ILLINOIS INSTITUTE OF TECHNOLOGY

Virtualization at the Edge

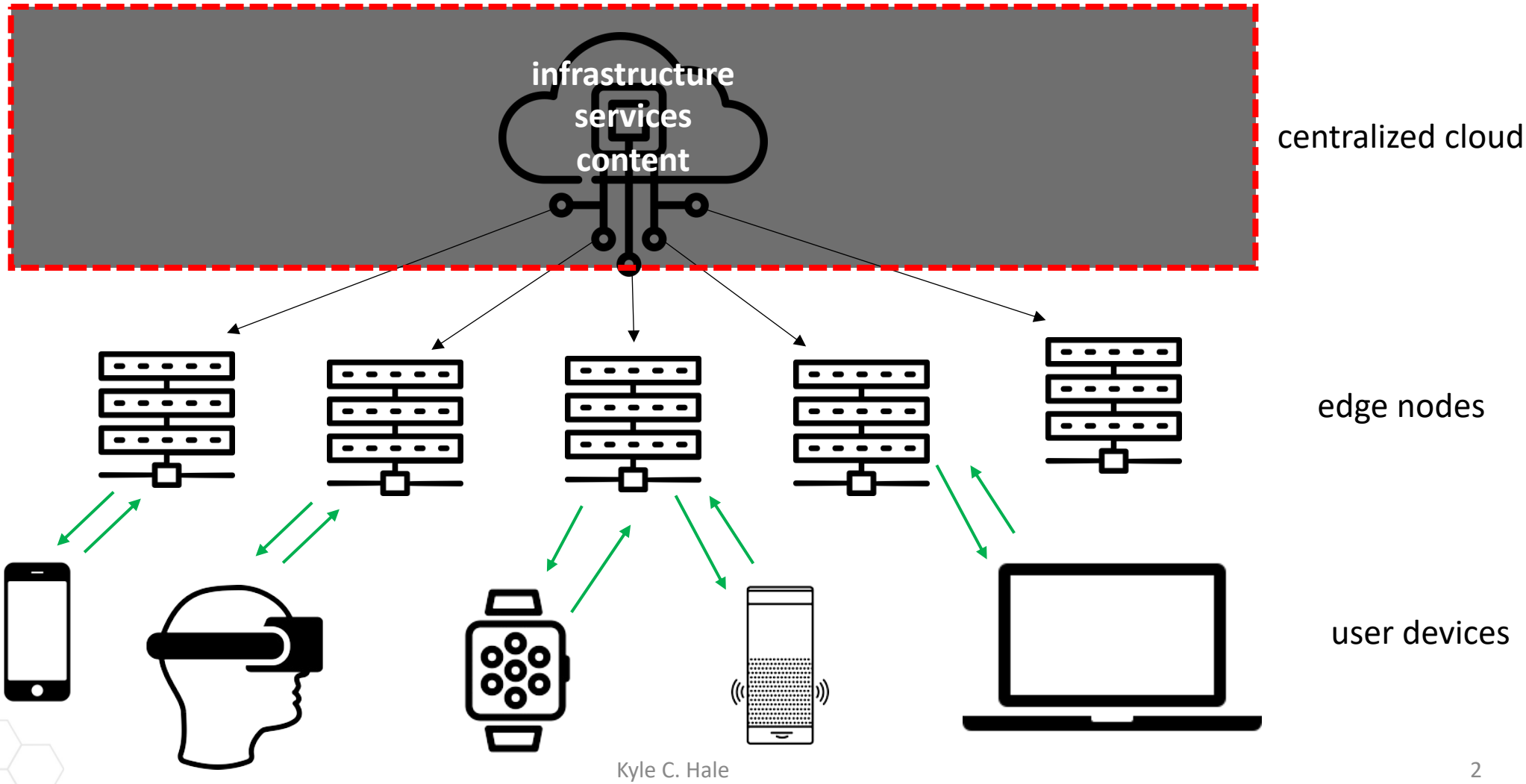
Kyle C. Hale

Laboratory for High-Performance Experimental Systems and Architecture (HExSA)

Argonne/IIT Research Seminar - March 30, 2022

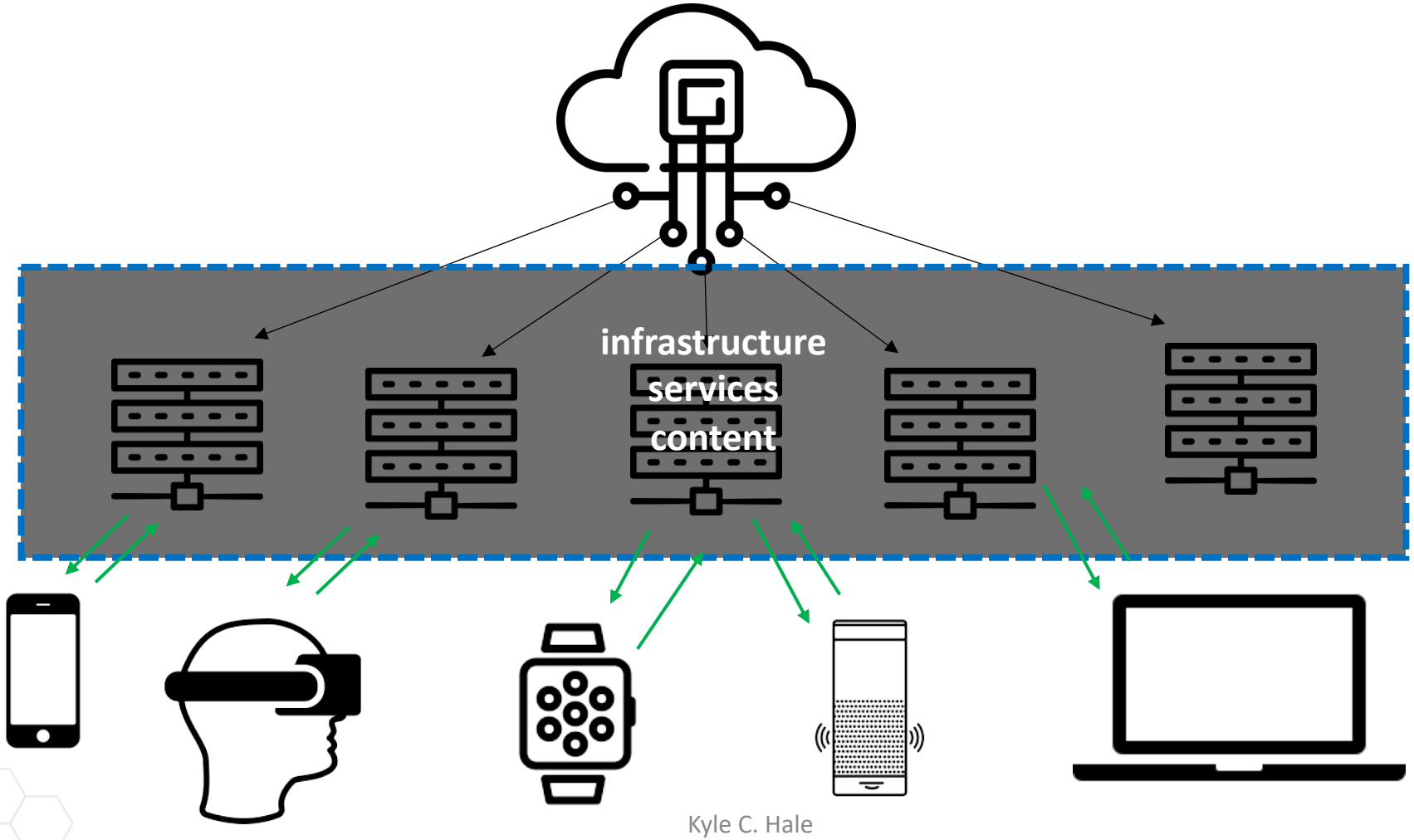
HExSA

The Cloud is Decentralizing: Services and Infrastructure at the Edge



HExSA

The Cloud is Decentralizing: Services and Infrastructure at the Edge



centralized cloud

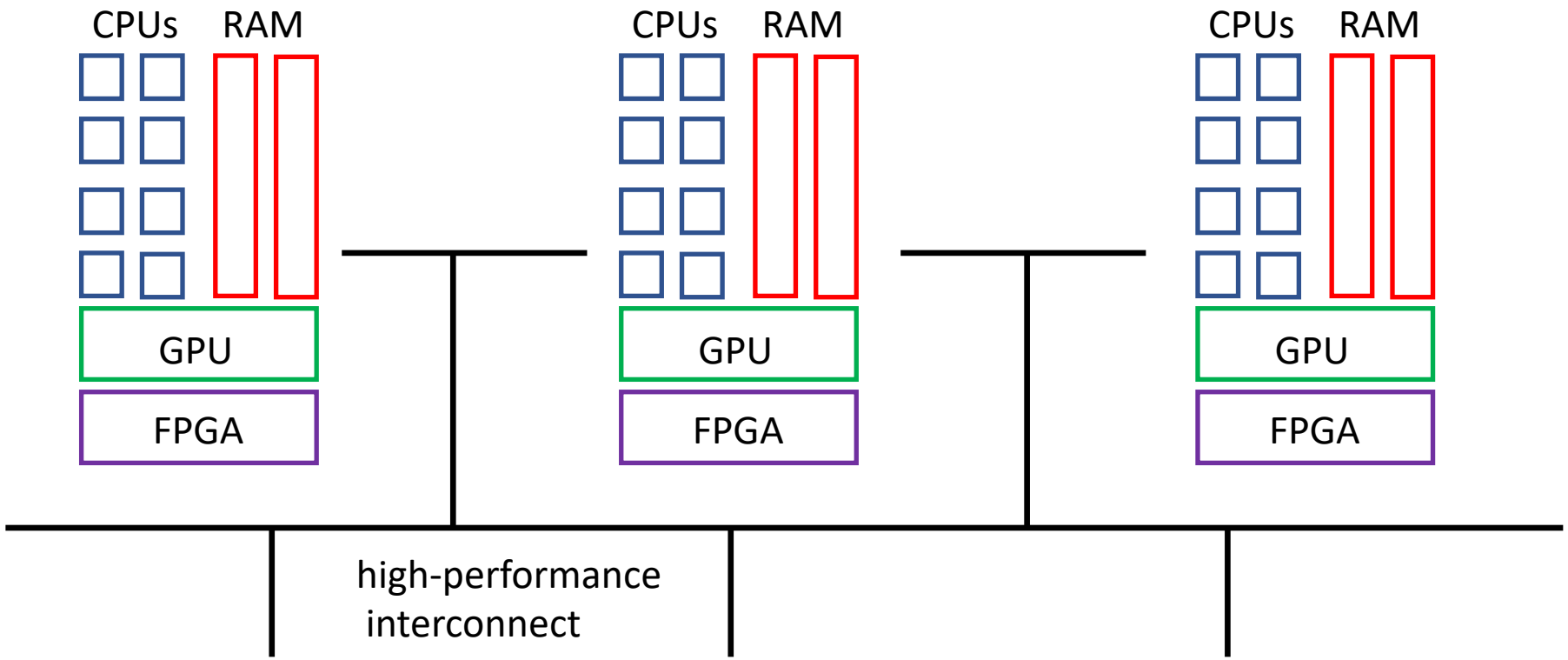
edge nodes

user devices

HExSA

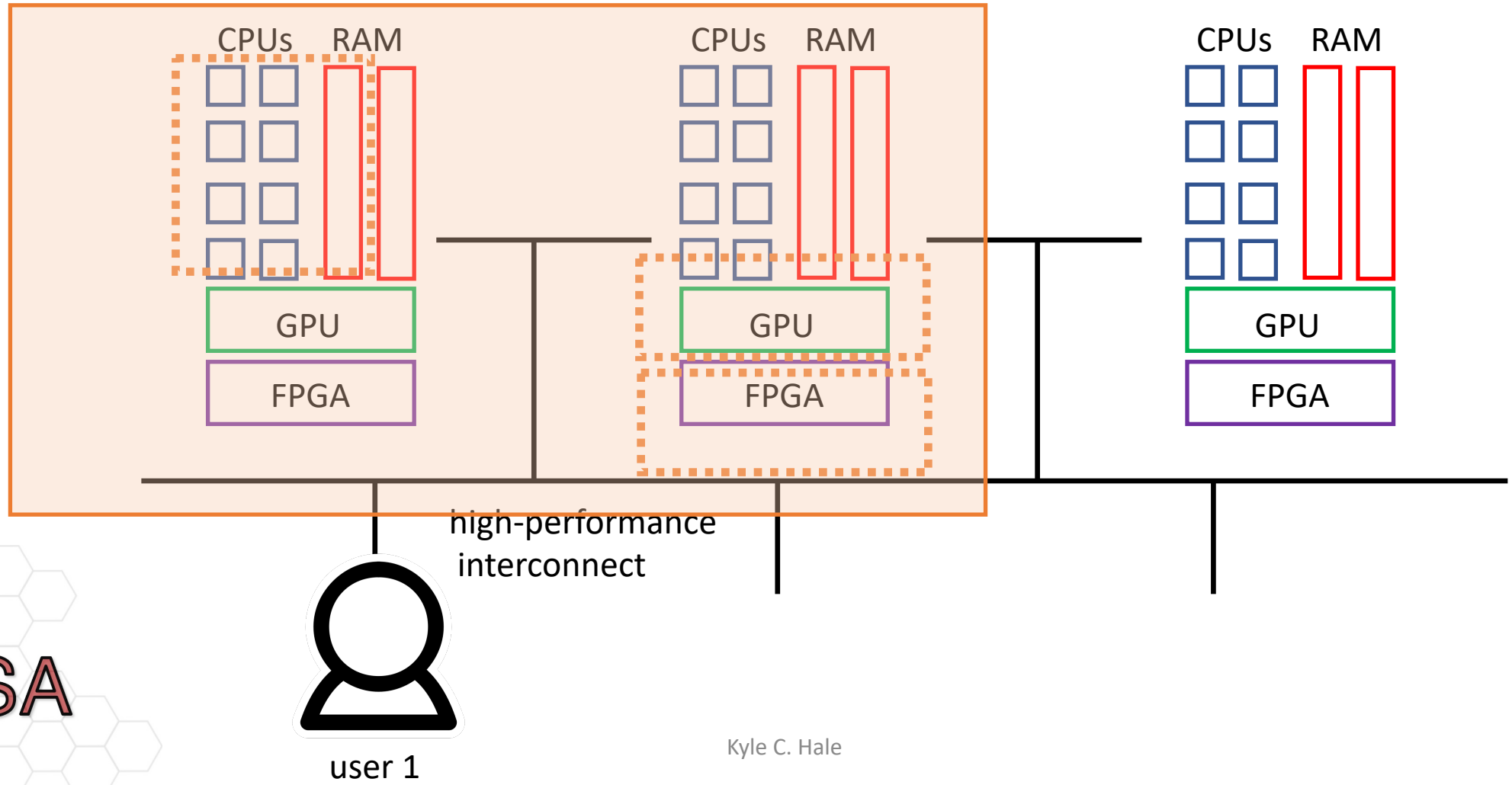
ILLINOIS INSTITUTE OF TECHNOLOGY

Resources are becoming *disaggregated* in the datacenter

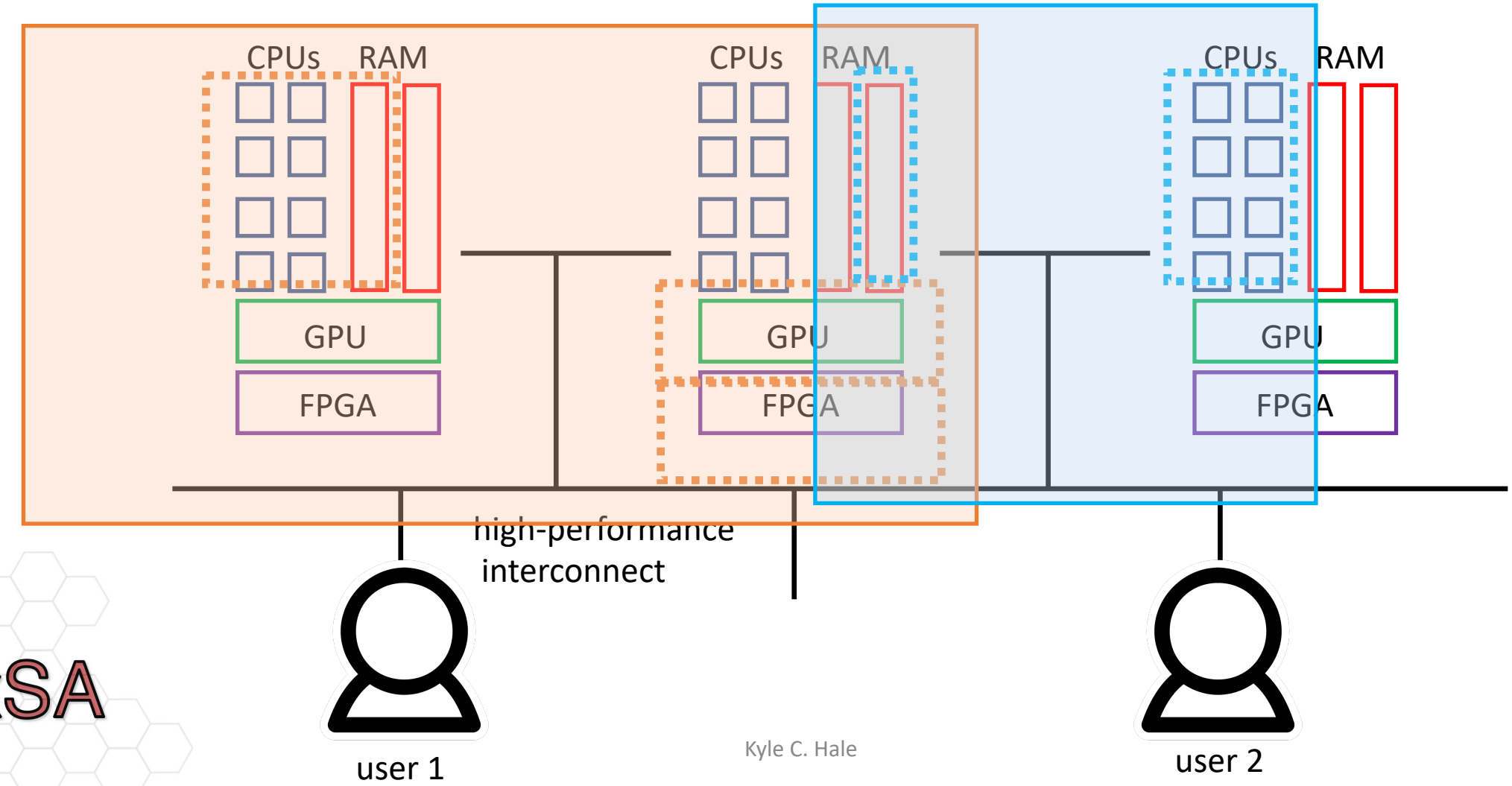


HExSA

Resources are becoming *disaggregated* in the datacenter

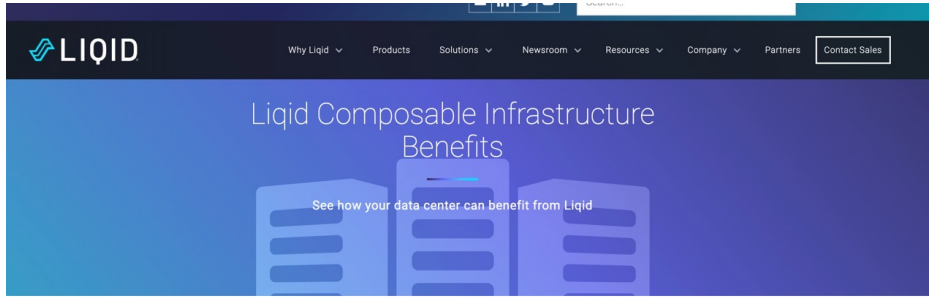


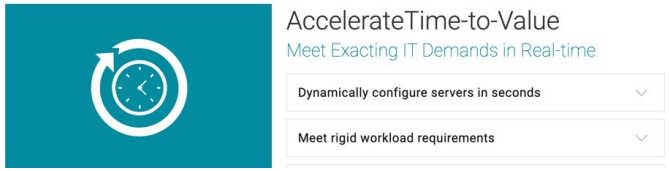
Resources are becoming *disaggregated* in the datacenter



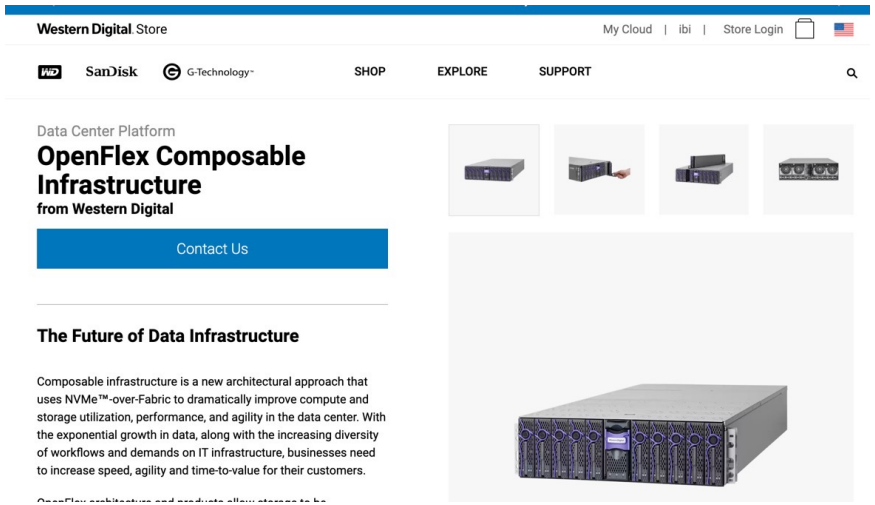
HExSA

Composable Infrastructure

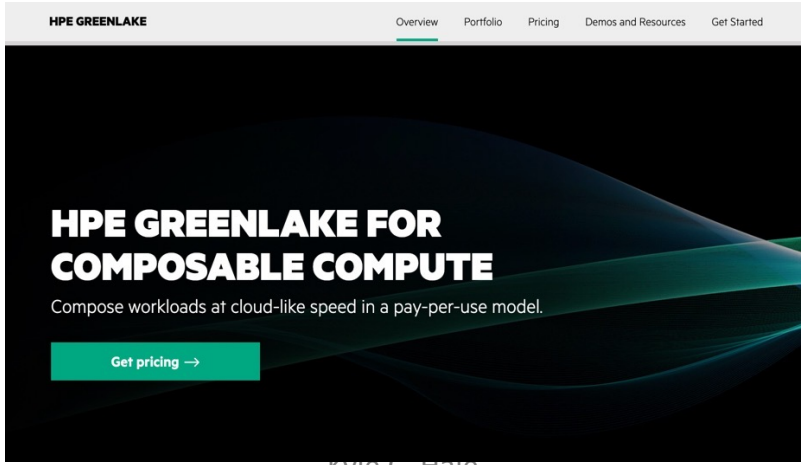




source: https://www.liqid.com/why-liqid/benefits



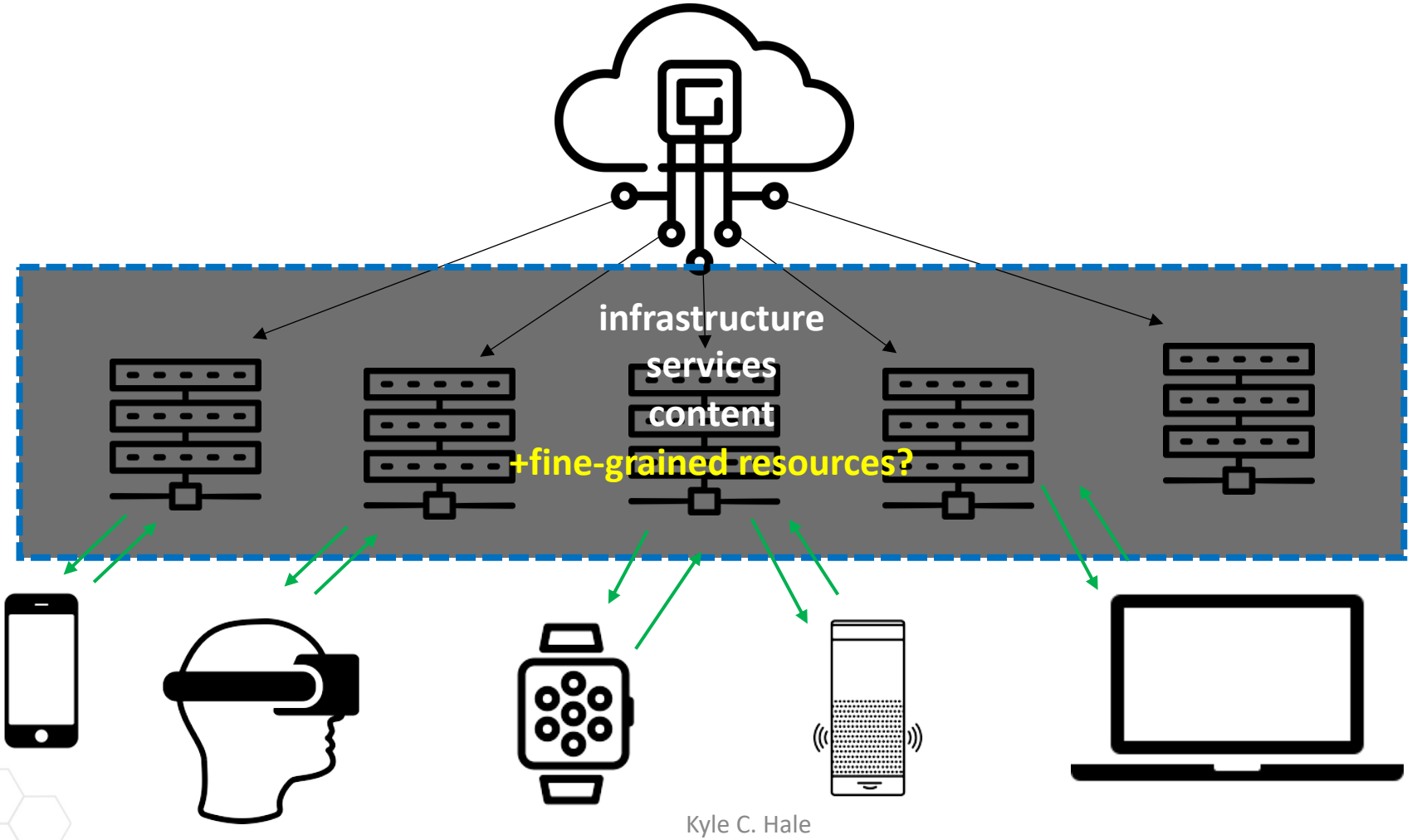
source: https://www.westerndigital.com/products/data-center-platforms/openflex-composable-infrastructure



https://www.hpe.com/us/en/greenlake/composable-compute.html

HExSA

Disaggregation at the Edge



centralized cloud

edge nodes

user devices

HExSA

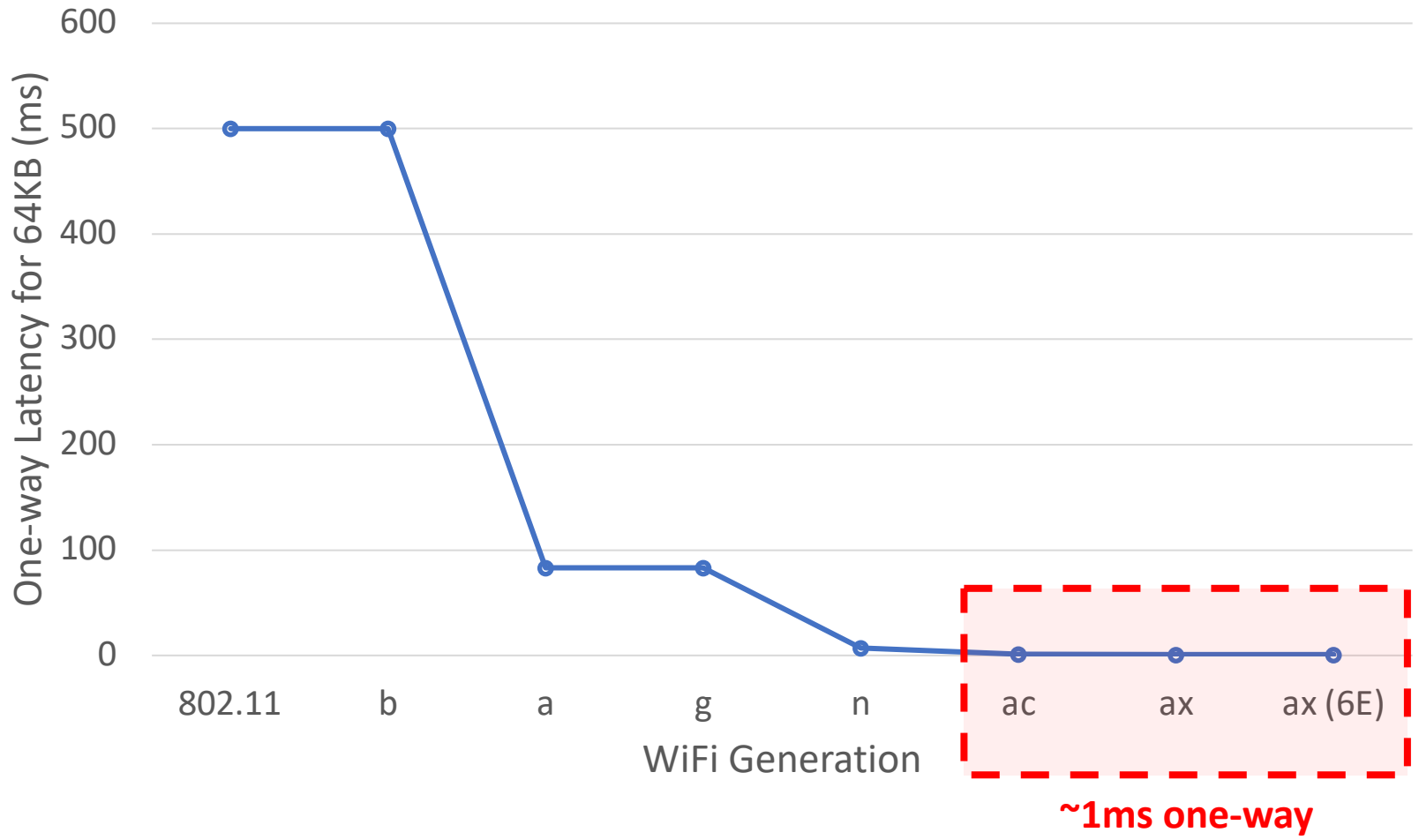
To date, there have been no compelling mass-market applications that require low latencies that cannot be achieved as a web service and that also are too computationally or energy intensive for modern smart phones to run locally. This might be a "chicken or the egg" problem: the lack of cyber foraging infrastructure could potentially be hindering the development of such applications. Later, we discuss one emerging class of applications that could prove to be the compelling application that cyber foraging needs.

R.K. Balan and J. Flinn, "Cyber Foraging Fifteen Years Later," IEEE Pervasive Computing, 16(3), July 2017.

What's Changed?

- Virtualization technology has improved significantly
- Infrastructure provisioning has become more sophisticated (NB serverless research)
- Composable infrastructure
- Hardware design more democratic
- AR/VR/XR is here

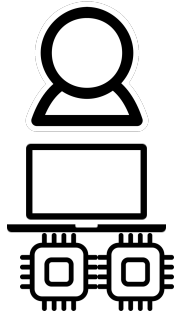
…also, wireless latency continues to drop

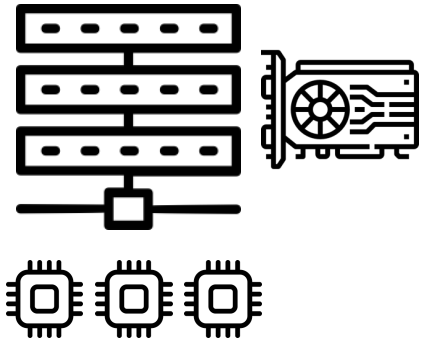


HExSA

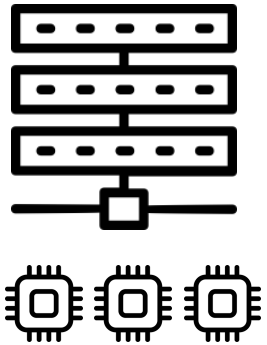
Coalescent Computing

Ephemeral Single-System Image at the Edge





location 1

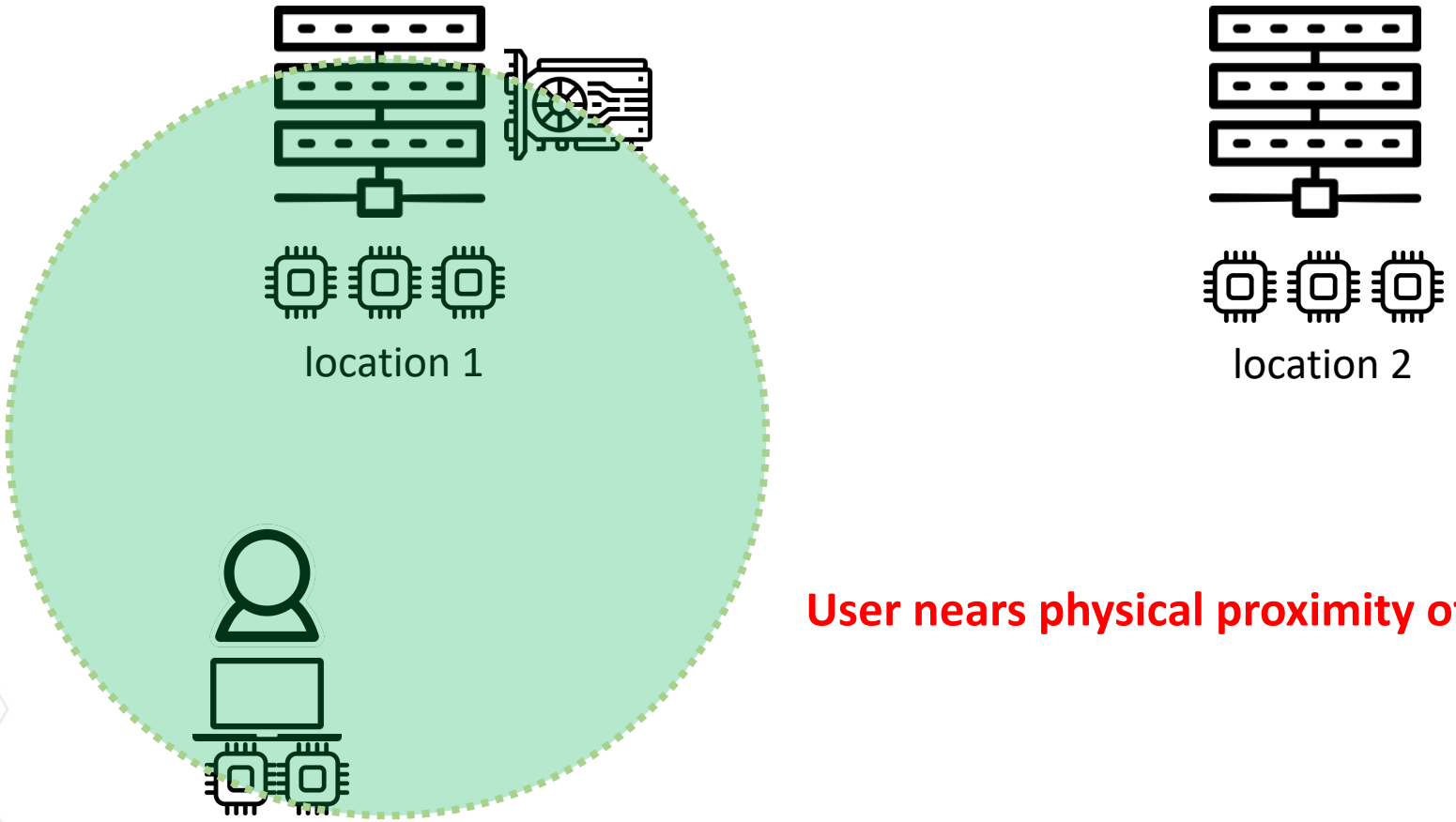


location 2

HExSA

Coalescent Computing

Ephemeral Single-System Image at the Edge

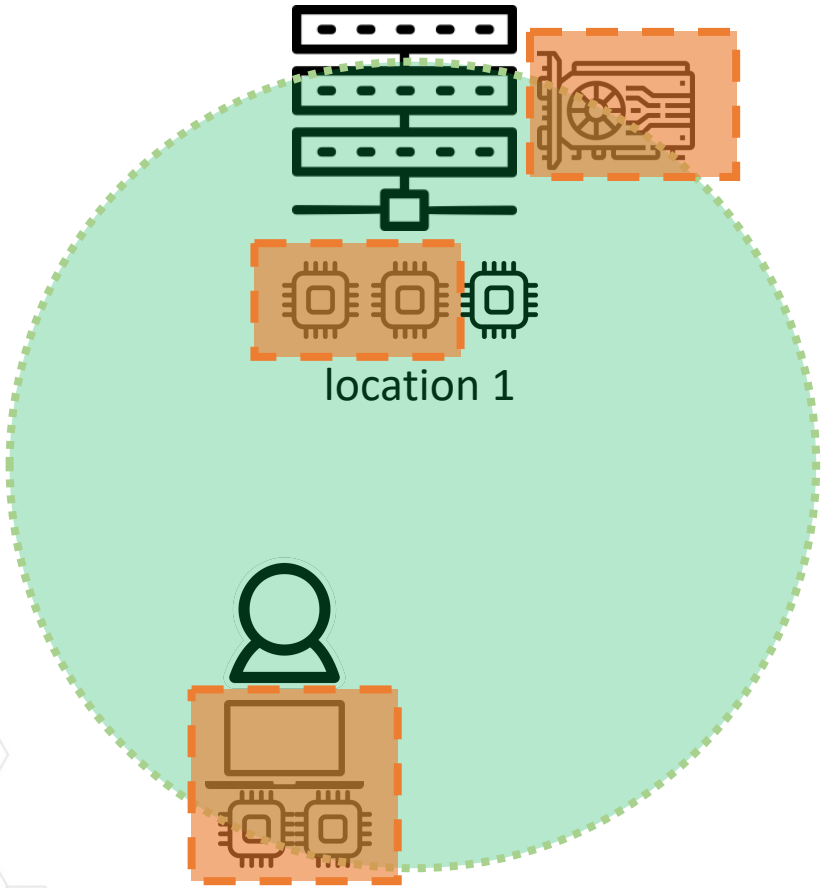


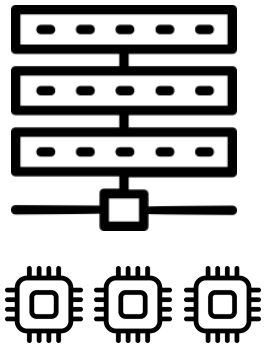
User nears physical proximity of edge system

HExSA

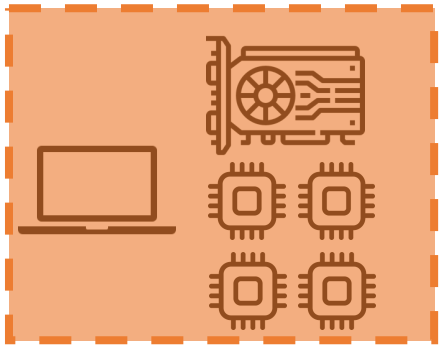
Coalescent Computing

Ephemeral Single-System Image at the Edge





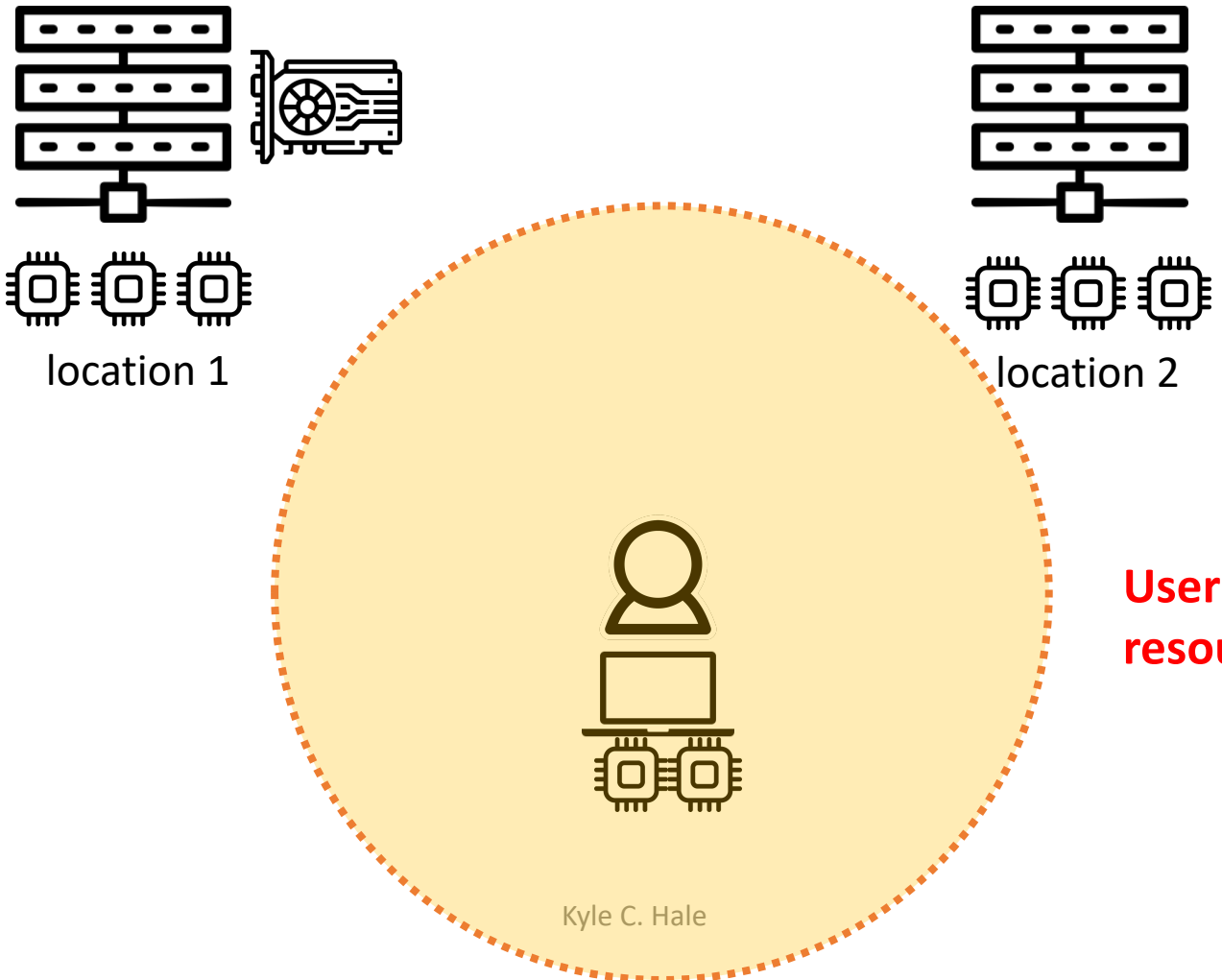
Resources coalesced into one logical system



HExSA

Coalescent Computing

Ephemeral Single-System Image at the Edge

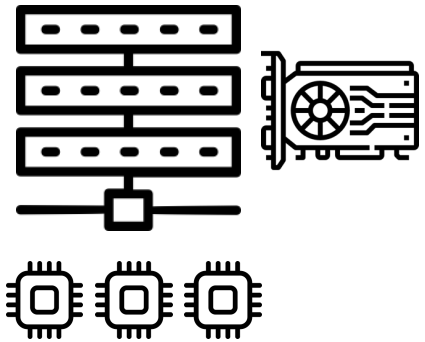


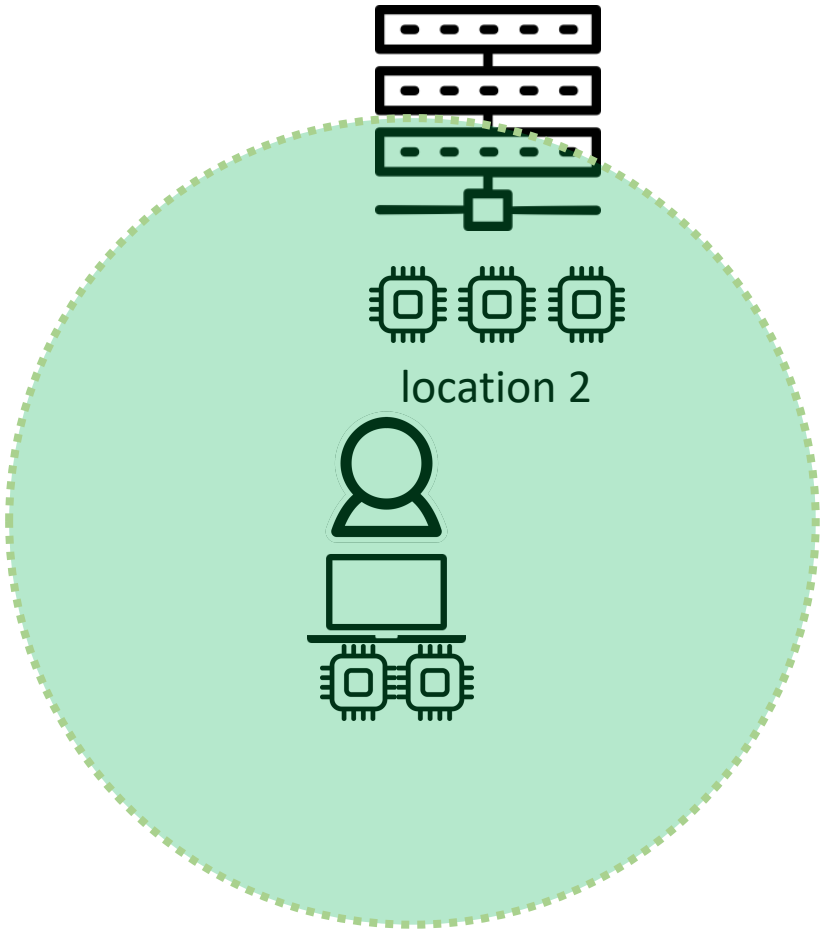
User leaves environment, resources relinquished

HExSA

Coalescent Computing

Ephemeral Single-System Image at the Edge



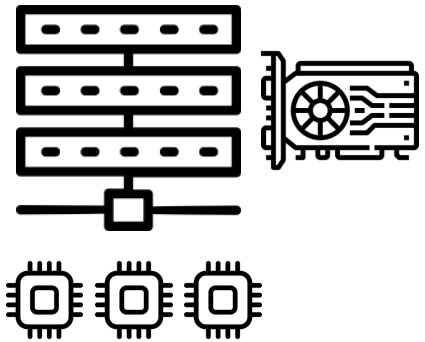


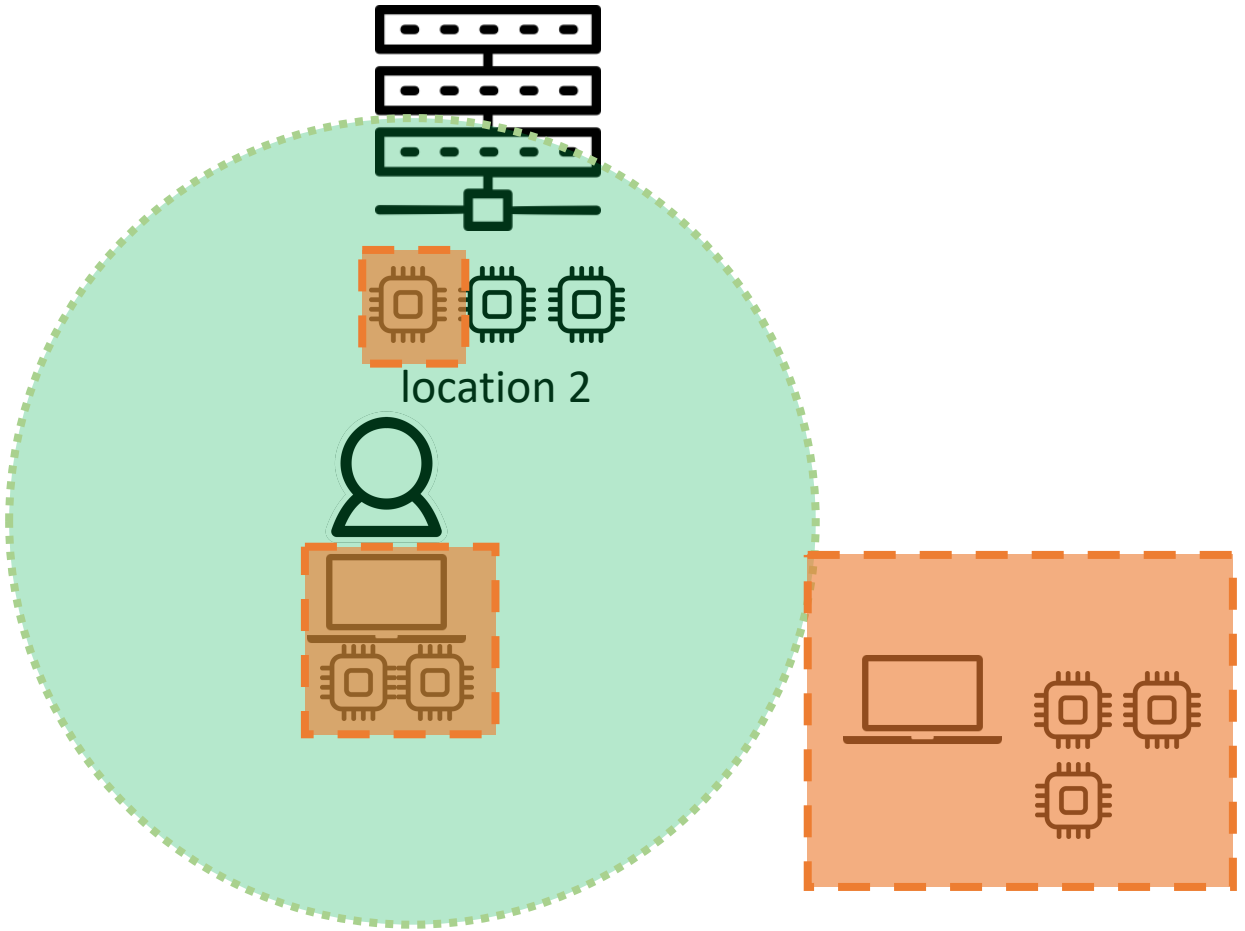
User approaches another edge system

HExSA

Coalescent Computing

Ephemeral Single-System Image at the Edge

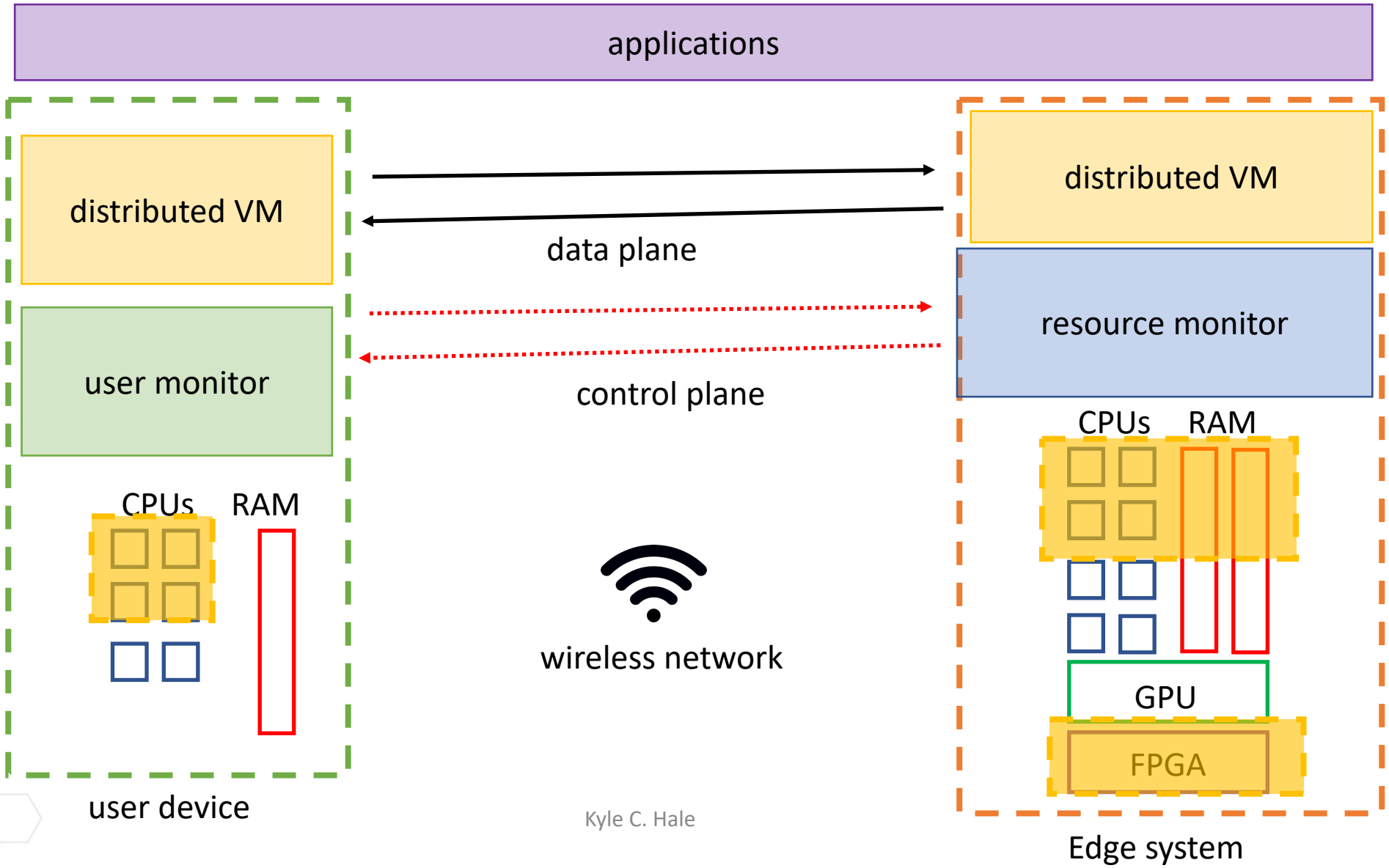




Resources coalesced again, subject to performance, policy constraints

[Hale, Coalescent Computing, APSys '21]

Coalescent System Software



HExSA

Current Work

- Adapting DSM-based approaches to the edge (e.g., GiantVM)
- Building a prototype co-designed hypervisor/OS for CC
- Applying PL techniques for coalescent offloading policies (collab. with Stefan Muller)

Virtines

Isolating Functions at the Hardware Limit (to appear in EuroSys ‘22)

HExSA

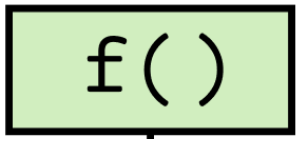
Developer-Friendly, Fast, Function Isolation

- Function isolation
  - Web browsers – sandboxing [CVE-2009-2555, CVE-2009-2935, CVE-2017-2505...]
  - Serverless/FaaS – containers, vms
  - DB *UDF* - high level languages
- Low latency Startup, Short Lived Runtime
  - Spawn and manage many functions w/o significant impact
- Easy programming interface

HExSA

What might function isolation look like?

- Ephemeral state by default (call stack)
- `return` destroys the context



Macro Goals

Function Granularity

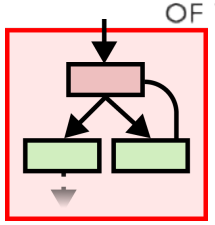
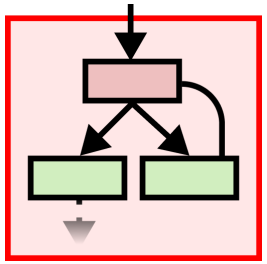
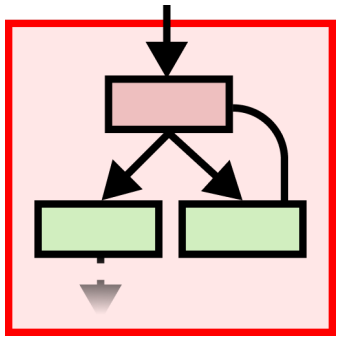
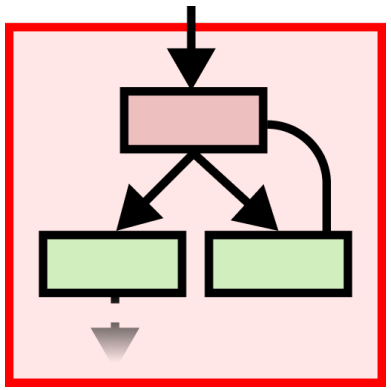
Low Latency Startup

Easy to integrate

HExSA

Virtines: Virtual Subroutines

- **Hardware-virtualized** isolated functions
- ***Microsecond*** level boot times
- Paravirtualization
- General purpose

HExSA

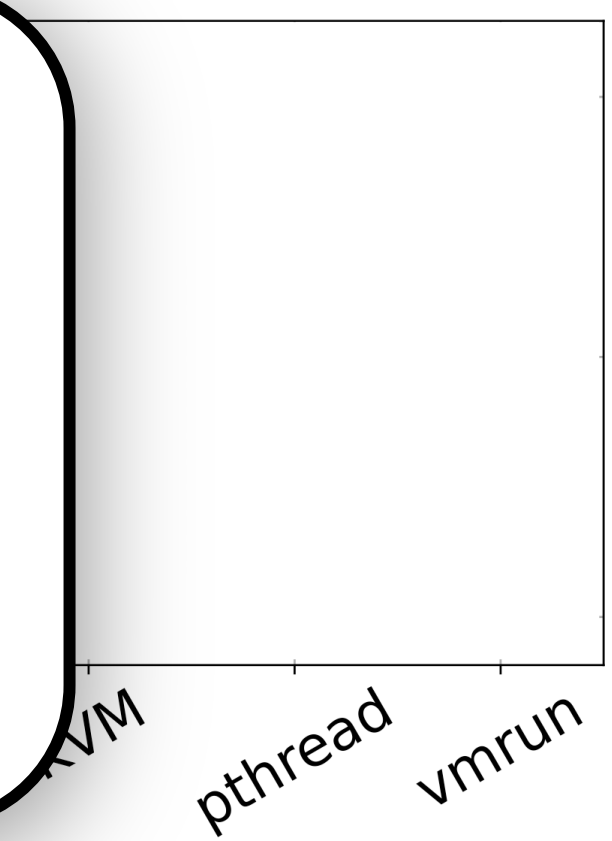
The *lower bounds* of virtualization

It's not the hardware that's expensive…

- What is the laten
  VM creation?
  - HW/SW state

SW VM Allocation is ***expensive***

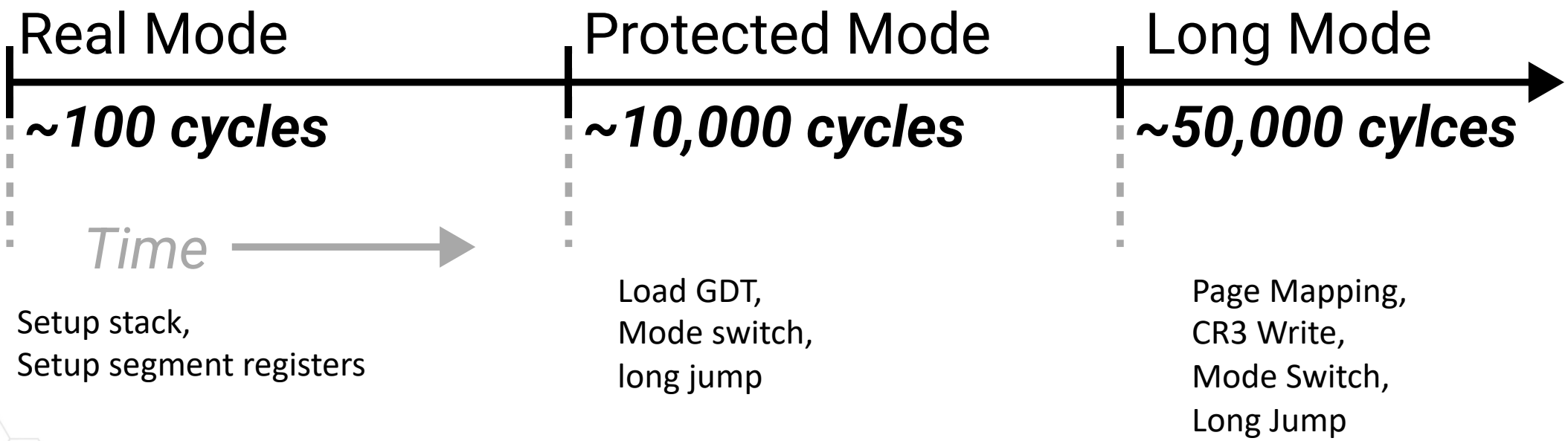
VM Interaction is ***cheap***



(on AMD EPYC 7281)

HExSA

Bootstrapping woes

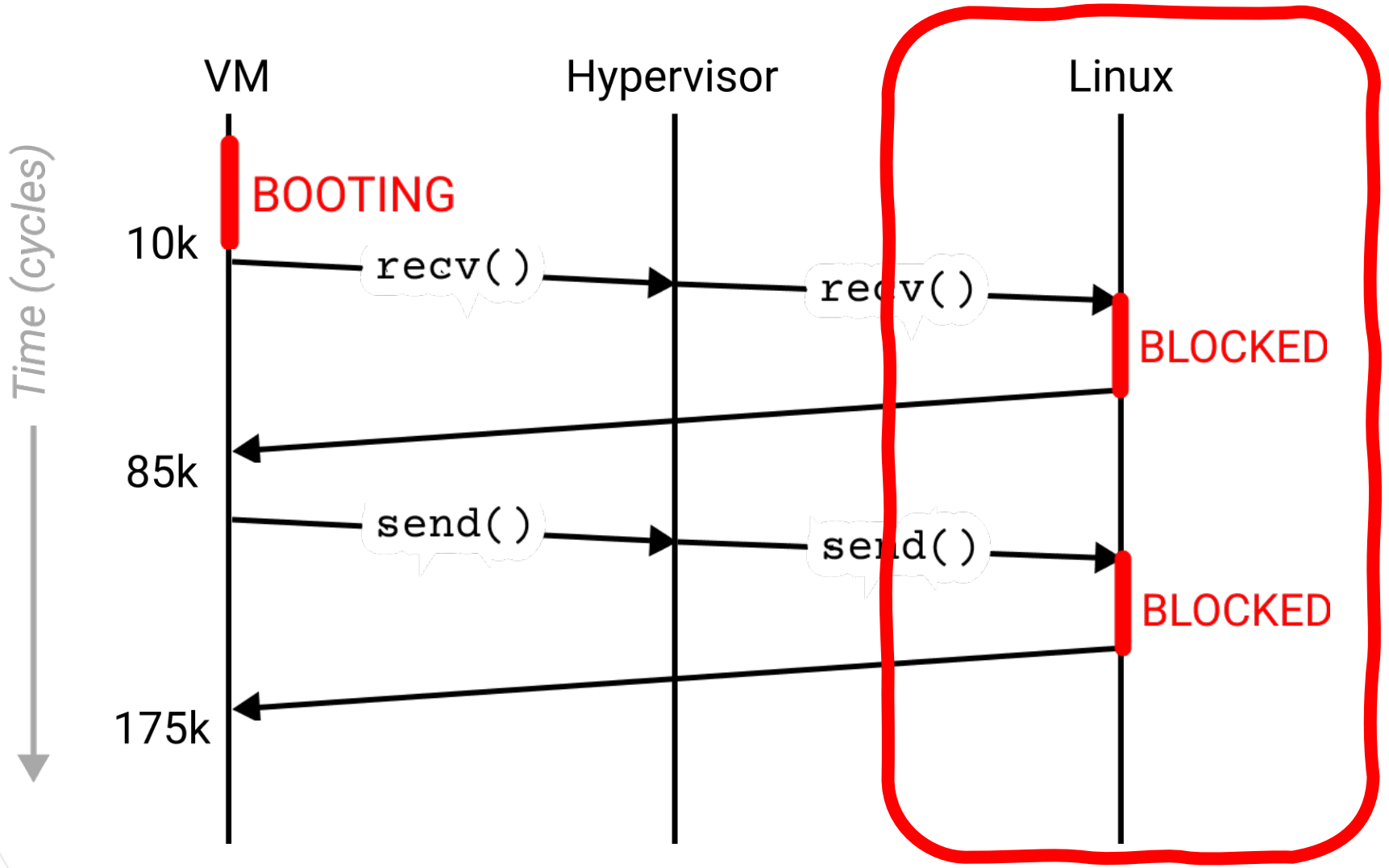


HExSA

Traditional I/O is *very* expensive

- Try to make a VM feel like real hardware
  - Requires large device drivers
  - Lots of VM Exits for single ops (Expensive!)
- Paravirtualization
  - Codesigned, VM aware of the Hypervisor
  - I/O via ***hypercalls***

HTTP server using hypercalls



HExSA

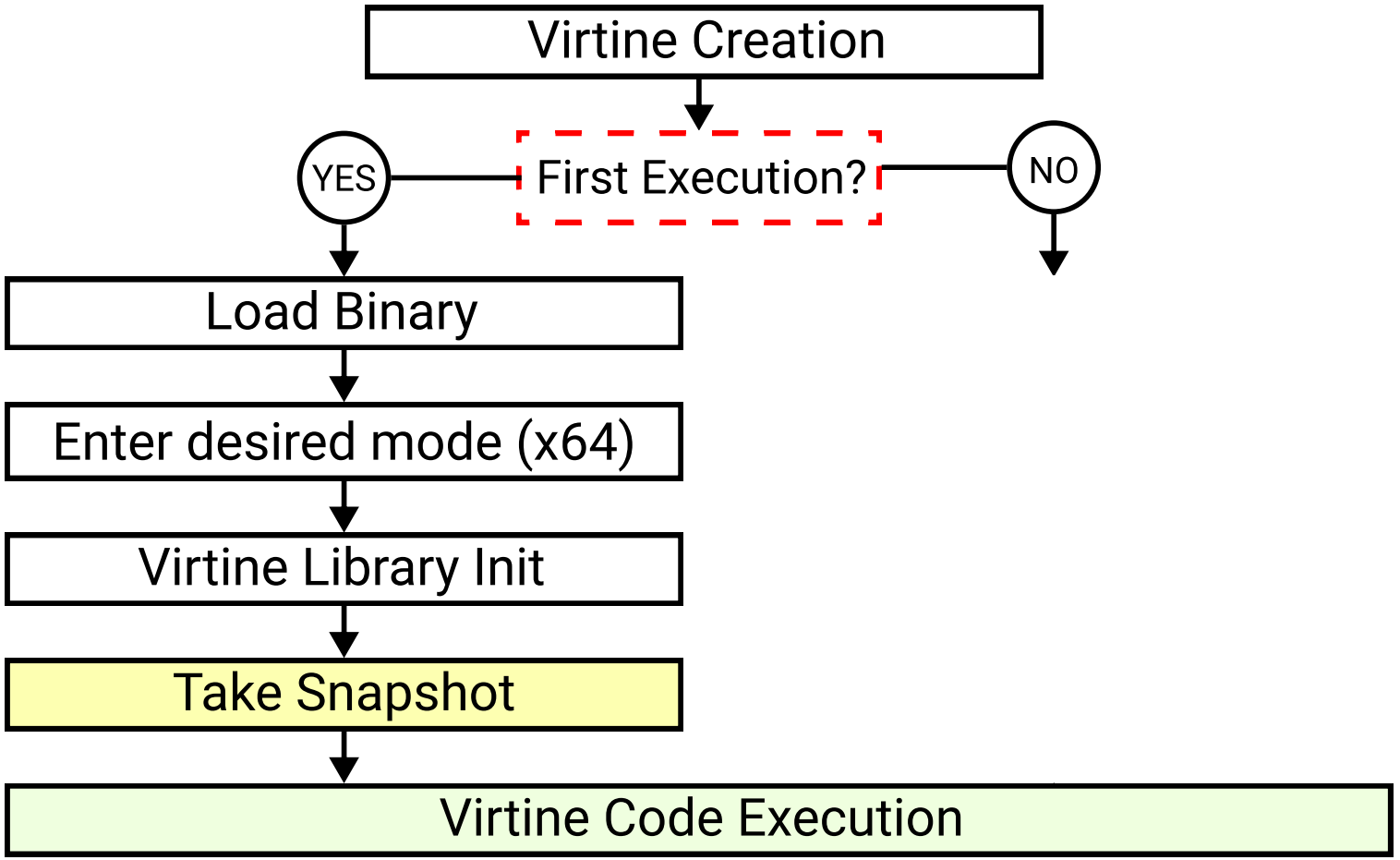
Wasp: an implementation of Virtines

Wasp

- A micro-hypervisor library
- Abstracts hardware specific interfaces
- Lean
- ***Heavily optimized***

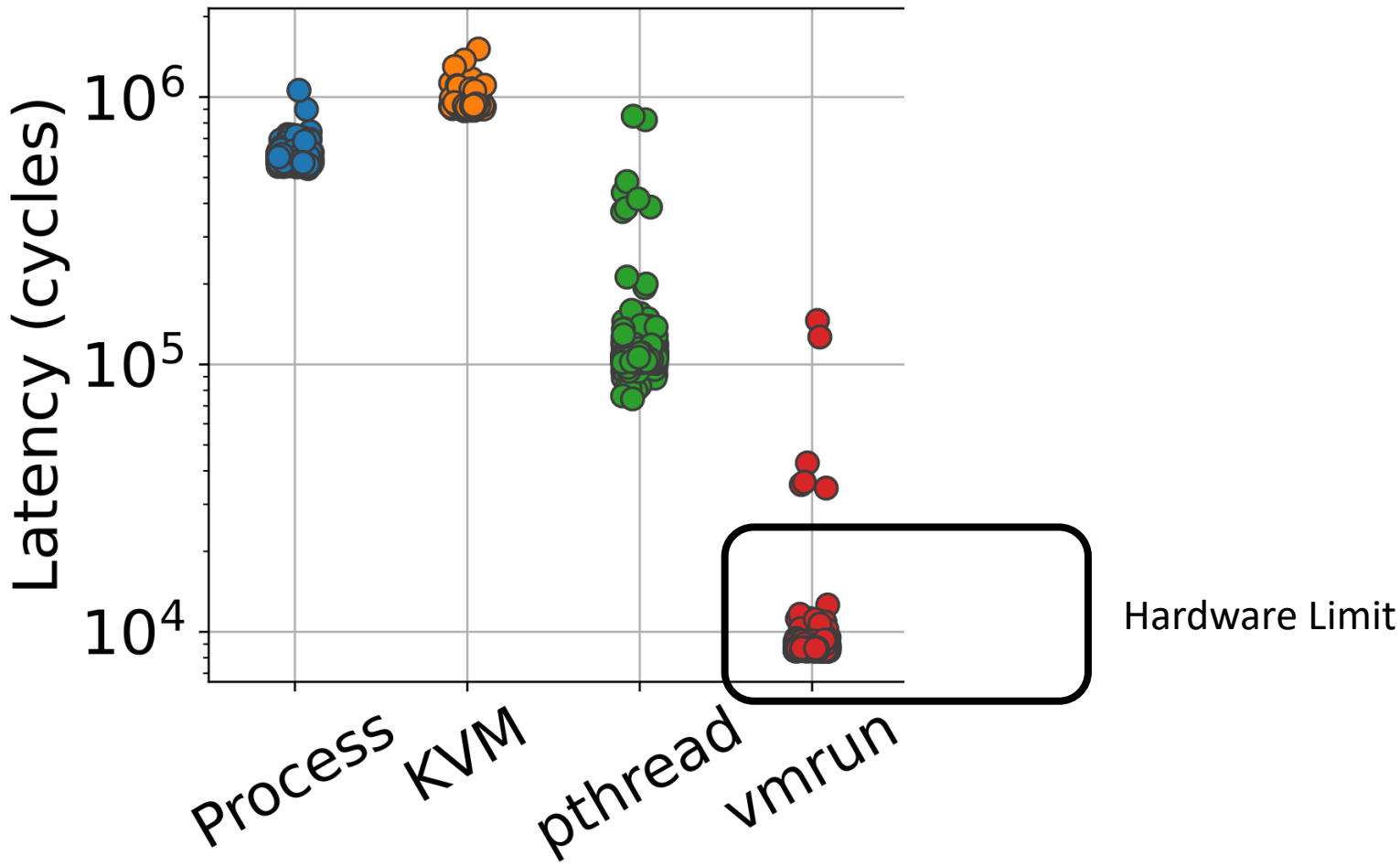
HExSA

Allocating virtines with Wasp



HExSA

Wasp is close to the hardware limit



HExSA

C Language Extension

```
void db_run_udf(db_udf_t *code) {
  // ...
}
```

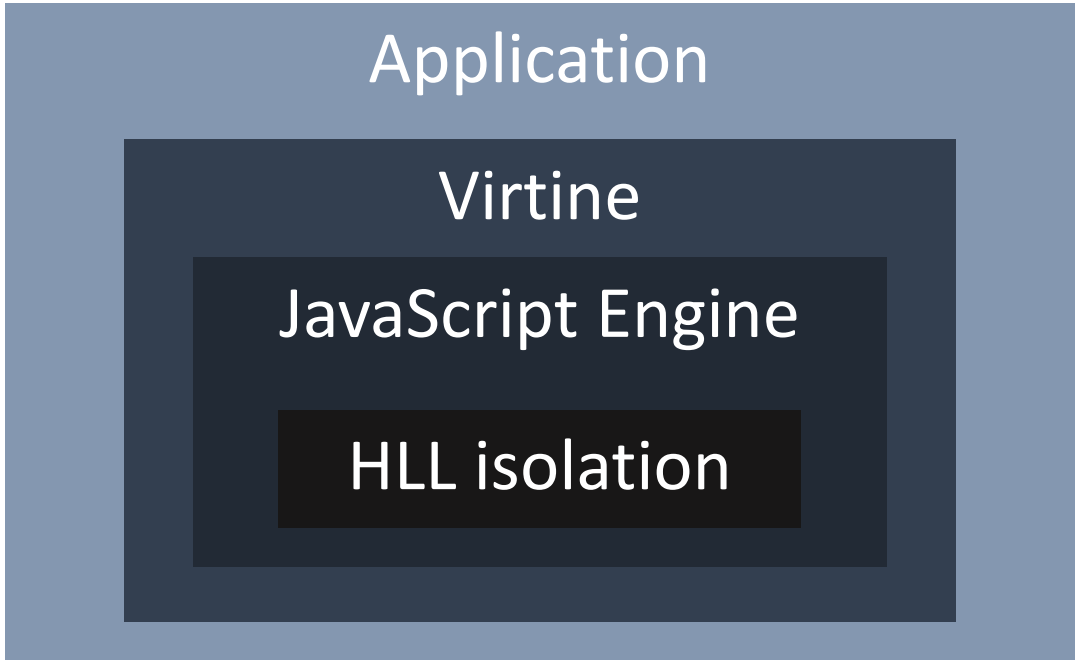
```
virtine void db_run_udf(db_udf_t *code) {
  // ...
}
```

- Default-deny access to host services
- Custom LLVM module pass to compile and manage virtines

HExSA

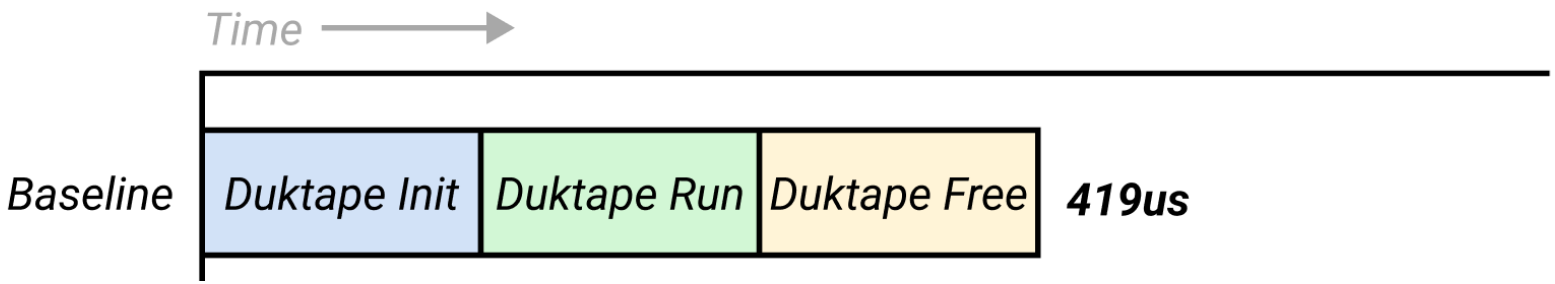
Duktape JavaScript Engine

- **Duktape** JavaScript Engine: https://duktape.org/
  - Embeddable, Portable



HExSA

Do our latency optimizations work?



Effectiveness of language extensions

```
void handl                                    ) {
  // ...
}


virtine v                                    (int fd) {
  // ...
}
```

Baseline: **50us,** σ≈10us

Virtines: **80us**, σ≈30us

How easy is it to integrate?

- ~20 lines of code changed
  - Mostly glue logic
- Significantly slower... But we expect that
- 21kb

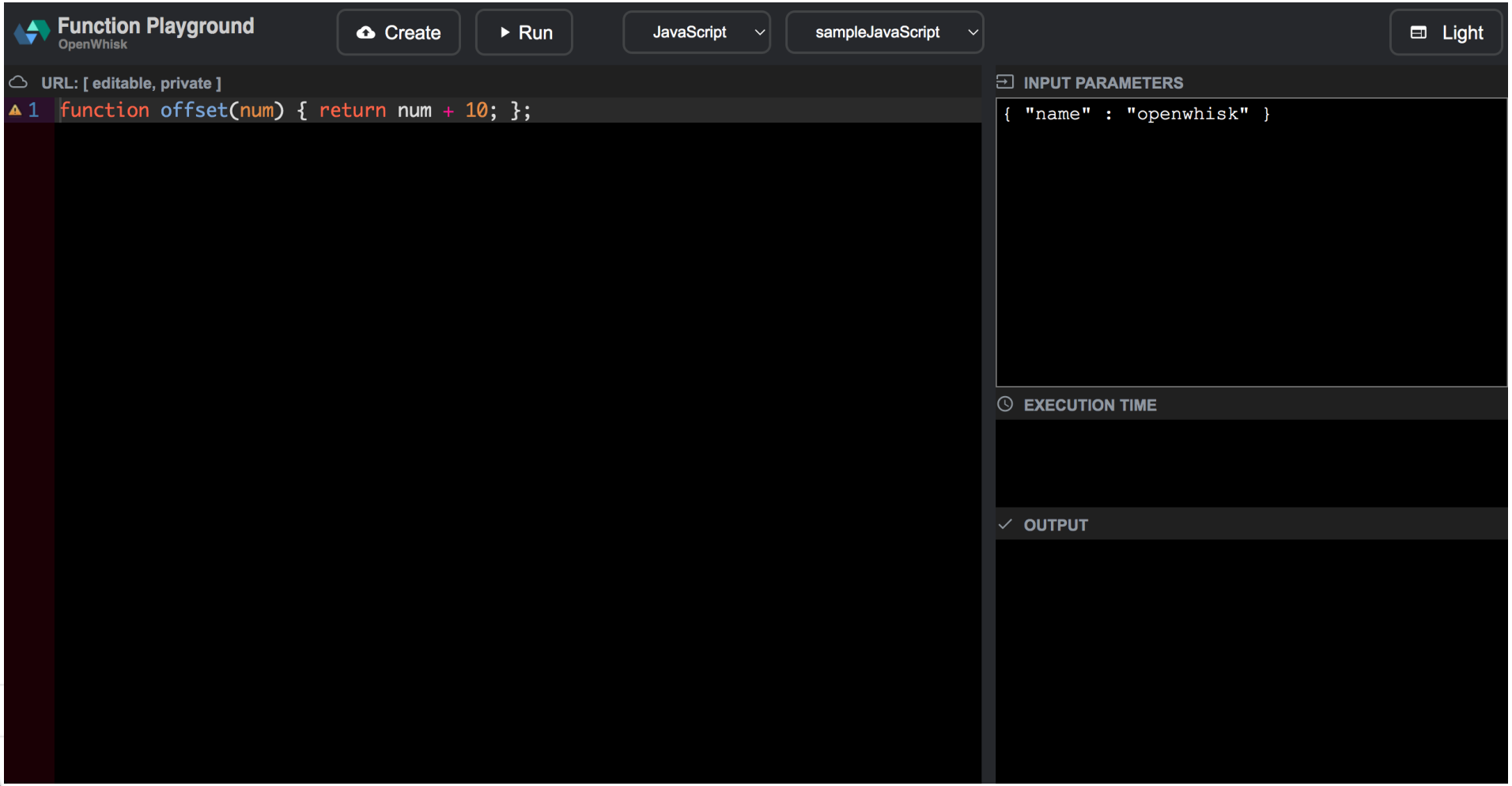
```
virtine void do_vpaes_cbc_encrypt(
    struct virtine_aes_state *state,
    const AES_KEY key,
    int encrypt) {
  // ...
}
```

HExSA

Serverless Virtines

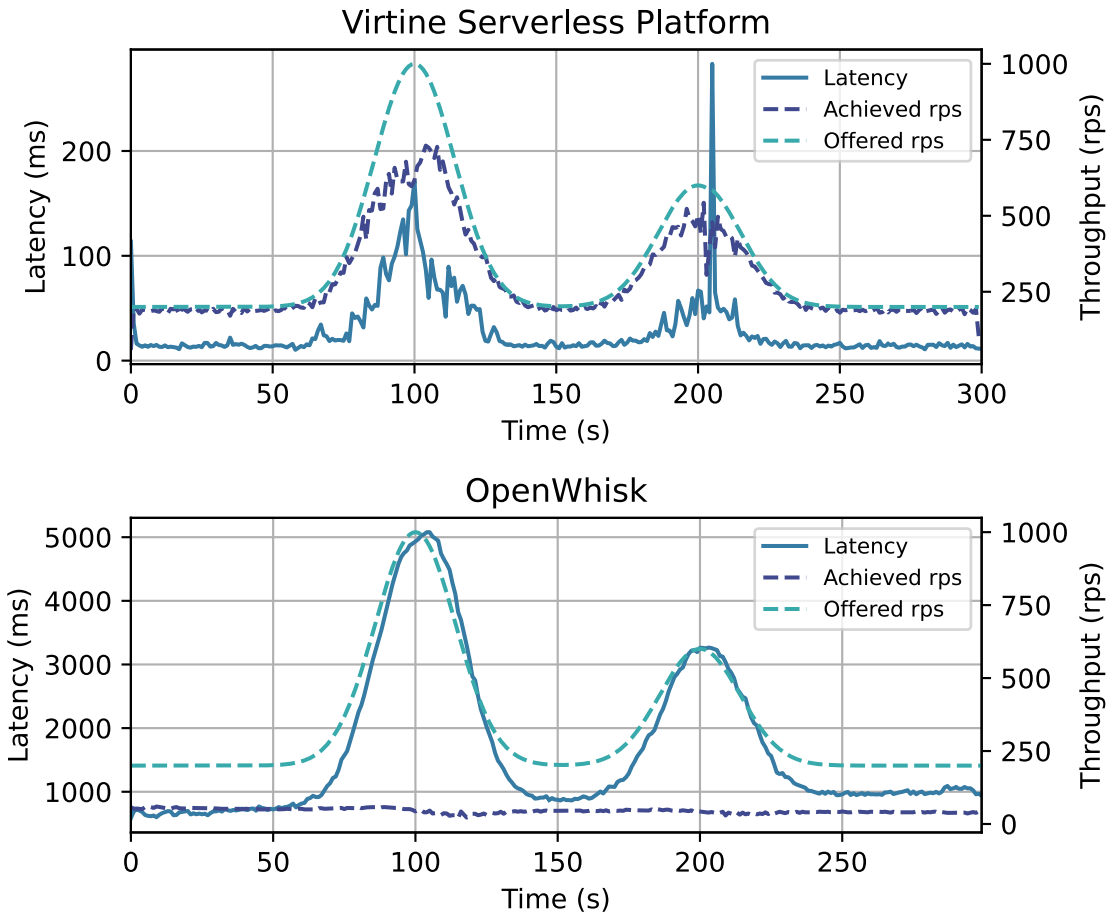
- OpenWhisk/AWS Lambda are good examples of modern serverless platforms
- Weak isolation between function instances!
- We developed our own using vitrines (based on OW)

Serverless Platform Interface



HExSA

We can use virtines as drop-in replacement for containers!



Isolating Functions at the Hardware Limit with Virtines

Nicholas C. Wanninger*
ncw@u.northwestern.edu
Northwestern University
Evanston, Illinois, USA

Joshua J. Bowden
jbowden@hawk.iit.edu
Illinois Institute of Technology
Chicago, Illinois, USA

Kirtankumar Shetty
kshetty11@hawk.iit.edu
Illinois Institute of Technology
Chicago, Illinois, USA

Ayush Garg
agarg22@hawk.iit.edu
Illinois Institute of Technology
Chicago, Illinois, USA

Kyle C. Hale
khale@cs.iit.edu
Illinois Institute of Technology
Chicago, Illinois, USA

Abstract

An important class of applications, including programs that leverage third-party libraries, programs that use user-defined functions in databases, and serverless applications, benefit from isolating the execution of untrusted code at the granularity of individual functions or function invocations. However, existing isolation mechanisms were not designed for this use case; rather, they have been adapted to it. We introduce

Keywords: virtines, virtualization, isolation

ACM Reference Format:
Nicholas C. Wanninger, Joshua J. Bowden, Kirtankumar Shetty, Ayush Garg, and Kyle C. Hale. 2022. Isolating Functions at the Hardware Limit with Virtines. In *Seventeenth European Conference on Computer Systems (EuroSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3492321.3519552

HExSA

Thanks!

- HExSA Lab: hexsa.halek.co
- Thanks to MD Ali, Conghao Liu, Brian Tauro, Stefan Muller
- Virtines code available: **github.com/virtines**
- **Thanks to Nick Wanninger (virtines lead author)**
  - Email: ncw@u.northwestern.edu
  - Website: https://nickw.io
- Also Josh Bowden, Kirtan Shetty, Ayush Garg

HExSA