Selium

Be productive in minutes.
Not days.

What is it?

- A messaging platform
- It’s composable
- It’s written in Rust

Why?

CONFLUENT

Apache ACTIVEMQ

RabbitMQ

ØMQ

kafka

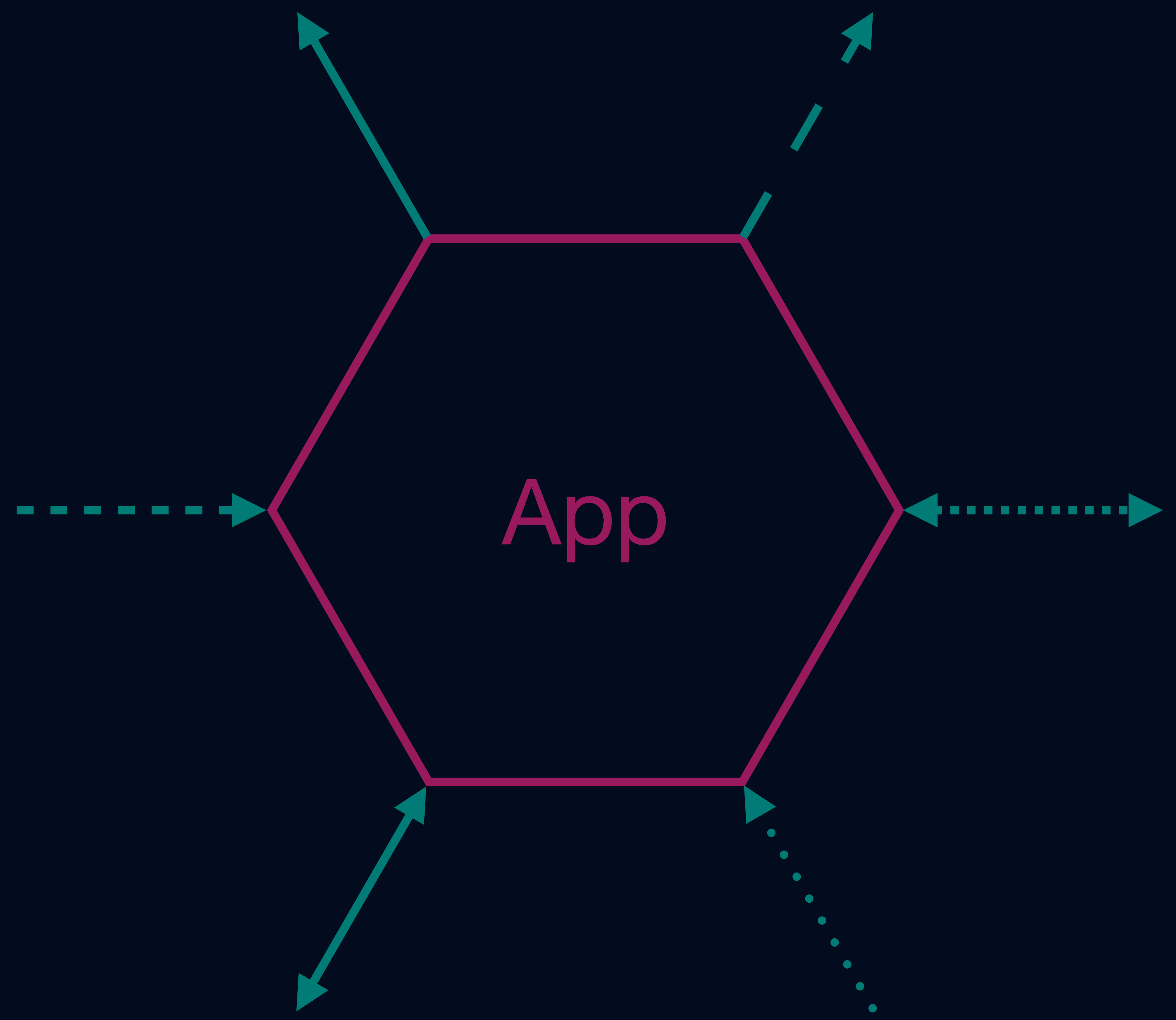
amazon KINESIS

IBM MQ

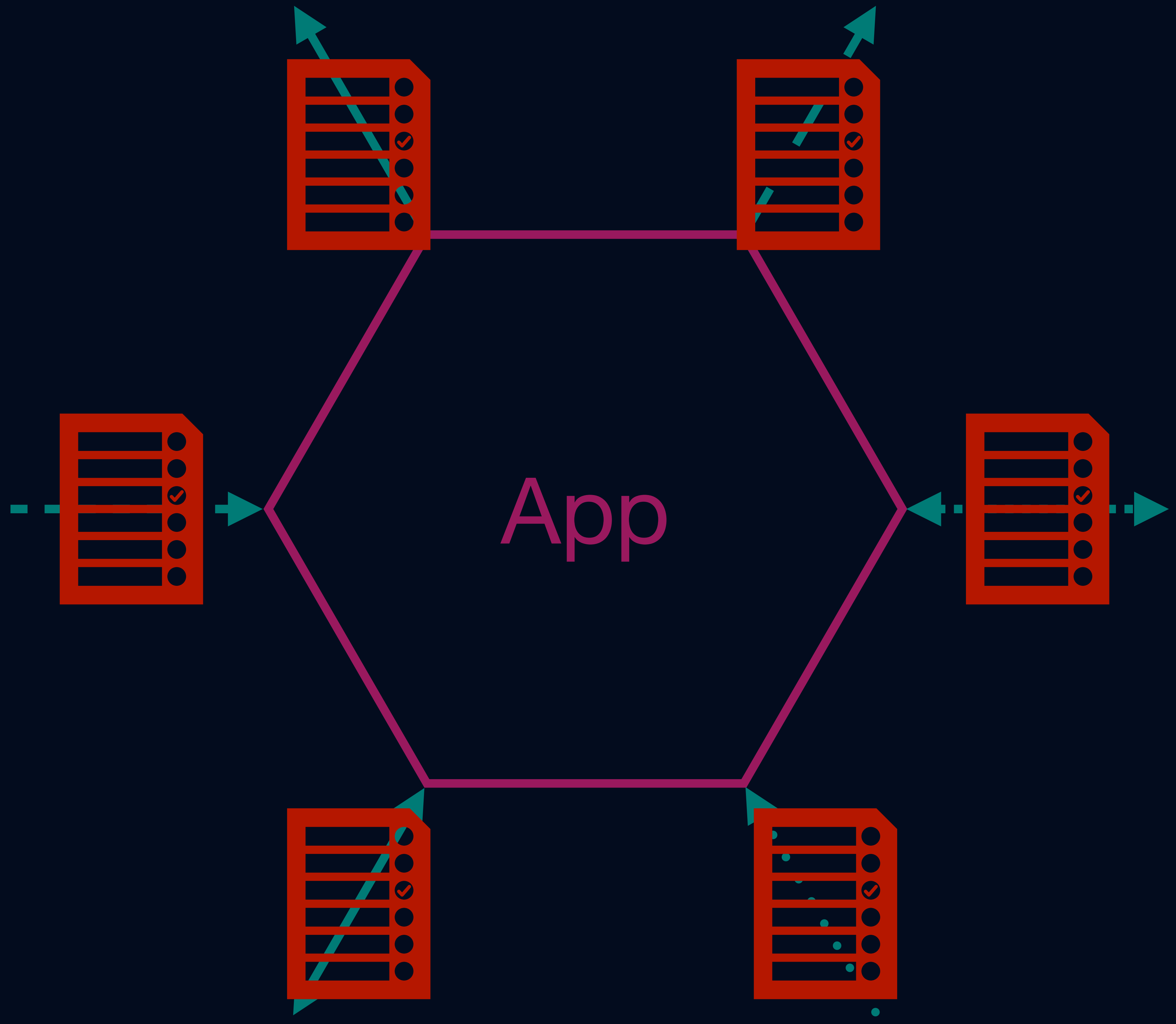
redis

Google Cloud Pub/Sub

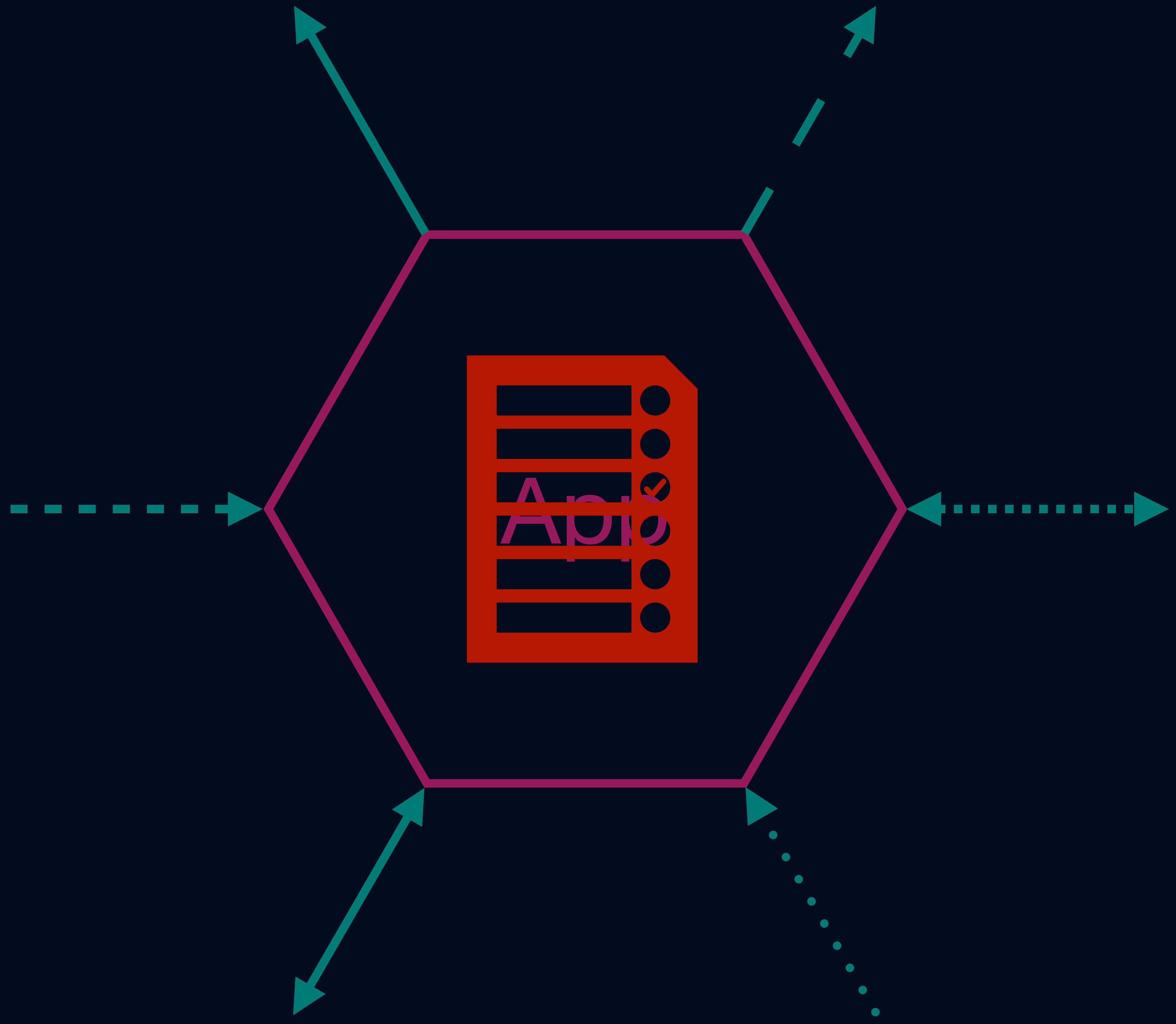
Apps have different types of communication.



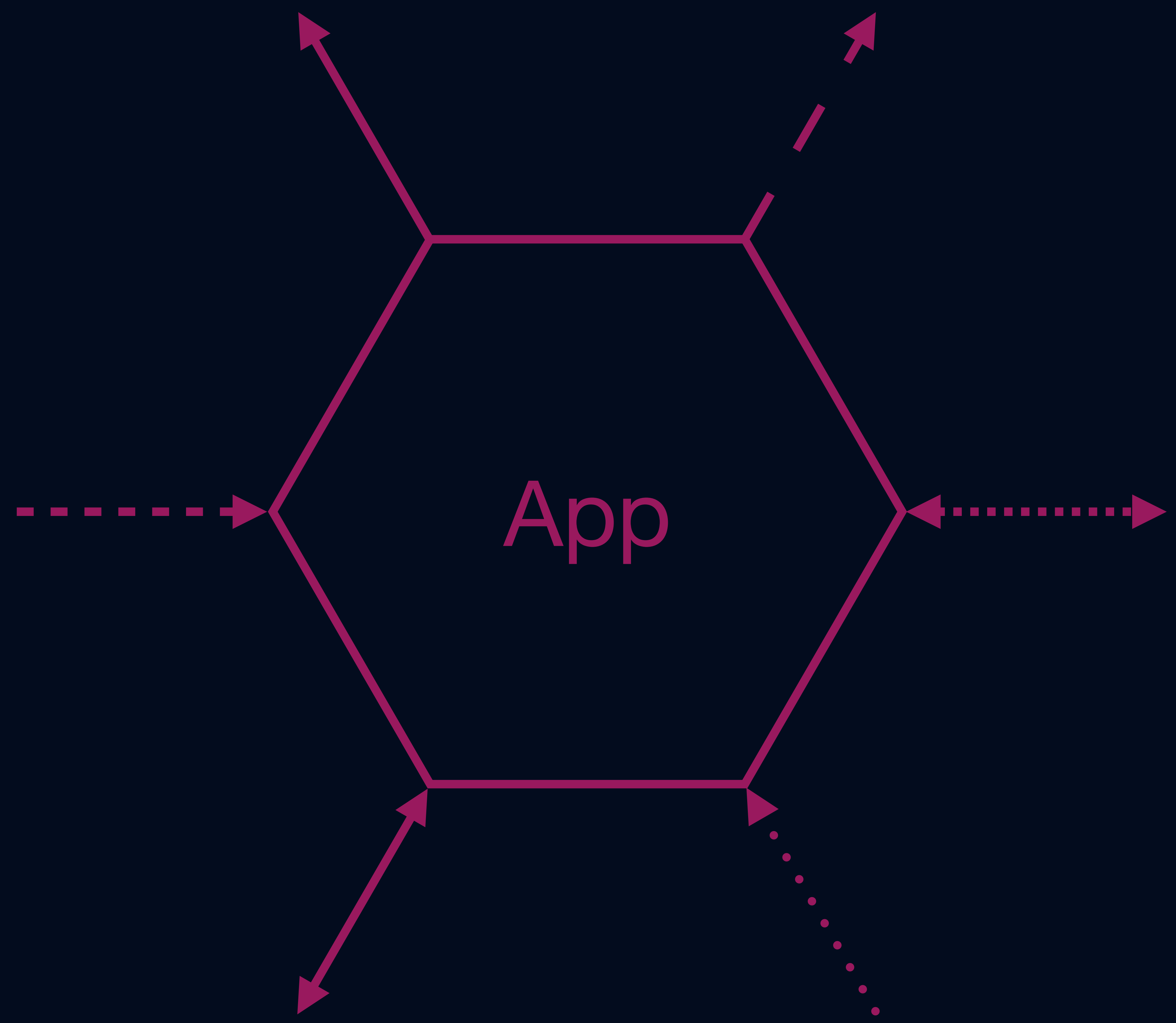
Apps have different types of communication.



Apps have different types of communication.



Make communication just work.



Let’s publish a Struct

```rust
#[derive(Serialize, Deserialize)]
struct StockEvent {
    ticker: String,
    change: f64,
}
```

Let’s publish a Struct (Connect)

```rust
// Connect to Selium
let connection = selium::custom()
    .endpoint("127.0.0.1:7001")
    .with_certificate_authority("../certs/client/ca.der")?
    .with_cert_and_key(
        "../certs/client/localhost.der",
        "../certs/client/localhost.key.der",
    )?
    .connect()
    .await?;
```

Let’s publish a Struct (Open a Stream)

```rust
// Open a publish stream
let mut publisher = connection
    .publisher("/acmeco/stocks")
    .with_encoder(BincodeCodec::default())
    .open()
    .await?;
```

Let’s publish a Struct (Send events)

```rust
// Publish some `StockEvent`s
publisher.send(StockEvent { ticker: "APPL".into(), change: 3.5 }).await?;
publisher.send(StockEvent { ticker: "INTC".into(), change: -9.0 }).await?;
```

Let’s publish a Struct (Receive events)

```rust
let subscriber = connection
    .subscriber("/acmeco/stocks")
    .with_decoder(BincodeCodec::<StockEvent>::default())
    .open()
    .await?;

subscriber
    .try_for_each(|event| {
        println!("Stock {} has changed by {}", event.ticker, event.change);
        // Stock AAPL has changed by 3.5
        // Stock INTC has changed by -9.0
        futures::future::ok(())
    })
    .await?;
```

Let’s query a server

```rust
#[derive(Debug, Serialize, Deserialize, Clone)]
enum Request {
    HelloWorld(Option<String>),
}

#[derive(Debug, Serialize, Deserialize, PartialEq, Clone)]
enum Response {
    HelloWorld(String),
}
```

Let’s query a server (Request)

```rust
let mut requestor = connection
    .requestor("/some/endpoint")
    .with_request_encoder(BincodeCodec::default())
    .with_reply_decoder(BincodeCodec::<Response>::default())
    .with_request_timeout(Duration::from_secs(3))?
    .open()
    .await?;

let request: Request = Request::HelloWorld(Some("idiot".into()));

let res = requestor.request(request).await.unwrap();
match res {
    Response::HelloWorld(greeting) => println!("{greeting}"),
    // Hello, idiot!
}
```

Let’s query a server (Response)

```rust
async fn setup(connection: &Client) -> Result<()> {
    let replier = connection
        .replier("/some/endpoint")
        .with_request_decoder(BincodeCodec::default())
        .with_reply_encoder(BincodeCodec::default())
        .with_handler(handler)
        .open()
        .await?;

    replier.listen().await?; // Blocks thread...

    Ok(())
}

async fn handler(req: Request) -> Result<Response, SeliumError> {
    match req {
        Request::HelloWorld(name) => Ok(Response::HelloWorld(format!("Hello, {name}!"))),
    }
}
```

Some fun specs

- Open source (github.com/seliumlabs)
- Supports **pub/sub** and **request/reply** over the same connection
- Reliable UDP with QUIC
- > 4gbps throughput on *benchmarks*
- mTLS authentication + crypto
- Implements native **Stream** and **Sink** traits
- Pluggable compression + batching + wire protocols
- **Zero config!**

Get your free account

for Selium Cloud (Beta)

at selium.com

Selium. Be productive in minutes.

Not days.

hello@selium.com

www.selium.com