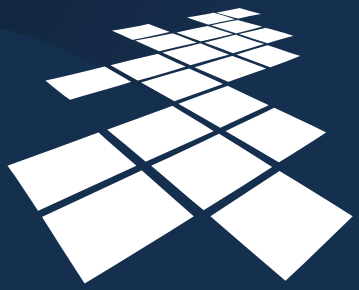
Systemy rozproszone

Komunikacja grupowa

Marek Libuda
Jerzy Brzeziński
Cezary Sobaniec



UCZELNIA
ONLINE

Wykład omawia teoretyczne podstawy komunikacji grupowej – gałęzi przetwarzania rozproszonego zajmującej się systemami, w których osłabia się założenie o stałej liczbie procesów uczestniczących w przetwarzaniu. Procesy organizowane są w dynamicznie zmieniające się grupy, a wszelka komunikacja odbywa się z uwzględnieniem postrzeganego przez procesy stanu grupy.

Pierwsza część wykładu prezentuje założenia oraz specyfikacje związane z własnościami usługi członkostwa, natomiast w drugiej przedstawione są wybrane algorytmy rozsyłania (ang. *multicast*), powszechnie stosowane w komunikacji grupowej.

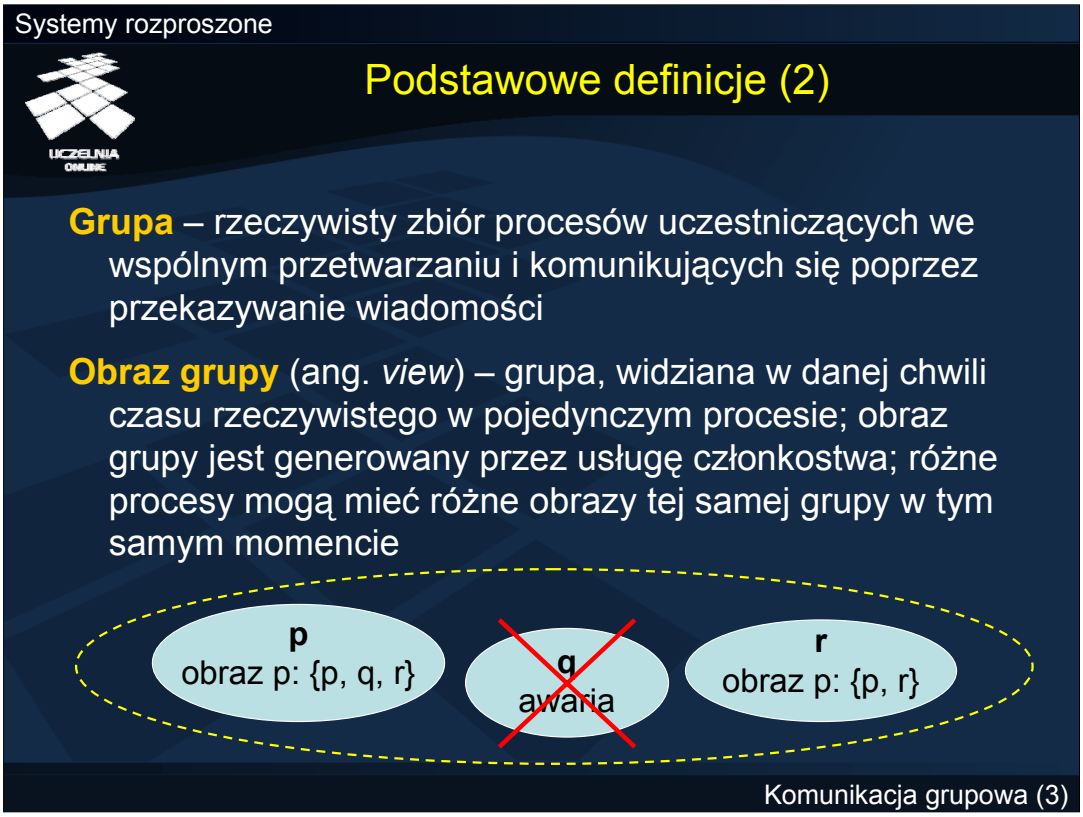
Podstawowe definicje (1)

- Komunikacja grupowa (ang. *group communication*) to mechanizm umożliwiający rozsyłanie (ang. *multicast*) poprzez organizowanie procesów (szerzej – obiektów) w grupy
- Obejmuje dwa aspekty: zarządzanie grupami procesów (usługa członkostwa, ang. *membership service*) oraz algorytmy (niezawodnego) rozsyłania wiadomości w grupie
- Pozwala modelować niezawodną komunikację procesów przy założeniu, że zbiór procesów dynamicznie się zmienia

Komunikację grupową definiuje się jako mechanizm umożliwiający procesom systemu rozproszonego organizowanie się w sposób dynamiczny w grupy i niezawodną wymianę wiadomości w ramach grupy. Za każdy z tych dwóch aspektów odpowiada odrębny mechanizm: usługa członkostwa zajmuje się zarządzaniem składem grupy (dynamiczne dołączanie i odłączanie procesów z grupy), a usługa rozsyłania dostarcza mechanizmu rozgłaszania wiadomości pomiędzy procesami w grupie zgodnie z przyjętym poziomem niezawodności.

Od strony teoretycznej komunikacja grupowa służy jako narzędzie do modelowania komunikacji o podwyższonym poziomie niezawodności (w skrócie - komunikacji niezawodnej) pomiędzy procesami przy zmieniającym się zbiorze uczestniczących w przetwarzaniu procesów.

Systemy komunikacji grupowej GCS (od nazwy angielskiej *Group Communication System*) powszechnie wykorzystuje się w aplikacjach, które potrzebują skorzystać na niższym poziomie z mechanizmów łączenia procesów w grupy oraz niezawodnej komunikacji pomiędzy procesami w grupie. Wśród przykładowych zastosowań można wymienić: systemy konferencyjne, gry sieciowe czy systemy o podwyższonej niezawodności, wykorzystujące zwielokrotnianie (ang. *replication*).



Model systemu zakłada, że procesy organizowane są w grupy oraz że komunikacja pomiędzy procesami ma charakter grupowy.

Grupa to rzeczywisty zbiór procesów, uczestniczących w danej chwili we wspólnym przetwarzaniu (np. obliczenia inżynierskie, systemy medyczne czy militarne) i komunikujących się poprzez przekazywanie wiadomości (ang. *message passing*). **Obraz grupy** to z kolei stan grupy widziany w danej chwili czasu rzeczywistego w procesie.

Obraz grupy może się znacząco różnić od rzeczywistej grupy, a ponadto obrazy grupy mogą być różne w różnych procesach. W przykładzie na rysunku, proces *q* ulega awarii, co z punktu widzenia komunikacji grupowej oznacza opuszczenie grupy. W obrazie procesu *r* proces *q* opuścił już grupę, zaś w obrazie procesu *p* – jeszcze nie, a przez to obrazy procesów *p* i *r* są różne.

Należy podkreślić, że *z punktu widzenia aplikacji nie rozróżnia się awarii procesu od jawnego opuszczenia przez niego grupy* – obydwa te przypadki są aplikacji przedstawiane po prostu jako opuszczenie grupy.

Model matematyczny

Zbiory:

- $\mathcal{P}$ – zbiór procesów w systemie
- $\mathcal{M}$ – zbiór wiadomości przesyłanych w systemie
- $\mathcal{VID}$ – zbiór identyfikatorów obrazów grup

Elementarne zdarzenia:

- **wyślij**(*p*, *m*), $p \in \mathcal{P}$, $m \in \mathcal{M}$
- **odbierz**(*p*, *m*), $p \in \mathcal{P}$, $m \in \mathcal{M}$
- **zmiana_obrazu**(*p*, <*id*, *członkowie*>), $p \in \mathcal{P}$, $id \in \mathcal{VID}$, *członkowie* $\in 2^{\mathcal{P}}$
- **awaria**(*p*), $p \in \mathcal{P}$
- **powrót** (*p*), $p \in \mathcal{P}$

W definicjach specyfikacji wystąpią następujące zbiory:

P – zbiór procesów w systemie;

M – zbiór wiadomości przesyłanych w systemie;

VID – zbiór identyfikatorów obrazów grup.

Wyróżnia się następujące elementarne zdarzenia w systemie:

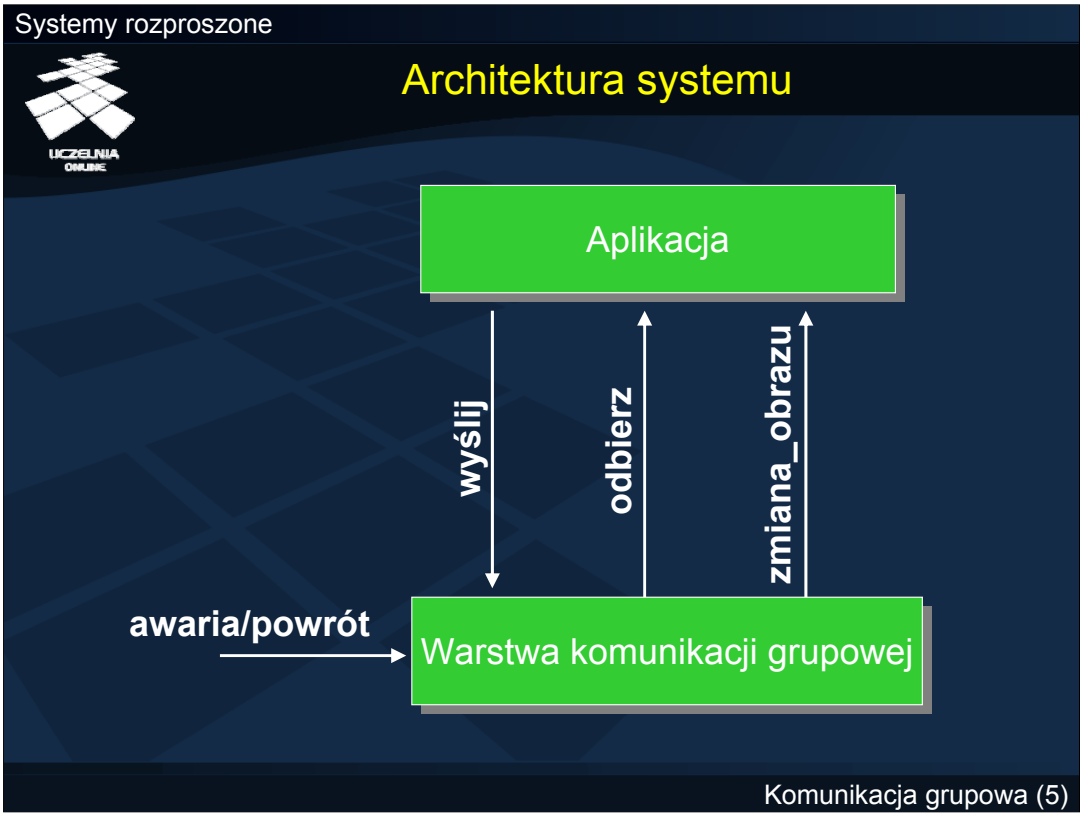
wyślij(p, m) – proces *p* wysyła wiadomość *m*; zauważmy, że nie określa się odbiorcy, gdyż wiadomość wysyłana jest do całej grupy

odbierz(p, m) – proces *p* odbiera wiadomość *m*; nadawca wiadomości może być określony, ale tutaj dla uproszczenia nie jest;

zmiana_obrazu(p, <id, członkowie>) – w procesie *p* następuje zmiana obrazu; nowy obraz o identyfikatorze *id* zawiera procesy określone zmienną *członkowie*. Sam obraz zaś jest parą (*id*, *członkowie*), obejmującą globalny identyfikator obrazu i należące do niego procesy. Dwa obrazy V i V' są więc równe wtedy i tylko wtedy, gdy równe są ich identyfikatory oraz zbiory procesów.

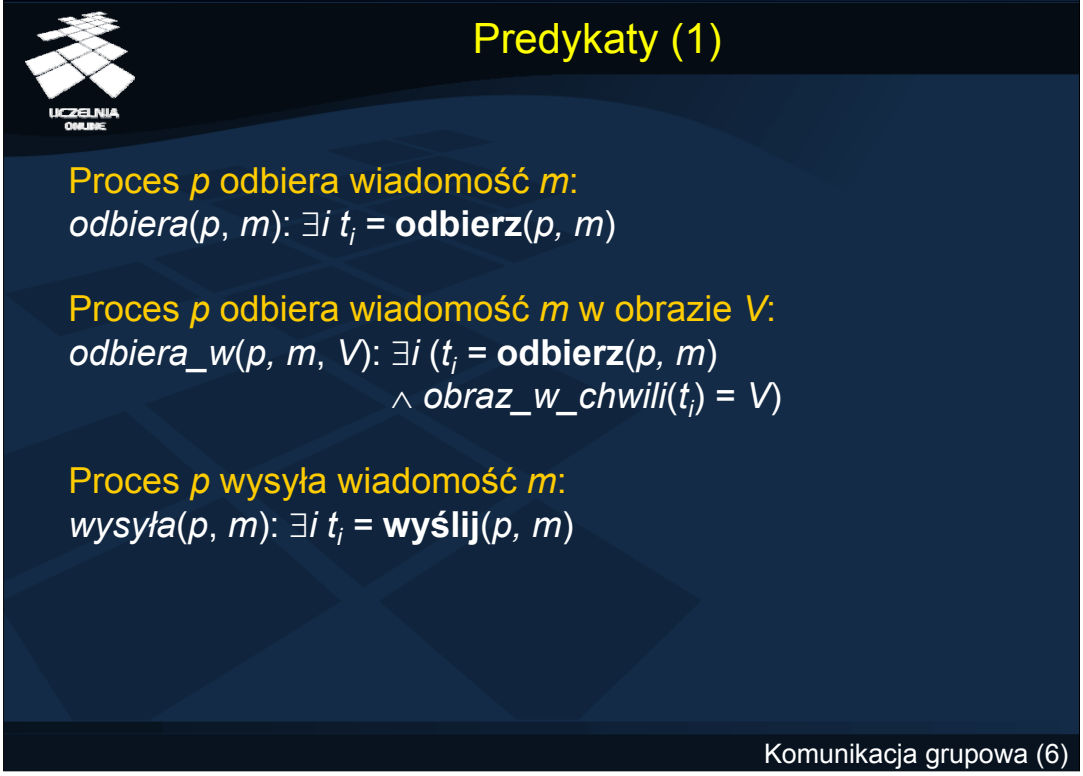
awaria(p) – proces *p* ulega awarii

powrót(p) – proces *p* wznawia działanie po awarii



Oprogramowanie realizujące komunikację grupową stanowi warstwę usługową dla aplikacji, stąd znajduje się poniżej aplikacji. Wejściowe zdarzenie *wyślij* zachodzi z inicjatywy aplikacji, zgłaszającej wysłanie wiadomości do grupy. Wyjściowe zdarzenia *odbierz* i *zmiana_obrazu* są natomiast zgłaszane przez warstwę komunikacji grupowej. Wejściowe zdarzenia *awaria* i *powrót* zgłasza detektor uszkodzeń, którego istnienie i poprawne działanie tutaj zakładamy.

Warstwa komunikacji grupowej, realizująca system komunikacji grupowej, jest zazwyczaj złożona z wielu podwarstw, realizujących takie funkcje, jak np. wspomniane już zarządzanie grupami (ang. *membership*), rozsyłanie (ang. *multicast*), funkcje wznawiania pracy procesu po awarii, uzgadnianie i inne. Na styku z aplikacją dostarcza jednak przejrzystego interfejsu, dzięki czemu projekt i implementacja aplikacji są znacząco uproszczone.



Powyższe predykaty budowane są na podstawie elementarnych zdarzeń, przedstawionych wcześniej.

Predykat *obraz_w_chwili* definiuje obraz, jaki w procesie p istnieje w chwili zajścia zdarzenia t_i.

Predykat *odbiera* zachodzi, gdy proces p odbierze wiadomość m w dowolnej chwili. Predykat *odbiera_w* określa dodatkowo, w jakim obrazie proces *p* odbiera wiadomość *m*.

Predykat *wysyła* jest prawdą, jeśli proces *p* wysłał wiadomość *m* w dowolnej chwili.

Predykaty (2)

Proces *p* wysyła wiadomość *m* w obrazie *V*:
wysyła_w(*p*, *m*, *V*): $\exists i\ (t_i$ = **wyślij**(*p*, *m*)
$\wedge$ *obraz_w_chwili*(t_i) = *V*)

Proces *p* instaluje obraz *V*:
instaluje(*p*, *V*): $\exists i\ t_i$ = **zmiana_obrazu**(*p*, *V*)

Proces *p* instaluje obraz *V* w obrazie *V'*:
instaluje_w(*p*, *V*, *V'*): $\exists i\ (t_i$ = **zmiana_obrazu**(*p*, *V*)
$\wedge$ *obraz_w_chwili*(t_i) = *V'*)

Predykat *wysyła_w* jest prawdziwy, gdy proces *p* wyśle wiadomość *m* w konkretnym obrazie *V*.

Predykat *instaluje* zostaje spełniony, gdy proces *p* zmieni obraz grupy na *V*.

Predykat *instaluje_w* oznacza zmianę obrazu z dotychczasowego *V'* na nowy, *V*. Zmiana obrazu może być spowodowana dołączeniem lub odłączeniem pewnego zbioru procesów do- lub z grupy. Należy pamiętać, że odłączenie procesu modeluje jego faktyczne odłączenie po zakończeniu obliczeń albo awarię, wykrytą i zgłoszoną przez detektor uszkodzeń.

Ponieważ predykat *instaluje_w* uwzględnia poprzedni obraz grupy, jest szczególnie przydatny w tych własnościach, które uwzględniają historyczne obrazy w procesie (najczęściej tylko poprzedni); przykładem takiej własności jest wirtualna synchronizacja, omówiona w dalszej części.

Predykaty (3)

Proces p ulega awarii w obrazie V:
$awaria\_w(p, V)$: $\exists i\ (t_i = \mathbf{awaria}(p) \wedge obraz\_w\_chwili(t_i) = V)$

Zdarzenie t_i następuje po zdarzeniu t_j w procesie p:
$następne\_zdarzenie(i, j, p)$:
$j < i \wedge pid(t_i) = pid(t_j) = p \wedge \neg\exists k\ (pid(t_k) = p \wedge j < k < i)$

Zdarzenie t_i poprzedza zdarzenie t_j w procesie p:
$poprzednie\_zdarzenie(i, j, p)$:
$j > i \wedge pid(t_i) = pid(t_j) = p \wedge \neg\exists k\ (pid(t_k) = p \wedge j > k > i)$

Predykat *awaria_w* zachodzi wówczas, gdy awaria procesu p nastąpi w chwili, gdy w procesie tym pamiętany jest obraz grupy V.

O bezpośrednim poprzedzaniu i następstwie zdarzeń w jednym procesie mówią predykaty *następne_zdarzenie* i *poprzednie_zdarzenie*.

Przedstawione predykaty posłużą w dalszej części prezentacji do definiowania różnych własności systemów komunikacji grupowej.

Założenia odnośnie środowiska

1. **Integralność wykonania** — po awarii procesu następnym zdarzeniem w tym procesie jest powrót, a ostatnim zdarzeniem w procesie przed zdarzeniem powrotu jest awaria. Formalnie:

   $t_j = \textbf{awaria}(p) \wedge \textit{następne\_zdarzenie}(i, j, p) \Rightarrow t_i = \textbf{powrót}(p)$

   $t_j = \textbf{powrót}(p) \Rightarrow \exists i\, (\textit{poprzednie\_zdarzenie}(i, j, p) \wedge t_i = \textbf{awaria}(p))$

2. **Unikalność wiadomości**

   $t_i = \textbf{wyślij}(p, m) \wedge t_j = \textbf{wyślij}(q, m) \Rightarrow i = j$

Założenie o integralności wykonania (ang. *execution integrity*) wymaga, by pomiędzy zdarzeniami **awaria** i **powrót** w jednym procesie nie następowało żadne inne zdarzenie. Innymi słowy, przyjmuje się model awarii znany pod nazwą awaria-wznowienie (ang. *crash-recovery*). W modelu tym, w odróżnieniu od modelu awarii bizantyjskich, przyjmuje się, że proces po awarii nie wykonuje żadnych akcji (np. nie wysyła i nie odbiera wiadomości).

W jeszcze prostszym modelu awarii, nazywanym modelem „awaria-stop” (ang. *fail-stop*) nie dopuszcza się nawet wznowienia pracy po awarii – proces, który jej uległ, przestaje działać na zawsze. W uproszczeniu można powiedzieć, że różnica pomiędzy modelami awaria-wznowienie i awaria-stop polega na tym, że w tym drugim nie ma zdarzenia powrotu z awarii i wznowienia pracy procesu.

Założenie o unikalności wiadomości wymaga, by każda wiadomość *m* była niepowtarzalna; najczęściej osiąga się to poprzez przypisanie każdej wiadomości unikalnego identyfikatora wiadomości, złożonego na przykład z identyfikatora nadawcy połączonego z numerem sekwencyjnym lub znacznikiem czasowym. Dzięki temu założeniu można wymagać, by każda wiadomość w systemie była wysłana najwyżej jeden raz.

Usługa członkostwa – własności

1. **Samozawieranie** — proces należy do własnego obrazu grupy
   $instaluje(p, V) \Rightarrow p \in V.członkowie$
2. **Monotoniczność lokalna** — identyfikator danego obrazu grupy jest większy od identyfikatorów wcześniejszych obrazów grupy
   $t_i = \textbf{zmiana\_obrazu}(p, V) \wedge t_j = \textbf{zmiana\_obrazu}(p, V') \wedge j > i \Rightarrow V'.id > V.id$
3. **Początkowy obraz** — każde wysłanie lub odbiór wiadomości odbywa się w jakimś obrazie
   $t_i = \textbf{wyślij}(p, m) \vee t_i = \textbf{odbierz}(p, m) \Rightarrow obraz\_w\_chwili(t_i) \neq \varnothing$

Przez **własność** należy rozumieć gwarancję dostarczaną przez system. Oczywiście w różnych systemach dostępne są różne gwarancje, zależne od decyzji twórców systemu.

Własność **samozawierania** (ang. *self inclusion*) mówi, że dany proces powinien należeć do własnego obrazu grupy. Ponieważ każdy proces zawsze jest w stanie komunikować się z sobą samym, gwarancji tej dostarczają wszystkie znane systemy komunikacji grupowej.

Własność **monotoniczności lokalnej** (ang. *local monotonicity*) wymaga, by identyfikator danego obrazu grupy był większy od identyfikatorów wcześniejszych obrazów.

Własność **początkowego obrazu** (ang. *initial view event*) wymaga zaś, by każde wysłanie lub odbiór wiadomości następowało w pewnym obrazie. W szczególności oznacza to, że na samym początku przetwarzania musi w procesie istnieć pewien obraz grupy.

Usługa rozsyłania – własności (1)

1. **Brak samogeneracji** — każda odebrana wiadomość została wcześniej wysłana; wiadomości nie pojawiają się „znikąd”

   t_i = **odbierz**(p, m) $\Rightarrow \exists q\, \exists j\, (j < i \wedge t_j$ = **wyślij**(q, m))

2. **Brak powielania** — jeden proces może odebrać daną wiadomość najwyżej jeden raz

   t_i = **odbierz**(p, m) $\wedge\ t_j$ = **odbierz**(p, m) $\Rightarrow i = j$

Własności **braku samogeneracji** (ang. *delivery integrity*) oraz **braku powielania** (ang. *no duplication*) stawiają wymagania kanałom komunikacyjnym. Pierwsza z nich stanowi, że każda odebrana wiadomość musi mieć swojego nadawcę, który wcześniej ją wysłał. Innymi słowy, kanał sam z siebie nie może wygenerować wiadomości, bo z punktu widzenia systemu pojawiłaby się ona „znikąd”. Druga zaś własność wymaga, by jeden proces nie mógł odebrać danej wiadomości więcej, niż jeden raz.

Usługa rozsyłania – własności (2)

3. **Dostarczenie w obrazie z momentu wysłania** — jeśli proces dostarczył wiadomość *m*, to ma pewność, że dostarczył ją w tym samym obrazie, w którym została ona wysłana

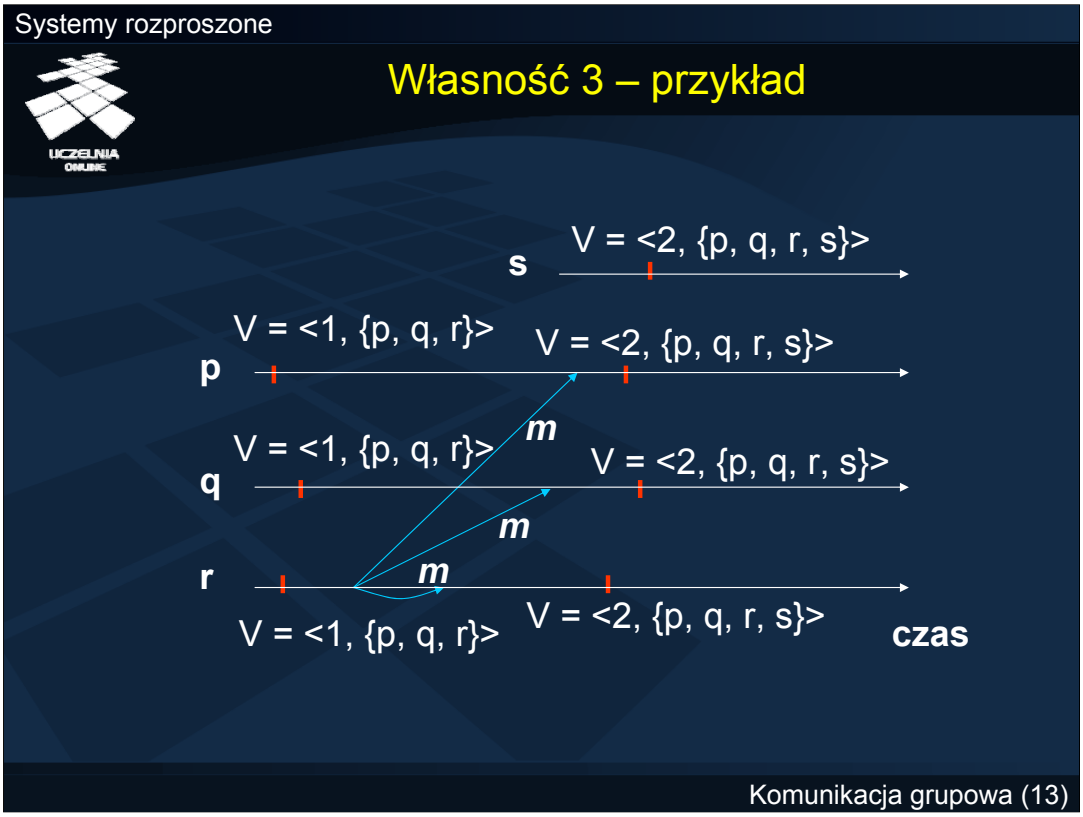
   $odbiera\_w(p, m, V) \wedge wysyła\_w(q, m, V') \Rightarrow V = V'$

4. **Dostarczenie w tym samym obrazie** — jeśli proces dostarczył wiadomość *m*, to ma pewność, że dostarczył ją w tym samym obrazie, w którym dostarczyły ją inne procesy.

   $odbiera\_w(p, m, V) \wedge odbiera\_w(q, m, V') \Rightarrow V = V'$

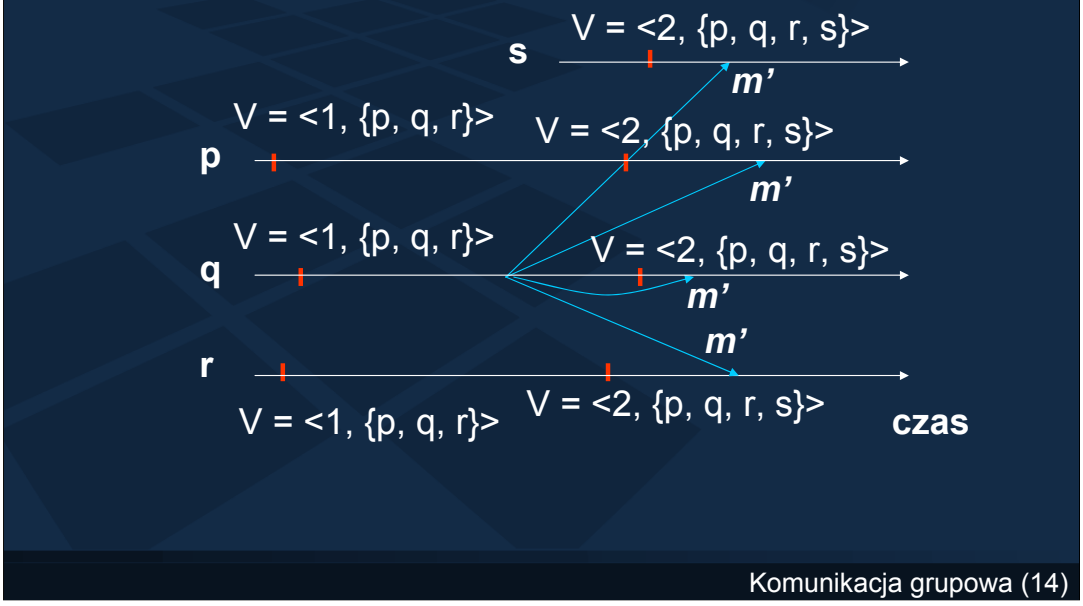
Własność 3, **dostarczenia w obrazie z momentu wysłania** (ang. *sending view delivery*), wymaga, by wiadomość została dostarczona do wszystkich odbierających ją procesów w tym samym obrazie, w którym została wysłana. Oznacza to, że pomiędzy zdarzeniem wysłania a zdarzeniem dostarczenia do ostatniego procesu nie mogła zajść żadna zmiana w obrazie grupy, w szczególności więc również awaria dowolnego procesu. Należy podkreślić, że własność ta (podobnie jak własność 4) nie wymaga dostarczenia wiadomości do wszystkich procesów w grupie. Można ją wyrazić inaczej w sposób następujący: *jeśli* proces *q* dostarcza wiadomość *m*, to odbiera ją *na pewno* w tym samym obrazie, w którym wysłał ją proces *p*.

Słabsza własność 4, **dostarczenie w tym samym obrazie** (ang. *same view delivery*), nie wymaga już, by obraz w procesie odbierającym był równy obrazowi z momentu wysłania. Nadal jednak wymaga, by obrazy wszystkich procesów odbierających były sobie równe.



W przykładzie na rysunku dla wiadomości *m* zagwarantowano własność dostarczenia w obrazie z momentu wysłania, gdyż została dostarczona do odbierających ją procesów w tym samym obrazie, w którym proces *r* ją wysłał. W przykładzie odbierają ją wszystkie procesy należące do obrazu, choć przypomnijmy, że nie musiałoby tak być – ważne jest tylko, by w odbierających procesach dostarczenie wiadomości nastąpiło w tym samym obrazie, w którym zaszło wysłanie.

Własność 4 – przykład



Dla wiadomości *m'* natomiast zagwarantowano własność dostarczenia w tym samym obrazie – dostarczenie zachodzi w momencie, gdy wszystkie odbierające procesy mają ten sam obraz, choć nie jest to już ten sam obraz, w którym wysyłano wiadomość.

Usługa rozsyłania – własności (3)

5. Wirtualna synchronizacja

Jeśli procesy p i q instalują nowy, identyczny obraz V w poprzednim, identycznym obrazie V', to każda wiadomość dostarczona w obrazie V' przez proces p jest również dostarczona obrazie V' przez proces q

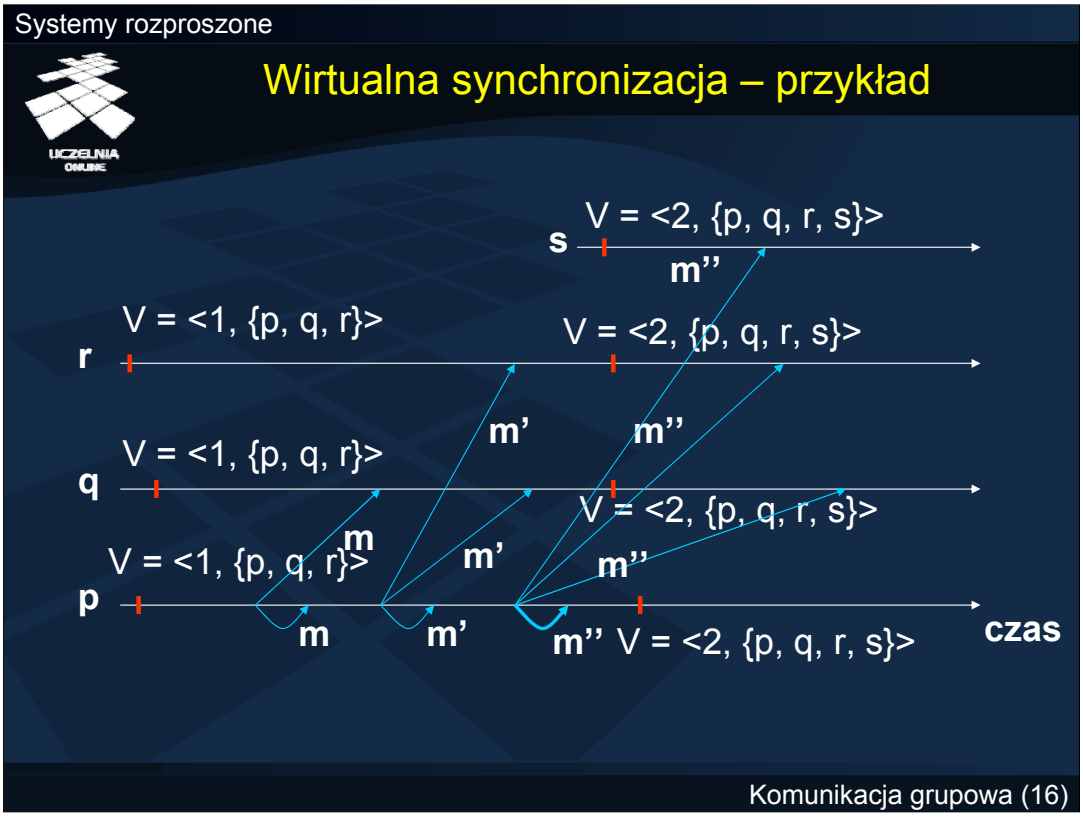
$$instaluje\_w(p, V, V') \wedge instaluje\_w(q, V, V') \wedge odbiera\_w(p, m, V') \Rightarrow odbiera\_w(q, m, V')$$

Najpopularniejsza we współczesnych systemach komunikacji grupowej jest własność **wirtualnej synchronizacji** (ang. *virtual synchrony* lub *view synchrony*). Wymaga ona, by wszystkie procesy, które ze starego obrazu *V'* przechodzą do nowego obrazu *V*, dostarczyły przed zainstalowaniem nowego obrazu (czyli jeszcze w starym) każdą wiadomość, którą dostarczył choć jeden proces. Innymi słowy, wymaga się od procesów dostarczania tych samych zbiorów wiadomości w tych samych obrazach. Intuicyjnie oznacza to, że procesy są synchronizowane kolejnymi zmianami obrazów – zmiana obrazu jest barierą, której proces nie może przekroczyć, jeśli nie odebrał wymaganych wiadomości.

Własność ta jest bardzo przydatna na przykład w systemach rozproszonych wykorzystujących zwielokrotnianie, w których kopie danego obiektu najczęściej tworzą grupę. Wiele modeli spójności na niższym poziomie wymaga zachowania pewnych gwarancji komunikacji grupowej, w tym często właśnie wirtualnej synchronizacji.

Należy jeszcze podkreślić zasadniczą różnicę pomiędzy wirtualną synchronizacją a własnościami dostarczenia w obrazie z momentu wysłania czy dostarczenia w tym samym obrazie.. Otóż wirtualna synchronizacja stawia wymagania wszystkim procesom zmieniającym obraz, co w praktyce najczęściej oznacza wszystkie procesy w grupie.

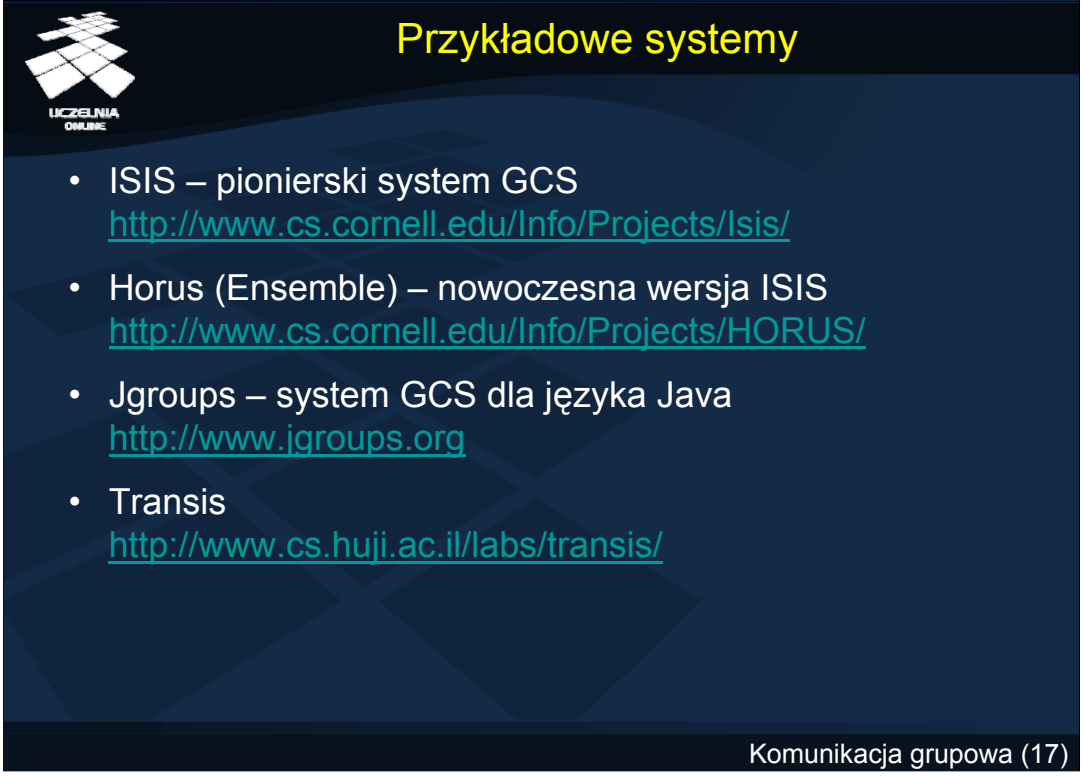
Można ponadto zauważyć, że wirtualna synchronizacja jest ściśle silniejsza od własności dostarczenia w tym samym obrazie, natomiast jest nieporównywalna z własnością dostarczenia w obrazie z momentu wysłania.



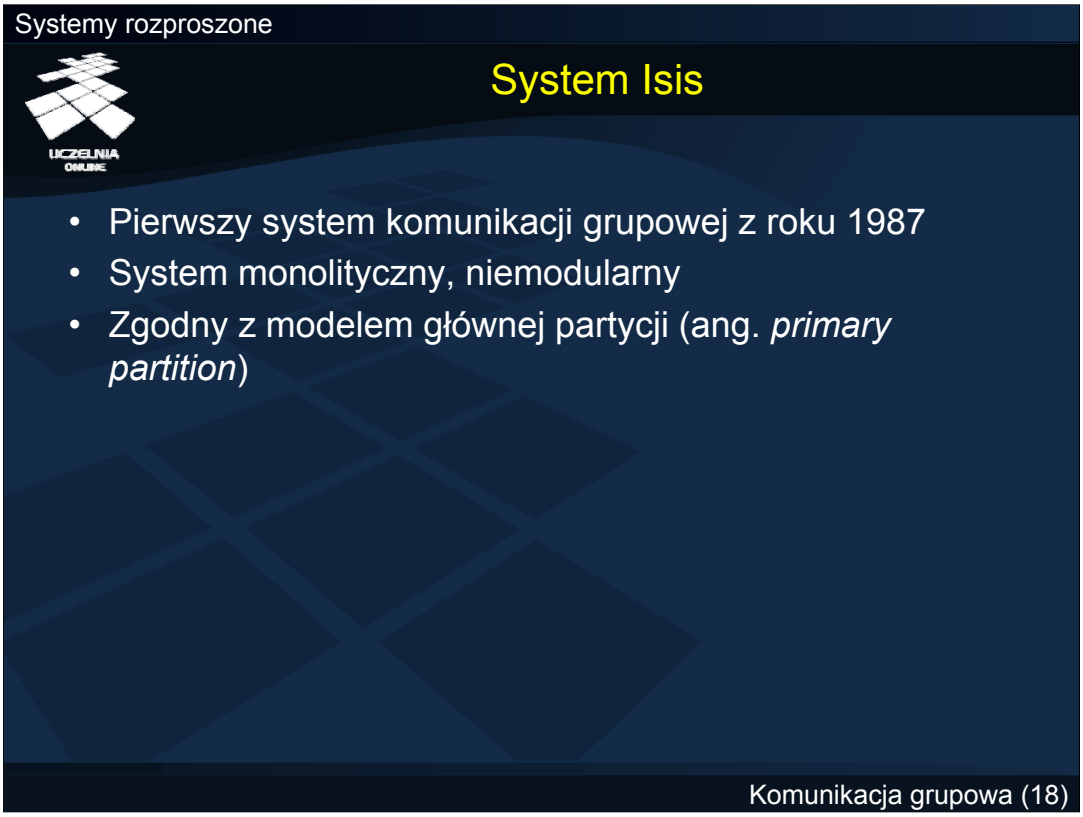
W przedstawionym przykładzie, dla wiadomości *m* zachowana jest własność dostarczenia w obrazie z momentu wysłania (odbierające procesy *p* i *q* dostarczają wiadomość w obrazie 1, w którym została ona wysłana), a tym samym własność dostarczenia w tym samym obrazie (wszystkie odbierające procesy dostarczają wiadomość w tym samym obrazie), nie jest natomiast zachowana własność wirtualnej synchronizacji, gdyż nie wszystkie procesy zmieniające obraz grupy ją dostarczyły.

Dla wiadomości *m'* zachowano każdą z trzech własności. W przypadku wiadomości *m''* zaś nie jest zachowana żadna z trzech własności, gdyż proces *p* dostarcza tę wiadomość w innym obrazie, niż pozostałe procesy.

Należy zaznaczyć, że zazwyczaj należy dostarczać tylko jednej konkretnej gwarancji, zależnej od wymagań aplikacji w zakresie niezawodności. Ponieważ jednak celem rysunku jest ukazanie różnic pomiędzy wspomnianymi własnościami, wspomina się dla każdej wiadomości o każdej własności.

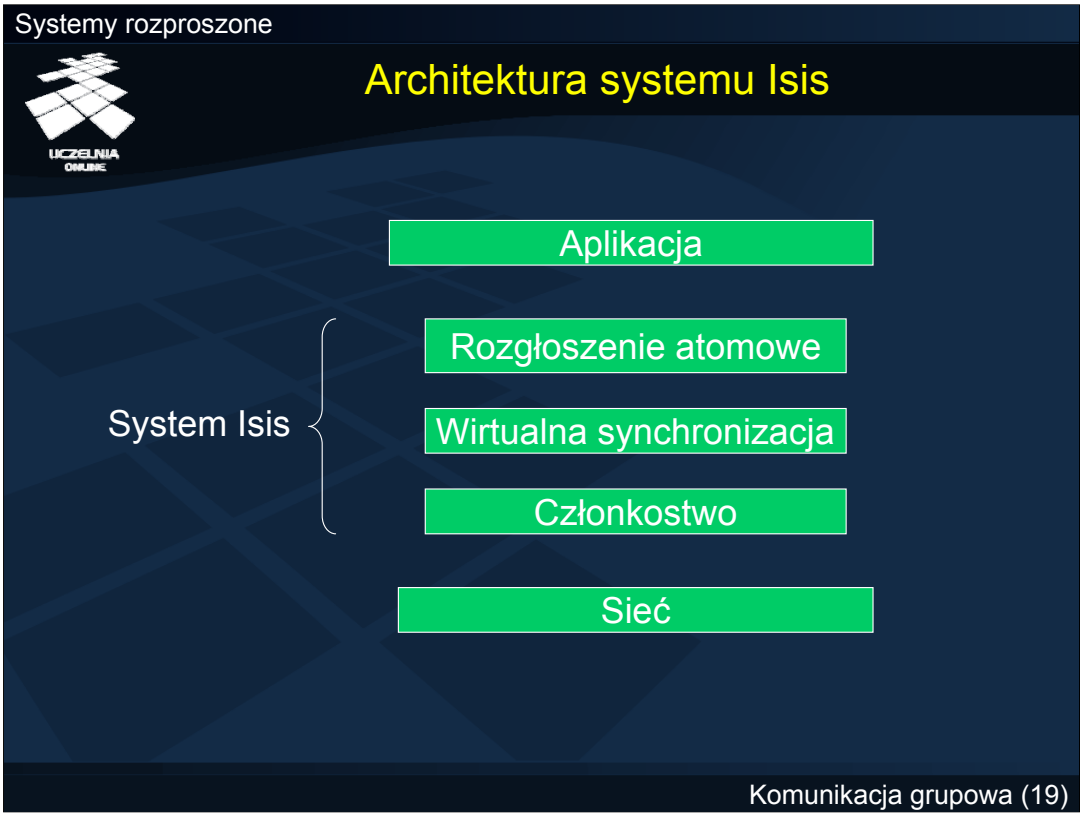


Wymienione tu projekty to przykłady popularnych systemów komunikacji grupowej, nazywanych skrótowo systemami GCS (ang. *Group Communication System*). Są one realizowane jako warstwa pośrednia oprogramowania (ang. *middleware*), oferująca usługi w zakresie zarządzania grupami i niezawodnej komunikacji w grupach. Dwa z nich zostaną omówione bardziej szczegółowo na kolejnych slajdach. Opis oraz rysunki oparto na pracy „A Step Towards a New Generation of Group Communication Systems", wymienionej w bibliografii.



Isis jest pierwszym systemem, w którym zaproponowano wyodrębnienie mechanizmów komunikacji grupowej. Należy do grupy systemów monolitycznych, czyli niemodularnych, a więc takich, w których architektura jest stała i nie może być dostosowana do konkretnych potrzeb aplikacji. Oczywistą wadą takich systemów jest konieczność dostosowania aplikacji do nich oraz konieczność użycia wszystkiego, co oferuje system – nawet, jeśli byłoby to zbędne i skutkowało spadkiem efektywności pracy.

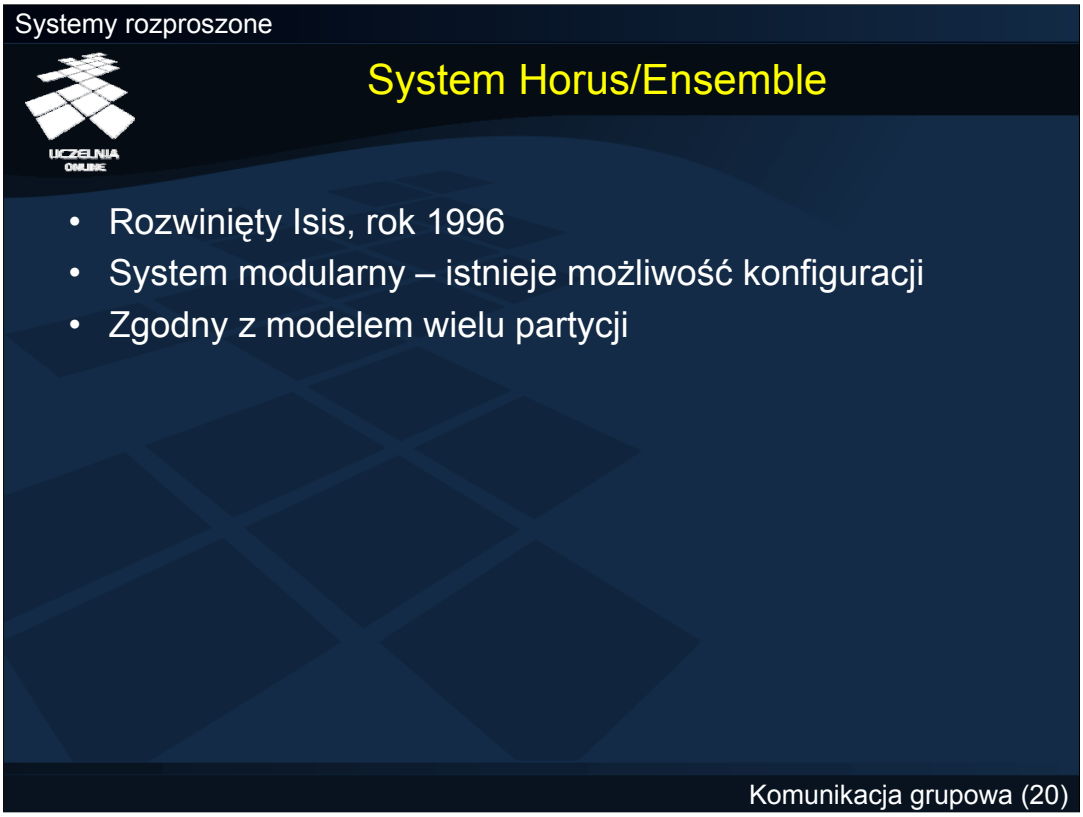
Ponadto, Isis jest systemem zgodnym z modelem głównej partycji (ang. *primary partition*), co oznacza, że przy podziale sieci przetwarzanie jest kontynuowane tylko w głównym jej fragmencie, w tak zwanej głównej partycji. W szczególności oznacza to, że wszystkie procesy głównej partycji otrzymują taką samą sekwencję obrazów grupy.



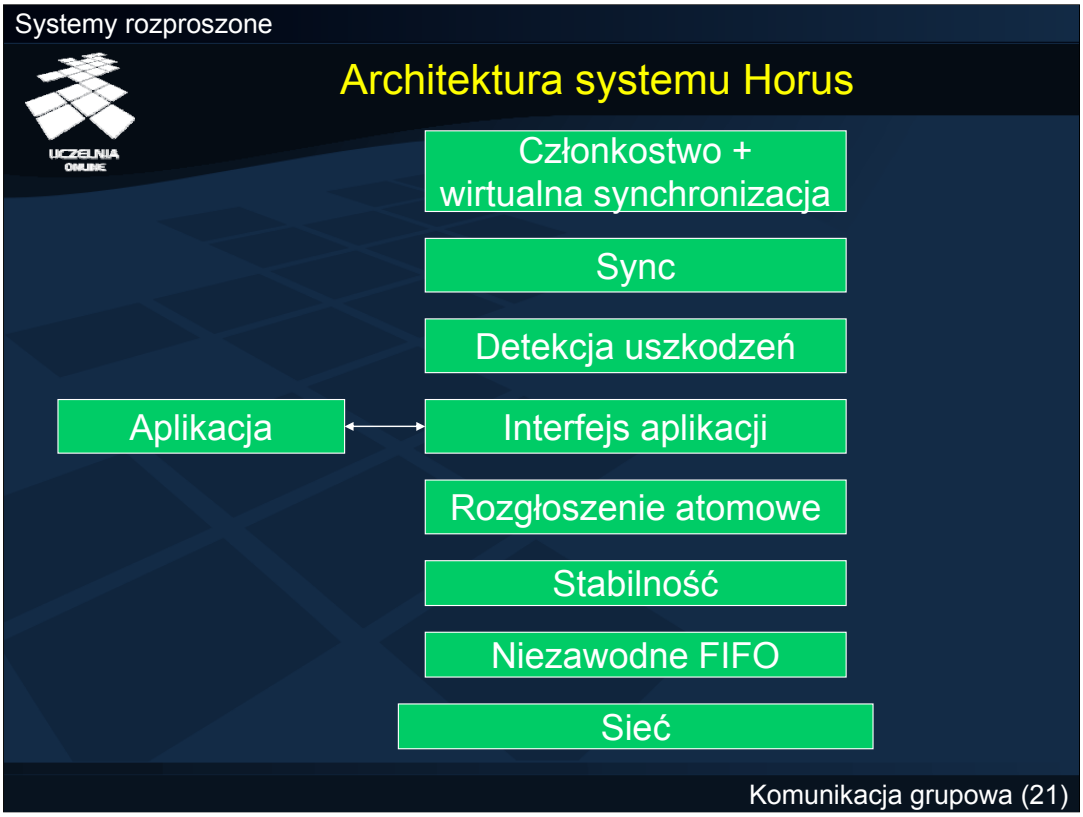
Warstwa członkostwa jest odpowiedzialna za zarządzanie składem grupy, obejmujące monitorowanie wykrywanie zdarzeń *leave*, oznaczających opuszczenie grupy (na skutek zakończenia przetwarzania lub awarii procesu) oraz obsługę zdarzeń *join* dołączenia do niej nowego procesu. Warstwa ta zapewnia procesom dostarczanie nowych obrazów grupy w takiej samej kolejności, w *całkowitym* uporządkowaniu (ang. *total order*).

Warstwa członkostwa nie dostarcza jednak żadnych mechanizmów komunikacyjnych. Z tego powodu została rozszerzona o warstwę **wirtualnej synchronizacji**, realizującą rozgłaszanie wiadomości do aktualnych członków grupy.

Najwyższa warstwa, **rozgłaszania atomowego**, zapewnia, że wiadomości aplikacyjne są dostarczane w tej samej kolejności przez wszystkie procesy. Rozgłaszanie atomowe jest tutaj zrealizowane w oparciu o znajdującą się niżej warstwę wirtualnej synchronizacji.



System Horus, czy jego wersja zaimplementowana w języku OCaml i znana pod nazwą Ensemble, jest następcą systemu Isis. Jako system modularny, niemonolityczny, pozwala użytkownikowi wybrać tylko te funkcje, których on potrzebuje. Ponadto, jest zgodny z modelem wielu partycji (ang. *partitionable membership*), co oznacza, że po podziale sieci dopuszcza się kontynuowanie przetwarzania w różnych jej fragmentach.



Na rysunku przedstawiono przykładowy stos protokołów systemu Horus:

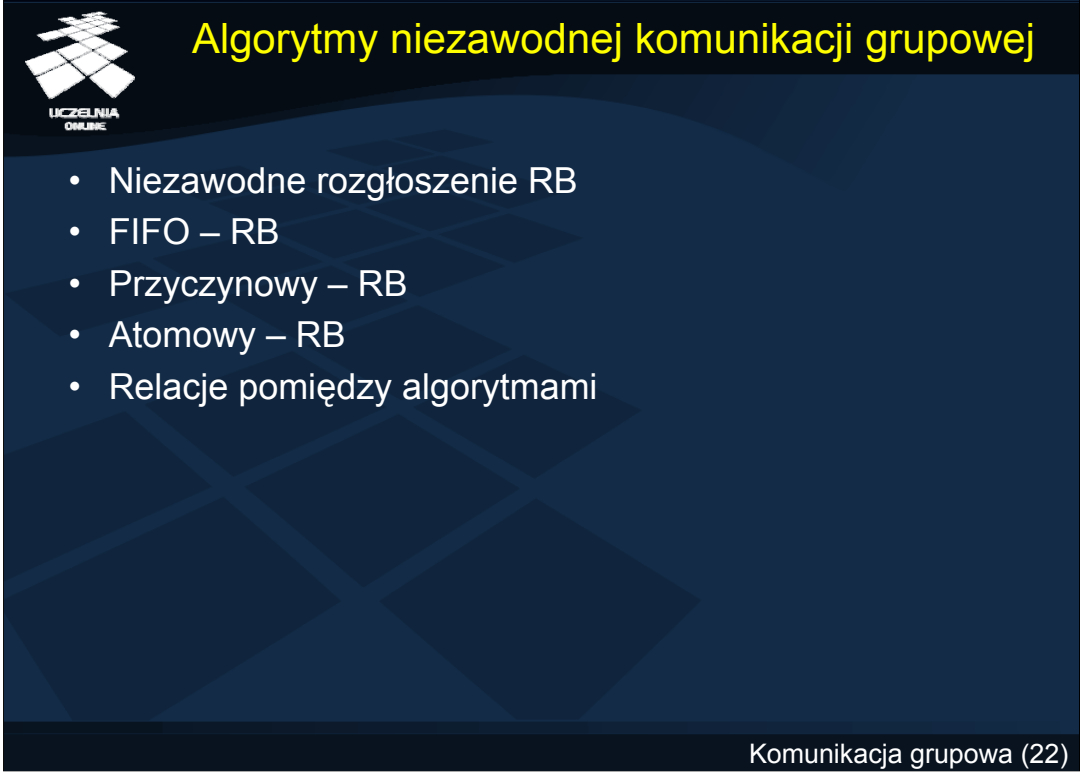
Należy zaznaczyć, że informacje o zdarzeniach w systemie Horus przekazywane są *w górę stosu*, począwszy od najniżej działającej warstwy. Warstwa generująca dane zdarzenie propaguje je najpierw w dół stosu, a następnie informacja o nim „odbija się" od najniższej warstwy i wędruje w górę, do najwyższej.

Komponent *Niezawodne FIFO* realizuje kanały komunikacyjne, zachowujące uporządkowanie wiadomości FIFO, polegające na dostarczaniu wiadomości od danego nadawcy zgodnie z kolejnością ich wysłania przez tego nadawcę (szerzej omówiono to zagadnienie w drugiej części wykładu, poświęconej wybranym algorytmom komunikacji grupowej).

Warstwa *Stabilność* zajmuje się sprawdzaniem, czy wiadomości na danym poziomie zostały odebrane przez wszystkie procesy w grupie. O takich wiadomościach mówi się, że są *stabilne*. Wiadomości stabilne wykorzystuje się w aplikacjach wymagających wysokiej niezawodności przy dostarczaniu wiadomości, na przykład do zaimplementowania wirtualnej synchronizacji. Najczęściej stabilne wiadomości zostają trwale zapisane, by mogły przetrwać ewentualną awarię maszyny lub aplikacji i zostać dostarczone po wznowieniu pracy. Warstwa Stabilność może zostać umieszczona w wielu miejscach stosu; wybór jej lokalizacji ma jednak wpływ na efektywność pracy.

Warstwa *rozgłaszania atomowego* porządkuje wiadomości, ale robi to tylko w sytuacjach bezawaryjnych. Gdyby więc nie skorzystano z dodatkowych komponentów odpowiedzialnych za detekcję uszkodzeń i członkostwo, wówczas różne algorytmy rozgłaszania atomowego blokowałyby się w przypadku awarii lub podziału sieci.

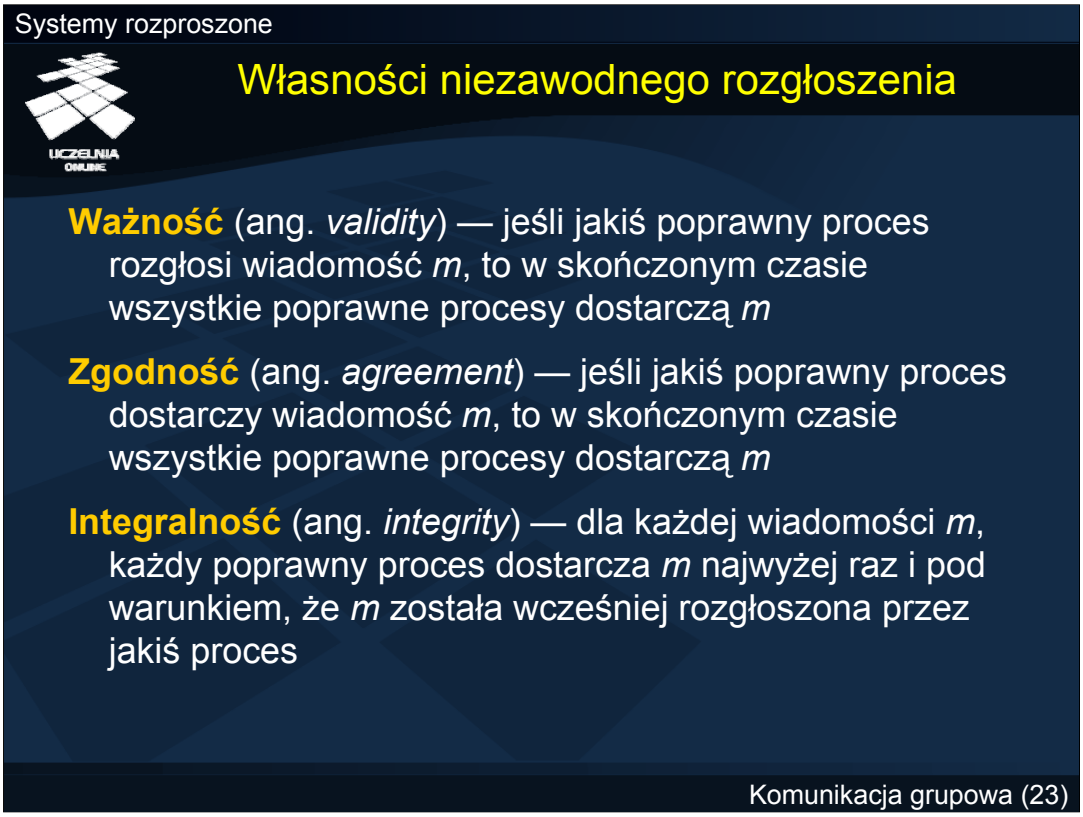
Co ważne, *aplikacja nie jest* umieszczona w najwyższej warstwie. Dzieje się tak z przyczyn wydajnościowych – gdyby aplikacja była na samej górze, dostarczanie do niej informacji o zachodzących zdarzeniach trwałoby dłużej. Bardziej efektywnym rozwiązaniem jest umieszczenie komponentów aktywnych



Druga część wykładu dotyczy algorytmów niezawodnego rozsyłania (ang. *multicast*) wiadomości pomiędzy procesami w grupie. Omówione zostaną podstawowe algorytmy: niezawodne rozesłanie bez dodatkowych gwarancji na kolejność wiadomości (RB), niezawodne rozesłanie z dodatkową gwarancją uporządkowania FIFO (FIFO-RB), niezawodne rozesłanie z gwarancją uporządkowania przyczynowego wiadomości (Przyczynowy-RB) oraz niezawodne rozesłanie atomowe.

Dla każdego algorytmu dopuszcza się awarie procesów oraz kanałów komunikacyjnych. Szczegółowe założenia będą podane przy konkretnych algorytmach.

W dalszej części prezentacji terminy „rozesłanie” i „rozgłoszenie” są używane zamiennie i oznaczają to samo – wysłanie wiadomości do wszystkich procesów w grupie.

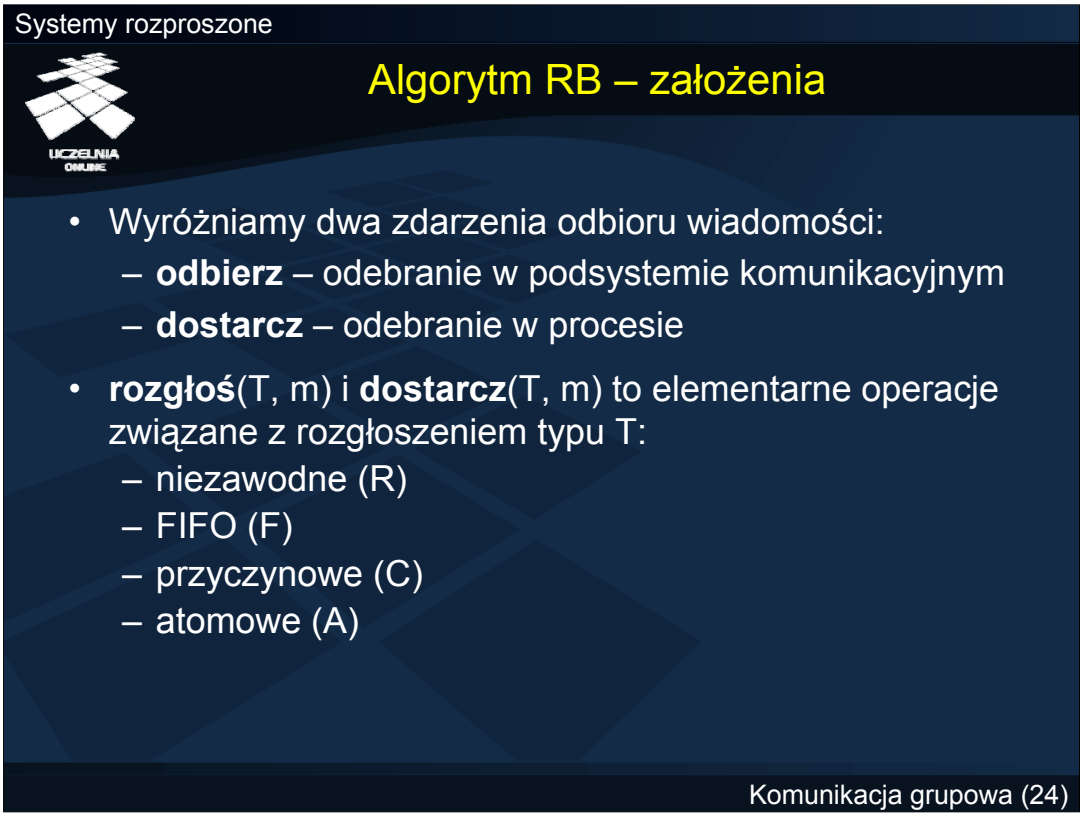


Zadaniem algorytmu **niezawodnego rozgłaszenia** (ang. *Reliable Broadcast – RB*) jest dostarczenie wiadomości wszystkim poprawnym procesom (tzn. procesom, które na pewno nie ulegną awarii w trakcie przebiegu algorytmu) pomimo awarii innych procesów lub kanałów komunikacyjnych. Specyfikacja algorytmu obejmuje następujące własności:

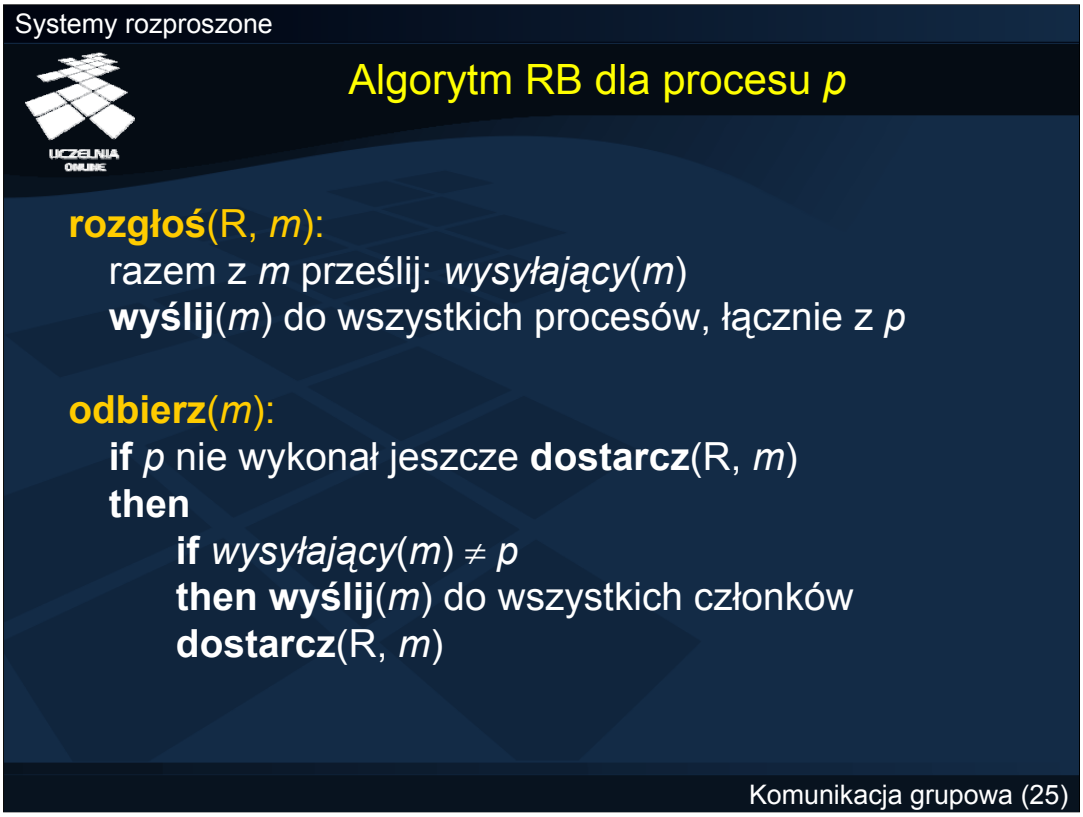
Ważność – wymaga, by jeśli poprawny proces rozgłosił wiadomość, wszystkie poprawne procesy w skończonym czasie dostarczyły tę wiadomość.

Zgodność – wymaga, by jeśli choć jeden poprawny proces dostarczył wiadomość, wszystkie procesy dostarczyły tę wiadomość, nawet jeśli proces nadawcy był niepoprawny (np. uległ awarii podczas rozsyłania i nie zdołał wiadomości wysłać do wszystkich procesów).

Integralność – wymaga, by daną wiadomość każdy proces dostarczył najwyżej jeden raz i to pod warunkiem, że została ona wcześniej rozgłoszona przez jakiś (niekoniecznie poprawny) proces.



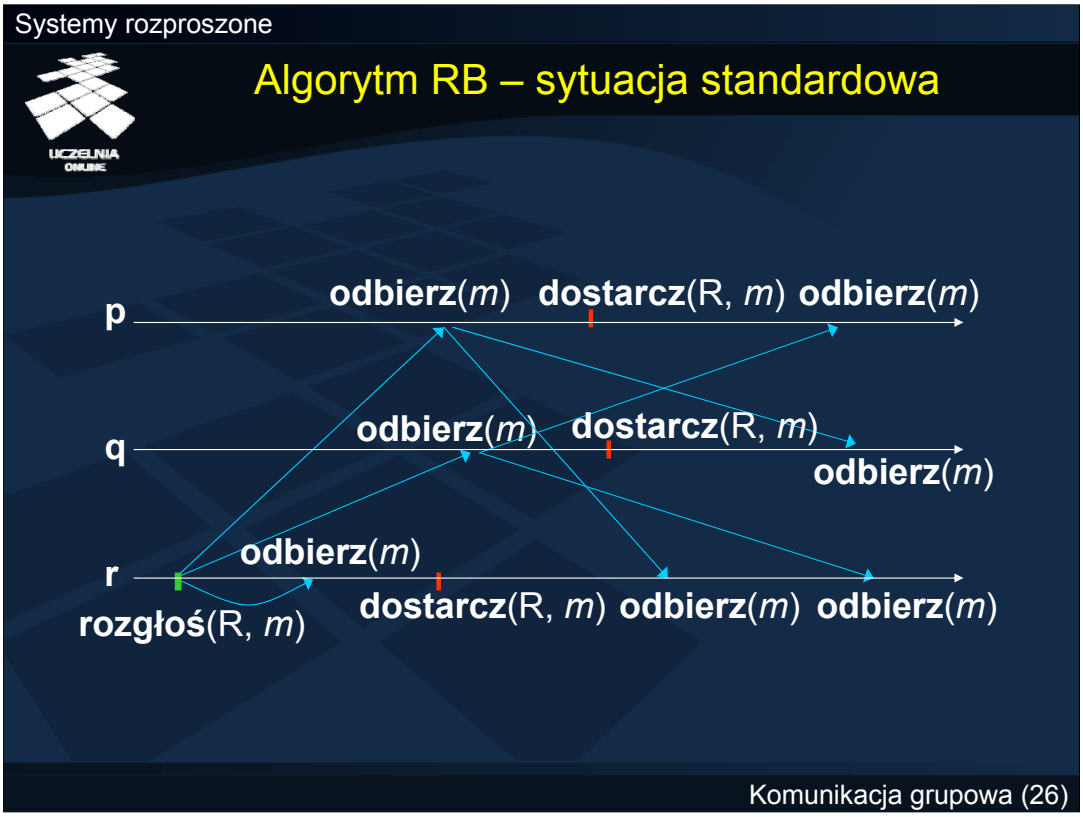
Uwaga: W dalszej części prezentacji wyróżniamy dwa zdarzenia odbioru wiadomości, wykorzystane w przedstawianych algorytmach:
odbierz – wiadomość została już odebrana przez podwarstwę komunikacji grupowej, ale jeszcze nie przez proces
dostarcz – wiadomość odbiera proces/aplikacja.
Ponadto, przez **rozgłoś**(T, m) i **dostarcz**(T, m) oznaczamy dwie elementarne operacje związane z rozgłoszeniem typu T (R – niezawodne, F – FIFO, C – przyczynowe, A – atomowe).



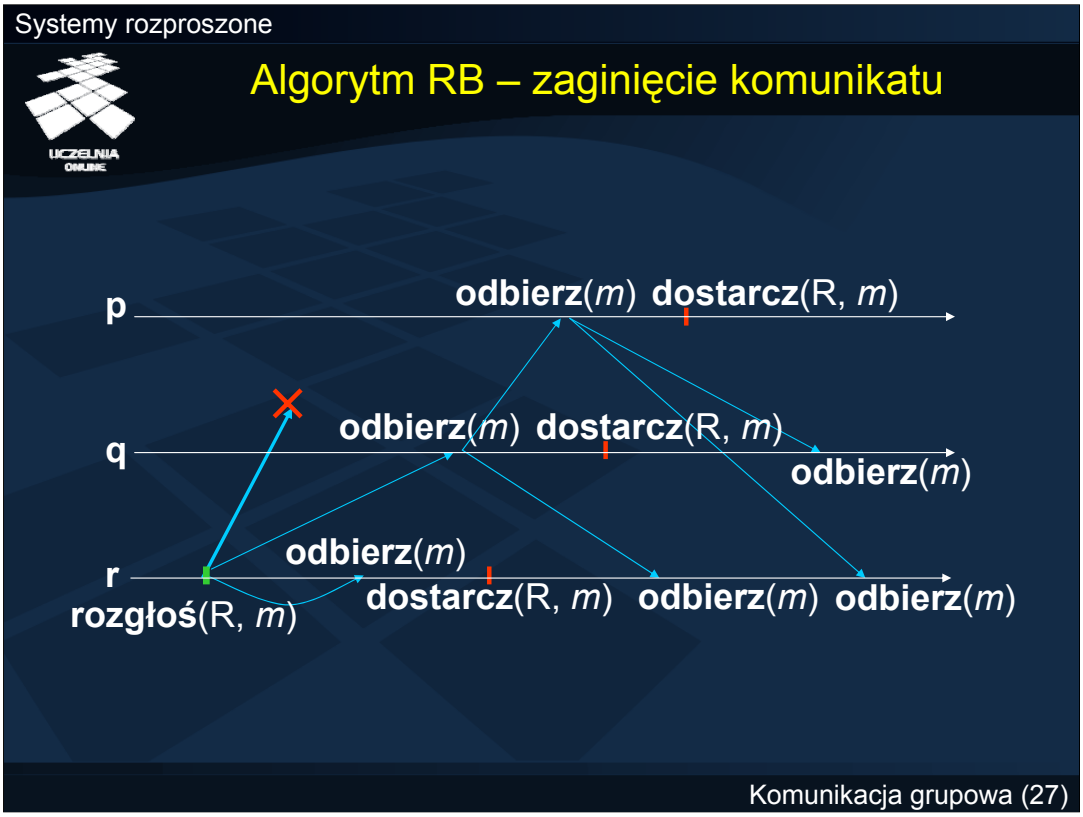
Ideą algorytmu jest, by natychmiast po odebraniu wiadomości przez warstwę komunikacji grupowej, rozgłosić ją do wszystkich procesów w grupie, a dopiero potem dostarczyć procesowi aplikacyjnemu. Takie ponowne rozgłoszenie ma na celu umożliwić dalszą propagację wiadomości pomimo awarii w innych procesach, a w szczególności – w procesie nadawcy.

Należy podkreślić, że dla poprawnego działania algorytmu potrzebne jest, by do każdej wiadomości proces nadawcy dołączał swój identyfikator oraz numer sekwencyjny wiadomości; te dwa parametry tworzą razem unikalny identyfikator wiadomości.

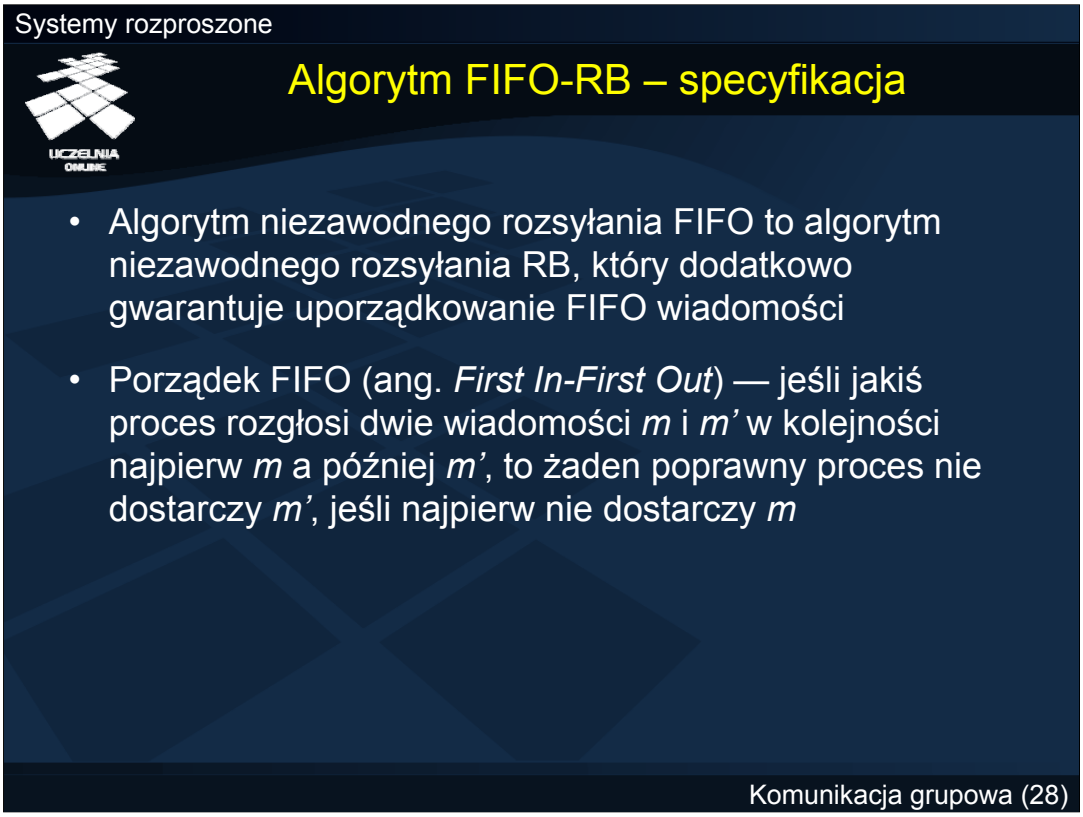
Twierdzenie: Przy założeniu, że w systemie każde dwa poprawne procesy są połączone przez ścieżkę poprawnych łączy i procesów, można udowodnić, że algorytm RB realizuje niezawodne rozgłoszenie w takim systemie.



Jak widać na rysunku, dostarczenie wiadomości do każdego procesu następuje jeden raz; późniejsze zdarzenia odebrania wiadomości nie skutkują jej dostarczeniem. W sytuacji, gdy nie zachodzi żadna awaria, dodatkowe komunikaty nie wnoszą nic do algorytmu, natomiast zwiększają złożoność komunikacyjną. Podniesienie poziomu niezawodności odbywa się więc kosztem zwiększenia złożoności komunikacyjnej i czasowej.



Rysunek uwidacznia, jak algorytm RB gwarantuje dostarczenie wiadomości, pomimo awarii kanału komunikacyjnego między procesami *p* i *r* – wiadomość dociera do procesu r dzięki temu, że ponownie rozgłasza ją proces *q*.



Porządek **FIFO** wymaga, by wiadomości wysyłane przez dany proces były dostarczane do odbiorców w takiej samej kolejności, w jakiej wysyłał je ten proces. Do zagwarantowania tego wystarcza numerowanie w procesie nadawcy wysyłanych przez niego wiadomości.

Algorytm FIFO-RB to taki algorytm niezawodnego rozgłaszania, który dodatkowo gwarantuje uporządkowanie FIFO wiadomości. Algorytm FIFO-RB korzysta na niższym poziomie z algorytmu RB.

Studentowi pozostawia się jako ćwiczenie podanie przykładu przebiegu przestrzenno-czasowego, w którym zostanie naruszone uporządkowanie FIFO.

Niezawodne rozgłaszanie w algorytmie FIFO-RB polega na rozgłoszeniu zgodnym z algorytmem RB. Należy pamiętać, że każda wiadomość posiada numer sekwencyjny, nadany jej przed wysłaniem przez nadawcę.

Zanim wiadomość zostanie dostarczona zgodne z uporządkowaniem FIFO, najpierw następuje niezawodne dostarczenie jej przez działający poniżej algorytm RB. Następnie sprawdza się, czy dana wiadomość jest *oczekiwaną* wiadomością, tzn. czy jest *kolejną* wiadomością, której proces odbierający spodziewa się od danego nadawcy. Fakt, że dana wiadomość jest kolejną, stwierdza się po wartości towarzyszącego jej numeru sekwencyjnego – powinien on być o jeden większy, niż pamiętany w procesie odbiorcy.

Zmienna *worek* pamięta wszystkie te wiadomości, które zostały już dostarczone przez algorytm RB, ale jeszcze nie zostały dostarczone przez algorytm FIFO-RB.

W tablicy *następna* są zaś pamiętane numery sekwencyjne odpowiadające poszczególnym procesom-nadawcom w grupie.

/*zbiór wiadomości, które *p* R-dostarczył, ale jeszcze nie F-dostarczył*/

/* numer sekwencyjny następnej wiadomości, którą *p* F-dostarczy*/

Relacja poprzedzania przyczynowego

$$t_i \rightarrow t_j \begin{cases} 1)\ pid(t_i)=pid(t_j) \wedge j \geq i \\ 2)\ t_i = \textbf{wyślij}(p, m) \wedge t_j = \textbf{odbierz}(q, m) \\ 3)\ \exists k\ (t_i \rightarrow t_k \wedge t_k \rightarrow t_j) \end{cases}$$

- Intuicyjnie: przyczyna zawsze poprzedza skutek
- Jeśli zachodzi $t_i \rightarrow t_j$, to t_i zaszło w czasie na pewno prędzej niż t_j
- Uporządkowanie zgodne z relacją poprzedzania przyczynowego nazywamy „porządkiem przyczynowym”

Relacja poprzedzania przyczynowego, nazywana również porządkiem przyczynowym, bierze swoją nazwę od angielskiej nazwy *Causal Order*. Formalna definicja relacji poprzedzania przyczynowego jest następująca:

Zdarzenie t_j jest przyczynowo zależne od zdarzenia t_i (lub inaczej – t_i poprzedza t_j), gdy:

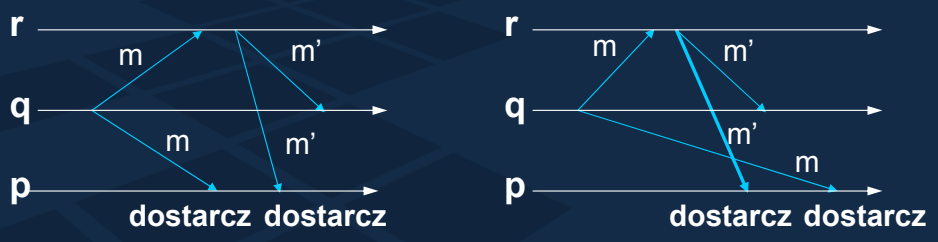
- obydwa te zdarzenia zachodzą w tym samym procesie i t_i wystąpiło wcześniej niż t_j, lub

- t_i jest zdarzeniem wysłania wiadomości, a t_j jest - zdarzeniem jej odebrania, lub

- gdy pomiędzy zdarzeniami t_i a t_j zachodzi inne zdarzenie, t_k, takie że poprzedza je t_i, ale jednocześnie ono samo poprzedza t_j.

Ostatni warunek w powyższej definicji ma charakter rekurencyjny. Zdarzeń pośredniczących pomiędzy t_i a t_j może być wiele, ale relacja poprzedzania przyczynowego może zachodzić pomiędzy kolejnymi ich parami, a przez to również pomiędzy t_i a t_j.

Rozgłaszanie przyczynowe – przykłady



a) Dostarczenie zgodne z porządkiem przyczynowym, *m* przed *m'*

b) Dostarczenie łamiące porządek przyczynowy, *m'* przed *m*

Na rysunku a) pokazano przykład dostarczenia przez proces *p* dwóch wiadomości, *m* i *m'*. Ponieważ wysłanie wiadomości *m'* następuje w procesie *r* po zdarzeniu odebrania wiadomości *m*, więc zgodnie definicją porządku przyczynowego (warunek 1) wiadomość *m'* jest przyczynowo zależna od *m* i jako taka powinna być dostarczona w każdym procesie *później* niż wiadomość *m*. Proces *p* właśnie tak dostarcza wiadomości *m* i *m'*, dzięki czemu porządek przyczynowy zostaje zachowany.

Na rysunku b) zaś widać przykład złamania porządku przyczynowego. Wiadomość *m'* zależna przyczynowo od *m* została dostarczona w procesie *p* *przed* wiadomością *m*; porządek przyczynowy został tutaj złamany.

Ze względu na to, że własność uporządkowania przyczynowego jest ściśle silniejsza niż własność uporządkowania FIFO, zachowanie porządku przyczynowego implikuje zachowanie porządku FIFO.

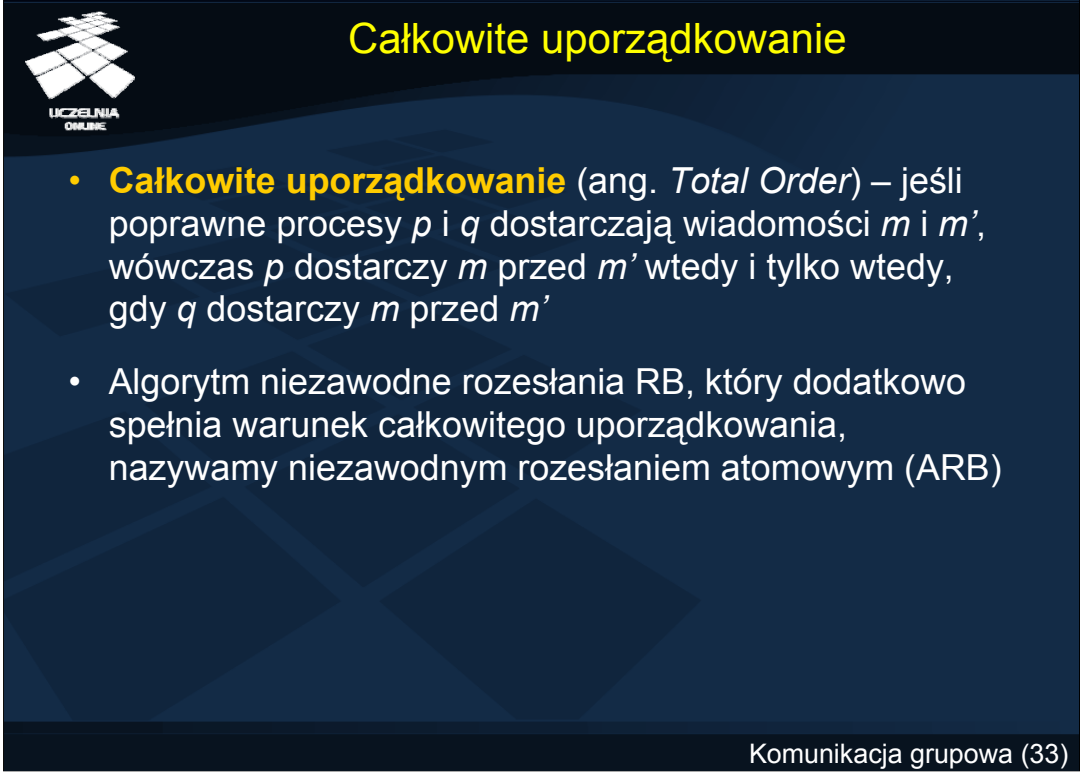
O wiadomościach, pomiędzy którymi nie zachodzi zależność przyczynowa mówi się, że są *przyczynowo niezależne*. Wiadomości przyczynowo niezależne mogą być dostarczone w dowolnej kolejności, a co za tym idzie – różne procesy mogą dostarczyć dwie (lub więcej) niezależne przyczynowo wiadomości w różnej kolejności. Wykonanie przykładu ilustrującego to pozostawia się Studentowi jako ćwiczenie.

Algorytm Przyczynowy-RB realizuje niezawodne rozgłoszenie z zachowaniem dodatkowo przyczynowego uporządkowania wiadomości. Algorytm opiera się na algorytmie FIFO-RB.

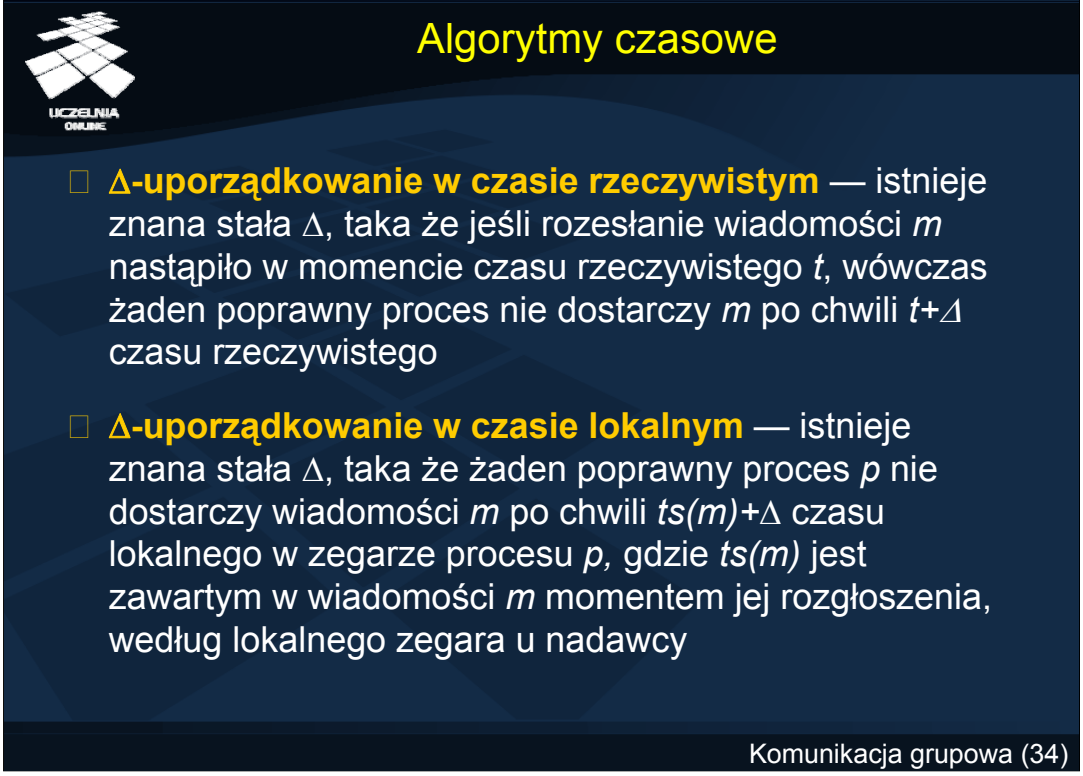
Główna idea algorytmu polega na rozsyłaniu danej wiadomości wraz *z całą sekwencją wiadomości ją poprzedzających*, czyli wiadomości, które proces nadawcy dostarczył przed rozesłaniem swojej. Ponieważ sekwencja jest uporządkowana, kolejność wiadomości w niej odzwierciedla porządek przyczynowy.

W momencie dostarczenia danej wiadomości proces dołącza ją do sekwencji *poprzednie*. Następnie wykorzystuje tę sekwencję, gdy sam rozgłasza swoją wiadomość. Sekwencja ta niesie wówczas informację o wiadomościach, które proces nadawcy odebrał przed rozesłaniem własnej, a więc o wiadomościach poprzedzających przyczynowo rozsyłaną właśnie wiadomość. Sekwencja może zostać wyzerowana po rozgłoszeniu dowolnego komunikatu, ponieważ rozgłoszenie spowoduje przekazanie do wszystkich innych procesów informacji o uporządkowaniu komunikatów zaobserwowanym lokalnie.

/*sekwencja wiadomości, które *p* C-dostarczył od momentu swojego ostatniego C-rozgłoszenia*/



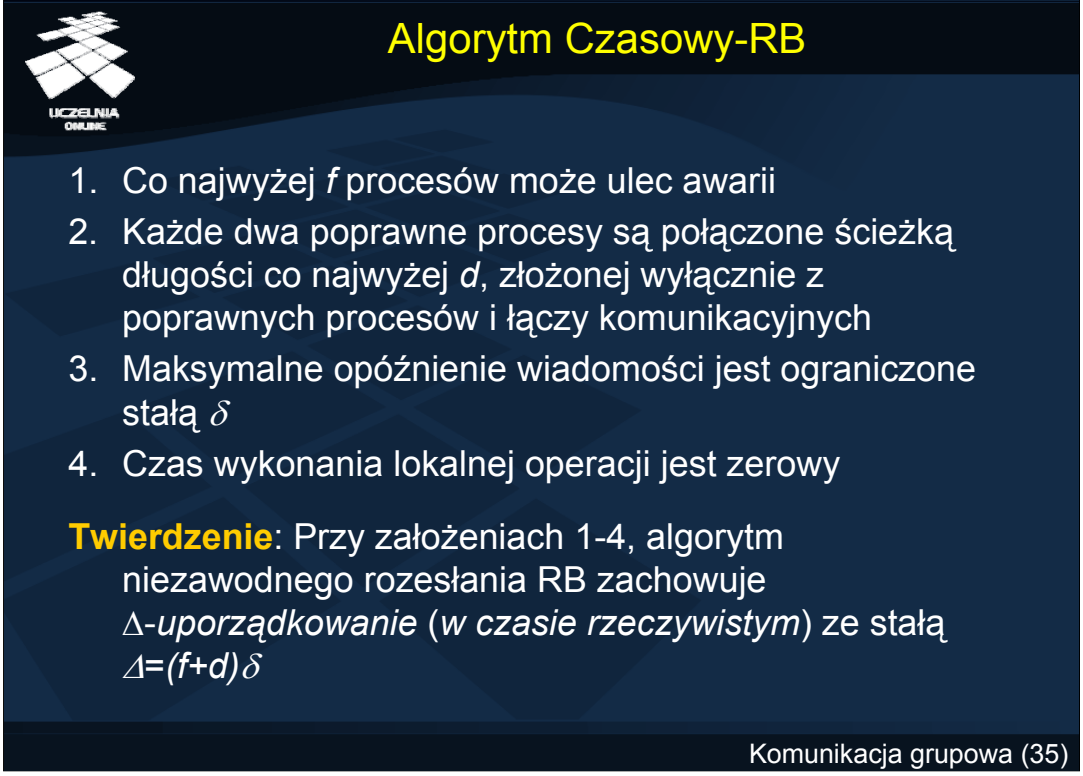
Własność **całkowitego uporządkowania** oznacza, że wszystkie wiadomości są dostarczane do wszystkich procesów w identycznej kolejności. Algorytm niezawodnego rozgłoszenia RB, który dodatkowo zachowuje całkowite uporządkowanie wiadomości, nazywamy niezawodnym rozgłoszeniem atomowym, w skrócie rozgłoszeniem atomowym RB lub ARB, od angielskiej nazwy Atomic Reliable Broadcast.



Algorytmy czasowe umożliwiają porządkowanie wiadomości w czasie rzeczywistym. Oparte są na założeniach o istnieniu maksymalnego, nieprzekraczalnego opóźnienia przy przesyłaniu wiadomości w systemie komunikacyjnym. Wyróżnia się dwa poziomy gwarancji algorytmów czasowych:

Uporządkowanie (ang. *Timeliness*) **w czasie rzeczywistym** porządkuje wiadomości *względem momentu ich wysłania mierzonego w czasie rzeczywistym*, a więc bezwzględnym. Zrealizowanie go jest jednak możliwe tylko wtedy, gdy w systemie istnieje globalny zegar czasu rzeczywistego.

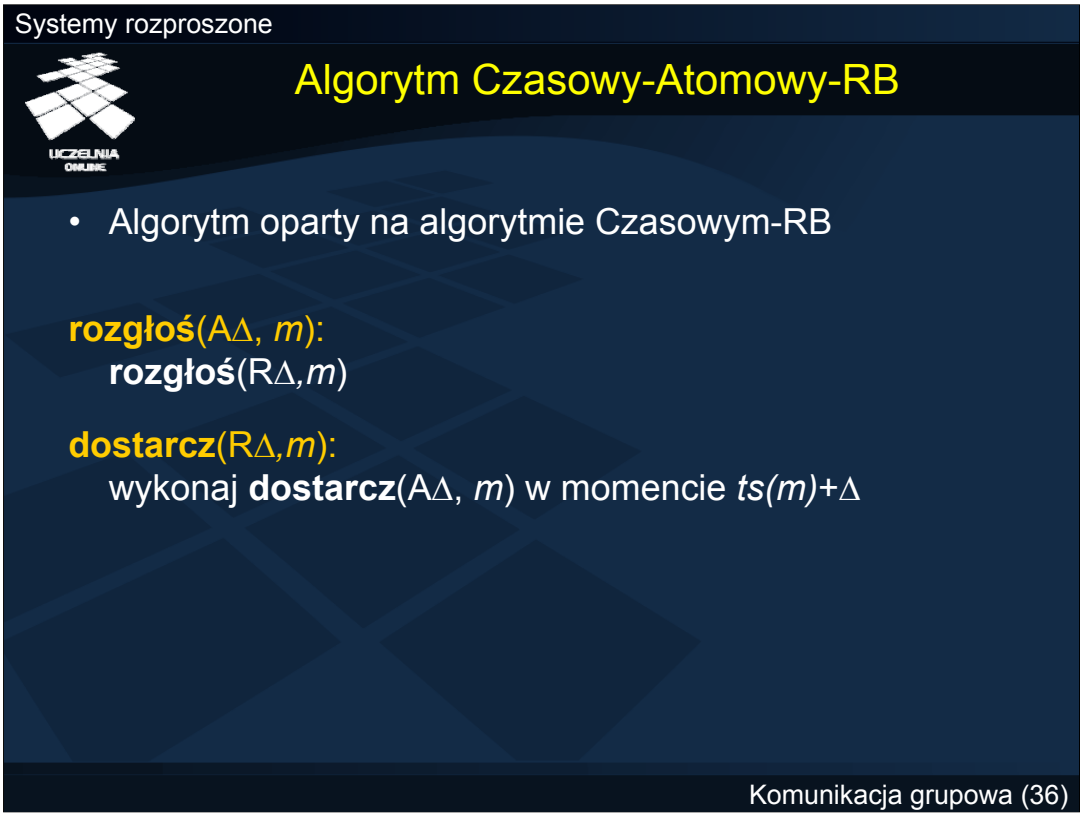
Uporządkowanie w czasie lokalnym porządkuje wiadomości względem *lokalnego zegara u nadawcy*. Każdej rozsyłanej wiadomości przypisywany jest przez nadawcę tzw. „stempel czasowy” (ang. *Timestamp*), oznaczony na slajdzie przez *ts*(*m*).



Powyższy algorytm będzie w definicji algorytmów oznaczany przez R

Załóżmy, że podsieć komunikacyjna ma następujące właściwości:
1. Co najwyżej *f* procesów może ulec awarii.
2. Każde dwa poprawne procesy są połączone ścieżką długości co najwyżej *d*, złożonej wyłącznie z poprawnych procesów i łączy komunikacyjnych.
3. Maksymalne opóźnienie wiadomości jest ograniczone stałą δ.
4. Czas wykonania lokalnej operacji jest zerowy.

Można udowodnić następujące **twierdzenie**:
Przy założeniach 1-4, algorytm niezawodnego rozesłania RB zachowuje Δ-*uporządkowanie* (*w czasie rzeczywistym*) ze stałą $\Delta=(f+d)\delta$.

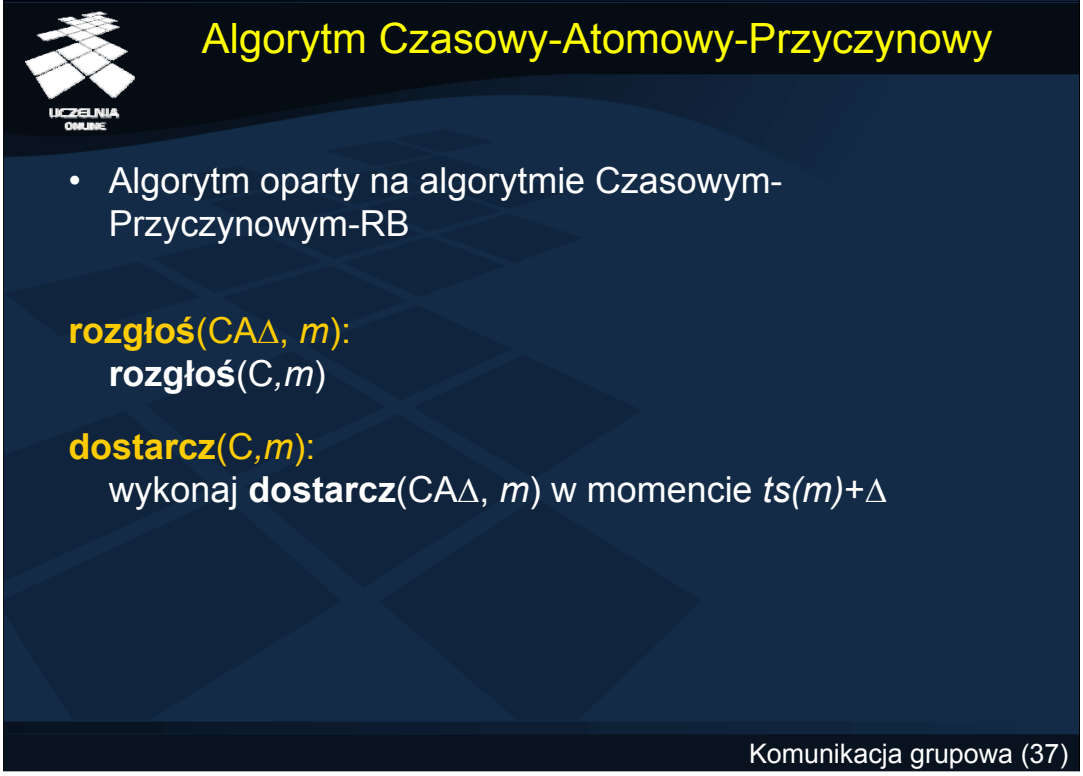


W powyższym algorytmie wiadomość jest dostarczana w chwili $ts(m)+$

o

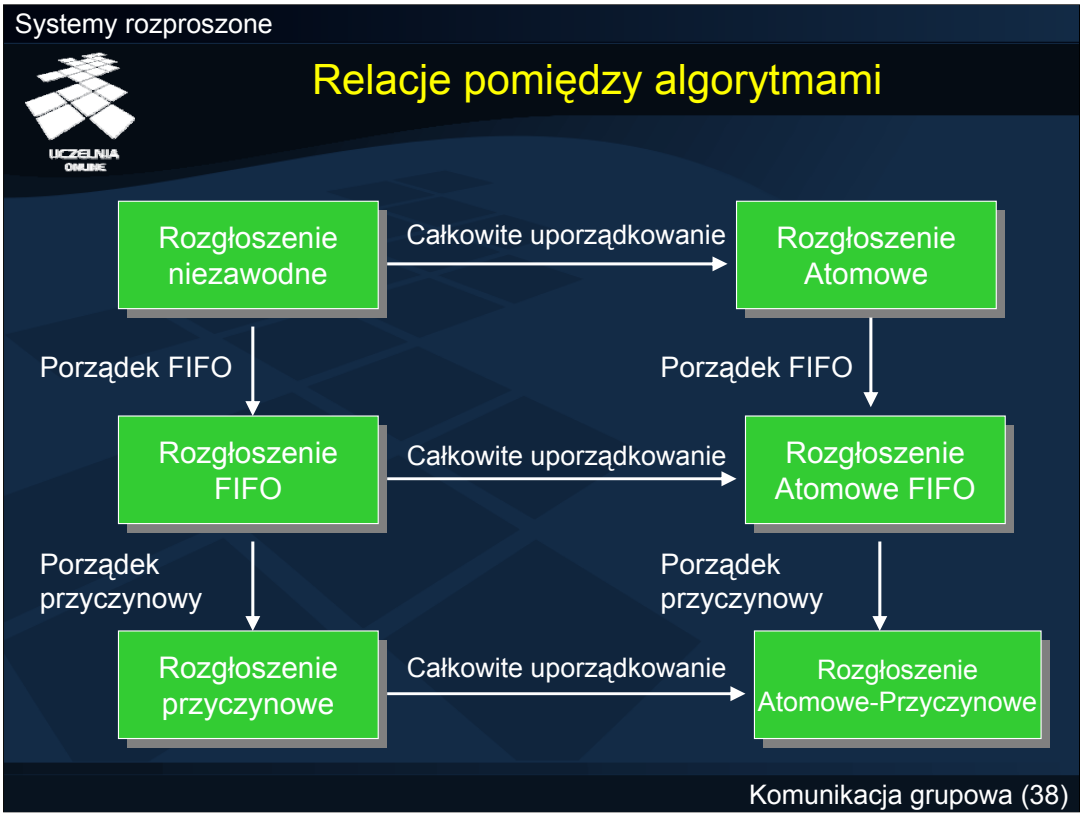
Algorytm zachowuje całkowite uporządkowanie wiadomości, ponieważ można uszeregować liniowo stemple czasowe wszystkich otrzymywanych wiadomości i w każdym procesie uszeregowanie to będzie jednakowe. Dodanie do stempli

czasowych stałej



Powyższy algorytm zachowuje całkowite uporządkowanie wiadomości i jednocześnie uporządkowanie przyczynowe. Opiera się na działającym poniżej czasowym algorytmie przyczynowym-RB, gwarantującym uporządkowanie przyczynowe i dodatkowe gwarancje czasowe, i dodaje własność uporządkowania całkowitego. Skrót „CA” w nazwie algorytmu pochodzi od pełnej angielskiej nazwy *Causal Atomic*, oznaczającej „przyczynowy-atomowy”.

Należy zauważyć, że algorytm bazowy, czasowy przyczynowy-RB, jest po prostu algorytmem przyczynowym przedstawionym wcześniej, lecz bazującym na czasowym algorytmie RB.



Rysunek ukazuje, jak można w sposób warstwowy konstruować algorytmy dostarczające coraz bardziej wymagających gwarancji.

Podstawowy mechanizm rozgłoszenia niezawodnego może zostać wykorzystany jako podstawa dla wszystkich pozostałych. Algorytm z dodatkową własnością uporządkowania FIFO może z kolei posłużyć jako podstawa do implementacji algorytmu z uporządkowaniem przyczynowym. Ponadto, z prawej strony pokazano, że niezależnie od tych dwóch własności dany algorytm może zostać wyposażony w gwarancję całkowitego uporządkowania.

Literatura

- G. V. Chockler, I. Keidar, R. Vitenberg: Group Communication Specifications: A Comprehensive Study, ACM Computing Surveys, 2001
- Sergio Mena, André Schiper, Paweł T. Wojciechowski: A Step Towards a New Generation of Group Communication Systems. Middleware 2003, strony 414-432
- A. S. Tanenbaum: Systemy rozproszone – zasady i paradygmaty, PWN, 2006