Rethinking REST in a Microservices World

James Roper

@jroper

Agenda

- What is asynchronous communication?
- Asynchronous vs synchronous architectures
- Asynchronous communication in practice

Asynchronous communication

Asynchronous communication

- Not asynchronous IO

Asynchronous communication

- Not asynchronous IO
  - Asynchronous IO is good…

Asynchronous communication

- Not asynchronous IO
  - Asynchronous IO is good…
  - … but that's not what this presentation is about

Asynchronous IO

- Don't consume threads while waiting for things

Asynchronous IO

- Don't consume threads while waiting for things
  - Responses are handled via callbacks

```
CompletionStage<Response> futureResponse = makeRequest();

CompletionStage<MyModel> futureModel =
  futureResponse.thenApply(response -> {
    MyModel model = decode(response.getBody());
    return model;
  });
```

Asynchronous IO

- Don't consume threads while waiting for things
  - Responses are handled via callbacks

```
CompletionStage<Response> futureResponse = makeRequest();

CompletionStage<MyModel> futureModel =
  futureResponse.thenApply(response -> {
    MyModel model = decode(response.getBody());
    return model;
  });
```

  - Callback only takes a thread when it's executed

Asynchronous communication

Asynchronous communication

- Not asynchronous IO
  - Asynchronous IO is good…
  - … but that's not what this presentation is about

Asynchronous communication

- Not asynchronous IO
  - Asynchronous IO is good…
  - … but that's not what this presentation is about
  - Asynchronous IO is about not blocking threads

Asynchronous communication

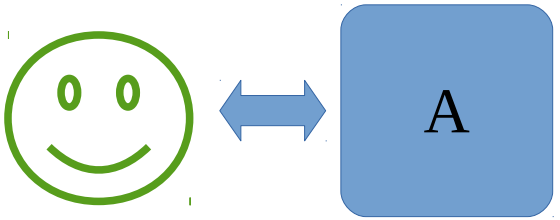
- Not asynchronous IO
  - Asynchronous IO is good…
  - … but that's not what this presentation is about
  - Asynchronous IO is about not blocking threads
- Async communication is not blocking requests

Asynchronous communication

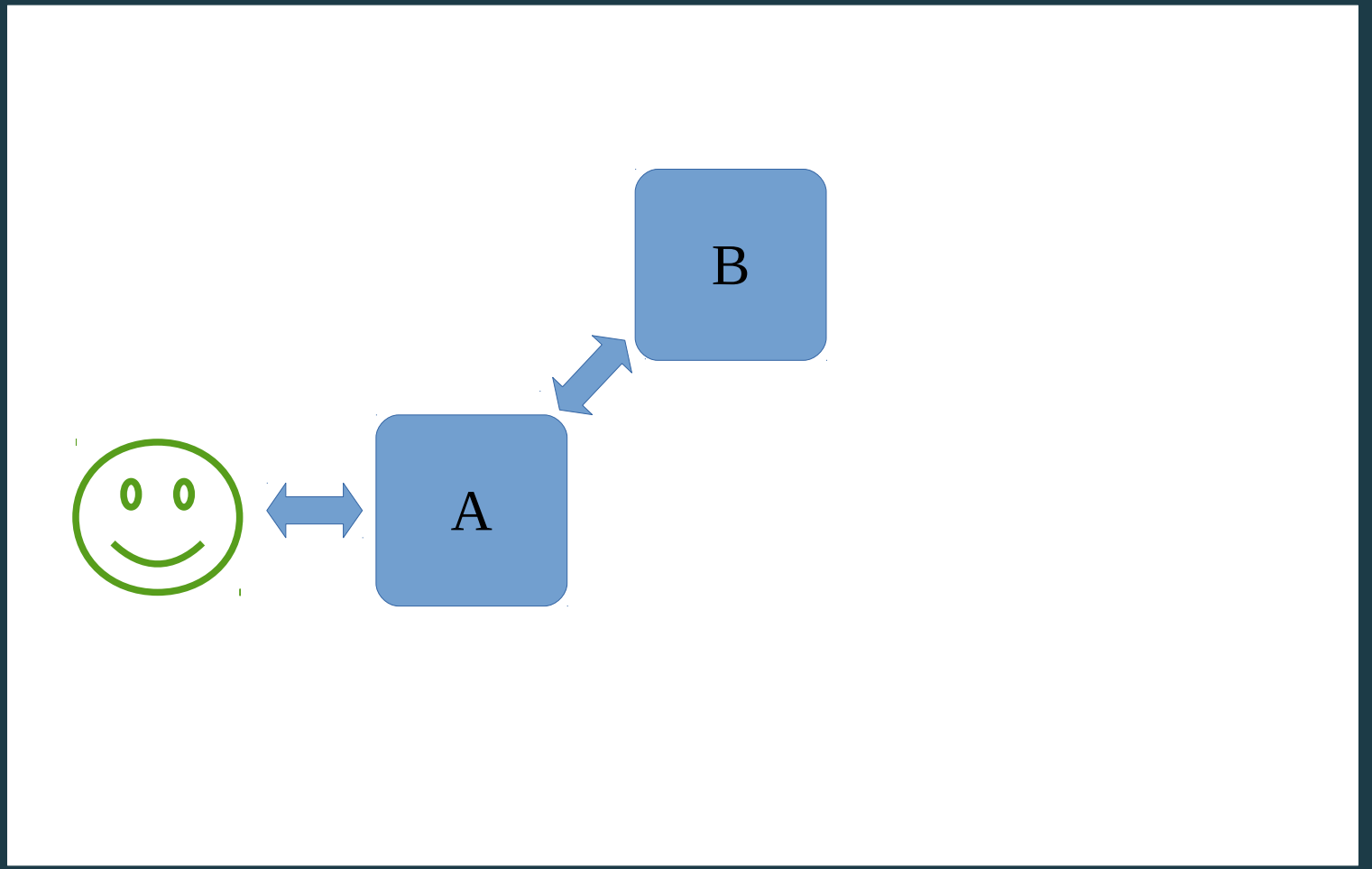
- Not asynchronous IO
  - Asynchronous IO is good…
  - … but that's not what this presentation is about
  - Asynchronous IO is about not blocking threads
- Async communication is not blocking requests
  - A service processing a request shouldn't block on another service

Synchronous communication

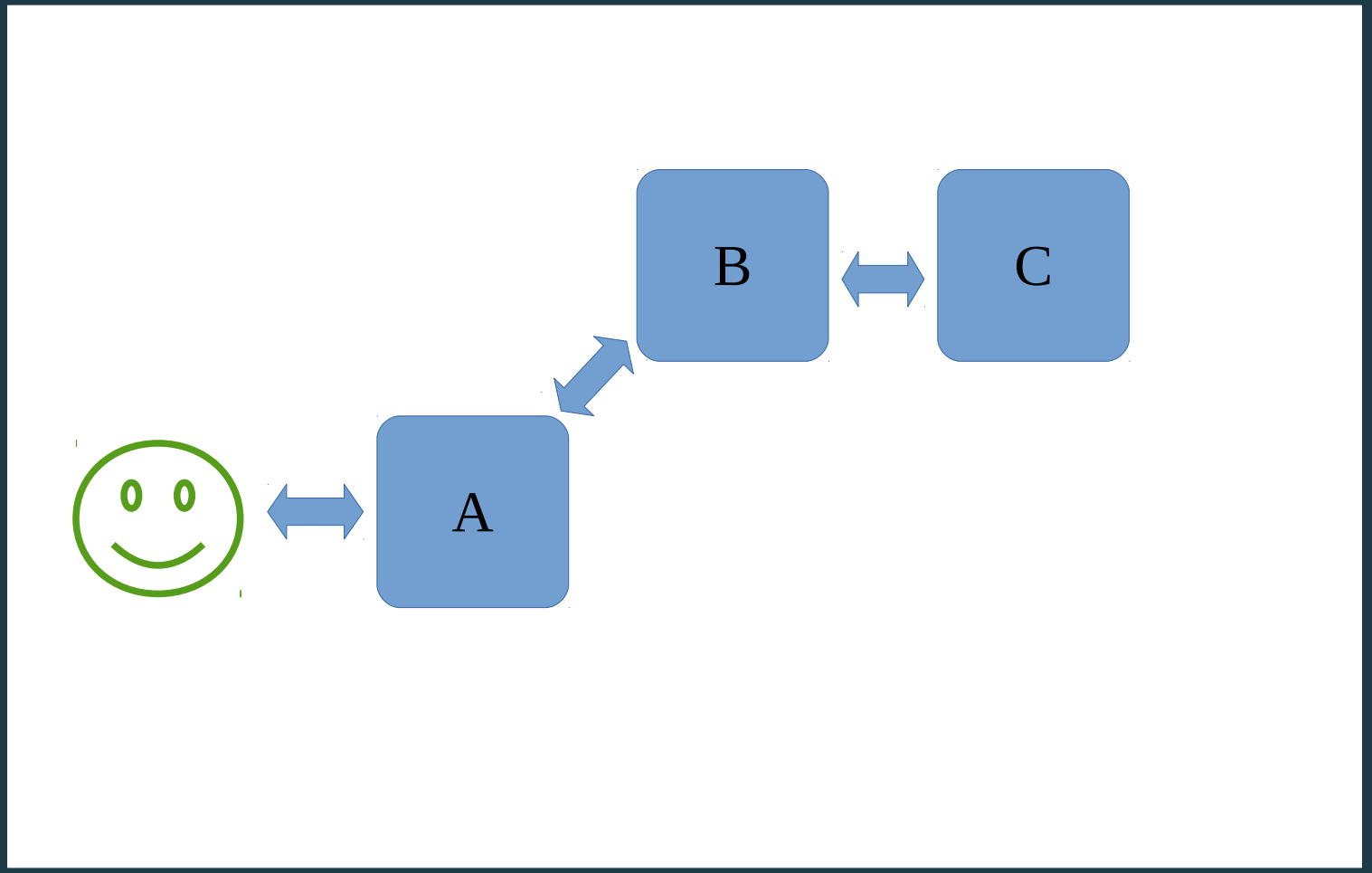
Synchronous communication



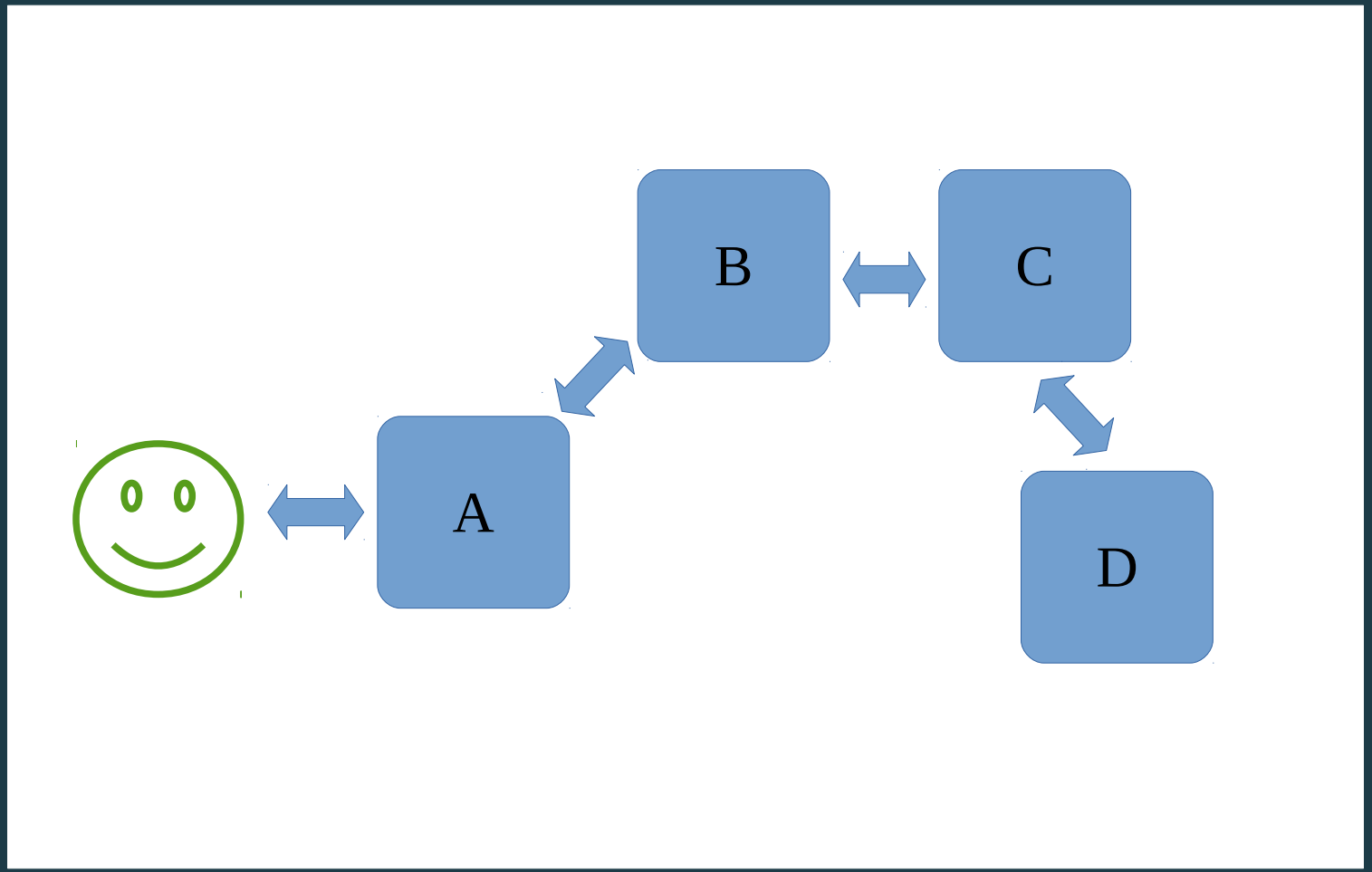
Synchronous communication



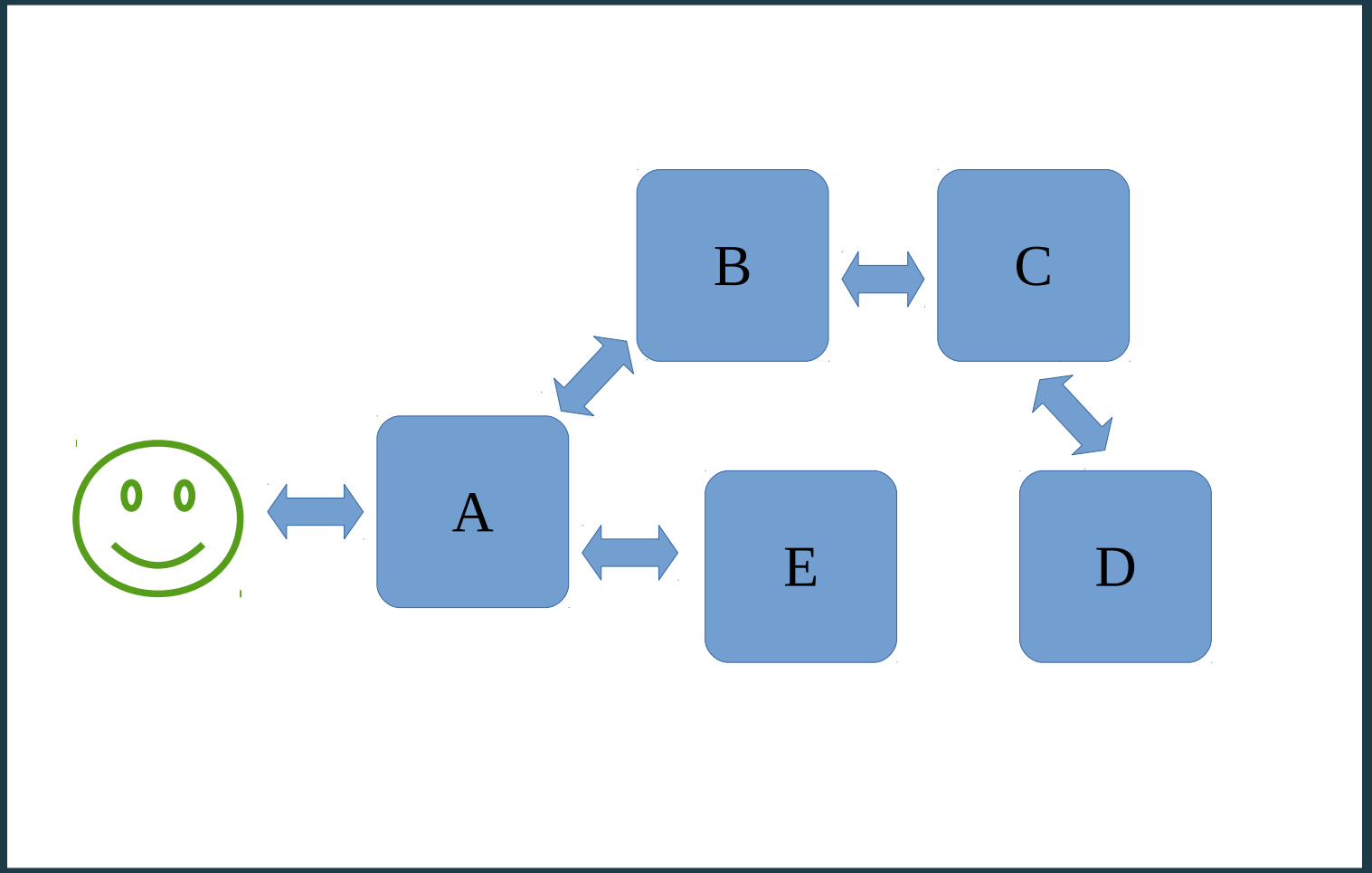
Synchronous communication



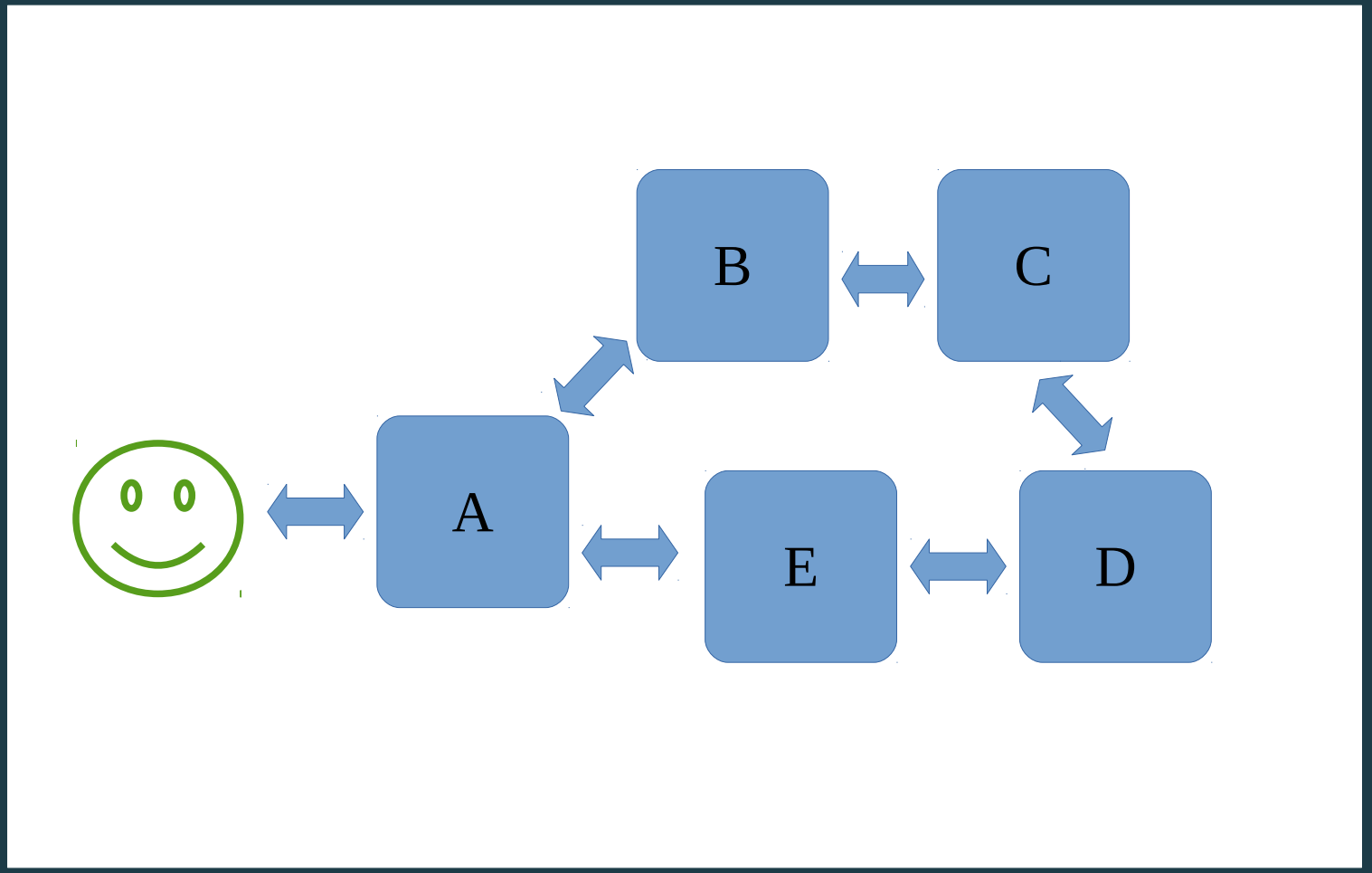
Synchronous communication



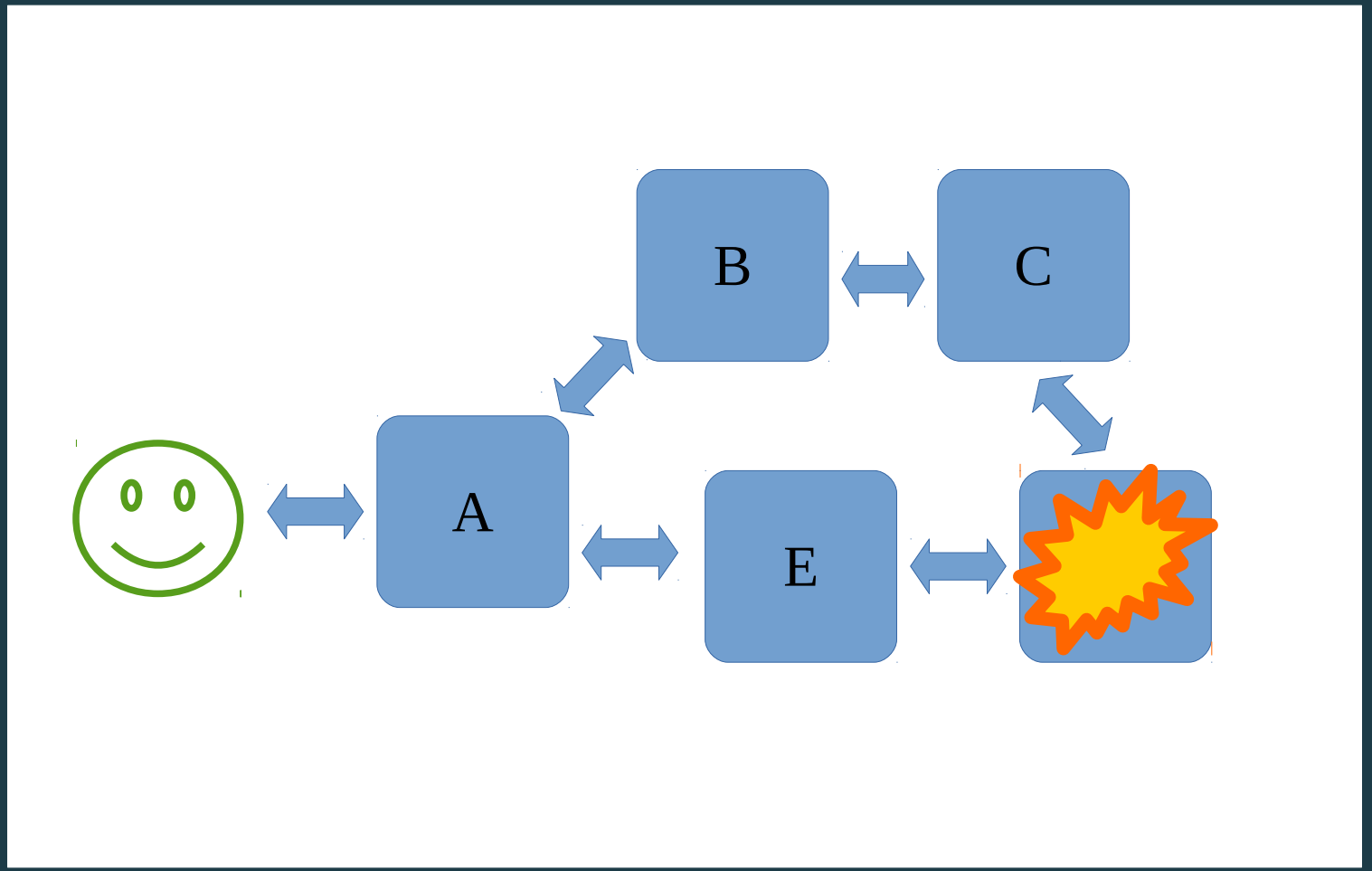
Synchronous communication



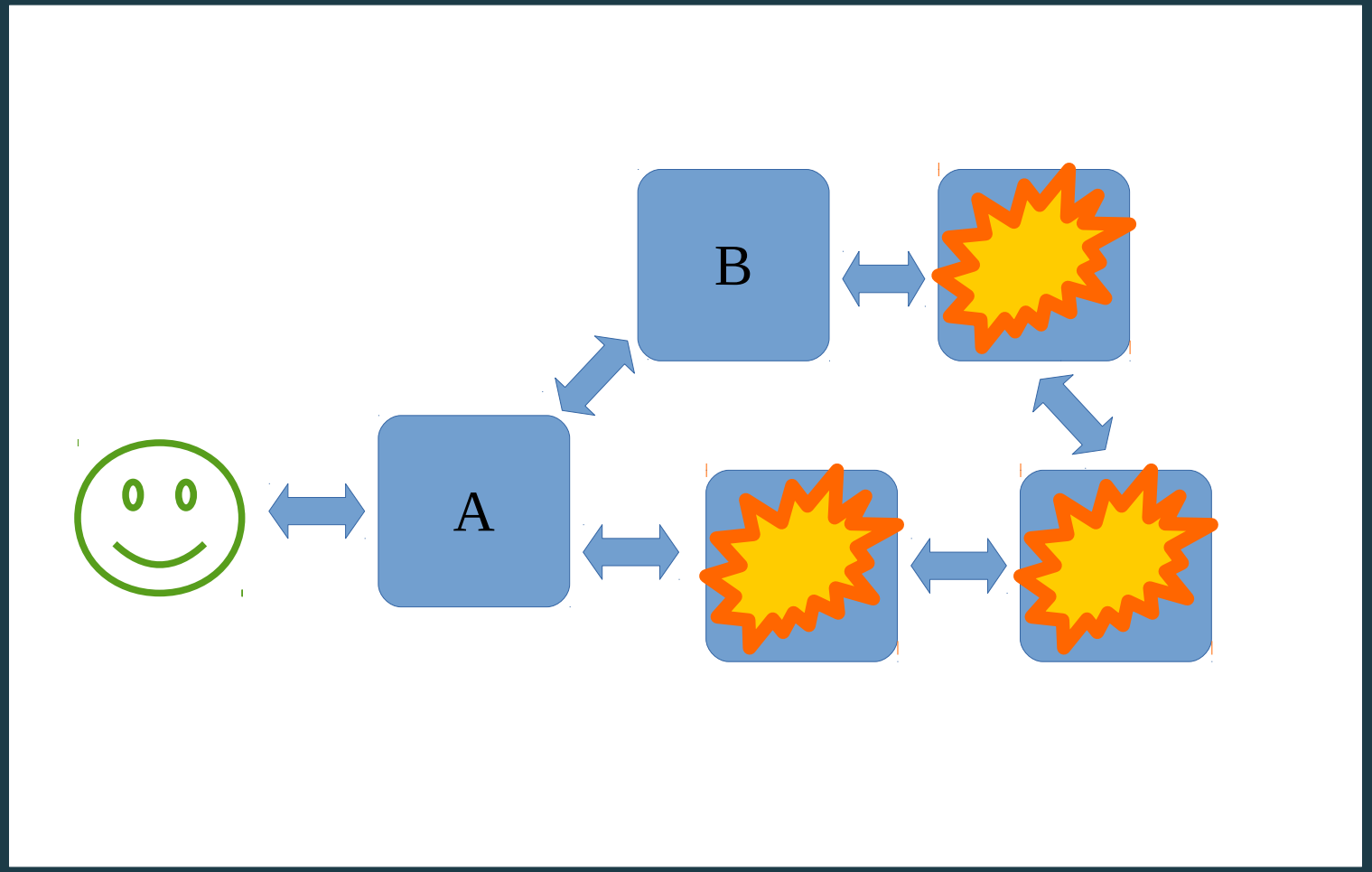
Synchronous communication



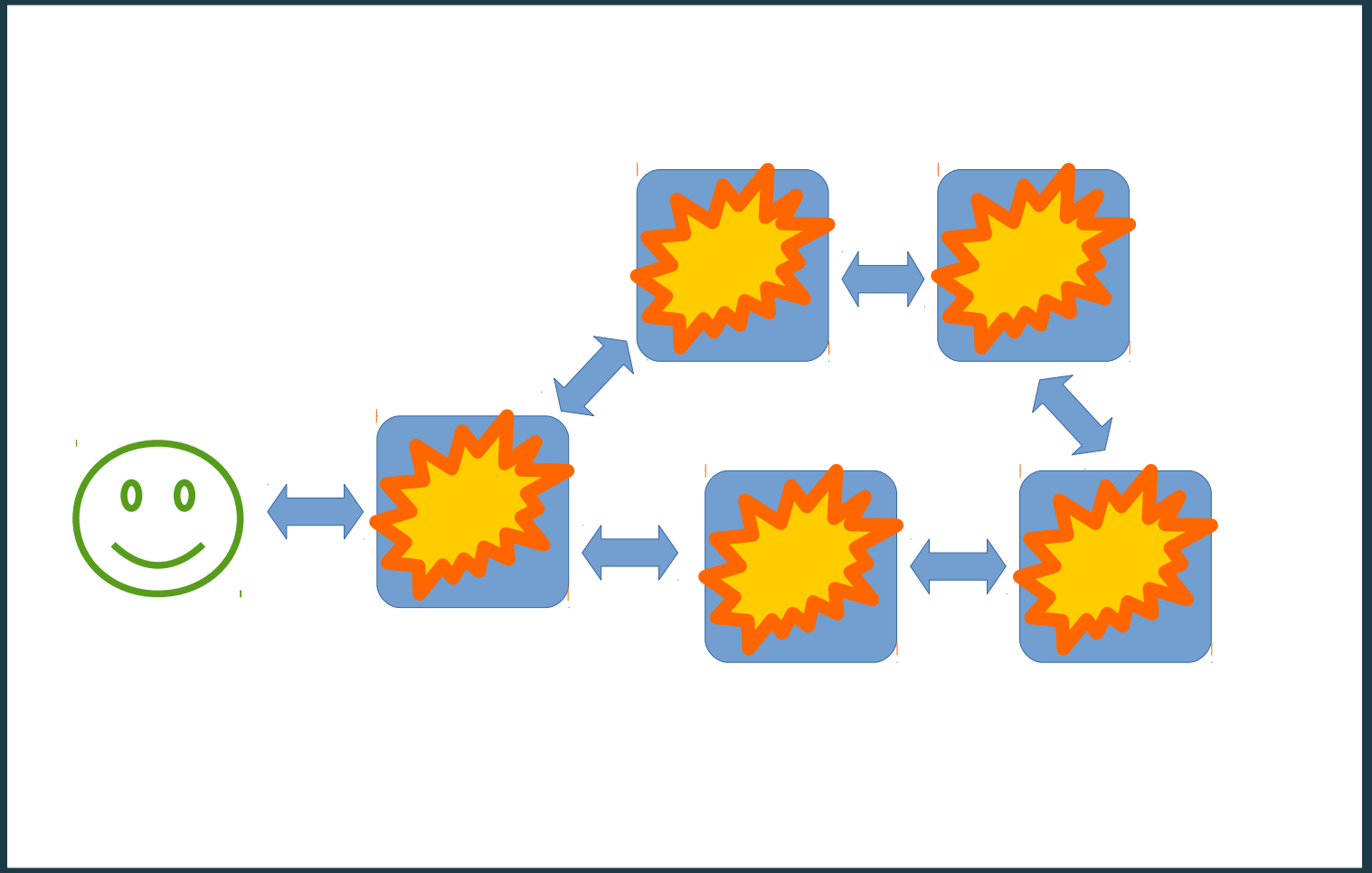
Synchronous communication



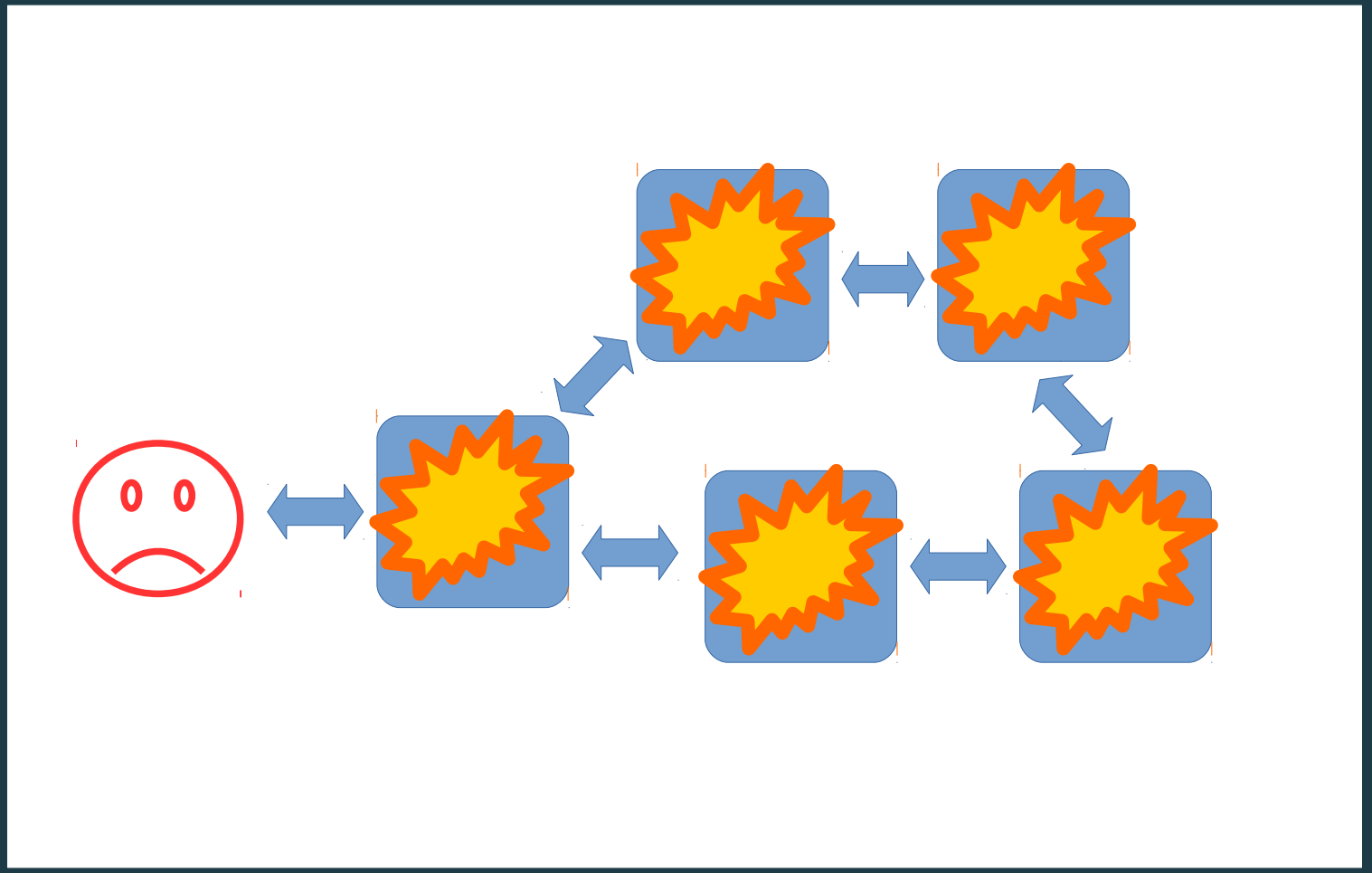
Synchronous communication



Synchronous communication



Synchronous communication



Synchronous communication

Synchronous communication

- Failure bubbles back up

Synchronous communication

- Failure bubbles back up
- Slow response times bubble back up

Synchronous communication

- Failure bubbles back up
- Slow response times bubble back up
- Some services will be needed by everything

Synchronous communication

- Failure bubbles back up
- Slow response times bubble back up
- Some services will be needed by everything
  - Bottleneck to load

Synchronous communication

- Failure bubbles back up
- Slow response times bubble back up
- Some services will be needed by everything
  - Bottleneck to load
  - Single point of failure

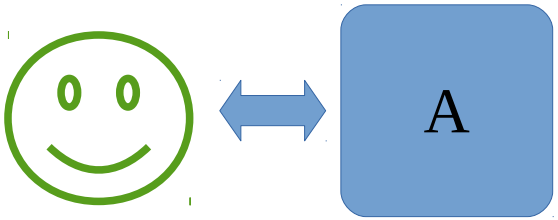
Synchronous communication

- Failure bubbles back up
- Slow response times bubble back up
- Some services will be needed by everything
  - Bottleneck to load
  - Single point of failure
- Often whole system needs to be up to serve any request

https://flic.kr/p/4h6k

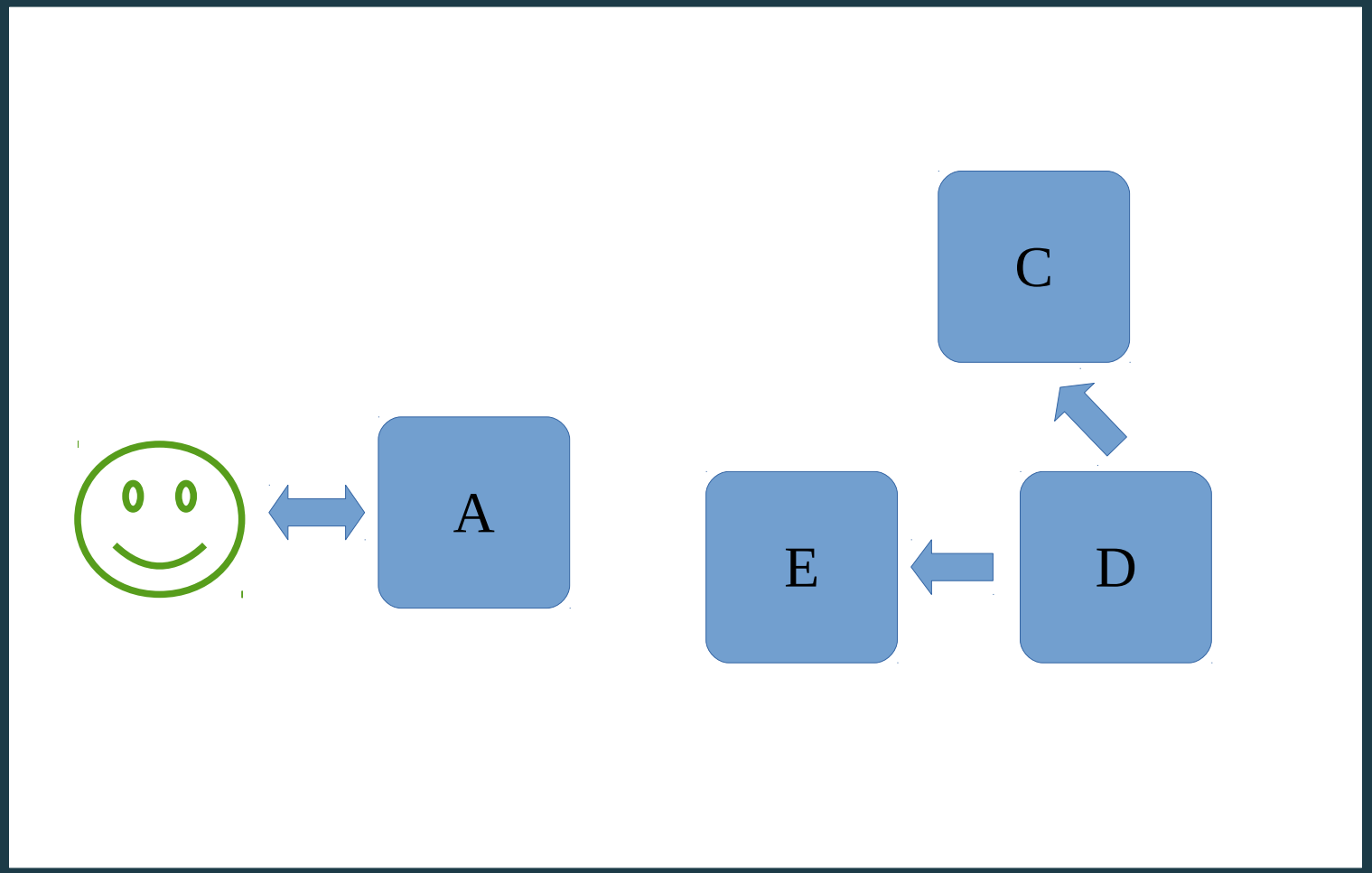
Asynchronous communication

Asynchronous communication

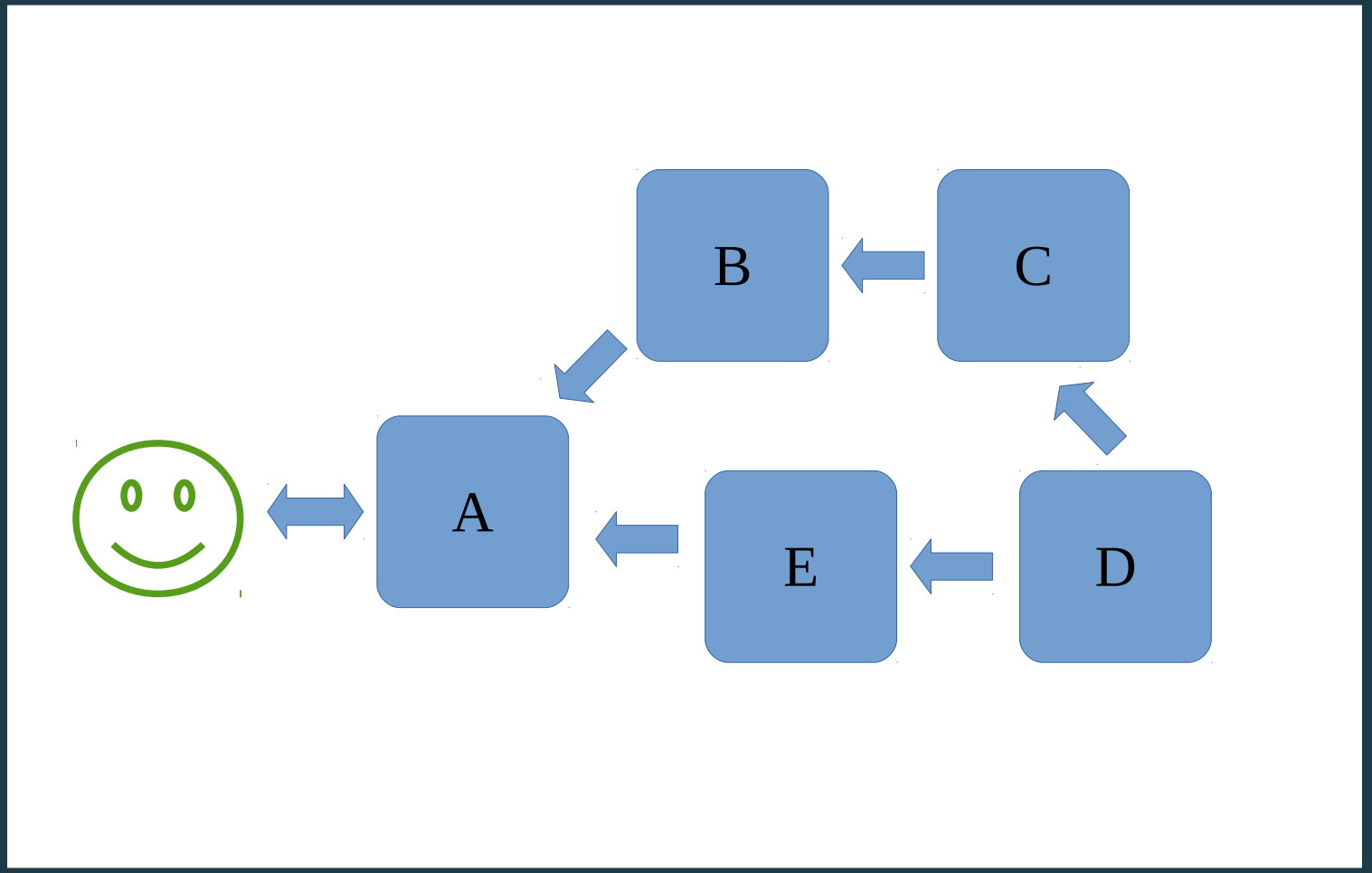


Asynchronous communication

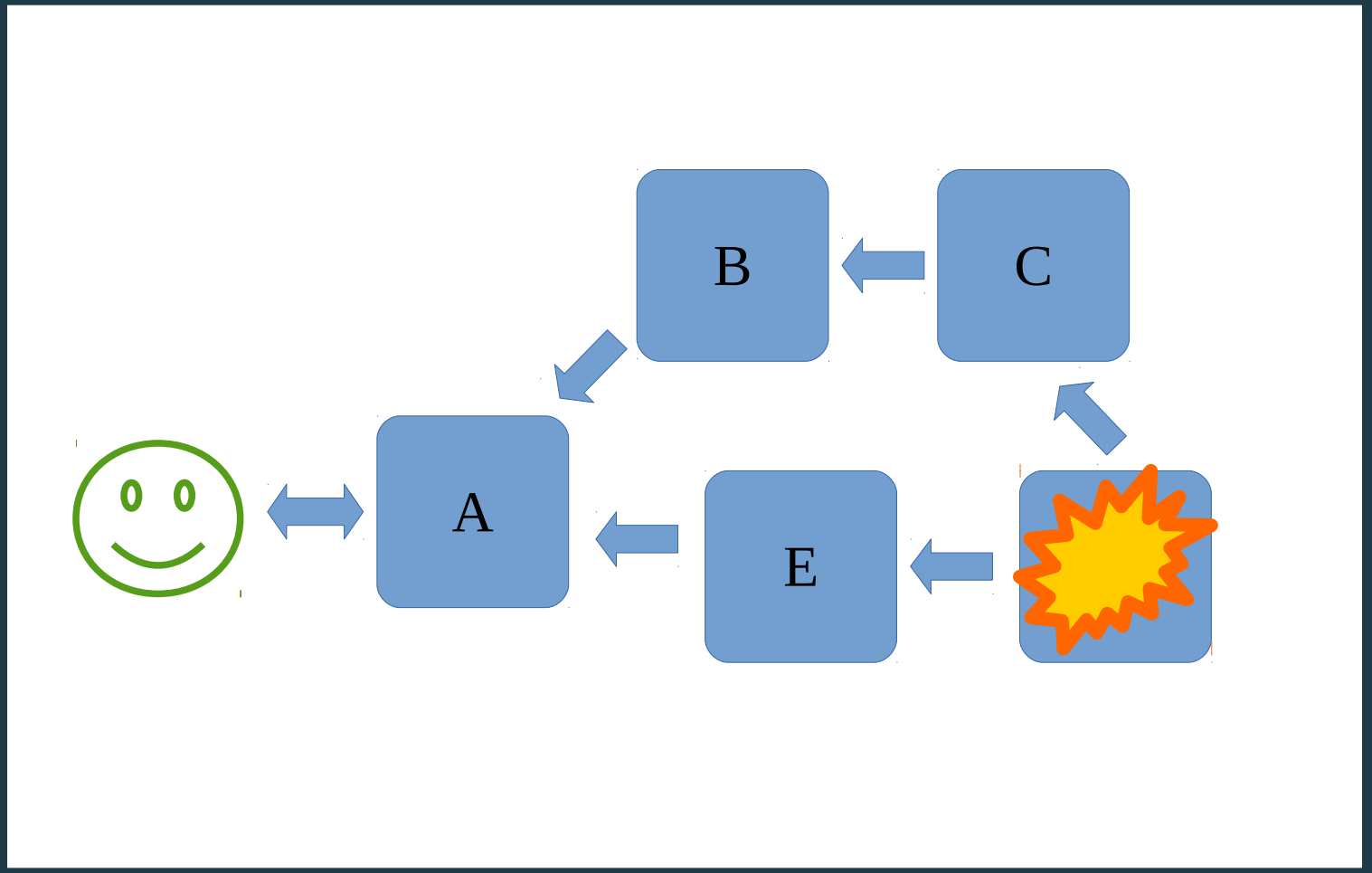
Asynchronous communication



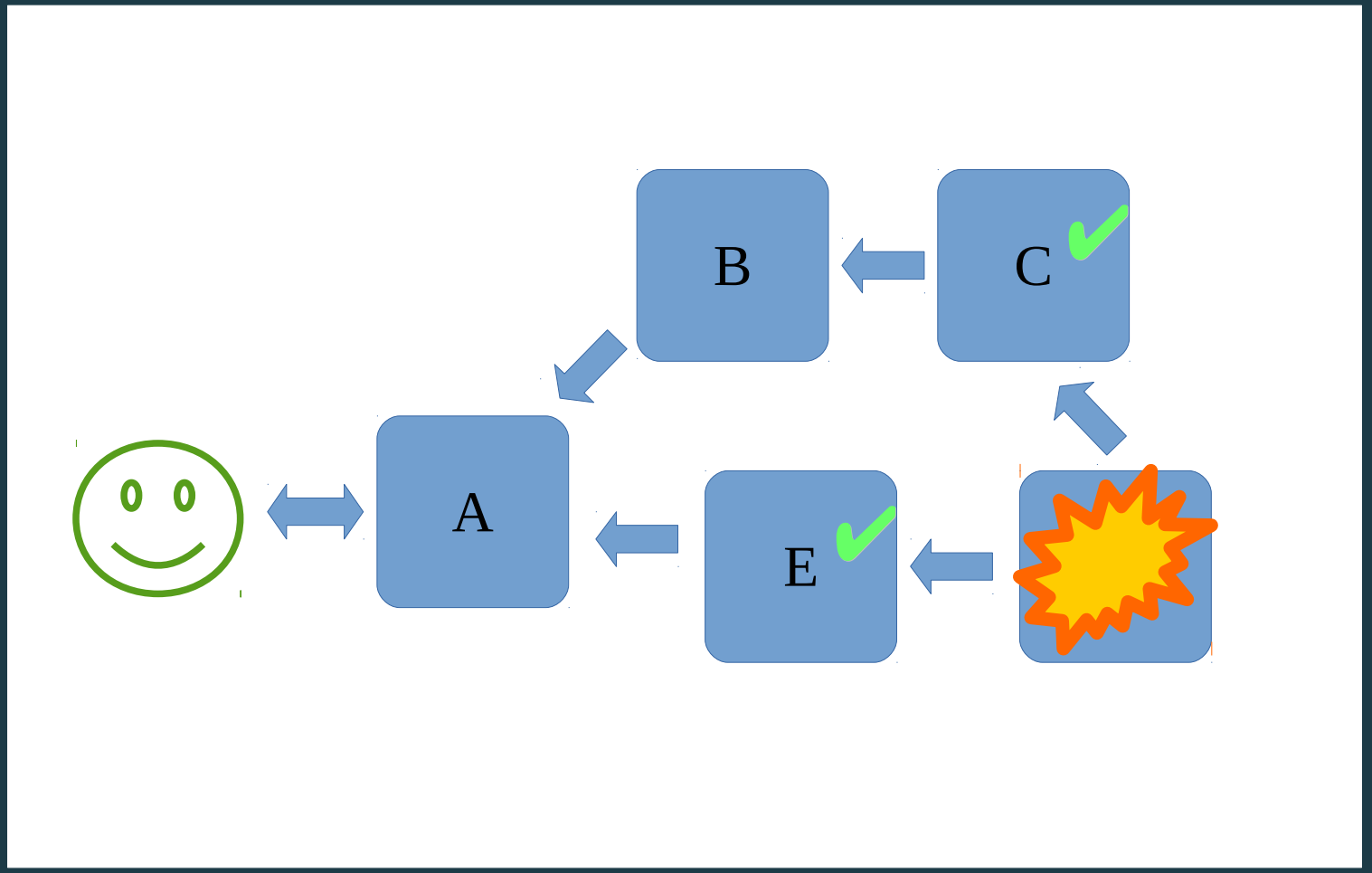
Asynchronous communication



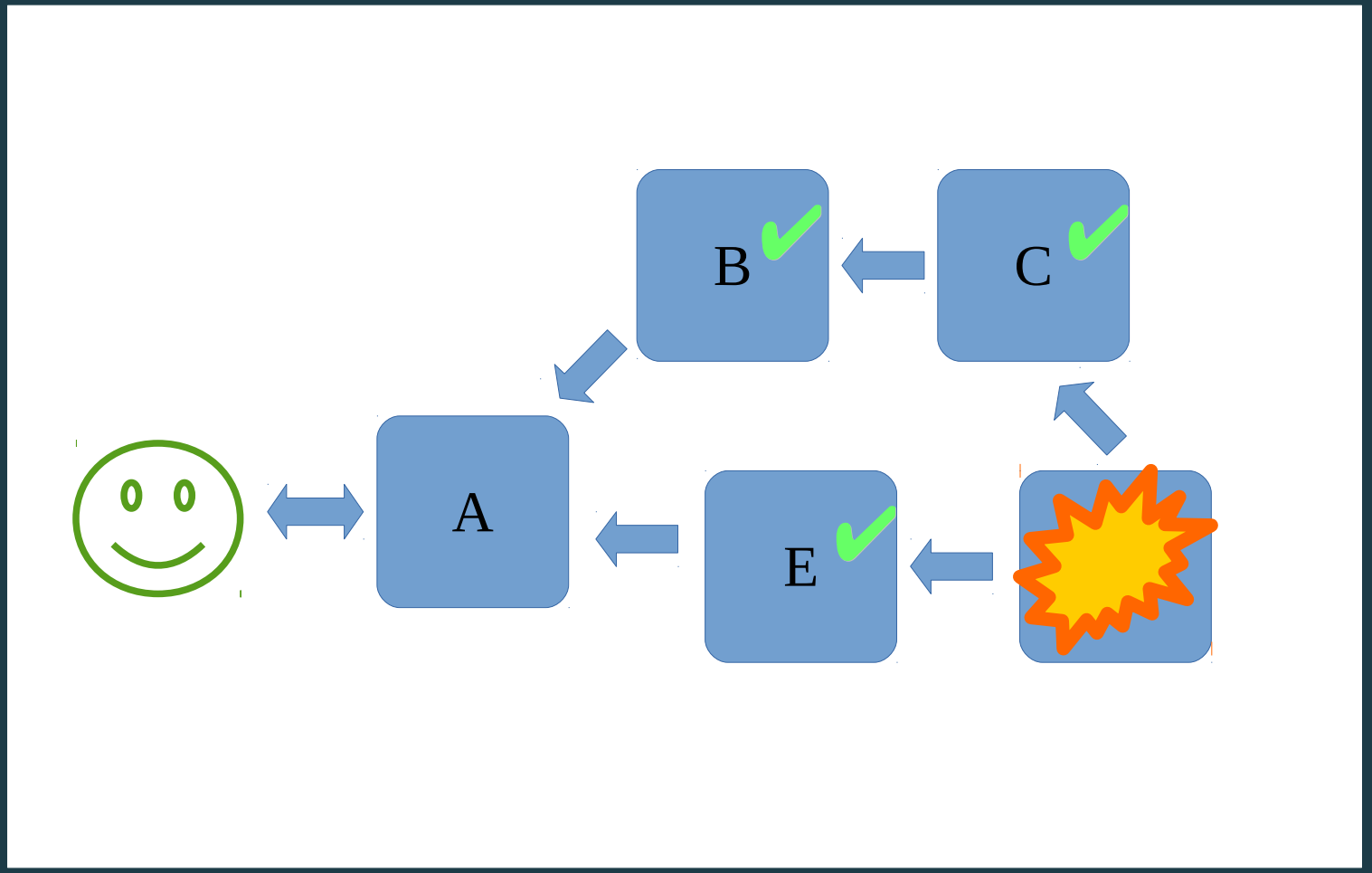
Asynchronous communication



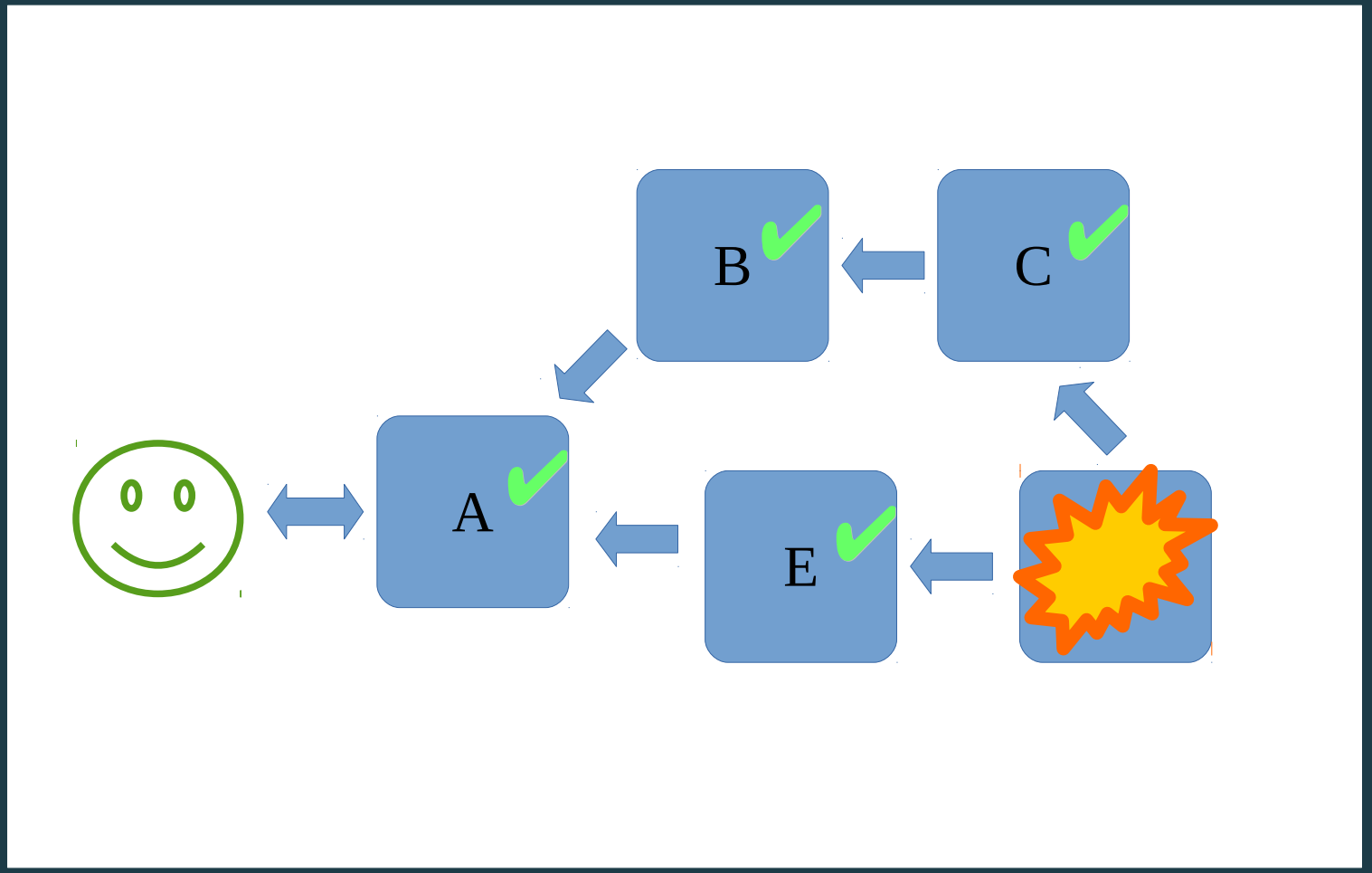
Asynchronous communication



Asynchronous communication



Asynchronous communication



Asynchronous communication

Asynchronous communication

- Failure stops at the failed component

Asynchronous communication

- Failure stops at the failed component
- Slow response time stops at slow component

Asynchronous communication

- Failure stops at the failed component
- Slow response time stops at slow component
- Removes bottlenecks/single points of failure

Asynchronous communication

- Failure stops at the failed component
- Slow response time stops at slow component
- Removes bottlenecks/single points of failure
- Consistency may lag

Asynchronous communication

- Failure stops at the failed component
- Slow response time stops at slow component
- Removes bottlenecks/single points of failure
- Consistency may lag
  - But can be eventually consistent

Asynchronous communication

- Failure stops at the failed component
- Slow response time stops at slow component
- Removes bottlenecks/single points of failure
- Consistency may lag
  - But can be eventually consistent
- Components are isolated

REST

REST

- REST is a synchronous messaging protocol

REST

- REST is a synchronous messaging protocol
  - Client sends a request, server sends a response

REST

- REST is a synchronous messaging protocol
  - Client sends a request, server sends a response
- Can be used for asynchronous messaging

REST

- REST is a synchronous messaging protocol
  - Client sends a request, server sends a response
- Can be used for asynchronous messaging
  - 204 No Content

REST

- REST is a synchronous messaging protocol
  - Client sends a request, server sends a response
- Can be used for asynchronous messaging
  - 204 No Content
- It's not really the problem

REST

- REST is a synchronous messaging protocol
  - Client sends a request, server sends a response
- Can be used for asynchronous messaging
  - 204 No Content
- It's not really the problem
  - But...

Danger!

Danger!

- We have a monolith called Chirper

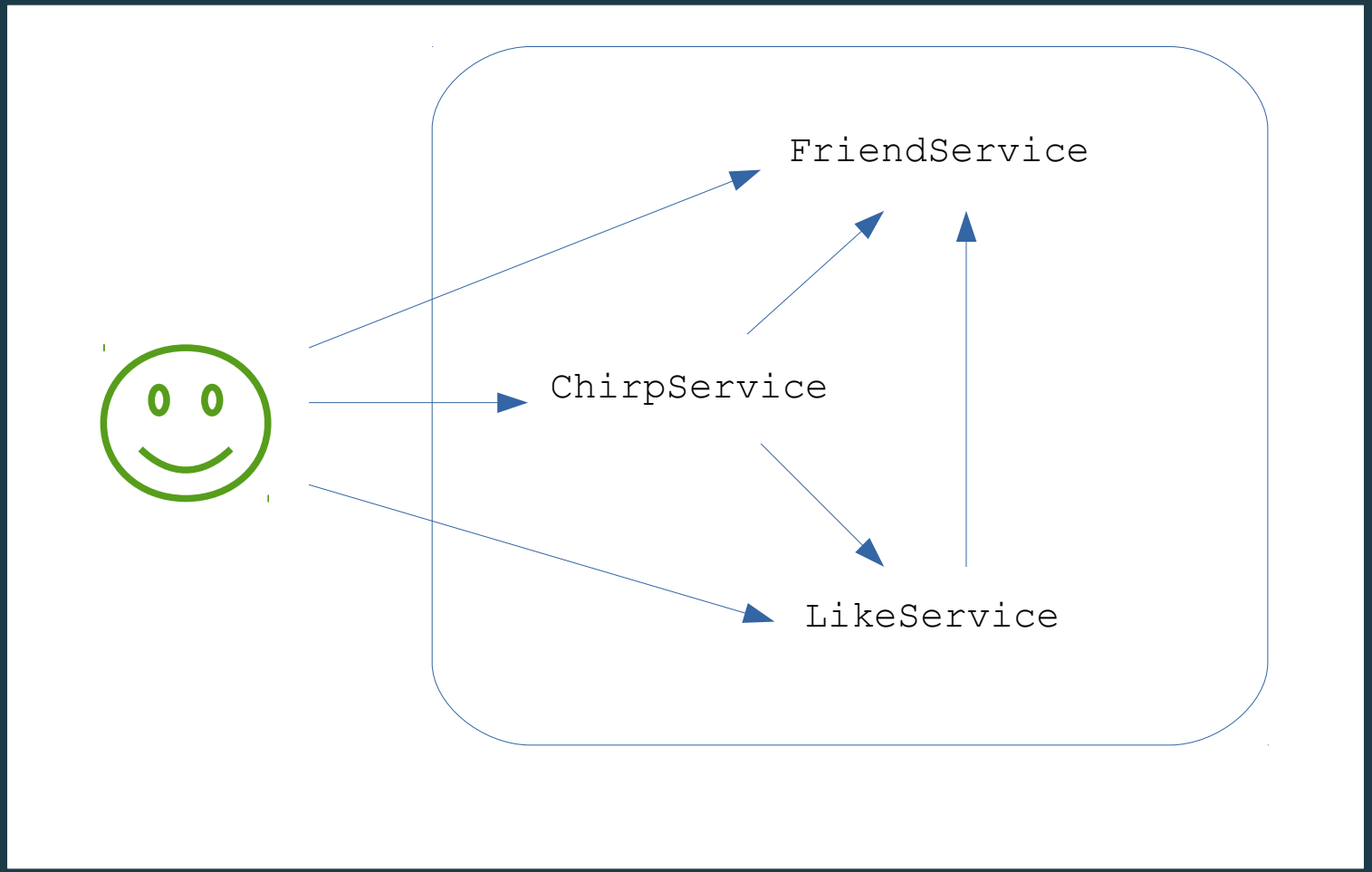
Danger!

- We have a monolith called Chirper
- Monoliths are bad!

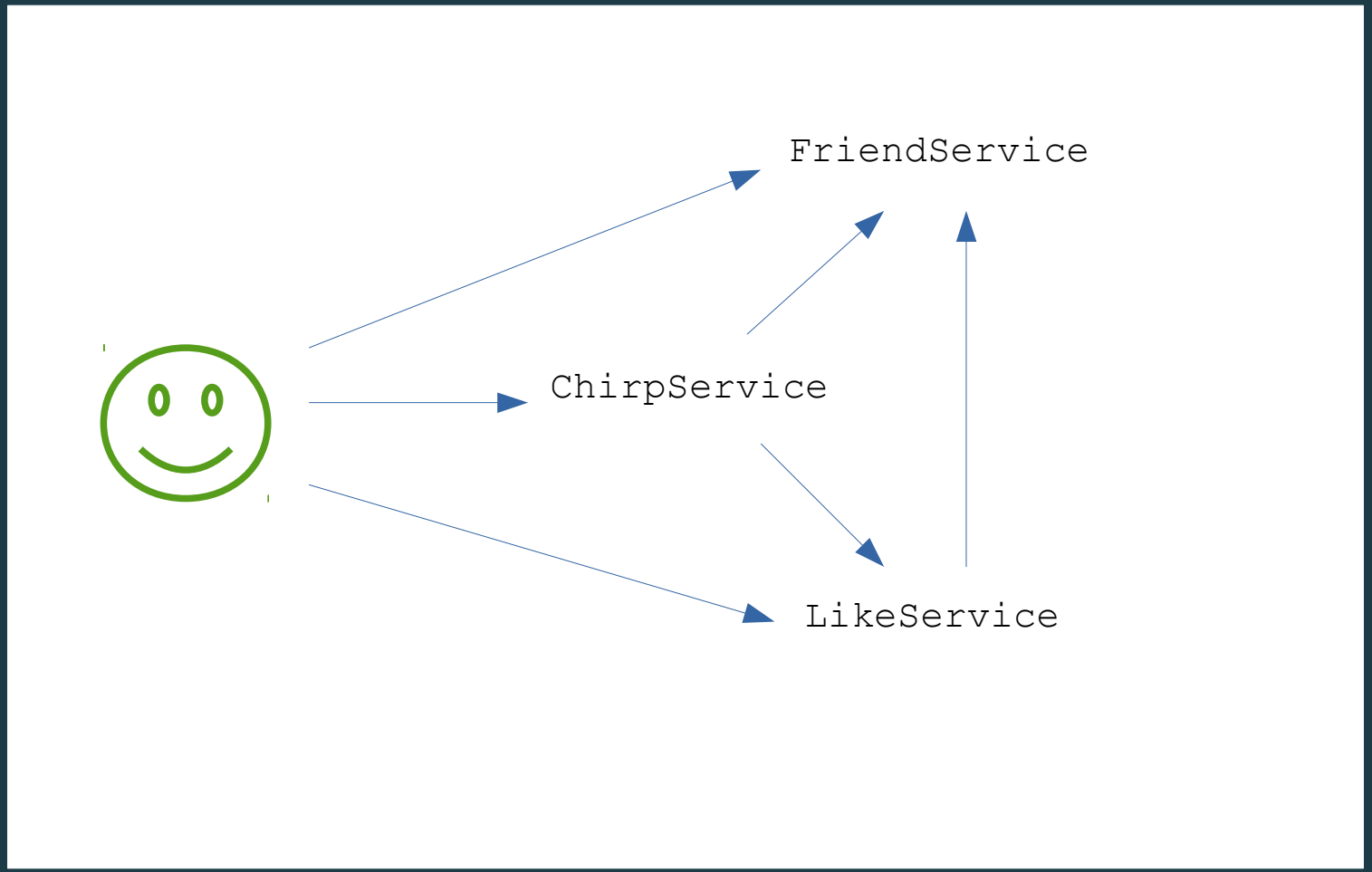
Danger!

- We have a monolith called Chirper
- Monoliths are bad!
- Let's convert to microservices

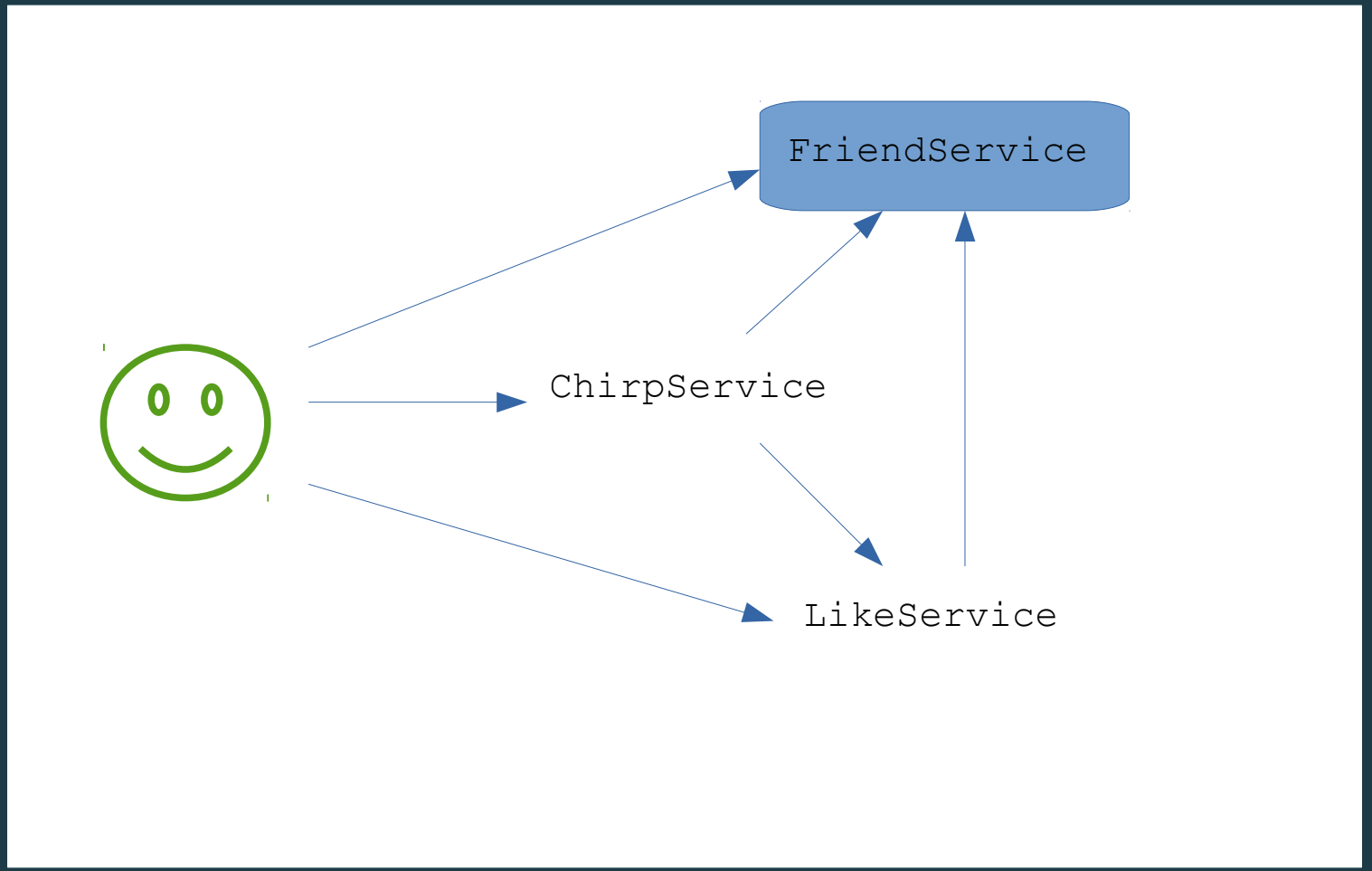
Chirper monolith



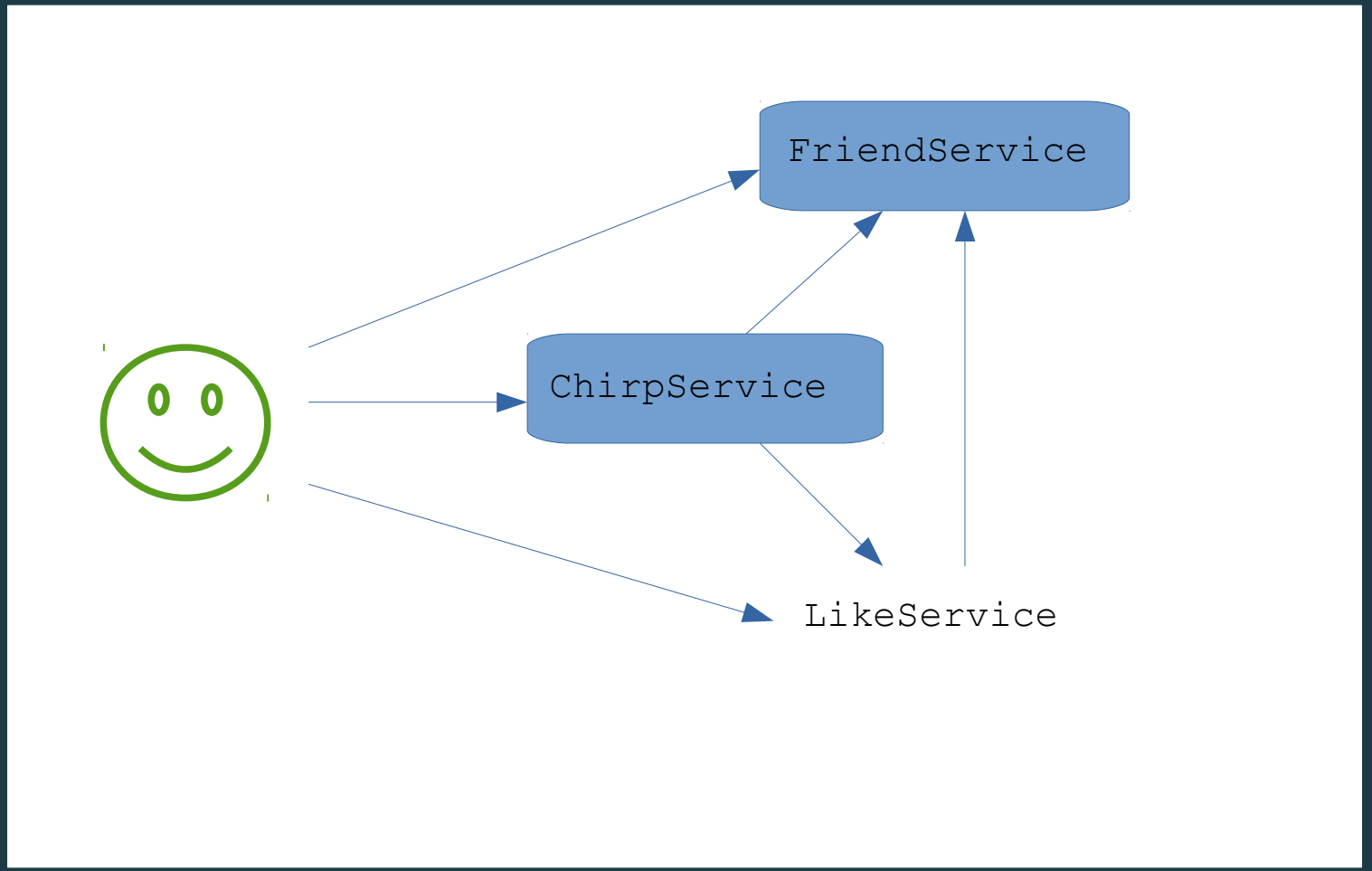
Chirper monolith



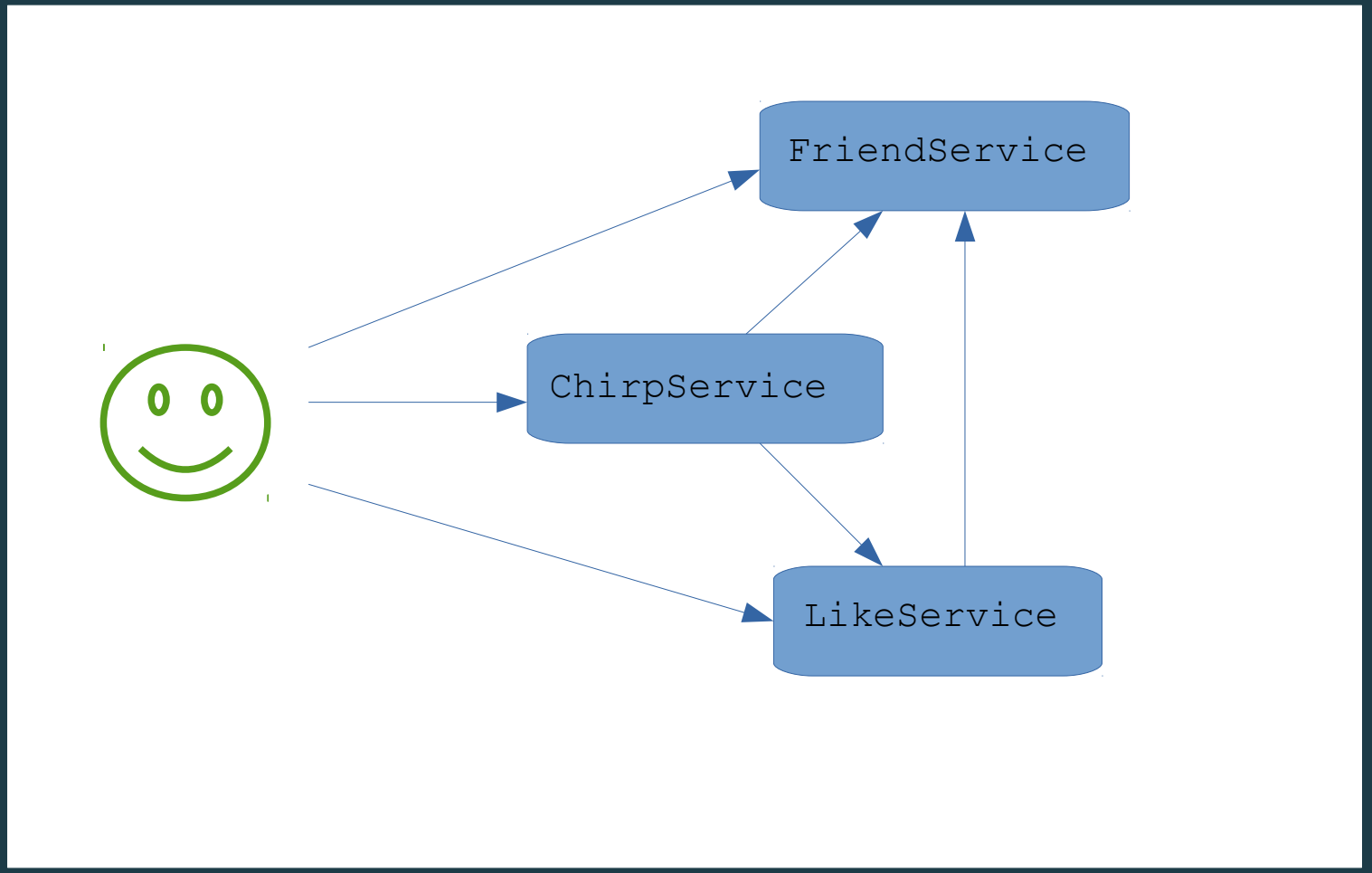
Chirper monolith



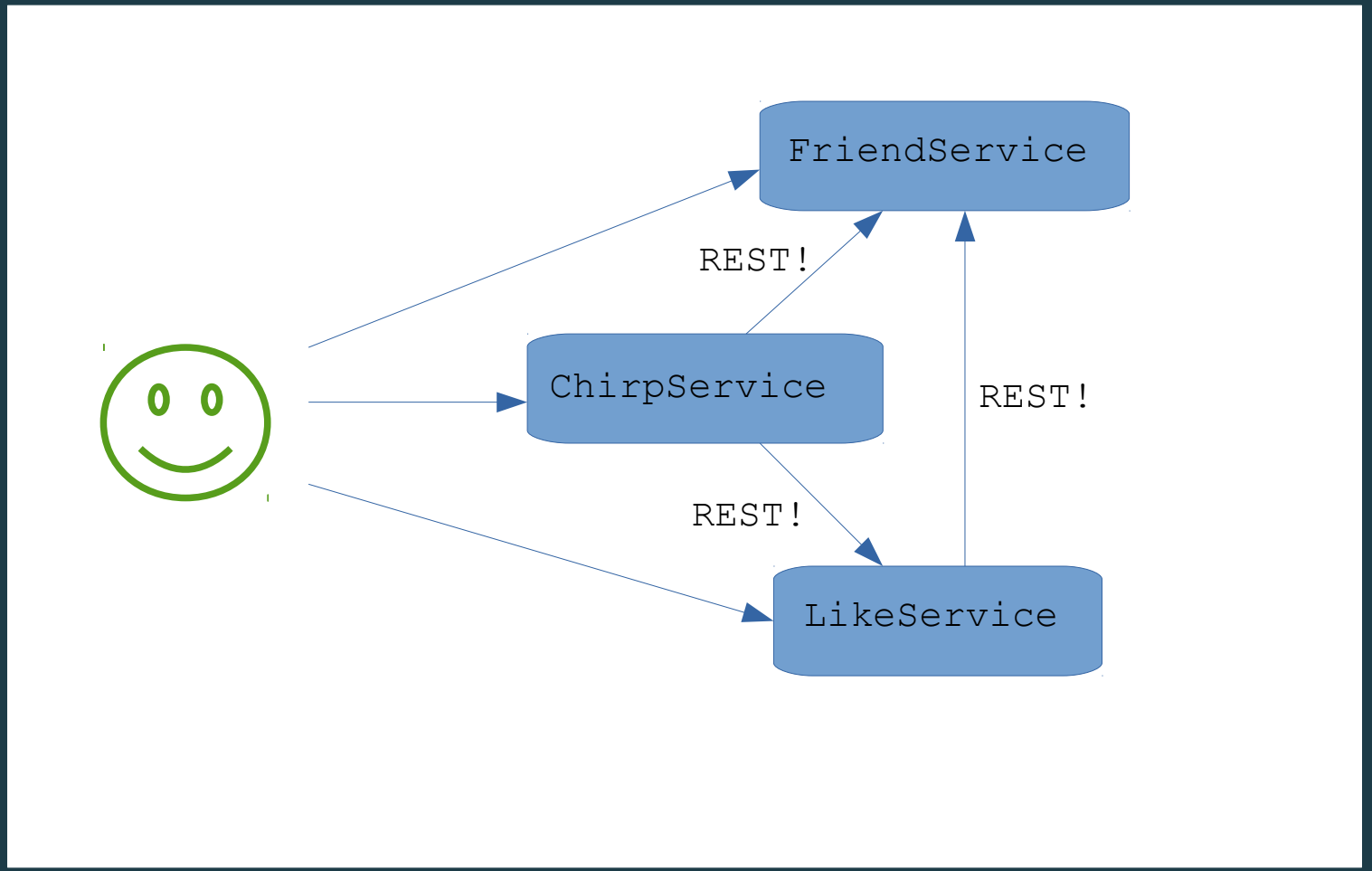
Chirper monolith



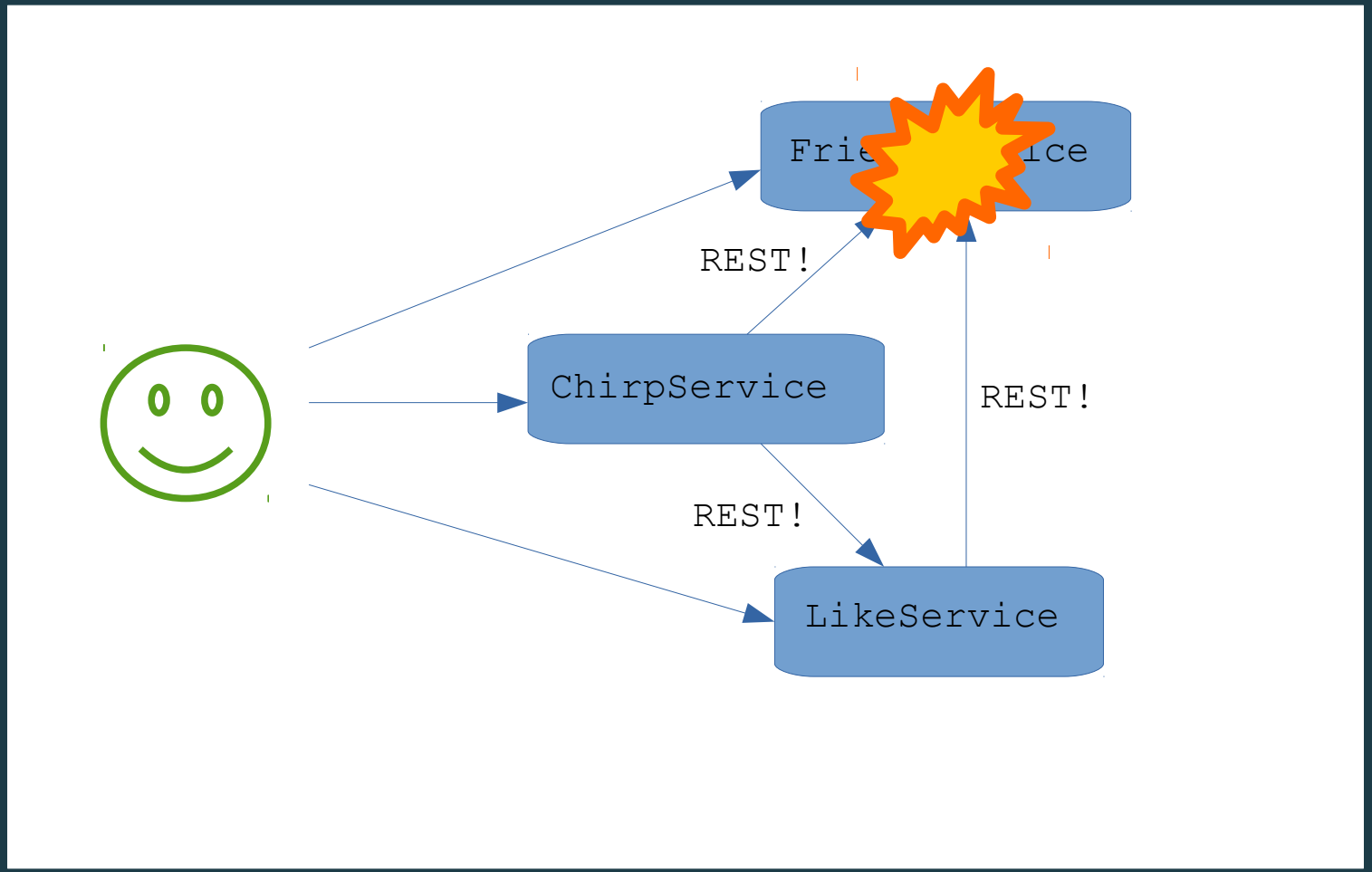
Chirper monolith



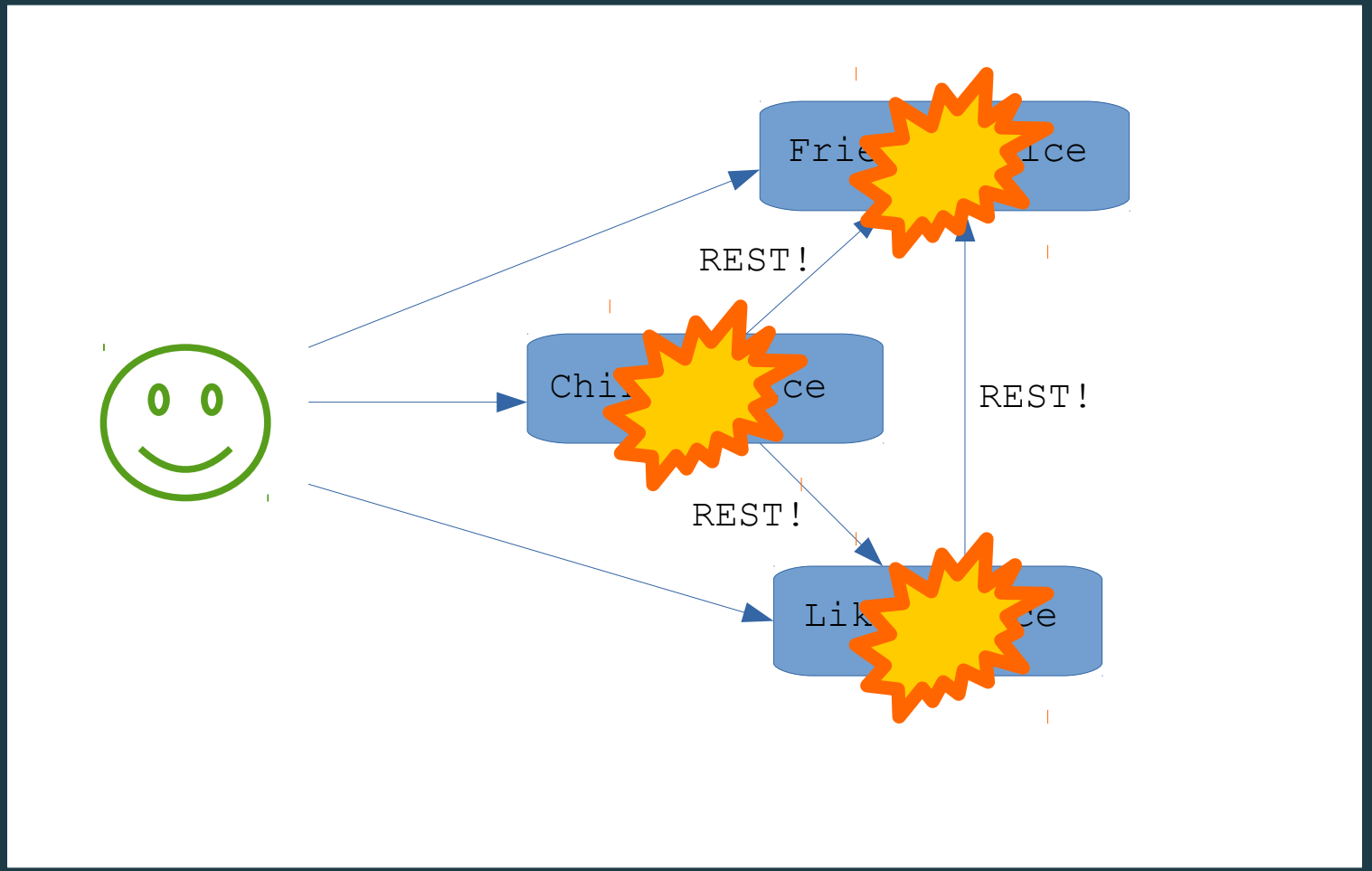
Chirper monolith



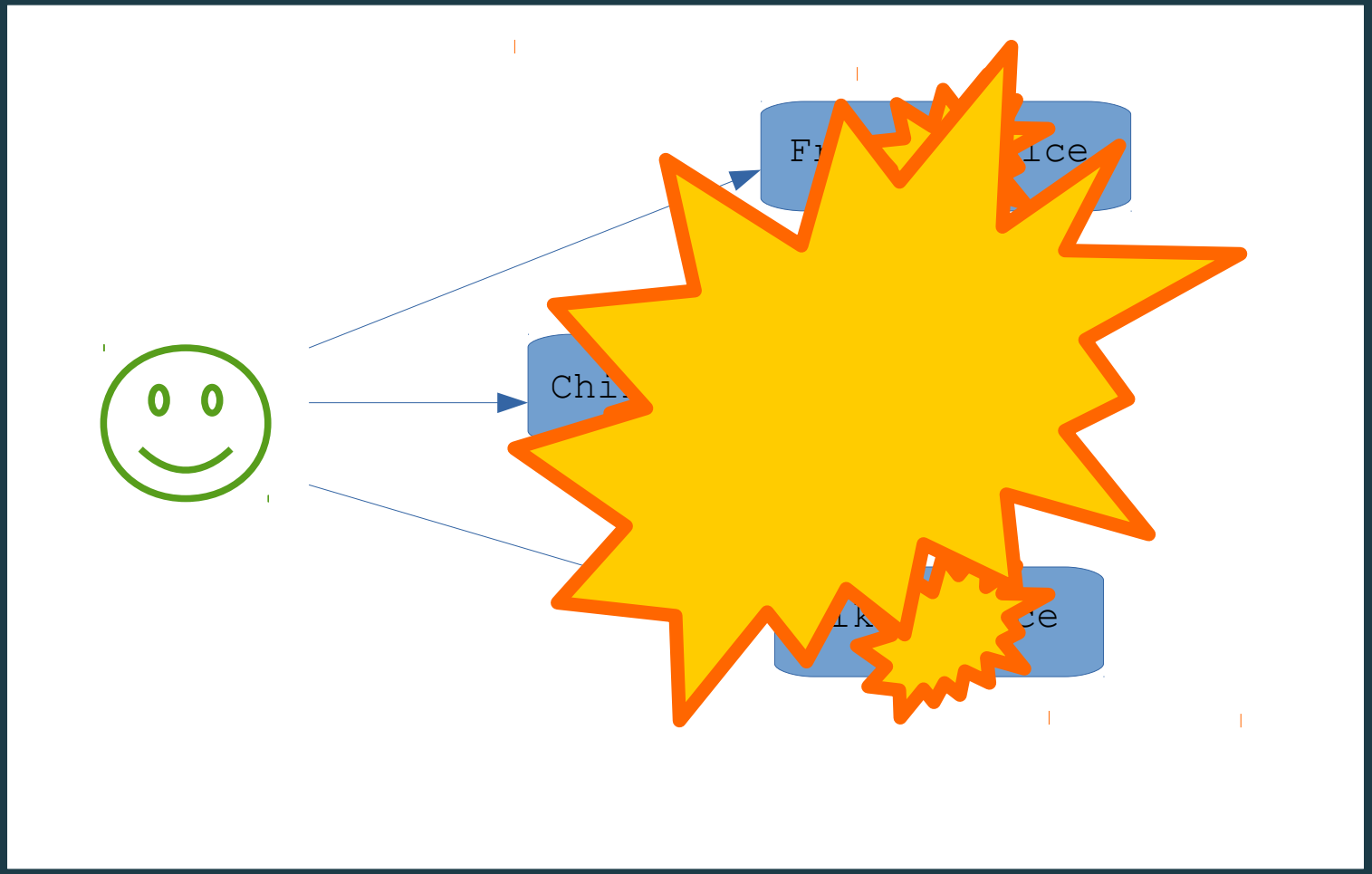
Chirper monolith



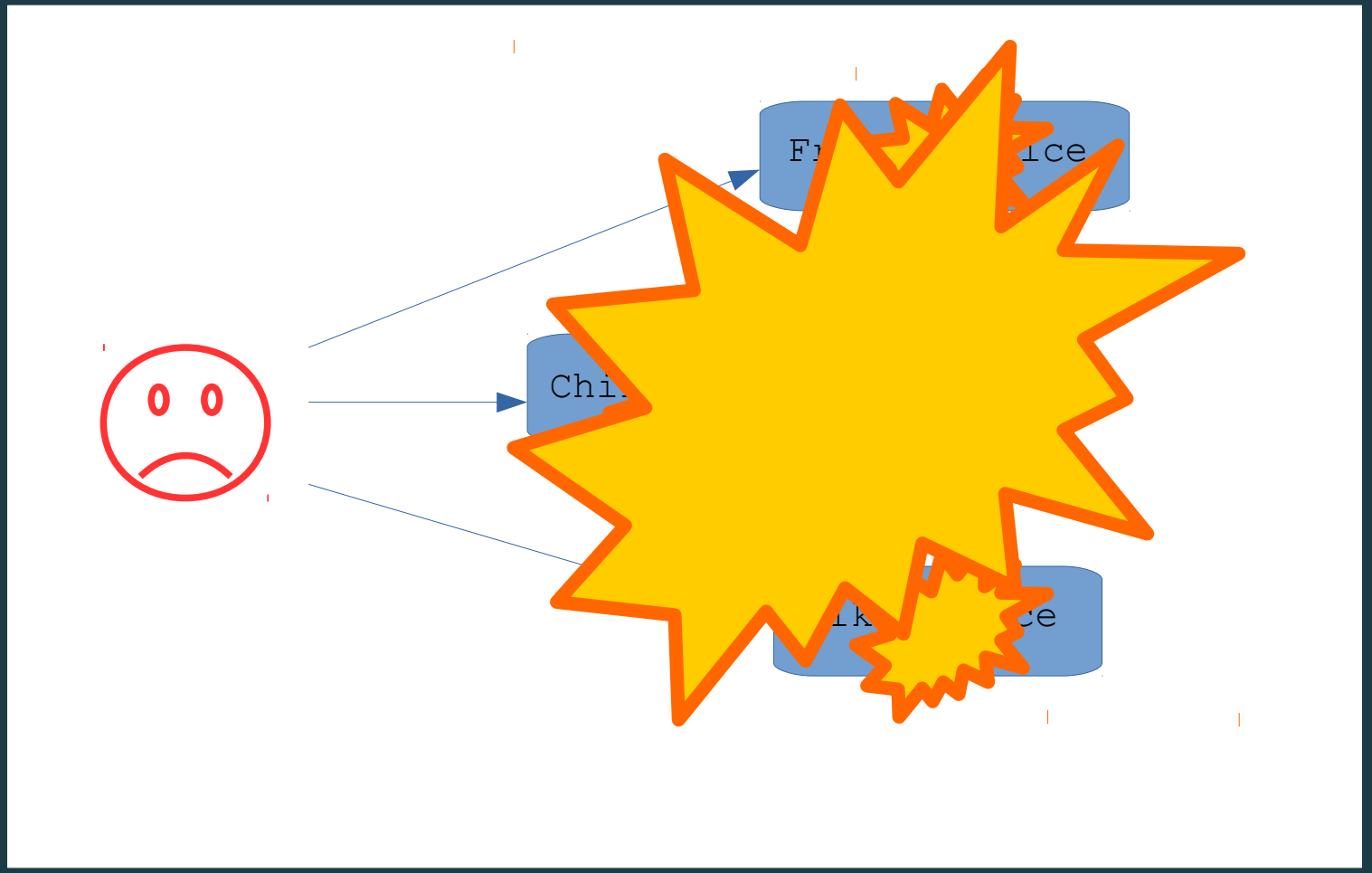
Chirper monolith



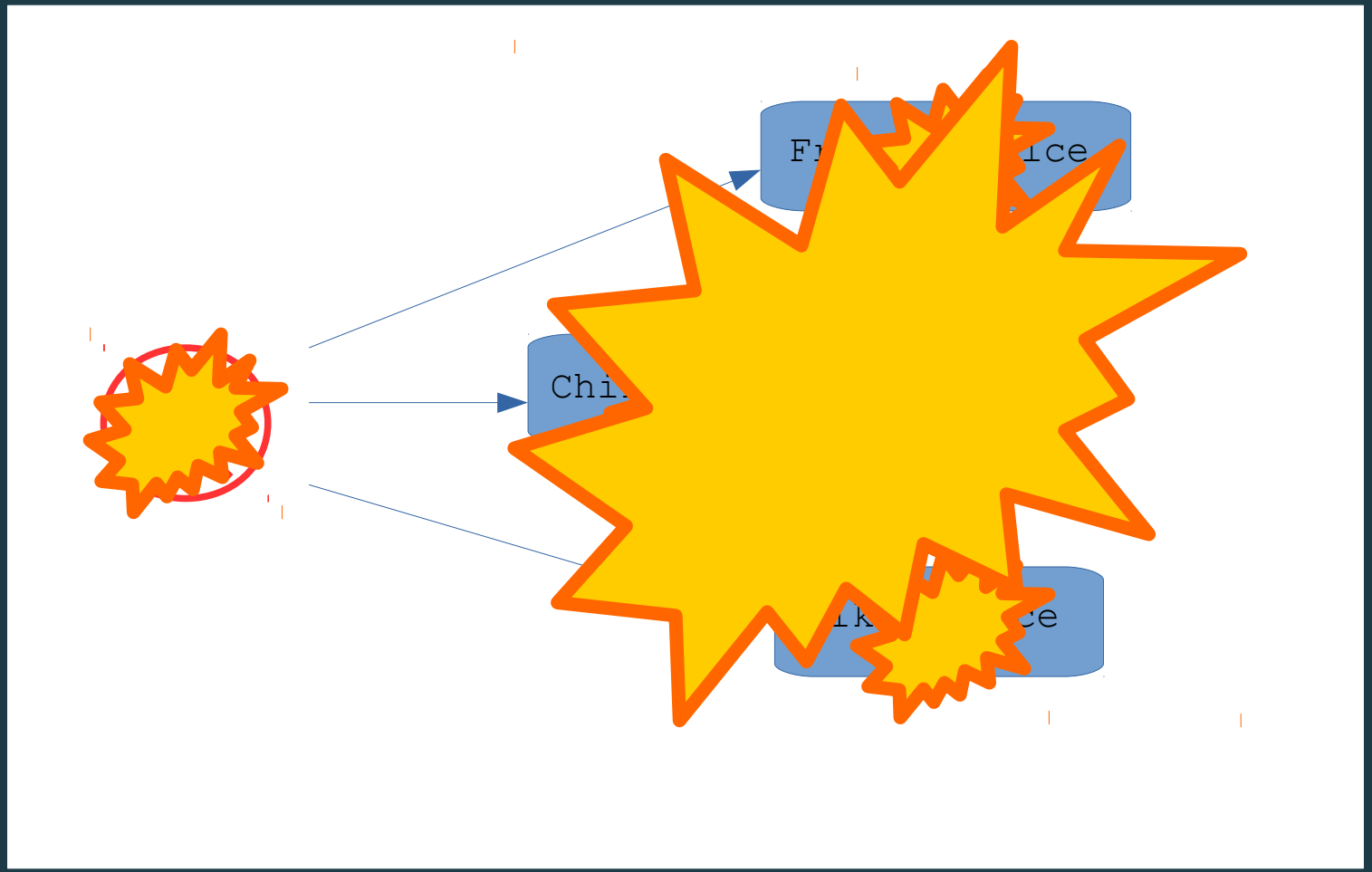
Chirper monolith



Chirper monolith



Chirper monolith



Monolith to microservices

Monolith to microservices

- Problem is that REST is our go to

Monolith to microservices

- Problem is that REST is our go to
  - Let's just replace method calls with REST calls!

Monolith to microservices

- Problem is that REST is our go to
  - Let's just replace method calls with REST calls!
- When moving to microservices

Monolith to microservices

- Problem is that REST is our go to
  - Let's just replace method calls with REST calls!
- When moving to microservices
  - Don't just do 1:1 service/interface replacement

Monolith to microservices

- Problem is that REST is our go to
  - Let's just replace method calls with REST calls!
- When moving to microservices
  - Don't just do 1:1 service/interface replacement
  - Design for asynchronous architecture

Asynchronous in practice

Asynchronous in practice

- Let's look at two approaches

Asynchronous in practice

- Let's look at two approaches
  1) Denormalize and push data

Asynchronous in practice

- Let's look at two approaches
  1) Denormalize and push data
  2) Communicate through the client

Dernormalize and push

Dernormalize and push

- Services store all data they need

Dernormalize and push

- Services store all data they need
  - This may mean duplicating, aka denormalizing data

Dernormalize and push

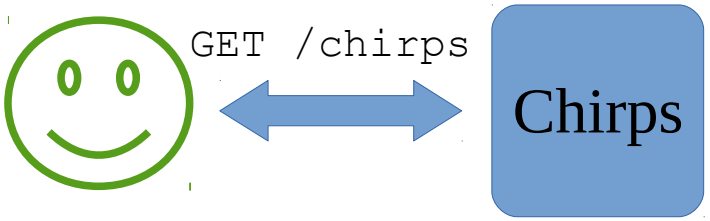
- Services store all data they need
  - This may mean duplicating, aka denormalizing data
- Consistency needs to be addressed

Dernormalize and push

- Services store all data they need
  - This may mean duplicating, aka denormalizing data
- Consistency needs to be addressed
  - One approach is at least once messaging and idempotency

Chirper Likes

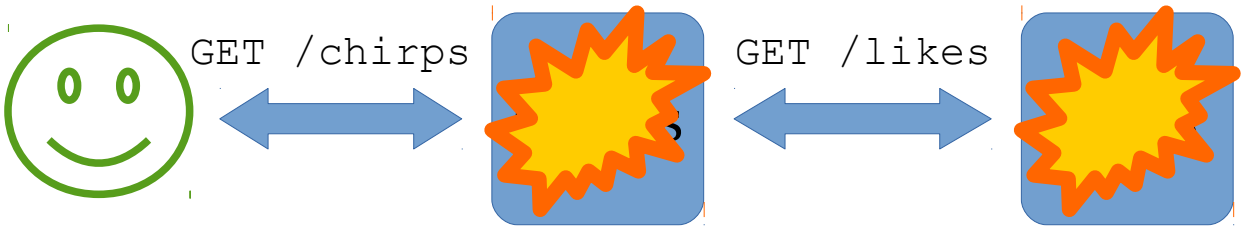
Chirper Likes



Chirper Likes

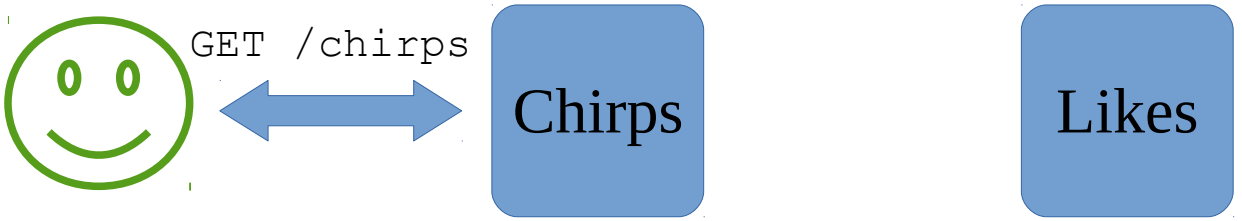
Chirper Likes

Chirper Likes



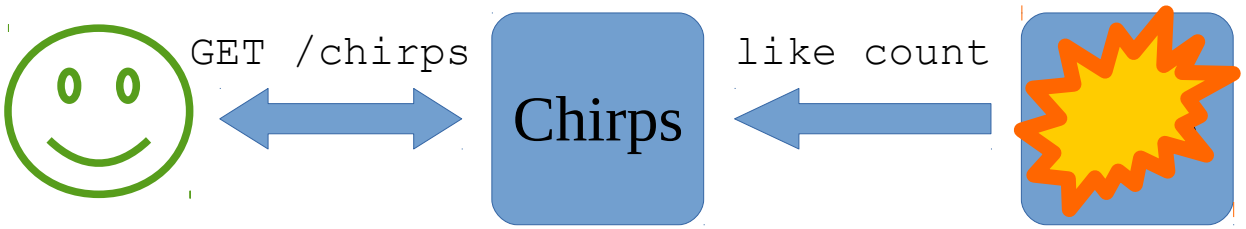
Chirper Likes

Chirper Likes

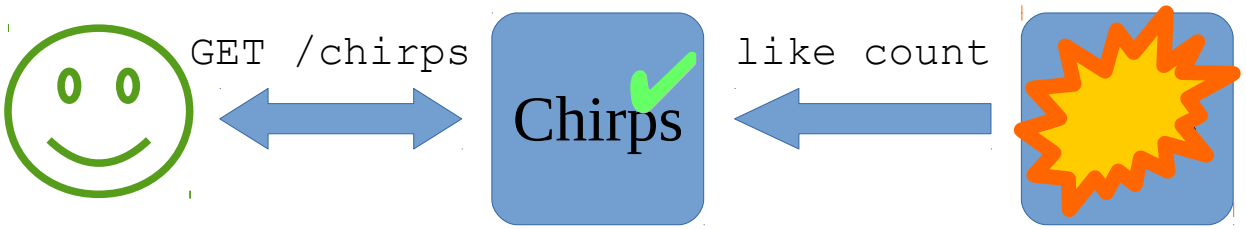


Chirper Likes

Chirper Likes



Chirper Likes



Demo

Demo

- Chirper implemented using Lagom

Demo

- Chirper implemented using Lagom
  - Reactive microservices framework for the JVM

Demo

- Chirper implemented using Lagom
  - Reactive microservices framework for the JVM
  - First class support for asynchronous messaging

Demo

- Chirper implemented using Lagom
  - Reactive microservices framework for the JVM
  - First class support for asynchronous messaging
  - High productivity development environment

Demo

- Chirper implemented using Lagom
  - Reactive microservices framework for the JVM
  - First class support for asynchronous messaging
  - High productivity development environment
  - Guides you all the way to production

Communicate through client

Communicate through client

- Client can cache data

Communicate through client

- Client can cache data
  - Mobile clients in particular

Communicate through client

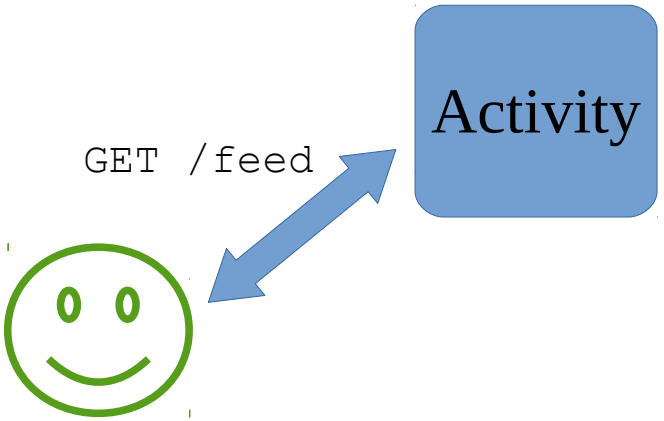
- Client can cache data
  - Mobile clients in particular
  - Browsers with localStorage

Communicate through client

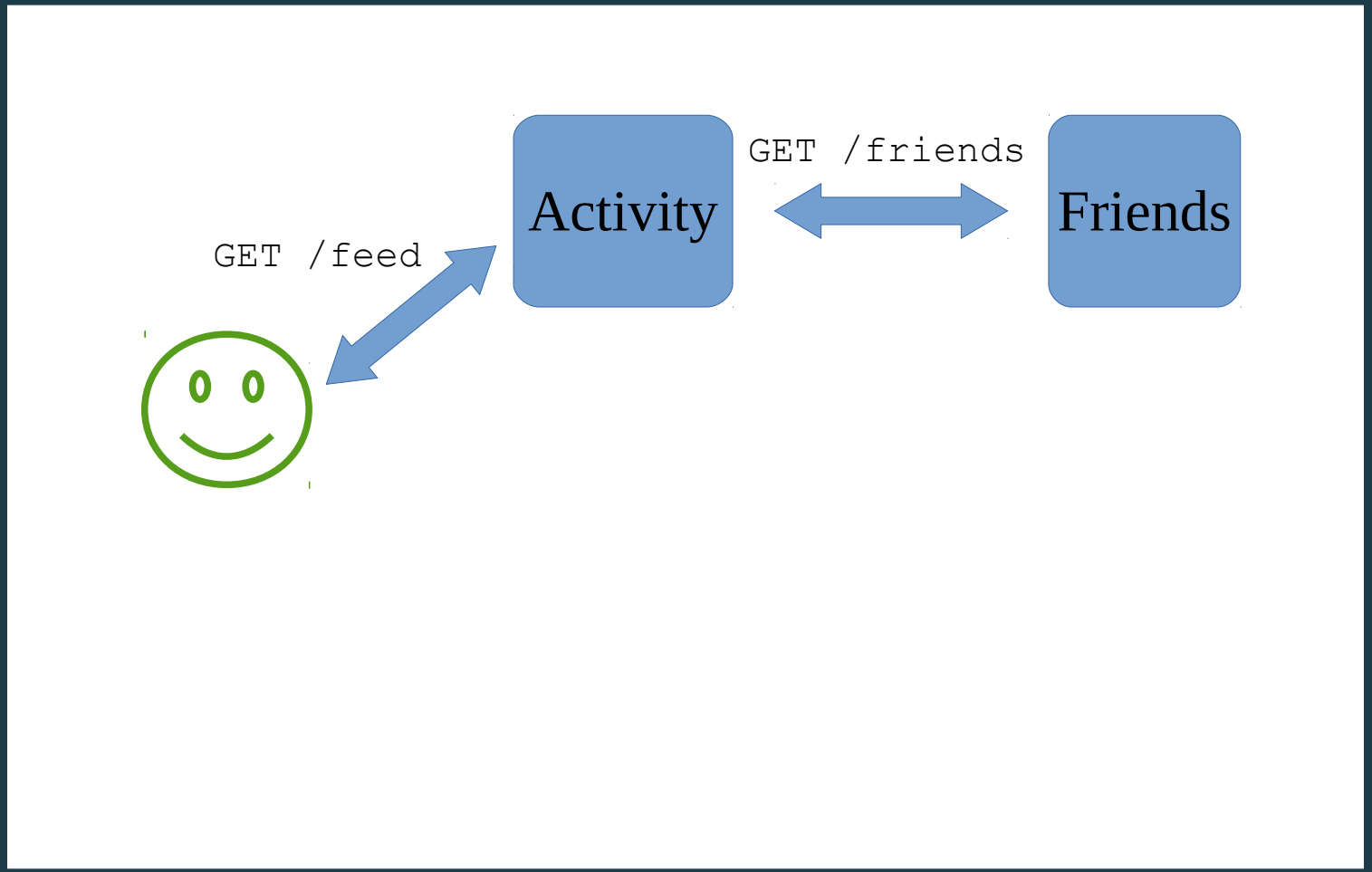
- Client can cache data
  - Mobile clients in particular
  - Browsers with localStorage
- Client can use own cache as fallback

Chirper Activity Stream

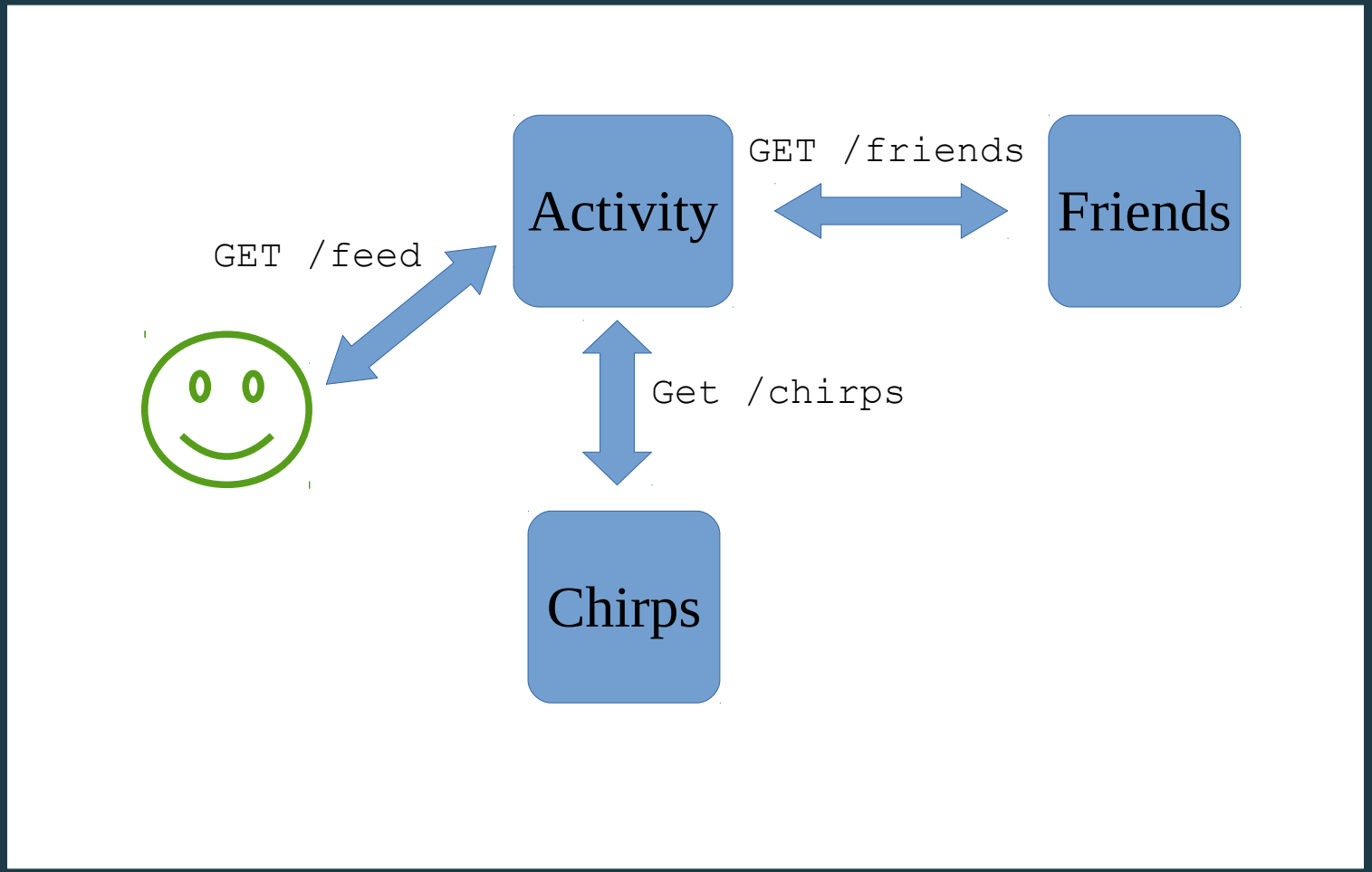
Chirper Activity Stream



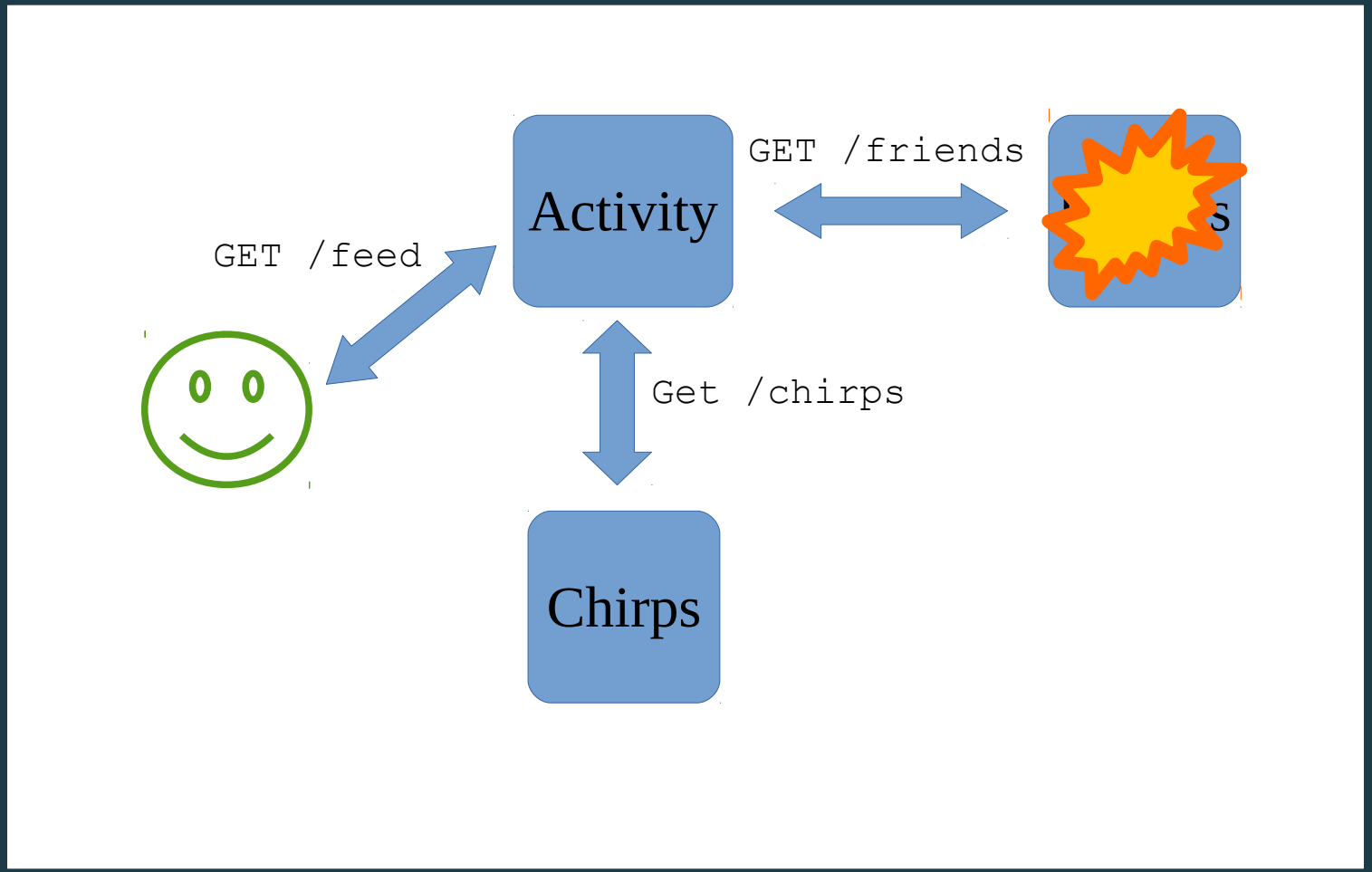
Chirper Activity Stream



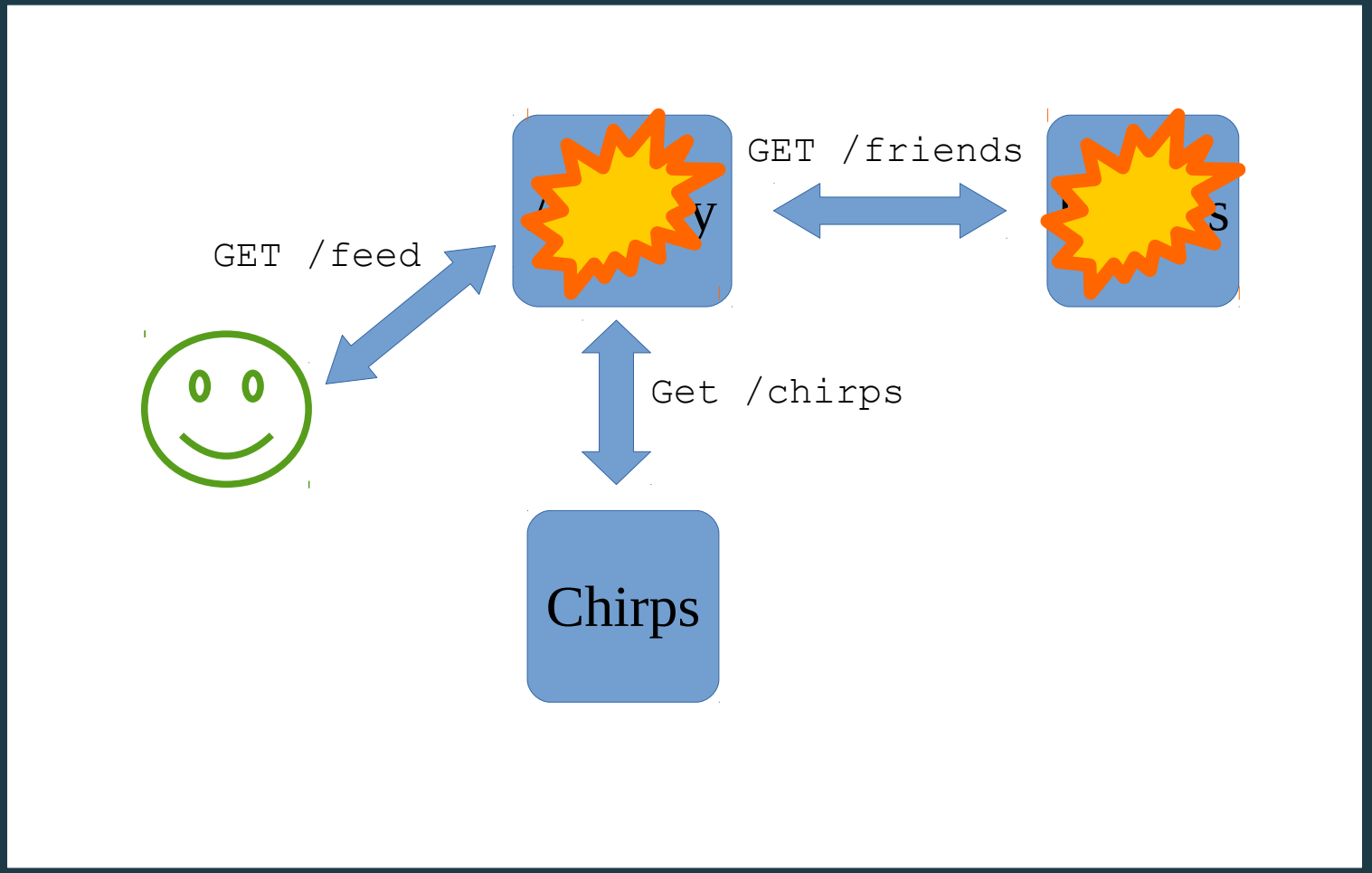
Chirper Activity Stream



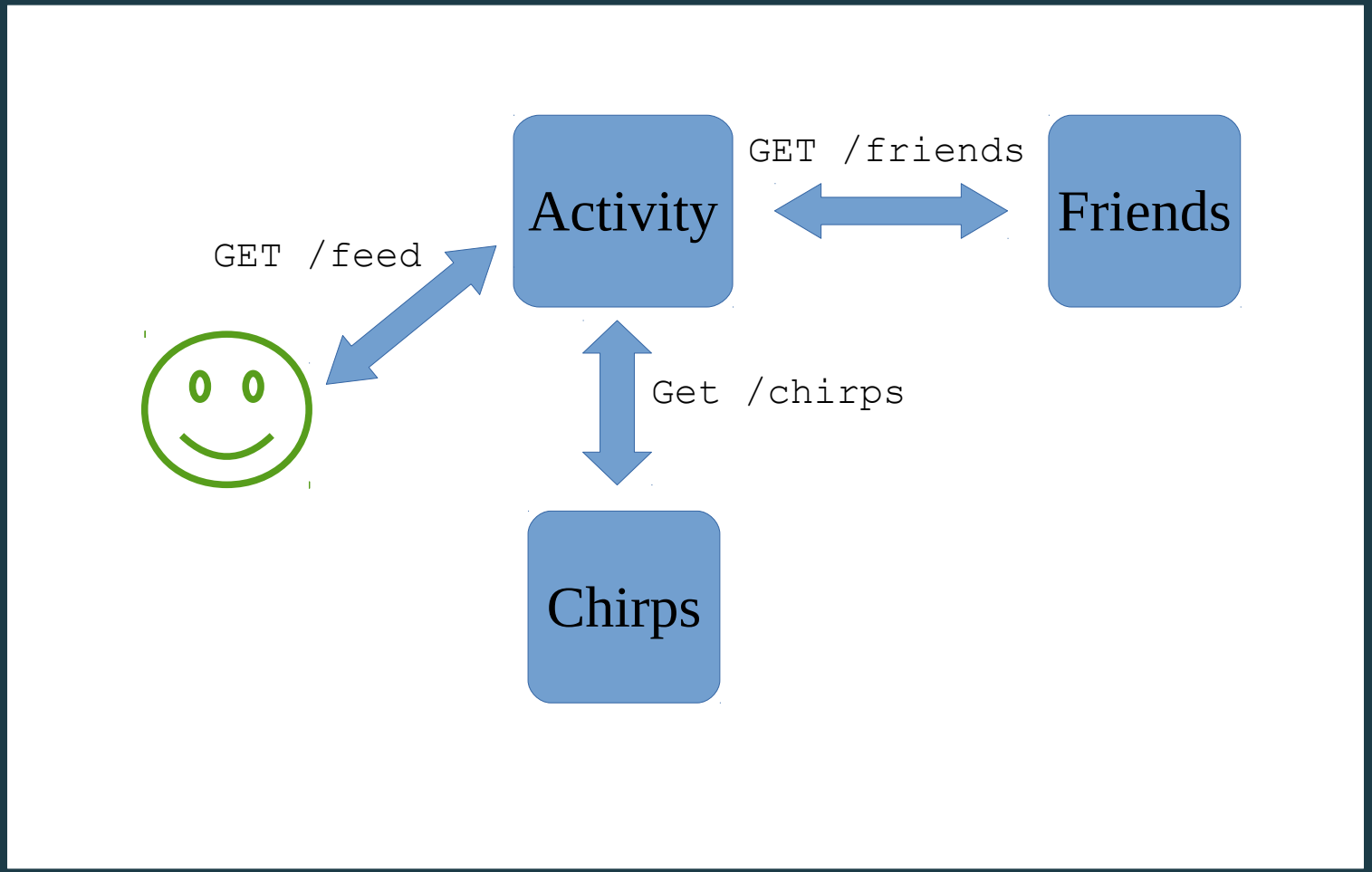
Chirper Activity Stream



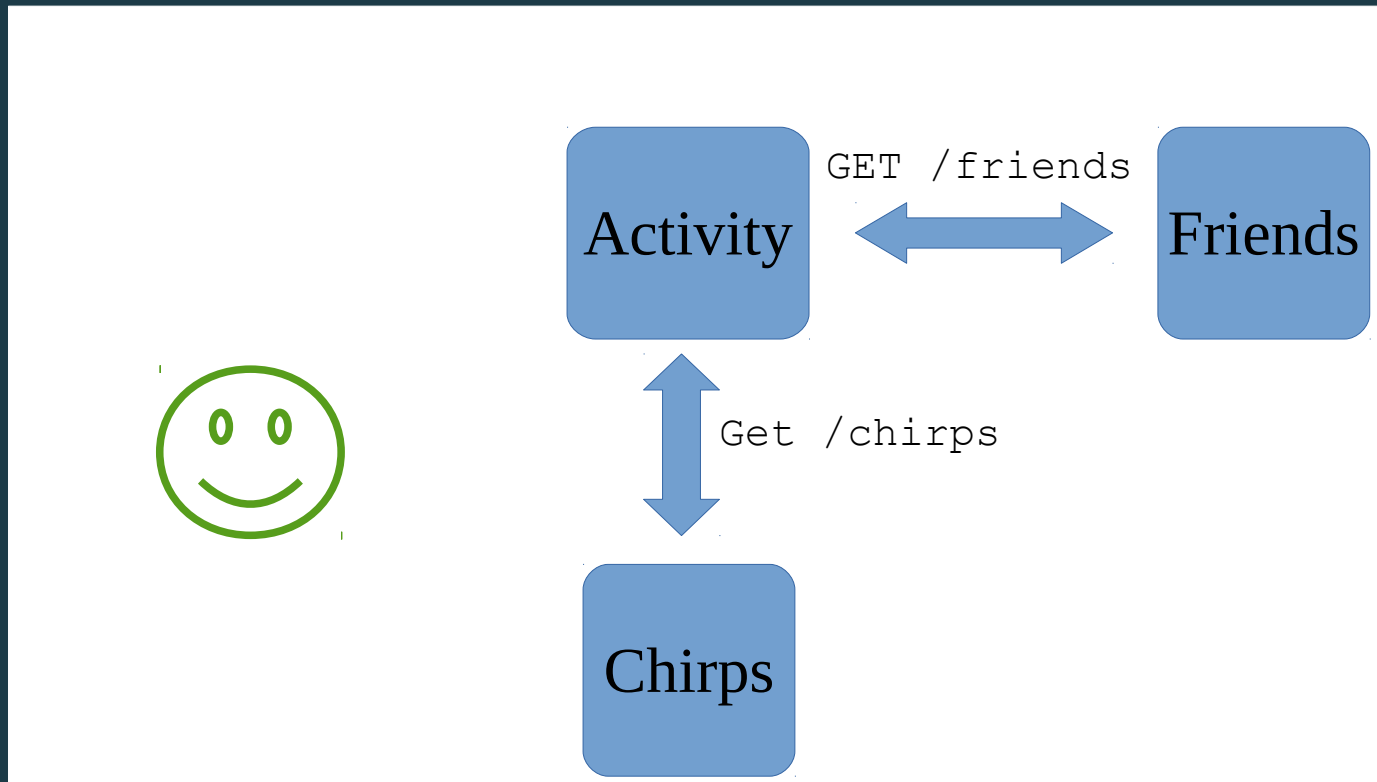
Chirper Activity Stream



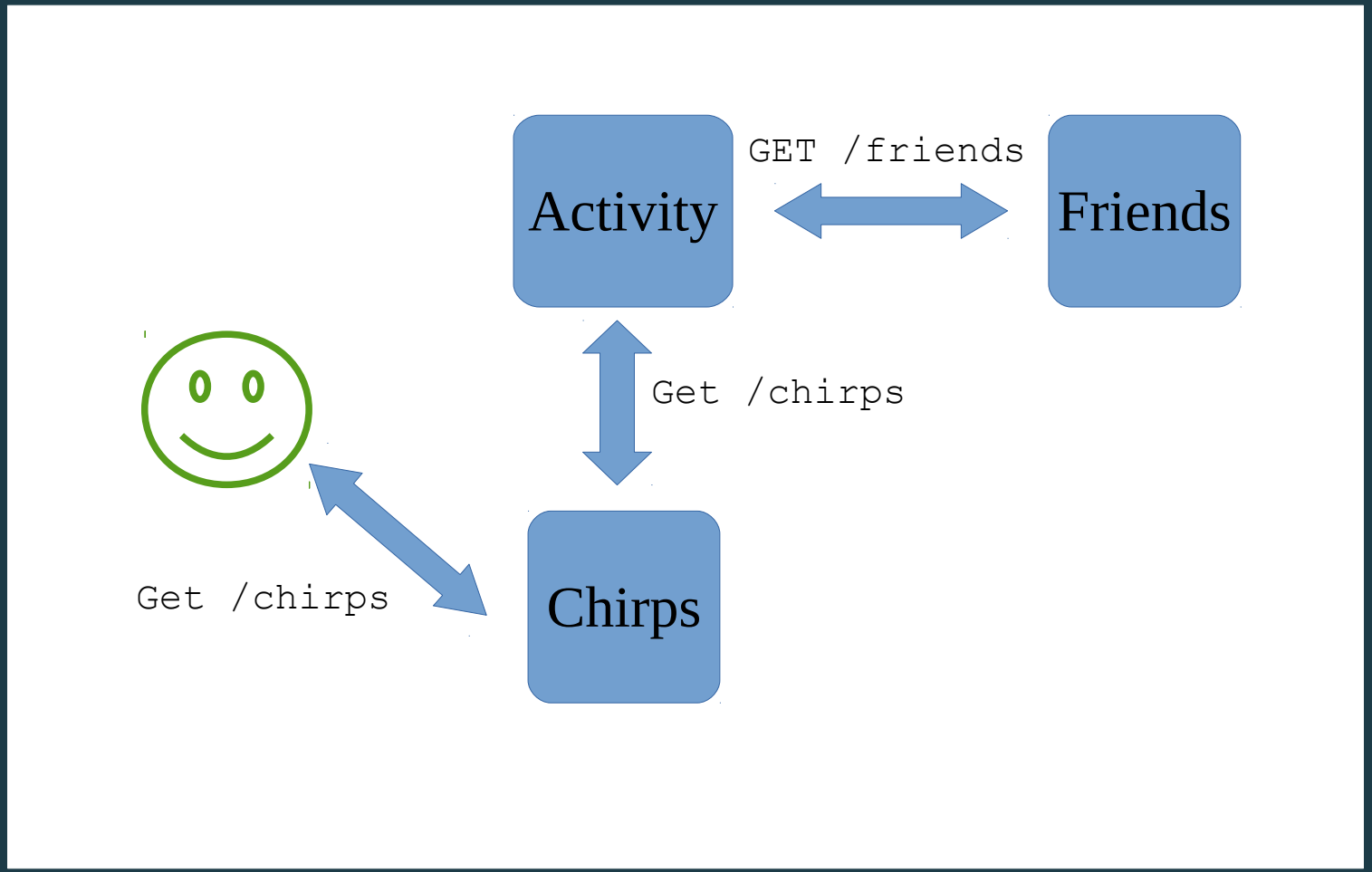
Chirper Activity Stream



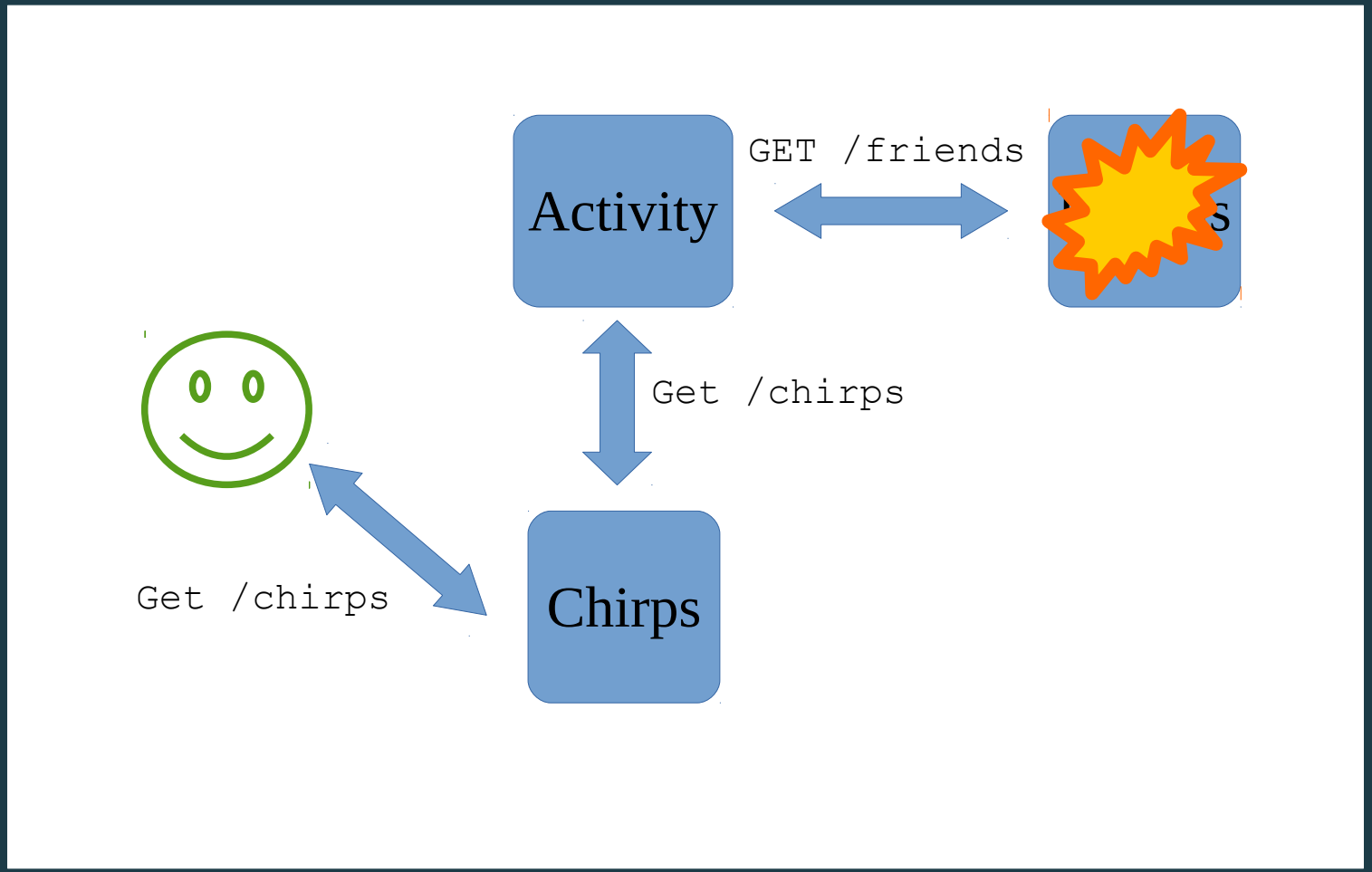
Chirper Activity Stream



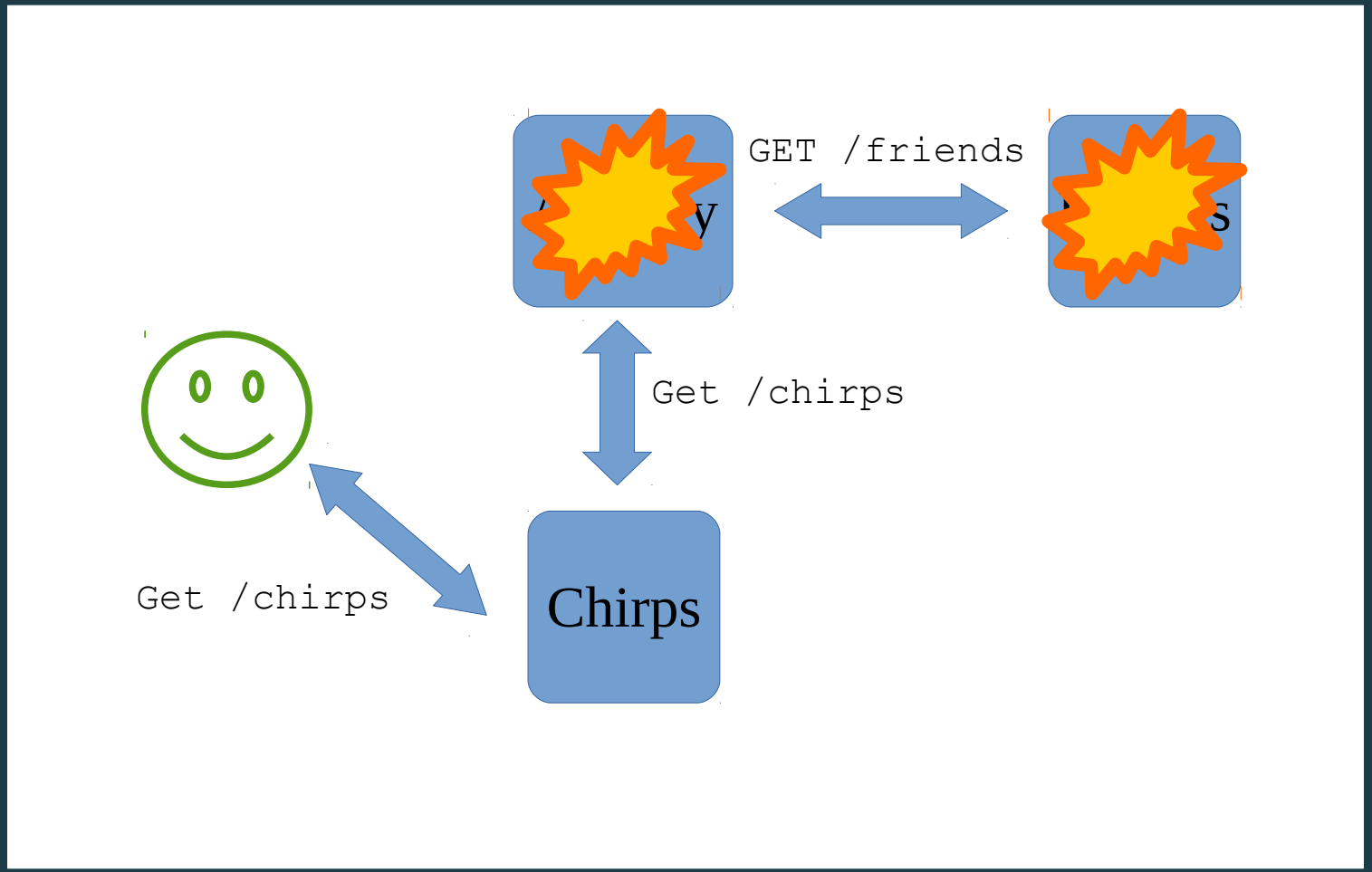
Chirper Activity Stream



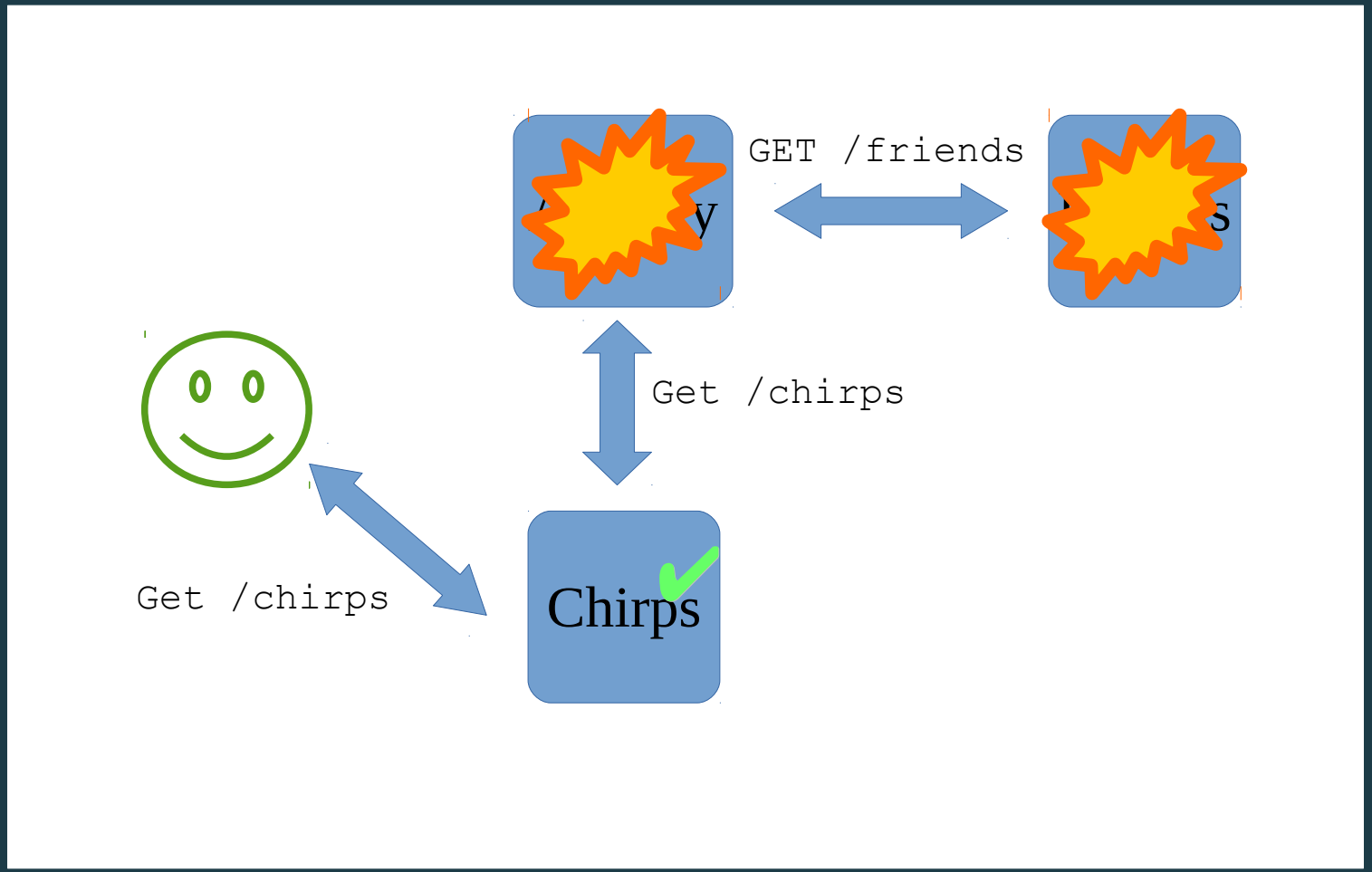
Chirper Activity Stream



Chirper Activity Stream



Chirper Activity Stream



Demo

Client authorization

Client authorization

- Communicate authorization through client

Client authorization

- Communicate authorization through client
  - Use authorization tokens, eg JWT

Client authorization

- Communicate authorization through client
  - Use authorization tokens, eg JWT
  - Load token with entity – eg friends

Client authorization

- Communicate authorization through client
  - Use authorization tokens, eg JWT
  - Load token with entity – eg friends
  - Send that token with actions

Client authorization

- Communicate authorization through client
  - Use authorization tokens, eg JWT
  - Load token with entity – eg friends
  - Send that token with actions
  - Receiving service authorizes using token

Next steps

- Try out Lagom
  - https://lightbend.com/lagom
- Read Jonas Bonér's *Reactive Services Architecture*
  - https://lightbend.com/reactive-microservices-architecture
- Check out this presentation
  - https://github.com/jroper/rethinking-rest