
VERTX

About me..

Independent Consultant,
Programmer. Trainer, Author

Just loves doing this kind of stuff.

(Sometimes comes with beard, sometimes not)

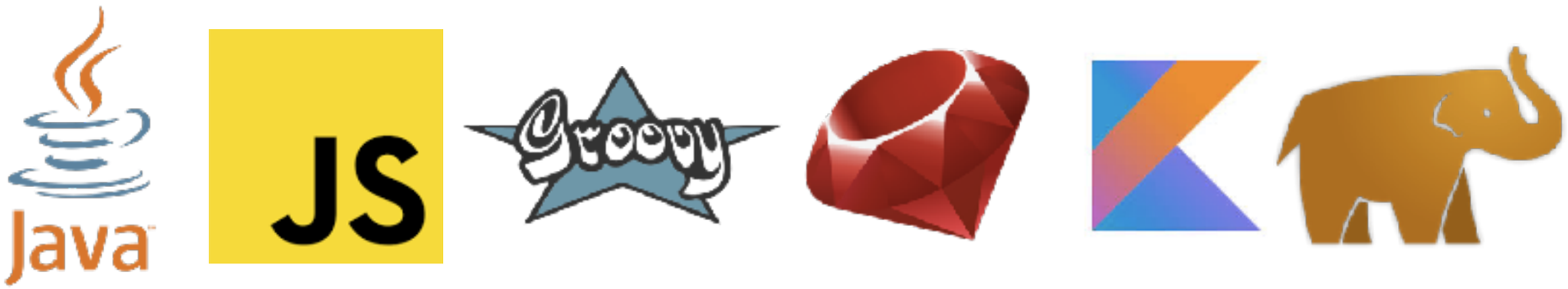
Slides and Code Available

https://github.com/dhinojosa/vertx-study

Vert.x is…

- Functional! Highly integrated with Java 8 Functions!
- Fast!
- Asynchronous until stated otherwise
- `Future<T>` Based (although it's own kind of `Future<T>`)
- Fun!

Polyglot Vert.x



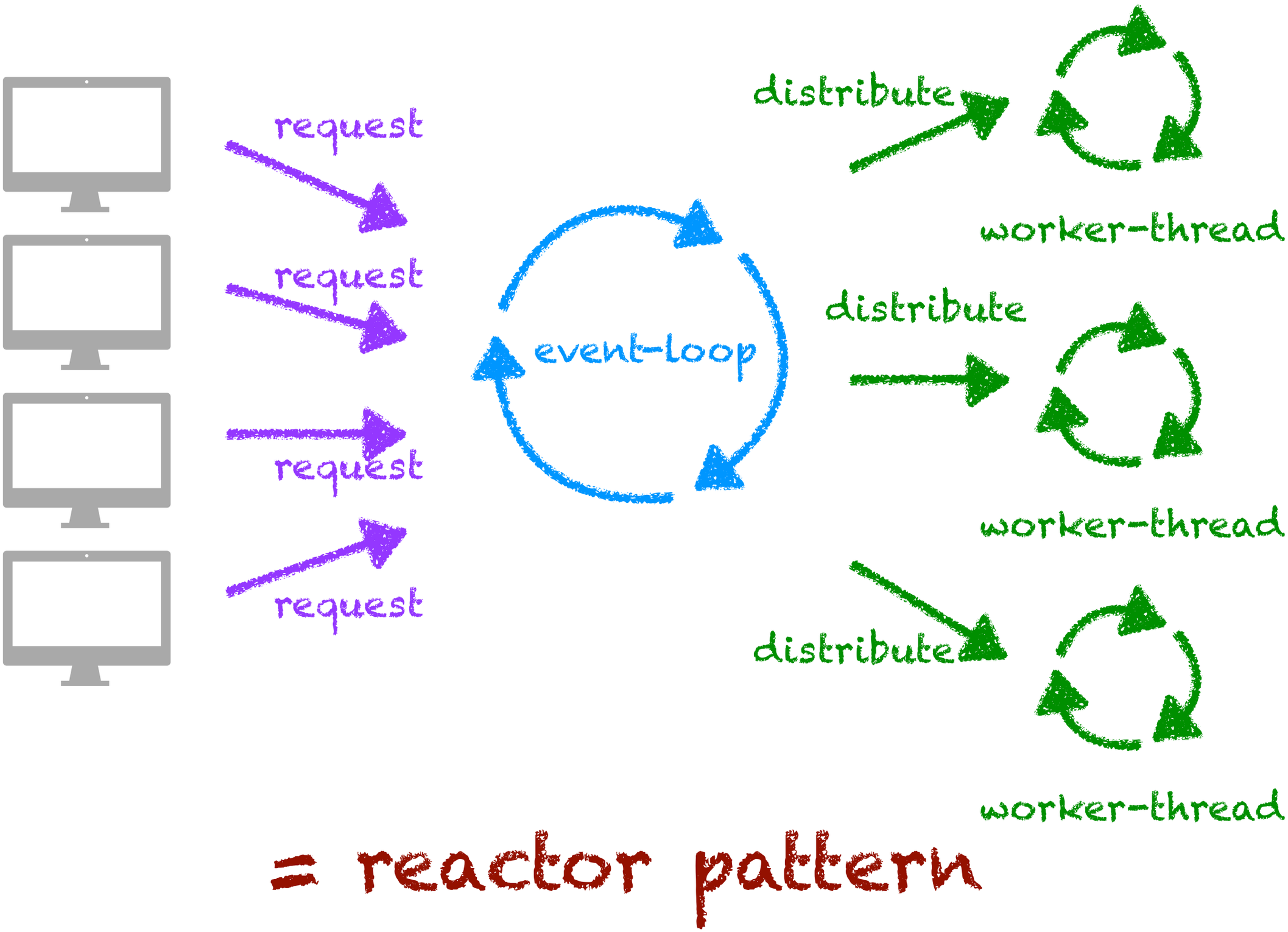
Reactive Vert.x

http://www.reactivemanifesto.org/

Triggered!

- Trigger Events Based on:
  - Timers
  - User Requests
  - Disk Reads
  - Exceptions

Vert.x Event Loop

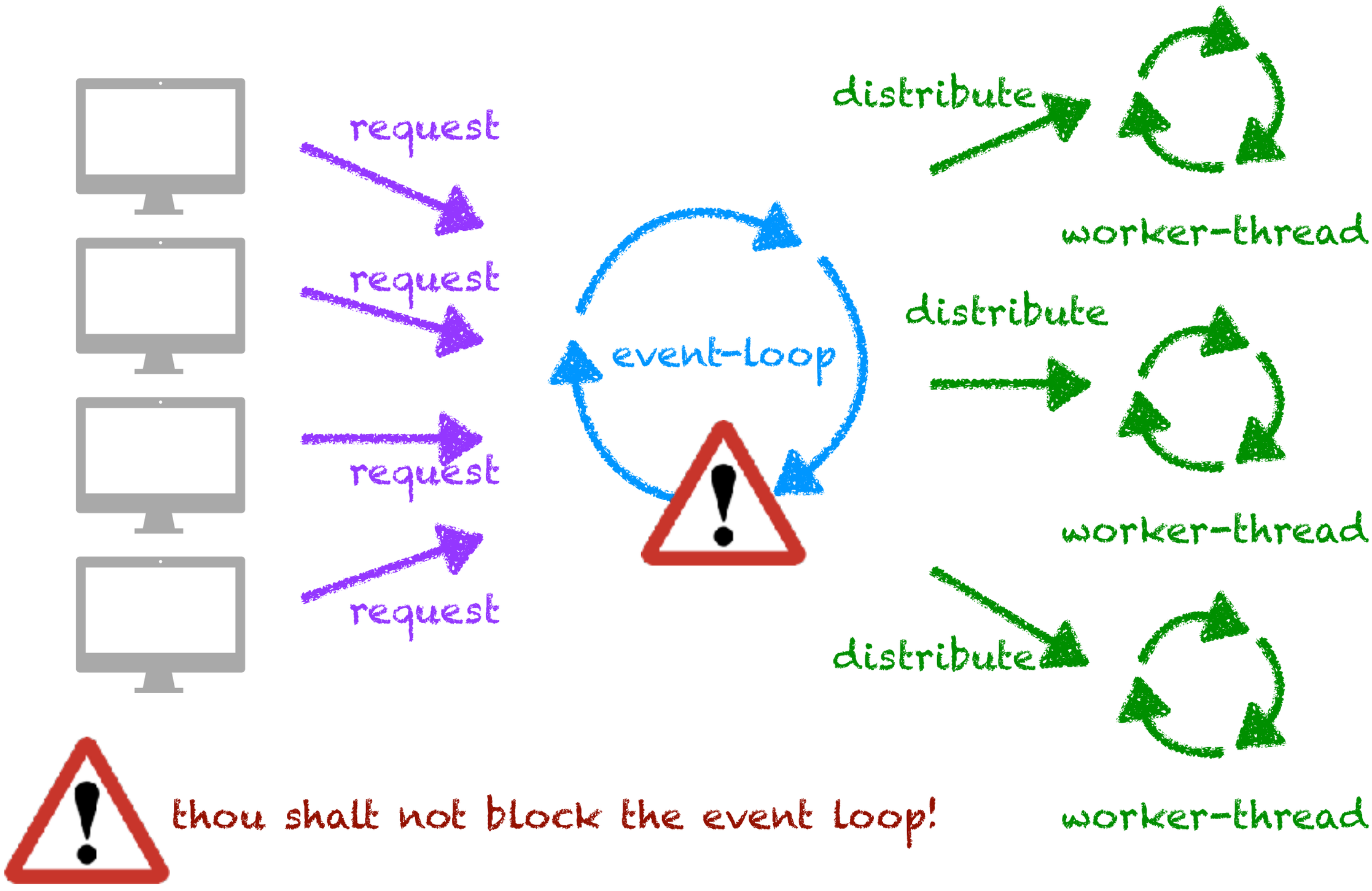


= reactor pattern

The reactor design pattern is an event handling pattern for handling service requests delivered concurrently to a service handler by one or more inputs. The service handler then demultiplexes (separates) the incoming requests and dispatches them synchronously to the associated request handlers.

Wikipedia - Reactor Pattern

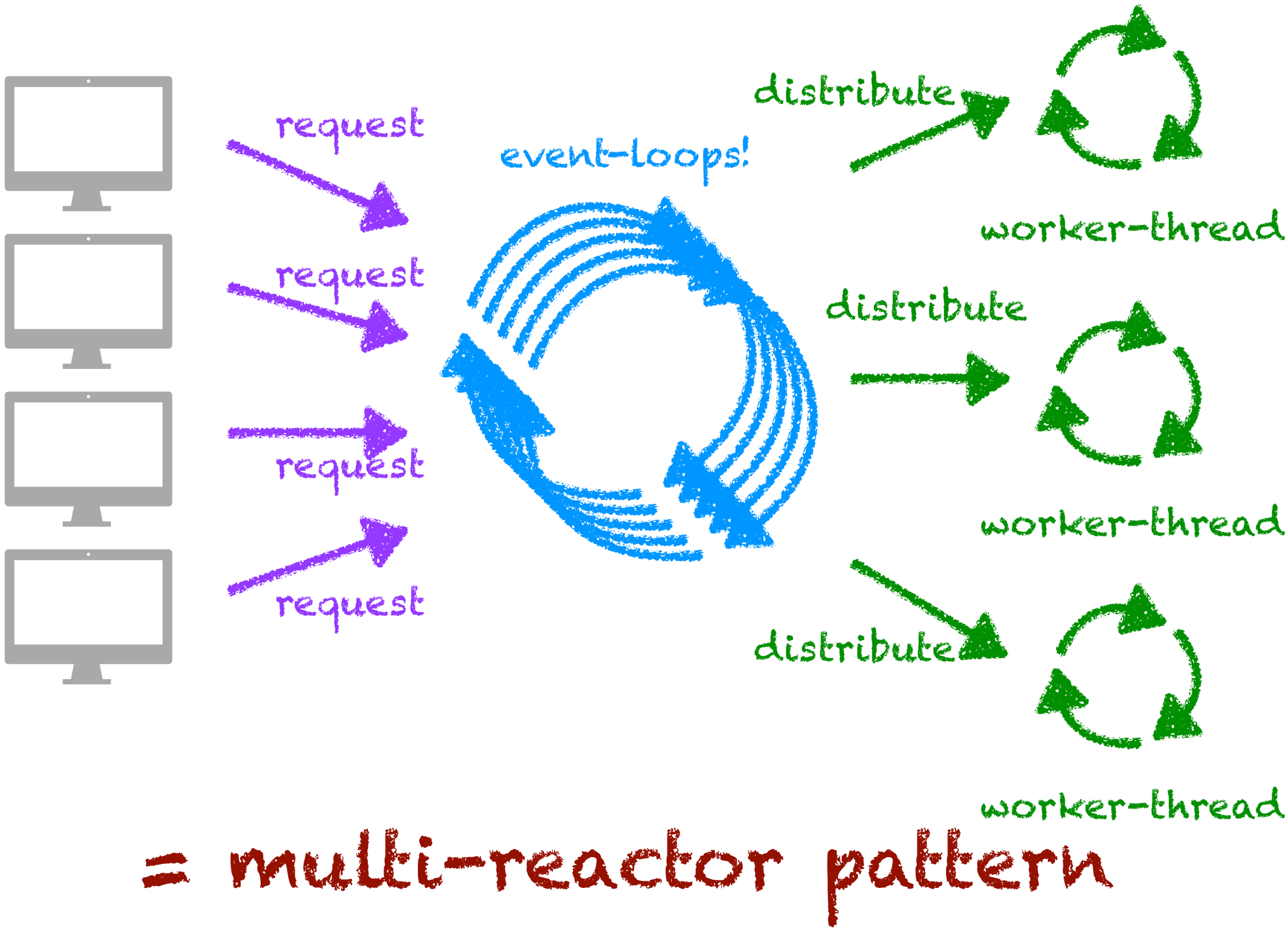
Vert.x Event Loop Warning



What blocks?

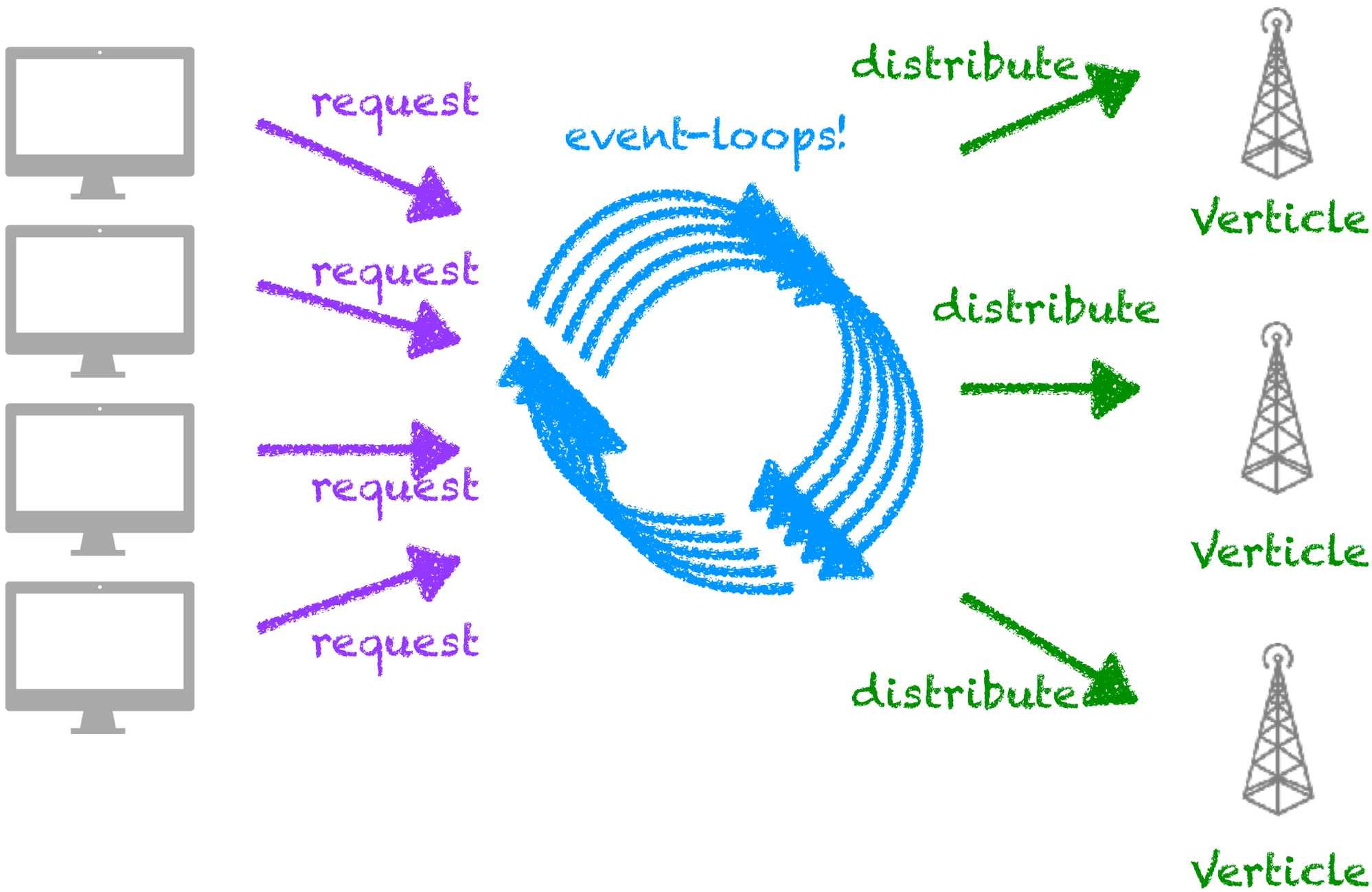
- `Thread.sleep()`
- Waiting for a monitor
- Long lived database operation
- Complex calculation
- Infinite or long running loop

Vert.x Event Loop



= multi-reactor pattern

Vert.x Event Loop & Verticles



Demo: Project & Dependencies

Running Blocking Code

Blocking Code

- Remember never block the event loop.
- But when you need to process something that blocks, use either a blocking `Verticle` (later) or `executeBlocking()`
- Takes a `Future<T>` that processes the blocking call
- Optionally can be provided a different pool to process information.

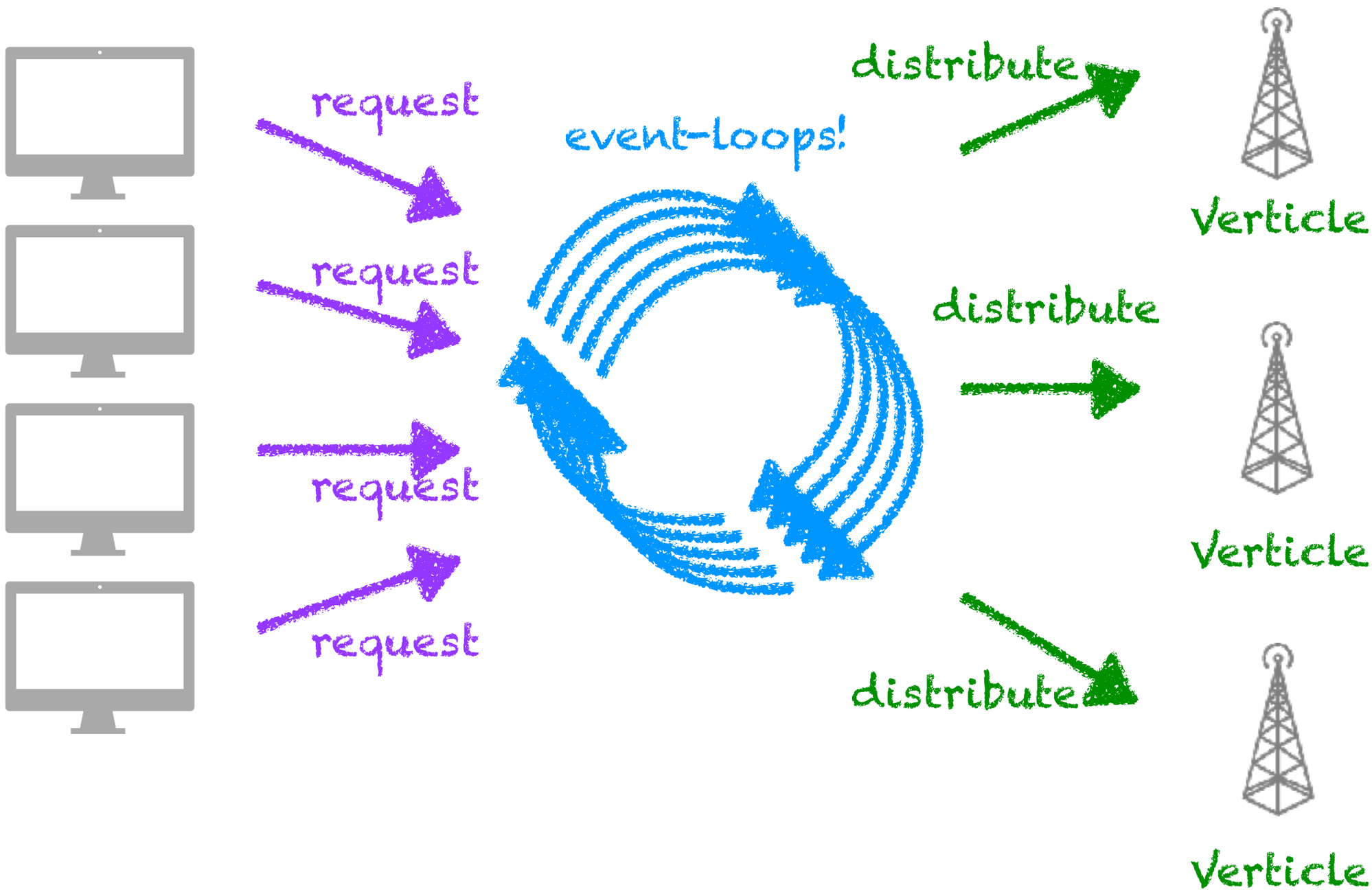
Demo: Running Vert.x to a blocking thread

Demo: Running Vert.x to a worker pool

Verticles

- Actor-like deployment
- Unit of work for Vert.x
- Optional to Use
- Backed by (number of cores x 2) threads

Vert.x Event Loop & Verticles

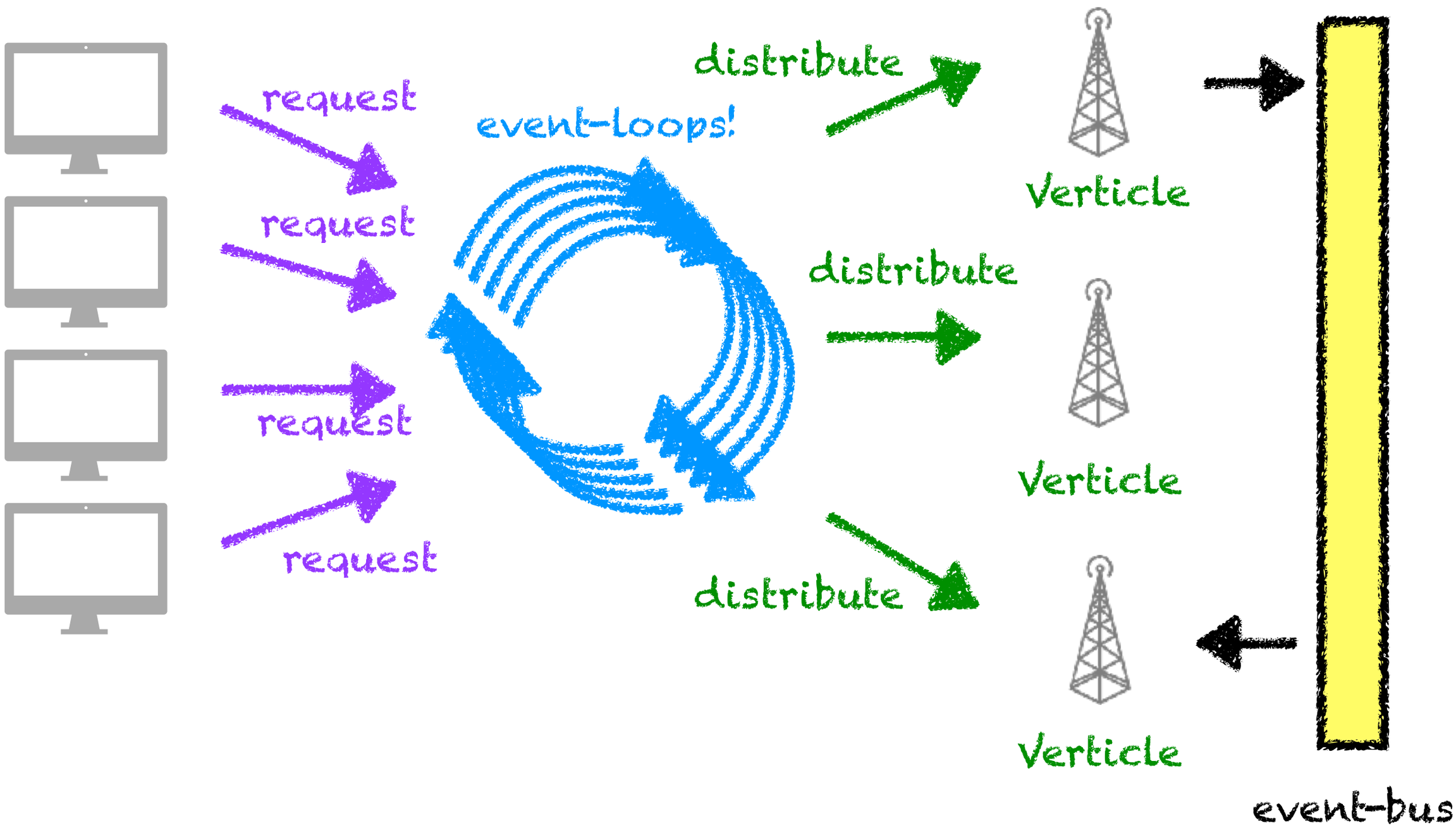


Demo: Creating Basic Verticles

Event Bus

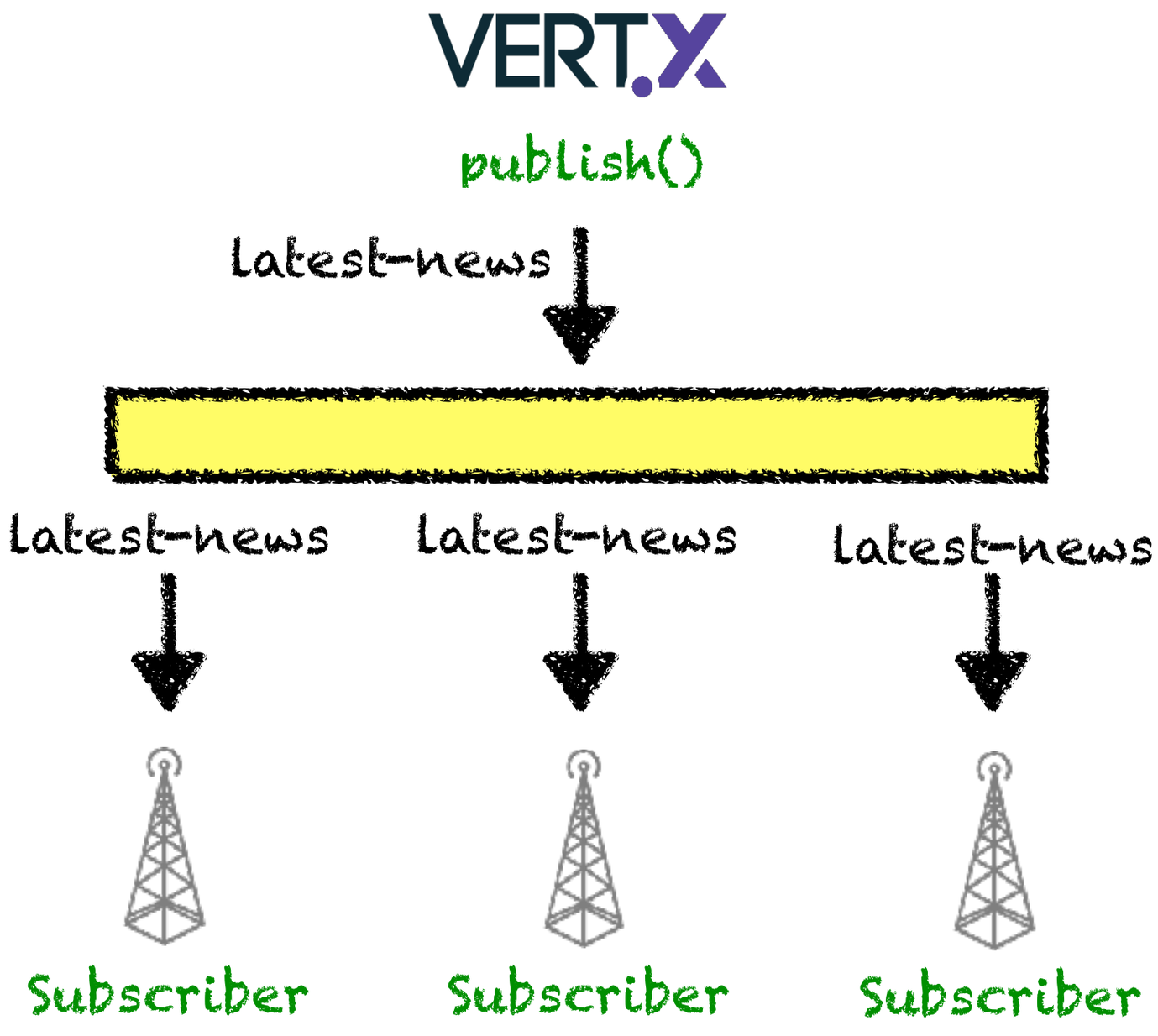
- Backbone messaging system for Vert.x Verticles
- Single Event Bus for Vert.x
- Can be called via different languages
- Can be called over the network sharing the same event bus

Vert.x Event Loop, `Verticles`, and Event Bus

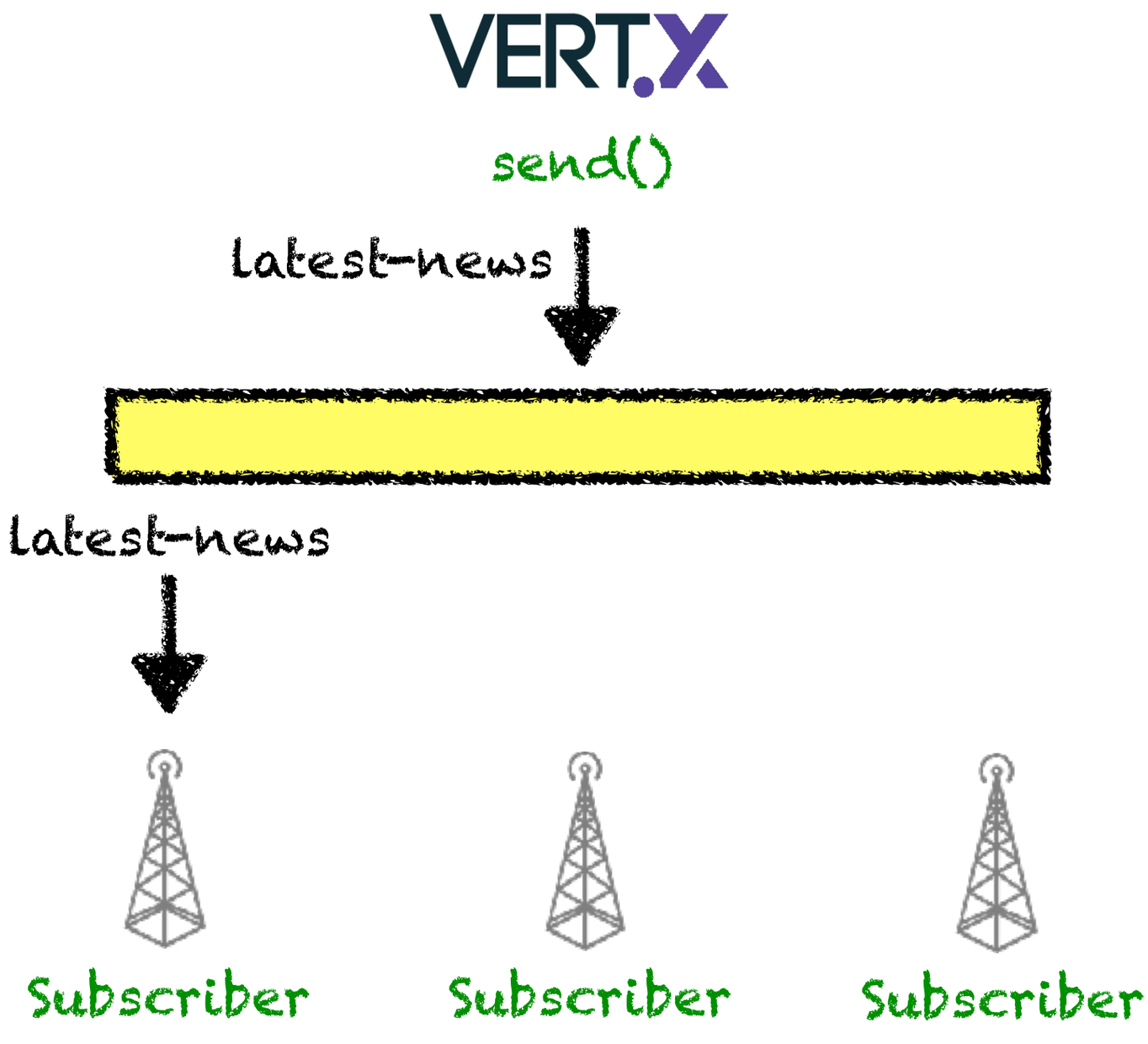


Messaging Types

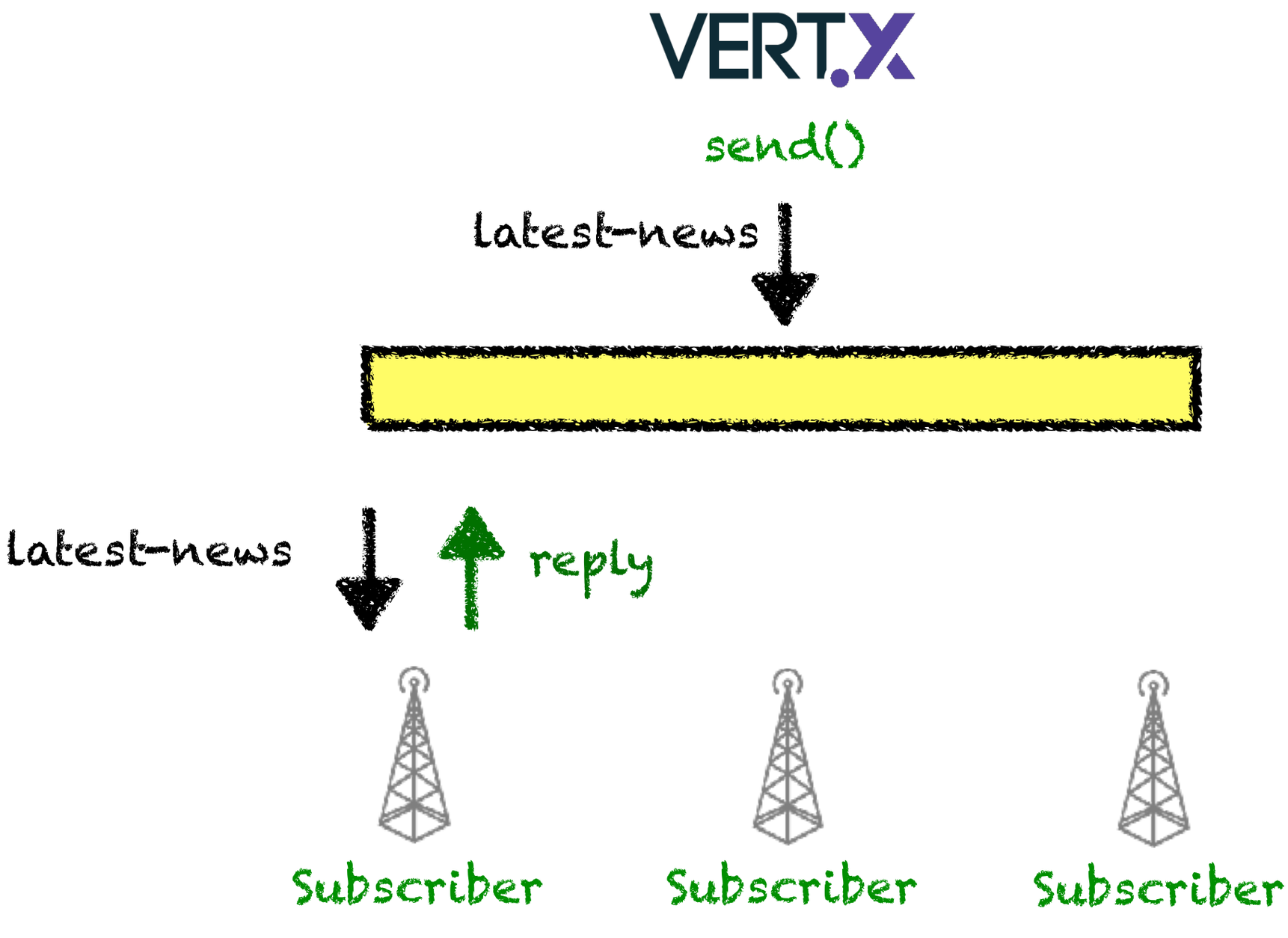
Event Bus Publish/Subscribe



Event Bus Point To Point



Event Bus Point To Point



Verticle Types

Verticle Types

Standard Verticles
These are the most common and useful type - they are always executed using an event loop thread.

Worker Verticles
These run using a thread from the worker pool. Worker verticles are designed for calling blocking code, as they won't block any event loops.

Multi-threaded worker verticles
These run using a thread from the worker pool. An instance **can be** executed concurrently by more than one thread. May need the use of locks, atomic variables. These should be rare

Demo: EventBus

with `publish`, `send`

Timers

Timers

- Timers will send message either:
  - As a one time delivery
  - Periodically
- All timers are cancellable with `cancel`

Demo: Timers and Periodic

Command Line Verticles

Command Line Verticles

- Verticles can be run via a command line
- Required that you download a distribution from `http://vertx.io/download/` and download the `.zip` or `tar.gz` file
- Add the `bin` directory to your `PATH`

Command Line Verticles

```
vertx run my-verticle.js
vertx run my-verticle.groovy
vertx run my-verticle.rb
vertx run io.vertx.example.MyVerticle
vertx run io.vertx.example.MVerticle -cp my-verticle.jar
vertx run MyVerticle.java
```

Demo: Command Line Verticles

Buffers

Buffers

- Buffers are sized content that is used to transfer binary data across Vert.x
- Analogous to a `ByteArray`
- `Buffer.buffer` creates the `Buffer`
- `Buffer` can be created with a size and be automatically adjusted later
- Content will either have to be a `Buffer` or `JSON` or a basic type otherwise a `Codec` will need to be defined
- e.g. No message codec for type: class `com.xyzcorp.Foo`

Demo: Buffers

JSON

JSON

- JSON is the preferred type of messaging in Vert.x
- Wrapped around Jackson to manipulate JSON
- Can be used with `Buffer`

JSON Creation

```
new JSONObject(“{\“count\” : 1}”);
new JSONObject(map); // Using key, value
```

JSON Arrays

```
new JSONArray(“[1,2,3,4]”)

ja = new JSONArray();
ja.add(1).add(2).add(3);
```

JSON Adding Items

```
jsonObject.put(“tickerPrice”, 40.00);
```

JSON Getting Items

```
Double price =
jsonObject.getDouble(“tickerPrice”);
```

VERT.X

Demo: Sending JSON

HTTP Servers

HTTP Servers

- Non-blocking Servers
- Supports HTTP 1.0, HTTP 1.1, HTTP 1.2
- Content requests can be received in chunks in the form of `Buffer`
- `HTTPServer` can be launched outside or inside a `Verticle`

Demo: HTTP

Clustering

Clustering in Vert.x

- Clustering can be done programmatically or declaratively at a terminal level
- Uses Hazelcast as the default
- Uses Multicasting to determine other Vert.x instances on your network
- Maintains a network wide EventBus automatically
- Has it's own settings by default, but you can make your own with a `cluster.xml`

Demo: Clustering

Local and Clustered Shared Maps

Local and Clustered Shared Maps

- Local Shared Map are shared between different event loops in the same Vert.x instance
- Clustered Shared Map are shared between different event loops across different Vert.x instances

```
SharedData sd = vertx.sharedData();
sd.getClusteredWideMap(“shared-map”);
sd.getLocalMap(“same-vertx-map”);
```

Vert.x Client

Vert.x Client

- Client is a separate library
- JSON encoding/decoding
- Form Submissions
- All Vertx Basics

Demo: Vertx Client

Extensions

Extensions

- **Vertx-web:** Higher level web application library that is analogous with Sinatra and Express. Offers authentication, CSRF Support, Routing
- **Vertx-client:** A very easy higher level application library for connecting to website. With Vert.x JSON Integration makes consumption easy.
- **Various-Vertx Datastore Clients:** MongoDB, MySQL, PostgreSQL, Redis Clients, Straightforward JDBC Communication
- **Various Messaging Clients:** Kafka, AMQP-Bridge, JavaMail

Performance

Web Framework Benchmarks

Although speed is not always the point, it is worthwhile to investigate

https://www.techempower.com/benchmarks/

Praises

Praises

- Easy to comprehend, easier to use than many frameworks
- Non-Blocking & Reactive
- JSON Implementation Ready
- Clustering Built-In
- Lot of Libraries to Extend Vert.x
- Growing Ecosystem

Complaints

Complaints

- Wish deployments returned `Future<T>`
- Wish that it used Java 8 `Future<T>` but may likely work against polyglot communication.
- Strange errors when clustering at times

Thank You