Jaka wirtualna maszyna równoległa?

PVM/MPI, Java RMI, ... czy **PVLM**

prof. dr hab. Andrzej Salwicki
A.Salwicki@itl.waw.pl

Instytut Łączności

9 grudnia 2009

Streszczenie

Zamierzamy omówić dwa tematy:

- PVLM - Równoległa Wirtualna Maszyna Loglanowska,
  Znana jest od wielu lat koncepcja PVM - maszyny równoległej powstającej przez połączenie komputerów siecią.
  My proponujemy łączyć Loglanowskie Maszyny Wirtualne VLP.
  - Zyskujemy wieloplatformowość, bo VLP może działac i na Windows i na Linuksie.
  - Zyskujemy znaczne uproszczenie i redukcję liczby operacji.

  Autorem tej koncepcji jest dr Oskar Świda.
- alien call - protokół obcego wołania procedur
  Postaramy się wykazać, że protokoły takie jak:
  Sunowski RPC, CORBA, Java RMI są znacznie bardziej skomplikowane od alien call.
  Protokół alien call wymyślił Bolesław Ciesielski w 1988,

Nasz plan

1. Połączone witualne maszyny loglanowskie = PLVM
2. klasa Process – wzorzec wg którego powstają obiekty aktywne
3. Obiekty Aktywne
4. Obce wołanie procedury
5. Zadanie z nagrodą

Plan

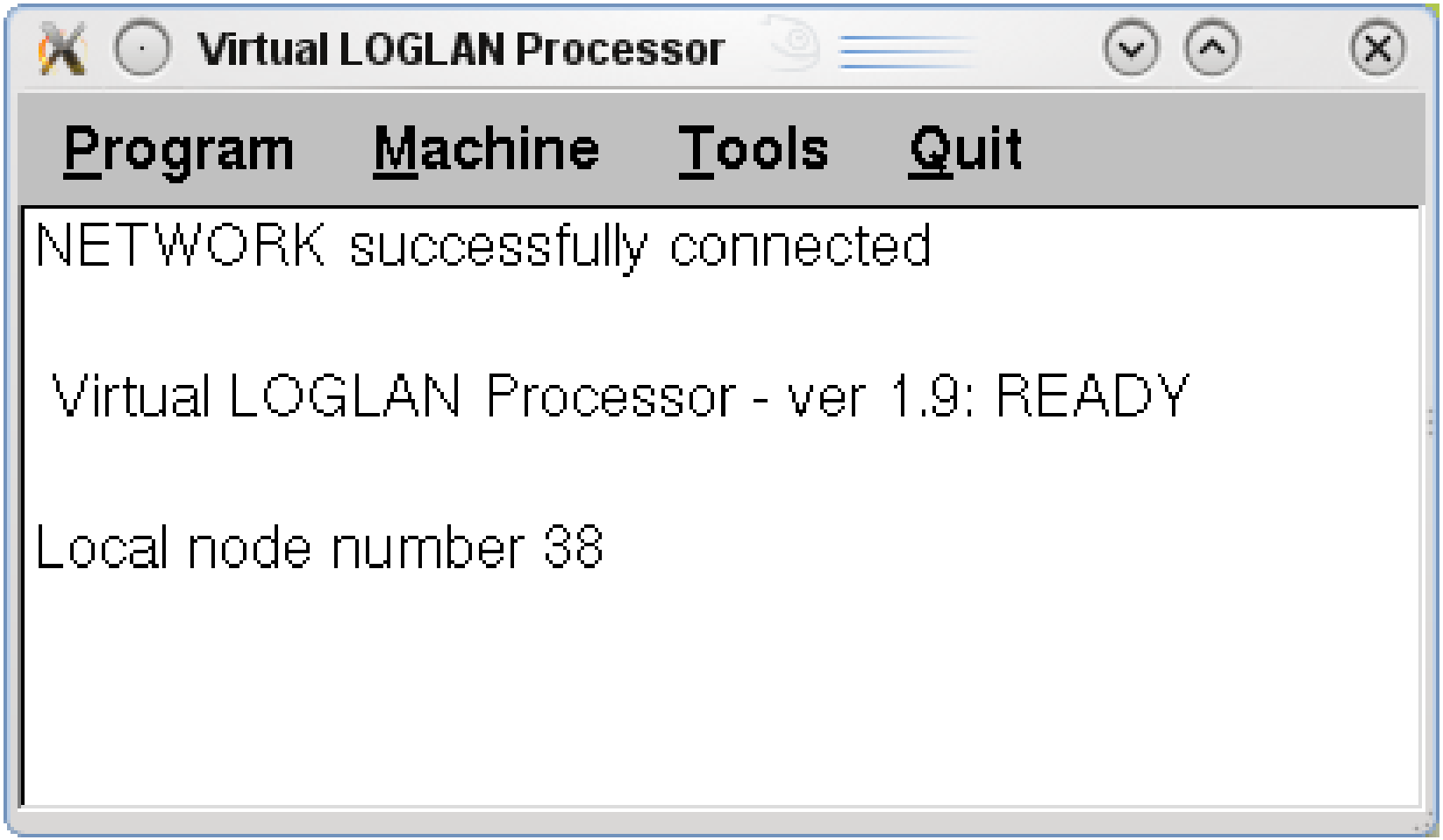
Wstęp

Maszyny wieloprocesorowe są dziś coraz tańsze. Ale mają dwa ograniczenia: jedno, to koszt, drugie to fakt, że potrzeby użytkowników szybko wyczerpują zasoby takiego komputera. Łączenie komputerów siecią po to by osiągnąć wieksze możliwości obliczeń równoległych wydaje się dobrym pomysłem. Istnieje wiele realizacji tego pomysłu, z których najwcześniejsza to PVM - Parallel Virtual Machine. Produkt ten jest wciąż popularny i używany. Inne podejście do pracy w sieciach komputerowych to MPI - Message Passing Interface.

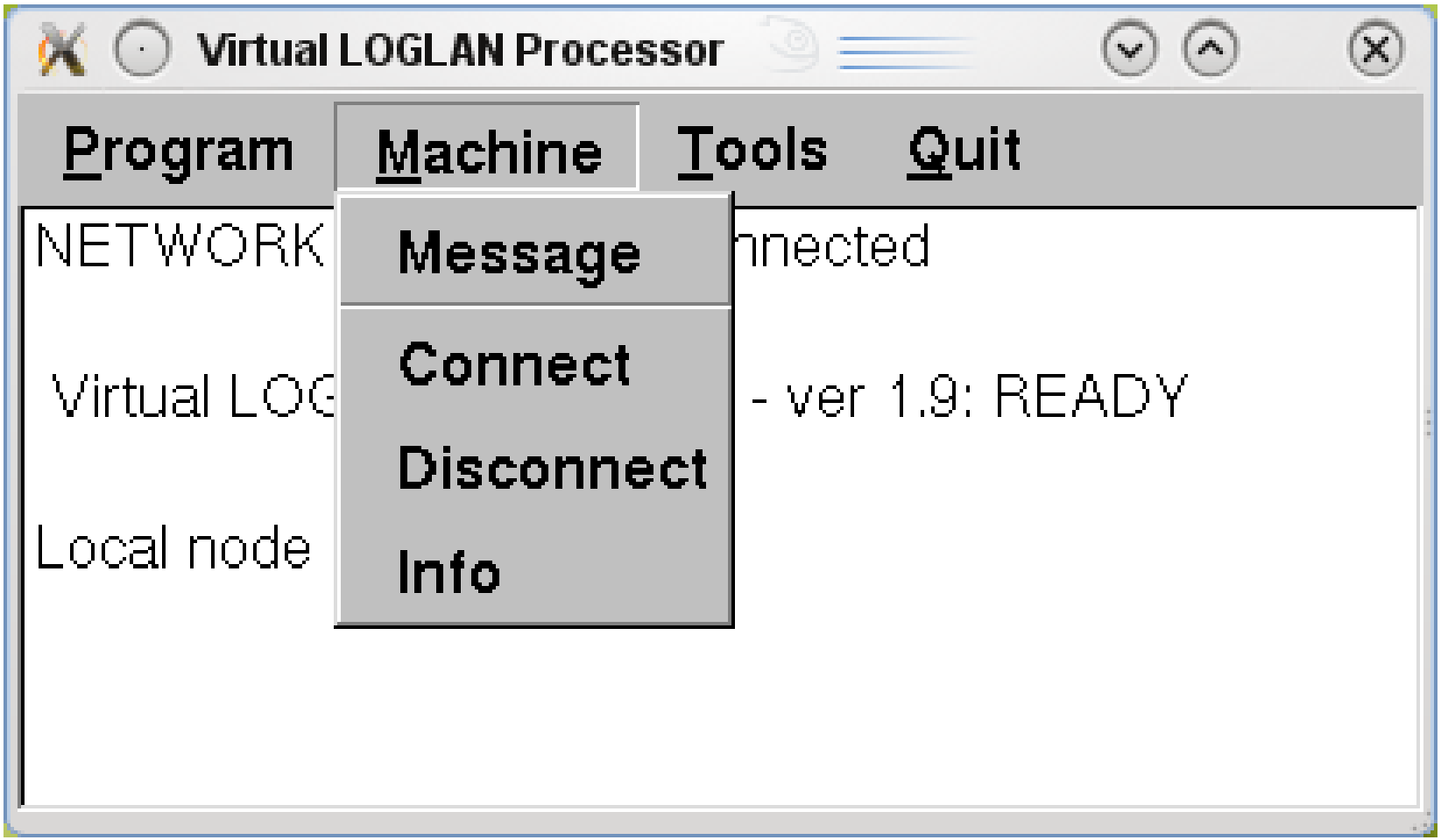
Łączenie Loglanowskich maszyn wirtualnych VLP

1. uruchomienie VLP: wykonaj polecenie `logker` Otworzy się konsola VLP, zobaczysz w niej Node numer.
2. Z menu `Machine`, wybierz polecenie `Connect`. W okienku dialogowym podaj numer IP komputera na którym działa inna maszyna VLP - zatwierdź! Możesz sprawdzić status rozproszonego (równoległego) loglanowskiego komputera wirtualnego PLVM wydając polecenie `Info`. Powinieneś zobaczyć listę maszyn VLP połączonych z Twoją maszyną VLP.
   Sprawdź czy każda z tych maszyn ma inny numer Node.
3. Teraz programy (loglanowskie) wykonywane przez Ciebie mogą alokować obiekty aktywne na jakimkolwiek procesorze - trzeba podać numer maszyny VLP = Node Number.
4. Możesz użyć polecenia `Disconnect`. *Ostrożnie!*.
5. Możesz wysłać "SMS" do innego VLP: tj. przesłać `Message` - krótki tekst.

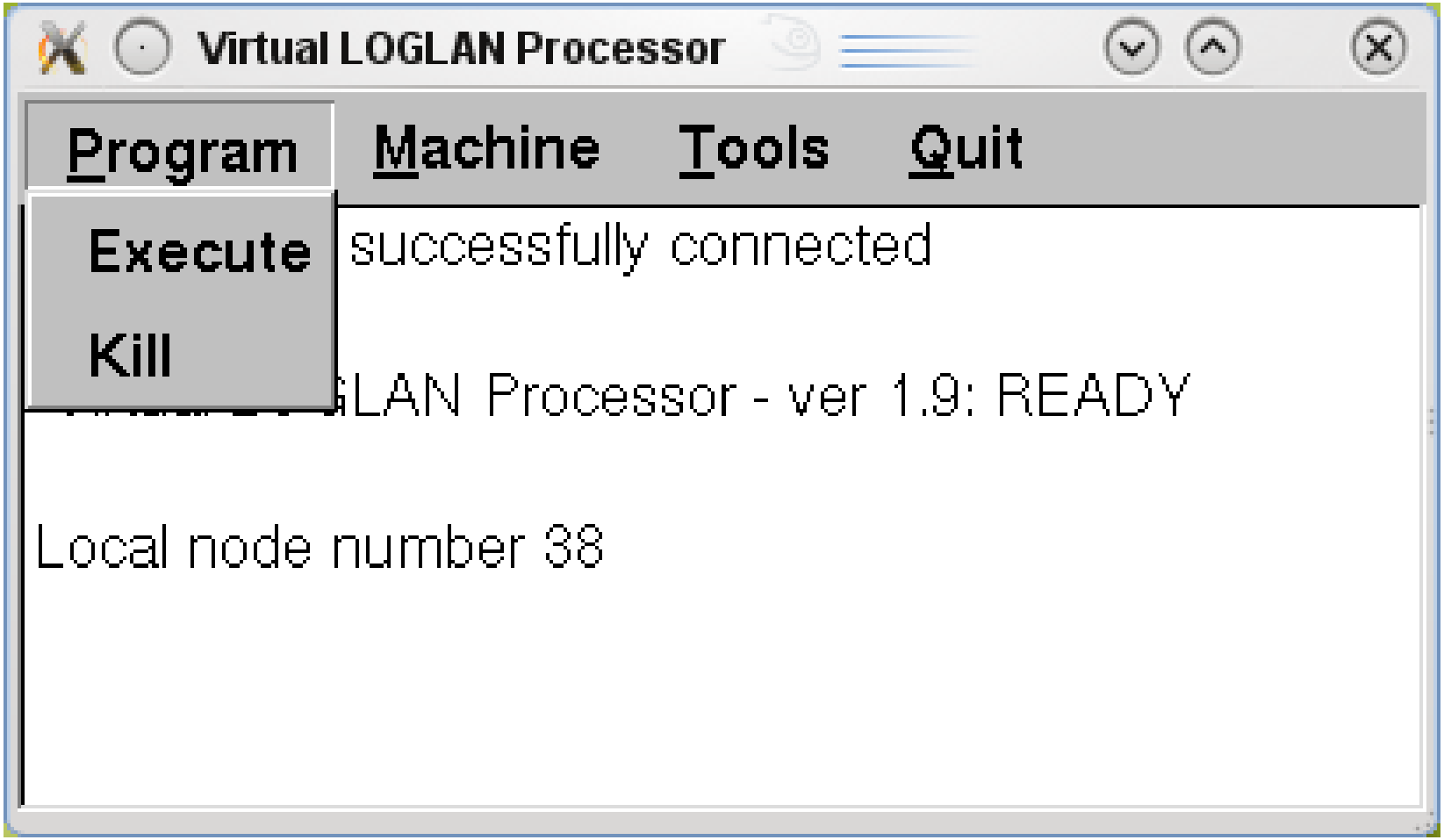
Konsola VLP I



Konsola VLP II



Konsola VLP III



Praca w PVM i MPI

Polecam wprowadzenie i porównanie tych dwu podejść opisane w:
http://linuxgazette.net/issue65/joshi.html

Plan

Obiekty aktywne

Obliczenia i ... dane są rozpraszane w obiektach aktywnych.

1. Obiekt aktywny jest tworzony wg wzorca jakim jest predefiniowana klasa **process**.
2. Obiekt aktywny posiada instrukcje wątku.
3. Wszystkie pola obiektu aktywnego są prywatne.
4. Przy tworzeniu obiektu aktywnego wskazujemy na numer węzła - komputera na którym go alokujemy. W dalszym ciagu posługujemy się tylko odnośnikiem (sieciowym) do tego obiektu aktywnego.
5. Metody obiektu aktywnego mogą dynamicznie zmieniać swój status z private na public i z powrotem.

Proces - wzorzec obiektów aktywnych

Wyróżniony rodzaj klas - proces pozwala zadeklarować wzorce wg których tworzone będą obiekty aktywne.

Składnia procesu

unit A : {B} **process**(parametry) <*deklaracje lokalne*> **begin** <*konstruktor*> **return**; <*wątek*> **end** A	*nagłówek, proces A dziedziczy z* klasy B! *wielkości lokalne są niedostępne z zewnątrz* *te instrukcje inicjalizują obiekt aktywny* *oddziela instr. inicjalizacji od wątku* *tu umieszczamy instrukcje wątku*

Tworzenie obiektu aktywnego

Niech p będzie zmienną typu A.

```
var p:  A;
```

Wykonaj instrukcję przypisania

```
p:=new A(nodeNr,<inneparametry>);
```

Utworzony obiekt będzie umieszczony na wirtualnym komputerze loglanowskim o numerze `NodeNr`.
`inneparametry` będą wykorzystane podczas inicjalizacji tego obiektu.
Zmienna p będzie wskazywać na nowoutworzony obiekt. Obiekt p na razie znajduje sie w stanie *Pasywny*.
Zmienna p może być parametrem aktualnym przekazywanym podczas obcego wołanie procedury (ang. alien call) do innego obiektu aktywnego, na innym być może komputerze. Wartościa p na tym drugim komputerze jest ten sam obiekt aktywny. Mówimy o referencji sieciowej.

Ograniczenia i rozszerzenia

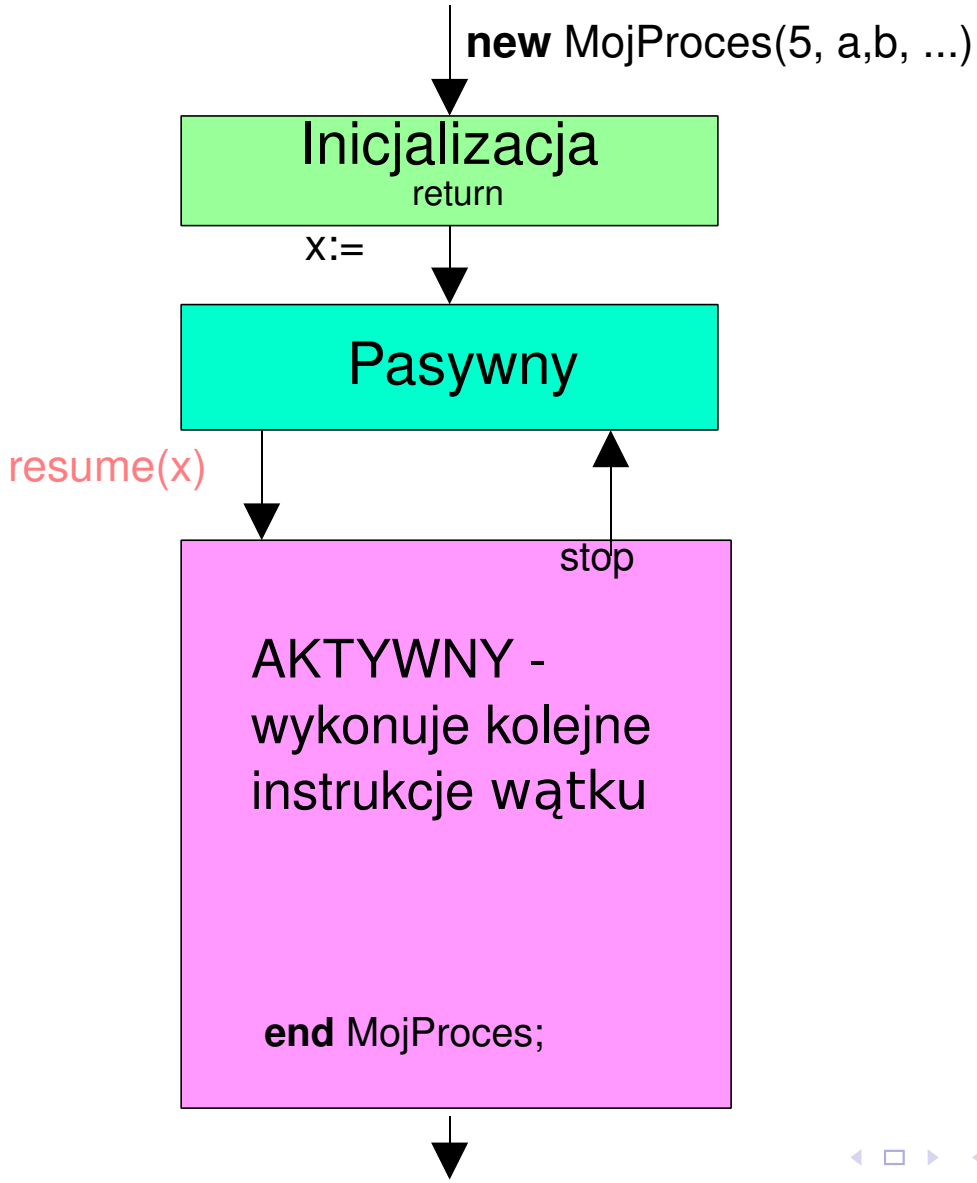
- Pierwszy parametr musi być typu integer, jego wartość określa nr procesora na którym będzie alokowany obiekt aktywny np.
  aa:= new A(0, xxx) - obiekt aa będzie na tym samym procesorze co wątek wykonujący to polecenie (współbieżność)
  b:= new A(3, yyy) - obiekt b będzie na procesorze nr 3 (rozproszenie).
- Proces może dziedziczyć z modułu B - klasy lub procesu, tzn. może go rozszerzać.
- Deklaracje lokalne i parametry lokalne obiektów a lub b, nie są widoczne z zewnątrz przez zdalny dostęp. Np. wyrażenie a.x jest błędne.
  Można zdalnie wywoływać metody. Ale semantyka tych poleceń jest opisana protokołem "alien call" tzw. obce wołanie metody. Zob. poniżej.

Plan

Tworzenie, alokacja i uruchamianie

Obiekty aktywne tworzymy wykonując instrukcję
aa := new A(*nr*, <inne parametry>);
Tworzony jest wtedy obiekt o nazwie aa, alokowany na maszynie *nr*. Obiekt ten jest pasywny. Można uruchomić wykonywanie wątku instrukcji w tym obiekcie wykonując polecenie
resume(aa);

Stany obiektu



Opis scenariusza

Po utworzeniu nowy obiekt aktywny jest w stanie pasywnym, nie wykonuje żadnych instrukcji. Pewien obiekt aktywny może go uaktywnić wykonując polecenie resume(c) Warunkiem jest posiadanie odnosnika do obiektu, zwykle posiada go obiekt aktywny w którym wykonano polecenie “c := new MojProces(parametry-Aktualne);"
Od tego momentu instrukcje obiektu aktywnego c wykonują się współbieżnie z instrukcjami innych wątków.
Proces c może zawiesić swój wątek - instrukcja ”stop“.

Ograniczenia

Parametrami aktualnymi nie mogą byc obiekty klas, procedury lub funkcje. Mogą nimi być wielkości typów pierwotnych: integer, real, boolean, string, char
oraz
obiekty aktywne procesów, a także metody procesów (procedury lub funkcje).

Maska

Niech p będzie obiektem aktywnym typu A. Każdy obiekt aktywny ma predefiniowany atrybut MASK - maska. Wartościami tego atrybutu są podzbiory zbioru metod obiektu, zadeklarowanych w procesie A.
Początkową wartością maski jest zbiór pusty $MASK = \emptyset$.
Instrukcje *enable* i *disable* zmieniają stan maski obiektu aktywnego.
`enable(m1, m2);` // $MASK := MASK \cup \{m1, m2\}$
`disable(m2,m3);` // $MASK := MASK \setminus \{m2, m3\}$
Stan maski dyktuje czy obcy proces tj. obiekt aktywny może zdalnie wezwać metodę danego obiektu.

Plan

alien call – składnia

Składnia obcego wołanie metody jest taka sama jak składnia wołania metody w obiekcie zwykłej klasy:

```
call P.meth(parametry_aktualne)
```

Poczym więc poznajemy która z dwu instrukcji `call A.procedura1(...)` i `call B.procedura2(...)` jest instrukcją obcego wołania procedury?

Odpowiedź: po typie wyrażenia `A`, odpowiednio `B`. Jeśli typ wyrażenia `A` jest procesem to jest to instrukcja obcego wołania procedury.

Alien call – nieformalny opis semantyki

Obce wołanie metody nie jest:

- wywoływaniem metody w zwykłym tj nieaktywnym obiekcie jakiejś klasy,
- wywoływaniem wg protokołu RPC firmy Sun
- wywoływaniem RMI (Java RMI Remote Method Invocation)
- opisane w architekturze CORBA – klient-serwer.

Czym więc jest alien call?
Jest protokołem w którym dwa obiekty aktywne WSPÓLNIE wykonują metodę opisaną w obiekcie wzywanym na prośbę i z argumentami obiektu wzywanego.
Jest to protokół w pełni obiektowy: opisany w terminach obiektu aktywnego, metody obiektu, argumentów metody i MASKi – systemowego atrybutu obiektów aktywnych.

Alien call – opis protokołu

Niech aa będzie obiektem aktywnym typu `A`. Obiekt aktywny przyjmie zlecenie wykonania swej metody *m* od innego obiektu aktywnego call aa.m(...) jeśli:
jest w stanie AKTYWNY i metoda *m* znajduje się w masce. W przeciwnym przypadku obiekt wzywający będzie oczekiwać.

Obce wołanie metody meth – przypadek asynchroniczny

w procesie P	*w procesie Q*
–	`enable meth;`
`call a.meth(params)`	`instr1`
–	`instr2`
–	–
–	`disable meth;`

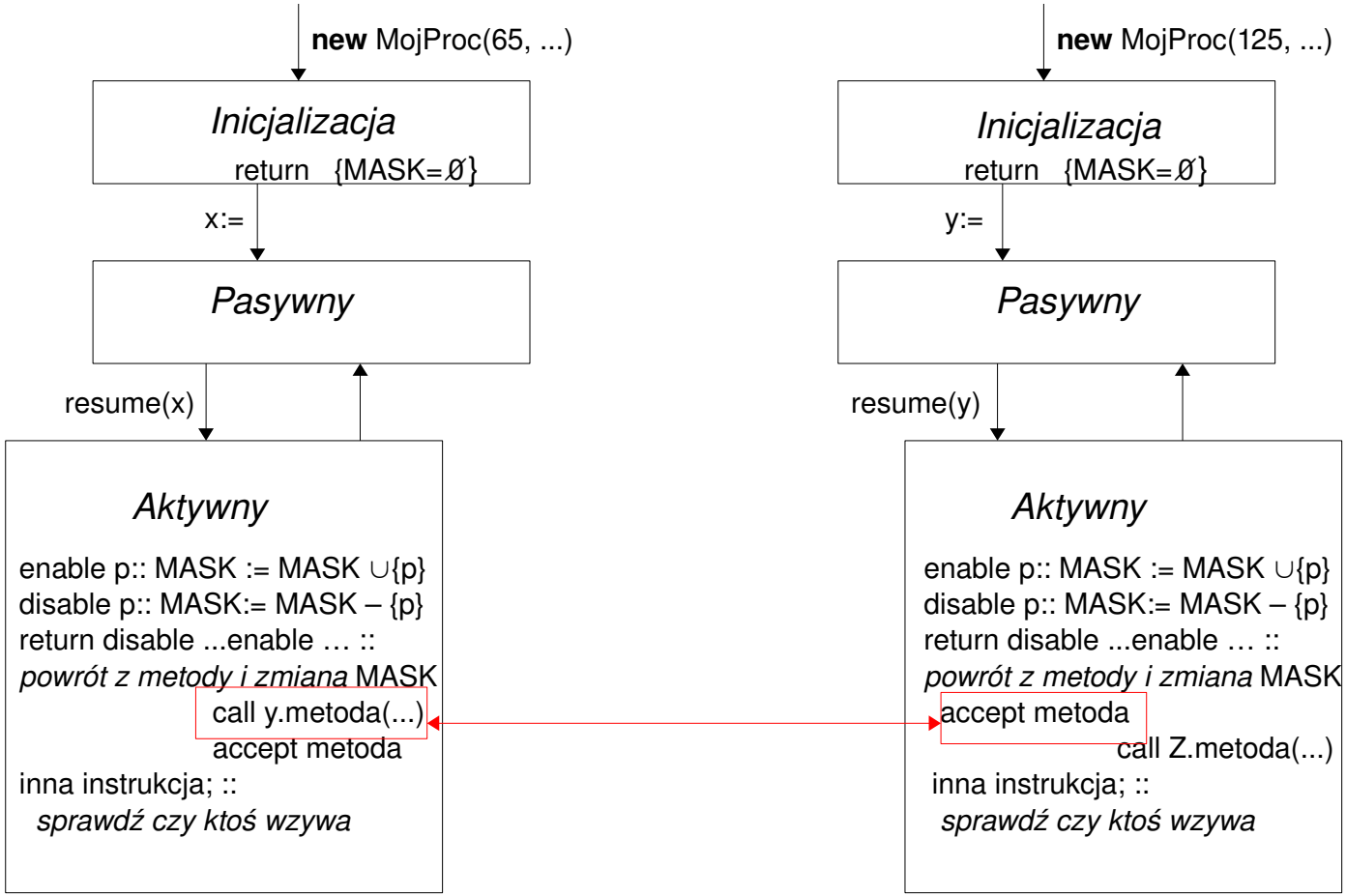
Objaśnienie.
Metoda meth znajduje sie w MASCe procesu Q od wykonania instrukcji `enable meth;` do wykonania instrukcji `disable meth;`
Jeśli w tym czasie proces P zacznie wykonywać instrukcję `call a.meth(params)` to dojdzie do **przerwania** wykonywania ciągu instrukcji procesu Q, wykonania instrukcji procedury call meth w procesie Q,

Obce wołanie metody meth – przypadek synchroniczny

w procesie P	*w procesie Q*
–	–
`call a.meth(params)`	`instr1`
–	`accept meth`
–	–
–	–

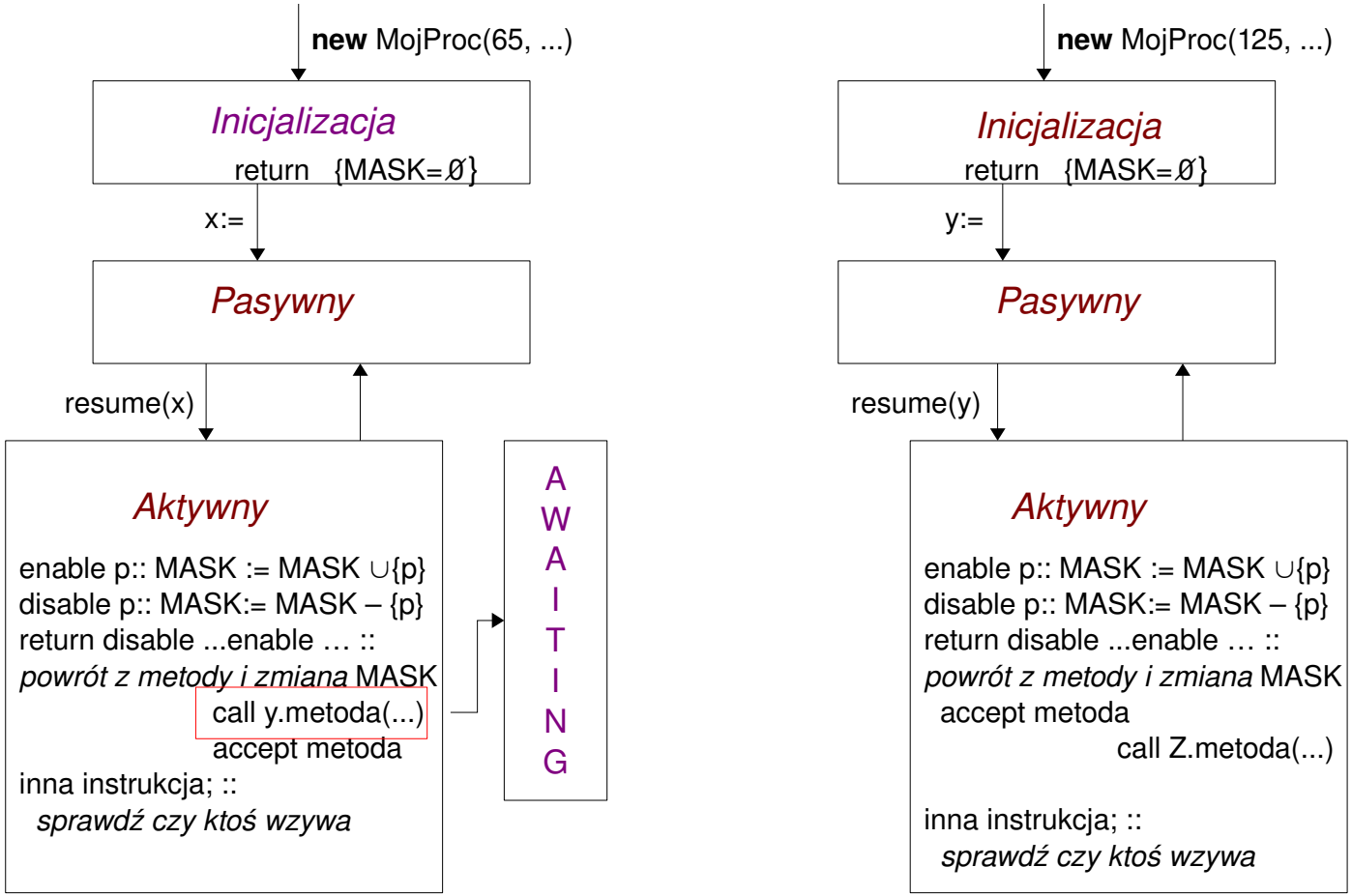
Objaśnienie.
Proces Q wstawia metodę meth do MASKi i oczekuje na to by jakiś proces wykonał instrukcję obcego wołania procedury meth.
Jeśli proces P zacznie wykonywać instrukcję `call a.meth(params)` to dojdzie do **spotkania** dwu obiektów aktywnych P i Q i w konsekwencji do wspólnego wykonania instrukcji procedury call meth w procesie Q,

Protokół alien call – cz. 1



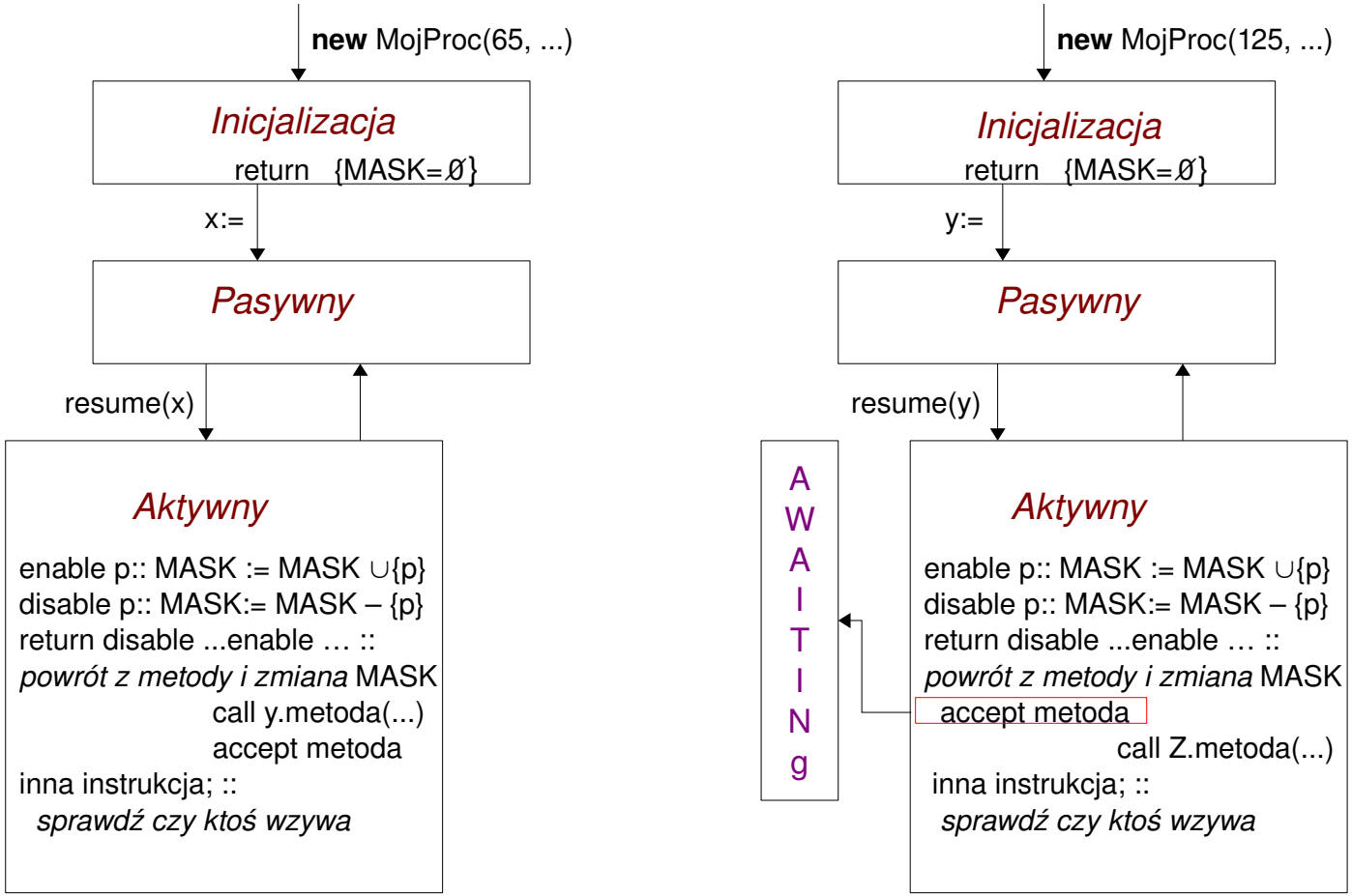
Rys. 1 Spotkanie instrukcji call y.metoda(...) wątku x z instrukcją accept metoda w wątku y.

Protokół alien call – cz. 2



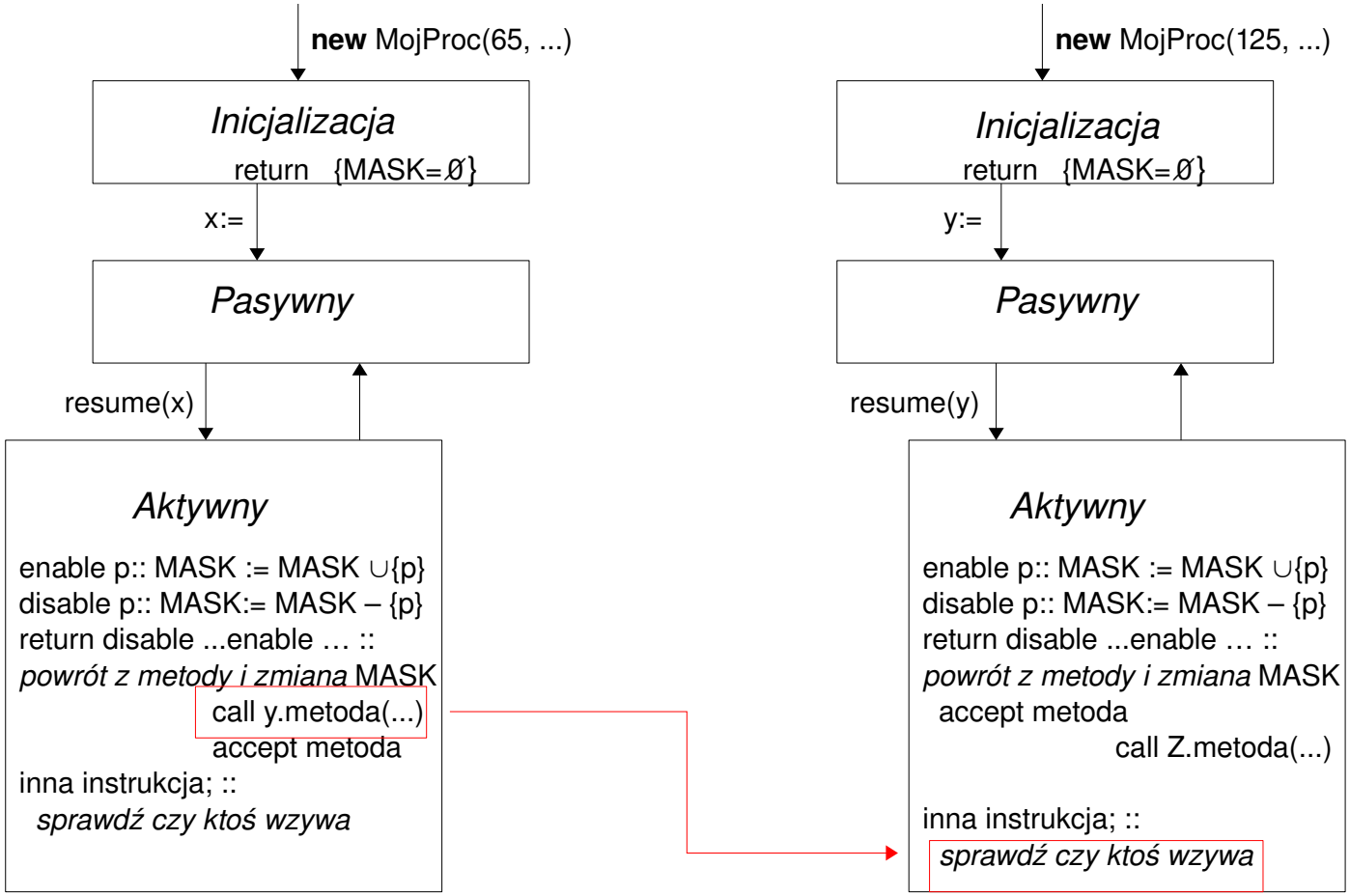
Rys/ 2 Gdy metoda $\notin$ MASK wątku y, to wątek x oczekuje

Protokół alien call – cz. 3



Rys. 3 Gdy żaden wątek nie wykonał jeszcze instrukcji „call y.metoda;" to wątek y oczekuje,
Na czas wykonania instrukcji accept: MASK wątku y := MASK∪ {metoda}

Protokół alien call – cz. 4



Rys. 4 Gdy metoda ∈ MASK wątku y, to wątek y przerywa swe obliczenia i wykonuje metoda(...) jako usługę dla watku x

wspólna część protokołu – spotkanie

Gdy dojdzie do spotkania dwu obiektów aktywnych w celu wspólnego wykonania metody:

a) obiekt wzywany zapamietuje swoją MASKę,

b) MASKa jest zerowana, $MASK := \emptyset$,

c) obiekt wzywany odbiera argumenty i wykonuje metodę,

d) po zakończeniu metody MASKa jest odtwarzana

e) wykonywana jest instrukcja return lub return disable p,q enable r,

f) obiekt wzywający odbiera wyniki, tj. parametry przekazywane dla out

g) wątki obiektów rozchodzą się

Plan

Anegdota o Lwowie i o żywej gęsi

Zadanie

Czy możliwa jest realizacja protokołu alien call w Javie (lub w C#)? Odpowiedź pozytywna lub negatywna zostanie nagrodzona skrzynką niezłego wina. Odpowiedź pozytywna ma mieć postac oprogramowania realizującego protokół, wraz z analizą stwierdzającą poprawność implementacji. Odpowiedż negatywna ma miec postać stwierdzenia popartego niepodważalnymi argumentami.

Literatura

mój artykuł w TiTI z r. 2008 o alien call