



**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>



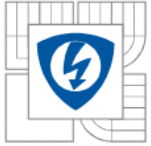
# Operační Systémy Reálného Času

## Úvod

Pavel Kučera

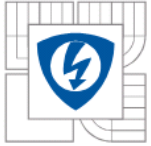
CAK, E112

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



# Obsah

1. Klasické řízení
2. Paralelismus a OS
3. Co je RTOS
4. Základní pojmy
5. Příklady
6. RTX – Clocks & Sleep



RTOS - I



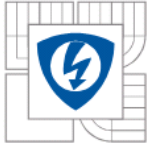
# Literatura

Douglas, B., P. *Doing Hard Time*, Reading, Mass.:Addison-Wesley, 2000.

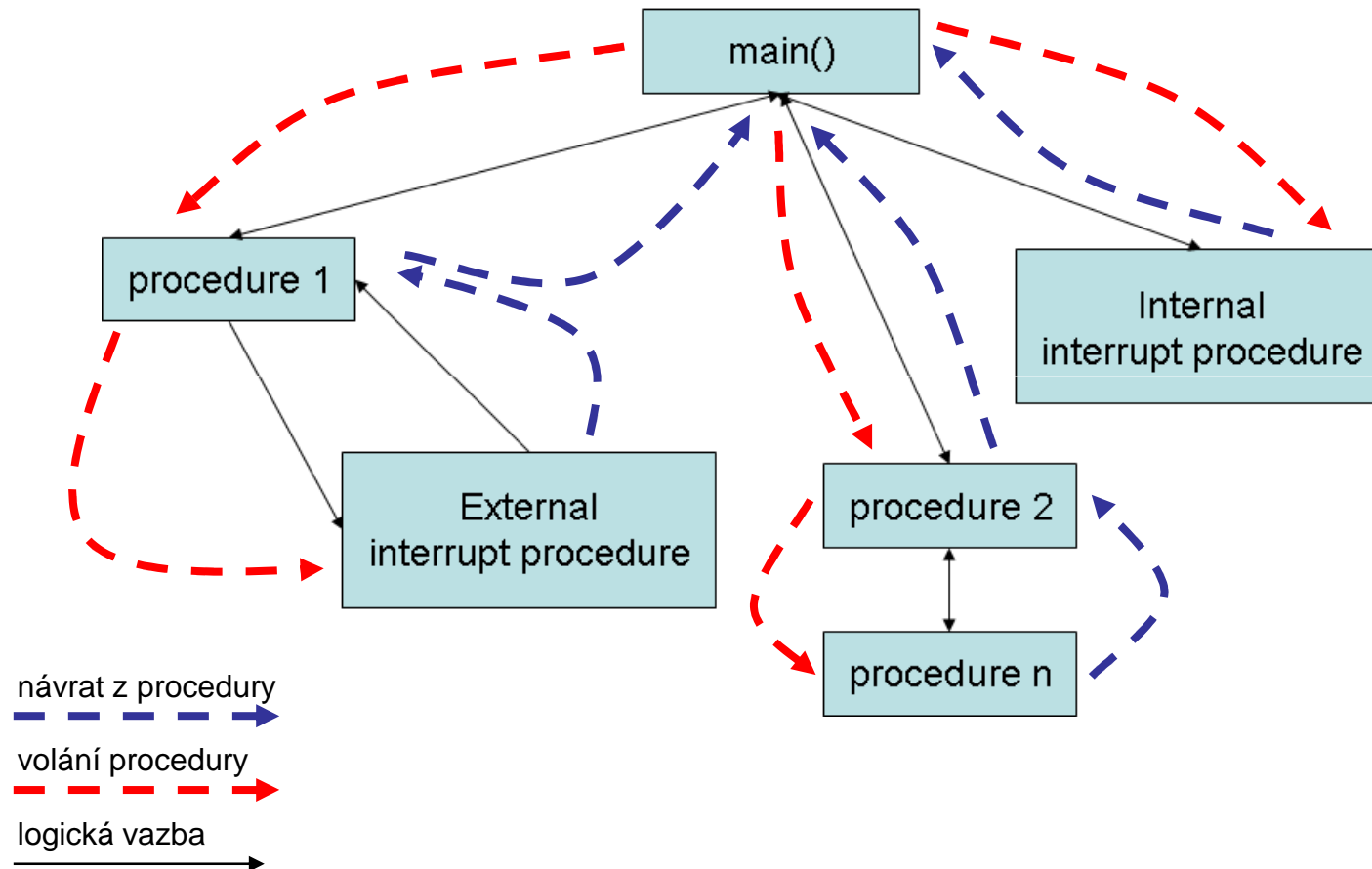
Li, Q., Yao, C. *Real-Time Concepts for Embedded Systems*:CMP Books, 2003.

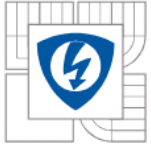
[www.real-time.org](http://www.real-time.org)

[www.taceo.eu](http://www.taceo.eu)

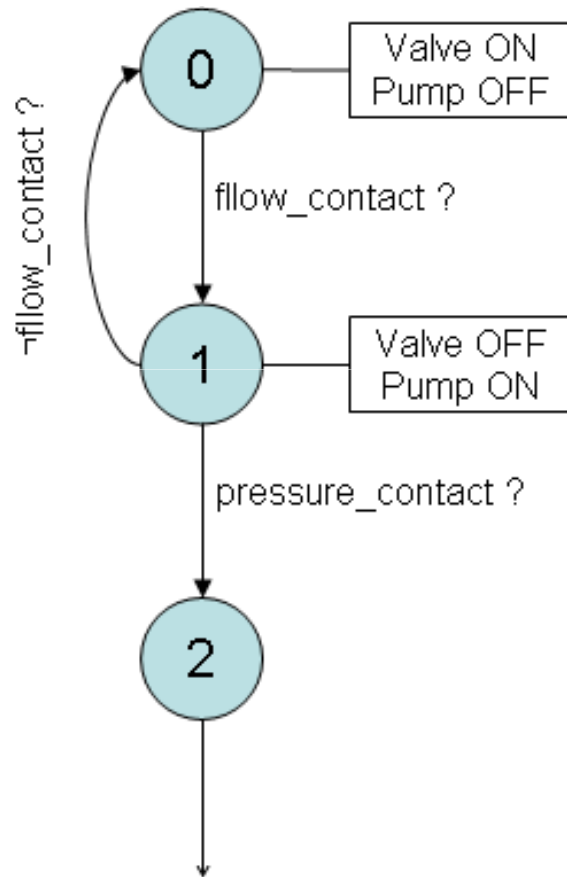


# Řízení v supersmyčce





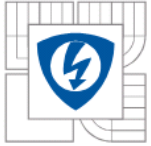
# Řízení v supersmyčce



stavový diagram úlohy

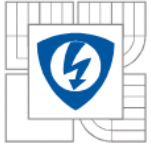
```
switch (actual_state) {  
  case 0: ActionInState0(); ConditionsInState0(); break;  
  case 1: ActionInState1(); ConditionsInState1(); break;  
  case 2: ActionInState2(); ConditionsInState2(); break;  
  .  
  .  
  case n: ActionInStateN(); ConditionsInStateN(); break;  
  default: DefaultAction(); DefaultConditions(); break;  
}  
  
ActionInState0() {  
  Valve = 1;  
  Pump = 0;  
}  
  
ConditionsInState0() {  
  if (flow_contact) actual_state = 1;  
}  
  
ActionInState1() {  
  Valve = 0;  
  Pump = 1;  
}  
  
ConditionsInState1() {  
  if (flow_contact) actual_state = 1;  
  if (!flow_contact) actual_state = 0;  
  if (pressure_contact) actual_state = 2;  
}
```

realizace v „super“ smyčce



## Řízení v supersmyčce

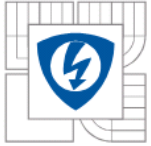
- Intuitivní řešení
  - Není potřeba OS
  - Snadná implementace jednoduchých úloh
- 
- + Jednoduchost
  - + Rychlost
  - Není zaručena real-time odezva
  - Úpravy



# Paralelismus

## Motivace pro použití víceúlohového operačního systému:

- Dekompozice úlohy na samostatné procesy vede ke zjednodušení procedur a ke snadnějšímu ladění/ověření jejich funkce.
- Selhání jednoho procesu nemusí nutně vést k selhání celého systému; důležité v safety and fault-tolerant systémech.
- Procesům lze přiřadit priority a ovlivňovat tak jejich šance při soupeření o jednotné zdroje (CPU, paměť, HW ...).
- Paralelismus se systémem priorit a deterministickým plánovačem jádra (kernelem) je **nutnou** podmínkou (nikoliv dostačující) pro real-time chování řídicí úlohy.

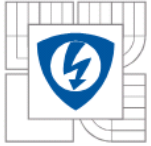


# Nepreemptivní OS

V kooperativním multitaskingu dostávají postupně jednotlivé úlohy čas ke své exekuci.

Plánovač úloh (proces v jádře OS, který je zodpovědný za multitaskingové chování systému) je v tomto případě založen na některém z těchto principů:

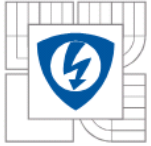
- úloha, která přijde dřív (nebo která má větší prioritu) dostane CPU
- FIFO (FCFS) zásobník úloh, který determinuje pořadí úloh
- Cyclic executive zásobník, který determinuje pořadí úloh



# Nepreemptivní OS

Jakmile úloha vstoupí do CPU, setrvává tam tak dlouho, dokud sama neskončí nebo se sama neuspí.

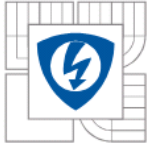
Nepreemptivní OS má minimální nebo žádnou možnost úlohu, zastavit/přerušit/odstranit a dát tak příležitost jiné úloze. V extrémních případech může dokonce úloha zablokovat činnost samotného OS (typický *problém* ve Windows 3.X aj. “kancelářských“ OS).



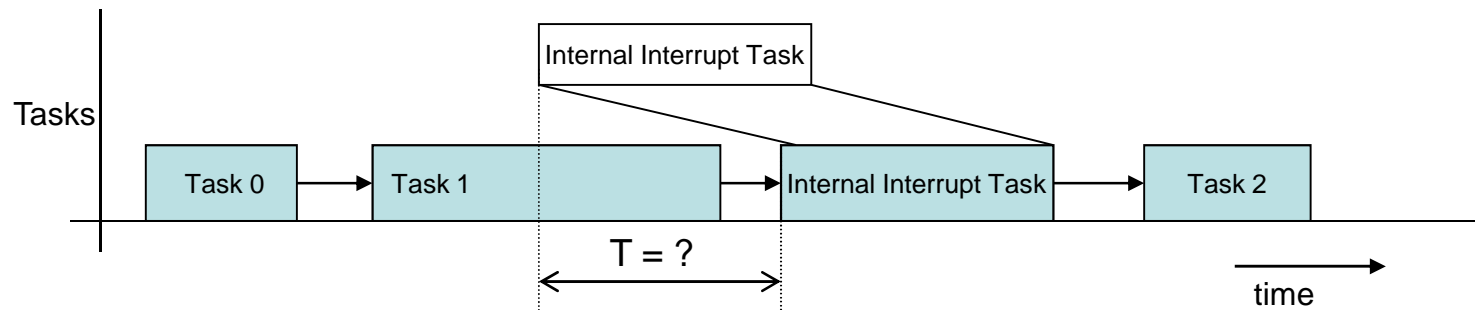
# Nepreemptivní OS

Nepreemptivní OS se vyznačují jednoduchou implementací, malými nároky na operační paměť a CPU. Taktéž umožňují efektivní způsob přepínání mezi úlohami, protože nedochází k přerušování úloh (preemci) a tím k častému ukládání stavu úlohy do paměti (zásobníku)

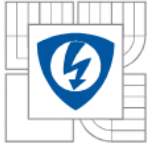
Chování úlohy v nepreemptivním OS je ale časově nedeterministické. Pro vícero spuštěných úloh je téměř nemožné zjistit doby obsluhy (přibližně lze statisticky) natož pak definovat real-time chování aplikace.



# Nepreemptivní OS



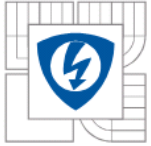
Na obrázku je znázorněn časový sled vykonávání úloh v nepreemptivním OS. Úloha 0 skončila a plánovač spustil úlohu 1. Uprostřed vykonávání této úlohy však přišel požadavek na vykonání jiné úlohy, zde označené jako Internal Interrupt (např. uživatel stiskl tlačítko). Obsluha tohoto přerušení však nemůže být vykonána dokud CPU zpracovává úlohu 1. Až ta skončí nebo se uspí, tak se spustí obsluha Internal Interrupt. Čas mezi okamžikem, kdy přišel požadavek na obsluhu přerušení a kdy k tomu skutečně došlo nelze předvídat a chování celého systému je časově nedeterministické.



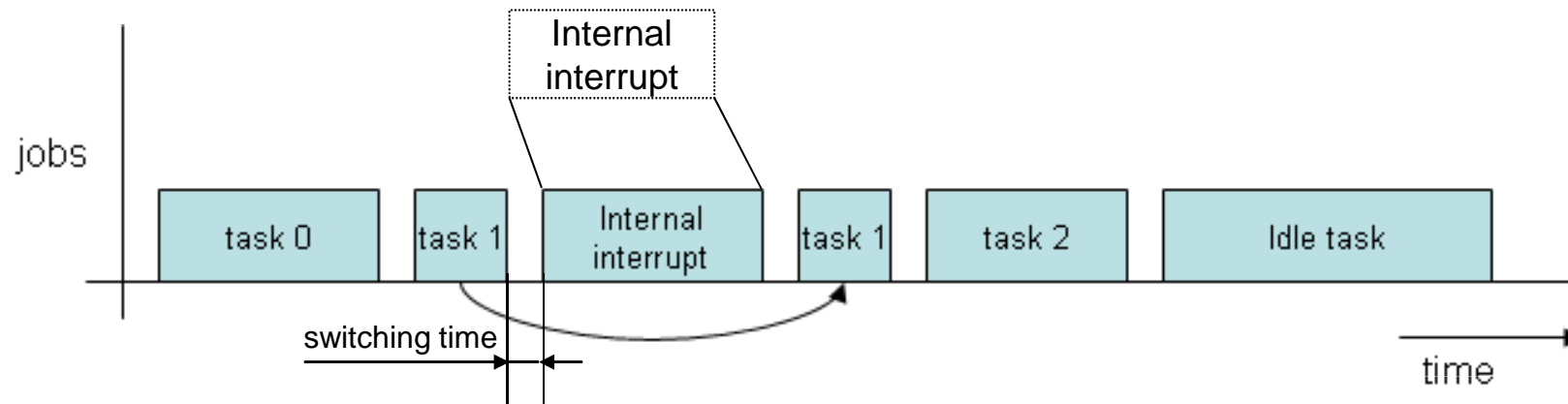
# Nepreemptivní OS

Operační systém založený na nepreemptivním multitaskingu umožňuje vytvářet řídicí úlohy stejným způsobem jako při řízení ve smyčce s tím, že máme k dispozici více-úlohové prostředí, často s možností nastavení priorit jednotlivým úlohám.

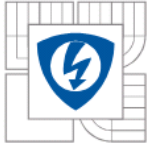
- + Jednoduchost
- + Rychlost
- + Úpravy
- **Není zaručena real-time odezva**



# Preemptivní OS

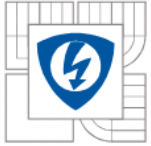


- + Úpravy
- + Možnost zaručit real-time odezvu
- Složitost OS
- Režije



# Preemptivní OS

- Preemptivní OS není sám o sobě RTOS. Tím se **může** stát, pokud obsahuje deterministický plánovač jádra.
- Systém řízení realizovaný v RTOS neběží automaticky v reálném čase! Podmínka realizace v RTOS je pouze nutná, nikoliv však dostačující. Programátor **musí** vždy řídicí úlohu přizpůsobit schopnostem a možnostem daného RTOS na dané HW platformě.



# Motivace

## *1982 - Selhání terapeutického přístroje Therac-25*

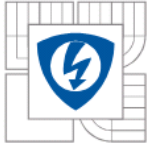
Kompletní řízení systému většinou pomocí SW kvůli snížení ceny a zvýšení flexibility. Díky fatálním chybám v návrhu celého systému (nebyl použit ověřený RTOS) došlo postupně k usmrcení 6-ti pacientů.

zdroj: Levenson, Nancy. *Safeware*, Reading, Mass.:Addison-Wesley, 1995

## *1991 - Selhání systému Patriot během války v Perském zálivu*

Řídicí systém pracoval více než 100 hodin bez restartu na což nebyl testován. Díky zaokrouhlování hodin při ukládání do paměti došlo k odchylce 0.3433 s oproti reálnému času a radar při druhém zaměření hledal střelu Scud na nesprávném místě (o 687 metrů dál). Střela Patriot nebyla vypuštěna a Scud zasáhla kasárna, kde usmrtila 28 lidí a zranila 97.

zdroj: <http://shelley.toich.net/projects/CS201/patriot.html>



RTOS - I



# Motivace

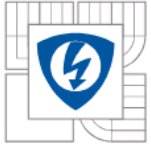
## *2008, Casting steel*

Inoperability of the real-time diagnostic system of the continuous casting process caused 20 mil. EUR financial lost in 2008.

## *2009, D.C. Metro Red Line Crash*

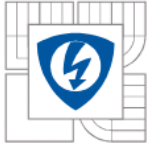
A Red Line Metrorail train crashed into a stationary train between Ft. Totten and Takoma stations. Nine people died and more than 70 people were injured. A train control system that should have prevented this deadly Metro crash failed in a test conducted by federal investigators.

source: [www.washingtonpost.com](http://www.washingtonpost.com)



## Co jsou RTOS ?

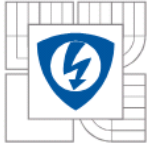
- assembler, C, 8-bit 4MHz CPU, 2K ROM, 256B RAM pro ledničky, pračky, mikrovlnné trouby, kardiostimulátory, řízení přímého vstřikování ...
- C, C++, Ada, 64-bit 3GHz CPU, MB RAM, TB HDD pro distribuované nebo centralizované řízení továren, letadel, elektráren ...
- často bez klávesnice, monitoru, komunikačních sběrnic
- musí pracovat dny/měsíce/roky bez přerušení a bez chyby (Fail-Safe, Fail-Stop, Fault-Tolerant)
- často za ztížených okolních podmínek (-40...+85°C, vlhkost 0...100%, vibrace ...)



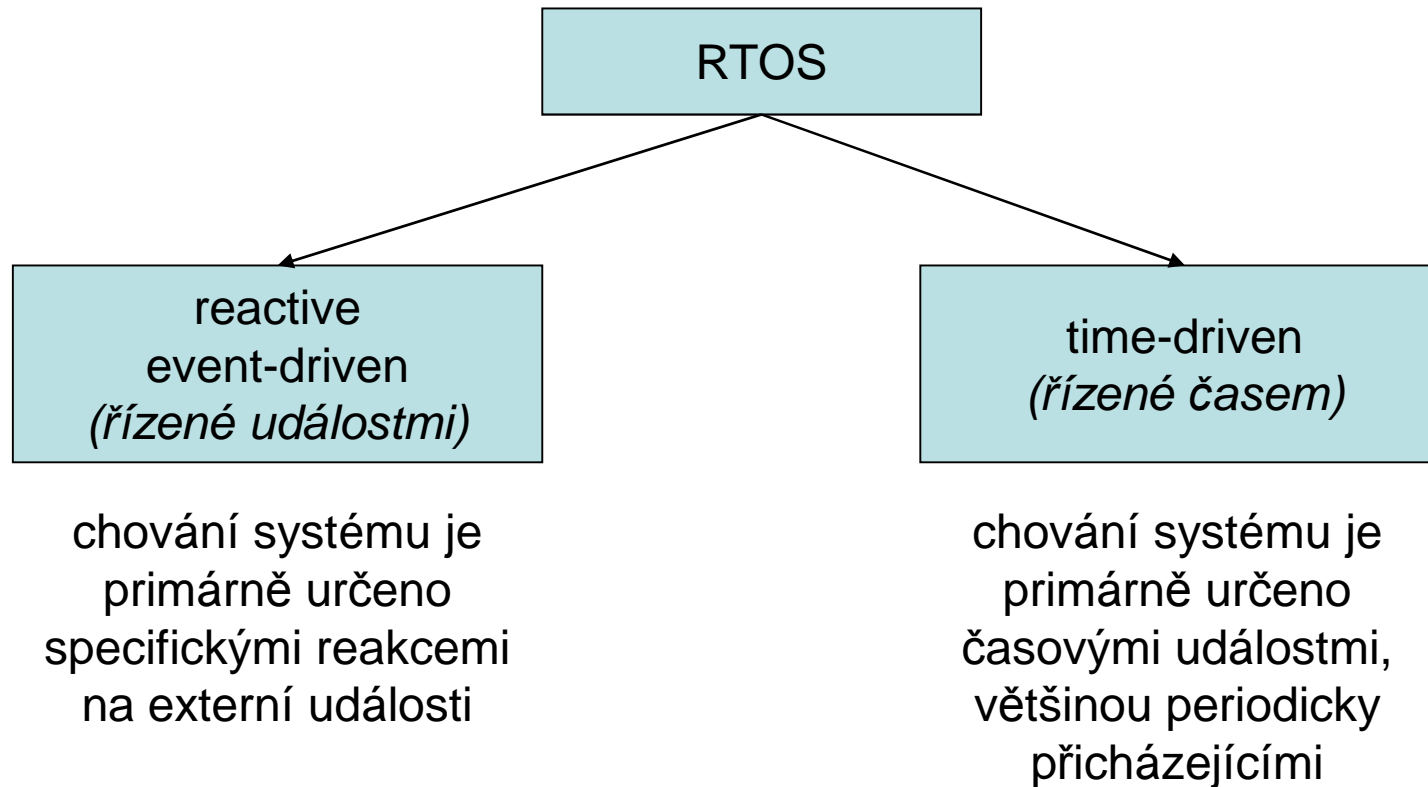
# Definice RTOS

**RTOS** je takový OS, který je schopen provádět výpočty a reagovat na události v definovaných časových intervalech – angl. deadlines.

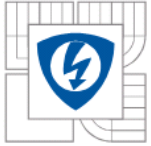
V informatickém pojetí je RTOS takový OS, který zaručuje, že je každá **správná** informace ve **správný** čas na **správném** místě.



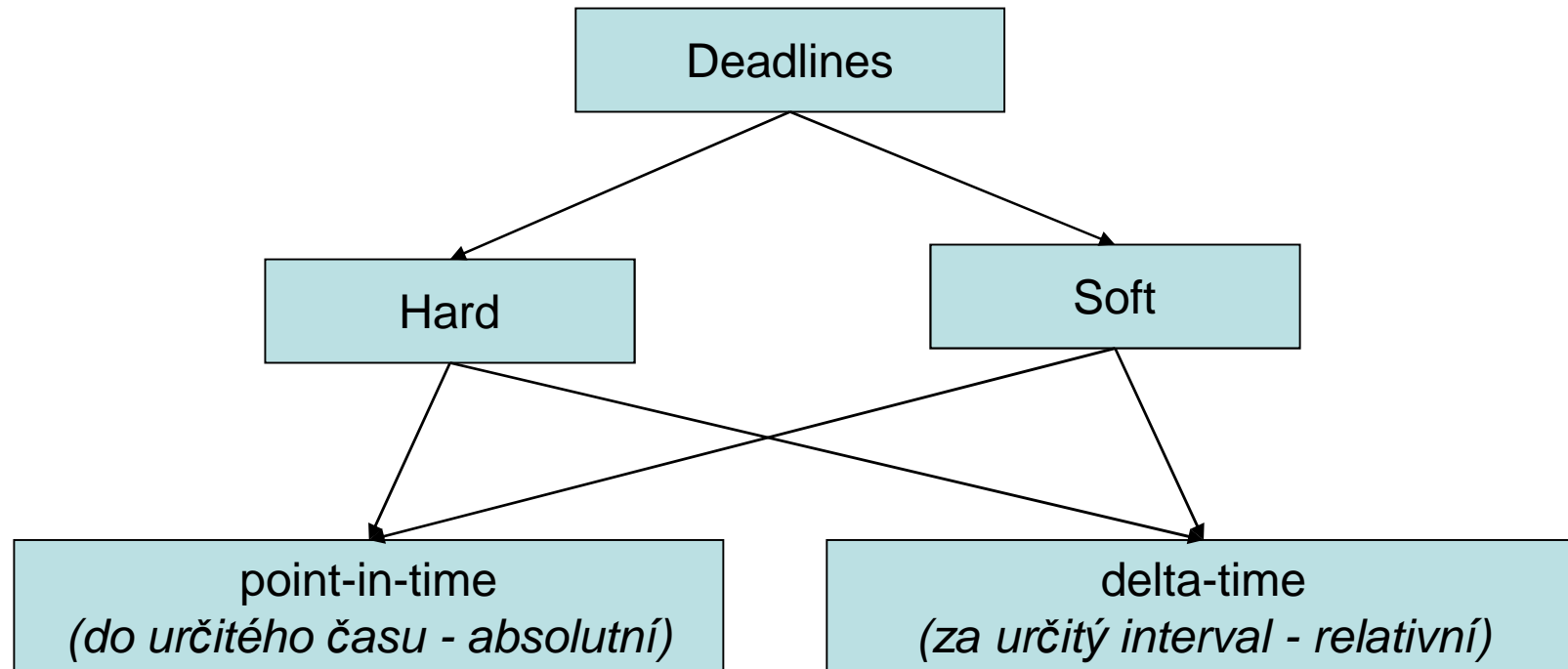
# Definice RTOS

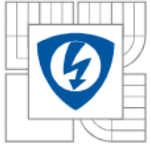


V praxi se lze často setkat s oběma požadavky současně.

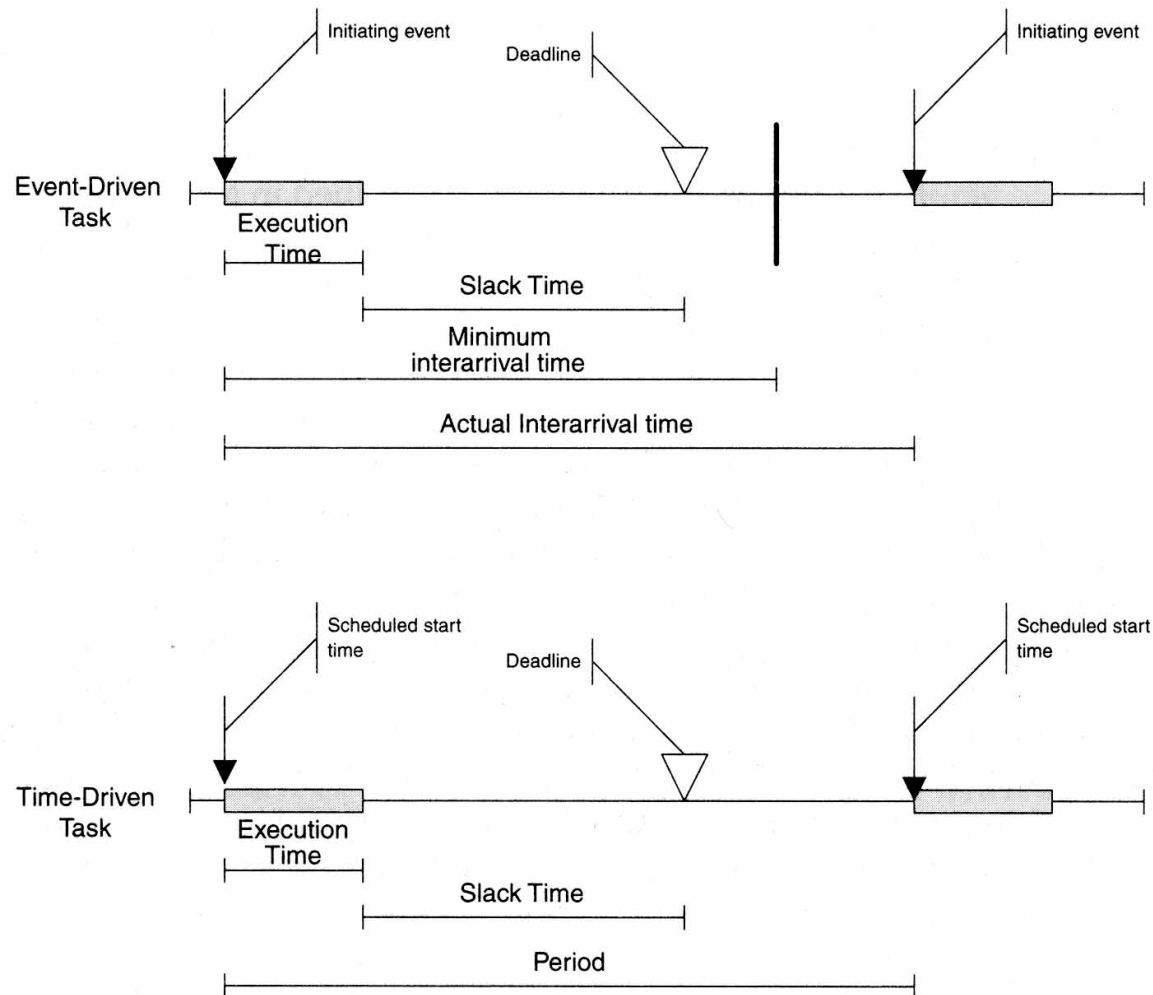


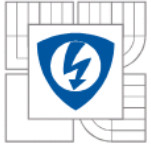
# Deadlines





# Deadlines





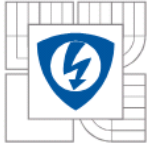
# Deadlines

## Hard deadlines

jsou takové požadavky na chování systému, které **musí** být **bezpodmínečně** splněny při každé vyskytnuvší se externí nebo časové události v systému.

Pokud by splněny nebyly, došlo by k selhání systému se všemi z toho vyplývajícími důsledky (finanční ztráta, zranění osob, ztráty na životech).

V informatickém pojetí jsou v hard deadlines systémech data, která přijdou pozdě bezcenná, popřípadě neplatná a mnohdy i potenciálně nebezpečná. Systém s nimi často **nesmí** dále pracovat.



# Deadlines

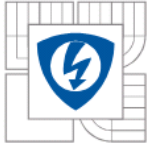
## Soft deadlines

jsou požadavky, u nichž stačí, když budou splněny v nějakém *průměrném* časovém intervalu nebo do určitého *průměrného* času.

V informatickém pojetí jsou v soft deadlines systémech data, které přijdou pozdě stále platná data s nimiž může systém dále pracovat (s jistými omezujícími faktory).

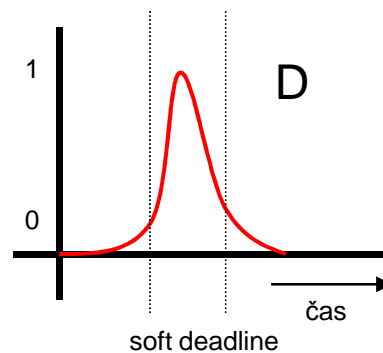
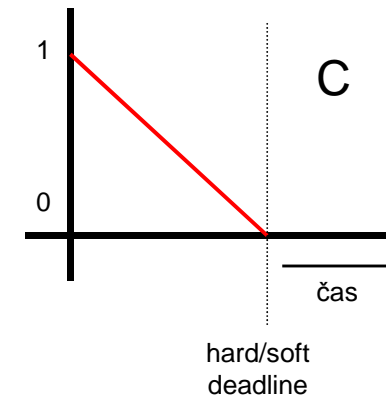
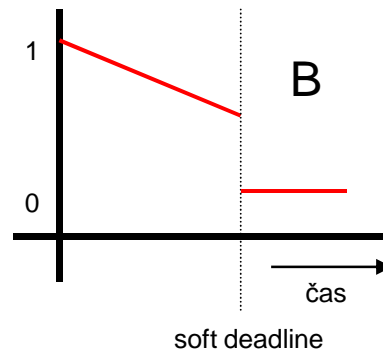
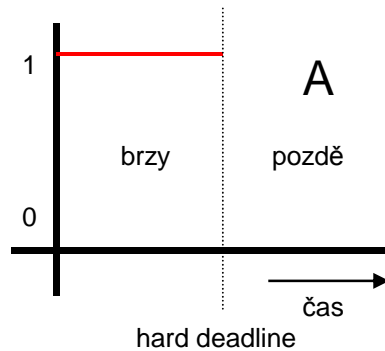
## Hard real-time X Soft real-time

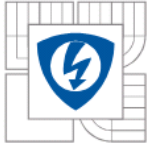
V praxi a zejména ve velkých systémech se často setkáváme s oběma požadavky v rámci jednomu systému.



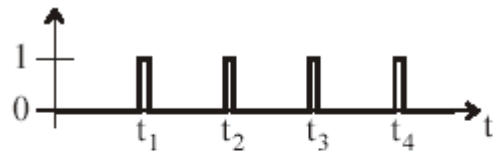
# Utility function

Česky bychom řekli užítková funkce nebo funkce užitečnosti či využitelnosti; popisuje schopnost systému využít informace v závislosti na čase kdy přišla.





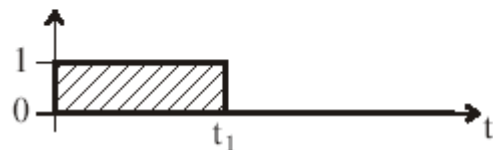
## Včasnost v řídicích úlohách



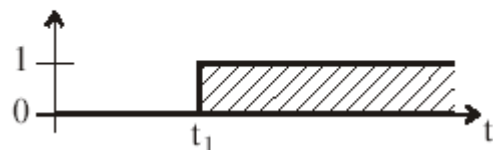
Akce musí být provedena v **přesně** stanovených časech  $t_1, t_2, \dots$ , přičemž povolen je jen malý jitter (vzorkování, synchronizace, komunikace ...).



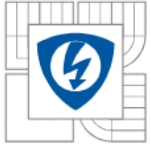
Akce musí být provedena v **daném** časovém intervalu (obsluha externích/interních přerušení).



Akce musí být provedena **nejpozději** do určitého časového okamžiku  $t_1$  (zpracování dat, přístup na sběrnici ...).



Akce může být provedena **nejdříve** po určitém časovém okamžiku  $t_1$  (čekání na události, invariantní podmínky v časovaných stavových automatech ...).



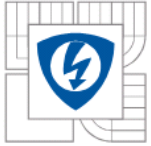
# Konečné stavové automaty a RTOS

Finite State Automata (FSA) je běžný způsob popisu systému diskretních událostí. FSA jsou založeny na diskretní matematice a vychází z těchto předpokladů, které nejsou vždy v RTOS splněny.

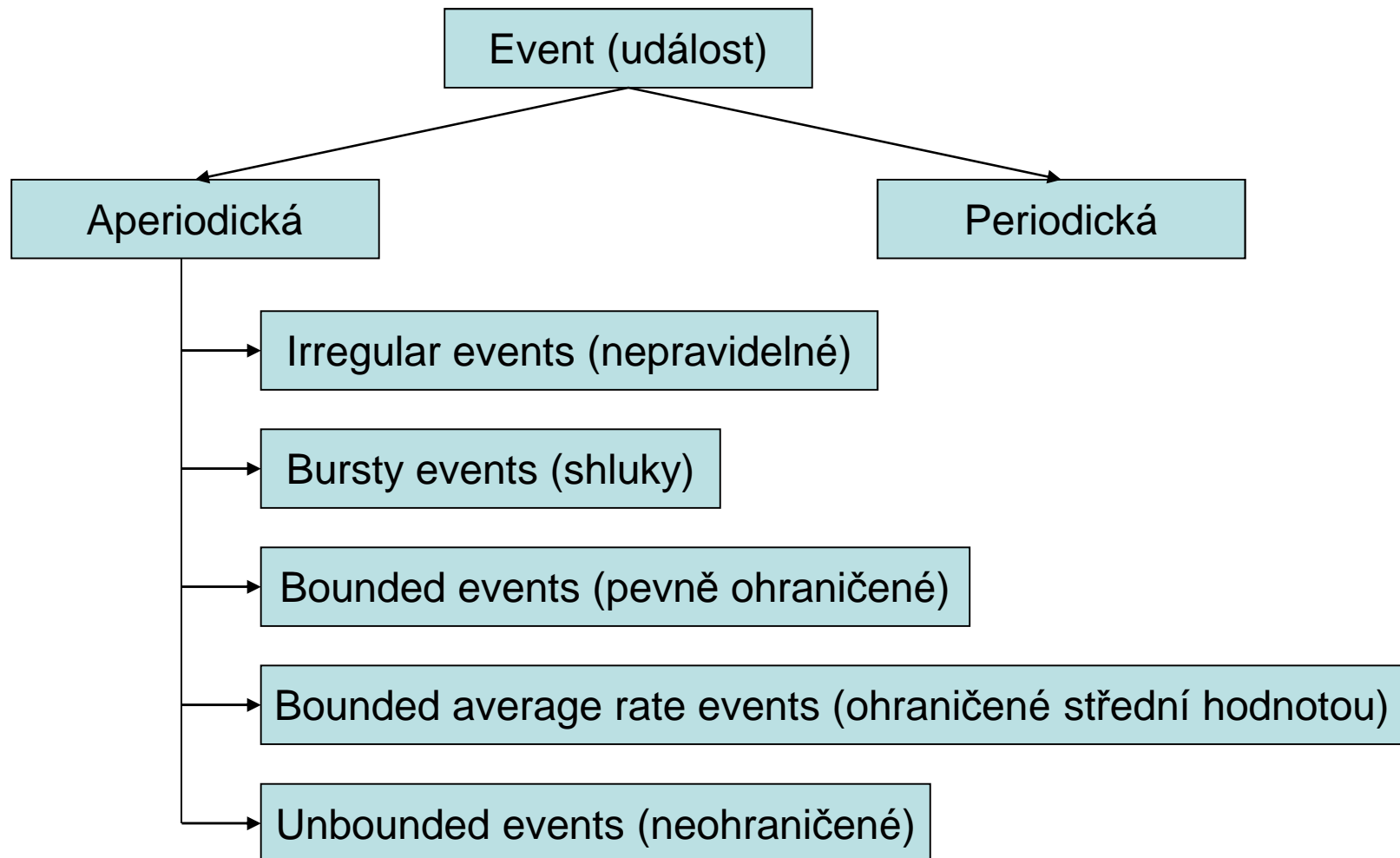
1. FSA se může nacházet v každém časovém okamžiku pouze v jednom definovaném stavu.
2. Přechody mezi stavy nelze přerušit.
3. Akce jsou atomické a provádějí se v čase jdoucím k nule.

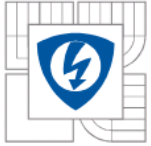
Mohou být vykonány:

- při vstupu do stavu
- při opuštění stavu
- během přechodu z jednoho stavu do druhého

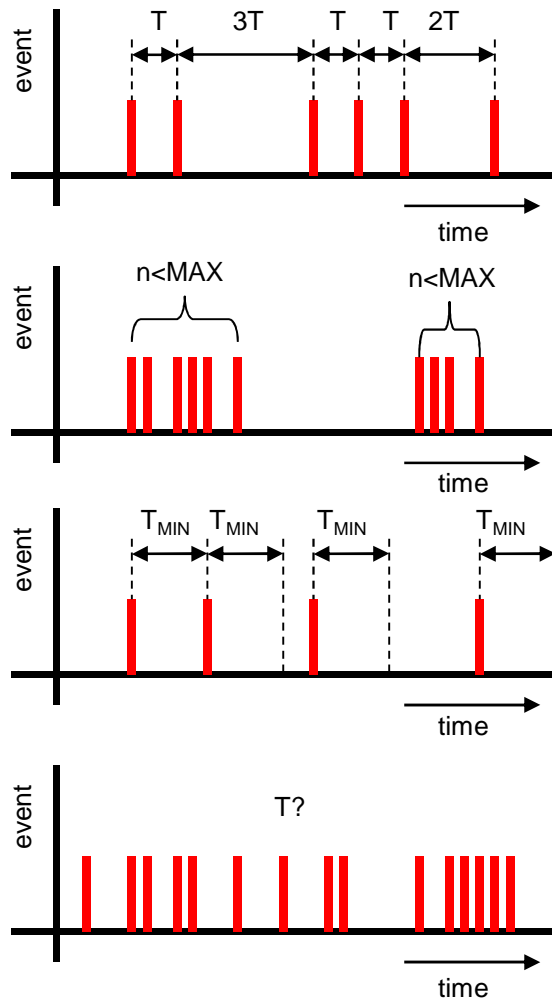


# Vzory externích událostí





## Vzory externích událostí



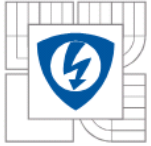
**Irregular** – známé, ale v čase se měnící intervaly mezi událostmi.

**Bursty** – sekvence událostí libovolně blízko sebe, ale jejich počet je shora ohraničený.

**Bounded** – sekvence událostí které jsou od sebe vzdáleny o jistý minimální interval.

**Bounded average rate** – totéž co bounded, ale interval je určen střední hodnotou.

**Unbounded** – sekvence událostí jejichž časové intervaly příchodu je možné určit pouze statisticky. Řídí se potom funkcí hustoty pravděpodobnosti a to buďto jako vzájemně nezávislý výběr intervalů podléhající společnému rozložení nebo vzájemně závislé výběry intervalů.



## RTOS základní pojmy

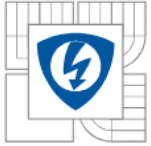
**Task** (úloha) – zapouzdřené logické sekvence operací, které jsou vykonávány nezávisle na jiných úlohách. Často, ale nikoliv vždy, je task rozdělen do vláken (threads).

**Thread** (vlákno) – primitivní exekuční blok (složen z instrukcí), který však může mít krom programového čítače i přidělený vlastní stack či statickou paměť.

**Proces** – prostředí, ve kterém běží jednotlivá vlákna. Proces jako takový nevykonává žádný exekuční kód. Vždy však obsahuje alespoň jedno vlákno.

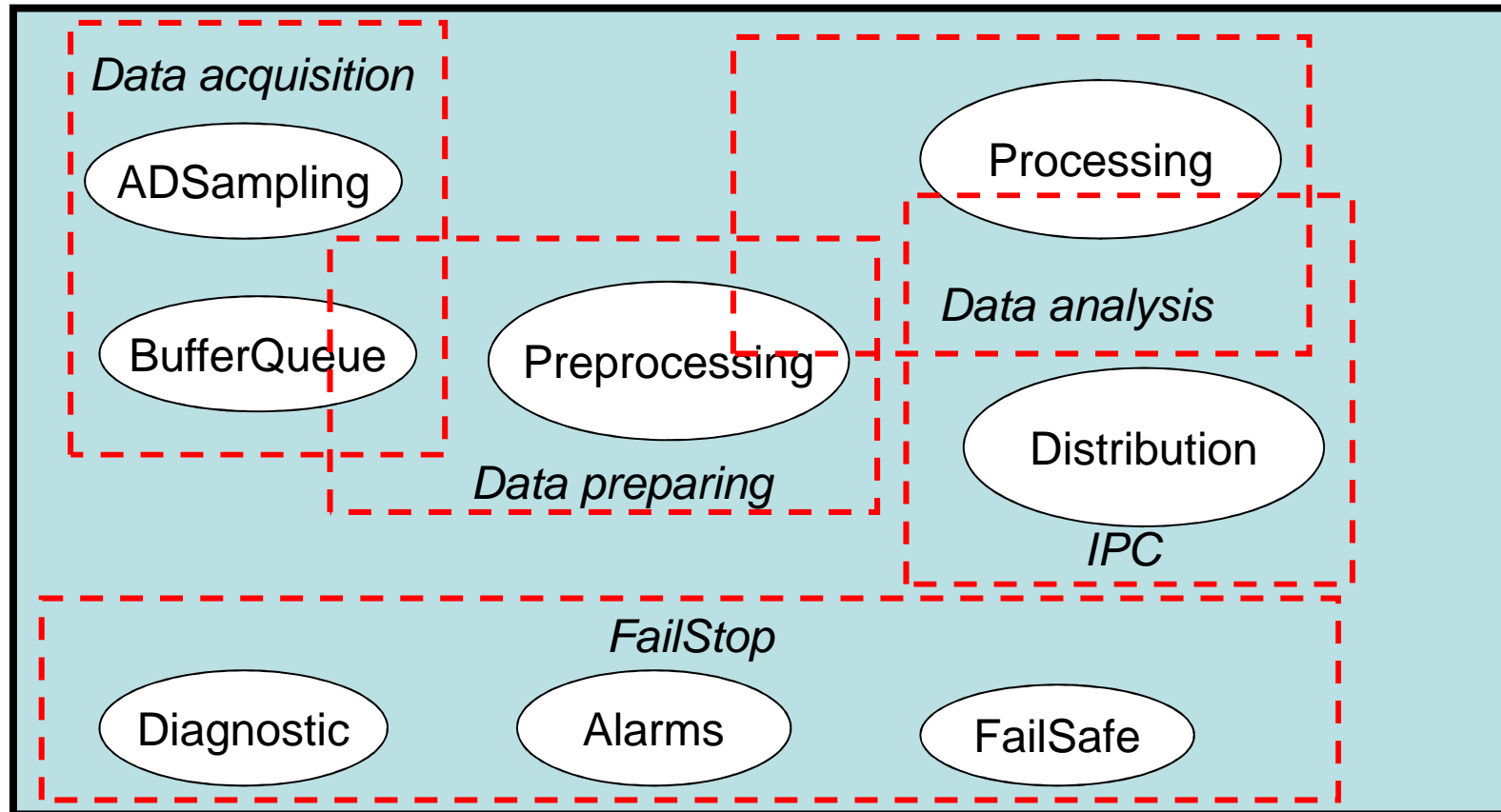
**Kernel** (jádro) – v RTOS je jeho hlavní funkcí (krom mnoha jiných) zajistit deterministické plánování exekuce jednotlivých vláken.

**Context switch** (přepnutí kontextu) – vykonávání kódu je přepnuto z jednoho vlákna na druhé.



# Vztah Task-Thread-Process

process: MACHINE DIAGNOSTIC



Process

Thread

Task