03-A

列表

循位置访问

Don't lose the link.

- Robin Milner

邓俊辉

deng@tsinghua.edu.cn

从静态到动态

❖ 根据是否修改数据结构，所有操作大致分为两类方式

- 静态：　仅读取，数据结构的内容及组成一般不变：get、search
- 动态：　需写入，数据结构的局部或整体将改变：put、insert、remove

❖ 与操作方式相对应地，数据元素的存储与组织方式也分为两种

- 静态：　数据空间整体创建或销毁

  数据元素的物理存储次序与其逻辑次序严格一致；可支持高效的静态操作

  比如向量，元素的物理地址与其逻辑次序线性对应

- 动态：　为各数据元素动态地分配和回收的物理空间

  相邻元素记录彼此的物理地址，在逻辑上形成一个整体；可支持高效的动态操作

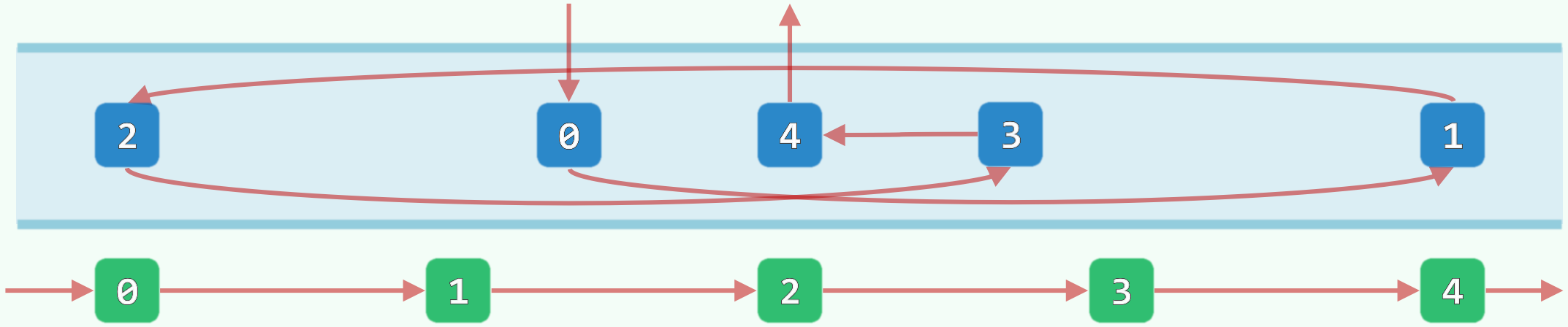
从向量到列表

❖ 列表（list）是采用动态储存策略的典型结构

- 其中的元素称作节点（node），通过指针或引用彼此联接
- 在逻辑上构成一个线性序列：L = { a_0, a_1, ..., a_{n-1} }

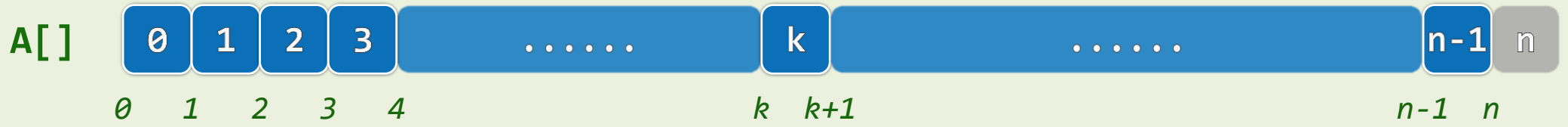
❖ 相邻节点彼此互称前驱（predecessor）或后继（successor）

- 没有前驱/后继的节点称作首（first/front）/末（last/rear）节点



从秩到位置

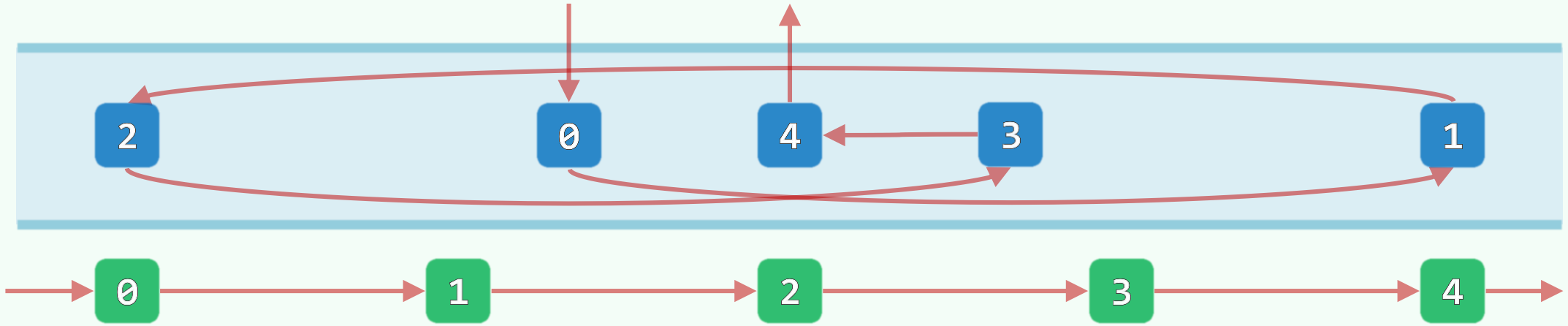
❖ 向量支持循秩访问（call-by-rank）：根据元素的秩，可在O(1)时间内直接确定其物理地址



❖ 这种高效的方式，可否被列表沿用？

❖ 比如，从头/尾端出发，沿后继/前驱引用...

```
//List::operator[](Rank r)
```



从秩到位置

- 然而，此时的循秩访问成本过高，已不合时宜
- 因此，应改用循位置访问（call-by-position）的方式

  亦即，转而利用节点之间的相互引用，找到特定的节点
- 比喻：找到我的朋友A的亲戚B的同事C的战友D的...的同学Z