
KubeCon
CloudNativeCon
North America 2019

Beyond Getting Started: Using OpenTelemetry to its Full Potential

Sergey Kanzhelev & *Morgan McLean,*
Microsoft *Google*

Who we are

Sergey Kanzhelev
SWE at Microsoft

Morgan McLean
PM at Google

OpenTelemetry

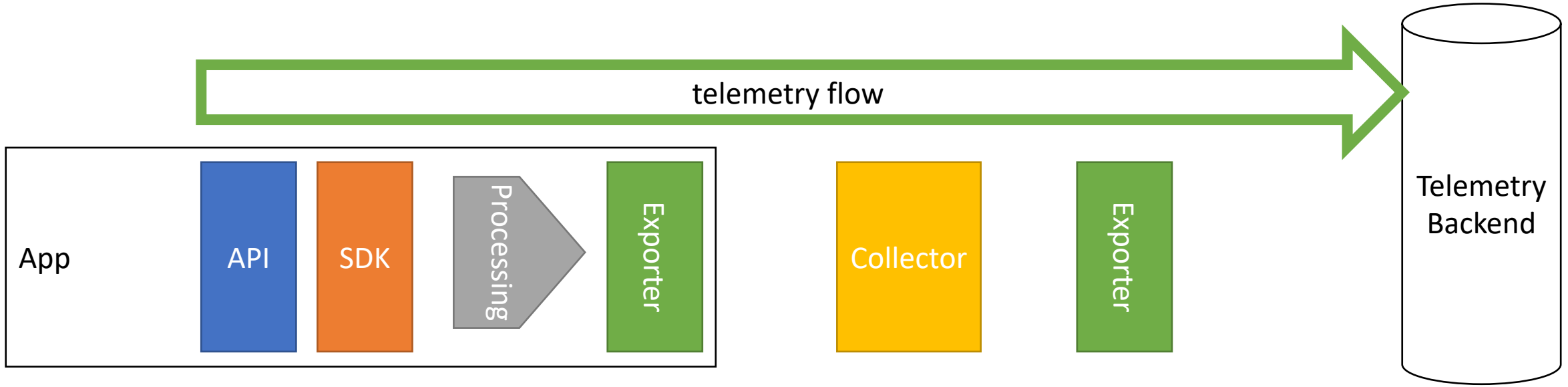
OpenTelemetry makes robust, portable telemetry a built-in feature of cloud-native software.

OpenTelemetry

APIs Integrations Libraries Exporters Collectors

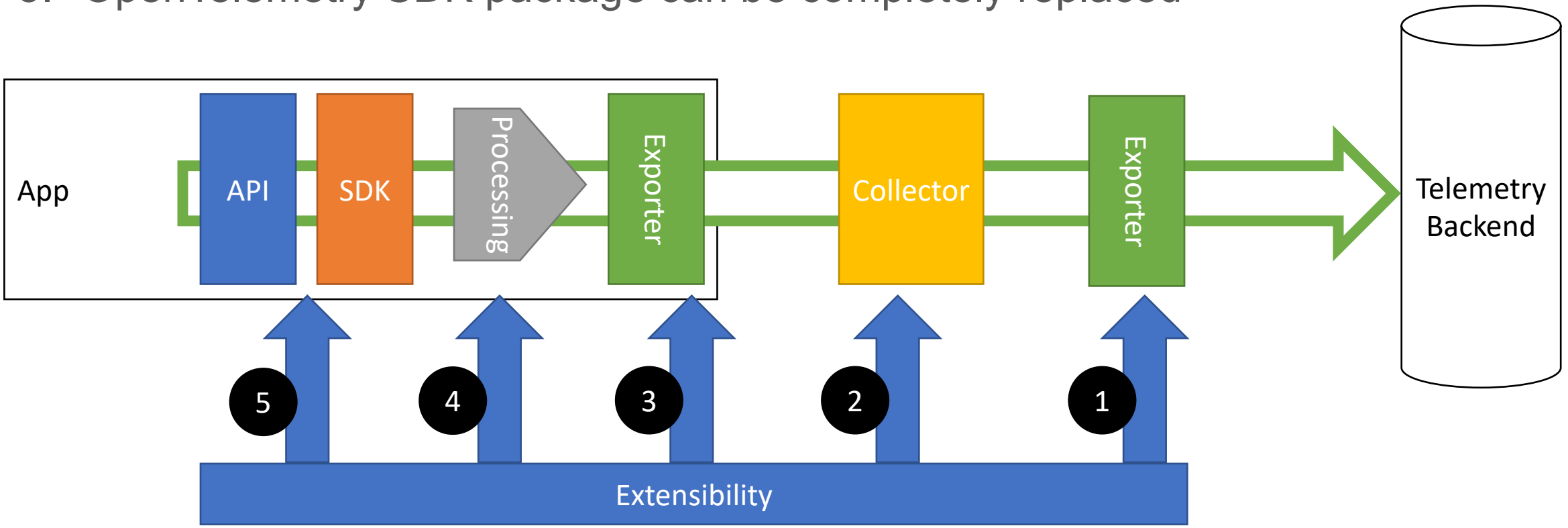
OpenTelemetry architecture

OpenTelemetry is a complete solution for your telemetry collection needs:



OpenTelemetry architecture

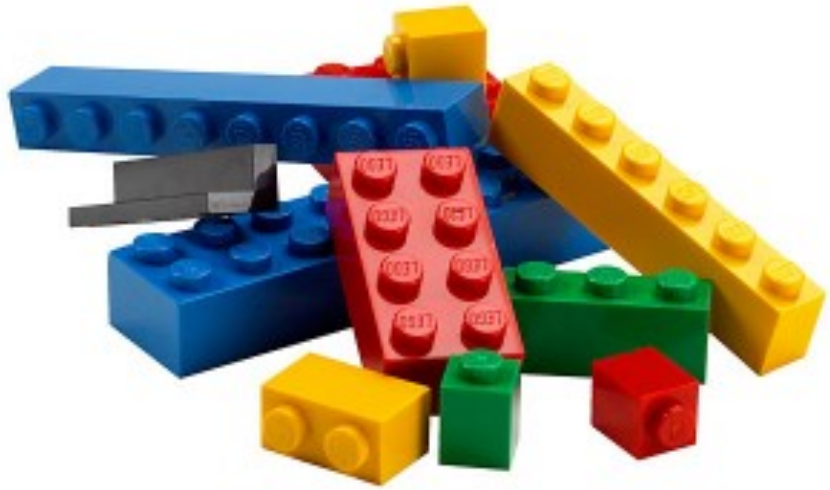
1. Collectors can communicate with various backends via exporters
2. Configuration controls aggregation, batching, and processing
3. In-proc exporters are easily replaceable to work with different backend
4. SDK allows various extensions: sampling, filtering, enrichments
5. OpenTelemetry SDK package can be completely replaced



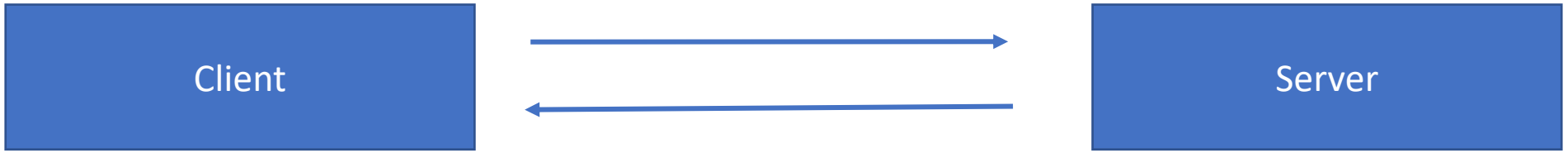
OpenTelemetry API surfaces

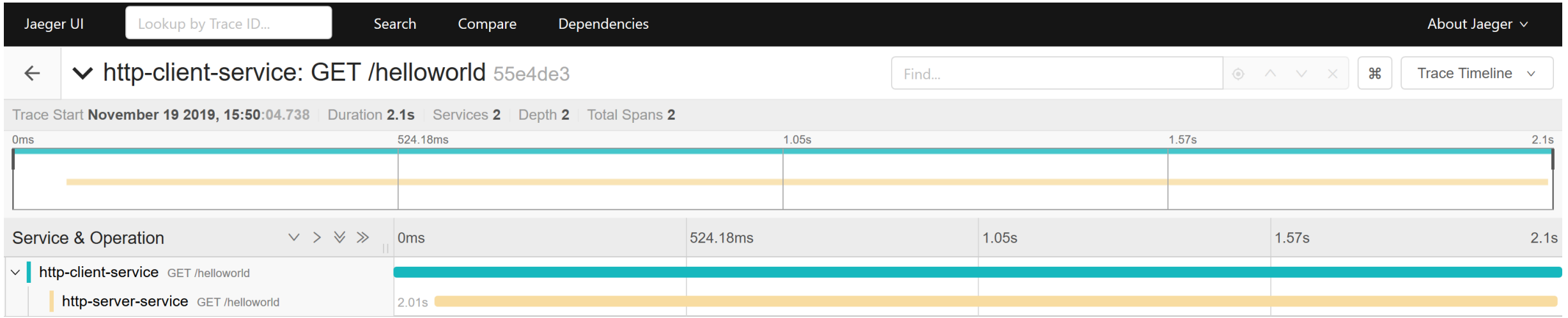
OpenTelemetry has four API surfaces:

- Configuration of SDK
- API for code instrumentation
- Processing and enriching of telemetry
- Exporters development



Getting started





KubeCon | CloudNativeCon

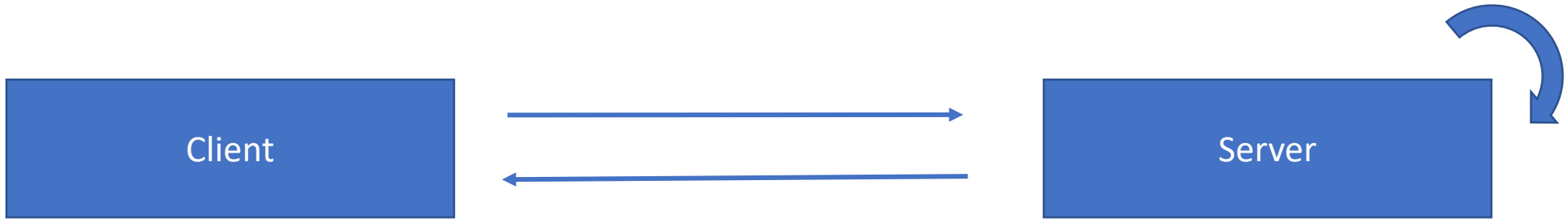
North America 2019

Beyond Getting Started: Using OpenTelemetry to Its Full Potential

Sergey Kanzhelev & *Morgan McLean*

Microsoft *Google*

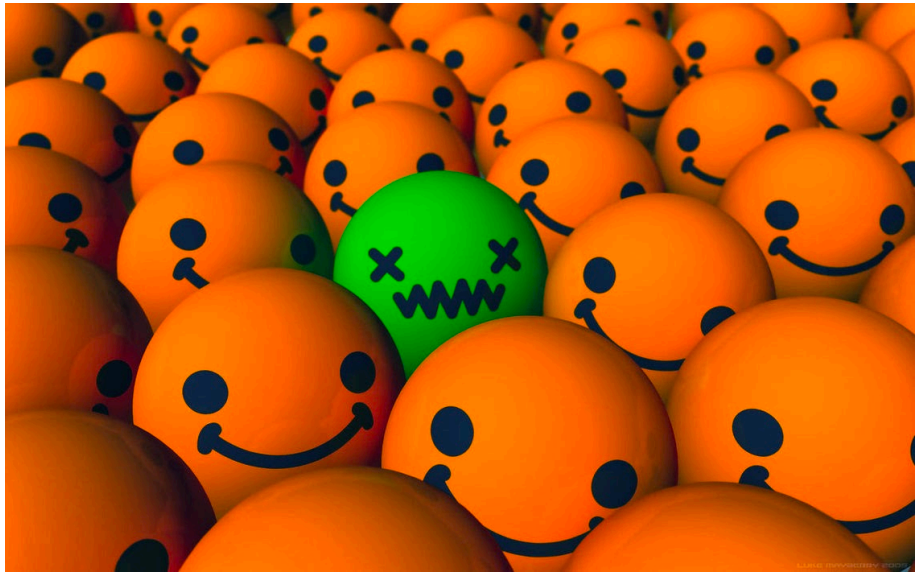
Long-running tasks



Basic sampling

Synthetic traffic may hide the real user problems.

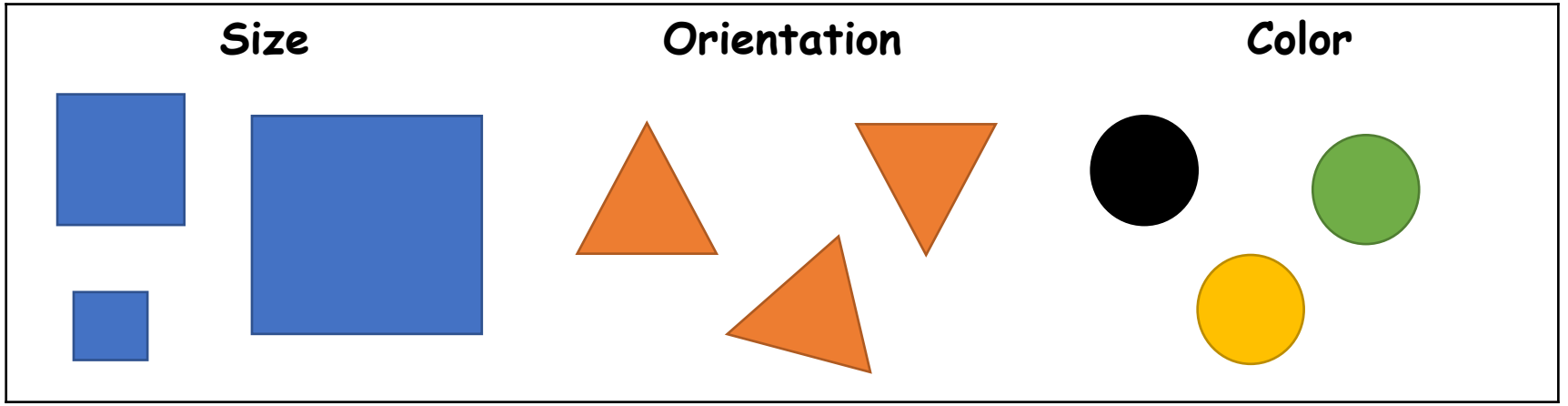
Use custom Sampler to filter out synthetic traffic like the calls to `"/health"` endpoint.



Custom attributes

Add custom properties for easier querying and differentiating telemetry.

Some ideas of custom attributes:
- Business details: productID, logical operation name
- User session attributes: free tier/paid customer, user anonymized id
- Capture values from http headers



Resource API

Your app is deployed in different environments. Environment name is a very important custom attribute that will be used to slice the telemetry.

The resource API is used to define resource attributes, which are distinct from regular attributes

- Deployment name and location
- App name and version
- Hosting environment

Custom attributes as a dimensions

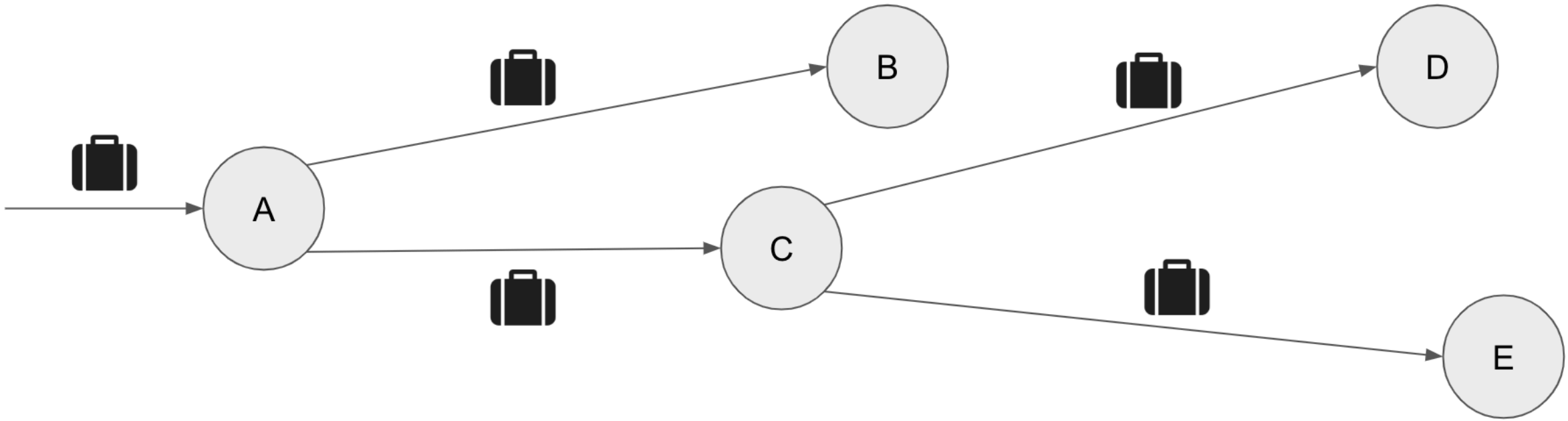
When app is using A/B testing and feature “flights”. Telemetry should be attributed with this FlightID.

Not a simple telemetry attribution as you’d typically also need to configure separate metrics dimension and potentially have a better sampling logic accounting for those attributes.

Propagation of custom attributes

"FlightID" propagation across components:

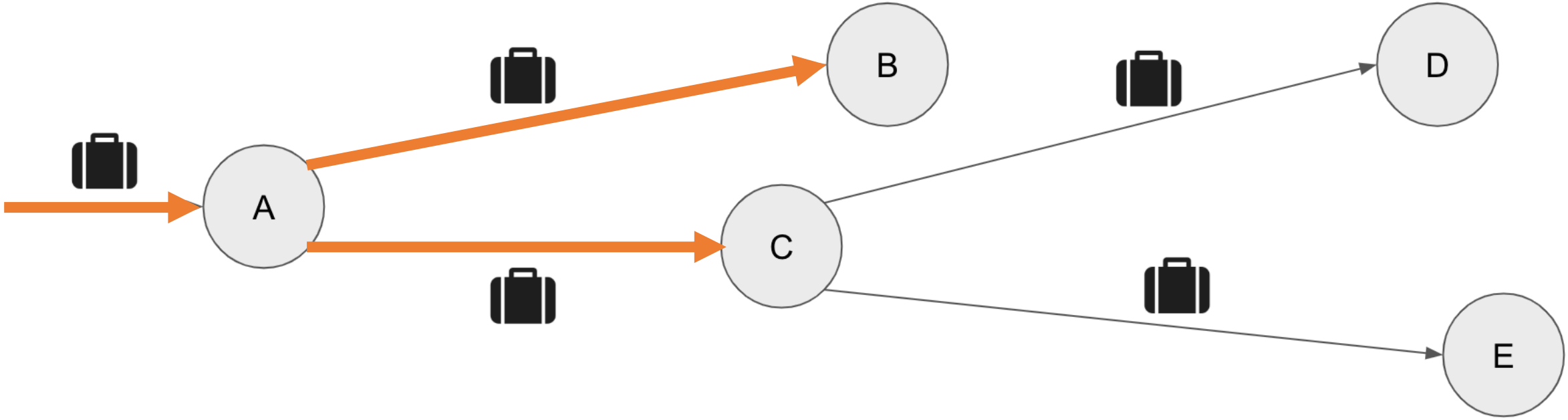
- use it as a metrics dimension or
- attribute spans



Context propagation

When custom RPC is used, you will need a custom propagation mechanism implemented.

OpenTelemetry helps with propagation, but for custom protocols it must be propagated using propagation API.



Instrumentation API

Creating integrations by instrumenting shared code (storage clients, RPC libraries, etc.) is why OpenTelemetry exists!

You have two choices:

1. Build an OpenTelemetry integration that hooks into callbacks or performance APIs provided by the client

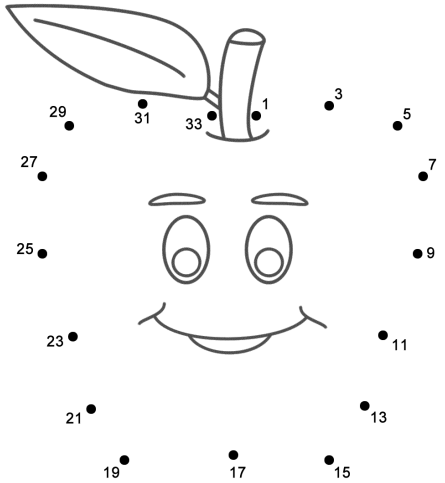
2. Instrument the shared code with OpenTelemetry APIs

#2 is preferred: it's generally more performant and doesn't break when clients are updated

IsRecording?

If SDK was NOT enabled, nothing needs to be captured:

- Always propagate the context




```
using (_tracer.StartActiveSpan("Execute", SpanKind.Client, out var span))
{
    if (span.IsRecording)
    {
        span.AddAttribute("state”, this.CalculateState());
    }

    _tracer.TextFormat.Inject(
        span.Context,
        restObj,
        (restObj, k, v) => restObj.Metadata[k] = v);

    restObj.Execute();
}
```

Named tracers

OpenTelemetry uses named tracers
- Improves data visualization and analysis
- Save costs by disabling tracers
- Simplifies troubleshooting


```
private readonly ITracer _tracer;

public MyClientLibrary()
{
    _tracer = TracerFactoryBase.Default
        .GetTracer("MyClientLibrary", version);
}
```

Metrics

Metrics and distributed traces are coming together.

Use metrics:
- Not affected by sampling
- Lightweight as semantics is easier
- Aggregation dimensions can be decided on later

```
var meter = MeterFactoryBase.Default.GetMeter("MyClientLibrary", version);
var reqCount = meter.CreateLongCounter("requests count");

reqCount.Add(DistributedContext.Current, 1,
meter.GetLabelSet(new Dictionary<string, string>() { {"success", "true" } }));
```

Performance best practices

The art of instrumenting for telemetry: just enough telemetry for the price

1. Only create spans for longer-running tasks that are worth tracking,
2. Don’t create spans for every function call!
3. Use time event to indicate event occurrence vs. child span
4. Use smart defaults and allow to configure additional details collection

Tell us your scenarios

We want to know more about our users! OpenTelemetry doesn't report analytics back to us, so we only know about your experience if you tell us

Tell us about your scenarios:

- What environments you use it
- How do you use it, what do you like the most
- What's missing

Reach out to us via:

- Gitter: https://gitter.im/open-telemetry/community
- GitHub: https://github.com/open-telemetry/community
- E-mails: cncf-opentelemetry-community@lists.cncf.io
- SIG and community meetings: calendar

Get involved

https://opentelemetry.io

Come to out maintainers track session:

Thursday, November 21 • 10:55am - 12:25pm

OpenTelemetry: The First Release, What's Next, and How to Get Involved

Chris Kleinknecht, Google, Morgan McLean, Google; Sergey Kanzhelev, Microsoft; Tristan Sloughter, Postmates;

https://sched.co/Uake