INF5750

Summary

University of Oslo
Department of Informatics

Outline

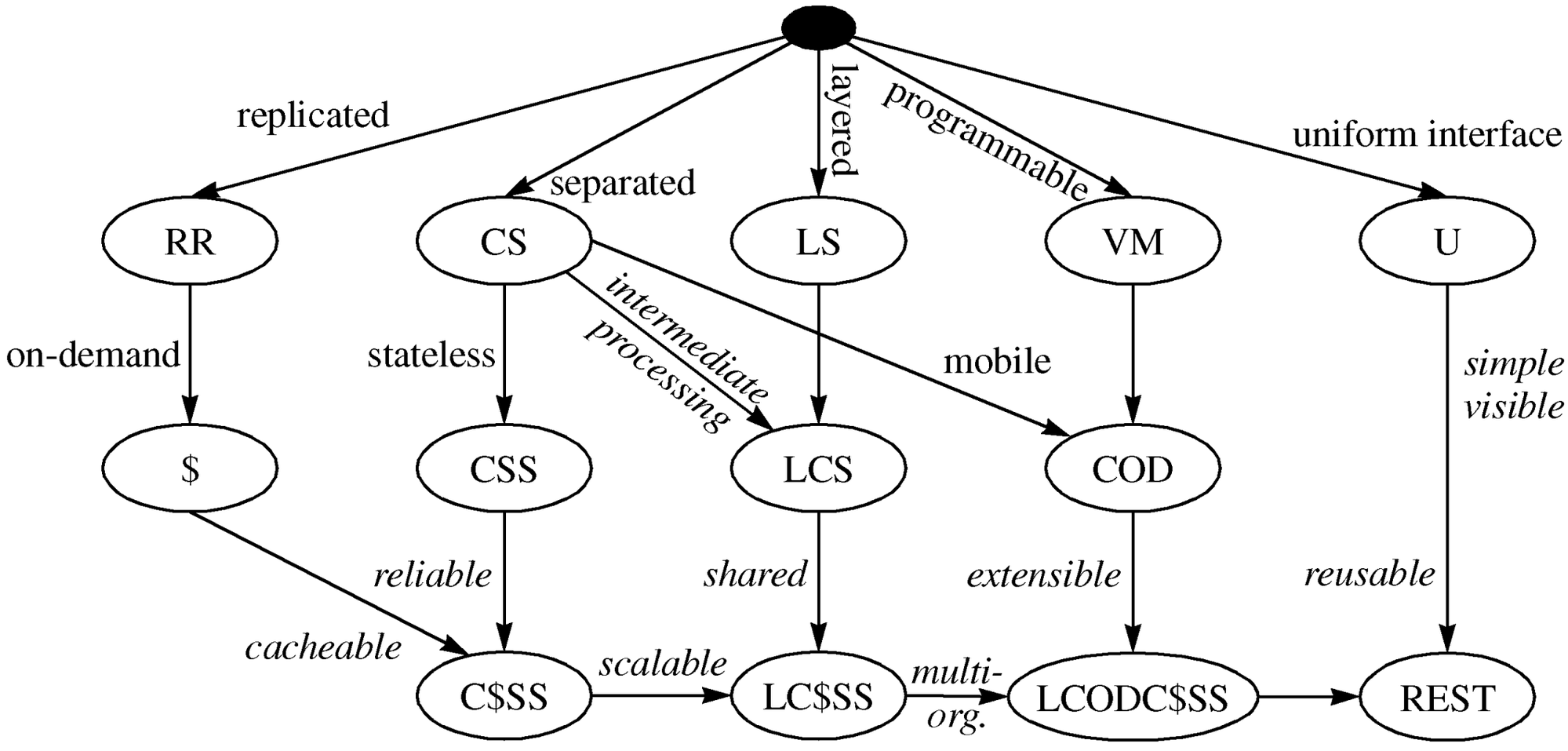
- Summary of key topics
  - REST architecture
  - Free and Open Source Software
  - Software Platform Ecosystems
- Linking theory and practice
- Practical info on exam and project presentation

REST

REpresentational **S**tate **T**ransfer

- REST is an *architectural style*
- Defined by a set of *architectural constraints*
- These guided the development of HTTP
- HTTP is a standard, REST is not

REST architectural constraints



Source: Fielding and Taylor (2002)

REST constraints

- *Addressability* - all resources have a unique and stable identifier
- *Uniform interface* - a uniform interface with a small set of standard methods support all interactions
- *Stateless interactions* - each session is for a single interaction, and session state is not stored by server
- *Self-describing messages* - interaction happens though requests and response message that contain both data and metadata
- *Hypermedia* - resources include links to related resources, enabling decentralised discovery

REST elements

Data element	Example
resource	link to Web service
resource identifier	URL
representation	HTML document, XML document, image file
representation metadata	media type, last-modified
resource metadata	source link, alternates
control data	cache-control

Resources

- Resources are the key information elements in REST
- Any information that can be named can be a resource - image, service, document
- Resources refer to conceptual mappings, not particular entities or values
- Abstract definition of resources enables:
  - generality - information is not divided by type, implementation
  - late binding to representation - representation (format) can be decided based on request
  - we can refer/link to (persistent) concepts rather than specific instances of a concept

Resource identifiers

- Each resource needs an identifier

- Identifier is defined by the "author" of the resource, not centralised

Representations

- *Resources* are not transferred between components in the architecture, but *representations* of resources

- Representations consists of both data and metadata describing the data

- Resource metadata provide information about the resource not specific to the representation

- Control data provides information about the message, such as for caching

REST and RESTful

- REST is an architectural style

- RESTful web services are used to describe web services designed according to the REST style

- "RESTful Web services are software services which are published on the Web, taking full advantage and making correct use of the HTTP protocol"

Maturity of RESTful WS

- Whether a web service is RESTful is not *either or*
- Can be seen as a maturity model with levels of adherence to the REST architecture

Level 0 - HTTP as a tunnel

Level 1 - Use of multiple identifiers and resources

Level 2 - *Proper* use of uniform resource interface and verbs

Level 3 - Use of hypermedia to model relationships

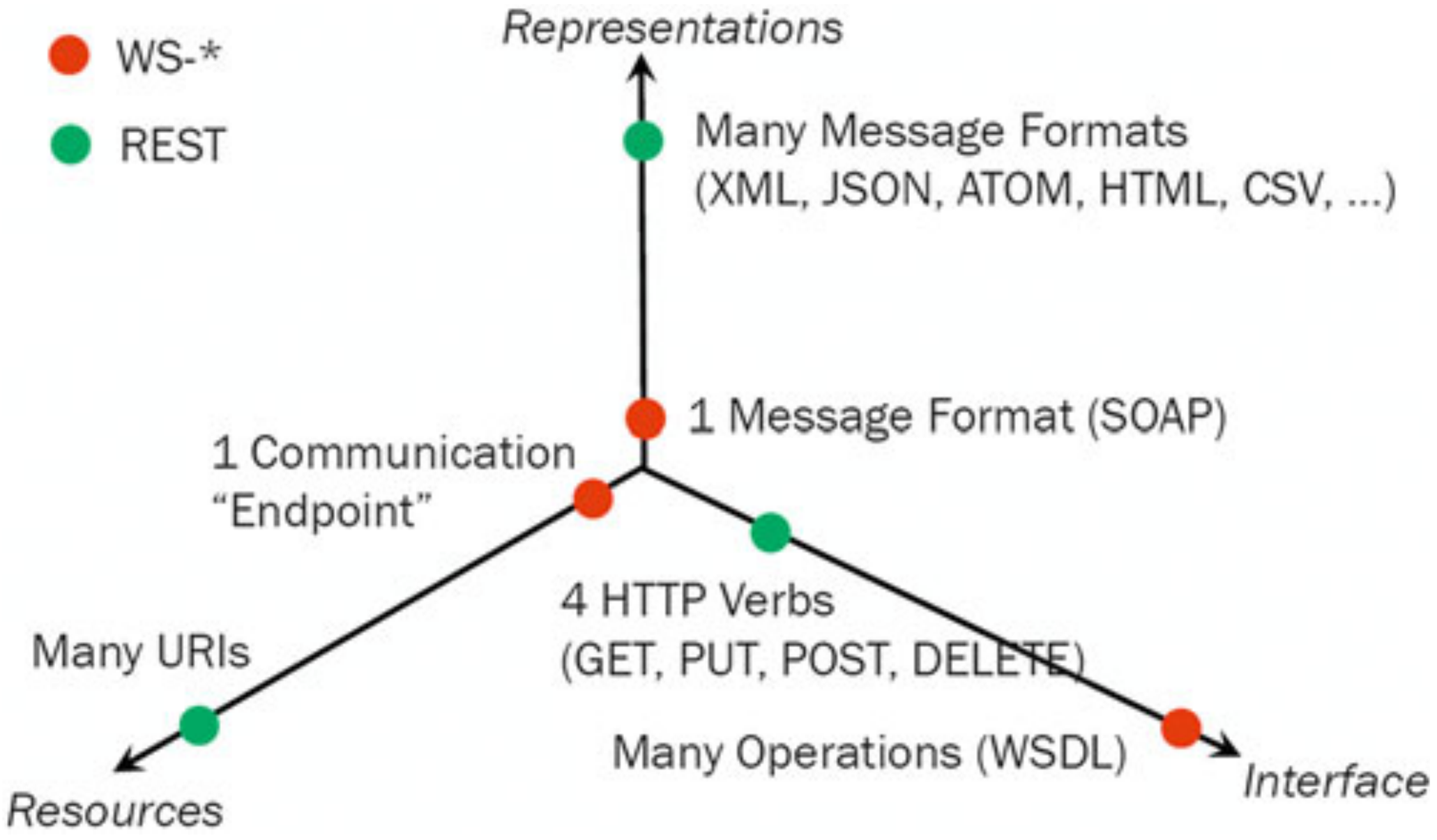
"Big" Web Services

- Traditional (non-RESTful) web services are often called "big" web services
- Commonly based on using two standards:
    - WSDL (Web Services Description Language) - XML format for describing/defining the web service
    - SOAP - XML format for communication

"Big" web services

- Based on interacting with *services* e.g. through remote procedure calls (RPCs)
- All operations are typically POSTed to one/few endpoint(s)
- Operations to be performed is based on content of SOAP (or similar) message rather than an HTTP verb
- Extensions to SOAP for specific functionality - WS-Security, WS-Policy, WS-Addressing etc

RESTful vs other WS



Open Source

FOSS

- Software is created by an author and is subject to *copyright*

- A *license* is needed for software to be used by others

- The term o*pen source* coined by Open Source Initiative (OSI), established in 1999

- OSI has a list of 10 criteria for OSS to comply with

OSI Criteria for OSS

1. Free redistribution
2. Source code available
3. Derived works allowed
4. Integrity of author’s source code allowed
5. No discriminations against persons or groups
6. No discrimination against fields of endeavour
7. Distribution of license
8. License must not be specific to product
9. License must not restrict other software
10. License must be technology neutral

Free vs Open

- Philosophical differences between *free* and *open*
- Free software refer to freedom, not cost - "free speech", not "free beer"
- Based on promoting social solidarity and sharing
- *Free* software meet the 10 criteria for open source
- Practical difference: *free* licenses (e.g. GPL) *require* derivative work to be open source

Models for production of software

1. Managerial command systems - firms and organisations with "lines of command"

2. Markets - transaction costs define the production

3. Commons Based Peer Production

- OSS can follow any of the models, but peer production is perhaps the "typical" example

The open source approach

- Feller and Fitzgerald (2002) analyses the OSS development approach along 5 dimensions:

  What, Why, When and Where, How, Who

- Fitzgerald (2006) argues that open source is transforming from its "free software" origins to a more mainstream and commercially viable approach

What

- OSS is defined by adherence to the OSI definition
- Dominated by operating and networking system software, development tools and infrastructural component
- Examples:
  - Linux operating system
  - Apache web server
  - Perl, Python programming languages
  - V8 javascript engine
  - React, Angular, Vue, Ember++ javascript frameworks

Why

- Three levels of motivations for open source software:
  - Technical
  - Economic
  - Socio-political

Why - technical and economical motivation

- OSS seen as having potential to address "Software crisis" - software taking too long to develop, not working well when delivered, and costing too much

  - Speed - OSS characterised by short development cycles. "Adding manpower to a late software project makes it later" vs "given enough eye-balls, every bug looks shallow".

  - Quality - peer review of source code. Some argue OSS devs are among the most talented and motivated.

  - Cost - shared costs and shared risks of development.

Why - socio-technical motivation

- Motivation of individual developers often socio-technical
- Studies point to "rush" of being able to produce something that get feedback and is used by others
- Meritocracy, where quality of code speaks for itself
- Arena for demonstrating skills for potential employers
- Different in OSS projects where developers are paid

When and Where

- Decentralised geographically - distribution of work

- Rapid evolution with frequent, incremental releases

How

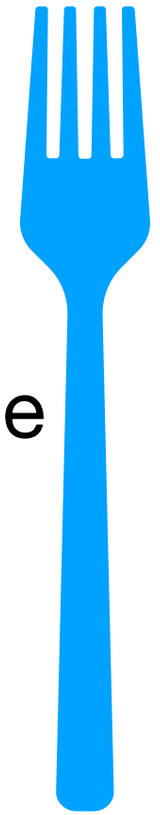
- Classic (early) example:
    - One single or a small group of developers establishes a project and its direction
    - Other developers submit patches to fix bugs or add functionality
    - Examples: apache web server, fetchmail, emacs

How

- Increasingly (OSS 2.0):
  - Companies establish OSS projects as part of a purposeful strategy
  - Developers are paid to contribute
  - Examples:
    - React and Angular largely developed by Facebook and Google
    - Linux kernel top 10 contributors include Intel, Red Hat, Samsung, IBM

How - forks

- Often no written rules within open source projects - customs and taboos must be learned by experience



- The right to **fork** is central to OSS - making a copy of the source code which is then developed separately

- However, forking can be seen as bad practice

Who

- Three key stakeholders on OSS development:
    - Individual developers - often perceived as "hobbyists", but in reality often full-time developers
    - Companies supporting development and distribution
    - Users - experts and early adopters, often the same people who contribute to open source projects

Business models

- Business model: how an organisation creates value
- Major organisations base their business on OSS - Red Hat, SUSE, Canonical, Apache Foundation, Mozilla, eZ System
- Other organisations use OSS without having it as a main business - IBM, Google, Apple, Oracle
- Different business models apply to open source software

Process	FOSS	OSS 2.0

Development Life Cycle	• Planning—"an itch worth scratching"
	• Analysis—part of conventional agreed-upon knowledge in software development
	• Design—firmly based on principles of modularity to accomplish separation of concerns
	• Implementation
	  ○ Code
	  ○ Review
	  ○ Pre-commit test
	  ○ Development release
	  ○ Parallel Debugging
	  ○ Production Release
	(often the planning, analysis, and design phases are done by one person/core group who serve as "a tail-light to follow" in the bazaar)
		• Planning—purposive strategies by major players trying to gain competitive advantage
		• Analysis and design—more complex in spread to vertical domains where business requirements not universally understood
		• Implementation subphases as with FOSS, but the overall development process becomes *less* bazaar-like
		• Increasingly, developers being paid to work on open source

Product Domains	• Horizontal infrastructure (operating systems, utilities, compilers, DBMS, web and print servers)	• More visible IS applications in vertical domains

Primary Business Strategies	• Value-added service-enabling
	• Loss-leader/market-creating
		• Value-added service enabling
		  ○ Bootstrapping
		• Market-creating
		  ○ Loss-leader
		  ○ Dual product/licensing
		  ○ Cost reduction
		  ○ Accessorizing
		• Leveraging community development
		• Leveraging the open source brand

Product Support	• Fairly haphazard—much reliance on e-mail lists/bulletin boards, or on support provided by specialized software firms	• Customers willing to pay for a professional, whole-product approach

Intellectual Property

- Intellectual property:
  - "Non-physical property that is the product of original thought".
    Stanford Encyclopaedia of Philosophy
  - "[IP] refers to creations of the intellect for which a monopoly is assigned to designated owners by law".
    Wikipedia

- Intellectual property **rights** do not address the abstract idea, but the physical manifestation or expression of ideas

- Covered by international treaties (e.g. Bern convention from 1886) and national law in most of the world, but laws differ

IPR protection

- Protection of IPR are mainly through:
  - Copyrights
  - Patents
  - Trade secrets
  - Trademarks

Copyrights and Patents

- Patents and copyrights are the main instruments of IPR law

- History and purpose are different:

  - Patents are issued by authorities to regulate use of **inventions and ideas** for commercial uses

  - Copyrights applies to the **expression of works** like printed material, sound recordings, software - not ideas

Balancing act

protecting rights of author/inventor to incentivise creation

making works and inventions available to the benefit of the public

- IPR law aims to strike balance between incentivising creators and making sure society benefits from creations

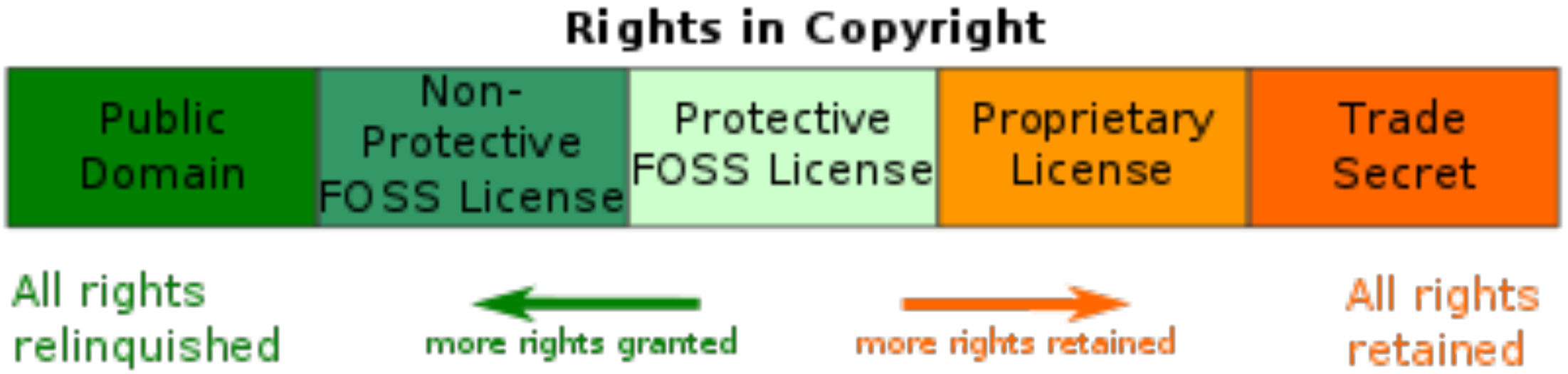
Licensing

- IPR grant rights to authors of their work - including authors of software

- Providing content without license information is legal, but can create confusion

- To use intellectual property written by someone else a *license* is required - including for software in binary or source code format

Software licenses



Source: Mark Webbink

Restrictive vs Permissive

- Permissive licenses *allow* distribution of source code, but only *require* attribution - "minimal restrictions on future behaviour" (FreeBSD)

- Restrictive (copyleft) licenses *require* source code to be distributed along with binary code - aim to keep software free in the future

Free vs Open

- Goes back to philosophical differences between *free* and *open*

- Free software refers to freedom, not cost - "free speech", not "free beer"

- Based on promoting social solidarity and sharing

- *Free* software meets the 10 criteria for open source

- *Open source* software does not necessarily adhere to the requirement of *free software* licenses (e.g. GPL) that require derivative work to also be open source

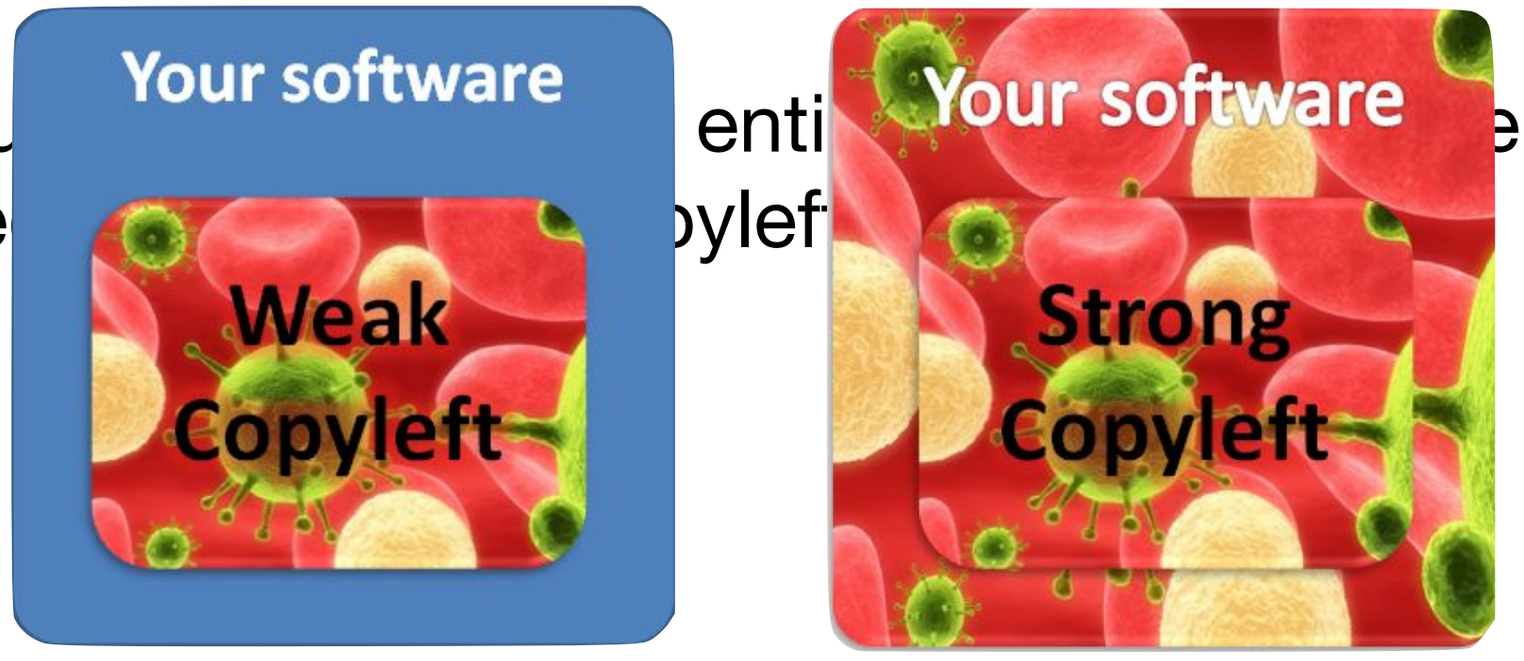
Restrictive licenses

- Weak restrictive/copyleft license:
    - If software with weak copyleft is used *that module/ library's source code must be distributed/made available*
- Strong restrictive/copyleft license:
    - If software with strong copyleft is used *the entire software's source code must be distributed/made available*

Viral licenses

- Strong copyleft is *viral*

- When u enti e
  releas oylef



Distributing OSS

- Requirement to distribute source code in open source licenses is linked to distribution of object/binary code

- *Internal* modification and use of OSS software does not usually trigger requirement to publish modified code

- Businesses may (should) have a list of accepted OSS licenses and used OSS modules - example

License violations

- Automated tools can be used for detecting licensing issues

- Review of source code (including licenses) would typically be part of "due diligence" in the sale of a company

- With violations of open source (copyleft) licenses, you could be taken to court and forced to release the source code

- Topic of research, e.g. *We et al (2017)* on inconsistencies of licensing within OSS projects

Software Platform Ecosystems

What are platforms?

- Platforms where third party complementors add to platform capabilities and functionality

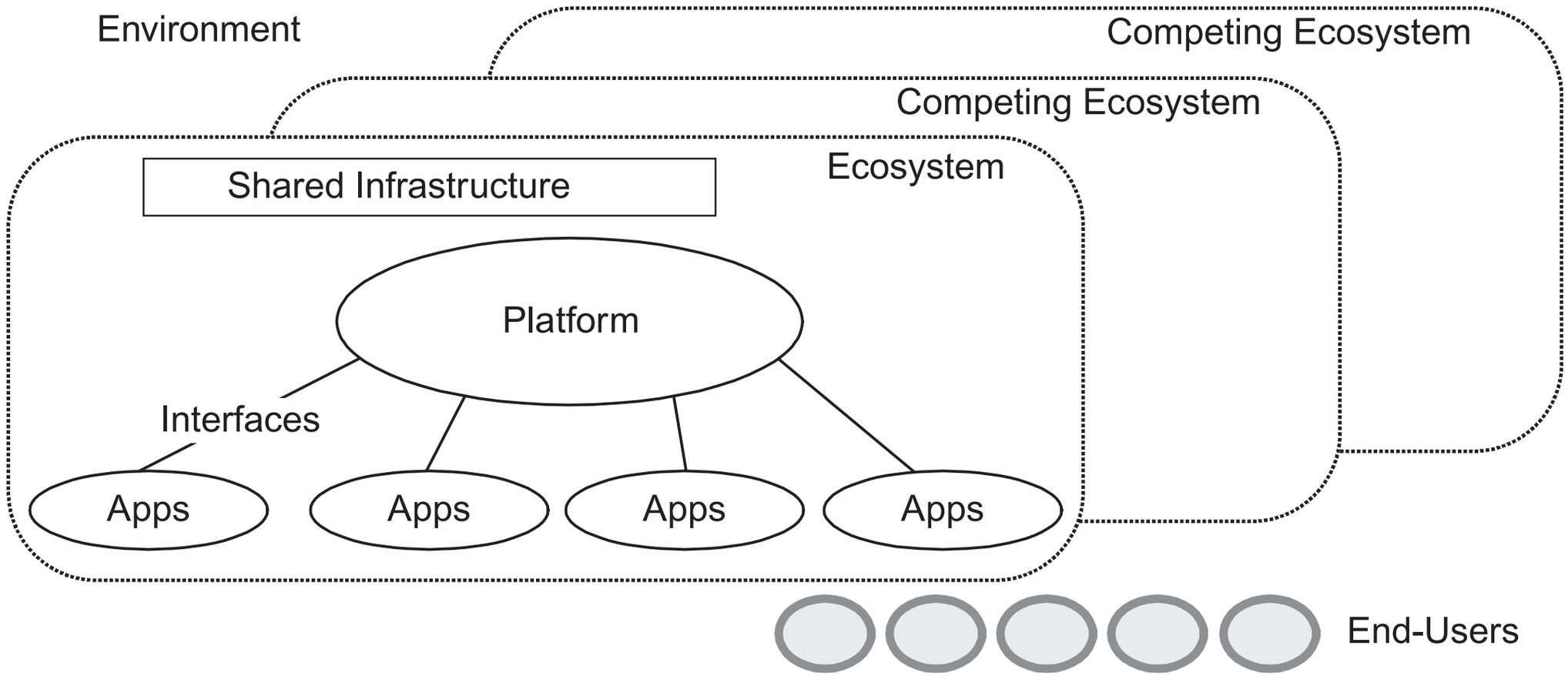
- Possibilities for hundreds or thousands of actors to add functionality to the same ecosystem

- End users can uniquely mix-and-match "downstream complements" – innovation and adoption in the downstream central for success of failure

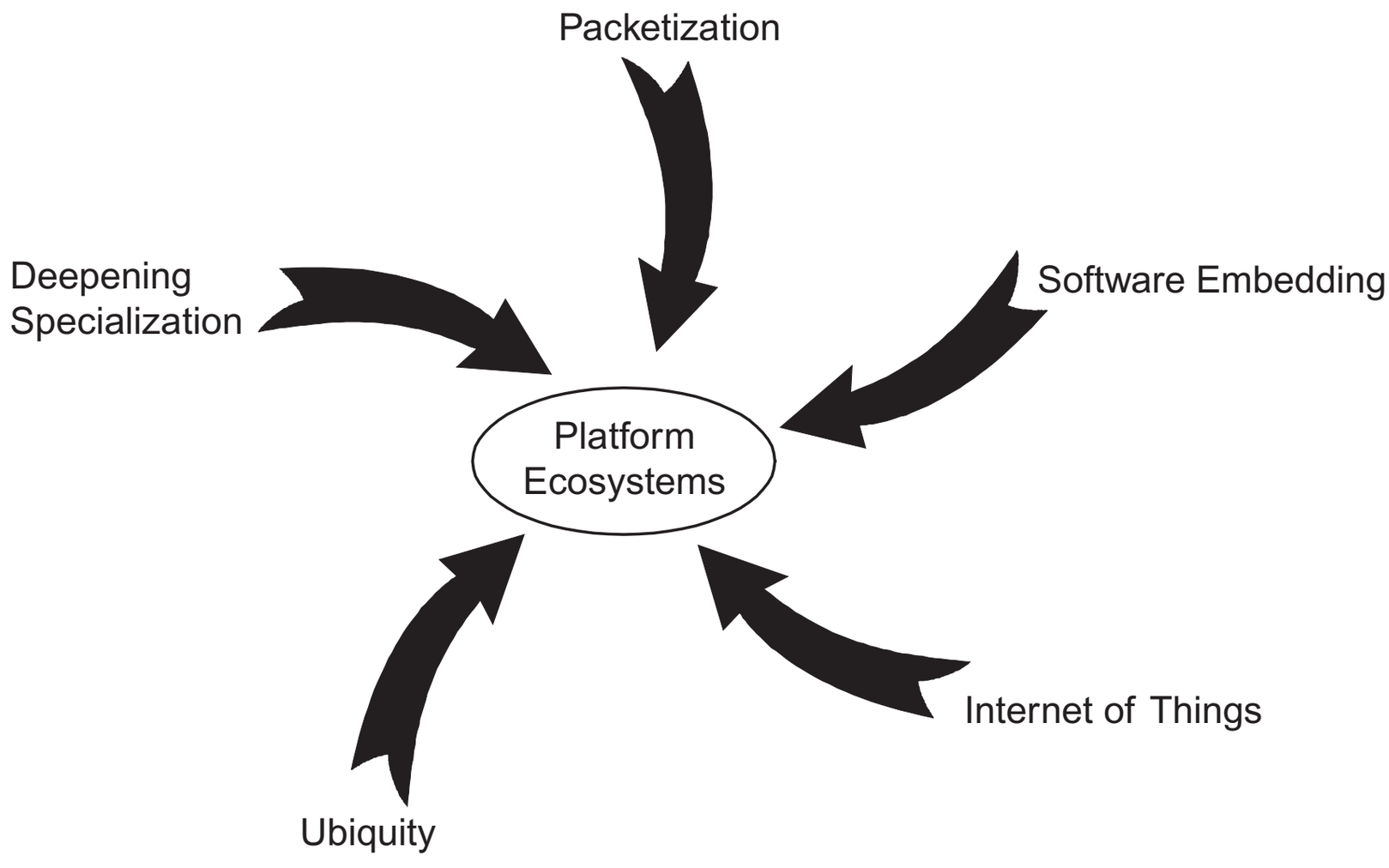
- True platforms must be at least two-sided - spanning at least app developers and end-users that interact through the platform

- Most successful platforms began as standalone products or services: iOS, Windows, Facebook, Amazon, eBay, Google, Firefox, Salesforce, and Dropbox

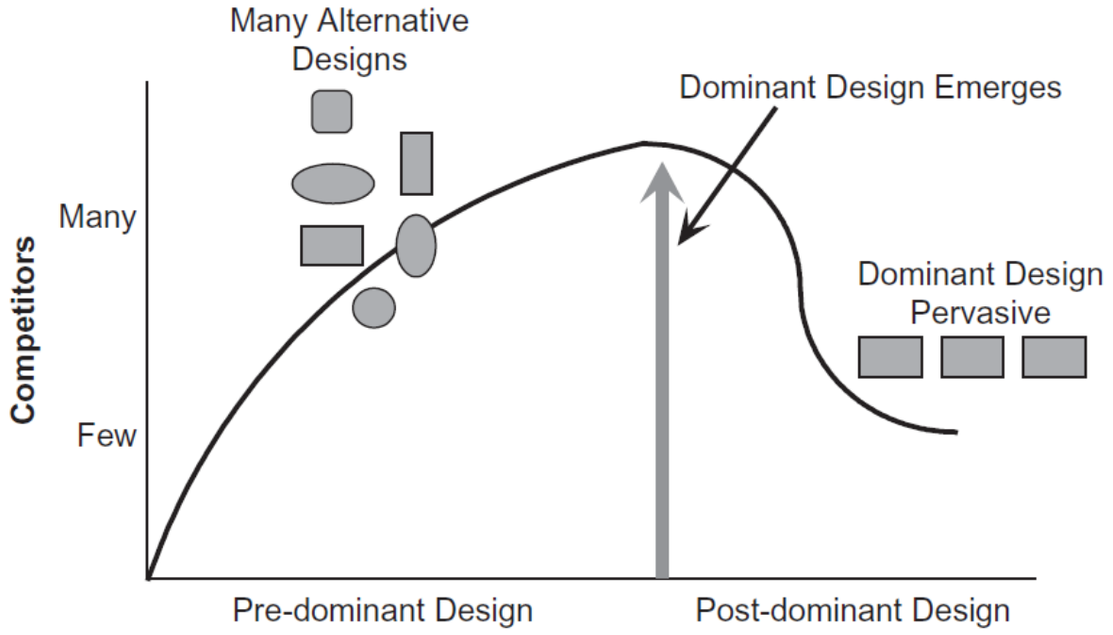
Core elements

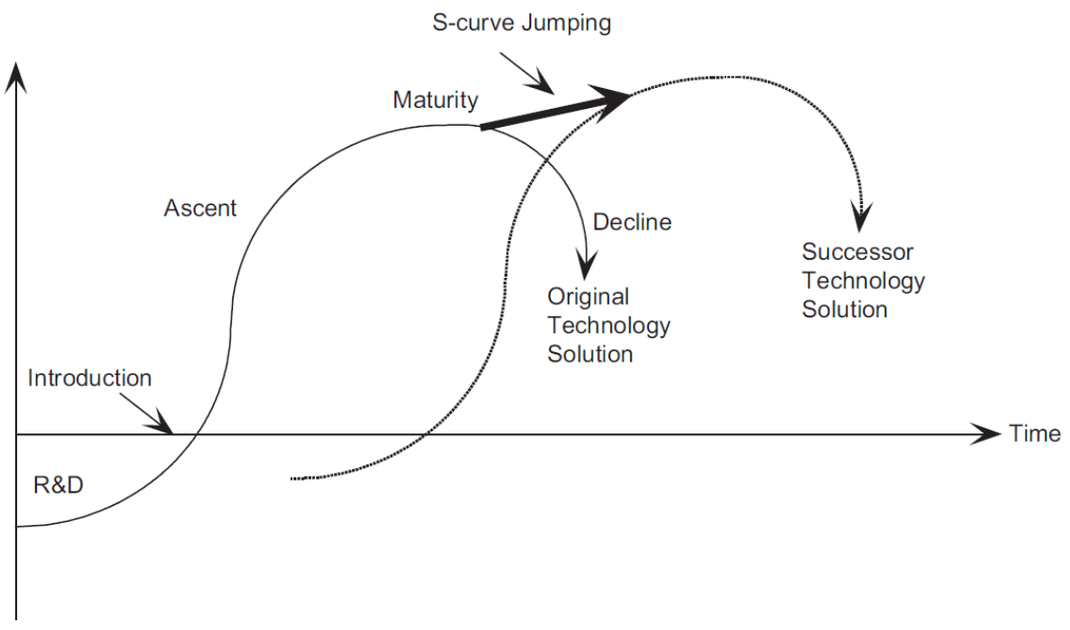


Drivers towards platformization



Platform lifecycle





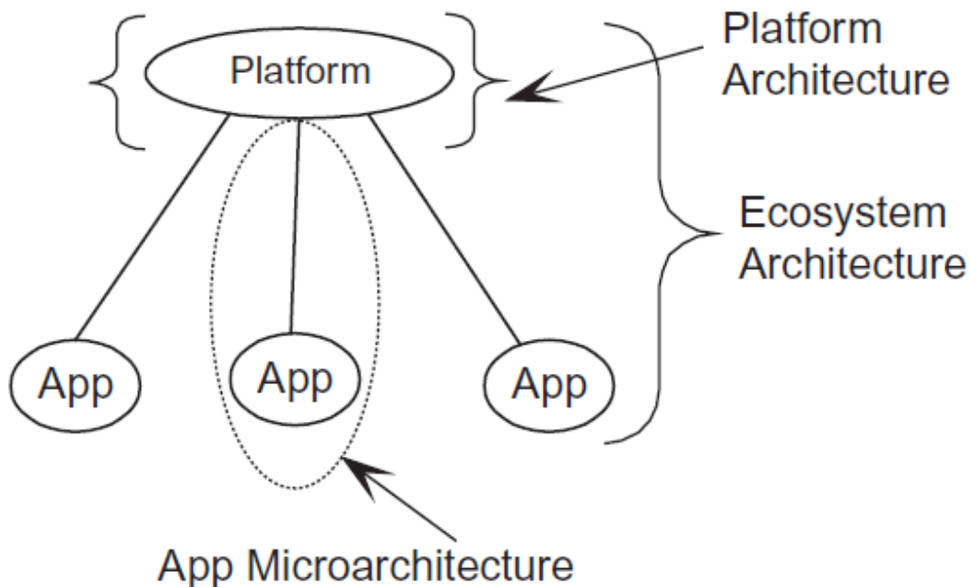
Guiding principles

Table 2.3 Summary of the Nine Guiding Principles in Platform Markets

	Principle	Key Idea
Startup	Red Queen effect	The increased pressure to adapt faster just to survive is driven by an increase in the evolutionary pace of rival technology solutions
Startup	Chicken-or-egg problem	The dilemma that neither side will find a two-sided technology solution with potential network effects attractive enough to join without a large presence of the other side
Startup	The penguin problem	When potential adopters of a platform with potentially strong network effects stall in adopting it because they are unsure whether others will adopt it as well
Evolution	Emergence	Properties of a platform that arise spontaneously as its participants pursue their own interests based on their own expertise but adapt to what other ecosystem participants are doing
Design	Seesaw problem	The challenge of managing the delicate balance between app developers' autonomy to freely innovate and ensuring that apps seamlessly interoperate with the platform
Design	Humpty Dumpty problem	When separating an app from the platform makes it difficult to subsequently reintegrate them
Design	Mirroring principle	The organizational structure of a platform's ecosystem must mirror its architecture
Evolution	Coevolution	Simultaneously adjusting architecture and governance of a platform or an app to maintain alignment between them
Evolution	Goldilocks rule	Humans gravitate toward the middle over the two extreme choices given any three ordered choices

Architecture

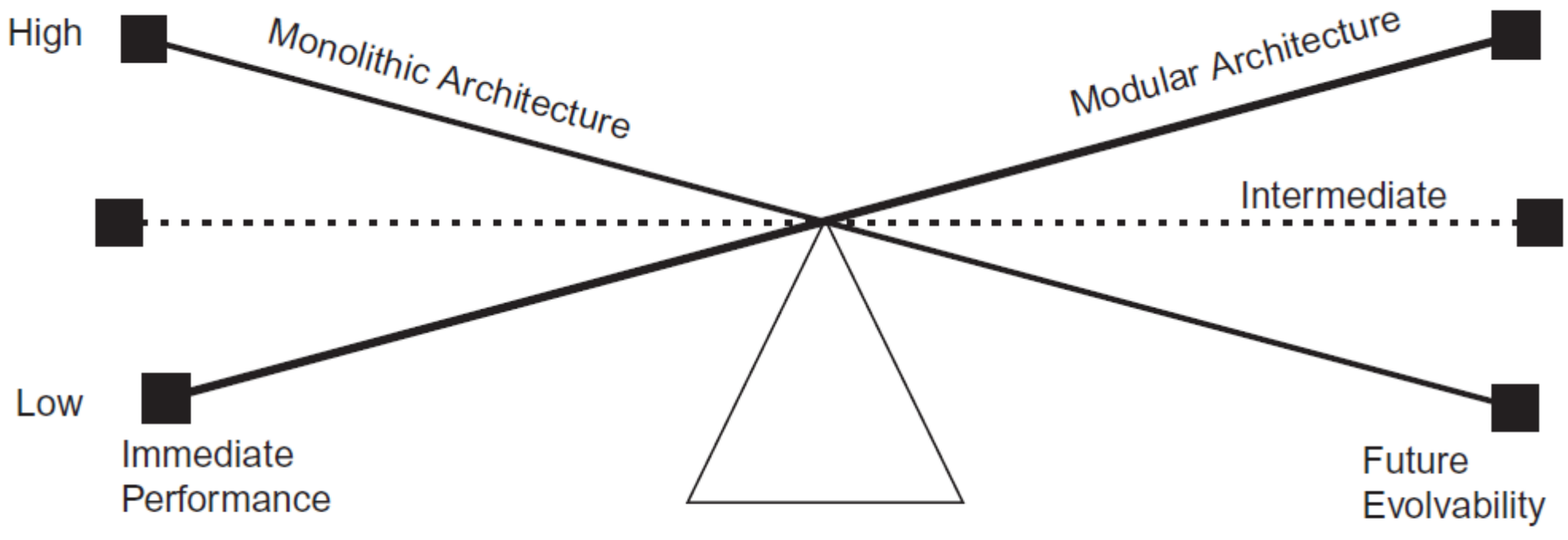
- Platform architecture enable third parties to participate in the platform ecosystem
- Important to manage complexity
- Three main components: platform core, interfaces, apps
- Overall: stable core and dynamic apps
- Architecture should enable:
  - Partitioning/modularization
  - System integration



Platform architecture

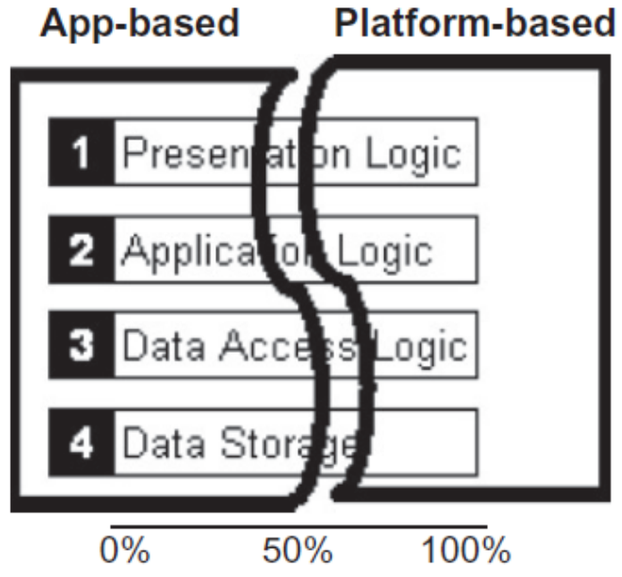
- Path dependent - in practice irreversible

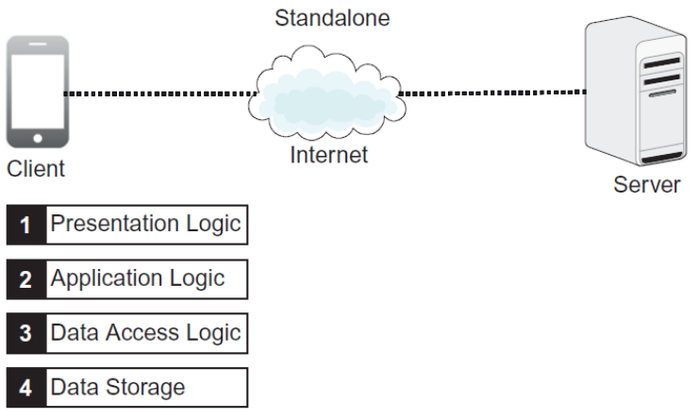
- Balancing act between the different desirable properties:

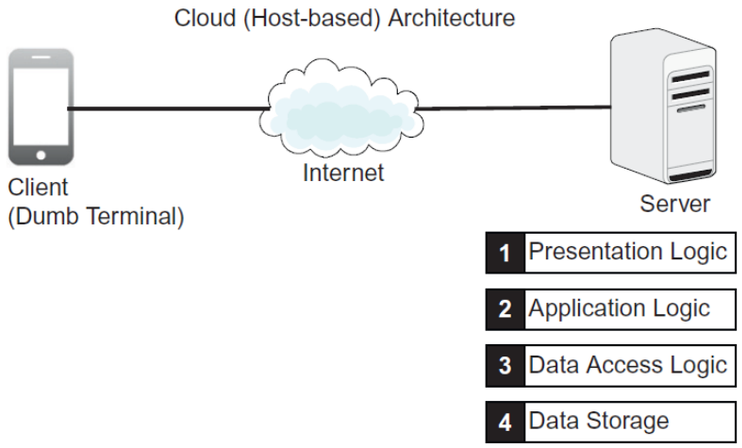


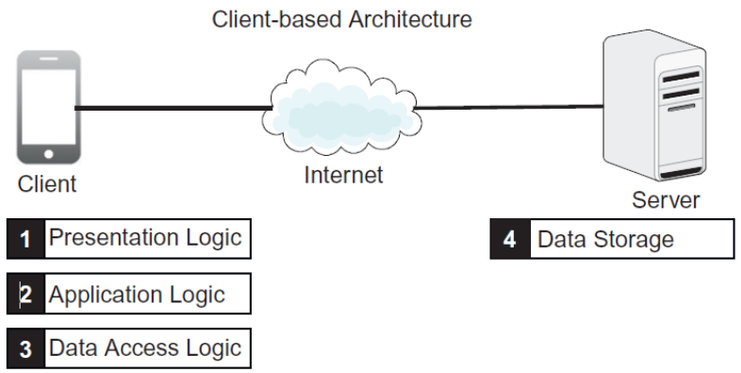
App (micro)architecture

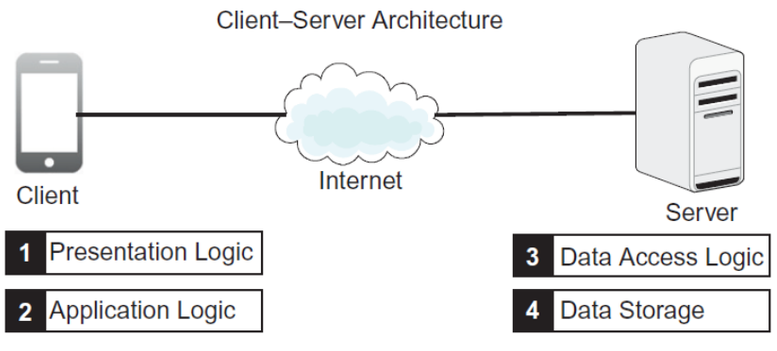
- Partitioning of functionality between app and platform
- Path dependent - difficult to change
- Some characteristics of different solutions are visible immediately, others long-term

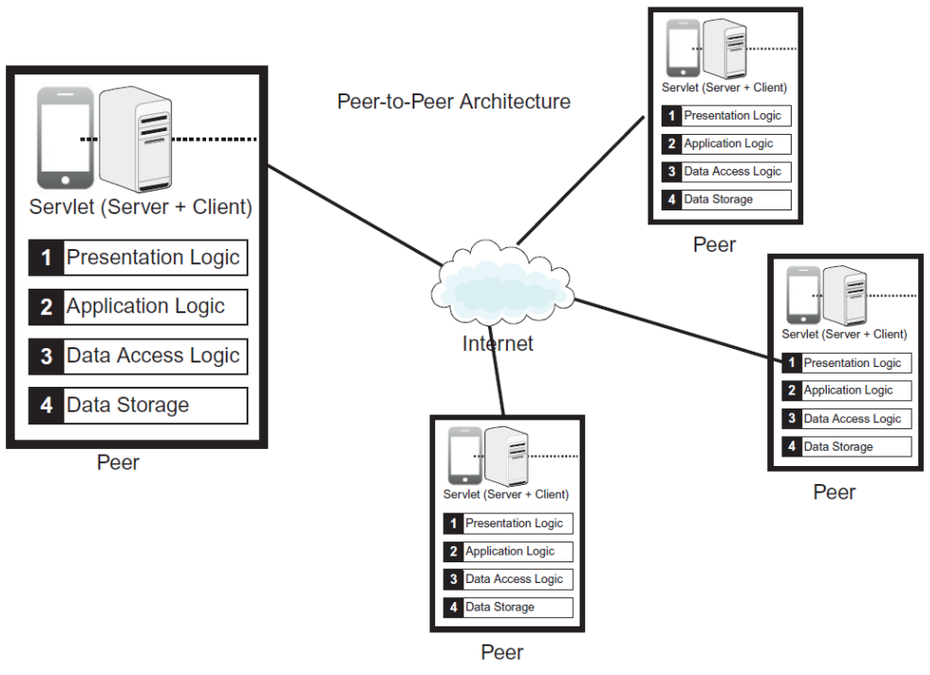


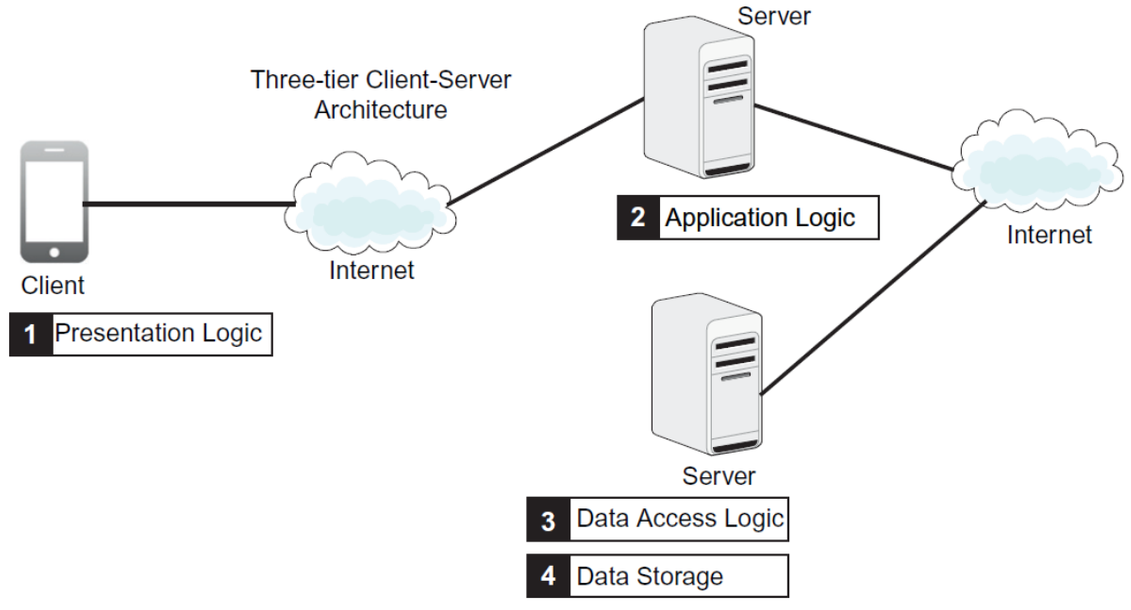

Standalone
Client
Internet
Server
1 Presentation Logic
2 Application Logic
3 Data Access Logic
4 Data Storage


Cloud (Host-based) Architecture
Client
(Dumb Terminal)
Internet
Server
1 Presentation Logic
2 Application Logic
3 Data Access Logic
4 Data Storage

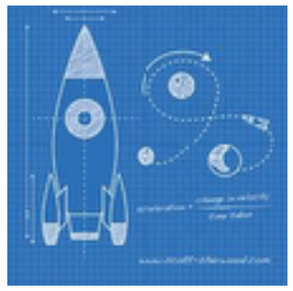

Client-based Architecture
Client
Internet
Server
1 Presentation Logic
2 Application Logic
3 Data Access Logic
4 Data Storage


Client–Server Architecture
Client
Internet
Server
1 Presentation Logic
2 Application Logic
3 Data Access Logic
4 Data Storage


Peer-to-Peer Architecture
Servlet (Server + Client)
1 Presentation Logic
2 Application Logic
3 Data Access Logic
4 Data Storage
Peer
Internet
Servlet (Server + Client)
1 Presentation Logic
2 Application Logic
3 Data Access Logic
4 Data Storage
Peer
Servlet (Server + Client)
1 Presentation Logic
2 Application Logic
3 Data Access Logic
4 Data Storage
Peer
Servlet (Server + Client)
1 Presentation Logic
2 Application Logic
3 Data Access Logic
4 Data Storage
Peer


Three-tier Client-Server
Architecture
Client
1 Presentation Logic
Internet
Server
2 Application Logic
Internet
Server
3 Data Access Logic
4 Data Storage

Theory and Practice



Ecosystem

Platform

Interface

Apps

Apps

Apps

Apps

Apps

Ecosystem

dhis2

RESTful API

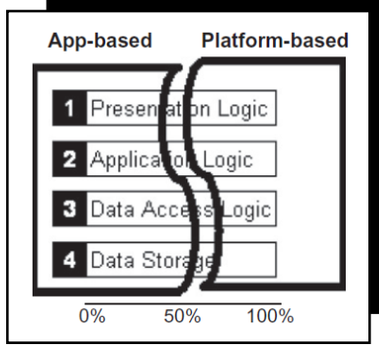
Stock viz

MFL

HIV cascade

Sharing settings

Subnat. analysis



Ecosystem

dhis2

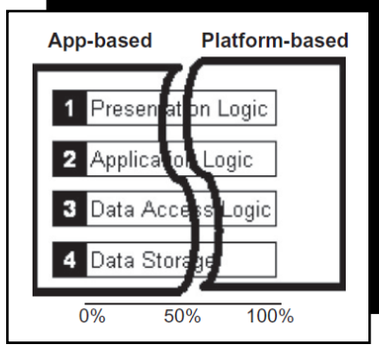
RESTful API

Stock viz

MFL

HIV cascade

Sharing settings



Exam and Presentation

Exam

- December 1 at 2:30 PM, Silurveien 2, Sal 3B
- 4 hour written digital exam
- Similar in style to exam from 2016
- Questions from curriculum, with some links to group projects

Group presentations

- 6 and 7 December - let me know by email if one of the days are impossible
- Each group have **strictly** 20 minutes to present
- More on what the presentation should include here
- Make your repositories public the day before presenting

?