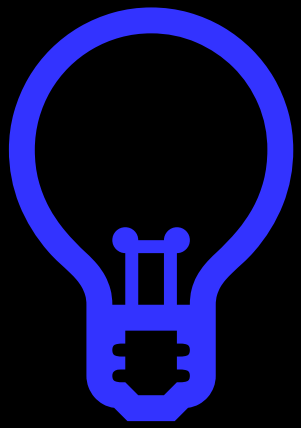


aplicații Web: aspecte arhitecturale

„Fiecare vis începe cu un visător.”

Harriet Tubman

De ce dezvoltăm aplicații Web?

Scop

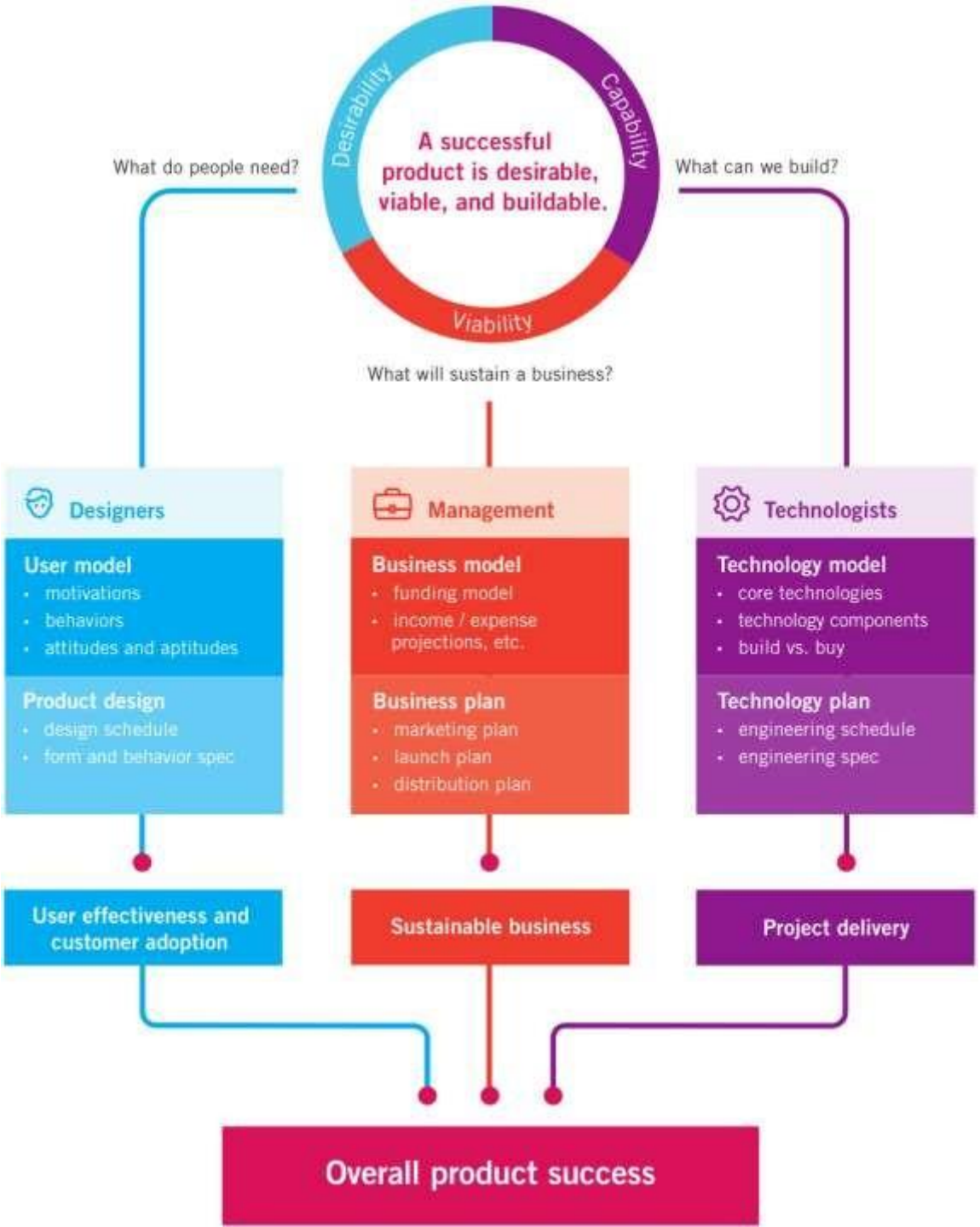
crearea de produse digitale
(recurgând la tehnologii Web)

scopuri psihologie comportament	interacțiune controale limbi naturale	funcționalități tehnologii algoritmi	indexare structurare meta-date	instrumente metodologii stimuli

utilizatori	interfață	software	conținut	creatori

Building successful digital products

actori principali:
designers
technologists
management



Alan Cooper *et al.*,
About Face (4th Edition), 2014

Care sunt mijloacele de interacțiune dintre utilizatori și o aplicație?

interaction

/ɪntərˈakʃ(ə)n/

noun

noun: **interaction**; plural noun: **interactions**

reciprocal action or influence.
"ongoing interaction between the two languages"

- PHYSICS
  a particular way in which matter, fields, and atomic and subatomic particles affect one another, e.g. through gravitation or electromagnetism.

Translate interaction to Romanian

noun

1. interacţiune
2. acţiune reciprocă
3. influenţă reciprocă

vezi și cursul *Human-Computer Interaction* de la master

profs.info.uaic.ro/~busaco/teach/courses/hci/

Interacțiunea dintre utilizator(i) și software se realizează via o **interfață** (*user interface*)

Interacțiunea dintre utilizator(i) și software se realizează via o **interfață** (*user interface*)

API (*Application Programming Interface*)
versus
UI (*User Interface*)

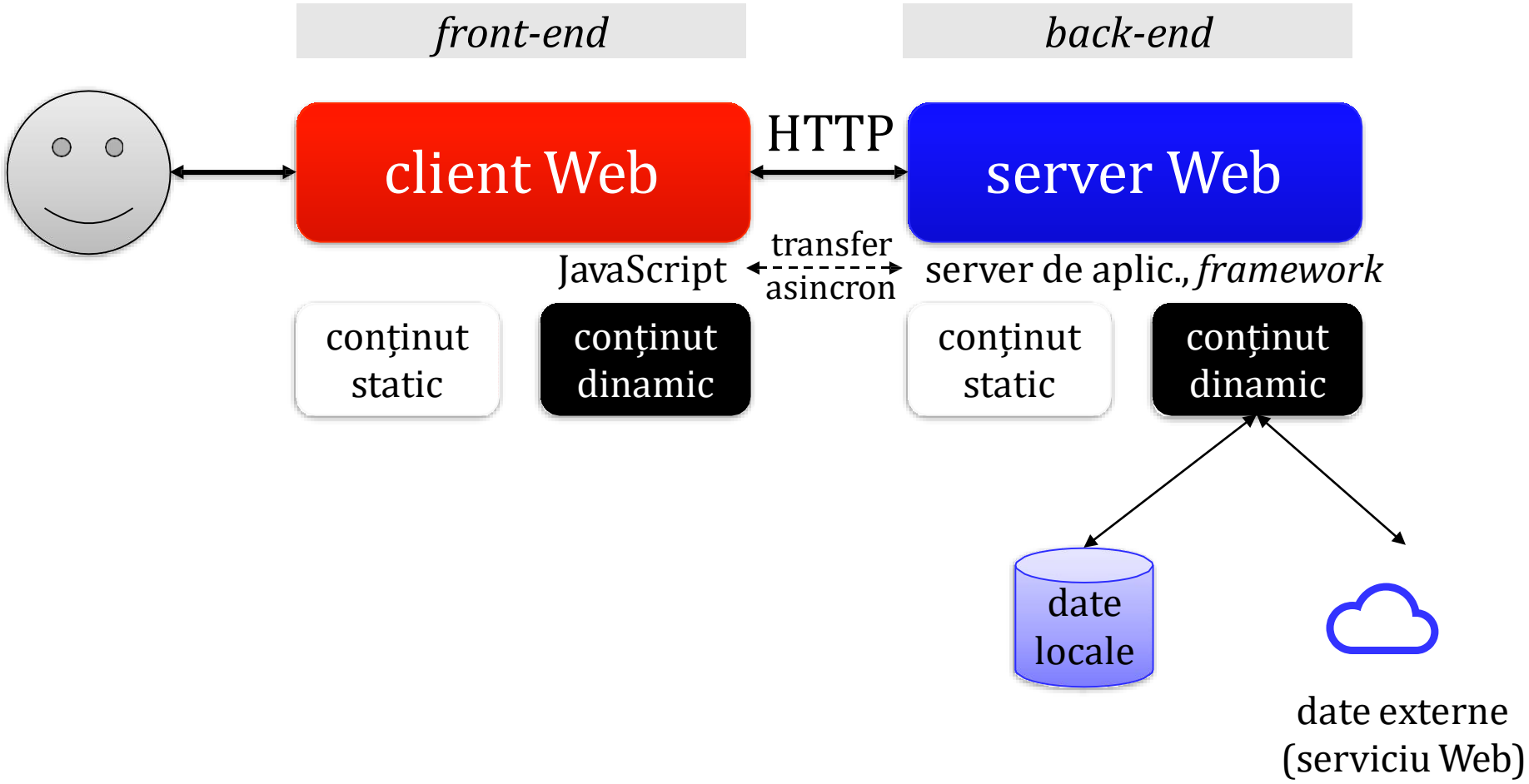
Aplicații Web

colecție interconectată de pagini Web
cu conținut generat dinamic,
oferind o funcționalitate specifică

Aplicații Web

prezintă o interfață cu utilizatorul exploatabilă
la nivel de client (*i.e.* navigator Web)

recurg la standarde/formate de date deschise
(HTTP, URL, HTML, CSS, JSON, SVG, XML, RDF,…)



via o interfață Web, utilizatorul interacționează cu clientul (*front-end*) și inițiază acțiuni – *e.g.*, cereri HTTP (a)sincrone – ce vor fi executate pe diverse componente implementate la nivel de server (*back-end*), pentru a obține date

Aplicații Web

interfața Web

browser – interacțiune limitată via controale HTML
hipertext/hipermedia
RIA (*Rich Internet Applications*): în prezent, HTML5
interacțiune facilitată de transfer (a)sincron: Ajax *et al.*
audiență globală

Așteptările utilizatorilor referitoare la interfața Web (Peter Morville)

utilă – *useful*
utilizabilă – *usable*
apreciată – *valuable*
dezirabilă – *desirable*
disponibilă – *findable*
accesibilă – *accessible*
credibilă – *credible*

Interfața – *desktop*, Web,… – cu utilizatorul

parte a aplicației – *desktop*, Web, miniaturală,… – care permite utilizatorilor să-și exprime intențiile de operare asupra software-ului și să interpreteze rezultatele acțiunilor efectuate de mașină

Interfața – *desktop*, Web,... – cu utilizatorul

percepută nu doar ca parte vizuală a software-ului

din punctul de vedere al utilizatorului,
reprezintă întregul sistem – aplicația *per se*

Interfața – *desktop*, Web,... – cu utilizatorul

utilitate (*utility*)

oferirea facilităților dorite de utilizator

Interfața – *desktop*, Web,... – cu utilizatorul

utilizabilitate (*usability*)

cât de ușor și de plăcut pot fi folosite facilitățile?

Interfața – *desktop*, Web,... – cu utilizatorul

utilă (*useful*)

usability + *utility*

UX (***User Experience***)

modul de percepție a produsului/serviciului
de către persoanele care-l folosesc
și plăcerea/satisfacția înregistrată

Care sunt arhitecturile software tipice pe baza cărora sunt dezvoltate aplicațiile Web de anvergură?

Calitatea aplicațiilor Web este influențată de arhitectura pe care se bazează

arhitecturi: principii

start with needs

do less

design with data

do the hard work to make it simple

iterate. then iterate again

build for inclusion

understand context

build digital services, not Websites

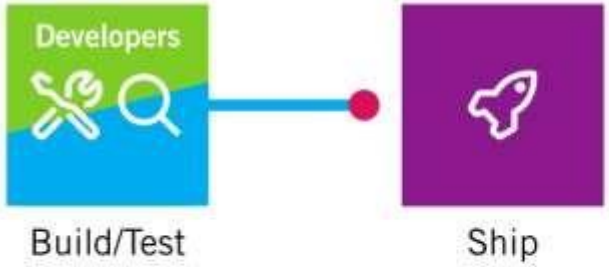
be consistent, not uniform

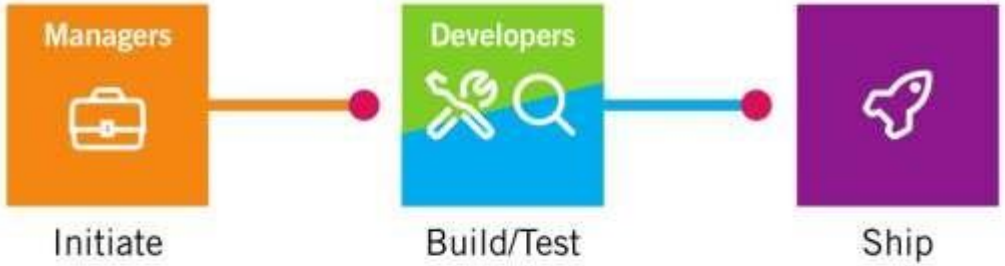
make things open; it makes things better

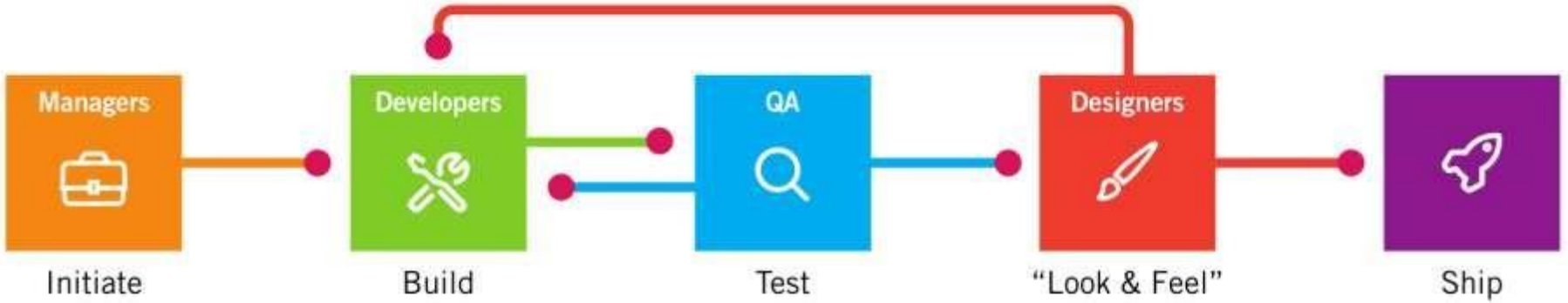
exemplu pentru gov.uk – Paul Downey & David Heath (2013)

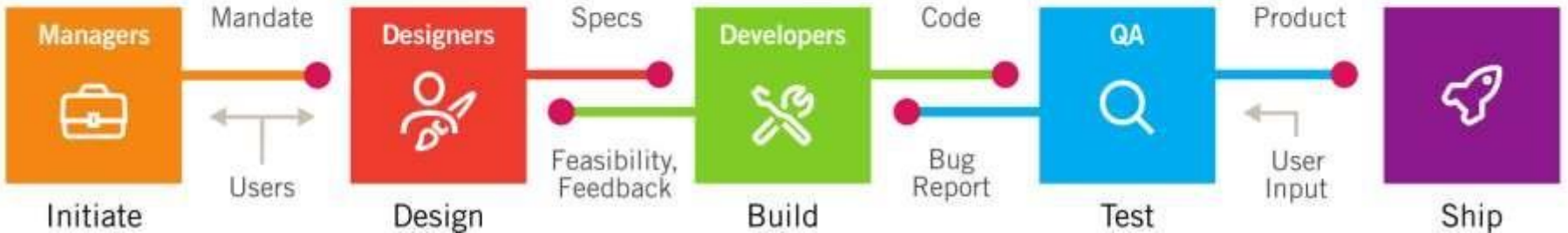
evoluția manierei de dezvoltare a produselor digitale (software)

Alan Cooper *et al.*, 2014









Complexitatea aplicațiilor actuale este mai mare decât a produselor tangibile (fizice)

"*If your UI even vaguely resembles an airplane cockpit, you're doing it wrong.*"
John Gruber

Dezvoltarea unei arhitecturi software ia în calcul:

cerințe funcționale

impuse de clienți,
vizitatori,
concurență,
factori decizionali (management),
evoluție socială/tehnologică,
...

Atragerea experților
– *subject matter expert* (SME) sau *domain expert* –
în domeniul problemei
ce trebuie soluționată de aplicația Web

Dezvoltarea unei arhitecturi software ia în calcul:

factori calitativi

utilizabilitate
performanță
securitate
refolosire a datelor/codului
etc.

Dezvoltarea unei arhitecturi software ia în calcul:

aspecte tehn(olog)ice

platforma hardware/software (sistem de operare)
infrastructura *middleware*
servicii disponibile – *e.g.*, via API-uri publice
limbaj(e) de programare
sisteme tradiționale (*legacy*)
…

Inițial:
oferirea funcționalităților esențiale – *less is more*

Inițial:
oferirea funcționalităților esențiale – *less is more*

Versiuni ulterioare:
extinderea aplicației Web
– uzual, via o interfață de programare (API) publică,
încurajând dezvoltarea de soluții propuse de utilizatori

Dezvoltarea unei arhitecturi software ia în calcul:

experiența

recurgerea la arhitecturi și platforme existente
șabloane de proiectare (*design patterns*)
soluții „la cheie”: biblioteci, *framework*-uri, instrumente,…
management de proiecte
etc.

Metodologii moderne – exemple:

aim42 – practici și șabloane privind evoluția, mentenanța, migrarea și îmbunătățirea sistemelor software
aim42.github.io

12 Factor App – vizând aplicațiile aliniate paradigmei SaaS (*Software As A Service*)
12factor.net

12 Factor CLI Apps – aplicații în linia de comandă (J. Dickey, 2018)
medium.com/@jdxcode/12-factor-cli-apps-dd3c227a0e46

client(i)

mandatar (*proxy*)

zid de protecție (*firewall*)

intermediar(i) (*middleware*)

server(e) Web

server(e) de aplicații Web

cadre de lucru, biblioteci, alte componente

server(e) de stocare persistentă – *e.g.*, baze de date

server(e) de conținut multimedia

server(e) de management al conținutului – *e.g.*, CMS, *wiki*

aplicații/sisteme tradiționale (*legacy*)

„ingrediente” tipice

client(i)

mandatar (*proxy*)

zid de protecție (*firewall*)

intermediar(i) (*middleware*)

server(e) Web

server(e) de aplicații Web

cadre de lucru, biblioteci, alte componente

server(e) de stocare persistentă – *e.g.*, baze de date

server(e) de conținut multimedia

server(e) de management al conținutului – *e.g.*, CMS, *wiki*

aplicații/sisteme tradiționale (*legacy*)

eventual, recurgând la servicii în „nori” – ***cloud computing***

partajarea la cerere a resurselor de calcul și a datelor cu alte calculatoare/dispozitive pe baza tehnologiilor Internet (găzduire, infrastructură scalabilă, procesare paralelă, monitorizare,…)

Esențialmente, de considerat:

preluarea și dirijarea cererilor – *dispatch*

oferirea funcționalităților de bază – *core services*

asocierea dintre construcții/abstracțiuni software (*e.g.*, obiecte) și modele de date – *mapping*

managementul datelor – *data*

monitorizarea și evaluarea sistemului – *metrics*

adaptare după Matt Ranney, "*What I Wish I Had Known Before Scaling Uber to 1000 Services*", GOTO Chicago 2016

highscalability.com/blog/2016/10/12/lessons-learned-from-scaling-uber-to-2000-engineers-1000-ser.html

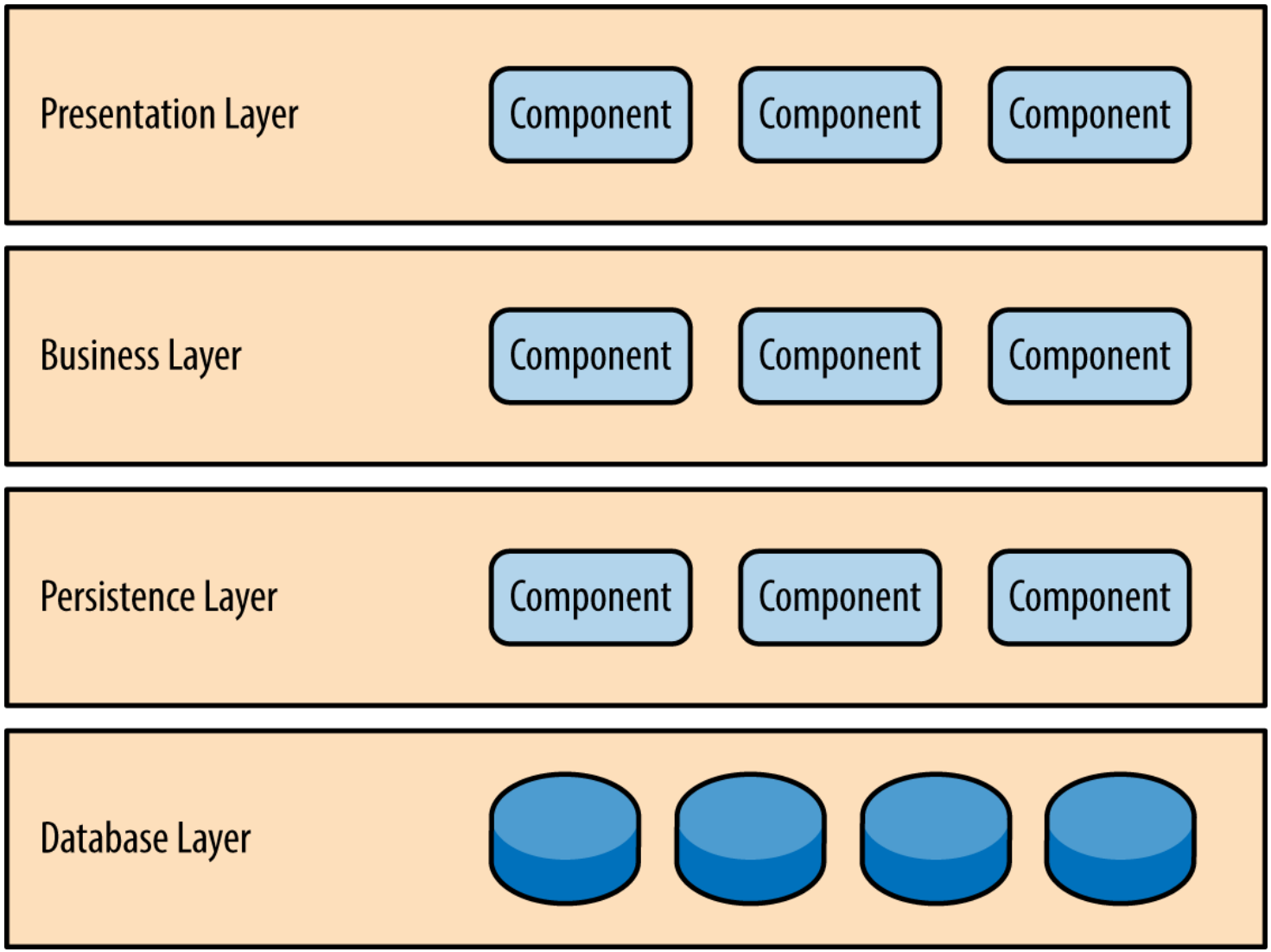
arhitecturi web

Stratificate (*layered*)
Conduse de evenimente (*event-driven*)
Extensibile (*microkernel / plug-in*)
Folosind microservicii (*microservices*)
„În nori” (*space-based*, *cloud*)

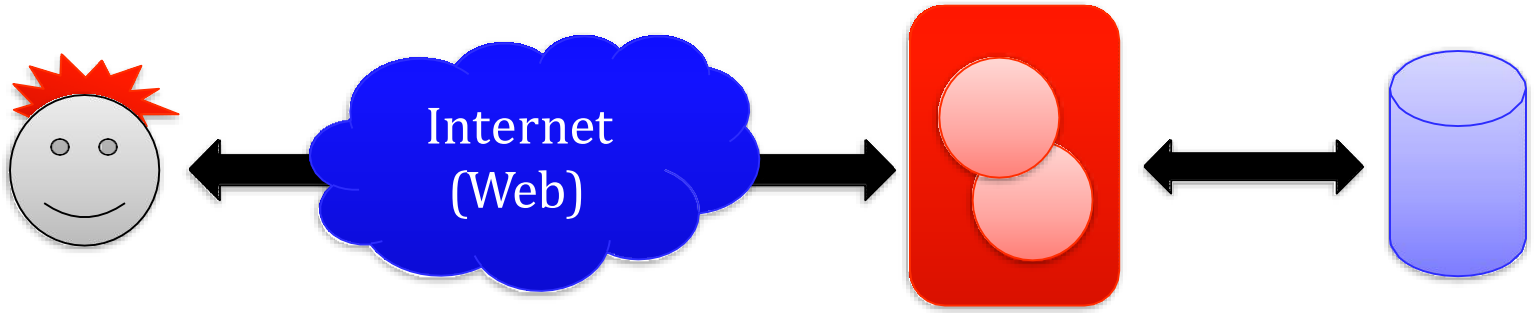
conform M. Richards, *Software Architecture Patterns*, O’Reilly, 2015
www.oreilly.com/programming/free/files/software-architecture-patterns.pdf

Stratificate (*layered*)

N-tier architecture – abordare *de facto*



aplicație Web = **interfață** + **program** + **conținut** (date)
trei strate (*3-tier application*)



Client
(*interface*)

Server de aplicații
(*application*)

Stocare
(*persistence*)

C. Henderson, *Scalable Web Architectures*, Web 2.0 Expo, 2007: iamcal.com/talks/

Stratificate (*layered*): principii

demarcarea responsabilităților (*separation of concerns*)
fiecare strat are un rol bine-stabilit, componentele unui strat vizând funcționalitățile acestuia

modelul de structurare a datelor este separat de maniera de procesare (controlul aplicației, *business logic*) și de modul de prezentare a acestora (interfața Web)

Stratificate (*layered*): principii

izolare (*layers of isolation*)

modificările operate la un anumit strat nu au impact sau nu afectează componentele din alt strat

Stratificate (*layered*): principii

architecture sinkhole anti-pattern

fluxul de cereri traversează fiecare strat, fără a se efectua procesări semnificative în cadrul acestuia

arhitecturi web

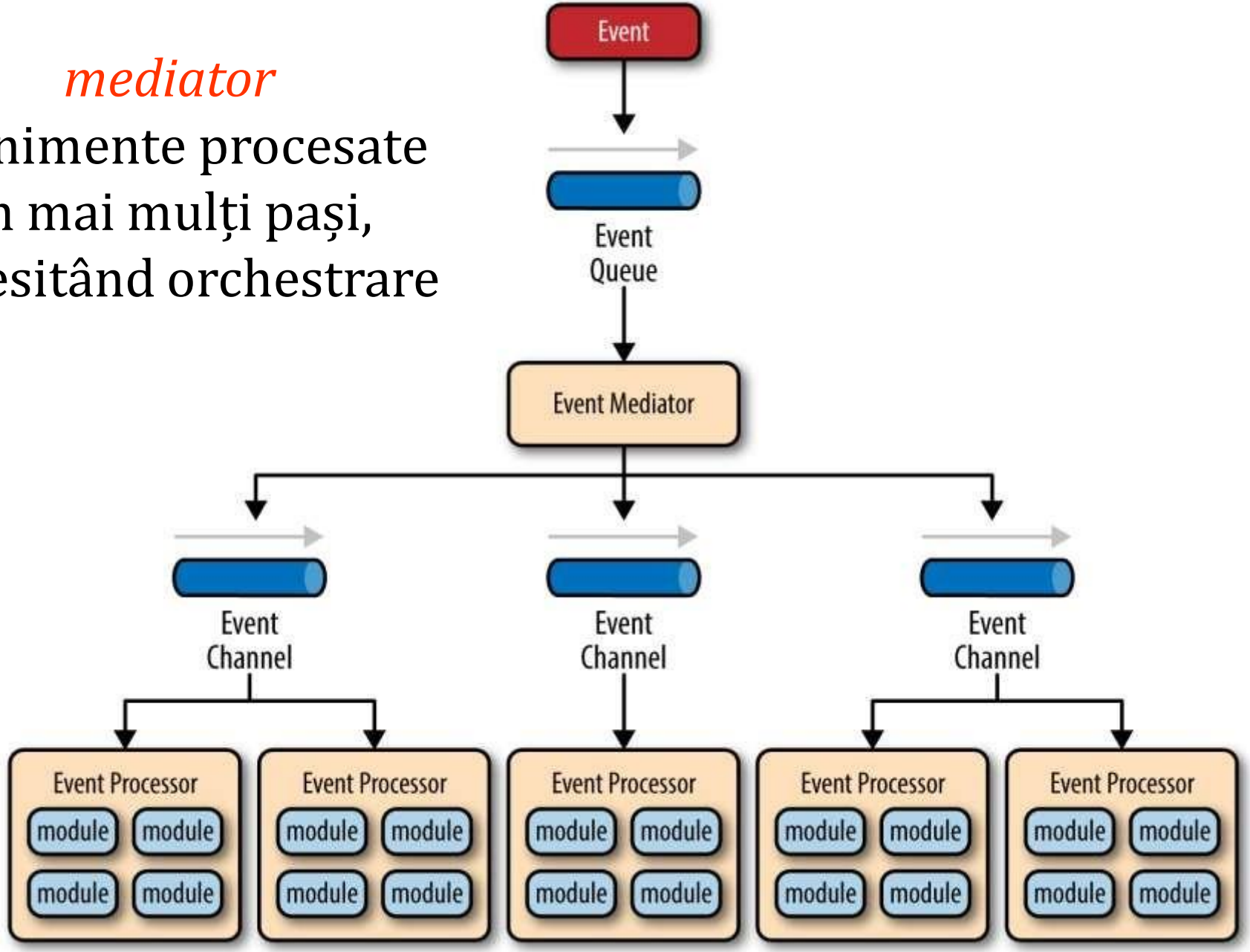
Conduse de evenimente (*event-driven*)
uzual în contextul aplicațiilor distribuite asincrone
- scalabilitate

topologii principale:
mediator
sau
broker

Conduse de evenimente (*event-driven*)

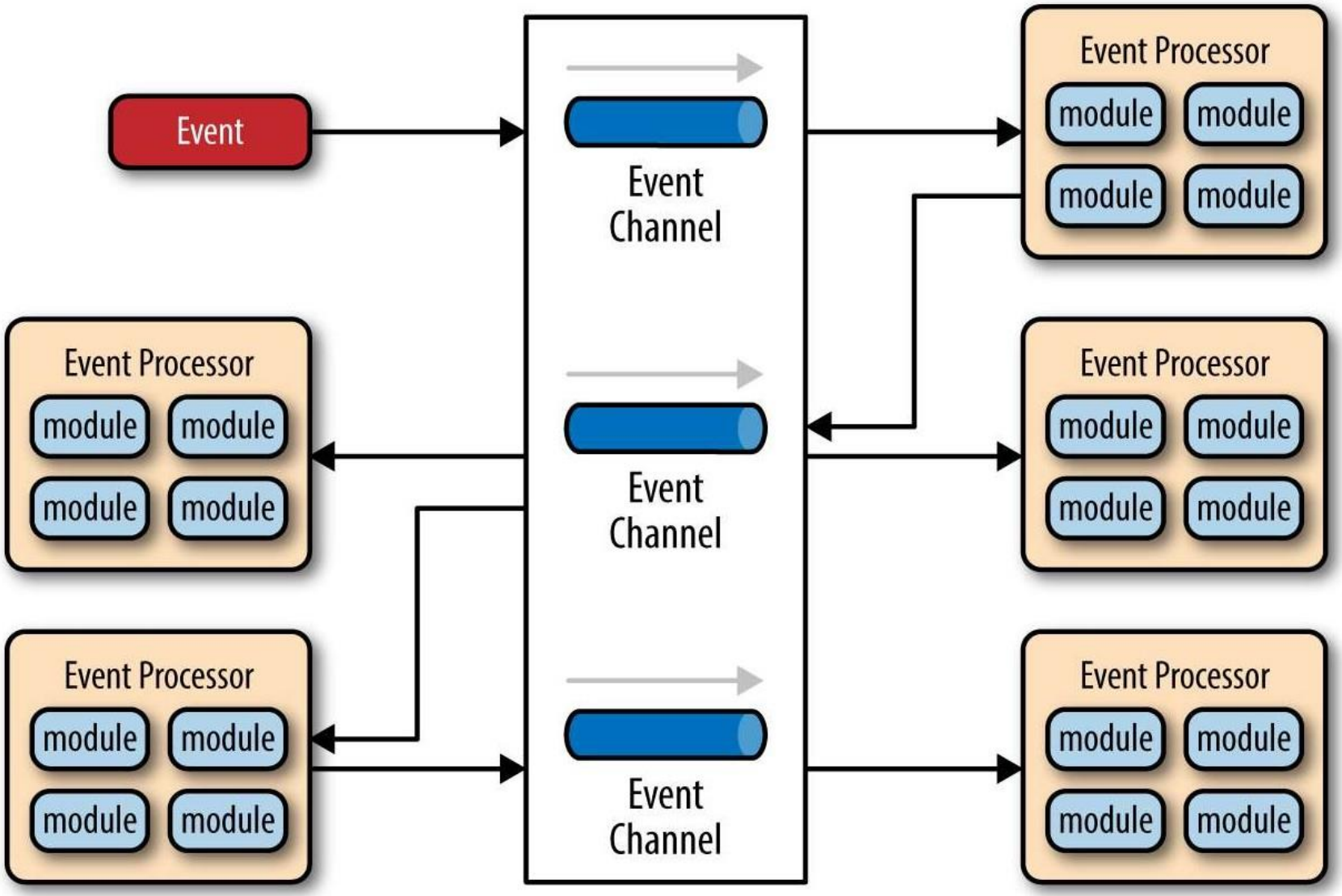
mediator

evenimente procesate
în mai mulți pași,
necesitând orchestrare



Conduse de evenimente (*event-driven*)

broker – fluxul de mesaje este distribuit componentelor de procesare a evenimentelor

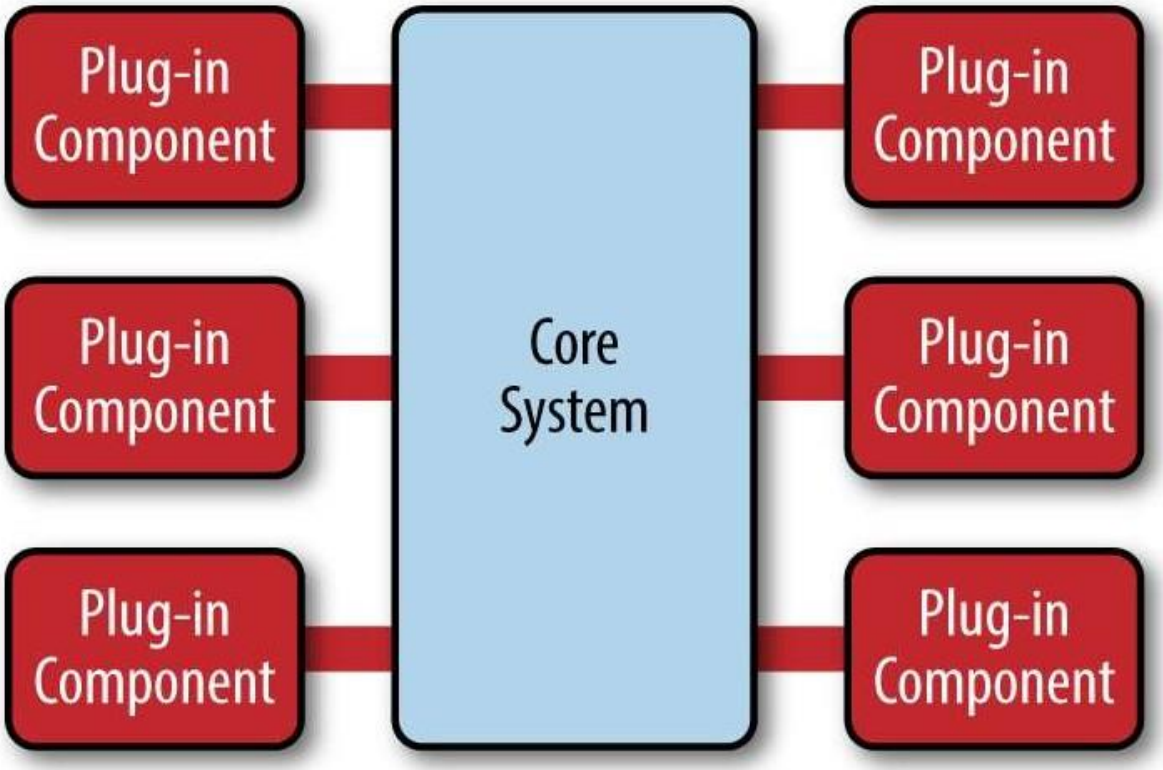


Extensibile (*microkernel / plug-in*)

sistem principal (*core system*)

+

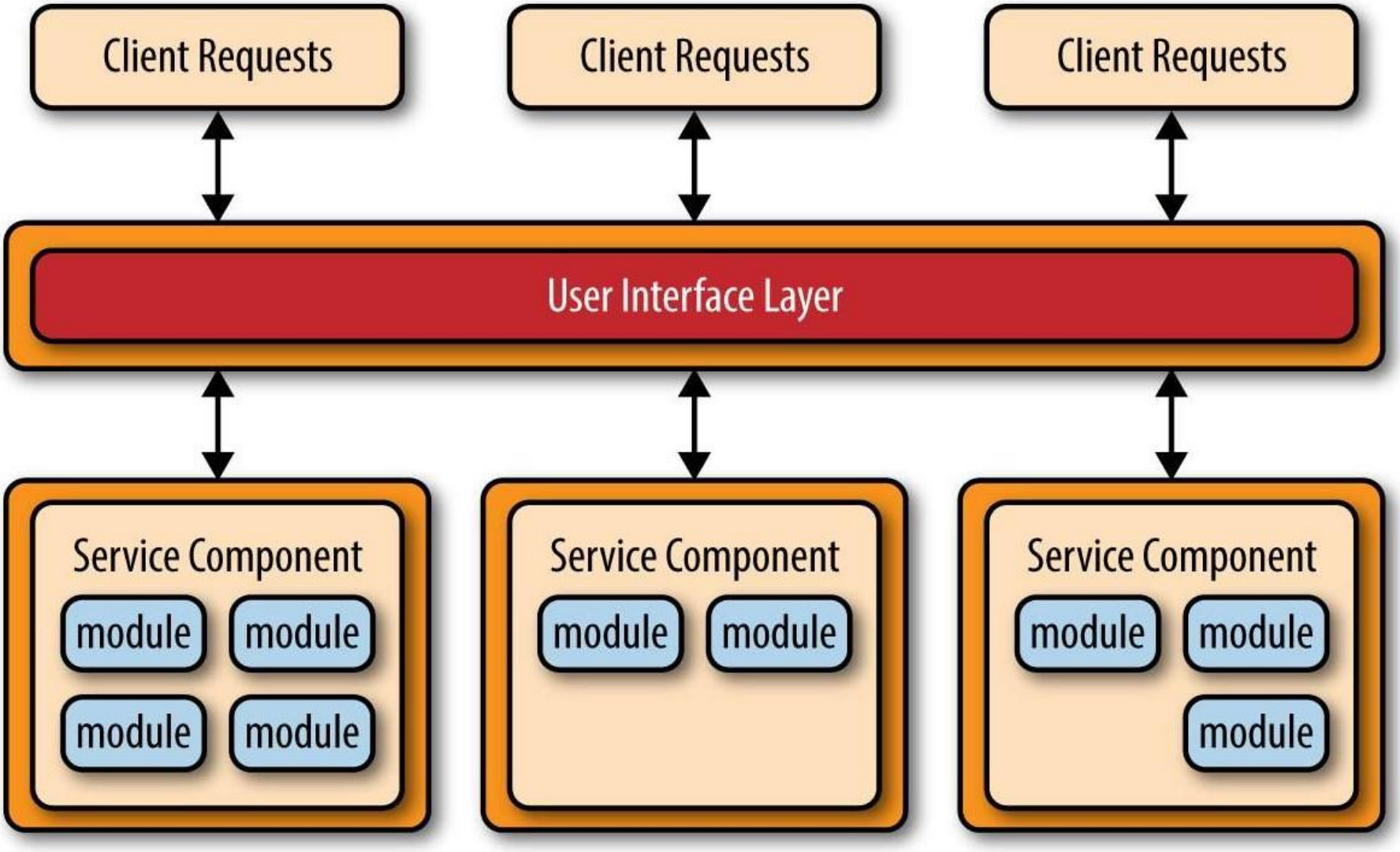
module independente de tip extensie (*plug-in*)



o astfel de arhitectură poate fi inclusă/utilizată ca parte a altei abordări arhitecturale

Folosind microservicii (*microservices*)

componente separate, distribuite
(*separately deployed units*) ▸ decuplare maximă



arhitecturi web

Folosind micro-servicii (*microservices*)

abordări:

bazate pe API-uri (*API-based*)

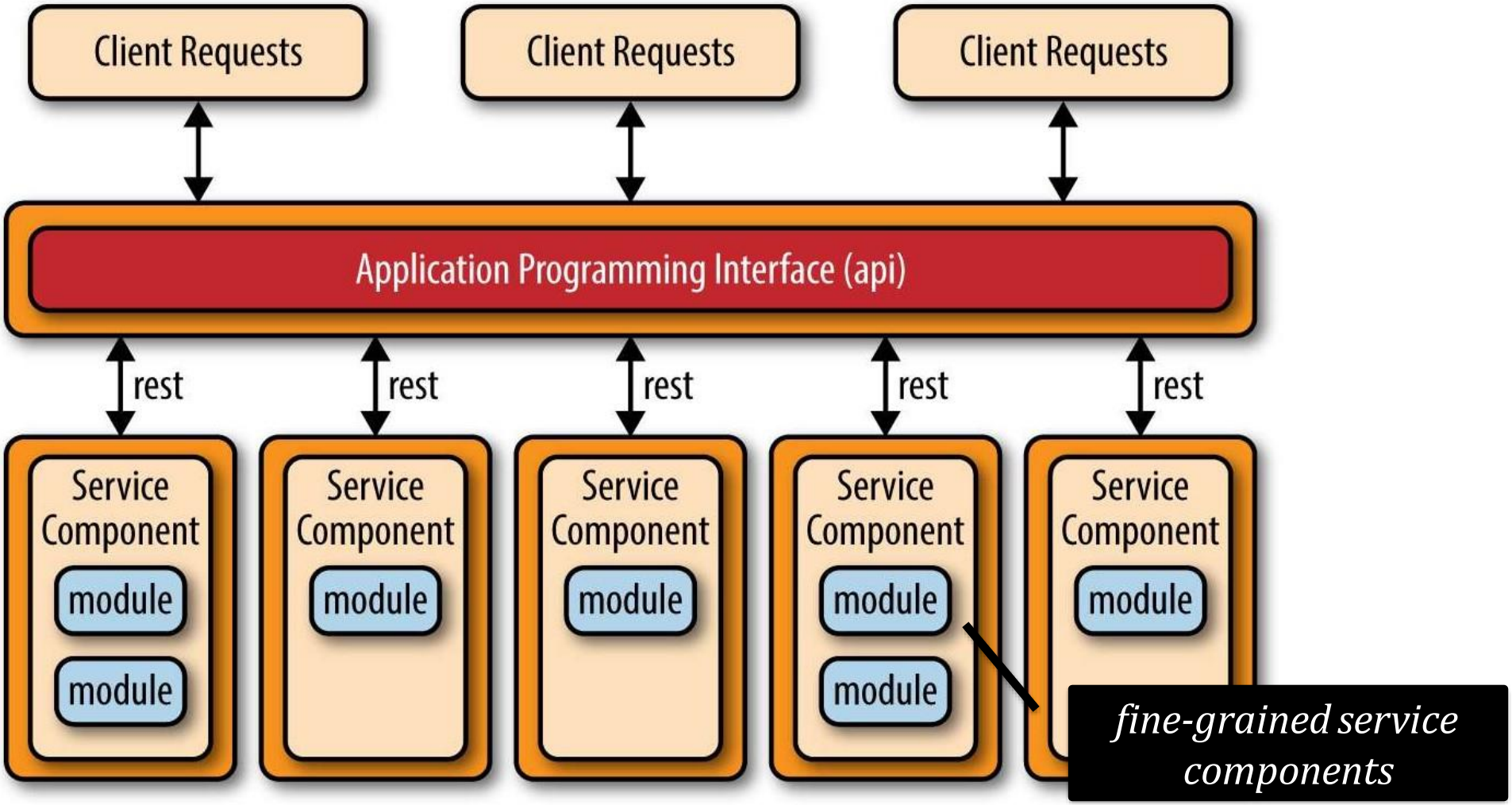
aplicație recurgând la REST (*application REST-based*)

mesagerie centralizată (*centralized messaging*)

Folosind microservicii (*microservices*)

API-based

aplicația Web expune servicii individuale, punctuale, de sine-stătătoare (*self-contained*) via un API

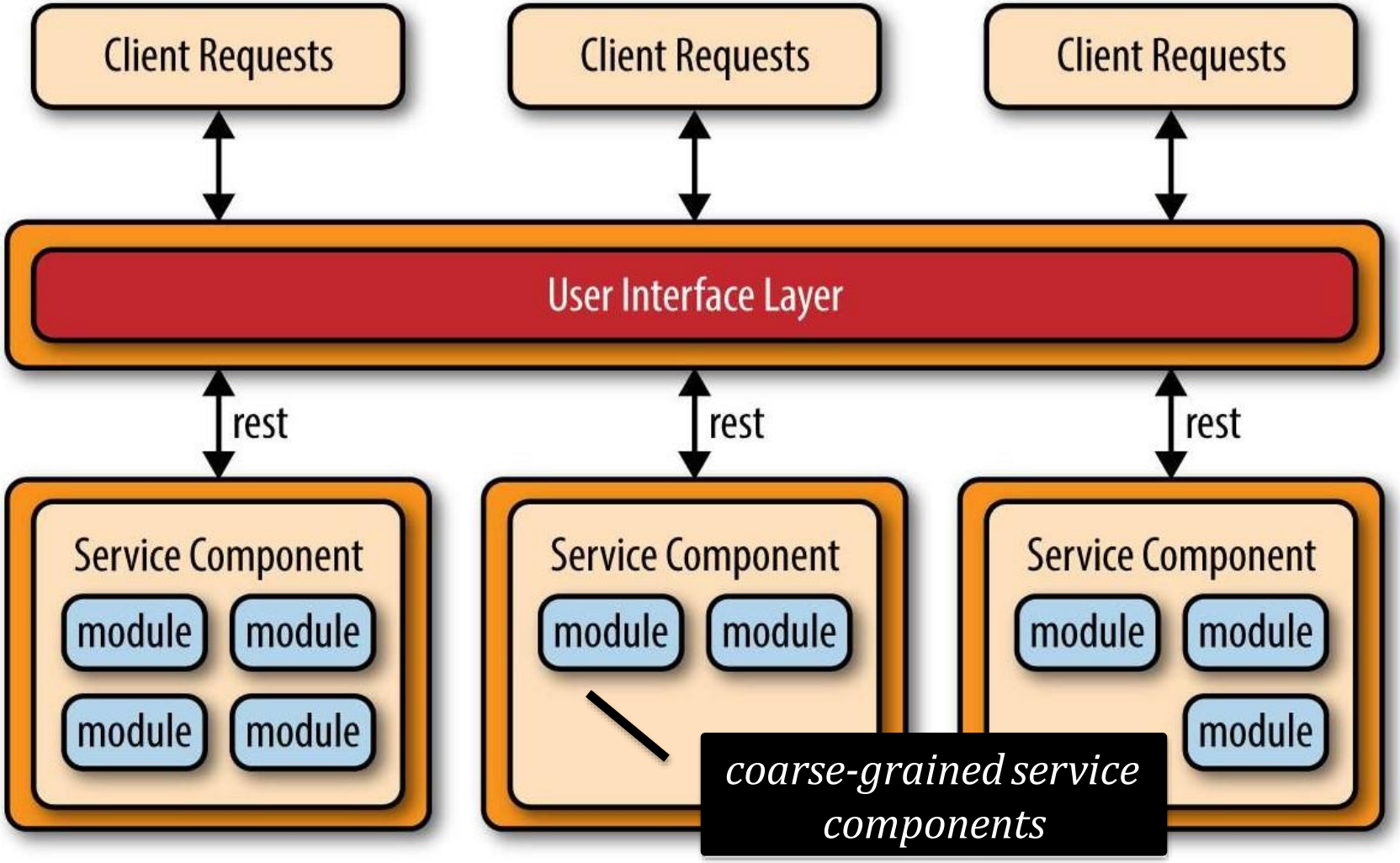


Folosind microservicii (*microservices*)

application REST-based

cererile sunt recepționate tradițional (nu prin API)

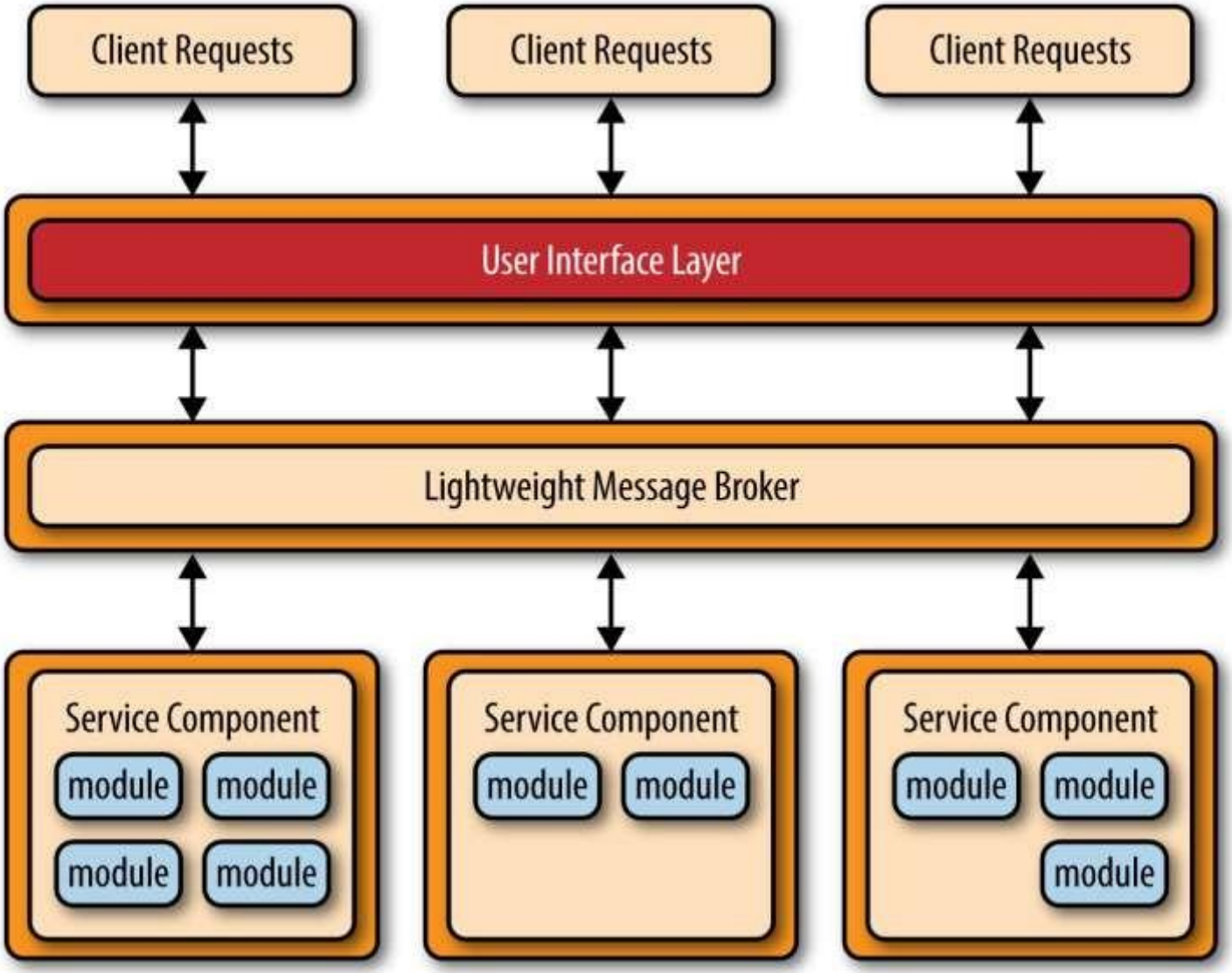
fiecare funcționalitate este accesată intern via REST



Folosind microservicii (*microservices*)

centralized messaging

accesare a componentelor interne via un *broker* „ușor”



arhitecturi web

„În nori” (*space-based*, *cloud*)
consideră și rezolvă problemele vizând scalabilitatea și concurența unui volum impredictibil de mare de cereri

arhitecturi web

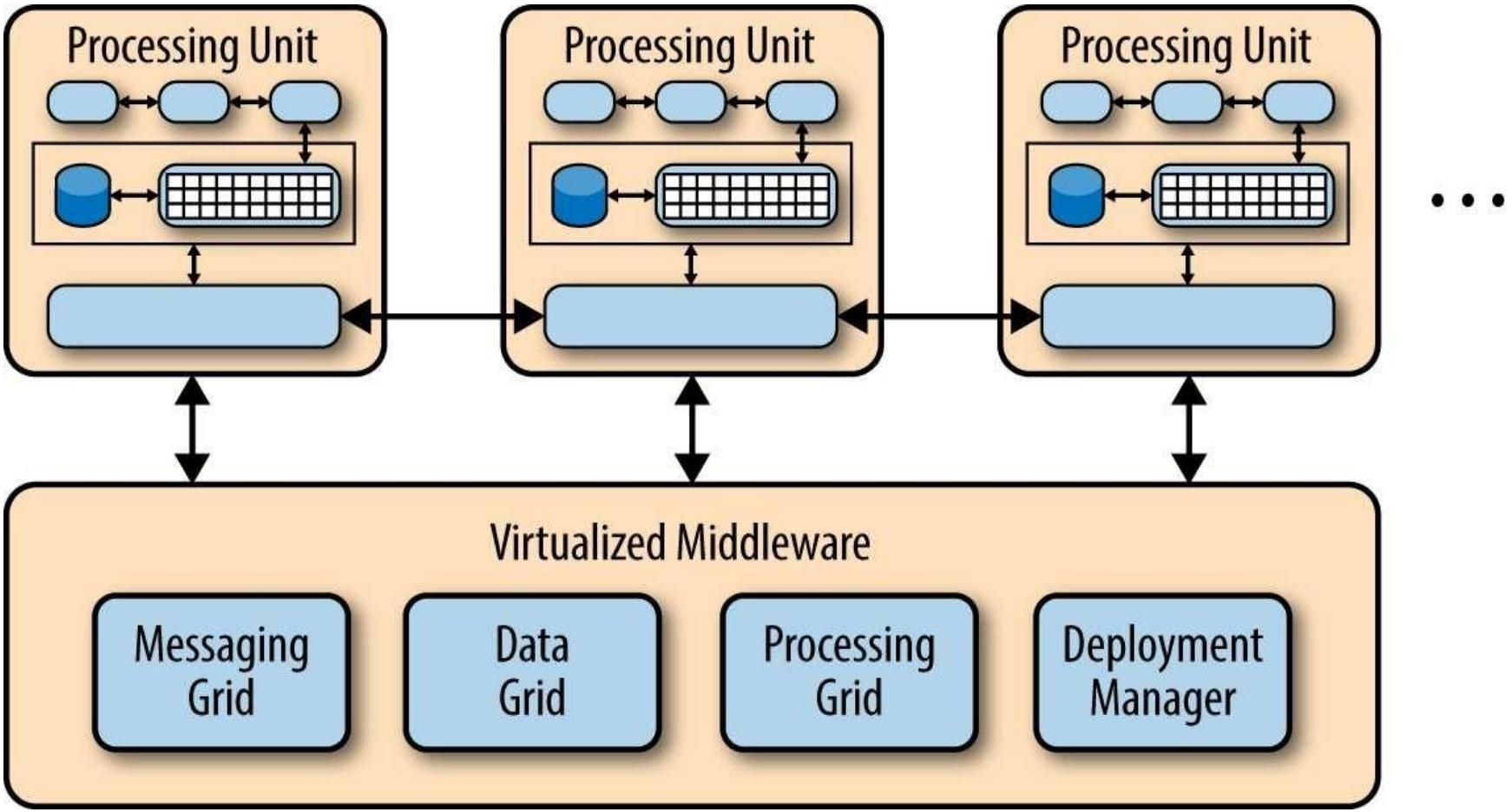
„În nori” (*space-based*, *cloud*)
consideră și rezolvă problemele vizând scalabilitatea și concurența unui volum impredictibil de mare de cereri

tuple space
datele aplicației sunt păstrate în memorie și replicate de toate unitățile de procesare active

fără stocare centralizată ▸ *distributed shared memory*

wiki.c2.com/?TupleSpace

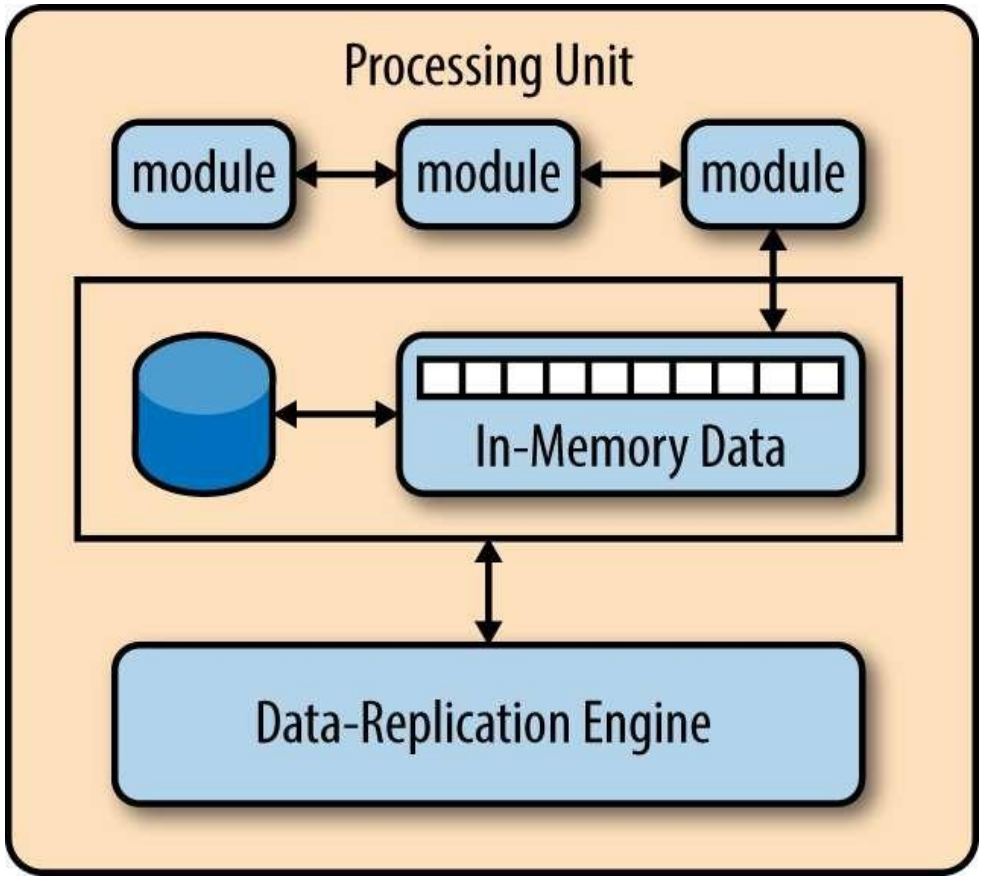
„În nori” (*space-based*, *cloud*)



middleware virtualizat

include componente controlând sincronizarea datelor,
procesarea cererilor,
accesul la platforma de execuție (*deployment*),...

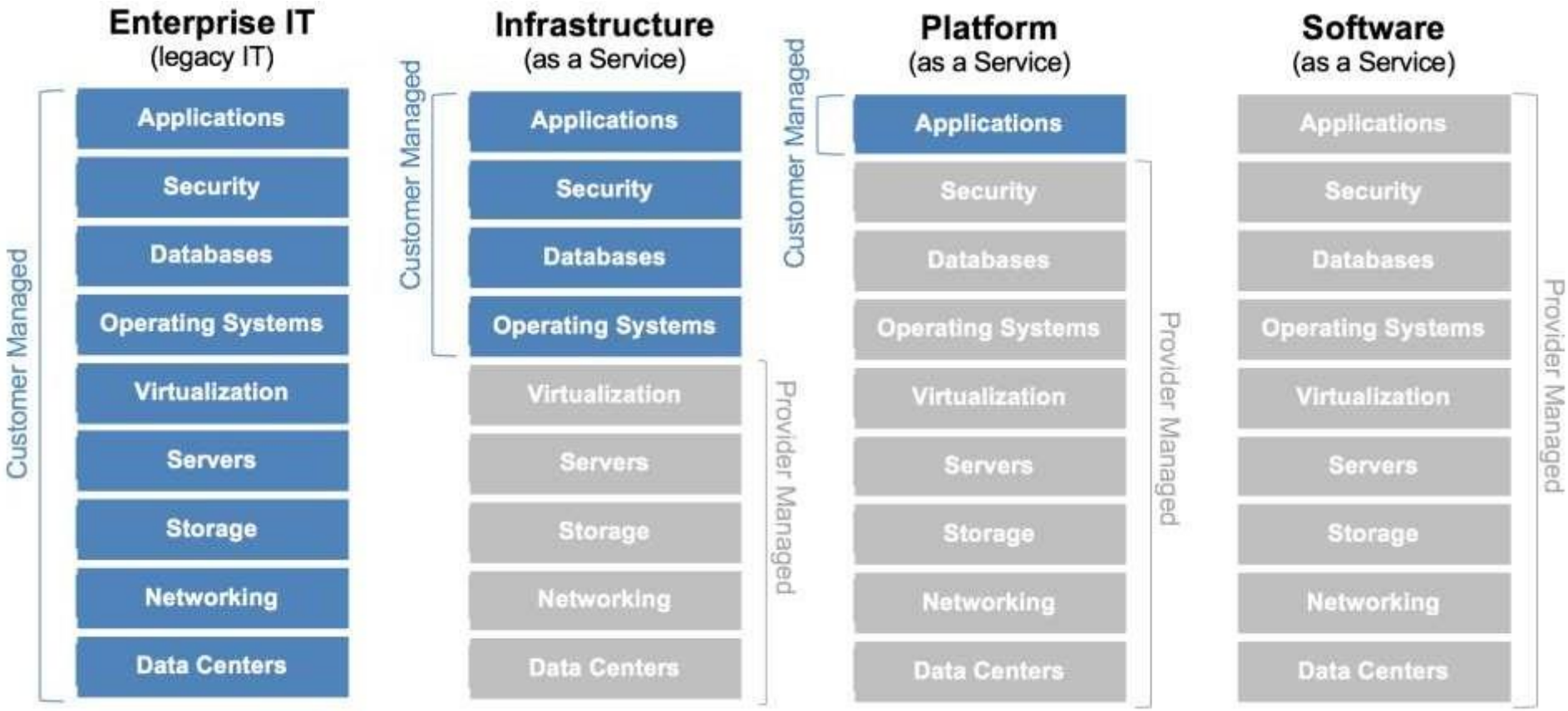
„În nori” (*space-based*, *cloud*)



unitate de procesare

reprezentată de un (micro-)serviciu Web

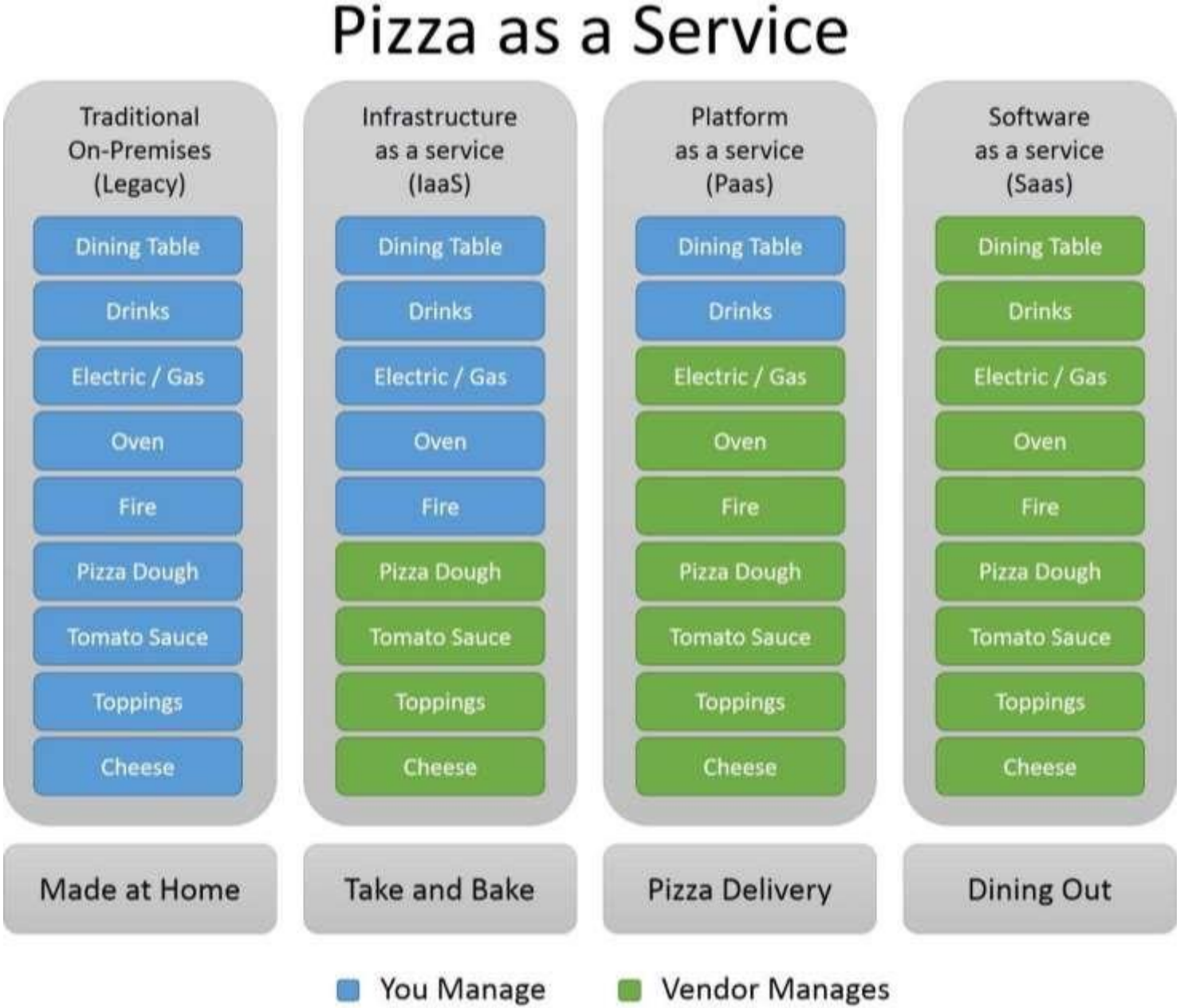
sau o componentă software tradițională la nivel de *backend*



anumite funcționalități pot fi gestionate
„în propria ogradă” (*on-premises*)
sau de un furnizor de servicii disponibile „în nori”

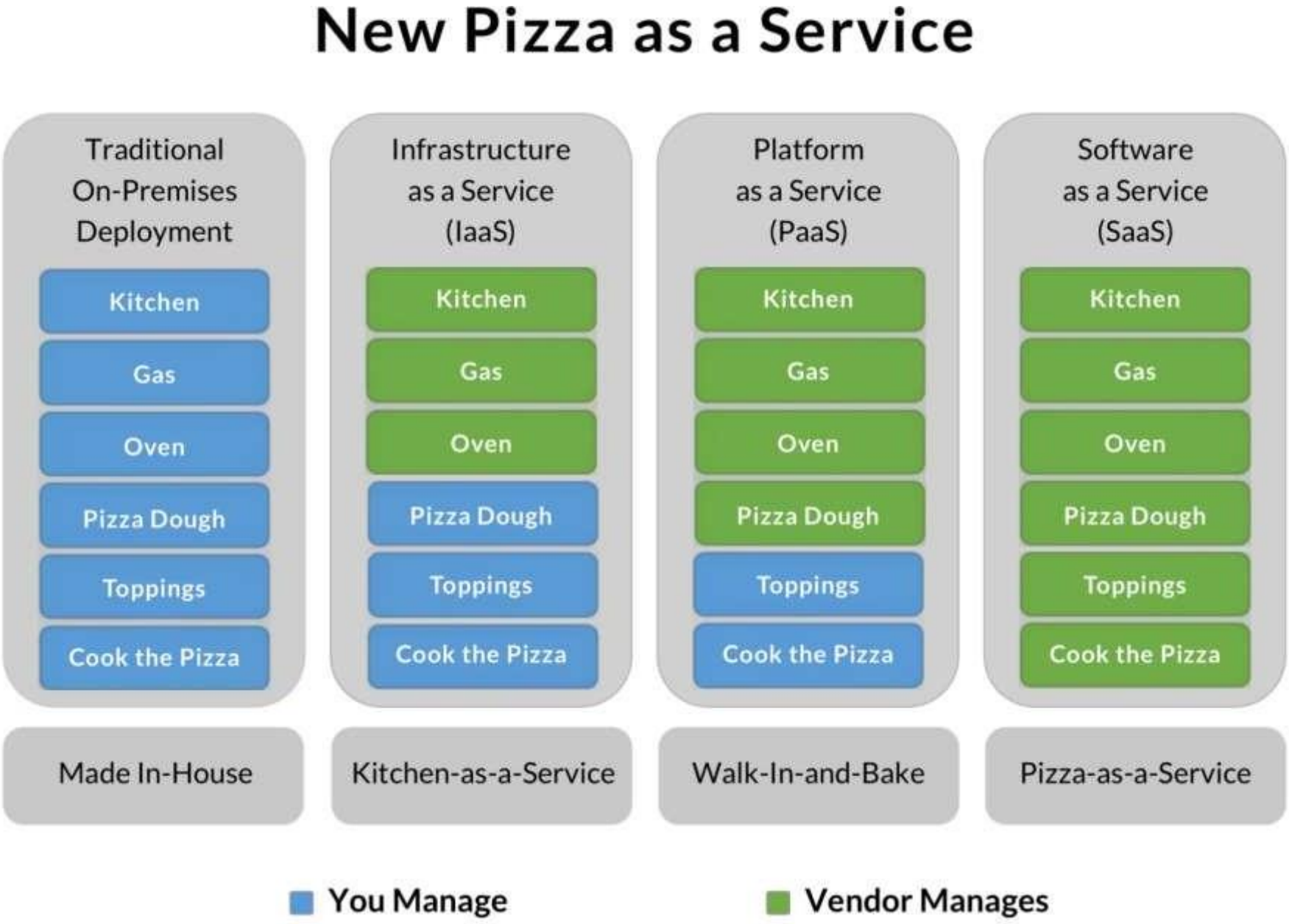
conform (Eizadirad, 2017)

www.linkedin.com/pulse/iaas-paas-saas-explained-compared-arsalan-eizadirad



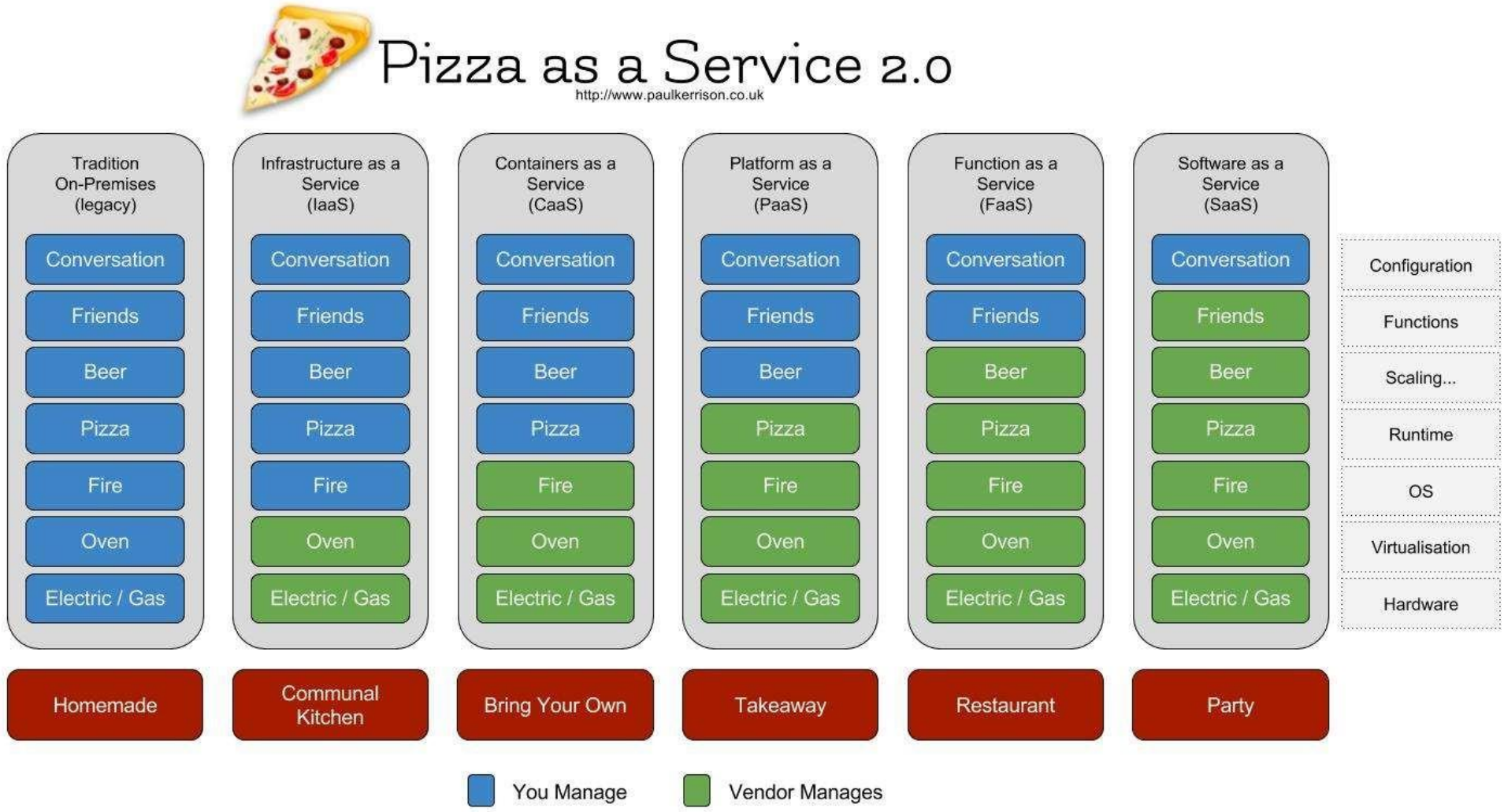
A. Barron, *Pizza As A Service* (2014)

www.linkedin.com/pulse/20140730172610-9679881-pizza-as-a-service



D. Ng, *SaaS, PaaS and IaaS explained in one graphic* (2017)

m.oursky.com/saas-paas-and-iaas-explained-in-one-graphic-d56c3e6f4606



P. Kerrison, *Pizza As A Service 2.0* (2017)

www.paulkerrison.co.uk/random/pizza-as-a-service-2-0

Prin ce mijloace poate fi implementată o aplicație Web?

Server de aplicații Web

scop:
eficientizarea proceselor de dezvoltare
a aplicațiilor Web de anvergură

Server de aplicații Web

simplifică invocarea de programe (*script*-uri)

- generarea de conținut dinamic pe partea de server

(re)vezi prezentarea despre inginerie Web:
profs.info.uaic.ro/~busaco/teach/courses/web/web-film.html#week3

Server de aplicații Web

poate fi integrat în unul/mai multe servere Web

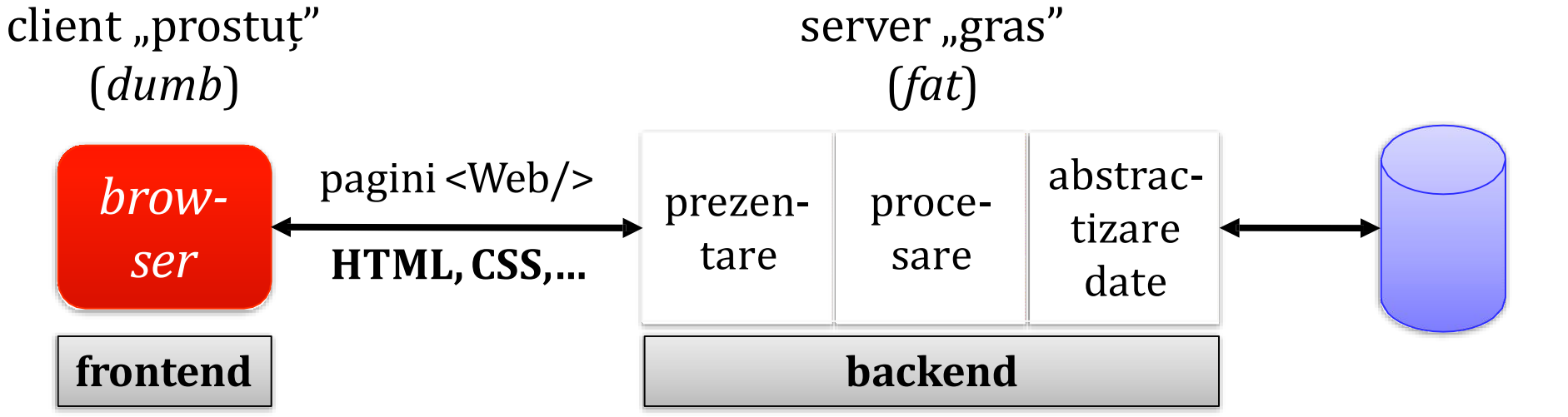
de asemenea, poate oferi propriul server Web
sau mediu de execuție

Server de aplicații Web

poate încuraja sau impune o viziune arhitecturală privind dezvoltarea de aplicații Web

situație tipică:
MVC ori variații (Herberto Graca, 2017)
herbertograca.com/2017/08/17/mvc-and-its-variants/

arhitectura aplicațiilor Web: **abordarea MV* tradițională**



frecvent, **aplicație monolitică**
(*e.g.*, un WAR: 2.2 M linii de cod, 418 .jar-uri, startare în 12 min. – conform plainoldobjects.com)

www.leaseweblabs.com/2013/10/api-first-architecture-fat-vs-thin-server-debate/

Framework (**cadru de lucru**)

facilitează dezvoltarea de aplicații Web complexe, simplificând unele operații uzuale (*e.g.*, acces la baze de date, *caching*, generare de cod, management de sesiuni, control al accesului) și/sau încurajând reutilizarea codului-sursă

Framework-uri JavaScript

la nivel de server (*back-end*) – specifice Node.js

Express

Hapi

Koa

LoopBack

Meteor

Next.js

...

Framework-uri JavaScript

la nivel de client (*front-end*) – rulând în *browser*

Angular

Aurelia

Backbone

Ember

MithrilJS

Vue.js

...

Bibliotecă Web (*library*)

colecție de resurse computaționale reutilizabile
– *i.e.*, structuri de date + cod –
oferind funcționalități (comportamente) specifice
implementate într-un limbaj de programare

Bibliotecă Web (*library*)

colecție de resurse computaționale reutilizabile
– *i.e.*, structuri de date + cod –
oferind funcționalități (comportamente) specifice
implementate într-un limbaj de programare

poate fi referită de alt cod-sursă (software):
server de aplicații, *framework*, bibliotecă,
serviciu, API ori componentă Web

Biblioteci JS cu acces liber la codul-sursă

exemple:

CesiumJS

D3.js

Leaflet

Lodash

PDF.js

Raphaël

React

Sgvizler

TensorFlow.js

Three.js

...

Serviciu Web

software – utilizat la distanță de alte aplicații/servicii – oferind o funcționalitate specifică

- SOA (*Service Oriented Architecture*)

Serviciu Web

software – utilizat la distanță de alte aplicații/servicii – oferind o funcționalitate specifică

- SOA (*Service Oriented Architecture*)

implementarea sa nu trebuie cunoscută
de programatorul ce invocă serviciul

Micro-serviciu

implementează o funcționalitate specifică,
oferită la nivel de unic proces

self-contained system

componentă la nivel de *backend* dezvoltată cu scopul
de a fi înlocuită, nu de a fi reutilizată

small

each running in its own process

lightweight communication mechanisms (usual, HTTP)

built around business capabilities

independently deployable

minimum of centralized management

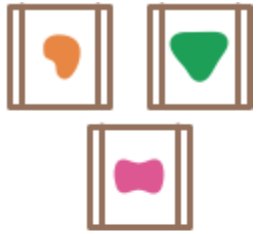
may be written in different programming languages

may use different data storage mechanisms

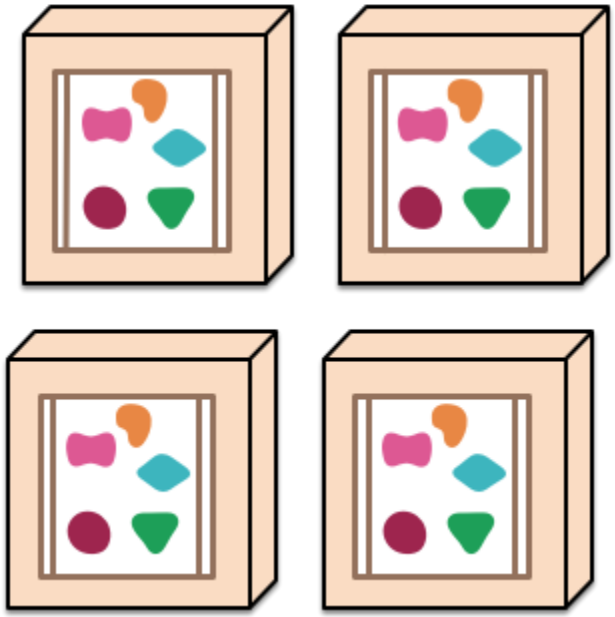
caracteristici ale micro-serviciilor conform
James Lewis & Martin Fowler, *Microservices* (2014)
martinfowler.com/articles/microservices.html

A monolithic application puts all its functionality into a single process...

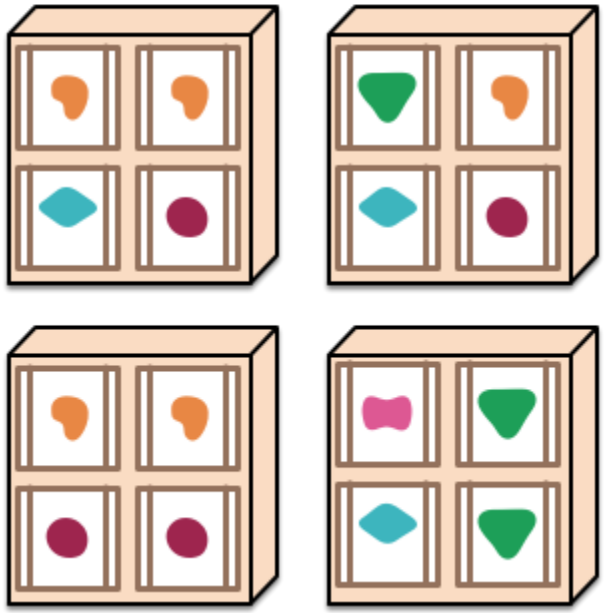
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers



... and scales by distributing these services across servers, replicating as needed.



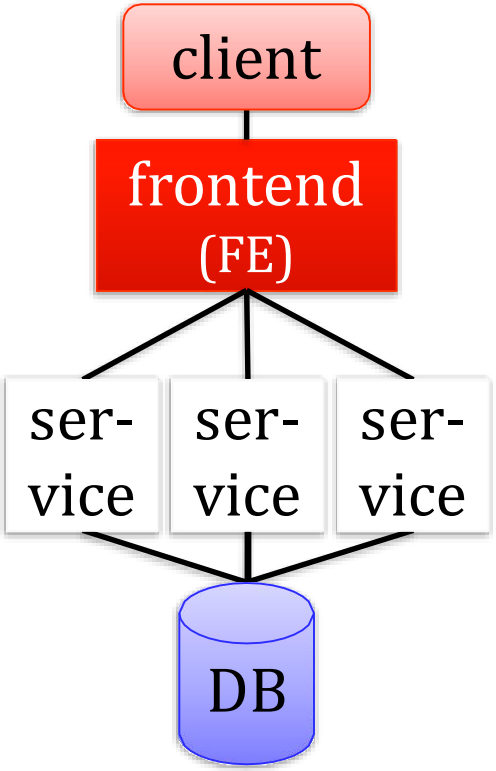
micro-serviciu

modularitate, descentralizare și evoluție permanentă

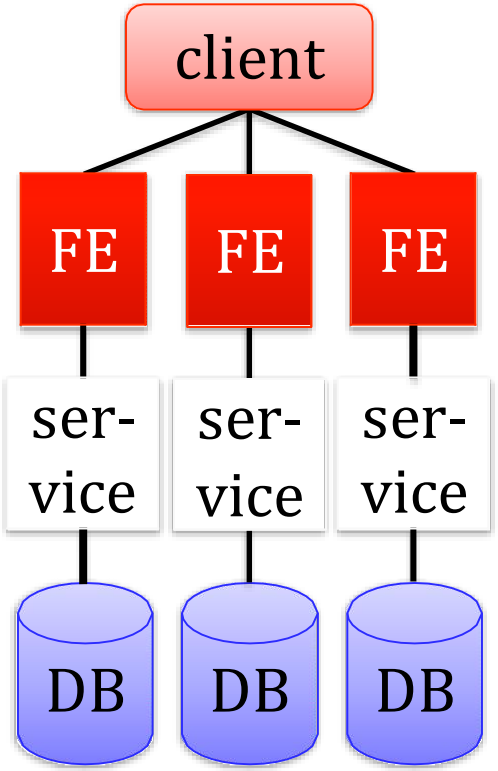
exemple de bună practică: **microservices.io**

Uzual, arhitecturile ce recurg la micro-servicii
nu includ componente *middleware*
și nu oferă suport pentru abstractizarea interacțiunii
dintre producătorii și consumatorii de servicii
(*contract decoupling*)

- **µSOA** – *Microservice Oriented Architecture*



arhitectură bazată
pe servicii Web



arhitectură recurgând
la microservicii

Z. Dehghani, *How to break a Monolith into Microservices* (2018)
martinfowler.com/articles/break-monolith-into-microservices.html

cazuri concrete: Amazon, Groupon, Netflix,...
de studiat prezentările lui Stefan Tilkov: **speakerdeck.com/stilkov**

API (***Application Programming Interface***)

accesul la un (micro-)serviciu are loc uzual
pe baza unei interfețe de programare a aplicației – API

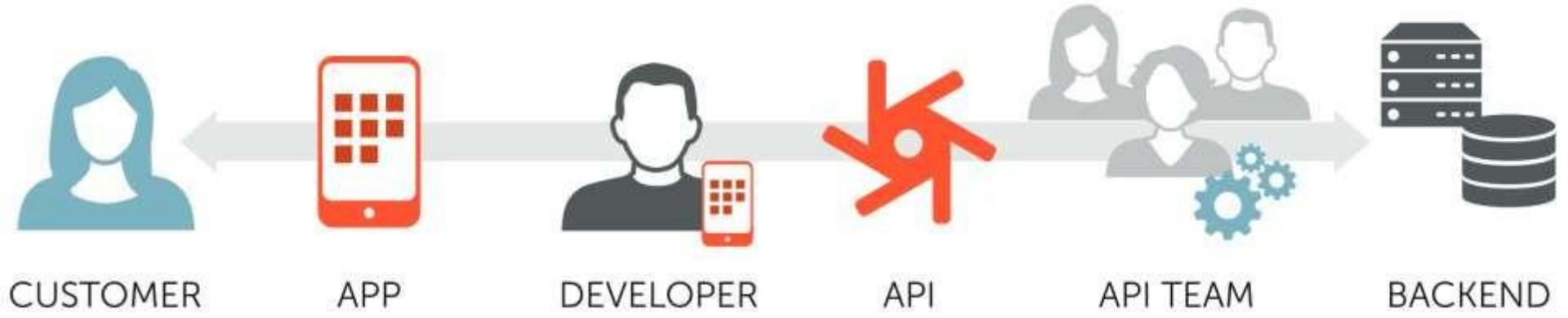
API (*Application Programming Interface*)

"any well-defined interface that defines the service that one component, module, or application provides to other software elements"
(de Souza *et al.*, 2004)

Componentă software concepută și invocată via tehnologiile Web actuale (URI, HTTP, formate de date: JSON, XML)

poate fi dezvoltată conform unui stil arhitectural *e.g.*, REST (*REpresentational State Transfer*)

De la aplicații la API-uri și servere de aplicații



Brian Mulloy, *Web API Design*, Apigee, 2016

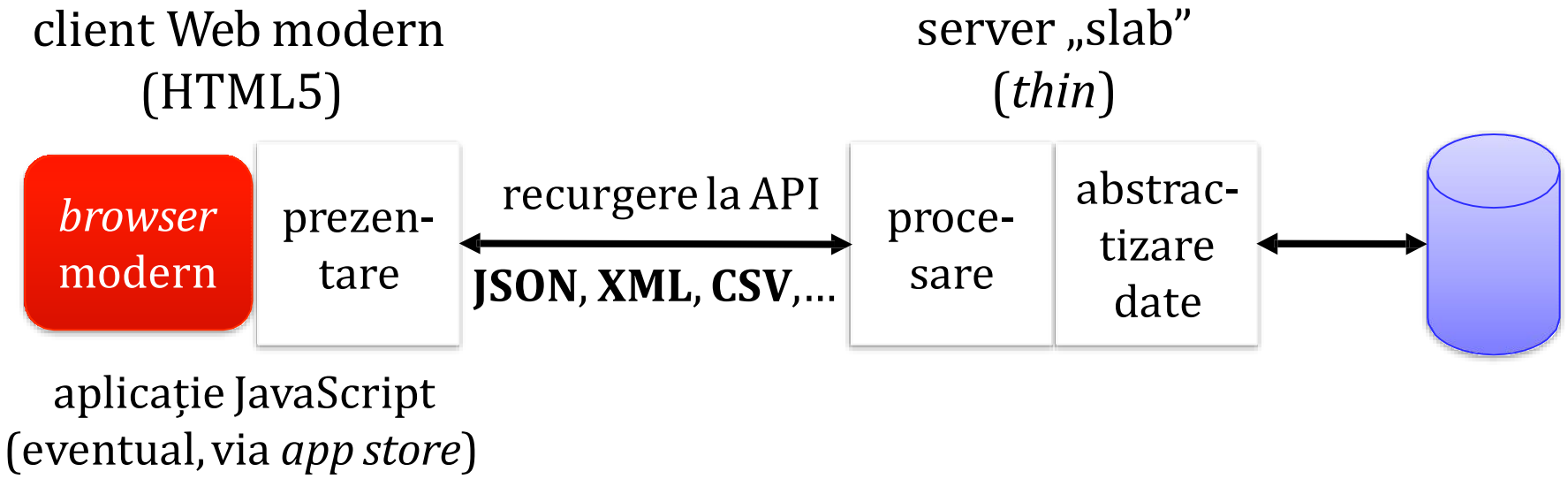
docs-apis.apigee.io/files/Web-design-the-missing-link-ebook-2016-11.pdf

API public
(disponibil pe baza unei licențe de utilizare)

versus

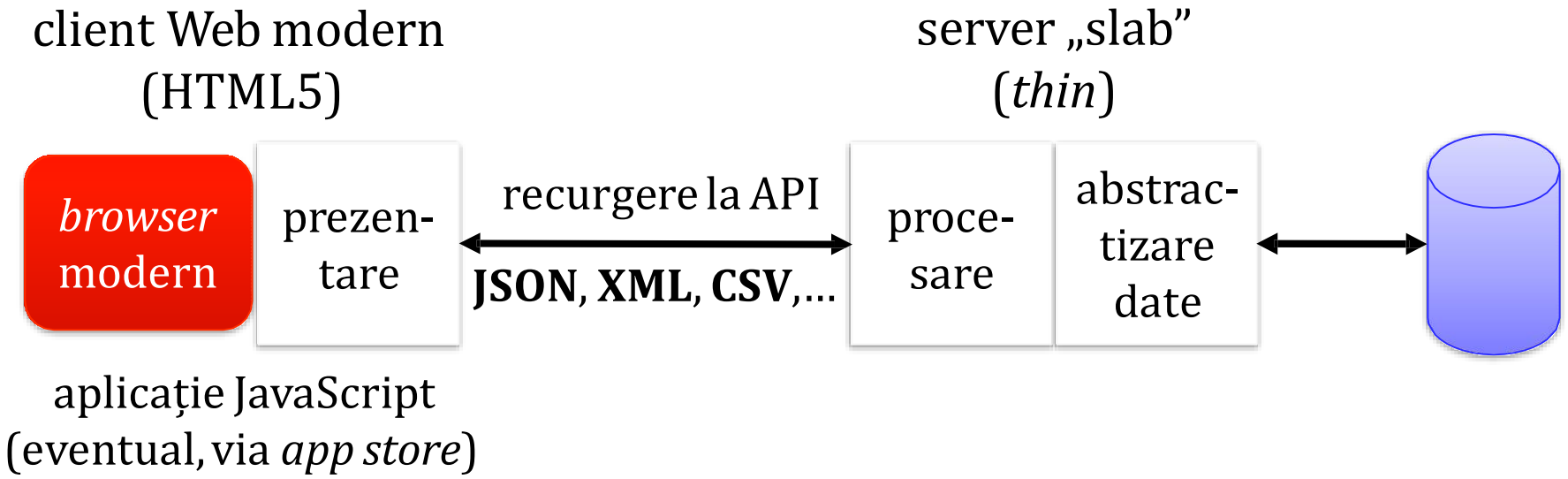
API privat
(pentru uz intern)

API: abordare client – JavaScript



browser Web pe calculatoare convenționale, dispozitive mobile și altele

API: abordare client – JavaScript

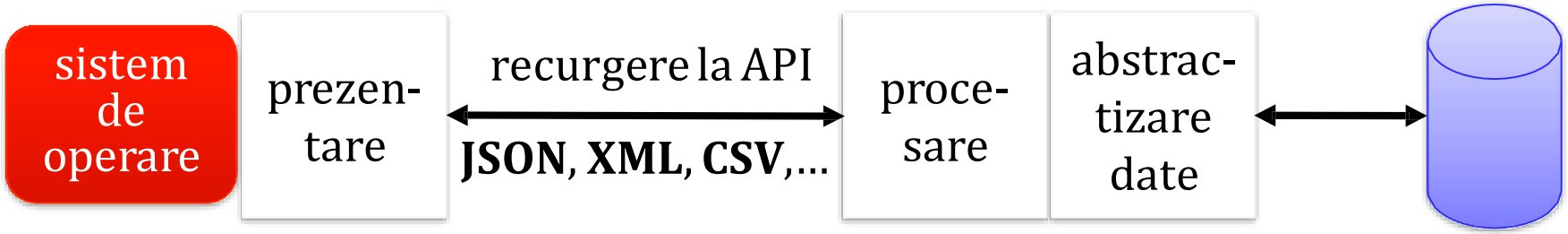


implementarea aplicației JavaScript poate recurge la biblioteci, *framework*-uri, componente specifice
e.g., Angular, React, Vue

API: aplicații native

client nativ modern
(*smart device*)

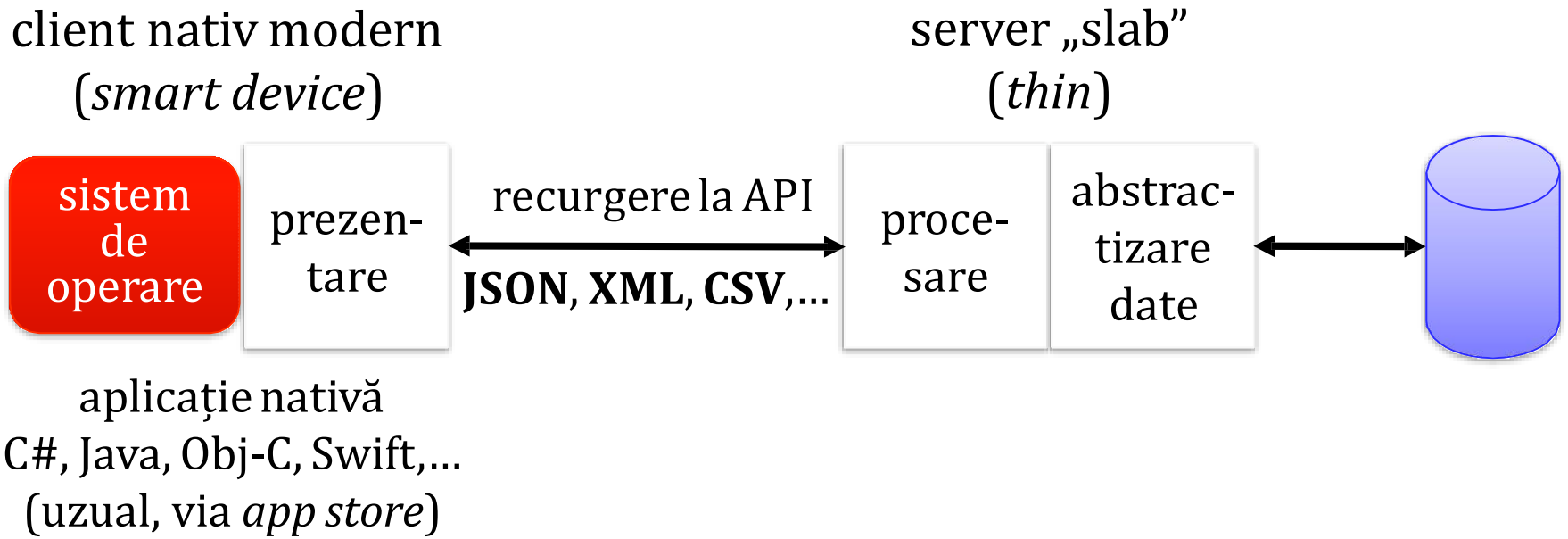
server „slab”
(*thin*)



aplicație nativă
C#, Java, Obj-C, Swift,...
(uzual, via *app store*)

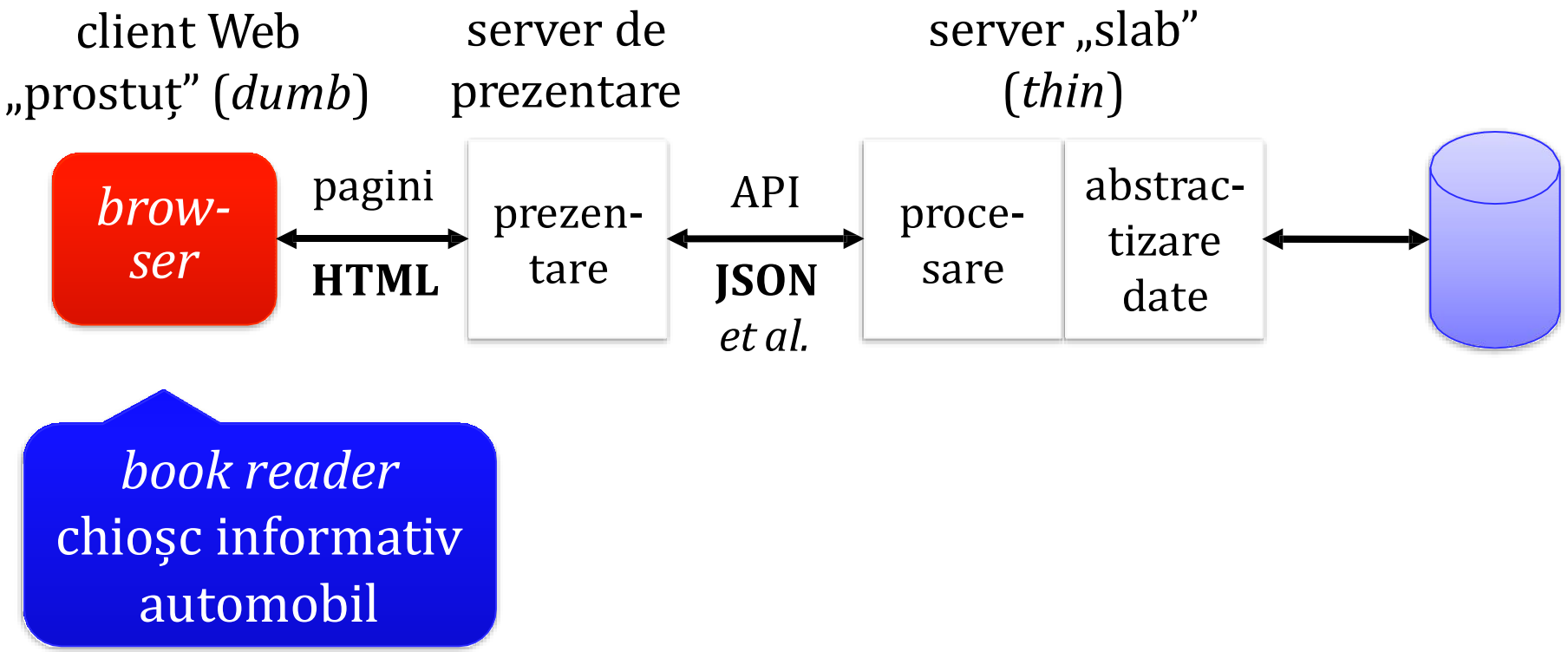
desktop și/sau mobile,
smart TV, *home appliance*,
dispozitiv ambiental

API: aplicații native



implementarea aplicației native poate recurge la biblioteci, *framework*-uri, componente specifice
e.g., Apache Cordova, Flutter, Ionic, React Native, NW.js

API: abordare bazată pe intermediari



www.leaseweblabs.com/2013/10/api-first-architecture-fat-vs-thin-server-debate/

API: în contextul *serverless*

Aplicația depinde semnificativ de componente externe, disponibile în „nori”

(micro-)servicii expuse via API

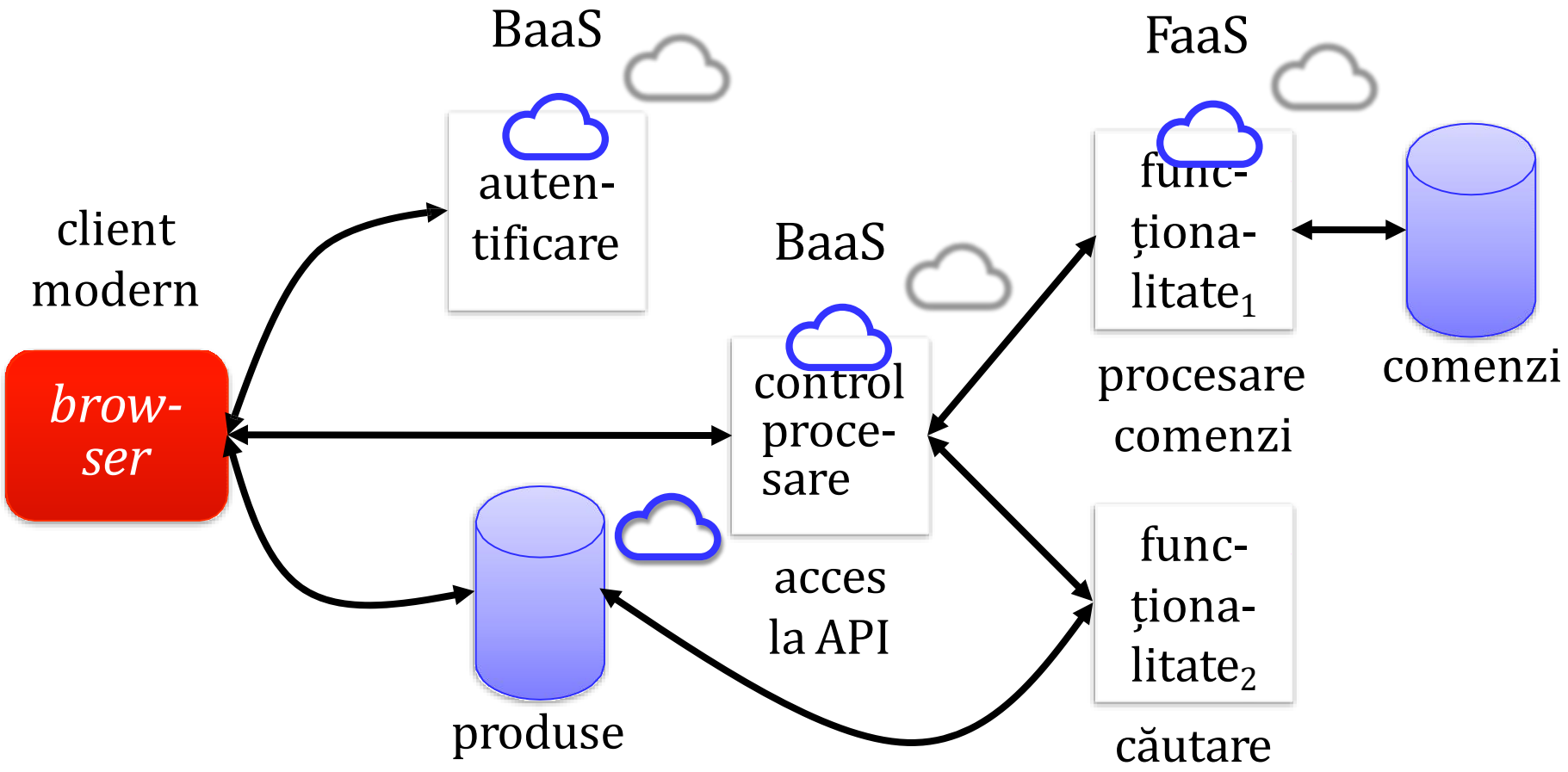
▼

abordarea ***serverless***

Serverless

strat de abstractizare a accesului la resursele unei platforme de tip *cloud*

Mike Roberts (2018)
martinfowler.com/articles/serverless.html



BaaS = (*Mobile*) *Backend As A Service* **FaaS** = *Functions As A Service*

FaaS – *Functions As A Service*

funcții (*cloud functions*) implementând funcționalități expuse consumatorului de servicii

as small as possible

uzual, implementări sub 100 de linii de cod

FaaS – *Functions As A Service*

executate – la nivel de server – independent și asincron, fără a cauza efecte colaterale

declanșate de evenimente

utilizatorul nu e preocupat de managementul resurselor și alte sarcini

BaaS – ***Backend As A Service***

încapsulează servicii de infrastructură ce implementează activități non-funcționale
(autentificare, autorizare, jurnalizare, monitorizare etc.)

private – nu sunt expuse în exterior

pot fi partajate de serviciile interne

Serverless computing = FaaS + BaaS

a se consulta și articolul
Sabin Buraga, *Aspecte arhitecturale vizând dezvoltarea de aplicații serverless* (2019)
itransfer.space/aspecte-arhitecturale-vizand-dezvoltarea-de-aplicatii-serverless/

resurse + soluții software:
github.com/anaibol/awesome-serverless

Cum poate fi descrisă interfața unui API?

Modalități de descriere abstractă:

OpenAPI Specification (ex-Swagger) – **openapis.org**

RAML (*RESTful API Modeling Language*) – **raml.org**

API Blueprint – **apiblueprint.org**

alte resurse de interes:

github.com/Kikobeats/awesome-api

OpenAPI Specification

soluție modernă de a declara – independent de platformă – interfața publică a unui API REST

versiunea curentă: OpenAPI 3.0.2 (septembrie 2019)

formate folosite: JSON și/sau
YAML (*Yet Another Markup Language*)

OpenAPI Specification

bibliotecă JS de procesare:
Spectral (JavaScript, TypeScript)

OpenAPI Specification

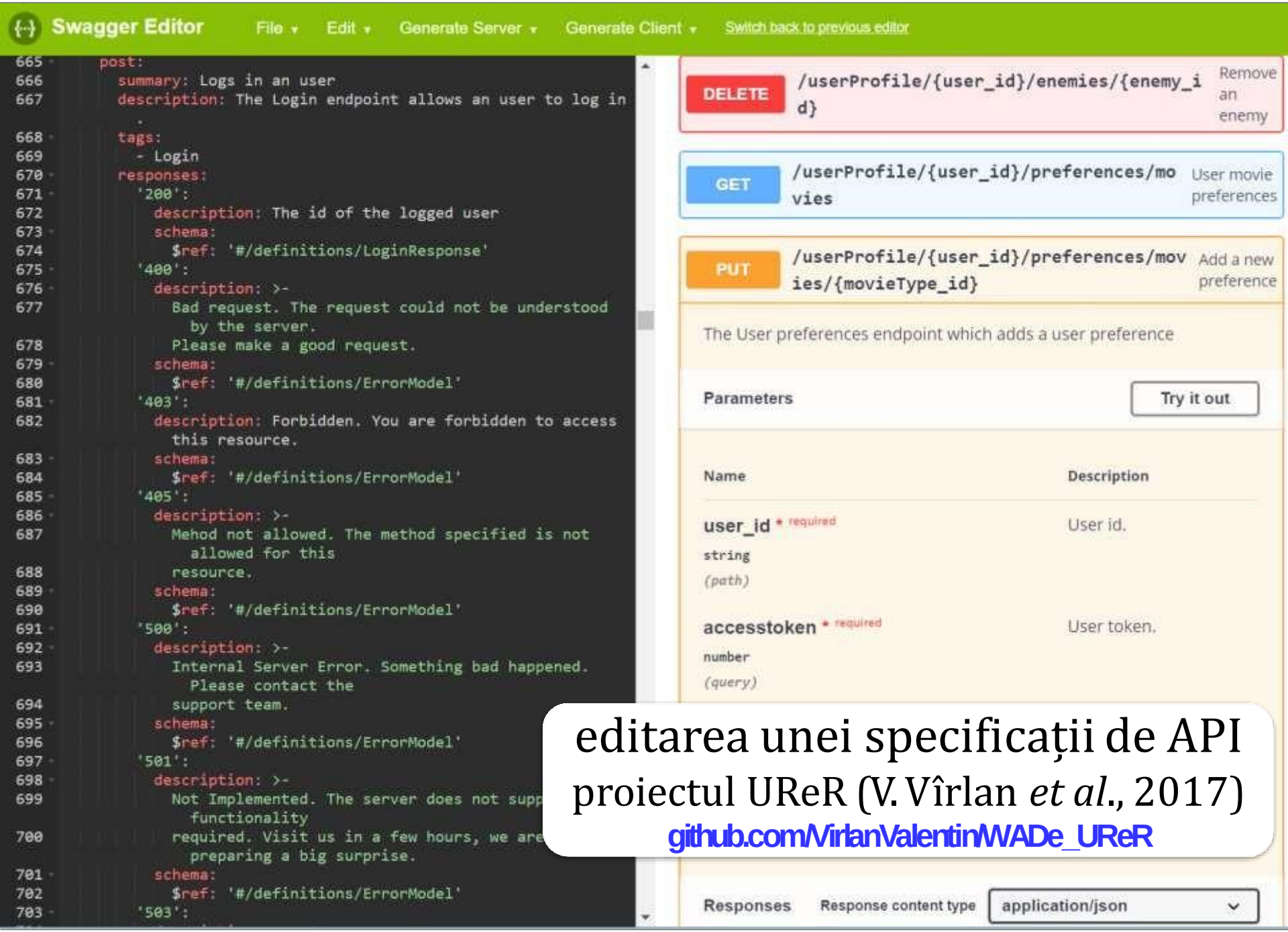
creare de servicii (puncte terminale – *end-points*)
pe baza unui document OpenAPI:
Exegesis (Node.js)
Fusio (PHP, JavaScript)
Vert.x (Java, Kotlin, JS, Ruby, Scala,…)

OpenAPI Specification

generator de cod – exemplificare:
BaucisJS (Node.js)

```
{
 "openapi": "3.0.0",
 "paths": {
   "/resource": {
     "get": {
       "operationId": "service",
       "parameters": [ {
         "name": "parameter",
         "in": "query",
         "schema": { "type" : "string" }
       } ],
       "responses": {
         "200": {
                  "description": "Success",
                  "schema": {
                           "$ref":"#/definitions/Response"
                  }
         }
       }
   }
}
```

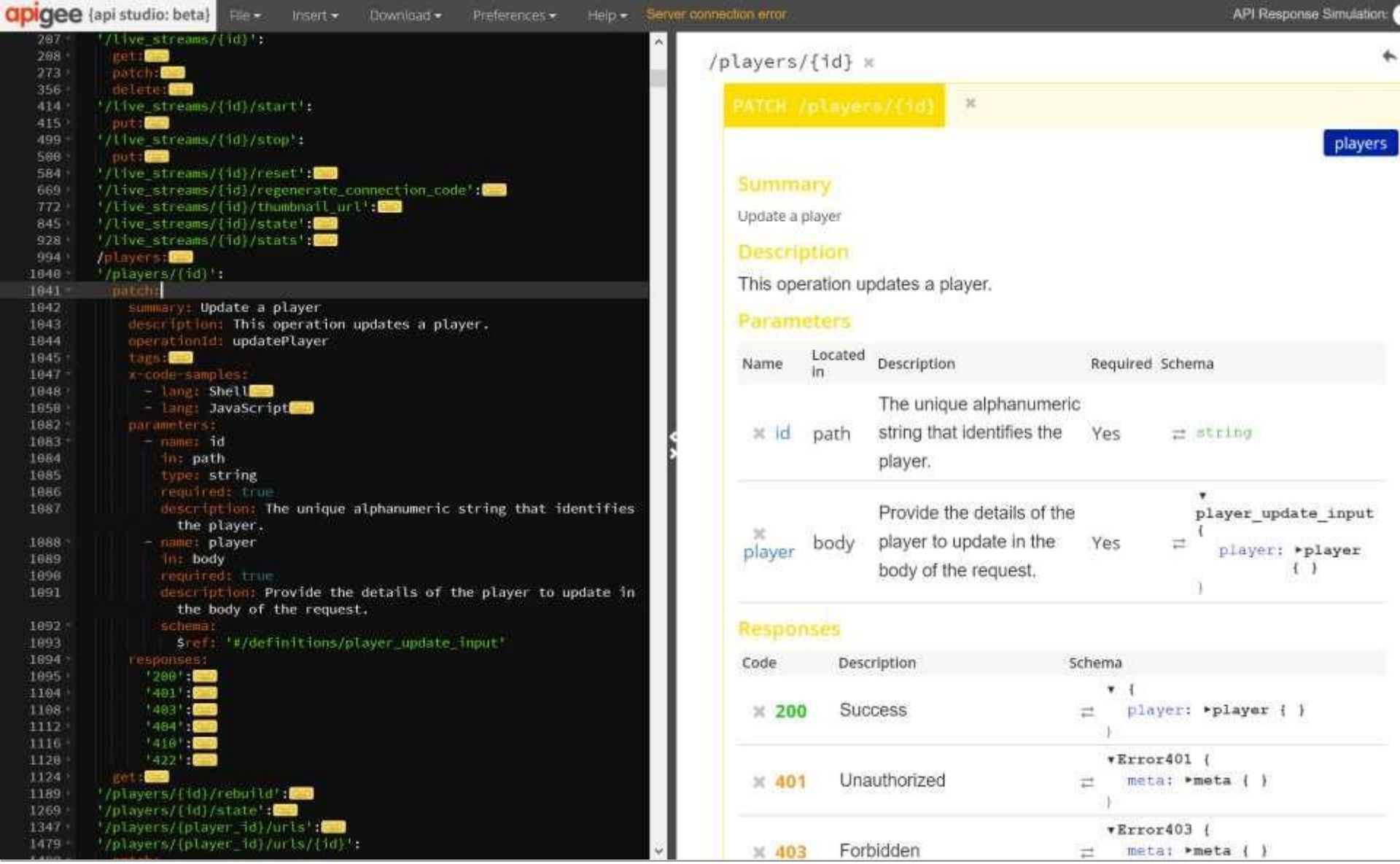
„scheletul” unui document OpenAPI specificând un API



editarea unei specificații de API

proiectul UReR (V. Vîrlan *et al.*, 2017)

github.com/VirlanValentinWADe_UReR



caz concret: Wowza Streaming Engine REST API

specificația OpenAPI editată cu {API Studio}: apistudio.io

SDK (***Software Development Kit***)

încapsulează funcționalitățile API-ului într-o bibliotecă (implementată într-un anumit limbaj de programare, pentru o platformă software/hardware specifică)

API façade pattern

SDK (***Software Development Kit***)

încapsulează funcționalitățile API-ului într-o bibliotecă (implementată într-un anumit limbaj de programare, pentru o platformă software/hardware specifică)

exemplu: Octokit (Go, Java, .NET, Node.js, Ruby,…)
oferit de Github – **developer.github.com/v3/libraries/**

Mash-ups

combinarea – la nivel de client și/sau server – a datelor ce provin din surse (situri) multiple, oferindu-se o funcționalitate/experiență nouă

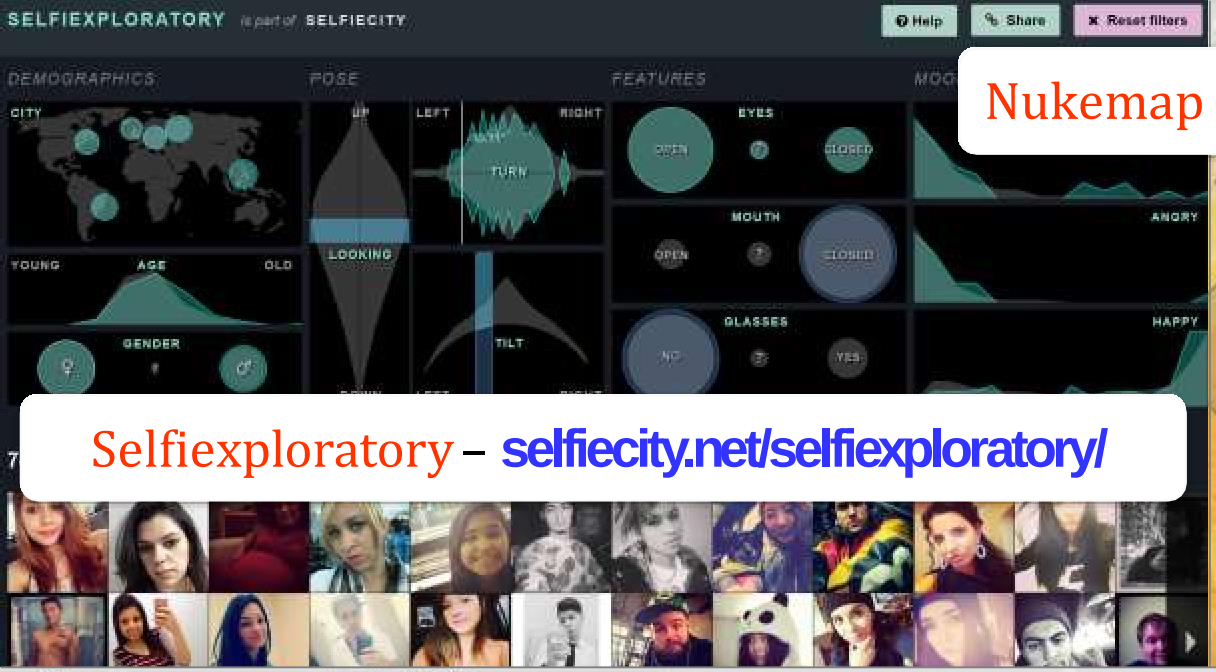
„curentul” SaaS (*Software As A Service*)

Mash-ups

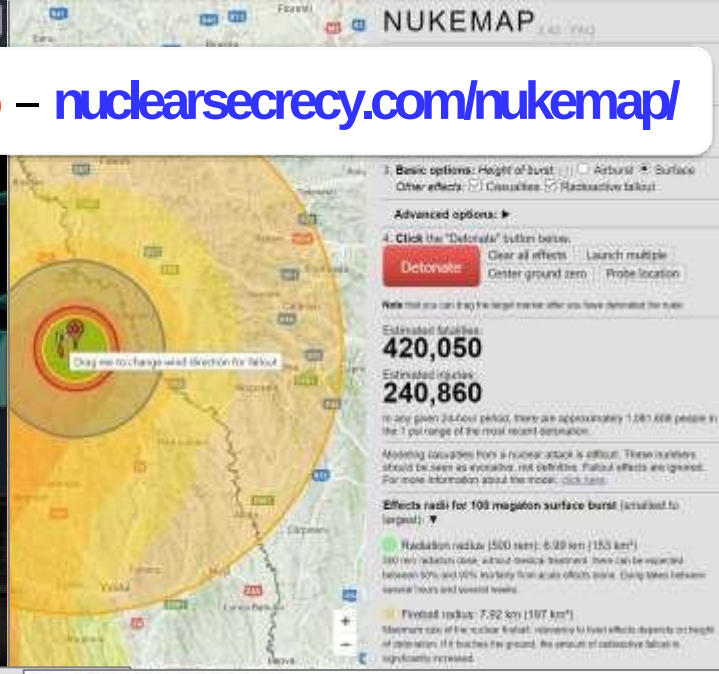
combinare – surse de date (eterogene) multiple

agregare – analizarea datelor
e.g., via *machine/deep learning*, deducții automate,…

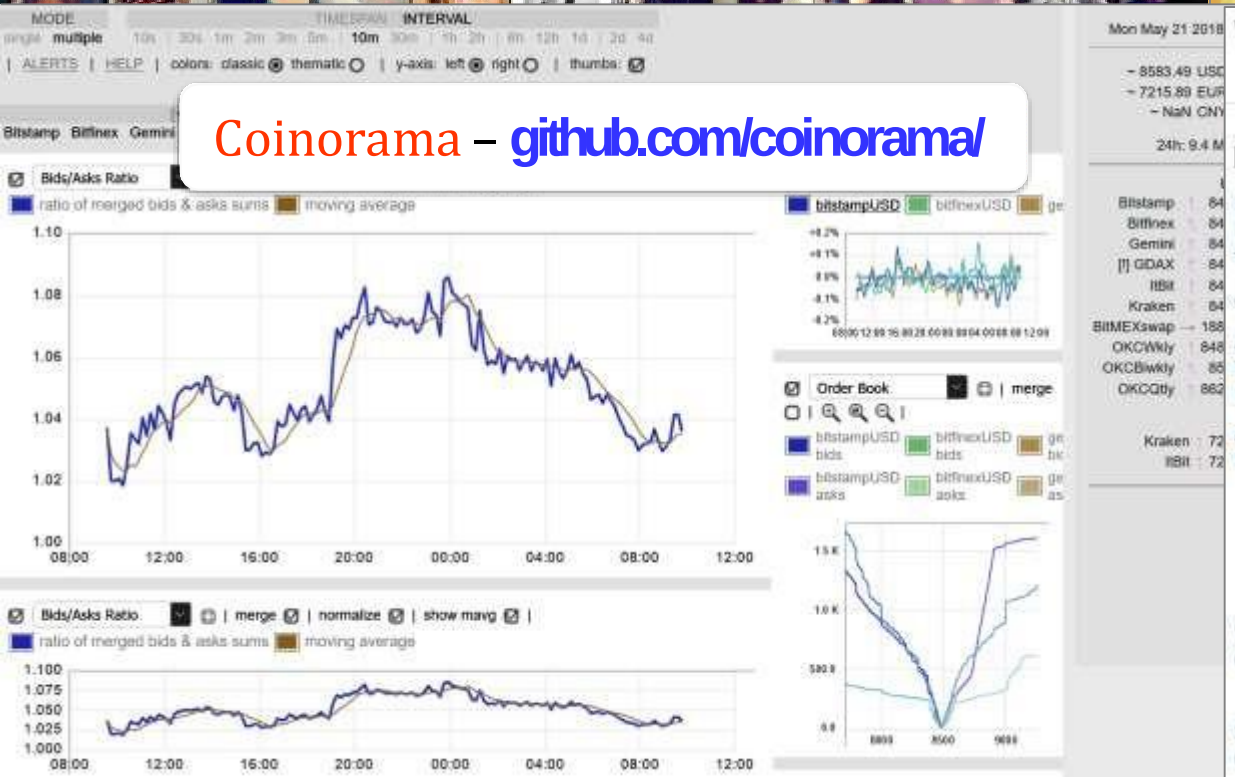
vizualizare – redarea datelor agregate



Selfiexploratory – selfiecity.net/selfiexploratory/



Nukemap – nuclearsecrecy.com/nukemap/



Coinorama – github.com/coinorama/

PopURLS – popurls.com

Web component

parte a unei interfețe Web cu utilizatorul
ce încapsulează o suită de funcții înrudite

e.g., calendar, cititor de fluxuri de știri,
buton de partajare a URL-ului în altă aplicație

Web component

dezvoltare bazată pe o bibliotecă/*framework* JS

soluții – uzual, la nivel de client: Polymer, React, X-Tag,…

în lucru la Consorțiul Web (septembrie 2019)

github.com/w3c/webcomponents/

Widget

aplicație – de sine-stătătoare sau
inclusă într-un container (*e.g.*, un document HTML) –
ce oferă o funcționalitate specifică

rulează la nivel de client (platformă pusă la dispoziție
de sistemul de operare și/sau de navigatorul Web)

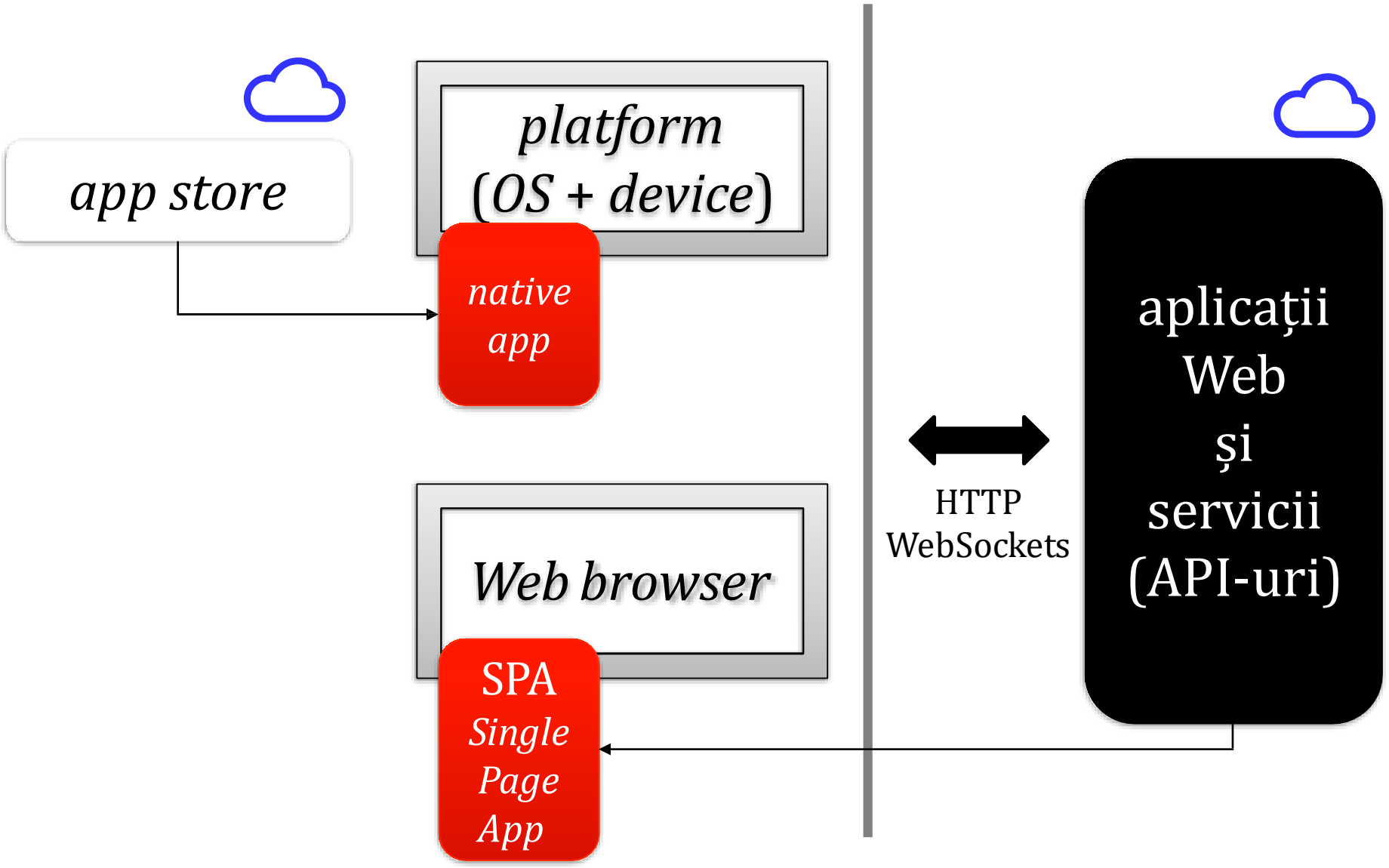
(*Web*) *app*

o aplicație (Web) instalabilă
care folosește API-urile oferite de o platformă:
browser, server de aplicații, sistem de operare,…

(***Web***) ***app***

a distributed computer software application designed for optimal use on specific screen sizes *and with particular* interface technologies

Robert Shilston, 2013



adaptare după Adrian Colyer (2012)

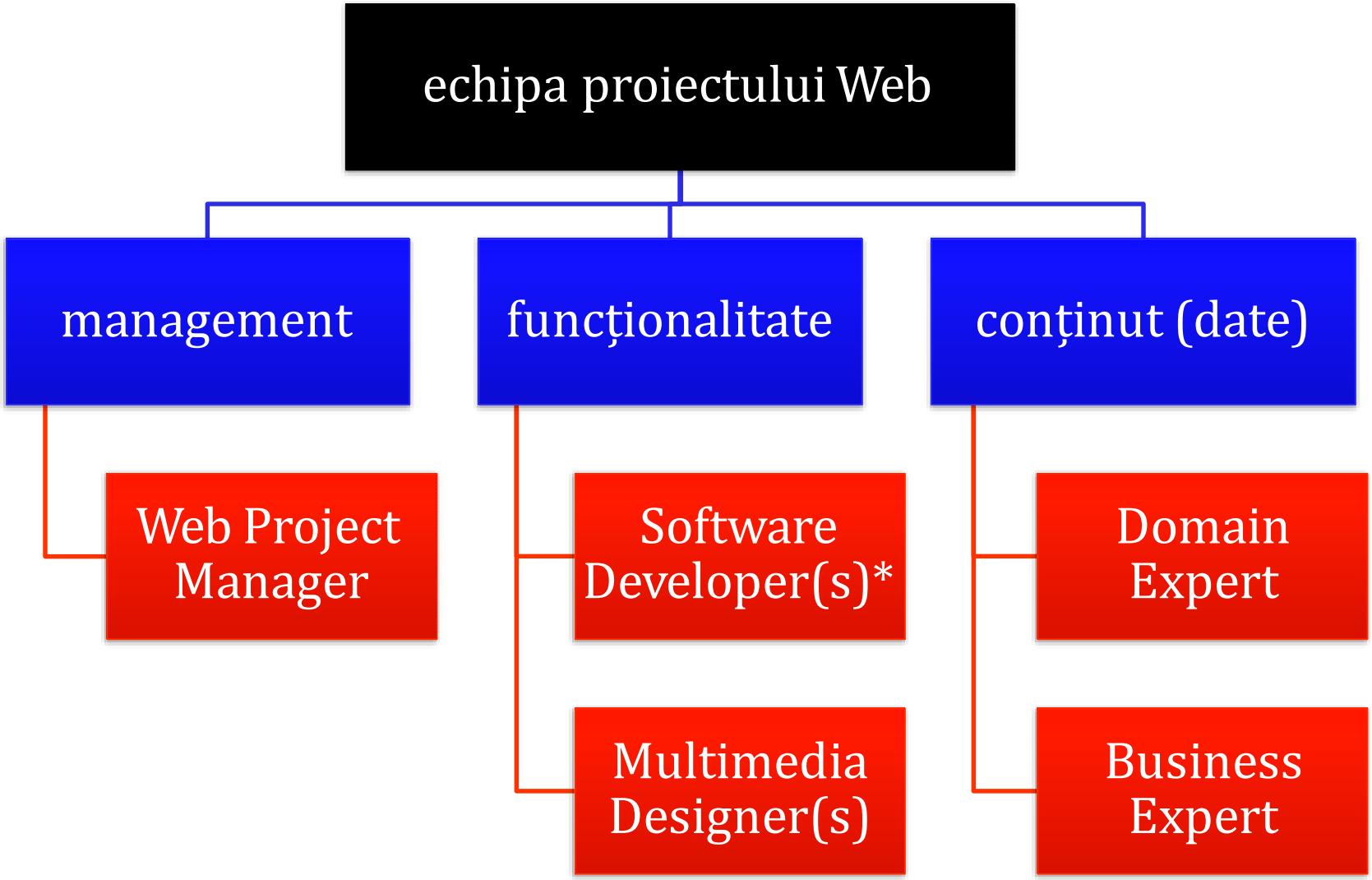
Add-on

denumire generică a aplicațiilor asociate unui *browser*
(extensii, teme vizuale, dicționare,
maniere de căutare pe Web, *plug-in*-uri etc.)

exemplificare: **addons.mozilla.org**

parametrii unui proiect web

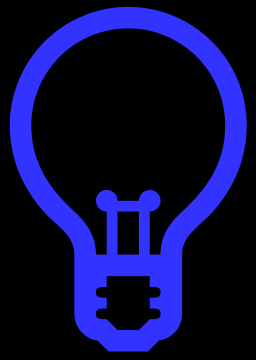
obiectiv principal
durată
cost
abordare
tehnologii
procese
rezultat
resurse umane
profilul echipei



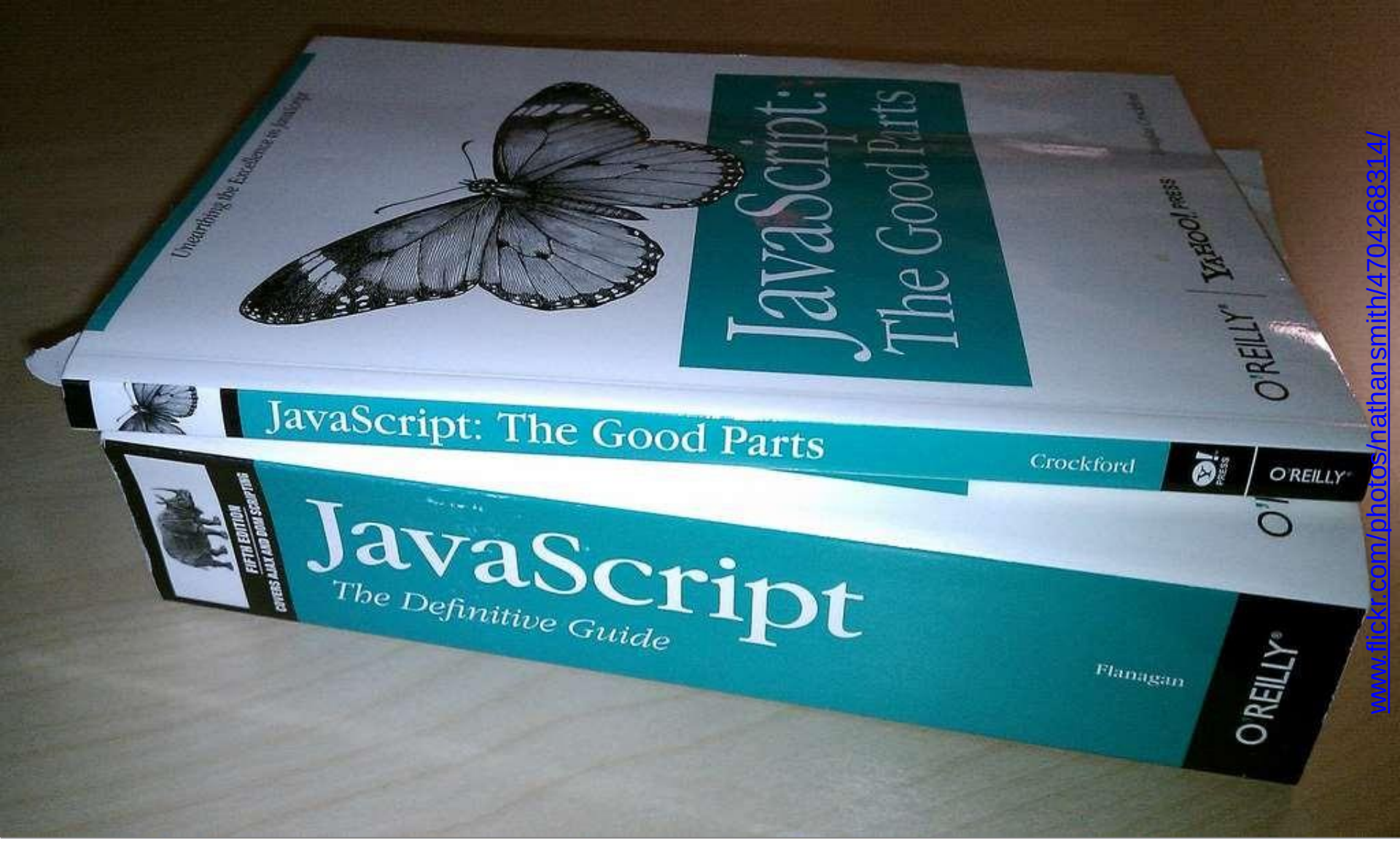
* ***frontend*** sau ***backend*** sau ***full-stack*** (*frontend* + *backend*)

www.slideshare.net/busaco/sabin-buraga-dezvoltator-web-2019/

rezumat



considerații privind arhitectura aplicațiilor Web



episodul viitor: **limbajul JavaScript**