffer 的使用权

消息发出后，SubscriptionIntraProcess的条件变量rcl\_guard\_condition\_t的监听条件被设置为ready，以便于被rclcpp::spin方法监视

与SubscriptionIntraProcess对象关联的条件变量唤醒rclcpp::spin，

SubscriptionIntraProcess::execute()函数被执行，这将从ring buffer提取第一个消息，

然后SubscriptionIntraProcess调用AnySubscriptionCallback::diapatch\_intra\_process()方法trigger关联的callback

Spin函数是一个executor的封装，executor通过add\_node添加多个节点，然后调用executor::spin方法，通过while循环不断监控，执行get\_next\_executable()函数，它遍历查询当前节点的所有subscriber句柄，如果有可执行的回调，那么就拿来执行

TODO

更底层的比如具体写到buffer里是怎么写的，修改的真的是类int值吗？

为什么会有丢包的情况（10Hz收到变成20Hz）