

# How Lyft Used Envoy to Rethink Microservice Development

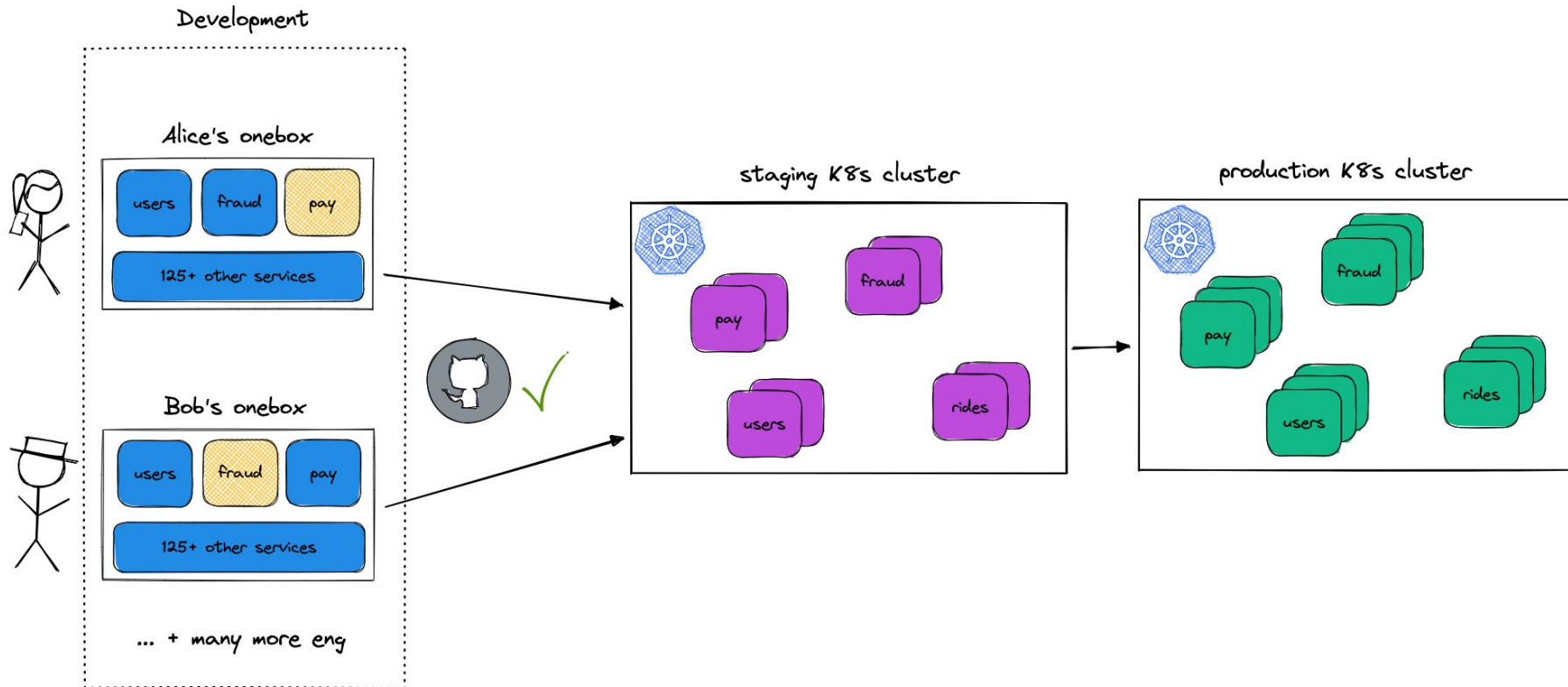
*Matt Grossman*

# Developer Experience:

*Create infrastructure and tooling to safely accelerate developer productivity*

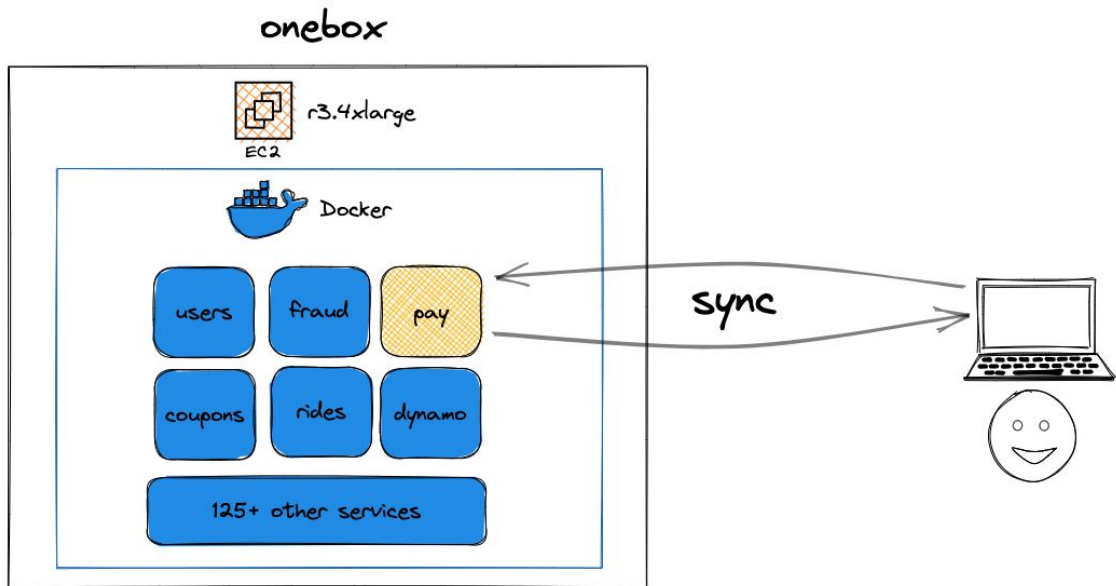
# Problems with Lyft-in-a-box

# The change pipeline



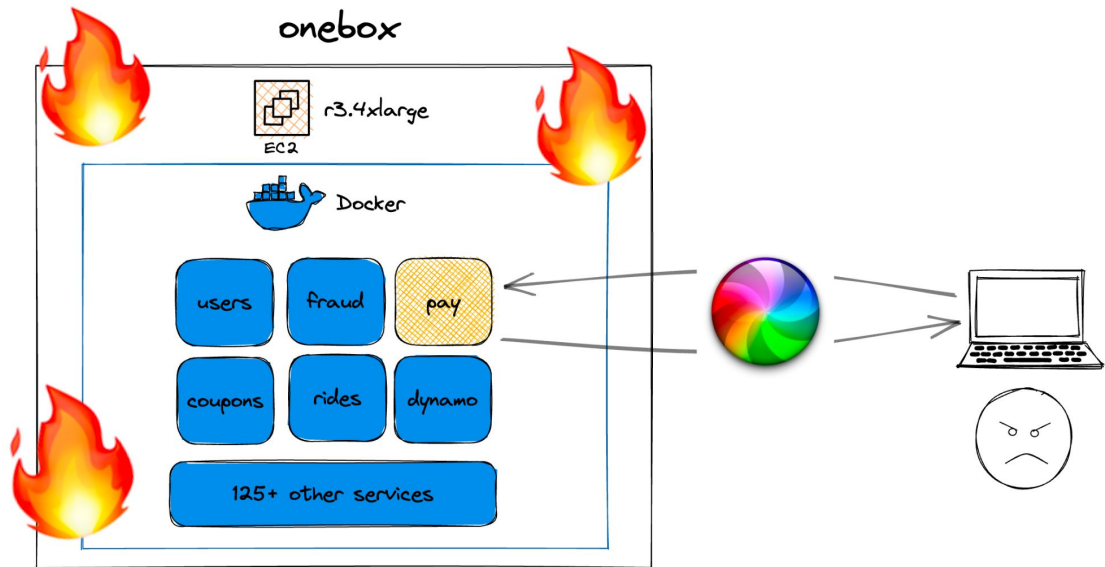
# The current environment

- Containers in XL EC2s
- Sync from local → remote for hot-reload
- Run every service at Lyft



# Problems arise

- Scales by  
ENG ~~X~~ Services
- Environment drift
- Unclear ownership



# An unfortunately normal workflow

- Provision a onebox
- Wait an hour
- Re-provision until environment works
- Give up
- Deploy and test in staging

# Sharing a developer environment

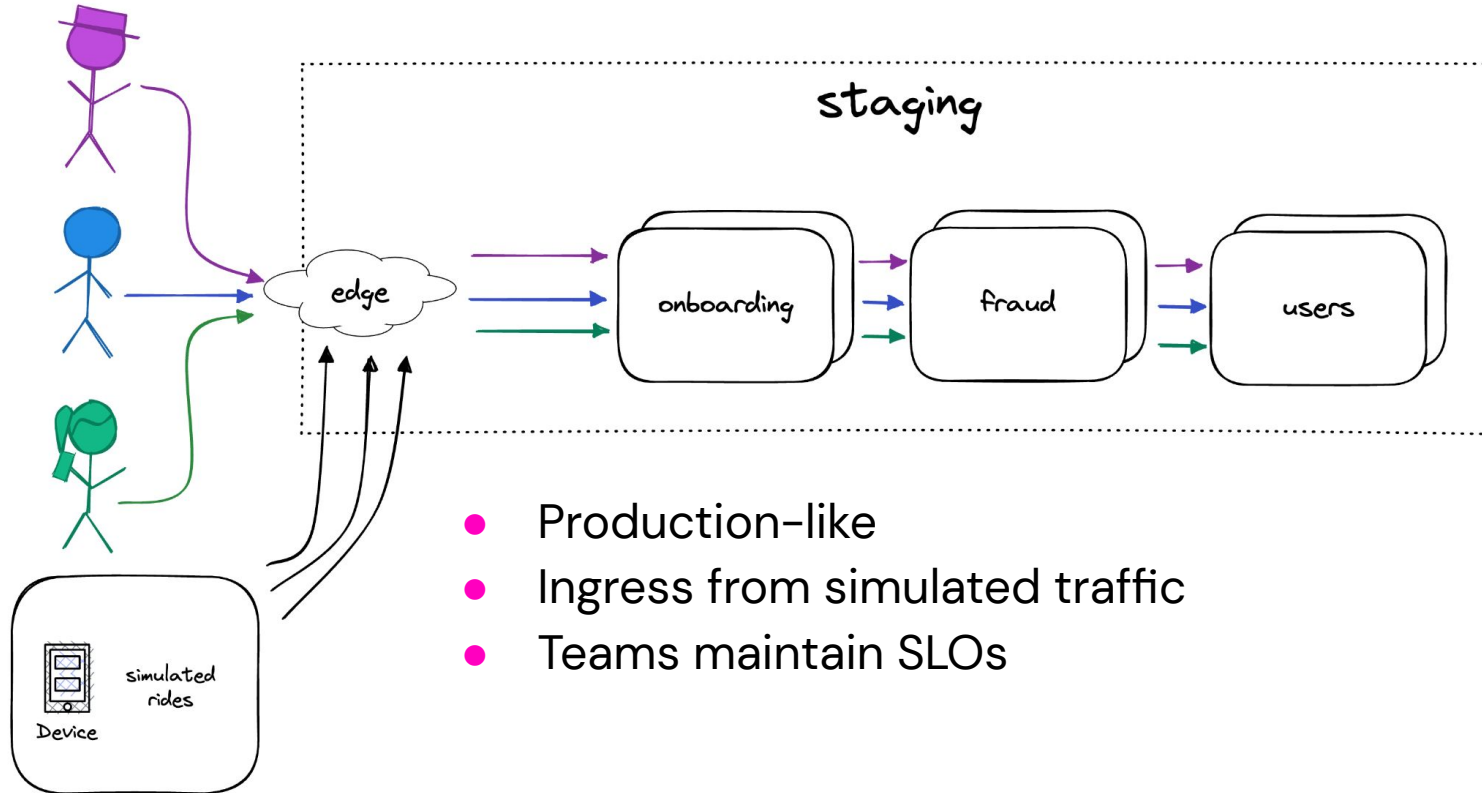


# What's so wrong with deploying to staging?

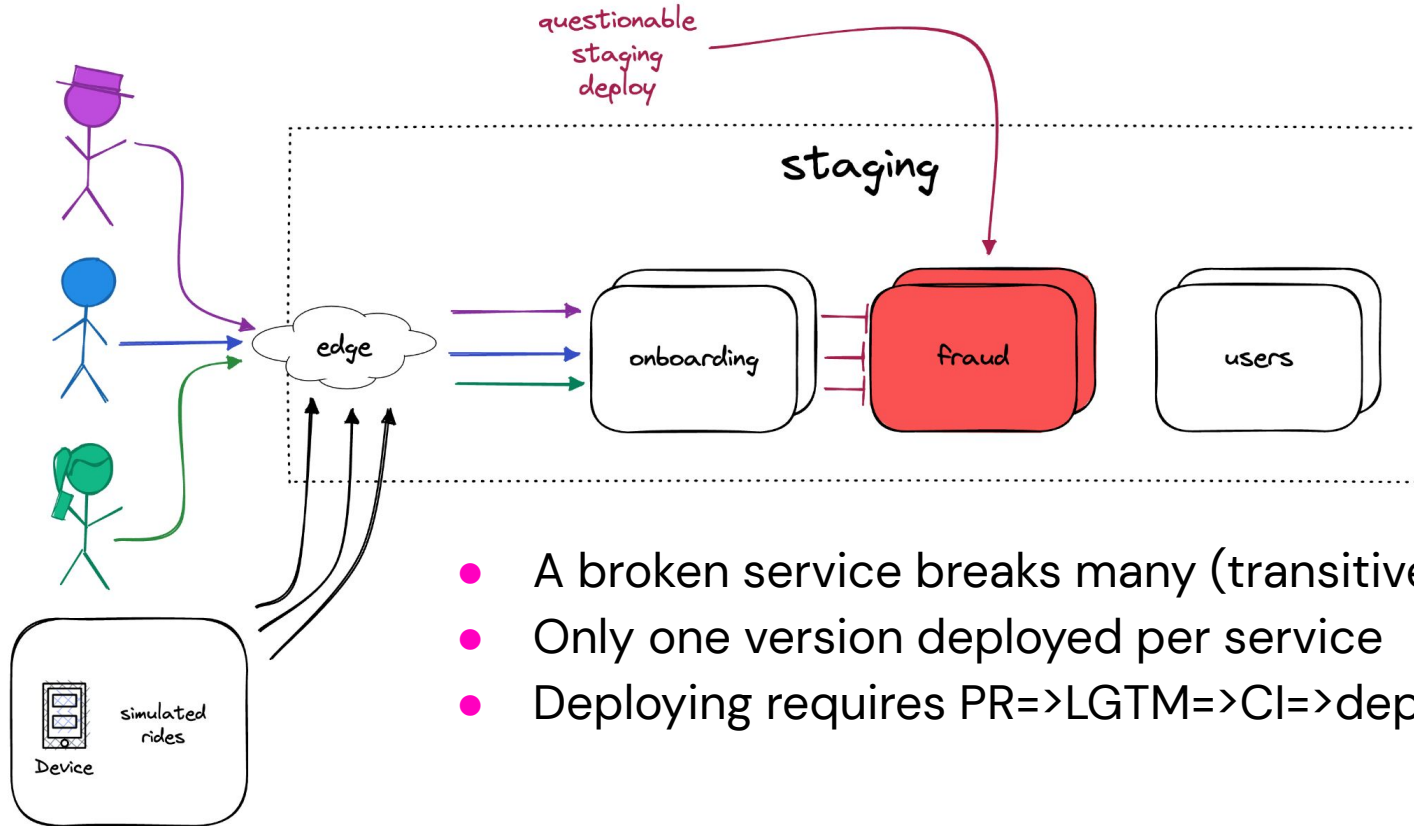
- Provision a onebox
- Wait an hour
- Re-provision until environment works
- Give up
- ***Deploy and test in staging***



# Advantages of staging



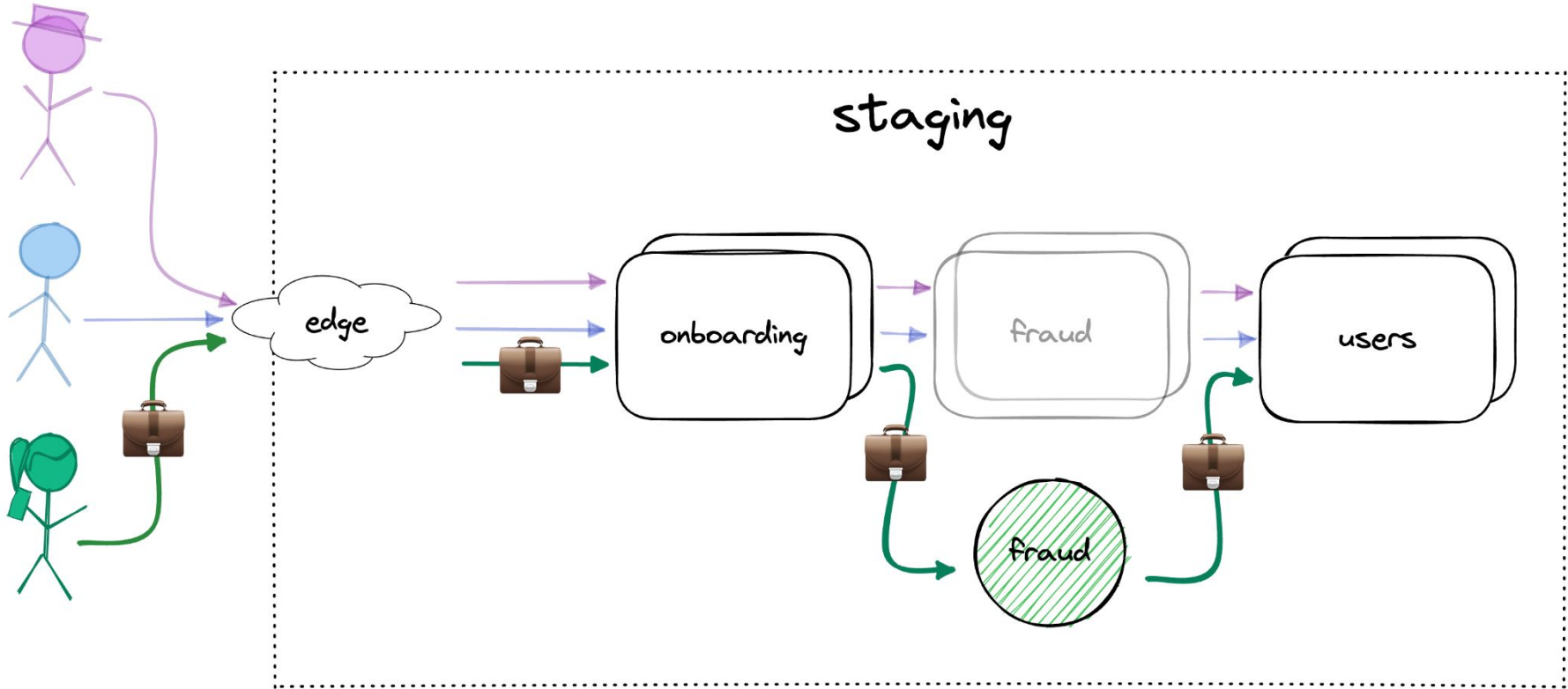
# Problems with shared environments



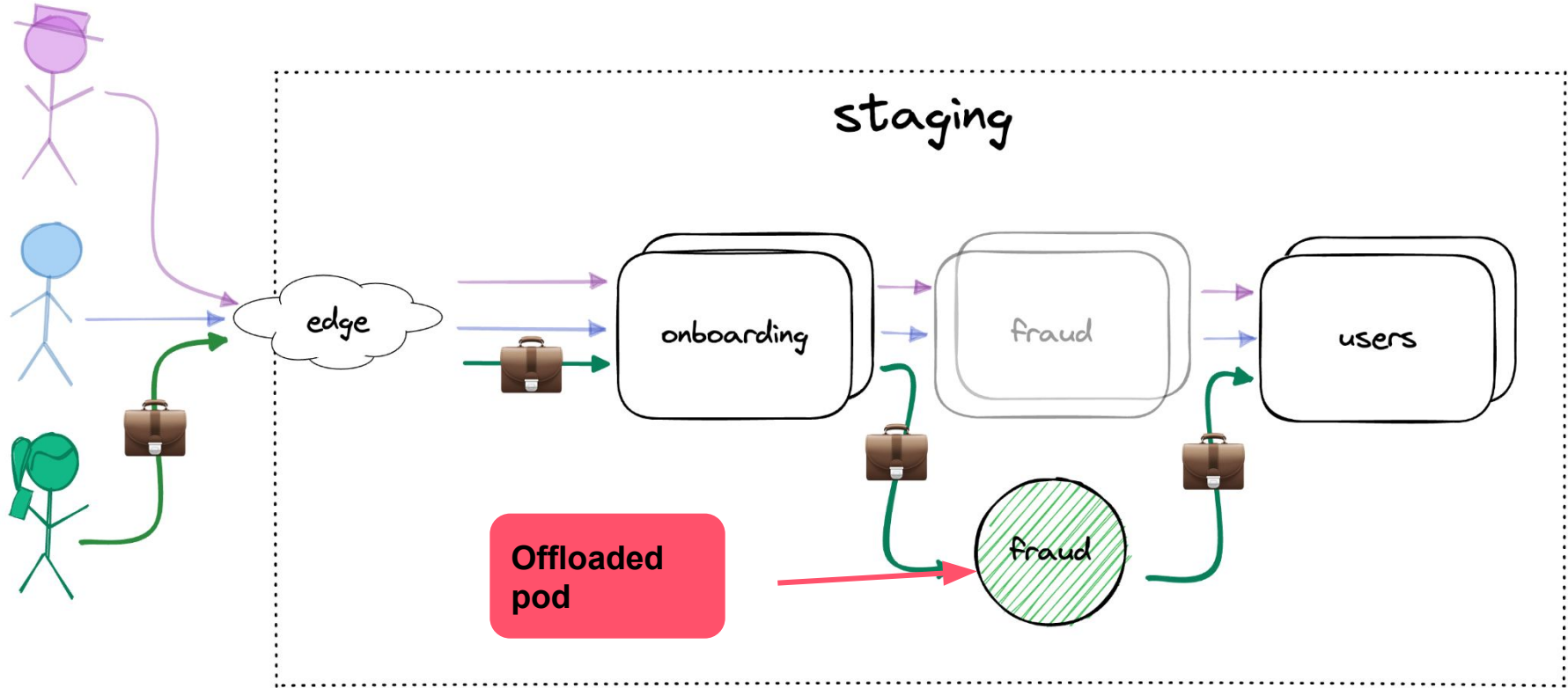
- A broken service breaks many (transitive?) consumers
- Only one version deployed per service
- Deploying requires PR=>LGTM=>CI=>deploy

Staging  
overrides

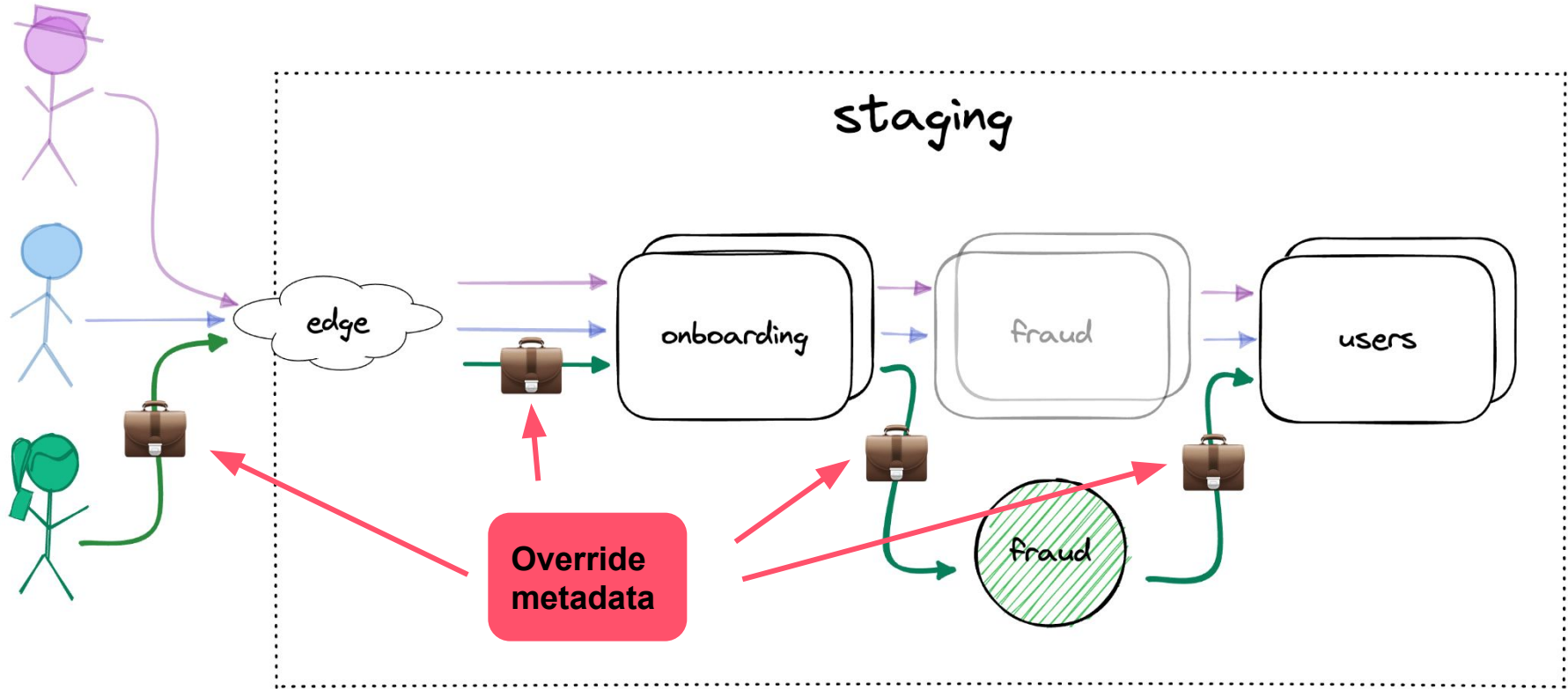
# Goal: provide isolation within staging



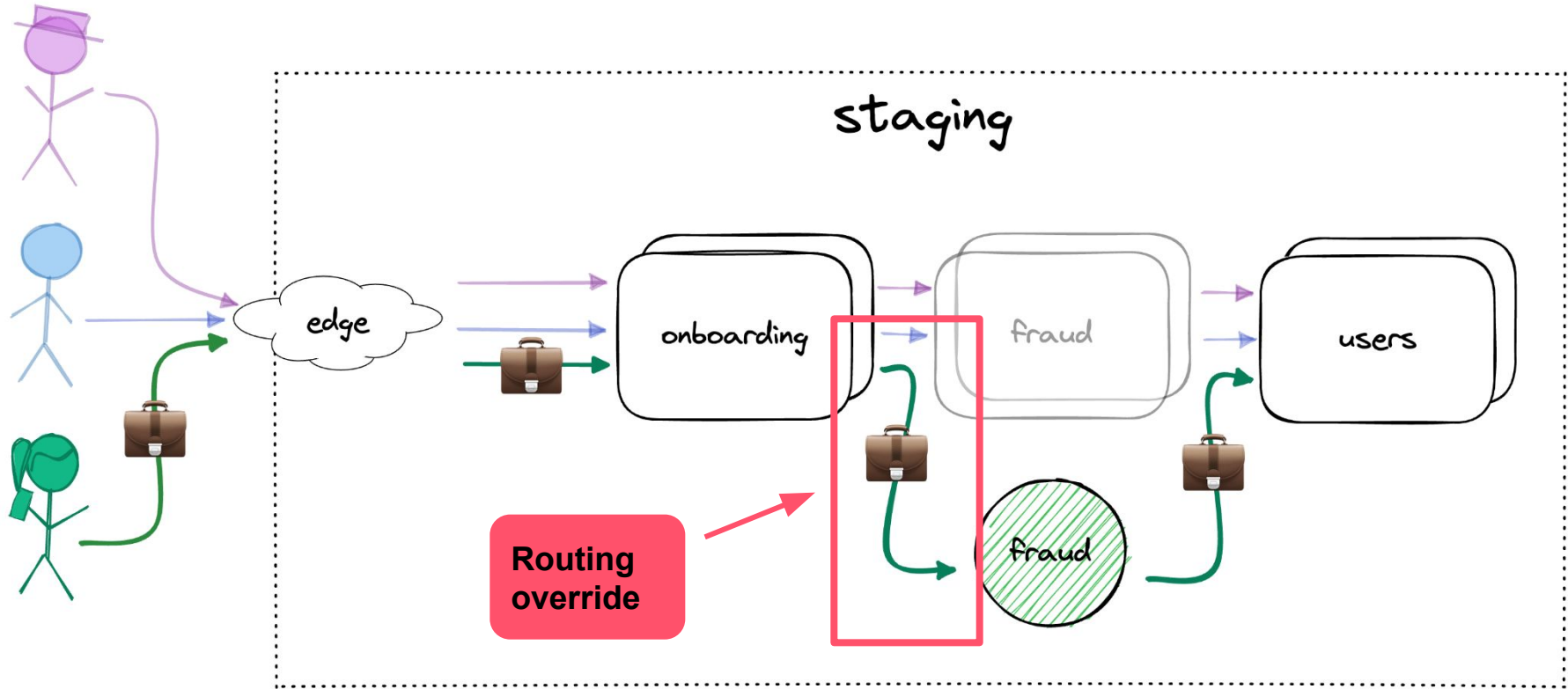
# Goal: provide isolation within staging



# Goal: provide isolation within staging



# Goal: provide isolation within staging





# Introducing: Staging Overrides

- Unregistered “offloaded” pods
- Propagated override metadata
- Dynamic routing overrides

# Introducing: Staging Overrides

- Unregistered “offloaded” pods
  - ➔ EDS exclusion in control-plane
- Propagated override metadata
- Dynamic routing overrides


# Introducing: Staging Overrides


- Unregistered “offloaded” pods
  - ➔ EDS exclusion in control-plane
- Propagated override metadata
  - ➔ OpenTracing baggage
- Dynamic routing overrides

# Introducing: Staging Overrides


- Unregistered “offloaded” pods
  - ➔ EDS exclusion in control-plane
- Propagated override metadata
  - ➔ OpenTracing baggage
- Dynamic routing overrides
  - ➔ Custom filter + ORIGINAL\_DST

# Offloaded pods: EDS exclusion

-o-  add in final endpoint; ready to test e2e ● 13c13a9

 **matthewgrossman** commented 1 minute ago Author 😊 ...

/offload

 **offloaded-deploy-bot** bot commented 1 minute ago 😊 ...

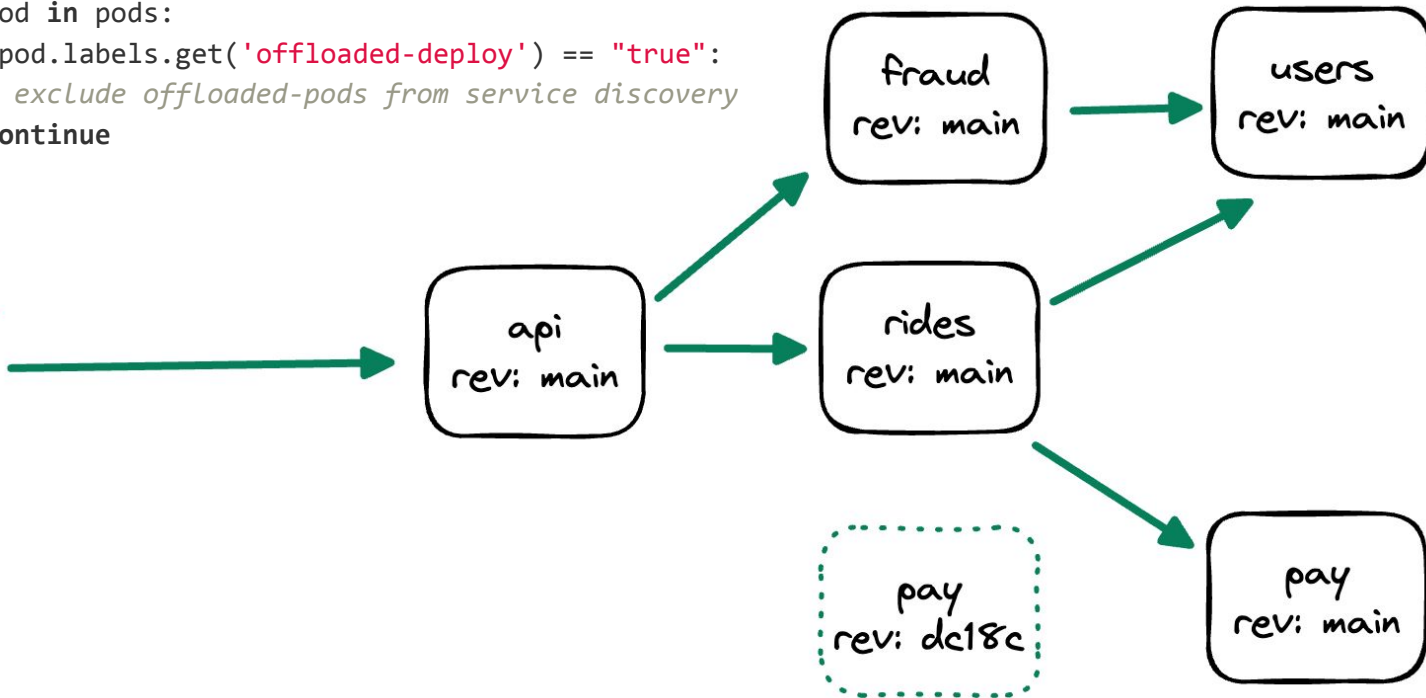
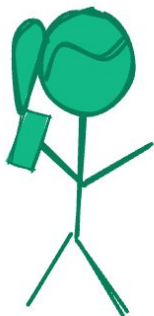
Success! Offloaded deploy for SHA [13c13a9](#) staging will execute shortly.  
You can monitor the deploy progress at <https://deploys.lyft.net/exampleservice/646367>




```
...  
app=foo  
environment=staging  
offloaded-deploy=true  
...
```

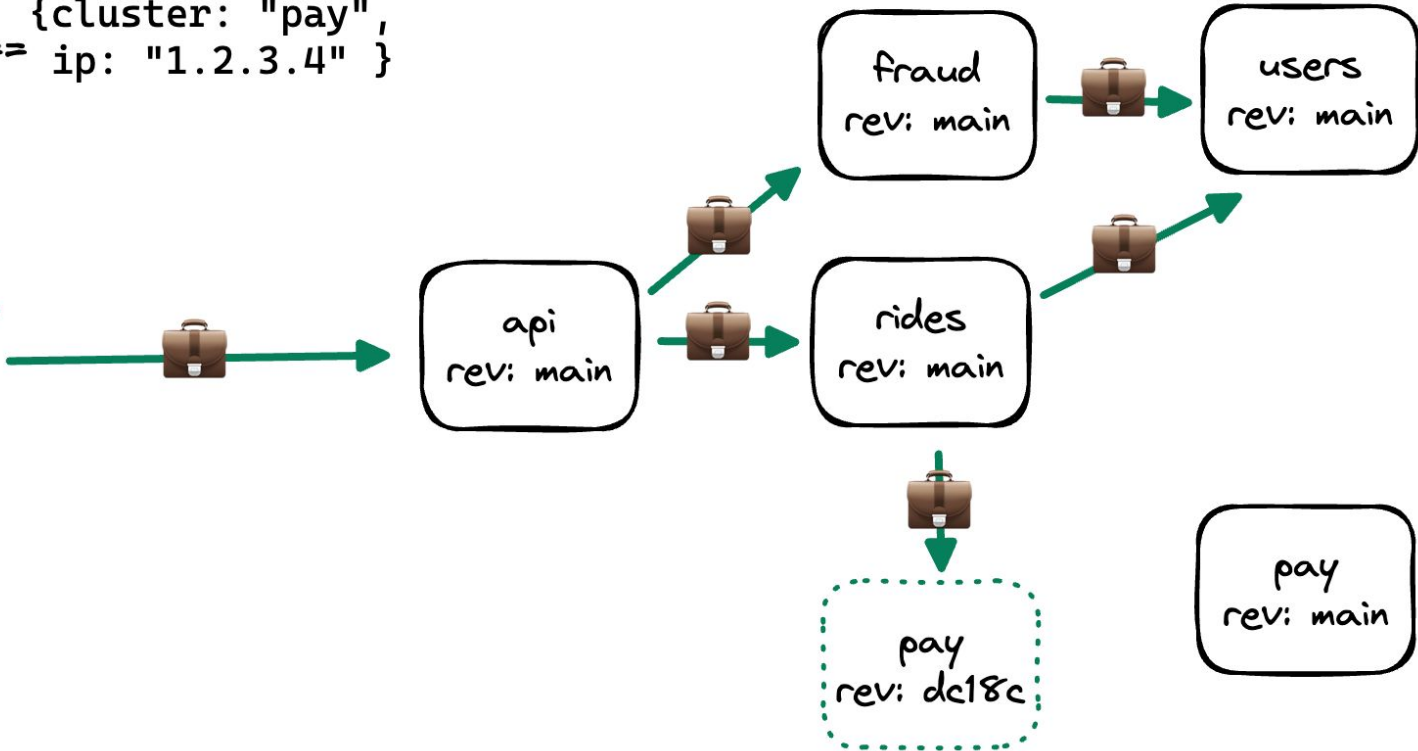
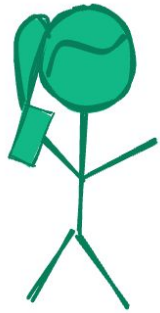
# Offloaded pod deploy

```
for pod in pods:  
    if pod.labels.get('offloaded-deploy') == "true":  
        # exclude offloaded-pods from service discovery  
        continue
```



# With override metadata

 `{cluster: "pay",  
== ip: "1.2.3.4" }`

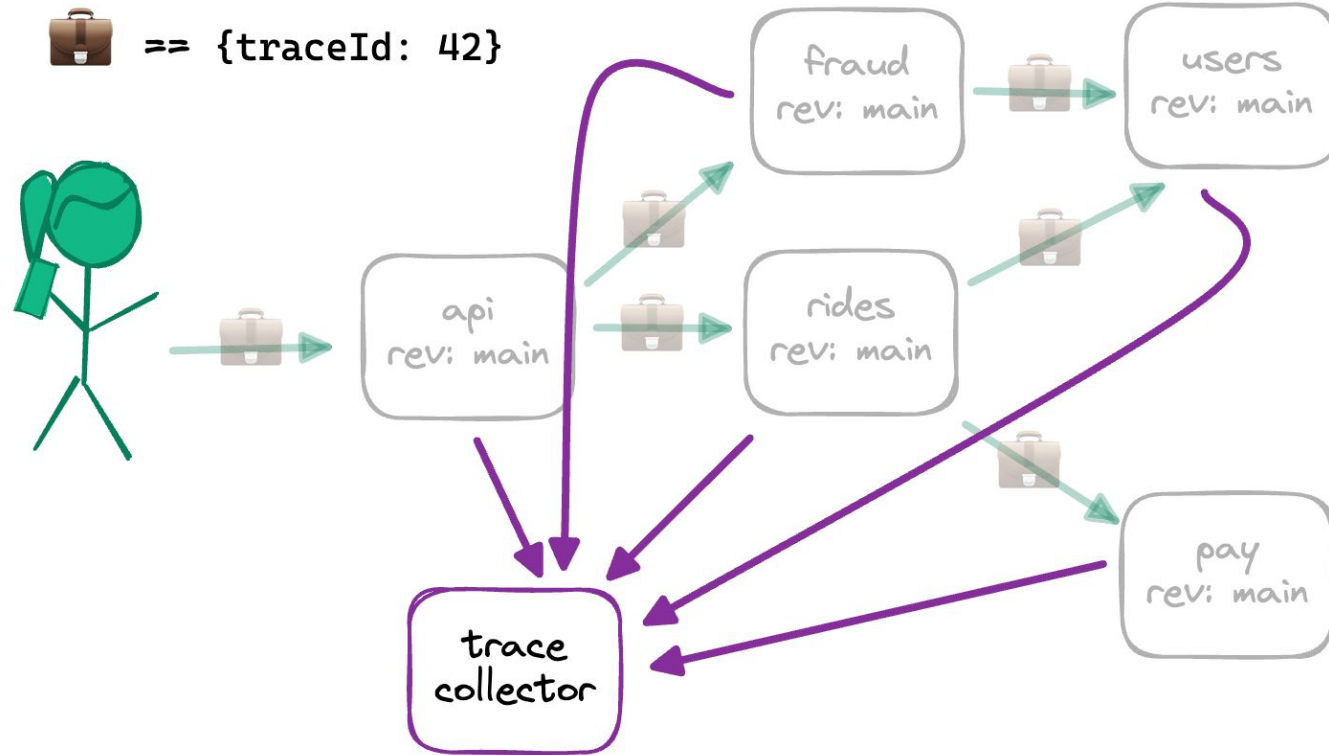


# Override metadata

```
{
  "envoy_overrides": [
    { "cluster_name": "pay",
      "ip_address": "10.0.0.42:8080"
    },
    { "cluster_name": "api",
      "ip_address": "5.4.3.2:4444"
    },
  ],
}
```



# Distributed tracing



# Distributed tracing

- Propagated trace data helps clarify complex flows



GET /v1/rides - 4s

GET /v1/user/x - 3s

SELECT ... FROM userdb - 2.5s

SELECT ... FROM ridesdb - 2s

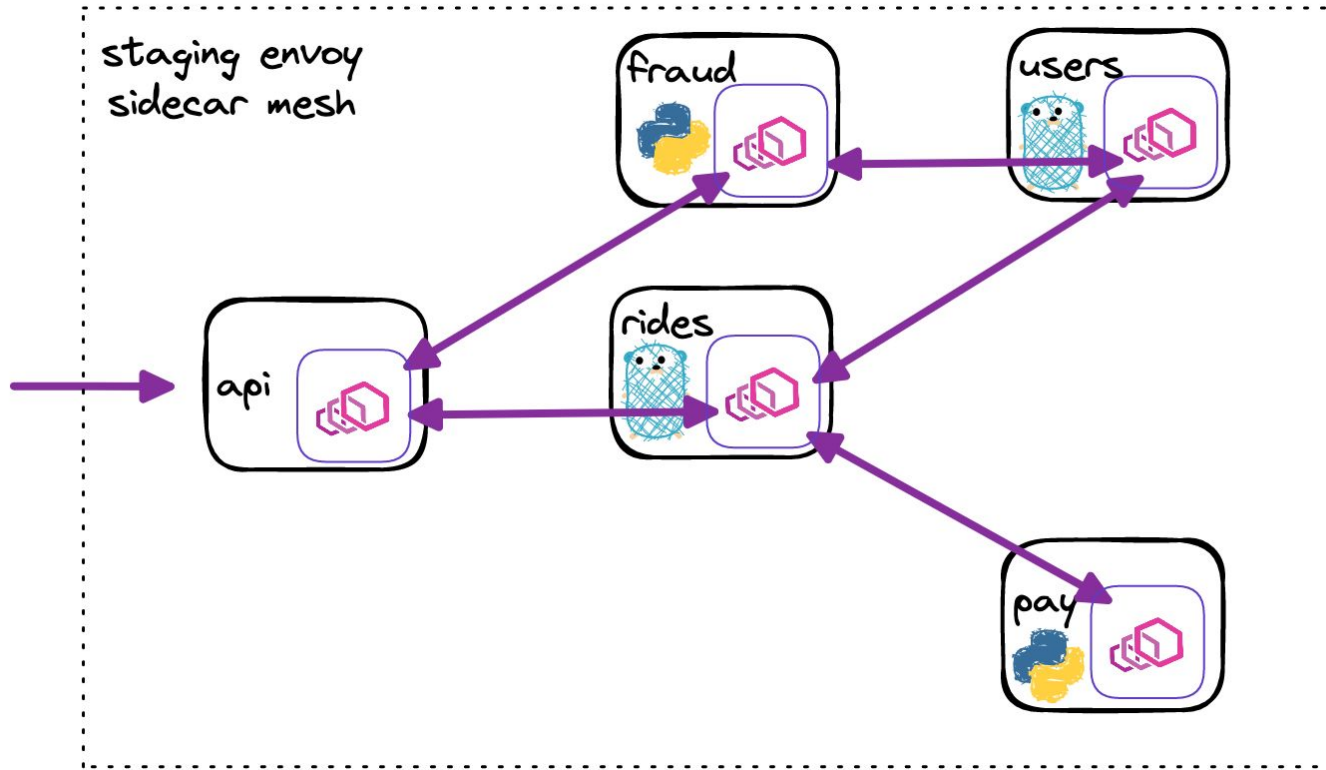
process data - 1s

# OpenTracing (OpenTelemetry)

- We already propagate this header (“context propagation”)
- OpenTracing header embeds “baggage”, which contains arbitrary `key:value` pairs

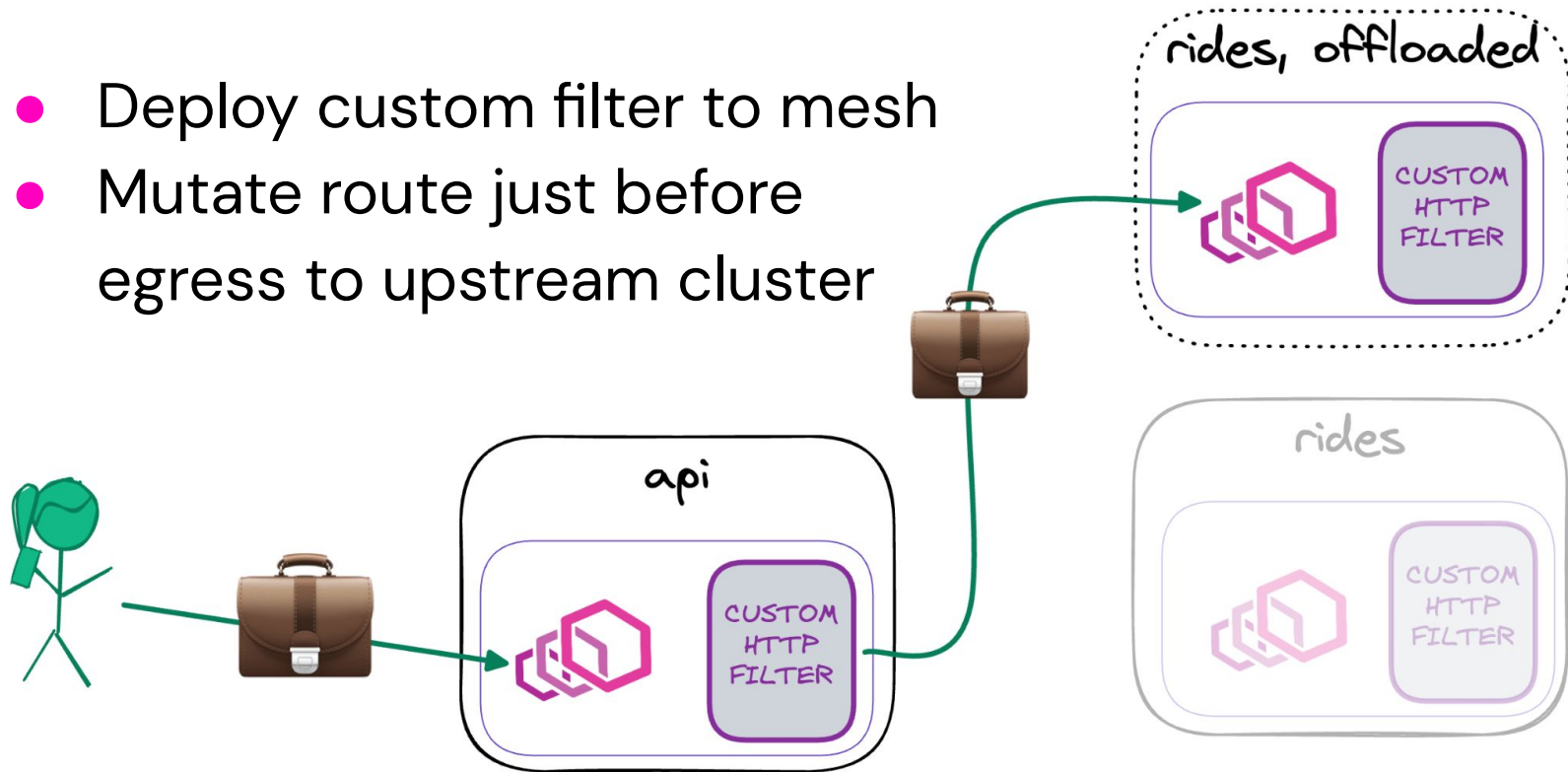
# Envoy filters

# Envoy @ Lyft



# HTTP filters

- Deploy custom filter to mesh
- Mutate route just before egress to upstream cluster



# Filter logic

- Extract overrides (SVC:IP pairs) from tracing baggage
- See if any overrides match where we “should” be going
- If so, forward to override IP instead

# Implementation

```
Http::FilterHeadersStatus OverridesFilter::decodeHeaders(Http::RequestHeaderMap& headers, bool) {
    // Get the baggage from the active_span, decode, and convert to our format
    const std::string baggage = decoder_callbacks_->activeSpan().getBaggage("overrides");
    RequestContext request_context;
    request_context.ParseFromString(baggage)
    // Iterate through potential overrides
    for (auto field : request_context.envoy_overrides()) {
        if (field.cluster_name() == cached_cluster_name) {
            // Wrap the real route in a subclass of Router::DelegatingRoute
            auto route_override = std::make_shared<OverrideDelegatingRoute>(
                // DelegatingRoute "delegates" all calls by default to the wrapped
                // route, in this case a ptr to the current route
                decoder_callbacks_->route(),
                // Our subclass always returns our chosen cluster for
                // route_override.routeEntry().clusterName()
                controller_->originalDstClusterName(),
            );
            // Set our new route as the chosen route
            decoder_callbacks_->setRoute(route_override);

            // Set ip:port pair to the ORIGINAL_DST header
            headers.setReferenceKey(Http::Headers::get().EnvoyOriginalDstHost, field.ip_address());
            break;
        }
    }
    return Http::FilterHeadersStatus::Continue;
}
```



# Extract override data

```
Http::FilterHeadersStatus OverridesFilter::decodeHeaders(Http::RequestHeaderMap&
headers, bool) {
    // Get the baggage from the active_span, decode, and convert to our format
    const std::string baggage =
        decoder_callbacks_->activeSpan().getBaggage("overrides");
    RequestContext request_context;
    request_context.ParseFromString(baggage)
    // Iterate through potential overrides
    for (auto field : request_context.envoy_overrides()) {
```

# Modify the route

```
// Iterate through potential overrides
for (auto field : request_context.envoy_overrides()) {
    if (field.cluster_name() == cached_cluster_name) {
        // Wrap the real route in a subclass of Router::DelegatingRoute
        auto route_override = std::make_shared<OverrideDelegatingRoute>(
            // DelegatingRoute "delegates" all calls by default to the wrapped
            // route, in this case a ptr to the current route
            decoder_callbacks_->route(),
            controller_->originalDstClusterName(),
        );
        // Set our new route as the chosen route
        decoder_callbacks_->setRoute(route_override);
    }
}
```

# Original Destination Cluster

clusters:

...

```
- name: context_propagation_cluster
  connect_timeout: 1s
  type: ORIGINAL_DST
  lb_policy: CLUSTER_PROVIDED
  original_dst_lb_config:
    use_http_header: true
```

```
$ curl -H 'x-envoy-original-dst-host: 10.0.0.42:8080' ...
```

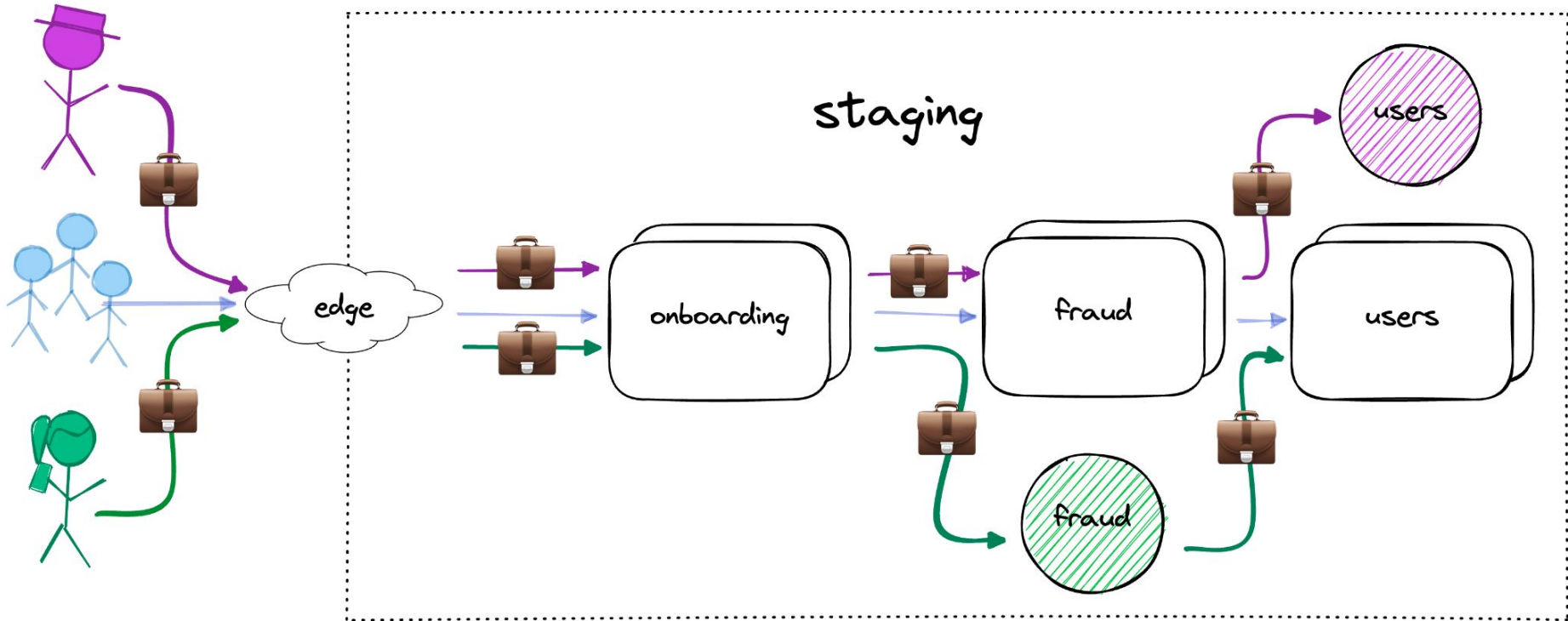
# Set Original Destination Cluster

```
    decoder_callbacks_->route(),
    // Our subclass always returns our chosen cluster for
    // route_override.routeEntry().clusterName()
    controller_->originalDstClusterName(),
);
decoder_callbacks_->setRoute(route_override);
// Set ip:port pair to the ORIGINAL_DST header
headers.setReferenceKey(Http::Headers::get().EnvoyOriginalDstHost,
    field.ip_address());
}
}
return Http::FilterHeadersStatus::Continue;
}
```

# Summary

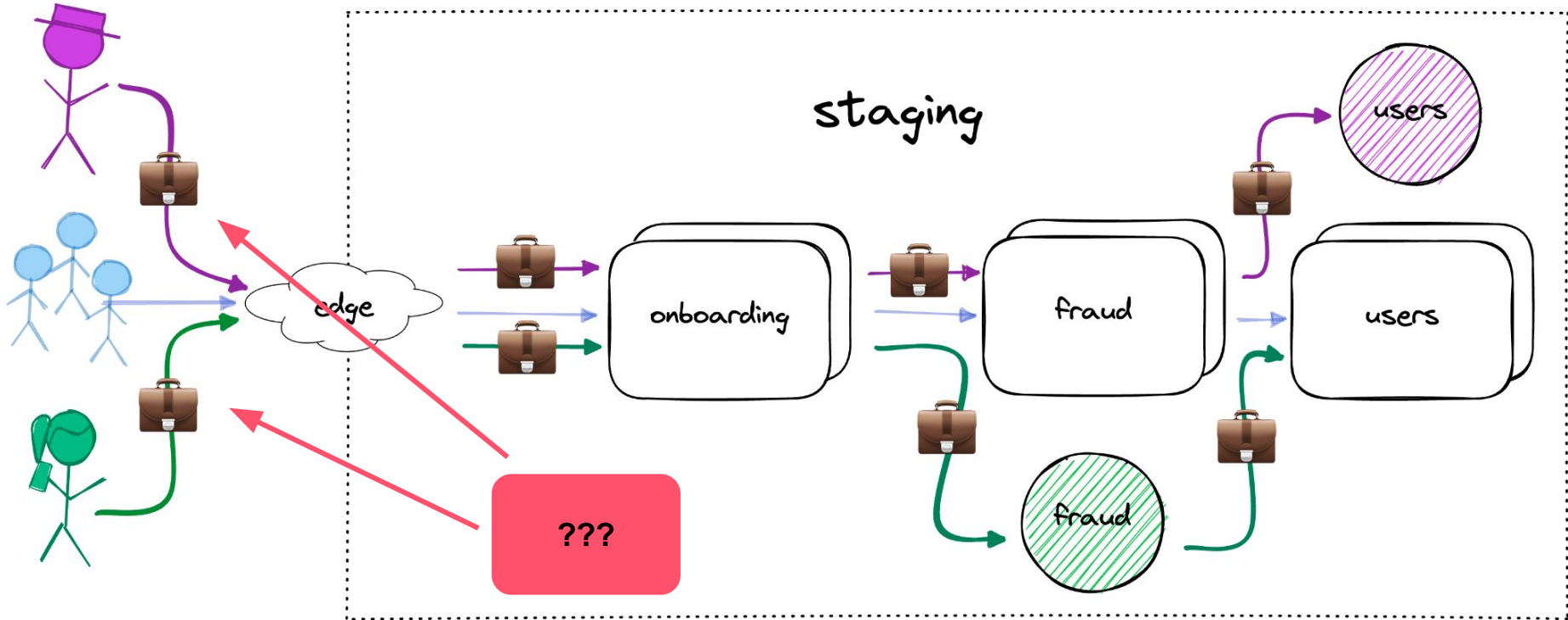
- Extract overrides from baggage from tracing header
- Wrap chosen route and set `clusterName()` to `original_dst`
- `setRoute()` to the wrapped route
- Add IP address to `x-envoy-original-dst-host` header

# Staging Overrides v1



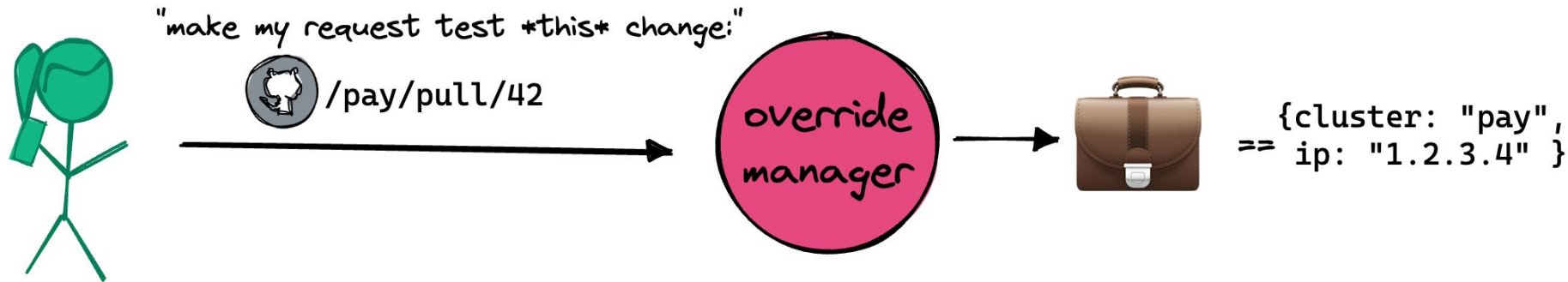
# Extending overrides

# Baggage-attachment tooling





# Need to map branch → baggage



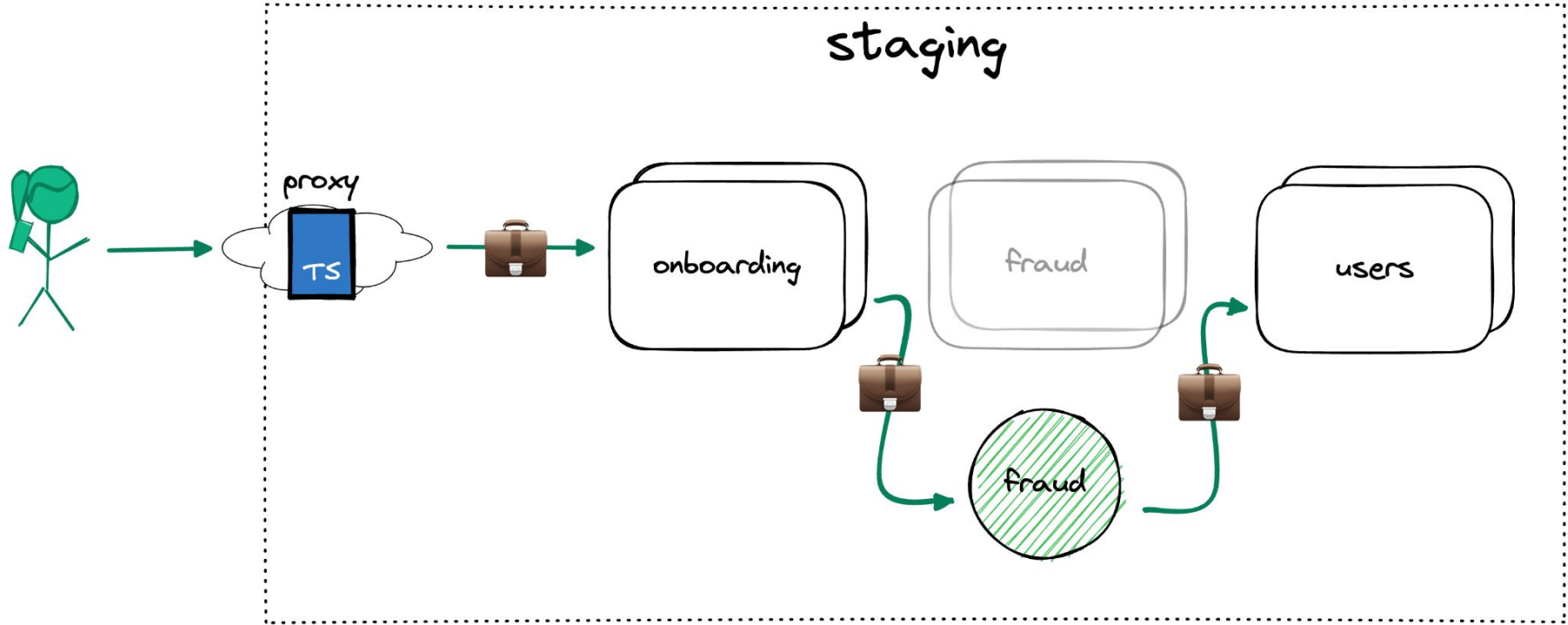
- ***drivers-4242***.dev.lyft.net/api/routes
- curl -H 'x-dev-id: ***drivers-4242***' ...
- Send it through a proxy

# Our scriptable ingress proxy

- Originally for mobile engineers to mock BEs
- Typescript scriptable to modify req/resp

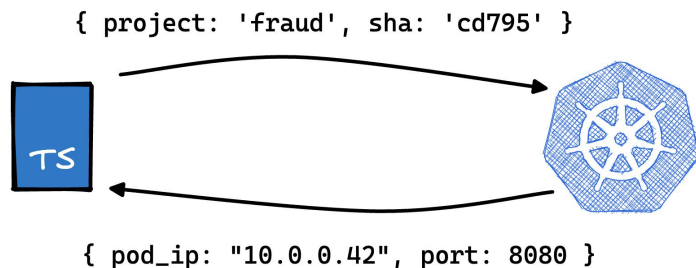
```
3 ✓ get('/pyexample37/api/hello', async (ctx) => {  
4   |   ctx.request.headers['user-agent'] = 'test user agent'  
5   |   await ctx.fetchApiResponse()  
6   |   ctx.response.body['msg2'] = 'Hello, envoycon!'  
7   | })
```

# How users actually make requests

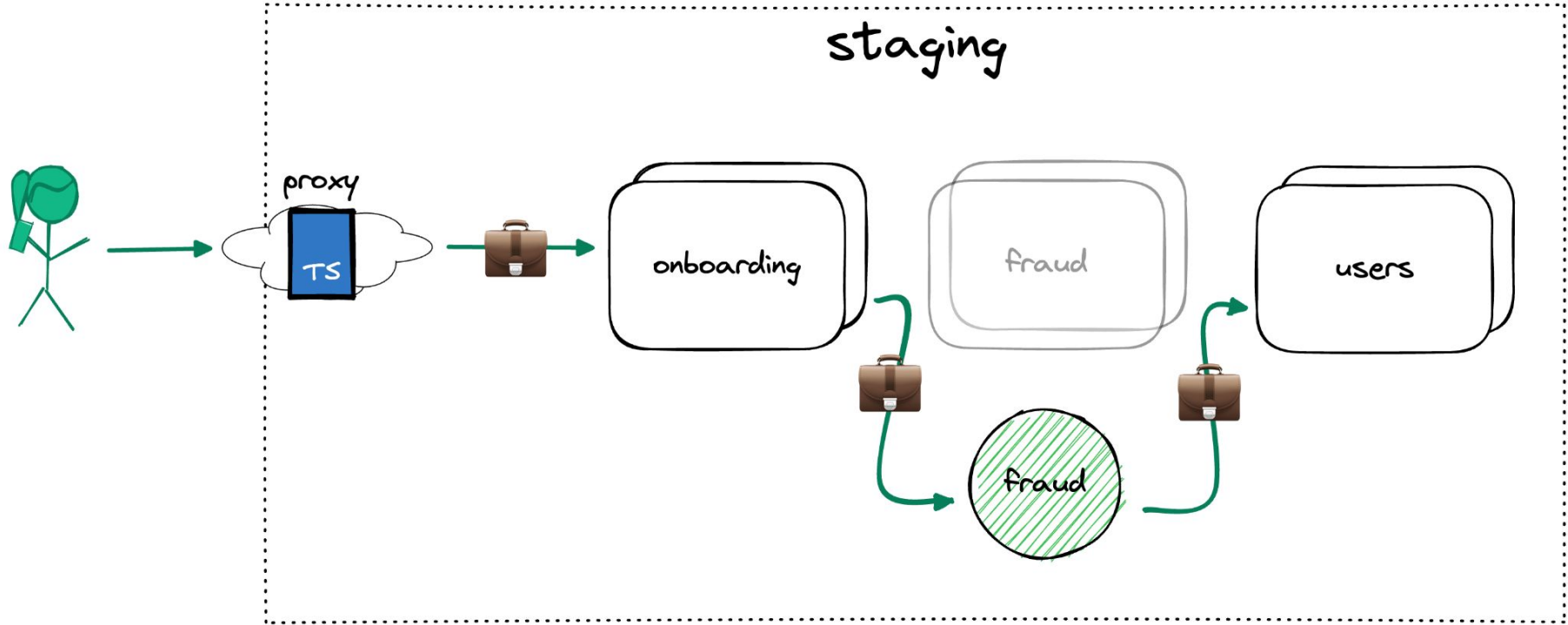


# Custom typescript proxy

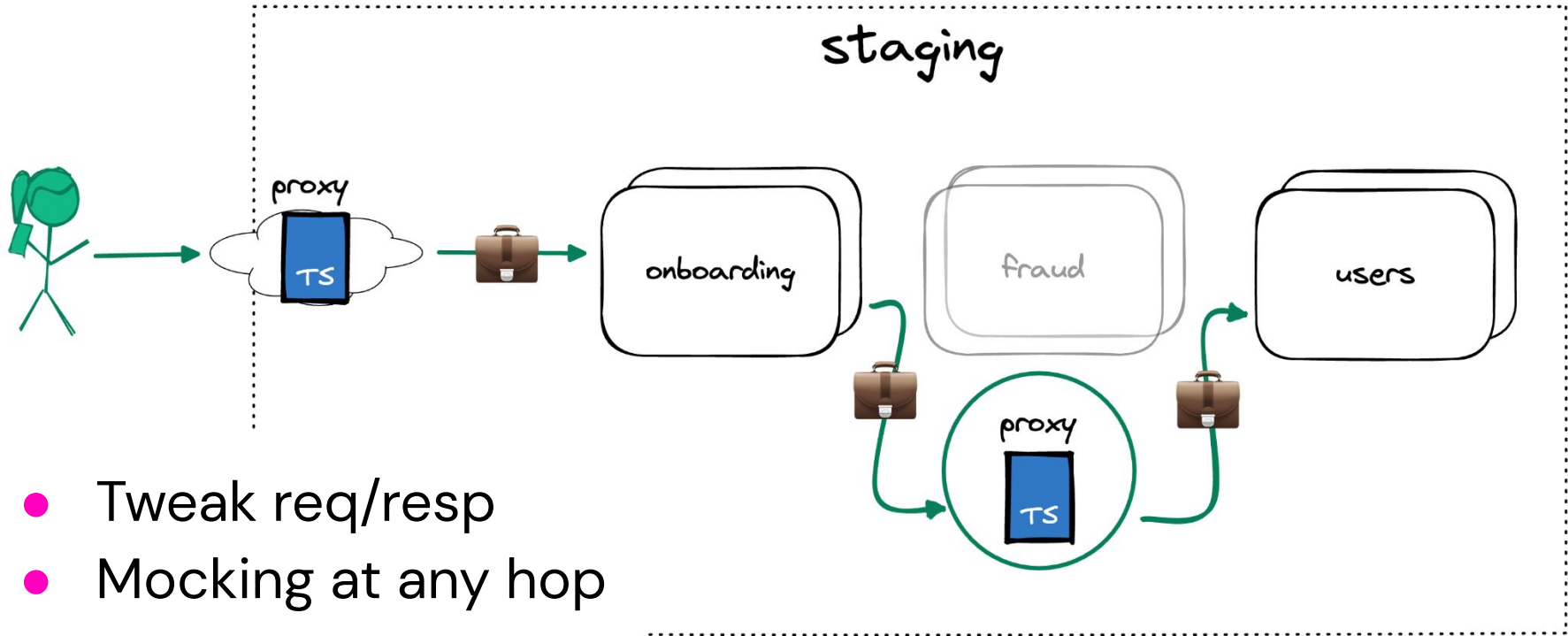
```
3 ✓ get('/pyexample37/api/hello', async (ctx) => {
4   |   ctx.request.headers['user-agent'] = 'test user agent'
5   |   await ctx.fetchApiResponse()
6   |   ctx.response.body['msg2'] = 'Hello, envoycon!'
7   | })
8
9 ✓ use(
10 ✓   setEnvoyOverride({
11   |     project: 'fraud',
12   |     sha: 'cd795',
13   |   })
14 )
```



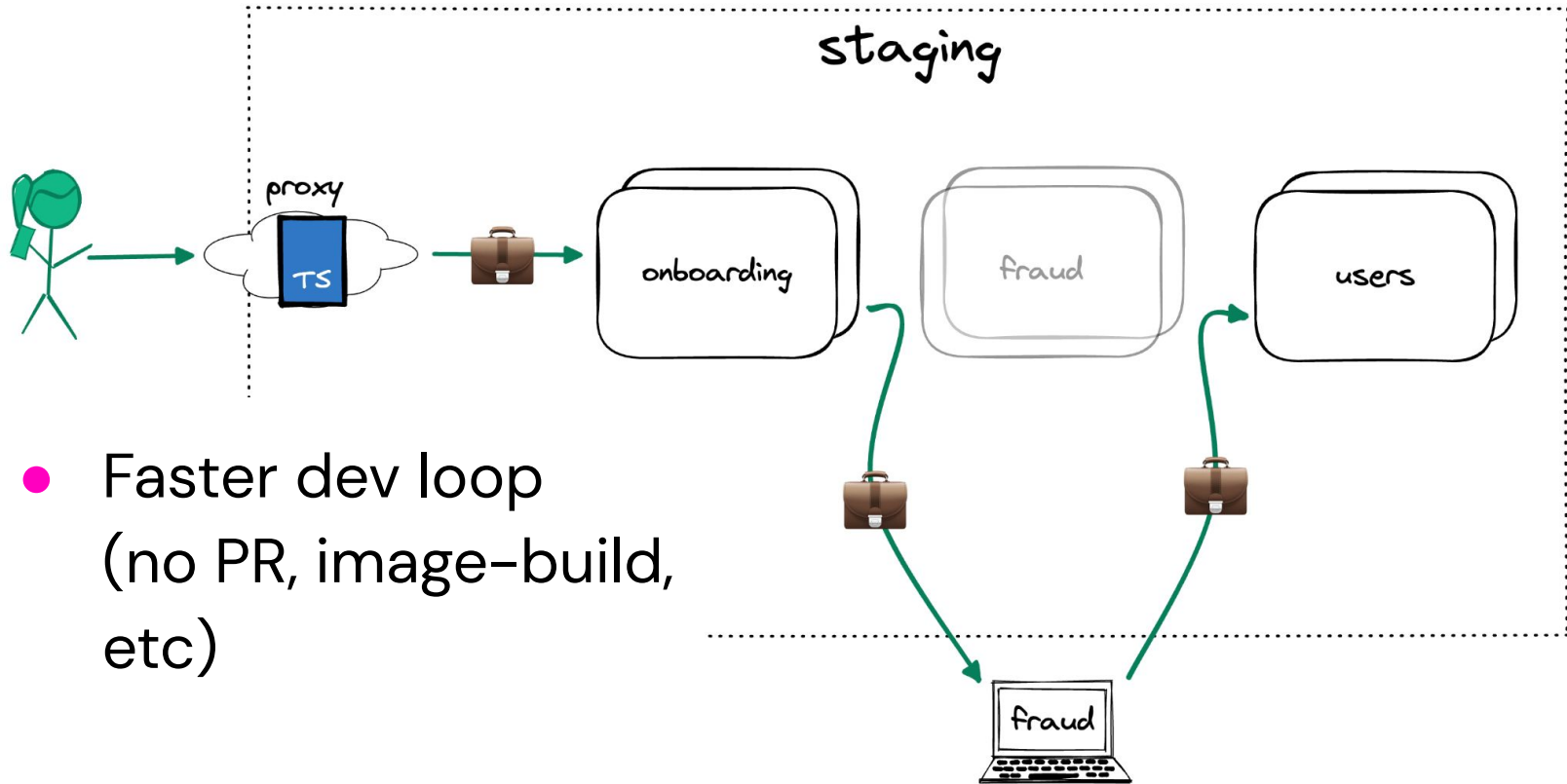
# What else can we do?



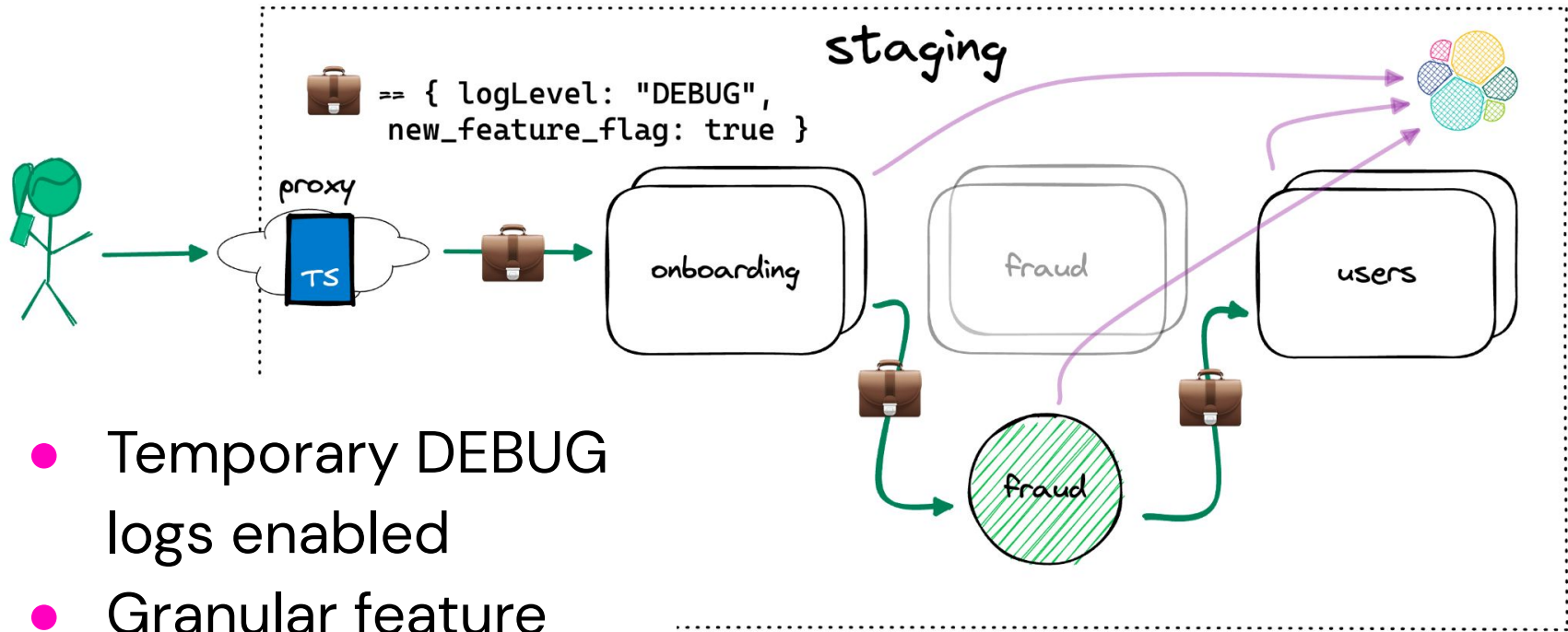
# What if we put our proxy into the mesh



# What if we redirect to traffic to laptops



# What if we add more controls to baggage?



- Temporary DEBUG logs enabled
- Granular feature flags



# Long-term vision for staging overrides

- Give complete control over request flow at every hop
- Isolate requests to allow reuse of shared environments
- Provide local+cluster tooling to give devs visibility and understanding

# Conclusion

# Results

- **Provisioning:** 1hr w/ onebox, to 10min with staging overrides
- **Parity:** Infra parity and functional parity lead to fewer surprises
- **New framework:** superpowers at every hop; just getting started

# Challenges / retro

- Context propagation (library updates)
- Data isolation (stats, DBs)
- General new paradigm / teaching
- If redone, could we utilize other tech?  
(telepresence?)

**Thanks!**

Matt Grossman

[matt@mrgrossman.com](mailto:matt@mrgrossman.com)