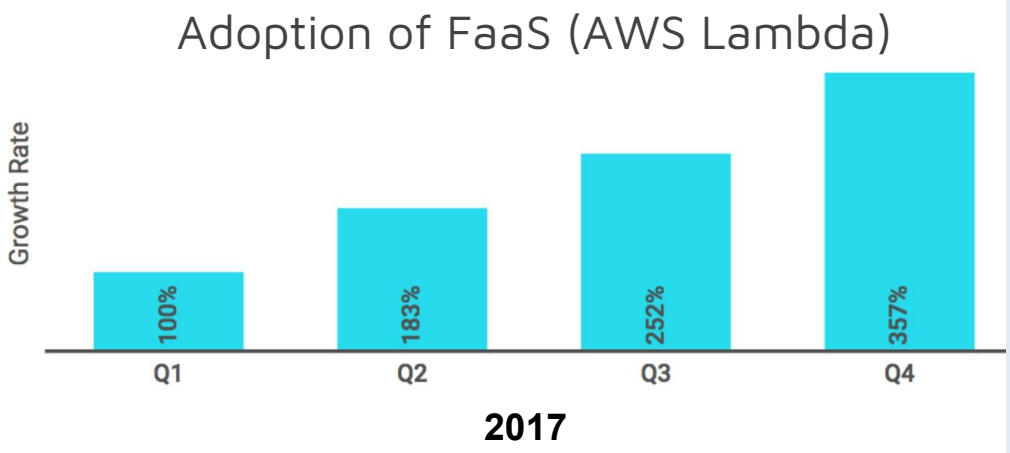
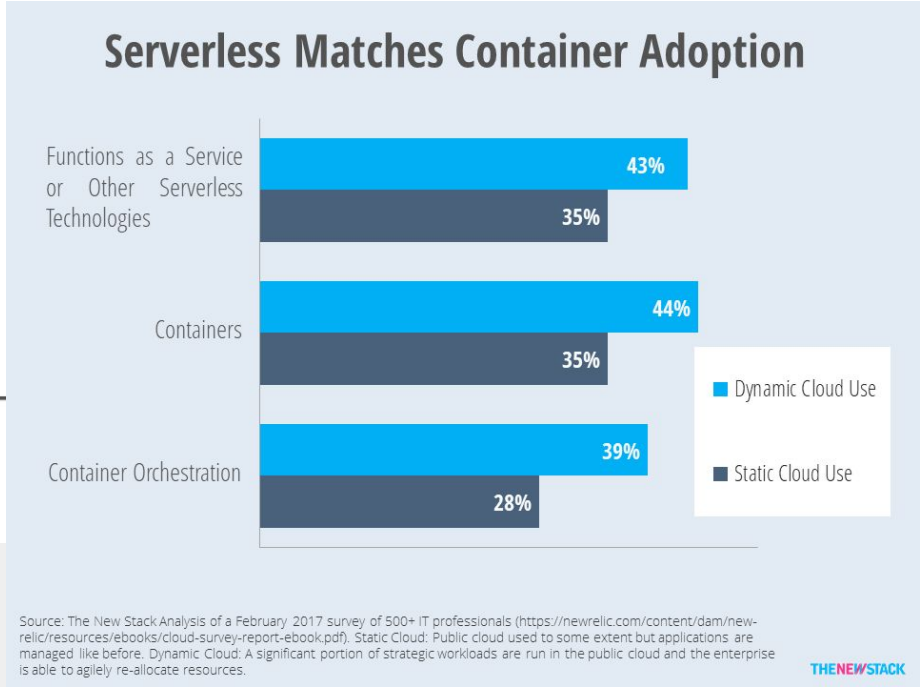
Optimizing Latency in Function-as-a-Service with Distributed Promises

Soam Vasani, **Erwin van Eyk**

Platform9 Systems Inc.

Function-as-a-Service Market estimated to be worth **$7.72 Billion** by 2021





Gartner 2017 Function-as-a-Service is one of the top trends in cloud computing.

https://www.marketsandmarkets.com/Market-Reports/function-as-a-service-market-127202409.html
http://get.cloudability.com/ebook-state-of-cloud-2018.html

Research into Function-as-a-Service

SPEC RG CLOUD
Serverless Activity

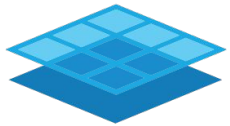
Combined industry and academia effort to address high-level (performance) challenges

Terminology
Challenges
Reference Architectures
Benchmarking

@Large Research
Massivizing Computer Systems

Actively exploring challenges in FaaS

Complex FaaS function composition using **workflows**.
On-demand **Graph Processing** with FaaS



PLATFORM9

Open Source Software as a Service

IaaS OpenStack
PaaS Kubernetes
FaaS Fission (own product)

Serverless Cloud Native Landscape

v2.1

See the serverless interactive landscape at **s.cncf.io**

Greyed logos are not open source

Libraries

python-λ

Tools

.arc Architect, Dashbird, IO|pipe, LambCI, Node Lambda, STACKERY, Stackvana, THUNDRA

Frameworks

APEX, aws Chalice, BUSTLE, Claudia.js, Dawson, FLOGO, Gordon, gun.io, kappaIO, MITOC GROUP, SAM Local, serverless, SPARTA, Spring Cloud Function

Platforms

ALGORITHMIA, OpenWhisk, AppScale, AWS Lambda, Azure Functions, Clay, Google Cloud Functions, HYPER.SH, IBM Cloud, Iron.io, NANO-LAMBDA, OVERCLOCK, OVH, PubNub FUNCTIONS, spotinst, stdlib, syncano, twilio, WEBLAB, webtask, ZEIT

Hybrid: BINARIS, cloudboost, fn, fx, GALACTIC FOG, LunchBadger, nuc

Kubernetes-Native: fission, funktion, Kubeless, OPENFAAS, OpenLambda, Oracle Application Container Cloud, PubNub, riff

Security

PURESEC, snyk

github.com/cncf/landscape

Serverless comp... to a new model of cloud native computing, enabled by archite... that do not require server management to build and run applications. This landscape illustrates a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and billed in response to the exact demand needed at the moment.

CLOUD NATIVE Landscape

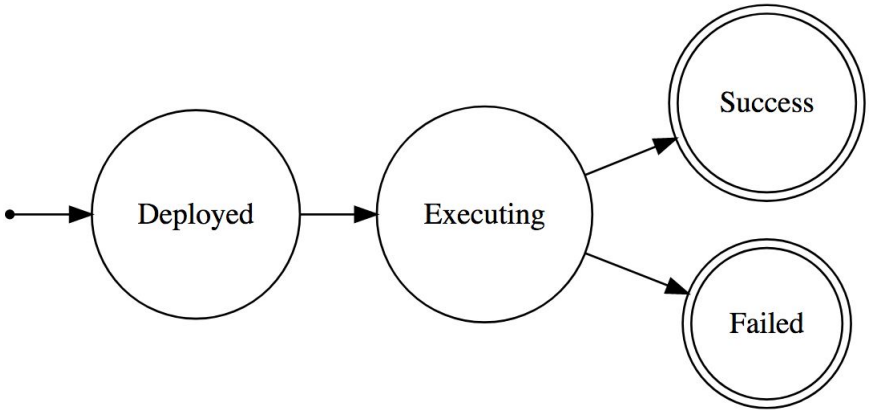
CLOUD NATIVE COMPUTING FOUNDATION

Redpoint

FaaS Function Model

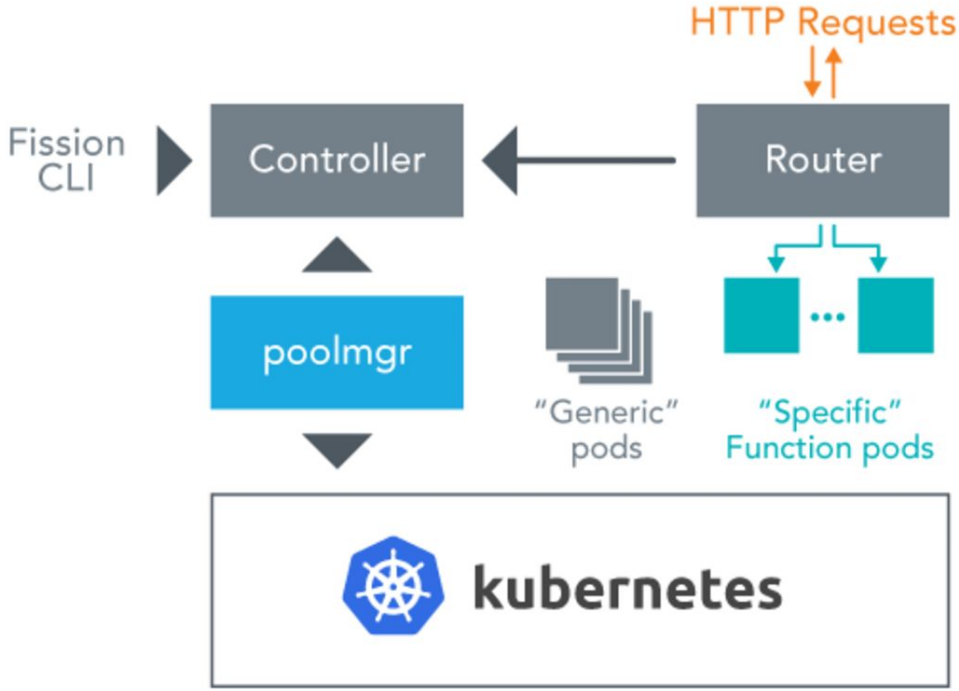
Function characteristics

- Atomic
- Finite runtime
- Self-contained
- Stateless



fission

Controller manages function definitions and exposes API.
PoolMgr maintains a pool of running ‘generic’ containers to reduce deployment time of functions.
Router directs requests to the ‘right’ deployed function.



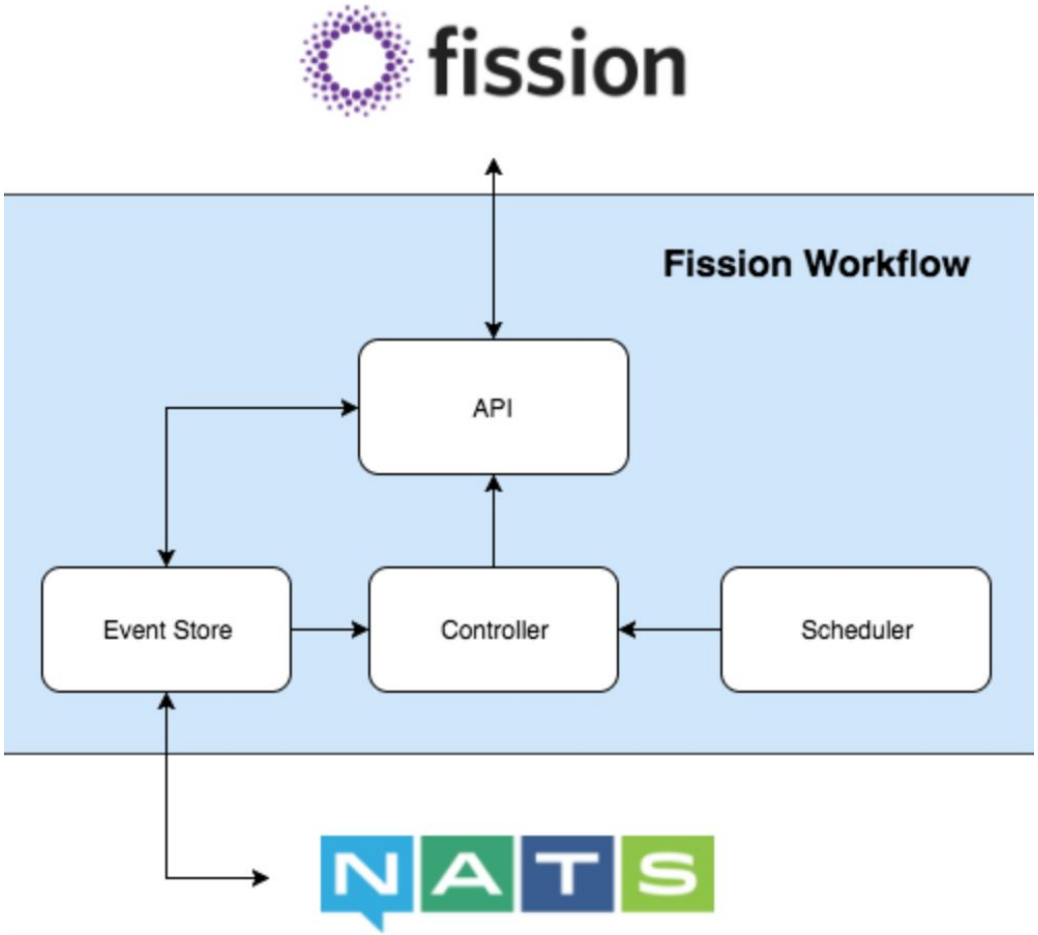
Composing Complex Functions

Challenges

- Dealing with communication logic
- Structure hard to understand
- Implicit dependencies
- Performance overhead of function calls (eg. cold starts)

fission workflows

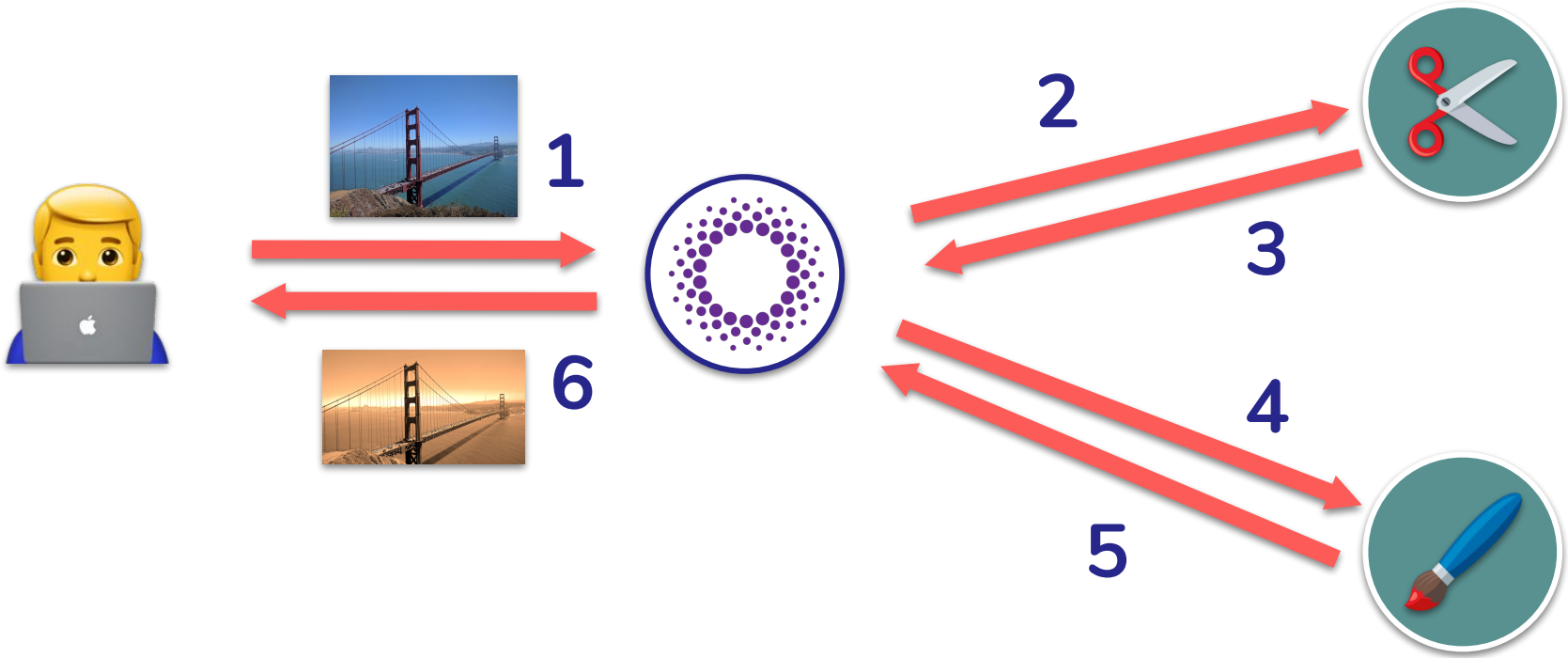
Event Store persist all events related to the workflows.
Controller watches active workflows and execute scheduled actions.
Scheduler decides which actions need to be taken based on current state.



Example: Photo Transform Workflow

- **Data-intensive**, on-demand functions
- Several distinct **computation** steps for transformation

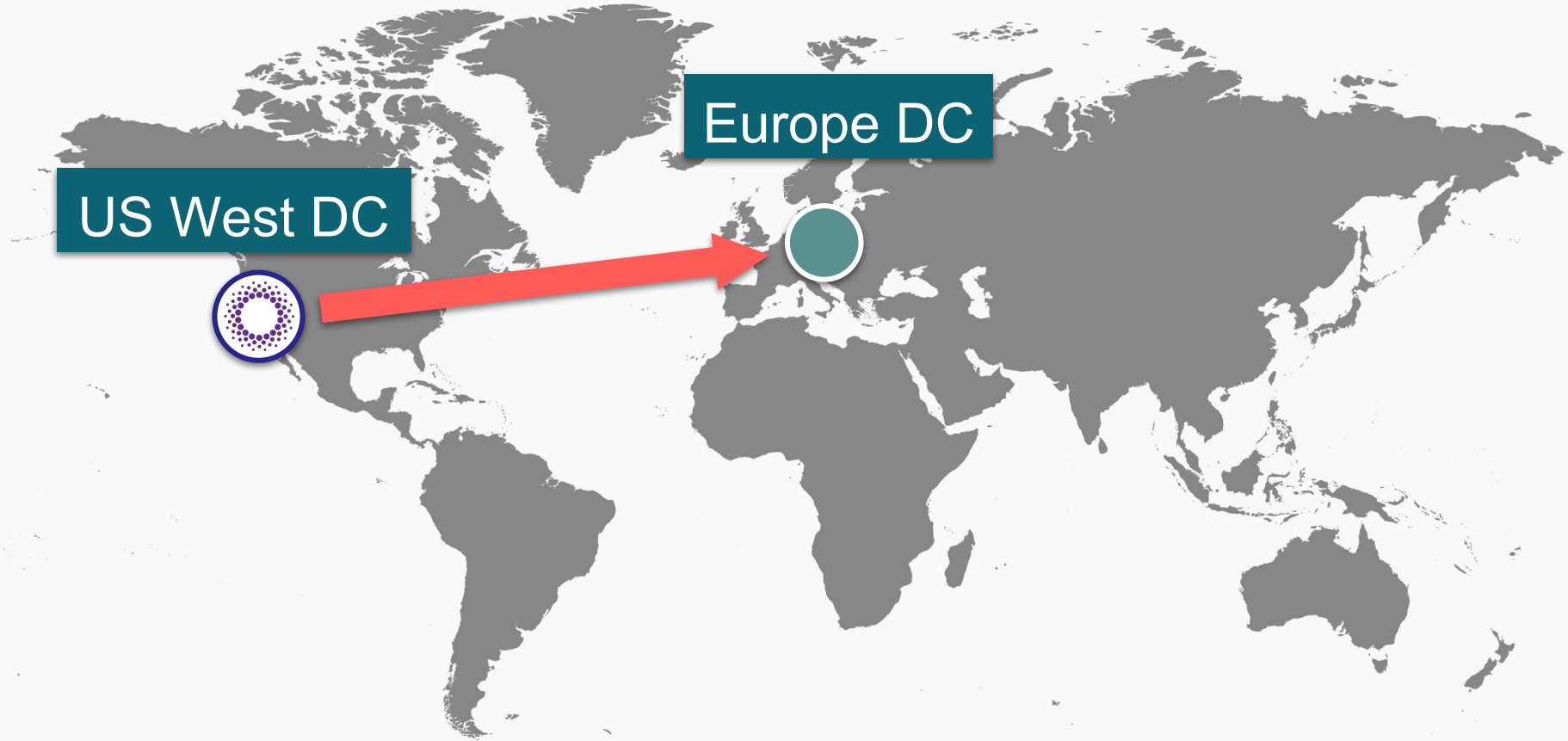
Photo Transform Workflow

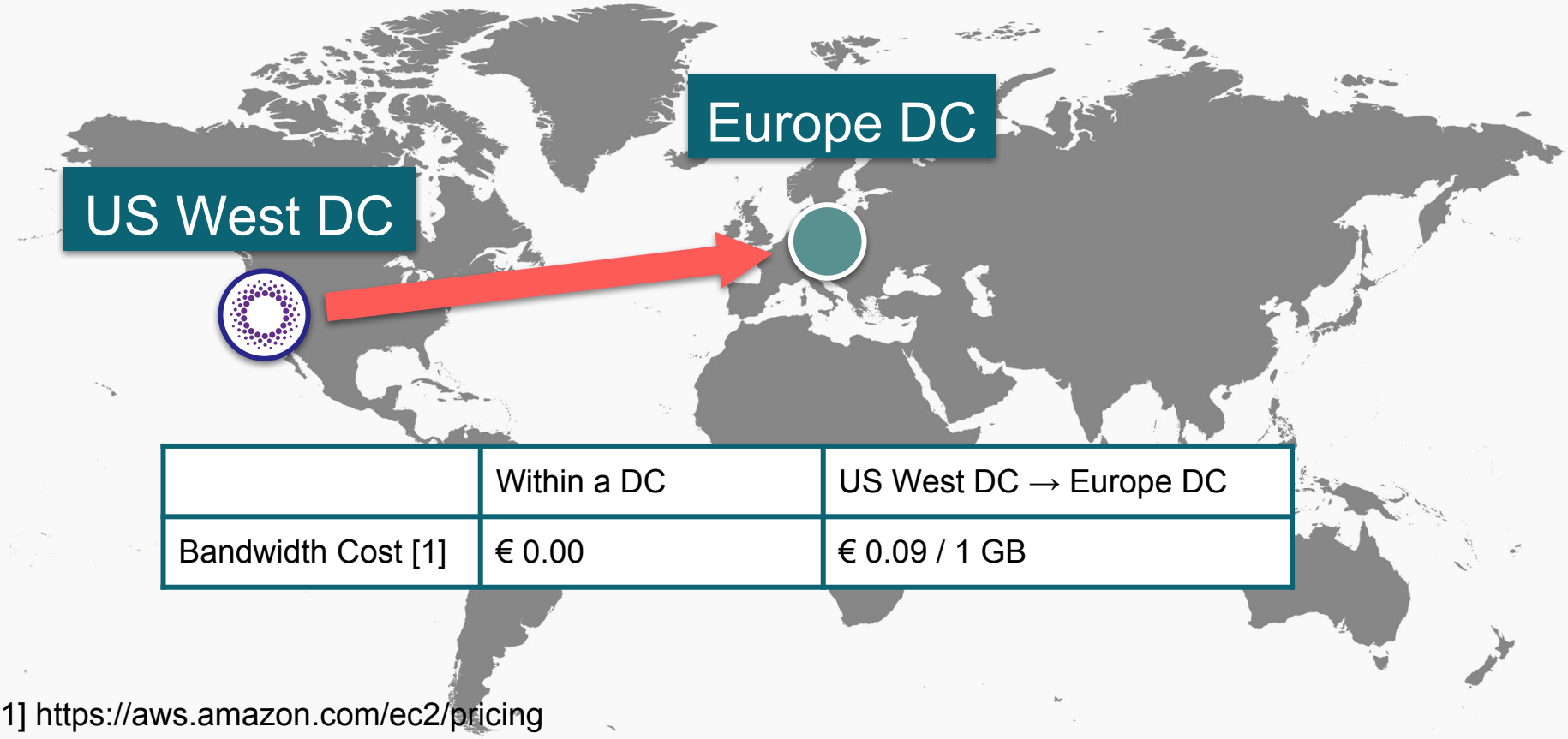


The Problem

Data (Transfer)

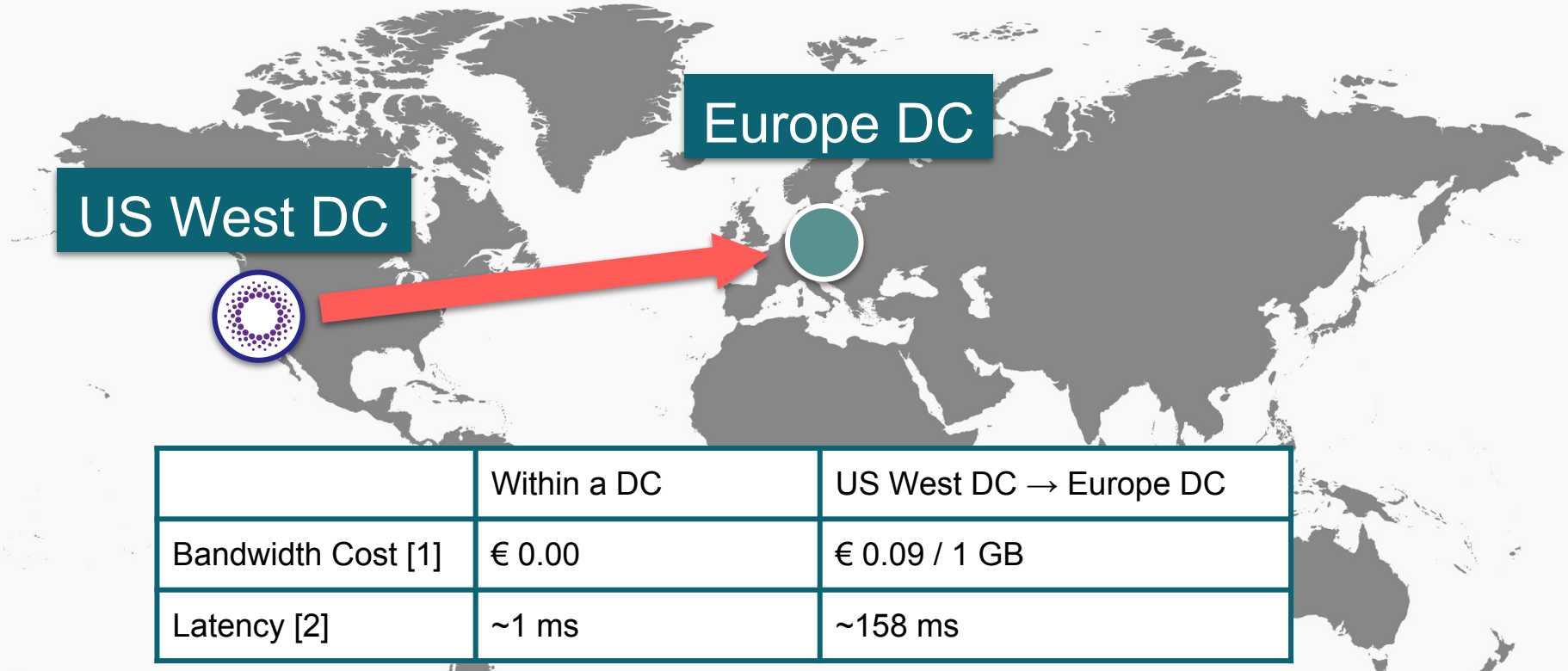
and lots of it!





	Within a DC	US West DC → Europe DC
Bandwidth Cost [1]	€ 0.00	€ 0.09 / 1 GB

[1] https://aws.amazon.com/ec2/pricing

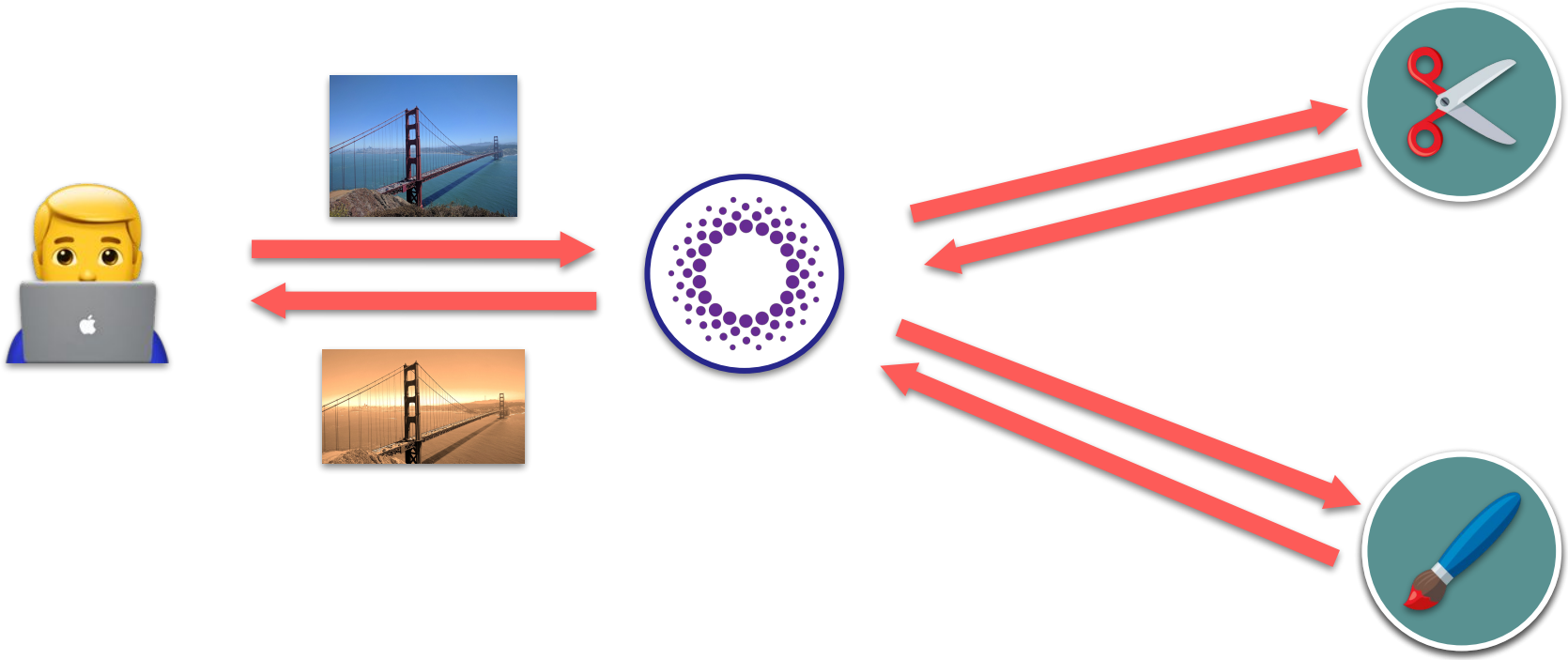


	Within a DC	US West DC → Europe DC
Bandwidth Cost [1]	€ 0.00	€ 0.09 / 1 GB
Latency [2]	~1 ms	~158 ms

[1] https://aws.amazon.com/ec2/pricing

[2] Gandhi, A., & Chan, J. (2015). Analyzing the Network for AWS Distributed Cloud Computing. ACM SIGMETRICS Performance Evaluation Review

Naive Approach

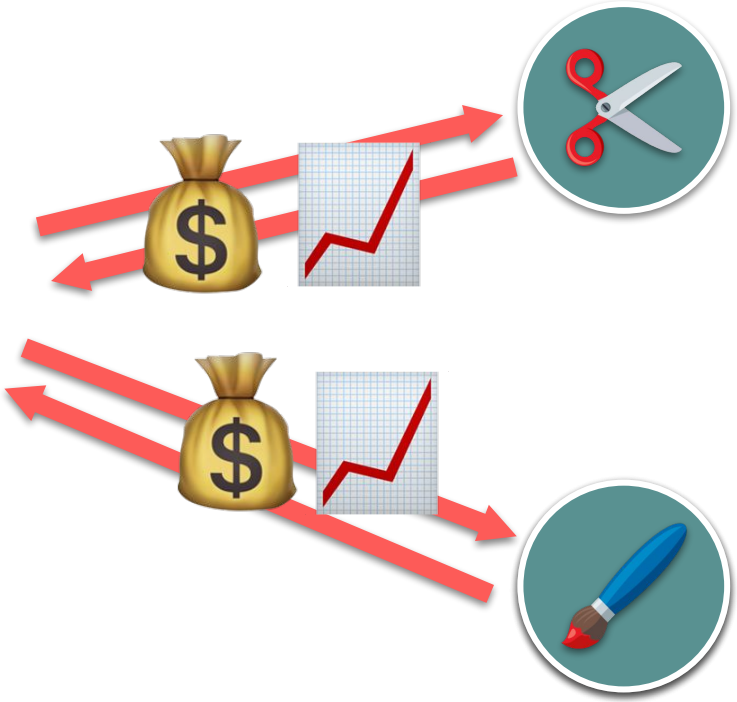


Naive Approach: expensive and slow!

Color Task can only start after controller receives the data!

Costs € 0.09 * 4 = **€ 0.36** per GB

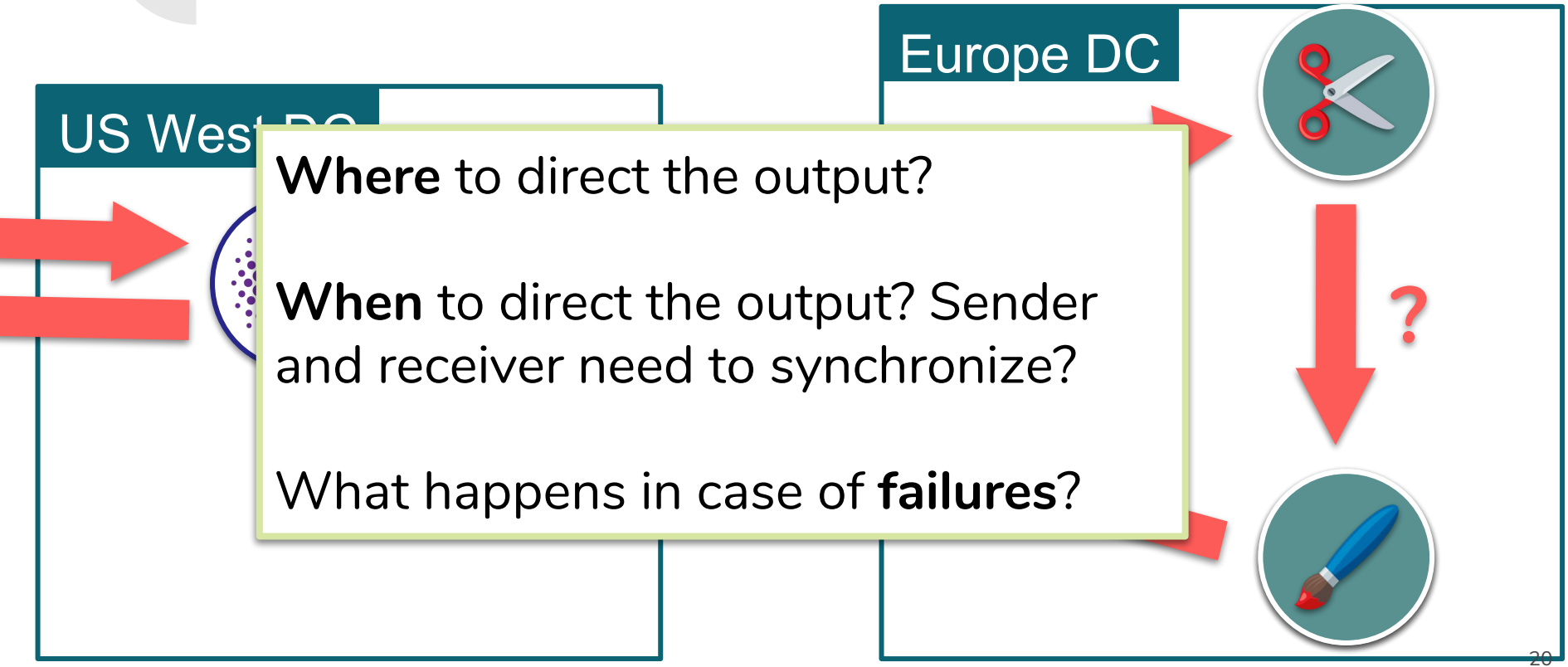
Latency ~158 ms * 4 = ~**632 ms**



Solution

Keep the **data** - the intermediate results - in the same datacenter.

Approach: direct pipelining

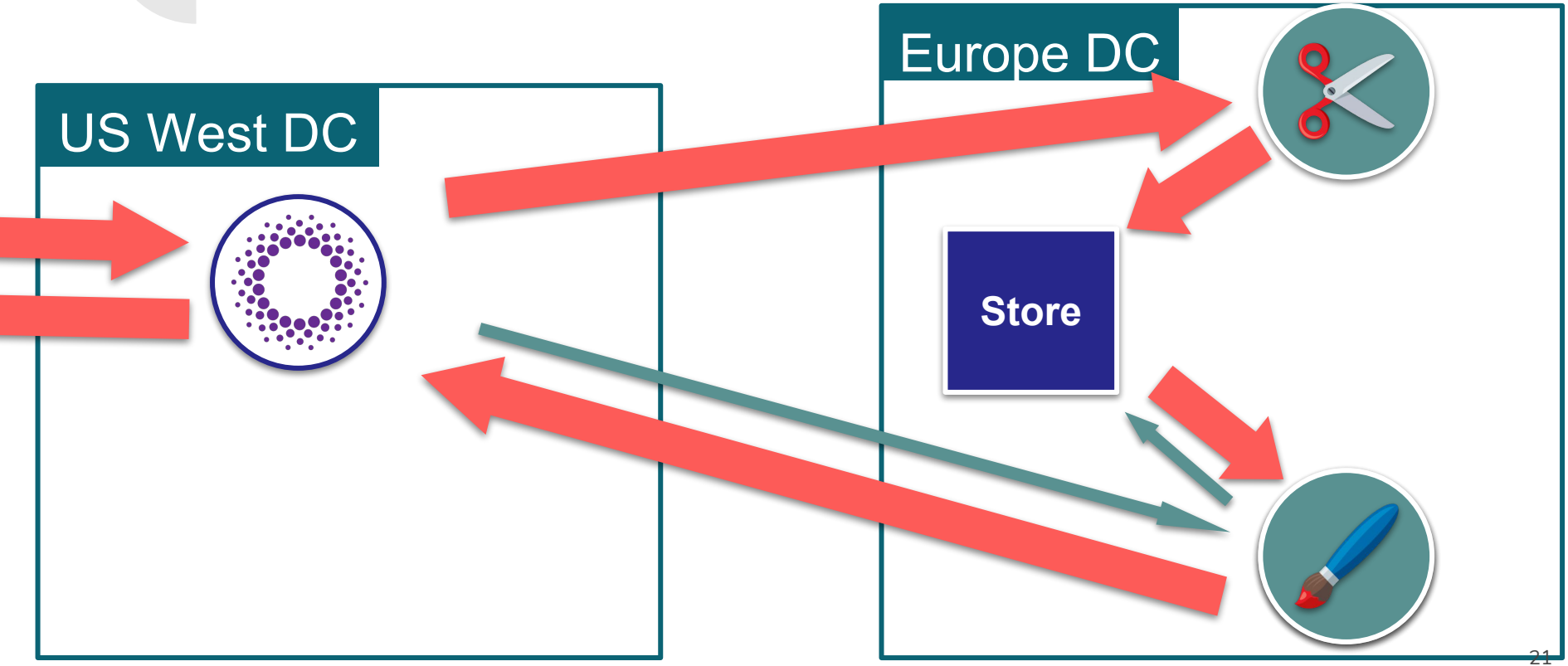


Where to direct the output?

When to direct the output? Sender and receiver need to synchronize?

What happens in case of **failures**?

Approach: "Intermediate Result" Store



“Intermediate Result” = Promise

A promise holds a (future) value; API:

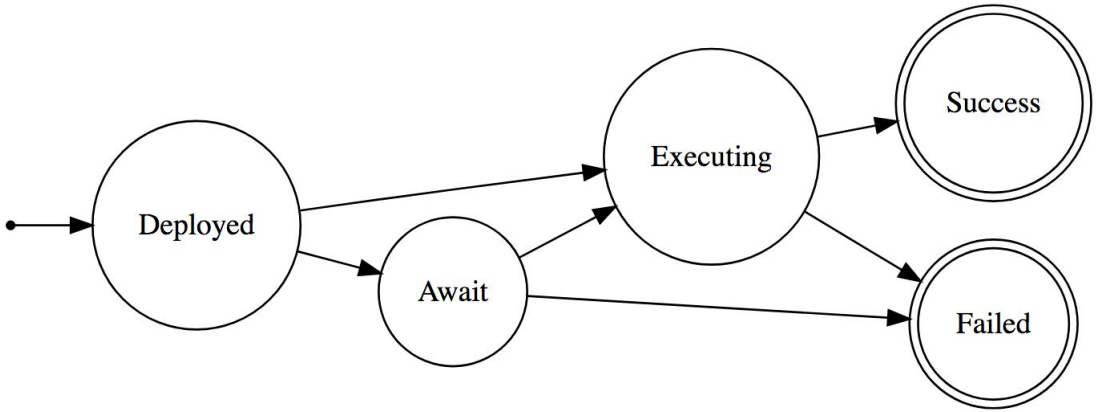
- Get() *Returns the value or error*
- Resolve(value) *Makes Get() return VALUE*
- Reject(error) *Makes Get() throw ERROR*

Optionally: *watch promise*

“Distributed” Promises

- Promises across the context of a single process
- “Promises allow a caller to run in parallel with a call and to pick up the results of the call, including any exceptions it raises.”
  - Liskov ‘88, *Distributed Programming in* ***Argus***
  - Liskov and Shrira ‘88, *Promises: Linguistic Support for Efficient Async Procedure Calls in Distributed Systems*

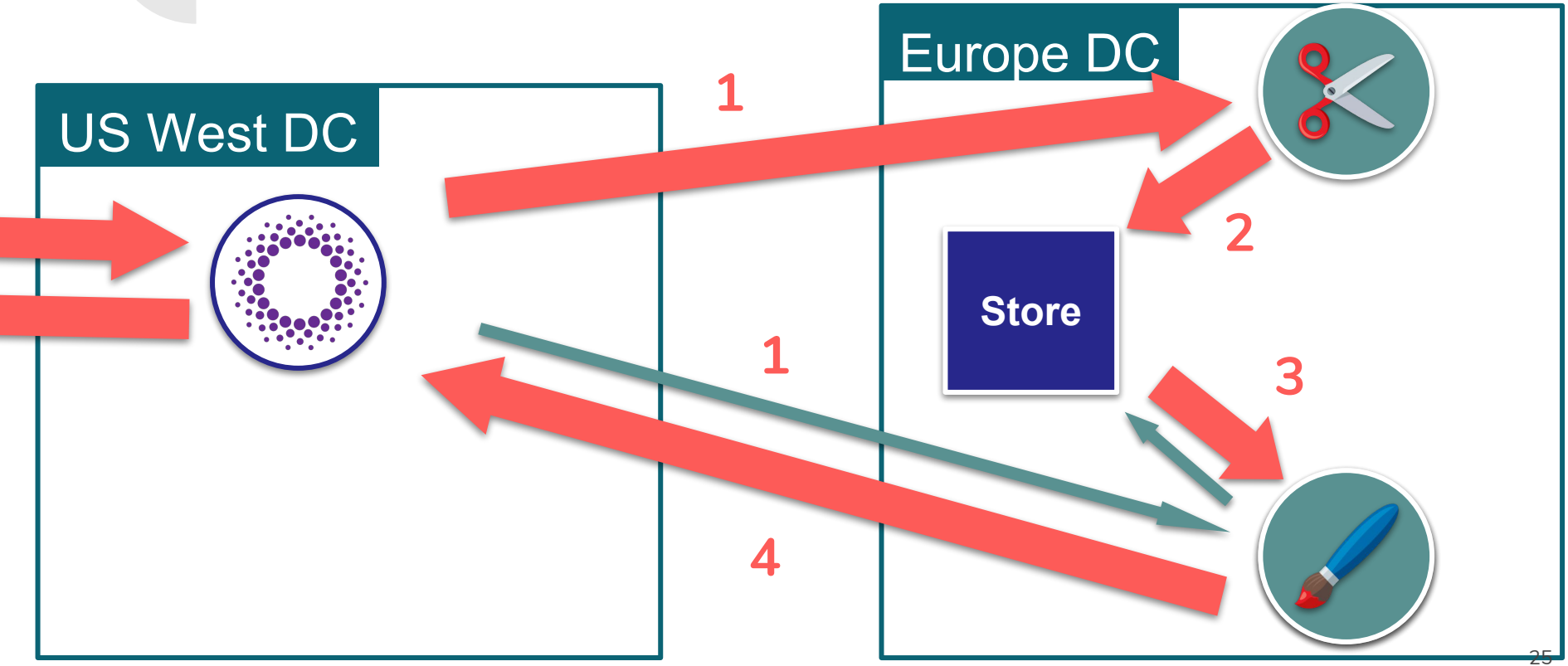
Promises in the FaaS Function model



Await:

- Promise is resolved → function starts executing.
- Promise is rejected → function fails without executing

Approach: Promise Store

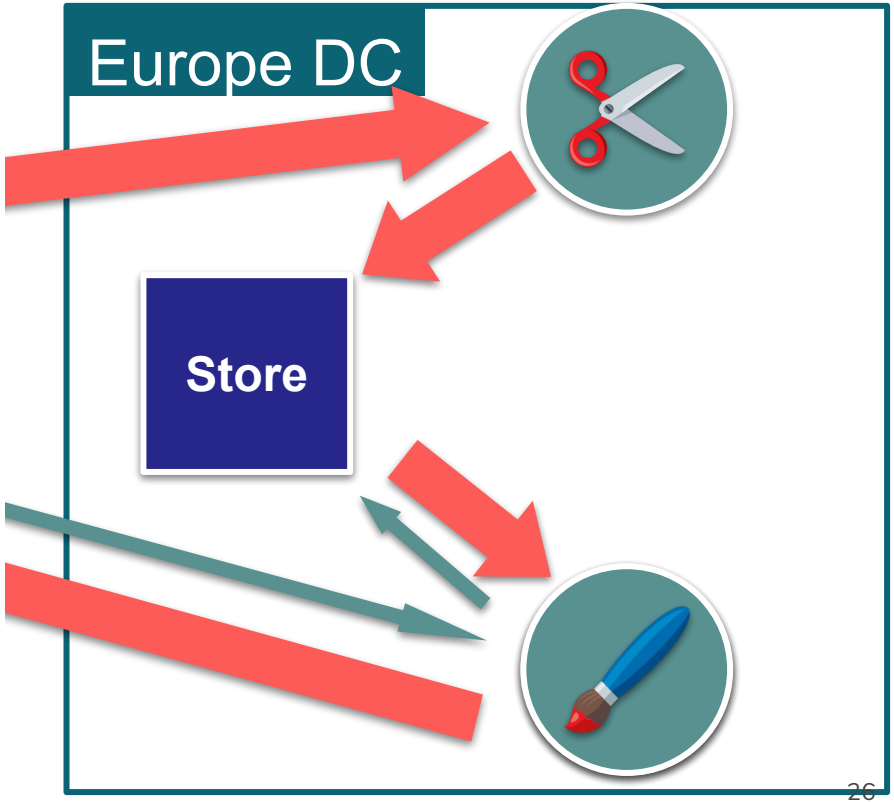


Approach: Promise Store

Costs € 0.09 * 2 = **€ 0.18** per GB

Latency ~158 ms * 2 = ~**316 ms**

Difference \w naive: 2 trips / 50%



Dual Optimization Problem

Computation: where do we execute the functions?

Data: where do we store the promises?

How do **computation** and **data** placements influence each other?

How to optimize the overall performance in presence of these two factors?

The use of promises is a promising candidate for data-intensive functions in FaaS.

Thank you!

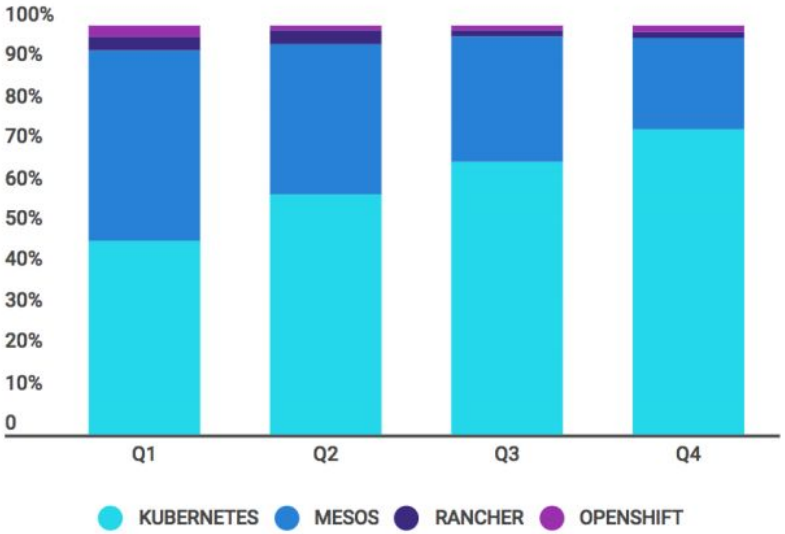
https://fission.io/ - https://fission.io/workflows

@erwinvaneyk

erwin@platform9.com

Additional Slides

Kubernetes is eating the resource orchestration ecosystem.



http://get.cloudability.com/ebook-state-of-cloud-2018.html

“Distributed” Promises

- Promises across the context of a single process
- See: the Argus system
  - Liskov ‘88, *Distributed Programming in Argus*
  - Liskov and Shrira ‘88, *Promises: Linguistic Support for Efficient Async Procedure Calls in Distributed Systems*