



15th ANNUAL WORKSHOP 2019

A DATA STORE FOR RESOURCE EFFICIENT SERVERLESS COMPUTING

Adrian Schuepbach, Bernard Metzler, Patrick Stuedi, Jonas Pfefferle

IBM Zurich Research

[March, 2019]

[LOGO HERE]


Agenda

- Why Serverless Computing
- Current limitations and challenges
- Handling ephemeral data
- How to integrate an ephemeral data store?
- A prototype integration
- Summary

A Decade ago: From Managed Clusters to Cloud Computing

The promise of Cloud Computing

- ✓ Virtually infinite computing resources on demand.
- ✓ No up-front commitment by cloud users.
- ✓ Pay for use of computing resources as needed.
- ✓ Economies of scale (large data centers).
- ✓ Simplified operation via resource virtualization.
- ✓ Higher hardware utilization by multiplexing workloads, even from different organizations.



“The data center is now the computer”

...Cloud simplified operation?

- From managing physical resources to managing virtual resources
- Scaling for changing load requirements
- Resource idling/overprovisioning for stateful services like data bases
- Monitoring resource health
- System upgrades,
-

Serverless Computing

User

- Writes Cloud Function
- Defines event to trigger running the function

Serverless system

- Server instantiation
- Resource scaling/elasticity ... auto-configuration
- Fault tolerance
- Logging
- Maintenance

- ✓ User: No cluster setup/management
- ✓ User: Fine grained, sub-seconds billing
- ✓ Provider: optimize resource utilization



AWS Lambda, IBM Cloud Functions, Google Functions, Azure Functions, databricks serverless
...

Fundamental Differences to conventional Cloud Computing

The abstraction of executing a piece of code instead of first allocating resources on which to execute that code then.

.

.

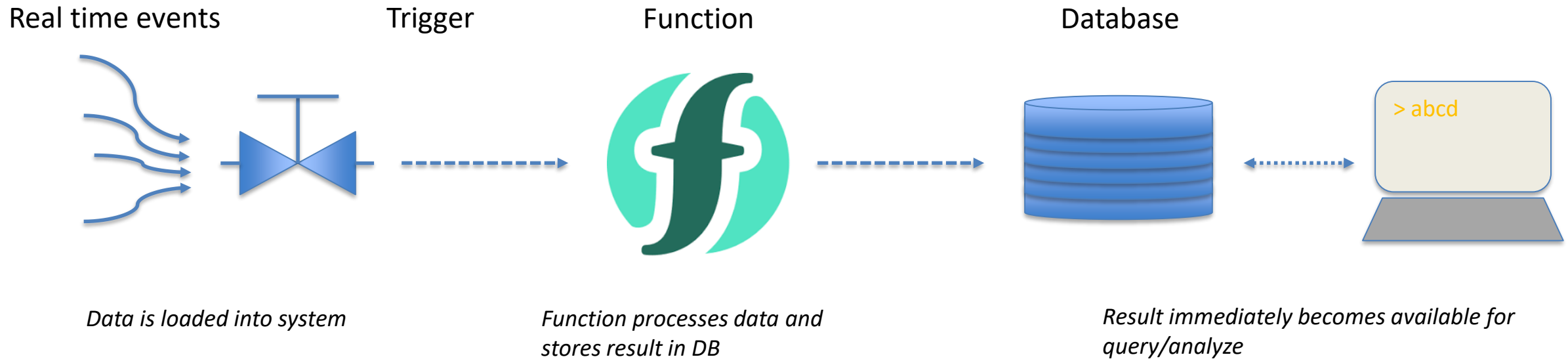
Paying for the code execution instead of paying for resources allocated to executing the code.

.

.

Decoupling of computation and storage; they scale separately and are priced independently.

A simple Serverless Function



Web applications/backends (IoT, chatbots, ...)
Real time data processing (image transcoding, ...)

Can we use Serverless Computing for Analytics and ML?

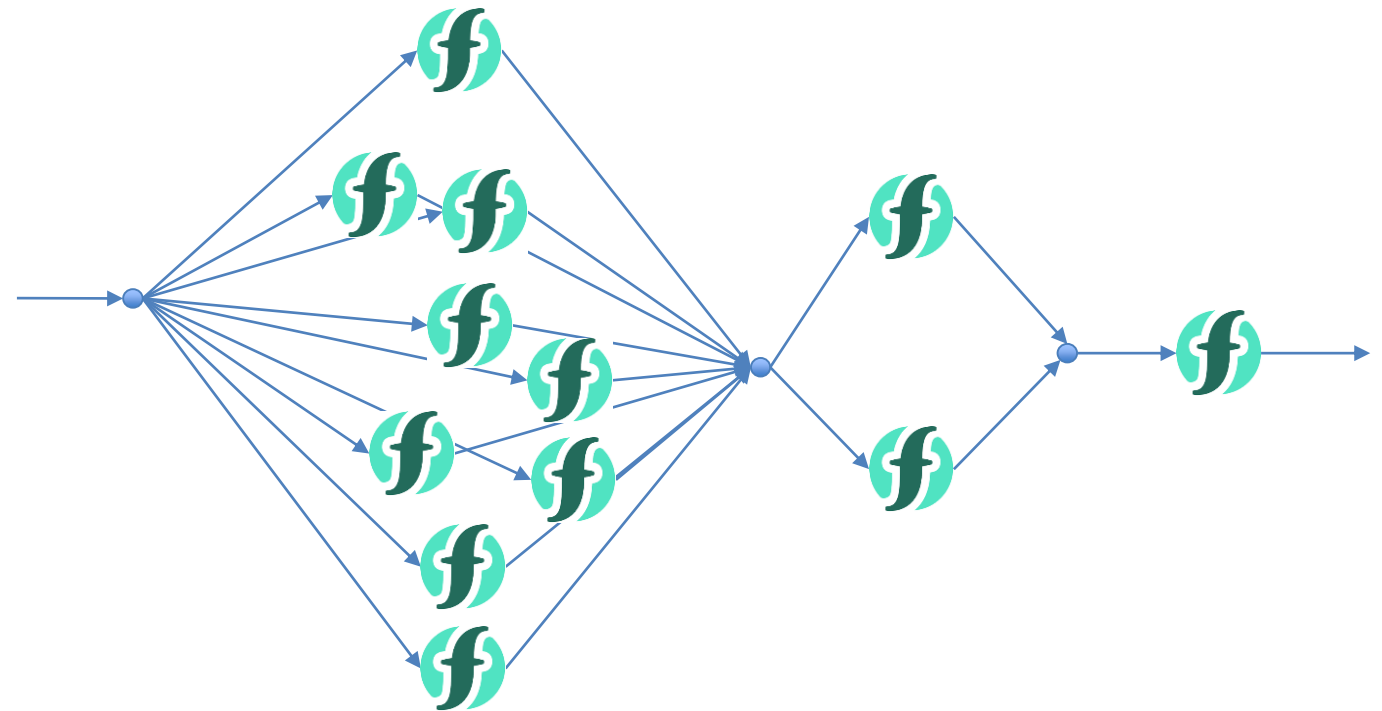
While preserving Serverless Model:

- ✓ Exploit massive parallelism
- ✓ Compose multiple stages reflecting changing resource requirements

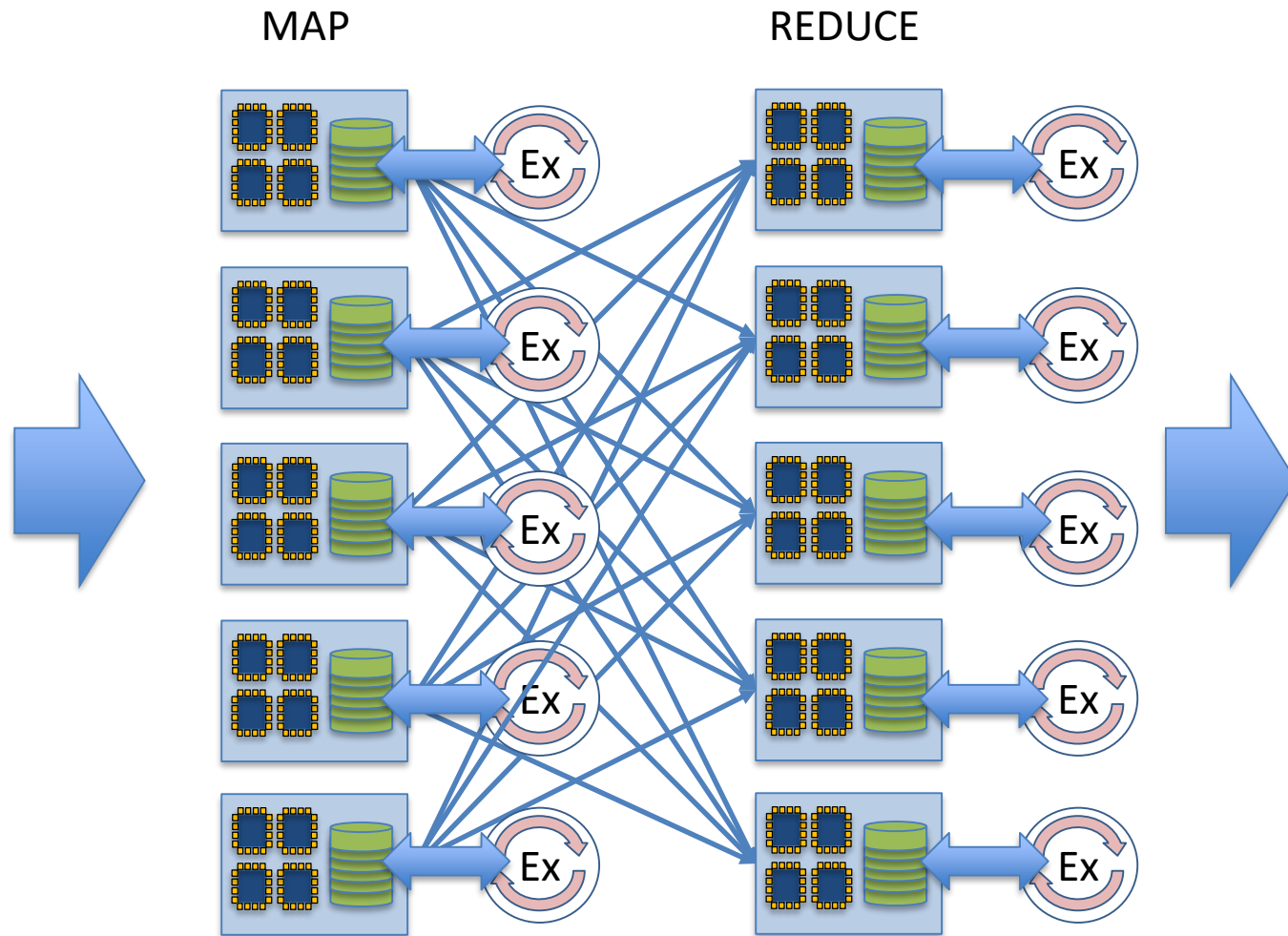
Challenges:

- Efficient function instantiation
- Efficient and fine grained coordination between functions and stages (pub/sub, DQ)
- Efficient communication of **ephemeral data** between stages

Why is the handling of those **Ephemeral Data** so difficult?

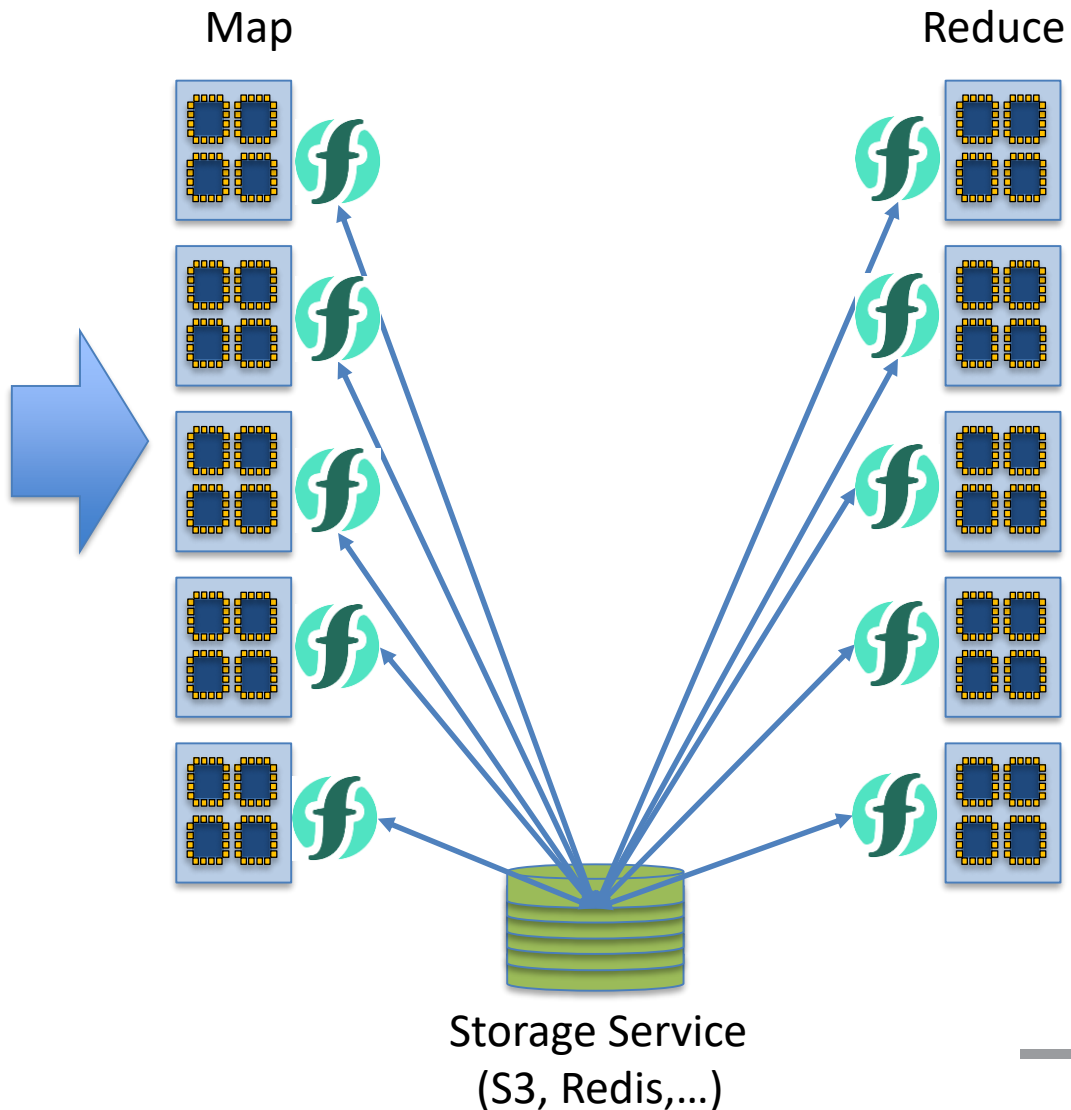


A MapReduce Job on a managed Cluster



- Ephemeral data mostly written and read **locally**
- Ephemeral data **directly** exchanged between tasks

MapReduce in a Serverless World



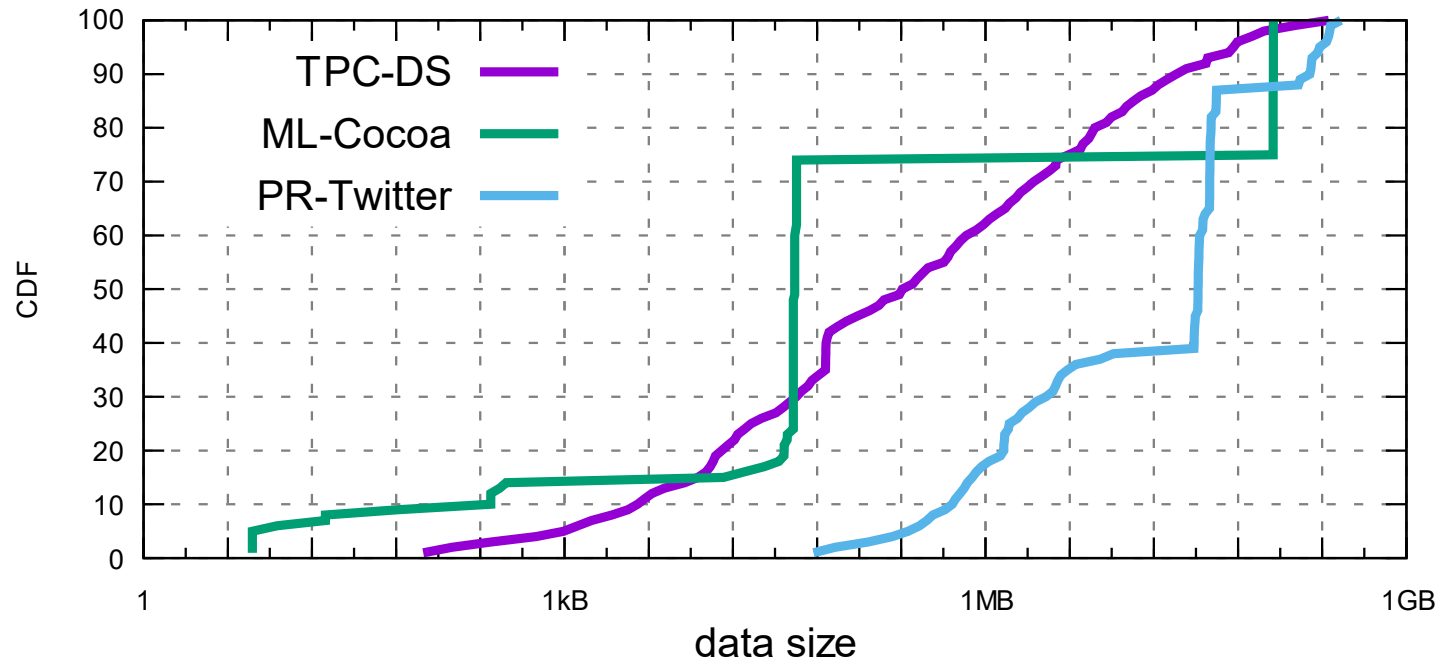
- No direct communication between tasks
- Tasks are short living
 - Tasks are stateless
 - Read and write from distributed storage service
 - Cannot directly pass state and data from function to function

Efficiency of job execution highly dependent on storage service's

- **Performance**, and
- **Elasticity**

Requirements for Serverless Ephemeral Storage: Performance

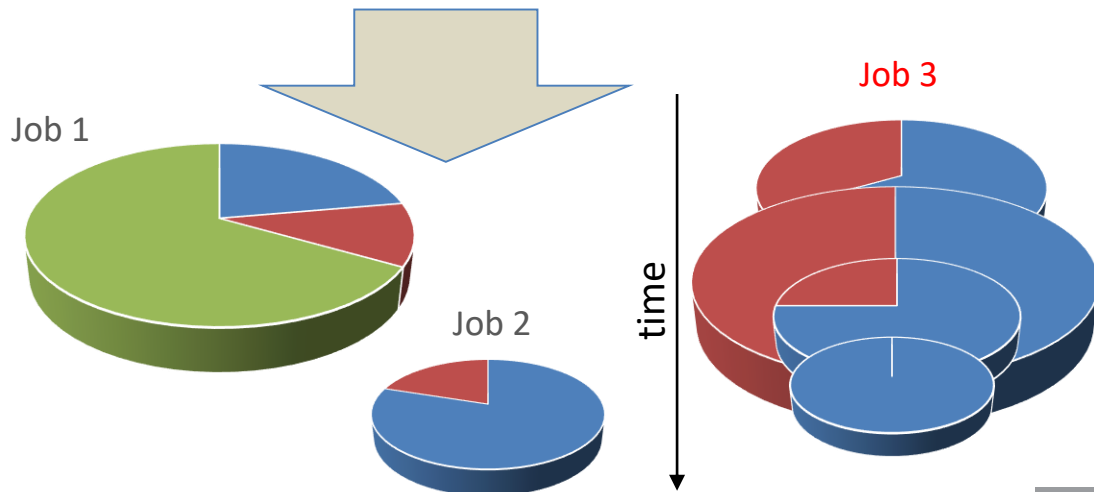
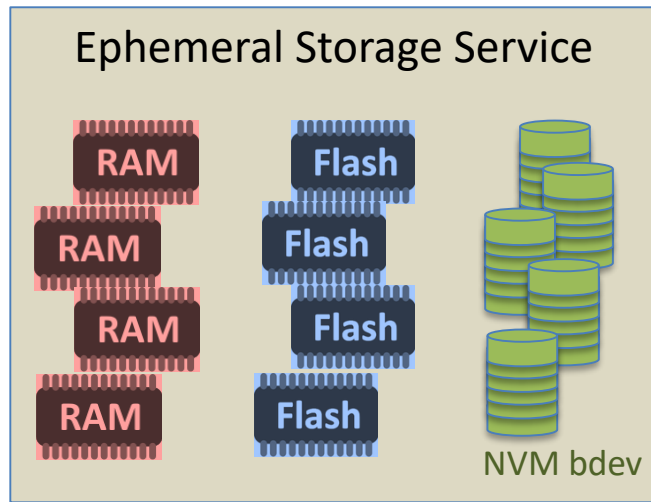
CDF distribution of different types of workloads



High performance for a wide range of object sizes

- Today's KV stores have value size limits
- Object stores have access delay limits

Requirements for Serverless Ephemeral Storage: Elasticity



Fine grained billing reflecting used resources:

- Dynamically decide on storage service requirements
 1. Amount of storage required
 2. Storage performance required (including storage media type)
 3. Data persistency/lifetime (inter-function vs. intra-function)

- Efficiently support jobs with varying storage requirements *during* job execution

- ✓ User benefit:
 - Allows for fine grained billing
 - Allows for cost optimization
- ✓ Provider benefit
 - Allows for optimal resource usage

Which Data Store for Ephemeral Data?

- **Object Store, such as S3**

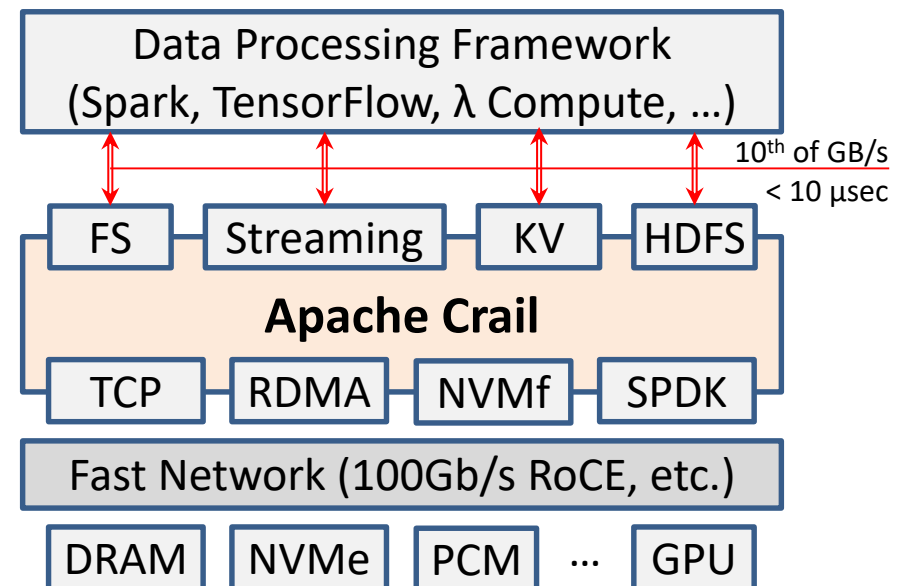
- + Good for large data sets
- Inefficient for small data sets

- **KV Store, such as Redis, Memcached, AWS ElastiCache**

- + Good for small data sets
- Inconvenient to store large data sets (indirection)
- No dynamic scaling
- DRAM only (costly)

- **Apache Crail (incubating) data store**

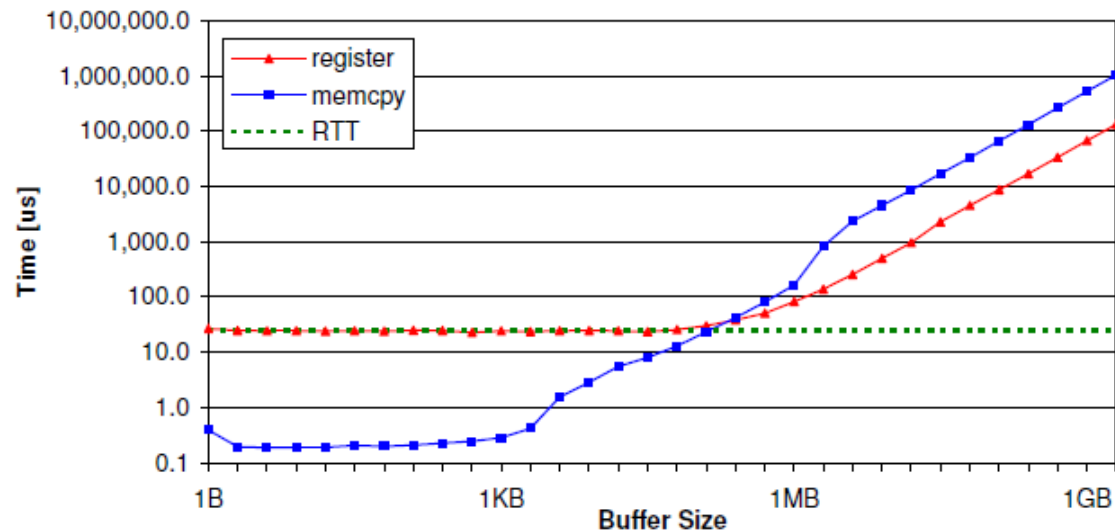
- + Lowest delay for small data sets
- + Highest throughput for large data sets
- + Multi-tier (DRAM RDMA/TCP, NVMeF, block store) in unified namespace
- + Explicit data location control possible
- + Can use RDMA
- No dynamic scaling (yet)



Is RDMA a good fit for Serverless Computing?

PRO: Known RDMA Usage Benefits

- **Low delay access**
- **Arbitrary sized objects**
- **Low CPU load**



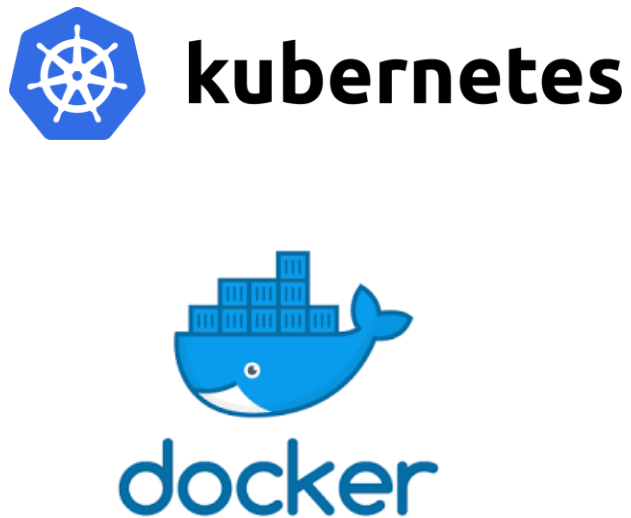
CONTRA: The RDMA Setup Costs

- **Function resource setup costs**
 - RDMA objects (QP, PD, CQ, ...)
 - Memory Registration
 - Local memory pinning and translation
- **Remote service setup costs**
 - RDMA objects (some RDMA objects may be reusable)
- **Both sides (function + storage service)**
 - Connection setup
 - Think about connectionless RDMA
 - Credential exchange with peer (for RDMA Read/Write)

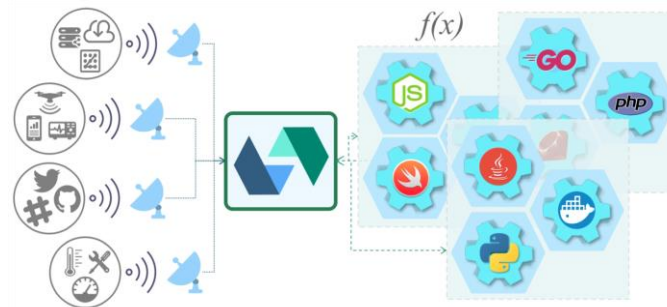
The Development Environment

- Runtime environment: **Kubernetes, Docker**
- Cloud Functions: **Apache OpenWhisk**
- Ephemeral data store: **Apache Crail**

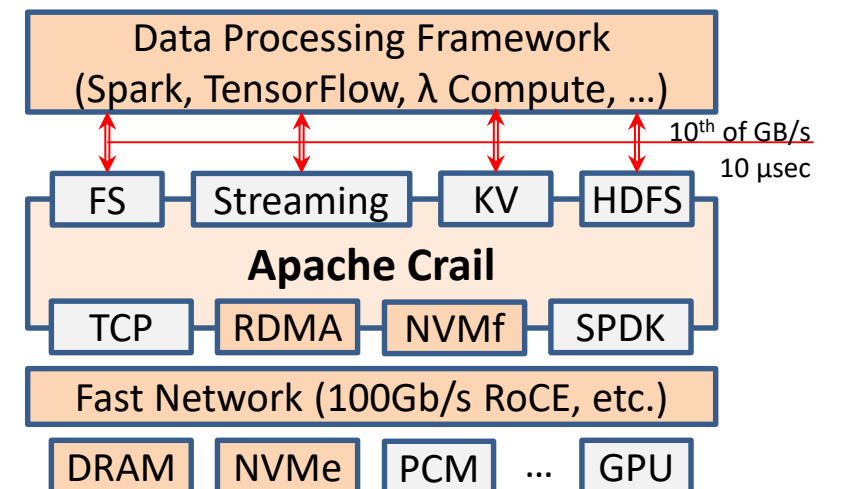
- Physics:
 - 4 nodes x86 cluster
 - 100Gbs RoCE
 - Samsung 960 pro NVMe m.2



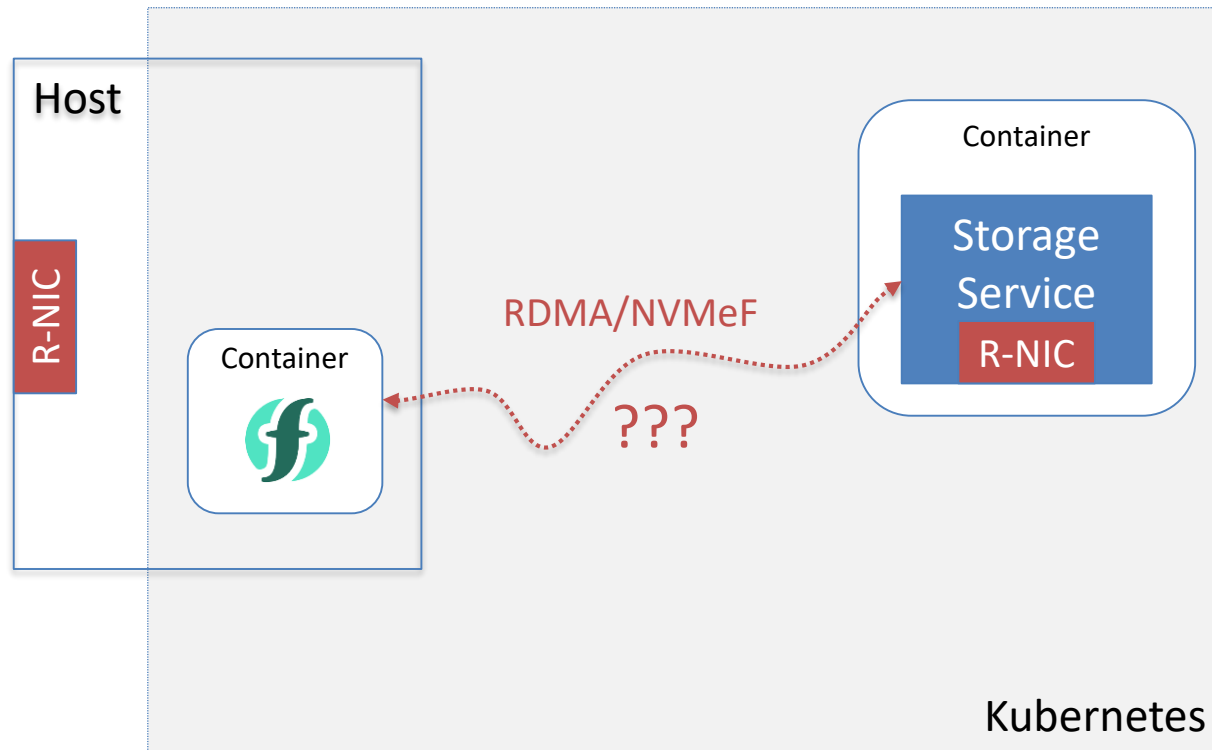
<https://openwhisk.apache.org/>



<http://crail.apache.org/>

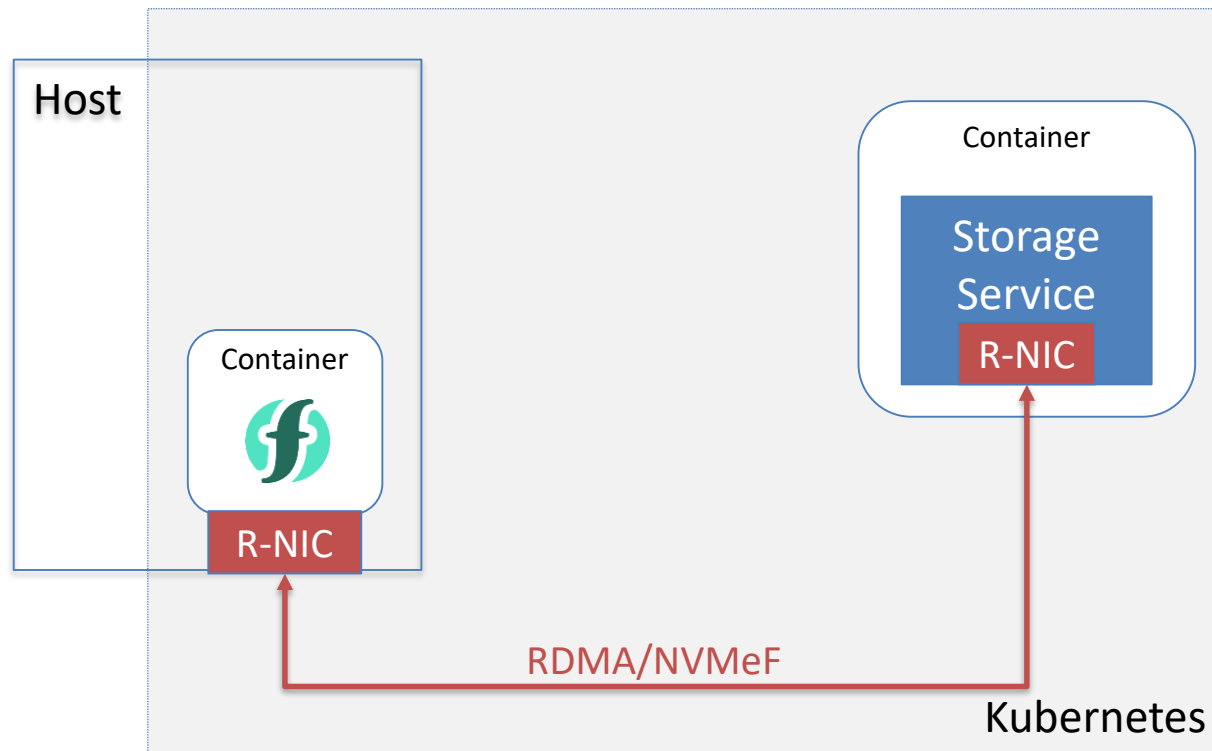


How to attach Serverless Function to an RDMA enabled Ephemeral Data Store?



- How to balance performance and elasticity?
- How to make integration as seamless as possible?

Function directly talks RDMA to Data Store



PRO

- Native RDMA integration
- Fast communication

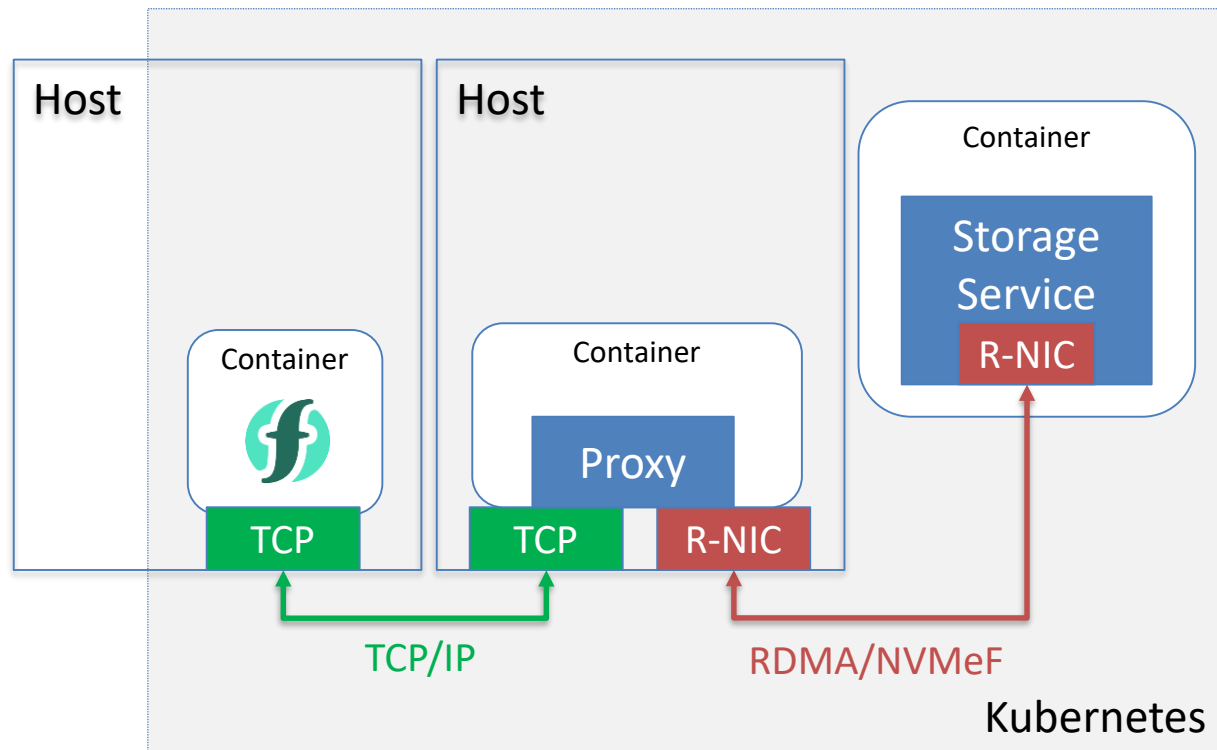
CONTRA

- Unacceptable resource setup cost on each function invocation
- Pass-through of RDMA device (Kubernetes, Docker)

POTENTIAL SOLUTION

- Warm container caches:
 - Functions
 - Connections
 - RDMA resources (memory registration etc.)
- Connectionless RDMA service

Function talks via remote Proxy to Data Store



PRO

- Serverless framework does not need to integrate with RDMA
- Elastic: no persistent state in function container
- Proxy can be persistent, keeping RDMA resources and storage state

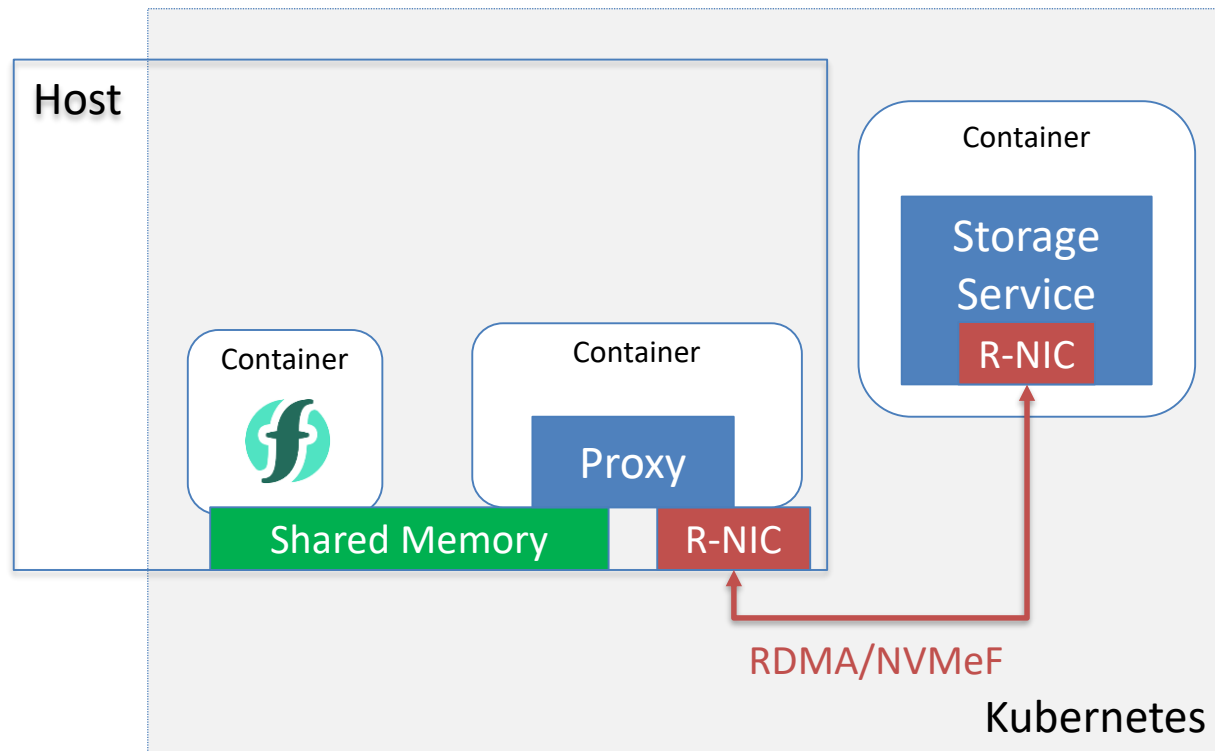
CONTRA

- Performance penalty
 - Data copy
 - Data two times on the wire

POTENTIAL SOLUTION

- Try co-locating function and proxy containers on same physical host

Function talks via host local Proxy to Data Store



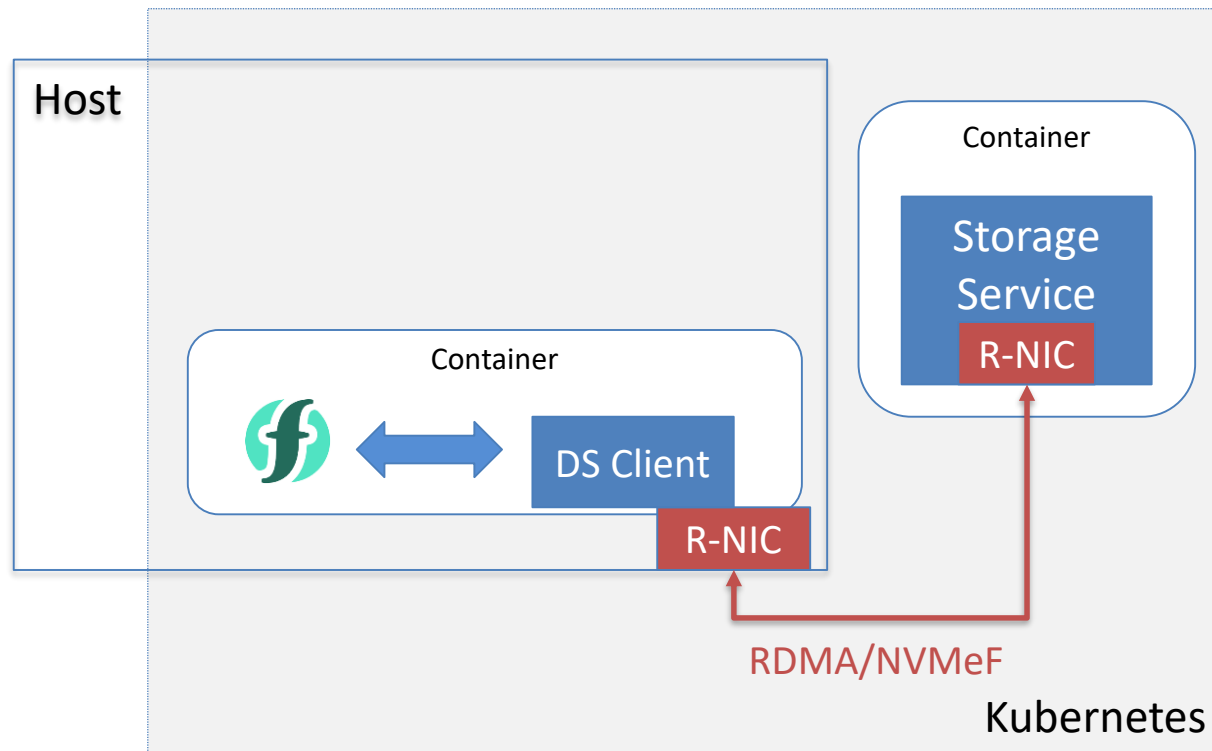
PRO

- Serverless framework does not need to integrate with RDMA
- Elastic: no persistent state in function container
- Proxy can be persistent, keeping RDMA resources and storage state
- Shared memory: zero copy possible
 - Application handles data in shared memory
 - Shared memory is RDMA registered

CONTRA

- One storage proxy per physical host

RDMA Storage Client as Service of Function's Container



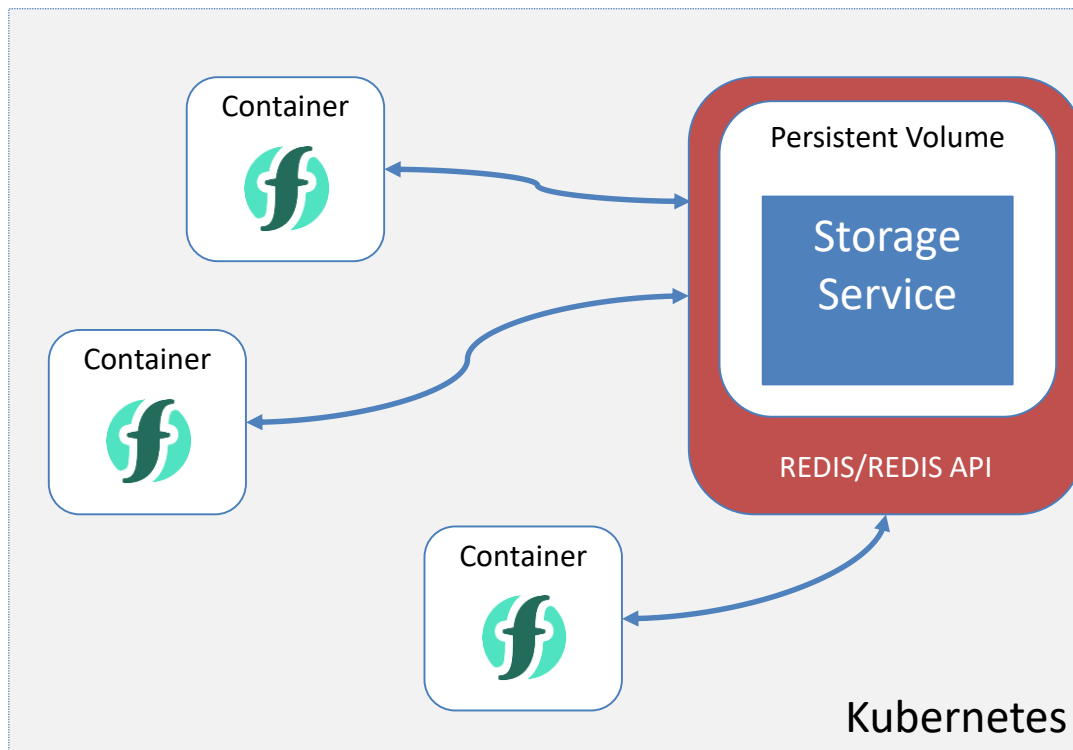
PRO

- Serverless framework does not need to integrate with RDMA
- Direct communication between function and storage client
- RDMA resources and storage state can be kept persistent for 'hot' container

CONTRA

- Resource setup efficiency limited to lifetime of 'hot' container
- Pass-through of RDMA device (Kubernetes, Docker)

RDMA Storage as attachable Persistent Volume



PRO

- Clean integration like any other Volume (Redis, NFS, S3, ...)
- RDMA resource caching at own discretion

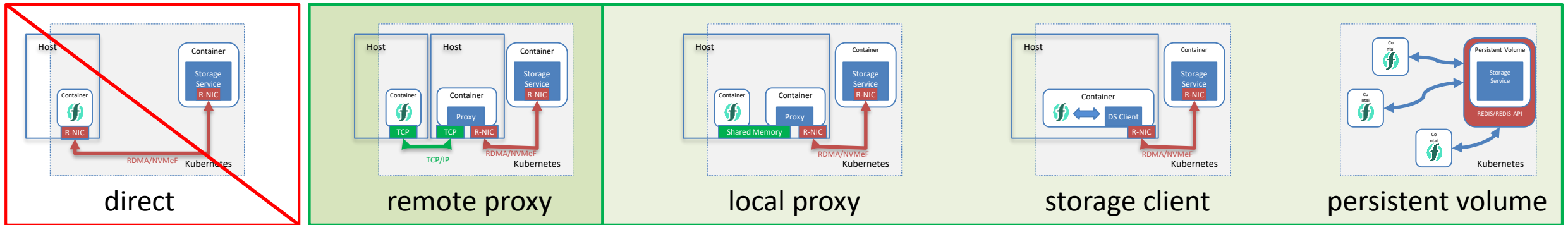
CONTRA

- Potentially more layers of software hiding RDMA benefits (FUSE, ...)
- Serverless framework today unaware of persistent volumes
 - Containers are instantiated w/o persistent volumes
 - No Function's API to access mounted volume

POTENTIAL SOLUTION

- Extend serverless framework to work with persistent volumes
 - Mounting
 - Storage API

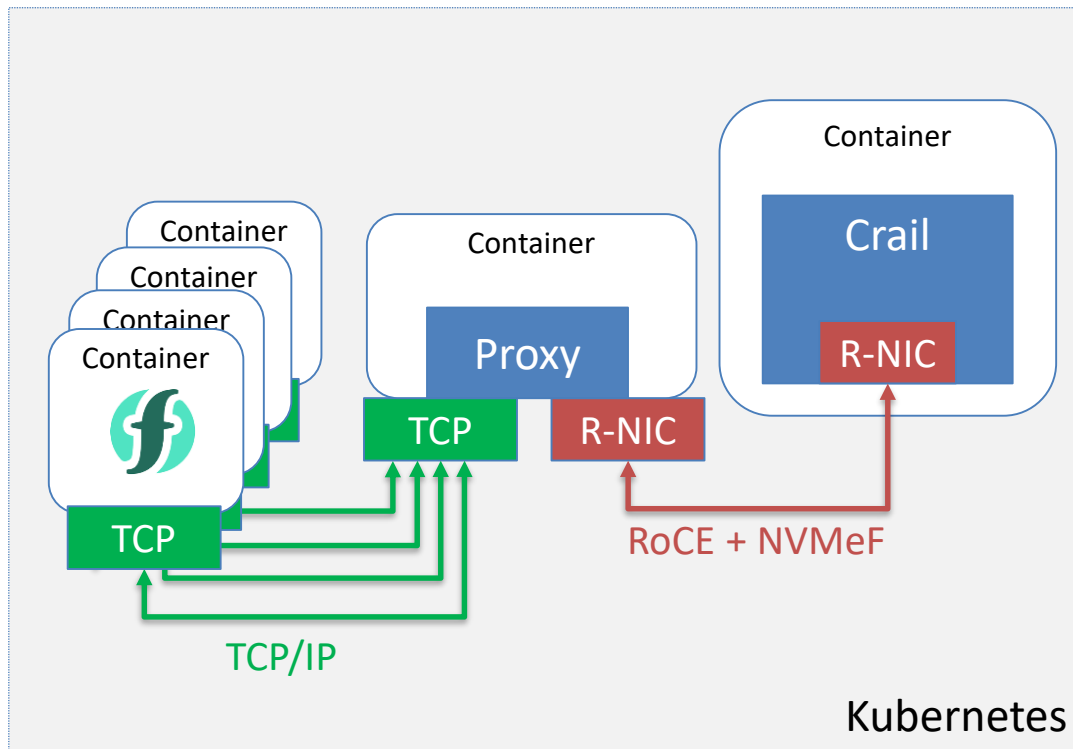
Our (tentative) Choice



- **Direct Cloud Function attached RDMA interface seems unreasonable**
- **Investigating storage proxy, storage client, and persistent volume variants**
- **Persistent Volume too ambitious...for now...work in progress...**

- **Started evaluation with remote proxy**

Detailed System Setup



- **Containerized Apache Crail instance**
 - RDMA/DRAM tier
 - NVMeF tier
- **Physically distributed data store**
 - Data nodes on all machines (DRAM + NVMeF)
- **Single containerized storage proxy**
 - Runs Crail storage client on some physical node
 - Maintains RDMA connections to Crail name node and data nodes
 - Accepts connections from remote container's functions via TCP/IP
 - Converts storage requests
 - Ready to move closer to function's container

Our first test Application: MapReduce Bucket Sorting

MapReduce Sorting:

1. Read data from persistent store into map functions
2. Write intermediate data from map functions to Ephemeral Data store
3. Read intermediate data from Ephemeral Data store into reduce functions
4. Merge, reduce and write back to long term storage

Bucket sort algorithm

- Map and Reduce functions written for Apache OpenWhisk

Sorting 100 GB

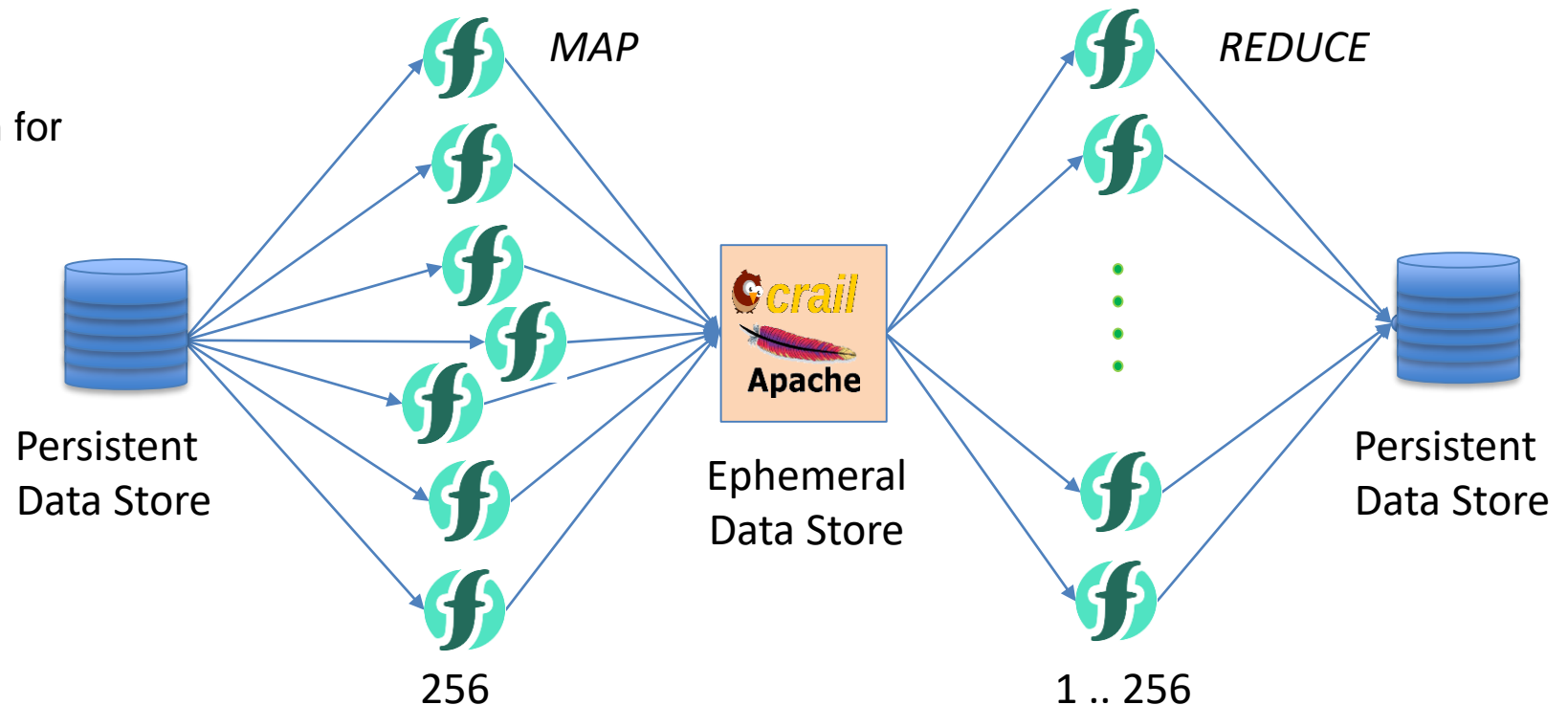
Two storage tiers

- NVMeF
- DRAM

4 Systems

256 map functions

1 ... 256 reduce functions



First Prototype Measurements



```
"logs": [
  "2019-03-19T13:12:00.434215247Z stdout: Unhandled exception",
  "2019-03-19T13:12:00.434282669Z stdout: Type=Bus error vmState=0xffffffff",
  "2019-03-19T13:12:00.434288514Z stdout: J9Generic_Signal_Number=00000008 Signal_Value=00000000 Error_Value=00000000 Signal_Code=00000002",
  "2019-03-19T13:12:00.434293426Z stdout: Handler1=00007F743E924B0 Handler2=00007F743E797C40 InaccessibleAddress=00000000FFE00008",
  "2019-03-19T13:12:00.434297256Z stdout: RDI=00007F743815AF60 RSI=00007F743FDE5060 RAX=00007F743C817130 RBX=0000000000000001",
  "2019-03-19T13:12:00.434301063Z stdout: RCX=00000000FFE00000 RDX=0000000001000000 R8=00000000FFF00000 R9=0000000000000000",
  "2019-03-19T13:12:00.434304917Z stdout: R10=00007F743C817CD0 R11=00007F743C817CD0 R12=0000000001000000 R13=00007F743815AF60",
  "2019-03-19T13:12:00.434308773Z stdout: R14=00000000FFF00000 R15=00000000FFE00000",
  "2019-03-19T13:12:00.434312422Z stdout: RIP=00007F743C554599 GS=0000 FS=0000 RSP=00007F743FDE4DE0",
  "2019-03-19T13:12:00.434316048Z stdout: EFlags=00000000 IOPID=10202 CS=0033 RBP=00000000FFE00000 ERR=0000000000000006",
  "2019-03-19T13:12:00.434319761Z stdout: TRAPNO=00000000 OLDMASK=0000000000000000 CR2=00000000FFE00008",
  "2019-03-19T13:12:00.434324186Z stdout: xmm0 72 f6d654 f4c3a (f: 1298091392.000000, d: 1.676463e+243)",
  "2019-03-19T13:12:00.434331529Z stdout: xmm2 ff ff00 000000 (f: 0.000000, d: -nan)",
  "2019-03-19T13:12:00.434338749Z stdout: xmm4 0000000000000000 (f: 0.000000, d: 0.000000e+00)",
  "2019-03-19T13:12:00.434342253Z stdout: xmm5 0000000000000000 (f: 0.000000, d: 0.000000e+00)",
  "2019-03-19T13:12:00.434349357Z stdout: xmm7 4059800000000000 (f: 0.000000, d: 1.000000e+02)",
  "2019-03-19T13:12:00.434353064Z stdout: xmm8 4059800000000000 (f: 0.000000, d: 1.000000e+02)",
  "2019-03-19T13:12:00.434356624Z stdout: xmm9 41a0000002000000 (f: 33554432000000.000000, d: 1.347777e+08)",
  "2019-03-19T13:12:00.434360095Z stdout: xmm10 be63bb060ea0baeb (f: 245414040.000000, d: -675.43e-08)",
  "2019-03-19T13:12:00.434363743Z stdout: xmm11 3c4cd58858eb8000 (f: 1491828736.000000, d: 3.17620e-18)",
  "2019-03-19T13:12:00.434367305Z stdout: xmm12 0000000000000000 (f: 0.000000, d: 0.000000e+00)",
  "2019-03-19T13:12:00.434370933Z stdout: xmm13 403298c812e00000 (f: 316669952.000000, d: 1.859680e+01)",
  "2019-03-19T13:12:00.434376032Z stdout: xmm14 0000000000000000 (f: 0.000000, d: 0.000000e+00)",
  "2019-03-19T13:12:00.434379661Z stdout: xmm15 bc5b4805a84f36de (f: 2823763712.000000, d: -5.915697e-18)",
  "2019-03-19T13:12:00.434383262Z stdout: Module=/opt/java/openjdk/jre/lib/amd64/compressedrefs/libj9gc29.so",
  "2019-03-19T13:12:00.434387021Z stdout: Module_base_address=00007F743C393000",
  "2019-03-19T13:12:00.434390473Z stdout: Target=2_90_20180813_291 (Linux 4.15.0-33-generic)",
  "2019-03-19T13:12:00.434393984Z stdout: CPU=amd64 (32 logical CPUs) (0x178681b000 RAM)",
  "2019-03-19T13:12:00.434912438Z stdout: JVM_DUMP039I Processing dump event \"gpf\", detail \"\" at 2019/03/19 13:12:00 - please wait.",
  "2019-03-19T13:12:00.435907288Z stdout: JVM_DUMP032I JVM requested System dump using '/core.20190319.131200.1.0001.dmp' in response to an event",
  "2019-03-19T13:12:00.545887744Z stdout: JVM_PORT030W /proc/sys/kernel/core_pattern setting \"|/usr/share/apport/apport %p %s %c %d %P\" specifies that the core dump is to be piped to an external program. Attempting to rename either core or core.8."
],
  1,
```

Check 'Pocket' project with architectural similarities for very promising results w/o using RDMA

<https://www.usenix.org/conference/osdi18/presentation/klimovic>

Conclusions

■ **Serverless Computing**

- Promising technology, the next natural step after Cloud computing
 - Ease of programming
 - Automated resource management
 - Pay-as-you-go
- Many unsolved issues (security, ephemeral and durable storage, networking, management, ...)
- Solving those issues will make Serverless Computing attractive for almost all applications

■ **Our current effort:**

- Focus on the Ephemeral Storage aspect
- Integrate Apache Crail backed Ephemeral Data Store
 - Promising high performance, cost efficiency
 - May use RDMA if available

■ **Other related activities:**

- Flexible provisioning/autoscaling with Apache Crail
- Add distributed communication primitives to the store



15th ANNUAL WORKSHOP 2019

THANK YOU

[LOGO HERE]