

**BRNO UNIVERSITY OF TECHNOLOGY**

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

Kolejní 4, 616 00 Brno

tel.: +420 5 4114 1113 fax: +420 5 4114 1123

E-mail: [kucera@feec.vutbr.cz](mailto:kucera@feec.vutbr.cz), <http://taceo.eu>



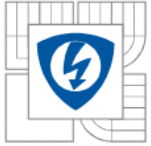
# Operační Systémy Reálného Času

III.

Pavel Kučera

CAK, E112

[kucera@feec.vutbr.cz](mailto:kucera@feec.vutbr.cz)

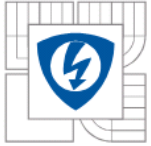


RTOS - III



# Obsah

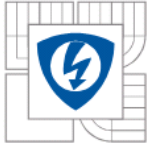
1. Víceprocesorové architektury
2. Plánovací algoritmy
3. Souborový systém v RTX



## Scheduling (plánování)

Víceúlohové OS vyžadují paralelní běh několika úloh. Paralelního běhu lze v zásadě docílit dvěma způsoby:

- na jednoprocesorovém systému pseudokonkurencí,
- na víceprocesorovém systému skutečným paralelním během (pravá konkurence). Víceprocesorové architektury se pak dle organizace CPU a paměti dělí na:
  - Symetric multiprocessing (**SMP**)
  - Asymetric multiprocessing (**ASMP**)
  - Non-Uniform Memory Access (**NUMA**)

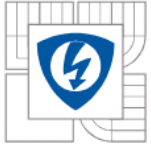


## Paralelizace úlohy

Důležitý a poměrně komplikovaný aspekt programování, zejména pak v RTOS.

V ideálním případě přidáním  $n$ -výpočetních jader k jedno-jádrovému systému povede k  $n$ -násobnému zvýšení výpočetního výkonu.

To v reálných úlohách často **neplatí**, neboť úlohy nelze efektivně paralelizovat bez ztráty výkonu způsobené mezi-jádrovou (procesorovou) komunikací a koordinací.



## Amdahlův zákon

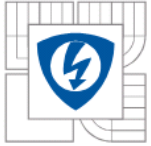
Zrychlení úlohy ve více-jádrovém systému je omezeno časem, který trvá vykonání téže úlohy sekvenčně.

Zrychlení úlohy **S** je definováno jako poměr mezi časem, který bude potřebovat jedno-jádrový systém versus n-jádrový systém.

Maximální zrychlení je pak Amdahlovým zákonem omezeno naší schopností úlohu paralelizovat. **p** je poměr paralelně běžících částí úlohy k celkovému počtu běžících částí. Výpočet celé úlohy bude v n-jádrovém systému trvat (za předpokladu je že jedno jádro zvládne úlohu za normalizovanou jednotku času):

$$1 - p + \frac{p}{n}$$

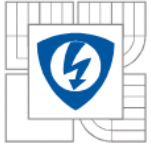
$\underbrace{\hspace{1.5cm}}_{\text{sekvenční část}} \quad \underbrace{\hspace{1.5cm}}_{\text{paralelní část}}$



## Amdahlův zákon

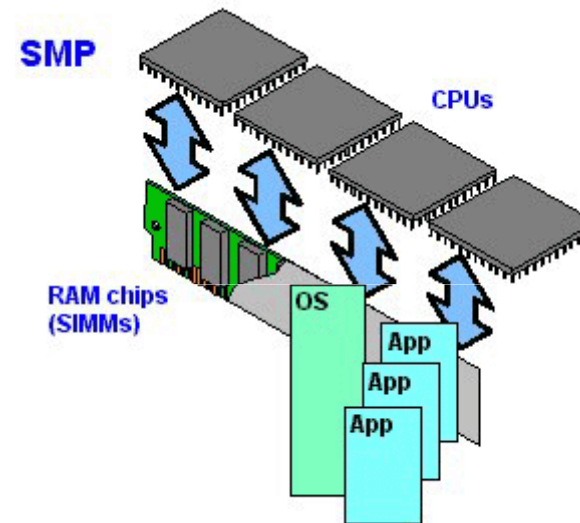
$$S = \frac{1}{1 - p + \frac{p}{n}}$$

Pokud na 10-ti jádrovém systému zvládneme paralelizovat 90 % úlohy a zbývajících 10 % bude prováděno sekvenčně, pak bude výsledné zrychlení přibližně pětinasobné, nikoliv však desetinásobné. Těchto 10 %, které jsme nezvládli paralelizovat sníží výpočetní výkon (vytíženost) systému 2x. Z toho pramení, že má smysl investovat námahu do paralelizace zbývajících 10 % úlohy, protože s každým procentem, které paralelizujeme se výsledné zrychlení zvětší o 0.3.



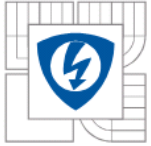
# Architektura SMP

- nutná podpora v OS, popřípadě v compileru
- zrychlují běh i procesů navržených pro uniproceorové platformy
- rychlá výměna dat mezi procesy
- pomalejší přístup do společné paměti



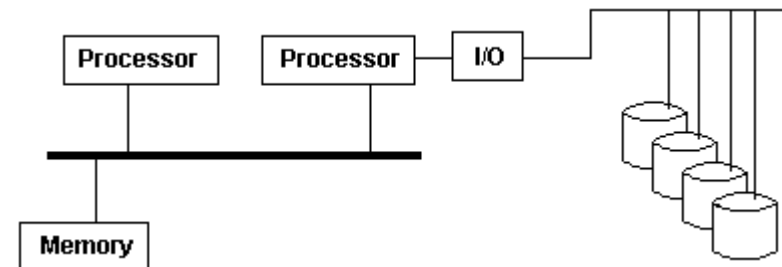
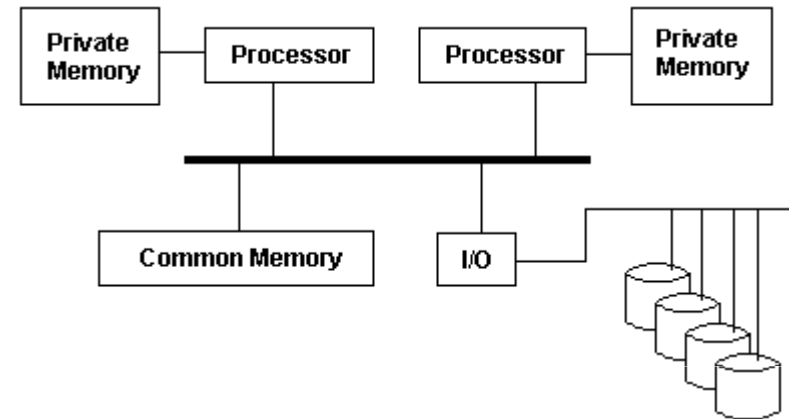
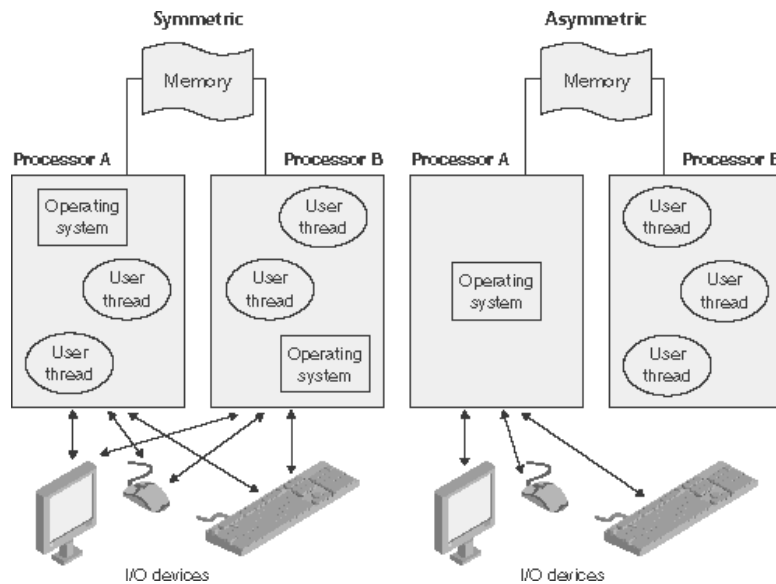
Zdroj: [www.zdnet.co.kr](http://www.zdnet.co.kr)

QNX  
Mac OS  
OS/2  
VxWorks  
Windows NT (2k, XP, 2003)  
Unix  
Linux  
Solaris

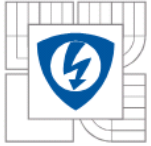


# Architektura ASMP

- řešení je často na uživateli, popřípadě je šité na míru
- pomalá výměna dat mezi procesy
- rychlý přístup do dedikované paměti



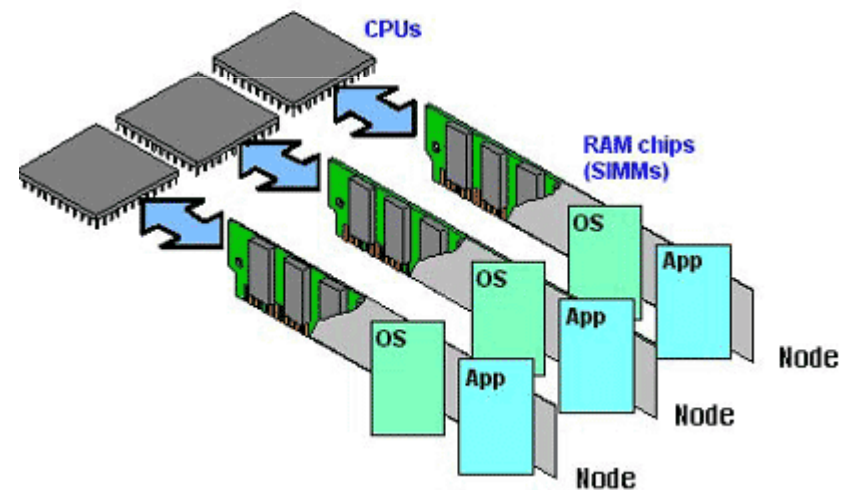
Zdroj: ohlandl.ipv7.net



# Architektura NUMA

- speciální (ale častý) případ ASMP
- extrémně rychlý přístup do přidělených pamětí (často i fyzicky)
- problémy se synchronizací a sdílenými daty – pomalý přenos dat mezi procesy

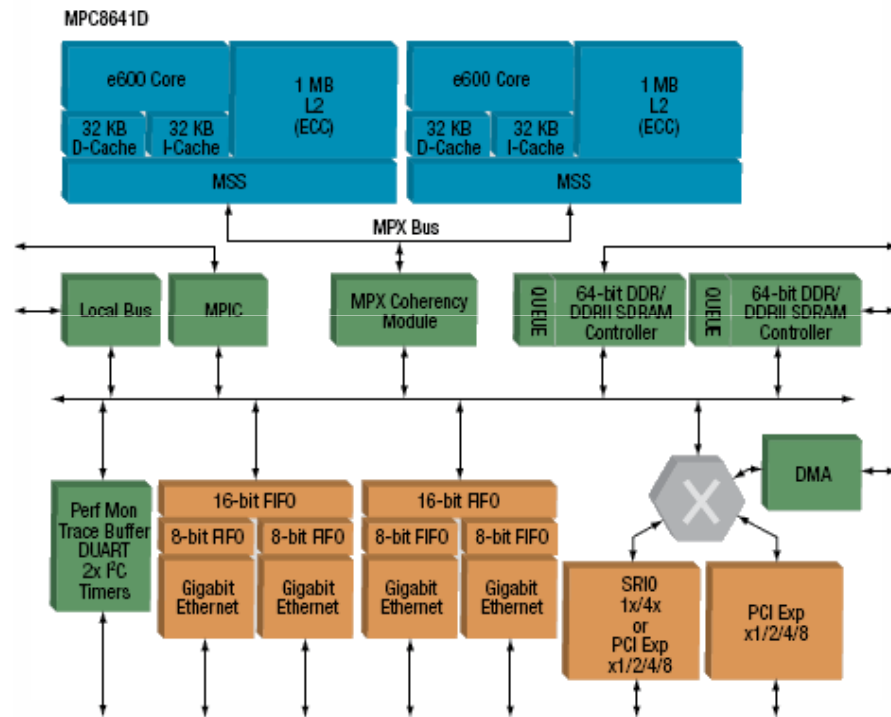
Intel Itanium  
AMD Opetron  
SGI  
DEC  
Alpha EV7  
NUMA for Linux  
NUMA for Windows  
OpenSolaris NUMA



Zdroj: [www.zdnet.co.kr](http://www.zdnet.co.kr)

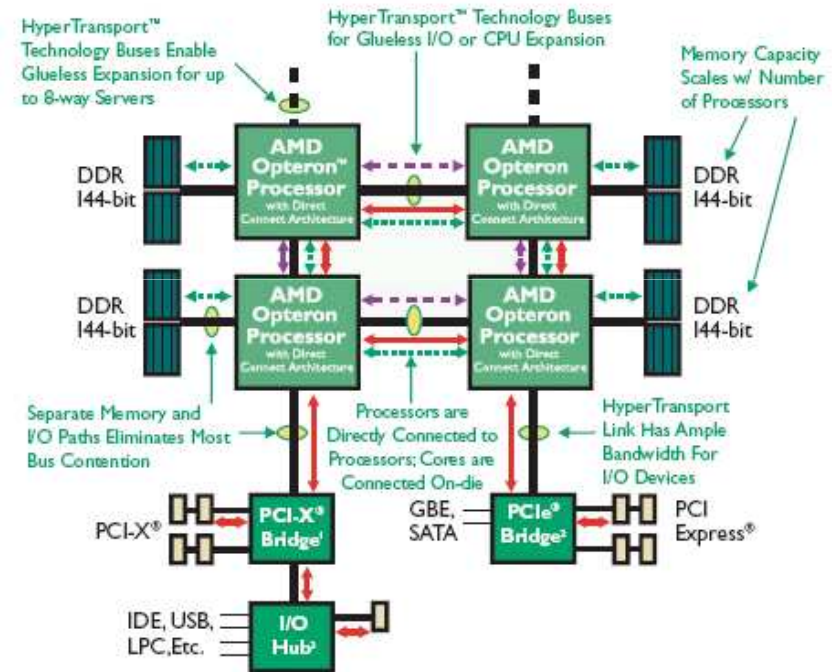


# SMP vs NUMA

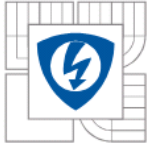


Zdroj: [www.rtc magazine.com](http://www.rtc magazine.com)

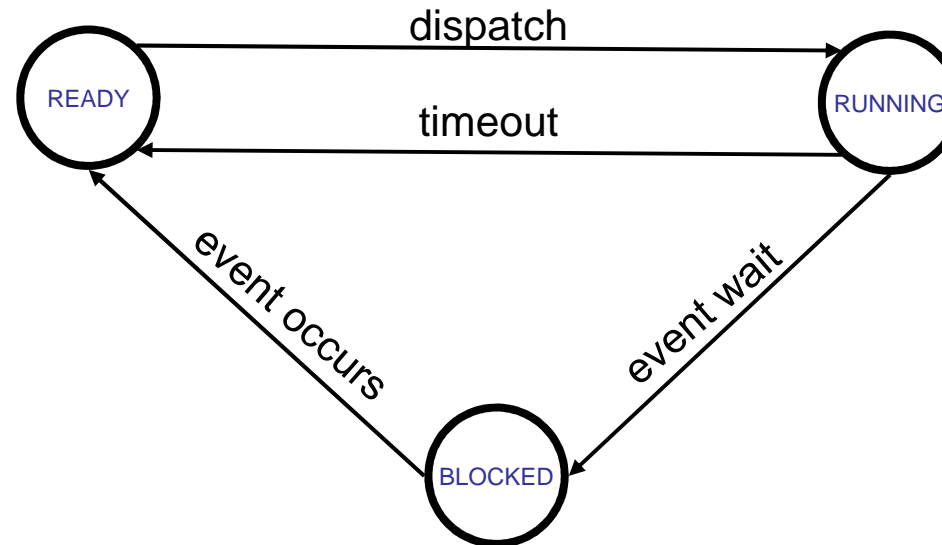
## AMD OPTERON



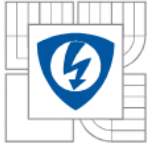
Zdroj: [developer.amd.com](http://developer.amd.com)



# Stavový automat úlohy

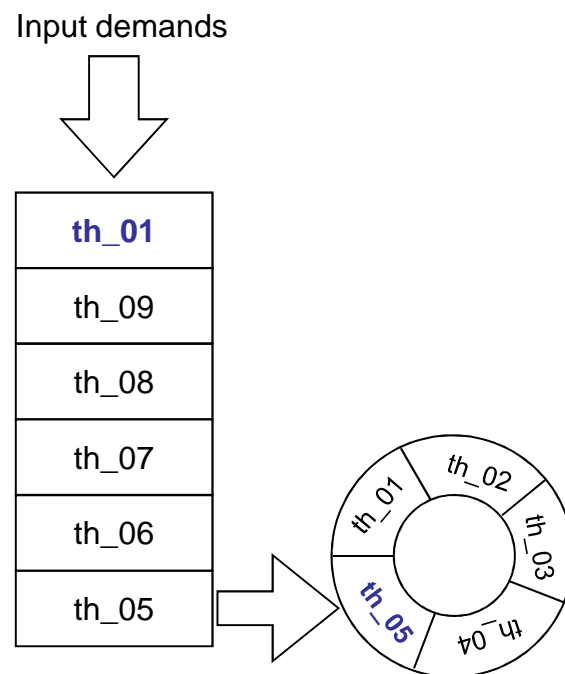


- READY**      úloha je připravena ke spuštění
- RUNNING**    úloha je vykonávána
- BLOCKED**    úloha se nevykonává, protože čeká na událost



# FIFO - FCFS

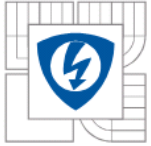
## FCFS First Come, First Served



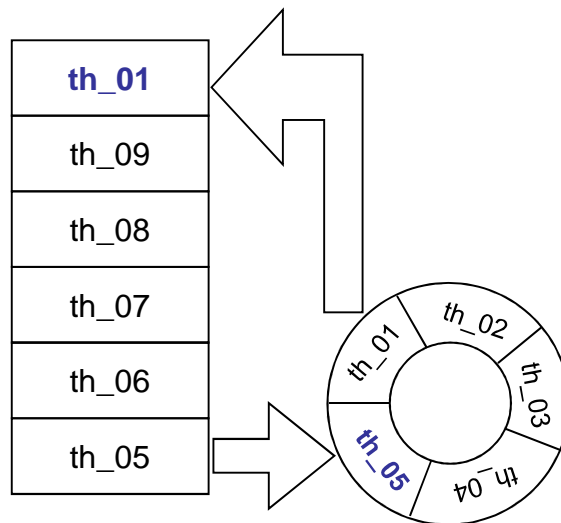
Obvykle implementováno jako nezávislá množina obsluh přerušení.

Klade nízké nebo vůbec žádné požadavky na plánovač jádra.

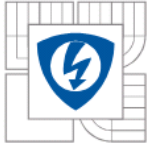
To však neúměrně stěžuje návrh systému.



# Cyclic executive



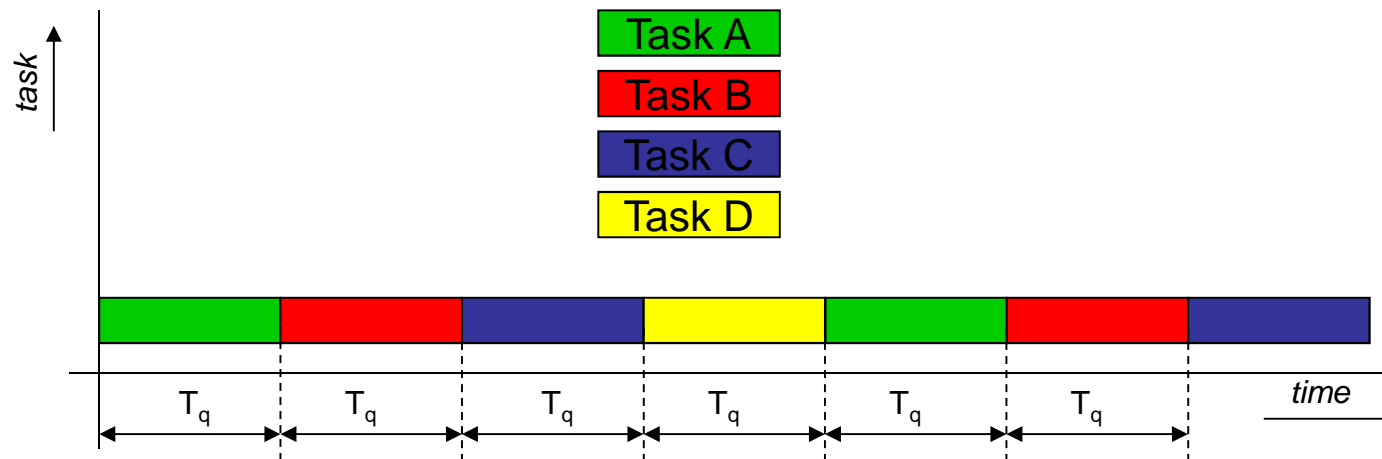
- velmi podobný přístupu “supersmyčka“
- jednoduchá implementace i ověření
- složité časování při modifikaci úlohy
- kvůli jednoduchosti používáno v safety-critical a fail-safe aplikacích

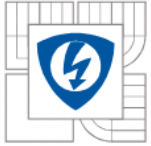


## Round Robin (time-slicing)

Všechny úlohy mají k CPU přístup po stejné časové kvantum  $T_q$ .

Nevýhodou je, že úloha s velkou prioritou, která potřebuje CPU se k němu dostane až za dlouhý interval.

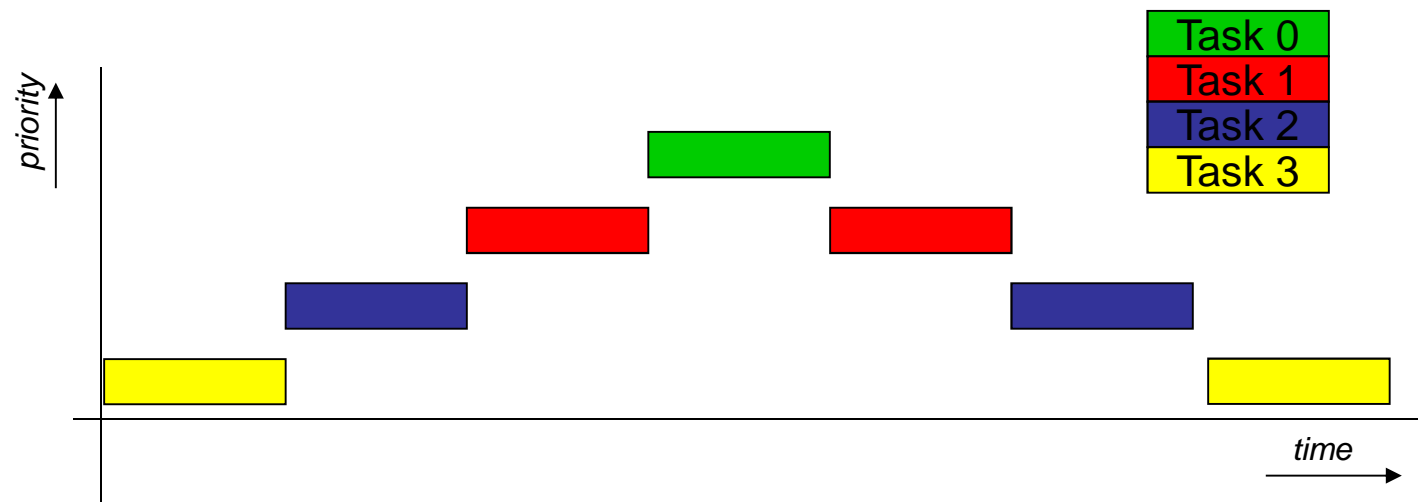


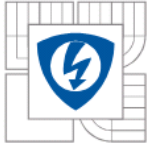


# Fixed Priority Scheduling

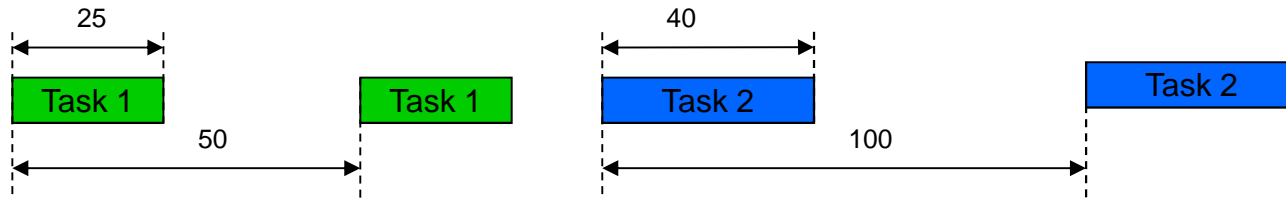
Každá úloha má nastavenou prioritu (celé číslo, *obvykle* nižší hodnota značí vyšší prioritu).

Pokud dvě úlohy mají stejnou prioritu, tak ta která přijde dřív dostane CPU a nebude přerušena.

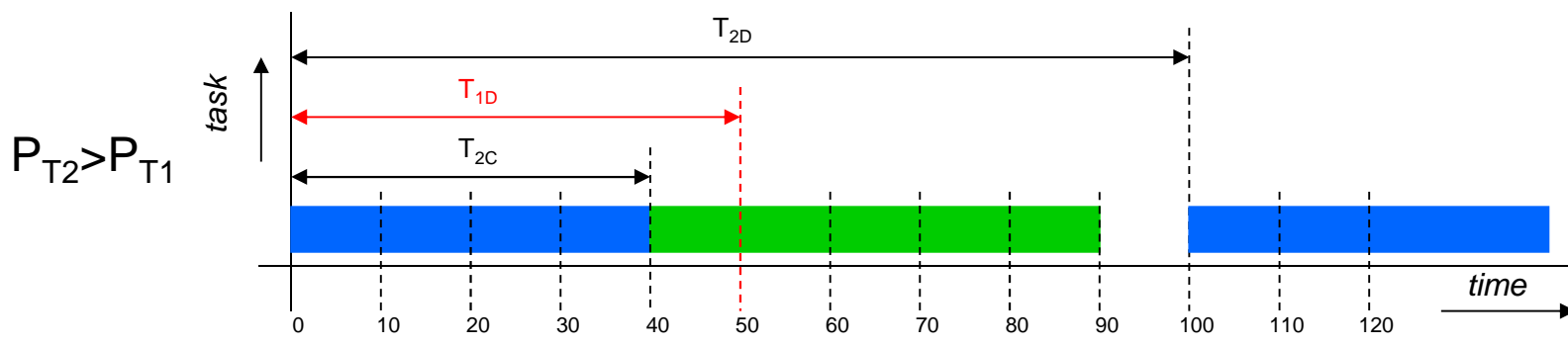
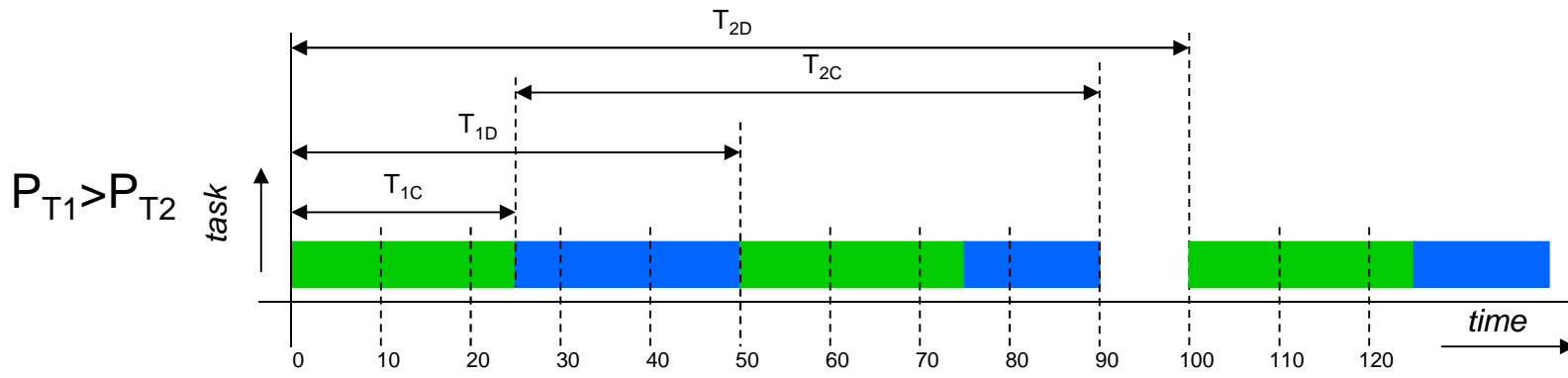


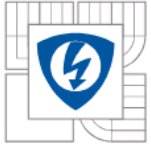


# Fixed Priority Scheduling Problem



$$U = \sum_{i=1}^n \frac{C_i}{T_i}$$





## Rate Monotonic Algorithm RMA

Algoritmus, který umožní nastavit prioritu úlohám s ohledem na splnění jejich deadlines.

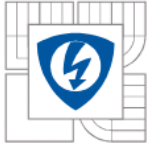
Každé úloze je přiřazena priorita dle periody jejího vykonávání. Kratší perioda->vyšší priorita.

RMA je optimálním statickým plánovacím algoritmem; selže-li, pak selže jakýkoliv jiný statický plánovací algoritmus.

$$U \leq n(\sqrt[n]{2} - 1)$$

Pro  $n \rightarrow \infty$  je  $U \leq 69,3\%$ . Zbývajících 30,7% lze tedy využít pro ostatní (non-realtime) úlohy.

Nevýhodou je, že často periody a délky exekuce neznáme.



# Fixed-Priority Preemptive Round-Robin Scheduling

Pokud je naplánováno (ve stavu READY) více úloh se stejnou prioritou, pak se tyto plánují podle round-robin algoritmu.

