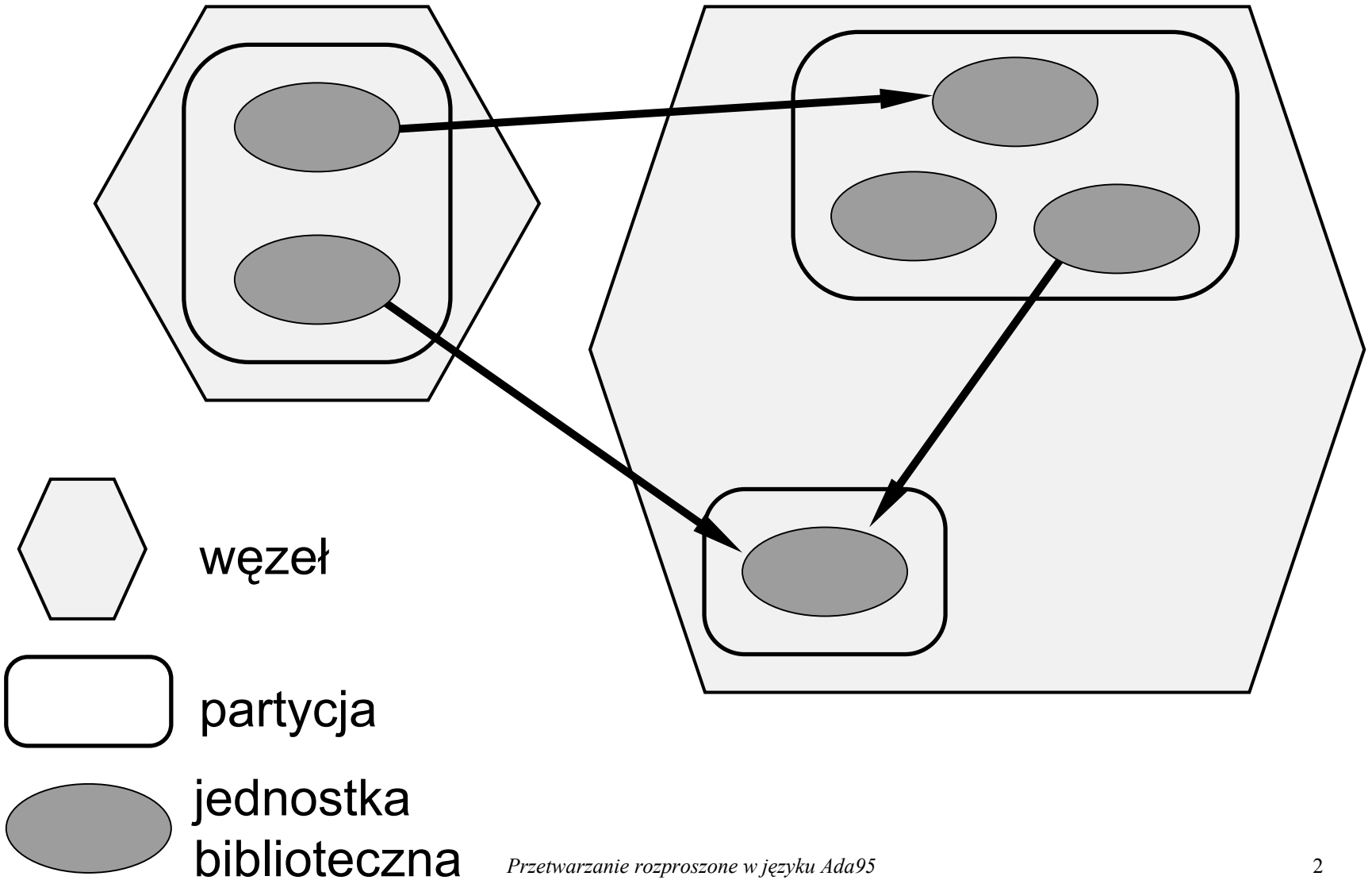
Ada95 — przetwarzanie rozproszone

1. Model systemu rozproszonego
2. Partycje i jednostki kompilacji
3. Glade
   - narzędzie **`gnatdist`**
   - język opisu konfiguracji
4. Przykład programu rozproszonego

Model systemu rozproszonego



Węzły (ang. nodes)

- Węzły są jednostkami w modelu przetwarzania rozproszonego w Adzie, na których rezydują programy i dane.
- W zależności od przeznaczenia wyróżnia się
  - węzły przetwarzające (ang. processing nodes) — węzły, na których odbywa się przetwarzanie danych i na których mogą być umieszczane partycje aktywne,
  - węzły składnice (ang. storage nodes) — węzły, które składują i udostępniają dane i na których mogą być umieszczane tylko partycje pasywne.

Partycje

- Partycje są jednostkami tworzącymi program rozproszony w Adzie, umożliwiającymi komunikację rozproszoną pomiędzy różnymi węzłami.
- Partycja grupuje jedną lub kilka odpowiednio zaetykietowanych jednostek bibliotecznych.
- Rodzaje partycji:
  - aktywne — rezydują i wykonują się w węzłach przetwarzających i muszą mieć swój wątek sterowania,
  - pasywne — rezydują w węzłach-składnicach lub węzłach przetwarzających i mogą zawierać tylko jednostki zadeklarowane jako **`Pure`** lub **`Shared_Passive`**.

Kategorie jednostek bibliotecznych w partycjach

- **`Pure`** — jednostka opracowana, która nie zmienia się w trakcie wykonywania programu,
- **`Shared_Passive`** — jednostka umożliwiająca współdzielenie danych i programów przez partycje aktywne,
- **`Remote_Types`** — jednostka podobna do `Pure`, ale dopuszcza zdalne typy wskaźnikowe,
- **`Remote_Call_Interface`** — jednostka zawierająca definicję interfejsu do komunikacji pomiędzy aktywnymi partycjami,
- normalna jednostka biblioteczna

Kategoria **Pure**

- ⇨ Jednostka kategorii **Pure** nie może zawierać deklaracji żadnej zmiennej ani definicji typu wskaźnikowego.
- ⇨ W jednostce kategorii **Pure** najczęściej występują definicje stałych, typów i podprogramów, wykorzystywanych w innych jednostkach.
- ⇨ Jednostka kategorii **Pure** może być powielana w wielu partycjach systemu rozproszonego.
- ⇨ Jednostka kategorii **Pure** może pojawić się w kontekście dowolnej innej jednostki (dowolna inna jednostka może być od niej zależna semantycznie).
- ⇨ Jednostka kategorii **Pure** może być semantycznie zależna tylko od innej jednostki kategorii **Pure**.

Kategoria **`Shared_Passive`**

- Obiekty jednostki kategorii **`Shared_Passive`** odwzorowywane są we współdzieloną przestrzeń adresową (np. plik, pamięć).
- Jednostka kategorii **`Shared_Passive`** umożliwia komunikację pomiędzy aktywnymi partycjami poprzez zapis i odczyt deklarowanych w niej obiektów współdzielonych.
- Jednostka kategorii **`Shared_Passive`** może być semantycznie zależna tylko od innej jednostki kategorii **`Pure`** lub **`Shared_Passive`**.
- Jednostka kategorii **`Shared_Passive`** nie może zawierać zadań ani obiektów chronionych z wejściami (alokowane są na partycjach pasywnych, więc nie mają sterowania) oraz wywołań zdalnych.

Kategoria **`Remote_Types`**

- ⇨ W porównaniu z jednostką kategorii **`Pure`** jednostka kategorii **`Remote_Types`** umożliwia definiowanie zdalnych typów wskaźnikowych.
- ⇨ Zdalny typ wskaźnikowy obejmuje identyfikator węzła oraz lokalizację wskazywanego obiektu na tym węźle (ang. fat pointer).
- ⇨ Jednostki kategorii **`Remote_Types`** wykorzystywane są najczęściej do definiowanie zdalnych (rozproszonych) obiektów dostępnych przez wskaźniki oraz operacji na tych obiektach.
- ⇨ Jednostka kategorii **`Remote_Types`** może być semantycznie zależna tylko od innej jednostki kategorii **`Pure`**, **`Shared_Passive`** lub **`Remote_Types`**.

Kategoria **`Remote_Call_Interface`**

- ⇨ Jednostka kategorii **`Remote_Call_Interface`** udostępnia zdalnie wywoływane podprogramy.
- ⇨ Przy wywołaniu zdalnego podprogramu obowiązuje semantyka *co najwyżej raz* (ang. at most once).
- ⇨ Jednostka kategorii **`Remote_Call_Interface`** może też udostępniać zdalne typy wskaźnikowe.
- ⇨ Jednostka kategorii **`Remote_Call_Interface`** może być semantycznie zależna od innej jednostki kategorii **`Pure`**, **`Shared_Passive`**, **`Remote_Types`** lub **`Remote_Call_Interface`**.

Asynchroniczne wywołania zdalne

- ⇨ Wywołanie asynchroniczne występuje wówczas, gdy wywoływany zdalny podprogram (procedura) wskazana jest po stronie serwera jako asynchroniczna za pomocą deklaracji **`pragma Asynchronous`** (*nazwa procedury*)
- ⇨ Procedura asynchroniczna może mieć tylko parametry wejściowe.
- ⇨ Nie może być asynchronicznej funkcji.
- ⇨ Wszystkie wyjątki zgłaszane podczas wykonania procedury asynchronicznej są gubione.

Wiązanie

- Wiązanie statyczne — na etapie kompilacji, gdy odwołujemy się do konkretnego obiektu (procedury) zdalnego, specyfikując odpowiednio jego nazwę w programie.
- Wiązanie dynamiczne — na etapie wykonania, gdy posługujemy się wskaźnikiem do zdalnego obiektu (procedury), którego wartość ustalana jest dopiero w czasie wykonania.

Narzędzie **`gnatdist`**

- Aneks E (DSA) specyfikacji Ada95 Reference Manual nie precyzuje wymagań odnośnie narzędzia do budowy aplikacji rozproszonych, pozostawiając swobodę decyzji w zakresie rozwiązań projektantom.
- Narzędzie **`gnatdist`** jest jednym z rozwiązań, stanowiącym część pakietu w Glade.
- Narzędzie **`gnatdist`** służy do budowania aplikacji na podstawie odpowiednio zaetykietowanych jednostek bibliotecznych oraz opisu konfiguracji tych jednostek (ich przydziału do poszczególnych węzłów).
- Język opisu konfiguracji oparty jest na składni języka Ada i obejmuje deklaracje poszczególnych partycji oraz definicje różnych atrybutów.

Narzędzie **`gnatdist`** — sposób użycia

1. przygotowanie odpowiednio zaetykietowanego programu,
2. ewentualne zbudowanie aplikacji zwartej i jej przetestowanie,
3. przygotowanie opisu konfiguracji w pliku z rozszerzeniem **`cfg`** (np. **`conf.cfg`**),
4. Uruchomienie narzędzia **`gnatdist`** np.:

```
gnatdist conf.cfg
```

Język opisu konfiguracji

- Słowa kluczowe języka Ada 95 są zastrzeżone (nawet gdy nie są używane w języku opisu konfiguracji)
- Nowe (w porównaniu z językiem Ada 95) słowa kluczowe: **`configuration`**, **`Partition`**, **`Channel`**.
- Konfiguracja musie mieć swoją nazwę, która jest zgodna z nazwą pliku zawierającego jej opis.
- **`Partition`** i **`Channel`** są traktowane jako nazwy typów danych.

Deklaracja konfiguracji

`configuration` *nazwa* **`is`**

część deklaracyjna

`begin`

ciąg instrukcji

`end` *nazwa* **`;`**

Składowe części deklaracyjnej

- ⇨ deklaracja partycji
- ⇨ deklaracja kanału komunikacyjnego pomiędzy partycjami
- ⇨ deklaracja podprogramu
- ⇨ klauzula reprezentacji
- ⇨ pragma

Deklaracja partycji

nazwa `: Partition;`

nazwa1`,` *nazwa2* `: Partition;`

nazwa `: Partition :=(`*jednostka1*`,` *jednostka2*`);`

Atrybuty partycji

- **`Host`** — statyczna lub dynamiczna specyfikacja nazwy maszyny, na której ma być uruchomiona dana partycja.
- **`Main`** — specyfikację alternatywnej procedury głównej.
- **`Storage_Dir`** — specyfikacja katalogu z binariami.
- **`Termination`** — specyfikacja sposobu zakończeniu działania danej partycji (**`Global_Termination`** — domyślne, **`Local_Termination`**).
- **`Reconection`** — specyfikacja sposobu reakcji na awarie partycji.
- **`Task_Pool`** — specyfikacja stopnia współbieżności obsługi żądań.
- **`Command_Line`** — argumenty linii poleceń przekazywane do danej partycji przy uruchamianiu.

Specyfikacja nazwy maszyny

nazwa `: Partition;`

`for` *nazwa* `'Host use` *nazwa_maszyny*`; --` statyczna

`function` *nazwa_funkcji* `(s: in String)`
`return String;`

`for` *nazwa* `'Host use` *nazwa_funkcji*`; --` dynamiczna

Specyfikacja procedury głównej

nazwa_partycji **: Partition;**

-- zasadnicza procedura główna

procedure *nazwa_procedury* **is in** *nazwa_partycji***;**

-- alternatywna procedura główna

for *nazwa_ partycji* **'Main use** *nazwa_ procedury***;**

Specyfikacja katalogu z binariami

nazwa`: Partition;`

`for` *nazwa* `'Storage_Dir use` *ścieżka*`;`

Specyfikacja sposobu zakończeniu

nazwa **: Partition;**

for *nazwa*’**Termination use**
Global_Termination;

for *nazwa*’**Termination use**
Local_Termination;

for *nazwa*’**Termination use**
Defered_Termination;

Specyfikacja sposobu reakcji na awarie partycji

nazwa `: Partition;`

`for` *nazwa*`'Reconection use`
`Reject_On_Restart; --` brak restartu

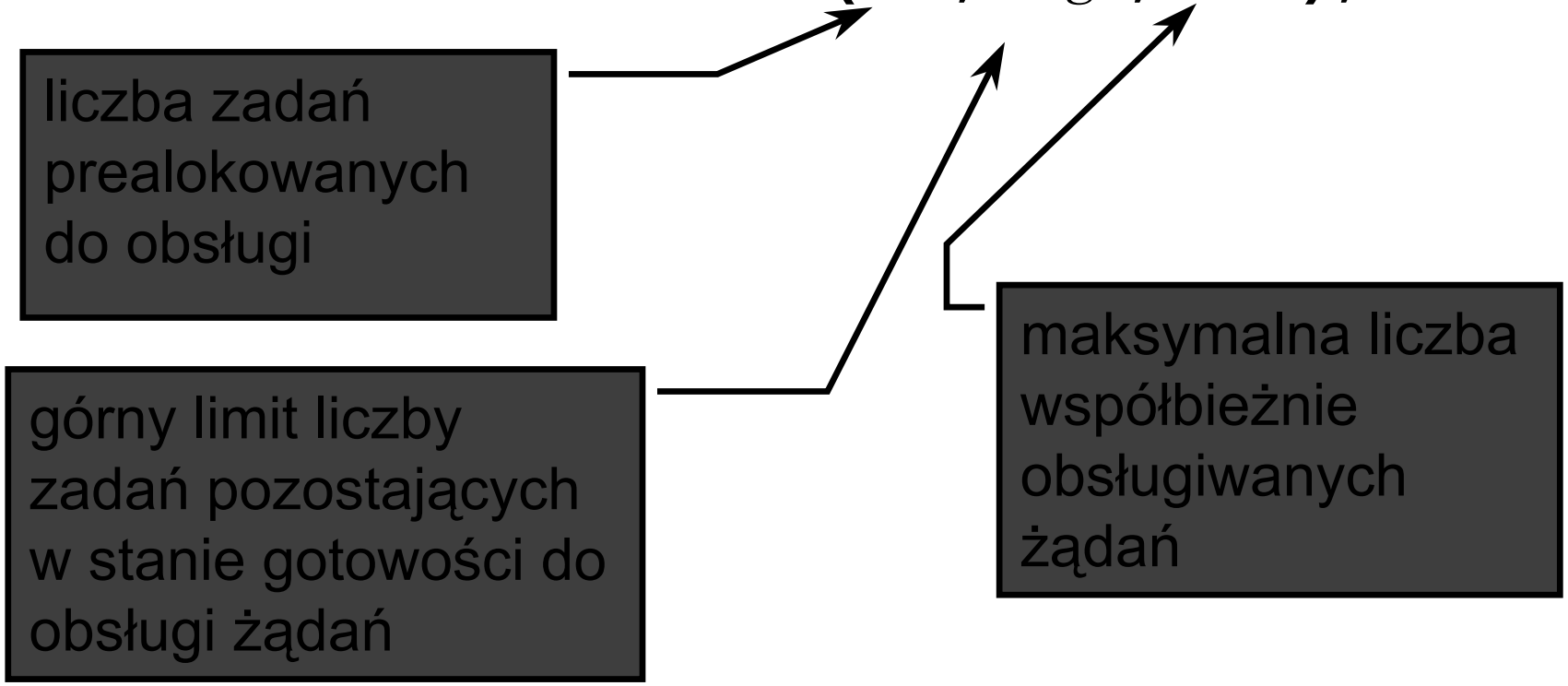
`for` *nazwa*`'Reconection use`
`Fail_Until_Restart; --` odrzucanie wywołań do momentu restartu

`for` *nazwa*`'Reconection use`
`Wait_Until_Restart; --` zawieszenie wywołań do momentu restartu

Specyfikacja stopnia współbieżności obsługi żądań

nazwa **: Partition;**

for *nazwa*'**Task_Pool use (***min***,** *high***,** *max***);**



Deklaracja kanału komunikacyjnego i specyfikacja atrybutu **`Filter`**

nazwa **`: Channel;`**

nazwa1`,` *nazwa2* **`: Channel;`**

nazwa **`: Channel :=(`***partycja1*`,` *partycja2*`);`**

`for` *nazwa*`'`**`Filter use "ZIP";`**

Specyfikacja **`pragma`**

- **`Starter`** — sposób uruchamiania zadań
  - Ada — procedura startująca w języku Ada,
  - Shell — wykorzystanie skryptu shell'a,
  - None — uruchamianie ręczne,
- **`Boot_Location`** — specyfikacja węzła startowego wg. standardu `<protocół>://<dane_protokołu>` np. `tcp://localhost:5557`.
- **`Import`** — importowanie skryptów, jako procedur głównych programu lub procedur do określania hostów

pragma Starter

```
pragma Starter (None);
pragma Starter (Ada);
pragma Starter (Shell);
```

pragma Boot_Location

```
pragma Boot_Location (
      "tcp",
      "localhost:5557");
```

Przykład specyfikacji jednostki kategorii **Remote_Call_Interface**

```
package Server is

   pragma Remote_Call_Interface;

   type Back_Reference is access procedure (
                                ret : Positive);

   procedure  F1 (a,b : Positive;
                       bptr : Back_Reference);
   pragma asynchronous (F1);

end Server;
```

Przykład specyfikacji pakietu klienta (z zamiarem udostępniania procedury *call back*)

```
package Client is

   pragma Remote_Call_Interface;

   procedure Call_Back(val : Positive);

end Client;
```

Implementacja pakietu klienta

```
with Text_IO; use Text_IO;

package body Client is

  procedure Call_Back(val : Positive) is
  begin
      Put_Line("Client: Got the result
                  :"&Integer'Image(val));
  end Call_Back;

end Client;
```

Przykład procedury głównej

```
with Text_IO; use Text_IO;
with Server; use Server;
with Client; use Client;

procedure start is
  i : Integer;
begin
  Put_Line("Calling service");
  Server.F1(3,4, Call_Back'Access);
  Put_Line("Call done...");
  for i in Integer range 1..20 loop
      Put_Line("..."); Delay(0.5);
  end loop;
  Put_Line("Client tarminated...");
end start;
```

Przykład opisu konfiguracji

```
configuration CallBack is
  pragma Starter (None);
  -- programs are launched manually.

  pragma Boot_Server ("tcp",
                      "localhost:5556");
  Msg_Client: Partition := (Client);
  Msg_Server: Partition := (Server);
  Procedure start is in Msg_Client;
end CallBack;
```