实时调度

李曦

llxx@ustc.edu.cn

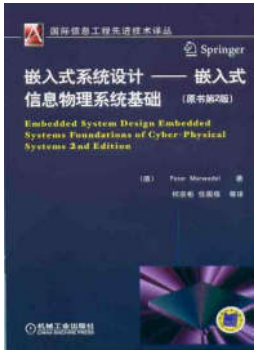
计算机系计算机应用研究室

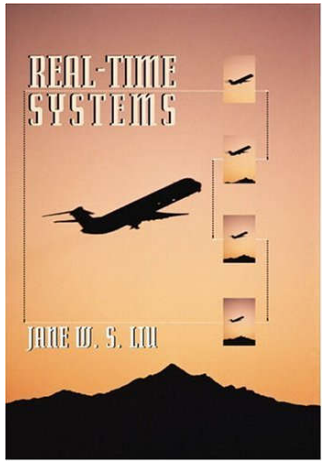
内容提要

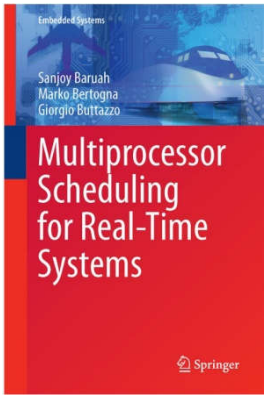
- 实时任务约束模型
  - Assumptions about task timing，interaction，。。。
- 任务调度算法Scheduling Algorithm
  - Scheduling mode and selection function
    - Timeliness：deadline，worst response time，。。。
    - Efficiency：average response time，makespan
  - Prioritized goals
    - Temporal predictability first，performance 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 $D \leq T$
- 多处理器调度：优先级贪心调度，Pfair调度
- Timing Analysis(WCET分析)

参考文献

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





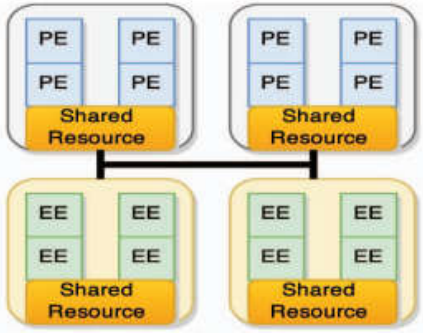


Multiprocessor/distributed

- Multiprocessors systems
  - is a set of autonomous processors which have software to coordinate them self and share resources
  - Share the same clock
  - Share the same main memory
- Distributed systems
  - is a set of autonomous processors that are connected by a network and which have software to coordinate themself or share resources
  - No shared memory. Message passing communication.
  - A clock for each processor.

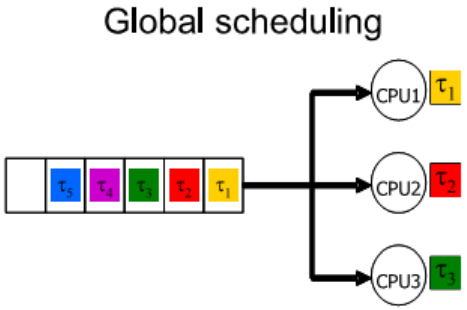
Multiprocessors/MultiCore Arch.

- homogeneous processors
  - processors have the same computing capability and run task at the same rate
- heterogeneous processors
- Cluster Heterogeneous MPSoCs
  - ARM的big.LITTLE架构
    - Cortex-A15 MPCore+Cortex-A7
    - 理论上可以使电池的使用寿命延长高达70%

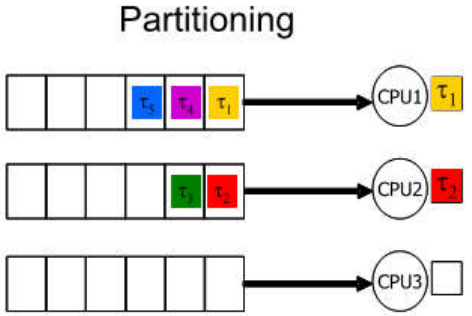


multiprocessor scheduling

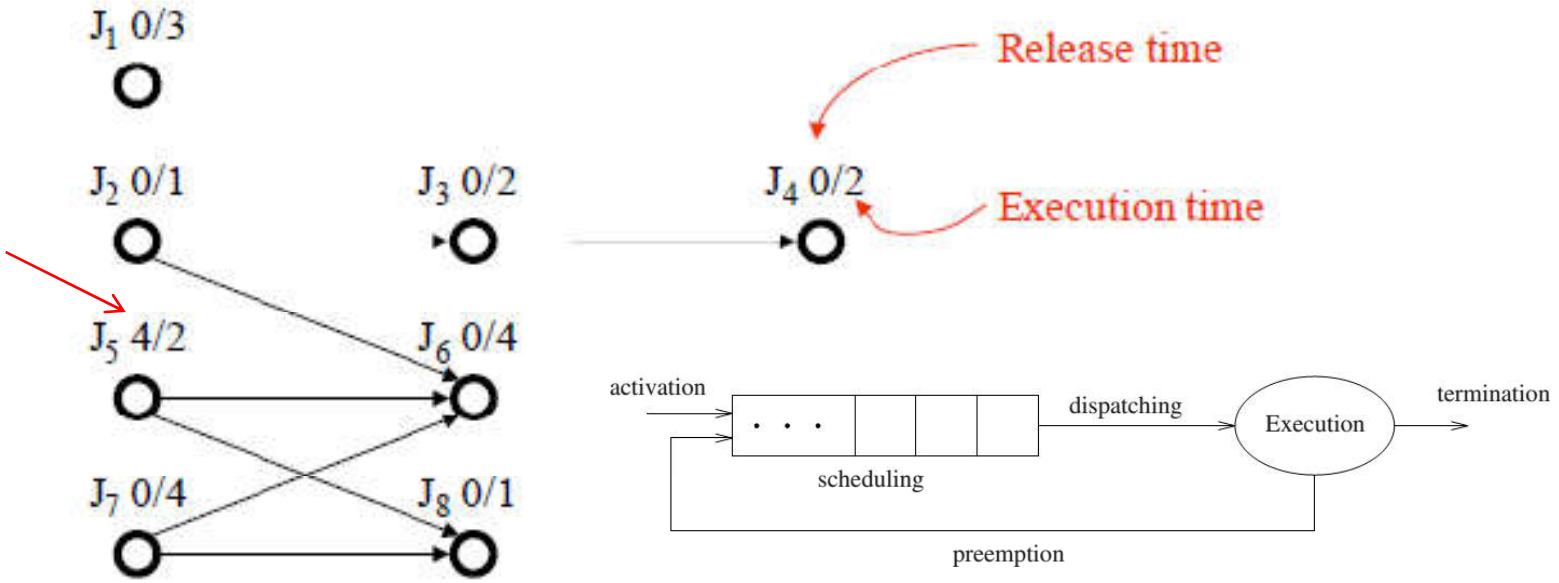
- Where and When
- Global scheduling
  - on-line：为任务分配/抢占一个空闲的处理器。可迁移
    - the level of migration：Task/Job level migration
  - 可迁移造成分析困难
  - 适于多处理器系统：优先级贪心调度，Pfair调度
  - expect optimal processor usage
    - busy processors, less preemptions ...



- Partitioning scheduling
  - off-line：每个处理器一个任务队列。非迁移
  - “bin-packing” problem：NP-hard
  - 适于分布式系统（ARINC 653）
  - expect minimize the number of processors, the number of communications, latencies, ...
- hierarchical scheduling
- heuristic



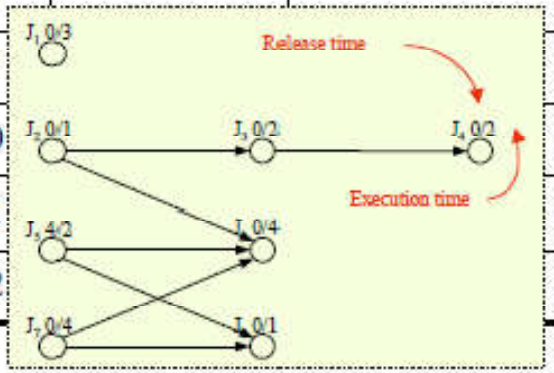
多处理器优先约束：优先级调度法



- *P*1、*P*2两个处理器
  - 任务基于共享内存通信（因此，通信开销可忽略）
- Priority-Driven Scheduling：任务号*i*小，则优先级高
  - The schedulers keep one common priority queue of ready jobs
  - 贪心：尽量不让处理器空闲。局部最优。
- 所有任务可抢占：ET?
  - 调度时刻：任务就绪（ready）或完成
    - 注意：因为存在优先约束，ready≠release

Priority-Driven Scheduling List

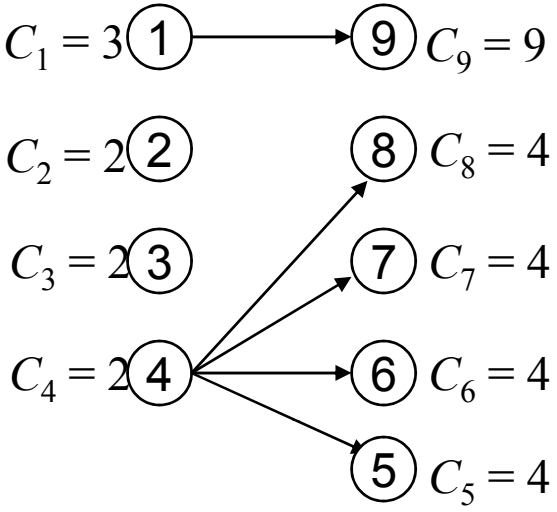
Time	Not yet released	Released but not yet ready to run	Ready to run	P_1	P_2	Completed
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						



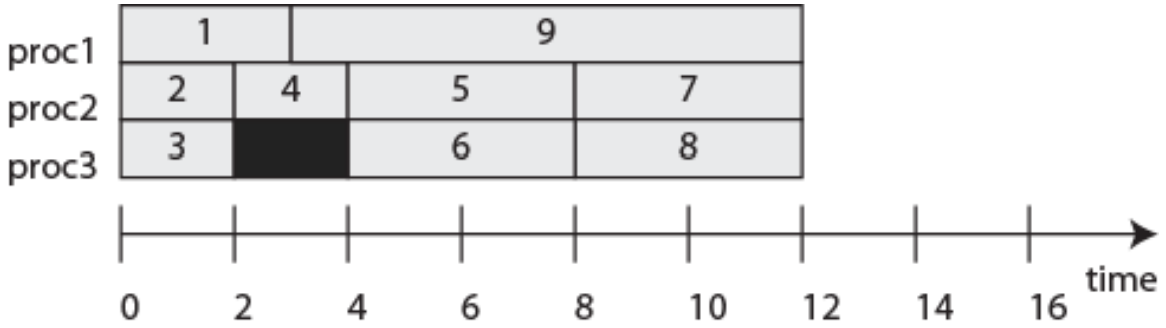
Theorem (Richard Graham, 1976)

- Brittleness
  - In general, all thread scheduling algorithms are brittle: Small changes can have big, unexpected consequences.
- Richard's Anomalies
  - If a task set with fixed priorities, execution times, and precedence constraints is scheduled according to priorities on a fixed number of processors, then increasing the number of processors, reducing execution times, or weakening precedence constraints can increase the *schedule length*.

A Priority-based 3 processor schedule

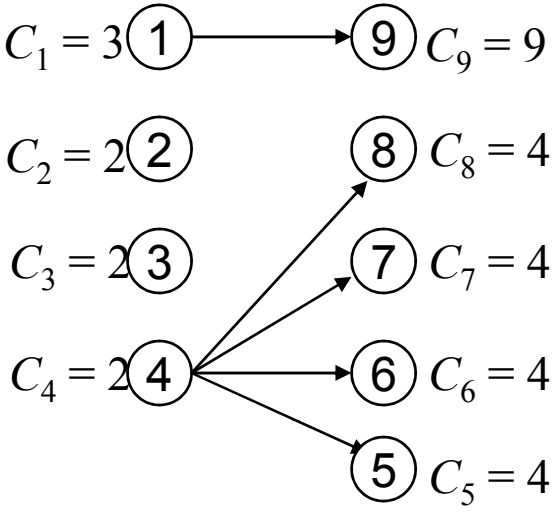


9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Priority-based 3 processor schedule:

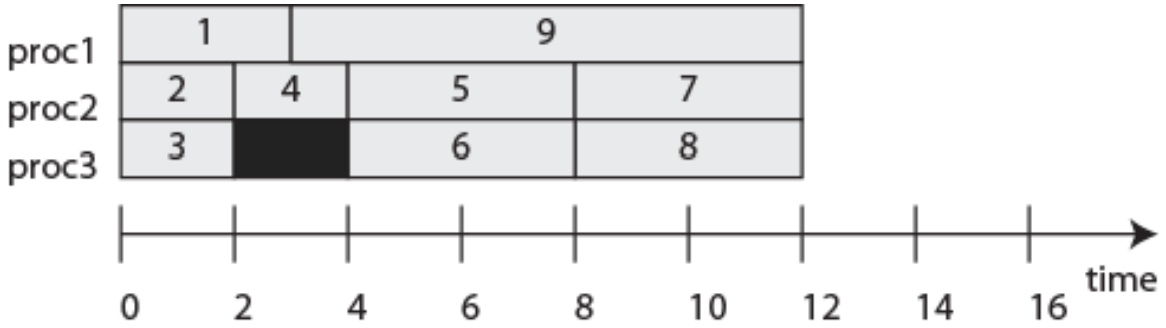


- Priority-based scheduling is "greedy"
- What happens if you increase the number of processors to four?

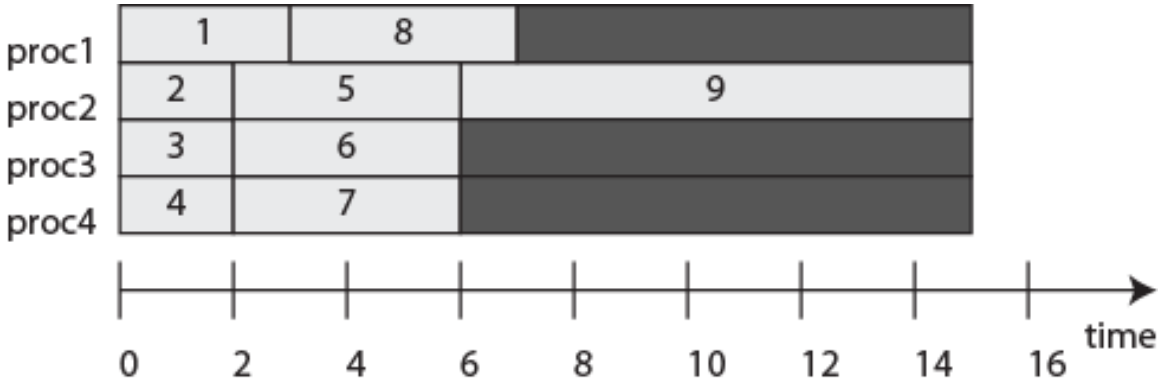
Richard's Anomalies: Increasing the number of processors



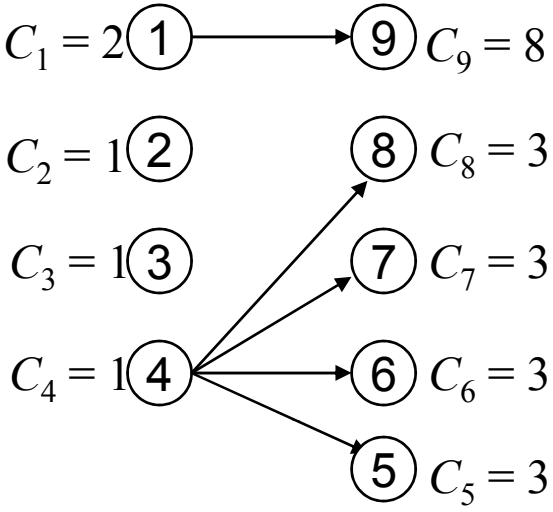
9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Priority-based 3 processor schedule:



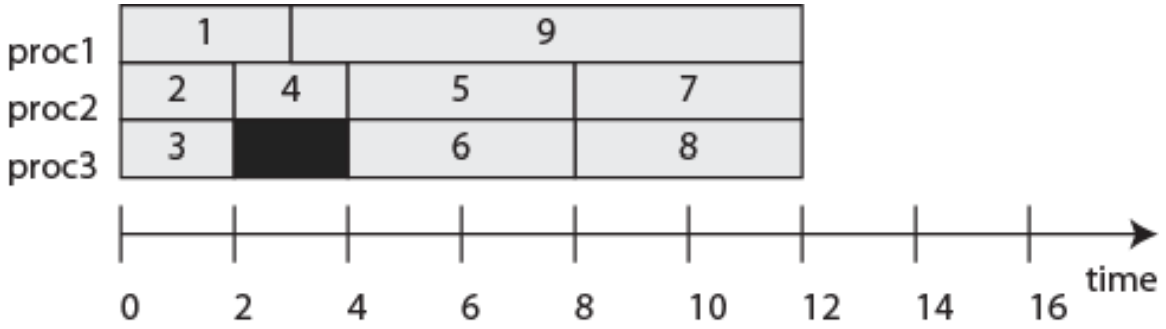
•The priority-based schedule with four processors has a longer execution time.



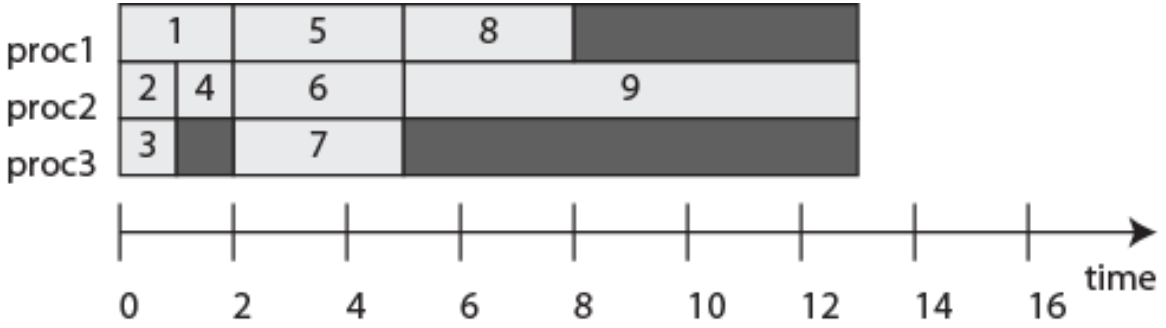
Richard's Anomalies: Reducing computation times by 1



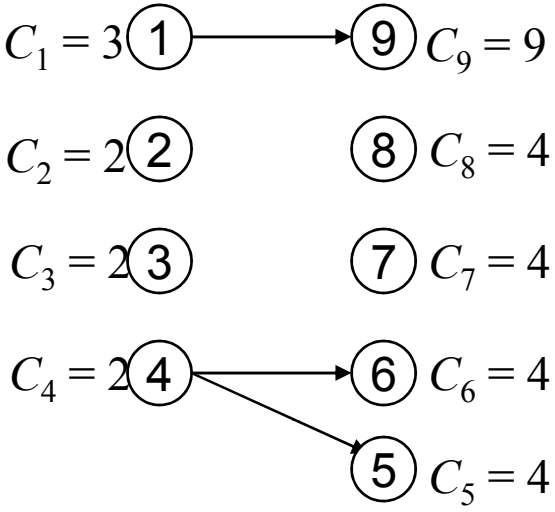
9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Priority-based 3 processor schedule:



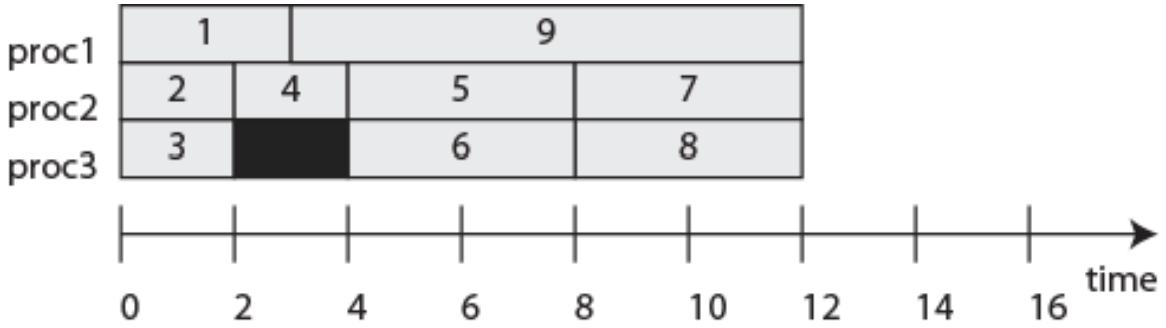
•Reducing the computation times by 1 also results in a longer execution time.



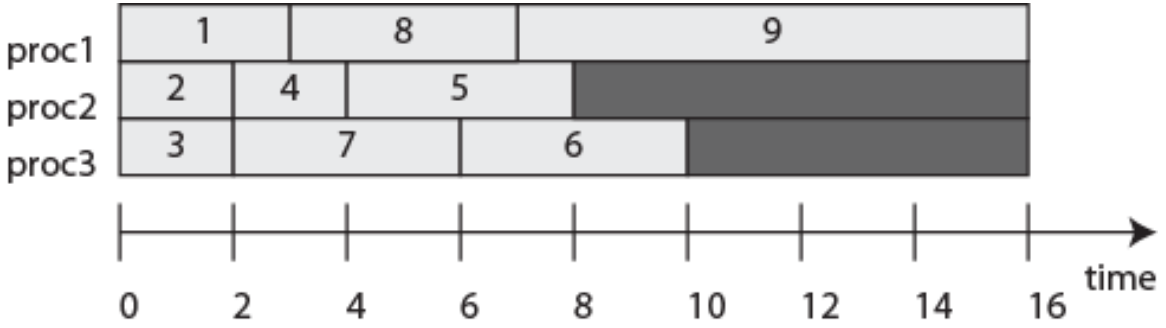
Richard's Anomalies: Weakening the precedence constraints



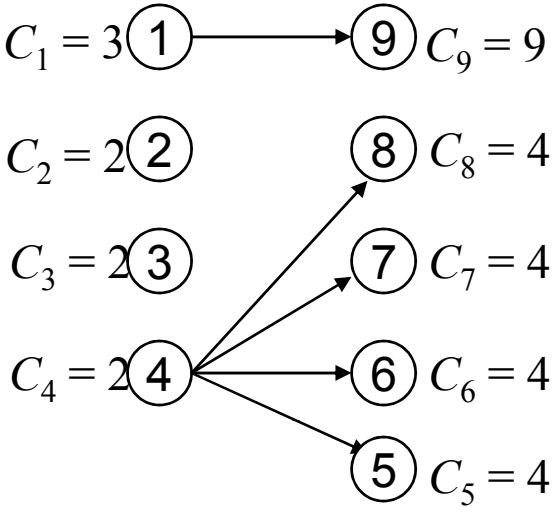
9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Priority-based 3 processor schedule:



•Remove the precedence constraints (4,8) and (4,7)

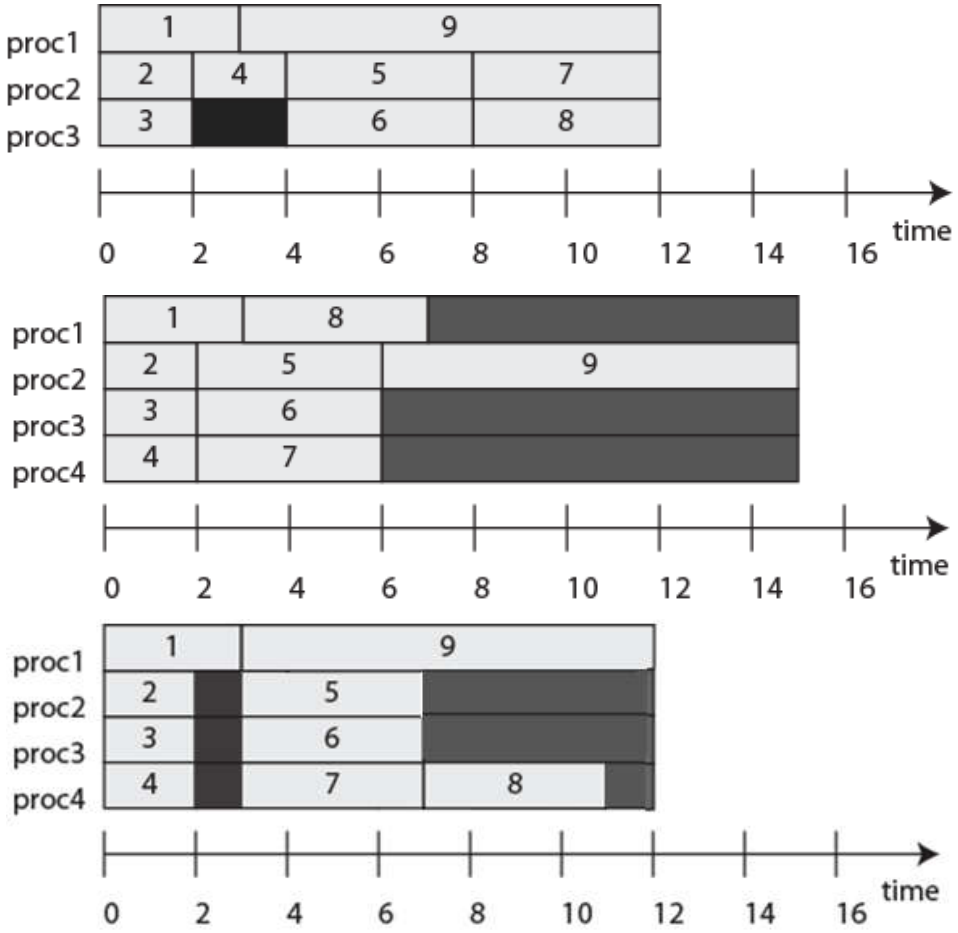


non-greedy，non-priority

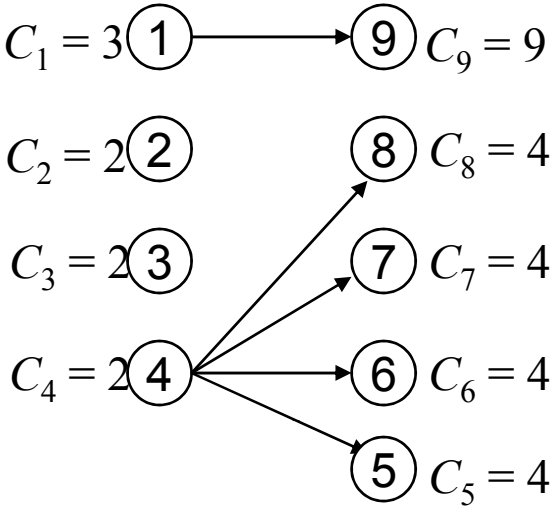


•A smarter scheduler for this example could hold off scheduling 5, 6, or 7, leaving a processor idle for one time unit.

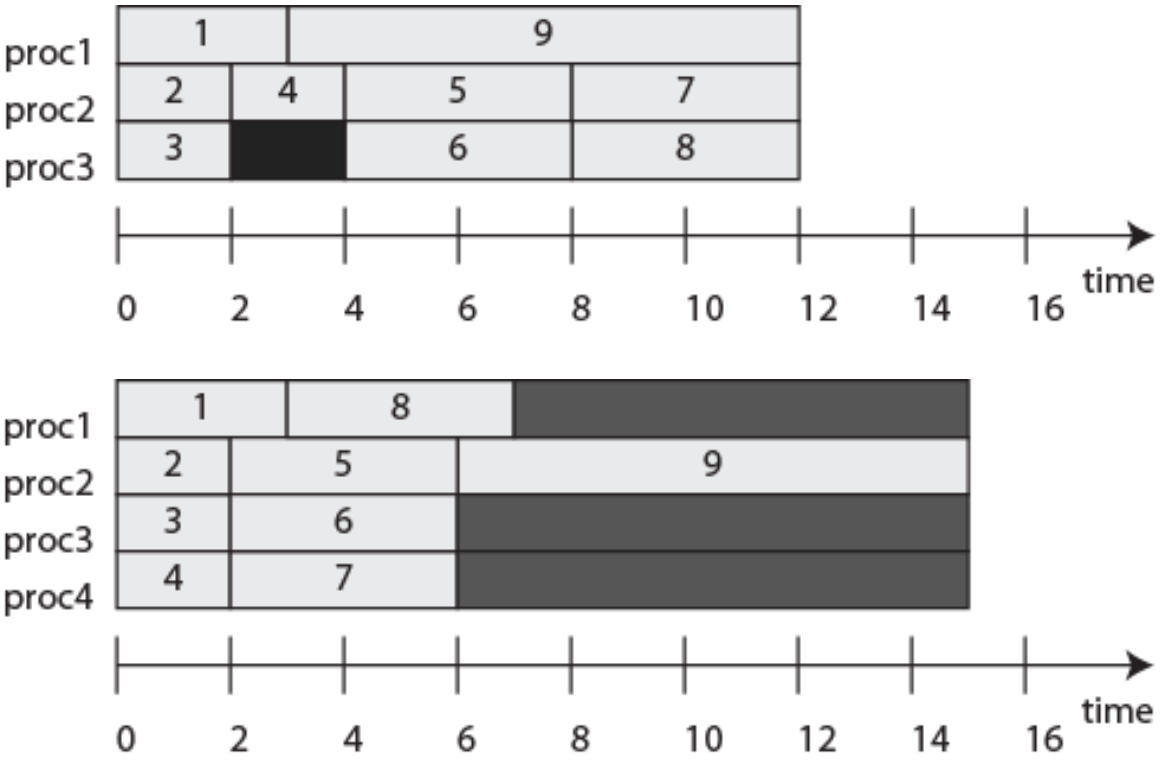
9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Priority-based 3 processor schedule:



Greedy scheduling may be the only practical option.



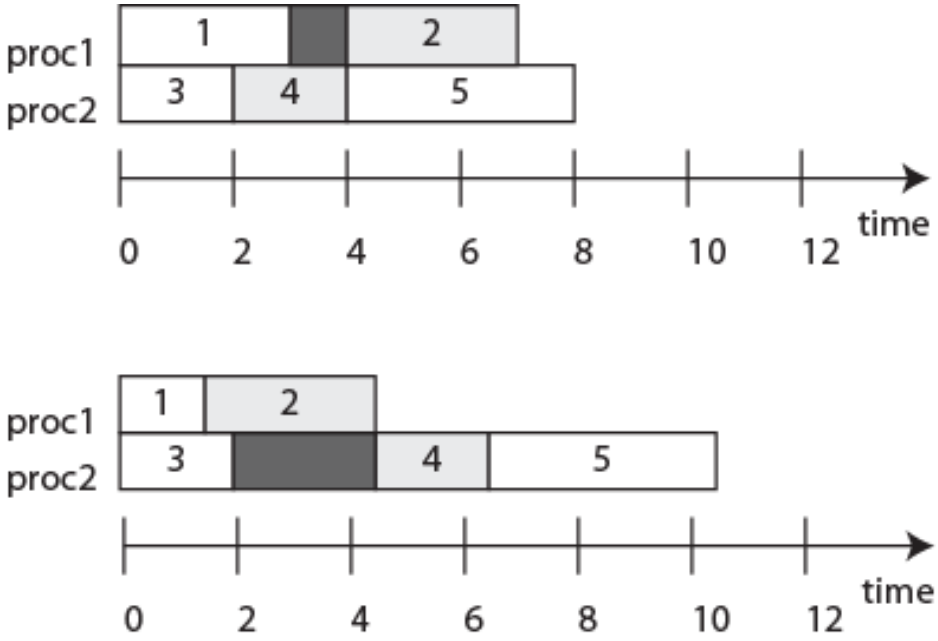
9 tasks with precedences and the shown execution times, where lower numbered tasks have higher priority than higher numbered tasks. Priority-based 3 processor schedule:



•如果任务只能在其前趋完成后才arrive(become known to the scheduler)，则只有贪心调度才是实际可行的。

Richard's Anomalies with Mutexes: Reducing Execution Time

•Assume tasks 2 and 4 share the same resource in exclusive mode, and tasks are statically allocated to processors. Then if the execution time of task 1 is reduced, the schedule length increases。

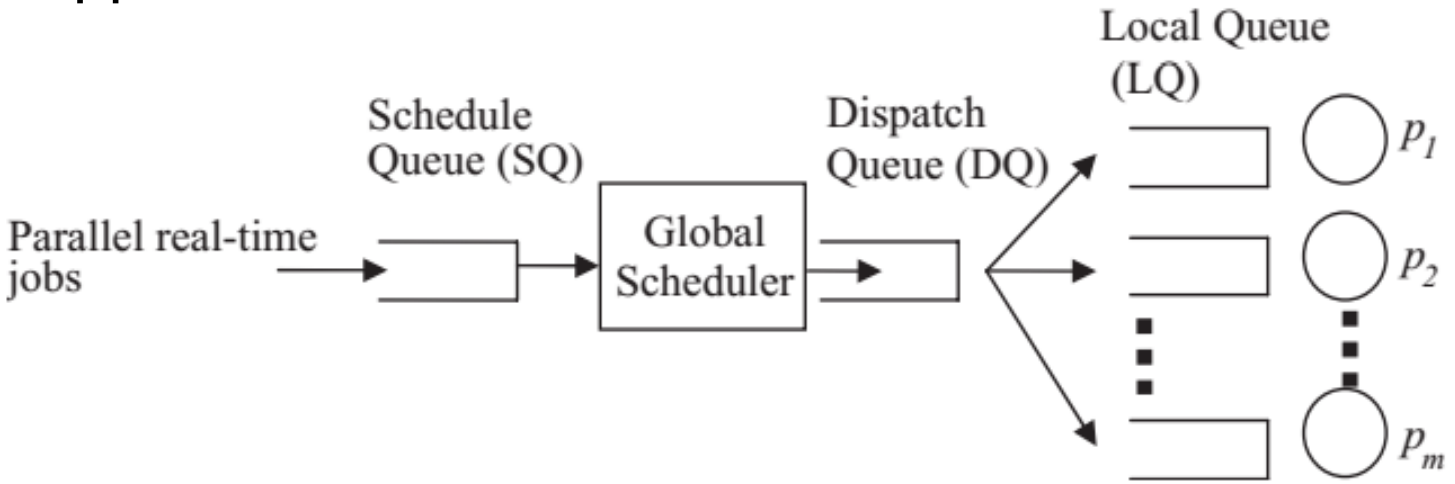


Pfair scheduling，Baruah1993

- Proportionate Fair：各处理器的利用率相同
  - “每个任务的执行速率与任务量相一致”
  - Given *n* processors and a periodic task set with utilization *n*
  - split task capacity in sub-tasks with a capacity of one unit of time
  - During each time slot a given processor runs only one task
- Task level migrations.
- Optimal scheduler with identical processors and synchronous periodic tasks with deadline on request.
- Lead to many context switches.

Scheduling in Multi-core Systems

- Multi-core chips：两类，Partitioned比Global优
  - Partitioned approaches（两步顺序可能颠倒）
    - Where：启发式方法（表调度list-scheduling算法，遗传算法）
    - When：RM，EDF
  - Global approaches

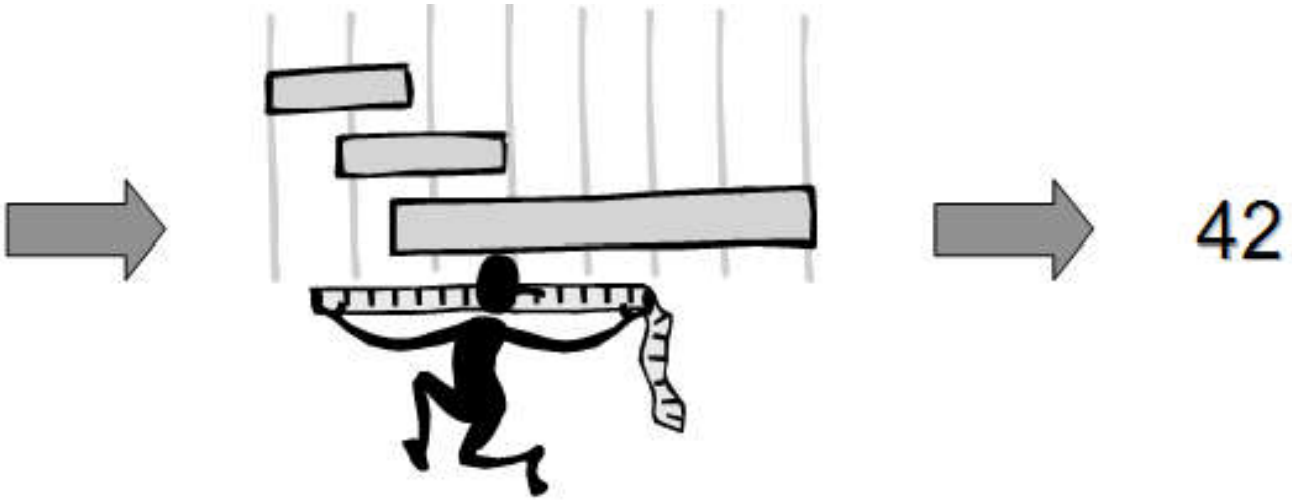


- Heterogeneous Multi-core Systems
  - 已证明asymmetric的性能和能耗优于symmetric
    - 如一个大核+多个小核：WCET不同

程序执行时间分析

- Execution Time of a Program
  - 是可调度性分析的关键输入

```
for I:=1 to N loop
begin
  if A > K
    then A:=K-1;
    else A:=K+1;
  if A < K
    then A:=K;
    else A:=K-1;
end;
```



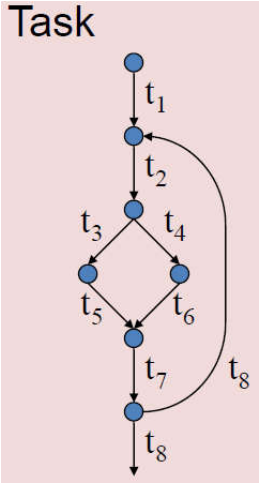
- 插桩法：

```
declare
  Old_Time, New_Time : Time;
  Interval : Duration;
begin
  Old_Time := Clock;
  -- other computations
  New_Time := Clock;
  Interval := New_Time - Old_Time;
end;
```

A simple (yet challenging) example

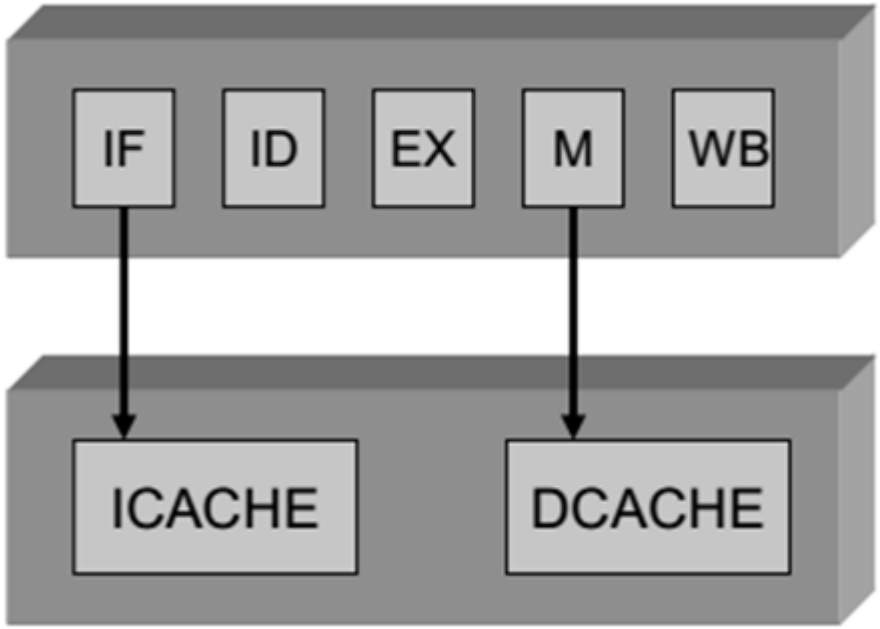
- Issues to consider
  - Input data is unknown
    - Iteration bounds must be known to facilitate analysis
  - Path explosion
    - 4^N paths in this example
  - Exclusion of non-executable (false) paths
    - T1 + E2 is a false path in the example
- 取决于程序的结构、目标处理器、执行环境
  - 分支路径、循环控制参数、数据存储位置
  - ILP、Cache、中断

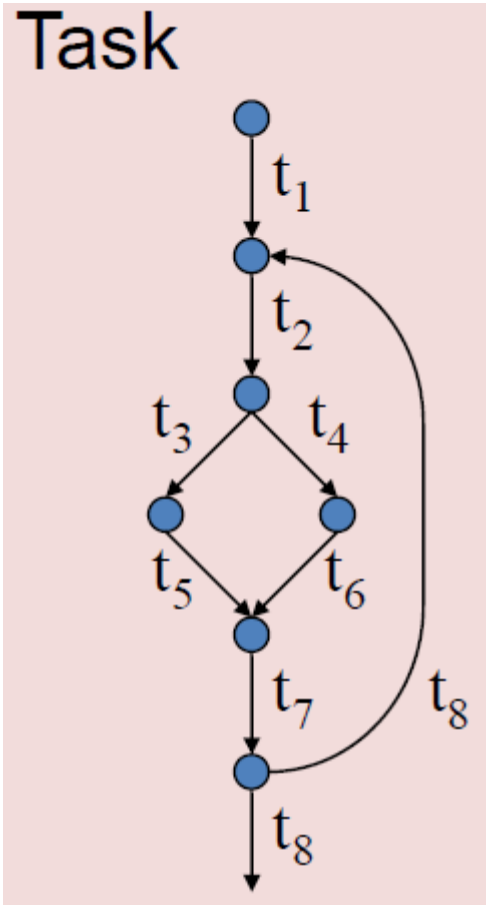
```
for I:=1 to N loop
begin
   if A > K
      then A:=K-1;  (T1)
      else A:=K+1;  (E1)
   if A < K
      then A:=K;    (T2)
      else A:=K-1;  (E2)
end;
```



Processor with pipeline

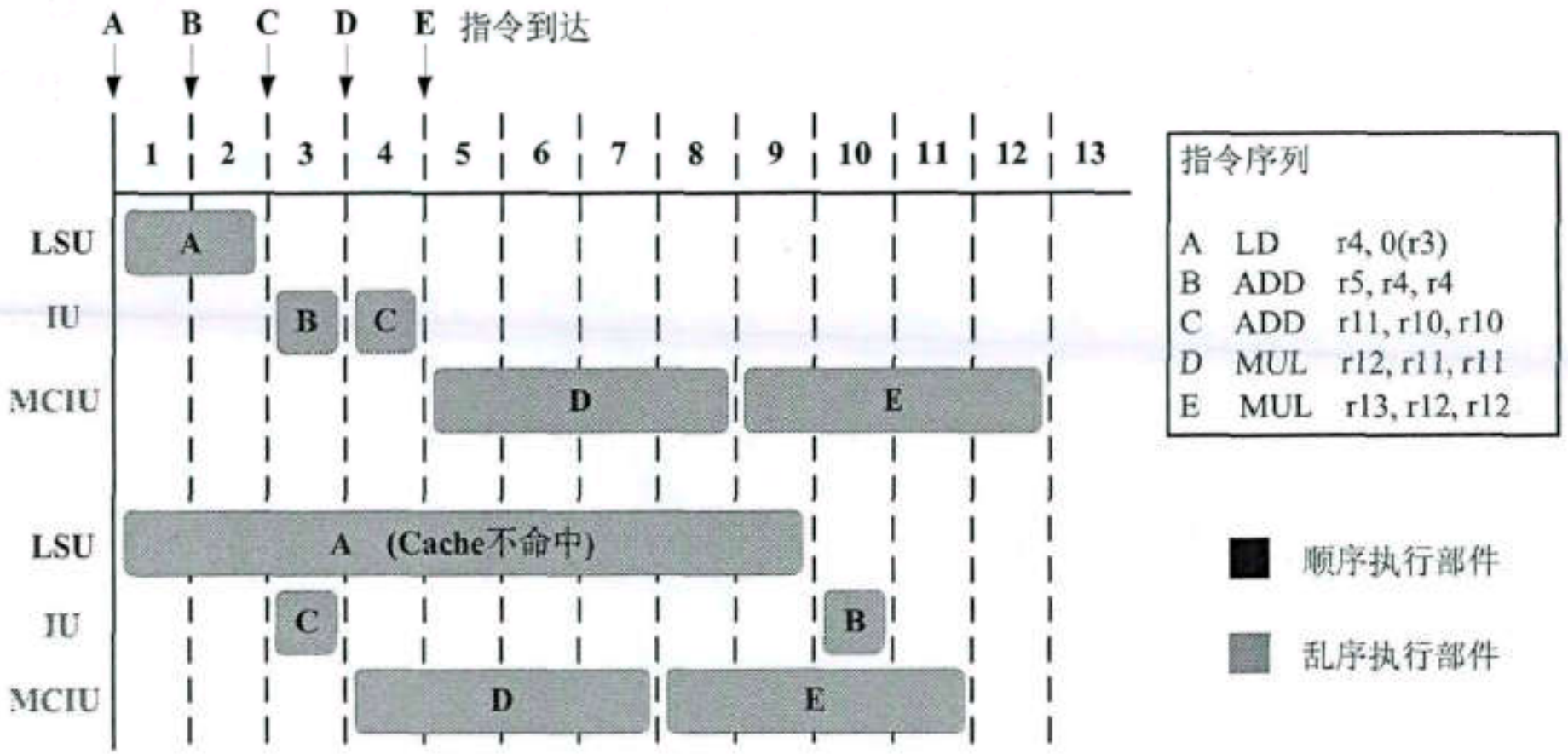
- Sources of time variations:
  - structural conflicts
  - data conflicts
  - branch conflicts





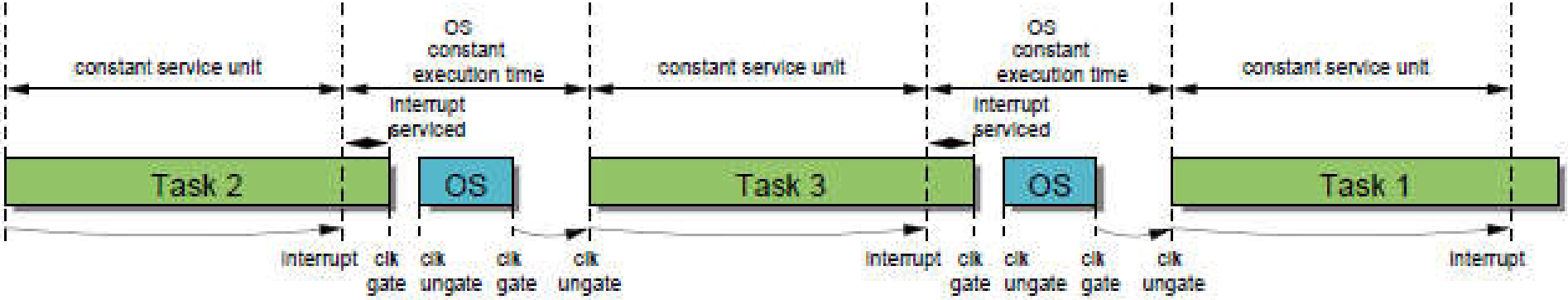
处理器行为分析的“时序异常”timing anomaly

- Cache不命中却使整体执行时间缩短



RTOS的影响

- The operating system is the part most responsible for a predictable behavior.
- Concurrency control must be enforced by:
  - appropriate scheduling algorithms
  - appropriate syncronization protocols
  - efficient communication mechanisms
  - predictable interrupt handling



Impact of interference on shared resources

websearch

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
LLC (small)	134%	103%	96%	96%	109%	102%	100%	96%	96%	104%	99%	100%	101%	100%	104%	103%	104%	103%	99%
LLC (med)	152%	106%	99%	99%	116%	111%	109%	103%	105%	116%	109%	108%	107%	110%	123%	125%	114%	111%	101%
LLC (big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	264%	222%	123%	102%
DRAM	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	270%	228%	122%	103%
HyperThread	81%	109%	106%	106%	104%	113%	106%	114%	113%	105%	114%	117%	118%	119%	122%	136%	>300%	>300%	>300%
CPU power	190%	124%	110%	107%	134%	115%	106%	108%	102%	114%	107%	105%	104%	101%	105%	100%	98%	99%	97%
Network	35%	35%	36%	36%	36%	36%	36%	37%	37%	38%	39%	41%	44%	48%	51%	55%	58%	64%	95%
brain	158%	165%	157%	173%	160%	168%	180%	230%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%

ml_cluster

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
LLC (small)	101%	88%	99%	84%	91%	110%	96%	93%	100%	216%	117%	106%	119%	105%	182%	206%	109%	202%	203%
LLC (med)	98%	88%	102%	91%	112%	115%	105%	104%	111%	>300%	282%	212%	237%	220%	220%	212%	215%	205%	201%
LLC (big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	276%	250%	223%	214%	206%
DRAM	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	287%	230%	223%	211%
HyperThread	113%	109%	110%	111%	104%	100%	97%	107%	111%	112%	114%	114%	114%	119%	121%	130%	259%	262%	262%
CPU power	112%	101%	97%	89%	91%	86%	89%	90%	89%	92%	91%	90%	89%	89%	90%	92%	94%	97%	106%
Network	57%	56%	58%	60%	58%	58%	58%	58%	59%	59%	59%	59%	59%	63%	63%	67%	76%	89%	113%
brain	151%	149%	174%	189%	193%	202%	209%	217%	225%	239%	>300%	>300%	279%	>300%	>300%	>300%	>300%	>300%	>300%

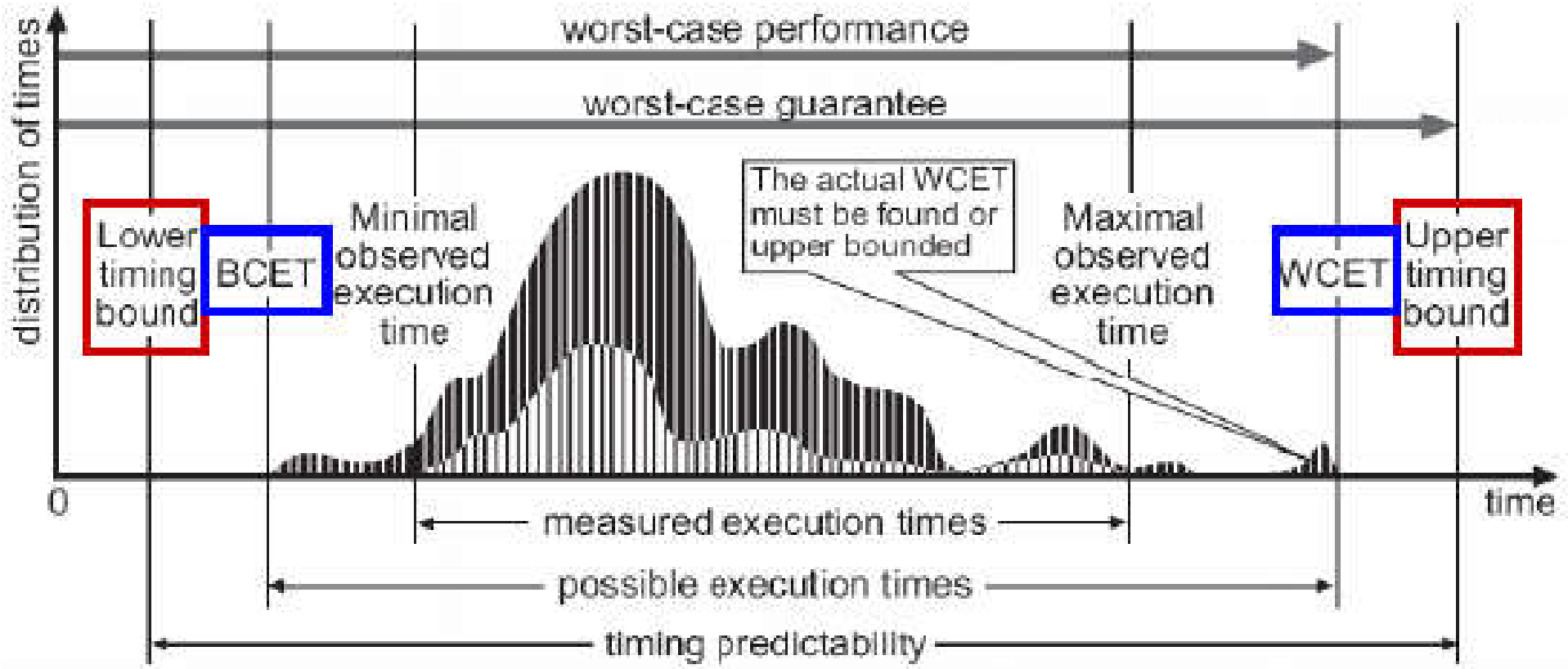
memkeyval

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
LLC (small)	115%	88%	88%	91%	99%	101%	79%	91%	97%	101%	135%	138%	148%	140%	134%	150%	114%	78%	70%
LLC (med)	209%	148%	159%	107%	207%	119%	96%	108%	117%	138%	170%	230%	182%	181%	167%	162%	144%	100%	104%
LLC (big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	280%	225%	222%	170%	79%	85%
DRAM	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	252%	234%	199%	103%	100%
HyperThread	26%	31%	32%	32%	32%	32%	33%	35%	39%	43%	48%	51%	56%	62%	81%	119%	116%	153%	>300%
CPU power	192%	277%	237%	294%	>300%	>300%	219%	>300%	292%	224%	>300%	252%	227%	193%	163%	167%	122%	82%	123%
Network	27%	28%	28%	29%	29%	27%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%
brain	197%	232%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%

Each entry is color-coded as follows: 140% is ≥120%, 110% is between 100% and 120%, and 65% is ≤100%.

Heracles: Improving Resource Efficiency at Scale, ISCA2015

WCET & BCET

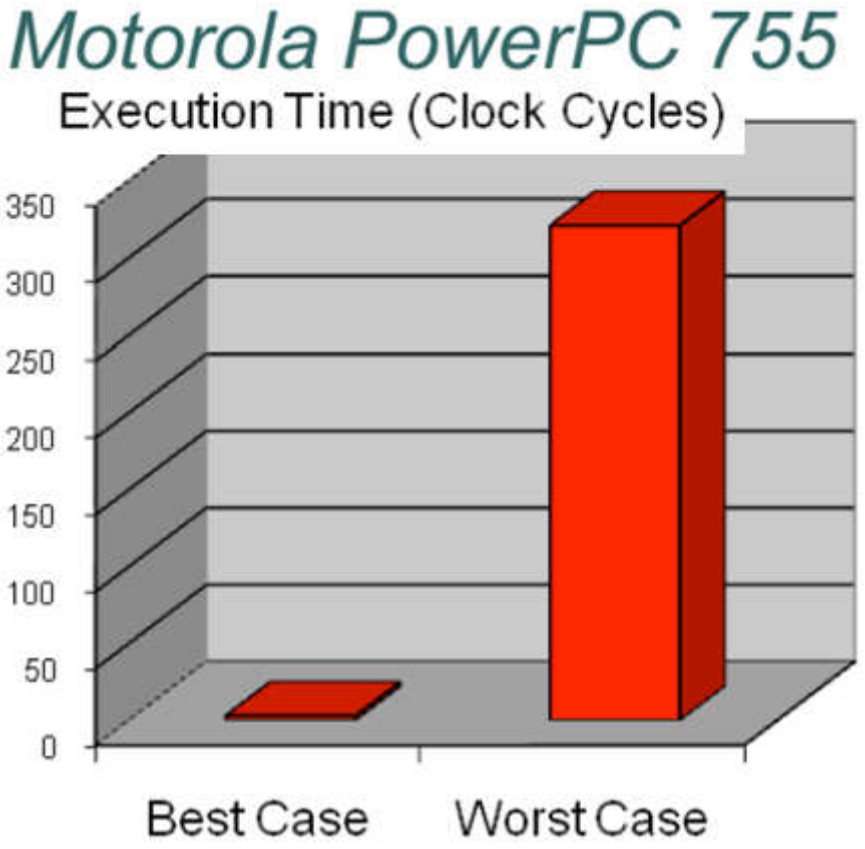


- Underestimating：catastrophic disasters
- Overestimating：oversizing of the running hardware
- 工业界的设计往往预留30%余量

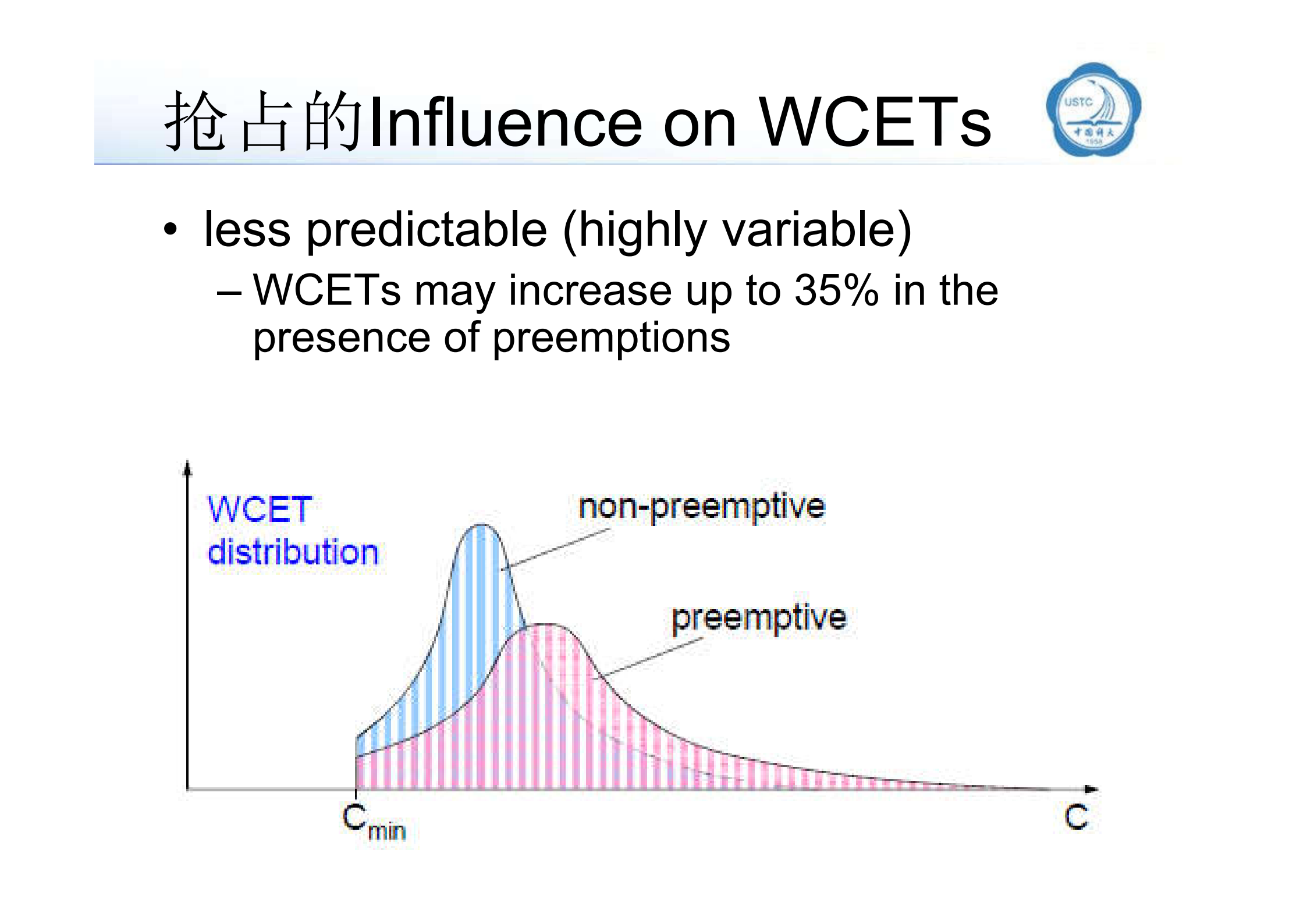
WCET & BCET

x=a+b; →

```
LOAD   r2, _a
LOAD   r1, _b
ADD    r3,r2,r1
```



抢占的Influence on WCETs



- less predictable (highly variable)
  - WCETs may increase up to 35% in the presence of preemptions

Timing Analysis（WCET分析）

- 定义：计算给定应用程序代码片断执行时间的上限
  - 给定：程序代码，运行平台（OS＋HW）
  - 求：任务的WCET
    - 有解？No，由于不可预测性
    - 结果要求：安全性(safety，紧致性(tightness)，精确性(precision)
- 方法
  - 动态时间分析：很难保证所得到结果是安全的
    - 当前工业界采用测量的方法
    - 随机方法、基于遗传算法的进化方法(EvolutionMethod)、模拟退火方法和统计方法。
  - 静态分析：保证分析结果的安全性

Simple-Task model

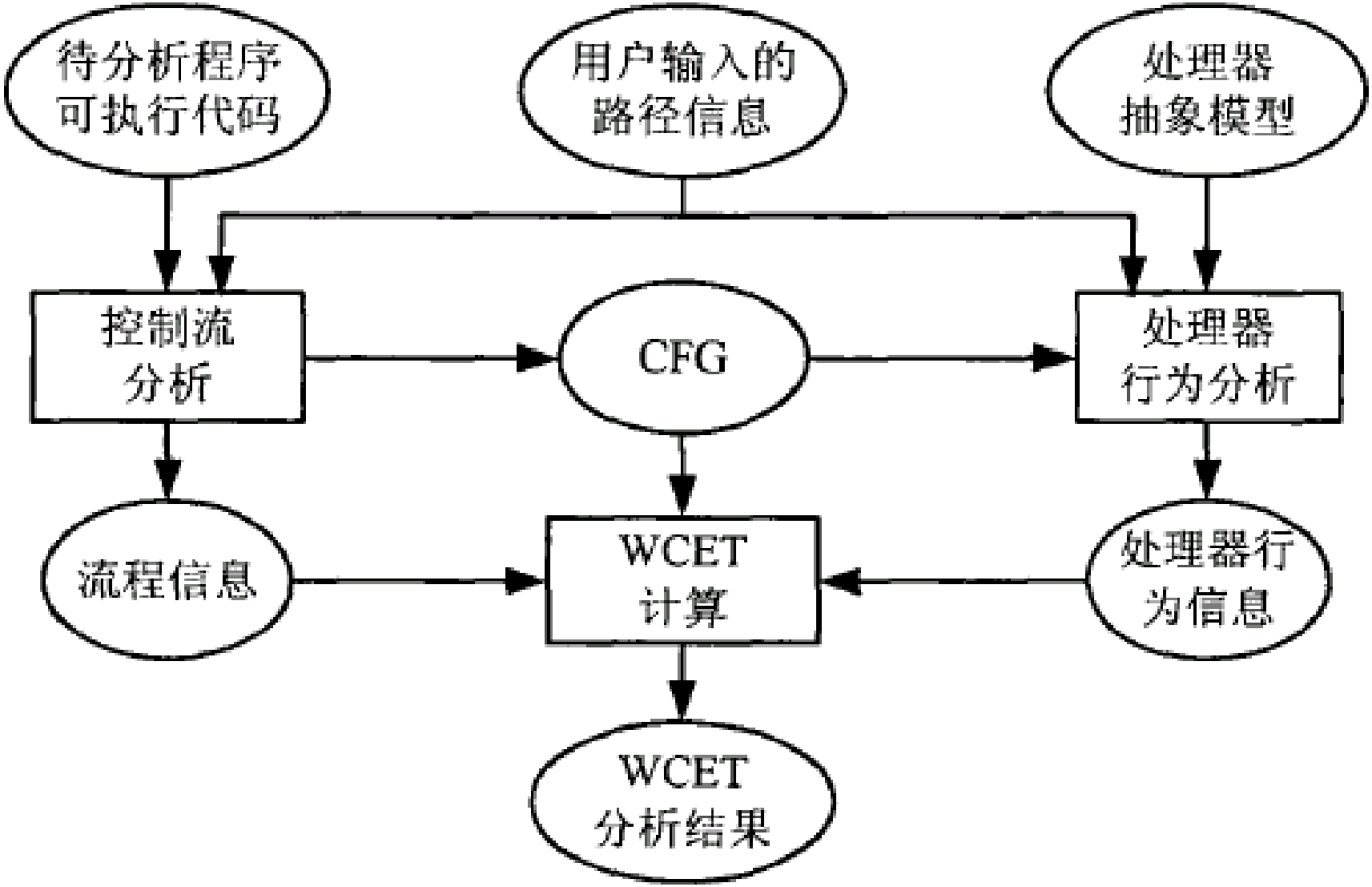
- have a plain input – processing – output structure
  - Inputs available at start
  - Outputs ready at the end
  - No blocking inside
  - No synchronization or communication inside
- Execution time variations only due to differences in
  - inputs
  - task state at start time (no external disturbances)

```
while(1) {
    read_sensors();
    compute();
    write_to_actuators();
}
```

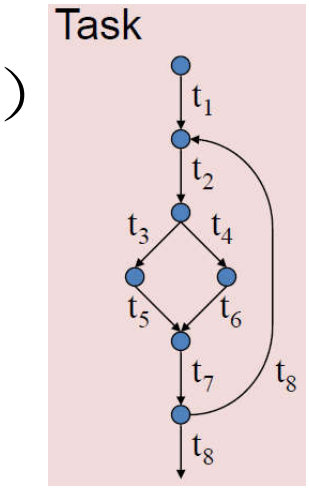
This code typically has:
- loops with finite bounds
- no recursion

Additional assumptions:
- runs uninterrupted
- single-threaded

静态WCET分析方法

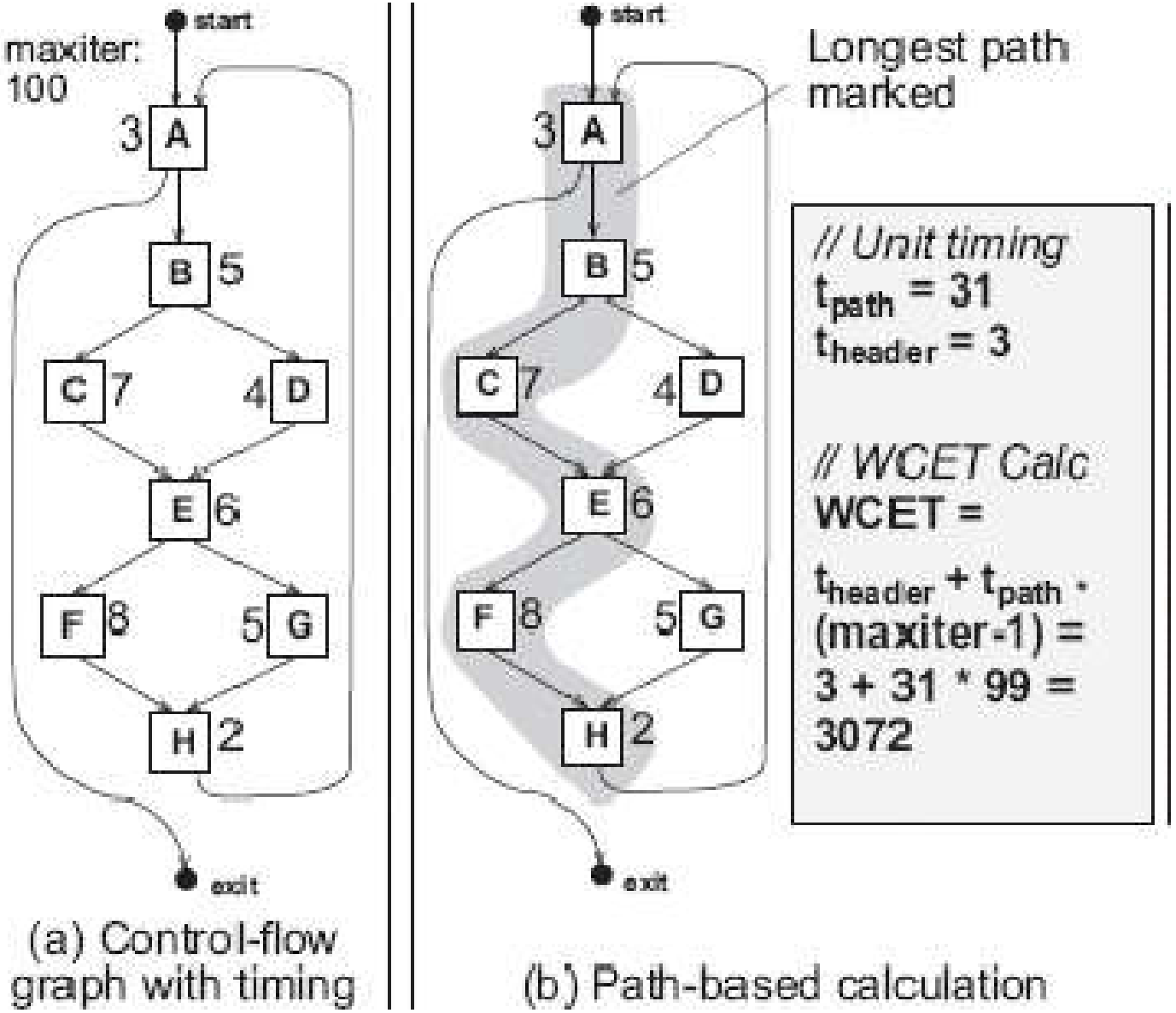

待分析程序
可执行代码
用户输入的
路径信息
处理器
抽象模型
控制流
分析
CFG
处理器
行为分析
流程信息
WCET
计算
处理器行
为信息
WCET
分析结果

静态WCET分析方法：含三步



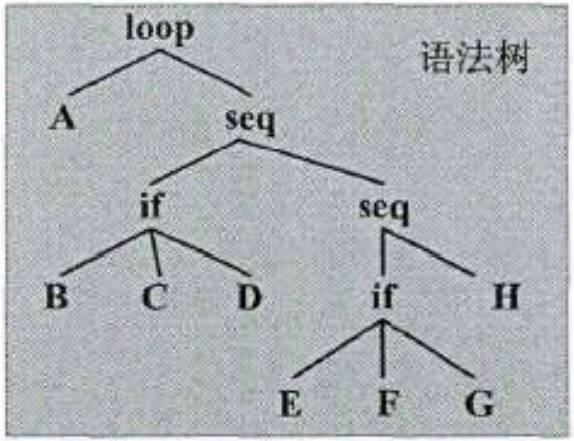
- 程序流分析：路径分析（CFG，call flow graph）
  - 最长路径：所有可能的路径，循环的界（bound）
    - 分支、循环以及函数调用
- 处理器行为分析：也称low level分析
  - 建立执行时间模型
    - 对非ILP处理器：每条指令执行时间的累加
    - 考虑cache、流水线stall、分支预测等
- WCET计算
  - 基于路径(Path-based)：不能处理三角形的循环结构
  - 基于树(Tree-based，又称Timing Schema) ：自底向上遍历语法树进行，有时会过于悲观
  - 隐式路径枚举技术(IPET-based) ：将程序路径和各基本模块的执行时间用代数和(或)逻辑的限制条件表示
    - 描述复杂程序控制流信息的能力很有限
    - 进而限制了处理器行为分析所能采用的技术

基于路径：带时间约束的CFG分析



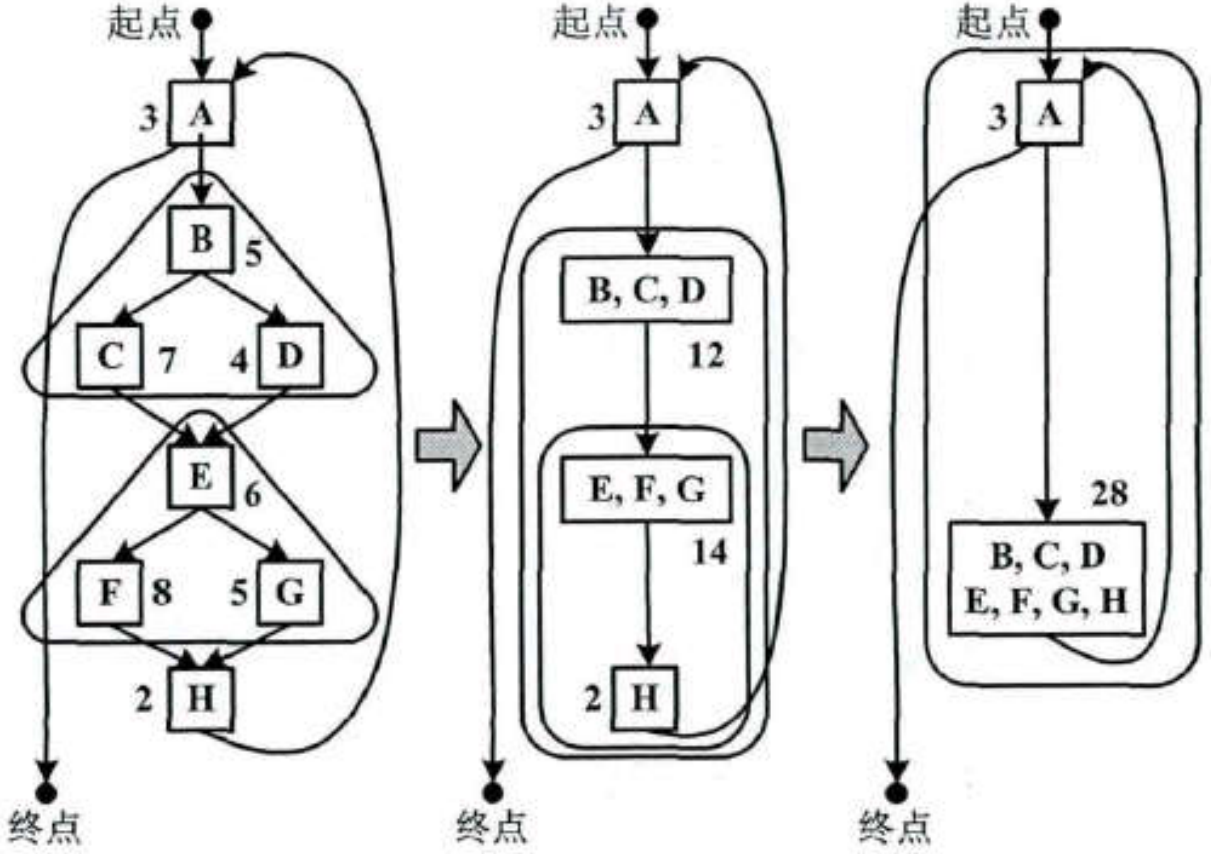
基于树(Tree-based，又称Timing Schema)

- 自底向上对程序的语法分析树进行遍历



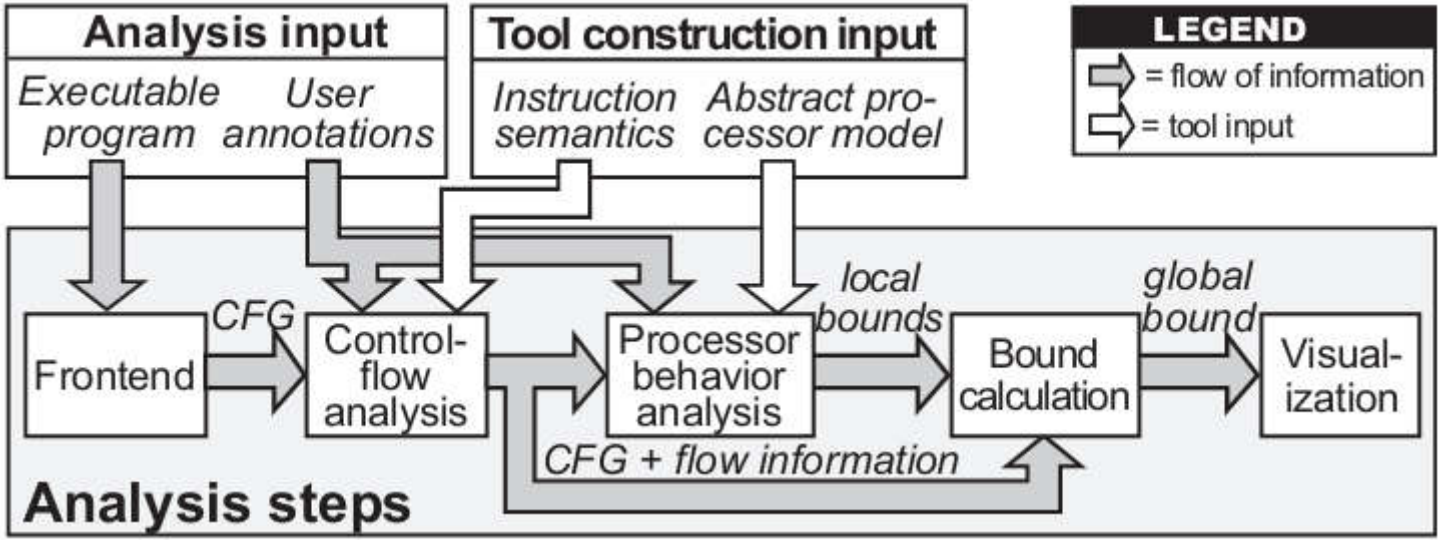
规约法则

T(seq(S1,S2)) = T(S1)+T(S2)
T(if(Exp) S1 else S2) =
T(Exp)+max(T(S1),T(s2))
T(loop(Exp, Body)) =
T(Exp) +
(T(Exp)+T(Body))*(maxiter-1)



WCET分析工具

- 商业：aiT, RapiTime, …
- 学院：SWEET(Malardalen Univ), Chronos(National Univ of Singapore), OTAWA(Toulouse) …
  - is an open-source static WCET analysis tool.
  - The input to Chronos is a task written in C and the configuration of the target processor.



Safety standards

- WCET and response time must be checked and verified!
  - aerospace systems：DO178C, WG 48-1999
  - electrical, electronic systems：IEC-61508
  - automotive systems：ISO-26262
  - railway systems：CENELEC EN-50128
  - medical app：EN-60601 and IEC-62304
  - nuclear plants：IEC-60880

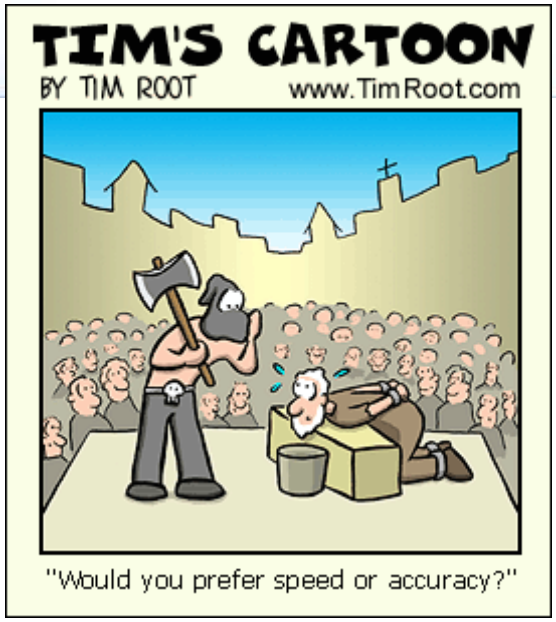
ARINC Standard WG 48-1999

- *The Avionics Computing Resource (ACR) shall include internal hardware and software management methods as necessary to ensure that time, space and I/O allocations are deterministic and static.*
  - *"Deterministic and static" means in this context, that time, space and I/O allocations are determined at compilation, assembly or link time, remain identical at each and every initialization of a program or process, and are not dynamically altered during runtime.*

作业

- 任务的执行时间（WCET）分析工具调研
  - 典型工具有哪些？
  - 试用一种，测试冒泡排序的WCET

- 多核处理器上任务调度模型有哪些？归纳各自的特点。


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


Thank You