Technologies and Mindsets: Trends in Cloud-Native Applications

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

August 11, 2017 | Universidad Nacional de Asunción

Mi ~vinculación a la UNA...

material exclusivo

Ciudad del Este 2004

Photography Mindset

2000 2004 2017

film roll
=> 36 photos

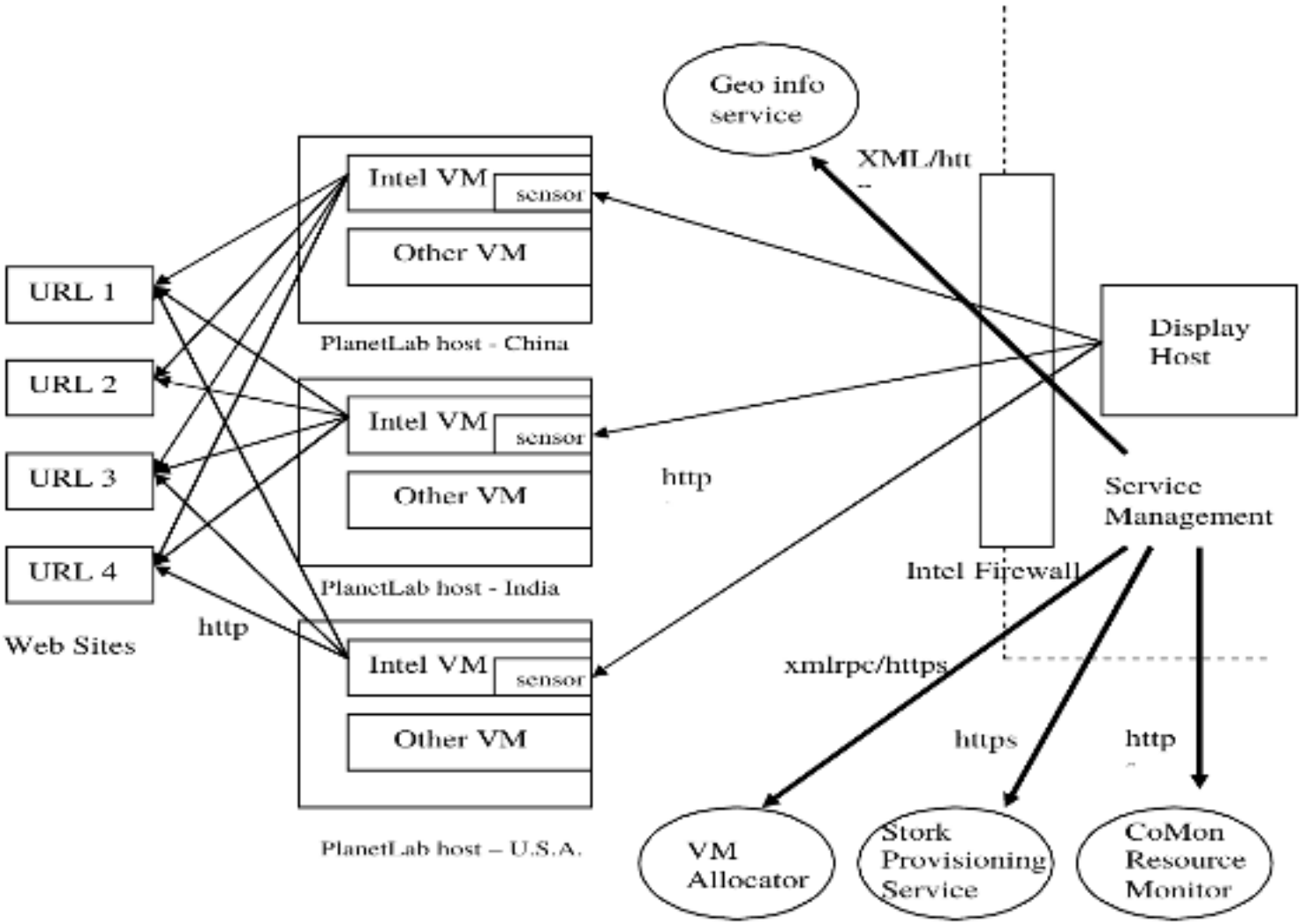
memory card 16 - 128 MB
ca. 1 up to 2 MB per picture (JPG)
=> up to 100 photos

memory card 32 GB
ca. 5.5 MB per picture
=> up to 6000 photos

2 innovations:
- technology: analog photography → digital
- mindset: scarce resource → disposable

ICCLAB SPLAB

Cloud Applications 2006



[Jeff Sedayao @ iiWAS'08]

Cloud Applications Mindset

2000 2006 2017

physical servers

programmable infrastructure with virtual machines

light-weight containers

2 innovations:
- technology: raw server access → fully managed infrastructure & platforms
- mindset: scarce resource → disposable (again!)

Observations

Evolving technology enables changing mindset.

Design for resource cornucopia and scale. (Cheap & plenty)

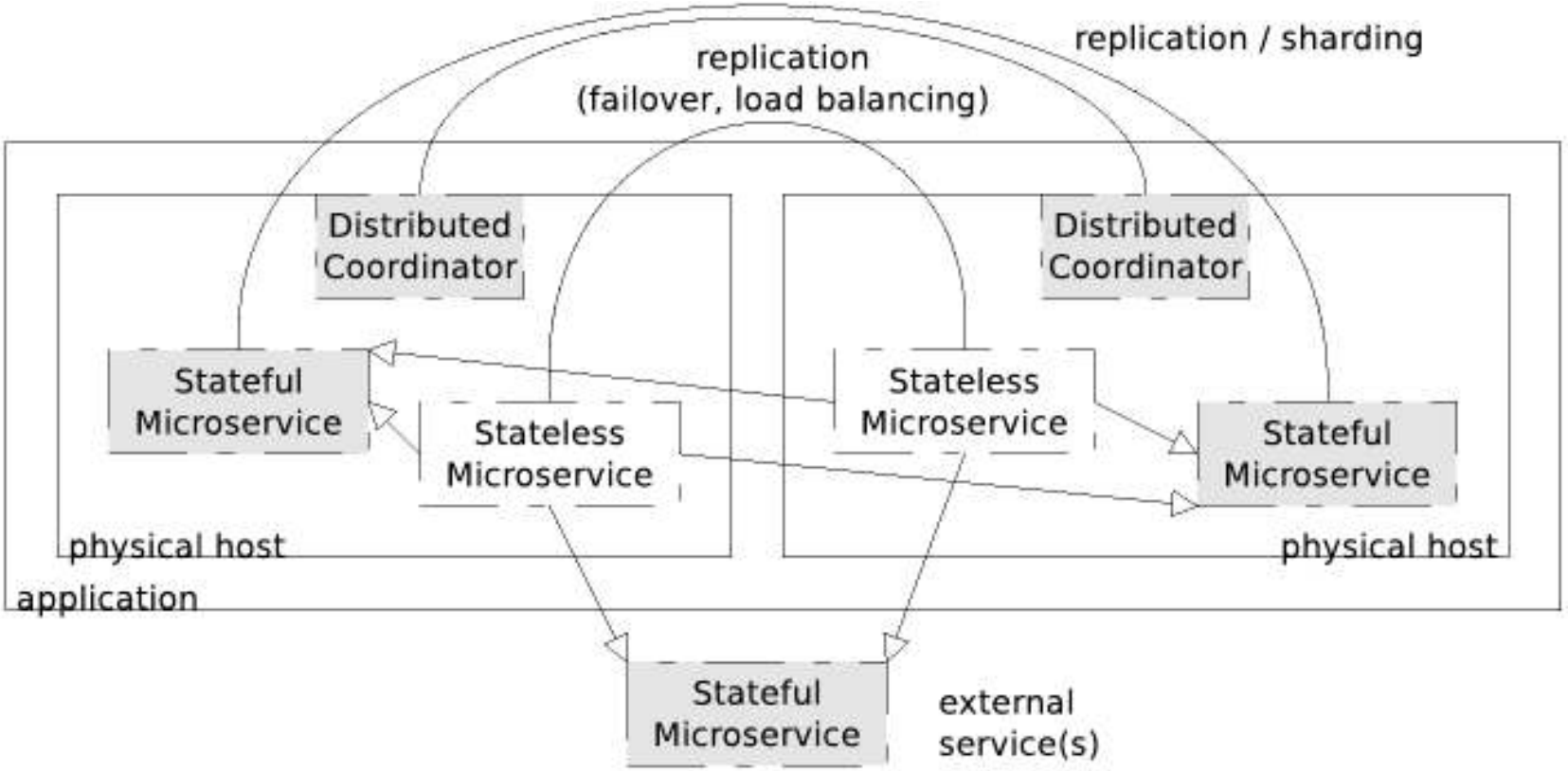
Design for failure. (Other risks: leaks, delays)

Design for portability and adaptivity.

Focus on cloud-native applications.

Suitable designs

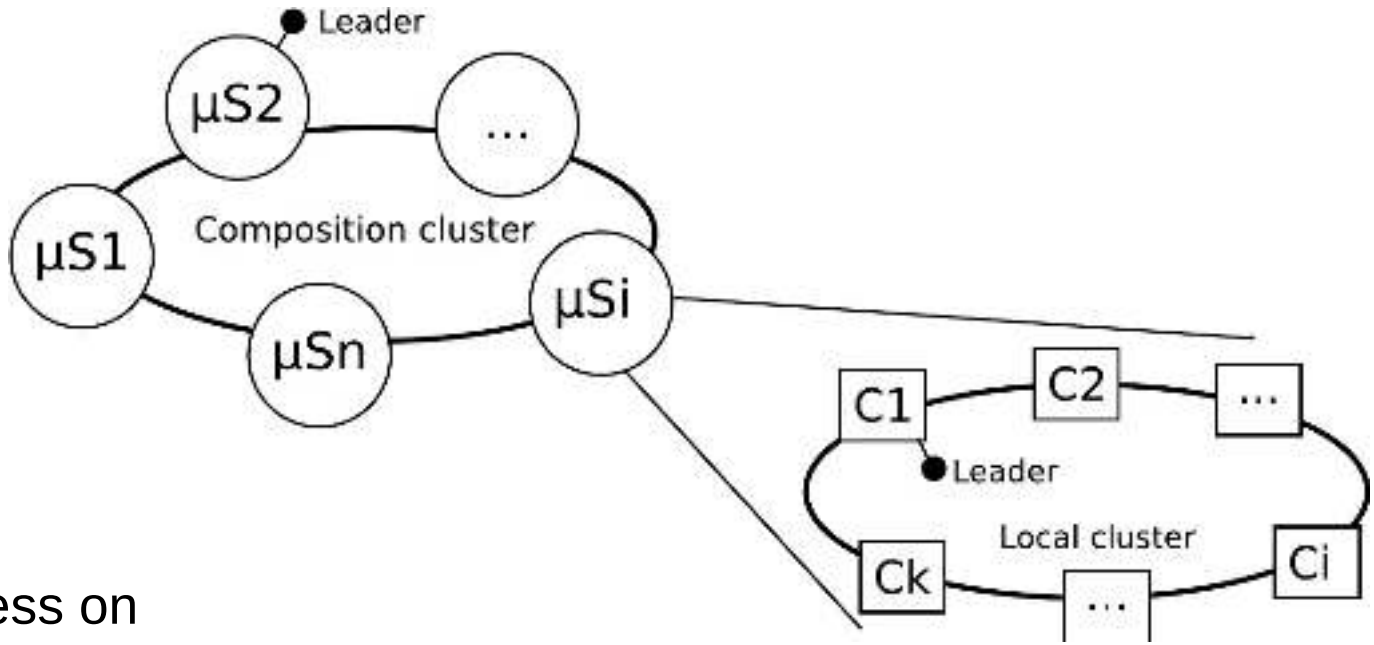
Microservice compositions



Suitable designs

Self-managed microservice compositions

- stateless microservices → horizontal scalability
- self management → resilience

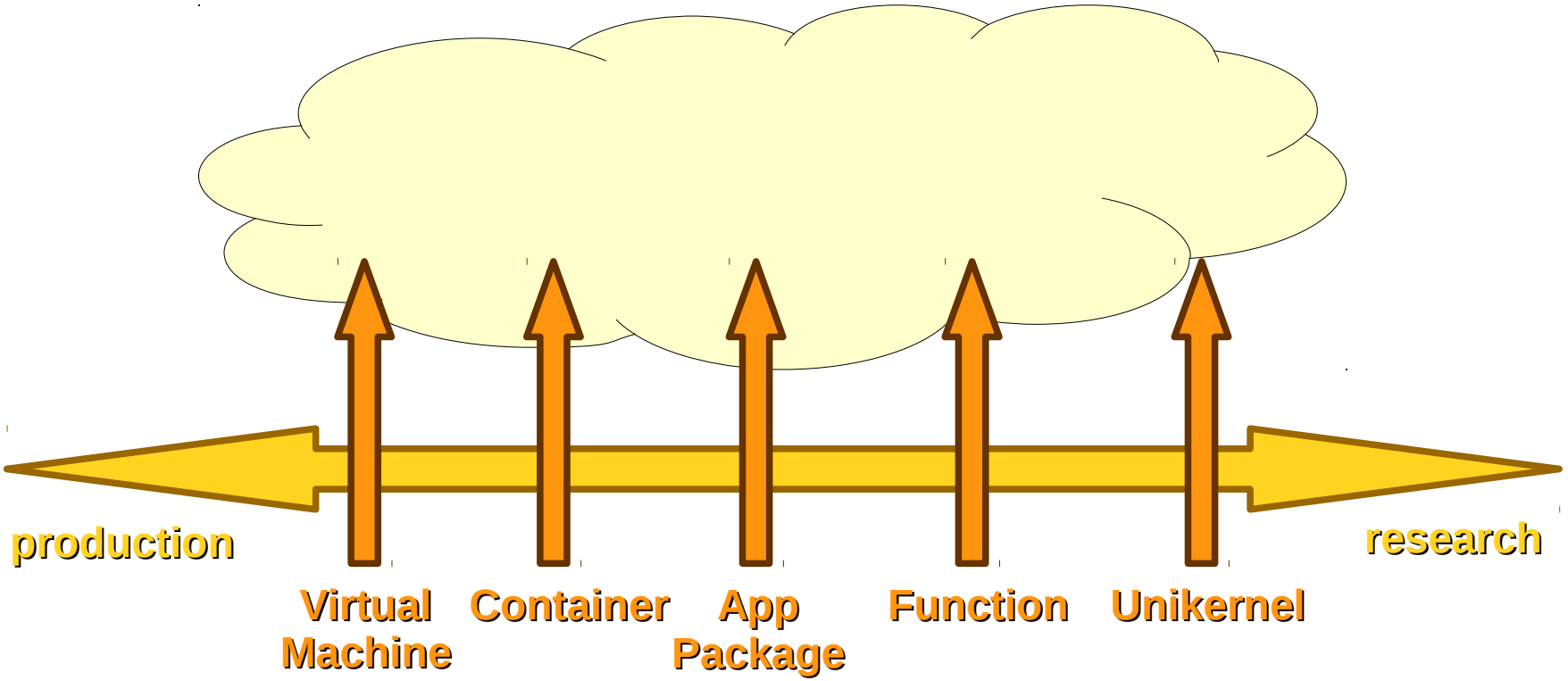


Progress on

- design patterns
- composition patterns

Suitable technologies

All (five) ways lead into the cloud...



Cloud-native affine: containers, functions (horizontal scalability)

Suitable technologies

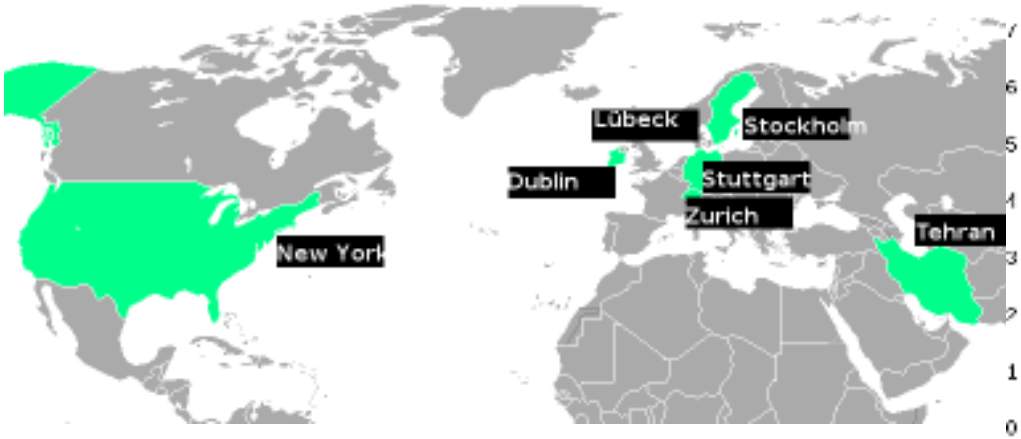
Container technologies (image formats, image production tools, runtimes)

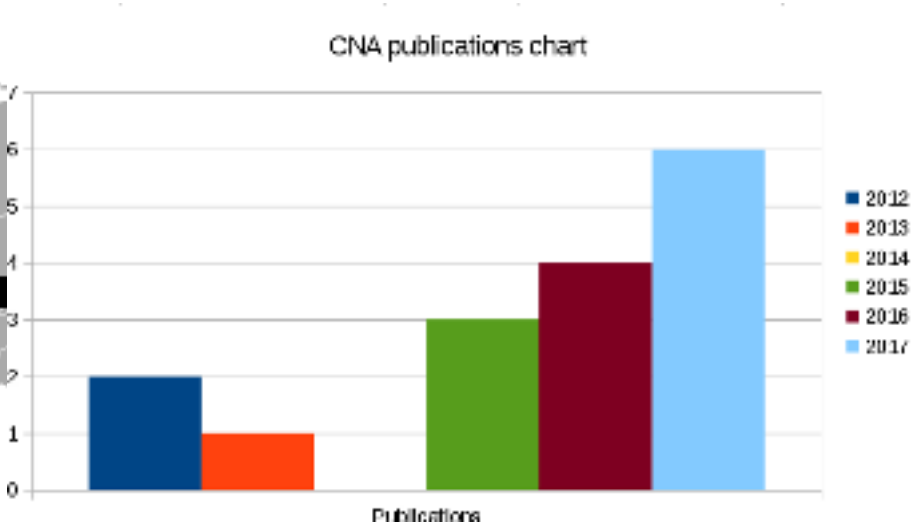
- Docker ← talk mostly refers to these
- Rkt
- LXC
- OCI

Function (“serverless“) technologies

- too many to mention here

Cloud-native applications challenges



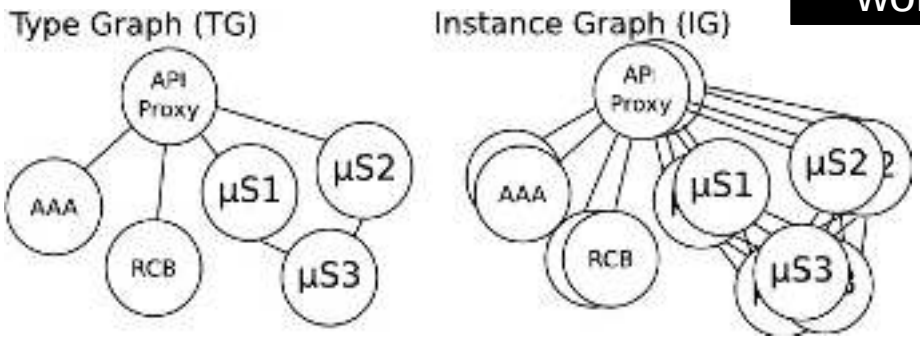


Important research topics
- rightsizing
- self management
- stateful services
- stealth properties

Rightsizing cloud applications / 1

Deployment into constrained environments

Application:



workstation

server/platform B

restricted environment

Artefacts:
- containers
- composition description
- deployment description

server/platform A

Rightsizing cloud applications / 2

Platform B restrictions: Kubernetes example

Resource type	Used	Max
CPU (Limit)	0 cores	2 cores
Memory (Limit)	0	3 GiB
Pods	0	15

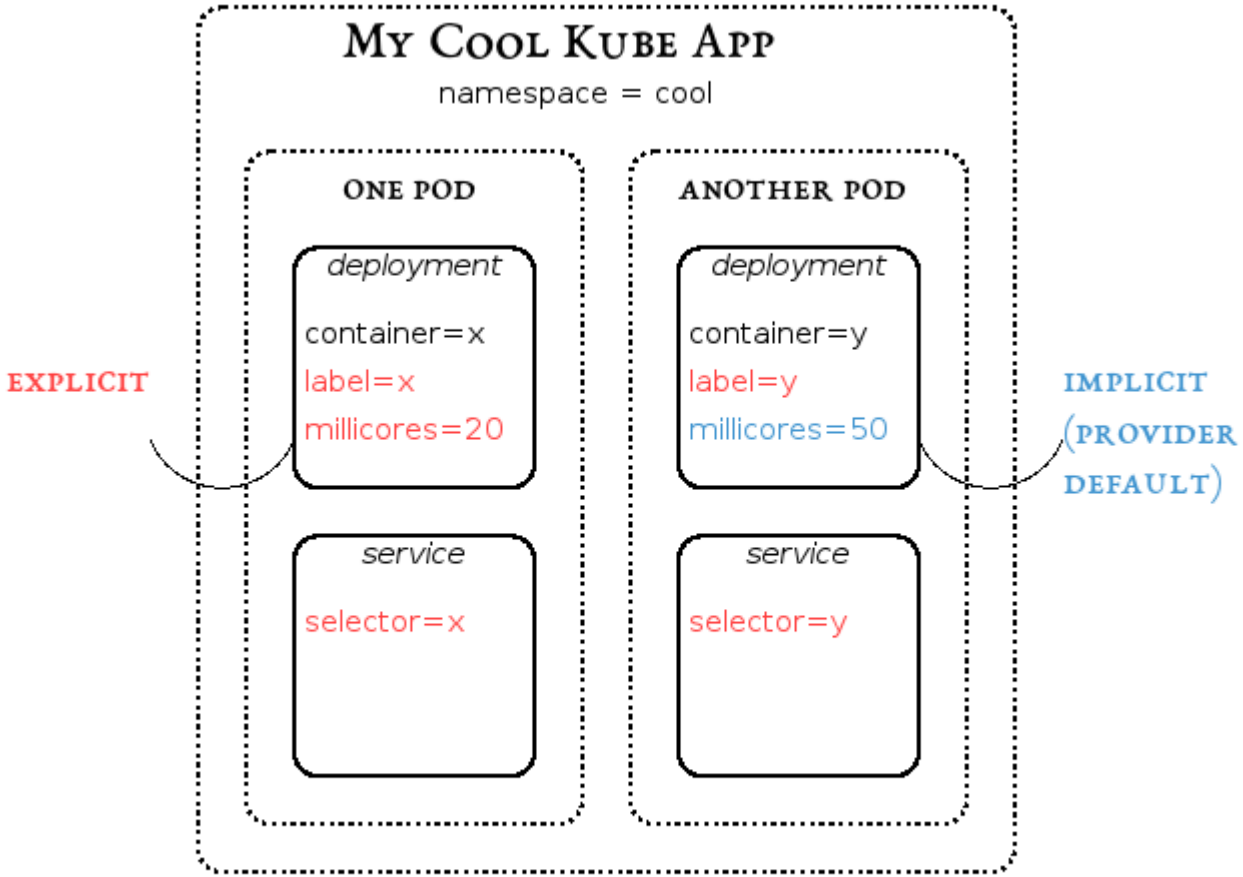
Resource type	Used	Max
Config Maps	0	10
Persistent Volume Claims	1	5
Replication Controllers	0	50
Secrets	9	30
Services	0	20

Resource type	Min ⓘ	Max ⓘ	Default Request ⓘ	Default Limit ⓘ	Max Limit/Request Ratio ⓘ
Pod CPU	50 millicores	1 core	—	—	—
Pod Memory	50 MiB	2 GiB	—	—	—
Container CPU	50 millicores	1 core	100 millicores	500 millicores	—
Container Memory	50 MiB	2 GiB	100 MiB	512 MiB	—

ICCLAB SPLAB Service Engineering Research Area

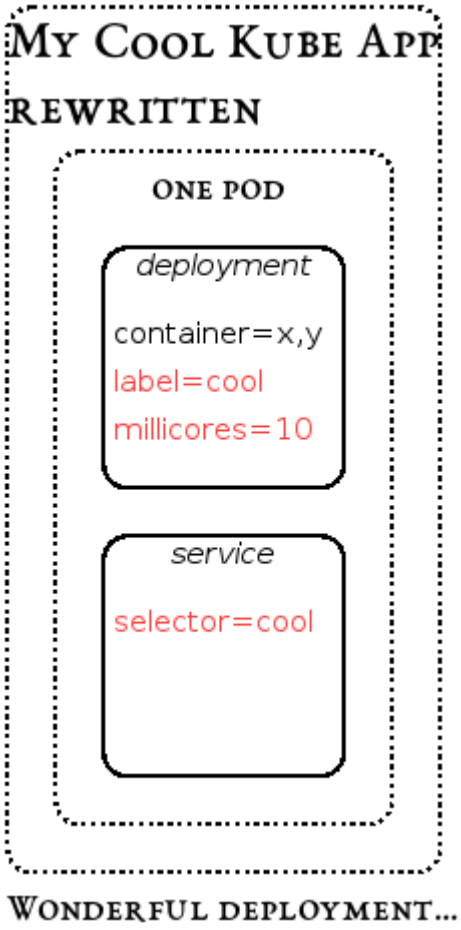
Rightsizing cloud applications / 3

Application analysis via deployment description



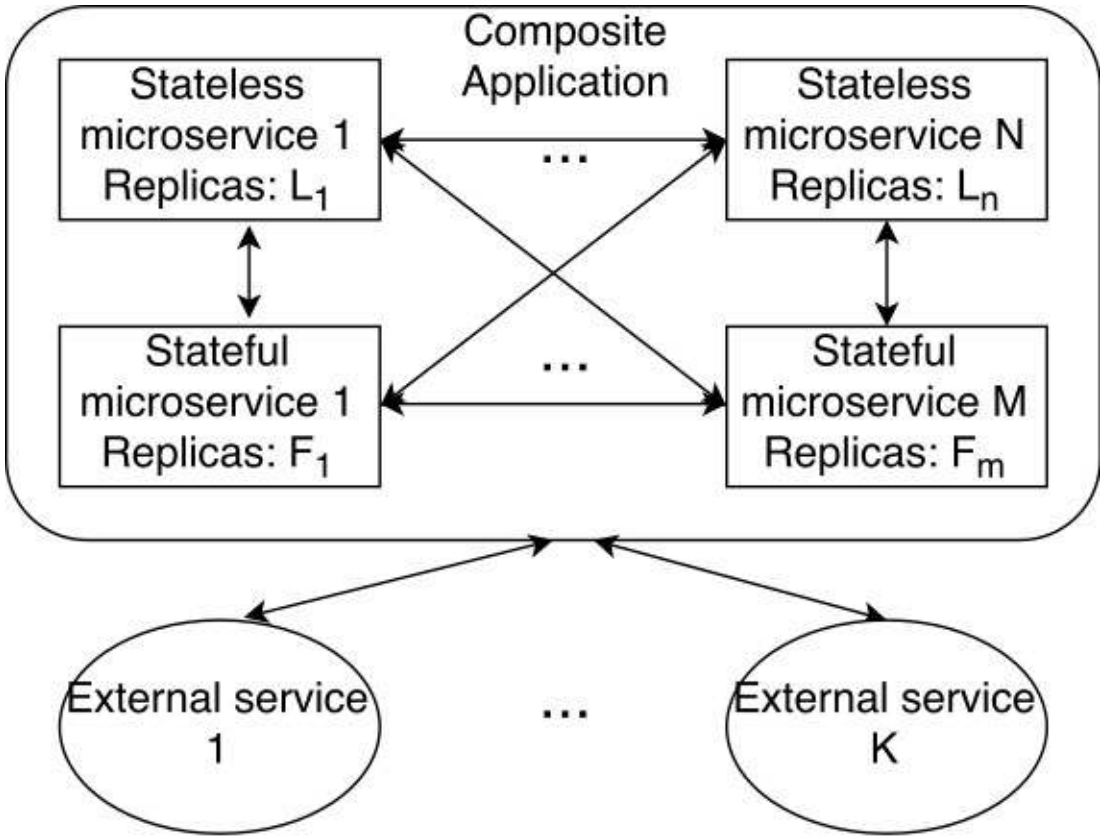
Rightsizing cloud applications / 4

Automatic rewriting to fit into constraints
(trade-off: less scalable or manageable)



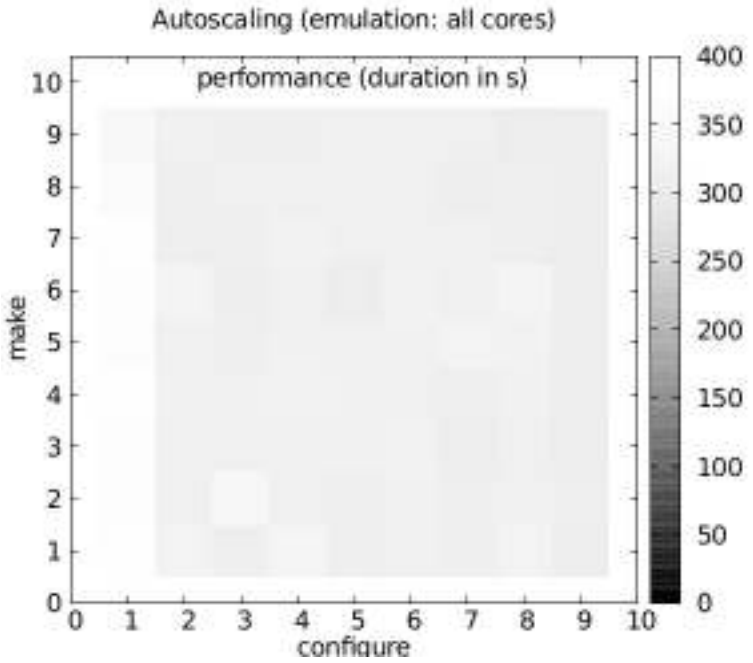
Rightsizing cloud applications / 5

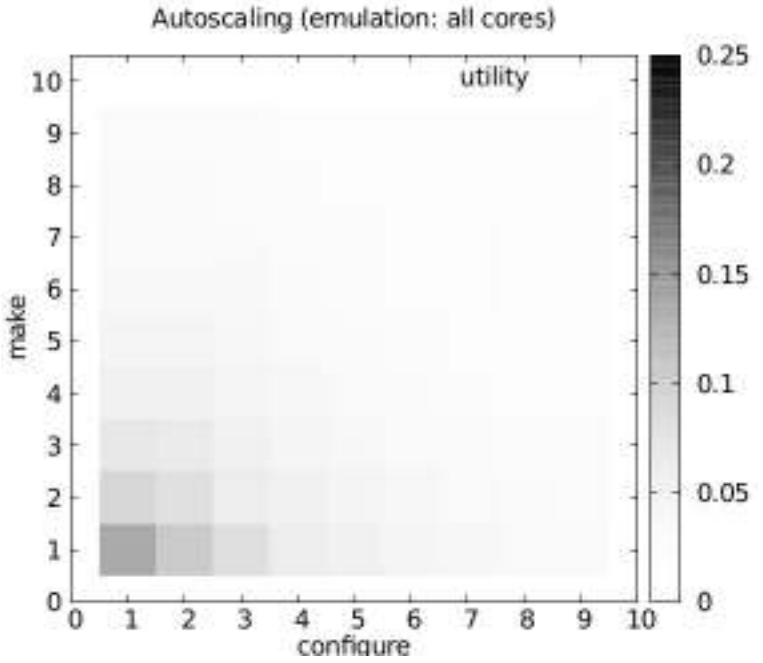
Assumption: n-replicas per microservice. How to determine n?



Rightsizing cloud applications / 6

Autoscaling behaviour in contemporary platforms

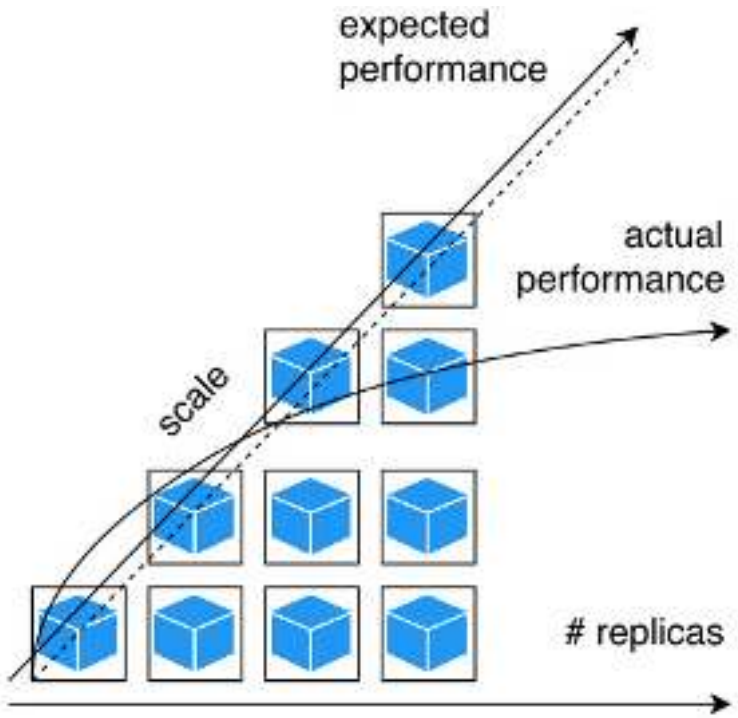




Conclusion: Autoscaling is not sufficient.

Rightsizing cloud applications / 7

Simplified scaling graph



Considerations:
- stateful vs. stateless microservices
- desired policy: fastest, cheapest, acceptable rate compromise

Rightsizing cloud applications / 8

Measured combinatorial scaling depending on desired policy (using „scalable-containers“ reference application)

Title	Policy	max_μ	max_κ	Rate	#S-ful	#S-less	Cost	Makespan
baseline	fastest	X	X	X	2	7	0.83	35.92
baseline	cheapest	X	X	X	1	1	0.33	89.16
with C	fastest	45.0	0.8	X	2	5	0.75	40.07
with C	cheapest	45.0	0.8	X	1	5	0.5	43.79
with C&R	fastest	45.0	0.8	1.06	1	7	0.58	41.88
with C&R	cheapest	45.0	0.8	1.2	1	7	0.58	41.88
Legend: S-less = stateless, S-ful = stateful, C = constraints, R = rate								

Self-managing applications / 1

Self management: fulfilment of requirements (scalability, resilience) independent of surrounding platform capabilities.

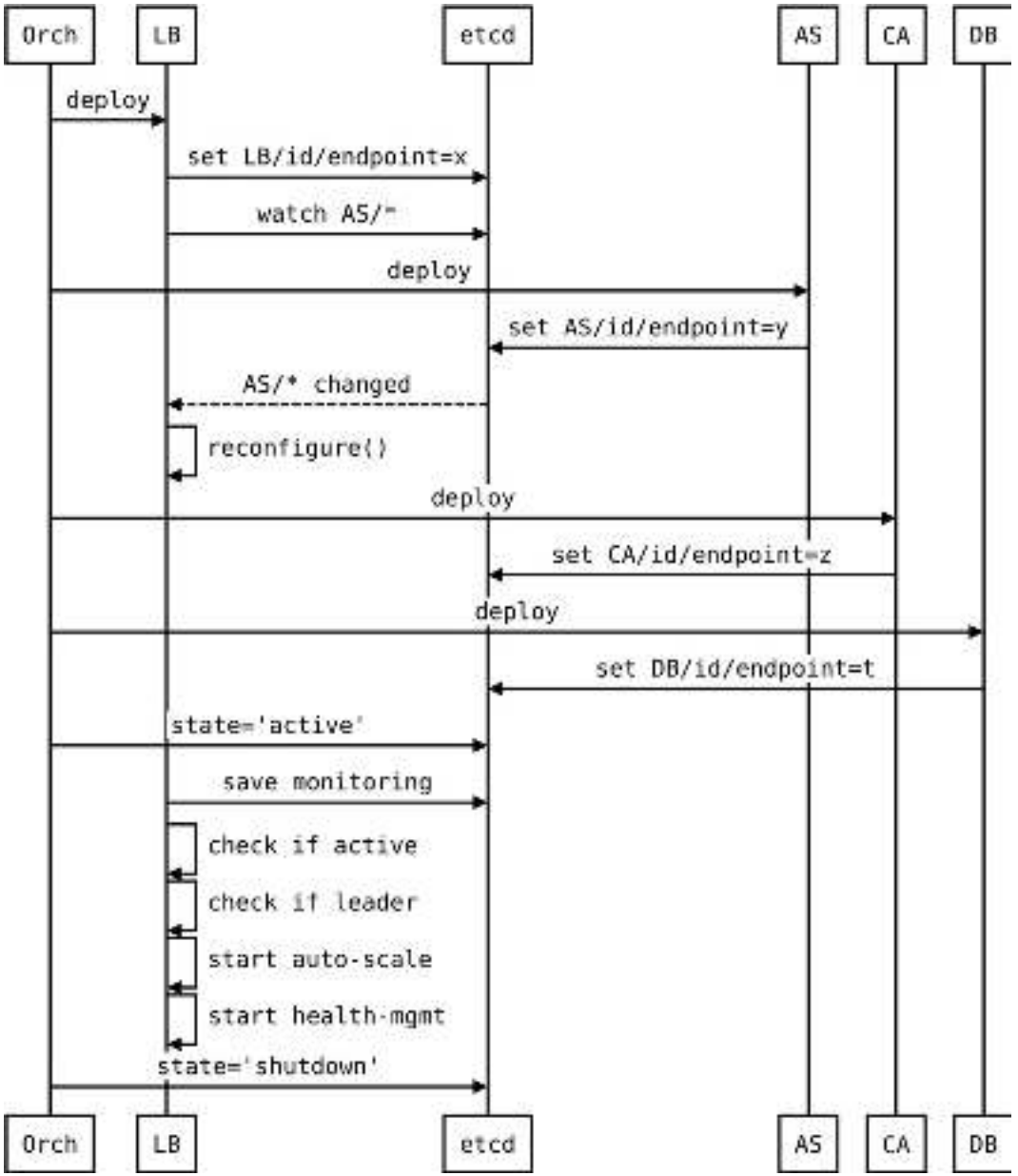
Distributed applications for management functionalities
- health management
- autoscaling
- adaptive service placement

Architectural basis: distributed in-memory key-value stores (for coordination)
- Consul, Zookeeper, Etcd, Dynamo, Pahoehoe
- stateful microservices using consensus algorithms
- differences in strong/eventual consistency guarantees

Self-managing applications / 2

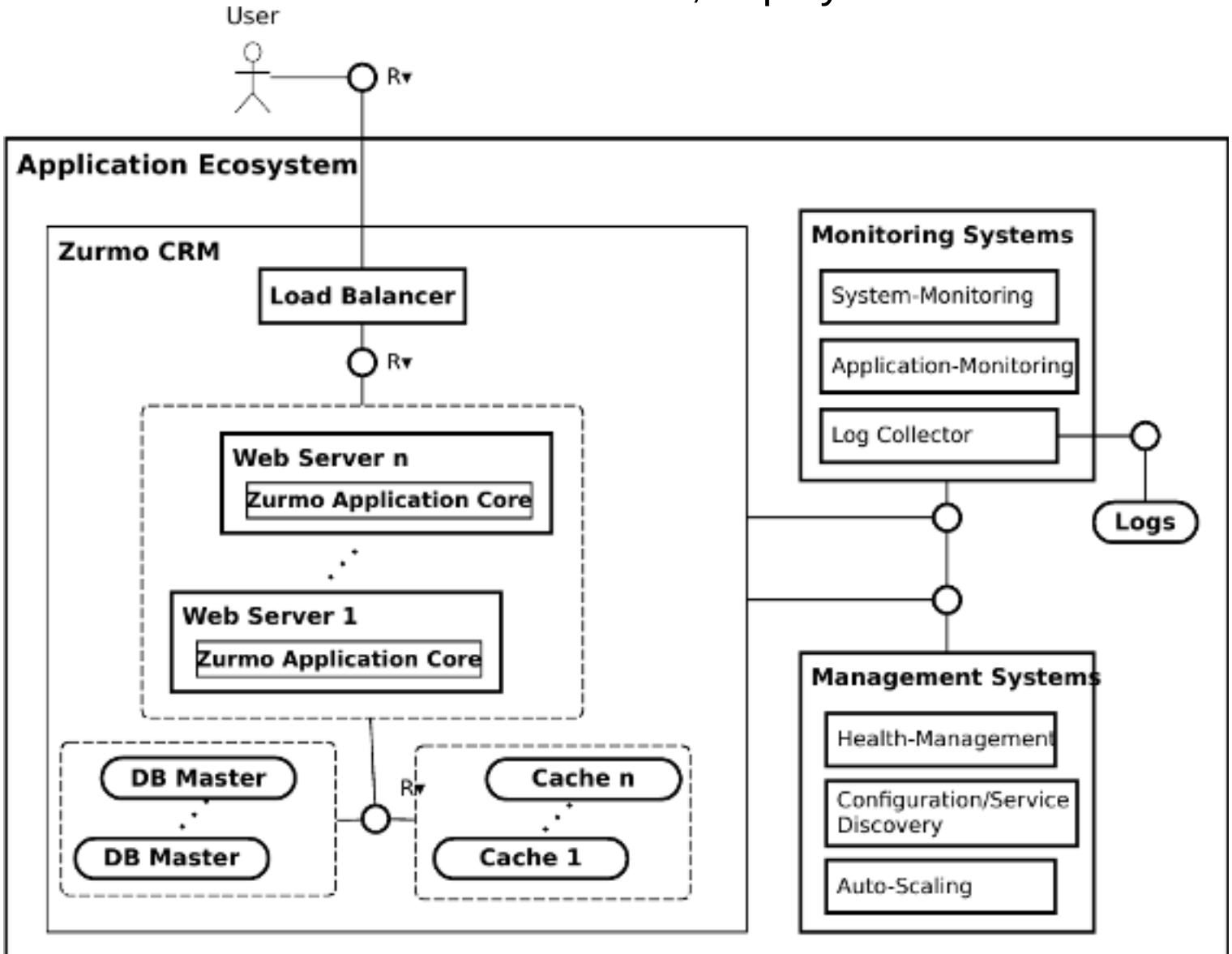
Instantiation/
deinstantiation
sequence
diagram for
caching (CA) and
application server (AS)
connected to database (DB)

(Orch: orchestrator;
LB: load balancer;
etcd: coordination kv-store)



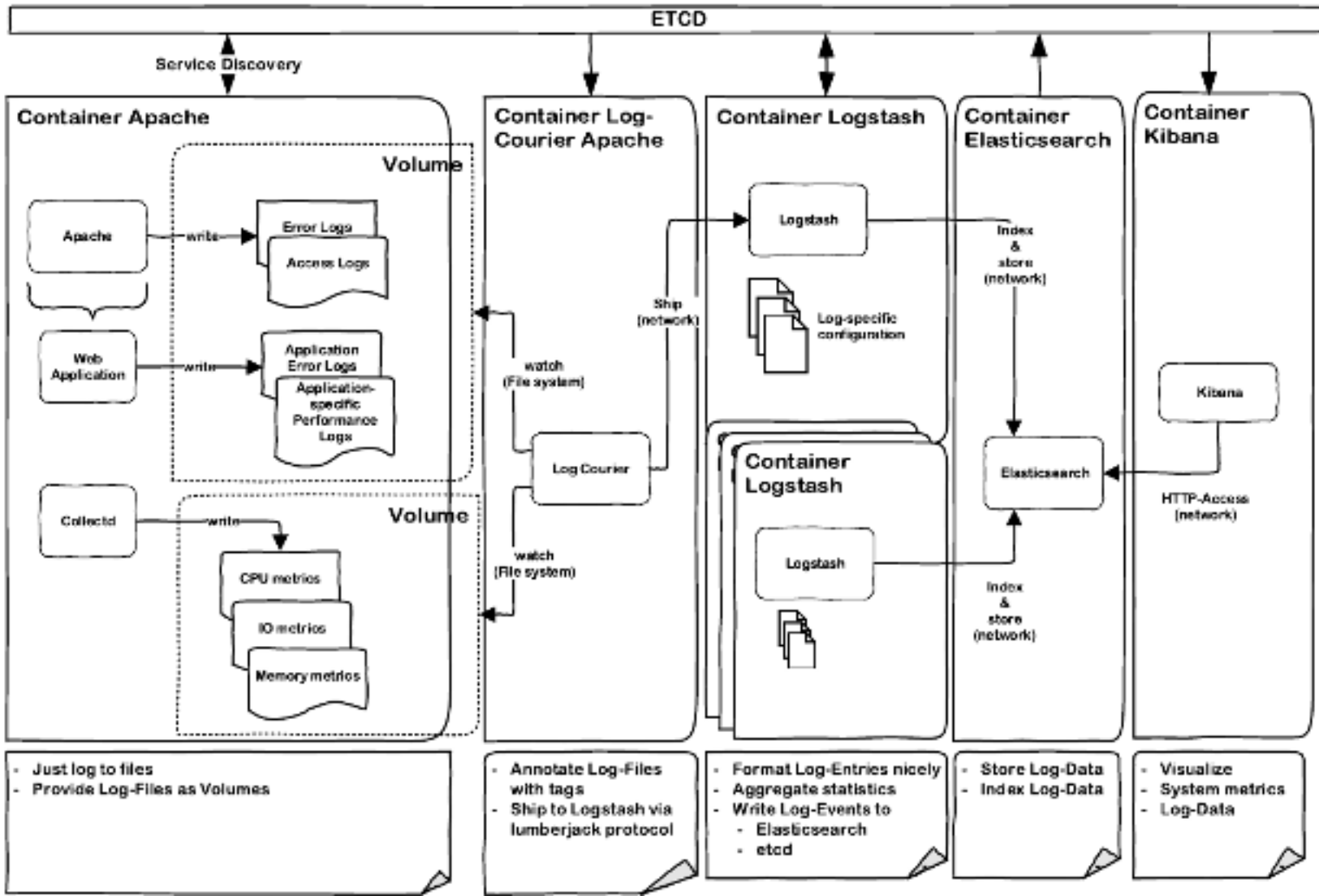
Self-managing applications / 3

CNA-ified Zurmo CRM software architecture, deployed on fleet/CoreOS



Self-managing applications / 4

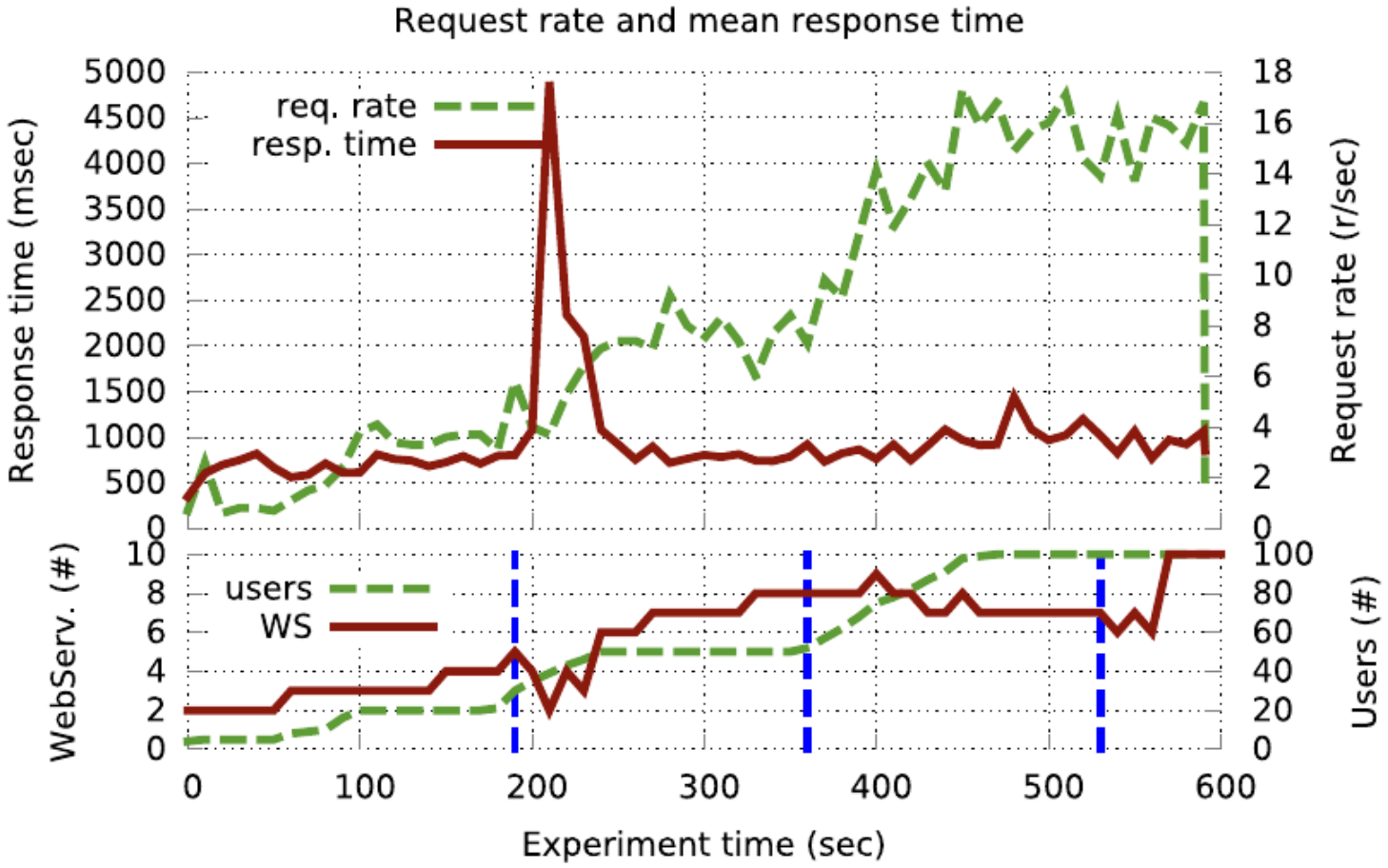
Monitoring and logging processes within the CNA-ified architecture



Service Engineering Research Area
ICCLAB SPLAB

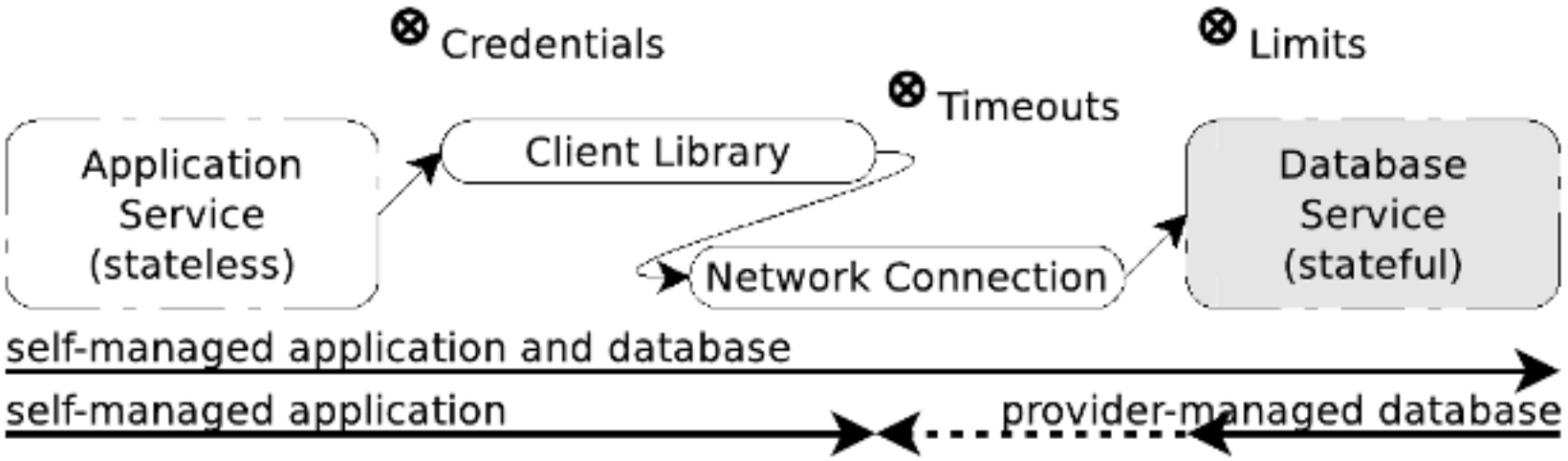
Self-managing applications / 5

Benefits of the architecture in case of (induced) failures



Stateful services / 1

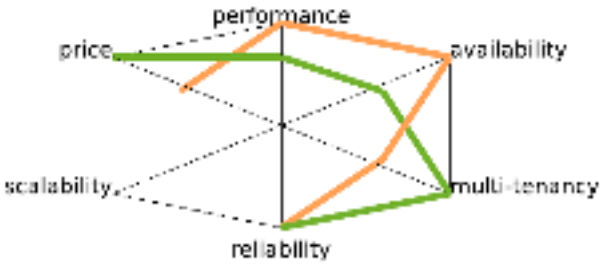
Example: databases in CNA → CNDB



Stateful services / 2

Are provider-managed databases a good idea?

Pricing for MySQL service (orange) and self-managed container (green) at Google Cloud:

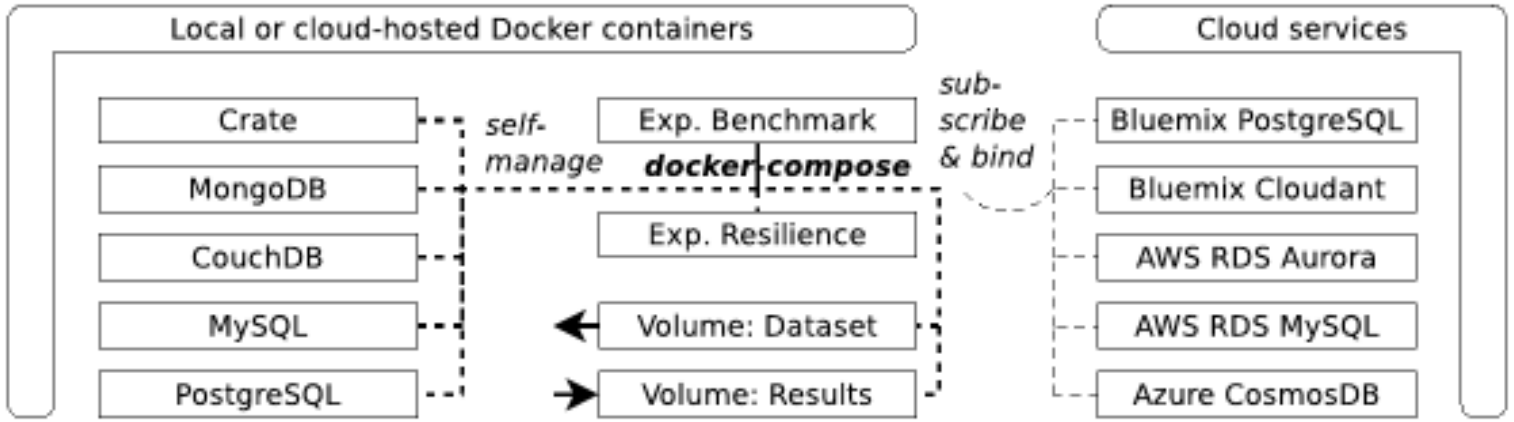


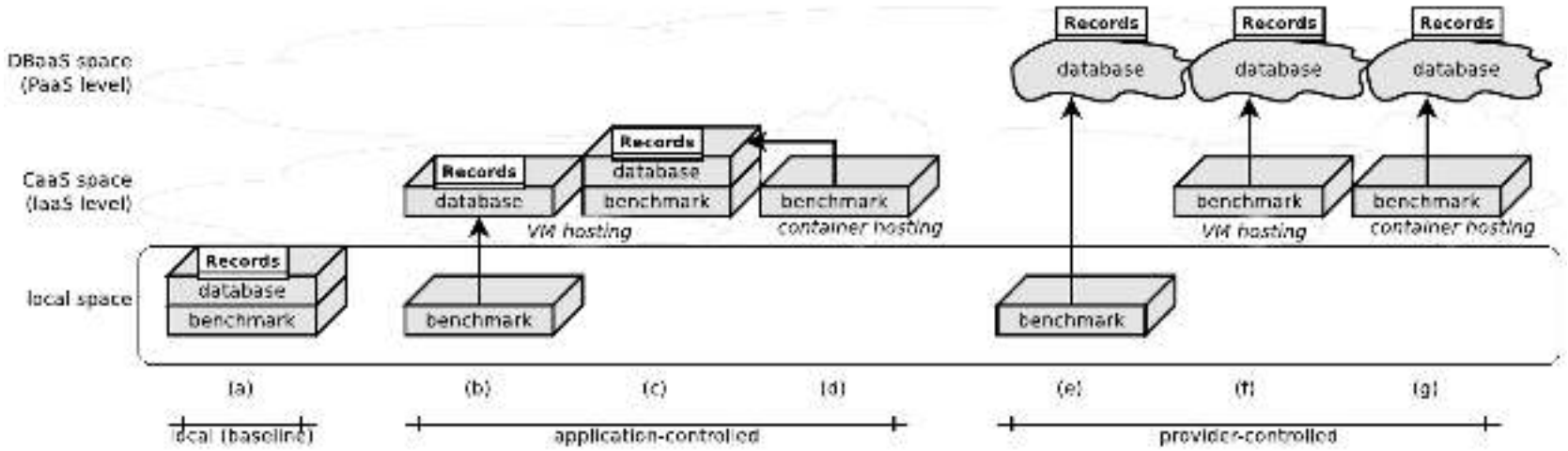
Amazon Web Services	Google Cloud	Microsoft Azure	IBM Bluemix	APPUiO	Swisscom AC	Application Interface	Implementation
	X[2]						MySQL *
				X	X		MariaDB *
			X	X			PostgreSQL *
							Aurora
							Oracle DB
		X[3]					SQL Server
			X				DB2
		(X)[4]		X	X		MongoDB *
			X[5]				CouchDB *
X						JSON QL or similar (Mango etc.)	DynamoDB
		X					CosmosDB
		X					TableStorage
	X						Datastore
	X					other	BigTable (*)
X[6]	X[7]	X			X		Redis *

Notes: 1: RDS, 2: Cloud SQL, 3: Database Service, 4: CosmosDB adapter, 5: as Cloudant NoSQL, 6: as ElastiCache; 7: via external RedisLabs service

Stateful services / 3

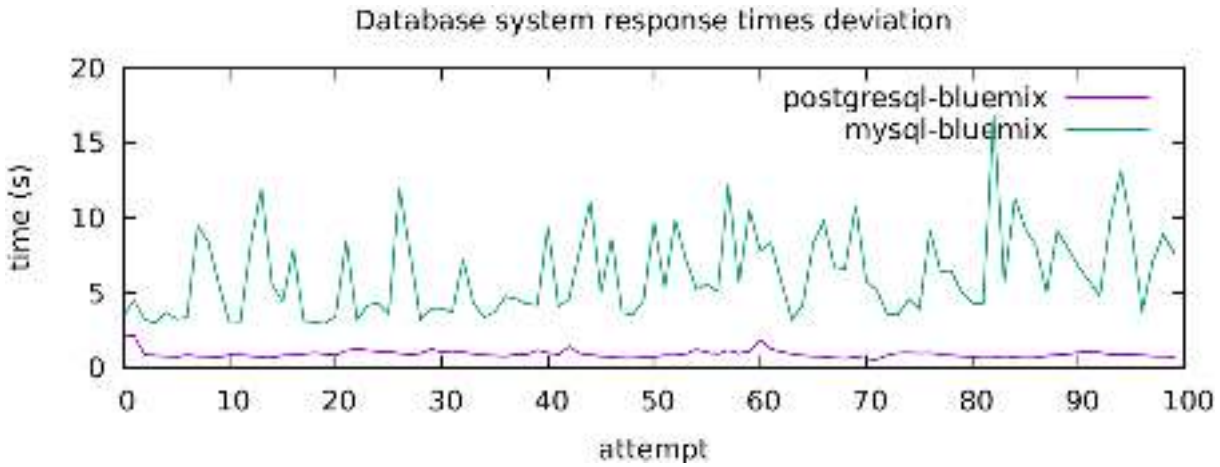
Benchmarking with CNDBbench

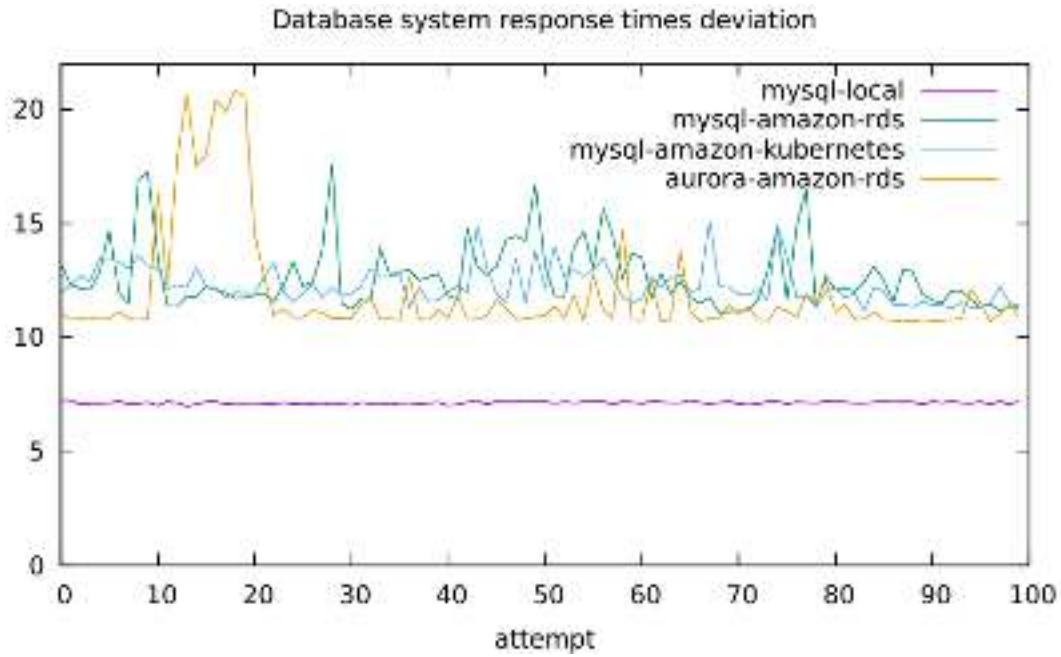




Stateful services / 4

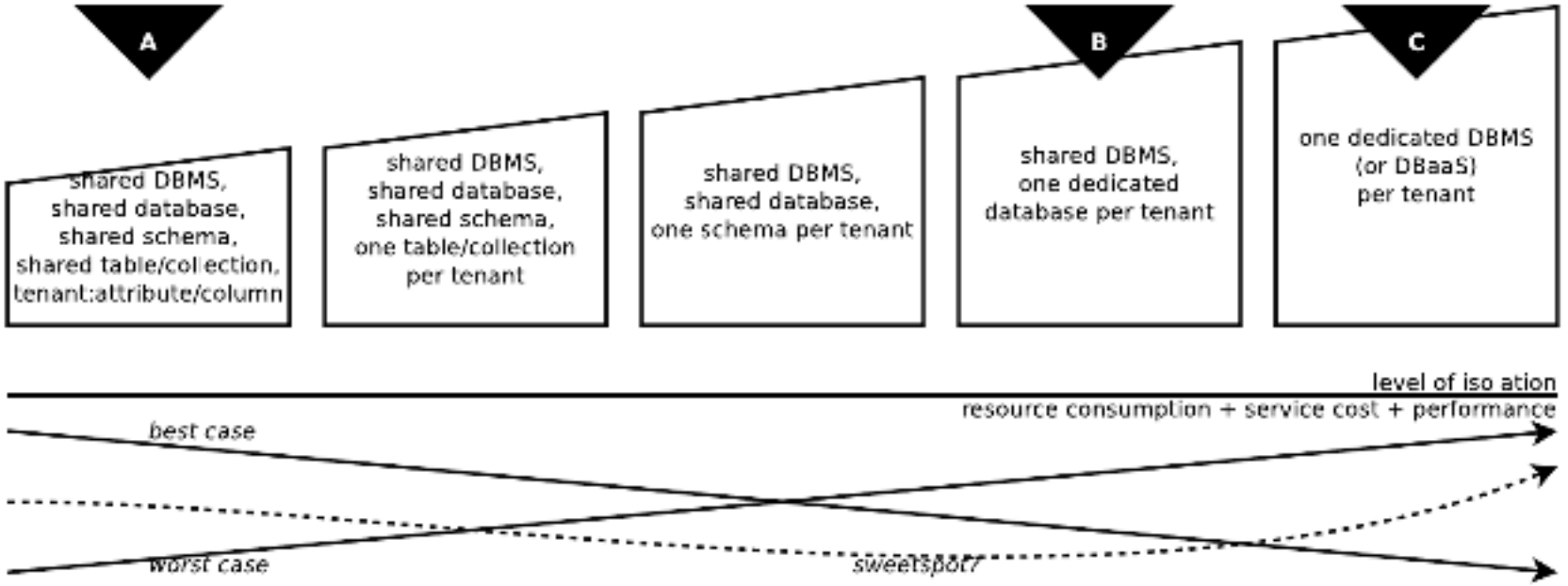
Selected benchmark results





Stateful services / 5

Multi-tenancy considerations



Stealth properties / 1

Basic assumptions: no trust in dependency cloud services

- may disappear
- may cause data leaks or irreversible deletion
- may be too slow
- lock-in effects

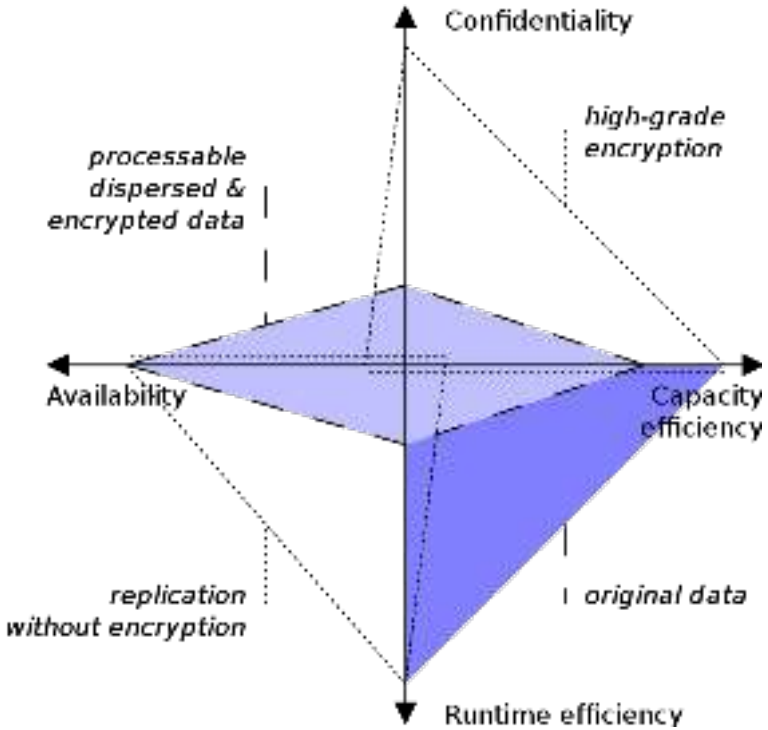
Solution idea:

- selective redundancy to achieve controlled trade-offs
- transmission, storage, processing over any number of nodes
- nodes can be spread across providers, no cooperation required (nor desired in most cases)

Stealth properties / 2

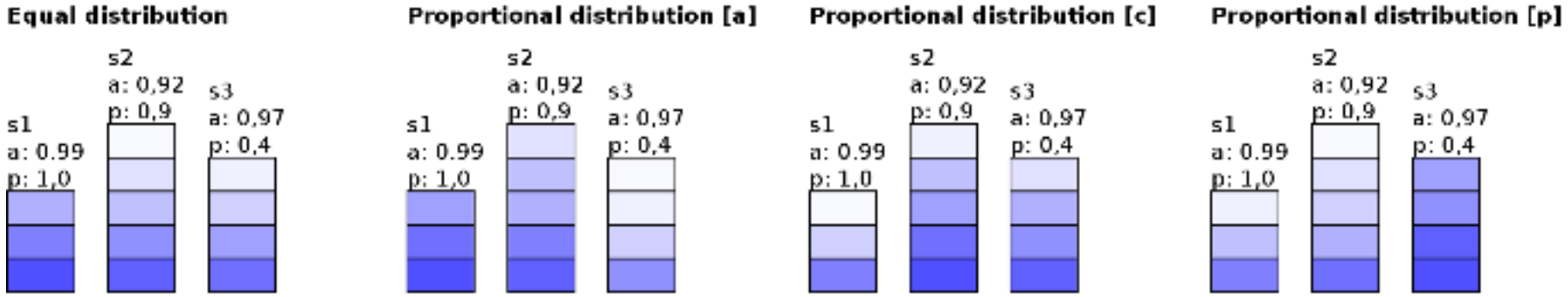
Data coding choices

Data coding trade-offs

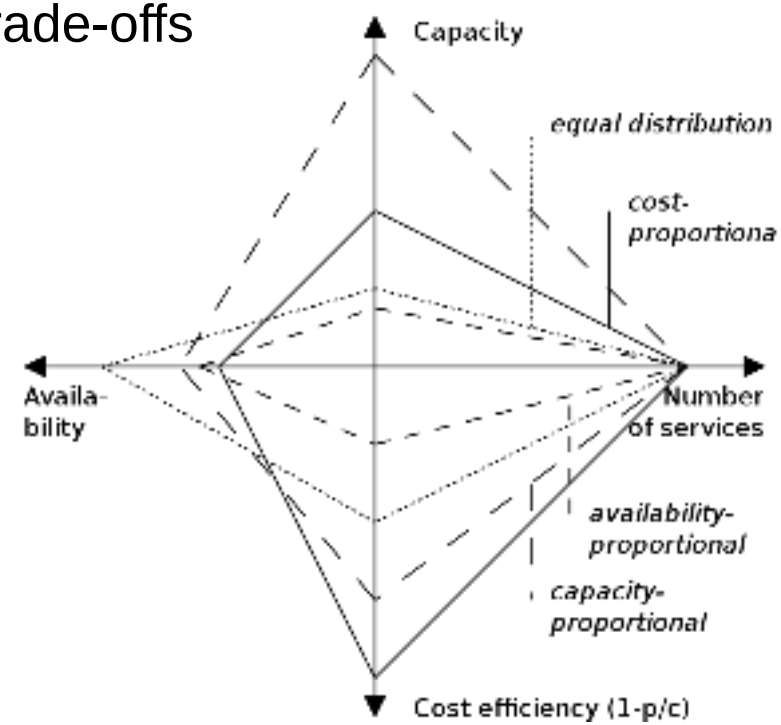


Stealth properties / 3

Data distribution choices



Data distribution trade-offs



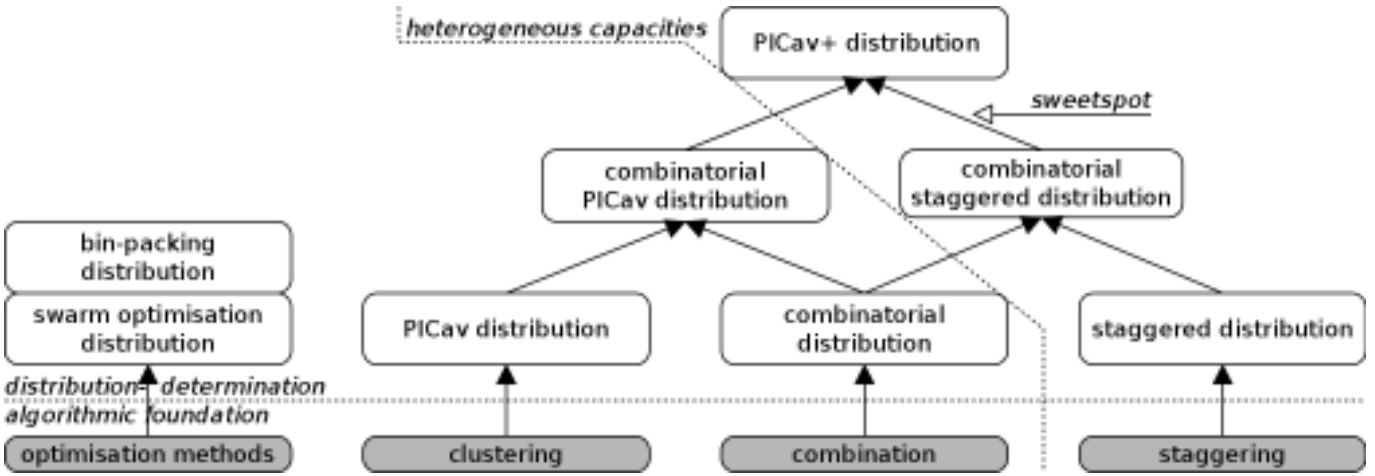
Stealth properties / 4

Data distribution algorithm: PICav+ (fast & precise)

- powerful: optimises for capacity, price, availability constraints & runtime
- staggered: considers all elements in powerset of candidate service set
- sliced: capacity-maximising calculation rings
- iterative: finds some result first, finds best result eventually

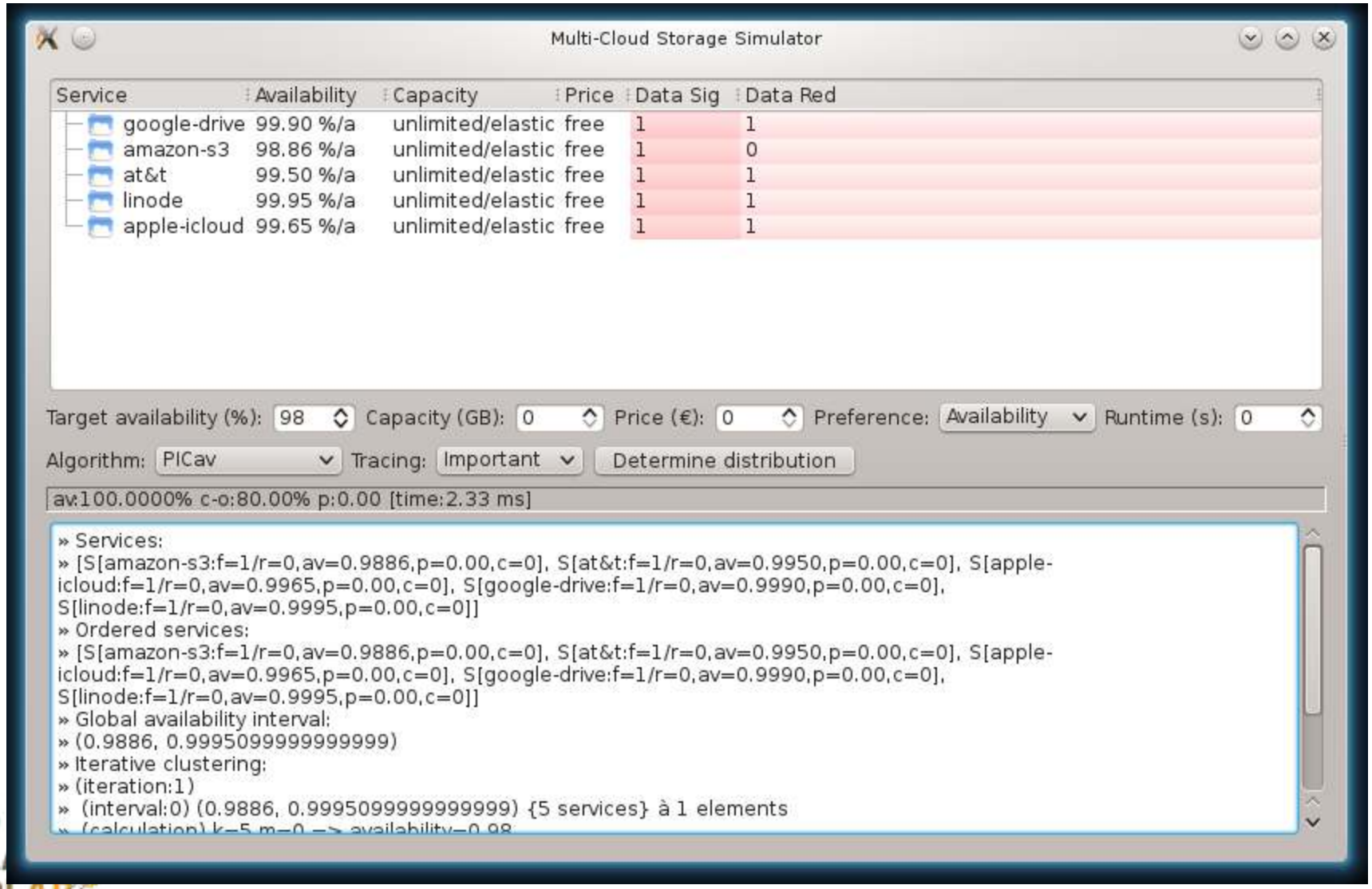
Homogeneous complexity: $$availability = \sum_{i=k}^{n} \binom{n}{i} a_1^i (n - a_1)^{n-i}$$

Heterogeneous complexity: (each per slice) $$availability = \sum_{S \in P_{>k}(C)} \left(\prod_{i \in S} a_i \cdot \prod_{i \in C \setminus S} 1 - a_i \right)$$



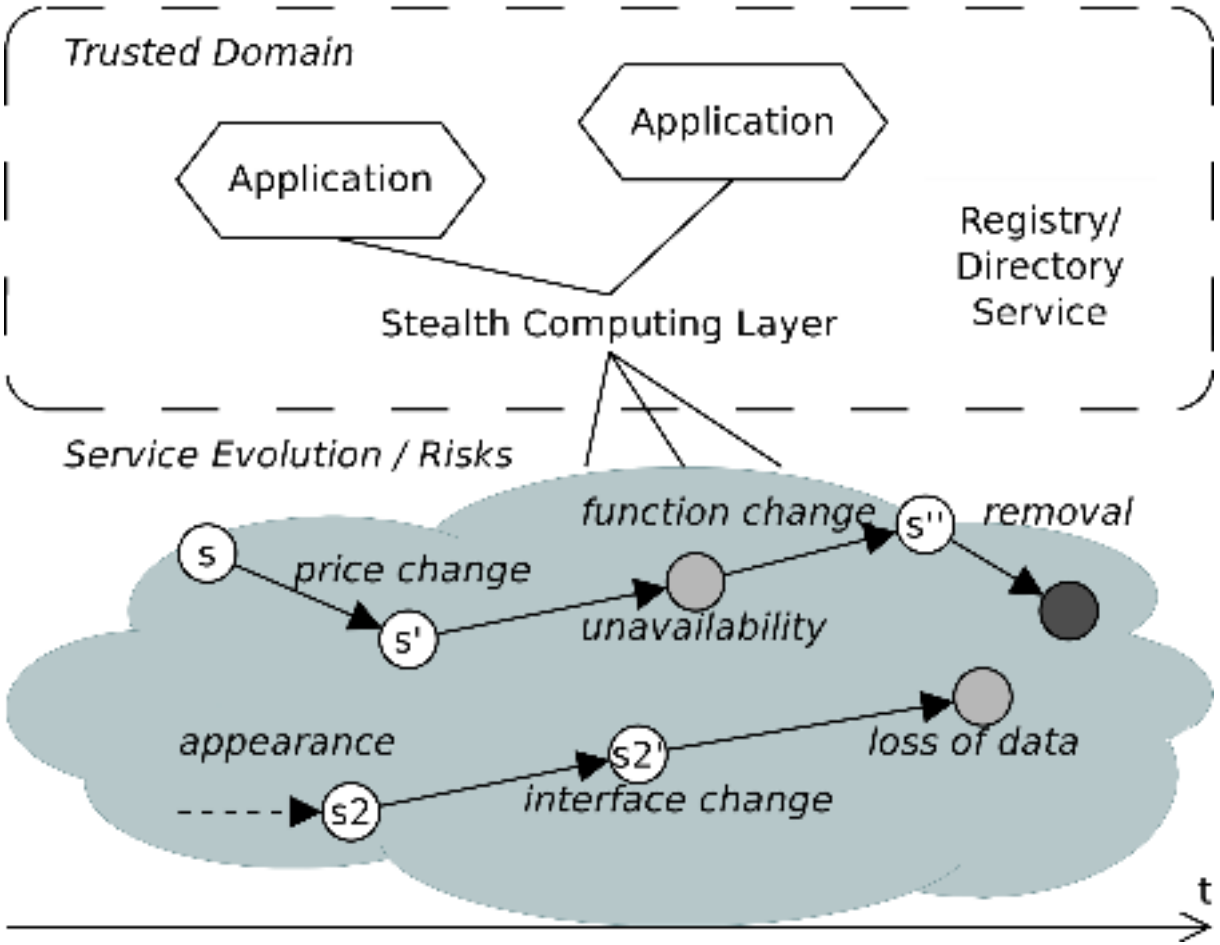
Stealth properties / 5

Simulation/emulation tool: MC-SIM/MC-EMU



Stealth properties / 6

Stealth layer for applications



data_locations = {0-10s: {s}, 11-20s: {s,s2}, 21-60s: {s2}}

Conclusions

Future applications will be predominantly cloud-native

- self-managed microservice compositions
- elastically scalable, resilient, stealthy, adaptive to target environment
- but: effort-intensive, requires design reviews, testing, benchmarking and fault injection

Bibliography

G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, T. M. Bohnert: *"Self-managing cloud-native applications: design, implementation, and experience"*, Future Generation Computer Systems (FGCS) 72:165--179, July 2017. DOI: 10.1016/j.future.2016.09.002

Submitted manuscript: J. Spillner, G. Toffetti, M. R. López: *"Cloud-Native Databases: An Application Perspective"*, July 2017

Submitted manuscript: M. R. López, J. Spillner: *"Towards Quantifiable Boundaries for Elastic Horizontal Scaling of Microservices"*, August 2017

M. R. López: *"Cloud-Native Microservices Reference Architecture"*, SPLab blog, July 2017
J. Spillner: *"Rightsizing Kubernetes applications"*, SPLab blog, June 2017

Blog site: https://blog.zhaw.ch/icclab/