No More, No Less

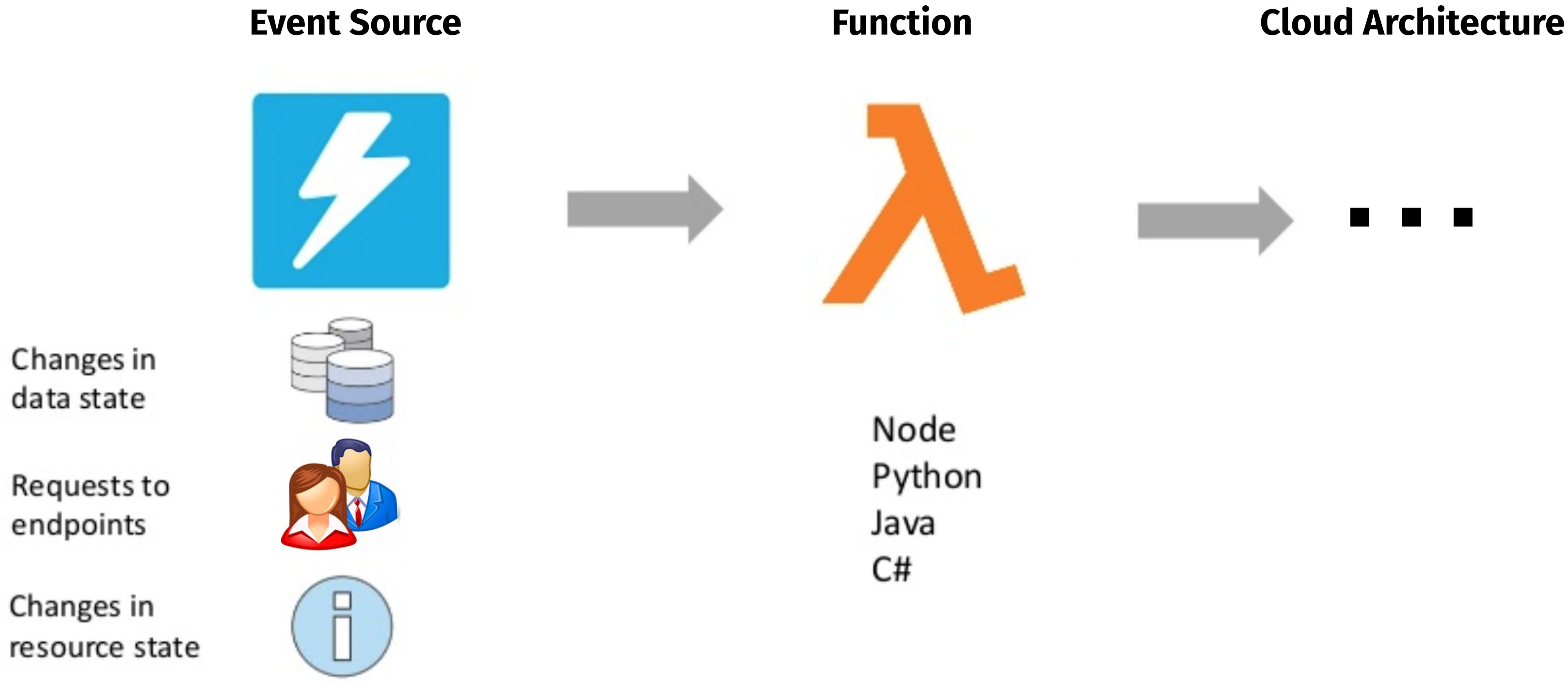
A Formal Model for Serverless Computing

Maurizio Gabbrielli, Ivan Lanese, Stefano Pio Zingaro
INRIA, France / Università di Bologna, Italy

Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti
University of Southern Denmark, Denmark

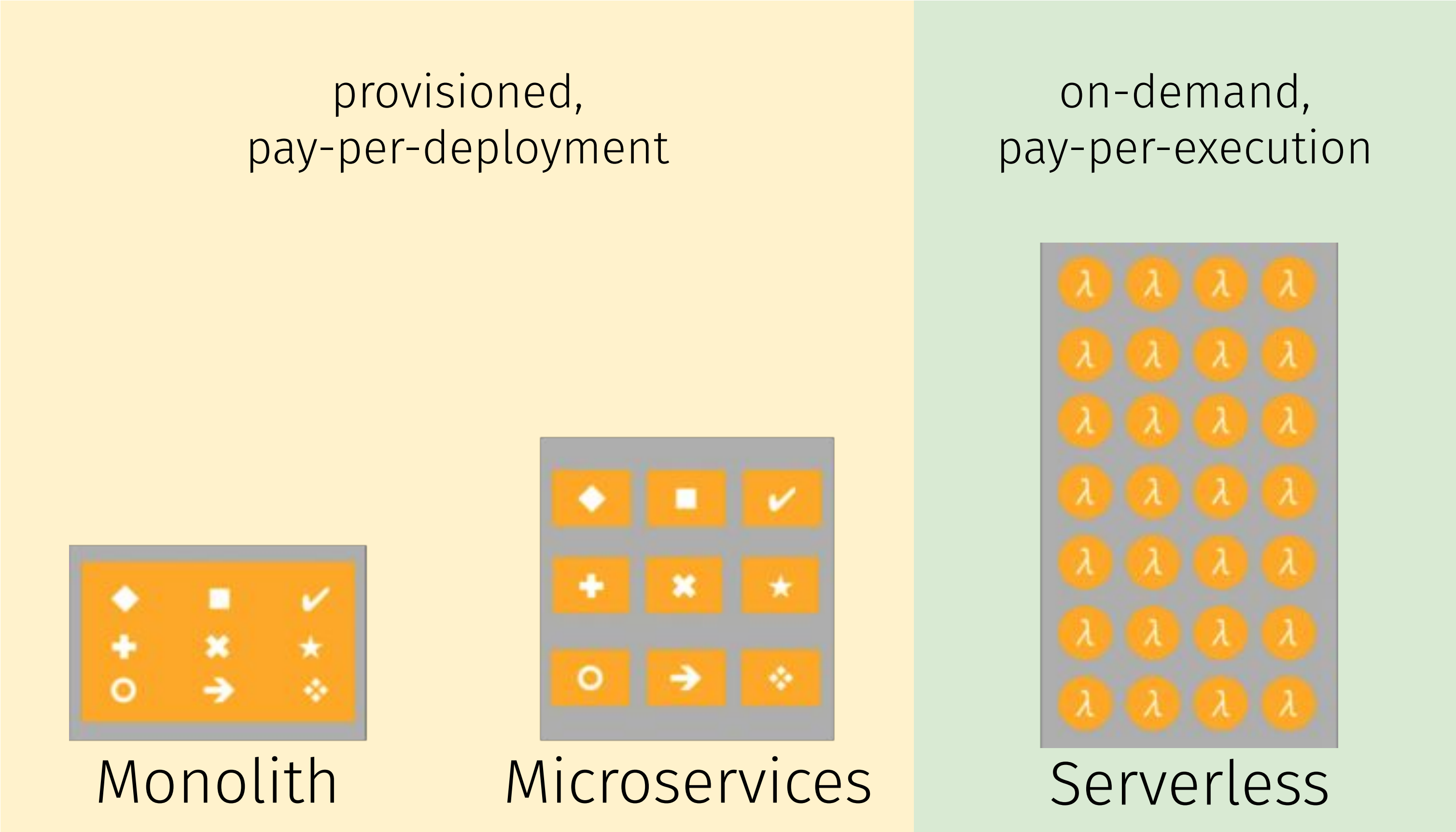
Coordination 2019

A gentle introduction to “serverless”…

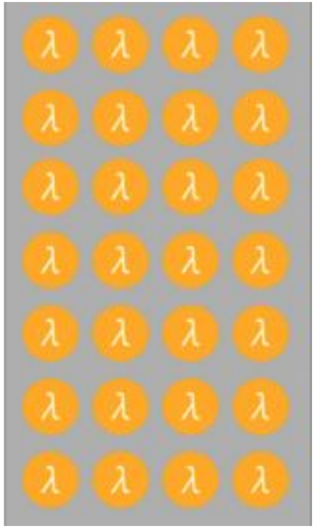


Adapted from “Serverless Architecture Patterns” by Abby Fuller, AWS

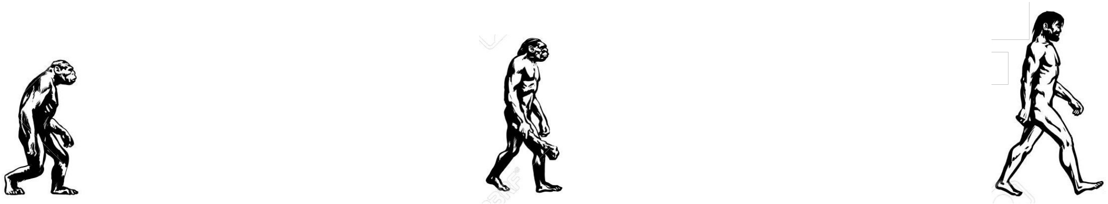
A gentle introduction to “serverless”…



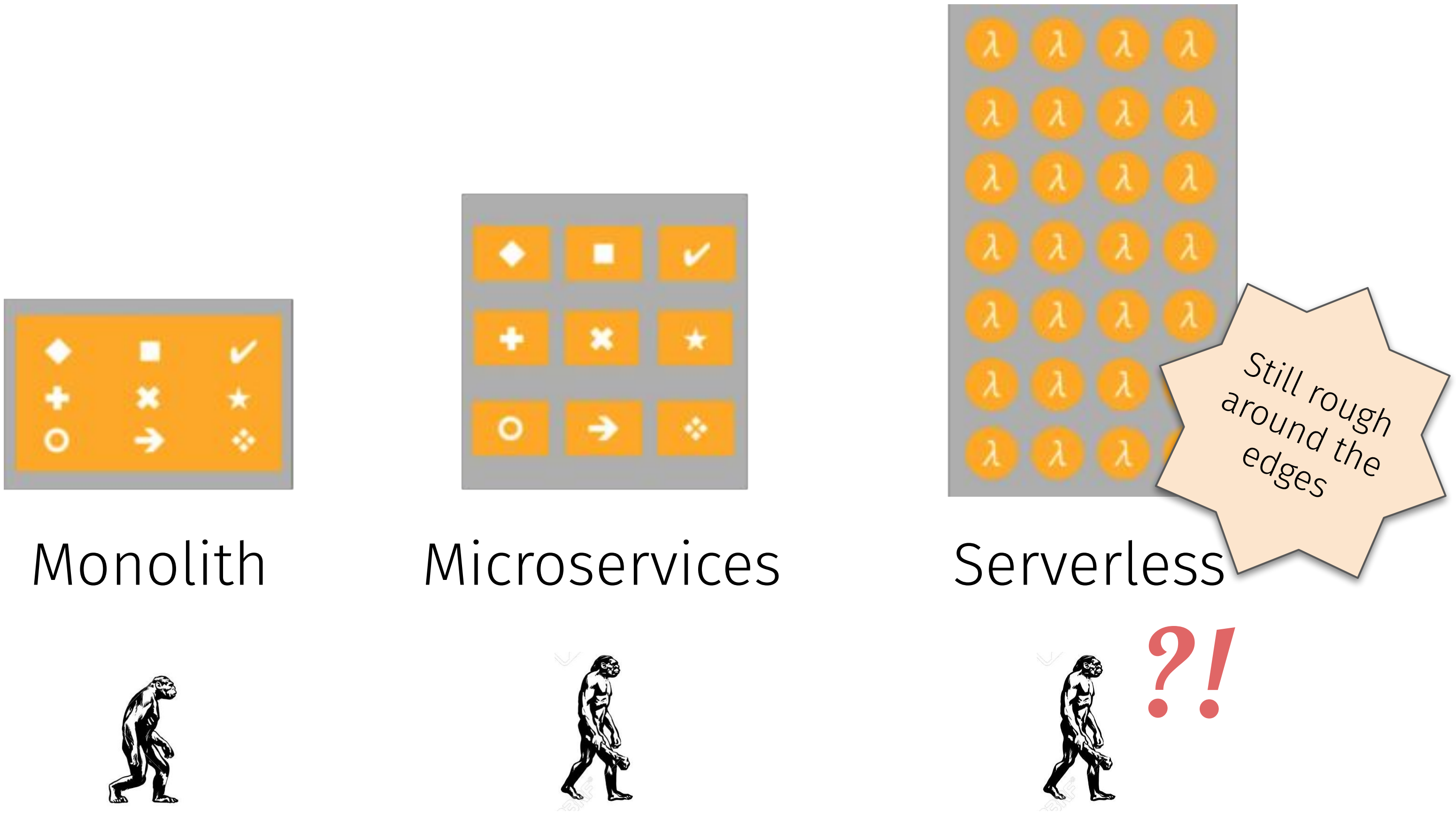
A gentle introduction to “serverless”…



Monolith Microservices Serverless



A gentle introduction to "serverless"…

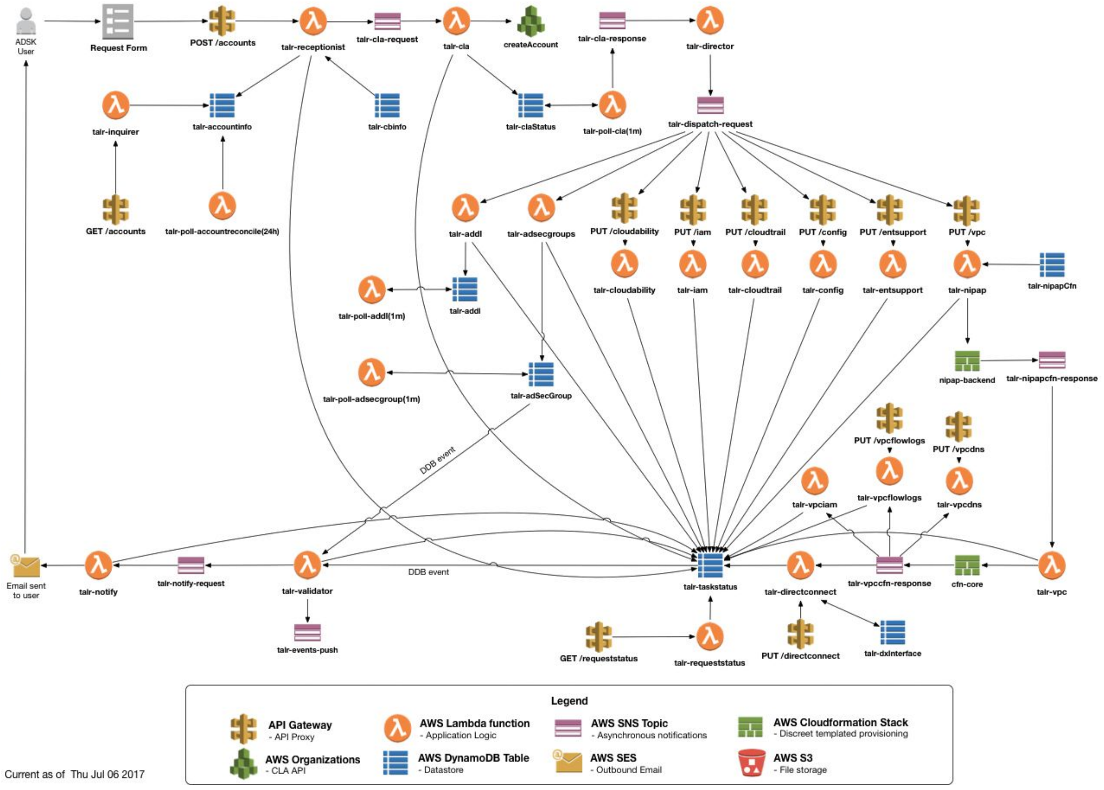


A gentle introduction to “serverless”...

AUTODESK

Tailor

Autodesk Goes Serverless in the AWS Cloud, Reduces Account-Creation Time by 99%



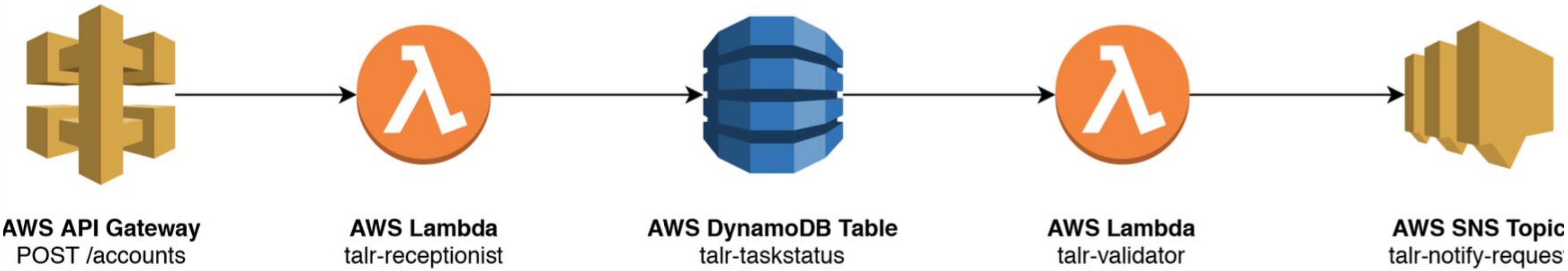
https://aws.amazon.com/serverless/

A gentle introduction to “serverless”…

AUTODESK

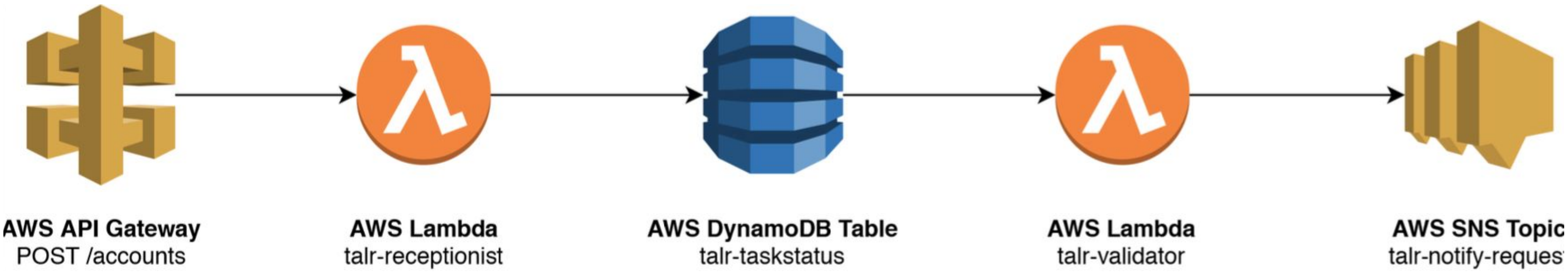
Tailor

an excerpt



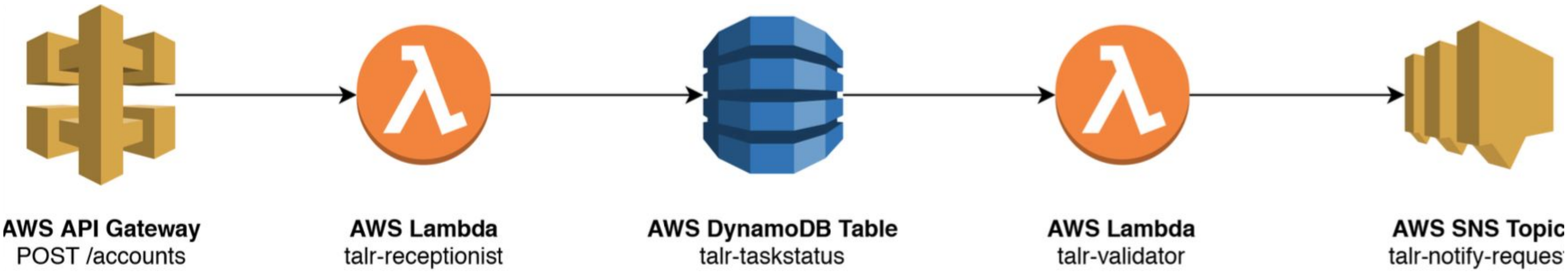
https://github.com/alanwill/aws-tailor

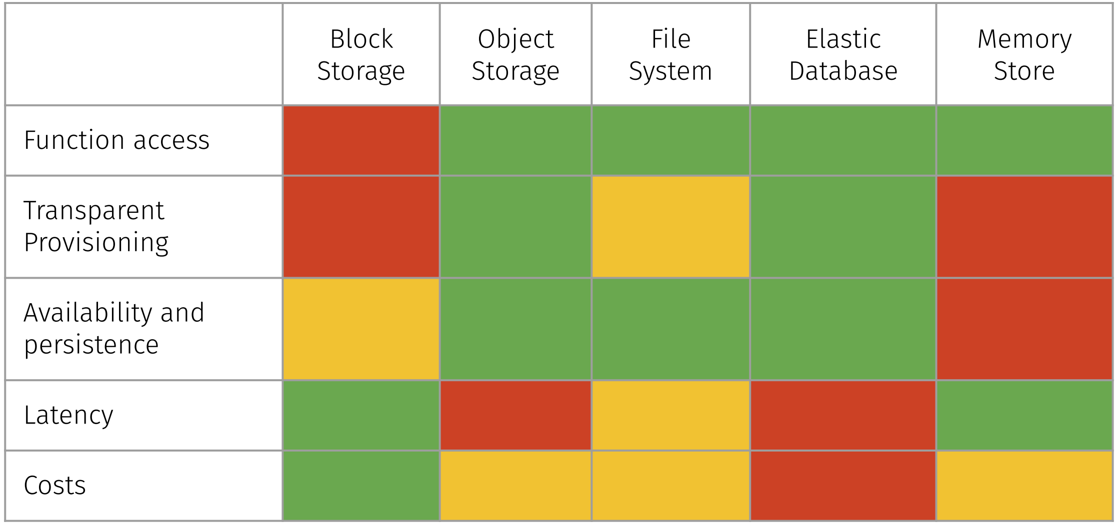
… and its current limitations (non-exhaustive)



- Currently ~15’ execution timeout;
- No function-to-function invocation. Functions need an in-between stateful service to call each other;
- Sparse coordination logic.

… and its current limitations (non-exhaustive)

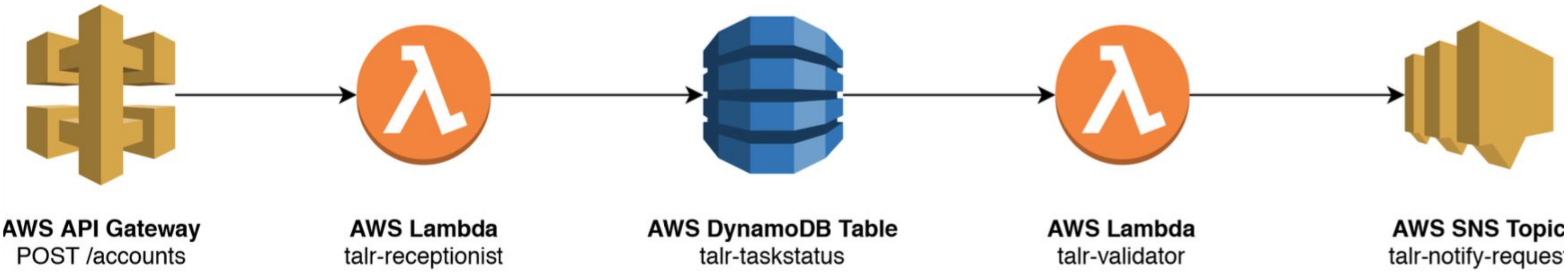




- Traffic-dependent scaling of functions implies:
    - complex cost model;
    - complex system load estimations.
- Poor performance for standard communication patterns

Jonas, Eric, et al. "*Cloud Programming Simplified: A Berkeley View on Serverless Computing.*"

… and its current limitations (non-exhaustive)



Lock-in due to absence of standards on:

- function support/execution environments;
- function call semantics;
- semantics of stateful services.

Direction

- We want to study S.C. avoiding any vendor/technology specifics
- We need a formal model for Serverless Computing that
  - Captures current incarnations of S.C.

    (e.g. event-based, storage-mediated as in AWS)
  - Supports proposed approaches/features

    (e.g. function-to-function invocation, updatable function definitions)

Serverless Kernel Calculus (SKC)

Serverless Kernel Calculus (SKC)

- Systems: $\langle \mathsf{S} \,, \mathscr{D} \rangle$

Serverless Kernel Calculus (SKC)

- Systems: $\langle \mathtt{S}, \mathscr{D} \rangle$

  Repository of function definitions

- Function definitions: $\mathtt{f} \mapsto \mathtt{M}$

  Function name

  λ-term

Serverless Kernel Calculus (SKC)

- Systems: $\langle \mathtt{S}, \mathscr{D} \rangle$

Network of running functions

Repository of function definitions

- Function definitions: $\mathtt{f} \mapsto \mathtt{M}$

Function name

λ-term

- Running functions: $\mathtt{c} \blacktriangleleft \mathtt{M}$

Promise

λ-term

SKC function terms

```
M ::= M M' | V

V ::= x | λx.M
```

SKC function terms

$$M ::= M\ M' \mid V$$

Function invocation

$$\mid f$$

$$\langle \mathcal{E}[f], \mathscr{D}[f \mapsto M] \rangle \longrightarrow \langle \mathcal{E}[M], \mathscr{D} \rangle$$

$$V ::= x \mid \lambda x.M$$

SKC function terms

$$M ::= M\ M' \mid V$$

Function invocation $\mid f$

Asynchronous eval $\mid \texttt{async}\ M$

$$\langle \mathcal{E}[\texttt{async}\ M], \mathcal{D} \rangle \longrightarrow \langle \mathcal{E}[c] \mid c \blacktriangleleft M, \mathcal{D} \rangle$$

$$V ::= x \mid \lambda x.M$$

Futures $\mid c$

$$\langle c \blacktriangleleft V \mid S, \mathcal{D} \rangle \longrightarrow \langle S[V/c], \mathcal{D} \rangle$$

SKC function terms

$$M ::= M\ M' \mid V$$

Function invocation $\mid f$

Asynchronous eval $\mid \texttt{async}\ M$

Function Repository Updates

$\mid \texttt{set}\ f\ M$ — $\langle \mathcal{E}[\texttt{set}\ f\ M], \mathscr{D}\rangle \longrightarrow \langle \mathcal{E}[f], \mathscr{D}[f \mapsto M]\rangle$

$\mid \texttt{take}\ f$ — $\langle \mathcal{E}[\texttt{take}\ f], \mathscr{D}[f \mapsto M]\rangle \longrightarrow \langle \mathcal{E}[M], \mathscr{D} \setminus f\rangle$

$$V ::= x \mid \lambda x.M$$

Futures $\mid c$

SKC function terms

λ + Futures + Function Repository

```
M ::= M M' | V
```

Function invocation

```
  | f
```

$\langle \mathcal{E}[\mathtt{f}], \mathcal{D}[\mathtt{f} \mapsto \mathtt{M}] \rangle \longrightarrow \langle \mathcal{E}[\mathtt{M}], \mathcal{D} \rangle$

Asynchronous eval

```
  | async M
```

$\langle \mathcal{E}[\mathtt{async\ M}], \mathcal{D} \rangle \longrightarrow \langle \mathcal{E}[\mathtt{c}] \mid \mathtt{c} \blacktriangleleft \mathtt{M}, \mathcal{D} \rangle$

Function Repository Updates

```
  | set f M
```

$\langle \mathcal{E}[\mathtt{set\ f\ M}], \mathcal{D} \rangle \longrightarrow \langle \mathcal{E}[\mathtt{f}], \mathcal{D}[\mathtt{f} \mapsto \mathtt{M}] \rangle$

```
  | take f
```

$\langle \mathcal{E}[\mathtt{take\ f}], \mathcal{D}[\mathtt{f} \mapsto \mathtt{M}] \rangle \longrightarrow \langle \mathcal{E}[\mathtt{M}], \mathcal{D} \setminus \mathtt{f} \rangle$

```
V ::= x | λx.M
```

Futures

```
  | c
```

$\langle \mathtt{c} \blacktriangleleft \mathtt{V} \mid \mathtt{S}, \mathcal{D} \rangle \longrightarrow \langle \mathtt{S}[\mathtt{V}/\mathtt{c}], \mathcal{D} \rangle$

Programmable events in SKC

- Store handlers for event e in the definition repository $\mathscr{D}$:

```
install_handler ↦ λe.λhandler.
     let old_handler = take e
     let new_handler = λv.do async (handler v)
                                  (current_handler v)
     set e new_handler
```

- Raise event e (with v) by invoking its handlers in $\mathscr{D}$:

```
e v
```

- (See the paper for AUTODESK Tailor in SKC)

Programmable events in SKC

- Store handlers for event e in the definition repository $\mathscr{D}$:

```
install_handler ↦ λe.λhandler.
     let old_handler = take e
     let new_handler = λv.do async (handler v)
                                (current_handler v)
     set e new_handler
```

Updatable function definitions

- Raise event e (with v) by invoking its handlers in $\mathscr{D}$:

```
e v
```

- (See the paper for AUTODESK. Tailor in SKC)

Programmable events in SKC

- Store handlers for event e in the definition repository $\mathscr{D}$:

```
install_handler ↦ λe.λhandler.
      let old_handler = take e
      let new_handler = λv.do async (handler v)
                                    (current_handler v)
      set e new_handler
```

Updatable function definitions

- Raise event e (with v) by invoking its handlers in $\mathscr{D}$:

```
e v
```

Function-to-function invocation

- (See the paper for AUTODESK. Tailor in SKC)

Conclusions and future work

We introduced SKC a Kernel Calculus for Serverless Computing:

- Build on established models (λ-calculus + futures + function repository)
- captures current programming models
- supports next-gen features e.g. function-to-function invocation

Serverless: current challenges	SKC: research direction
The coordination logic is spare and loosely-consistent	Choreographic Programming targeting SKC.
Estimation of performance and costs is complex	Quantitative SKC

Thanks for your attention

No *M*ore, No Less

λ + Futures + Function Repository

=

A formal model for Serverless Computing