

Implementacja maszyny wirtualnej dla funkcyjnych języków programowania wspierających przetwarzanie współbieżne.

Kajetan Rzepecki

**Wydział EAIiB
Katedra Informatyki Stosowanej**

30 grudnia 2013

Maszyna wirtualna - środowisko uruchomieniowe języków programowania uniezależniające je od platformy sprzętowej.

Maszyna wirtualna - środowisko uruchomieniowe języków programowania niezależniające je od platformy sprzętowej.

W skład pracy wchodzi:

- ✚ Implementacja interpretera kodu bajtowego.

Maszyna wirtualna - środowisko uruchomieniowe języków programowania uniezależniające je od platformy sprzętowej.

W skład pracy wchodzi:

- ✚ Implementacja interpretera kodu bajtowego.
- ✚ Implementacja kolektora obiektów nieosiągalnych.

Maszyna wirtualna - środowisko uruchomieniowe języków programowania uniezależniające je od platformy sprzętowej.

W skład pracy wchodzi:

- ✚ Implementacja interpretera kodu bajtowego.
- ✚ Implementacja kolektora obiektów nieosiągalnych.
- ✚ Implementacja Modelu Aktorowego (ang. Actor Model).

```
transmoglify(Data) ->  
  Pids = framework:spawn_bajilion_procs(fun do_stuff/1),  
  JSON = json:decode(Data),  
  framework:map_reduce(Pids, JSON). %% $#%~@
```

```
transmoglify(Data) ->
```

```
  Pids = framework:spawn_bajilion_procs(fun do_stuff/1),  
  JSON = json:decode(Data),  
  framework:map_reduce(Pids, JSON). %% $#%~@
```

✚ Mniejsza logika przepływu danych.

```
transmoglify(Data) ->
```

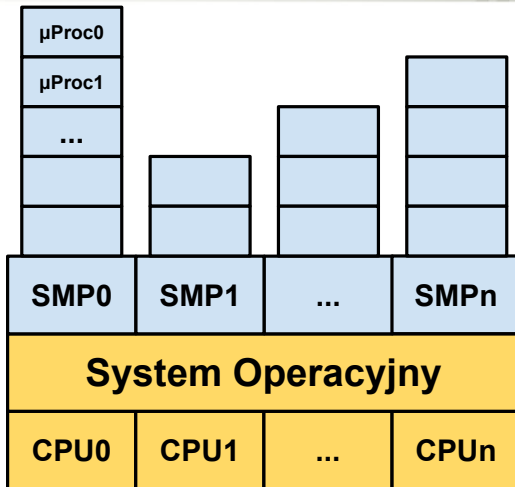
```
  Pids = framework:spawn_bajilion_procs(fun do_stuff/1),  
  JSON = json:decode(Data),  
  framework:map_reduce(Pids, JSON). %% $#%~@
```

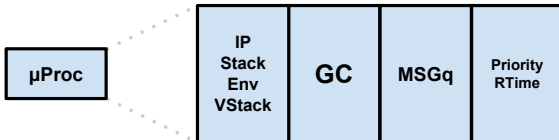
- ✚ Mniejsza logika przepływu danych.
- ✚ Zwielokrotnienie parsowania pliku JSON.

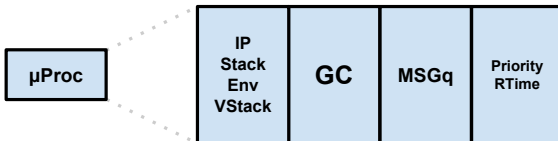

```
transmoglify(Data) ->
```

```
Pids = framework:spawn_bajilion_procs(fun do_stuff/1),  
JSON = json:decode(Data),  
framework:map_reduce(Pids, JSON). %% $#%~@
```

- ✚ Mniejsza logika przepływu danych.
- ✚ Zwielokrotnienie parsowania pliku JSON.
- ✚ Działa szybciej. (!?)

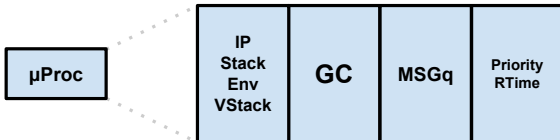






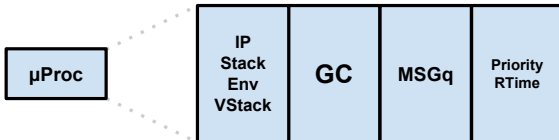
Oparty o **Three Instruction Machine**:

✚ Niewielka ilość rejestrów.



Oparty o **Three Instruction Machine**:

- ✚ Niewielka ilość rejestrów.
- ✚ Niewielka ilość instrukcji.



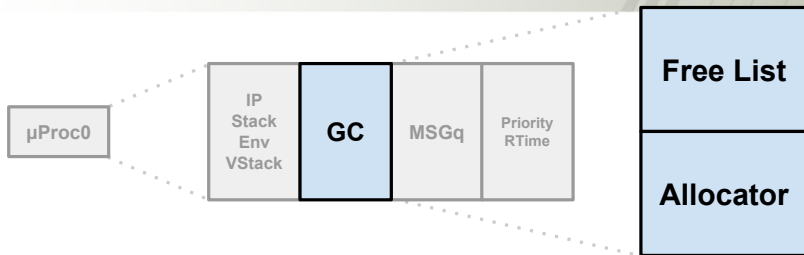
Oparty o **Three Instruction Machine**:

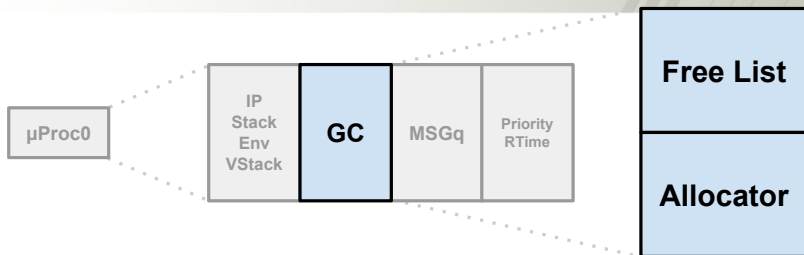
- ✚ Niewielka ilość rejestrów.
- ✚ Niewielka ilość instrukcji.
- ✚ Architektura **CISC**.

```
(define (add a b)
  (+ a b))
(add 2 2)
```

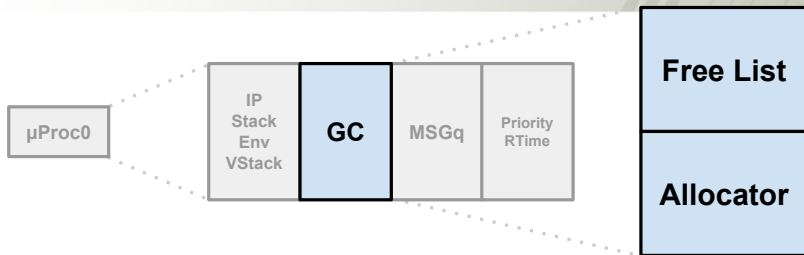
```
(define (add a b)
  (+ a b))
(add 2 2)
```

```
__start: PUSH 2      # Połóż "a" na stosie.
          PUSH 2
          ENTER add   # Wejdz do domknięcia "add".
add:      TAKE 2      # Pobierz dwa argumenty ze stosu.
          PUSH __add0
          ENTER 1     # PUSHC 2, RETURN
__add0:   PUSH __add1
          ENTER 0
__add1:   OP_ADD
          RETURN
```

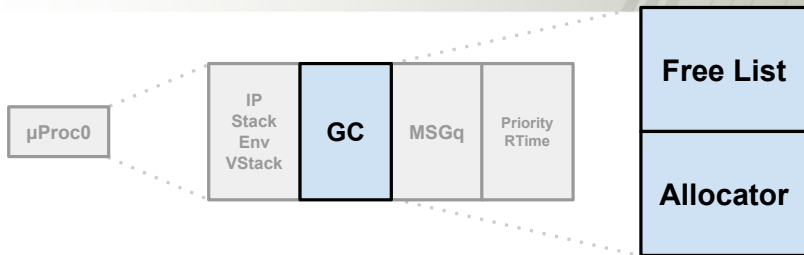





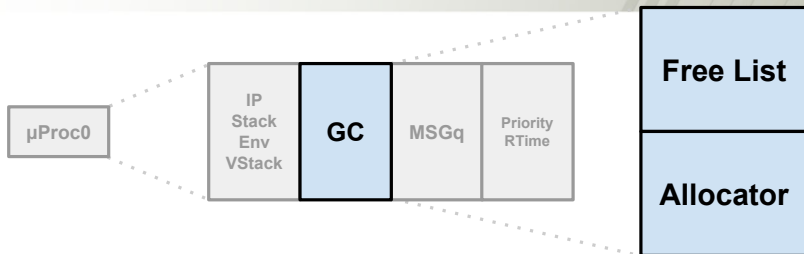
✚ Wykorzystuje leniwe zliczanie referencji.



- ✚ Wykorzystuje leniwe zliczanie referencji.
- ✚ Kolekcja pamięci procesu nie zależy od innych procesów.



- ✚ Wykorzystuje leniwe zliczanie referencji.
- ✚ Kolekcja pamięci procesu nie zależy od innych procesów.
- ✚ “Ostatni gasi światło.”



- ✦ Wykorzystuje leniwe zliczanie referencji.
- ✦ Kolekcja pamięci procesu nie zależy od innych procesów.
- ✦ “Ostatni gasi światło.”
- ✦ Proste obiekty (≤ 8 bajtów) są kopiowane.

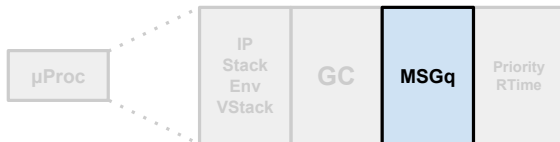
- ✚ Opóźnienie kolekcji obiektu do następnej alokacji.

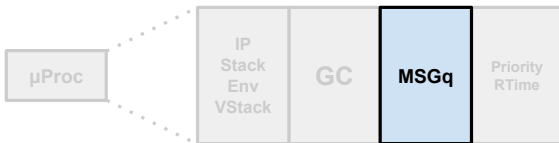
- ✦ Opóźnienie kolekcji obiektu do następnej alokacji.
- ✦ Szybkie dealokacje.

- ✦ Opóźnienie kolekcji obiektu do następnej alokacji.
- ✦ Szybkie dealokacje.
- ✦ Szybkie alokacje zamortyzowane listą wolnych obiektów.

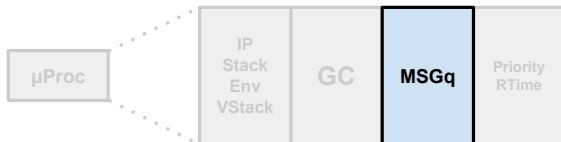
- ✦ Opóźnienie kolekcji obiektu do następnej alokacji.
- ✦ Szybkie dealokacje.
- ✦ Szybkie alokacje zamortyzowane listą wolnych obiektów.
- ✦ Pamięć nie jest natychmiastowo zwracana do Systemu Operacyjnego.

- ✦ Opóźnienie kolekcji obiektu do następnej alokacji.
- ✦ Szybkie dealokacje.
- ✦ Szybkie alokacje zamortyzowane listą wolnych obiektów.
- ✦ Pamięć nie jest natychmiastowo zwracana do Systemu Operacyjnego.
- ✦ Wymaga atomowych operacji na liczniku referencji oraz barier pamięci.

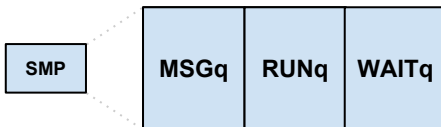


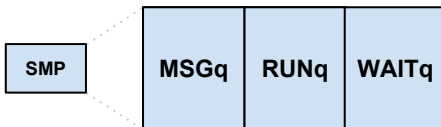


✚ Identyfikator procesu (pid) to wskaźnik na kontekst procesu.

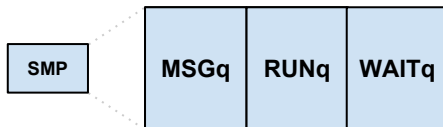


- ✚ Identyfikator procesu (pid) to wskaźnik na kontekst procesu.
- ✚ Wykorzystuje kolejki nieblokujące.

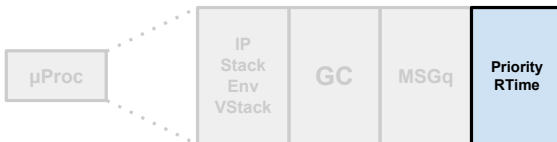




✚ Wykorzystuje Model Aktorowy!



✚ Wykorzystuje Model Aktorowy!



✚ Procesy są wywłaszczane (ang. preemptive concurrency).

Projekt implementuje:

- ✚ Interpreter kodu bajtowego oparty o **Three Instruction Machine**.

Projekt implementuje:

- ✚ Interpreter kodu bajtowego oparty o **Three Instruction Machine**.
- ✚ Kompilator kodu bajtowego.

Projekt implementuje:

- ✦ Interpreter kodu bajtowego oparty o **Three Instruction Machine**.
- ✦ Kompilator kodu bajtowego.
- ✦ Kolektor obiektów nieosiągalnych oparty o **leniwe zliczanie referencji**.

Projekt implementuje:

- ✦ Interpreter kodu bajtowego oparty o **Three Instruction Machine**.
- ✦ Kompilator kodu bajtowego.
- ✦ Kolektor obiektów nieosiągalnych oparty o **leniwe zliczanie referencji**.
- ✦ Architekturę SMP oraz Model Aktorowy oparty o **kolejki nieblokujące**.

Dziękuję za uwagę.

Kajetan Rzepecki