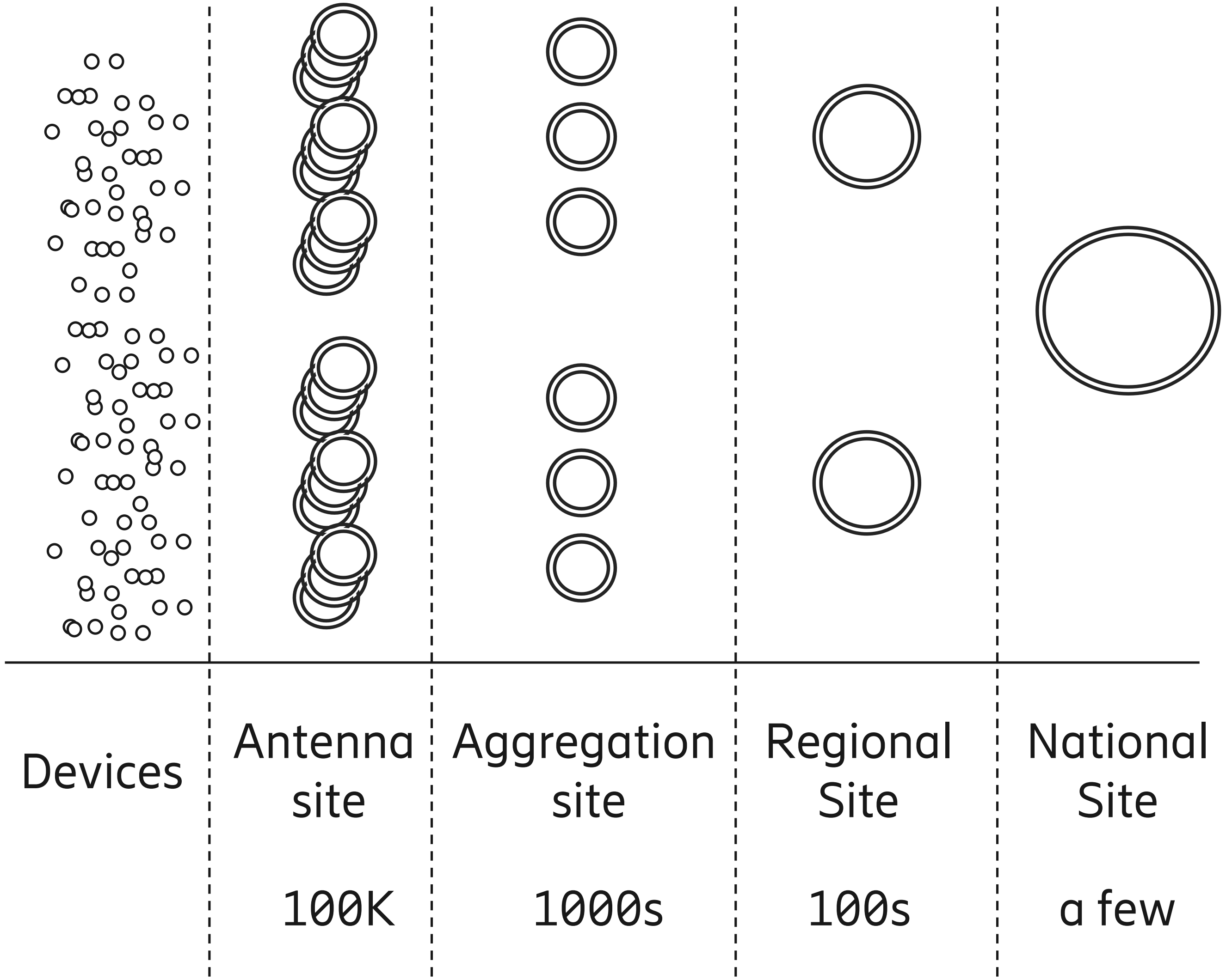
Approaches to Data Sharing in Edge FaaS

Zoltán Richárd Turányi
Expert, Ericsson Research
Hungary

Problem Statement: Mobile Cloud Apps

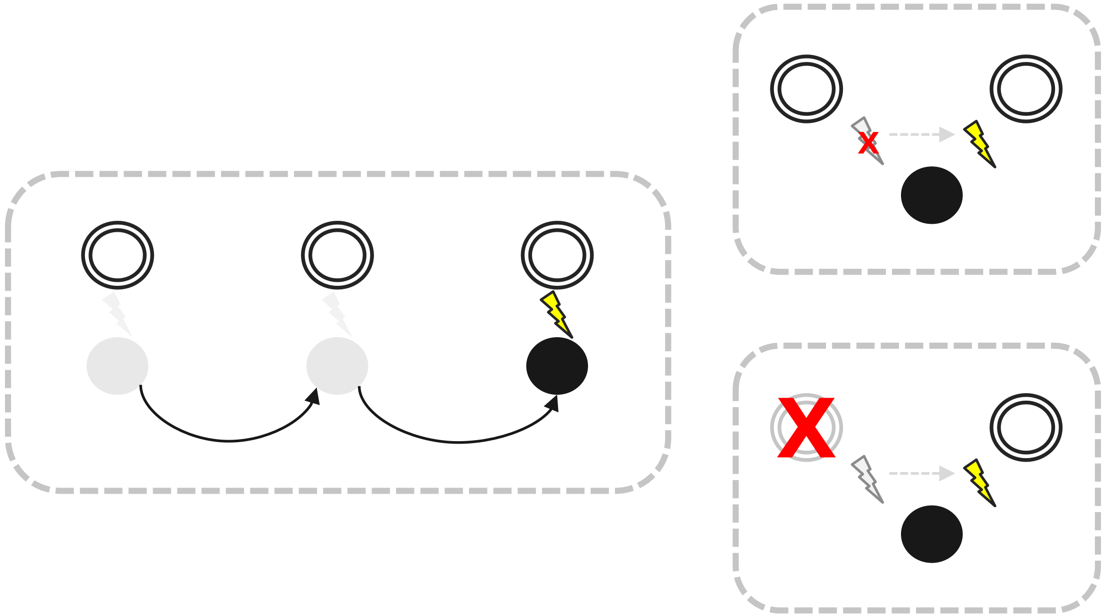
- Cellular networks are hierarchical
  - Centralized componets are cheap to build and maintain
  - But for radio reasons nodes must be distributed
- Other reasons to deploy application components distributed
  - Low latency towards end-user
  - Local processing to save bandwidth
  - Fate sharing with user
- 5G introduces new, low-latency modes
  - Ultra Reliability Low Latency Communication (URLLC)



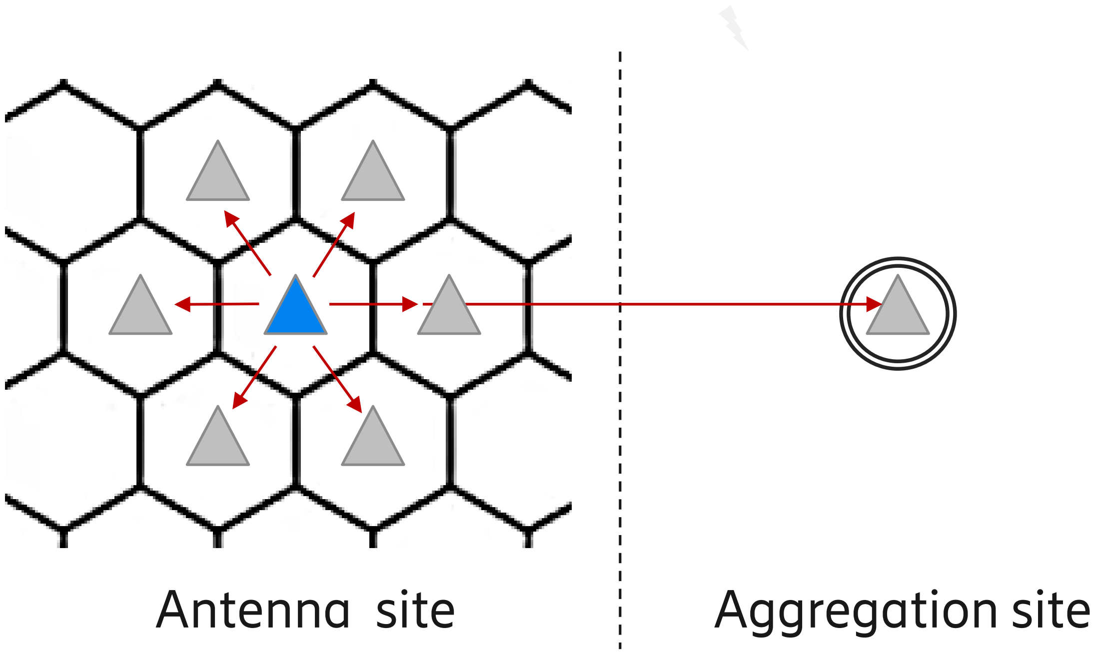
Low latency use cases

1. Cloud Virtual & Augmented Reality – Real-time Computer Rendering Gaming/Modeling
2. Connected Automotive – ToD, Platooning, Autonomous Driving
3. Smart Manufacturing – Cloud Based Wireless Robot Control
4. Connected Energy – Feeder Automation
5. Wireless eHealth – Remote Diagnosis With Force-Feedback
6. Wireless Home Entertainment – UHD 8K Video & Cloud Gaming
7. Connected Drones – Professional Inspection & Security

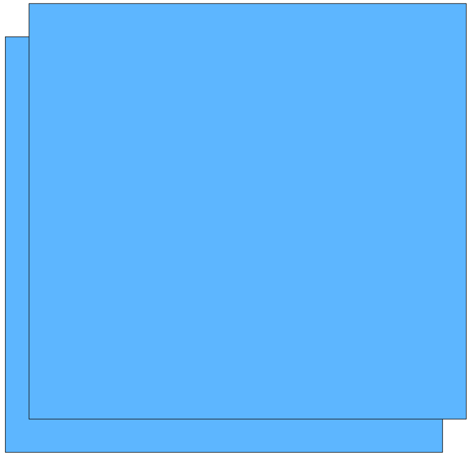
Mobility



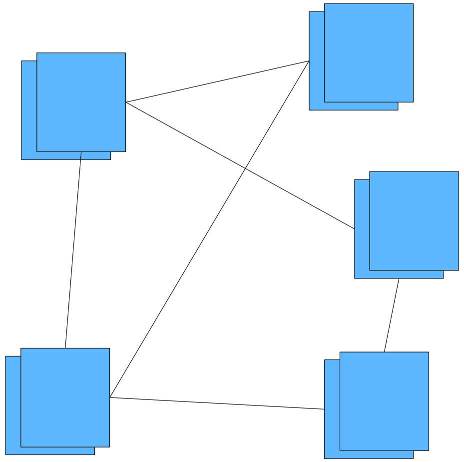
- Users move
  - Physical mobility
  - Change in radio conditions
  - Node and link failures
- States related to users need to be
  - moved
  - replicated
- Replicaton can be
  - To neighbouring edge sites (handy at mobility)
  - To central site



Problem Statement: Function-as-a-Service

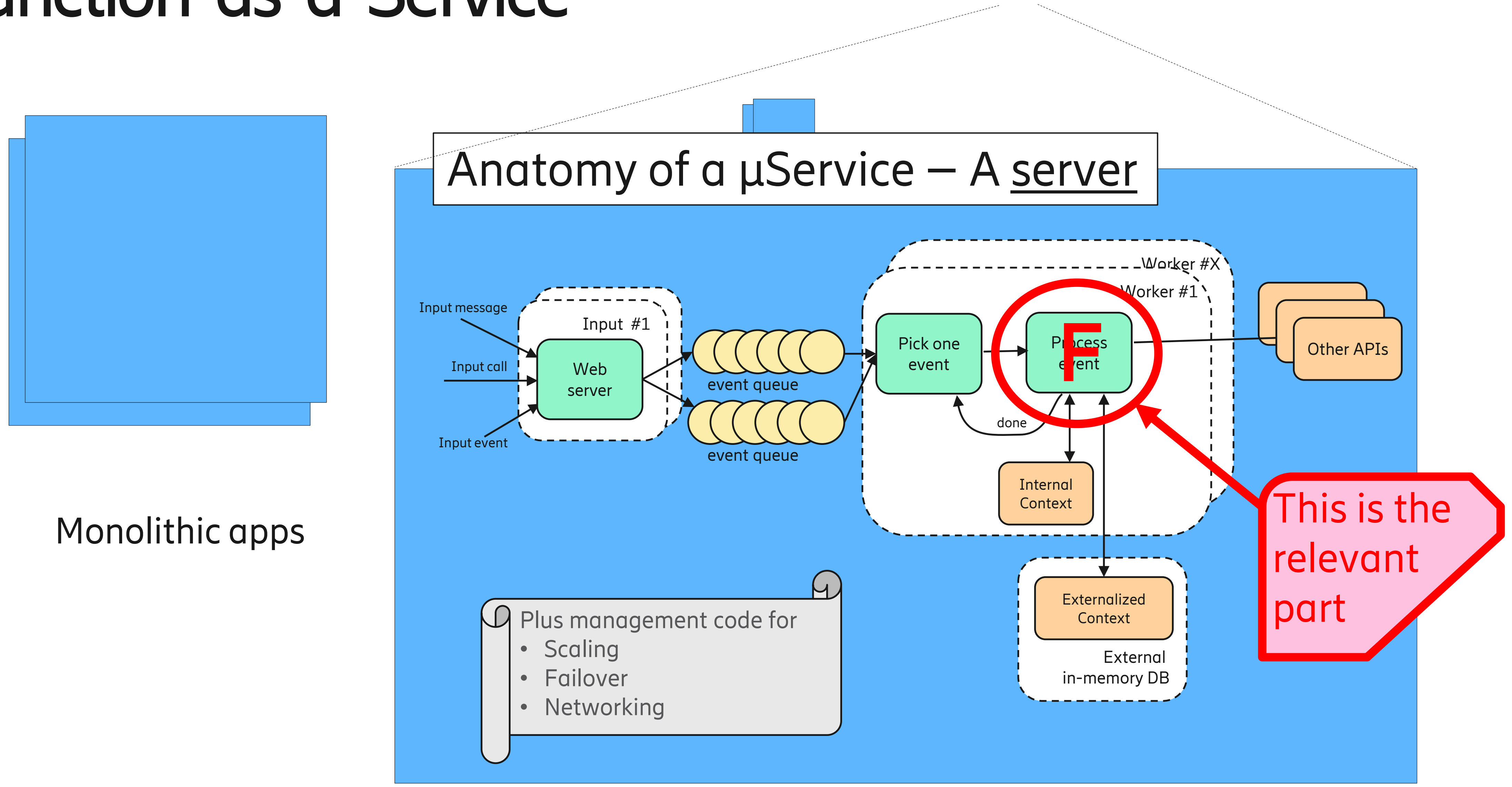


Monolithic apps

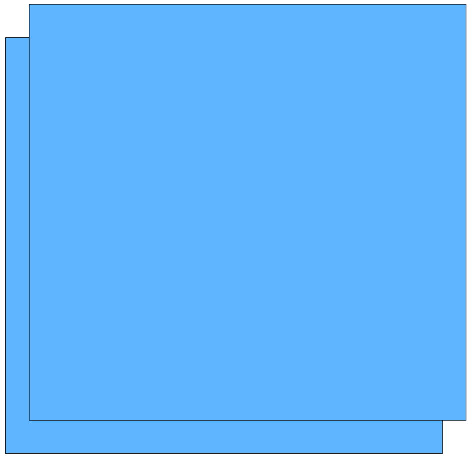


MicroServices

Problem Statement: Function-as-a-Service

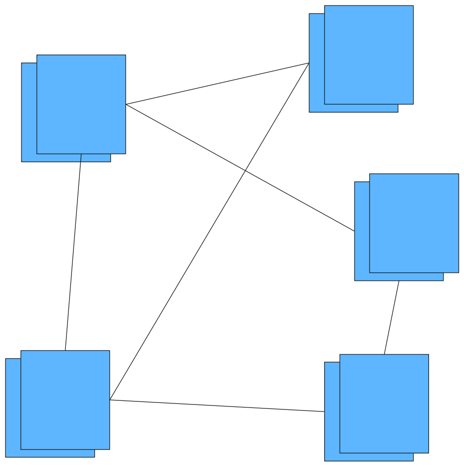


Problem Statement: Function-as-a-Service



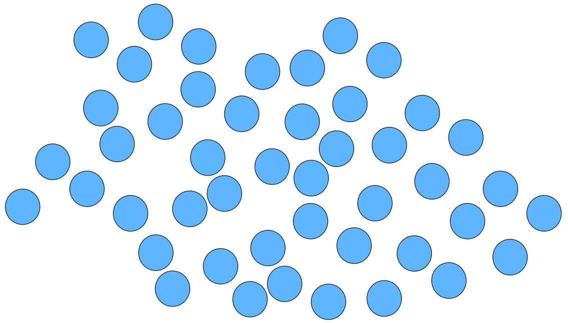
Monolithic apps

- All-in-one
- Scales in big blocks
- Upgrades monolithically



MicroServices

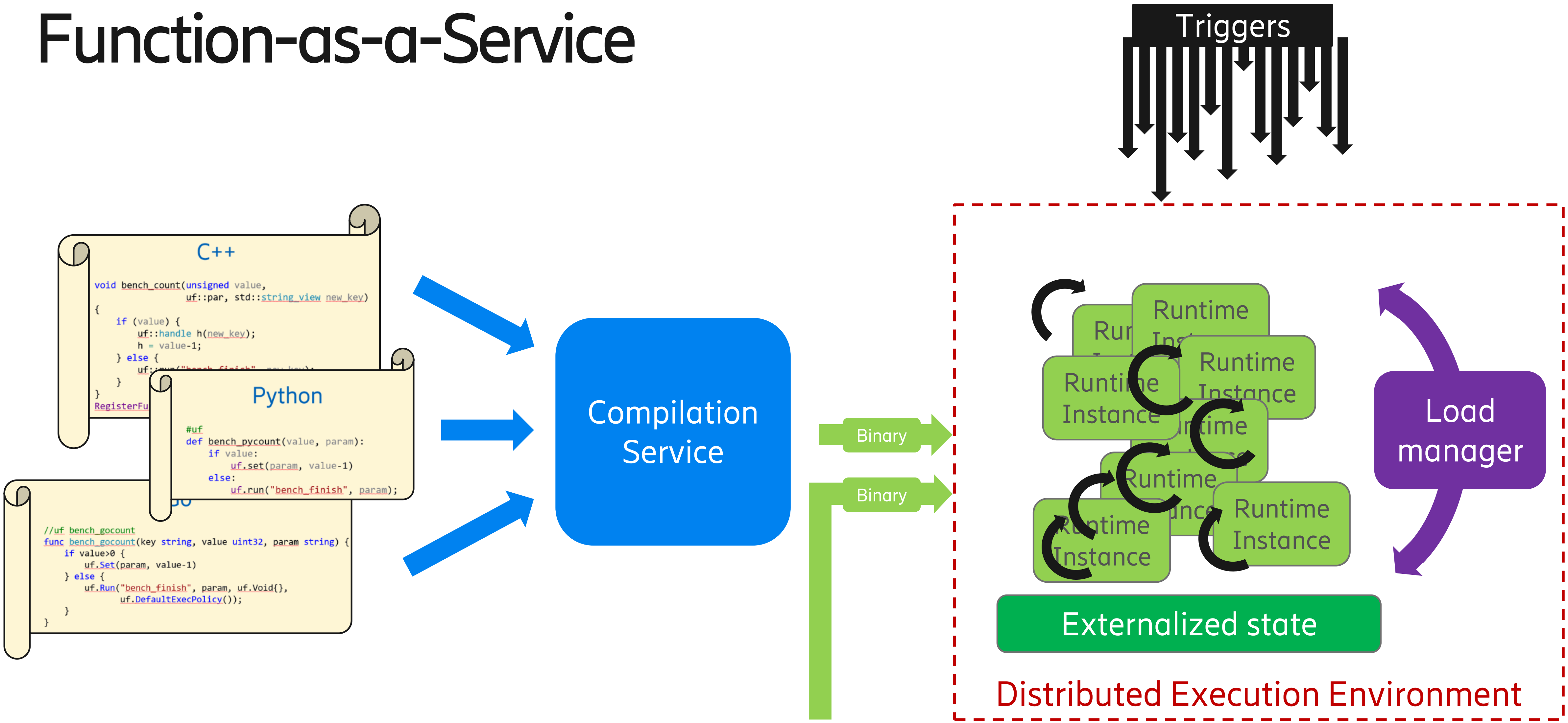
- Loosely coupled (hard)
- Data enclosed
- Overhead: Web servers, HTTP, sidecars
- Individual scaling, failover
- Developers do a lot besides business logic



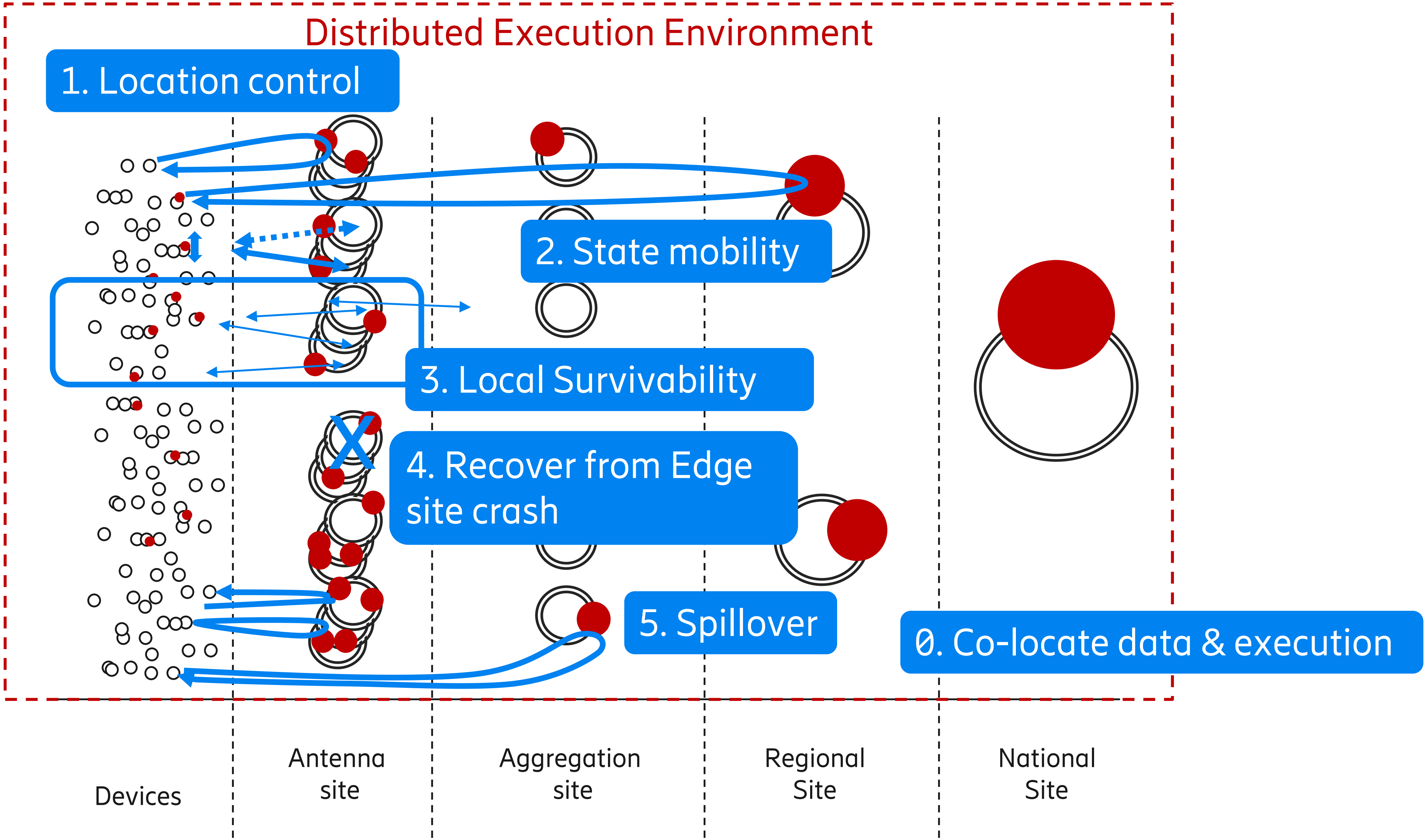
Functions

- Externalized state
- Developer focus
- Platform does scaling, failover
- Very fluid
- Full interworking with uServices

Problem Statement: Function-as-a-Service

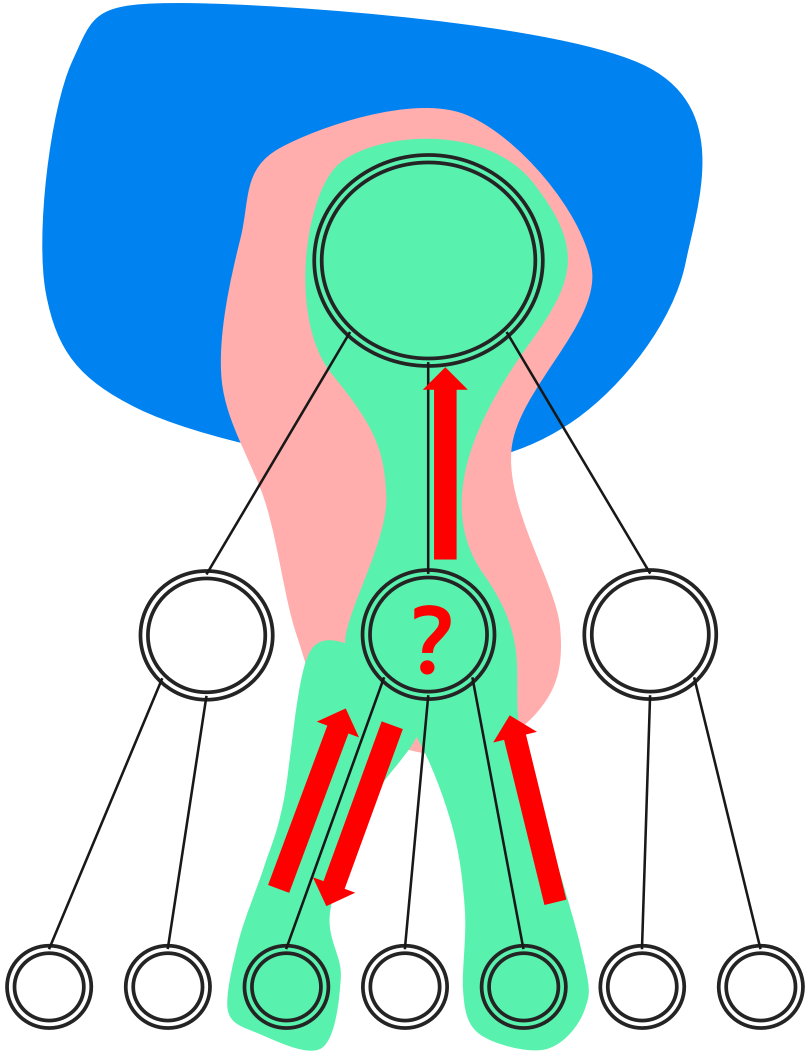


Problem Statement: Mobile FaaS Apps



CloudPath

- Hierarchical execution model: nodes may have children and parents
  - Children are usually less capable than parents
- Developers may mark functions to execute at specific hierarchy levels
- PathStore
  - Children cache a part of the parent's database locally
    - The root has everything
  - Reads fetch the relevant part (and subscribe updates)
    - Cold entries are automatically removed
  - Writes take effect locally then propagate upwards
    - Tightly synchronized GPS clocks are used to timestamp writes
    - Write conflicts are resolved using the timestamps
  - Eventually consistent

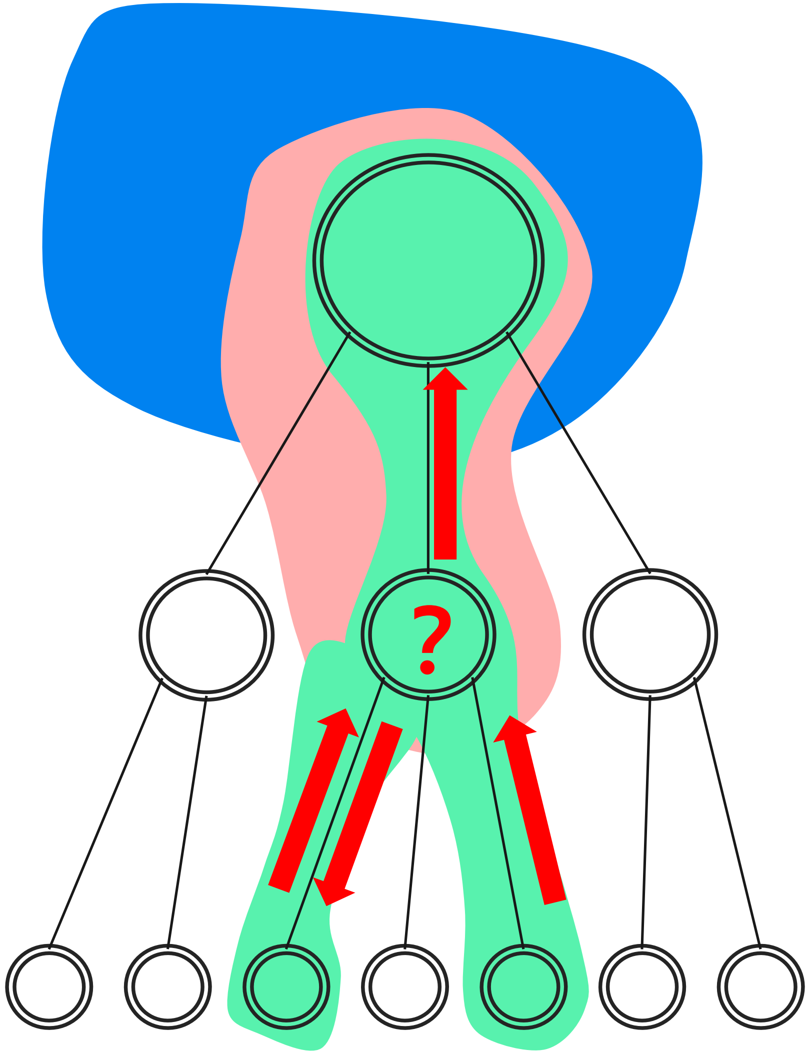


[CloudPath: A Multi-Tier Cloud Computing Framework]
Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, Eyal de Lara
2nd ACM/IEEE Symposium on Edge Computing (SEC), San Jose, CA, October 2017

CloudPath

- Good reliability
  - Data is stored at multiple levels
- Fast reads after caching
- Fast local writes
- Possible to add mobility
- May handle local survivability
  - If all needed data is locally cached

- Does not handle simultaneous writes very well
- No atomic updates possible (like a counter)



[CloudPath: A Multi-Tier Cloud Computing Framework]
Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, Eyal de Lara
2nd ACM/IEEE Symposium on Edge Computing (SEC), San Jose, CA, October 2017

What should the ideal database be like?

Note: possible to have more than one in an app.

Assumptions

- Most requests come from the edge
  - Goal is to serve these fast
- Execution
  - Functions are short lived and partake serving one request
  - Function Execution is possible everywhere
  - Already running functions do not move
- Database has the ability to move data around
  - Result in two phase lookups
  - Distributed hash tables are out
  - Caching of locations and subscribing to location updates help

Merging or serializing database

- Merging
  - Let local writes diverge the history
  - Merge changes in a distributed fashion

Super fast locally

Good merging strategy is needed.

Application dependent, custom merging logic.

- Serializing
  - Maintain a logical order of updates same everywhere
  - Results in a single location handling all updates for the same data

Super fast locally at master site.

Slow remotely.

Good with dominant accessor.

Versioning enables atomic read-update-write operations.

Replication

- Conflicting requirements
  - Handle Edge site failure
  - Provide Local Survivability

- Inter-site and intra site
- Robustness
  - Wait for replication to complete; or
  - Proceed logic in the meantime
- Location
  - From Edge to Central
  - Edge to Edge
    - Predict mobility or not
    - Controlled handover process

Fine control is needed by the programmer.

- Future-like mechanisms to have writes in parallel
- API to control replica locations & master mobility

Function Execution Location

- Programmer may designate both data and execution in the system by hand
  - Does not support e.g., spillover or edge site failure
- Two kind of automatic strategies
- Function mobility
  - Move the function's execution where its data is
  - Need to know what kind of data the function accesses
    - Provided by developer, Statically analysed, Measured
- Data mobility
  - Move the data to where functions execute
  - Best if there is some consistent execution of functions (including sharding)
  - Data may migrate to servers not functions

As simple as falling back to centralized execution if data not available locally

Optimization:
Co-locate functions working on same data

Multi-key or single-key transactions

- Single-key transactions
  - Each transaction affects only one addressable data element
    - E.g., plain Key-Value stores

- Multi-key transactions
  - Easy to program
  - Difficult and complex to implement
  - Very slow for data scattered all around

Workarounds
- Large, composite values
- Multi-step updates via ‘lock’ keys
  1. Write into a key to take a lock
  2. Update several keys
  3. Release the lock

We can send code around
- Decompose transaction code
- Execute close to data in parallel
- Have the ability to roll back if needed

Summary

- 5G and Edge computing will enable many exciting low-latency use cases
- FaaS is emergent programming paradigm for the Cloud
- Selecting the right external database for Edge FaaS is a challenge