Cloud Computing Technologies

DAT515 - Fall 2024

Serverless Functions

Prof. Hein Meling

University of Stavanger


Serverless Functions
Serverless
FUNCTIONS
Simple coding
Events
Evet Events
Serverless
Functions
& Event Triggers
Fime Uploads
Databasnges
File Uploads
File Events
Web Requase
HHP
H4B
REQUEST
HIP Requase
Database Changes
Database changes

What are Serverless Functions?

What are Serverless Functions?

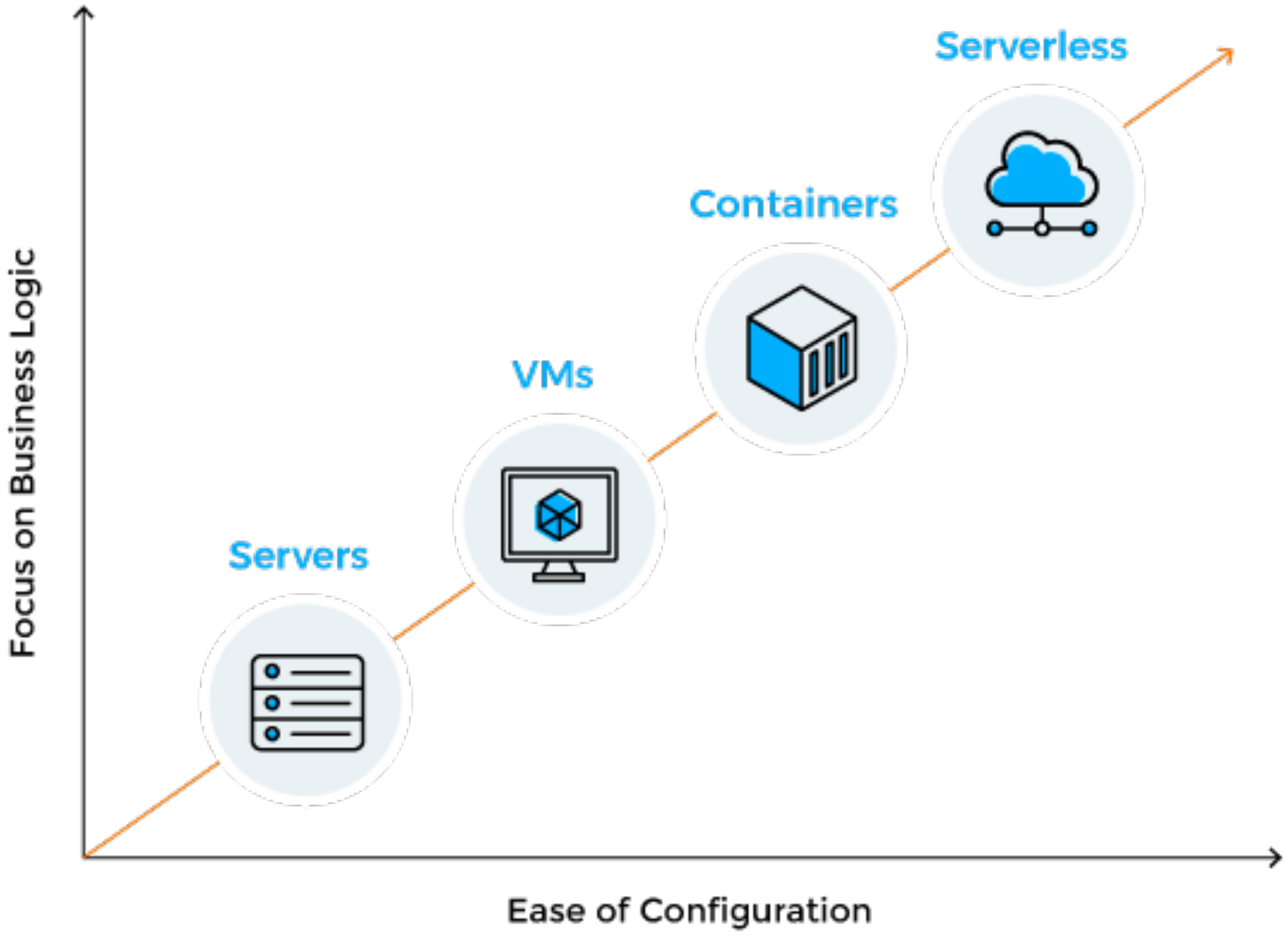
Definition of the Serverless Model

- Functions-as-a-Service (FaaS)
  - Specify **function** definition and **events** that triggers it
- Automate
  - **Provisioning** and **scaling** of machines
  - **Distribution** of functions over machines

What are Serverless Functions?

Benefits of the Serverless Model

- Reduces development effort
- Provides elastic scaling
- Cost-effective



Service Model Comparison

On-site	IaaS	PaaS	SaaS
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
Operating System	Operating System	Operating System	Operating System
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Service Model Comparison

On-site	IaaS	PaaS	FaaS	SaaS
Functions	Functions	Functions	Functions	Functions
Applications	Applications	Applications	Applications	Applications
Data	Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware	Middleware
Operating System	Operating System	Operating System	Operating System	Operating System
Virtualization	Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking	Networking

What are Serverless Functions?

Example Serverless Function and Trigger

```python
import azure.functions as func
import urllib.request

def main(req: func.HttpRequest):
    url = req.get_body().decode("utf-8")
    fid = urllib.request.urlopen(url)
    webpage = fid.read().decode('utf-8')
    found_free = webpage.find("free") >= 0
    return str(found_free)
```

```json
"bindings": [ { "authLevel": "anonymous",
                "name": "req",
                "type": "httpTrigger",
                "direction": "in",
                "route": "checkWebPage",
                "methods": [ "get" ] },
              { "name": "$return",
                "type": "http",
                "direction": "out" } ]
```

- Can scale out automatically to handle 100s of kops per second
- Operate cheaply under low load — due to load-based billing

When to use Serverless Functions

Ideal Use Cases

- Compute-intensive highly parallelizable workloads
- Workflow processing
- Event-driven applications
- Microservices
- Real-time data processing

Not Ideal For

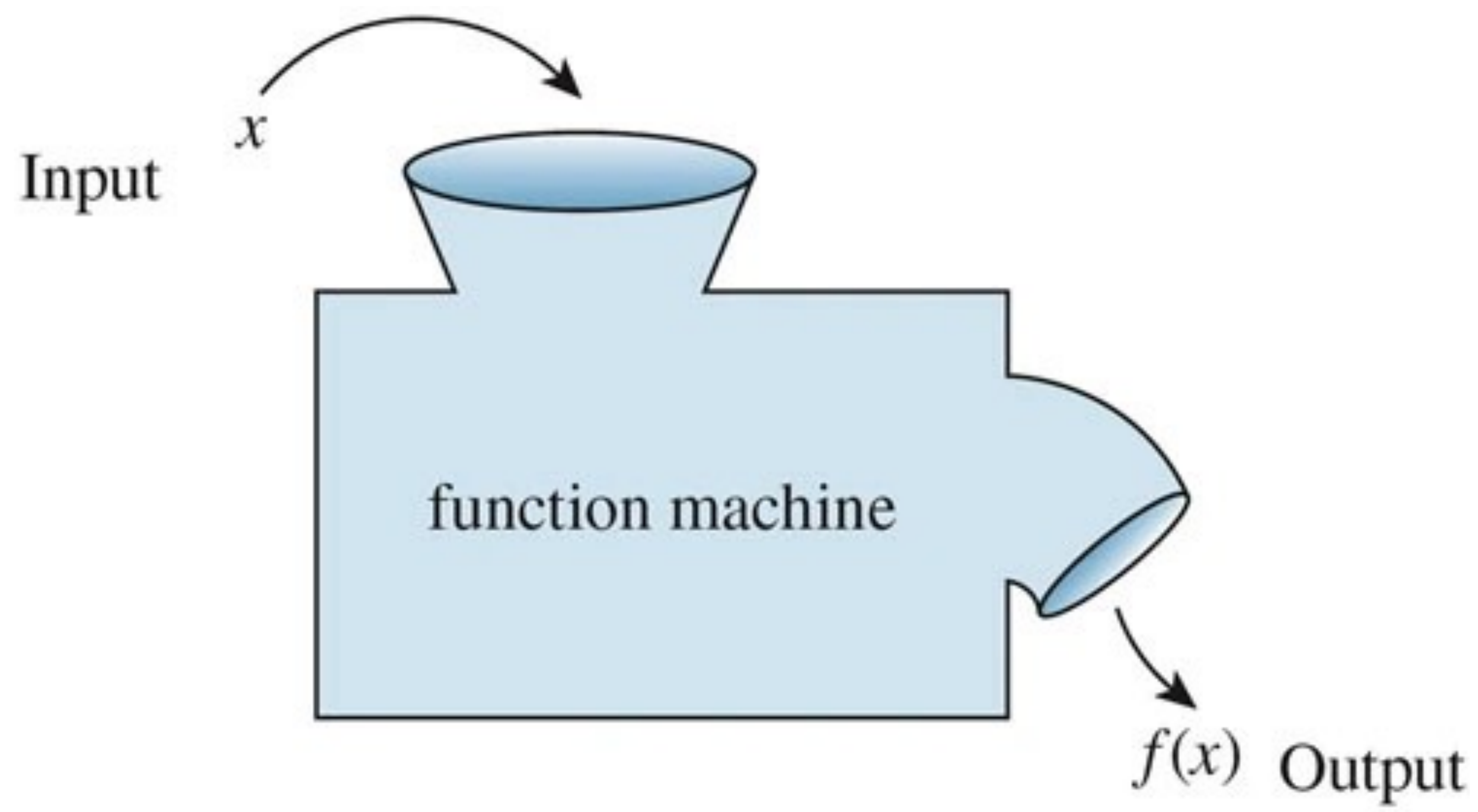
- Long-running processes
- High performance (over time)

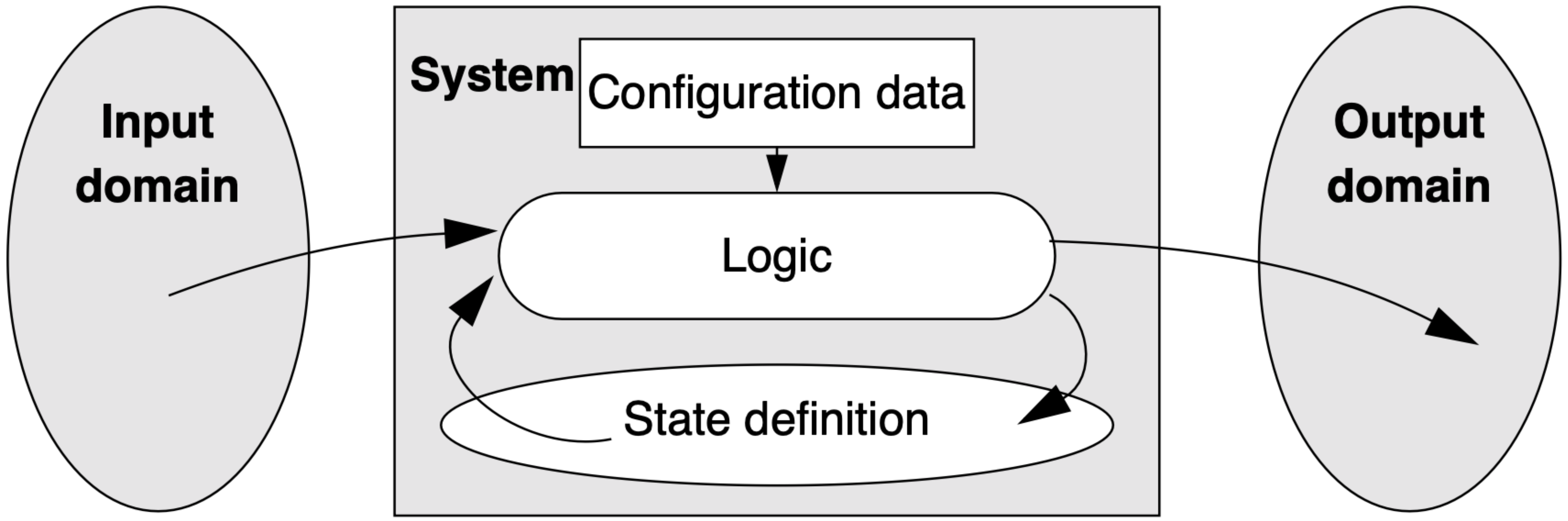
Decision Factors

- Scalability needs
- Cost considerations
- Development speed and flexibility

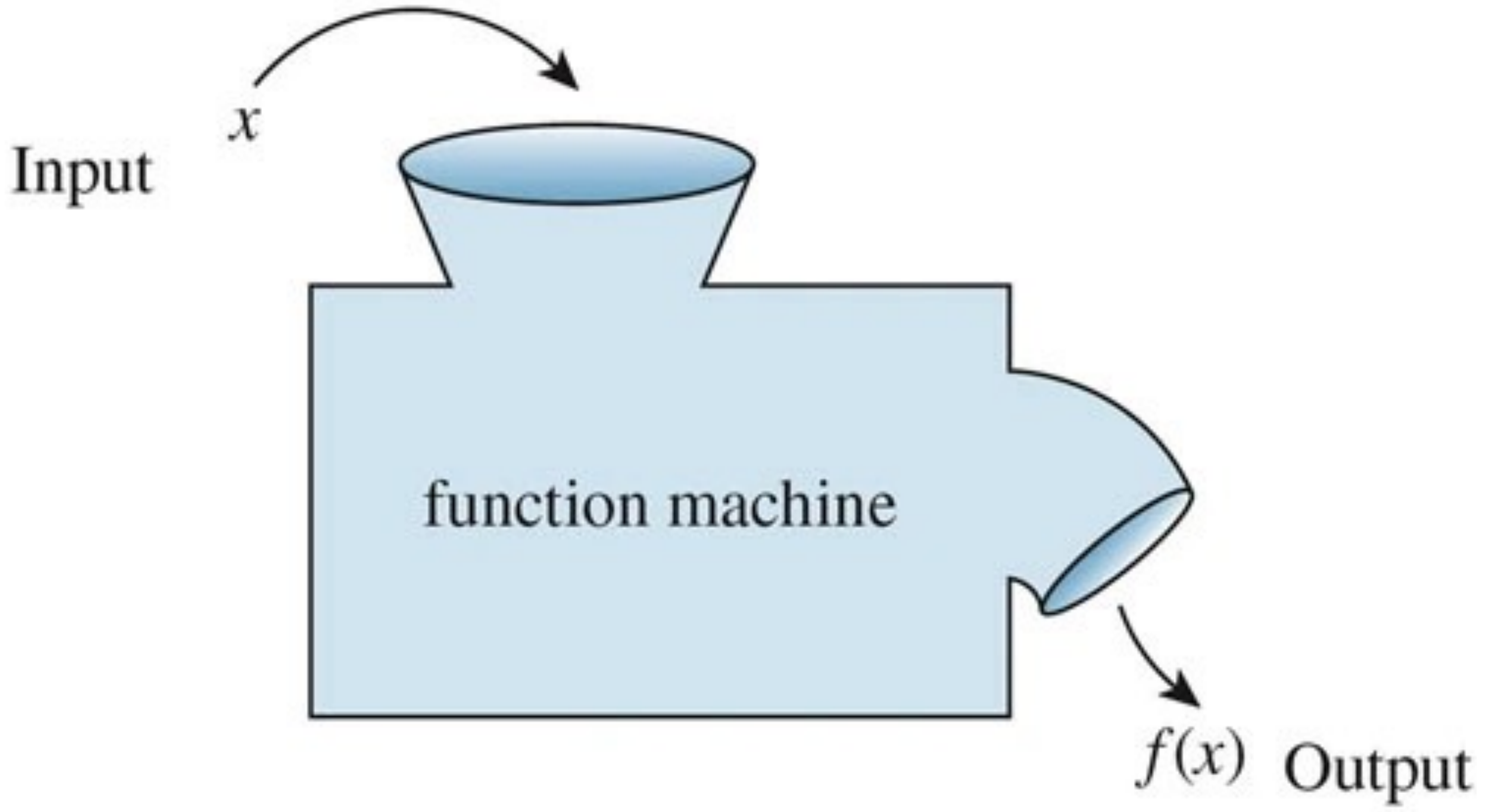
The Function Machine

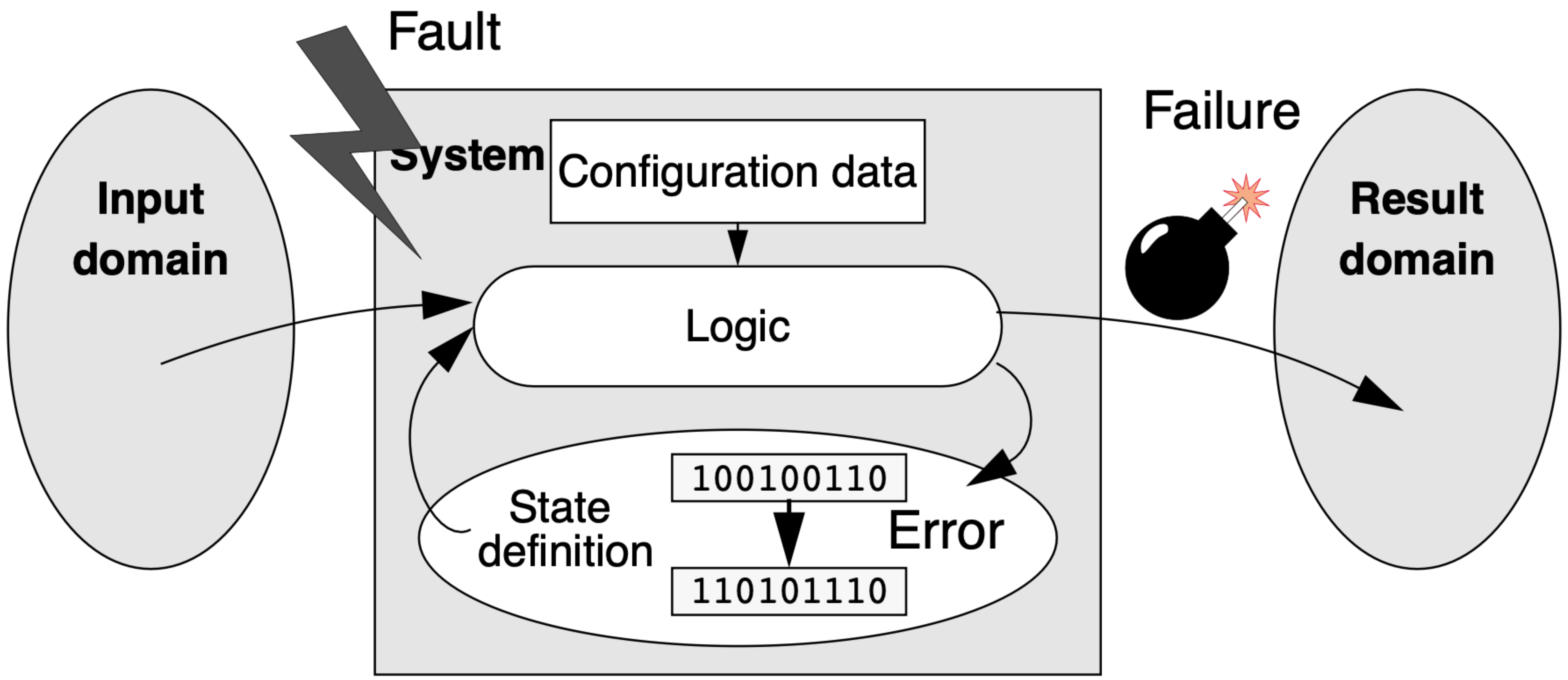
Function Machine vs Moore-Mealy Machine





Function Machine vs Moore-Mealy Machine

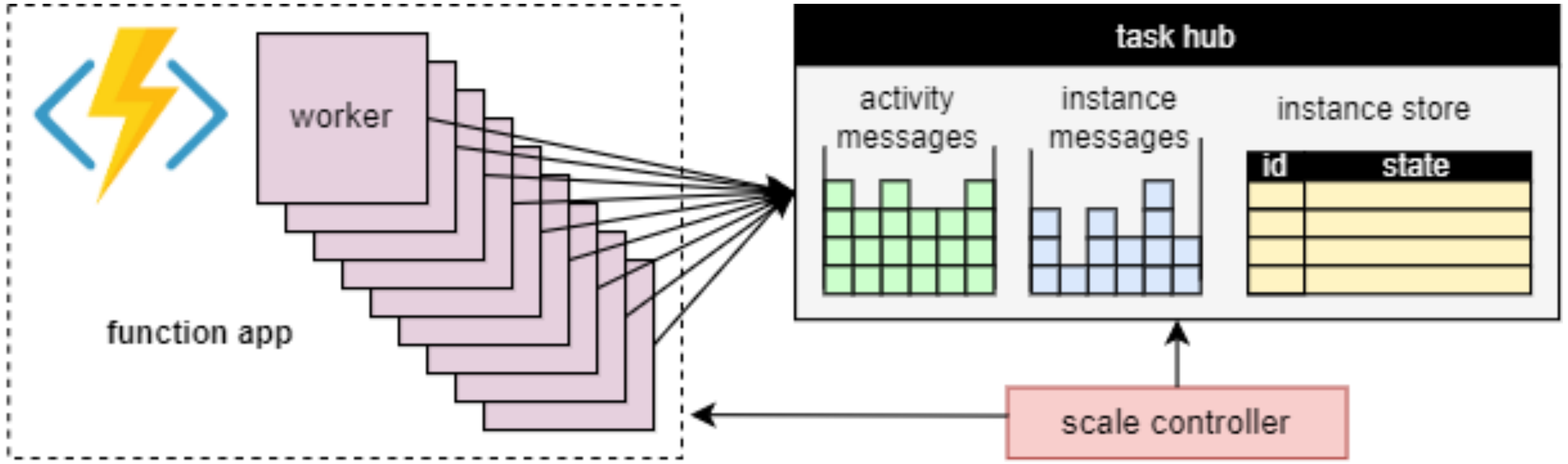




Serverless Assumption

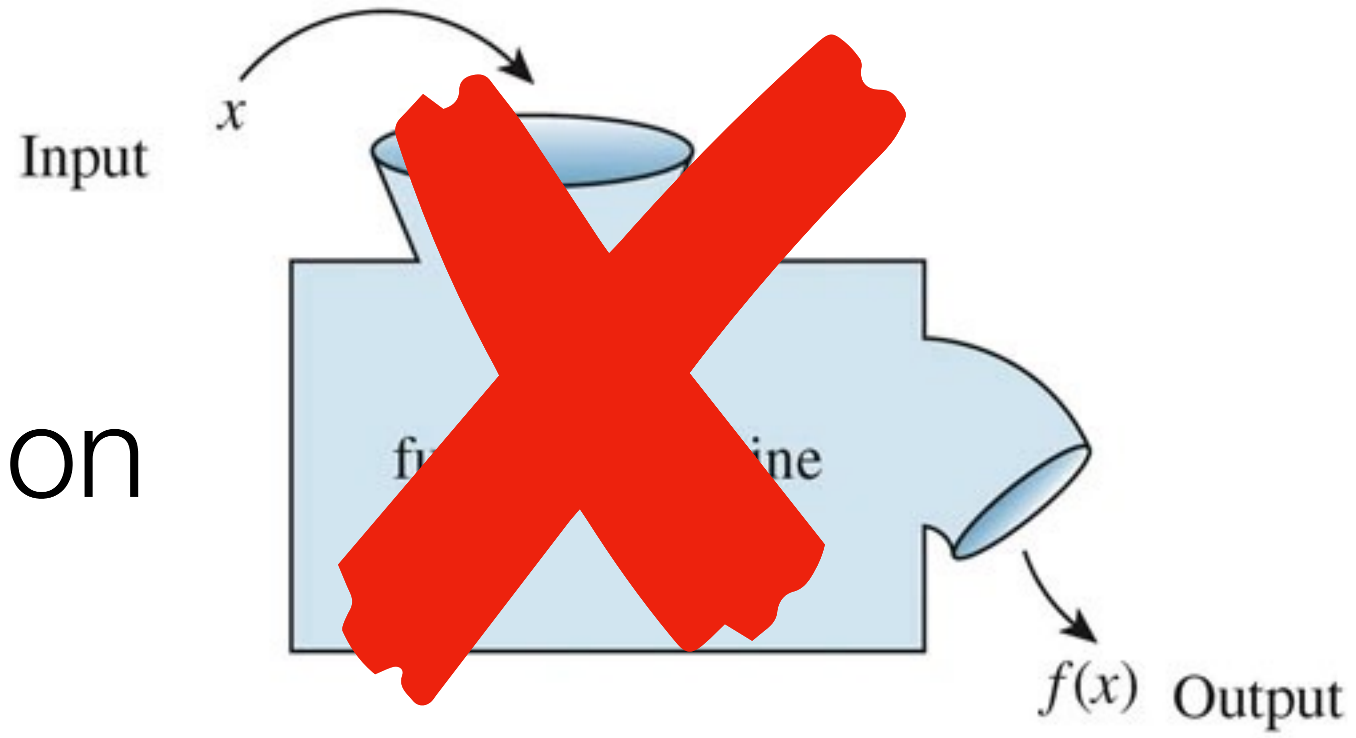
The Compute-Storage Separation

- Is *becoming* a dominant **architectural principle** for cloud services
- Separating the application into **ephemeral workers** and a **storage** service
  - State of the application decoupled from compute
  - Making the state available when a worker crashes (or is shut down due to low load)
  - Scale workers independently of storage

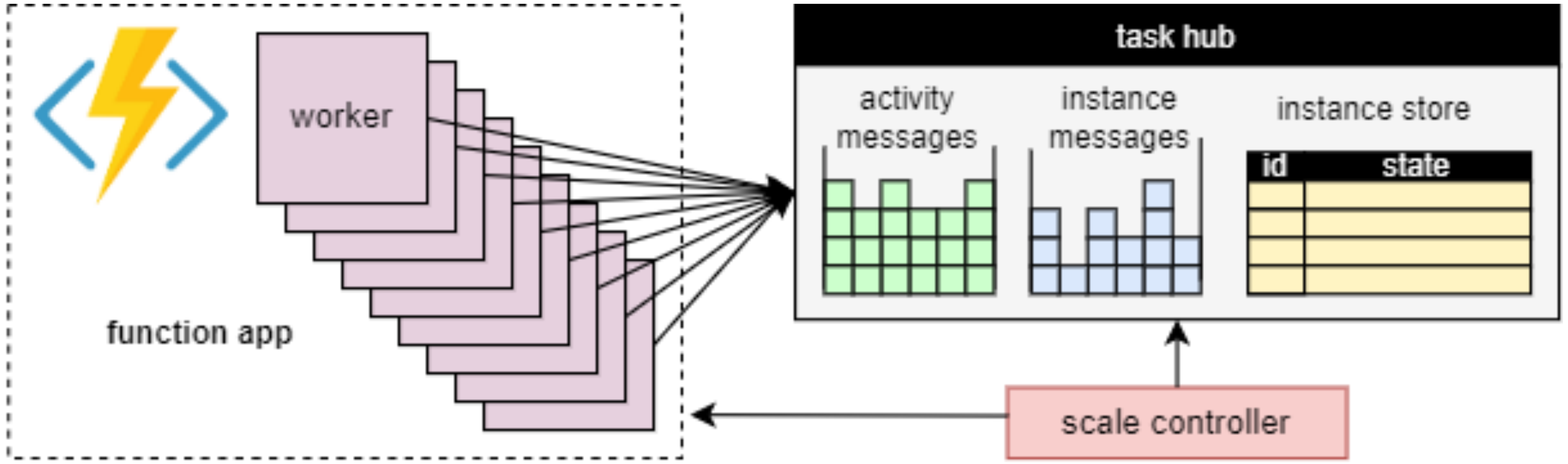


Serverless Assumption

The Compute-Storage Separation



- Not limited to stateless input/output computation
- Functions can call **external services**
    - Key-value stores, queues, or databases
- Compute-storage separation can be **cumbersome** for developers



Joke time

Why did the developer go serverless?

Because she didn't want to waste cycles on managing servers!

Serverless Developer Challenges

Serverless Challenges

Cloud Environment is Inherently and Pervasively

- Concurrent
- Parallel
- Distributed
- Failure-prone

Serverless Developer Challenges

Weak Execution Guarantees for Serverless Functions

- **Execution Time Limit**, e.g., 5 minutes
  - Problem for long-running computations that don't finish
- **Partial Execution**
  - Functions are subject to **failures** (OOM error, VM shut down, HW)
- **At-Least-Once Triggers**
  - Triggers designed to **retry** if uncertain if Function completed
  - Single event can result in **multiple Function executions**

Threats to Consistency: Non-Atomic Updates

- Example
  - Function must update two account balances (in multiple steps)
  - Partial execution may update only one account

Threats to Consistency: Concurrency Control

- Functions are inherently parallel, which can cause **races**
- Example

```
def main(message: QueueTrigger) -> bool:
    current = storage.read("messagecount");
    current = current + 1;
    storage.write("messagecount", current);
```

  - Two concurrent invocations may interleave so that
  - Counter incremented only once
- Need **external synchronization** mechanism

Threats to Consistency: Effect Duplication

- At-least-once triggers can lead to effect duplication
- Example

```
def main(message: QueueTrigger) -> bool:
    current = storage.read("messagecount");
    current = current + 1;
    storage.write("messagecount", current);
```

  - Over-count number of messages if
  - Invoked multiple times for same message
- Standard advice: Make function **idempotent**
  - Not always easy!

Stateful Abstractions

The Promise of Serverless

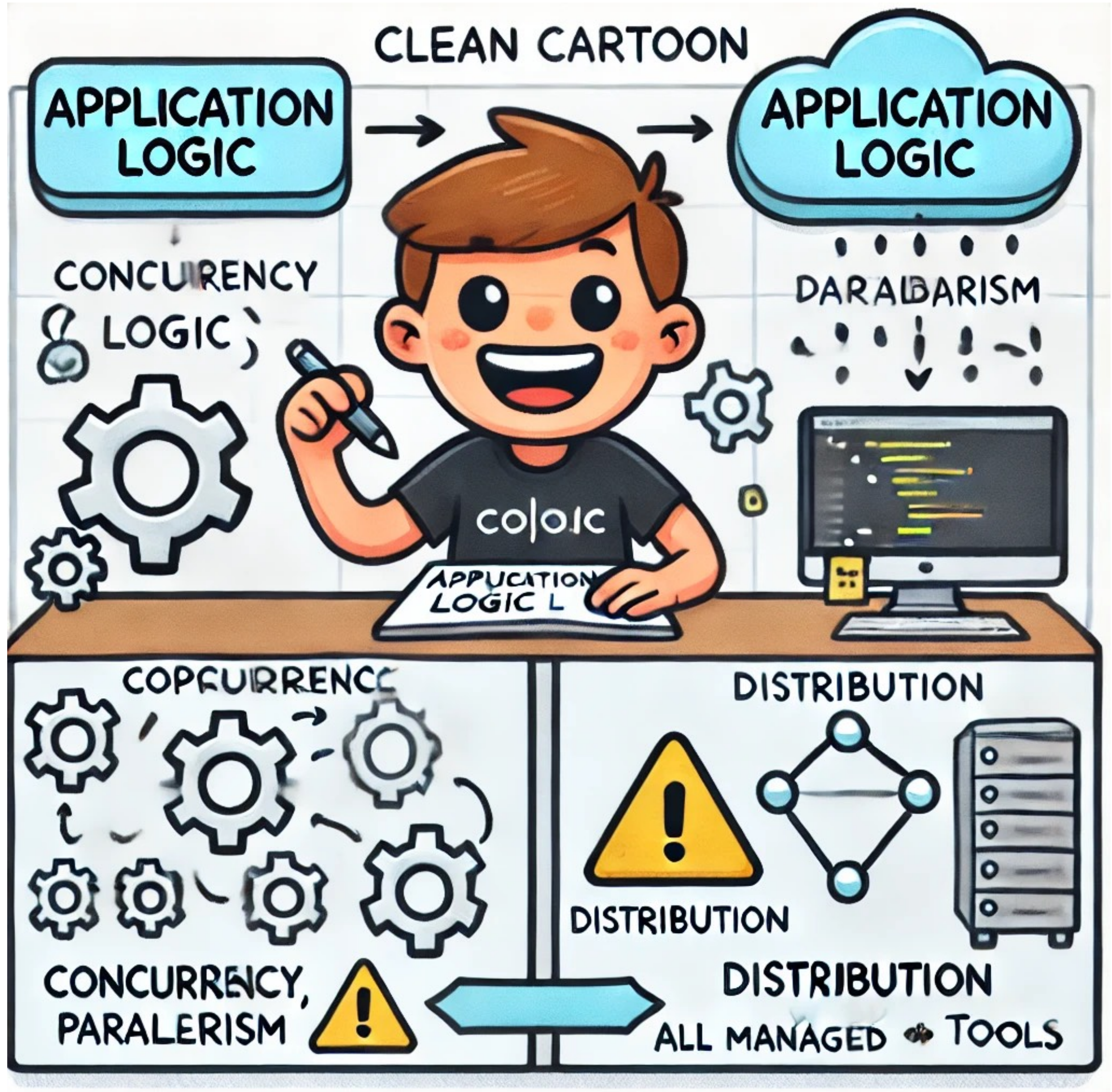
Cloud Environment is Inherently and Pervasively

- Concurrent
- Parallel
- Distributed
- Failure-prone

The Promise of Serverless

Challenging to Achieve

- Clean separation of
  - Application logic from
  - Concurrency, parallelism, distribution, and failures



The Promise of Serverless

Motivates Augmented Programming Model

- **Durable Functions** need
- Abstractions for **state** and **synchronization**
  - Hide challenges in the runtime
  - Simplify application development

Joke time

What did the event say to the serverless function?

"I've triggered something special in you!"

Durable Functions

Durable Functions

Microsoft's FaaS Platform

- Durable Functions is a component of Azure Functions
- Supports: Python, C#, JavaScript, PowerShell…

Programming Model

- **Activities** — durable equivalent of stateless FaaS functions
- **Entities** — actors that
  - Encapsulate application state
  - Process operations one at a time
- **Orchestrations** — task-parallel async-await style code to
  - Coordinate **activities** and **entities**

Programming Model

- Durability is implicit with Durable Functions
  - **State** of entities **automatically** saved to storage
  - **Transparently** restored after faults

Activities

- Automatically retried — under partial execution
- Unless time limit exceeded

```python
import azure.functions as func
import urllib.request

def main(req: func.HttpRequest):
    url = req.get_body().decode("utf-8")
    fid = urllib.request.urlopen(url)
    webpage = fid.read().decode('utf-8')
    found_free = webpage.find("free") >= 0
    return str(found_free)
```

```json
"bindings": [ { "authLevel": "anonymous",
                "name": "req",
                "type": "httpTrigger",
                "direction": "in",
                "route": "checkWebPage",
                "methods": [ "get" ] },
              { "name": "$return",
                "type": "http",
                "direction": "out" } ]
```

Orchestrations

- Decompose computation into tasks: *activities, entity operations, timers, sub-orchestrations*
- Example: Sequencing three activities DownloadData, Process, and Summarize

```python
def orchestrate_pipeline(context: df.DurableOrchestrationContext):
    try:
        dataset = yield context.call_activity('DownloadData')
        outputs = yield context.call_activity('Process', dataset)
        summary = yield context.call_activity('Summarize', outputs)
        return summary
    except Exception as exc:
        yield context.call_activity('CleanUp')
        return f"Something went wrong" {exc}"
```

Durable Functions

Entities

- Allow applications to **encapsulate** durable state
- Define **operations** on them
  - Operation requests stored in a queue
  - Execute them one at time

```python
class Account:
    # Initialize the entity state, which defaults to 0 balance
    def __init__(self):
        self.balance = 0

    # Return the balance
    def get(self):
        return self.balance

    # Deposit an amount, increasing the balance
    def deposit(self, amount):
        self.balance += amount
        return self.balance

    # Withdraw an amount, reducing the balance
    def withdraw(self, amount):
        self.balance -= amount
        return self.balance
```

Critical Sections

Concurrent Invocations Require Atomicity

- Invocations across **entities** must happen at once, atomically to
  - Prevent violating consistency
- Critical sections — regions of code where
  - Only **one orchestration** can call specific entities

Concurrent Invocations Require Atomicity

```
def transfer_safe_orchestration(context: df.DurableOrchestrationContext):
    # From input, get entity ids for source and destination account, and the amount
    [source, dest, amount] = context.get_input()
    # Critical Section: Acquire a lock for each entity
    with (yield context.lock([source, dest])):
        # Make sure that the source account has adequate balance to avoid overdraft
        source_balance = yield context.call_entity(source, "get")
        if source_balance < amount:
            return False
        else:
            # Wait for transfer to complete
            yield context.task_all([context.call_entity(source, "withdraw", amount),
                                    context.call_entity(dest, "deposit", amount)]);
            return True
```

Serverless Technologies

When to use Serverless Functions

Proprietary

- AWS Lambda
- Azure Functions
- Google Cloud Run

Edge Computing

- Cloudflare Workers
- Fastly Compute@Edge
- AWS Lambda@Edge

Open-Source

- OpenFaaS
- Apache OpenWhisk
- Kubeless
- Knative
- Fission
- Serverless Framework (Framework Agnostic)

Questions?

References

- Burckhardt et al, Durable Functions: Semantics for Stateful Serverless, OOPSLA 2021