

Integrarea sistemelor informatice



Suport curs nr. 6

Programator >> Arhitect

Probleme legate de integrare

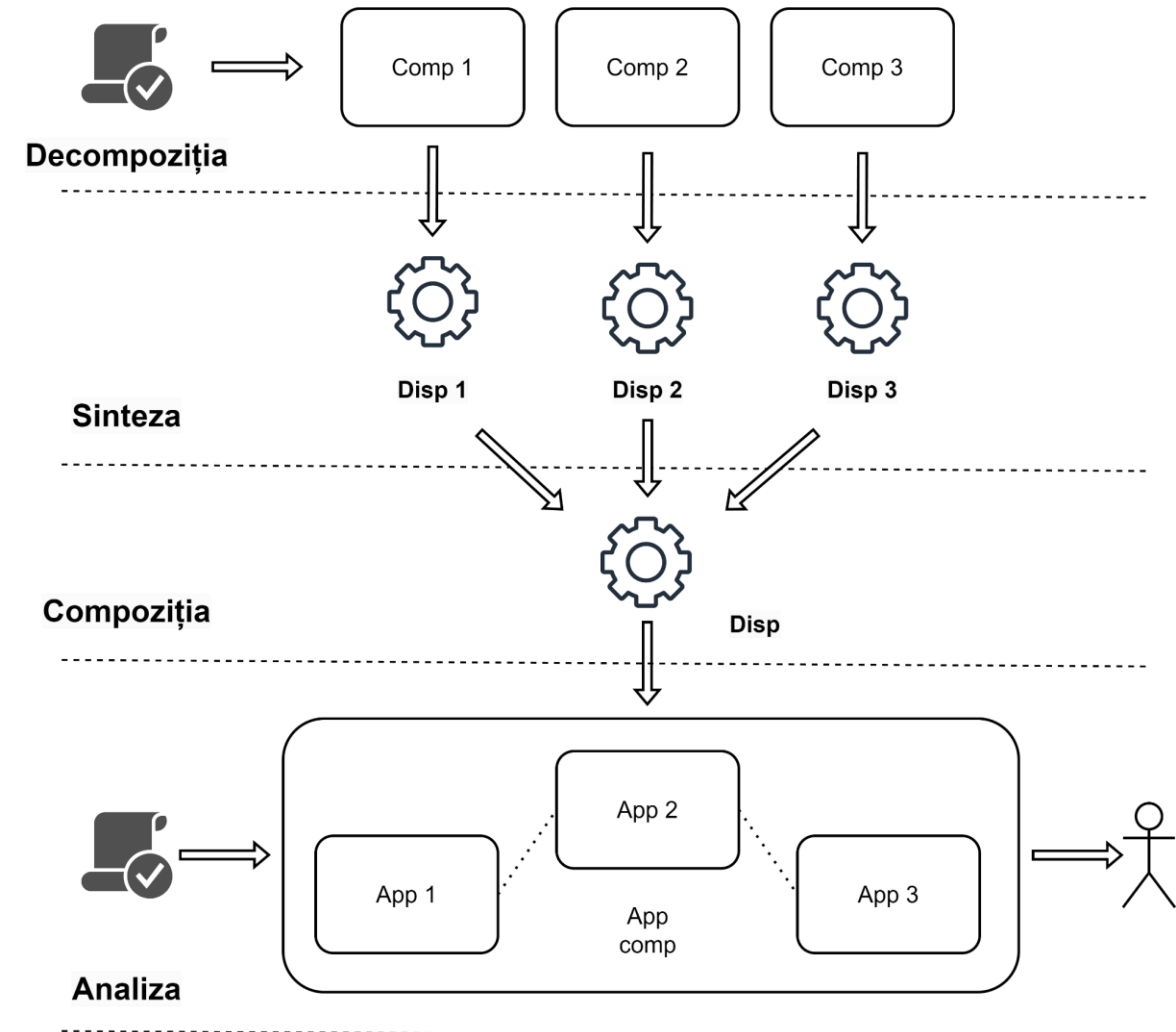
2023-2024

Obiective

- Înțelegerea tipurilor de probleme în integrarea sistemelor
- Identificarea conflictelor de integrare
- Identificarea metodelor de soluționare

Recapitulare – aplicații compozite

- **Ingineria proiectării** aplicațiilor compozite implică următoarele activități de bază
 - **Decompoziția** - descompunerea funcționalităților în sub-comportamente
 - **Sinteza** - identificarea resurselor și adaptarea acestora la roluri
 - **Compoziția** - proiectarea unui (singur) sistem care să înglobeze mai multe funcțiuni /comportamente ale unor subsisteme
 - **Analiza** - identificarea constrângerilor legate de interacțiune, conflicte și compromisuri



Recapitulare – aplicații compozite

Tipuri de arhitecturi

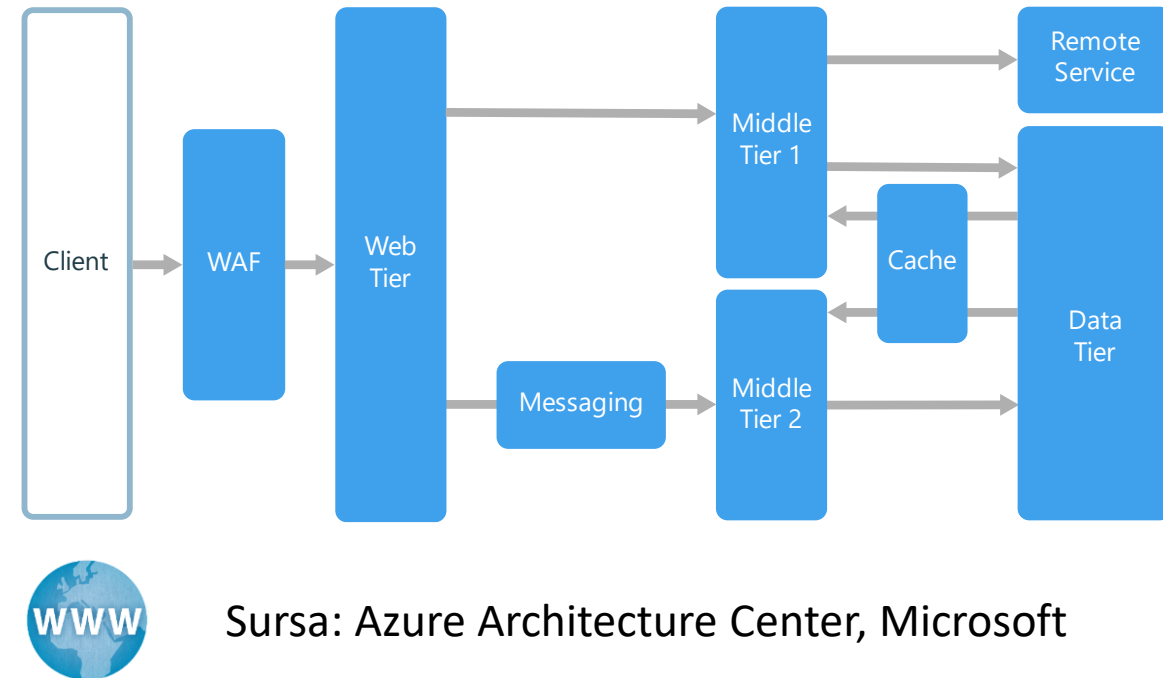
- Arhitectura centralizată
- Arhitectura “two-tier”
- Arhitectura “three-tier”
- SOA (Service-Oriented Architecture)
- Arhitectura bazată pe microservicii
- Arhitecturi serverless



Image by Freepik

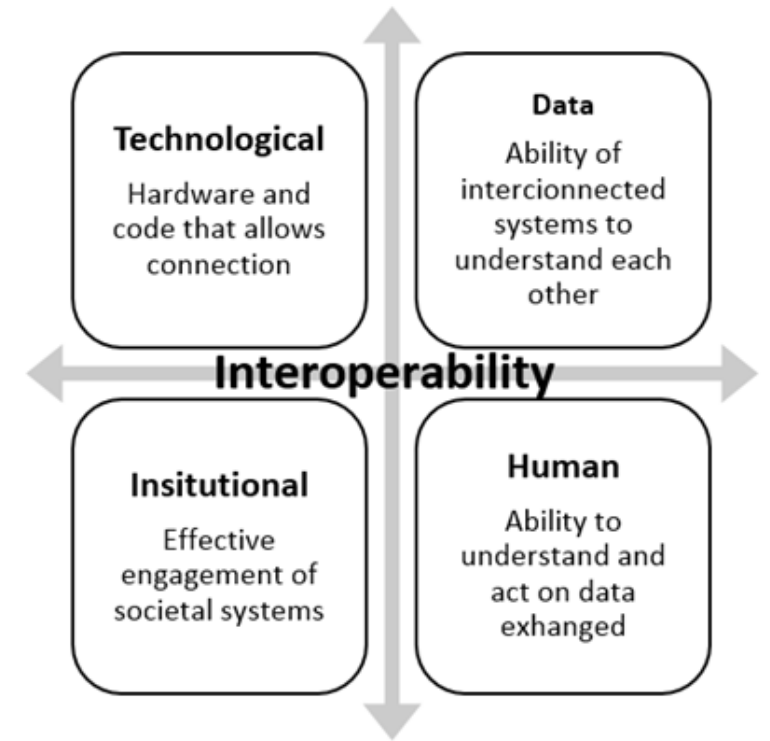
Recapitulare – Arhitectura N-Tier

- Arhitectura pe N niveluri (N-Tier)
 - Aplicații compozite
 - Fiecare nivel poate corespunde unor funcționalități sau servicii specifice
- Exemplu (4-Tier)
 - Presentation/User interface
 - Application/Business logic
 - Data Management
 - Infrastructure/Integration



Probleme de integrare

- Atunci când avem mai multe componente / niveluri arhitecturale este necesară o analiză a **interoperabilității**
 - Se potrivesc componentele? (structură, comportamente, interfețe)
 - Corespund acestea cerințelor sistemului integrat? (cerințe funcționale / non-funcționale)
 - Ce probleme pot să apară la integrarea lor și cum pot fi ele soluționate?

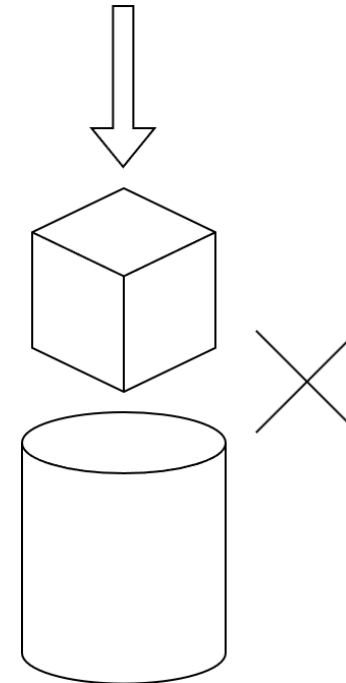


Interoperability | The IT Law Wiki | Fandom



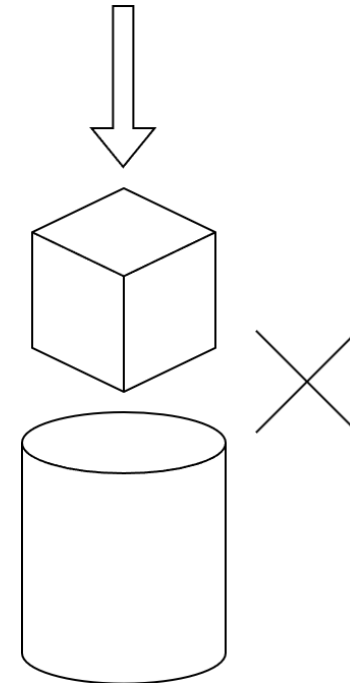
Probleme de natură tehnică

- Interoperabilitate
 - Capacitatea sistemelor / componentelor de a furniza sau accepta serviciile altor sisteme / componente în vederea cooperării eficiente
- Noțiuni conexe:
 - Modularitate - Capacitatea de a delimita capabilitățile sistemului în blocuri funcționale
 - Integrabilitate - Capacitatea de a incorpora componente în cadrul unui sistem mai mare / funcțional / unificat



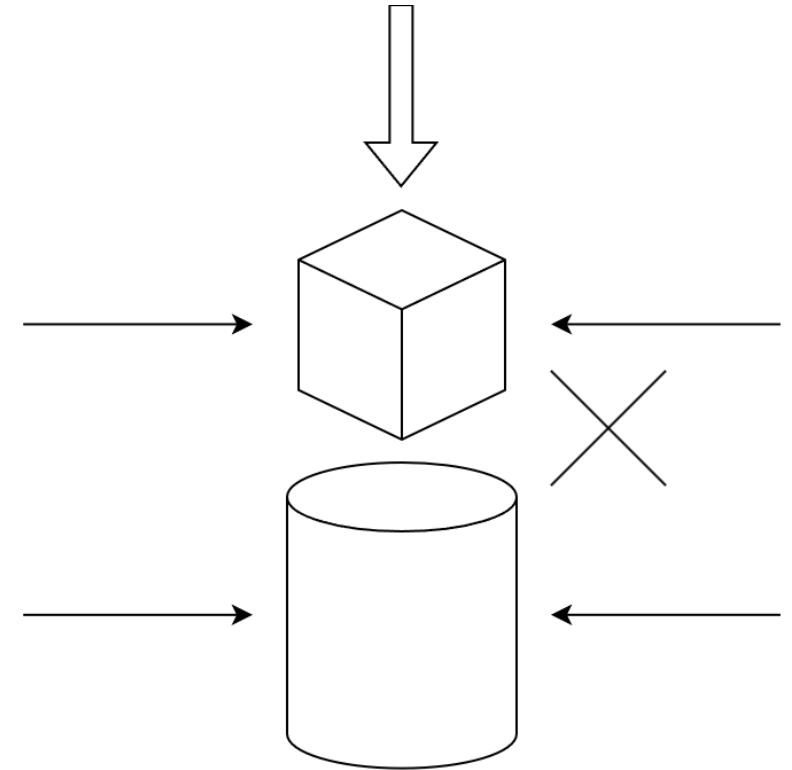
Probleme de natură tehnică

- Nepotriviri între **comportamentul** și **reprezentarea** informațiilor la nivelul unei interfețe, inclusiv protocoale folosite, structuri de date, reprezentări ale datelor, tehnologii de comunicare
- Sunt probleme de **interoperabilitate**



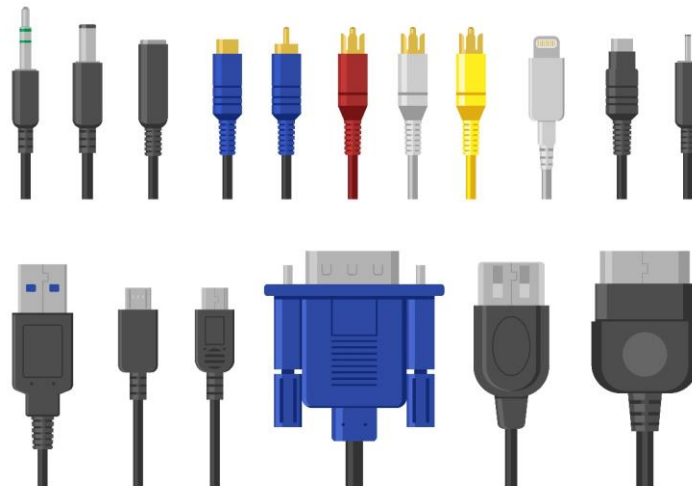
Conflicte de conexiune

- Apar când două resurse nu se pun de acord în privința **specificațiilor tehnice** necesare realizării conexiunii
- **Interfețele** specifică attributele tehnice ale comunicării cu o anumită componentă
 - mecanisme de comunicare
 - protocoale



Conflicte de conexiune

- *Conflict la nivel de mecanism de comunicare* – apare când interfețele a două componente nu folosesc același **mecanism de comunicare**
- *Conflict la nivel de protocol* – apare când două interfețe nu se potrivesc în ceea ce privește **protocolul de comunicare** la nivel aplicație și la toate nivelurile inferioare



Soluție: Transformarea conectorilor

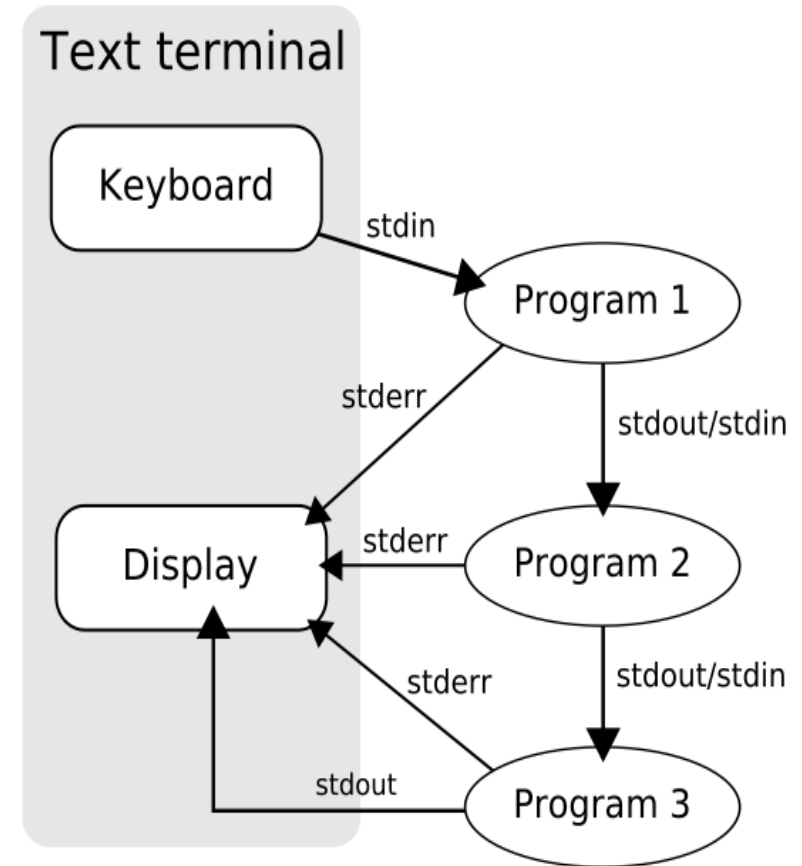
aplicabilă dacă:

- nu este nevoie să reorganizăm conținutul comunicării (mesajele)
- este suficient să modificăm mecanismele sau protocoalele prin care se realizează comunicarea



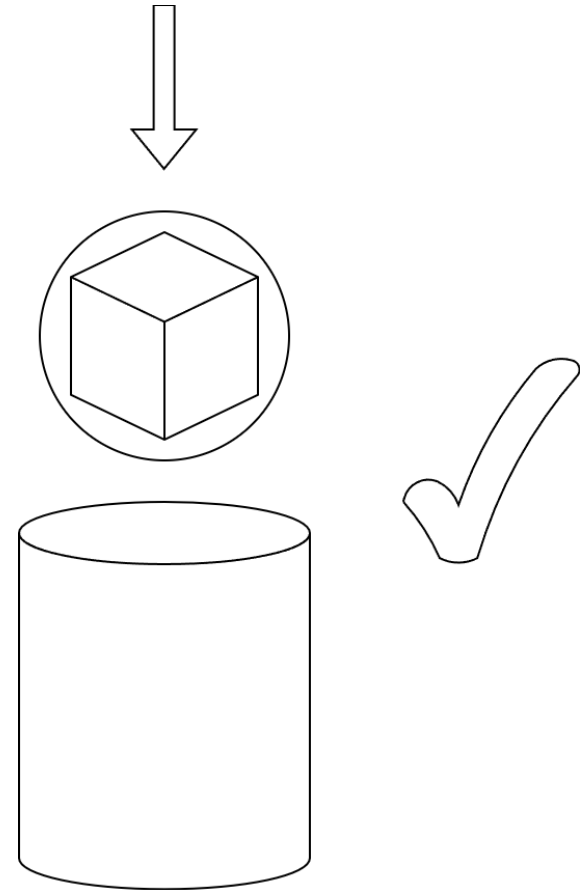
Exemplu: pipeline și fișiere

- O componentă a sistemului vrea sa citească date dintr-un pipeline, dar componenta care transmite datele poate doar să citească și să scrie în fișier
- Rezolvare: folosirea unui adaptor pipe-to-file cu ajutorul unui limbaj de scripting: redirectarea output-ului



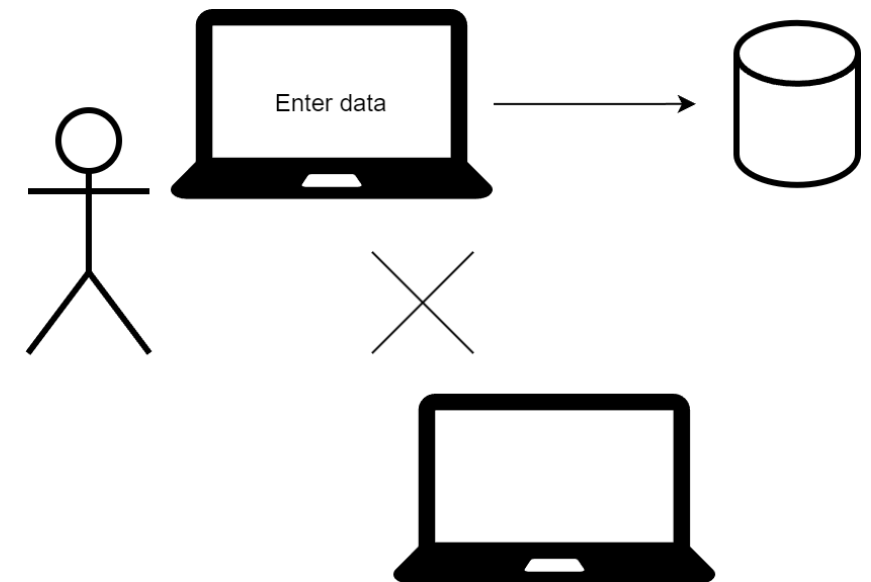
Soluție: Împachetarea

- Folosită atunci când trebuie să reorganizăm structura informației, pentru a putea fi transmisă între două componente
- “wrappers” – transformare de protocol



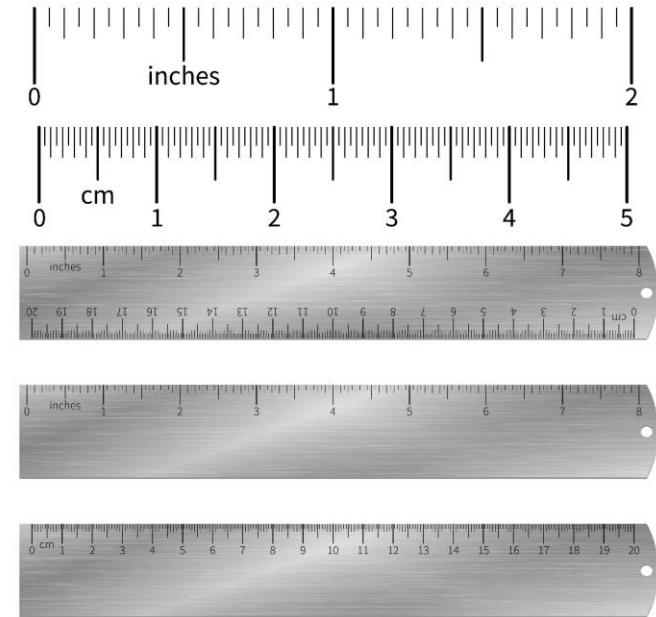
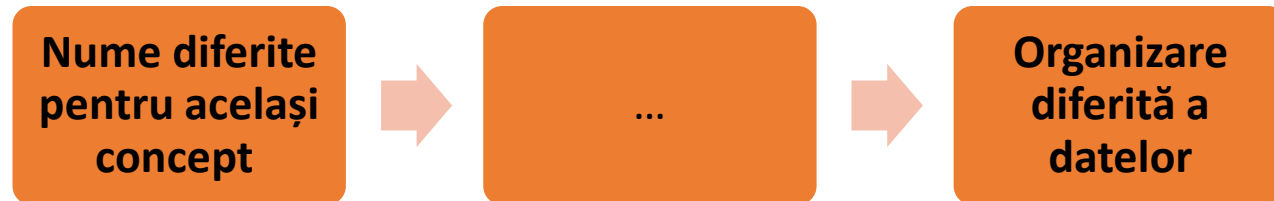
Soluție: Modificarea componentei

- Exemplu: O componentă trebuie să transmită date unei aplicații, care are o interfață unică pentru date – **interfață grafică**, destinată informațiilor introduse manual, de către om
- Nicio transformare asupra conectorilor nu este fezabilă în acest caz, singura soluție ar fi adăugarea unei noi interfețe pentru datele aplicației, care să permită **recepționarea datelor** în mod automat
- Această soluție presupune accesul la codul intern al aplicației



Conflicte sintactice

- Apar când componentele care comunică folosesc **structuri de date** și **reprezentări** diferite pentru același concept (obiect, acțiune, proprietate, relație)



Conflicte sintactice

Soluție

- Existența unei componente intermediare care face “traducerea” între cele două reprezentări
- În multe cazuri, această **componentă intermediară** va face atât o **transformare a protocolului**, cât și o **traducere sintactică**
- Exemplu: conversie fișiere multimedia, conversie /serializare obiecte

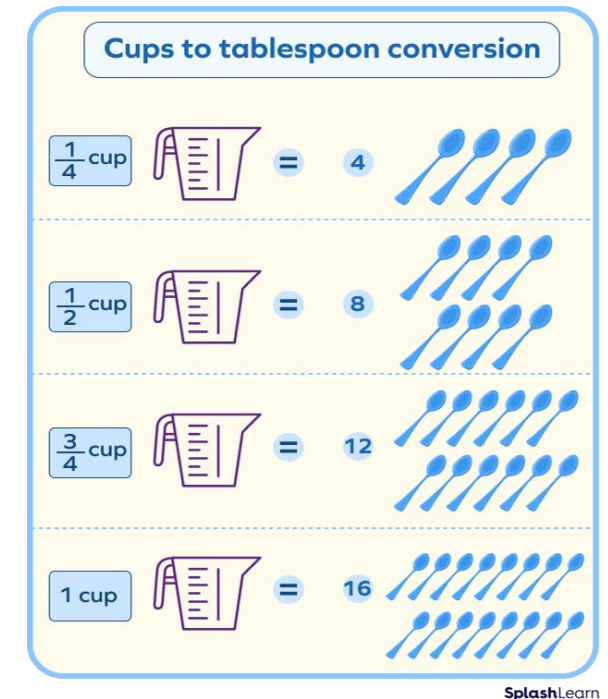


Conflicte sintactice

Soluție

O astfel de componentă intermediară se poate obține în 3 pași:

- 1) Folosirea unui **translator** pentru a converti reprezentarea schematică a datelor sau modelul interfeței componentei sursă într-un limbaj apropiat componentei țintă
- 2) Folosind un limbaj apropiat convențiilor de reprezentare a componentei țintă, specificarea unui **model de conversii** (de nume și de reorganizare a datelor) din reprezentările folosite în componenta sursă, către reprezentările folosite în componenta destinație
- 3) Construirea componentei intermediare cuplând cele două translatoare

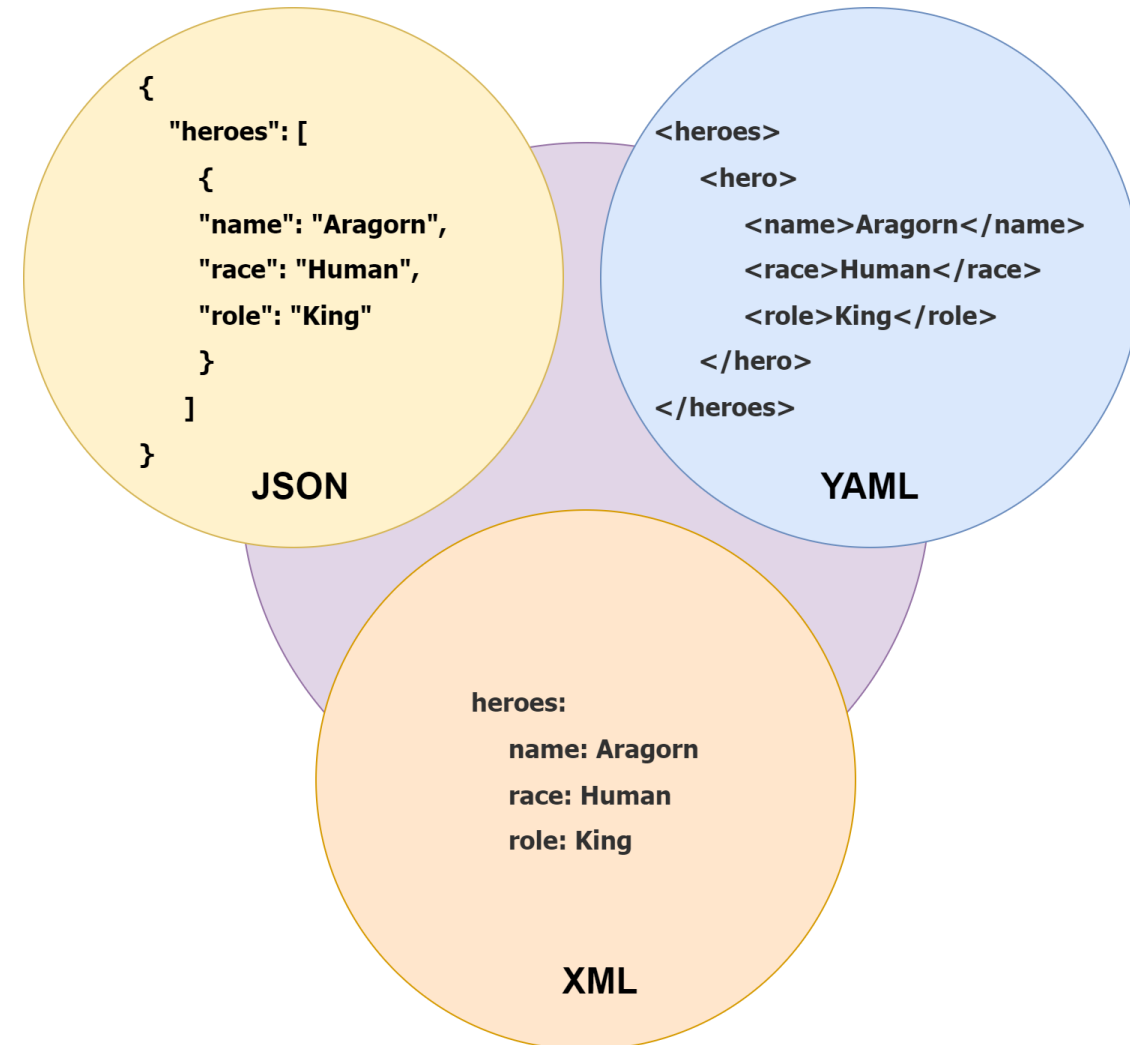


Sursa: SplashLearn

Conflicte sintactice

Studiu de caz: serializare obiecte

- Reprezintă un mod prin care aplicații diferite pot comunica între ele
- Exemple de sintaxă:
 - JSON (JavaScript Object Notation)
 - XML (eXtensible Markup Language)
 - YAML (YAML Ain't Markup Language)
- Exemple de protocol transfer:
 - Transfer date: HTTP, WebSocket
 - Transfer fișiere: FTP
 - Sisteme de fișiere: NTFS, FAT32



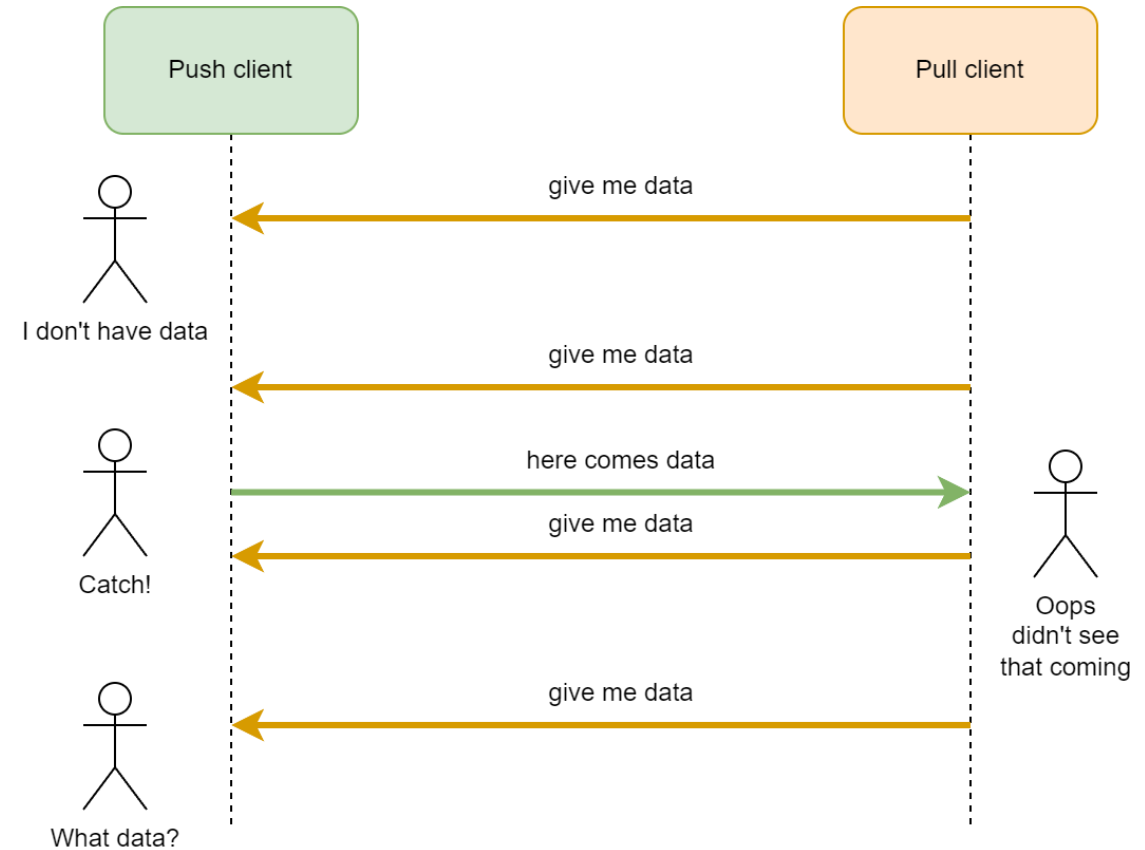
Conflicte de control

- Apar când componentele se bazează pe ipoteze diferite legate de **fluxul controlului** în interacțiunea dintre ele
- Este o problemă tehnică, când două componente nu “cad de acord” în ceea ce privește rolurile lor funcționale: cine furnizează informații cui, cine face o anumită acțiune, etc.

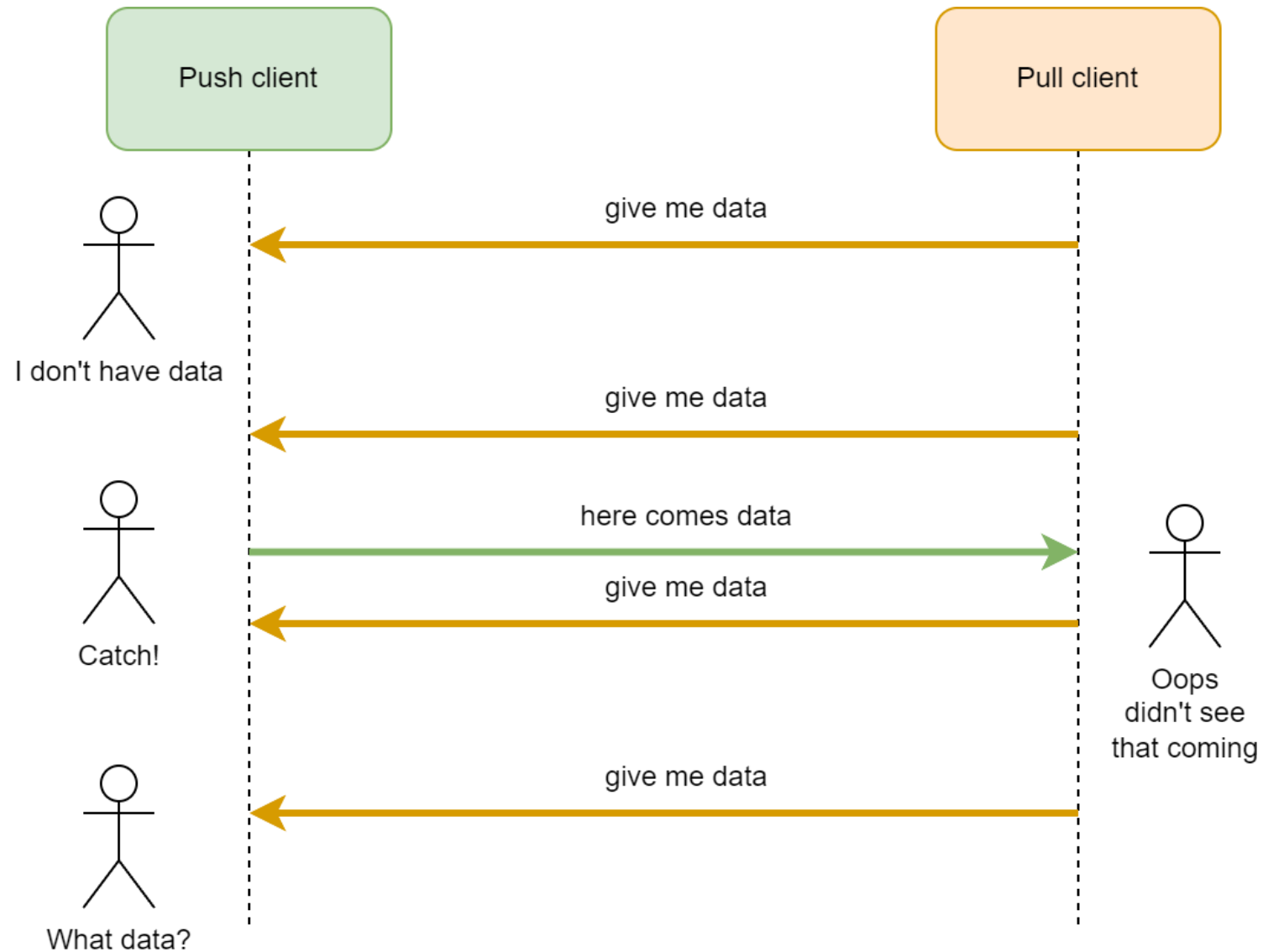


Problemă – “prea mulți lideri”

- Ambele componente se așteaptă să fie client – care invocă un serviciu de la o componentă server, dar nicio componentă nu își asumă rolul de server și nu răspunde la cererile celorlalți clienți
- Exemplu: un client “push” transmite informații unui client “pull”

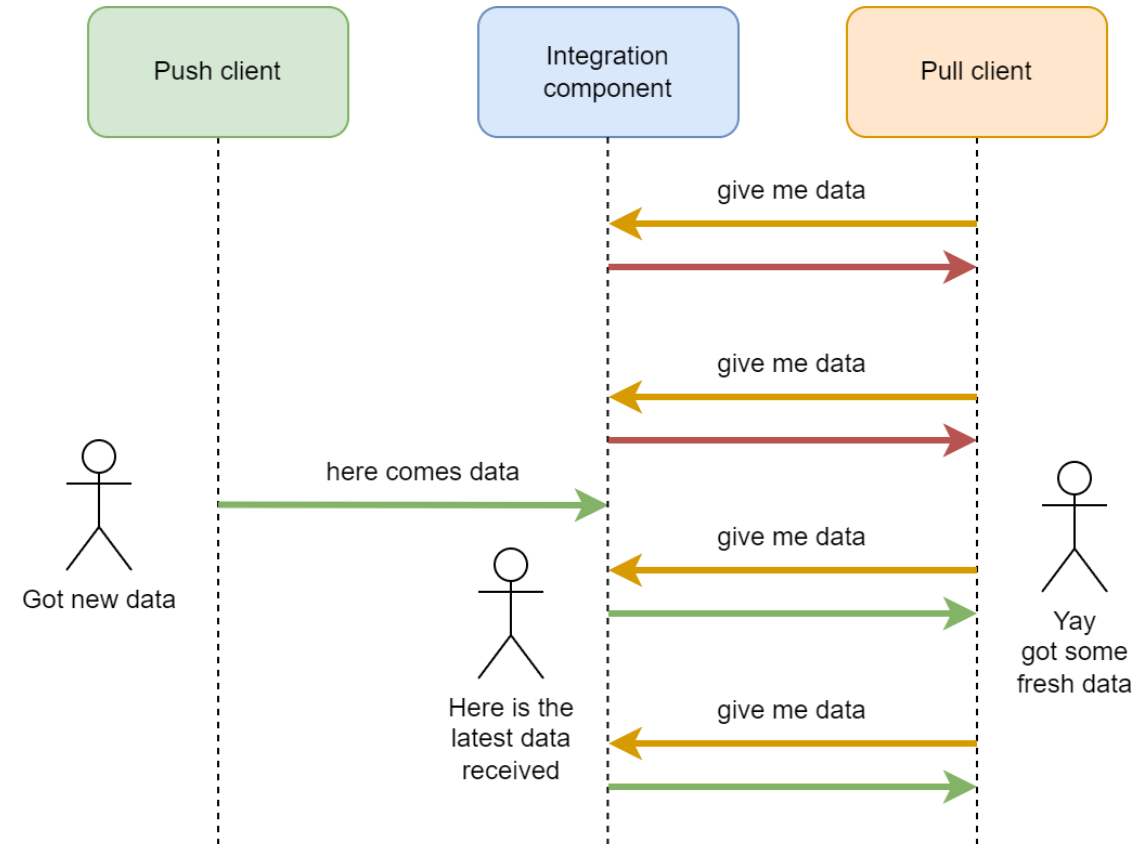


Problemă – “prea mulți lideri”

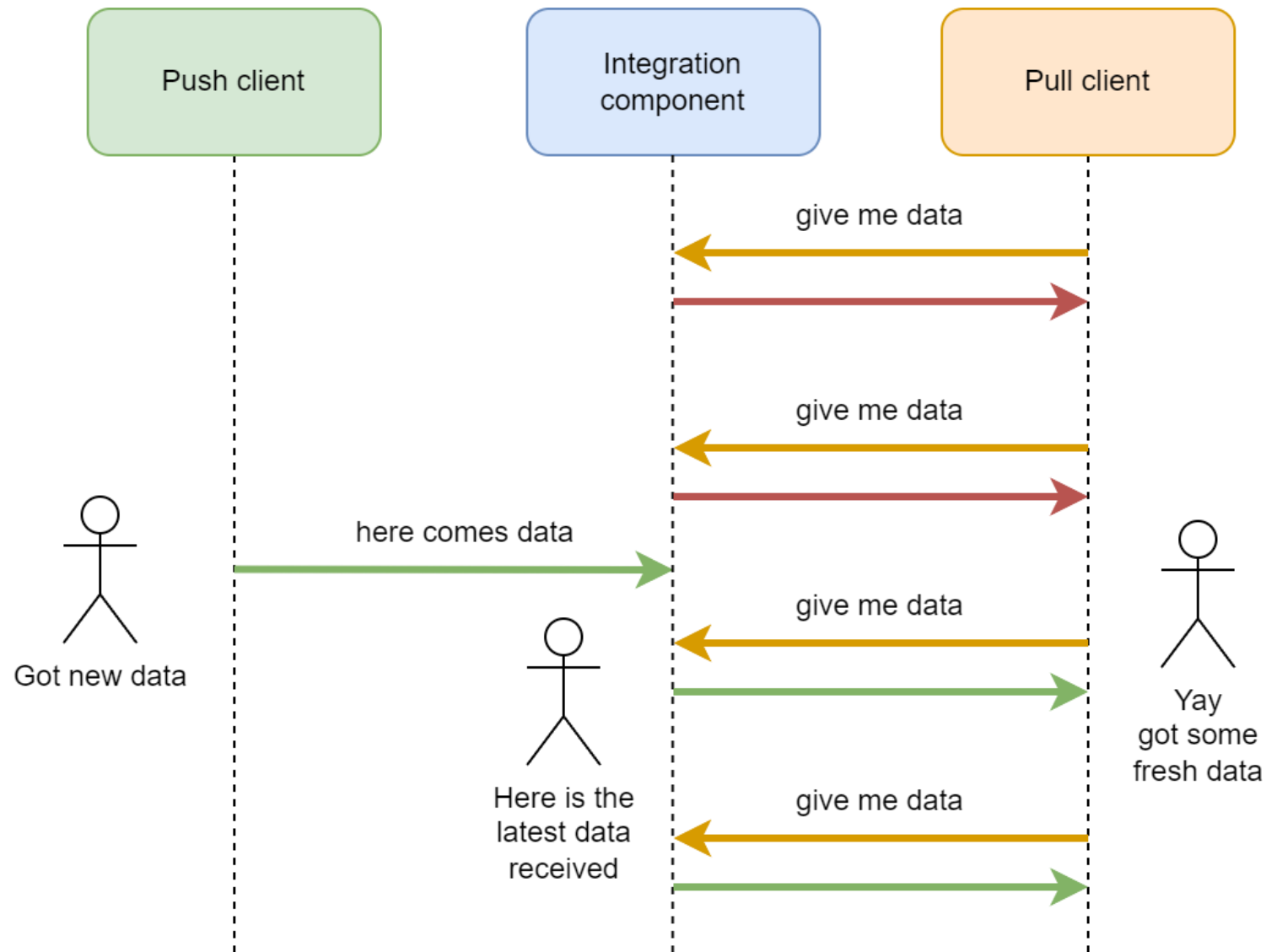


Soluție

- Unul dintre clienți preia rolul de server (dacă se poate modifica)
- În majoritatea cazurilor, cel care integrează sistemul trebuie să implementeze o **componentă intermediară** care să fie
 - atât un server push, care acceptă și pune în coadă mesajele de la clienții push
 - cât și un server pull, care livrează mesajele la clienții pull

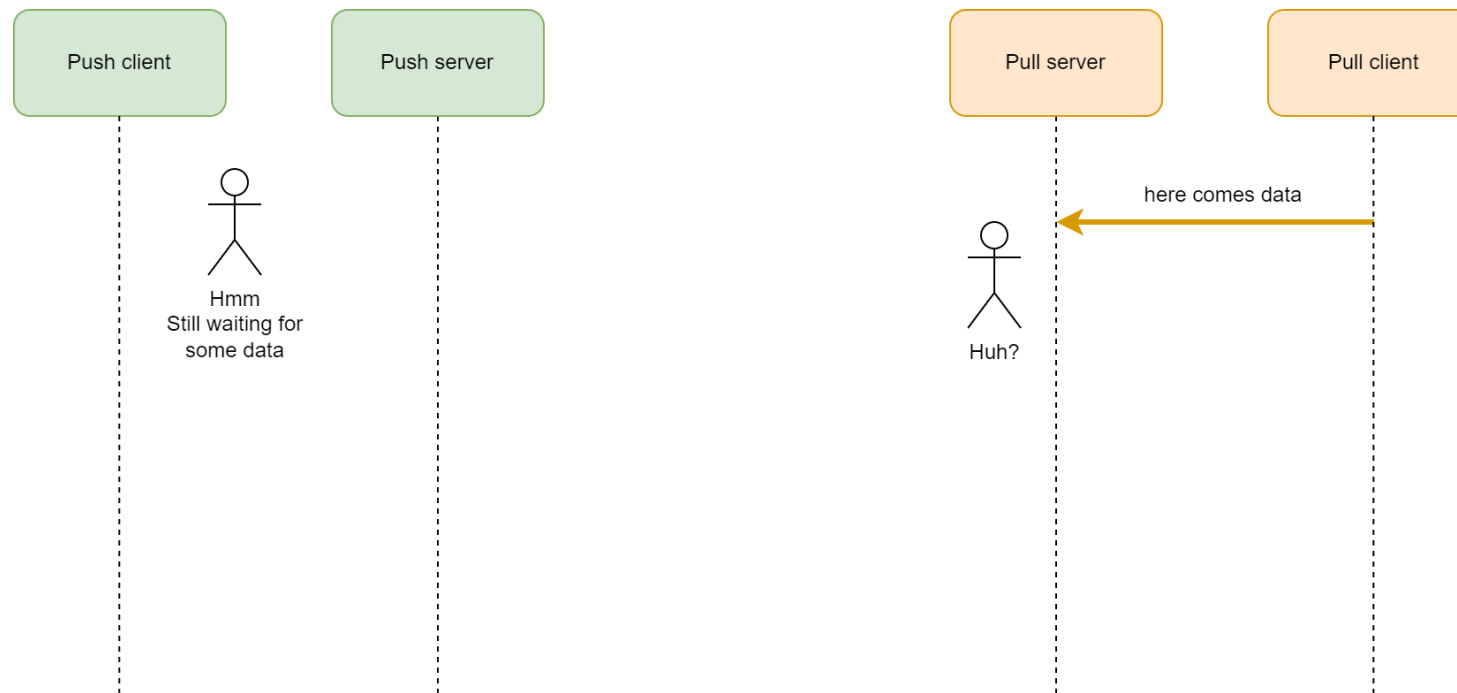


Soluție – “prea mulți lideri” – componentă intermediară

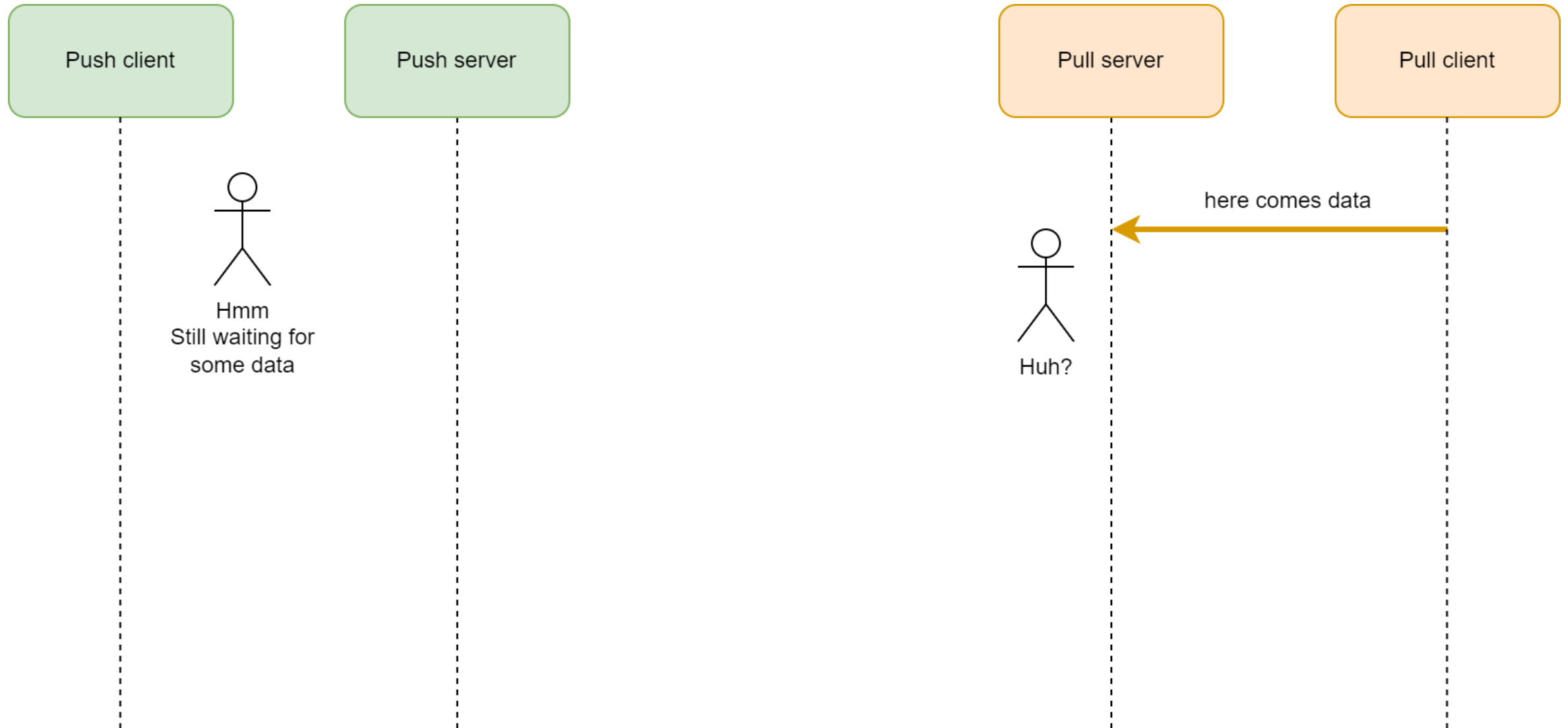


Problemă – “niciun lider”

- Ambele componente preiau rolul de server, dar niciuna nu preia rolul de client care sa inițieze cererile către server
- Ex. un server pull trebuie să transmită informații unui server push

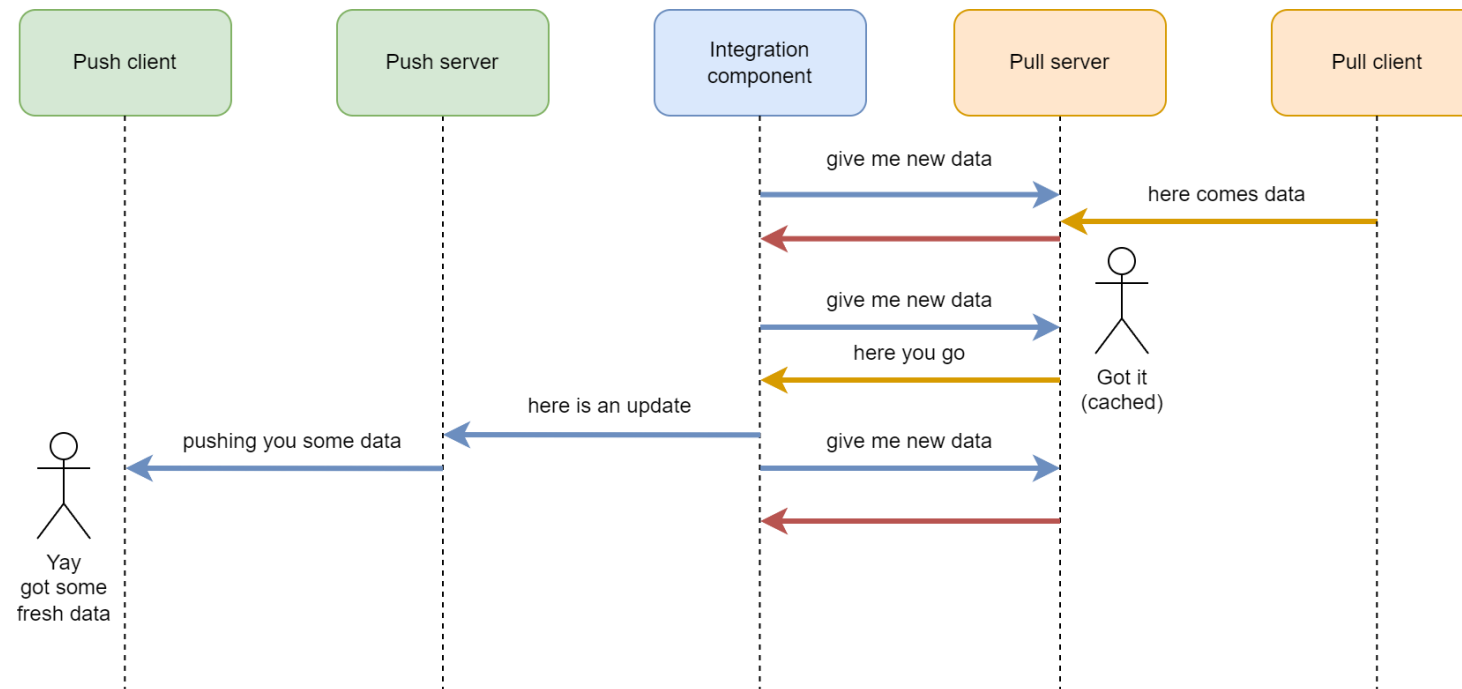


Problemă – “niciun lider”

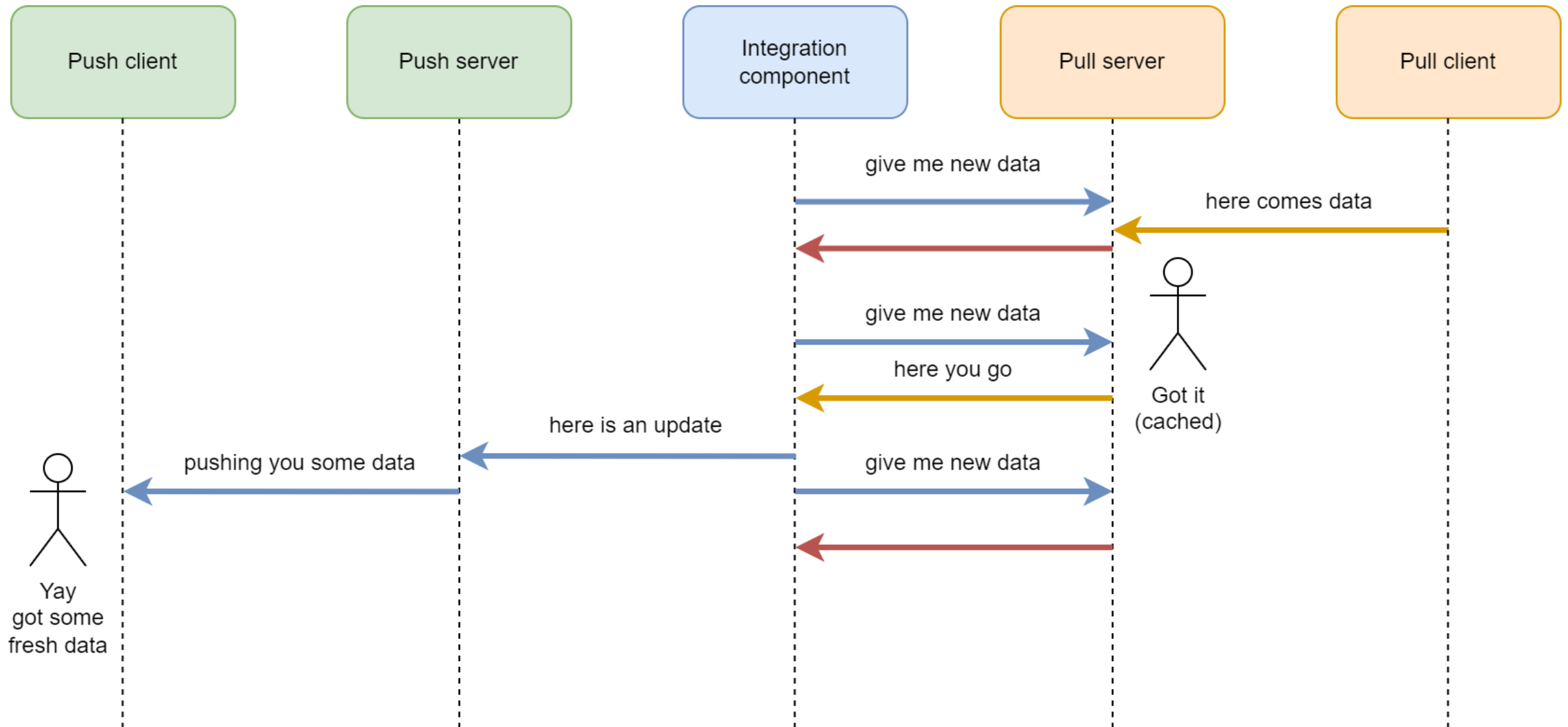


Soluție

- Integratorul adaugă o componentă care implementează schimbul, fiind un client pull pentru serverul pull și un client push pentru serverul push



Soluție – “niciun lider” – componentă intermediară



Conflicte legate de calitatea serviciului

- Apar când comportamentul unei componente nu satisface anumite cerințe tehnice, care se referă la performanță, siguranță, etc.
- Apar și când o componentă îndeplinește anumite cerințe tehnice și se așteaptă ca și componentele cu care comunică să îndeplinească aceleași cerințe tehnice, chiar dacă sistemul nu are aceste cerințe



Image by Freepik

Exemple

- O componentă client cere ca rezultatele să fie exprimate cu 10 cifre după virgulă, dar serverul efectuează calcule în precizie simplă, cu 7-8 cifre după virgulă
- Se așteaptă ca o componentă să lucreze în timp real și să răspundă într-un anumit interval de timp
- Se așteaptă ca o componentă să folosească un mecanism de comunicare securizată, pentru anumite tranzacții
- Se așteaptă ca o componentă să repornească automat după o eroare și să își refacă starea internă, pentru a fi tolerantă la defecte

Soluții

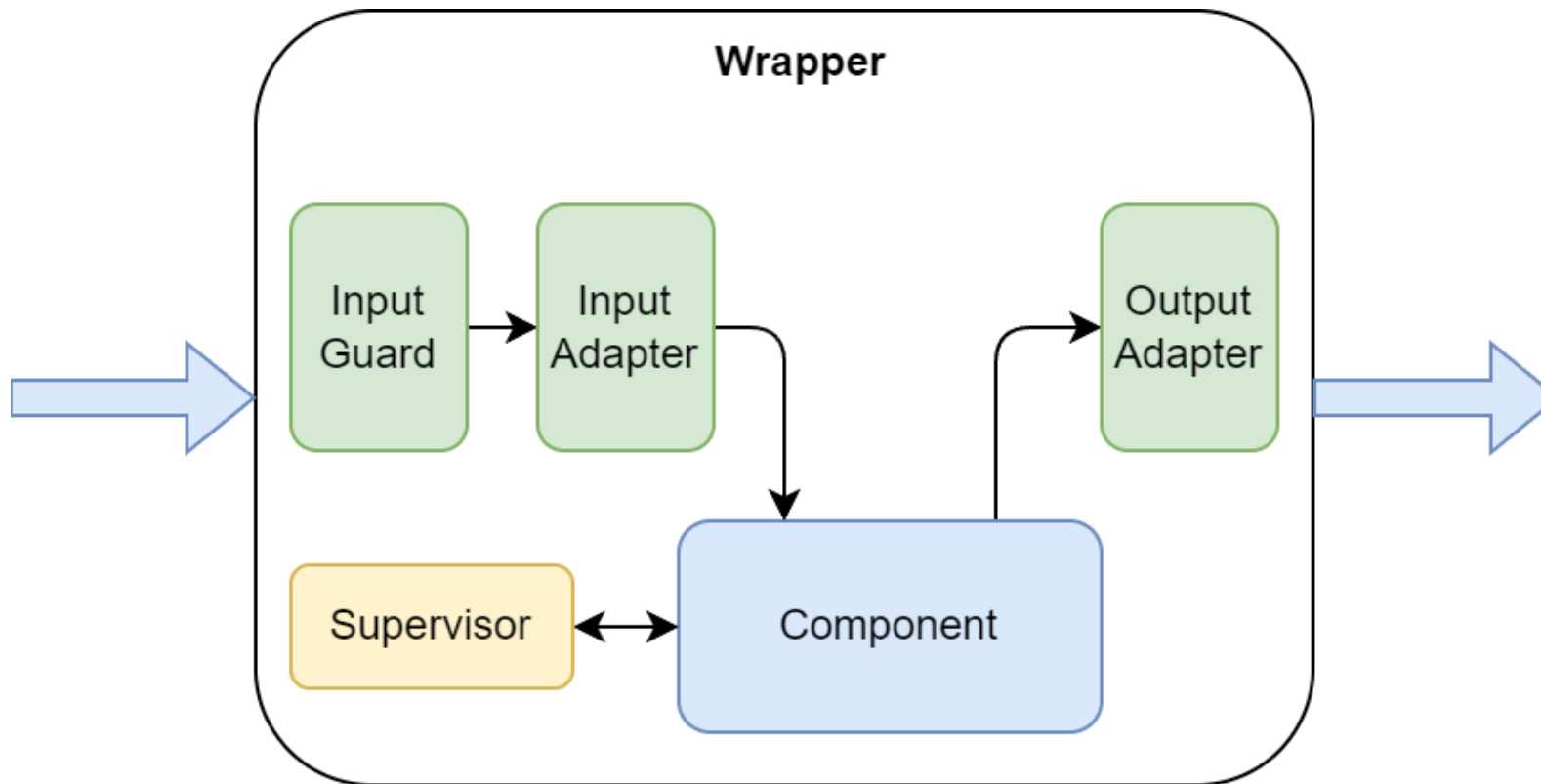


- În general este necesară modificarea componentelor
- Alte posibilități – împachetare (wrapper)



- Preprocesor de securitate care validează sau anulează fiecare tranzacție (dacă validarea nu presupune date interne ale componentei)
- Coadă de procesare care reține ultima actualizare și transmite comenzile (tot nu va funcționa în timp real, dar va returna datele /controlul către componenta apelantă)
- Supervizor (script) care asigură restartarea automată (recuperarea stării interne va fi aproximativă)

Soluții – împachetarea componentei

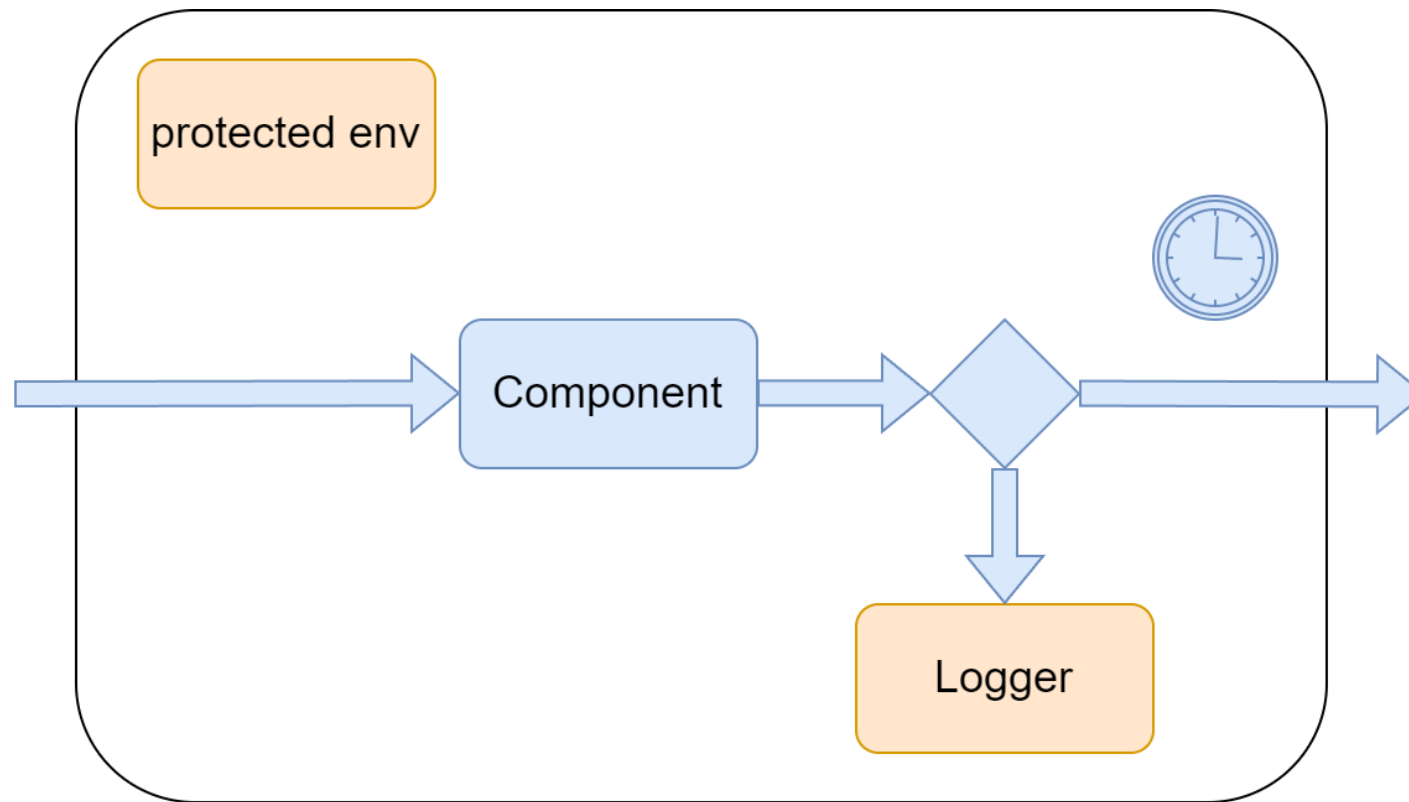


Soluții

- Alte posibilități – Reducerea / relaxarea cerințelor
 - Termenele limită pot fi setate astfel încât să permită componentei să răspundă în timp util
 - Atributele de securitate pot fi setate corespunzător modului public sau protejat (adică nu atât de securizate ca pentru modul privat)
 - Mesajele ce trebuie recuperate în caz de eroare software pot fi interceptate de o componentă intermediară care le transmite mai departe și așteaptă o confirmare sau le ignoră

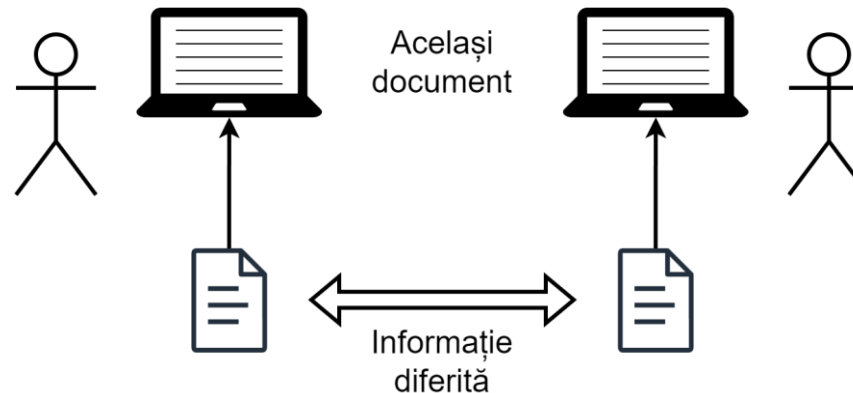


Soluții – relaxarea cerințelor



Conflicte legate de consistența datelor

- O interacțiune între două componente se poate baza pe niște date care sunt disponibile ambelor componente și nu direct comunicate între ele
- Când acele date/informații au fost obținute la momente de timp diferite, din locuri diferite sau deduse în moduri diferite, poate apărea o **inconsistență a datelor** între cele două componente
- Un conflict legat de consistența datelor apare când avem o astfel de inconsistență care afectează comportamentul componentelor din punct de vedere al acțiunilor și funcțiilor sistemului



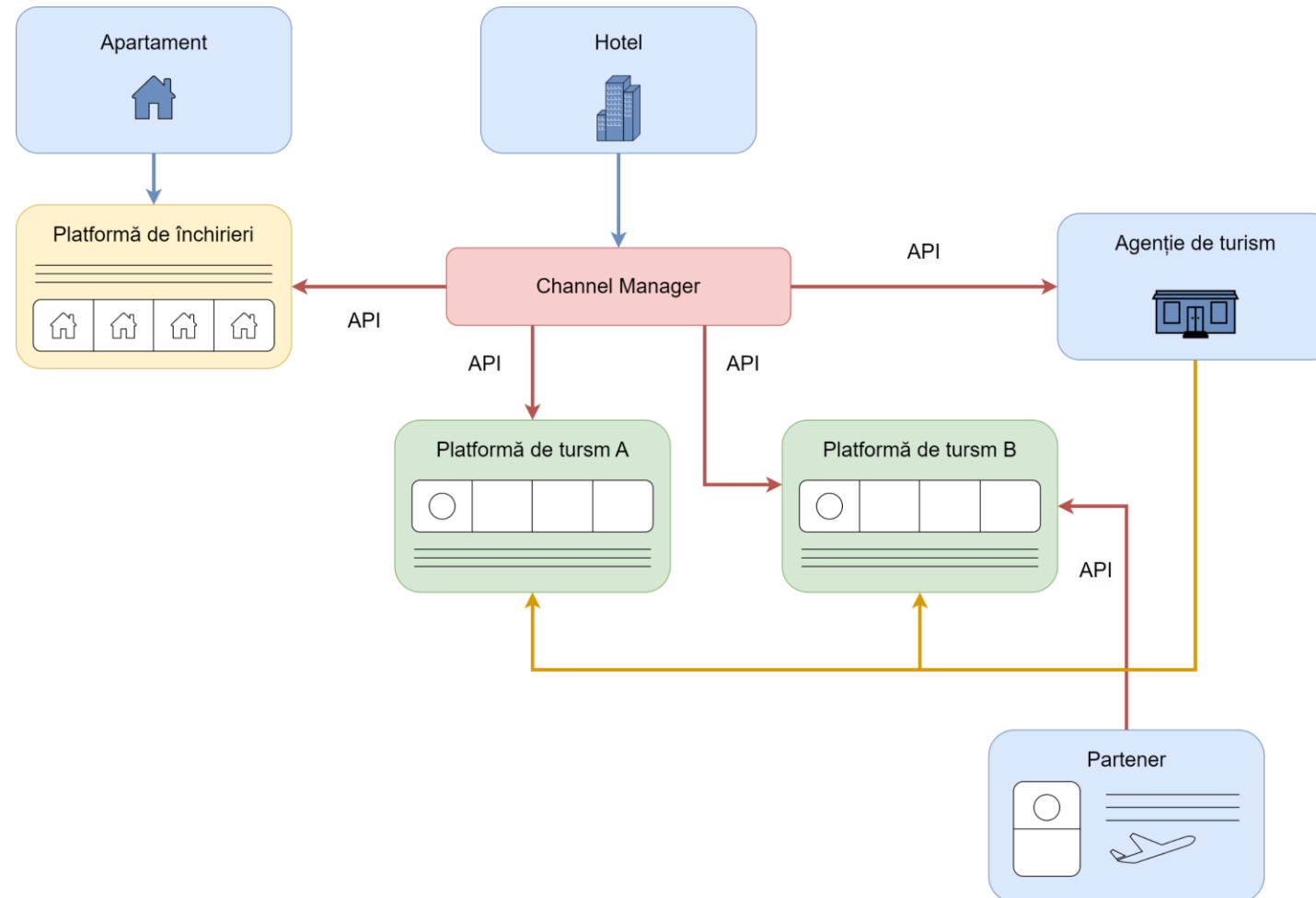
Conflicte legate de consistența datelor

Exemple

- Un sistem ERP (Enterprise Resource Planning) inițiază o comandă de la un furnizor bazată pe o intrare în catalogul intern, dar sistemul de achiziții nu poate plasa comanda deoarece catalogul online al furnizorului nu mai conține acele produse
- Când componenta care plasează comenzi calculează numărul de obiecte ca fiind $numar_obiecte_produse - numar_obiecte_consumate$, dar mai sunt și obiecte care sunt în procesul de a fi consumate, se poate întâmpla ca acea componentă să comande mai multe obiecte decât există pe stoc

Conflicte legate de consistența datelor

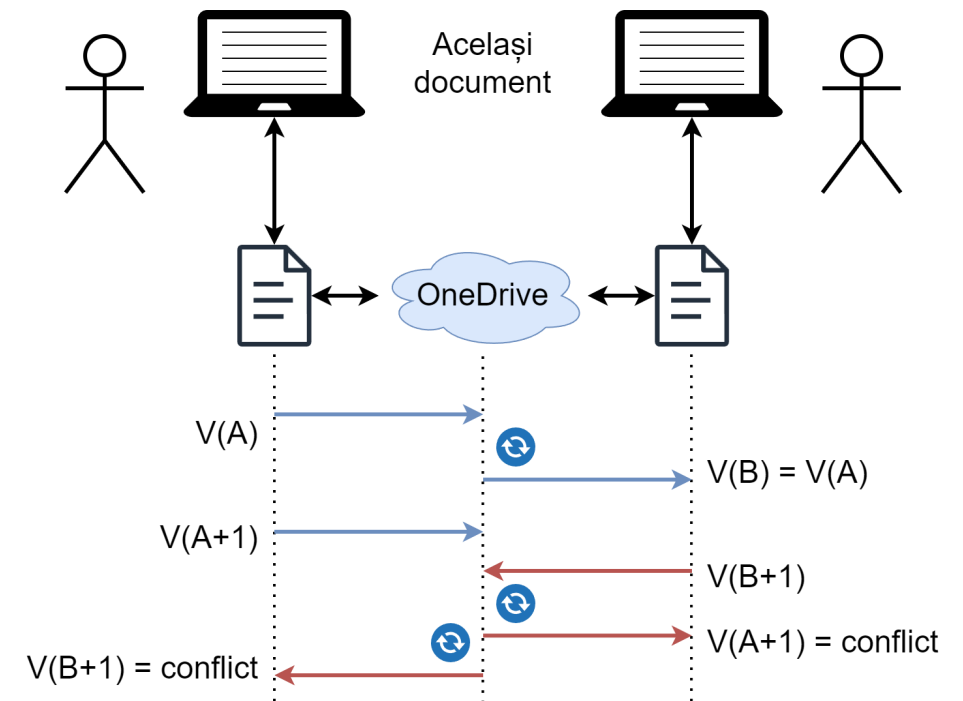
Exemplu: sistem de rezervări camere hotel (identificați scenariile problematice)



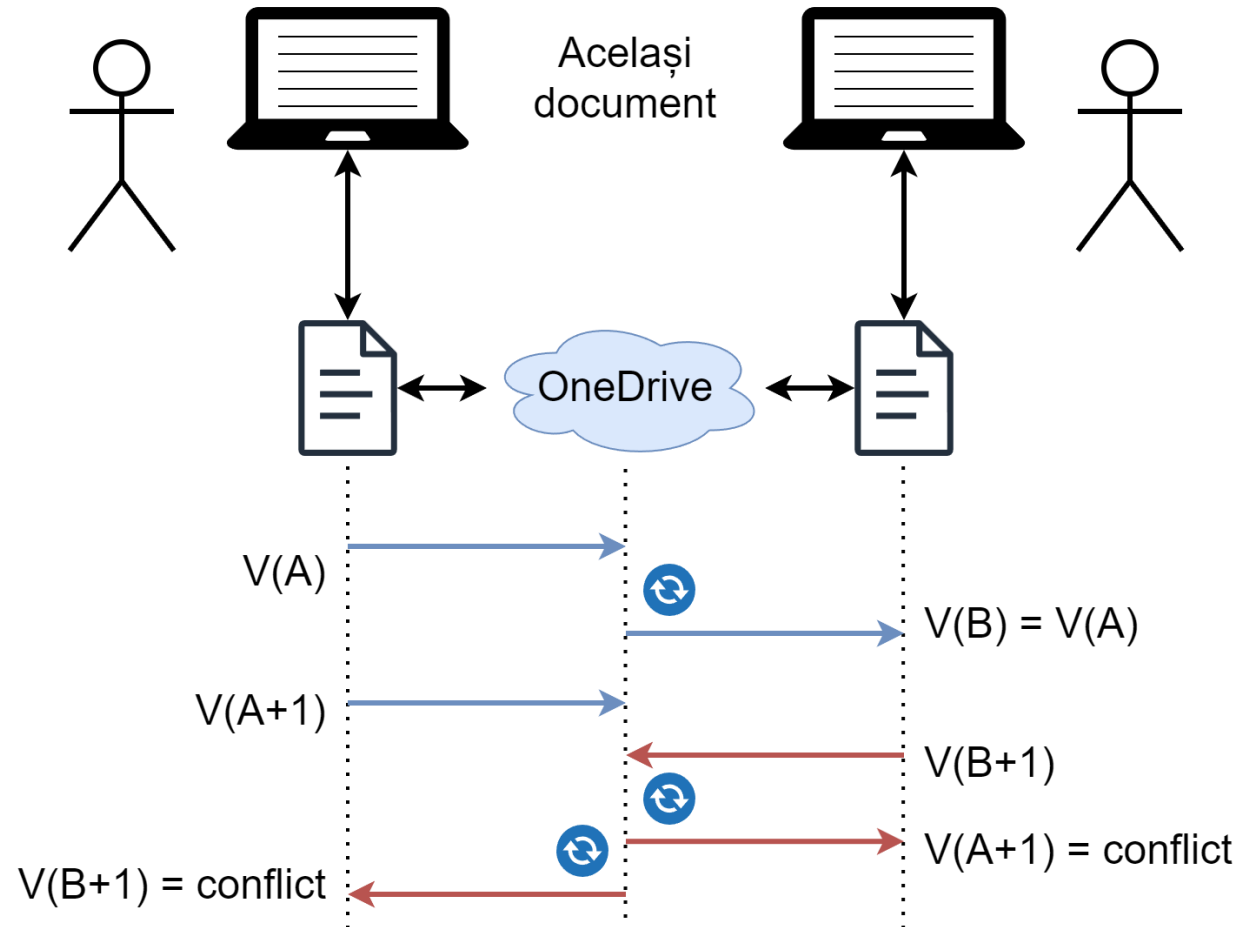
Conflicte legate de consistența datelor



- Când problema este accesarea aceleiași surse de informații, la momente de timp diferite, se poate întâmpla ca între momentul t_1 și momentul t_2 informația să fie modificată astfel încât cele două componente să primească date diferite, din aceeași sursă = “**race condition**” (rezultatul diferă în funcție de cine accesează primul informația)

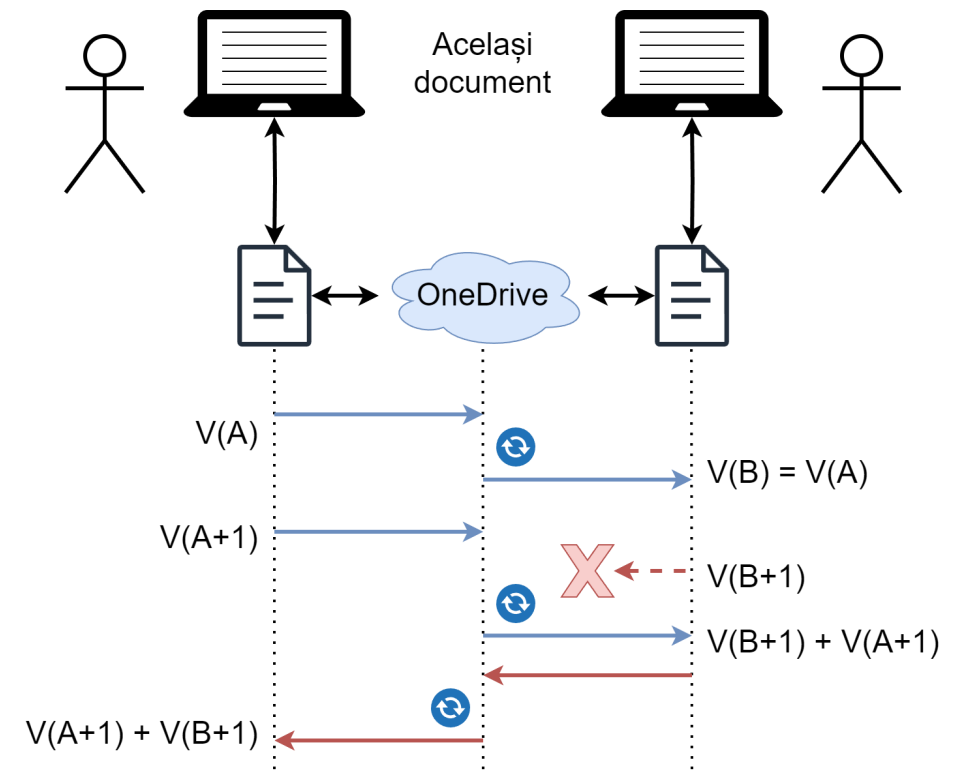


Conflicte legate de consistența datelor – “race condition”



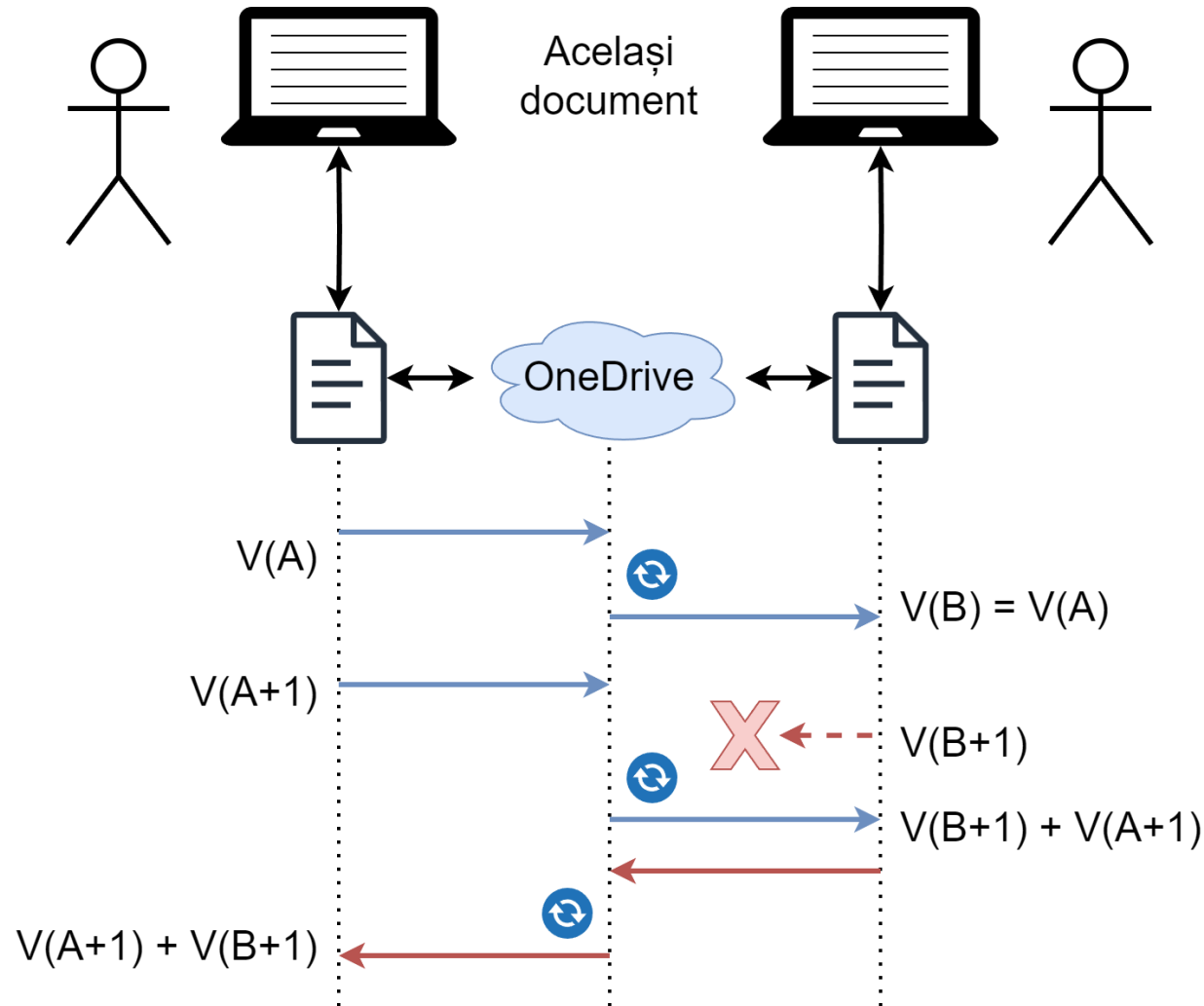
Conflicte legate de consistența datelor

- Soluții diferite, în funcție de caz
 - programarea acțiunilor sau sincronizarea lor, astfel încât la momentul t_1 ambele componente să citească datele și să nu se poată modifica valoarea până când nu au citit ambele componente informația
 - forțarea unei componente să recitească datele, după ce acestea au fost modificate
 - adăugarea structurii informaționale în comunicarea dintre cele două componente



Conflicte legate de consistența datelor – “race condition”

Soluție: forțarea unei componente să recitească datele, după ce acestea au fost modificate



Conflicte legate de consistența datelor

- Spre deosebire de celelalte conflicte tehnice care au o metodă clară de rezolvare, conflictele de consistență a datelor nu pot fi prea ușor automatizate pentru că ele necesită o **analiză detaliată** pentru a determina exact care este problema și ce soluție ar funcționa mai bine pentru sistemul în discuție.



Image by Freepik

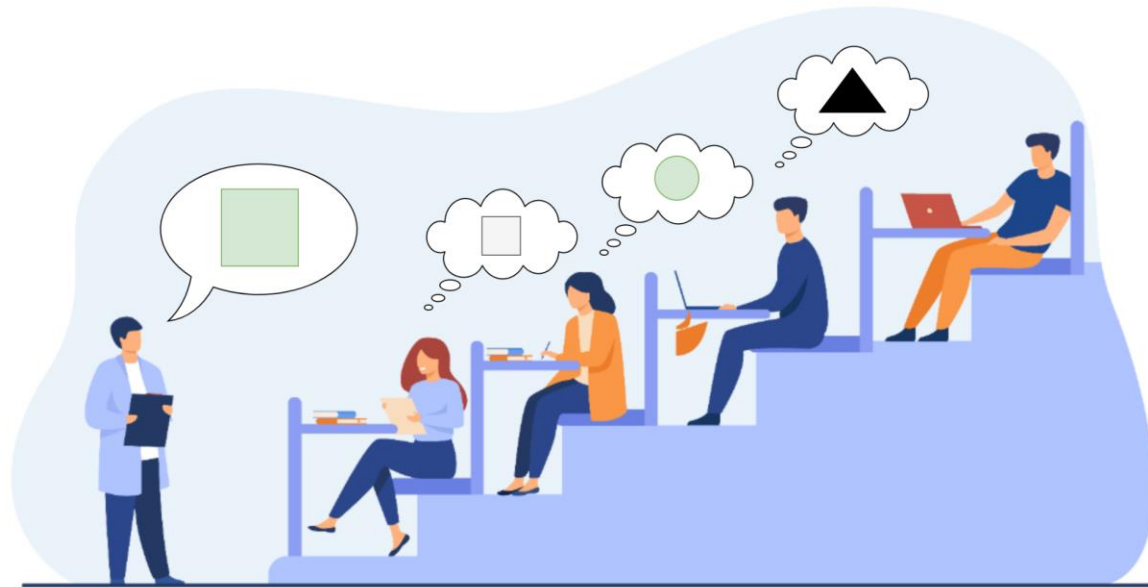
Conflicte semantice

- Cerința semantică pentru integrare este ca părțile ce comunică să se bazeze pe **concepte comune** și să fie de acord cu privire la **regulile de folosire** ale acelor concepte
- Conflictele semantice rezultă din
 - neconcordanțele între comunicarea dintre componente, în
 - modelul conceptual de date și spațiile de activitate în care acestea trebuie să interacționeze, sau în
 - interpretarea referințelor de tipuri de obiecte și a situațiilor în acele spații



Conflicte semantice

- Tipuri de comunicare între componente:
 - directă
 - indirectă
- Context comunicare:
 - comunicare 1-to-1
 - comunicare 1-to-many

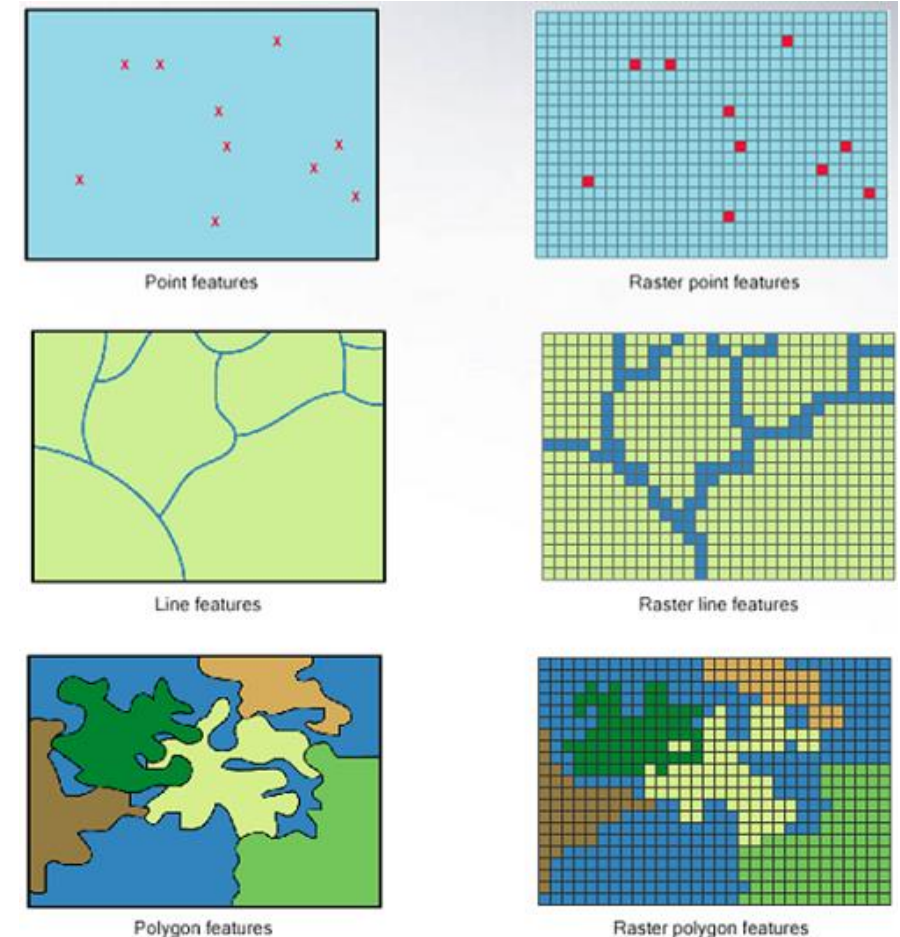


Tipuri de conflicte semantice

- conflicte de conceptualizare
- conflicte de domeniu de aplicare
- conflicte de interpretare
- conflicte de referință

Conflicte semantice – de conceptualizare

- Un **conflict de conceptualizare** apare atunci când componentele comunicante au modele de interfață reprezentând **implementări incompatibile** din același domeniu conceptual
- Exemplu: grafică vectorială vs grafică raster/bitmap



Conflicte semantice – de conceptualizare

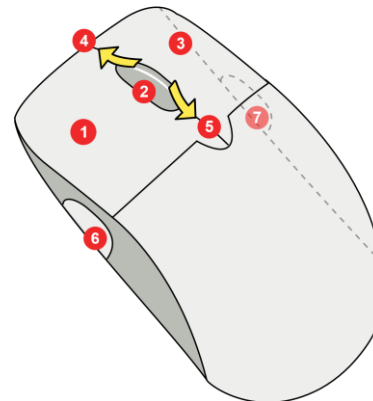
Soluții



- modele de translatare
 - se pot pierde informații utile
 - cazuri rare de translatare bidirecțională: $A \rightarrow B$ și $B \rightarrow A$
 - unele pot fi chiar netraductibile
- folosirea aceleiași implementări pentru un concept

Conflicte semantice – domeniu de aplicare

- Un **conflict generat de domeniul de aplicare** are loc atunci când un concept care este important pentru interacțiune din punctul de vedere al unei componente nu este prezent în modelul interfeței celeilalte componente
- Exemplu:
 - Mouse cu 3 butoane comunicând pe un protocol de mouse cu 2 butoane



Conflicte semantice – domeniu de aplicare

Soluții



- În cazul în care domeniul de aplicare al unei componente este **mai larg** decât este necesar pentru sistemul integrat, informațiile inutile care se produc pot fi ignorate
- Dacă domeniul de aplicare al unei componente este **mai îngust** decât este necesar pentru sistemul integrat, componenta este doar parțial integrabilă și funcționalitatea care lipsește trebuie să fie furnizată de către o alta componentă add-on

Conflicte semantice – de interpretare

Un conflict de interpretare apare atunci când **înțelegerea exprimării sistemului** de către componentă duce la un efect diferit de cel așteptat de către sistem sau inginerului de sistem.

Soluții



- O astfel de problemă este rezolvată printr-o împachetare (en. wrapper) care traduce unități de informație de la emițător către sistemul ascultător
- Folosirea interfețelor standard

Conflicte semantice – de referință

- Un conflict de referință apare atunci când componentele comunicante folosesc **diferite sisteme de referință** pentru a identifica anumite concepte
- Exemplu: cerc vs elipsă

Conflicte semantice – de referință

```
class Ellipse
{
    private double dx;
    private double dy;
    public Ellipse(double x, double y)
    {
        dx = x;
        dy = y;
    }
    public void setLengthOfAxisX(double x)
    {
        dx = x;
    }
    public void setLengthOfAxisY(double y)
    {
        dy = y;
    }
}
```

```
class Circle extends Ellipse
{
    public Circle(double radius)
    {
        super(radius, radius);
    }
    public void setRadius(double radius)
    {
        setLengthOfAxisX(radius);
        setLengthOfAxisY(radius);
    }
}

...

Circle c = new Circle(10);
c.setLengthOfAxisX(20); // nu mai e cerc
```

Conflicte semantice – de referință

- Cum se rezolvă problema de mai devreme?
- Principiul Substituției Liskov este totuși un principiu fundamental al OOP din conceptul SOLID
 - Single responsibility principle
 - Open closed principle
 - Liskov substitution principle
 - Interface segregation principle
 - Dependency inversion principle
- Este o problemă de semantică – înțelegerea obiectelor folosite pentru definirea conceptelor reprezentate



Conflicte semantice – de referință

Soluții



- Construirea unui **wrapper** sau a unui **intermediar** pentru a translata o sintaxă de referință la alta, folosind regulile de mapare corespunzătoare
- Conflictele de referință sunt strâns legate de conflictele sintactice, iar cele mai simple probleme pot fi rezolvate în același mod, de obicei, ca parte a **transformărilor de sintaxă**

Conflicte semantice – de referință

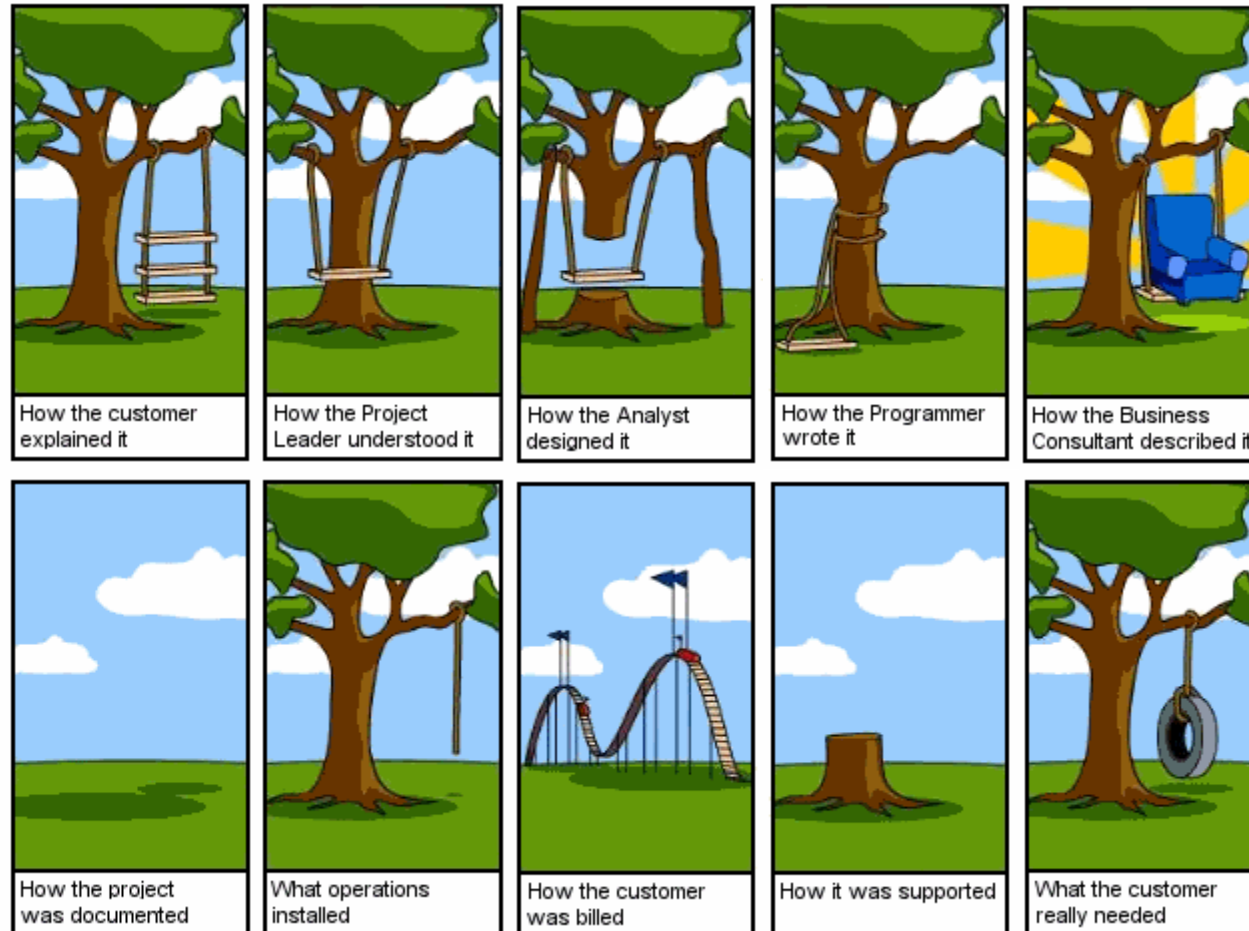
Soluții

- “Composition over inheritance” – este un principiu OOP de reutilizare a codului ce permite o flexibilitate mai mare
- Implementarea clasei *Circle* este un “wrapper” peste clasa *Ellipse*



```
class Circle
{
    private Ellipse e;
    public Circle(double radius)
    {
        e = new Ellipse(radius, radius);
    }
    public void setRadius(double radius)
    {
        e.setLengthOfAxisX(radius);
        e.setLengthOfAxisY(radius);
    }
    public Ellipse asEllipse()
    {
        return new Ellipse(e);
    }
}
```

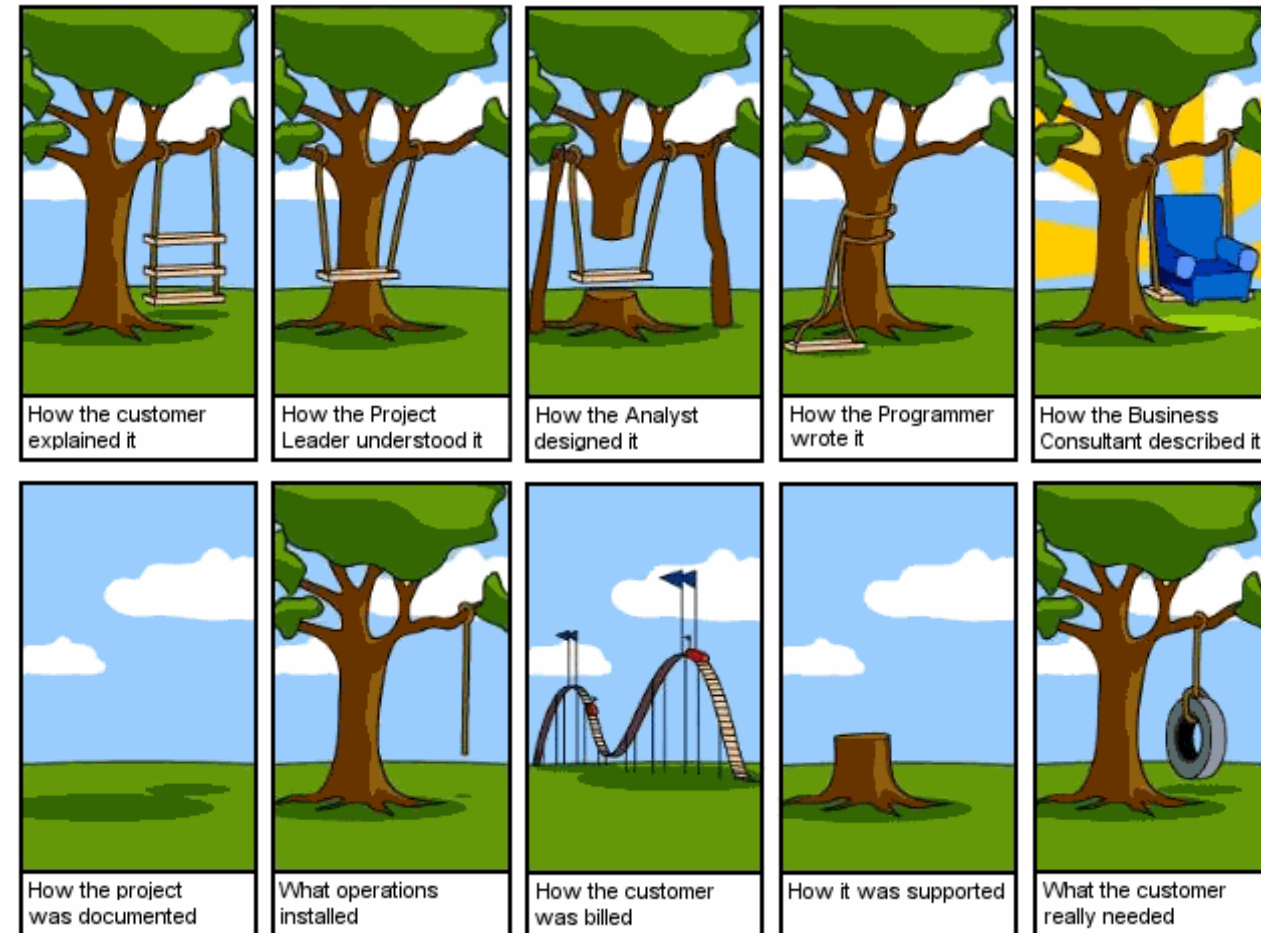
Conflicte funcționale



Conflicte funcționale

Conflictele funcționale apar atunci când **comportamentul** unei resurse diferă de cel așteptat de către altă resursă, cu care comunică

- Conflicte datorate **modelului funcțional**
- Conflicte datorate **domeniului de utilizare**
- Conflicte **intrinseci**
- Conflicte date de **utilizarea în alte scopuri** decât cele pentru care a fost destinată componenta




Probleme legate de politica sistemului

- Problemele legate de politica sistemului sunt date de necorelarea suficientă între comportamentul sistemului și **funcționalitățile așteptate**

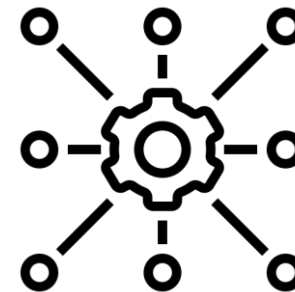
- Securitate
- Corectitudine, credibilitate și optimalitate
- Performanță / Răspuns în timp
- Siguranța în funcționare



-  • Nu întotdeauna se pot îndeplini toate criteriile simultan
 - Pot fi necesare anumite compromisuri
 - Ex. performanță vs corectitudine, performanță vs securitate

Probleme legate de logistică

- se referă la comportamentul unui sistem și al componentelor acestuia care ar putea influența proiectarea acestuia, dar nu sunt strâns legate de funcțiunile sistemului
- este strâns legată de **partea de management**
- 6P: cantitatea potrivită a bunurilor potrivite, la timpul potrivit, de calitate potrivită, la costurile potrivite, la locul potrivit
- Exemple: dimensionarea mediului de rulare, alocarea conturilor utilizator, gestionarea permisiunilor



Întrebări?

