

Chapter 3

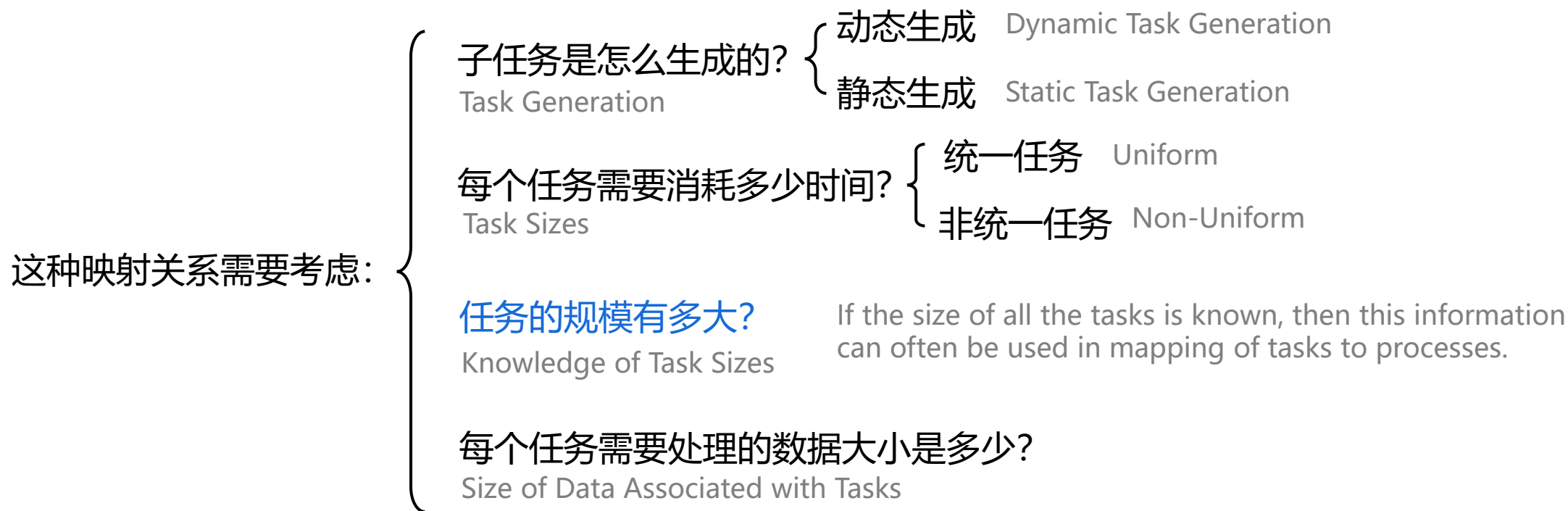
Principles of Parallel Algorithm Design

2024/06/11

3.3 Characteristics of Tasks and Interactions

问题:

把不同的任务分配（映射）到不同的进程上计算时，需要考虑任务的哪些特性？



3.3 Characteristics of Tasks and Interactions

问题:

在不同的任务并行时，需要考虑任务交互的哪些特性？

任务开始之前是否可以确定交互关系？

- 静态交互
- 动态交互

并行的任务在其空间位置上是否存在关系？

- 有规则的
- 无规则的

并行任务导致的数据共享关系？

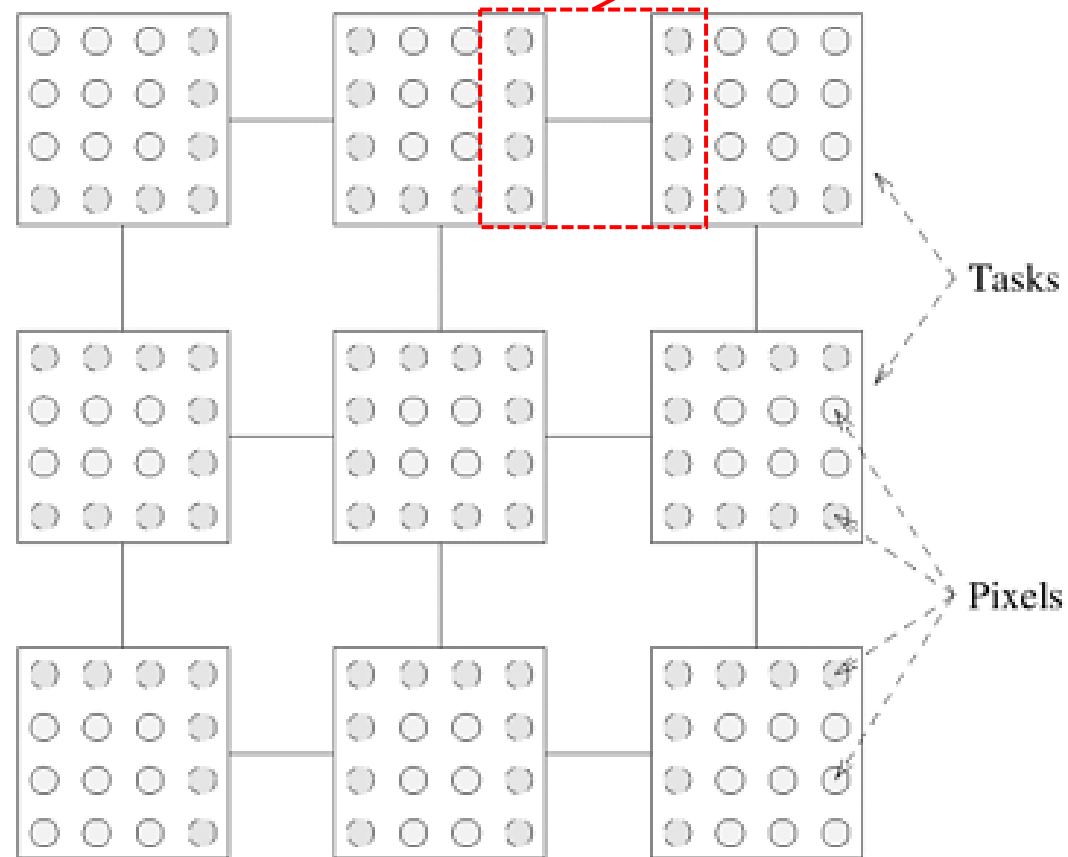
- 只读数据
- 读写数据

不同的数据共享关系决定着不同的映射方法

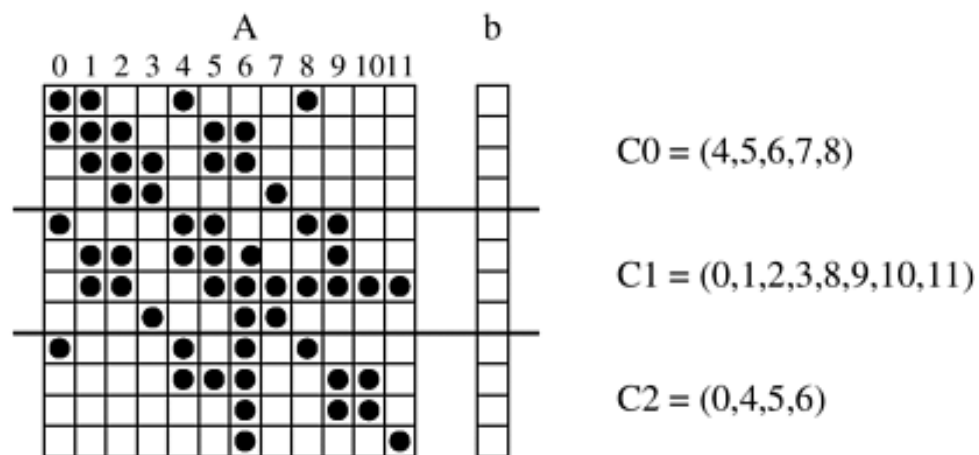
3.3 Characteristics of Tasks and Interactions

有规则与无规则 (Regular versus Irregular)

图像抖动 – 有规则的交互



稀疏矩阵乘法 – 无规则的交互



3.4 Mapping Techniques for Load Balancing

问题:

如何实现不同的效率最大化?



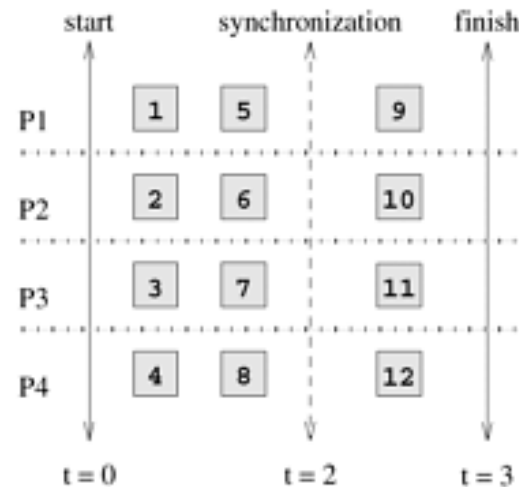
问题:

如何使不同的进程同时完成?

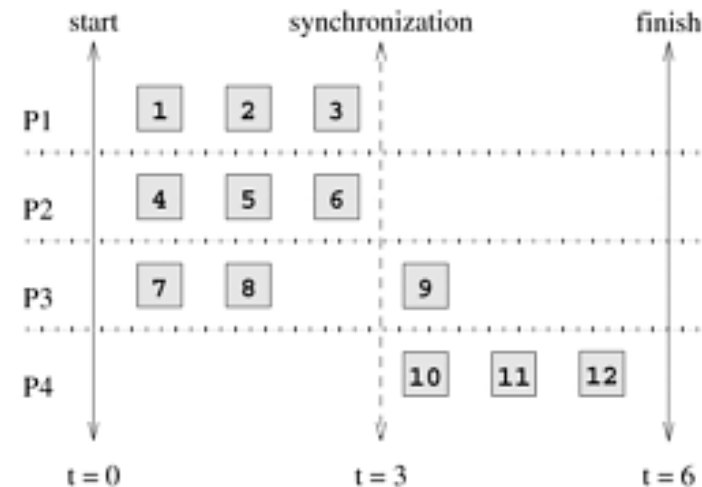


问题:

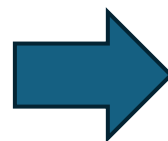
如何把不同的任务放到合适的进程上?



(a)



(b)



映射技术与负载均衡

动态映射

静态映射

3.4 Mapping Techniques for Load Balancing

静态映射 - 基于数据 - Block Distributions

row-wise distribution

P_0
P_1
P_2
P_3
P_4
P_5
P_6
P_7

column-wise distribution

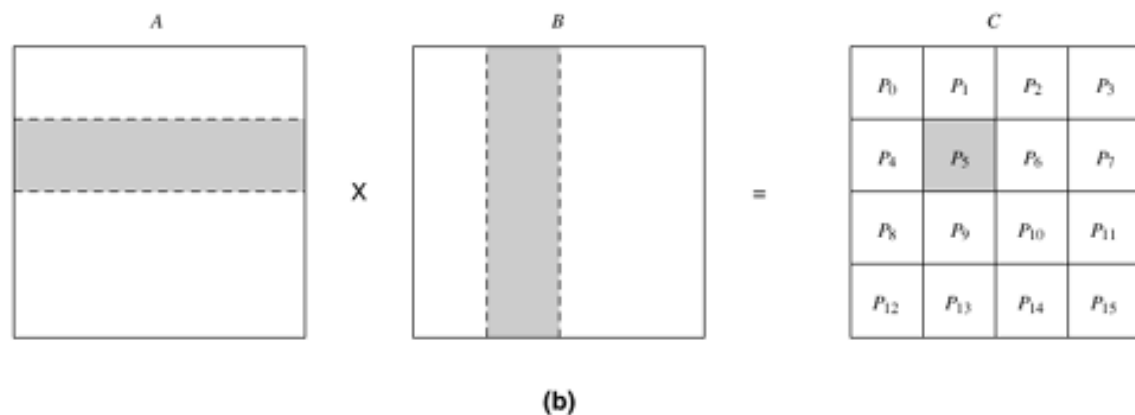
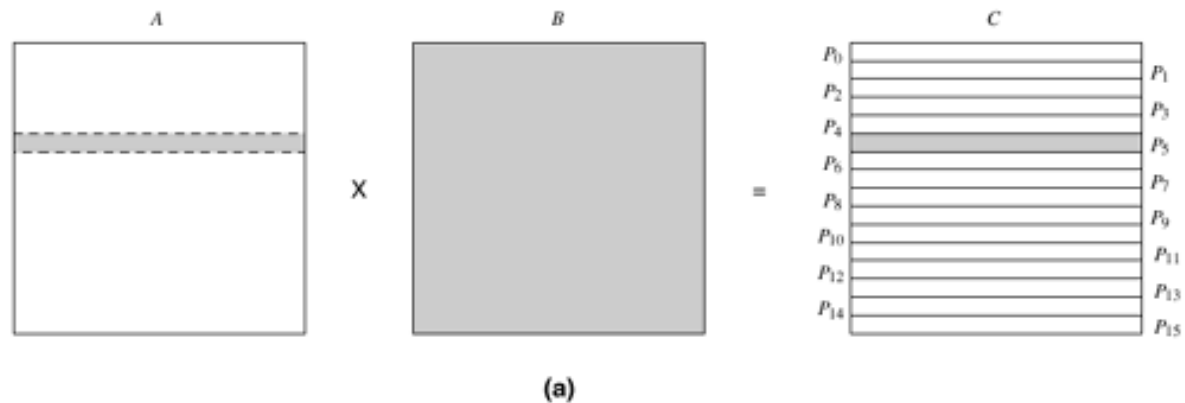
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
-------	-------	-------	-------	-------	-------	-------	-------

一维数据分割块分配映射

P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}

二维数据分割块分配映射

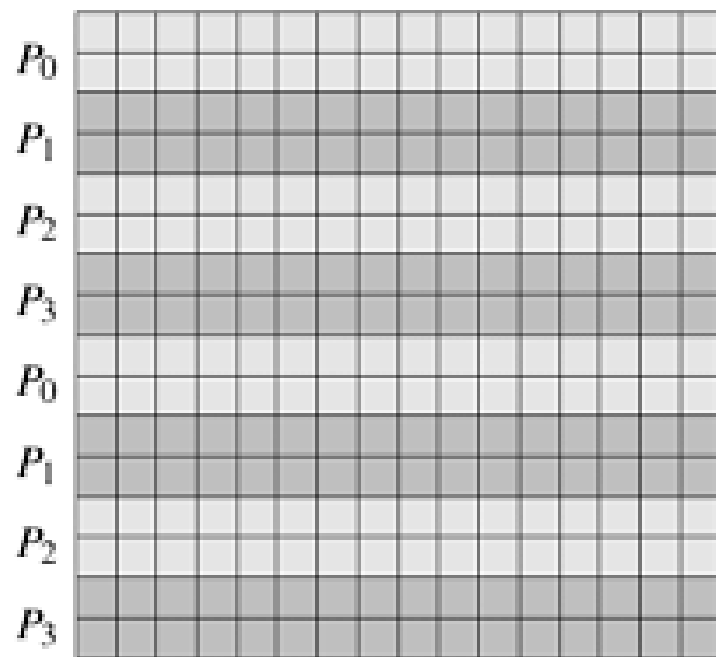


不同的块分割方法导致的共享数据量不同

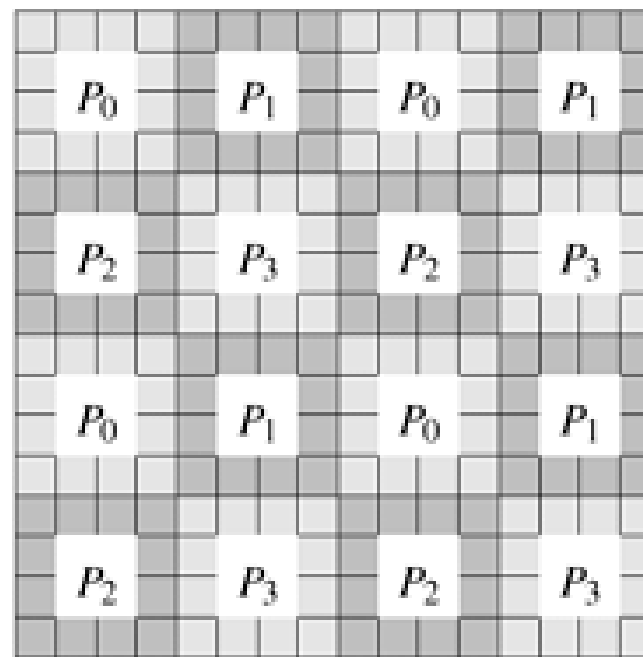
如果拆分出来的块的数量多于进程数量，怎么进行映射？

3.4 Mapping Techniques for Load Balancing

静态映射 - 基于数据 - Cyclic and Block-Cyclic Distributions



(a)



(b)

将数据拆分成足够多的小块，按照顺序给到指定的进程进行计算

假如块分割出来的任务耗时不同，该怎么处理？

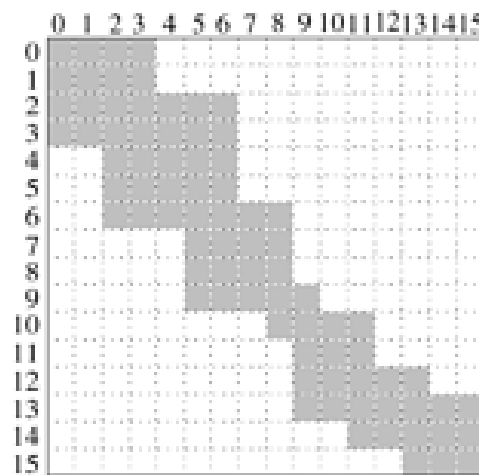
3.4 Mapping Techniques for Load Balancing

静态映射 - 基于数据 - Randomized Block Distributions

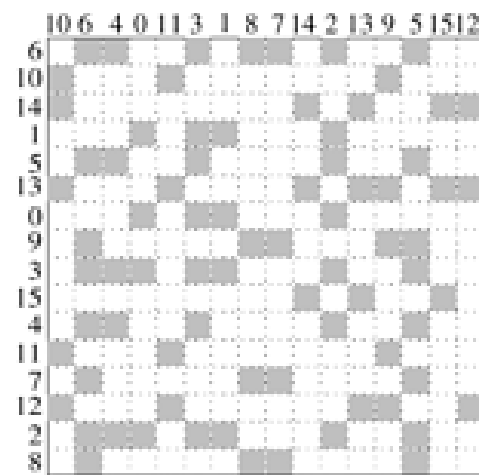
$V = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$

$\text{random}(V) = [8, 2, 6, 0, 3, 7, 11, 1, 9, 5, 4, 10]$

mapping = 8 2 6 0 3 7 11 1 9 5 4 10
 └─┘ └─┘ └─┘ └─┘
 P₀ P₁ P₂ P₃



(a)



(b)

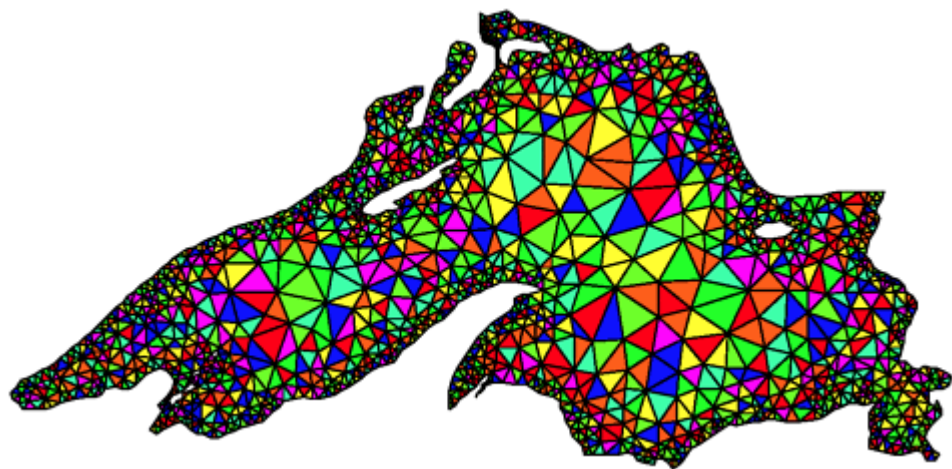
P ₀	P ₁	P ₂	P ₃
P ₄	P ₅	P ₆	P ₇
P ₈	P ₉	P ₁₀	P ₁₁
P ₁₂	P ₁₃	P ₁₄	P ₁₅

(c)

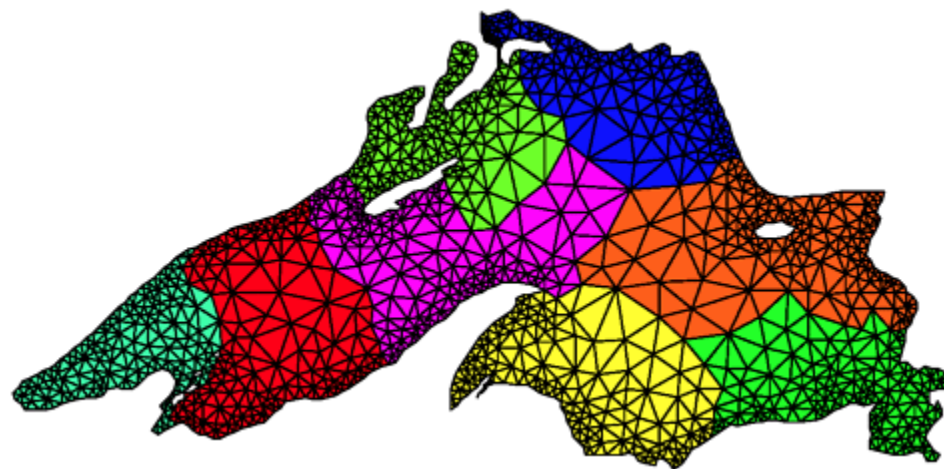
如果数据之间的交互不规则怎么办?

3.4 Mapping Techniques for Load Balancing

静态映射 - 基于数据 - Graph Partitioning



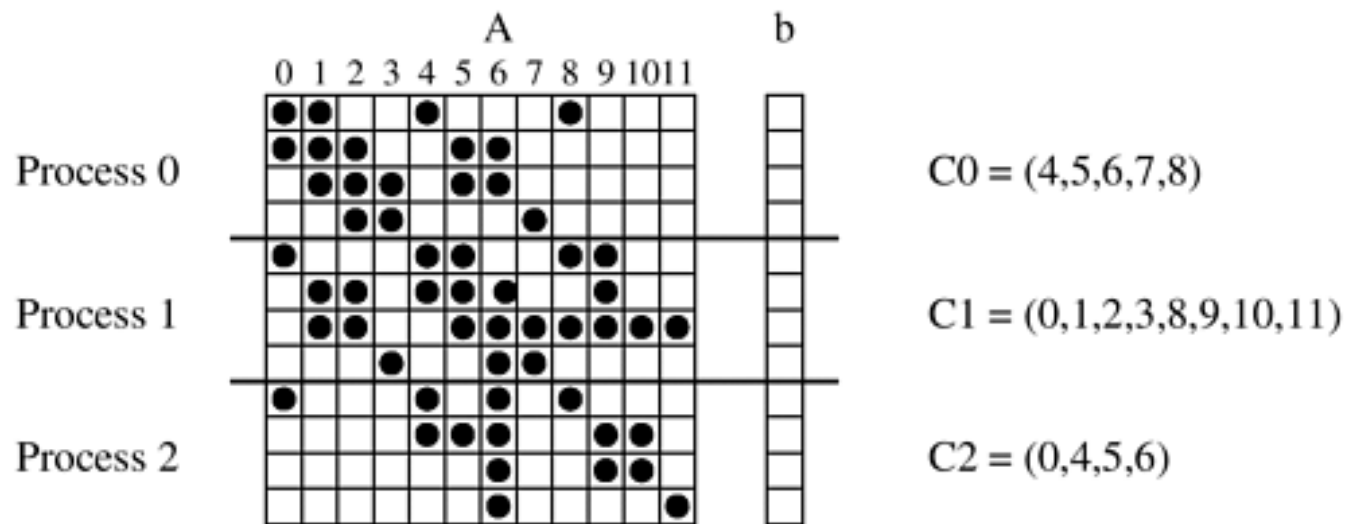
使用随机映射
导致数据不连续，需要大量的任务交互



使用图分区映射
大大降低了任务交互的数据量

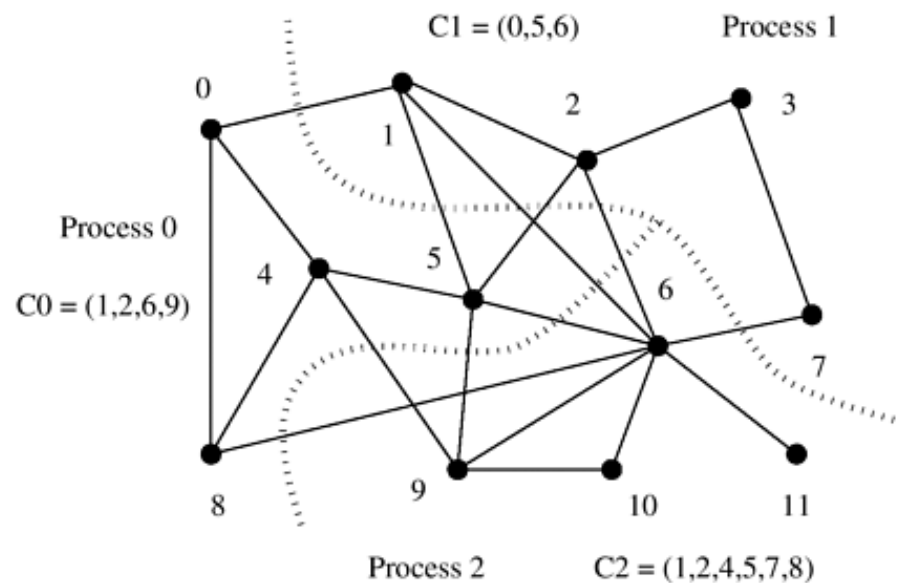
3.4 Mapping Techniques for Load Balancing

静态映射 - 基于任务划分



按照每一行一个任务，循环分给不同的进程

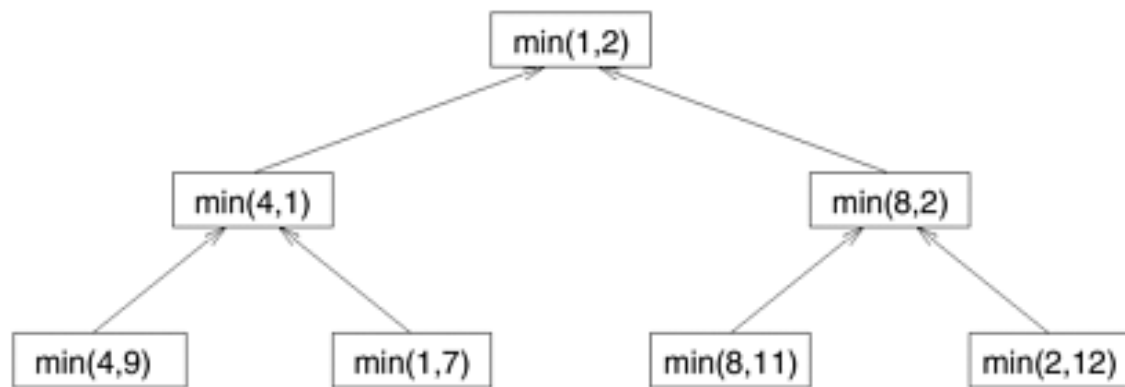
进程C1需要大量的依赖其他进程的数据



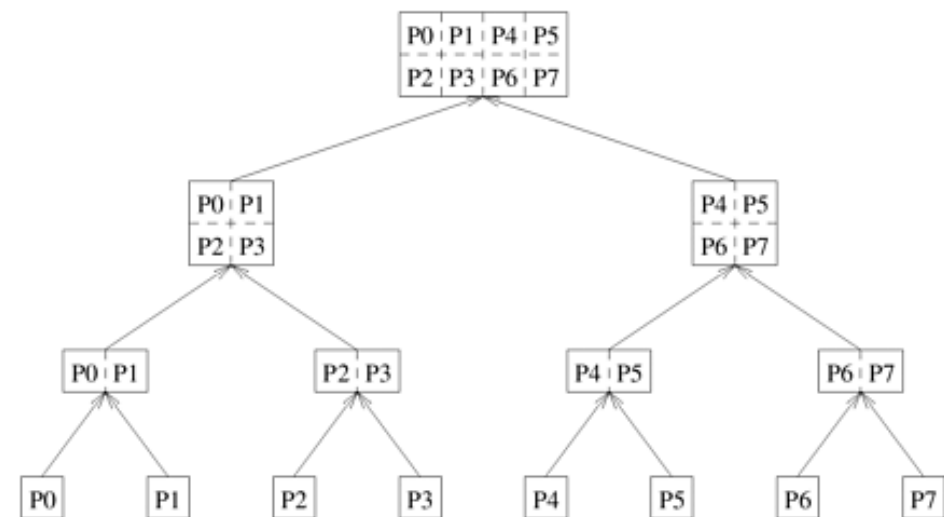
优化后的映射关系

3.4 Mapping Techniques for Load Balancing

静态映射 - 分级映射



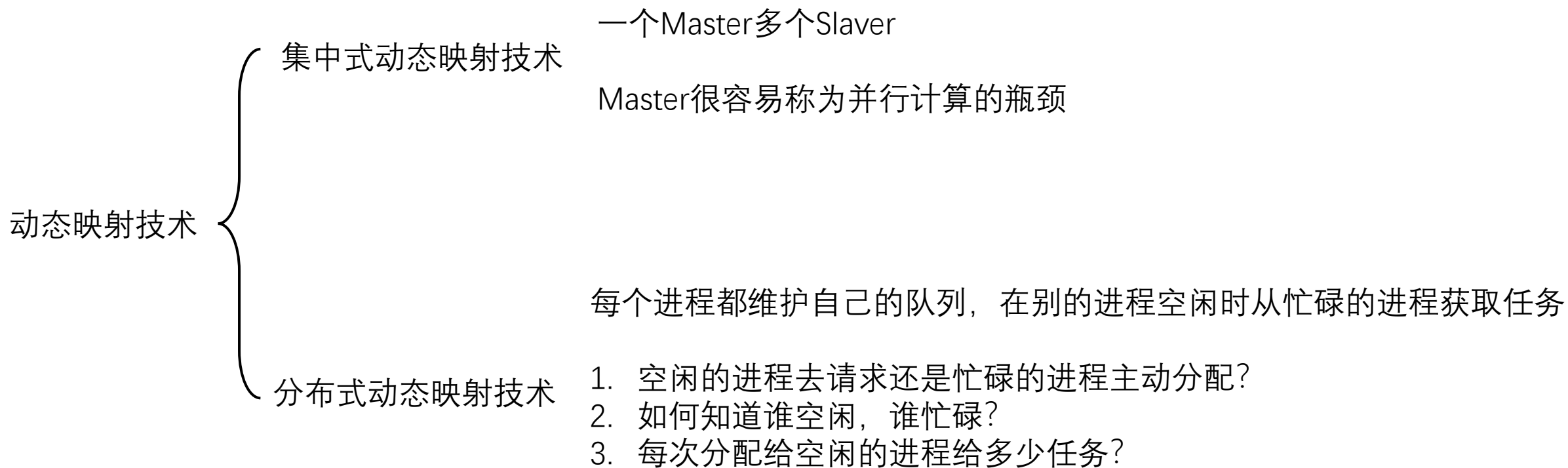
寻找最小值的任务



寻找最小值的任务的分级映射关系

3.4 Mapping Techniques for Load Balancing

动态映射



Chapter 3

Principles of Parallel Algorithm Design

2024/06/16

3.5 Methods for Containing Interaction Overheads

计算加速



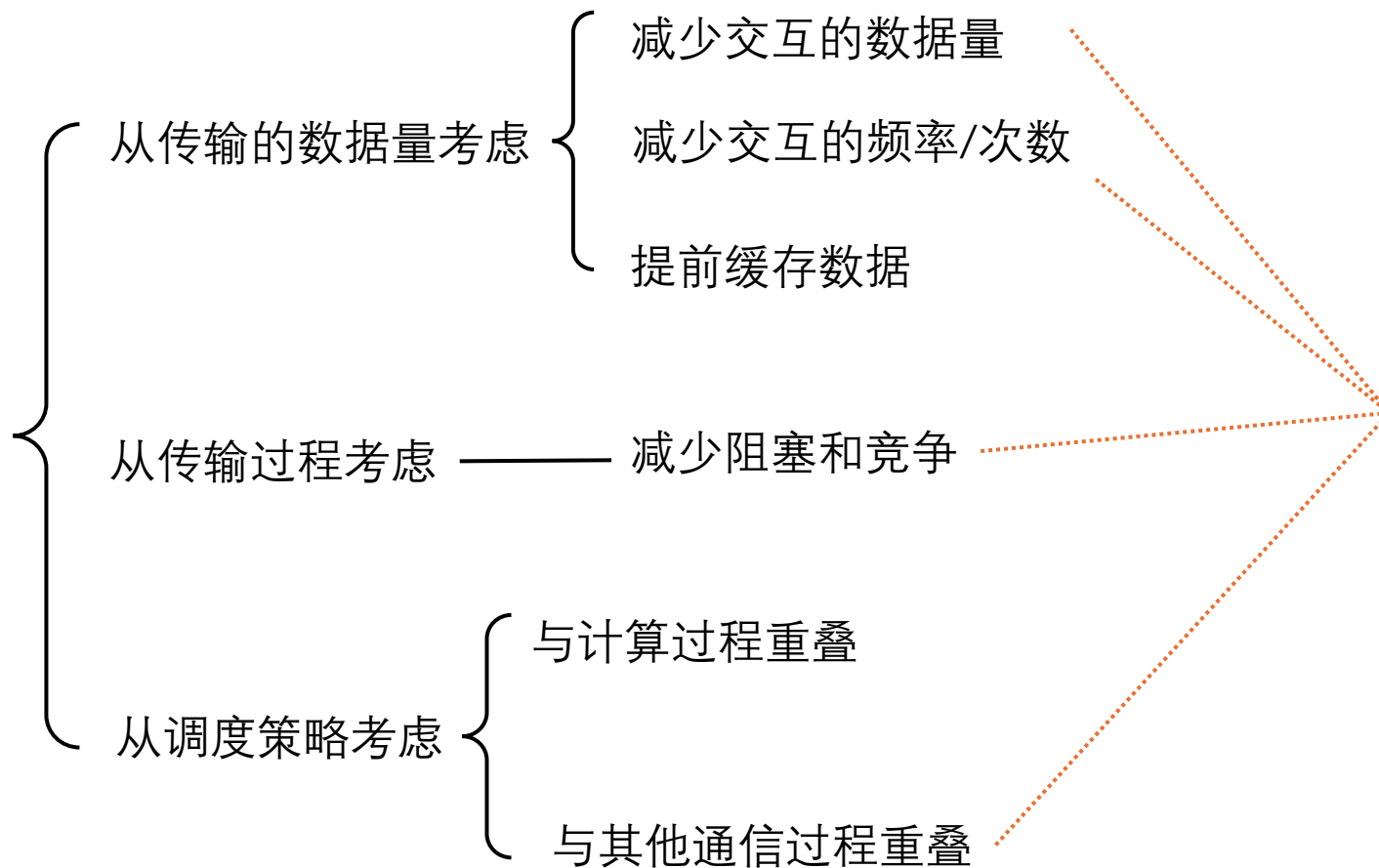
任务拆分 / 映射



额外的交互开销



如何降低
交互造成的开销?

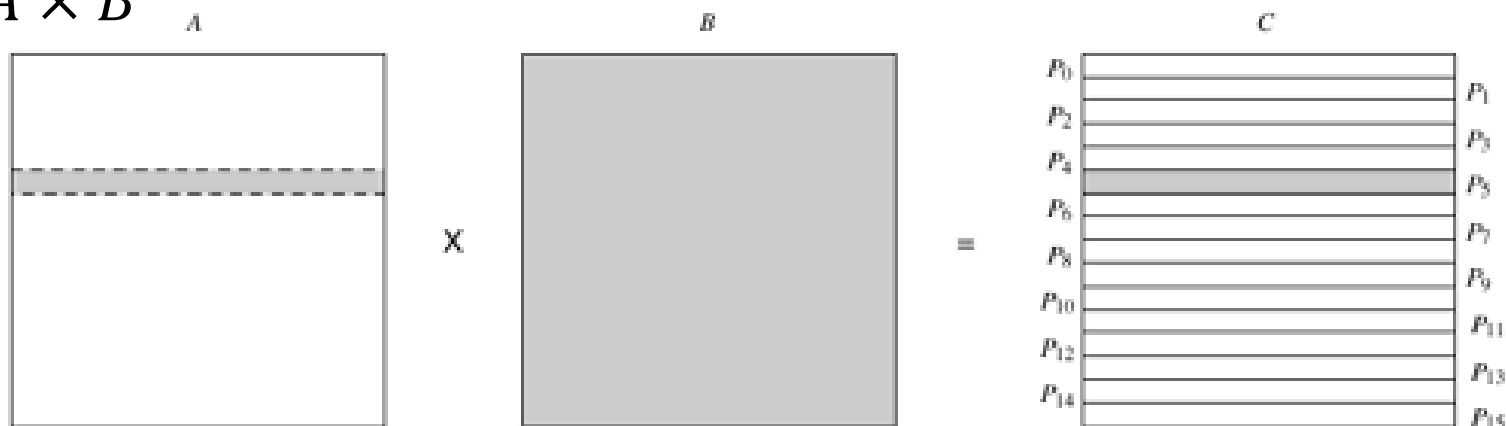


Chapter 4
集合通信算法

3.5.1 从传输数据的角度考虑降低交互开销

目标 $C = A \times B$

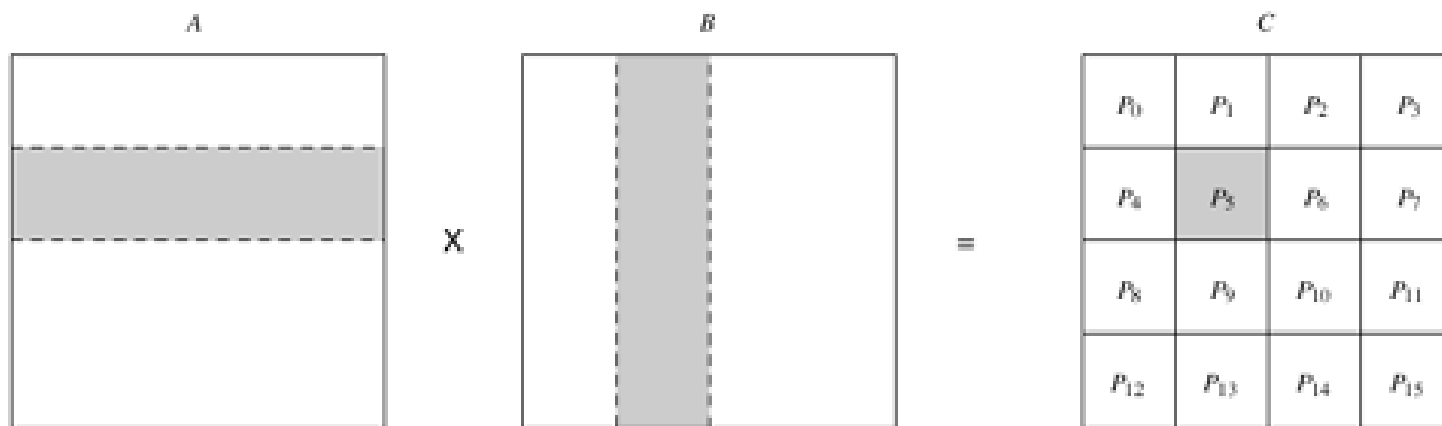
方案1



每次从共享内存中
传输的数据量为

$$n^2/p + n^2$$

方案2



$$2n^2/\sqrt{p}$$

减少交互的数据量

通过改变任务的切分方式，尽可能减小数据的交换量以降低交互的开销

3.5.1 从传输数据的角度考虑降低交互开销

目标 $C = A \cdot B$

P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}

×

P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}

方案一：

每次运算取出对应位置的元素的值到任务中计算
访问 $2n^4/p$ 次

方案二：

一次性把需要运算的子矩阵的所有元素缓存到本地
访问 2 次

减少交互的频率

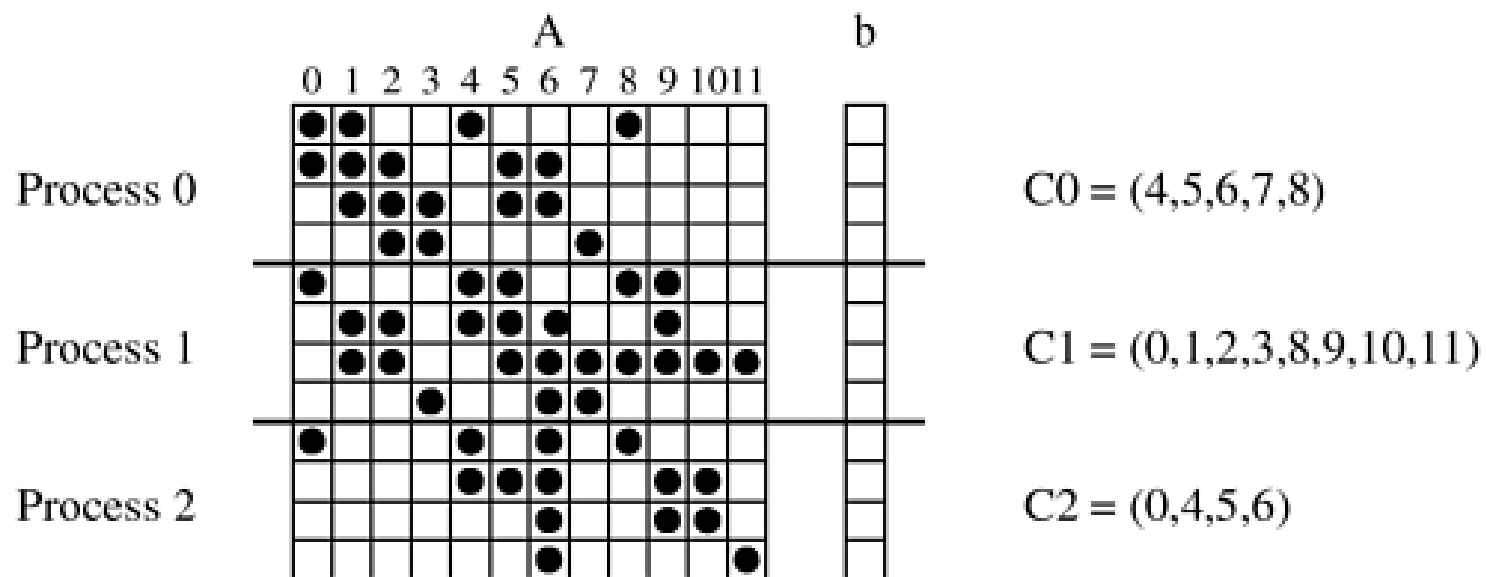
每次通信都会有一个启动时间的开销，因此减少交互的频率可以有效的降低交互开销

问题

这种降低交互开销的方法带来的弊端是需要增加本地的缓存空间

3.5.1 从传输数据的角度考虑降低交互开销

目标 稀疏矩阵乘法 $y = A \times b$



1. 每个Task / Process在进行计算前进行扫描
2. 每个Task / Process可以准确的获知交互目标
3. 在计算开始前将所需的数据储存在本地

提前缓存所需的数据到本地

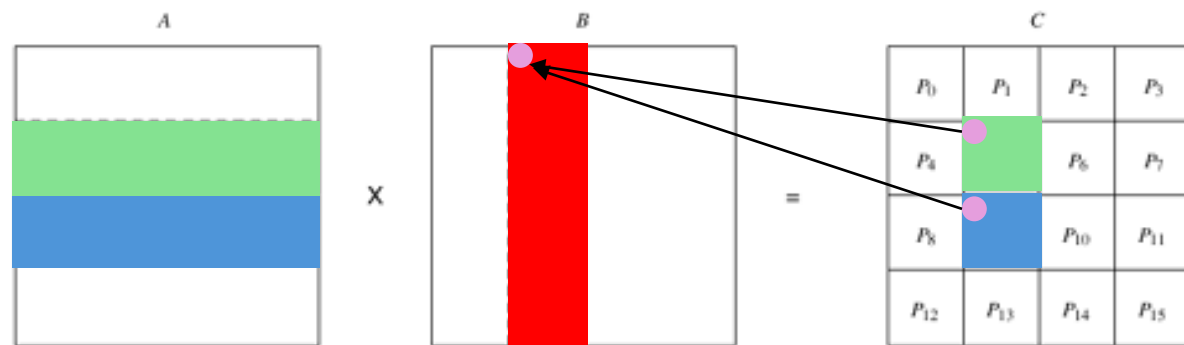
通过改变任务的切分方式，尽可能减小数据的交换量以降低交互的开销



上一章节的降低交互频率的方法是基于任务的数据分解方法进行的

提前缓存的方法适用于多种任务分解方法，在类似这种稀疏矩阵乘法的任务下尤其有效

3.5.2 从传输过程考虑进行阻塞控制策略

目标 $C = A \times B$



Task  和 Task  在计算的时候
都需要访问矩阵B中**同一位置**的数据块，
这种行为可能会造成**访问阻塞**

$$C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,k} \times B_{k,j}$$

不同Task可能同时访问 $B_{k,j}$



$$C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,(i+j+k)\% \sqrt{p}} \cdot B_{(i+j+k)\% \sqrt{p},j}$$

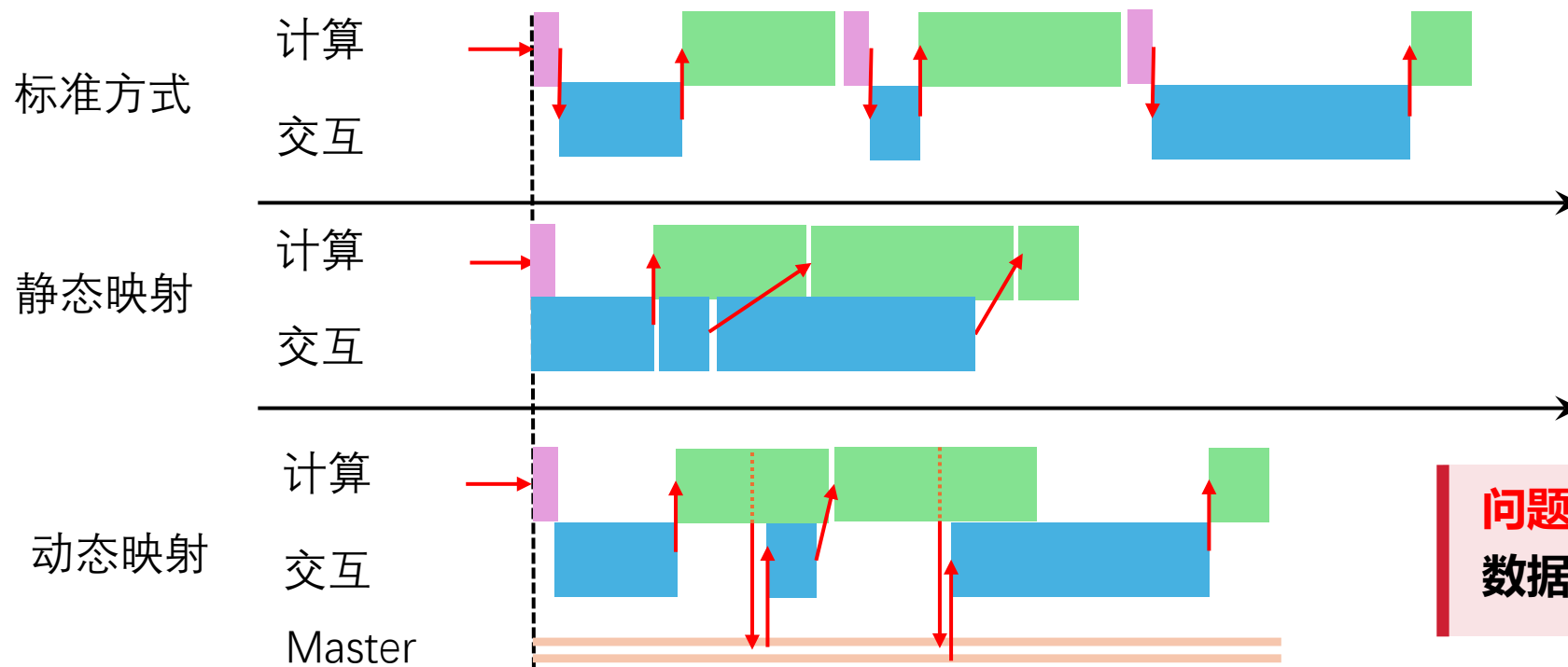
不同Task不会访问 $B_{k,j}$

阻塞控制策略

通过打乱每个Task的计算顺序使得其对内存访问的顺序不同，进而实现无阻塞的交互

在RoCE网络上有其他的阻塞控制方案，例如：TIMELY、FPC、DCQCN、DCTCP、HPCC、PINT等
在IB网络上原生支持使用信令的流控制技术进行阻塞控制和负载均衡

3.5.3 从调度策略考虑降低交互开销

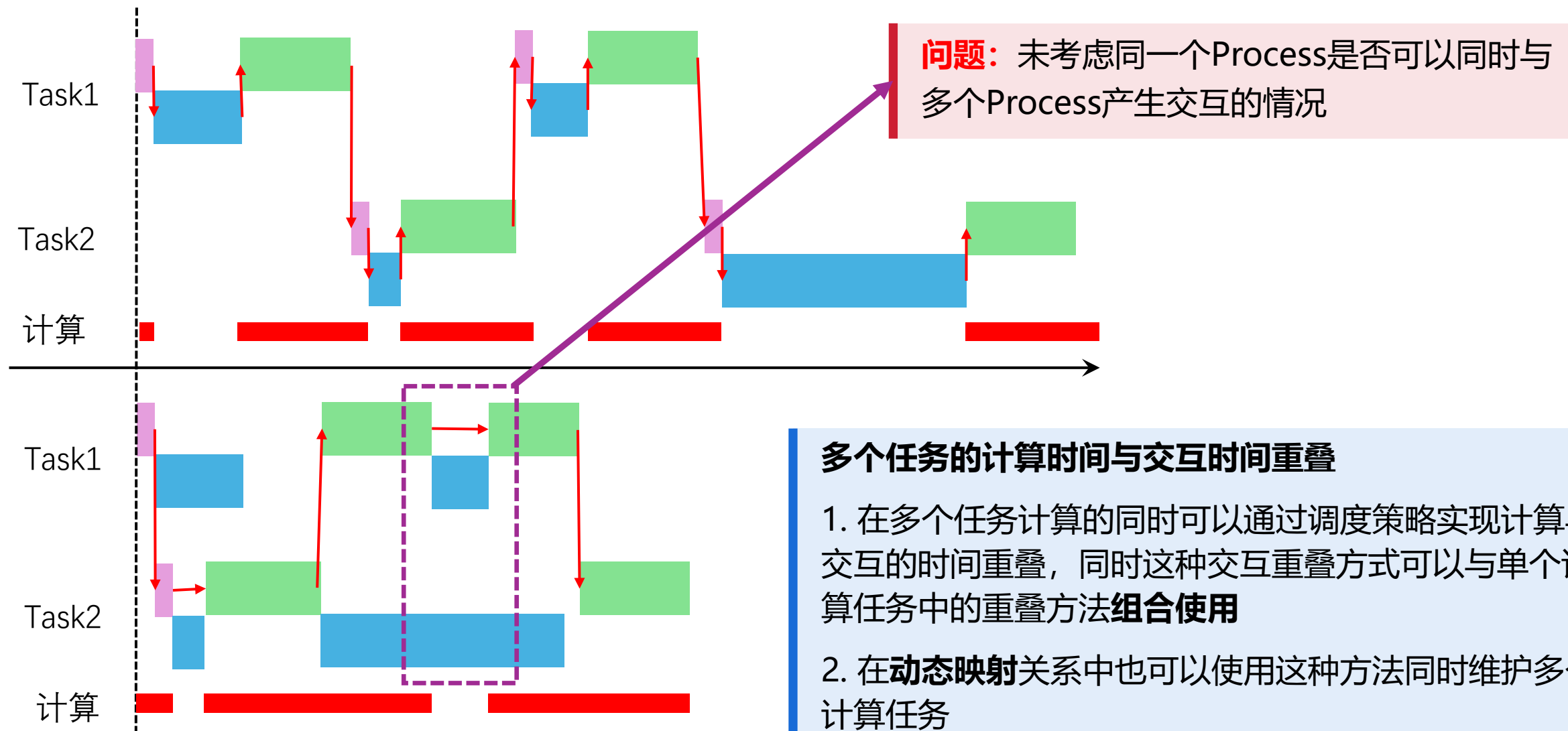


问题: 准确预测任务的**结束时间**或者**数据交互时间**可能较为困难

单个任务的计算时间与交互时间重叠

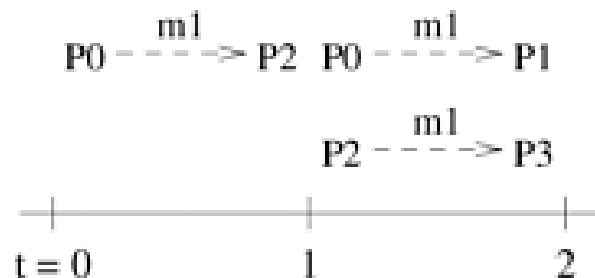
1. 如果是**静态映射**关系, 计算任务开始前可以提前加载所需的数据到本地
2. 如果是**动态映射**关系, 可以在当前任务即将结束前提前向Master / 其他Process获取下一个任务的信息

3.5.3 从调度策略考虑降低交互开销

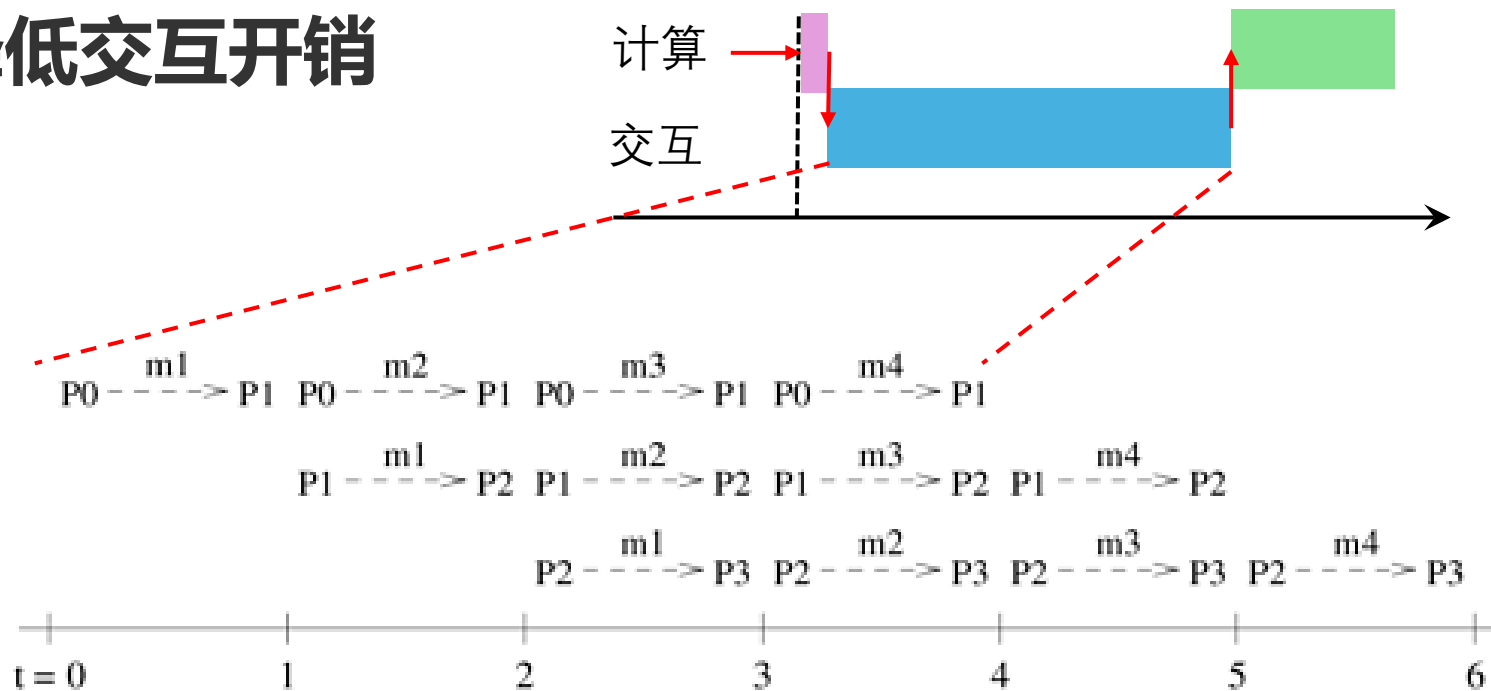


3.5.3 从调度策略考虑降低交互开销

目标 One-to-All Broadcast



图a. 从原始的3步优化到2步交互



图b. 常见的重叠需要6步

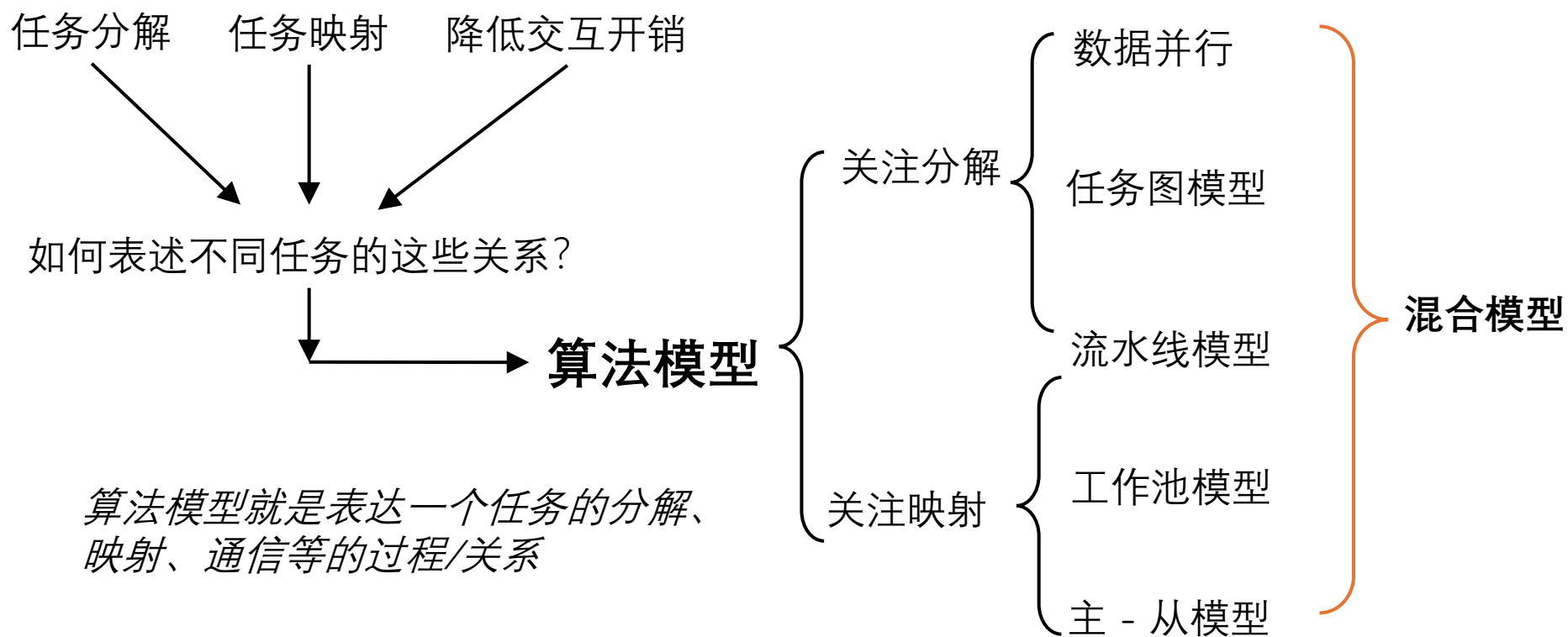
与其他交互时间重叠

1. 与其他交互时间重叠可以有效的降低交互开销 (图a)
2. 选择合适的重叠策略可以达到不同的交互开销效果 (图b)

提示

关于对图b的优化效果, 可以参考
Chapter 4 集合通信的内容

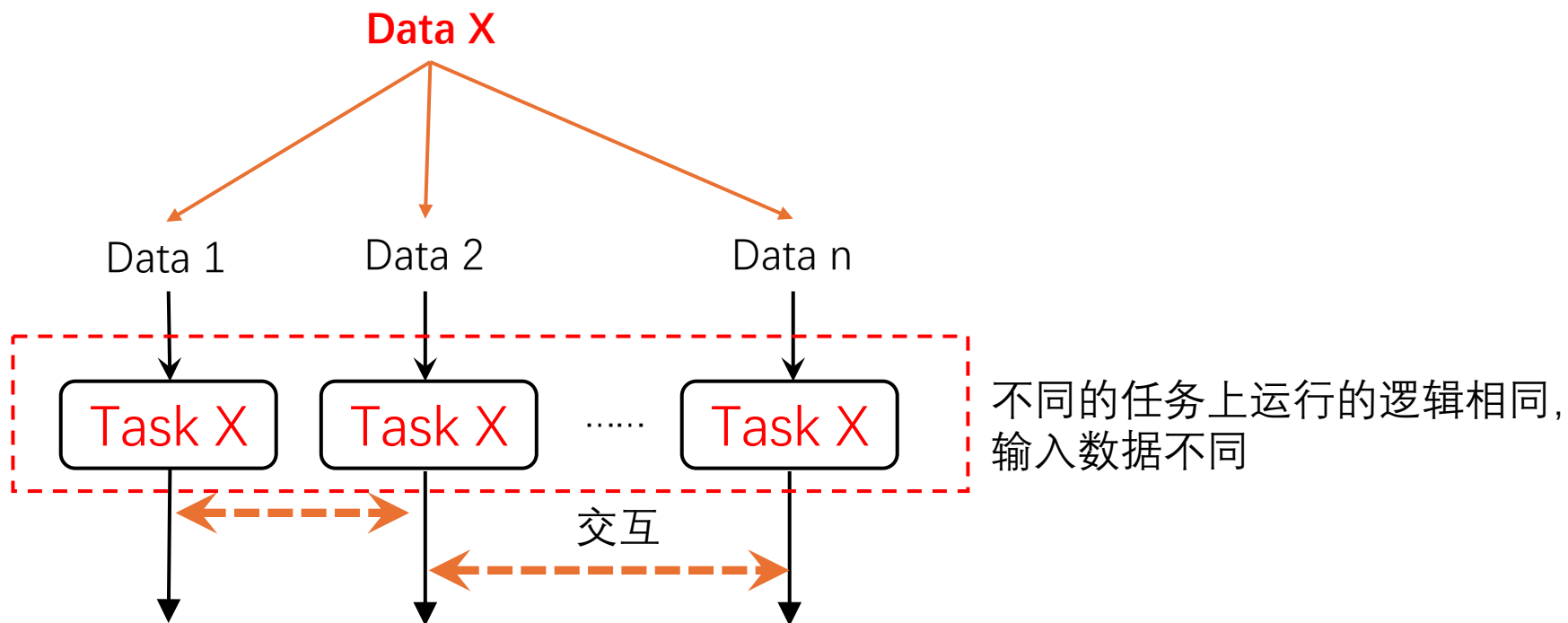
3.6 Parallel Algorithm Models



算法模型通常是通过选择分解和映射技术，并采用适当的策略来最小化交互作用，从而构建并行算法的一种方法。

An algorithm model is typically a way of structuring a parallel algorithm by selecting a decomposition and mapping technique and applying the appropriate strategy to minimize interactions.

3.6 关注分解（数据并行）



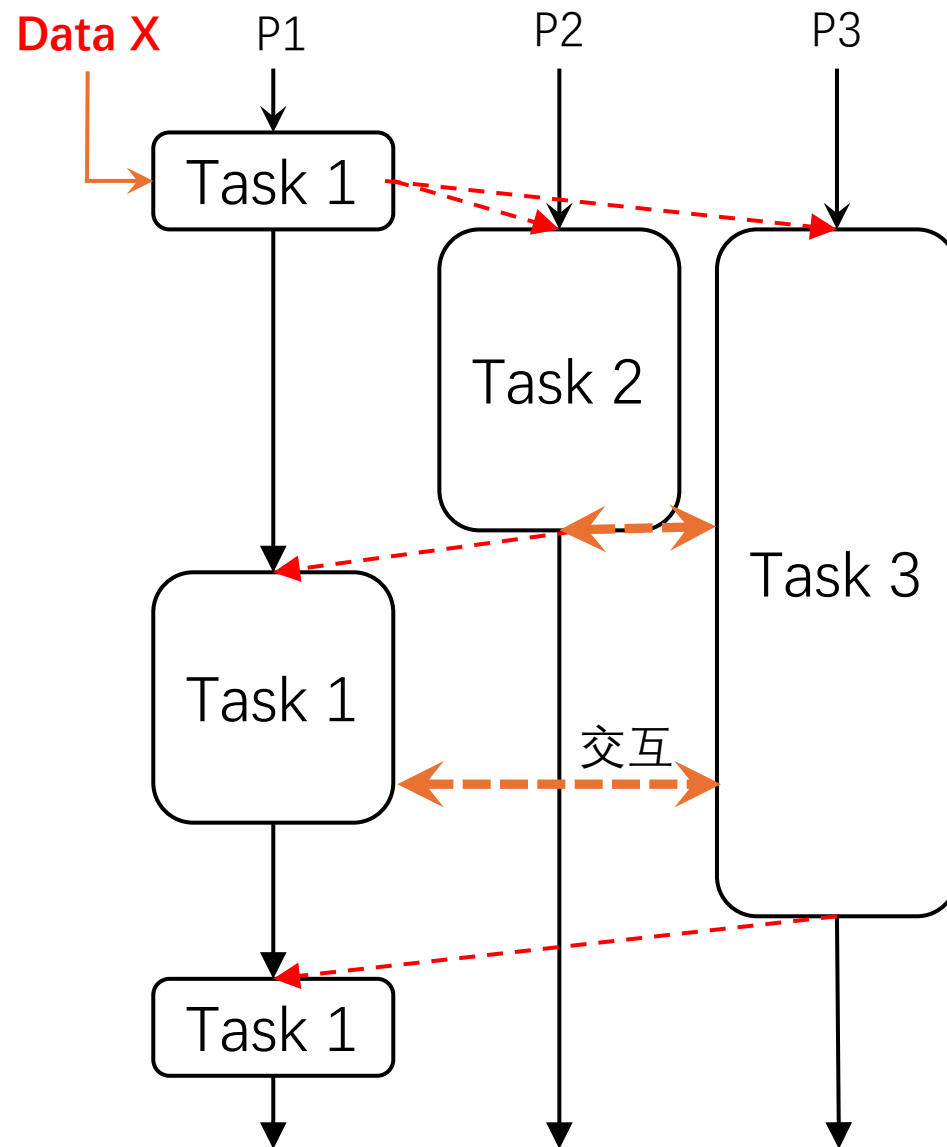
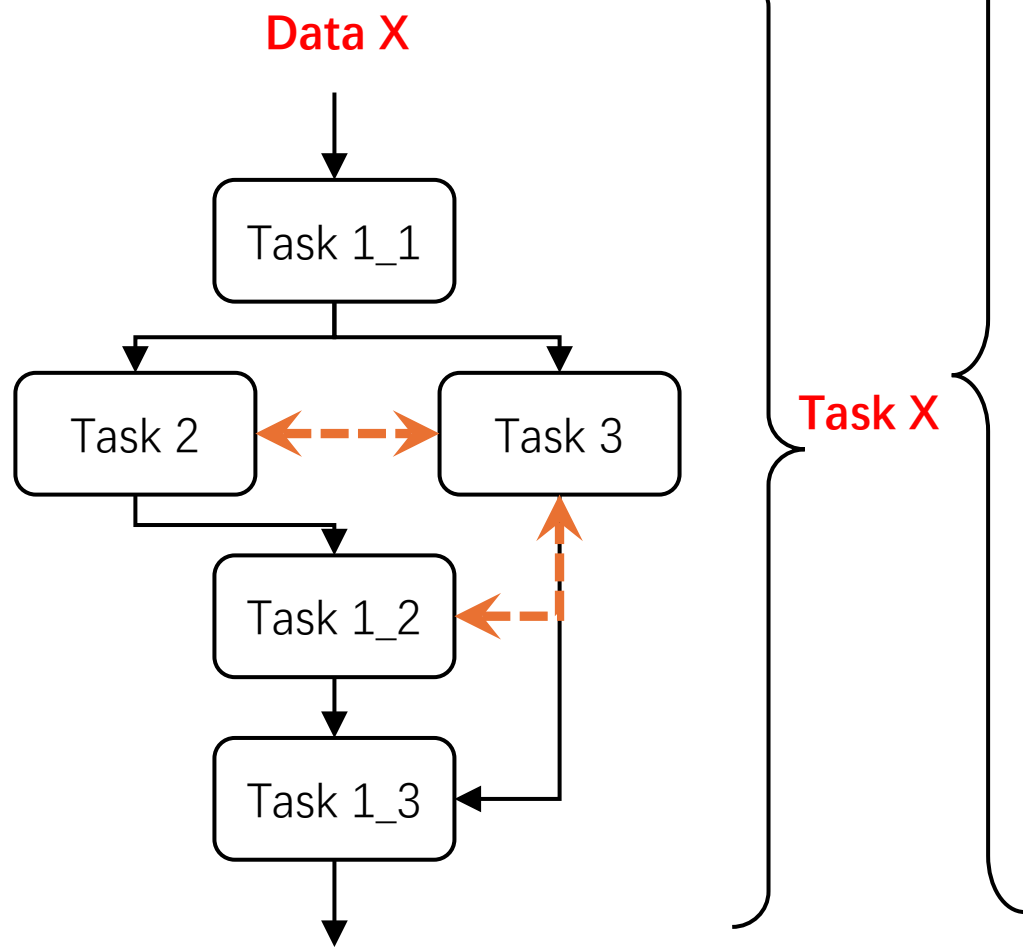
Task X 表示一个待解决的大的任务，后续中可以拆分成多个Task，例如Task 1、Task 2等

Data X 表示Task X对应的完整的数据对象，可以拆分成多个Data，例如Data 1、Data 2等

3.6 关注分解 (任务图)

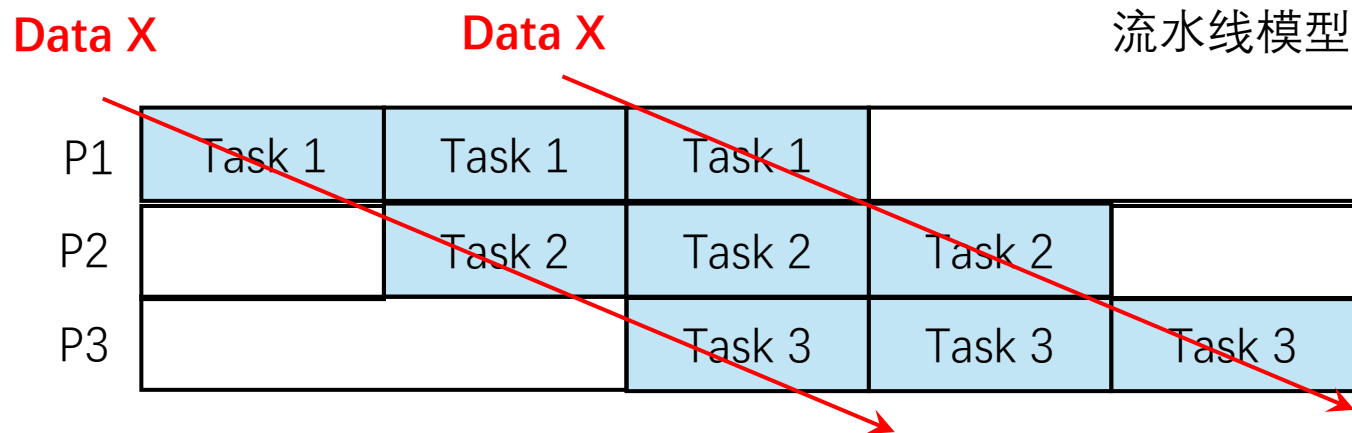
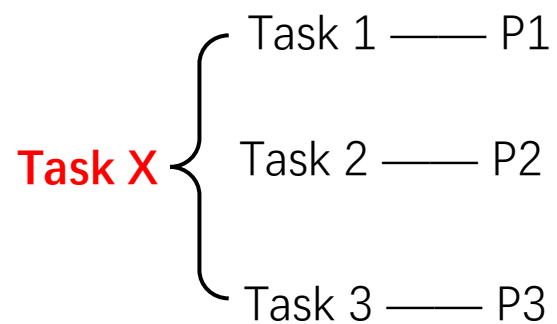
任务直接存在依赖
任何任务都可以视为任务图模型

任务图
不关注映射关系

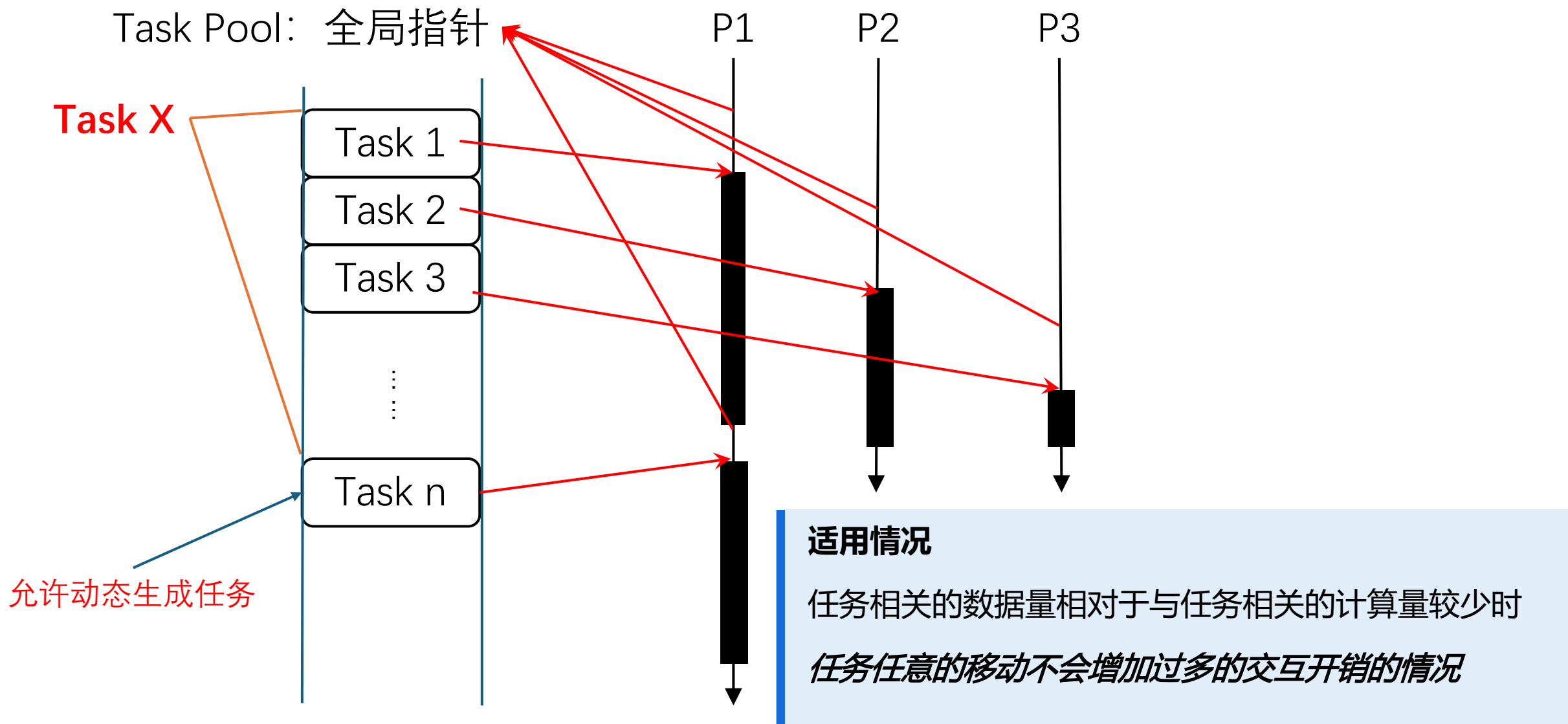


3.6 关注分解 / 关注映射（流水线模型）

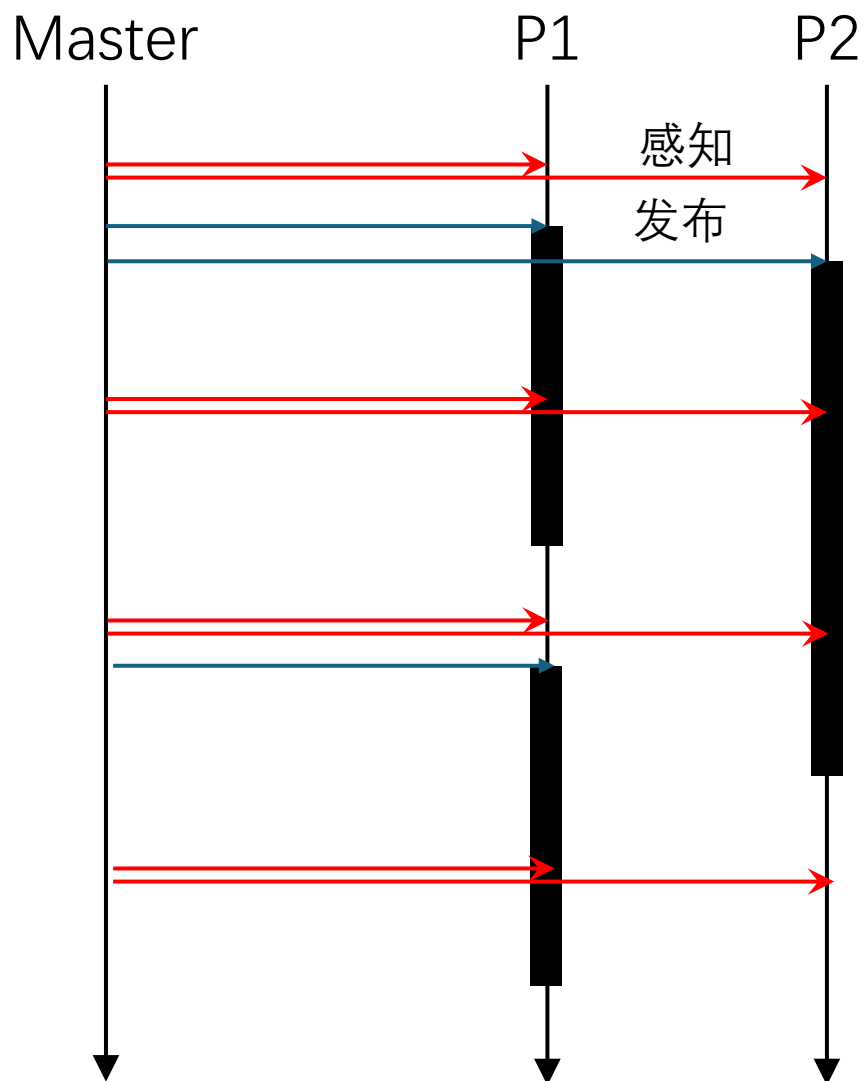
任务的分解



3.6 关注映射（任务池模型）



3.6 关注映射（主-从模式）



主-从模式不同于任务池，**Master需要感知和评估任务的状态信息**，由Master将任务分配给对应的进程进行预算

这种主从模式可以扩展为**分级主从模式**，由Master管理多个SubMaster，再进而管理多个Worker

主从模式可以是**集中式**的，也可以是**分布式**的，分布式主从模式可参考任务到进程的映射章节

问题

Master的性能容易称为瓶颈

3.1 Preliminaries

3.1.2 任务交互 Task-Interaction

总结

问题

提示