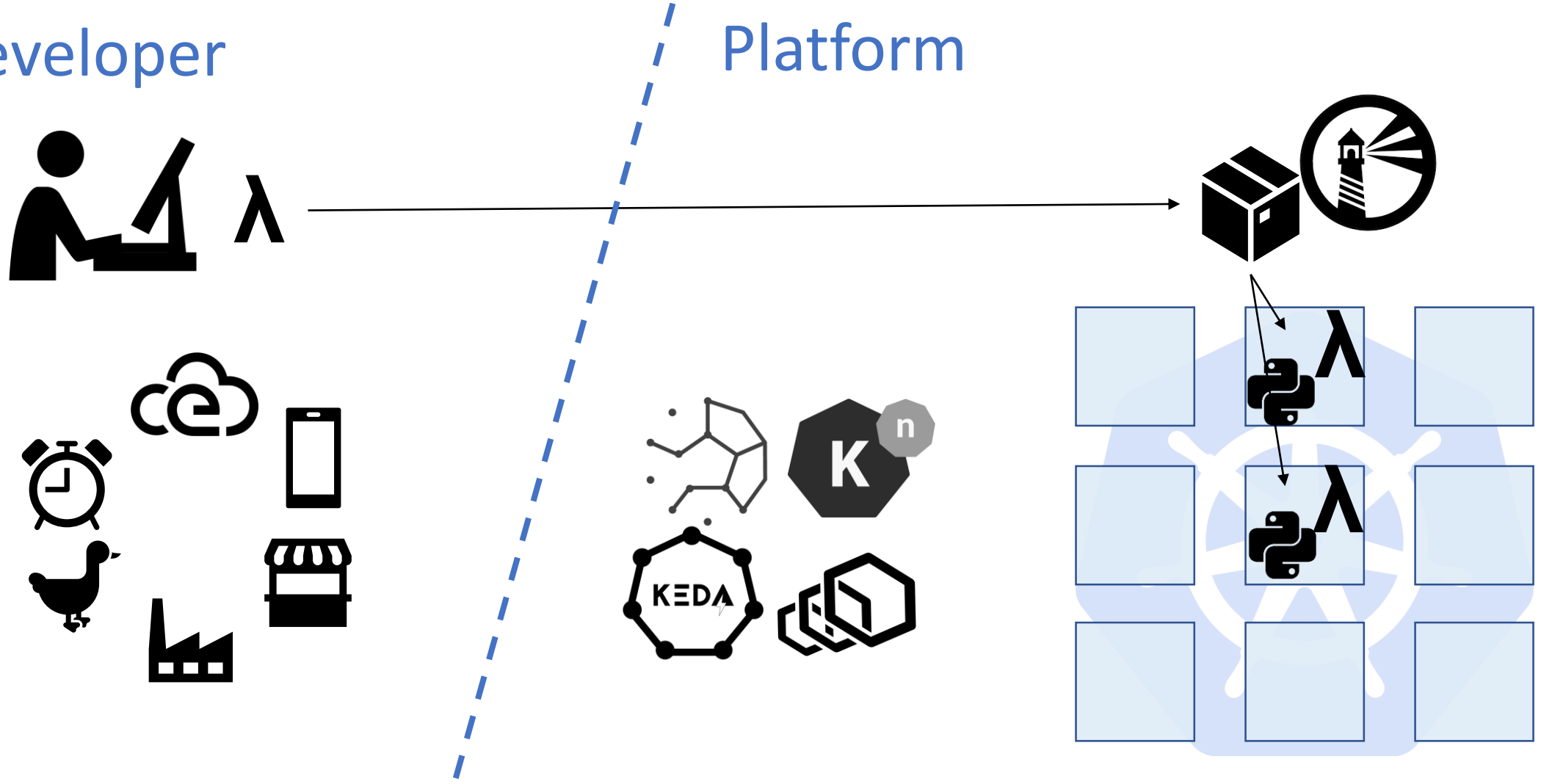
Leveraging Service Meshes for Accelerating Serverless Workflows

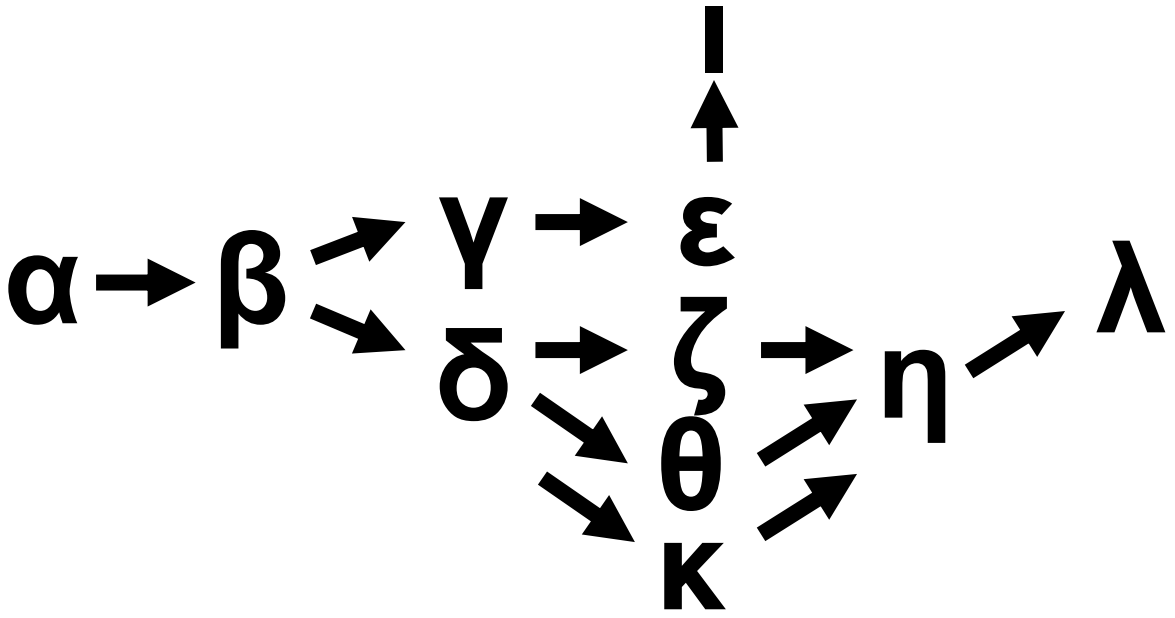
Paarijaat Aditya & Manuel Stein
Nokia Bell Labs

Serverless



Applications are not just one function

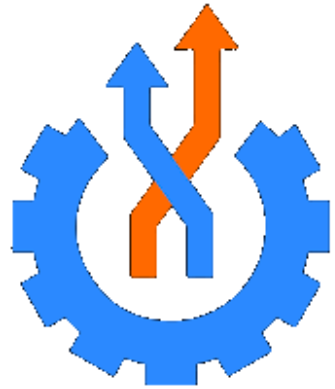
λ κ ι θ η ζ ε δ γ β α



Serverless Workflows

What's in a workflow?

- Compose multiple invocations (actions, steps, tasks)
- Specifies a control flow (as task graph, flowchart or state chart)
- Provides a common context to invocations (artifacts, workflow data)



serverlessworkflow.io

How can a platform achieve fast workflow completion?

- Invoke actions according to the control flow
- Pass the workflow context from one execution to the next

Outline

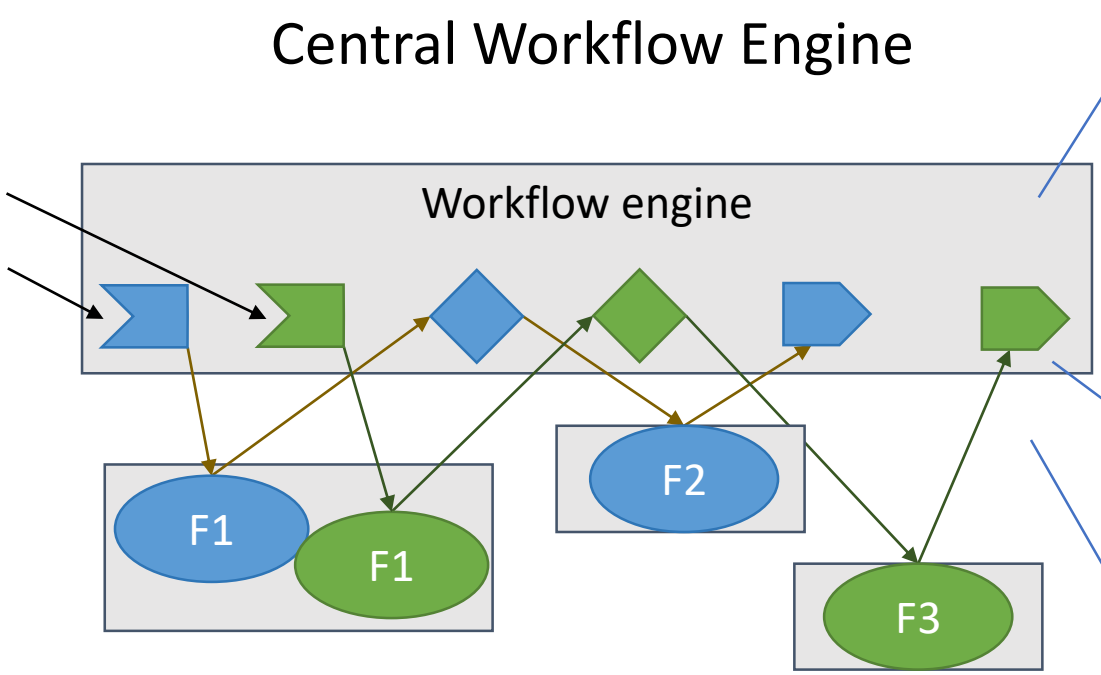
Communication

- The decentralized approach
- Which communication pattern to use
  1) Simple Services
  2) Knative Serving
  3) Knative Eventing Sequence
  4) Eventing + Serving

Grouping and Load Balancing

- Colocating functions to avoid communication
- Balance load across allocated resources
- Service mesh as an enabler for dynamic rebalancing

Platform design



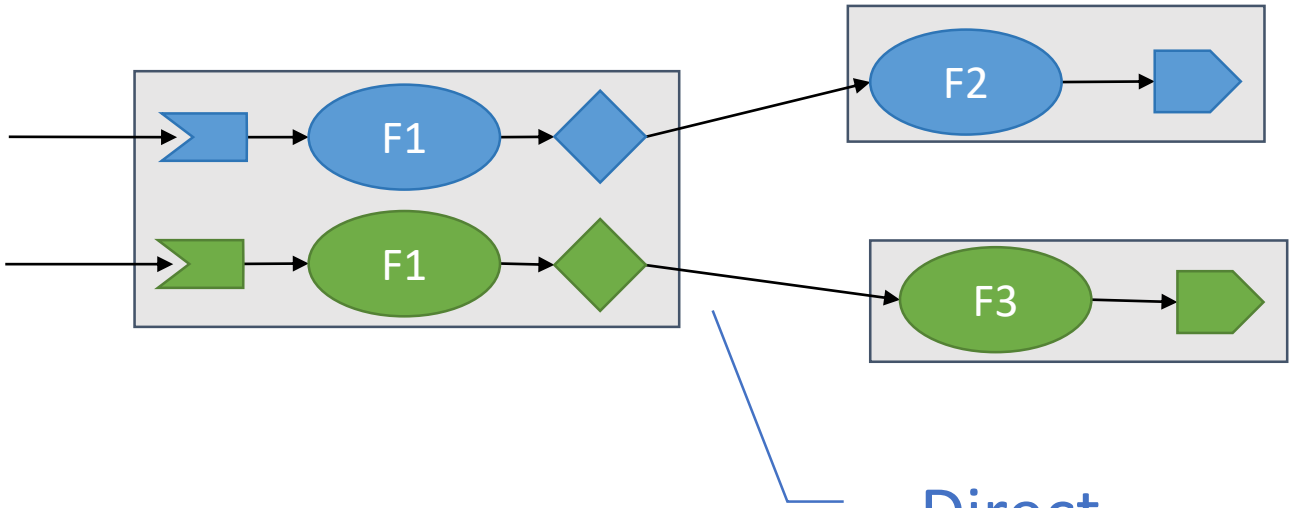
Central component involved in all communication

Additional latency ☹

Potential bottleneck ☹

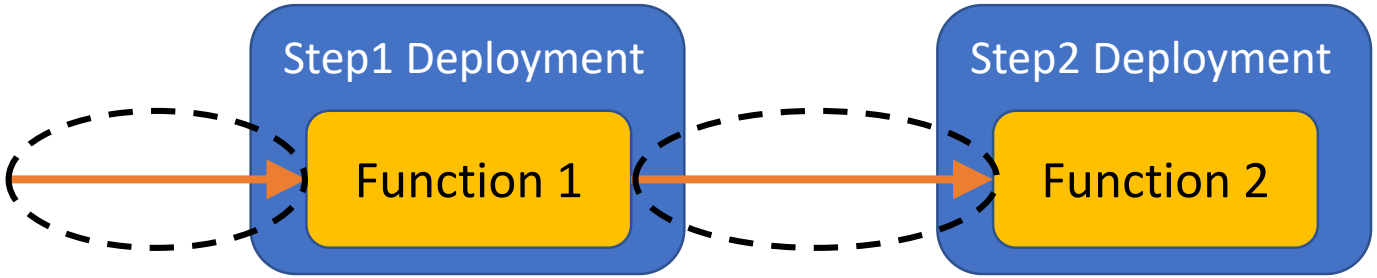
Platform design

Decentralized logic and functions

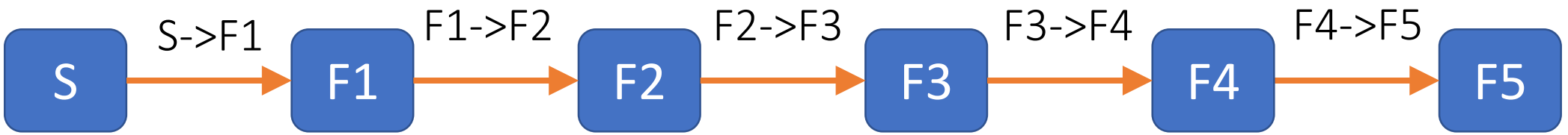


Direct communication ☺

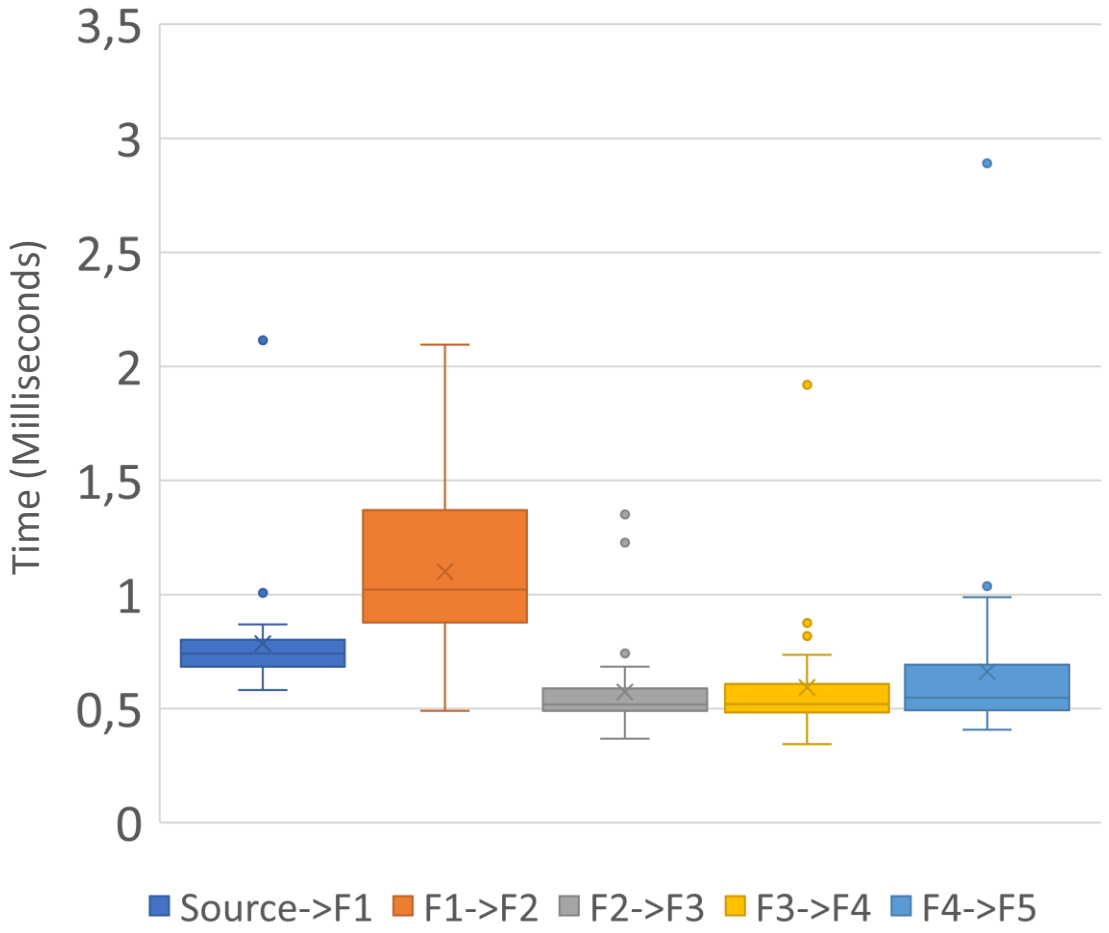
1. Basic Services



- Kubernetes Deployments exposed through a ClusterIP Service
- Event loop implements asynchronous delivery



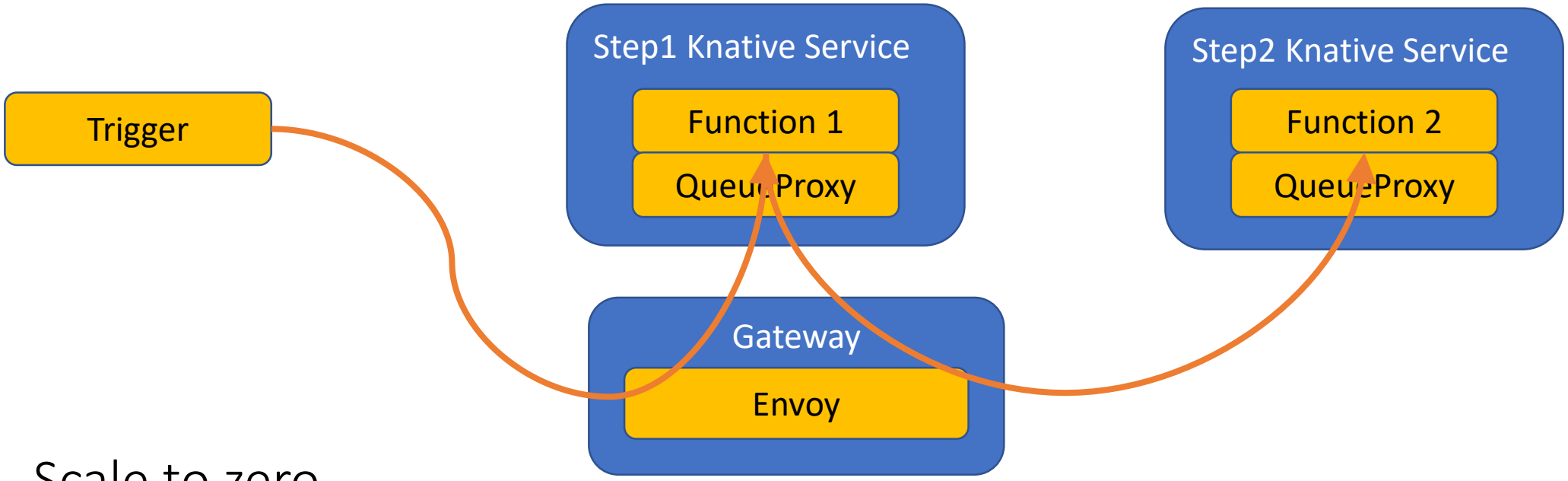
1. Basic Services



Series of 5 Services

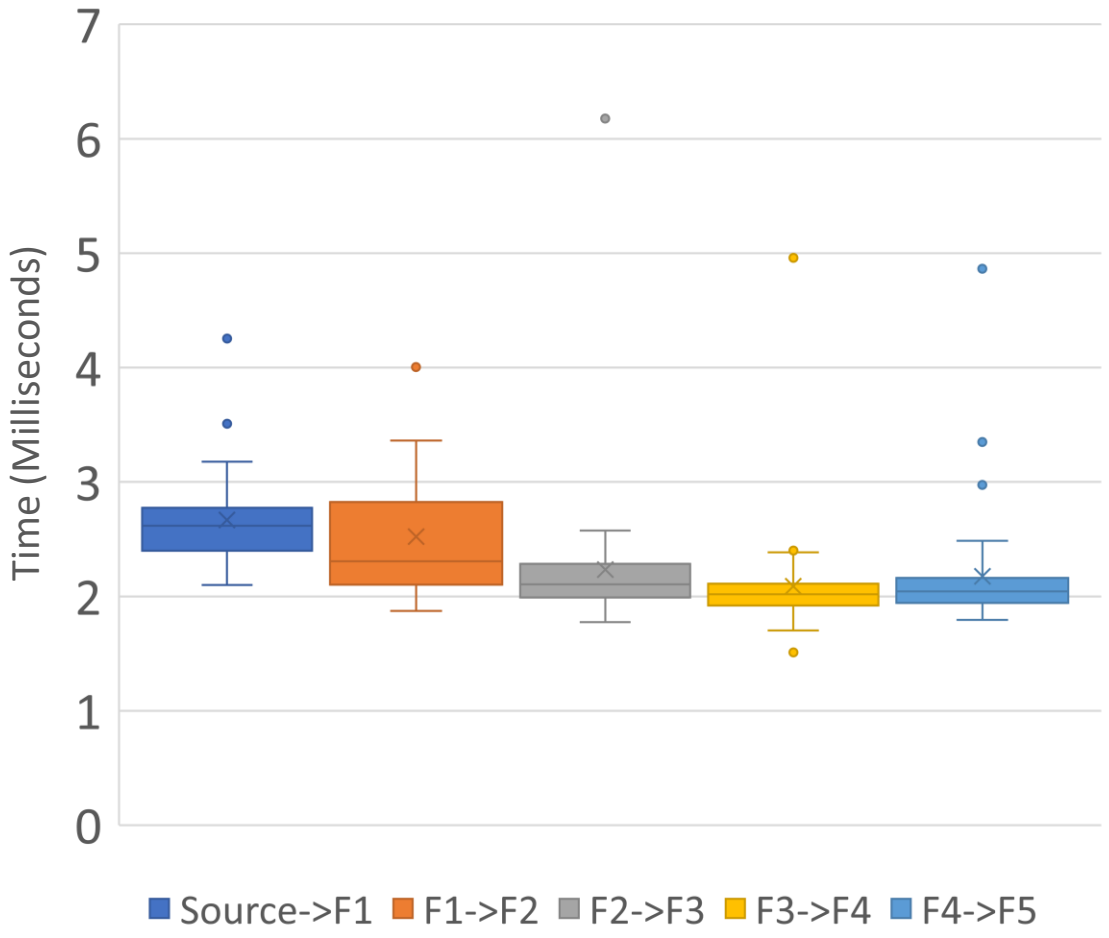
- CloudEvent with 1kB data
- Time between steps
- Median time is 0.62ms

2. Knative Serving



- Scale to zero
- Traffic splitting
- Revision management

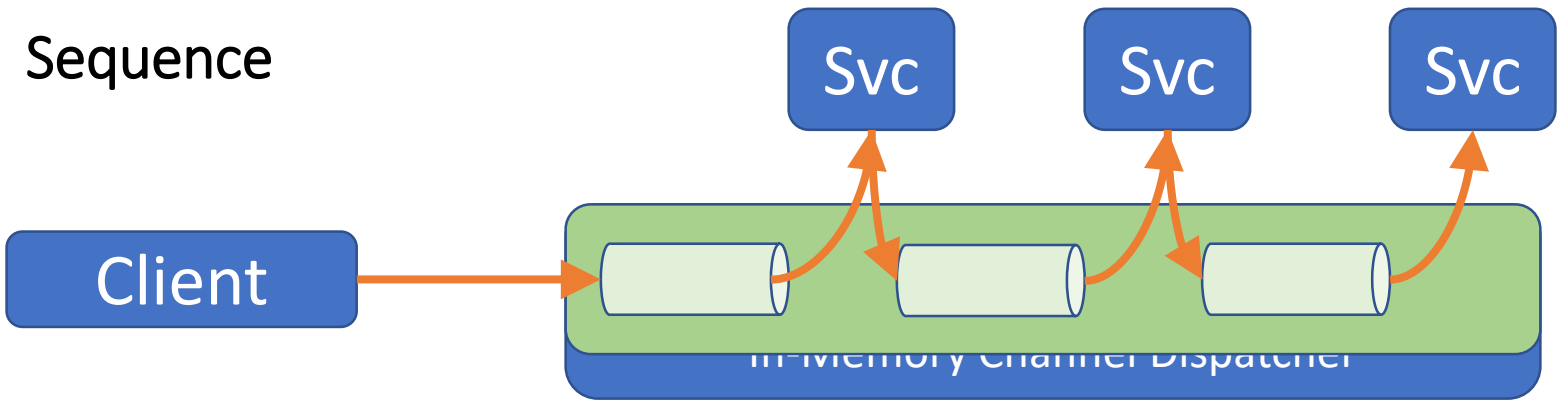
2. Knative Serving



Series of 5 Knative Services

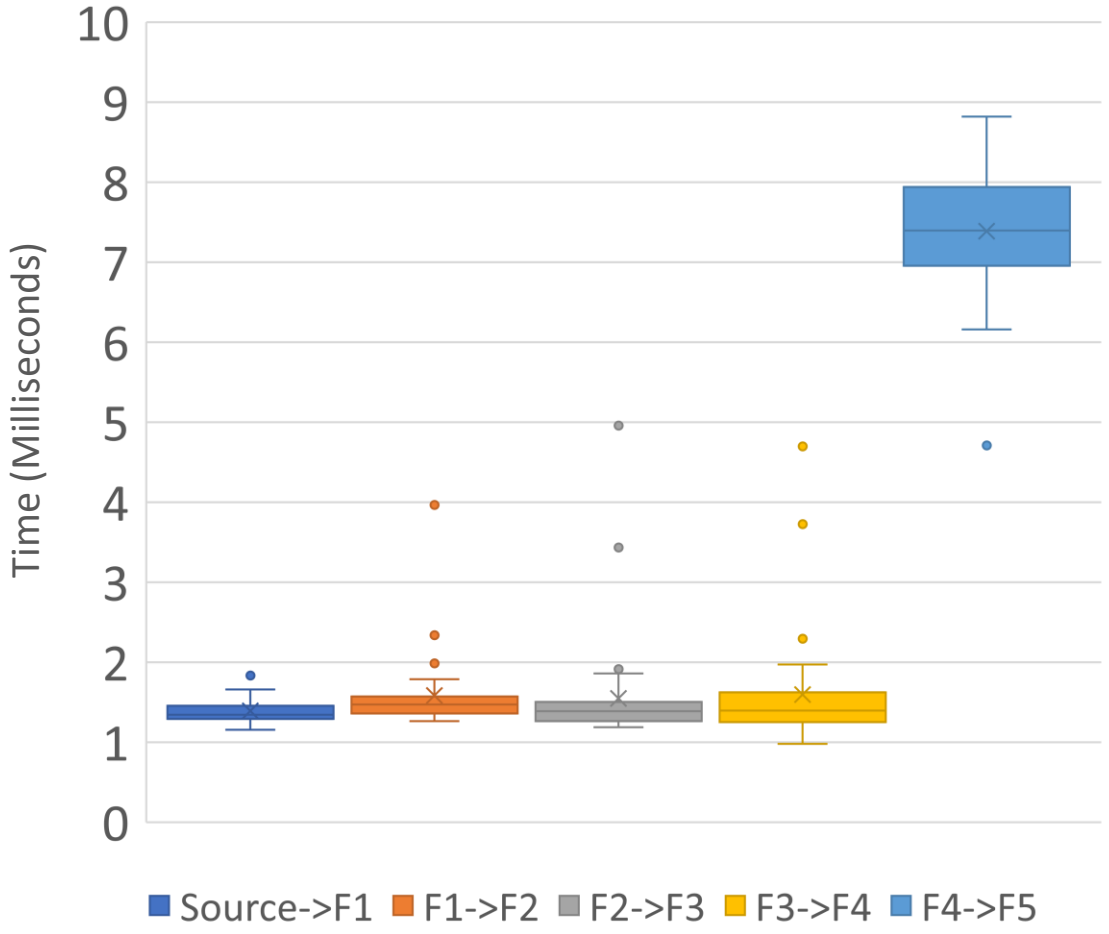
- CloudEvent with 1kB data
- Median time is 2.15 ms

3. Knative Eventing Sequence



- A sequence is a simple pipeline using channels
- Channel calls a destination (step)
- Step‘s response is fed to the next channel
- Technology is pluggable (In-memory / NATS / Kafka)
- Decoupling typically uses a store-and-forward pattern

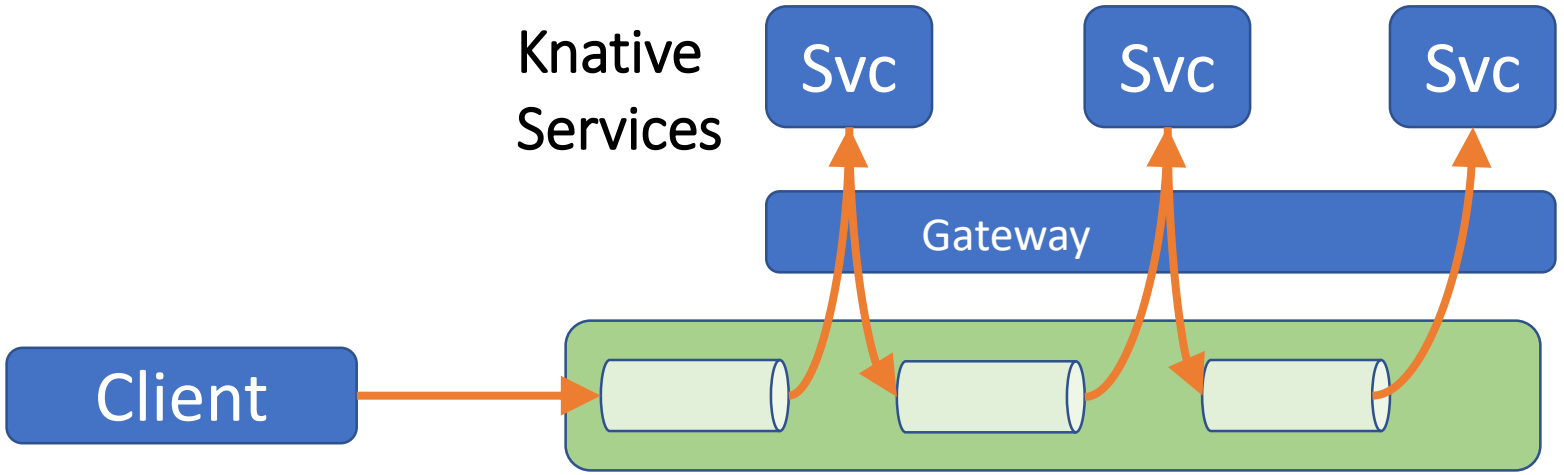
3. Knative Eventing Sequence



Sequence of 5 steps

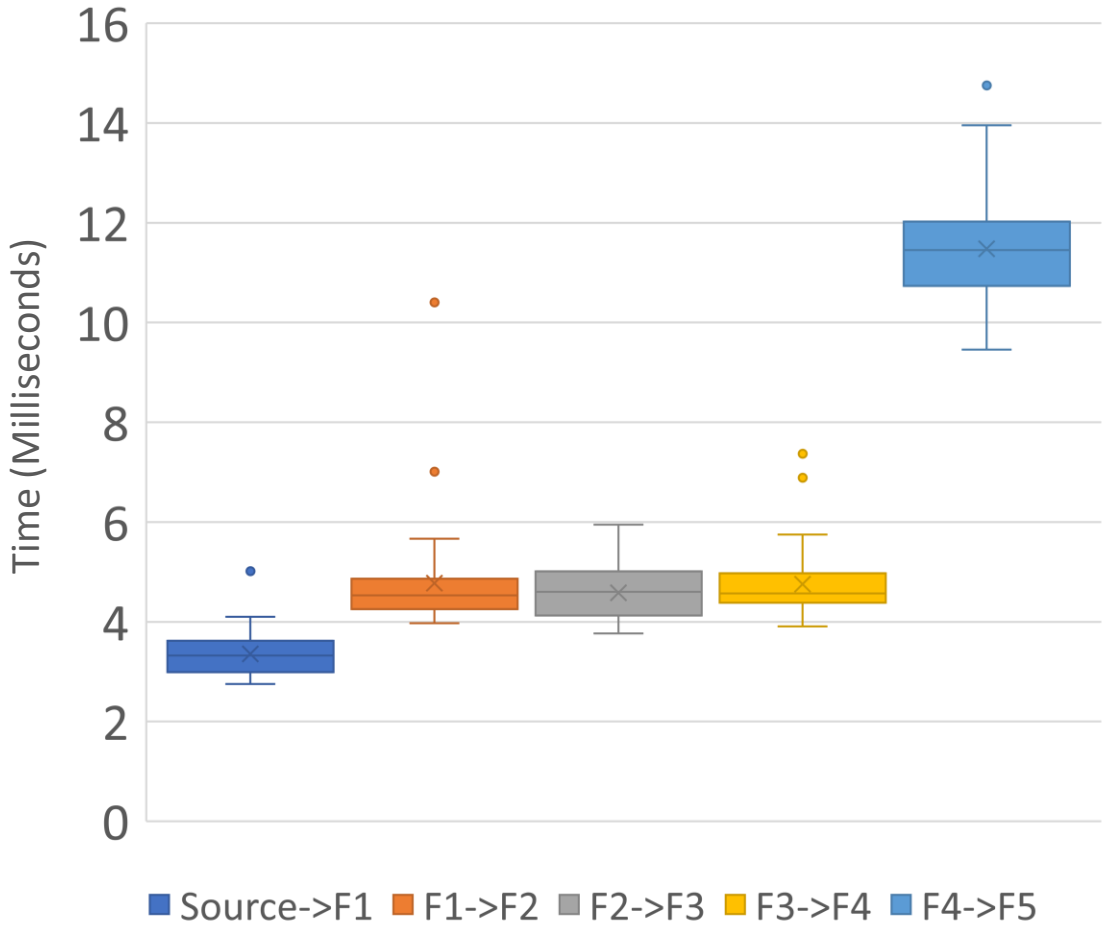
- Destinations are basic Services
- Delivery time from one response to the next destination
- CloudEvent with 1kB data
- Median latency is 1.45 ms
- (1.40 ms without last step)

4. Eventing + Serving



- Decoupling and late-binding
- Scale-to-zero and revision management

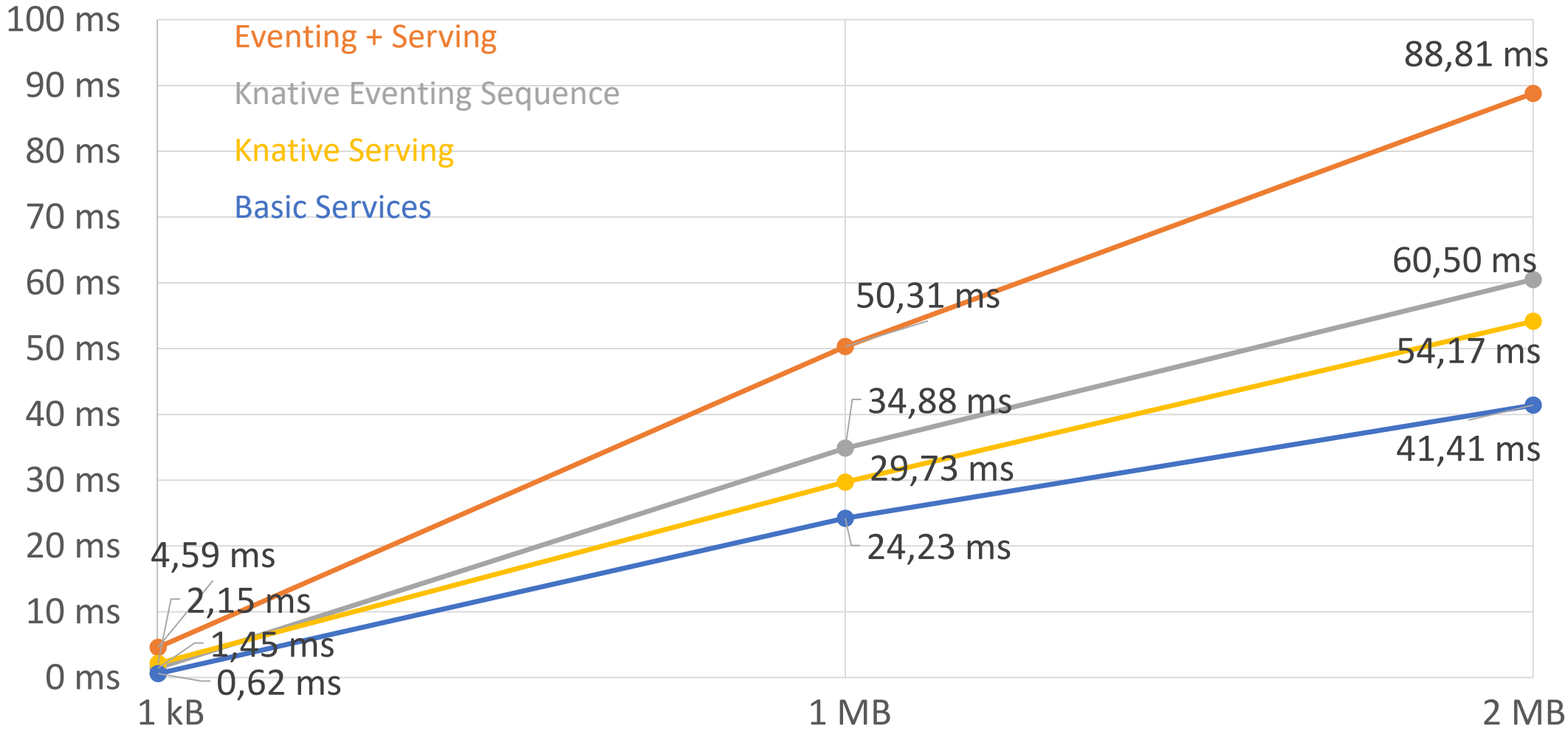
4. Eventing + Serving



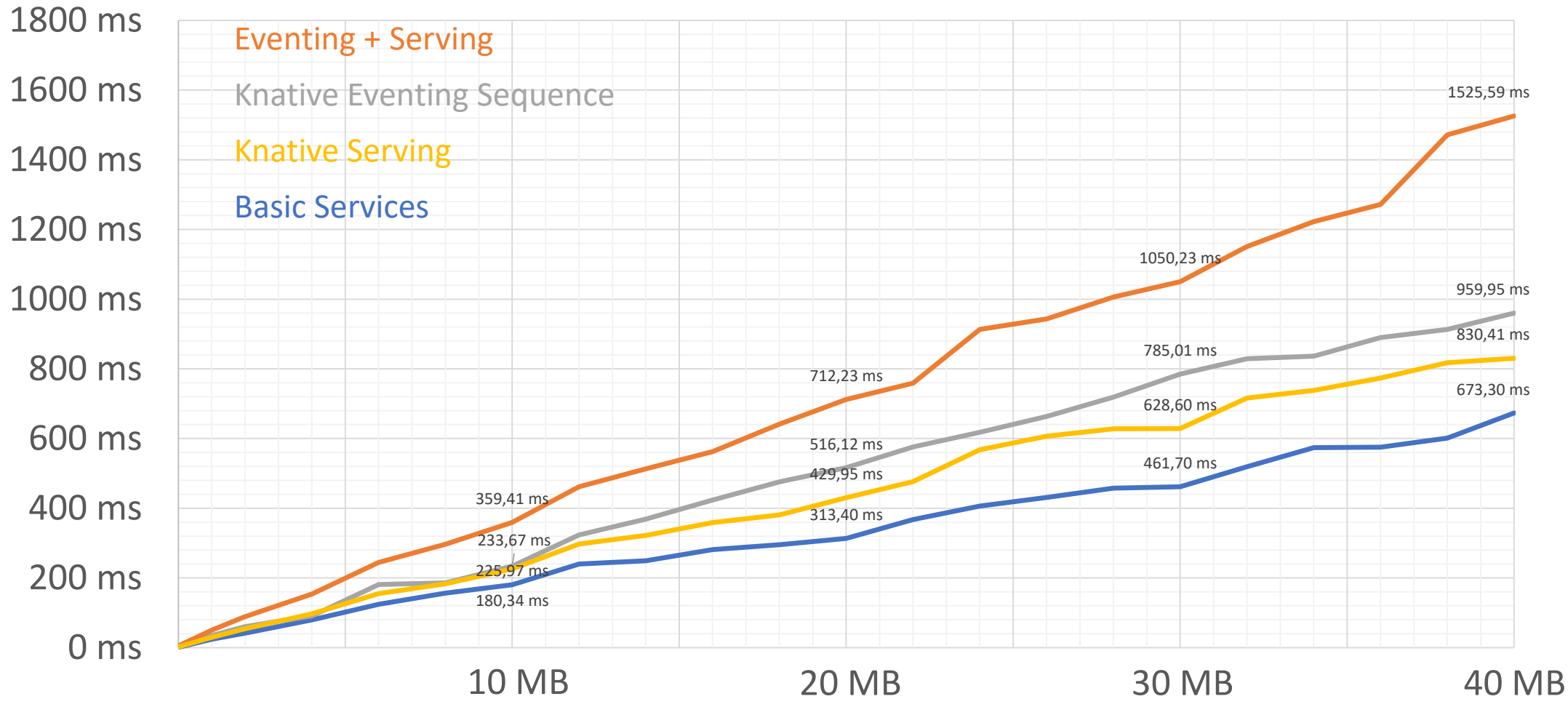
Sequence of 5 steps

- Destinations are Knative Services
- Delivery time for 1kB CloudEvent
- Median latency is 4.59 ms
- (4.37 ms without last step)

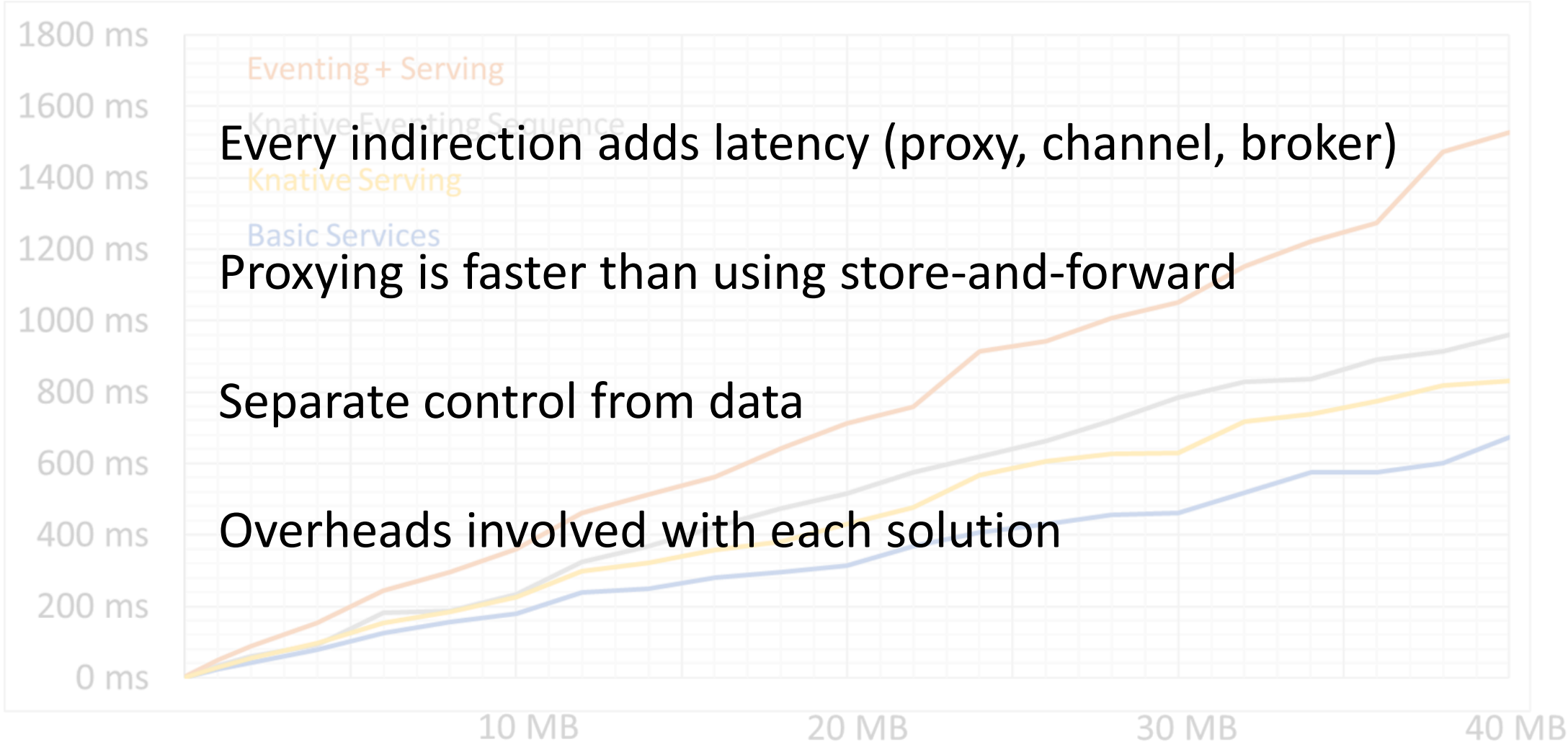
Different data sizes



Different data sizes



Summary



Every indirection adds latency (proxy, channel, broker)

Proxying is faster than using store-and-forward

Separate control from data

Overheads involved with each solution

Outline

Communication patterns

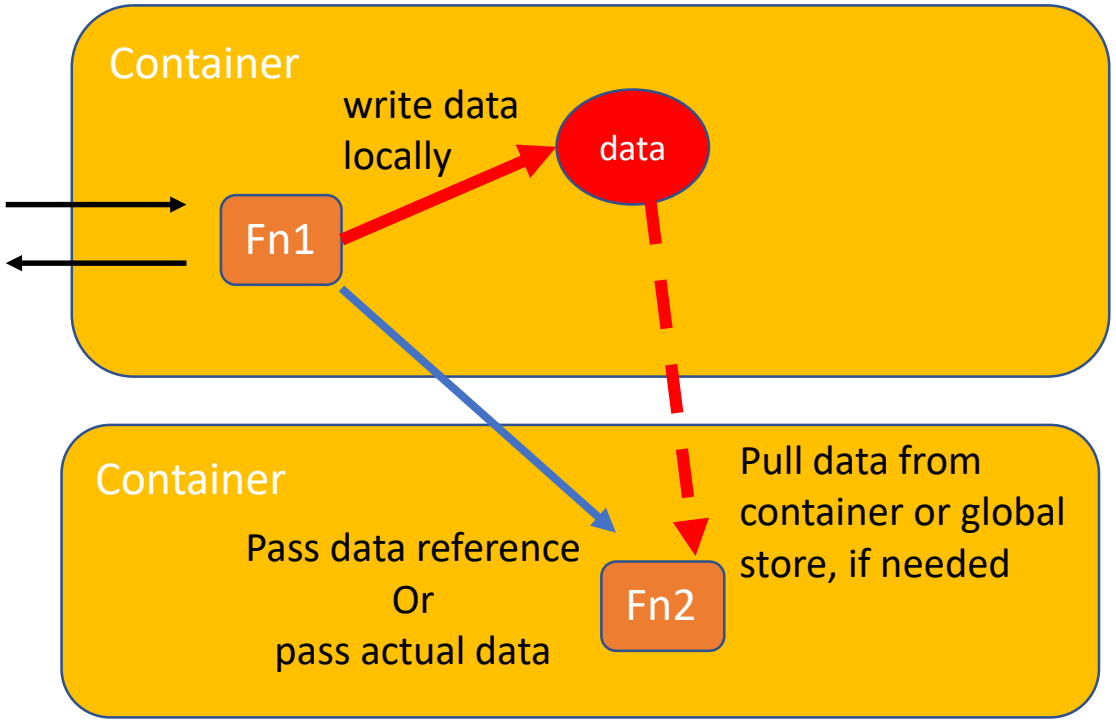
- The decentralized approach
- Which communication pattern to use
  1) Simple Services
  2) Knative Serving
  3) Knative Eventing Sequence
  4) Eventing + Serving

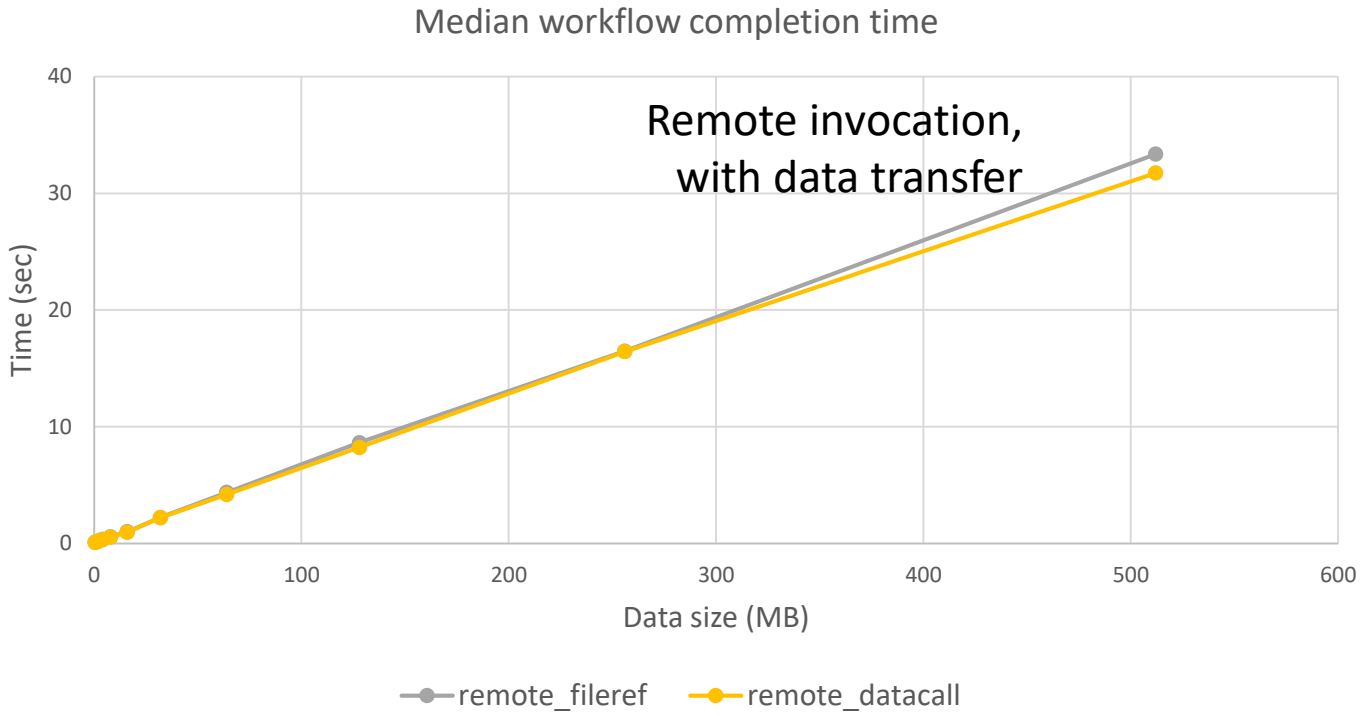
Grouping and Load Balancing

- Colocating functions to avoid communication
- Balance load across allocations
- Service mesh as an enabler for dynamic rebalancing

Overheads in transferring large data

- Previous approaches assume that each function is located in a different container
- Transferring large amount of data may create a significant overhead and slow down workflow completion time



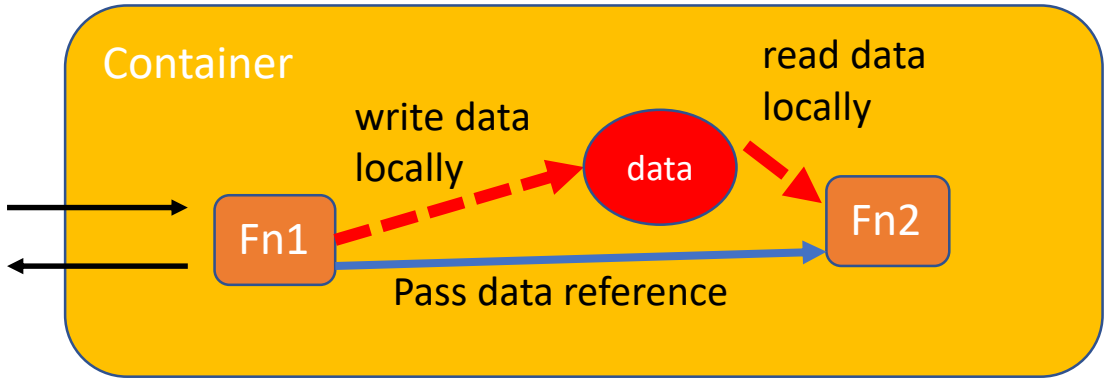


'remote' invocation of f2, with a data 'reference'

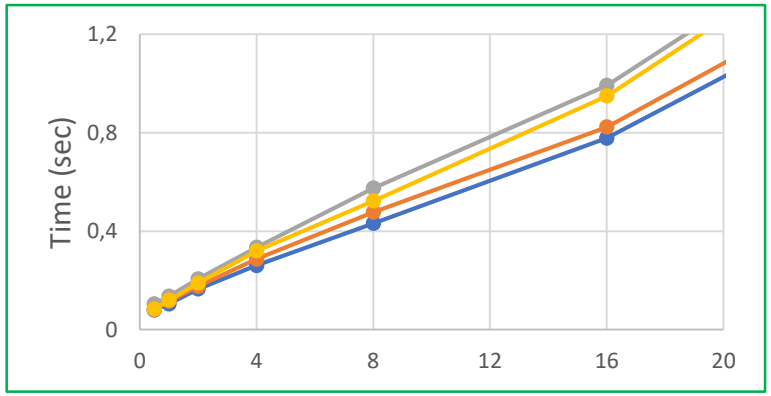
'fileref' = sending reference to data, and then pulling data if needed
'datacall' = sending actual data

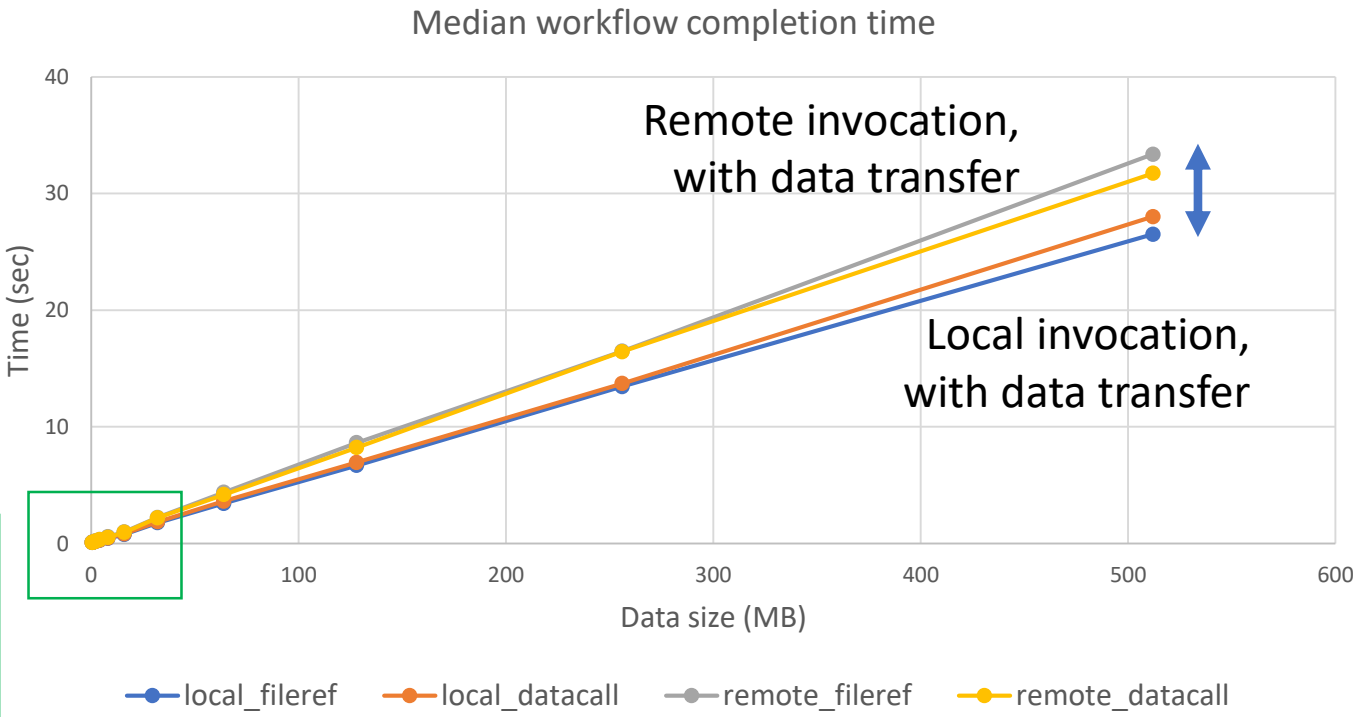
Colocating functions to accelarate workflows

- Colocate multiple functions in a container to further accelerte workflow completion time
- Keep data local (local or shared-mem filesystem), pass a reference to data



'local' invocation of f2, with a data 'reference'

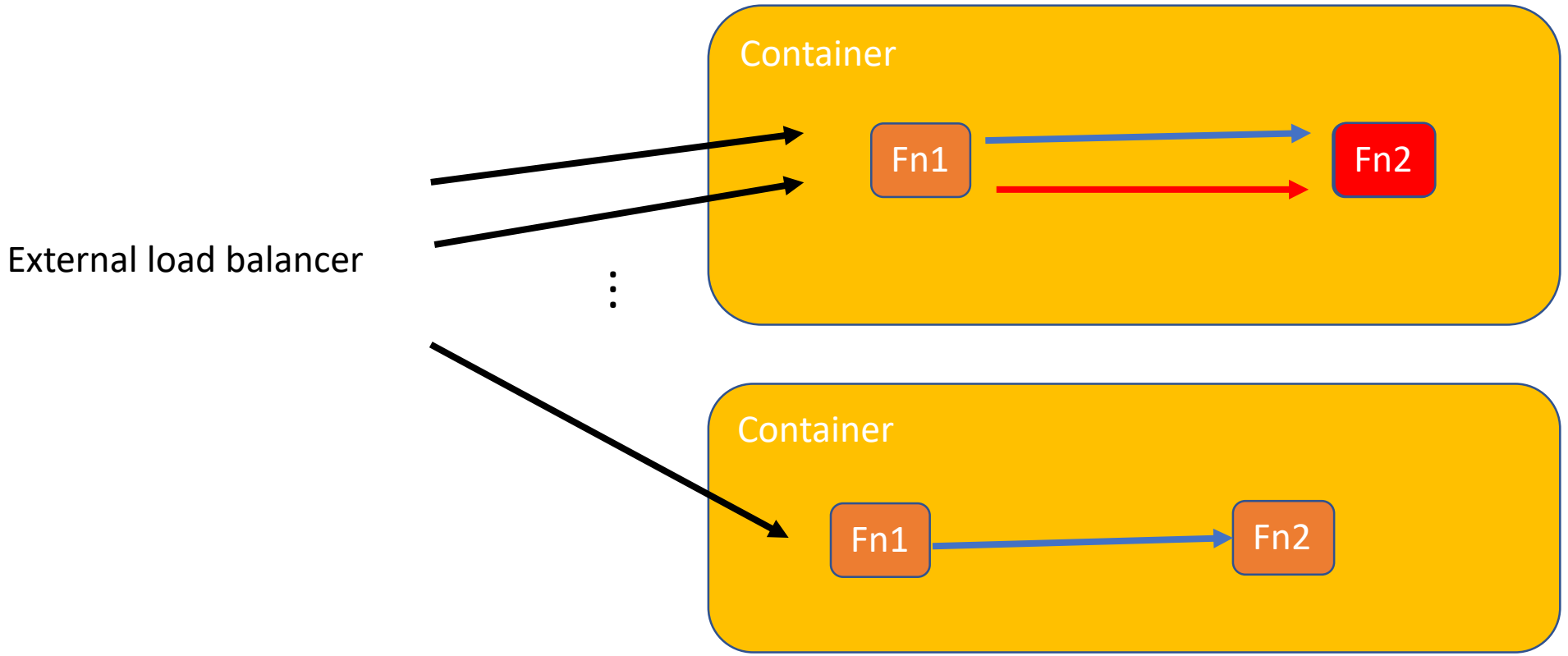




Keeping data local is better

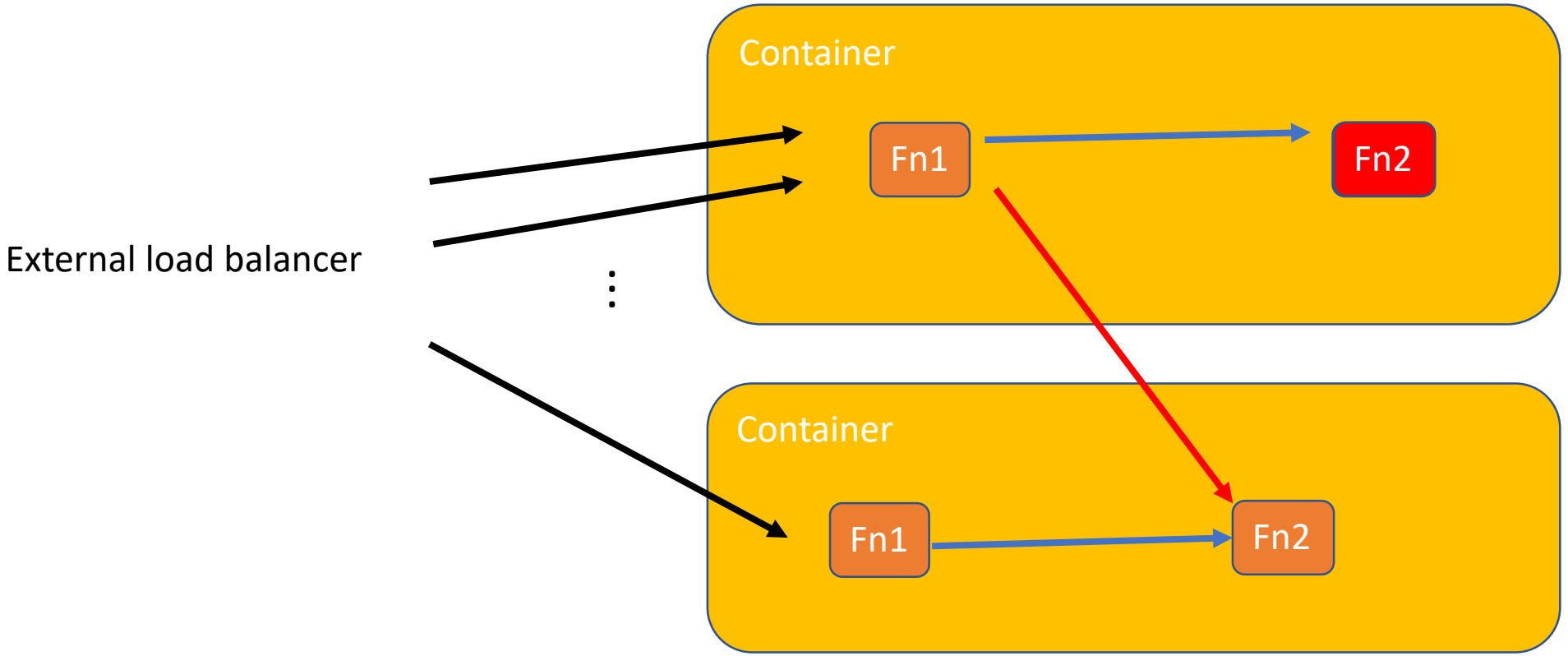
Need for internal load rebalance

- At the time of admitting a request, the container may not be aware of congestion in a latter part of a workflow
- The container may admit more requests that it is able to process downstream



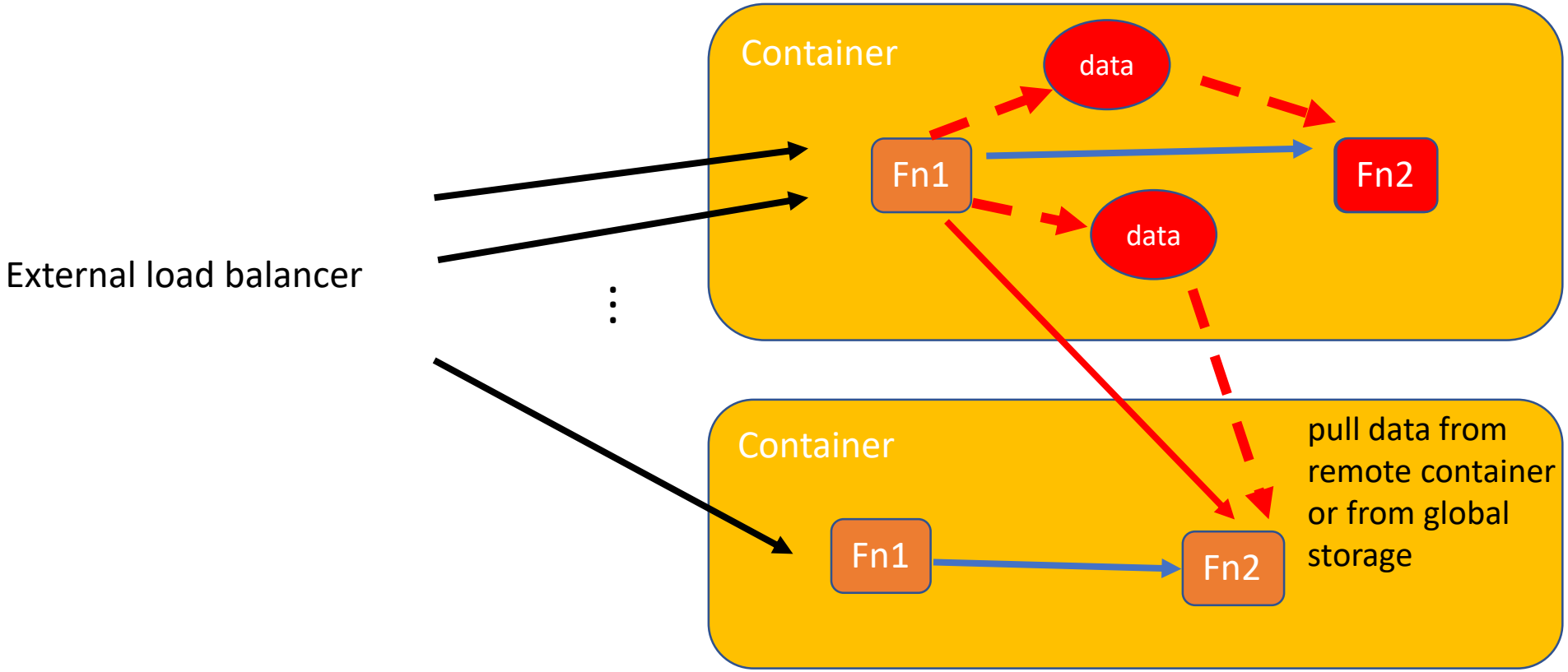
Need for internal load rebalance

- Need to rebalance already admitted requests
- Internally route requests to other replicas of downstream function

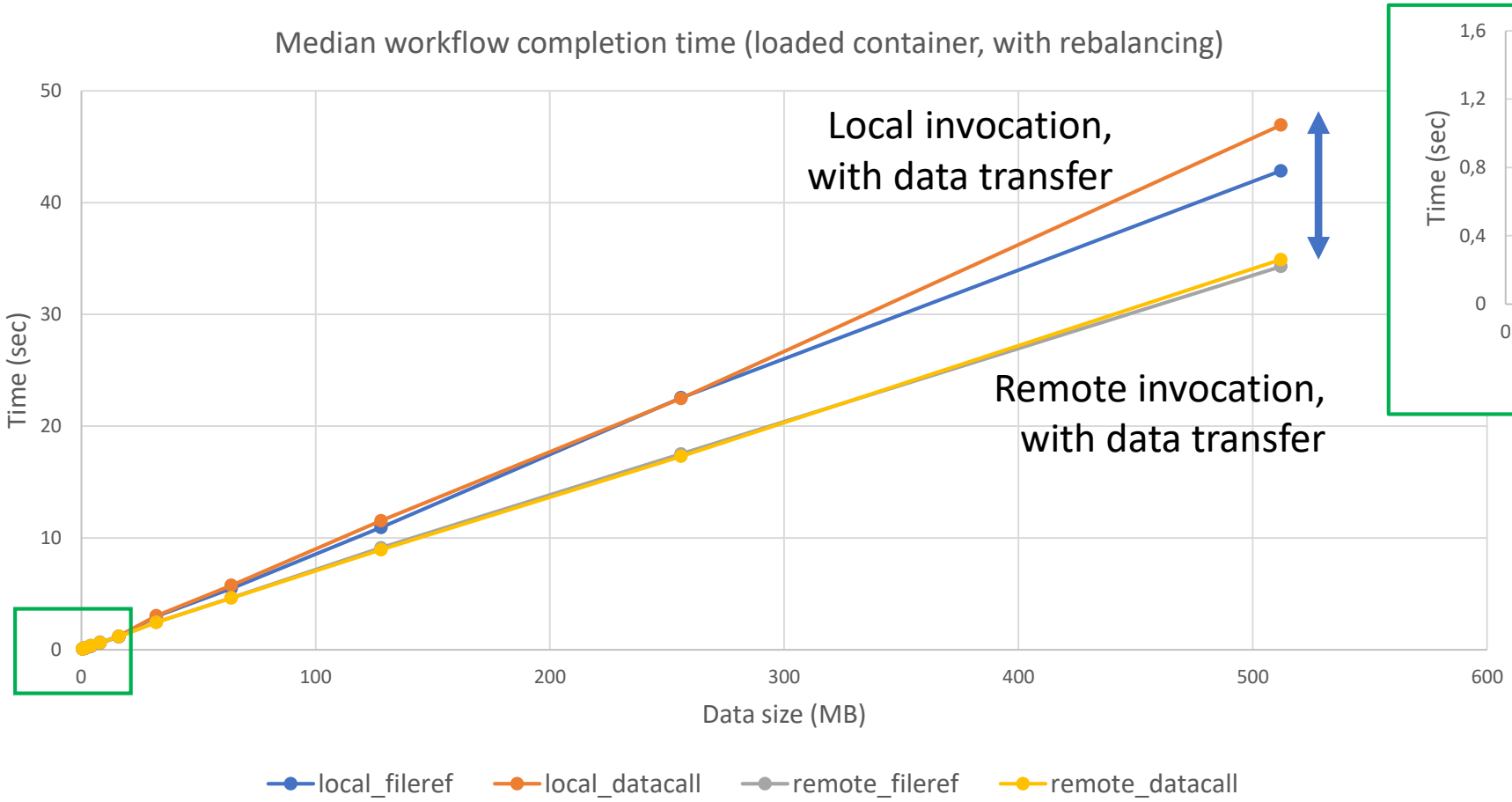


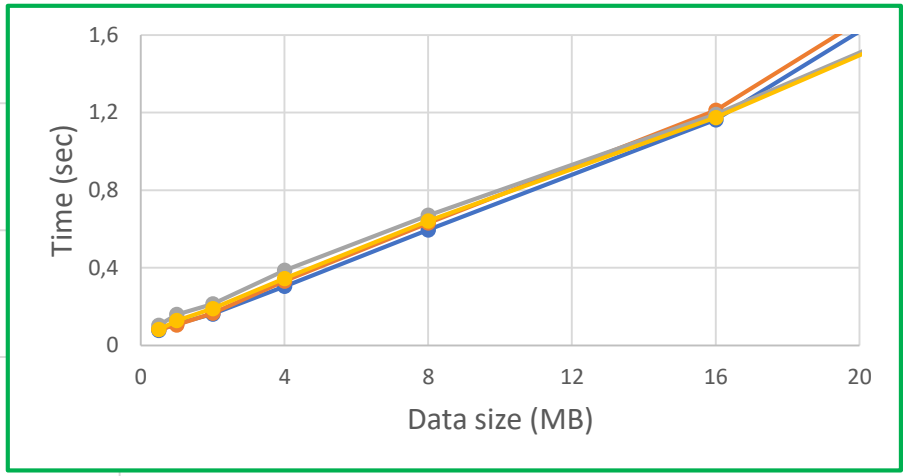
Load rebalancing with data transfer

- Transferring data makes rebalancing even more challenging



Load rebalancing with data transfer





'fileref' = sending reference to data, and then pulling data if needed

'datacall' = sending actual data

Under load, it may be better to rebalance to a replica of F2 in a different container, even if it involves data transfer

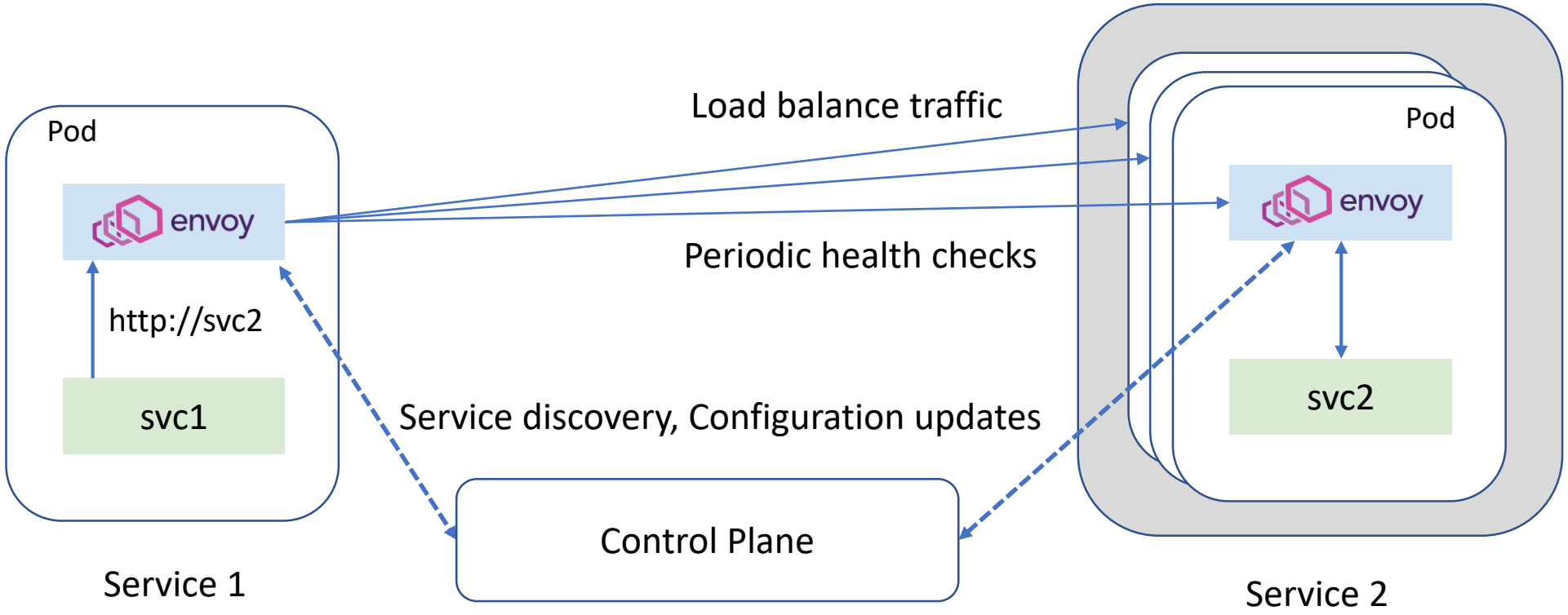
Dynamically configurable communication

- Co-locating multiple functions of a workflow can accelerate workflows
- But now you are dealing with a large unit of deployment
- May have to dynamically rebalance load from somewhere in the middle of a workflow, which you not know beforehand

- Need a flexible, dynamically configurable routing/communication mechanism
- Need fine grained observability
- Need configurable load balancing without modifying core logic of the app

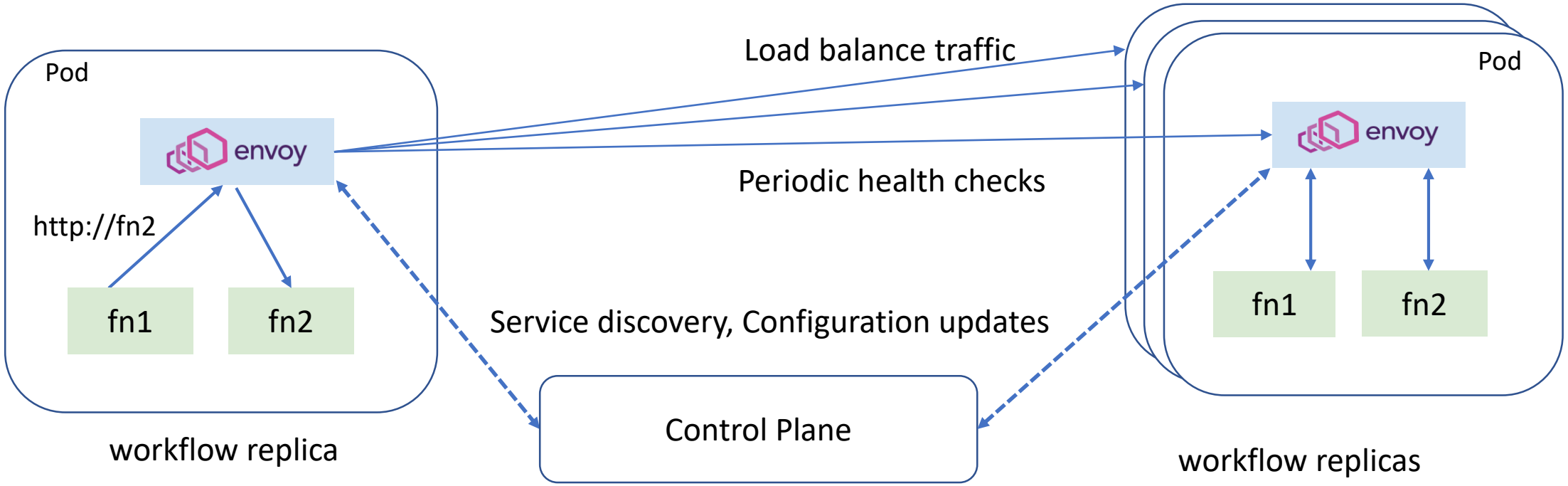
Service mesh as an enabler

- Provides dynamically configurable routing at runtime,
- Configurable load balancing policies (e.g round-robin, weighted, maglev, etc),
- Detailed observability, periodic health checks



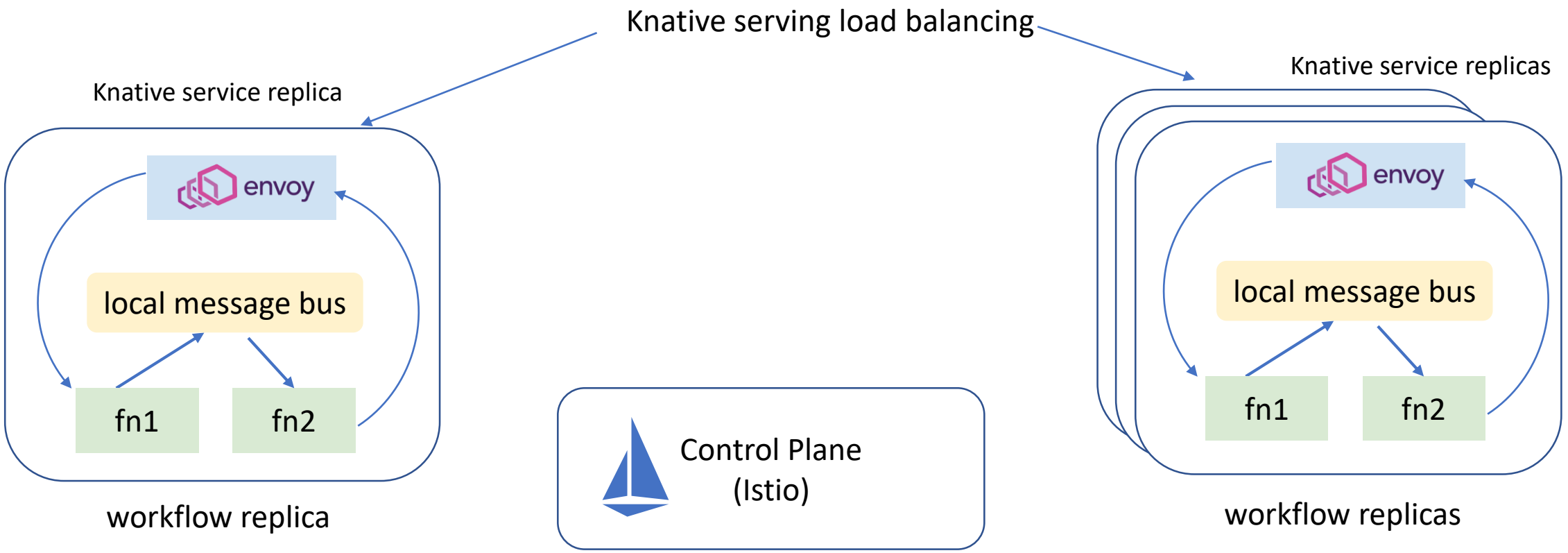
Service mesh as an enabler

- Reuse envoy proxy for intra-container communication (in addition to inter-container comm.)
- Preferentially send requests to local downstream function (e.g. weighted load-balancing)
- Allow functions to exert back pressure (e.g. via 503 responses to health checks)



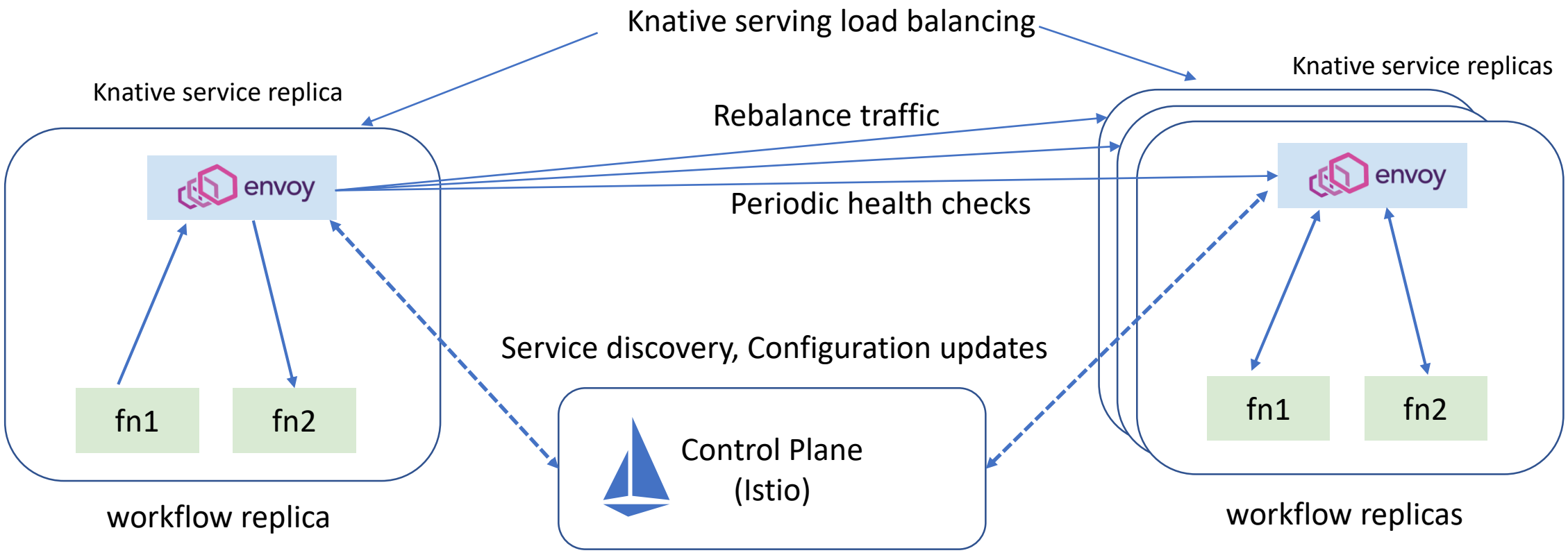
KNIX Microfunctions (knix.io)

- Colocate functions of a workflow inside single container (wrapped in a Knative service). Provides a custom local message bus within the container



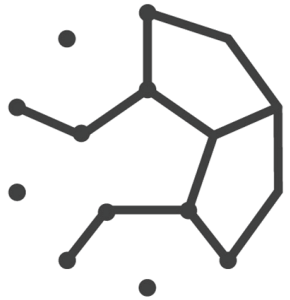
KNIX Microfunctions (knix.io)

- Colocate functions of a workflow inside single container (wrapped in a Knative service). Provides a custom local message bus within the container
- *[Coming soon]* Extending to utilize the envoy proxy (+ control plane) as a unified communication mechanism for intra- and inter-container communication and load rebalancing



Source code

KNIX Website: https://knix.io



KNIX Source code: https://github.com/knix-microfunctions/knix

KNIX Slack channel: https://knix.slack.com

Code for benchmark experiments presented in this talk:
https://github.com/knix-microfunctions/workflowmesh

Thank you!