第10章 多处理器和实时调度

- 主要内容
  - 10.1 多处理器调度 ★★★★★
  - 10.2 实时调度 ★★★★★
  - 10.3 Linux调度（自学）
  - 10.4 UNIX SVR4调度（自学）
  - 10.5 FreeBSD调度器（自学）
  - 10.6 Windows调度（自学）
  - 10.7 Linux虚拟机进程调度（自学）

10.1 多处理器调度 ★★★★★

多处理器系统分类

- 松耦合、分布式多处理器、集群
  - 由一系列相对自治的系统组成，每个处理器有自己的内存和I/O通道。
- 专门功能的处理器
  - 有一个通用的主处理器，专用处理器受主处理器的控制，并给主处理器提供服务。
- 紧耦合多处理
  - 由一系列共享同一个内存并在操作系统完全控制下的处理器组成。

10.1.1 粒度

- 同步粒度和进程

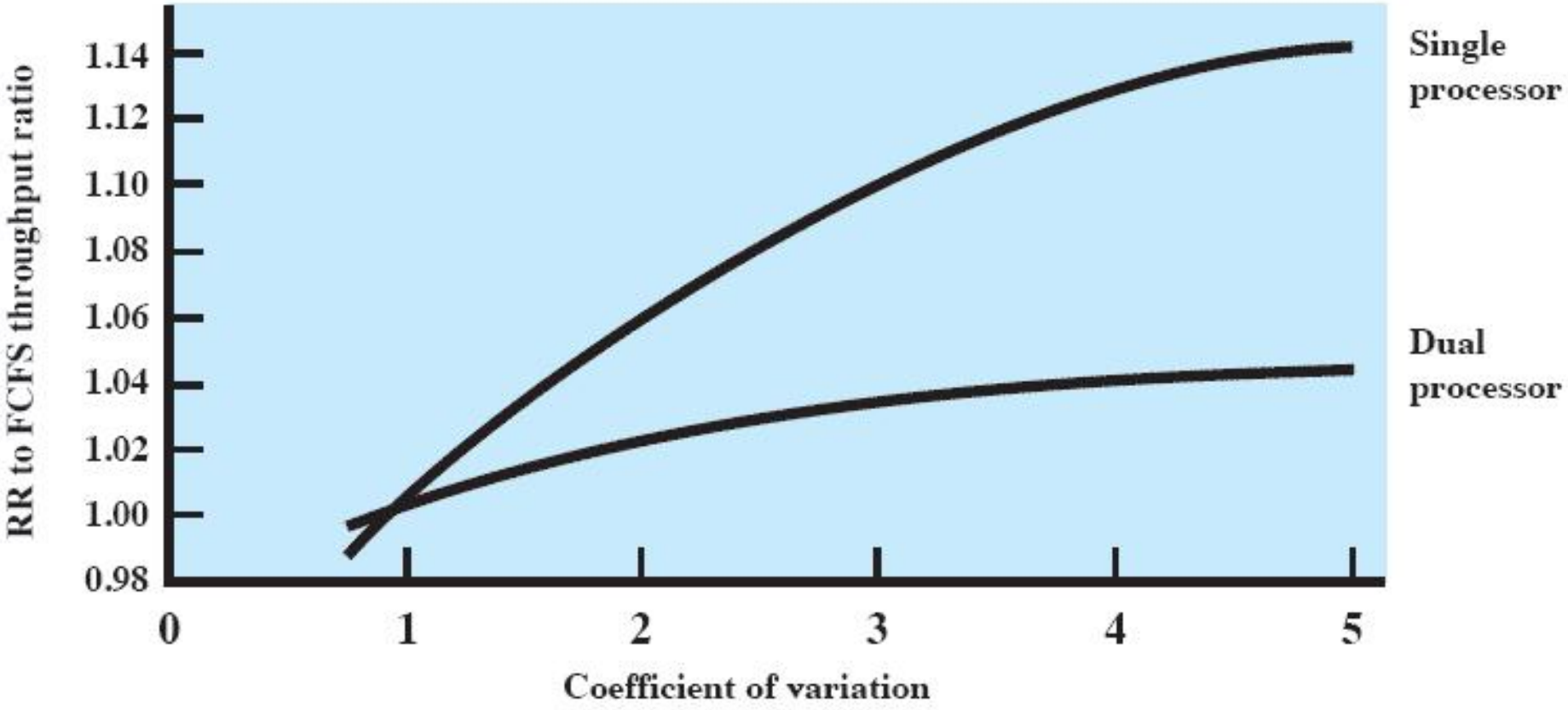
粒度大小	说明	同步间隔
细	单指令流中固有的并行	<20
中等	在一个单独应用中的并行处理或多任务处理	20~200
粗	在多道程序环境中并发进程的多处理	200~2000
非常粗	在网络节点上进行分布处理，以形成一个计算环境	2000~1M
无约束	多个无关进程	不适用

10.1.2 设计问题

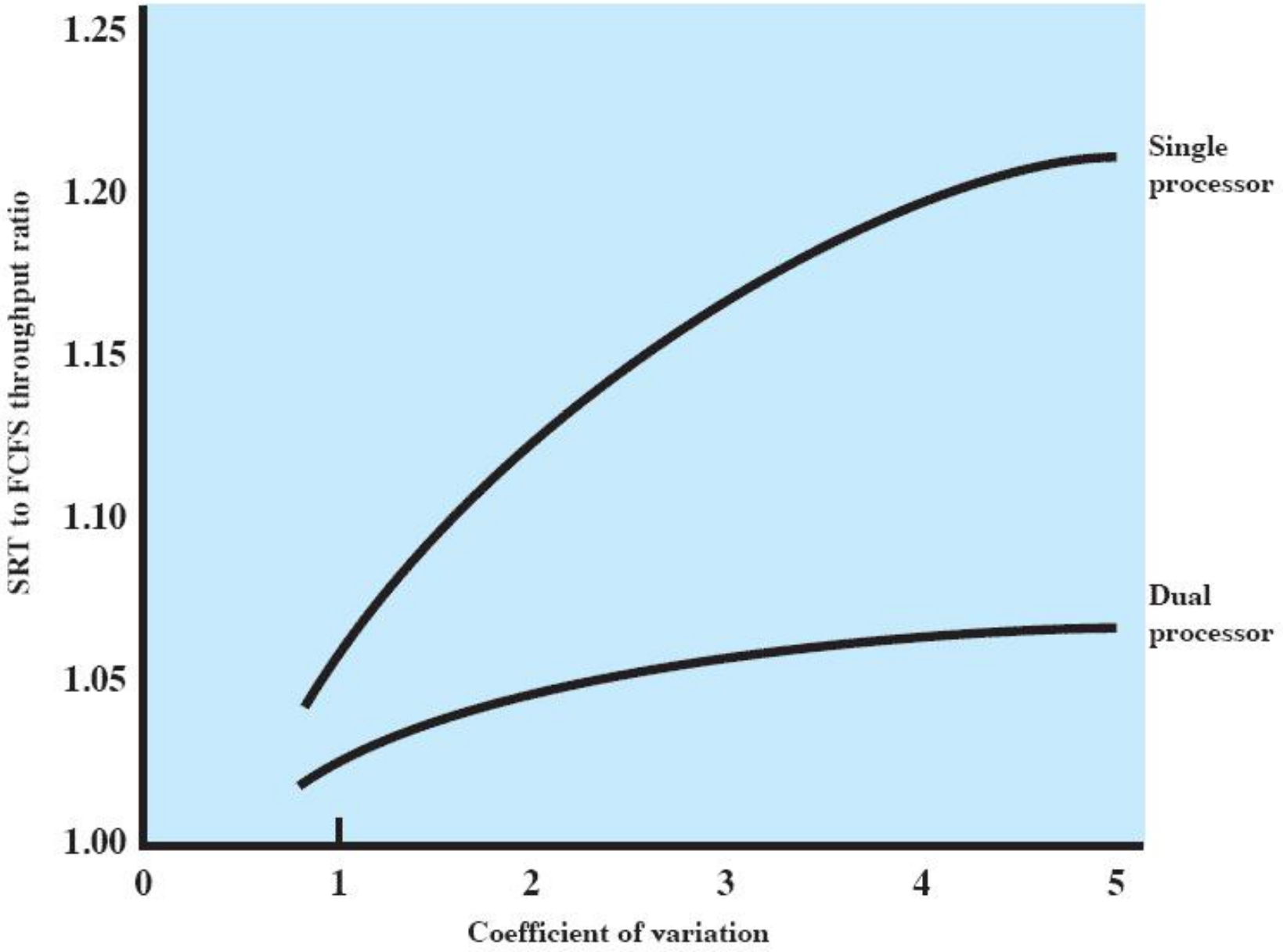
- 把进程分配到处理器
  - 静态分配（每个处理器维护一个专门的短程队列）、动态分配（所有进程都进入一个全局队列）
  - 主从式（OS核心功能总在某个特定处理器上运行）、对等式（OS内核可以在任意处理器上运行）
- 在单个处理器上使用多道程序设计
  - 传统多处理器处理粗粒度或无约束同步粒度，单处理器使用多道程序设计。
  - 运行在多处理器系统中的中等粒度应用程序，更加关注如何能为应用提供更好的平均性能，单个处理器是否多道并不重要。
- 一个进程的实际分派：考虑多处理器时，使用简单的调度方法会更有效，降低开销

10.1.3 进程调度

- 多处理器情况下，调度原则的选择没有在单处理器中显得重要。
- 多处理系统中一般使用简单的FCFS或者在静态优先级方案中使用FCFS。



(a) Comparison of RR and FCFS



(b) Comparison of SRT and FCFS

10.1.4 线程调度

- 负载分配
  - 系统维护一个就绪线程的全局队列，每个处理器只要空闲就从队列中选择一个线程。
- 组调度
  - 一组相关的线程基于一对一的原则，同时调度到一组处理器上运行。
- 专用处理器调度
  - 组调度的一种极端形式，在一个应用程序执行期间，把一组处理器专门分配给这个应用程序。
- 动态调度
  - 某些应用程序允许动态地改变进程中线程数目，需要动态调度。操作系统负责分配处理器给作业，作业自行调度。

10.1.4 线程调度

- 负载分配
  - 系统维护一个就绪线程的全局队列，每个处理器只要空闲就从队列中选择一个线程。
  - 三种不同的负载分配方案：
    - 先来先服务
    - 最少线程数优先
    - 可抢占的最少线程数优先

10.1.4 线程调度

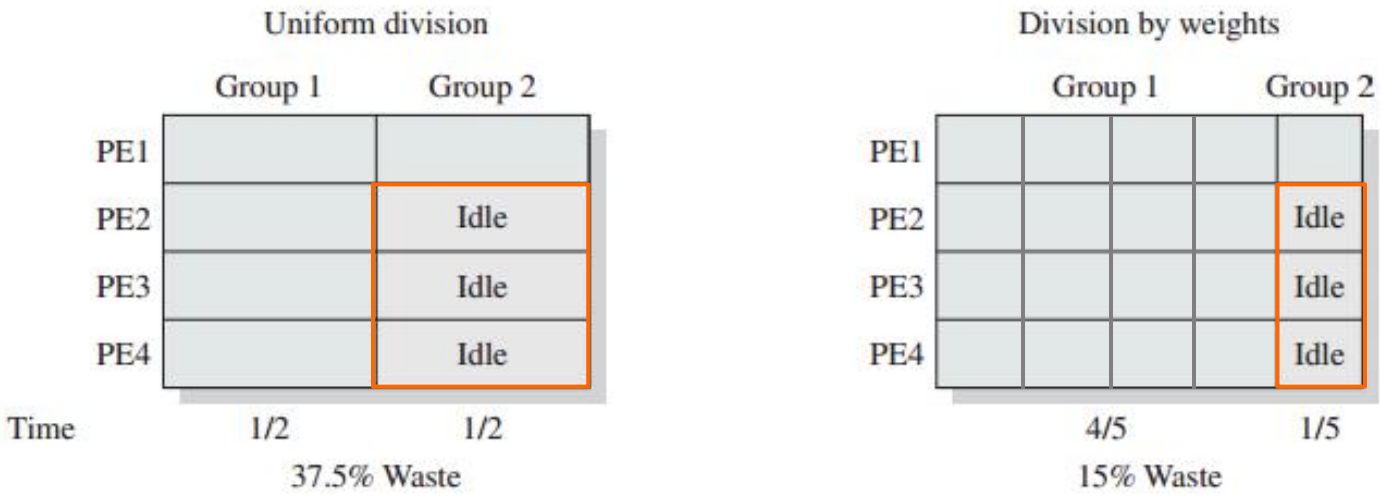
- 负载分配
  - 系统维护一个就绪线程的全局队列，每个处理器只要空闲就从队列中选择一个线程。
  - 优点：
    - 把负载均分到所有的可用处理器上，保证了处理器效率的提高。
    - 不需要一个集中的调度程序，一旦一个处理器空闲，操作系统的调度程序就可以运行在该处理器上以选择下一个运行的线程。
    - 运行线程的选择可以采用各种可行的策略（雷同与前面介绍的各种进程调度算法）。

10.1.4 线程调度

- 负载分配
  - 系统维护一个就绪线程的全局队列，每个处理器只要空闲就从队列中选择一个线程。
  - 不足：
    - ☒就绪线程队列必须被互斥访问， 当系统包括很多处理器， 并且同时有多个处理器同时挑选运行线程时，它将成为性能的瓶颈。
    - ☒被抢占的线程很难在同一个处理器上恢复运行， 因此当处理器带有高速缓存时，恢复高速缓存的信息会带来性能的下降。
    - ☒如果所有的线程都被放在一个公共的线程池中的话，所有的线程获得处理器的机会是相同的。如果一个程序的线程希望获得较高的优先级， 进程切换将导致性能的折衷。

10.1.4 线程调度

- 组调度
  - 一组相关的线程基于一对一的原则，同时调度到一组处理器上运行。
  - 优点：
    - ☒组内的进程相关或大致平等时，同步阻塞会减少，且可能只需要很少进程切换，因此性能会提高。
    - ☒由于一次性同时调度一组处理器，调度的代价也将减少。
  - 引发对处理器的分配要求



10.1.4 线程调度

- 专用处理器调度
  - 组调度的一种极端形式，在一个应用程序执行期间，把一组处理器专门分配给这个应用程序。
  - 对于高度并行的计算机系统来说，可能包括几十或数百个处理器，它们完全可以不考虑处理器的使用效率，而集中关注于提高计算效率。处理器专派调度算法适用于此类系统的调度。
  - 在一个程序的生命周期中避免进程切换会加快程序的执行速度。
  - 这一调度算法追求的是通过高度并行来达到最快的执行速度。
  - 限制活跃线程数量，使其不超过处理器的数量。

10.1.4 线程调度

- 动态调度
  - 某些应用程序允许动态地改变进程中线程数目，需要动态调度。操作系统负责分配处理器给应用进程，应用进程在分配给它的处理器上执行可运行线程的子集，哪一些线程应该执行，哪一些线程应该挂起（当然系统可能提供一组缺省的运行库例程）。
  - 在这一算法中，当一个进程达到或要求新的处理器时，OS调度程序主要限制处理器的分配，并且按照下面的步骤处理：
    - ☒如果有空闲的处理器，满足要求。
    - ☒否则，对于新到达进程，则从当前分配了一个以上处理器的进程中分一个处理器给该进程。
    - ☒如果一部分要求不能被满足， 则保留申请直到出现可用的处理器或要求取消。
    - ☒当释放了一个或多个处理器后， 扫描申请处理器的进程队列， 按照先来先服务的原则把处理器逐一分配给每个申请进程直到没有可用处理器。

10.2 实时调度 ★★★★★

10.2.1 背景

- 实时计算
  - 系统的正确性不仅取决于计算的逻辑结果，还依赖于产生结果的时间。
- 实时任务
  - 硬实时任务、软实时任务
  - 周期性任务、非周期性任务
- 实时系统应用的例子
  - 目前：实验控制、过程控制、机器人、空中交通管制、电信、军事指挥与控制系统。
  - 下一代：自动驾驶汽车、具有弹性关节的机器人控制器、智能化生产中的系统查找、空间站和海底勘探。

10.2.2 实时操作系统的特点

要求

- 可确定性
  - 按照固定的、预先确定的时间或时间间隔执行操作。
- 可响应性
  - 为中断提供服务的时间。
- 用户控制
  - 允许用户细粒度地控制任务优先级，指定一些特性等。
- 可靠性
  - 性能的损失或降低可能产生灾难性的后果。
- 故障弱化操作
  - 系统在故障时尽可能多地保存其性能和数据的能力。

特征

- 快速的进程或线程切换。
- 体积小（只具备最小限度的功能）。
- 迅速响应外部中断的能力。
- 通过诸如信号量、信号和时间之类的进程间通信工具，实现多任务处理。
- 使用特殊的顺序文件，可以快速存储数据。
- 基于优先级的抢占式调度。
- 最小化禁止中断的时间间隔。
- 用于使任务延迟一段固定的时间或暂停/恢复任务的原语。
- 特别的警报和超时设定。

10.2.3 实时调度

考虑的问题

- 一个系统是否执行可调度性分析；
- 如果执行，是静态的还是动态的；
- 分析结果自身是否根据在运行时分派的任务产生一个调度或计划。

调度算法

- 静态表法
  - 执行关于可行调度的静态分析。分析的结果是一个调度，它用于确定在运行时一个任务何时必须开始执行。
- 静态优先级抢占法
  - 执行一个静态分析，但是没有制定调度，通过给任务制定优先级，使用传统的基于优先级的抢占式调度。
- 基于动态规划调度法
  - 在运行时动态地确定可行性，可行性分析的结果是一个调度或规划，可用于确定何时分派这个任务。
- 动态尽力调度法
  - 不执行可行性分析。系统试图满足所有的最后期限，并终止任何已经开始运行但错过最后期限的进程。

10.2.4 限期调度 ★★★★★

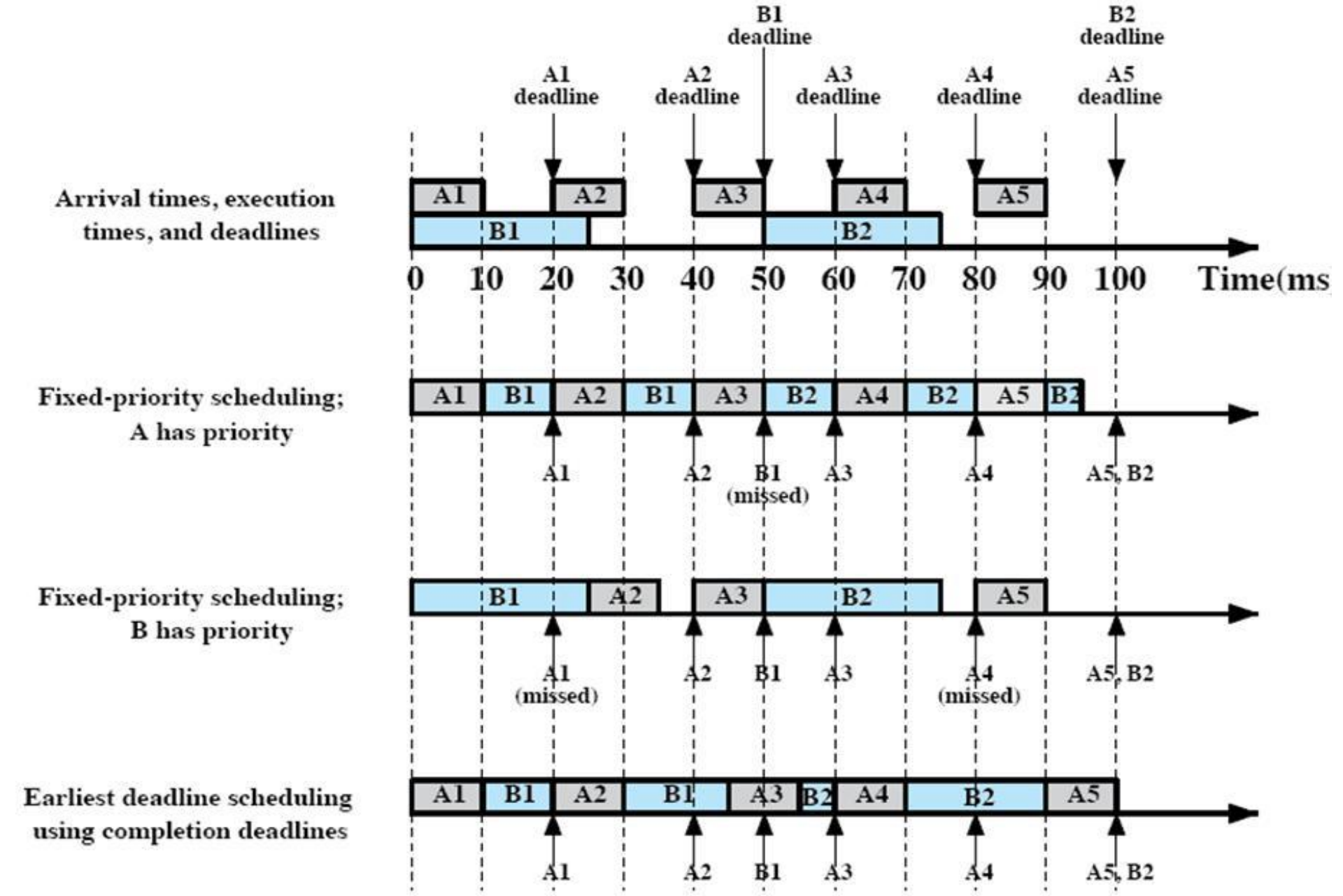
- 实时应用程序通常并不关注绝对速度，它关注的是在最有价值的时间完成（或启动）任务。
- 限期调度：最早最后期限优先。
- 最后期限
  - 启动最后期限：任务必须开始的时间。
  - 完成最后期限：任务必须完成的时间。

例1：具有完成最后期限的周期性任务调度

两个周期性任务的执行简表

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
•	•	•	•
•	•	•	•
•	•	•	•
B(1)	0	25	50
B(2)	50	25	100
•	•	•	•
•	•	•	•
•	•	•	•

计算机能够每隔10ms进行一次调度决策。

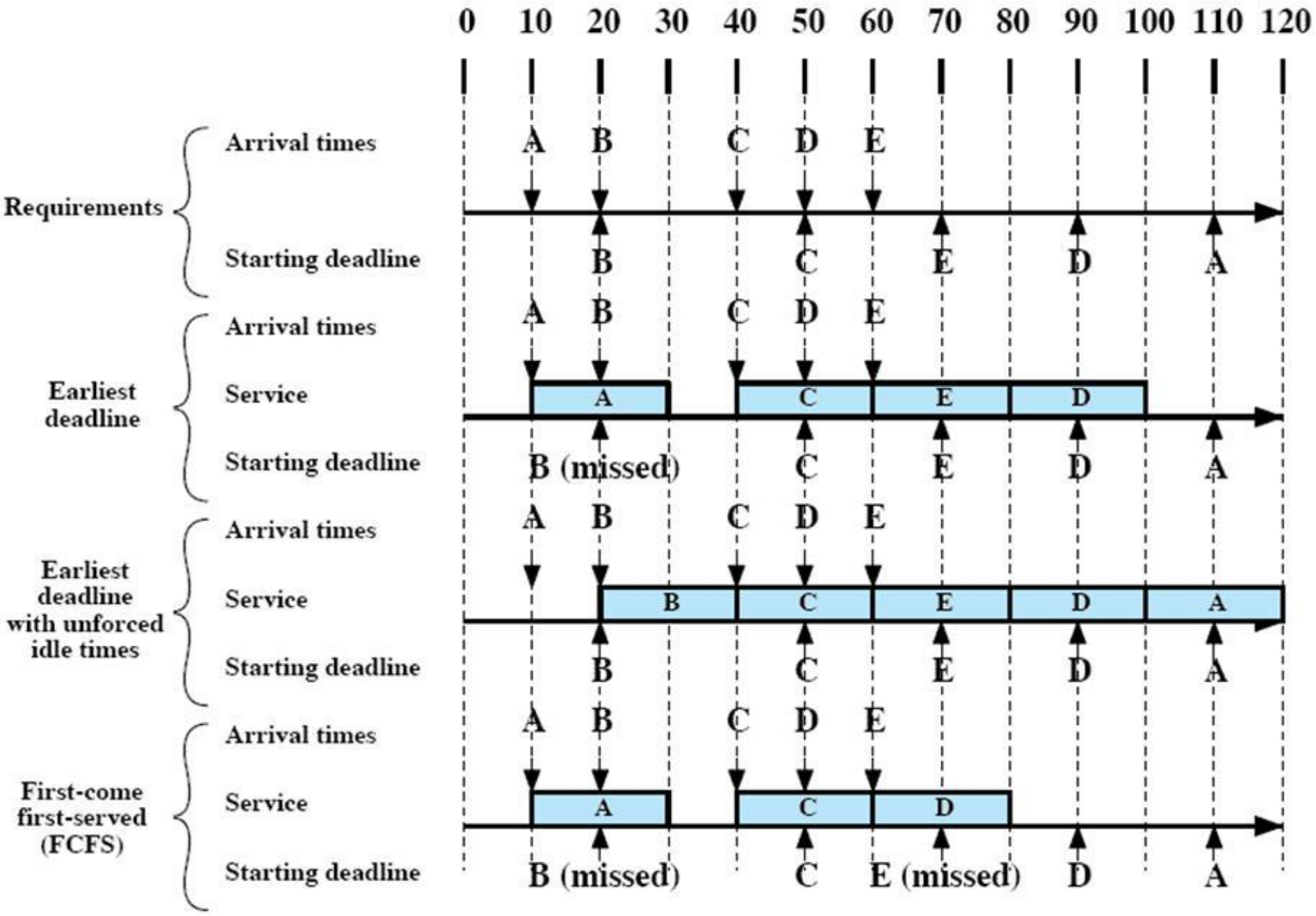

B1 deadline
B2 deadline
A1 deadline
A2 deadline
A3 deadline
A4 deadline
A5 deadline
Arrival times, execution times, and deadlines
A1
A2
A3
A4
A5
B1
B2
0
10
20
30
40
50
60
70
80
90
100
Time(ms
Fixed-priority scheduling; A has priority
A1
B1
A2
B1
A3
B2
A4
B2
A5
B2
A1
A2
B1 (missed)
A3
A4
A5, B2
Fixed-priority scheduling; B has priority
B1
A2
A3
B2
A5
A1 (missed)
A2
B1
A3
A4 (missed)
A5, B2
Earliest deadline scheduling using completion deadlines
A1
B1
A2
B1
A3
B2
A4
B2
A5
A1
A2
B1
A3
A4
A5, B2

例2：具有启动最后期限的非周期性任务调度

有自愿空闲时间的最早最后期限：**总是调度最后期限合格任务，并让该任务运行直到完成。**

5个非周期性任务的执行简表

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70


0 10 20 30 40 50 60 70 80 90 100 110 120
Requirements
Arrival times
A B C D E
Starting deadline
B C E D A
Earliest deadline
Arrival times
A B C D E
Service
A C E D
Starting deadline
B (missed) C E D A
Earliest deadline with unforced idle times
Arrival times
A B C D E
Service
B C E D A
Starting deadline
B C E D A
First-come first-served (FCFS)
Arrival times
A B C D E
Service
A C D
Starting deadline
B (missed) C E (missed) D A

10.2.5 速率单调调度（RMS）★★★★★

- 适应于周期性任务调度，最短周期的任务具有最高优先级，次短周期的任务具有次高的优先级，以此类推。
- 当同时有多个任务可以被执行时，最短周期的任务被优先执行。
- 衡量周期调度算法有效性的标准：是否能够保证满足所有硬最后期限，对于RMS，须满足：

$$C_1/T_1+C_2/T_2+\ldots+C_n/T_n \leqq n(2^{1/n}-1)$$

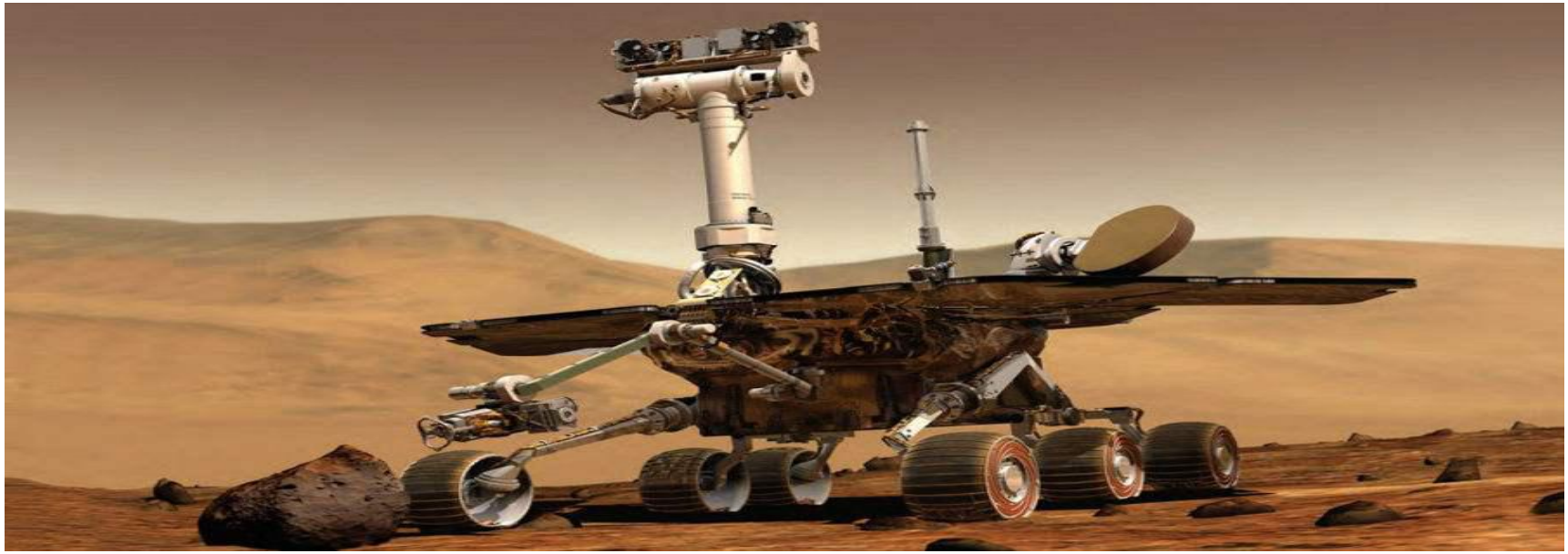
- 任务速度（单位为赫兹）为其周期（单位为秒）的倒数。C_i是任务i的执行时间，T_i是任务i的周期。

例

- 任务P_1：C_1=20;T_1=100;U_1=0.2
- 任务P_2：C_2=40;T_2=150;U_2=0.267
- 任务P_3：C_3=100;T_3=350;U_3=0.286
- 三个任务的总利用率为0.2+0.267+0.286=0.753，使用RMS，三个任务的可调度性上界根据公式计算为0.779，0.753<0.779，故所有的任务可以成功得到调度。

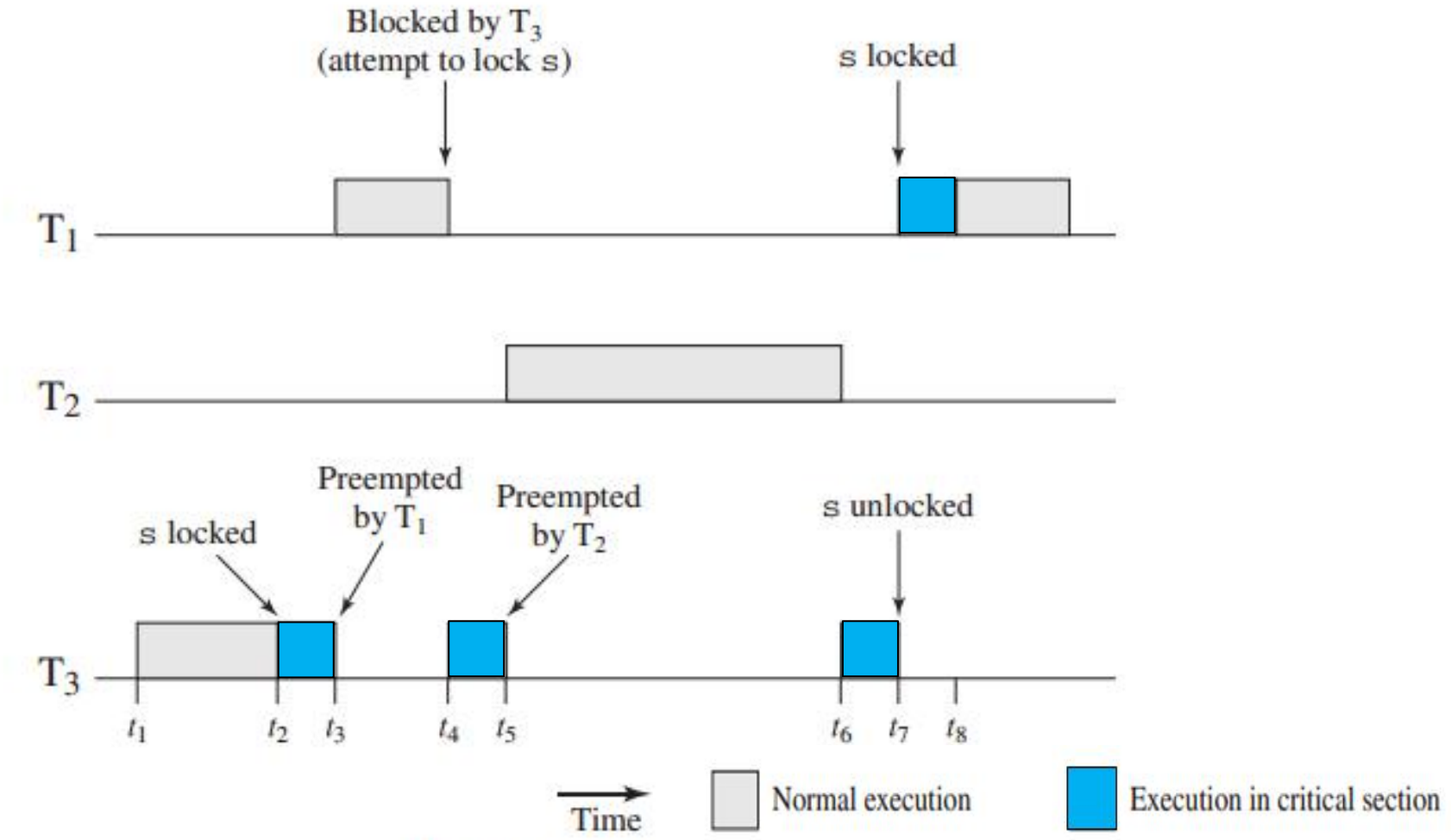
10.2.6 优先级反转 ★★★★★

- 优先级反转是在任何基于优先级的可抢占调度方案中都会出现的一种现象。
- 在任何优先级调度方案中，系统应该不停地执行具有最高优先级的任务。当系统内的环境迫使一个较高优先级的任务去等待一个较低优先级的任务时，优先级反转就会发生。



10.2.6 优先级反转 ★★★★★

- 优先级反转
- 无界限优先级反转



(a) Unbounded priority inversion

10.2.6 优先级反转 ★★★★★

- **避免优先级反转的解决方案：**
  - **优先级继承**
  - **优先级置顶**

10.2.6 优先级反转 ★★★★★

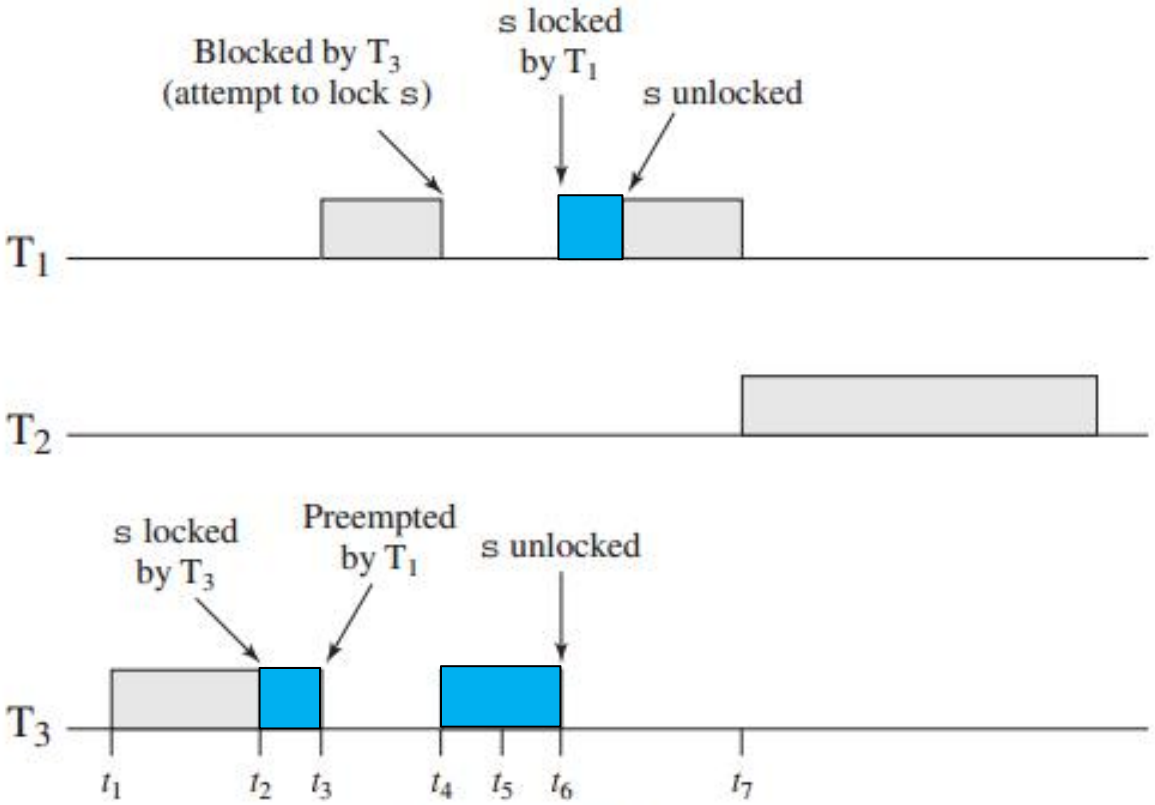


Figure 10.9 Priority Inversion

作业

- 习题
  - 10.1（只做“最早完成最后期限调度”）
  - 10.2（只做“有自愿空闲时间的最早最后期限调度”）