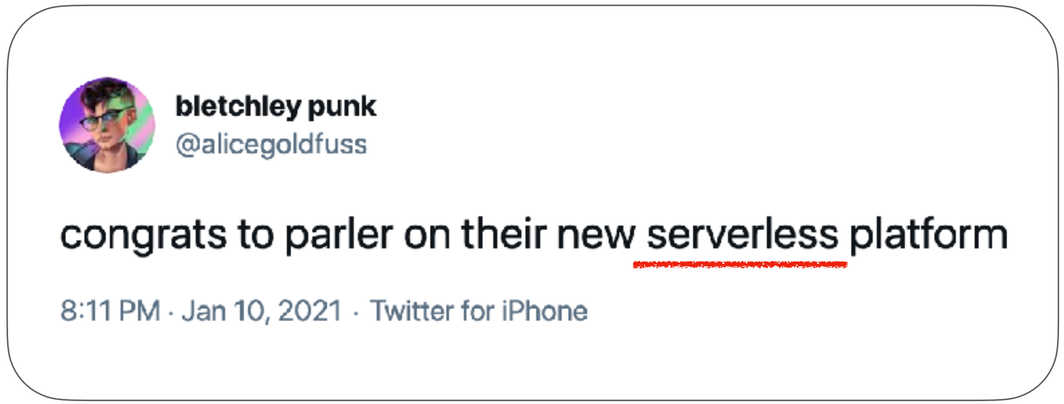
Serverless V Microservices

Saverio Giallorenzo

Università di Bologna (IT) and INRIA (FR)

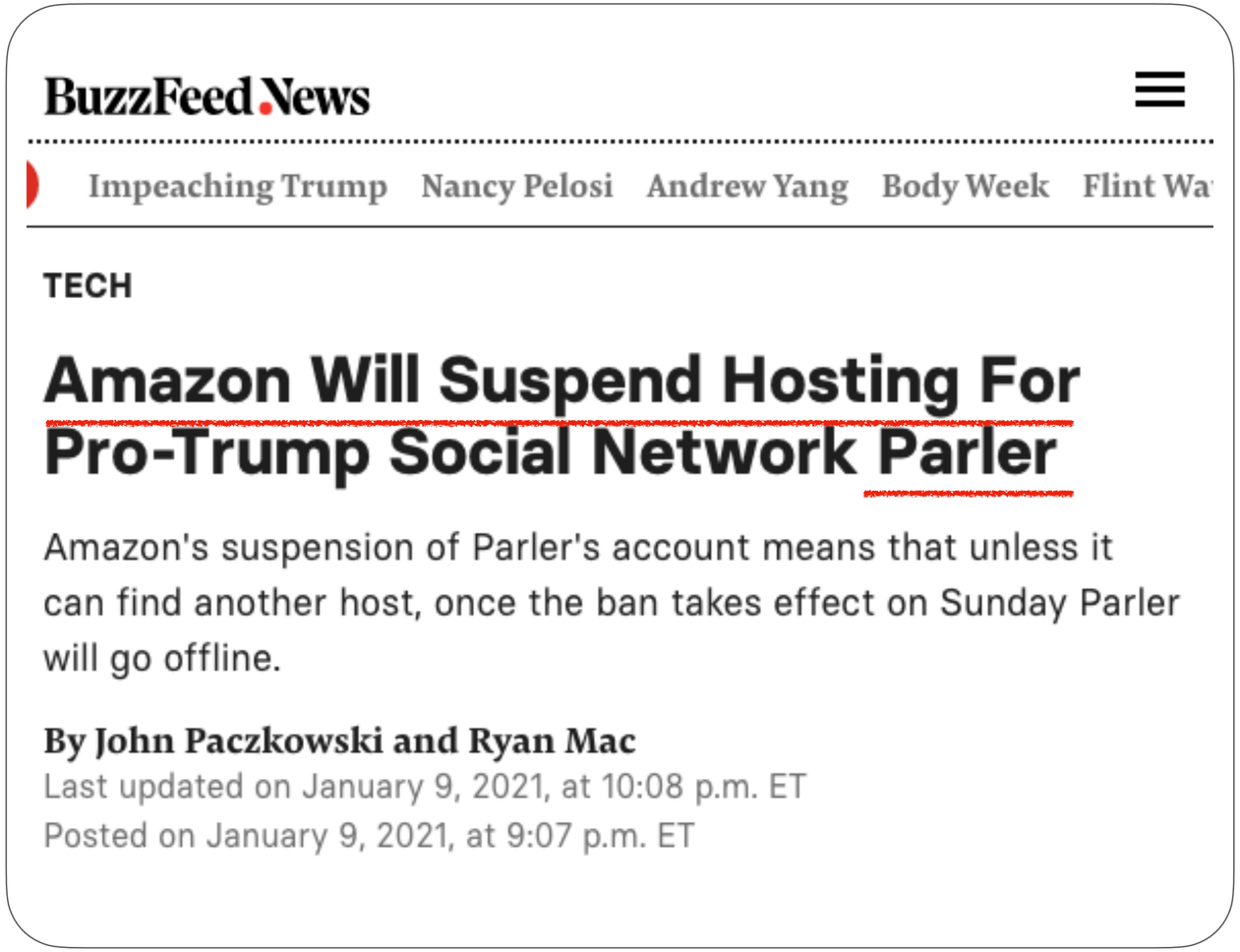
Server... less?



bletchley punk
@alicegoldfuss

congrats to parler on their new serverless platform

8:11 PM · Jan 10, 2021 · Twitter for iPhone



BuzzFeed News

Impeaching Trump Nancy Pelosi Andrew Yang Body Week Flint Wa

TECH

Amazon Will Suspend Hosting For Pro-Trump Social Network Parler

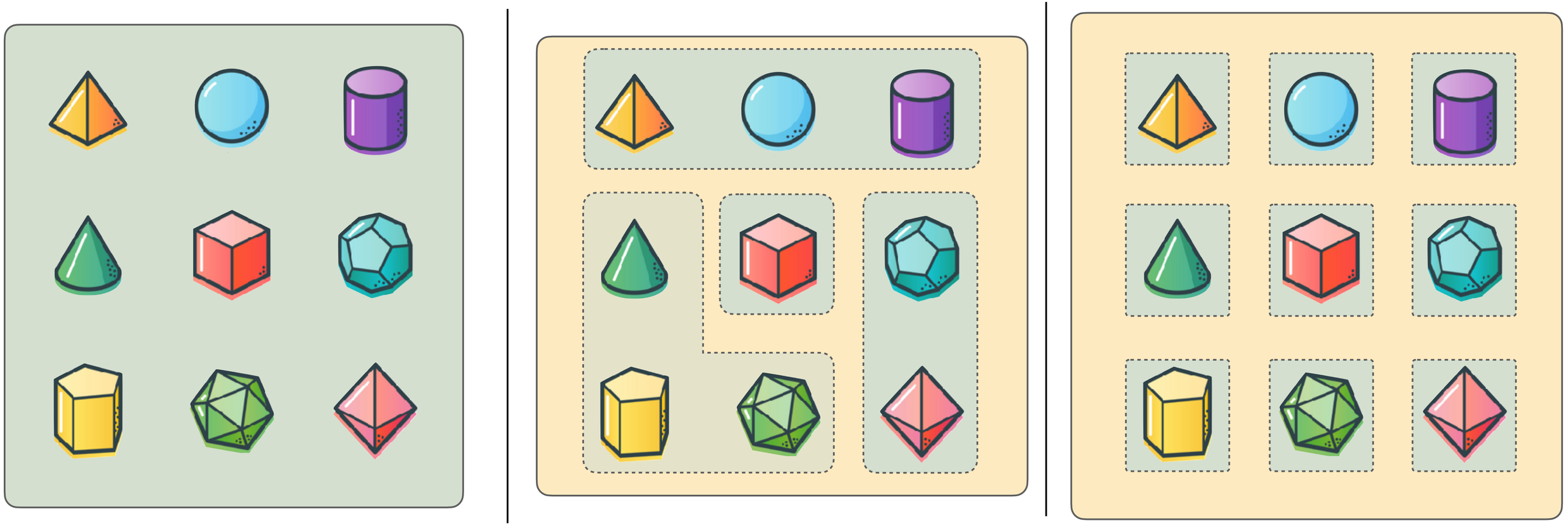
Amazon's suspension of Parler's account means that unless it can find another host, once the ban takes effect on Sunday Parler will go offline.

By John Paczkowski and Ryan Mac
Last updated on January 9, 2021, at 10:08 p.m. ET
Posted on January 9, 2021, at 9:07 p.m. ET

Of Monoliths, Microservices, and Serverless

provisioned, pay-per-deployment

on-demand, pay-per-execution



Monolith

Microservices

Serverless

Function

Software Unit

Runtime Environment

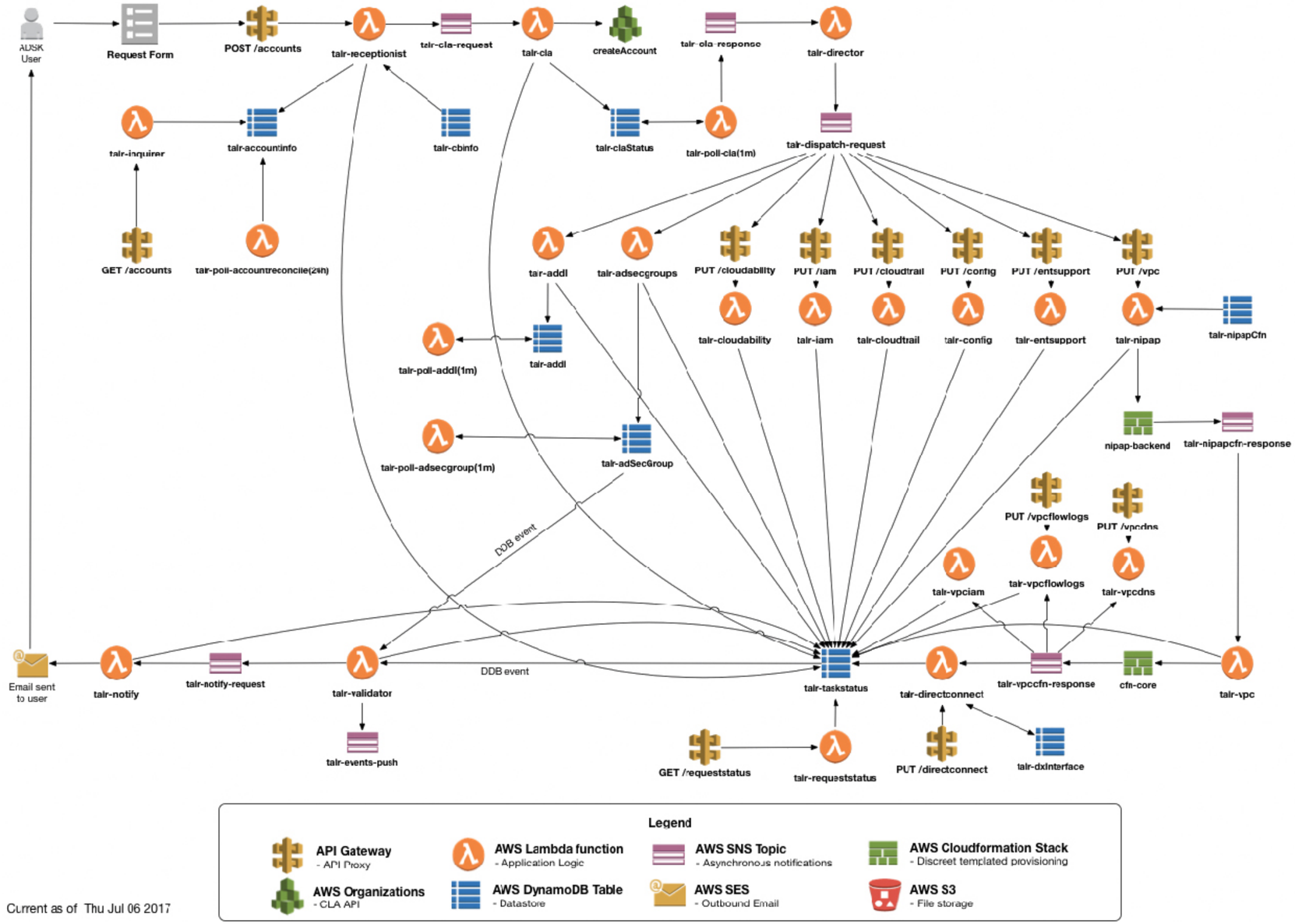
Serverless • Use Case

AUTODESK.

Tailor

Going Serverless with AWS Lambda relieves us from managing servers and lets us concentrate on building features.

— Alan Williams
Autodesk Enterprise Architect



Current as of Thu Jul 06 2017

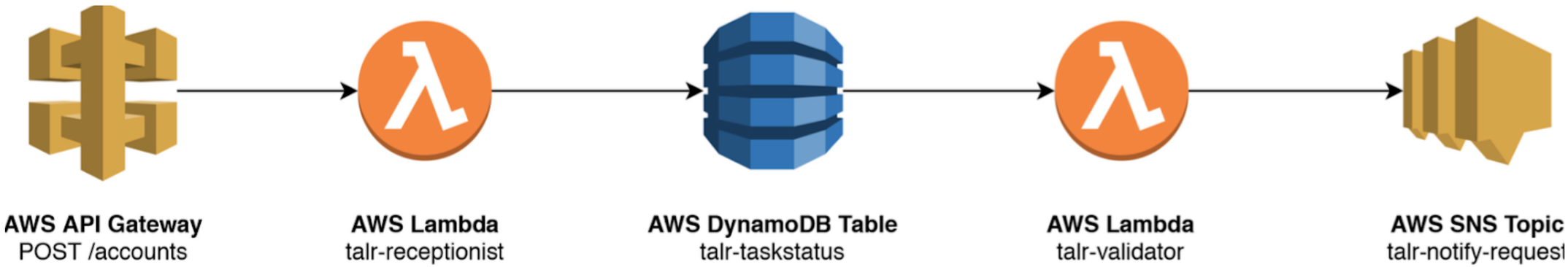
https://aws.amazon.com/solutions/case-studies/autodesk-serverless/

Serverless • Use Case

AUTODESK.

Tailor

an excerpt



https://github.com/alanwill/aws-tailor

Serverless V Microservices

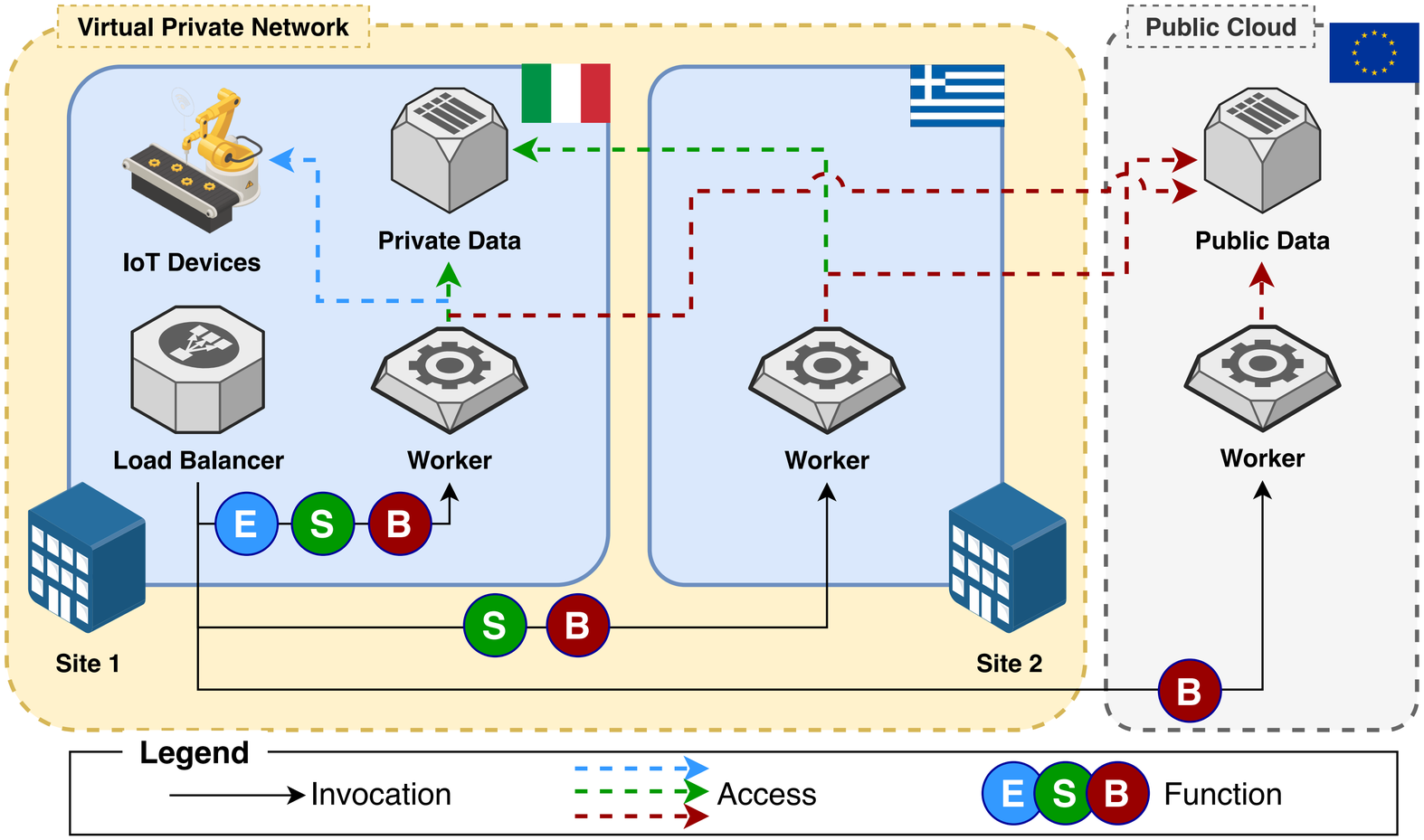
	Microservices	Serverless
Common	• Need migration • Complex testing (integration) • Technology-agnostic Architecture • Component flexibility and code reuse	
Neuter	• Managed scalability • Stateful and Stateless	• Responsive scalability • Stateless by construction • Bound to specific platforms for deployment • Time bounds (e.g., 15-minute timeout)
+	• Architecture-defined component granularity • Winning pricing model for steady traffic	• Winning pricing model for variable (intraday) traffic • No servers to managed (minimised Ops costs) • Simpler release cycles
-	• Ops costs • Complex release cycles (mainly due to statefulness) • Complex deployment chains (due to statefulness and dependencies) • Without centralised orchestration, fragmented flow of control	• Performance/Platform-dependent granularity • Fragmented flow of control (decentralised by construction) • Possible lock-in depending on the platform

Research • Allocation Priority Policies

```
Function_E:
  - workers:
    - worker_site1
  followup: fail

Function_S:
  - workers:
    - worker_site2
    - worker_site1
    strategy: random
  followup: fail

Function_B:
  - workers:
    - worker_public_cloud
    - worker_site2
    - worker_site1
  strategy: best_first
  followup: fail
```



De Palma, G., Giallorenzo, S., Mauro, J., & Zavattaro, G. (2020). **Allocation Priority Policies for Serverless Function-Execution Scheduling Optimisation**

Research • Jolie – Microservices V Serverless V IOT

One **specification**

```
interface TwiceInterface {
  requestResponse:
   twice( int )( int )
}
```

+

```
main
{
  twice( number )( result ) {
    result = number * 2
  }
}
```

One **behaviour**

Many **deployments**

```
inputPort ServerlessPort {
  location: "hook://myhook"
  protocol: AWS_lambda
  interfaces: TwiceInterface
}
```

```
inputPort MicroservicePort {
  location: "socket://myhost:8000"
  protocol: http
  interfaces: TwiceInterface
}
```

```
inputPort IOTPort {
  location: "socket://myhost:8000"
  protocol: mqtt {
    broker = "socket://broker.com:1883"
   }
  interfaces: TwiceInterface
}
```


THANK
YOU