

# MONOLITH TO REACTIVE

*It's all about architecture*

James Roper

@jroper



Lightbend

# AGENDA

# AGENDA

- Identify pitfalls of monolith conversions

# AGENDA

- Identify pitfalls of monolith conversions
- Architect reactive solutions

# AGENDA

- Identify pitfalls of monolith conversions
- Architect reactive solutions
- See Lagom in action

# AGENDA

- Identify pitfalls of monolith conversions
- Architect reactive solutions
- See Lagom in action
- Live coding!

# LAGOM AUCTION

# LAGOM AUCTION

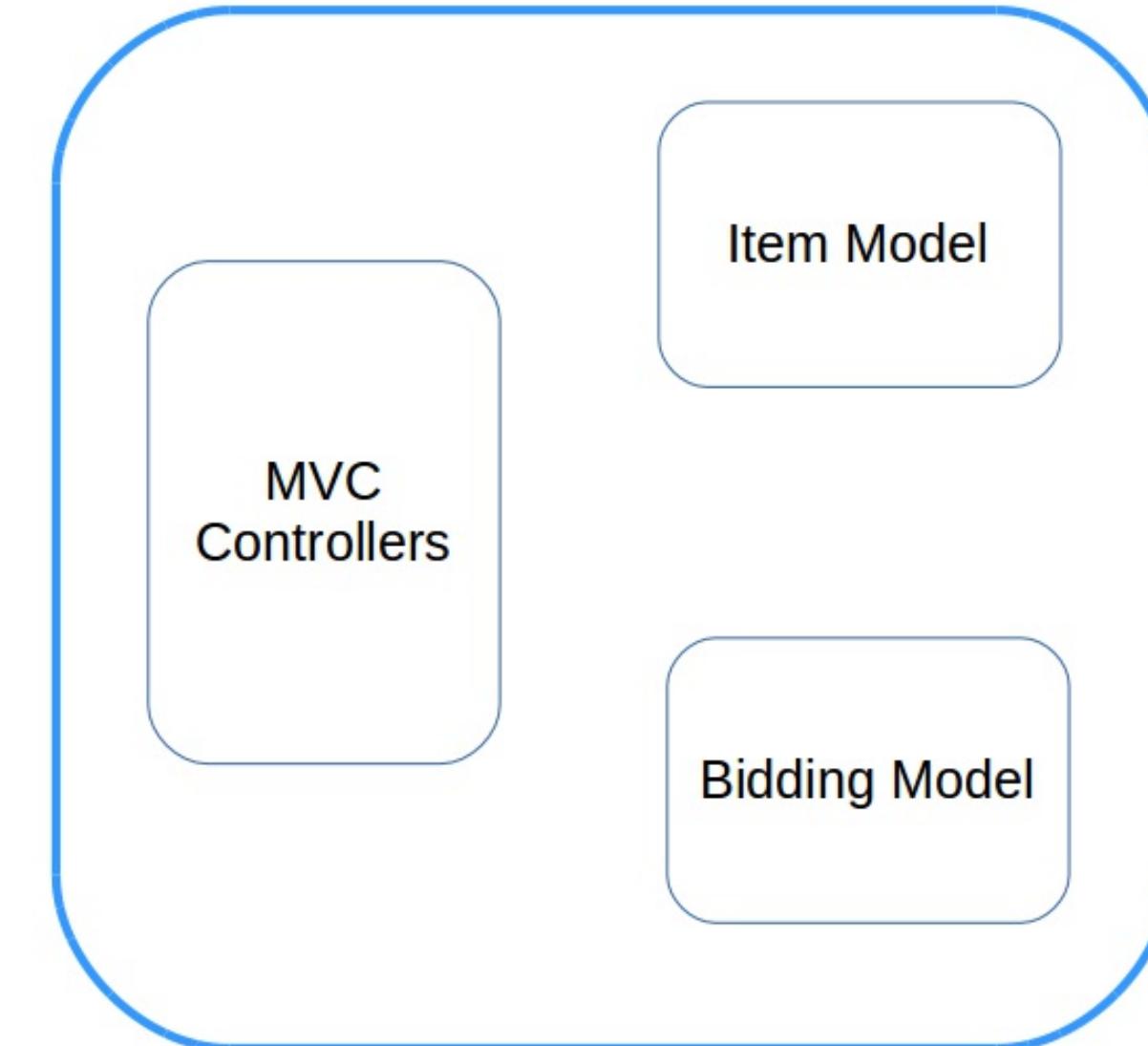
- ebay clone

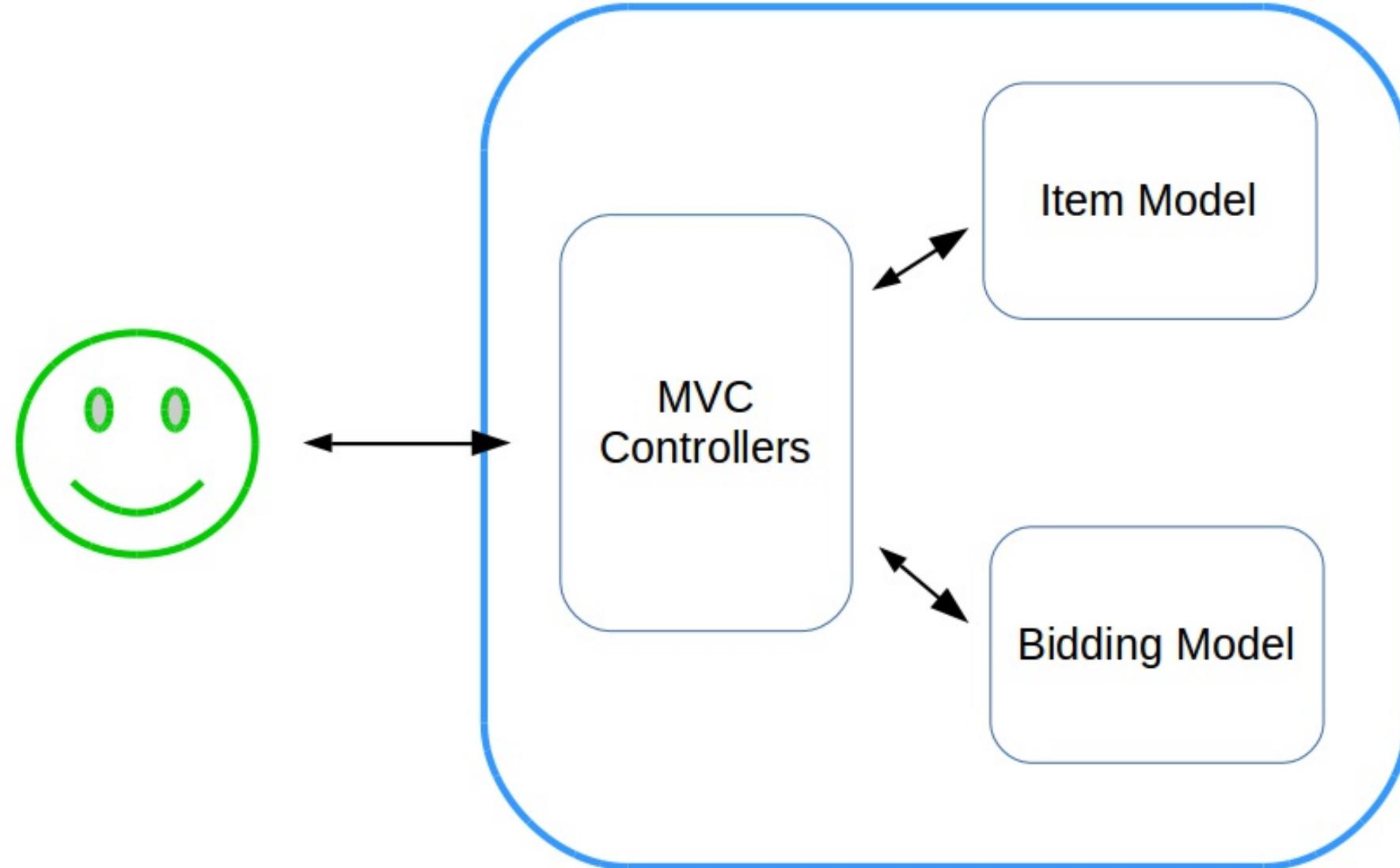
# LAGOM AUCTION

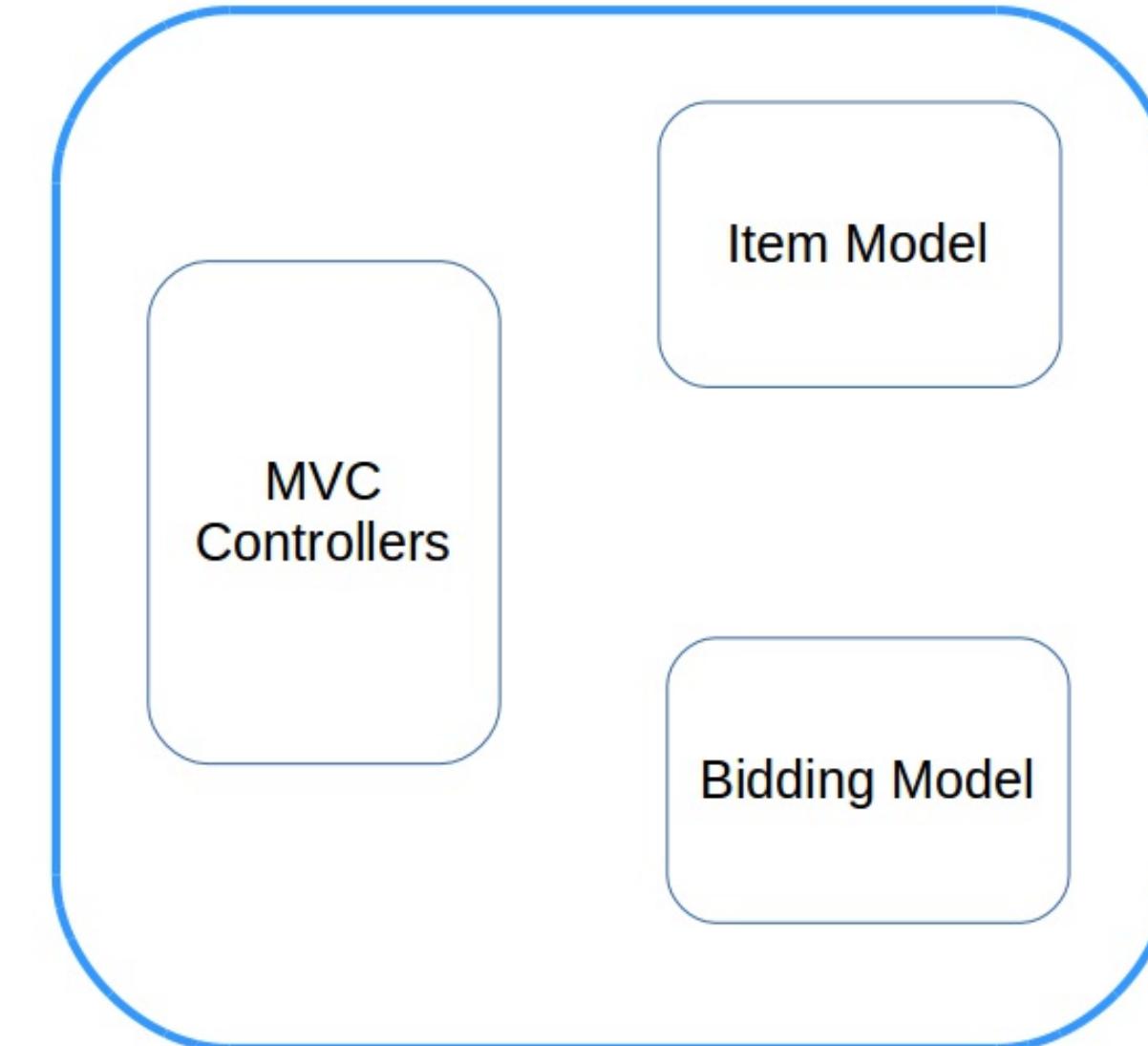
- ebay clone
- Was a monolith, converted to microservices

# LAGOM AUCTION

- ebay clone
- Was a monolith, converted to microservices
- Will one day overtake ebay!









MVC  
Controllers

Item Model

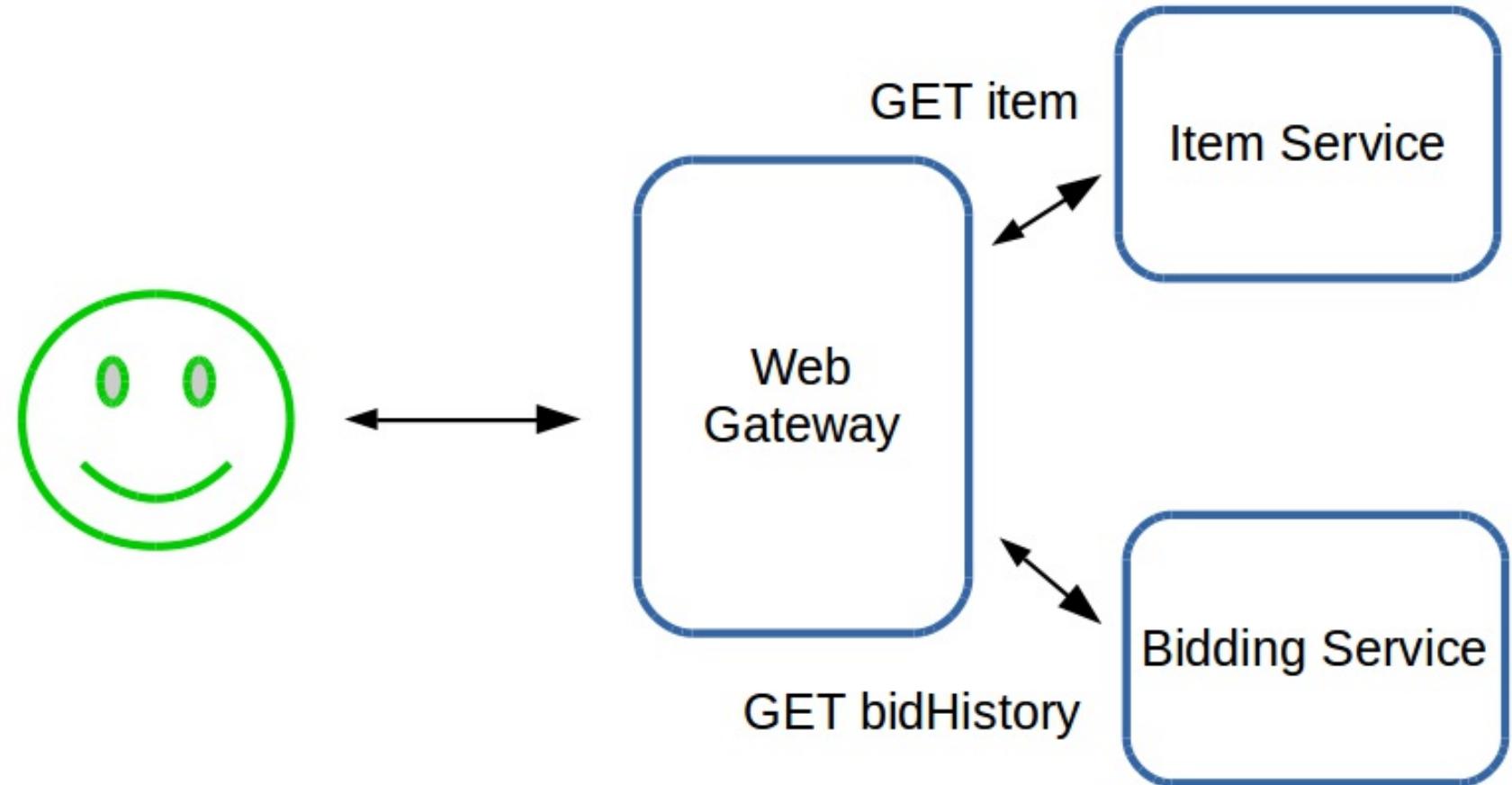
Bidding Model



Web  
Gateway

Item Model

Bidding Model



# WHAT IF SOMETHING GOES WRONG?

# WHAT IF SOMETHING GOES WRONG?

- Microservices means more moving parts

# WHAT IF SOMETHING GOES WRONG?

- Microservices means more moving parts
  - More chance for failure

# WHAT IF SOMETHING GOES WRONG?

- Microservices means more moving parts
  - More chance for failure
  - More chance for inconsistency

# SYNCHRONOUS COMMUNICATION

# SYNCHRONOUS COMMUNICATION

**synchronous** *adj.* - existing or occurring at the same time.

# SYNCHRONOUS COMMUNICATION

# SYNCHRONOUS COMMUNICATION

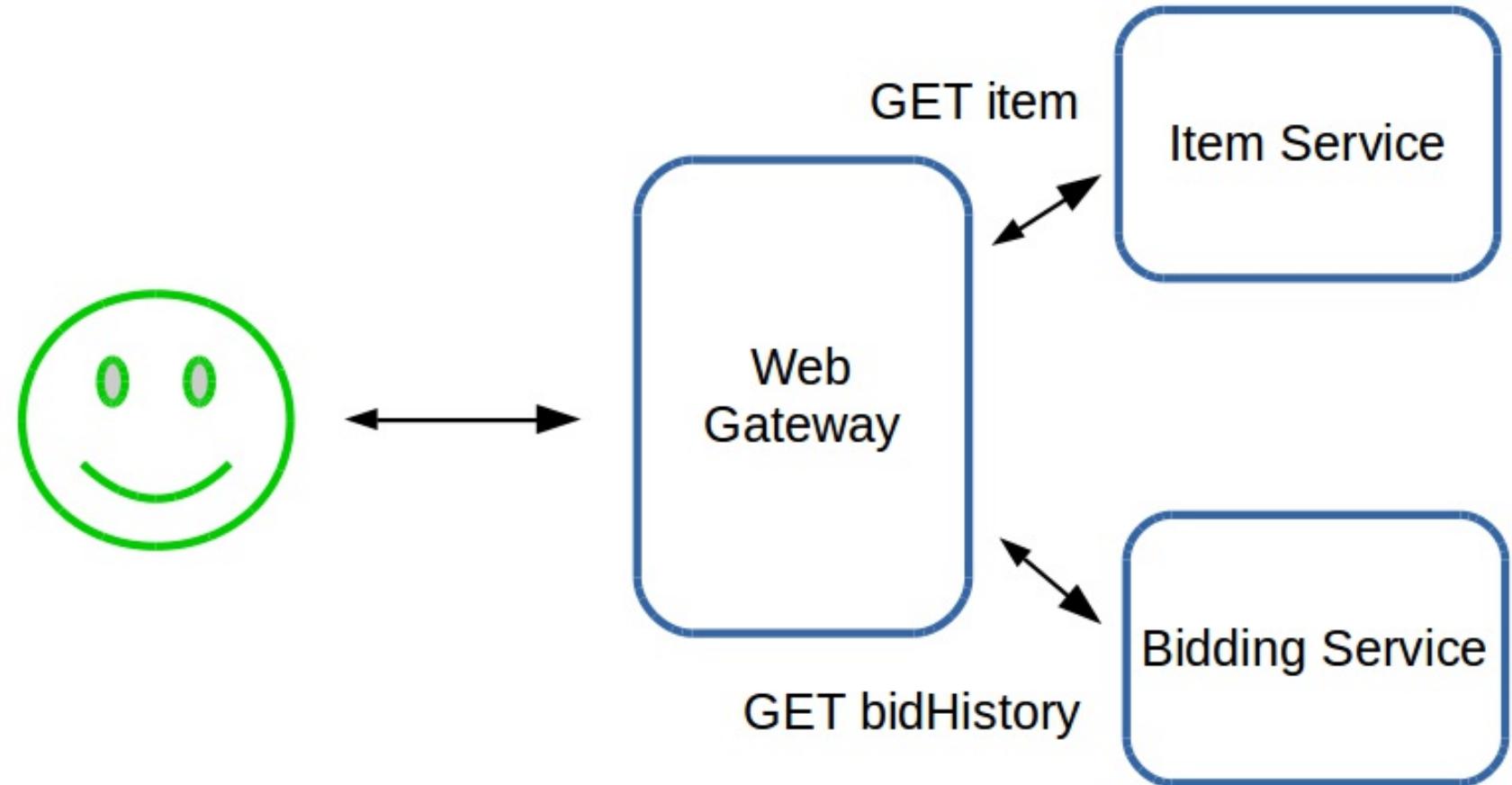
- Typically request/response

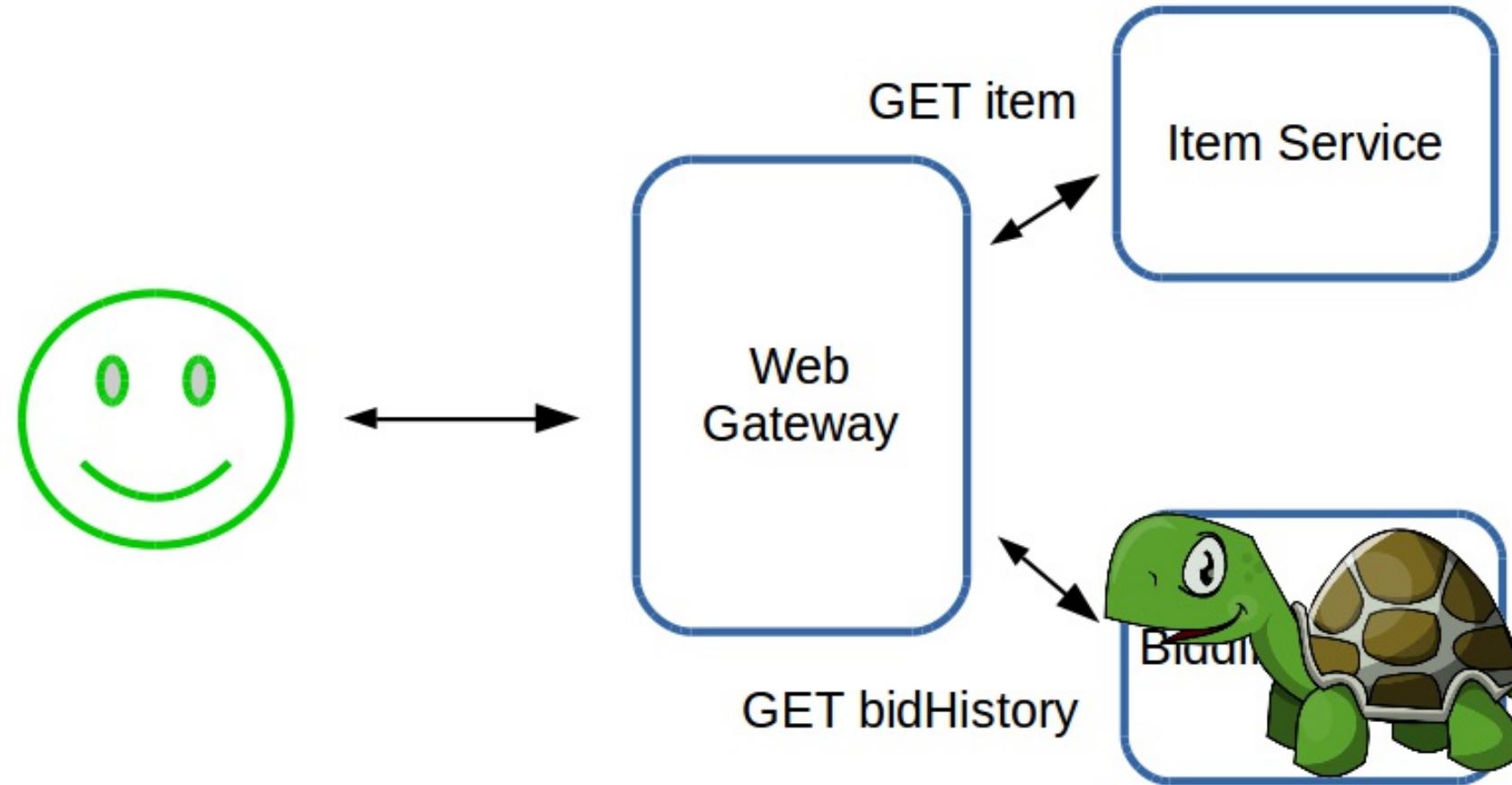
# SYNCHRONOUS COMMUNICATION

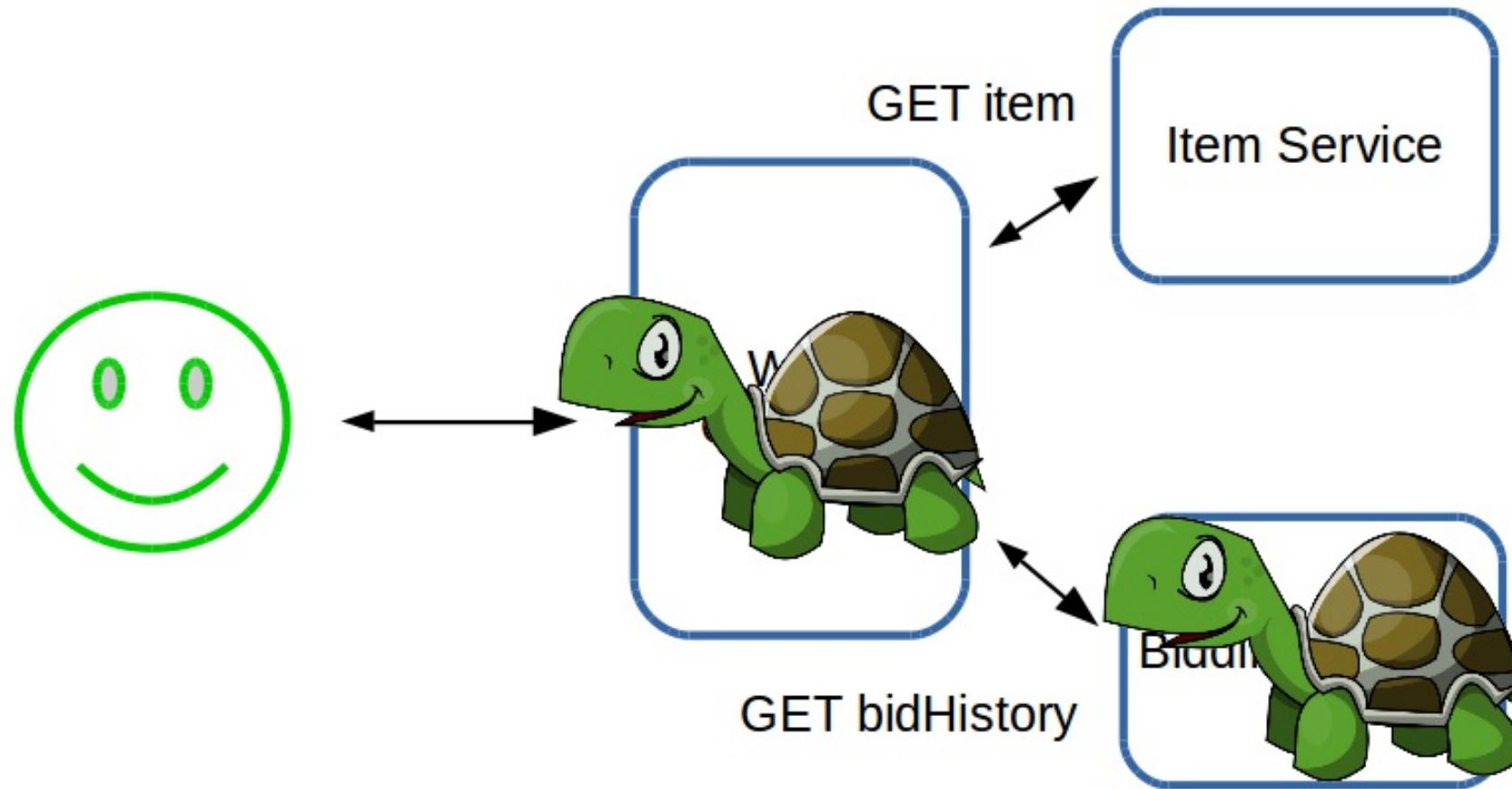
- Typically request/response
  - e.g. REST

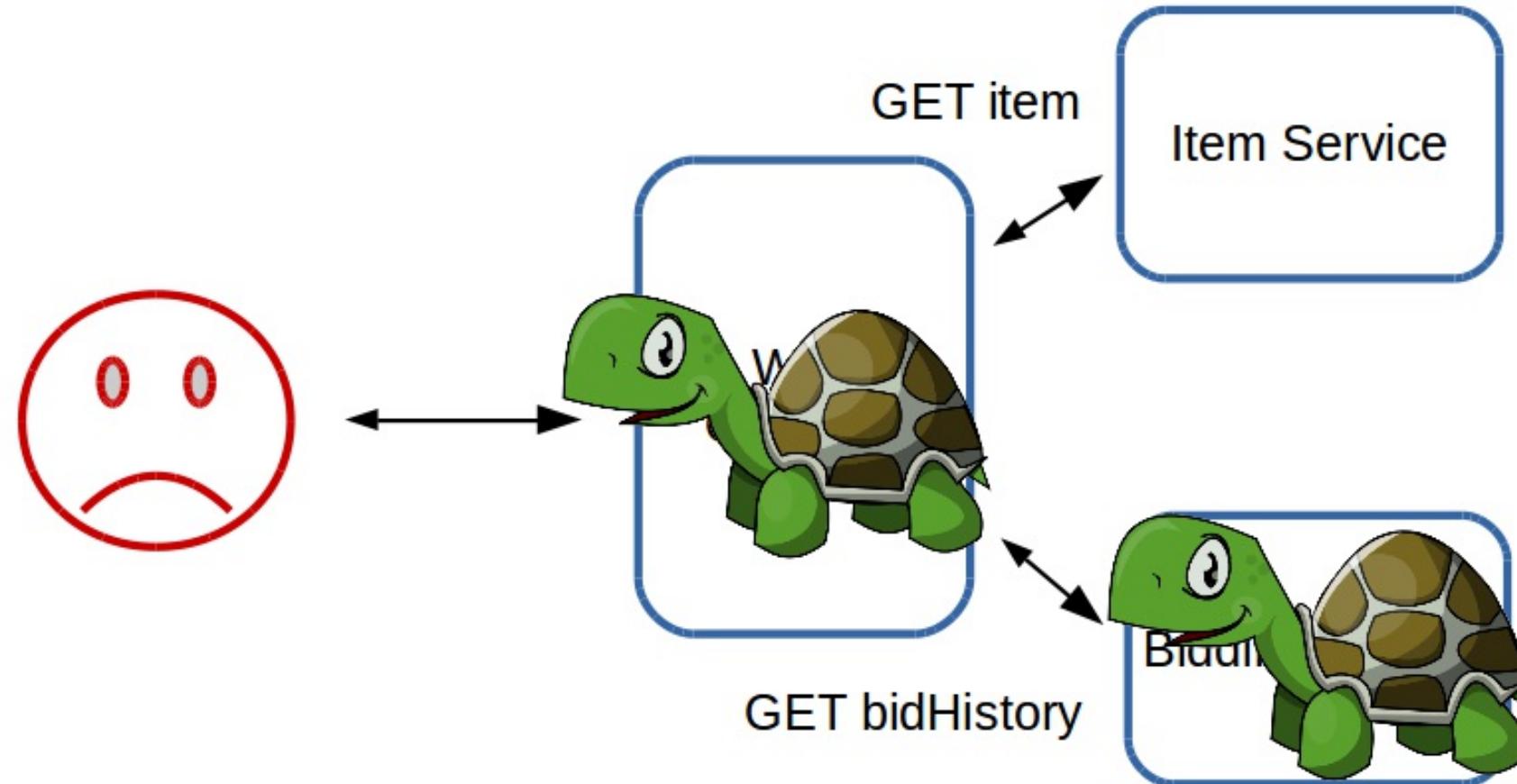
# SYNCHRONOUS COMMUNICATION

- Typically request/response
  - e.g. REST
- Both systems must be responsive at the same time









# PATTERN 1: CIRCUIT BREAKERS

# PATTERN 1: CIRCUIT BREAKERS

- A gate that opens in the event of failure

# PATTERN 1: CIRCUIT BREAKERS

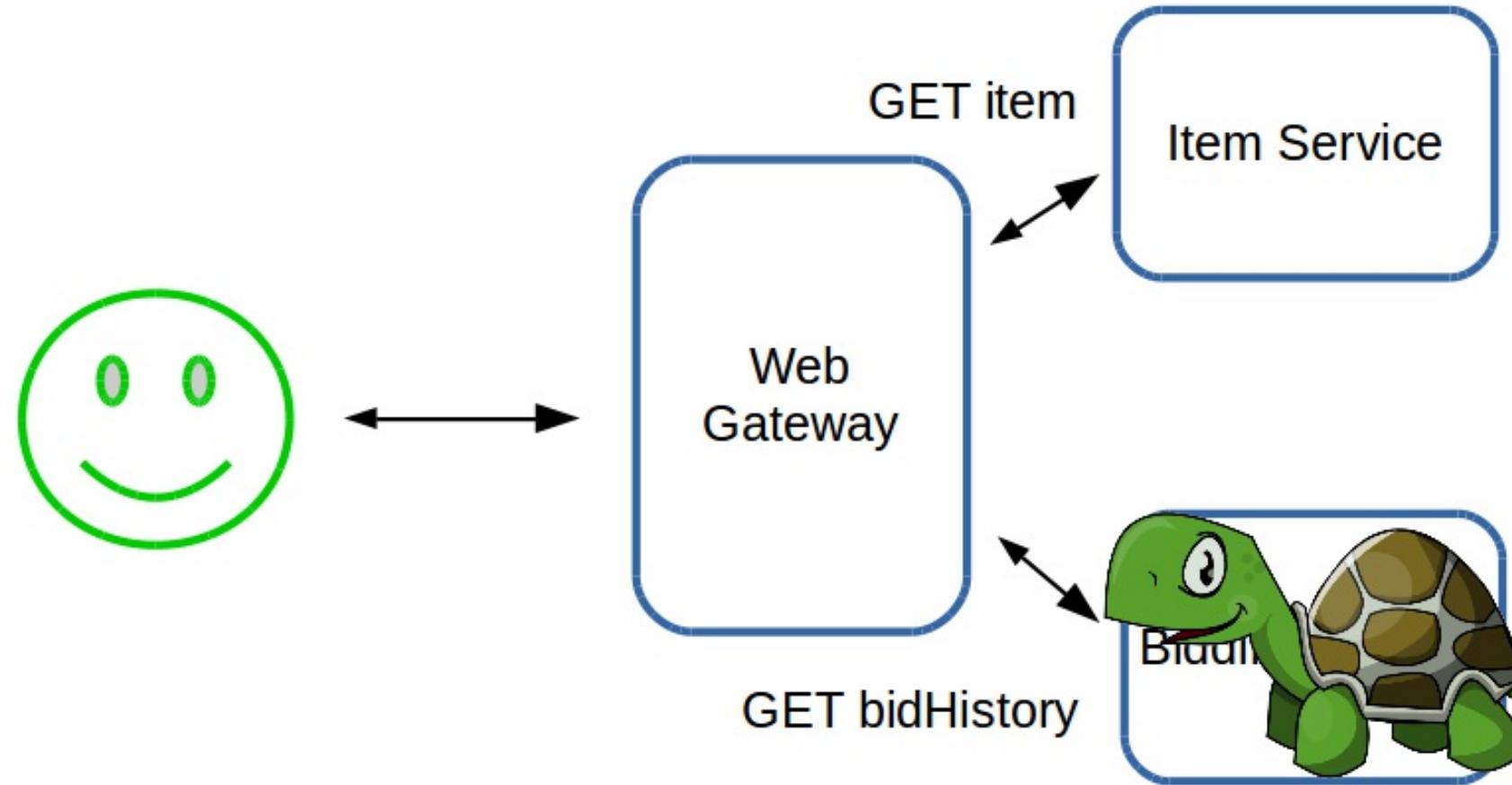
- A gate that opens in the event of failure
  - Including timeouts

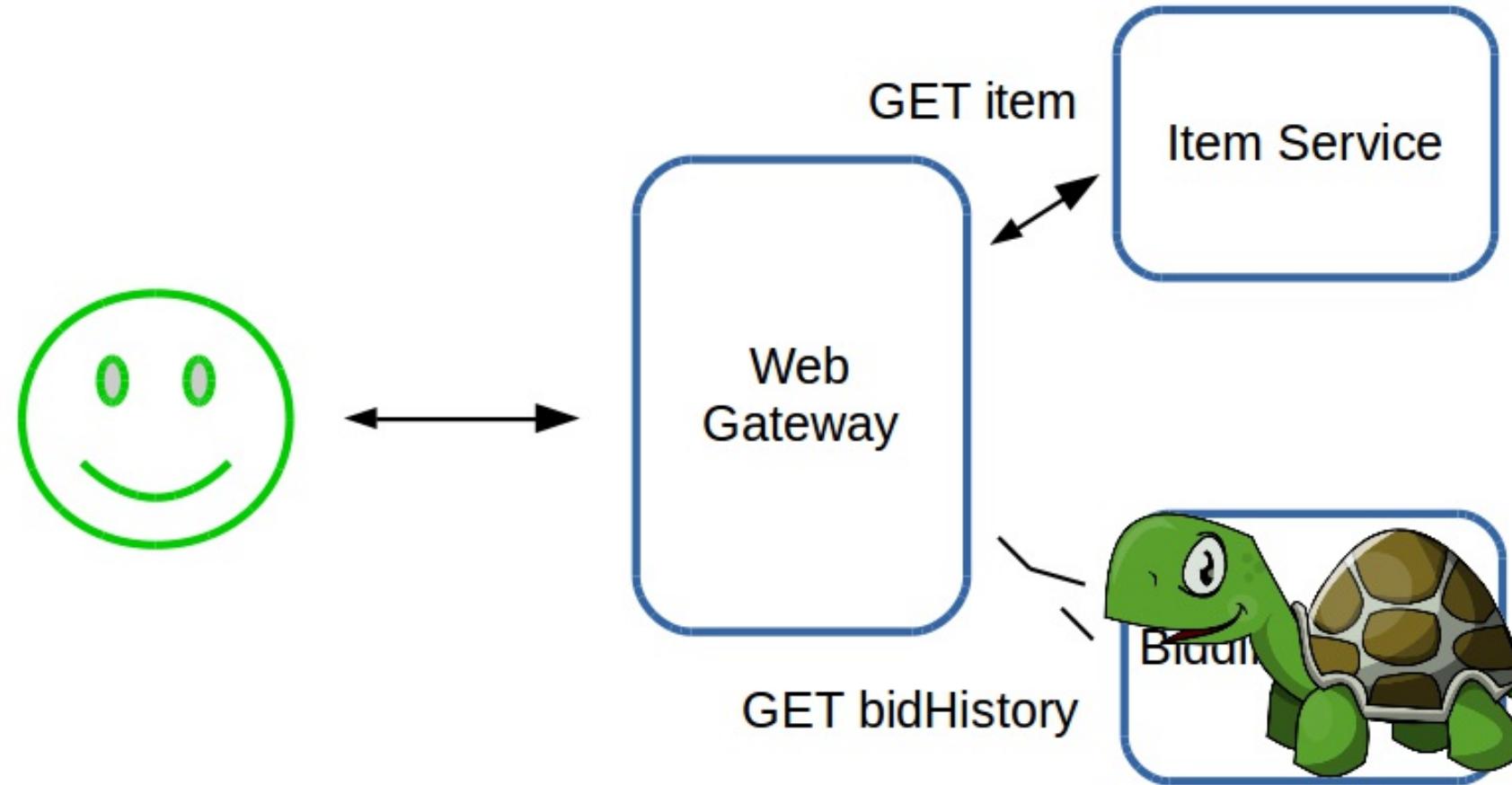
# PATTERN 1: CIRCUIT BREAKERS

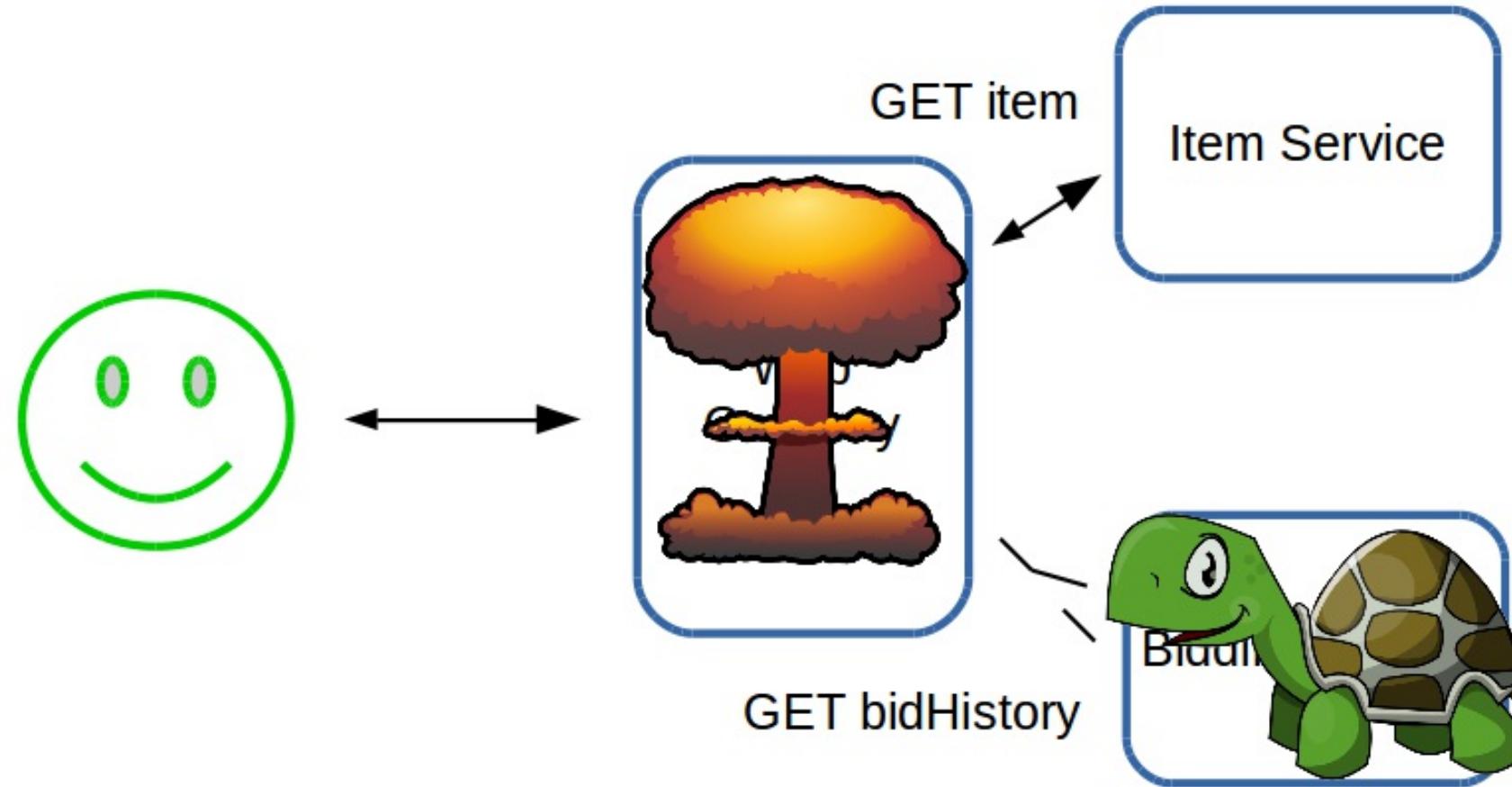
- A gate that opens in the event of failure
  - Including timeouts
- Protects already failing services

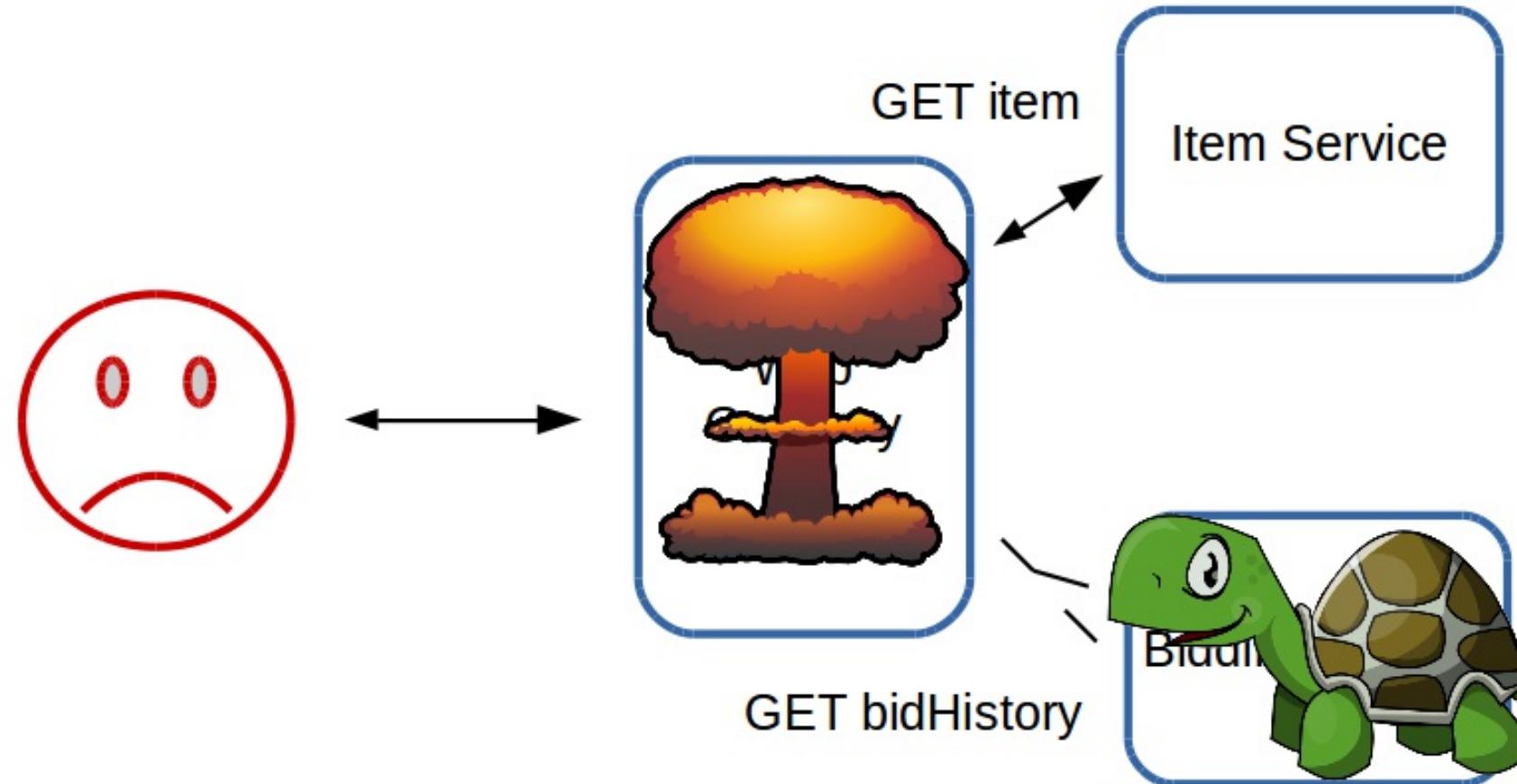
# PATTERN 1: CIRCUIT BREAKERS

- A gate that opens in the event of failure
  - Including timeouts
- Protects already failing services
- Allows fail fast handling









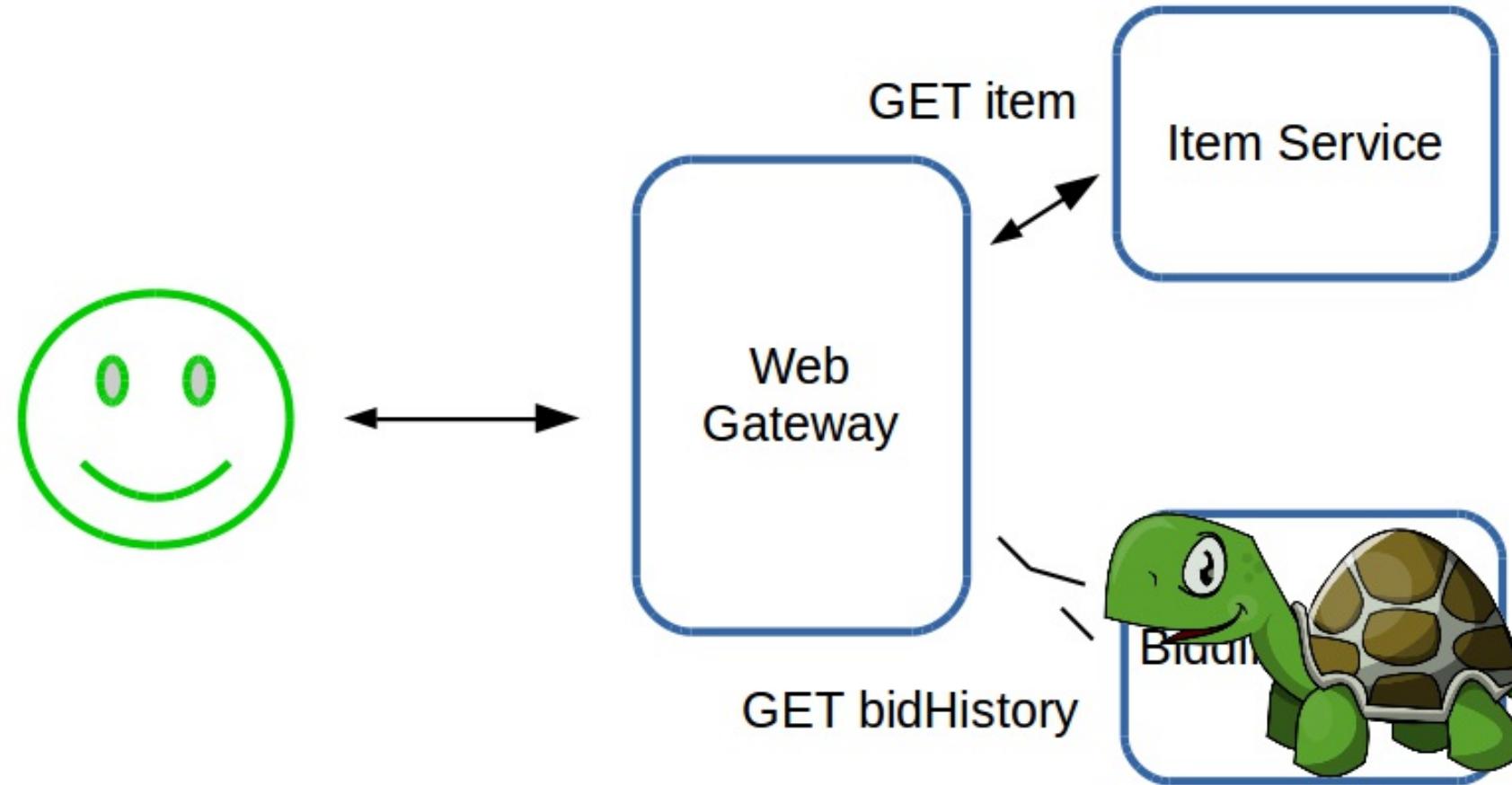
## PATTERN 2: FAILURE RECOVERY

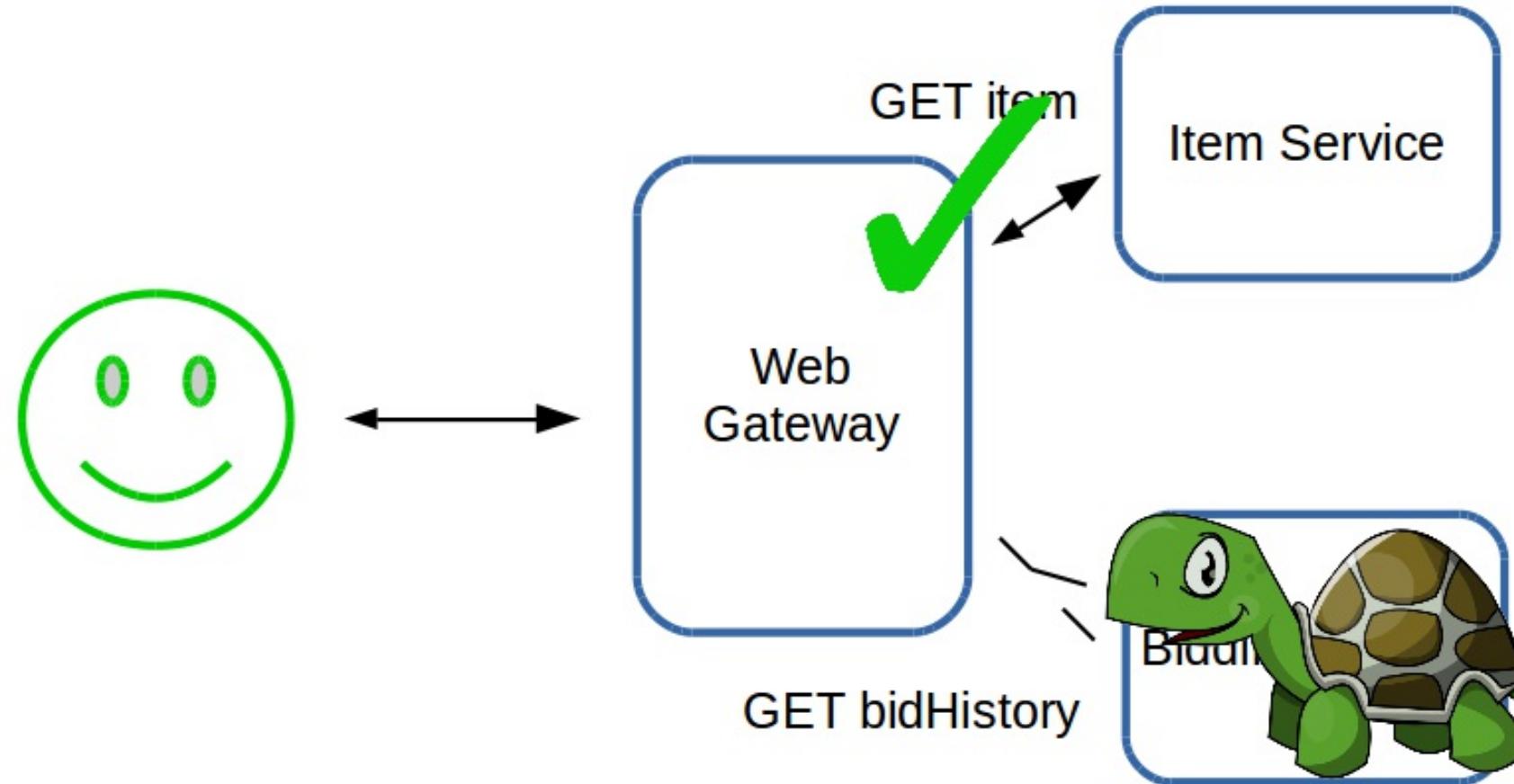
## PATTERN 2: FAILURE RECOVERY

- Work around failure by degrading

## PATTERN 2: FAILURE RECOVERY

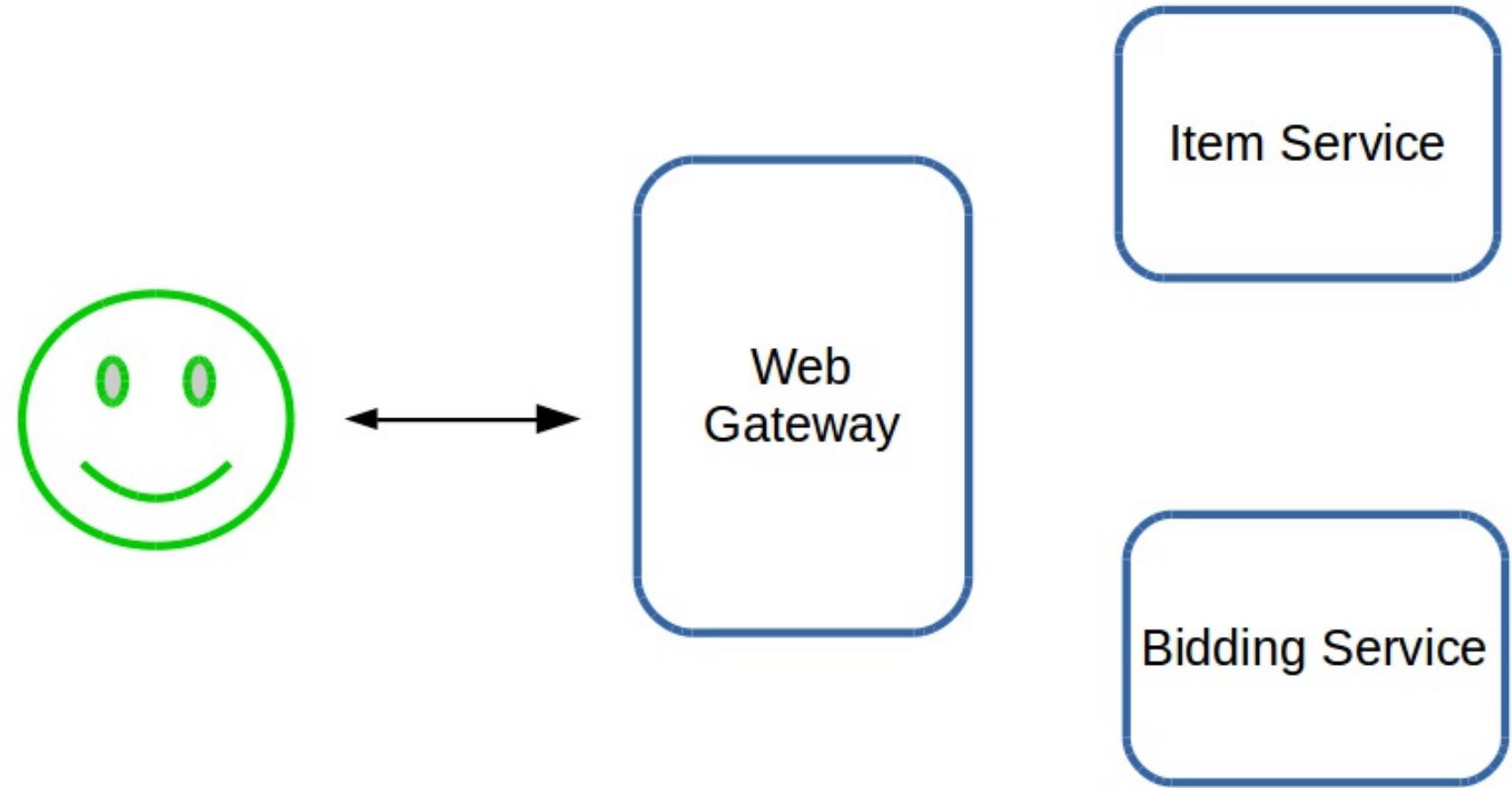
- Work around failure by degrading
- Not every call is necessary to render every page

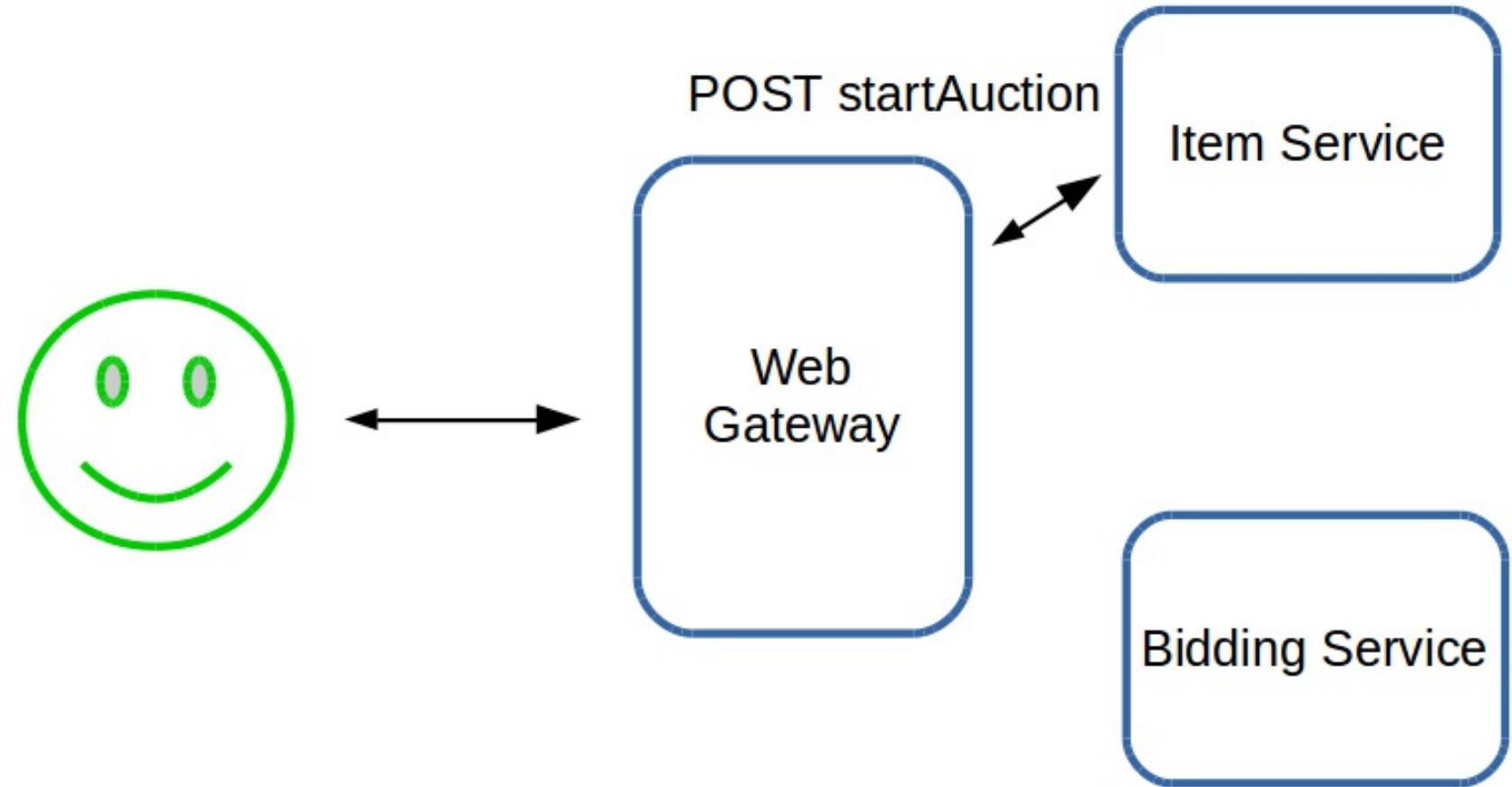


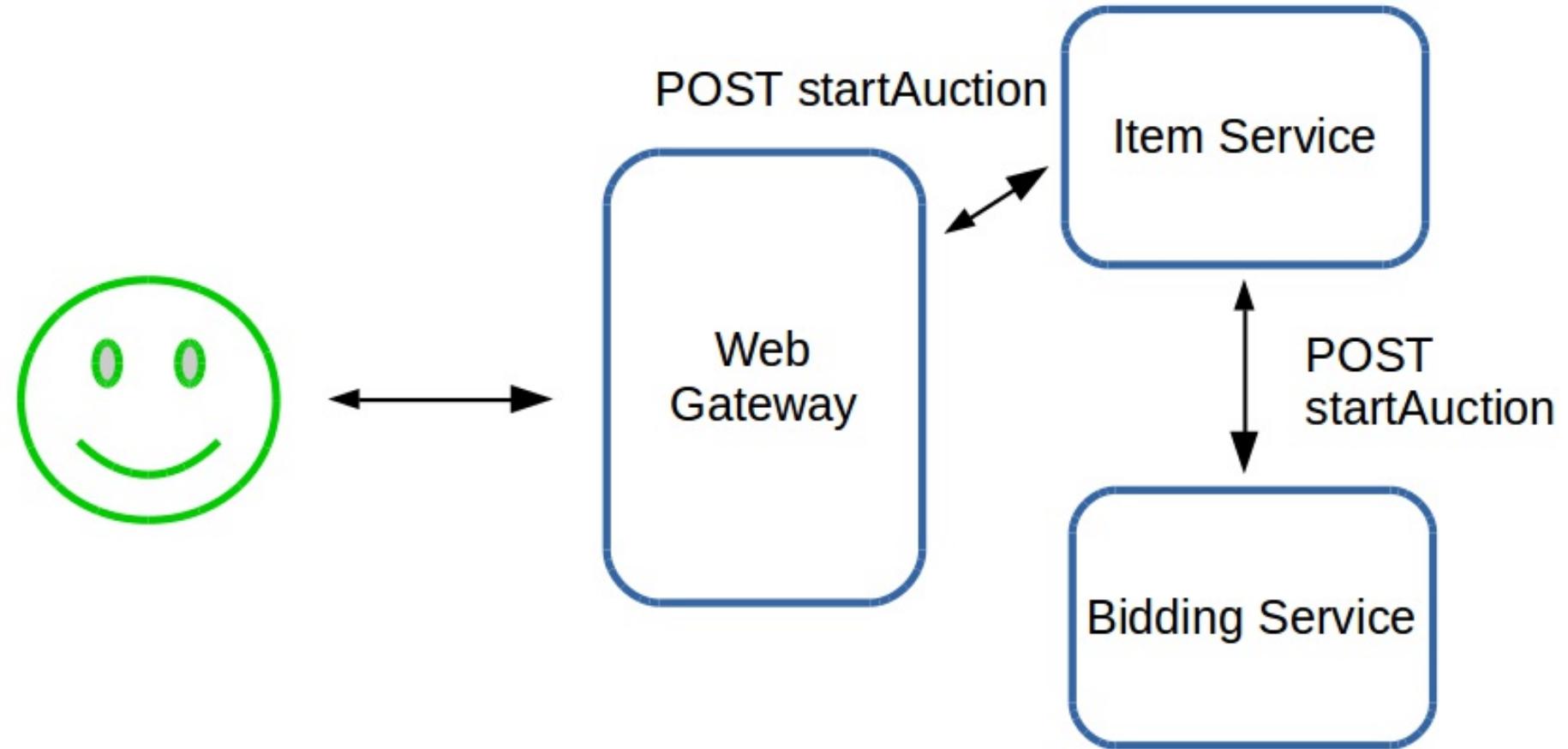


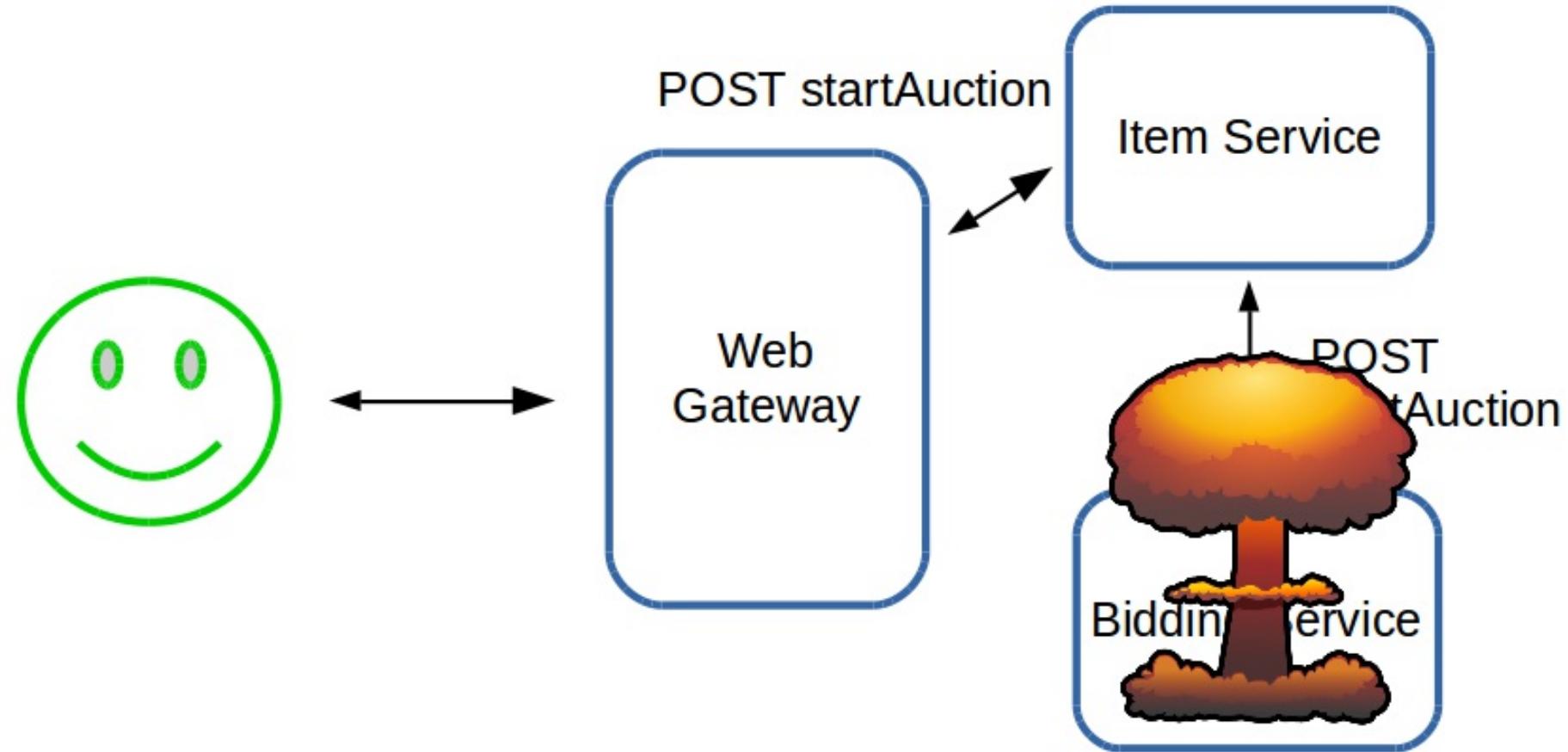


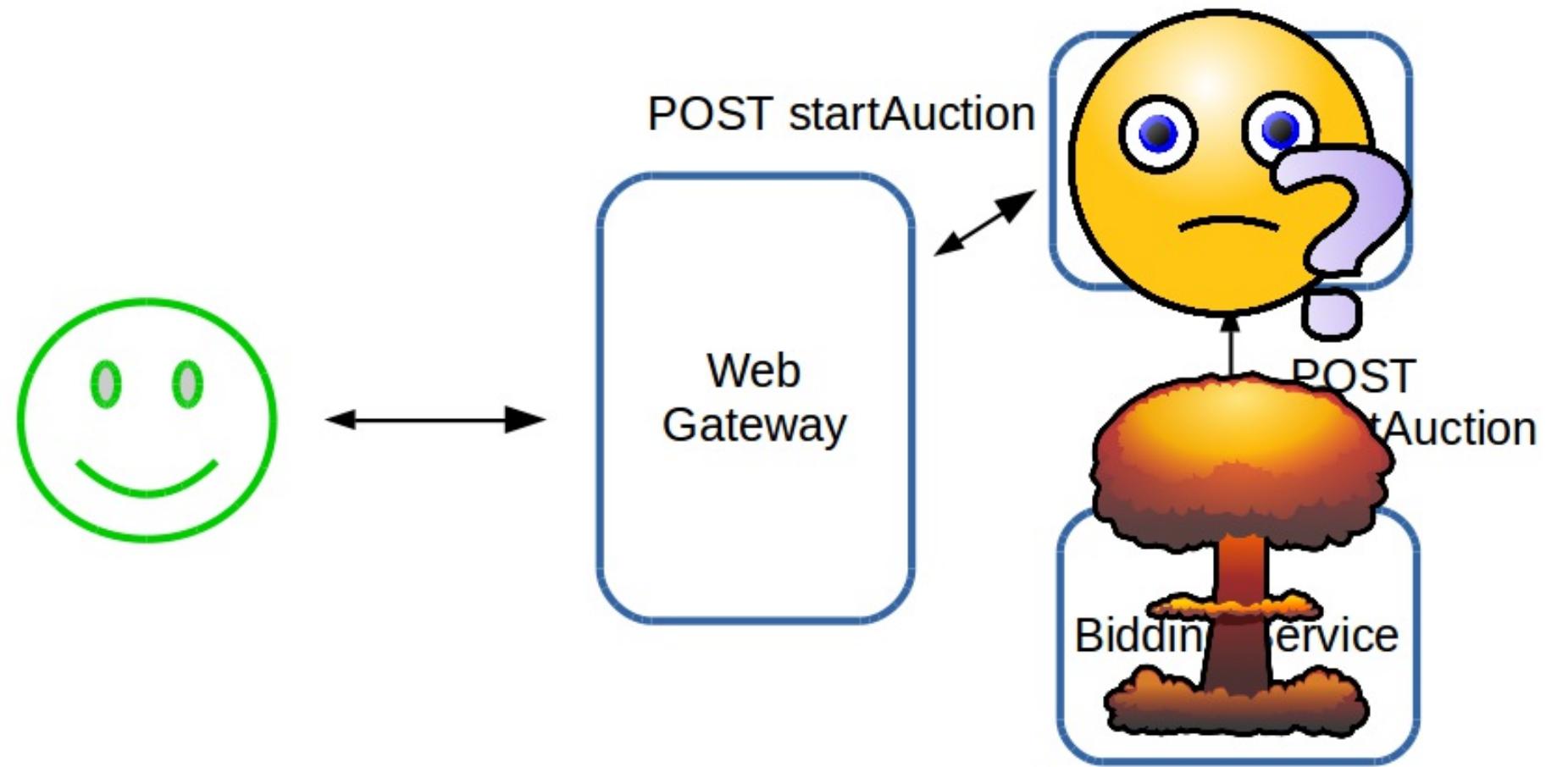
FAILURE CAN LEAD TO  
INCONSISTENCY

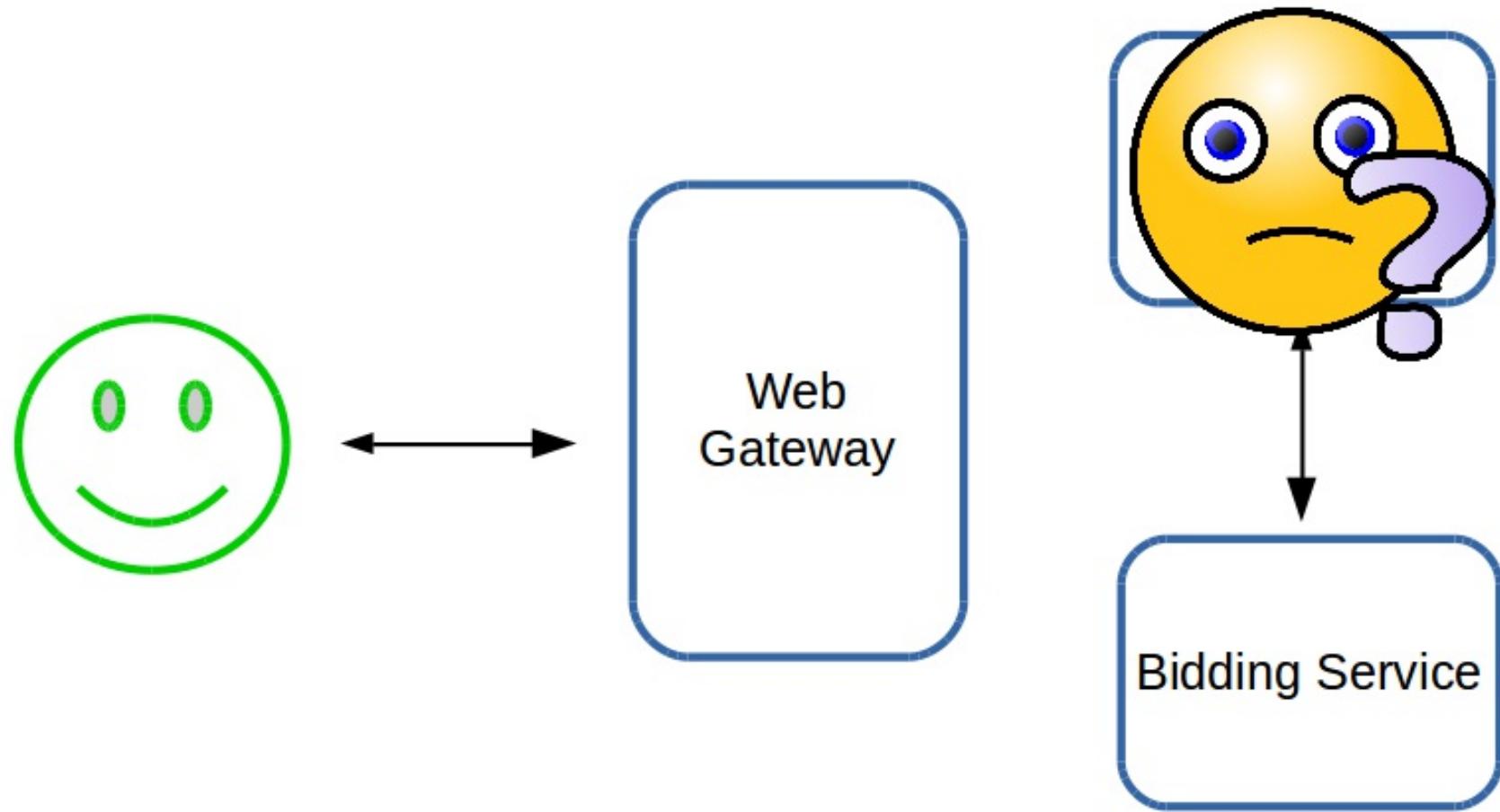


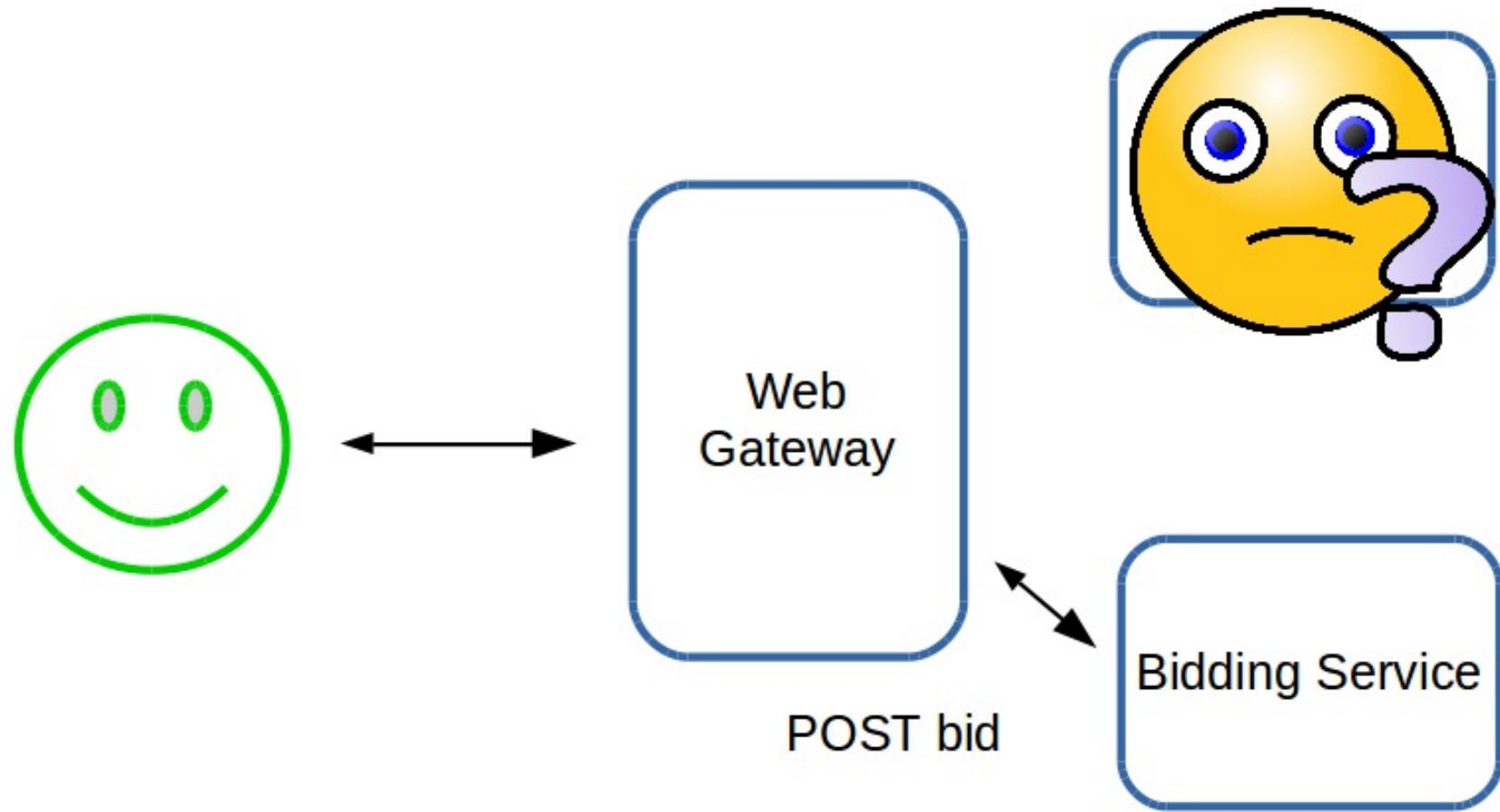


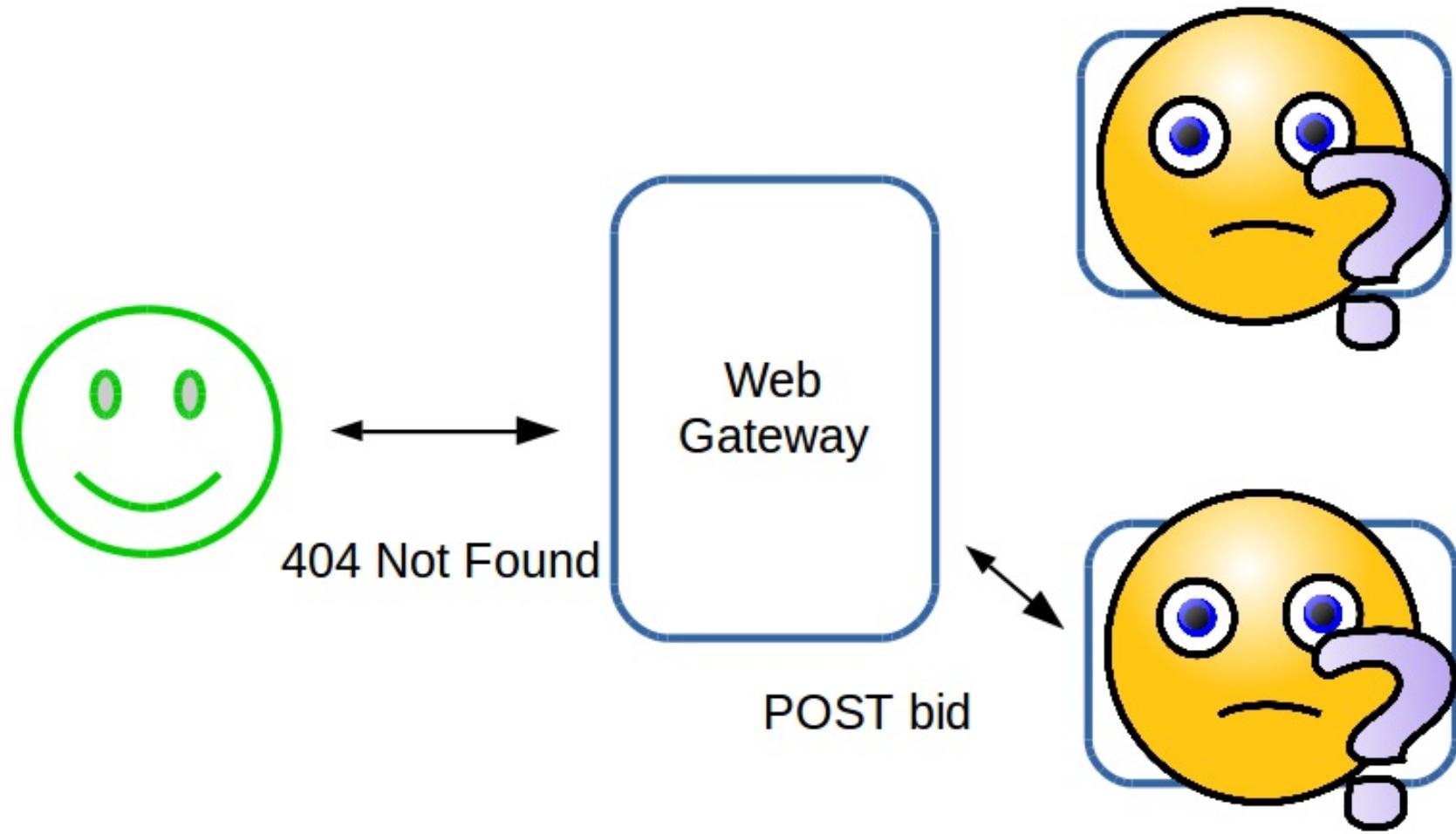


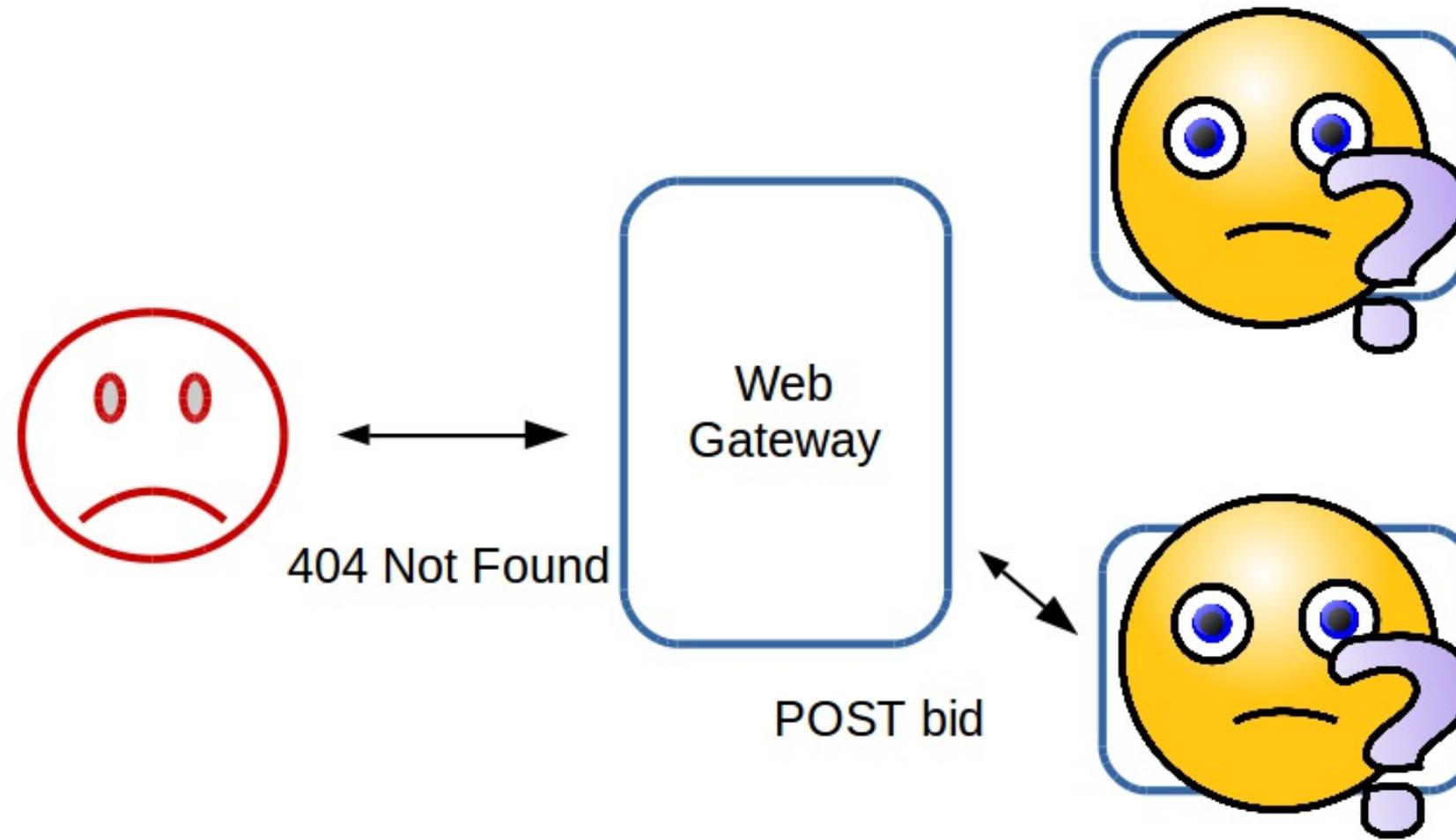












# INCONSISTENCY FROM FAILURE

# INCONSISTENCY FROM FAILURE

- Synchronous "at same time" communication of updates is dangerous

# INCONSISTENCY FROM FAILURE

- Synchronous "at same time" communication of updates is dangerous
  - Transactions can't span service boundaries

# PATTERN 3: ASYNCHRONOUS MESSAGING

## PATTERN 3: ASYNCHRONOUS MESSAGING

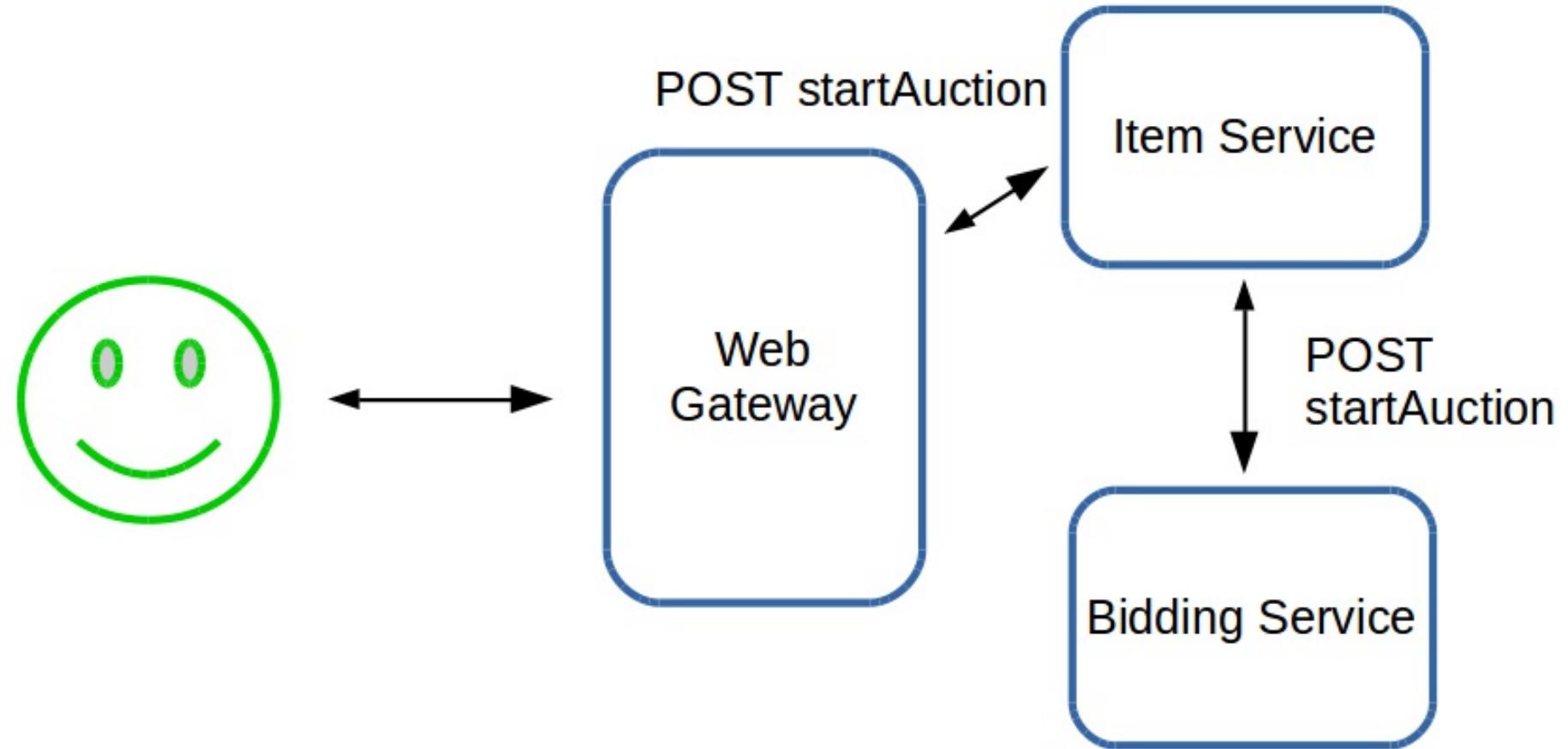
- Does not require both systems to be responsive

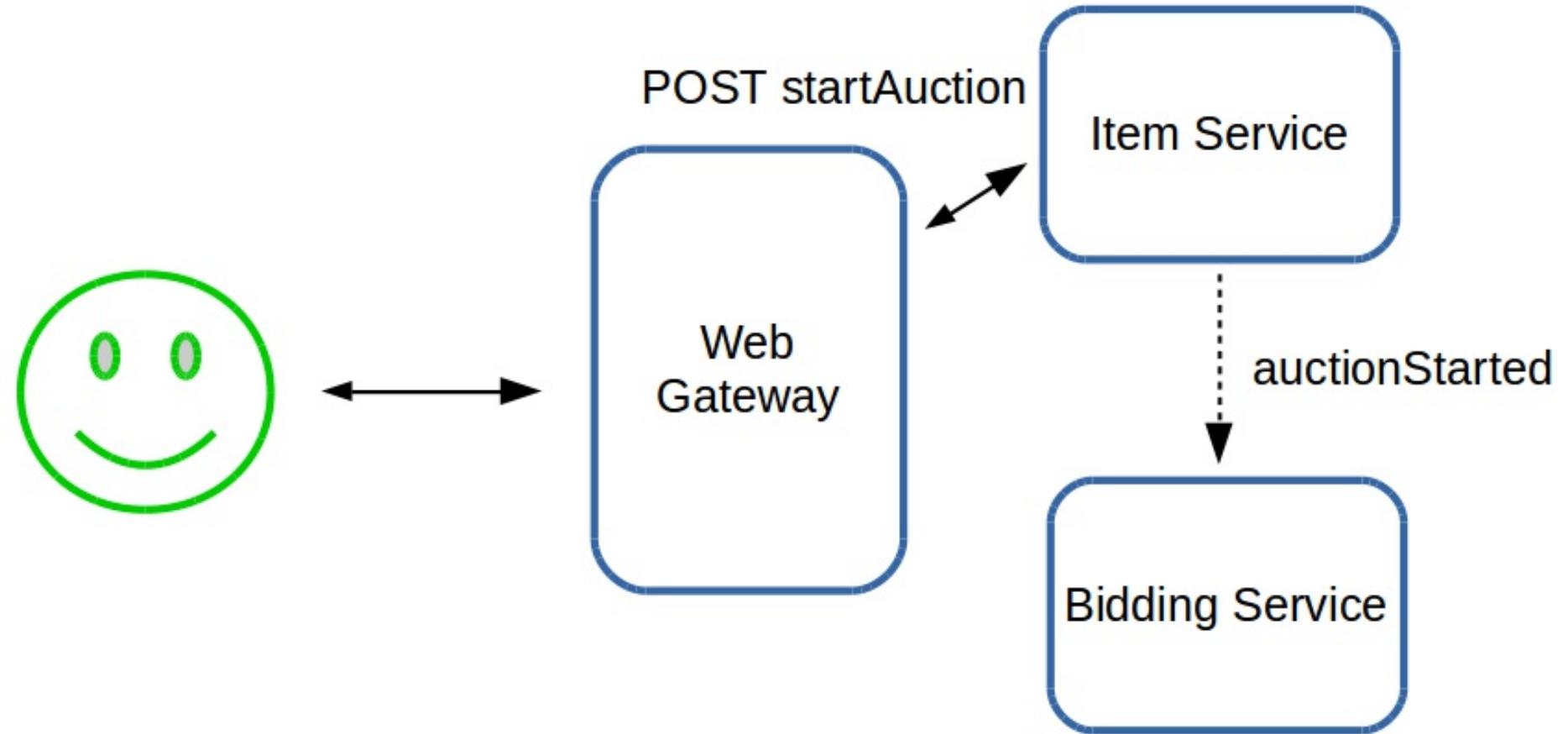
## PATTERN 3: ASYNCHRONOUS MESSAGING

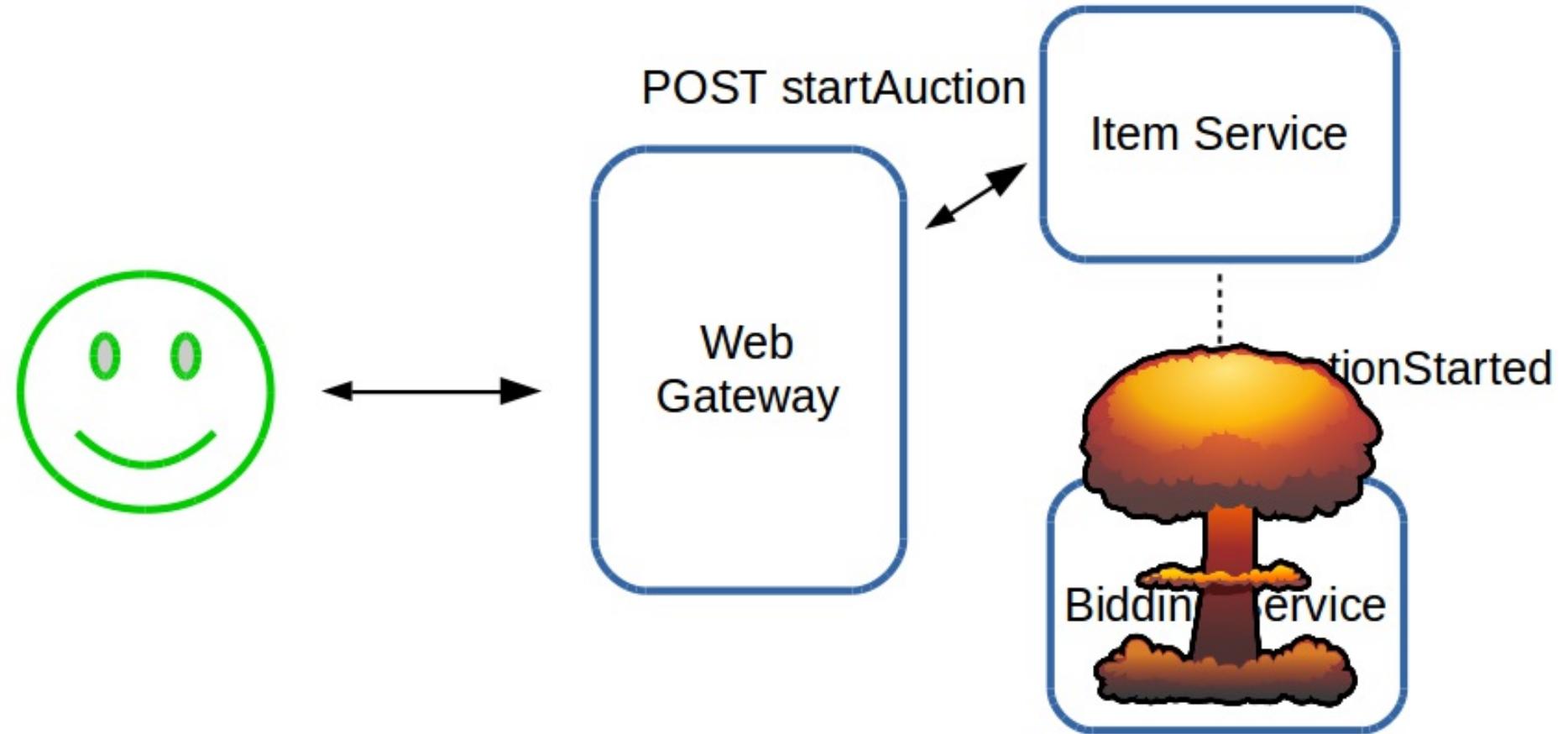
- Does not require both systems to be responsive
- Perfect if you already persist events

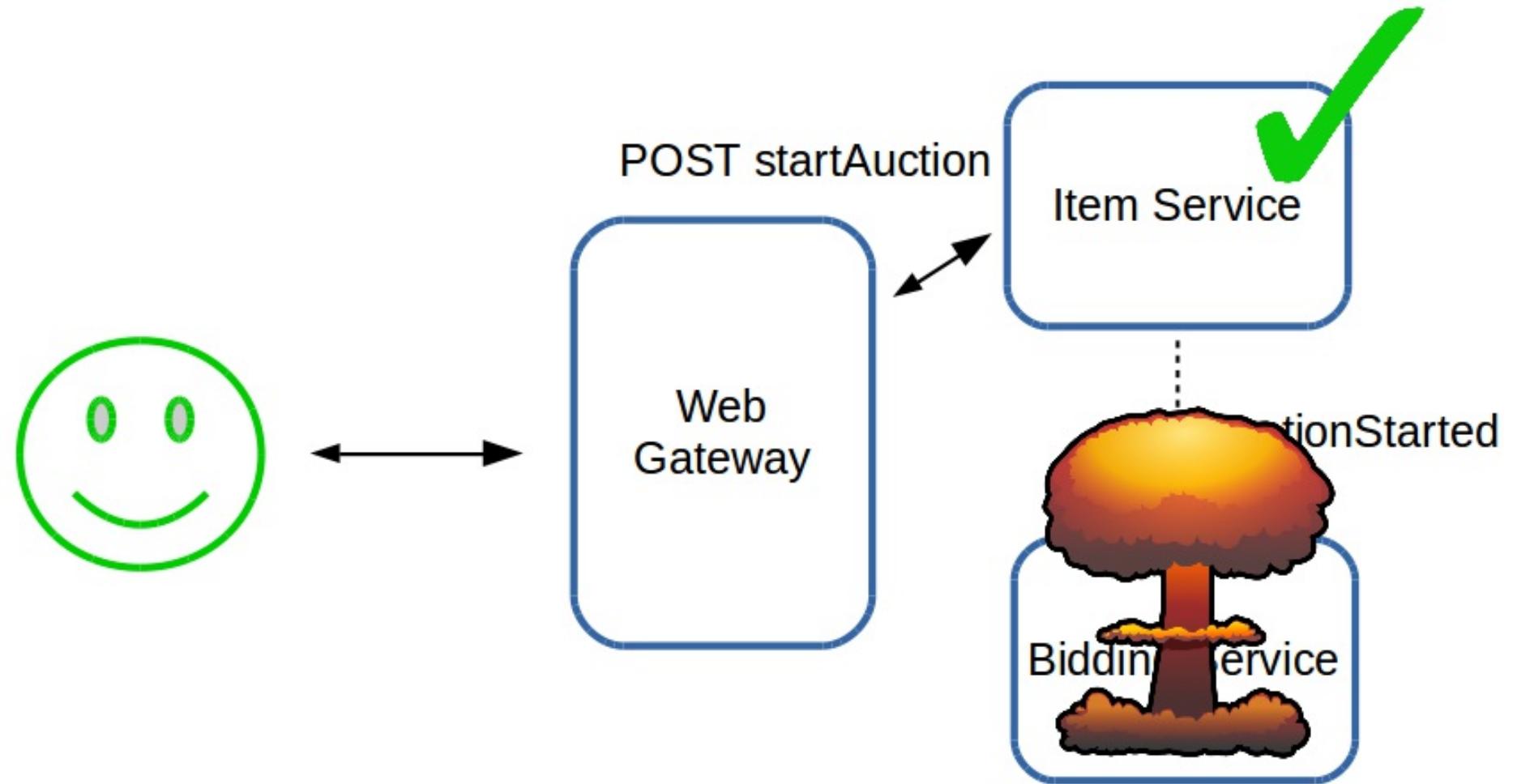
# PATTERN 3: ASYNCHRONOUS MESSAGING

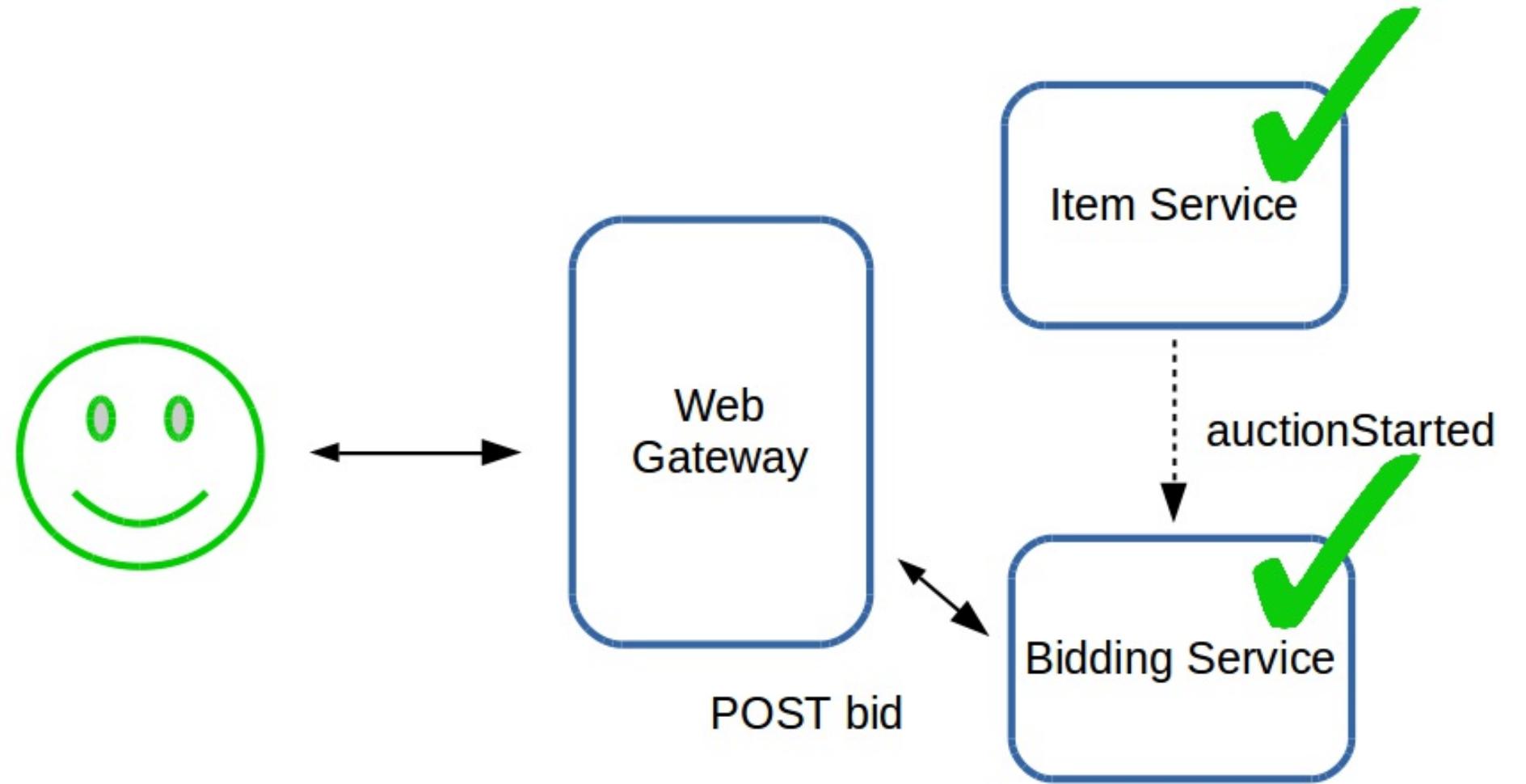
- Does not require both systems to be responsive
- Perfect if you already persist events
- Use persistent events as a source of messages











# UNACCEPTABLE DEGRADATION

# UNACCEPTABLE DEGRADATION

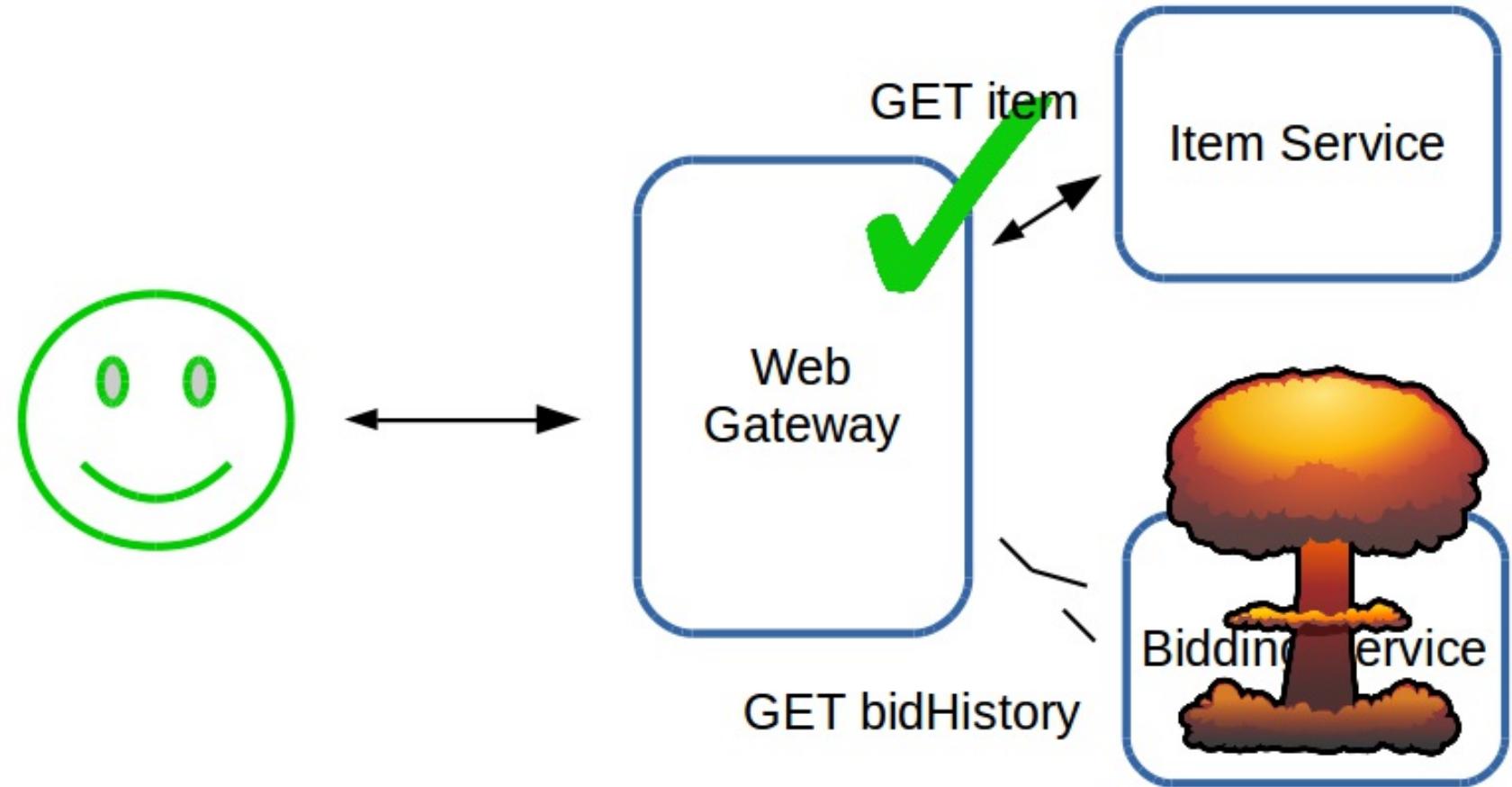
- Earlier we degraded the item page with empty bid history

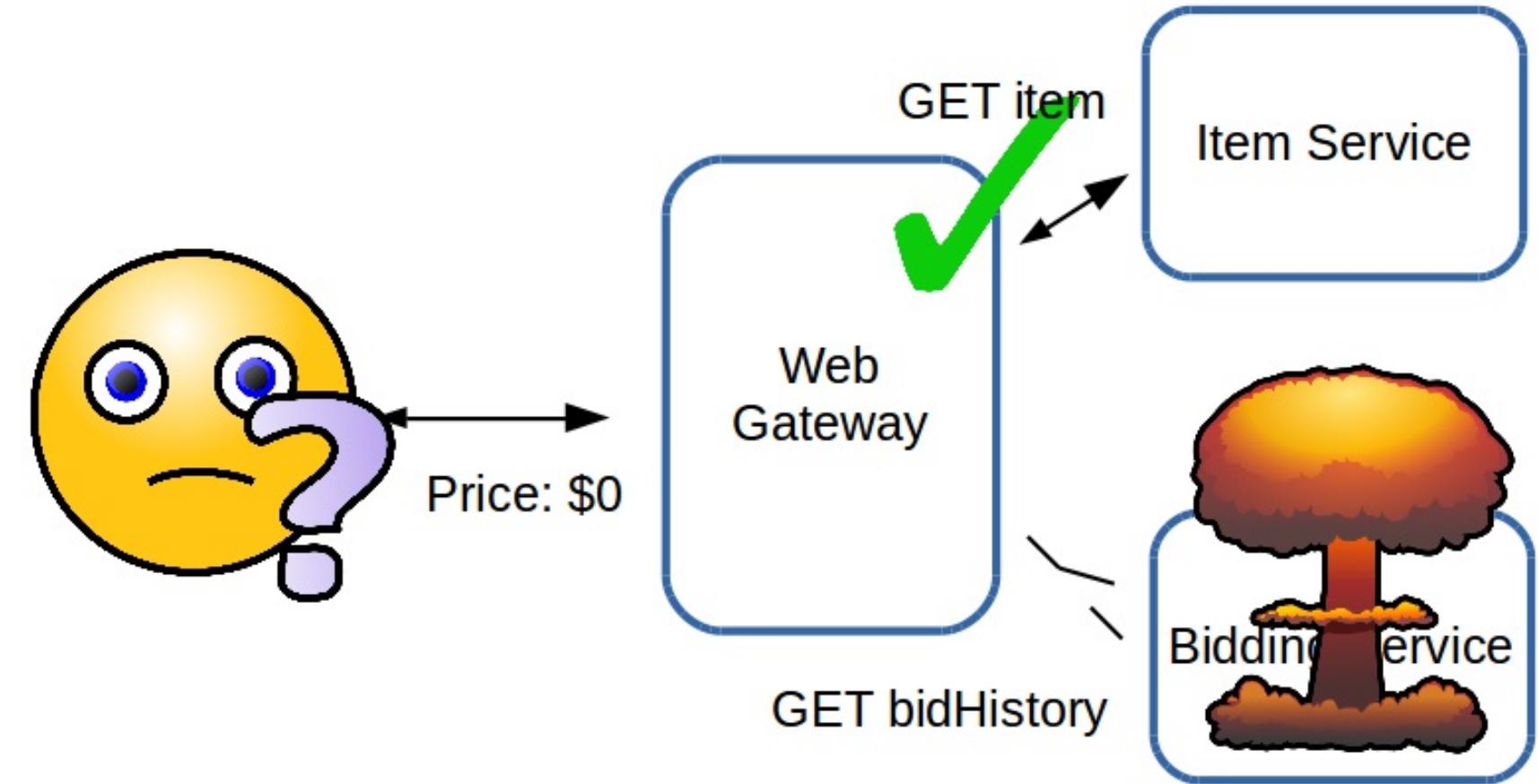
# UNACCEPTABLE DEGRADATION

- Earlier we degraded the item page with empty bid history
- Price was also \$0

# UNACCEPTABLE DEGRADATION

- Earlier we degraded the item page with empty bid history
- Price was also \$0
- Users may tolerate no history, but not wrong price





# PATTERN 4: DENORMALIZE

## PATTERN 4: DENORMALIZE

- Push important information to other services

## PATTERN 4: DENORMALIZE

- Push important information to other services
  - Important for system functions

## PATTERN 4: DENORMALIZE

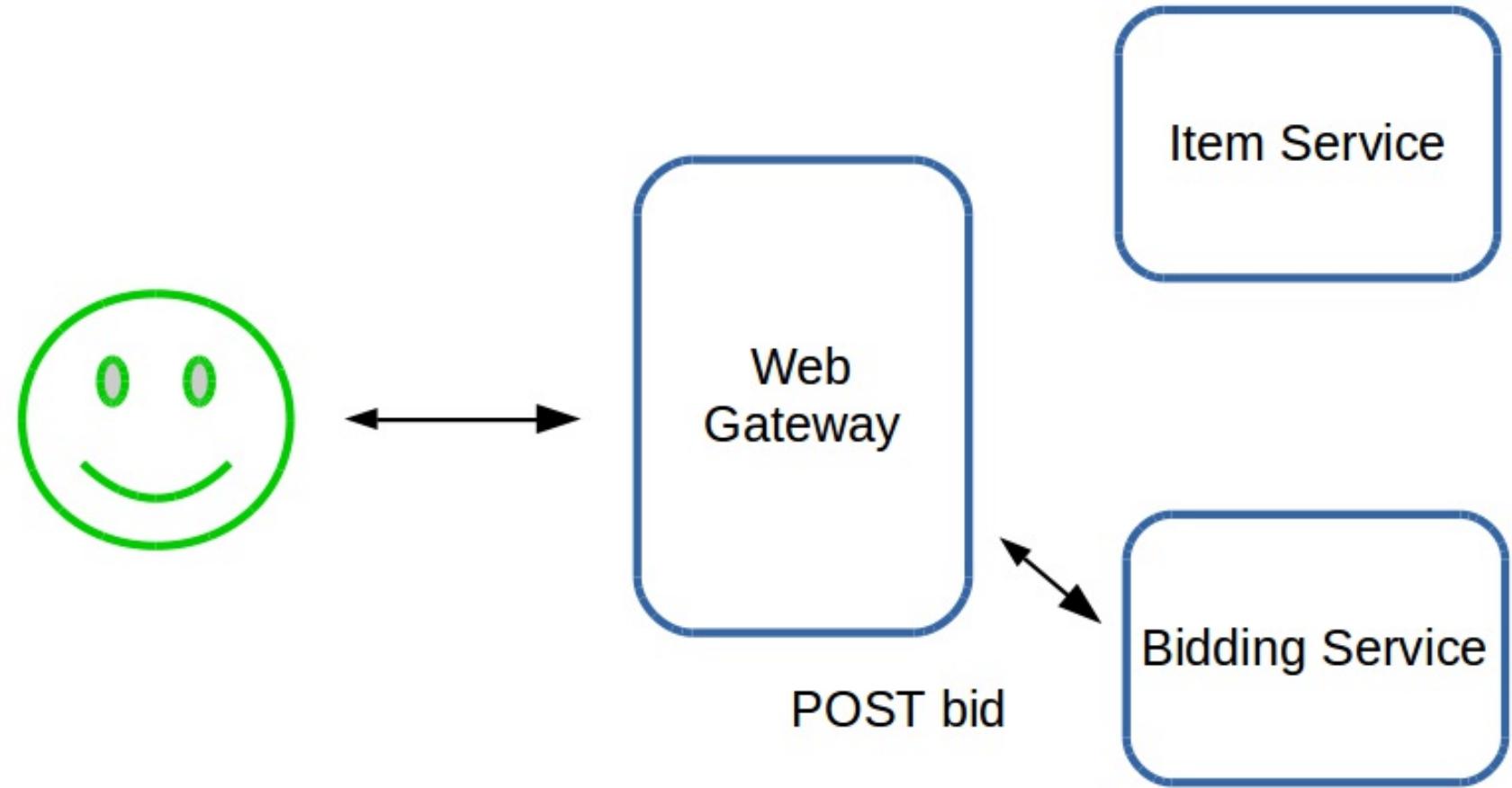
- Push important information to other services
  - Important for system functions
  - Important for business functions

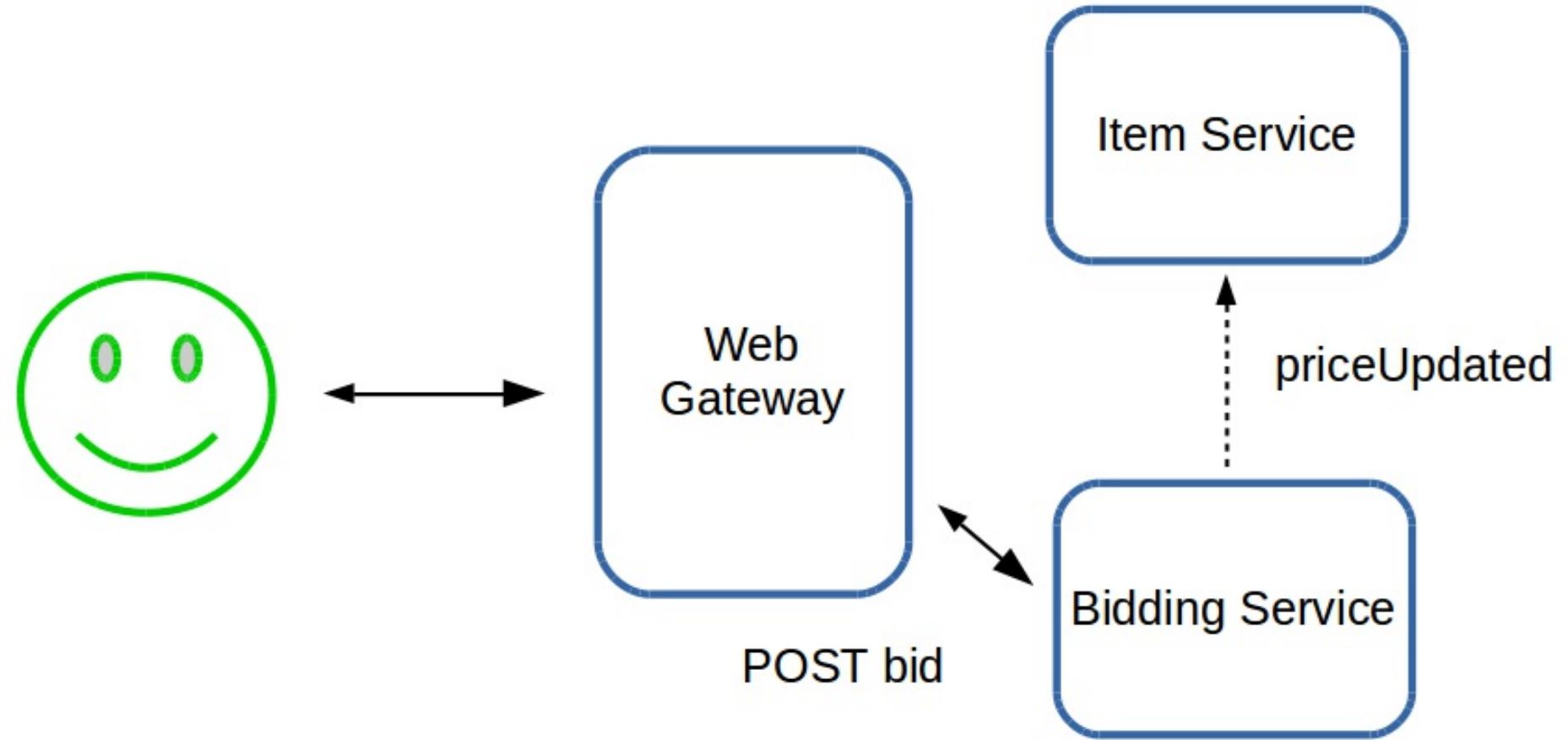
## PATTERN 4: DENORMALIZE

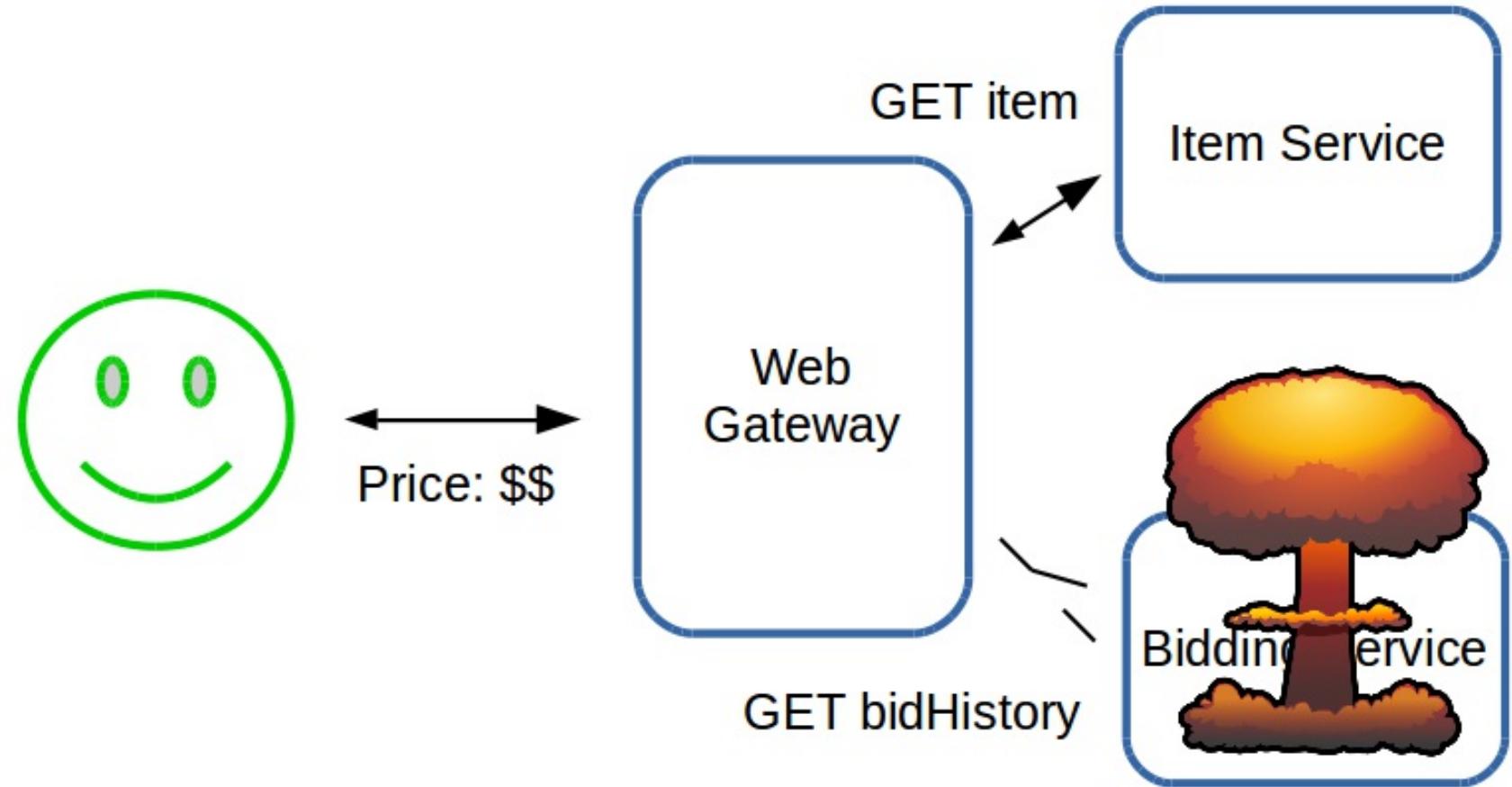
- Push important information to other services
  - Important for system functions
  - Important for business functions
- Store duplicated information in those services

## PATTERN 4: DENORMALIZE

- Push important information to other services
  - Important for system functions
  - Important for business functions
- Store duplicated information in those services
  - AKA denormalization







# SUMMARY



# SUMMARY

- Monolith to microservices requires rearchitecting data flows

# SUMMARY

- Monolith to microservices requires rearchitecting data flows
- Failure and inconsistency must be managed

# SUMMARY

- Monolith to microservices requires rearchitecting data flows
- Failure and inconsistency must be managed
- Use:

# SUMMARY

- Monolith to microservices requires rearchitecting data flows
- Failure and inconsistency must be managed
- Use:
  - Circuit breakers

# SUMMARY

- Monolith to microservices requires rearchitecting data flows
- Failure and inconsistency must be managed
- Use:
  - Circuit breakers
  - Failure degradation

# SUMMARY

- Monolith to microservices requires rearchitecting data flows
- Failure and inconsistency must be managed
- Use:
  - Circuit breakers
  - Failure degradation
  - Asynchronous messaging

# SUMMARY

- Monolith to microservices requires rearchitecting data flows
- Failure and inconsistency must be managed
- Use:
  - Circuit breakers
  - Failure degradation
  - Asynchronous messaging
  - Denormalization

## NEXT STEPS

<http://lagomframework.com>

<https://github.com/jroper/mircoservices-architecture>