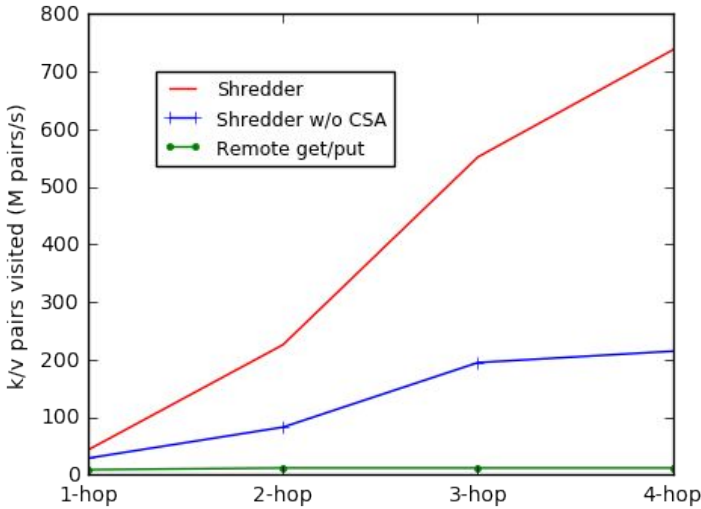
Narrowing the Gap Between Serverless and its State with Storage Functions

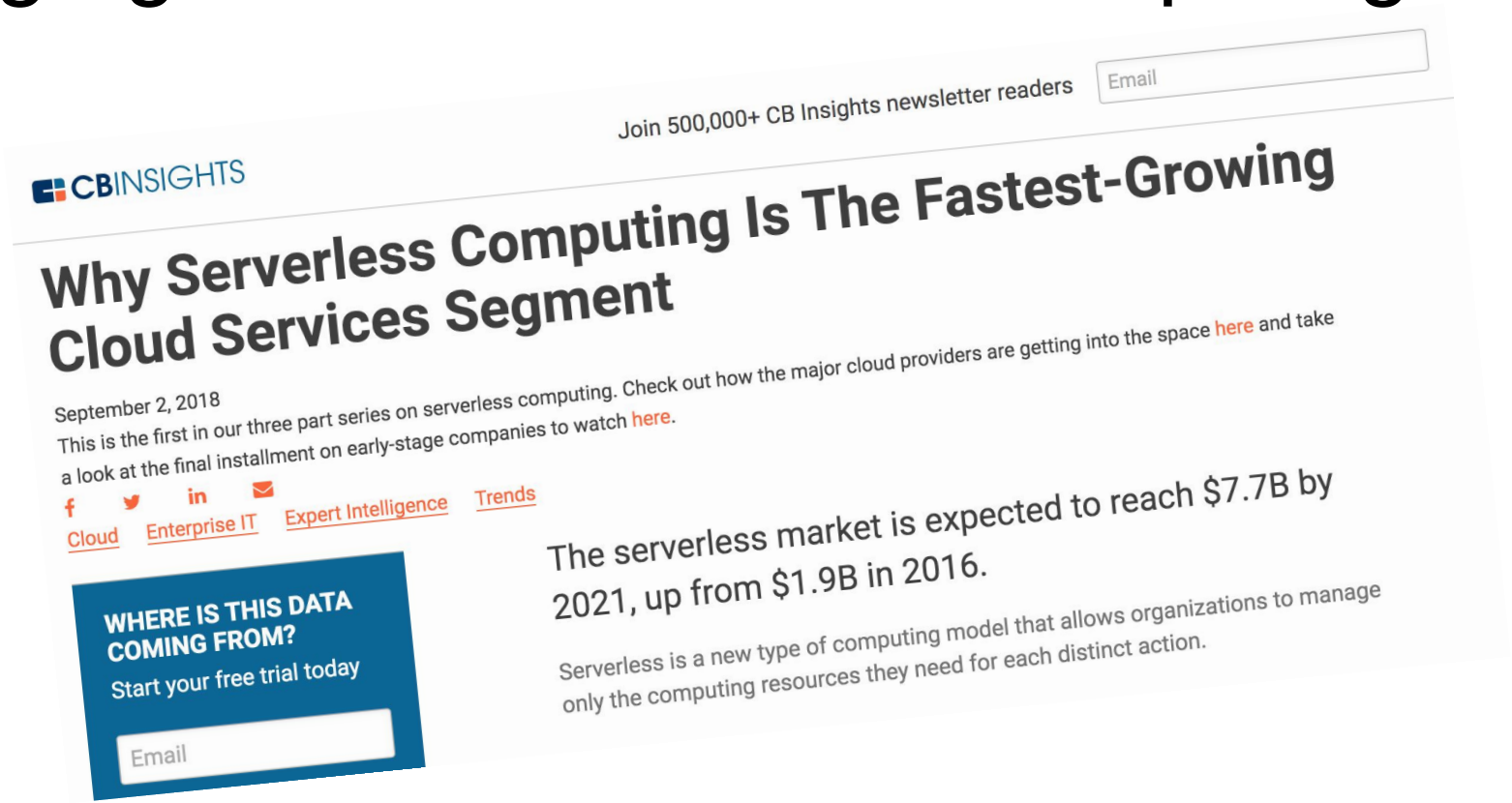
Tian Zhang, Dong Xie, Feifei Li, Ryan Stutsman

Shredder

- A multi-tenant in-memory key-value store.
- Extensible with user-provided storage function.
- **5 M** ops/s per machine, **~20 µs** latency
- In-runtime data access method, able to access **10s of GB** of data per second.



High growth of serverless computing



CBINSIGHTS

Join 500,000+ CB Insights newsletter readers

Email

Why Serverless Computing Is The Fastest-Growing Cloud Services Segment

September 2, 2018

This is the first in our three part series on serverless computing. Check out how the major cloud providers are getting into the space here and take a look at the final installment on early-stage companies to watch here.

Cloud Enterprise IT Expert Intelligence Trends

WHERE IS THIS DATA COMING FROM?

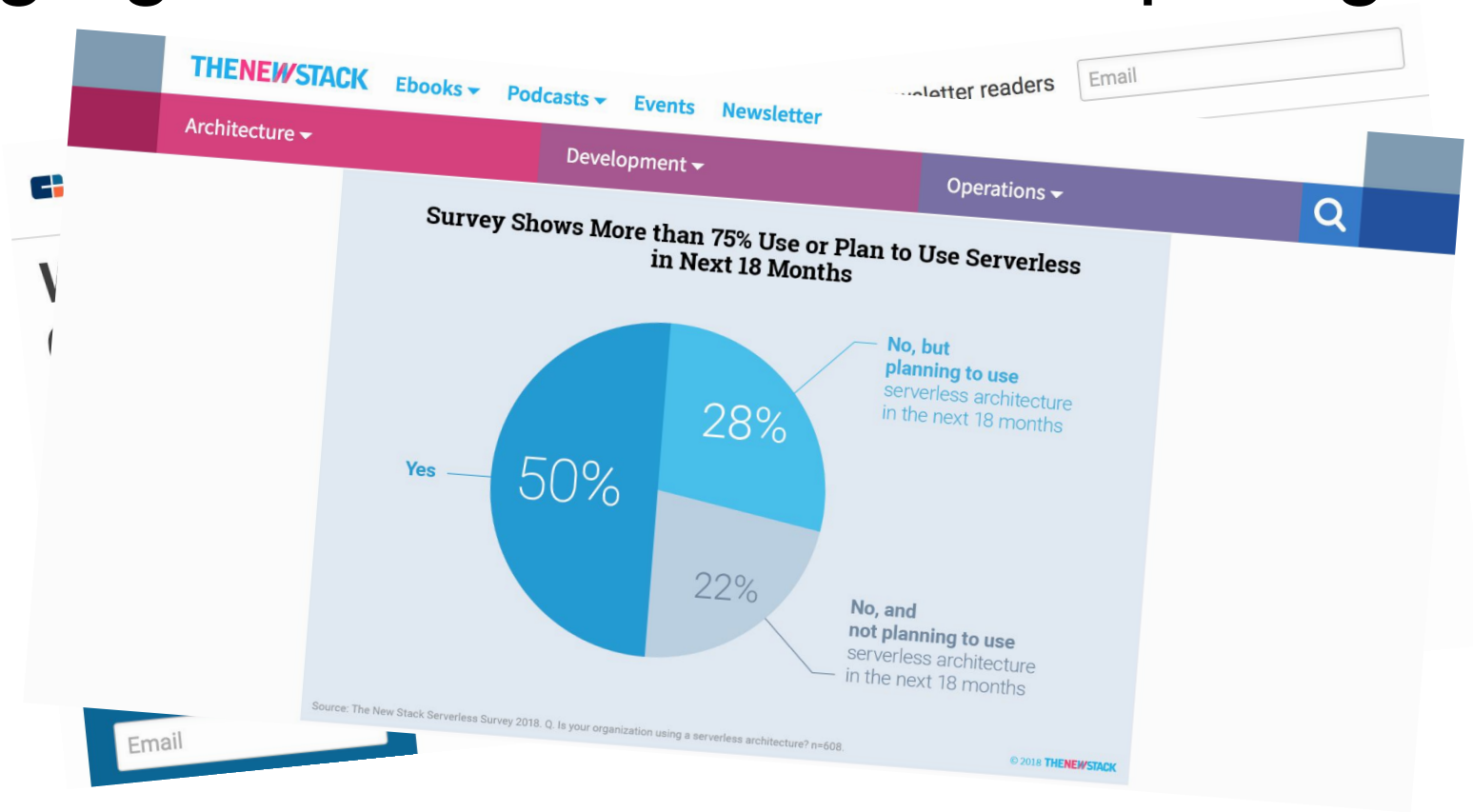
Start your free trial today

Email

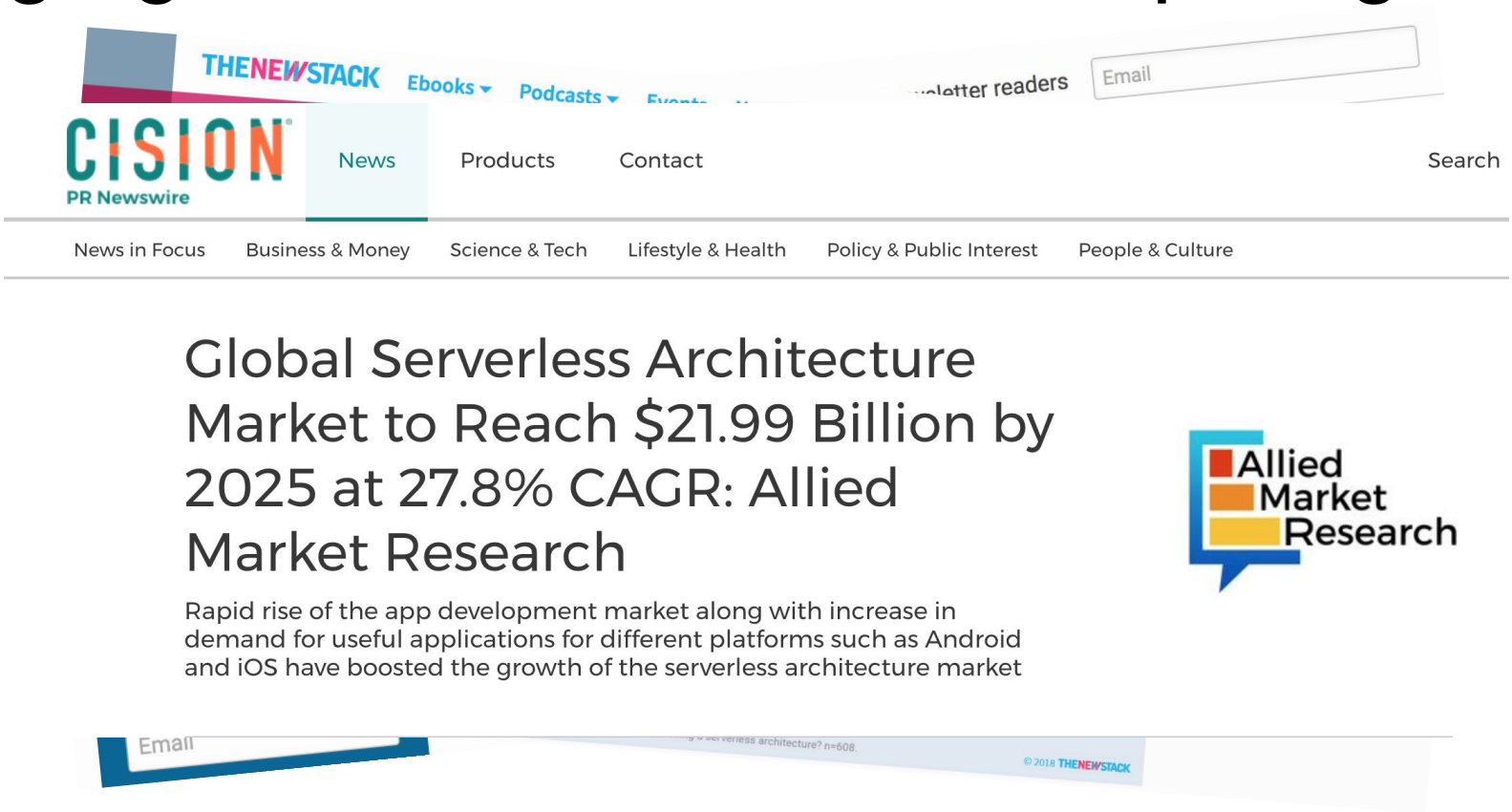
The serverless market is expected to reach $7.7B by 2021, up from $1.9B in 2016.

Serverless is a new type of computing model that allows organizations to manage only the computing resources they need for each distinct action.

High growth of serverless computing

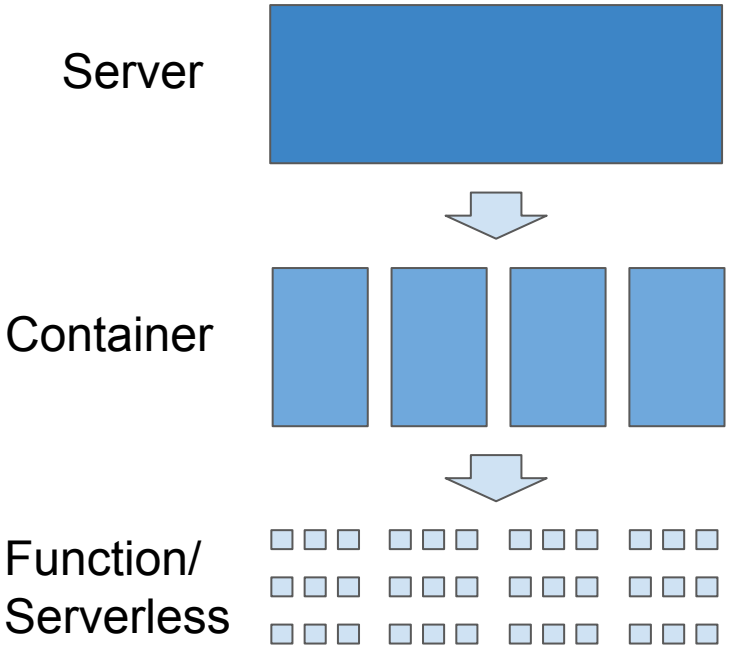


High growth of serverless computing



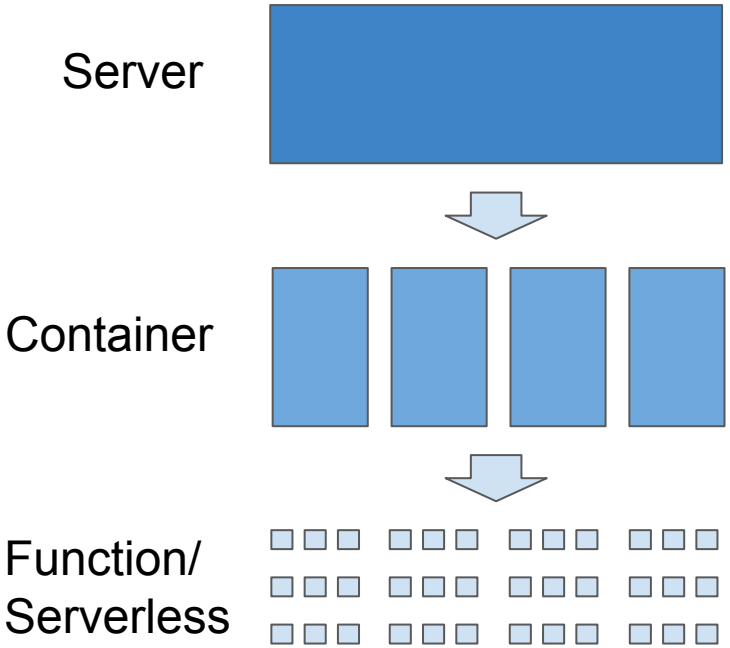
Advantages of serverless computing

- **Fine-grained** resource provisioning.

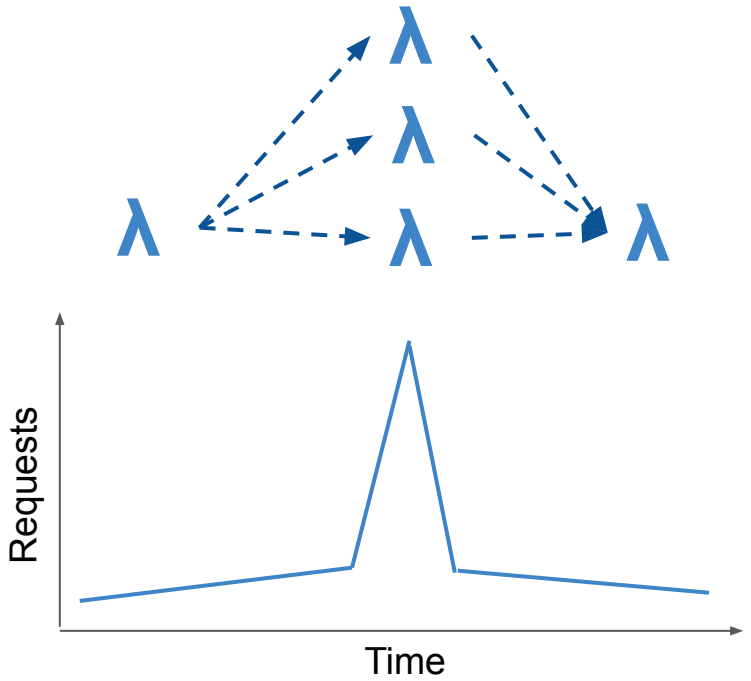


Advantages of serverless computing

- **Fine-grained** resource provisioning.

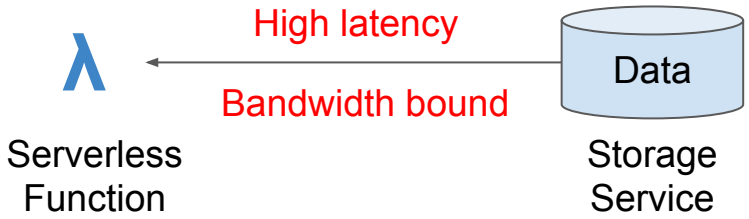


- **On-demand** scaling.



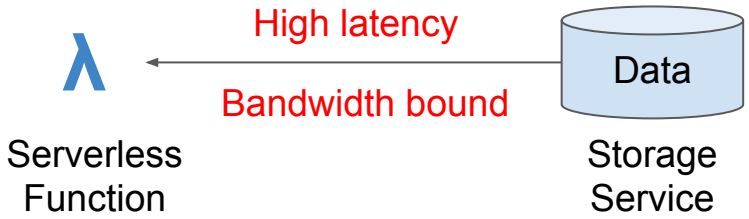
Problems of serverless computing

- **Shipping data to code** paradigm.

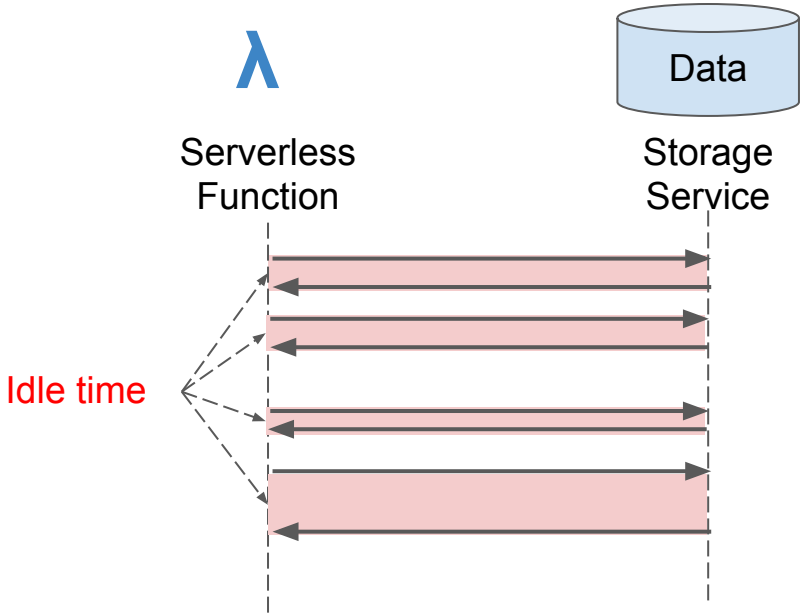


Problems of serverless computing

- **Shipping data to code** paradigm.



- User pay for additional **idle** time.



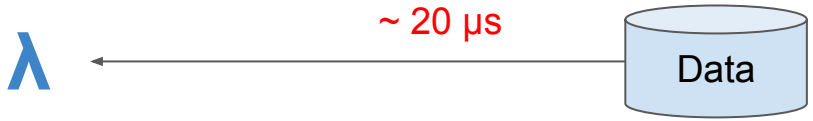
Narrowing the gap

Network costs between servers

Narrowing the gap

Network costs between servers

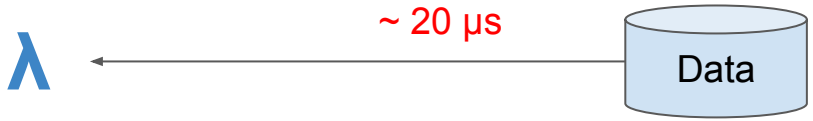
Kernel bypass to reduce latency



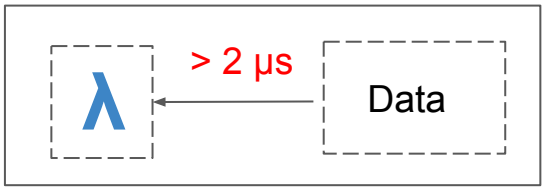
Narrowing the gap

Network costs between servers

Kernel bypass to reduce latency



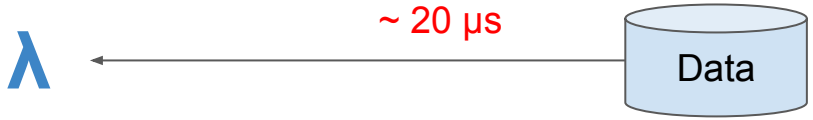
Push code to data, process isolation cost



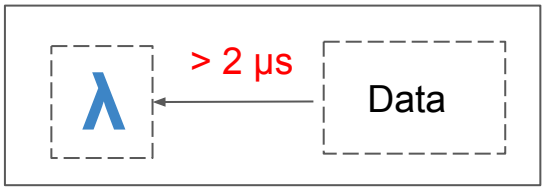
Narrowing the gap

Network costs between servers

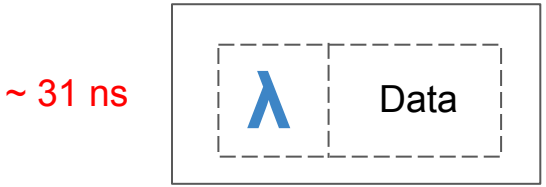
Kernel bypass to reduce latency



Push code to data, process isolation cost



V8 runtime isolation, boundary crossing cost

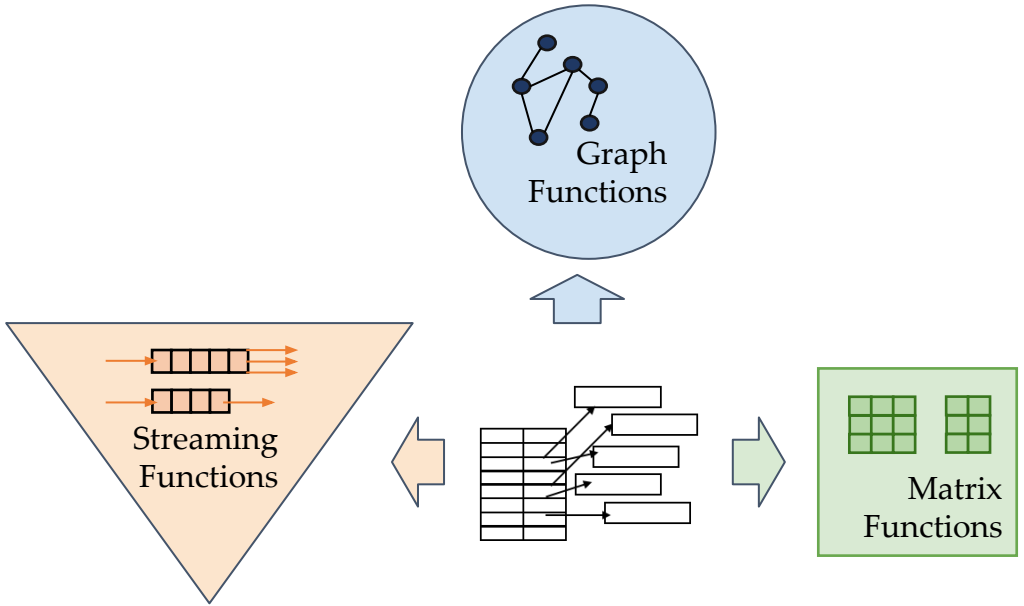


Shredder design goals

- **Programmability** - flexibility to implement any custom logic.
- **Isolation** - functions should be safely isolated.
- **High Density and Granularity** - should support thousands of tenants.
- **Performance** - optimize performance as much as possible.

Why JavaScript

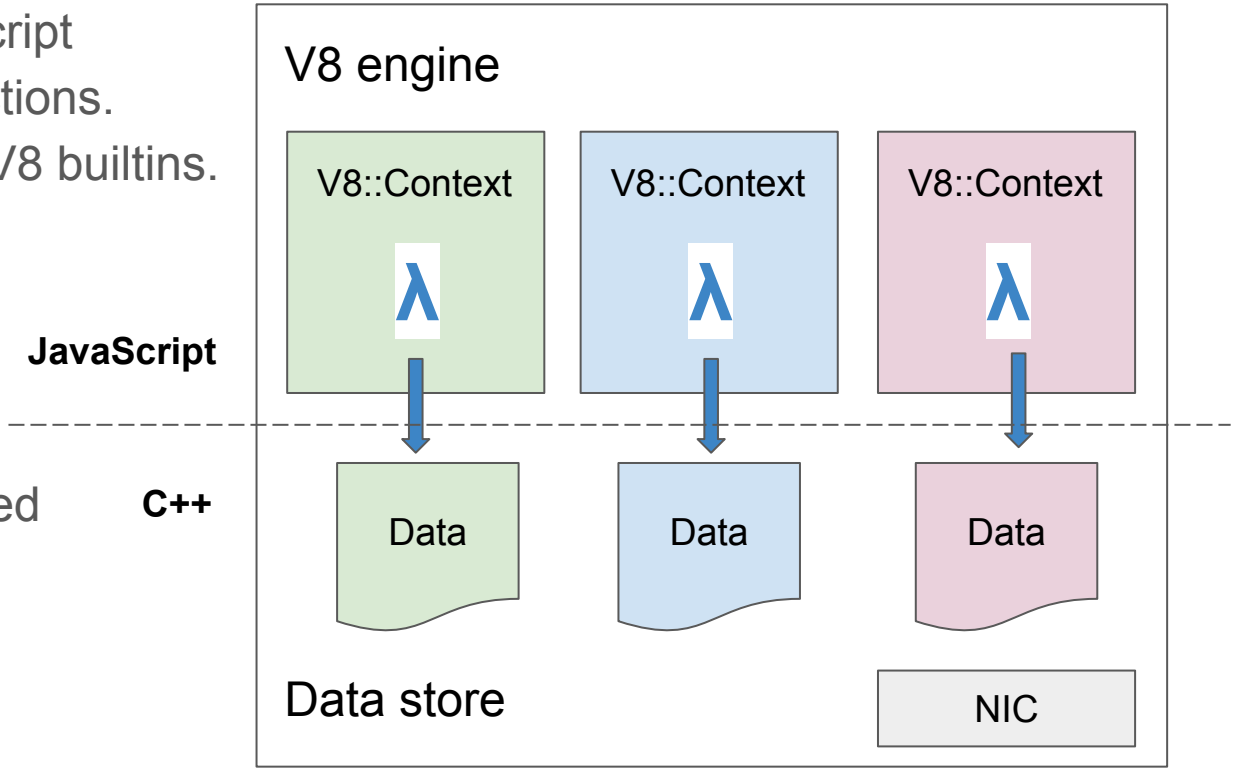
- Flexibility of general programming language.
- Easier to implement customized data structures and logics than SQL.



Shredder design

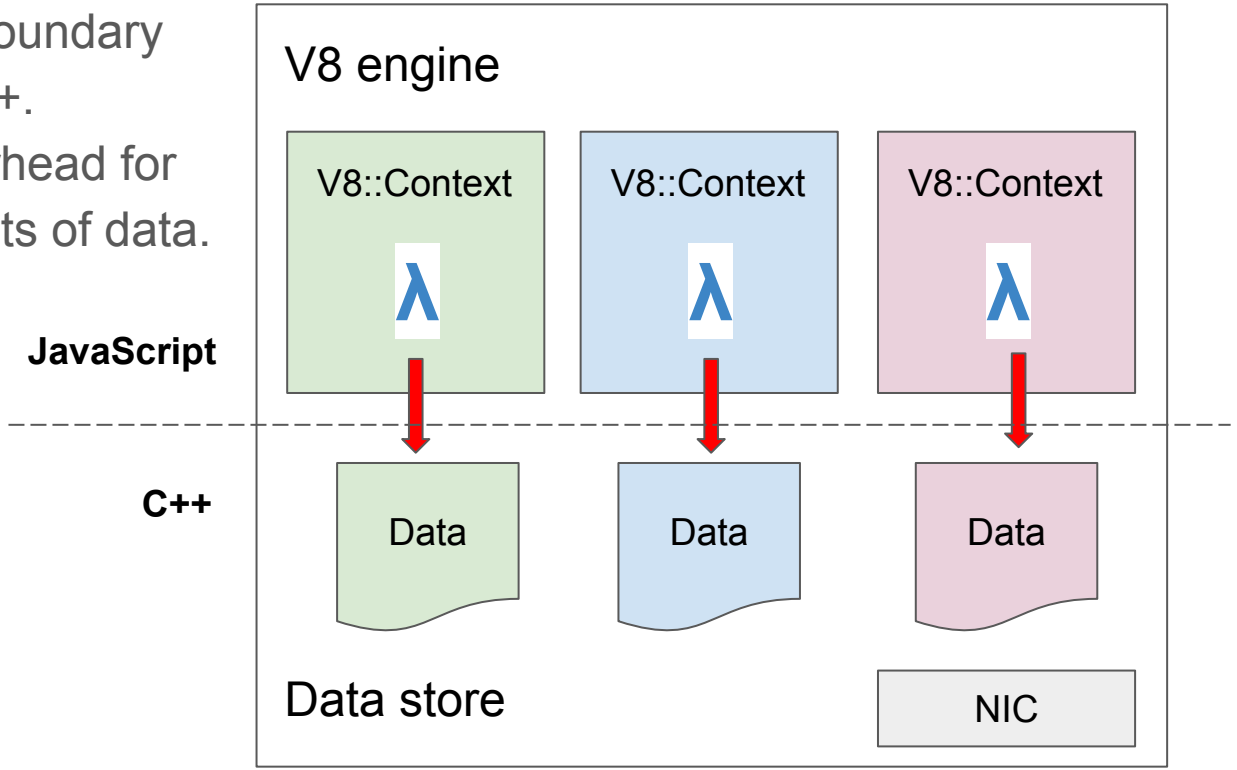
- Embedded **V8** JavaScript runtime to isolate functions.
- Data access through V8 builtins.

- Data store implemented in C++ native code.
- Networking, data management, etc.



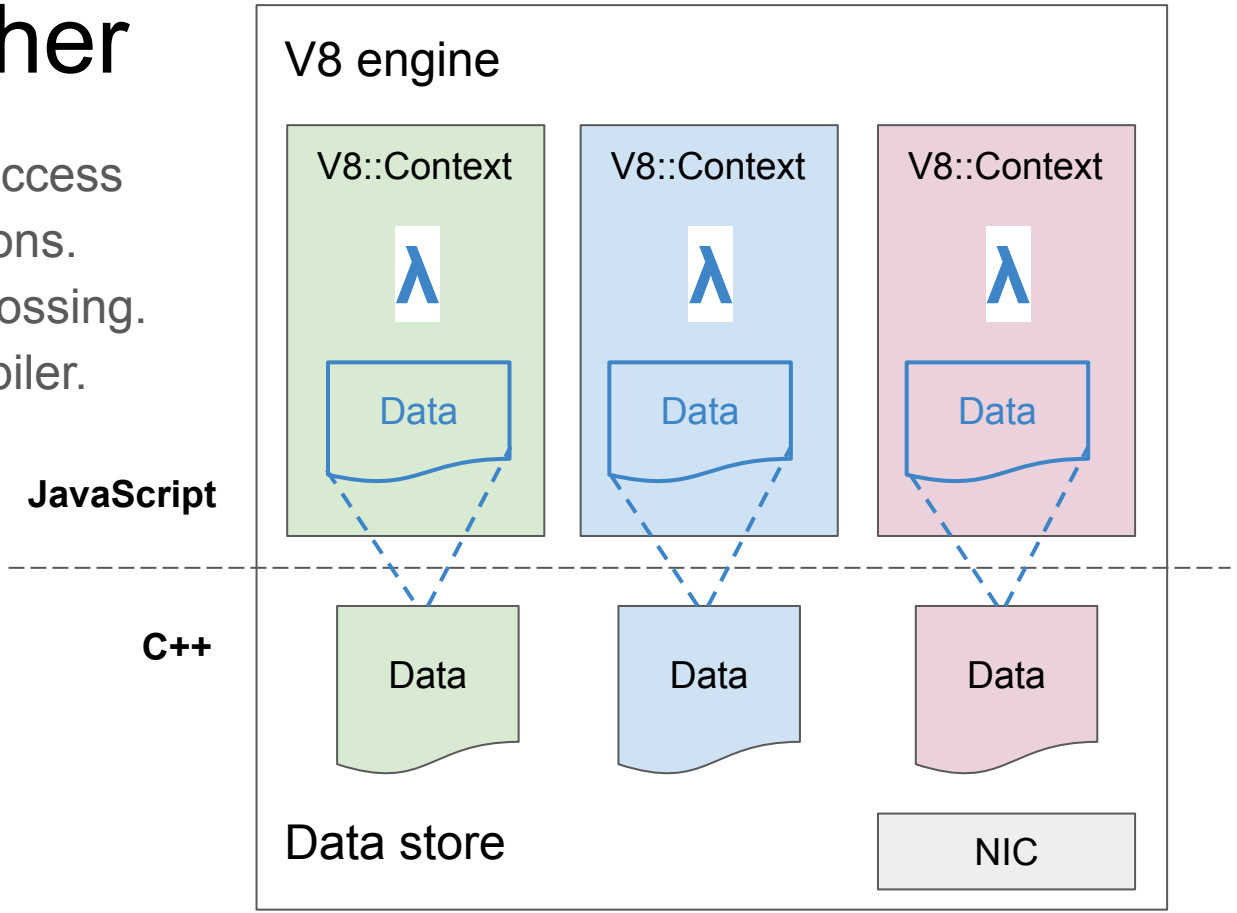
Problem: runtime exit costs add up

- Data access across boundary from JavaScript to C++.
- Add up to a lot of overhead for functions accessing lots of data.



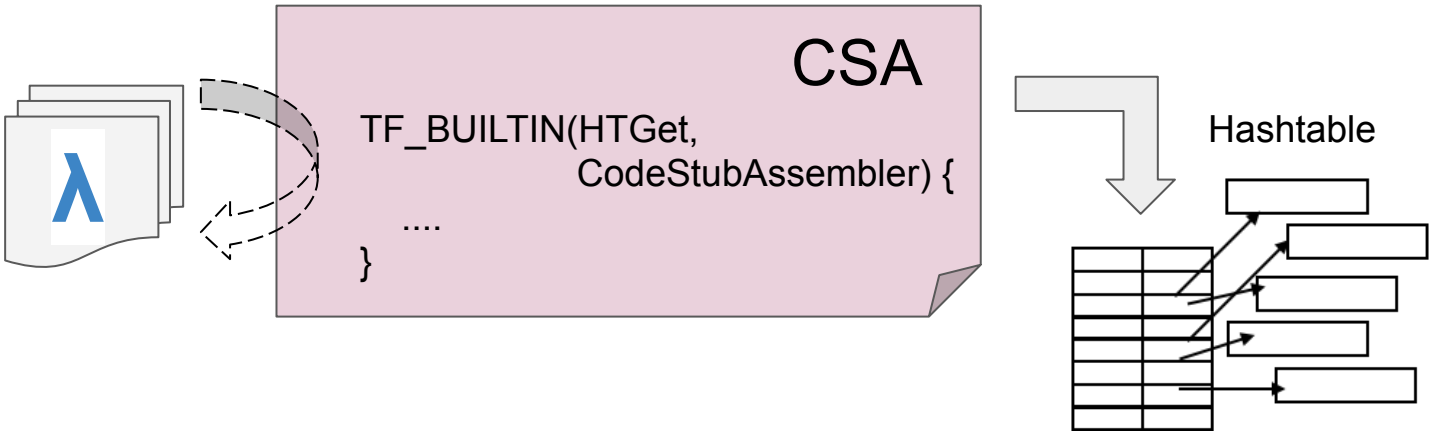
One step further

- Direct and safe data access from serverless functions.
- Eliminate boundary crossing.
- Leverage V8 JIT compiler.



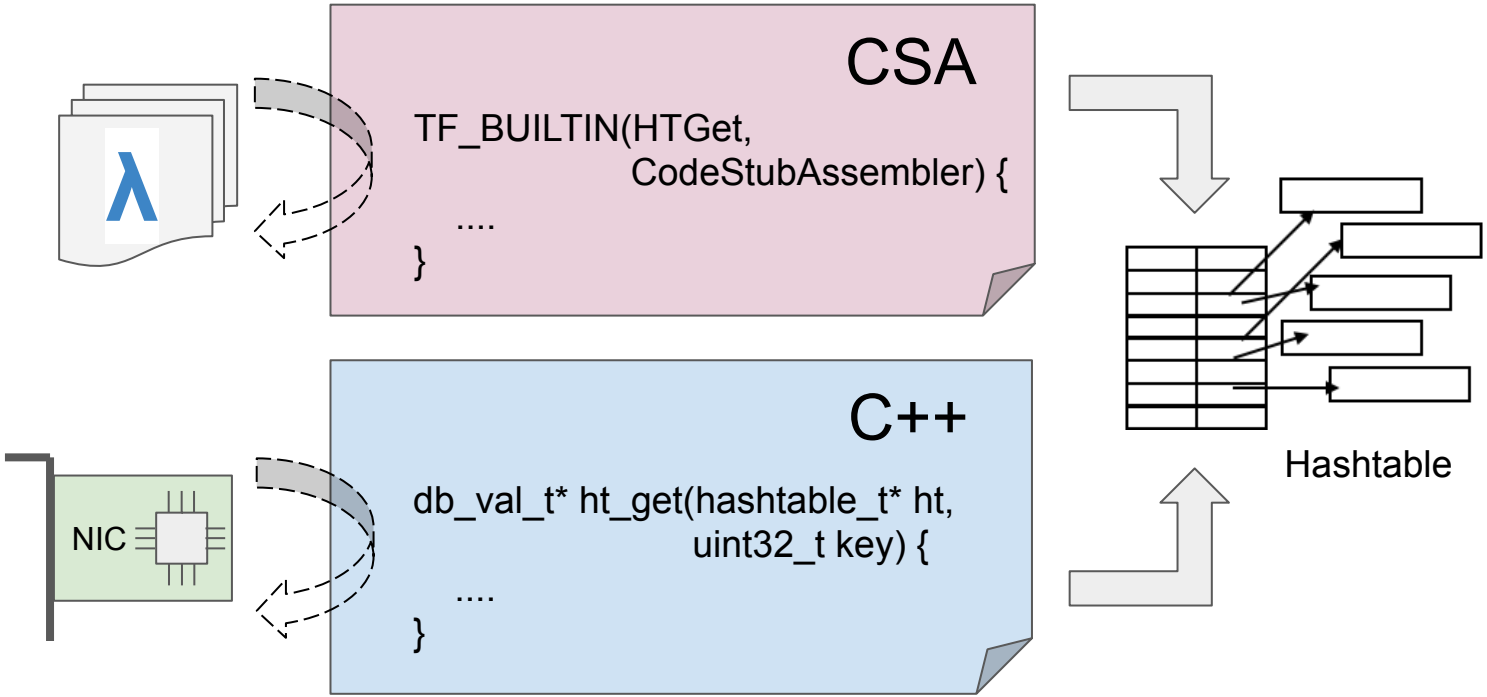
CSA to eliminate boundary crossing

- Implement data access builtin in **CSA** (CodeStubAssembler), the V8 internal IR.
- Eliminating boundary crossing to C++.
- Runtime can inline CSA to improve performance.



Data store and CSA builtin co-design

- CSA builtin and data store implement the same data lookup logic over shared data.



Threat Model

- V8 contexts ensure fault isolation and no cross-tenant data access
  - Data is never shared across tenants
- TCB includes store, networking stack, OS, hardware, and V8 runtime

- Speculative execution attacks complicate secrecy
  - Users could craft speculative gadgets
  - Speculative gadgets could transmit restricted state through cache timing side channel
  - Landscape of attacks still evolving; unclear if runtime/compiler will be able to resolve them

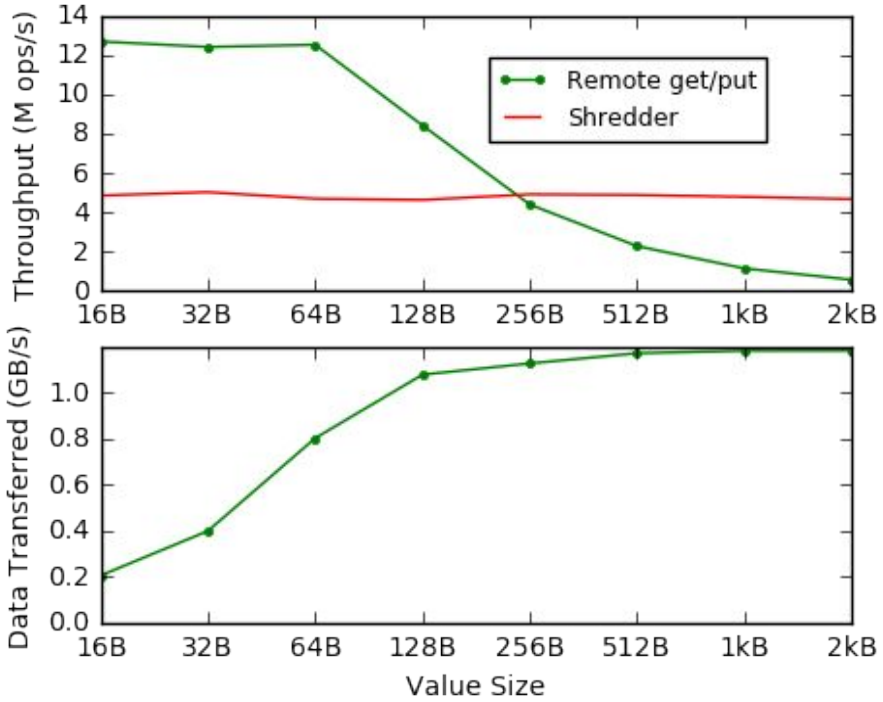
- For now, a shared storage server is only safe with some mutual trust
  - Two-level isolation model possible
  - Process per-tenant; different functions in different runtimes

Evaluations

- 2 x 2.4 GHz Xeon with total 16 physical cores.
- 64 GB memory.
- Intel X710 10GbE.
- DPDK for kernel bypass.

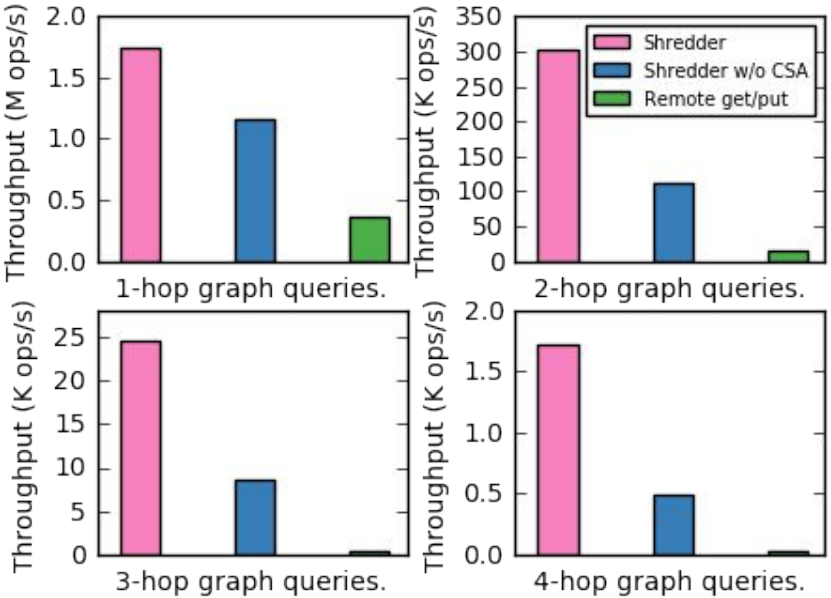
Reduce data movements over network

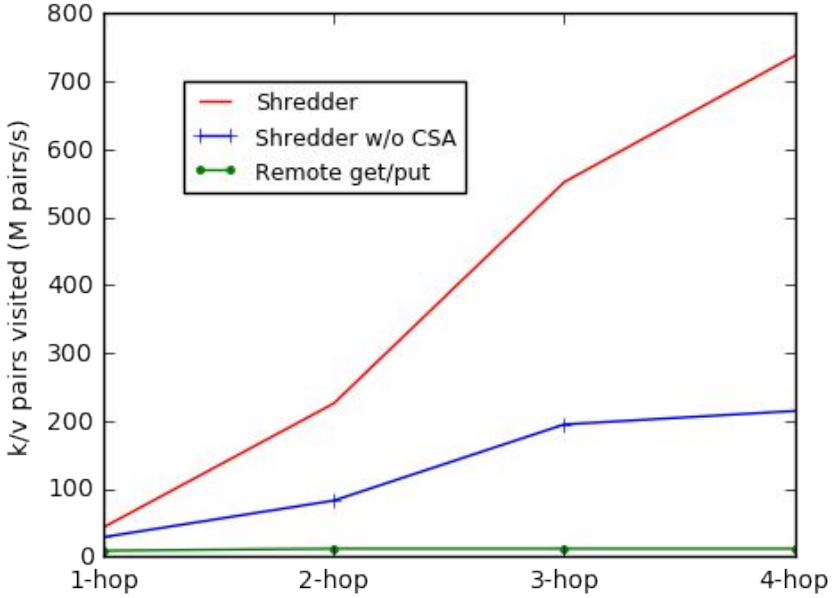
- **Projection**, queries the first 4 bytes of a value.
- Pushing projection to Shredder reduces data movements, compared to baseline which fetches each whole value.



Data intensive functions

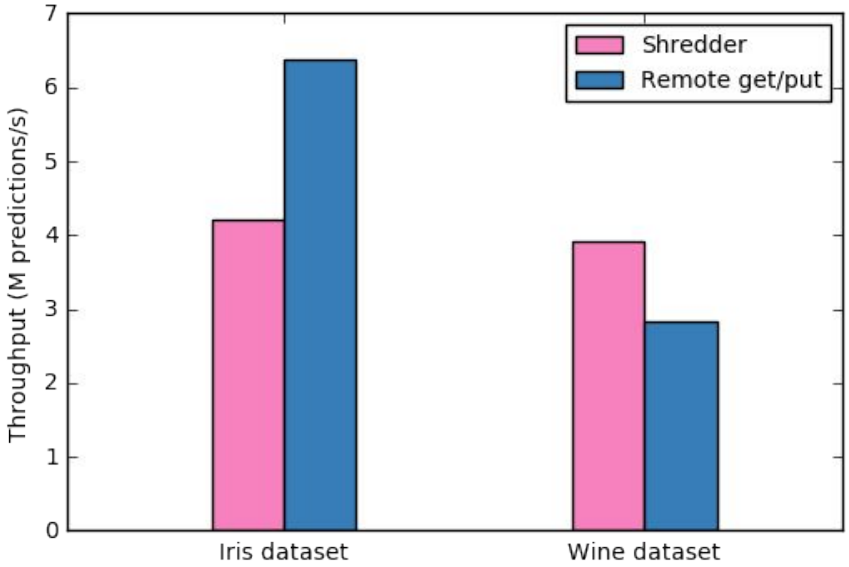
- Traverse Facebook social graph.
- Access **10s of GB** of data per second.
- Shredder **60X** better performance.
- CSA brings **3X** performance gain.





Compute intensive functions

- Neural network inference functions.
- Shredder at disadvantage for compute intensive functions.
- Performance gain still possible if reduces enough data movements to offset inefficiency of JS code.



Related works

- Extensible stores:
  - Comet: An active distributed key-value store. OSDI 2010.
  - Malacology: A Programmable Storage System. EuroSys 17.
  - Splinter: Bare-Metal Extensions for Multi-Tenant Low-Latency Storage. OSDI 18.
- Serverless state store:
  - Pocket: Elastic ephemeral storage for serverless analytics. OSDI 18.

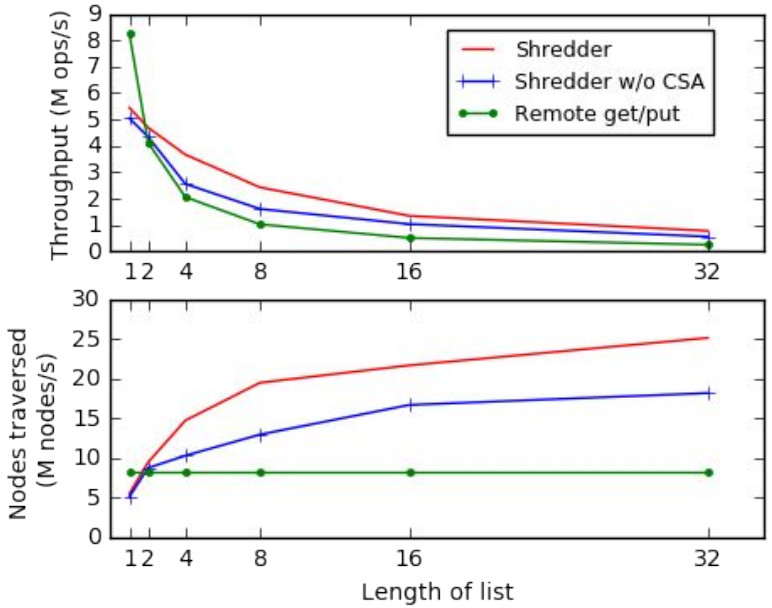
Conclusion

- Gap between functions and persistent states is costly
- Moving functions to storage eliminates some overhead
- Runtimes lower isolations costs, but boundary crossings still add up
- Data-intensive functions benefit from tighter integration of code and data
- Key idea: embed storage access methods within runtime
  - Both storage server and functions can both access data at low cost
- Result: achieves 3X better performance with in-runtime data access.

Thank you!

Backup

Kernel bypass



No kernel bypass