

Problem Konsensusu: Fundament Porozumienia

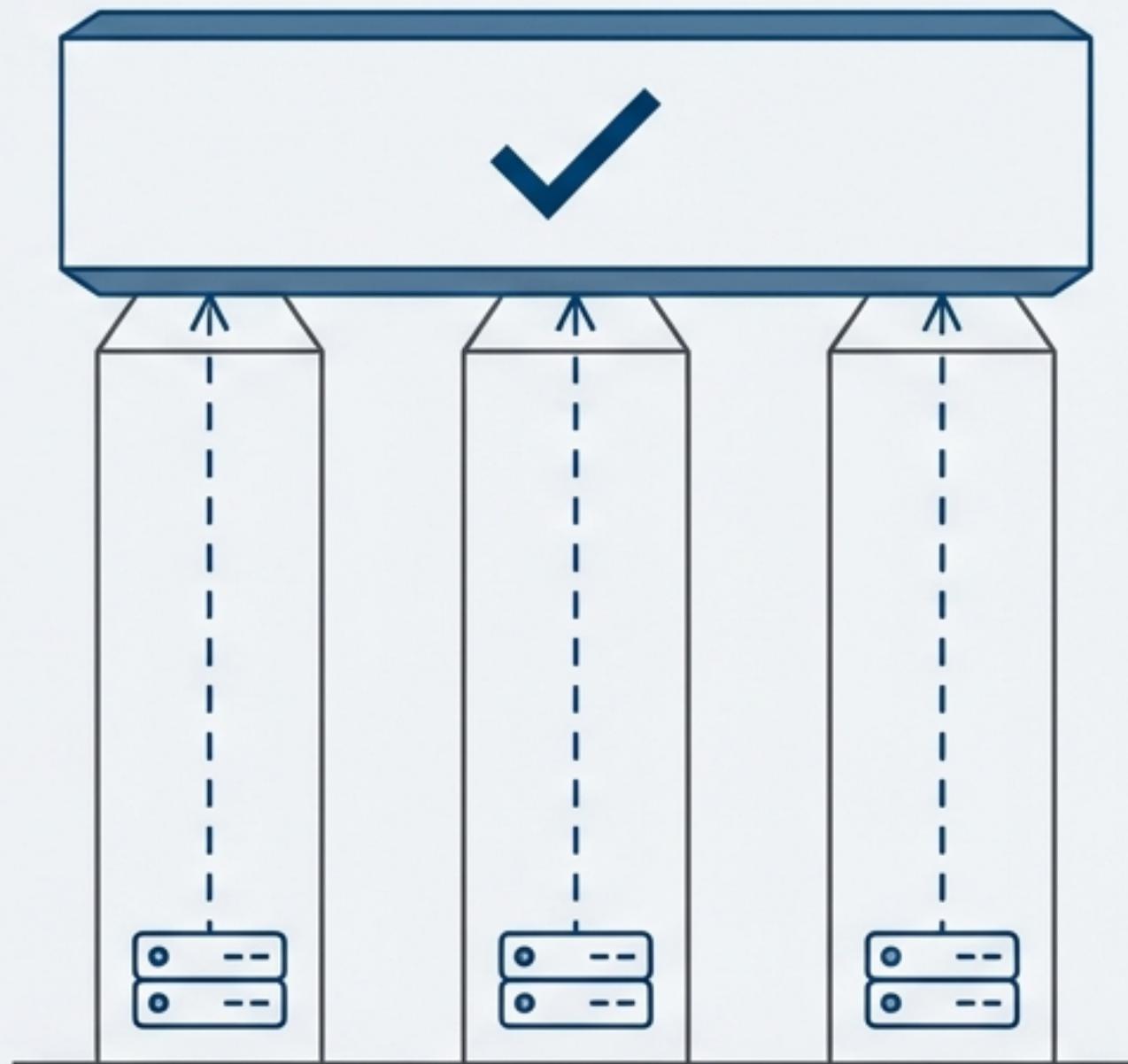
Systemy rozproszone muszą uzgodnić jedną, wspólną wartość. Jest to jeden z najbardziej fundamentalnych problemów w informatyce rozproszonej.

Przykład: Transakcja bankowa, taka jak rezerwacja lotu, musi zostać albo w pełni zatwierdzona (commit), albo w pełni odrzucona (abort) przez wszystkie zaangażowane systemy (linię lotniczą, bank, system rezerwacji).

Krytyczny wymóg: Wszystkie poprawne procesy muszą podjąć tę samą decyzję, aby zachować spójność bazy danych.

Nie ma miejsca na niespójność. Rzeczywiste systemy, takie jak gieldy papierów wartościowych, bazy danych w chmurze i systemy bankowe, polegają na jednogłośnym porozumieniu.

Wyzwanie pojawia się, gdy procesy mogą ulegać awariom, np. poprzez nagłe zatrzymanie działania.



Model Asynchroniczny: Prawa Fizyki Systemów

- Całkowicie asynchroniczne przetwarzanie jest kluczowym założeniem dowodu niemożliwości.
- Brak założeń dotyczących względnej prędkości procesów – mogą działać dowolnie szybko lub wolno.
- Brak założeń dotyczących opóźnień w dostarczaniu wiadomości – wiadomości ostatecznie docierają, ale nie wiadomo kiedy. Mogą być dostarczane w innej kolejności niż zostały wysłane.
- Brak dostępu do zsynchronizowanych zegarów, co uniemożliwia stosowanie algorytmów opartych na limitach czasu (timeout).
- Kluczowy problem: Nie można odróżnić procesu, który uległ awarii (zatrzymał się całkowicie), od procesu, który działa bardzo wolno. Ta niepewność leży u podstaw niemożliwości.



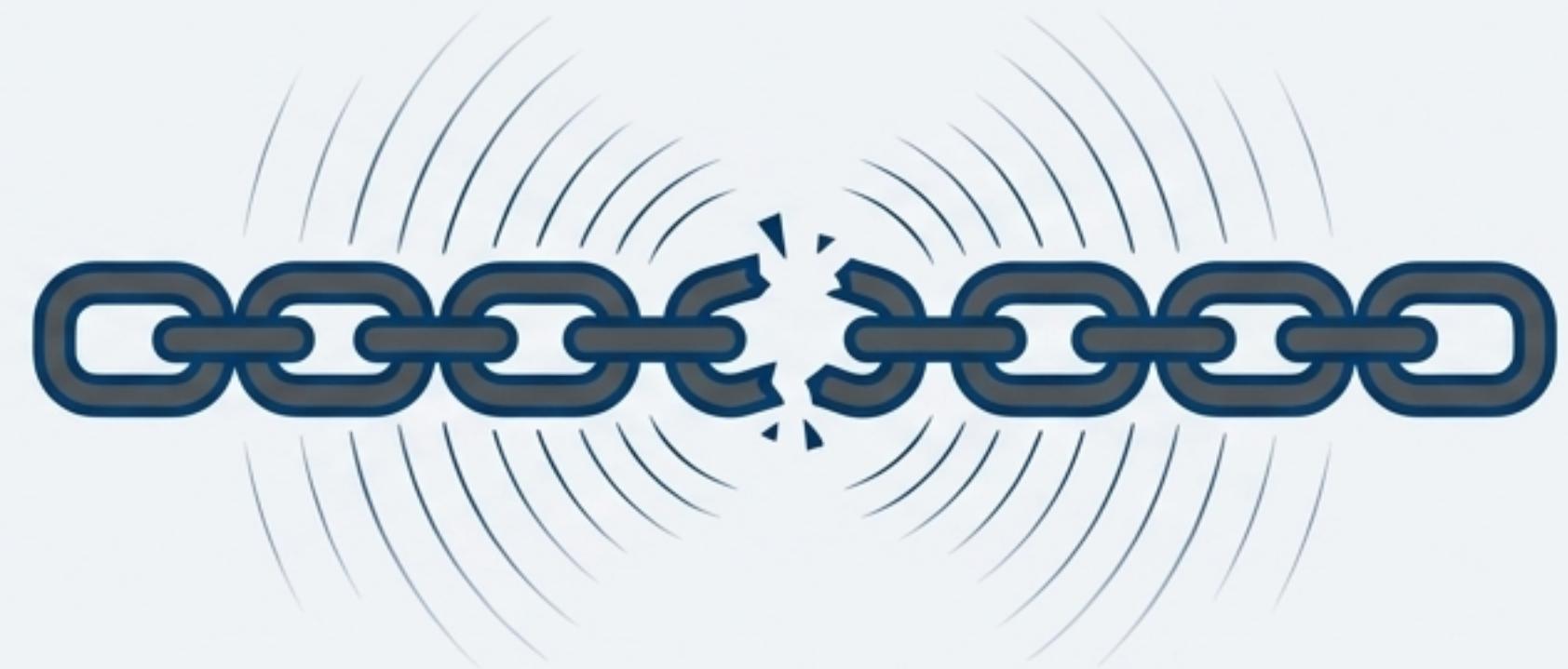
Brak globalnego czasu



Nieograniczone opóźnienie

Twierdzenie FLP: Fundamentalne Ograniczenie

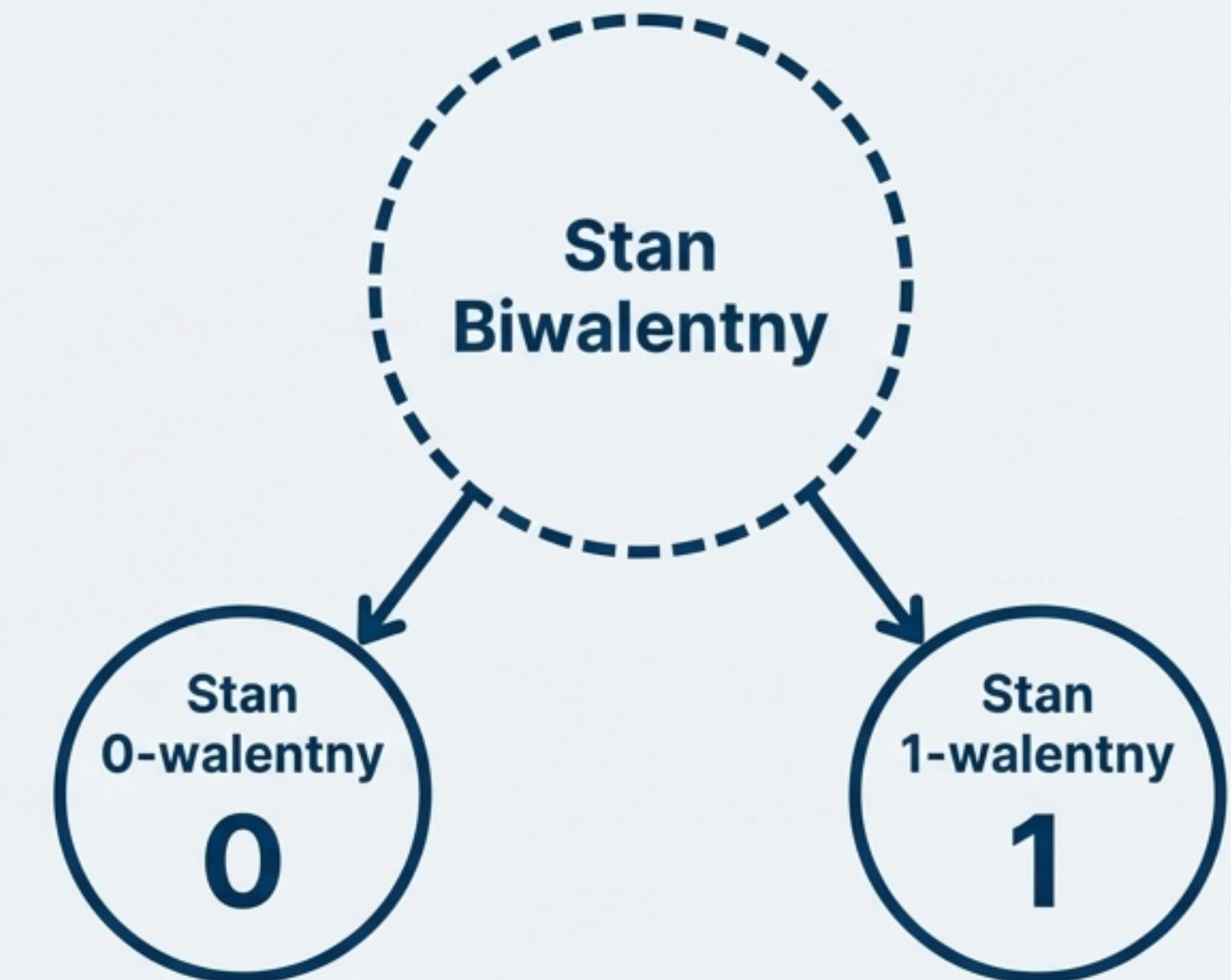
- Opublikowane w 1985 roku przez Michaela J. Fischer, Nancy A. Lynch i Michaela S. Patersona.
- Tytuł pracy: "Impossibility of Distributed Consensus with One Faulty Process".
- Główne twierdzenie: Żaden deterministyczny protokół konsensusu nie może zagwarantować osiągnięcia porozumienia w całkowicie asynchronicznym systemie, jeśli może wystąpić choćby jedna awaria procesu.
- Każdy protokół tego problemu ma możliwość niezatrzymania się (nontermination), nawet przy jednym wadliwym procesie.
- Pojedyncza awaria w niefortunnym momencie może teoretycznie zawiesić cały system na zawsze. Jest to matematycznie udowodnione, fundamentalne ograniczenie.



“Impossibility of Distributed Consensus with One Faulty Process”

Stany Systemu: Uniwarentne vs. Biwalentne

- Konfiguracja (C): Migawka systemu zawierająca wewnętrzny stan każdego procesu oraz zawartość bufora wiadomości.
- Stan uniarentny: Stan, z którego osiągalny jest tylko jeden wynik decyzji (0 lub 1). Dzieli się na:
 - 0-walentny: Wszystkie możliwe wykonania prowadzą do decyzji '0'.
 - 1-walentny: Wszystkie możliwe wykonania prowadzą do decyzji '1'.
 - Wynik jest już przesądzony, nawet jeśli nie wszystkie procesy o tym wiedzą.
- Stan biwalentny: Stan, z którego można osiągnąć zarówno decyzję '0', jak i '1'.
 - Wynik jest wciąż nieokreślony. System znajduje się w krytycznym punkcie, w którym może pójść w obie strony.



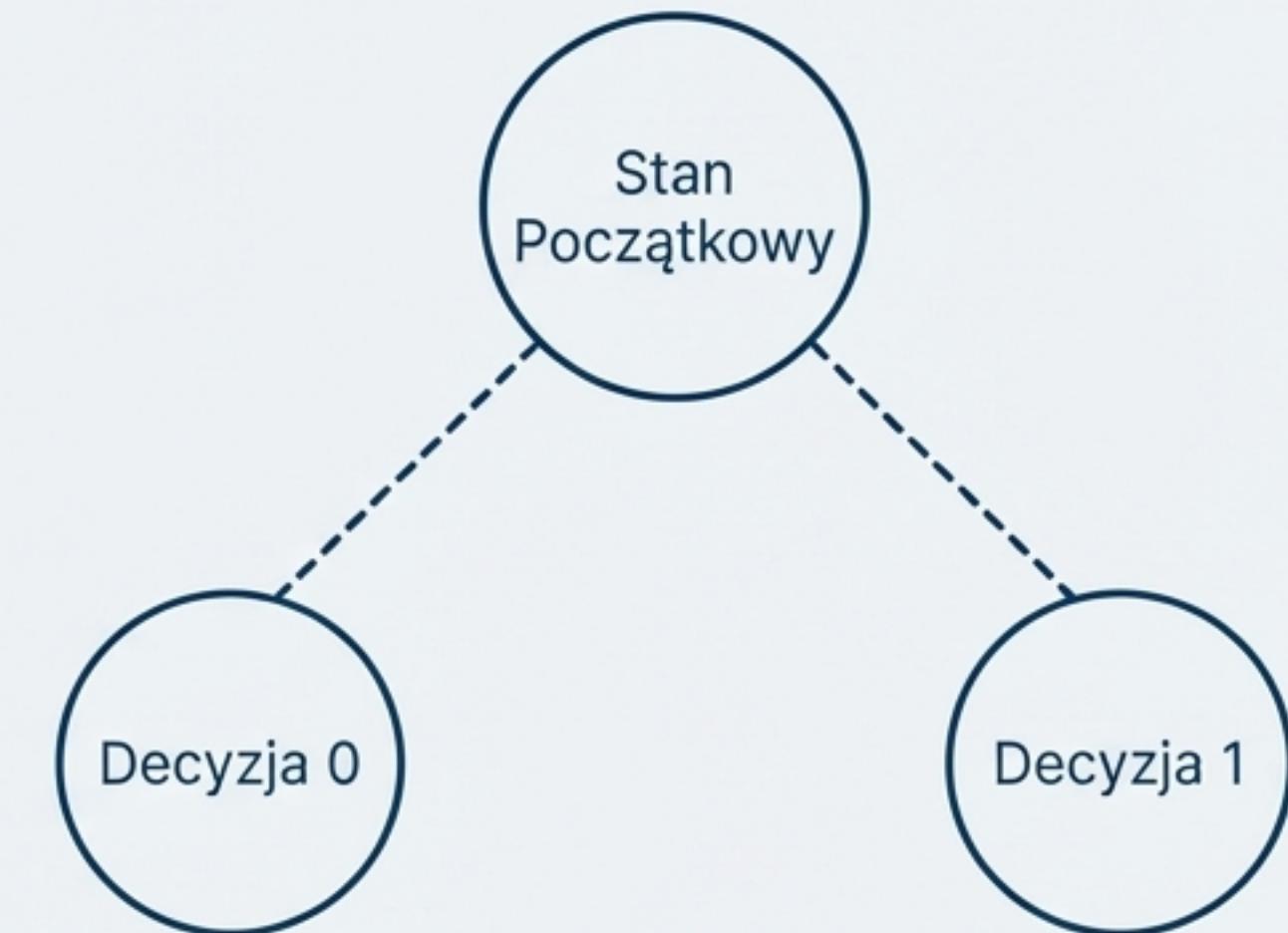
Dowód Niemożliwości (Część 1): Punkt Wyjścia

Lemat 2*: Każdy nietrywialny protokół konsensusu musi mieć co najmniej jedną biwalentną konfigurację początkową.

****Uzasadnienie**:**

- Protokół musi dopuszczać zarówno decyzję '0', jak i '1' (dla różnych danych wejściowych).
- Gdyby wszystkie konfiguracje początkowe były uniwarentne, wynik byłby z góry określony przed rozpoczęciem komunikacji, co zaprzecza idei konsensusu.
- Musi istnieć para sąsiadujących konfiguracji początkowych (różniących się tylko stanem jednego procesu), z których jedna jest 0-walentna, a druga 1-walentna. Dowód pokazuje, że to implikuje istnienie stanu biwalentnego.

Ten początkowy stan niepewności jest fundamentem, na którym opiera się reszta dowodu.



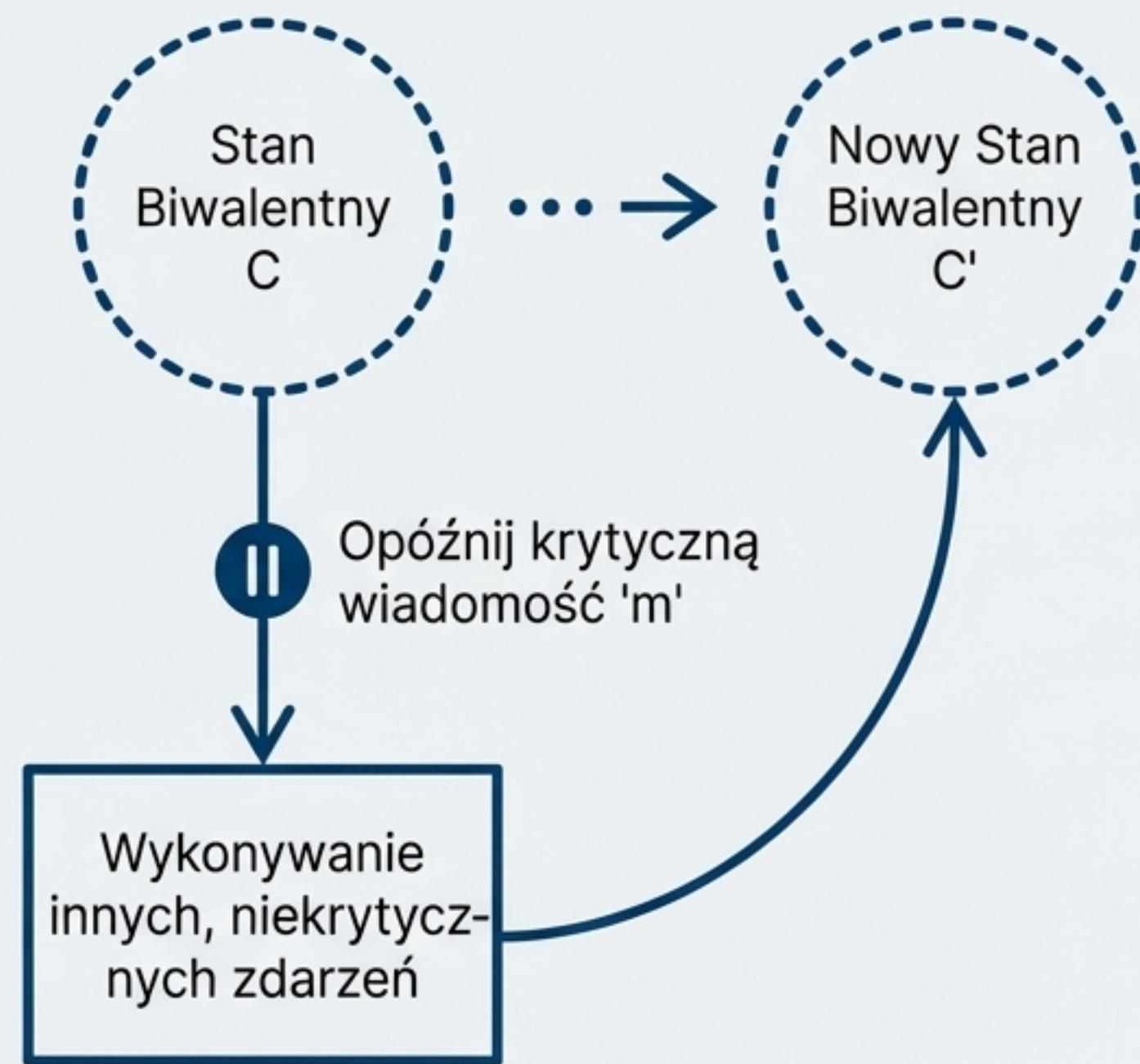
Dowód Niemożliwości (Część 2): Wieczna Niezdecydowanie

Lemat 3: Z każdej konfiguracji biwalentnej 'C' zawsze można znaleźć inną konfigurację biwalentną 'C'', którą można z niej osiągnąć.

Mechanizm:

1. Znajdź "krytyczne" zdarzenie 'e' (dostarczenie wiadomości 'm' do procesu 'p'), które, jeśli zostanie wykonane, przekształci system w stan uniwarentny.
2. Zamiast dostarczać wiadomość 'm', opóźnij ją.
3. Pozwól innym procesom wykonać swoje kroki.
4. Dowód (Lemat 3) pokazuje, że zawsze istnieje taka sekwencja kroków (harmonogram), która pozwala na opóźnienie krytycznego zdarzenia 'e' i doprowadzenie systemu do nowego stanu biwalentnego.

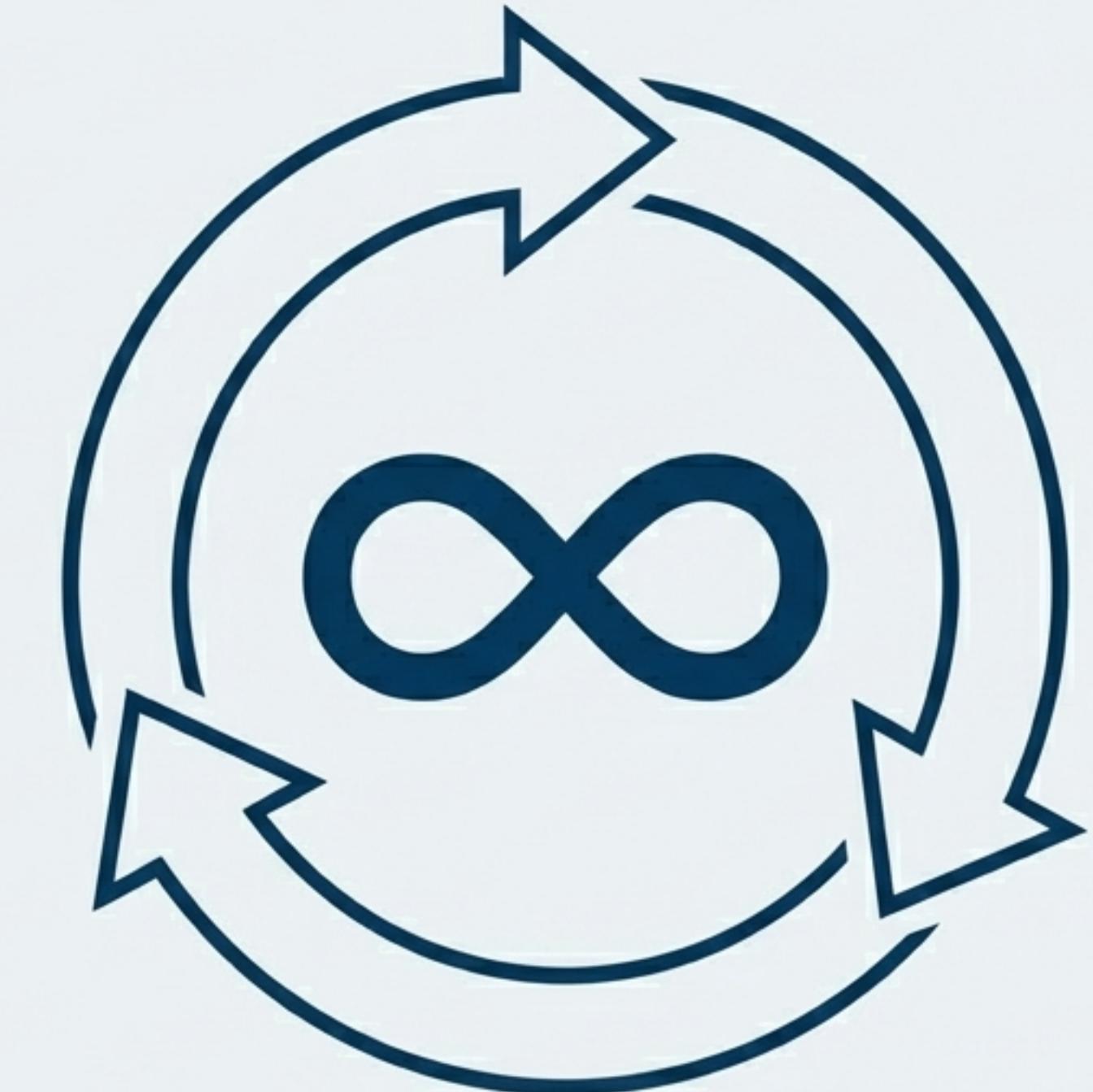
Powtarzając ten proces, można skonstruować nieskończone wykonanie (run), w którym system nigdy nie opuszcza stanu biwalentnego.



Wniosek z Dowodu: Teoretyczny Sabotaż

- Konstrukcja dowodu tworzy 'dopuszczalne wykonanie' (admissible run), w którym:
 - Co najwyżej jeden proces ulega awarii (w tym przypadku żaden, jest on tylko nieskończony opóźniany).
 - Wszystkie wysłane wiadomości są ostatecznie dostarczane (poprzez konstrukcję 'kolejki procesów' w dowodzie, która zapewnia, że każdy proces w końcu otrzymuje swoje wiadomości).
- Jednak to wykonanie nigdy nie osiąga stanu decyzyjnego, ponieważ system jest utrzymywany w stanie biwalentnym w nieskończoność.
- To łamie kluczowy warunek protokołu konsensusu: **zatrzymanie się** (termination).

Dla każdego deterministycznego algorytmu istnieje pechowy scenariusz wykonania, który nigdy nie prowadzi do decyzji. Niemożliwość została udowodniona.



Dlaczego Cokolwiek Działa w Praktyce?

Wyniki FLP nie oznaczają, że problemu nie da się "rozwiązać" w praktyce. Wskazują one na potrzebę udoskonalenia modeli obliczeniowych.

Praktyczne systemy obchodzą niemożliwość poprzez osłabienie założeń czystego modelu asynchronicznego.

- Najczęstsze obejście: Limity czasu (timeouts).
 - Proces czeka na wiadomość przez określony czas (np. 30 sekund). Jeśli jej nie otrzyma, zakłada, że nadawca uległ awarii.
 - To wprowadza założenie o ograniczonym opóźnieniu, które jest zabronione w modelu FLP.

Jest to pragmatyczny kompromis: poświęcamy matematyczną czystość na rzecz funkcjonalności w świecie rzeczywistym.



Pragmatyczne
założenie o
czasie

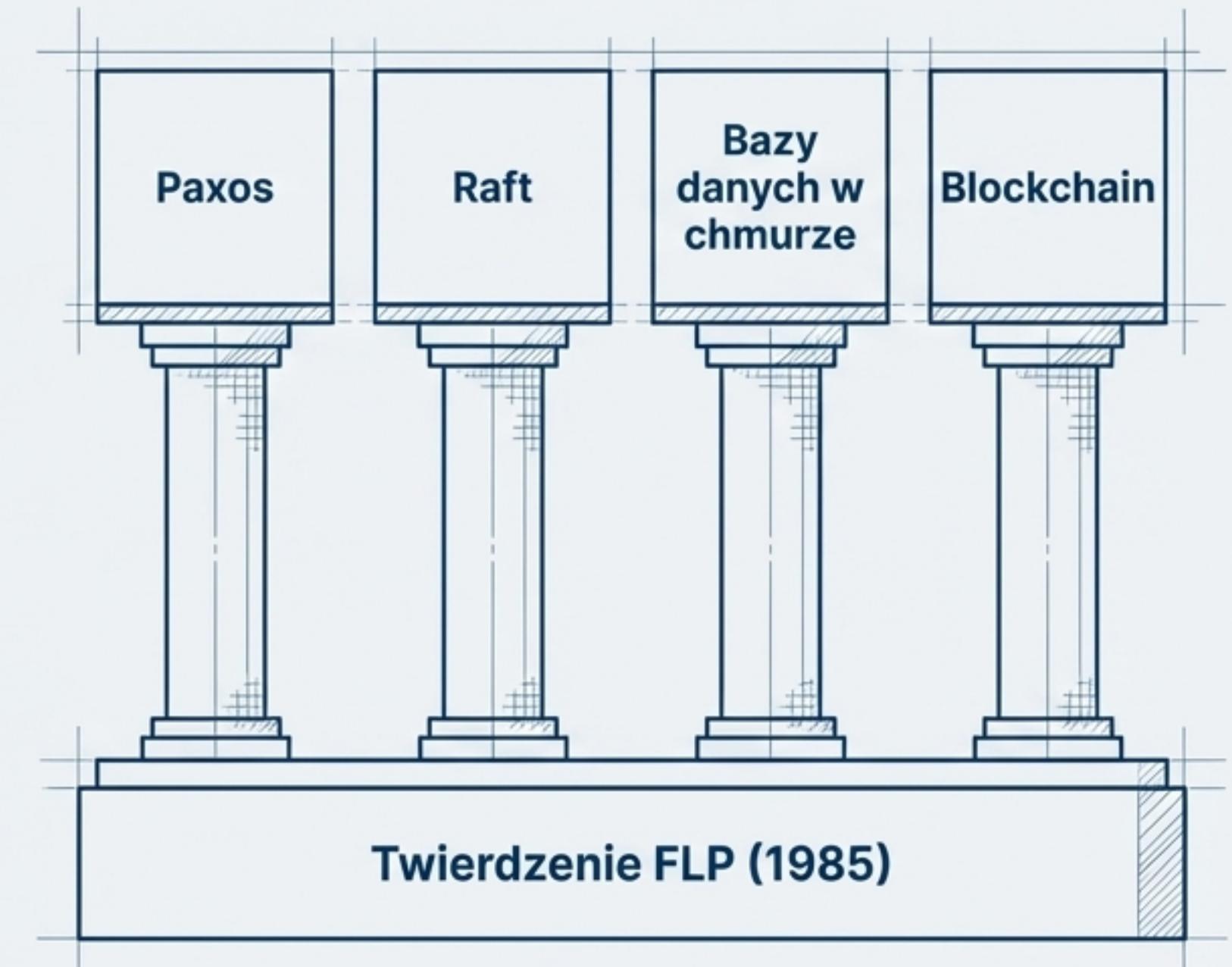
Praktyczne Obejścia: Detektory Awarii

- **Detektor awarii** to moduł, który 'podejrzewa' inne procesy o awarię.
- Działa na zasadzie prawdopodobieństwa, a nie pewności. Może się mylić.
 - Przykład: Jeśli proces nie odeśle 'sygnału życia' (heartbeat) w oczekiwany czasie, detektor może go uznać za uszkodzony.
- Ryzyko: Detektor może fałszywie oskarżyć powolny, ale działający proces o awarię.
- Jednak nawet niedoskonały detektor awarii jest wystarczający do przełamania niemożliwości FLP. Pozwala systemowi podjąć decyzję, nawet jeśli opiera się ona na świadomym zgadywaniu.



Dziedzictwo i Wpływ: Budowanie na Niemożliwości

- FLP precyzyjnie zidentyfikowało **całkowity brak założeń czasowych** jako kluczowy czynnik uniemożliwiający konsensus.
- Praca kontrastuje ten wynik z **Problemem Bizantyjskich Generałów**, który jest rozwiązywalny w systemach synchronicznych (z ograniczeniami czasowymi).
- Zrozumienie, co jest niemożliwe, pozwoliło inżynierom budować to, co jest możliwe.
- Twierdzenie FLP stało się fundamentem dla algorytmów takich jak **Paxos** i **Raft**, które stanowią kręgosłup nowoczesnych usług chmurowych (Google, Amazon, Microsoft).
- Krytyczne znaczenie dla technologii **blockchain**, gdzie tysiące anonimowych komputerów muszą osiągnąć globalny konsensus.



Pytanie do Ciebie: Problem Bizantyjskich Generałów

- Twierdzenie FLP dotyczy awarii typu 'crash-stop', gdzie proces po prostu przestaje działać.
- **A co jeśli proces, zamiast się wyłączyć, zacznie działać w sposób złośliwy, wysyłając sprzeczne, fałszywe informacje do różnych części systemu?**
- W pracy FLP autorzy zaznaczają: "Nie rozważamy awarii bizantyjskich (...) w których wadliwe procesy mogą całkowicie oszaleć, być może nawet wysyłając wiadomości zgodnie z jakimś złośliwym planem."
- To jest właśnie słynny Problem Bizantyjskich Generałów – trudniejsza klasa problemów wymagająca innych rozwiązań.

