

PYTANIE 1: Automaty i klasy języków (AISDI)

Porównać “siłę wyrazu” automatu skończonego, automatu ze stosem oraz maszyny Turinga. Jakie klasy języków rozpoznaje każdy z nich?

Porównanie siły wyrazu

Siła wyrazu (expressive power) — klasa języków, które automat rozpoznaje. Im szersza klasa, tym większa siła:

$FA \subset PDA \subset LBA \subset TM$

- **FA < PDA:** FA nie rozpoznaje $a^n b^n$ (brak pamięci do liczenia), PDA tak (stos zlicza).
- **PDA < LBA:** PDA nie rozpoznaje $a^n b^n c^n$ (stos zużyty po a/b), LBA tak (taśma ogr. R/W).
- **LBA < TM:** LBA ograniczona do $|w|$ komórek, TM ma nieskończoną taśmę.

Hierarchia Chomsky’ego (1956)

Typ 0: Rek. przeliczalne (TM) \supset Typ 1: Kontekstowe (LBA) \supset Typ 2: Bezkontekstowe (PDA) \supset Typ 3:

FA — Typ 3: Języki regularne

$M = (Q, \Sigma, \delta, q_0, F)$. **Pamięć:** brak — tylko stan. **DFA \equiv NFA**. Równoważne: wyrażenia regularne, gramatyki regularne. Przykłady: identyfikatory, podzielność. Nie: $a^n b^n$, nawiasy, palindromy.

PDA — Typ 2: Języki bezkontekstowe

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. **Pamięć:** stos LIFO. **DPDA \subset NPDA!** Równoważne: gramatyki bezkontekstowe (CFG). Przykłady: $a^n b^n$, nawiasy, ww^R . Nie: $a^n b^n c^n$, ww .

LBA — Typ 1: Języki kontekstowe

TM z taśmą ograniczoną do $|w|$. **DLBA =? NLBA** — problem otwarty! Równoważne: gramatyki kontekstowe (CSG): $\alpha \rightarrow \beta, |\alpha| \leq |\beta|$. Przykłady: $a^n b^n c^n$, ww .

TM — Typ 0: Rekurencyjnie przeliczalne

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$. **Pamięć:** taśma \propto R/W. **DTM \equiv NTM** (moc). Teza Churcha-Turinga: TM modeluje każde obliczenie. Nie: komplement problemu stopu.

Tabela porównawcza

Cecha	FA	PDA	LBA	TM
Pamięć	Brak	Stos (LIFO)	Taśma ogr. (R/W)	Taśma \propto (R/W)
Klasa języków	Regularne	Bezkontekstowe	Kontekstowe	Rek. przeliczalne
DET = NIEDET?	TAK	NIE	Otwarte!	TAK (moc)
Domknięcie n/\neg	TAK/TAK	NIE/NIE	TAK/TAK	TAK/NIE
Zastosowanie	Leksery	Parsery	Weryfikacja ogr.	Obliczenia ogólne

Etymologia nazw

Automaty: *Finite Automaton* — „skończony” = skończona liczba stanów (cała pamięć to stan). *Pushdown Automaton* — „pushdown” od spring-loaded tray dispenser (dozownik tac w stołówce: push down = zepchnij na stos). *LBA* — taśma liniowo proporcjonalna do wejścia (Myhill 1960, Kuroda 1964). *Maszyna Turinga* — Alan Turing (1936, „On Computable Numbers”), formalizacja obliczalności; odpowiedź na Entscheidungsproblem Hilberta; złamał Enigmę w WWII. *Hierarchia Chomsky’ego* — Noam Chomsky (MIT, 1956), lingwista; hierarchia gramatyk dla języków naturalnych okazała się fundamentem informatyki.

Języki: *Regularne* — od „regular expressions” (Kleene 1956); „regular” = podlegające stałej regule (łac. reguła). *Bezkontekstowe (Context-Free)* — produkcje $A \rightarrow \alpha$ stosowane BEZ patrzenia na kontekst wokół A; nieterminal przepisany niezależnie od otoczenia. *Kontekstowe (Context-Sensitive)* — produkcje $\alpha A \beta \rightarrow \alpha \gamma \beta$: przepisanie A ZALEŻY od kontekstu α i β . *Rekurencyjnie przeliczalne (Recursively Enumerable)* — istnieje TM wyliczająca (enumerate) wszystkie słowa języka; „rekurencyjnie” = przez procedurę obliczeniową (ale może nie zatrzymać się na nie-członkach).

Jak zapamiętać

- **„Raz Bardzo Kolorowy Rekin”** — Regularny \subset Bezkontekstowy \subset Kontekstowy \subset Rek.przeliczalny
- **Pamięć:** Brak \rightarrow Stos \rightarrow Taśma ogr. \rightarrow Taśma ∞
- **Kontrprzykłady:** $a^n b^n$ łamie FA, $a^n b^n c^n$ łamie PDA
- **Kontekst nazw:** Context-Free = A przepisane bez kontekstu; Context-Sensitive = kontekst $\alpha A \beta$ decyduje

PYTANIE 2: Algorytmy najkrótszej ścieżki (AISDI)

****Omówić i porównać algorytmy: Dijkstry, Bellmana-Forda, A*.****

Dijkstra — zachłanny, SSSP

Ograniczenie: wagi ≥ 0 . **Idea:** Relaksacja krawędzi; zawsze przetwarzaj wierzchołek o najmniejszym $d[v]$. **Złożoność:** $O(V^2)$ z tablicą, $O((V+E) \log V)$ z kopcem, $O(V \log V + E)$ z kopcem Fibonacciego. **Dlaczego nie ujemne wagi?** Raz oznaczony wierzchołek nie jest rewidowany — ujemna krawędź może go poprawić.

Bellman-Ford — programowanie dynamiczne, SSSP

Zaletą: obsługuje ujemne wagi + **wykrywa cykle ujemne**. **Idea:** $|V|-1$ iteracji relaksacji WSZYSTKICH krawędzi. Jeśli w iteracji V nadal można poprawić \rightarrow cykl ujemny. **Złożoność:** $O(V \cdot E)$ — zawsze.

A* — heurystyczny, Single-Pair

Rozszerzenie Dijkstry: $f(n) = g(n) + h(n)$, gdzie $h(n)$ to heurystyka. **Wymóg:** h dopuszczalna (admissible): $h(n) \leq \text{rzeczywisty koszt } n \rightarrow \text{cel}$. Jeśli h spójna (consistent): $h(n) \leq w(n,m) + h(m)$ — optymalne. **Złożoność:** zależy od h ; najlepszy przypadek $O(V)$, najgorszy jak Dijkstra.

Porównanie

Cecha	Dijkstra	Bellman-Ford	A*
Typ	Zachłanny	Prog. dynamiczne	Heurystyczny
Problem	SSSP	SSSP	Single-pair
Ujemne wagi	NIE	TAK	NIE
Wykrywa cykle-	NIE	TAK	NIE
Złożoność	$O((V+E) \log V)$	$O(VE)$	Zależy od h

Etymologia

Dijkstra — Edsger W. Dijkstra (Holandia, 1959); pionier informatyki (Turing Award 1972). **Bellman-Ford** — Richard Bellman (twórca programowania dynamicznego) + Lester Ford Jr. (1956). **A*** — Hart, Nilsson, Raphael (Stanford, 1968); „A*” = ulepszona wersja algorytmu „A”. **Zachłanny (Greedy)** — algorytm „chciwie” bierze lokalnie najlepszą opcję. **SSSP** — Single-Source Shortest Path. **Programowanie dynamiczne** — Bellman wybrał „dynamic” by brzmiało imponująco dla polityków (nie miało związku z dynamiką!). **Heurystyka** — grec. „heuriskein” = znajdować (to samo co „Eureka!” Archimedes). **Relaksacja** — „rozluźnianie” górnego ograniczenia na odległość $d[v]$.

Jak zapamiętać

- **Dijkstra = chciwy**, bierze minimum — ale „nie patrzy wstecz” (stąd problem z ujemnymi wagami)
- **Bellman-Ford = brute force x (V-1)** — relaksuj wszystko, $V-1$ razy, bo najdłuższa ścieżka ma $V-1$ krawędzi
- **A* = Dijkstra + „GPS”** — heurystyka mówi w którą stronę jest cel

PYTANIE 3: Redundancja i normalizacja (BD2)

Omówić zagadnienia redundancji i normalizacji w relacyjnej bazie danych.

Redundancja — powtarzanie danych

Prowadzi do trzech **anomalii**: 1. **Wstawiania** — nie można dodać danych bez zbędnych powiązań 2. **Usuwania** — usunięcie rekordu kasuje niezwiązane informacje 3. **Modyfikacji** — zmiana jednej informacji wymaga wielu aktualizacji

Normalizacja — eliminacja redundancji przez dekompozycję

Zależność funkcyjna: $X \rightarrow Y$ — wartość X jednoznacznie określa Y.

Postacie normalne ($5NF \subset 4NF \subset BCNF \subset 3NF \subset 2NF \subset 1NF$)

1NF: Atomowe wartości, brak list/tablic w komórkach, istnieje klucz główny.

2NF: 1NF + każdy atrybut wtórny zależy od CAŁEGO klucza (dotyczy kluczy złożonych). - Naruszenie: NazwaKursu zależy tylko od KursID, nie od (StudentID, KursID).

3NF: 2NF + brak zależności przechodnich (atrybut wtórny nie zależy od innego wtórnego). - Naruszenie: StudentID \rightarrow WydziałID \rightarrow NazwaWydziału.

BCNF: Dla każdej nietrywialnej FD $X \rightarrow A$, X jest nadkluczem. Silniejsza niż 3NF.

4NF: BCNF + brak wielowartościowych zależności nietrywialnych.

Denormalizacja

Świadome wprowadzanie redundancji dla wydajności (mniej JOIN-ów). Stosowane w systemach analitycznych (OLAP), data warehousing.

Etymologia

Redundancja — łac. „redundantia” = nadmiar/przelewanie się. **Normalizacja** — Edgar F. Codd (IBM, 1970, „A Relational Model of Data”); 1NF–3NF w oryginalnej pracy. **BCNF** — Raymond Boyce + Codd (1974). **Anomalia** — grec. „anomalia” = nieregularność. **„Klucz, cały klucz i tylko klucz”** — parafraza przysięgi sądowej; przypisywana Coddowi. **Zależność funkcyjna** — jak funkcja mat.: X jednoznacznie wyznacza Y.

Jak zapamiętać

- „**Klucz, cały klucz i tylko klucz — tak mi dopomóż Codd**” — 1NF (klucz), 2NF (cały klucz), 3NF (tylko klucz)
- **3 anomalie:** Wstawianie, Usuwanie, Modyfikacja — „WUM”
- **BCNF:** jak 3NF, ale lewa strona FD zawsze nadklucz (bez wyjątku dla atrybutów pierwszych)

PYTANIE 4: Baza danych jako fundament systemów (BD2)

Dlaczego baza danych stanowi dobry fundament do budowy wielu systemów informatycznych?

1. Transakcyjność ACID

Właściwość	Znaczenie
Atomicity	Transakcja — albo cała, albo nic
Consistency	Spójny stan → spójny stan
Isolation	Równoległe transakcje nie interferują
Durability	Zatwierdzone zmiany przetrwają awarię

2. Niezależność danych (3-poziomowa architektura ANSI/SPARC)

- **Fizyczna:** zmiana indeksów/partycjonowania nie wpływa na aplikacje
- **Logiczna:** zmiana schematu minimalizuje wpływ na aplikacje (widoki)

3. Współbieżność — mechanizmy: blokady, MVCC, snapshot isolation

4. Integralność — klucze obce, CHECK, trigger, procedury składowane

5. Optymalizator zapytań — automatyczny wybór planu wykonania

6. Bezpieczeństwo — GRANT/REVOKE, role, szyfrowanie, audyt

7. Skalowalność — replikacja, sharding, klastry

8. Standardowy interfejs — SQL jako uniwersalny język zapytań

Etymologia

ACID — akronim: Reuter & Härder (1983); celowo łatwy do zapamiętania. **ANSI/SPARC** — American National Standards Institute / Standards Planning And Requirements Committee (1975). **SQL** — oryginalnie SEQUEL (Structured English Query Language, Chamberlin & Boyce, IBM 1974); zmieniono na SQL przez konflikt znaku towarowego. **MVCC** — Multi-Version Concurrency Control. **Transakcja** — łac. „transactio” = doprowadzenie do końca.

Jak zapamiętać

- **ACID** — zapamiętaj przelew bankowy: bez A tracisz pieniądze, bez C saldo < 0, bez I widać stan pośredni, bez D znika po crashu
- „**DB = centralne źródło prawdy**” — jedna baza vs. pliki rozproszone po systemach
- Kluczowe słowa: trwałość, współbieżność, integralność, niezależność

PYTANIE 5: Kategorie STL (PROI)

Omówić główne kategorie elementów biblioteki STL.

Cztery filary STL

KONTENERY → ITERATORY → ALGORYTMY → FUNKTORY
(co?) (jak?) (operacje) (parametryzacja)

1. Kontenery

Sekwencyjne: vector (tablica dynamiczna), deque, list, forward_list, array **Asocjacyjne (drzewo R-B):** set, multiset, map, multimap — $O(\log n)$ **Nieuporządkowane (hash):** unordered_set/map — $O(1)$ średnio **Adaptory:** stack (LIFO), queue (FIFO), priority_queue (heap)

2. Iteratory — uogólnione wskaźniki

Hierarchia: Input/Output → Forward → Bidirectional → Random Access → Contiguous - vector: Random Access - list: Bidirectional - forward_list: Forward

3. Algorytmy — operacje na zakresach [begin, end)

sort, find, transform, copy, accumulate, count_if, remove_if... Kluczowa cecha: operują na iteratorach, nie na kontenerach bezpośrednio.

4. Funktory (obiekty funkcyjne) + lambdy

less, greater, plus — parametryzują algorytmy. C++11: lambdy `[] (int a, int b) { return a < b; }`

Wzajemne powiązania — architektura ortogonalna

Algorytmy nie znają kontenerów — komunikują się przez iteratory. M kontenerów × N algorytmów = M+N komponentów (nie M×N).

```
sort(vec.begin(), vec.end()); // vector
sort(deq.begin(), deq.end()); // deque
// Ten sam algorytm, różne kontenery!
```

Etymologia

STL — Standard Template Library; Alexander Stepanov + Meng Lee (HP, 1994); Stepanov od lat 70. marzył o programowaniu generycznym. **Iterator** — łac. „iter” = podróż/ścieżka; ten, kto przemierza kolekcję. **Funktor** — z teorii kategorii (matematyka); obiekt zachowujący się jak funkcja. **Deque** — Double-Ended QUEUE. **Vector** — łac. „vector” = nośnik; tablica dynamiczna. **Lambda** — od greckiej litery λ; Alonzo Church, rachunek lambda (1930s).

Jak zapamiętać

- „**KIAF**” — Kontenery, Iteratory, Algorytmy, Funktory
- **Ortogonalność:** algorytmy + kontenery połączone iteratorami
- **vector** — domyślny wybór; list gdy dużo insert/erase w środku; map/set gdy potrzebne sortowanie i wyszukiwanie

PYTANIE 6: Reużywalność kodu w OOP (PROI)

Omówić metody reużywalności kodu i struktur danych w obiektowych językach programowania.

Główne metody

1. Dziedziczenie (Inheritance) — relacja „jest” (is-a)

- Klasa pochodna przejmuję atrybuty i metody bazowej
- Typy: pojedyncze, wielokrotne, wielopoziomowe
- Problem diamentu → dziedziczenie wirtualne w C++
- Polimorfizm (virtual, override)

2. Kompozycja (Composition) — relacja „zawiera” (has-a)

- **„Favor composition over inheritance”**
- Stack nie JEST wektorem → Stack ZAWIERA wektor
- Silniejsza enkapsulacja, luźne wiązanie
- Typy: kompozycja (owns), agregacja (uses), asocjacja (knows)

3. Programowanie generyczne (Templates/Generics)

- Kod niezależny od typu: `template<typename T> T max(T a, T b)`
- STL jest oparta na templates
- Java/C#: Generics (`List<T>`)

4. Interfejsy i klasy abstrakcyjne

- Kontrakt bez implementacji (pure virtual w C++, interface w Java)
- Umożliwiają multiple inheritance bez diamond problem

5. Wzorce projektowe (Design Patterns)

- Strategy, Observer, Factory, Decorator — reużywalne rozwiązania
- GoF (Gang of Four) — 23 wzorce

6. Biblioteki, frameworki, traity/mixiny

Etymologia

OOP — Alan Kay (Smalltalk, 1970s), sam ukuł termin „object-oriented”. **GoF** — Gang of Four: Gamma, Helm, Johnson, Vlissides (1994). **Polimorfizm** — grec. „poly” (wiele) + „morphē” (forma) = wiele postaci. **Enkapsulacja** — łac. „capsula” = pudełeczko. **Design Pattern** — z architektury: Christopher Alexander „A Pattern Language” (1977); GoF zaadaptowali do IT. **Kompozycja > Dziedziczenie** — zasada z GoF: „favor object composition over class inheritance”.

Jak zapamiętać

- **„Kompozycja > Dziedziczenie”** — najważniejsza zasada
- Dziedziczenie: silne wiązanie, krucha klasa bazowa, diamond problem
- Kompozycja: elastyczna, testowalna, preferowana
- Granica: dziedziczenie dla prawdziwego „is-a” z polimorfizmem; kompozycja dla reszty

PYTANIE 7: DNS i caching (SKM)

Które serwery DNS zyskują najwięcej na cachingu? Jakie znasz rodzaje serwerów DNS?

Rodzaje serwerów DNS

1. **Root Servers** (.) — 13 logicznych (a..m.root-servers.net), setki fizycznych (anycast)
2. **TLD Servers** (.com, .pl, .org) — zarządzane przez rejestry
3. **Authoritative NS** — Primary (master, edytowalny) i Secondary (slave, kopia)
4. **Recursive Resolvers** — wykonują pełne rozwiązywanie (ISP, Google 8.8.8.8, Cloudflare 1.1.1.1)
5. **Stub Resolvers** — prosty klient w OS, wysyła do recursive
6. **Forwarding Servers** — przekazują zapytania dalej

Proces rozwiązywania

Klient → Recursive Resolver → Root → TLD → Authoritative → odpowiedź

ODPOWIEDŹ: ROOT i TLD zyskują NAJWIĘCEJ na cachingu

Dlaczego: - 13 root servers vs miliardy zapytań dziennie - BEZ cache: każde zapytanie o DOWOLNĄ domenę musi przejść przez root i TLD - Z cache: resolver pyta root RAZ o .com, cachuje referral na 48h+ - Root referrals: TTL 48h-7 dni (!); TLD referrals: TTL 24h-48h - Redukcja ruchu do root: z ~100% do ~0.01% zapytań

Etymologia

DNS — Domain Name System; Paul Mockapetris (1983, RFC 882/883). **Cache** — fr. „cacher” = ukrywać; ukryte szybkie przechowywanie. **TTL** — Time To Live. **Anycast** — ten sam IP z wielu lokalizacji; klient dostaje odpowiedź od najbliższego serwera. **Root servers** — 13 logicznych identyfikatorów (a-m); infrastruktura krytyczna internetu. **Recursive resolver** — „rekurencyjny” bo iteracyjnie pyta kolejne poziomy hierarchii aż do odpowiedzi.

Jak zapamiętać

- **„Piramida DNS”** — root (wierzchołek, najmniej serwerów) → TLD → Auth (podstawa, miliony)
- Im mniej serwerów na poziomie, tym większy zysk z cache
- **TTL** = Time To Live — im dłuższy, tym rzadziej odświeżany cache

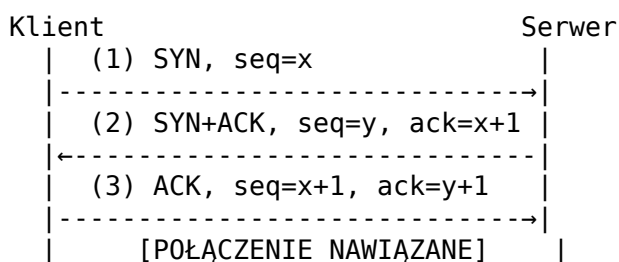
PYTANIE 8: TCP Three-Way Handshake (SKM)

Cel, interpretacja numerów sekwencyjnych, wartość początkowa ISN.

Cel handshake'u

1. Nawiązanie połączenia — obie strony się zgadzają
2. Synchronizacja ISN (Initial Sequence Number)
3. Uzgodnienie parametrów (MSS, Window Scale, SACK)

Przebieg



Numery sekwencyjne (SEQ)

- SEQ = numer pierwszego bajtu danych w segmencie
- Funkcje: kolejność, duplikaty, braki, potwierdzenia

Numery potwierdzenia (ACK)

- ACK = numer **następnego oczekiwanego** bajtu (kumulatywne)
- SACK — opcja potwierdzania niesąsiednich bloków

Wartość początkowa ISN

- **NIE zaczyna od 0** — bezpieczeństwo + unikanie kolizji z poprzednimi połączeniami
- RFC 793: $ISN = \text{timer} \cdot 4\mu s \bmod 2^{32}$
- RFC 6528: $ISN = M + F(\text{adresy}, \text{porty}, \text{secret_key})$ — kryptograficznie

Etymologia

TCP — Transmission Control Protocol; Vint Cerf + Bob Kahn (1974, „A Protocol for Packet Network Intercommunication”). **Handshake** — metafora uścisku dłoni = wzajemna zgoda na komunikację. **SYN** — Synchronize. **ACK** — Acknowledge. **ISN** — Initial Sequence Number. **MSS** — Maximum Segment Size. **SACK** — Selective ACK. **RFC** — Request For Comments; tradycja ARPANET (1969).

Jak zapamiętać

- „**SYN, SYN-ACK, ACK**” — 3 kroki, jak potwierdzenie rozmowy: „Hej!” „Hej, słyszę!” „OK, gadamy!”
- **SEQ = numer bajtu**, ACK = „czekam na bajt numer...”
- **ISN losowy** — bo inaczej atakujący może zgadnąć i przejąć sesję

PYTANIE 9: Procesy i wątki (SOI)

Budowa, szybkość, zastosowanie. Problemy komunikacji i synchronizacji.

Proces — program w trakcie wykonania

Pamięć: TEXT (kod) | DATA | BSS | HEAP | STACK — oddzielna przestrzeń adresowa. PCB: PID, stan, rejestry CPU, tablice stron, otwarte pliki. Stany: NEW → READY ↔ RUNNING → BLOCKED, TERMINATED.

Wątek — lekka jednostka wykonania

Współdzielone z procesem: kod, dane globalne, heap, pliki. **Prywatne:** stos, rejestry CPU, PC, TID.

Porównanie

Cecha	Proces	Wątek
Przestrzeń addr	Własna, izolowana	Współdzielona
Tworzenie	~1-10 ms	~10-100 μs
Przełączanie	Wolne (TLB flush)	Szybkie (rejestry)
Komunikacja	IPC (pipe, shm)	Współdzielona pamięć
Izolacja	Pełna	Brak

Komunikacja międzyprocesowa (IPC)

Pipe, Named Pipe (FIFO), Message Queue, Shared Memory, Sockets, Signals, Memory-mapped files.

Synchronizacja — problemy

- **Wyścig (race condition)** — wynik zależy od kolejności operacji
- **Sekcja krytyczna** — fragment kodu wymagający wyłącznego dostępu
- **Zakleszczenie (deadlock)** — wzajemne oczekiwanie (warunki Coffmana: mutual exclusion, hold & wait, no preemption, circular wait)
- **Zagłodzenie (starvation)** — wątek nigdy nie dostaje zasobu

Mechanizmy synchronizacji

Mutex, Semaphore, Monitor, Condition Variable, Spinlock, Read-Write Lock, Barrier.

Etymologia

Proces — łac. „processus” = posuwanie się naprzód. **Wątek (Thread)** — metafora nitki wykonania (jak nić Ariadny). **Mutex** — portmanteau MUTual EXclusion. **Semafor** — Dijkstra (1965); od semaforów kolejowych; P() = hol. „proberen” (próbować), V() = hol. „verhogen” (podnosić). **Coffman** — Edward Coffman Jr. et al. (1971): 4 warunki konieczne deadlocka. **Deadlock (zakleszczenie)** — jak zablokowane koła zębate. **IPC** — Inter-Process Communication.

Jak zapamiętać

- **„Proces = mieszkanie, Wątek = pokój”** — każde mieszkanie ma adres (przestrzeń), pokoje dzielą kuchnię (heap)
- Wątki szybsze bo nie trzeba zmieniać „mieszkania” (TLB flush)
- **4 warunki Coffmana** zakleszczenia: złam jeden → brak deadlocka

PYTANIE 10: Zarządzanie pamięcią (SOI)

Problemy i mechanizmy. Stronicowanie vs segmentacja.

Problemy

1. **Fragmentacja** — zewnętrzna (wolna pamięć rozproszona) i wewnętrzna (przydzielony blok > potrzebny)
2. **Ochrona** — procesy nie mogą czytać cudzej pamięci
3. **Relokacja** — program musi działać pod różnymi adresami
4. **Współdzielenie** — biblioteki, COW (Copy-on-Write)
5. **Ograniczona pamięć** — więcej procesów niż RAM

Stronicowanie (Paging)

- Pamięć wirtualna → **strony** (pages, np. 4KB), fizyczna → **ramki** (frames)
- **Tablica stron** mapuje strony na ramki
- Translacja: adres = (numer strony | offset) → (numer ramki | offset)
- **Wielopoziomowe tablice** — oszczędność pamięci (32-bit: 2-level, 64-bit: 4-level)
- **TLB** (Translation Lookaside Buffer) — cache translacji
- **Page fault** — strona nie w RAM → ładuj z dysku (swap)
- **Algorytmy wymiany**: FIFO, LRU, Clock (Second Chance), Optimal

Segmentacja (Segmentation)

- Pamięć dzielona na **segmenty logiczne** (kod, dane, stos) o różnych rozmiarach
- Adres = (numer segmentu, offset), tablica segmentów: (baza, limit)
- Ochrona per-segment (R, W, X)

Porównanie

Cecha	Stronicowanie	Segmentacja
Jednostka	Strona (stały rozmiar)	Segment (zmienny)
Fragmentacja	Wewnętrzna	Zewnętrzna
Widok programisty	Niewidoczne	Widoczne (logiczne)
Ochrona	Per-strona	Per-segment
Współdzielenie	Per-strona	Per-segment (naturalne)
Współczesne OS	Dominuje (x86-64)	Rzadko (Intel porzucił)

Etymologia

Stronicowanie (Paging) — pamięć dzielona na „strony” jak w książce. **TLB** — Translation Lookaside Buffer; „lookaside” = sprawdź z boku (cache) zanim sięgniesz do tablicy. **Segmentacja** — łac. „segmentum” = odcięty kawałek. **COW** — Copy-on-Write: kopiuj dopiero przy modyfikacji. **LRU** — Least Recently Used. **FIFO** — First In, First Out. **Page fault** — „fault” to wyjątek sprzętowy, nie błąd programisty.

Jak zapamiętać

- **Stronicowanie = szuflady jednakowej wielkości** (proste, mała fragmentacja wewnętrzna)
- **Segmentacja = pudełka różnej wielkości** (logiczne, ale fragmentacja zewnętrzna)
- Współcześnie: **stronicowanie wygrało** — segmentacja prawie zniknęła (x86-64 = flat segments + paging)

PYTANIE 11: Modelowanie procesów biznesowych (WSYZ)

Scharakteryzować standardy i narzędzia do modelowania procesów biznesowych.

Główne standardy

BPMN 2.0 (Business Process Model and Notation) — OMG

- **Uniwersalny standard** — dla analityków, architektów, programistów
- Elementy: Zdarzenia (○ start, ⊙ pośrednie, ● końcowe), Czynności (prostokąty), Bramki (◇ XOR, ◆ AND, ○◇ OR)
- Łączniki: Sequence Flow (→), Message Flow (- - →), Association (···→)
- Swimlanes: Pool (organizacja) / Lane (dział)
- Format XML → automatyzacja (BPMS)

UML Activity Diagrams

- Część UML 2.x — rozszerzenie flowchartów + sieci Petriego
- Elementy: akcje, decyzje (◇), fork/join (■), pin (object flow)
- Lepsze dla procesów technicznych/software

EPC (Event-driven Process Chain)

- Naprzemienne zdarzenia i funkcje, łączniki AND/OR/XOR
- Popularny w SAP (ARIS framework)

Inne: IDEF0 (modelowanie funkcji), VSM (Value Stream Map, Lean), Flowcharty

BPMN vs UML Activity

Cecha	BPMN	UML Activity
Cel	Procesy biznesowe	Przepływ sterowania
Odbiorcy	Biznes + IT	Głównie IT
Komunikacja	Pools + Message Flow	Partitions
Automatyzacja	Tak (BPMN 2.0 XML)	Ograniczona

Narzędzia

Bizagi Modeler, Camunda, Signavio, Lucidchart, draw.io, Enterprise Architect

Etymologia

BPMN — Business Process Model and Notation; OMG (Object Management Group). **UML** — Unified Modeling Language; „Unified” bo połączył metody Boocha, Rumbaugh i Jacobsona („Three Amigos”, 1990s). **EPC** — Event-driven Process Chain; August-Wilhelm Scheer (Saarland, 1990s; podstawa SAP ARIS). **Swimlane** — metafora torów na basenie: każdy uczestnik na swoim „torze”. **IDEFO** — Integration DEfinition; US Air Force (1970s).

Jak zapamiętać

- **BPMN = „standard nr 1” dla biznesu** — bramki, swimlanes, zdarzenia
- **3 typy bramek: XOR (jeden), AND (wszystkie), OR (jeden lub więcej)**
- UML Activity → programiści; BPMN → wszyscy

PYTANIE 12: Sieciowe modele optymalizacji (WSYZ)

Przedstawić sieciowe modele optymalizacji stosowane w systemach zarządzania.

1. **Najkrótsza ścieżka** — GPS, routing (Dijkstra, Bellman-Ford, A*)
2. **Maksymalny przepływ** — przepustowość linii, dystrybucja (Ford-Fulkerson, Edmonds-Karp)
3. **Minimalny koszt przepływu** — minimalizacja kosztów transportu przy zadanym przepływie
4. **Problem przydziału** — n zadań do n osób, minimalizacja kosztów (algorytm węgierski, $O(n^3)$)
5. **TSP (komiwojażer)** — odwiedź wszystkie miasta raz, minimalizuj trasę (NP-trudny, heurystyki)
6. **CPM/PERT** — harmonogramowanie projektów, ścieżka krytyczna
7. **MST (drzewo rozpinające)** — połącz wszystkie węzły minimalnym kosztem (Kruskal, Prim)

Model	Złożoność	Zastosowanie
Najkrótsza ścieżka	$O(E \log V)$	Logistyka, routing
Max Flow	$O(VE^2)$	Planowanie, dystrybucja
Przydział	$O(n^3)$	HR, grafiki
TSP	NP-trudny	Trasy kurierów
CPM	$O(V+E)$	Zarządzanie projektami
MST	$O(E \log V)$	Sieci infrastrukturalne

Etymologia

Ford-Fulkerson — Lester Ford Jr. + Delbert Fulkerson (1956). **Edmonds-Karp** — Jack Edmonds + Richard Karp (1972); BFS-owa wersja Ford-Fulkerson. **TSP (komiwojażer)** — z XIX-wiecznych niemieckich podręczników dla handlowców. **CPM** — Critical Path Method (DuPont, 1957). **PERT** — Program Evaluation and Review Technique (US Navy, Polaris, 1958). **Kruskal** — Joseph Kruskal (1956). **Prim** — Robert Prim (1957; niezależnie Jarník 1930). **Algorytm węgierski** — Harold Kuhn (1955); od prac Węgrów: Königa i Egerváry'ego.

Jak zapamiętać

- **6 modeli na grafach:** Ścieżka, Przepływ, Min-koszt-przepływ, Przydział, TSP, Harmonogram, MST
- Wszystko sprowadza się do: „węzły + krawędzie + wagi → optymalizuj”

PYTANIE 13/27: Modelowanie architektury systemów informatycznych (AIS)

Cele i metody modelowania architektury.

Cele: komunikacja, dokumentacja, analiza jakości, planowanie, zarządzanie złożonością

Frameworki

TOGAF — metodyka ADM (Architecture Development Method), 4 domeny: Business, Data, Application, Technology.

4+1 View Model (Kruchten): - Logical (funkcjonalność), Process (współbieżność), Development (organizacja kodu), Physical (wdrożenie), + Scenarios (use cases)

Zachman Framework — taksonomia 6×6: What/How/Where/Who/When/Why × poziomy abstrakcji.

C4 Model (Simon Brown): - Level 1: System Context → Level 2: Container → Level 3: Component → Level 4: Code

ArchiMate — 3 warstwy: Business, Application, Technology × 3 aspekty: Active, Behavior, Passive.

Notacje: UML (Component, Deployment, Sequence), ArchiMate, C4, ADR (Architecture Decision Records)

Analiza: ATAM (Architecture Tradeoff Analysis Method), Quality Attributes (ISO 25010: Performance, Security, Scalability, Maintainability, Reliability)

Etymologia

TOGAF — The Open Group Architecture Framework. **Zachman** — John Zachman (IBM, 1987); framework nazwany od twórcy. **C4** — 4 × C: Context, Container, Component, Code (Simon Brown, 2006). **ArchiMate** — „Architecture” + „animate” (The Open Group). **ATAM** — Architecture Tradeoff Analysis Method (SEI, Carnegie Mellon). **Kruchten** — Philippe Kruchten (Rational/IBM, 1995); 4+1 View Model. **ISO 25010** — międzynarodowy standard atrybutów jakości oprogramowania.

Jak zapamiętać

- **TOGAF = JAK** budować architekturę; **Zachman = CO** dokumentować
- **C4 = 4 poziomy zoomu** (Context → Container → Component → Code)
- **4+1 = LDPP+S** (Logical, Development, Process, Physical + Scenarios)

PYTANIE 14/28: Wzorce architektoniczne (AIS)

Czemu służą? Jak powstają? Jak są katalogowane? Przykłady.

Cel: reużywalne rozwiązania typowych problemów, wspólne słownictwo, dokumentacja wiedzy

Powstawanie: Problem powtarzalny → Podobne rozwiązania → Uogólnienie → Dokumentacja → Walidacja → Katalogowanie

Katalogi: POSA (wzorce architektoniczne), GoF (projektowe), EIP (integracja), PoEAA (Fowler), Cloud Patterns

Przykładowe wzorce

Layered (Warstwy): Presentation → Business Logic → Data Access → DB. Separacja odpowiedzialności. Szttywne, boilerplate.

Microservices: Niezależne serwisy, osobne wdrożenia, skalowalność. Złożoność operacyjna.

Event-Driven (EDA): Producer → Event Broker (Kafka) → Consumers. Loose coupling, eventual consistency.

CQRS: Osobne modele Read/Write. Optymalizacja per-strona. Złożoność.

Hexagonal (Ports & Adapters): Core niezależny od frameworków. Testowalność.

Wzorzec	Skalowalność	Złożoność	Use Case
Monolith	Niska	Niska	MVP, małe zespoły
Layered	Średnia	Niska	Enterprise CRUD
Microservices	Wysoka	Wysoka	Duże systemy
Event-Driven	Wysoka	Średnia	Real-time, IoT

Etymologia

POSA — Pattern-Oriented Software Architecture (Buschmann et al., 1996). **GoF** — Gang of Four: Gamma, Helm, Johnson, Vlissides (1994, „Design Patterns”). **EIP** — Enterprise Integration Patterns (Hohpe & Woolf, 2003). **PoEAA** — Patterns of Enterprise Application Architecture (Martin Fowler, 2002). **Hexagonal** — Alistair Cockburn (2005); kształt sześciokąta nie ma specjalnego znaczenia. **CQRS** — Command Query Responsibility Segregation (Greg Young, ~2010); oparty na CQS Bertranda Meyera. **Microservices** — termin spopularyzowany ~2012 (James Lewis, Martin Fowler).

Jak zapamiętać

- „**Monolith first**” — rozdzielaj gdy znasz granice domen
- **Wzorzec = Nazwa + Problem + Rozwiązanie + Konsekwencje**
- Katalogi: POSA = architektura, GoF = klasy/obiekty, EIP = messaging

PYTANIE 15: Agent upostaciowiony w robotyce

Jak wykorzystuje się agenta upostaciowionego do specyfikacji sterowników robotów?

Agent upostaciowiony = ciało fizyczne + sensory + efektory + środowisko

Cykl: **Percepcja → Deliberacja → Akcja** (See-Think-Act)

Architektura sterownika — 3 warstwy (3T Architecture)

1. **PLANNER** (deliberacja) — planowanie symboliczne, sekundy-minuty
2. **SEQUENCER** (wykonawca) — FSM/Behavior Trees, 100ms-sekundy
3. **CONTROLLER** (reaktywny) — PID, unikanie kolizji, milisekundy

Model formalny BDI

- **Beliefs** — mapa, pozycja, stan
- **Desires** — cel nawigacji
- **Intentions** — aktualny plan

Specyfikacja w logice temporalnej (LTL)

- Bezpieczeństwo: $\Box(\text{obstacle} \rightarrow \neg \text{move_forward})$
- Żywotność: $\Diamond(\text{at_goal})$

Behavior Trees — nowoczesna specyfikacja zachowań

- Selector (?): wykonaj pierwszy sukces
- Sequence (→): wykonaj wszystkie po kolei
- Action/Condition jako liście

ROS (Robot Operating System) — middleware pub/sub dla robotów

Etymologia

Agent upostaciowiony (Embodied) — łac. „corpus” = ciało; agent posiadający ciało fizyczne w środowisku. **BDI** — Beliefs-Desires-Intentions; Michael Bratman (filozof, 1987); Rao & Georgeff (1991) przenieśli do AI. **LTL** — Linear Temporal Logic; Amir Pnueli (1977, Turing Award 1996). **PID** — Proportional-Integral-Derivative; Nicolas Minorsky (1922, sterowanie okrętami). **ROS** — Robot Operating System (Willow Garage, 2007). **Behavior Tree** — z game AI (Halo 2, ~2004); zaadaptowane w robotyce.

Jak zapamiętać

- **„See-Think-Act”** = Percepcja → Deliberacja → Akcja
- **3T = Plan-Sequence-Control** (od abstrakcji do sprzętu)
- BDI = Beliefs, Desires, Intentions

PYTANIE 16: Języki programowania robotów

Omówić specjalizowane języki. Uwypuklić klasyfikację.

Klasyfikacja wg poziomu abstrakcji: T-R-M-S

1. **Task-level** — „Podnieś A, połóż na B” (PDDL, Behavior Trees)
2. **Robot-level** — `move_to()`, `grasp()` (RAPID, KRL, Karel, ROS)
3. **Motion-level** — trajektorie, kinematyka odwrotna (MoveIt, OMPL)
4. **Servo-level** — PID, sterowanie silnikami (C/C++, FPGA)

Klasyfikacja wg metody: Online (teach-in, pendant) vs Offline (symulacja, CAD)

Języki producentów

Producent	Język	Ruchy
ABB	RAPID	MoveJ, MoveL, MoveC
KUKA	KRL	PTP, LIN, CIRC
FANUC	Karel	MOVE TO
Comau	PDL2	MOVE LINEAR TO

Uniwersalne: ROS + Python/C++, MoveIt (planowanie manipulatora), Orocos (real-time)

Graficzne: RobotStudio (ABB), ROBOGUIDE (FANUC), Blockly (edukacja)

Etymologia

RAPID — Robotics Application Programming Interactive Dialogue (ABB). **KRL** — KUKA Robot Language. **Karel** — od Karla Čapka, czeskiego pisarza, który ukuł słowo „robot” (cz. „robota” = ciężka/przymusowa praca) w sztuce R.U.R. (1920). **PDDL** — Planning Domain Definition Language. **MoveIt** — open source do planowania ruchu manipulatora (Willow Garage/PickNik). **Robot** — cz. „robota” = pańszczyzna; Karel Čapek, R.U.R. (1920).

Jak zapamiętać

- „Od zadania do serwa: T-R-M-S”
- Każdy producent ma WŁASNY język (vendor lock-in)
- ROS próbuje ujednolicić, ale nie dla hard real-time

PYTANIE 17: Szeregowanie zadań

Cechy klasyfikacji. Przykładowa metoda.

Notacja Grahama: $\alpha | \beta | \gamma$

- α — środowisko maszynowe: 1 (jedna), Pm (m równoległych), F (flow shop), J (job shop)
- β — charakterystyki zadań: r_j (release dates), d_j (due dates), pmtn (preemption), prec (precedencje)
- γ — kryterium: C_{\max} (makespan), ΣC_j , L_{\max} , ΣT_j , ΣU_j

Klasyczne reguły optymalne

Reguła	Problem	Opis
SPT	1 ΣC_j	Najkrótsze najpierw
EDD	1 L_{\max}	Najwcześniejszy termin
Johnson	F2 C_{\max}	Optymalny dla 2-maszynowego flow shop

Przykład: 1 || ΣC_j z regułą SPT

Zadania: J1(5), J2(3), J3(8), J4(2), J5(6) Posortowane SPT: J4(2), J2(3), J1(5), J5(6), J3(8) Czasy zakończenia: 2, 5, 10, 16, 24 $\Sigma C_j = 57$ (optymalne!)

Dowód: zamiana sąsiednich i, j gdzie $p_i > p_j$ zawsze zwiększa ΣC .

Złożoność: Większość problemów NP-trudna (Job shop, Pm|| C_{\max} dla $m \geq 2$)

Etymologia

Notacja Grahama — Ronald Graham (Bell Labs, 1966–1979); znany też z liczby Grahama. **SPT** — Shortest Processing Time. **EDD** — Earliest Due Date. **Johnson** — Selmer Johnson (RAND, 1954). **Makespan** — „make” (ukończyć) + „span” (rozpiętość); czas od startu do końca wszystkich zadań. **Flow shop** — zadania „płyną” przez maszyny w tej samej kolejności (jak taśma). **Job shop** — każde zadanie ma indywidualną trasę. **NP-trudny** — Non-deterministic Polynomial-time hard.

Jak zapamiętać

- $\alpha|\beta|\gamma$ = Maszyny|Zadania|Cel
- SPT = „Short first” — krótsze pierwsze dla sumy C_j
- EDD = „Early Due Date” — najwcześniejszy termin dla L_{\max}

PYTANIE 18: Zarządzanie zapasami w łańcuchu dostaw

Problemy i przykładowy model.

Problemy

- **Bullwhip Effect** — amplifikacja wahań popytu w górę łańcucha (detaliści → dystrybutorzy → producenci → dostawcy). Przyczyny: forecasting, batching, promocje.
- Stockouts vs Overstock, obsolescence, lead time variability, demand uncertainty

Koszty: Utrzymania (h) + Zamawiania (K) + Braku (p)

Model EOQ (Economic Order Quantity) — Harris-Wilson

Założenia: popyt stały D, lead time = 0, koszt zamówienia K, koszt utrzymania h.

$$TC(Q) = K \cdot D/Q + h \cdot Q/2$$

Optymalna wielkość: $Q^* = \sqrt{(2KD/h)}$

Opt. koszt: $TC^* = \sqrt{(2KDh)}$

Przykład: D=10000, K=100, h=2 → Q=1000, TC=2000 PLN/rok

Punkt zamawiania (ROP)

$ROP = d \times L + SS$ (popyt dzienny × lead time + safety stock)

$SS = z \times \sigma_L$ (z z tablic normalnych, np. 1.65 dla 95%)

Modele zaawansowane: (s,Q), (s,S), (R,S), VMI (Vendor Managed Inventory)

Etymologia

EOQ — Economic Order Quantity; Ford W. Harris (1913, „How Many Parts To Make At Once”) — jedno z najstarszych zastosowań badań operacyjnych. **Bullwhip Effect** — Procter & Gamble (1990s) nadali nazwę; Jay Forrester (MIT, 1961) pierwszy opisał jako „demand amplification”; „bullwhip” = bicz pasterski. **ROP** — Reorder Point. **VMI** — Vendor Managed Inventory (Walmart + P&G, lata 80.). **Safety stock** — zapas bezpieczeństwa na wypadek wahań popytu/dostaw.

Jak zapamiętać

- **EOQ** = $\sqrt{(2KD/h)}$ — zapamiętaj formułę!
- **ROP** = $d \times L + SS$ — kiedy zamawiać
- **Bullwhip** = **bicz** — małe wahania na końcu → duże na początku łańcucha

PYTANIE 19/29: Model Publish-Subscribe

Scharakteryzować model i przykładowe rozwiązania techniczne.

Model Pub/Sub

Publishers → **Broker** (router/message bus) → Subscribers - Luźne powiązanie (publisher nie zna subscriberów) - Asynchroniczne, skalowalne (1:N, N:M)

Typy subskrypcji: topic-based, content-based, type-based, hierarchical (wildcards)

Gwarancje dostarczenia (QoS): At-most-once, At-least-once, Exactly-once

Rozwiązania techniczne

Technologia	Model	Persistence	Throughput	Use Case
Kafka	Pull (log)	Tak	Bardzo wysoki	Event streaming
RabbitMQ	Push (queue)	Opcjonalne	Wysoki	Task queues
MQTT	Push	Retained	Niski-średni	IoT
Redis	Push	Nie	Wysoki	Real-time

Kafka: Partycje + Consumer Groups, distributed log, exactly-once z transakcjami. **RabbitMQ:** AMQP, Exchange types (Direct, Topic, Fanout, Headers), flexible routing. **MQTT:** Lekki (2-byte header), idealny dla IoT/constrained devices, brokers: Mosquitto, HiveMQ.

Zalety / Wady

Zalety: decoupling, skalowalność, asynchroniczność, broadcast. Wady: debugging trudniejszy, ordering challenges, broker = SPOF.

Etymologia

Kafka — od Franza Kafki (pisarz); Jay Kreps (LinkedIn, 2011): „system zoptymalizowany do pisania — a Kafka był pisarzem”. **RabbitMQ** — „rabbit” = szybkość; AMQP = Advanced Message Queuing Protocol. **MQTT** — Message Queuing Telemetry Transport; Andy Stanford-Clark (IBM) + Arlen Nipper (1999); do monitoringu rurociągów naftowych przez satelitę. **Redis** — REmote DIctionary Server (Salvatore Sanfilippo, 2009). **Pub/Sub** — publish-subscribe; wzorzec z systemów event-driven (lata 80/90). **QoS** — Quality of Service.

Jak zapamiętać

- „**Pub/Sub** = **Radio**” — nadawca nadaje, kto chce słucha
- **Kafka** = **Log** (przechowuje historię), **RabbitMQ** = **Queue** (konsumuje i kasuje)
- **MQTT** = **IoT** — lekki, mały overhead

PYTANIE 20/30: Analityka danych strumieniowych

Rozwiązania analityczne na danych strumieniowych.

Charakterystyka strumieni

- Nieograniczone (unbounded), ciągłe, niska latencja wymagana
- Event Time vs Processing Time — mogą się różnić, out-of-order

Okna czasowe (Windowing)

1. **Tumbling** — rozłączne, stały rozmiar
2. **Sliding** — nakładające się (size + slide)
3. **Session** — oparte na aktywności (gap between events)
4. **Global** — jedno okno, trigger decyduje emisję

Platformy

Cecha	Kafka Streams	Flink	Spark Streaming
Model	True streaming	True streaming	Micro-batch
Deployment	Library	Cluster	Cluster
Latency	Niska	Bardzo niska	Średnia (~100ms)
Exactly-once	Tak	Tak	Tak

Algorytmy strumieniowe

- **HyperLogLog** — zliczanie unikalnych, $O(1)$ space, ~2% error
- **Count-Min Sketch** — estymacja częstości, overestimates
- **Reservoir Sampling** — równomierne próbkowanie k z n nieznanego

Obsługa opóźnień: Watermarks + late data strategies (drop, recompute, side output, allowed lateness)

Etymologia

Flink — niem. „flink” = zwinny/szybki (TU Berlin, 2014). **Spark** — „iskra”; Matei Zaharia (UC Berkeley, 2012). **HyperLogLog** — Philippe Flajolet et al. (2007); „Hyper” = ulepszenie LogLog; „LogLog” = zużywa $\log(\log(n))$ pamięci. **Count-Min Sketch** — Cormode & Muthukrishnan (2005); „sketch” = probabilistyczny skrót danych. **Reservoir Sampling** — Jeffrey Vitter (1985); „reservoir” = stały zbiornik prób. **Watermark** — znacznik postępu czasu zdarzeń w strumieniu.

Jak zapamiętać

- **4 okna: „TSSG”** — Tumbling, Sliding, Session, Global
- **Flink = szybki (true streaming)**, Spark = safe (micro-batch)
- **HyperLogLog = „ile unikalnych?” z kilobajtem pamięci**

PYTANIE 21: Zegary logiczne i wektory stempli czasowych

Koncepcja i przeznaczenie.

Problem: brak globalnego zegara w systemach rozproszonych (drift, opóźnienia)

Relacja happened-before (\rightarrow) — Lamport 1978

- a, b w jednym procesie, a przed b $\rightarrow a \rightarrow b$
- a = wysłanie msg, b = odbiór $\rightarrow a \rightarrow b$
- Przechodniość
- a || b (współbieżne) gdy $\neg(a \rightarrow b) \wedge \neg(b \rightarrow a)$

Zegar Lamporta (skalaryny)

Algorytm: przed zdarzeniem C_i++ ; wysyłając dołącz C_i ; odbierając $C_i = \max(C_i, t) + 1$.

Właściwość	Lamport
$a \rightarrow b \implies C(a) < C(b)$	TAK
$C(a) < C(b) \implies a \rightarrow b$	NIE

Zegary wektorowe

Każdy z N procesów ma wektor $V[1..N]$. Przed zdarzeniem: $V[i]++$; wysyłając dołącz V; odbierając: $V[j] = \max(V[j], T[j]) \forall j$, potem $V[i]++$.

Właściwość	Vector Clock
$a \rightarrow b \iff V(a) < V(b)$	TAK
Wykrycie współbieżności	TAK

Porównanie: $V \leq W \iff \forall i: V[i] \leq W[i]$; $V || W$ gdy $\neg(V \leq W) \wedge \neg(W \leq V)$

Lamport: O(1) rozmiar, ale nie wykrywa współbieżności.

Vector: O(N) rozmiar, ale pełna charakteryzacja happened-before.

Zastosowania: replikacja (Dynamo — version vectors), causal broadcast, distributed debugging

Etymologia

Lamport — Leslie Lamport (1978, „Time, Clocks, and the Ordering of Events...“); Turing Award 2013; twórca LaTeX-a! **Vector clocks** — Friedemann Mattern + Colin Fidge (niezależnie, 1988). **Happened-before** — Lamportowski termin; relacja częściowego porządku. **Dynamo** — Amazon (2007); wektory wersji do wykrywania konfliktów. **Causal broadcast** — „causal” od łac. „causa” = przyczyna; wiadomości dostarczane w porządku przyczynowym.

Jak zapamiętać

- **Lamport = 1 liczba** — „wie że było wcześniej, ale nie wie czy współbieżnie”
- **Vector = wektor N liczb** — „każdy wie o każdym” \rightarrow pełna informacja
- **$V(a) < V(b) \iff a \rightarrow b$** — kluczowa równoważność vector clocks

PYTANIE 22: Modele spójności danych w systemach rozproszonych

Silne i słabe modele spójności.

Spektrum (od silnego do słabego)

Linearizability → Sequential → Causal → Session → Eventual

Silne modele

Linearizability: Każda operacja wygląda atomowo w momencie między wywołaniem a odpowiedzią. Najsilniejsza. „Globalny czas rzeczywisty.” Kosztowna (consensus). Przykład: Spanner.

Sequential Consistency: Globalny porządek operacji zgodny z porządkiem programu każdego procesu, ale NIE z czasem rzeczywistym. Przykład: Zookeeper.

Słabe modele

Causal Consistency: Operacje przyczynowo zależne widziane w tej samej kolejności. Niezależne mogą być w różnej. Wymaga vector clocks. Przykład: MongoDB.

Session Guarantees: Read Your Writes, Monotonic Reads, Monotonic Writes, Writes Follow Reads.

Eventual Consistency: Jeśli brak nowych zapisów, ostatecznie wszystkie odczyty zwrócą tę samą wartość. Najłabszy, najszybszy. Przykład: DNS, Cassandra.

CAP Theorem: Consistency + Availability + Partition tolerance — wybierz 2 (w obecności partycji: C lub A).

Rozwiązywanie konfliktów: LWW (Last-Writer-Wins), Siblings (multi-value), CRDTs (automatycznie zbieżne struktury).

Etymologia

Linearizability — Maurice Herlihy + Jeannette Wing (1990); łac. „linearis” = w linii. **Sequential consistency** — Leslie Lamport (1979). **Causal consistency** — Ahamad et al. (1995); łac. „causa” = przyczyna. **CAP** — Eric Brewer (UC Berkeley, 2000, „Brewer’s conjecture”); udowniony przez Gilbert & Lynch (2002). **CRDTs** — Conflict-free Replicated Data Types (Marc Shapiro et al., 2011). **Quorum** — łac. „of whom” = minimalna liczba głosów (z prawa rzymskiego).

Jak zapamiętać

- **Linearizable** = „natychmiast, atomowo, jak 1 kopia”
- **Eventual** = „kiedyś się zsynchronizuje” (ale kiedy?)
- **CAP** = „Partition → wybierz C albo A”
- **Quorum: $W+R > N$** gwarantuje odczyt najnowszej wartości

PYTANIE 23: Segmentacja obrazu

Problem, strategie klasyczne i sieci neuronowe.

Definicja: Przypisać każdemu pikselowi etykietę klasy/regionu.

Typy: **Semantic** (klasa per piksel), **Instance** (rozróżnia instancje), **Panoptic** (unified)

Metody klasyczne

Metoda	Idea	Wada
Thresholding	Próg intensywności (Otsu = automat.)	Tylko 2 klasy
Region Growing	Rozszerzaj od seeda wg podobieństwa	Over-segmentation
Watershed	„Zalewanie” topografii obrazu	Over-segmentation
Mean Shift	Przesuń do max gęstości, grupuj zbieżne	Wolny
Normalized Cuts	Graf: min-cut z normalizacją	$O(n^3)$

Metody deep learning

FCN (2015): Pierwsza sieć fully-convolutional; encoder → upsampling; skip connections. **U-Net (2015):** Encoder-decoder w kształcie U; skip connections (concat); popularne w medycynie. **DeepLab v3+ :** Atrous/dilated convolutions (większe receptive field bez parametrów); ASPP (multi-scale). **SegFormer, Mask2Former:** Transformer-based; SOTA.

Metryki: mIoU (mean Intersection over Union) — standard; Dice, Pixel Accuracy.

Loss: Cross-Entropy, Dice Loss, Focal Loss (class imbalance).

Etymologia

Segmentacja — łac. „segmentum” = odcięty kawałek; podział obrazu na regiony. **Otsu** — Nobuyuki Otsu (1979); automatyczny dobór progu. **Watershed** — metafora: woda spływająca z grani do dolin (z geografii). **U-Net** — Ronneberger et al. (Freiburg, 2015); „U” od kształtu architektury. **FCN** — Fully Convolutional Network (Long, Shelhamer, Darrell, 2015). **DeepLab** — Google (2015–2018); „Atrous” z fr. „à trous” = „z dziurami” (dilated convolutions). **mIoU** — mean Intersection over Union.

Jak zapamiętać

- **U-Net** = „U-shape + skip connections” — encoder-decoder
- **DeepLab** = „Atrous (dilated) convolutions + ASPP”
- **mIoU** = Intersection / Union, uśrednione per klasa

PYTANIE 24: Detekcja obiektów

Problem, metody klasyczne, deep learning. Jak zbudować detektor z klasyfikatora?

Definicja: Lokalizacja (bounding box) + klasyfikacja obiektów. **Wynik:** (class, bbox, confidence).

Metody klasyczne

Sliding Window + HOG/SVM: Przesuwaj okno, wyekstrahuj cechy HOG, klasyfikuj SVM. Wolne!
Viola-Jones (2001): Haar features + Integral Image + AdaBoost cascade. Face detection real-time.

Deep Learning

Two-stage (R-CNN family): - R-CNN: Selective Search → 2000 regionów → CNN per region → SVM. 50 sec/image. - Fast R-CNN: CNN raz, ROI Pooling. ~2 sec. - **Faster R-CNN:** RPN (Region Proposal Network) zamiast Selective Search. ~5 fps.

One-stage: - **YOLO:** Grid S×S, każda cell predykuje B bbox + C klas. 45-155 fps! Gorszy dla małych obiektów. - **SSD:** Multi-scale feature maps + anchors. Łączy szybkość YOLO z multi-scale.

Nowoczesne: YOLOv8 (anchor-free), DETR (transformer, no NMS), RT-DETR.

Jak zbudować detektor z klasyfikatora?

1. **Sliding Window:** crop → classify → NMS. Bardzo wolne.
2. **Region Proposals + Classifier:** Selective Search → crop → classify → NMS. Szybsze.
3. **Fine-tune na detekcję:** Pretrained classifier jako backbone + detection head (bbox regression + cls). **Najlepsza jakość!**

NMS (Non-Maximum Suppression): Sortuj po confidence; weź najlepszą; usuń overlapping (IoU > threshold); powtórz.

Etymologia

YOLO — You Only Look Once (Joseph Redmon et al., 2016). **R-CNN** — Region-based CNN (Ross Girshick, 2014). **HOG** — Histogram of Oriented Gradients (Dalal & Triggs, 2005). **SVM** — Support Vector Machine (Vapnik, 1995). **Viola-Jones** — Paul Viola + Michael Jones (2001). **DETR** — Detection TRansformer (Facebook AI, 2020). **SSD** — Single Shot MultiBox Detector (Liu et al., 2016). **NMS** — Non-Maximum Suppression; tłumienie nie-maksymalnych detekcji.

Jak zapamiętać

- **YOLO = „You Only Look Once”** — jednoetapowy, szybki
- **Faster R-CNN = CNN + RPN + ROI Pool** — dwuetapowy, dokładny
- **Detektor z klasyfikatora:** sliding window (wolno) → proposals (lepiej) → fine-tune backbone (najlepiej)

PYTANIE 25: Prawo Amdahla — przyspieszenie równoległe

Oszacować przyspieszenie. Co osłabia ograniczenie?

Prawo Amdahla

$$S(n) = 1 / ((1-p) + p/n)$$

- p = część równoległa, n = procesory
- Maks. przyspieszenie ($n \rightarrow \infty$): **$S_{\max} = 1/(1-p)$**
- 10% sekwencyjnego kodu \rightarrow max **10x** nawet z ∞ procesorami!

Tabela przykładów

p	n=4	n=16	n= ∞
90%	3.08	5.93	10
95%	3.48	9.52	20
99%	3.88	13.91	100

Co osłabia ograniczenie?

Prawo Gustafsona: $S = 1 - p + p \cdot n$. Skaluj problem (więcej danych), nie procesory. Dla $p=90\%$, $n=100 \rightarrow S=90.1x$ (vs Amdahl: $\sim 10x$).

Techniki: algorytmy równoległe, lock-free structures, pipelining, speculative execution, ukrywanie latencji (async I/O, prefetching).

Czynniki zmniejszające RZECZYWISTE przyspieszenie

- Overhead synchronizacji (mutex contention)
- Komunikacja (latencja, bandwidth)
- Load imbalance
- Cache effects (false sharing, NUMA)
- Thread management

Efektywność: $E(n) = S(n)/n$ — spada z n

Etymologia

Gene Amdahl (IBM, 1967, „Validity of the single processor approach...“); współtwórca IBM System/360. **John Gustafson** (Sandia Labs, 1988, „Reevaluating Amdahl’s Law”); weak scaling vs strong scaling. **Speedup (przyspieszenie)** — stosunek czasu sekwencyjnego do równoległego. **Efektywność** — ile z dodanych procesorów jest naprawdę wykorzystane. **Lock-free** — struktury danych bez blokad (CAS — Compare-And-Swap).

Jak zapamiętać

- **$S = 1/((1-p) + p/n)$** — zapamiętaj wzór!
- **„10% seq = max 10x”** — sekwencyjna część limituje WSZYSTKO
- **Gustafson = „zwiększ problem, nie procesory”** — weak scaling
- **„FLOP” = False sharing, Load imbalance, Overhead, Poor locality**

PYTANIE 26: Komunikacja synchroniczna/asynchroniczna, blokująca/nieblokująca

Definicje. Jak uniknąć zakleszczenia w symetrycznych procesach (Jacobi)?

Definicje

Synchroniczna: Nadawca czeka aż odbiorca odbierze (oba zsynchronizowane). **Asynchroniczna:** Nadawca wysyła do bufora i kontynuuje (nie czeka na odbiorcę).

Blokująca: Funkcja nie wraca dopóki operacja nie skończona. **Nieblokująca:** Funkcja wraca natychmiast; operacja w tle; sprawdzaj wait()/test()).

MPI

Funkcja	Blok?	Sync?
MPI_Send	Blok	Zależy
MPI_Ssend	Blok	Sync
MPI_Bsend	Blok	Async
MPI_Isend	Nie	Async
MPI_Recv	Blok	-
MPI_Irecv	Nie	-

Problem: Zakleszczenie w symetrycznym kodzie

```
// DEADLOCK!  
Proc 0: Send(to=1); Recv(from=1);  
Proc 1: Send(to=0); Recv(from=0);  
// Oba czekają aż partner odbierze, nikt nie robi Recv!
```

Rozwiązania

1. **Zmiana kolejności:** Proc 0: Send→Recv; Proc 1: Recv→Send (asymetria)
2. **Nieblokujące:** Irecv + Isend + Wait (oba procesy symetrycznie)
3. **MPI_Sendrecv** — jedna funkcja, automatycznie bezpieczna
4. **Bsend** — buforowane wysyłanie (kopiuje do bufora i wraca)

Etymologia

MPI — Message Passing Interface (MPI Forum, 1994); standard komunikacji w obliczeniach równoległych. **Jacobi** — Carl Gustav Jacob Jacobi (1804–1851, mat. niemiecki); metoda iteracyjna rozwiązywania układów równań. **Synchroniczna** — grec. „syn” (razem) + „chronos” (czas) = w tym samym czasie. **Asynchroniczna** — grec. „a-” (nie) + synchronous = nie w tym samym czasie. **Blokująca** — funkcja „blokuje” wątek aż operacja się skończy.

Jak zapamiętać

- **Deadlock = Send-Send** — oba czekają, nikt nie odbiera
- **Sendrecv = „safe exchange”** — jedna funkcja, zero deadlocków
- **I = Immediate = Non-blocking** (MPI_Isend, MPI_Irecv)
- **S = Synchronous** (MPI_Ssend — czeka na recv)

PYTANIE 31: Interaktywne wspomaganie decyzji w warunkach ryzyka

Przedstawić metody interaktywne.

Warunki: pewność (determinizm) → ryzyko (znane prawdopodobieństwa) → niepewność (brak prawdopodobieństw)

Interaktywność = dialog z decydentem → odkrycie preferencji (funkcji użyteczności)

Metody

1. Metoda loterii: Ustal $U(\text{worst})=0$, $U(\text{best})=1$. Pytaj: „Wolisz x_{mid} na pewno, czy loterię (p : best, $1-p$: worst)?” Punkt obojętności $p^* = U(x_{\text{mid}})$.

2. Certainty Equivalent (CE): $CE(L)$ = pewna kwota równoważna loterii L . - $CE < E[X]$ → risk averse (wklęsła U) - $CE = E[X]$ → risk neutral - $CE > E[X]$ → risk seeking - Risk Premium = $E[X] - CE$

3. AHP (Analytic Hierarchy Process): Hierarchia: Cel → Kryteria → Alternatywy. Porównania parami (skala 1-9) → eigenvalue → wagi. Consistency Ratio $CR < 0.1$.

4. PROMETHEE: Funkcje preferencji per kryterium; agregacja; przepływy Φ^+ , Φ^- , Φ (net); ranking.

5. ELECTRE: Concordance (zgoda) + Discordance (sprzeciw) → outranking aSb.

Etymologia

AHP — Thomas Saaty (U. of Pittsburgh, 1970s); Analytic Hierarchy Process. **PROMETHEE** — Preference Ranking Organization METHod for Enrichment Evaluations (Jean-Pierre Brans, 1982). **ELECTRE** — ÉLimination Et Choix Traduisant la REalité (Bernard Roy, 1965) = „Eliminacja i Wybór Odzwierciedlający Rzeczywistość”. **Certainty Equivalent** — z teorii użyteczności von Neumanna-Morgensterna (1944). **Funkcja użyteczności** — Daniel Bernoulli (1738) wprowadził koncepcję; vN-M sformalizowali aksjomatycznie.

Jak zapamiętać

- **CE** = „ile dałbyś za pewniaka zamiast loterii?” → miara awersji do ryzyka
- **AHP** = „porównaj parami, policz wagi” (macierz → eigenvalue)
- **PROMETHEE** = „przepływy” (Φ^+ outgoing, Φ^- incoming)

PYTANIE 32: Dominacja stochastyczna

FSD i SSD. Jak mogą być użyte w modelach wyboru?

Idea: Porównaj rozkłady BEZ znajomości dokładnej U. Jeśli A dominuje B → KAŻDY (z danej klasy) wybierze A.

FSD (First-order Stochastic Dominance)

$$A \geq_{\text{FSD}} B \iff F_A(x) \leq F_B(x) \quad \forall x$$

- Warunek na U: $U'(x) \geq 0$ (monotoniczność — „więcej = lepiej”)
- Klasa: WSZYSCY racjonalni (nienasyćeni)
- Interpretacja: A ma zawsze \geq prawdopodobieństwo przekroczenia dowolnego progu
- **Rzadka** w praktyce

SSD (Second-order Stochastic Dominance)

$$A \geq_{\text{SSD}} B \iff \int_{-\infty}^x F_A(t) dt \leq \int_{-\infty}^x F_B(t) dt \quad \forall x$$

- Warunek na U: $U' \geq 0$ i $U'' \leq 0$ (monotoniczne + wklęsłe)
- Klasa: **Risk-averse** (awersja do ryzyka)
- Dystrybuanty mogą się przecinać, ale skumulowane pole nie
- **Częstsza** niż FSD
- Mean-Preserving Spread: $B = A + \varepsilon$ ($E[\varepsilon|A]=0$) → A SSD B

Relacja: FSD \implies SSD \implies TSD... (ale nie odwrotnie)

Zastosowania

- **Portfolio selection:** eliminuj zdominowane portfele bez znajomości U
- **Ubezpieczenia:** fair ubezpieczenie SSD-dominuje brak ubezpieczenia (dla risk-averse)
- **Ocena inwestycji:** A: $N(10\%, 15\%)$, B: $N(8\%, 20\%)$ → $E[A] > E[B]$, $\sigma[A] < \sigma[B]$ → A SSD B

Cecha	FSD	SSD
Warunek Na U	$F_A(x) \leq F_B(x) \quad \forall x$ $U' \geq 0$	$\int F_A \leq \int F_B \quad \forall x$ $U' \geq 0, U'' \leq 0$
Decydenci	Wszyscy racjonalni	Risk-averse
Częstość	Rzadka	Częstsza

Etymologia

Stochastyczna — grec. „stochastos” = zdolny do celowania, od „stochazein” = mierzyć; w probabilistyce: losowy. **FSD/SSD** — Hadar & Russell (1969); Rothschild & Stiglitz (1970) niezależnie. **Mean-Preserving Spread** — Rothschild & Stiglitz: ten sam średni wynik, ale większy rozrzut = gorsze dla risk-averse. **Dominacja** — łac. „dominari” = panować; A dominuje B gdy jest zawsze co najmniej tak dobre.

Jak zapamiętać

- **FSD = „F always below”** — dystrybuanta A zawsze \leq B
- **SSD = „Second = Sum (integral)”** — całka z $F_A \leq$ całka z F_B
- **FSD → wszyscy; SSD → risk-averse**
- FSD implikuje SSD, ale nie odwrotnie