AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Implementacja maszyny wirtualnej dla funkcyjnych języków programowania wspierających przetwarzanie współbieżne.

Kajetan Rzepecki

Wydział EAIiIB
Katedra Informatyki Stosowanej

20 stycznia 2014

Wprowadzenie

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

Wprowadzenie

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

Cele pracy:

- Implementacja interpretera kodu bajtowego.

Wprowadzenie

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

Cele pracy:

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

Wprowadzenie

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

Cele pracy:

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

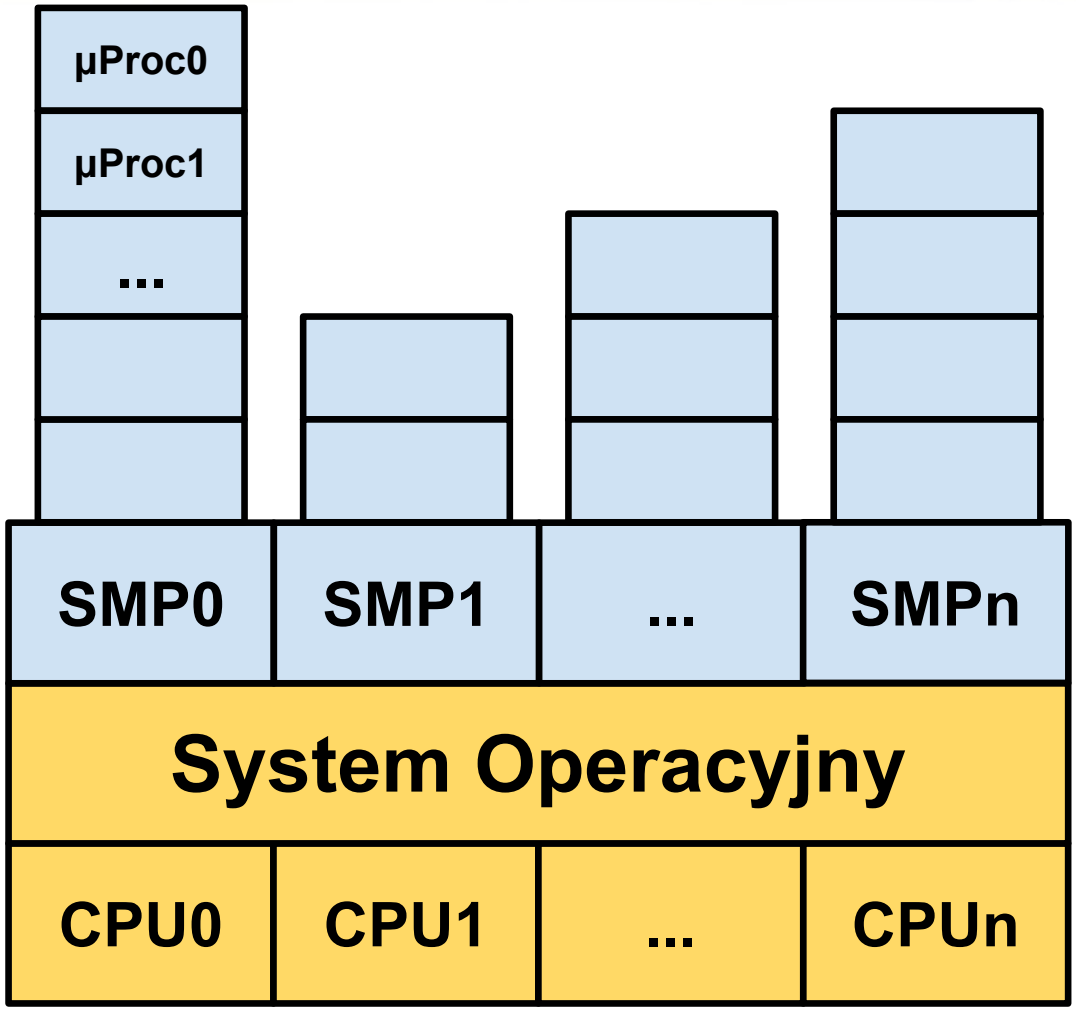
Wprowadzenie

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

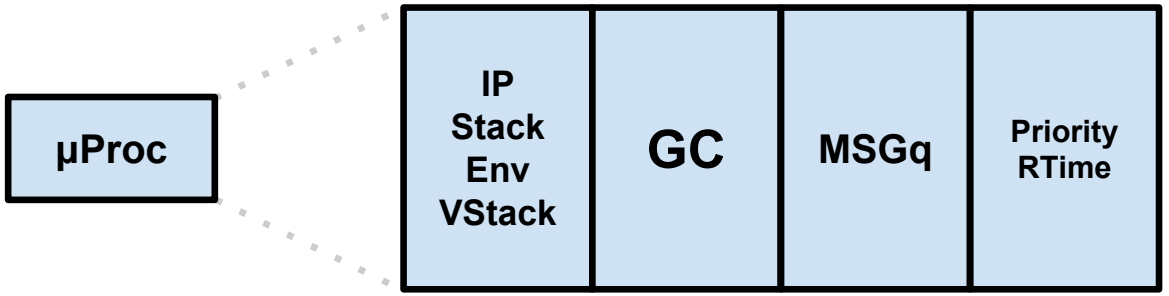
Cele pracy:

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

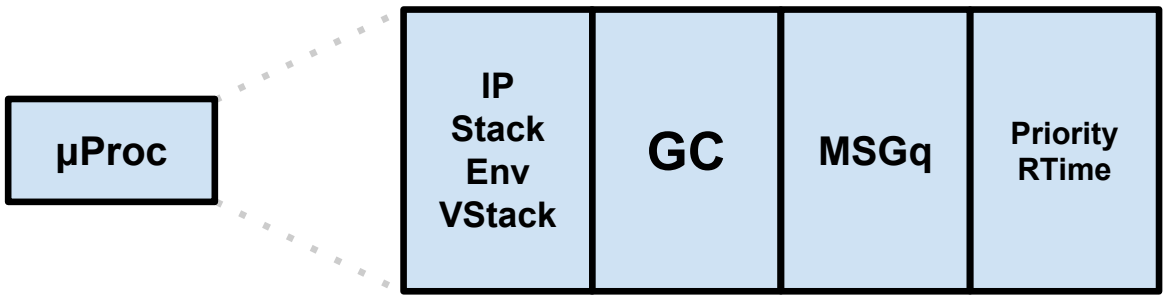
ThesisVM - Architektura



ThesisVM - Interpreter kodu bajtowego



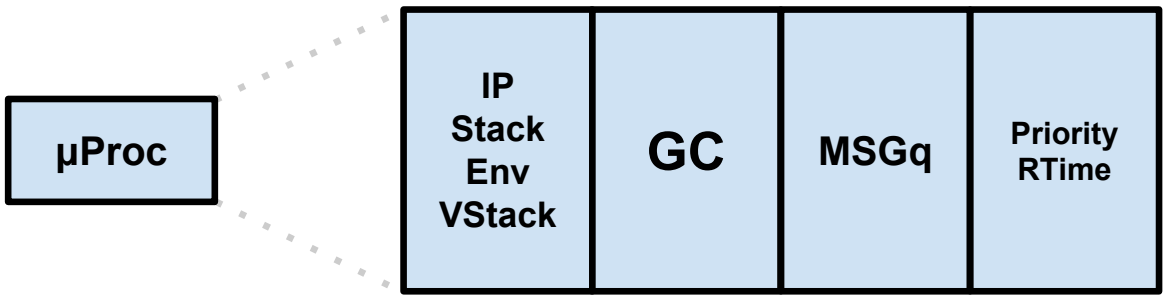
ThesisVM - Interpreter kodu bajtowego



Oparty o **Three Instruction Machine**:

- Niewielka ilość rejestrów.

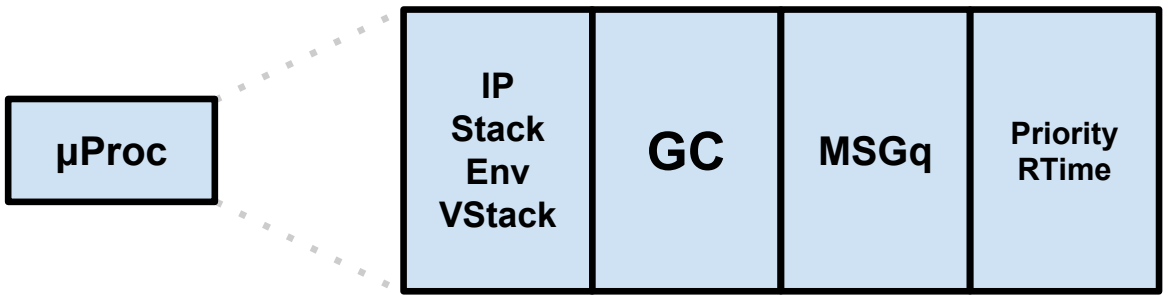
ThesisVM - Interpreter kodu bajtowego



Oparty o **Three Instruction Machine**:

- Niewielka ilość rejestrów.
- Niewielka ilość instrukcji.

ThesisVM - Interpreter kodu bajtowego



Oparty o **Three Instruction Machine**:

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

ThesisVM - Przykład kodu bajtowego

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

ThesisVM - Przykład kodu bajtowego

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

```
__start:  NEXT 2
          NEXT 2
          ENTER add
add:      TAKE
          TAKE
          NEXT __add0
          ENTER 1
__add0:   NEXT __add1
          ENTER 0
__add1:   PRIMOP '+'
          RETURN
```

ThesisVM - Kolekcja obiektów nieosiągalnych

ThesisVM - Kolekcja obiektów nieosiągalnych

- Pamięć jest współdzielona pomiędzy procesami.

ThesisVM - Kolekcja obiektów nieosiągalnych

- Pamięć jest współdzielona pomiędzy procesami.
- Wykorzystuje zliczanie referencji.

ThesisVM - Kolekcja obiektów nieosiągalnych

- Pamięć jest współdzielona pomiędzy procesami.
- Wykorzystuje zliczanie referencji.
- Kolekcja pamięci procesu nie zależy od innych procesów.

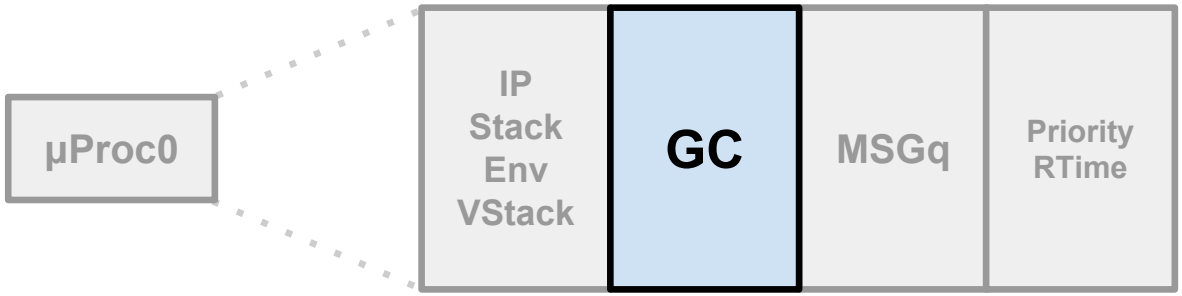
ThesisVM - Kolekcja obiektów nieosiągalnych

- Pamięć jest współdzielona pomiędzy procesami.
- Wykorzystuje zliczanie referencji.
- Kolekcja pamięci procesu nie zależy od innych procesów.
- “Ostatni gasi światło.”

ThesisVM - Kolekcja obiektów nieosiągalnych

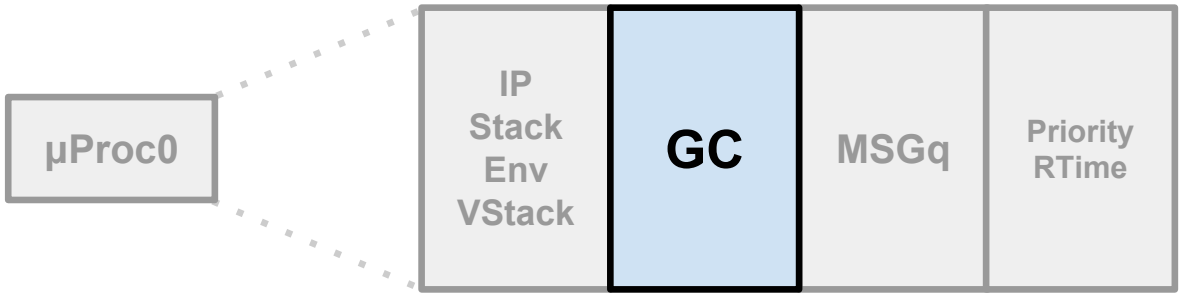
- Pamięć jest współdzielona pomiędzy procesami.
- Wykorzystuje zliczanie referencji.
- Kolekcja pamięci procesu nie zależy od innych procesów.
- “Ostatni gasi światło.”
- Proste obiekty ($\leq$ 8 bajtów) są kopiowane.

ThesisVM - Leniwe zliczanie referencji



Polega na opóźnieniu kolekcji obiektu do następnej alokacji poprzez wykorzystanie listy zwolnionych obiektów.

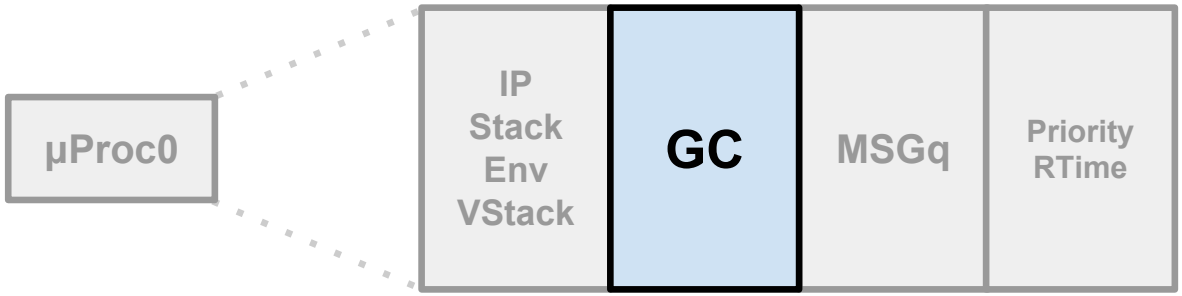
ThesisVM - Leniwe zliczanie referencji



Polega na opóźnieniu kolekcji obiektu do następnej alokacji poprzez wykorzystanie listy zwolnionych obiektów.

- Szybkie dealokacje.

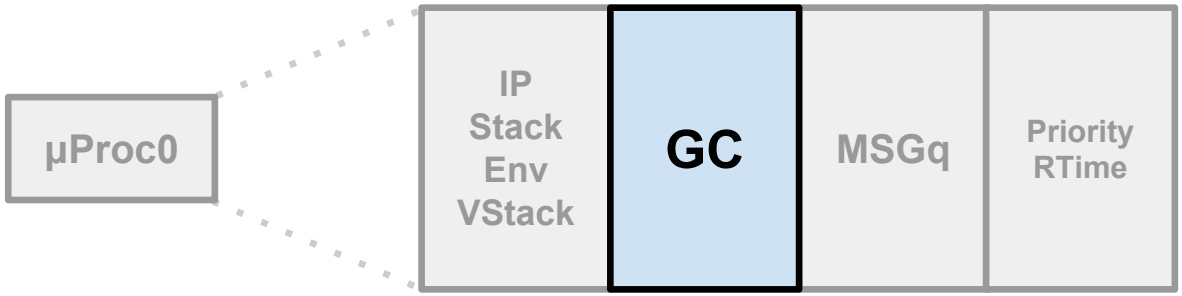
ThesisVM - Leniwe zliczanie referencji



Polega na opóźnieniu kolekcji obiektu do następnej alokacji poprzez wykorzystanie listy zwolnionych obiektów.

- Szybkie dealokacje.
- Szybkie alokacje zamortyzowane listą wolnych obiektów.

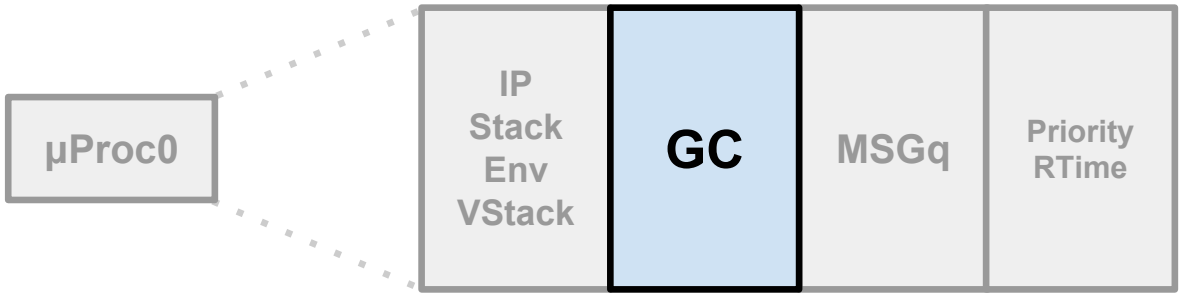
ThesisVM - Leniwe zliczanie referencji



Polega na opóźnieniu kolekcji obiektu do następnej alokacji poprzez wykorzystanie listy zwolnionych obiektów.

- Szybkie dealokacje.
- Szybkie alokacje zamortyzowane listą wolnych obiektów.
- Pamięć nie jest natychmiastowo zwracana do Systemu Operacyjnego.

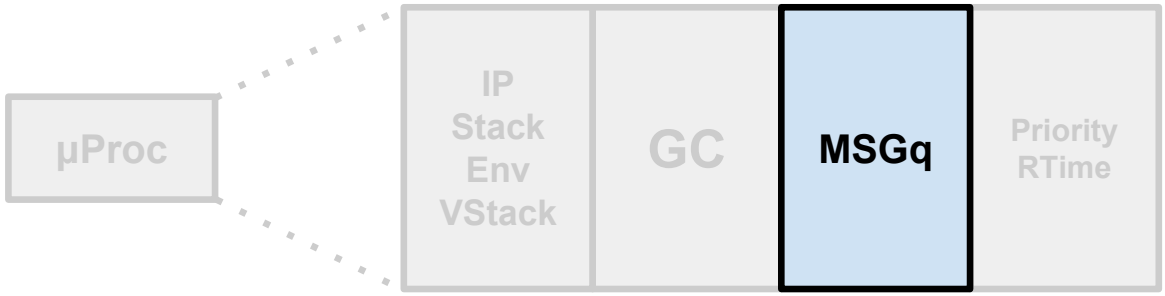
ThesisVM - Leniwe zliczanie referencji



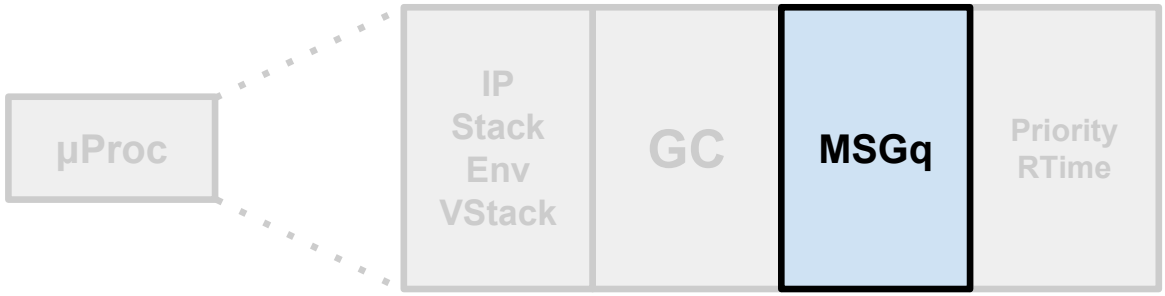
Polega na opóźnieniu kolekcji obiektu do następnej alokacji poprzez wykorzystanie listy zwolnionych obiektów.

- 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.

ThesisVM - Model Aktorowy

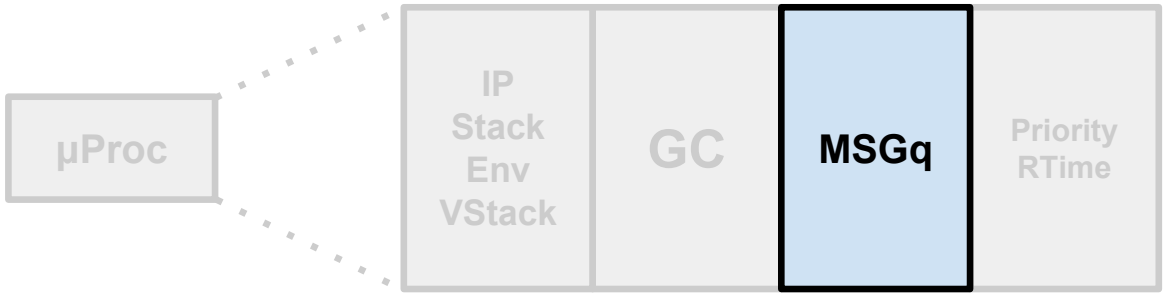


ThesisVM - Model Aktorowy



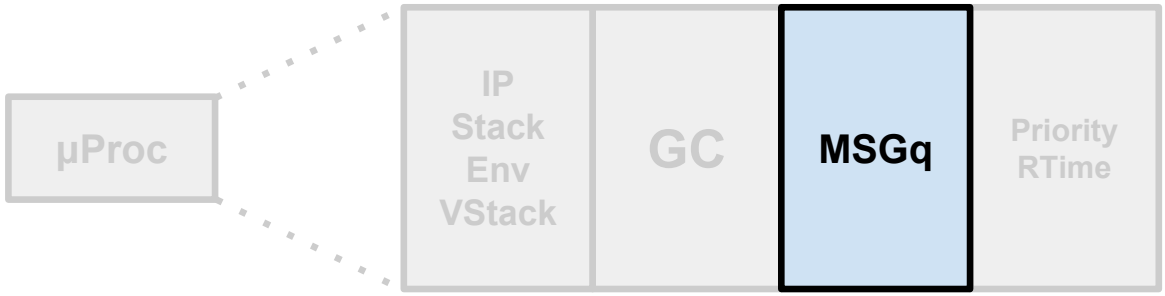
- Procesy są obiektami “pierwszej klasy”.

ThesisVM - Model Aktorowy



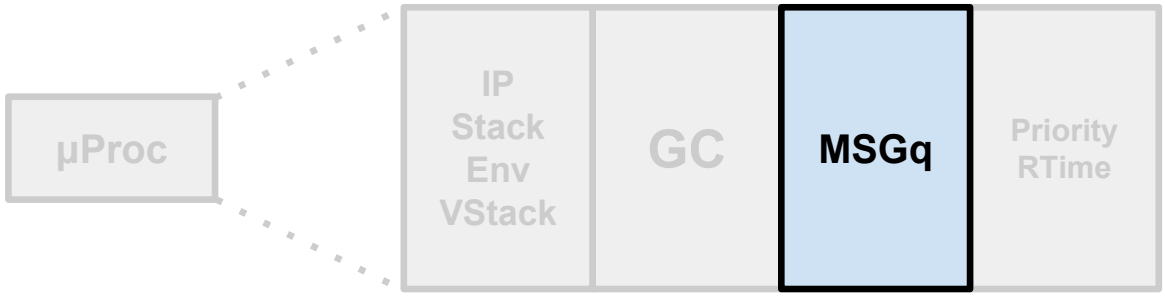
- Procesy są obiektami “pierwszej klasy”.
- Idetyfikator procesu (pid) to wskaźnik na kontekst procesu.

ThesisVM - Model Aktorowy



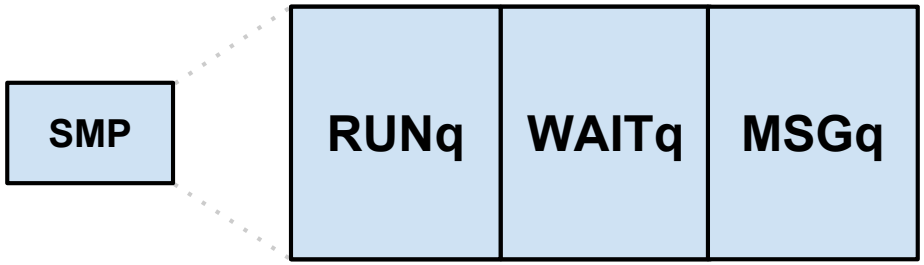
- Procesy są obiektami “pierwszej klasy”.
- Idetyfikator procesu (pid) to wskaźnik na kontekst procesu.
- Wiadomości są przesyłane asynchronicznie.

ThesisVM - Model Aktorowy

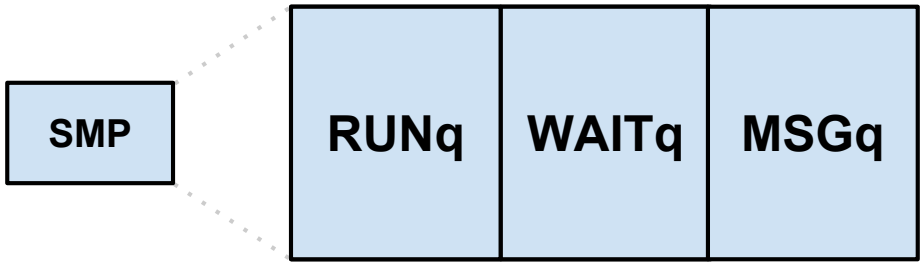


- Procesy są obiektami “pierwszej klasy”.
- Idetyfikator procesu (pid) to wskaźnik na kontekst procesu.
- Wiadomości są przesyłane asynchronicznie.
- Implementacja wykorzystuje kolejki nieblokujące.

ThesisVM - Symetryczny Multiprocessing

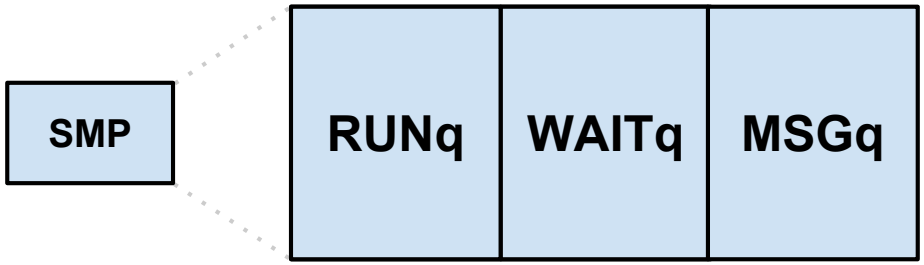


ThesisVM - Symetryczny Multiprocessing

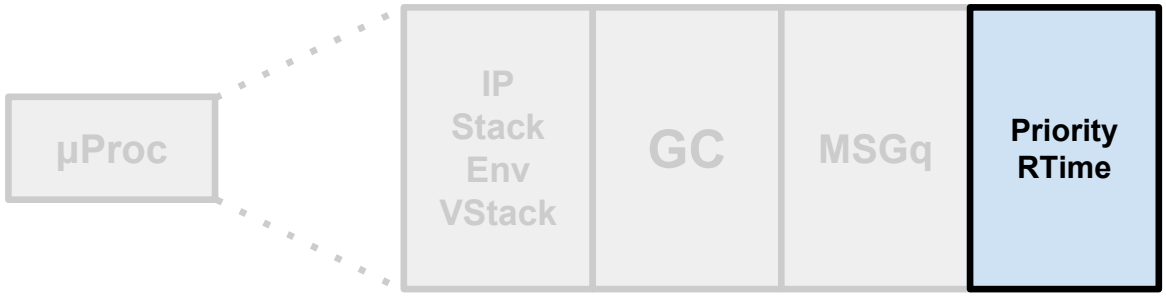


- Implementacja wykorzystuje Model Aktorowy!

ThesisVM - Symetryczny Multiprocessing

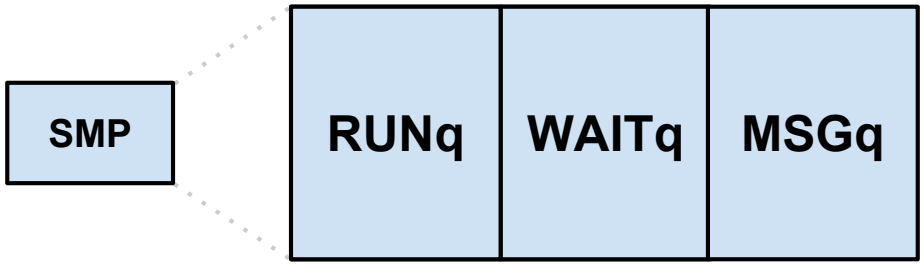


- Implementacja wykorzystuje Model Aktorowy!

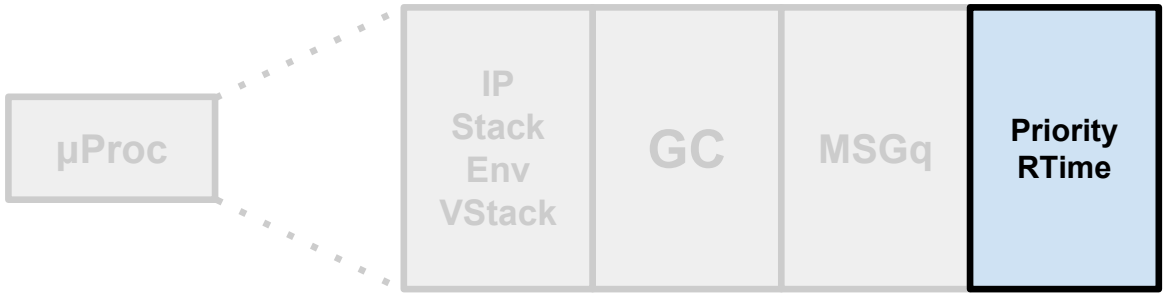


- Procesy są wywłaszczane (ang. preemptive concurrency).

ThesisVM - Symetryczny Multiprocessing



- Implementacja wykorzystuje Model Aktorowy!



- Procesy są wywłaszczane (ang. preemptive concurrency).
- Wykorzystuje algorytm **Completely Fair Scheduling**.

Podsumowanie

Projekt implementuje:

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

Podsumowanie

Projekt implementuje:

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

Podsumowanie

Projekt implementuje:

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

Podsumowanie

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**.