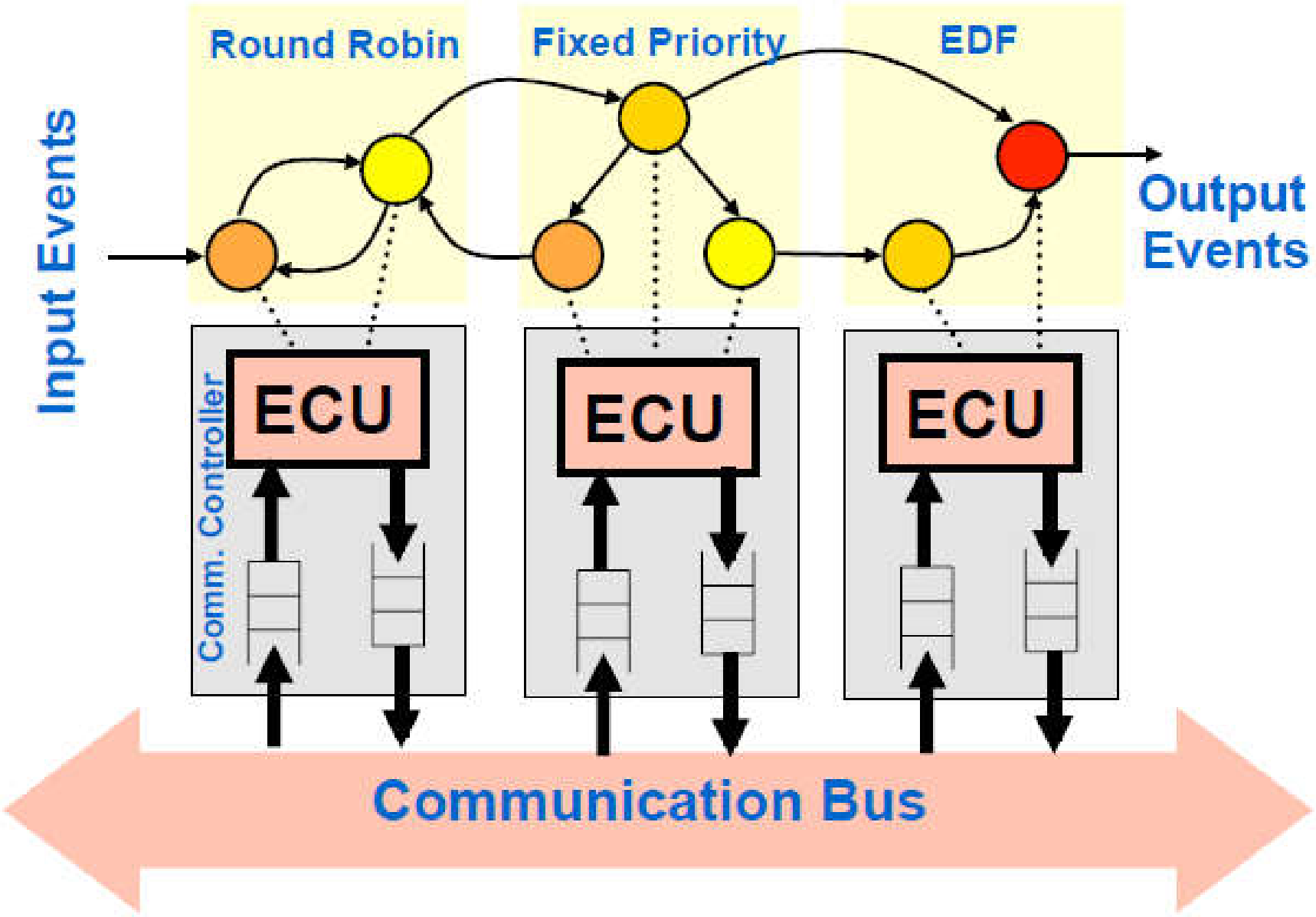
实时调度理论

李曦

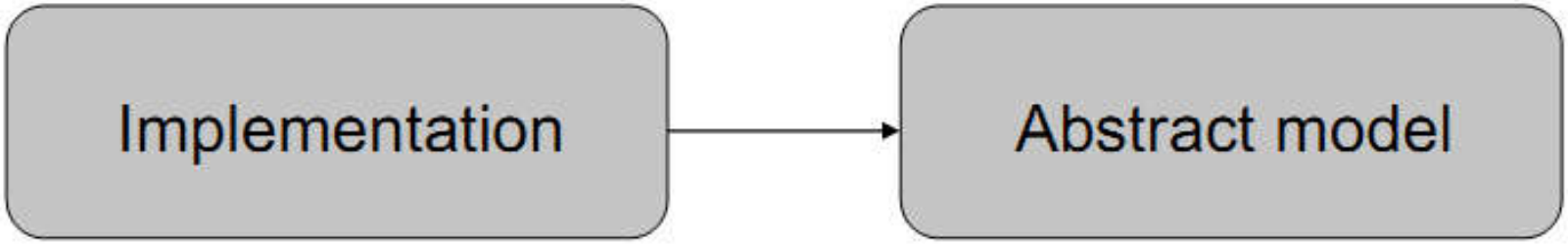
llxx@ustc.edu.cn

计算机系计算机应用研究室

Architecture model of CPS

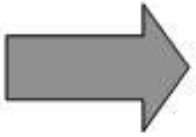


Task model： one function = one task




```
task body P1 is
   Interval : constant Duration := 5.0;
   Next_Time : Time;
begin
   Next_Time := Clock + Interval;
   loop
      Action;
      delay until Next_Time;
      Next_Time := Next_Time + Interval;
   end loop;
end P1;

task body P2 is
   Interval : constant Duration := 7.0;
   Next_Time : Time;
begin
   Next_Time := Clock + Interval;
   loop
      Action;
      delay until Next_Time;
      Next_Time := Next_Time + Interval;
   end loop;
end P2;
```



τ_1 $\tau_1 = \{ C_1, T_1, D_1, O_1 \}$

τ_2 $\tau_2 = \{ C_2, T_2, D_2, O_2 \}$

任务特征：按调度复杂性

- 1级——到达时间的可预测性
  - 纯周期，无异步事件，且计算要求不变
  - 小导弹，无人驾驶靶机
- 2级
  - 大部分是周期的，少量异步事件和“突发”计算负载（错误恢复，外部命令）
  - 航电，空间系统
- 3级
  - 异步的（或事件驱动），周期处理少
  - 通讯，雷达
- 两种调度方法：周期执行法（1级），多任务处理法（3级）

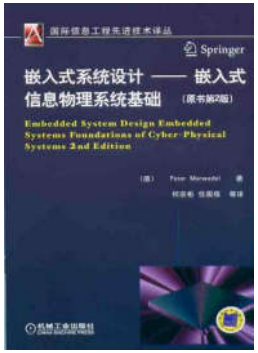
	任务	任务执行	复杂共享数据	可预测性
周期法	同步、顺序	非抢占	难（执行顺序决定）	好
多任务	异步、并发	可抢占	易（同步原语）	难

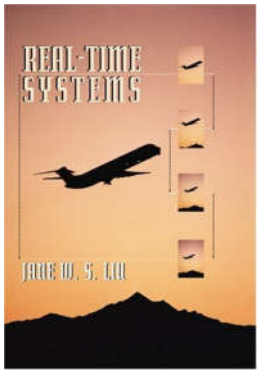
内容提要

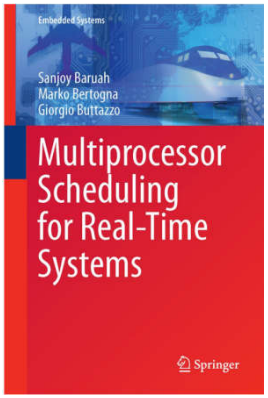
- 实时任务类型与约束模型
  - Assumptions about task timing，interaction，。。。
- 任务调度算法Scheduling Algorithm
  - Scheduling mode and selection function
    - Timeliness：deadline，worst response time，。。。
    - Efficiency：average response time，makespan
  - Prioritized goals
    - Temporal predictability first，performance second
- 可调度分析Schedulability Test
  - Prediction of worst-case behavior
  - 基于CPU利用率(workload analysis)
    - for preemptive and strictly periodic tasks?
  - WCRT(Response time analysis)
    - for preemptively feasible task sets with $D \leq T$

参考文献

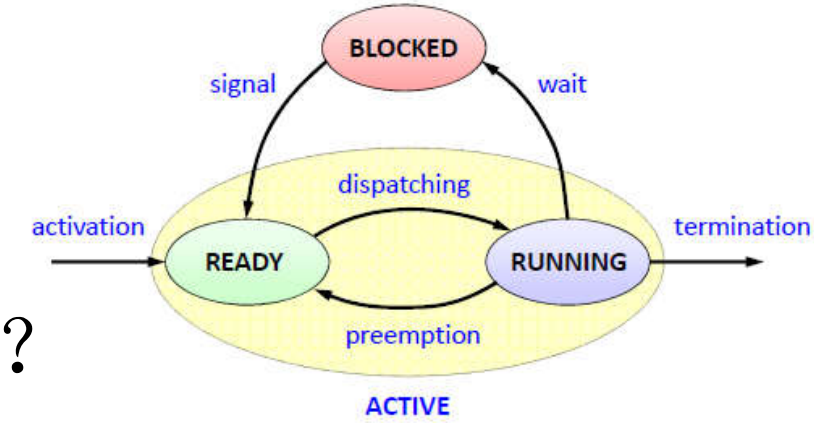
- Peter Marwedel（TU Dortmund教授）
  - 《嵌入式系统设计·嵌入式CPS系统基础》，第2版，2011。译错多☹
- Giorgio C Buttazzo，RETIS Lab, TeCIP Insitute
  - Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications，第三版，2011
  - Multiprocessor Scheduling for Real-Time Systems，2015
- Jane W.S.Liu(張韻詩，美)
  - 《实时系统》



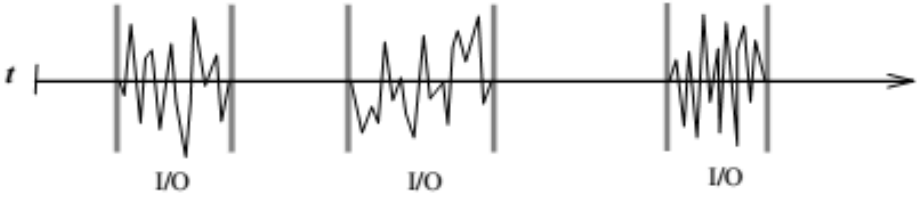


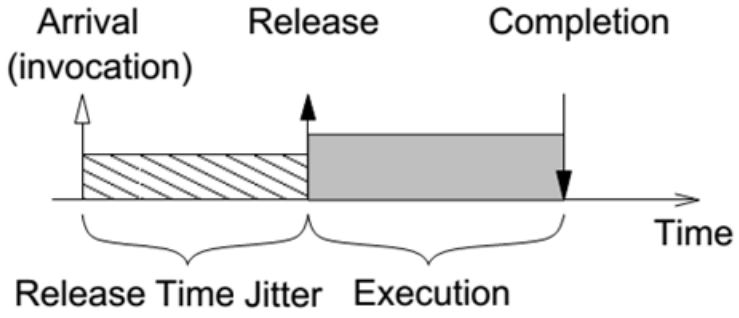


状态（队列）、定时、ET/TT？



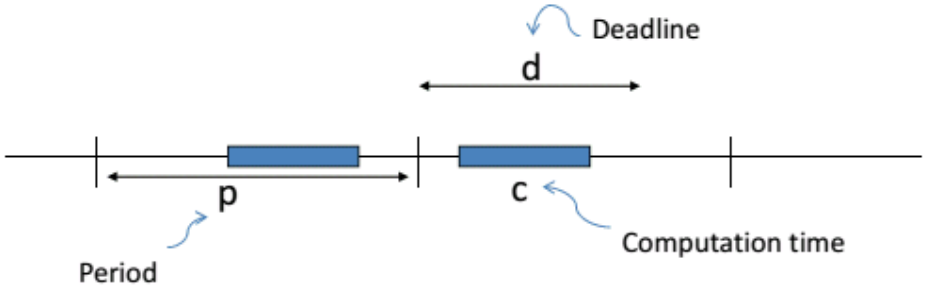
- 等待队列
- 就绪队列（运行队列）
- active = arrive = release?
  - active ：任务被激活（创建）
    - “make ready a new task，either immediately or at some future point in time”（Kopetz P220）
  - Arrive：TT的任务到达，或ET的事件到达
    - = release=start

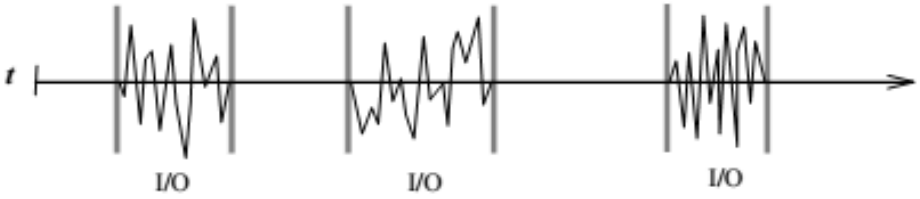




任务：Tasks，jobs，三类

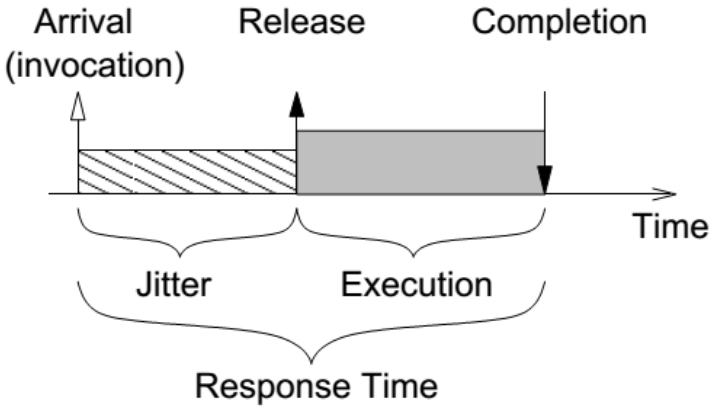
- A *task T= {J1,J2,…,Jn}* is a set of related jobs that together perform some operation
- A *Job* is a unit of work scheduled and executed by the system（the unit of concurrency）
  - If *jobs* occur on a regular cycle, the task is periodic
  - A sporadic task is an aperiodic task where the jobs have deadlines once released
    - 存在“最小到达间隔”！
  - If *jobs* have unpredictable release times, a task is termed aperiodic
    - Appear once only? Have average deadlines（soft-RT）？
  - 非周期两种类型：到达模式，软硬实时性（DL）



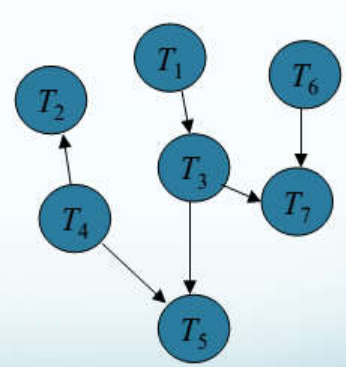


Constraints on Tasks（RTOS view）

- Timing(定时) constraints
  - Release time
    - Relative to arriving time
  - Deadline
    - Absolute deadline



- Precedence(优先) constraints，偏序
  - Temporal order：Temporal dependence
    - temporal distance：“完成时间”之差
  - Causal order：Causal dependence
    - Happen-before(Lamport)：基于msg
- Resource constraints
  - *Data dependence?*
  - Mutual exclusion constraints
    - Resource access protocols



任务的定时模型timing model

a：到达时刻arrival/request

T：切换时间

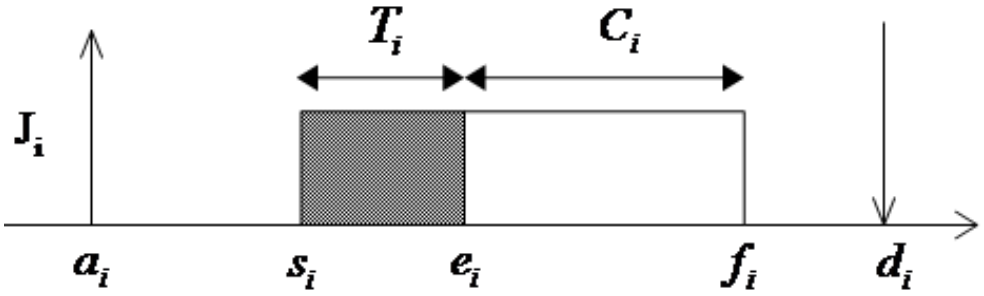
s：调度开始时刻start

e：调度结束时刻/release

C：capacity/执行时间/Workload

f：完成时刻

R：响应时间



当*T=0* 且 *Jitter=0* 时：Arrival = Start = Release

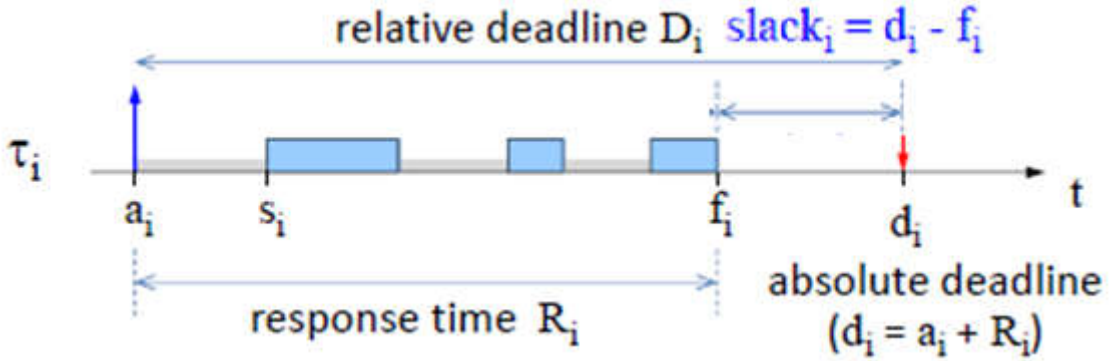
截止期deadline

绝对*d*（*时刻*）

相对*D*（*时长*）

松弛时间slack，裕度laxity

延迟Lateness：*d* – *f*



开始时限

Delay?

抖动jitter

周期任务的定时模型

Static task parameters:

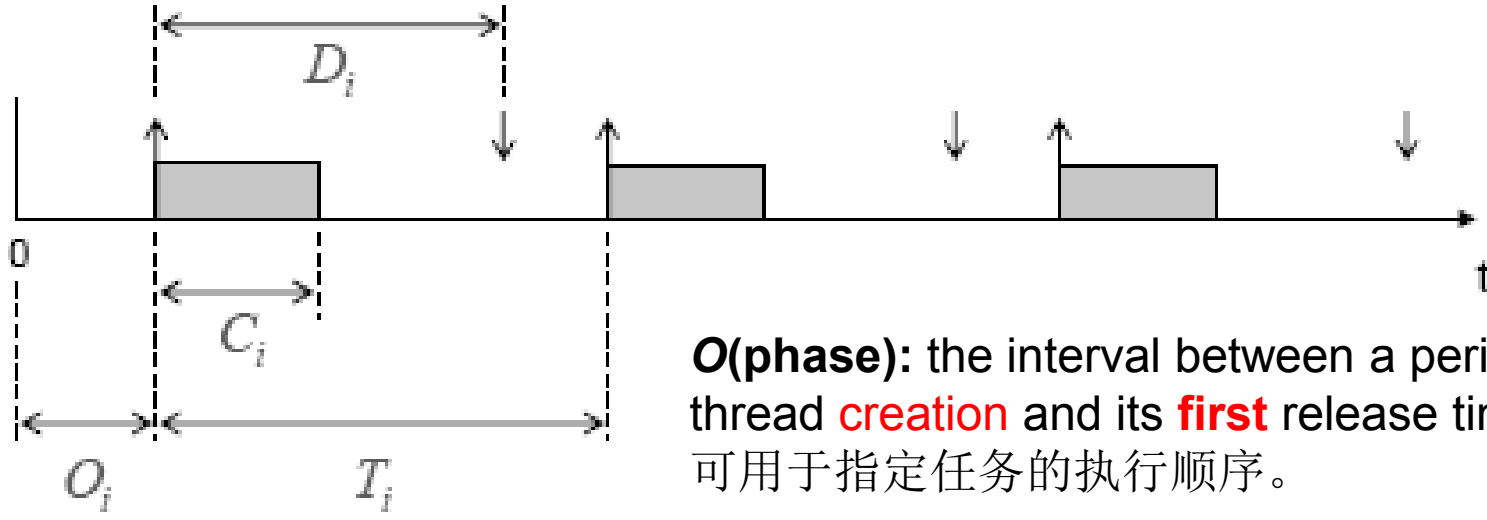
τ_i $\quad \tau_i = \{C_i, T_i, D_i, O_i\}$

C_i : (undisturbed) WCET

T_i : period

D_i : (relative) deadline

O_i : (absolute) time offset

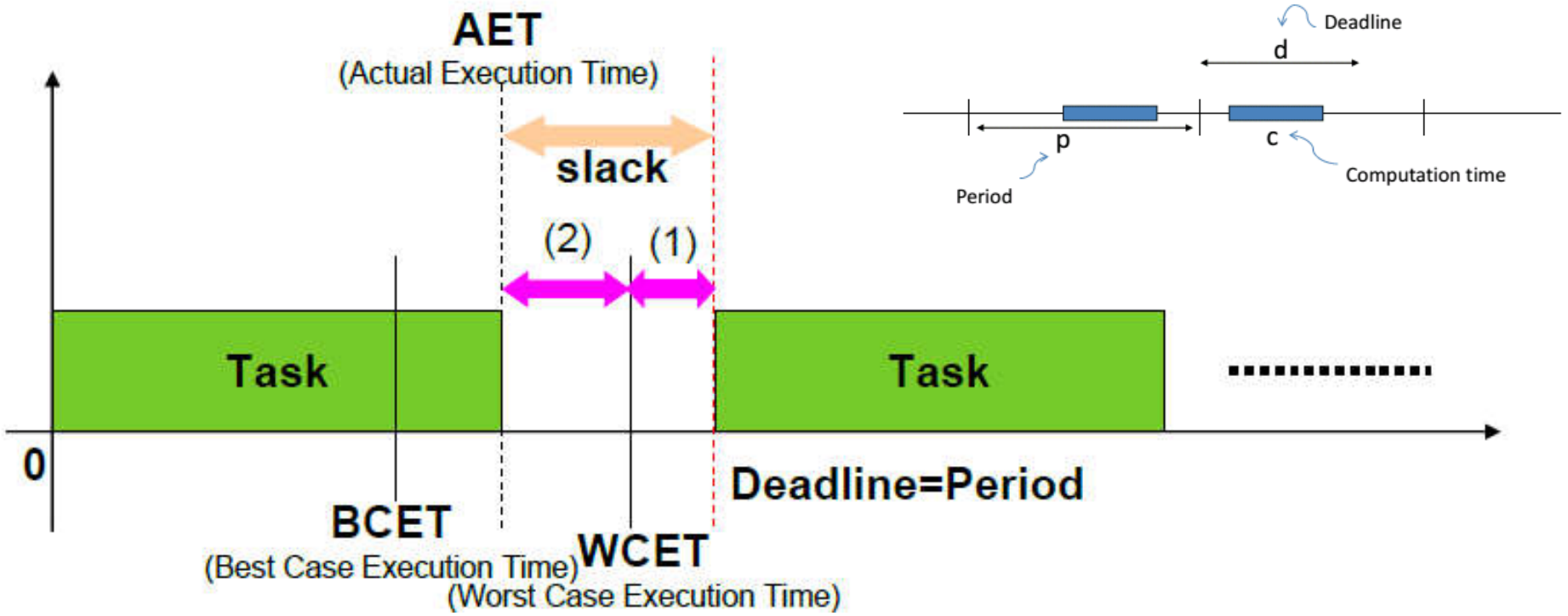


***O*(phase):** the interval between a periodic thread creation and its **first** release time。
可用于指定任务的执行顺序。

注意：未考虑调度切换开销！

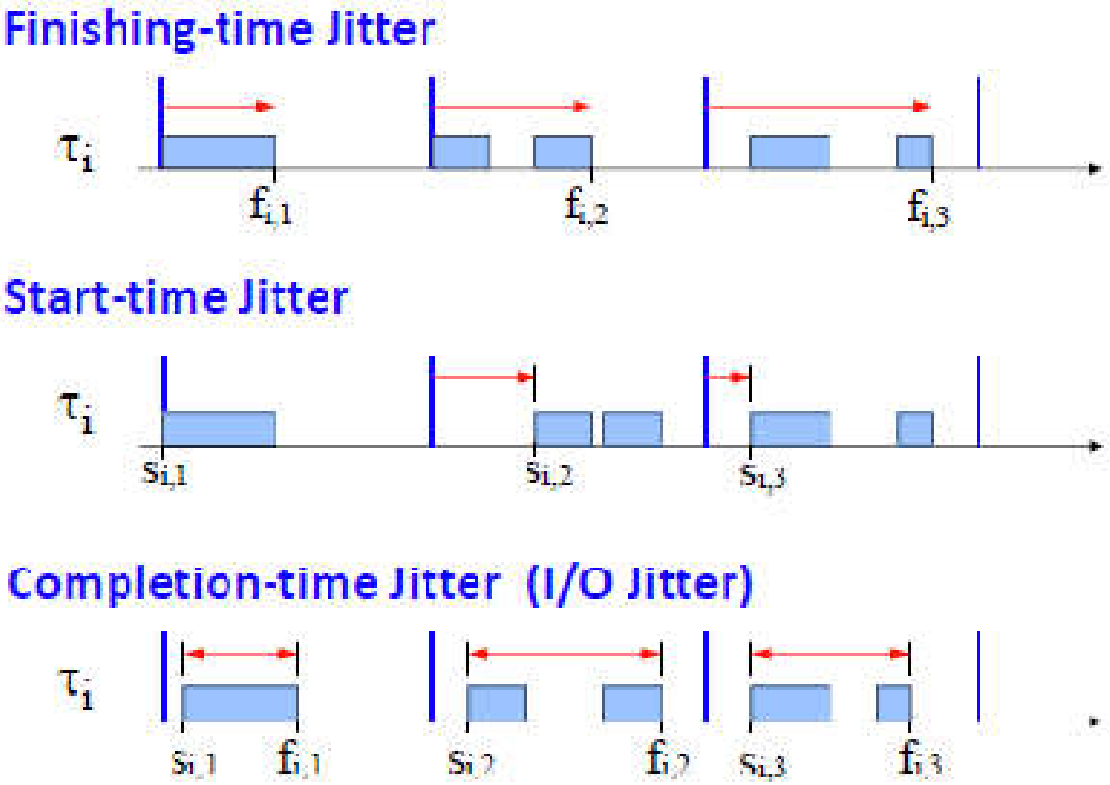
截止时间***D***可以等于或不等于周期***T***!

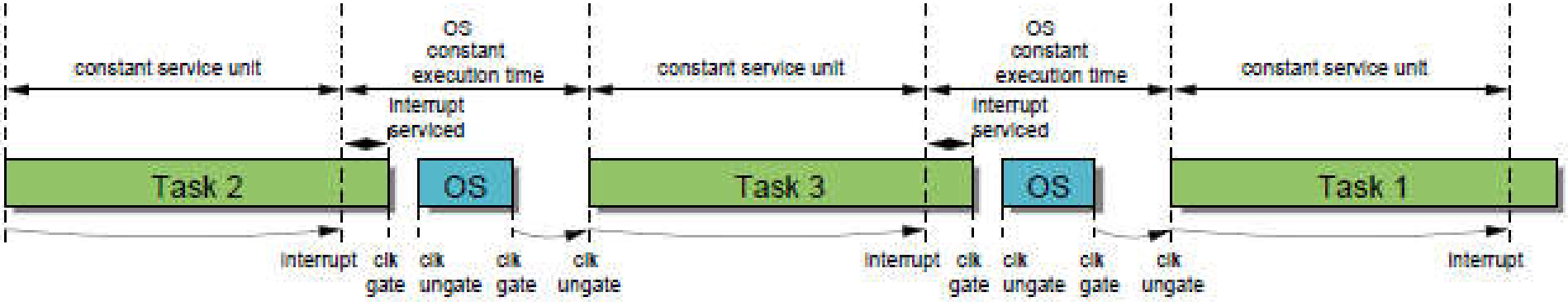
任务执行时间



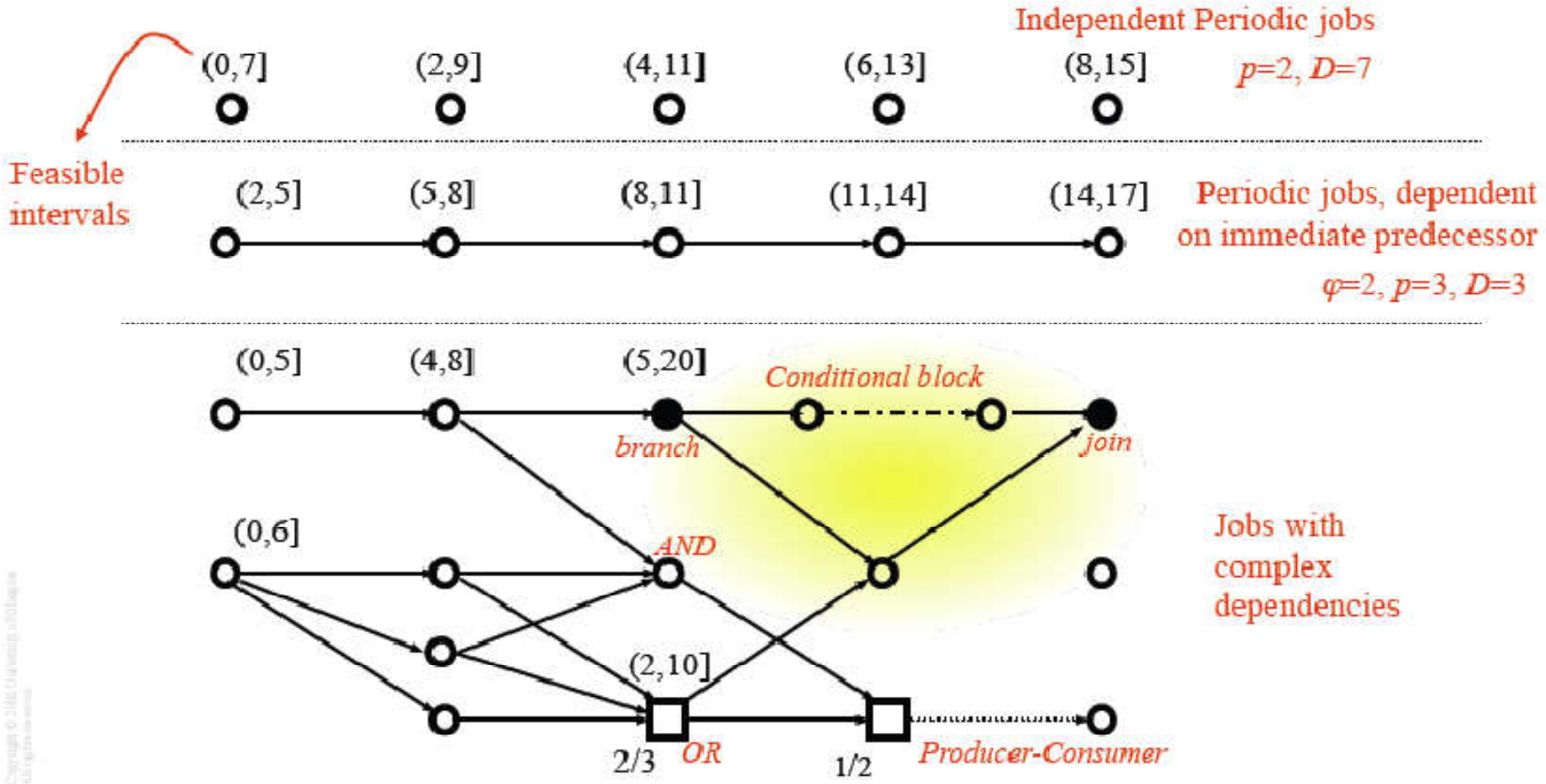
- BCET
- WCET

Types of Jitter：原因？





precedence graph，task graphs

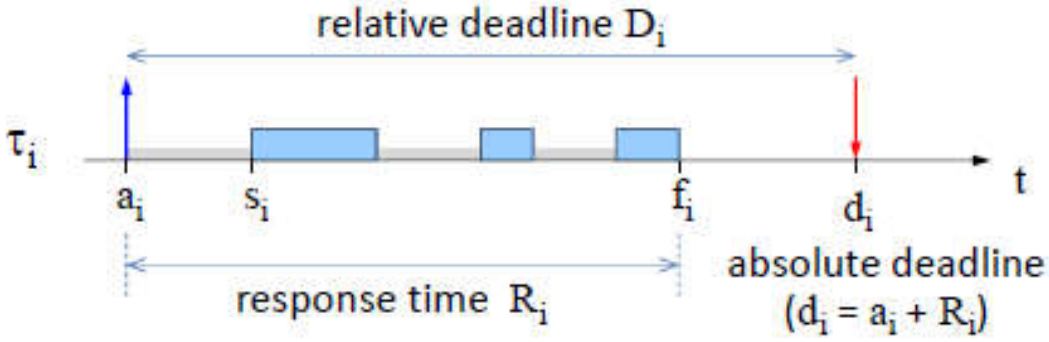


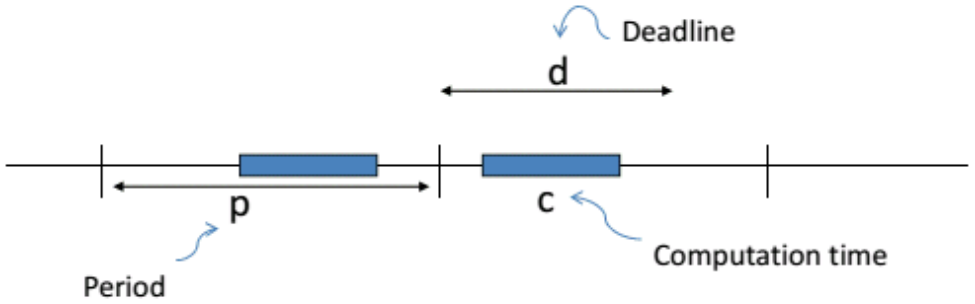
- 三类：chains、intree(AND/OR)、outtree(branch)
- 顺序依赖边（偏序）、数据依赖边、时间依赖边

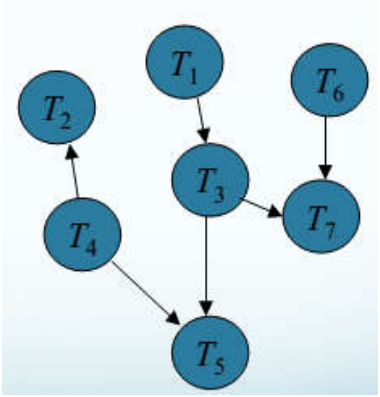
maintained by RTOS

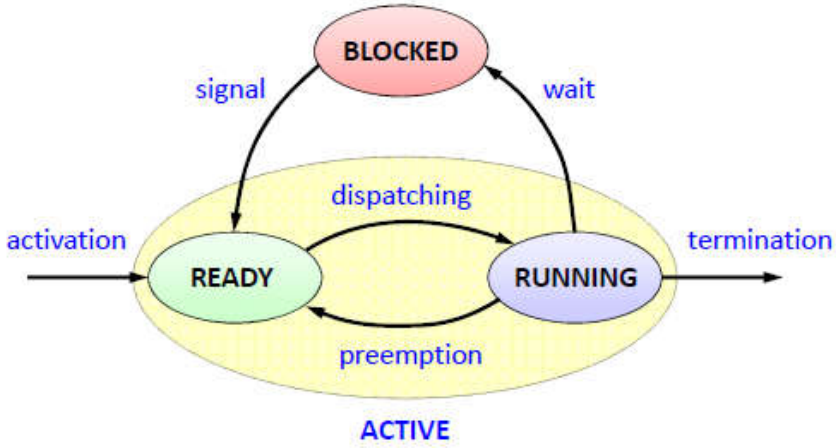
Task Control Block

task identifier
task address
task type
criticalness
priority
state
computation time
period
relative deadline
absolute deadline
utilization factor
context pointer
precedence pointer
resource pointer
pointer to the next TCB



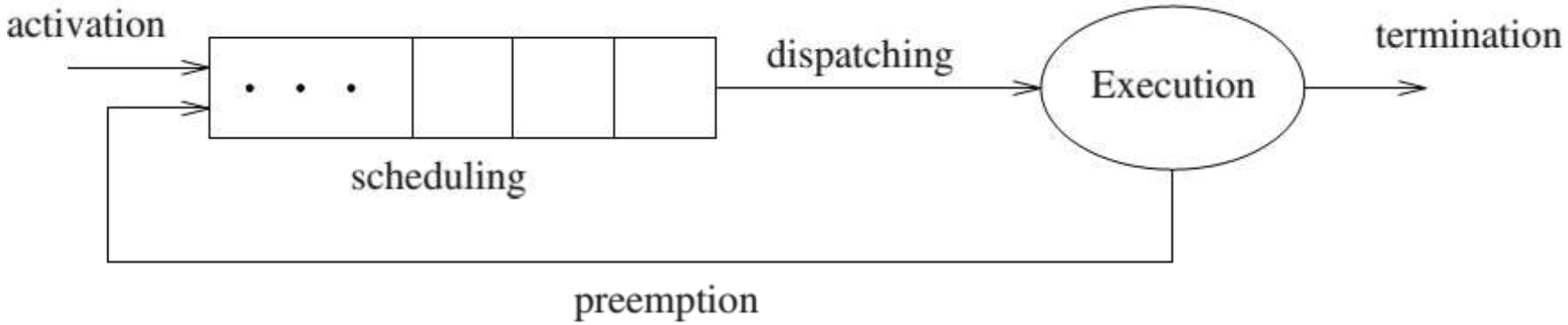


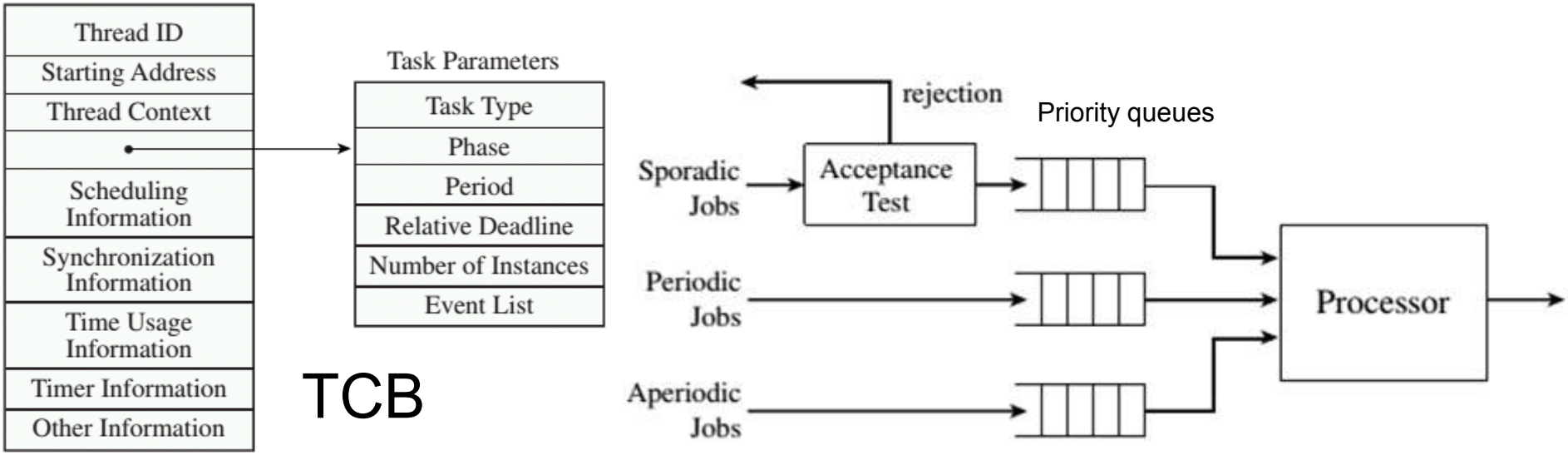




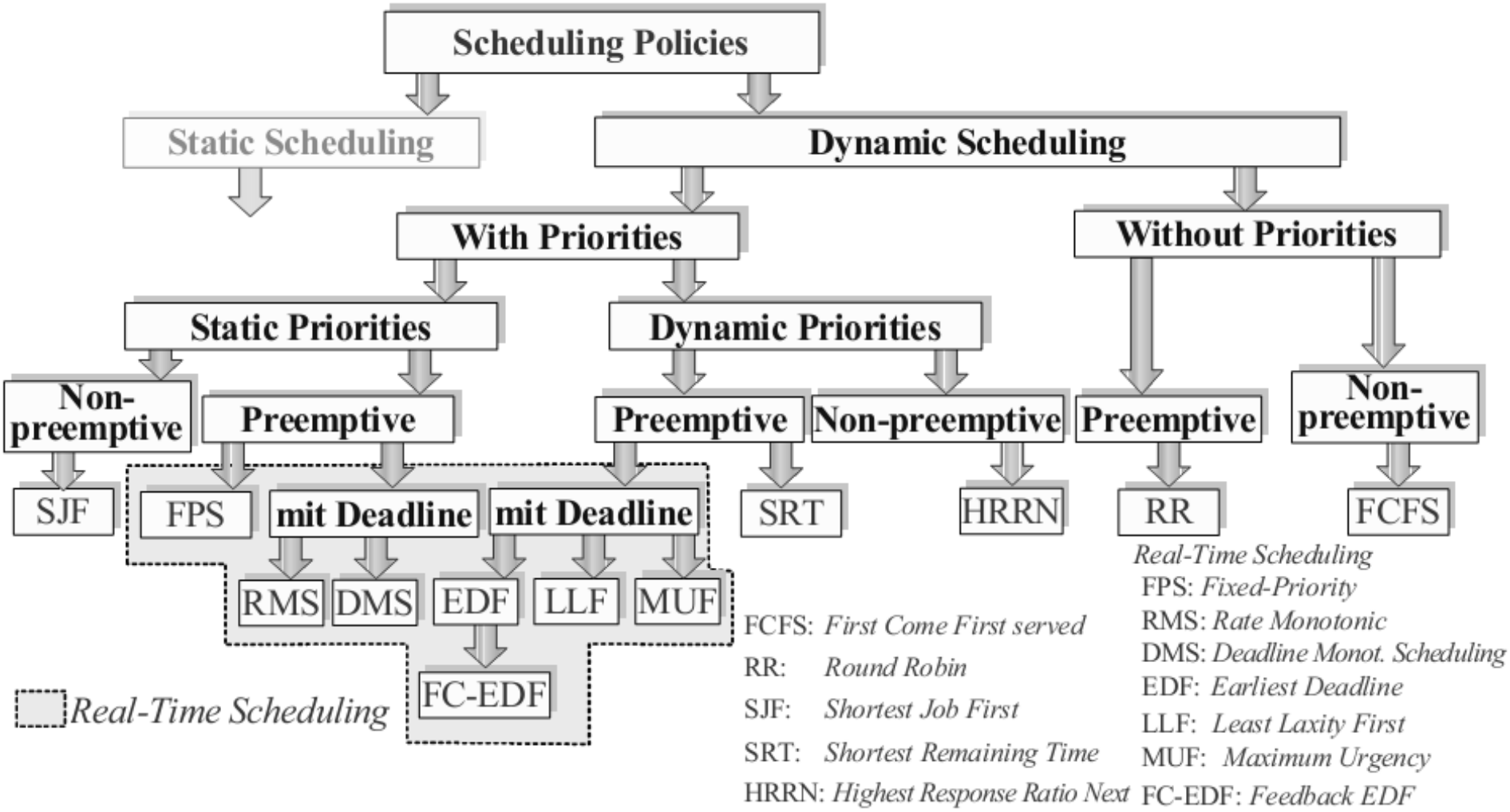
Task scheduling

- 逐一调度并发任务执行，满足约束条件
  - 功能：排序、定时、分配（单处理器、多处理器、多核）



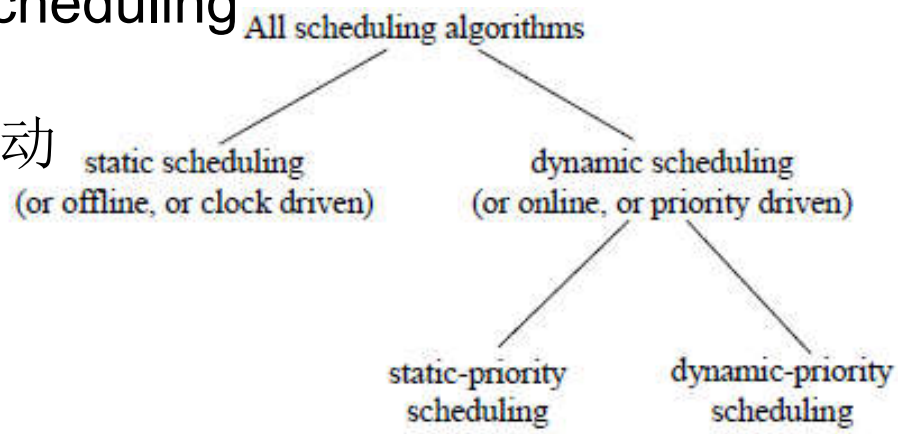


Uniprocessor scheduling algorithms



实时调度算法分类

- 按调度方案生成方式
  - 静态调度：off-line，Static Table-Driven Scheduling，可预测性好
    - Time-driven scheduling，任务时间表，固定每个任务的起始时间
  - 动态调度：on-line
    - 如：Priority-driven scheduling，生成任务队列，顺序执行
- 按调度策略：如何进行任务排序？
  - 基于时间的调度算法（Time-driven scheduling）：离线静态
  - 基于优先级的调度算法（Priority-driven scheduling）：动态在线
- 按调度机制：触发任务调度的方式
  - 时间触发：Clock driven/tick scheduling
    - 固定时间片/可变时间片
  - 事件触发：优先级驱动/中断驱动
  - Hybrid：
  - 可抢占/不可抢占
- uc/OS？



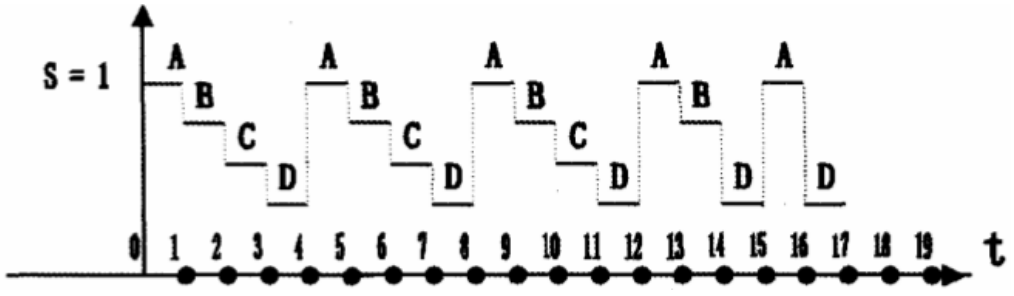
Scheduling Objective

- User-oriented Criteria
  - Response time
  - Turnaround time
  - Deadline
- System-oriented Criteria
  - Throughput
  - CPU utilization
  - Fairness

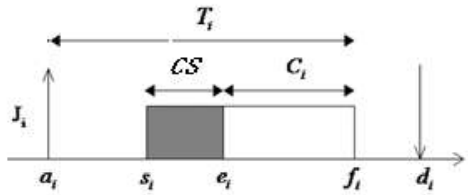
Algorithm	Decision Mode	Priority Function	Arbitration Rule
FCFS	non-preemptive	$P(r) = r$	Random
SJF	non-preemptive	$P(t) = -t$	FIFO/Random
SRT	preemptive (at job arrival)	$P(a,t) = a - t$	FIFO/Random
RR	preemptive (at quantum)	$P() = k$	Cyclic
MLF	preemptive (at quantum)	$P(a)$, n-levels	FIFO/Cyclic

a：attained time；w：waiting time；r：real time, r = a + w；t：total time required

Round-Robin



- 抢占式时间片轮转调度
  - 任务的当前时间片用完后，OS将CPU使用权给下一个Ready任务或被中断的任务（Interrupted）
  - 基于CPU使用率的共享式调度（Share-driven scheduling）
    - “共享”= 按比例分享！
  - 适用：响应时间要求不高者
- 时间片确定
  - 周转时间T_i：任务从提交OS直至完成的时间（*f* - *a*）
    - 称*turn around time*（就是实时任务模型中的“响应时间”）
  - 带权周转时间W_i：一个任务的周转时间T_i与OS实际给它的服务时间T_{Si}之比。
  - 调度目标：*n*个任务的平均周转时间*T*或平均带权周转时间*W*短
    - 即：硬件资源利用率高（吞吐率最大）
  - 时间片大或小好？



$$T = \frac{1}{n}\left[\sum_{i=1}^{n} T_i\right] \qquad W = \frac{1}{n}\left[\sum_{i=1}^{n} \frac{T_i}{T_{Si}}\right]$$

例：

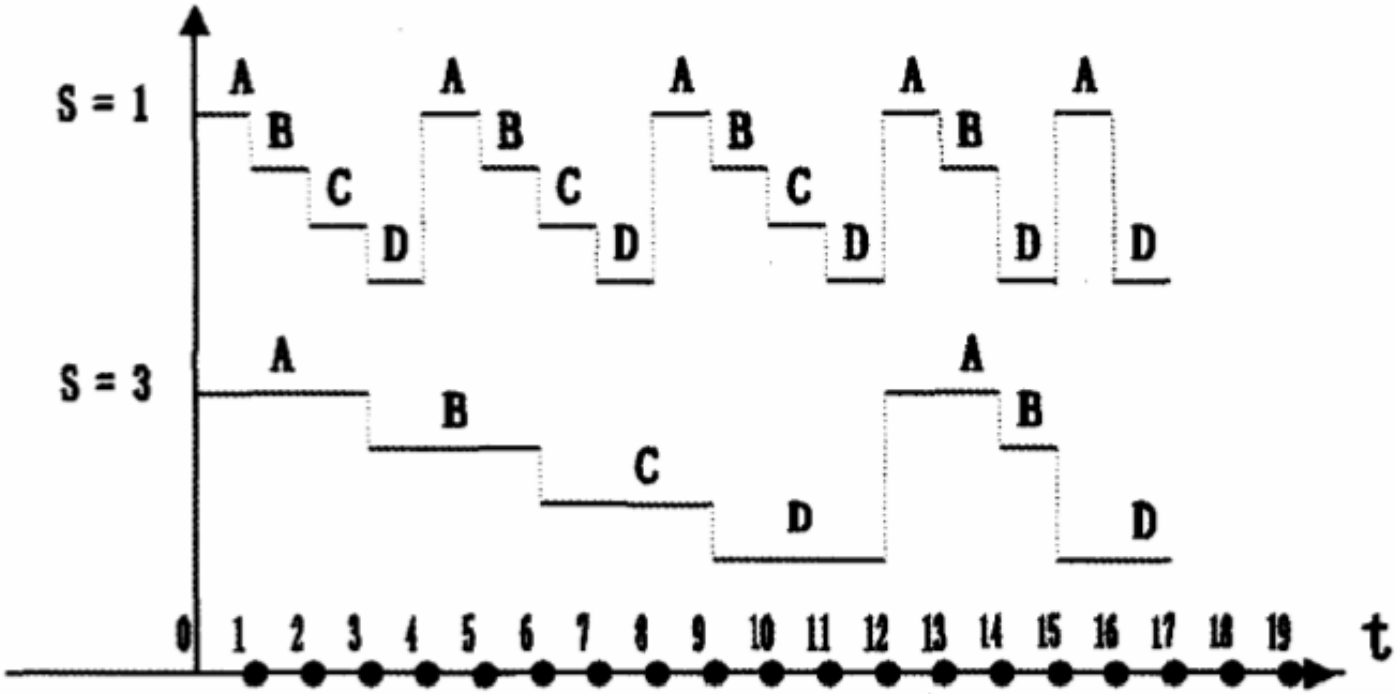
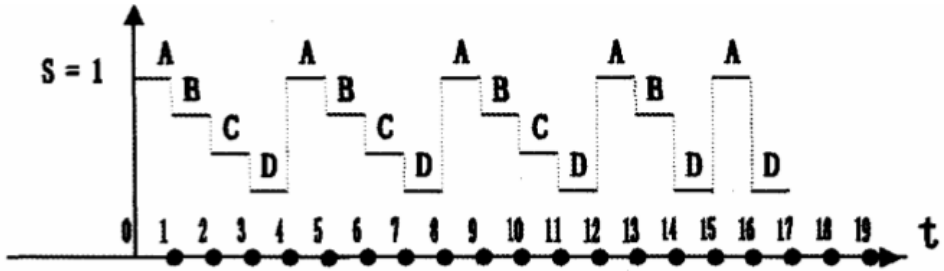


图1 S=1 和 S=3 时任务运行图

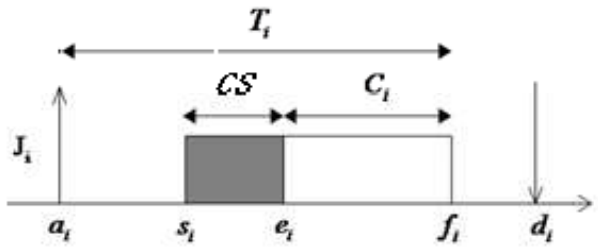
表 1 反映时间片的大小对平均周转周期和平均带权周期的影响。

任务情况 / 时间片	任务名	A	B	C	D	平均
	到达时间	0	1	2	3	
	服务时间	5	4	3	5	
S = 1	完成时间	16	14	11	17	
	周转时间	16	13	9	14	**13**
	带权周转时间	3.2	3.25	3	2.8	**3.0625**
S = 3	完成时间	14	15	9	17	
	周转时间	14	14	7	14	**12.25**
	带权周转时间	**2.8**	**3.5**	**2.3**	**2.8**	**2.85**

RR系统的最坏周转时间



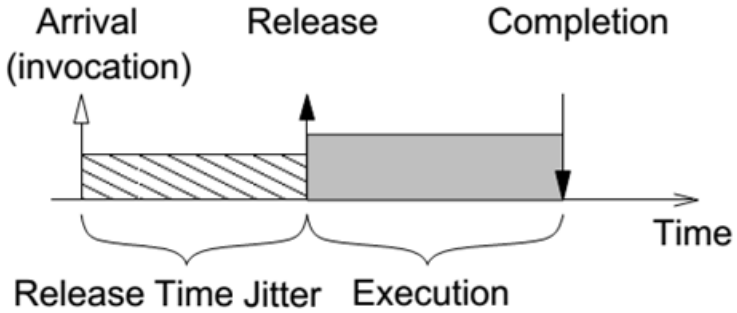
- 完成所有任务的最坏时间
- 假设
  - 同时释放*n*个任务
  - 系统启动后，不会有新的任务，任务也不会提前结束
  - 任务执行的顺序不预定，但固定
  - 最大的任务的执行时间为*c*
  - 时间片为*q*



- 则
  - 一个任务下一次执行的等待时间 $(n-1)q$
  - *每个*任务最多要执行 $\left\lceil \frac{c}{q} \right\rceil$ 次
  - 任务的最坏完成时间：$$T = (n-1)\left\lceil \frac{c}{q} \right\rceil q + c$$
  - 考虑调度切换开销*o*时：$$T = \left[(n-1)q + n \cdot o\right]\left\lceil \frac{c}{q} \right\rceil + c$$

实时任务调度

- 算法选择需考虑任务（集）属性
  - 任务数：固定/可变
  - 可抢占性
  - 时限：开始，结束（截止期）
  - 周期性
  - 优先约束：独立/优先顺序
    - 同步原语 or 调度控制？
  - 到达时刻：同时到达/异步到达
  - *关键性*
- 执行调度的时机：ET/TT
- 算法性能：
  - 响应性、可预测性、调度器开销
  - 可行调度（feasible schedule）
  - 最优调度（optimal schedule）
    - 其他算法可行，最优算法一定可行



静态调度： table-driven & cyclic scheduler

- table-driven（上）：预定任务的所有执行时刻
- Cyclic：重复预定的调度，只存储major cycle
  - major cycle：所有任务周期的最小公倍数LCM
    - 最好包含各frame的slack times
  - major cycle被划分成frame (=minor cycle?)，需考虑：
    - 上下文切换最小：任务尽量在一个frame中完成
    - 最小化调度表：主周期应包含整数个frame
    - 满足deadline： *D*至少应为一个完整的frame
  - Periodic timer：用于确定frame的开始
    - 调度点：frame的边界（选择任务，检查超时）
    - Periodic task：frame开始处开始执行
    - sporadic/aperiodic：利用frame的slack time

Task	Start Time in milli Seconds
T_1	0
T_2	3
T_3	10
T_4	12
T_5	17

Task Number	Frame Number
T3	F1
T1	F2
T3	F3
T4	F2

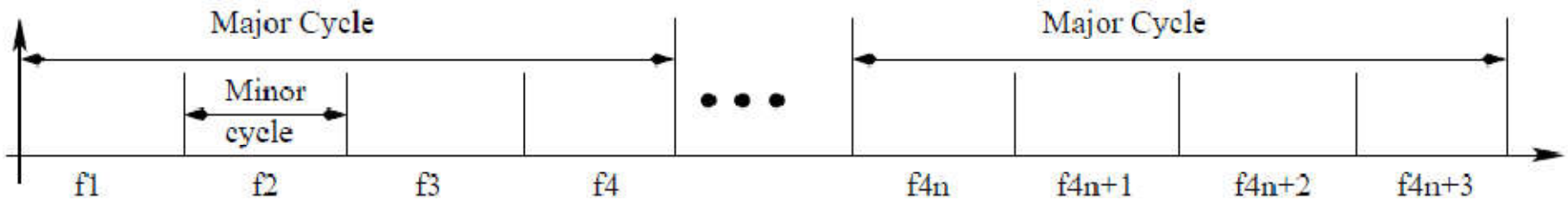
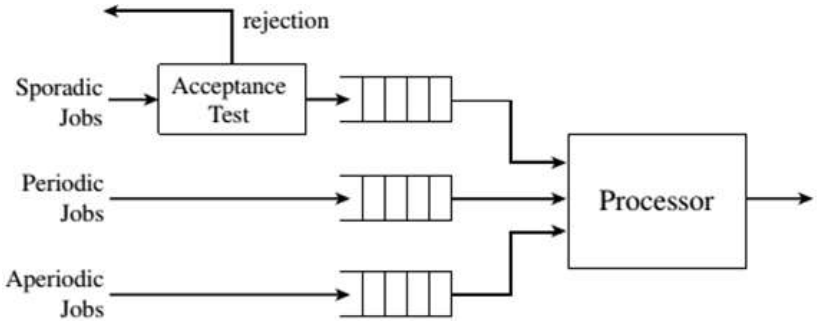


Figure 6: Major and Minor Cycles in a Cyclic Scheduler

A Cyclic Scheduler

Task Number	Frame Number
T3	F1
T1	F2
T3	F3
T4	F2

```
cyclic-scheduler() {
current-task T = Schedule-Table[k];
k = k + 1;
k = k mod N;                    // N is the total number of tasks
                                //in the schedule table
dispatch-current-Task(T);
schedule-sporadic-tasks();      //Current task T completed early,
                                //sporadic tasks can be taken up.
schedule-aperiodic-tasks();     //At the end of the frame, the running
                                //task is preempted, if not complete.
idle(),                         // No task to run, idle.
}
```



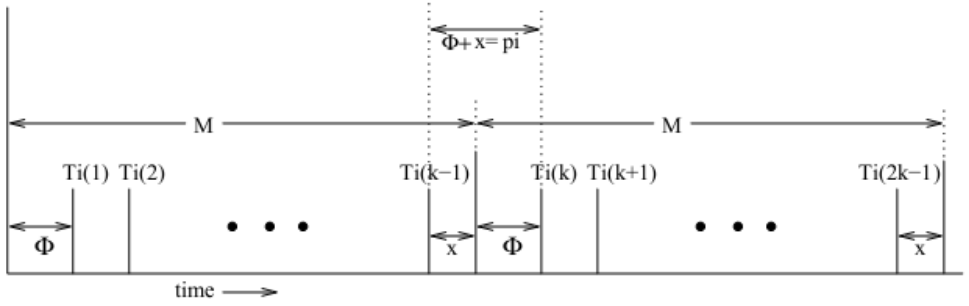


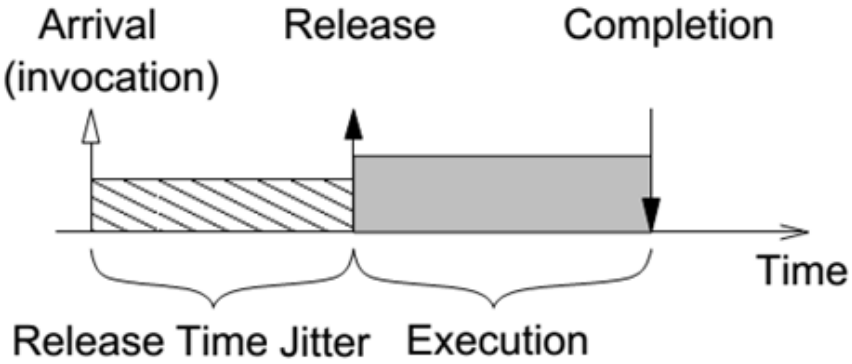
Figure 4: Major Cycle When A Task T_i Has Non-Zero Phasing

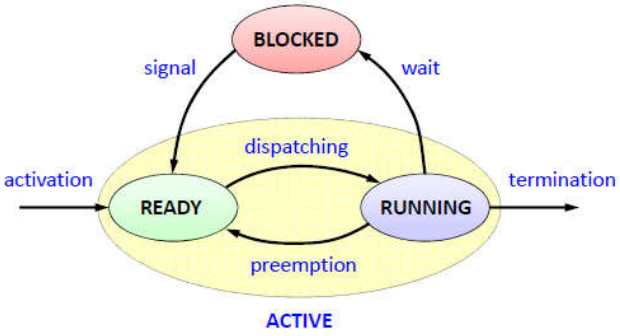
时间限定(deadline)调度算法

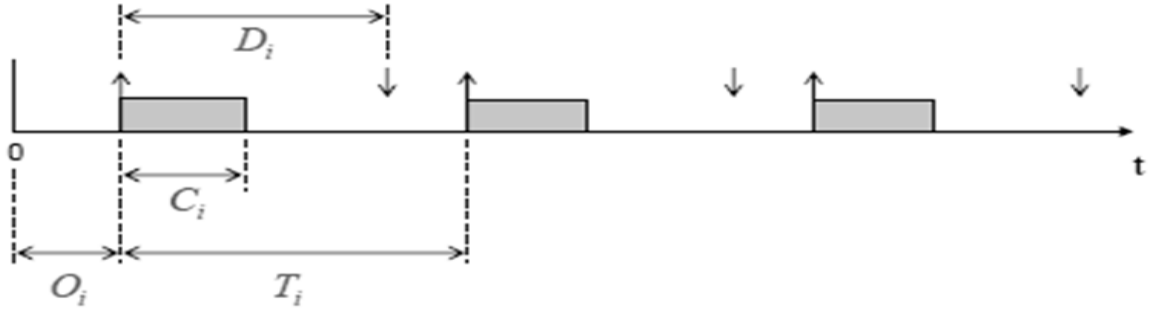
- 静态优先级算法：每个job相同不变
  - 速率单调算法（RM，Rate Monotonic Scheduiing）
    - 周期任务(Multi-rate)，优先级由任务周期决定，小优先
    - 截止期等于任务周期的最优静态优先级单处理器算法
  - 截止期单调算法（DM，Deadline Monotonic）
    - 周期任务，优先级由任务截止期决定，小优先
    - 截止期不等于任务周期时的最优静态优先级单处理器算法
- 动态优先级算法：每个job可不同
  - 最早时限优先（EDF，earliest deadline first）
    - 优先级由任务时限决定，早优先
    - 最优的动态优先级单处理器调度算法
  - 最小裕度算法（LLF，least laxity first）
    - 裕度小优先
    - 最优调度
- 属单处理器online算法
  - 周期任务：RM、DM、EDF
  - 非周期任务：EDF、LLF（不能周期？）

对周期任务，RM/EDF的假设

- Homogeneous task sets
  - 所有任务都是周期性的
    - 相位固定？
  - 时限（截止期）等于任务周期
    - 相对 or 绝对？
- Simultaneous activations
  - 所有任务预先创建（激活）
    - RM需要为每个任务指定固定的优先级
- Fully preemptive tasks
  - 任何时刻都可抢占，且抢占开销可忽略
- 任务独立
  - No precedence constraints，No resource constraints



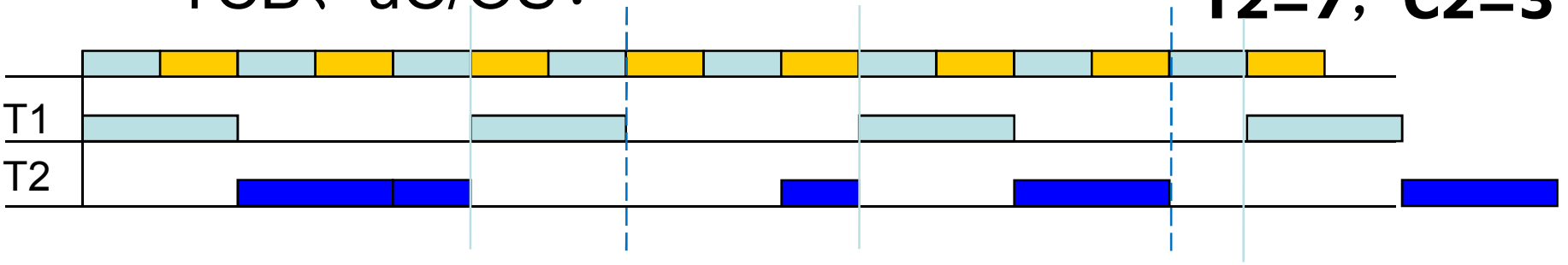




RM算法：固定优先级调度算法

- 任务的**周期=截止期**
  - 任务优先级静态分配：周期小任务优先
    - 周期小 = 频率高
  - T2错失时限的情况？
- 实现：静态优先级分配，固定优先级
  - TT、ET？一定要tick？
  - TCB、uC/OS？

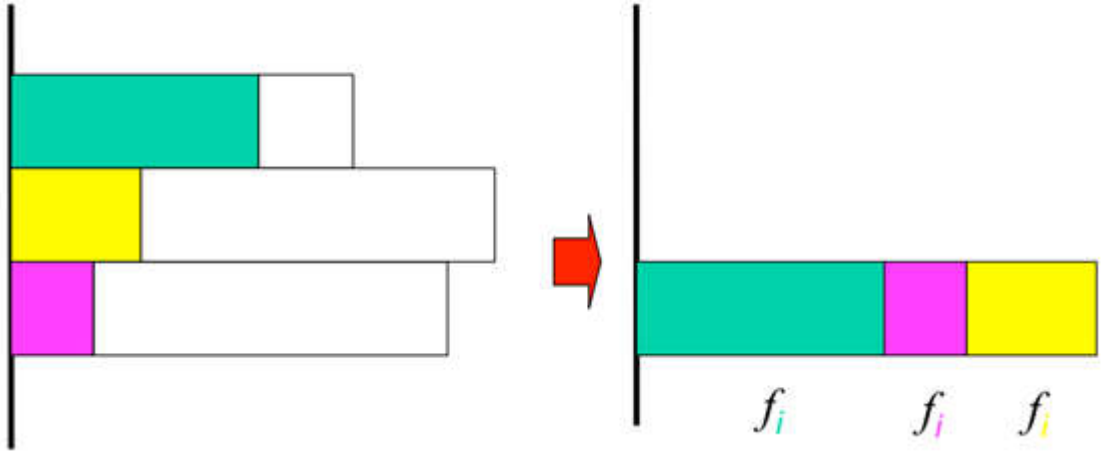
T1=5，C1=2
T2=7，C2=3



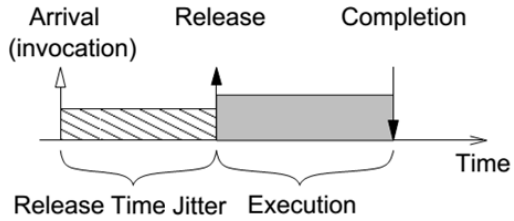
- DM算法：相对时限小的任务优先
  - 相对时限与周期正比时：DM=RM
  - 相对时限任意时：可能DM可行而RM不可行

EDD for uniprocessor

- 任务独立，with equal arrival times
  - Preemption is useless！
  - EDF支持到达时间不同的任务集
- Execute task with **earliest due date** (deadline) first
  - sorted by their deadlines：complexity is O(n log(n))
    - llxx：按绝对时限还是相对时限？
  - 如果两个任务时限相同，则其相对顺序任意
- 最优：指“最小化最大延迟”？

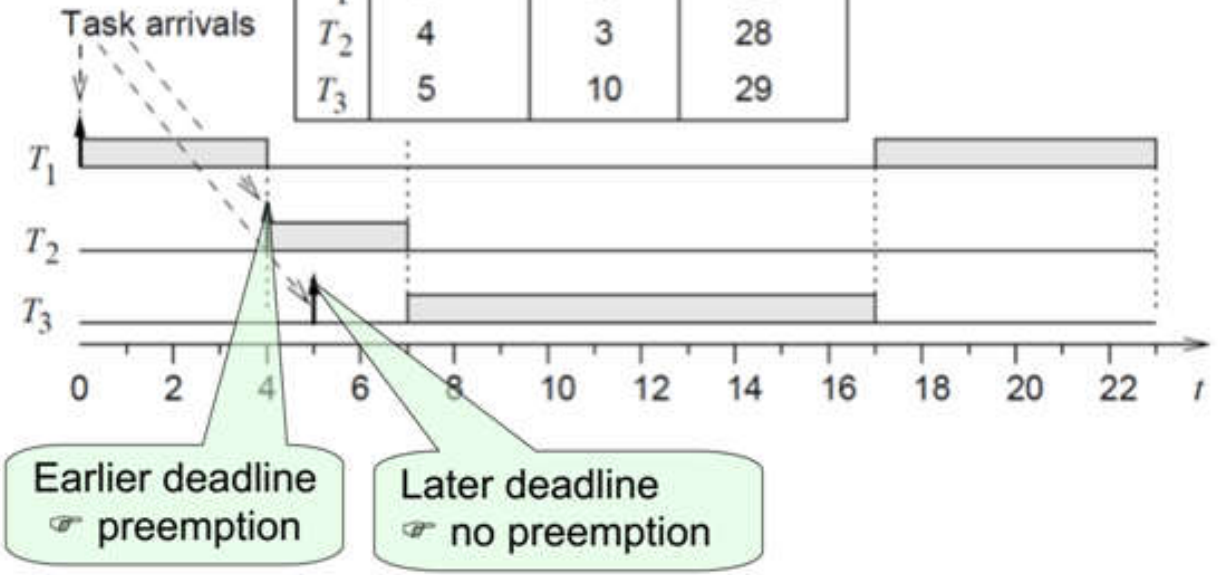


EDF算法：动态优先级调度



- Tasks may arrive at any time
  - 调度时刻：depends on arrival?
- 动态优先级分配
  - 优先级按绝对截止期dl排序，最早者优先，可抢占
    - 需要时间戳?
  - （几乎）已过截止期的任务，仍因优先级最高被调度
    - 可能造成多个任务都错过dl（domino effect）
- 实现：
  - uc/OS固定优先级?
  - ET/TT?

	arrival	duration	deadline
T_1	0	10	33
T_2	4	3	28
T_3	5	10	29

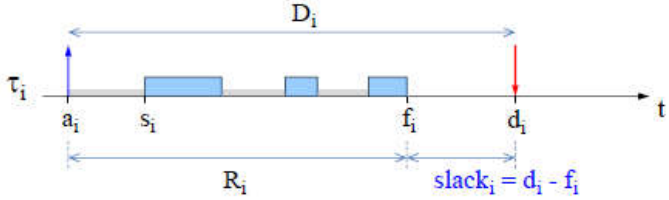


最小裕度Laxity算法（LLF/LSF/LST/MLF）

- 裕度：任务执行的富裕时间d_1
  - 记t为系统当前时间，E为任务估算执行时间，p为任务已经执行的时间，d为绝对截止期。则d_1：

$$d_1 = d - (t + E - p) \geq 0$$

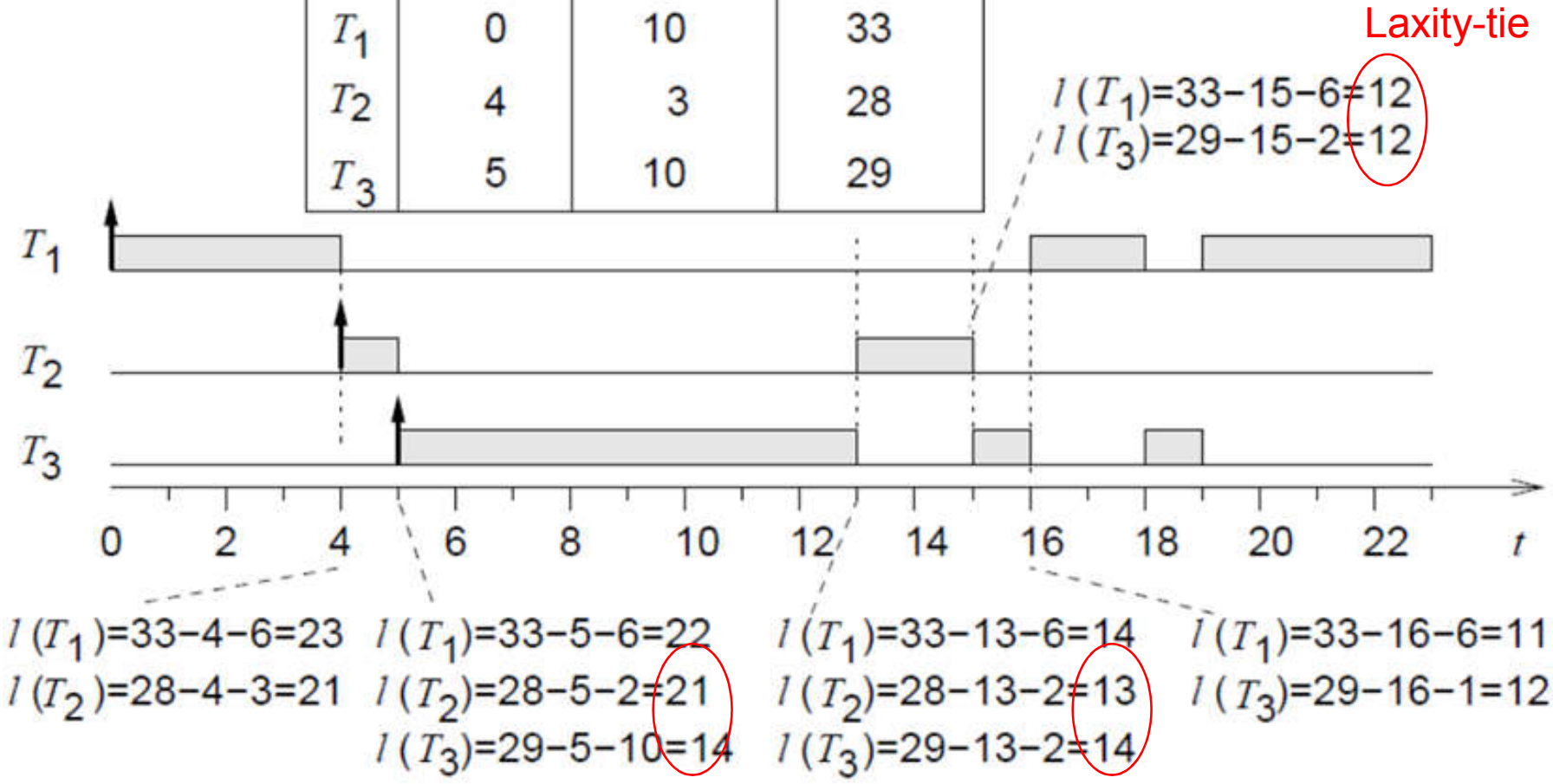
- 按裕度排序：裕度小的，优先级高，抢占
  - 正在运行的任务，其裕度不变；
  - 就绪队列上的任务，其裕度随着时间的推移而减少，从而使得它们的优先权动态地发生变化。
- 实现：时钟的滴答中断
  - 需累计任务的已执行时间
  - 对就绪队列上的任务的裕度进行累减排序



Least Laxity First示例

- detects missed deadlines early

	arrival	duration	deadline
T_1	0	10	33
T_2	4	3	28
T_3	5	10	29



RM与EDF（Buttazzo2003）

- RM和EDF是两种常用的算法
  - 商用RTOS多实现RM，EDF实现代价高，多限于研究用？
- 谁优？各自的适用场景?
  - 周期或时限相同的任务？

	FP/RM	EDF
Applications	critical, static	dynamic, less critical
RTOS implementation	easy	more difficult
Tasks	Periodic only	Aperiodic and periodic
Efficiency	upto 69 %	upto 100 %
Predictability	high	less than FP/RM if $U > 1$

- critical = periodic，non critical = aperiodic

RM、EDF、LSF

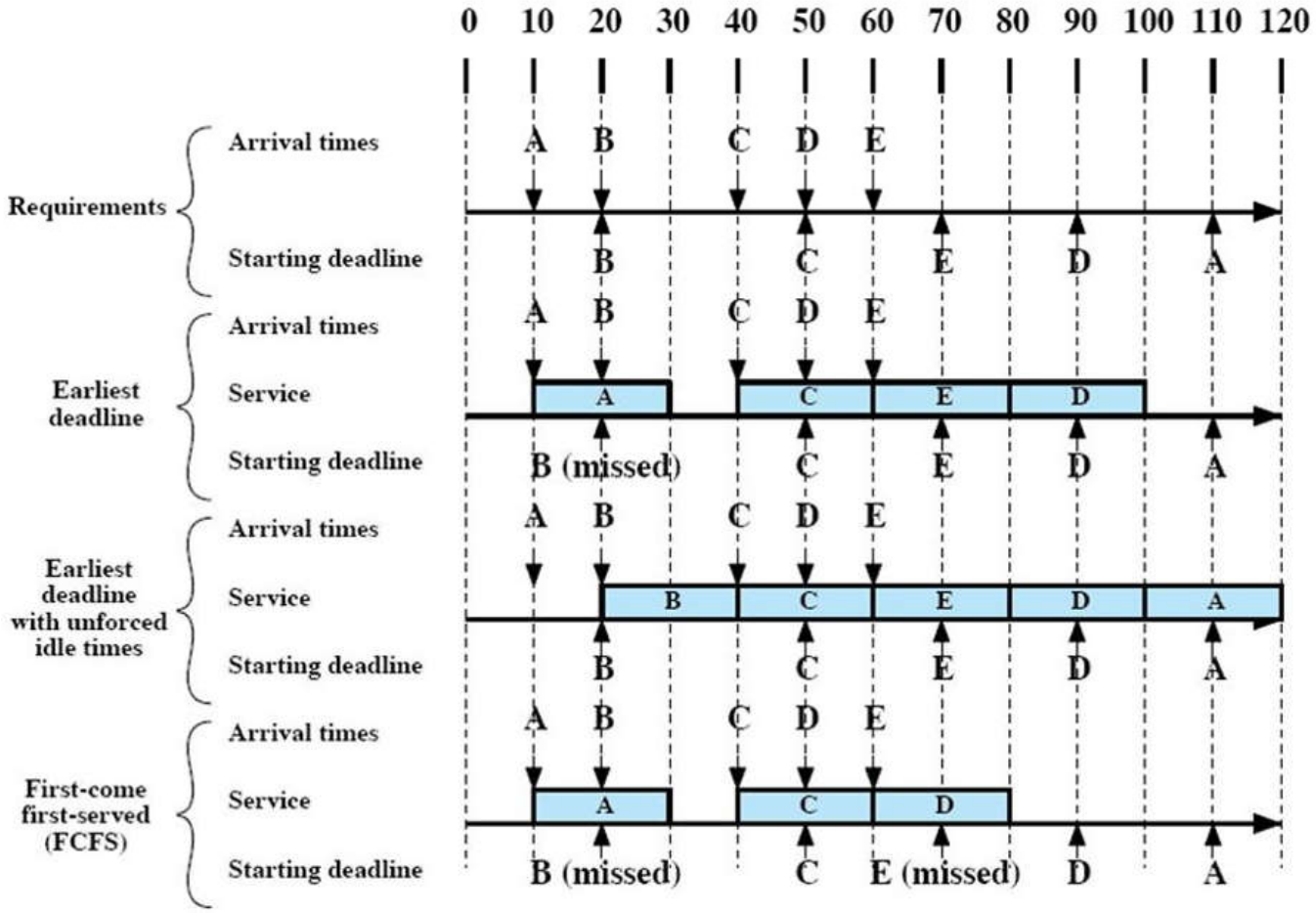
- RM实现简单，运行开销低
  - 需预留较大余量以保证最坏情况，导致平均利用率低。
  - 如果正在执行的任务行为不当，可能造成其他任务饥饿。
  - 简化实现：Bitmap Scheduler算法
- EDF/LSF实现复杂，执行开销大。
  - 满足定时约束效果较好(利用率高)
    - 动态调整每个job的优先级也需要systick
    - 需要monitor检查任务执行时间。
  - EDF比LSF抢占率低（任务切换少）
    - EDF每次job到达时计算其优先级，基于绝对时限排序
      - 根据预知的相对时限和当前时间计算该job的绝对时限
      - 注意：周期任务的到达时刻固定，非周期任务不定
    - LSF每个systick需要计算所有任务的slack，实现代价比EDF更高
      - 且存在Laxity-tie问题：多个任务余量相等时会不断进行切换。

LRF（反向EDF）：释放时间

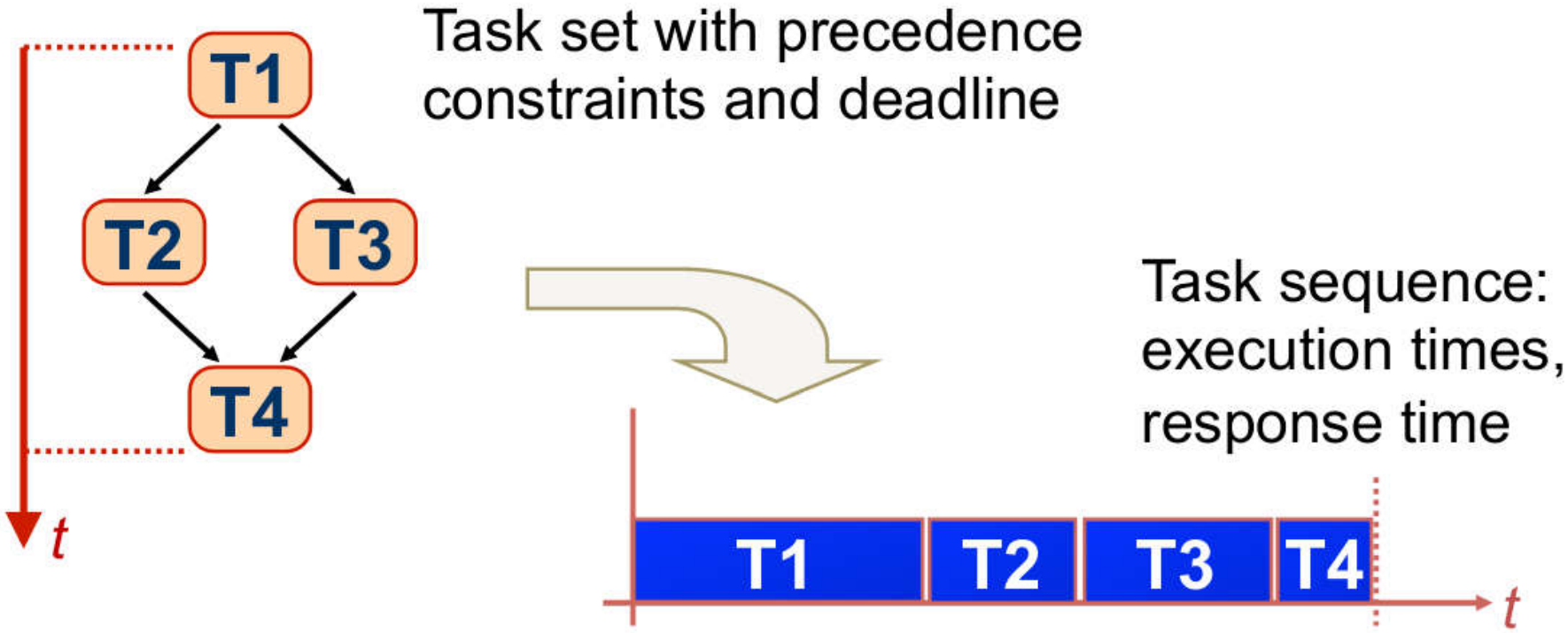
- 当调度目标是满足截止期时，过早完成任务不一定有益
- 因为某种原因推迟任务执行
  - 如：为了保证弱实时任务的响应时间
- 反向EDF
  - 从截止期最迟者开始调度
    - 将释放时间作为截止期，截止期作为释放时间
  - 释放时间越迟，优先级越高

具有开始时限的非周期性任务调度

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



带precedence约束的任务集

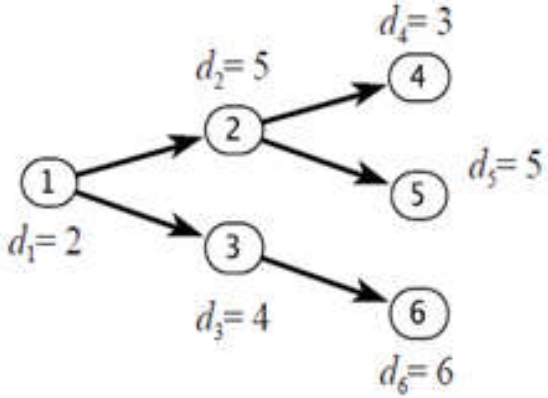


Can we guarantee that: response time < deadline?

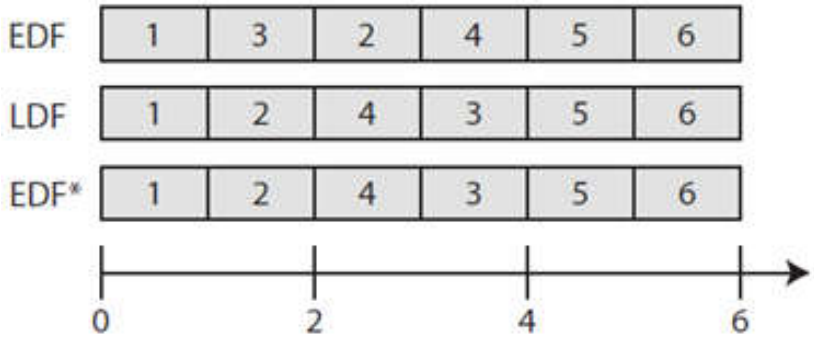
- 基于优先图

偏序约束任务调度：非周期，静态？

- 最大延迟最小化的最优算法
- LDF(Latest Deadline First,1973)
  - 任务同时到达（！）=> 非抢占(无须抢占)
    - 调度表：拓扑排序=>优先图，DAG
      - 后向调度：先插入无继承者，再插入有继承者
      - 并发：绝对时限最晚者优先(LDF)
    - 执行：按调度表逆序分派



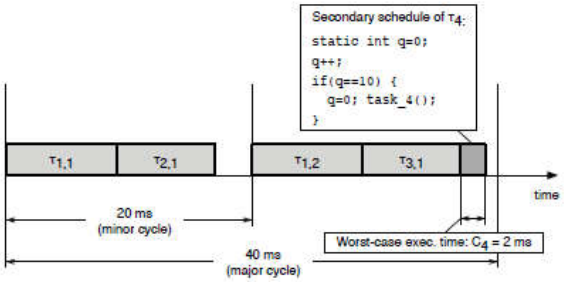
- EDF*：1990
  - 任务异步到达，可抢占
  - 将优先约束转为定时约束
    - 优先图：修改释放时间和dl
      - 有效释放时间：后继任务b须在前驱a完成后再释放，Rb*=max(Rb, max(Ra*+Ca))
      - 有效截止时间：前驱任务a须在后续b开始前完成：Da*=min(Da, min(Db*-Cb))
    - 有些实时参数只有在释放运行时方可获得
    - 需要时刻记录系统当前所有任务的偏序约束关系，存储开销大



资源约束：非周期

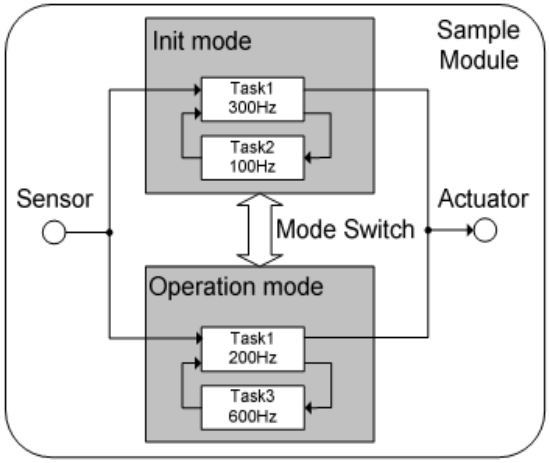
- Peter M. 的LS（List）和FDS（force-directed scheduling）算法

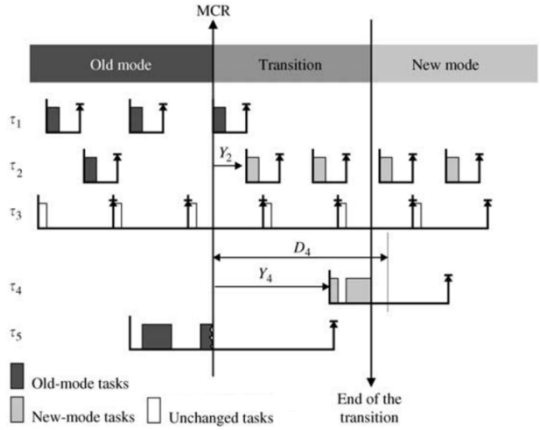
周期任务和非周期任务的混合调度



- 后台运行(background)法
  - 在处理器的空闲时间运行非周期任务
- 带宽保留(bandwidth-preserving)法
  - 另外增加一个专门用来执行非周期任务的周期任务
    - 要求：重复发生的偶发性任务必然存在最小时间间隔
      - 如果对任务工作总量无限制，则无法保证时限
  - 当该周期任务执行完现有的非周期任务，还有剩余的执行时间时，如何处理这些剩余的时间？
    - 优先级交换PE(priority exchange)
    - 可延期服务器DS(deferable server)
    - 偶发服务器SS(sporadic server)
- 时间挪用(stealing time)法：非周期任务很多时
  - 在保证满足已有硬实时周期任务的时限要求时，尽量挪用出部分时间来执行非周期任务（采用FCFS策略）
- EDF?

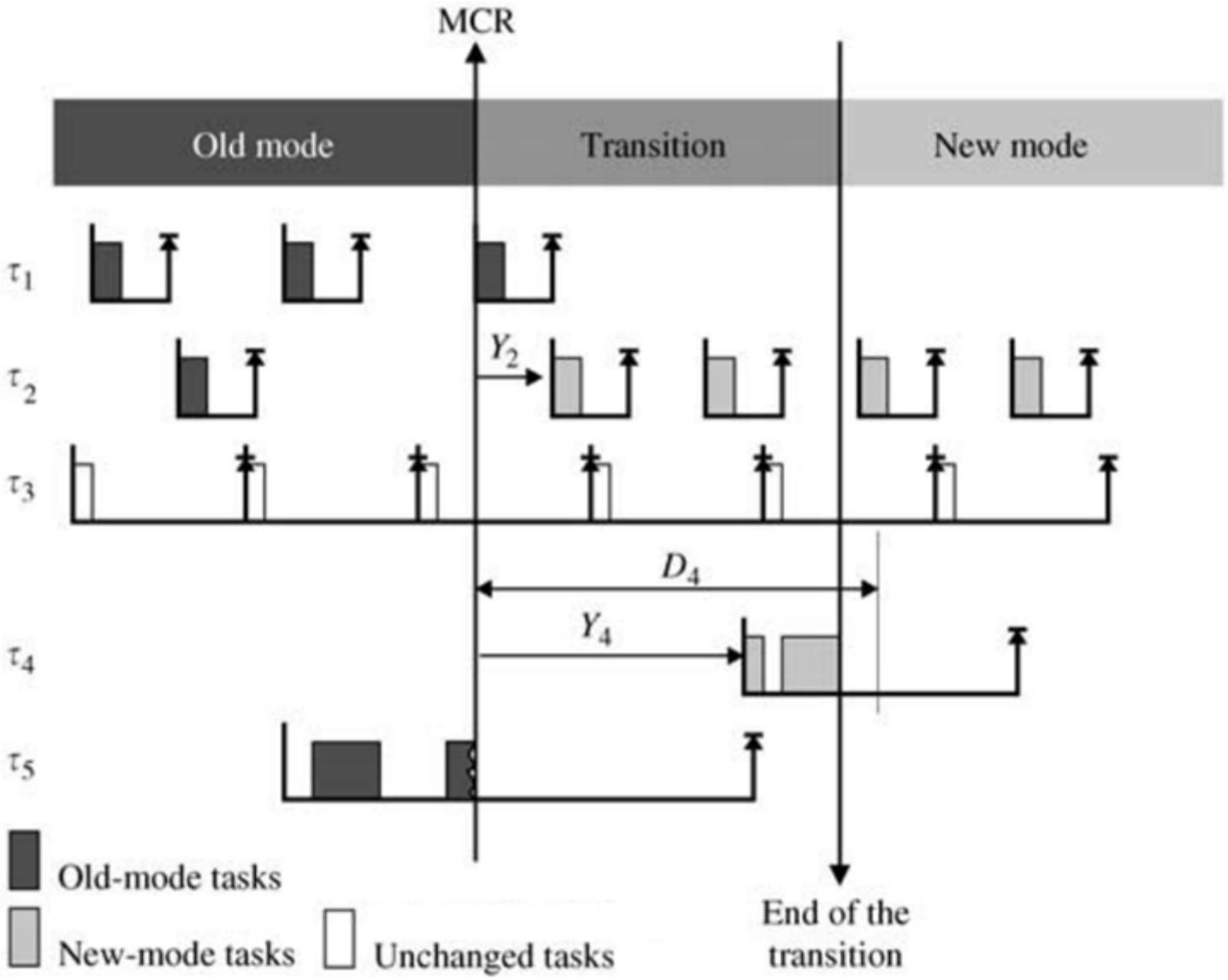
Mode Changes





- Mode：a set of tasks active at the same time
  - Different functionalities
    - changing operational modes
  - Different timing constraints
    - changing some parameters of tasks（eg. period）
- mode switch at run-time
  - Delete tasks：no longer released
  - Add tasks
  - not allowed to occur during a transition stage
- Transition stage
  - 由MCR（mode-change request events）触发
  - might execute tasks from both the old and new mode
    - 可能Overload
    - 解决：delaying the release(*offset*) of the first instance of a new-mode task
    - 存在mode-transition latency（*D*）
- Mode-change protocols
  - When：changing the schedule at an appropriate time instant
  - Rules：for adding or deleting a task

Mode change：transition phase



mode-change protocols criteria

- 在MCR时刻，旧任务立即放弃执行，还是正常执行直至MCR完成?
- 转换期间，不变任务按原周期执行，还是有相移?
- 新任务引入时机
  - 同步式：旧任务完成后才引入新任务
    - 新旧任务没有相互影响
  - 异步式：在转换期间组合执行新旧任务
- The schedulability can be tested by using the processor utilization equation or the response-time analysis

Complexities Arising in Real Systems

- Tick Scheduling问题
  - In many systems, the scheduler is activated only at clock interrupts.
    - also called time-based scheduling, or quantum-based scheduling.
  - must regard the scheduler itself as a high-priority periodic task.
  - may have additional blocking times due to the possibility that a job can be released between clock interrupts.
- Task model问题：假设不合理，如
  - all tasks的执行时间已知
  - there is no penalty for preemption
  - preemptions may occur at any time（立即抢占）
    - Assumed that the scheduler is activated whenever a job is released.
  - an unlimited number of priority levels exists.

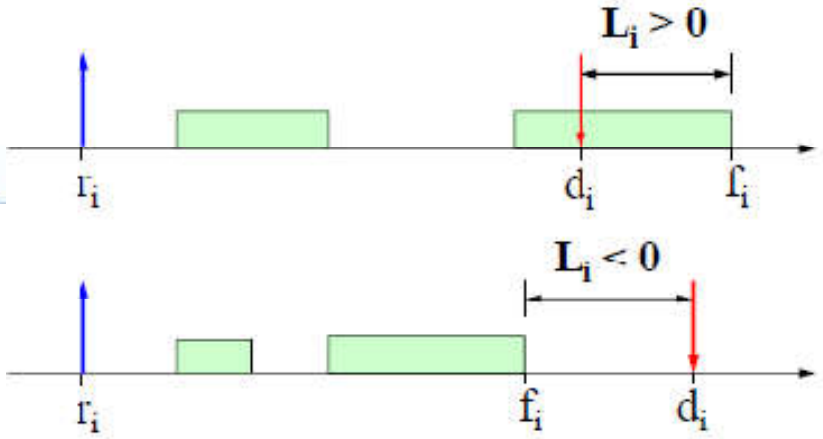
其他问题

- “fault tolerant” or “graceful degradation” solution
  - all deadlines are met：Each task has two implementations
  - Imprecise computation model：减少计算时间
    - 需要进行Sensitivity analysis，找到满足可调度性的计算时间
  - Importance Based Scheduling
- 多机/多处理器/多核：Multi-Machine Scheduling
  - MMS：不但要解决何时执行，而且还要解决何地执行
    - 同时涉及任务的分配与调度，也涉及某一个处理器的资源、网络通信等许多问题，大大增加了问题的难度
      - 已经证明此类调度多为NP-完全问题。
    - 在多处理器系统中，调度原则和算法的选择没有单处理器重要，往往使用FCFS就足够了。

The Real-time Scheduling Problem

- In a *valid* schedule for a set of jobs
  - Processors are assigned at most one job at once
  - jobs are assigned at most one processor at once
  - No job is scheduled before its release
  - Processor time assigned to each job equals its maximum execution time
  - All the precedence and resource usage constraints are satisfied
  - No timing constraints!
- A *feasible* schedule is valid
  - **and** jobs meet timing constraints（deadline）
  - not all valid schedules are feasible
- An *optimal* scheduling algorithm will always find a feasible schedule if it exists
- RT Scheduling：Given a set of tasks (ready queue)
  - Check if all deadlines can be met with a given schedule (schedulability check)
  - Construct a ”feasible” schedule to meet all deadlines
  - Construct an optimal schedule e.g. minimizing response times

Optimality criteria



- Feasibility
- Optimality
  - Find a feasible schedule if there exists one
  - Minimize the maximum lateness *Lmax*
    - **if** (*Lmax* < *0*) **then** no task misses its deadline
    - RM、DM、EDD、EDF、LLF、LDF、EDF*
  - Minimize the number of deadline miss
- Assign a value to each task, then maximize the value of the feasible tasks
- 最优 ≠ 实现开销小

调度算法的可调度性判定问题

- 不同调度算法对同一任务集的可调度性不同
  - 周期任务T1和T2同时开始执行，其执行时间C1、C2和周期P1、P2分别为：C1=C2=30ms，P1=50ms，P2=80ms。
- 如何确定是否可调度？ Timeline法

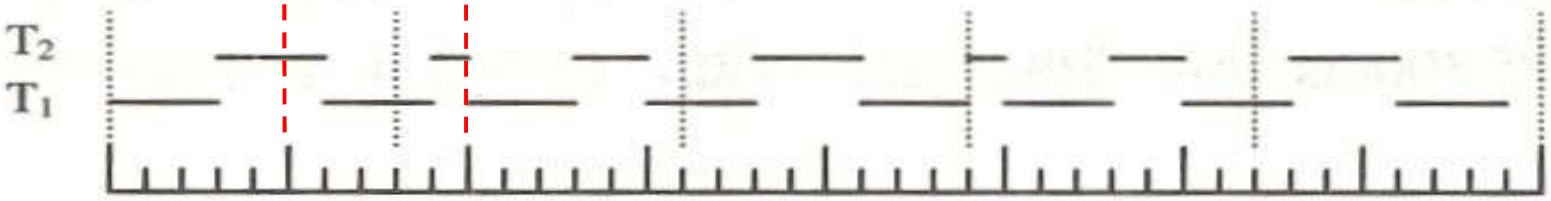


图 3.6　采用截止期优先算法时 T_1、T_2 的调度执行情况

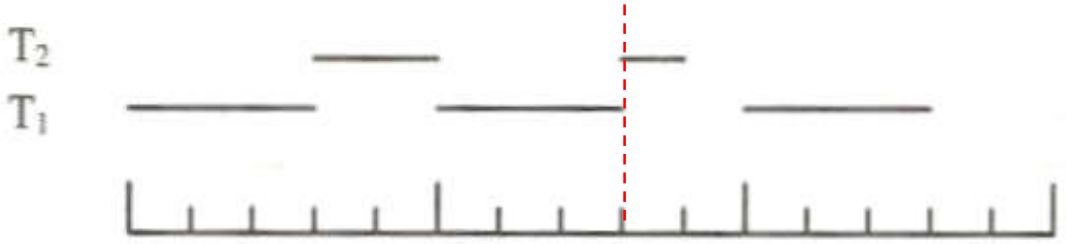
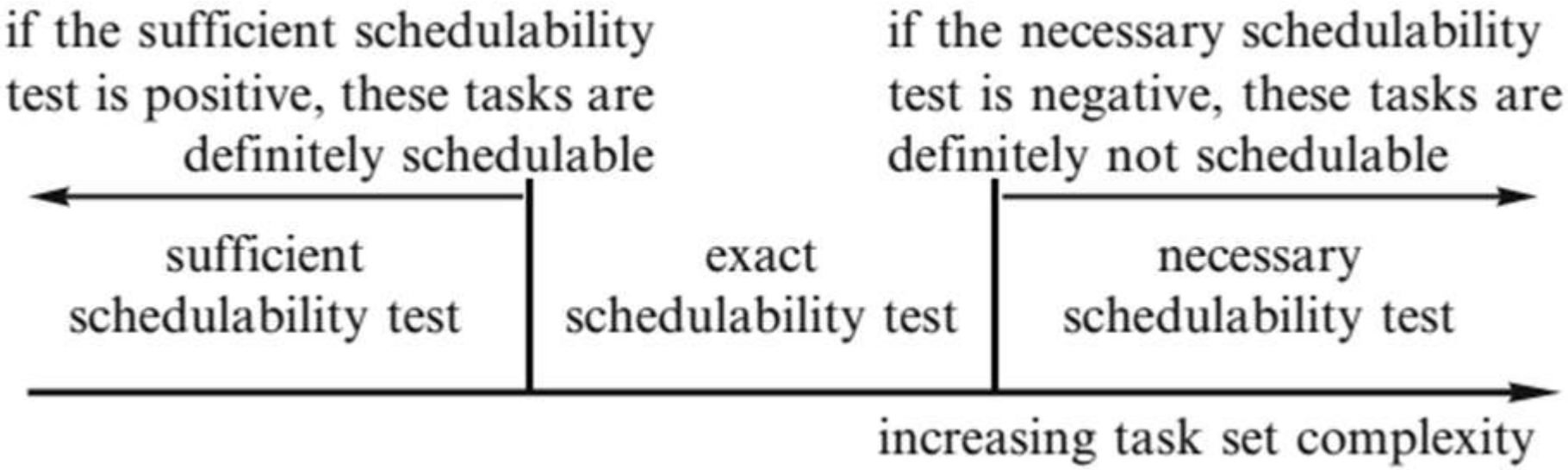


图 3.7　采用速率单调算法时 T_1、T_2 不可调度

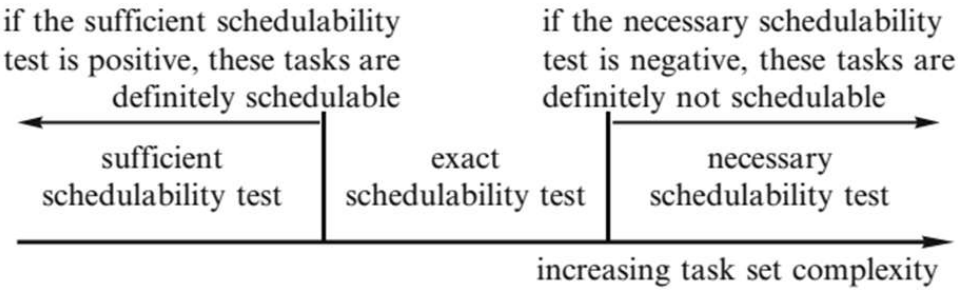
可调度性测试/分析/判定

- 问题：给定任务集和算法，调度算法可行？
  - 判断时间约束（时限 or 响应时间）是否满足
    - 检查最坏情况下系统资源的分配情况
    - critical instants：多个任务同时到达，且所有高优先级任务的抢占都发生了。
- 求解： Timeline法，计算法
- 依赖于任务集的复杂度



可调度性测试

- CPU利用率：周期=DL？
  - 测试任务集是否符合“处理器使用规则”
  - RM算法
    - 定理1“RM处理器使用规则”：某些超过RM使用率的任务集仍然是可行的
    - 定理2“第一deadline规则”： RM算法的完成时间
  - EDF算法：小于1
  - DM算法
- 响应时间分析法：周期≠ DL？
  - Timeline法：各任务每次执行的RT均不超过其时限，则可行
  - 计算法：RTA/DMA
    - 考虑fixed-period的任务间抢占，求每个任务的WCRT
    - 较基于处理器利用率分析准确，但可能不收敛。
- 非周期任务
  - Queuing theory
  - EDF算法：
    - “超周期任务时限满足分析”
- 非抢占？



可调度性测试

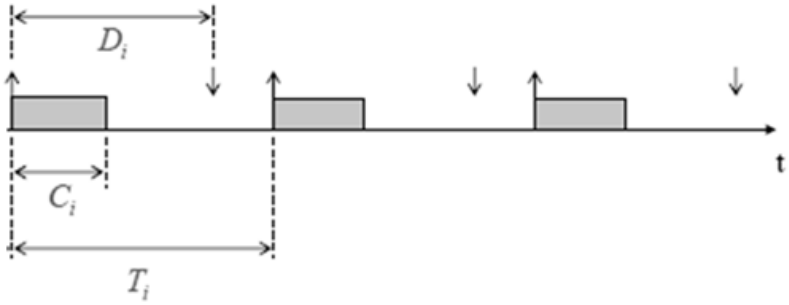
- U = (time executing / total time) * 100%

- **CPU利用率限定：“处理器使用规则”**
  - 有M个周期性的硬实时任务，处理时间为C_i，周期为T_i (or P_i)，时限=周期，N个处理机（器）
  - 可调度（**有效调度**）必须满足条件：
    - 单处理机系统：

      $$\sum（ C_i / T_i ）\leq 1$$

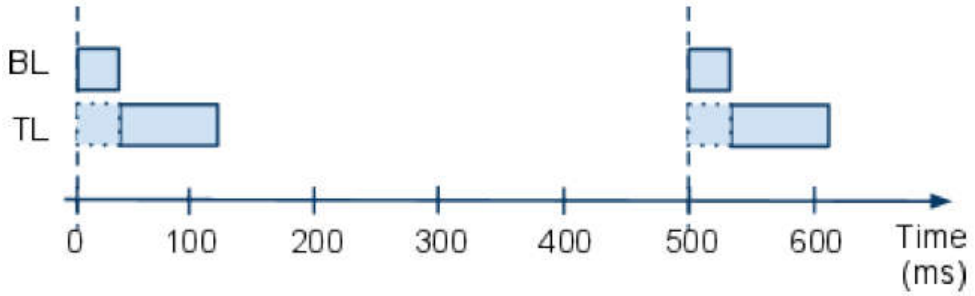
    - 多处理机系统

      $$\sum（ C_i / T_i ）\leq N$$

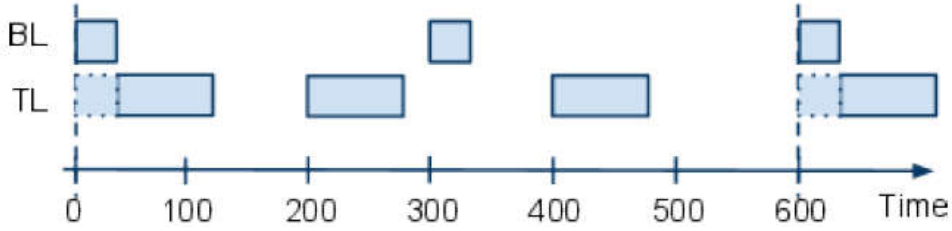


Utilization for multiple tasks

- *U* = (time executing / total time) * 100%
- 任务周期相同
  - BL: (500, 30)
  - TL: (500, 90)
  - *U* = (30 ms + 90 ms) / 500 ms = 24%

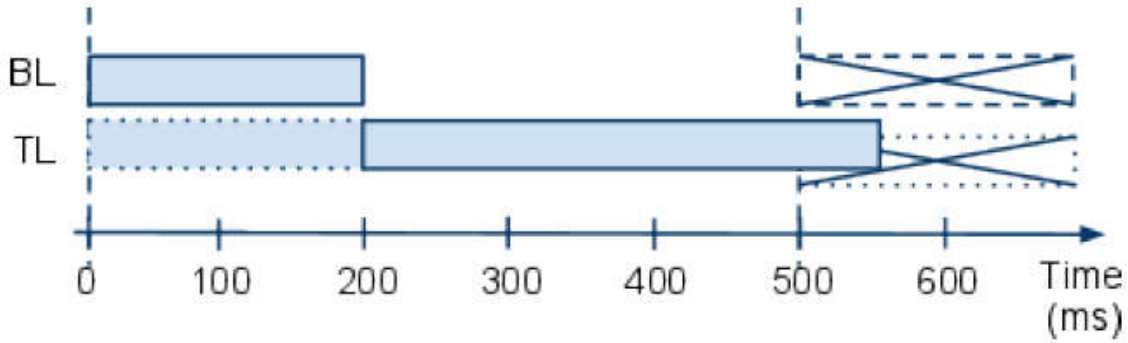


- 任务周期不同：Hyperperiod *U*
  - LCM of periods
  - BL: (300, 30)
  - TL: (200, 90)
  - *U* = (2*30 + 3*90)/600 = 30/300+90/200=55%

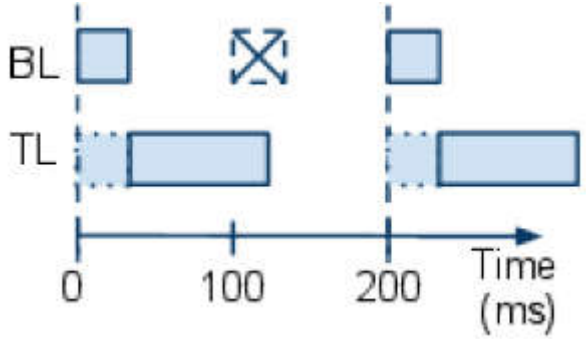


Overrun：过载

- Utilization > 100%
  - (200 + 350) / 500 = 110%
    - BL(500, 200)
    - TL(500, 350)



- Utilization < 100%，may still occur（非抢占?）
  - BL(100, 30)
  - TL(200, 90)
  - U = 75%



Overload：算法的鲁棒性（robust）

- Causes of Overload
  - Device faults delaying receipt of release time
  - Variable computation time of tasks which take longer than expected
    - Unanticipated resource contention or unavailability
    - switching to battery backup causes CPU to slow clock speed
- RM and EDF
  - exhibit poor performance when overloaded
    - domino effect：one timing fault can cascade into causing subsequent task(s) to miss their deadline(s)
- 过载处理：瞬间过载
  - 平均执行时间，最坏执行时间
    - 平均时间满足RM，有好有坏时满足
  - 关键性任务，非关键性任务
    - 系统QoS降级：使用任务的不同版本

利用率限定问题：是必要条件（有效调度）

- CPU利用率限定： $\rho = \sum_{i=1}^{n} \frac{C_i}{P_i} \leqslant 1$
  - RM: necessary condition（valid？）
  - EDF: necessary and sufficient condition（ feasible？ ）
- 例：周期P1、P2和执行时间C1、C2分别为：P1=50ms，P2=80ms，C1=C2=30ms。
  - 则ρ = 0.975。EDF或LLF可调度，而RM不可调度!
  - 验证：检查n个任务在P1×P2……×Pn时间里（超周期？）的执行情况

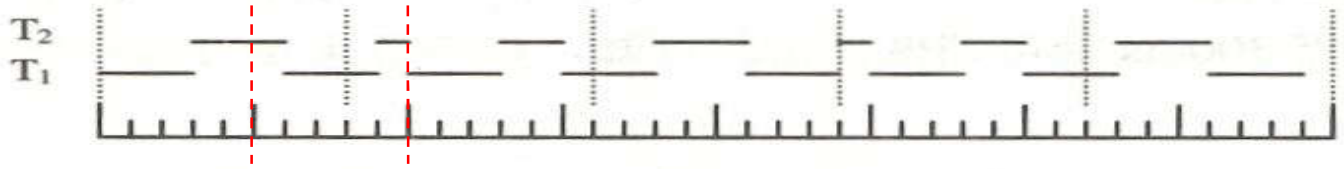


图 3.6　采用截止期优先算法时 T_1、T_2 的调度执行情况

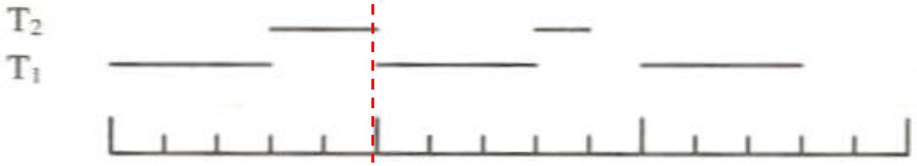


图 3.7　采用速率单调算法时 T_1、T_2 不可调度

RM算法的使用率限定定理（定理1）

- 所有的任务在同时开始。n个独立周期任务会在周期结束之前完成运行，只要满足条件(sufficient condition)：

$$\frac{C_1}{T_1}+\cdots+\frac{C_n}{T_n}\leqslant n(2^{1/n}-1)=U(n)$$

- 有界性：当任务的数量趋于无穷时，U(n)的上限值趋向于69%。
  - 当任务集的负载小于U(n)时，RM可行；
  - 但是当负载大于U(n)时，并不能判定RM 算法的可行性。

任务数 n	使用率界限 U(n)
1	1.000
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
7	0.728
8	0.724
9	0.720
无穷大	0.690

$$\rho=\sum_{i}^{n}\frac{C_i}{P_i}\leqslant n(\sqrt[n]{2}-1)$$

$$\lim_{n\to\infty} n(\sqrt[n]{2}-1)=\ln 2\approx 0.693147\ldots$$

RM算法的完成时间定理（定理2）

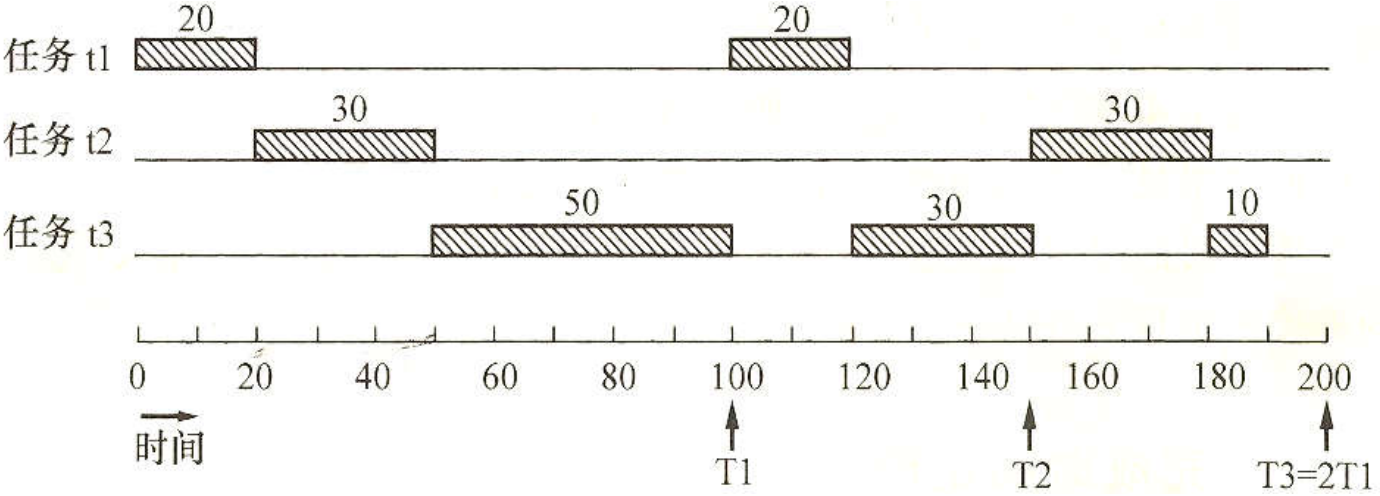
- 定理1的问题：不满足U(n)时，有些任务集仍可调度
  - 需检查*n*个任务在*T1*×*T2*……×*Tn*时间里的调度执行情况
- 定理2：完成时间定理（也称“第一deadline规则”）
  - 第一deadline：周期最大的任务的第一个调度点
    - 一组独立的周期任务，所有任务同时启动（arrive？）。
    - 如果每个任务都可以在其第一个周期时间内完成执行，那么启动事件的任意组合都可以使这些任务在各自的周期内完成执行。

任务 t_1：$C_1=20$；$T_1=100$；$U_1=0.2$

任务 t_2：$C_2=30$；$T_2=150$；$U_2=0.2$

任务 t_3：$C_3=90$；$T_3=200$；$U_3=0.45$

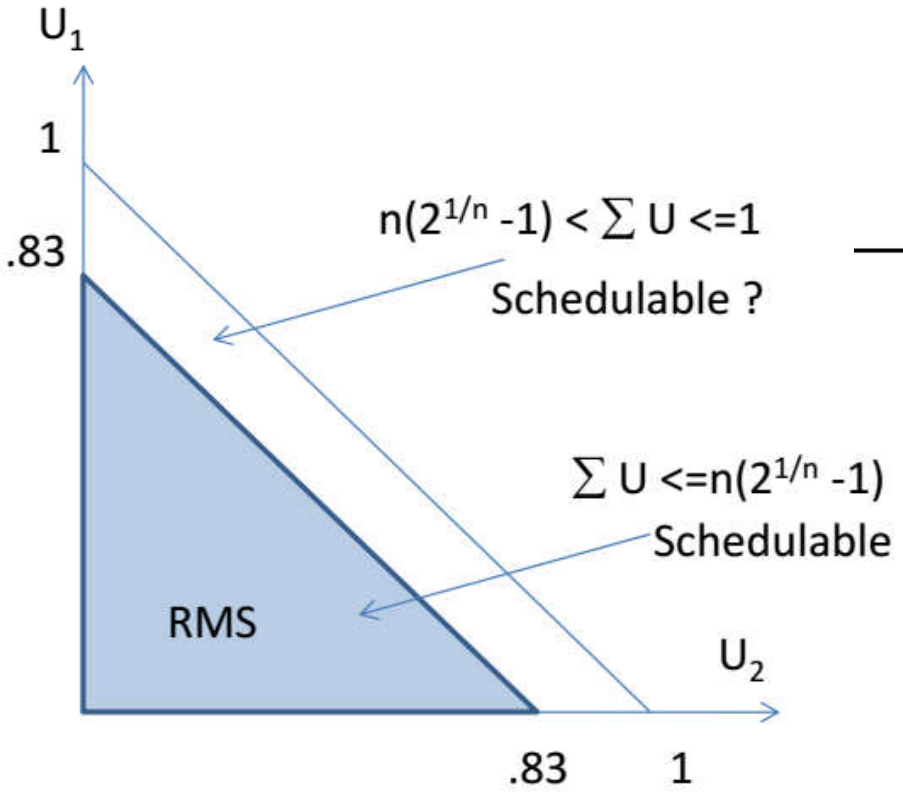
U(n)=85%，超过了77.9%

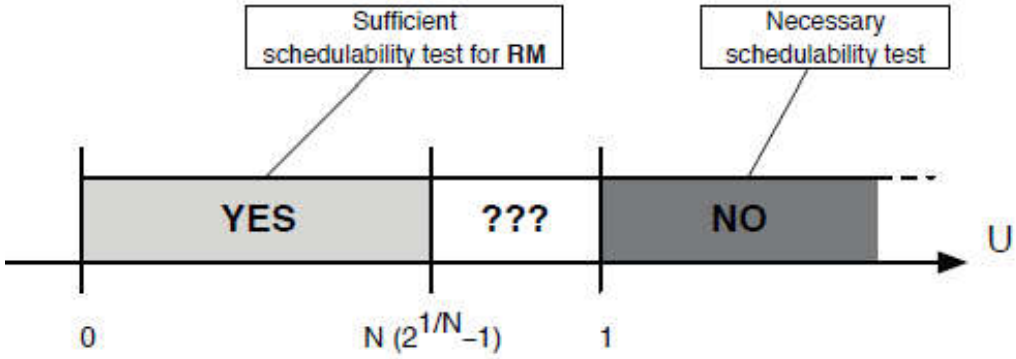


任务数 n	使用率界限 U(n)
1	1.000
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
7	0.728
8	0.724
9	0.720
无穷大	0.690

Schedulability region for RMS

- 可调度区间：Consider two tasks
  - Sufficient but not necessary condition: $U \le n(2^{1/n} - 1)$
    - 对*均匀分布*的任务集，处理器的平均使用率极限是0.88。
  - Necessary but not sufficient condition: $U \le 1$





任务数 n	使用率界限 U(n)
1	1.000
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
7	0.728
8	0.724
9	0.720
无穷大	0.690

DM schedulability analysis(DMA)

- sufficient condition：不是利用率！！！

$$\sum_{i=1}^{n} \frac{WCET_i}{Deadline_i} \le n\left(2^{\frac{1}{n}} - 1\right) \text{ with } Deadline_i \le Period_i$$

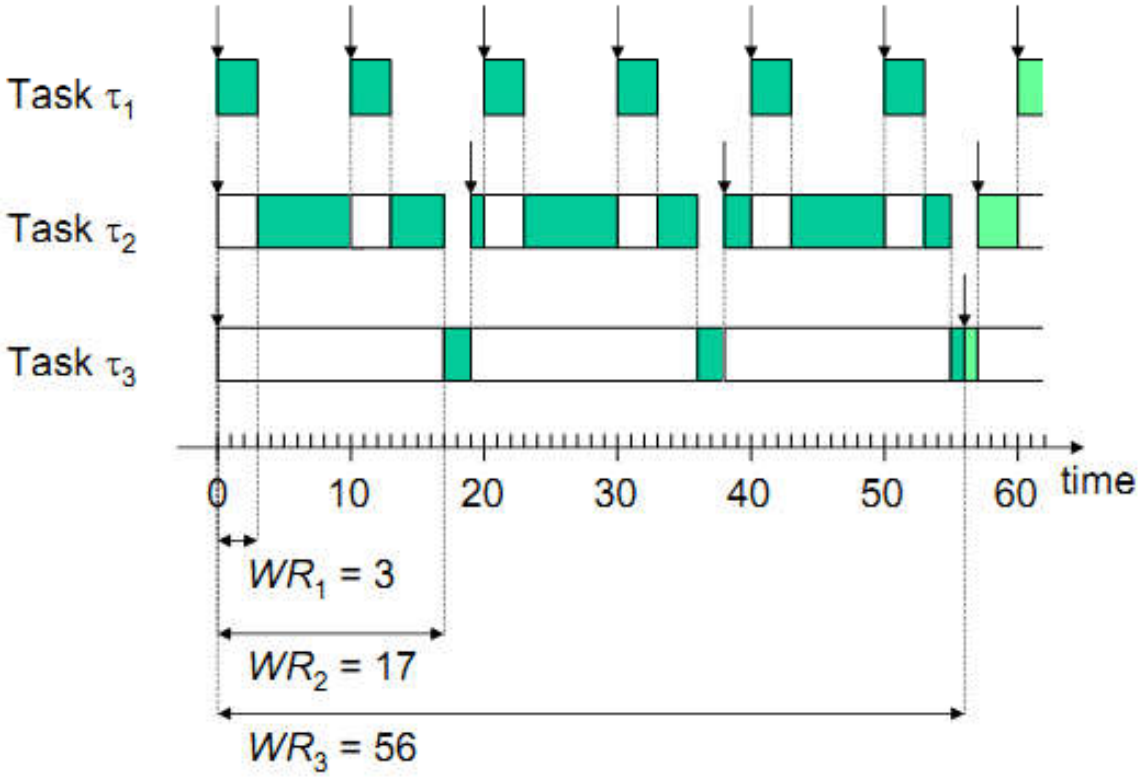
  - Schedulability region for DM？
    - 第二定理适用？

- RTA：Response Time Analysis
  - WCRT计算法：当所有任务同时请求执行时，如果每个任务的最坏响应时间都满足各自的截止期，则调度算法可行。

critical instant

- 一个任务的临界时刻就是比这个任务优先级高的所有任务同时发出请求的时刻.
- WCRT例

Task	Period T_j	Execution time C_j	Utilization U_j
τ_1	10	3	0.3
τ_2	19	11	0.58
τ_3	56	5	0.09



RTA：Response Time Analysis

- 求任务τ_i的WCRT R_i(到达/就绪到执行完成)
  - 对最高优先级任务，WCRT=执行时间。
  - τ_i 的WCRT计算：$R_i = e_i + I_i$
    - 此任务的周期*pi*，最大执行时间 e_i
    - 由所有高优先级任务干扰而造成的最大延迟I_i
      - 某个高优先级任务 τ_j 造成的（被）抢占延时 $= \lceil R_i/p_j \rceil e_j$
      - 所有高优先级任务造成的延时：$I_i = \sum_{j \in hp(i)} \lceil R_i/p_j \rceil e_j$
    - 则Recursive equation：

$$R_i = e_i + \sum_{j \in hp(i)} \lceil R_i/p_j \rceil e_j$$

- 非周期？

求解 $R_i = e_i + \sum_{j \in hp(i)} \lceil R_i / p_j \rceil e_j$

- 第n（$n=0,1,2,\ldots$）次迭代 $R_i^{n+1} = e_i + \sum_{j \in hp(i)} \lceil R_i^n / p_j \rceil e_j$
  - 当 $R_i^{m+1} = R_i^m$ 时，$R_i^m = R_i$
- 例：RM调度，三个任务
  - 短周期优先：1>2>3
  - 最高优先级任务1的响应时间 $R_1 = 3$
  - 任务2 = 7，因为：$R_2^0 = 4$

$$R_2^1 = 4 + \lceil 4/9 \rceil 3 = 7$$

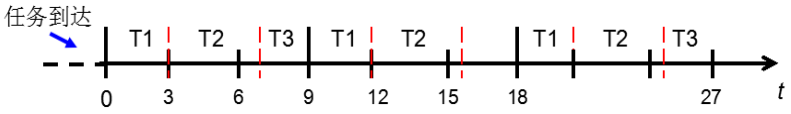
$$R_2^2 = 4 + \lceil 7/9 \rceil 3 = 7$$

  - 任务3 = 9，因为：$R_3^0 = 2$

$$R_3^1 = 2 + \lceil 2/9 \rceil 3 + \lceil 2/12 \rceil 4 = 9$$

$$R_3^2 = 2 + \lceil 9/9 \rceil 3 + \lceil 9/12 \rceil 4 = 9$$

τ_i	e_i	p_i
τ_1	3	9
τ_2	4	12
τ_3	2	18



例：分析WCRT？

- 有两个周期任务T1和T2同时开始执行，其执行时间C1、C2和周期P1、P2分别为：C1=C2=30ms，P1=50ms，P2=80ms。

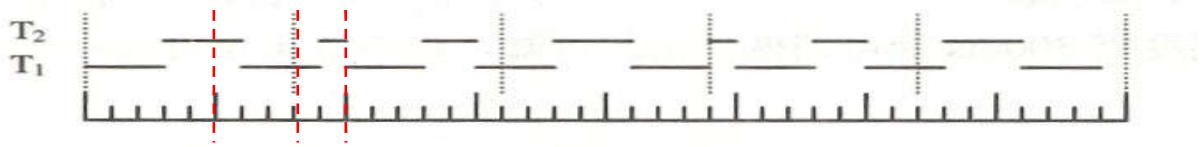
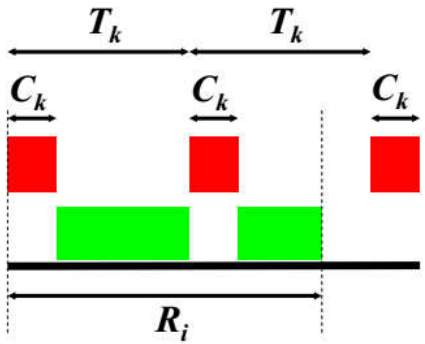


图 3.6　采用截止期优先算法时 T_1、T_2 的调度执行情况

图 3.7　采用速率单调算法时 T_1、T_2 不可调度



The interference is:

$$I_i = C_k + C_k$$

blocking factor

- 共享资源的周期任务集可调度性分析
  - 基于semaphore，PIP/PCP
- blocking factor：申请信号量时的最大延迟

$$R_i = B_i + C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- 影响*Bi*的因子
  - 持有者（优先级）
    - 占有时间
      - WCET of critical section
    - 阻塞类型：资源访问控制
  - 申请者优先级

Semaphore	Locked by	Time locked
S2	A	3
S4	G	3
S1	C	9
S2	E	13
S3	E	4
S3	F	4
S4	B	1
S5	G	7
S5	H	7

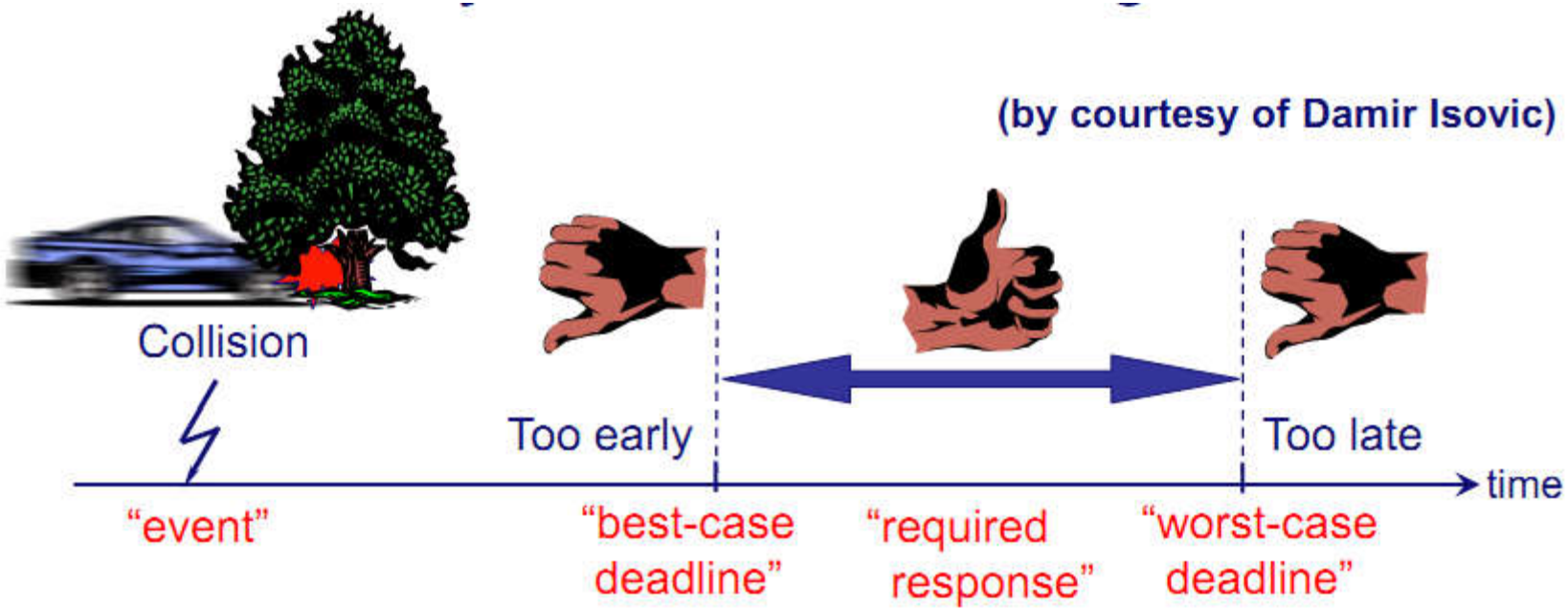
周期任务间的互斥、同步与通信

- 周期任务的临界区长度应固定

- 周期任务可以会合
  - 但应限制周期长度
- 不应允许周期任务与偶发任务会合
  - 偶发任务可能一直不发生

- 只有相容的任务才能进行通信（？）
  - 相容性：如果两个任务的周期是倍数关系，称两个任务是相容的（compatible）

例: inflation of an air bag



- Schedulability condition:
  - all jobs of all tasks must meet their deadline constraints

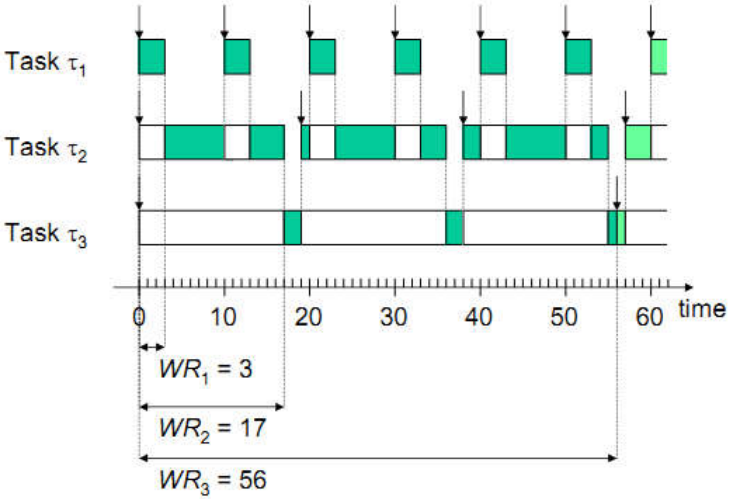
BCRT：Optimal instant

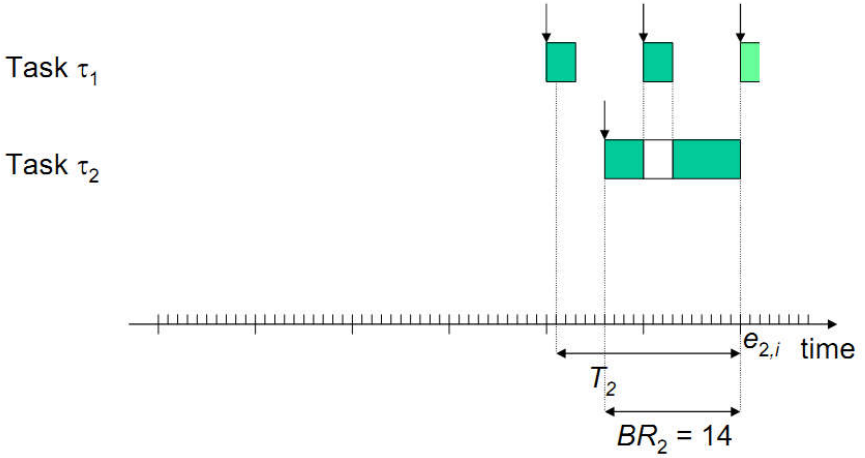
- Job *Ti* ends simultaneously with the release of all tasks with a higher priority, and *Ti*'s release time is equal to its start time.
  - 在所有高优先级任务释放时，Ti结束
- The lowest amount of preemption of a task is found before a simultaneous release of higher priority tasks.
  - 在所有高优先级任务释放前，任务被抢占次数最少
- Specific for each task？
- 可用于分析完成时间抖动

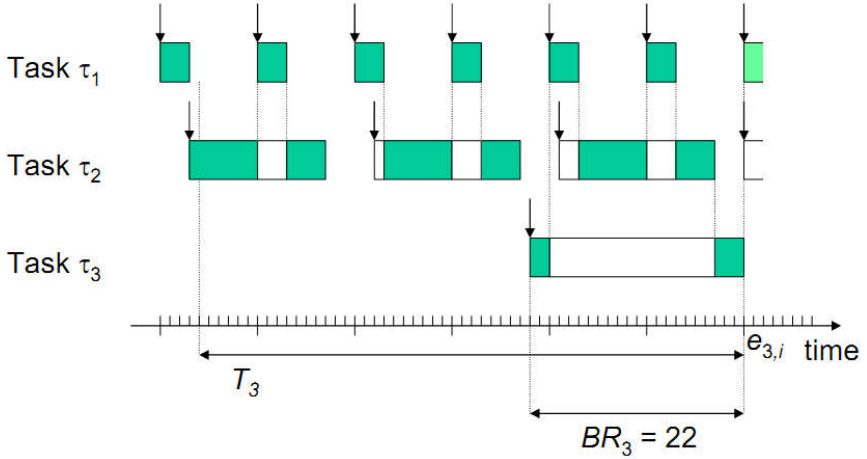
Example of optimal instant

- 任务在所有高优先级任务发出请求的时刻同时执行完成，且高优先级任务的抢占发生最少

Task	Period T_j	Execution time C_j	Utilization U_j
τ_1	10	3	0.3
τ_2	19	11	0.58
τ_3	56	5	0.09







BCRT Calculation

- Recursive equation

$$BR_j = C_j + \sum_{i<j} \left(\left\lceil \frac{BR_j}{T_i} \right\rceil - 1 \right) C_i$$

- Iterative procedure

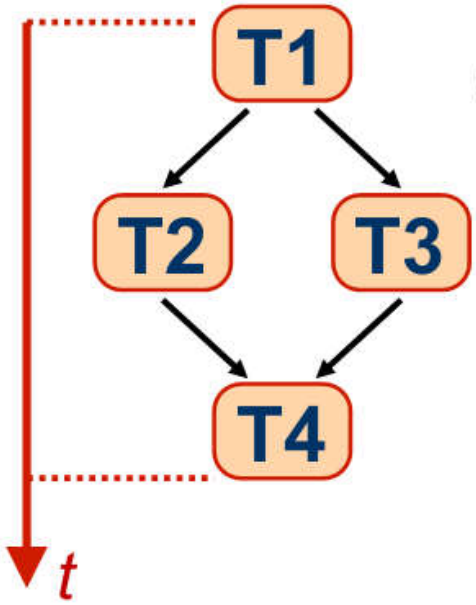
$$BR_j^{(0)} = WR_j$$

$$BR_j^{(k+1)} = C_j + \sum_{i<j} \left(\left\lceil \frac{BR_j^{(k)}}{T_i} \right\rceil - 1 \right) C_i$$

  - Stopped when:
    - the same value is found for two successive iterations.

带precedence约束的任务集

- WCRT分析法

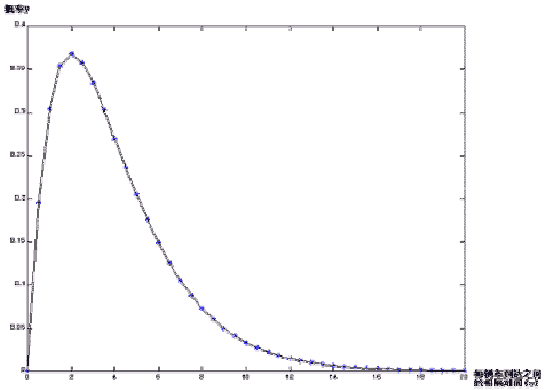


Task set with precedence constraints and deadline

Task sequence: execution times, response time

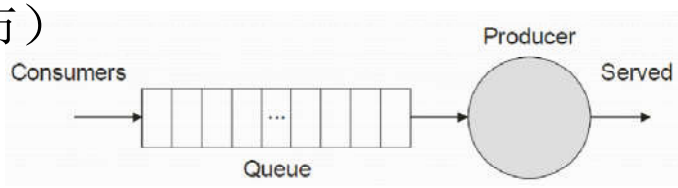
Can we guarantee that: response time < deadline?

sporadic和aperiodic的响应时间



- 响应时间 = 等待时间+服务时间
- Queuing theory
  - M/M/1 queue：默认为FCFS
    - 到达时间间隔/服务时间/服务器数
    - M：表示负指数分布（如Poisson分布）
    - 平均到达时间与服务时间之比：

$$\rho = \lambda / \mu$$



  - 队列中平均任务数 $\overline{N} = \frac{\rho}{1-\rho}$

$$P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!}$$

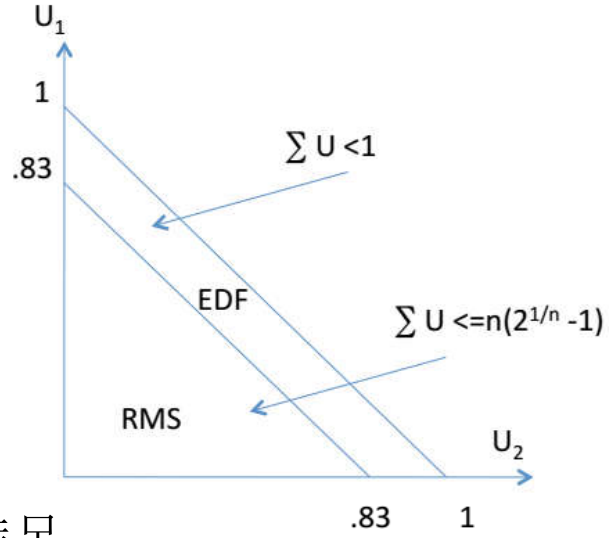
  - 平均任务服务时间（响应时间） $T = \frac{1/\mu}{1-\rho}$

  - 队列中至少有*k*个任务的概率 $P[\geq k \text{ in system}] = \rho^k$
    - 推论：两个事件同时到达的概率远大于三个以上同时到达的概率（~无需考虑）！

实时调度与分析示例

- 有4个任务：2个任务是定期任务，2个是非定期任务。其中一个非定期任务t_a是中断驱动任务，必须在中断到达的200毫秒之内执行，否则就会丢失数据。另一个非定期任务t_2的最坏情况间隔时间为T_2，这个时间要作为等价定期任务的周期。
- 各个任务的详细特征如下，所有的时间均以毫秒为单位，使用率$U_i=C_i/T_i$：
  - 定期任务 t_1：$C_1=20$；$T_1=100$；$U_1=0.2$
  - 非定期任务 t_2：$C_2=15$；$T_2=150$；$U_2=0.1$
  - 中断驱动非定期任务 t_a：$C_a=4$；$T_a=200$；$U_a=0.02$
  - 定期任务 t_3：$C_3=30$；$T_3=300$；$U_3=0.1$
- 另外，任务t_1、t_2、t_3都访问相同的数据存储，这个数据存储由信号量s保护。
- 4个任务是否满足最后期限？

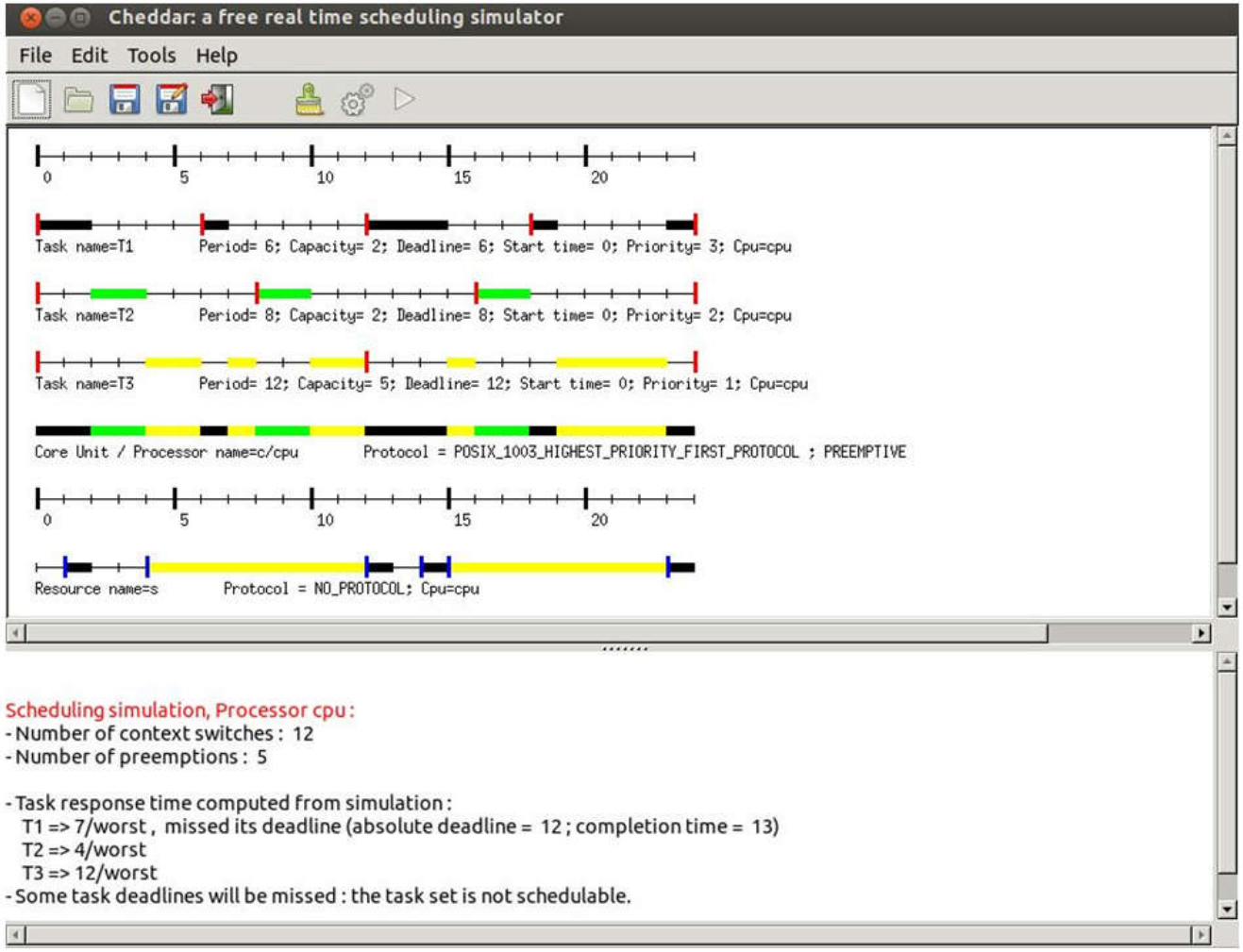
算法可调度性测试小结：响应时间满足时限？



- 周期任务
  - U ≤ 1：必要条件
    - EDF的充要条件
  - RMA：D = P
    - RM定理1：U ≤ U(n)，充分条件
      - 某些任务集 U(N) < U < 1仍可调度
      - *必要条件：Lehoczky89（计算复杂度高）*
    - RM定理2：所有任务在第一deadline都满足
      - 检查n个任务在P1×P2……×Pn（谐波为超周期）时间中的执行情况
  - DMA：D ≤ P
    - U' ≤ U'(n)充分条件。Schedulability region用RTA分析。
  - 带precedence约束的任务集：timeline法或RTA计算法
- 非周期
  - 任务集在某一时段内的计算时间必须小于可用的时间
    - Queuing theory
    - *EDF：Processor Demand Criterion法*

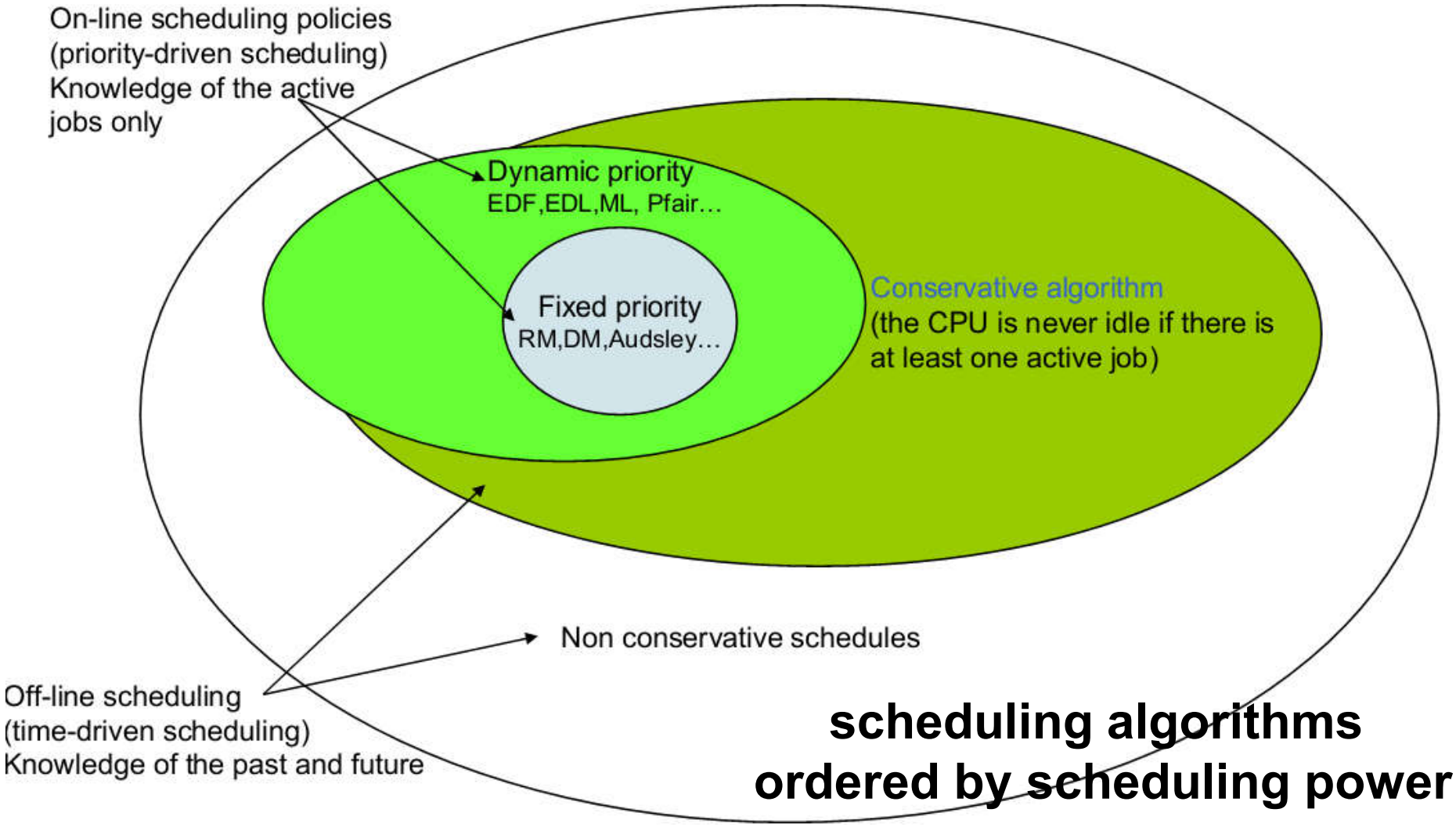
Scheduling Analysis Tools

- TimeWiz
- Cheddar



小结：调度算法

周期/非周期，抢占/非抢占，时间片/优先级（静态/动态），离线/在线

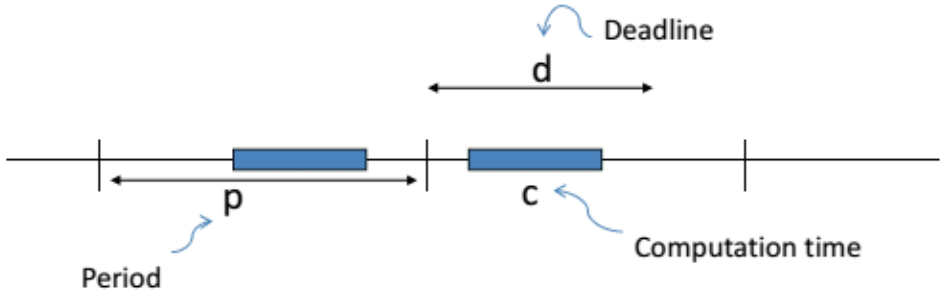


scheduling algorithms ordered by scheduling power

小结：实时任务与调度算法的特征

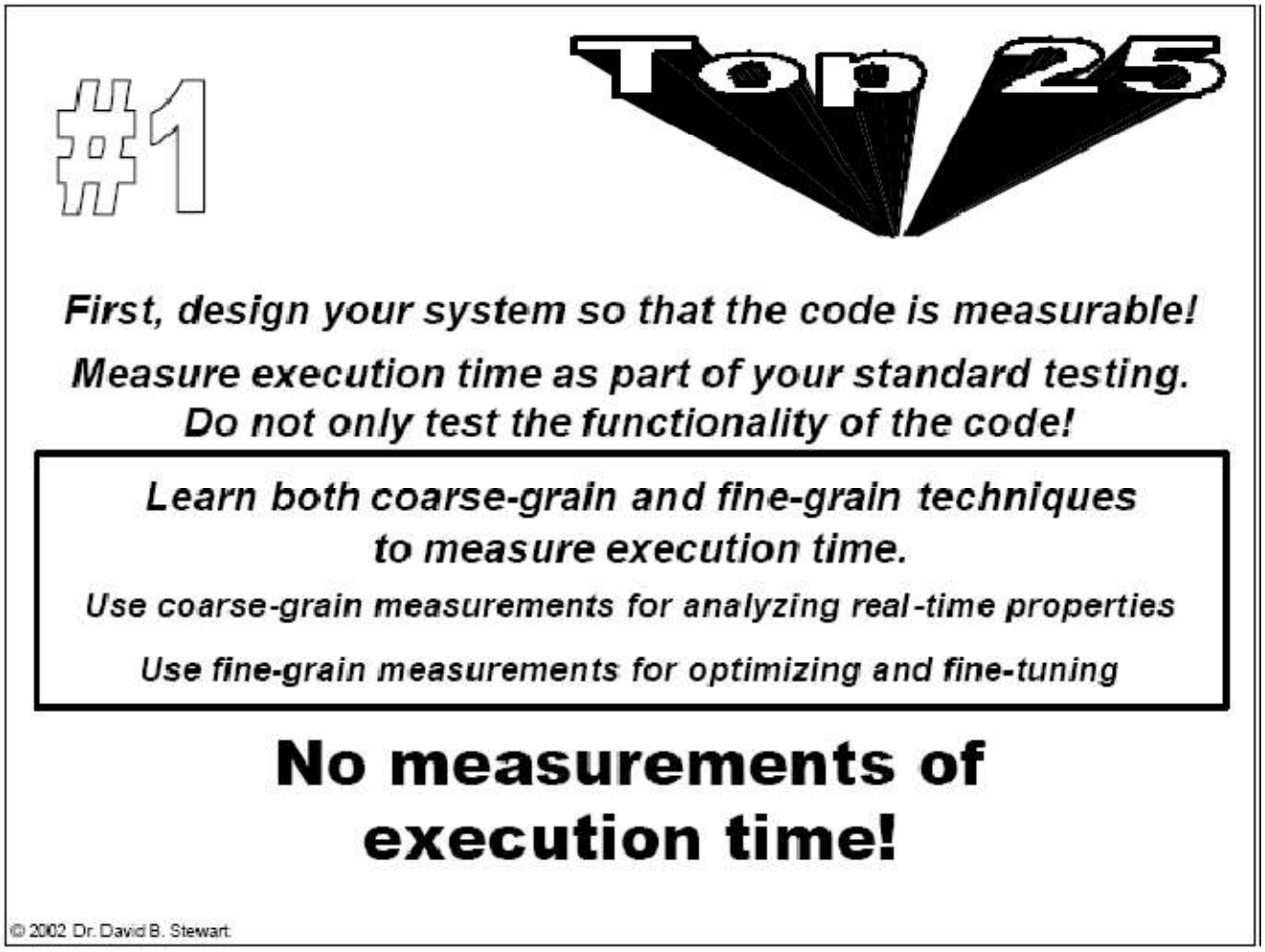
- Soft and hard deadlines
- Scheduling for periodic and aperodic tasks
  - sporadic tasks
- Preememptive vs non-preemptive
  - Suspend tasks. Can result in upredictable delays
- Static and dynamic scheduling
  - Static. Uses a priori knowledge about deadlines and arrival times
    - Timer triggers dispatch based on table. Predictable
  - Dynamic useful in reacting to sporadic events
    - Based on only what know so far
- Dependent vs independent tasks
- Works with OS with fixed priorities
  - uc/OS支持RM？EDF？

小结



- 任务类型
  - 周期：deadline = or ≠ period
    - periodic “world”：all real tasks of real applications are periodic！
  - 非周期：aperiodic，sporadic
- 时间模型
  - arriving time/release time，deadline，response time
- 调度策略与实现机制：调度时机？
  - Offline调度是valid的
    - 除了时间约束，还可以考虑顺序和资源约束
    - need to know everything: zero flexibility
    - *deterministic,* i.e., know exact what is going on when
  - 任务调度算法与资源访问控制协议
    - RM：无死锁，且采用PCP时，优先级翻转不超过一次
- 实时系统时间验证
  - 可调度性分析，WCRT分析

Most Common Mistakes with Real-Time Software Development



作业（选4）

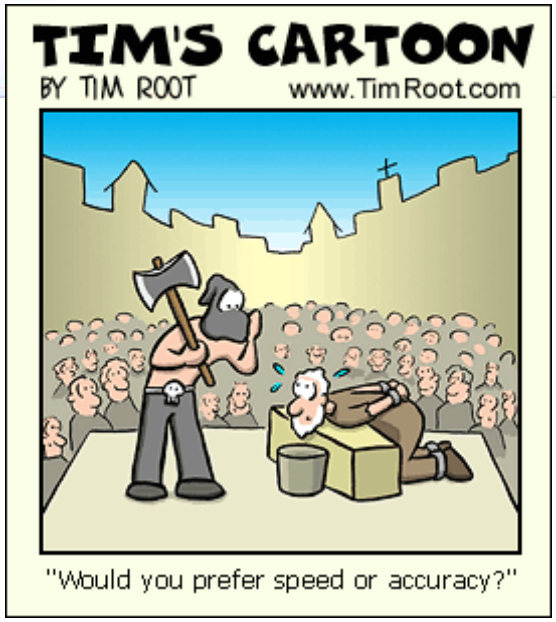
- 何时需要同优先级？
- EDF是否支持周期任务？
- EDF最坏响应时间分析？

- 可调度性分析工具TimeWiz、Cheddar比较

- Dependent Tasks的调度问题？

- uC/OS如何支持周期任务?
- uC/OS如何支持非抢占?
- uC/OS如何支持EDF?

- Liu《实时系统》作业4.1


TIM'S CARTOON
BY TIM ROOT
www.TimRoot.com
"Would you prefer speed or accuracy?"

Thank You