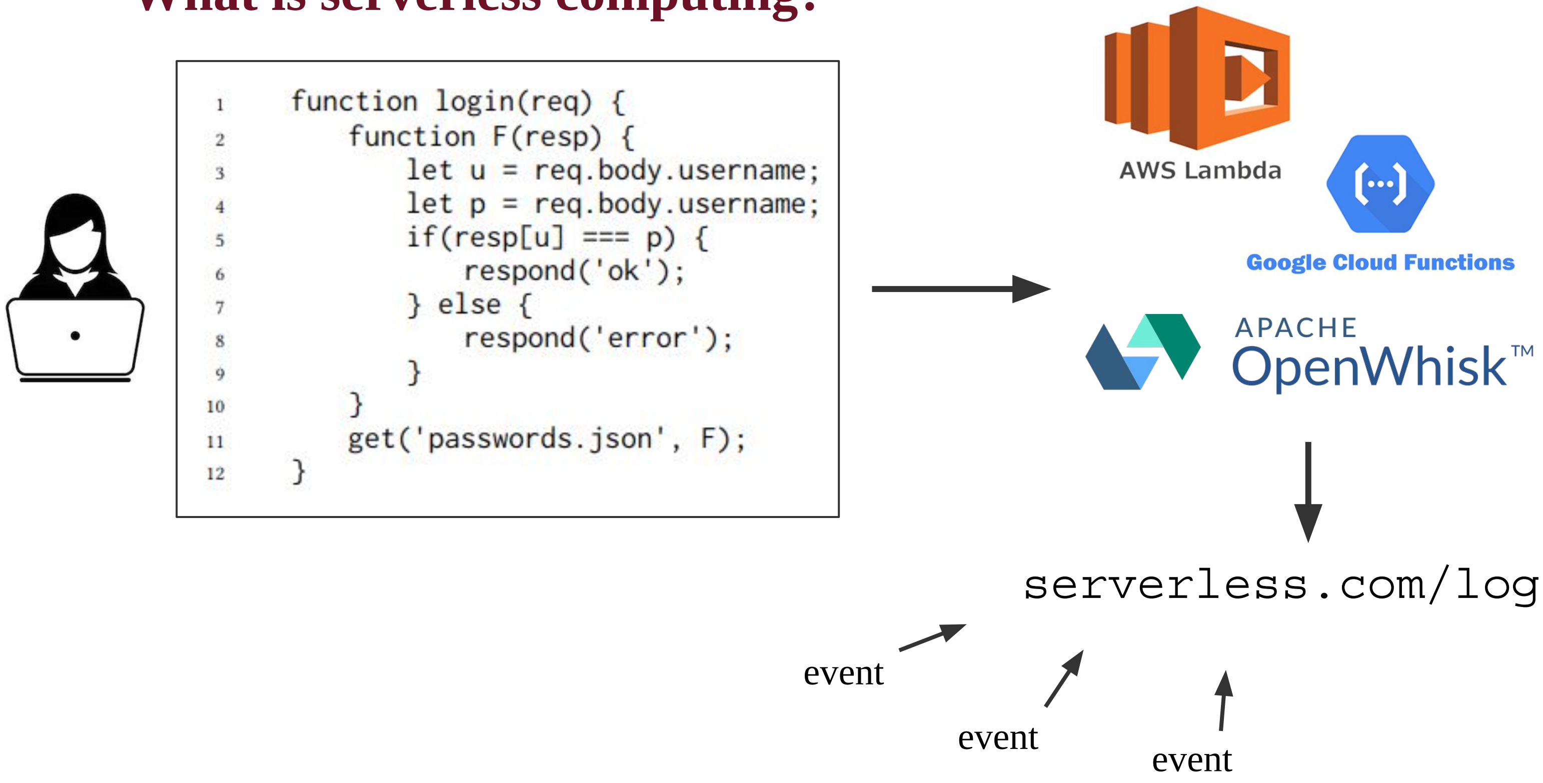
A Language-based Serverless Function Accelerator

Emily Herbert
Arjun Guha

What is serverless computing?

Approach to cloud computing… without servers… with servers

What is serverless computing?



```
function login(req) {
    function F(resp) {
        let u = req.body.username;
        let p = req.body.username;
        if(resp[u] === p) {
            respond('ok');
        } else {
            respond('error');
        }
    }
    get('passwords.json', F);
}
```

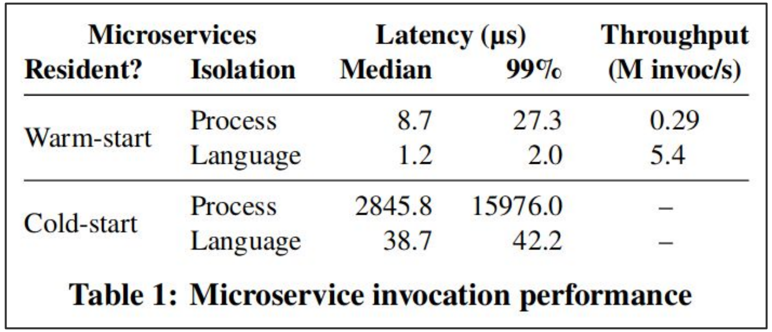
Problems with serverless computing

- JavaScript is ill-suited for serverless computing
  - Can consume a significant amount of time and memory
  - Require an operating system sandbox
- These sandboxes incur slowdowns [1]

[1] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. 2019.Architectural Implications of Function-as-a-Service Computing. In IEEE/ACM International Symposium on Microarchitecture (MICRO)

Rust as an alternative

- Boucher et al. present a serverless platform that runs functions written in Rust [2]
- Leverages Rust's language-level guarantees to run multiple serverless functions in one process



Microservices		Latency (μs)		Throughput
Resident?	**Isolation**	**Median**	**99%**	**(M invoc/s)**
Warm-start	Process	8.7	27.3	0.29
	Language	1.2	2.0	5.4
Cold-start	Process	2845.8	15976.0	–
	Language	38.7	42.2	–

Table 1: Microservice invocation performance

[2] Sol Boucher, Anuj Kalia, David G Andersen, and Michael Kaminsky. 2018. Putting the "Micro" back in microservices. In USENIX Annual Technical Conference (ATC).

Rust as an alternative

- Rust is difficult to learn
- Rust’s safety alone is not strong enough for serverless computing
  - CPU monopolization
  - deadlocks
  - memory leaks
  - ...

Containerless

- Serverless function accelerator
- Seeks to improve serverless computing performance
- Uses **language-based isolation** instead of container-based isolation

Containerless

- **Transforms JavaScript code to Rust code** by means of a **traced-based intermediate representation**
- Employs the Rust type system to ensure memory-safety (language-based isolation)
- Runs serverless functions using the new language-based isolation

$$\text{JavaScript} \longrightarrow \text{trace IR} \longrightarrow \text{Rust}$$

Containerless

- Why use a IR?
- Compiling directly would suffer from **impedance mismatch**
  - Dynamic types v. static types
  - Garbage collection v. explicit memory management
  - Pointer aliasing
  - ...

Containerless

- Domain specific
- Utilizes common features of serverless functions
  - idempotent
  - short-lived
- Not a general purpose JavaScript to Rust compiler

Components

Three general components:

1. JavaScript to IR
2. IR to Rust
3. invoker

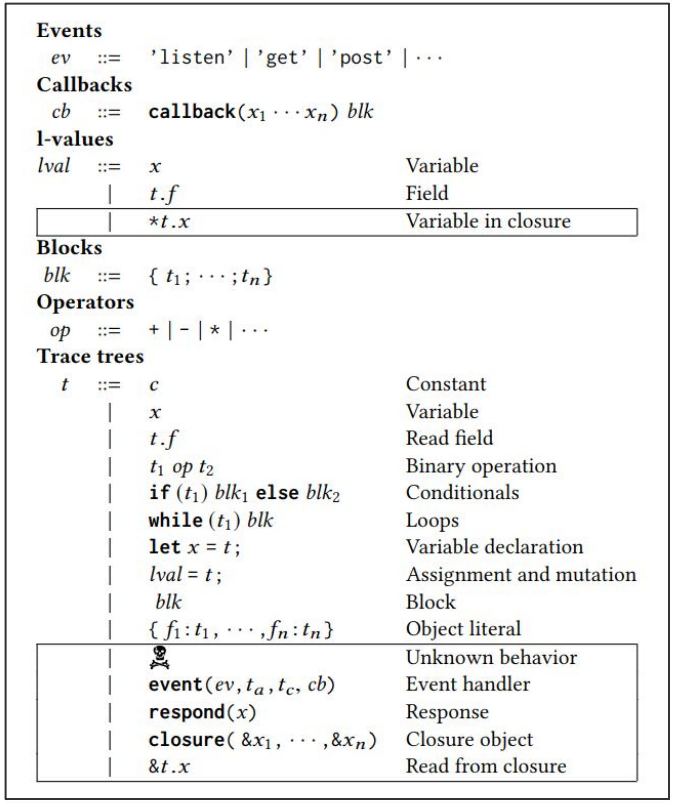
JavaScript to IR

- IR is **trace-tree built over multiple executions of the function**
- Similar to an execution trace, but a tree

JavaScript to IR

Key features:

1. ~~Functions~~
2. Closures (**closure**)
3. Unknown behavior (☠)
4. Callbacks (*cb*) and events (**event**)



Events			
ev	::=	`'listen'` | `'get'` | `'post'` | $\cdots$	
Callbacks			
cb	::=	`callback`$(x_1 \cdots x_n)$ blk	
l-values			
$lval$	::=	x	Variable
	|	$t.f$	Field
	|	$*t.x$	Variable in closure
Blocks			
blk	::=	$\{ t_1 ; \cdots ; t_n \}$	
Operators			
op	::=	$+ \mid - \mid * \mid \cdots$	
Trace trees			
t	::=	c	Constant
	|	x	Variable
	|	$t.f$	Read field
	|	$t_1 \; op \; t_2$	Binary operation
	|	`if` (t_1) blk_1 `else` blk_2	Conditionals
	|	`while` (t_1) blk	Loops
	|	`let` $x = t;$	Variable declaration
	|	$lval = t;$	Assignment and mutation
	|	blk	Block
	|	$\{ f_1 : t_1 , \cdots , f_n : t_n \}$	Object literal
	|	☠	Unknown behavior
	|	`event`(ev, t_a, t_c, cb)	Event handler
	|	`respond`(x)	Response
	|	`closure`$(\&x_1 , \cdots , \&x_n)$	Closure object
	|	$\&t.x$	Read from closure

JavaScript to IR

1. Instrument function with trace-building runtime statements

```
let c = require('containerless');

function main(req) {
    function F(resp) {
        let u = req.body.username;
        let p = req.body.password;
        if (resp[u] === p) {
            c.respond('ok');
        } else {
            c.respond('error');
        }
    }
    c.get('passwords.json', F);
}

c.listen(main);
```

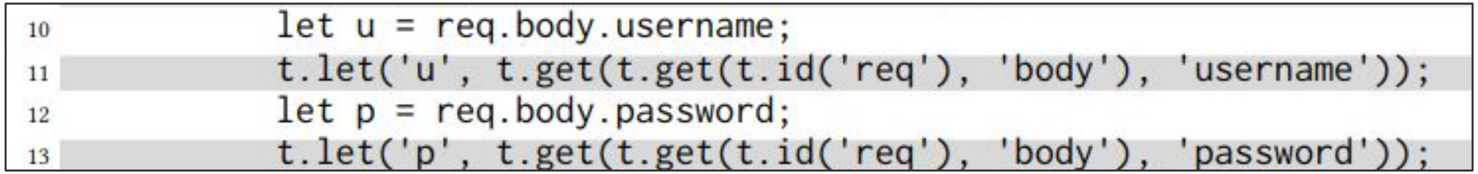
```
let c = require('containerless');
let t = require('containerless/tracing');

function main(req) {
    let [_req] = t.popArgs();
    function F(resp) {
        let [_resp] = t.popArgs();
        let _clos = t.popClosure();
        t.let('req', t.getClos(_clos, 'req'));
        let u = req.body.username;
        t.let('u', t.get(t.get(t.id('req'), 'body'), 'username'));
        let p = req.body.password;
        t.let('p', t.get(t.get(t.id('req'), 'body'), 'password'));
        t.if(t.eq(t.vget(_resp, t.id('u')), t.id('p')));
        if (resp[u] === p) {
            t.ifTrue();
            t.pushArgs(t.str('ok'));
            c.respond('ok');
            t.popResult();
        } else {
            t.ifFalse();
            t.pushArgs(t.str('error'));
            c.respond('error');
            t.popResult();
        }
        t.exitIf();
        t.exitFunction(t.undefined);
    }
    t.let('F', t.closure({ 'req': _req }));
    t.pushArgs([t.str('passwords.json'), t.id('F')]);
    c.get('passwords.json', F);
    t.popResult();
}

c.listen(main);
```

JavaScript to IR

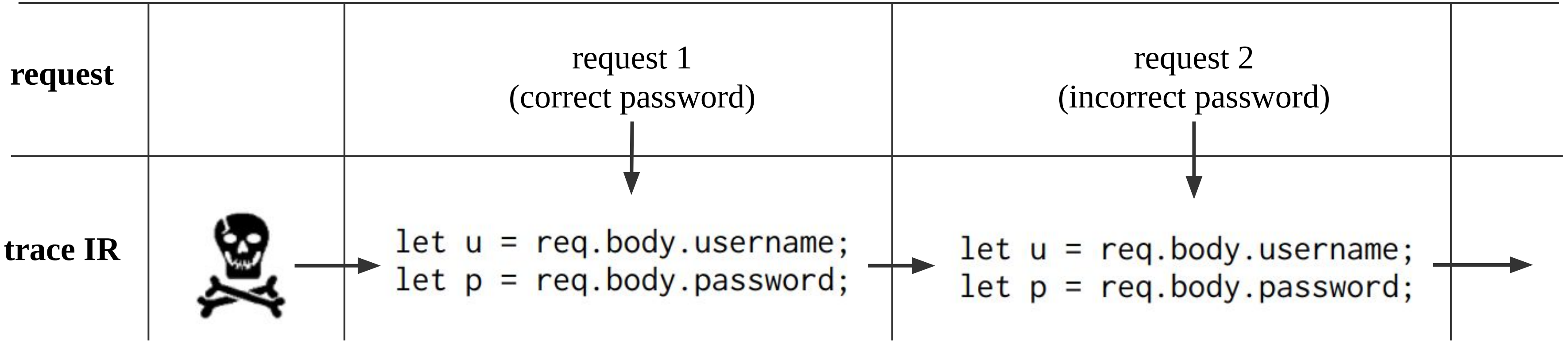
2. Execute function multiple times to build a trace tree

- Linked with library
- Builds incrementally
- Tree fragments are merged

UMass Amherst | College of Information & Computer Sciences

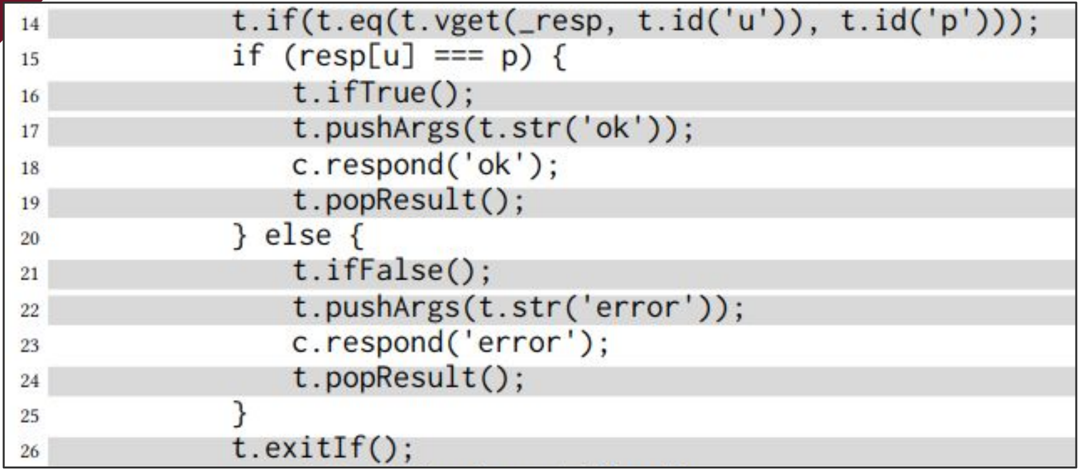


```
        let u = req.body.username;
        t.let('u', t.get(t.get(t.id('req'), 'body'), 'username'));
        let p = req.body.password;
        t.let('p', t.get(t.get(t.id('req'), 'body'), 'password'));
```

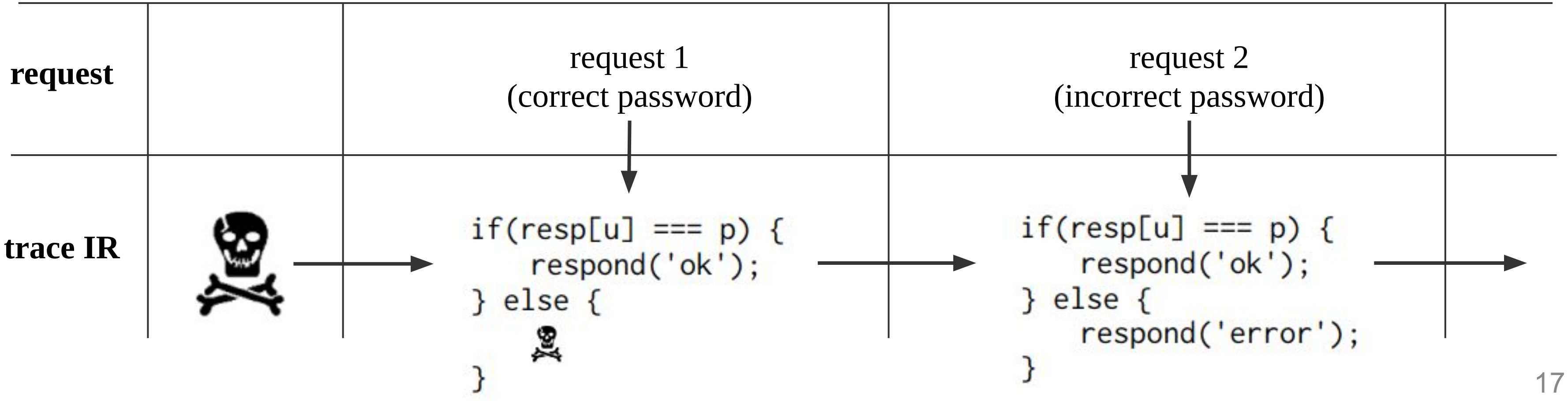


```
let u = req.body.username;
let p = req.body.password;
```

```
let u = req.body.username;
let p = req.body.password;
```



```
        t.if(t.eq(t.vget(_resp, t.id('u')), t.id('p')));
        if (resp[u] === p) {
            t.ifTrue();
            t.pushArgs(t.str('ok'));
            c.respond('ok');
            t.popResult();
        } else {
            t.ifFalse();
            t.pushArgs(t.str('error'));
            c.respond('error');
            t.popResult();
        }
        t.exitIf();
```



```
if(resp[u] === p) {
    respond('ok');
} else {

}
```

```
if(resp[u] === p) {
    respond('ok');
} else {
    respond('error');
}
```

UMass Amherst | College of Information & Computer Sciences

```
    function main(req) {
        let [_req] = t.popArgs();
        function F(resp) {
            let [_resp] = t.popArgs();
            let _clos = t.popClosure();
            t.let('req', t.getClos(_clos, 'req'));
```

```
            t.exitFunction(t.undefined);
        }
        t.let('F', t.closure({ 'req': _req }));
        t.pushArgs([t.str('passwords.json'), t.id('F')]);
        c.get('passwords.json', F);
        t.popResult();
    }

    c.listen(main);
```

request		request 1 (correct password)	request 2 (incorrect password)

trace IR		`event('listen', [], closure(), callback (clos, req) {`
		`  event('get', ['passwords.json'], closure(&req),`
		`    callback(clos, resp) {`
		`      let req = *(clos.req);`
		`      // body`
		`  });`
		`});`
			`event('listen', [], closure`
			`  event('get', ['passwords.`
			`    callback(clos, resp) {`
			`      let req = *(clos.req);`
			`      // body`
			`  });`
			`});`

```
let c = require('containerless');
let t = require('containerless/tracing');

function main(req) {
    let [_req] = t.popArgs();
    function F(resp) {
        let [_resp] = t.popArgs();
        let _clos = t.popClosure();
        t.let('req', t.getClos(_clos, 'req'));
        let u = req.body.username;
        t.let('u', t.get(t.get(t.id('req'), 'body'), 'username'));
        let p = req.body.password;
        t.let('p', t.get(t.get(t.id('req'), 'body'), 'password'));
        t.if(t.eq(t.vget(_resp, t.id('u')), t.id('p')));
        if (resp[u] === p) {
            t.ifTrue();
            t.pushArgs(t.str('ok'));
            c.respond('ok');
            t.popResult();
        } else {
            t.ifFalse();
            t.pushArgs(t.str('error'));
            c.respond('error');
            t.popResult();
        }
        t.exitIf();
        t.exitFunction(t.undefined);
    }
    t.let('F', t.closure({ 'req': _req }));
    t.pushArgs([t.str('passwords.json'), t.id('F')]);
    c.get('passwords.json', F);
    t.popResult();
}

c.listen(main);
```

3. Produce trace IR!

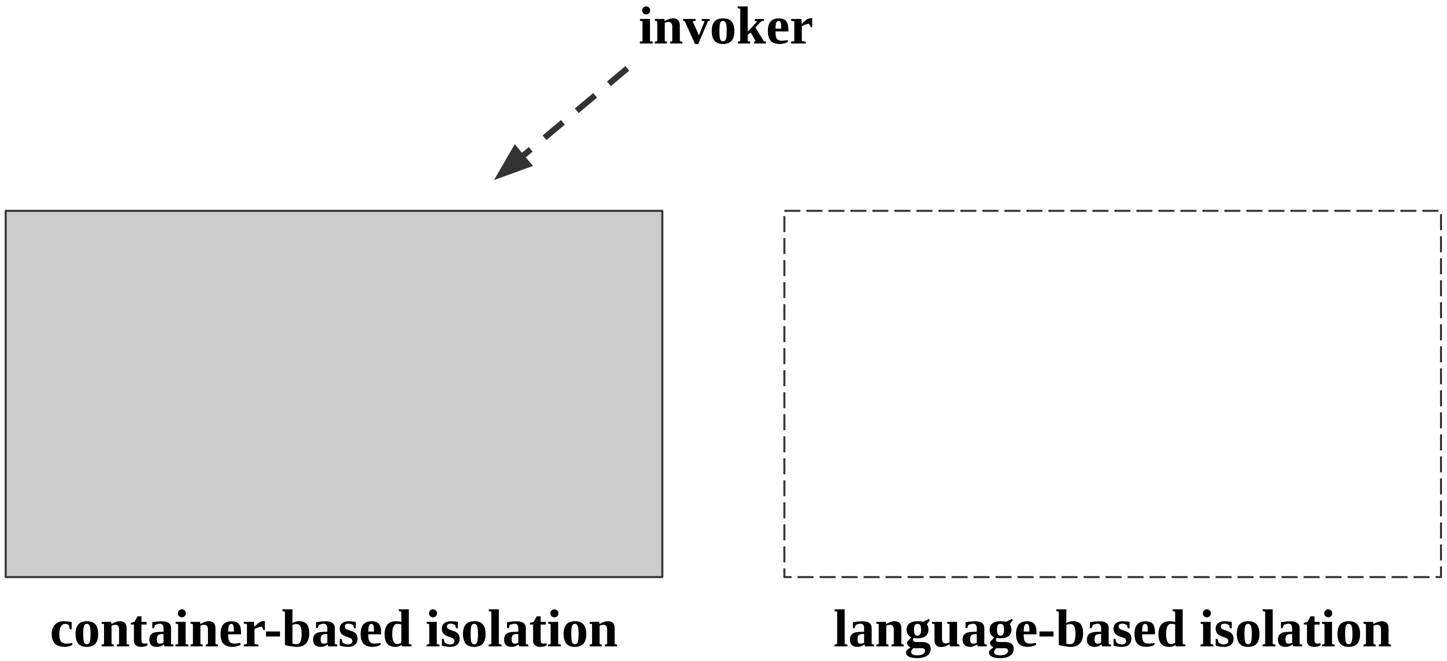
```
event('listen', [], closure(), callback (clos, req) {
  event('get', ['passwords.json'],
    closure(&req), callback(clos, resp) {
      let req = *(clos.req);
      let u = req.body.username;
      let p = req.body.password;
      if (resp[u] === p) {
        respond('ok');
      } else {
        respond('error');
      }
    });
});
```

IR to Rust

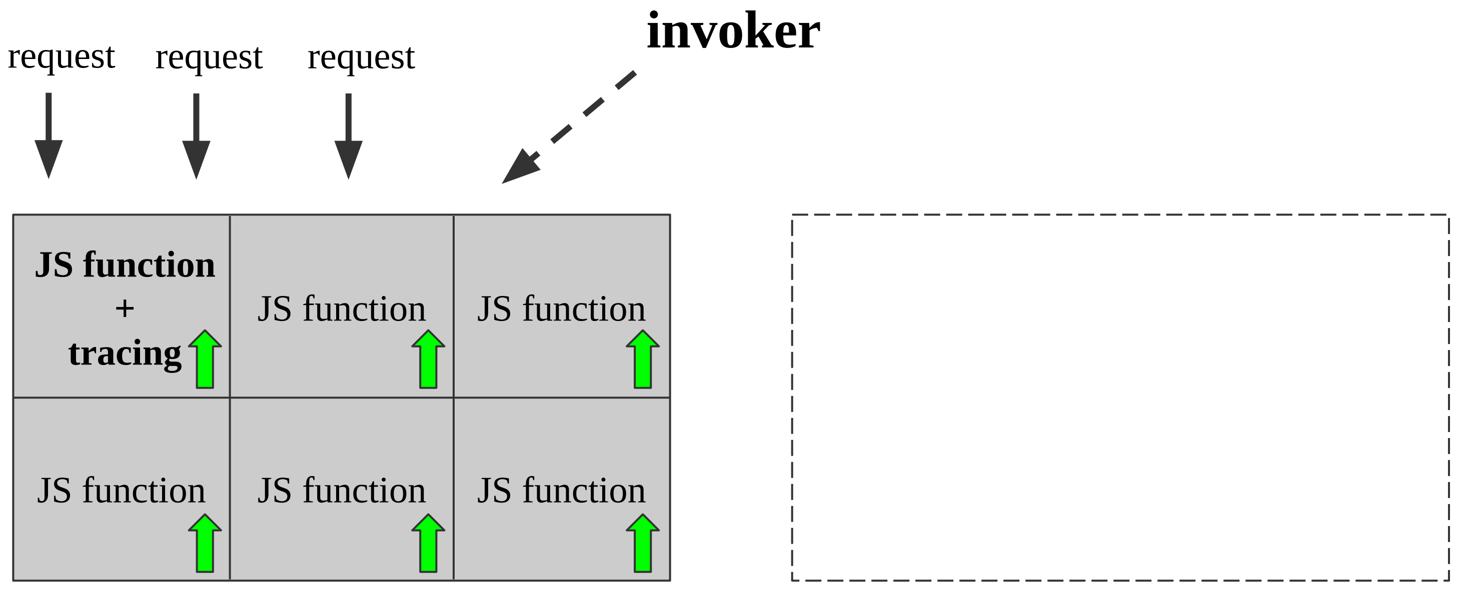
1. Transform callbacks in the trace IR to a state machine
2. Impose CPU and memory limits on the program
3. Inject all values into a **dynamic type**
4. Use **arena allocation** to resolve Rust lifetimes
5. Produce Rust code!

UMass Amherst | College of Information & Computer Sciences

Invoker

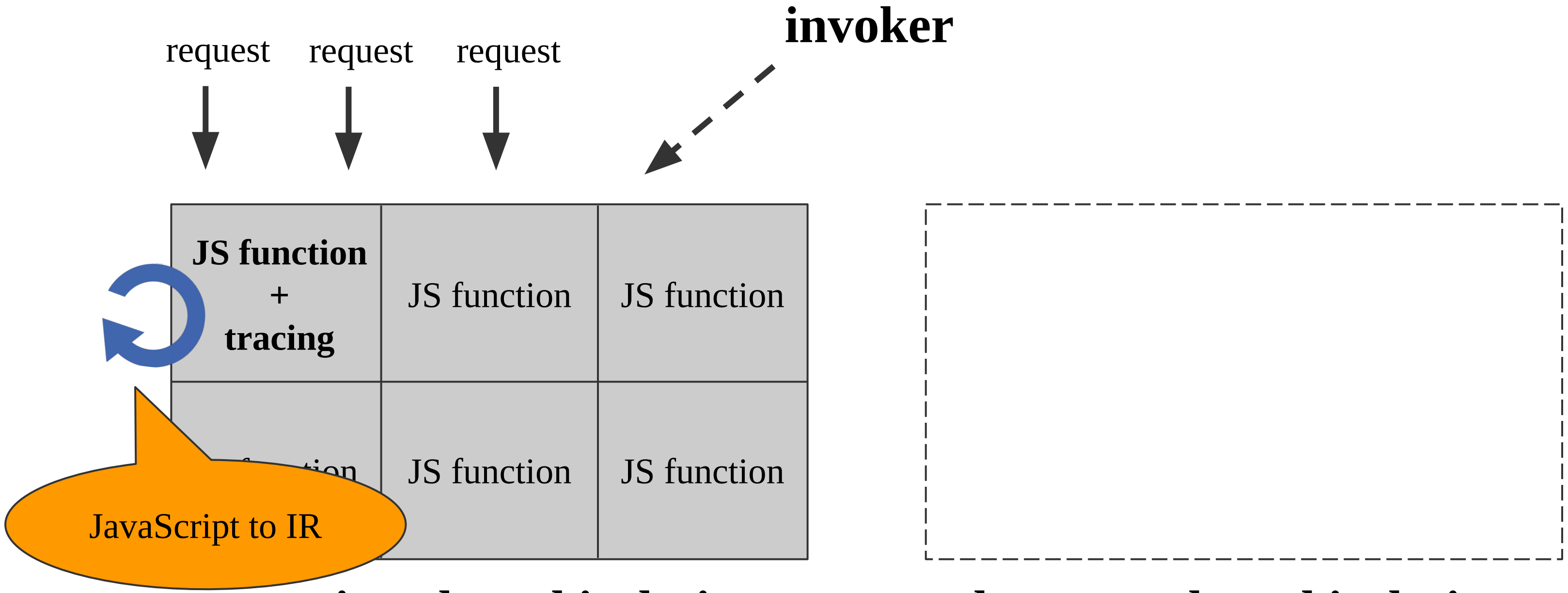


Invoker



container-based isolation **language-based isolation**

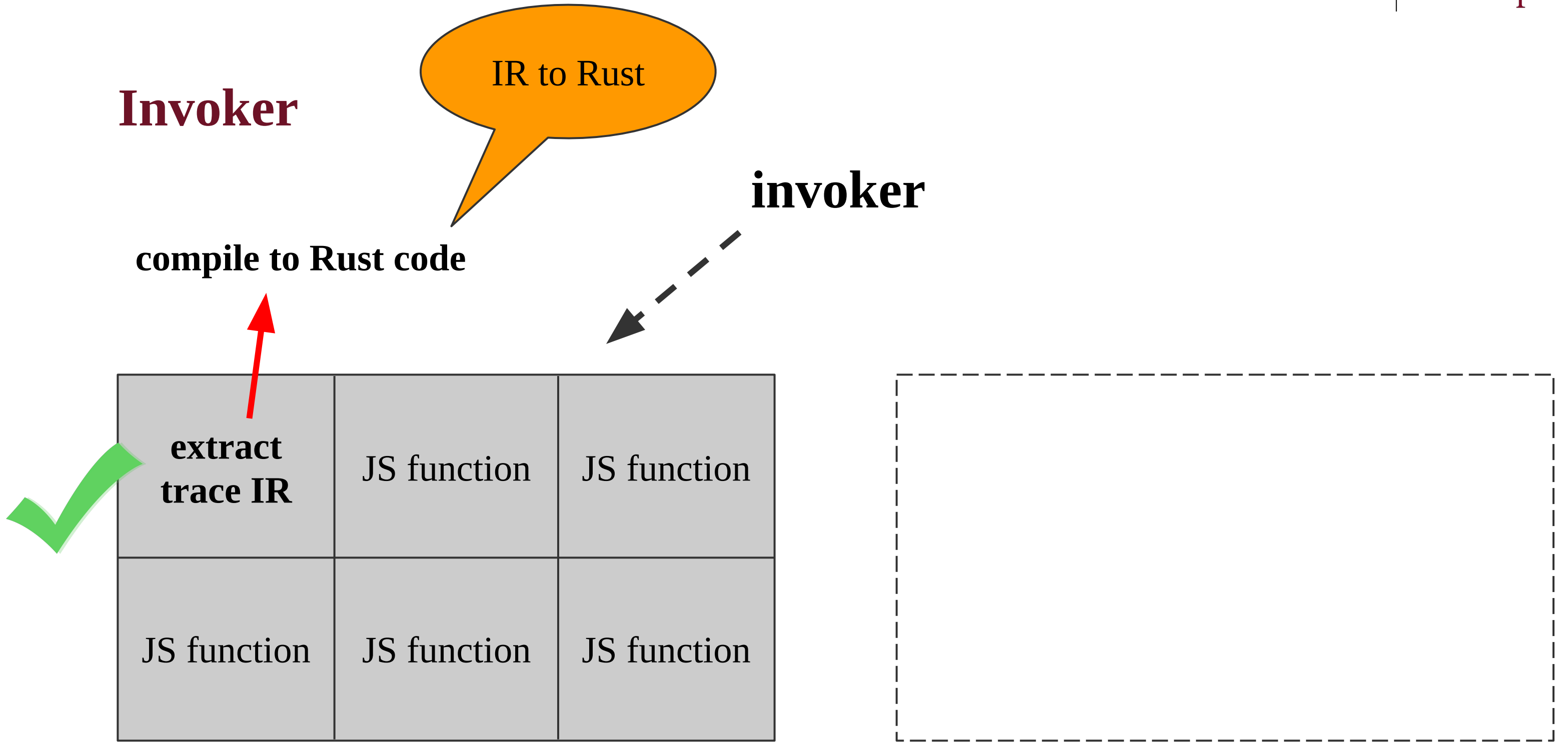
Invoker



container-based isolation

language-based isolation

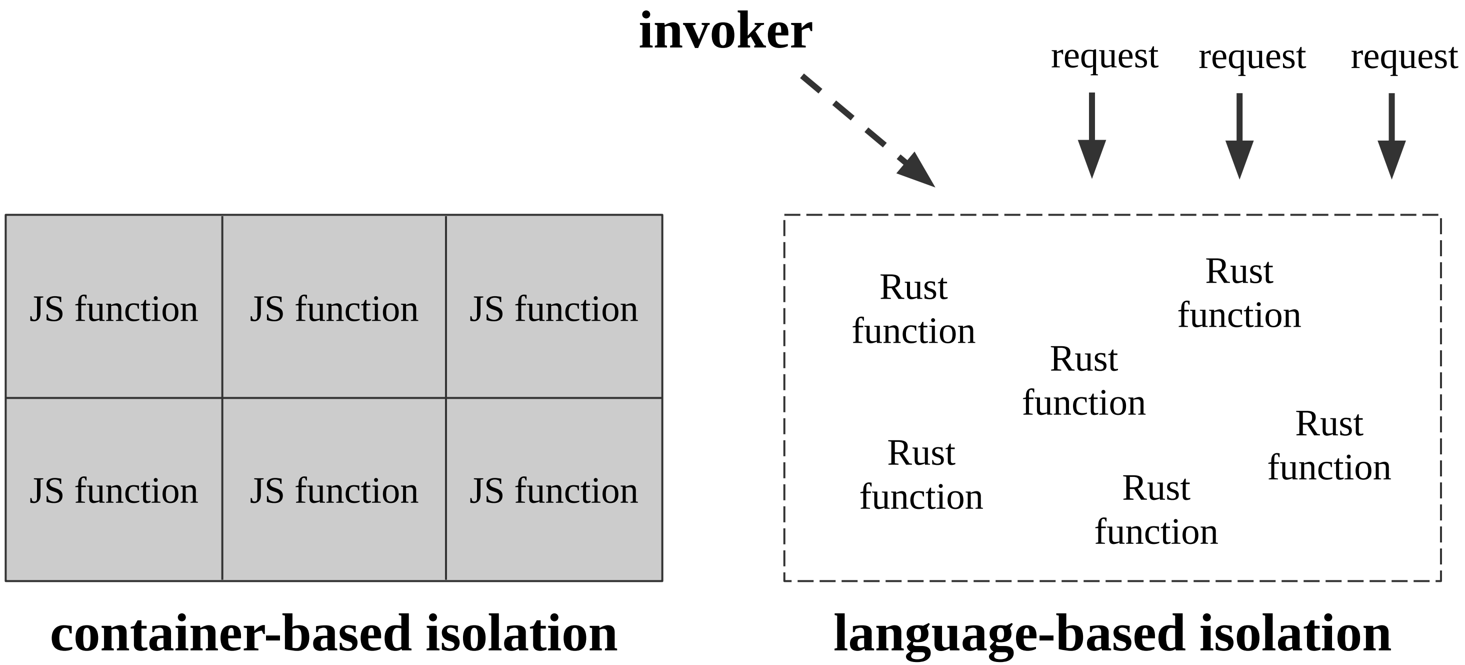
Invoker



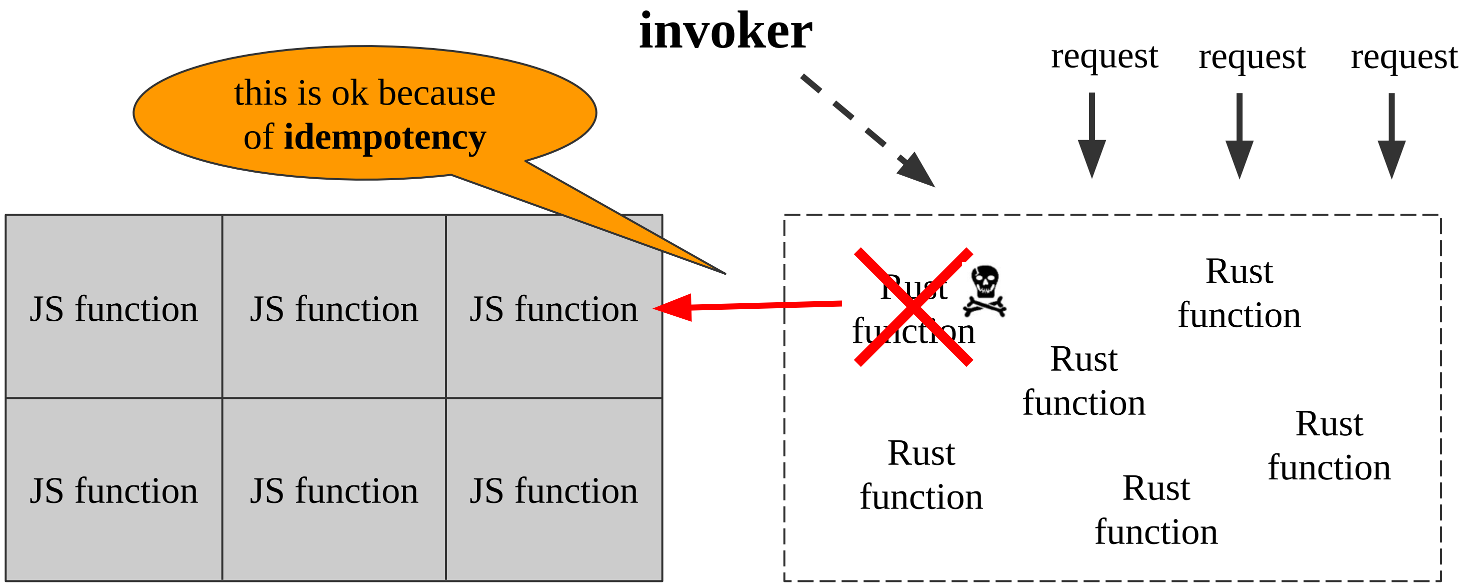
container-based isolation

language-based isolation

Invoker


invoker
request
request
request
JS function
JS function
JS function
JS function
JS function
JS function
Rust function
Rust function
Rust function
Rust function
Rust function
Rust function
container-based isolation
language-based isolation

Invoker



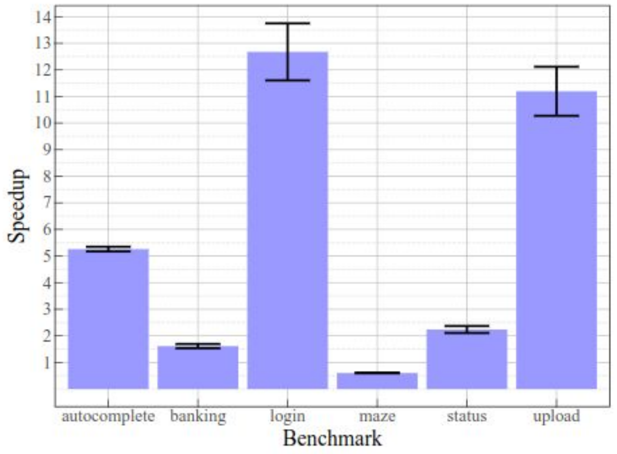
container-based isolation **language-based isolation**

Containerless

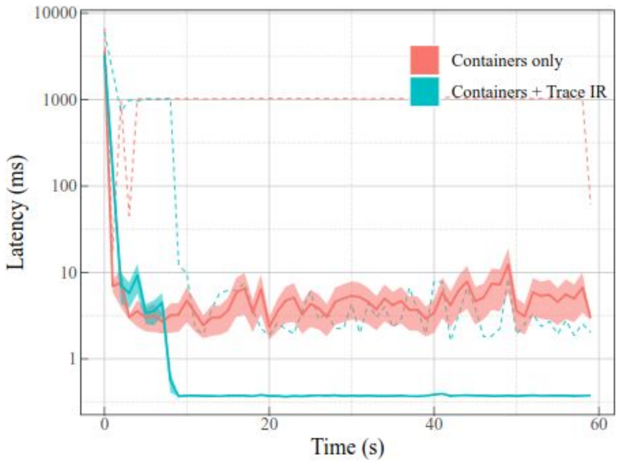
Three general components:

1. JavaScript to IR $\longrightarrow$ Eliminates functions, etc.
2. IR to Rust $\longrightarrow$ Dynamic type, arena allocation, etc.
3. invoker $\longrightarrow$ Manages language-based isolation

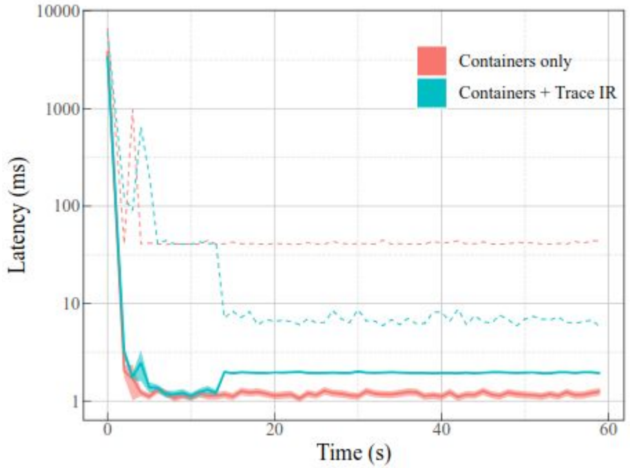
Latency



all benchmarks



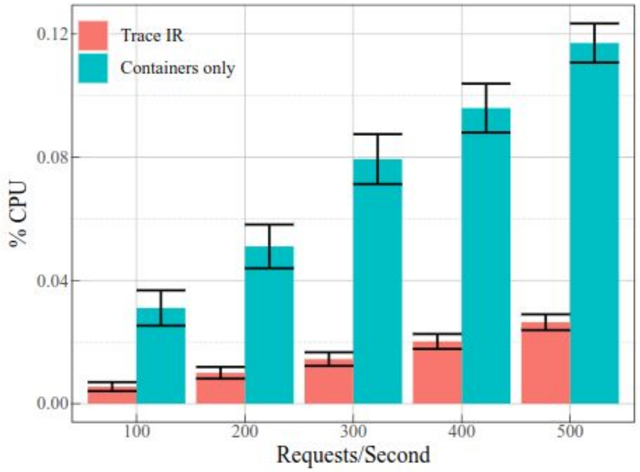
authorize benchmark



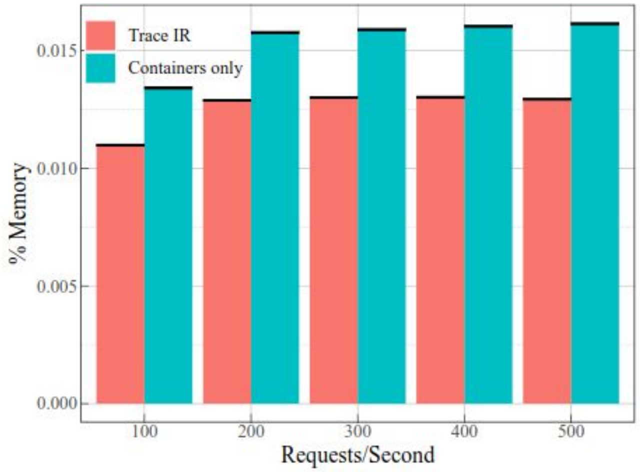
maze benchmark

UMass Amherst | College of Information & Computer Sciences

Utilization



CPU utilization



memory utilization

UMass Amherst | College of Information & Computer Sciences

Thanks!