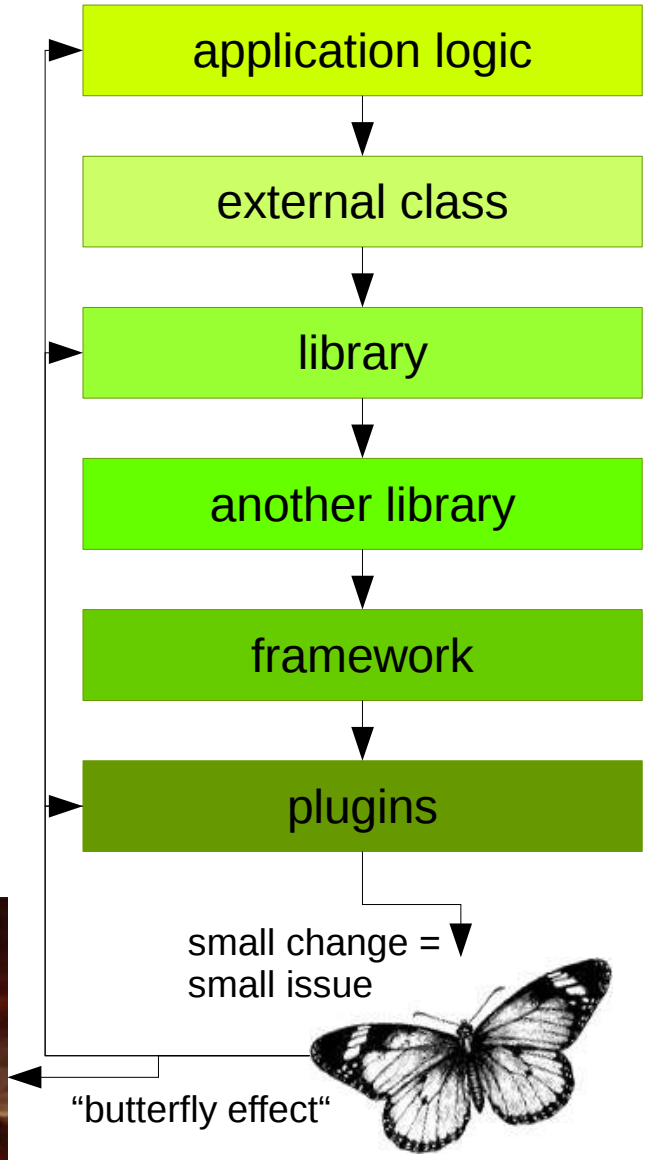
Cloud service dependencies in practice

Josef Spillner <josef.spillner@zhaw.ch>
Service Prototyping Lab (blog.zhaw.ch/splab)

Sep 04, 2019 | INFORTE Summer School, Tampere

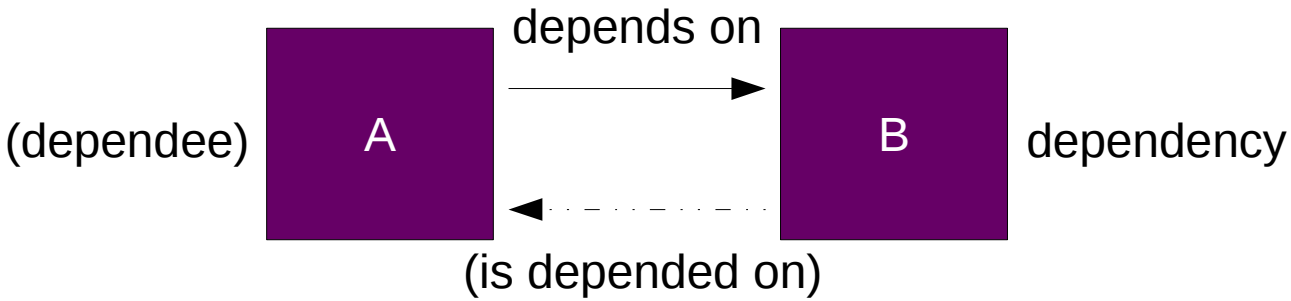
I.
Dependencies Primer

“Standing on the shoulders of giants“



Overview about dependencies

Abstract considerations



Distinguish:

- static vs. dynamic
- visible/recognisable vs. hidden
- explicit vs. implicit
- strict vs. tolerant / strong vs. weak

System model: deps & artefacts

Dependencies in software

- digital artefacts
  - code (executable) - also: components, services
  - configuration (declarative)
  - data (binary)
- specialisations of software engineering process
  - CBSE - component-based
  - SOSE - service-oriented
  - microservice-oriented...

Compositional model

“Compositio“ and “Componens“

[citysuburb.wix.com]

Composition
- of multiple components
- some equal, some with distinct function
  - homogeneous
  - heterogeneous
- goal: whole > Σ parts

Component
- piece of hardware (out of scope)
- piece of software
- restricted view
  - black box: only interface
  - grey box: partial information about the inside and behaviour

Roles: Each object can be component and composition at the same time.

Composition of services

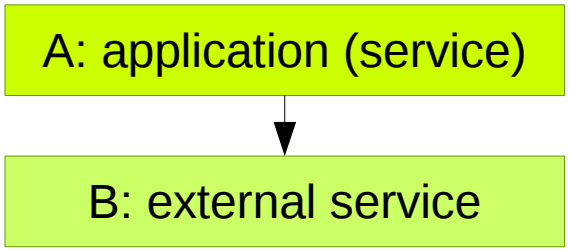
Scenario: New mobile app for mensa food delivery to lecture room

- registration e-mail: re-use existing software?
- menu selection: re-use current menu listing?
- payment: handle different methods?

→ main drivers: re-use, extensibility

Service dependencies

Importance
- if B goes down, A goes down, too
- hidden cost



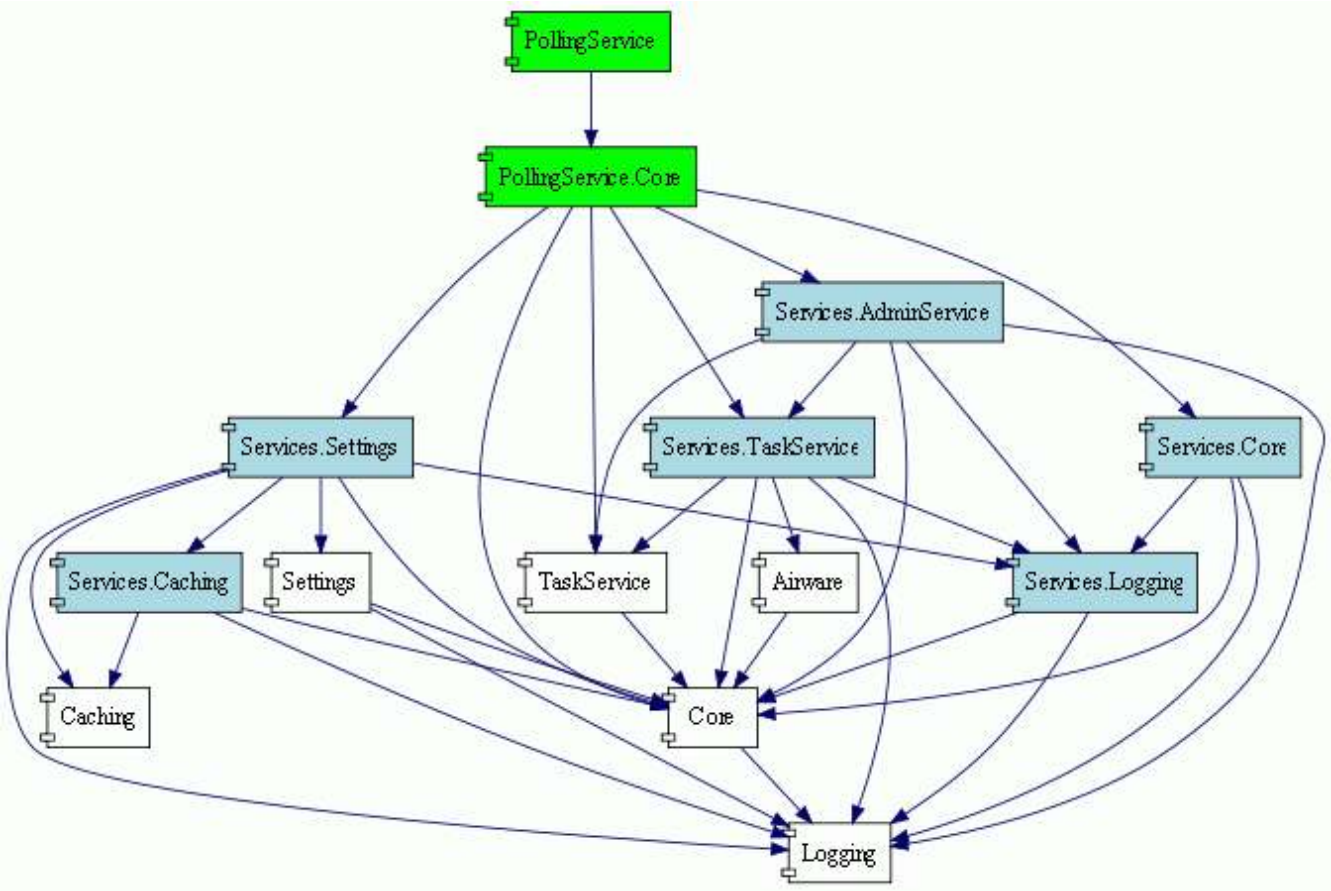
Dependency manifestation
- tight coupling / fixed binding
- loose coupling / late binding

Dependency type
- inter-service
- intra-service (e.g. libraries, backend services)

Dependency consideration
- data sources
- transitivity

Dependency graphs

directed, cyclic (to be avoided) or acyclic, weighted or unweighted graph

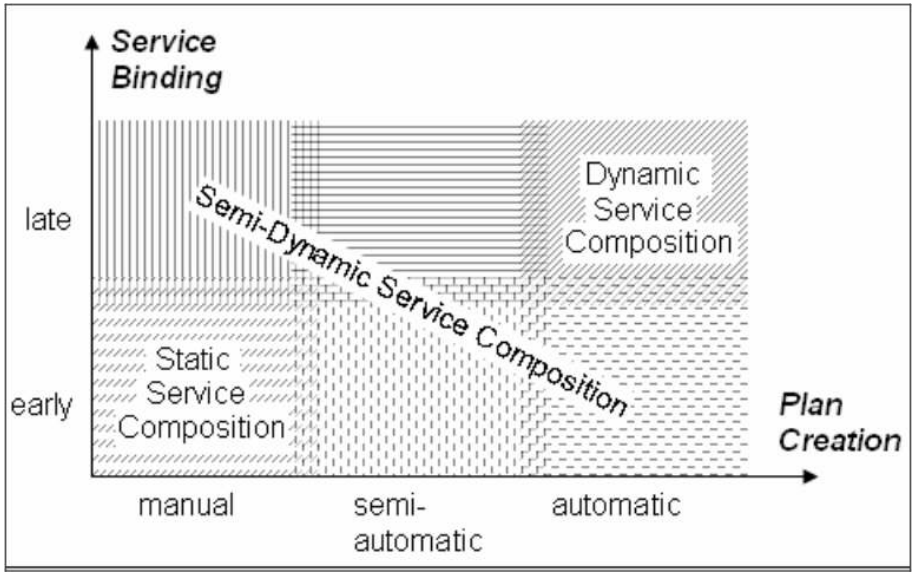


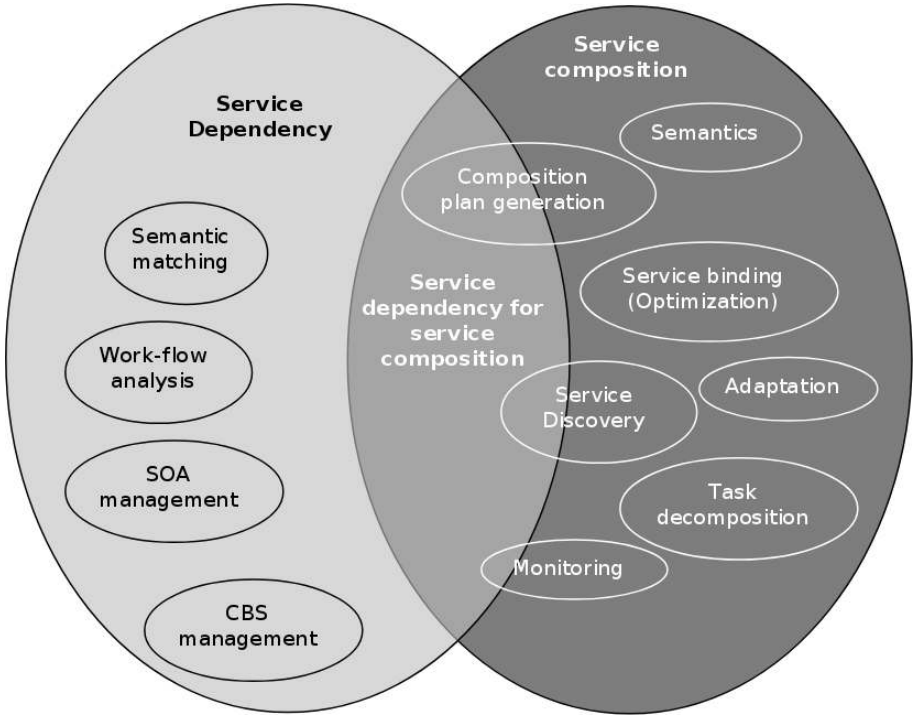
[mellekoning.nl]

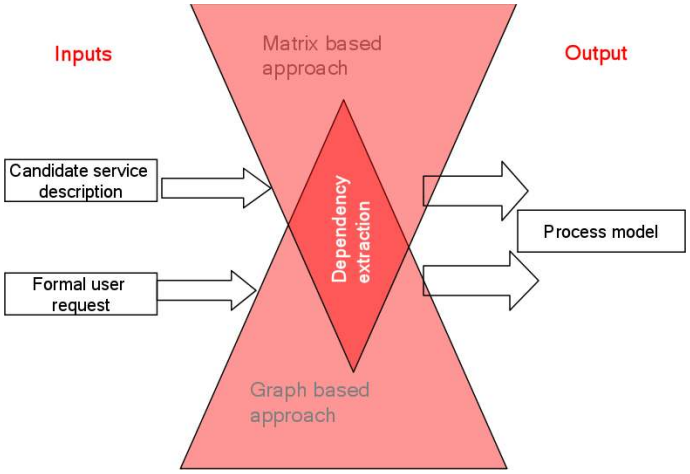
Dependency graphs

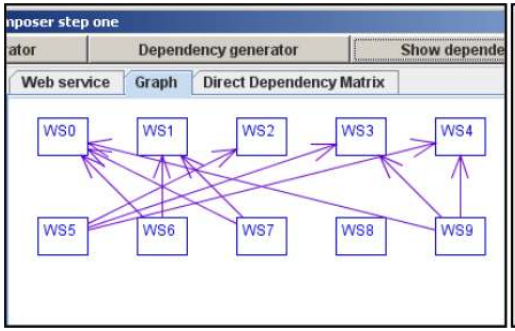
Service composition classification + automated creation

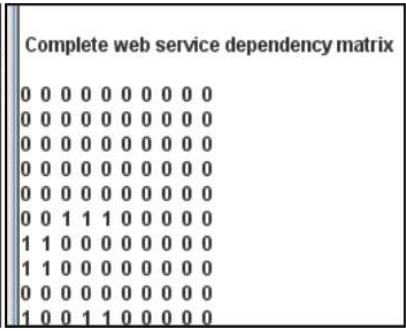
[Omer'11]







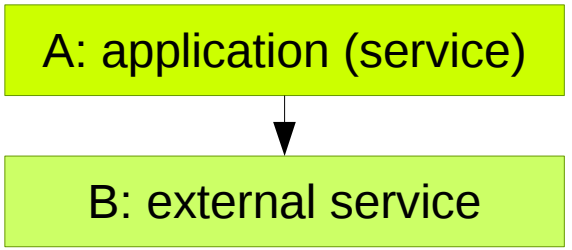




II.
Detection/Resolution

Dependency analysis

A depends on B if
- declared as such
- A invokes B (traceable)
- logic of A depends on state within B (not traceable)



Analysis techniques
- statically on description
- dynamically during execution

Dynamic analysis requirements
- correlation vs. causation (coincidental vs. true)
- small data sets (infrequent/conditional use)
- clustering of services (e.g. backup/failover)

Dependency analysis after Peddycord

[USENIX LISA'12]

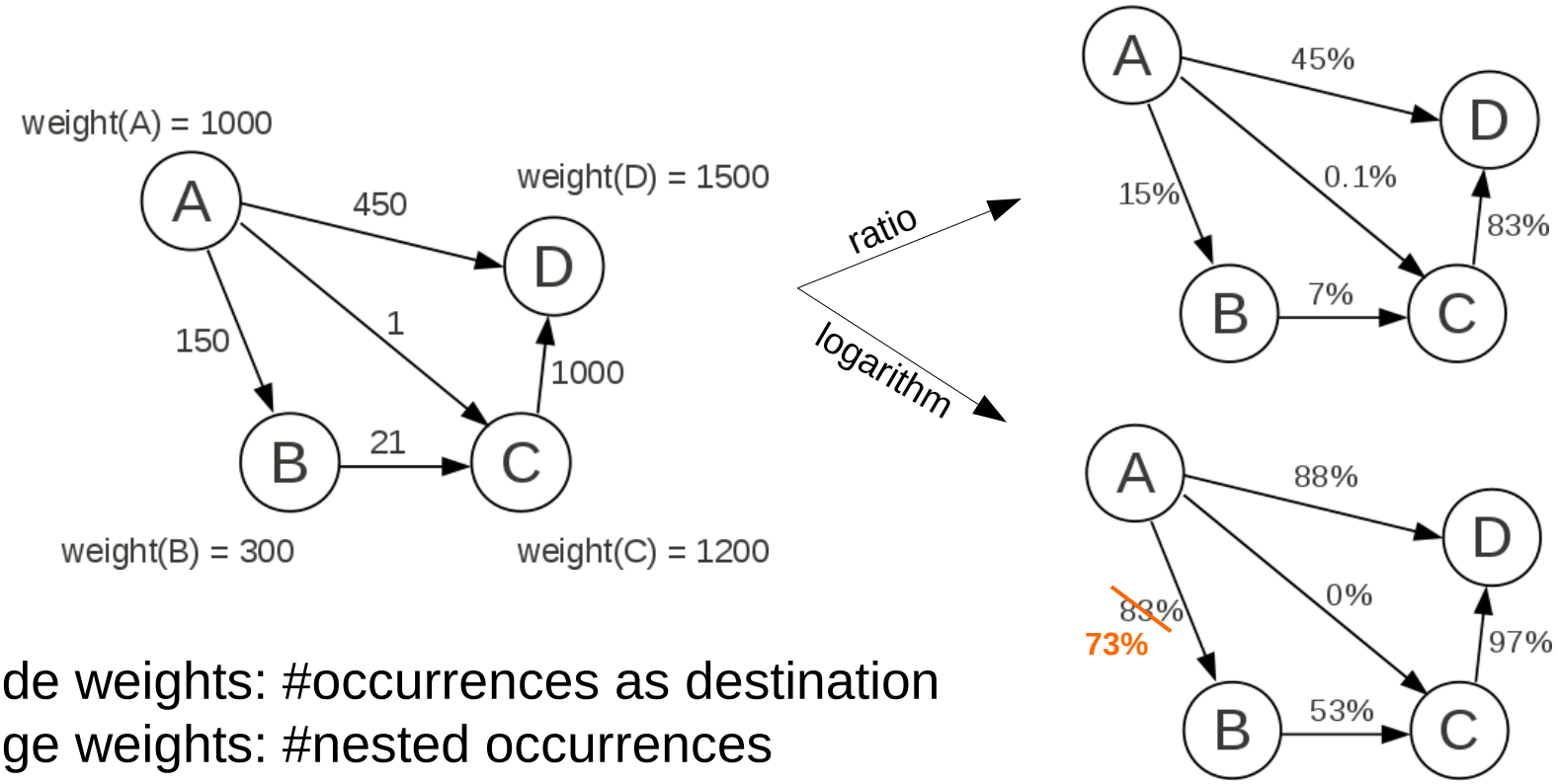
3 novel identification techniques

basis:

- passive network monitoring and analysis
- logarithm-based ranking scheme
- frequency inference

implementation: extended NSDMiner (NSD: Network Service Dependency)
model: communications graph
heuristic calculation: confidence in dependency candidates

Dependency analysis: example



node weights: #occurrences as destination
edge weights: #nested occurrences

confidence as ratio-based ranking: weight(A→B) / weight(A)
confidence as logarithm-based ranking: $\log_{\text{weight(A)}}$ weight(A→B)

Dependency resolution

Immediate resolution
- simple transitive inclusion
- solution not guaranteed

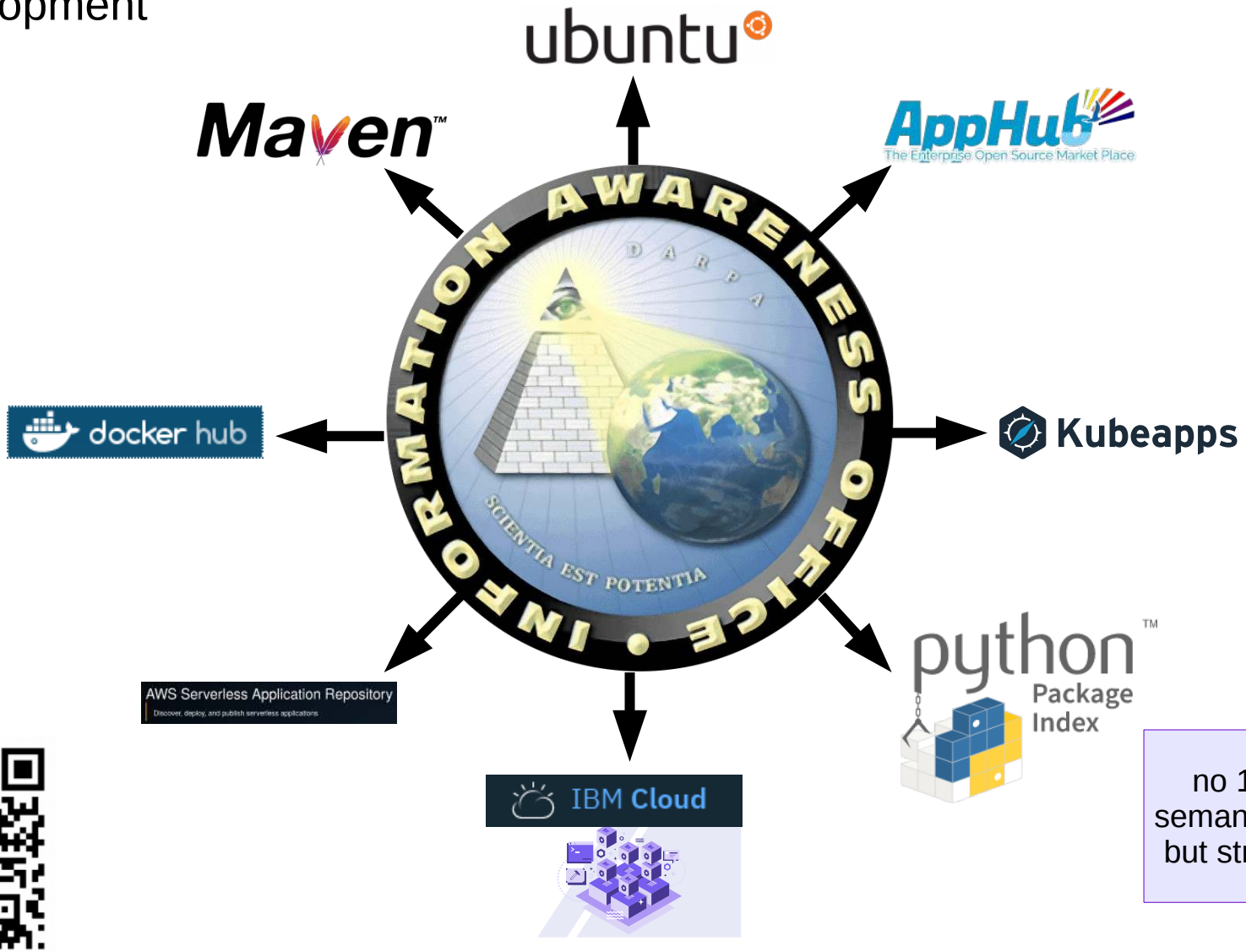
Interactive resolution
- with feedback
- iterative improvements
- may complement immediate resolution upon escalation

SAT (satisfiable) solver resolution
- a solution which exists will be found
- generation of proof for unsolvable problems
- NP-complete calculation

III.
Composite Applications

Practical service dependencies

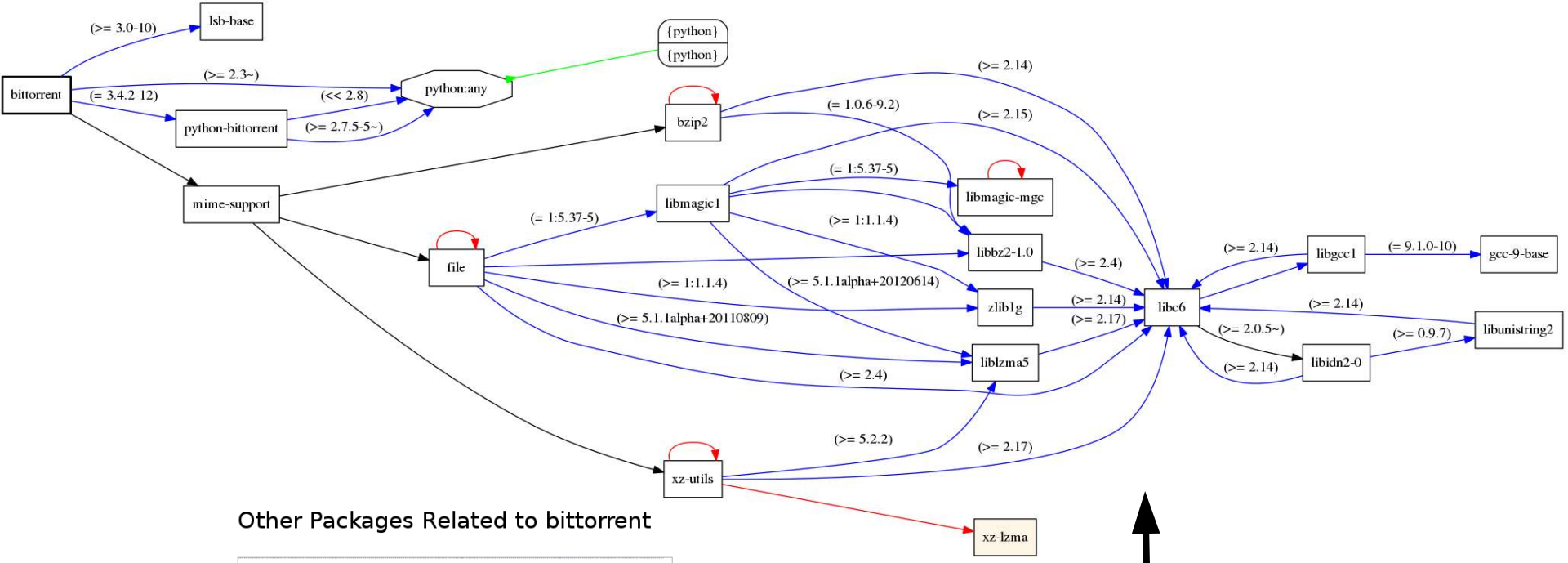
Artefact repositories: The backbone of today‘s (cloud) software development



no 1:1 service semantics mapping but strong relation

Basis: software dependencies

Linux distribution packaging example: Bittorrent



Other Packages Related to bittorrent

● depends ■ recommends ◆ suggests ▲ enhances

- ● **dep: lsb-base (>= 3.0-10)**
  Linux Standard Base init script functionality
- ● **dep: python**
  interactive high-level object-oriented language (Python2 version)
- ● **dep: python-bittorrent (= 3.4.2-12)**
  Scatter-gather network file transfer
- ■ **rec: mime-support**
  MIME files 'mime.types' & 'mailcap', and support programs
- ◆ **sug: bittorrent-gui**
  Original BitTorrent client - GUI tools

```
Build-Depends:          dark gold, bold
Build-Depends-Indep:    light gold
Pre-Depends:            purple, bold
Depends:                blue
Recommends:             black
Suggests:               black, dotted
Conflicts:              red
Provides:               green, inverted arrow
```

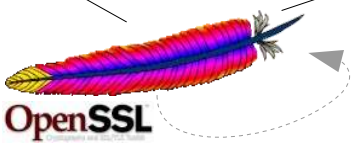
◯ not applicable to service dependencies

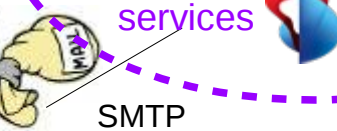
Practical service dependencies

Doodle example

HTTP GET/POST
improper REST

JSF scaffold
JSON data





3rd party services

SMTP

JSON documents

HTTP
ICS/
iCal

3rd party services

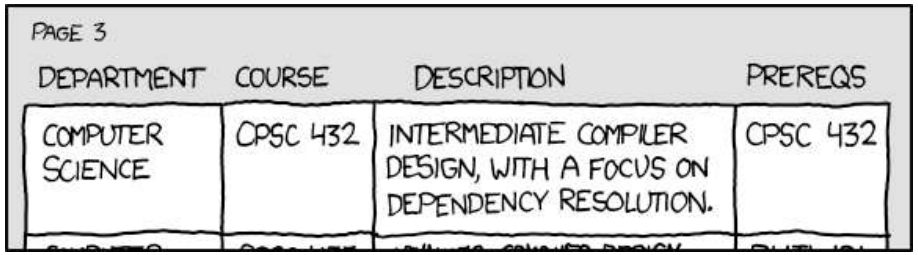
Practical service dependencies

Restlet-based applications: dependencies on .ext.atom, .ext.gae, .ext.sip etc.

- expressed in pom.xml files

Note: No generic standard notation available

Circular/recursive dependencies: should be avoided, but...


PAGE 3

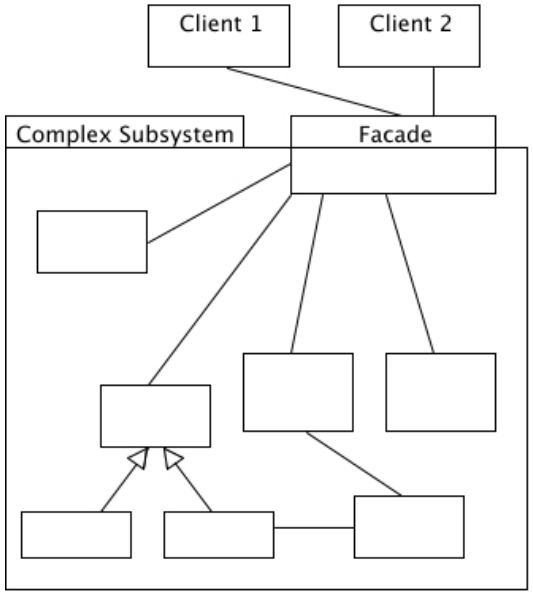
DEPARTMENT	COURSE	DESCRIPTION	PREREQS
COMPUTER SCIENCE	CPSC 432	INTERMEDIATE COMPILER DESIGN, WITH A FOCUS ON DEPENDENCY RESOLUTION.	CPSC 432

Composite services (code deps)

Characteristics
- offer a single service interface
- distribute requests to multiple services within the composition (in parallel, serially, or more complex routing)
- require knowledge of dependencies (internally or by the caller)

Remember software engineering – design patterns
- Composite / Façade
- Factory method

Advantages for services
- improved QoS – e.g. higher availability
- improved QoE – e.g. flexibility to switch



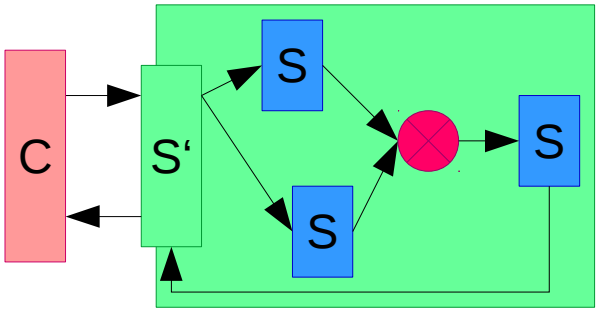
Composition techniques/patterns

- Orchestration
- Choreography
- Bundling
- Multiplexing
- Mashup (out of scope)
- Service Bus (out of scope)

Orchestration

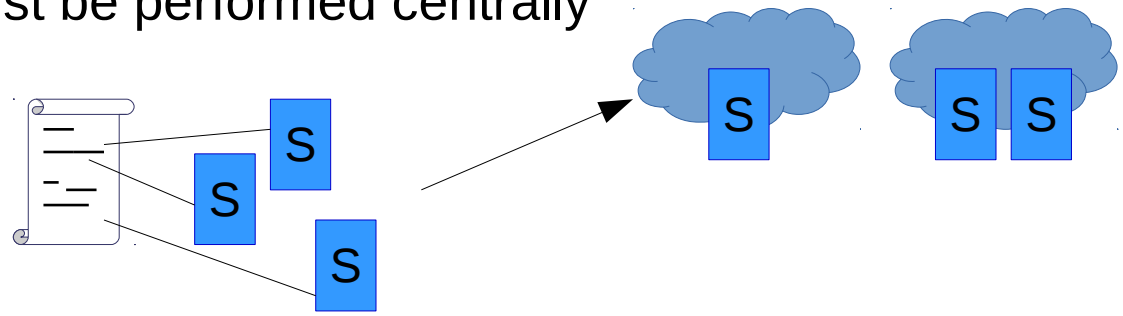
Service Orchestration:

- multiple services form another one
- centrally managed and executed
- based on workflows
- workflows require service interfaces



Resource Orchestration (e.g. for cloud computing):

- workflows require resources (e.g. software implementation of service)
- resource allocation must be performed centrally



Service orchestration

Definition: coordinated arrangement of service invocations; may be executable as another service

Potential benefits: creation of value-added services by re-using others
Potential risks: issues with dependency services (unavailable, faulty)

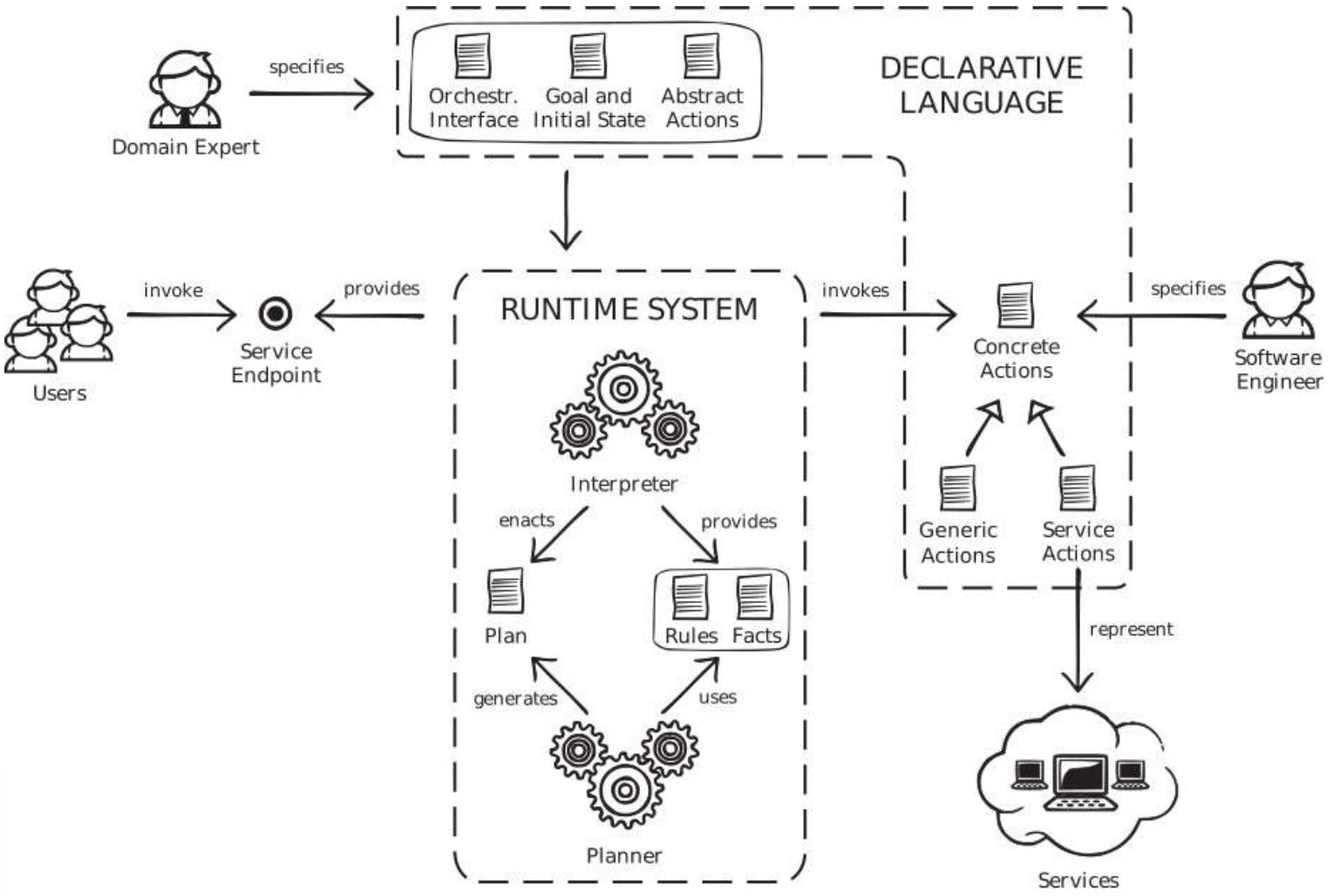
Example excerpt (in DSOL):

```
<services>
<service type='rest' name='poll' id='doodle' method='GET' url='.../#{pollId}' />
<service type='soap' name='translate' id='translate' operation='Detect' wsdl='...' />
</services>

@WebService
public interfacePollTranslator {
  @WebResult(name='translatedPoll')
  public Poll getTranslatedPoll(@WebParam(name='pollId') Stringid,
                                @WebParam(name='language') String desiredLanguage);
}

...
```

Orchestration with DSOL - example



Orchestration languages

Service Orchestration

- Business Process Execution Language (BPEL)
- Yet Another Workflow Language (YAWL)
- Dynamic Service Orchestration Language (DSOL)
- Workflows
  - Montage etc.
- Cloud function workflows
  - AWS Step Functions
  - Fission Workflows
  - IBM Composer

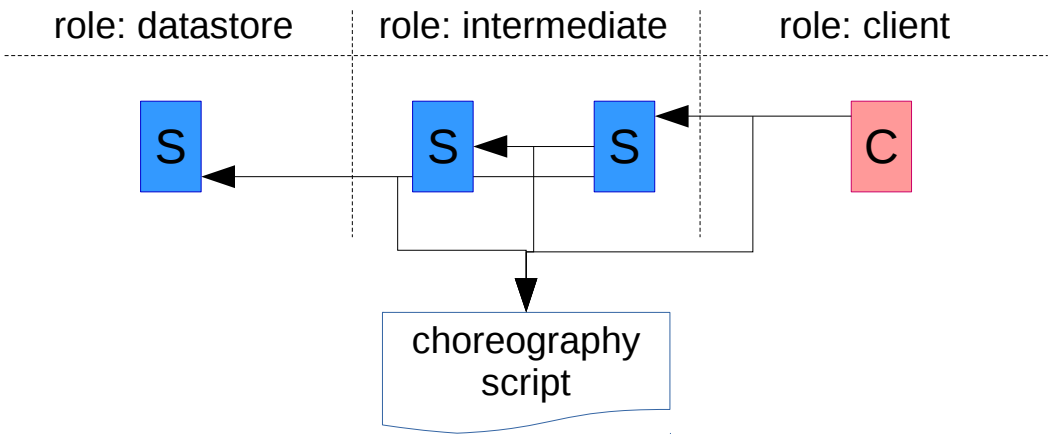
Resource/Implementation Orchestration

- Heat Orchestration Template (HOT)
- AWS CloudFormation
- Vamp Blueprints + Workflows
  - Docker Compose
  - Kubernetes Descriptors
  - Juju Charms

Choreography

Definition: global interaction protocol between autonomous service partners

Potential benefits: no central point of control; declarative messaging behaviour
Potential risks: difficult decentralised enactment; little industry acceptance



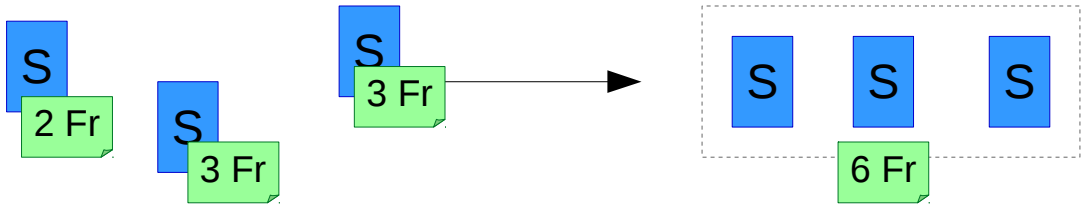
Languages: WS-CDL (historic), BPMN 2.0, Chor, CHOReVOLUTION

Bundling

Definition: multiple services offered/used in a bundle

(Mixed bundles: service access + tools, clients, other products)

Potential benefits: cheaper, less administrative overhead
Potential risks: less flexibility for exchanging single service



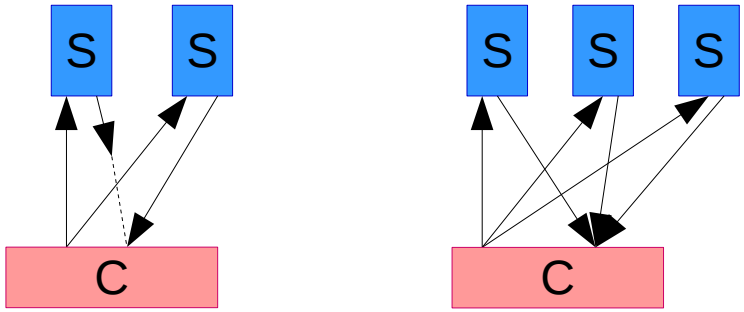
Languages: USDL

Multiplexing

Definition: multiple services used in parallel handling partial requests

(compare load balancing or failover: serial use)

Potential benefits: flexible redundancy schemes, “survival of the fittest“
Potential risks: more administrative overhead, higher cost due to candidates



Languages: none

Composite microservices

Definition: single-function-oriented services which scale elastically and operate resiliently backed by a portable implementation.

Inherited definition from arbitrary services:

- loosely coupled
- uniformly described and invoked
- composable
- re-usable

Implementations: containers, hosted functions, unikernels...

Microservices management platforms

Tasks
- deployment and management of microservice representations (i.e. containers)
- partial upgrades without downtimes, honouring dependencies
- canary testing
- monitoring, migration, scaling, ... without downtimes
- ... and: rapid prototyping!

Implementations (selection)
- Vamp
- Kubernetes / OpenShift
- Apache OpenWhisk
- provider-specific: AWS ECS, Azure CS, Google CE, IBM CS for Bluemix, ...

APACHE OpenWhisk™

vamp

Composition & management example

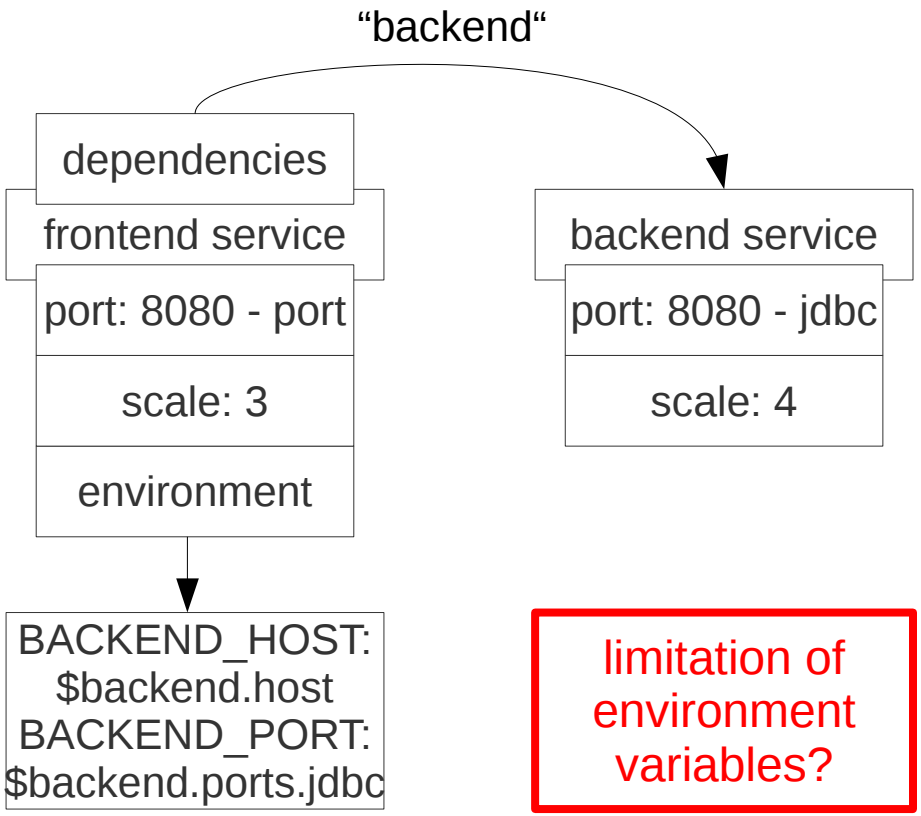
Vamp

- using “blueprints“
- ingredients: services, clusters, gateways, conditions, escalations, etc.

```
---
name: my_blueprint                          # Custom blueprint name
gateways:
  8080/http: my_frontend/port
clusters:
  my_frontend:                              # Custom cluster name.
    gateways:                               # Gateway for this cluster services.
      routes:                               # Makes sense only with
        some_cool_breed:                    # multiple services per cluster.
          weight: 95%
          condition: User-Agent = Chrome
        some_other_breed:                   # Second service.
          weight: 5%
    services:                               # List of services
      -
        breed:
          ref: some_cool_breed
        scale:                              # Scale for this service.
          cpu: 2                            # Number of CPUs per instance.
          memory: 2048MB                    # Memory per instance (MB/GB units).
          instances: 2                      # Number of instances
```

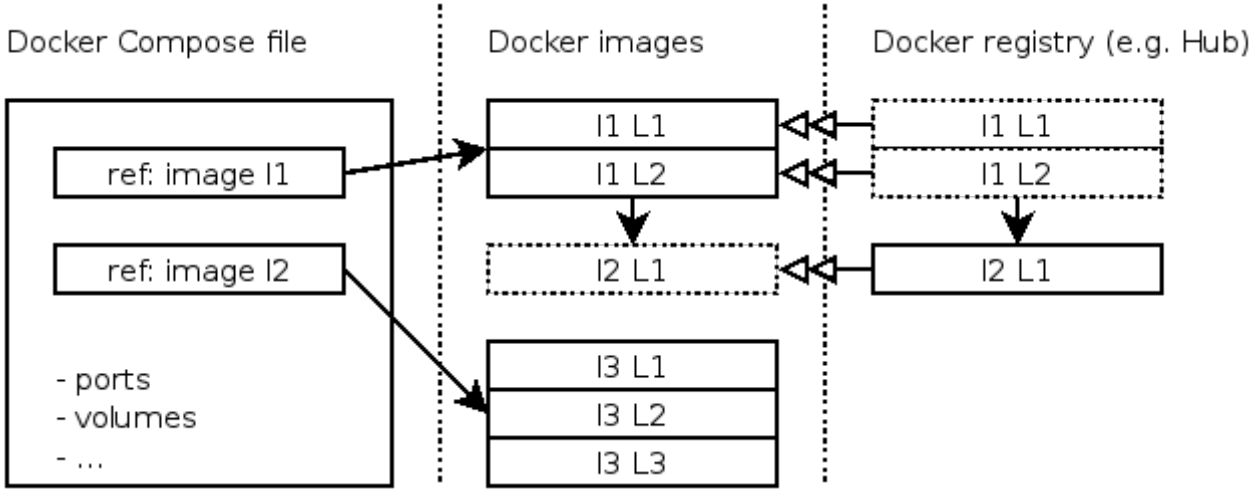
Service discovery example

Again, in Vamp:



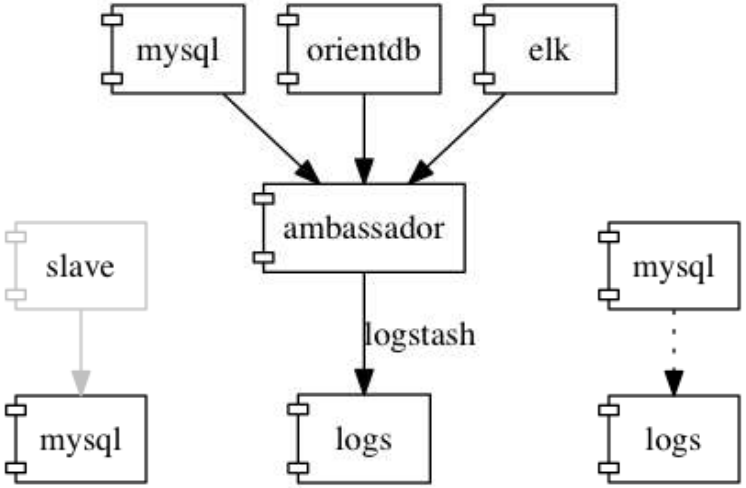
Dependencies in Docker Compose

Dependencies are *by reference* → need resolution before execution



Internal + external service dependencies: links + deps

Volume dependencies



Issues with deps in Docker Compose

- Docker images
  - do not (any longer) exist
  - are not accessible (i.e. registry requires login)
  - brings along defect or security vulnerability

- Volumes
  - do not not (any longer) exist
  - are not accessible (permission issues)
  - are full

- External services
  - are not accessible (networking level, authorisation)
  - are not yet ready → wait scripts like wait-for-it.sh

Dependencies in Helm charts

Dependencies are *by embedding* - no resolution needed - but: *transitivity*!

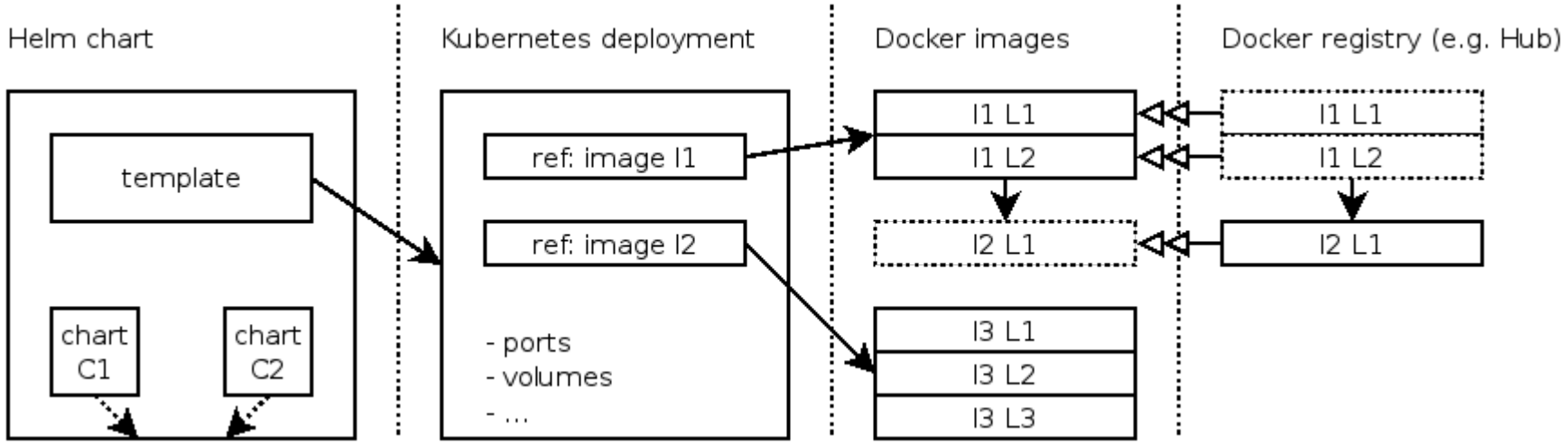
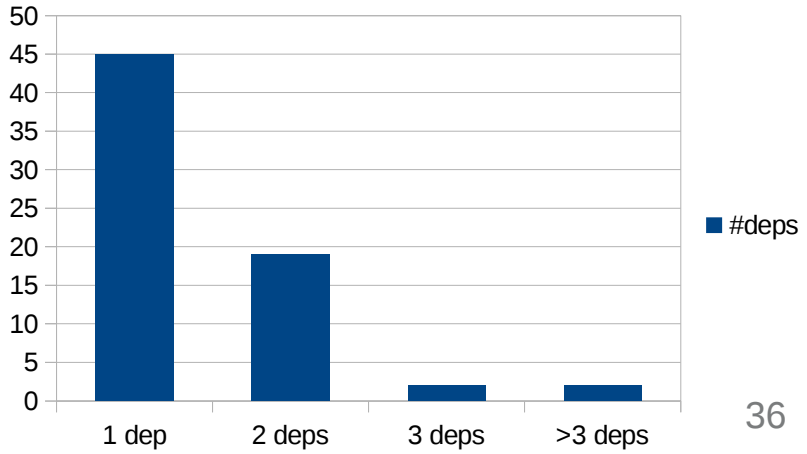


Chart structure →

```
chartname/Chart.yml
chartname/charts/dep1/Chart.yml
chartname/charts/dep2/Chart.yml
```

23% of all charts (on KubeApps Hub) have dependencies; distribution [Jul'19] →



Issues with deps in Helm charts

Issues:

- in 0.3% of all cases: incorrect dependencies (chartname/dep1/Chart.yml in cert-manager)
- among all charts with deps:
  - 23.5% have incompletely specified dependency versions (e.g. sugarcrm, quassel, sonarqube); among the remainder:
  - some are fine (e.g. elasticsearch-exporter: provides 1.0.2; elastic-stack requires 1.0.2)
  - some are not so fine (e.g. mongodb: provides 4.0.11; charts like rocketchat require 4.0.10, 4.0.9, 3.6.4) - totalling 37.2%

    → subject to bitrot, vulnerabilities, deprecations, failure to resolve docker images...

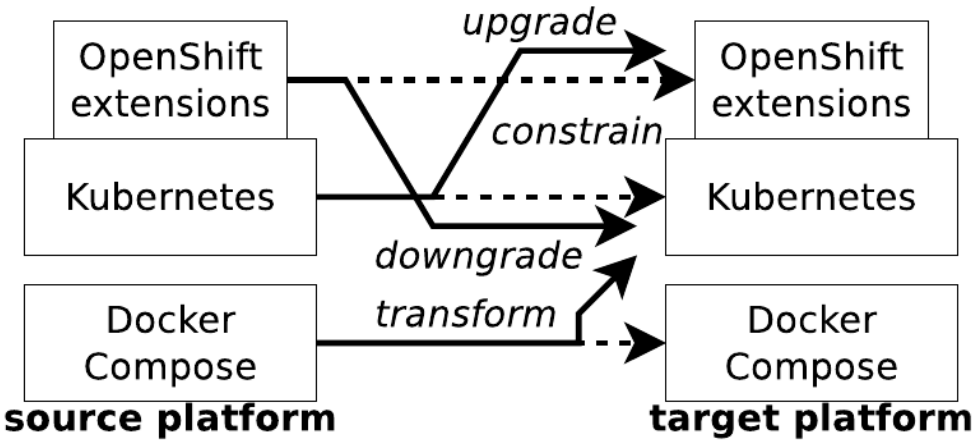
Issue score: 0.3% + 23%*23.5% + 76.5%*37.2% = **34.2%** of all Helm charts can‘t do deps

Application Portability

Portability and migration

Portability ← → dependency of application/service on platform
Migration ← → resolving dependency

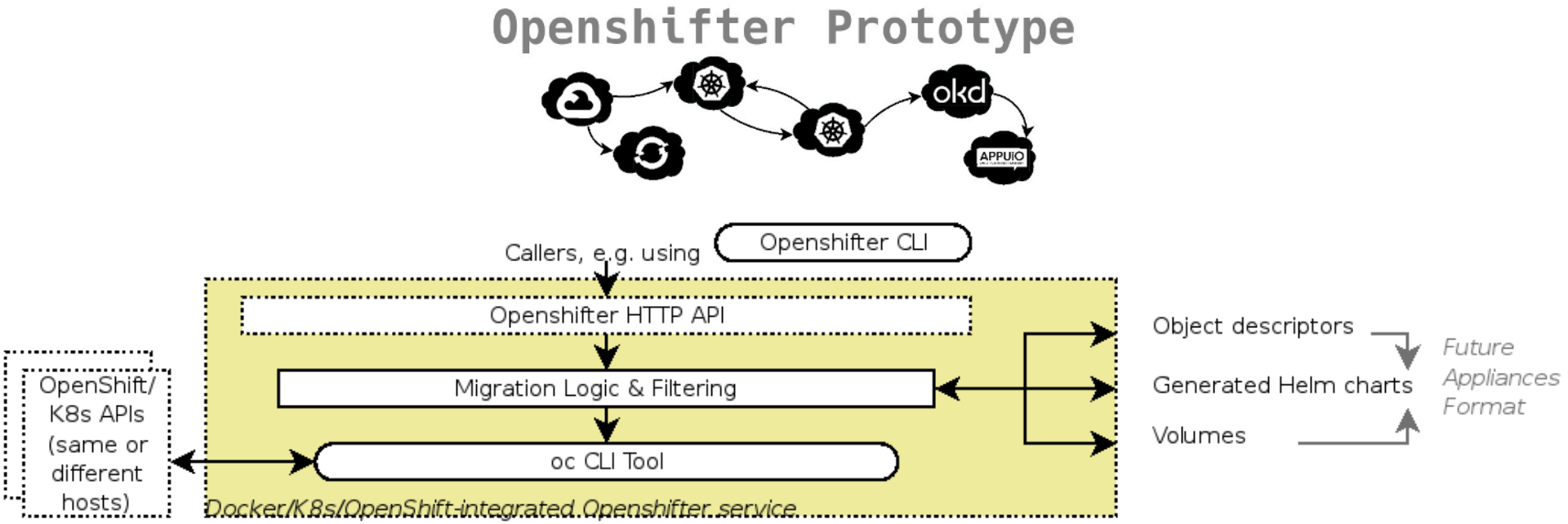
Complexity of migration
- homogeneous
- inhomogeneous
- heterogeneous



Types of microservice migrations
- onboarding: dev → prod
- region change: prod1 → prod1
- upgrades (e.g. storage): prod1 → prod1‘
- external backup: prod → backup

Scenario: OpenShifter

Migration of OpenShift application + volumes




```
curl https://localhost:8443/export/<baseURL>/<project>/<user>/<password> >
_output
base64 -d < _output> _output.tgz
```

```
curl https://localhost:8443/delete/<baseURL>/<project>/<user>/<password>
```

```
curl -X POST --data-urlencode @_output.tgz https://localhost:8443/import
/<baseURL>/<project>/<user>/<password>
```

Scenario: FaaS Converter

Migration of cloud functions

- entry point function signature differences
  - → faasconverter
- management API (control plane) differences
  - → snafu-import

AWS Lambda:

```
def lambda_handler(event, context):
    '''
    event: dict
    context: meta information obj
    returns: dict, string, int, ...
    '''
    # ...
    return "result"
```

OpenWhisk/IBM Functions:

```
def handler(input):
    '''
    input: dict
    returns: dict
    '''
    # ...
    return {}
```

OVH Functions:

```
def handler(input):
    '''
    input: dict
    returns: str
    '''
    # ...
    return ""
```

Fission:

```
def main():
    '''
    input: flask.request.get_data()
    returns: str
    '''
    # ...
    return "result"
```

Azure Functions:

```
def main():
    from AzureHTTPHelper \
    import HTTPHelper
    input = HTTPHelper().post
    # ...
    open(os.environ["res"], "w").\
    write(json.dumps({"body": "..."}))
main()
```

Further differences:

- function naming (mangling on client or service side)
- function granularity (number of entry points)

```
./faasconverter --file test.py --function foo --providers aws,  azure --just-wrap False --all-together True
```

Options

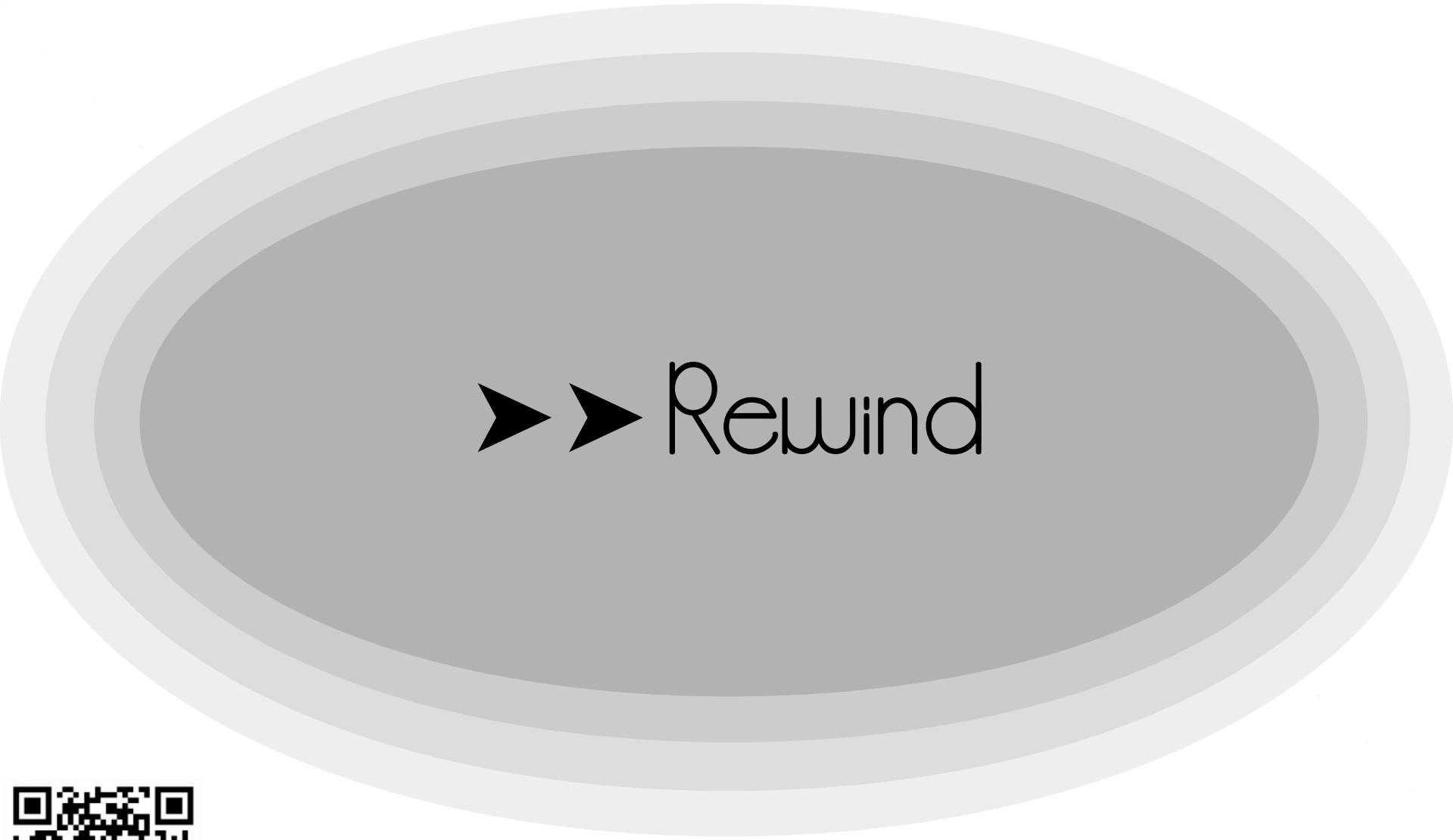
file File to convert to the selected provider sintax

function Selected function to convert to the providers sintax

just-wrap To only add the sintax wrapper on the end of the selected file

providers List of selected providers to which to convert the functions, available = aws, ibm, azure, ovh, fission

all-together Put all the wrappers together and add it to the the end of each file


Rewind


SPLAB

Summary

Dependency lifecycle & representation

- extraction/generation/manual specification + analysis/identification + resolution + optimisation/adaptation
- graph, matrix

Complexity of dependency considerations

- services vs. software artefacts in service-oriented compositions
- variety: static vs. dynamic, visible vs. hidden, strong vs. weak

Duality with compositional models

- orchestration, choreography, bundling, multiplexing
- applicability to composite microservices (& artefacts)